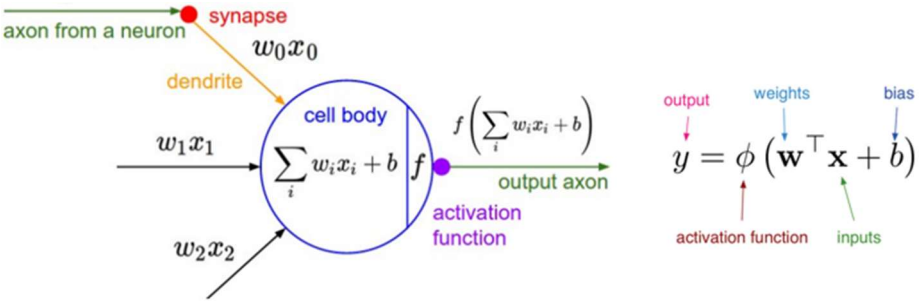
4. Deep Neural Networks

פרק זה עוסק ברשתות נוירונים עמוקות. רשת נוירונים הינה חיבור של יחידות עיבוד בסיסיות (נוירונים מלאכותיים) על ידי משקלים ופונקציות לא ליניאריות. 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 מכליל בצורה טובה את הדאטה עליו הוא מאומן.

4.1 MLP – Multilayer Perceptron

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 ליניארית. באופן סכמתי ניתן לתאר את הנוירון הבודד כך:

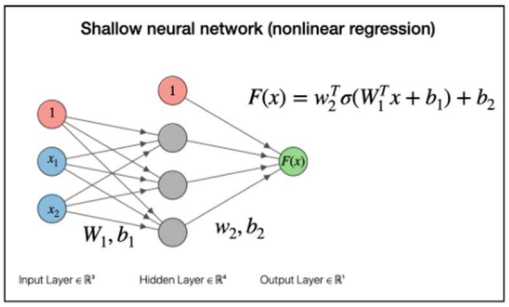


איור 4.1 ייצוג של נוירון מלאכותי, המקבל קלט, סוכם אותו ומעביר את התוצאה בפונקציית הפעלה.

הקלט של הנוירון הוא סט input מוכפל במשקלים: w^Tx כאשר $w, x \in \mathbb{R}^d$, ואיבר bias. הקלט עובר דרך סוכם, ומתקבל הביטוי $\sum_{i=1}^{d} w_ix_i + b$. לאחר מכן הסכום עובר דרך פונקציית הפעלה, ומתקבל המוצא $f\left(\sum_{i=1}^{d} w_ix_i + b\right)$. במקרה הפרטי בו פונקציית ההפעלה 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 ליניארי, 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: $w_2 \cdot f_1(w_1^Tx + b_1) + b_2$. ביטוי זה עובר בפונקציית הפעלה נוספת ומתקבל המוצא:

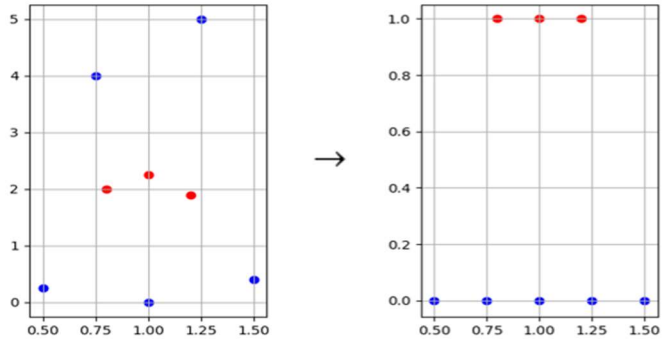
$$\hat{y} = f_2(w_2 \cdot f_1(w_1^Tx + b_1) + b_2)$$



איור 4.2 רשת נוירונים בעלת שכבה חבויה אחת.

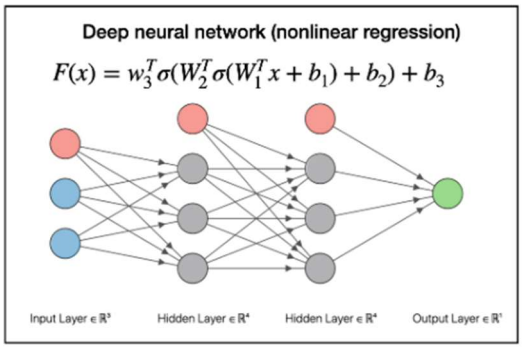
חשוב להדגיש שמטרת הרשת היא לבצע פעולות לא ליניאריות על ה-input כך שהוא יסודר באופן חדש הניתן להפרדה ליניארית. למעשה לא מבטלים את ההפרדה הליניארית הנעשית בעזרת הרגרסיה, אלא מבצעים לפניה שלב מקדים של העתקה לא ליניארית. תהליך זה נקרא למידת ייצוגים (representation learning), כאשר בכל שכבה

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 (רגרסיה לינארית או לוגיסטית).



איור 4.3 העתקה לא לינארית של דוגמאות על ידי המשוואה $\tilde{y} = \begin{cases} 1, if\ 3 \le (x^2 + y^2) \le 8 \\ 0, else \end{cases}$. 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 נוירונים בעלת שתי שכבות חבויות.

רשת נוירונים 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.

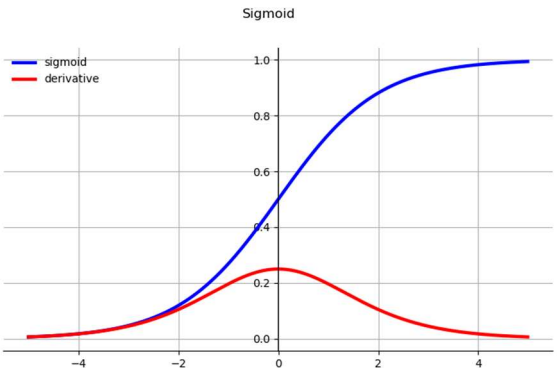
4.1.2 Activation Function

האלמנט המרכזי בכל נוירון הוא פונקציית ההפעלה, ההופכת אותו ליחידת עיבוד לא לינארית. יש מספר פונקציות הפעלה מקובלות – Sigmoid, tanh, ReLU.

Sigmoid

פונקציית הסיגמואיד הוצגה בפרק של רגרסיה לוגיסטית, וכעת נרחיב עליה. הפונקציה והנגזרת שלה הן מהצורה:

$$\sigma(z) = \frac{1}{1 + e^{-z}}, \frac{\partial}{\partial z}\sigma(z) = \sigma(z)\big(1 - \sigma(z)\big)$$



איור 4.5 פונקציית סיגמואיד והנגזרת שלה.

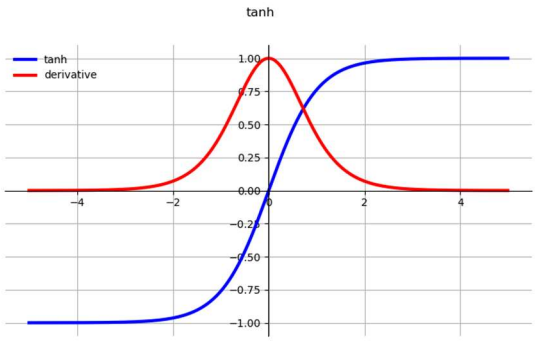
יש לפונקציה זו שלושה חסרונות:

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:

$$\tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}, \frac{\partial}{\partial z}\tanh(z) = 1 - (\tanh(z))^2$$



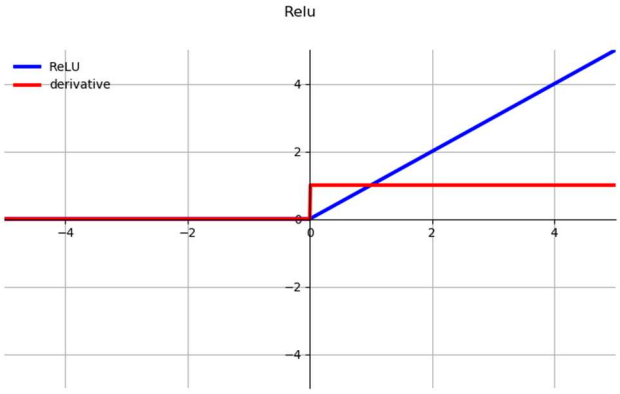
איור 4.6 פונקציית טנגנס היפרבולי והנגזרת שלה.

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:

$$ReLU(z) = \max(0, z), \frac{\partial}{\partial z} ReLU(z) = 1_{\{z>0\}} = \begin{cases} 1, z > 0 \\ 0, z \leq 0 \end{cases}$$



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 הנוירונים מתאפסים וזה יחסית בזבזני. כדי להתגבר על הבעיה האחרונה ניתן להשתמש בוורסיות של הפונקציה, כמו למשל PReLU ו-ELU:

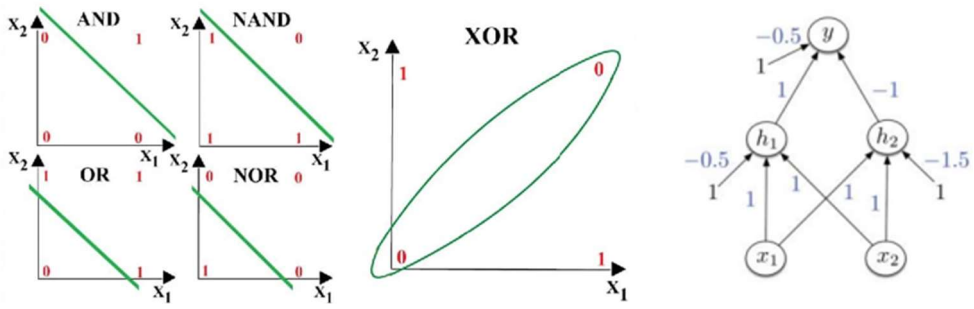
$$PRelU(z) = \max(\alpha x, x)\,, ELU(z) = \begin{cases} x, x > 0 \\ \alpha(e^x - 1) \end{cases}$$

בפונקציית PReLU, המקרה הפרטי בו $\alpha = 0.1$ 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 נוירונים בעלת שכבה חבויה אחת ניתן לייצר מודל שקול לאופרטור Xor.

בדוגמא המובאת באיור הכניסות עוברות דרך שכבה חבויה אחת בעלת שני נוירונים - h_1, h_2, המקבלים בנוסף גם bias. פונקציית ההפעלה של נוירונים אלו היא פונקציית הסימן, וניתן לכתוב את המוצא של שכבה זו כך:

$$h_1 = sign(x_1 + x_2 - 0.5), h_2 = sign(x_1 + x_2 - 1.5)$$

לאחר השכבה החבויה הנוירונים מחוברים למוצא, שגם לו יש bias, והסכום של הכניסות וה-bias עוברים במסווג:

$$y = sign(h_1 - h_2 - 0.5) = \begin{cases} 1\ if\ h_1 - h_2 - 0.5 > 0 \\ 0\ if\ h_1 - h_2 - 0.5 < 0 \end{cases}$$

נבחן את המשמעות של הנוירונים: הנוירון h_1 יהיה 0 אם שתי הכניסות שוות 0, אחרת הוא יהיה שווה 1. הנוירון 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 $y = sign(0.5) = 1$. 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 נוירונים 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 והנוירונים המבצעים פעולות לא לינאריות. בעוד הפעולות אותן מבצעים הנוירונים 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 בנוירונים 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*:*

$$\overline{w_N} = 1$$

For i in $\mathrm{N} - 1 \dots 1$:

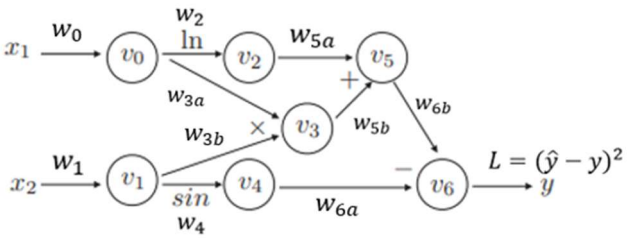
$$\frac{\partial L}{\partial w_i} = \frac{\partial L}{\partial w_N} \cdot \frac{\partial w}{\partial w_{N-1}} \cdots \frac{\partial w_{i+1}}{\partial w_i}$$

$$\overline{w_i} = w_i - \epsilon \frac{\partial L}{\partial w_i}$$

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 בדוגמא 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 לינארית, 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:

$$v_0 = x_1, v_1 = x_2$$

$$v_2 = \ln(v_0), v_3 = v_0 \cdot v_1, v_4 = \sin(v_1)$$

$$v_5 = v_2 + v_3$$

$$\hat{\boldsymbol{y}} = v_6 = v_5 - v_4$$

לאחר שבוצע החישוב עבור $\hat{y}$, ניתן לחשב את אחורה את הנגזרות החלקיות, בעזרת כלל השרשרת:

$$\frac{\partial L}{\partial w_{6a}} = -1, \frac{\partial L}{\partial w_{6b}} = -1$$

$$\frac{\partial L}{\partial w_{5a}} = \frac{\partial L}{\partial w_{6b}} \frac{\partial w_{6b}}{\partial w_{5a}} = -1 \cdot 1 = 1, \qquad \frac{\partial L}{\partial w_{5b}} = \frac{\partial L}{\partial w_{6b}} \frac{\partial w_{6b}}{\partial w_{5b}} = -1 \cdot 1 = -1$$

$$\frac{\partial L}{\partial w_4} = \frac{\partial L}{\partial w_{6a}} \frac{\partial w_{6a}}{\partial w_4} = -1 \cdot (-\cos w_4) = \cos w_4$$

$$\frac{\partial L}{\partial w_{3a}} = \frac{\partial L}{\partial w_{6b}} \frac{\partial w_{6b}}{\partial w_{5b}} \frac{\partial w_{5b}}{\partial w_{3a}} = -1 \cdot 1 \cdot w_{3b}, \qquad \frac{\partial L}{\partial w_{3b}} = \frac{\partial L}{\partial w_{6b}} \frac{\partial w_{6b}}{\partial w_{5b}} \frac{\partial w_{5b}}{\partial w_{3b}} = -1 \cdot 1 \cdot w_{3a}$$

$$\frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial w_{6b}} \frac{\partial w_{6b}}{\partial w_{5a}} \frac{\partial w_{5a}}{\partial w_2} = -1 \cdot 1 \cdot \frac{1}{\ln w_2}$$

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 backpropagation הוכיח את עצמו, למרות שהמשקלים לא בהכרח הגיעו לאופטימום שלהם.

4.3 Optimization

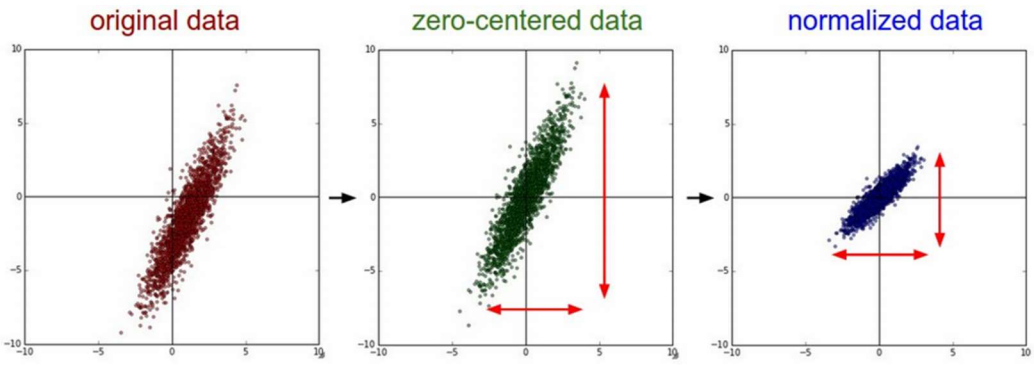
מציאת אופטימום למשקלים על פני כל העומק של הרשת היא בעיה לא קמורה, ולכן אין לה בהכרח מינימום גלובאלי. לכן מלבד עדכון המשקלים בשיטת backpropagation יש לבצע אופטימיזציות נוספות על הרשת על מנת לשפר את הביצועים שלה.

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:

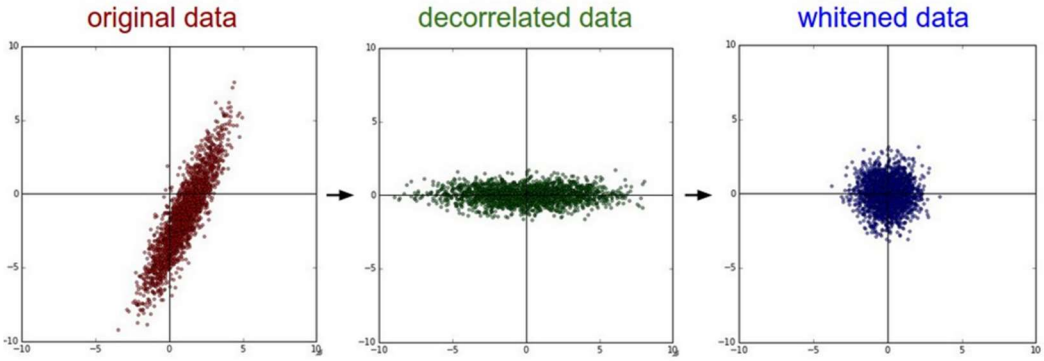
$$X_i = \frac{X_i - \mu_i}{\sigma_i}$$

ובאופן חזותי:



איור 4.10 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 4.11 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 , גם כאן הגבולות יהיו תלויים בגודל השכבות – $U\left[-\frac{1}{\sqrt{n}}, \frac{1}{\sqrt{n}}\right]$.

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 החבויות. התהליך נעשה בשלושה שלבים:

א. עבור כל נוירון בעל פונקציית הפעלה לא לינארית, 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 כך:

$$\mu_{\mathcal{B}} = \frac{1}{m}\sum_{i=1}^{m} x_i, \sigma_{\mathcal{B}}^2 = \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2$$

$$\hat{x}_i = \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}}$$

$$y_i = \gamma \hat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i)$$

כאשר γ, β הם פרמטרים נלמדים (עבור כל נוירון יש פרמטרים שונים).

בשלב המבחן, השונות והתוחלת שבעזרתם מבצעים את הנרמול אינם נלקחים מהיציאות של הנוירונים, 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, אופטימיזציית רשת נוירונים 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 אוכף 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 טיפוסי 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:

$$m_{i+1} = \mu m_i - \epsilon \frac{\partial L(w)}{\partial w} = \mu^2 m_{i-1} - \mu\epsilon \frac{\partial L(w)}{\partial w} - \epsilon \frac{\partial L(w)}{\partial w} = \cdots = -\epsilon \frac{\partial L}{\partial w}(1 + \mu + \mu^2)$$

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 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:

א. שינוי הפרמטר בכל כמה Epochs. מספרים טיפוסיים הם הקטנה בחצי כל 5 epochs או חלוקה ב10 כל 20 epochs. באופן כללי ניתן לומר שכאשר גרף הלמידה של ה-validation בקושי משתפר, יש להקטין את ה-lr.

ב. 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.

ג. 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:

$$\alpha_i = \beta_1 \alpha_{i-1} + (1-\beta_1)\left(\frac{\partial L}{\partial w}\right)^2, m_i = \beta_2 m_{i-1} + (1-\beta_2)\frac{\partial L}{\partial w}$$

$$\hat{\alpha}_i = \frac{\alpha_i}{1-\beta_1^i} \hat{m}_i = \frac{m_i}{1-\beta_2^i}$$

$$w_{i+1} = w_i - \frac{\epsilon}{\sqrt{\widehat{\alpha}_i + \epsilon_0}} \widehat{m}_i$$

מספרים טיפוסיים: $\beta_1 = 0.9, \beta_2 = 0.99, \epsilon = 10^{-2}\ or\ 5^{-4}$. 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 הראשוניות, 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.

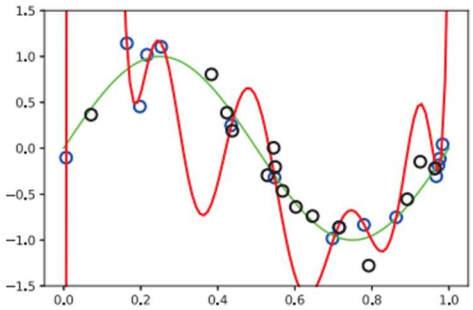
4.4 Generalization

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 נוירונים, 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 יתר. ככל שהגרף של ה-validation accuracy קרוב יותר ל-training accuracy, כך יש פחות overfitting.



איור 4.12 בדיקת overfitting בעזרת validation set. הנקודות הכחולות שייכות ל-training והשחורות שייכות 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, ונבחן אותם כעת.

4.4.2 Weight Decay

בדומה לרגולריזציה של linear regression, גם ברשת נוירונים 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:

$$Cost(w;x,y) = L(w;x,y) + \lambda\sum_i |w|$$

ואז העדכון יהיה:

$$w_{i+1} = w_i - \epsilon\left(\frac{\partial L}{\partial w} + \lambda \cdot sign(w)\right)$$

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 נוירון אחד או יותר. אם יש רשת מסוימת ומוחקים את אחד הנוירונים – למעשה מקבלים רשת אחרת, ובפועל אפשר לקבל ensemble בעזרת רשת אחת שכל פעם מוחקים ממנה נוירון אחד או יותר. היתרון של יצירת ensemble בדרך הזו הוא שהרשתות חולקות את אותן פרמטרים ולבסוף מקבלים רשת אחת מלאה עם כל הנוירונים והמשקלים. בפועל עבור כל דגימה מגרילים רשת (מוחקים כל נוירון בהסתברות $p = 0.5$) וכך לומדים במקביל הרבה רשתות שונות עם אותן פרמטרים. באופן הזה כל נוירון מוכרח להיות יותר משמעותי בלי אפשרות להסתמך על נוירונים 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 הנוירונים, כאשר מחלקים את כל המשקלים בחצי. הסיבה לכך היא שניתן להניח שבשלב האימון חצי מהפעמים המשקל היה 0 כיוון שהנוירון המקושר אליו נמחק, ובחצי מהפעמים היה משקל שנלמד. ניתן גם לקחת הסתברות אחרת למחיקת נוירונים, 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 (rescaling) של התמונה בפקטור מסוים – בדרך כלל הפקטור שייך לתחום $\left[\frac{1}{1.6}, 1.6\right]$.
- 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