

1. Introduction

1.1 What is Machine Learning?

1.1.1 The Basic Concept

Artificial Intelligence (AI)

בינה מלאכותית הינו תחום בו תוכנות מחשב או מנגנון טכנולוגי אחר מחקה מנגנון חשיבה אנושי. בתחום רחב זה יש רמות שונות של בינה מלאכותית – יש מערכות שמסוגלות ללמוד דפוסי התנהגות ולהתאים את עצמן לשינויים, ואילו יש מערכות שאמנם מחקות מנגנון חשיבה אנושי אך הן לא מתוככמות מעבר למה שתכנתו אותן בהתחלה. שואב רובוטי היודע לחשב את גודל החדר ואת מסלול הניקוי האופטימלי פועל לפי פרוצדורה ידועה מראש, ואין בו תחכום מעבר לתכנות הראשוני שלו. לעומת זאת תוכנה היודעת לסנן רעשים באופן מסתגל, או להמליץ על שירים בנגן מוזיקה בהתאם לסגנון של המשתמש, משתמשות בבינה מלאכותית ברמה גבוהה יותר, כיוון שהן לומדות עם הזמן דברים חדשים.

המונח בינה מלאכותית מתייחס בדרך כלל למערכת שמחקה התנהגות אנושית, אך היא שגרתית, לא לומדת משהו חדש, ועושה את אותו הדבר כל הזמן. מערכת זו יכולה להיות משוכללת ולחשב דברים מסובכים ואף להסיק מסקנות על דוגמאות חדשות שהיא מעולם לא ראתה, אך תמיד עבור אותו הקלט (Input), יהיה אותו הפלט (Output).

ניקח לדוגמה מערכת סטרימינג של סרטים, למשל Netflix. כחלק משיפור המערכת והגדלת זמני הצפייה, ניתן לבנות מנגנון המלצות הבנוי על היסטוריית השימוש של הלקוחות שלי במערכת – איזה סרטים הם רואים, איזה ז'אנרים ומתי. כשיש מעט צופים ומעט סרטים, ניתן לעשות זאת באופן ידני – למלא טבלאות של הנתונים, לנתח אותם ידנית ולבנות מערכת חוקים שמהווה מנוע המלצות מבוסס AI. ניקח לדוגמה אדם שצופה ב"פארק היורה" וב"אינדיאנה ג'ונס" – סביר שהמערכת תמליץ לו לצפות גם ב-"פולטרגייסט". אדם שצופה לעומת זאת ב-"אהבה בין הכרמים" ו"הבית על האגם", ככל הנראה כדאי להמליץ לו על "הגשרים של מחוז מדיון".

מערכת זו יכולה לעבוד טוב, אך בנקודה מסוימת כבר לא ניתן לנסח אותה כפרוצדורה מסודרת וכאוסף של חוקים ידוע מראש. מאגר הסרטים גדל, נוספים סוגים נוספים של סרטים (כמו למשל סדרות, תוכניות ריאליטי ועוד) ובנוסף רוצים להתייחס לפרמטרים נוספים – האם הצופה ראה את כל הסרט או הפסיק באמצע, מה גיל הצופה ועוד. מערכת הבנויה באופן קלאסי אינה מסוגלת להתמודד עם כמויות המידע הקיימות, וכמות הכללים שנדרש לחשוב עליהם מראש היא עצומה ומורכבת לחישוב.

נתבונן על דוגמה נוספת – מערכת לניווט רכב. ניתן להגדיר כלל פשוט בו אם משתמש יוצא מתל אביב ורוצה להגיע לפתח תקווה, אז האפליקציה תיקח אותו דרך מסלול ספציפי שנבחר מראש. מסלול זה לא מתחשב בפרמטרים קריטיים כמו מה השעה, האם יש פקקים או חסימות ועוד. כמות הפרמטרים שיש להתייחס אליהם איננה ניתנת לטיפול על ידי מערכת כללים ידועה מראש, וגם הפונקציונאליות המתאפשרת היא מוגבלת מאוד – למשל לא ניתן לחזות מה תהיה שעת ההגעה וכדומה.

Machine Learning (ML)

למידת מכונה הוא תת תחום של בינה מלאכותית, הבא להתמודד עם שני האתגרים שתוארו קודם – היכולת לתכנת מערכת על בסיס מסות של נתונים ופרמטרים, וחיזוי דברים חדשים כתלות בפרמטרים רבים שיכולים להשתנות עם הזמן. מנגנוני ML מנתחים כמויות אדירות של דאטה ומנסות להגיע לאיזו תוצאה. אם מדובר באפליקציית ניווט, המערכת תנתח את כל אותם הפקטורים ותנסה לחשב את משך הנסיעה המשוער. נניח והיא חזתה 20 דקות נסיעה. אם בסופו של דבר הנסיעה ארכה 30 דקות, האלגוריתם ינסה להבין איזה פקטור השתנה במהלך הדרך ומדוע הוא נכשל בחיזוי (למשל: הכביש בן ארבעה נתיבים, אבל במקטע מסוים הוא מצטמצם לאחד וזה מייצר עיכוב, וזה עיכוב קבוע ברוב שעות היממה ולא פקק אקראי). בהינתן מספיק מקרים כאלה, האלגוריתם "מבין" שהוא טועה, והוא פשוט יתקן את עצמו ויכניס למערך החישובים גם פקטור של מספר נתיבים ויוריד אולי את המשקל של הטמפרטורה בחוץ. וככה באופן חזרתי האלגוריתם שוב ושוב מקבל קלט, מוציא פלט ובודק את התוצאה הסופית. לאחר מכן הוא בודק היכן הוא טעה, משנה את עצמו, מתקן את המשקל שהוא נותן לפקטורים שונים ומשתכלל מנסיעה לנסיעה.

במערכות אלה הקלט נשאר לכאורה קבוע, אבל הפלט משתנה – עבור זמני יציאה שונים, האלגוריתם יעריך זמני נסיעה שונים, כתלות במגוון הפרמטרים הרלוונטיים.

מערכות ML משמשות את כל רשתות הפרסום הגדולות. כל אחת מנסה בדרכה שלה לחזות למשל, איזה משתמש שהקליק על המודעה צפוי שיבצע רכישה. הפלטפורמות השונות מנסות לזהות כוונה (Intent) על ידי למידה

מניסיון. בהתחלה הן פשוט ניחשו על פי כמה פקטורים שהוזנו להם על ידי בני אדם. נניח, גוגל החליטה שמי שצופה בסרטוני יוטיוב של Unboxing הוא ב-Intent גבוה של רכישה. בהמשך הדרך, בהנחה והמשתמש מבצע רכישה כלשהי, האלגוריתם מקבל "נקודה טובה". אם הוא לא קנה, האלגוריתם מקבל "נקודה רעה". ככל שהוא מקבל יותר נקודות טובות ורעות, האלגוריתם יודע לשפר את עצמו, לתת משקל גדול יותר לפרמטרים טובים ולהזניח פרמטרים פחות משמעותיים. אבל רגע, מי אמר למערכת להסתכל בכלל בסרטוני Unboxing?

האמת שהיא שאף אחד. מישוהו, בנאדם, אמר למערכת לזהות את כל הסרטונים שמשתמש צופה בהם ביוטיוב, לזהות מתוך הסרטון, האודיו, תיאור הסרטון ומילות המפתח וכו' – איזה סוג סרטון זה. ייתכן שאחרי מיליארדי צפיות בסרטונים, האלגוריתם מתחיל למצוא קשר בין סוג מסוים של סרטונים לבין פעולות כמו רכישה באתר. באופן הזה, גוגל מזינה את האלגוריתם בכל הפעולות שהמשתמש מבצע. המיילים שהוא קורא, המקומות שהוא מסתובב בהם, התמונות שהוא מעלה לענן, ההודעות שהוא שולח, כל מידע שיש אליו גישה. הכל נשפך לתוך מאגר הנתונים העצום בו מנסה גוגל לבנות פרופילים ולמצוא קשר בין פרופיל האדם לבין הסיכוי שלו לרכוש או כל פעולה אחרת שבא לה לזהות.

המכונה המופלאה הזו לומדת כל הזמן דברים חדשים ומנסה כל הזמן למצוא הקשרים, לחזות תוצאה, לבדוק אם היא הצליחה, ואם לא לתקן את עצמה שוב ושוב עד שהיא פוגעת במטרה. חשוב לציין שלמכונה אין סנטימנטים, כל המידע קביל ואם היא תמצא קשר מוכח בין מידת הנעליים של בנאדם לבין סרטונים של בייבי שארק, אז היא תשתמש בו גם אם זה לא נשמע הגיוני.

חשוב לשים לב לעניין המטרה – המטרה היא לא המצאה של האלגוריתם. הוא לא קם בבוקר ומחליט מה האפליקציה שלכם צריכה לעשות. המטרה מוגדרת על ידי היוצר של המערכת. למשל – חישוב זמן נסיעה, בניית מסלול אופטימלי בין A ל-B וכו'. המטרה של גוגל – שמשתמש יבצע רכישה, והכל מתנקז לזה בסוף, כי גוגל בראש ובראשונה היא מערכת פרסום. אגב, גם ההגדרה של מסלול "אופטימלי" היא מעשה ידי אדם. המכונה לא יודעת מה זה אופטימלי, זו רק מילה. אז צריך לעזור לה ולהגיד לה שאופטימלי זה מינימום זמן, מעט עצירות, כמה שפחות רמזורים וכו'. לסיכום, המטרה מאפיינת על ידי האדם ולא על ידי המכונה. המכונה רק חותרת למטרה שהוגדר לה.

יש מנגנוני ML המתבססים על דאטה מסודר ומתויג כמו ב-Netflix, עם כל המאפיינים של הסרטים אבל גם עם המאפיינים של הצופים (מדינה, גיל, שעת צפייה וכו'). לעומת זאת יש מנגנוני ML שמקבלים טיפה יותר חופש ומתבססים על מידע חלקי מאד (יש להם מידע על כל על הסרטים, אבל אין להם מידע על הצופה). מנגנונים אלו לא בהכרח מנסים לבנות מנוע המלצות אלא מנסים למצוא חוקיות בנתונים, חריגות וכו'.

כך או כך, המערך הסבוך הזה הקרוי ML בנוי מאלגוריתמים שונים המיומנים בניתוח טקסט, אלגוריתמים אחרים המתמקדים בעיבוד אודיו, כאלה המנתחים היסטוריית גלישה או זיהוי מתוך דף ה-Web בו אתם צופים ועוד. עשרות או מאות מנגנונים כאלה מסתובבים ורצים ובונים את המפה השלמה. ככה רוב רשתות הפרסום הגדולות עובדות. ככל שהמכונה של גוגל/פייסבוק תהיה חכמה יותר, ככה היא תדע להציג את המודעה המתאימה למשתמש הנכון, בזמן הנכון ועל ה-Device המתאים.

1.1.2 Data, Tasks and Learning

כאמור, המטרה הבסיסית של למידת מכונה היא היכולת להכליל מתוך הניסיון, ולבצע משימות באופן מדויק ככל הניתן על דאטה חדש שעדיין לא נצפה, על בסיס צבירת ניסיון מדאטה קיים. באופן כללי ניתן לדבר על שלושה סוגים של למידה:

למידה מונחית (supervised learning) – הדאטה הקיים הינו אוסף של דוגמאות, ולכל דוגמא יש תווית (label). מטרת האלגוריתמים במקרה זה היא לסווג דוגמאות חדשות שלא נצפו בתהליך הלמידה. באופן פורמלי, עבור דאטה $x \in \mathbb{R}^{n \times d}$, יש אוסף labels $y \in \mathbb{R}^{1 \times d}$, ומחפשים את האלגוריתם שמבצע את המיפוי $g: X \rightarrow Y$ בצורה הטובה ביותר, כלומר בהינתן דוגמא חדשה $x \in \mathbb{R}^n$, המטרה היא למצוא עבורה את ה- y הנכון. המיפוי נמדד ביחס לפונקציות מחיר, כפי שיוסבר בהמשך בנוגע לתהליך הלמידה.

למידה לא מונחית (unsupervised learning) – הדאטה הקיים הינו אוסף של דוגמאות במרחב, בלי שנתון עליהן מידע כלשהו המבחין ביניהן. במקרה זה, בדרך כלל האלגוריתמים יחפשו מודל המסביר את התפלגות הנקודות – למשל חלוקה לקבוצות שונות וכדומה.

למידה באמצעות חיזוקים (reinforcement learning) – הדאטה בו נעזרים אינו מצוי בתחילת התוכנית אלא נאסף עם הזמן. ישנם סוכנים הנמצאים בסביבה מסוימת ומעבירים מידע למשתמש, והוא בתורו ילמד אסטרטגיה בה הסוכנים ינקטו בצעדים הטובים עבורם.

האלגוריתמים השונים של הלמידה מתחלקים לשתי קבוצות – מודלים דיסקרימינטיביים המוציאים פלט על בסיס מידע נתון, אך לא יכולים ליצור מידע חדש בעצמם, ומודלים גנרטיביים, שלא רק לומדים להכליל את הדאטה הנלמד גם עבור דוגמאות חדשות, אלא יכולים גם להבין את מה שהם ראו וליצור מידע חדש על בסיס הדוגמאות שנלמדו.

כאמור, בשביל לבנות מודל יש צורך בדאטה. מודל טוב הוא מודל שמצליח להכליל מהדאטה הקיים גם לדאטה חדש. המודל למעשה מנסה למצוא דפוסים בדאטה הקיים, מהם הוא יוכל להסיק מסקנות גם על דוגמאות חדשות. כדי לוודא שהמודל אכן מצליח להכליל גם על דוגמאות חדשות, בדרך כלל מחלקים את הדאטה הקיים לשניים – קבוצת אימון (training set) וקבוצת מבחן (test set). סט האימון מאפשר לתת מדד להצלחת המודל – אם המודל מצליח למצוא דפוסים בסט האימון שנכונים גם עבור סט המבחן, זה סימן שהמודל הצליח למצוא כללים שיכולים להיות נכונים גם לדוגמאות חדשות שיבואו. לעיתים סט האימון מחולקת בעצמה לשניים – קבוצת דוגמאות עליהן המודל מתאמן, וקבוצת ולידציה (validation set) המסייעת להימנע מ-overfitting כפי שיוסבר בהמשך.

מגוון התחומים בהם משתמשים בכלים של למידה הוא עצום, עד כדי כך שכמעט ואין תחום בו לא נכנס השימוש באלגוריתמים לומדים. דוגמאות בולטות למשימות בהם משתמשים באלגוריתמים לומדים: סיווג, רגרסיה (מצאת קשר בין משתנים), חלוקה לקבוצות, מערכת המלצות, הורדת ממד, ראייה ממוחשבת, עיבוד שפה טבעית ועוד.

1.2 Applied Math

האלגוריתמים של למידת מכונה נסמכים בעיקרם על שלושה ענפים מתמטיים; אלגברה ליניארית, חשבון דיפרנציאלי והסתברות. בפרק זה נציג את העקרונות הנדרשים בלבד, ללא הרחבה, על מנת להבין את הנושאים הנדונים בספר זה.

1.2.1 Linear Algebra

וקטורים ומרחבים וקטורים

באופן מתמטי מופשט, וקטורים, המסומנים בדרך כלל ע"י \vec{x} או על ידי x , הינם אובייקטים הנמצאים במרחב וקטורי $(V, +)$ מעל שדה \mathbb{F} . מהו אותו מרחב וקטורי?

ראשית, השדה, \mathbb{F} , הוא קבוצת מספרים המקיימים תכונות מתמטיות מסוימות. לדיון בספר זה, השדה הוא קבוצת המספרים הממשיים \mathbb{R} , או קבוצת המספרים המרוכבים \mathbb{C} . שנית, נשים לב כי המרחב הווקטורי דורש גם הגדרת פעולת חיבור $(+)$.

כעת, $(V, +)$ היא מרחב וקטורי אם הוא מקיים את התכונות הבאות:

$$(I) \quad \text{קיים איבר אפס (וקטור אפס) כך שכל } \vec{x} \text{ בקבוצה } V \text{ מקיים: } \vec{x} + \vec{0} = \vec{0} + \vec{x} = \vec{x} \quad (I)$$

$$(II) \quad \text{לכל איבר בשדה } a \text{ ולכל } \vec{x} \text{ בקבוצה } V, \text{ גם } a \cdot \vec{x} + \vec{y} \text{ הינו איבר בקבוצה } V. \quad (II)$$

הערה: קיימות דרישות נוספות למרחב וקטורי, אך הן מעבר לנדרש בספר זה.

דוגמאות:

א. וקטורים גאומטריים:

מערך חד ממדי $\vec{x} = (x_1, x_2, \dots, x_n)$ (n -יה סדורה) נקרא וקטור גאומטרי n ממדי, כאשר רכיבי הווקטור הם איברים בשדה \mathbb{F} . האיבר x_i , המיוצג על ידי האינדקס i מתאר את מיקום האיבר. מרחב זה מסומן ע"י \mathbb{F}^n . נראה שמרחב זה הוא אכן מרחב וקטורי:

חיבור וקטורים:

$$\vec{x} = (x_1, x_2, \dots, x_n), \quad \vec{y} = (y_1, y_2, \dots, y_n) \quad \rightarrow \quad \vec{x} + \vec{y} = (x_1 + y_1, x_2 + y_2, \dots, x_n + y_n)$$

וקטור אפס:

$$\vec{0} = (0, 0, \dots, 0)$$

כפל בסקלר:

$$\vec{x} = (x_1, x_2, \dots, x_n) \quad \rightarrow \quad a \vec{x} = (a x_1, a x_2, \dots, a x_n)$$

הערה: לשם פשטות, בהמשך, נכנה וקטור גאומטרי כ"וקטור" בלבד.

ב. מטריצות:

מעריך דו ממדי $\begin{pmatrix} A_{11} & \dots & A_{1m} \\ \vdots & \ddots & \vdots \\ A_{n1} & \dots & A_{nm} \end{pmatrix}$, אשר רכיביו הם איברים בשדה \mathbb{F} , נקרא מטריצה מסדר $n \times m$, כאשר n הוא מספר השורות ו- m הוא מספר העמודות במערך. האיברים במטריצה A_{ij} מיוצגים ע"י שני אינדקסים – i, j , המתארים את השורה והעמודה בהתאמה. מרחב זה מסומן בדרך כלל ע"י $\mathbb{F}^{n \times m}$. נוכיח שמרחב זה הוא אכן מרחב וקטורי:

חיבור מטריצות:

$$\hat{A} = \begin{pmatrix} A_{11} & \dots & A_{1m} \\ \vdots & \ddots & \vdots \\ A_{n1} & \dots & A_{nm} \end{pmatrix}, \hat{B} = \begin{pmatrix} B_{11} & \dots & B_{1m} \\ \vdots & \ddots & \vdots \\ B_{n1} & \dots & B_{nm} \end{pmatrix} \rightarrow \hat{A} + \hat{B} = \begin{pmatrix} A_{11} + B_{11} & \dots & A_{1m} + B_{1m} \\ \vdots & \ddots & \vdots \\ A_{n1} + B_{n1} & \dots & A_{nm} + B_{nm} \end{pmatrix}$$

מטריצת אפס:

$$\hat{0} = \begin{pmatrix} 0 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & 0 \end{pmatrix}$$

כפל בסקלר:

$$\hat{A} = \begin{pmatrix} A_{11} & \dots & A_{1m} \\ \vdots & \ddots & \vdots \\ A_{n1} & \dots & A_{nm} \end{pmatrix} \rightarrow a \hat{A} = \begin{pmatrix} a A_{11} & \dots & a A_{1m} \\ \vdots & \ddots & \vdots \\ a A_{n1} & \dots & a A_{nm} \end{pmatrix}$$

ניתן להבחין כי הווקטורים הגיאומטריים שהוגדרו בדוגמא א, הם בעצם מטריצות בממד $n \times 1$.

ג. פולינומים:

פולינומים מסדר n הינם ביטויים מהסוג $a_0 + a_1x + a_2x^2 + \dots + a_nx^n$, כאשר n מייצג את החזקה הגדולה ביותר ו- a_i הם איברים בשדה. מרחב זה מסומן בדרך כלל ע"י $P_n(x)$.

בכל הדוגמאות לעיל קל להראות שהן אכן מהוות מרחב וקטורי. רשימה חלקית לדוגמאות נוספות לווקטורים (ולמרחבים וקטורים) כוללת למשל מרחבי פונקציות או אפילו אותות אלקטרומגנטיים. כאן בחרנו להציג רק את הדוגמאות הרלוונטיות לספר זה.

פעולות חשבון על מטריצות ווקטורים:

כמו שנזכר לעיל, הווקטורים הגיאומטריים שהוגדרו בדוגמא א, הם בעצם מטריצות בממד $n \times 1$. לכן, פעולות החשבון מוגדרות באופן זהה.

• **חיבור וחסור בין שתי מטריצות:**

$\hat{A}, \hat{B} \in \mathbb{F}^{n \times m}$ כאשר A_{ij}, B_{ij} הם האיברים בשורה i בעמודה j של המטריצות \hat{A}, \hat{B} בהתאמה. אז, האיבר בשורה i בעמודה j של מטריצת הסכום (או הפרש) הינו

$$(A \pm B)_{ij} = A_{ij} \pm B_{ij}$$

(הגדרת חיבור המטריצות בעצם כבר ניתנה בדוגמא א לעיל)

שים לב: ניתן לחבר ולחסר מטריצות רק בעלות אותו הממד.

• **כפל בין שתי מטריצות:**

$\hat{A} \in \mathbb{F}^{n \times k}, \hat{B} \in \mathbb{F}^{k \times m}$ הן שתי מטריצות, כאשר מספר העמודות במטריצה \hat{A} שווה למספר השורות של מטריצה \hat{B} (אך שתי המטריצות אינן בהכרח בעלות אותו ממד). במקרה כזה, מכפלת המטריצות מוגדרת על ידי:

$$\hat{A} \cdot \hat{B} = \begin{pmatrix} A_{11}B_{11} + A_{12}B_{21} + \dots + A_{1k}B_{k1} & \dots & A_{11}B_{1n} + \dots + A_{1k}B_{kn} \\ \vdots & \ddots & \vdots \\ A_{m1}B_{11} + \dots + A_{mk}B_{kn} & \dots & A_{n1}B_{1m} + \dots + A_{nk}B_{km} \end{pmatrix}$$

למעשה כל איבר בתוצאה הינו סכום של מכפלת שורה i ממטריצה A בעמודה j ממטריצה B :

$$(\hat{A} \cdot \hat{B})_{ij} = \sum_r A_{ir} B_{rj}$$

שים לב: על מנת שכפל המטריצות יהיה מוגדר מספר העמודות ב- \hat{A} שווה למספר השורות ב- \hat{B} . עבור מטריצות ריבועיות (מסדר $n \times n$), מוגדר גם הכפל $\hat{A}\hat{B}$ וגם הכפל $\hat{B}\hat{A}$, אולם ייתכן ש- $\hat{A}\hat{B} \neq \hat{B}\hat{A}$.

• **שחלוף (transpose):**

החלפת שורות בעמודות, או 'סיבוב' המטריצה. נניח מטריצה $\hat{A} \in \mathbb{F}^{n \times m}$, אז השחלוף שלה, המסומן כ- \hat{A}^T הוא:

$$(\hat{A}^T)_{ij} = A_{ji}$$

ובאופן מפורש:

$$\hat{A} = \begin{pmatrix} A_{11} & \dots & A_{1m} \\ \vdots & \ddots & \vdots \\ A_{n1} & \dots & A_{nm} \end{pmatrix} \rightarrow \hat{A}^T = \begin{pmatrix} A_{11} & \dots & A_{n1} \\ \vdots & \ddots & \vdots \\ A_{1m} & \dots & A_{nm} \end{pmatrix}$$

שים לב שהמטריצה החדשה \hat{A}^T הינה בממד $m \times n$. בנוסף ניתן להוכיח כי מתקיים: $\hat{A} = (\hat{A}^T)^T$. שחלוף של וקטור שורה, נותן וקטור עמודה ולהפך.

• **מטריצת יחידה:**

מטריצת יחידה, הינה מטריצה ריבועית (מסדר $n \times n$), המסומנת על ידי \mathbb{I}_n ומוגדרת כך שכל איבריה אפס מלבד איברי האלכסון הראשי המקבילים את הערך 1:

$$(\mathbb{I}_n)_{ij} = \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases}$$

ובאופן מפורש:

$$\mathbb{I}_n = \begin{pmatrix} 1 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & 1 \end{pmatrix}$$

מטריצה זו מקיימת $\hat{A} \cdot \mathbb{I}_m = \mathbb{I}_n \cdot \hat{A} = \hat{A}$ לכל מטריצה \hat{A} מסדר $n \times m$. הערה: לעיתים סדר מטריצת היחידה אינו משנה או טריוויאלי, ולכן המטריצה מסומנת רק על ידי \mathbb{I} ללא ציון הממד.

• **מטריצה הופכית:**

למטריצות ריבועיות (מטריצות עם מספר זהה של שורות ועמודות; מסדר $n \times n$) ייתכן שיש מטריצה הופכית \hat{A}^{-1} שמקיימת את הקשר:

$$\hat{A} \cdot \hat{A}^{-1} = \hat{A}^{-1} \cdot \hat{A} = \mathbb{I}_n$$

דוגמא: $\hat{A} = \hat{A}^{-1} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$, לכן, במקרה הזה $\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} = \mathbb{I}_2$.

• **מטריצה צמודה/הרמיטית:**

עבור מטריצה A , המטריצה $A^* = A^\dagger$ נקראת הצמוד הרמיטי של A , ומתקיים:

$$(A^*)_{ij} = \overline{A_{ji}}$$

הצמוד הרמיטי הוא שחלוף של A , כאשר לכל איבר במטריצה המשוחלפת לוקחים את הצמוד המרוכב. אם A מטריצה ממשית, המטריצה הצמודה שלה היא למעשה המטריצה המשוחלפת של A .

• **מטריצה אוניטרית:**

מטריצה אוניטרית היא מטריצה ריבועית מעל המספרים המרוכבים המקיימת את התנאי:

$$A^*A = AA^* = I$$

מערכת משוואות לינאריות:

מערכת משוואות לינאריות מוצגת באופן כללי באופן הבא:

$$\begin{matrix} A_{11}x_1 + A_{12}x_2 + & \dots & + A_{1n}x_n = b_1 \\ & \vdots & \vdots \\ A_{m1}x_1 + A_{m2}x_2 + & \dots & + A_{mn}x_n = b_m \end{matrix}$$

נשים לב כי מערכת משוואות לינארית ניתנת לייצוג באופן קומפקטי על ידי הפרדה בין רשימת המשתנים, המקדמים של משתנה, והאיבר החופשי, באופן הבא:

$$\hat{A} \vec{x} = \vec{b} = \begin{pmatrix} A_{11} & \dots & A_{1n} \\ \vdots & \ddots & \vdots \\ A_{m1} & \dots & A_{mn} \end{pmatrix} \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ \vdots \\ b_m \end{pmatrix}$$

מטריצה $\hat{A} = \begin{pmatrix} A_{11} & \dots & A_{1n} \\ \vdots & \ddots & \vdots \\ A_{m1} & \dots & A_{mn} \end{pmatrix}$ הינה מטריצת המקדמים מסדר $n \times m$, כאשר n הוא מספר המשתנים, ו- m הוא מספר המשוואות במערכת.

וקטור $\vec{x} = \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix}$, הינו וקטור עמודה (לעיתים גם מסומן על ידי $(x_1, x_2, \dots, x_n)^T$), המייצג את וקטור המשתנים.

וקטור $\vec{b} = \begin{pmatrix} b_1 \\ \vdots \\ b_m \end{pmatrix}$, הינו וקטור עמודה, שאיבריו הם האיבר החופשי.

הפתרונות של מערכת המשוואות הלינארית, $\hat{A} \vec{x} = \vec{b}$, אם הם קיימים ויחידים, נתונים ע"י $\vec{x} = \hat{A}^{-1} \vec{b}$ (בהנחה והמטריצה ריבועית).

מכפלה פנימית, נורמה, אורתוגונליות

מרחב מכפלה פנימית, מוגדר על ידי מרחב וקטורי V (המוגדר על גבי שדה \mathbb{F}) ועל ידי פעולת "מכפלה פנימית". מכפלה פנימית, הנקראת לעיתים רק מכפלה, הינה בעצם פונקציה המקבלת שני וקטורים ממרחב וקטורי V ומחזירה סקלר (=מספר) בשדה \mathbb{F} . מכפלה זו, מסומנת בדרך כלל ע"י $\langle \cdot, \cdot \rangle$ (או ע"י $\langle \cdot | \cdot \rangle$), חייבת לקיים מספר תכונות.

לכל $\vec{v}, \vec{u}, \vec{w} \in V$ (כל שלושה וקטורים במרחב הווקטורי V), ולכל $\lambda \in \mathbb{F}$ (סקלר בשדה \mathbb{F}):

- $\langle \vec{v} + \vec{u}, \vec{w} \rangle = \langle \vec{v}, \vec{w} \rangle + \langle \vec{u}, \vec{w} \rangle$
- $\langle \lambda \vec{v}, \vec{u} \rangle = \lambda \langle \vec{v}, \vec{u} \rangle$
- $\overline{\langle \vec{v}, \vec{u} \rangle} = \langle \vec{u}, \vec{v} \rangle$
- $\langle \vec{v}, \vec{v} \rangle \geq 0$

ההגדרה עצמה של המכפלה משתנה כתלות במרחב הווקטורי הנתון. לדוגמא:

א. מכפלה סקלרית על מרחב הווקטורים הגיאומטריים:

נתונים $\vec{v}, \vec{u} \in \mathbb{C}^n$ וקטורים גיאומטריים מסדר n מעל שדה המספרים המרוכבים. מכפלה פנימית בין שני וקטורים אלו, נקראת גם מכפלה סקלרית, המוגדרת על ידי:

$$\langle \vec{v}, \vec{u} \rangle = \vec{v}^T \cdot \vec{u} = \sum_{i=1}^n \bar{v}_i u_i$$

כאשר \bar{v}_i הינו הצמוד המרוכב של v_i .

ב. מרחב הילברט – מרחב מכפלה פנימית על מרחב הפונקציות:

נניח שתי פונקציות מרוכבות $f: \mathbb{C} \rightarrow \mathbb{C}$ אינטגרביליות בתחום כלשהו I (כמו שהוזכר לעיל, גם מרחב הפונקציות הוא מרחב וקטורי), אז המכפלה פנימית מוגדרת על ידי:

$$\langle f(x), g(x) \rangle = \int_I f^*(x)g(x)dx$$

כאשר $f^*(x)$ הינו הצמוד המרוכב של $f(x)$.

ניתן להגדיר גם מרחבי מכפלה פנימית נוספים, נניח עבור מרחב המטריצות.

נורמה:

נורמה, מוגדרת על ידי מכפלה פנימית של וקטור בעצמו, ומסומנת ע"י $\|\cdot\|$, זאת אומרת:

$$\|\vec{v}\| = \sqrt{\langle \vec{v}, \vec{v} \rangle} \geq 0$$

שוויון מתקיים אך ורק עבור וקטור האפס; $\|\vec{v}\| = 0 \Leftrightarrow \vec{v} = 0$.

תכונה נוספת, נקראת אי-שיוון המשולש, מתוארת על ידי:

$$\|\vec{v} + \vec{u}\| \leq \|\vec{v}\| + \|\vec{u}\|$$

אי שיוויון נוסף הקשור לנורמות נקרא אי שיוויון קושי שוורץ (Cauchy-Schwarz inequality):

$$|\langle x, y \rangle| \leq \|x\| \cdot \|y\|$$

כאשר $\langle x, y \rangle$ הינה המכפלה הפנימית בין שני הווקטורים, המוגדרת מעל הטבעיים כך: $\langle x, y \rangle = \sum_i x_i \cdot y_i$, והביטוי $\|x\| \cdot \|y\|$ הוא מכפלת הנורמות.

דוגמא:

א. במרחב הווקטורים הגיאומטריים, הגדרת הנורמה היא בעצם הגדרת אורך (או גודל הווקטור). נניח עבור הווקטורים הגיאומטריים התלת-ממדיים, $V = \mathbb{R}^3$, אז עבור $\vec{v} = (x, y, z) \in V$, הנורמה מוגדרת ע"י $\|\vec{v}\| = \sqrt{x^2 + y^2 + z^2}$.

ב. במרחב הילברט נורמה של פונקציה $f: \mathbb{C} \rightarrow \mathbb{C}$ הינה $\|f\|^2 = \int_I |f(x)|^2 dx$.

אורתוגונליות

הגדרת מכפלה פנימית מאפשרת לנו להגדיר אורתוגונליות (או אנכיות) של שני וקטורים במרחב מכפלה פנימית מסוים. שני וקטורים $\vec{v}, \vec{u} \in V$ נקראים אורתוגונליים זה לזה אם ורק אם המכפלה הפנימית שלהם הינה אפס:

$$\vec{v} \perp \vec{u} \Leftrightarrow \langle \vec{v}, \vec{u} \rangle = 0$$

כאשר מתייחסים למרחב הווקטורים הגיאומטריים, קל להבין את מושג האורתוגונליות.

אורתוגונליות היא הכללה של תכונת הניצבות המוכרת מגאומטריה. בגאומטריה, שני ישרים במישור האוקלידי ניצבים זה לזה אם הזווית הנוצרת בנקודת החיתוך שלהם היא זווית ישרה (בת 90 מעלות). מושג האורתוגונליות מכליל תכונה זו גם למרחבים ווקטוריים n -ממדיים. על מנת להכליל את מושג הניצבות יש ראשית להגדיר זווית בין שני וקטורים:

$$\cos(\theta) = \frac{\langle x, y \rangle}{\|x\| \cdot \|y\|}$$

לפי אי שיוויון קושי שוורץ מתקיים: $\langle x, y \rangle \leq \|x\| \cdot \|y\|$, ולכן הביטוי באגף ימין תמיד קטן או שווה בערכו המוחלט ל-1. כיוון שכך, תמיד ניתן לחשב זווית בין שני וקטורים בעזרת מכפלה פנימית.

לוקטורים אורתוגונליים חשיבות רבה כאשר חוקרים מרחבים וקטורים. לבסיס של מרחב וקטורי יש מספר תכונות נוחות כאשר הוא אורתונורמלי (כל אבריו אורתוגונליים זה לזה ובעלי אורך 1). יתר על כן, מתברר שבהינתן בסיס כלשהו למרחב וקטורי ניתן לקבל ממנו בסיס חדש שכל אבריו אורתוגונליים זה לזה, כך שתמיד ניתן למצוא בסיס נוח שכזה. דבר זה נעשה על ידי תהליך גרם-שמידט (gram-schmidt).

שני וקטורים אורתוגונליים יסומנו על ידי \perp . עבור וקטורים אורתוגונליים מתקיימות התכונות הבאות:

- אם $u \perp v$, אז $u \perp \lambda v$.
- אם $u \perp v$, אז לכל סקלר λ גם $\lambda u \perp v$.
- אם $u \perp v$ וגם $w \perp v$, אז $(w + u) \perp v$.
- אם וקטור אורתוגונלי לקבוצה של וקטורים אזי הוא גם אורתוגונלי לכל צירוף לינארי שלהם (נובע משתי התכונות הקודמות).

וקטורים עצמיים וערכים עצמיים

תהי $A \in \mathbb{F}^{n \times n}$ מטריצה ריבועית, וקטור $v \in \mathbb{F}^n$ ו- $\lambda \in \mathbb{F}$ סקלר. λ נקרא ערך עצמי של A ו- $v \neq 0$ נקרא הווקטור העצמי המתאים אם מתקיים:

$$A \cdot v = \lambda \cdot v$$

ניתן להראות שעבור מטריצה A , הווקטורים העצמיים המתאימים לסקלר λ הם כל פתרונות המשוואה ההומוגנית $(A - \lambda I_n)v = 0$.

אם נסמן $V = [v_1, \dots, v_n]$ ו- $\Lambda = [\lambda_1, \dots, \lambda_n]$, אזי מתקיים:

$$A = V \text{diag}(\Lambda) V^{-1}$$

כאשר $\text{diag}(\Lambda)$ הוא ערכי האלכסון של המטריצה Λ .

פירוק לערכים סינגולריים

ניתן לפרק מטריצה $M \in \mathbb{R}^{m \times n}$ למכפלה של שלוש מטריצות באופן הבא:

$$M = U \Sigma V^*$$

כאשר $U \in \mathbb{C}^{m \times m}$ היא מטריצה אוניטרית מרוכבת (או ממשית), $\Sigma \in \mathbb{R}^{m \times n}$ היא מטריצה אלכסונית שכל איברי האלכסון שלה ממשיים ואי-שליליים, ו- $V^* \in \mathbb{C}^{n \times n}$ היא מטריצה אוניטרית מרוכבת (או ממשית). פירוק זה נקרא פירוק לערכים סינגולריים (Singular value decomposition - SVD).

ערכי האלכסון של Σ – מסודרים מהגדול לקטן, והם נקראים הערכים הסינגולריים של M . בנוסף, m העמודות של U נקראות הווקטורים הסינגולריים השמאליים של M , ובהתאמה n העמודות של V הן הווקטורים הסינגולריים הימניים של M . שלוש המטריצות מקיימות את התכונות הבאות:

- הווקטורים הסינגולריים השמאליים של M הם וקטורים עצמיים של MM^* .
- הווקטורים הסינגולריים הימניים של M הם וקטורים עצמיים של M^*M .
- הערכים הסינגולריים (איברי האלכסון של Σ) שאינם אפס הם שורשים ריבועיים של הערכים עצמיים השונים מאפס של MM^* ושל M^*M .

לפירוק SVD יש שימושים בתחומים רבים, ואף ניתן להגדיר בעזרתו נורמות חדשות.

1.2.2 Calculus

פונקציה

פונקציה הינה התאמה (או העתקה), המתאימה לכל איבר x (בתחום מסוים), ערך יחיד y , ומסומנת באופן הבא: $y = f(x)$. קבוצת ה- x ים, נקראת *תחום*, וקבוצת ה- y ים נקראת *טווח*. קבוצות התחום והטווח יכולות להיות רציפות (למשל מספרים ממשיים חיוביים) או בדידות (למשל קבוצה $\{0,1\}$). בדרך כלל הסימון מופיע כך: $f: X \rightarrow Y$, כאשר X ו- Y הינם התחום והטווח בהתאמה.

דוגמא: $\mathbb{R}^2 \rightarrow \mathbb{R}^+$: $\|\cdot\|$, הינה פונקציה, הלוקחת וקטורים גיאומטריים דו-ממדים, ומחזירה מספר ממשי אי שלילי. הפונקציה עצמה $\|\cdot\|$ היא הנורמה של הווקטור, כפי שהוגדרה בפרק הקודם.

נגזרת

עבור פונקציות ממשיות, נגזרת מוגדרת על ידי מידת השתנות של הפונקציה $f(x)$ על ידי שינוי קטן (אינפיניטסימלי) dx . באופן גיאומטרי, הנגזרת הינה השיפוע של הפונקציה בנקודה x . נגזרת מסומנת בדרך כלל ע"י $f'(x) = \frac{df}{dx}$.

נגזרות של פונקציות אלמנטריות ניתן לחשב באמצעות כללים ידועים. לדוגמא:

- לכל $n \neq 0$ מתקיים: $\frac{d(x^n)}{dx} = nx^{n-1}$.
- חיבור או חיסור פונקציות: $\frac{d(f(x)+g(x))}{dx} = \frac{df(x)}{dx} + \frac{dg(x)}{dx}$.
- מכפלת שתי פונקציות: $\frac{d(f(x) \cdot g(x))}{dx} = f(x) \frac{dg(x)}{dx} + g(x) \frac{df(x)}{dx}$.
- כלל שרשרת: $\frac{df(g(x))}{dx} = \frac{df}{dg} \frac{dg}{dx}$.

כיוון שנגזרת של פונקציה ממשית מכמתת את קצב שינוי הפונקציה, אז בתחום שבו הפונקציה יורדת הנגזרת שם תהיה שלילית, ובתחום שבו היא עולה הנגזרת תהיה חיובית. ככל שקצב ההשתנות גדול יותר כך ערכה המוחלט של הנגזרת גדל.

הערה: לא לכל פונקציה מוגדרת נגזרת. לספר זה נניח שהפונקציה אנליטית ולכן גזירה.

הערה נוספת: כיוון שנגזרת של פונקציה היא גם פונקציה, ניתן גם להגדיר נגזרת שניה או נגזרת מסדרים גבוהים יותר. בדרך כלל הסימון הינו $f^{(n)}(x) = \frac{d^n f}{dx^n}$ לנגזרת מסדר שני וכולי

נקודות אקסטremום

נקודות אקסטremום של פונקציה, הן נקודות שבהם הפונקציה מקבלת ערך מקסימום או מינימום באופן מקומי. בנקודות אלו, הנגזרת של הפונקציה "משנה כיוון" (מפונקציה עולה לפונקציה יורדת או להפך) ולכן מקבלת את הערך אפס. יש לשים לב שהתאפסות הנגזרת בנקודות המינימום והמקסימום היא תנאי הכרחי אך לא מספיק. ייתכן שהנגזרת מתאפסת בנקודה מסוימת, אך נקודה זו אינה מינימום או מקסימום מקומי, אלא נקודת פיתול.

לדוגמא: $f(x) = x^3$. נגזרת הפונקציה הינה $f'(x) = 3x^2$ והיא מתאפסת בנקודה $x = 0$.

גרדיאנט, יעקוביאן והסיאן

עבור פונקציה מרובת משתנים, נגזרת חלקית מוגדרת להיות הנגזרת של הפונקציה לפי אחד המשתנים שלה, והיא מסומנת ב- $\frac{\partial f(x_1, \dots, x_n)}{\partial x_i}$. כאשר גוזרים לפי משתנה מסוים, שאר המשתנים הם קבועים ביחס לנגזרת. בהינתן הפונקציה $f(x_1, \dots, x_n)$, וקטור הנגזרות לפי כל המשתנים נקרא גרדיאנט:

$$\nabla f(x_1, \dots, x_n) = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \vdots \\ \frac{\partial f}{\partial x_n} \end{bmatrix} \leftrightarrow [\nabla f]_i = \frac{\partial f}{\partial x_i}$$

עבור m פונקציות התלויות ב- n משתנים, היעקוביאן הוא מטריצת הנגזרות החלקיות:

$$\mathcal{J}_f = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \dots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \dots & \frac{\partial f_m}{\partial x_n} \end{bmatrix}_{n \times m} \leftrightarrow [\mathcal{J}_f]_{ij} = \frac{\partial f_i}{\partial x_j}$$

עבור פונקציה $f(x_1, \dots, x_n)$, מטריצת הנגזרות מסדר שני נקראת הסיאן:

$$\mathcal{H}_f = \nabla^2 f = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix}_{n \times n} \leftrightarrow [\mathcal{H}_f]_{ij} = \frac{\partial^2 f}{\partial x_i \partial x_j}$$

שני כללים חשובים בחשבון נגזרות של מטריצות:

$$\nabla_x(a^T x) = a$$

$$\nabla_x(x^T A x) = (A + A^T)x$$

1.2.3 Probability

תורת ההסתברות היא תחום המספק כלי ניתוח למאורעות המכילים ממד של אקראיות ואינם דטרמיניסטיים. הסתברות של מאורע הוא ערך מספרי למידת הסבירות שהוא יתרחש, כאשר ערך זה נע בין 0 ל-1 – מאורע בלתי אפשרי הוא בעל הסתברות 0, ומאורע ודאי הוא בעל הסתברות 1.

הגדרות בסיסיות

Ω = מרחב המדגם – מכלול האפשרויות השונות של ניסוי. לדוגמה עבור הטלת קוביה: $\Omega = \{1,2,3,4,5,6\}$.

קבוצה – חלק ממרחב המדגם. לדוגמה עבור הטלת קוביה: $A = \{2, 4, 6\} = \text{even number}$.

מאורע – תוצאה אפשרית של ניסוי.

הסתברות – סיכוי של מאורע להתרחש. עבור תת קבוצה A של מרחב המדגם Ω , ההסתברות לקיום מאורע מקבוצת A שווה לחלק היחסי של מספר איברי הקבוצה מתוך קבוצת המדגם:

$$p(A) = \frac{\#A}{\#\Omega}, 0 \leq p(A) \leq 1$$

$A \cup B$ = איחוד – איחוד של שתי קבוצות הוא אוסף האיברים של שתי הקבוצות. איחוד של הקבוצות A ו- B הוא אוסף האיברים המופיעים לפחות באחת משתי הקבוצות A או B . לדוגמה עבור הטלת קוביה:

$$A = \text{even number} = \{2, 4, 6\}, B = \text{lower than 4} = \{1,2,3\}$$

$$\rightarrow A \cup B = \{1,2,3,4,6\}, \quad p(A \cup B) = \frac{5}{6}$$

$A \cap B$ = חיתוך – חיתוך של שתי קבוצות הוא אוסף האיברים המופיעים בשתי הקבוצות. חיתוך של הקבוצות A ו- B הוא אוסף האיברים המופיעים גם ב- A וגם ב- B . עבור הדוגמה הקודמת:

$$A \cap B = \{2\}, p(A \cap B) = \frac{1}{6}$$

מאורעות זרים – מאורעות שהחיתוך שלהם ריק, כלומר אין להם איברים משותפים:

$$A \cap B = \emptyset, p(A \cap B) = 0$$

מאורע משלים – מאורע המכיל את כל האיברים שאינם נמצאים בקבוצה מסוימת:

$$A \cup A^c = \Omega \rightarrow p(A \cup A^c) = 1, p(A) = 1 - p(A^c)$$

מאורעות בלתי תלויים: $P(A \cap B) = P(A) \cdot P(B)$. באופן אינטואיטיבי ניתן לחשוב על כך שבמקרה כזה ידיעת האחד אינה משפיעה על הסיכוי של השני.

אם המאורעות זרים (והם בעלי סיכוי שונה מ-0), הם בהכרח תלויים:

$$P(A \cap B) = 0 \neq P(A) \cdot P(B) > 0$$

$p(A|B)$ = הסתברות מותנית – בהינתן מידע מסוים, מה ההסתברות של מאורע כלשהו:

$$p(A|B) = \frac{p(A \cap B)}{p(B)} \leftrightarrow p(A|B) \cdot p(B) = p(A \cap B) = p(B|A) \cdot p(A)$$

בעזרת ההגדרה של הסתברות מותנית ניתן לתת הגדרה נוספת למאורעות בלתי תלויים:

$$A, B \text{ בלתי תלויים} \leftrightarrow p(A|B) = p(A)$$

נשים לב שהמשמעות של שתי ההגדרות זהה – המידע על B לא משנה את חישוב ההסתברות של A .

נוסחת ההסתברות השלמה וחוק בייס

נוסחת ההסתברות השלמה היא נוסחה פשוטה המאפשרת לחשב מאורעות מסובכים. ניתן לפרק מרחב הסתברות לאיברים זרים, ואז לחשב את ההסתברות של כל איבר בפני עצמו. אם ניקח את כל ההסתברויות המתקבלות, ונכפיל כל אחת מהן במשקל של אותו איבר, נקבל את נוסחת ההסתברות השלמה:

$$P(B) = \sum_i P(B|A_i) \cdot P(A_i)$$

מתוך נוסחה זו מגיעים בקלות לחוק בייס, המאפשרת לחשב הסתברות מותנית באמצעות ההתניה הפוכה:

$$p(A|B) = \frac{p(A \cap B)}{p(B)} = \frac{p(B|A)p(A)}{p(B)}$$

משפט ההכלה וההדחה

כדי לספור עצמים בקבוצה, אפשר לכלול ולהוציא את אותו עצם שוב ושוב, כל עוד בסוף ההליך נספר כל עצם פעם אחת. עקרון פשוט זה מתורגם לנוסחה הבאה:

$$\left| \bigcup_{i=1}^n A_i \right| = \sum_{i=1}^n |A_i| - \sum_{1 \leq i < j \leq n} |A_i \cap A_j| + \sum_{1 \leq i < j < k \leq n} |A_i \cap A_j \cap A_k| - \dots + (-1)^{n-1} |A_1 \cap \dots \cap A_n|$$

עבור 2 קבוצות הנוסחה נהיית יותר פשוטה:

$$|A \cup B| = |A| + |B| - |A \cap B|$$

במקרה זה, כאשר A, B זרות, אז $|A \cap B| = 0$.

עבור שלוש קבוצות מתקבלת הנוסחה:

$$|A \cup B \cup C| = |A| + |B| + |C| - |A \cap B| - |B \cap C| - |A \cap C| + |A \cap B \cap C|$$

משתנים אקראיים

$X: \Omega \rightarrow \mathbb{R}$ משתנה מקרי – פונקציה המתאימה לכל מאורע השייך למרחב ההסתברות ערך מספרי, המהווה את הסיכוי של המאורע להתרחש.

פונקציית ההסתברות של משתנה מקרי X נותנת את הסיכוי של כל אפשרי:

$$f_X: \mathbb{R} \rightarrow [0,1] = p(X = x)$$

פונקציה זו מקיימת שלוש אקסיומות:

- הסתברות של כל מאורע במרחב המדגם גדולה או שווה ל-0.
- סכום ההסתברויות של כל המאורעות במרחב שווה ל-1: $\sum p(X = x) = p(\Omega) = 1$.
- סכום ההסתברויות של שני מאורעות זרים שווה להסתברות של איחוד המאורעות.

עבור משתנה מקרי רציף יש אינסוף מאורעות אפשריים, לכן ההסתברות של כל מאורע יחיד היא 0. לכן עבור משתנה מקרי רציף מכלילים את פונקציית ההסתברות לפונקציה הנקראת פונקציית ההתפלגות (או פונקציית הצפיפות המצטברת), המחשבת את ההסתברות שמאורע יהיה קטן מערך מסוים:

$$F_X(a) = p(X \leq a) = \int_{-\infty}^a f_X(x) dx$$

ניתן לחשב בעזרת פונקציה זו את ההסתברות שמאורע יהיה בטווח מסוים:

$$p(a \leq X \leq b) = F_X(b) - F_X(a) = \int_a^b f_X(x) dx$$

פונקציית ההתפלגות מקיימת את התכונות הבאות:

- $\lim_{a \rightarrow -\infty} F_X(a) = 0$
- $\lim_{a \rightarrow \infty} F_X(a) = 1$
- $\int_{-\infty}^{\infty} f_X(x) dx = 1$
- הפונקציה מונוטונית עולה במובן החלש: לכל $a \leq b$ מתקיים $F_X(a) \leq F_X(b)$
- $p(X \geq a) = 1 - F_X(a)$
- הפונקציה גזירה ומתקיים $\frac{d}{dx} F(x) = f(x)$

תכונות ופרמטרים עבור משתנה מקרי

תוחלת – ממוצע משוקלל של כל הערכים האפשריים, כאשר כל ערך מוכפל בהסתברות שלו (במקרה הבדיד – סכום ובמקרה הרציף – אינטגרל):

$$\mathbb{E}[X] = \sum_i x_i P(X = x_i) / \int_{-\infty}^{\infty} x f(x) dx$$

תכונות:

- $\mathbb{E}[c] = c$
- $\mathbb{E}[\mathbb{E}[X]] = \mathbb{E}[X]$
- לינאריות התוחלת: $\mathbb{E}[aX + b] = a\mathbb{E}[X] + b$, $\mathbb{E}[X + Y] = \mathbb{E}[X] + \mathbb{E}[Y]$

שונות – מדד פיזור הערכים ביחס לממוצע המשוקלל (-התוחלת):

$$Var[x] = E[(X - \mathbb{E}[X])^2] = \mathbb{E}[X^2] - (\mathbb{E}[X])^2 = \int_{-\infty}^{\infty} x^2 f(x) dx - (\mathbb{E}[X])^2$$

סטיית תקן מוגדרת להיות שורש השונות: $\sigma = \sqrt{Var[X]}$

תכונות:

- אי שליליות: $Var[x] \geq 0$
- $Var[aX + b] = a^2 V[X]$

שונות משותפת – מדד ליחס אפשרי בין שני משתנים מקריים:

$$cov(X, Y) = \mathbb{E}[X \cdot Y] - \mathbb{E}[X] \cdot \mathbb{E}[Y]$$

כאשר: $\mathbb{E}[X \cdot Y] = \sum_i \sum_j x_i y_j P(X = x_i \cap Y = y_j)$. אם המשתנים בלתי תלויים אז מתקיים $\mathbb{E}[X \cdot Y] = 0$.

מקדם המתאם – נרמול של השונות המשותפת: $\rho(X, Y) = \frac{cov(X, Y)}{\sqrt{V(X)V(Y)}}$. המקדם מקיים: $|\rho| \leq 1$.

שני משתנים מקריים מוגדרים בלתי מתואמים אם $cov(X, Y) = 0$. אם המשתנים בלתי תלויים אז הם בהכרח בלתי מתואמים.

בעזרת השונות המשותפת ניתן לכתוב: $V[X + Y] = V[X] + V[Y] + 2 \cdot cov(X, Y)$

פונקציה יוצרת מומנטים (התמרת לפלס של פונקציית הצפיפות – $M_X(t) = \mathcal{L}\{f_X(-t)\}$)

$$M_X(t) = \mathbb{E}[e^{tX}] = \begin{cases} \sum_{i=0}^n e^{t \cdot x_i} p_X(x_i) \\ \int_s e^{t \cdot x} f_X(x) dx \end{cases}$$

בעזרת פונקציה זו ניתן ליצור מומנטים, שמסייעים ללמוד על המשתנים:

$$\frac{d^n M_X(t)}{dt^n} \Big|_{t=0} = \mathbb{E}[X^n]$$

המומנט הראשון הוא התוחלת והמומנט השני הוא כמעט זהה לשונות (מומנט השני הוא $M''(0) = E[X^2]$, לעומת השונות שהיא $Var(X) = E[X^2] - E[X]^2$).

התפלגויות מיוחדות (בדיד)

ישנן כל מיני התפלגויות מיוחדות, שמופיעות בטבע בכל מיני מקרים ויש להן נוסחאות ידועות.

התפלגות ברנולי: $X \sim Ber(p)$

ניסוי בעל שתי תוצאות אפשריות "הצלחה" או "כישלון". המשתנה המקרי מקבל שני ערכים בלבד – 0 או 1, בהתאם להצלחה וכישלון.

$$P(X = k) = \begin{cases} 1, p \\ 0, q \end{cases}, \mathbb{E}[X] = p, Var[X] = pq = p(1 - p)$$

התפלגות בינומית: $X \sim B(n, p)$

בהתפלגות בינומית חוזרים על אותו ניסוי ברנולי n פעמים באופן בלתי תלוי זה בזה. מגדירים את X להיות מספר ההצלחות שהתקבלו בסה"כ. נסמן ב- p סיכוי להצלחה בניסוי בודד וב- q סיכוי לכישלון בניסוי בודד.

$$P(X = k) = \binom{n}{k} p^k (1 - p)^{n-k}, \mathbb{E}[X] = np, Var[X] = npq$$

צריך לוודא 3 דברים: (1) חוזרים על אותו ניסוי באופן בלתי תלוי. (2) חוזרים על הניסוי n פעמים. (3) X מוגדר כמספר ההצלחות המתקבלות בסה"כ.

התפלגות גיאומטרית: $X \sim G(p)$

חוזרים על ניסוי ברנולי. כאשר X מבטא את מספר הניסויים שבוצעו עד ההצלחה הראשונה. p מסמן את הסתברות ההצלחה בניסוי בודד.

$$P(X = k) = pq^{k-1}, \mathbb{E}[X] = \frac{1}{p}, Var[X] = \frac{q}{p^2}$$

להתפלגות זו יש שתי תכונות נוספות מיוחדות:

$$(1) \text{ "תכונת חוסר זיכרון": } P[X = (n + k) | X > k] = P(n)$$

$$(2) \text{ ההסתברות שיעברו } k \text{ ניסויים ללא הצלחה: } P(X > k) = q^k$$

כמו כן, אם מעוניינים לדעת את מספר הניסיונות הממוצע הנדרש עד להצלחה ראשונה – יש לחשב את התוחלת של המשתנה המקרי X .

התפלגות אחידה: $X \sim U[a, b]$

בהתפלגות זו לכל תוצאה יש את אותה הסתברות. הערכים המתקבלים בהתפלגויות החל מ- a ועד b הינם בקפיצות של (לדוגמה הגרלה של מספר שלם בין 1-100):

$$P(X = k) = \frac{1}{b - a + 1}, k = a, a + 1, \dots, b, \mathbb{E}[X] = \frac{a + b}{2}, Var[X] = \frac{(b - a + 1)^2 - 1}{12}$$

התפלגות פואסונית: $X \sim \text{poi}(\lambda)$

התפלגות זאת מתאפיינת במספר אירועים ליחידת זמן כאשר λ הוא פרמטר המייצג את קצב האירועים ליחידת זמן הנבחרת.

$$P(X = k) = \frac{e^{-\lambda} \lambda^k}{k!}, k = 1 \dots \infty, \mathbb{E}[X] = \text{Var}[X] = \lambda$$

יש לשים לב שכאן ההתפלגות נמדדת ליחידת זמן.

התפלגות היפר גאומטרית: $X \sim H(N, D, n)$

נתונה אוכלוסייה שמכילה N פריטים סה"כ, מתוכה D פריטים "מיוחדים" בעלי תכונה מסוימת. בוחרים מאותה אוכלוסייה n פריטים ללא החזרה. מגדירים את X להיות מספר הפריטים ה-"מיוחדים" שנדגמו.

$$P(X = k) = \frac{\binom{D}{k} \binom{N-D}{n-k}}{\binom{N}{n}}, \mathbb{E}[X] = \frac{nD}{N}, \text{Var}[X] = \frac{nD}{N} \left(1 - \frac{D}{N}\right) \frac{N-n}{N-1}$$

התפלגות בינומית שלילית: $X \sim NB(r, p)$

חוזרים על אותו ניסוי ברנולי בזה אחר זה באופן בלתי תלוי עד אשר מצליחים בפעם ה- r . כלומר, מבצעים את הניסוי עד שמצליחים r פעמים. מגדירים את X להיות מספר החזרות עד שהתקבלו r הצלחות.

$$P(X = k) = \binom{k-1}{r-1} p^r (1-p)^{k-r}, k = r, r+1, \dots, \infty \quad \mathbb{E}[X] = \frac{r}{p}, \text{Var}[X] = \frac{r(1-p)}{p^2}$$

התפלגויות מיוחדות (רציף)

התפלגות מעריכית: $X \sim \text{exp}(\lambda)$

התפלגות רציפה המאפיינת את הזמן עד להתרחשות מאורע מסוים. λ הוא ממוצע מספר האירועים המתרחשים ביחידת זמן (אותו פרמטר מההתפלגות הפואסונית). $\lambda < 0, X \sim \text{exp}(\lambda)$.

גם בהתפלגות זו יש את תכונות חוסר הזיכרון: $P(X > (a+b) | X > a) = P(X > b)$.

התפלגות אחידה: $X \sim U(a, b)$

זו התפלגות שפונקציית הצפיפות שלה קבועה בין a ל- b .

פונקציית הצפיפות: $f(x) = \frac{1}{b-a}, a < x < b$. פונקציית ההתפלגות המצטברת: $F(t) = \frac{t-a}{b-a}$.

$$\mathbb{E}[X] = \frac{a+b}{2}, \text{Var}[X] = \frac{(b-a)^2}{12}$$

התפלגות נורמלית: $X \sim \mathcal{N}(\mu, \sigma^2)$

התפלגות נורמלית היא התפלגות חשובה מאוד כיוון שהיא מופיעה בהמון מקרים. פונקציית הצפיפות של ההתפלגות הנורמלית נראית כמו פעמון, כאשר לעקומה קוראים גם עקומת גאוס. ההתפלגויות הנורמליות נבדלות אחת מהשנייה באמצעות הממוצע וסטיית התקן (-הפרמטרים שמאפיינים את ההתפלגות). התפלגות נורמלית סטנדרטית היא התפלגות נורמלית בעלת תוחלת 0 ושונות 1:

$$X \sim \mathcal{N}(0, 1)$$

עבור תוחלת ושונות μ, σ , פונקציית הצפיפות של משתנה נורמלי הינה:

$$f_X(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

ניתן להשתמש במומנטים בכדי למצוא קשרים בין התפלגויות. למשל עבור שני משתנים המתפלגים נורמלית:

$$X \sim \mathcal{N}(\mu_x, \sigma_x^2), Y \sim \mathcal{N}(\mu_y, \sigma_y^2)$$

המומנטים מקיימים:

$$M_X(t) \cdot M_Y(t) = e^{\mu_x t + \frac{1}{2}\sigma_x^2 t^2} \cdot e^{\mu_y t + \frac{1}{2}\sigma_y^2 t^2} = e^{(\mu_x + \mu_y)t + \frac{1}{2}(\sigma_x^2 + \sigma_y^2)t^2} = M_{X+Y}(t)$$

ולכן ניתן לחשב את ההתפלגות של $X + Y$:

$$X + Y \sim \mathcal{N}(\mu_x + \mu_y, \sigma_x^2 + \sigma_y^2)$$

אי שיוניים

מרקוב

בהינתן $X \geq 0$, התוחלת $\mathbb{E}[X]$, עבור פרמטר $a > 0$ מתקיים:

$$p(X \geq a) \leq \frac{\mathbb{E}[X]}{a}$$

צ'בישב

בהינתן התוחלת $\mathbb{E}[X]$ והשונות $\text{Var}[X]$, עבור פרמטר $a > 0$ מתקיים:

$$p(|X - \mathbb{E}[X]| \geq a) \leq \frac{\text{Var}[X]}{a^2}$$

צ'רנוף

בהינתן התוחלת $\mathbb{E}[X]$, עבור שני פרמטרים $a, t > 0$ מתקיים:

$$p(X \geq a) \leq \frac{\mathbb{E}[e^{tx}]}{e^{ta}} = e^{-ta} M(t)$$

כאשר $M(t)$ היא פונקציה יוצרת מומנטים של X .

ינסן

עבור משתנה מקרי X בעל תוחלת, עבור פונקציה קמורה $g: \mathbb{R} \rightarrow \mathbb{R}$ מתקיים:

$$g(\mathbb{E}[x]) \leq \mathbb{E}[g(x)]$$

התפלגות דו ממדית

$$F_{x,y}(a, b) = P(x \leq a, y \leq b)$$

תכונות:

$$\lim_{a,b \rightarrow \infty} F_{x,y}(a, b) = 1$$

$$\lim_{a \rightarrow -\infty} F_{x,y}(a, b) = \lim_{b \rightarrow -\infty} F_{x,y}(a, b) = 0$$

$$\begin{aligned} P(c < x < a, d < y < b) &= P(x < a, y < b) - P(x < a, y < d) - P(x < c, y < b) + P(x < c, y < d) \\ &= F_{x,y}(a, b) - F_{x,y}(a, d) - F_{x,y}(c, b) + F_{x,y}(c, d) \end{aligned}$$

אם x, y בלתי תלויים אז מתקיים:

$$\forall a, b \quad F_{x,y}(a, b) = F_x(a) \cdot F_y(b)$$

זוג משתנים נקרא דו-ממדי רציף אם קיימת פונקציית צפיפות דו-ממדית $f_{x,y}(s, t)$, כך שמתקיים:

$$P(x, y \in A) = \int f_{x,y}(s, t) ds dt$$

באופן שקול מתקיים:

$$f_{x,y}(s,t) = \frac{\partial^2}{\partial s \partial t} F_{x,y}(s,t) = \frac{\partial^2}{\partial t \partial s} F_{x,y}(s,t)$$

התפלגות שולית:

$$F_x(s) = P(x \leq s) = P(x \leq s, y \leq \infty) = \int_{-\infty}^s \int_{-\infty}^{\infty} f_{x,y}(x,y) dx dy$$

נוסחת ההסתברות השלמה לצפיפות (באופן שקול גם ל- $f_y(t)$):

$$f_x(s) = \frac{d}{ds} F(x_s) = \int_{-\infty}^{\infty} f_{x,y}(s,y) dy$$

כעת ניתן גם לכתוב תנאי שקול למשתנים בלתי תלויים x, y – בלתי תלויים אם מתקיים:

$$\forall x, y \quad f_{x,y}(X, Y) = f_x(X) \cdot f_y(Y)$$

סטטיסטיקה היסקית

אם יודעים את סוג ההתפלגות אבל לא יודעים את מרכיביה, ניתן לאמוד את המרכיבים בעזרת מדגם. המדגם מאפשר לנו להשתמש באומדן עבור מספר מאורעות שווי התפלגות. דוגמא – נניח רוצם למדוד גובה של קבוצה מסוימת – כלל התלמידים בבית ספר מסוים. ידוע שגובה מתפלג נורמלית, אבל לא יודעים כאן את התוחלת והשונות. לשם כך ניתן להשתמש באומדן – פונקציה שמנסה לנתח את המאורעות ומתוך כך להסיק את התוחלת והשונות.

בניתוח נצא מנקודת הנחה שידוע כי הערכים במדגם נלקחים כולם מתוך התפלגות X , השייכת למשפחה של התפלגויות שתלויות בפרמטר אחד או יותר שאינם ידועים. (למשל בדוגמא: $X \sim N(\mu, \sigma^2)$ כאשר μ, σ לא ידועים). בפועל נתונות n דיגמות בלתי תלויות מתוך ההתפלגות: X_1, X_2, \dots, X_n , ורוצים לאמוד את הפרמטרים הלא ידועים (כפונקציה של הערכים שדגמנו).

אומד בלתי מוטה: אומד מוגדר להיות בלתי מוטה אם התוחלת של האומד שווה לפרמטר אותו אנו מנסים לאמוד, כלומר, אם $\mathbb{E}(\hat{\theta}) = \theta$, אז האומד הוא חסר הטיה. במילים אחרות – אומד יהיה חסר הטיה אם התוחלת של המשתנה המקרי המחושב לפי θ שווה ל- θ עבור כל θ .

דוגמאות לאומדים בלתי מוטים:

אומד בלתי מוטה לתוחלת – ממוצע חשבוני:

$$\hat{\theta} = \frac{1}{n} \sum_{i=1}^n x_i$$

$$\mathbb{E}(\hat{\theta}) = \mathbb{E} \left[\frac{1}{n} \sum_{i=1}^n x_i \right] = \frac{1}{n} \sum_{i=1}^n \mathbb{E}[x_i] = \mathbb{E}[x_i] = \theta$$

אומד בלתי מוטה לשונות:

$$\mathbb{E}[s^2] = \frac{1}{n-1} \sum_{i=1}^n (x_i - \bar{x})^2$$

הוכחה:

$$\mathbb{E}[s^2] = \mathbb{E} \left[\frac{1}{n-1} \cdot \sum_i (x_i - \bar{x})^2 \right] = \frac{1}{n-1} \sum_i \mathbb{E}(x_i - \bar{x})^2$$

$$\begin{aligned}
& \frac{1}{n-1} \sum_i \mathbb{E}[(x_i - \mu) - (\bar{x} - \mu)]^2 \\
& \frac{1}{n-1} \sum_i \mathbb{E}[(x_i - \mu)^2 - 2(x_i - \mu)(\bar{x} - \mu) + (\bar{x} - \mu)^2] \\
& \frac{1}{n-1} \sum_i \mathbb{E}[(x_i - \mu)^2] - \mathbb{E}[2(x_i - \mu)(\bar{x} - \mu)] + \mathbb{E}[(\bar{x} - \mu)^2] \\
& \frac{1}{n-1} \sum_i \sigma^2 - 2 \left(\frac{1}{n} \sum_j \mathbb{E}[(x_i - \mu)(x_j - \mu)] \right) + \frac{1}{n^2} \sum_j \sum_k \mathbb{E}[(x_j - \mu)(x_k - \mu)] \\
& \frac{1}{n-1} \sum_i \left[\sigma^2 - \frac{2\sigma^2}{n} + \frac{\sigma^2}{n} \right] \\
& \frac{1}{n-1} \sum_i \left[\frac{(n-1)\sigma^2}{n} \right] = \frac{n-1}{n(n-1)} \sum_i \sigma^2 = \sigma^2 \blacksquare
\end{aligned}$$

אומד נראות מרבית (MLE) – Maximum likelihood estimator

בהינתן סדרת דגימות מתוך התפלגות עם פרמטר לא ידוע, נגדיר את פונקציית הנראות שלהן כמכפלת ההסתברויות של כל הדגימות, או "הנראות של המדגם":

$$L(x_1, x_2 \dots x_n | p(\theta)) = \prod_i P_\theta(x_i)$$

זוהי פונקציה הן של הדגימות והן של הפרמטר.

אם ההתפלגות רציפה מגדירים במקום זאת את פונקציית הנראות להיות מכפלת הצפיפויות:

$$L(x_1, x_2 \dots x_n | p(\theta)) = \prod_i f_\theta(x_i)$$

אומדן הנראות המקסימלית הוא פשוט הערך של הפרמטר שממקסם את פונקציית הנראות. כלומר, $\hat{\theta}$ הוא אומדן נראות מקסימלי עבור θ אם $\hat{\theta} = \arg \max_\theta L(x_1, x_2 \dots x_n | p(\theta))$.

מכוון ש- \log הינה מונוטונית, למקסם את L שקול למקסם את $\log(L)$ (log likelihood), וזה לרוב יותר קל, מכיוון שהמכפלה הופכת לסכום:

$$\log L(x_1, x_2 \dots x_n | p(\theta)) = \sum_{i=1}^n \log f_\theta(x_i)$$

נראה מספר דוגמאות לחישוב ה-MLE:

א. מציאת הפרמטר λ בהתפלגות פואסונית:

$$X \sim \text{poi}(\lambda)$$

שלב א' – נגדיר את אומדן הנראות – $L = (x_1, x_2 \dots x_n | p_\lambda) = \prod_i P_\lambda(x_i)$ בהתפלגות פואסונית מקיימת: $P(X = k) = \frac{e^{-\lambda} \lambda^k}{k!}$, לכן בעצם צריך למצוא מקסימום לביטוי:

$$\prod_i P_\lambda(x_i) = \prod_i \frac{e^{-\lambda} \lambda^{x_i}}{x_i!}$$

מסובך למצוא לזה מקסימום, לכן נוציא לוג:

$$\begin{aligned} \ln\left(\prod_i \frac{e^{-\lambda} \lambda^{x_i}}{x_i!}\right) &= \sum_i \ln\left(\frac{e^{-\lambda} \lambda^{x_i}}{x_i!}\right) \\ &= \sum_i \ln(e^{-\lambda} \lambda^{x_i}) - \ln(x_i!) = \sum_i \ln(e^{-\lambda}) + \ln(\lambda^{x_i}) - \ln(x_i!) \\ &= \sum_i x_i \ln(\lambda) - \lambda - \ln(x_i!) \end{aligned}$$

כעת נגזור:

$$\frac{\partial L}{\partial \lambda} = \sum_i \frac{x_i}{\lambda} - 1 = \sum_i \frac{x_i}{\lambda} - \sum_i 1 = \sum_i \frac{x_i}{\lambda} - n$$

וכשנשווה ל-0 נקבל:

$$\sum_i \frac{x_i}{\lambda} = n$$

נבודד את הפרמטר אותו מנסים לאמוד:

$$\lambda = \frac{\sum_i x_i}{n}$$

וקיבלנו אומד עבור הפרמטר λ , וכאשר נתון סט התוצאות, פשוט נציב אותן, ונמצא מפורשות את הערך של האומד. זה בעצם תהליך מציאת ה- MLE . כעת נבדוק האם האומד הוא מוטה או לא, כאשר נשתמש בעובדה שעבור התפלגות פואסונית $\mathbb{E}(x) = \lambda$:

$$\mathbb{E}(\lambda) = \mathbb{E}\left(\frac{\sum_i x_i}{n}\right) = \frac{1}{n} \sum_i \mathbb{E}[x_i] = \frac{n\lambda}{n} = \lambda$$

קיבלנו שתוחלת האומד שווה לפרמטר, ולכן הוא בלתי מוטה.

ב. התפלגות נורמלית:

$$X \sim (\mu, \sigma^2)$$

פה יש שני פרמטרים לאמוד – התוחלת והשונות. ראשית נגדיר את הנראות:

$$f(X) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2\sigma^2}(x-\mu)^2}$$

לכן הנראות תהיה (נשים לב שהמכפלה תעבור לסכום במעריך של האקספוננט):

$$\prod_i f(x) = \prod_i \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2\sigma^2}(x_i-\mu)^2} = \left(\frac{1}{\sqrt{2\pi\sigma^2}}\right)^n e^{-\frac{1}{2\sigma^2}\sum_i (x_i-\mu)^2}$$

נוציא לוג:

$$\begin{aligned} \ln(L) &= \ln\left(\left(\frac{1}{\sqrt{2\pi\sigma^2}}\right)^n\right) + \ln\left(e^{-\frac{1}{2\sigma^2}\sum_i (x_i-\mu)^2}\right) \\ &= n \ln\left(\frac{1}{\sqrt{2\pi\sigma^2}}\right) - \ln\left(e^{\frac{1}{2\sigma^2}\sum_i (x_i-\mu)^2}\right) \end{aligned}$$

נשים לב שבביטוי הראשון מה שיש בתוך ה- \ln זה בעצם $(\sigma^2)^{-\frac{1}{2}} + (2\pi)^{-\frac{1}{2}}$, ואז המעריך יכול לרדת מחוץ ל- \ln :

$$= -\frac{n}{2} \ln(2\pi) - \frac{n}{2} \ln(\sigma^2) - \frac{1}{2\sigma^2} \sum (x_i - \mu)^2$$

כעת בשביל לאמוד את התוחלת יש לגזור לפי μ , וכדי לאמוד את השונות יש לגזור לפי σ^2 :

$$\frac{\partial L}{\partial \mu} = -\frac{(-2)}{2\sigma^2} \sum_i (x_i - \mu) = \frac{1}{\sigma^2} \sum_i (x_i - \mu)$$

וכשנשווה ל-0 נקבל:

$$\hat{\mu} = \frac{\sum_i x_i}{n}$$

ניתן להבחין כי עבור התוחלת האומד הוא בעצם הממוצע של המדגם. אפשר לבצע תהליך דומה על השונות, ומתקבל הביטוי:

$$\hat{\sigma}^2 = \frac{\sum_i (x_i - \hat{\mu})^2}{n}$$

1. References

Intro:

<https://www.analytics.org.il/2019/12/ai-vs-deep-learning-vs-machine-learning/>

2. Machine Learning

2.1 Supervised Learning Algorithms

2.1.1 Support Vector Machines (SVM)

Support Vector Machine (SVM) הינו מודל למידה מונחית המשמש לניתוח נתונים לצורך סיווג, חיזוי ורגרסיה. המודל מקבל אוסף של דוגמאות מתויגות במרחב n -ממדי, ומנסה למצוא מישור המפריד בצורה טובה כמה שניתן בין דוגמאות האימון השייכות לקטגוריות השונות.

המסווג הנוצר באמצעות מודל SVM הינו לינארי, כאשר חלוקת הדוגמאות במרחב הווקטורי נעשית באופן כזה ששיוצר מרווח גדול ככל האפשר בין המישור המפריד לבין הנקודות הממוקמות הכי קרוב אליו. מרווח זה מכונה שוליים (margin), כאשר בצד האחד של השוליים נמצאות דוגמאות עם label אחד, ובצד השני נמצאות הדוגמאות עם ה-label השני. את המישור המפריד ניתן לייצג באמצעות אוסף הנקודות \vec{x} המקיימות $\vec{w} \cdot \vec{x} - b = 0$, כאשר \vec{w} הוא וקטור נורמלי של המישור.

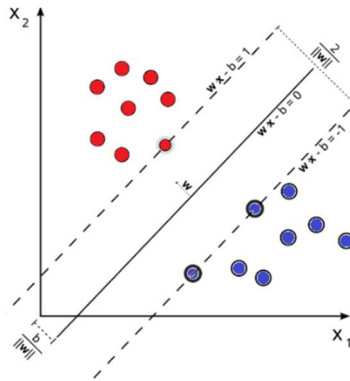
ננסח את האלגוריתם באופן פורמלי: נתון אוסף של n נקודות $(x_1, y_1) \dots (x_n, y_n)$, כאשר $y_i \in \{-1, 1\}$ מייצג את התיג המתאים לדוגמא i , ו- $x_i \in \mathbb{R}^d$ הוא וקטור המאפיינים המתארים את דוגמא i . מודל ה-SVM מייצר מישור המפריד את המרחב לשני מרחבים שכל אחד מהם אמור להכיל בעיקר דוגמאות מסוג תיג אחד. בנוסף, המודל מייצר שני מישורים מקבילים לו, אחד מכל צד, במרחק זהה וגדול ככל האפשר:

$$w^* = \operatorname{argmin}_w \left(\frac{1}{2} \|w\|^2 \right), s.t \forall i y_i (w \cdot x_i) \geq 1$$

כלומר, רוצים למצוא את וקטור המשקולות w^* המייצר שוליים $\text{margin} = \frac{1}{2} \|w\|^2$, כך שהדוגמאות מתויגות נכון $(y_i (w \cdot x_i) \geq 0)$ ואינן בתוך השוליים (לא מתקיים: $0 < y_i (w \cdot x_i) < \frac{1}{2} \|w\|^2$). ישנם מספר גישות למציאת המפריד, ונפרט על כמה מהן.

Hard-Margin (hard SVM)

במצב הפשוט ביותר, המשוואה עבור כל אחד מצדדיו של המפריד הינה פונקציה לינארית של המאפיינים וכל הדוגמאות אשר סווגו נכונה. מצב זה מכונה "הפרדה קשיחה" בו האלגוריתם מוצא את המישור עם השול הרחב ביותר האפשרי, ולא מאפשר לדוגמאות להיות בין הווקטורים התומכים. זוהי למעשה הפרדה מושלמת, והווקטורים התומכים הם למעשה הנקודות בקצוות השוליים, כפי שניתן לראות באיור:



איור 2.1 סיווג באמצעות אלגוריתם SVM עם מפריד בעל השוליים הרחבים ביותר. הקו האמצעי מייצג את המפריד, הקווים המקווקיים מייצגים את מישורי השוליים. דוגמאות האימון המתלכדות עם מישורי השוליים נקראות וקטורים תומכים (support vectors), ומכאן נגזר שם האלגוריתם.

את המישורים בקצוות השוליים ניתן לייצג באמצעות $\vec{w} \cdot \vec{x} - b = 1$ or $\vec{w} \cdot \vec{x} - b = -1$. גאומטרית, המרחק בין שני המישורים הוא $\frac{2}{\|w\|}$, ולכן על מנת למקסם את המרחק הזה, יש להביא למינימום את $\|w\|$. על מנת שדוגמאות האימון לא יכללו בשוליים המפרידים, יש להוסיף אילוץ לכל דוגמא i , באופן הבא:

$$y_i \cdot (\vec{w} \cdot \vec{x}_i - b) \geq 1$$

אילוץ זה מחייב שכל דוגמא תימצא בצד הנכון של המפריד. לכן, במקרה זה יש לקיים את הדרישה הבאה:

$$\min_{w,b} \|w\|^2$$

$$s.t. y_i (\vec{w} \cdot \vec{x}_i - b) \geq 1 \quad \forall i = 1 \dots n$$

Soft-Margin (soft SVM)

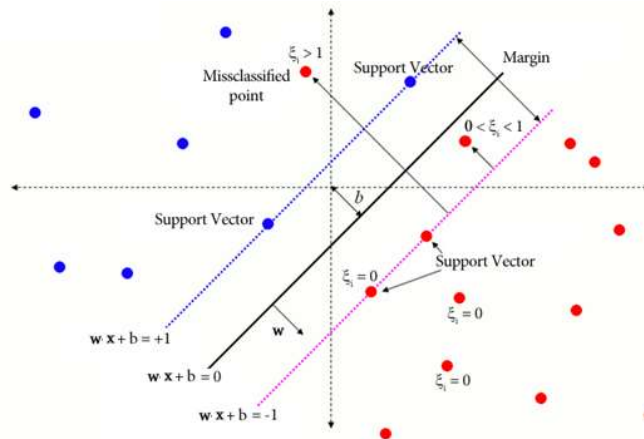
הפרדה מושלמת באמצעות מישור לינארי לעיתים קרובות איננה אפשרית, ולכן ניתן להרחיב את המודל כך שיאפשר לנקודות מסוימות לא להיות ב"צד" המתאים להן. הרחבה זו, היוצרת "הפרדה רכה", מאפשרת לטפל בבעיות שבהן אין הפרדה לינארית בין הקבוצות, כמו למשל שיש נקודות חריגות. משמעות ההרחבה היא שכל וקטור מפר לפחות אחד מהאילוץ, אך עם זאת, נרצה להגיע למצב בו האילוץ מופרים "כמה שפחות". הפרדה רכה יוצרת מצב בו יש trade-off בין רוחב השול לבין השגיאות ומציאת המשקלים האופטימליים של המסווג. בגרסת זו יש לרשום באופן מעט שונה את בעיית האופטימיזציה, כאשר מתווסף משתנה המתייחס לנקודות שאינן נמצאות בסיווג המתאים להן לפי המפריד:

$$\min \|w\|^2 + C \sum_{i=1}^n \xi_i$$

$$s.t. y_i (\vec{w} \cdot \vec{x}_i - b) \geq 1 - \xi_i \quad \forall i = 1 \dots n, \xi_i \geq 0$$

לשם קבלת אינטואיציה, נשים לב לתפקיד המשתנים:

אם $\xi_i = 0$, מתקיים התנאי שנדרש בהפרדה קשיחה, כלומר הנקודה x_i גם נמצאת בצד הנכון של המפריד וגם מתקיימת הדרישה לשמירה על השוליים. אם $0 < \xi_i < 1$ אז הנקודה x_i נמצאת בצד הנכון של המפריד המסווג, אבל המסווג קרוב אליה, כך שהנקודה נמצאת בתוך השוליים. ערך C גבוה פירושו העדפת הסיווג הנכון על פני שוליים רחבים, ואילו C נמוך מעדיף הכללה (שוליים רחבים), גם במחיר שדוגמאות האימון הספציפיות אינן מסווגות נכון.



איור 2.2 סיווג באמצעות אלגוריתם SVM עם הפרדה רכה. המשתנה ξ_i שווה לאפס אם הנקודה ממוקמת בצד הנכון של המפריד. וגדול מאפס כאשר הנקודות נמצאות בצד הלא נכון של המפריד.

Non-linear Separation

מסווגים לינאריים מוגבלים ביכולת ההכללה שלהם בגלל הפשטות שלהם. לכן, כאשר לא ניתן להפריד אוסף דוגמאות באמצעות מפריד לינארי, משתמשים ב"הפרדה א-לינארית". גישה זו מאפשרת להשתמש ב-SVM לסיווג לא לינארי, על ידי טרנספורמציה לא לינארית, כמו למשל "תעלול הגרעין" (Kernel Trick). בגישה זו מבצעים מיפוי לדאטה למרחב אחר, בו ניתן למצוא עבורו הפרדה לינארית, וממילא יהיה אפשר להשתמש באלגוריתם SVM. כך למשל, קיימת אפשרות ליצור מאפיינים חדשים על ידי העלאת ערכי המאפיינים הקיימים בחזקה מסוימת, הכפלתם בפונקציות טריגונומטריות וכו'.

באופן פורמלי, נחפש פונקציית מיפוי להעתקת מרחב $\chi \rightarrow F$ כן שבמרחב F ניתן יהיה להפריד את הנתונים $\{\psi(x_i), y_i\}_{i=1}^N$ באמצעות מסווג לינארי. לשם כך, משתמשים בטריק קרנל שמקבל כקלט וקטורים במרחב המקורי ומחזיר את המכפלה הפנימית (dot product) של הווקטורים במרחב החדש (נקרא גם מרחב התכונות – feature space):

$$K(\vec{x}_i, \vec{x}_j) = \psi(\vec{x}_i)^T \psi(\vec{x}_j)$$

דוגמאות של פונקציות קרנל נפוצות:

קרנל לינארי:

$$K(\vec{x}_i, \vec{x}_j) = \vec{x}_i \cdot \vec{x}_j$$

זוהי הפונקציה הכי פשוטה, המוגדרת על ידי מכפלה פנימית של הווקטורים. במקרה זה מרחב התכונות ומרחב הקלט זהים ונחזור לפתרון בעזרת SVM לינארי:

קרנל פולינומי:

$$K(\vec{x}_i, \vec{x}_j) = (\vec{x}_i \cdot \vec{x}_j + c)^d$$

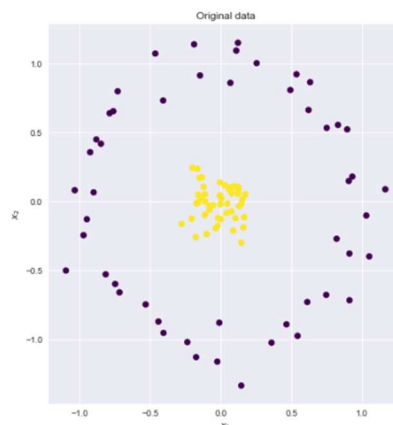
העתקה מהמרחב המקורי למרחב שמהווה פולינום ממעלה $d \geq 2$. $c \geq 0$ הוא פרמטר חופשי המשפיע על היחס בין סדר גבוה לעומת סדר נמוך בפולינום. כאשר $c = 0$, הקרנל נקרא הומוגני.

קרנל גאוסיאני:

$$K(\vec{x}_i, \vec{x}_j) = \exp(-\gamma(\vec{x}_i - \vec{x}_j)^2), \gamma > 0$$

הפרמטר γ ממלא תפקיד חשוב, ויש לבחור אותו בהתאם לבעיה העומדת בפנינו. אם הערך שלו קטן מאוד, האקספוננט ינהג כמעט באופן ליניארי וההטלה למרחב אחר מממד גבוה יותר יתחיל לאבד מכוחו הלא ליניארי. מצד שני, אם נעריך אותו יתר על המידה, הפונקציה לא תהיה סדירה וגבול ההחלטה יהיה רגיש מאוד לרעש בנתוני האימון.

המהות של טריק קרנל היא שניתן לבצע את ההעתקה גם מבלי לדעת מהי הפונקציה ψ , אלא הידיעה של K מספיקה. לצורך קבלת אינטואיציה והמחשה נביא דוגמא. נתון מערך הנתונים הבא:



ניתן לראות שלא ניתן להפריד בין הנקודות הצהובות לסגולות על ידי מישור הפרדה לינארי. לכן נחפש מרחב אחר, מאותו ממד או בעל ממד גבוה יותר, בו ניתן יהיה להפריד בין נקודות אלה באופן לינארי. לצורך כך נבצע את הפעולות הבאות:

- א. נמפה את התכונות המקוריות למרחב הגבוה יותר (מיפוי תכונות).
- ב. נבצע SVM לינארי במרחב החדש.
- ג. נמצא את קבוצת המשקולות התואמות את מישור גבול ההחלטה.
- ד. נמפה את מישור המפריד בחזרה למרחב הדו-ממדי המקורי כדי לקבל גבול החלטה לא לינארי.

ישנם הרבה מרחבים מממדים גבוהים יותר בהם נקודות אלה ניתנות להפרדה לינארית. נציג דוגמא אחת:

$$x_1, x_2 \mapsto z_1, z_2, z_3$$

$$z_1 = \sqrt{2}x_1x_2 \quad z_2 = x_1^2 \quad z_3 = x_2^2$$

למעשה נעזרנו בטריק קרנל. כאמור, בהינתן שקיימת פונקציה שממפה $\varphi: \mathbb{R}^n \rightarrow \mathbb{R}^m$ את הווקטורים ממרחב \mathbb{R}^n למרחב תכונות כלשהו \mathbb{R}^m , אז המכפלה הפנימית של x_1 ו- x_2 במרחב הזה היא $\varphi(x_1)^T \varphi(x_2)$. קרנל היא פונקציה K השייכת למכפלה פנימית זו, כלומר $K(x_1, x_2) = \varphi(x_1)^T \varphi(x_2)$. אם נוכל למצוא פונקציית קרנל המקבילה למפת התכונות שלעיל, נוכל להשתמש בפונקציה יחד עם SVM לינארי וכך לבצע את החישובים בעיליות.

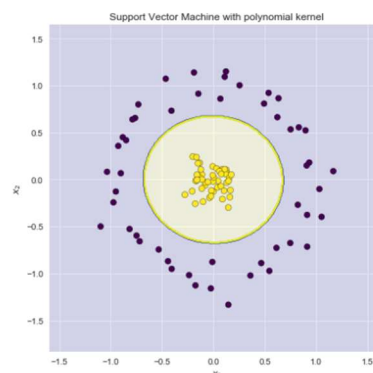
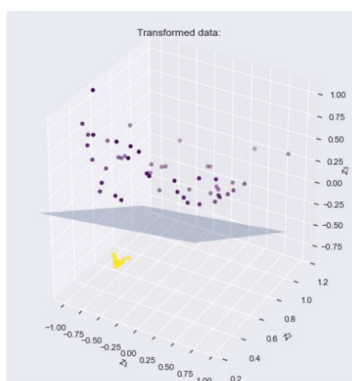
מתברר שמרחב התכונות שלעיל תואם את השימוש בקרנל פולינומי ידוע: $K(x, x') = (x^T x')^d$. נבחר $d = 2$, נסמן $x = (x_1, x_2)^T$ ונקבל:

$$K\left(\begin{pmatrix} x_1 \\ x_2 \end{pmatrix}, \begin{pmatrix} x_1' \\ x_2' \end{pmatrix}\right) = (x_1x_2' + x_2x_1')^2 =$$

$$2x_1x_1'x_2x_2' + (x_1x_1')^2 + (x_2x_2')^2 = (\sqrt{2}x_1x_2x_1'x_2') \begin{pmatrix} \sqrt{2}x_1'x_2' \\ x_1'^2 \\ x_2'^2 \end{pmatrix}$$

$$K\left(\begin{pmatrix} x_1 \\ x_2 \end{pmatrix}, \begin{pmatrix} x_1' \\ x_2' \end{pmatrix}\right) = \varphi(x)^T \varphi(x')$$

$$\varphi\left(\begin{pmatrix} x_1 \\ x_2 \end{pmatrix}\right) = \begin{pmatrix} \sqrt{2}x_1x_2 \\ x_1^2 \\ x_2^2 \end{pmatrix}$$



איור 2.3 שימוש ב-SVM לצורך הפרדה לאחר ביצוע Kernel trick. המעגל באיור הימני ממופה למישור הפרדה לינארי במרחב מממד גבוה יותר, כפי שניתן לראות באיור השמאלי. ניתן לראות שאחרי המיפוי שנעשה בעזרת kernel trick, הנקודות אכן מופרדות בצורה לינארית.

2.1.2 Naive Bayes

סיווג בייסיאני הוא מודל המשתמש בחוק בייס על מנת לסווג אובייקט $x \in \mathbb{R}^n$ בעל n מאפיינים לאחת מ- K קטגוריות אפשריות. יחד עם השימוש בחוק בייס, המודל מניח "נאיביות" – בהינתן סיווג של אובייקט מסוים, אין תלות בין המאפיינים השונים שלו.

נניח שיש מודל המקבל וקטור מאפיינים בינאריים {כאב ראש, משתעל, חום גבוה}, ומסווג האם אדם בעל תכונות אלה חולה בשפעת או לא. באופן כללי ניתן לומר שיש תלות בין שיעול לבין חום גבוה, כלומר העובדה שיש לאדם חום מעלה את ההסתברות שהוא גם משתעל. למרות זאת, ניתן להניח באופן "נאיבי" שאם כבר יודעים שאדם חולה בשפעת, אז כבר אין יותר תלות בין היותו משתעל להיותו בעל חום. באופן פורמלי, אמנם סביר להניח שמתקיים $p(\text{חום}|\text{משתעל}) > p(\text{חום}|\text{משתעל})$, אך ניתן להניח נאיביות ולקבל: $p(\text{שפעת}|\text{משתעל}) = p(\text{חום}|\text{משתעל})$.

באופן כללי סיווג בייסיאני נאיבי מניח שבהינתן הסיווג של אובייקט מסוים, המאפיינים שלו בלתי תלויים. הנחה זו מובן לא תמיד מדויקת, וממילא גם ערכי ההסתברויות הנובעים ממנה ומשמשים לסיווג אינם מדויקים, אך ההנחה

מקלה מאוד על חישוב ההסתברויות של הסיווג הבייסיאני, ובמקרים רבים תחת ההנחה זו התקבלו תוצאות סיווג. הסיבה להצלחת המודל נעוצה בכך שבבעיית סיווג העיקר הוא למצוא את הסיווג הסביר ביותר לאובייקט (שפעת או לא-שפעת לנבדק בדוגמא), ולא דווקא לקבל הסתברות מדויקת לכל סיווג. במקרים רבים למרות שההסתברות הנובעת מההנחה הנאיבית אינה מדויקת עבור שני סיווגים אפשריים, היא בכל זאת שומרת על סדר ההסתברות שלהם.

נתבונן בוקטור מאפיינים $x \in \mathbb{R}^n = (x_1, \dots, x_n)$, היכול להיות שייך לאחת מ-K קטגוריות $y = (y_1, \dots, y_k)$. ההתפלגות הפריורות $p(y_k)$ ידועה, ובנוסף ידועות ההתפלגויות המותנות של המאפיינים בהנתן הסיווג – $p(x_i|y_k)$. בעזרת הנתונים האלה רוצים לסווג את x לאחת מהקטגוריות, כלומר למצוא את y_k שעבורו הביטוי $p(y_k|x)$ הוא מקסימלי. באופן פורמלי ניתן לנסח זאת כך:

$$y = \arg \max_k p(y_k|x), k = 1 \dots K$$

בשביל למצוא את y_k האופטימלי ניתן להיעזר בחוק בייס:

$$p(y_k|x) = \frac{p(y_k, x)}{p(x)}$$

המכנה לא תלוי ב- k , ולכן מספיק למצוא את y_k שעבור המונה מקסימלי. לפי כלל השרשרת מתקיים:

$$\begin{aligned} p(y_k, x) &= p(y_k, x_1, \dots, x_n) = p(x_1|y_k, x_2, \dots, x_n) \cdot p(y_k, x_2, \dots, x_n) \\ &= p(x_1|y_k, x_2, \dots, x_n) \cdot p(x_2|y_k, x_3, \dots, x_n) \cdot p(y_k, x_3, \dots, x_n) \\ &= \dots = p(x_1|y_k, x_2, \dots, x_n) \cdot p(x_2|y_k, x_3, \dots, x_n) \cdots p(x_{n-1}|y_k, x_n) \cdot p(x_n|y_k) p(y_k) \end{aligned}$$

כעת נשתמש בהנחת הנאיביות, לפי בהינתן הסיווג y_k , אין תלות בין המאפיינים. לפי הנחה זו נוכל לפשט את הביטוי:

$$\begin{aligned} &= p(x_1|y_k) \cdot p(x_2|y_k) \cdots p(x_{n-1}|y_k) \cdot p(x_n|y_k) p(y_k) \\ &= p(y_k) \prod_{i=1}^n p(x_i|y_k) \end{aligned}$$

בביטוי זה כל האיברים ידועים, ולכן כל שנותר זה רק להציב את הנתונים ולקבל את y_k עבורו ביטוי זה הכי גדול:

$$y = \arg \max_k p(y_k) \prod_{i=1}^n p(x_i|y_k)$$

בדוגמא שהובאה לעיל, המאפיינים קיבלו ערכים ביניים, ולכן היה ניתן לחשב את ההסתברות המותנית של כל מאפיין $p(x_i|y_k)$ על ידי ספירת כמות הפעמים שמופיע כל מאפיין באוכלוסייה הנדגמת ולחלק בגודל המדגם. עבור ערכים רציפים (כמו למשל מחיר מניה, גובה של אדם וכדו'), אין אפשרות לחשב כך את ההסתברות המותנית. במקרים כאלה יש להניח התפלגות מסוימת עבור המדגם, ולחשב את הפרמטרים של ההתפלגות שיטות שונות (למשל בעזרת נראות מרבית – MLE). עבור מדגם המתפלג נורמלית, ההסתברות המותנית היא גאוסיאן:

$$p(x_i|y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} e^{-\frac{(x_i-\mu_y)^2}{2\sigma_y^2}}$$

כאשר μ_y, σ_y^2 הם הפרמטרים של ההתפלגות, וכאמור הם משוערכים בעזרת MLE או שיטת שערך אחרת. אם ההתפלגות היא לא נורמלית, ניתן להשתמש באלגוריתם [Kernel density estimation](#) עבור שערך ההתפלגות. גישה אחרת להתמודדות עם מאפיינים היכולים לקבל ערכים רציפים היא לבצע דיסקרטיזציה לערכים אותם המאפיינים יכולים לקבל.

במקרה [המולטינומי](#), בו ההתפלגות היא רב ממדית ומציינת תוצאה של סדרה בלתי תלויה, יש לחשב את הנראות באופן המתאים להתפלגות מולטינומית. בכדי להבין את החישוב נביא קודם בדוגמא – נניח רוצים לבנות מודל סיווג בייסיאני המזהה הודעות ספאם. נתונות 12 הודעות, מתוכן 8 אמיתיות ו-4 ספאם. כעת נניח וכל ההודעות מורכבות מאוסף של ארבע מילים, בהתפלגות הבאה:

Real (R) – {Dear, Friend, Lunch, Money} = {8, 5, 3, 1}.

Spam (S) – {Dear, Friend, Lunch, Money} = {2, 1, 0, 4}.

נחשב את הנראות – ההסתברות של כל מילה בהינתן הסיווג:

$$p(\text{Dear}|R) = \frac{8}{17}, p(\text{Friend}|R) = \frac{5}{17}, p(\text{Lunch}|R) = \frac{3}{17}, p(\text{Money}|R) = \frac{1}{17}$$

$$p(\text{Dear}|S) = \frac{2}{7}, p(\text{Friend}|S) = \frac{1}{7}, p(\text{Lunch}|S) = 0, p(\text{Money}|S) = \frac{4}{7}$$

כעת נבחן מה ההסתברות שהצירוף "Dear friend" הוא מהודעה אמיתית (הצירוף הוא למעשה התפלגות מולטינומית, כיוון שהוא מכיל שתי מילים שאין בין ההסתברויות שלהן קשר ישיר):

$$p(\text{Dear friend is R}) = p(R) \cdot p(\text{Dear}|R) \cdot p(\text{Friend}|R) = 0.67 \cdot 0.47 \cdot 0.29 = 0.09$$

$$p(\text{Dear friend is S}) = p(S) \cdot p(\text{Dear}|S) \cdot p(\text{Friend}|S) = 0.33 \cdot 0.29 \cdot 0.14 = 0.01$$

ממספרים אלה ניתן להסיק שהצירוף "Dear friend" אינו ספאם.

באופן כללי, עבור וקטור מאפיינים $x \in \mathbb{R}^n = (x_1, \dots, x_n)$, הנראות מחושבת באופן הבא:

$$p(x|y_k) = \frac{(\sum_i x_i)!}{\prod_i x_i!} \prod_i p(y_{ki})^{x_i}$$

על הציר הלוגריתמי, בעזרת נוסחה זו ניתן לבנות מסווג לינארי:

$$p(y_k|x) = \frac{p(y_k, x)}{p(x)} \propto p(y_k) \cdot \prod_i p(y_{ki})^{x_i}$$

$$\rightarrow \log p(y_k|x) \propto \log p(y_k) \cdot \prod_i p(y_{ki})^{x_i} = \log p(y_k) + \sum_i x_i \cdot \log p(y_{ki}) \equiv b + w^T x$$

החיסרון בשימוש במסווג בייסיאני נאיבי בבעיות מולטינומיות נעוץ בכך שיש הרבה צירופים שלא מופיעים יחד בסט האימון, ולכן הנראות שלהם תמיד תהיה 0, מה שפוגם באמינות התוצאות.

מקרה דומה להתפלגות מולטינומית הוא מקרה בו המאפיינים הם משתני ברנולי, המקבלים ערכים בינאריים. במקרה זה הנראות הינה:

$$p(x|y_k) = \prod_{i=1}^n p_i^{x_i} (1 - p_i)^{1-x_i}$$

עבור דאטה לא מאוזן, ניתן להשתמש באלגוריתם שנקרא complement naive Bayes (CNB). לפי אלגוריתם זה, במקום לקחת את $\arg \max_k p(y_k) \prod_{i=1}^n p(x_i|y_k)$, לוקחים את המינימום של הפונקציה ההופכית:

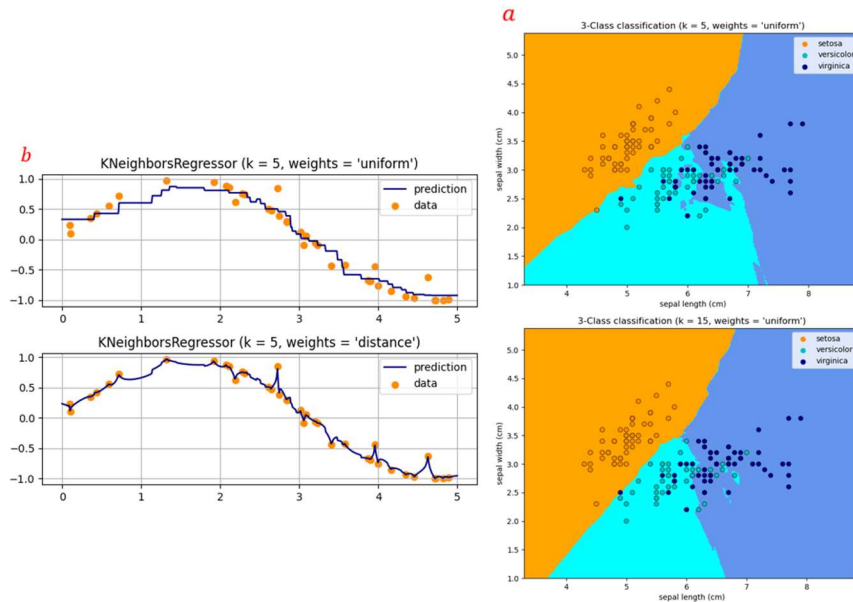
$$\arg \min_k p(y_k) \prod_{i=1}^n \frac{1}{p(x_i|y_k)}$$

שימוש באלגוריתם זה הוכח כיעיל במקרים בהם הדאטה אינו מאוזן והביצועים של מסווגים בייסיאנים אחרים (גאוסיאני או מולטינומי) היה לא מספיק טוב.

2.1.3 K-Nearest Neighbors (K-NN)

אלגוריתם השכן הקרוב הינו אלגוריתם של למידה מונחית, בו נתונות מספר דוגמאות ובנוסף ידוע ה-label של כל אחת מהן. אלגוריתם זה מתאים הן לבעיות סיווג (שיוך נקודה חדשה למחלקה מסוימת) והן לבעיות רגרסיה (נתינת ערך מאפיין לנקודה חדשה). האלגוריתם הינו מודל חסר פרמטרים, והוא מבצע סיווג לנתונים בעזרת הכרעת הרוב. עבור כל נקודה במדגם, המודל בוחן את ה-labels של K הנקודות הקרובות אליו ביותר, ומסווג את הנקודה לפי ה-label שקיבל את מרבית הקולות. מספר הנקודות הקרובות, K, הוא היפר-פרמטר שנקבע מראש.

אלגוריתם השכן הקרוב הוא אחד המודל הנפוצים והפשוטים ביותר בלמידת מכונה, וכאמור בנוסף לסיווג הוא מתאים גם לבעיות רגרסיה. המודל יפעל בצורה דומה בשני המקרים, כאשר ברגרסיה יתבצע שקלול של ממוצע בין השכנים הקרובים, ולא הכרעת הרוב, כלומר, התוצאה לא תהיה סיווג ל-label מסוים לפי הערך הנפוץ ביותר בקרב K השכנים הקרובים, אלא חישוב ממוצע של כל ה-labels השכנים. התוצאה המתקבלת היא ערך רציף, המייצג את הערכים בסביבת התצפית. ניתן להתחשב במרחק של כל שכן מהתצפית בצורה שווה (uniform), וניתן לתת משקל שונה לכל שכן בהתאם למרחק שלו מהנקודה אותה רוצים לחשב, כך שכלל ששכן מסוים קרוב יותר לנקודה אותה רוצים לחשב כך הוא יותר ישיפיע עליה, ביחס של הופכי המרחק בין השכן לבין הנקודה (distance).



איור 2.4 (a) סיווג בעזרת אלגוריתם K-NN: מסווגים את המרחב לאזורים בהתאם ל- K השכנים הקרובים ביותר, כך שאם תבוא נקודה חדשה היא תהיה מסווגת בהתאם לצבע של האזור שלה, הנקבע כאמור לפי השכנים הקרובים ביותר. ניתן לראות שיש הבדל בין ערכי K שונים, וככל ש- K יותר גבוה ככה האזורים יותר חלקים ויש פחות מובלעות. (b) רגרסיה בעזרת אלגוריתם K-NN: קביעת ערך ה- γ בהתאם ל- K השכנים הקרובים ביותר. ניתן לתת משקלים שווים לכל השכנים, או לתת משקל ביחס למרחק של כל שכן מהנקודה אותה רוצים לחשב.

לעיתים נאמר על המודל שהוא "עצלן". הסיבה לכך היא שבשלב האימון לא מתבצע תהליך משמעותי, מלבד השמה של המשתנים וה-labels כאובייקטים של המחלקה, כלומר כל נקודה משויכת למחלקה מסוימת. עקב כך, כל מדגם האימון (או רובו) נדרש לצורך התחזית, מה שעשוי להפוך את המודל לאיטי כאשר יש הרבה דאטה. למרות זאת, המודל נחשב לאחד המודלים הקלאסיים הבולטים, בזכות היתרונות שלו. הוא פשוט וקל לפירוש, עובד היטב עם מספר רב של מחלקות, ומתאים לבעיות רגרסיה וסיווג. בנוסף הוא נחשב אמין במיוחד, כיוון שהוא לא מניח הנחות לגבי התפלגות הנתונים (כמו רגרסיה לינארית למשל).

מנגד, יש לו מספר חסרונות. עקב העובדה שהוא דורש את כל נתוני האימון בשביל התחזית, הוא עשוי להיות איטי כאשר מדובר על דאטה עשיר. מסיבה זו הוא גם אינו יעיל מבחינת זיכרון. מכיוון שהמודל דורש את כל נתוני האימון לצורך המבחן, כושר ההכללה שלו עשוי להיפגם (Generalization). ניקח לדוגמא מורה של כיתה בבית ספר, המנסה לסווג את התלמידים למספר קבוצות. אם יעשה זאת לפי צבע שיער ועיניים, לדוגמא, סביר להניח שלא יתקשה בכך; אם לעומת זאת הוא ינסה לסווג לפי צבע שיער, עיניים, חולצה, מכנסיים, נעליים, וכו' – סביר שיתקל בקושי. במצב כזה, כל תלמיד רחוק מרעהו באופן שווה כיוון שאין שני תלמידים שזהים לחלוטין בכל הפרמטרים, מה שמקשה על חישוב המרחק. בעיה זו מכונה קללת הממדיות (Curse of dimensionality), ולכן מומלץ להיעזר באמצעים להורדת הממד (Dimensionality reduction).

קושי נוסף הקיים במודל הוא הצורך בבחירת ה- K הנכון, מטלה שעשויה להיות לא קלה לעיתים. בכל מימוש של אלגוריתם השכן הקרוב, K הינו היפר-פרמטר שצריך להיקבע מראש. היפר פרמטר זה קובע את מספר הנקודות אשר האלגוריתם יתחשב בהן בעת בחירת סיווג התצפית. בחירת היפר-פרמטר קטן מידי, לדוגמא $K = 1$, יכולה לגרום למצב בו המודל מותאם יתר על המידה לנתוני האימון, מה שמוביל לדיוק גבוה בנתוני האימון, ודיוק נמוך בנתוני המבחן. מן העבר השני, כאשר K גבוה מידי, למשל $K = 100$, נוצר המצב ההפוך – מודל שמתחשב יותר מדי בדאטה ולא מצליח למצוא הכללה נכונה לסיווג. מומלץ לבחור K אי-זוגי בגלל אופן הפעולה של האלגוריתם – הכרעת

הרוב. כאשר בוחרים K זוגי, עלולים להיתקל במצב של שוויון אשר עשוי להוביל לתוצאה מוטעית, ולכן כדי להימנע מתיקו כדאי לבחור K אי זוגי.

כמו אלגוריתמים רבים מבוססי מרחק, אלגוריתם השכן הקרוב רגיש לערכים קיצוניים (Outliers) ושימוש באלגוריתם ללא טיפול בערכים קיצוניים עשוי להוביל לתוצאות מוטות. מלבד זאת, חשוב לנרמל את הנתונים לפי שימוש במודל. הסיבה לכך היא שהאלגוריתם מבוסס מרחק; במצב זה, ייתכנו מרחקים בין תצפיות אשר עשויים להשפיע על החלטת המודל, למרות שמרחקים אלו הם חסרי משמעות לצורך הסיווג. דוגמא לכך היא משתנה שעושה שימוש ביחידות מידה שונות (מיילס/קילומטרים). ההחלטה האם להשתמש בקילומטרים או במיילים עלולה להטות את תוצאת המודל, למרות שבפועל לא השתנה דבר.

השיטה הנפוצה ביותר למדידת מרחק בין משתנים רציפים היא מרחק אוקלידי – עבור שתי נקודות במישור, המרחק ביניהם יחושב לפי הנוסחה: $d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$. במידה ומדובר במשתנים בדידים, כגון טקסט, ניתן להשתמש במטריקות אחרות כגון מרחק המינג, המודד את מספר השינויים הדרושים בכדי להפוך מחרוזת אחת למחרוזת שנייה, ובכך למדוד את הדמיון ביניהן.

לפני שימוש באלגוריתם השכן הקרוב, יש הכרח לוודא שהמחלקות מאוזנות. במידה ומספר דוגמאות האימון באחת המחלקות גבוה מאשר בשאר המחלקות, האלגוריתם ייטה לסווג למחלקה זאת. הסיבה לכך היא שבשל מספרן הגדול, מחלקה זו צפויה להיות נפוצה הרבה יותר בקרב K השכנים של כל תצפית. הדבר עשוי להביא לתוצאות מוטות, ולכן יש לוודא מראש שאכן יש איזון בין המחלקות השונות.

2.1.4 Quadratic\Linear Discriminant Analysis (QDA\LDA)

Quadratic Discriminant Analysis הינו מודל נוסף לסיווג של דאטה לקבוצות שונות, המניח שבהינתן סיווג של אובייקט מסוים – מתקבלת התפלגות נורמלית, כלומר בהינתן $\{y_k, k \in \{1, \dots, K\}\}$ מתקיים:

$$x|y_k \sim N(\mu_k, \Sigma_k)$$

ובאופן מפורש, עבור $x \in \mathbb{R}^{n \times d}$ הפילוג המותנה הוא:

$$p(x|y = k; \mu_k, \Sigma_k) = \frac{1}{\sqrt{(2\pi)^d |\Sigma_k|}} \exp\left(-\frac{1}{2}(x - \mu_k)^T \Sigma_k^{-1} (x - \mu_k)\right)$$

בעזרת הנחה זו, ניתן למצוא מסווג אופטימלי עבור $y = \arg \max_k p(y_k|x)$. לפי חוק בייס:

$$p(y_k|x) = \frac{p(x|y = k)p(y)}{p(x)}$$

המכנה קבוע ביחס ל- y_k , ואת $p(y)$ ניתן לשערך בקלות על פי השכיחות של כל label במדגם – $p(y = k) = \frac{1_{y=k}}{n}$. את הביטוי הנוסף במונה – $p(x|y = k)$ – שכאמור מתפלג נורמלית, ניתן לשערך בעזרת הנראות מרבית (MLE). נסמן את הפרמטרים של המודל ב: $\theta = \{\mu_1, \Sigma_1, \dots, \mu_K, \Sigma_K\}$, ונקבל:

$$\theta_{MLE} = \arg \max_{\theta} p(x|y) = \arg \max_{\theta} \log p(x|y; \theta)$$

$$= \arg \max_{\theta} \log \sum_{i=1}^n p(x_i|y_i; \theta)$$

ניתן לפרק את הסכום לפי ה-label של כל דגימה:

$$= \arg \max_{\theta} \log \sum_{i \in y_i=1} p(x_i|y_i = 1; \theta) + \log \sum_{i \in y_i=2} p(x_i|y_i = 2; \theta) + \dots + \log \sum_{i \in y_i=K} p(x_i|y_i = K; \theta)$$

נעת בשביל לחשב פרמטרים עבור כל y_k מספיק להסתכל על הדגימות עבורן $y = k$, כלומר:

$$\theta_{k_{MLE}} = \arg \max_{\theta_k} \log \sum_{i \in y_i=k} p(x_i|y_i = k; \theta_k)$$

על ידי גזירה והשוואה ל-0 ניתן לחשב את הפרמטרים האופטימליים:

$$\mu_k = \frac{\sum_{i \in y_i=k} x_i}{\sum_i \mathbb{1}_{y_i=k}}$$

$$\Sigma_k = \frac{\sum_{i \in y_i=k} (x_i - \mu_k)(x_i - \mu_k)^T}{\sum_i \mathbb{1}_{y_i=k}}$$

ניתן לשים לב שהתוחלת μ_k היא למעשה ממוצע הדגימות עבורן $y = k$. בעזרת הפרמטרים המשוערים ניתן לבנות את המסווג:

$$y = \arg \max_k p(y_k | x; \mu_k, \Sigma_k) = \arg \max_k \log p(x | y = k) p(y)$$

$$= \arg \max_k -\frac{1}{2} \log |\Sigma_k| - \frac{1}{2} (x - \mu_k)^T \Sigma_k^{-1} (x - \mu_k) + \log p(y)$$

עבור המקרה בו מטריצת ה-covariance היא אלכסונית, כלומר אין תלות בין משתנים שונים, מתקבל המסווג הבייסיאני הגאוסיאני (תוצאה זו הגיונית כיוון שהמסווג הבייסיאני מניח שבהינתן סיווג של אובייקט מסוים אין יותר תלות בין המשתנים).

עבור המקרה הבינארי, בו $y \in \{0, 1\}$, מתקבל סיווג בצורה של משוואה ריבועית:

$$y = 1 \Leftrightarrow -\frac{1}{2} \log |\Sigma_1| - \frac{1}{2} (x - \mu_1)^T \Sigma_1^{-1} (x - \mu_1) + \log p(y = 1) > -\frac{1}{2} \log |\Sigma_0| - \frac{1}{2} (x - \mu_0)^T \Sigma_0^{-1} (x - \mu_0) + \log p(y = 0)$$

נסמן:

$$a = \frac{1}{2} (\Sigma_1^{-1} - \Sigma_0^{-1})$$

$$b = \Sigma_1^{-1} \mu_1 - \Sigma_0^{-1} \mu_0$$

$$c = \frac{1}{2} (\mu_0^T \Sigma_0^{-1} \mu_0 - \mu_1^T \Sigma_1^{-1} \mu_1) + \log \frac{p(y = 1)}{p(y = 0)} - \log \frac{\sqrt{|\Sigma_1|}}{\sqrt{|\Sigma_0|}}$$

ונקבל:

$$y = 1 \Leftrightarrow x^T a x + b^T x + c > 0$$

זהו משטח הפרדה ריבועי.

Linear Discriminant Analysis הינו מקרה פרטי של Quadratic Discriminant Analysis, בו מניחים כי מטריצת ה-covariance זהה לכל ה-labels, כלומר $\Sigma_k = \Sigma$. במקרה זה מתקבל:

$$\log p(x | y = k) p(y) = -\frac{1}{2} (x - \mu_k)^T \Sigma^{-1} (x - \mu_k) + \log p(y)$$

הביטוי $(x - \mu_k)^T \Sigma^{-1} (x - \mu_k)$ נקרא מרחק מהלונביס, והוא מבטא את מידת הקשר בין x לבין μ_k תוך כדי התחשבות בשונות של כל משתנה. למעשה ניתן להסתכל על מסווג LDA כמסווג המשייך אובייקט ל-label עבורו המרחק על פי מטריקת מהלונביס הוא הכי קטן. על ידי גזירה והשוואה ל-0 מתקבל השערוך:

$$\mu_k = \frac{\sum_{i \in y_i=k} x_i}{\sum_i \mathbb{1}_{y_i=k}}$$

$$\Sigma_k = \frac{1}{n} \sum_i (x_i - \mu_k)(x_i - \mu_k)^T$$

והמסווג המתקבל הינו:

$$y = \arg \max_k p(y_k | x; \mu_k, \Sigma) p(y) = \arg \max_k -\frac{1}{2} (x - \mu_k)^T \Sigma^{-1} (x - \mu_k) + \log p(y = k)$$

$$= \arg \max_k -x^T \Sigma^{-1} \mu_k + \frac{1}{2} \mu_k^T \Sigma^{-1} \mu_k + \log p(y = k)$$

ניתן לסמן:

$$a = \Sigma^{-1} \mu_k$$

$$b = \frac{1}{2} \mu_k^T \Sigma^{-1} \mu_k + \log p(y = k)$$

ומתקבל מסווג לינארי (ומכאן השם של האלגוריתם):

$$y = \arg \max_k ax^T + b$$

מסווג זה מחלק כל שני אזורים בעזרת מישור לינארי, כאשר בסך הכל יש K קווי הפרדה. עבור המקרה הבינארי מתקיים:

$$a = \Sigma^{-1}(\mu_1 - \mu_0)$$

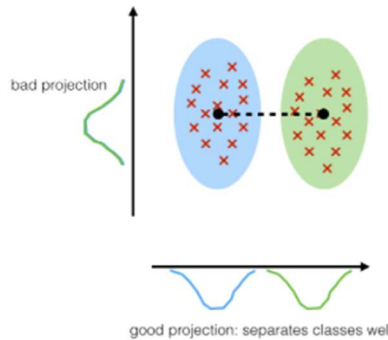
$$b = \frac{1}{2}(\mu_0^T \Sigma^{-1} \mu_0 - \mu_1^T \Sigma^{-1} \mu_1) + \log \frac{p(y = 1)}{p(y = 0)}$$

והסיווג הינו:

$$y = 1 \Leftrightarrow a^T x + b > 0$$

אלגוריתם LDA פשוט יותר מאלגוריתם QDA כיוון שיש פחות פרמטרים לשערך, אך יש לו שני חסרונות עיקריים – הוא לא גמיש אלא לינארי, ובנוסף הוא מניח שמטריצת ה-covariance זהה לכל ה-labels, מה שיכול לגרום לשגיאות בסיווג. כדי להתמודד עם הבעיה השנייה ניתן להשתמש באלגוריתמים המנסים למצוא את מטריצת ה-covariance שתביא לסיווג הטוב ביותר האפשרי (למשל Oracle Shrinkage Approximating ו-Ledoit-Wolf estimator).

באופן גרפי ניתן להסתכל על אלגוריתם LDA כמציאת כיוון ההפרדה בו יש את השונות הגדולה ביותר בין שתי התפלגויות נורמליות, ובנוסף יש בו את ההפרדה המקסימלית בין הקבוצות השונות. לאחר מציאת הקו האופטימלי, ניתן לחשב את ההתפלגויות של הקבוצות השונות כהתפלגויות נורמליות על הישר המאונך לקו ההפרדה:



איור 2.5 אלגוריתם LDA באופן גרפי: מציאת הכיוון של ההתפלגויות והטלת המידע על הציר האנכי לכיוון ההפרדה.

2.1.5 Decision Trees

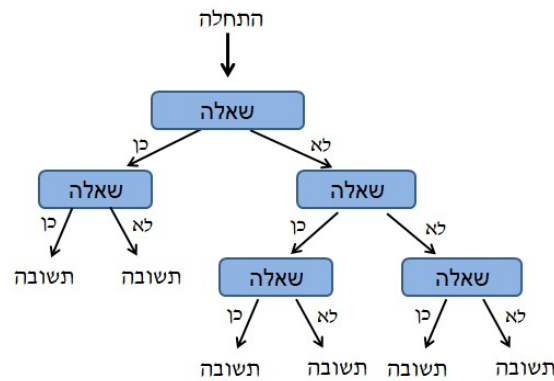
1. הקדמה

עץ החלטה הינו אלגוריתם לומד היכול לשמש הן לבעיות סיווג והן לבעיות רגרסיה. באופן כללי, עצי החלטה לוקחים את המשתנה שברצוננו להסביר (משתנה המטרה/החיזוי), ומחלקים את המרחב שלו לקבוצות (segments). לכל קבוצה של סט האימון מחשבים "ממוצע" (Mean) או "שכיח" (Mode), וכאשר מתקבלת תצפית חדשה משייכים אותה לקבוצה בעלת הממוצע או השכיח הדומה ביותר. מאחר וכללי הסיווג לקבוצות השונות יכולים להצטייר בצורת עץ, אלגוריתם זה נקרא "עץ החלטה".

הגישות השונות בעצי החלטה פשוטות ואינטואיטיביות להבנה, אולם הן לא מצליחות להתחרות במדדי הדיוק של מודלים אחרים של Supervised Learning. לכן, בפרקים הבאים נציג שיטות ensembles, בהן מתבצעת בנייה של כמה עצי-החלטה במקביל, אשר משולבים בסופו של דבר למודל יחיד. ניתן להראות ששימוש במספר גדול של עצים

יכול לשפר דרמטית את מדדי הדיוק של המודל. עם זאת, ככל שמשתמשים ביותר עצים כך יכולת הפרשנות של המודל נהיית מורכבת יותר ופחות אינטואיטיבית לצופה שאינו מעורב בבניית המודל (למשל גורם עסקי בארגון שאנחנו מעוניינים להסביר לו את תוצאות המודל).

ראשית נתבונן במבנה בסיסי של עץ ונגדיר עבורו את המושגים הרלוונטיים:



איור 2.6 דיאגרמה של עץ החלטה.

על מנת להבין את מבנה העץ ולייצר שפה משותפת נציג את השמות המקובלים בעבודה עם עצים:

- Root (שורש) – נקודת הכניסה לעץ (חלקו העליון ביותר של העץ).
- Node (צומת) – נקודת ההחלטה/פיצול של העץ – השאלות.
- Leaves (עלים) – הקצוות של העצים – התשובות. נקראים גם terminal nodes.
- Branch (ענף) – חלק מתוך העץ המלא (תת-עץ)
- Depth (עומק) – מספר ה-nodes במסלול הארוך ביותר בעץ.

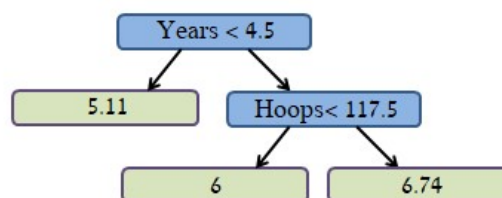
יסודות של עצי החלטה

עצי רגרסיה:

כאמור, עצי החלטה יכולים לשמש הן עבור סיווג והן עבור רגרסיה, ונתחיל עם דוגמא פשוטה לבניית עץ עבור בעיית רגרסיה – חיזוי שכר (באלפי שקלים) שחקני כדורסל באמצעות עצי החלטה. נרצה לחזות את השכר של שחקן בהתבסס על הנתונים הבאים:

- שנים - מספר העונות שאותו שחקן שיחק בליגת העל.
- סלים - מספר הסלים שהוא קלע בשנה הקודמת.

תחילה נסיר תצפיות ללא ערכים עבור המשתנה המוסבר שלנו, הלא הוא "שכר" ובנוסף נבצע על אותו משתנה Log-transform בכדי שהוא יהיה ככל הניתן בקירוב להתפלגות נורמלית (עקומת גאוס/פערמון).



איור 2.7 דוגמא של עץ החלטה עבור בעיית רגרסיה של עץ החלטה.

כאמור, האיור מתאר עץ המנסה לחזות את Log השכר (באלפי שקלים) של שחקן בהינתן מספר עונות הותק שלו וכמות הסלים שקלע בעונה הקודמת. ניתן לראות שחלקו העליון של העץ (ה-Root) מתחלק ל-2 ענפים. הענף השמאלי מתייחס לשחקנים בעלי ותק של יותר מ-4.5 עונות ($years < 4.5$), והענף הימני מתייחס לשחקנים פחות ותיקים. לעץ יש 2 צמתים (=שאלות/Internal nodes) ו-3 עלים (=תשובות/Terminal nodes). המספר בכל עלה שבתחתית העץ הוא הממוצע של כל התצפיות שסווגו לעלה הזה. לאחר שבניית העץ הסתיימה, כל תצפית חדשה

שתסווג לעלה מסוים תקבל את הערך הממוצע של התצפיות שעל בסיסם נבנה העץ. בהמשך הפרק יוסבר כיצד לבנות את העץ וכיצד לקבוע את כללי הפיצול בהתאם לדאטה הקיים.

בסופו של דבר העץ מחלק את השחקנים לשלוש קבוצות:

- שחקנים ששיחקו 4 עונות או פחות:

$$R_1 = \{X|Years < 4.5, Hoops\} = 5.11$$

- שחקנים ששיחקו 5 עונות או יותר וקלעו פחות מ-118 סלים בעונה שחלפה:

$$R_2 = \{X|Years > 4.5, Hoops < 117.5\} = 6$$

- שחקנים ששיחקו 5 עונות או יותר וקלעו יותר מ-118 סלים בעונה שחלפה:

$$R_3 = \{X|Years > 4.5, Hoops > 117.5\} = 6.74$$

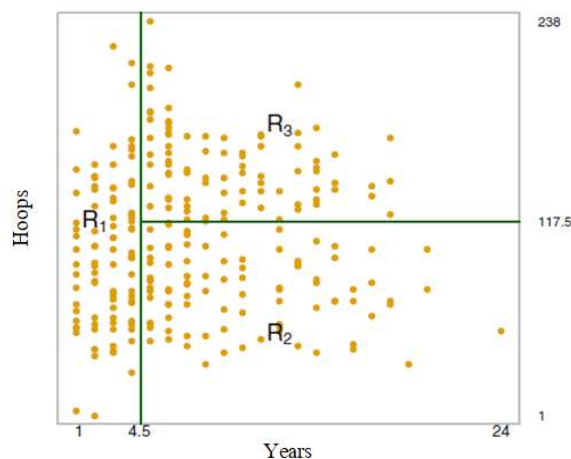
בהתאם לאנלוגיה של העץ, הקבוצות R_1, R_2, R_3 מכונות בשם Terminal nodes, או "עלים" של העץ.

אפשר לפרש את עץ הרגרסיה שבדוגמא הצורה נוספת: Years הוא הפרמטר המרכזי ביותר בקביעה מה יהיה גובה השכר, ושחקנים עם פחות שנות ניסיון ירוויחו פחות משחקנים מנוסים יותר. עבור שחקן יחסית חדש (עד 5 שנים בליגה) הפרמטר של כמות הסלים אותה הוא קלע בעונה הקודמת מתברר כפחות משפיע על השכר. כלומר, כל עוד לשחקן אין 5 שנות ניסיון, כמות הסלים היא יחסית שולית ביחס לחוסר הניסיון עבור קביעת שכרו. לעומת זאת, בקרב שחקנים עם 4.5 שנות ניסיון או יותר, כמות הסלים שהם קלעו בשנה הקודמת כן משפיעה על השכר, ושחקנים שקלעו הרבה סלים בשנה הקודמת כנראה ירוויחו יותר כסף מאשר שחקנים עם אותו ניסיון אך קלעו פחות סלים.

עץ הרגרסיה שהובא בדוגמא מפשט קצת "יתר על המידה" את הקשר "האמיתי" בין Hoops, years ו-Salary, והחיזוי של העץ לא מספיק טוב ביחס למודלים אחרים של רגרסיה, כמו למשל רגרסיה ליניארית שתוסבר בפרק 3. עם זאת, מודל זה פשוט יותר להבנה ופרשנות וקל יחסית לייצוג באמצעים גרפיים.

2. חיזוי באמצעות ריבוד (Stratification) של מרחב המשתנים:

עד כה הוסבר באופן אינטואיטיבי מהו עץ החלטה וכיצד הוא פועל. האתגר המרכזי הוא לבנות את העץ בצורה טובה כך שהחיזוי שלו אכן יהיה תואם למציאות. בכדי להבין כיצד בונים עץ בצורה יעילה, נציג את הדוגמא הקודמת באופן נוסף:



איור 2.8 דוגמא של עץ החלטה עבור בעיית רגרסיה של עץ החלטה.

באיור זה ניתן לראות שהנתונים נפרסו על פני מרחב דו ממדי, וחולקו בעזרת קווי החלטה בהתאם ל-Internal nodes. כעת נגדיר את תהליך החלוקה והחיזוי בשני שלבים:

1. מחלקים את מרחב הערכים האפשריים של המשתנה אותו רוצים לחזות ל- J אזורים נפרדים R_1, \dots, R_J כתלות בפרמטרים השונים X_1, \dots, X_n .
2. לאחר החלוקה, כל תצפית שנופלת באזור R_i תקבל את ערך הממוצע של הנקודות מסט האימון שמרכיבות את הקבוצה R_i .

באופן תיאורטי, לכל אזור יכולה להיות כל צורה שהיא. אולם לטובת הפשטות, אנו בוחרים לחלק את המשתנה ל- "אזורים מרובעים". המטרה היא למצוא את האזורים שמביאים למינימום את סכום ריבועי ההפרשים בין התוויות של הנתונים בסט האימון לבין הערכים של כל אזור. מדד זה נקרא residual sum of squares (RSS), שמוגדר כך:

$$RSS = \sum_{j=1}^J \sum_{i \in R_j} (y_i - \hat{y}_{R_j})^2$$

כאשר \hat{y}_{R_j} הוא ממוצע המשתנים של תצפיות האימון באזור j , ו- $y_i - \hat{y}_{R_j}$ הוא המרחק בין כל תצפית לבין הערך הממוצע באזור זה. כאמור, המטרה של מדד זה היא למצוא את מקבץ התצפיות בכל אזור, כך שריבוע המרחק בין הערך הממוצע לבין כל תצפית יהיה מינימלי. כיוון שבדיקת כל החלוקות השונות האפשריות אינה ריאלית מבחינה חישובית, לרוב משתמשים באלגוריתם חמדן (greedy) אשר עובד "מלמעלה למטה". גישה זו ביחס לעץ החלטה נקראת "פיצול בינארי רקורסיבי" (recursive binary splitting), והיא נקראת "מלמעלה למטה" כיוון שהיא מתחילה מראש העץ (root), היכן שכל התצפיות עדיין שייכות לקבוצה אחת גדולה. לאחר סימון נקודת ההתחלה, האלגוריתם מפצל את מרחב הערכים של המשתנה אותו רוצים לחזות, כאשר כל פיצול מסומן באמצעות שני ענפים חדשים בהמשך העץ.

האלגוריתם כאמור הוא חמדן, וזה מתבטא בכך שבכל שלב בתהליך בניית העץ בוחרים לבצע את הפיצול הטוב ביותר עבור השלב הנוכחי, מבלי להתחשב כמה שלבים קדימה. בגישה זו יתכן ובשלב הנוכחי יש פיצול יותר יעיל לטווח ארוך, אך הוא לא יבחר אם הוא לא הפיצול האופטימלי בשלב זה. ניתן להמחיש את דרך הפעולה של אלגוריתם חמדן לעמידה בפקק תנועה, ומתוך ניסיון לעקוף את הפקק נבחרת הפניה הראשונה שנראית פחות פקוקה, מבלי להתחשב בפקק שעשוי לבוא בהמשך. כמובן שפניה זו לא בהכרח תוביל לדרך יותר מהירה, ויתכן שבראייה יותר ארוכת טווח דווקא היה מוטב להימנע מפנייה זו כיוון שהיא מובילה לפקק גדול יותר בהמשך.

כדי לבצע את אותו "פיצול בינארי רקורסיבי", יש לבצע את התהליך האיטרטיבי הבא:

עבור כל פרמטר אפשרי X_j וקו חלוקה s , נקבל שתי קבוצות:

$$R_1(j, s) = \{X | X_j < s\}$$

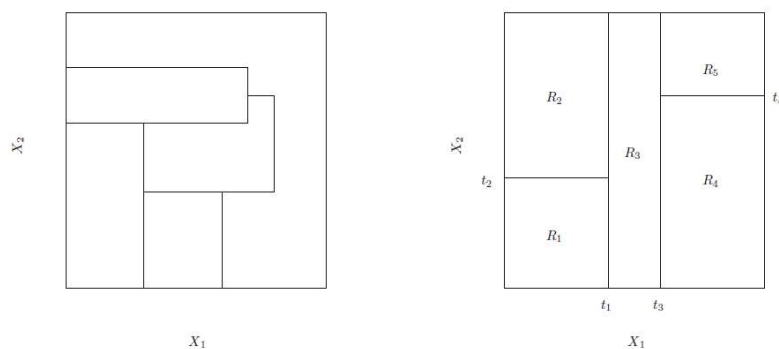
$$R_2(j, s) = \{X | X_j \geq s\}$$

עבור שתי קבוצות אלה נחפש את הערכים של j ו- s שמביאים למינימום את המשוואה הבאה:

$$\sum_{i: x_i \in R_1(j, s)} (y_i - \hat{y}_{R_1})^2 + \sum_{i: x_i \in R_2(j, s)} (y_i - \hat{y}_{R_2})^2$$

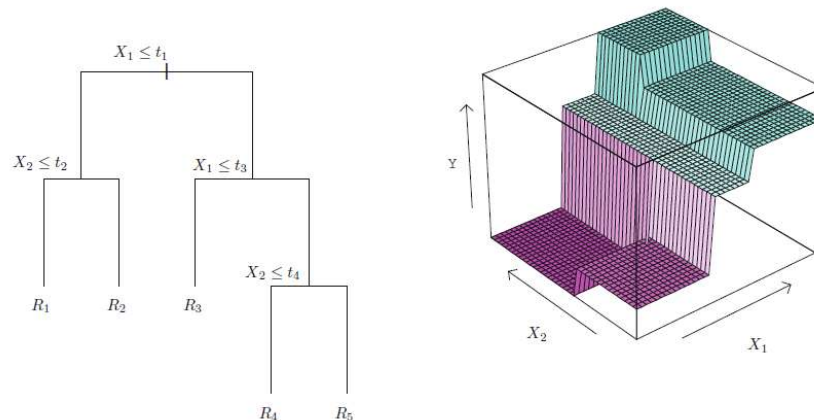
מציאת ערכי j, s שמביאים למינימום את המשוואה יכולה להתבצע די בקלות כשכמות הפרמטרים לא גדולה מדי, אך זה תהליך שעלול להיות די סבוך כשיש הרבה פרמטרים.

למעשה תהליך זה מייצר ענף אחד שיוצא מה-root, ולו 2 עלים. כעת מבצעים את התהליך שוב, מתוך מטרה למצוא את הפרמטר הבא שמביא למינימום את RSS בכל קבוצה, ובכך לקבל 3 קבוצות. ובאופן דומה, נרצה לפצל את אחת מאותן 3 קבוצות שכבר יש לנו כדי לצמצם עוד את ה-RSS. התהליך הזה נמשך איטרטיבית עד שמגיעים "לקריטריון עצירה" מסוים, כמו למשל מספר פיצולים, מספר מקסימלי של תצפיות בכל אזור וכו'. אחרי שהתבצעה החלוקה לקבוצות R_1, \dots, R_j , ניתן להפוך את הקבוצות לעץ, ולהשתמש בו בכדי לחזות נקודות חדשות



איור 2.9 חלוקה של משתנים לאזורים.

באיור המצורף ניתן לראות שתי חלוקות – החלוקה הימנית הינה חלוקה דו-ממדית של שני משתנים באמצעות פיצול בינארי רקורסיבי. החלוקה השמאלית היא גם חלוקה דו-ממדית של שני משתנים, אך היא לא יכלה להיווצר באמצעות פיצול בינארי רקורסיבי, כיוון שהיא מורכבת מידי ואינה תואמת את הגישה החמדנית שרוצה "חלוקה פשוטה ומהירה".



איור 2.10 תיאור אזורי חלוקה כעץ בינארי. מצד שמאל מוצג העץ התואם לחלוקה מהאיור הקודם, ומצד ימין ישנה פרספקטיבה רחבה כיצד חיצויים מתבצעים באמצעות העץ.

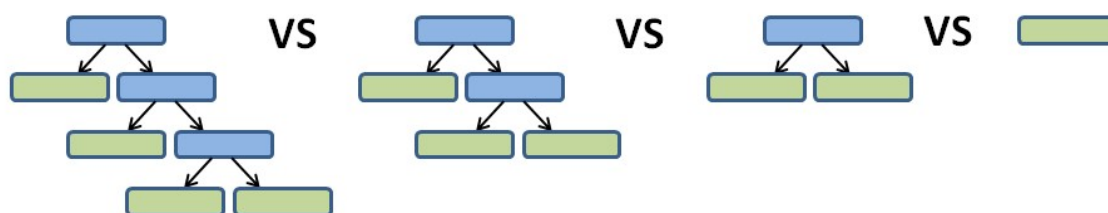
3. גיזום (pruning):

התהליך שתואר משקף את התהליך הקלאסי של יצירת עץ רגרסיה, והוא מסוגל לנפק חיצויים טובים (מבחינת מדד RSS ושונות נמוכה). עם זאת העץ עשוי לבצע התאמת-יתר (Over-fitting) על הנתונים, מה שיוביל לביצועים לא טובים עבור דאטה חדש. בעיה זו עלולה לנבוע מכך שהעץ שנבנה בתהליך האימון עשוי להיות "מורכב מידי" ומותאם יתר על המידה לנתוני האימון, מה שפוגע ביכולת ההכללה שלו. לעומת עץ מורכב, עץ דליל יותר בעל פחות פיצולים (כלומר פחות קבוצות R_1, \dots, R_j) ייתן חיצויים בעלי הטיה (Bias) גדולה יותר בהשוואה לאלטרנטיבה הראשונה, אך נראה יוביל לשונות קטנה יותר בין סט האימון לבין סט המבחן, ובנוסף מודל כזה יהיה קל יותר לפרשנות.

גישה פשטנית בכדי להתמודד עם בעיה זו היא לקבוע רף כלשהו, כך שבכל פיצול הירידה ב-RSS תעלה על רף זה. כלומר, פיצול שמוביל לירידה שולית יחסית ב-RSS לא יתבצע. באופן הזה העצים שיתקבלו יהיו קטנים יותר ויש פחות חשש ל-over-fitting. החיסרון בגישה זו נעוץ בכך שהיא קצרת-ראייה. יתכן מצב בו יש פיצול בעל תרומה קטנה יחסית אך הוא יכול להוביל לפיצול אחר בעל תרומה גדולה, כזה שיביא לירידה גדולה ב-RSS בהמשך. שיטה זו תדלג על אותו פיצול בעל ערך קטן, מאחר והוא לא עובר את הרף שהוגדר.

גישה אחרת, שמתברר והיא טובה יותר, הולכת בכיוון ההפוך. בשלב הראשוני יבנה עץ גדול מאוד (T_0), ולאחר מכן נגזום ממנו כל מיני פיצולים בכדי להימנע מ-over-fitting. את מקומם של הענפים שהסרנו יתפוס עלה בודד שמקבל ערך ממוצע המתחשב במספר גדול יותר של תצפיות. כמובן שצעד זה יגרום לירידה בביצועים על פני סט האימון, אך השאיפה היא שזה ישתלם בבדיקת השגיאה בסט המבחן.

השאלה הגדולה היא כמובן מהי הדרך הטובה ביותר לגזום את העץ. אינטואיטיבית, המטרה שלנו היא לבחור subtree מתוך העץ המקורי שמוביל לשגיאה הקטנה ביותר. אומדן זה יכול להתבצע על ידי בדיקת השגיאה של העץ החדש ביחס לסט המבחן. כיוון שאי אפשר לבחון את כל תתי העצים האפשריים, נהוג להשתמש בגישה הנקראת Cost complexity pruning (ידועה גם בשם weakest link pruning). בגישה זו, במקום להתחשב בכל subtree אפשרי, אנו מסתכלים על רצף מסוים של subtrees שהם למעשה חלקים שונים המרכיבים את T_0 – העץ המקורי שנבנה בהתחלה:

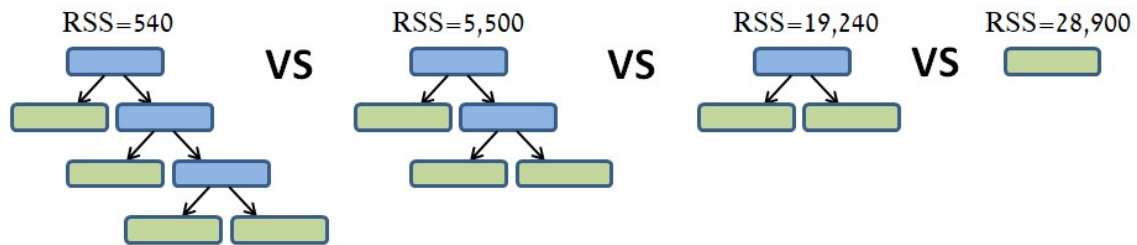


איור 2.11 חיפוש תת עץ אופטימלי ביחס ל- T_0 . מתחילים מ- T_0 (העץ השמאלי) וכל פעם מורידים פיצול יחיד עד שמגיעים ל-root.

כעת מחשבים עבור כל אחד מרצף העצים שהתקבל את ה-RSS שלו. לפני שנסביר באיזה עץ לבחור, נסכם את השלבים שבוצעו עד כה:

- בניית עץ רגרסיה גדול מכל הדאטה (ולא רק מסט האימון) בגישת ה- recursive binary splitting (הגישה החמדנית שתוארה לעיל). העץ ימשיך להבנות עד קריטריון עצירה מסוים (כמו למשל – הגעה למינימום של תצפיות). עץ הזה יסומן על ידי T_0 .
- כל עלה בעץ מקבל את הערך הממוצע של תצפיות הפרמטר שבאותו עלה, ועבור כל עלה פיצול מחושב ה-RSS.
- סכימת כל ערכי ה-RSS שקיבלנו בכל העלים. הסכום שהתקבל הוא ה-RSS של כל העץ T_0 .
- כעת מבצעים את שלבים א-ג ל-subtree הבא ברצף. זהו עץ-משנה שכמעט זהה לעץ המקורי, למעט שני עלים שנגזמו מהעץ הקודם. כך חוזרים על התהליך עבור כל ה-subtrees, עד ל-subtree האחרון שמורכב רק מה-root של העץ המקורי.

התוצר של השלב האחרון הוא RSS לכל subtree ברצף העצים. ניתן להבחין כי בכל פעם שענף מסוים הוסר, ה-RSS שהתקבל נהיה גדול יותר לעומת ה-subtree מהשלב הקודם. תוצאה זו הגיונית, שהרי כל פיצול במקור נועד להקטין את השגיאה באמצעות הקטנת ה-RSS, ואילו גיזום עושה את ההיפך – הוא נועד למנוע over-fitting, גם במחיר של שגיאה גדולה יותר.



איור 2.12 ביצוע גיזום וחישוב RSS לכל תת עץ (subtree) שהתקבל.

ה. כעת יש לבחור את אחד מה-subtrees, וכאמור זה נעשה בגישת Cost complexity pruning. גישה זו משקללת עבור כל תת עץ את מדד ה-RSS שלו יחד עם מספר העלים בעץ. באופן פורמלי:

$$Tree\ score = RSS + \alpha T$$

כאשר T הוא מספר העלים בעץ (Terminal nodes) ו- α הוא פרמטר רגולריזציה הקובע את היחס בין מספר העלים לבין מדד ה-RSS. פרמטר זה מחושב באמצעות Cross-validation המבצע trade-off בין הרצון להגדיל את העץ ובכך להקטין את ה-RSS לבין הרצון להימנע עד כמה שניתן מ-over-fitting. המחבר αT מכונה Tree Complexity Penalty, וכאמור הוא נועד "לפצות" על ההפרש במספר העלים שבין ה-subtrees השונים שנבחנו.

משלב ד' כבר קיבלנו RSS לכל subtree, וכעת נשאר רק לחשב לכל אחד מהם את ה- Tree score.

נשים לב כי:

- כאשר $\alpha = 0$, העץ המקורי (T_0) יהיה בהכרח בעל ה-Tree score הנמוך ביותר, כיוון שכאשר $\alpha = 0$ אזי כל הרכיב αT בנוסחה שווה ל-0. במקרה זה ה-Tree score יהיה לזהה לגמרי למדד ה-RSS:

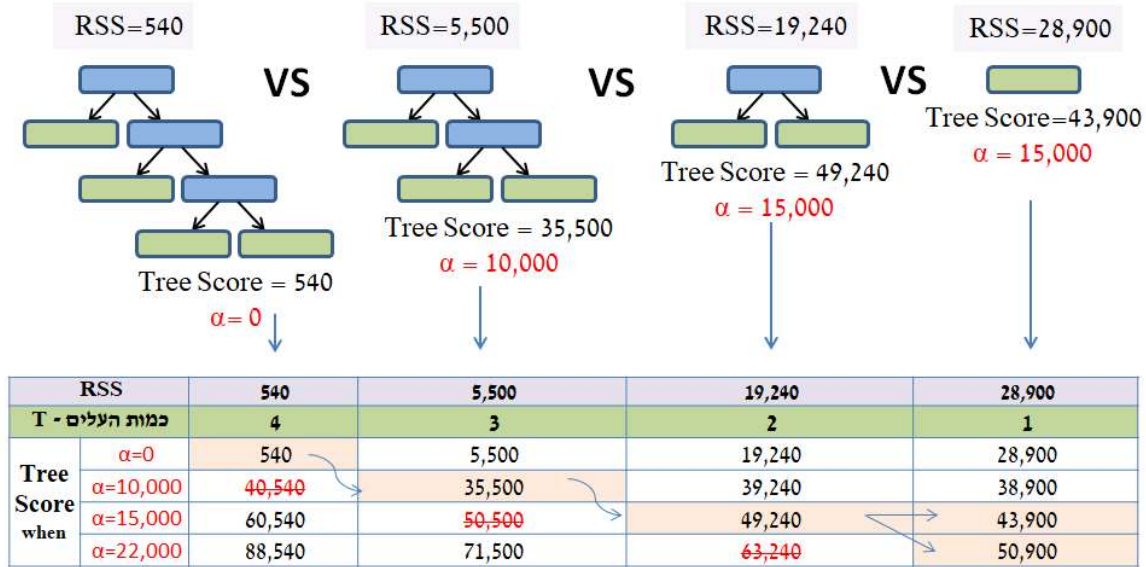
$$Tree\ score = RSS + 0 \cdot T = RSS$$

ממילא שאר שלבי החישוב מיותרים, כי אנחנו כבר יודעים שהעץ T_0 הוא בעל ה-RSS הנמוך ביותר מכל ה-subtrees, וכן בעל ה-Tree Score הנמוך ביותר. לכן, נרצה לקבוע ש- $\alpha > 0$. נתבונן מה קורה עבור ערכי α שונים:

- עבור $\alpha = 10,000$ הציון של T_0 יהיה 40,540. ולכן שווה לנו לגזום 2 עלים ולקבל עבור אותו α ציון נמוך יותר מ-40,540 (העץ השני משמאל עם ציון של 35,500). ושוב, אם נשאר ב- $\alpha = 10,000$ ובמקביל נגזום 2 עלים נוספים (אנחנו כעת בעץ השלישי משמאל), נקבל ציון של 39,240 – שזה עדיין גבוה יותר מ-35,500 ולכן זה לא טוב לנו.

עבור $\alpha = 15,000$ ניתן לראות שהציון המתקבל יהיה נמוך יותר מה-subtree הקודם. כעת נשים לב שבמעבר ל-subtree האחרון (ה-root) לא משנה אם α יגדל או יישאר אותו דבר, בכל מקרה הציון של ה-root יהיה נמוך יותר מה-subtree הקודם.

ז. למעשה חזרנו על אותם צעדים עד שאין יותר מה לגזום. בסופו של תהליך ערכים שונים של α יתנו רצף של עצים, מעץ מלא ועד לעלה בודד. יוצא מכך שלכל ערך של α קיים subtree מתאים $T \subset T_0$ כך שה-Tree score המתקבל יהיה קטן ככל האפשר.

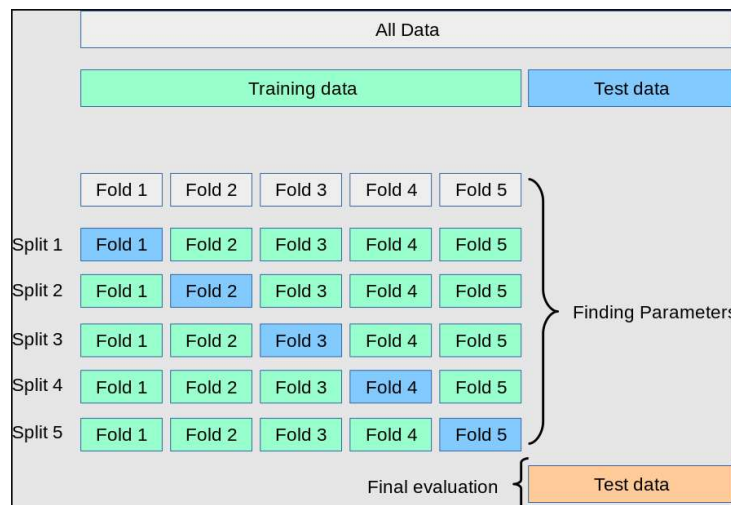


איור 2.13 חישוב ערך α מותאם לכל עץ משנה כך שיתקבל Tree score קטן ככל האפשר.

ח. כעת, נבחר ב-subtree עם ה-score הנמוך ביותר – כלומר העץ השני משמאל עם ציון של 35,500. במקביל יש לזכור מהם ערכי ה- α של ה-subtrees השונים שהתקבלו (בסעיף הקודם), כיוון שהם יהיו שימושיים לחישוב ערך α אופטימלי:

○ ערכי ה- α חשובים, כיוון שלכל α עשוי להתקבל עץ אחר בעל ציון מינימלי. עד לשלב זה העץ נבנה ביחס לכל הדאטה (בלי חלוקה ל-Train ו-Test), אך המטרה היא לבחון מהו ערך ה- α שייתן את התוצאה האופטימלית בעזרת העץ הגזום עבור דאטה חדש.

ט. כדי לבחור את ערך ה- α האופטימלי ניתן להשתמש ב-Cross-Validation. לשם כך, יש לקחת את הדאטה המלא (ממנו נבנה העץ הראשון – T_0), ולחלק אותו באופן הבא:



איור 2.14 cross-validation עם $K_{fold} = 5$.

- ב-cross-validation המודל מתאמן תחילה לפי 1 split (איור 2.14 לעיל) על התצפיות שבחלקים הירוקים, ובוחן את החיזויים על סט התצפיות הכחול.
- התהליך הזה נעשה K פעמים, כאשר בכל איטרציה האימון נעשה את התצפיות שבחלקים הירוקים, ובחינת החיזויים (וחישוב ה-RSS) נעשה על התצפיות שבחלק הכחול.

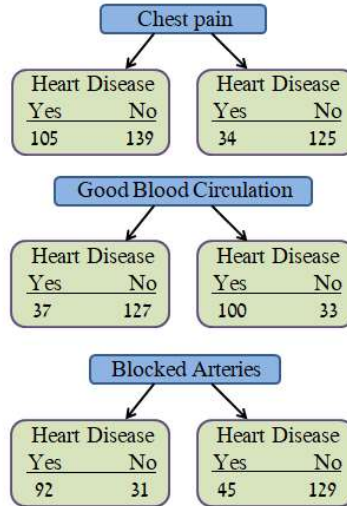
- בכל אחד מה-splits ב-cross-validation המודל משתמש רק ב-Training data כדי לבנות עץ מלא (נסמן אותו הפעם ב- T_1) ורצף (sequence) חדש של subtrees שמביאים למינימום את ה-Tree Score (אותו סדר פעולות כמו בסעיפים א'-ד'), בעזרת אותם ערכי α שהתקבלו בסעיף ז'.
- א. כעת יש לחשב את ה-RSS לכל subtree באמצעות שימוש ב-Test data בלבד, ולזכור מיהו ה-Subtree שקיבל את ה-RSS הנמוך ביותר. נניח שבאיטרציה הראשונה העץ שקיבל את ה-RSS הנמוך ביותר ב-Testing data הוא דווקא העץ שבו $\alpha = 10,000$.
- ב. את התהליך של סעיפים י-יא יש לבצע K פעמים – פעם אחת עבור כל split, כאשר בכל איטרציה האימון מתבצע על התצפיות שבחלקים הירוקים, ובחינת החיזויים (וחישוב ה-RSS) מתבצע על התצפיות שבחלק הכחול.
- ג. בסופו של התהליך, כל איטרציה תניב subtrees בעלי RSS מסוים, וכן ערכי α שנקבעו כבר מראש בסעיף ז'. לאחר K האיטרציות יש לבדוק מיהו העץ עם ה-RSS המינימלי מבין כל האיטרציות, ומהו ערך ה- α של העץ הזה, וזה יהיה הערך הסופי של α .
- ד. לבסוף, יש לחזור לעץ המקורי T_0 וה-subtrees שנבנו מה-data set המלא (שמכיל גם את ה-Training וגם את ה-Test), ולבחור את העץ שתואם לערך ה- α הנבחר. ה-subtree הזה יהיה "העץ הגזום הסופי" שיבחר.

לסיכום:

- אחרי כל החישובים מצאנו שהעץ T_0 הוא בעל ה-RSS הנמוך ביותר, אך יתכן והוא יסבול מ-Overfit. כדי לפצות על כך, נוסף רכיב שנועד להתחשב גם ביתר העצים שהינם בעלי RSS גבוה יותר, ו-"מעניש" את העץ המקורי (T_0) עקב ריבוי העלים שבו.
- באופן הזה התקבל ציון המאפשר להשוות בין העצים ולבחור את העץ הטוב ביותר. העץ הזה מהווה מעין איזון בין הרצון ל-RSS מינימלי לבין שונות נמוכה בין ה-Train ל-Test.
- בכדי למצוא את ה- α האופטימלי, שמצד אחד נותן עץ בעל RSS אופטימלי ומצד שני נמנע כמה שניתן מ-Overfitting ניתן להיעזר ב-cross-validation.

4. עץ סיווג

עץ סיווג די דומה לעץ רגרסיה, רק שהמטרה היא שונה – במקום לקבל תשובה כמותית כמו ברגרסיה, עץ סיווג ייתן תווית (label) לתצפית המבוקשת. כאמור לעיל, עץ רגרסיה מספק חיזוי לתצפית מסוימת בהתאם לערך הממוצע של תצפיות האימון ששייכות לאותו terminal node. בעץ סיווג לעומת זאת, כל תצפית תשוּיך לקבוצה (class) בעל תווית משותפת. למשל, נניח ומעוניינים לסווג מטופל מסוים האם יש לו מחלת לב או לא. אנו יכולים לבנות עץ החלטה על בסיס מאפיינים של חולים שאובחנו בעבר ואנו יודעים להגיד מי מהם באמת חולה לב ומי לא, ועל בסיס העץ הזה להחליט עבור כל מטופל חדש האם הוא דומה במאפיינים שלו למטופלים שאובחנו בעבר כחולי לב או לא. כך שהתשובה שהעץ נותן היא לא "ערך ממוצע" כמו שראינו בעצי רגרסיה, אלא פשוט החלטה - "כן חולה לב" או "לא חולה לב". מלבד החיזוי של התווית, עץ סיווג מספק גם יחסים בין הקבוצות השונות בקרב תצפיות האימון שנופלים באותו אזור. נתבונן בדוגמה שתמחיש את העניין:



איור 2.15 השפעת פרמטרים שונים על הסיכוי לחלות במחלת לב.

באיור לעיל ניתן לראות שלושה פרמטרים (שבמקרה הזה הם סימפטומים של מטופל) בעזרתם מנסים לסווג האם למטופל יש מחלת לב או לא. כפי שניתן לראות אף אחד מהמשתנים אינו יכול לענות על שאלה זו בפני עצמו, כיוון שבאף אחד מהעלים אין אחידות בתצפיות. משתנים כאלה, אשר אינם יכולים בפני עצמם לספק סיווג מושלם, נקראים משתנים לא הומוגניים (impure - לא טהורים). כיוון שברוב המקרים כל המשתנים אינם הומוגניים, יש למצוא דרך כיצד לבחור באחד מהמשתנים להיות המשתנה שבראש העץ (Root node). כלומר, יש לייצר מדד הבוחן ומשווה את רמת ה-impurity של כל משתנה. ישנם מספר מדדים, ונתמקד בשניים מהם – Entropy ו-Gini index.

א. מדד Gini index:

נסמן את ההסתברות לשיוך תוויית מסוימת לקבוצה j ב- p_j , ונגדיר:

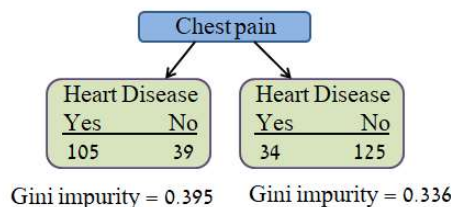
$$Gini = 1 - \sum_{j=1}^J p_j^2$$

באופן אינטואיטיבי, מדד Gini מייצג את הסיכוי לקבלת סיווג שגוי עבור בחירה רנדומלית של נקודה מהדאטה בהתאם לפרופורציות של כל class בדאטה. נדגים זאת על אחד הפרמטרים שבדוגמה הקודמת – Chest pain. עבור הענף הימני מתקיים:

$$Gini = 1 - \left(\frac{34}{34 + 125}\right)^2 - \left(\frac{125}{34 + 125}\right)^2 = 0.336$$

ועבור הענף השמאלי מתקיים:

$$Gini = 1 - \left(\frac{39}{39 + 105}\right)^2 - \left(\frac{105}{39 + 105}\right)^2 = 0.395$$



איור 2.16 חישוב מדד Gini עבור הפרמטר Chest pain.

אחרי שחישבנו את ה-Gini Impurity לשני העלים, נחשב את מדד Gini הכולל של כל המשתנה Chest Pain. בשביל חישוב זה יש לאזן בין מספר התצפיות שבכל עלה, באופן הבא:

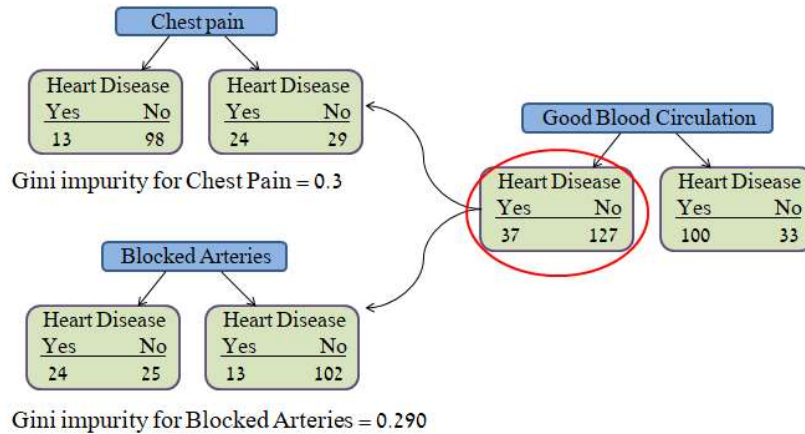
$$\text{Gini impurity for chest pain} = \left(\frac{144}{144 + 159}\right) \times 0.395 + \left(\frac{159}{144 + 159}\right) \times 0.336 = 0.364$$

באופן דומה ניתן לחשב את מדד Gini גם עבור יתר המשתנים ונקבל:

$$\text{Gini impurity for good blood circulation} = 0.36$$

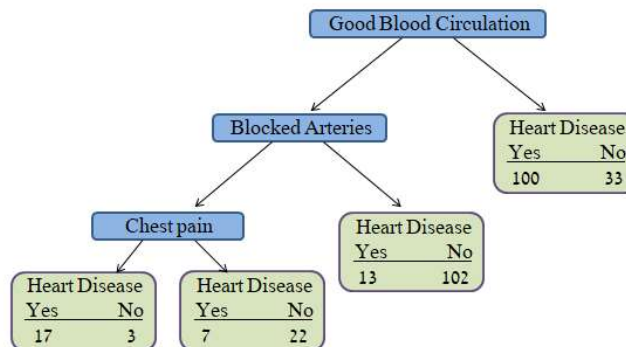
$$\text{Gini impurity for blocked arteries} = 0.381$$

למשתנה Good blood Circulation יש את הציון הכי נמוך, מה שאומר שהוא מסווג הכי טוב את המטופלים עם ובלי מחלת לב, ולכן נשתמש בו כמשתנה המסווג הראשון בראש העץ (ה-root). בכדי לקבוע את הפיצול הבא, יש להתבונן כיצד שאר הפרמטרים מסווגים את התצפיות של ה-root. נניח למשל ומתקיים:



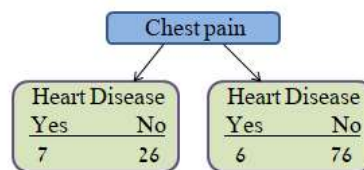
איור 2.17 חישוב מדד Gini עבור שאר הפרמטרים לאחר קביעת ה-root.

כפי שניתן לראות, למשתנה Blocked arteries יש ציון Gini נמוך יותר, ולכן הוא זה שנבחר להיות הפיצול הבא. לבסוף נבחן כיצד הפרמטר האחרון מסביר את התצפיות של הפיצול שלפניו, ונקבל את העץ הבא:



איור 2.18 חישוב מדד Gini עבור פיצול לפי הפרמטר Chest pain ביחס לשאר העץ.

נשים לב שניתן לפצל גם את העלה 13/102 לפי הפרמטר Chest pain, באופן הבא (המספרים כמובן תלויים בתצפיות האמיתיות):



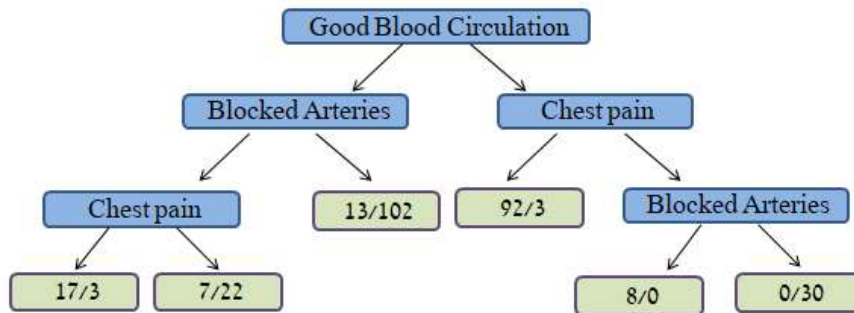
$$\text{Gini impurity for Chest Pain} = 0.29$$

איור 2.19 חישוב מדד Gini עבור פיצול לפי הפרמטר Chest pain ביחס לעלה 13/102.

מדד Gini שהתקבל הינו 0.29, בעוד שבלי הפיצול המדד של העלה היה 0.2, ולכן במקרה הזה עדיף להשאיר אותו כפי שהיה לפני ניסיון הפיצול. כעת באופן דומה נבנה גם את הענף הימני של העץ:

1. נחשב את מדדי Gini.
2. אם לענף הקיים יש ציון Gini נמוך יותר, אז אין טעם לפצל עוד, והוא הופך להיות terminal node.
3. אם פיצול הענף הקיים מביא לשיפור, בוחרים את המשתנה המפצל בעל ציון Gini הנמוך ביותר.

עץ טיפוסו לאחר סיום התהליך נראה כך:

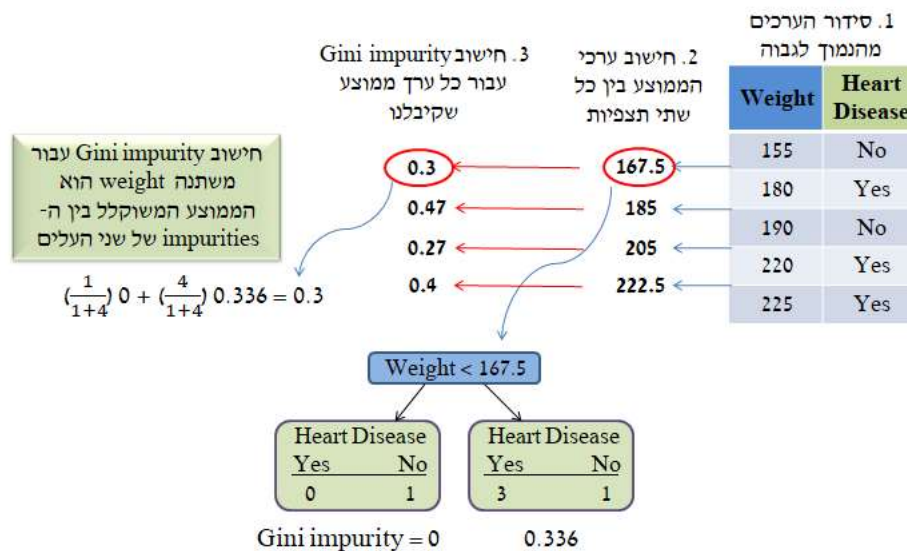


איור 2.20 חישוב מדד Gini עבור פיצול לפי הפרמטר Chest pain ביחס לעלה 13/102.

מדד Gini שהתקבל הינו 0.29, בעוד שבלי הפיצול המדד של העלה היה

התהליך שתואר מתאים למצבים בהם הפרמטרים מתפצלים באופן בינארי, כלומר הפיצול של כל פרמטר נקבע על ידי שאלה שעליה יש תשובה של כן או לא. במקרים בהם ישנם משתנים רציפים, הפיצול דורש כמה שלבים מקדימים:

1. סידור ערכי הפרמטרים מהערך הנמוך ביותר לערך הגבוה ביותר.
2. חישוב הערך הממוצע בין כל 2 תצפיות.
3. לחישוב Gini impurity עבור כל ערך ממוצע שהתקבל בשלב הקודם.



איור 2.21 פיצול פרמטרים רציפים לפי מדד Gini.

אנחנו מקבלים את ה-Gini הנמוך ביותר מתי שאנחנו קובעים Threshold של $weight < 205$.

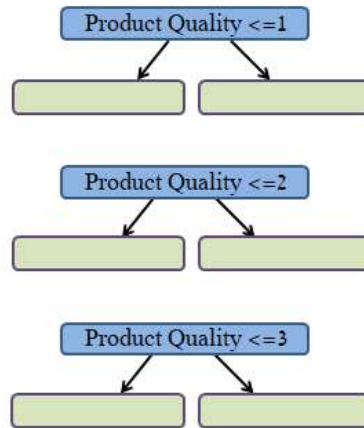
אז זהו ה-cutoff וערך ה-Gini שנשתמש בהם כשאנחנו משווים את המשתנה weight ליתר המשתנים.

עד עכשיו דיברנו על איך מחלקים משתנה רציף ואיך מחלקים משתנה בינארי (שאלות כן/לא). כעת נדבר איך מחלקים משתנים קטגוריאליים. למשל: משתנה מדורג שמקבל ערכים מ-1 עד 4. למשל: טיב מוצר מ-1-4. או משתנה שמכיל מספר קטגוריות. למשל: צבע מועדף (מכיל ערכים: אדום, ירוק, כחול וכו').

משתנה מדורג מאוד דומה למשתנה רציף, חוץ מזה שאנחנו צריכים לחשב בו את הציון עבור כל חלוקה אפשרית.

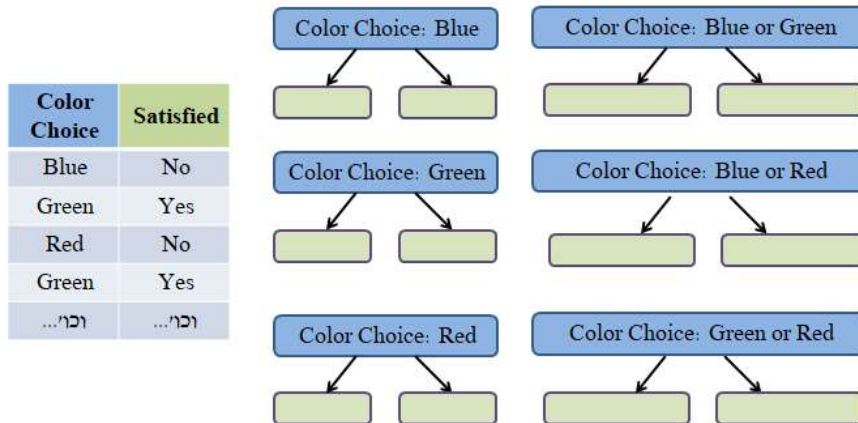
Product Quality	Satisfied
1	No
1	Yes
3	No
1	Yes
... וכי...	... וכי...

שימו לב: אנחנו לא צריכים לחשב את ה-impurity score ל- $prod. quality \leq 4$ בגלל שזה יכול בעצם את כולם



איור 2.22 פיצול פרמטרים קטגוריאליים לפי מדד Gini.

כשיש משתנה קטגוריאלי עם כמה קטגוריות, אפשר לחשב את ציון ה-Gini עבור כל קטגוריה, כמו גם עבור כל קומבינציה אפשרית:



איור 2.23 פיצול פרמטרים קטגוריאליים לפי מדד Gini.

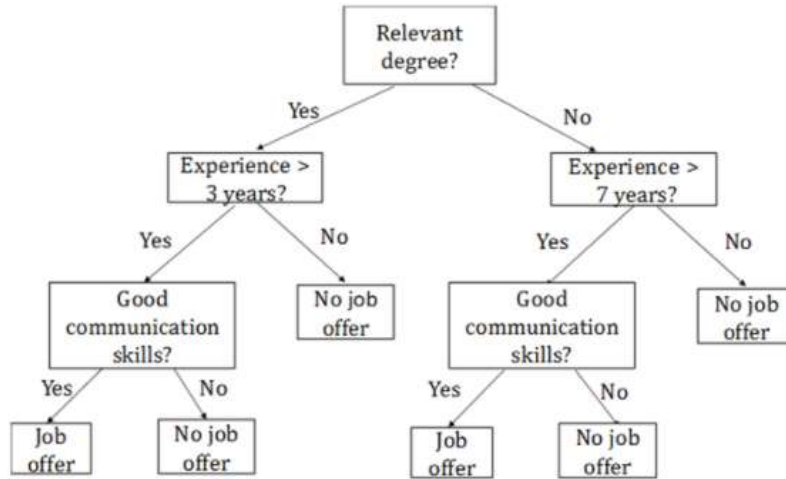
ב. מדד Entropy & Information Gain:

מדד נוסף לפיצול צמתי העץ מתבסס על האנטרופיה של העלים, בעזרתה ניתן לבחון את ה-information gain המקסימלי מכל פיצול. אנטרופיה באה למדוד את השגיאה של התפלגות המשתנה הנבחן מול משתנה המטרה. נניח וישנן n תוצאות אפשריות, כל אחת מהן בעלת הסתברות p_i , אז האנטרופיה מוגדרת באופן הבא:

$$H(x) = - \sum_{i=1}^n p_i \log p_i$$

בדומה למדד Gini, הפיצול האופטימלי נבחר על ידי המשתנה בעל מדד האנטרופיה הנמוך ביותר. אם כל התצפיות בעלה מסוים משויכות לאותו class, אזי מדד האנטרופיה יהיה 0. מאידך, כאשר בעלה מסוים יש התפלגות שווה בין 2-classים של המשתנה המוסבר, מדד האנטרופיה יהיה 1 (שזה הערך המקסימלי שמדד אנטרופיה יכול לקבל).

לעיל פירטנו שלב אחרי שלב את חישוב מדד Gini ל-Use-Case של חולי לב. כעת ניקח דוגמא אחרת פשוטה יותר הנוגעת לקבלת מועמד לתפקיד מסוים, כאשר מטרת העץ היא להחזיר "Yes" אם רוצים לקבל את המועמד, אחרת יוחזר "No".



איור 2.23 עץ סיווג עבור קבלת מועמד לתפקיד מסוים.

הדוגמה מתארת את אחד המאפיינים העיקריים של עצי החלטה – ההחלטה מתקבלת על ידי הסתכלות היררכית על הפרמטרים השונים, כאשר בכל פעם מתמקדים בפרמטר אחד ולא על כולם בבת אחת. תחילה, נלקח בחשבון המאפיין החשוב ביותר (תואר רלוונטי במקרה שלפנינו), לאחר מכן נלקחים שנות הניסיון, וכך הלאה.

נניח שהסיכוי בקרב כלל העובדים להתקבל לעבודה הוא 20%, והסיכוי לא להתקבל הוא 80%. במקרה זה האנטרופיה תחושב באופן הבא (הלוגריתם יכול להיות מחושב בכל בסיס, פה נלקח הבסיס הטבעי):

$$H = -0.2 \ln 0.2 - 0.8 \ln 0.8 = 0.5004$$

כעת נניח בנוסף ש-30% מהמועמדים בעלי תואר רלוונטי מקבלים הצעת עבודה ואילו מתוך אלה שאינם בעלי תואר רלוונטי רק 10% מקבלים הצעת עבודה. האנטרופיה עבור מועמד בעל תואר הינה:

$$H = -0.3 \ln 0.3 - 0.7 \ln 0.7 = 0.61$$

ועבור מועמד ללא תואר רלוונטי לתחום:

$$H = -0.1 \ln 0.1 - 0.9 \ln 0.9 = 0.32$$

נניח והמועמדים מתפלגים באופן שווה בין בעלי תואר לכאלה שאינם בעלי תואר, כלומר ל-50% מהמועמדים יש תואר רלוונטי ול-50% אין, הרי שתוחלת האנטרופיה במקרה זה הינה:

$$E_{H(x)} = 0.5 \times 0.61 + 0.5 \times 0.32 = 0.46$$

לאחר כל החישובים, נוכל לבחון את רמת ה-impurity שמתקבלת מהידיעה האם למועמד מסוים יש תואר רלוונטי או לא. כלומר, כמה הידיעה שלמועמד מסוים יש תואר רלוונטי מפחיתה מחוסר הוודאות שלו לקבל את המשרה. אם אי הוודאות נמדדת באמצעות מדד ה-Entropy, הרי שהרווח מהמידע הוא:

$$\text{Information gain} = 0.5004 - 0.4680 = 0.0324$$

הן עבור מועמדים בעלי תואר והן עבור כאלה ללא תואר, המשתנה שממקסם את הרווח מהמידע הצפוי (ירידה באנטרופיה הצפוי) הוא מספר שנות הניסיון. כאשר למועמד יש תואר רלוונטי, הסף עבור "שנות ניסיון" הממקסם את הרווח מהמידע הצפוי הוא 3 שנים. עבור הענף התואם למועמד שאין לו תואר רלוונטי, הסף של שנות ניסיון הממקסם את הרווח מהמידע הצפוי הוא 7 שנים. לפיכך, שני הענפים הבאים הם: "ניסיון < 7" ו-"ניסיון ≥ 7". באותו האופן בונים את יתר העץ.

Misclassification rate

לאחר בניית עץ הסיווג, יש לבדוק את רמת הדיוק שלו על דאטה חדש. בעץ גרסיה זה נעשה בעזרת מדד RSS, ובבעיות סיווג מקובל למדוד את ה-Misclassification rate. מדד זה בא לכמת את היחס בין כמות התצפיות שהמודל סיווג באופן שגוי לכמות הכוללת של התצפיות. במקרה זה פונקציית המחיר תהיה:

$$\mathcal{L}(\tilde{y}, y) = I\{\tilde{y} \neq y\}$$

פונקציית מחיר זו נקראת zero-one loss, כאשר תחת פונקציה זו חיזויים נכונים יקבלו ציון 0 ושגיאות יקבלו ציון 1, ללא תלות בגודל השגיאה. פונקציית ה misclassification rate נראית כך:

$$R(h) = \mathbb{E}[I\{h(x) \neq y\}]$$

סיכום

עץ החלטה (Decision Tree) הינו אלגוריתם לסיווג או לחיזוי ערכו של משתנה, כאשר המאפיינים מסודרים לפי סדר החשיבות. לצורך סיווג, קיימים שני מדדים אלטרנטיביים לאי וודאות: מדד אנטרופיה (Entropy) ומדד ג'יני (Gini). כאשר ערכו של משתנה מסוים נחזה, אי הוודאות נמדדת באמצעות RSS. חשיבותו של מאפיין הינו הרווח מהמידע הצפוי שלו (Expected Information Gain). הרווח מהמידע הצפוי נמדד על ידי הירידה באי הוודאות הצפויה אשר תתרחש כאשר יתקבל מידע אודות המאפיין.

במקרה של חלוקת **משתנה קטגוריאלי**, המידע המתקבל הינו על פי רוב אודות קטגוריה (Label) של המשתנה למשל: צבע מועדף (מכיל ערכים: אדום, ירוק, כחול וכו'). במקרה של **משתנה רציף** יש לקבוע ערך סף (Threshold) אחד (או יותר) המגדיר שני טווחים (או יותר) עבור ערכי המשתנה. ערכי סף אלו נקבעים באופן שממקסם את ה-information gain הצפוי.

אלגוריתם עץ ההחלטה קובע תחילה את צומת השורש (Root Node) האופטימלי של העץ באמצעות קריטריון "מקסום הרווח מהמידע" שהוגדר לעיל. לאחר מכן הוא ממשיך לעשות אותו הדבר עבור הצמתים העוקבים. הקצוות של הענפים הסופיים של העץ מכונים צמתי עלים (Leaf Nodes או Terminal Nodes).

במקרים בהם עץ ההחלטה משמש לסיווג, צמתי העלים כוללים בתוכם את ההסתברויות של כל אחת מהקטגוריות להיות הקטגוריה הנכונה. כאשר עץ ההחלטה משמש לחיזוי ערך נומרי לעומת זאת, אז צמתי העלים מספקים את ערך התוחלת של היעד. הגיאומטריה של העץ נקבעת באמצעות סט האימון (Training Set), אך הסטטיסטיקה שעוסקת ברמת הדיוק של העץ צריכה כמו תמיד בלמידת מכונה לבוא מתוך סט הבדיקה (Test Set) ולא רק מתוך סט האימון.

אחרית דבר:

מדדי RSS ומדד misclassification rate שהוצגו בפרק זה הינם רק חלק מהמדדים המקובלים. לכל מדד יתרונות וחסרונות, והמדד הרלוונטי יבחר בהתאם לסוג הנתונים והבעיה העסקית. נדון מעט בחוזקות ובחולשות של עצי החלטה:

יתרונות:

- בהשוואה לאלגוריתמים אחרים, עצי החלטה דורשים פחות השקעה בתהליך הכנת הנתונים (pre-processing).
- עץ החלטה לא דורש נרמול של הדאטה.
- ערכים חסרים בדאטה לא משפיעים על תהליך בניית העץ.
- עץ החלטה תואם לאופן שבו מרבית בני האדם חושבים על בעיה מסוימת והוא פשוט להסבר למי שאינם מומחים.
- אין שום דרישה שהקשר בין המשתנה המוסבר והמשתנים המסבירים יהיה ליניארי.
- העץ בוחר אוטומטית במשתנים הטובים ביותר על מנת לבצע את החיזוי.
- עץ החלטה רגיש פחות לתצפיות חריגות מאשר רגרסיה.

חסרונות:

- שינוי קטן בנתונים יכול לגרום לשינוי גדול במבנה עץ החלטה ולגרום לחוסר יציבות.
- לפעמים החישוב בעצי החלטה יכול להיות מורכב מאוד ביחס לאלגוריתמים אחרים.
- עצי החלטה לעיתים תכופות דורשים יותר זמן הרצה לאימון המודל, ומשך מדובר באלגוריתם "יקר" במשאבים.
- האלגוריתם של עץ החלטה אינו מספיק ליישום רגרסיה ולניבוי ערכים רציפים.
- עץ החלטה נוטה לעיתים תכופות לנטות ל-Overfitting.

2.2 Unsupervised Learning Algorithms

2.2.1 K-means

אלגוריתם K-means הינו אלגוריתם של למידה לא מונחית, בו מתבצעת תחזית על נתונים כאשר ה-label אינו נתון. אלגוריתם זה מתאים לבעיות של חלוקה לאשכולות (Clustering), ובנוסף יכול לשמש בשלב הצגת וניקוי הנתונים (EDA). עבור כל נקודה במדגם, המודל ממזער את סכום ריבוע המרחקים (WCSS) מכל מרכז אשכול (סנטרואיד - centroid), ולאחר תהליך של התכנסות – נקבעים האשכולות והסנטרואידים הסופיים. מספר האשכולות הנדרש הוא היפר-פרמטר שנקבע מראש. כמו כל האלגוריתם השייכים ללמידה הבלתי-מונחית, ב-K-means לא מתבצע אימון, ולמעשה התחזית מתבצעת על כל הדאטה הנתון.

סנטרואיד הוא מונח מתחום הגיאומטריה, והוא מתאר את הממוצע האריתמטי של כל הנקודות שמתפרסות על פני צורה כלשהי. באופן אינטואיטיבי ניתן לחשוב על סנטרואיד כנקודת איזון של צורה גיאומטרית כלשהי, כך שאם ננסה להניח צורה, משולש לדוגמא, באופן מאוזן, הסנטרואיד הוא הנקודה שבה המשולש יתאזן ולא ייפול לאחד הצדדים.

בפועל, סביר שהצורות איתן מתמודדים במציאות יותר מורכבות ממשולש. במצב כזה, הסנטרואיד יהיה הנקודה בה סכום המרחקים של כל נקודה באשכול מהסנטרואיד יהיה מינימלי. כלומר, המודל ימקם את מרכזו של כל אשכול כך שסכום המרחקים של כל הנקודות מהסנטרואיד יהיה נמוך ככל האפשר. למעשה, זוהי ההגדרה הבסיסית של K-means: אלגוריתם מבוסס סנטרואידים הממזער את סכום ריבוע המרחק של כל הנקודות באשכול. מדד זה נקרא WCSS, והוא מדד משמעותי ביותר בקרב אלגוריתמים שמבצעים חלוקה לאשכולות, K-means בפרט. הסיבה לחזקה במשוואה היא שאנו רוצים להגביר את ההשפעה של המרחק, מעין "עונש" לתצפיות רחוקות מהמרכז.

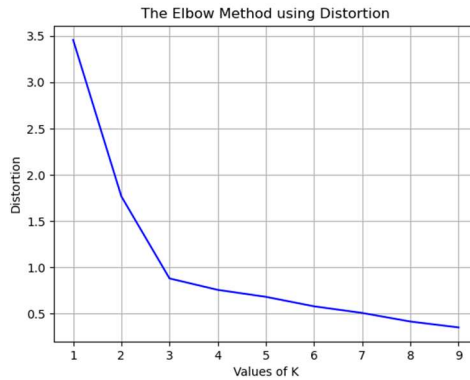
מדד WCSS הוא אחד הדרכים המקובלות ביותר להעריך את תוצאות החלוקה לאשכולות ב-K-means. היתרון של מדד זה הוא האפשרות לראות באופן כמותי את מידת ההצלחה של המודל, כלומר לקבל מספר ממשי שמכמת את הצלחת המודל. מנגד, WCSS הוא מספר ללא תחום מסוים והוא דורש פרשנות, כיוון שהערך והמשמעות שלו משתנים ממודל למודל. ערך מסוים יכול להיחשב תוצאה טובה במקרה מסוים, ובמקרה אחר זאת עשויה להיחשב תוצאה רעה מאוד. ניתן להשוות WCSS בין מודלים אך ורק כאשר יש להם את אותו מספר אשכולות ואותו מספר תצפיות. באופן פורמלי, ערך זה מחושב באופן הבא:

$$WCSS = \sum_{j=1}^k \sum_{i=1}^n \|x_i^{(j)} - c_j\|^2$$

כאשר K הוא מספר האשכולות, ו-n הוא מספר הנקודות במדגם.

ישנו trade-off בין השאיפה למזער את מדד ה-WCSS ובין מספר האשכולות הרצוי: ככל שמספר האשכולות גדול יותר, כך ה-WCSS יקטן. הדבר מתיישב עם ההיגיון – פיזור סנטרואידים רבים (כלומר, חלוקה ליותר אשכולות) על פני הנתונים יוביל לכך שבהכרח סכום המרחקים של התצפיות מהסנטרואידים יקטן או לא ישתנה. כיוון שתצפית משויכת לסנטרואיד הקרוב אליה ביותר, אם התווסף סנטרואיד שקרוב לנקודה מסוימת – ה-WCSS קטן. ואם הסנטרואיד רחוק מכל שאר הנקודות במדגם יותר מהסנטרואידים הקיימים – חלוקת התצפיות לאשכולות לא תשתנה, וערך ה-WCSS לא ישתנה.

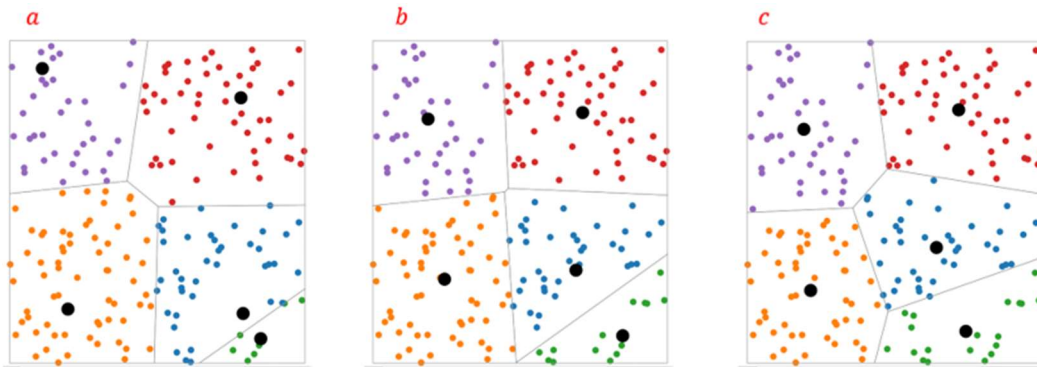
לכן מצד אחד, נרצה לבחור K גדול שימזער את ה-WCSS; מצד שני, הסיבה שהשתמשנו ב-K-means מלכתחילה היא בכדי לפשט את הנתונים למספר סביר של אשכולות, כזה שיאפשר לנו לערוך אנליזה נוחה. שיטת המרפק (Elbow method) היא טכניקה שמשמשת לפתרון סוגייה זו. הרעיון הוא לבחור את ה-K הקטן ביותר שממנו השיפור במדד ה-WCSS הוא מתון במידה סבירה. שיטה זו היא היוריסטית ואין דרך חד משמעית לקבוע שה-K הנבחר הוא האופטימלי. בדרך זו ניתן לשכנע מדוע ה-K שנבחר הוא הנכון, אך ההחלטה הסופית נתונה לשיקול דעתו של המשתמש.



איור 2.6 Elbow method – שיטה היוריסטית למציאת מספר האשכולות האופטימלי. בדוגמא זו ניתן לראות שבמעבר של K מ-2 ל-3 יש ירידה משמעותית בערך של ה-WCSS. המעבר מ-3 ל-4 לעומת זאת מוביל לשינוי זניח ב-WCSS (וכך גם במעברים הבאים). לכן ניתן להסיק שבמקרה כזה בחירה של $K = 3$ הינה בחירה טובה.

כאמור, האלגוריתם מחלק את הנתונים לאשכולות בדרך שממזערת את סך ריבועי המרחקים של כל תצפית ממרכז האשכול. באופן פורמלי האלגוריתם מתבצע ב-4 שלבים:

- א. **אתחול:** המודל מציב את הסנטרואידים באופן רנדומלי.
- ב. **שיוך:** כל תצפית משויכת לסנטרואיד הקרוב אליה ביותר.
- ג. **עדכון:** הסנטרואיד מוזז שכך שה-WCSS של המודל ימוזער.
- ד. חזרה על שלבים ב, ג עד אשר הסנטרואידים לא זזים לאחר העדכון, כלומר יש התכנסות.



איור 2.7 K-means 2.7. (a) אתחול 6 סנטרואידים באופן רנדומלי. (b) שיוך כל נקודה לסנטרואיד הקרוב ביותר אליה, ועדכון הסנטרואידים לפי מדד ה-WCSS. (c) חזרה על b עד להתכנסות.

K-means ידוע בכך שהוא אלגוריתם פשוט ומהיר. לרוב, הבחירה הראשונה בפתרון בעיות של חלוקה לאשכולות תהיה ב-K-means. עם זאת, לאלגוריתם ישנם גם חסרונות. ראשית, בחירת ה-K הנכון עשויה להוות אתגר במרבית המקרים. בנוסף, האלגוריתם רגיש מאוד לערכים קיצוניים (Outliers). אופן הפעולה של האלגוריתם מאפשר לו ליצור אשכולות רק בצורה של ספירות, והדבר אינו אופטימלי בחלק מן המקרים.

בעיה נוספת יכולה להתעורר בבחירת המיקום הראשוני של הסנטרואידים – כיוון שהבחירה היא רנדומלית, ניתן להיקלע להתכנסות במינימום מקומי שהוא אינו המינימום הגלובלי. כדי להתמודד עם בעיה זו ניתן להשתמש באלגוריתם K++. בשלב ראשון האלגוריתם בוחר למקם סנטרואיד אחד באופן רנדומלי. לכל תצפית, האלגוריתם מחשב את המרחק בין התצפית לסנטרואיד הקרוב אליה ביותר. לאחר מכן, תצפית רנדומלית נבחרת להיות הסנטרואיד החדש. התצפית נבחרת בהתאם להתפלגות משוקללת של המרחקים, כך שכל שתצפית יותר רחוקה – כך גובר הסיכוי שהיא תבחר. שני השלבים האחרונים נמשכים עד שנבחרו K סנטרואידים. כאשר כל הסנטרואידים מוקמו, מבצעים K-means עם דילוג על שלב האתחול (השלב בו ממקמים את הסנטרואידים). K++ מוביל להתכנסות מהירה יותר, ומוריד את הסיכוי להתכנס לאופטימום מקומי.

2.2.2 Mixture Models

אלגוריתם K-means מחלק n נקודות ל-K קבוצות על פי מרחק של כל נקודה ממרכז מסוים. בדומה ל-K-means גם אלגוריתם mixture model הוא אלגוריתם של clustering, אך במקום להסתכל על כל קבוצה של נקודות כשתיכות למרכז מסוים, המודל משייך נקודות להתפלגויות שונות. המודל מניח שכל קבוצה היא למעשה דגימות של התפלגות מסוימת, וכל הדאטה הוא ערבוב דגימות ממספר התפלגויות. הקושי בשיטה זה הוא האתחול של כל קבוצה – כיצד

ניתן לדעת על איזה דוגמאות לנסות ולמצוא התפלגות מסוימת? עקב בעיה זו, לעיתים משתמשים קודם באלגוריתם K-means על מנת לבצע חלוקה ראשונית לקבוצות, ולאחר מכן מנסים למצוא לכל קבוצה של נקודות התפלגות מסוימת.

ראשית נניח שיש k אשכולות, אזי נוכל לרשום את ההסתברות לכל אשכול:

$$p(y = i) = \alpha_i, i = 1, \dots, k$$

וכמובן לפי חוק ההסתברות השלמה מתקיים $\sum_i \alpha_i = 1$.

בנוסף נניח שכל אשכול מתפלג נורמלית עם פרמטרים $\theta_i = (\mu_i, \sigma_i)$, אזי נקודה השייכת לאשכול i מקיימת:

$$x|y = i \sim \mathcal{N}(\mu_i, \sigma_i), i = 1 \dots k$$

אם מגיעה נקודה חדשה ורוצים לשייך אותה לאחד האשכולות, אז צריך למעשה למצוא את האשכול i שעבורו הביטוי $p(y = i|x)$ הוא הכי גדול. לפי חוק בייס מתקיים:

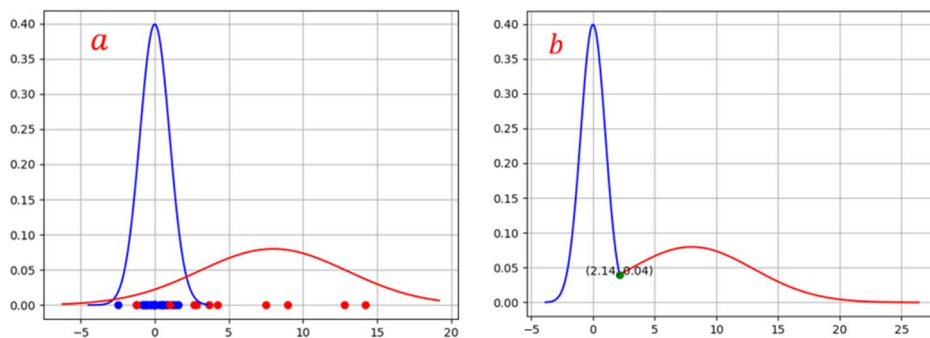
$$p(y = i|x) = \frac{p(y = i) \cdot p(x|y = i)}{p(x)}$$

המכנה למעשה נתון, כיוון שההתפלגות של כל אשכול ידועה ונותר לחשב את המכנה:

$$f(x) = f(x; \theta) = \sum_i p(y = i) f(x|y = i) = \sum_i \alpha_i \mathcal{N}(x; \mu_i, \sigma_i)$$

ובסך הכל:

$$p(y = i|x) = \frac{\alpha_i \cdot \mathcal{N}(x; \mu_i, \sigma_i)}{\sum_j \alpha_j \mathcal{N}(x; \mu_j, \sigma_j)}$$

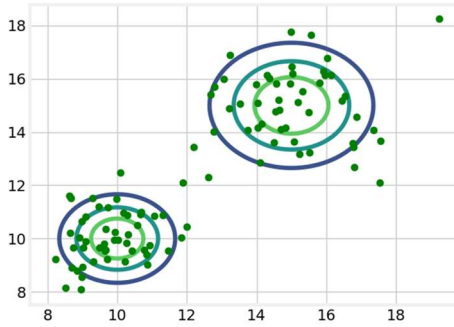


איור 2.8 (a) תערובת של שני גאוסיאנים בממד אחד: בשלב ראשון מחלקים את הנקודות לשני אשכולות ומתאימים לכל אשכול התפלגות מסוימת. במקרה זה אשכול אחד (מסומן בכחול) הותאם להתפלגות $\mathcal{N}(0,1)$, ואשכול אחד (מסומן באדום) הותאם להתפלגות $\mathcal{N}(8,5)$. נקודה חדשה x תסווג לאשכול הכחול אם $x < 2.14$, כיוון שבתחום זה $\mathcal{N}(0,1) > \mathcal{N}(8,5)$. באופן דומה, הנקודה x תסווג לאשכול האדום אם $x > 2.14$, כיוון שבתחום זה $\mathcal{N}(0,1) < \mathcal{N}(8,5)$.

כאמור, כדי לשייך נקודה חדשה x לאחד מהאשכולות, יש לבדוק את ערך ההתפלגות בנקודה החדשה. ההתפלגות שעבורה ההסתברות $p(x)$ היא הגדולה ביותר, היא זאת שאליה תהיה משויכת הנקודה. ההתפלגויות יכולות להיות בחד ממד, אך הן יכולות להיות גם בממד יותר גבוה. למשל אם מסתכלים על מישור, ניתן להתאים לכל אשכול התפלגות נורמלית דו-ממדית. במקרה ה- n ממדי, התפלגות נורמלית $X \sim \mathcal{N}(\mu, \Sigma)$ היא בעלת הצפיפות:

$$f_X(x_1, \dots, x_n) = \frac{1}{\sqrt{(2\pi)^n |\Sigma|}} e^{-\frac{1}{2}(x-\mu)^T \Sigma^{-1} (x-\mu)}$$

כאשר $|\Sigma|$ הוא הדטרמיננטה של מטריצת ה-covariance.



איור 2.9 תערובת של שני גאוסיאנים בדו-ממד: אשכול אחד מתאים לגאוסיאן עם וקטור תוחלות $\mu_1 = [10, 10]$ ומטריצת covariance: $\Sigma = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}$ והאשכול השני מתאים לגאוסיאן עם וקטור תוחלות $\mu_2 = [15, 15]$ ומטריצת covariance: $\Sigma = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$

כיוון שהאלגוריתם mixture model מספק התפלגויות, ניתן להשתמש בו כמודל גרטיבי, כלומר מודל שיודע לייצר דוגמאות חדשות. לאחר התאמת התפלגות לכל אשכול, ניתן לדגום מההתפלגויות השונות ובכך לקבל דוגמאות חדשות.

2.2.3 Expectation-maximization (EM)

אלגוריתם מקסום התוחלת הינו שיטה איטרטיבית למציאת הפרמטרים האופטימליים של התפלגויות שונות, במקרים בהם אין נוסחה סגורה למציאת הפרמטרים. נתבונן על מקרה של Mixture of Gaussians, ונניח שיש אשכול מסוים המתפלג נורמלית עם תוחלת ושונות $\theta = (\mu, \sigma)$, ומשיכות אליו n נקודות. כדי לחשב את ההתפלגות של אשכול זה ניתן להשתמש בלוג הנראות המרבית:

$$L(\theta|x_1, \dots, x_n) = \log \prod_{i=1}^n \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x_i-\mu)^2}{2\sigma^2}} = \sum_{i=1}^n \log \frac{1}{\sqrt{2\pi\sigma^2}} - \frac{(x_i - \mu)^2}{2\sigma^2}$$

כדי למצוא את הפרמטרים האופטימליים ניתן לגזור ולהשוות ל-0:

$$\frac{\partial L(\theta)}{\partial \mu} = \sum_{i=1}^n \frac{x_i - \mu}{\sigma^2} \rightarrow \mu_{MLE} = \frac{1}{n} \sum_{i=1}^n x_i$$

$$\frac{\partial L(\theta)}{\partial \sigma^2} = \frac{1}{2\sigma^2} \left(-n + \frac{1}{\sigma^2} \sum_{i=1}^n (x_i - \mu)^2 \right) \rightarrow \sigma_{MLE}^2 = \frac{1}{n} \sum_{i=1}^n (x_i - \mu)^2$$

כעת נניח ויש k אשכולות וכל אחד מתפלג נורמלית. כעת סט הפרמטרים אותם צריך להעריך הינו:

$$\theta = \{\mu_1, \dots, \mu_k, \sigma_1^2, \dots, \sigma_k^2, \alpha_1, \dots, \alpha_k\}$$

עבור מקרה זה, הלוג של פונקציית הנראות המרבית יהיה:

$$L(\theta|x_1, \dots, x_n) = \log \prod_{i=1}^n \sum_{j=1}^k \alpha_j \mathcal{N}(x_i, \mu_j, \sigma_j^2) = \sum_{i=1}^n \log \left(\sum_{j=1}^k \alpha_j \mathcal{N}(x_i, \mu_j, \sigma_j^2) \right)$$

אם נגזור ונשווה ל-0 נקבל בדומה למקרה הפשוט:

$$\sum_{i=1}^n \frac{1}{\sum_{j=1}^k \alpha_j \mathcal{N}(x_i, \mu_j, \sigma_j^2)} \alpha_j \mathcal{N}(x_i, \mu_j, \sigma_j^2) \frac{(x_i - \mu_j)}{\sigma_j^2} = 0$$

נוסחה זו אינה ניתנת לפתרון אנליטי, ולכן יש הכרח למצוא דרך אחרת בכדי לחשב את הפרמטרים האופטימליים של ההתפלגויות הרצויות. נתבונן בחלק מהביטוי שקיבלנו:

$$\frac{1}{\sum_{j=1}^k \alpha_j \mathcal{N}(x_i, \mu_j, \sigma_j^2)} \alpha_j \mathcal{N}(x_i, \mu_j, \sigma_j^2) = \frac{p(y_i = j) \cdot p(x_i|y = j)}{p(x_i)} = p(y_i = j|x_i) \equiv w_{ij}$$

קיבלנו למעשה את הפוסטריור y_i (האשכול אליו רוצים לשייך את x_i), אך הוא לא נתון אלא הוא חבוי. כדי לחשב את המבוקש ננחש ערך התחלתי ל- θ ובעזרתו נחשב את y_i , ואז בהינתן y_i נבצע עדכון לפרמטרים – נבחן מהו סט הפרמטרים שמסביר בצורה הטובה ביותר את האשכולות שהתקבלו בחישוב ה- y_i . באופן פורמלי שני השלבים מנוסחים כך:

E-step – בהינתן אוסף נקודות x וערך עבור הפרמטר θ נחשב את האשכול המתאים לכל נקודה, כלומר כל נקודה x_i תותאם לאשכול מסוים y_i . עבור כל הנקודות y_i נחשב תוחלת ובעזרתה נגדיר את הפונקציה $Q(\theta, \theta_0)$, כאשר θ הוא פרמטר חדש ו- θ_0 הוא סט הפרמטרים הנוכחי:

$$Q(\theta, \theta_0) = \sum_{i=1}^n \sum_{j=1}^k p(y_i = j | x_i; \theta_0) \log p(y_i = j, x_i; \theta) = \sum_{i=1}^n \sum_{j=1}^k w_{ij} \log p(y_i = j, x_i; \theta)$$

$$\sum_{i=1}^n \mathbb{E}_p(y_i | x_i; \theta_0) \log p(y_i = j, x_i; \theta)$$

M-step – מחשבים את הפרמטר θ שביא למקסימום את $Q(\theta, \theta_0)$ ואז מעדכנים את θ_0 ל- θ החדש:

$$\theta = \arg \max_{\theta} Q(\theta, \theta_0)$$

$$\theta_0 \leftarrow \theta$$

חוזרים על התהליך באופן איטרטיבי עד להתכנסות.

עבור Mixture of Gaussians נוכל לחשב באופן מפורש את הביטויים:

$$Q(\theta, \theta_0) = \sum_{i=1}^n \sum_{j=1}^k w_{ij} \log p(y_i = j, x_i; \theta)$$

$$= \sum_{i=1}^n \sum_{j=1}^k w_{ij} \log p(y_i = j; \theta) + \sum_{i=1}^n \sum_{j=1}^k w_{ij} \log p(x_i | y_i = j; \theta)$$

$$= \sum_{i=1}^n \sum_{j=1}^k w_{ij} \log \alpha_j + \sum_{i=1}^n \sum_{j=1}^k w_{ij} \log \mathcal{N}(\mu_j, \sigma_j^2)$$

$$= \sum_{i=1}^n \sum_{j=1}^k w_{ij} \log \alpha_j - \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^k w_{ij} \left(\log \sigma_j^2 + \frac{(x_i - \mu_j)^2}{\sigma_j^2} \right)$$

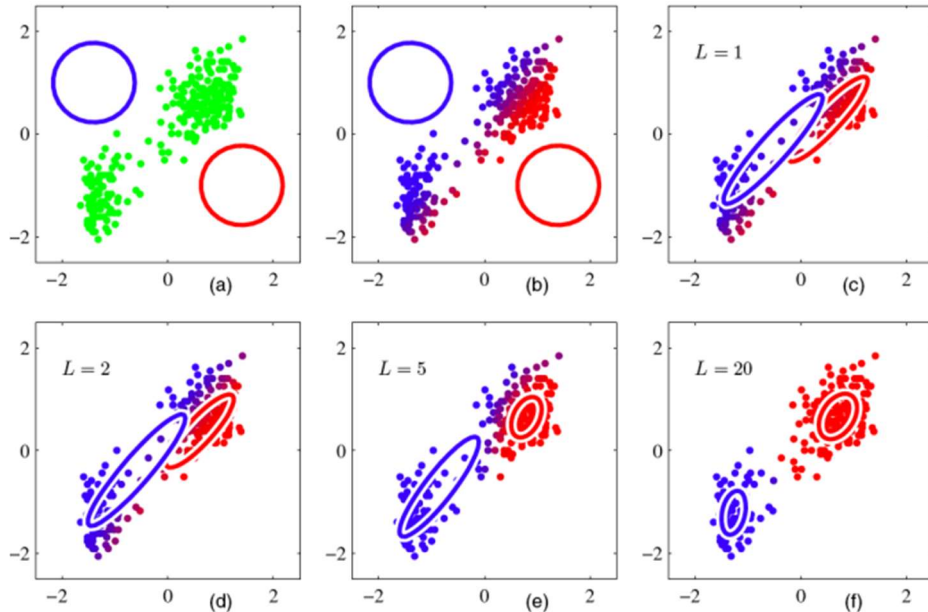
וכעת ניתן לגזור ולמצוא אופטימום:

$$\hat{\alpha}_j = \frac{1}{n} \sum_{i=1}^n w_{ij}$$

$$\hat{\mu}_j = \frac{\sum_{i=1}^n w_{ij} x_i}{\sum_{i=1}^n w_{ij}}$$

$$\hat{\sigma}_j^2 = \frac{\sum_{i=1}^n w_{ij} (x_i - \mu_j)^2}{\sum_{i=1}^n w_{ij}}$$

עבור התפלגויות שונות שאינן בהכרח נורמליות יש לחזור לביטוי של $Q(\theta, \theta_0)$ ולבצע עבורו את האלגוריתם.



איור 2.10 20 איטרציות של אלגוריתם EM. מתחילים מניחוש אקראי של ההתפלגויות, ובכל איטרציה יש שיפור כך שההתפלגויות מייצגות בצורה יותר טובה את הדאטה המקורי.

נוכיח שהאלגוריתם משתפר בכל איטרציה, כלומר שעבור כל (θ, θ_0) מתקיים: $\log p(x; \theta) \geq \log p(x; \theta_0)$:

$$\begin{aligned} \log p(x; \theta) &= \sum_y p(y|x; \theta_0) \log p(x; \theta) = \sum_y p(y|x; \theta_0) \frac{\log p(x, y; \theta)}{\log p(y|x; \theta)} \\ &= \sum_y p(y|x; \theta_0) (\log p(x, y; \theta) - \log p(y|x; \theta)) \\ &= \sum_y p(y|x; \theta_0) \log p(x, y; \theta) - p(y|x; \theta_0) \log p(y|x; \theta) \end{aligned}$$

נשים לב שהאיבר הראשון הוא בדיוק $Q(\theta, \theta_0)$. האיבר השני לפי הגדרה הוא האנטרופיה של ההתפלגות $p(x|y; \theta_0)$:

$$H(\theta, \theta_0) = - \sum_y p(y|x; \theta_0) \log p(y|x; \theta_0)$$

כעת עבור שני ערכים שונים של θ מתקיים:

$$\begin{aligned} \log p(x; \theta) - \log p(x; \theta_0) &= Q(\theta, \theta_0) + H(\theta, \theta_0) - Q(\theta_0, \theta_0) - H(\theta_0, \theta_0) \\ &= Q(\theta, \theta_0) - Q(\theta_0, \theta_0) + H(\theta, \theta_0) - H(\theta_0, \theta_0) \end{aligned}$$

לפי [אי-שוויון גיבס](#) מתקיים $H(\theta, \theta_0) \geq H(\theta_0, \theta_0)$, לכן:

$$\log p(x; \theta) - \log p(x; \theta_0) \geq Q(\theta, \theta_0) - Q(\theta_0, \theta_0)$$

ולכן עבור כל עדכון של θ שמביא לאופטימום את $Q(\theta, \theta_0)$, הביטוי $Q(\theta, \theta_0) - Q(\theta_0, \theta_0)$ יהיה חיובי וממילא יהיה שיפור ב- $\log p(x; \theta)$.

2.2.4 Hierarchical Clustering

אלגוריתם נוסף של למידה לא מונחית עבור חלוקת n נקודות ל- K אשכולות נקרא Hierarchical Clustering, והוא מחולק לשתי שיטות שונות:

agglomerative clustering – בשלב הראשוני מגדירים כל נקודה כאשכול, ואז בכל פעם מאחדים שני אשכולות ובכך מורידים את מספר האשכולות ב-1, עד שמגיעים ל- K אשכולות. האיחוד בכל שלב נעשה על ידי מציאת שני

האשכולות הקרובים ביותר זה לזה ואיחודם לאשכול אחד. ראשית יש לבחור מטריקה לחישוב מרחק בין שתי נקודות (למשל מרחק אוקלידי, מרחק מנהטן ועוד), ולאחר מכן לחשב מרחק בין האשכולות, כאשר יש מספר דרכים להגדיר את המרחק הזה, למשל:

complete-linkage clustering: $\max\{d(a, b) : a \in A, b \in B\}$.

single-linkage clustering: $\min\{d(a, b) : a \in A, b \in B\}$.

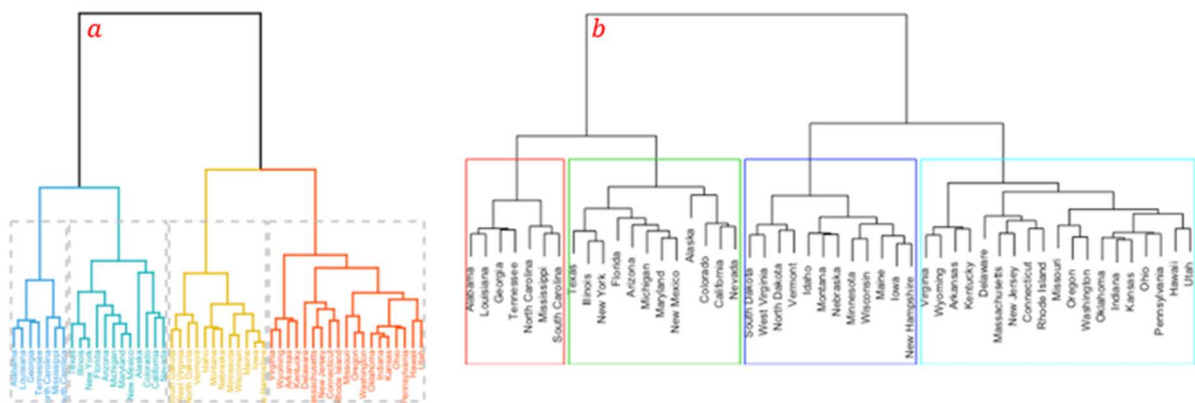
Unweighted average linkage clustering (UPGMA): $\frac{1}{|A| \cdot |B|} \sum_{a \in A} \sum_{b \in B} d(a, b)$.

Centroid linkage clustering (UPGMC): $\|c_s - c_t\|$ where c_s, c_t are centroids of clusters s, t , respectively.

עם התקדמות התהליך יש פחות אשכולות, כאשר האשכולות כבר לא מכילים נקודה אחת בלבד אלא הם הולכים וגדלים. שיטה זו מכונה "bottom-up" כיוון שבהתחלה כל נקודה הינה אשכול עצמאי ובכל צעד של האלגוריתם מספר האשכולות קטן באחד. במילים אחרות, האלגוריתם בונה את האשכולות ממצב שבו אין למעשה חלוקה לאשכולות למצב שבו נוצרים אשכולות ההולכים וגדלים.

divisive clustering – בשיטה זו מבצעים פעולה הפוכה – מסתכלים על כל הנקודות כאשכול אחד, ואז בכל שלב מבצעים חלוקה של אחד האשכולות לפי כלל חלוקה שנקבע מראש, עד שמגיעים ל-K אשכולות. כיוון שיש 2^n דרכים לחלק את המדגם, יש הכרח לנקוט בשיטות היוריסטיות כדי לקבוע את כלל החלוקה המתאים בכל שלב. שיטה מקובלת לביצוע החלוקה נקראת DIANA (DIvisive ANALysis Clustering), ולפיה בכל שלב בוחרים את האשכול בעל השונות הכי גדולה ומחלקים אותו לשניים. שיטה זו מכונה "top-down" כיוון שבהתחלה יש אשכול יחיד ובכל צעד של האלגוריתם מתווסף עוד אשכול.

את התצוגה של האלגוריתם ניתן להראות בצורה נוחה באמצעות dendrogram – דיאגרמה הבנויה כעץ המייצג קשרים בין קבוצות.



איור 2.11 תצוגה של Hierarchical Clustering בעזרת dendrogram (a) divisive – התחלה מאשכול יחיד ופיצול עד שמגיעים למספר האשכולות הרצוי (במקרה זה $K = 4$). (b) agglomerative – בהתחלה כל נקודה הינה אשכול, ובכל צעד מחברים שני אשכולות עד שמגיעים למספר האשכולות הרצוי.

2.2.5 Local Outlier Factor (LOF)

אלגוריתם Local Outlier Factor הינו אלגוריתם של למידה לא מונחית למציאת נקודות חריגות (Outliers). האלגוריתם מחשב לכל נקודה ערך הנקרא Local Outlier Factor (LOF), ועל פי ערך זה ניתן לקבוע עד כמה הנקודה היא חלק מקבוצה או לחילופין חריגה ויוצאת דופן.

בשלב ראשון בוחרים ערך k מסוים. עבור כל נקודה x_i , נסמן את k השכנים הקרובים ביותר שלה ב- $N_k(x_i)$. כעת נגדיר את k -distance של כל נקודה כמרחק שלה מהשכן הרחוק ביותר מבין השכנים ב- $N_k(x_i)$. אם למשל $k = 3$, אזי $N_k(x_i)$ הוא סט המכיל את שלושת השכנים הקרובים ביותר ל- x_i , וה- k -distance שלה הוא המרחק מהשכן השלישי הכי קרוב. חישוב המרחק בין שני שכנים נתון לבחירה – זה יכול להיות למשל מרחק אוקלידי, מרחק מנהטן ועוד. עבור בחירה של מרחק אוקלידי, ניתן להסתכל על k -distance כמעגל – הרדיוס של מעגל המינימלי המכיל את כל הנקודות השייכות ל- $N_k(x_i)$ הוא ה- k -distance.

לאחר חישוב ה- k -distance של כל נקודה, מחשבים לכל נקודה Local Reachability Density (LRD) באופן הבא:

$$LRD_k(x_i) = \frac{1}{\sum_{x_j \in N_k(x_i)} \frac{RD(x_i, x_j)}{k}}$$

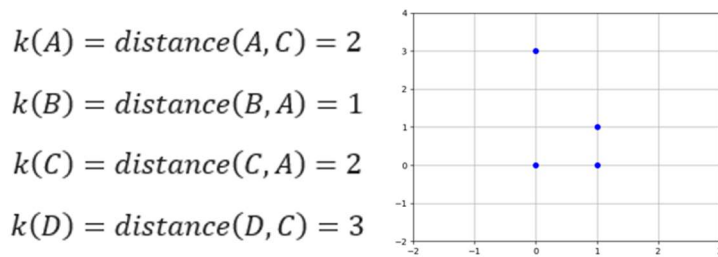
כאשר $RD(x_i, x_j) = \max(k - \text{distance of } x_i, \text{distance}(x_i, x_j))$. הגודל LRD מחשב את ההופכי של ממוצע המרחקים בין x_i לבין k השכנים הקרובים אליו. ככל שנקודה יותר קרובה ל- k השכנים שלה כך ה-LRD שלה גדול יותר, ו-LRD קטן משמעותו שהנקודה יחסית רחוקה מאשכול הקרוב אליה.

בשלב האחרון בוחנים עבור כל נקודה x_i את היחס בין ה-LRD שלה וה-LRD של $N_k(x_i)$. היחס הזה הוא ה-LOF, והוא מחושב באופן הבא:

$$LOF_k(x_i) = \frac{\sum_{x_j \in N_k(x_i)} LRD(x_j)}{k} \times \frac{1}{LRD(x_i)}$$

הביטוי הראשון במכפלה הוא ממוצע ה-LRD של k השכנים של נקודה x_i , ולאחר חישוב הממוצע מחלקים אותו ב-LRD של הנקודה x_i עצמה. אם הערכים קרובים, אז ה-LOF יהיה שווה בקירוב ל-1, ואם הנקודה x_i באמת לא שייכת לאשכול של נקודות, אז ה-LRD שלה יהיה נמוך משמעותית מהממוצע של ה-LRD של השכנים שלה, וממילא ה-LOF שלה יהיה גבוה. אם עבור נקודה x_i מתקבל $LOF \approx 1$, אז סביר שהיא חלק מאשכול מסוים.

כדי להמחיש את התהליך נסתכל על האוסף הבא: $\{A = (0,0), B = (1,0), C = (1,1), D = (0,3)\}$, ונקבע $k = 2$. נחשב את ה-k-distance של כל נקודה במונחים של מרחק מנהטן:



נחשב את ה-LRD:

$$LRD_2(A) = \frac{1}{\frac{RD(A, B) + RD(A, C)}{k}} = \frac{2}{1 + 2} = 0.667$$

$$LRD_2(B) = \frac{1}{\frac{RD(B, A) + RD(B, C)}{k}} = \frac{2}{2 + 2} = 0.5$$

$$LRD_2(C) = \frac{1}{\frac{RD(C, B) + RD(C, A)}{k}} = \frac{2}{1 + 2} = 0.667$$

$$LRD_2(D) = \frac{1}{\frac{RD(D, A) + RD(D, C)}{k}} = \frac{2}{3 + 3} = 0.334$$

ולבסוף נחשב את ה-LOF:

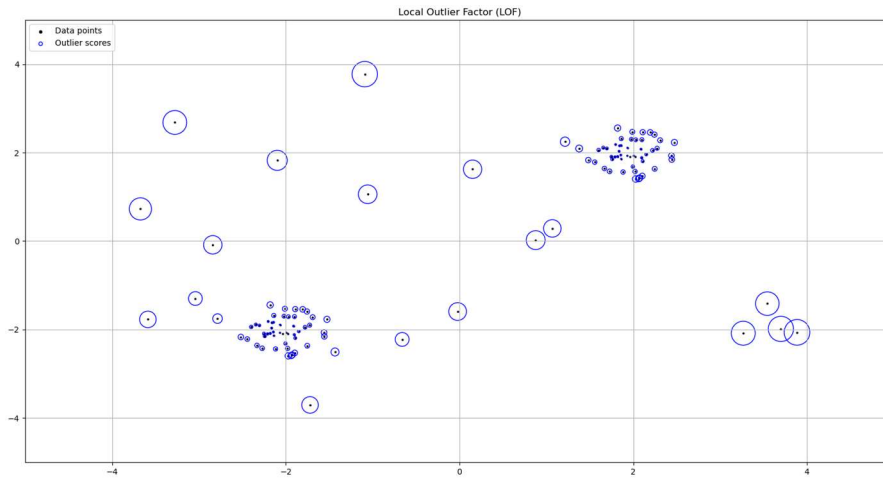
$$LOF_2(A) = \frac{LRD_2(B) + LRD_2(C)}{k} \times \frac{1}{LRD_2(A)} = 0.87$$

$$LOF_2(B) = \frac{LRD_2(A) + LRD_2(C)}{k} \times \frac{1}{LRD_2(B)} = 1.334$$

$$LOF_2(C) = \frac{LRD_2(B) + LRD_2(A)}{k} \times \frac{1}{LRD_2(C)} = 0.87$$

$$LOF_2(D) = \frac{LRD_2(A) + LRD_2(C)}{k} \times \frac{1}{LRD_2(D)} = 2$$

כיוון ש- $LOF_2(D) \gg 1$ באופן יחסי לשאר הנקודות, נסיק כי נקודה D היא outlier.



איור 2.12 Local Outlier Factor (LOF) – מציאת נקודות חריגות על ידי השוואת ערך ה- LRD של כל נקודה לממוצע ה- LRD של k השכנים שלה. ככל שה- LOF גדול יותר (העיגול הכחול), ככה הנקודה יותר רחוקה מאשכול של נקודות.

יש שני אתגרים מרכזיים בשימוש באלגוריתם זה – ראשית יש לבחור k מתאים, כאשר k יחסית קטן יהיה טוב עבור נקודות רועשות, אך יכול להיות בעייתי במקרים בהם יש הרבה מאוד נקודות הצמודות אחת לשנייה, ונקודה שמעט רחוקה מאוסף תזוהה כחריגה למרות שהיא באמת כן שייכת אליו. k גדול לעומת זאת יתגבר על בעיה זו, אך הוא לא יזהה נקודות חריגות שנמצאות בקירוב לאשכולות של נקודות. מלבד אתגר זה, יש צורך לתת פרשנות לתוצאות המתקבלות, ולהחליט על סף מסוים שך LOF , שהחל ממנו נקודה מסווגת כחריגה. LOF קטן מ-1 הוא בוודאי לא outlier אך עבור ערכי LOF גדולים מ-1 אין כלל חד משמעי עבור איזה ערך הנקודה היא outlier ועבור איזה ערך היא לא. כדי להתמודד עם אתגרים אלו הוצעו הרחבות לשיטה המקורית, כמו למשל שימוש בסטטיסטיקות שונות המורידות את התלות בבחירת הערך k (Local Outlier Probability – LoOP), או שיטות סטטיסטיות העוזרות לתת פרשנות לערכים המתקבלים (Interpreting and Unifying Outlier Scores).

2.3 Dimensionally Reduction

הורדת ממד (Dimensionality Reduction) הינה טרנספורמציה הממפה דאטה לממד נמוך יותר מהממד המקורי, כאשר נרצה שהורדת הממד לא תשנה באופן מהותי את מאפייני הדאטה המקורי. הורדת הממד של דאטה נתון נדרשת משתי סיבות עיקריות – אחת טכנית והשנייה מהותית:

- א. ביצוע חישובים ופעולות על מערכת מרובת ממדים הינה בעלת סיבוכיות גבוהה, ולעיתים אף בלתי ניתנת לביצוע.
- ב. הורדת הממד של הדאטה קשורה לניסיון להבין מהם המשתנים העיקריים ומהם המשתנים המשניים, הפחות חשובים להבנת הדאטה (אלו שפחות מאפיינים דוגמה נתונה ביחס לדוגמאות אחרות). לעיתים התחשבות במשתנים המשניים משפיעה לרעה על ביצועי המודל, למשל על ידי הוספת רעש ולא מידע. תופעה זו נקראת קללת הממדיות (curse of dimensionality). יתרון נוסף של הורדת ממד טמון בוויזואליזציה של המידע, כך שניתן להציגו על ידי 2 או 3 ממדים עיקריים, בעזרת גרף דו-ממדי או תלת-ממדי בהתאמה.

דוגמה למערכת מרובת ממדים יכולה להיות מדידת רמות חלבונים (פרוטאינים) של גנים (genes) המבוטאים בתא חי, כאשר כל ממד, או מאפיין (פיצ'ר), מתאים לגן אחר. באופן כללי, ייתכן ונמדדים בכל ניסוי מאות תאים, כאשר לכל תא נמדדות רמות ביטוי של מאות או אלפי גנים. כמות עצומה זו של מידע בממד גבוה (אלפי תאים ואלפי גנים בכל תא) מאתגרת את המחקר – הן מבחינת זיהוי המאפיינים, או רמות הגנים המבוטאים, הרלוונטיים והמשפיעים ביותר, והן מבחינת ניסיון למדל את הדאטה בצורה כמה שיותר פשוטה. במחקר משנת 2007 נלקחו 105 דגימות של תאי סרטן שד, כאשר לכל דגימה (או דוגמא) נמדדו רמות התבטאות של 27,648 גנים שונים. כמובן שלנתח את המידע בצורה הגולמית זו משימה בלתי אפשרית, ויש הכרח לבצע עליו מניפולציה כלשהיא כדי שיהיה אפשר לעבוד איתו.

ישנן שיטות מרובות לביצוע הורדת ממד לדאטה, כאשר ניתן לסווגן לשתי קטגוריות עיקריות: בחירת מאפיינים (feature selection), והטלת מאפיינים (features projection). השיטות השייכות לקטגוריה הראשונה מנסות לבחור את המאפיינים (המשתנים) המתארים באופן מספק את המידע הנתון. שיטות מהקטגוריה השנייה, המתוארות בפרק זה, נוקטות בגישה של הטלה, טרנספורמציה, של המאפיינים הקיימים לייצוג על ידי סט של מאפיינים חדשים במרחב אחר (ולרוב פשוט יותר). חשוב להדגיש שבשיטות מהקטגוריה הראשונה, מאפיינים פחות רלוונטיים מושמטים. בניגוד לכך, בשיטות שנדון בהן בפרק זה, המבוססות על הטלת המאפיינים, כל מאפיין חדש הינו טרנספורמציה של כל האחרים, ולא רק של חלקם. כך, המאפיינים החדשים מכילים, או לוקחים בחשבון, כל אחד מהמאפיינים הנמדדים המקוריים, ללא השמטה.

ניתן לבצע הטלת מאפיינים באמצעות טרנספורמציות ליניאריות או לא-ליניאריות. בפרק זה נעסוק בטרנספורמציה ליניארית אחת, הנקראת ניתוח גורמים ראשיים (Principle Component Analysis) ובשתי טרנספורמציות לא-ליניאריות (t-SNE, UMAP). נציין כי קיימות עוד טרנספורמציות, ליניאריות ולא-ליניאריות, המשמשות להורדת ממד של דאטה שאינן יוזכרו כאן.

2.3.1 Principal Components Analysis (PCA)

כפי שהוזכר לעיל, ניתוח גורמים ראשיים מבוסס על טרנספורמציה ליניארית של המאפיינים הקיימים. המטרה של אלגוריתם זה היא לבנות ייצוג חדש ובעל ממד נמוך יותר מאשר הממד המקורי של הדאטה, מתוך מטרה לשמר כמה שיותר את השונות של המאפיינים (features) של הדאטה המקורי. מדוע השונות כה משמעותית? ניקח למשל מאפיין המופיע בכל הדוגמאות בדאטה הנתון, ובכולן מאפיין זה הוא בעל אותו ערך. מאפיין כזה הוא בעל שונות אפס, ולמעשה הוא לא מכיל שום מידע על הדאטה ונרצה להיפטר ממנו. בדומה לכך, אם יש שני מאפיינים בעלי תלות ליניארית – אין טעם לשמור את שניהם כיוון שידעתם האחד מאפשרת לדעת גם את השני. במקרה זה הקורלציה בין שני המאפיינים היא 1, והשארית המאפיין השני לא תתרום "לשונות הכוללת" של ייצוג הדאטה. נראה בהמשך שייצוג שנבנה באמצעות PCA לא יכיל פיצ'רים כאלו. עוד נעיר כי האלגוריתם דואג לכך שהמקדמים של כל צירוף לינארי יהיו וקטורים בעלי אורך של 1, בכדי לא לנפח באופן מלאכותי את השונות של המאפיינים החדשים (לאחר הורדת ממד).

לאחר הקדמה זו, נפרט כיצד מחושבים המאפיינים החדשים, הנקראים למעשה "הגורמים העיקריים". הגורם הראשי הראשון (first principal component, PCA_1) הינו הצירוף הלינארי של המאפיינים המקוריים בעל השונות הגדולה ביותר. הגורם הראשי השני (second principle component, PCA_2) הוא גם צירוף לינארי של המאפיינים הנתונים, השונות שלו היא השנייה הגדולה ביותר, ובנוסף דורשים ש- PCA_2 יהיה אורתוגונלי ל- PCA_1 : $PCA_1 \perp PCA_2$. הגורם השלישי, הוא צירוף לינארי בעל השונות השלישית הגדולה ביותר, ומאונך לשני הגורמים הראשונים – $PCA_2 \perp PCA_3$ וגם $PCA_1 \perp PCA_3$, וכן הלאה כך שהגורם הראשי מסדר i , הוא בעל השונות ה- i -ית הגדולה ביותר תחת אילוץ של גורמים מאונכים – $PCA_i \perp PCA_j, \forall i < j$. הורדת הממד מתבצעת על ידי לקיחת מספר גורמים ראשיים ראשונים, והזנחת הקטנים ביותר.

לאחר שאפיינו את הגורמים הראשיים בהם אנו מעוניינים, עולה השאלה כיצד ניתן לבצע טרנספורמציה לינארית שבעזרתה ניתן למצוא את הגורמים הראשיים האלו. נניח שבידינו דאטה $\hat{X} \in \mathbb{R}^{M \times N}$, כלומר נתונות M דוגמאות שונות, שכל אחת מהן היא בעלת N מאפיינים [למשל, עבור הדוגמא של תאי סרטן השד, נתון מידע מ- $M = 105$ תאים שונים, כאשר עבור כל תא נמדדו רמות ביטוי של $N = 27,648$ גנים שונים]. מטריצה זו לעיתים נקראת ה-design matrix של הדאטה, ונסמן אותה באופן הבא:

$$\hat{X} = \begin{bmatrix} \vec{X}_1 \\ \vdots \\ \vec{X}_M \end{bmatrix} = [\vec{X}^1, \dots, \vec{X}^N] \in \mathbb{R}^{M \times N}$$

כאשר כל וקטור שורה, $\vec{X}_m, m \in \{1, \dots, M\}$ הינו נתוני המדידות של המאפיינים השונים בדוגמא מספר m , ובהתאמה, וקטור עמודה $\vec{X}^n, n \in \{1, \dots, N\}$ (שימו לב לשינוי סימון, אינדקס עליון עבור וקטורי עמודה), הינו נתוני המדידות של מאפיין מסוים על כל הדוגמאות. נניח שממוצע המדידות עבור כל מאפיין הוא אפס, זאת אומרת שלכל מאפיין n -י מתקיים:

$$mean(\vec{X}^n) = \sum_{m=1}^M X_{m,n} = \vec{0}$$

מכיוון שכל עמודה של המטריצה מסמלת ערכים של מאפיין מסוים במדידות שונות, סכום כל עמודה במטריצה \hat{X} הוא אפס. כעת, נרצה לבצע הטלה (טרנספורמציה) לינארית, זאת אומרת נכפיל את מטריצה \hat{X} במטריצת משקלים \hat{W} :

$$\hat{T} = \hat{X} \cdot \hat{W}$$

אם נסמן את השורה ה- m ית במטריצה \hat{T} על ידי \vec{T}_m , נקבל:

$$\vec{T}_m = \vec{X}_m \cdot \hat{W}$$

כאשר המטריצה $\hat{W} \in \mathbb{R}^{N \times K}$, כך ש- $\hat{T} \in \mathbb{R}^{M \times K}$. הטלה זו מביאה לכך שלאחר הטרנספורמציה נשארים רק K מאפיינים. כיוון שאנו מעוניינים בהורדת הממד, קרי הורדת מספר המאפיינים, נדרוש $K \leq N$.

את תהליך מציאת מטריצת המשקלים ניתן לנסח באופן פורמלי על ידי שלושה תנאים:

(1) כל עמודה של מטריצת המשקלים הינה בעלת נורמה השווה ל-1, כלומר:

$$\|\hat{W}^k\|^2 = \sum_{m=1}^M (W_{m,k})^2 = 1$$

(2) השונות עבור המאפיין ה- k , המוגדרת על ידי $s_k^2 = (\vec{T}^k)^T \vec{T}^k = \sum_{m=1}^M (T_{mk})^2$, מקיימת: $s_k^2 > s_{k+1}^2$.

(3) העמודות של \hat{W} אורתוגונליות זו לזו, זאת אומרת $\hat{W}^k \perp \hat{W}^{k'}$ לכל שתי עמודות k, k' .

נראה זאת באופן מפורש: נתחיל במציאת העמודה הראשונה \hat{W}^1 . נדרוש:

$$\hat{W}^1 = \operatorname{argmax}_{\|\hat{W}\|=1} (s_1^2)$$

זאת אומרת:

$$\begin{aligned} \hat{W}_1 &= \operatorname{argmax}_{\|\hat{W}\|=1} (s_1^2) = \operatorname{argmax}_{\|\hat{W}\|=1} \left((\vec{T}^1)^T \cdot \vec{T}^1 \right) = \operatorname{argmax}_{\|\hat{W}\|=1} \left((\hat{X} \hat{W}^1)^T \cdot \hat{X} \hat{W}^1 \right) \\ &= \operatorname{argmax}_{\|\hat{W}\|=1} \left((\hat{W}^1)^T (\hat{X})^T \cdot \hat{X} \hat{W}^1 \right) \end{aligned}$$

ולכן העמודה הראשונה של מטריצת המשקלים \hat{W}^1 נתונה על ידי:

$$\hat{W}^1 = \operatorname{argmax}_{\|\hat{W}\|=1} \left((\hat{W}^1)^T \cdot \hat{S} \cdot \hat{W}^1 \right)$$

כאשר מטריצה $\hat{S} \in \mathbb{R}^{(N \times N)}$ הינה מטריצת השונות המשותפת (covariance), המוגדרת על ידי $\hat{S} = (\hat{X})^T \cdot \hat{X}$. מטריצה זו, מסדר $N \times N$, מגדירה את השונות המשותפת בין צמד מאפיינים, כאשר $S_{v_1, v_2} = \sum_{m=1}^M X_{v_1, m} X_{m, v_2}$. ניתן לשים לב כי מטריצה זו סימטרית וממשית (ולכן הרמיטית).

לפי משפט המינימום-מקסימום (קורנט-פישר-ויל, מובא כנספח לפרק): עבור \hat{S} מטריצה הרמיטית ($S_{ij} = S_{ji}^*$), בעלת ערכים עצמיים $\lambda_1 \geq \dots \geq \lambda_K$, מתקיים:

$$\lambda_1 = \max_{\|\hat{W}\|=1} \left((\hat{W}^1)^T \cdot \hat{S} \cdot \hat{W}^1 \right)$$

כאשר \hat{W}^1 הינו הווקטור העצמי המתאים לערך העצמי המקסימלי של \hat{S} . λ_1 .

כעת, כדי למצוא את הווקטור העצמי הבא, \hat{W}^2 , והערך העצמי המתאים לו λ_2 , נגדיר מטריצה חדשה \hat{X} :

$$\hat{X} = \hat{X} - \hat{X} \hat{W}^1 (\hat{W}^1)^T$$

$$\begin{aligned}\hat{W}^2 &= \operatorname{argmax}_{\|\hat{W}\|=1} (s_2^2) = \operatorname{argmax}_{\|\hat{W}\|=1} \left((\vec{T}^2)^T \cdot \vec{T}^2 \right) \\ &= \operatorname{argmax}_{\|\hat{W}\|=1} \left((\hat{W}^2)^T (\tilde{X} + \hat{X} \hat{W}^1 (\hat{W}^1)^T)^T \cdot (\tilde{X} + \hat{X} \hat{W}^1 (\hat{W}^1)^T) \hat{W}^2 \right) \\ &= \operatorname{argmax}_{\|\hat{W}\|=1} \left((\hat{W}^2)^T (\tilde{X})^T \cdot (\tilde{X}) \hat{W}^2 \right)\end{aligned}$$

כאשר \hat{W}^2 הינו הווקטור העצמי המתאים לערך העצמי המקסימלי של $(\tilde{X})^T \tilde{X}$, ובעצם הוא הערך העצמי השני בגודלו עבור מטריצה $\hat{S} = \hat{X}^T \hat{X}$ (בחישוב השתמשנו בעובדה כי $(\hat{W}^1 \perp \hat{W}^2)$).

באופן כללי, כדי למצוא את \hat{W}^k והערך העצמי המתאים לו λ_k , נגדיר מטריצה חדשה $\tilde{\tilde{X}}$ באופן הבא:

$$\tilde{\tilde{X}} = \hat{X} - \sum_{i=1}^{k-1} \hat{X} \hat{W}^i (\hat{W}^i)^T$$

$$\hat{W}^k = \operatorname{argmax}_{\|\hat{W}\|=1} \left((\hat{W}^k)^T (\tilde{\tilde{X}})^T \cdot (\tilde{\tilde{X}}) \hat{W}^k \right)$$

כך ש λ_k הינו הערך העצמי המקסימלי ה- k -י של מטריצת השונות המשותפת $\hat{S} = \hat{X}^T \hat{X}$

ניתן גם, באופן פשוט יותר, להשתמש בשיטת פירוק לערכים סינגולריים (SVD), כאשר נמצא את הפירוק המתאים למטריצת השונות המשותפת:

$$\hat{S} = \hat{W} \cdot \hat{\Lambda} \cdot \hat{W}^T$$

כאשר $\hat{\Lambda}$ הינה מטריצה אלכסונית, ו- $\Lambda_{ii} = \lambda_i$ הינם הערכים העצמיים של \hat{S} המסודרים לפי גודלם מהגדול לקטן - $\lambda_1 \geq \lambda_2 \geq \dots \geq \lambda_M$, ומטריצה \hat{W} מורכבת מווקטורי עמודה שהינם הווקטורים העצמיים המתאימים לערכים העצמיים. הווקטורים העצמיים בהגדרתם הינם אורתוגונליים זה לזה, וכיון ש- $\|\hat{W}^k\| = 1$ לכל k , הם בעצם אורתונורמליים.

לסיכום, על מנת למצוא את הגורמים הראשיים עבור המידע הנתון \hat{X} :

א. "מרכז" את הנתונים כך שהמוצע עבור כל מאפיין הוא אפס: $\hat{X}^m = \hat{X}^m - \operatorname{mean}_n(\hat{X}^m)$

ב. מצא את מטריצת השונות המשותפת $\hat{S} = (\hat{X}^m)^T \hat{X}^m$

ג. מצא את $\hat{S} = \hat{W} \cdot \hat{\Lambda} \cdot \hat{W}^T$ - ה-SVD של \hat{S} .

ד. חשב $\hat{T} = \hat{X} \cdot \hat{W}$.

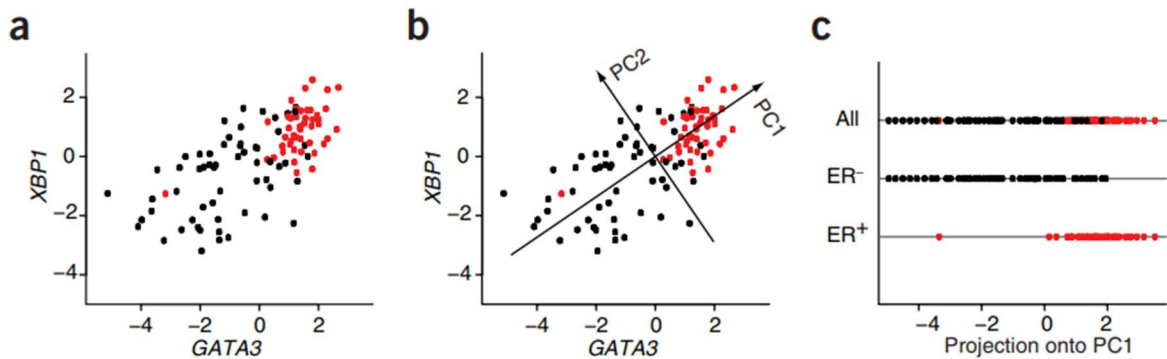
ה. הגורמים הראשיים נתונים על ידי וקטורי העמודה $\vec{W}^k \equiv PCA_k$, וההטלה של מדידה m למערכת המאפיינים החדשה נתונה על ידי $\vec{T}^m = \hat{X}^m \hat{W}$.

נציין שלשיטת הניתוח של גורמים ראשיים יש מספר מגבלות. ראשית, היא נותנת "משקל יתר" על מאפיינים שהשונות בהם גדולה, ללא קשר לחשיבותם, או ליחידות שבהן המאפיין נמדד (זאת אומרת לדוגמה שלגובה שנמדד בסנטימטרים יינתן "משקל" גבוה יותר מאשר גובה הנמדד במטרים). שנית, שיטת זו מניחה כי המדד החשוב הוא השונות המשותפת שהיא בעצם קורלציה לינארית בין שני משתנים, אולם ייתכן במערכות מסוימות שדווקא הקורלציה הלא-לינארית היא החשובה יותר. כמו כן, לעיתים "מרכז" המידע גורם לתוצאות לאבד ממשמעותן.

כדי להתגבר על המגבלות בשיטת ה-PCA שהצגנו לעיל פותחו שיטות נוספות או משלימות. לדוגמה, ניתן למזער את השפעת יחידות המידה על המאפיינים על ידי הפיכתם לחסרי יחידות. בנוסף, יש שיטות הלווקחות בחשבון קורלציות לא-לינאריות, לדוגמה שיטת kernel PCA, או שיטות להתמודדות עם בעיית המרכז על ידי דרישת משתנים חיוביים (NMF).

לצורך המחשה של תהליך חישוב הגורמים העיקריים ניתן שתי דוגמאות. ראשית נחזור לדוגמה שהזכרנו בתחילת פרק זה - מחקר שפורסם בשנת 2007 ובו נלקחו 105 דגימות של תאי סרטן שד, כאשר לכל דגימה נמדדו רמות התבטאות של 27,648 גנים שונים. לשם הדגמה, נשתמש בניתוח שפורסם כשנה לאחר מכן (ב-2008) על ידי אחד מעורכי המחקר המקורי. שם, החוקר מציג רמות של שני חלבונים; האחד בשם GATA3, והשני בשם XBP1, כאשר

דגימות תאי הסרטן מסווגות לפי סוג קולטני האסטרוגן שלהם (+ או -). כעת, על ידי "סיבוב" מערכת הצירים – בעזרת טרנספורמציה ליניארית PCA כפי שהוסבר לעיל – נמצא כי ניתן לסווג, ללא איבוד מידע רב, את מצב קולטני האסטרוגן בתאי סרטן השד על ידי הגורם הראשי הראשון PCA_1 , כפי שניתן לראות באיור. יש לשים לב שהגורם הראשי הראשון, PCA_1 , מכיל מידע משני החלבונים.



איור 2.13 (a) רמות ביטוי של שתי חלבונים GATA3 (ציר ה-X) ו-XBP1 (ציר ה-Y). קולטני אסטרוגן חיוביים או שלילים מסומנים באדום ושחור בהתאמה. (b) מציאת הגורמים הראשיים, וסיבוב מערכת הצירים. בהתאם לתיאוריה, ניתן להבחין כי השונות של המידע על גבי הציר החדש PCA_1 הינה מקסימלית. (c) הצגת תוצאות המדידה כפונקציה של PCA_1 בלבד. בגרף זה ניתן לראות בבירור כיצד הורדת הממד מסייעת למצוא הבחנה פשוטה (בממד אחד) בין קולטני האסטרוגן.

נביא בנוסף דוגמה חישובית מפורטת. נניח ונתון המערך הדו-ממדי הבא:

$$X = \begin{pmatrix} -0.5 & -0.4 \\ -0.4 & -0.1 \\ 0.1 & 0 \\ 0.3 & 0.3 \\ 0.5 & 0.2 \end{pmatrix}$$

מערך הנתונים מכיל 5 דוגמאות, ולכל דוגמה נמדדו שני מאפיינים ($M = 5, N = 2$). שורות המטריצה מציגות את המדידות השונות, והעמודות מייצגות את מאפיינים.

מערך זה כבר ממורכז, כלומר המאפיין הראשון מקיים:

$$mean_1(X^m) = \sum_{m=1}^5 X_{m1} = -0.5 - 0.4 + 0.1 + 0.3 + 0.5 = 0$$

ועבור המאפיין השני:

$$mean_2(X^m) = \sum_{m=1}^5 X_{m2} = -0.4 - 0.1 + 0 + 0.3 + 0.2 = 0$$

נחשב את מטריצת השונות המשותפת:

$$S = (\hat{X})^T \hat{X} = \begin{pmatrix} -0.5 & -0.4 & 0.1 & 0.3 & 0.5 \\ -0.4 & -0.1 & 0 & 0.3 & 0.2 \end{pmatrix} \begin{pmatrix} -0.5 & -0.4 \\ -0.4 & -0.1 \\ 0.1 & 0 \\ 0.3 & 0.3 \\ 0.5 & 0.2 \end{pmatrix}$$

$$= \begin{pmatrix} 0.5^2 + 0.4^2 + 0.1^2 + 0.3^2 + 0.5^2 & 0.5 \cdot 0.4 + 0.4 \cdot 0.1 + 0.1 \cdot 0 + 0.3^2 + 0.5 \cdot 0.2 \\ 0.5 \cdot 0.4 + 0.4 \cdot 0.1 + 0.1 \cdot 0 + 0.3^2 + 0.5 \cdot 0.2 & 0.4^2 + 0.1^2 + 0^2 + 0.3^2 + 0.2^2 \end{pmatrix}$$

$$= \begin{pmatrix} 0.76 & 0.43 \\ 0.43 & 0.3 \end{pmatrix}$$

על מנת ללכסן מטריצה זו, נמצא את הערכים העצמיים שלה המהווים שורשים של המשוואה האופיינית הבאה:

$$0 = |\hat{S} - \lambda I| = \begin{vmatrix} 0.76 - \lambda & 0.43 \\ 0.43 & 0.3 - \lambda \end{vmatrix} = (0.76 - \lambda)(0.3 - \lambda) - 0.43^2 \approx (\lambda - 1.02)(\lambda - 0.04)$$

כאשר לפולינום אופייני זה שני שורשים: $\lambda_1 \approx 1.02, \lambda_2 \approx 0.04$ (יש לשים לב שבחרנו $\lambda_1 > \lambda_2$). נמצא כעת את הווקטור העצמי המתאים לערך העצמי הגדול מבין השניים λ_1 . וקטור זה, המסומן על ידי \widehat{W}^1 , מקיים:

$$\widehat{S}\widehat{W}^1 = \lambda_1\widehat{W}^1$$

כך ש:

$$0 = (\widehat{S} - \lambda_1\widehat{I})\widehat{W}^1 \approx \begin{pmatrix} -0.83 & 0.107 \\ 0.107 & -0.94 \end{pmatrix} \begin{pmatrix} W_{11} \\ W_{21} \end{pmatrix} \Rightarrow \widehat{W}^1 \approx \begin{pmatrix} 0.86 \\ 0.51 \end{pmatrix}$$

הווקטור העצמי השני, \widehat{W}^2 , המתאים לערך העצמי $\lambda_2 \approx 0.04$, מחושב באותו אופן, ומתקבל: $\widehat{W}^2 \approx \begin{pmatrix} 0.51 \\ -0.86 \end{pmatrix}$.

כך שמטריצת המשקלים נתונה על ידי:

$$\widehat{W} = (\widehat{W}^1 \quad \widehat{W}^2) = \begin{pmatrix} 0.86 & 0.51 \\ 0.51 & -0.86 \end{pmatrix}$$

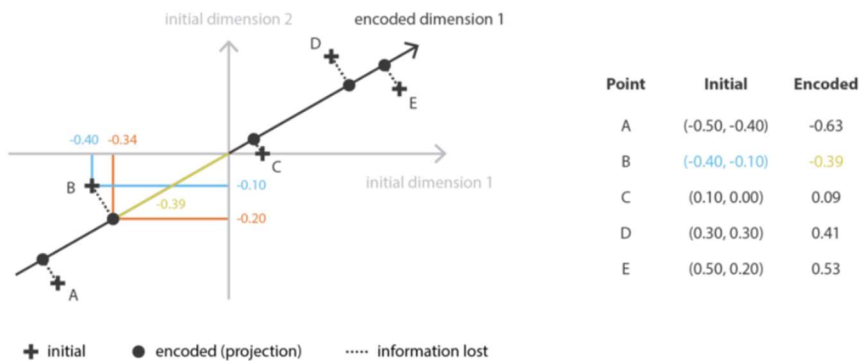
הטלת המדידות למערכת המאפיינים החדשה נתונה על ידי:

$$\widehat{T} = \widehat{X} \cdot \widehat{W}$$

לכן, המדידות של הגורם הראשי הראשון, נתונות על ידי

$$\widehat{T}^1 = \widehat{X} \cdot \widehat{W}^1 \approx \begin{pmatrix} -0.5 & -0.4 \\ -0.4 & -0.1 \\ 0.1 & 0 \\ 0.3 & 0.3 \\ 0.5 & 0.2 \end{pmatrix} \begin{pmatrix} 0.86 \\ 0.51 \end{pmatrix} = \begin{pmatrix} -0.5 \cdot 0.86 - 0.4 \cdot 0.51 \\ -0.4 \cdot 0.86 - 0.1 \cdot 0.51 \\ 0.1 \cdot 0.86 \\ 0.3 \cdot 0.86 + 0.51 \cdot 0.3 \\ 0.5 \cdot 0.86 + 0.2 \cdot 0.51 \end{pmatrix} \approx \begin{pmatrix} -0.63 \\ -0.39 \\ 0.09 \\ 0.41 \\ 0.53 \end{pmatrix}$$

נראה זאת באופן גרפי:



איור 2.14 הורדת ממד של דאטה דו-ממדי לממד אחד.

נספח: משפט המינימום-מקסימום (קורנט-פישר-ויל):

עבור $\widehat{S} \in \mathbb{R}^{M \times M}$ הרמיטית ($S_{ij} = S_{ji}^*$) מסדר עם ערכים עצמיים $\lambda_1 \geq \lambda_2 \geq \dots \geq \lambda_M$, מתקיים:

$$\lambda_m = \min_U \left\{ \max_{\substack{\widehat{x} \in U, \\ \|\widehat{x}\|=1}} \{ \widehat{x}^\dagger \widehat{S} \widehat{x} \mid x \in U, x \neq 0 \} \mid \dim(U) = M - m + 1 \right\}$$

$$= \min_U \left\{ \max_{\widehat{x} \in U} \left\{ \frac{\widehat{x}^\dagger \widehat{S} \widehat{x}}{\widehat{x}^\dagger \widehat{x}} \mid x \in U, x \neq 0 \right\} \mid \dim(U) = M - m + 1 \right\}$$

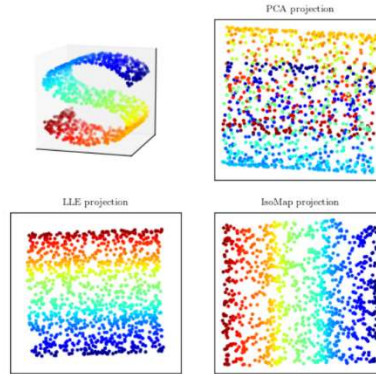
הערך העצמי המקסימלי מקיים:

$$\lambda_1 = \max_{\|\widehat{W}\|=1} ((\widehat{W}^1)^\dagger \cdot \widehat{S} \cdot \widehat{W}^1)$$

כאשר \hat{W}^{-1} , הינו הערך העצמי המתאים ל- λ_1 – ערך העצמי המקסימלי של \hat{S} .

2.3.2 t-distributed Stochastic Neighbors Embedding (t-SNE)

אלגוריתם הורדת הממד PCA פועל באופן לינארי, מה שמקל על תהליך החישוב שלו, אך מגביל את יכולות ההכללה שלו. אלגוריתם אחר, לא לינארי, נקראת t-SNE, והוא מנסה לקחת את הדאטה בממד גבוה ובאמצעות שימוש בכלים סטטיסטיים למפות אותו למערכת דו-ממדית או תלת-ממדית.



איר 2.15 הדגמה של מגבלת שיטת PCA ויתרונות השיטות הלא-לינאריות להורדת ממד. באיור השמאלי העליון מוצג דאטה תלת ממדי בעל מבנה בצורת האות S. בנוסף, לדאטה ישנו מאפיין נוסף המתבטא בצבעים שונים ובמעבר שלהם מכחול לאדום. טרנספורמציה לינארית PCA (סיבוב/מתיחה) אינה מצליחה לשמר את מבנה הדאטה כגרדיאנט צבע מאדום לכחול (ראה איור עליון ימני). אולם, שיטות לא לינאריות יכולות ומצליחות להוריד את הממדיות של הבעיה ולשמר את מבנה היריעה הטופולוגית (ראה שני איורים תחתונים).

לשם כך, נשתמש באותו מערך נתונים, $\vec{X} \in \mathbb{R}^{M \times N}$, כאשר M הוא מספר הדוגמאות, ו- N הוא מספר המאפיינים (או המשתנים/פיצורים). חשוב לשים לב כי כל מדידה מיוצגת על ידי וקטור שורה \vec{X}_m . הרעיון הכללי של השיטה הוא למפות את סט המדידות באופן כזה שמדידות דומות יותר, קרי מדידות "קרובות" יותר במרחב ה- N ממדי, יוצגו על ידי נקודות קרובות יותר במרחב חדש K -ממדי, כאשר לרוב $K \leq 3$. נסמן את המרחב המקורי ה- X ואת המרחב החדש ב- Y , כאשר בשני המרחבים המדידות מוצגות על ידי נקודות בגרף פיזור (scatter plot). המטריקה המשמשת למדידת דמיון (similarity) בין שתי נקודות במרחב המקורי X הינה הסתברותית. עבור שתי מדידות m_1, m_2 במרחב המקורי ה- N -ממדי, ההתפלגות הנורמלית המשותפת P_{m_1, m_2} הינה:

$$P_{m_1, m_2} = \frac{Z_1^{-1}}{2N} \exp\left(-\frac{\|\vec{X}_{m_1} - \vec{X}_{m_2}\|^2}{2\sigma_1^2}\right) + \frac{Z_2^{-1}}{2N} \exp\left(-\frac{\|\vec{X}_{m_1} - \vec{X}_{m_2}\|^2}{2\sigma_2^2}\right)$$

כאשר σ_i נקרא פרפלקסיות (perplexity) והוא פרמטר שנקבע מראש, ו- Z הינו קבוע הנורמליזציה, המוגדר על ידי $Z_i = \sum_{k \neq i} \exp\left(-\frac{\|\vec{X}_i - \vec{X}_k\|^2}{2\sigma_i^2}\right)$. עבור נקודות קרובות יותר, עבורן הביטוי $\|\vec{X}_{m_1} - \vec{X}_{m_2}\|$ קטן, ההסתברות שהנקודה \vec{X}_{m_1} שכנה של \vec{X}_{m_2} גדולה. לעומת זאת כאשר הנקודות רחוקות זו מזו, כלומר $\|\vec{X}_{m_1} - \vec{X}_{m_2}\|$ גדול, ההסתברות שהנקודה \vec{X}_{m_1} שכנה של \vec{X}_{m_2} קטנה מאוד עד אפסית.

כעת, כפי שהוזכר לעיל, נרצה למפות את סט המדידות: $\begin{bmatrix} \vec{X}_1 \\ \vdots \\ \vec{X}_M \end{bmatrix} \rightarrow \begin{bmatrix} \vec{Y}_1 \\ \vdots \\ \vec{Y}_M \end{bmatrix}$, כך שהממד של \vec{Y}_m הינו נמוך (2 או 3 ממדים).

בנוסף, נדרוש שנקודות דומות ("שכנות") במרחב X , ישארו שכנות לאחר המיפוי למרחב Y . מתברר שפונקציית ההסתברות המותנית, המתאימה לתיאור דמיון בין נקודות שכנות במרחב החדש Y , הינה התפלגות t, הנקראת גם התפלגות סטודנט עם דרגת חופש אחת (נדון ברעיון לבחור בפונקציות הסתברות אלו בהמשך). כך, נכמת את הדמיון בין m_1 לבין m_2 , על ידי ההסתברות המשותפת Q_{m_1, m_2} המוגדרת באופן הבא:

$$Q_{m_1, m_2} = 3^{-1} \frac{1}{1 + \|\vec{Y}_{m_1} - \vec{Y}_{m_2}\|^2}$$

כאשר $3 = \sum_{k \neq j} (1 + \|\vec{Y}_k - \vec{Y}_j\|^2)^{-1}$ הינו קבוע נורמליזציה.

המיפוי בין מרחב המקורי X לבין המרחב החדש \mathcal{Y} הוא מיטבי אם הוא "משמר" את השכנות של נקודות (מדידות) קרובות. לשם כך נגדיר את פונקציית המחיר על ידי Kullback-Leibler divergence, הבוחן מרחק בין שתי התפלגויות:

$$C = \mathcal{D}_{KL}(P|Q) \equiv \sum_{m_1} \sum_{m_2} P_{m_1, m_2} \log \left(\frac{P_{m_1, m_2}}{Q_{m_1, m_2}} \right)$$

נרצה למצוא את הווקטור $\begin{bmatrix} \vec{Y}_1 \\ \vdots \\ \vec{Y}_M \end{bmatrix}$ עבורו פונקציית המחיר מינימלית, ולשם כך נשתמש בגרדיאנט לפי \vec{Y}_{m_i} :

$$\begin{aligned} \frac{\delta C}{\delta \vec{Y}_{m_i}} &= \frac{\delta}{\delta \vec{Y}_{m_i}} \left[\sum_{m_1} P_{m_1, m_i} \log \left(\frac{P_{m_1, m_i}}{Q_{m_1, m_i}} \right) + \sum_{m_2} P_{m_1, m_2} \log \left(\frac{P_{m_i, m_2}}{Q_{m_i, m_2}} \right) \right] \\ &= 4 \sum_{m_1} (P_{m_1, m_i} - Q_{m_1, m_i}) (1 + \|\vec{Y}_{m_1} - \vec{Y}_{m_i}\|^2)^{-1} (\vec{Y}_{m_1} - \vec{Y}_{m_i}) \end{aligned}$$

חישוב המינימום באופן אנליטי לא תמיד אפשרי או לא תמיד יעיל, ולכן מקובל להשתמש בשיטת gradient descent, שהינה שיטה איטרטיבית למציאת נקודת המינימום של פונקציה (פירוט על שיטה זו ווריאציות שונות שלה מופיע בחלק 4.3.5). עבור הורדת הממד, חישוב המינימום בעזרת שיטה זו יעשה באופן הבא:

- א. אתחול: * נתון $X \in \mathbb{R}^{M \times N}$.
- * היפר-פרמטר לפונקציית הדמיון: בחירת השונות σ^2 .
- * בחירת שיטת אופטימיזציה (למשל: ADAM, SGD), והיפר פרמטרים כגון: קצב הלמידה ϵ , מומנטום $\alpha(t)$, גודל ה-batch וכו'.
- ב. חשב את P_{m_1, m_2} .
- ג. אתחל את המיפוי $\mathcal{Y}^{(0)} = \{\vec{Y}_1, \vec{Y}_2, \dots, \vec{Y}_M\} \sim N(0, s\hat{I}_M)$ [ז"א בחר את הערכים ההתחלתיים לפי התפלגות גאוסיאנית עם ממוצע 0 וסטיית תקן s (s נבחר להיות קטן, נניח $s = 10^{-4}$). \hat{I}_M מטריצת יחידה].
- ד. עבור איטרציה t :

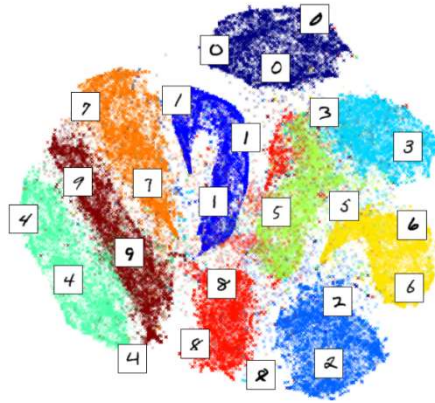
* חשב את Q_{m_1, m_2} עבור ה-batch הנבחר.

* חשב את הגרדיאנט של פונקציית המחיר $\frac{\delta C}{\delta \mathcal{Y}}$.

* עדכן: $\mathcal{Y}^{(t)} = \mathcal{Y}^{(t-1)} + \eta \frac{\delta C}{\delta \mathcal{Y}} + \alpha(t)[\mathcal{Y}^{(t-1)} - \mathcal{Y}^{(t-2)}]$.

נעיר כי בשפות תכנות רבות, האלגוריתם עצמו כבר מוגדר על ידי פונקציות מובנות, ויש רק להגדיר את הפרמטרים הדרושים.

במאמר המקורי שהציג את השיטה הובאה דוגמה של שימוש באלגוריתם עבור הטלה של הספרות 0 עד 9, המיוצגות על ידי תמונות בממד גבוה $\mathbb{R}^{28 \times 28}$, למרחב דו ממדי. בדוגמה זו נלקחו 6,000 תמונות של ספרות ומיפו אותן למרחב דו-ממדי. במרחב זה ניתן לראות בבירור כיצד כל תמונה מופתה לאזור אחר, כיוון שבפועל נוצרו עשרה אשכולות שונים, המובחנים בצורה ברורה אחד מהשני. בייצוג הדו-ממדי אין משמעות לצירים, כיוון שבאלגוריתם זה יש חשיבות רק למרחק היחסי בין הנקודות.



איור 2.15 הצגה (ויזואליזציה) דו-ממדית של מערך נתונים עבור כתב-יד של ספרות (MNIST) על ידי שיטת t-SNE. כל דוגמא \vec{X}_m מאופיינת על ידי $28 \times 28 = 784$ ערכים (פיקסלים בגווי אפור) ומסוגלת להיות ספרה בין 0 ל-9. באיור מוצגות 6000 נקודות (מדידות) כאלו, כאשר צבעים שונים מייצגים ספרות שונות. מלבד ההבחנה בין הספרות, ניתן לראות שספרות דומות קרובות זו לזו גם במרחב החדש (למשל הספרה 1 קרובה לספרה 7, שבתורה קרובה לספרה 9).

כאמור, פונקציית הדמיון בין שתי נקודות במרחב המקורי הינה הפילוג הנורמלי המשותף של שתי הנקודות, ואילו במרחב החדש פונקציית הדמיון הינה התפלגות t . שתי הערות חשובות על בחירות אלו:

א. סימטריה:

פונקציית הדמיון הגאומטרית בין שתי נקודות במרחב \mathcal{X} הינה פונקציה סימטרית, כלומר $P_{m_1, m_2} = P_{m_2, m_1}$. אולם ניתן להגדיר גם פונקציית דמיון א-סימטרית, המבוססת על התפלגות מותנית (במקום התפלגות משותפת). הפונקציה המותנית נתונה על ידי:

$$P_{m_1|m_2} = Z_2^{-1} \exp\left(-\frac{\|\vec{X}_{m_1} - \vec{X}_{m_2}\|^2}{2\sigma_2^2}\right)$$

כך ש:

$$P_{m_1, m_2} = \frac{P_{m_1|m_2} + P_{m_2|m_1}}{2N}$$

ב. בחירת פונקציית הדמיון במקום פונקציית t :

באלגוריתם שתואר, פונקציית הדמיון בין שתי נקודות במרחב ה- \mathcal{Y} נתונה על ידי התפלגות t . ניתן להגדיר גם פונקציה אחרת, למשל את פונקציית דמיון גאומטרית עבור שתי מדידות במרחב \mathcal{Y} . שיטה זו נקראת SNE, והגרדיאנט של פונקציית המחיר במקרה זה נתונה על ידי:

$$\frac{\delta C}{\delta \vec{Y}_{m_i}} = 4 \sum_{m_1} (P_{m_1, m_i} - Q_{m_1, m_i}) (\vec{Y}_{m_1} - \vec{Y}_{m_i})$$

אולם, פונקציית דמיון גאומטרית במרחב \mathcal{Y} יכולה לגרום לכך שנקודות לא מאוד קרובות במרחב \mathcal{X} , ימופו לנקודות קרובות במרחב \mathcal{Y} . זה קורה מכיוון שהתפלגות גאומטרית במרחב \mathcal{Y} גורמת לאטרקטור (משיכה) יחסית חזק בין שתי נקודות, גם במקרים בהם הנקודות אינן מאוד קרובות. לעומת זאת, כאשר פונקציית הדמיון הינה התפלגות סטודנט t , שהינה התפלגות עם זנב כבד יותר, שתי נקודות שאינן מאוד קרובות ימופו בצורה ראויה למרחב \mathcal{Y} כך שאינן "נמשכות" או מתקרבות זו לזו. שיטה אחרת, הנקראת UNI-SNE, מציעה להשתמש בהתפלגות אחידה, אך גם לה חסרון דומה ל-SNE, כאשר שתי נקודות לא מאוד דומות זו לזו, אינן "דוחות" אחת את השנייה.

לשיטת t-SNE יש שלוש מגבלות עיקריות

א. הורדת ממד: השיטה משמשת לוויזואליזציה של מידע מממד גבוה בדו-ממד או תלת-ממד. אולם, באופן עקרוני, ייתכן ונרצה להוריד את הממד לא לשם הצגתו, אלא לצרכים אחרים, כאשר הממד החדש הינו גדול מ-3. ייתכן ובממד גבוה פונקציית התפלגות סטודנט t עם דרגת חופש אחת, אשר לה משקל גבוה יחסית

- במרחקים גבוהים, לא תשמר את המבנה של המידע המקורי. לכן, כאשר נרצה להוריד לממד גבוה מ-3, פונקציית התפלגות t עם יותר מדרגת חופש אחת מתאימות יותר.
- ב. קללת הממדיות: t-SNE מבוססת על מאפיינים מקומיים בין נקודות. השיטה, המבוססת על מטריקת מרחק אוקלידית, וכך מניחה לינאריות מקומית על גבי היריעה המתמטית בה מתקיימות הנקודות. אולם, במערך נתונים בו הממד הפנימי גבוה, שיטת t-SNE עלולה להיכשל כיוון שהנחת הלינאריות לא מתקיימת. למרות שישנן מספר שיטות למזער תופעה זו, עדיין, בהגדרה, כאשר הממד הפנימי גבוה, לא ניתן להוריד ממד כך שמבנה המידע ישמר באופן מלא.
- ג. פונקציית מחיר לא קמורה: הרבה שיטות למידה מבוססות על פונקציית הפסד קמורה, כך שתיאורטית מציאת אופטימיזציה (יחידה) לפונקציה זו אפשרית תמיד. אולם, בשיטת t-SNE, פונקציית המחיר אינה קמורה, והפתרון המתקבל על ידי האופטימיזציה משתנה בהתאם לפרמטרים הנבחרים.

2.3.3 Uniform Manifold Approximation (UMAP)

UMAP הינה שיטה לא לינארית נוספת עבור הורדת ממד, המבוססת על קירוב של יריעה טופולוגית (manifold). השיטה מתבססת על מספר הנחות בסיס: ראשית, מניחים כי הנקודות/הדוגמאות מתפלגות באופן אחיד על גבי יריעה טופולוגית כלשהיא. בנוסף, מניחים כי יריעה זו קשירה באופן מקומי. מטרת קירוב זה היא לשמר את המבנה של הדוגמאות על גבי היריעה. נעיר כי הוכחה ריגורוזית של השיטה מערבת מתמטיקה מתקדמת (מעבר לנלמד בתואר הראשון), אך ניתן להבין את הרעיון העיקרי גם ללא ההוכחה המדויקת, ועל כן נציג ראשית את הרעיון העיקרי, ולאחר מכן את עיקרי הבסיס המתמטי העומד מאחוריו. השיטה מחולקת לשני שלבים עיקריים – בשלב הראשון מחשבים משקל של קשת בין כל שתי דוגמאות במרחב, ולאחר מכן מבצעים את הורדת הממד על סמך משקלי הקשתות:

- ראשית, נרצה לתאר את היריעה הטופולוגית בה נמצאות כל הדוגמאות במקורי, \hat{X} , על ידי גרף ממושקל-קשיר. על מנת ליצור גרף זה יש לבצע את השלבים הבאים עבור כל דוגמא X_{m_i} (שהיא כאמור נקודה במרחב המקורי ה- N -ממדי):

- מציאת את k השכנים הקרובים ביותר על ידי שימוש במרחק האוקלידי $d_{i,j} \equiv d(X_{m_i}, X_{m_j})$.
- חישוב המרחק לנקודה הקרובה ביותר – נסמן אותו ב- ρ_i .
- חישוב הגודל σ_i על ידי פתרון המשוואה:

$$\log_2(k) = \sum_{j=1}^k \exp\left(-\frac{\max\{0, d_{ij} - \rho_i\}}{\sigma_i}\right)$$

המשקל (הסימטרי) של קשת בין שתי נקודות - X_{m_i} ו- X_{m_j} הינו:

$$P_{ij} \equiv P_{i|j} + P_{j|i} - P_{i|j} \cdot P_{j|i}$$

כאשר:

$$P_{i|j} = \exp\left(-\frac{\max\{0, d_{ij} - \rho_i\}}{\sigma_i}\right)$$

כלומר: עבור X_{m_j} שאינו בקבוצת k השכנים הקרובים של X_{m_i} מתקיים: $P_{i|j} = 0$, ואם שתי הנקודות הן כן

$$P_{i|j} = \exp\left(\frac{\rho_i - d_{ij}}{\sigma_i}\right).$$

- על מנת לבצע הורדת ממד, נרצה לייצר הצגה של הגרף הממושקל בממד נמוך ממרחב \hat{Y} , כאשר לגרף החדש יש "מבנה דומה" לגרף המקורי. לשם כך, נגדיר פונקציית דמיון באופן הבא:

$$q_{i,j} \equiv \left(1 + a \|\vec{Y}_{m_1} - \vec{Y}_{m_2}\|^{2b}\right)^{-1}$$

כאשר פונקציית המחיר הינה:

$$C = \sum_{i \neq j} P_{ij} \log\left(\frac{P_{ij}}{q_{ij}}\right) + (1 - P_{ij}) \log\left(\frac{1 - P_{ij}}{1 - q_{ij}}\right)$$

נעיר כי מספר השכנים הקרובים k , והמרחק המינימלי האפקטיבי (הנקבע על ידי a, b) הינם היפר-פרמטרים הנבחרים על ידי המשתמש.

הבדלים עיקריים בין שיטת t-SNE לבין UMAP:

- א. בשיטת t-SNE המרחקים האוקלידיים מחושבים כל זוג נקודות. בשיטת UMAP, נלקחים בחשבון רק קבוצת שכנים קרובים.
- ב. הסימטריזציה בשיטת t-SNE נתונה על ידי: $P_{i,j} = \frac{P_{ij} + P_{ji}}{2N}$. לעומת זאת, הסימטריזציה בשיטת UMAP נתונה על ידי $P_{ij} \equiv P_{ij} + P_{j|i} - P_{i|j} \cdot P_{j|i}$.
- ג. פונקציית הדמיון במרחב המקורי בשיטת t-SNE הינה גאוסיאנית. לעומת זאת, בשיטת UMAP פונקציית הדמיון במרחב \mathcal{X} הינה אקספוננט.
- ד. עבור UMAP פונקציית המשקל (הדמיון) במרחב החדש \mathcal{Y} הינה $q_{i,j} \equiv \left(1 + a \left\| \vec{Y}_{m_i} - \vec{Y}_{m_j} \right\|^{2b}\right)^{-1}$ כאשר $a = b = 1$ ובנוסף פונקציה זו מנורמלת (מוכפלת במקדם 3^{-1}), אנו מקבלים את פונקציית המשקל של מרחב \mathcal{Y} עבור t-SNE.
- ה. לפונקציית המחיר (או הפוטנציאל) עבור שיטת UMAP נוסף איבר מהצורה $(1 - P_{ij}) \log\left(\frac{1 - P_{ij}}{1 - q_{ij}}\right)$. איבר זה מוריד את סיכוי קבלת שתי נקודות קרובות מאוד במרחב החדש (זאת אומרת כאשר $q_{ij} \sim 1$) כאשר הם אינן קרובות במרחב המקורי.

תיאור מתמטי של השיטה:

נניח שהיריעה הטופולוגית בה המידע (data) נמצא, אינה ידועה מראש, אולם בכל זאת נרצה להגדיר מרחקים גאודזיים על גבי יריעה זו. בנוסף, נניח כי הדוגמאות מפוזרות באופן אחיד על גבי יריעה זו. באופן כללי, הגדרת מטריקת רימן (מטריקה, פונקציה מסויימת, שמאפשרת לנו הגדרת מרחקים, ראה דוגמא למטה), תלויה בסביבת הנקודה. המטרה הראשונית שלנו היא להגדיר את היריעה הטופולוגית בה הנקודות נמצאות על ידי גרף ממושקל קשיר.

כעת, נניח שבסביבת נקודה p המטריקה g_p אלכסונית וקבועה. לכן, ניתן להגדיר את המרחק בין נקודה p לנקודה q שנמצאת בסביבת p ע"י המטריקה הקיימת במרחב n ממדי.

מתוך ההנחה שהנקודות מפוזרות בצורה אחידה על גבי היריעה, מתקיים הדרישה הבאה. רדיוס של כדור סביב נקודה X_i (הדוגמא ה- i -ית של מערך הנתונים הרב ממדי) כך שהכדור מכיל k שכנים קרובים ל- X_i , הוא קבוע ואינו משתנה בין הנקודות.

אם כך, בידנו משפחת מטריקות מקומיות, אחת לכל דוגמא i -ית, שעל ידה ניתן להגדיר את המרחקים ל- k שכנים קרובים. נרצה "לאחד" את משפחת המטריקות המקומיות הללו, למרחב גלובלי. לשם כך, נשתמש ב-fuzzy simplicial sets [לצורך הדיון קבוצות אלו מכילות סימפלקס – מבנה מסוים של גרף/רשת - כך שלכל איבר יש דרגת שייכות לקבוצה]. נשים לב כי על ידי איחוד הקבוצות הללו, יצרנו גרף ממושקל-קשיר. עבור גרף זה, כל נקודה מקושרת אך ורק לקבוצת k השכנים הקרובים לה, כאשר עבור שכנים קרובים ביותר, משקל הקשת המחברת גבוה יותר ממשקל קשת עבור שכנים רחוקים יותר. נשים לב כי חישוב המשקלים ו"קישור" עבור שכנים קרובים בלבד מאפשרת לשחזר את מבנה היריעה הטופולוגית ביתר דיוק (ראה איור).

נניח, בדומה לפרק הקודם, כי מערך נתונים מקורי מסומן על ידי $\hat{X} \in \mathbb{R}^{M \times N}$, כאשר M הוא מספר הדוגמאות, ו- N הוא מספר המאפיינים (או המשתנים). חשוב לשים לב כי כל מדידה מיוצגת על ידי וקטור שורה \vec{X}_m . נרצה למפות מדידות אלו ל $\hat{Y} \in \mathbb{R}^{M \times D}$, $\{\vec{Y}_1, \dots, \vec{Y}_M\}$, כך שמדידה i במרחב המקורי \mathcal{X} : \vec{X}_i מיוצגת על ידי וקטור חדש \vec{Y}_i במרחב החדש \mathcal{Y} , כך שמתקיים $D \ll N$, זאת אומרת הממד של הווקטור החדש הרבה יותר קטן מהממד המקורי.

כעת, על מנת להשוות בין ייצוג ב-fuzzy simplicial sets עבור \hat{X} ו- \hat{Y} , נשתמש ב-cross entropy. נניח שדרגת השייכות של איבר a בקבוצה A נתון על ידי פונקציה $\mu: A \rightarrow [0,1]$. הסימון המקובל עבור fuzzy set זה הוא (A, μ) . פונקציית cross entropy עבור (A, μ) ו (A, ν) מוגדרת להיות:

$$C[(A, \mu), (A, \nu)] \equiv \sum_{a \in A} \left[\mu(a) \log \left(\frac{\mu(a)}{\nu(a)} \right) + (1 - \mu(a)) \log \left(\frac{1 - \mu(a)}{1 - \nu(a)} \right) \right]$$

עבור קבוצת המדידות במרחב המקורי: $\hat{X} = \{\vec{X}_1, \vec{X}_2, \dots, \vec{X}_M\}$, נסמן מטריקה $d: \hat{X} \times \hat{X} \rightarrow \mathbb{R}^+ \cup \{0\}$ המכמתת את אי-הדמיון בין שתי מדידות; קרי מדידות דומות יציגו מרחק קטן ביניהן, לעומת מדידות שאינן דומות בהן המרחק המוגדר על ידי המטריקה ימצא להיות גדול. נגדיר תת-קבוצה של k השכנים הקרובים, על פי המטריקה d , של

מדידה \vec{X}_i , ונסמנה באופן הבא: $\{\vec{X}_{i_1}, \vec{X}_{i_2}, \dots, \vec{X}_{i_k}\}$. עבור כל מדידה \vec{X}_i , נחפש את המרחק הקטן ביותר בינה לבין השכנים שלה, זאת אומרת:

$$\rho_i = \min \left\{ d(\vec{X}_i, \vec{X}_{i_j}) : 1 \leq j \leq k, d(\vec{X}_i, \vec{X}_{i_j}) > 0 \right\}$$

בנוסף, נחפש σ_i כך שמתקיים:

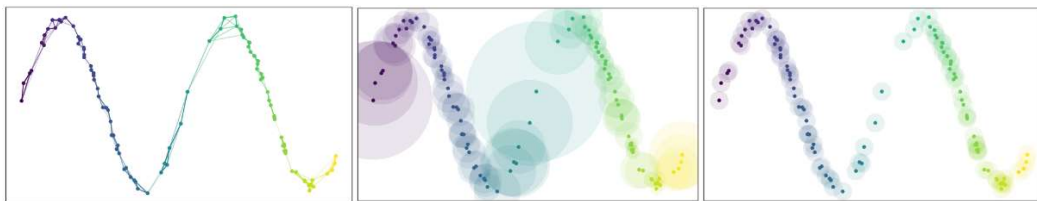
$$\sum_{j=1}^k \exp \left[-\frac{\max(0, d(\vec{X}_i, \vec{X}_{i_j}) - \rho_i)}{\sigma_i} \right] = \log_2 k$$

כעת, נוכל להגדיר גרף מכוון וממושקל (weighted directed graph) $\bar{G} = (V, E, w)$ כך שקבוצת הקודקודים V הינה קבוצת המדידות \vec{X} , קבוצת הקשתות (edges) נקבעת על ידי k השכנים הקרובים לכל נקודת מדידה:

$$E = \left\{ (\vec{X}_i, \vec{X}_{i_j}) : 1 \leq i \leq M, 1 \leq j \leq k \right\}$$

והמשקל עבור כל קשת מוגדר על ידי:

$$w[(\vec{X}_i, \vec{X}_{i_j})] = \sum_{j=1}^k \exp \left[-\frac{\max(0, d(\vec{X}_i, \vec{X}_{i_j}) - \rho_i)}{\sigma_i} \right] \equiv p_{i|j}$$



איר 2.16 לפנינו נקודות הנמצאות על גבי יריעה טופולוגית שהינה בקירוב בצורת סינוס. באיור מימין ניתן לראות כי הנקודות אינן מפוזרות באופן אחיד על גבי יריעה זו, לכן אם נקשר רק נקודות שכונות (הנמצאות בתוך סביבת כל נקודה) נקבל גרף / יריעה שאינה קשירה. אם לעומת זאת נדרוש שסביבת הנקודה המכילה k שכנים קרובים הינו כדור בעל רדיוס מסוים, נמצא כי רדיוס כדור זה אינו קבוע בין הנקודות (ראה איר אמצעי). לשם כך, ההנחה הבסיסית של שטית $UMAP$ הינה שהנקודות מפולגות בצורה אחידה על גבי היריעה. כעת, לאחר שהנחנו התפלגות אחידה של הנקודות על גבי היריעה, נוכל להגדיר את משקל הקשתות בין שכנים קרובים (ראה איר שמאלי). נשים לב כי באופן זה מבנה היריעה המתואר על ידי הגרף הקשיר אכן נראה בצורת סינוס כנדרש.

נספח מתמטי:

בהינתן מטריקת רימן $g_{\mu,\nu}(p)$ עבור סביבת נקודה p על גבי היריעה, האורך של העקום $\gamma(x_i, x_f)$ נתון ע"י:

$$\|\gamma(x_i, x_f)\|_g \equiv \int_{x_i}^{x_f} \sqrt{\sum_{\mu} \sum_{\nu} g_{\mu,\nu}(p) dx_{\mu} dx_{\nu}} = \int_{x_i}^{x_f} \sqrt{\sum_{\mu} \sum_{\nu} g_{\mu,\nu}(p) \frac{\partial x_{\mu}}{\partial t} \frac{\partial x_{\nu}}{\partial t}} dt$$

כאשר הסכימה הכפולה (μ, ν) על וקטורי הבסיס של העקומה. נבחר את העקומה שנותנת מרחק מינימלי.

דוגמאות:

דוגמא ליריעה דו-ממדית הינו משטח אוקלידי דו-ממדי \mathbb{R}^2 . מטריקת רימן עבור יריעה זו, אינו תלוי בנקודה p , ומוגדר על ידי: $g_{\mu,\nu} = \delta_{\mu,\nu}$. עבור עקום במישור הדו-ממדי: $(x, y) = (x, f(x))$. פרמטריזציה של עקום זה נתונה על ידי $\gamma(t) = (t, f(t))$. אורך עקום זה נתון על ידי:

$$\|\gamma(x_i, x_f)\| \equiv \int_{x_i}^{x_f} \sqrt{\sum_{\mu=\{x,y\}} \sum_{\nu=\{x,y\}} \delta_{\mu,\nu} \frac{\partial x_{\mu}}{\partial t} \frac{\partial x_{\nu}}{\partial t}} dt = \int_{x_i}^{x_f} \sqrt{\left(\frac{\partial x}{\partial t}\right)^2 + \left(\frac{\partial f(t)}{\partial t}\right)^2} dt$$

עקומה $\gamma(t)$ שנותנת מרחק מינימלי בין נקודות מוגדרת על ידי $\gamma(t) = (t, mt + n)$ כאשר $m, n \in \mathbb{R}$, כך ש:

$$\int_{x_i}^{x_f} \sqrt{1 + m^2} dt = \sqrt{1 + m^2}(x_f - x_i) = \sqrt{(x_f - x_i)^2 + m^2(x_f - x_i)^2} = \sqrt{(x_f - x_i)^2 + (y_f - y_i)^2}$$

גם ספירה דו-ממדית (מסומנת לעיתים על ידי S^2 , לדוגמא גלובוס) מהווה יריעה דו ממדית. שם, מטריקת רימן מוגדרת על ידי

$$g = \begin{pmatrix} 1 & 0 \\ 0 & \sin^2 \theta \end{pmatrix}. \text{ לכן, עבור עקום על גבי גלובוס; } \gamma(t) = (\theta, \phi) \text{ נקבל:}$$

$$\|\gamma\| \equiv \int_{x_i}^{x_f} \sqrt{\left(\frac{\partial \theta}{\partial t}\right)^2 + \sin^2 \theta \left(\frac{\partial \phi}{\partial t}\right)^2} dt$$

2.4 Ensemble Learning

2.4.1 Introduction to Ensemble Learning

נניח ויש בידינו אוסף נתונים מסוים, ורוצים לבנות מודל המנתח את הנתונים האלו שמתבסס על אלגוריתם מסוים. כמעט תמיד, המודל לא יהיה מדויק במאה אחוז, והוא יהיה בעל שונות או בעל הטיה. ניתן להשתמש במכלול (Ensemble) של מודלים שונים המבוססים על אותו אלגוריתם רצוי, ובכך לקבל מודל משוקלל בעל שונות/הטיה נמוכים יותר מאשר מודל שמתבסס על אותו אלגוריתם אך נבנה באופן פשוט.

בכדי להבין את יותר טוב את החשיבות של שימוש ב-ensembles, יש להרחיב על ה-Trade off בין שונות המודל להטיה שבו. מודל אופטימלי יתאפיין בשונות נמוכה ובהטיה נמוכה. כלומר, השוני בין התחזיות לא יהיה מהותי, ובממוצע התחזית תהיה קרובה מאוד לערך האמיתי. מודל כזה יהיה מודל אמין, ונוכל לבסס עליו את צעדנו. למרבה הצער, מודל שכזה לרוב אינו אפשרי. סוג אחר של מודל יהיה המודל הגרוע, ההפוך למודל האופטימלי. זהו מודל עם שונות גבוהה והטיה גדולה. מודל שכזה יציג טווח רחב של תחזיות על אותם נתונים, ובממוצע יהיה רחוק מאוד מהערך האמיתי. מודל זה כלל אינו שימושי.

בפועל, המודלים במציאות ינועו לאורך שני קצוות: מודלים עם שונות גבוהה והטיה נמוכה, ומודלים עם שונות נמוכה והטיה גבוהה. הזיהוי של המיקום שלנו לאורך ציר זה קריטי, כיוון שהוא מאפשר לנו לבחור את דרך ההתמודדות הטובה ביותר. יש מספר משפחות של model ensembles, ושני העיקריים שבהם נקראים Bagging and Boosting. כאשר ניתקל במודלים עם שונות גבוהה, כלומר מודל הסובל מ-Overfitting, לרוב נרצה להשתמש באנסמבל מסוג Bagging על-מנת להוריד את השונות במודל הסופי. אלגוריתם מסוג Boosting יטפל במקרה השני, בו ההטיה גבוהה והשונות נמוכה.

2.4.2 Bootstrap aggregating (Bagging)

Bagging היא משפחת אלגוריתמים אשר פועלת כ-ensemble, כלומר – מספר אלגוריתמים שפועלים ביחד, על-מנת להגיע לתוצאה משופרת. כאמור, אלגוריתמים מסוג bagging נועדו להגדיל את יציבות המודל והעלאת הדיוק שלו, זאת תוך הורדת השונות והימנעות מ-overfitting. Bagging מורכב ממספר רב של אלגוריתמים, המכונים "לומדים חלשים" (Weak learners), כאשר כל אחד מהם מבצע למידה ותחזית על חלק מן הנתונים, מתוך מטרה להגיע לתוצאה איכותית. אלגוריתם bagging הינו שיטה נפוצה ופשוטה יחסית לשיפור ביצועים, אם כי הוא עשוי להיות יקרה מבחינה חישובית.

מודל "פשוט" יקבל את הנתונים, יתאמן עליהם ויבצע תחזית על נתונים חדשים. זהו תהליך הלמידה והמבחן אשר ידוע לנו ממודלים כגון עץ החלטה (Decision Tree), גרסיה לינארית וכו'. כפי שהוסבר לעיל, מצב כזה עשוי להוביל להתאמת יתר של המודל לנתוני האימון, דבר שעשוי להוביל למודל בעל שונות גבוהה. בכדי להתמודד עם בעיה זו, אלגוריתמים מסוג bagging מפרקים את התהליך לשני שלבים: Bootstrapping and Aggregating.

בשלב ה-Bootstrapping, יוצרים מהנתונים המקוריים קבוצות חדשות, כאשר כל קבוצה נוצרת על ידי דגימה (עם חזרות) של איברים מהקבוצה המקורית, באופן כזה שגודל כל קבוצה חדשה הוא בגודל של הדאטה המקורי. בשלב השני, ה-Aggregating, הקבוצות החדשות נכנסות כקלט ל"לומדים חלשים", אלגוריתמים פשוטים יותר, אשר עובדים במקביל על תחזית, כלומר, יוצרים מודל נפרד לכל קבוצה של נתונים. בשלב הסופי, יתבצע איחוד של כל המודלים על מנת ליצור מודל משוקלל בעל שונות קטנה יותר מאשר מודל המסתמך על הדאטה המקורי כפי שהוא.

אופן חיבור המודלים המתבצע ב-bagging מבוסס על אותו רעיון של K-NN, כאשר מודלים אלו יכולים לשמש הן למטרות סיווג והן למטרות רגרסיה. כאמור לעיל (בפרק 2.1.3), באלגוריתם השכן הקרוב כל "שכן" העיד על התווית שלו, ולאחר הכרעת הרוב נקבעה התווית של התצפית החדשה. במקרה שבו נספור את תדירות כל התוויות השכנות, התווית הנבחרת תהיה של התצפית הנפוצה ביותר; נעשה זאת כאשר K-NN יעבוד כמסווג. במקרים בהם K-NN יעבוד כרגרסיה, יתבצע ממוצע של כל התוויות השכנות, וזאת גם תהיה התחזית. כאשר bagging יעבוד כמסווג, כל

weak learner יבצע תחזית, והתוויית השכיחה ביותר תהיה התוצאה של האנסמבל (-הכרעת הרוב). כאשר bagging עבוד כרגרסיה, כל מודל יבצע תחזית, אבל התוצאה של האנסמבל תהיה הממוצע של כל המודלים.

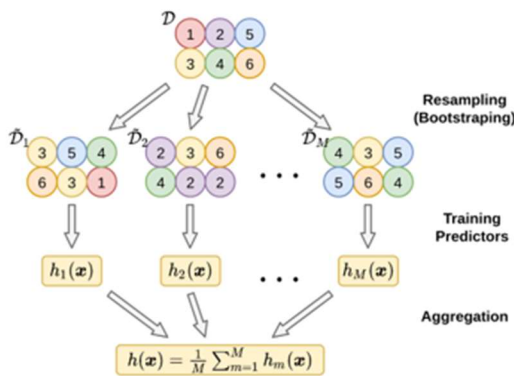
באופן פורמלי, עבור דאטה $X \in \mathbb{R}^{n \times d}$, ניצור M קבוצות חדשות באותו גודל של הדאטה המקורי – $\{X_m \in \mathbb{R}^{n \times d}\}_{m=1}^M$, ועבור כל קבוצה X_m נבנה מודל $c_m(x)$. עבור בעיות סיווג ההחלטה תתקבל על פי הצבעת הרוב:

$$C(x) = \text{majority}(\{c_1(x), \dots, c_M(x)\})$$

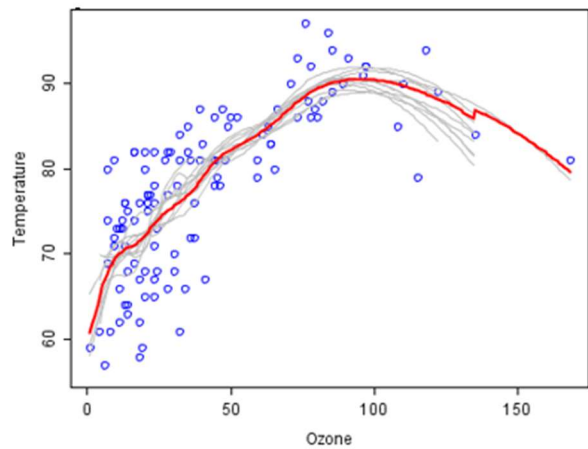
ועבור בעיות רגרסיה ההחלטה תתבצע בעזרת מיצוע כל המודלים:

$$C(x) = \frac{1}{M} \sum_{m=1}^M c_m(x)$$

a



b



איר 2.17 a) Bagging: בשלב ראשון יוצרים הרבה מחלקות שונות שנוצרות מהדאטה המקורי (Bootstrapping), לאחר מכן בונים מודל המתאים לכל מחלקה (Aggregating), ולבסוף יוצרים מודל יחיד המבוסס על כל המודלים הקודמים. b) דוגמא לבניית מודל רגרסיה בעזרת אלגוריתם Bagging. ניתן לראות שהמודל המשוקלל הוא בעל שונות קטנה יותר מכל שאר המודלים.

בין אם משתמשים בהכרעת הרוב ובין אם משתמשים במיצוע של המודלים, המודל המשוקלל שנוצר הופך להיות חלק יותר ובעל פחות שיפועים חדים, מה שמקטין את ה-overfitting, וממילא מפחית את השונות. ניתן להבין זאת על ידי דוגמא פשוטה – נניח ויש התפלגות נורמלית $\mathcal{N}(\mu, \sigma^2)$, אז השונות של n דגימות בלתי תלויות הינה $\frac{\sigma^2}{n}$. כעת נניח ומבצעים m ניסויים שבכל אחד מהם דוגמים n נקודות, ובין כל שתי קבוצות יש קורלציה ρ . השונות הממוצעת של הניסויים הינה:

$$\text{Var}\left(\frac{1}{m} \sum_{i=1}^m \text{single cycle}\right) = \frac{1}{m} (1 - \rho)\sigma^2 + \rho\sigma^2$$

אם נבצע הרבה מאוד ניסויים, כלומר ניקח m גדול מאוד, נקבל:

$$\lim_{m \rightarrow \infty} \frac{1}{m} (1 - \rho)\sigma^2 + \rho\sigma^2 = \rho\sigma^2$$

ובסך הכל השונות הסופית הינה בקירוב $\rho\sigma^2$, וביטוי זה לרוב קטן מאשר השונות של מודל המבוסס על הדאטה המקורי ללא שימוש ב-ensembles. ניתן לשים לב שכלל שהקורלציה בין הקבוצות קטנה, כך השונות של המודל המשוקלל גם כן קטנה יותר.

מודל נפוץ מאוד מסוג bagging הוא Random Forest. אלגוריתם זה משלב בין עצי החלטה לבין הרעיון הבסיסי של bagging, כאשר הוא מפצל את הנתונים ואת המשתנים לעצי החלטה רבים, וכל אחד מהם מקבל חלק מסוים מן השלם. העצים הם בעלי שונות גבוהה, כלומר – כל אחד מהם הוא overfitting בפני עצמו, אך עם זאת הקורלציה ביניהם נמוכה, מה שמקל על הורדת השונות והימנעות מ-overfitting. לבסוף, השקלול של כל המודלים ביחד מצליח לייצר מודל בעל שונות נמוכה, ומוביל לתוצאות טובות.

ל-bagging יתרונות רבים. הוא מוריד את השונות, והוא גם חסין לערכים קיצוניים (Outliers). יכולת העבודה שלו במקביל עשויה לאפשר לו להגיע לתוצאות באופן מהיר יותר.

עם זאת, ל-bagging יש גם חסרונות. הוא אינו מוריד את ההטיה, ולכן עשוי לא להתאים במקרים רבים. במודלים של בינה מלאכותית יש חשיבות רבה ליכולת הפרשנות של המודל; לרוב, יידרש הסבר פחות טכני של תוצאות המודל למקבלי ההחלטות או לצרכנים. הם עשויים לא לקבל כלל החלטות של מודל שיראה כ"קופסא שחורה". יש קושי רב לתת פרשנות להחלטות של מודלים מבוססי bagging, והדבר מקשה על השימוש בו. מעבר לכך, bagging עשוי להיות יקר מבחינה חישובית. עקב כך, הוא שימושי מאוד במקרים בהן שיפור זעיר עשוי להוביל להצלחה, אך לרוב תינתן עדיפות למודלים פשוטים יותר של ensembles.

2.4.3 Boosting

כאמור, המושג boosting מתייחס למשפחת אלגוריתמים המשתמשים באוסף של מודלים "חלשים" על מנת ליצור מודל אחד "חזק", כאשר מודלים אלו מתמקדים בניסיון להפחית את ההטיה שיש למודל. מבחינה אינטואיטיבית, מודל חלש הוא כזה שתוצאותיו מעט טובות יותר מניחוש אקראי בעוד שאחד חזק מתקרב לביצועים אופטימליים. בניגוד לטכניקות ensemble אחרות שפועלות במקביל, העקרון המנחה כאן הוא לשרשר את המודלים באופן כזה שכל מודל שמתווסף יטפל בשגיאות שקודמיו פספסו. היופי נעוץ בכך ש-boosting מוכיח כי למידה חלשה בהכרח מצביעה על קיום של שיטת למידה חזקה, לרוב, מודלים מבוססי boosting מתמקדים בבעיות סיווג בינארי.

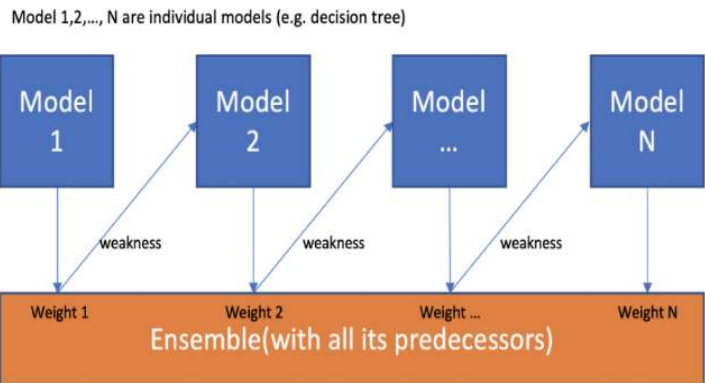
באופן פורמלי, המושגים "לומד חלש" ו-"לומד חזק" עבור בעיית סיווג בינארי מוגדרים כך: אלגוריתם נקרא לומד חזק אם לכל $\epsilon, \delta > 0$, האלגוריתם מסוגל (עבור אוסף נתונים גדול מספיק) לבנות מסוג $c(x)$ שמקיים $p(c(x) \neq y) < \epsilon$ בהסתברות גדולה מ- $1 - \delta$. לומד חלש הינו אלגוריתם שלכל $\delta > 0$ קיים $\gamma > 0$ כך שעבור אוסף נתוני מספיק גדול, האלגוריתם מסוגל לבנות מסוגל שמקיים $p(c(x) \neq y) < \frac{1}{2} - \gamma$ בהסתברות גדולה מ- $1 - \delta$. כאמור, המטרה של boosting הינה לקחת אוסף של מודלים חלשים ובעזרתם ליצור מודל חזק, כאשר הצליחו להוכיח שניתן להפוך כל לומד חלש ללומד חזק על ידי בניית קומביניצייה לינארית של מסווגים אשר נוצרו בעזרת הלומד החלש.

נמחיש את הרעיון של boosting בעזרת דוגמא: נניח יש בידינו אוסף נתונים X , המחולק באופן אקראי לשלוש קבוצות שוות (כל אחת מכילה שליש מהנתונים) x_1, x_2, x_3 . כעת בונינו מודל לצורך סיווג בינארי, המסומן ב- h_1 . נמצא כי h_1 מתאים בצורה טובה רק לקבוצות x_1, x_2 אך מסווג בצורה לא טובה את פריטי הקבוצה x_3 . כיוון ש- x_3 מכיל שליש מסך הנתונים, שגיאת הסיווג גדולה ו- $c_1(x)$ הוא מודל חלש, ונרצה לשפר אותו. בכדי לעשות זאת ניקח רק חלק מהנתונים, $X' \in X$, ונדאג לכך ש- X' יכיל הרבה מאיברי x_3 . כעת נבנה מודל נוסף $c_2(x)$ על בסיס X' , מתוך כוונה שמודל זה יתמקד גם בקבוצה x_3 ויסווג את איבריה בצורה טובה. כעת נניח שמודל זה אכן מסווג בצורה נאותה את איברי x_3 , אך הפעם המודל שוגה בצורה גסה בסיווג איברי x_2 . עקב השגיאה בסיווג x_2 המודל השני גם הוא מודל חלש, אך כעת יש בידינו שני מודלים חלשים שהחולשה בכל אחד נובעת מקבוצת איברים אחד של אוסף הנתונים המקורי X . אם נמצא דרך הולמת לחבר את שני המסווגים, נוכל ליצור מודל בעל פוטנציאל להצליח לסווג את X כמו שצריך.

באופן כללי, אם רוצים לאמן מודל $C(x)$ בעזרת אלגוריתם L על אוסף הנתונים \mathcal{D} , יש לבצע את השלבים הבאים:

1. אתחול הנתונים: $\mathcal{D}_1 = \mathcal{D}$.
2. עבור $t = 1, \dots, T$:
 אימון מודל חלש על \mathcal{D}_t : $c_t(x) = L(\mathcal{D}_t)$
 חישוב שגיאת המסווג: $\epsilon_t = P_{x \sim \mathcal{D}_t}(c_t(x) \neq f(x))$.
3. התאמת הנתונים עבור האיטרציה הבאה: $\mathcal{D}_{t+1} = \text{Adjust Distribution}(\mathcal{D}_t, \epsilon_t)$
 איחוד המודלים החלשים: $C(x) = \text{combine outputs}\{c_1(x), \dots, c_T(x)\}$.

יש כל מיני שיטות כיצד לבצע את השלבים השונים באלגוריתם boosting, ונפרט את המרכזיות שבהן.



איור 2.18 – סכמה כללית של boosting. המודלים (במקרה זה מדובר בעץ החלטה רדוד, אך זה תקף לכל מודל חלש) מחוברים אחד לשני באופן שכל אחד לומד מהתפלגות המשוקללת בהתאם לשגיאות של המודלים הקודמים.

Adaptive-Boosting (AdaBoost)

Adaboost היא אחת הטכניקות הראשונות של boosting, ועל אף שקיימות טכניקות boosting נוספות, Adaboost היא בין הפופולריות ביותר בתחום (אם כי יש לה מספר לא מבוטל של וריאנטים). העוצמה הגלומה בטכניקה זו נובעת מכך שגם בהינתן מספר מאפיינים רב, האלגוריתם מצליח להיפגע פחות מ"קללת הממדיות" ולשמור על יכולות ניבוי טובות, בניגוד לאלגוריתמים אחרים של סיווג, כמו למשל SVM או אפילו רשתות נוירונים.

כזכור, תחת ההנחה שקיים אלגוריתם לומד חלש $c(x)$, המטרה היא למצוא דרך להפוך אותו למודל חזק $C(x)$. באופן אינטואיטיבי היה ניתן לחשוב שאפשר פשוט לאמן מספר מודלים על תת קבוצות של הדאטה המקורי (עם אפשרות לחפיפות בין תת-קבוצות), להשתמש ב-majority vote, ובכך לשרשר את ההיפותוזות של כל המודלים לפלט אחד. גישה זו כמובן נאיבית ופשטנית, ואינה לוקחת בחשבון מקרה בו מרבית המודלים שוגים. גישה טובה יותר תהיה לבנות מודל על בסיס חלק מהדאטה, לבחון את מידת הצלחה של המודל על יתר הדאטה, ולפי הצלחה שלו במשימה זו לתת משקל ל-vote של המודל. ניתן להוסיף תחכום לרעיון זה, כך שבכל שלב יינתן יותר דגש איברים בדאטה שהמודלים הקודמים שגו בסיווג שלהם, ובכך בכל שלב בו מאמנים מודל נוסף תהיה הצלחה יותר גדולה מאשר המודל הקודם. חלק זה הינו החלק האדפטיבי (Adaptive) באלגוריתם, על שמו נקרא האלגוריתם Adaboost.

כעת, נסביר כיצד ניתן להרכיב מסווג חזק באמצעות אוסף של מסווגים חלשים עבור אוסף נתונים $X \in \mathbb{R}^N$

1. ראשית יש לאתחל משקולות באופן אחיד עבור כל אחת מ- N הדוגמאות בסט הנתונים $w_i^{t=0} = \frac{1}{N}$.

2. לאחר מכן יש לבצע איטרציות באופן הבא:

בניית מסווג אופטימלי $c_t(x)$ ביחס לאוסף הנתונים המשוקלל.

חישוב שגיאת הסיווג של $c_t(x)$: $\epsilon_t = \sum_i w_i^t \{c_t(x) \neq y_i\}$

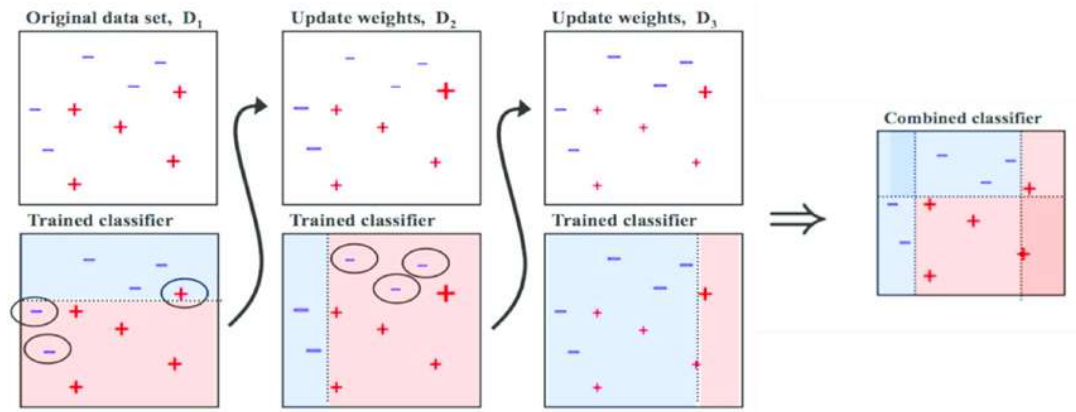
חישוב משקל עבור מסווג זה: $\alpha_t = \frac{1}{2} \ln \left(\frac{1-\epsilon_t}{\epsilon_t} \right)$

עדכון המשקלים: $w_i^{t+1} = w_i^t \exp(-\alpha_t y_i c_t(x_i))$

נרמול המשקלים בהתאם לסכומם הכולל: $N_{t+1} = \sum_i w_i^t \rightarrow w_i^{t+1} = \frac{w_i^t}{N_{t+1}}$

3. חישוב המסווג המשוקלל, שהינו קומבינציה לינארית של המסווגים החלשים:

$$C(x) = \text{sign} \left(\sum_t \alpha_t c_t(x) \right)$$



איור 2.19 – דוגמא לשימוש ב-AdaBoost עבור מודל סיווג בינארי.

2. References

SVM:

https://commons.wikimedia.org/wiki/File:Svm_max_sep_hyperplane_with_margin.png

<https://svm.michalhaltuf.cz/support-vector-machines/>

<https://medium.com/analytics-vidhya/how-to-classify-non-linear-data-to-linear-data-bb2df1a6b781>

https://xavierbourretsicotte.github.io/Kernel_feature_map.html

Naïve Bayes:

https://en.wikipedia.org/wiki/Naive_Bayes_classifier

https://scikit-learn.org/stable/modules/naive_bayes.html

K-NN:

<https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html>

EM:

https://www.cs.toronto.edu/~urtasun/courses/CSC411_Fall16/13_mog.pdf

https://stephens999.github.io/fiveMinuteStats/intro_to_em.html

Hierarchical Clustering:

<https://www.datanovia.com/en/lessons/agglomerative-hierarchical-clustering/>

LOF:

<https://towardsdatascience.com/local-outlier-factor-lof-algorithm-for-outlier-identification-8efb887d9843>

PCA:

Saal, L.H. et al. (2007). *Proc. Natl. Acad. Sci. USA* 104, 7564–7569.

3. Linear Neural Networks

פרק זה עוסק בבעיות רגרסיה – כיצד ניתן בעזרת אוסף דוגמאות נתון לבנות מודל המסוגל לספק מידע על נקודות חדשות שיגיעו וחסר עליהן מידע. המודלים שיוצגו בפרק זה מתייחסים לדאטה שניתן למצוא עבורו הפרדה לינארית, כלומר, ניתן למצוא קווים לינאריים המחלקים את הדאטה לקבוצות שונות. החלק הראשון של הפרק יעסוק ברגרסיה לינארית (Linear regression) והחלק השני יעסוק ברגרסיה לוגיסטית (Logistic regression). לבסוף יוצג מבנה שקול לבעיות הרגרסיה בעזרת רשת נירונים פשוטה, ומבנה זה יהיה הבסיס לפרק הבא העוסק ברשתות נירונים עמוקות, הבאות להתמודד עם דאטה שאינו ניתן לבצע עבורו הפרדה לינארית.

3.1 Linear Regression

3.1.1 The Basic Concept

המודל הפשוט ביותר הינו linear regression. מודל זה מנסה למצוא קשר לינארי בין מספר משתנים או מספר מאפיינים. בהנחה שמתקיים יחס לינארי בין סט משתנים בלתי תלויים $x \in \mathbb{R}^d$ לבין משתנה תלוי $y \in \mathbb{R}$, ניתן לכתוב את הקשר ביניהם בצורה הבאה:

$$\hat{y} = w^T x + b = w_1 x_1 + w_2 x_2 + \dots + w_d x_d + b$$

כאשר $w \in \mathbb{R}^d$ הם המשקלים ו- $b \in \mathbb{R}$ נקרא bias.

דוגמא: ניתן לטעון כי מחיר הבתים באזור מסוים נמצא ביחס לינארי למספר פרמטרים: גודל הדירה, איזה קומה היא נמצאת, וכמה שנים הבניין קיים. תחת הנחה זו, יש לבחון את המודל עבור דוגמאות ידועות ובכך למצוא את המשקלים וה-bias. לאחר מכן ניתן יהיה לקחת את המודל ולנחש את מחיר הדירה עבור בתים שמחירם לא ידוע, אך הפרמטרים שלהם כן נתונים.

בכדי לבנות מודל המאפשר לשערך בצורה טובה את y בהינתן סט מאפיינים, יש לדעת את המשקלים וה-bias. כיוון שהם לא ידועים, יש לחשב אותם בעזרת אוסף של דוגמאות ידועות. ראשית יש להגדיר פונקציה מחיר (Loss), הקובעת עד כמה הביצועים של מודל מסוים טובים. פונקציית המחיר היא פונקציה של הפרמטרים הנלמדים - $L(w, b)$, והבאתה למינימום תספק את הערכים האופטימליים של המשקלים וה-bias. פונקציית מחיר מקובלת הינה השגיאה הריבועית הממוצעת (MSE) – המחשבת את ריבוע ההפרש בין החיזוי לבין הפלט האמיתי:

$$L^{(i)}(w, b) = \frac{1}{2} (y_i - \hat{y}_i)^2$$

כאשר נתונות n דוגמאות ידועות, יש לסכום את כל הפרשים האלו:

$$L(w, b) = \frac{1}{n} \sum_{i=1}^n \frac{1}{2} (y_i - \hat{y}_i)^2 = \frac{1}{n} \sum_{i=1}^n \frac{1}{2} (y_i - w^T x_i - b)^2$$

כעת בשביל למצוא את הפרמטרים האופטימליים, יש למצוא את w, b שמביאים את פונקציית המחיר למינימום:

$$\hat{w}, \hat{b} \equiv \hat{\theta} = \arg \min L(w, b)$$

עבור המקרה הסקלרי בו $d = 1$, כלומר יש מאפיין יחיד ומנסים למצוא קשר בינו לבין פלט מסוים, הקשר הלינארי הוא $\hat{y} = wx + b$. עבור המקרה הזה, פונקציית המחיר תהיה:

$$L(\theta) = \frac{1}{n} \sum_{i=1}^n \frac{1}{2} (y_i - w^T x_i - b)^2$$

ובכדי למצוא אופטימום יש לגזור ולהשוות ל-0:

$$\frac{\partial L}{\partial w} = \frac{1}{n} \sum_{i=1}^n (y_i - w^T x_i - b) \cdot (-x_i) = 0$$

$$\frac{\partial L}{\partial b} = \frac{1}{n} \sum_{i=1}^n (y_i - w^T x_i - b) \cdot (-1) = 0$$

מתקבלות סט משוואות לינאריות:

$$w \sum x_i^2 + b \sum x_i = \sum y_i x_i$$

$$w \sum x_i + bn = \sum y_i$$

ובכתיב מטריציוני:

$$\begin{pmatrix} \sum x_i^2 & \sum x_i \\ \sum x_i & n \end{pmatrix} \begin{pmatrix} w \\ b \end{pmatrix} = \begin{pmatrix} \sum y_i x_i \\ \sum y_i \end{pmatrix}$$

על ידי הצבה של הדוגמאות הנתונות ניתן לקבל את הפרמטרים של הקשר הלינארי.

לשם הנוחות ניתן לסמן את ה-bias כפרמטר נוסף:

$$\hat{y} = w^T x + b = (w^T \ b) \begin{pmatrix} x \\ 1 \end{pmatrix} = \tilde{w}^T \tilde{x}, \quad \tilde{w}, \tilde{x} \in \mathbb{R}^{d+1}$$

עבור המקרה הווקטורי יש n דוגמאות, כלומר יש n מאפיינים בלתי תלויים ומנסים למצוא את הקשר ביניהם לבין פלט מסוים. במקרה זה $X_{n \times d+1} = (x_1, \dots, x_n)^T, Y = (y_1, \dots, y_n)^T$ ופונקציית המחיר הינה:

$$L(\theta) = \frac{1}{n} \sum_{i=1}^n \frac{1}{2} (y_i - w^T x_i)^2$$

המינימום של הביטוי הזה שקול למינימום של $\|Y - Xw\|^2$:

$$\frac{\partial L}{\partial w} = \frac{1}{n} \sum_{i=1}^n (y_i - w^T x_i) \cdot (-x_i) = 0$$

$$\rightarrow X^T (Xw - Y) = 0$$

$$\hat{w} = (X^T X)^{-1} X^T Y$$

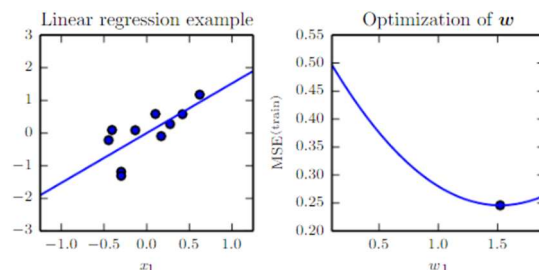
ובהינתן אוסף דוגמאות:

$$X = \begin{pmatrix} x_1 & 1 \\ \vdots & \vdots \\ x_n & 1 \end{pmatrix}, \quad \begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix}$$

אזי הפתרון של הרגרסיה הלינארית הינו:

$$\hat{w} = (X^T X)^{-1} X^T Y = \begin{pmatrix} \sum x_i^2 & \sum x_i \\ \sum x_i & n \end{pmatrix}^{-1} \begin{pmatrix} \sum y_i x_i \\ \sum y_i \end{pmatrix}$$

דוגמא למציאת קו הרגרסיה והמשקל האופטימלי עבור בעיה סקלרית:



איור 3.1 רגרסיה לינארית אופטימלית עבור אוסף דוגמאות נתון (שמאל) ואופטימיזציה עבור המשקל w ביחס לפונקציית המחיר (ימין).

3.1.2 Gradient Descent

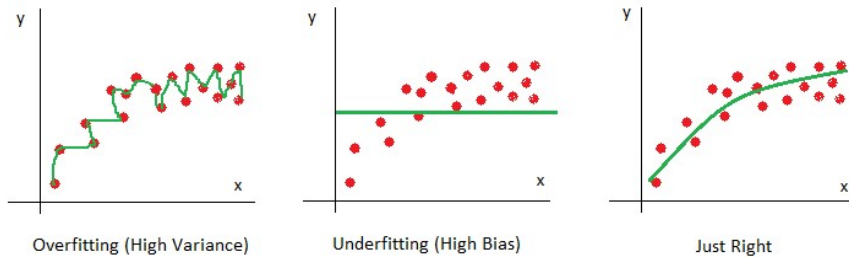
הרבה פעמים מציאת המינימום של פונקציית המחיר היא משימה קשה. דרך מקובלת להתמודד עם חישוב הפרמטר האופטימלי היא שיטת gradient descent (GD). בשיטה זו מתחילים מניחוש מסוים עבור הפרמטרים, וכל פעם מבצעים צעד לכיוון הגרדיאנט השלילי. הגרדיאנט הוא הנגזרת של הפונקציה, והוא מגדיר את הכיוון שערך הפונקציה עולה בו בצורה מקסימלית. אם לוקחים את הכיוון השלילי של הגרדיאנט, בעצם הולכים לכיוון בו יש את הירידה הכי גדולה, ולכן כדי להגיע למינימום יש לבצע את הצעד בכיוון הגרדיאנט השלילי. בכדי להימנע מהיקלעות לנקודת אוקף, מוסיפים איבר הנקרא learning rate (lr) (מסומן באת ϵ). מבצעים את הגזירה ושינוי הפרמטרים באופן איטרטיבי עד נקודת עצירה מסוימת. באופן פורמלי, עבור ניחוש התחלתי θ_0 , בכל צעד יבוצע הקידום באופן הבא (העדכון מתבצע באופן סימולטני עבור כל θ_{j+1}):

$$\hat{\theta}_{j+1} = \hat{\theta}_j - \epsilon \cdot \frac{\partial}{\partial \theta_j} L(\hat{\theta}_j)$$

קידום זה יבוצע שוב ושוב עד התכנסות לערך מסוים. כיוון שהבעיה קמורה מובטח שתהיה התכנסות למינימום, אך היא יכולה להיות איטית עקב צעדי עדכון גדולים או קטנים מדי. פרמטר ה-learning rate, ϵ , קובע את קצב ההתכנסות, לכן רצוי לבחור פרמטר לא קטן מדי כדי לא להאט את ההתכנסות ולא גדול מדי כדי למנוע התכנסות.

3.1.3 Regularization and Cross Validation

אחד האתגרים המרכזיים של בעיית הרגרסיה (ושאר בעיות הלמידה) הוא לפתח מודל שיהיה מוצלח לא רק עבור אוסף הדוגמאות הידוע (-סט האימון), אלא שיהיה מספיק טוב גם עבור דוגמאות חדשות ולא מוכרות (-קבוצת מבחן). כל מודל יכול לסבול מהטיה לשני כיוונים – Overfitting ו-Underfitting. Overfitting הוא מצב בו ניתנת הערכת יתר לכל נקודה בסט האימון, מה שגורר מודל מסדר גבוה בעל שונות גדולה. במצב זה המודל מתאים רק לסט האימון, אך הוא לא מצליח להכליל גם נקודות חדשות. Underfitting הוא המצב ההפוך – מודל שלא מצליח למצוא קו מגמה המכיל מספיק מידע על הדוגמאות הנתונות, ויש לו רעש חזק.



איור 3.2 Overfitting – נתינת משקל יתר לכל נקודה גורמת למצב בו המודל הוא מסדר גבוה ובעל שונות גבוהה (שמאל). Underfitting – מודל בעל רעש חזק מייצג בצורה מספיק טובה את המידע (אמצע). מצב מאוזן – מודל בעל שגיאה מינימלית, המתאר בצורה טובה את המידע, ובנוסף נמנע משגיאת יתר עבור דוגמאות חדשות (ימין).

כדי להימנע מהטיית אלו, יש לבצע Regularization – הוספת אילוץ המונע מהמודל להיות מוטה באופן הפוגע בתוצאות. לאחר הוספת האילוץ, פונקציית המחיר תהיה בצורה:

$$\text{Regularized Loss} = \text{Loss Function} + \text{Constraint}$$

יש מספר דרכים לבצע את ה-regularization:

Ridge Regression / L2 Regularization

דרך אחת לבצע את ה-regularization היא להוסיף איבר נוסף המתייחס לריבוע הפרמטרים:

$$L(\theta) = \text{MSE}_{\text{train}} + \lambda w^T w = \frac{1}{n} \sum_{i=1}^n \frac{1}{2} (y_i - w^T x_i)^2 + \lambda \|w\|^2$$

כעת האופטימום של הביטוי הינו:

$$\hat{w} = (X^T X + \lambda I)^{-1} X^T Y$$

הוספת האילוץ גורמת לכך שבנוסף לחיזוי מדויק של משתנה המטרה, המודל מנסה למזער את ריבוע הפרמטרים, ובכך לנסות להקטין עד כמה שניתן את הערך של כל פרמטר ולהימנע ממצב בו נותנים משקל יתר לחלק מהפרמטרים. למעשה האילוץ מקטין את השונות של המודל ובכך עשוי למנוע overfitting.

Lasso / L1 Regularization

דרך נוספת לבצע את ה-Regularization היא להוסיף אילוץ המתייחס לערך המוחלט של הפרמטרים:

$$L(\theta) = \text{MSE}_{\text{train}} + \lambda|w| = \frac{1}{n} \sum_{i=1}^n \frac{1}{2} (y_i - w^T x_i)^2 + \lambda|w|$$

הוספת האילוץ מכריחה את סכום הפרמטרים להיות כמה שיותר קטן, כדי למזער כמה שניתן את פונקציית המחיר. בפועל אילוץ זה מביא ל"רידוד משקלים" (sparse), כלומר כופה חלק מהמקדמים להיות אפס, וכך למעשה יש מעין feature selection – בחירת הפרמטרים המשמעותיים יותר.

ניתן לשים לב כי עבור L2 ההשפעה של המשקלים על פונקציית המחיר היא ריבועית. לכן במקרה זה הרגולריזציה תשאף להקטין את הפרמטרים הגדולים, ובאופן כללי תנסה לדאוג לכך שכל הפרמטרים יהיו קטנים, ובאותו סדר גודל. L1 לעומת זאת שואף להקטין את כל האיברים כמה שיותר ללא קשר לגודלם, ולהקטנת פרמטר מסוים מ-10 ל-9 יש את אותה השפעה כמו הקטנה של פרמטר מ-1000 ל-999. לכן במקרה זה הרגולריזציה תגרום לפרמטרים הפחות חשובים להתאפס, והמודל נהיה פשוט יותר.

Elastic Net

ניתן לשלב בין Ridge Regression לבין Lasso, ובכך לנסות לכוון את המודל עבור היתרונות של כל שיטה – גם להימנע מנתינת משקל יתר לפרמטרים וגם ניסיון לאפס פרמטרים, ובכך לקבל מודל פשוט ככל הניתן. פונקציית המחיר במקרה זה תהיה מהצורה:

$$L(\theta) = \text{MSE}_{\text{train}} + \lambda\|w\|^2 + \lambda_2|w|$$

עבור כל אחת מהדרכים, יש למצוא את הפרמטר λ האופטימלי עבור ה-Regularization (במקרה של Elastic Net, הפרמטר λ הוא למעשה וקטור: $\vec{\lambda} = [\lambda_1, \lambda_2]$). שיטה מקובלת למציאת λ האופטימלי היא Cross validation – חלוקת n הדוגמאות של סט האימון ל- k קבוצות, אימון כל תתי הקבוצות מלבד אחת, ואז בדיקת הפרמטרים שהתקבלו בשלב האימון על הקבוצה שנותרה. בכל איטרציה מוציאים חלק מאוסף הדוגמאות והופכים אותן לקבוצת מבחן, וכך מוצאים את הפרמטר λ האופטימלי המונע מהמשקלים להגיע מ-fitting (בדרך כלל לוקחים את הממוצע של כל ה- λ מכל האיטרציות). נפוץ להשתמש ב- $k = 5$, ולמעשה עבור בחירה טיפוסית זו יהיו 5 איטרציות, שבכל אחת מהן האימון יתבצע על 80% מסט האימון, ולאחר מכן תתבצע הבדיקה של הפרמטרים שנלמדו על ה-20% הנותרים.

Split 1	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5
Split 2	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5
Split 3	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5
Split 4	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5
Split 5	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5

איר 3.3 Cross validation עם חלוקה ל-5 קבוצות ($k = 5$). בכל פעם קבוצה אחת משמשת ל-validation (הקבוצה הכחולה).

בחירה של $k = n$ נקראת leave-one out cross validation כיוון שלמעשה בכל איטרציה יש דוגמא אחת בלבד שלא נכללת בסט האימון ועליה מתבצעת הבדיקה של הפרמטרים שנלמדו.

3.1.4 Linear Regression as Classifier

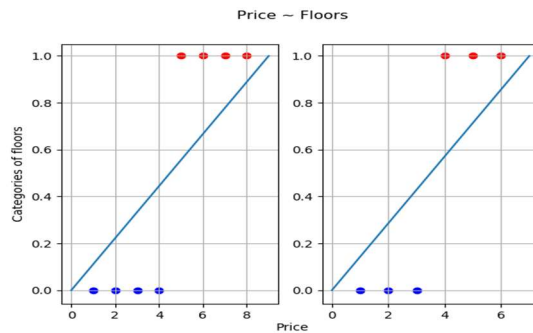
משימת סיווג מוגדרת באופן הבא: בהינתן סט פרמטרים מסוים $X = \{x_1, \dots, x_n\}$ השייך לתצפית מסוימת, יש לסווג אותו לאחת מתוך m קטגוריות אפשריות: $y \in \{1, \dots, m\}$. לדוגמא: נתונה תמונה בעלת n פיקסלים המייצגת חיה, ויש לקבוע איזו חיה היא, כאשר הבחירה נעשית מתוך הווקטור y . לרוב הווקטור y מורכב ממספרים שלמים, שכל

אחד מהם מייצג בחירה מסוימת. בדוגמא של החיות, ניתן לקחת לדוגמא $m = 3$, כלומר $y \in \{1,2,3\}$, כאשר המספרים מייצגים את סט החיות $\{dog, cat, chicken\}$.

ניתן להשתמש במודל של רגרסיה לינארית למשימות של סיווג. עבור המקרה של $m = 2$, יש שתי קטגוריות אפשריות, ולמעשה יש צורך למפות כל נקודה לאחת משתי הקטגוריות. בעזרת רגרסיה לינארית ניתן לבצע מיפוי מ- \mathbb{R} ל- $\{0,1\}$, כלומר כל נקודה במרחב ממופה לאחד משני ערכים: קובעים ערך סף $T = 0.5$, ועבור נקודה חדשה x_n בודקים מה היחס בין הביטוי $w^T x_n + b$ לבין ערך הסף. אם הנקודה החדשה מקיימת: $w^T x_n + b < 0.5$ אז הנקודה החדשה תתויג בקטגוריה 1. אחרת, הנקודה החדשה תתויג בקטגוריה 0. באופן פורמלי:

$$y = \text{sign}(w^T x_{new} + b - 0.5) = \begin{cases} 1 & w^T x_{new} + b > 0.5 \\ 0 & w^T x_{new} + b < 0.5 \end{cases}$$

הבחירה בערך הסף $T = 0.5$ נובעת מכך שיש שתי קטגוריות $\{0,1\}$, וערך הסף נקבע להיות נקודת האמצע ביניהן. לדוגמא: נתונים n בתים ועבור כל אחד מהם ידוע מה מחירו והאם יש בו קומה אחת או שתיים. כעת רוצים לבחון את היחס בין המחיר למספר הקומות ולקבוע עבור מחיר בית נתון מה מספר הקומות שלו. במקרה זה יש 2 קטגוריות: $\{0,1\} = \{1 \text{ floor}, 2 \text{ floors}\}$, ויש להיעזר בידע על n הבתים בכדי לבנות מודל מסווג. הדרך לעשות זאת היא לבצע רגרסיה לינארית, ואז כשבוחנים מחיר של בית, יש לבדוק אם $w^T \cdot \text{price} + b$ גדול מ-0.5 או קטן ממנו, כאשר (w, b) הם הפרמטרים של הרגרסיה הלינארית.



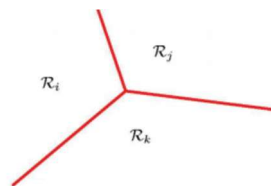
איור 3.4 רגרסיה לינארית כמסווג בינארי: מיפוי הנקודות בהתאם למיקומן ביחס לקו הפרדה של הרגרסיה הלינארית. בדוגמא הימנית ערך הסף מתקבל עבור $x_T = 3.5$, כלומר $w^T \cdot 3.5 + b = 0.5$, ובדוגמא השמאלית ערך הסף מתקבל עבור $x_T = 4.5$. עבור כל בית חדש, בהינתן מחירו ניתן יהיה לסווג אותו לאחת משתי הקטגוריות, בהתאם ליחסו לערך הסף 0.5.

עבור n נקודות ידועות – $(x_1, y_1) \dots (x_n, y_n), y_i \in \{0,1\}$, פונקציית המחיר הינה:

$$L(\theta) = \sum_{i=1}^n 1_{\{y_i \neq \text{sign}(w^T x_i + b + 0.5)\}}$$

הפונקציה $L(\theta)$ מכילה סט של פרמטרים – $\theta = (w, b)$. כיוון שהנגזרת של הפונקציה לפי כל אחד מהפרמטרים שלה w_i לא תלויה רק באותו פרמטר, קשה למצוא את θ המביאים למינימום את $L(\theta)$.

ניתן להרחיב את המסווג גם עבור מקרים בהם יש יותר משתי קטגוריות (multi-class). סט האימון תראה כמו במקרה הבינארי, ואילו y_i מכיל כעת m קטגוריות: $y_i \in \{1, \dots, m\}$. במקרים אלו יש לייצר מספר קווים לינאריים, המפרידים בין אזורים שונים. כדי לחשב את הקווים מבצעים התהליך שנקרא one versus all, בו בכל פעם לוקחים קטגוריה אחת ובודקים מהו קו הפרדה בינה לבין שאר הקטגוריות. הפרמטרים הנלמדים של קווי הפרדה יהיו הסט המורכב מכל הפרמטרים של הרגרסיה: $\theta = \{w_1, b_1, \dots, w_m, b_m\}$.



איור 3.5 רגרסיה לינארית מרובה – הפרדה בין מספר אזורים שונים על ידי מספר קווים לינאריים. במקרה הזה, נקודה חדשה תסווג לקטגוריה לפי הביטוי הבא:

$$y(x) = \arg \max_i (w_i^T x + b_i)$$

וכל אזור יוגדר לפי:

$$R_i = \{x|y(x) = i\}$$

בדומה למקרה הבינארי, פונקציית המחיר תהיה:

$$L(\theta) = \sum_{i=1}^n 1_{\{y_i \neq \hat{y}_i\}} \text{ s.t. } \hat{y}_i = \arg \max_i (w_i^T x + b_i)$$

המסווג האופטימלי יהיה וקטור הפרמטרים המביא את פונקציית המחיר למינימום:

$$\hat{\theta} = \arg \min_{\theta} L(\theta)$$

גם במקרה הזה, כיוון שהנגזרת של פונקציית המחיר לפי כל פרמטר אינה תלויה רק באותו פרמטר, בפועל קשה למצוא את θ האופטימלי המביא את $L(\theta)$ למינימום.

3.2 Softmax Regression

3.2.1 Logistic Regression

המסווג הנוצר מהרגרסיה הלינארית הינו "מסווג קשה" – כל דוגמא חדשה שמתקבלת מסווגת לקטגוריה מסוימת, ואין שום מידע עד כמה הדוגמא הזו דומה לקטגוריות האחרות. מסווג כזה אינו מספיק טוב עבור מגוון בעיות, בהן מעוניינים לדעת לא רק את הקטגוריה, אלא גם מידע נוסף על היחס בין הדוגמא החדשה לבין כלל הקטגוריות. לדוגמא: בהינתן ממד של גידול מסוים רוצים לדעת אם הוא ממאיר או שפיר. במקרה זה ההכרעה היא לא תמיד חד משמעית, ויש עניין לדעת מה הסיכוי של הגידול להיות ממאיר או שפיר, שהרי יתכן שהטיפול יהיה שונה בין מקרה בו יש 1% שהגידול הזה הוא מסוג מסוים לבין מקרה בו יש 40% שהגידול הוא מהסוג הזה. כדי להימנע מהסיווג הקטגורי, יש ליצור מודל הסתברותי, בו כל קטגוריה מקבלת הסתברות מסוימת. אחד המודלים הבסיסיים הינו רגרסיה לוגיסטית (Logistic regression). עבור המסווג הזה ראשית יש להגדיר את פונקציית הסיגמואיד:

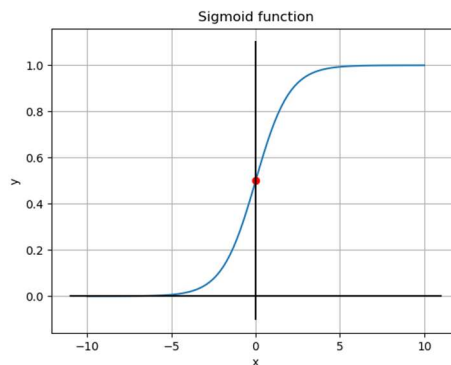
$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

פונקציה זו רציפה על כל הישר, ובעזרתה ניתן להגדיר מסווג עבור המקרה הבינארי:

$$p(y = 1|x; \theta) = \sigma(w^T x + b) = \frac{1}{1 + e^{-(w^T x + b)}}$$

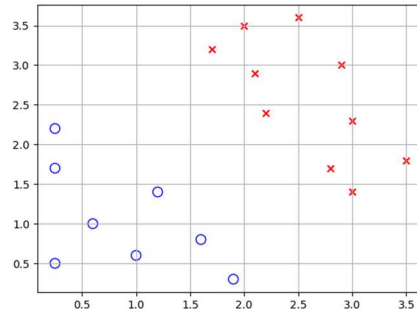
$$p(y = 0|x; \theta) = 1 - \sigma(w^T x + b) = 1 - \frac{1}{1 + e^{-(w^T x + b)}} = \frac{e^{-(w^T x + b)}}{1 + e^{-(w^T x + b)}}$$

המסווג לוקח את קו ההחלטה הלינארי, ומעביר אותו בפונקציה המחזירה ערך בטווח $[0, 1]$, כאשר הערך המוחזר הוא ההסתברות להיות בקטגוריה מסוימת. בכדי להבין יותר טוב את משמעות המסווג, יש להסתכל על גרף הסיגמואיד:



איור 3.6 גרף הפונקציה: $y = \frac{1}{1+e^{-x}}$. הנקודה $(0,0.5)$ מודגשת באדום.

כאשר הפונקציה $\theta(x) = w^T x + b$ שווה בדיוק ל-0, אזי $\sigma(w^T x + b) = 0.5$. המשמעות של התוצאה הזו היא שאם עבור סט פרמטרים x_n מתקיים $w^T x_n + b > 0$, אז ההסתברות של הנקודה הזו להיות משויכת לקטגוריה 1 גדולה מחצי, כיוון ש- $\sigma(w^T x + b) > 0.5$. באופן סימטרי אם $w^T x_n + b < 0$, אז ההסתברות של הנקודה x_n להיות משויכת לקטגוריה 1 קטנה מחצי. כעת עולה השאלה מתי $w^T x + b = 0$, והתשובה היא שזה תלוי בקו ההפרדה שבין הקטגוריות. לשם המחשה נניח ונתונות מספר מדידות על שני פרמטרים - x_1, x_2 , ועבור כל נקודה (x_1, x_2) נתון גם מה הקטגוריה שלה $(y \in \{0,1\} = \{blue\ o, red\ x\})$:



איור 3.7 דוגמא למספר מדידות התלויות בשני פרמטרים x_1, x_2 , ומשויכות לאחת משתי קטגוריות: $y \in \{0,1\} = \{blue\ o, red\ x\}$.

כיוון שנתונות הנקודות, ניתן לייצר בעזרתן קו רגרסיה. לצורך הדוגמא נניח שיש שלושה פרמטרים והם מקיימים: $[w_1, w_2, b]^T = [1, 1, -3]^T$. הפרמטרים האלו מרכיבים את הקו הליניארי $x_1 + x_2 = 3$, כלומר, עבור כל נקודה אם מתקיים $x_1 + x_2 > 3$ אזי היא תהיה מסווגת כ-'red x', אחרת היא תהיה מסווגת כ-'blue o'. קו זה הוא למעשה קו הפרדה שניתן בעזרתו לסווג נקודות חדשות. קו זה מקיים את המשוואה $w^T x + b = 0$, ולכן אם תהיה נקודה חדשה שגם מקיימת $w^T x_n + b = 0$, המשמעות היא שנקודה זו נמצאת בדיוק על קו ההפרדה. נקודה כזו תקבל הסתברות של 50% להיות משויכת לכל אחת מהקטגוריות. ככל שהנקודה החדשה תתרחק מקו ההפרדה, כך הביטוי $w^T x + b$ יתרחק מה-0, ולכן גם $\sigma(w^T x + b)$ יתרחק מהערך חצי ויתקרב לאחד מערכי הקצה 0 או 1, והמשמעות היא כמובן שיש יותר סיכוי שנקודה זה שייכת לקטגוריה אחת ולא לאחרת.

כמובן שניתן לקחת גם את המסווג ההסתברותי הזה ולהשתמש בו כמסווג קשה: עבור דוגמא חדשה לוקחים את ההסתברויות שלה לכל אחת מהקטגוריות, ומסווגים את הדוגמא לקטגוריה בעלת ההסתברות הגבוהה ביותר. במקרה הבינארי וקטור ההסתברויות הינו $\hat{y} = [p(y = 1|x), p(y = 0|x)]$, והקטגוריה של \hat{y} תהיה $\arg \max_i \hat{y}_i$ שזהו ה- \hat{y}_i בעל ההסתברות הגדולה ביותר.

3.2.2 Cross Entropy and Gradient descent

בכדי למצוא את הפרמטרים $\theta = (w, b)$ האופטימליים בהינתן n דוגמאות, ניתן להחליף את קריטריון השיגאה הריבועית הממוצעת בקריטריון אחר למזעור פונקציית המחיר - Cross entropy. קריטריון זה אומר שיש להביא למינימום את מינוס הלוג של סך הדוגמאות (הביטוי נובע משערוך הנראות המרבית - [Maximum likelihood](#)):

$$-\log P(Y|X; \theta) = -\frac{1}{n} \sum_{i=1}^n \log p(y_i|x_i; \theta) = L(\theta)$$

למעשה, יש למצוא את סט הפרמטרים $\hat{\theta}$ המביא את הביטוי למינימום: $\hat{\theta} = \arg \min_{\theta} L(\theta)$.

בכדי לחשב את הביטוי יש לפתח קודם את הביטוי עבור נגזרת הסיגמואיד:

$$\sigma(z) = \frac{1}{1 + e^{-z}} \rightarrow \frac{\partial \sigma(z)}{\partial z} = \frac{-1}{(1 + e^{-z})^2} \cdot e^{-z} \cdot (-1) = \frac{1}{1 + e^{-z}} \cdot \frac{e^{-z}}{1 + e^{-z}} = \sigma(z)(1 - \sigma(z))$$

זכור, $1 - \sigma(z)$, כזכור, $p(y = 0|x; w, b) = 1 - p(y = 1|x; \theta) = 1 - \sigma(\theta)$, לכן יש לחשב גם את הנגזרת עבור $1 - \sigma(z)$:

$$\frac{\partial(1 - \sigma(z))}{\partial z} = -\sigma(z)(1 - \sigma(z))$$

בהתאם, הנגזרות של לוג סיגמואיד הן:

$$\frac{\partial \log \sigma(z)}{\partial z} = \frac{1}{\sigma(z)} \cdot \frac{\partial \sigma(z)}{\partial z} = (1 - \sigma(z))$$

$$\frac{\partial \log(1 - \sigma(z))}{\partial z} = \frac{1}{1 - \sigma(z)} \cdot \frac{\partial(1 - \sigma(z))}{\partial z} = -\sigma(z)$$

כעת יש לשים לב שהנגזרת של $\log p(y = 1|z)$ הינה $1 - \sigma(z) = y - \sigma(z)$, והנגזרת של $\log p(y = 0|z)$ הינה $-\sigma(z)$. לכן אם $y \in \{0,1\}$, אז ניתן לרשום בקיצור: $\frac{\partial}{\partial \theta} \log p(y_i|x_i; \theta) = y_i - \sigma(z)$. במקרה של רגרסיה לוגיסטית, מחפשים את הנגזרת של $\frac{\partial}{\partial \theta} \log p(y_i|x_i; \theta)$, ולפי הפיתוח המקדים ניתן לרשום את זה כך:

$$\frac{\partial}{\partial \theta} \log p(y_i|x_i; \theta) = (y_i - \sigma(w^T x + b)) \cdot \frac{\partial}{\partial w} (w^T x + b) = (y_i - p(y_i = 1|x_i; \theta)) \cdot x_i$$

כעת לאחר הפיתוח ניתן לחזור חזרה לביטוי $\arg \min_{\theta} L(\theta)$ ולהציב:

$$\frac{\partial L(\theta)}{\partial \theta} = -\frac{1}{n} \sum_{i=1}^n \frac{\partial}{\partial \theta} \log p(y_i|x_i; \theta) = -\frac{1}{n} \sum_{i=1}^n (y_i - \sigma(w^T x + b)) x_i = -\frac{1}{n} \sum_{i=1}^n (y_i - p(y_i = 1|\theta; x)) x_i$$

3.2.3 Optimization

בדומה לרגרסיה לינארית, גם כאן חישוב הערך האופטימלי של $\hat{\theta}$ יהיה איטרטיבי בשיטת gradient descent:

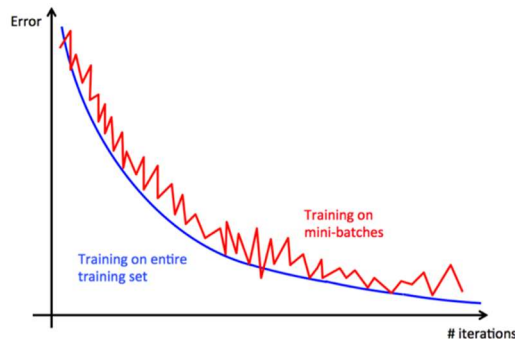
$$\hat{\theta}_{j+1} = \hat{\theta}_j - \epsilon \cdot \frac{\partial}{\partial \theta_j} L(\theta)$$

כאשר ϵ הוא הפרמטר של ה-learning rate. כיוון שפונקציית המחיר $L(\theta)$ קעורה, מובטח שתהיה התכנסות ל- $\hat{\theta}$.

במקרים רבים הדאטה סט הוא גדול, ולחשב את הגרדיאנט עבור כל הדאטה צורך הרבה חישוב. בכל צעד של קידום ניתן לחשב את הגרדיאנט עבור חלק מהדאטה, ולבצע את הקידום לפי הכיוון של הגרדיאנט המתקבל. למשל ניתן לבחור באופן אקראי נקודה אחת ולחשב עליה את הגרדיאנט. בחירה כזו נקראת Stochastic Gradient Descent (SGD), כיוון שבכל צעד יש בחירה אקראית של נקודה. חישוב בשיטת SGD יכול לגרום לשונות גדולה ככל שהחישוב מתקדם, ולכן עדיף לקחת מספר נקודות. חישוב הגרדיאנט בשיטה זו נקרא mini-batch learning (לעומת חישוב המתבצע על כל הדאטה הנקרא batch learning). באופן פורמלי, הגרדיאנט בשיטת mini-batch הינו:

$$\frac{\partial L}{\partial \theta} = \frac{\partial}{\partial \theta} \left[-\frac{1}{|V|} \sum_{i \in v} \log p(y_i|x_i; \theta) \right] \approx \frac{\partial}{\partial \theta} \left[-\frac{1}{n} \sum_{i=1}^n \log p(y_i|x_i; \theta) \right]$$

אמנם כל צעד הוא קירוב לגרדיאנט, אך החישוב מאוד מהיר ביחס לגרדיאנט המדויק, וזה יתרון משמעותי שיש לשיטה זו על פני batch learning. בנוסף, ניתן להוכיח שבשיטה זו מתקבל [משערכ חסר הטיה](#) לגרדיאנט האמיתי.



איור 3.8 השגיאה של $\hat{\theta}$ כפונקציה של האיטרציות בשיטת gradient descent. הגרף הכחול מייצג את השגיאה בשיטת batch learning, בה הגרדיאנט בכל צעד מחושב על כל הדאטה, והגרף האדום מייצג את השגיאה בשיטת mini-batch learning, בה בכל צעד הגרדיאנט מחושב רק על חלק מהדאטה הנבחר באופן אקראי.

בדומה ל-linear regression, גם ב-logistic regression קיים עניין הרגולריזציה, שנועד למנוע מהמודל לתת משקל יתר לכל נקודה (Overfitting) או לא לייצג את הדאטה בצורה מספיק טובה (Underfitting). ניתן להוסיף למשל אילוץ ביחס לריבוע הפרמטר:

$$L(\theta) = -\frac{1}{n} \sum_{i=1}^n \log p(y_i|x_i; \theta) + \lambda \|\theta\|^2$$

ואז הנגזרת הינה:

$$\frac{\partial L}{\partial \theta} = -\frac{1}{n} \sum_{i=1}^n \frac{\partial}{\partial \theta} \log p(y_i|x_i; \theta) + 2\lambda \|\theta\|$$

הפרמטר λ האופטימלי מחושב על ידי ביצוע Cross validation על הדאטה.

3.2.4 SoftMax Regression – Multi Class Logistic Regression

בדומה ל-linear regression, גם ב-logistic regression ניתן להרחיב את המסווג גם עבור multi-class (מקרה בו יש יותר משתי קטגוריות). גם בהכללה למקרה מרובה קטגוריות יש מיפוי של כל קטגוריה להסתברות בתחום $[0, 1]$ רק כעת הפונקציה בה משתמשים היא SoftMax במקום סיגמואיד. SoftMax היא פונקציה המופעלת על סדרה, והיא מוגדרת כך:

$$\text{SoftMax}(z_1, \dots, z_n) = \left(\frac{e^{z_1}}{\sum_{j=1}^n e^{z_j}}, \dots, \frac{e^{z_n}}{\sum_{j=1}^n e^{z_j}} \right)$$

המונה מחשב אקספוננט בחזקת z_i , והמכנה מנרמל את התוצאה, כך שסך כל האיברים לאחר הפונקציה הוא 1. במקרה בו יש מספר קטגוריות – יש מספר קווי הפרדה, ולכל אחד מהם יש סט פרמטרים θ . בהינתן נקודה חדשה, ניתן בעזרת SoftMax לתת הסתברות לכל קטגוריה:

$$p(y = i|x; \theta) = \text{SoftMax}(w_1^T x + b, \dots, w_n^T x + b_n)$$

ואם מעוניינים לקבל סיווג קשה, לוקחים את האיבר בעל ההסתברות הגבוהה ביותר. גם במקרה זה פונקציית המחיר תהיה ה-Cross entropy:

$$L(\theta) = -\frac{1}{n} \sum_{i=1}^n \log p(y_i|x_i; \theta)$$

נחשב את הנגזרת של הביטוי בתוך הסכום לפי θ_i :

$$\begin{aligned} \frac{\partial}{\partial \theta_i} \log p(y_i = s|x_i; \theta) &= \frac{\partial}{\partial \theta_i} \log \frac{\exp(w_s^T x + b)}{\sum_{j=1}^n \exp(w_j^T x + b)} = \frac{\partial}{\partial \theta_i} \left(w_s^T x + b - \log \sum_{j=1}^n \exp(w_j^T x + b) \right) \\ &= 1_{\{i=s\}} x - \frac{\exp(w_i^T x + b) x}{\sum_{j=1}^n \exp(w_j^T x + b)} = (1_{\{i=s\}} - p(y = i|x)) x \end{aligned}$$

כאשר הסימון $1_{\{i=s\}}$ הינו 1 אם $i = s$ ו-0 אחרת. כעת ניתן להציב את הביטוי האחרון בנגזרת של $L(\theta)$:

$$\frac{\partial L}{\partial \theta_i} = -\frac{1}{n} \sum_{t=1}^n (1_{\{y_t=k\}} - p(y_t = i|x_t; \theta)) x$$

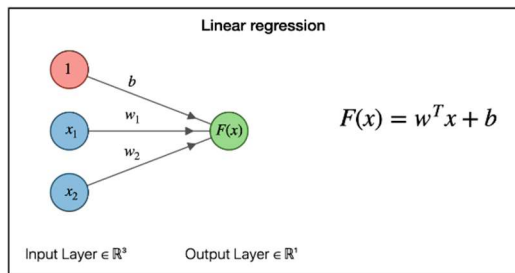
כעת ניתן לחשב את θ האופטימלי בשיטת gradient descent:

$$\theta_{i+1} = \theta_i - \epsilon \frac{\partial L}{\partial \theta}$$

3.2.5 SoftMax Regression as Neural Network

לשיטת logistic regression יש מספר יתרונות: היא יחסית קלה לאימון, מספקת דיוק טוב לדאטה-סטים פשוטים, יציבה ל-overfitting, מציעה סיווג הסתברותי ומתאימה גם למקרה בו יש יותר משתי קטגוריות. עם זאת, יש לה חסרון משמעותי – קווי ההפרדה של המודל הינם לינאריים, וזו הפרדה שאינה מספיק טובה עבור בעיות מורכבות. יש מגוון בעיות בהן על מנת לבנות מודל המסוגל להפריד בין קטגוריות שונות, יש צורך במנגנון הפרדה לא לינארי.

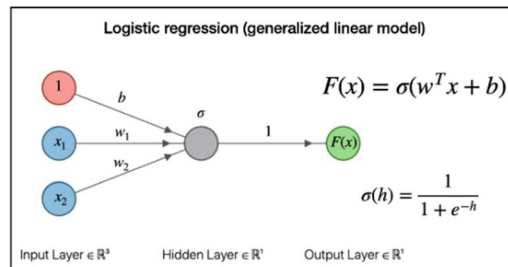
דרך מקובלת לבניית מודלים לא לינאריים היא שימוש ברשתות נוירונים עמוקות, ובכדי להבין את הקונספט שלהן היטב, ראשית יש לייצג את המודלים הלינאריים כשכבה של נוירונים, כאשר המודל הזה שקול לחלוטין לכל מה שהוצג עד כה. בעיית Linear regression לוקחת סט של מאפיינים ומכפילה כל אחד מהם במשקל, ולאחר מכן סוכמת את כל האלמנטים (בצירוף bias) לכדי משתנה יחיד הקובע מה הקטגוריה של סט זה. ניתן לייצג את המודל על ידי התיאור הגרפי הבא:



איור 3.9 ייצוג רגרסיה לינארית כרשת נוירונים עם שכבה אחת.

בתיאור זה יש 2 מאפיינים המהווים את ה-input, וכל אחד מהם מחובר למוצא בתוספת הכפלה במשקל. בנוסף יש bias, ובצירוף המאפיינים המוכפלים במשקלים וה-bias מתקבל המוצא: $F(x) = w^T x + b = w_1 x_1 + w_2 x_2 + b$. כל עיגול באיור נקרא נוירון מלאכותי – אלמנט היכול לקבל קלט, לבצע פעולה חישובית ולהוציא קלט.

רגרסיה לוגיסטית ניתנת לתיאור באופן דומה, כאשר הנוירונים של סט ה-input לא מחוברים ישירות במוצא אלא עוברים דרך סיגמואיד במקרה הבינארי או דרך SoftMax במקרה בו יש יותר משתי קטגוריות:



איור 3.10 ייצוג רגרסיה לוגיסטית כרשת נוירונים עם שכבה אחת.

מלבד המעבר בפונקציית הסיגמואיד, יש הבדל נוסף בין הייצוג של הרגרסיה הלינארית לייצוג של הרגרסיה הלוגיסטית: בעוד הרגרסיה הלינארית מספקת במוצא מספר יחיד במוצא (מסווג קשה), הרגרסיה הלוגיסטית מספקת במוצא וקטור באורך של מספר הקטגוריות, באופן כזה שלכל קטגוריה יש הסתברות מסוימת שה-input שייך לאותה קטגוריה.

בפרק הבא יוצג מבנה בעל מספר שכבות של נוירונים, כאשר בין שכבה לשכבה יש פונקציה לא לינארית. באופן הזה המודל שיתקבל יהיה מיפוי של סט מאפיינים באופן לא לינארי לוקטור הסתברויות במוצא. הגמישות של המודל תאפשר להתמודד עם משימות בעלות דאטה מורכב.

References

<https://www.deeplearningbook.org/>

Fitting:

<https://www.calloftechies.com/2019/08/solving-overfitting-underfitting-in-machine-learning.html>

Cross validation:

https://scikit-learn.org/stable/_images/grid_search_cross_validation.png

linear regression:

מצגות מהקורס של פרופ' יעקב גולדברגר

<https://joshuagoings.com/2020/05/05/neural-network/>

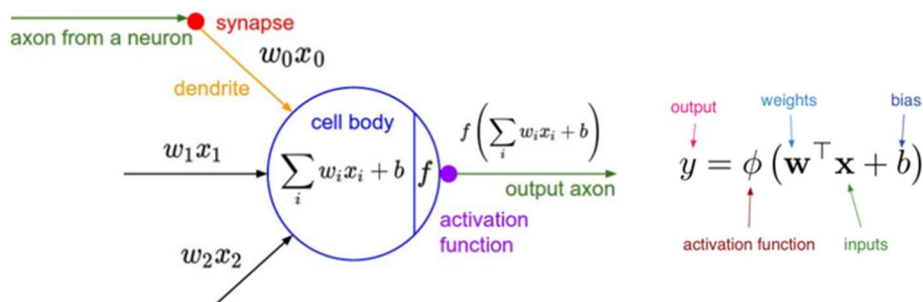
4. Deep Neural Networks

פרק זה עוסק ברשתות נוירונים עמוקות. רשת נוירונים הינה חיבור של יחידות עיבוד בסיסיות (נוירונים מלאכותיים) על ידי משקלים ופונקציות לא לינאריות. רשת נוירונים נקראת עמוקה אם היא מכילה יותר משכבה חבויה אחת. לאחר הצגת הבסיס הרעיוני והפורמלי, יוסבר כיצד ניתן לחשב את המשקלים של הרשת בצורה יעילה בעזרת מבנה המכונה Computational Graph. לאחר מכן יוצגו שני תחומים העוסקים בשיפור הרשת – שיטות אופטימיזציה לתהליך הלמידה ושיטות לבחון עד כמה המודל המתקבל אכן מכיל בצורה טובה את הדאטה עליו הוא מאומן.

4.1 Multilayer Perceptron (MLP)

4.1.1 From a Single Neuron to Deep Neural Network

ראשית יש לתאר את המבנה של יחידת העיבוד הבסיסית – נוירון מלאכותי. יחידת עיבוד זו נקראת כך עקב הדמיון שלה לנוירון פיזיולוגי – יחידת העיבוד הבסיסית במח האדם האנושי. הנוירון יכול לקבל מספר קלטים ולחבר אותם, ואז להעביר את התוצאה בפונקציית הפעלה (activation function) שאינה בהכרח לינארית. באופן סכמתי ניתן לתאר את הנוירון הבודד כך:

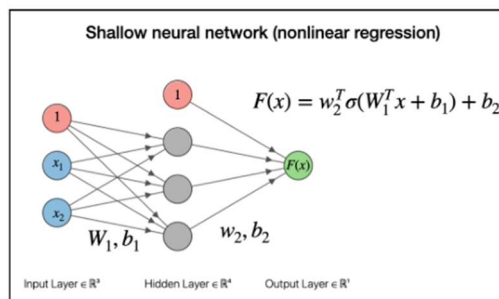


איור 4.1 ייצוג של נוירון מלאכותי, המקבל קלט, סוכם אותו ומעביר את התוצאה בפונקציית הפעלה.

הקלט של הנוירון הוא סט input מוכפל במשקלים: כאשר $w, x \in \mathbb{R}^d$, ואיבר bias. הקלט עובר דרך סוכם, ומתקבל הביטוי $\sum_{i=1}^d w_i x_i + b$. לאחר מכן הסוכם עובר דרך פונקציית הפעלה, ומתקבל המוצא $f(\sum_{i=1}^d w_i x_i + b)$. במקרה הפרטי בו פונקציית הפעלה היא סיגמואיד/SoftMax והמוצא לא מחובר לשכבה נוספת, אז למעשה מקבלים את הרגרסיה הלוגיסטית.

במקרה בו הנוירונים המחוברים ל-input אינם מהווים את המוצא אלא הם מוכפלים במשקלים ומתחברים לשכבה נוספת של נוירונים, אז השכבה המחוברת ל-input נקראת שכבה חבויה (hidden layer). אם יש יותר משכבה חבויה אחת, הרשת מכונה רשת נוירונים עמוקה. במקרה בו יש לפחות שכבה חבויה אחת, הקשר בין הכניסה למוצא אינו לינארי, וזה היתרון שיש למודל זה. נתבונן במקרה של שכבה חבויה ונחשב את הקשר בין הכניסה למוצא: נסמן את המשקלים בין הכניסה לבין השכבה החבויה ב- w_1, b_1 ואת המשקלים בין השכבה החבויה לבין המוצא ב- w_2, b_2 , ונקבל שלאחר השכבה החבויה מתקבל הביטוי: $w_2 \cdot f_1(w_1^T x + b_1) + b_2$. ביטוי זה עובר בפונקציית הפעלה נוספת ומתקבל המוצא:

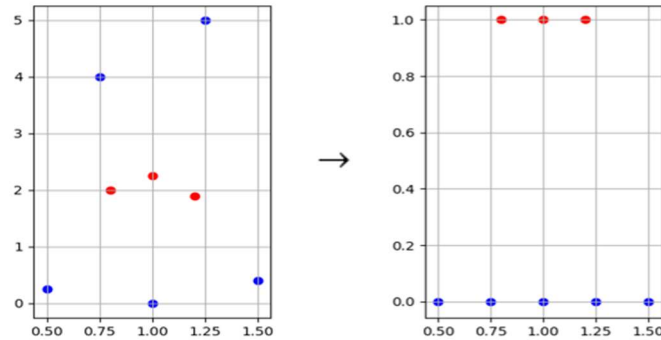
$$\hat{y} = f_2(w_2 \cdot f_1(w_1^T x + b_1) + b_2)$$



איור 4.2 רשת נוירונים בעלת שכבה חבויה אחת.

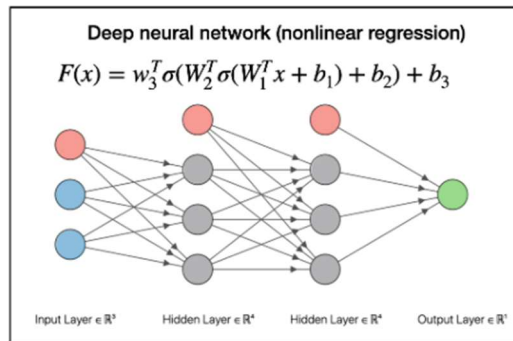
חשוב להדגיש שמטרת הרשת היא לבצע פעולות לא לינאריות על ה-input כך שהוא יסודר באופן חדש הניתן להפרדה לינארית. למעשה לא מבטלים את ההפרדה הלינארית הנעשית בעזרת הרגרסיה, אלא מבצעים לפניה שלב מקדים של העתקה לא לינארית. תהליך זה נקרא למידת ייצוגים (representation learning), כאשר בכל שכבה

מנסים ללמוד ייצוג פשוט יותר לדאטה על מנת שהוא יוכל להיות מופרד באופן לינארי. המיקוד של הרשת הוא אינו במשימת סיווג אלא במשימת ייצוג, כך שבסופו של דבר ניתן יהיה לסווג את הדאטה בעזרת סיווג לינארי פשוט (גרסיה לינארית או לוגיסטית).



איור 4.3 העתקה לא לינארית של דוגמאות על ידי המשוואה $\tilde{y} = \begin{cases} 1, & \text{if } 3 \leq (x^2 + y^2) \leq 8 \\ 0, & \text{else} \end{cases}$. העתקה זו מאפשרת להבחין בין הדוגמאות בעזרת קו הפרדה לינארי.

כאשר מחברים יותר משכבה חבויה אחת, מקבלים רשת עמוקה. החיבור בין השכבות נעשה באופן זהה – הכפלה של משקלים, סכימה והעברה בפונקציית הפעלה.



איור 4.4 רשת נירונים בעלת שתי שכבות חבויות.

רשת נירונים בעלת לפחות שכבה חבויה אחת הינה Universal approximation, כלומר, ניתן לייצג בקירוב כל התפלגות מותנית בעזרת הארכיטקטורה הזו. ככל שהרשת יותר עמוקה, כך היכולת שלה להשיג דיוק טוב יותר גדלה.

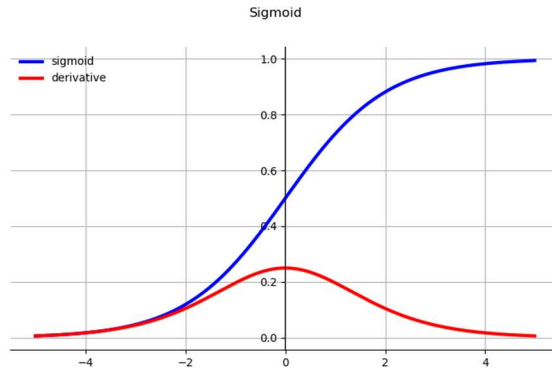
4.1.2 Activation Function

האלמנט המרכזי בכל נירון הוא פונקציית הפעלה, ההופכת אותו ליחידת עיבוד לא לינארית. יש מספר פונקציות הפעלה מקובלות – Sigmoid, tanh, ReLU.

Sigmoid

פונקציית הסיגמואיד הוצגה בפרק של גרסיה לוגיסטית, וכעת נרחיב עליה. הפונקציה והנגזרת שלה הן מהצורה:

$$\sigma(z) = \frac{1}{1 + e^{-z}}, \frac{\partial}{\partial z} \sigma(z) = \sigma(z)(1 - \sigma(z))$$



איור 4.5 פונקציית סיגמואיד והנגזרת שלה.

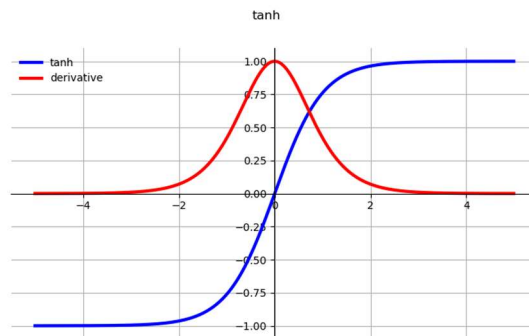
יש לפונקציה זו שלושה חסרונות:

- א. עבור ערכים גדולים, הנגזרת שואפת ל-0. זה כמובן יוצר בעיה בחישוב הפרמטר האופטימלי בשיטת Gradient descent, שהרי בכל צעד התוספת תלויה בגרדיאנט, ואם הוא מתאפס – לא ניתן לחשב את הפרמטר האופטימלי.
- ב. הסיגמואיד לא ממורכז סביב ה-0, וזה יוצא בעיה עבור דאטה שאינו מנורמל.
- ג. הן הפונקציה והן הנגזרת דורשות חישוב של אקספוננט, ובאופן יחסי זו פעולה יקרה לחישוב.

tanh

פונקציית טנגנס היפרבולי הינה מהצורה:

$$\tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}, \quad \frac{\partial}{\partial z} \tanh(z) = 1 - (\tanh(z))^2$$



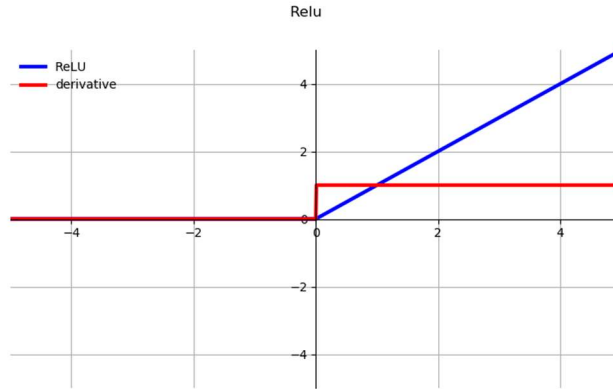
איור 4.6 פונקציית טנגנס היפרבולי והנגזרת שלה.

גם בפונקציה זו יש את הבעיות של חישוב אקספוננט והתאפסות הגרדיאנט עבור ערכים גדולים, אך היתרון שלה הוא שהיא ממורכזת סביב 0.

ReLU (Rectified Linear Unit)

פונקציית Relu מאפסת ערכים שלילים ואדישה כלפי ערכים חיוביים. הפונקציה מחזירה את המקסימום מבין המספר שהיא מקבלת ובין 0. באופן פורמלי צורת המשוואה הינה:

$$ReLU(z) = \max(0, z), \quad \frac{\partial}{\partial z} ReLU(z) = 1_{\{z>0\}} = \begin{cases} 1, & z > 0 \\ 0, & z \leq 0 \end{cases}$$



איור 4.7 פונקציית ReLU והנגזרת שלה.

פונקציית ReLU יעילה יותר לחישוב מהפונקציות הקודמות, כיוון שיש בה רק בדיקה של סימן המספר, ואין בה כפל או אקספוננט. בנוסף, בפונקציה זו הגרדיאנט לא מתאפס בערכים גבוהים. יתרון נוסף שיש לפונקציה זו – היא מתכנסת יותר מהר מהפונקציות הקודמות (x6). לפונקציה יש שני חסרונות עיקריים: היא לא ממורכזת סביב 0, ועבור אתחול משקלים לא טוב מרבית הנירונים מתאפסים וזה יחסית בזבזני. כדי להתגבר על הבעיה האחרונה ניתן להשתמש בוורסיות של הפונקציה, כמו למשל PReLU ו-ELU:

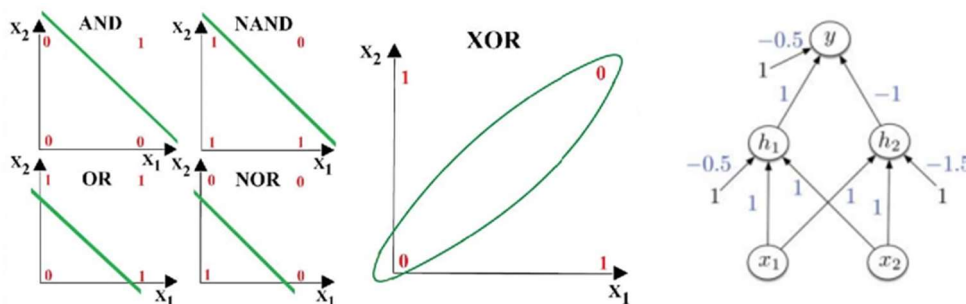
$$PReLU(x) = \max(\alpha x, x), ELU(x) = \begin{cases} x, & x > 0 \\ \alpha(e^x - 1) & \end{cases}$$

בפונקציית PReLU, המקרה הפרטי בו $\alpha = 0.01$ נקרא Leaky ReLU. בפונקציית ELU, הפרמטר α הוא פרמטר נלמד.

ישנן עוד פונקציות, אך אלה הן העיקריות, כאשר לרוב מקובל להשתמש ב-ReLU ובוורסיות שלו.

4.1.3 Xor

אחת הדוגמאות הידועות ביותר שאינן ניתנות להפרדה לינארית היא בעיית ה-Xor. יש שתי כניסות - x_1, x_2 והמוצא הוא 0 אם הכניסות שוות ו-1 אם הן שונות. פונקציה זו ממפה שתי כניסות ליציאה, כאשר יש שתי קטגוריות במוצא, ואין אפשרות להעביר קו לינארי שיבחין בין הדוגמאות השונות. לעומת זאת, ניתן לבצע שלב מקדים של הפרדה לא לינארית, ולאחריה ניתן יהיה לבנות מסווג על בסיס קו הפרדה לינארי.



איור 4.8 אופרטור Xor אינו ניתן להפרדה לינארית, בשונה משאר האופרטורים הלוגיים. בעזרת רשת ניורונים בעלת שכבה חבויה אחת ניתן לייצר מודל שקול לאופרטור Xor.

בדוגמה המובאת באיור הכניסות עוברות דרך שכבה חבויה אחת בעלת שני ניורונים - h_1, h_2 , המקבלים בנוסף גם bias. פונקציית ההפעלה של ניורונים אלו היא פונקציית הסימן, וניתן לכתוב את המוצא של שכבה זו כך:

$$h_1 = \text{sign}(x_1 + x_2 - 0.5), h_2 = \text{sign}(x_1 + x_2 - 1.5)$$

לאחר השכבה החבויה הנירונים מחוברים למוצא, שגם לו יש bias, והסכום של הכניסות וה-bias עוברים במסווג:

$$y = \text{sign}(h_1 - h_2 - 0.5) = \begin{cases} 1 & \text{if } h_1 - h_2 - 0.5 > 0 \\ 0 & \text{if } h_1 - h_2 - 0.5 < 0 \end{cases}$$

נבחן את המשמעות של הנירונים: הנירון h_1 יהיה 0 אם שתי הכניסות שוות 0, אחרת הוא יהיה שווה 1. הנירון h_2 יהיה שווה 1 אם שתי הכניסות שוות 1, ובכל מצב אחר הוא יהיה שווה 0. באופן הזה לאחר השכבה החבויה הראשונה,

אם גם h_1 שונה מ-0 אז יש לפחות כניסה אחת ששווה 1, וצריך לבדוק בעזרת h_2 את המצב של הכניסה השנייה. אם גם הכניסה השנייה שווה 1, אז בכניסה של y (יחד עם ה-bias) יתקבל מספר שלילי, ובמוצא יתקבל 0. אם הכניסה השנייה היא 0, אז $y = \text{sign}(0.5) = 1$. במצב בו שתי הכניסות הן 0, יתקיים $h_1 = h_2 = 0$, ואז רק ה-bias ישפיע, וכיוון שהוא שלילי שוב יתקבל 0 במוצא.

נרשום בפירוט את הערכים בכל שלב, עבור על הכניסות האפשריות:

x_1	x_2	h_1	h_2	$h_1 - h_2 - 0.5$	y
0	0	0	0	-0.5	0
0	1	1	0	0.5	1
1	0	1	0	0.5	1
1	1	1	1	-1.5	0

4.2 Computational Graphs and propagation

4.2.1 Computational Graphs

כפי שהוסבר לעיל, רשת נזירונים עמוקה היא רשת בעלת לפחות שכבה עמוקה אחת, והמטרה של כל שכבה היא ללמוד ייצוג פשוט יותר של המידע שנכנס אליה, כך שבסופו של דבר ניתן יהיה להבחין בין קטגוריות שונות בעזרת הפרדה לינארית. מה שקובע את השינוי של הדאטה במעבר שלו ברשת הם המשקלים והנזירונים המבצעים פעולות לא לינאריות. בעוד הפעולות אותן מבצעים הנזירונים קבועות (סכימה ולאחר מכן פונקציית הפעלה), המשקלים נקבעים בהתחלה באופן אקראי, ובעזרת הדוגמאות הידועות ניתן לאמן את הרשת ולשנות את המשקלים כך שיבצעו את למידת הייצוג החדש בצורה אופטימלית.

תהליך האימון מתבצע בשני שלבים – ראשית מכניסים דוגמא ידועה לתחילת הרשת ו"מפעפעים" אותה עד למוצא (Forward propagation), כלומר, מחשבים את השינוי שהיא עוברת כאשר היא מוכפלת במשקלים ועוברת בנזירונים החבויים. לאחר שמגיעים למוצא, משווים את מה שהתקבל למה שאמור להיות במוצא לפי מה שידוע על דוגמא זו, ואז מבצעים פעפוע לאחור (Backward propagation), שמטרתו לתקן את המשקלים בהתאם למה שהתקבל במוצא. השלב השני הוא למעשה חישוב יעיל של GD על פני כל שכבות הרשת – מחשבים את הנגזרת בין המשקל w_i לבין פונקציית המחיר $L(\theta)$, ואז מבצעים עדכון בשיטת GD – $w_{i+1} = w_i - \epsilon \frac{\partial L(\theta)}{\partial w_i}$. כיוון שהרשת יכולה להכיל מיליוני משקלים, יש למצוא דרך יעילה לחישוב הגרדיאנט עבור כל משקל.

נח לעשות את התהליך הדו-שלבי הזה בעזרת Computational Graphs, שזהו למעשה גרף הבנוי מצמתים המייצגים את התהליך שהדאטה עובר בתוך הרשת. הגרף יכול לייצג כל רשת, וניתן באמצעותו לחשב נגזרות מורכבות באופן פשוט יחסית. לאחר השלב הראשון בו מעבירים דוגמא בכל חלקי הגרף, ניתן למשל לחשב את השגיאה הריבועית הממוצעת $(\hat{y} - y)^2$, להגדיר אותה כפונקציית המחיר, ולמצוא את הנגזרת של כל משקל לפי פונקציה זו – $\frac{\partial L(\theta)}{\partial w_i}$, כאשר הנגזרות החלקיות מחושבות בעזרת כלל השרשרת.

4.2.2 Forward and Backward propagation

באופן פורמלי, עבור N משקלים התהליך מנוסח כך:

Forward pass:

For i in 1 ... N:

Compute w_i as function of $w_0 \dots w_{i-1}$

Backward pass:

$$\bar{w}_N = 1$$

For i in N - 1 ... 1:

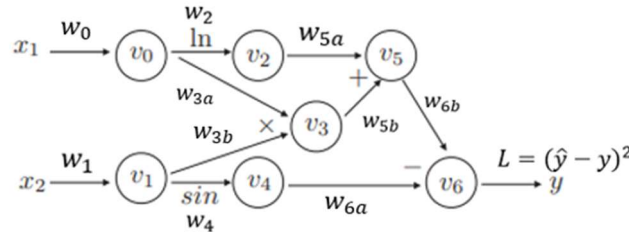
$$\frac{\partial L}{\partial w_i} = \frac{\partial L}{\partial w_N} \cdot \frac{\partial w}{\partial w_{N-1}} \dots \frac{\partial w_{i+1}}{\partial w_i}$$

$$\bar{w}_i = w_i - \epsilon \frac{\partial L}{\partial w_i}$$

בשלב הראשון מחשבים כל צומת על סמך הצמתים הקודמים לו, ובשלב השני בו חוזרים אחורה, מחשבים את הנגזרת של כל משקל בעזרת כלל השרשרת החל מהמוצא ועד לאותו משקל, ומעדכנים את המשקל. נסתכל למשל בדוגמה הבאה:

$$y = f(x_1, x_2) = \ln(x_1) + x_1 x_2 - \sin(x_2)$$

לפונקציה זו שתי כניסות, העוברות כל אחת בנפרד דרך פונקציה לא ליניארית, ובנוסף מוכפלות אחת בשנייה. באופן גרפי ניתן לאייר את הפונקציה כך:



איור 4.9 הפונקציה $y = f(x_1, x_2) = \ln(x_1) + x_1 x_2 - \sin(x_2)$ מתוארת באופן גרפי.

בגרף זה יש 7 צמתים:

$$v_0 = x_1, v_1 = x_2$$

$$v_2 = \ln(v_0), v_3 = v_0 \cdot v_1, v_4 = \sin(v_1)$$

$$v_5 = v_2 + v_3$$

$$\hat{y} = v_6 = v_5 - v_4$$

לאחר שבוצע החישוב עבור \hat{y} , ניתן לחשב את אחורה את הנגזרות החלקיות, בעזרת כלל השרשרת:

$$\frac{\partial L}{\partial w_{6a}} = -1, \frac{\partial L}{\partial w_{6b}} = -1$$

$$\frac{\partial L}{\partial w_{5a}} = \frac{\partial L}{\partial w_{6b}} \frac{\partial w_{6b}}{\partial w_{5a}} = -1 \cdot 1 = 1, \quad \frac{\partial L}{\partial w_{5b}} = \frac{\partial L}{\partial w_{6b}} \frac{\partial w_{6b}}{\partial w_{5b}} = -1 \cdot 1 = -1$$

$$\frac{\partial L}{\partial w_4} = \frac{\partial L}{\partial w_{6a}} \frac{\partial w_{6a}}{\partial w_4} = -1 \cdot (-\cos w_4) = \cos w_4$$

$$\frac{\partial L}{\partial w_{3a}} = \frac{\partial L}{\partial w_{6b}} \frac{\partial w_{6b}}{\partial w_{5b}} \frac{\partial w_{5b}}{\partial w_{3a}} = -1 \cdot 1 \cdot w_{3b}, \quad \frac{\partial L}{\partial w_{3b}} = \frac{\partial L}{\partial w_{6b}} \frac{\partial w_{6b}}{\partial w_{5b}} \frac{\partial w_{5b}}{\partial w_{3b}} = -1 \cdot 1 \cdot w_{3a}$$

$$\frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial w_{6b}} \frac{\partial w_{6b}}{\partial w_{5a}} \frac{\partial w_{5a}}{\partial w_2} = -1 \cdot 1 \cdot \frac{1}{\ln w_2}$$

המשקלים בכניסה, w_0, w_1 , רק מעבירים ללא שינוי את הכניסות לצמתים v_0, v_1 , לכן הם שווים 1.

לאחר שכל הנגזרות החלקיות חושבו, ניתן לעדכן את המשקלים לפי העיקרון של GD: $w_{i+1} = w_i + \epsilon \frac{\partial L}{\partial w_i}$.

היתרון הגדול של חלוקת הרשת לגרף עם צמתים נובע מכך שכאשר כותבים את הנגזרת של $L(\theta)$ בעזרת כלל השרשרת, אז כל איבר בשרשרת בפני עצמו הוא יחסית פשוט לחישוב. למשל – נגזרת של חיבור היא 1, נגזרת של כפל היא המקדם של המשתנה לפיו גוזרים, וכן באותו אופן עבור כל אופרטור שמפעילים בצומת מסוים. לשיטה זו קוראים backpropagation והיא מאוד נפוצה ברשתות עמוקות עקב יעילותה בחישוב המשקלים. בשונה מבעיות רגרסיה, חישוב האופטימום ברשתות עמוקות היא לא בעיה קמורה, ולכן לא תמיד יש לה בהכרח מינימום גלובאלי.

עם זאת, עדכון המשקלים בשיטת Back propagation הוכיח את עצמו, למרות שהמשקלים לא בהכרח הגיעו לאופטימום שלהם.

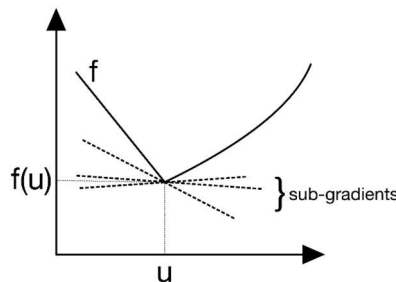
4.2.3 Back Propagation and Stochastic Gradient Descent

כיוון שהנושא של backpropagation הוא מאוד בסיסי ברשתות נירונים, נרחיב עליו את הדיבור ונבסס את העקרונות המתמטיים שלו בצורה יותר עמוקה.

הקדמה – סאב גרדיאנט: נאמר ש- $g \in V$ הוא סאב-גרדיאנט של פונקציה $f: V \rightarrow \mathbb{R}$ אם לכל $v \in V$ מתקיים:

$$f(v) \geq f(u) + \langle g, v - u \rangle$$

קבוצת כל הסאב-גרדיאנטים של f בנקודה u מסומנת ב- $\partial f(u)$. באופן גאומטרי, $\partial f(u)$ היא קבוצת כל הישרים שנמצאים מתחת לגרף f בנקודה u . בפרט, עבור פונקציה f קמורה וגזירה בנקודה u ישנו רק ישר אחד מתחת לגרף – זהו הישר המשיק ל- f , דהיינו $\partial f(u) = \{\nabla f(u)\}$.



הסאב-גרדיאנט משמש כהכללה למושג הגרדיאנט כאשר אנו עוסקים בפונקציות שאינן בהכרח גזירות, ולעיתים שימושי גם באיטרציה של Stochastic Gradient Descent (SGD) כאשר עוסקים בפונקציה שאיננה גזירה אנליטית. את הסאב-גרדיאנט של פונקציית מחיר l התלויה במשקולות w ומשוערך בנקודה (x, y) נסמן ב- $g \in \partial l(w, (x, y))$.

כדוגמה קצרה לפני שנמשיך, ניקח למשל את פונקציית הערך המוחלט: $f(v) = |v|$. לכל $v > 0$ הסאב-גרדיאנט יחיד $\partial f(v > 0) = \{+1\}$ וכנ"ל עבור הנקודות שמקיימות $v < 0$, עבורן $\partial f(v < 0) = \{-1\}$. בנקודה $v = 0$ הפונקציה אינה גזירה, אך נוכל לרשום מפורשות ש- g הוא סאב-גרדיאנט של f אם מתקיים $|u| \geq gu$, מה שנוכח רק אם $g \in [-1, 1]$. משום כך נסיק $\partial f(0) \in [-1, 1]$.

אלגוריתם backpropagation: נהוג לסמן איטרציית SGD בצורתה הכללית על ידי $w^{(t+1)} = w^{(t)} - \eta_{n+1} g^{(t)}$, כאשר g הוא סאב-גרדיאנט של פונקציית הלוס l , דהיינו $g^{(t)} \in \partial l(w^{(t)}, (x_t, y_t))$ למען הפשטות אנו נניח כרגע ש $g^{(t)} = \nabla l$ (כלומר, אנו נניח שלפונקציית הלוס יש גרדיאנט) ונתעמק בדרך החישוב המהירה של הביטוי הנ"ל, הידועה בתור אלגוריתם backpropagation.

המסגרת הכללית שלנו היא רשת נירונים בסיסית (MLP) ופונקציית מחיר l_2 (-כלומר ריבוע ההפרש בין הפלט של הרשת לבין הפלט האמיתי). הצעד הראשון בתהליך יהיה להתחיל מרשת בסיסית עם שכבת קלט בעלת נירון יחיד, 2 שכבות נסתרות המכילות כל אחת נירון יחיד, ושכבת פלט המכילה גם היא נירון בודד. לכל נירון (פרט לנירון הקלט) יש גם bias (אלה הם b_i), וכל זוג נירונים מחוברים באמצעות קשת עם משקל (אלה הם w_i), כך שבסך הכל, פונקציית המחיר שלנו L היא פונקציה של המשתנים הבאים: $L = L(w_1, b_1, w_2, b_2, w_3, b_3)$. כזכור, במוצא של כל נירון יש פונקציית אקטיבציה (בדרך כלל לא לינארית), והמטרה הכללית היא למצוא את ערכי w, b שמביאים את השגיאה של L למינימום.



איור 4.10 רשת נירונים עם קלט יחיד, שתי שכבות חבויות בעלות נירון בודד בכל אחת מהן, ומוצא בעל נירון יחיד. לכל אחד מהנירונים פרט לכניסה יש גם bias.

נתמקד בקשר בין 2 הנירונים האחרונים, כאשר נסמן את האקטיבציה של הנירון ברמה ה- i ב- $a^{(i)}$. השגיאה של הרשת בנקודה ה-0 בדאטה (למשל התמונה הראשונה מתוך 50,000) היא:

$$L_0(\dots) = (a^{(o)} - y)^2, (o \text{ is output})$$

כאשר נוכל לסמן את המוצא של האקטיבציה ברמה i באמצעות האקטיבציה של הרמה הקודמת והערכים של הנירון הנוכחי:

$$a^{(i)} = \sigma(w^{(i)}a^{(i-1)} + b^{(i)}) = \sigma(z^{(L)})$$

המטרה כעת היא להבין כמה פונקציית המחיר משתנה ביחס למשקל, כדי שנוכל לעדכן את המשקלים בהתאם:

$$\frac{\partial L_0}{\partial w^{(o)}} = \frac{\partial z^{(o)}}{\partial w^{(o)}} \cdot \frac{\partial a^{(o)}}{\partial z^{(o)}} \cdot \frac{\partial L_0}{\partial a^{(o)}}$$

נחשב את הביטוי באופן מפורש:

$$L = (a^{(o)} - y)^2 \rightarrow \frac{\partial L}{\partial a^{(o)}} = 2(a^{(o)} - y)$$

$$a^{(o)} = \sigma(z^{(o)}) \rightarrow \frac{\partial a^{(o)}}{\partial z^{(o)}} = \sigma'(z^{(o)})$$

$$z^{(o)} = w^{(o)}a^{(o-1)} + b^{(o)} \rightarrow \frac{\partial z^{(o)}}{\partial w^{(o)}} = a^{(o-1)}$$

ולכן נקבל:

$$\frac{\partial L}{\partial w^{(o)}} = a^{(o-1)} \sigma'(z^{(o)}) 2(a^{(o)} - y)$$

עבור כל הדאטה שלנו, אנו לוקחים ממוצע משוקלל:

$$\frac{\partial L}{\partial w^o} = \frac{1}{n} \sum_{k=0}^{n-1} \frac{\partial L_k}{\partial w^{(o)}}$$

הביטוי הזה הוא רק אחד מרכיבי הגרדיאנט אותו אנו רוצים לחשב (במודגש):

$$\nabla L = \left(\frac{\partial L}{\partial w^{(o)}}, \frac{\partial L}{\partial b^{(o)}}, \dots, \frac{\partial L}{\partial w^{(o)}}, \frac{\partial L}{\partial b^{(o)}} \right)^T$$

באופן דומה נוכל לרשום את הגרדיאנט של איבר ה-bias:

$$\frac{\partial L_0}{\partial b^o} = \frac{\partial z^{(o)}}{\partial b^{(o)}} \cdot \frac{\partial a^{(o)}}{\partial z^{(o)}} \cdot \frac{\partial L_0}{\partial a^{(o)}}$$

רק האיבר הראשון במכפלה (הנגזרת של z לפי b) משתנה, והיתר נשארים זהים:

$$z^{(o)} = w^{(o)}a^{(o-1)} + b^{(o)} \rightarrow \frac{\partial z^{(o)}}{\partial b^{(o)}} = 1$$

ולכן נקבל:

$$\frac{\partial L_0}{\partial b^{(o)}} = 1 \cdot \sigma'(z^{(o)}) \cdot 2(a^{(o)} - y)$$

והממוצע על פני כל הדאטה:

$$\frac{\partial L}{\partial b^{(o)}} = \frac{1}{n} \sum_{k=0}^{n-1} \frac{\partial L_k}{\partial b^{(o)}}$$

כל הפיתוח שעשינו מתייחס רק לנוירון של הפלט, וחישובנו כיצד פונקציית המחיר מושפעת משינויים בקלטים של נוירון זה. נרצה להכליל את הביטויים גם עבור יתר השכבות ברשת. נתבונן למשל על השכבה אחת לפני אחרונה:

$$\frac{\partial L_0}{\partial w^{(o-1)}} = \frac{\partial z^{(o-1)}}{\partial w^{(o-1)}} \cdot \frac{\partial a^{(o-1)}}{\partial z^{(o-1)}} \cdot \frac{\partial L_0}{\partial a^{(o-1)}}$$

שני האיברים הראשונים ניתנים לחישוב באופן מפורש:

$$\frac{\partial z^{(o-1)}}{\partial w^{(o-1)}} = a^{(o-2)}$$

$$\frac{\partial a^{(o-1)}}{\partial z^{(o-1)}} = \sigma'(z^{(o-1)})$$

האיבר האחרון במכפלה ניתן לחישוב בעזרת כלל השרשרת:

$$\frac{\partial L_0}{\partial a^{(o-1)}} = \frac{\partial z^{(o)}}{\partial a^{(o-1)}} \cdot \frac{\partial a^{(o)}}{\partial z^{(o)}} \cdot \frac{\partial L_0}{\partial a^{(o)}} = w^{(o)} \cdot \sigma'(z^{(o)}) \cdot 2(a^{(o)} - y)$$

פרט לביטוי $\frac{\partial z^{(o)}}{\partial a^{(o-1)}}$, את היתר חישובנו בשלבים קודמים, וכיוון שביטוי זה שווה בדיוק ל- $w^{(o)}$, נוכל לרשום:

$$\frac{\partial L_0}{\partial w^{(o-1)}} = a^{(o-2)} \cdot \sigma'(z^{(o-1)}) \cdot w^{(o)} \cdot \sigma'(z^{(o)}) \cdot 2(a^{(o)} - y)$$

אם ננסח זאת במילים – לצורך החישוב של $\frac{\partial L_0}{\partial w^{(o-1)}}$ היינו צריכים לדעת את $\frac{\partial L_0}{\partial a^{(o-1)}}$, ואף הוא נתון לנו על ידי החישובים שביצענו באיטרציה הקודמת.

עד כה התייחסנו לרשת נוירונים בה בכל שכבה יש נוירון בודד. כעת נרחיב את הדיון גם למקרים בהם יש שכבות אם יותר מנוירון אחד. מלבד האינדקס העליון שיש לכל איבר (המייצג את השכבה), נוסיף לכל איבר עוד משתנה (sub script) שמייצג את מספר הנוירון באותה שכבה. הביטוי של L_0 יחושב דומה, אלא שכעת יש לקחת בחשבון את כל הנוירונים ברמה האחרונה (נניח שיש n_o כאלה):

$$L_0 = \sum_{j=0}^{n_o-1} (a_j^{(o)} - y_j)^2$$

כעת נסמן את המשקל בין $a_k^{(o-1)}$ ו- $a_j^{(o)}$ ב- $w_{jk}^{(L)}$. בהתאם, כל אקטיבציה תהיה מוגדרת כך:

$$a_j^{(o)} = \sigma \left(w_{j,0}^{(o)} a_0^{(o-1)} + \dots + w_{j,n_o-1}^{(o)} a_{n_o-1}^{(o-1)} + b_j^{(o)} \right) = \sigma \left(z_j^{(o)} \right)$$

נשים לב שהמשקלים מייצגים את כל הצלעות בין הנוירון j - ברמה o - לבין כל הנוירונים שברמה הקודמת – $(o-1)$:

$$z_j^{(o)} = \sum_{k=0}^{n_{o-1}-1} w_{jk}^{(o)} a_k^{(o-1)} + b_j^{(o)}$$

בשלב זה כלל השרשרת ייראה כך:

$$\frac{\partial L_0}{\partial w_{jk}^{(o)}} = \frac{\partial z_j^{(o)}}{\partial w_{jk}^{(o)}} \cdot \frac{\partial a_j^{(o)}}{\partial z_j^{(o)}} \cdot \frac{\partial L_0}{\partial a_j^{(o)}}$$

כאשר:

$$\frac{\partial z_j^{(o)}}{\partial w_{jk}^{(o)}} = a_k^{(o-1)}, \quad \frac{\partial a_j^{(o)}}{\partial z_j^{(o)}} = \sigma'(z_j^{(o)}), \quad \frac{\partial L_0}{\partial a_j^{(o)}} = 2 \cdot (a_j^{(o)} - y_j)$$

לכן בסך הכל נקבל שעבור דוגמה בודדת, הנגזרת של פונקציית המחריר ביחס למשקלים ברמה o הינה:

$$\frac{\partial L_0}{\partial w_{jk}^{(o)}} = a_k^{(o-1)} \cdot \sigma'(z_j^{(o)}) \cdot 2(a_j^{(o)} - y_j)$$

וכאשר ממצעים את הנגזרת עבור n דוגמאות:

$$\frac{\partial L}{\partial w_{jk}^{(o)}} = \frac{1}{n} \sum_{i=0}^{n-1} \frac{\partial L_i}{\partial w_{jk}^{(o)}}$$

ועבור ה-bias:

$$\frac{\partial L_0}{\partial b_j^{(o)}} = \frac{\partial z_j^{(o)}}{\partial b_j^{(o)}} \cdot \frac{\partial a_j^{(o)}}{\partial z_j^{(o)}} \cdot \frac{\partial L_0}{\partial a_j^{(o)}} = 1 \cdot \sigma'(z_j^{(o)}) \cdot 2(a_j^{(o)} - y_j)$$

$$\frac{\partial L}{\partial b_j^{(o)}} = \frac{1}{n} \sum_{i=0}^{n-1} \frac{\partial L_i}{\partial b_j^{(o)}}$$

כזכור, אינו בחלק הקודם (בדיון על רשת בסיסית) שלצורך החישוב של $\frac{\partial L_0}{\partial w^{(o-1)}}$ היינו צריכים לדעת את $\frac{\partial L_0}{\partial a^{(o-1)}}$, אך ביטוי זה היה נתון לנו מחישובים שביצענו באיטרציות קודמות. למעשה מה שהתקבל כאן הוא שהשינוי של פונקציית המחריר כפונקציה של משקלי הרשת תלוי בשינוי של פונקציית המחריר כפונקציה של האקטיבציות של השכבות הקודמות, ביטוי אשר חישבנו מפורשות בכל שלב. אם כן, יש לנו צורך מובהק בחישוב הקשר $\frac{\partial L_0}{\partial a_k^{(o-1)}}$ ולשם כך נרשום:

$$\frac{\partial C_0}{\partial a_k^{(o-1)}} = \sum_{j=0}^{n_L-1} \frac{\partial z_j^{(o)}}{\partial a_k^{(o-1)}} \cdot \frac{\partial a_j^{(o)}}{\partial z_j^{(o)}} \cdot \frac{\partial L_0}{\partial a_j^{(o)}}$$

שהרי בשונה מהמקרה בו יש רק קדקוד אחד, התלות של L_0 באקטיבציה של $a_k^{(o-1)}$ היא ביטוי של כל הניורונים המחוברים אליה בשכבה הבאה, ולא רק לאחד.

נסכם את הכל באלגוריתם:

לעדכון המשקל של השכבה ה- l :

$$\frac{\partial L_0}{\partial w_{jk}^{(l)}} = a_k^{(l-1)} \cdot \sigma'(z_j^{(l)}) \cdot \frac{\partial L_0}{\partial a_j^{(l)}}, \quad \text{average} \rightarrow \frac{\partial C}{\partial w_{jk}^{(l)}} = \frac{1}{n} \sum_{i=0}^{n-1} \frac{\partial L_i}{\partial w_{jk}^{(l)}}$$

לעדכון ה-bias של השכבה ה- l :

$$\frac{\partial L_0}{\partial b_j^{(l)}} = \sigma'(z_j^{(l)}) \frac{\partial L_0}{\partial a_j^{(l)}}, \quad \text{average} \rightarrow \frac{\partial L}{\partial b_j^{(l)}} = \frac{1}{n} \sum_{i=0}^{n-1} \frac{\partial L_i}{\partial b_j^{(l)}}$$

זאת כאשר:

$$\frac{\partial L_0}{\partial a_j^{(l)}} = \begin{cases} \sum_{j=0}^{n_{l+1}-1} w_{jk}^{l+1} \cdot \sigma'(z_j^{l+1}) \cdot \frac{\partial L_0}{\partial a_j^{(l+1)}}, & l < L \\ 2(a_j^{(L)} - y_j), & l = L \end{cases}$$

הפלט בסופו של דבר הוא הווקטור שכניסותיו הם

$$\frac{\partial L}{\partial w_{jk}^{(l)}} = \frac{1}{n} \sum_{i=1}^{n-1} \frac{\partial L_i}{\partial w_{jk}^{(l)}}$$

$$\frac{\partial L}{\partial b_j^{(l)}} = \frac{1}{n} \sum_{i=1}^{n-1} \frac{\partial L_i}{\partial b_j^{(l)}}$$

4.3 Optimization

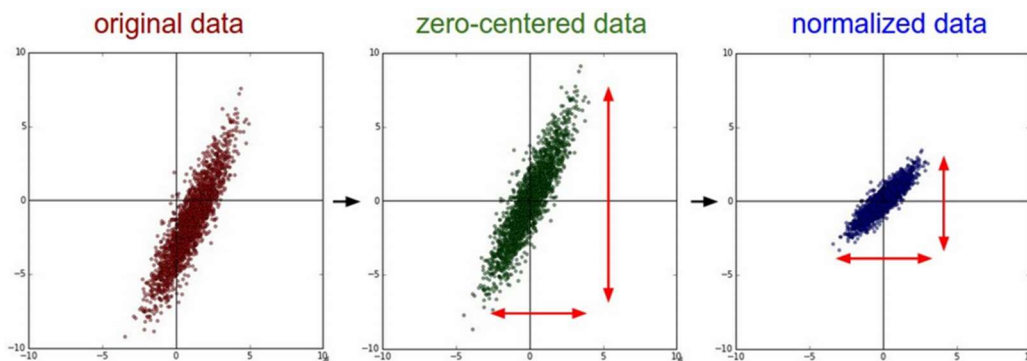
מציאת אופטימום למשקלים על פני כל העומק של הרשת היא בעיה לא קמורה, ולכן אין לה בהכרח מינימום גלובאלי. לכן מלבד עדכון המשקלים בשיטת backpropagation יש לבצע אופטימיזציות נוספות על הרשת על מנת לשפר את הביצועים שלה.

4.3.1 Data Normalization

חלק מפונקציות ההפעלה אינן ממורכזות סביב ה-0, ועבור ערכים גבוהים הן קבועות בקירוב ולכן הגרדיאנט בערכים אלו מתאפס, דבר שאינו מאפשר לעדכן את המשקלים בשיטת GD. כדי להימנע מהגעה לתחום ה"רוויה" בו הגרדיאנט מתאפס, ניתן לנרמל את הדאטה כך שיהיה בעל תוחלת 0 ושונות 1, ובכך הוא יהיה ממורכז סביב ה-0:

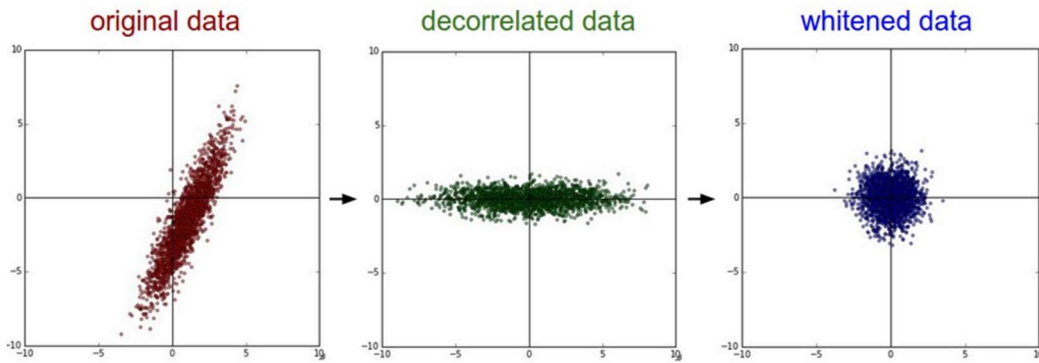
$$X_i = \frac{X_i - \mu_i}{\sigma_i}$$

ובאופן חזותי:



איור 4.11 נרמול דאטה בשני שלבים – איפוס התוחלת (ירוק) ונרמול השונות ל-1 (כחול).

שלב זה הוא למעשה שלב pre-processing הנועד להכין את הדאטה לפני כניסתו לרשת, בכדי לשפר את אימון הרשת. ישנם אופנים נוספים לנרמל את הדאטה – ללכסן את מטריצת ה-Covariance של הדאטה או להפוך אותה למטריצת היחידה:



איור 4.12 דרכים נוספות לנרמל את הדאטה – ללכסן את מטריצת ה-covariance (ירוק) או להפוך אותה למטריצת היחידה (כחול).

4.3.2 Weight Initialization

עניין נוסף שיכול להשפיע על האימון וניתן להתייחס אליו עוד בשלב ה-pre-processing הוא אתחול המשקלים. אם כל המשקלים מאותחלים ב-0, אז המוצא וכל הגרדיאנטים יהיו גם כן 0, ולא יתבצע עדכון למשקלים. לכן יש לבחור את המשקלים ההתחלתיים בצורה מושכלת, כלומר, להגריל אותם מהתפלגות מסוימת שתאפשר אימון טוב של הרשת.

אפשרות אחת לאתחול היא להגריל עבור כל משקל ערך קטן מהתפלגות נורמלית עם שונות קטנה – $N(0, \alpha)$, כאשר $\alpha = 0.01$ or 0.1 . אתחול באופן הזה עובד טוב לרשתות קטנות יחסית, אך ברשתות עם הרבה שכבות אתחול בערכים קטנים גורם לאיפוס הגרדיאנט מהר מדי. כדי להתמודד עם בעיה זו, ניתן לבחור $\alpha = 1$, אך זה יכול לגרום להתבדרות הגרדיאנט. שיטה יעילה יותר נקראת Xavier Initialization, הלוקחת בחשבון את הגודל של השכבות – האתחול יתבצע בעזרת התפלגות נורמלית, אך השונות לא תהיה מספר ללא משמעות, אלא תהיה תלויה במספר השכבות – $\alpha = \frac{1}{\sqrt{n}}$. שיטה זו טובה גם לרשתות עם הרבה שכבות, אך היא בעייתית במקרה בו פונקציית ההפעלה הינה ReLU, כיוון שהאתחול מניח שפונקציית ההפעלה ממורכזת סביב 0 (כמו למשל \tanh). כדי לאפשר גמישות גם

מבחינת פונקציית ההפעלה, ניתן לבחור $\alpha = \sqrt{\frac{2}{n}}$, ואז האתחול יתאים גם ל-ReLU.

אפשרות נוספת לאתחול הפרמטרים היא להגריל מהתפלגות אחידה, כאשר באופן דומה ל-Xavier-Initialization,

$$U\left[-\frac{1}{\sqrt{n}}, \frac{1}{\sqrt{n}}\right] - \text{גם כאן הגבולות יהיו תלויים בגודל השכבות}$$

4.3.3 Batch Normalization

כאשר מבצעים Data normalization, למעשה דואגים לכך שבכניסה לרשת הדאטה יהיה מנורמל סביב ה-0. באופן הזה נמנעים מהגעה למצב בו יש ערכים גבוהים בעומק הרשת, הגורמים להתאפסות או להתבדרות של הגרדיאנט. בפועל, הנרמול הזה לא תמיד מספיק טוב עבור כל השכבות, ואחרי כמה שכבות של הכפלה במשקלים ומעבר בפונקציות הפעלה הרבה פעמים מתקבלים ערכים גבוהים. באופן דומה ל-Data normalization המתבצע לפני האימון, ניתן תוך כדי האימון לבצע Batch normalization שדואג לנרמול הערכים שנכנסים לנוירונים בשכבות החביוות. התהליך נעשה בשלושה שלבים:

- א. עבור כל נוירון בעל פונקציית הפעלה לא ליניארית, מחשבים את התוחלת והשונות של כל הערכים היוצאים ממנו.
- ב. מנרמלים את כל היציאות – מחסירים מכל יציאה את התוחלת ומחלקים את התוצאה בשונות (בתוספת אפסילון, כדי להימנע מחלוקה ב-0).
- ג. הנרמול יכול לגרום לאיבוד מידע, לכן מבצעים לתוצאה המנורמלת scale and shift – הזזה ושינוי קנה המידה. התיקון מתבצע בעזרת פרמטרים נלמדים.

עבור שכבות גדולות חישוב התוחלת והשונות יקר כיוון שלנוירון יש הרבה יציאות, לכן לוקחים רק חלק מהיציאות – Mini Batch: $\mathcal{B} = \{x_1 \dots m\}$.

באופן פורמלי ניתן לנסח את ה-Mini Batch Normalizing transform (Mini) כך:

$$\mu_B = \frac{1}{m} \sum_{i=1}^m x_i, \sigma_B^2 = \frac{1}{m} \sum_{i=1}^m (x_i - \mu_B)^2$$

$$\hat{x}_i = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$$

$$y_i = \gamma \hat{x}_i + \beta \equiv \text{BN}_{\gamma, \beta}(x_i)$$

כאשר γ, β הם פרמטרים נלמדים (עבור כל נירון יש פרמטרים שונים).

בשלב המבחן, השונות והתוחלת שבעזרתם מבצעים את הנרמול אינם נלקחים מהיציאות של הניורונים, אלא לוקחים ממוצע של כמה מה- Mini Batch האחרונים.

יש כמה יתרונות לשימוש ב-Batch normalization: האימון נעשה מהר יותר, יש פחות רגישות לאתחול של המשקלים, מאפשר שימוש ב-learning rate גדול יותר (מונע מהגרדיאנט להתבדר או להתאפס), מאפשר שימוש במגוון פונקציות הפעלה (גם כאלה שאינן ממורכזות סביב 0) ומספק באופן חלקי גם רגולריזציה (שונות נמוכה במוצא).

4.3.4 Mini Batch

במקרים רבים הדאטה-סט גדול, ולחשב את הגרדיאנט עבור כל הדאטה צורך הרבה חישוב. בכל צעד של קידום ניתן לחשב את הגרדיאנט עבור חלק מהדאטה, ולבצע את הקידום לפי הכיוון של הגרדיאנט המתקבל. למשל, ניתן לבחור באופן אקראי נקודה אחת ולחשב עליה את הגרדיאנט. בחירה כזו נקראת Stochastic Gradient Descent (SGD), כיוון שבכל צעד יש בחירה אקראית של נקודה. בחירה אקראית של נקודה בודדת יכולה לגרום לשונות גדולה ככל שהחישוב מתקדם, ולכן בדרך כלל מבצעים mini-batch learning – חישוב הגרדיאנט על חלק מהדאטה. באופן הזה גם יש הפחתה של כמות החישובים, וגם אין שונות גבוהה. אם מבצעים את החישוב בשיטה זו יש לדאוג שהדאטה מעורבב כדי שהמשקלים אכן יתעדכנו בצורה נכונה, ובנוסף שה- mini-batch יהיה מספיק גדול כך שיהיה בו ייצוג לכל הדאטה. כל מעבר על פני כל הדאטה-סט נקרא Epoch (אם הדאטה הוא בגודל N, והגודל של כל- mini-batch הוא S, אז כל Epoch הוא N/S איטרציות).

אמנם כל צעד הוא קירוב לגרדיאנט, אך החישוב מאוד מהיר ביחס לגרדיאנט המדויק, וזה יתרון משמעותי שיש לשיטה זו על פני batch learning. בנוסף, המשקלים שמתקבלים קרובים מאוד לאלו שהיו מתקבלים באמצעות batch learning, כפי שמופיע באיור 3.8.

4.3.5 Gradient Descent Optimization Algorithms

בשיטת GD, עדכון המשקלים בכל צעד הוא: $w_{i+1} = w_i - \epsilon \frac{\partial L}{\partial w}$, כאשר ϵ הוא פרמטר שנקרא Learning Rate (lr), והוא קובע עד כמה יש לשנות המשקל בכיוון הגרדיאנט. בניגוד לבעיות רגרסיה, אופטימיזציה רשת נירונים היא לרוב בעיה שאינה קמורה, לכן לא מובטחת התכנסות למינימום הגלובאלי. משום כך, אם בכל צעד הולכים יותר מדי לכיוון הגרדיאנט השלילי, ניתן להתכנס לנקודת אוסף או למינימום לוקאלי שהוא אינו בהכרח המינימום הגלובאלי. מצד שני אם מתקדמים מעט מדי לכיוון הגרדיאנט, המשקל בקושי מתעדכן. פרמטר ה-lr נועד להתגבר על בעיות אלו, לכן צריך שהוא לא יהיה גדול מדי (אחרת תהיה התבדרות של המשקלים או התכנסות למינימום לוקאלי) ושלא יהיה קטן מדי (אחרת לא תהיה התקדמות או שהיא תהיה מאוד איטית). כיוון שאין ערך אבסולוטי שמתאים לכל הבעיות, יש מגוון שיטות המנסות למצוא את העדכון האופטימלי בכל צעד. יש שיטות שמשתמשות בפרמטר משתנה – adaptive lr, ויש שיטות שמוסיפות פרמטרים אחרים לביטוי של העדכון.

Momentum

ישנם מצבים בהם יש כל מיני פיתולים בדרך לנקודת מינימום. במצב זה, בכל צעד הגרדיאנט יפנה לכיוון אחר, וההתכנסות לנקודת מינימום תהיה איטית. הדבר דומה לנחל שזורם לים, אך הוא לא זורם ישר אלא יש לו הרבה פיתולים. כדי להאיץ את ההתכנסות במקרה זה, ניתן לנסות לבחון את הכיוון הכללי של הגרדיאנט על סמך כמה צעדים, ולהוסיף התקדמות גם לכיוון הזה. שיטה זו נקראת מומנטום, כיוון שהיא מחפשת את המומנטום הכללי של הגרדיאנט. החישוב של המומנטום מתבצע בנוסחה רקורסיבית:

$$m_{i+1} = \mu m_i - \epsilon \frac{\partial L}{\partial w}$$

ואז העדכון הינו:

$$w_{i+1} = w_i + m_{i+1}$$

הפרמטר μ הינו פרמטר דעיכה עם ערך טיפוסו בטווח $[0.9, 0.99]$. ניתן להבין את משמעותו על ידי פיתוח של עוד איבר בנוסחת המומנטום:

$$m_{i+1} = \mu m_i - \epsilon \frac{\partial L(w_i)}{\partial w} = \mu^2 m_{i-1} - \mu \epsilon \frac{\partial L(w_{i-1})}{\partial w} - \epsilon \frac{\partial L(w_i)}{\partial w}$$

ניתן לראות שככל שהולכים אחורה בצעדים, כך החזקה של μ גדלה. אם $\mu < 1$, אז עם הזמן הביטוי μ^n ילך ויקטן, וכך תהיה פחות השפעה לצעדים שכבר היו לפני הרבה עדכונים. תחת הנחה שהגרדיאנט זהה לכל הפרמטרים, ניתן לפתח נוסחה סגורה לרקורסיה:

$$m_{i+1} = \mu m_i - \epsilon \frac{\partial L(w)}{\partial w} = \mu^2 m_{i-1} - \mu \epsilon \frac{\partial L(w)}{\partial w} - \epsilon \frac{\partial L(w)}{\partial w} = \dots = -\epsilon \frac{\partial L}{\partial w} (1 + \mu + \mu^2)$$

הביטוי שמתקבל הוא סדרה הנדסית מתכנסת, ובסך הכל מתקבל הביטוי:

$$w_{i+1} = w_i - \frac{\epsilon}{1 - \mu} \frac{\partial L}{\partial w}$$

היעילות של המומנטום תלויה בבעיה – לפעמים היא מאיצה את ההתכנסות ולפעמים כמעט ואין לה השפעה, אך היא לא יכולה להזיק.

וריאציה של שיטת המומנטום נקראת Nesterov Momentum. בשיטה זו לא מחשבים את הגרדיאנט על הצעד הקודם, אלא על המומנטום הקודם:

$$m_{i+1} = \mu m_i - \epsilon \frac{\partial L}{\partial w}(w_i + \mu m_i)$$

$$w_{i+1} = w_i + m_{i+1} = (w_i + \mu m_i) - \epsilon \frac{\partial L}{\partial w}(w_i + \mu m_i)$$

שיטה זו עובדת טוב יותר עבור בעיות קמורות, כלומר היא מצליחה להתכנס יותר טוב מאשר המומנטום הרגיל, אך היא איטית יותר.

learning rate decay

באימון רשתות עמוקות בדרך כלל כדאי להקטין את ה- lr עם הזמן. הסיבה לכך היא שככל שמתקדמים לכיוון המינימום, יש צורך בצעדים יותר קטנים כדי להצליח להתכנס אליו ולא לזוז מסביבו מצד לצד. עם זאת, קשה לקבוע כיצד בדיוק להקטין את ה- lr : הקטנה מהירה שלו תימנע הגעה לאזור של המינימום, והקטנה איטית שלו לא תעזור להתכנס למינימום כאשר מגיעים לאזור שלו. ישנם שלושה סוגים נפוצים של שינוי הפרמטר:

- שינוי הפרמטר בכל כמה Epochs. מספרים טיפוסיים הם הקטנה בחצי כל 5 epochs או חלוקה ב-10 כל 20 epochs. באופן כללי ניתן לומר שכאשר גרף הלמידה של ה-validation בקושי משתפר, יש להקטין את ה- lr .
- דעיכה אקספוננציאלית של ה- lr : $\epsilon = \epsilon_0 \cdot e^{-kt}$, כאשר ϵ_0, k הם היפר-פרמטרים, ו- t יכול להיות צעד או epoch.
- דעיכה לפי $1/t$: $\epsilon = \frac{\epsilon_0}{1+k}$, כאשר ϵ_0, k הם היפר-פרמטרים, ו- t הינו צעד של עדכון.

Adagrad and RMSprop

בעוד השיטה הקודמת מעדכנת את ה- lr בצורה קבועה מראש, ניתן לשנות אותו גם באופן מסתגל לפי ההתקדמות בכיוון הגרדיאנט. בכל צעד ניתן לבחון עד כמה גדול היה השינוי בצעדים הקודמים, ובהתאם לכך אפשר לקחת lr מתאים, מתוך מגמה להקטין אותו ככל שמתקדמים לכיוון המינימום. באופן פורמלי, אלגוריתם Adagrad מוגדר כך:

$$w_{i+1} = w_i - \epsilon_i \frac{\partial L}{\partial w}, \epsilon_i = \frac{\epsilon}{\sqrt{\alpha_i + \epsilon_0}}, \alpha_i = \sum_{j=1}^i \left(\frac{\partial L}{\partial w_j} \right)^2$$

כאשר ϵ_0 הוא מספר קטן הנועד למנוע חלוקה ב-0. כיוון ש- α_i הולך וגדל, הביטוי $\frac{\epsilon}{\sqrt{\alpha_i + \epsilon_0}}$ הולך וקטן, וקצב הדעיכה הוא ביחס ישר לקצב ההתקדמות בכיוון הגרדיאנט. בכך מרוויחים דעיכה של ה- lr , בקצב המשתנה לפי ההתקדמות. באופן יחסי, הדעיכה של ה- lr מהירה, כיוון שהסכום $\alpha_i = \sum_{j=1}^i \left(\frac{\partial L}{\partial w_j}\right)^2$ גדל במהירות. כדי להאט את קצב הדעיכה, יש שיטות בהן נותנים יותר משקל לצעדים האחרונים ופחות לצעדים שכבר עברו מזמן. השיטה הפופולרית נקראת RMSprop, ובשיטה זו במקום לסכום את ריבוע הגרדיאנט של כל הצעדים הקודמים באופן שווה, מבצעים moving average, וככל שעברו יותר צעדים מצעד מסוים עד לצעד הנוכחי, כך תהיה לו פחות השפעה על דעיכת ה- lr :

$$w_{i+1} = w_i - \epsilon_i \frac{\partial L}{\partial w}, \epsilon_i = \frac{\epsilon}{\sqrt{\alpha_i + \epsilon_0}}, \alpha_i = \beta \alpha_{i-1} + (1 - \beta) \left(\frac{\partial L}{\partial w}\right)^2$$

Adam

ניתן לשלב בין הרעיון של מומנטום לבין adaptive learning rate:

$$\alpha_i = \beta_1 \alpha_{i-1} + (1 - \beta_1) \left(\frac{\partial L}{\partial w}\right)^2, m_i = \beta_2 m_{i-1} + (1 - \beta_2) \frac{\partial L}{\partial w}$$

$$\hat{\alpha}_i = \frac{\alpha_i}{1 - \beta_1^i} \hat{m}_i = \frac{m_i}{1 - \beta_2^i}$$

$$w_{i+1} = w_i - \frac{\epsilon}{\sqrt{\hat{\alpha}_i + \epsilon_0}} \hat{m}_i$$

מספרים טיפוסיים: $\epsilon = 10^{-2}$ or 5^{-4} , $\beta_1 = 0.9$, $\beta_2 = 0.99$. האלגוריתם למעשה גם מוסיף התקדמות בכיוון המומנטום (הכיוון הכללי של הגרדיאנט), וגם מביא לדעיכה אדפטיבית של ה- lr . זה האלגוריתם הכי פופולרי ברשתות עמוקות, אך הוא לא מושלם ויש לו שתי בעיות עיקריות: האימון הראשוני לא יציב, כיוון שבתחילת האימון יש מעט נקודות לחישוב הממוצע עבור m_i . בנוסף, המודל המתקבל נוטה ל- $overfitting$ ביחס ל-SGD עם מומנטום.

יש הרבה וריאציות חדשות על בסיס Adam שנועדו להתגבר על בעיות אלו. ניתן למשל להתחיל לאמן בקצב נמוך, וכאשר המודל מתגבר על בעיית ההתייצבות הראשונית, להגביר את הקצב (Learning rate warm-up). במקביל, ניתן להתחיל עם Adam ולהחליף ל-SGD כאשר קריטריונים מסוימים מתקיימים. כך ניתן לנצל את ההתכנסות המהירה של Adam בתחילת האימון, ואת יכולת ההכללה של SGD.

4.4 Generalization

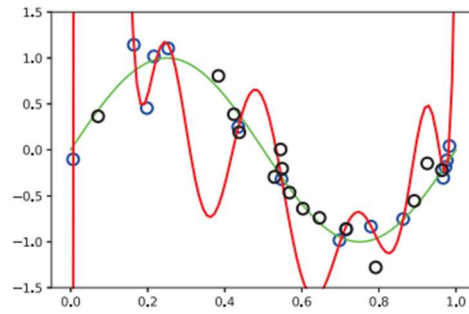
כל מודל שנבנה נסמך על דאטה קיים, מתוך מגמה שהמודל יתאים גם לדאטה חדש. לכן יש חשיבות גדולה שהמודל ידע להכליל כמה שיותר טוב, על מנת שהוא יתאים בצורה טובה לא רק לדאטה הקיים אלא גם לדאטה חדש. במילים אחרות, יש לוודא שהמודל לא מתאים את הפרמטרים שלו רק לדוגמאות שהוא רואה, אלא שינסה להבין מתוך הדוגמאות מה החוקיות הכללית, שמתאימה גם לדוגמאות אחרות.

4.4.1 Regularization

כפי שהוסבר בפרק 3.1.3, מודל יכול לסבול מהטיה לשני כיוונים – $Overfitting$ ו- $Underfitting$. $Overfitting$ הוא מצב בו ניתנת הערכת יתר לכל נקודה בסט האימון, מה שגורר מודל מסדר גבוה בעל שונות גדולה. במצב זה המודל מתאים רק לסט האימון, אך הוא לא מצליח להכליל גם נקודות חדשות. $Underfitting$ הוא המצב ההפוך – מודל שלא מצליח למצוא קו מגמה המכיל מספיק מידע על הדוגמאות הנתונות, ויש לו רעש חזק.

ברשת נורונים, ככל שמספר הפרמטרים גדל, כך השגיאה של ה- $training$ קטנה. לגבי ה- $test$ השגיאה הולכת ויורדת עד נקודה מסוימת, ומשם היא גדלה בחזרה. בהתחלה השגיאה יורדת כיוון שמצליחים לבנות מודל יותר מדויק ונמנעים מ- $underfitting$, אך בנקודה מסוימת יש יותר מדי פרמטרים והם נהיים מותאמים יותר מדי לסט האימון ומתקבל $overfitting$. למעשה צריך למצוא את היחס הנכון בין מספר הפרמטרים (סדר המודל) לבין גודל הדאטה. כיוון שאי אפשר לזהות $Overfitting$ בעזרת ה- $training$ בלבד, שהרי ה- $Loss$ קטן ככל שיש יותר פרמטרים, ניתן לחלק את הדאטה לשני חלקים – $training$ and $validation$. בשלב ראשון בונים מודל בהינתן ה- $training$ ולאחר מכן בוחנים את המודל על ה- $validation$ – אם המודל לא מתאים ל- $validation$ סימן שיש $overfitting$, כלומר המודל

מתאים רק לדוגמאות שהוא ראה והוא נתן להם הערכת יותר. ככל שהגרף של ה-validation accuracy קרוב יותר ל-training accuracy, כך יש פחות overfitting.



איור 4.13 בדיקת overfitting בעזרת validation set. הנקודות הכחולות שייכות ל-training והשחורות ל-validation. המודל האדום מתאים רק לנקודות הכחולות, לכן אפשר לומר שהוא נוטה ל-overfitting. המודל הירוק לעומת זאת מתאים גם לנקודות השחורות, אותן הוא לא ראה בשלב האימון, כלומר הוא הצליח להכליל טוב גם לדוגמאות חדשות.

האפשרות הכי פשוטה להימנע מ-overfitting היא פשוט להוריד פרמטרים, כלומר להקטין את גודל הרשת. בנוסף ניתן לבצע Early stopping – לחשב בכל Epoch את גרף ה-Loss של ה-validation, וכאשר הוא מתחיל לעלות להפסיק את האימון. שיטות אלה פשוטות מאוד ליישום, אך ישנן שיטות אחרות שמספקות ביצועים יותר טובים, ובכך אותם כעת.

4.4.2 Weight Decay

דומה לרגולריזציה של linear regression, גם ברשת נירונים ניתן להוסיף איבר ריבועי לפונקציית המחיר, מה שמכונה L2 Regularization:

$$Cost(w; x, y) = L(w; x, y) + \frac{\lambda}{2} \|w\|^2$$

ההוספה של הביטוי האחרון דואגת לכך שהמשקל לא יהיה גדול מדי, שהרי רוצים למזער את פונקציית המחיר, לכן נשאף לכך שהביטוי הריבועי יהיה כמה שיותר קטן. בתוספת האיבר עדכון של המשקלים יהיה:

$$w_{i+1} = w_i - \epsilon \left(\frac{\partial L}{\partial w} + \lambda w \right) = (1 - \epsilon \lambda) w - \epsilon \frac{\partial L}{\partial w}$$

הביטוי הזה דומה מאוד ל-GD רגיל, כאשר נוסף איבר $\epsilon \lambda w$. אם $0 < \epsilon \lambda < 1$, אז ללא קשר לגרדיאנט המשקל יורד בכל צעד, וזה נקרא "Weight decay".

ניתן לבצע רגולריזציה עם איבר לא ריבועי, מה שמכונה L1 Regularization:

$$Cost(w; x, y) = L(w; x, y) + \lambda \sum_i |w_i|$$

ואז העדכון יהיה:

$$w_{i+1} = w_i - \epsilon \left(\frac{\partial L}{\partial w} + \lambda \cdot \text{sign}(w) \right)$$

בעוד L2 Regularization התייחס למשקל יחיד וניסה להקטין אותו, L1 Regularization "מעניש" אם סכום המשקלים בערך מוחלט גדול, מה שיגרום לחלק מהמשקלים להתאפס ולדילול מספר הפרמטרים של הרשת.

4.4.3 Model Ensembles and Drop Out

עבור דאטה קיים ניתן לבנות מספר מודלים, ואז כשבאים לבחון דאטה חדש בודקים אותו על כל המודלים ולוקחים את הממוצע. סט המודלים נקרא ensemble. ניתן לבנות מודלים שונים במספר דרכים:

א. לאמן רשת עם אתחולים שונים למשקלים.

ב. לאמת מספר רשתות על חלקים שונים של הדאטה.

ג. לאמן רשת במספר ארכיטקטורות.

יצירת ensemble בדרכים אלה יכולה לעזור בהכללה, אך יקר ליצור את ה-ensemble ולפעמים קשה לשלב בין מודלים שונים.

יש דרך נוספת ליצור ensemble – לבצע Dropout, כלומר למחוק באופן אקראי נירון אחד או יותר. אם יש רשת מסוימת ומוחקים את אחד הנירונים – למעשה מקבלים רשת אחרת, ובפועל אפשר לקבל ensemble בעזרת רשת אחת שכל פעם מוחקים ממנה נירון אחד או יותר. היתרון של יצירת ensemble בדרך הזו הוא שהרשתות חולקות את אותן פרמטרים ולבסוף מקבלים רשת אחת מלאה עם כל הנירונים והמשקלים. בפועל עבור כל דגימה מגרילים רשת (מוחקים כל נירון בהסתברות $p = 0.5$) וכך לומדים במקביל הרבה רשתות שונות עם אותן פרמטרים. באופן הזה כל נירון מוכרח להיות יותר משמעותי בלי אפשרות להסתמך על נירונים אחרים שיעשו את הלמידה, כיוון שלא תמיד הם קיימים. אמנם כל ריצה יחידה יכולה להיות בעלת שונות גבוהה אך הממוצע של המשקלים מביא לשונות נמוכה.

בשלב המבחן, לא מפעילים את ה-Dropout אלא לוקחים את כל הנירונים, כאשר מחלקים את כל המשקלים בחצי. הסיבה לכך היא שניתן להניח שבשלב האימון חצי מהפעמים המשקל היה 0 כיוון שהנירון המקושר אליו נמחק, ובחצי מהפעמים היה משקל שנלמד. ניתן גם לקחת הסתברות אחרת למחיקת נירונים, למשל $p = 0.25$, ואז כשמסכמים את כל הרשתות השונות יש לחלק בהסתברות המתאימה. החיסרון של שיטה זו הוא שלוקח לה זמן להתכנס.

4.4.4 Data Augmentation

שיטה אחרת להימנע מ-overfitting היא להגדיל את סט האימון, וכך המודל שנוצר יתאים ליותר דוגמאות. ניתן לעשות זאת על ידי יצירת וריאציות של הדוגמאות הקיימות. שיטה זו נקראת Data Augmentation, והרעיון הוא לבצע עיוות קטן לכל דוגמא כך שהיא עדיין תשמור על המשמעות המקורית שלה, אך תהיה מספיק שונה מהמקור בכדי להיות דוגמא נוספת משמעותית בסט האימון. בדומיין של תמונות האוגמנטציות הנפוצות הן:

- סיבוב תמונה בזווית מסוימת (rotate), הנבחרת מהתפלגות אחידה מהתחום $[0, 2\pi]$.
- הוספת רעש לכל פיקסל, כאשר הרעש משתנה מפיקסל לפיקסל, והוא קטן מ- $|\epsilon|$.
- שינוי הגודל (resizing) של התמונה בפקטור מסוים – בדרך כלל הפקטור שייך לתחום $[\frac{1}{1.6}, 1.6]$.
- שיקוף התמונה (flip).
- מתיחה ומריחה של התמונה (shearing and stretching).

References

MLP:

מצגות מהקורס של פרופ' יעקב גולדברגר

<https://joshuagoings.com/2020/05/05/neural-network/>

Xor:

<https://www.semanticscholar.org/paper/Simulations-of-threshold-logic-unit-problems-using-Chowdhury-Ayman/ecd5cb65f0ef50e855098fa6e244c2b6ce02fd48>

5. Convolutional Neural Networks (CNNs)

הרשתות שתוארו עד כה הינן Fully-Connected (FC), כלומר, כל נירון מחובר לכל הנירונים בשכבה שלפניו ולכל הנירונים בשכבה שאחריו. גישה זו יקרה מבחינה חישובית, ופעמים רבות אין צורך בכל הקשרים בין הנירונים. לדוגמה, תמונה בגווי אפור (grayscale) בעלת $256 \times 256 \times 1$ פיקסלים, המוזנת לרשת FC עם $N = 1000$ קטגוריות במוצא, מכילה יותר מ-65 מיליון קשרים בין נירונים, כאשר כל קשר הינו משקל המתעדכן במהלך הלמידה. אם יש מספר שכבות רב המספר נהיה עצום ממש, ולכן כמות הקשרים והפרמטרים גדלה, באופן כזה שבלתי מעשי לתחזק את הרשת. מלבד בעיית הגודל, בפועל לא תמיד יש צורך בכל הקשרים, כיוון שלא תמיד יש קשר בין כל איברי הכניסה. למשל, עבור תמונה המוזנת לרשת, במשימות רבות קשר בין פיקסלים רחוקים בתמונה אינו משמעותי, לכן אין חשיבות לחבר את הכניסה לכל הנירונים בשכבה הראשונה ולקשר בין כל שתי שכבות סמוכות באופן מלא. כדי להימנע מבעיות אלו, לרוב יהיה כדאי להשתמש ברשתות או שכבות קונבולוציה, שאינן מקשרות בין כל שני נירונים, אלא רק בין איברים קרובים, כפי שיפורט. רשתות מודרניות רבות מבוססות על שכבות קונבולוציה, כאשר על גבי המבנה הבסיסי נבנו ארכיטקטורות מתקדמות.

5.1 Convolutional Layers

5.1.1 From Fully-Connected Layers to Convolutions

האלמנט הבסיסי ביותר ברשתות קונבולוציה הינו שכבת קונבולוציה, המבצעת קונבולוציה לינארית על פני דאטה בכדי לקבל ייצוג אחר ופשוט יותר שלו. לרוב, שכבת קונבולוציה מבצעת פעולת קרוס-קורלציה בין וקטור המשקלים לבין input מסוים (וקטור הכניסה או וקטור היוצא משכבה חבויה). וקטור המשקלים נקרא גרעין הקונבולוציה (convolution kernel) או מסנן (filter), ובעזרתו מבוצעת פעולת הקרוס-קורלציה הבאה:

$$y[n] = \sum_{m=1}^{K-1} x[n-m]w[m]$$

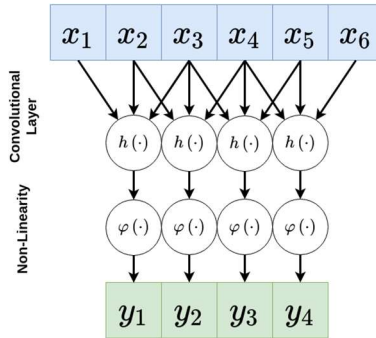
כאשר $x \in \mathbb{R}^n$ הוא וקטור הכניסה ואילו $w \in \mathbb{R}^K$ הוא וקטור המשקלים אשר נלמדים במהלך האימון. וקטור המשקלים w זהה לכל הכניסות בשכבה ולכן מספר הפרמטרים הנלמדים לעומת שכבת FC הינו קטן בהרבה – שכבת FC מכילה $N_{inputs} \times N_{outputs}$ משקלים ואילו שכבת קונבולוציה מכילה K משקלים בלבד (לרוב מתקיים $K \ll N_{inputs} \times N_{outputs}$).

מלבד הקטנת כמות המשקלים, השימוש בגרעין קונבולוציה מסייע לזיהוי דפוסים ולמציאת מאפיינים. יכולות אלו נובעות מאופי פעולת הקונבולוציה, הבודקת חפיפה בין חלקים מווקטור הכניסה לבין גרעין הקונבולוציה. הקונבולוציה יכולה למצוא מאפיינים בסיגנל, וישנם גרעיני קונבולוציה שיכולים לבצע אוסף פעולות שימושיות, כמו למשל החלקה, נגזרת ועוד. אם מטילים על תמונה הרבה גרעינים שונים, ניתן למצוא בה כל מיני מאפיינים – למשל אם הגרעין הוא בצורה של עין או אף, אז הוא מסוגל למצוא את האזורים בתמונה המקורית הדומים לעין או אף.



איור 5.1 (a) קונבולוציה חד ממדית בין שתי פונקציות: x_1 הינו מלבן בגובה 1 עם רעש קטן (כחול), ו- x_2 הינו גרעין קונבולוציה מלבני שרץ על פני כל הישר (כתום). פעולת הקונבולוציה (שחור) בודקת את החפיפה בין הסיגנל לבין הגרעין, וניתן לראות שאכן סביב $x = 0.5 \pm 0.1$ יש אזור עם הרבה חפיפה. (b) קונבולוציה דו ממדית למציאת קווי מתאר של בתוך תמונה.

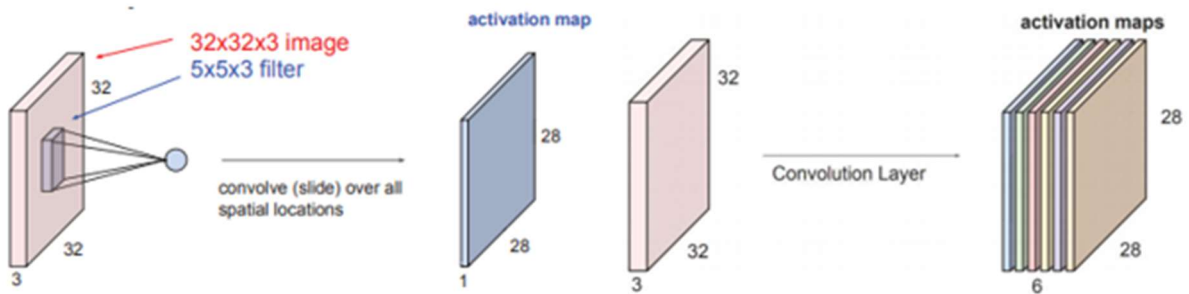
המוצא של שכבת הקונבולוציה עובר בפונקציית הפעלה לא לינארית (בדרך כלל tanh או ReLU), והוא מכונה מפת הפעלה (activation map) או מפת מאפיינים (feature map). הקונבולוציה יחד עם האקטיבציה נראות כך:



איור 5.2 דאטה x עובר דרך שכבת קונבולוציה ולאחריה פונקציית הפעלה, ובמוצא מתקבלת מפת אקטיביציה y .

לרוב בכל שכבת קונבולוציה יהיו כמה מסננים, אשר כל אחד מהם אמור ללמוד מאפיין אחר בתמונה. ככל שהרשת הולכת ומעמיקה, כך המאפיינים בתמונה אמורים להיות מובחנים באופן חד יותר אחד מהשני, ולכן המסננים בשכבות העמוקות אמורים להבדיל בין דברים מורכבים יותר. למשל, פעמים רבות ניתן להבחין כי המסננים הראשונות יהיו את שפות האלמנטים שבתמונה או בצורות אבסטרקטיות, ואילו מסננים בשכבות העמוקות יותר יהיו אלמנטים מורכבים יותר כמו איברים או חפצים שלמים בעלי צורה ידועה ומוגדרת.

הקלט של שכבת הקונבולוציה יכול להיות רב ערוצי (למשל, תמונה צבעונית המיוצגת לרוב בעזרת ערכי RGB). במקרה זה הקונבולוציה יכולה לבצע פעולה על כל הערוצים יחד ולספק פלט חד ערוצי והיא יכולה גם לבצע פעולה על כל ערוץ בנפרד ובכך לספק פלט רב ערוצי. גרעין הקונבולוציה יכול להיות חד ממדי, כלומר וקטור שפועל על קלט מסוים, אך הוא יכול להיות גם מממד גבוה יותר. לרוב, מסננים הפועלים על תמונות הינם דו ממדיים, ופעולת הקונבולוציה מבצעת בכל שלב כפל בין המסנן לבין אזור דו ממדי אחר בתמונה.



איור 5.3 מסנן $F \in \mathbb{R}^{5 \times 5 \times 3}$ פועל על קלט $x \in \mathbb{R}^{32 \times 32 \times 3}$ ומתקבלת מפת אקטיביציה $y \in \mathbb{R}^{28 \times 28}$ (שמאל). הקלט יכול לעבור דרך מספר מסננים ולייצר מפת אקטיביציה עם מספר שכבות – עבור שישה מסננים הממד של המפה יהיו $y \in \mathbb{R}^{28 \times 28 \times 6}$ (ימין).

5.1.2 Padding, Stride and Dilation

כמו ברשת FC, גם ברשת קונבולוציה יש היפר-פרמטרים הנקבעים מראש וקובעים את אופן פעולת הרשת. ישנם שני פרמטרים של שכבת הקונבולוציה – גודל המסנן ומספר ערוצי הקלט וכן שלושה פרמטרים עיקריים נוספים הקובעים את אופן פעולת הקונבולוציה:

ריפוד (Padding): פעולת הקונבולוציה המוגדרת בעזרת המסנן הינה פעולה מרחבית, כלומר, המסנן פועל על מספר איברים בכל פעולה. בנוסף, נשים לב כי פעולת הקונבולוציה לא מוגדרת על איברי הקצוות לכן לא נוכל להפעיל את המסנן במקומות אלו. באיור 5.2 ניתן לראות כיצד פעולה על תמונה בממד של 32×32 מקטינה את ממד הפלט ל- 28×28 , דבר הנובע מכך שהקונבולוציה לא מוגדרת על הפיקסלים בקצוות התמונה ולכן לא מופעלת עליהם. אם רוצים לבצע את הקונבולוציה גם על הקצוות, ניתן לרפד את שולי הקלט (באפסים או שכפול של ערכי הקצה). עבור

$$\text{Padding} = \frac{K-1}{2}$$

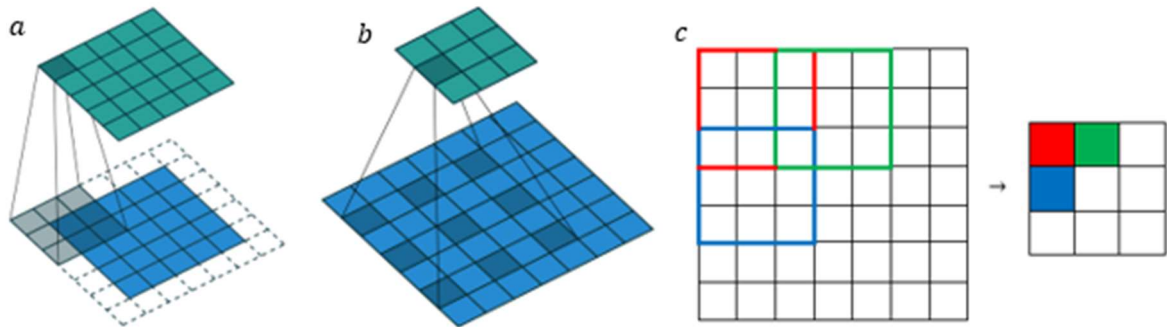
מסנן בגודל $K \times K$, גודל הריפוד הנדרש יהיה:

התרחבות (Dilation): על מנת לצמצם עוד במספר החישובים, אפשר לפעול על אזורים יותר גדולים מתוך הנחה שערכים קרובים גיאוגרפית הם בעלי ערך זהה. לשם כך ניתן להרחיב את פעולת הקונבולוציה תוך השמטה של ערכים קרובים. התרחבות טיפוסית הינה בעלת פרמטר $d = 2$.

גודל צעד (Stride): ניתן להניח שלרוב הקשר המרחבי נשמר באזורים קרובים, לכן על מנת להקטין בחישוביות ניתן לדלג על הפלט ולהפעיל את פעולת הקונבולוציה באופן יותר דליל. כלומר, אין צורך להטיל את המסנן על כל האזורים

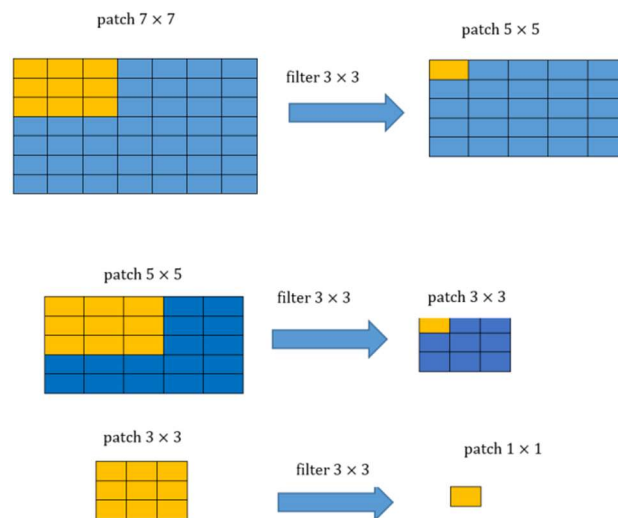
האפשריים ברשת, אלא ניתן לבצע דילוגים, כך שלאחר כל חישוב קונבולוציה יבוצע דילוג בגודל הצעד לפני הקונבולוציה הבאה. גודל צעד טיפוסי הינו $s = 2$.

גודל שכבת הפלט לאחר ביצוע הקונבולוציה תלוי בגדלים של הכניסה והמסנן, בריפוד באפסים ובגודל הצעד. באופן פורמלי ניתן לחשב את גודל שכבת הפלט לפי הנוסחה: $O = \frac{W-K+2P}{s} + 1$, כאשר W הוא גודל הכניסה, K הוא גודל המסנן, P זה הריפוד באפסים ו- s זה גודל הצעד. מספר שכבות הפלט הינו כמספר המסננים (כאשר שכבת פלט יכולה להיות רב ערוצית). יש לשים לב שערכי ההיפר-פרמטרים (padding, dilation and stride) וכן גודל הגרעין נדרשים להיות מספרים טבעיים אשר מקיימים את נוסחת גודל שכבת הפלט (0) הנ"ל, כך שגם O הינו מספר טבעי.



איור 5.4 (a) ריפוד באפסים על מנת ביצוע קונבולוציה גם על הקצוות של הדאטה. (b) התרחבות ($d = 2$): ביצוע הקונבולוציה תוך השמטת איברים סמוכים מתוך הנחה שכנראה הם דומים. (c) הזזת המסנן בצעד של $s = 2$.

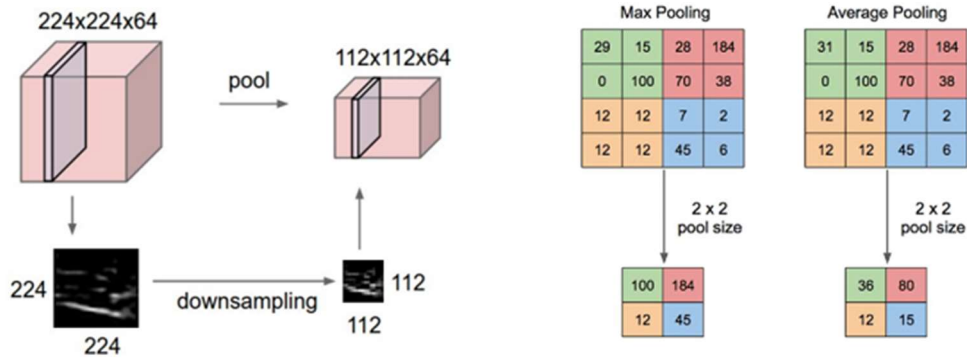
תמך (Receptive field) של איבר ברשת מוגדר להיות כל התחום בכניסה אשר משפיע על אותו איבר לאורך השכבות.



איור 5.5 Receptive field של ערך מסוים במוצא של שלוש שכבות קונבולוציה רצופות עם מסנן בגודל 3×3 .

5.1.3 Pooling

פעמים רבות דאטה מרחבי מאופיין בכך שאיברים קרובים דומים אחד לשני, למשל – פיקסלים סמוכים לרוב יהיו בעלי אותו ערך. ניתן לנצל עובדה זו בכדי להוריד את מספר החישובים הדרוש בעזרת דילוגים (Strides) או הרחבה (dilation) כפי שתואר לעיל. שיטה אחרת לניצול עובדה זו היא לבצע Pooling – אחרי כל ביצוע קונבולוציה, דגימת ערך יחיד מאזור בעל ערכים מרובים, המייצג את האזור. את צורת חישוב הערך של תוצאת ה-pooling ניתן לבחור בכמה דרכים, כאשר המקובלות הן בחירת האיבר הגדול ביותר באזור שלו (max pooling) או את הממוצע של האיברים (average pooling).



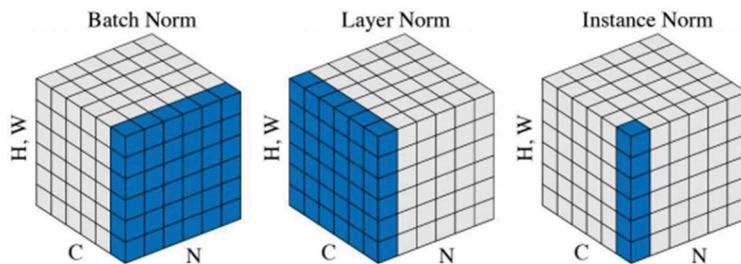
איור 5.6 הקטנת הממד של הדאטה בעזרת Pooling (שמאל), והמחשה מספרית של ביצוע max/average pooling בגודל של 2×2 .

5.1.4 Training

ככלל, תהליך האימון של רשת קונבולוציה זהה לאימון של רשת FC, כאשר ההבדל היחיד הוא בארכיטקטורה של הרשת. יש לשים לב שהמסננים מופעלים על הרבה אזורים שונים, כאשר המשקלים של המסננים בכל צעד שווים, ולכן אותם משקלים פועלים על אזורים שונים. לשם הפשטות נניח ויש מסנן יחיד, כלומר מטריצה אחת נלמדת של משקלים. מטריצה זו מוכפלת בכל אחד מהאזורים השונים של הדאטה, וכדי לבצע עדכון למשקלים שלה יש לשקלל את הגרדיאנטים של כל האזורים השונים. בפועל, הגרדיאנט בכל צעד יהיה הסכום של הגרדיאנטים על פני כל הדאטה, ועבור המקרה הכללי בו יש N אזורים שונים עליהם מופעל המסנן הגרדיאנט יהיה:

$$\frac{\partial L}{\partial w_k} = \sum_{i=1}^N \frac{\partial L}{\partial w_k(i)}$$

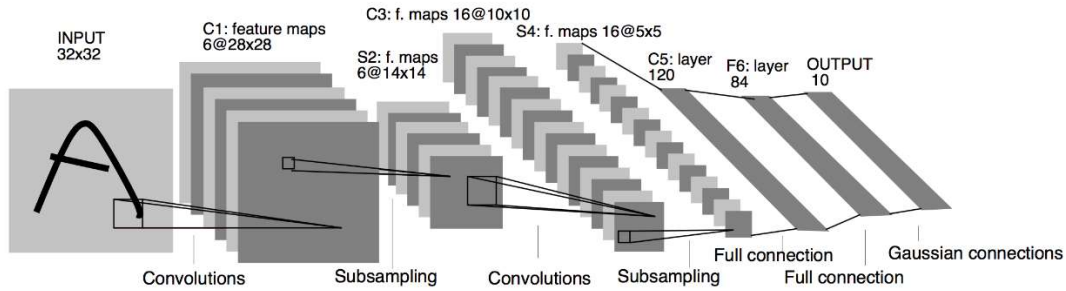
בדומה ל-FC, גם ב-CNN ניתן לבצע Mini-Batch Normalization, כאשר יש כמה אפשרויות לבצע את הנרמול על סט של וקטורים מסוימים (לשם הנוחות נתייחס לווקטורים של הדאטה כתמונות). אפשרות פשוטה ונפוצה היא לנרמל כל מסנן בפני עצמו על פני כמה תמונות (Batch Norm), כלומר לקחת את כל הפיקסלים בסט של תמונות ולנרמל בתוחלת ובשונות שלהם. אפשרות נוספת היא לקחת חלק מהמידע של סט תמונות, אך לנרמל אותו ביחס לאותו מידע על פני מסננים אחרים (Layer Norm). יש וריאציות של הנרמולים האלה, כמו למשל Instance Norm, הלוקח מסנן אחד ותמונה אחת ומנרמל את הפיקסלים של אותה תמונה.



איור 5.7 נרמול שכבות של רשת קונבולוציה.

5.1.5 Convolutional Neural Networks (LeNet)

בעזרת שרשרת של שכבות וחיבור כל האלמנטים השייכים לקונבולוציה ניתן לבנות רשת שלמה עבור מגוון משימות שונות. לרוב במוצא שכבות הקונבולוציה יש שכבה אחת או מספר שכבות FC. מטרת ה-FC היא לאפשר חיבור של המידע המוכל במאפיינים שנאספו במהלך שכבות הקונבולוציה. ניתן להסתכל על הרשת הכוללת כשני שלבים – בשלב הראשון מבצעים קונבולוציה עם מסננים שונים, שכל אחד מהם נועד לזהות מאפיין אחר, ובשלב השני מחברים חזרה את כל המידע שנאסף על ידי חיבור כל הנוירונים באמצעות FC. לראשונה השתמשו בארכיטקטורה זו בשנת 1998, ברשת הנקראת LeNet (על שם Yann LeCun), ומוצגת באיור 5.8. רשת זו השיגה דיוק של 98.9% בזיהוי ספרות, כאשר המבנה שלה הוא שתי שכבות של קונבולוציה ושלוש שכבות FC, כאשר לאחר כל אחת משכבות הקונבולוציה מבצעים pooling.



איור 5.8 ארכיטקטורת LeNet.

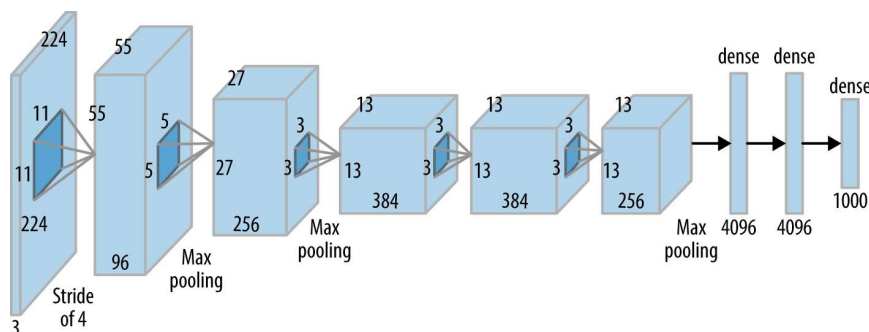
5.2 CNN Architectures

בשנים שלאחר LeNet העיסוק ברשתות נירונים עמוקות די נזנח, עקב חוסר המשאבים לבצע חישובים רבים ביעילות ובמהירות. בשנת 2012 רשת בשם AlexNet המבוססת על שכבות קונבולוציה ניצחה בתחרות ImageNet (תחרות לזיהוי תמונות), כאשר היא הציגה שיפור משמעותי מהתוצאה הכי טובה בשנה שלפני. יחד עם התפתחות יכולות החישוב, העיסוק ברשתות עמוקות חזר להיות מרכזי ופותחו הרבה מאוד ארכיטקטורות מתקדמות.

5.2.1 AlexNet

רשת AlexNet שאבה את ההשראה למבנה שלה מארכיטקטורת LeNet, כאשר היכולת שלה להתמודד עם משימות יותר מורכבות מאשר LeNet נובעת מכך שהיה דאטה-סטים גדולים מאוד שניתן לאמן עליהם את הרשת, ובנוסף כבר היה קיים GPU שבעזרתו ניתן לבצע חישובים מורכבים. הארכיטקטורה של הרשת מורכבת מחמש שכבות קונבולוציה ושלוש שכבות FC, כאשר לאחר שתי השכבות הראשונות של הקונבולוציה מתבצע pooling ו-normalization. ה-input הוא מממד $224 \times 224 \times 3$, ומופעלים עליו 96 מסננים בגודל 11×11 , עם גודל צעד $s = 4$ וללא ריפוד באפסים. לכן המוצא של הקונבולוציה הינו מממד $55 \times 55 \times 96$. לאחר מכן מתבצע max-pooling שמפחית את שני הממדים הראשונים, ומתקבלת שכבה בממד $27 \times 27 \times 96$. בשכבת הקונבולוציה השנייה יש 256 מסננים בגודל 5×5 עם גודל צעד $s = 1$ וריפוד באפסים $p = 2$, לכן במוצא הממד הוא $27 \times 27 \times 256$, ואחרי max-pooling מתקבלת שכבה בממד $13 \times 13 \times 256$. לאחר מכן יש עוד 2 שכבות של קונבולוציה עם מסננים בממד $3 \times 3 \times 384$, גודל צעד $s = 1$ וריפוד $p = 1$, ואז שכבת קונבולוציה אחרונה עם 256 מסננים בממד 3×3 , עם $s = p = 1$. במוצא הקונבולוציות יש עוד max-pooling, ואז שלוש שכבות FC, כאשר המוצא של השכבה האחרונה הוא וקטור באורך 1000, המייצג 1000 קטגוריות שונות שיש בדאטה-סט ImageNet.

פונקציית האקטיבציה של הרשת הינה ReLU (בשונה מ-LeNet שהשתמשה ב-tanh), וההיפר פרמטרים הם: $\text{Dropout}=0.5$, $\text{batch size}=128$, $\text{SGD+momentum}=0.9$, $\text{lr}=1e-2$. בסך הכל מספר הפרמטרים ברשת הינו בערך 60 מיליון.

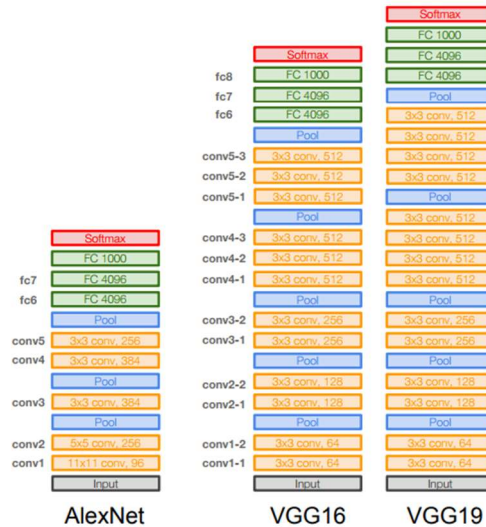


איור 5.9 ארכיטקטורת AlexNet.

שנה לאחר הפרסום של רשת AlexNet, פותחה רשת דומה בשם ZFNet, הבנויה באותה ארכיטקטורה עם הבדלים קטנים בהיפר-פרמטרים ובמספר הפילטרים: השכבה הראשונה של הקונבולוציה הפכה מ: $11 \times 11, s = 4$ ל: $7 \times 7, s = 2$ ובשכבות 3-4-5 מספר הפילטרים הוא 512, 1024, 512 בהתאמה. הרשת השיגה שיפור של כ-5% על פני AlexNet. הממד של השכבות בשתי הארכיטקטורות אינו נובע מסיבה מסוימת אלא מניסוי וטעיה –נוסו תצורות רבות ומתוכן נבחרה זו בעלת הביצועים הטובים ביותר. לאחר שהרשתות מבוססות קונבולוציה הוכיחו את כוחן, השלב הבא היה לבנות רשתות עמוקות, ובעלות ארכיטקטורה הנשענת לא רק על ניסויים אלא גם על היגיון מסוים.

5.2.2 VGG

שנה לאחר ZFNet הוצגה בתחרות רשת עמוקה – בעלת 19 שכבות, המנצלת יותר טוב את שכבות הקונבולוציה. מפתחי הרשת הראו כי ניתן להחליף שכבת פילטרים של 7×7 בשלוש שכבות של 3×3 ולקבל את אותו תמך (receptive field), כאשר מרוויחים חסכון משמעותי במספר הפרמטרים הנלמדים. לפילטר בגודל $d \times d$ הפועל על c ערוצי קלט ופלט יש $d^2 c^2$ פרמטרים נלמדים, לכן לפילטר של 7×7 יש $49c^2$ פרמטרים נלמדים ואילו לשלוש שכבות של 3×3 יש $3 \cdot 3^2 \cdot c^2 = 27c^2$ פרמטרים נלמדים – חיסכון של 45%. הרשת המקורית שפיתחו נקראת VGG16 והיא מכילה 138 מיליון פרמטרים, ויש לה וריאציה המוסיפה עוד שתי שכבות קונבולוציה ומכונה VGG19.

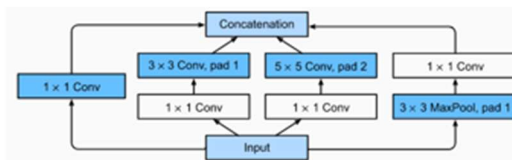
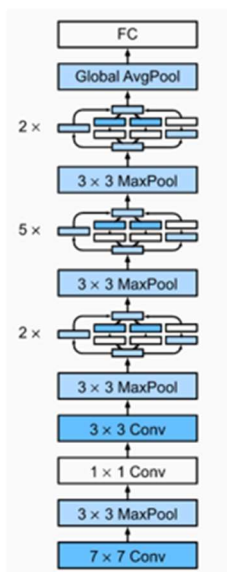


איור 5.10 ארכיטקטורת VGG (ימין) ביחס לארכיטקטורת AlexNet (שמאל).

5.2.3 GoogleNet

המודלים הקודמים היו יקרים חישובית עקב מספר הפרמטרים הגדול. בכדי להצליח להגיע לאותם ביצועים עם אותו עומק אבל עם הרבה פחות פרמטרים, קבוצת מפתחים מגוגל הציגו קונספט שנקרא inception module. בלוק המבצע הרבה פעולות פשוטות במקביל, במקום לבצע פעולה אחת מורכבת. כל בלוק מקבל input ומבצע עליו ארבעה חישובים במקביל, כאשר הממדים של מוצאי כל הענפים שווים כך שניתן לשרשר אותם יחד. ארבעת הענפים הם: קונבולוציה 1×1 , קונבולוציה 1×1 ולאחריה קונבולוציה 3×3 עם padding בגודל 1, קונבולוציה 1×1 ולאחריה קונבולוציה 5×5 עם padding בגודל 2, ו- 3×3 max pooling עם padding 1 ולאחריו קונבולוציה 1×1 . לבסוף, הפלטים של ארבעת הענפים משורשרים יחד ומהווים את פלט הבלוק.

המבנה הזה שקול למספר רשתות במקביל, כאשר היתרון של המבנה הזה הוא כפול: כמות פרמטרים נמוכה ביחס לרשתות קודמות וחישובים יחסית מהירים כיוון שהם נעשים במקביל. ניתן לחבר שכבות קונבולוציה רגילות עם בלוקים כאלה, ולקבל רשת עמוקה. נעשו הרבה ניסויים כדי למצוא את היחס הנכון בין הרכיבים והממדים בכל שכבה המביאים לביצועים אופטימליים.



איור 5.11 Inception Block יחיד (ימין), וארכיטקטורת GoogleNet מלאה (שמאל).

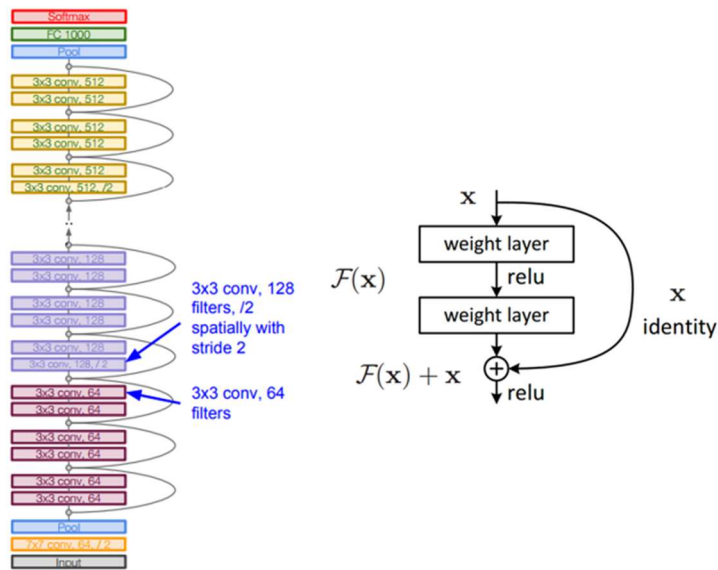
5.2.4 Residual Networks (ResNet)

לאחר שראו שככל שהרשת עמוקה יותר כך היא משיגה תוצאות טובות יותר, ניסו לבנות רשתות עם מאות שכבות, אך הן השיגו תוצאות פחות טובות מהרשתות הקודמות שהיו בעלות סדר גודל של 20 שכבות. הבעיה המרכזית של הרשתות העמוקות נבעה מכך שלאחר מספר שכבות מסוים התקבל ייצוג מספיק טוב, ולכן השכבות היו צריכות לא לשנות את הקלט אלא להעביר את הייצוג כמו שהוא. בשביל לבצע זאת המשקלים בשכבות אלו צריכים להיות 1. הסתבר שלשכבות קשה ללמוד את פונקציית הזהות והן למעשה פגעו בתוצאה. אתגר נוסף ברשתות עמוקות נבע מהקושי לבצע אופטימיזציה כמו שצריך למשקלים בשכבות עמוקות.

ניתן לנסח את הבעיה המרכזית באופן מעט שונה – בהינתן רשת עם N שכבות, יש טעם להוסיף שכבה נוספת רק אם היא תוסיף מידע שלא קיים עד עכשיו. כדי להבטיח ששכבה תוסיף מידע, או לכל הפחות לא תפגע במידע הקיים, בנו רשת חדשה בעזרת Residual Blocks – יצירת בלוקים של שכבות קונבולוציה, כאשר בנוסף למעבר של המידע בתוך הבלוק, מחברים גם בין הכניסה למוצא שלו. כעת אם בלוק מבצע פונקציה מסוימת $\mathcal{F}(x)$, אזי המוצא הינו $\mathcal{F}(x) + x$. באופן הזה כל בלוק ממוקד בללמוד משהו שונה ממה שנלמד עד עכשיו, ואם אין מה להוסיף – הפונקציה $\mathcal{F}(x)$ פשוט נשארת 0. בנוסף, המבנה של הבלוקים מונע מהגרדיאנט בשכבות העמוקות להתבדר או להתאפס, והאימון מצליח להתכנס.

באופן הזה פותחה רשת בעלת 152 שכבות אשר הציגה ביצועים מעולים ביחס לכל שאר הרשתות באותה תקופה. השכבות היו מורכבות משלשות של בלוקים, כאשר בכל בלוק יש שתי שכבות קונבולוציה. בין כל שלשה יש הכפלה של מספר המסגנים והורדה של הממד פי שניים בעזרת pooling. ההיפר-פרמטרים הם: Batch normalization, lr=0.1, SGD+momentum=0.9, Xavier initialization, שיטת משקלים בתחילת האימון, validation error-שתיישר, batch size=256, weight decay=1e-5.

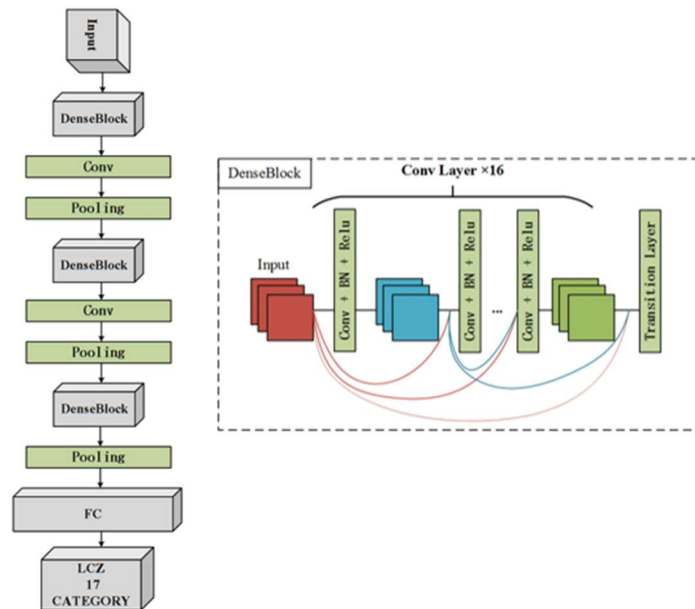
רשתות מתקדמות יותר שילבו את גישת ה-inception יחד עם ResNet על מנת לשלב בין היתרונות של שתי השיטות.



איור 5.12 Residual Block יחיד (ימין), וארכיטקטורת ResNet מלאה (שמאל).

5.2.5 Densely Connected Networks (DenseNet)

ניתן להרחיב את הרעיון של Residual Block כך שלא רק מחברים את הכניסה של כל בלוק למוצא שלו, אלא גם שומרים את הכניסה בפני עצמה, ובודקים את היחס שלה לשכבות יותר עמוקות. Dense block הוא בלוק בעל כמה שכבות, הבנוי כך שכניסה של כל שכבה מחוברת לכל הכניסות של השכבות אחריה. ניתן כמובן לשרשר כמה בלוקים כאלה יחד ולבצע ביניהם כל מיני פעולות כמו pooling או אפילו שכבת קונבולוציה עצמאית. כיוון שמשלבים כמה כניסות של בלוקים שונים, יש בעיה של התאמת ממדים, משום שכל בלוק מגדיל את מספר הערוצים, חיבור של כמה בלוקים יכולים ליצור מודל מורכב מדי. כדי להתגבר על בעיה זו הוספו שכבות transition בסוף כל dense block המבצעות קונבולוציות 1×1 עם רוחב צעד $s = 2$, ובכך מספר הערוצים נותר סביר והמודל לא נעשה מורכב מדי.

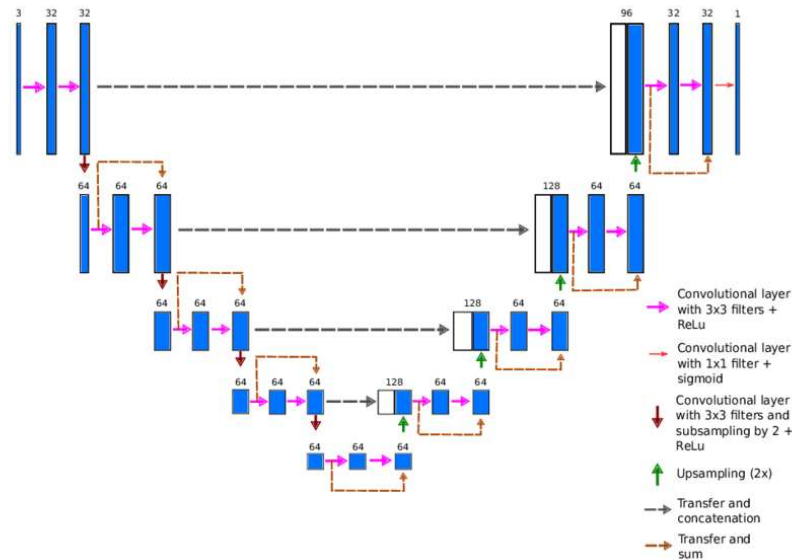


איור 5.13 Dense Block יחיד (ימין), וארכיטקטורת DenseNet מלאה (שמאל).

5.2.6 U-Net

ברשתות קונבולוציה המיועדות לסיווג, בסוף התהליך מתקבל וקטור של הסתברויות, כאשר כל איבר הוא הסתברות של label מסוים. במשימת סגמנטציה זה בעייתי, כיוון שצריך בסוף התהליך לא רק ללמוד את המאפיינים שבתמונה ועל פיהם לקבוע מה יש בתמונה, אלא צריך גם לשחזר את מיקומי הפיקסלים והתייגים שלהם ביחס לתמונה המקורית עם הסגמנטציה המתאימה. כדי להתמודד עם בעיה זו הוצעו את ארכיטקטורת U-Net, המכילה שלושה חלקים עיקריים: כיווץ, צוואר בקבוק והרחבה (contraction, bottleneck, and expansion section). כפי שניתן לראות

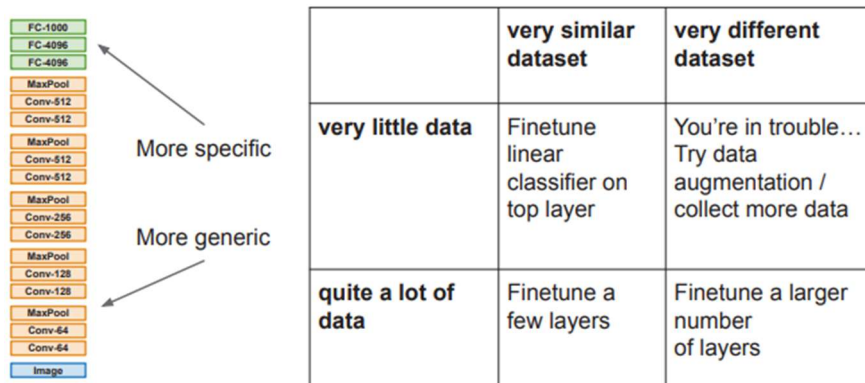
באזור, בחלק הראשון יש טופולוגיה רגילה של רשת קונבולוציה, המבוצעת בעזרת שכבות קונבולוציה וביצוע pooling. השוני בין השלב הזה לבין רשת קונבולוציה קלאסית הוא החיבור שיש בין כל שלב בתהליך לבין חלקים בהמשך התהליך. לאחר המעבר בצוואר הבקבוק יש למעשה שחזור של התמונה עם הסגמנטציה. השחזור נעשה בעזרת up-sampling על הווקטור שהתקבל במוצא צוואר הבקבוק יחד עם המידע שנשאר מהחלק הראשון של התהליך. פונקציית המחיר שמשמשים ברשת זו נקראת pixel-wise cross entropy loss, הבודקת כל פיקסל ביחס ל-label האמיתי אליו הוא שייך.



איור 5.14 ארכיטקטורת U-Net.

5.2.7 Transfer Learning

כאשר נתקלים במשימה חדשה, אפשר לתכנן עבורה ארכיטקטורה מסוימת ולאמן רשת עמוקה. בפועל זה יקר ומסובך להתאים רשת מיוחדת לכל בעיה ולאמן אותה מהתחלה, ולכן ניתן להשתמש ברשתות הקיימות שאומנו כבר ולהתאים אותן לבעיות אחרות. גישה זו נקראת Transfer Learning, וההיגיון מאחוריה טוען שעבור כמעט כל סוג דאטה השכבות הראשונות לומדות אותו דבר (זיהוי שפות, קווים וצורות כלליות, מאפיינים כללים וכו') ולכן ניתן להשתמש בהן פעמים רבות ללא שינוי כלל. משום כך, בפועל בדרך כלל לוקחים רשת קיימת ומחליפים בה את השכבות האחרונות או מוסיפים לה עוד שכבות בסופה, ואז מאמנים את השכבות החדשות על הדאטה החדש כך שהן תהיינה מוכונות לדאטה הספציפי של המשימה החדשה. ככל שיש יותר דאטה חדש ניתן להוסיף יותר שכבות ולקבל דיוק יותר טוב, וככל שהמשימה החדשה דומה יותר למשימה המקורית של הרשת כך יש צורך בפחות שכבות חדשות. כמו כן, משום שבשיטה זו נדרשים לאמן מספר שכבות קטן יותר, קטן ה-overfitting הנובע ממחוסר בדאטה.



איור 5.15 Transfer Learning.

5. References

Convolutional:

<https://github.com/technion046195/technion046195>

מצגות מהקורס של פרופ' יעקב גולדברגר

AlexNet:

<https://towardsdatascience.com/the-w3h-of-alexnet-vggnet-resnet-and-inception-7baaaecccc96>

VGG

<https://arxiv.org/abs/1409.1556>

GoogleNet

http://d2l.ai/chapter_convolutional-modern/vgg.html

ResNet

<https://arxiv.org/abs/1512.03385>

DenseNet

<https://arxiv.org/abs/1608.06993>

<https://towardsdatascience.com/review-densenet-image-classification-b6631a8ef803>

U-Net

<https://arxiv.org/abs/1505.04597>

6. Recurrent Neural Networks

היתרון של שכבות קונבולוציה על פני FC הוא ניצול הקשר המרחבי שיש בין איברים שונים בדאטה, כמו למשל פיקסלים בתמונה. יש סוגי דאטה בהם האיברים השונים יוצרים סדרה שיש לסדר האיברים חשיבות, כמו למשל טקסט, גלי קול, רצף DNA ועוד. כמובן שדאטה מהסוג הזה דורש מודל הנותן חשיבות לסדר של האיברים, מה שלא קיים ברשתות קונבולוציה. בנוסף, הרבה פעמים הממד של הקלט לא ידוע או משתנה, כמו למשל מספר המילים במשפט, וגם לכך יש לתת את הדעת. כדי להתמודד עם אתגרים אלו יש לבנות ארכיטקטורה שמקבלת סדרה של וקטורים ומוציאה וקטור יחיד, כאשר הווקטור היחיד מקודד קשרים על הדאטה המקורי שנכנס אליו. את וקטור המוצא ניתן להעביר בשכבת FC או בכל מסווג אחר, תלוי באופי המשימה.

6.1 Sequence Models

6.1.1 Vanilla Recurrent Neural Networks

רשתות רקורסיביות הן הכללה של רשתות נוירונים עמוקות, כאשר הן מכילות משקולות המאפשרות להן לתת משמעות לסדר של איברי הכניסה. ניתן להסתכל על משקולות אלה כרכיב זיכרון פנימי, כאשר כל איבר שנכנס משוקלל ביחס לפונקציה קבועה בתוספת רכיב משתנה שתלוי בערכי העבר. כאשר נכנס וקטור x , הוא מוכפל במשקל w_{xh} ונכנס לרכיב זיכרון h_t , כאשר h_t הוא פונקציה של x_t, h_{t-1} :

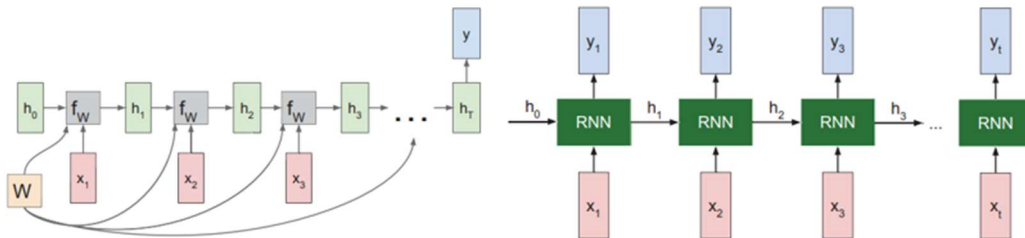
$$h_t = f(h_{t-1}, x_t)$$

מלבד המשקלים הפועלים על וקטור הכניסה, יש גם משקלים שפועלים על המשקולות הפנימיות (רכיב הזיכרון) – w_{hh} , ומשקלים הפועלים על המוצא של רכיב זה – w_{hy} . המשקלים w_{hx}, w_{hh}, w_{hy} זהים לכל השלבים, והם מתעדכנים ביחד. כמו כן, הפונקציה f_w היא קבועה לכל האיברים, למשל $\tanh, \text{sigmoid}$, או ReLU . באופן פורמלי התהליך נראה כך:

$$h_t = f_w(w_{hh}h_{t-1} + w_{xh}x_t), f_w = \tanh/\text{ReLU}/\text{sigmoid}$$

$$y_t = w_{hy}h_t$$

באופן סכמתי התהליך נראה כך:



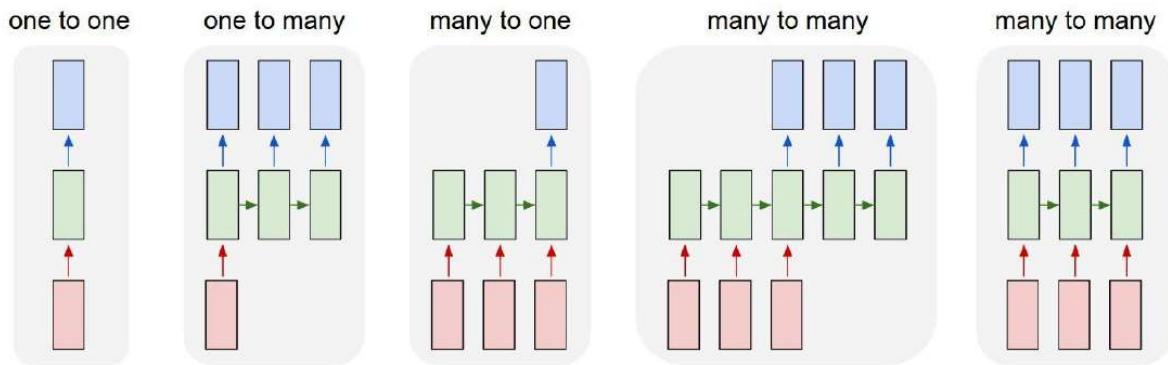
איור 6.1 ארכיטקטורות RNN בסיסיות: Many to Many (מימין) ו-Many to One (משמאל). על כל חץ יש משקל מתאים עליו מתבצעת הלמידה.

כמובן שניתן גם לשרשר שכבות חבויות ולקבל רשת עמוקה, כאשר פלט של שכבה מסוימת הופך להיות הקלט של השכבה הבאה. ישנם מודלים שונים של RNN, המתאימים לבעיות שונות:

One to many – יש קלט יחיד ורוצים להוציא סדרת פלטים, למשל מכניסים תמונה לרשת ורוצים משפט שיתאר אותה (Image captioning).

Many to one – רוצים לקבוע משהו יחיד עבור קלט סדרתי, למשל מקבלים משפט ורוצים לסווג את הסנטימנט שלו – האם הוא חיובי או שלילי.

Many to many – עבור כל סדרת קלט יש סדרת פלט, למשל תרגום משפה אחת לשפה אחרת – מקבלים משפט ומוציאים משפט.



איור 6.2 מודלים שונים של RNN.

6.1.2 Learning Parameters

הלמידה של הרשת נעשית בצורה דומה לרשתות שבפרקים הקודמים. עבור דאטה $x = (x_1, \dots, x_n), (y_1, \dots, y_n)$ נגדיר את פונקציית המחיר:

$$L(\theta) = \frac{1}{n} \sum_i L(\hat{y}_i, y_i, \theta)$$

כאשר הפונקציה $L(\hat{y}_i, y_i, \theta)$ תותאם למשימה – עבור משימות סיווג נשתמש ב-cross entropy ועבור בעיות רגרסיה נשתמש בקריטריון MSE. האימון יתבצע בעזרת GD, אך לא ניתן להשתמש ב-backpropagation הרגיל כיוון שכל משקל מופיע מספר פעמים – למשל w_{hx} פועל על כל הכניסות ו- w_{hh} פועל על כל רכיבי הזיכרון. כדי לבצע את עדכון המשקלים משתמשים ב-backpropagation through time (BPTT) – מסתכלים על הרשת הנפרשת כרשת אחת גדולה, מחשבים את הגרדיאנט עבור כל משקל, ואז סוכמים או ממצעים את כל הגרדיאנטים. אם הדאטה בכניסה הוא בגודל n , כלומר יש n דגימות בזמן, אז יש n רכיבי זיכרון, ו- $n - 1$ משקלים w_{hh} . לכן הגרדיאנט המשוקלל יהיה:

$$\frac{\partial L}{\partial w_{hh}} = \sum_{n-1} \frac{\partial L}{\partial w_{hh}(t)} \quad \text{or} \quad \frac{\partial L}{\partial w_{hh}} = \frac{1}{n-1} \sum_{n-1} \frac{\partial L}{\partial w_{hh}(t)}$$

כיוון שהמשקלים זהים לאורך כל הרשת, $w_{hh}(t) = w_{hh}$ והשינוי בזמן יהיה רק לאחר ביצוע ה-BPTT ויהיה רלוונטי רק לוקטור הבא.

הצורה הפשוטה של ה-BPTT יוצרת בעיה עם הגרדיאנט. נניח שרכיב הזיכרון מיוצג בעזרת הפונקציה הבאה:

$$h_t = f(z_t) = f(w_{hh}h_{t-1} + w_{hx}x_t + b_h)$$

לפי כלל השרשרת:

$$\frac{\partial h_n}{\partial x_1} = \frac{\partial h_n}{\partial h_{n-1}} \times \frac{\partial h_{n-1}}{\partial h_{n-2}} \times \dots \times \frac{\partial h_2}{\partial h_1} \times \frac{\partial h_1}{\partial x_1}$$

כיוון ש- w_{hh} קבוע ביחס לזמן עבור וקטור כניסה יחיד, מתקבל:

$$\frac{\partial h_t}{\partial h_{t-1}} = f'(z_t) \cdot w_{hh}$$

אם נציב זאת בכלל השרשרת, נקבל שעבור חישוב הנגזרת $\frac{\partial h_n}{\partial x_1}$ מכפילים $n - 1$ פעמים ב- w_{hh} . לכן אם מתקיים $\|w_{hh}\| > 1$ אז הגרדיאנט יתבדר, ואם $\|w_{hh}\| < 1$ הגרדיאנט יתאפס. בעיה זו, של התבדרות או התאפסות הגרדיאנט, יכולה להופיע גם ברשתות אחרות, אבל בגלל המבנה של RNN והלינאריות של ה-BPTT ברשתות רקורסיביות זה קורה כמעט תמיד.

עבור הבעיה של התבדרות הגרדיאנט ניתן לבצע clipping – אם הגרדיאנט גדול מקבוע מסוים, מנרמלים אותו:

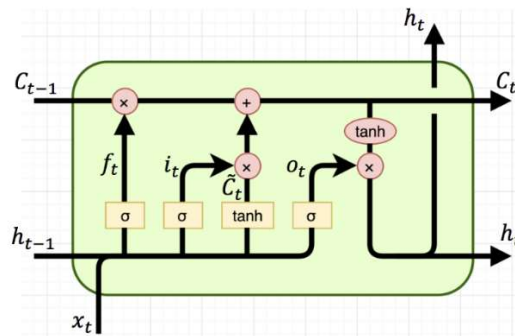
$$\text{if } \|g\| > c, \text{ then } g = \frac{cg}{\|g\|}$$

הבעיה של התאפסות הגרדיאנט אמנם לא גורמת לחישובים של מספרים עצומים, אך היא בעצם מבטלת את ההשפעה של איברים שנמצאים רחוק אחד מהשני. אם למשל יש משפט ארוך, אז במקרה בו הגרדיאנט דועך במהלך ה-BPTT, כמעט ואין השפעה של המילה הראשונה על המילה האחרונה. במילים אחרות – התאפסות הגרדיאנט גוררת בעיה של Long-term, כלומר קשה ללמוד דאטה בעל תלות בטווח ארוך, כמו משפט ארוך או תופעות שמשותפות לאט. בגלל הבעיה הזו לא משתמשים ב-RNN הקלאסי (שנקרא גם Vanilla RNN), אלא מבצעים עליו שיפורים, כפי שיוסבר בפרק הבא.

6.2 RNN Architectures

6.2.1 Long Short-Term Memory (LSTM)

כדי להתגבר על בעיית דעיכת הגרדיאנט המונעת מהרשת להשתמש בזיכרון ארוך טווח, ניתן להוסיף אלמנטים לרכיב הזיכרון כך שהוא לא יכיל רק מידע על העבר, אלא יהיה גם בעל שליטה על איך וכמה להשתמש במידע. ב-RNN הפשוט לרכיב הזיכרון יש שתי כניסות – h_{t-1}, x_t , ובעזרתן מחשבים את המוצא על ידי שימוש בפונקציה $f_w(h_{t-1}, x_t)$. למעשה רכיב הזיכרון הוא קבוע והלמידה מתבצעת רק במשקלים. ב-LSTM יש שני שינויים עיקריים – מלבד הכניסות הרגילות יש עוד כניסה הנקראת memory cell state ומסומנת ב- c_{t-1} , ובנוסף לכך h_t מחושב בצורה מורכבת יותר. באופן הזה האלמנט c_t דואג לזיכרון ארוך טווח של דברים, ו- h_t אחראי על הזיכרון של הטווח הקצר. נתבונן בארכיטקטורה של תא הזיכרון:



איור 6.3 תא זיכרון בארכיטקטורת LSTM.

הצמד $[x_t, h_{t-1}]$ נכנס לתא ומוכפל במשקל w , ולאחר מכן עובר בנפרד דרך ארבעה שערים (יש לשים לב שלא מבצעים פעולה בין x_t ל- h_{t-1} אלא הם נשארים בנפרד ואת כל הפעולות עושים על כל איבר בנפרד). **השער הראשון** נושא חדש, אז שער זה אמור למחוק את הנושא שהיה שמור בזיכרון. **השער השני** i_t הוא שער זיכרון והוא אחראי על כמה יש לזכור את המידע החדש לטווח ארוך. אם לדוגמה אכן יש במשפט מסוים נושא חדש, אז השער יחליט שיש לזכור את המידע הזה. אם לעומת זאת המידע החדש הוא תיאור שלא רלוונטי להמשך אז אין טעם לזכור אותו. **השער הרביעי** o_t הוא שער מוצא והוא אחראי על כמה מהמידע רלוונטי לדאטה הנוכחי x_t , כלומר מה יהיה הפלט של התא בהינתן מידע העבר. שלושת השערים האלו נקראים מסכות (Masks), והם מקבלים ערכים בין 0 ל-1 כיוון שהם עוברים דרך סיגמואיד. יש **שער נוסף** \tilde{c}_t (לפעמים מסומן באות g) שאחראי על השאלה כמה מהזיכרון להעביר לתא הבא. שער זה משקלל את המידע המתקבל יחד עם i_t שאומר עד כמה יש לזכור להמשך את המידע החדש.

באופן הזה מקבלים הן את h_t שאחראי על הזיכרון לטווח הקצר כמו ב-Vanilla RNN, והן את c_t שאחראי על זיכרון של כל העבר. ארכיטקטורת הרכיב מאפשרת להתייחס לאלמנטים נוספים הקשורים לזיכרון – ניתן לשכוח חלקים לא רלוונטיים של התא הקודם (f_t), להתייחס באופן סלקטיבי לכניסה (i_t) ולהוציא רק חלק מהמידע המשוקלל הקיים (o_t). באופן פורמלי ניתן לנסח את פעולת התא כך:

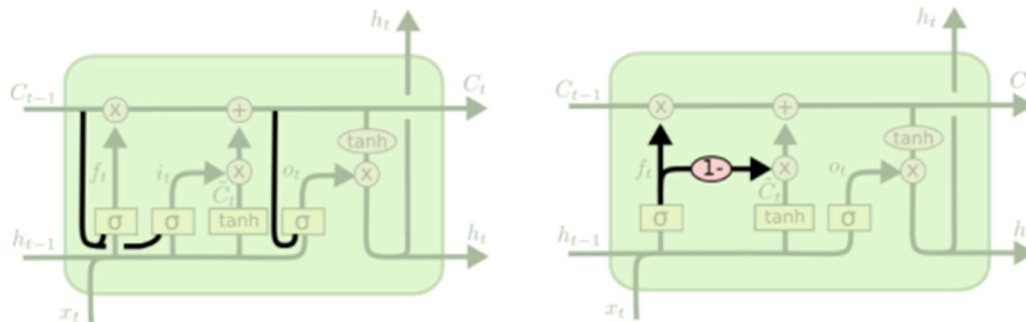
$$\begin{pmatrix} i \\ f \\ o \\ \tilde{c} \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix} = \begin{pmatrix} \sigma(w_i \cdot [x_t, h_{t-1}] + b_i) \\ \sigma(w_f \cdot [x_t, h_{t-1}] + b_f) \\ \sigma(w_o \cdot [x_t, h_{t-1}] + b_o) \\ \tanh(w_{\tilde{c}} \cdot [x_t, h_{t-1}] + b_{\tilde{c}}) \end{pmatrix}$$

$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t, h_t = o_t \odot \tanh(c_t)$$

כאשר האופרטור \odot מסמל כפל איבר איבר (כיוון שלשערים נכנס הזוג $[x_t, h_{t-1}]$, אם במוצא מבצעים מכפלה מסוימת, יש לבצע אותה על כל אחד מהאיברים).

יש וריאציות שונות של רכיבי LSTM – ניתן למשל לחבר את c_{t-1} לא רק למוצא h_t אלא גם לשאר השערים. חיבור כזה נקרא peepholes, כיוון שהוא מאפשר לשערים להתבונן ב- c_{t-1} באופן ישיר. יש ארכיטקטורות שמחברות את c_{t-1} לכלל השערים, ויש ארכיטקטורות שמחברות אותו רק לחלק מהשערים. חיבור כל השערים ל- c_{t-1} משנה כמובן את משוואות השערים. במקום $\sigma(w \cdot [x_t, h_{t-1}] + b)$, המשוואה החדשה תהיה $\sigma(w \cdot [c_{t-1}, x_t, h_{t-1}] + b)$.

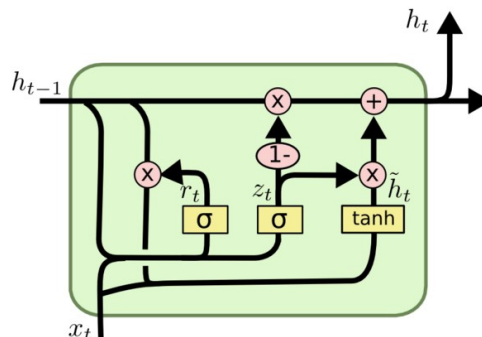
וריאציה אחרת של LSTM מאחדת בין שער השכחה f_t לבין שער הזיכרון i_t , וההחלטה עד כמה יש למחוק מידע מהזיכרון מתקבלת יחד עם ההחלטה כמה מידע חדש יש לכתוב. שינוי זה ישפיע על ה-memory cell, כאשר במקום $c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t$, משוואת העדכון תהיה: $c_t = f_t \odot c_{t-1} + (1 - f_t) \odot \tilde{c}_t$.



איור 6.4 וריאציות של LSTM – peephole connections (ימין) ו-coupled forget and input gates (שמאל).

6.2.2 Gated Recurrent Units (GRU)

ישנה ארכיטקטורה נוספת של תא זיכרון הנקראת Gated Recurrent Units (GRU), והיא כוללת מספר שינויים ביחס ל-LSTM:



איור 6.5 תא זיכרון בארכיטקטורת GRU.

השינוי המשמעותי מ-LSTM הוא העובדה שאין memory cell state, וכל השערים מתבססים רק על הקלט והמוצא של התא הקודם. כדי לאפשר זיכרון הן לטווח ארוך והן לטווח קצר, יש שני שערים – Update gate ו-Reset gate, והם מחושבים על פי הנוסחאות הבאות:

$$\text{Update: } z_t = \sigma(w_z \cdot [x_t, h_{t-1}])$$

$$\text{Reset: } r_t = \sigma(w_r \cdot [x_t, h_{t-1}])$$

בעזרת שער ה-reset מחשבים Candidate hidden state:

$$\tilde{h}_t = \tanh(w \cdot [x_t, r_t \odot h_{t-1}])$$

ראשית יש לשים לב כי $r_t \in [0, 1]$ כיוון שהוא תוצאה של סיגמואיד. כעת נתבונן על \tilde{h}_t ביחס לרכיב זיכרון פשוט של Vanilla RNN: $h_t = f_W(w_{hh}h_{t-1} + w_{hx}x_t)$. כאשר r_t קרוב ל-1 מתקבל הביטוי:

$$\tilde{h}_t = \tanh(w \cdot [x_t, r_t \odot h_{t-1}]) \approx \tanh(w[x_t, h_{t-1}]) = \tanh(w_{hx}x_t + w_{hh}h_{t-1})$$

המשמעות היא שעבור $r_t \rightarrow 1$ מתקבל רכיב הזיכרון הקלאסי, השומר על זיכרון לטווח קצר. אם לעומת זאת $r_t \rightarrow 0$, אז מתקבל $\tilde{h}_t \approx \tanh(w \cdot [x_t, 0 \odot h_{t-1}]) = \tanh(w_{xh} x_t)$, ולמעשה הזיכרון של הטווח הקצר מתאפס (reset).

לאחר החישוב של \tilde{h}_t מחשבים את המוצא של המצב החבוי בעזרת z_t , שגם הוא מקבל ערכים בין 0 ל-1:

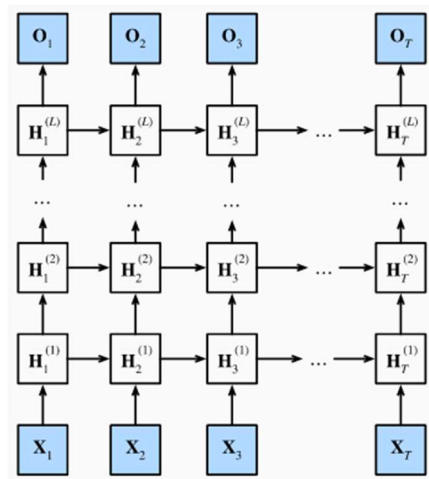
$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t$$

אם $z_t \rightarrow 0$, אז $h_t \approx h_{t-1}$, כלומר לא מתחשבים ב- \tilde{h}_t , ולמעשה מעבירים את המצב הקודם כמו שהוא. אם לעומת זאת $z_t \rightarrow 1$, אז $h_t \approx \tilde{h}_t$, כלומר מתעלמים מהמצב הקודם כמו שהוא ולוקחים את ה-Candidate hidden state, שהוא למעשה שקלול של המצב הקודם עם איבר הקלט הנוכחי. עבור כל ערך אחר של z_t , מקבלים שקלול של המצב החבוי הקודם וה-Candidate hidden state.

ארכיטקטורה זו מאפשרת גם לזכור דברים לאורך זמן, וגם מצליחה להתמודד עם בעיית הגרדיאנט. אם שער העדכון קרוב ל-1 כל הזמן, אז בעצם מעבירים את המצב החבוי כמו שהוא, ולמעשה הזיכרון נשמר לאורך זמן. בנוסף, אין בעיה של התבדרות הגרדיאנט, כיוון שאם השינוי בין תא לתא לא גדול, אז הגרדיאנט קרוב ל-1 כל הזמן ולא מתבדר.

6.2.3 Deep RNN

עד כה דובר על רכיבי זיכרון יחידים, שניתן לשרשר אותם יחד ולקבל שכבה שיכולה לנתח. ניתן להרחיב את המודל הפשוט לרשת בעל מספר שכבות עמוקות.



איור 6.6 ארכיטקטורת Deep RNN.

נתאר את הרשת באופן פורמלי. בכל נקודת זמן t יש וקטור כניסה $x_t \in \mathbb{R}^{n \times d}$ (וקטור בעל n איברים, כאשר כל איבר הוא מממד d). איברי הסדרה נכנסים לרשת בעלת L שכבות ו- T איברים בכניסה, כאשר עבור כל נקודת זמן יש L שכבות (או מצבים חבויים). כל שכבה מכילה h מצבים חבויים, כאשר השכבה ה- ℓ בנקודת זמן t מסומנת בתור $H_t^{(\ell)} \in \mathbb{R}^{n \times h}$. בכל נקודת זמן יש גם וקטור מוצא באורך n - $o_t \in \mathbb{R}^{n \times q}$. נסמן: $H_t^{(0)} = x_t$, ונניח שבין שכבה אחת לשנייה משתמשים באקטיבציה לא ליניארית ϕ . בעזרת סימונים אלה נקבל את הנוסחה הבאה:

$$H_t^{(\ell)} = \phi_\ell \left(H_t^{(\ell-1)} w_{xh}^{(\ell)} + H_{t-1}^{(\ell)} w_{hh}^{(\ell)} + b_h^{(\ell)} \right)$$

כאשר $w_{xh}^{(\ell)} \in \mathbb{R}^{h \times d}$, $w_{hh}^{(\ell)} \in \mathbb{R}^{h \times h}$, $b_h^{(\ell)} \in \mathbb{R}^{1 \times h}$. הפלט o_t תלוי באופן ישיר רק בשכבה ה- L , והוא מחושב על ידי:

$$o_t = H_t^{(L)} w_{hq} + b_q$$

כאשר $w_{hq}^{(L)} \in \mathbb{R}^{h \times q}$, $b_q^{(L)} \in \mathbb{R}^{1 \times q}$ הם הפרמטרים של שכבת הפלט.

ניתן כמובן להשתמש במצבים החבויים ברכיבי זיכרון GRU או LSTM, וכך לקבל Deep Gated RNN.

6.2.4 Bidirectional RNN

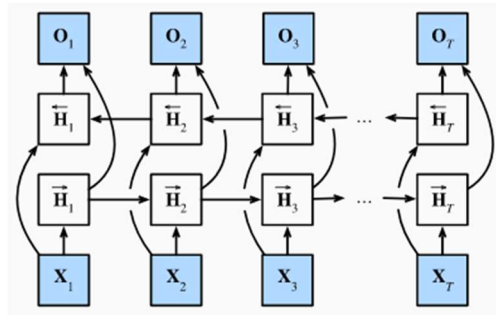
כל הרכיבים והרשתות שנידונו עד כה עסקו בסדרות סיבתיות, כלומר, סדרות בהן כל איבר מושפע מקודמיו אך לא מאלו הבאים אחריו. למשל, ערך מניה ביום מסוים קשור לערכה בימים הקודמים, אך הערך שלה ביום המחרת (שכלל עוד לא ידוע) לא משפיע בשום צורה על ערכה ביום הנוכחי. דוגמא נוספת – התפתחות של גל בזמן תלויה בערכי הקודמים של הגל אך אינה מושפעת ממצבי הגל בעתיד. זהו אמנם המצב היותר מצוי, אך ישנם מצבים בהם יש סדרה לאו דווקא סיבתית, כך שניתן לבחון את הקשר בין איבריה משני הכיוונים. ניקח לדוגמא את משימת ההשלמה הבאה:

I am __.

I am __ hungry.

I am __ hungry, and I can eat a big meal.

כעת נניח שבכל אחד מהמשפטים צריך לבחור את אחת מהמילים שבסט {happy, not, very}. כמובן שסוף הביטוי, במקרה וקיים, תורם מידע משמעותי על איזו מילה לבחור. מודל שאינו מסוגל לנצל את המידע לאחר המילה החסרה יכול לפספס מידע חשוב, ולרוב יכול לנחש מילה שאינה מסתדרת עם המשך המשפט הן מבחינה תחבירית והן מבחינת המשמעות. כדי לבנות מודל שמתייחס לכל חלקי המשפט, יש לתכנן ארכיטקטורה שמאפשרת לנתח סדרה משני הכיוונים שלה. ארכיטקטורה כזו נקראת Bidirectional RNN, והיא למעשה מבצעת ניתוח של סדרה משני הכיוונים שלה במקביל. באופן הזה כל מצב חבוי נקבע בו זמנית על ידי הנתונים של שני מצבים חבויים אחרים – זה שלפניו זה שאחריו.



איור 6.6 ארכיטקטורת Bidirectional RNN.

עבור כל כניסה $x_t \in \mathbb{R}^{n \times d}$ נחשב במקביל שני מצבים חבויים – $\vec{H}_t \in \mathbb{R}^{n \times h}$, $\overleftarrow{H}_t \in \mathbb{R}^{n \times h}$ כאשר h זה מספר רכיבי הזיכרון בכל מצב חבוי. כל מצב מחושב באופן הבא:

$$\vec{H}_t = \phi(x_t w_{xh}^{(f)} + \vec{H}_{t-1} w_{hh}^{(f)} + b_h^{(f)})$$

$$\overleftarrow{H}_t = \phi(x_t w_{xh}^{(b)} + \overleftarrow{H}_{t+1} w_{hh}^{(b)} + b_h^{(b)})$$

כאשר $w_{xh}^{(b)} \in \mathbb{R}^{d \times h}$, $w_{hh}^{(b)} \in \mathbb{R}^{h \times h}$, $b_h^{(b)} \in \mathbb{R}^{1 \times h}$ ו- $w_{xh}^{(f)} \in \mathbb{R}^{d \times h}$, $w_{hh}^{(f)} \in \mathbb{R}^{h \times h}$, $b_h^{(f)} \in \mathbb{R}^{1 \times h}$ הם הפרמטרים של המודל. לאחר החישוב של \vec{H}_t ו- \overleftarrow{H}_t משרשרים אותם יחד ומקבלים את $H_t \in \mathbb{R}^{n \times 2h}$, ובעזרתו מחשבים את המוצא:

$$o_t = H_t w_{hq} + b_q$$

כאשר $w_{hq} \in \mathbb{R}^{2h \times q}$, $b_q \in \mathbb{R}^{1 \times q}$ הם הפרמטרים של שכבת הפלט.

6.2.5 Sequence to Sequence Learning

ב

<https://buomssoo-kim.github.io/blog/tags/#attention-mechanism>

6. References

Vanilla:

<http://karpathy.github.io/2015/05/21/rnn-effectiveness/>

LSTM, GRU:

<http://colah.github.io/posts/2015-08-Understanding-LSTMs/>

Deep RNN, Bidirectional RNN:

http://d2l.ai/chapter_recurrent-modern/index.html

7. Deep Generative Models

המודלים שהוצגו בפרקים הקודמים הינם מודלים דיסקרימינטיביים, קרי הם מאומנים לבצע פעולות על בסיס דאטה נתון, אך לא יכולים ליצור פיסות מידע או דוגמאות חדשות בעצמם. בניגוד אליהם קיימים מודלים גנרטיביים, המסוגלים ליצור פיסות מידע חדשות על בסיס הדוגמאות שנלמדו. באופן פורמלי, בהינתן אוסף דוגמאות $X \in \mathbb{R}^{n \times d}$ ואוסף תגיות $Y \in \mathbb{R}^n$, מודל דיסקרימינטיבי מאומן לשערך את ההסתברות $\Pr(y|x)$. מודל גנרטיבי לעומת זאת לומד את ההסתברות $\Pr(x, y)$ (או את $\Pr(x)$ במקרה שהתגיות אינן נתונות), כאשר x, y הן צמד נתון של דוגמא ו-label, מתוכן ניתן לייצר דוגמאות חדשות.

ישנם שני סוגים עיקריים של מודלים גנרטיביים: סוג אחד של מודלים מאומן למצוא באופן מפורש את פונקציית הפילוג של הדאטה הנתון, ובעזרת הפילוג לייצר דוגמאות חדשות (על ידי דגימה מההתפלגות שנלמדה). סוג שני של מודלים אינו עוסק בשערך הפילוג של הדאטה המקורי, אלא מסוגל לייצר דוגמאות חדשות בדרכים אחרות. בפרק זה נדון במודלים הפופולריים בתחום – VAE, GANs, והשני של המודלים הגנרטיביים.

7.1 Variational AutoEncoder (VAE)

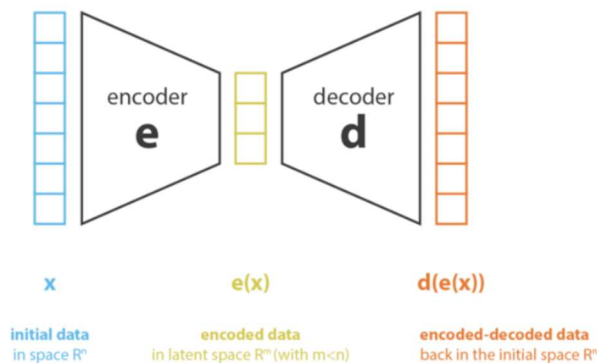
המודל הראשון הינו VAE, וכדי להבין כיצד ניתן בעזרתו לייצר מידע חדש, יש להסביר קודם מהם Autoencoders, כיצד הם עובדים ומה החסרונות שלהם. Autoencoder הינו רשת המאומנת לבצע הורדת ממד לדאטה, תוך איבוד מינימלי של מידע. בכדי להבין את המבנה ואופי הפעולה שלו, נקדים מעט על הורדת ממד באופן כללי.

7.1.1 Dimensionality Reduction

במקרים רבים, הדאטה אותו רוצים לנתח הוא בעל ממד גבוה, כלומר, לכל דגימה יש מספר רב של מאפיינים (features). לרוב, לא כל המאפיינים משמעותיים באותה מידה. לדוגמא – מחיר מניה של חברה מסוימת מושפע ממספר רב של גורמים, אך ככל הנראה גובה ההכנסות של החברה משפיע על מחיר המניה הרבה יותר מאשר הגיל הממוצע של העובדים. דוגמא נוספת – במשימת חיזוי גיל של אדם על פי תמונת הפנים שלו, לא כל הפיקסלים בתמונת הפנים יהיו בעלי אותה חשיבות ליצור החיזוי. כיוון שקשה לנתח דאטה מממד גבוה ולבנות מודלים עבור דאטה כזה, במקרים רבים מנסים להוריד את הממד של הדאטה תוך איבוד מידע מינימלי עד כמה שניתן. בתהליך הורדת הממד מנסים לקבל ייצוג חדש של הדאטה בעל ממד יותר נמוך, כאשר הייצוג הזה מורכב מהמאפיינים הכי משמעותיים של הדאטה. יש מגוון שיטות להורדת הממד כאשר הרעיון המשותף לכולן הוא לייצג את הדאטה בממד נמוך יותר, בו באים לידי ביטוי רק המאפיינים המשמעותיים של הדאטה.

הייצוג של הדאטה בממד נמוך נקרא הייצוג הלטנטי (חבוי) או הקוד הלטנטי, וכאמור, יותר קל לבנות מודלים למשימות שונות על סמך הייצוג הלטנטי של הדאטה מאשר לעבוד עם הדאטה המקורי. בכדי לקבל ייצוג לטנטי איכותי, ניתן לאמן אותו באמצעות מנגנון הנקרא decoder, הבוחן את יכולת השחזור של הדאטה מהייצוג הלטנטי שלו. ככל שניתן לשחזר בצורה מדויקת יותר את הדאטה מהייצוג הלטנטי, כלומר אובדן המידע בתהליך הוא קטן יותר, כך הקוד הלטנטי אכן מייצג בצורה אמינה את הדאטה המקורי.

תהליך האימון הוא דו שלבי: פיסת מידע המיוצגת על ידי וקטור המאפיינים $x \in \mathbb{R}^n$ עובר דרך encoder, שמטרתו להפיק מהדאטה את הייצוג הלטנטי שלו $e(x) \in \mathbb{R}^m$, כאשר $m < n$. לאחר מכן התוצאה מוכנסת ל-decoder בכדי לשחזר את הדאטה המקורי, ולבסוף מתקבל וקטור $d(e(x)) \in \mathbb{R}^n$. אם מתקיים השוויון $x = d(e(x))$ אז למעשה לא אבד שום מידע בתהליך, אך אם לעומת זאת $x \neq d(e(x))$ אז מידע מסוים אבד עקב הורדת הממד ולא היה ניתן לשחזר אותו במלואו בפענוח. באופן אינטואיטיבי, אם אנו מצליחים לשחזר את הדאטה המקורי מהייצוג שלו בממד הנמוך בדיוק טוב מספיק, כנראה שהייצוג הלטנטי הצליח להפיק את המאפיינים המשמעותיים של הדאטה המקורי.



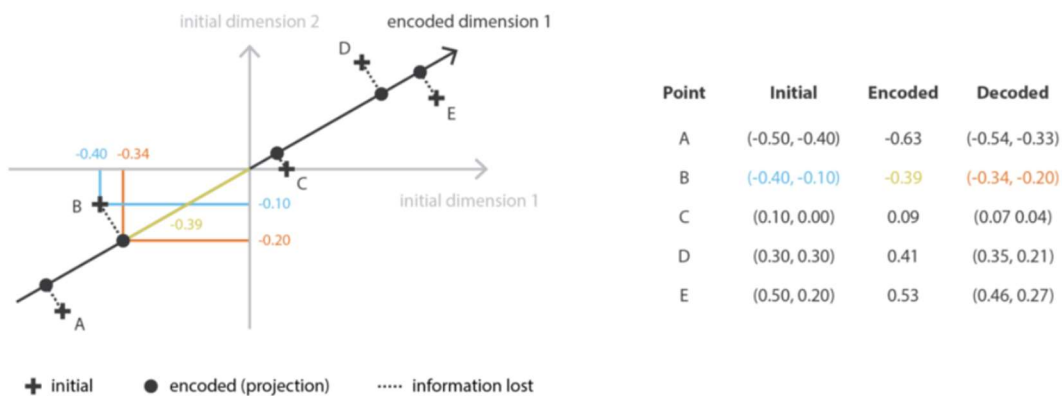
איור 7.1 ארכיטקטורת encoder-decoder.

כאמור, המטרה העיקרית של השיטות להורדת ממד הינה לקבל ייצוג לטנטי איכותי עד כמה שניתן. הדרך לעשות זאת היא לאמן את זוג ה-encoder-decoder השומרים על מקסימום מידע בעת הקידוד, ומביאים למינימום את שגיאת שחזור בעת הפענוח. אם נסמן בהתאמה E ו- D את כל הזוגות של encoder-decoder האפשריים, ניתן לנסח את בעיית הורדת הממד באופן הבא:

$$(e^*, d^*) = \arg \min_{(e,d) \in E \times D} \epsilon(x, d(e(x)))$$

כאשר $\epsilon(x, d(e(x)))$ הוא שגיאת השחזור שבין הדאטה המקורי לבין הדאטה המשוחזר.

אחת השיטות השימושיות להורדת ממד שאפשר להסתכל עליה בצורה הזו היא Principal Components Analysis (PCA). בשיטה זו מטילים (בצורה לינארית) דאטה מממד n לממד m , כך שהמאפיינים של הייצוג הלטנטי של הדוגמאות המקוריות יהיו אורתוגונליים. תהליך זה נקרא גם feature decorrelation, והמטרה שלו היא למזער את המרחק האוקלידי בין הדאטה המקורי לדאטה המשוחזר, בצורה לינארית גם כן, מהייצוג החדש במרחב ה- m ממדי.

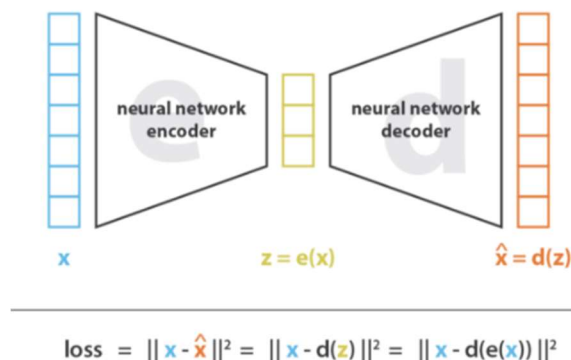


איור 7.2 דוגמא להורדת ממד בשיטת PCA.

במונחים של encoder-decoder, ניתן להראות כי אלגוריתם PCA מחפש את הזוג של encoder-decoder שמקיימים שני תנאים: א. ה-encoder מבצע טרנספורמציה לינארית על הדאטה כך שהמאפיינים החדשים (בממד נמוך) של הדאטה יהיו אורתוגונליים. ב. ה-decoder הלינארי המתאים יביא לשגיאה מינימלית במונחים של מרחק אוקלידי בין הדאטה המקורי לבין זה המשוחזר מהייצוג החדש. ניתן להוכיח שה-encoder האופטימלי מכיל את הווקטורים העצמיים של מטריצת ה-covariance של מטרצת ה-design, וה-decoder הוא השחלוף של ה-encoder.

7.1.2 Autoencoders (AE)

ניתן לקחת את המבנה של ה-encoder-decoder המתואר בפרק הקודם ולהשתמש ברשת נירונים עבור בניית הייצוג החדש ועבור השחזור. מבנה זה נקרא Autoencoder:

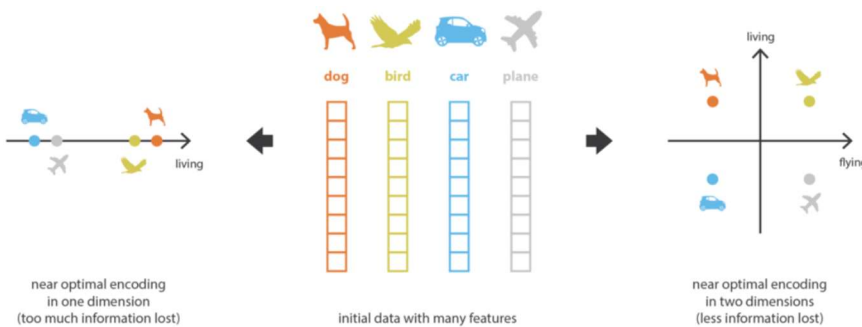


איור 7.3 Autoencoder – שימוש ברשתות נירונים עבור הורדת הממד והשחזור.

באופן הזה, הארכיטקטורה יוצרת לדאטה צוואר בקבוק מידעי (information bottleneck), שמבטיח שרק המאפיינים החשובים של הדאטה, שבאמצעותם ניתן לשחזר אותה בדיוק טוב, ישמשו לייצוג במרחב הלטנטי. במקרה הפשוט בו בכל רשת יש רק שכבה חבויה אחת והיא לא משתמשת בפונקציות הפעלה (activation functions) לא לינאריות, ניתן לראות כי ה-autoencoder יחפש טרנספורמציה לינארית של הדאטה, שבאמצעותו ניתן לשחזרו

באופן לינארי גם כן. בדומה ל-PCA, גם רשת כזו תחפש להוריד את הממד באמצעות טרנספורמציות לינאריות של המאפיינים המקוריים אך הייצוג בממד נמוך המופק על ידה לא יהיה בהכרח זהה לזה של PCA, כיוון שהבדיל מ-PCA המאפיינים החדשים (לאחר הורדת ממד) עשויים לצאת לא אורתוגונליים (-קורלציה שונה מ-0).

כעת נניח שהרשתות של ה-encoder וה-decoder הן רשתות עמוקות ומשתמשות בפונקציות הפעלה לא לינאריות. במקרה כזה, ככל שהארכיטקטורה של הרשתות מורכבת יותר, כך רשת ה-encoder יכולה להוריד יותר ממדים תוך יכולת באמצעות ה-decoder לבצע שחזור ללא כל איבוד מידע. באופן תיאורטי, אם ל-encoder ול-decoder יש מספיק דרגות חופש (למשל מספיק שכבות ברשת נורונים), ניתן להפחית ממד של כל דאטה לחד-ממד ללא כל איבוד מידע. עם זאת, הפחתת ממד דרסטית שכזו עלולה לגרום לדאטה המשוחזר לאבד את המבנה שלו. לכן יש חשיבות גדולה בבחירת מספר הממדים של המרחב הלטנטי, כך שמצד אחד אכן יתבצע ניפוי של מאפיינים פחות משמעותיים ומצד שני המידע עדיין יהיה בעל משמעות למשימות downstream שונות. כדי להמחיש את המתואר לעיל, ניקח לדוגמה מערכת שמקבלת תמונות של כלב, ציפור, מכונת ומטוס ומנסה למצוא את הפרמטרים העיקריים המבחינים ביניהם:



איור 7.4 דוגמה לשימוש ב-Autoencoder.

לפריטים אלו יש הרבה מאפיינים, וקשה לבנות מודל שמבחין ביניהם על סמך כל המאפיינים. רשת נורונים מורכבת מספיק מאפשרת לבנות ייצוג של כל הדוגמאות על קו ישר, כך שככל שפרט מסוים נמצא יותר ימינה, כך הוא יותר "חי". באופן הזה אמנם מתקבל ייצוג חד-ממדי, אבל הוא גורם לאיבוד המבנה הסמנטי של הדוגמאות ולא באמת ניתן להבין את ההפרדה ביניהן. לעומת זאת ניתן להוריד את הממד של תמונות אלו לדו-ממד ולהתייחס רק לפרמטרים "חי" ו"עף", וכך לקבל הבחנה יותר ברורה בין הדוגמאות. כמובן שהפרדה זו היא הרבה יותר פשוטה מאשר הסתכלות על כל הפרמטרים (-הפיקסלים) של הדוגמאות. דוגמה זו מראה את החשיבות שיש בבחירת הממדים של ה-encoder.

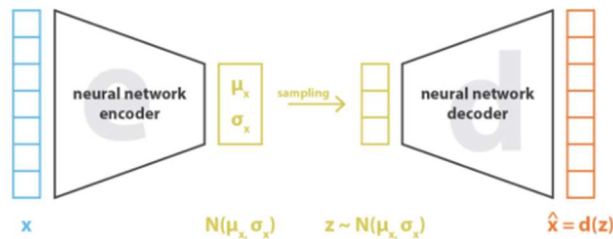
7.1.3 Variational AutoEncoders (VAE)

ניתן לקחת את ה-AE ולהפוך אותו למודל גנרטיבי, כלומר מודל שמסוגל לייצר בעצמו דוגמאות חדשות שאכן מתפלגות כמו הפילוג של הדאטה המקורי. אם מדובר בדומיין של תמונות למשל, אז נרצה שהמודל יהיה מסוגל לייצר תמונות שנראות אותנטיות ביחס לדאטה עליו אומן. AE מאומן לייצג את הדאטה בממד נמוך, שלוקח בחשבון את המאפיינים העיקריים, ולאחר מכן משחזר את התוצאה לממד המקורי. אולם, מנגנון זה אינו מאפשר להשפיע על האופן בו הדאטה מיוצג במרחב הלטנטי. אם יוגרל וקטור כלשהו מהמרחב הלטנטי – קרוב לוודאי שהוא לא יהווה ייצוג שדומה לדאטה המקורי. אם היינו מכניסים אותו ל-decoder, סביר להניח שהתוצאה לא תהיה דומה בכלל לדאטה המקורי. למשל אם AE אומן על אוסף של תמונות של כלבים ודוגמים באקראי וקטור מהמרחב הלטנטי שלו, הסיכוי לקבל תמונת כלב כלשהו לאחר השחזור של ה-decoder הינו אפסי.

כדי להתמודד עם בעיה זו, ניתן להשתמש ב-Variational AutoEncoder (VAE). בשונה מ-AE שלוקח דאטה ורק בונה לו ייצוג מממד נמוך, VAE קובע התפלגות פרירורית למרחב הלטנטי z – למשל התפלגות נורמלית עם תוחלת 0 ומטריצת covariance Σ . בהינתן התפלגות זו, רשת ה-encoder מאומנת לקבל דאטה x ולהוציא פרמטרים של התפלגות פוסטרירורית $z|x$, מתוך מטרה למזער כמה שניתן את המרחק בין ההתפלגויות $z|x$ ו- z . לאחר מכן דוגמים וקטורים מההתפלגות הפוסטרירורית $z|x$ (הנתונה על ידי הפרמטרים המחושבים ב-encoder), ומעבירים אותם דרך ה-decoder כדי לייצר פרמטרים של ההתפלגות $x|z$. חשוב להבהיר שאם הדאטה המקורי הוא תמונה המורכבת מאוסף של פיקסלים, אזי במוצא יתקבל $x|z$ לכל פיקסל בנפרד ומההתפלגות הזו דוגמים נקודה שתייצג את ערך הפיקסל בתמונה המשוחזרת.

באופן הזה, הלמידה דואגת לא רק להורדת הממד של הדאטה, אלא גם להתפלגות המושרית על המרחב הלטנטי. כאשר ההתפלגות המותנית במוצא $x|z$ טובה, קרי קרובה להתפלגות המקורית של x , ניתן בעזרתה גם ליצור דוגמאות חדשות, ובעצם מתקבל מודל גנרטיבי.

כאמור, ה-encoder מנסה לייצג את הדאטה המקורי באמצעות התפלגות בממד נמוך יותר, למשל התפלגות נורמלית עם תוחלת ומטריצת covariance: $z \sim p(z|x) = N(\mu_x, \sigma_x)$. חשוב לשים לב להבדל בתפקיד של ה-decoder – בעוד שב-AE הוא נועד לתהליך האימון בלבד ובפועל מה שחשוב זה הייצוג הלטנטי, ב-VAE ה-decoder חשוב לא פחות מאשר הייצוג הלטנטי, כיוון שהוא זה שמשמש ליצירת דאטה חדש לאחר תהליך האימון, או במילים אחרות, הוא הופך את המערכת למודל גנרטיבי.



איור 7.5 ארכיטקטורה של VAE.

לאחר שהוצג המבנה הכללי של VAE, ניתן לתאר את תהליך האימון, ולשם כך נפריד בשלב זה בין שני החלקים של ה-VAE. ה-encoder מאמן רשת שמקבלת דוגמאות מסט האימון, ומנסה להפיק מהן פרמטרים של התפלגות $z|x$ הקרובים כמה שניתן לפרמטרים של ההתפלגות הפריורית z , שכאמור נקבעה מראש. מההתפלגות הנלמדת הזו דוגמים וקטורים לטנטיים חדשים ומעבירים אותם ל-decoder. ה-decoder מבצע את הפעולה ההפוכה – לוקח וקטור שנדגם מהמרחב הלטנטי $z|x$, ומייצר באמצעותו דוגמה חדשה שאמורה להיות דומה לדאטה המקורי. תהליך האימון יהיה כזה שימזער את השגיאה של שני חלקי ה-VAE – גם $x|z$ שבמוצא יהיה כמה שיותר קרוב ל- x המקורי, וגם ההתפלגות $z|x$ תהיה כמה שיותר קרובה להתפלגות הפריורית z .

נתאר באופן פורמלי את בעיית האופטימיזציה ש-VAE מנסה לפתור. נסמן את הווקטורים של המרחב הלטנטי ב- z , את הפרמטרים של ה-decoder ב- θ , ואת הפרמטרים של ה-encoder ב- λ . כדי למצוא את הפרמטרים האופטימליים של שתי הרשתות, נרצה להביא למקסימום את $p(X; \theta)$, כלומר למקסם את הנראות המרבית של סט האימון תחת θ . כיוון שפונקציית \log מונוטונית, נוכל לקחת את לוג ההסתברות:

$$L(\theta) = \log p(x; \theta)$$

אם נביא למקסימום את הביטוי הזה, נקבל את ה- θ האופטימלי. כיוון שלא ניתן לחשב במפורש את $p(x; \theta)$, יש להשתמש בקירוב. נניח והפלט של ה-encoder הוא בעל התפלגות $q(z|x; \lambda)$ (מה ההסתברות לקבל את z בהינתן x בכניסה), ונסה לייצג את ההתפלגות הזו בעזרת רשת נירונים עם סט פרמטרים λ . כעת ניתן לחלק ולהכפיל את $L(\theta)$ ב- $q(z; \lambda)$:

$$\log p(x; \theta) = \log \sum_z p(x, z; \theta) = \log \sum_z q(z; \lambda) \frac{p(x, z; \theta)}{q(z; \lambda)} \geq \sum_z q(z; \lambda) \log \frac{p(x, z; \theta)}{q(z; \lambda)}$$

כאשר אי השוויון האחרון נובע [מאי-שוויון ינסן](#), והביטוי שמימין לאי השוויון נקרא Evidence Lower Bound ($ELBO(\theta, \lambda)$). ניתן להוכיח שההפרש בין ה- $ELBO$ לבין הערך שלפני הקירוב הוא המרחק בין שתי ההתפלגויות $p(z|x)$, $q(z)$ שנקרא Kullback-Leibler divergence ומסומן ב- \mathcal{D}_{KL} :

$$\log p(x; \theta) = ELBO(\theta, \lambda) + \mathcal{D}_{KL}(q(z; \lambda) || p(z|x; \theta))$$

אם שתי ההתפלגויות זהות, אזי מרחק \mathcal{D}_{KL} ביניהן הוא 0 ומתקבל שוויון: $\log p(x; \theta) = ELBO(\theta, \lambda)$. כזכור, אנחנו מחפשים למקסם את פונקציית המחיר $\log p(x; \theta)$, וכעת בעזרת הקירוב ניתן לרשום:

$$L(\theta) = \log p(x; \theta) \geq ELBO(\theta, \lambda)$$

$$\rightarrow \theta_{ML} = \arg \max_{\theta} L(\theta) = \arg \max_{\theta} \max_{\lambda} ELBO(\theta, \lambda)$$

כעת ניתן בעזרת שיטת Gradient Descent (GD) למצוא את האופטימום של הביטוי, וממנו להפיק את הפרמטרים האופטימליים של ה-encoder ושל ה-decoder. נפתח יותר את ה- $ELBO(\theta, \lambda)$ עבור VAE, ביחס לשתי התפלגויות:

$p(x|z; \theta)$ – ההסתברות ש-decoder עם סט פרמטרים θ יוציא x בהינתן z .
 $q(z|x; \lambda)$ – ההסתברות ש-encoder עם סט פרמטרים λ יוציא את z_i בהינתן x בכניסה.
 לפי הגדרה:

$$\begin{aligned}
 ELBO(\theta, \lambda) &= \sum_z q(z|x; \lambda) \log p(x, z; \theta) - \sum_z q(z|x; \lambda) \log q(z|x; \lambda) \\
 & \text{את הביטוי } \log p(x, z; \theta) \text{ ניתן לפתוח לפי חוק בייס } p(x, z) = p(x|z) \cdot p(z) \\
 &= \sum_z q(z|x; \lambda) (\log p(x|z; \theta) + \log p(z; \theta)) - \sum_z q(z|x; \lambda) \log q(z|x; \lambda) \\
 &= \sum_z q(z|x; \lambda) \log p(x|z; \theta) - \sum_z q(z|x; \lambda) (\log q(z|x; \lambda) - \log p(z; \theta)) \\
 &= \sum_z q(z|x; \lambda) \log p(x|z; \theta) - \sum_z q(z|x; \lambda) \frac{\log q(z|x; \lambda)}{\log p(z; \theta)} \\
 & \text{הביטוי השני לפי הגדרה שווה ל-} \mathcal{D}_{KL}(q(z|x; \lambda) \| p(z; \theta)), \text{ לכן מתקבל:} \\
 &= \sum_z q(z|x; \lambda) \log p(x|z; \theta) - \mathcal{D}_{KL}(q(z|x; \lambda) \| p(z))
 \end{aligned}$$

הביטוי הראשון הוא בדיוק התוחלת של $\log p(x|z; \theta)$. תחת ההנחה ש- z מתפלג נורמלית, ניתן לרשום:

$$= \mathbb{E}_{q(z|x; \lambda)} \log N(x; \mu_\theta(z), \sigma_\theta(z)) - \mathcal{D}_{KL}(N(\mu_\lambda(x), \sigma_\lambda(x)) \| N(0, I))$$

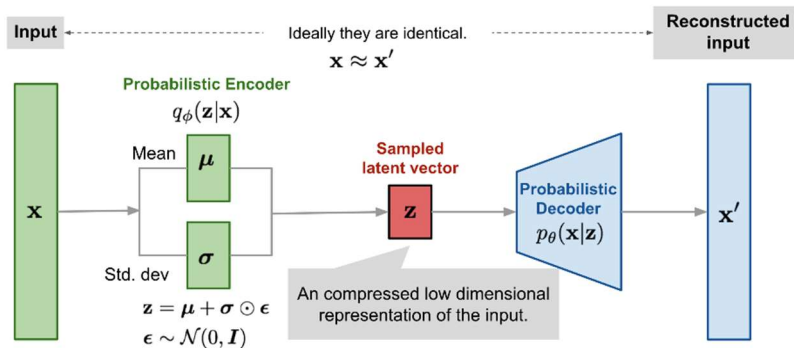
כדי לחשב את התוחלת ניתן פשוט לדגום דוגמאות מההתפלגות $z|x \sim N(\mu_\theta(x), \sigma_\theta(x))$ ולקבל:

$$\mathbb{E}_{q(z|x; \lambda)} \log N(x; \mu_\theta(z), \sigma_\theta(z)) \approx \log N(x; \mu_\theta(x), \sigma_\theta(x))$$

ועבור הביטוי השני יש נוסחה סגורה:

$$\mathcal{D}_{KL}(N(\mu, \sigma^2) \| N(0, I)) = \frac{1}{2}(\mu^2 + \sigma^2 - \log \sigma^2)$$

כעת משיש בידינו נוסחה לחישוב פונקציית המחיר, נוכל לבצע את תהליך הלמידה. יש לשים לב שפונקציית המחיר המקורית הייתה תלויה רק ב- θ , אך באופן שפיתחנו אותה היא למעשה דואגת גם למזעור הפרש בין הכניסה של VAE לבין המוצא שלו, וגם למזעור המרחק בין ההתפלגות הפריורית של z לבין ההתפלגות $z|x$ שבמוצא ה-encoder.



$$\begin{aligned}
 x_t &\rightarrow \mu_\lambda(x_t), \Sigma_\lambda(x_t) \rightarrow z_t \sim \mathcal{N}(\mu_\lambda(x_t), \Sigma_\lambda(x_t)) \rightarrow \mu_\theta(z_t), \Sigma_\theta(z_t) \\
 ELBO &= \sum_t \log \mathcal{N}(x_t; \mu_\theta(z_t), \Sigma_\theta(z_t)) - \mathcal{D}_{KL}(\mathcal{N}(\mu_\lambda(x_t), \Sigma_\lambda(x_t)) \| \mathcal{N}(0, I))
 \end{aligned}$$

איור 7.6 תהליך הלמידה של VAE.

כאשר נתון אוסף דוגמאות X , ניתן להעביר כל דוגמא x_t ב-encoder ולקבל עבודה את $\mu_\lambda, \sigma_\lambda$. לאחר מכן דוגמים וקטור לטנטי z מההתפלגות עם פרמטרים אלו, מעבירים אותו ב-decoder ומקבלים את $\mu_\theta, \sigma_\theta$. לאחר התהליך ניתן להציב את הפרמטרים המתקבלים ב-ELBO ולחשב את ערך פונקציית המחיר. ניתן לשים לב שה-ELBO מורכב משני איברים – האיבר הראשון משערך את הדמיון בין הדוגמא שבכניסה לבין ההתפלגות שמתקבלת במוצא, והאיבר השני מבצע רגולריזציה להתפלגות הפריורית במרחב הלטנטי. הרגולריזציה גורמת לכך שההתפלגות במרחב הלטנטי $z|x$ תהיה קרובה עד כמה שניתן להתפלגות הפריורית z . אם ההתפלגות במרחב הלטנטי קרובה להתפלגות הפריורית, אז ניתן בעזרת ה-decoder ליצור דוגמאות חדשות, ובמובן הזה ה-VAE הוא מודל גנרטיבי.

הדגימה של z מההתפלגות במרחב הלטנטי יוצרת קושי בחישוב הגרדיאנט של ה-ELBO, לכן בדרך כלל מבצעים Reparameterization trick – דוגמים z_0 מהתפלגות נורמלית סטנדרטית, ואז כדי לקבל את ערך הדגימה של z משתמשים בפרמטרים של ה-encoder: $z = z_0 \sigma_\lambda(x) + \mu_\lambda(x)$. בגישה הזו כל התהליך נהיה דטרמיניסטי – מגרילים z_0 מראש ואז רק נשאר לחשב באופן סכמתי את התפשטות הערך ברשת.

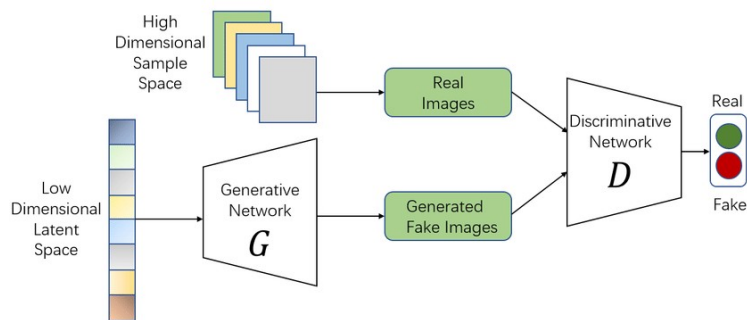
7.2 Generative Adversarial Networks (GANs)

גישה אחרת של מודל גנרטיבי נקראת Generative Adversarial Networks או בקיצור GANs, ובשונה מ-VAE בגישה זו לא מנסים לשערך התפלגות של דאטה בצורה מפורשת (על ידי מציאת הפרמטרים הממקסמים את הנראות המירבית של סט האימון), אלא יוצרים דאטה באופן אחר. הרעיון הוא לאמן שתי רשתות במקביל – רשת אחת שלומדת לייצר דוגמאות, ורשת שניה שלומדת להבחין בין דוגמא אמיתית מסט האימון לבין תמונה סינתטית שנוצרה על ידי הרשת הראשונה. הרשת הראשונה מאומנת ליצור דוגמאות שיגרמו לרשת השנייה לחשוב שהן אמיתיות, בזמן שהמטרה של הרשת השנייה היא לא לתת לרשת הראשונה לבלבל אותה. באופן הזה הרשת הראשונה מהווה למעשה מודל גנרטיבי, שלאחר שלב האימון היא מסוגלת לייצר דאטה סינתטי שלא ניתן להבחין בינו לבין דאטה אמיתי.

7.2.1 Generator and Discriminator

בפרק זה נסביר את המבנה של ה-GAN הקלאסי שהומצא בשנת 2014 על ידי Ian Goodfellow ושותפיו. נציין כי קיימים מאות רבות של וריאנטים שונים של GAN שהוצעו מאז, ועדיין תחום זה פעיל מאוד מבחינה מחקרית.

כאמור, GAN מבוסס על שני אלמנטים מרכזיים – רשת שיוצרת דאטה (generator) ורשת שמכריעה האם הדאטה הזו סינתטי או אמיתי (discriminator), כאשר האימון נעשה על שתי הרשתות יחד. ה-discriminator מקבל כקלט הן את הדוגמאות האמיתיות והן את הפלט של ה-generator, כדי ללמוד להבחין בין דאטה אמיתי לבין דאטה סינתטי. ה-generator מייצר דוגמאות ומקבל פידבק מה-discriminator וכך לומד לייצר דוגמאות שנראות אמיתיות. נסמן את ה-generator ב-G ואת ה-discriminator ב-D, ונקבל את הסכמה הבאה:



איור 7.7 ארכיטקטורת GAN.

ה-discriminator D הוא למעשה מסווג שהפלט שלו הוא ההסתברות שהקלט הינו דוגמא אמיתית, ונסמן ב- $D(x)$ את ההסתברות הזו. כדי לאמן את ה-discriminator נרצה להשיג שני דברים: א. למקסם את $D(x)$ עבור x מסט האימון, כלומר, לטעות כמה שפחות בזיהוי דאטה אמיתי. ב. למזער את $D(x)$ עבור דאטה סינתטי, כלומר, לזהות נכון כמה שיותר דוגמאות סינתטיות שיוצרו על ידי ה-generator. באופן דומה נרצה לאמן את ה-generator כך שהדגימות שהוא מייצר תהיינה כמה שיותר דומות לדוגמאות אמיתיות, כלומר ה-generator מעוניין לגרום ל-discriminator להוציא ערכים כמה שיותר גבוהים עבור הדאטה הסינתטי שהוא מייצר. בשביל לאמן יחד את שני חלקי המודל, נבנה פונקציית מחיר בעלת שני איברים, באופן הבא:

$$V(D, G) = \min_G \max_D \mathbb{E}_{x \sim \text{Data}} \log D(x) + \mathbb{E}_{z \sim \text{Noise}} \log(1 - D(G(z)))$$

נסביר את הביטוי המתקבל: ה-discriminator מעוניין למקסם את פונקציית המחיר, כך שהערך של $D(x)$ יהיה כמה שיותר קרוב ל-1 ו- $D(G(z))$ יהיה כמה שיותר קרוב ל-0. ה-generator לעומת זאת רוצה להביא למינימום את פונקציית המחיר, כך ש- $D(G(z))$ יהיה כמה שיותר קרוב ל-1, כלומר ה-discriminator יחשוב ש- $G(z)$ הוא דאטה אמיתי.

כעת האימון נעשה באופן איטרטיבי, כאשר פעם אחת מקבעים את G ומאמנים את D , ופעם אחת מקבעים את D ומאמנים את G . אם מקבעים את G , אז למעשה מאמנים מסוג בינארי, כאשר מחפשים את האופטימום התלוי בוקטור הפרמטרים ϕ_d :

$$\max_{\phi_d} \mathbb{E}_{x \sim \text{Data}} \log D_{\phi_d}(x) + \mathbb{E}_{z \sim \text{Noise}} \log \left(1 - D_{\phi_d}(G_{\theta_g}(z)) \right)$$

אם לעומת זאת מקבעים את D , אז ניתן להתעלם מהאיבר הראשון כיוון שהוא פונקציה של ϕ_d בלבד וקבוע ביחס ל- θ_g . לכן נשאר רק לבדוק את הביטוי השני, שמחפש את ה-generator שמייצר דאטה שנראה אמיתי בצורה הטובה ביותר:

$$\min_{\theta_g} \mathbb{E}_{z \sim \text{Noise}} \log \left(1 - D_{\phi_d}(G_{\theta_g}(z)) \right)$$

כאמור, המטרה היא לאמן את G בעזרת D (במצבו הנוכחי), כדי שיהיה מסוגל ליצור דוגמאות הנראות אותנטיות. האימון של ה-generator נעשה באמצעות Gradient Descent (מזעור פונקציית המחיר ביחס ל- θ_g), והאימון של ה-discriminator נעשה באמצעות Gradient Ascent (מקסום פונקציית המחיר ביחס ל- ϕ_d). האימון מתבצע במשך מספר מסוים של Epochs, כאשר כאמור מאמנים לסירוגין את G ו- D . בפועל דוגמים mini-batch בגודל m מסת האימון (x_1, \dots, x_m) ו- m דגימות של רעש (z_1, \dots, z_m) , ומכניסים את הקלט ל- G . הגרדיאנט של פונקציית המחיר לפי הפרמטרים של ה-generator במהלך האימון מחושב באופן הבא:

$$\nabla_{\theta} V(G_{\theta}, D_{\phi}) = \frac{1}{m} \nabla_{\theta} \sum_{i=1}^m \log \left(1 - D_{\phi}(G_{\theta}(z_i)) \right)$$

וכאשר מאמנים את ה-discriminator, הגרדיאנט נראה כך:

$$\nabla_{\phi} V(G_{\theta}, D_{\phi}) = \frac{1}{m} \nabla_{\phi} \sum_{i=1}^m \log D_{\phi}(x_i) + \log \left(1 - D_{\phi}(G_{\theta}(z_i)) \right)$$

נהוג לבצע מודיפיקציה קטנה על פונקציית המטרה של ה-generator. כיוון שבהתחלה הדגימות המיוצרות על ידי ה-generator לא דומות לחלוטין לאלו מסת האימון, ה-discriminator מזהה אותן בקלות כמזויפות. כתוצאה מכך הביטוי $D(G(z))$ מקבל ערכים מאוד קרובים ל-0, וממילא גם הביטוי $\mathbb{E}_{z \sim \text{Noise}} \log \left(1 - D(G(z)) \right)$ קרוב ל-0. עניין זה גורם לכך שהגרדיאנט של ה-generator גם יהיה מאוד קטן, ולכן כמעט ולא מתבצע שיפור ב-generator. לכן במקום לחפש מינימום של הביטוי $\mathbb{E}_{z \sim \text{Noise}} \log \left(1 - D(G(z)) \right)$ מחפשים מינימום לביטוי $-\mathbb{E}_{z \sim \text{Noise}} \log \left(D(G(z)) \right)$. הביטויים לא שווים לגמרי אך שניהם מובילים לאותו פתרון של בעיית האופטימיזציה אותה הם מייצגים, והביטוי החדש עובד יותר טוב נומרית ומצליח לשפר את ה-generator בצורה יעילה יותר.

הערכים האופטימליים של G ו- D :

כזכור, פונקציית המחיר הינה:

$$V(D, G) = \min_G \max_D \mathbb{E}_{x \sim \text{Data}} \log D(x) + \mathbb{E}_{z \sim \text{Noise}} \log \left(1 - D(G(z)) \right)$$

כעת נרצה לחשב מה הערך האופטימלי של ה-discriminator עבור generator נתון, ועבורו לחשב את הערך של פונקציית המחיר. לשם הנוחות נסמן את התפלגות הדאטה האמיתי ב- p_r , ואת התפלגות הדאטה הסינתטי המיוצר על ידי ה-generator ב- p_g . עבור G קבוע, ניתן לרשום את פונקציית המחיר כך:

$$V(D, G) = \int_x p_r(x) \log D(x) + p_g(x) \log(1 - D(x)) dx$$

כדי להביא את הביטוי הזה למקסימום, נרצה למקסם את האינטגרל עבור כל ערכי x האפשריים. לכן הפונקציה לה מעוניינים למצוא אופטימום הינה:

$$f(D(x)) = p_r(x) \log D(x) + p_g(x) \log(1 - D(x))$$

נגזור את הביטוי האחרון ונשווה ל-0 בכדי למצוא את הערך האופטימלי של $D(x)$ עבור x נתון:

$$\frac{\partial f(D(x))}{\partial D(x)} = \frac{p_r(x)}{D(x)} - \frac{p_g(x)}{1 - D(x)} = 0$$

$$\rightarrow p_r(x)(1 - D(x)) - p_g(x)D(x) = 0$$

$$D(x)_{opt} = \frac{p_r(x)}{p_r(x) + p_g(x)}$$

הביטוי שהתקבל הינו הערך האופטימלי של ה-discriminator עבור generator קבוע (ביחס לקלט x נתון). נשים לב שעבור המקרה בו ה-GAN מצליח לייצר דוגמאות שנראות אמיתיות לחלוטין, כלומר $p_g(x) = p_r(x)$, אז מתקיים $D(x) = \frac{1}{2}$. הסתברות זו משמעותה שה-discriminator לא יודע להחליט לגבי הקלט המתקבל, והוא קובע שההסתברות שהקלט אמיתי זהה לזו שהקלט סינתטי.

כעת נבחן מהו ערך פונקציית המחיר כאשר D אופטימלי:

$$\begin{aligned} V(G, D) &= \mathbb{E}_{x \sim Data} \log D(x) + \mathbb{E}_{z \sim Noise} \log(1 - D(G(z))) \\ &= \mathbb{E}_{x \sim Data} \log\left(\frac{p_r(x)}{p_r(x) + p_g(x)}\right) + \mathbb{E}_{z \sim Noise} \log\left(1 - \left(\frac{p_r(x)}{p_r(x) + p_g(x)}\right)\right) \\ &= \mathbb{E}_{x \sim Data} \log\left(\frac{p_r(x)}{p_r(x) + p_g(x)}\right) + \mathbb{E}_{z \sim Noise} \log\left(\frac{p_g(x)}{p_r(x) + p_g(x)}\right) \\ &= \mathbb{E}_{x \sim Data} \log\left(\frac{p_r(x)}{\frac{(p_r(x) + p_g(x))}{2}}\right) + \mathbb{E}_{z \sim Noise} \log\left(\frac{p_g(x)}{\frac{(p_r(x) + p_g(x))}{2}}\right) - \log 4 \end{aligned}$$

הביטוי המתקבל הינו המרחק בין ההתפלגויות p_r ו- p_g , והוא נקרא Jensen-Shannon divergence ומסומן ב- \mathcal{D}_{JS} . מרחק זה הינו גרסה סימטרית של Kullback-Leibler divergence (\mathcal{D}_{KL}), ועבור שתי התפלגויות P, Q הוא מוגדר באופן הבא:

$$\mathcal{D}_{JS} = \frac{1}{2} \mathcal{D}_{KL}(P||M) + \frac{1}{2} \mathcal{D}_{KL}(Q||M), M = \frac{1}{2}(P + Q)$$

קיבלנו שעבור D אופטימלי, פונקציית המחיר שווה למרחק \mathcal{D}_{JS} בין p_r לבין p_g עד כדי קבוע, ובאופן מפורש:

$$V(G, D_{opt}) = \mathcal{D}_{JS}(p_r, p_g) - \log 4$$

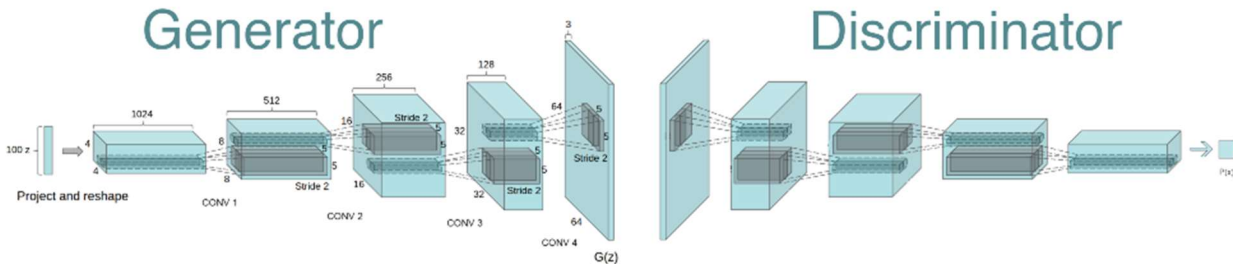
כאשר G אופטימלי ומתקיים $p_g(x) = p_r(x)$, אז המרחק בין ההתפלגויות שווה 0, כלומר $\mathcal{D}_{JS}(p_r, p_g) = 0$, ולכן מתקבל:

$$V(G_{opt}, D_{opt}) = -\log 4$$

יש משמעות גדולה לביטוי שהתקבל – ככל שנצליח למזער יותר את $\mathcal{D}_{JS}(p_r, p_g)$, כך נצליח לקבל GAN יותר טוב.

7.2.2 Deep Convolutional GAN (DCGAN)

כפי שהוסבר בפרק 5, רשתות קונבולוציה יעילות יותר בדומיין של תמונות מאשר רשתות FC. לכן היה טבעי לקחת רשתות קונבולוציה ולבנות בעזרתן generator ו-discriminator עבור דומיין של תמונות. ה-generator מקבל וקטור אקראי ומעביר אותו דרך רשת קונבולוציה על מנת ליצור תמונה, וה-discriminator מקבל תמונה ומעביר אותה דרך רשת קונבולוציה שעושה סיווג בינארי אם התמונה אמיתית או סינתטית. DCGAN הומצא ב-2015 ומאז פותחו רשתות שמייצרות תמונות יותר איכותיות הן מבחינת הרזולוציה והן מבחינת הדמיון שלהן לתמונות אמיתיות, אך החשיבות של המאמר נעוצה בשימוש ברשתות קונבולוציה עבור GAN שמיועד לדומיין של תמונות.



איור 7.8 ארכיטקטורת DCGAN.

7.2.3 Conditional GAN (cGAN)

לעיתים מודל גנרטיבי נדרש לייצר דוגמא בעלת מאפיין ספציפי ולא רק דוגמא שנראית אותנטית. למשל, עבור אוסף תמונות המייצגות את הספרות מ-0 עד 9, ונרצה שה-GAN ייצר תמונה של ספרה מסוימת. במקרים אלו, בנוסף לווקטור הכניסה z , ה-GAN מקבל תנאי נוסף על הפלט אותו הוא צריך לייצר, כמו למשל ספרה ספציפית אותה רוצים לקבל. GAN כזה נקרא conditional GAN (או בקיצור cGAN), ופונקציית המחיר שלו דומה מאוד לפונקציית המחיר של GAN רגיל למעט העובדה שהביטויים הופכים להיות מותנים:

$$\mathcal{L}_c(D, G) = \min_G \max_D \mathbb{E}_{x \sim \text{Data}} \log D(x|y) + \mathbb{E}_{z \sim \text{Noise}} \log(1 - D(G(z|y)))$$

7.2.4 Pix2Pix

כפי שראינו, ה-GAN הקלאסי שתואר לעיל מסוגל לייצר דוגמאות חדשות מווקטור אקראי z , המוגרל מהתפלגות מסוימת (בדרך כלל התפלגות גאוסית סטנדרטית, אך זה לא מוכרח). ישנן גישות נוספות ליצור דאטה חדש, כמו למשל ייצור תמונה חדשה על בסיס קווי מתאר כלליים שלה. סט האימון במקרה זה בנויה מזוגות של תמונות והסקיצות שלהן.

שיטת Pix2Pix משתמשת בארכיטקטורה של GAN אך במקום לדגום את וקטור z מהתפלגות כלשהי, מקבלת סקיצה של תמונה בתור קלט, וה-generator לומד להפוך את הסקיצה לתמונה אמיתית. הארכיטקטורה של ה-generator נשארת ללא שינוי ביחס למה שתואר קודם לכן (פרט להתאמה למבנה הקלט), אך ה-discriminator נבנה כעת – במקום לקבל תמונה ולבצע עליה סיווג בינארי, הוא מקבל זוג תמונות – את הסקיצה ואת התמונה (פעם תמונה מסט האימון המתאימה לסקיצה S ופעם זאת שמיוצרת על ידי ה-generator על בסיס S). על ה-discriminator לקבוע האם התמונה היא אכן תמונה אמיתית של הסקיצה או תמונה סינתטית. ווריאציה זו של ה-GAN משנה גם את פונקציית המחיר – כעת ה-generator צריך ללמוד שני דברים – גם ליצור תמונות טובות כך שה-discriminator יאמין שהן אמיתיות, וגם למזער את המרחק בין התמונה שנוצרת לבין תמונה אמיתית השייכת לסקיצה.

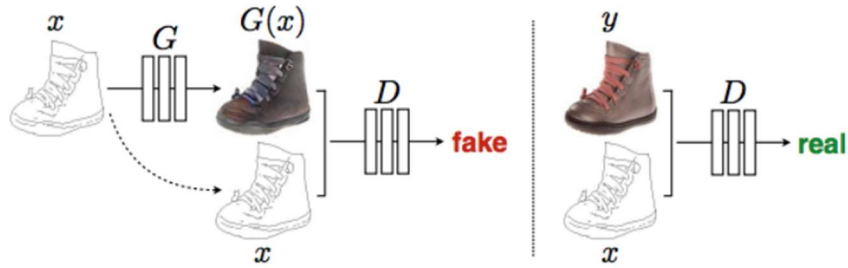
כעת נסמן תמונה אמיתית השייכת לסקיצה ב- y , ונרשום את פונקציית המחיר כשני חלקים נפרדים – cross entropy רגיל של GAN ומרחק אוקלידי L_1 בין תמונת המקור לבין הפלט:

$$V(D, G) = \min_G \max_D \mathbb{E}_{x, y} (\log D(x, y) + \log(1 - D(x, G(x))))$$

$$\mathcal{L}_{L_1}(G) = \min_{\theta_g} \mathbb{E}_{x, y} \|G(x) - y\|_1$$

$$\mathcal{L}(G, D) = V(D, G) + \lambda \mathcal{L}_{L_1}(G)$$

ניתן להסתכל על pix2pix בתור GAN הממפה תמונה לתמונה (image-to-image translation). נציין שבמקרה זה הקלט והפלט של pix2pix שייכים לתחומים (domains) שונים (סקיצה ותמונה רגילה).

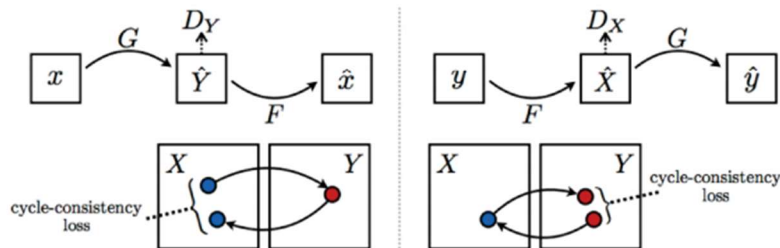


איור 7.9 ארכיטקטורת Pix2Pix - Image-to-Image Translation

7.2.5 CycleGAN

ב-Pix2Pix הדאטה המקורי הגיע בזוגות – סקיצה ואיתה תמונה אמיתית. זוגות של תמונות זה לא דבר כל כך זמין, ולכן שיפרו את תהליך האימון כך שיהיה ניתן לבצע אותו על שני סטים של דאטה מתחומים שונים. הארכיטקטורה עבור המשימה הזו מורכבת משני generators – בהתחלה מכניסים דוגמא מהדומיין הראשון x ל- G , שמנסה להפוך אותו לדוגמא מהדומיין השני y , והפלט נכנס ל- D_y discriminator שנועד לזהות האם התמונה שהתקבלה הינה אמיתית או לא (עבור F לא רק ל- F אלא גם ל- D_y discriminator שנועד לזהות האם התמונה שהתקבלה הינה אמיתית או לא (עבור הדומיין של y). ניתן לבצע את התהליך הזה באופן דואלי עבור y – מכניסים את y ל- F על מנת לקבל את x ואת המוצא מכניסים ל- D_x discriminator בכדי לבצע סיווג בינארי ול- G על מנת לנסות לשחזר את המקור. ה-generator השני F נועד לשפר את תהליך הלמידה – לאחר ש- x הופך ל- y דרך G , ניתן לקבל חזרה את x אם נעביר את y דרך F מתוך ציפייה לקבל $x \approx F(G(x))$. התהליך של השוואת הכניסה למוצא נקרא cycle-consistency, והוא מוסיף עוד איבר לפונקציית המחר, שמטרתו למזער עד כמה שניתן את המרחק בין התמונה המקורית לתמונה המשוחזרת:

$$V(D_x, D_y, G, F) = \mathcal{L}_{GAN}(G, D_y, x, y) + \mathcal{L}_{GAN}(F, D_x, x, y) + \lambda (\mathbb{E}_x \|F(G(x)) - x\|_1 + \mathbb{E}_y \|G(F(y)) - y\|_1)$$

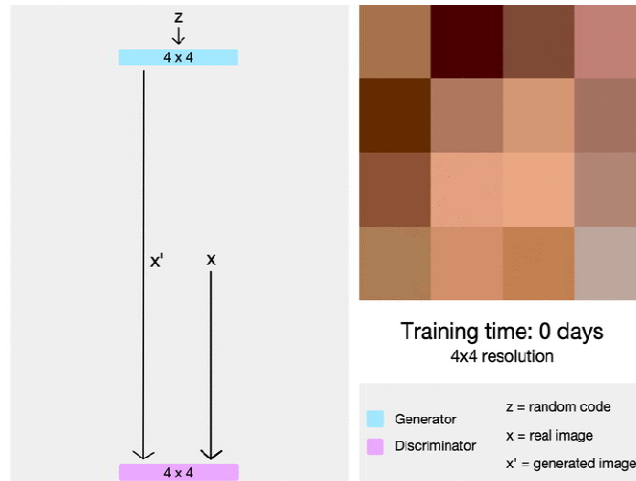


איור 7.10 ארכיטקטורת CycleGAN

7.2.6 Progressively Growing GAN (ProGAN)

כאמור לעיל, עבור דומיין של תמונות, הגיוני להשתמש ברשתות קונבולוציה עבור יצירת תמונות חדשות, וזה הרעיון הבסיסי שמאחורי DCGAN. למרות היכולת המרשימה של DCGAN ביצירה של תמונה באיכות גבוהה, יכולת זאת מוגבלת לתמונות בגודל מסוים. ככל שהרזולוציה של תמונה גבוהה יותר, כך יותר קל להבחין אם תמונה זו אמיתית או נוצרה על ידי רשת גנרטיבית. בעוד ש-DCGAN מצליח ליצור תמונות שנראות אותנטיות בגדלים של 32×32 , 64×64 , ואפילו 128×128 , הוא מתקשה ביצירת תמונות ברזולוציות גבוהות יותר, כמו למשל רזולוציה של 256×256 . ProGAN בא לתת מענה לכך, והוא היה ה-GAN הראשון שפרץ את מחסום הרזולוציה והצליח ליצור תמונות איכותיות מאוד (במאמר המקורי של ProGAN – עד רזולוציה של 1024×1024) בלי שיהיה ניתן להבחין שתמונות אלה סינתטיות. אמנם עוד לפני ProGAN היו GANs שהצליחו ליצור תמונה בעלת רזולוציה גבוהה מתמונה אחרת ברזולוציה גבוהה (pix2pix), אך זו משימה אחרת, מכיוון שבשבילה צריך רק ללמוד לשנות תכונות של תמונת קלט, ולא לייצר תמונה חדשה לגמרי מאפס.

הרעיון העיקרי מאחורי ProGAN, שהוצע ב-2017 על ידי חוקרים מחברת Nvidia, הינו לייצר תמונות ברזולוציה הולכת וגדלה בצורה הדרגתית. כלומר, במקום לנסות לאמן את כל השכבות של ה-generator בבת אחת, כפי שנעשה בכל ה-GANs לפני כן, ניתן לאמן אותו לייצר תמונות ברזולוציה משתנה – בהתחלה הוא מתאמן לייצר תמונות ברזולוציות מאוד נמוכה (4×4), לאחר מכן המשיכו ליצירת תמונות ברזולוציה 8×8 , אחר כך 16×16 , וכך הלאה עד יצירה של תמונה ברזולוציה של 1024×1024 .



איור 7.11 ארכיטקטורת ProGAN.

כדי לאמן GAN לייצר תמונות בגודל 4×4 , התמונות מסט האימון הוקטנו לגודל זה (down-sampling). אחרי שה-GAN לומד לייצר תמונות בגודל 4×4 , מוסיפים לו עוד שכבה המאפשרת להכפיל את גודל התמונות המיוצרות, קרי ליצור תמונות בגודל 8×8 . יש לציין שהאימון של הרשת עם השכבה הנוספת מתחיל עם המשקלים שאומנו קודם לכן, אך לא "מקפיאיים" אותם, כלומר הם מעודכנים גם כן תוך כדי אימון הרשת בשביל ליצור תמונה ברזולוציה כפולה. הגדלה הדרגתית של הרזולוציה מאלצת את הרשתות להתמקד תחילה בפרטים "הגסים" של התמונה (דפוסים בתמונה מטושטשת מאוד). לאחר מכן הרשת "לומדת" לבצע up-sampling (להכפיל את הרזולוציה) של התמונות המטושטשות האלה. תהליך זה משפר את איכות התמונה הסופית כיוון שבאופן זה הסבירות שהרשת תלמד דפוסים שגויים קטנה משמעותית.

7.2.7 StyleGAN

StyleGAN, שיצא בשלהי שנת 2018, מציע גרסה משודרגת של ProGAN, עם דגש על רשת ה-generator. מחברי המאמר שמו לב כי היתרון הפוטנציאלי של שכבות ProGAN המייצרות תמונה בצורה הדרגתית נובע מיכולתן לשלוט בתכונות (מאפיינים) ויזואליות שונות של התמונה, אם משתמשים בהן כראוי. ככל שהשכבה והרזולוציה נמוכה יותר, כך התכונות שהיא משפיעה עליהן גסות יותר.

למעשה, StyleGAN הינו ה-GAN הראשון שנותן יכולת לשלוט במאפיינים ויזואליים (אומנם לא בצורה מלאה) של התמונה הנוצרת. מחברי StyleGAN חילקו את התכונות הוויזואליות של תמונה ל-3 סוגים:

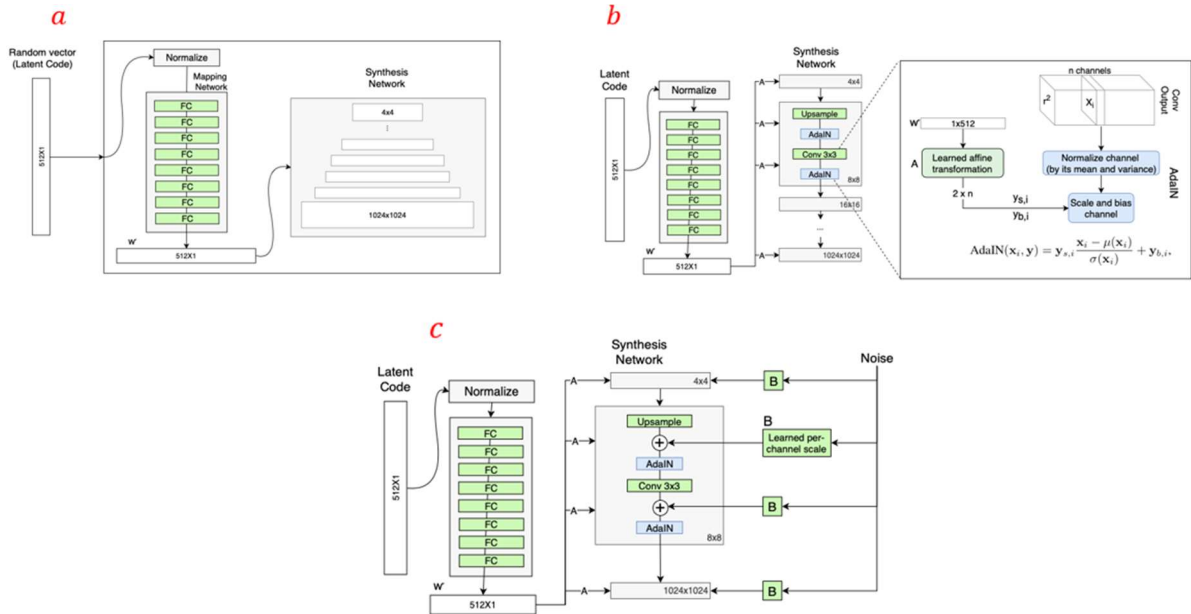
- **גס:** משפיע על תנוחה, סגנון שיער כללי, צורת פנים וכו'.
- **אמצעית:** משפיעה על תווי פנים עדינים יותר, סגנון שיער, עיניים פקוחות/עצומות וכדו'.
- **רזולוציה דקה:** משפיעה על צבע (עיניים/שיער/עור) ועל שאר תכונות המיקרו של תמונה.

כדי להעניק ל-StyleGAN את היכולות האלו, נדרשים מספר שינויים ביחס לארכיטקטורה של ProGAN (נתאר רק את שלושת השינויים החשובים ביותר כאן):

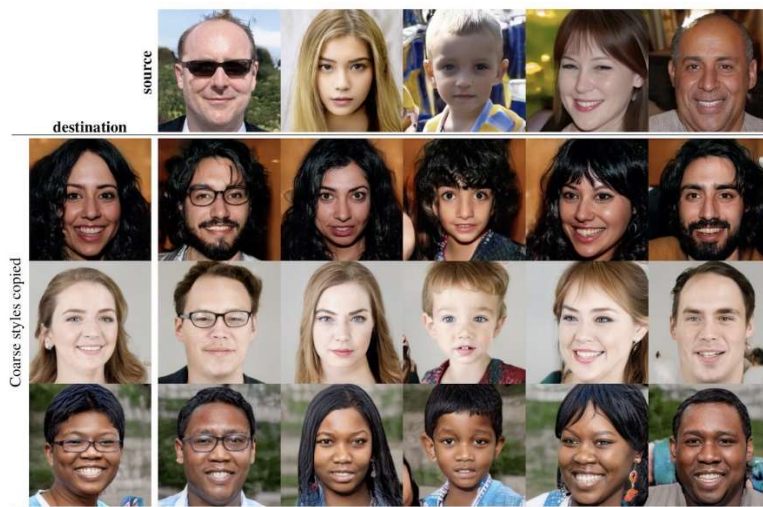
- **הוספת רשת מיפוי:** מטרת רשת המיפוי היא קידוד וקטור הקלט לווקטור ביניים w (הנקרא וקטור סגנון) אשר האיברים השונים שלו שולטים בתכונות ויזואליות שונות של התמונה הנוצרת. זהו תהליך לא טריוויאלי מכיוון שהיכולת של הרשת לשלוט בתכונות ויזואליות באמצעות וקטור הקלט הינה מוגבלת. הסיבה לכך טמונה בעובדה שווקטור הקלט נאלץ "לעקוב אחר צפיפות ההסתברות של סט האימון" שגורם לתופעה הנקראת feature entanglement (FE) (-ערבוב מאפיינים). FE בין תכונות צבע השיער והמגדר יכול להופיע אם למשל בסט האימון יש מגמה כללית של גברים עם שיער קצר ונשים בעלות שיער ארוך. במקרה זה הרשת תלמד שגברים יכולים להיות בעלי שיער קצר בלבד ולהיפך אצל נשים. כתוצאה מכך, אם "נשחק" עם רכיבי וקטור הקלט כדי לייצר תמונה של גבר בעל שיער ארוך, בסופו של דבר מגדרו ישתנה גם כן ונקבל תמונה של אישה.

רשת המיפוי שהתווספה לארכיטקטורה הופכת את וקטור הקלט לווקטור ביניים w שאינו צריך לעקוב אחר התפלגות של סט האימון, וכך יש פחות ערבוב המאפיינים. במילים אחרות, רשת זו מאפשרת את היכולת לשלוט במאפיינים ויזואליים של התמונה הנוצרת באמצעות שינוי רכיביו של וקטור w . רשת המיפוי מורכבת משמונה שכבות FC וגודל הפלט שלה זהה לגודל הקלט.

- **החלפת BN ב-AdaIN:** רשתות הקונבולוציה של ה-generator, שנועדו ליצירת תמונות ברזולוציות שונות משתמשות במנגנון שנקרא AdaIN (במקום Batch Normalization). בשונה מ-BN, הפרמטרים של הממוצע ושל השונות בגישת AdaIN נלמדים מווקטור הסגנון w (הם בעצם טרנספורמציה לינארית של w עם משקלים נלמדים). להבדיל מ-AdaIN, במנגנון BN סטנדרטי פרמטרים אלו נלמדים כמו המשקלים האחרים ולא תלויים במוצא של שכבה כלשהי.
- **יתור על אקראיות של וקטור קלט:** ב-StyleGAN וקטור הקלט אינו וקטור המוגרל מהתפלגות גאוסית אלא וקטור דטרמיניסטי עם רכיבים נלמדים. וקטורי הרעש מתווספים ישירות לפלטים של ערוצי קונבולוציה ברשתות ה-generator כאשר העוצמה שלהן נלמדת לכל ערוץ בנפרד. שימוש בוקטור קלט דטרמיניסטי במקום בוקטור אקראי מקל ככל הנראה על הפרדת המאפיינים על ידי רשת המיפוי (יותר קל לעשות זאת על וקטור קבוע מאשר להתאים את משקלי רשת המיפוי לווקטורי כניסה אקראיים).



איור 7.12 השינויים העיקריים בארכיטקטורת StyleGAN. (a) הוספת רשת מיפוי. (b) שימוש ב-AdaIN. (c) שימוש בקלט דטרמיניסטי. יש עוד כמה שינויים יותר מינוריים ב-StyleGAN יחסית ל-ProGAN, כמו שינוי של היפר פרמטרים של הרשתות, פונקציית מחיר וכו'. התוצאות הן לא פחות ממרשימות – StyleGAN יוצר תמונות שנראות ממש אמיתיות ובנוסף מקנה יכולת לשלוט בחלק מהתכונות החזותיות של התמונות.



איור 7.13 תמונות שיוצרו באמצעות StyleGAN.

7.2.8 Wasserstein GAN

אחד סוגי ה-GAN החשובים ביותר הינו Wasserstein GAN, והוא נוגע בבעיה שיש בפונקציית המחיר בה משתמשים הרבה וריאנטים של GAN-ים. כאמור, תהליך הלמידה של הרשת המייצרת דאטה – ה-generator – נעשה באמצעות משוב המתקבל מה-discriminator. בעוד שה-discriminator מאומן להבחין בין דאטה אמיתי לדאטה סינתטי הן בעזרת דאטה אמיתי והן בעזרת דאטה שה-generator מייצר, ה-generator לא מסתמך על דוגמאות אמיתיות אלא רק על המשוב מה-discriminator. משום כך, בתחילת הלמידה, כאשר ה-generator עוד לא מאומן, הדוגמאות הסינתטיות שהוא מייצר אינן דומות כלל לדאטה האמיתי, וה-discriminator מבחין בקלות ביניהם. במילים אחרות, בתחילת תהליך הלמידה ה-discriminator טוב יותר מאשר ה-generator. פער זה יוצר בעיה בתהליך ההשתפרות של ה-generator, כיוון שהשיפור מתבסס על ה"דע" העובר ל-generator בעזרת הגרדיאנט של פונקציית המחיר (loss) שלו, התלוי בערכים אותם מוציא ה-discriminator. כדי להבין מדוע תהליך העברת המידע באופן הזה בעייתי, יש להרחיב מעט על תהליך היצירה של הדאטה על ידי ה-generator ואיך ה-discriminator מסתכל על דאטה זה.

ההנחה היסודית ברוב המודלים הגנרטיביים, ובפרט ב-GANs, הינה שהדאטה הרב ממדי (למשל תמונות) "חי" במשטח מממד נמוך בתוכו. אפשר להסתכל על משטח בתור הכללה של תת-מרחב וקטורי מממד נמוך הנפרס על ידי תת-קבוצה של וקטורי בסיס של מרחב וקטורי מממד גבוה יותר. גם המשטח נוצר מתת-קבוצה של וקטורי הבסיס של "מרחב האם", אך ההבדל בינו לבין תת-מרחב וקטורי מתבטא בכך שלמשטח עשויה להיות צורה מאוד מורכבת יחסית לתת-מרחב וקטורי. משתמע מכך שניתן לייצר דאטה רב ממדי על ידי טרנספורמציה של וקטור ממרחב בעל ממד נמוך (וקטור לטנטי). למשל, ניתן בעזרת רשת נוירונים לייצר תמונה בגודל $12k > 64 \times 64 \times 3$ פיקסלים מווקטור באורך 100 בלבד. זאת אומרת, שגם התפלגות התמונות של הרשת הגנרטיבית וגם ההתפלגות של הדאטה האמיתי נמצאים ב"משטח בעל ממד נמוך" בתוך מרחב בעל ממד גבוה של הדאטה המקורי. באופן פורמלי יותר, משטח זה נקרא יריעה (manifold), וההשערה שתוארה מעלה מהווה הנחת יסוד בתחום הנקרא למידת יריעות (manifold learning). מכיוון שמדובר במשטחים בעלי ממד נמוך בתוך מרחב בעל ממד גבוה, קיימת סבירות גבוהה שלא יהיה שום חיתוך בין המשטח בו "חי" הדאטה האמיתי לבין זה של הדאטה הסינתטי (לכל הפחות בתחילת תהליך האימון של ה-GAN), ויתרה מכך, המרחק בין משטחים אלה עשוי להיות די גדול. מכך נובע שה-D discriminator עשוי ללמוד להבחין בין הדאטה האמיתי לסינתטי בקלות, כיוון שבמרחב מממד גבוה יש מרחק גדול בין יריעה אמיתית לבין היריעה של הדאטה הסינתטי. בנוסף, D כנראה ייתן לדוגמאות סינתטיות ציונים (score) ממש קרובים לאפס כי אכן קל מאוד למצוא "משטח הפרדה" בין שתי היריעות – זה של הדוגמאות האמיתיות וזה של הסינתטיות, כיוון שהם נוטים להיות רחוקים מאוד אחד מהשני.

רקע זה מסייע להבין מדוע הפער שיש בין ה-generator וה-discriminator מבחינת אופי הלמידה מהווה בעיה. כאמור, ה-generator מעדכן את המשקלים שלו על סמך הציונים שהוא מקבל מה-discriminator (דרך פונקציית המחיר של ה-GAN). אבל אם ה-discriminator כל הזמן מוציא ציונים מאוד נמוכים (עקב מרחק גדול בין היריעות שתואר מעלה) לדוגמאות המיוצרות על ידי ה-generator, ה-generator פשוט לא יצליח לשפר את איכות התמונות שהוא מייצר. במילים פשוטות, D "פשוט הרבה יותר מדי טוב יחסית ל-G". אתגר זה בא לידי ביטוי גם בצורה של פונקציית המחיר, שלא מאפשרת "העברה יעילה של ידע" מה-discriminator ל-generator.

יש מספר לא קטן של שיטות הבאות לשפר את תהליך האימון של GAN, אך אף אחת מהן אינה מטפלת בבעיה זו באמצעות שינוי של פונקציית המחיר. השיטות הבולטות הן:

- התאמת פיצ'רים (feature matching).
- minibatch discrimination.
- virtual batch normalization.
- מיצוע היסטורי.

כפי שהוסבר, הבעיה של המרחק בין היריעות משתקפת במבנה של פונקציית המחיר, וכיוון שכך, ניתן לנסות ולפתור את הבעיה מהשורש על ידי שימוש בפונקציית מחיר יותר מתאימה. לשם כך ראשית נסמן את התפלגות הדאטה האמיתי ב- p_r , ואת התפלגות הדאטה הסינתטי המיוצר על ידי ה-generator ב- p_g . לעיל הראינו שפונקציית המחיר האופטימלית הממזער את המרחק בין ההתפלגויות p_r, p_g , מתוארת על ידי Jensen-Shannon divergence – \mathcal{D}_{JS} .

ניתן להוכיח כי מרחק \mathcal{D}_{JS} בין ההתפלגויות p_r, p_g לא רגיש לשינויים ב- p_g כאשר המשטחים שבהם "חיים" p_r ו- p_g רחוקים אחד מהשני. כלומר, מרחק \mathcal{D}_{JS} כמעט ולא ישתנה אחרי עדכון המשקלים של ה-generator, וממילא לא ישקף את המרחק המעודכן בין שתי ההתפלגויות p_r ו- p_g . זו למעשה הבעיה המהותית ביותר עם פונקציית המחיר המקורית של ה-generator, שעדכון המשקלים לא משפיע כמעט על \mathcal{D}_{JS} , כיוון שמראש ההתפלגויות p_r ו- p_g רחוקות אחת מהשנייה.

Wasserstein GAN בא להתמודד עם בעיה זו, ולשם כך הוא משתמש בפונקציית מחיר אחרת, בה עדכון המשקלים של ה-generator משתקף גם במרחק בין ההתפלגויות p_r ו- p_g . פונקציית המחיר החדשה מבוססת על מרחק הנקרא Earth Mover (EM), המהווה מקרה פרטי של מרחק וטרשטיין המסומן ב- D_W . מרחק וטרשטיין מסדר $p \geq 1$ בין שתי מידות הסתברות μ, ν על מרחב M מוגדר באופן הבא:

$$W_p(\mu, \nu) = \inf_{\gamma \in \Gamma(\mu, \nu)} \mathbb{E}_{(x,y) \sim \gamma} [\|x - y\|^p] = \left(\inf_{\gamma \in \Gamma(\mu, \nu)} \int_{M \times M} d(x, y)^p d\gamma(x, y) \right)^{\frac{1}{p}}$$

כאשר $\Gamma(\mu, \nu)$ הן כל מידות הסתברות על מרחב המכפלה (product space) של M עם עצמו (זהו למעשה מרחב המכיל את כל הזוגות האפשריים של האלמנטים של M) עם פונקציות שוליות (marginal) השוות ל- μ, ν בהתאמה. תחת סימן האינטגרל יש את המרחק האוקלידי מסדר p בין הנקודות. מרחק EM הינו מקרה פרטי של מרחק וטרשטיין, כאשר $p = 1$, ובאופן מפורש:

$$EM = W_1(\mu, \nu) = \inf_{\gamma \in \Gamma(\mu, \nu)} \int_{M \times M} d(x, y) d\gamma(x, y)$$

הגדרה זו נראית מאוד מסובכת ונסה לתת עבודה אינטואיציה, ולהבין מדוע עבור $p = 1$, מרחק וטרשטיין נקרא EM. לשם הפשטות נניח שהמרחב M הינו חד ממדי, כלומר קו ישר, ועליו עשר משקולות של 0.1_{kg} כל אחת המפוזרות באופן הבא: 6 משקולות (0.6_{kg}) בנקודה $x = 0$, ו-4 משקולות (0.4_{kg}) בנקודה $x = 1$. כעת נרצה להזיז את המשקולות כך שתהיינה מפוזרות באופן הבא: בנקודה $x = 4$ יהיה משקל של 0.3_{kg} , בנקודה $x = 5$ יהיה משקל של 0.5_{kg} ושאר המשקולות (0.2_{kg}) יהיו בנקודה $x = 8$.

כמובן שיש הרבה דרכים לבצע את הזזת המשקולות, ונרצה למצוא את הדרך היעילה ביותר. לשם כך נגדיר מאמץ כמכפלה של משקל במרחק אותו מזיזים את המשקל (בפיזיקה מושג זה נקרא עבודה - כוח המופעל על גוף לאורך מסלול). בדוגמה המובאת, המאמץ המינימלי מתקבל על ידי הזזת המשקולות באופן הבא:

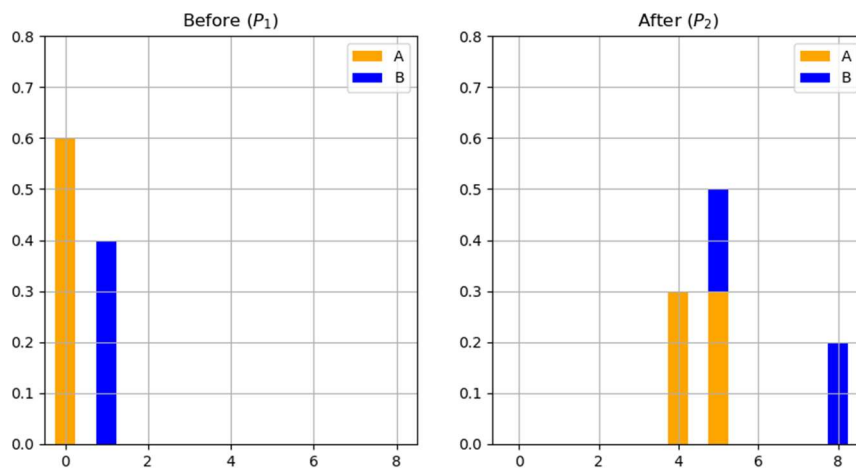
$$0.3_{kg} \text{ מועברים מ-} x = 0 \text{ ל-} x = 4, \text{ כאשר המאמץ הנדרש לכך הינו } 1.2 = (4 - 0) \cdot 0.3$$

$$0.3_{kg} \text{ מועברים מ-} x = 0 \text{ ל-} x = 5, \text{ כאשר המאמץ הנדרש לכך הינו } 1.5 = (5 - 0) \cdot 0.3$$

$$0.2_{kg} \text{ מועברים מ-} x = 1 \text{ ל-} x = 5, \text{ כאשר המאמץ הנדרש לכך הינו } 0.8 = (5 - 1) \cdot 0.2$$

$$0.2_{kg} \text{ מועברים מ-} x = 1 \text{ ל-} x = 8, \text{ כאשר המאמץ הנדרש לכך הינו } 1.4 = (8 - 1) \cdot 0.2$$

$$\text{סך המאמץ המינימלי שווה במקרה הזה ל: } 1.2 + 1.5 + 0.8 + 1.4 = 4.9$$



איור 7.14 העברת משקולות באופן אופטימלי. P_1 מייצג את המצב ההתחלתי, ו- P_2 הינו המצב לאחר הזזת המשקולות.

כעת, במקום להסתכל על משקלים, נתייחס להתפלגויות p_1, p_2 , המוגדרות באופן הבא:

$$p_1(x) = \begin{cases} 0.6, & x = 0 \\ 0.4, & x = 1 \\ 0, & \text{else} \end{cases}, p_2(x) = \begin{cases} 0.3, & x = 4 \\ 0.5, & x = 5 \\ 0.2, & x = 8 \\ 0, & \text{else} \end{cases}$$

השאלה כיצד ניתן להעביר מסה הסתברותית מ- p_1 כך שתתקבל ההתפלגות p_2 , שקולה לדוגמה של הזזת המשקולות. מרחק EM בין שתי התפלגויות p_1, p_2 מוגדר להיות "מאמץ" המינימלי הנדרש בשביל להעביר את המסה ההסתברותית מ- p_1 ל- p_2 , או במילים אחרות – מרחק EM מגדיר מהי כמות ה"עבודה" (מאמץ) המינימלית הנדרשת בשביל להפוך p_1 ל- p_2 . אם נחזור לדוגמה של המשקולות, נוכל להבין מדוע D_w עבור $p = 1$ נקרא מרחק Earth Mover – מרחק בין שתי התפלגויות שקול לכמה מאמץ נדרש להעביר כמות אדמה במשקל מסוים כדי לעבור מחלוקה מסוימת של אדמה לחלוקה אחרת. באופן יותר פורמלי – מידת ההסתברות על מרחב המכפלה בנוסחה של מרחק EM מתארת את האופן שבו אנחנו מעבירים את המסה ההסתברותית (משקל מסוים של אדמה), כאשר הביטוי $\gamma(x, y)$ מתאר כמה מסה הסתברותית מועברת מנקודה x לנקודה y .

לאחר שהוסבר מהו מרחק וסרשטיין D_w ומהו מרחק EM, ניתן להבין כיצד אפשר להשתמש במושגים אלו עבור פונקציית מחיר של GAN. נציין כי מרחק D_w בין מידות ההסתברות מתחשב בתכונות של הקבוצות עליהן מידות אלו מוגדרות בצורה מפורשת, על ידי התחשבות במרחקים בין בקבוצות אלו. תכונה זו היא למעשה בדיוק מה שצריך בשביל למדוד את המרחק בין ההתפלגות האמיתית של דאטה p_r לבין התפלגות של הדאטה הסינתטי p_g . מרחק EM ידע לשערך בצורה טובה את המרחק בין היריעות שבהן "חיות" שתי ההתפלגויות, כלומר אם מזיזים את היריעה של הדאטה הסינתטי, נוכל לדעת בעזרת מרחק EM עד כמה השתנה המרחק בין היריעות. נציין שזה לא קורה כאשר משתמשים בפונקציית המחיר המקורית הנמדדת באמצעות D_{JS} . כעת, בעזרת פונקציית המחיר החדשה המבוססת על מרחק EM, ניתן לדעת עד כמה עדכון המשקלים מקרב או מרחיק את p_g מ- p_r .

באופן תיאורטי זה מצוין, אך עדיין זה לא מספיק, כיוון שצריך למצוא דרך לחשב את D_w , או לכל הפחות את המקרה הפרטי שלו עבור $p = 1$, כלומר את מרחק EM. במקור מרחק זה מוגדר כבעיית אופטימיזציה של מידות הסתברות על מרחב המכפלה, וצריך למצוא דרך להשתמש בו כפונקציית מחיר. בשביל לבצע זאת, ניתן להשתמש בצורה דואלית של D_w עבור $p = 1$ – שיוויין RK (Rubinstein-Kantorovich), לפיו ניתן לחשב את $D_w, p = 1$ באופן הבא:

$$W(p_r, p_g) = \frac{1}{K} \sup_{\|f\|_L} E_{x \sim p_r}[f(x)] - E_{x \sim p_g}[f(x)]$$

כאשר $f(x)$ הינה פונקציית K-ליפשיץ רציפה (כלומר, פונקציה רציפה עם קצב השתנות החסום על ידי K). כעת נניח ש- $f(w)$ הינה פונקציית K-ליפשיץ רציפה המתארת discriminator בעל סט הפרמטרים w . ה-discriminator מחשב באופן מקורב את המרחק בין ההתפלגויות באופן הבא:

$$L(p(r), p(g)) = W(p(r), p(g)) = \max_{w \in W} E_{x \sim p_r}[f_w(x)] - E_{z \sim p_r(z)}[f_w(g_\theta(z))]$$

פונקציית מחיר זו מודדת את המרחק D_w בין ההתפלגויות p_r, p_g , וככל שפונקציה זו תקבל ערכים יותר נמוכים ככה ה-generator יצליח לייצר דוגמאות שמתפלגות באופן יותר דומה לדאטה המקורי. בשונה מ-GAN קלאסי בו ה-discriminator מוציא הסתברות עד כמה הדוגמא אותה הוא מקבל אמיתית, פה ה-discriminator לא מאומן להבחין בין דוגמא אמיתית לסינתטית, אלא מאומן ללמוד פונקציית K-ליפשיץ רציפה המודדת את D_w בין ההתפלגויות p_r, p_g . ה-generator לעומת זאת מאומן למזער את $L(p_r, p_g)$ (כאשר רק האיבר השני שתלוי ב- θ), וככל שפונקציית המחיר הולכת וקטנה, כך p_g מתקרב יותר ל- p_r .

כאמור, תנאי הכרחי לשימוש במרחק זה בפונקציית המחיר הינו שהפונקציה תהיה K-ליפשיץ רציפה. מסתבר שקיום תנאי זה אינו משימה קלה כלל. כדי להבטיח את קיומו, המאמר המקורי הציע לבצע קטימה של משקלי ה-discriminator לטווח סופי מסוים, נניח $[-0.01, 0.01]$. ניתן להראות כי קטימה זו מבטיחה את ש- f_w תהיה K-ליפשיץ רציפה. אולם, כמו שכותבי המאמר מודים בעצמם, ביצוע קטימה בכדי לדאוג לקיום תנאי ליפשיץ יכול לגרום לבעיות אחרות. למעשה, כאשר חלון הקטימה של המשקלים צר מדי, הגרדיאנטים של Wasserstein GAN עלולים להתאפס, מה שיאט את תהליך הלמידה. מצד שני, כאשר חלון זה רחב מדי, ההתכנסות עלולה להיות מאוד איטית. נציין שיש עוד מספר דרכים לכפות על f_w להיות ליפשיץ-רציפה למשל gradient penalty.

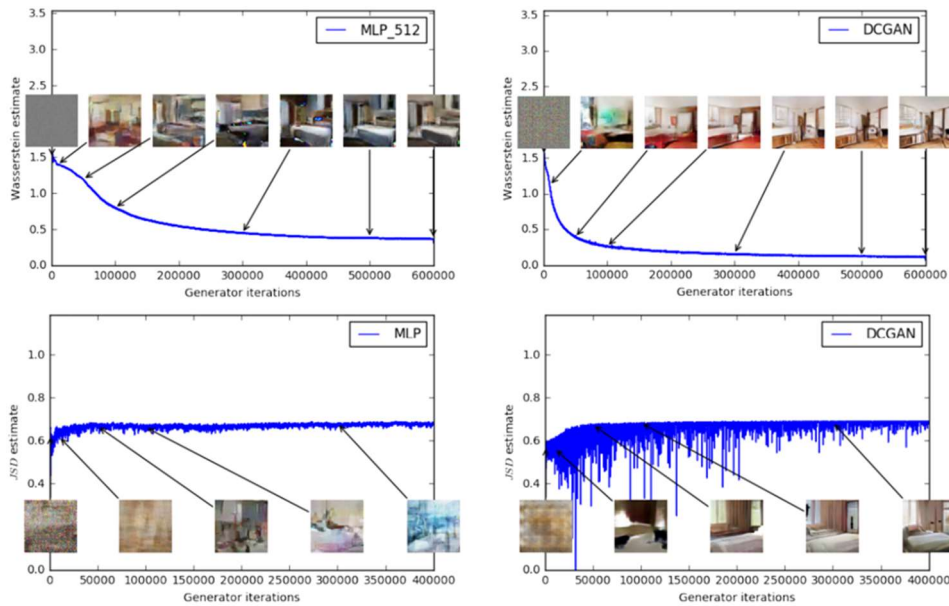
האימון של Wasserstein GAN דומה לאימון של ה-GAN המקורי, למעט שני הבדלים עיקריים:

א. קיצוץ טווח המשקלים על מנת לשמור על רציפות-ליפשיץ.

ב. פונקציית מחיר המסתמכת על D_w במקום על D_{JS} .

תהליך הלמידה מתבצע באופן הבא – לאחר כל עדכון משקלים של ה-discriminator (באמצעות gradient ascent), מקצצים את טווח המשקלים. לאחר מכן מבצעים עדכון רגיל של משקלי ה-generator תוך ביצוע של איטרציה של gradient descent.

Wasserstein GAN מצליח לגרום לכך שהקורלציה בין איכות התמונה הנוצרת על ידי ה-generator לבין ערך של פונקציית לוס תהיה הרבה יותר בולטת מאשר ב-GAN רגיל בעל אותה ארכיטקטורה. ניתן להמחיש זאת היטב באמצעות גרפים הבוחנים את היחס בין D_w לבין D_{JS} :



איור 7.15 שערור מרחק D_W בין p_g -ל- p_r (כפונקציה של מספר האיטרציות) של מספר האיטרציות (בגרפים העליונים), לעומת שערור מרחק D_{JS} בין p_g -ל- p_r כפונקציה של מספר האיטרציות (בגרפים התחתונים).

ניתן לראות בבירור כי ככל שאיכות התמונות שה-generator מייצר עולה, כך D_W הולך וקטן, ואילו מרחק D_{JS} לא מראה שום סימן של ירידה. הצלחה זו נובעת מהשינוי בפונקציית המחיר, שגרם לאימון להיות יותר יעיל, והביא לכך שהדוגמאות הסינתטיות תהיינה דומות הרבה יותר לדאטה המקורי.

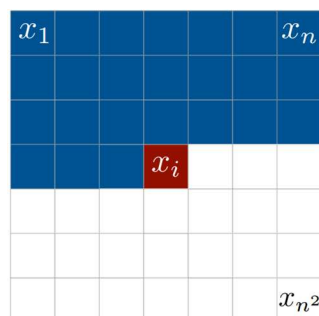
נקודה נוספת שיכולה להסביר את ההצלחה היחסית של השימוש ב- D_W נובעת מכך שמטריקת וסרשטיין חלשה יחסית למטריקת JS, ונסה להבהיר נקודה זו.

באופן אידיאלי, כאשר אנו מאמנים מודל היינו רוצים להיות בטוחים שאם ננהג בצורה נאותה ובכל צעד נעדכן המודל בדיוק על פי הוראות הגרדיאנט, נסיים את האימון בנקודה כמעט אופטימלית. אולם בפועל זה לא תמיד כך, כיוון שישנן בעיות שעבורן מטריקות מסוימות יגיעו לנקודה זו ואחרות לא. ניקח לדוגמא שני אנשים שעומדים על סף תהום ורוצים להגיע לעמק. האחד מודד את הגובה ומתקדם על פיו, ולכן הוא יגיע למטה בקלות יחסית. האחר מתעניין במיקומו על ציר צפון דרום, ולכן הוא עשוי להיתקל בקשיים במהלך הירידה, וגם אם הוא אכן יגיע למטה, זה בהכרח יהיה בתהליך איטי יותר. באופן דומה, כאשר לוקחים זוג מטריקות, באופן פורמלי ניתן להגדיר שאם התכנסות של סדרת התפלגויות תחת מטריקה אחת גוררת התכנסות של הסדרה תחת מטריקה אחרת, אזי המטריקה הראשונה חזקה יותר מהמטריקה השנייה. העובדה ש- D_W חלש יותר מ- D_{JS} בעצם אומרת שיתכן ויש בעיות שעבורן תתקבל תוצאה אופטימלית עבור D_W אך לא עבור D_{JS} .

7.3 Auto-Regressive Generative Models

משפחה נוספת של מודלים גנרטיביים נקראת Auto-Regressive Generative Models, ובדומה ל-VAE גם מודלים אלו מוצאים התפלגות מפורשת של מרחב מסוים ובעזרת התפלגות זו מייצרים דאטה חדש. עם זאת, בעוד VAE מוצא קירוב להתפלגות של המרחב הלטנטי, שיטות AR מנסות לחשב במדויק התפלגות מסוימת, וממנה לדגום ולייצר דאטה חדש.

תמונה x בגודל $n \times n$ היא למעשה רצף של n^2 פיקסלים. כאשר רוצים ליצור תמונה, ניתן ליצור כל פעם כל פיקסל באופן כזה שהוא יהיה תלוי בכל הפיקסלים שלפניו.



איור 7.15 תמונה כרצף של פיקסלים.

כל פיקסל הוא בעל התפלגות מותנית:

$$p(x_i | x_1 \dots x_{i-1})$$

כאשר כל פיקסל מורכב משלושה צבעים (RGB), לכן ההסתברות המדויקת היא:

$$p(x_{i,R} | x_{<i}) p(x_{i,G} | x_{<i}, x_{i,R}) p(x_{i,B} | x_{<i}, x_{i,R}, x_{i,G})$$

כל התמונה השלמה היא מכפלת ההסתברויות המותנות:

$$p(x) = \prod_{i=1}^{n^2} p(x_i) = \prod_{i=1}^{n^2} p(x_i | x_1 \dots x_{i-1})$$

הביטוי $p(x)$ הוא ההסתברות של דאטה מסוים לייצג תמונה אמיתית, לכן נרצה למקסם את הביטוי הזה כדי לקבל מודל שמייצג תמונות שנראות אותנטיות עד כמה שניתן.

7.3.1 PixelRNN

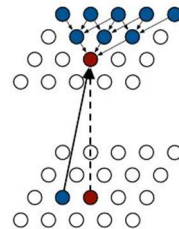
אפשרות אחת לחשב את $p(x)$ היא להשתמש ברכיבי זיכרון כמו LSTM עבור כל פיקסל. באופן טבעי היינו רוצים לקשר כל פיקסל לשכנים שלו:

$$\text{Hidden State } (i, j) = f(\text{Hidden State } (i - 1, j), \text{Hidden State } (i, j - 1))$$

הבעיה בחישוב זה היא הזמן שלוקח לבצע אותו. כיוון שכל פיקסל דורש לדעת את הפיקסל שלפניו – לא ניתן לבצע אימון מקבילי לרכיבי ה-LSTM. כדי להתגבר על בעיה זו הוצעו כמה שיטות שנועדו לאפשר חישוב מקבילי.

Row LSTM

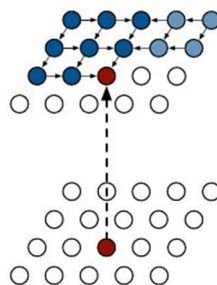
במקום להשתמש במצב החבוי של הפיקסל הקודם, ניתן להשתמש רק בשורה שמעל הפיקסל אותו רוצים לחשב. שורה זו בעצמה מחושבת לפני כן על ידי השורה שמעליה, ובכך למעשה לכל פיקסל יש receptive field של משולש. בשיטה זו ניתן לחשב באופן מקבילי כל שורה בנפרד, אך יש לכך מחיר של איבוד הקשר בין פיקסלים באותה שורה (loss context).



איור 7.16 Row LSTM – כל פיקסל מחושב על ידי $k \geq 3$ פיקסלים בשורה שמעליו.

Diagonal BiLSTM

כדי לאפשר גם חישוב מקבילי וגם שמירה על קשר עם כל הפיקסלים, ניתן להשתמש ברכיבי זיכרון דו כיווניים. בכל שלב מחשבים את רכיבי הזיכרון משני הצדדים של כל שורה, וכך כל פיקסל מחושב גם בעזרת הפיקסל שלידו וגם על ידי זה שמעליו. באופן הזה ה-receptive field גדול יותר ואין loss context, אך החישוב יותר איטי מהשיטה הקודמת, כיוון שהשורות לא מחושבות בפעם אחת אלא כל פעם שני פיקסלים.

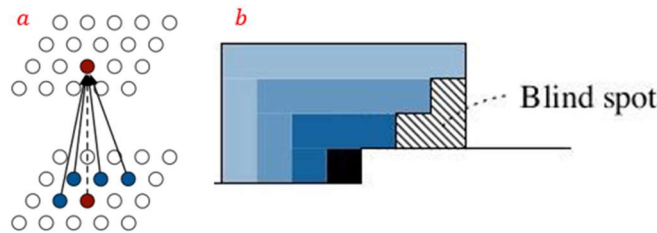


איור 7.17 Diagonal BLSTM – כל פיקסל מחושב על ידי $k \geq 3$ פיקסלים בשורה שמעליו.

כדי לשפר את השיטות שמשמשות ברכיבי זיכרון ניתן להוסיף עוד שכבות, כמו למשל Residual blocks שעוזרים להאיץ את ההתכנסות ו-Masked convolutions כדי להפריד את התלות של הערוצים השונים של כל פיקסל.

7.3.2 PixelCNN

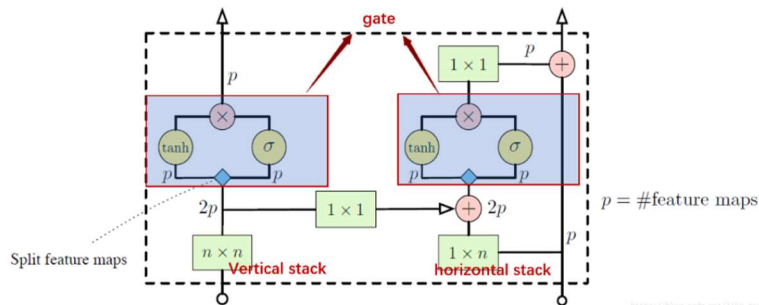
החיסרון העיקרי של PixelRNN נובע מהאימון האיטי שלו. במקום רכיבי זיכרון ניתן להשתמש ברשת קונבולוציה, ובכך להאיץ את תהליך הלמידה ולהגדיל את ה-receptive field. גם בשיטה זו מתחילים מהפיקסל הפינתי, רק כעת הלמידה היא לא בעזרת רכיבי זיכרון אלא באמצעות שכבות קונבולוציה. היתרון של שיטה זו על פני PixelRNN מתבטא בקיצור משמעותי של תהליך האימון, אך התוצאות פחות טובות. חיסרון נוסף בשיטה זו נובע מהמבנה של המסננים ה-receptive field – כל פיקסל מתבסס על שלושה פיקסלים שמעליו, והם בתורם כל אחד תלוי בשלושה פיקסלים בשורה שמעל. מבנה זה מנתק את התלות בין פיקסלים קרובים יחסית אך אינם ב-receptive field.



איור 7.18 (a) receptive field של PixelCNN. (b) החיסרון של PixelCNN – ניתוק בין פיקסלים יחסית קרובים.

7.3.3 Gated PixelCNN

כדי להתגבר על בעיות אלו – ביצועים לא מספיק טובים והתעלמות מפיקסלים יחסית קרובים שאינם ב-receptive field – נעשה שימוש ברכיבי זיכרון הדומה ל-LSTM, המשלב את רשתות הקונבולוציה בתוך RNN.



איור 7.19 שכבה של Gated PixelCNN.

כל רכיב זיכרון בנוי משני חלקים – horizontal stack and vertical stack, כאשר כל אחד מהם הוא למעשה שכבת קונבולוציה. ה-vertical stack בנוי מזיכרון של כל השורות שהיו עד כה בתמונה, וה-horizontal stack יחיד על הקלט הנוכחי. ה-horizontal stack עובר דרך שער של אקטיבציות לא ליניאריות ובנוסף מתחבר ל-vertical stack, כאשר גם החיבור ביניהם עובר דרך שער של אקטיבציות לא ליניאריות. לפני כל כניסה של stack לתוך שער, המסננים מתפצלים – חצי עוברים דרך tanh וחצי דרך סיגמואיד. בסך הכל המוצא של כל שער הינו:

$$y = \tanh(w_f * x) \odot \sigma(w_g * x)$$

7.3.4 PixelCnn++

שיפור אחר של PixelCNN הוצע על ידי OpenAI, והוא מבוסס על מספר מודיפיקציות:

- שכבת ה-SoftMax שקובעת את צבע הפיקסל צורכת הרבה זיכרון, כיוון שיש הרבה צבעים אפשריים. בנוסף, היא גורמת לגרדיאנט להתאפס מהר. כדי להתגבר על כך ניתן לבצע דיסקרטיזציה לצבעים, ולאפשר טווח צבעים קטן יותר. באופן הזה קל יותר לקבוע את ערכו של כל פיקסל, ובנוסף תהליך האימון יותר יעיל.
- במקום לבצע בכל פיקסל את ההתניה על כל צבע בפנפרד (כפי שהראינו בפתיחה), ניתן לבצע את ההתניה על כל הצבעים יחד.
- אחד האתגרים של PixelCNN הוא היכולת המוגבלת למצוא תלויות בין פיקסלים רחוקים. כדי להתגבר על כך ניתן לבצע down sampling, ובכך להפחית את מספר הפיקסלים בכל מסנן, מה שמאפשר לשמור את הקשרים בין פיקסלים בשורות רחוקות.

- בדומה ל-U-Net, ניתן לבצע חיבורים בעזרת Residual blocks ולשמור על יציבות במהלך הלמידה.
- שימוש ב-Dropout לצורך רגולריזציה והימנעות מ-fitting.

7. References

VAE:

<https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73>

<https://jaan.io/what-is-variational-autoencoder-vae-tutorial/>

<https://lilianweng.github.io/lil-log/2018/08/12/from-autoencoder-to-beta-vae.html>

GANs:

<https://arxiv.org/abs/1406.2661>

<https://arxiv.org/pdf/1511.06434.pdf>

<https://phillipi.github.io/pix2pix/>

<https://junyanz.github.io/CycleGAN/>

<https://arxiv.org/abs/1710.10196>

<https://arxiv.org/abs/1812.04948>

<https://towardsdatascience.com/explained-a-style-based-generator-architecture-for-gans-generating-and-tuning-realistic-6cb2be0f431>

<https://arxiv.org/abs/1701.07875>

AR models:

<https://arxiv.org/abs/1601.06759>

<https://arxiv.org/abs/1606.05328>

<https://arxiv.org/pdf/1701.05517.pdf>

<https://towardsdatascience.com/auto-regressive-generative-models-pixelrnn-pixelcnn-32d192911173>

https://wiki.math.uwaterloo.ca/statwiki/index.php?title=STAT946F17/Conditional_Image_Generation_with_PixelCNN_Decoders#Gated_PixelCNN

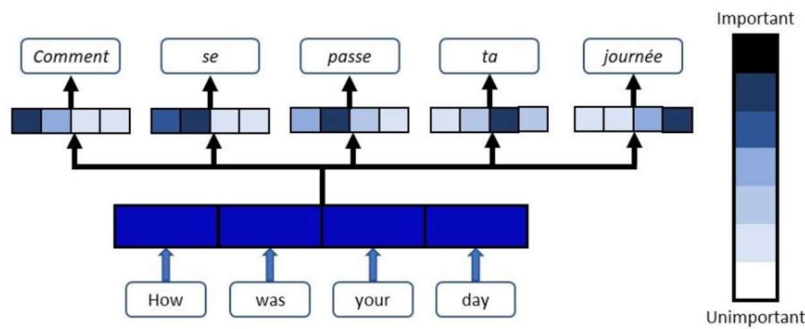
8. Attention Mechanism

8.1 Sequence to Sequence Learning and Attention

8.1.1 Attention in Seq2Seq Models

ניתוח סדרות בהן יש קשר בין האיברים יכול להיעשות בעזרת רשתות עם רכיבי זיכרון, כפי שהוסבר באריכות בפרק 6. ברשתות אלו הסדרה הנכנסת לרשת עוברת דרך encoder היוצר וקטור בגודל ידוע מראש המייצג את הסדרה המקורית, תוך התחשבות בסדר של איברי הסדרה ובקשר ביניהם. לאחר מכן וקטור זה עובר ב-decoder שיכול לפענח את המידע שיש בווקטור ולהציג אותו בצורה אחרת. למשל בתרגום משפה לשפה – מודל של seq2seq מקודד משפט בשפה אחת לווקטור מסוים ולאחר מכן מפענח את הווקטור למשפט בשפה השנייה.

הדרך המקובלת ליצור את הווקטור ולפענח אותו הייתה שימוש בארכיטקטורות שונות של RNN, כמו למשל רשת עמוקה מסוג LSTM או GRU המכילה רכיבי זיכרון. מודלים אלו נתקלו בבעיה בסדרות ארוכות, כיוון שהווקטור מוגבל ביכולת שלו להכיל קשרים בין מספר רב של איברים. כדי להתמודד עם בעיה זו ניתן לנקוט בגישה שונה – במקום ליצור וקטור במוצא ה-encoder, ניתן להשתמש במצבים החבויים של ה-encoder בשילוב המצבים החבויים של ה-decoder, וכך למצוא תלויות בין איברי סדרת הקלט לאיברי סדרת הפלט (general attention) וקשרים בין איברי סדרת הקלט עצמם (self-attention). ניקח לדוגמה תרגום של המשפט "How was your day" מאנגלית לשפה אחרת – במקרה זה מנגנון ה-attention מייצר וקטור חדש עבור כל מילה בסדרת הפלט, כאשר כל רכיב בווקטור מכמת עד כמה המילה הנוכחית במוצא קשורה לכל אחת מהמילים במשפט המקורי. באופן הזה כל איבר בסדרת הפלט ממשקל כל אחד מאיברי סדרת הקלט. מנגנון זה נקרא attention.



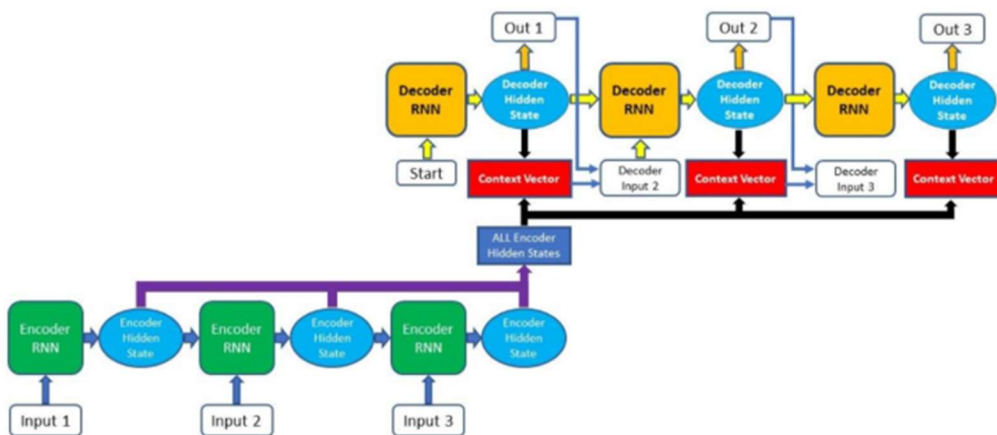
איור 8.1 מנגנון attention – נתינת משקל לכל אחת ממילות הקלט ביחס לכל אחת ממילות הפלט.

במאמר משנת 2017 שנקרא "Attention is All You Need", הוצע להשתמש ב-attention בלבד ללא רשתות מסוג LSTM או GRU, ומאמר זה פרץ דרך לשימושים רבים במנגנון זה תוך קבלת ביצועים מעולים.

בחלק זה יוצגו הגישות המשלבות בין רשתות RNN לבין attention ולאחר מכן יוסבר על ה-transformer שמשתמש ב-self-attention וב-positional encoding בנוסף ל-attention הרגיל, ובכך מייתר את הצורך ברכיבי זיכרון.

8.1.2 Bahdanau Attention and Luong Attention

הגישה הראשונה שהוצעה נקראת Bahdanau Attention על שם הממציא שלה – Dzmitry Bahdanau.



איור 8.2 ארכיטקטורת Bahdanau Attention.

הרעיון של גישה זו היא לבנות ארכיטקטורה בה משתמשים בכל המצבים החבויים של רכיבי הזיכרון ב-encoder ומעבירים אותם ל-decoder. כתוצאה מכך ה-decoder מחשב את המוצא לא רק על סמך מצביו הקודמים, אלא משקלל אותם יחד עם המצבים החבויים של ה-encoder. עבור כל אחד מאיברי סדרת הפלט מחשבים alignment score בין המצב החבוי של רכיבי הזיכרון הקודם בסדרת הפלט לבין כל המצבים החבויים של ה-encoder, וכך יוצרים context vector שבעזרתו מחשבים את הפלט עבור האיבר הנוכחי ב-decoder. ביצוע הפעולה הזו הוא הלב של מנגנון ה-attention, כיוון שהוא קושר בין הקלט לפלט, ובנוסף מחשב עבור כל איבר של סדרת קלט כמה משקל יש לתת לכל אחד מאיברי הקלט האחרים.

ביצוע פעולה זו יוצרת לכל אחד מאיברי הפלט context vector ייחודי משלו הנבנה גם מהמצב הקודם וגם מאיברי ה-encoder, בשונה מהארכיטקטורות הקודמות של seq2seq בהן לא היה ניתן להעביר מידע באופן ישיר מהמצבים החבויים של ה-encoder ל-decoder. את ה-context vector מחברים למוצא של האיבר הקודם ב-decoder, ויחד עם המצב החבוי הקודם יוצרים את המצב החבוי הבא, שבעזרתו מוצאים את הפלט של האיבר הנוכחי.

באופן פורמלי, אם נסמן ב- H_e, H_d את המצבים החבויים של ה-encoder וה-decoder, ה-alignment score יתקבל על ידי:

$$\text{alignment score} = w_{\text{alignment}} \times \tanh(w_d H_d + w_e H_e)$$

כאשר $w_{\text{alignment}}, w_d, w_e$ הם המשקלים הנלמדים של ה-encoder, ה-decoder והחיבור ביניהם. את התוצאה מעבירים דרך SoftMax, מכפילים ב- H_e ומקבלים את ה-context vector:

$$\text{context vector} = H_e \times \text{SoftMax}(\text{alignment score})$$

הווקטור המתקבל מכיל משקל של כל אחד מאיברי הקלט ביחס לאיבר הפלט הנוכחי. את התוצאה כאמור מחברים לפלט של האיבר הקודם, ובעזרת המצב החבוי הקודם מחשבים את המצב החבוי הנוכחי, שממנו מחלצים את הפלט של האיבר הנוכחי.

ישנו שיפור של Bahdanau attention הנקרא Loung attention. שני הבדלים עיקריים יש בין שני המנגנונים: חישוב ה-alignment score מתבצע באופן שונה, ובנוסף בכל שלב לא משתמשים במצב החבוי הקודם של ה-decoder כמו שהוא אלא יוצרים מצב חבוי חדש ובעזרתו מחשבים את ה-alignment score.

8.2 Transformer

לאחר שמנגנון ה-attention התחיל לצבור תאוצה, הומצאה ארכיטקטורה המבוססת על attention בלבד ללא שום רכיבי זיכרון. ארכיטקטורה זו הנקראת transformer מציעה שני אלמנטים חדשים על מנת למצוא קשרים בין איברים בסדרה מסוימת – positional encoding ו-self-attention.

8.2.1 Positional Encoding

ארכיטקטורות מבוססות RNN משתמשות ברכיבי זיכרון בשביל לקחת בחשבון את הסדר של האיברים בסדרה. גישה אחרת לייצוג הסדר בין איברי הסדרה נקראת positional encoding, בה מוסיפים לכל אחד מאיברי הקלט פיסת מידע לגבי המיקום שלה בסדרה, והוספה זו כאמור באה כתחליף לרכיבי הזיכרון ברשתות RNN. באופן פורמלי, עבור סדרת קלט $x \in \mathbb{R}^d$, מחשבים וקטור בממד $d \times 1$ באופן הבא:

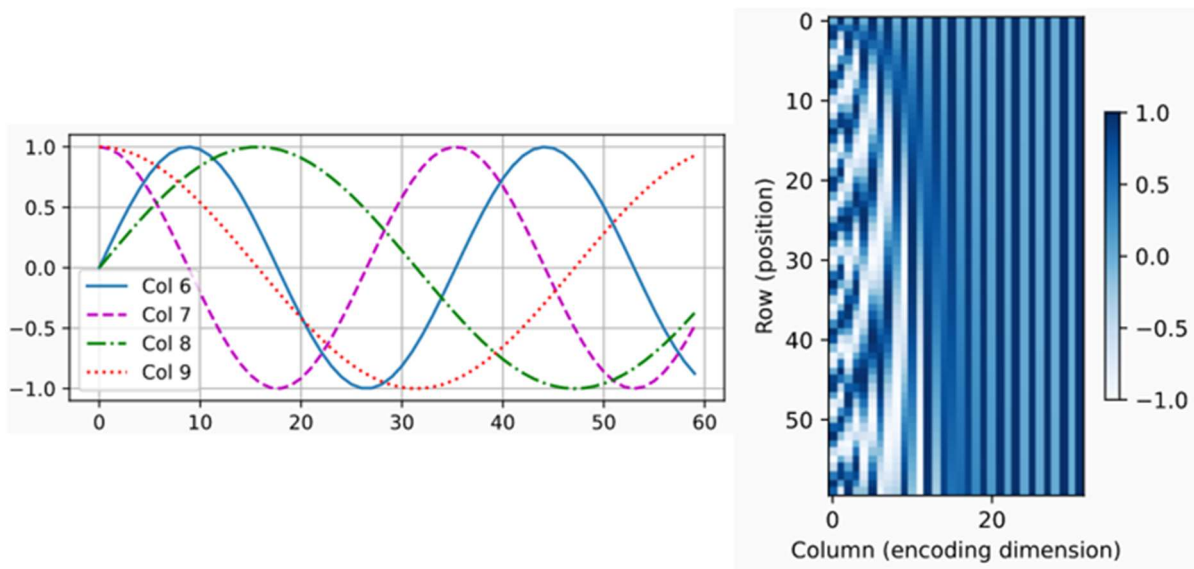
$$p_t(i) = \begin{cases} \sin(\omega_k t), & i \text{ is even} \\ \cos(\omega_k t), & i \text{ is odd} \end{cases}, \omega_k = \frac{1}{10000^{\frac{2k}{d}}} \rightarrow p_t = \begin{bmatrix} \sin(\omega_1 t) \\ \cos(\omega_1 t) \\ \sin(\omega_2 t) \\ \cos(\omega_2 t) \\ \vdots \\ \sin\left(\frac{\omega_d t}{2}\right) \\ \cos\left(\frac{\omega_d t}{2}\right) \end{bmatrix}_{d \times 1}$$

בכדי להבין כיצד וקטור זה מכיל משמעות של סדר בין דברים, נציג את הרעיון שהוא מייצג בצורה יותר פשוטה. אם נרצה לקחת רצף של מספרים ולייצג אותם בצורה בינארית, נוכל לראות שככל שלביט יש משקל גדול יותר, כך הוא משתנה בתדירות נמוכה יותר, ולמעשה תדירות שינוי הביט היא אינדיקציה למיקום שלו.

0:	0 0 0 0	8:	1 0 0 0
1:	0 0 0 1	9:	1 0 0 1
2:	0 0 1 0	10:	1 0 1 0
3:	0 0 1 1	11:	1 0 1 1
4:	0 1 0 0	12:	1 1 0 0
5:	0 1 0 1	13:	1 1 0 1
6:	0 1 1 0	14:	1 1 1 0
7:	0 1 1 1	15:	1 1 1 1

איור 8.3 ייצוג בינארי של מספרים. ה-MSB משתנה בתדירות הכי נמוכה, ואילו ה-LSB משתנה בתדירות הכי גבוהה.

כיוון שמתעסקים במספרים שאינם בהכרח שלמים, הייצוג הבינארי של מספרים שלמים הוא יחסית בזבזני, ולכן כדאי לקחת גרסה רציפה של אותו רעיון – פונקציות טריגונומטריות עם תדירות הולכת וגדלה. זהו בעצם הווקטור p – הוא מכיל הרבה פונקציות טריגונומטריות בעלות תדירות הולכת וקטנה, ולפי התדירות שמתווספת לכל איבר בסדרה המקורית ניתן לקבל אינדיקציה על מיקומו.



איור 8.4 Positional encoding. דוגמא למספר פונקציות בעלות תדירות הולכת וקטנה, בהתאם לאיבר אותן הן מייצגות (שמאל). המחשה לקצב השינוי של כל פונקציה בהתאם למיקום של האיבר אותו היא מייצגת – מעין גרסה רציפה לקצב שינוי הביטים בייצוג בינארי של מספרים שלמים (ימין).

ישנו יתרון נוסף שיש לשימוש בפונקציות הטריגונומטריות – עבור כל צמד פונקציות בעלות אותו תדר $\begin{bmatrix} \sin(\omega_k t) \\ \cos(\omega_k t) \end{bmatrix}$ ניתן לבצע טרנספורמציה לינארית ולקבל תדר אחר (Relative Positional Information):

$$M \cdot \begin{bmatrix} \sin(\omega_k t) \\ \cos(\omega_k t) \end{bmatrix} = \begin{bmatrix} \sin(\omega_k t + \phi) \\ \cos(\omega_k t + \phi) \end{bmatrix}, M = \begin{bmatrix} \cos(\omega_t \phi) & \sin(\omega_t \phi) \\ -\sin(\omega_t \phi) & \cos(\omega_t \phi) \end{bmatrix}$$

באופן הזה מקבלים באופן מידי ייחוס בין כל ה-positions, מה שיכול לעזור בניתוח הקשרים שבין איברים שונים.

8.2.2 Self-Attention Layer

בנוסף ל-positional encoding, עלה הרעיון לבצע attention לא רק בין איברי הקלט לאיברי הפלט, אלא גם בין איברי הקלט בעצמם. הרעיון הוא לייצר ייצוג חדש של סדרת הקלט באותו אורך כמו הסדרה המקורית, כאשר כל איבר בסדרה החדשה ייצג איבר בסדרה המקורית בתוספת מידע על הקשר שלו לשאר האיברים. הרעיון הכללי אומר שיש לקחת כל איבר בסדרה, ולחשב את הדמיון שלו לאר האיברים בסדרה. איברים דומים (קרובים) בסדרה יקבלו ערכי דמיון גבוהים, ואילו איברים שונים (רחוקים) בסדרה יתנו ערכים נמוכים (ב-NLP זה יכול להיות מילים שסביר שיופיעו בסמיכות, ובתמונה זה יכול להיות פיקסלים דומים). דמיון בין איברים נמדד על פי הקשר שיש ביניהם, והוא מחושב באמצעות מכפלה פנימית בין וקטורי ייצוג של האיברים. כל מכפלה פנימית בין שני איברים נותנת מקדם שהוא מספר ממשי, וכך ניתן לסכם את מכפלות כל המקדמים באיברים המקוריים, ולקבל ייצוג חדש לאיבר המקורי

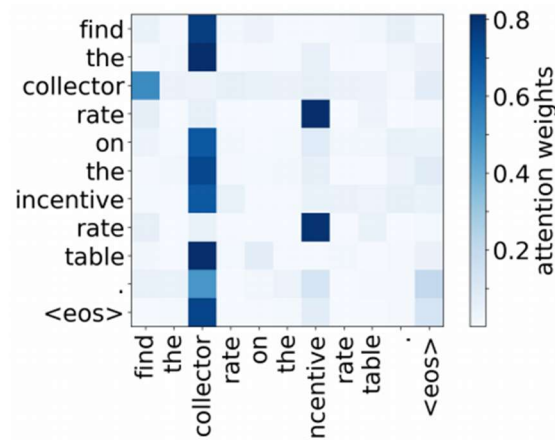
המכיל גם קשר בין האיבר הנוכחי לאיברים דומים בסדרה. במילים אחרות, ניתן להסתכל על וקטור המכיל את הקשרים של איבר מסוים בסדרה כייצוג החדש המשקף את קשריו עם שאר איברי הסדרה.

באופן פורמלי, בשביל לחשב את ה-self-attention יוצרים שלוש מטריצות של מקדמים עבור סדרת הכניסה. המטריצות נקראות **Query**, **Key**, **Value**, כאשר כל אחת מהן נוצרת על ידי הכפלה של מטריצת משקלים באיבר הקלט. בעזרת מטריצות אלו מחשבים את ה-attention score:

$$\text{Attention}(\text{Query}, \text{Key}, \text{Value}) = \text{SoftMax}\left(\frac{Q \cdot K}{\sqrt{d_k}}\right) \cdot V$$

כדי להבין כיצד הנוסחה הזו מסייעת במציאת קשר בין איברים, נבחן כל איבר שלה בנפרד. עבור סדרת קלט x מקבלים שלוש מטריצות, כאשר כל איבר בסדרה המקורית x_i יוצר שורה בכל אחת מהמטריצות. כאשר לוקחים את השורה $q_i = Q \cdot x_i$ ומכפילים אותה בכל אחת מהשורות במטריצה K , מקבלים וקטור חדש, שכל איבר j בוקטור אומר עד כמה יש קשר בין האיברים i, j בסדרה המקורית. ביצוע ההכפלה הזו עבור כל סדרת הקלט יוצר מטריצה חדשה בה כל שורה מייצגת את הקשר בין איבר מסוים לשאר איברי הסדרה. ההכפלה הזו היא בעצם $Q \cdot K$, כאשר כל מכפלה $q_i^T k_j$ מייצגת את הקשר בין האיבר i לאיבר j . את התוצאה מחלקים בשורש של ממד ה-embedding כדי לשמור על יציבות הגרדיאנט, ולאחר מכן מנרמלים על ידי SoftMax . באופן הזה מקבלים מטריצה של מספרים בטווח $[0, 1]$, המייצגים כאמור את הקשר בין כל שני איברים בסדרה המקורית. נסמן כל איבר במטריצה ב- w_{ij} , ונוכל לקבל אותו ישירות על ידי הנוסחה:

$$w_{ij} = \text{SoftMax}\left(\frac{q_i \cdot k_j}{\sqrt{d_k}}\right) = \frac{\exp\left(\frac{q_i^T k_j}{\sqrt{d_k}}\right)}{\sum_{s=1}^n \exp\left(\frac{q_i^T k_s}{\sqrt{d_k}}\right)}$$

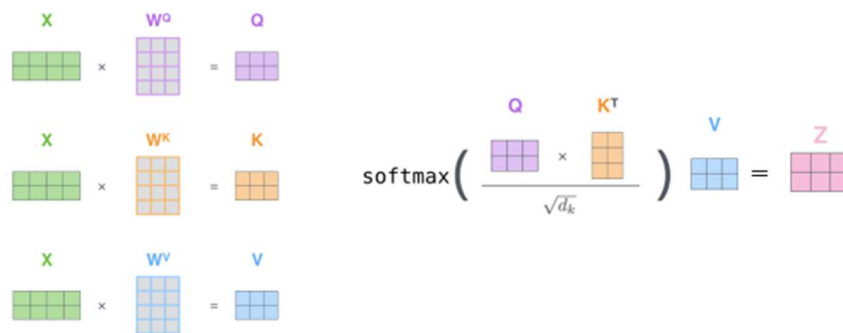


איור 8.5 מטריצת משקלים של המשפט "Find the collector rate on the incentive rate table". ככל שקשר בין שתי מילים חזק יותר, כך המשקל ביניהם גבוה יותר. כמובן שיש גם משמעות לסדר – המשקל בין "Find" ל-"collector" שונה מהמשקל שבין "collector" ל-"Find".

כעת בעזרת משקלים אלו בונים ייצוג חדש לסדרה המקורית, על ידי הכפלתם בוקטור V :

$$z_i = \sum_{j=1}^n w_{ij} v_j = \frac{\sum_{j=1}^n \exp(q_i^T k_j) v_j}{\sum_{s=1}^n \exp(q_i^T k_s)}$$

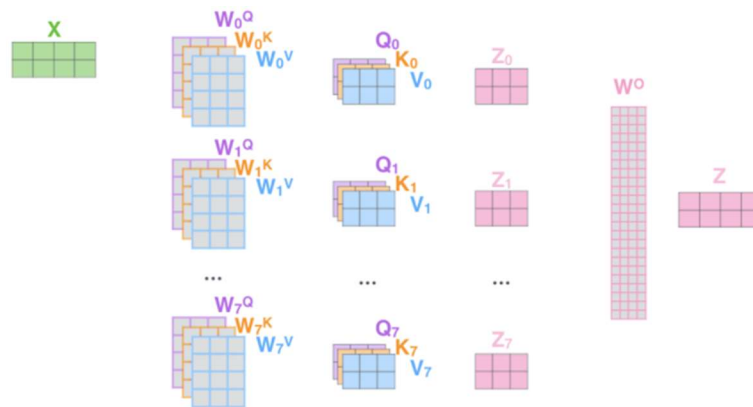
הסדרה המתקבלת z היא למעשה ייצוג חדש של הסדרה המקורית, כאשר כל איבר z_i מייצג איבר בסדרה המקורית יחד עם מידע על הקשרים בינו לבין שאר איברי הסדרה. את הסדרה המתקבלת ניתן להעביר ב-decoder המכיל שכבות נוספות, ובכך לבצע כל מיני משימות, כפי שיוסבר בהמשך.



איור 8.6 ביצוע Self-attention – יצירת מטריצות Query, Key, Value (שמאל) וחישוב ה-attention score (ימין).

8.2.3 Multi Head Attention

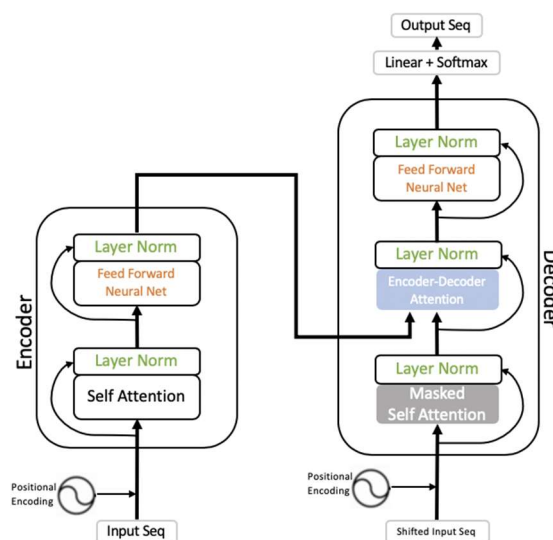
ניתן להשתמש במנגנון self-attention מספר פעמים במקביל. כל פעם מקבלים שלשת מטריצות (Q_r, K_r, V_r) , ובעזרתה מחשבים את הייצוגים החדשים של איברי הסדרה (attention score). כל מנגנון כזה נקרא attention head, וחיבור במקביל של כמה attention heads נקרא Multi-head attention. באופן הזה לכל איבר כניסה x_i יש כמה ייצוגים שונים z_{ir} , אותם ניתן להכפיל במטריצת משקלים w_o ולקבל את הייצוג המשוקלל של אותו איבר באמצעות attention heads מרובים.



איור 8.7 Self-attention with 8 heads.

8.2.4 Transformer End to End

בעזרת מנגנוני multi head attention ו-positional encoding ניתן לבנות Transformer – ארכיטקטורה עבור סדרות המבוססת רק על attention ללא רכיבי זיכרון.



איור 8.8 Transformer.

כפי שניתן לראות באיור ה-transformer מורכב משני חלקים – encoder ו-decoder. ה-encoder מקבל סדרה מסוימת x (לרוב אחרי שעברה embedding מסוים) ומבצע עליה positional encoding. לאחר מכן הסדרה עוברת דרך residual block של self-attention, ומתקבלת התוצאה $x + \text{attention}(x)$. על תוצאה זו מבצעים layer normalization (כפי שהוסבר בפרק 5.1.4). לאחר מכן יש residual block נוסף, המכיל שכבת fully connected, ומשם יוצא הפלט לכיוון ה-encoder.

ה-decoder בנוי בצורה מאוד דומה, עם שני הבדלים עיקריים: הקלט שלו הוא איברי הפלט שהיו עד כה, ובנוסף יש בתחילת ה-decoder שכבה של masked self-attention. שכבה זו מקבלת את כל איברי הפלט שהיו עד כה, ומטרת ה-decoder היא ללמוד בעצמו מה האיבר הבא של הפלט. בשלב הראשון ה-decoder מבצע self-attention על איברי הסדרה שהתקבלו עד כה, וכך לומד ייצוג חדש שלהם, המכיל גם את הקשר בין איברי סדרה זו.

לאחר השכבה הראשונה יש שכבת multi head attention נוספת הנקראת Encoder-Decoder Attention, כיוון שהיא שילוב של ה-encoder וה-decoder: המטריצות Key, Value נלקחות מה-encoder וה-Query מגיע מה-decoder. כעת כשמבצעים את המכפלה $Q \cdot K$, לא מחפשים דמיון בין איברים של אותה סדרה אלא בין האיברים של סדרת הפלט (בייצוג שלהם לאחר ה-encoder) לבין איברי סדרת הקלט (בייצוג שלהם לאחר שכבת ה-masked). שלב זה דומה מאוד ל-attention המקורי, רק שהייצוגים שהתקבלו לא נעזרו ברכיבי זיכרון. כאמור, המכפלה $Q \cdot K$ מייצרת מטריצת משקלים שכל איבר בה אומר מה היחס בין איבר בסדרה המקורית לבין איבר בסדרת הפלט. את המטריצה הזו מכפילים ב- V וכך מקבלים וקטור מסוים שהוא ייצוג חדש של איבר הפלט הבא. וקטור זה עובר בשכבת FC וב-SoftMax, וכך מתקבל איבר הפלט.

ניקח דוגמא ממאמר שנקרא [DETR](#) המראה כיצד ניתן להשתמש ב-transformer בשביל זיהוי אובייקטים בתמונה. בשלב הראשון לוקחים כל פיקסל בתמונה ומשווים אותו לשאר הפיקסלים (זוהי בעצם המכפלה $Q \cdot K$). באופן הזה ניתן למצוא אזורים דומים ושונים בתמונה, כאשר דמיון ושוני זה לאו דווקא פיקסלים עם ערכים קרובים, אלא זה יכול להיות למשל שני אזורים שונים בפנים של אדם. לאחר מכן מייצרים ייצוג חדש לתמונה, בעזרת המשקלים והכפלתם ב- V . שלב זה למעשה מאפשר לבצע זיהוי של אובייקטים, בלי לדעת מה הם אותם אובייקטים. בשביל לבצע סיווג לכל אובייקט שזוהה, מעבירים את הייצוג החדש של התמונה ב-decoder, כאשר ה-Query שמכניסים זה כל מיני לייבלים אפשריים, ומחפשים מבין כל ה-Query את הפלט של ה-decoder שמצליח לייצר תמונה שהכי דומה ל-Query.

אם למשל יש תמונה גדולה ויש אזור מסוים בו יש חתול, אז ה-encoder מוצא איפה החתול בתמונה, וה-decoder משווה את האזור הזה לכל מיני חיות אפשריות. כל Query שלא יהיה חתול, המכפלה $Q \cdot K$ תהיה קרובה ל-0, וה-decoder יזהה שה-Query הנוכחי לא תואם לאובייקט שזוהה. אך כאשר ה-Query יהיה חתול, אז כיוון ש- $Q \cdot K$ דומים אחד לשני, המכפלה $Q \cdot K$ תביא לכך שהייצוג החדש $z_i = \sum_{j=1}^n w_{ij} v_j$ כן יהיה דומה לחתול. ייצוג זה עובר בשכבת FC, ולאחר מכן ה-SoftMax יסווג את התמונה הזו כחתול.

8.2.5 Transformer Applications

ה-transformer הציג ביצועי state-of-the-art במגוון משימות, והוא היווה השראה להמון יישומים הנשענים על attention בלבד. מלבד הרמה הגבוהה של הביצועים, תהליך האימון של transformer הוא הרבה יותר מהיר מרשתות קונבולוציה או רשתות רקורסיביות. כמו במודלים אחרים, גם עם transformer ניתן לבצע transfer learning, כלומר לקחת transformer שאומן על משימה מסוימת, ולהתאים אותו למשימה חדשה שדומה למשימה המקורית. בפועל לא כל היישומים משתמשים בכל ה-transformer, אלא בהתאם למשימה לוקחים חלקים מסוימים שלו ובונים בעזרתם מודל עבור משימה מסוימת. נביא מספר דוגמאות:

Machine Translation – תרגום משפטים בין שפות שונות הוא יישום טריוויאלי של ה-transformer המלא. המשימה היא לקחת משפט ולהוציא משפט בשפה אחרת, וזה נעשה בעזרת ייצוג המשפט המקורי באופן חדש בעזרת self-attention ולאחר מכן המרתו בעזרת Encoder-Decoder Attention לשפה אחרת.

Bidirectional Encoder Representations from Transformers (BERT) – מודל שפה מבוסס encoder. מודל שפה הוא פונקציה המקבלת כקלט טקסט ומחזירה את ההתפלגות למילה הבאה על פי כל המילים במילון. השימוש הכי מוכר ואינטואיטיבי של מודל שפה הוא השלמה אוטומטית, שמציעה את המילה או המילים הכי סבירות בהינתן מה שהשתמש הקליד עד כה. כאשר מבצעים self-attention על משפטים, למעשה מקבלים ייצוגים חדשים שלהם יחד עם ההקשרים בין המילים השונות. לכן ה-encoder ב-transformer יכול ליצור מודל שפה, אם מאמנים אותו בצורה מתאימה. המפתחים של BERT בנו encoder המקבל כל מיני משפטים בשני כשני הכיוונים – גם מההתחלה לסוף וגם מהסוף להתחלה, וכך הייצוגים שנלמדו קיבלו קונטקסט שלם יותר. בנוסף, הם אימנו את המודל על משפטים בהם כל פעם באופן רנדומלי עושים masking למילים מסוימות, ומטרת המודל הוא לחזות את המילים החסרות.

Generative Pre-Training (GPT) – מודל לחיזוי המילה הבאה במשפט. ניתן לקחת משפט שקטוע באמצע, ולבחון מהי המילה הבאה באמצעות decoder בלבד. מכניסים משפט קטוע ל-decoder ואז עוברים על המון מילים ובודקים את ההתאמה שלהן למשפט הנתון, והמילה שהכי מתאימה נבחרת להיות המילה הבאה. המשפט הקטוע הוא למעשה ה-**Key**, וה-**Query** שנכנס הוא כל פעם מילה אחרת במילון, וכך בעזרת attention בוחנים איזה **Query** יתאים בצורה הטובה ביותר ל-**Key** הנתון.

References

<https://arxiv.org/abs/1409.0473>

<https://arxiv.org/abs/1706.03762>

<https://towardsdatascience.com/day-1-2-attention-seq2seq-models-65df3f49e263>

<https://towardsdatascience.com/transformer-attention-is-all-you-need-1e455701fdd9>

<https://arxiv.org/abs/1810.04805>

9. Computer Vision

9.1 Object Detection

9.1.1 Introduction to Object Detection

זיהוי אובייקטים היא משימה מאוד נפוצה בעולם של ראייה ממוחשבת, ויש לה המון יישומים מגוונים. ניקח למשל מכונית אוטונומית, שבכל רגע צריכה לזהות את האובייקטים שסביבה ולקבל תמונת מצב עדכנית על המתרחש, או לחילופין מצלמה של טלפון נייד שיודעת לזהות פנים של בנאדם בכדי לבצע עליהם פוקוס או לתקן רעשי רקע, ועוד המון יישומים נרחבים בכל מיני תחומים.

בשלב ראשון יש להגדיר באופן מדויק את המשימה – מה הכוונה לזהות אובייקט בתמונה? יש כמה רמות שונות של זיהוי. המשימה הקלאסית של סיווג (classification) מניחה שיש אובייקט יחיד בתמונה, והמטרה היא לסווג אותו בצורה נכונה, כלומר לקבוע מה ה-class שלו. משימה יותר מתקדמת היא לא רק לומר איזה אובייקט נמצא בתמונה, אלא גם לומר איפה בדיוק הוא נמצא. גם פה יש שתי רמות – ניתן לסמן את האזור בו הוא נמצא בעזרת מלבן (bounding box) שמקיף את השוליים שלו, וניתן לסווג כל פיקסל בפני עצמו האם הוא שייך לאובייקט או לרקע. הזיהוי מהסוג הראשון נכנס תחת התחום של Object Detection ואילו זיהוי סמנטי של הפיקסלים נכנס תחת התחום של Segmentation.

ניתן להכליל את משימת הזיהוי גם ל-multiple object. כלומר, יש מספר לא ידוע של אובייקטים בתמונה, והמטרה היא למצוא היכן הם נמצאים ולסווג כל אחד מהם ל-class המתאים. בעצם משימה זו מורכבת משתי תתי משימות – מציאת המיקום של האובייקט (Localization/Regression) וסיווג האובייקט ל-class הנכון (Classification). נשים לב שעבור משימת Object Detection, ההכללה מאובייקט יחיד למספר אובייקטים הינה ישירה – על המודל לספק מספר bounding boxes ועבור כל אחד מהם להתאים class. במשימת Segmentation לעומת זאת, ההכללה למספר אובייקטים יכולה להיעשות בשתי דרכים: (1) Semantic Segmentation – שיוך כל פיקסל ל-class מסוים ללא הבחנה בין פיקסלים השייכים לאובייקטים שונים בעלי אותו class. במקרה זה אם יהיו שני כלבים בתמונה, אז במפת הסגמנטציה נראה הרבה פיקסלים המשוויכים ל-class של כלב, אך לא נוכל להבין בין הכלבים השונים, ואפילו לא נדע שיש יותר מכלב אחד. (2) Instance Segmentation – שיוך כל פיקסל ל-class מסוים תוך הבחנה בין פיקסלים המשתייכים ל-class ספציפי אך שייכים לאובייקטים נפרדים.

בפרק זה נדון במשימת Object Detection ובפרק הבא ב-Segmentation. בשביל להגדיר את המשימה ולקבוע מה נחשבת הצלחה, ראשית יש להגדיר מה נחשב השטח בו נמצא האובייקט, ובהתאם מה אנו מצפים מהמודל שישפק לנו כפלט. בתחום של Object Detection, המוסכמה היא לסמן אובייקט באמצעות מלבן, שנקרא bounding box (או בקיצור bbox), כאשר קצוות האובייקט משיקים ל-bbox. נניח ויש תמונה ובה בנאדם, אז ה-bbox יהיה מלבן המכיל את כל הגוף שלו, והמלבן ישיק לקצוות הגוף – החלק העליון של המלבן ישיק לנקודה הגבוהה ביותר של הראש, החלק התחתון ישיק לכפות הרגליים, ובאופן דומה בשני הצדדים. אם יש כמה אובייקטים, אז לכל אובייקט ישוּך bbox בנפרד, כפי שניתן לראות באיור הבא:

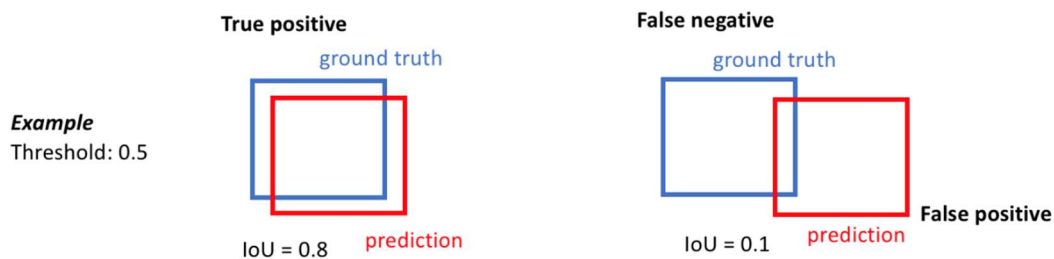


איור 9.1 תמונה ובה מספר אובייקטים, כאשר כל אובייקט מסומן באמצעות bounding box – מלבן המקיף אותו ומשיק לקצוות שלו מכל כיוון.

לאחר שהגדרנו בצורה מדויקת מה נחשב השטח בו נמצא האובייקט – נוכל לקחת תמונה ולסמן בה את האובייקטים בהתאם למוסכמה זו, כאשר נהוג לכנות כל אובייקט מתויג כ-groundtruth. אם נאמן גלאי (detector) לפי המוסכמה הזו, אז נרצה שהפלט שלו יהיה אוסף של דטקציות (detections) – מלבנים מסביב לאובייקטים יחד עם סיווג תואם

לכל אחד מהאובייקטים. כאשר נריץ את הגלאי המאומן על תמונה מסוימת, נוכל להשוות בין כל האובייקטים שתויגו (groundtruths), לבין הדטקציות של המודל. מקובל לבצע את ההשוואה בעזרת מדד שנקרא Intersection Over Union, או בקיצור IoU. מדד זה בוחן עבור על אובייקט את היחס של גודל החפיפה בין ה-groundtruth לבין הדטקציה התואמת עבורו. אם החפיפה גדולה מסף מסוים, אז נוכל לומר שהגלאי זיהה בהצלחה שיש פה אובייקט. סף זה נקרא IoU threshold, וערך טיפוסי עבורו הינו 50%, כלומר חפיפה של מעל 50% בין מיקומו האמיתי של האובייקט לבין הדטקציה של המודל נחשבת כמצאה נכונה של האובייקט. אם גם הסיווג של אותו אובייקט על פי המודל זהה ל-class האמיתי (בהתאם למה שתויג מראש), אז יש פה התאמה מלאה ואותו אובייקט הוא בעצם true positive – דוגמה חיובית שזוהתה באופן נכון.

אם יש אובייקט שאין אף דטקציה שחופפת לאזור בו הוא נמצא, אז אותו אובייקט הינו false negative (groundtruth שלא זוהה). אובייקט יהיה false negative גם במצבים בהם יש חפיפה אך היא אינה עוברת את הסף שנקבע, או לחילופין יש חפיפה מספיקה אך הסיווג שגוי. נשים לב שבשני המקרים האחרונים יש גם false positive – הדטקציה אינה מתאימה לאף groundtruth, ולכן היא למעשה זיהוי חיובי של אובייקט שבאמת לא קיים. הגדרות אלו מאפשרות להשתמש במשימת Object Detection במטריות מקובלות כמו precision ו-recall באופן דומה לשימוש הקלאסי שלהם במשימות classification, כפי שיוסבר בהרחבה בהמשך.



איור 9.2 Intersection Over Union (IoU) – מדד חפיפה בין groundtruth לבין דטקציה. עבור IoU threshold=0.5, דטקציה שחופפת בלפחות 50% ל-groundtruth הינה true positive, ואילו דטקציה שאינה עוברת את סף החפיפה הינה false negative ובנוסף ה-groundtruth הינו false negative.

לאחר שהגדרנו בצורה מדויקת ומפורטת מהי משימת זיהוי אובייקטים נסביר בקצרה על הגישות השונות בתחום והתפתחותן לאורך השנים. אמנם בשנים האחרונות כל המודלים הינם מבוססי Deep Learning, אך כמובן שמשמית זיהוי אובייקטים הייתה קיימת עוד לפני התפתחות הענף והיו אלגוריתמים קלאסיים לצורך כך, כמו למשל אלגוריתם Scale-Invariant Feature Transform (SIFT) שהיו פופולרי מאוד בעשור הראשון של המאה. לרוב, גם האלגוריתמים הקלאסיים וגם אלו שמבוססי רשתות נוירונים פועלים על פי אותו רעיון, שהינו פשוט ואינטואיטיבי. כיוון שמשמית זיהוי אובייקטים כלולה משני חלקים – Localization and Classification, נרצה שהמודל שלנו גם הוא יפעל בשני חלקים – בשלב הראשון נמצא אזורי עניין בתמונה, כלומר אזורים החשודים ככאלה שיש בהם אובייקטים (Region Of Interest – ROI), ולאחר מכן נבצע שני דברים במקביל – ננסה לחזות את ה-bbox המדויק של אובייקט הנמצא באותו אזור, ובנוסף ננסה לסווג אותו לקלאס מסוים.

למרות שגישה זו פשוטה יחסית, יש בה שתי בעיות מובנות. ראשית כל – יתכנו אזורים שהמודל יחשוד שיש בהם אובייקט אך בפועל באמת אין בהם אובייקט אלא הם מכילים רק רקע. שנית – הרבה אלגוריתמים מתוכננים באופן כזה שהקלט שלהם הוא בגודל קבוע וידוע מראש, והם גם מוציאים פלט בגודל קבוע. כיוון שמספר האובייקטים הוא משתנה, פלט בגודל קבוע יכול להיות בעייתי. למעשה יש פתרון פשוט לשתי הבעיות – נדרוש מהמודל לצרף לכל אחת מהדטקציות מספר הסתברותי בין 0 ל-1 שמשמעותו היא "עד כמה אני בטוח שבאמת יש פה אובייקט". במקביל לכך, על המשתמש להחליט על סף מסוים (confidence threshold), כך שרק אובייקטים בעלי הסתברות הגבוהה מסף זה יחשבו כזיהויים של המודל, ומכל יתר הזיהויים פשוט נתעלם. באופן הזה נוכל לקבוע מראש שהמודל יוציא מספר קבוע של זיהויים, למשל 100, אך נתעלם מכל אלה שלא עברו את הסף. מודל טוב ייתן הסתברות גבוהה רק לאזורים בהם באמת יש אובייקטים, ואילו כל יתר האזורים החשודים יהיו בעלי הסתברות נמוכה. בנוסף, מודל כזה ידע להתעלם מאזורי עניין שנמצאו כחשודים בשלב הראשון שלו ולתת להם הסתברות נמוכה. (בהמשך יוסבר על אלגוריתם NMS שגם מתקשר לעניין של מספר הדטקציות).

כאמור לעיל, בשנים האחרונות כל הגישות המובילות בתחום זה הן מבוססות Deep Learning, כאשר הן פועלות על פי אותו עיקרון – חילוץ ROIs בשלב הראשון, ולאחר מכן ביצוע רגרסיה ל-bbox של האובייקט יחד עם סיווג שלו ל-class ספציפי. באופן כללי ניתן לחלק את הגלאים לשלוש קבוצות:

גלאי דו-שלבי (Two-stage detector):

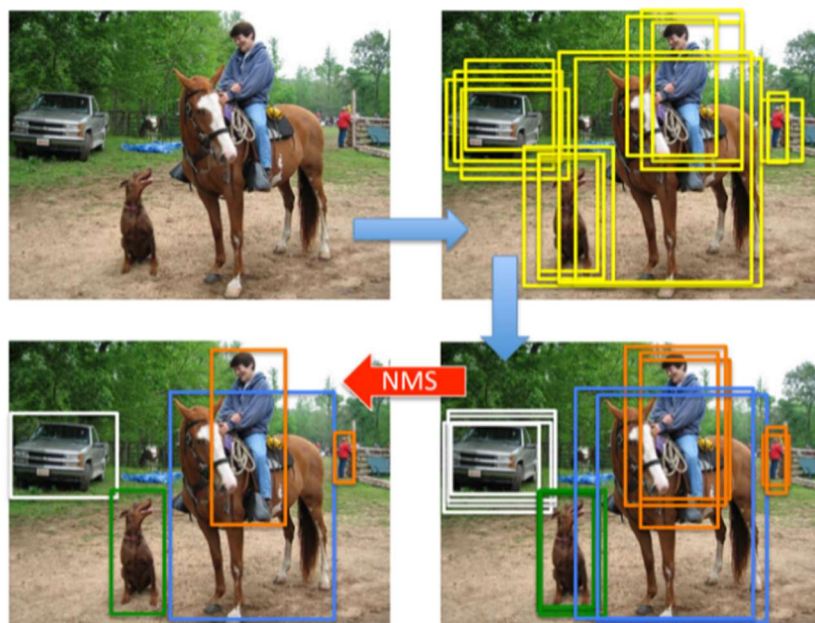
גישה אחת, הנקראת two-stage detection, אכן פועלת לפי שני השלבים שתיארנו קודם: בהתחלה מופעל אלגוריתם לזיהוי ROIs, כאשר אלגוריתם זה הוא רשת נוירונים שמחלצת פיצ'רים בתמונה נתונה ומזהה בה אזורי עניין, ולאחר מכן אזורים אלו נכנסים לרשת נוספת המבצעת רגרסיה ל-bbox וסיווג ל-class. המוצא של הרשת השנייה בנוי בצורה של two head detection, כאשר לרוב ראש הרגרסיה מנסה ללמוד בתהליך האימון (ולחזות ב-inference) ארבעה פרמטרים – את מרכז האובייקט (x, y) והאורך והרוחב שלו (h, w) .

גישה זו היא בעצם השיטה הכי אינטואיטיבית, ומשפחת הגלאים הראשונה ואולי הכי מפורסמת הפועלת לפי עיקרון זה של two-stage היא R-CNN Family. משפחה זו הינה סדרה של גלאים שפותחו בשנים 2014-2017, כאשר כל גלאי מתבסס על קודמו ומשפר אותו, וכולם יחד בנויים בצורה של two-stage. עם הזמן התפתחו עוד גלאים הפועלים באופן דומה, כמו למשל SSP ו-FPN שנדון בהם בהרחבה בהמשך הפרק.

גלאי חד-שלבי (One-stage detector):

גישה נוספת, דומה בקונספט אך שונה בדרך הביצוע, נקראת one-stage detection. גם בגישה זו הרעיון הוא למצוא אזורי עניין ועבורם לבצע רגרסיה של bbox וסיווג ל-class, אך הפעם אזורי העניין אינם נקבעים באמצעות אלגוריתם כלשהו, אלא אנו **קובעים אותם מראש**. אנו למעשה אומרים מראש למודל איפה בדיוק לחפש אובייקטים, כאשר אותם אזורים מסומנים מראש נקראים anchors. בדרך כלל קובעים מראש הרבה anchors, כאשר יש anchors בגדלים ובצורות שונות, על מנת שנוכל לזהות אובייקטים במגוון גדלים וצורות. למשל – נחלק את התמונה לגרידים בגדלים שונים וכל מלבן יהיה anchor, אך נרצה להוסיף גם anchors בצורת מלבנים רוחביים בכדי לזהות מכוניות, וכן נרצה anchors בצורת מלבנים אופקיים בכדי לזהות אנשים עומדים, וכך הלאה. העיקרון הוא שנקבע מראש הרבה anchors כך שכל אזור בתמונה יכוסה במספר anchors בגדלים שונים, בכדי לאפשר זיהוי של אובייקטים בעלי אופי שונה. גלאי מפורסם שעובד לפי העיקרון של one-stage הינו YOLO – You Only Look Once, וכשמו – הוא עובר על התמונה רק פעם אחת (ולא כמו ב-two-stage), שם יש צורך לעבור פעמיים על התמונה – פעם אחת לחילוף ROIs (פעם שנייה לרגרסיה וסיווג). גלאים נוספים הפועלים לפי גישה זו הינם SSD ו-RetinaNet, שגם בהם נדון בהרחבה בהמשך.

בשונה מגלאי דו-שלבי המחלץ בעצמו את אזורי העניין, גלאי המשתמש ב-anchors קובע מראש את מספר אזורי העניין. עובדה זו יכולה לגרום לכך שיהיו כמה אזורי עניין שונים המכילים את אותו אובייקט. אם למשל חילקנו את התמונה לגריד של 10×10 ויש אובייקט שתופס רבע מהתמונה, אז יהיו כ-25 anchors שיכילו בתוכם את האובייקט. כדי לאחד את כל הזיהויים של ה-anchors האלה מקובל להשתמש באלגוריתם קלאסי שנקרא Non Maximum Suppression, או בקיצור NMS. הרעיון של האלגוריתם הוא לעבור על כל הדטקציות, ואם יש דטקציות השייכות לאותו קלאס ויש ביניהן חפיפה מסוימת, אז הן יתאחדו לכדי דטקציה אחת. השימוש באלגוריתם זה נפוץ גם בגלאי דו-שלבי – אמנם הגלאים מחלצים לבד את אזורי העניין, אך ישנם גלאים בהם מספר אזורי העניין קבוע מראש, ולכן יתכנו הרבה דטקציות של אותו אובייקט, ויש צורך לאחד אותן.



איור 9.3 Non-Maximum-Suppression (NMS) – איחוד דטקציות השייכות לאותו אובייקט. בשלב הראשוני מחפשים אזורי עניין, לאחר מכן מסווגים כל אחד מהם, ולבסוף מאחדים זיהויים השייכים לאותו class ויש ביניהם חפיפה גבוהה.

גלאי מבוסס טרנספורמר (Transformer-based detector)

כניסתם של הטרנספורמרים לעולם ה-vision לא דילגה על Object Detection. ב-2020 יצא מאמר שנקרא DETR (קיצור של DEtection Transformers) שהראה כיצד ניתן להשתמש במנגנון של self-attention בשביל למצוא אובייקטים בתמונה ולאחר מכן להשתמש במנגנון של attention בשביל לסווג אותם, כפי שיפורט בהרחבה בהמשך הפרק. מאז יצאו הרבה מאמרים המבוססים על הרעיונות שהונחו ב-DETR, כאשר מאמרים אלו מנסים לשפר את התוצאות תוך התייחסות למספר בעיות מובנות שעולות כאשר משתמשים בטרנספורמרים במשימות של ראייה ממוחשבת. בפרט, יש מספר אתגרים בסיסיים: א. הסיבוכיות של טרנספורמר היא ריבועית בגודל הקלט. כאשר הקלט הוא משפט, או אפילו פסקה, זה לא מהווה בעיה, אך כאשר מדובר בתמונות ברזולוציה גבוהה (למשל 4K), אז שימוש בטרנספורמרים נהיה לא יעיל, ועבור משימות זמן אמת אפילו בלתי אפשרי. ב. טרנספורמרים תוכננו במקור לקלט חד ממדי, כמו למשל משפט שהוא סדרה של מילים. בשביל להשתמש בהם עבור תמונות, יש להפוך את התמונות מייצוג דו-ממדי לייצוג חד ממדי. מעבר זה יכול לגרום לאיבוד קשרים מרחביים הקיימים בתמונה. ג. מנגנון ה-attention יכול לזהות תבניות מרחביות, אך הוא לא מתייחס לממד הצבעים, שכמובן יש לו משמעות בעיבוד התמונה.

לכל אחת מהגישות יש יתרונות וחסרונות וכמובן שיש כל מיני עבודות שמנסות לשלב בין היתרונות של הגישות השונות. באופן פשוט ניתן לומר ביחס לשתי הגישות הראשונות, שהן מקיימות trade-off בין דיוק לבין מהירות – גלאי חד-שלבי הוא כמובן יותר מהיר, אך עם זאת הוא פחות מדויק. עבור מכשירי קצה שרצים בזמן אמת ואין להם כוח חישוב רב, כמו למשל פלאפונים, הצרכים העיקריים של המשתמש הם מהירות חישוב וחסכון בסוללה, ולכן כנראה שתהיה עדיפות לגלאי חד-שלבי. אם לעומת זאת המשתמש צריך דיוק גבוה ויש לו משאבים חישוב חזקים, כמו למשל רופא שמתעסק בפענוח בדיקות רפואיות, אז תהיה לו עדיפות לגלאי דו-שלבי או לגלאי מבוסס טרנספורמר, המספקים תוצאות מעט יותר טובות.

כאמור לעיל, כל הגלאים בשנים האחרונות הם מבוססי רשתות נוירונים – רשתות עבור זיהוי ROIs, רשתות רגרסיה וסיווג, רשתות של טרנספורמרים ועוד. בשביל לאמן את הרשתות האלה יש צורך בדאטה, וכמובן שצריך שהוא יהיה מתויג. בשונה מתיוג תמונות למשימות סיווג, שם כל שנדרש זה רק לתת label לכל תמונה, תיוג של תמונות עבור משימות Object Detection היא משימה מורכבת פי כמה – גם יש כמה אובייקטים בתמונה, וגם צריך לסמן לכל אחד מהם bbox בצורה מדויקת. עקב המורכבות של תיוג כזה, התפתחו אלגוריתמים של semi-supervised, המשלבים למידה מבוססת דאטה מתויג יחד עם למידה שאינה דורשת תיוג.

נציין בקצרה רעיון אחד של semi-supervised learning עבור משימת זיהוי אובייקטים, המאפשר לאמן מודל בלי הרבה דאטה מתויג. הרעיון הוא לאמן את המודל בשני שלבים – בשלב הראשון משתמשים בדאטה המתויג ומאמנים מודל באחת הדרכים שראינו עד כה. לאחר מכן משפרים את הלוקליזציה באופן הבא – מריצים אלגוריתם unsupervised שיועד לזהות אובייקטים (הוא בעצם עושה סוג של clustering לתמונה ומבחין בין אובייקטים לבין אזורים שאינם אובייקטים), ומשווים בין הפלט שלו לפלט של המודל המקורי שאימנו. כיוון שהאלגוריתם השני מאמן רק לבצע לוקליזציה והוא אינו נדרש להבחין בין אובייקטים שונים, ניתן להשתמש עבורו בהרבה דאטה, ולכן הנחה סבירה היא שהוא מזהה אובייקטים בצורה טובה. אם כן, ניתן לבצע אימון שינסה לעשות fine tune ללוקליזציה של המודל הראשון כך שה-bounding boxes הוא מספק יהיו דומה עד כמה שניתן לפלט של האלגוריתם השני.

בדומה לכך, ישנם אלגוריתמים של self-supervised שאינם מצריכים בכלל דאטה מתויג. אלגוריתמים אלה לומדים לזהות אובייקטים ולהבחין בין אובייקטים שונים, אך הם אינם יכולים לדעת מה ה-label של כל קבוצה, כיוון שאין להם שום דאטה מתויג. זה יכול להיות שימושי למשל עבור רופאים המסתכלים על תוצאות של צילומים רפואיים – האלגוריתם יזהה את האובייקטים ויסווג כל אובייקט לקבוצה מסוימת, ואז הרופא יסתכל על התוצאות וידע לתת את ה-label המתאים לכל אחת מהקבוצות.

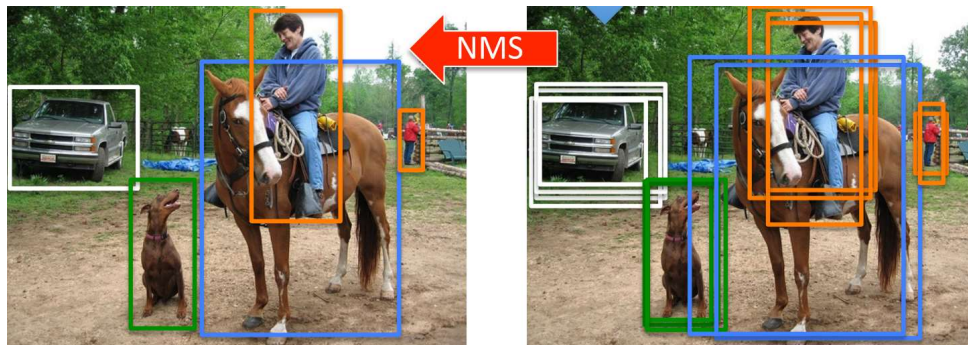
9.1.3 You Only Look Once (YOLO)

עד שנת 2016 כל הגלאים (detectors) המבוססים על למידה עמוקה (כמו למשל R-CNN family) היו גלאים דו-שלביים – השלב הראשון יצר אלפי הצעות (proposals) למסגרות מלבניות החשודות כמכילות אובייקטים, ואלו הוזנו בזו אחר זו לשלב השני אשר דייק את המסגרות וביצע סיווג לאובייקטים המוכללים בהן. ארכיטקטורת YOLO, הנגזרת מראשי התיבות של YOU ONLY LOOK ONCE, הוצגה על ידי ג'וזף רדמן ב-2016, והייתה הגלאי הראשון שמרכב משלב יחיד, ובו הרשת מנבאת את המסגרות וגם מסווגת את האובייקטים שבתוכן בבת אחת. בנוסף, ארכיטקטורת YOLO מתאפיינת במיעוט פרמטרים וכמות פעולות אריתמטיות. היא אמנם משלמת

על המבנה הרזה והאלגנטי שלה בדיוק נמוך יותר, אך המהירות הגבוהה (שנובעת בעיקר מהיעדר האילוץ לעבד מסגרת יחידה בשלב השני בכל מעבר של הלולאה) הפכה את גישת השלב היחיד לאטרקטיבית מאוד, במיוחד למעבדים קטנים כדוגמת מכשירי mobile. בעקבות עבודה זו פותחו גלאים רבים על בסיס שלב יחיד, כולל גרסאות מתקדמות יותר של YOLO (הגרסה המתקדמת ביותר כיום היא v5).

NMS (Non-Maximum Suppression)

כמעט כל אלגוריתם של זיהוי אובייקטים מייצר מספר רב של מסגרות חשודות, כאשר רובן מיותרות ויש צורך לדלל את מספרן. הסיבה ליצירת מספר גדול של מסגרות נובע מאופי פעולת הגלאים. בזכות התכונות הלוקאליות של פעולת הקונבולוציה, מפת הפיצ'רים במוצא הגלאי ניתנת לתיאור כמטריצת משבצות כאשר כל משבצת שקולה לריבוע של הרבה פיקסלים בתמונה המקורית. רוב הגלאים פועלים בשיטת עוגנים (anchors), כאשר כל משבצת במוצא הגלאי מנבאת מספר קבוע של מסגרות שעשויות להכיל אובייקט (למשל, ב-YOLOv2 המספר הוא 5, ובגרסאות המתקדמות יותר המספר הוא 3). השיטה הזו יוצרת אלפי מסגרות שרק מעטות מהן הן משמעותיות. בנוסף – יש ריבוי של מסגרות דומות בסביבת כל אובייקט (למשל – YOLOv3 מנבאת יותר מ-7000 מסגרות לכל תמונה). אחת הדרכים הפופולריות לסנן את אלפי המסגרות ולהשאיר רק את המשמעותיות נקראת NMS. בשיטה זו מתבצעת השוואה בין זוגות של קופסאות מאותה המחלקה (למשל – חתול), ובמידה שיש ביניהן חפיפה גבוהה – מוחקים את המסגרת בעלת הוודאות הנמוכה ביותר ונשארים רק עם המסגרת בעלת רמת הוודאות הגבוהה. שיטה זו בזבזנית בחישוב (סיבוכיות פרופורציונלית לריבוע מספר המסגרות) ואינה חלק מהמודל המתאמן, אך עם זאת הינה אינטואיטיבית יחסית למימוש, ומשום כך נמצאת בשימוש נפוץ בגלאים, כולל ב-YOLO.

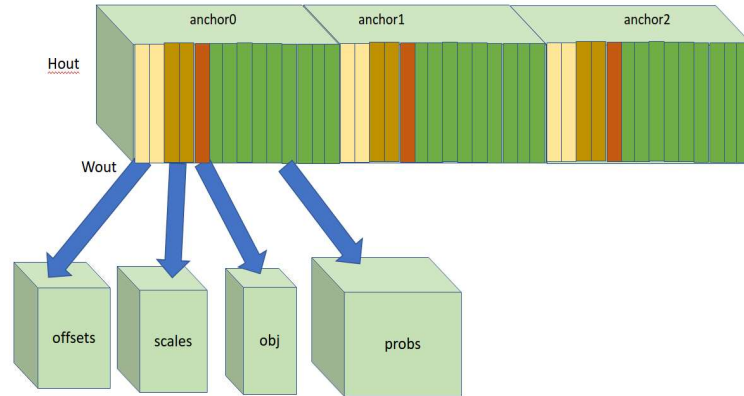


איור 9.1 אלגוריתם NMS.

YOLO Head

כמו רוב הגלאים, YOLO הינו מבוסס-עוגנים (anchor-based), שהינן תיבות מלבניות קבועות ושונות זו מזו בצורתן. לכל עוגן מוקצה מקטע של פיצ'רים במפת המוצא של הרשת וכל הניבויים במקטע הזה מקודדים כסטיות (offsets) ביחס לממדי העוגן. כפי שניתן לראות באיור 9.2, הפיצ'רים של כל תא מרחבי במפת המוצא מחולקים למקטעים על פי העוגנים (שלושה עוגנים במקרה הזה).

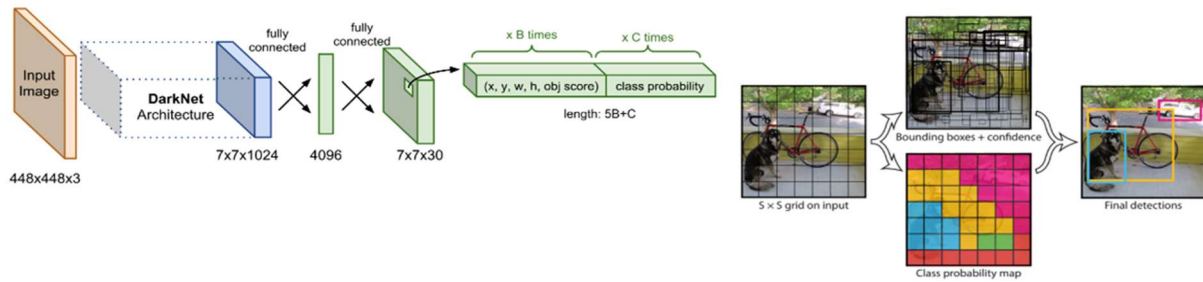
ניבוי הוא מסגרת שמרכזה נמצא בנקודה כלשהי בשטח התא (השקול לריבוע של מספר פיקסלים בתמונה המקורית). ההסטה המדויקת של מרכז המסגרת ביחס לתא ניתנת על ידי שני הפיצ'רים הראשונים ברצף. לוג הממדים של הקופסא (ביחס לממדי העוגן) ניתן באמצעות שני הפיצ'רים הבאים ברצף. הפיצ'ר החמישי לומד את מידת ה-objectness, כפי שהוסברה לעיל. שאר הפיצ'רים ברצף של העוגן הנ"ל הם ההסתברויות המותנות לכל מחלקה (אם אוסף הנתונים מכיל 80 מחלקות, יהיו 80 פיצ'רים כאלה). על מנת לקבל רמת ודאות סופית, יש להכפיל את מדד ה-objectness במדד ההסתברות המותנה לכל מחלקה.



איור 9.2 ראש YOLO.

YOLOv1

מודלי YOLO מבוססים על גרסאות Backbone הנקראות Darknet ומשמשות לעיבוד פיצ'רים מתוך התמונה, ומראש detection המקבל את הפיצ'רים האלה ומתאמן לייצר מהם ניבויים למסגרות סביב אובייקטים. המודל מחלק את התמונה לרשת בעלת $S \times S$ משבצות, כאשר כל משבצת מנבאת N מסגרות של אובייקטים בשיטת העוגנים, כאמור לעיל. כל ניבוי כולל את מספר ערכים: הסטת הקואורדינטות x, y של מרכז המסגרת ביחס למשבצת, הגובה והרוחב של המסגרת, ורמת ה-objectness, כפי שהוסברה לעיל. בנוסף, כל מסגרת מבצעת גם סיווג, כלומר מנבאת את רמת הוודאות של השתייכות האובייקט לכל אחת מהמחלקות האפשריות. החידוש באלגוריתם נעוץ בעובדה שחיזוי המסגרות וסיווגן לאובייקטים נעשה במקביל, ולא באופן דו-שלבי. הרעיון הוא להתייחס לסוג האובייקט כעוד פיצ'ר שהרשת מנסה לחזות בנוסף למיקום וגודל של המסגרת.

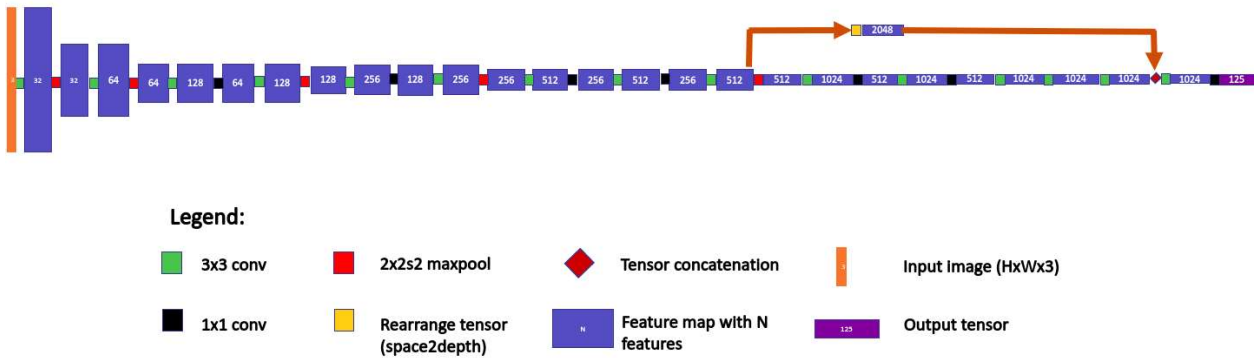


איור 9.3 ארכיטקטורת YOLO.

הגרסה הראשונה של ארכיטקטורת YOLO כללה שכבת fully connected שהוסרה בגרסאות הבאות. בנוסף, פונקציית ה-LOSS וסדר התכונות של המסגרות במוצא הרשת השתנו, אבל הרעיון נותר זהה.

YOLOv2

גרסה זו משתמשת ברשת Backbone הנקראת Darknet19 ובה 19 קונבולוציות. המודל כולו כולל 22 קונבולוציות (מלבד 5 שכבות ה-MAXPOOL המקובלות על מנת למצוא את הפיצ'רים), ועוד מסלול עוקף בסמוך לסוף הרשת המחזק את יכולת העיבוד. הכותב של המאמר המקורי, רדמון, נהג לפתח גם גרסאות "Tiny" לכל מודל. הווריאנט YOLOv2 Tiny מכיל רשת Backbone קצרה יותר וללא מסלול עוקף ומבנה לינארי ופשוט מאוד. ביצועיו אמנם נמוכים יותר אך הוא מהיר מאוד (207 תמונות לשנייה לעומת 67 של מודל YOLOv2, על מעבד TitanX). מעניין לציין ש-YOLOv2 הוא המודל הראשון שאומן על תמונות בממדים משתנים, תהליך המשפר את דיוק המודל.



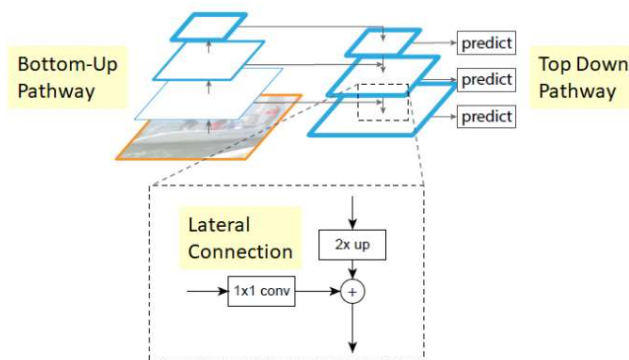
איור 9.4 ארכיטקטורת YOLOv2.

YOLOv3

כל דור נוסף של YOLO הציג חידושים ארכיטקטוניים שהגדילו את מורכבות החישוב וגודל המודל ושיפרו את ביצועיו. גרסה מספר 3 מבוססת על רשת Backbone גדולה בהרבה שנקראת Darknet53 ובה, כפי שעולה משמה, 53 קונבולוציות. כמו כן הרשת מכילה צוואר של ארכיטקטורת Feature Pyramid Network (FPN) בעלת שלושה ראשי גילוי, כאשר כל אחד מהם הוא בעל רזולוציה שונה (19 × 19, 38 × 38 ו-76 × 76).

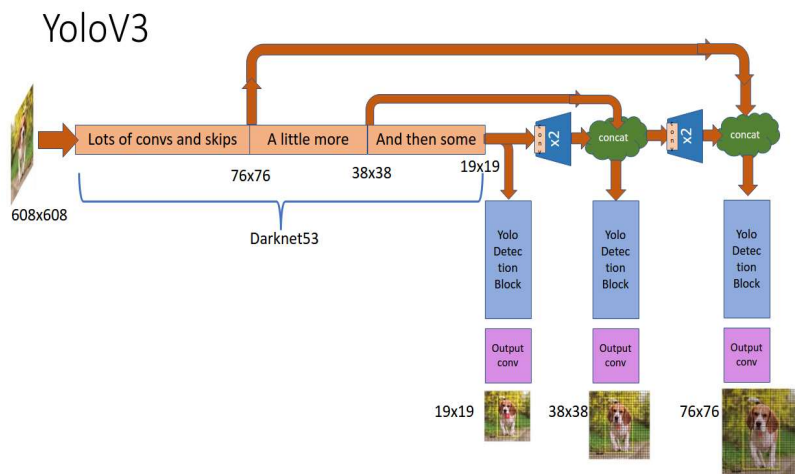
ארכיטקטורת FPN היא תוספת בקצה ה-Backbone, אשר מגדילה חזרה באופן הדרגתי את מפת הפיצ'רים. תוספת זו נועדה ליצור מסלול Top Down במקביל למסלול ה-Feed Forward ברשת ה-Backbone, שבנוי למעשה בצורת Bottom Up, בו ככל שמתקדמים בשכבות ה-Backbone מתבצע עיבוד ברמה גבוהה יותר והרזולוציה המרחבית של מפת הפיצ'רים הולכת וקטנה. בעזרת השימוש ברשת ה-FPN המודל לומד לנצל את המיטב משני עולמות: הוא משתמש במידע שטמון במפת הפיצ'רים הגדולה, שהיא אמנם פחות מעובדת אך בעלת פיצ'רים ברזולוציה מרחבית גבוהה, ובנוסף הוא מנצל גם את המידע ממפת הפיצ'רים הקטנה, שהיא אמנם בעלת פיצ'רים ברזולוציה מרחבית נמוכה, אך עם זאת היא מעובדת יותר.

לאחר כל הגדלה של מפת הפיצ'רים מתבצע חיבור בין התוצאה לבין מפת פיצ'רים קדומה יותר בממדים זהים (מתוך ה-Backbone). זאת בדומה לחיבורים העקיפים ברשת ResNet המסייעים להתכנסות האימון. השכבות השונות של רשת FPN מאפשרות לגלאי למצוא מיקום מדויק יותר של האובייקט ברזולוציות השונות, מה שמעניק לרשת זו את היכולת להבחין באובייקטים קטנים בתמונה גדולה.



איור 9.5 ארכיטקטורת Feature Pyramid Network (FPN), המשלבת מסלול top down לאחר ה-bottom up.

לראש המודל של YOLOv3 יש מספר ענפי detection, אשר כל אחד מהם פועל על מפת פיצ'רים ברזולוציה שונה ובאופן טבעי מתמחה בגילוי אובייקטים בגודל שונה (הענף בעל הרזולוציה הנמוכה מתמחה בגילוי אובייקטים גדולים, והענף בעל הרזולוציה הגבוהה מתמחה בגילוי אובייקטים קטנים).



איור 9.6 ארכיטקטורת YOLOv3.

YOLOv4

רשת YOLOv4 היא בעלת ראש זהה לזה של שתי הגרסאות הקודמות, אך ה-Backbone שונה ומורכב יותר. הוא נקרא CSPDarknet53 כאשר CSPNET היא קיצור של Cross-Stage Partial Network. רשת זו מפצלת מפות פיצורים לטובת קונבולוציה בחלקים ואיחוד מחדש. פיצול זה מאפשר, כמוזכר במאמר המקורי, חלחול טוב יותר של הגרדיאנטים בשלב האימון. בנוסף, נעשה ברשת זו שימוש בפונקציית אקטיבציה הנקראת Mish (ולא Leaky ReLU) כמו בגרסאות הקודמות).

אנקדוטה מעניינת – החל מגרסה זו התצורות אינן של היוצר המקורי ג'וזף רדמון, שהחליט לפרוש ממחקר ראייה ממוחשבת בגלל שיקולים אתיים של שימושים צבאיים או שימושים הפוגעים בפרטיות.

YOLOv5

רשת YOLOv5 מוסיפה עוד שכלולים על רשת ה-Backbone על פניו זה של הדור הקודם, ומציגה אופרטור חדש המארגן פיקסלים סמוכים בתמונת בממד הפיצורים. אופרטור זה דואג לכך שהכניסה לרשת היא לא בעומק 3 פיצורים כמקובל (RGB), אלא 12 פיצורים, תוך הקטנת הממד המרחבי. באופן זה הרשת מתאמנת לעבד תמונות ברזולוציה גבוהה, ואף לזהות אובייקטים גדולים בקלות רבה יותר, שכן שדה התמך (receptive field) של הקונבולוציות מכיל מידע משטח תמונה גדול יותר.

9.1.5 Spatial Pyramid Pooling (SPP-net)

Spatial Pyramid Pooling (SPP) הינה שכבת pooling ברשת נירונים, שמטרתה להסיר את האילוף של רשתות קונבולוציה, הדורש ששכבת הכניסה לרשת תכיל תמונה בגודל קבוע (כמו למשל רשת VGG, המקבלת רק תמונות בגודל 224×224).

כיום קיימים מכשירי צילום רבים – החל ממצלמות ניידות, מקצועיות, מצלמות אבטחה ואף מצלמות בטלפונים סלולריים ורחפנים. מצלמות שונות עשויות להוציא כפלט תמונות בגדלים שונים, ממגוון סיבות (למשל איכות התמונה או מטרת המכשיר). אם נרצה לבצע סיווג באותה רשת נירונים לכל אותן תמונות, נאלץ לבצע שלב נוסף בתחילת הדרך – התאמת התמונה בכניסה לרשת.

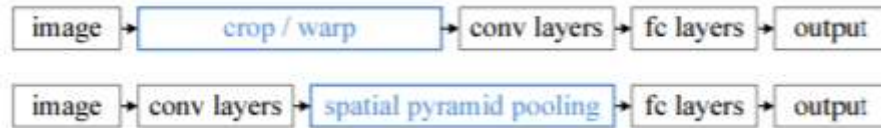
נשים לב כי על מנת להתאים תמונה כלשהי לגודל מסוים, לרוב יבוצע חיתוך (crop), או שינוי גודל (resize/wrap). פעולות אלה עלולות לפגוע בזיהוי עקב שינוי היחס בתמונה (מתיחה/כיווץ), החסרה של פרטים או שילוב של השניים. מוטיבציה זו היא שהובילה לשימוש בטכניקת SPP.

ניתן לראות דוגמה לשינוי גודל באמצעות מתיחה ולחיתוך באיור הבא:



למעשה, הדרישה לתמונת קלט בגודל קבוע בכניסה לרשתות אלה אינה הכרחית, שכן שכבות הקונבולוציה יכולות לחלץ מאפייני קלט (feature maps) בכל גודל. לעומתן, שכבות ה-FC בעומק הרשת הן השכבות שדורשות בכניסה להן קלט בגודל קבוע.

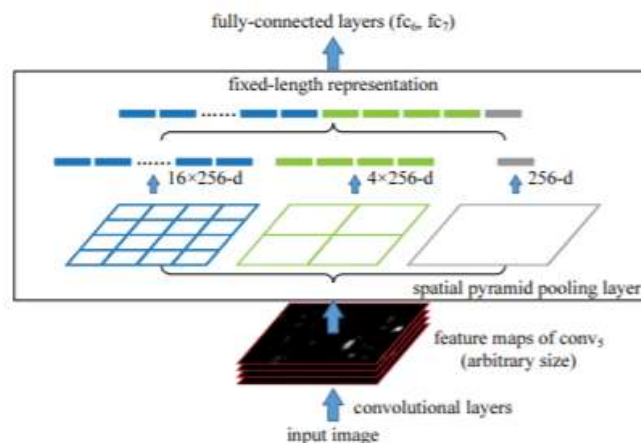
כעת, לאחר שהובהרה המוטיבציה ליצירת רשת זו, ניתן להבין את אופן פעולתה. על בסיס שכבת ה-SPP מתבססת רשת SPP-Net. החידוש במבנה הרשת, הוא שבמקום שבכניסה לרשת (לפני שכבות הקונבולוציה) תבוצע התאמת תמונת הקלט לגודל הנדרש ברשת. ההתאמה המבוצעת בשכבת SPP תבצע לאחר מציאת המאפיינים בשכבות הקונבולוציה, אך לפני שכבת ה-FC, כמתואר באיור הבא:



איור 9.8 – מבנה של רשת CNN קלאסית (למעלה) לעומת מבנה של רשת SPP-Net (למטה).

הרעיון מאחורי שכבת SPP הוא חלוקה של הפלט של שכבות הקונבולוציה, ביצוע max-pooling בכל חלק ושרשור של התוצאות לוקטור שגודלו אחיד. במילים אחרות, עבור כל ה-feature maps המתקבלים לאחר שכבות הקונבולוציה, מייצרים שלושה וקטורים לכל feature:

- וקטור בגודל 1 המתקבל באמצעות ביצוע max-pooling על כל הערכים באותו ה-feature.
 - חלוקה של ה-feature ל-4 תת-חלקים (2×2) וביצוע max-pooling בכל חלק מתוכם. מתקבל וקטור בגודל 4.
 - חלוקה של ה-feature maps ל-16 תת-חלקים (4×4) וביצוע max-pooling בכל חלק מתוכם. מתקבל וקטור בגודל 16.
 - שרשור כל הווקטורים יחד לוקטור בעל 21 ערכים.
 - בסיום התהליך, תתקבל שכבה בייצוג אחיד בגודל 21, בהכפלה במספר ה-features שהתקבלו משכבות הקונבולוציה. שכבה זו "מחליפה" את שכבת ה-pooling הממוקמת לאחר שכבת הקונבולוציה האחרונה ותוצרה הוא הקלט לשכבת ה-FC.
- ניתן לראות את מבנה הרשת באיור הבא, בו התקבלו 256 features:



איור 9.9: ארכיטקטורת SPP-Net.

9.2 Segmentation

אחד האתגרים הכי משמעותיים בעולם הראייה הממוחשבת הוא זיהוי אובייקטים בתמונה והבנת המתרחש בה. אחת הטכניקות הקלאסיות להתמודדות עם משימה זו הינה ביצוע סגמנטציה, כלומר, התאמת label לכל פיקסל בתמונה. בתהליך הסגמנטציה מבצעים חלוקה/בידול בין עצמים שונים בתמונה המצולמת באמצעות סיווג ברמת הפיקסל, כלומר כל פיקסל בתמונה יסווג וישויך למחלקה מסוימת.

ישנם שימושים מגוונים באלגוריתמים של סגמנטציה – הפרדה של עצמים מסוימים מהרקע שמאחוריהם, מציאת קשרים בין עצמים ועוד. לדוגמא, תוכנות של שיחות ועידה, כמו zoom, skype, teams וכדו', מאפשרות בחירת רקעים שונים עבור המשתמש, כאשר מלבד הרקע הנבחר רק הגוף של המשתתף מוצג בווידאו. הפרדת גוף האדם מהרקע והטמעת רקע אחר מתבצעות באמצעות אלגוריתמים של סגמנטציה. דוגמא נוספת – ניתן לזהות בתמונה אדם, כלב, וביניהם רצועה, ומכך ניתן להסיק שתוכן התמונה הוא אדם מחזיק כלב בעזרת רצועה. במקרה זה, הסגמנטציה נועדה למצוא קשר בין עצמים ולהבין את המתרחש.

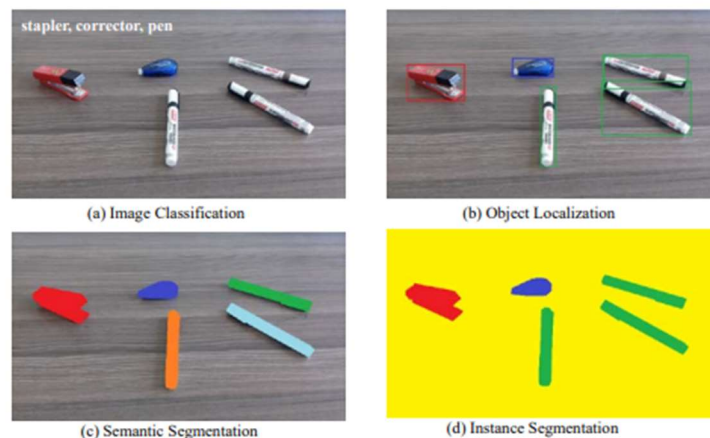
9.2.1 Semantic Segmentation vs. Instance Segmentation

קיימים שני סוגים עיקריים של סגמנטציה:

Semantic segmentation (חלוקה סמנטית) - חלוקה של כל פיקסל בתמונה למחלקה אליה העצם אותו הוא מייצג שייך. למשל, פיקסל יכול להיות משויך לכלי רכב, בן אנוש, מבנה וכדו'.

Instance segmentation (חלוקה מופעית) - חלוקה של פיקסל בתמונה למופע של אותה מחלקה אליה העצם אותו הוא מייצג שייך. במקרה זה, בתמונה בה מופיעים מספר כלי רכב, תבוצע חלוקה של כל פיקסל לאיזה כלי רכב אותו פיקסל מייצג – מכונית 1, מכונית 2, אופנוע 1, משאית 1 וכדו'.

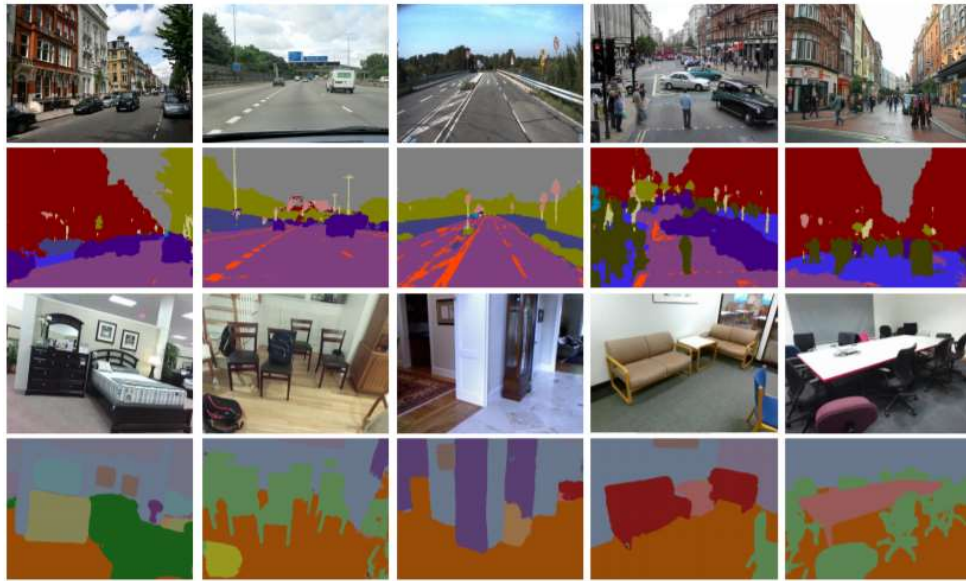
ההבדל העיקרי בין שני סוגים אלו הוא ברמת עומק המיפוי של פיקסל - המיפוי עשוי לסווג את הפיקסל למחלקה כלשהי, או לעצם ספציפי בתמונה. עומק המיפוי משליך גם על עלות המיפוי. החלוקה הסמנטית מבוצעת ישירות, בעוד שהחלוקה המופעית דורשת בנוסף ביצוע של זיהוי אובייקטים כדי לסווג מופעים שונים של המחלקות.



איור 9.7 משימות שונות תחת התחום של Computer Vision. ניתן להבחין בהבדל שבין Semantic segmentation (התאמת כל פיקסל למחלקה מסוימת) לבין Instance segmentation (התאמת כל פיקסל למופע של מחלקה מסוימת).

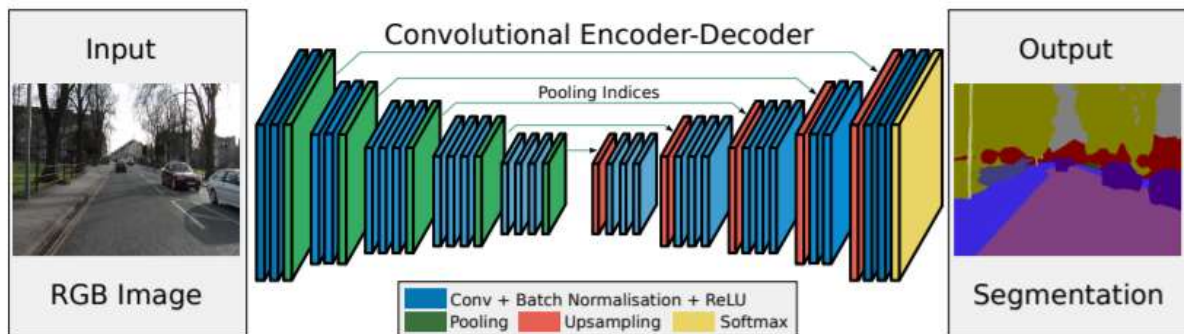
9.2.2 SegNet neural network

רשת SegNet הינה רשת קונבולוציה עמוקה, שמטרתה לבצע חלוקה סמנטית (Semantic segmentation) לתמונת הקלט. בתחילה הרשת פותחה להבנה של תמונות חוץ (למשל כביש עם מכוניות ובצדדים בתים והולכי רגל) ותמונות פנים (למשל חדר עם מיטה וכיסאות). הרשת נבנתה מתוך מטרה להיות יעילה בהיבטי זיכרון וזמן חישוב, תוך שמירה על דיוק מעשי.



איור 9.7 סיווג סמנטי בעזרת רשת SegNet עבור תמונות פנים וחוץ.

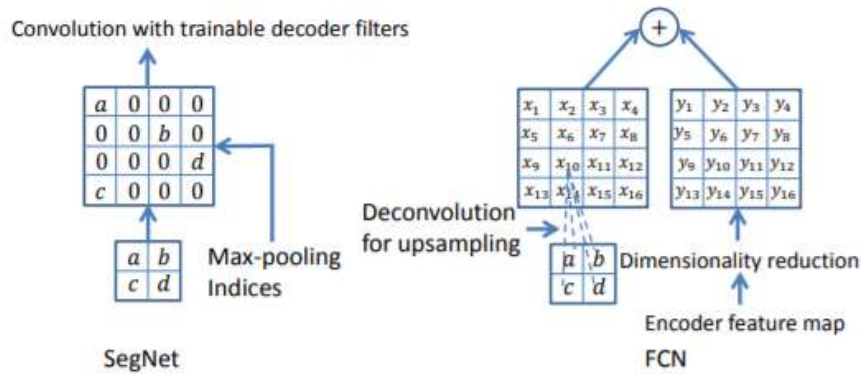
הרשת בנויה כארכיטקטורת מקודד-מפענח (encoder-decoder). המקודד מורכב מ-13 השכבות הראשונות של רשת VGG16, ומטרתו לחלץ את מפת המאפיינים (feature maps). לכל שכבת קידוד יש שכבת פיענוח תואמת ומכאן שגם רשת המפענח מכילה 13 שכבות. מטרת המפענח היא לבצע up-sampling ולייצר תמונת מאפיינים בגודל המקורי. את מוצא המפענח מעבירים דרך מסווג SoftMax, המתאים את המחלקה בעלת ההסתברות הגבוהה ביותר לכל פיקסל בנפרד.



איור 9.8 ארכיטקטורת רשת SegNet.

המקודד מורכב משכבות קונבולוציה, אחריהן שכבות נרמול (batch normalization) ושכבות אקטיבציה מסוג ReLU. לאחר שכבות אלו, ישנה שכבת max-pooling המבצעת subsampling, בעלת גודל חלון 2×2 ומרווח (stride) בגודל 2.

החידוש ברשת זו הוא אופן הפעולה של המפענח. בשונה מרשתות אחרות, בהן תהליך ה-up-sampling גורר ביצוע חישובים עבור הפיענוח, כמתואר באיור הבא, הרעיון ברשת זו הוא שמירת מיקומי ערכי המקסימום הנבחרים מכל רביעייה. רק הערכים שנבחרו כמקסימום ישוחררו בתהליך ושאר הערכים יתאפסו.



איור 9.9 שכבת פענוח ברשת SegNet לעומת רשת FCN.

ארכיטקטורה זו מביאה את הרשת לביצועים טובים בהיבטי זמן חישוב, על חשבון פגיעה מסוימת בדיוק הרשת. למרות זאת, ביצועי הרשת מתאימים לשימושים פרקטיים והפגיעה בדיוק קטנה מאוד.

בדומה למקודד, לאחר כל שכבת up-sampling במפענח, יופיעו שכבות קונבולוציה, שכבות נרמול ושכבות אקטיבציה מסוג ReLU. את מוצא המפענח מעבירים דרך שכבת SoftMax המבצעת סיווג ברמת הפיקסל. מוצא הרשת, שהוא גם מוצא שכבת ה-SoftMax, הינו מטריצת הסתברויות, כאשר עבור כל פיקסל יש וקטור באורך K , כאשר K הוא מספר המחלקות לסיווג. כמובן שהסיווג מתבצע בהתאמה להסתברות הגבוהה ביותר המתאימה לכל פיקסל.

אימון הרשת בוצע על בסיס מידע קטן יחסית של 600 תמונות דרך צבעוניות בגודל 360×480 , שנלקחו מבסיס המידע CamVid road scene dataset. סט האימון הכיל 367 תמונות וסט הבדיקה הכיל את 233 התמונות הנותרות. המטרה הייתה לזהות בתמונות אלה 11 מחלקות (דרך, בניין, מכונית, הולכי רגל וכדומה). כל תמונה עברה נרמול מקומי לערך הניגודיות של תמונת הקלט לפני הכניסה לרשת. האימון בוצע בשיטת ה-SGD, עם קצב למידה קבוע שערכו 0.1 ומומנטום שערכו 0.9. האימון נמשך עד ששיגאת האימון התכנסה. לפני כל epoch סט האימון עורבב וחולק ל-mini-batch של 12 תמונות. פונקציית המחיר הינה cross-entropy.

לעיתים, נדרש לבצע איזון-מחלקות (class balancing). מונח זה מתאר מצב בו קיים שוני גדול בין כמות הפיקסלים המשויכים לכל מחלקה, למשל כאשר קיימת הטיה מסוימת - סצינה שברובה מכילה בניינים / דרכים. במצב זה, יבוצע משקול מחדש לפונקציית השיגאה, באמצעות תהליך "איזון התדירות החציונית" (median frequency balancing). התהליך ממשקל מחדש את המחלקות בפונקציית המחיר, באופן יחסי לחציון של תדירות הופעות המחלקות בכל סט האימון, תוך חלוקה בתדירות הופעת המחלקה:

$$\alpha_c = \frac{\text{median freq}}{\text{freq}(c)}$$

משקול זה משנה את היחסים בפונקציית המחיר כך שהתרומה של כל המחלקות לפונקציית המחיר תהיה שווה. לכן, הוא מעניק למחלקות הגדולות יותר משקל נמוך יותר ולמחלקות הקטנות משקל גבוה יותר.

9.2.3 Atrous Convolutions (Dilated Convolutions)

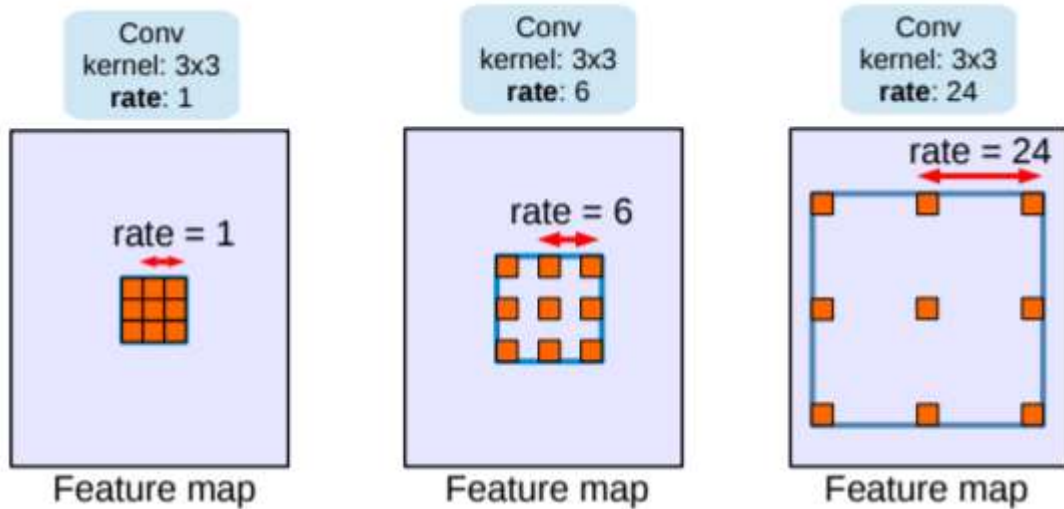
המונח Atrous מקורו בשפה הצרפתית – "à trous", שמשמעותו "עם חורים". לכן, ניתן לתרגם את המונח Atrous convolution כ-"קונבולוציה מחוררת", ובמשמעות מעט יותר מתאימה – קונבולוציה מרווחת (או בשם אחר – Dilated convolution – קונבולוציה מורחבת).

בטכניקת קונבולוציה זו, יש שימוש פרמטר נוסף בנוסחת הקונבולוציה – dilation rate. פרמטר זה מסמן את המרווח בין כל איבר בגרעין הקונבולוציה (הרחבה על פרמטר זה – פרק 5.1.2). נוסחת הקונבולוציה עבור המקרה החד ממדי יחד עם פרמטר ההתרחבות r ניתנת לתיאור באופן הבא:

$$y[i] = \sum_{k=1}^K x[i + r \cdot k]w[k]$$

עבור $r = 1$ מתקבלת הקונבולוציה הרגילה, וכאשר ישנו dilation של $r > 1$, מתקבלת קונבולוציה מרווחת בפקטור r . יתרונה של קונבולוציה נעוץ בכך שעבור אותו קרנל ועבור אותה כמות חישובים, מרחיבים את ה-field of view (FoV) של הקונבולוציה.

ניתן לראות את הרחבת ה-field of view באיור הבא:



איור 9.10: קונבולוציה מרווחת דו-ממדית, עם קרנל בגודל 3×3 ופרמטר התרחבות $r = 1, 6, 24$. בהתאם לכל פרמטר מתקבל field-of-view בגודל שונה – $3 \times 3, 16 \times 16, 49 \times 49$ בהתאמה.

9.2.4 Atrous Spatial Pyramidal Pooling

9.2.5 Deeplab V3+

ניקח רגע צעד אחורה בעת שנגדיר את שלוש הבעיות המרכזיות של שימוש ברשתות קונבולוציה עמוקות (Deep convolutional neural networks) במסגרת Semantic segmentation:

1. **צמצום ברזולוציית הפיצ'רים** - קיימות שתי סיבות מרכזיות לתופעה זאת. הסיבה הראשונה היא שאימון של Semantic segmentation מורכב ויקר בהרבה מאשר קלסיפיקציה לתמונה. לכן, נעדיף להשתמש ב-Transfer learning מאשר ברשת שאומנה זה מכבר ובעלת ידע קודם. יש לכך יתרונות וחסרונות, היתרון העיקרי הוא הפחתה המשמעותית בעלויות. לעומת זאת, אחד החסרונות הוא שרוב הרשתות אומנו על Imagenet-dataset שמכיל תמונות ואובייקטים רבים, אך התמונות בו בעלות רזולוצייה נמוכה, מה שעלול להוביל לרזולוציית פיצ'רים נמוכה.

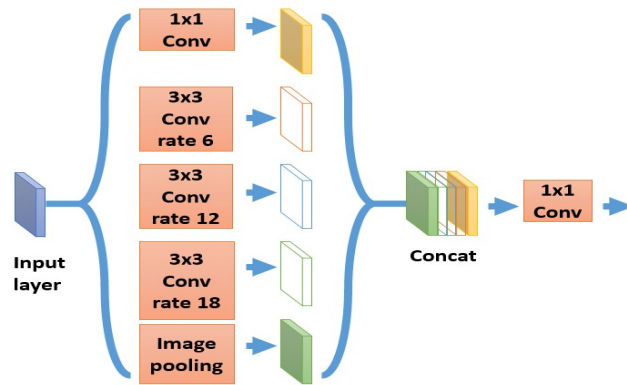
הסיבה השנייה היא שכאשר אנחנו משתמשים ברשתות קונבולוציה עמוקות אנחנו מנסים לקבל הבנה לגבי האובייקט שמסתתר לנו בתמונה. על ידי זיקוק האובייקט מתוך התמונה הגדולה נוצרת באופן אינהרנטי ירידה ברזולוציית הפיצ'רים.

2. **הימצאות של אובייקטים בגדלים שונים** – ניקח לדוגמה תמונה בה מופיעה מכונית גדולה מאוד ומכונית קטנה. במקרים מסוג זה, הרשת תתקשה לשייך את שני האובייקטים הללו לאותה קטגוריה.

3. **הפחתה בדיוק המקומי בגלל השונות של רשתות קונבולוציה עמוקות** - במילים אחרות, רשתות קונבולוציה עושות עבודה מעולה בלזהות דפוסים, אך יש פה סוג של trade off - הרשת תזהה את הדפוס הכללי אבל "תשכח" איפה בדיוק מופיע האובייקט בתמונה.

כותבי המאמר של ה-Deeplab V3+ תיארו שיטות להתמודדות עם הבעיות שתיארנו. הם השתמשו בשתי ארכיטקטורות מרכזיות, חיברו אותן ועם מספר שיפורים יצרו רשת חזקה עם תוצאות יפות על ה-datasets המרכזיים.

נדון כעת בשתי השיטות המרכזיות שהופיעו במאמר -



איור 1 - (ASPP) Atrous Spatial Pyramid Pooling

השיטה הראשונה היא **קונבולוציה מרווחת** (Atrous convolution) כפי שנכתב לעיל, מדובר בכלי מאוד חזק. כלי זה מאפשר לנו ללכוד מידע קונטקסטואלי עשיר ביותר ע"י איגוד של פיצורים בגדלים שונים. כאשר אנו עושים שימוש במרווחים שונים אנו מקבלים ראייה מרחבית משתנה ולא קבועה, איגוד התמונות לתמונה אחת מאפשר לרשת כוח אדיר של הבנה. באמצעות זאת, הרשת מסוגלת ליצור קשרים בין אובייקטים שונים ואף בין אותו אובייקט שמופיע בגדלים שונים בתמונה.

האינטואיציה שעומדת מאחורי ASPP היא שכאשר אנו עושים קונבולוציה רגילה אנחנו בוחרים להתרכז בחלק מאוד ספציפי (בגודל קבוע) בתמונה. באופן זה נקבל את הפיצורים מאותו החלק אבל ההקשר הסביבתי יהיה מוגבל. זאת לעומת שימוש ב- ASPP אשר מקנה הקשר סביבתי שונה וגדול יותר (בגלל הפרמטר של המרווח המשתנה). כלומר, כאשר מאגדים את התמונות לתמונה אחת אנחנו מחזקים את ההבנה הגלובלית שיש לרשת על התמונה. זאת לאור העובדה שכבר לא מדובר בהסתכלות על תמונה אחת אלא על הרבה זוויות שונות שעוזרת לרשת להבין קשרים והקשרים בצורה יותר ברורה. נאיין באיור 2 שלפנינו -

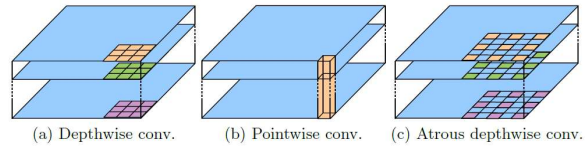


איור 2 – דוגמה לשימוש בשיטה

אם נרצה שהרשת "תסתכל" רק על הכלי המופיע בריבוע הצהוב היא תחשוב שאולי מדובר במכונית. אך ברגע שהרשת "תתפוס" את המים מסביב ואת התמונה הרחבה יותר -- היא תקבל הבנה עמוקה יותרו תבין שמדובר בסירה ולא במכונית. דבר דומה יתרחש כאשר יהיה מופע בתמונה של אותו אובייקט בגדלים שונים: האיגוד של התמונות עם המרווחים השונים עוזר לרשת לקשר בין אותו האובייקט.

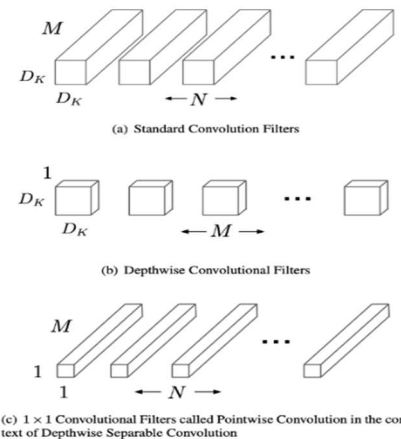
מבחינה טכנית, הפעולות שאנו מבצעים ב- ASPP על התמונה מורכבות מ- 4 קונבולוציות מרווחות. בכל אחת מהן אנו מקבלים תמונה באותו הגודל, אבל בעלת ראייה מרחבית שונה. לאחר כל קונבולוציה נבצע נרמול ונפעיל פונקציית אקטיבציה (Relu).

בהמשך לביצוע הקונבולוציות, נבצע פעולת Image pooling על התמונה. מימושים שונים למאמר השתמשו ב- Adaptive average pooling. לאחר פעולת ה-pooling, נבצע bilinear interpolation על מנת שנוכל לחזור לגודל התמונה ולבצע שרשרת בין כל התמונות.



איור 3 - Atrous Separable Convolution

בהיבט של עלויות חישוב, כותבי המאמר משתמשים בטכניקת Atrous Separable Convolution שמטרתה להוריד בצורה ניכרת את מספר הפרמטרים והחישובים שהרשת צריכה לבצע.



איור 4 - Separable convolution

שיטת ה- Separable convolution מכילה בתוכה שתי טכניקות עיקריות – Depthwise convolution ו- Pointwise convolution.

נסביר אותן בקצרה: ב-depthwise conv אנחנו מבצעים את הפעולות על מרחב מימד התמונה (ציר ה x ו-y) וב-pointwise conv הפעולות מתבצעות על מרחב הערוץ.

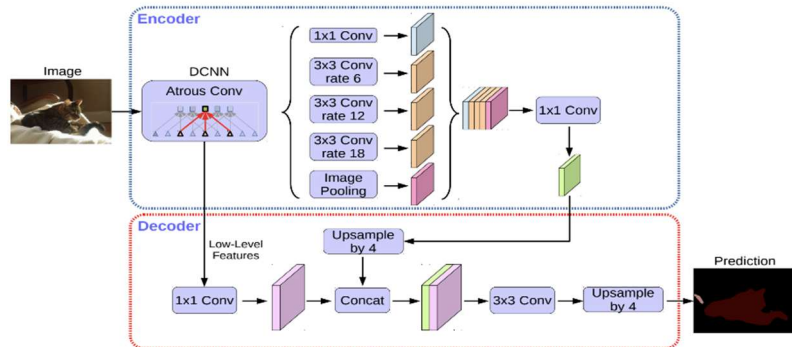
מבחינה מתמטית, נניח שה- input feature maps בגודל $F \in D_F \times D_F \times M$ ושה- output feature maps בגודל $G \in D_F \times D_F \times N$. במידה והיינו משתמשים בקונבולוציה רגילה, ה- convolution kernel צריכה להיות בגודל $K \in D_K \times D_K \times M \times N$ ולכן עלות החישוב תהיה $D_F \times D_F \times D_K \times D_K \times M \times N$.

אבל אם נבחר להשתמש ב- separable convolution, ראשית נבצע depthwise conv, בה פילטר הקונבולוציה הוא מימד $K \in D_K \times D_K \times M$. כלומר, במצב זה אנחנו פועלים רק על מימד התמונה ולא על מימד הערוץ. לכן העלות תהיה $D_F \times D_F \times D_K \times D_K \times M$. לאחר מכן נבצע pointwise conv בה אנחנו פועלים במימד הערוץ. פילטר הקונבולוציה במקרה זה יהיה בגודל $K \in 1 \times 1 \times M$ ולכן עלות החישוב תהיה $D_F \times D_F \times M \times N$. אם נחבר את שניהם עלות החישוב תהיה:

חיסכון משמעותי בעלויות החישוב ובמספר הפרמטרים. ניתן לראות שבניגוד לקונבולוציה הרגילה, במצב זה נוצר

כאשר נשתמש בקונבולוציה מרווחת נקבל atrous separable convolution.

ארכיטקטורת מקודד מפענח (encoder decoder)



איור 5 - ?

נביט על ארכיטקטורת המקודד ונראה שכותבי המאמר למעשה העתיקו את המקודד שבו השתמשו ברשת Deeplab V3 המקודד מחולק ל-3 חלקים:

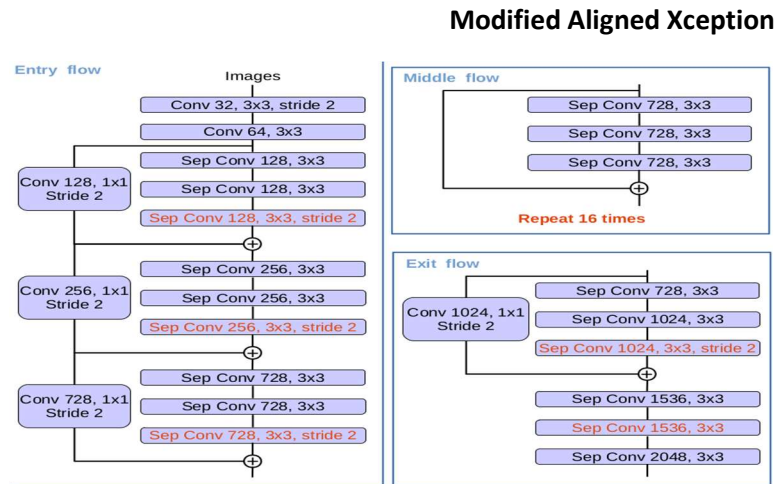
1. רשת backbone שמטרתה לחלץ פיצ'רים מהתמונה - במאמר משתמשים בוורסיה של Aligned Xception (נרחיב עליה בהמשך). בנוסף לכך רשת ה-backbone מחזירה גם low level feature שאותו היא מקבלת לאחר ה-block הראשון של ה-entry flow.
2. לאחר מכן אנו מבצעים ASPP על התמונה.
3. לבסוף ביצוע קונבולוציה 1X1 על מנת לעבור למספר הערוצים הרצוי.

לעומת זאת המפענח עבר שינוי דרסטי מכיוון שכותבי המאמר ראו שהשחזור של התמונה לגודל המקורי לוקה בחסר ופוגע ביכולת הדייק של הרשת, שלוש הטכניקות שבהן הם השתמשו:

1. שימוש ב-low level feature לביצוע השחזור.
2. שימוש ב-upsample בהדרגה.
3. הפעלת קונבולוציה 3X3 על מנת לקבל אובייקטים חדים וברורים יותר.

מבחינה טכנית הסכמה של המפענח מתבצעת כך:

- מתקבלת תמונה מהמקודד ועליה ניתן לבצע upsample פי 4 (לדוגמה עם התמונה היא בגודל 32X32 היא עכשיו תהיה בגודל 128X128).
- לאחר מכן, יש לבצע קונבולוציה 1X1 על הקלט שהקבל מה-low level feature על מנת להקטין את גודל מימד הערוץ.
- לאחר מכן, יש לבצע שרשור של התמונה עם ה-low level feature.
- יש לבצע קונבולוציה 3X3 כדי לחזק את האובייקטים שבתמונה בנוסף, מספר הערוצים שבתמונה ישתנה למספר הקטגוריות שיש לנו ב-dataset.
- לבסוף יש לבצע upsample נוסף לתמונה בגודל פי 4 על מנת לחזור לגודל המקורי של התמונה.



איור 6- Modified Aligned Xception

כותבי המאמר בחרו להשתמש ברשת זו כ- backbone בגלל ש- Xception ידועה בביצועים טובים ומהירים במשימת הקלסיפיקציה ועם עלויות מאוד נמוכות. כמו כן, Aligned Xception הוכיחה את עצמה בביצועים טובים במשימת ה- object detection. כותבי המאמר ביצעו מספר שינויים קלים ברשת שיתאימו ספציפית למשימה שלנו.

- במקום max pooling או מבצעים atrous separable convolution עם stride 2.
- לאחר כל ביצוע של 3x3 depthwise conv או נבצע נרמול ואקטיבציה.

ביצועים

Deeplab V3+ מצליחים לייצר state of the art חדש גם ב- PASCAL VOC 2012 Test Set עם Miou 89 וגם ב- Cityscapes dataset עם Miou 82.1.

9.3 Face Recognition and Pose Estimation

9.3.1 Face Recognition

אחד מהיישומים החשובים בראייה ממוחשבת הינו זיהוי פנים, כאשר ניתן לחלק משימה זו לשלושה שלבים:

1. Detection – מציאת הפרצופים בתמונה.
2. Embedding – מיפוי כל פרצוף למרחב חדש, בו המאפיינים שאינם קשורים לתיאור הפנים (למשל: זווית, מיקום, תאורה וכדו') אינם משפיעים על הייצוג.
3. Searching – חיפוש במאגר של תמונות למציאת תמונת פנים הקרובה לתמונת הפנים שחולצה מהתמונה המקורית.

גישה פשטנית, כמו למשל בניית מסווג המכיל מספר יציאות כמספר הפנים אותם רוצים לזהות, הינה בעייתית משתי סיבות עיקריות: ראשית יש צורך באלפי דוגמאות לכל אדם (שלא ניתן בהכרח להשיג). כמו כן, נצטרך ללמד את

המערכת מחדש בכל פעם שרוצים להוסיף מישהו חדש. כדי להתגבר על בעיות אלו מבצעים "למידת מטריקה" (metric learning) בה מזקקים מאפיינים של פנים ויוצרים וקטור יחסית קצר, למשל באורך 128, המכיל את האלמנטים המרכזיים בתמונת הפנים. כעת נפרט את שלושת השלבים:

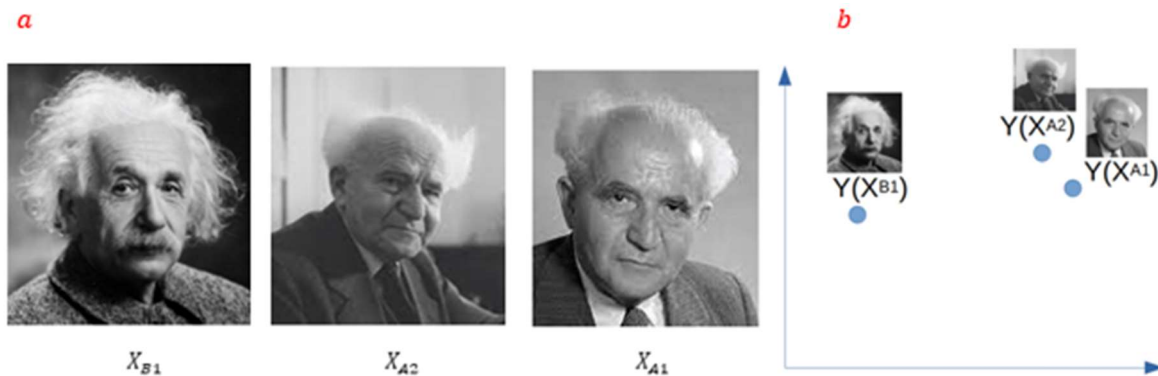
1. מציאת פנים.

כדי למצוא פרצופים בתמונה ניתן להשתמש ברשתות המבצעות detection, כפי שתואר בפרק 9.1. שיטה מקובלת למשימה זו הינה Yolo, המבוססת על חלוקת התמונה למשבצות, כאשר עבור כל משבצת בוחנים האם יש בה אובייקט מסוים, מהו אותו אובייקט, ומה ה-bounding box שלו.

2. תיאור פנים.

כאמור, המשימה בתיאור פנים נעשית בעזרת metric learning, כאשר הרעיון הוא לזקק פנים לוקטור שאינו מושפע ממאפיינים שלא שייכים באופן מהותי לפנים הספציפיות האלה, כגון זווית צילום, רמת תאורה וכדו'. בכדי לעשות זאת יש לבנות רשת המקבלת פנים של בנאדם ומחזירה וקטור, כאשר הדרישה היא שעבור שתי תמונות של אותו אדם יתקבלו וקטורים מאוד דומים, ועבור פרצופים של אנשים שונים יתקבלו וקטורים שונים. למעשה, פונקציית ה-loss תקבל בכל פעם minibatch, ותעניש בהתאם לקרבה בין וקטורים של אנשים שונים וריחוק בין וקטורים של אותו אדם.

כעת נניח שיש לנו קלט X , המכיל אוסף פרצופים. כל איש יסומן באות אחרת – A, B, C, ותמונות שונות של אותו אדם יסומנו על ידי אות ומספר, כך שלמשל X_{A1} זוהי התמונה הראשונה של אדם A בסט הקלט X , וכמובן ש- X_{A1} ו- X_{A2} הן שתי תמונות של אותו אדם. באופן גרפי, בדו-ממד ניתן לתאר זאת כך (בפועל הווקטורים המייצגים פנים יהיו בממד גבוה יותר):



איור 9.10 (a) דוגמאות מסט הפרצופים X . (b) איך נרצה שהדאטה ימופה לממד חדש Y .

כאמור, נרצה לבנות פונקציית loss שמעודדת קירבה בין X_{A2} ו- X_{A1} , וריחוק בין X_{B1} ו- X_{A1} . פונקציית ה-loss מורכבת משני איברים, המודדים מרחק אוקלידי בין וקטורים שונים:

$$L = \sum_X \left(\|Y(X^{Ai}) - Y(X^{Aj})\| - \|Y(X^{Ai}) - Y(X^{Bj})\| \right)$$

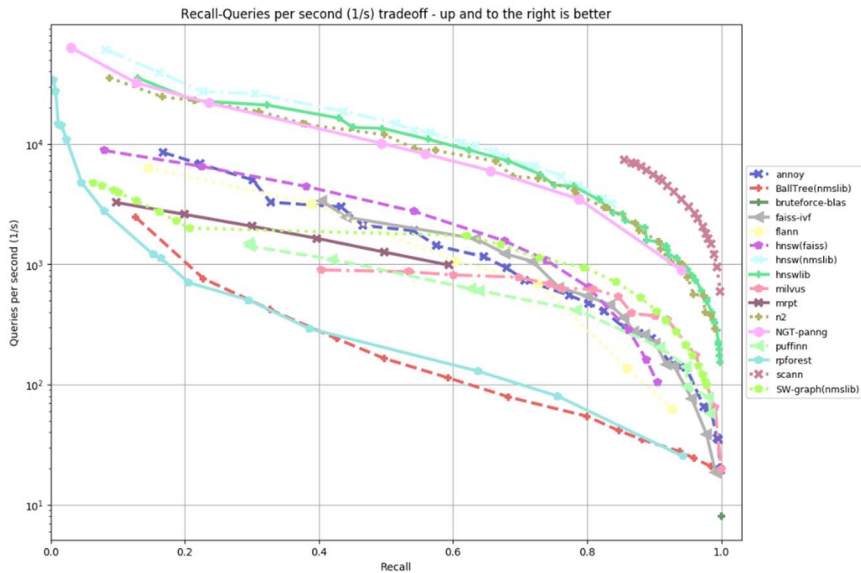
כאשר האיבר הראשון ינסה להביא למינימום וקטורים של אותו אדם, והאיבר השני ינסה להביא למקסימום וקטורים של פרצופים שאינם שייכים לאותו אדם. כיוון שנרצה להימנע מקבלת ערכים שליליים, נוסיף פונקציית מקסימום. בנוסף, ניתן 'להרחיק' תוצאות של פרצופים שונים על ידי הוספת קבוע k , כך שהפרש בין המרחק של פרצופים של אנשים שונים לבין המרחק של פרצופים של אותו איש יהיה לפחות k :

$$L = \sum_X \max(\|Y(X^{Ai}) - Y(X^{Aj})\| - \|Y(X^{Ai}) - Y(X^{Bj})\| + k, 0)$$

loss כזה נקרא triplet loss, כיוון שיש לו שלושה איברי קלט – שתי תמונות של אותו אדם ואחת של מישהו אחר. כאמור, הפלט של הרשת הנלמדת צריך להיות וקטור המאפיין פנים של אדם, ומטרת הרשת היא למפות פרצופים שונים של אותו אדם לווקטורים דומים עד כמה שניתן, ואילו פרצופים של אנשים שונים יקבלו וקטורים רחוקים זה מזה.

3. מציאת האדם

בשלב הקודם, בו התבצע האימון, יצרנו למעשה מאגר של פרצופים במרחב חדש. כעת כשיגיע פרצוף חדש, כל שנותר זה למפות אותו למרחב החדש, ולחפש במרחב זה את הווקטור הקרוב ביותר לוקטור המייצג את הפנים החדשות. בכדי לעשות זאת ניתן להשתמש בשיטות קלאסיות של machine learning, כמו למשל חיפוש שכן קרוב (כפי שהוסבר בחלק 2.1.3). שיטות אלו יכולות להיות איטיות עבור מאגרים המכילים מיליוני וקטורים, וישנן שיטות חיפוש מהירות יותר (ובדרך כלל המהירות באה על חשבון הדיוק). בעזרת השיטה המובילה כרגע (SCANN) ניתן להגיע לכמה מאות חיפושים שלמים בשנייה (החיפוש ב-100 ממדים מתוך מאגר של 10000 דוגמות).



איור 9.11 השוואת ביצועים של שיטות חיפוש שונות. עבור פרצוף נתון, מחפשים עבורו וקטור תואם בממד החדש המכיל ייצוג וקטורי של הפרצופים הידועים. בכל שיטה יש טרייד-אוף בין מהירות החיפוש לבין הדיוק.

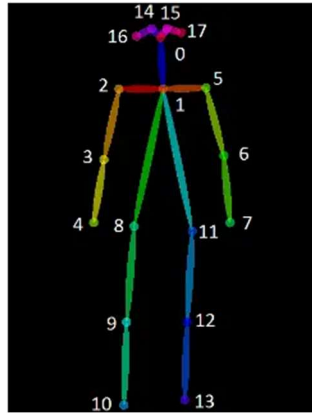
מלבד זיהוי וסיווג פנים, יש גם שיטות של מציאת אלמנטים של פנים הכוללות אף, עיניים וכו'. אחת השיטות המקובלות משתמשת בשערוך הצורה של פנים אנושיות, וניסיון למצוא את איברי הפנים לפי הצורה הסטנדרטית. בשיטה זו ראשית מבצעים יישור של הפנים והתאמה לסקאלה אנושית (על פי מרחק בין האיברים השונים בפנים), ולאחר מכן מטילים 68 נקודות עניין מרכזיות על התמונה המיושרת, מתוך ניסיון להתאים בין הצורה הידועה לבין התמונה המבוקשת.



איור 9.12 זיהוי אזורים בפנים של אדם על ידי התאמת פנים לסקאלה אנושית והשוואה למבנה של פנים המכיל 68 נקודות מרכזיות.

9.3.2 Pose Estimation

יישום פופולרי נוסף של אלגוריתמים השייכים לראייה ממוחשבת הינו קביעת תנוחה של אדם – האם הוא עומד או יושב, מה התנוחה שלו, באיזה זווית האיברים נמצאים וכו'. ניתן להשתמש בניתוח התנוחה עבור מגוון תחומים – ספורט, פיזיותרפיה, משחקים שונים ועוד. לרוב, תנוחה מיוצגת על ידי המיקומים של חלקי גוף עיקריים כגון ראש, כתפיים, מרפקים וכו'. ישנם כמה סטנדרטים מקובלים, למשל COCO, COCO pose net ועוד. ב-COCO התנוחה מיוצגת בעזרת מערך של 17 נקודות (בדו-מימד):



איור 9.13 מיפוי תנוחה ל-17 נקודות מרכזיות.

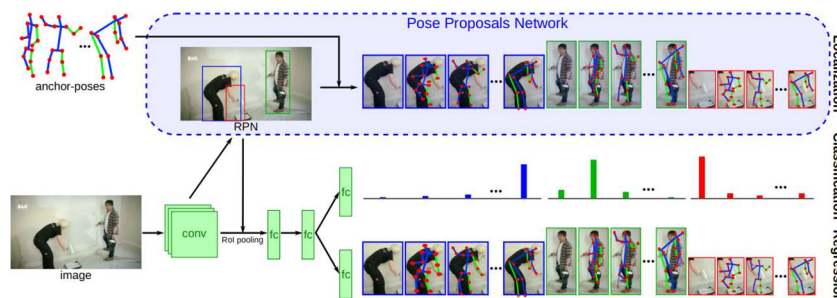
שאר הפורמטים דומים; מוסיפים עוד מידע (למשל מיקום הפה), משתמשים בתלת ממד במקום בדו ממד, משתמשים בסידור אחר וכדו'. כאשר רוצים לאסוף נתונים על מנחי גוף שונים, ניתן לשים חיישנים על אותן נקודות מרכזיות וככה לקבל מידע על מנח הגוף לאורך זמן.

רשת לצורך קביעת תנוחה יכולה לטפל במקרה כללי, בו יש מספר אנשים בתמונה, או במקרה הפרטי של גוף יחיד. המקרה השני כמובן יותר פשוט, מכיוון שישנו פלט יחיד, אותו ניתן לחזות באמצעות רגרסיה (למשל, 34 מספרים שמתארים את המיקומים של 17 הנקודות בפורמט COCO בדו-ממד). במקרה זה ניתן לעשות למידה סטנדרטית לחלוטין של בעיית רגרסיה בעלת 34 יציאות.

ישנן מספר גישות כיצד להכליל את הרשת כך שתוכל לטפל גם במקרה הכללי בו יש יותר מגוף אחד בתמונה. באופן נאיבי ניתן לבצע תהליך מקדים של מציאת כל האנשים בתמונה, ואז להפעיל על כל אחד מהם בנפרד את הרשת שמבצעת רגרסיה, כפי שתואר לעיל. שיטה נוספת פועלת בכיוון הפוך – ראשית כל הרשת מוצאת את כל האיברים בתמונה נתונה, ולאחר מכן משייכת אותם לאנשים שונים. השיטה השנייה נקראת "מלמטה למעלה" (bottom-up), כיוון שקודם כל היא מוצאת את הפרטים ולאחר מכן מכלילה אותם. גישה זו יעילה למקרה בו יש הרבה חפיפה בין האנשים בתמונה, כיוון שאין לה צורך לבצע תהליך מקדים של הפרדת האנשים. השיטה הראשונה, הנקראת "מלמעלה למטה" (top-down), תהיה פשוטה יותר עבור מקרים בהם אין חפיפה בין האנשים בתמונה וכל אחד מהם נמצא באזור שונה בתמונה, כיוון שבמקרה זה אין צורך לשייך איברים לאנשים.

רשת פופולרית לקביעת תנוחה נקראת Multi-person pose estimation, המורכבת למעשה משתי תת-רשתות ופועלת בשיטת bottom-up. הרשת שאחראית על שיוך חלקי גוף לאדם מסוים, נקראת (PAF) part affinity fields, והרעיון שלה הוא לייצג כל איבר כשדה וקטורי. בייצוג זה הווקטורים השונים מצביעים לכיוון איבר הגוף ה'בא בתור' (למשל זרוע מצביעה ליד), וככה ניתן לשייך איברים שונים אחד לשני, ואת כל יחד לגוף מסוים.

רשת פופולרית אחרת, הפועלת בגישת top-down, נקראת LCR-NET, והיא מבוססת על רעיון של 'מיקום-סיווג-רגרסיה' (Localization-Classification-Regression). בשלב הראשון יש תת-רשת המייצרת עוגנים עבור אנשים, כלומר, אזורים בהם הרשת חושבת שנמצא בן אדם, ולאחר מכן הרשת משערכת את התנוחות שלהם. בשלב השני מתבצע clustering לכל העוגנים, כלומר כל עוגן מקבל ציון המייצג את טיב השערוך של העוגן והתנוחה של האדם הנמצא בתוכו. השלב השלישי מלטש את העוגנים ומשקלל את השערוך הסופי בעזרת מיצוע של הרבה עוגנים. שלושת השלבים משתמשים ברשת קונבולוציה משותפת, כמתואר באיור.



איור 9.14 Localization-Classification-Regression.

9.4 Few-Shot Learning

יכולת הצלחתם של אלגוריתמי למידה עמוקה נשענת על כמות ואיכות הדאטה לאימון. עבור משימת סיווג תמונות (Image Classification), נדרש שעבור כל קטגוריית סיווג תהיה כמות גדולה של תמונות מגוונות (עם הבדלי רקעים, בהירות, זוויות וכו'), ובנוסף יש צורך בכמות דומה של דוגמאות בכל קטגוריות הסיווג. חוסר איזון בין כמות התמונות בקטגוריות השונות משפיע על יכולת הלמידה של האלגוריתם את הקטגוריות השונות ועל כן עלול ליצור הטיה בתוצאות הסיווג לטובת הקטגוריות להן יש יותר דוגמאות. בפרק זה נעסוק בשיטות כיצד ניתן להתמודד עם מצבים בהם הדאטה אינו מאוזן.

9.4.1 The Problem

התחום של למידה ממיעוט דוגמאות (Few-Shot Learning) נוצר על מנת להתמודד עם מצב של חוסר איזון קיצוני בין כמות הדוגמאות של כל קטגוריה לאימון הרשת. באופן פורמלי, קיימות קטגוריות הנקראות קטגוריות בסיס (base classes), עבורן יש כמות גדולה של דוגמאות, ובנוסף ישנן קטגוריות חדשניות (novel classes), עבורן יש כמות קטנה מאוד של דוגמאות. בכדי להגדיר את היחס, משתמשים בשני פרמטרים: פרמטר k המייצג את מספר ה-shots, כלומר מספר הדוגמאות הקיימות בסט האימון מכל קטגוריה חדשנית, ופרמטר n שמייצג את מספר הקטגוריות החדשניות הקיימות סך הכל. כל בעיה מוגדרת על ידי "k-shot n-way", ולמשל "one-shot 5-way learning" מתאר מצב בו יש חמש קטגוריות חדשניות, ומכל אחת מהן יש רק דוגמא אחת לאימון הרשת. ככלל, בקטגוריות הבסיס תהיה כמות גדולה של דוגמאות. למשל בסט התמונות האופייני לבעיות אלו, mini-ImageNet, יש 600 דוגמאות לכל קטגוריית בסיס ולרוב 1-5 דוגמאות עבור הקטגוריות החדשניות.

האתגר בלמידה ממיעוט דוגמאות נובע מהצורך להכניס לרשת כמות ידע קטנה נוספת על הידע הנרחב הקיים, תוך הימנעות מ-overfitting כתוצאה מכמות הפרמטרים הגדולה של הרשת לעומת הכמות המועטה של הדאטה. לכן, גישה נאיבית כמו אימון מחדש של רשת על מעט דוגמאות נוספות עלולה ליצור הטיה בתוצאות.

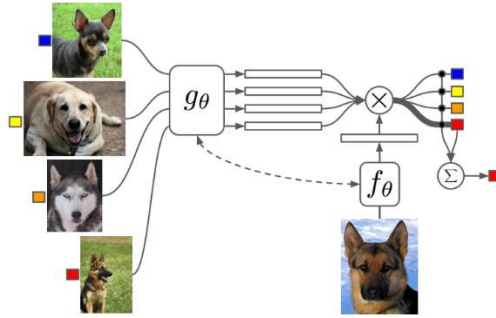
יש לציין כי בכל בעיות הלמידה ממיעוט דוגמאות, אמנם חסר דאטה עבור האימון, אך השאיפה היא להצליח באופן זהה בזיהוי כל הקטגוריות בשלב המבחן, בו לא יהיה חוסר איזון. לכן בעיות אלו רלוונטיות לשימושים רבים כמו: זיהוי חיות נדירות באופן זהה לחיות יותר נפוצות, מערכת זיהוי טילים שצריכה להתמודד גם עם איזמים נדירים יותר (ניתן לחשוב למשל על פצצת אטום), מערכות זיהוי פנים שצריכות לעבוד טוב עבור כל אדם ללא תלות בדאטה שהיה קיים באימון הרשת.

בפרק זה נתאר את שלוש הגישות העיקריות לפתרון בעיות למידה ממיעוט דוגמאות. עבור כל גישה נציג את האלגוריתמים המשמעותיים ביותר שנקטו בגישה זו. לאחרונה, מפותחים יותר ויותר אלגוריתמי למידה ממיעוט דוגמאות שמשלבים יחד רעיונות השאובים ממספר גישות יחד אך נשענים על האלגוריתמים המשמעותיים מהעבר. לבסוף, נציג את התחום של Zero-Shot Learning, כלומר יכולת למידה של קטגוריה חדשה כאשר לא קיימת אף דוגמא שלה לאימון.

9.4.2 Metric Learning

שיטות להתמודדות עם למידה ממיעוט דוגמאות הנוקטות בגישת למידת מטריקה, שואפות לייצג את הדוגמאות כווקטורים של מאפיינים במרחב רב-ממדי, כך שניתן יהיה למצוא בקלות את השיוך הקטגורי של דוגמא חדשה, גם אם היא תהיה מקטגוריה חדשנית. שיטות אלו מבוססות על עיקרון הגדלת המרחק בין ייצוגים וקטורים של דוגמאות מקטגוריות שונות (inter-class dissimilarity), בד בבד שמירה על מרחק קטן בין הייצוג הווקטורי של דוגמאות מאותה הקטגוריה (intra-class similarity).

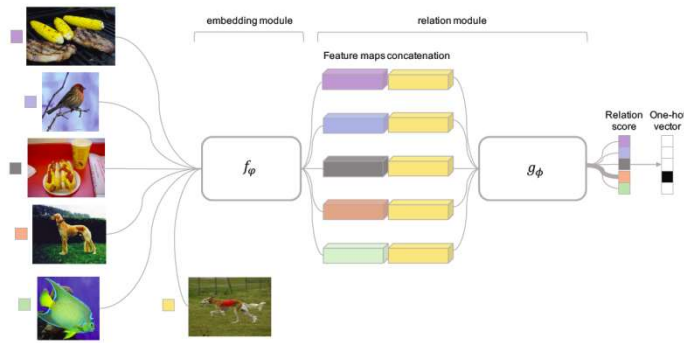
התקדמות משמעותית של שיטות אלו הוצגה במאמר Matching Networks for One Shot Learning בשנת 2016. שיטה זו משתמשת בזיכרון שהגישה אליו נעשית באמצעות מנגנון Attention, על מנת לחשב את ההסתברות של דוגמא להיות שייכת לכל קטגוריה, בדומה לשיטות השכן הקרוב (Nearest Neighbors).



איור 9.15 אילוסטרציה של שיטת Matching Networks.

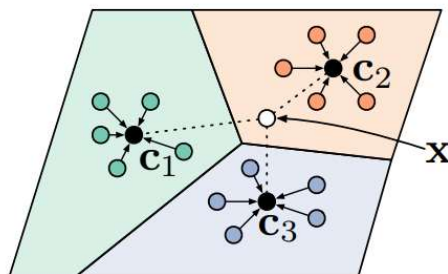
החידוש המשמעותי בשיטת Matching Networks נעוץ בשיטת האימון המבוצעת באפיזודות (Episodes). בשיטה זו האימון מכיל כמות של משימות, כאשר כל משימה היא למעשה מדגם של הדאטה שבו יש קטגוריות מסוימות שהן חדשניות ואחרות שהן קטגוריות בסיס. על ידי דגימות רבות ויצירת משימות מלאכותיות כאלו, בהן בכל פעם נלקחות קטגוריות אחרות לייצג את החדשניות, מתבצע אימון המתאים לבעיה של מיעוט דוגמאות. שיטת אימון באפיזודות הפכה לנפוצה ביותר בלמידה ממיעוט דוגמאות, גם בגישת המטריקה וגם בגישות שנראה בהמשך.

שיטות רבות מתבססות על הרעיונות של מאמר זה. למשל שיטת Relation Network משרשרת וקטורי מאפיינים של דוגמת מבחן לבין כל דוגמא של קטגוריות האימון. אלו נכנסים למודל המשערך מדד דמיון בעזרתו ניתן לסווג את דוגמת המבחן.



איור 9.16 Learning to Compare: Relation Network for Few-Shot Learning.

שיטה משמעותית נוספת הנוקטת בגישת למידת מטריקה נקראת Prototypical Networks. בגישה זו כל קבוצת דוגמאות של קטגוריה מסוימת במרחב וקטורי המאפיינים מקבלת נקודת אב-טיפוס אופיינית המחושבת על ידי הממוצע של הדוגמאות בקטגוריה זו. בכך מחשבים מסווג לינארי המפריד בין הקטגוריות. בעת המבחן נסווג דוגמא חדשה על סמך מרחק אוקלידי מנקודות האב-טיפוס.



איור 9.17 Prototypical Networks for Few-Shot Learning.

בטבלה הבאה ניתן לראות השוואת ביצועים של שיטות למידת המטריקה שהוזכרו על הקטגוריות החדשניות. יש להדגיש כי כל השיטות מגיעות לאחוזי דיוק נמוכים משמעותית מאחוזי הדיוק המדווחים במקרים של איזון בין כמות הדוגמאות בקטגוריות השונות (לרוב מעל 90% דיוק).

Method	5-way 1-Shot	5-way 5-Shot
Matching Networks	46.6%	60.0%
Prototypical Networks	49.42%	68.20%
Learning To Compare	50.44%	65.32%

איור 9.18 השוואת ביצועי דיוק של שיטות למידת מטריקה על קטגוריות חדשניות עבור mini-ImageNet.

9.4.3 Meta-Learning (Learning-to-Learn)

גישה שניה להתמודדות עם מיעוט דוגמאות וחוסר איזון בין הקטגוריות נקראת מטא-למידה (או: ללמוד איך ללמוד). באופן כללי בלמידת מכונה, כאשר מדובר על מטא-למידה, מתכוונים לרשת שלומדת על סמך התוצאות של רשת אחרת. בלמידה ממיעוט דוגמאות הרעיון הוא שהרשת תלמד בעצמה איך להתמודד עם מיעוט הדאטה על ידי עדכון הפרמטרים שלה לאופטימיזציה של בעיה של סיווג ממיעוט דוגמאות. לשם כך משתמשים באפיזודות של משימות למידה ממיעוט דוגמאות.

שיטה חשובה בגישה זו היא Model-Agnostic Meta-Learning (MAML). בשיטה זו, שאינה מיועדת ספציפית לסיווג תמונות ממיעוט דוגמאות, בעזרת מספר צעדים מעטים בכיוון הגרדיאנט ניתן ללמד את הרשת התאמה מהירה (fast adaptation) למשימה חדשה. כזכור, כל משימה באימון היא אפיזודה שבה קטגוריות מסוימות נבחרות רנדומלית לדמות את הקטגוריות החדשניות. בכל משימה כזו נלמדים פרמטרים של המודל האגנוסטי כך שעדכונם בכיוון הגרדיאנט יוביל להתאמה למשימה החדשה. הכותבים מציינים שמנקודת מבט של מערכות דינאמיות, ניתן להתבונן על תהליך הלמידה שלהם ככזה שממקסם את רגישות פונקציית המחיר של משימות חדשות ביחס לפרמטרים. כאשר הרגישות גבוהה, שינויי פרמטרים קטנים יכולים להוביל לשיפור משמעותי במחיר של המשימה. מתמטית, פרמטרי המודל, המיוצגים על ידי θ , משתנים עבור כל משימה T_i להיות θ'_i , כאשר:

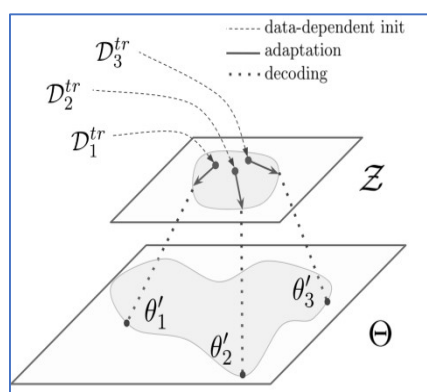
$$\theta'_i = \theta - \alpha \nabla_{\theta} L_{T_i}(f_{\theta})$$

עבור פונקציית מחיר L והפרמטר α . כאשר מבצעים מטא-למידה לעדכון הפרמטרים, מחשבים למעשה SGD:

$$\theta \leftarrow \theta - \beta \nabla_{\theta} \sum_{T_i \sim p(T)} (L_{T_i}(f_{\theta}))$$

כאשר המשימות נדגמות מתוך $p(T)$ ו- β הוא גודל הצעד של המטא-למידה.

שיטה מעניינת נוספת בשם Latent Embedding Classification (LEO) מיישמת את הגישה של מטא-למידה במרחב ייצוגי לטנטי בעל ממדים נמוכים, ולאחר מכן עובר בחזרה למרחב המאפיינים הרב ממדי.



איור 9.19 שיטת Latent Embedding Classification (LEO).

השימוש במרחב מאפיינים בעל ממדים נמוכים המשמרים את המאפיינים החשובים לייצוג הקטגוריות, שיפר באופן ניכר את תוצאות הסיווג, כפי שניתן לראות בטבלה הבאה:

Method	5-way 1-Shot	5-way 5-Shot
MAML	48.7%	63.11%
LEO	61.76%	77.59%

איור 9.20 השוואת ביצועי דיוק של שיטות מטא-למידה על קטגוריות חדשניות עבור mini-ImageNet.

9.4.4 Data Augmentation

גישה שונה להתמודדות עם מיעוט דוגמאות נוקטת ביצירת דוגמאות כדי להימנע מהטיה. שיטות אוגמנטציה למעשה יוצרות דאטה חדש על סמך הדאטה הקיים. השיטות הפשוטות יותר מייצרות מהתמונות הקיימות תמונות ראי, שינויי תאורה וקונטרסט, שינויי סקאלה, שינויי זוויות, ואף הוספת רעש רנדומלי. כל אלו הראו שיפורים ביכולות הרשתות ללמוד קטגוריות שהיו במצב של חוסר איזון. דרך נוספת היא שימוש ברשתות גנרטיביות (GANs) על מנת ליצור דוגמאות רלוונטיות, למשל דוגמאות של אותו האובייקט מזוויות שונות. שיטה מעניינת של אוגמנטציות היא CutMix, בה פאצ'ים של תמונות נחתכים ומודבקים בתמונות האימון וגם התיוגים מעורבבים בהתאם. שיטה זו הגיעה לביצועים מרשימים בסיווג תמונות וגם בזיהוי אובייקטים, ככל הנראה בגלל שהיא מאפשרת למודל להיות גנרי יותר בהתייחסות לחלקים שונים מהתמונה המשפיעים על הסיווג לקטגוריה.

9. References

Detection:

<https://arxiv.org/pdf/1406.4729.pdf>

Segmentation:

<https://arxiv.org/ftp/arxiv/papers/2007/2007.00047.pdf>

SegNet:

<https://arxiv.org/pdf/1511.00561.pdf>

<https://mi.eng.cam.ac.uk/projects/segnet/#demo>

<https://arxiv.org/pdf/1409.1556.pdf>

<https://arxiv.org/pdf/1502.01852.pdf>

Face recognition:

https://docs.opencv.org/master/d2/d42/tutorial_face_landmark_detection_in_an_image.html

<http://blog.dlib.net/2014/08/real-time-face-pose-estimation.html>

10. Natural Language Processing

עיבוד שפה טבעית (NLP) הוא תחום העוסק בפיתוח אלגוריתמים לעיבוד וניתוח של דאטה טקסטואלי. המטרה העיקרית היא לפתח שיטות ומודלים שיאפשרו למכונות "להבין" את התוכן הטקסטואלי, ואת הניואנסים וההקשרים של השפה. עולם ה-NLP מורכב ממספר רב של תתי משימות, כגון ניתוח סנטימנט של טקסט (Sentiment analysis), תמצות/סיכום אוטומטי של טקסט (Text summarization), מציאת תשובה עבור שאלה נתונה (Question answering) ועוד משימות רבות אחרות. פיתוח כלים המסוגלים להתמודד עם משימות אלה יאפשר לנו (בין היתר) לפתח אפליקציות שיעזרו לנו ביום יום כגון עוזרות קוליות, מערכות תרגום, מערכות אוטומטיות לבדיקת דקדוק ועוד. כמה דוגמאות לאפליקציות כאלה שכולנו מכירים הן Siri, העוזרת הקולית של אפל, ההשלמה האוטומטית בתיבת החיפוש ב-Google ו-Grammarly, מערכת לתיקון תחבירי לטקסט באנגלית.

10.1 Language Models and Word Representation

בפרקים הקרובים נדון בשניים מהנושאים הבסיסיים ביותר בתחום ה-NLP – מודלי שפה וייצוגי מילים: נראה כיצד מייצרים מודל שפה מדאטה סט טקסטואלי (שנקרא גם "Corpus"), וכיצד מייצגים את הטקסט כך שיהיה אפשר לאמן בעזרתו מודל. כדי להקנות למכונה יכולת הבנה של דאטה טקסטואלי יש לבנות מודל הסתברותי הקובע את ההתפלגות של המילים בשפה. המודל מנסה לכמת את הסיכוי להופעה של סדרות שונות של מילים, ובאופן יותר כללי הוא קובע מה ההסתברות של כל רצף אפשרי להופיע. מודל כזה אפשר לבנות בעזרת אימון על גבי דאטה טקסטואלי, והוא נקרא "מודל שפה" (Language Model). לפני ביצוע האימון, יש לבצע מיפוי של הטקסט לייצוג מסוים (Word Representation), המאפשר למכונה לקחת את הדאטה הקיים, לעבד אותו ולנסות לבנות בעזרתו את מודל השפה.

ראשית נגדיר מהו מודל שפה: מודל שפה הינו מודל סטטיסטי המגדיר התפלגות מעל המרחב המורכב מכל הסדרות האפשריות של מילים (כלומר משפטים, פסקאות וכדומה). מודל זה מקבל כקלט סדרה של מילים ותפקידו הוא לחזות מה היא המילה הבאה שתניב את ההסתברות המרבית לרצף ביחד עם המילה הנוספת.

כעת נתאר בצורה מתמטית מהו מודל שפה. נניח ונתון משפט עם n מילים: $[w_1, \dots, w_n]$, אז ההסתברות לקבל את המשפט הזה הינה:

$$P(w_1, \dots, w_n) = \prod_{i=1}^n P(w_i | w_1, \dots, w_{i-1})$$

ניקח למשל את המשפט הבא:

Take a big corpus

ההסתברות של משפט זה ניתנת לחישוב אופן הבא:

$$P(\text{Take, a, big, corpus}) = P(\text{Take}) \cdot P(\text{a}|\text{Take}) \cdot P(\text{big}|\text{Take, a}) \cdot P(\text{corpus}|\text{Take, a, big})$$

במילים, נוכל לתאר את המשוואה הזו כך: ההסתברות לקבל את המשפט הנתון שווה להסתברות של המילה Take להופיע כפול ההסתברות של המילה a להופיע אחרי המילה Take, כפול ההסתברות של המילה big להופיע אחרי הצירוף Take a, כפול ההסתברות של המילה corpus להופיע אחרי הצירוף Take a big.

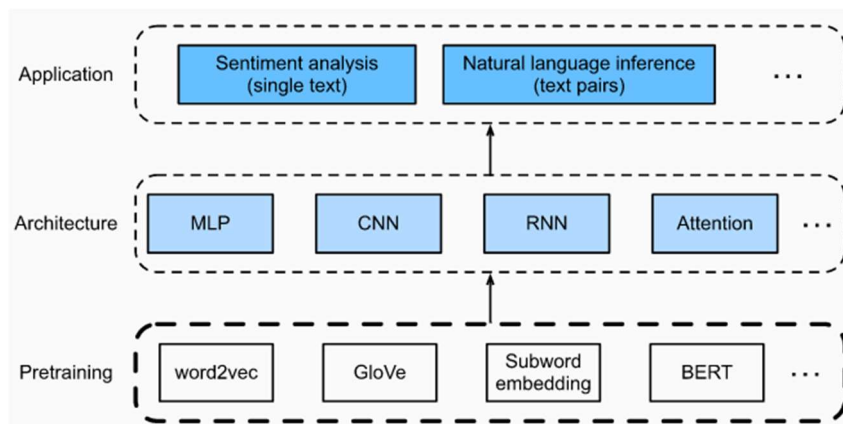
באופן הפשטני ביותר נוכל לשערך את ההסתברויות האלו באופן הבא: ניקח corpus מספיק גדול ועשיר בעל N מילים שונות, כמו למשל כל ערכי ויקיפדיה, ופשוט נספור את כל המילים והצירופים שמופיעים בו. ההסתברות של כל מילה תהיה אחוז הפעמים שהיא מופיעה בטקסט, וההסתברות המותנית תחושב באופן דומה – על ידי מניית מספר המופעים של צירוף כלשהו וחלוקה במספר הפעמים שהמילה עצמה מופיעה. באופן פורמלי נוכל לרשום זאת כך:

$$P(W_i) = \frac{\#W_i}{N}, P(W_i|W_j) = \frac{\#W_i, W_j}{\#W_i}$$

בין אם נחליט לקחת את מודל השפה הזה או שנייצר מודלים מתוחכמים יותר כפי שנראה בהמשך, המודל הוא אחד הדברים היסודיים ביותר במשימות שפה, כיוון שבעזרתו ניתן לבצע מגוון משימות.

כאמור, בכדי שנוכל לבנות מודל שפה או לאמן כל מודל אחר, נצטרך קודם כל לייצג את הטקסט בצורה כלשהיא. בשיטה שהוצגה לעיל, כל מילה (Token) מיוצגת באמצעות צירוף אותיות. כך למשל המילה הראשונה במשפט

שראינו מורכבת מצירוף של האותיות T, a, k, e. כפי שיפורט בהמשך, נוכל גם לדבר על ייצוג למילים עצמן כך שכל יחידה אטומית תהיה מילה וצירוף של היחידות האלה ירכיבו ייצוג של משפט. באופן סכמתי, ניתן לתאר את משימת עיבוד השפה מקצה לקצה באופן הבא:



איור 11.1 תהליך פיתוח אלגוריתם של מודל שפה: א. מייצגים את הטקסט בצורה כלשהיא (ניתן כמובן לקחת ייצוג קיים שנבנה על בסיס דאטהסט אחר). ב. מאמנים מודל שמקבל כ-input את הטקסט אותו ייצגנו בדרך כלשהיא בשלב הראשון, והמודל מוציא output מסוים. למודל כזה יכולות להיות ארכיטקטורות שונות. ג. באמצעות המודל המאומן ניתן לבצע משימות קצה שונות.

בהמשך פרק זה נתמקד בשכבה התחתונה של התרשים: נתאר מספר שיטות מרכזיות לייצוג טקסטים, ונראה כיצד ניתן לאמן מודלי שפה שונים היכולים לבצע כל מיני משימות.

10.1.1 Basic Language Models

מודל השפה הראשון אותו נציג הינו N-Grams, שהינו מודל סטטיסטי המניח שהסתברות למילה הבאה תלויה אך ורק ב-1 N המילים שקדמו לה בסדרה. הנחה זאת נקראת "הנחת מרקוב" (Markov assumption), ובאופן כללי יותר, מודלי מרקוב (או שרשראות מרקוב במקרה הדיסקרטי) מסדר n הם מודלי הסתברות המניחים שניתן לחזות הסתברות של אירוע עתידי T בהתבסס על האירועים שהתרחשו ב- n נקודות הזמן שקדמו למאורע T מבלי להתחשב באירועי עבר רחוקים מדי.

מודל ה-N-Gram הפשוט ביותר נקרא unigram. במודל זה אנחנו חוזים את המילה הבאה לפי התדירות של המילה עצמה ב-corpora מבלי להתחשב במה שקדם לה. כמובן שחיוזי כזה הינו בעייתי, מכיוון שהמילה הבאה **חייבת להיות תלויה במילים שקדמו לה**, וכמו כן יהיו מילים כמו the שמופיעות באופן תדיר בטקסט שאינן בהכרח משפיעות על ההקשר. לכן, נסתכל על מודל קצת יותר מורכב הנקרא bigram, המתייחס למילה האחרונה הקודמת למילה הנחזית. במודל bigram אנחנו מניחים את המשוואה הבאה:

$$P(w_n | w_{n-1}, w_{n-2}, \dots, w_1) = P(w_n | w_{n-1})$$

כלומר, רק למילה האחרונה יש השפעה על החיוזי של המילה הבאה, וכל המילים שלפניה הן חסרות השפעה על ההתפלגות של המילה הנחזית (וממילא גם על המשך המשפט). באופן כללי, מודל N-grams מניח את המשוואה הבאה:

$$P(w_n | w_{n-1}, w_{n-2}, \dots, w_1) = P(w_n | w_{n-1}, w_{n-2}, \dots, w_{n-N}), N \leq n$$

ניקח לדוגמא מספר משפטים וננתח אותם בשיטת bigram:

- I know you
- I am happy
- I do not know Jonathan

נניח ונרצה לבחון את ההסתברות שהמילה Jonathan היא המילה הבאה אחרי הסדרה I do not know. ראשית נגדיר את המילון, המכיל את כל המילים האפשריות בשפה:

$$V = \{I, know, you, am, happy, do, not, Jonathan\}$$

כעת נוכל להעריך את ההסתברות לכך על ידי ספירת כמות הפעמים שהצמד $\langle know|Jonathan \rangle$ מופיע בטקסט, ולנרמל בכמות הפעמים ש-know מופיע בטקסט עם מילה כלשהי (כולל הפעמים שמופיע עם Jonathan). באופן פורמלי נגדיר את המשוואה הבאה:

$$P(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n)}{\sum_{w \in V} C(w_{n-1}w)}$$

כאשר האות C מסמנת את מספר הפעמים שצמד מסוים בטקסט. ניתן לשים לב שהביטוי במכנה למעשה סופר את כמות הפעמים ש- w_{n-1} קיים בטקסט, ולכן נוכל לפשט את המשוואה האחרונה ולרשום במקומה:

$$P(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n)}{C(w_{n-1})}$$

אם כך, ההסתברות לכך שהמילה Jonathan תופיע אחרי המילה know היא:

$$P(Jonathan|know) = \frac{1}{2}$$

באופן דומה ניתן לנסות לשערך את ההסתברות של המילה הבאה על סמך יותר ממילה אחת אחורה, למשל לקחת 2 מילים אחורה. מודל זה נקרא trigram, וכפי שנאמר לעיל, באופן כללי מודל המסתמך על $N - 1$ מילים לצורך חישוב ההסתברות של המילה הבאה בטקסט נקרא N-gram.

המודל המרקובי סובל ממספר בעיות:

1. טבלת ההסתברויות המתקבלת מאוד דלילה. כיוון שה-corpus (שהוא סט האימון) הינו בגודל מוגבל, לעולם לא נוכל לראות את כל הקומבינציות הקיימות, מה שיוביל להערכה של הסתברות 0 עבור צמדים שלא מופיעים בטקסט האימון.

2. כאשר נרצה לחזות בעזרת מודל זה את ההסתברויות על טקסט חדש, כנראה שנתקל במילים שלא נתקלנו בהן בטקסט האימון ולכן לא נוכל להגדיר את ההסתברות עבור ה-N-Grams המכילים מילים אלו.

בשביל להתמודד עם בעיות אלו – באופן מלא או חלקי – נוכל להפעיל שיטות החלקה (Smoothing) על ההסתברויות של צמדים שהופיעו מעט/כלל לא הופיעו בטקסט האימון. שיטות החלקה במהותן לוקחות קצת מסת הסתברות מהאירועים השכיחים (כלומר, ה-N-Grams המופיעים הכי הרבה) ו"תורמות" לאירועים פחות שכיחים. ישנן מגוון שיטות להחלקת הסתברות ונסקור כעת חלק מהן.

Laplace Smoothing

הדרך הפשוטה ביותר לבצע החלקה היא להוסיף מספר קבוע \mathcal{K} לכל האירועים. באופן הזה אנו דואגים לתת משמעות גם לצירופים נדירים שמופיעים מספר מועט של פעמים (או אפילו לא מופיעים בכלל). אם למשל יש צירוף שמופיע רק פעם אחת ולעומתו יש צירוף שמופיע 1000 פעמים, אז הוספת הקבוע \mathcal{K} משמעותה שעכשיו נמנה את הצירוף הראשון ככזה המופיע $(1 + \mathcal{K})$ פעמים וביחס לצירוף השני נתייחס ככזה שהופיע $(1000 + \mathcal{K})$ פעמים. הוספה זו כמעט ואינה משפיעה על הצירוף השני, אך היא יכולה להכפיל פי כמה את ההסתברות של הצירוף הראשון. כמובן שכאשר אנחנו מתעסקים עם הסתברויות נרצה לאחר הוספת הקבוע במונה לתקן גם את מקדם הנרמול. באופן פורמלי, החלקת לפסל מוגדרת באופן הבא:

$$P_{Laplace}(w_n|w_{n-1}) = P_{Add-\mathcal{K}}(w_n, w_{n-1}) = \frac{C(w_{n-1}w_n) + \mathcal{K}}{\sum_w C(w_{n-1}w) + \mathcal{K}} = \frac{C(w_{n-1}w_n) + \mathcal{K}}{C(w_{n-1}) + \mathcal{K} \cdot |V|}$$

כאשר $|V|$ הוא גודל המילון (כמות המילים המופיעה ב-Corpus). היתרון בשיטה זאת היא הפשטות שלה, אך עם זאת יש לה חסרון בולט הנובע מכך שהיא משנה באופן משמעותי את ההסתברויות של מאורעות נדירים ובכך בעצם משבשת את ההסתברויות שנלמדו מהטקסט. כפועל יוצא יותר מדי מסת הסתברות עוברת מהמאורעות השכיחים למאורעות עם הסתברות נמוכה. השיטה הבאה מנסה לתקן בדיוק את זה.

כמובן שניתן לבחור כל ערך \mathcal{K} שרוצים ואותו להוסיף למכנה. באופן זה כן ניתן לשלוט ברמה מסוימת עד כמה להגדיל את ההסתברויות לקומבינציות שאינן מופיעות בסט האימון על חשבון הקומבינציות השכיחות (כתלות ב- \mathcal{K}). בחירה למשל של $\mathcal{K} = 0.5$ מאפשרת למזער את העיוות היכול להתקבל משינוי הספירה.

Backoff and Interpolation

שיטת ההחלקה בסעיף הקודם נותנת פתרון לקביעת ההסתברות של מאורעות המקבלים הסתברות 0, וישנן גישות נוספות שנותנת מענה למגבלות האלה. נניח ואנחנו משתמשים במודל trigram, מודל המניח שההסתברות למילה הבאה תלויה בשתי המילים שבאות לפניה. אם נבצע את השערוך של כל שלישייה לפי הגדרה (=ספירת המופעים שלהם בטקסט), כל שלישיית מילים שאינה מופיעה כרצף בטקסט תקבל הסתברות 0, ובעצם תקיים את המשוואה:

$$P(w_n | w_{n-1}, w_{n-2}) = 0$$

כדי להימנע מלתת הסתברות 0 בכזה מצב, ניתן להיעזר גם בהסתברויות של bigram וה-unigram:

$$P(w_n | w_{n-1}), P(w_n)$$

שיטת ה-backoff מציעה לקחת את כל המקרים בהם קיבלנו 0 ולשערך אותם מחדש באמצעות מודל bigram. לאחר מכן ניקח את כל המילים שעדיין נותרו עם הסתברות 0 (כלומר, צמדי מילים שאינן מופיעים בטקסט) ונשערך אותם באמצעות unigram. באופן הזה מקווים להגיע למצב בו מספר המילים בעלי הסתברות 0 היא קטנה (עד אפסית), כיוון ששימוש במודלים יותר פשוטים מאפשר שימוש במגוון רחב יותר של קומבינציות הקיימות בטקסט. שיטה דומה נקראת Interpolation, ובה במקום לקחת את הרצפים בעלי הסתברות 0 ולשערך אותם במודל הנמצא בדרגה אחת מתחת (למשל – לשערך בעזרת bigram במקום trigram), המודל מראש מתייחס לקומבינציה כלשהי של ה-unigram, bigram and trigram (למשל קומבינציה לינארית). בשיטה זו גם מקרים שאינם בעלי הסתברות 0 נעזרים ב-unigram and bigram לשערוך ההסתברות ובניית מודל השפה.

10.1.2 Word representation (Vectors) and Word Embeddings

עד כה, המילים השונות היו מיוצגות בעזרת אותיות. כך לדוגמא, המילה כלב תיוצג על ידי צירוף האותיות DOG בעוד שהמילה חתול תיוצג על ידי הצירוף CAT. ייצוג זה מכיל מאפיינים סינטקטיים (=תחביריים) של השפה, קרי איך המילה נכתבת. עם זאת, ייצוג זה חסר מאוד, כיוון שהוא יתקשה בלמידת ייצוג של מאפיינים סמנטיים. דוגמא להבנה סמנטית של שפה היא ההבנה שכלב וחתול הן לא מילים נרדפות, אך הן כן קשורות אחת לשנייה באופן כלשהו – שתיהן מייצגות חיות מחמד שאנשים מגדלים בביתם. מודל המבוסס על ייצוג טקסטואלי של השפה לא יצליח להגיע להבנה סמנטית שלה, ולכן נרצה שהייצוג שלנו יהיה מספיק עשיר ויכיל הן מאפיינים תחביריים והן מאפיינים סמנטיים של השפה.

בפועל, נוכל למפות את הייצוג הטקסטואלי לייצוג נומרי בצורה פשוטה בעזרת One-Hot vectors – מערך בגודל המילון שלנו, המייצג כל מילה במילון בעזרת 1 באיבר המתאים במערך. לדוגמא נתון המילון הבא:

Index	Word
0	Dog
1	Cat
2	Lion

נוכל לייצג את המילים השונות בעזרת וקטורים באורך 3, באופן הבא:

$$\text{Dog} \rightarrow [1, 0, 0]$$

$$\text{Cat} \rightarrow [0, 1, 0]$$

$$\text{Lion} \rightarrow [0, 0, 1]$$

כך, לכל מילה יהיה ייצוג וקטורי ייחודי. עם זאת, ייצוג פשוט זה הינו בעייתי מכיוון שיהיה קשה ללמוד ממנו מאפיינים סמנטיים. כדי להבין את הסיבה לכך ראשית יש להגדיר את מושג הדמיון בעולם של וקטורים.

Cosine similarity

מעבר לייצוג וקטורי של מילים דורש מאיתנו להגדיר דמיון בין וקטורים במרחב שנוצר. אחת מההגדרות הפופולריות לדמיון בין וקטורים היא ה-Cosine similarity. כמו רוב השיטות לחישוב דמיון בין וקטורים, גם פונקציית דמיון זו מבוססת על מכפלה פנימית של וקטורים. נניח ונתונים 2 וקטורים v, w , שניהם בעלי אותו הממד N . המכפלה הפנימית (dot product) בין הווקטורים האלה מוגדרת באופן הבא:

$$v \cdot w = v^T w = \sum_{i=1}^N v_i w_i$$

באמצעות הגדרה זו ננסה לאמוד את הדמיון בין הייצוג של המילים כלב וחתול במרחב הווקטורי שנוצר בעקבות ייצוג בעזרת One-Hot vectors. כאמור, הייצוג של המילה חתול במרחב זה הוא: $Cat \rightarrow [0, 1, 0]$, בעוד שהייצוג של המילה כלב באותו מרחב הינה: $Dog \rightarrow [1, 0, 0]$. לכן, לכן המכפלה הפנימית במרחב זה תהיה:

$$[0, 1, 0] \cdot [1, 0, 0] = 0 \cdot 1 + 1 \cdot 0 + 0 \cdot 0 = 0$$

תוצאה זו מדגימה את הבעייתיות בייצוג פשוטי זה, מכיוון שהינו רוצים שווקטורים אלה כן יהיו דומים במובן מסוים, ולא שהתוצאה תהיה 0, המלמדת על כך שכלל אין קשר בין שתי המילים. למעשה בשיטה זו הדמיון בין ייצוגים של כל זוג מילים יהיה אפס.

Bag-of-words

דרך אחת לייצר קשר בין מילים בעלות קשר סמנטי היא להתייחס לא רק למילים בודדות אלא גם להקשרים בתוך המשפט עצמו. באופן הזה נוכל להגדיר ייצוג וקטורי של מילה על ידי ספירה של כמות הפעמים שמילה אחרת נמצאת איתה באותו ההקשר. שיטה זאת נקראת Bag-of-Words, ולפני שנוכל להדגים אותה, נצטרך להגדיר מהו הקשר של מילה. באופן פשוט הקשר של מילה זה המשפט בו היא מופיעה, אך ניתן גם לקחת רק חלון של מספר מילים מתוך המשפט (לרוב אורך החלון קטן מאורך המשפט). לדוגמא, נניח ונתונים לנו הטקסט והמילון הבאים:

טקסט:

- [The dog is a domestic mammal, not wild mammal], is a domesticated descendant of the wolf, characterized by an upturning tail.
- [The cat is a domestic species of small mammal], It is the only domesticated species in the family.

מילון:

1. The	5. Mammal	9. Animal	13. Tail
2. Is	6. Not	10. Descendant	14. Cat
3. A	7. Natural	11. Dog	15. Species
4. Domestic	8. Wild	12. Wolf	16. Small

כעת נרצה לייצג את המילים Dog ו-Cat בשיטת Bag-of-words באמצעות חלון באורך 7 (=כמות המילים לפני ואחרי המילה שנרצה לייצג. חלון זה מסומן בטקסט בסוגריים מרובעות בצבע אדום). נבנה מטריצה עם מספר עמודות כאורך המילון ומספר שורות כמספר המילים אותן נרצה לייצג (לרוב מספר השורות יהיה כמספר המילים במילון, אך לשם הדוגמא נציג כאן טבלה קטנה יותר). עבור כל מילה, נבדוק כמה פעמים היא נמצאת בטקסט באותו חלון יחד עם מילים אחרות:

Index	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Dog	1	1	1	1	2	1	1	1	0	0	0	0	0	0	0	0
Cat	1	1	1	1	1	0	0	0	0	0	0	0	0	0	1	1

כעת נוכל לראות שהמכפלה הפנימית בין שני הווקטורים המייצגים את המילים Dog ו-Cat אינה 0. עם זאת, ישנן שתי בעיות נוספות הנובעות מפשטות פתרון זה:

1. מילים פחות משמעותיות כמו is, the, a נכללות בספירה למרות שהן לא בהכרח מוסיפות מידע משמעותי לייצוג.
2. מילים המופיעות בטקסט לעיתים רחוקות יהיו בעלות ייצוג וקטורי מאוד דליל, מה שמגדיל שוב את הסיכוי למכפלה פנימית בעלת ערך קטן מאוד (עד אפסי) עם רוב הווקטורים האחרים.

TF-IDF

הדרך הטבעית לפתרון של הבעיה הראשונה – רעש שנוצר ממילים בעלות תדירות גבוהה שאינן תורמות בייצוג, היא סינון של המילים האלה מהמילון. אולם, פתרון זה אינו ישים, כיוון שהתדירות של מילים משתנה בין תחומים/דומיינים שמהם נלקח הטקסט. לכן פותחה שיטת ייצוג הנקראת TF-IDF, ומטרתה להפחית את רעש זה באופן אוטומטי.

בשיטת TF-IDF, מגדירים פונקציה ניקוד אחרת בעלת שני רכיבים. במקום להסתכל על חלון יחיד מסביב לכל מילה, ניתן להסתכל על כל המילים בחלונות הנוצרים מסביב למילה המיוצגת, ולתת לה ניקוד לפי התדירות של אותה מילה. באופן פורמלי, tf מוגדר בצורה הבאה:

$$tf = \log(C(w, c) + 1)$$

משמעות הביטוי היא שאנו סופרים כמה המילה c מופיעה בקונטקסט (=בחלון) של המילה המיוצגת w (על התוצאה מפעילים \log בשביל rescaling של ערכים גבוהים מאוד).

עד כה השיטה אינה שונה במהותה מספירה כמו שראינו בBag-of-words, אך מה שעושה את TF-IDF שונה הוא הביטוי השני. הביטוי idf מוגדר בצורה הבאה:

$$df = \text{count}(w \in \text{Context})$$

$$idf = \log\left(\frac{N}{df}\right)$$

המונח df מייצג את כמות הפעמים שהמילה w מופיעה בקונטקסטים אחרים, בעוד ש- N מייצג את כמות המילים במילון. נשים לב שאם מילה מסוימת, למשל the , מופיעה בכל הקונטקסטים של כל המילים במילון, אז הביטוי בתוך הלוג יהיה 1 ולכן idf יהיה 0. ולעומת זאת, אם מילה מסוימת מופיעה אך ורק בקונטקסט יחיד, אז הערך יהיה N . בתוך הלוג יהיה N .

לבסוף, TF-IDF מוגדר באופן הבא:

$$TF - IDF = tf \cdot idf$$

מדד משוקלל זה מצליח עבור ייצוג של מילה מסוימת לתת משקל גדול למילים אחרות הנמצאות איתה בקונטקסט באופן תדיר אך אינן נמצאות בקונטקסט של מילים אחרות.

PPMI - Positive Pointwise Mutual Information

כעת נרצה לפתור את הבעיה השנייה – בעיית הייצוג הדליל למילים שאינן תדירות בטקסט. שיטת PPMI מגדירה פונקציית ניקוד המחשבת את היחס בין הסיכוי של שתי המילים להמצא יחד לעומת הסיכוי לראותן בנפרד – $\frac{P(x,y)}{P(x)P(y)}$. כעת בשביל לחשב את הערך של התא המייצג את המילה y בוקטור הייצוג של המילה x נשתמש בהסתברות הנייל ונפעיל לוג. אם ההסתברות לראות את המילים x, y ביחד שווה לכפל ההסתברויות לראות כל אחת לחוד, נקבל שערך הביטוי הוא $\log(1)$ כלומר 0. לעומת זאת, אם הסיכוי לראות את המילים האלו ביחד גדול מהסיכוי שיראו אותם לחוד אז נקבל ערך הגדול מ-1. ישנו מקרה נוסף, בו הסיכוי לראות את הביטויים ביחד קטן מהסיכוי לראות אותם לחוד. במקרה זה הביטוי שנקבל יהיה קטן מ-1 ולכן הלוג יהיה שלילי, אך מכיוון שהערכים השלילים נוטים להיות לא אמינים (אלא אם הטקסט שלנו גדול מספיק), נוסיף עוד אלמנט קטן לפונקציית החישוב:

$$PPMI = \max\left(\log\frac{P(x,y)}{P(x)P(y)}, 0\right)$$

באופן זה נוכל לנרמל את הערך הנמוך עבור מילים נדירות בטקסט.

Word2Vec

השיטות שראינו עד כה לחישוב וקטורי הייצוג של המילים מאפשרות לנו לקודד מאפיינים סמנטיים בייצוג המילים. עם זאת, יש כמה חסרונות לשיטות אלו: ראשית, הן יוצרות וקטורים מאוד דלילים, ובנוסף לכך גודל הווקטורים תלוי בכמות המילים שיש לנו במילון, מה שיצר וקטורים גדולים שמכבידים על החישובים במשימות השפה השונות. למשל – ראינו קודם שניתן לייצג מילה באמצעות וקטור שכולו אפסים למעט תא אחד עם הערך 1 במיקום ייחודי לכל מילה. עבור שפה עם אלפי מילים ואף יותר מכך, כל וקטור המייצג מילה הוא באורך עצום, ועם זאת הוא מאוד דליל כיוון

שיש בו רק מספר תאים מועט שערכם שונה מ-0. לכן, נרצה לפתח שיטה שתיצור וקטורי ייצוג דחוסים (dense) בעלי ממד קטן יותר.

שיטת Word2Vec הינה שיטת Self-Supervised שפותחה למטרת יצירה של וקטורי ייצוג דחוסים של מילים. הפרדיגמה של למידת Self-Supervised "דומה" ללמידה מונחית (supervised learning) רגילה, אך התייגים אינם נתונים אלא נוצרים באופן אוטומטי מתוך הדאטה הלא מתייג. באופן זה ניתן לאמן מודלים עם כמות גדולה של דאטה לא מתייג בצורה יעילה וללא צורך בתייג (שעלול להיות מאוד יקר). בהקשר זה, אלגוריתם Word2Vec משתמש ב-Self-supervision באופן של Skip-Grams-With-Negative-Sampling, או בקיצור –SGNS. הרעיון מאחורי גישה זו הוא להגדיר בעיית סיווג שמטרתה לחזות מה ההסתברות של מילים שונות להיות בקונטקסט של מילה נתונה. בסוף האימון לוקחים את המשקלים שנוצרו בעקבות תהליך אימון המשימה הראשית, והם יהיו הייצוג של המילה. בכדי להבין זאת לעומק, נבחן טקסט פשוט יחסית בעזרת אלגוריתם Word2Vec. נניח ונתון המשפט הבא כחלק מהטקסט האימון שלנו:

Folklore, legends, myths, and fairy tales have followed childhood through the ages.

ראשית נקבע את אורך החלון (=מספר המילים עליהן מסתכלים בסביבות כל מילה) – 3. כעת נעבור על הטקסט וניצור תיגים בין כל מילה במילון ליתר המילים. למשל עבור המילה tales נוסיף לדאטה סט שלנו דוגמאות חיוביות של המילים שנמצאות בקונטקסט עם tales. בנוסף, בכדי למנוע התנוונות של כל הייצוגים לוקטור בודד, נצטרך "להראות" למודל איך נראות דוגמאות שליליות ולכן נשתמש בשיטה הנקראת negative sampling. בשיטה זו דוגמים מהמילון בהסתברות פרופורציונלית לתדירות המילה (עם תיקון קטן שנותן קצת יותר סיכוי למילים נדירות) את המילים שישימשו אותנו כדוגמאות שליליות. הרעיון מאחורי תהליך זה הוא שכאשר יש לנו מילון גדול המילים שנגריל לא יהיו קשורות למילה שעבורה אנחנו יוצרים את הדוגמאות השליליות.

Folklore, legends, [myths and fairy [tales] have followed childhood] through the ages.

word	context	Label
tales	myths	+
tales	and	+
tales	fairy	+
tales	have	+
tales	followed	+
tales	childhood	+
tales	great	-
tales	April	-
tales	the	-
tales	young	-
tales	orphan	-
tales	dishes	-

כך נוכל ליצור דאטה מתייג עבור משימת הסיווג, ונוכל להשתמש בתייגים אלה בשביל לאמן את המודל. הכוונה במשימת סיווג בהקשר זה מעט שונה מסיווג במובן הפשוט של המילה: מטרת המודל היא שבהינתן מילה w (במקרה שלנו tales), נרצה שהייצוג של מילים המופיעות באותו קונטקסט עם כל מילה (במקרה שלנו – אלו שמופיעות עם

tales באותו חלון) יהיה קרוב לייצוג של tales בעוד שמילים שאינן מופיעות באותו קונטקסט יהיו בעלות ייצוג "שונה" (מבחינת מכפלה פנימית או cosine similarity). נניח שבחרנו שהייצוג של כל מילה יהיה וקטור בגודל 100. נתחיל את התהליך כך שכל מילה מקבלת וקטור רנדומלי. נסמן את וקטור הייצוג של המילה w ב- e_w ואת הווקטור של מילת קונטקסט c ב- e_c . השאיפה היא שהמכפלה הפנימית של וקטורי הייצוג של המילים בקונטקסט של w יהיה גבוה יחסית, בעוד שהמכפלה הפנימית של וקטורי יצוג של מילים שאינן מופיעות באותו קונטקסט יהיה נמוך. כאמור לעיל, Cosine similarity (המטריקה המגדירה דמיון בין וקטורים) היא בעצם מכפלת פנימית של הווקטורים (=מכפלת dot עם נרמול). כעת נוכל להגדיר בעיית logistic regression באופן הבא:

$$p(+|w, c) = \sigma(e_w, e_c) = \frac{1}{1 + e^{-e_w e_c}}$$

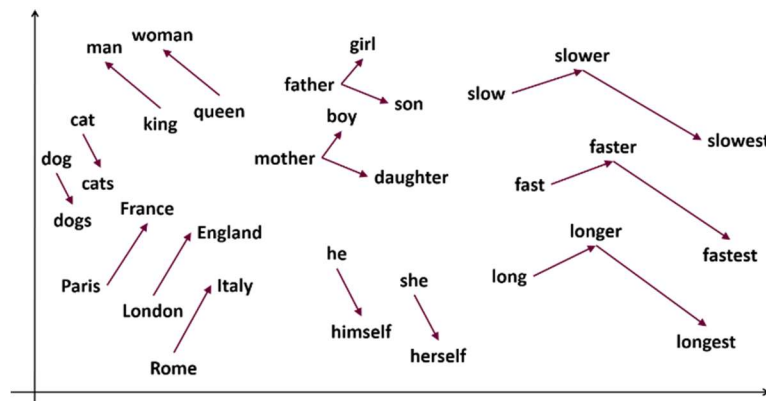
$$p(+|w, c) = 1 - p(-|w, c)$$

ובהתאם לכך פונקציית המטרה (Loss) תוגדר באופן הבא:

$$\begin{aligned} L &= -\log[p(+|w, c_{pos}) \prod_{i=1}^k p(-|w, c_{neg_i})] \\ &= -\left[\log(p(+|w, c_{pos})) + \log\left(\prod_{i=1}^k p(-|w, c_{neg_i})\right) \right] \\ &= -\left[\log(\sigma(e_w \cdot e_{c_{pos}})) + \sum_{i=1}^k \log(1 - \sigma(e_w \cdot e_{c_{neg_i}})) \right] \end{aligned}$$

נשים לב שאנחנו מניחים אי תלות בייצוג של הדוגמאות השליליות. בכך שנבצע מינימיזציה לפונקציית מטרה זו נגרום למכפלה הפנימית בין הייצוג של מילה לבין מילת הקונטקסט להיות גבוהה ובו בזמן למכפלה הפנימית בין וקטורים שאינם בקונטקסט להיות נמוכה. כך הייצוג של המילה tales יהיה "דומה" (=קרוב במונחים של cosine similarity) לייצוג של המילים בקונטקסט ושונה מייצוגן של המילים שאינן בקונטקסט. את תהליך המינימיזציה במשך האימון נוכל לבצע באמצעות stochastic gradient descent.

אחת התוצאות היפות והחשובות של שיטת Word2Vec ניתנת להמחשה על ידי פריסת וקטורי הייצוג בממד נמוך. בעזרת שיטות מתקדמות להורדת ממד (כפי שהוסבר בהרחבה בפרק 2), ניתן לצייר בדו-ממד או תלת ממד את וקטורי הייצוג של המילים לאחר האימון.



איור 11.2 וקטורי הייצוג של מילים לאחר ביצוע embedding באמצעות word2vec. צמדי מילים בעלי משמעות דומה מיוצגים על ידי וקטורים באותו כיוון.

נוכל להבחין שהמרחב הווקטורי מקודד מאפיינים סמנטיים. לדוגמה, ניתן לראות שהווקטור המחבר בין וקטורי הייצוג של המילים *King, Man* מקביל ובעל אורך דומה לווקטור המחבר בין הייצוג של *Queen, Woman*. דוגמה נוספת – הווקטור בין שם של ארץ לעיר הבירה שלה מקביל ובעל אורך דומה לקטור שבין ארץ אחרת ועיר הבירה המתאימה. כמובן שניתן ללמוד מכך על קשרים סמנטיים, כמו למשל שהיחס בין *King, Man* זהה ליחס שבין *Queen, Woman*.

10.1.3 Contextual Embeddings

מנגוני בניית ייצוגי מילים (embedding) שראינו עד כה למדו ייצוג **סטטי** עבור כל מילה. אך דבר זה יכול להיות בעייתי מכיוון ששפה טבעית היא דינמית ותלויה הקשר, ולאותה מילים יכולה להיות כמה פירושים. בשביל להבין את הבעייתיות נסתכל על המשפטים הבאים:

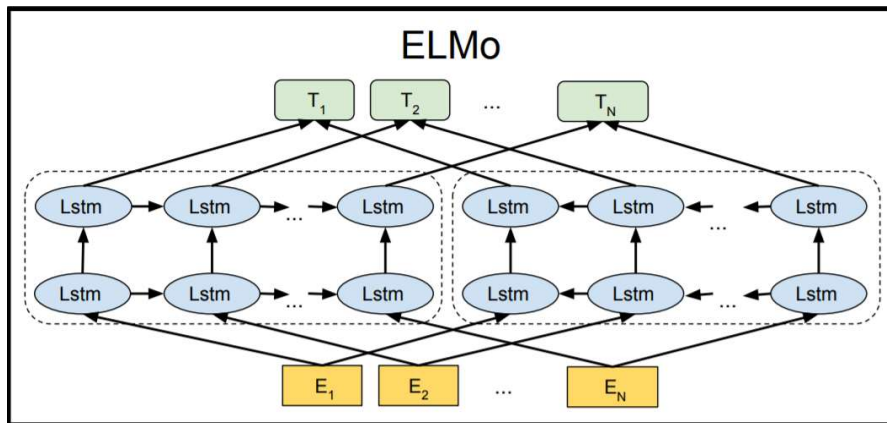
1. We need to **book** the flight as soon as possible
2. I read the **book** already

למילה book יש כפל משמעות במשפטים האלו. במשפט הראשון המילה משמשת כפועל ובמשפט השני כשם עצם עם תפקיד סמנטי שונה במשפט. אם כך ברור שההקשר שבו המילה מופיעה משפיע על המשמעות שלה אך מנגנוני embedding כמו word2vec ייצגו את המילה באותו וקטור ייצוג עבור שני המופעים.

לכן נרצה לפתח מנגנון embedding עבור המילים, שבאמצעותו וקטור המייצג מילה יהיה תלוי בהקשר בו היא מופיעה.

Embeddings from Language Models (ELMo)

אחת השיטות הראשונות שהציגה טכניקה ללמידת ייצוג תלוי הקשר למילים הינה Embeddings from Language Models, או בקיצור ELMo – ארכיטקטורה הבונה לכל מילה ייצוג שתלוי בהקשר שלה בתוך המשפט. הרעיון במודל זה הוא לקחת ייצוג של מילה, להוסיף לו מידע נוסף מההקשר של המילה במשפט ולקבל ייצוג חדש התלוי גם בהקשר שלה. בניסוח אחר ניתן לומר ש-ELMo הינה פונקציה המקבלת משפט שבו כל מילה מיוצגת בדרך כלשהיא (למשל Word2Vec – ומוסיפה לייצוג זה גם את ההקשר של המילה בתוך כלל המשפט. בפועל זה נעשה על ידי משימת אימון מודל שפה דו-כיווני – המודל לומד לחזות גם את המילה הבאה בטקסט וגם את המילה הקודמת, ובכך הוא לומד לתת למילה גם את ההקשר שלה. ארכיטקטורת הרשת נראית כך:



איור 11.3 ארכיטקטורת ELMo. הקלט הינו משפט המיוצג כלשהוא, והפלט הוא אותו משפט אך כל מילה קיבלה מידע נוספת על ההקשר שלה וכעת מיוצגת באופן חדש. תהליך האימון והוספת ההקשר בין המילים נעשה באמצעות שכבות של רכיבי LSTM.

כפי שמתואר בפרק 6.2.1, כל בלוק של LSTM מקבל כקלט שני רכיבים המייצגים את ההיסטוריה של המשפט עד הנקודה בה מופיעה המילה של ה-timestamp הנוכחי (c_t, h_t) , וקלט נוסף של האיבר הנוכחי בסדרה, שבמקרה שלנו זה המילה הנוכחית (x_t) . המוצא של ה-LSTM הינו ייצוג חדש המשקלל את רכיבי ההיסטוריה יחד עם הייצוג הנוכחי של המילה.

בדומה לשיטות אחרות ליצירת ייצוג וקטורי למילים, אנחנו מאמנים את המודל בעזרת משימת מידול שפה וחוזים את המילה הבאה בהינתן המילים הקודמות. אך בשונה מאלגוריתמים אחרים, ELMo משתמש בארכיטקטורה דו-כיוונית, כך שבתהליך האימון משולבת משימת שפה נוספת המנסה לחזות את המילה הקודמת בהינתן הסוף של המשפט. הארכיטקטורה של ELMo בנויה ממספר שכבות של LSTM שמורכבות זו על גבי זו, ולפי כותבי המאמר השכבות התחתונות מצליחות ללמוד פיצ'רים פשוטים (למשל מאפיינים סינטקטיים למיניהם), בעוד שהשכבות העליונות לומדות פיצ'רים מורכבים (למשל מאפיינים סמנטיים, כמו משמעות המילה בהקשר).

לאחר תהליך האימון ניתן להקפיא את הפרמטרים של המודל ולהשתמש בו עבור משימות אחרות. הכותבים מציעים לשרשר את הייצוג הווקטורי של LSTM בכל שכבה ככה שיכיל אינפורמציה גם מתחילת המשפט עד המילה הנבדקת וגם מסוף המשפט עד המילה הנבדקת. מה שקורה בפועל זה שהשכבות החבויות (hidden layers) של LSTM הם עצמם מהווים את ייצוגי המילים בשיטת ייצוג כזו, כלומר כל מילה במשפט מיוצגת על ידי התא המקביל בשכבת ה-LSTM שמעליה. בנוסף הם מוסיפים מספר פרמטרים קטן שמאפשר כיוול (Fine tune) עבור משימה ספציפית. כך לדוגמא נוכל להתאים את הייצוג של המילים למשימת סיווג של משפט לעומת משימת תיוג של ישויות במשפט.

פה חשוב להדגיש נקודה מרכזית – בסופו של דבר התוצר של ELMo הינו **מודל שפה הלוקח ייצוג של טקסט והופך אותו לייצוג תלוי הקשר**. שכבות ה-LSTM השונות מאמנות מודל שפה על מנת ליצור ייצוג חדש עבור המילים,

המתייחס גם להקשר. לאחר סיום האימון של מודל השפה (pre-training), ניתן לקחת אותו ולבצע transfer learning, כלומר להשתמש בייצוגים שהוא מפיק גם למשימות אחרות על ידי הוספת שכבות בקצה. לאחר פרסום המאמר, Sebastian Ruder (חוקר NLP מפורסם) טען כי:

"It is very likely that in a year's time NLP practitioners will download pretrained language models rather than pre-trained word embeddings"

כלומר, כעת מי שירצה לבצע משימת שפה כבר לא יתבסס רק על ייצוג סטטי של המילים אלא הוא יסתמך על מודל שפה מאומן שיועד לקחת ייצוג התחלתי של מילים ולהפוך אותם לייצוגים קונטקסטואליים. ELMo ועוד מאמרים רבים אחריו אימנו מודלי שפה מאומנים שניתם לקחת אותם ולהשתמש בהם עבור משימות קצה שונות על ידי הוספה של כמה שכבות וכיול המודל.

Bidirectional Encoder Representations from Transformers (BERT)

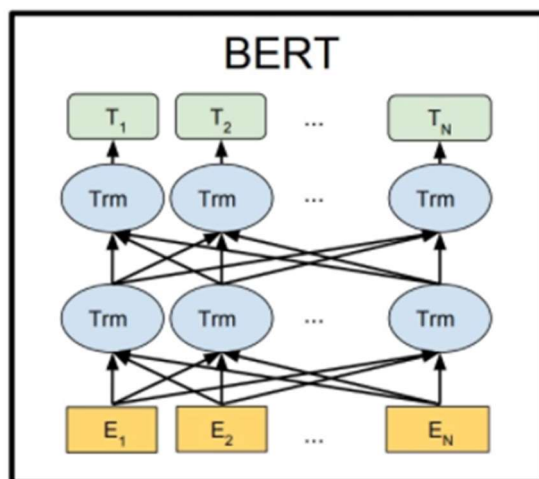
כאמור, הרעיון של ELMo הוא לייצר contextual embedding, ובכך לקבל מודל שפה המאפשר לקחת ייצוג וקטורי של מילים ולהעשיר אותו במידע על ההקשר של כל מילה בטקסט. למרות ש-ELMo משתמש בשני הכיוונים של המשפט (חוזה את המשך המשפט מתחילתו ואת תחילת המשפט מסופו) הוא אינו לומד משני הכיוונים בתהליך אחד, אלא צריך לחלק את הלמידה לשני חלקים שונים. בנוסף, כפי שהוסבר בהקדמה לפרק 8, כאשר מתעסקים עם סדרות ארוכות של מילים, וקטור הייצוג של כל איבר נהיה בעייתי כיוון שהוא מוגבל ביכולת שלו להכיל קשרים בין מספר רב של איברים. במילים אחרות, כאשר רוצים להוסיף לווקטור הייצוג של מילה מסוימת קשר למילים רחוקות, אנו מאלצים את הייצוג "לזכור" מידע רב, אך הייצוג הווקטורי של המילים ב-ELMo אינו מצליח לעשות זאת בצורה מספיק טובה. לכן, על אף הצלחת גישה זו במשימות שונות, היא עדיין התקשה במשימות בהן נדרשת יכולת לנתח טקסטים ארוכים (כמו למשל משימה של summarization). בנוסף לכך, האלגוריתם יחסית איטי, כיוון שכל פעם הוא מסתכל על מילה אחת בלבד.

בכדי להתמודד עם בעיות אלו וליצור ייצוג המסוגל להכיל מידע איכותי גם ברצפים ארוכים, ניתן להשתמש ב-attention (מוסבר בהרחבה בפרק 8). אחד השימושים הראשונים במנגנון ה-attention עבור משימת עיבוד שפה היה בטרנספורמרים, ובפרט בארכיטקטורת רשת הנקראת BERT, המבוססת על ה-encoder של הטרנספורמר המקורי. שימוש זה היווה פריצת דרך בתחום, וכיום ברוב המוחלט של המחקר והפיתוח בתחום ה-NLP משתמשים בארכיטקטורת רשת מבוססת attention. למעשה, BERT מציע שיטה לבניית ייצוג קונטקסטואלי של מילים הבא להתמודד עם החולשות הקיימות ב-ELMo. נתאר בקצרה את העקרונות של מנגנון ה-attention, שהוא הלב של BERT:

באופן הכי פשוט, בהקשר של עיבוד שפה self-attention הוא מנגנון שמשערך את הקשרים של כל מילה בטקסט כלשהו ביחס לשאר המילים באותו טקסט. כאשר מבצעים self-attention על קטע טקסט, מקבלים ייצוגים חדשים של המילים הלוקחים בחשבון גם את הקשרים בין המילים השונות באותו טקסט. בזכות אופיו של מנגנון ה-attention, ניתן לבנות ייצוג של מילה שתלוי בקשרים שלה עם מילים הנמצאות רחוק ממנה בקטע טקסט, כלומר ההקשרים המתקבלים בין המילים יכולים להיות מיוצגים בצורה טובה גם עבור רצפים ארוכים ומילים שאינן נמצאות בסמיכות יחסית (שכאמור זה היה אחד החסרונות הגדולים של ELMo). בנוסף, מנגנון זה מייצר את הצורך לעבור מילה אחר מילה בקטע טקסט לצורך בניית ייצוג המילים שבו. במקום מעבר זה, ה-encoder מקבל הקלט את כל קטע הטקסט כמקשה אחת, מה שעשוי להקטין את הזמן הנדרש עבור בניית הייצוג של המילים. לכן ה-encoder בטרנספורמר יכול לשמש מודל שפה, אם מאמנים אותו בצורה מתאימה.

בשונה ממודל LSTM ששומר את המצב בכל נקודת זמן ובעצם מקודד את המיקום של כל מילה בכך שהקלט מתקבל כמילה בודדת בכל פעם, מודל הטרנספורמר מקבל את כל הקלט בבת אחת. לכן בשביל לקחת בחשבון את המיקום של כל מילה במשפט אנחנו משתמשים באלמנט נוסף שנקרא Positional Embedding. אלמנט זה מקודד וקטור ייחודי לכל מיקום במשפט ובסוף מבצעים חיבור של הווקטור שנוצר מהקלט והווקטור שנוצר מהמיקום.

המפתחים של BERT אימצו מארכיטקטורת הטרנספורמר המקורית את ה-encoder, והגדירו משימת אימון חדשה בכדי להפוך אותו למודל שפה. בכדי לבנות מודל מוצלח, תהליך האימון של BERT כלל שתי משימות: 1. Masked Language Model (MLM) – באופן רנדומלי עושים masking למילים מסוימות, ומטרת המודל הוא לחזות את המילים החסרות. 2. Next Sentence Prediction (NSP) – המודל מקבל כקלט זוגות של משפטים מקטע טקסט, ומטרת המודל היא לחזות לחזות האם המשפט השני הוא המשכו של המשפט הראשון במסך המקורי. ארכיטקטורת הרשת נראית כך:



איור 11.4 ארכיטקטורת BERT. הקלט הינו משפט המיוצג כלשהוא, והפלט הוא אותו משפט אך כל מילה קיבלה מידע נוסף על ההקשר שלה וכעת מיוצגת באופן חדש. תהליך האימון והוספת ההקשר בין המילים נעשה באמצעות self-attention.

גם BERT, בדומה ל-ELMo, מציע בסופו של דבר מודל שפה מאומן היודע לקחת טקסט המיוצג באופן מסוים ולהוסיף לו מידע על היחס בין המילים השונות שבטקסט. תהליך יצירת המודל היה אמנם יקר, אך כעת ניתן לקחת אותו ויחסית בקלות לכייל אותו ואף להוסיף שכבות בקצה עבור משימות שפה שונות.

GPT: Generative Pre-trained Transformer

עם הכניסה של מנגנון ה-attention וטרנספורמרים לעולם ה-NLP, הוצעו יותר ויותר מודלי שפה מבוססי attention. לצורך ההמחשה ניתן לציין שבשנים הבודדות שעברו מאז יצא BERT, הוא צוטט כבר בעשרות אלפי מאמרים. אחד המודלים היותר מפורסמים הינו Generative Pre-Training (GPT). מודל ה-GPT הינו מודל שעובד בשיטת auto-regression, כלומר, כאשר המודל חוזר את המילה הבאה הוא מוסיף את המילה לקלט עבור האיטרציה הבאה. כך הוא יכול בעצם ליצר משפטים מהתחלה של מילה בודדת. אם נרצה לדייק, המודלים הללו לא תמיד משתמשים במילים כיחידה האטומית, לפעמים אנחנו נעבוד עם חלקי מילים ואפילו אותיות להם נקרא טוקנים או אסימונים. דבר זה יכול לעזור לנו בהכללה ולהקטין את הסיכוי לטוקן שלא נמצא במילון (Out of Vocabulary).

הארכיטקטורה של GPT בנויה מ-Transformers מה שמאפשר לבנות ארכיטקטורה עמוקה שמתחשבת בקונטקסט של המשפט עבור כל מילה (Contextual embeddings). ארכיטקטורת Transformer הינה היחידה המרכזית של GPT, כאשר בשונה מ-BERT ה-GPT משתמש רק ב-decoder (מנגנון ה-self-attention) שמקודד את הפיצ'רים, והפלט שלו הינו הטוקן הבא.

השכבה הראשונה בארכיטקטורה של GPT היא שכבה הנקראת Input encoding והיא הופכת את המילים (או ליתר דיוק הטוקנים) לטוקטורים, כלומר היא מבצעת word embedding.

לאחר קידוד הקלט נשתמש במודל ה-Transformer בכדי לקודד פיצ'רים שמהם נסיק את הטוקן הבא. התהליך הזה מתבצע בעזרת רכיב הנקרא Masked Self attention. בשונה ממנגנון self-attention רגיל שמקודד כל טוקן בעזרת הקונטקסט של כל שאר הטקסט, GPT צריך לקודד כל טוקן רק בעזרת הטוקנים שקדמו לו, כיוון שבשלב זה המידע היחיד שקיים זה הטוקנים שנוצרו עד כה (וכמובן שאין גישה לטוקנים שעדיין לא נוצרו). כאשר מקודדים את הייצוג עבור טוקן מסוים, רכיב Masked Self attention מאפס כל וקטור של טוקן שבא אחריו, כך שהמודל לא יכול ללמוד ייצוג התלוי מילים שבאות לאחר הטוקן המיוצג, אלא עליו להפיק את המירב מהטוקנים הקודמים לו.

כיוון ש-GPT פועל בצורה של auto-regressive, ניתן לאחר האימון ליצור טקסט באמצעותו – ניתן למודל התחלה קצרה של טקסט, ונבקש ממנו ליצור את המילים הבאות. כך בכל שלב ניתן לו קלט את הטקסט הראשוני ואת הטוקנים שיצר בשלבים הקודמים, והוא ימשיך וייצר עוד ועוד טקסט.

Perplexity

לאחר בניית מודל שפה, נרצה "למדוד" עד כמה הוא מוצלח. לצורך זה יש להגדיר מטריקה מתאימה. המטריקה הכי נפוצה למדידת "עוצמה" של מודל שפה הינה perplexity, שזהו מושג הלקוח מתורת האינפורמציה והוא מודד כמה טוב מודל השפה חוזה את השפה ב-Corpus שאותו ניסיון למדל.

לפני שנסביר את המושג באופן פורמלי ניתן אינטואיציה למה אנו מצפים לקבל מהמטריקה שנבחר. נניח ואנו מבצעים את הפעולה הבאה: ראשית לוקחים משפט שלם וחותכים ממנו את ההתחלה, ואז לוקחים את אותה התחלה ומכניסים כקלט למודל שפה ומבקשים מהמודל לחזות את המשך המשפט. כמובן שנרצה לקבל חיזוי שדומה ככל האפשר למשפט המקורי, ונוכל למדוד הצלחה של מודל על ידי השוואת הפלט שלו למשפט האמיתי. באופן יותר כללי ניתן לקחת טקסט המכיל כמות משפטים כרצוננו, להכניס חלקים ממנו למודל השפה, ולהשוות את הפלט המתקבל לטקסט המקורי. כיוון שמודל שפה הינו הסתברותי, השוואת הפלט למשפט המקורי באופן מילולי בלבד אינה מספיקה, כיוון שהיא אינה משקפת בצורה מספיק טובה את מידת ההצלחה שלו. אם למשל במשפט המקורי הייתה כתובה המילה "לבנה" ואילו המודל חזה את המילה "ירח", השוואת שתי המילים כשלעצמן מראה לכאורה שהמודל שגה לחלוטין, אך בפועל אנו יודעים שמילים אלו נרדפות ולכן הפלט של המודל במקרה זה הוא דווקא כן טוב. לכן, נרצה לבחור מדד המסוגל לבחון עד כמה סביר לקבל את הפלט של המודל בהינתן חלק מהמשפט המקורי.

מדד perplexity בא להתמודד עם אתגר זה, והוא אכן פועל בצורה שונה מהאופן בו תיארנו את ההשוואה הפשוטה בין טקסט המקור לבין הפלט של מודל השפה. מדד זה מסתכל רק על הטקסט המקורי, והוא עובר מילה-מילה בטקסט זה ובודק מה ההסתברות שמודל השפה ינבא את המילה הבאה בטקסט בהינתן כל המילים שלפניה. ככל שההסתברות יותר גבוהה, כך המודל יותר מוצלח. באופן פורמלי, perplexity מוגדר באופן הבא:

$$PP(W) = P(w_1, w_2, \dots, w_N)^{-\frac{1}{N}} = \sqrt[N]{\frac{1}{P(w_1, w_2, \dots, w_N)}}$$

ככל שהמודל מנבא בהסתברות גבוהה יותר את המילים של המשפט המקורי, כך המונה שבתוך השורש יהיה יותר גדול, וממילא כל הביטוי עצמו של ה-perplexity נהיה קטן יותר. כלומר, ככל שערך ה-perplexity קטן יותר, כך המודל מוצלח יותר. נפתח מעט את הביטוי האחרון בעזרת כלל השרשרת:

$$= \sqrt[N]{\frac{1}{\prod_{i=1}^N P(w_i | w_{i-1}, w_{i-2}, \dots, w_1)}}$$

למשל עבור מודל מבוסס bigram, המדד יהיה פשוט יותר ויראה כך:

$$= \sqrt[N]{\frac{1}{\prod_{i=1}^N P(w_i | w_{i-1})}}$$

כאמור לעיל, ככל שערך מדד perplexity נמוך יותר, כך מודל השפה איכותי יותר.

10. References

<http://d2l.ai/>

ELMo, BERT:

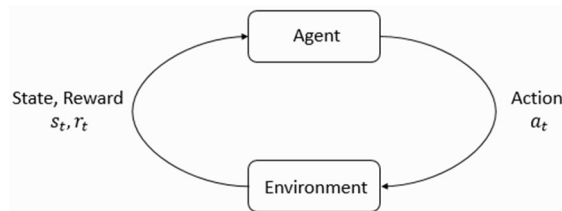
<https://ai.googleblog.com/2018/11/open-sourcing-bert-state-of-art-pre.html>

11. Reinforcement Learning (RL)

רוב האלגוריתמים של עולם הלמידה הינם מבוססי דאטה, כלומר, בהינתן מידע מסוים הם מנסים למצוא בו חוקיות מסוימת, ועל בסיסה לבנות מודל שיוכל להתאים למקרים נוספים. אלגוריתמים אלה מחולקים לשניים:

1. אלגוריתמים של למידה מונחית, המבוססים על דאטה $S = \{x, y\}$, כאשר $x \in \mathbb{R}^{n \times d}$ הינו אוסף של אובייקטים (למשל נקודות במרחב, אוסף של תמונות וכדו'), ו- $y \in \mathbb{R}^n$ הינו אוסף של labels. לכל אובייקט $x \in \mathbb{R}^d$ יש label מתאים $y \in \mathbb{R}^1$.
2. אלגוריתמים של למידה לא מונחית עבורם הדאטה $x \in \mathbb{R}^{n \times d}$ הוא אוסף של אובייקטים ללא labels, ומנסים למצוא כללים מסוימים על דאטה זה (למשל – חלוקה לאשכולות, הורדת ממד ועוד).

למידה מבוססת חיזוקים הינה פרדיגמה נוספת תחת התחום של למידת מכונה, כאשר במקרה זה הלמידה לא מסתמכת על דאטה קיים, אלא על חקירה של הסביבה ומציאת המדיניות/האסטרטגיה הטובה ביותר לפעולה. ישנו סוכן שנמצא בסביבה שאינה מוכרת, ועליו לבצע צעדים כך שהתגמול המצטבר אותו הוא יקבל יהיה מקסימלי. בלמידה מבוססת חיזוקים, בניגוד לפרדיגמות האחרות של למידת מכונה, הסביבה לא ידועה מבעוד מועד. הסוכן נמצא באי ודאות ואינו יודע בשום שלב מה הצעד הנכון לעשות, אלא הוא רק מקבל פידבק על הצעדים שלו, וכך הוא לומד מה כדאי לעשות וממה כדאי להימנע. באופן כללי ניתן לומר שמטרת הלמידה היא לייצר אסטרטגיה כך שבכל מיני מצבים לא ידועים הסוכן יבחר בפעולות שבאופן מצטבר יהיו הכי יעילות עבורו. נתאר את תהליך הלמידה באופן גרפי:



איור 11.1 מודל של סוכן וסביבה.

בכל צעד הסוכן נמצא במצב s_t ובחר פעולה a_t המעבירה אותו למצב s_{t+1} , ובהתאם לכך הוא מקבל מהסביבה תגמול r_t . האופן בה מתבצעת הלמידה היא בעזרת התגמול, כאשר נרצה שהסוכן יבצע פעולות המזכות אותו בתגמול חיובי (-חיזוק) וימנע מפעולות עבורן הוא מקבל תגמול שלילי, ובמצטבר הוא ימקסם את כלל התגמולים עבור כל הצעדים שהוא בחר לעשות. כדי להבין כיצד האלגוריתמים של למידה מבוססת חיזוקים עובדים ראשית יש להגדיר את המושגים השונים, ובנוסף יש לנסח באופן פורמלי את התיאור המתמטי של חלקי הבעיה השונים.

11.1 Introduction to RL

בפרק זה נגדיר באופן פורמלי תהליכי מרקוב, בעזרתם ניתן לתאר בעיות של למידה מבוססת חיזוקים, ונראה כיצד ניתן למצוא אופטימום לבעיות אלו בהינתן מודל וכל הפרמטרים שלו. לאחר מכן נדון בקצרה במספר שיטות המנסות למצוא אסטרטגיה אופטימלית עבור תהליך מרקוב כאשר לא כל הפרמטרים של המודל נתונים, ובפרקים הבאים נדבר על שיטות אלה בהרחבה. שיטות אלה הן למעשה הלב של למידה מבוססת חיזוקים, כיוון שהן מנסות למצוא אסטרטגיה אופטימלית על בסיס תגמולים ללא ידיעת הפרמטרים של המודל המרקובי עבורו רוצים למצוא אופטימום.

11.1.1 Markov Decision Process (MDP) and RL

המודל המתמטי העיקרי עליו בנויים האלגוריתמים השונים של RL הינו תהליך החלטה מרקובי, כלומר תהליך שבה המעברים בין המצבים מקיים את תכונת מרקוב, לפיה ההתפלגות של מצב מסוים תלויה רק במצב הקודם לו:

$$P(s_{t+1} = j | s_1, \dots, s_t) = P(s_{t+1} = j | s_t)$$

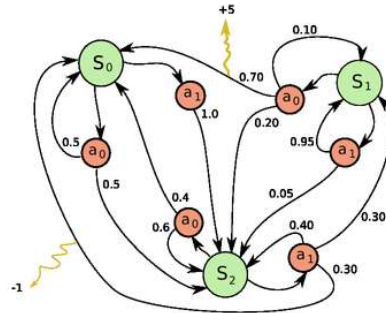
תהליך קבלת החלטות מרקובי מתואר על ידי סט הפרמטרים $\{\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}\}$:

- State space (\mathcal{S}) – מרחב המצבים של המערכת. המצב ההתחלתי מסומן ב- s_0 .
- Action space (\mathcal{A}) – מרחב הפעולות. A_s הוא מרחב הפעולות האפשריות במצב s .
- Transition (\mathcal{T}) – הביטוי: $T(s'|s, a) \rightarrow [0, 1]$ הינו פונקציית מעבר, המחשבת את ההסתברות לעבור בזמן t ממצב s_t למצב s_{t+1} על ידי הפעולה $a: T(s'|s, a) = \mathcal{P}(s_{t+1} = s' | s_t = s, a_t = a)$. ביטוי זה למעשה מייצג את המודל – מה ההסתברות שבחירת הפעולה a במצב s תביא את הסוכן למצב s' .
- Reward (\mathcal{R}) – הביטוי: $\mathcal{R}_a(s, s') \rightarrow \mathbb{R}$ הינו פונקציה הנותנת תגמול/רווח לכל פעולה a הגורמת למעבר ממצב s למצב s' , כאשר בדרך כלל $\mathcal{R}_a \in [0, 1]$. לעיתים מסמנים את התגמול של הצעד בזמן t ב- r_t .

המרקוביות של התהליך באה לידי ביטוי בכך שמצב s_t מכיל בתוכו את כל המידע הנחוץ בכדי לקבל החלטה לגבי a_t , או במילים אחרות – כל ההיסטוריה בעצם שמורה בתוך המצב s_t .

ריצה של MDP מאופיינת על ידי הרביעייה הסדורה $\{s_t, a_t, r_t, s_{t+1}\}$ – פעולה a_t המתרחשת בזמן t וגורמת למעבר ממצב s_t למצב s_{t+1} , ובנוסף מקבלת תגמול מיידי r_t , כאשר $r_t \sim \mathcal{R}(s_t, a_t)$ ו- $s_{t+1} \sim p(\cdot | s_t, a_t)$.

מסלול (trajectory) הינו סט של שלשות $\tau = \{s_0, a_0, r_0, \dots, s_t, a_t, r_t\}$, כאשר המצב התחלתי מוגרל מהתפלגות כלשהיא $s_0 \sim \rho_0(\cdot)$, והמעבר בין המצבים יכול להיות דטרמיניסטי $s_{t+1} = f(s_t, a_t)$ או סטוכסטי $s_{t+1} \sim p(\cdot | s_t, a_t)$.



איור 11.2 תהליך קבלת החלטות מרקובי. ישנם שלושה מצבים – $\{s_0, s_1, s_2\}$, ובכל אחד מהם יש שתי פעולות אפשריות (עם הסתברויות מעבר שונות) – $\{a_0, a_1\}$. עבור חלק מהפעולות יש תגמול שונה מ-0. מסלול יהיה מעבר על אוסף של מצבים דרך אוסף של פעולות, שלכל אחד מהן יש תגמול.

אסטרטגיה של סוכן, המסומנת ב- π , הינה בחירה של אוסף מהלכים. בבעיות של למידה מבוססת חיזוקים, נרצה למצוא אסטרטגיה אופטימלית (Optimal Policy) $\pi: S \rightarrow A$ הממקסמת את התגמול המצטבר $\sum_{t=0}^{\infty} \mathcal{R}(s_t, \pi(s_t))$ כיוון שלא תמיד אפשרי לחשב באופן ישיר את האסטרטגיה האופטימלית, ניתן להגדיר ערך החזרה (Return) המבטא סכום של תגמולים, ומנסים למקסם את התוחלת שלו $\mathbb{E}[Return | \mathcal{S}, \mathcal{A}]$. ערך החזרה הכי נפוץ נקרא discount return, והוא מוגדר באופן הבא: עבור פרמטר $\gamma \in (0, 1)$, ה-Return הינו הסכום הבא:

$$Return = \sum_{t=1}^T \gamma^{t-1} r_t$$

אם $\gamma = 0$, אז מתעניינים רק בתגמול המיידי, וככל ש- γ גדל כך נותנים יותר משמעות לתגמולים עתידיים. כיוון ש- $r_t \in [0, 1]$, הסכום חסום על ידי $\frac{1}{1-\gamma}$.

התוחלת של ערך החזרה נקראת Value function, והיא נותנת לכל מצב ערך מסוים המשקף את תוחלת התגמול שניתן להשיג דרך מצב זה. באופן פורמלי, כאשר מתחילים ממצב s , ה-Value function מוגדר להיות:

$$\mathcal{V}^\pi(s) = \mathbb{E}[R(\tau) | s_0 = s]$$

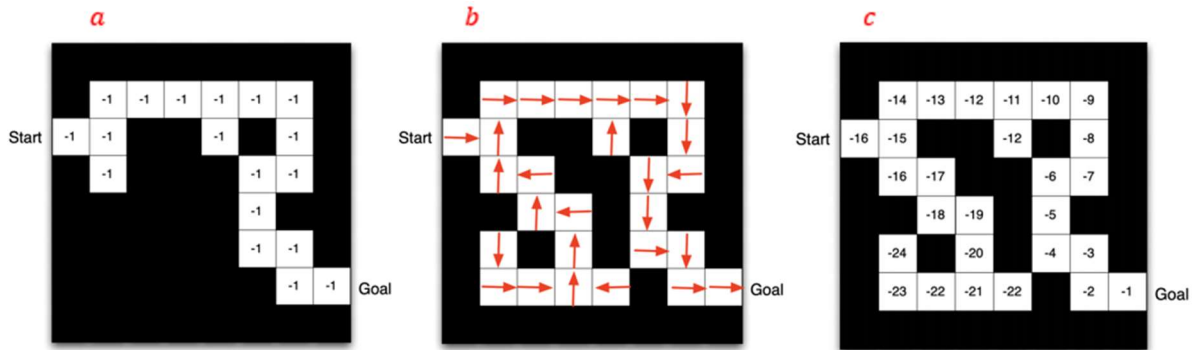
בעזרת ביטוי זה ניתן לחשב את האסטרטגיה האופטימלית, כאשר ניתן לנקוט בגישה ישירה ובגישה עקיפה. הגישה הישירה מנסה למצוא בכל מצב מה הפעולה הכי כדאית. בהתאם לכך, חישוב האסטרטגיה האופטימלית יעשה באופן הבא:

$$\pi(s) = \arg \max_a \sum_{s'} p_a(s, s') (\mathcal{R}_a(s, s') + \gamma \mathcal{V}(s'))$$

לעיתים החישוב הישיר מסובך, כיוון שהוא צריך לקחת בחשבון את כל הפעולות האפשריות, ולכן מסתכלים רק על ה-Value function. לאחר שלכל מצב יש ערך מסוים, בכל מצב הסוכן יעבור למצב בעל הערך הכי גדול מבין כל המצבים האפשריים אליהם ניתן לעבור. חישוב הערך של כל מצב נעשה באופן הבא:

$$\mathcal{V}(s) = \sum_{s'} p_\pi(s, s') (\mathcal{R}_\pi(s, s') + \gamma \mathcal{V}(s'))$$

ניתן לשים לב שבעוד הגישה הראשונה מתמקדת במציאת אסטרטגיה/מדיניות אופטימלית על בסיס הפעולות האפשריות בכל מצב, הגישה השנייה לא מסתכלת על הפעולות אלא על הערך של כל מצב, המשקף את תוחלת התגמול שניתן להשיג כאשר נמצאים במצב זה.



איור 11.3 (a) מודל: המצב של הסוכן הוא המשבצת בו הוא נמצא, הפעולות האפשריות הן ארבעת הכיוונים, כל פעולה גוררת תגמול של -1, והסתברויות המעבר נקבעות לפי הצבעים של המשבצות (אי אפשר ללכת למשבצות שחורות). (b) מדיניות – החלטה בכל מצב איזה צעד לבצע. (c) Value של כל משבצת.

לסיכום, ניתן לומר שכל התחום של RL מבוסס על שלוש אבני יסוד:

- מודל: האופן בו אנו מתארים את מרחב המצבים והפעולות. המודל יכול להיות נתון או שנצטרך לשערך אותו, והוא מורכב מהסתברויות מעבר בין מצבים ותגמול עבור כל צעד:

$$\mathcal{P}_{SS}^a = p_\pi(s, s') = P(s_{t+1} = s' | s_t = s, a_t = a)$$

$$\mathcal{R}_{SS}^a = \mathcal{R}_\pi(s, s') = \mathbb{E}[r_{t+1} | s_t = s, a_t = a, s_{t+1} = s']$$

- Value function – פונקציה המתארת את התוחלת של התגמולים העתידיים:

$$\mathcal{V}^\pi(s) = \mathbb{E}[R(\tau) | s_0 = s]$$

- מדיניות/אסטרטגיה (Policy) – בחירה (דטרמיניסטית או אקראית) של צעד בכל מצב נתון: $\pi(s|a)$

11.1.2 Bellman Equation

לאחר שהגדרנו את המטרה של למידה מבוססת חיזוקים, ניתן לדבר על שיטות לחישוב אסטרטגיה אופטימלית. בפרק זה נתייחס למקרה הספציפי בו נתון מודל מרקובי עם כל הפרמטרים שלו, כלומר אוסף המצבים, הפעולות והסתברויות המעבר ידועים. כאמור, Value function הינה התוחלת של ערך ההחזרה עבור אסטרטגיה נתונה π , כאשר מתחילים ממצב s :

$$\mathcal{V}^\pi(s) = \mathbb{E}[R(\tau) | s_0 = s] = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s \right]$$

ביטוי זה מסתכל על הערך של כל מצב, בלי להתייחס לפעולות המעבירות את הסוכן ממצב אחד למצב אחר. נתינת ערך לכל מצב יכולה לסייע במציאת אסטרטגיה אופטימלית, כיוון שהיא מדרגת את המצבים השונים של המודל. באופן דומה, ניתן להגדיר את ה-Action-Value function – התוחלת של ערך ההחזרה עבור אסטרטגיה נתונה π , כאשר במצב s מבצעים את פעולה a , ולאחר מכן ממשיכים לפי האסטרטגיה π :

$$Q^\pi(s, a) = \mathbb{E}[R(\tau) | s_0 = s, a_0 = a] = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s, a_t = t \right]$$

ביטוי זה מסתכל על הזוג (s_t, a_t) , כלומר בכל מצב יש התייחסות למצב הנוכחי ולפעולות האפשריות במצב זה. בדומה ל-Value function, גם ביטוי זה יכול לסייע במציאת אסטרטגיה אופטימלית, כיוון שהוא מדרג עבור כל מצב את הפעולות האפשריות.

נוכל לסמן ב- $\mathcal{V}^*(s)$ ו- $Q^*(s, a)$ את הערכים של האסטרטגיה האופטימלית π^* – Optimal Value function ו-Optimal Action-Value function. עבור אסטרטגיה זו מתקיים:

$$\mathcal{V}^*(s) = \max_{\pi} \mathbb{E}[R(\tau)|s_0 = s], Q^*(s, a) = \max_{\pi} \mathbb{E}[R(\tau)|s_0 = s, a_0 = a]$$

הרבה פעמים מתעניינים ביחס שבין \mathcal{V} ו- Q , וניתן להיעזר במעברים הבאים:

$$\mathcal{V}^{\pi}(s) = \mathbb{E}[Q^{\pi}(s, a)]$$

$$\mathcal{V}^*(s) = \max_{\pi} Q^*(s, a)$$

באופן קומפקטי ניתן לרשום את $\mathcal{V}^*(s)$ כך:

$$\forall s \in S \quad \mathcal{V}^*(s) = \max_{\pi} \mathcal{V}^{\pi}(s)$$

כלומר, האסטרטגיה π^* הינה האופטימלית עבור כל מצב s .

כעת נתון מודל מרקובי עם כל הפרמטרים שלו – אוסף המצבים והפעולות, הסתברויות המעבר והתגמול עבור כל פעולה, ומעוניינים למצוא דרך פעולה אופטימלית עבור מודל זה. ניתן לעשות זאת בשתי דרכים עיקריות – מציאת האסטרטגיה $\pi(a|s)$ האופטימלית, או חישוב ה-Value של כל מצב ובחירת מצבים בהתאם לערך זה. משימות אלו יכולות להיות מסובכות מאוד עבור משימות מורכבות וגדולות, ולכן לעיתים קרובות משתמשים בשיטות איטרטיביות ובקירובים על מנת לדעת כיצד לנהוג בכל מצב. הדרך הפשוטה לחישוב $\mathcal{V}^{\pi}(s)$ משתמשת ב-**Bellman equation**, המבוססת על תכנות דינמי. נפתח את הביטוי של $\mathcal{V}^{\pi}(s)$ מתוך ההגדרה שלו:

$$\mathcal{V}^{\pi}(s) = \mathbb{E}[R(\tau)|s_0 = s] = \mathbb{E}_{\pi} \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s \right]$$

נפצל את הסכום שבתוחלת לשני איברים – האיבר הראשון ויתר האיברים:

$$= \mathbb{E}_{\pi} \left[r_{t+1} + \gamma \cdot \sum_{k=0}^{\infty} \gamma^k r_{t+k+2} | s_t = s \right]$$

כעת נשתמש בהגדרת התוחלת ונקבל:

$$\begin{aligned} &= \sum_{a, s'} \pi(a|s) p_{\pi}(s, s') \left(\mathcal{R}_{\pi}(s, s') + \gamma \cdot \mathbb{E}_{\pi} \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+2} | s_t = s \right] \right) \\ &= \sum_{a, s'} \pi(a|s) p_{\pi}(s, s') \left(\mathcal{R}_{\pi}(s, s') + \gamma \cdot \mathcal{V}^{\pi}(s') \right) \end{aligned}$$

הביטוי המתקבל הוא מערכת משוואות לינאריות הניתנות לפתרון באופן אנליטי, אם כי סיבוכיות החישוב יקרה. נסמן:

$$V = [V_1, \dots, V_n]^T, R = [r_1, \dots, r_n]^T$$

$$T = \begin{pmatrix} p_{11} & \dots & p_{1n} \\ \vdots & \ddots & \vdots \\ p_{n1} & \dots & p_{nn} \end{pmatrix}$$

ונקבל משוואה מטריציאלי:

$$V = R + \gamma TV \rightarrow V = R + \gamma TV$$

$$\rightarrow \mathcal{V}^{\pi}(s) = (\mathbb{I}_n - \gamma T)^{-1} R$$

בגלל שהערכים העצמיים של T חסומים על ידי 1, בהכרח יהיה ניתן להפוך את $\mathbb{I}_n - \gamma T$ מה שמבטיח שיהיה פתרון למשוואה, ופתרון זה הוא אף יחיד עבור \mathcal{V}^{π} . כשמוצאים את V ניתן למצוא גם את Q^{π} על ידי הקשר:

$$Q^{\pi}(s, a) = \sum_{s'} p_{\pi}(s, s') \left(\mathcal{R}_{\pi}(s, s') + \gamma \mathcal{V}^{\pi}(s') \right) = \sum_{s'} p_{\pi}(s, s') \left(\mathcal{R}_{\pi}(s, s') + \gamma \sum_{a'} \pi(a'|s') Q^{\pi}(a'|s') \right)$$

Iterative Policy Evaluation

הסיבוכיות של היפוך מטריצה הינו $\mathcal{O}(n^3)$, ועבור n גדול החישוב נהיה מאוד יקר ולא יעיל. כדי לחשב את הפתרון באופן יעיל, ניתן כאמור להשתמש בשיטות איטרטיביות. שיטות אלו מבוססות על אופרטור בלמן, המוגדר באופן הבא:

$$BO(V) = R^\pi + \gamma T^\pi \cdot V$$

ניתן להוכיח שאופרטור זה הינו העתקה מכווצת (contractive mapping), כלומר הוא מקיים את התנאי:

$$\forall x, y: \|f(x) - f(y)\| < \gamma \|x - y\| \text{ for } 0 < \gamma < 1$$

במילים: עבור שני וקטורים במרחב, אופרטור $f(\cdot)$ ומספר γ החסום בין 0 ל-1, אם נפעיל את האופרטור על כל אחד מהווקטורים ונחשב את נורמת ההפרש, נקבל מספר קטן יותר מאשר הנורמה בין הווקטורים כפול הפקטור γ . אופרטור המקיים תכונה זו הינו העתקה מכווצת, כיוון שנורמת ההפרש של האופרטור על שני וקטורים קטנה מנורמת ההפרש בין הווקטורים עצמם. הוכחה:

$$\|f(u) - f(v)\|_\infty = \|R^\pi + \gamma T^\pi \cdot v - (R^\pi + \gamma T^\pi \cdot u)\|_\infty = \|\gamma T^\pi(v - u)\|_\infty$$

מטריקת אינסוף מוגדרת לפי: $\|u - v\|_\infty = \max_{s \in \mathcal{S}} |u(s) - v(s)|$. לכן נוכל לרשום:

$$\|\gamma T^\pi(v - u)\|_\infty \leq \|\gamma T^\pi\| \|u - v\|_\infty$$

הביטוי $\|T^\pi\|$ למעשה סוכם את כל ערכי מטריצת המעברים, לכן הוא מסתכם ל-1, ונקבל:

$$= \gamma \|u - v\|_\infty$$

ובכך הוכחנו את הדרוש.

לפי משפט נקודת השבת של בנך, להעתקה מכווצת יש נקודת שבת (fixed point) יחידה המקיימת $x = f(x)$ וסדרה $x_{t+1} = f(x_t)$ המתכנסת לאותה נקודת שבת. לכן נוכל להשתמש באלגוריתם איטרטיבי עבור \mathcal{V}^π שיביא אותנו לנקודת שבת, ולפי המשפט זוהי נקודת השבת היחידה וממילא הגענו להתכנסות. בפועל, נשתמש באלגוריתם האיטרטיבי הבא:

$$V_{k+1} = BO(V_k) = R^\pi + \gamma T^\pi \cdot V_k$$

נסתכל על הדוגמה הבאה:

$$T^\pi = \begin{pmatrix} 0.8 & 0.1 & 0.1 & 0 & 0 \\ 0.1 & 0.8 & 0.1 & 0 & 0 \\ 0 & 0.1 & 0.8 & 0.1 & 0 \\ 0 & 0 & 0.1 & 0.8 & 0.1 \\ 0 & 0 & 0.1 & 0.1 & 0.8 \end{pmatrix}, \mathcal{R}^\pi = \begin{pmatrix} 0.1 \\ 1.3 \\ 3.4 \\ 1.9 \\ 0.4 \end{pmatrix}, \gamma = 0.9$$

באמצעות השיטה האיטרטיבית נקבל:

$$V_0 = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}, V_1 = \begin{pmatrix} 0.1 \\ 1.3 \\ 3.4 \\ 1.9 \\ 0.4 \end{pmatrix}, V_2 = \begin{pmatrix} 0.6 \\ 2.6 \\ 6.1 \\ 3.7 \\ 1.2 \end{pmatrix}, \dots, V_{10} = \begin{pmatrix} 7.6 \\ 10.8 \\ 18.2 \\ 16.0 \\ 9.8 \end{pmatrix}, \dots, V_{50} = \begin{pmatrix} 14.5 \\ 17.1 \\ 26.4 \\ 26.8 \\ 18.4 \end{pmatrix}, V^\pi = \begin{pmatrix} 14.7 \\ 17.9 \\ 26.6 \\ 27.1 \\ 18.7 \end{pmatrix}$$

ניתן לשים לב שאחרי 50 איטרציות הפתרון המתקבל בצורה האיטרטיבית קרוב מאוד לפתרון המתקבל בצורה האנליטית.

Policy Iteration (PI)

חישוב ה-Value function מאפשר לנו לחשב את ערכו של $\mathcal{V}^\pi(s)$ עבור כל s , אך הוא אינו מבטיח שנגיע לאסטרגיה האופטימלית. נניח והצלחנו לחשב את $\mathcal{V}^\pi(s)$ וממנו אנו יודעים לגזור אסטרגיה, עדיין יתכן שקיימת פעולה a שיותר משתלמת מאשר הפעולה המוצעת לפי האסטרגיה הנגזרת מ- $\mathcal{V}^\pi(s)$. באופן פורמלי ניתן לתאר זאת בצורה פשוטה – נניח שחישבנו את $\mathcal{V}^\pi(s)$ ואת $Q^\pi(s, a)$ יתכן וקיימת פעולה עבורה:

for such s, a : $Q^\pi(s, a) > V^\pi(s)$

אם קיימת פעולה כזו, אז יש תלם לבחור בה ולאחר מכן לחזור לפעול בהתאם לאסטרטגיה $\pi(a|s)$ הנגזרת מחישוב ה-Value function. למעשה, ניתן לחפש את כל הפעולות עבורן כדאי לבצע פעולה מסיימת עבורה התגמול יהיה גבוה יותר מאשר האסטרטגיה של $V^\pi(s)$. באופן פורמלי יותר, נרצה להגדיר אסטרטגיה דטרמיניסטית, עבורה בהסתברות 1 ננקוט בכל מצב s בפעולה הכי כדאית a :

$$\pi'(a|s) = 1 \text{ for } a = \arg \max_{a'} Q^\pi(s, a')$$

נשים לב שרעיון זה הוא בעצם להשתמש באסטרטגיה גרידית – בכל מצב לנקוט בפעולה הכי משתלמת בטווח של צעד יחיד, ואז להמשיך עם האסטרטגיה הנתונה. השאלה העולה היא כמובן – מדוע זה בהכרח נכון? כלומר, האם הרעיון שאומר שלא משנה באיזה מצב אנו נמצאים, הבחירה של הפעולה האופטימלית בהכרח תוביל לקבלת אסטרטגיה יותר טובה מאשר האסטרטגיה הנוכחית? בכדי להוכיח זאת ננסח זאת כמשפט:

בהינתן 2 אסטרטגיות π, π' , כאשר π' דטרמיניסטית, אז כאשר $Q^\pi(s, \pi'(s)) > V^\pi(s)$ בהכרח לכל s יתקיים: $V^{\pi'}(s) > V^\pi(s)$. ראשית נפתח לפי הגדרה:

$$V^\pi(s) < Q^\pi(s, \pi'(s)) = \mathbb{E}_\pi[r_{t+1} + \gamma \cdot V^\pi(s_{t+1}) | s_t = s, a_t = \pi'(s)]$$

כיוון שהאסטרטגיה הינה דטרמיניסטית, הפעולה הנבחרת אינה רנדומלית ביחס ל- π' , ולכן נוכל לרשום:

$$= \mathbb{E}_{\pi'}[r_{t+1} + \gamma \cdot V^\pi(s_{t+1}) | s_t = s]$$

כעת לפי אותו אי שוויון שבהנחה נוכל לבצע את אותו חישוב גם לצעד הבא s_{t+2} :

$$< \mathbb{E}_{\pi'}[r_{t+1} + \gamma \cdot Q^\pi(s_{t+1}, \pi'(s_{t+1})) | s_t = s]$$

וזו שוב שווה ל:

$$= \mathbb{E}_{\pi'}[r_{t+1} + \gamma \cdot r_{t+2} + \gamma^2 \cdot V^\pi(s_{t+2}) | s_t = s]$$

וכך הלאה, ולמעשה הוכחנו את הדרוש – נקיטת הפעולה הכי יעילה בכל מצב תמיד תהיה יותר טובה מהפתרון של $V^\pi(s)$.

כעת יש בידינו שתי טכניקות שאנו יודעים לבצע:

Evaluation (E) – בהינתן אסטרטגיה מסוימת נוכל לפתור את משוואות בלמן ולקבל את $V^\pi(s)$ ו- $Q^\pi(s, a)$.

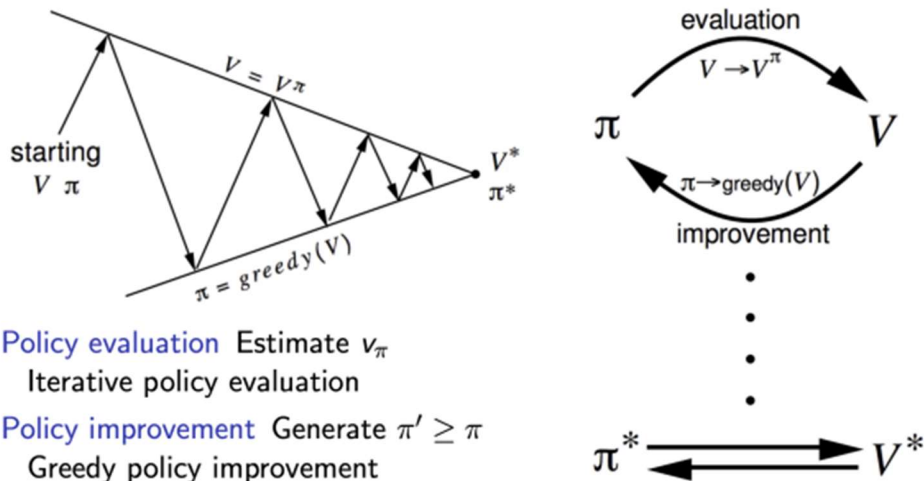
Improve (I) – בהינתן הערך של Value function, נוכל לשפר אותה באמצעות בחירה גרידית של פעולה.

בעזרת טכניקות אלו ניתן להתחיל מאסטרטגיה רנדומלית, ואז לבצע איטרציות המורכבות משתי הטכניקות האלה באופן הבא:

$$\pi_0 \xrightarrow{E} V^{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} V^{\pi_1} \xrightarrow{I} \pi_2 \xrightarrow{E} \dots$$

תהליך זה נקרא Policy iteration – בכל צעד בו יש לנו אסטרטגיה נפתור עבורה משוואות בלמן ובכך נחשב את ה-Value function שלה, ולאחר מכן נשפר את האסטרטגיה באמצעות policy improvement, שכאמור מבצע בחירה גרידית שבטווח הקצר טובה יותר מאשר ה-Value function שחישבנו. ניתן להוכיח שאחרי מספר סופי של איטרציות האסטרטגיה תתכנס לנקודת שבת (fixed point), ואז הפעולה הבאה לפי האסטרטגיה תהיה זהה לבחירה הגרידית:

$$\pi(s) = \arg \max_a Q^\pi(s, a) = \pi'(s)$$



Policy evaluation Estimate v_π
 Iterative policy evaluation
 Policy improvement Generate $\pi' \geq \pi$
 Greedy policy improvement

איור 11.4 Policy iteration – ביצוע איטרציות של Policy evaluation ו-Policy improvement על מנת למצוא בכל שלב את ה-value function ולשפר אותו באמצעות בחירה גרידית.

Bellman optimality equations

השלב הבא בשימוש ב-Policy iteration הוא **להוכיח** שהאסטרטגיה אליה מתכנסים הינה אופטימלית. נסמן את נקודת השבת ב- π^* ונקבל את הקשר הבא:

$$v^{\pi^*}(s) \equiv V^*(s) = \max_a Q^*(s, a) = \max_a \sum_{s'} p_\pi(s, s') (\mathcal{R}_\pi(s, s') + \gamma \cdot V^*(s'))$$

ובאופן דומה:

$$Q^*(s, a) = \sum_{s'} p_\pi(s, s') (\mathcal{R}_\pi(s, s') + \gamma \cdot \max_{a'} Q^*(s', a'))$$

משוואות אלה נקראות Bellman optimality equation. ניתן לשים לב שהן מאוד דומות למשוואות בלמן מהן יצאנו, אך במקום התוחלת שהייתה לנו בהתחלה, כעת יש \max . נרצה להראות שהפתרון של משוואות אלה הוא ה-Value של האסטרטגיה האופטימלית. ננסח את הטענה באופן הבא:

אסטרטגיה הינה אופטימלית אם ורק אם היא מקיימת את Bellman optimality equation. כיוון אחד להוכחה הוא טריוויאלי – אם האסטרטגיה הינה אופטימלית אז היא בהכרח מקיימת את משוואות האופטימליות, כיוון שהראינו שהן מתקבלות מנקודת השבת אליה האיטרציות מתכנסות. אם האסטרטגיה לא הייתה אופטימלית אז היה ניתן לשפר עוד את האסטרטגיה ולא היינו מגיעים עדיין לנקודת השבת. בשביל להוכיח את הכיוון השני נשתמש שוב ברעיון של העתקה מכווצת. נגדיר את האופרטור הבא:

$$BV(s) = \max_a \sum_{s'} p_\pi(s, s') (\mathcal{R}_\pi(s, s') + \gamma \cdot V(s))$$

ניתן להראות שאופרטור זה הינו העתקה מכווצת, וממילא לפי המשפט של בנך יש לו נקודת שבת יחידה. כיוון שהראינו ששימוש ב-Policy iteration מביא את האסטרטגיה לנקודת שבת מסוימת, נוכל לצרף לכך את העובדה שהאופרטור שהגדרנו הינו העתקה מכווצת וממילא נקבל שאותה נקודת שבת הינה יחידה, וממילא אופטימלית.

Value Iteration

הראנו שבעזרת שיטת Policy iteration ניתן להגיע לאסטרטגיה אופטימלית, אך התהליך יכול להיות איטי. ניתן לנקוט גם בגישה יותר ישירה ולנסות לחשב באופן ישיר את הפתרון של משוואות האופטימליות של בלמן (ופתרון הינו אופטימלי כיוון שהראינו שהפתרון הוא נקודת שבת יחידה). נתחיל עם פתרון רנדומלי V_0 ולאחר מכן נצבע איטרציות באופן הבא עד שנגיע להתכנסות:

$$\mathcal{V}_{k+1} = \max_a \sum_{s'} p_\pi(s, s') (\mathcal{R}_\pi(s, s') + \gamma \cdot \mathcal{V}_k(s'))$$

נשים לב שבשיטה זו אין לנו מידע לגבי האסטרטגיה אלא רק חישובנו את ה-Value function, אך ממנה ניתן לגזור את Q ואז לבחור באסטרטגיה גרידית, שהינה במקרה זה גם אופטימלית:

$$\pi(s) = \arg \max_a Q^\pi(s, a)$$

ניתן להראות כי בשיטה זו ההתכנסות מהירה יותר ודרושות פחות איטרציות מהשיטה הקודמת, אך כל איטרציה יותר מורכבת.

Limitations

לשתי השיטות – Policy iteration ו-Value iteration – יש שני חסרונות מרכזיים:

1. הן דורשות לדעת את המודל והסביבה באופן שלם ומדויק.
2. הן דורשות לעדכן בכל שלב את כל המצבים בו זמנית. עבור מערכות עם הרבה מצבים, זה לא מעשי.

11.1.3 Learning Algorithms

בפרק הקודם הוסבר כיצד ניתן לחשב את האסטרטגיה האופטימלית וערך ההחזרה בהינתן מודל מרקובי. השתמשנו בשתי הנחות עיקריות על מנת להתמודד עם הבעיה:

1. Tabular MDP – הנחנו שהבעיה סופית ולא גדולה מדי, כך שנוכל לייצג אותה בזיכרון ולפתור אותה.
 2. Known environment – הנחנו שהמודל ידוע לנו, כלומר נתונה לנו מטריצת המעברים שקובעת מה הסיכוי לעבור ממצב s למצב s' כשנוקטים בפעולה a (סימנו את זה בתור $\mathcal{P}_{ss'}^a = p_\pi(s, s')$, ובנוסף נתון לנו מה ה-reward המתקבל עבור כל action (סימנו את זה בתור $\mathcal{R}_{ss'}^a = \mathcal{R}_\pi(s, s')$).
- בעזרת שתי ההנחות פיתחנו את משוואות בלמן, כאשר היו לנו שני צמדים של משוואות. משוואות בלמן עבור אסטרטגיה נתונה נכתבות באופן הבא:

$$\mathcal{V}^\pi(s) = \sum_{a, s'} \pi(a|s) \mathcal{P}_{ss'}^a (\mathcal{R}_{ss'}^a + \gamma \cdot \mathcal{V}^\pi(s'))$$

$$Q^\pi(s, a) = \sum_{s'} \mathcal{P}_{ss'}^a \left(\mathcal{R}_{ss'}^a + \gamma \sum_{a'} Q^\pi(s', a') \right)$$

ובנוסף פיתחנו את המשוואות עבור הפתרון האופטימלי:

$$\mathcal{V}^*(s) = \max_a \sum_{s'} \mathcal{P}_{ss'}^a (\mathcal{R}_{ss'}^a + \gamma \cdot \mathcal{V}^*(s'))$$

$$Q^*(s, a) = \sum_{s'} \mathcal{P}_{ss'}^a \left(\mathcal{R}_{ss'}^a + \gamma \max_{a'} Q^*(s', a') \right)$$

הראינו שתי דרכים להגיע לפתרון האופטימלי:

1. Policy iteration המורכב מ-Policy evaluation ולאחריו Policy improvement.
2. Value iteration – פתרון משוואות בלמן ישיר בעזרת איטרציות על ה-Value function.

כאמור, דרכי פתרון אלו מניחים שהמודל ידוע, ובנוסף שמרחב המצבים אינו גדול מדי ויכול להיות מיוצג בזיכרון. האתגר האמיתי מתחיל בנקודה בה לפחות אחת מהנחות אלה אינה תקפה, ולמעשה פה מתחיל התפקיד של אלגוריתמי RL. עיקר ההתמקדות של אלגוריתמים אלו יהיה למצוא באופן יעיל את האסטרטגיה האופטימלית כאשר לא נתונים הפרמטרים של המודל, ואז צריך לשערך אותם (Model-based learning) או למצוא דרך אחרת לחישוב האסטרטגיה האופטימלית ללא שימוש במודל (Model free learning). אם למשל יש משחק בין משתמש לבין המחשב, אלגוריתמים השייכים ל-Model based learning ינסו ללמוד את המודל של המשחק או להשתמש במודל

קיים, ובעזרת המודל הם ינסו לבחון כיצד יגיב המשתמש לכל תור שהמחשב יבחר. לעומת זאת אלגוריתמים מסוג Model free learning לא יתעניינו בכך, אלא ינסו ללמוד ישירות את האסטרטגיה הטובה ביותר עבור המחשב.

היתרון המשמעותי של אלגוריתמים המסתכלים על המודל של הבעיה (Model-based) נובע מהיכולת לתכנן מספר צעדים קדימה, כאשר עבור כל בחירה של פעולה המודל בוחן את התגובות האפשריות, את הפעולות המתאימות לכל תגובה, וכך הלאה. דוגמה מפורסמת לכך היא תוכנת המחשב AlphaZero שאומנה לשחק משחקי לוח כגון שחמט או גו. במקרים אלו המודל הוא המשחק והחוקים שלו, והתוכנה משתמשת בידע הזה בכדי לבחון את כל הפעולות והתגובות למשך מספר צעדים רב ובחירה של הצעד הטוב ביותר.

עם זאת, בדרך כלל אף בשלב האימון אין לסוכן מידע חיצוני מהו הצעד הנכון באופן אולטימטיבי, ועליו ללמוד רק מהניסיון. עובדה זו מציבה כמה אתגרים, כאשר העיקרי ביניהם הוא הסכנה שהאסטרטגיה הנלמדת תהיה טובה רק עבור המקרים אותם ראה הסוכן, אך לא תתאים למקרים חדשים שיבואו. אלגוריתמים שמחפשים באופן ישיר את האסטרטגיה האופטימלית אמנם לא משתמשים בידע שיכול להגיע מבחינת צעדים עתידיים, אך הם הרבה יותר פשוטים למימוש ולאמון.

באופן מעט יותר פורמלי ניתן לנסח את ההבדל בין הגישות כך: גישת Model-based learning מנסה למצוא את הפרמטרים המגדירים את המודל $\{S, A, T, R\}$ ואז בעזרתם לחשב את האסטרטגיה האופטימלית (למשל בעזרת משוואות בלמן). הגישה השנייה לעומת זאת לא מעוניינת לחשב במפורש את הפרמטרים של המודל אלא למצוא באופן ישיר את האסטרטגיה האופטימלית $\pi(a_t|s_t)$ שעבור כל מצב קובעת באיזה פעולה לנקוט. ההבדל בין הגישות נוגע גם לפונקציית המחיר לה נרצה למצוא אופטימום.

בכל אחד משני סוגי הלמידה יש אלגוריתמים שונים, כאשר הם נבדלים אחד מהשני בשאלה מהו האובייקט אותו מעוניינים ללמוד.

Model-free learning

בגישה זו יש שתי קטגוריות מרכזיות של אלגוריתמים:

- א. Policy Optimization – ניסוח האסטרטגיה כבעיית אופטימיזציה של מציאת סט הפרמטרים θ המקסם את $\pi_\theta(a|s)$. פתרון בעיה זו יכול להיעשות באופן ישיר על ידי שיטת Gradient Ascent עבור פונקציית המחיר $J(\pi_\theta) = \mathbb{E}[R(\tau)]$, או בעזרת קירוב פונקציה זו ומציאת מקסימום עבורה.
- ב. Q-learning – שערך $Q^*(s, a)$ על ידי $Q_\theta(s, a)$. מציאת המשערך האופטימלי יכולה להתבצע על ידי חיפוש θ שיספק את השערך הטוב ביותר שניתן למצוא, או על ידי מציאת הפעולה שתמקסם את המשערך:
$$a(s) = \arg \max_a Q_\theta(s, a)$$

השיטות המנסות למצוא אופטימום לאסטרטגיה הן לרוב on-policy, כלומר כל פעולה נקבעת על בסיס האסטרטגיה המעודכנת לפי הפעולה הקודמת. Q-learning לעומת זאת הוא לרוב אלגוריתם off-policy, כלומר בכל פעולה ניתן להשתמש בכל המידע שנצבר עד כה. היתרון של שיטות האופטימיזציה נובע מכך שהן מנסות למצוא באופן ישיר את האסטרטגיה הטובה ביותר, בעוד שאלגוריתם Q-learning רק משערך את $Q^*(s, a)$, ולעיתים השערך לא מספיק ואז התוצאה המתקבלת אינה מספיק טובה. מצד שני, כאשר השערך מוצלח, הביצועים של Q-learning טובים יותר, כיוון שהשימוש במידע על העבר מנוצל בצורה יעילה יותר מאשר באלגוריתמים המבצעים אופטימיזציה של האסטרטגיה. שתי הגישות האלה אינן זרות לחלוטין, וישנם אלגוריתמים שמנסים לשלב בין הרעיונות ולנצל את החוזקות והיתרונות שיש לכל גישה.

Model-based learning

גם בגישה זו יש שתי קטגוריות מרכזיות של אלגוריתמים:

- א. Model-based RL with a learned model – אלגוריתמים המנסים ללמוד הן את המודל עצמו והן את ה-Value function או את האסטרטגיה π .
- ב. Model-based RL with a known model – אלגוריתמים המנסים למצוא את ה-Value function ו/או את האסטרטגיה כאשר המודל עצמו נתון.

ההבדל בין הקטגוריות טמון באתגר איתו מנסים להתמודד. במקרים בהם המודל ידוע, הממד של אי הוודאות לא קיים, ולכן ניתן להתמקד בביצועים אסימפטוטיים. במקרים בהם המודל אינו ידוע, הדגש העיקרי הוא על למידת המודל.

11.2 Model Free Prediction

לאחר שסקרנו בפרק המבוא את הבסיס המתמטי של בעיות RL והצגנו את משוואות בלמן ופתרון, בפרקים הבאים נציג גישות שונות להתמודדות עם בעיות RL עבורן פתרונות אלה אינם מספיקים – או מפני שהמודל אינו ידוע או מפני שהן Scale גדול יותר מזה שניתן לפתור באמצעות משוואות בלמן. בפרק זה נציג שתי שיטות הבאות להתמודד עם מקרים בהם המודל אינו ידוע (כלומר האלגוריתם הינו Model-Free), והדרך שלהן להתמודד עם אתגר זה הינו **לשערך** את האסטרטגיה האופטימלית בדרכים אחרות שאינן מצריכות את ידיעת המודל.

11.2.1 Monte-Carlo (MC) Policy Evaluation

האלגוריתם הראשון אותו נציג הינו Monte Carlo, והוא מציע דרך לשערך את ה-Value function בלי לדעת את המודל. ראשית נסביר בקצרה מהו אלגוריתם Monte Carlo ואז נראה כיצד ניתן ליישם אותו בבעיות RL.

נניח ונרצה לשערך תוחלת של פונקציית התפלגות כלשהיא – $\mathbb{E}_p[f(x)]$. התוחלת יכולה להיות סכום או אינטגרל שקשה מאוד לחשב. ניתן לשערך את התוחלת על ידי דגימות רנדומליות מההתפלגות וחישוב הממוצע של הדגימות:

$$x_1, \dots, x_n \sim p(x)$$

$$\mathbb{E}_p[f(x)] \approx \frac{1}{n} \sum_i f(x_i)$$

לפי חוק המספרים הגדולים הממוצע של הדגימות מתכנס לתוחלת. משערך זה הינו חסר הטיה, ובנוסף השונות שלה קטנה ביחס לינארי לכמות הדגימות:

$$\mathbb{E} \left[\frac{1}{n} \sum_i f(x_i) \right] = \mathbb{E}[f(x)]$$

$$\text{Var} \left[\frac{1}{n} \sum_i f(x_i) \right] = \frac{\text{Var}[f(x)]}{n}$$

כאמור לעיל, ה-Value function הינה תוחלת עבור אסטרטגיה נתונה π , כאשר מתחילים ממצב s :

$$v^\pi(s) = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s \right]$$

אמנם התוחלת היא על סכום אינסופי, אך עבור מקרים רבים נוכל להניח שהריצה היא סופית (Episodic MDP). בפועל נבצע את הפעולה הבאה: עבור אסטרטגיה נתונה π , נייצר ממנה ריצה של T צעדים, ואז ניקח מצב מסוים ונסתכל על כל ה-rewards שמתקבלים בריצה זו החל ממנו. באופן הזה קיבלנו value עבור הערך של אותו מצב. חזרה על אותה פעולה שוב ושוב תייצר ריצות שונות וממילא ערכים שונים למצב מסוים, ומיצוע על פני הערכים ייתן לנו שערך לערך האמיתי של אותו מצב. נעיר בהערת אגב שכיוון שמצבים יכולים לחזור על עצמם, נוצרת בעיה שבריצה כזו המצבים אינם בלתי תלויים. בכדי להתגבר על כך, אם מצב חוזר על עצמו יותר מפעם אחת מאפשרים לדגום רק את המופע הראשון של אותו מצב ולא את יתר המופעים (ישנן עוד דרכים להתגבר על כך, אך זוהי הדרך פשוטה ביותר). באופן פורמלי, נניח ויש לנו ריצה של T צעדים:

$$S_1, A_1, R_1, \dots, S_{T-1}, A_{T-1}, R_{T-1}, S_T$$

אז דגימה אחת מתוך ההתפלגות $p_\pi(\sum_{k=0}^{\infty} r_{t+k+1} | s_t = S_t)$ תראה באופן הבא:

$$G_t = R_{t+1} + \gamma R_{t+2} + \dots + \gamma^{T-t-1} R_T = \sum_{k=0}^{T-t-1} \gamma^k R_{t+k+1}$$

למשל, הדגימה G_1 תהיה מורכבת מכל ה-rewards שהגיעו לאחר הצעד הראשון: $R_1 + \gamma R_2 + \dots + \gamma^{T-1} R_T$. הסכום הזה ייתן לנו value עבור אותו מצב ממנו התחלנו (S_1), ועל ידי שערך אותו מצב שוב ושוב ביחס לריצות שונות נוכל לקבל ערכים שונים, ולאחר מכן למצע אותם בכדי לשערך את ה-value האמיתי של אותו מצב S_1 .

באופן פורמלי, העדכון של מצב לאחר כל דגימה נראה כך:

#sample of s_t : $N(S_t) = N(S_t) + 1$

$$\text{update the value of } s_t: \mathcal{V}(s_t) = \mathcal{V}(s_t) + \frac{1}{N(S_t)} (G_t - \mathcal{V}(s_t))$$

ולאחר הרבה דגימות השערוך מתקבל על ידי התוחלת שלהן:

$$\mathcal{V}(s) = \mathbb{E}_\pi[G_t | s_t = s]$$

באופן סכמתי ניתן לתאר את האלגוריתם באופן הבא:

First-visit MC prediction, for estimating $V \approx v_\pi$

Initialize:

$\pi \leftarrow$ policy to be evaluated
 $V \leftarrow$ an arbitrary state-value function
 $Returns(s) \leftarrow$ an empty list, for all $s \in \mathcal{S}$

Repeat forever:

Generate an episode using π
 For each state s appearing in the episode:
 $G \leftarrow$ the return that follows the first occurrence of s
 Append G to $Returns(s)$
 $V(s) \leftarrow$ average($Returns(s)$)

איור 11.5 אלגוריתם MC עבור שערוך ה-Value function בהינתן Policy. עבור כל מצב מאתחלים רשימה ריקה, ולאחר מכן מייצרים המון ריצות שונות. עבור כל ריצה, עוברים על כל המצבים ובודקים מה ה- G שלהם, ומוסיפים אותו לרשימה של אותו מצב. לבסוף מחשבים את ה-value של כל מצב על ידי מיצוע הרשימה (=ערכי G השונים) של אותו מצב.

יש מספר יתרונות לשיטה זו: היא מספקת משערוך חסר הטיה עבור ה-Value function, מבטיחה התכנסות אחרי מספיק איטרציות, ובנוסף ניתן לשערך באמצעותה את השגיאה. אפשר גם באותה דרך לשערך גם את $Q^\pi(s, a)$ אך זה יהיה יותר רועש וידרוש יותר דגימות. מן הצד השני יש גם חסרונות לשיטה זו: ראשית, היא מתאימה רק ל-Episodic MDP ולא לריצות אינסופיות. עם בעיה זו ניתן להתמודד בקלות כיוון שעבור בעיה אינסופית ניתן לקחת ריצה סופית ולחסום את השגיאה באמצעות γ . שנית, ההתכנסות יכולה להיות מאוד איטית, ובנוסף השונות יחסית גבוהה.

11.2.2 Temporal Difference (TD) – Bootstrapping

במקום להסתכל על ריצה שלמה, ניתן אחרי כל צעד לעדכן את ה-Value. ממשוואת בלמן ניתן להראות שהביטוי $R_{t+1} + \gamma \mathcal{V}^\pi(S_{t+1})$ הינו משערוך חסר הטיה עבור $\mathcal{V}^\pi(S_t)$. אי אפשר להשתמש במשערוך זה כמו שהוא, כיוון שאנחנו לא יודעים את ה-Value function, וממילא הביטוי $\mathcal{V}^\pi(S_{t+1})$ לא ידוע. בשביל בכל זאת לשערך את $\mathcal{V}^\pi(S)$ באמצעות אותו משערוך, ניתן להחליף את $\mathcal{V}^\pi(S_{t+1})$ ב- $\mathcal{V}(S_{t+1})$, ולבצע את השערוך באופן הבא:

$$\mathcal{V}^\pi(S_t) = R_{t+1} + \gamma \mathcal{V}(S_{t+1})$$

הרעיון מאחורי השימוש הזה הוא להיעזר במידע שיש לנו מ- R_t . נניח וננחש ערך כלשהוא עבור $\mathcal{V}^\pi(S_t)$ וניחוש זה יהיה גרוע. אפשר מעט לשפר את הניחוש באמצעות ניחוש $\mathcal{V}(S_{t+1})$ ושימוש ב-reward R_{t+1} שהתקבל עבור אותו מצב S_t , שבעצם מספק מידע כלשהוא על מצב זה. שערוך זה עדיף על ניחוש מוחלט כיוון שהאלמנט של הניחוש מקבל משקל נמוך יותר עקב המכפלה ב- γ , ויש יותר משקל ל- R_{t+1} שמספק מידע אמיתי על המצב S_t . צריך לשים לב שאנו מנסים לשערך את ה-Value function מתוך הערכים שלה בעצמה. מאתחלים את כל הערכים במספרים כלשהם (למשל – וקטור של 0), ואז עוברים צעד צעד ומעדכנים את הניחושים בעזרת התגמולים. אינטואיטיבית זה נראה מעט משונה, אך מסתבר שפתרון זה הוא אחד הכלים החזקים בבעיות RL. באופן פורמלי האלגוריתם נראה כך:

Tabular TD(0) for estimating v_π

Input: the policy π to be evaluated
 Initialize $V(s)$ arbitrarily (e.g., $V(s) = 0$, for all $s \in \mathcal{S}^+$)
 Repeat (for each episode):
 Initialize S
 Repeat (for each step of episode):
 $A \leftarrow$ action given by π for S
 Take action A , observe R, S'
 $V(S) \leftarrow V(S) + \alpha [R + \gamma V(S') - V(S)]$
 $S \leftarrow S'$
 until S is terminal

איור 11.6 אלגוריתם Temporal Difference (TD) עבור שערך ה-Value function בהינתן Policy. מנחשים ערך עבור כל מצב ואז באופן איטרטיבי משפרים את הניחושים בעזרת שערך התלוי ב-reward ובערך המצב הבא.

מגדירים את השגיאה של ה-TD כהפרש שבין הניחוש עבור ערך המצב לבין השערך שלו:

$$\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$$

בשונה משערך MC, השערך בשיטת TD הוא בעל הטיה, אך השונות קטנה יותר. בנוסף, כיוון שבכל צעד מבצעים שיפור לערך של מצב, תהליך השערך יותר מהיר מאשר ב-MC. הרבה פעמים מוסיפים פרמטר α לשערך (כפי שמופיע באיור 11.6):

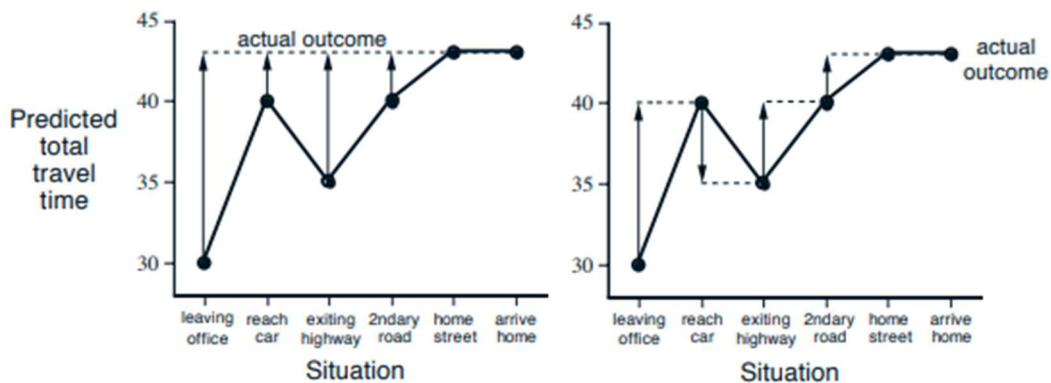
$$V(S_t) = V(S_t) + \alpha \cdot [R_{t+1} + \gamma V(S_{t+1}) - V(S_t)]$$

עבור ערכי α מתאימים, המשערך מתכנס לתוחלת האמיתית.

ניתן דוגמה שתמחיש את שיטת MC ושיטת TD ואת היחס ביניהם: נהג יצא מהמשרד שלו ונסע לביתו, ובדרך הוא ניסה לשערך את זמן ההגעה שלו וקיבל את הזמנים הבאים:

מצב	כמה זמן עבר	שערך הזמן שנותר לנסיעה	שערך זמן הנסיעה הכולל
יציאה מהמשרד	0	30	30
הליכה למכונית תחת גשם	5	35	40
הגעה לכביש מהיר	20	15	35
נסיעה מאחורי משאית	30	10	40
הגעה לרחוב של הבית	40	3	43
הגעה הביתה	43	0	43

נשרטט את שני המשערכים:



איור 11.7 שערך MC (משמאל) ושערך TD (מימין) ביחס לתצפית הנתונה.

כיוון ששיטת MC מספקת ערך לאחר ריצה שלמה, אז ניקח את זמן ההגעה בפועל של הנהג וניתן את הערך הזה לכל מצבי הביניים. שיטת TD לעומת זאת מעדכנת את הערך בכל מצב בהתאם למצב הבא. ניתן לראות שהשונות בשערך TD קטנה מזו שהתקבלה בשערך MC.

ניתן להסתכל על שיטת TD כבעיית רגרסיה "דינמית":

עבור כל צעד נרצה ש- $\mathcal{V}(S_t)$ יהיה שווה למשוואות בלמן, כלומר נרצה לשערך את $\mathcal{V}(S_t)$ כך שיתקיים:

$$\mathcal{V}(S_t) = \mathbb{E}_\pi[R_{t+1} + \gamma\mathcal{V}(S_{t+1})|s_t = S_t]$$

עבור בעיה זו נוכל להגדיר פונקציית מחיר (Loss) מתאימה:

$$L = \frac{1}{2} (\mathbb{E}_\pi[R_{t+1} + \gamma\mathcal{V}(S_{t+1})|s_t = S_t] - \mathcal{V}(S_t))^2$$

את הפונקציה הזו ניתן למזער על ידי דגימות סטוכסטיות של $R_{t+1} + \gamma\mathcal{V}(S_{t+1})$. נשים לב לדבר חשוב – המטרה שלנו היא לשערך את ההווה באמצעות העתיד ולא להיפך, כיוון שהעתיד הוא בעל יותר מידע – הוא ראה reward ומצב חדש. הבחנה זו משפיעה על איך שאנחנו רוצים שפונקציית המחיר תתנהג – אנחנו רוצים ש- $\mathcal{V}(S_t)$ יתקרב לערך של $\mathbb{E}_\pi[R_{t+1} + \gamma\mathcal{V}(S_{t+1})|s_t = S_t]$ ולא להיפך. בדוגמה של הנהג שמשערך את זמן הנסיעה – הוא רוצה לעדכן את השערוך של כל מצב בהתאם למצב הבא. אם למשל ההערכה שלו בזמן t הינה 30 דקות ואז הוא מגיע לפקק ומעדכן את ההערכה ל-35 דקות, אז הוא ירצה לתקן את השערוך הקודם ($\mathcal{V}(S_t)$) כך שיהיה דומה לשערוך הנוכחי ($R_{t+1} + \gamma\mathcal{V}(S_{t+1})$), ולא לעדכן את השערוך הנוכחי כך שיהיה דומה לקודם. בכדי לדאוג לכך, נתייחס לעתיד כקבוע ולא נחשב עבורו גרדיאנט (למרות ש- \mathcal{V} מופיע בו). באופן פורמלי נוכל לנסח זאת כך:

$$T(S_t) = \mathbb{E}_\pi[R_{t+1} + \gamma\mathcal{V}(S_{t+1})|s_t = S_t]$$

$$L = \frac{1}{2} (T(S_t) - \mathcal{V}(S_t))^2$$

ואז הגרדיאנט יהיה:

$$\frac{\partial L}{\partial \mathcal{V}} = T(S_t) - \mathcal{V}(S_t)$$

בהתאם לכך, הדגימה $R_{t+1} + \gamma\mathcal{V}(S_{t+1}) - \mathcal{V}(S_t)$ הינה שערוך סטוכסטי לגרדיאנט. כיוון שכל צעד תלוי בצעד הבא, אז המטרה $T(S_t)$ משתנה בכל צעד (אמנם קיבענו אותה בכל צעד יחיד, אך היא עדיין תלויה ב- $\mathcal{V}(S_t)$). במובן הזה בעיית הרגרסיה שהגדרנו הינה "דינמית", כיוון שהמטרה משתנה בכל צעד.

11.3.2 TD(λ)

ננסה לבחון את הקשר בין שתי השיטות שראינו. שערוך TD מבצע דגימה של ריצה ואז מעדכן את הערך של כל מצב בהתאם למצב הבא בלבד:

$$\mathcal{V}(S_t) = \mathcal{V}(S_t) + \alpha \cdot [R_{t+1} + \gamma\mathcal{V}(S_{t+1}) - \mathcal{V}(S_t)]$$

בגלל ההתייחסות למצב אחד בכל פעם השונות של המשערך נמוכה, אך יש הטיה.

שיטת MC לעומת זאת דוגמת ריצה ומעדכנת את הערך של כל מצב בהתאם לכל המצבים שבאים לאחר מכן:

$$G_t = R_{t+1} + \gamma R_{t+2} + \dots + \gamma^{T-t-1} R_T = \sum_{k=0}^{T-t-1} \gamma^k R_{t+k+1}$$

$$\mathcal{V}(s_t) = \mathcal{V}(s_t) + \frac{1}{N(S_t)} (G_t - \mathcal{V}(s_t))$$

משערך זה הינו חסר הטיה, אך עם זאת השונות שלו גבוהה.

נראה כיצד ניתן לחבר בין שני המשערכים ולמצוא שיטה שתהיה אופטימלית מבחינת היחס שבין ההטיה לשונות. ניתן להכליל את שני המשערכים לנוסחה כללית באופן הבא:

$$\mathcal{V}(s_t) = \mathcal{V}(s_t) + \alpha (G_t^{(n)} - \mathcal{V}(s_t))$$

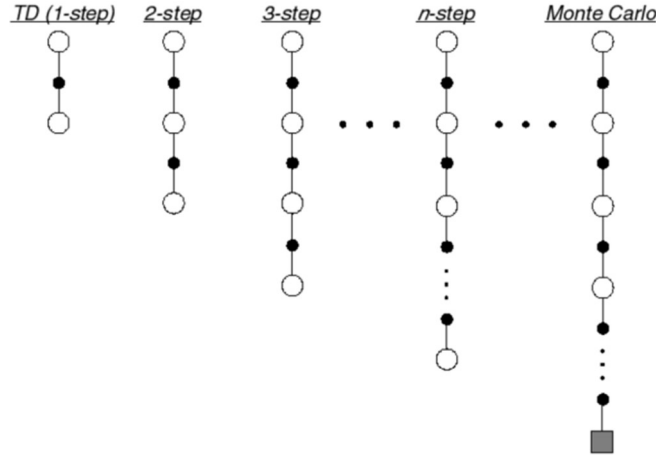
כאשר $G_t^{(n)}$ הוא הסכום של n ה-rewards הבאים:

$$G_t^{(n)} = R_{t+1} + \gamma R_{t+2} + \dots + \gamma^{n-1} R_{t+n} + \gamma^n \mathcal{V}(s_{t+n}) = \sum_{k=0}^{n-t-1} \gamma^k R_{t+k+1}$$

כעת נוכל להגדיר:

$$TD(n) \rightarrow \mathcal{V}(s_t) = \mathcal{V}(s_t) + \alpha (G_t^{(n)} - \mathcal{V}(s_t))$$

ולפי סימון זה נוכל לשים לב שמשערך MC הוא למעשה $TD(n \rightarrow \infty)$ ואילו משערך TD שווה ל- $TD(1)$.



איור 11.8 משערך $TD(n)$ MC הינו המקרה הפרטי עבורו $n \rightarrow \infty$ ואילו משערך TD הינו המקרה הפרטי בו $TD(n=1)$.

אם ניקח $1 < n < \infty$ נקבל משערך "ממוצע" בין MC לבין TD. ניתן להציע גרסה יותר טובה למשערך זה תחת ההנחה שככל שמאורעות סמוכים אחד לשני כך יש להם יותר השפעה. בדומה ל-discount factor שמוריד את ההשפעה של תגמול ככל שהוא יותר רחוק מהמצב הנוכחי, כך גם כאן ניתן לכלל מאורע עתידי משקל הולך וקטן. נדאג שכל המשקלים יסתכמו ל-1 ונקבל את השערוך הבא, הידוע גם בכינוי $TD(\lambda)$:

$$G_t^\lambda = (1 - \lambda) \sum_{n=1}^{\infty} \lambda^{n-1} G_t^{(n)}$$

$$\mathcal{V}(s_t) = \mathcal{V}(s_t) + \alpha (G_t^\lambda - \mathcal{V}(s_t))$$

שערוך זה הוא בעל שונות גדולה יותר מאשר TD, כיוון שהוא מתחשב ביותר צעדים, אך ההטיה שלו קטנה יותר. כאמור, הוא מנסה למצב בין שני המשערכים שראינו ולאזן בין השונות להטיה.

נסכם בקצרה את מה שראינו בפרק זה. התייחסנו למקרים בהם המודל אינו ידוע ואנו רוצים לשערך אותו, והצגנו שלוש גישות המנסות לשערך את המודל בעזרת דגימות סטוכסטיות: TD, MC (שזהו מקרה פרטי של $TD(n)$) וגישה המנסה לשלב בין השתיים – $TD(\lambda)$.

לסיום נעיר ששערוך האסטרטגיה כשלעצמה אינו מספיק, כיוון שהשערוך אמנם מספק אסטרטגיה מסוימת עבור הבעיה, אך היא אינה בהכרח אופטימלית. בפרק הקודם ראינו כיצד בהינתן המודל ניתן לשפר אותו ולמצוא את האסטרטגיה האופטימלית. יכולנו לעשות זאת כיוון שידענו את המודל במלואו, מה שאיפשר לבצע בכל צעד מהלך חמדני ובכך להתכנס לבסוף לאסטרטגיה האופטימלית. במקרים בהם אנו רק משערכים את האסטרטגיה ללא ידיעת המודל, אין לנו בהכרח מידע מספיק טוב עבור כל המצבים. מצבים ופעולות שלא נוסו כלל (או נוסו רק בפעמים נדירות), השערוך עבורם יכול להיות די גרוע, ואז השיפור באמצעות בחירה גרידית לא בהכרח יכול להביא את האסטרטגיה להיות אופטימלית.

11.3 Model Free Control

בפרק הקודם ראינו כיצד ניתן לשערך את המודל באמצעות דגימות סטוכסטיות. שיטות השערוך אפשרו לנו לקבל מידע על המודל, אך הן אינן התייחסו לשאלה האם הוא אופטימלי. בפרק הראשון ראינו כיצד ניתן לקחת מודל או

אסטרטגיה ולהביא אותם לאופטימליות, אך אי אפשר להשתמש בשיטה זו עבור מודל משוער. הסיבה לכך נעוצה שמודל זה לא בהכרח ראה את כל הזוגות האפשריים של המצבים והפעולות, וממילא הוא לא יכול לשפר את הבחירות שלו על סמך אסטרטגיה גרידית. ניקח לדוגמה מקרה בו עבור מצב מסוים האסטרטגיה הינה דטרמיניסטית והפעולה שנבחרת הינה תמיד ללכת למעלה. במקרה כזה אין לנו שום מידע על יתר הפעולות האפשריות במצב זה ולכן המודל שלנו לא שלם, וממילא לא נוכל להשתמש ב-Policy improvement המבוסס על כך שיש לנו מידע על כל המצבים והפעולות.

בכדי להתמודד עם בעיה זו נהיה חייבים לדאוג לכך שנבקר בכל המצבים. כמובן שנוכל לנקוט בגישה פשטנית של אסטרטגיה אקראית, שאחרי מספר מספיק גדול של פעולות קרוב לוודאי שנבקר בכל המצבים. אסטרטגיה זו אמנם טובה לבדיקת מצבים ופעולות חדשים, אך כמובן שהיא רחוקה מלהיות אופטימלית, לכן נרצה להשתמש באסטרטגיה שמצד אחד מנסה להיות אופטימלית ומצד שני יש בה ממד של אקראיות המביא לכך שנבקר גם במצבים שלא היינו מגיעים אליהם לפי האסטרטגיה הנוכחית. בחירת פעולות בהתאם לאסטרטגיה הנוכחית נקראת **Exploitation** ואילו בחירה של מצבים חדשים נקראת **Exploration**, והמטרה שלנו תהיה לאזן בין השניים תחת הדרישות הבאות:

1. ביקור בכל הזוגות של המצבים והפעולות אינסוף פעמים.

2. ככל שמספר הצעדים גדל, כך האסטרטגיה מתכנסת לאסטרטגיה הגרידית:

$$\lim_{k \rightarrow \infty} \pi_k(a|s) = 1 \left[a = \arg \max_{a'} Q(s, a') \right]$$

מצד אחד נרצה לבקר בכל המצבים ומצד שני נרצה שכלל שנעשה יותר צעדים ככה נשפר את האסטרטגיה שלנו. אסטרטגיה המקיימת דרישות אלה נקראת **Greedy in the Limit of Infinite Exploration (GLIE)**, וניתן להוכיח שקיום דרישות אלה מביא את האסטרטגיה להיות אופטימלית. דוגמה לאסטרטגיה פשוטה העונה על הדרישות הינה *greedy* - ϵ - בחירה של האסטרטגיה האופטימלית בהסתברות $(1 - \epsilon)$ ובחירה מצב אחר בהסתברות ϵ . עבור ϵ שהולך וקטן עד ל-0 בקצב שאינו מהיר מדי, אסטרטגיה זו הינה GLIE.

לעיל הראינו כיצד בהינתן מודל ניתן לשפר את האסטרטגיה (Policy improvement) על ידי כך שבחרנו באופן גרידי פעולות. כעת נרצה להראות שבאופן דומה מתקיים אותו רעיון גם עבור *greedy* - ϵ , כלומר שאם השתמשנו בשיטה זו והגענו ל-Value function, אז ניתן בצעד הבא לשפר את ה-Value על ידי אסטרטגיה ϵ -גרידית. אסטרטגיה זו בוחרת באופן גרידי את הפעולה האופטימלית לפי האסטרטגיה הנתונה, אך נותנת לכל יתר הפעולות הסתברות הגדולה או שווה להסתברות שהייתה לפעולה זו בשימוש באסטרטגיה ϵ -גרידית. ננסח את המשפט באופן פורמלי:

If Policy π_i has $\forall s, a: \pi_i(a|s) \geq \frac{\epsilon}{|A|}$ and π_{i+1} is ϵ - greedy w. r. t Q^{π_i} , then $V^{\pi_{i+1}} \geq V^{\pi_i}$

הוכחה: נסתכל על המקרה בו נוקטים צעד אחד גרידי לפי π_{i+1} ואז ממשיכים לפי האסטרטגיה הקודמת π_i :

$$\sum_a \pi_{i+1}(a|s) Q^{\pi_i}(s, a) = \sum_a \frac{\epsilon}{|A|} Q^{\pi_i}(s, a) + (1 - \epsilon) \max_a Q^{\pi_i}(s, a)$$

נרשום את $(1 - \epsilon)$ בצורה אחרת: במקום ϵ נרשום $\sum_a \frac{\epsilon}{|A|}$, ובמקום 1 נרשום $\sum_a \pi_i(a|s)$ (הסכום אכן שווה ל-1 כי סוכמים את כל האפשרויות עבור התפלגות נתונה). נקבל:

$$= \sum_a \frac{\epsilon}{|A|} Q^{\pi_i}(s, a) + \max_a Q^{\pi_i}(s, a) \sum_a \left(\pi_i(a|s) + \frac{\epsilon}{|A|} \right)$$

כעת נחליף את הביטוי $\max_a Q^{\pi_i}(s, a)$ באסטרטגיה עצמה $Q^{\pi_i}(s, a)$. כיוון שמתקיים $Q^{\pi_i}(s, a) \leq \max_a Q^{\pi_i}(s, a)$, נקבל:

$$\geq \sum_a \frac{\epsilon}{|A|} Q^{\pi_i}(s, a) + Q^{\pi_i}(s, a) \sum_a \left(\pi_i(a|s) - \frac{\epsilon}{|A|} \right)$$

כעת יש שני איברים זהים שמצטמצמים, ונשאר עם:

$$= \sum_a \pi_i(a|s) Q^{\pi_i}(s, a)$$

$$\sum_a \pi_{i+1}(a|s) Q^{\pi_i}(s, a) \geq \sum_a \pi_i(a|s) Q^{\pi_i}(s, a)$$

הביטוי שהתקבל מסמל את ה-value function כאשר עוקבים אחר האסטרטגיה π_i . יוצא מכך שאם עושים צעד אחד לפי האסטרטגיה π_{i+1} ואז ממשיכים לפי π_i , זה בהכרח יותר טוב מאשר גם את הצעד הראשון לעשות לפי π_i . כעת בדומה להוכחה שהראינו לעיל, ניתן להוכיח שמתקיים $V^{\pi_i}(s) \leq V^{\pi_{i+1}}(s)$ וביצוע השיפור שוב ושוב יביא את האסטרטגיה להתכנס לזו האופטימלית. הבעיה בשיטה זו היא חוסר היעילות שבה, כיוון שהיא דורשת המון דגימות והמון איטרציות. עקב כך שיטה זו לא פרקטית, ובמקומה נציג כעת שיטות אחרות המאפשרות לשפר את האסטרטגיה עבור מודל משוער. פורמלית, שיטות אלה נכנסות תחת קטגוריה הנקראת Model Free Control – שליטה (ושיפור) אסטרטגיה שאינה מתבססת על מודל ידוע מראש.

11.3.1 SARSA – On-Policy TD control

בפרק הקודם ראינו כיצד ניתן באמצעות שיעור TD לשפר את ה-Value function, כאשר העדכון בכל צעד מתקיים באופן הבא:

$$V(S_t) = V(S_t) + \alpha \cdot [R_{t+1} + \gamma V(S_{t+1}) - V(S_t)]$$

כעת אנחנו מתעניינים באסטרטגיה ולא רק ב-Value function, לכן במקום לעדכן את $V(S_t)$ נעדכן את פונקציית ה- $Q(S_t, A_t)$ state-action. באופן הזה נקבל טבלה בגודל $|S| \times |A|$: המכילה מידע על כל הזוגות האפשריים של (S, A) , ועבור כל זוג יש ערך מסוים (טבלה זו נקראת Q – table). בהתחלה הערכים בטבלה לא יסקפו את הערכים האמיתיים, אך עם התקדמות הלמידה הטבלה תשתפר ואולי אף תתכנס לאופטימליות. העדכון מתבצע באופן הבא:

$$Q(S_t, A_t) = Q(S_t, A_t) + \alpha \cdot [R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)]$$

העדכון בכל צעד מתבצע על סמך המצבים והפעולות של שתי יחידות זמן: $\{S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1}\}$ ולכן האלגוריתם נקרא SARSA. אלגוריתם זה הינו **On-Policy Learning**, כלומר, העדכון בכל צעד נעשה על סמך מידע המגיע מהאסטרטגיה הידועה באותו זמן: בוחרים לבצע פעולה A_t במצב S_t ($Q(S_t, A_t)$) בהתאם לאסטרטגיה, ואז מעדכנים אותה על סמך התגמול R_{t+1} שהתקבל בעקבות הפעולה A_t . באופן סכמתי איטרציה אחת של האלגוריתם מתוארת באופן הבא:

```

Initialize  $Q(s, a), \forall s \in S, a \in \mathcal{A}(s)$ , arbitrarily, and  $Q(\text{terminal-state}, \cdot) = 0$ 
Repeat (for each episode):
  Initialize  $S$ 
  Choose  $A$  from  $S$  using policy derived from  $Q$  (e.g.,  $\epsilon$ -greedy)
  Repeat (for each step of episode):
    Take action  $A$ , observe  $R, S'$ 
    Choose  $A'$  from  $S'$  using policy derived from  $Q$  (e.g.,  $\epsilon$ -greedy)
     $Q(S, A) \leftarrow Q(S, A) + \alpha [R + \gamma Q(S', A') - Q(S, A)]$ 
     $S \leftarrow S'; A \leftarrow A'$ 
  until  $S$  is terminal
    
```

איור 11.7 אלגוריתם SARSA. שיעור on-policy של $Q(s, a)$ בעזרת דגימות מאסטרטגיה ϵ -גרידית ביחס ל- Q הנוכחי.

בכל מצב הפעולה שנבחרת הינה ϵ -גרידית ביחס ל- $Q(s, a)$ הנוכחי. כלומר: אם במצב S_t יש לנקוט לפי האסטרטגיה את המצב $A_t = \bar{A}$, אזי האסטרטגיה ה- ϵ -גרידית ביחס לכך הינה:

$$A_t = \begin{cases} \bar{A} & w.p \ 1 - \epsilon \\ A \neq \bar{A} & w.p \ \epsilon \end{cases}$$

ניתן להוכיח שאלגוריתם SARSA מביא את האסטרטגיה לאופטימליות תחת שני תנאים:

א. שהאסטרטגיה תהיה GLIE.

ב. שיתקיים תנאי Robbins-Monroe עבור α (תנאי זה דואג לכך שנגיע בהכרח לכל המצבים ומצד שני $\alpha_i \rightarrow 0$):

$$\sum_{i=0}^{\infty} \alpha_i = \infty, \sum_{i=0}^{\infty} \alpha_i^2 < \infty$$

לגישה זו, הפועלת בגישת on-policy learning, יש מספר חסרונות:

1. המטרה היא ללמוד את האסטרטגיה האופטימלית אבל בפועל ה-exploration הוא ביחס לאסטרטגיה הנתונה בכל מצב.
2. לא ניתן להשתמש בצעדים ישנים, כיוון שהם מתייחסים לאסטרטגיה שכבר לא רלוונטית.
3. לא ניתן להשתמש במידע שמגיע מבחוץ.

11.3.2 Q-Learning

ניתן להפוך את אלגוריתם SARSA להיות off-policy, והאלגוריתם המתקבל, שנקרא Q-Learning, הוא אחד האלגוריתמים השימושיים בתחום של RL. נתבונן בפונקציית העדכון של SARSA:

$$Q(S_t, A_t) = Q(S_t, A_t) + \alpha \cdot [R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)]$$

האלמנט שתלוי באסטרטגיה הינו $Q(S_{t+1}, A_{t+1})$, כיוון שבו אנחנו נוקטים בפעולה A_{t+1} בהתאם לאסטרטגיה. במקום לבחור בפעולה זו, ניתן להיות גרידי ולקחת את הפעולה בעלת הערך הכי גדול בצעד הקרוב, ועל פיה לעדכן את ה- Q - value:

$$Q(S_t, A_t) = Q(S_t, A_t) + \alpha \cdot [R_{t+1} + \gamma \max_A Q(S_{t+1}, A) - Q(S_t, A_t)]$$

קעת האסטרטגיה אינה משפיעה על פונקציית העדכון, וממילא היא off-policy.

```

Initialize  $Q(s, a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$ , arbitrarily, and  $Q(\text{terminal-state}, \cdot) = 0$ 
Repeat (for each episode):
  Initialize  $S$ 
  Repeat (for each step of episode):
    Choose  $A$  from  $S$  using policy derived from  $Q$  (e.g.,  $\epsilon$ -greedy)
    Take action  $A$ , observe  $R, S'$ 
     $Q(S, A) \leftarrow Q(S, A) + \alpha [R + \gamma \max_a Q(S', a) - Q(S, A)]$ 
     $S \leftarrow S'$ ;
  until  $S$  is terminal
  
```

איור 11.8 אלגוריתם Q-Learning. שערך off-policy של $Q(s, a)$ בעזרת דגימות מאסטרטגיה ϵ -גרידית ועדכון ה- Q - value בהתאם לפעולה בעלת הערך הכי גדול בצעד הקרוב.

האסטרטגיה לא משפיעה על עדכון ה- Q - value, אך כן יש לה השפעה על כמות הפעמים שנבקר בכל מצב. בניגוד לאלגוריתם SARSA בו דרשנו שהאסטרטגיה תהיה GLIE, באלגוריתם Q-Learning הדרשה הינה רק שנבקר אינסוף פעמים בכל מצב. הבדל זה הוא משמעותי, כיוון ש-GLIE דורש שהאסטרטגיה תתכנס לאסטרטגיה הגרידית (כלומר, הדרשה היא ϵ -ילך ל-0). דרישה זו מקשה על הלמידה כיוון שצעדים גרידים מביאים מעט מאוד מידע חדש. הסרת הדרשה על ה- ϵ מאפשרת exploration בצורה יותר חופשית, והלמידה נעשית בצורה הרבה יותר מהירה. עם זאת, יותר מדי exploration זה גם לא טוב, כיוון שמבקרים בהרבה מצבים לא רלוונטיים מספר רב של פעמים.

נתבונן על האלגוריתמים שראינו עד כה ונשווה בין הפתרונות שהיו מבוססים על ידיעת המודל לבין פתרונות משוערכים:

Full Backup (Dynamic Programming)	Sample Backup (TD)
Iterative Policy Evaluation: $V(S) = \mathbb{E}[R + \gamma \cdot V(S') S]$	TD Learning: $V(S) = V(S) + \alpha \cdot [R_{t+1} + \gamma V(S') - V(S)]$
Q-Policy Iteration: $Q(S, A) = \mathbb{E}[R + \gamma \cdot Q(S', A') S, A]$	SARSA: $Q(S, A) = Q(S, A) + \alpha \cdot [R + \gamma Q(S', A') - Q(S, A)]$
Q-Value Iteration:	Q-Learning:

$Q(S, A) = \mathbb{E} \left[\mathcal{R} + \gamma \cdot \max_A Q(S, A) S, A \right]$	$Q(S, A) = Q(S, A) + \alpha \cdot \left[\mathcal{R} + \gamma \max_A Q(S', A) - Q(S, A) \right]$
--	--

הפתרונות המשוערכים מצליחים להתמודד עם בעיות בינוניות, אך הצורך בדגימות יכול להיות בעייתי משתי סיבות:
 א. האלגוריתמים נדרשים להמון דגימות בשביל להצליח. ב. חסרון נוסף בגישות שאינן model-based קשור ל-
 exploration עבור בעיות של העולם האמיתי, לא תמיד אפשרי לראות דוגמאות שליליות כדי ללמוד שהן לא טובות.
 רכב אוטונומי למשל, אם רוצים שהוא ילמד לא לנסוע ברמזור אדום, אי אפשר לאמן אותו על ידי זה שניתן לו יד
 חופשית לבצע exploration וכך הוא יגיע למצבים בהם הוא ייסע באור אדום ויקבל על כך תגמול שלילי. לעומת זאת,
 בעיות הכרוכות בסימולציה הן יותר מתאימות לאלגוריתמים שהצגנו המבוססים על דגימות ושערוך, כיוון שניתן בצורה
 יחסית זולה לבצע המון דגימות, ובנוסף אין בהן בעיות בטיחות בביצוע ה-exploration.

מלבד בעיית הדגימות והשערוך, האתגר העיקרי של גישות אלה נעוץ ביכולת ההכללה של המודלים הנלמדים.
 התוצאה של SARSA ו-Learning-Q הינה כאמור טבלה של ה-value-Q, ובטבלה זו אין שום קשר בין הערכים
 השונים בטבלה. כל איבר בטבלה עומד בפני עצמו, ואי אפשר ללמוד ממנו על שאר האיברים. אם למשל הגענו למצב
 חדש שעוד לא ראינו אך ראינו הרבה מצבים דומים לו, לא נוכל ללמוד מהם שום דבר לגבי המצב החדש. אתגר זה
 משליך גם על גודל הבעיות אותן ניתן לפתור – אם אי אפשר להכליל ממצב אחד למצב אחר, ממילא זה מגביל מאוד
 את גודל הבעיה איתה ניתן להתמודד בעזרת אלגוריתמים אלו. במקרים בהם מרחב המצבים הוא רציף, אז מרחב
 המצבים הוא אינסופי ואז בכלל לא ניתן להשתמש בגישות אלו.

11.3.3 Function Approximation

כאמור, אלגוריתמים שמנסים לשערך את ה-Q-table אינם ישימים בבעיות גדולות בעיקר בגלל חוסר היכולת שלהם
 להכליל ממצב אחד למצב אחר. גם אם יש בידינו אפשרות לשמור טבלאות ענקיות של value-Q, לא נוכל לשפר
 את הטבלה ולעדכן בה ערכים אופטימליים, כיוון שלא ניתן להשליך ממצבים בהם ביקרנו על מצבים חדשים. בכדי
 להתמודד עם בעיה זו, נרצה להחליף את ה-Q-table בפונקציה שמחזירה ערך עבור כל זוג של state-action.
 במקום לחשב טבלה ענקית $Q: |S| \times |A|$, נרצה למצוא פונקציה עם סט פרמטרים θ כך שיתקיים:

$$Q(S, A) \approx Q_\theta(S, A), V(S) \approx V_\theta(S)$$

היתרון של שימוש בפונקציה שמנסה לשערך את הערכים הינו כפול: א. לא צריך לשמור טבלה בגודל של מרחב
 המצבים. ב. כן ניתן ללמוד ממצבים שיש לנו עליהם ידע על מצבים חדשים. בכדי למצוא פרמטרים שישערכו את
 המודל בצורה איכותית יש לפתור בעיית אופטימיזציה, כפי שנגדיר בהמשך, אך הלמידה יכולה להיות מאוד לא
 יציבה. ראשית, כפי שראינו ב-TD, המטרה אותה רוצים לשערך ($V(S_t)$ או $Q(S_t, A_t)$) משתנה בכל צעד. בנוסף,
 בניגוד לאלגוריתמים הקודמים, כעת יש קשר בין המצבים, ולכן אם אנחנו משנים ערך מסוים, זה גם ישפיע על
 ערכים אחרים, מה שמקשה מאוד על יציבות הלמידה.

הדוגמה הפשוטה ביותר לפונקציה כזו הינה מודל לינארי מהצורה:

$$Q(S, A) = w^T \phi(S, A)$$

מודל זה מניח שיש סט של פרמטרים המקיים קשר לינארי בין ה-state-action ומפת פיצ'רים כלשהיא $\phi(S, A)$
 לבין הערך שלהם $Q(S, A)$. באופן הפשטני ביותר, בשביל למצוא את הפונקציה הזו, נרצה למזער כמה שיותר
 את המרחק שבין $V^\pi(S)$ לבין $V_\theta(S)$, ולשם כך נבנה את פונקציית המטרה הבאה:

$$L(\theta) = \frac{1}{2} \mathbb{E}_\theta \left[(V^\pi(S) - V_\theta(S))^2 \right]$$

כמובן שחישוב זה אינו אפשרי משום שאנחנו לא יודעים מה הוא $V^\pi(S)$ – זה בדיוק הביטוי אותו אנו רוצים
 לשערך! בנוסף, חישוב התוחלת יכול להיות מאוד מסובך, בטח במקרה בו אנו לא יודעים את כל הערכים
 האמיתיים של $V^\pi(S)$. בשביל להתגבר על בעיות אלו נשים לב כיצד כן ניתן לשפר את $V_\theta(S)$ – אם נגזור את
 פונקציית המטרה ונבצע צעד בכיוון הגרדיאנט, נקטין את ערכה, ולכן נרצה להיות מסוגלים לחשב את $\Delta\theta$. נבצע
 זאת בדומה בעזרת דגימות סטוכסטיות, כפי שכבר ראינו בפרקים קודמים, כלומר נרצה לדגום מצבים
 מהאסטרטגיה בשביל לחשב את הביטוי הבא:

$$\Delta\theta = (V^\pi(S_t) - V_\theta(S_t)) \nabla_\theta V_\theta(S_t)$$

עם ביטוי זה עדיין לא ניתן להתקדם כיוון ש- $\mathcal{V}^\pi(S)$ לא ידוע, אך גם אותו ניתן לשערך, למשל בעזרת MC או TD. בעזרת שיטת MC ניתן להריץ episode שלם, לחשב את התגמול המצטבר ולשים אותו במקום $\mathcal{V}^\pi(S)$, ובעזרת TD ניתן להריץ צעד אחד ולהחליף את $\mathcal{V}^\pi(S)$ בתגמול המיידי והשערך של המצב הבא:

$$\text{MC: } \Delta\theta = (G_t - \mathcal{V}_\theta(S_t)) \nabla_\theta \mathcal{V}_\theta(S_t)$$

$$\text{TD: } (R_{t+1} + \gamma \mathcal{V}_\theta(S_{t+1}) - \mathcal{V}_\theta(S_t)) \nabla_\theta \mathcal{V}_\theta(S_t)$$

יש לשים לב שבביטוי האחרון יש שני איברים שתלויים ב- θ , $\mathcal{V}_\theta(S_t)$ ו- $\gamma \mathcal{V}_\theta(S_{t+1})$, אך נגזור רק את הביטוי הראשון כיוון שנרצה לשנות אותו כך שיתקרב לביטוי השני ולא להיפרך. עם זאת, בניגוד למה שראינו לעיל בנוגע ל-TD, שם שינוי של $\mathcal{V}(S_t)$ לא השפיע על $\mathcal{V}(S_{t+1})$ כיוון שהערכים בטבלה היו בלתי תלויים אחד בשני, כאן כן יש השפעה בין שני ערכים אלו, מה שמקשה על היציבות של הלמידה.

בסופו של דבר, אנו יכולים לחשב את הביטוי $\Delta\theta$ ובעזרתו לנסות לאפסם את הפרמטרים באיזה שיטות אופטימיזציה סטנדרטיות (למשל – stochastic gradient descent).

בדומה לשערך שהצענו עבור MC ו-TD, ניתן גם לבצע control ולשערך את ה-Q-table:

$$\text{SARSA: } \Delta\theta = (R_{t+1} + \gamma Q_\theta(S_{t+1}, A_{t+1}) - Q_\theta(S_t, A_t)) \nabla_\theta Q_\theta(S_t, A_t)$$

$$Q\text{-Learning: } \Delta\theta = \left(R_{t+1} + \gamma \max_A Q_\theta(S_{t+1}, A) - Q_\theta(S_t, A_t) \right) \nabla_\theta Q_\theta(S_t, A_t)$$

וגם כאן, בגזירה נרצה להתייחס ל- $Q_\theta(S_{t+1}, A_t)$ ו- $Q_\theta(S_{t+1}, A_{t+1})$ כקבועים ולגזור רק את $Q_\theta(S_t, A_t)$ כיוון שנרצה לשנות את $Q_\theta(S_t, A_t)$ כך שיתקרב בערכו לשאר הביטויים המבוססים על מידע נוסף.

יש כמה אתגרים שיכולים להקשות מאוד על שיטות אלה להביא את המודל להתכנסות:

א. ראשית, הדרישה שאומרת שאנו רוצים לשנות את $Q_\theta(S_t, A_t)$ ביחס למידע העתידי הינה בעייתית, כיוון שכעת כל הביטויים תלויים באותו פרמטר θ .

ב. באלגוריתם שהוא off-policy אנו דוגמים מאסטרטגיה ואף פועלים לפי הדגימות האלו, אך אנו לא מנסים להביא לאופטימום דווקא את האסטרטגיה הזו (וממילא היא לא בהכרח האופטימלית עבורנו). כאשר ניסינו ללמוד tabular Q-table, זה היה יכול לגרום לכך ששכיחות המצבים שנדגמו אינה תואמת את השכיחות שלהם לפי האסטרטגיה האמיתית, מה שיכול להשפיע על קצב הלמידה, אך זה לא יגרום לשגיאות באסטרטגיה הנלמדת. כעת כאשר יש קשר בין המצבים על ידי הפרמטר θ , אז דגימה לא לפי השכיחות האמיתית יכולה להטות את הלמידה לכיוונים שגויים.

אתגרים אלו משפיעים על יכולת המודל להתכנס לאסטרטגיה האופטימלית. בעוד שעבור יכולנו להוכיח התכנסות, עבור functional-approximation זה כלל לא מובטח, כפי שמוצג בטבלה הבאה:

Algorithm	Tabular	Linear	Non-Linear
MC	Y	Y	Y
SARSA	Y	Y	N
Q-Learning	Y	N	N

אחת השיטות פורצות הדרך התחום הייתה שימוש ברשתות נוירונים בתור ה-functional-approximation, ובשם הנפוץ שלה – Deep Q – Learning. השימוש ברשת נוירונים בפני עצמה לא הייתה מספיקה כיוון שכאמור הלמידה היא מאוד לא יציבה ויש להכניס מנגנונים נוספים שיעזרו למודל להשתפר. נציין שתי טכניקות שהוטמעו בתהליך הלמידה:

1. כאשר דוגמים כל מיני מצבים, יש ביניהם קורלציה יחסית גבוהה, מה שמונע מהשונות של הגרדיאנט לקטון. כדי להתגבר על כך, במקום לעדכן את Q בכל שלב, שומרים את הרביעיות $(S_t, A_t, R_{t+1}, S_{t+1})$ של N המצבים האחרונים שראינו (למשל: $N = 1,000$), ואז מעדכנים לפי k מצבים (למשל: $k = 10$) אקראיים מתוך N המצבים הקודמים שיש לנו בזיכרון (לאחר שהזיכרון מתמלא, אז כל מצב חדש שמגיע דורס את המצב הכי ישן בזיכרון). בצורה הזו

השונות תקטן כי הדוגמאות שנעדכן לפיהן יהיו בעלות קורלציה נמוכה הרבה יותר. המחיר של טכניקה זו הוא השימוש בהרבה זיכרון, אבל יש לה השפעה משמעותית על הביצועים.

2. הזכרנו לעיל שאנו רוצים לשנות את $Q_\theta(S_t, A_t)$ ביחס למידע העתידי, אך זה יכול להיות בעייתי, כיוון שכעת כל הביטויים תלויים באותו פרמטר θ . במילים אחרות – אנו רוצים לקרב את האסטרטגיה למטרה מסוימת, אך שתיהן תלויות באותו פרמטר. בכדי לייצב את הרגרסיה, ניתן לשנות את המטרה באופן הרבה פחות תדיר. באופן פורמלי, נשמור סט פרמטרים $\hat{\theta}$ ונשנה אותו כל מספר קבוע של צעדים לסט הפרמטרים החדש, כלומר – כל k צעדים נעדכן את $\hat{\theta}$ להיות שווה ל- θ_t . יחד עם הפרמטרים החדשים, נשנה מעט את הביטוי אותו נרצה להביא לאופטימום:

$$\mathbb{E}_{S_t, R_{t+1}, A_t, S_{t+1}} \left[R_{t+1} + \gamma \max_A Q_{\hat{\theta}}(S_{t+1}, A_t) - Q_\theta(S_t, A_t) \right]$$

11.3.4 Policy-Based RL

יש כמה מגבלות וחסרונות בשיטות שראינו עד כה. שיטות אלו מתמקדות בלמידת ה- Q table, בין אם על ידי עדכון ישיר של ערכי הטבלה ובין אם על ידי למידת הפונקציה $Q_\theta(S, A)$ התלויה בפרמטר θ . ראינו שבבעיות גדולות הייצוג והלמידה יכולים להיות קשים. אתגר זה מתעצם ואף נהיה בלתי אפשרי עבור בעיות רציפות, בהן מרחב המצבים הוא כביכול אינסופי (למשל – תנועה של רובוט). חסרון נוסף שיש בגישות שראינו נעוץ במטרה של הלמידה – אנו מנסים ללמוד את $Q_\theta(S, A)$ אך בפועל מה שמעניין אותנו זה $\max_A Q_\theta(S, A)$, מה שיכול להביא לבזבז משאבים עבור למידה של דברים לא הכרחיים. למשל – אם אנחנו צריכים להחליט בין להשקיע כסף בשוק ההון לבין השקעה בתרמית פירמידה, אז אנחנו רק צריכים להחליט איזו אופציה עדיפה לנו, אך אנו לא נדרשים לדעת מה בדיוק כוללת כל אופציה. ההחלטה בין האופציות היא פשוטה, ואילו ללמוד מה ואיך להשקיע בשוק ההון זה דבר מאוד מורכב. לכן במקום ללמוד את כל המערכת, נוכל פשוט ללמוד מה הפעולות הנדרשות עבורנו, וגישה ישירה זו יכולה לחסוך במשאבים.

כאמור, במקום ללמוד את Q , ננסה ללמוד באופן ישיר את האסטרטגיה האופטימלית – $\pi_\theta(a|s)$. באופן מעט יותר פורמלי נגדיר את פונקציית המטרה:

$$J(\theta) = \mathbb{E}_{\pi_\theta} \left[\sum_{t=0}^{\infty} \gamma^t \mathcal{R}_{t+1} \right]$$

מציאת האסטרטגיה האופטימלית שקולה למציאת המקסימום של פונקציית המטרה $J(\theta)$.

נרצה להשתמש באסטרטגיה סטוכסטית מכמה סיבות. ראשית, הרבה פעמים יותר קל למצוא אופטימום עבור בעיה רציפה מאשר בעיה דיסקרטית, ואסטרטגיה סטוכסטית יכולה להפוך בעיה דיסקרטית לרציפה. שנית, יהיו מקרים בהם נתעניין דווקא באסטרטגיה הסטוכסטית ולא בזו הדטרמיניסטית. בנוסף, הסטוכסטיות מאפשרת exploration, וכפי שראינו זה מאפיין הכרחי בשביל להגיע לכל המצבים. במבט יותר רחב, אסטרטגיה סטוכסטית היא הכללה של אסטרטגיה דטרמיניסטית – אם כל ההסתברויות של המודל הינן 0 או 1. במובן הזה, השימוש באסטרטגיה סטוכסטית לא ממעיט מהיכולת של אסטרטגיה דטרמיניסטית אלא הוא רק מכליל אותה.

באופן הכי פשוט, עבור בעיה בה מרחב הפעולות הינו דיסקרטי, ניתן להפוך אסטרטגיה שמספקת ערך לכל פעולה $\phi(a; x, \theta)$ להתפלגות על ידי SoftMax:

$$\pi_\theta(a|s) = \frac{\exp(\phi(a; x, \theta))}{\sum_{a'} \exp(\phi(a'; x, \theta))}$$

אם מרחב הפעולות הוא רציף, ניתן להשתמש בגאוסיאן:

$$\pi_\theta(a|s) \sim \mathcal{N}(\mu(x; \theta), \Sigma(x; \theta))$$

ואם רוצים לאפשר יותר חופש פעולה, ניתן להשתמש ב-Gaussian mixture model. הפונקציה שבאמצעותה רוצים לייצג את האסטרטגיה יכולה להיות כרצוננו – פונקציה לינארית, רשת נוירונים וכו'.

רוב האלגוריתמים שעושים אופטימיזציה ל- $J(\theta)$ מבוססי גרדיאנט, אך ישנם גם אלגוריתמים אחרים, כמו למשל אלגוריתמים גנטיים, אך עליהם לא נדבר. הדרך הסטנדרטית לבצע אופטימיזציה היא להשתמש ב-gradient descent/ascent ובפיתוחים שלו (כמו למשל stochastic gradient descent/ascent), והמוקד שלנו יהיה בשאלה

כיצד לשערך את הגרדיאנט ופחות נתעסק בשיטות האופטימיזציה השונות. האלגוריתם הבסיסי מבוסס על עדכון מהצורה הבאה:

$$\theta_{t+1} = \theta_t + \alpha \nabla f(\theta)$$

נניח ויש לנו episodic MDP, ומהאסטרטגיה הנתונה נוכל לדגום N מסלולים $\tau_1, \dots, \tau_N \sim \pi_\theta$, כאשר כל מסלול הינו $G_i = \sum_{t=0}^{T(i)} \gamma^t R_{t+1}^{(i)}$. שערך מונטה-קרלו הרגיל ישערך את המטרה באופן הבא:

$$\mathcal{J}(\theta) = \mathbb{E}_{\pi_\theta} \left[\sum_{t=0}^{\infty} \gamma^t \mathcal{R}_{t+1} \right] \approx \frac{1}{N} \sum_i G_i$$

הבעיה היא שהאפקט של הפרמטר θ על המסלול G_i היא לא ישירה – אנחנו דוגמים מסלול מאסטרטגיה התלויה בפרמטר θ , ואז מחשבים את התגמול של המסלול וממנו מרכיבים את G . לכן אי אפשר פשוט לגזור את G_i לפי θ , שהרי הקשר ביניהם הוא עקיף ותלוי ב- $f(\theta)$. בכדי להתמודד עם בעיה זו נשתמש ב-log-derivative trick (לעיתים התהליך הזה נקרא Reinforce). נגדיר:

$$g(\theta) = \mathbb{E}_{\pi_\theta} [f(x)]$$

ואז הנגזרת הינה:

$$\nabla_\theta g = \mathbb{E}_{\pi_\theta} [f(x) \nabla \log(\pi_\theta(x))]$$

הוכחה:

$$\nabla_\theta g = \nabla_\theta \mathbb{E}_{\pi_\theta} [f(x)] = \nabla_\theta \int \pi_\theta(x) f(x) dx$$

נחליף את הסדר של הגרדיאנט והאינטגרל:

$$= \int \nabla_\theta \pi_\theta(x) f(x) dx$$

כעת נשתמש בקשר הבא (המתקיים עבור כל פונקציה): $\nabla_\theta \log(h(\theta)) = \frac{\nabla_\theta h(\theta)}{h(\theta)}$, ונקבל:

$$= \int \nabla_\theta \log(\pi_\theta(x)) \pi_\theta(x) f(x) dx$$

וזוהו בדיוק שווה לתוחלת הבאה:

$$= \mathbb{E}_{\pi_\theta} [f(x) \nabla \log(\pi_\theta(x))]$$

כעת, במקום לחשב את הגרדיאנט של G_i , נוכל לחשב את התוחלת $\mathbb{E}_{\pi_\theta} [f(x) \nabla \log(\pi_\theta(x))]$ על ידי דגימות ושערך מונטה קרלו, רק כעת הנגזרת אותה אנו צריכים לחשב אינה תלויה בפרמטר θ ולכן ניתן לחשב אותה בצורה ישירה. בצורה מפורשת, פונקציית המטרה שלנו הינה:

$$\mathcal{J}(\theta) = \mathbb{E}_{\tau \sim \pi_\theta} [G(\tau)], G(\tau) = \sum_{t=0}^{T(\tau)} \gamma^t R_{t+1}^{(\tau)}$$

שימוש ב-log-derivative trick יביא אותנו לביטוי:

$$\nabla \mathcal{J}(\theta) = \mathbb{E}_{\tau \sim \pi_\theta} [G(\tau) \nabla \log(\pi_\theta(\tau))]$$

נשים לב שהאסטרטגיה $\pi_\theta(\tau)$ מורכבת משני אלמנטים – הפעולה שאני בוחר לבצע, והמצב אליו אני מגיע יחד עם התגמול המתקבל מצעד זה. החלק הראשון הוא לגמרי בשליטה שלנו ואנו יודעים אותו, אך החלק השני הוא דינמי ותלוי בסטוכסטיות של המערכת, ולכן לכאורה הוא לא ידוע ולא יאפשר לבצע את הגזירה. אך אם נפתח את הביטוי שקיבלנו נראה שכל החלקים התלויים בדינמיות של המערכת אינם תלויים ב- θ ויתאפסו בגזירה. ניתן לרשום את האסטרטגיה כמכפלת ההסתברויות של כל הצעדים באופן הבא:

$$\pi_\theta(\tau_i) = P(S_0^i) \cdot \pi_\theta(A_0^i | S_0^i) \cdot P(S_1^i, R_1^i | S_0^i, A_0^i) \cdots \pi_\theta(A_{T^i-1}^i | S_{T^i-1}^i) \cdot P(S_{T^i}^i, R_{T^i}^i | S_{T^i-1}^i, A_{T^i-1}^i)$$

כעת אם נוציא לוג, המכפלה תהפוך לסכום:

$$\log(\pi_\theta(\tau_i)) = \log(P(S_0^i)) + \sum_{t=0}^{T^i-1} \log(\pi_\theta(A_t^i|S_t^i)) + \sum_{t=0}^{T^i-1} \log(P(R_{t+1}^i, S_{t+1}^i|S_t^i, A_t^i))$$

הביטוי האמצעי מבטא את בחירות הצעדים של המשתמש, והוא היחיד שתלוי בפרמטר θ . שני הביטויים הנוספים מבטאים את הדינמיות של המערכת שאין לנו מידע כלפיהם, והם אינם תלויים ב- θ , לכן בנגזרת לפי θ הם מתאפסים. עובדה זה בעצם מספקת יתרון נוסף לשימוש בטריק של הלוג – מלבד האפשרות לחשב את הנגזרת באופן ישיר, הלוג גם מפריד בין הצעדים של המשתמש לבין הדינמיות הלא ידועה של המערכת. לכן בסך הכול נקבל:

$$\nabla_\theta \log(\pi_\theta(\tau_i)) = \sum_{t=0}^{T^i-1} \nabla_\theta \log(\pi_\theta(A_t^i|S_t^i))$$

כעת נציב את זה בגרדיאנט של פונקציית המטרה ונשתמש בדיגימות מונטה קרלו:

$$\begin{aligned} \nabla J(\theta) &= \mathbb{E}_{\tau \sim \pi_\theta} [G(\tau) \nabla \log(\pi_\theta(\tau))] \approx \frac{1}{N} \sum_i G_i \nabla_\theta \log(\pi_\theta(\tau_i)) \\ &= \frac{1}{N} \sum_i \left(\sum_{t=0}^{T^i-1} \gamma^t \mathcal{R}_{t+1}^i \right) \left(\sum_{t=0}^{T^i-1} \nabla_\theta \log(\pi_\theta(A_t^i|S_t^i)) \right) \end{aligned}$$

חישוב הגרדיאנט בצורה זו נקרא בשם הפופולרי **Policy Gradient**. כעת משחיבנו את הגרדיאנט, נוכל להשתמש בשיטות אופטימיזציה סטנדרטיות על מנת למצוא את θ האופטימלי. בקצרה נוכל לסכם את כל התהליך ה-**Reinforce** בשלושה שלבים:

1. Run the policy and sample $\{\tau^i\}$ from $\pi_\theta(A_t|S_t)$.
2. Estimate gradients: $\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_i \left(\sum_{t=0}^{T^i-1} \gamma^t \mathcal{R}_{t+1}^i \right) \left(\sum_{t=0}^{T^i-1} \nabla_\theta \log(\pi_\theta(A_t^i|S_t^i)) \right)$.
3. Improve the policy using gradient descent/ascent: $\theta \leftarrow \theta_\alpha \nabla_\theta J(\theta)$.

נתבונן על הביטוי של הנגזרת שקיבלנו וננסה לנתח אותו. נניח ויש מומחה שיודע לייצר trajectories טובים (-imitation learning) – במקרה כזה נרצה לקחת את מה שהוא מייצר ולגרום לכך שהאסטרטגיה שלנו תייצר בהסתברות יותר גבוהה דגימות כמו שקיבלנו מומחה. במילים אחרות, כאשר נתונים לנו מסלולים מוצלחים, נרצה לגרום לאסטרטגיה שלנו "לחקות" את אותם מסלולים. הדרך הסטנדרטית לבצע כזו משימה היא בעזרת maximum likelihood – לקחת את המסלולים האלה ולסכום אותם, ואז לחפש את המקסימום באמצעות גזירה של הביטוי המתקבל. באופן שקול ניתן גם לחפש מקסימום עבור הסכום של **לוג** המסלולים, ובסך הכל נרצה לחשב את הביטוי:

$$\sum_{t=0}^{T^i-1} \nabla_\theta \log(\pi_\theta(\tau_i))$$

וזה בדיוק הביטוי השני בגרדיאנט שראינו קודם. בנוסף אליו יש גם את הגורם הראשון שממשקל כל מסלול בהתאם ל-reward, כך שמסלולים בעלי reward גבוה יקבלו משקל גבוה ואילו מסלולים בעלי reward נמוך יקבלו משקל נמוך. יוצא מכך, שהגרדיאנט מנסה להביא לכך שמסלולים "מוצלחים" יופיעו בהסתברות יותר גבוהה.

השערוך של הגרדיאנט באמצעות הביטוי שראינו הוא אמנם חסר הטיה, אך יש לו שונות גבוהה. בנוסף, אם כל הדגימות הם בעלי reward חיובי (או כולן בעלי reward שלילי), אז יהיה קשה להבחין איזה צעדים יותר טובים ומה לא כדאי לעשות. נראה שתי דרכים לשיפור היציבות של השימוש בגרדיאנט:

בסיבתיות (casualty) – כיוון שהצעד a_t שהתרחש בצעד t לא יכול להשפיע על כל הצעדים שקרו לפני כן $\{a_0, \dots, a_{t-1}\}$, ניתן להחליף את הביטוי $\sum_{t=0}^{T^i-1} \gamma^t \mathcal{R}_{t+1}$ בביטוי שמתחשב רק בצעדים שקרו החל מצעד t , כלומר: $-\sum_{k=t}^{T^i-1} \gamma^k \mathcal{R}_{k+1}$ – ביטוי זה נקרא גם **Reward to go**. החלפה זו מורידה את השונות כיוון שהיא גורמת לכך שכל

צעד יהיה תלוי בפחות איבריים אקראיים. הביטוי שהתעלמנו ממנו הוא $\sum_{k=0}^{t-1} \gamma^k \mathcal{R}_{k+1}^i$, וניתן לראות שאכן אין לו השפעה על העתיד.

Baseline – ניתן להראות שהוספה של קבוע ל-reward משפיעה על התוצאה של הגרדיאנט. קבוע מסוים יכול לגרום לכך שנרצה לשפר צעדים מסוימים ולהימנע מצעדים אחרים, ואילו קבוע אחר יכול לגרום לכך שנרצה לשפר דווקא צעדים אחרים. קבוע זה הוא למעשה פרמטר נוסף שניתן לשלוט בו, ועל ידי בחירה מושכלת שלו ניתן להפחית את השונות. באופן פורמלי, את הביטוי $\sum_{t=0}^{T^i-1} \gamma^t \mathcal{R}_{t+1}^i$ נחליף בביטוי זהה עם תוספת קבוע:

$$\sum_{t=0}^{T^i-1} \gamma^t \mathcal{R}_{t+1}^i - b$$

השאלה היא כיצד לבחור את הקבוע b בצורה טובה? באופן פשוט ניתן לבחור את הקבוע בתור הממוצע של ה-rewards. בחירה כזו תביא לכך שמה שמתחת לממוצע נרצה להוריד את ההסתברות שלו ומה שמעל הממוצע נרצה לחזק אותו. בחירה כזו אכן מורידה את השונות, אך היא אינה אופטימלית. ניתן לחשב את הקבוע האופטימלי בצורה מדויקת. הנגזרת של פונקציית המטרה יחד עם האיבר הנוסף הינה:

$$\nabla J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta} [(G(\tau) - b) \nabla \log(\pi_\theta(\tau))]$$

הממוצע אינו תלוי ב- b , לכן נוכל לרשום את השונות כך:

$$\begin{aligned} \text{Var} &= \mathbb{E}_{\tau \sim \pi_\theta} [(G(\tau) - b)^2 \nabla \log(\pi_\theta(\tau))^2] + C \\ &= \mathbb{E}_{\tau \sim \pi_\theta} [G(\tau)^2 \nabla \log(\pi_\theta(\tau))^2] + \mathbb{E}_{\tau \sim \pi_\theta} [b^2 \nabla \log(\pi_\theta(\tau))^2] - 2 \mathbb{E}_{\tau \sim \pi_\theta} [G(\tau) \cdot b \nabla \log(\pi_\theta(\tau))^2] + C \end{aligned}$$

כעת נגזור את השונות לפי b :

$$\begin{aligned} \frac{d\text{Var}}{db} &= 2 \mathbb{E}_{\tau \sim \pi_\theta} [b \nabla \log(\pi_\theta(\tau))^2] - 2 \mathbb{E}_{\tau \sim \pi_\theta} [G(\tau) \cdot b \nabla \log(\pi_\theta(\tau))^2] = 0 \\ \rightarrow b_{opt} &= \frac{\mathbb{E}_{\tau \sim \pi_\theta} [G(\tau) \nabla \log(\pi_\theta(\tau))^2]}{\mathbb{E}_{\tau \sim \pi_\theta} [b \nabla \log(\pi_\theta(\tau))^2]} \end{aligned}$$

הביטוי המתקבל הוא אכן הממוצע של ה-reward, אך מוכפל במשקל התלוי בנגזרת של לוג האסטרטגיה. השערוך של הביטוי המדויק יכול להיות מסובך ורועש, ולכן לרוב כן משתמשים פשוט ב-reward הממוצע.

אלגוריתם policy gradient הינו on-policy, כלומר האסטרטגיה שבעזרתה אנו דוגמים מסלולים הינה אותה אסטרטגיה שאנו מנסים להביא לאופטימום. כאמור לעיל, באלגוריתמים שהם on-policy, אי אפשר ל"מחזר" דגימות, כלומר בכל נקודת זמן ניתן להשתמש רק בדגימות שנלקחו מהאסטרטגיה הנתונה באותה נקודה. עבור מצבים בהם הדגימות יקרות, נרצה להשתמש באותן דגימות עבור נקודות זמן שונות, כלומר להפוך את האלגוריתם להיות off-policy. בכדי לעשות זאת יש לתקן את השיגאה שנוצרת מכך שבנקודות זמן מסוימות אנו משתמשים בדגימות שנלקחו מאסטרטגיה שכבר השתנתה וכעת ההתפלגות של האסטרטגיה שונה. במילים אחרות – יש לנו דגימות שנלקחו מתוחלת מסוימת, ובעזרתן אנו רוצים לשערך תוחלת אחרת. זו בעיה מתחום הסטטיסטיקה, והפתרון שלה נקרא **Importance sampling**:

נניח ואנו רוצים לשערך את $\mathbb{E}_p[f(x)]$, אך אנו לא יכולים לדגום מהתפלגות p אלא רק מהתפלגות q , כאשר ההנחה היחידה שלנו היא שאם $p > 0$ אז גם $q > 0$. באמצעות מעבר מתמטי די פשוט ניתן לייצג את התוחלת של p באמצעות התוחלת של q :

$$\mathbb{E}_p[f(x)] = \int f(x)p(x)dx = \int f(x) \frac{p(x)}{q(x)} q(x)dx = \mathbb{E}_q \left[f(x) \frac{p(x)}{q(x)} \right]$$

כעת נדגום $x_1, \dots, x_N \sim q$ ונשערך את התוחלת באמצעות מונטה קרלו:

$$\mathbb{E}_p[f(x)] = \mathbb{E}_q \left[f(x) \frac{p(x)}{q(x)} \right] \approx \frac{1}{N} \sum_i f(x_i) \frac{p(x_i)}{q(x_i)}$$

אם ההתפלגויות $p(x), q(x)$ הן יחסית דומות, אז התוצאה המתקבלת משערכת בצורה יחסית טובה את התוחלת הרצויה. אם זה לא המצב, השונות תהיה גדולה והתוצאה תהיה לא יציבה. מכל מקום, נוכל להשתמש ברעיון זה עבור שערך הגרדיאנט:

- For trajectory probability the dynamics does not depend on θ so

$$\frac{\pi_{\theta'}}{\pi_{\theta}} = \frac{\prod \pi_{\theta'}(a_t | s_t)}{\prod \pi_{\theta}(a_t | s_t)}$$

- Can now Compute $\nabla_{\theta'} J(\theta')$ with samples from π_{θ}

$$\bullet \sum_{i=1}^N \left(\sum_{t=0}^{T^{(i)}-1} \gamma^t r_{t+1}^{(i)} \right) \left(\sum_{t=0}^{T^{(i)}-1} \nabla_{\theta} \log \pi_{\theta}(a_t^{(i)} | s_t^{(i)}) \right) \frac{\prod \pi_{\theta'}(a_t | s_t)}{\prod \pi_{\theta}(a_t | s_t)}$$

- Can improve with causality

$$\bullet \sum_{i=1}^N \sum_{t=0}^{T^{(i)}-1} \nabla_{\theta} \log \pi_{\theta}(a_t^{(i)} | s_t^{(i)}) \frac{\prod_{k=0}^t \pi_{\theta'}(a_k | s_k)}{\prod_{k=0}^t \pi_{\theta}(a_k | s_k)} \left(\sum_{k=t}^{T^{(i)}-1} \gamma^k r_{k+1}^{(i)} \right)$$

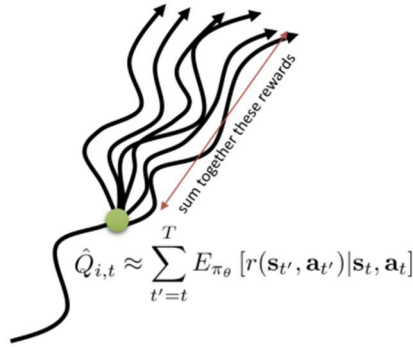
כאמור, שימוש ברעיון זה יכול להיות טוב אם ההתפלגויות θ, θ' יחסית דומות. אם הן רחוקות, התוצאה המתקבלת לא בהכרח מהווה שערך טוב, ולכן הלמידה לא תהיה מספיק טובה. לסיכום – אפשר להפוך את האלגוריתם של policy gradient להיות off-policy באמצעות importance sampling, אך גם שיטה יכולה לסבול משונות גבוהה.

11.3.5 Actor Critic

בפרקים הקודמים דיברנו על שתי גישות בכדי לשערך את האסטרטגיה בלי לדעת המודל. גישה אחת התמקדה בשערך ה-Action-Value function (SARSA/Q-Learning), בעוד הגישה השנייה ניסתה לשערך את האסטרטגיה באופן ישיר (Policy gradient). בפרק זה נציג אלגוריתם המנסה לשלב את שתי הגישות יחד. נתבונן שוב בביטוי של הגרדיאנט, כאשר בהתייחסות ל-rewards נתייחס רק ל-reward to go, כפי שהוסבר לעיל:

$$\frac{1}{N} \sum_i \left(\sum_{k=t}^{T^i-1} \gamma^k \mathcal{R}_{k+1}^i \right) \left(\sum_{t=0}^{T^i-1} \nabla_{\theta} \log (\pi_{\theta}(A_t^i | S_t^i)) \right)$$

ניתן להבחין כי הביטוי $\sum_{k=t}^{T^i-1} \gamma^k \mathcal{R}_{k+1}^i$ הוא למעשה דגימה של $Q^{\pi_0}(s_t, a_t)$, כלומר עבור נקודת זמן, ניתן לדגום מסלול באמצעות ה-Action-Value function ולקבל סכום של rewards. החיסרון בדגימה זו נעוץ בכך שיש לה שונות גבוהה והיא יכולה להיות מאוד רועשת. כפי שניתן לראות באיור, יתכנו הרבה מסלולים שונים היוצאים מאותה נקודה, ולכן הביטוי של ה-reward to go הוא מאוד רועש ביחד ל- $Q^{\pi_0}(s_t, a_t)$:



איור 11.9 מסלולים שונים אפשריים היוצאים מאותה נקודה. המיצוע/התוחלת של הביטוי אמנם מוריד את השונות ל-0, אך כל דגימה בפני עצמה היא בעלת שונות גבוהה.

הבחנה זו של הקשר בין go-to reward לבין $Q^{\pi_0}(s_t, a_t)$ מעלה את השאלה האם אפשר להחליף את בין הביטוי הרועש $\sum_{k=t}^{T-1} \gamma^k \mathcal{R}_{k+1}^i$ לבין הפונקציה עצמה $Q^{\pi_0}(s_t, a_t)$. בשביל לבצע החלפה זו, יש להוכיח שהיא אכן חוקית ושומרת על תוצאה נכונה, וכדי לעשות זאת נפתח את פונקציית המטרה כך שתופיע בצורה יותר נוחה. כזכור, פונקציית המטרה אותה אנו מעוניינים להביא לאופטימום הינה התוחלת של ה-rewards (מוכפלים ב-discount factor):

$$J(\theta) = \mathbb{E}_{\pi_\theta} \left[\sum_{t=0}^{\infty} \gamma^t \mathcal{R}_{t+1} \right]$$

המשמעות של התוחלת היא לבצע אינטגרל על כל צמד של action-state (s, a) , כפול המשקל של אותו צמד. כיוון שצמד יכול לחזור על עצמו בצעדים שונים (כלומר בשתי נקודות זמן t_1, t_2 המערכת יכולה להיות באותו מצב ולבצע את אותו פעולה), אז למעשה באינטגרל ישנם ביטויים שחוזרים על עצמם. בכדי להימנע מכך, ניתן לחשב כל ביטוי באינטגרל פעם אחת, ולהכפיל אותו במשקל המתאים למספר הביקורים באותו צמד action-state. עם זאת, עדיין צריך לזכור שהמופעים של אותו צמד נבדלים זה מזה ב-discount factor, כיוון שהמשקל של reward בזמן t_1 שונה מהמשקל שלו בזמן t_2 , ויש לקחת את ההבדל הזה בחשבון. כעת נתבונן על הביטוי המפורש של האינטגרל המתקבל, כאשר ראשית נחליף את סדר התוחלת והסיכוי:

$$J(\theta) = \mathbb{E}_{\pi_\theta} \left[\sum_{t=0}^{\infty} \gamma^t \mathcal{R}_{t+1} \right] = \sum_{t=0}^{\infty} \gamma^t \mathbb{E}_{\pi_\theta} [\mathcal{R}_{t+1}]$$

התוחלת של כל ה-rewards מורכבת ממיצוע שלהם על פני: (1) ההסתברות להתחיל ממצב s_0 (כפול 2) ההסתברות להגיע למצב s בזמן t בהנתן שהתחלנו מ- s_0 (כפול 3) הסיכוי לבצע פעולה a במצב s שהתקבל בזמן t . באופן פורמלי הביטוי המתקבל הינו

$$\begin{aligned} &= \sum_{t=0}^{\infty} \gamma^t \int_S \int_S \int_{\mathcal{A}} p(s_0) p_t^\pi(s|s_0) \pi(a|s) \mathcal{R}(s, a) ds_0 ds da \\ &= \int_S \left(\sum_{t=0}^{\infty} \gamma^t \int_S p(s_0) p_t^\pi(s|s_0) ds_0 \right) \int_{\mathcal{A}} \pi(a|s) \mathcal{R}(s, a) ds da \end{aligned}$$

נסמן את הביטוי שבסוגריים כ- $\rho^\pi(s)$, והמשמעות שלו היא מה הסיכוי להגיע ב- t צעדים ממצב s_0 למצב s , כאשר עבור כל t יש להכפיל ב-discount factor המתאים. ביטוי זה נקרא Discount state visitation measure, ולמעשה הוא כולל בתוכו את כל המופעים של צמד (s, a) , ובכך במקום לעשות אינטגרל על אותו צמד (s, a) מספר פעמים, עושים זאת פעם אחת עבור כל נקודות הזמן השונות ומכפילים ב-discount factor המתאים. באופן מפורש:

$$= \int_S \rho^\pi(s) \int_{\mathcal{A}} \pi(a|s) \mathcal{R}(s, a) ds da,$$

$$\rho^\pi(s) = \sum_{t=0}^{\infty} \gamma^t \int_s p(s_0) p_t^\pi(s|s_0) ds_0$$

יש לשים לב שהביטוי הכולל הוא לא בצורה הפשוטה של תוחלת – אינטגרל של הסתברות כפול פונקציית צפיפות, אך הוא עדיין משקף תוחלת ולכן ניתן לשערך אותו באמצעות מונטה-קרלו. כעת נתבונן על הגרדיאנט של פונקציית המטרה. אנחנו רוצים לגזור את הביטוי הבא לפי θ :

$$\mathcal{J}(\theta) = \int_S \rho^\pi(s) \int_{\mathcal{A}} \pi(a|s) \mathcal{R}(s, a) ds da$$

כל האיברים תלויים ב- θ אך ניתן להראות (שקופיות 10-11 במצגת 7) שהנגזרת תלויה רק ב- $\pi(a|s)$, כלומר:

$$\nabla_{\theta} \mathcal{J} = \int_S \rho^\pi(s) \int_{\mathcal{A}} \nabla_{\theta} (\pi(a|s)) Q^{\pi_{\theta}}(s, a) ds da$$

כעת ניתן להיעזר ב-log-derivative trick ולקבל:

$$\nabla_{\theta} \mathcal{J} = \int_S \rho^\pi(s) \int_{\mathcal{A}} \pi(a|s) \nabla_{\pi} \log(\pi(a|s)) Q^{\pi_{\theta}}(s, a) ds da$$

ואז לשערך את הביטוי באמצעות דגימות מונטה-קרלו:

$$\nabla_{\theta} \mathcal{J} \approx \frac{1}{N} \sum_{i=1}^N \sum_{t=0}^{T^{(i)}-1} \nabla_{\theta} \log \pi_{\theta}(a_t^{(i)} | s_t^{(i)}) Q^{\pi_{\theta}}(s_t^{(i)}, a_t^{(i)})$$

הביטוי המתקבל טוב מאוד מבחינת זה שהוא מצליח להפחית את השונות, אך הבעיה בו נעוצה בכך שהאיבר $Q^{\pi_{\theta}}(s_t^{(i)}, a_t^{(i)})$ אינו ידוע! אלגוריתם Actor Critic בא להתמודד עם בעיה זו, והרעיון הוא לנסות לשערך את Q^w באמצעות Q^w במקביל לשערך של הגרדיאנט. כלומר, בתהליך הלמידה אנו מנסים לשערך שני דברים במקביל:

π_{θ} (Actor) – הגורם המחליט איזה צעד לנקוט בכל מצב.

Q^w (Critic) – הגורם המבצע אבולוציה של Q על מנת לשפר את הבחירה של ה-Actor.

הערת אגב: לעיל ראינו שהשימוש ב-Learning – Q יכול להיות בעייתי בבעיות רציפות בגלל שמרחב המצבים הוא אינסופי ומאוד קשה לפתור את בעיית האופטימיזציה של בחירת action בעזרת הביטוי $\arg \max_A Q_{\theta}(s, a)$. כאן בעיה זאת איננה, כיוון שאנחנו לא נדרשים לבחור את ה-action באמצעות Q אלא רק לשפר באמצעותו את האסטרטגיה (או במילים אחרות – אנחנו לא צריכים לפתור בעיית אופטימיזציה על Q עבור מרחב מצבים רציף).

האלגוריתם פועל בהתאם לשלבים הבאים. ראשית נאתחל את θ, w . כעת נבצע T איטרציות באופן הבא:

- נדגום מצב התחלתי s_0 מתוך מרחב המצבים \mathcal{S} ופעולה מתוך האסטרטגיה $a_0 | s_0 \sim \pi_{\theta}(\cdot | s_0)$.
- כעת כל עוד לא הגענו למצב הסופי:

- נחשב את המצב החדש s ואת התגמול המתקבל r .
- בנוסף, נדגום פעולה חדשה $a \sim \pi_{\theta}(\cdot | s)$.
- נחשב את ה-TD error:

$$\delta = r + \gamma Q^w(new\ s, new\ a) - Q^w(s, a)$$

- נעדכן את הפרמטרים:

$$\begin{aligned} \theta &\leftarrow \theta + \alpha Q^w(s, a) \nabla_{\theta} \log \pi_{\theta}(a|s) \\ w &\leftarrow w + \beta \delta \nabla_w Q^w(s, a) \end{aligned}$$

- נעדכן את המצב והפעולה:

$$a = new\ a, s = new\ s$$

פסאודו קוד עבור האלגוריתם:

```

Initialize parameters  $s, \theta, w$  and learning rates  $\alpha_\theta, \alpha_w$ ; sample  $a \sim \pi_\theta(a|s)$ .
for  $t = 1 \dots T$ : do
  Sample reward  $r_t \sim R(s, a)$  and next state  $s' \sim P(s'|s, a)$ 
  Then sample the next action  $a' \sim \pi_\theta(a'|s')$ 
  Update the policy parameters:  $\theta \leftarrow \theta + \alpha_\theta Q_w(s, a) \nabla_\theta \log \pi_\theta(a|s)$ ; Compute
  the correction (TD error) for action-value at time  $t$ :
     $\delta_t = r_t + \gamma Q_w(s', a') - Q_w(s, a)$ 
  and use it to update the parameters of Q function:
     $w \leftarrow w + \alpha_w \delta_t \nabla_w Q_w(s, a)$ 
  Move to  $a \leftarrow a'$  and  $s \leftarrow s'$ 
end for

```

איור 11.10 אלגוריתם Actor-Critic. שערך הגרדיאנט באמצעות Q^w במקום הביטוי של ה-Policy gradient-to-reward.

חשוב להדגיש שהעדכון של הגרדיאנט מתבצע בדומה ל-SARSA ולא כמו Q -Learning. כאמור לעיל, ההבדל ביניהם הוא באופי האלגוריתם – SARSA הוא אלגוריתם on-policy, כלומר הוא מנסה לאפסם את האסטרטגיה ממנו הוא דוגם את המסלולים, בעוד ש- Q -Learning הוא אלגוריתם off-policy, כלומר המטרה היא לשפר את אותן הוא רואה. אלגוריתם Actor Critic רוצה לשפר את ה-Policy gradient, כלומר המטרה היא לשפר את האסטרטגיה הנתונה, ולכן יש לבצע למידה שהיא On policy.

בדומה ל-Policy gradient, גם ל-Actor Critic ניתן להוסיף Baseline על מנת לשפר את תהליך הלמידה. בניגוד ל-Policy gradient הוא התוספת היתה של קבוע, כאן נהוג להוסיף את $\mathcal{V}^\pi(s)$, שהוא תלוי ב- s . אם ה-baseline תלוי רק במצב s , אז זה בסדר כפי שנוכח מיד (שקף 16), אבל אם הוא תלוי גם ב- a , אז זה מוסיף bias והלמידה תפגע. הביטוי: $A^\pi(s, a) = Q^\pi(s, a) - \mathcal{V}^\pi(s)$ נקרא ה-advantage function, והסיבה לבחירה שלו כ-baseline היא פשוטה – אם הערך של $A^\pi(s, a)$ חיובי, אז זה סימן לכך שהבחירה בצעד a היא טובה, ולכן נרצה לחזק אותה. אם לעומת זאת הערך שלילי, זה אומר שהפעולה לא טובה וממילא נרצה להימנע ממנה. במילים אחרות: $\mathcal{V}^\pi(s)$ הינו הערך הממוצע של המצב s , והביטוי $Q^\pi(s, a)$ הינו הערך של פעולה ספציפית עבור מצב s , ולכן $A^\pi(s, a)$ אומר לנו עד כמה הפעולה a טובה או גרועה ביחס לממוצע של המצב בו אנו נמצאים, וממילא נוכל לדעת האם נרצה לחזק את הפעולה הזו או לא.

לאחר שראינו שניתן להשתמש ב-Baseline עבור Actor Critic ויש היגיון להשתמש ב- $\mathcal{V}^\pi(s)$ בתור ה-baseline, נראה כיצד ניתן ליישם זאת בפועל. אם מרחב המצבים הוא דיסקרטי, ניתן ללמוד את $Q^w(s)$ בשיטות הסטנדרטיות שראינו, ואז לחשב באופן מדויק את $\mathcal{V}^\pi(s)$ בעזרת הקשר $\mathcal{V}^\pi(s) = \sum_a \pi(a|s) Q^w(s, a)$. במצבים מסוימים ניתן להשתמש באותו רעיון גם עבור מרחב מצבים רציף – נניח ו- $Q^w(s)$ הוא לינארי ביחס למרחב הפעולות, כלומר מתקיים: $Q^w = \phi_w(s)^T \cdot a$, והאסטרטגיה היא גאוסיאנית: $\pi_\theta(a|s) \sim \mathcal{N}(\mu(s), \sigma^2(s))$ אז $\mathcal{V}^\pi(s)$ הוא התוחלת של $Q^w(s)$ על פני כל הפעולות: $\mathcal{V}^\pi(s) = \mathbb{E}_{a \sim \pi_\theta(\cdot|s)}[Q^w(s)]$. בעזרת הנוסחה הסגורה הזו ניתן להשתמש בשערך $Q^w(s)$ ולחשב את $\mathcal{V}^\pi(s)$ ואז להשתמש בו כ-Baseline.

כאשר לא ניתן להשתמש ב- $Q^w(s)$ על מנת לחשב את $\mathcal{V}^\pi(s)$, ניתן לנסות לשערך את שני הביטויים בנפרד, אך זה יכול להיות בזבזני. לחילופין, ניתן לבנות מודל (למשל רשת נוירונים) שתנסה לשערך במקביל את שני הביטויים, כלומר הפלט שלה יהיה גם $Q^w(s)$ וגם $\mathcal{V}^\pi(s)$. גישה אחרת מציעה להשתמש במשוואות בלמן. לפי משוואות אלו מתקיים בקירוב $Q(s_t, a_t) = r_{t+1} + \gamma \mathcal{V}(s_t)$, ולפי זה ה-advantage function יהיה:

$$A(s_t, a_t) = r_{t+1} + \gamma \mathcal{V}(s_{t+1}) - \mathcal{V}(s_t)$$

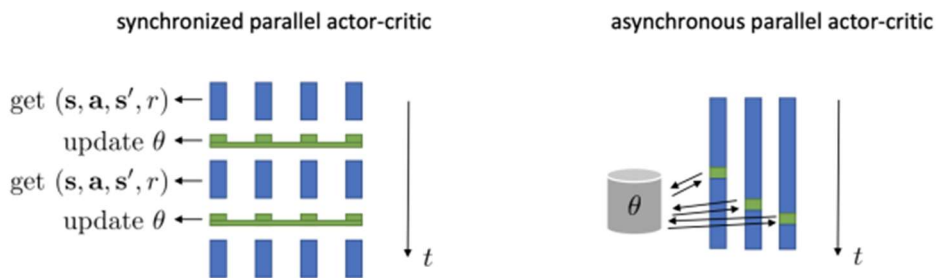
את הביטוי הזה נכניס לגרדיאנט ובכך נשערך את האסטרטגיה. נשים לב שביטוי זה תלוי במצב בלבד ולא בפעולה, מה שמקל על הלמידה, כיוון שיותר קל ללמוד את $\mathcal{V}(s)$ השייך למרחב בגודל \mathcal{S} מאשר את $Q(s, a)$ השייך למרחב בגודל $\mathcal{S} \times \mathcal{A}$. המשמעות הפרקטית של הביטוי היא שמסתכלים על שני מצבים, ובוחנים את הפרש הערכים שלהם ועוד ה-reward. אם הערך חיובי – אז נרצה לחזק את המעבר מהמצב הראשון למצב השני, ואם הערך שלילי, אז נרצה למנוע כזה מעבר.

סר – שקופית 19.

אחת הבעיות שיש גם ב- Q -Learning וגם ב-Actor Critic היא הקורלציה שיש בין הדגימות – עבור מסלול שנדגם, יש קשר בין המצבים השונים, ולכן השונות עדיין גבוהה. לעיל ראינו שבאלגוריתם Q -Learning ניתן להתגבר על

כך בעזרת שמירה של N מצבים ועדכון לפי k מצבים אקראיים מתוך אלה שנשמרו (למשל: $N = 1000, k = 10$). כן לא ניתן להשתמש בפתרון זה, שהרי הלמידה היא on-policy ואי אפשר להשתמש במצבים מדגימות קודמות, כיוון שהן נדגמו מאסטרטגיה שכרגע היא לא רלוונטית. ניתן בכל אופן להשתמש במצבים שנדגמו מה-epochs האחרונים, תחת הנחה שהאסטרטגיה לא השתנה הרבה (ובנוסף ניתן לתקן באמצעות Importance sampling).

פתרון יותר מוצלח הוא לאמן מספר סוכנים במקביל ולהשתמש במידע שמגיע מכולם, כאשר היתרון הגדול של שיטה זו הוא שהדגימות של הסוכנים השונים הן חסרות קורלציה. שיטה זו הציגה תוצאות מעולות ובזמן הרבה יותר מהיר מהשיטות הקודמות שראינו. יש שתי אפשרויות להריץ במקביל – ריצה סינכרונית (הסוכנים רצים במקביל, ולאחר שכולם מסיימים epoch מחשבים את הגרדיאנט ומעדכנים את האסטרטגיה) וריצה אסינכרונית (כל פעם שסוכן מסיים epoch הוא מחשב גרדיאנט ומעדכן האסטרטגיה). הריצה הסינכרונית יכולה להיות קצת יותר איטית כיוון שבכל עדכון יש לחכות שכל הסוכנים יסיימו. מצד שני הריצה האסינכרונית יכולה להיות פחות מדויקת כיוון שיתכנו מצבים שלסוכנים שונים תהיה אסטרטגיה מעט שונה (אם הם עדכנו את האסטרטגיה שלהם בזמנים שונים ובאמצע היה עדכון של הגרדיאנט). עם זאת ניתן להניח שההבדלים יחסית לא גדולים, כיוון שהסוכנים מעדכנים את האסטרטגיה שלהם בערך באותן נקודות זמן.



איור 11.11 אימון מספר סוכנים במקביל בשיטת Actor-Critic. ניתן לבצע את העדכון בצורה סינכרונית – כאשר כל הסוכנים מסיימים epoch מחשבים את הגרדיאנט ומעדכנים את האסטרטגיה, או בצורה אסינכרונית – כל סוכן מסיים epoch מעדכן את האסטרטגיה בהתאם.

לסיכום, שיטת Actor-Critic מנסה לשלב עקרונות מתוך Q -Learning יחד עם Policy gradient. הרעיון הבסיסי הוא להחליף את הביטוי של ה-Policy gradient reward-to-go בשערוך של Q , שהוא הרבה פחות רועש. את Q ניתן לשערך במקביל לגרדיאנט עצמו, אך יש לעשות זאת on-policy. אמנם השונות יורדת אך עדיין יש הטיה, כיוון שיש צורך לשערך גם את Q . בדומה ל-policy gradient, גם כאן ניתן להוסיף Baseline, וראינו שמקובל להוסיף את V .

11.4 Model Based Control

בפרק הקודם דיברנו על אלגוריתמים שלא מתעסקים במודל של הבעיה, אלא מנסים ללמוד ישירות מתוך הניסיון את הערך של כל מצב/מצב-פעולה ($Q(s, a)/V(s)$) ו/או את האסטרטגיה האופטימלית עבור הסוכן. בפרק זה נדבר על אלגוריתמים שמנסים ללמוד את האסטרטגיה מתוך המודל או הדינמיקה של הבעיה (בין אם הם יודעים ובין אם הם נלמדת).

יש מספר יתרונות ללמידה מתוך המודל ולא מתוך ניסיון בלבד. ראשית, הרבה פעמים הלמידה של מודל יכולה להיות משמעותית פשוטה יותר מאשר למידה של $V(s)/Q(s, a)$, מה שהופך את הלמידה להרבה יותר חסכונית. ניקח לדוגמה מבוך פשוט – הבנה של הדינמיקה שלו היא הרבה יותר פשוטה מאשר למידה של הערך של כל משבצת. בנוסף, למידה של המודל מאפשרת Self-exploration – נניח ואנו רוצים לאמן סוכן שמבצע נהיגה אוטונומית, אם הוא לא ידע את הדינמיקה של הסביבה אלא ינסה ללמוד רק על בסיס הניסיון, הוא יהיה חייב לבצע פעולות שליליות (-למשל לדרוס אנשים ולעשות תאונות) על מנת לקבל משוב שלילי וכך ללמוד שיש להימנע מפעולות אלה. אם לעומת זאת אנו יודעים את המודל והסביבה, ניתן לייצר סימולציה ובה לבה לבחון את המצבים השונים וכך לאמן את הסוכן.

עם זאת, ישנם גם חסרונות בלמידה של המודל. ראשית, הרבה פעמים יש פערים בין המודל לבין העולם האמיתי. נניח ואימנו סוכן של רכב אוטונומי בסביבה וירטואלית, המעבר לעולם האמיתי אינו טריוויאלי כי הסימולציה פשוט לא מספיק דומה לעולם האמיתי (למשל – רזולוציית תמונת הקלט של הסוכן בסימולציה הרבה יותר גבוהה מזו של מצלמה אמיתית המותקנת על רכב). בנוסף, למידה של מודל מוסיפה מקור נוסף של שגיאה – נניח ומנסים לשערך מודל ועל בסיס המודל רוצים ללמוד אסטרטגיה, אז יש פה שני שלבים של למידה וכל אחד מהם יכול לגרום לשגיאה.

11.4.1 Known Model – Dyna algorithm

נתחיל מהמקרה היותר פשוט, בו המודל ידוע ואנו מנסים באמצעותו ללמוד אסטרטגיה אופטימלית. בפרק המבוא ראינו כיצד ניתן להשתמש ב-Policy iteration עבור Tabular MDP בעל מרחב מצבים לא גדול על מנת ללמוד את האסטרטגיה האופטימלית, ובפרק זה נרצה להתמקד במקרים בהם פתרון זה לא מספיק טוב.

הרעיון הפשוט ביותר בו ניתן להשתמש הוא לייצר סימולציה, כלומר לקחת את המודל הידוע, ולאמן בעזרתה סוכן כאילו שהוא נמצא בעולם האמיתי. למידה של האסטרטגיה או של $V(s)/Q(s, a)$ באופן הזה לרוב תהיה זולה משמעותית מאשר למידה שלהם ללא עזרת המודל, כיוון שנצטרך הרבה פחות איטרציות והשימוש במודל הידוע מסייע להכווין את הלמידה. עם זאת, הבעיה היא שיש פער בין הסימולציה לבין העולם האמיתי, מה שיוצר אתגר כפול: א. הסימולציה לא מגיעה לרמת דיוק מספיק טובה בתיאור העולם האמיתי ולכן הלמידה לא מספיק איכותית. ב. נניח ויש לנו אפשרות ללמוד על העולם האמיתי, אז גם אם יש שגיאה – היא לרוב תהיה קטנה מאוד. בסימולציה לעומת זאת, שגיאה כזו יכולה להצטבר ולגרום לסוכן לשגיאות גדולות.

בכדי להתגבר על אתגרים אלו ניתן לשלב בין Model free ו-Model Based, וכעת נציג אלגוריתם בשם Dyna שעושה בדיוק את השילוב הזה. אלגוריתם זה משלב ידע משני מקורות – דאטה אמיתי ודאטה המגיע מסימולציה, כאשר הוא פועל באופן איטרטיבי לפי השלבים הבאים:

- ראשית מקבלים דאטה אמיתי, ומשפרים באמצעותו שני דברים – מודל שעליו מושתת הסימולציה וסוכן שמאומן בכלים של model free.
- לאחר מכן מייצרים דאטה מהסימולציה, ומשתמשים בו בשביל לשפר את הסוכן.

מבצעים את השלבים האלה שוב ושוב וככה משפרים במקביל גם את הסימולציה וגם את הסוכן (שלומד על בסיס דאטה מהסימולציה). באופן הזה אנחנו משפרים את הסימולציה כל הזמן, ואז גם אם יש בעיה בסימולציה והיא לא זהה בדיוק למודל של העולם האמיתי, אנחנו לא מתעלמים ממנה אלא מנסים לתקן אותה. בניגוד לסימולציה פשוטה שלא משתמשת בדאטה מהעולם האמיתי אלא רק מתבססת על המודל הידוע, כאן אנו לא זונחים את הדאטה האמיתי אלא עושים לו אוגמנטציות בעזרת הסימולציה (שבעצמה גם הולכת ומשתפרת). נניח מקרה פשוט של מודל דטרמיניסטי ובחירה של אלגוריתם Q-Learning – לאימון הסוכן, האלגוריתם יפעל לפי שלבים הבאים (כמובן שניתן להרחיב בצורה טריוויאלית לכל מודל ולכל אלגוריתם Model-Free):

```
Initialize  $Q(s, a)$  and  $Model(s, a)$  for all  $s \in \mathcal{S}$  and  $a \in \mathcal{A}(s)$ 
Do forever:
  (a)  $S \leftarrow$  current (nonterminal) state
  (b)  $A \leftarrow \epsilon$ -greedy( $S, Q$ )
  (c) Execute action  $A$ ; observe resultant reward,  $R$ , and state,  $S'$ 
  (d)  $Q(S, A) \leftarrow Q(S, A) + \alpha [R + \gamma \max_a Q(S', a) - Q(S, A)]$ 
  (e)  $Model(S, A) \leftarrow R, S'$  (assuming deterministic environment)
  (f) Repeat  $n$  times:
     $S \leftarrow$  random previously observed state
     $A \leftarrow$  random action previously taken in  $S$ 
     $R, S' \leftarrow Model(S, A)$ 
     $Q(S, A) \leftarrow Q(S, A) + \alpha [R + \gamma \max_a Q(S', a) - Q(S, A)]$ 
```

איור 11.12 אלגוריתם Dyna. שילוב של למידה מסימולציה המבוססת על מודל ידוע יחד עם סוכן שלומד בעזרת אלגוריתם Model-Free.

מלבד העובדה שתוך כדי הלמידה אנו משפרים את המודל, גם הדאטה שמיוצר הוא יותר רלוונטי ובעל ערך ללמידה – לא סתם מוגרלים מצבים אקראיים, אלא המצבים שהסוכן רואה קשורים לאסטרטגיה שנלמדה על בסיס הדאטה מהעולם האמיתי. באופן הזה יש שימוש כפול ו"ממצה" יותר של הדאטה מהעולם האמיתי – הוא גם מסייע לבנות את המודל וגם מסייע לסוכן להשתפר.

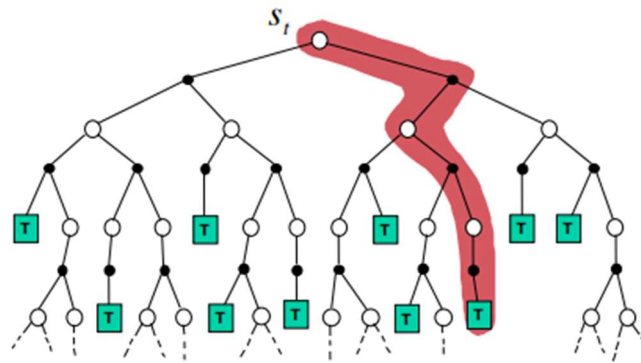
גרף ודוגמה – שקופיות 11, 12. שינוי מודל למודל לא נכון – שקופית 13 (מודל קבוע לא היה מצליח להשתפר לאחר השינוי).

11.4.2 Known Model – Tree Search

כאמור, אלגוריתם Dyna נעזר בידיעת המודל על מנת לאמן סוכן בעזרת אלגוריתם של Model-Free. יוצא מכך שאמנם הסוכן אינו תלוי במודל, אך עדיין האסטרטגיה אותה אנו מנסים ללמוד היא **גלובלית**, כלומר מנסה לדעת מה לעשות בכל סיטואציה אפשרית. עבור משימות מאוד מורכבות, כמו למשל משחק שחמט, זה עדיין לא מספיק, כיוון שאסטרטגיה כללית עבור כל המצבים אינה אפשרית ללמידה עקב ריבוי המצבים. בכדי להתמודד עם בעיות מורכבות

ניתן להפוך את הפתרון להיות **לוקאלי**, כלומר להסתכל על המצב הנוכחי בלבד ולנסות ללמוד בעזרת הסימולציות מה הפעולה שתביא את התוצאה הכי טובה.

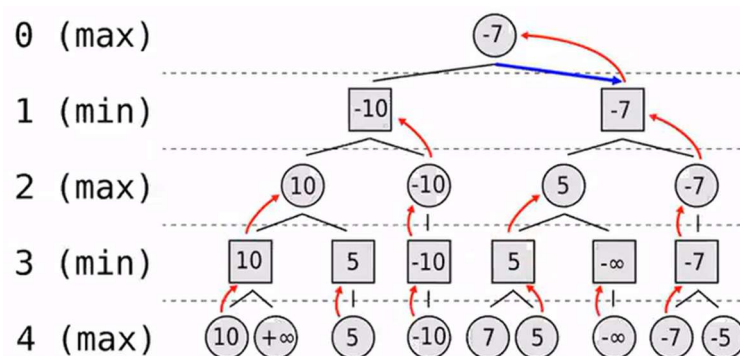
הרעיון של הלמידה הלוקאלית – חיפוש הפעולה הכי טובה עבור מצב נתון – קשור לתחום רחב שנקרא Forward Search (נעיר שבעיה זו קשורה להרבה תחומים ולא רק ל-RL, ופה נדון בעיקר במה שקשור לחיפוש באמצעות אלגוריתם של RL). תחום זה בא להתמודד עם מצבים בהם הבעיה הכללית היא מאוד מורכבת, אך אנו מנסים לפתור תת-בעיה המסתכלת רק על המצב הנוכחי ומה שמסתעף ממנו. ניתן לדמות את הסביבה לעץ בו כל צומת הוא מצב והוא מתפצל למצבים נוספים לפי הפעולות השונות האפשריות באותו מצב. בעזרת הסימולציה נבדוק כל מיני הסתעפויות שונות מאותו מצב, ונוכל לבחון מי מבניהן הטובה ביותר.



איור 11.13 Forward Search. חיפוש הפעולה הכי טובה ביחס למצב נתון. כל צומת בעץ מייצג מצב, וההסתעפויות מתפתחות בהתאם לפעולות האפשריות בכל מצב, כאשר terminal state מיוצג על ידי T. סימולטור המבוסס על מודל ידוע יכול לעזור לבחון את ההסתעפויות השונות.

כאשר כחלק מהבעיה יש גם יריב שהאסטרטגיה שלו אינה ידועה, יש כמה דרכים איך להתייחס להתנהגות שלו, כאשר הנפוצות שבהן הן לייחס ליריב את אותה אסטרטגיה כמו של הסוכן (Self-play) או לייחס אליו את האסטרטגיה האופטימלית (Minimax), כאשר יש להגדיר במדויק מהי אותה אסטרטגיה אופטימלית. אם נניח שמדובר במצבים של משחק סכום-אפס (בהם התגמול של שני המשתתפים הוא קבוע), אסטרטגיה אופטימלית מניחה שהיריב ינקוט בפעולות שימקסמו את התגמול שלו וימזערו עד כמה שניתן את התגמול של הסוכן. במקרה זה, הסוכן ירצה לבחור בפעולה שתמקסם את התגמול שלו ב-worst case, כלומר במקרה בו היריב בחר את הפעולות הטובות ביותר עבורו.

הבעיה שעולה מיד היא שבפעול אי אפשר להתחיל ממצב מסוים ולבדוק את כל ההסתעפויות עבור כל עומק אפשרי של העץ. לכן יש להגדיר עומק חיפוש מקסימלי, ואז כל מצב שהגענו אליו שנמצא באותו עומק יקבל ערך לפי נוסחה מוסכמת מראש (למשל – בשחמט ניתן לתת ערך למצב לפי שווים של כלי הסוכן הנמצאים באותו רגע על הלוח פחות שווים של כלי היריב). נתבונן על דוגמה של עץ Minimax בעומק 4 בכדי להבהיר את המצב אותו אנו מנסים ללמוד:



איור 11.14 עץ Minimax בעומק 4. העיגולים מסמנים את המצבים בהם תור הסוכן, והריבועים מסמנים את המצבים בהם תור היריב. החיצים האדומים מסמנים את הפעולות האופטימליות עבור היריב, ואנו מניחים שהיריב יבחר בהן.

הסוכן מתחיל מהמצב בשורה 0, ובעזרת סימולציה בודק את כל ההסתעפויות עד עומק 4. בשלב ראשון הסוכן משערך באמצעות הסימולציה רק את הערכים של הצמתים **בעומק המקסימלי**. אם זה מצב טרמינלי (כלומר המשחק הסתיים), הערך בעומק זה יהיה הערך שהתקבל בסימולציה, ואם המצב אינו טרמינלי אז הערך נקבע לפי היוריסטיקה קבועה מראש. לאחר מכן הסוכן הולך אחורה וממשקל את כל הצמתים בעומק הקודם תחת הנחת ה-worst case – כלומר מניחים שהיריב יבחר את ההתפצלות הגרועה עבור הסוכן (מסומן בחיצים האדומים). ככה למשל הצומת

השמאלי בעומק 3 יקבל את הערך 10, כי אנו מניחים שבצומת זה הסוכן יבחר ללכת שמאלה. בדומה, הצומת הימני בעומק 3 יקבל את הערך 7-, כי אנו מניחים שהוא יבחר ללכת שמאלה באותו צומת. כך נותנים ערכים לכל השורה הזאת, כאשר הכלל הוא שכל צומת מקבל את הערך המינימלי מבין כל אפשרויות הפיצול שלו. כעת עוברים לתת ערכים לשורה בעומק 2, כאשר כאן הכלל הוא הפוך – כיוון שהתור הוא של הסוכן, נניח שהוא יבחר בפעולה הכי טובה, ולכן כל צומת יקבל את הערך המקסימלי מבין הפיצולים האפשריים שלו. למשל הצומת השמאלי בעומק 2 יקבל את הערך 10, כיוון שזהו הערך של ההתפצלות הכי טובה שהסוכן יכול לבחור. באופן הזה עולים למעלה וממלאים את כל ערכי העץ עד השורש, עד שמקבלים ערך עבור המצב הנוכחי בו אנו נמצאים.

כמה הערות חשובות:

1. יתכן והיריב יעשה טעויות ולא יבחר את הצעדים הכי טובים עבורו, ואז בפועל נקבל ערך אחר למצב בו אנו נמצאים, אבל השיטה הזו היא יותר ביוטת היא להניח שהיריב יהיה אופטימלי ועל בסיס זה לתת ערכים לצמתים ולמצב הנוכחי. לאחר שביצענו את הפעולה הכי טובה בהתאם להערכה הנוכחית, היריב בתורו בוחר צעד ומתקבל מצב חדש, ואז הסוכן שוב יבצע סימולציות על מנת לחשב את כל ההסתעפויות מאותה נקודה ולתת ערך למצב החדש שהתקבל.

2. כיוון שהעץ גדל אקספוננציאלית, העומקים אותם ניתן לחשב הם לא גדולים, ולכן יש חשיבות להיוריסטיקה שקובעת את הערכים עבור הצמתים שבעומק המקסימלי שלא קיבלו ערך בסימולציה עצמה (כלומר מצבים שאינם טרמינליים).

3. במשחקים בהם יש גם גורם של מזל (כמו למשל הטלת קוביות), ניתן להוסיף שכבת של *chance nodes* בין כל שתי שורות בעץ, כאשר החישוב ביחס לשכבות החדשות הינו בעזרת התוחלת של התוצאה שלהן. עץ כזה נקרא **Expectiminimax Tree**.

אלגוריתם חיפוש זה עובד טוב עבור מגוון בעיות, אך הוא עדיין נתקל בבעיה עבור אתגרים בעלי מרחב מצבים גדול מאוד, כמו למשל המשחק Go. במקרים כאלה מספר הפיצולים גדול מדי בשביל שיהיה בר חישוב, וממילא לא ניתן לבחון את כל ההסתעפויות עבור עומק סטנדרטי, מה שלא מאפשר לחשב בצורה איכותית את ערכי הצמתים. לכן, במקום לבדוק את כל ההסתעפויות, ניתן להיעזר בסימולציה ולבחון רק חלק מהן, ועבור כל סימולציה נלך עד לעומק המקסימלי כדי שהתוצאה תהיה אמיתית ולא תלויה היוריסטיקה. כמובן שבחירת ההסתעפויות לא נעשית באופן אקראי, אלא היא מבוססת על מדיניות נלמדת של שני היריבים. באופן הזה בהתחלה הבעיה היא קלה כיוון שליריב אין אסטרטגיה טובה, אך ככל שאני משתפר כך גם היריב משתפר והבעיה הולכת ונעשית קשה, וממילא ניתן להשתפר עוד ועוד. אלגוריתם חיפוש זה נקרא **Monte-Carlo Search**, וניתן לנסח אותו באופן פורמלי כך:

נניח ונתונה אסטרטגיה התחלתית π שאינה מספיק טובה. נתון מצב s_t , ורוצים לדעת מה הערך של כל פעולה a אפשרית. נדגום K מסלולים עבור כל פעולה a אפשרית, ניעזר בסימולציה כדי לבצע את המסלול עד הסוף, ונחשב את תוחלת הרווח של כלל הסימולציות. באופן הזה, עבור המצב s_t הערך של פעולה a יהיה:

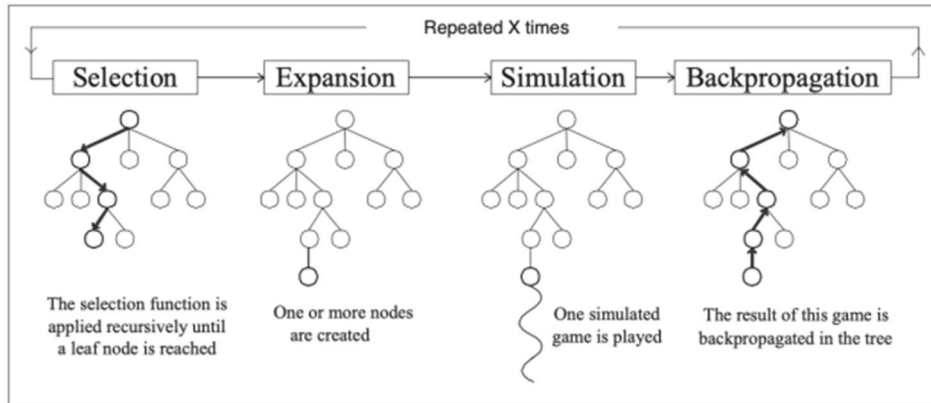
$$Q^\pi(s_t, a) = \frac{1}{K} \sum_k \sum_i \gamma^i \mathcal{R}_{t+i+1}^k$$

לאחר שחישבנו את הערך של כל הפעולות, נבחר את הפעולה בעלת הערך הכי גבוה:

$$\max_a Q^\pi(s_t, a)$$

יש להעיר שכדי שהאלגוריתם יעבוד, צריך שהאסטרטגיה ההתחלתית π תהיה ברמה סבירה, כיוון שאם היא גרועה, השיפור יהיה מאוד איטי ולא מספיק.

ניתן לקחת את הרעיון הזה ולהרחיב אותו לחיפוש על עץ כפי שתיארנו קודם. כאמור לעיל, האתגר המרכזי בחיפוש על עץ נעוץ בכך שהוא גדל אקספוננציאלית ולא ניתן לחשב באופן יעיל בעיות עם מספר מצבים גדול מאוד. בכדי להתגבר על כך ניתן להשתמש באלגוריתם **Monte-Carlo Tree Search (MCTS)**, שהוכיח את עצמו כמאוד חזק ואף יעיל לבעיות גדולות מאוד. אלגוריתם זה למעשה משלב בין שני חלקים שכל אחד מהם בפני עצמו לא מספיק טוב – מצד אחד יש לנו אסטרטגיה התחלתית, אך היא בינונית ולא מספיק איכותית, ומצד שני יש לנו סימולציה המבוססת על ידיעת המודל, אך היא אינה יכולה לבדוק את כל האפשרויות. הרעיון של MCTS הוא לקחת אסטרטגיה בינונית ולשפר אותה באמצעות בדיקה מושכלת של העתיד – ההרחבה של העץ לא נעשית באמצעות בחינת כל האפשרויות, אלא באמצעות בחירה של כמה אפשרויות שנראות סבירות ביחס לאסטרטגיה הנוכחית. האלגוריתם ניתן לתיאור באמצעות ארבעה שלבים, שחוזרים על עצמם באופן איטרטיבי x פעמים עבור כל פעולה a :



איור 11.14 אלגוריתם Monte-Carlo Tree Search: 1. בחירת הצומת הבא אותו רוצים לבחון. 2. הרחבת העץ מאותו צומת. 3. הרצת סימולציה מאותו צומת עד הסוף (=הגעה למצב טרמינלי). 4. נתינת ערך לכל צומת מהצומת הנבחר ואחורה עד השורש.

כאמור, האלגוריתם מורכב מארבעה שלבים:

1. בחירה (Selection) – ראשית יש לבחור את המצב s עבורו אנו רוצים לבדוק מה הפעולה הכי טובה במצב זה. לשם כך נבצע סדרה של צעדים עד שנגיע למצב s_t , כאשר את הצעדים ניתן לבחור על בסיס ה-score שלהם ואפשר גם לשלב רנדומליות עבור exploration כפי שנראה בהמשך.
2. הרחבה (Expansion) – משהגענו לאותו מצב s_t נרחיב אותו בצומת נוסף על ידי בחירה של אחת מהפעולות האפשריות. את הפעולה ניתן לבחור באופן רנדומלי או על בסיס האסטרטגיה הנתונה (שכאמור היא רחוקה מאופטימליות).
3. הרצת סימולציה (Simulation) – כעת נריץ סימולציה מהמצב החדש עד הסוף ונבדוק כיצד הסתיים המשחק. בהתאם לתוצאת המשחק ניתן לאותו מצב חדש ערך.
4. נתינת ערכים לצמתים (Backpropagation) – נחלחל אחורה את הערך שהתקבל ונעדכן את כל הצמתים שנבחרו. אם למשל המשחק הסתיים בניצחון, אז נעדכן את ערך הצומת החדש (שהרחיב את העץ) ל-1/1 (=ניצחון אחד מתוך משחק אחד). בהתאם נעדכן את כל יתר המצבים שעברנו בהם – אם למשל השורש ממנו יצאנו היה בעל יחס של 3/5, אז נעדכן אותו ל-4/6.