digital whisper
Technologic Papers

בונים רשת ניורונים

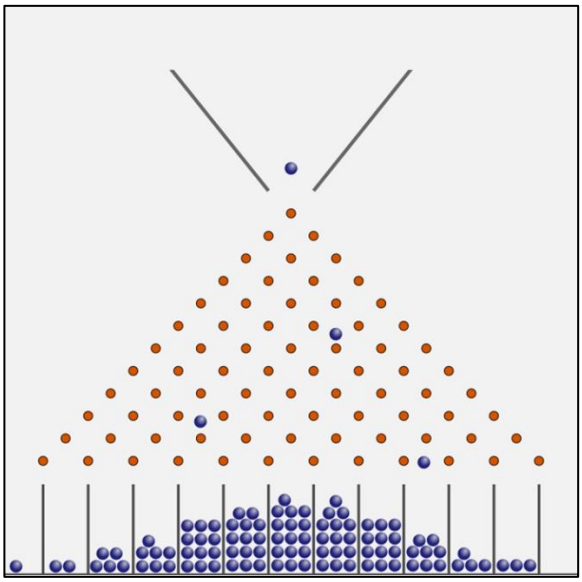
מאת שלום דימנט

הקדמה

בשנים האחרונות (ובטח בשנה האחרונה, מאז ש-ChatGPT נכנס לחיים שלנו) כולם שומעים על "בינה מלאכותית", אבל מה זה? ומה זה שונה מתכנות רגיל? במאמר הזה ננסה להבין מה זה אומר בינה מלאכותית, מה הכוונה "לאמן" מודל ואפילו נבנה מודל קטן שמקבל תמונה של מספר ויודע לזהות איזה מספר יש בתמונה. במאמר הזה אני אשתדל להימנע ממתמטיקה ולנסות להסביר את האינטואיציה שמאחורי הדברים ולכן אין פה נוסחאות בכלל (חוץ ממקום אחד שאני מדבר שם על נגזרות, שזה חומר שיש בבגרות. רציתי להראות שימוש לדברים שלמדנו בתיכון והיו נראים לנו לא קשורים לעולם).

מה זו בינה מלאכותית?

על מנת להבין מה זו בינה מלאכותית, בואו נראה דוגמא. נסו לדמיין תיבת גלטון, זהו לוח אנכי עם מספר שורות של מסמרים, כאשר מניחים כדור למעלה, הוא פוגע במסמרים וקופץ עד שהוא נופל לאחת התיבות בתחתית הלוח.



[תיבת גלטון. מקור: https://commons.wikimedia.org/wiki/File:Galton_Box.svg]

בהנחה שיש לנו שליטה מוחלטת בצורה שבה הכדור נכנס, הוא תמיד יפול לאותה התיבה. התיבה הספציפית שאליה הכדור יפול תלויה בגודל הכדור, משקל הכדור, החומר ממנו הוא עשוי, גודל המסמרים, מיקום המסמרים ופרמטרים נוספים.

בואו נדבר על נגר בשם נח, לנח יש כדורים בשלושה משקלים: קילו, 2 קילו ו-3 קילו. הוא רוצה לבנות תיבה שתמיין לו את הכדורים באופן הבא- כדור ששוקל קילו בתיבה מספר 1, כדור ששוקל 2 קילו בתיבה מספר 2, וכדור ששוקל 3 קילו, בתיבה מספר 3.

נח שלנו בונה תיבה וכמו נגר מקצועי, לפני שהוא תוקע את המסמרים, הולך להכין לו קפה שחור. בינתיים בא חם, הבן שלו ומתוך התלהבות מהקופסא, הוא תוקע את המסמרים בצורה אקראית. כאשר נח חוזר ורואה מה עשה הבן שלו הוא מנסה להכניס את הכדורים וכל כדור נופל לתיבה אחרת.

התכנון הקפדני של נח נהרס, ועכשיו הוא צריך לנסות לסדר את התיבה שיצאה כך שתצליח למיין את הכדורים. אין לו ברירה אלא לעשות זאת על ידי ניסוי וטעיה. נח מכניס כדור ברזל של קילו, והכדור נופל לתיבה מספר 2. אז הוא מזיז קצת את אחד המסמרים כדי שהכדור יפול לכיוון הנכון, ומכניס כדור עץ של 2 קילו. הכדור נופל לתיבה השלישית, אז נח מקפל מסמר אחר. וכך עוד כדור ועוד כדור, ובכל כדור הוא מתקן את אחד המסמרים. כמובן שכאשר נח מתקן את המסמרים כך שיתאימו לכדור עץ של 2 קילו, הוא פוגע בהתאמה שלהם לכדור מתכת של 2 קילו, אבל אחרי תיקון בעזרת אלפי כדורים הוא מכוון את התיבה כך שתפיל כל כדור לתיבה שלו.

כנמשל, אצלנו הקלט של התיבה היה כדור במשקל מסוים, והפלט היה התיבה שאליו הכדור נפל, שמייצגת את המשקל שלו. בצורה דומה, כל מודל של בינה מלאכותית בנוי מקלט ופלט, לדוגמא מודל אחד מקבל כקלט טקסט עם תיאור של תמונה (פרומפט) וכפלט הוא יוציא את התמונה (מודלים לדוגמא שאולי שמעתם עליהם: midjourney, DALL-E, stable-diffusion). מודל אחר יקבל תמונה כקלט, ויוציא ריבועים שמקיפים את האובייקטים המופיעים בתמונה (yolo לגרסאותיו), מודל אחר מקבל טקסט כקלט ומוציא כפלט קובץ שמע של הטקסט בקול אנושי (מודלים מסוג TTL), ומודל נוסף יקבל כקלט טקסט, וכפלט טקסט אחר (לדוגמא GPT, bard, claude) ואפילו מודל שמקבל טקסט ומוציא מוסיקה (Melobytes) או שיר (suno).

האימון

כדוגמא לכך, נתאר מודל שאמור לזהות רכבים (זה מה שאני עושה בעבודה ;)), בתחילה כשנכניס את התמונה, המודל יזרוק ריבועים בצורה אקראית, וכמו שנח כיוון את המסמרים בתיבה, כך גם אנחנו ולאט לאט נכוון את מאפייני המודל (ה"פרמטרים" שלה) כך שיסמן את הריבועים מסביב לרכבים שמופיעים בתמונה.

זאת אומרת, שכמו בתיבה שנח בנה בשביל לכוון את הרשת אנחנו צריכים זוגות של נתונים, משהו שנכנס לרשת (קלט) ומשהו שנרצה שהרשת תוציא (פלט). לדוגמא הקלט: כדור, הפלט: המשקל שלו. קלט: תמונה. פלט: מיקום הרכבים בתמונה וכן הלאה. הנקודה החשובה היא שבשביל הלימוד של הרשת, אנחנו צריכים לדעת עבור קלט מסוים, מהו הפלט שנרצה שהרשת תיתן.

לאחר שמתקבלים הזוגות האלה, מתחיל תהליך הכיוון של הרשת, אשר נקרא "אימון". באימון מכניסים את הקלט לרשת ומכוונים את הפרמטרים שיתאימו לפלט המתאים, עוד תמונה ועוד תמונה, כך עשרות אלפי, מאות אלפי, ומליוני פעמים עד שמגיעים למצב שבו הרשת כבר לא מצליחה להשתפר עוד. בשלב זה, תהליך האימון מסתיים והמודל מוכן לפעולה. השלב הזה נקרא "שלב הלמידה" מכיוון שבו הרשת לומדת מה היא צריכה לעשות.

השלב הבא הוא שלב ההרצה של הרשת, כאשר משתמשים בה בפועל כדי לזהות את מה שרצינו (או להשלים טקסט, ליצור תמונה/שיר, תלוי בסוג הרשת). בדוגמא שלנו, זה יהיה השלב שבו נח יקח כדורים שהוא לא יודע מה המשקל שלהם כדי למדוד את המשקל.

הנקודה החשובה היא, שבניגוד לאלגוריתמיקה "מסורתית" שבה אנחנו מתכננים ויודעים מה המטרה של כל מודול, ברשתות נוירונים אנחנו לא מתכננים (וברוב הפעמים גם לא יודעים) מה יעשה כל חלק במודל, הפרמטרים נקבעים לפי ניסוי וטעיה בשלב הלמידה של הרשת.

כמובן שבכל אחד מהשלבים האלה יש תתי שלבים, ויש טריקים לשיפור הביצועים, אבל השלבים האלה קיימים בכל מודלי הבינה המלאכותית.

לפני שניגש לקוד, נסביר כמה מושגים מתמטיים.

קצת קצת מתמטיקה

סקלר: זה מספר רגיל (לדוגמא ציון של מבחן בודד).

וקטור: מערך של מספרים (לדוגמא הציון של כל המבחנים של התלמיד במהלך הסמסטר).

מטריצה: מערך דו ממדי של מספרים (טבלה, לדוגמא הציון של כל התלמידים בכל המבחנים, כל שורה מייצגת רשימת הציונים של תלמיד ספציפי, וכל עמודה זו רשימת הציונים בקורס מסוים).

טנזור: מערך עם יותר משני מימדים (לדוגמא הציונים של כל התלמידים בכל הקורסים במהלך השנים, כל שנה זו מטריצה (כמו דף אחד), ושנה אחרת זה דף אחר, אם כן יש פה שלושה מימדים).

איך תמונה שמורה במחשב

תמונה צבעונית שמורה במחשב כטנזור, יש לה שלושה מימדים: רוחב וגובה (הרזולוציה) וערוצים (RGB) - עבור כל פיקסל, ישנם שלושה מספרים המייצגים את כמות האדום, הירוק והכחול של אותו הפיקסל.

שכבות ברשת נוירונים

מכפלת מטריצות: אני לא ארחיב פה מה זה, אפשר להסתכל בהסבר המצויין בוויקפדיה. לענייננו, מספיק להגיד שישנה הגדרה מתמטית שנקראת "מכפלת מטריצות" שבה אם מכפילים מטריצה אחת בשניה ומקבלים מטריצה שלישית.

קונבולוציה: גם במקרה הזה, אני לא ארחיב על ההסבר המדוייק של קונבולוציה, מספיק להגיד שקונבולוציה היא פעולה שעושים על מטריצה ומקבלים מטריצה חדשה. בניגוד למכפלת מטריצות, קונבולוציה "מודעת" להקשר המרחבי (התוצאה של קונבולוציה על פיקסל ספציפי מושפע גם מהערכים של הפיקסלים שסביבו).

גודל הקרנל: לקונבולוציה יש מושג שנקרא "גודל הקרנל", גודל הקרנל הוא גודל האיזור של ההשפעה מסביב לפיקסל (לדוגמא בקונבולוציה עם גודל קרנל של 3x3, התוצאה של הקונבולוציה על פיקסל ספציפי תהיה מושפעת מהפיקסל עצמו ועוד 8 הפיקסלים שמסביבו).

strides: הקפיצות של הקונבולוציה. לדוגמא קונבולוציה עם stride של 2, תתייחס לכל פיקסל שני ותדלג על שאר הפיקסלים.

max_pool: זמן הריצה של כל שכבה תלוי במספר הפרמטרים שהגיעו אליה (השכבה הראשונה לדוגמא רצה על כל התמונה, ברור שעבור תמונה גדולה זמן הריצה יהיה גדול יותר). אנחנו רוצים להקטין את המימדים של המידע שעובר ברשת, אבל מצד שני לא רוצים לאבד מידע. לכן הרבה פעמים אחרי שכבת קונבולוציה (שמודעת לערכים של כל הסביבה) נשים שכבת max_pool. גם בשכבה הזו יש גודל קרנל מסוים, לדוגמא עבור גודל קרנל של 2x2 ישנו חלון בגודל 2x2 והערך שעובר הלאה הוא הערך הגבוה ביותר מבין 4 הערכים שבחלון.

ישנן סוגי שכבות נוספות, אבל הבאתי את השכבות המרכזיות שנשתמש בהם ברשת שאנחנו הולכים לבנות.

הסבר על אימון רשתות נוירונים

loss: הוא מדד הטעות שלנו, מה גודל השגיאה של הרשת. אנחנו הולכים לבנות רשת שמקבלת תמונה עם מספר ואומרת מה המספר שבתמונה. הפלט של הרשת יהיה וקטור באורך 10 (מספר הקבוצות שלנו) כאשר המספר שנמצא בכל מקום במערך יהיה ההסתברות שבתמונה יש את הקבוצה שמתאימה לו.

נגזרת: כמו שלמדנו בתיכון (למי שזוכר), הנגזרת היא מדד לכמות ההשפעה של הפרמטרים השונים. לדוגמא כזכור (או שלא ☺) הנגזרת של הפונקציה: $y = x^2$ היא: $y' = 2x$. זאת אומרת שכאשר x שווה 0, הנגזרת שווה 0, המשמעות של זה היא שכאשר x משתנה באיזור של 0, הוא ישנה את הפונקציה קצת.

אם x=0, אז $y(x = 0) = 0^2 = 0$. ואם נקדם את x בחצי נקבל ש: $y(x = 0.5) = 0.5^2 = 0.25$. זאת אומרת שבאיזור של $x = 0$, שינוי של חצי ב-x, שינה את y רק ברבע. אבל במקרה ש: $x = 10$ הנגזרת היא 20 שהרי: $y'(x = 10) = 2 * 10 = 20$, וזה מצביע על שינוי גדול יותר של הפונקציה מכיוון שכאשר $x = 10$, אז $y(x = 10) = 10^2 = 10 * 10 = 100$. ואם כמו בדוגמה הקודמת, נקדם את x בחצי נקבל ש: $y(x = 10.5) = (10.5)^2 = 10.5 * 10.5 = 110.25$ ואז יצא לנו שכאשר $x = 0$, שינוי של חצי ב-x שינה את y ב-0.25, וכאשר $x = 10$, שינוי של חצי ב-x שינה את y ב-10.25, שזה שינוי הרבה יותר גדול.

כעת, אם יש לנו משוואה עם 2 משתנים: $y = x^2 + z$. אז:

- $y'(x) = 2x$ (הנגזרת לפי x)
- $y'(z) = 1$ (הנגזרת לפי z)

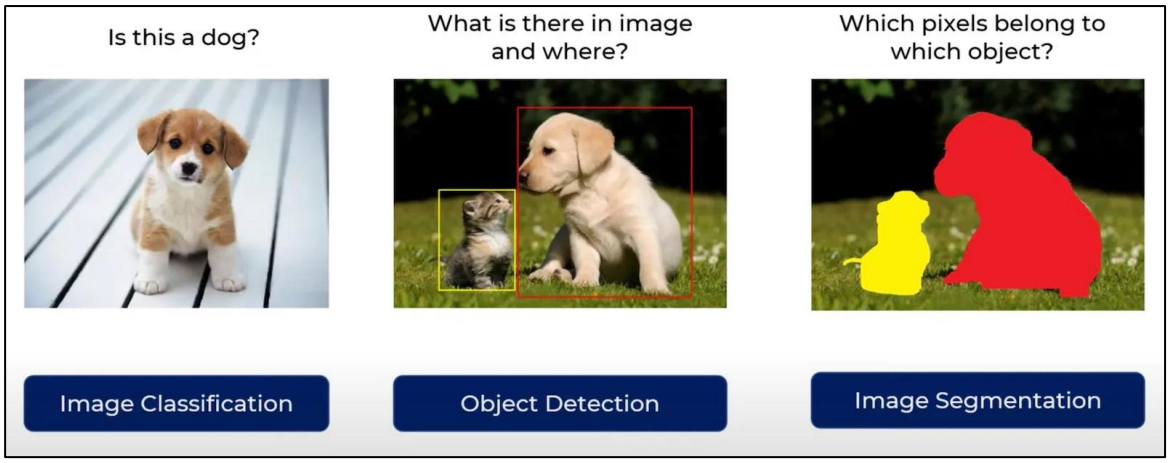
בדוגמא הזו, התוצאה הסופית מושפעת מ2 משתנים שונים, x ו-z. ולכל אחד מהם יש השפעה שונה. ההשפעה של x היא כמו שראינו בדוגמא הקודמת, וההשפעה של z היא קבועה ותמיד 1 (שינוי של 1 ב-z ישפיע על התוצאה הסופית ב1). ולכן אם אנחנו רוצים לשנות את התוצאה של המשוואה, איזה פרמטר נשנה? זה תלוי בהשפעה של כל אחד מהם על התוצאה הסופית, כמו שראינו, כאשר x=0 ההשפעה שלו אפסית, ולכן אם x הוא אפס, נשנה אותו מעט, כי הוא השפיע קצת על התוצאה הסופית. אבל אם x הוא 10, נשנה אותו יותר, כי ההשפעה שלו על התוצאה הסופית היא גדולה יותר.

אם ניקח לדוגמא את ספינת הפיראטים של קפטן הוק, הקפטן רוצה להגיע לאי הילדים ומפנה את הספינה לכיוון, ופתאום שומע תיק תק תיק תק, התנין מגיע לכיוונם. הקפטן מת מפחד ויורד לבטן האוניה. לפני שהוא יורד הוא משאיר הוראות לצוות איך להפליג. אחרי כמה זמן התנין מתייאש ושוחה לו משם והקפטן חוזר לסיפון. להפתעתו, הוא מגלה שהם פספו את האי בכמה קילומטרים. הוא מתחיל לחקור את צוות הספינה על מנת להבין מה הסיבה לפספוס ורואה שהרבה מהמלחים טעו. כל מלח יקבל עונש, שיעור עם הקפטן להבנת הטעות שלו. אבל המלח שאחראי על הנקיון בספינה בכלל לא היה אשם בפספוס של אי הילדים ולכן הקפטן פטר אותו מהשיעור. לעומתו הנווט היה אשם מאד, ולכן צריך לעבור סדרה של 12 שיעורים עם הקפטן. כך גם במקרה שלנו, ראינו שבעזרת הנגזרת אפשר להבין מה ההשפעה של כל פרמטר על השגיאה הסופית. ולכן נוכל לתקן את הערכים של הפרמטרים השונים ברשת לפי גודל ההשפעה שלהם על השגיאה (פרמטר שהשפיע הרבה על השגיאה נשנה אותו הרבה, ופרמטר שהשפיע קצת נשנה קצת).

סוגי רשתות ב-vision

ישנם 3 סוגי רשתות מרכזיים בעיבוד תמונה:

1. Classification
2. Detection
3. Segmentation



[מקור: codebasics]

Classification:

ברשת הזו יש מספר קבוצות (לדוגמא חתולים וכלבים), והרשת מקבלת כקלט תמונה שיש בה את אחד מהאובייקטים האלה (לדוגמא תמונה של כלב) והרשת צריכה לזהות שיש שם כלב. הפלט של הרשת הוא מערך באורך מספר הקלאסיים, כאשר המספר בכל איבר במערך הוא הציון ("ההסתברות") שבתמונה מופיע אובייקט מאותה הקבוצה. אם לדוגמא הקבוצה הראשונה היא כלב והקבוצה השניה היא חתול. אז אם באיבר הראשון במערך יש 0.8 ובאיבר השני יש 0.1 אז הרשת די בטוחה שבתמונה יש כלב.

Detection:

בניגוד לרשת הקודמת, כאן יכולים להיות כמה אובייקטים בתמונה אחת, ומטרת הרשת היא לזהות איפה ומה האובייקטים. הפלט של הרשת הוא ריבוע וסיווג של האובייקטים המופיעים בתמונה.

Segmentation:

הפלט של הרשת דומה לפלט במקרה הקודם, רק שבמקום ריבוע יש זיהוי ברמת הפיקסל. איזה פיקסל שייך לאיזה אובייקט.

אל הקוד!

הגענו לחלק הפרקטי, קצת hands-on ☺

ניקח לדוגמא רשת שמקבלת תמונה קטנה שבתוכה יש ספרה. מספר הקלאסיים שלנו הוא 10 (כל המספרים מ-0 עד 9), ולכן הפלט הוא וקטור באורך 10. הקלט שלנו תהיה תמונה בגודל 28*28 (התמונה לא צבעונית, ולכן היא עם 2 מימדים, אין לה עומק).

קוד של הרשת + אימון

זה הקוד שבונה את הרשת, קודם כל, הרשת יורשת מאובייקט torch.nn.Module, מה שמאפשר עבודה נוחה עם רשתות (כמו חישוב הנגזרות וכאלה):

```
10  class Net(nn.Module):
11      def __init__(self):
12          super(Net, self).__init__()
13          self.conv1 = nn.Conv2d(1, 32, kernel_size=3)
14          self.conv2 = nn.Conv2d(32, 64, kernel_size=3)
15          self.fc1 = nn.Linear(5 * 5 * 64, 128)
16          self.fc2 = nn.Linear(128, 10)
17
18      def forward(self, x):
19          x = F.relu(self.conv1(x))
20          x = F.max_pool2d(x, 2)
21          x = F.relu(self.conv2(x))
22          x = F.max_pool2d(x, 2)
23          x = x.view(-1, 5 * 5 * 64)
24          x = F.relu(self.fc1(x))
25          x = self.fc2(x)
26          return x
```

בשורה 13-16 אנחנו מגדירים את השכבות, בשורה 19-26 זה ההרצה עצמה.

נסביר את השכבות ברשת:

בשכבה הראשונה יש 32 קונבולוציות, נסביר את הפרמטרים שהיא מקבלת (**שורה 13**) הפרמטר הראשון זה העומק של המטריצה שעליה הקונבולוציות רצות (במקרה שלנו זה 1 כי מדובר בתמונה לא צבעונית).

הפרמטר השני זה מספר הקרנלים שירוצו (בעצם זה מספר הקונבולוציות) וזה גם העומק של הפלט של השכבה (מכיוון שכל קונבולוציה מוציאה את הפלט שלה, העומק הוא מספר הקונבולוציות).

הפרמטר השלישי הוא גודל הקרנל (גודל החלון) שירוץ.

לאחר מכן (**שורה 14**) אנחנו מגדירים שכבת קונבולוציה שניה שמורכבת מ-64 קונבולוציות.

לאחר מכן (**שורות 15, 16**) אנחנו מגדירים 2 שורות של מכפלת מטריצות. הפרמטר הראשון זה מספר הפיקסלים שנכנסים למכפלה, והשני זה מספר הפרמטרים שיהיה אחרי המכפלה.

הפרמטר השני של מכפלת המטריצות השניה (**שורה 16**) הוא 10, זה אומר שהפלט יהיה וקטור בגודל 10 מכיוון שישנם 10 קלאסיים.

לגבי ההרצה (**שורות 26-18**):

בשורות 19, 21 קוראים לשכבות הקונבולוציה ביחד עם שכבת אקטיבציה (לא הסברתי על שכבת האקטיבציה במאמר).

אחרי כל שכבת קונבולוציה נעשה שכבת max pool (**שורות 20, 22**).

שורה 23 משנה את הצורה של הפלט לצורה שמתאימה למכפלת מטריצות, ואז שורות 23, 24 מבצעות את מכפלת המטריצות.

לגבי גדלי המטריצות בכל שכבה. כל שכבת קונבולוציה מורידה 2 פיקסלים (זה נובע מצורת העבודה של הקונבולוציה), כל שכבת max pool מורידה את הרזולוציה בחצי. ולכן אם בהתחלה גודל הקלט הוא 1*28*28, אחרי שכבת הקונבולוציה הראשונה הגודל הוא 32*26*26 (ירד 2 ברוחב ובגובה, ולכן נשאר 26, ויש 32 קונבולוציות, כל קונבולוציה מייצרת מטריצה ולכן העומק הוא 32).

אחרי ה-max pool הרזולוציה ירדה בחצי ולכן הגודל הוא 32*13*13

אחרי שכבת הקונבולוציה השניה, ירד עוד 2 ברזולוציה, והעומק הוא 64 (כי יש 64 קונבולוציות בשכבה הזו) ולכן הרזולוציה היא 64*11*11.

אחרי שכבת ה-max pool הרזולוציה ירדה בחצי ולכן הרזולוציה היא 64*5.5*5.5, אלא שלא יכול להיות חצי פיקסל, ולכן הרזולוציה היא 64*5*5.

ולכן **בשורה 23** שינינו את השכבה לוקטור באורך 5*5*64.

פונקציית האימון:

```
30  def train_epoch(model, train_loader, optimizer, criterion, epoch, losses):
31      model.train()
32      for batch_idx, (data, target) in enumerate(train_loader):
33          optimizer.zero_grad()
34          output = model(data)
35          loss = criterion(output, target)
36          loss.backward()
37          optimizer.step()
38
39          losses.append(loss.item())
40          percentage = 100. * batch_idx / len(train_loader)
41          print(f'\rEpoch: {epoch} | Progress: {percentage:.2f}% | Loss: {loss.item():.6f}', end='')
42
43      return losses
44
45
46  # Main training loop
    1 usage (1 dynamic)
47  def train(model, train_loader, optimizer, criterion, epochs=2, save_weights_path="./model_weights.pth"):
48      losses = []
49      for epoch in range(epochs):
50          losses = train_epoch(model, train_loader, optimizer, criterion, epoch, losses)
51      print()  # New line after training
52
53      # Save the model weights after training
54      torch.save(model.state_dict(), save_weights_path)
```

שורות 47-54 מגדירות את פונקציית האימון הכללית, נסביר את הפרמטרים:

- **model**: הרשת, היא אובייקט מהקלאס שהגדרנו מקודם
- **train_loader**: הינו אובייקט שמחזיק את הדאטא שלנו. יש שם בעצם מערך של תמונות ולכל תמונה את הקלאס האמיתי שלה (איזו ספרה יש בתמונה).
- **optimizer**: הפונקציה שמשנה את הפרמטרים של הרשת (היא הפונקציה שמחליטה כמה לשנות כל פרמטר).
- **criterion**: הפונקציה שמחשבת את ה-loss, שזה בעצם השגיאה של הרשת, כך אנחנו יודעים כמה צריך לתקן את הפרמטרים.
- **epochs**: אם יש לנו X תמונות בדאטא שלנו, אימון על כל הכמות של התמונות נקראת epoch, ואנחנו רוצים לאמן באורך של 2 אפוקים, זאת אומרת שכל תמונה בדאטא שלנו תיכנס לרשת פעמיים. באימון אמיתי זה יכול להיות עשרות/מאות/אלפי פעמים. תלוי בגודל הרשת.
- הפרמטר האחרון מגדיר איפה לשמור את המשקולות של הרשת (התוצאות של האימון) כדי שנוכל לטעון אותם בזמן מאוחר יותר ולהריץ את הרשת שלנו.
- יש לנו לולאה באורך מספר האפוקים שקוראת לפונקציה שמאמנת אפוק אחד (שורות 49-50).

לאחר שהאימון מסתיים, המשקולות נשמרות (**שורה 54**).

שורות 30-43 מגדירות את פונקציית האימון עצמה.

שורה 31 מעבירה את הרשת למצב אימון.

שורה 32 מגדירה לולאה על כל התמונות ברשת, data זה מערך של התמונות. ו-targets זה מערך של התוצאות (האיבר ה-i במערך target הוא המספר שמופיע בתמונה שנמצאת באיבר ה-i במערך data).

שורה 33 מאפסת את הגרדיאנטים (הנגזרות) שחישבנו לרשת כדי שתוצאות החישוב יהיו תלויות רק בהרצה הנוכחית ללא תלות בהרצות הקודמות.

שורה 34 מריצה את התמונות במודל ושומרת את התוצאות למשתנה output. השורה הזו בעצם קוראת לפונקציה forward באובייקט של המודל. הפונקציה מוגדרת **בשורות 18-26**.

שורה 35 מחשבת את השגיאה של הרשת. התוצאות האמיתיות של התמונה שמורות במשתנה target והניבוי של הרשת במשתנה output. הפונקציה מחשבת את הטעות שיש ביניהם.

שורה 36-37 מחשבת כמה צריך לשנות כל פרמטר ברשת, ומשנה אותם בפועל (פה מתבצע הלימוד של הרשת).

שורות 39-41 מחשבות את הלוס הכללי ומדפיסים אותו ללוג.

```
30 def train_epoch(model, train_loader, optimizer, criterion, epoch, losses):
31     model.train()
32     for batch_idx, (data, target) in enumerate(train_loader):
33         optimizer.zero_grad()
34         output = model(data)
35         loss = criterion(output, target)
36         loss.backward()
37         optimizer.step()
38
39         losses.append(loss.item())
40         percentage = 100. * batch_idx / len(train_loader)
41         print(f'\rEpoch: {epoch} | Progress: {percentage:.2f}% | Loss: {loss.item():.6f}', end='')
42
43     return losses
44
45
46 # Main training loop
   1 usage (1 dynamic)
47 def train(model, train_loader, optimizer, criterion, epochs=2, save_weights_path="./model_weights.pth"):
48     losses = []
49     for epoch in range(epochs):
50         losses = train_epoch(model, train_loader, optimizer, criterion, epoch, losses)
51     print()  # New line after training
52
53     # Save the model weights after training
54     torch.save(model.state_dict(), save_weights_path)
```

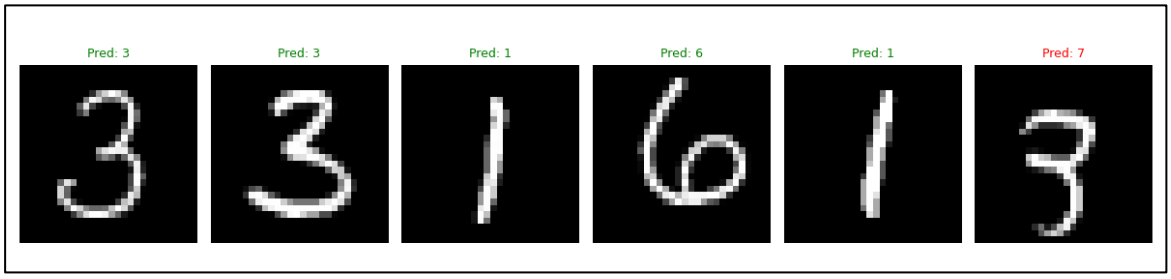
פונקציית ה-val בודקת את תוצאות הרשת. מקבלת כפרמטרים את הרשת המאומנת, ואוסף של תמונות שמשמשות לבדיקת תוצאות הרשת (תמונות שהרשת לא ראתה אותם בתהליך האימון).

שורה 57 מעבירה את הרשת למצב "הרצה" (ולא אימון), זה אומר לרשת האם לחשב את השגיאה במהלך ההרצה. במקרה של אימון נרצה לחשב את השגיאה, ובמקרה של הרצה לא נרצה.

שורות 58-36 זה לולאה עבור כל התמונות וספירה כמה מהתוצאות היו נכונות.

שורות 65-66 מחשבות את אחוזי ההצלחה ומדפיסות.

שורות 59-78 מציגות חלק מדגמי מהתוצאות בתמונה כזו:



מוצג פה הניבוי של הרשת (pred) ומתחתיו התמונה עצמה. ניתן לראות שבריבוע הכי ימני יש את הספרה 3, אולם הרשת חשבה שיש שם את הספרה 7 (חיזוי לא נכון מוצג באדום, חיזוי נכון מוצג בירוק).

```
81  batch_size = 64
82  lr = 0.01
83  momentum = 0.9
84
85  # Data preparation
86  transform = transforms.Compose([transforms.ToTensor(), transforms.Normalize( mean: (0.5,),  std: (0.5,))])
87  train_loader = torch.utils.data.DataLoader(datasets.MNIST( root: './data', train=True, download=True, transform=transform)
88                                             batch_size=batch_size, shuffle=True)
89  test_loader = torch.utils.data.DataLoader(datasets.MNIST( root: './data', train=False, transform=transform),
90                                            batch_size=batch_size, shuffle=True)
91
92  # Model, optimizer, and loss function setup
93  model = Net()
94  optimizer = optim.SGD(model.parameters(), lr=lr, momentum=momentum)
95  criterion = nn.CrossEntropyLoss()
96
97  # Training
98  train(model, train_loader, optimizer, criterion)
99
100 model_weights_path = "./model_weights.pth"
101 model.load_state_dict(torch.load(model_weights_path))
102 # Validation
103 val(model, test_loader)
```

שורות 81-83 כוללות את הגדרות לאימון הרשת

שורה 81: כמות התמונות שמכניסים לרשת בכל הרצה.

שורות 82-83: קצב הלמידה של הרשת.

שורה 86: פונקציות שמריצים על התמונה לפני ההכנסה לרשת, יש שם נירמול והעברה לאובייקט בשם טנזור, כמו שאמרנו, טנזור הוא מטריצה עם הרבה מימדים.

שורות 87-89: טוענים את התמונות של האימון והבדיקה.

בשורה 93 מגדירים אובייקט של הרשת שלנו.

שורות 94-95: הפונקציות שמאמנות את הרשת (מעבירים אותם כפרמטר לפונקציה train שראינו לפני כן).

שורה 98: קריאה לפונקציית האימון (במקרה ורוצים רק להריץ ללא אימון, אפשר לשים את השורה הזו בהערה).

שורות 100-101: טעינת המשקולות המאומנות שנשמרו בשלב האימון.

שורה 103: הרצת הרשת לצורך בדיקה.

סיכום

במאמר הסברנו מה היא בינה מלאכותית ואת תהליך האימון של רשתות נוירונים. הבאנו כדוגמא "תיבת גלטון" למיון כדורים לפי משקלם לצורך המחשת הרעיון של קלט ופלט במודלי בינה מלאכותית וכן להסברה איך רשת נוירונית לומדת ומתאמת את פרמטריה לפי שגיאות קודמות. בנוסף, הסברנו על מושגים בסיסיים באלגברה לינארית וחישוב נגזרות, שהם הבסיס להבנת האופן שבו רשתות נוירונים עובדות. כמו כן, תיארנו את שלושת סוגי הרשתות המרכזיות בעיבוד תמונה: סיווג, זיהוי וסגמנטציה.

בחלק הפרקטי של המאמר, בנינו רשת נוירונים לזיהוי מספרים בתמונות.

על המחבר

שלום דימנט הוא בעל תואר ראשון בפיזיקה ותואר שני בהנדסת תוכנה וכיום ראש צוות בינה מלאכותית בחברת WiseSight טכנולוגיות. חברת WiseSight מספקת פתרונות IoT חדשניים ופורצי דרך לערים חכמות. מוצר הדגל של החברה בישראל הינו מערכת בינה מלאכותית חדשנית לביצוע פיקוח אכיפה ותשלום אוטומטים בחניות רחוב (כחול לבן) ובחניונים.

קישור לקוד שהופיע במאמר ועוד, אפשר למצוא ב-GitHub:

https://github.com/sdimantsd/mnist_example