

הפקולטה להנדסה

המעבדה לננו - אלקטרוניקה

Neural Networks based on Ternary Design

נדב חצור

טום אשקורי

פרויקט שנה ד' לקראת תואר ראשון בהנדסה

מנחה: מר רומן גולמן

מנחה אקדמי : פרופ' אדי תימן

נובמבר 2022

תוכן עניינים

4 ----------------------------------------------------------------------------------------**הבעיה שאנו פותרים**

5 ----------------------------------------------------------------------------------------**מבוא ללמידה עמוקה**

5 ---------------------------------------------------------------- **רקע על המבנה של רשת נוירונים עמוקה**

7 ----------------------------------------------------------------------------------------שיטות תכנון נוירונים

8 ----------------------------------------------------------------------- אלגוריתם העברת החישובים ברשת

9 --------------------------------------------------------------------------**פונקציות אקטיבציה נפוצות**

11 ----------------------------------------------------------------------**אלגוריתמים לצורכי אימון הרשת**

11 ---------------------------------------------------------------------- Back Propagation

13 ---------------------------------------------------------------------- Adam Optimizer

15---------------------------------------------------------------------------------------**סוגי רשתות נוירונים**

16---------------------------------------------------------------------------------------רשת בינארית

17---------------------------------------------------------------------------------------רשת טרנרית

18 ----------------------------------------------------------------------------------------**זכרונות**

19----------------------------------------------------------------------------------------**Gain Cell**

21 ---------------------------------------------------------------------------------------- **הרשתות**

21---------------------------------------------------------------------------------------רשת בינארית

27---------------------------------------------------------------------------------------רשת טרנרית

36---------------------------------------------------------------------------------------מימוש ב-C

42 ---------------------------------------------------------------------------------------- **תוצאות**

5 4----------------------------------------------------------------------------------------**מסקנות**

הבעיה שאותה אנו פותרים

רשתות נוירונים עמוקות השתכללו במהלך השנים האחרונות והשתלבו בתחומים שונים כמו ראיה ממוחשבת ובפרט בביצוע משימות כמו גילוי,ניתוח וזיהוי אובייקטים.

כיום כאשר חברות ענק בתעשייה מייצרות מכשירים אלקטרונים כגון טלפונים חכמים וכל ציוד ממוחשב שהוא, עולות מגבלות של כוח חישוב ,זיכרון ושטח ולכן קיים הצורך לפתח שיטות חדשות ולייעל את השיטות הקיימות למימוש רשתות נוירונים עמוקות בחומרה.

היום כאשר ניגשים לפתח ישנם כמה רשתות נפוצות שבהם משתמשים, נציג וננתח בקצרה את היעילות והדיוק של כמה מהן :

רשת DNN(Deep neural Network) שבה טווח המשקולות הינו רחב ולכן החישובים מורכבים יותר ונדרש זיכרון רב לשם כך.

רשת BNN(Binary neural Network) שבה המשקולות הינן '1-' ו- '1' ולכן החישובים פשוטים יותר מאשר רשת הDNN ונדרש גודל קטן יותר של זיכרון.

רשת TNN(Ternary neural Network) שבה המשקולות הינן '1-' , '1' ו- '0' ובזכות זאת החישובים פחות מורכבים מאשר ה-DNN ועם זאת נדרש מעט יותר זיכרון מאשר ה-BNN.

היעילות והדיוק הגבוה של שיטות אלו נותנות מוטיבציה לפתח את התחום ולייעל אותו אף יותר הן בחומרה והן בדיוק.

על מנת לשפר ולייעל במידה ניכרת את מערכת הלמידה העמוקה אנו ננתח 2 שיטות שונות למימוש רשת נוירונים.

מבוא ללמידה עמוקה

את הפרויקט נממש באמצעות עולם הלמידה העמוקה, לכן נציג את הרעיונות, האלגוריתמים ומושגי היסוד הרלוונטיים לפרויקט זה.

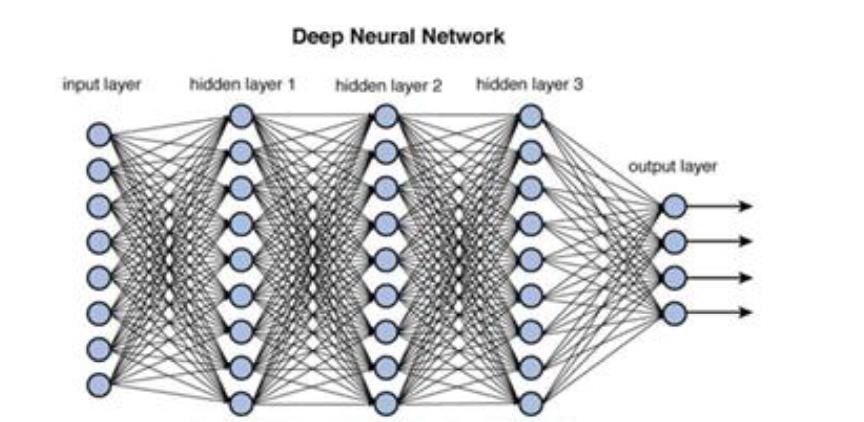
רשת נוירונים עמוקה

רשת נוירונים הינה מערכת למידה מפוקחת הבנויה ממספר גדול של אלמנטים פשוטים, המכונים נוירונים כאשר כל נוירון מסוגל לבצע בחירה של החלטה פשוטה ואת ההחלטה הזו הוא מעביר קדימה לנוירונים הבאים, כאשר הנוירונים מסודרים בצורה של שכבות אשר מחוברות זו לזו.

באמצעות הרשת, יהיה ניתן להביע כמעט כל פונקציה, ולכן ניתן לענות על משימות כמו מיון, סיווג וחישובים מסוגים שונים.

על מנת למטב את ביצועי הרשת נדאג לספק סט אימון רחב הכולל דוגמאות שונות של אותה בעיה אשר אנו מנסים לפתור.

המחשה של רשת נוירונים עמוקה המורכבת משכבות של נוירונים, כניסה ויציאה ושכבות נסתרות ונוירונים:



רקע על המבנה של רשת נוירונים עמוקה

השכבה השמאלית (input layer) - מתארת את שכבת הקלט.

השכבות האמצעיות(hidden layers) - אלו הן שכבות הביניים, כאשר מספר שכבות אלו איננו מוגבל.

בשכבות אלו מתבצע עיבוד ביניים לטובת חישוב השערוך, כל שכבה מכילה משקולות אשר מחושבות באמצעות גרדיאנטים, הנוירונים עוברים דרך פונקציית אקטיבציה לפני המעבר לשכבה הבאה .

השכבה הימנית (output layer) - שכבת המוצא, אשר מקבלת את המידע שהועבר אליה דרך השכבות הנסתרות ומציגה חיזוי סופי.

מבנה הרשת בפרויקט יכיל שכבת כניסה, מוצא ומספר משתנה של שכבות נסתרות, כל גודל של שכבה נסתרת ישתנה אף הוא. שכבת הכניסה בגודל של (784,1) כלומר מבנה שמדמה מערך עם ממד אחד, כיוון שהDataset שאיתו עבדנו הוא MNIST – מאגר תמונות בגודל 32\*32 של מספרים משורבטים. כל ערך פיקסל נע בין 0 ל – 255 (נקרא גם grayscale), לצורך נוחות חתכנו את התמונות לגודל של 28\*28 מה שמניב את גודל שכבת הכניסה.

שכבת המוצא בגודל 10 משום שאנו מלמדים את המודל להבחין בין 10 מספרים שונים (0-9).

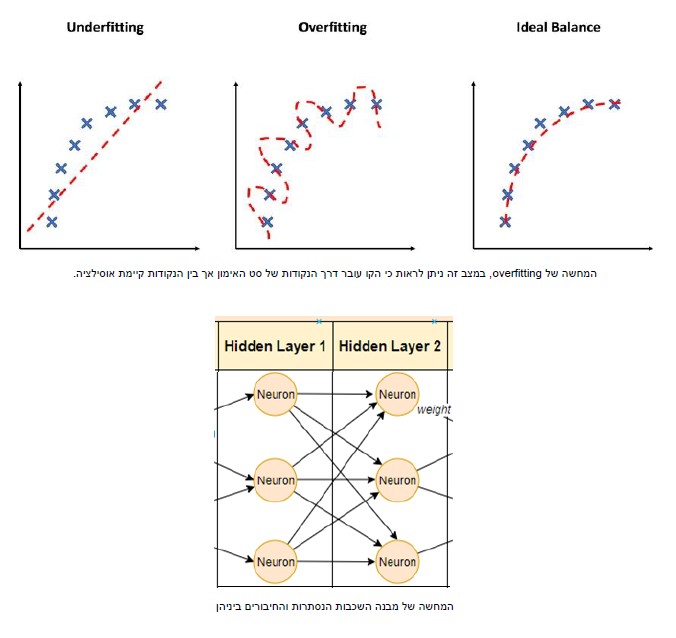
שכבות הביניים בדרך כלל יגיעו בחזקות של 2 עם כמות נוירונים הולכת וגדלה.

חשוב לציין שקביעת פרמטרי השכבות הנסתרות יכול להיות שרירותי ובלי חוקיות ובדרך כלל מנסים כמה גישות שונות ובוחרים בזאת שמניבה דיוק מירבי.

שיטות תכנון רשת נוירונים

הבעיה המרכזית ביותר בתכנון מערכות אלו היא בחירה של ההיפר פרמטרים, אלו הפרמטרים שאנו קובעים אשר עבורם הרשת תיתן את הביצועים הטובים ביותר.

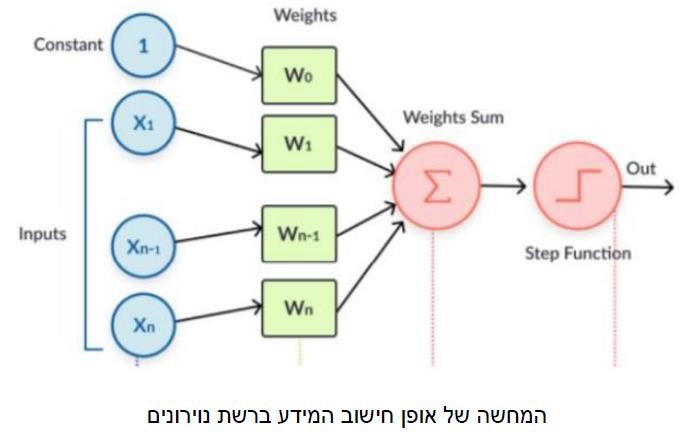
על מנת לבחור את ההיפר-פרמטרים, משתמשים בשיטה של קרוס ולידציה, כאשר מחלקים את הסט של האימון לסט של אימון וגם סט של קרוס-ולידציה (בדרך כלל נבחר את האחוז הדומיננטי של הדאטה סט לטובת האימון), את סט האימון נאמן דרך חישוב הגרדיאנטים ואת סט הקרוס ולידציה נעביר דרך הרשת ללא חישוב גרדיאנטים אך נשתמש בחישוב של השגיאה וכך נדע האם המערכת מצליחה לשערך מידע שהוא זר מהסט אימון, במילים אחרות אנחנו פותרים כך את בעיית ה overfitting שעלולה להגרם לרשת, במצב של overfitting אנחנו נראה ביצועים טובים של הרשת כלפי סט האימון אך לא כלפי מידע זר.



אלגוריתם העברת החישובים ברשת(Forward Passing):

נתאר את התהליך שהמידע עובר בתור הרשת ודרך הנוירונים.

1. הכניסות המוזנות לרשת מוכפלות במשקולות.
2. סכימה של הכניסות המוכפלות במשקולות.
3. הוספה של קבוע bias על מנת לבצע הזזה לכיוון השערוך.
4. העברה של הסכום דרך פונקציית אקטיבציה.
5. העברת התוצאה הכוללת ככניסה של הנוירון הבא ברשת.



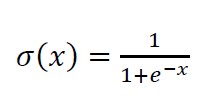
שכבות הביניים מורכבות משרשור של שכבות נוירונים וביניהן פונקציות שיכולות להיות לא ליניאריות בשם פונקציות אקטיבציה.

הנוירון בכל שכבה, הינו צירוף ליניארי של כל הערכי ם של הנוירונים בשכבה הקודמת ועליו מופעלת פונקציית האקטיבציה.

מטרת פונקציות האקטיבציה הינה מיפוי הסכום המשוקלל שאותו מקבל נוירון לערך בין 0 ל1 כמובן באפליקציות מורכבות יותר ניתן להישתמש בפונקציות אקטיבציה לא לינאריות וכך תחום הפונקציות אותן הרשת יכולה להביע גדל.

פונקציות אקטיבציה נפוצות:

שכבות הביניים מורכבות משרשור של שכבות נוירונים וביניהן פונקציות לא ליניאריות בשם פונקציות אקטיבציה. הנוירון בכל שכבה, הינו צירוף ליניארי של כל הנוירונים בשכבה הקודמת ועליו מופעלת פונקציית האקטיבציה. מטרת פונקציות האקטיבציה הינה הגדלת תחום הפונקציות אותן הרשת יכולה להביע. ללא פונקציות האקטיבציה, הרשת הייתה יכולה להביע רק פונקציות ליניאריות. פונקציות אקטיבציה עיקריות שבהן נשתמש בפרוייקט זה:



פונקציית סיגמונד:

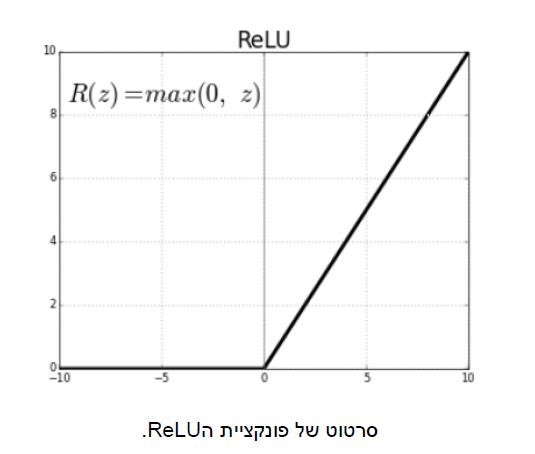
פונקציה זו מקבלת כך ערך ומנרמלת אותו לטווח מספרים בין 0 ל 1.

פונקציה זו איננה נמצאת בדרך כלל בשכבות הביניים, השימוש הנפוץ של פונקציה זו הוא בבעיות מסוג סיווג, לדוגמא, במערכת שתפקידה להחליט האם קלט של תמונה כלשהי מכיל חתול, סיגמוייד יציג לנו שערוך סופ י בטווח של 0 עד 1 וכך נדע את הסיכוי של המצאות החתול בתוך התמונה )למשל, עבור ערך יותר קרוב ל- 1, כנראה שמודבר בחתול.(

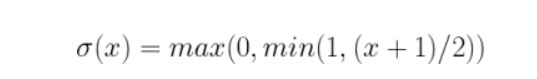
פונקציית ReLU

הפונקציה מאפסת כל ערך שלילי שהיא מקבלת ומעבירה הלאה כל ערך חיובי שהיא מקבלת .

לפונקציה זו שימוש נפוץ בעולם של רשתות נוירונים מאחר והיא זולה לשימוש כתוצאה מכך שישנם טווח ערכים שהיא מאפסת, ולכן חלק מהנוירונים לא יעבדו בו זמנית וכך נחסוך זמני חישוב.

מ2017 זוהי הפונקציה שזוכה לשימוש הנפוץ ביותר בקרב אלגוריתמים של ראייה ממוחשבת ועוד.

כיוון שנעסוק ברשתות בינאריות פונקציית הReLU תיראה למשל כך:

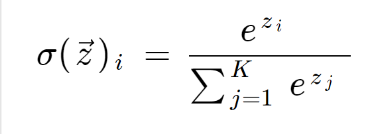


פונקצית הsoftmax

פונקציה שמטרתה להניב ערך הסתברותי עבור אותו קלאס ,כלומר עבור אינפוט מסויים נקבל בסופו של

התהליך ווקטור הסתברויות לכל קלאס (שסכומו 1 כמובן). ניתן להפוך פונקציית סיווג רכה (soft) לסיווג

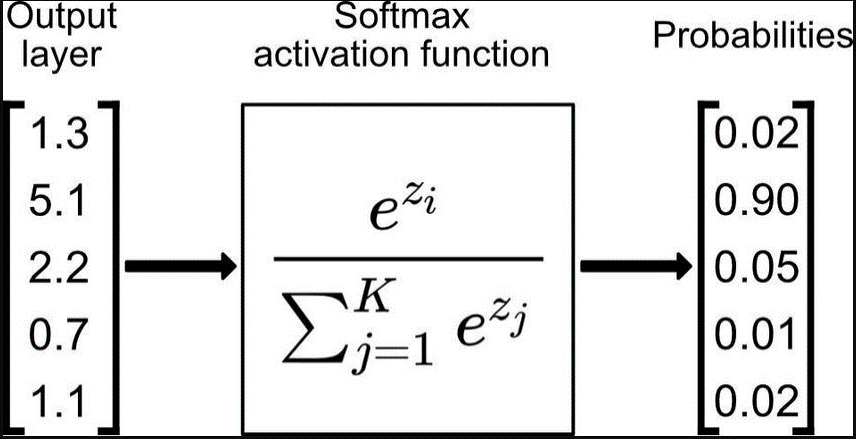
"קשה" באמצעות הסתכלות על הערך ההסתברותי המקסימלי באותו ווקטור הסתברויות וכך נוכל להגיע

להחלטה.

כאשר מייצג את וקטור הכניסה, מייצג את הפרמטרים (המשקולות וערך ה-bias) שמשויכים לקלאס i

הסכום התחתון מייצג את אותו חישוב עם אותו וקטור z רק עבור כל הפרמטרים של כל קלאס

וכך נקבל וקטור הסתברות.

**

Diagram

Description automatically generated

אלגוריתמים לצורכי אימון הרשת

אלגוריתם למידה והעברה של מידע אחורה(Back Propagation):

מאחר ואנחנו מאתחלים את רשת הנוירונים עם משקולות התחלתיות שניתן להגיד שהן לא אופטימליות, החיזוי הראשוני שמתקבל בתהליך ה Forward Passing)שכלול של הכניסות עם המשקולות( הוא לא החיזוי הטוב ביותר.

שימוש נפוץ למזעור השגיאה ברשת נוירונים הוא בעזרת Backpropagation .

המושג backpropagation כולל בתוכו משפחה של אלגוריתמים, שהרעיון המרכזי שלהם הוא למזער את השגיאה ולהביא את המשקולות של שכבות הביניים למצב אופטימלי בצורה מהירה יחסית, אפילו עבור רשת עם כמות גדולה של נוירונים.

ראשית, נתאר כמה מושגי יסוד עבור האלגוריתם:

Batch

מספר קבוע שמייצג קבוצה של דגימות מתוך סט המידע, בדרך כלל חזקה של 2 ,את סט האימון

נכניס לתוך הרשת בתוך Batch של דגימות, כאשר גודל הBatch נקבע שרירותית (אפשר לבצע שינוי אוטומטי של הערך הזה אבל לשם פשטות נמנענו מזאת).

Gradient Descent and Backpropagation

זוהי שיטת אופטימיזציה איטרטיבית מסדר ראשון למציאת מינימום מקומי של פונקציה.

בשיטה זו, נעשה צעד נגדי לגרדיאנט ביחס לנקודה הנוכחית.

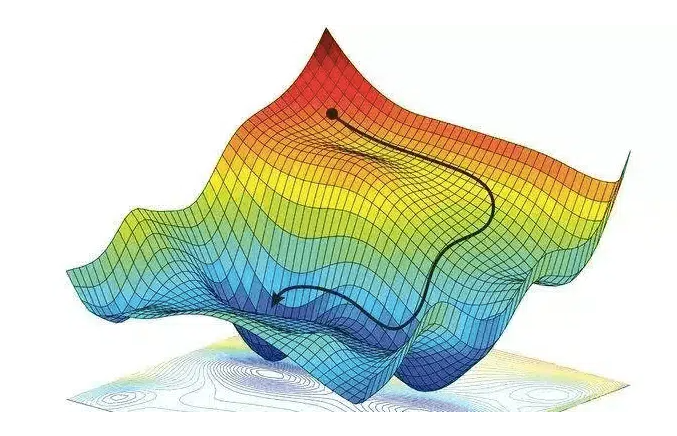
תהליך צמצום השגיאה פועל בצורה הבאה על כל Batch בנפרד:

- Feedforwarding - מזין את הdata training ולאחר מכן מקבלים חיזוי.

- Backpropagation - האלגוריתם מחשב את הנגזרות החלקיות בעזרת כלל השרשרת,

מפונקציית השגיאה לנוירון מסוים וכך בודק את גודל ההשפעה של המשקולות על החיזוי.

Weight Update - – מבצעים Backpropagation באמצעות Gradient Descent



נוסחאת עדכון המשקולות:

כאשר מייצג משקולת והחישוב מתבצע בעזרת הגרדיאנט.

את הרשת נרצה לאמן בצורה איטרטיבית, עבור סט רחב של דוגמאות, על פי הדרך שהצגנו.

כעת נציג מושגים שבאמצעות נממש את האימון של הרשת על גבי סט הדוגמאות:

Epoch

מתאר איטרציה על כל סט הדוגמאות שלנו, Size Epoch יהיה מספר האיטרציות שנקדיש לטובת שלב האימון של הרשת, זהו היפר-פרמטר נוסף אותו נקבע באופן שרירותי שיניב את הדיוק הרב ביותר.

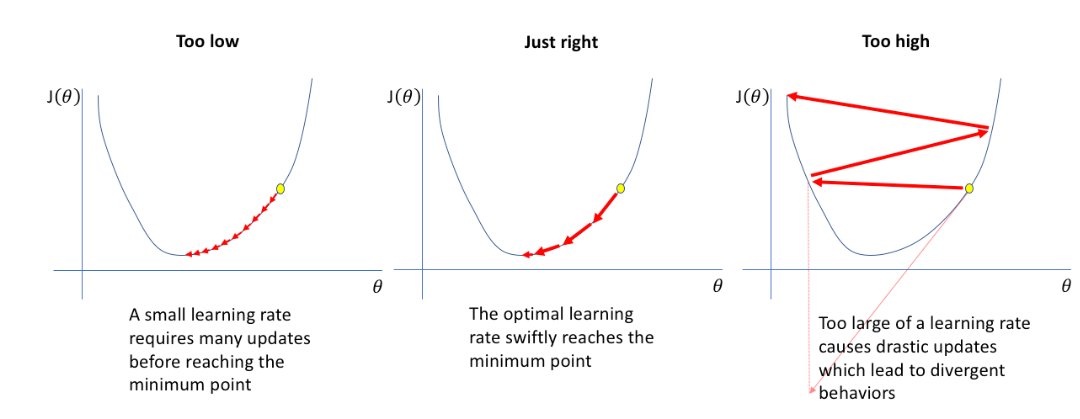
Learning Rate

קצב הלמידה הינו היפר-פרמטר גם כן, הוא קובע את קצב ההתקדמות של הגרדיאנט בכיוון

המינימום, ניתן להבין שקצב גבוה מדי עלול לגרום לקפיצות מעל המינימום ולחוסר התכנסות לתוכו,

וקפיצות קטנות מדי עלול לגרום לנו להיתקע על מינימום מקומי ולא על מינימום גלובלי.

ישנם אלגוריתמים בשם Scheduling אשך מאפשרים שינוי קצב הלמידה כתלות בהקדמות אימון הרשת, אך אצלנו בחרנו בגישה של להתחיל עם learning rate מסוים ולהקטין אותו עם כל epoch ב90 אחוז.



Adam Optimizer

בפרוייקט זה נשתמש באופטימייזר מסוג Adam לצורך אימון הרשת.

קיימים מגוון של אופטימייזרים, אך עבור פרוייקט זה נעדיף לעבור עם Adam מכיוון שהוא משלב בתוכו יתרונות של שיטות אחרות ,Adam משתמש בממוצע של המומנט השני (שונות) בשונה מאופטימייזר הRMSProp למשל, המתאים לעצמו את קצב למידת הפרמטרים על סמך ממוצע אמפליטודות הגראדינט שזהו המומנט הראשון (ממוצע).

האלגוריתם עובד בצורה הבאה:

א - moving average על הגרדיאנט והגר*דיאנט בריבוע*

ב- משערכים מהמומנט הראשון והשני לאחר נרמול

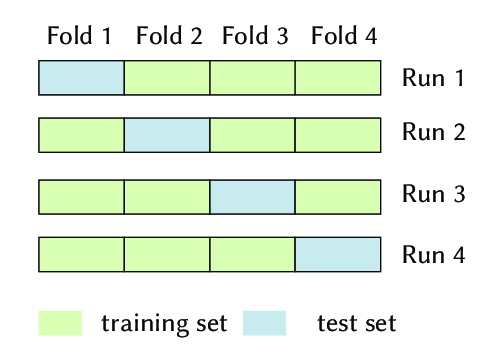
ג – עדכון המשקולות

יתרון נוסף של אלגוריתם Adam הוא שגודל צעד העדכון אינו משתנה כתלות במגניטודה ולכן יכול

לעזור במצבים של Saddle Point.

KFold

לצורך למידה יעילה יותר והכללה של הדפוסים המצויים בתוך סט המידע מחלקים את ה-data ל"פולדים" הכוונה לקבוצות זרות, בצורה כזאת שבכל איטרצית למידה אנו לוקחים את כל הפולדים פרט לאחד שבו משתמשים כסט ואלידציה. כך נוכל בכל ריצה לשנות במעט את סט הלמידה מה שיניב הכללה טובה יותר של המודל אל המידע.



עוד מושגי בסיס מעולם רשתות הנוירונים :

סט אימון

מטרת סט זה הינה עדכון משקלי רשת הנוירונים באמצעות descent Gradient.

סט ולידציה (קרוס- ולידציה)

סט זה מוגדר לצורך הערכה ודיוק המערכת.

ההערכה תתבצע לאחר כל איטרציה (epoch), כאשר בכל אחת מועבר מידע בגודל size-batch

(מוגדר מראש ככמות מידע המועברת בכל איטרציה).

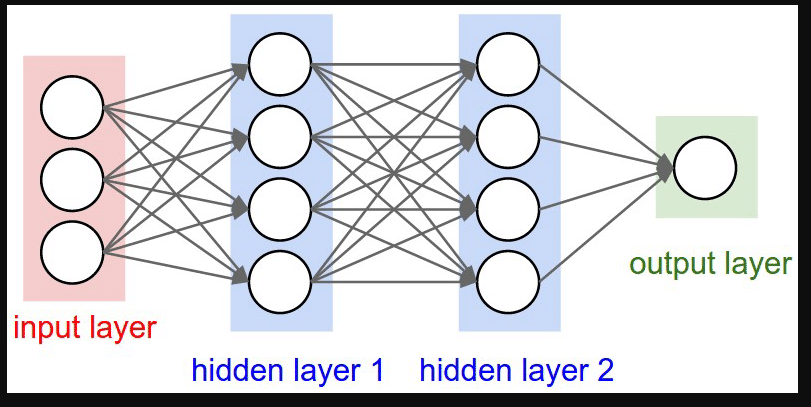
סט מבחן

סט זה יבחן את ביצועי המערכת עבור מידע חדש שלא אומנה לפיו.

סוגי רשתות נוירונים

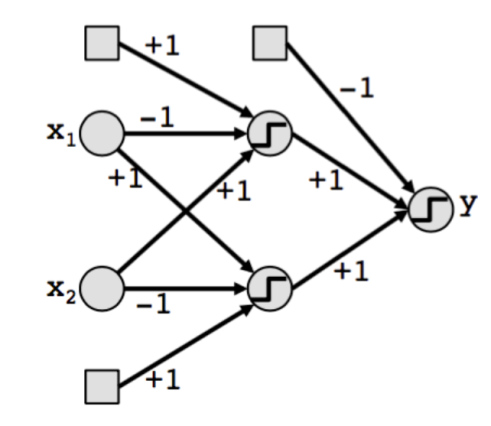
רשת נוירונים עמוקה(Deep Neural Network) –

רשת עצבית עמוקה (DNN) היא רשת עצבית מלאכותית (ANN) עם שכבות מרובות בין שכבות הקלט והפלט. ישנם סוגים שונים של רשתות עצביות אך הן תמיד מורכבות מאותם מרכיבים: נוירונים, משקלים, הטיות ותפקודים. רכיבים אלו פועלים באופן יחסית דומה למוח האנושי וניתן לאמן אותם כמו כל אלגוריתם ML אחר.



רשת נוירונים בינארית:

רשת נוירונים בינארית היא רשת נוירונים זהה במבנה למה שהוצג עד כה פרט לכך שהמשקלים שמשויכים לכלל הנוירונים הינם 1 או 1- שנשמר בזכרון כ- '1' או' 0' בהתאמה, דבר אשר מאיץ משמעותית את כלל החישובים במערכת על מנת להגיע להחלטה.



מימוש בחומרה:

בסופו של דבר אנו מנסים לחשב את החישוב הבא:

𝑎𝑖𝑙 = 𝜎(𝑎1𝑙−1𝑤1 + 𝑎2𝑙−2𝑤2 + ⋯ 𝑎𝑛𝑙−−𝑛1𝑤𝑛−1 + 𝑏)

כאשר 𝑤𝑖 הינו המשקולת, 𝑎𝑖 הינו הנוירון במיקום ה-b ,i הוא ערך מספרי משתנה בהתאם לנוירון וקובע את רמת הפעילות.

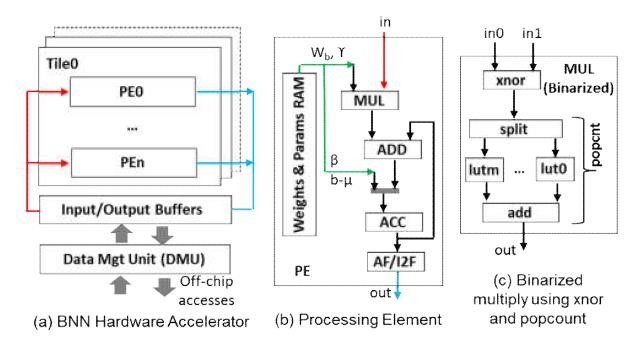
לכן בסך הכול זה הינו כפל המטריצות הבא:

𝑤0,0 ⋯ 𝑤0, 𝑛 𝑎00 𝑏0

[ ⋮ ⋱ ⋮ ] ∙ [ ⋮ ] + [ ⋮ ]

𝑤𝑘,0 ⋯ 𝑤𝑘,𝑛 𝑎𝑛0 𝑏𝑛

לכן מבחינת חומרה נצטרך full adder ומכפל. לכן דוגמא למימוש חומרתי יהיה:



רשת נוירונים טרנרית:

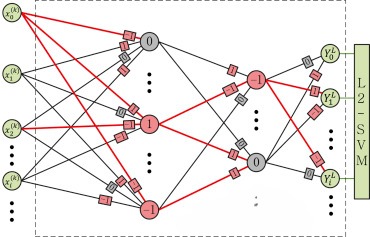
רשת נוירונים טרנרית היא רשת נוירונים זהה במבנה למה שהוצג עד כה פרט לכך שהמשקלים שמשויכים לכלל הנוירונים הינם 1 , 0 או 1- שנשמר בזכרון מיוחד שנקרא Gain Cell כ -

הייתרון בכך הוא שאנו מרוויחים דיוק גדול יותר בעזרת הוספת המשקולת אבל החישובים נישארים עדיין יחסית פשוטים למשל:

בעוד שבDNN משתמשים בALU כדי לבצע כפל ב-BNN אפשר פשוט להשתמש בשער XOR כדי לממש כפל כך שסיבוכיות הפעולה קטנה יותר.

כך גם ביצוע פעולות מתמטיות ברשת טרנרית לא דורשות כוח חישובי גדול.

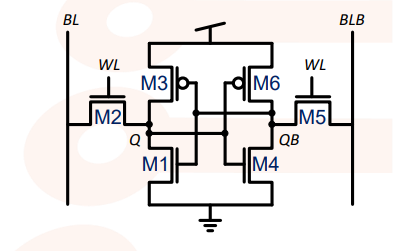
`



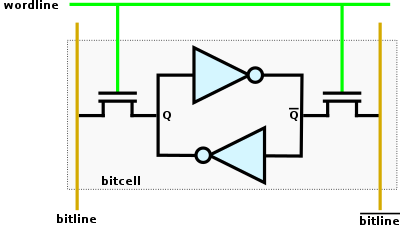
זכרונות

כל רכיב אלקטרוני מורכב מצריך שימוש בזכרונות, למשל לצורך שמירת מצב נוכחי או כאמצעי אחסון של כתובות/ערכים ובמקרה שלנו ערכי המשקולות.

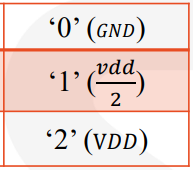
כיום השיטה הנפוצה ביותר היא שימוש ברכיב SRAM T6:



שני מהפכים שמחוברים בטור ועוד שני טרנזיסטורים שקובעים מצב:



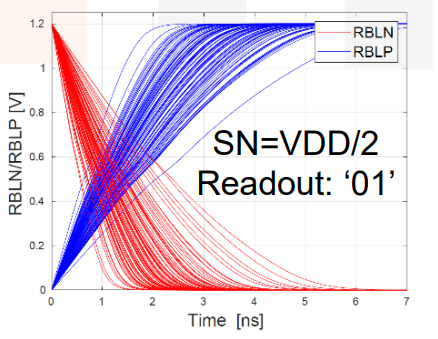
Gain Cell

בGain Cell עבור רשת טרנרית אנו משתמשים בחמישה טרנזיסטורים שעוזרים לנו לייצג 3 ערכים לוגים שונים.

ניתן לראות את פעולת המערכת עבור שלושה ערכים לוגים שונים:

Chart, line chart

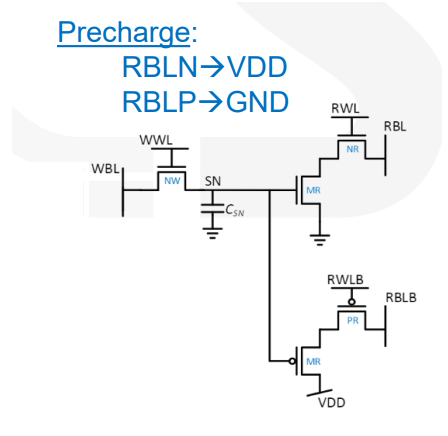
Description automatically generated



A picture containing graphical user interface

Description automatically generated

ניתן לראות את מימוש המערכת בתרשים הבא:



הרשתות

לצורך ביצוע הפרוייקט מימשנו רשת בינרית ורשת טרנרית כדהלן:

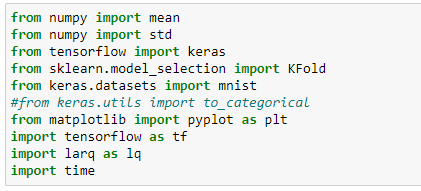
רשת בינרית:

את הרשת מימשנו באמצעות jupyter notebook בpython כלי נוח שמאפשר הרצה אונליין של הקוד וחילוקו לבלוקים.

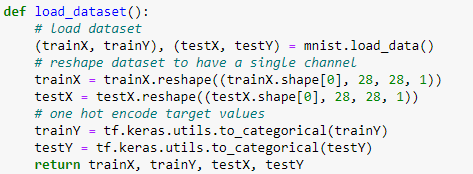
בנוסף את הרשת מימשנו באמצעות חבילת LARQ שמותאמת לרשתות נוירונים בינאריות והכלים הכלולים בתוכה עברו אופטימיזציה כבר.

חבילת LARQ מסתמכת על חבילה אחרת ידועה בתחום שנקראת tensorflow ובפרט על API שיושב על tensorflow שנקרא keras שמקבל מאוד את השימוש.

בנוסף השתמשנו בnumpy חבילת שמכילה מבני נתונים כמו מערכים שtensforflow והמון חבילות אחרות יודעות לעבוד איתן.

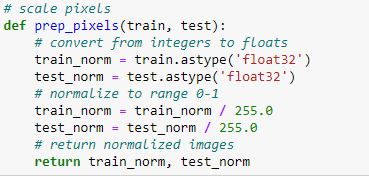


בנוסף הגדרנו פונקציות שונות לצורך הקלת השימוש והאימון ברשת.



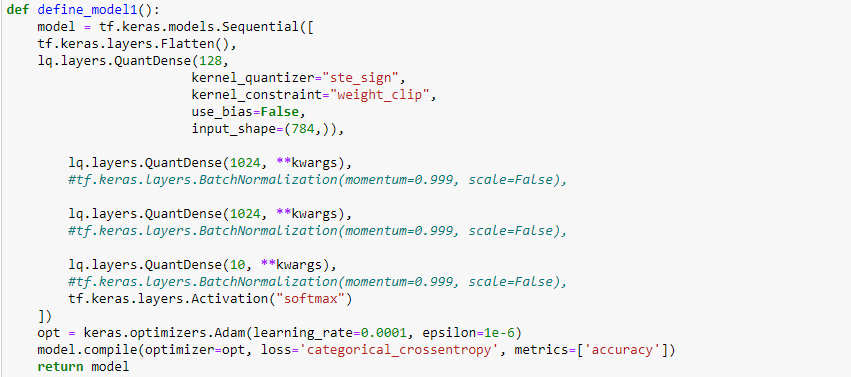
הפונקציה הנ"ל טוענת את סט המידע של mnist, ממירה את התמונות למימדים של 28\*28 פיקסלים ובנוסף מייצר לייבלים באמצעות קידוד “one hot” עבור על תמונה עם ערך הקלאס שמתאים לה למשל אם הקלאס הוא 0 (משוייך לתמונה של המספר 0) אז הלייבל המתאים לה יהיה: .

פונקציה נוספת:



לצורך עבודה יעילה יותר נרמלנו את ערכי הפיקסלים ע"י חילוק כל אחד מהם בערך 255 כך נקבל במקום סקלה של 0-255 אלא סקלה של 0-1.

הרשת עצמה:

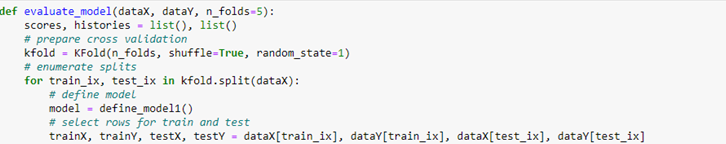


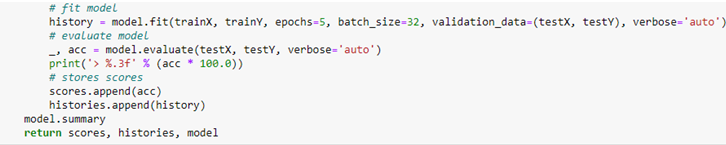
הגדרנו מודל סדרתי שמייצג מבנה שגרתי של רשת נוירונים אך לאחר מכן השתמשנו בLARQ כדי להגדיר שכבות Dense בינאריות ולבסוף שכבת פלט בעלת 10 נוירונים אשר כל אחד מהם מייצג קלאס אחר (0-9).

שכבת הקלט מקבלת תמונה בתצורה של מערך מגודל 784 כפי שניתן לראות בתמונה אותם נוירונים מחוברים ל128 אחרים.

כפי שכבר שהצגנו מספר הנוירונים, מספר השכבות, האופן שבוא המידע עובר (forward propagation) ועוד יכולים להשתנות במהלך הפרויקט.

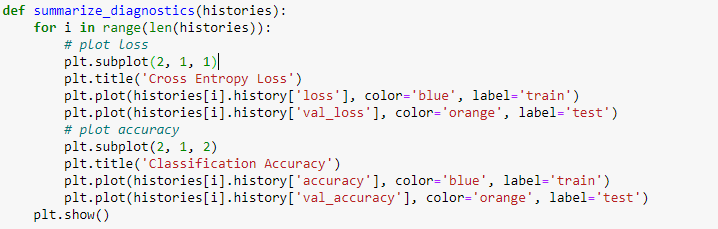
בנוסף הגדרנו אופטימייזר אדם אשר מבצע את הערכת המשקולות.





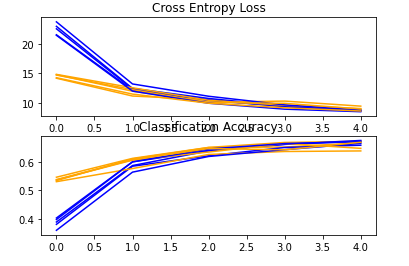
זוהי הפונקציה שמבצעת את הריצה עצמה ובה קורה הערכת המשקולות ומופקות סטטיסטיקות של דיוק לצורך ניתוח מאוחר יותר.

פונקציה נוספת:



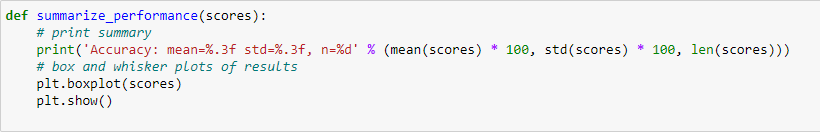
פונקציה שמטרתה לייצא גרפים על בסיס תוצאות האימון.

לדוגמא:

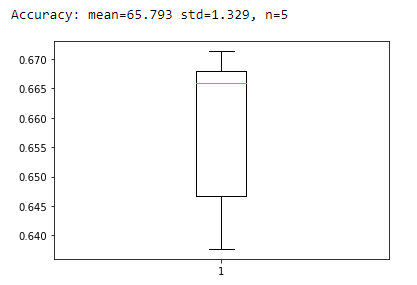


כך נוכל לאשש שהמודל לומד מתוך סט המידע ושפונקציית הלוס באמת יורדת ומתכנסת לערך מינימום ,בתקווה הגלובלי, והדיוק עולה.

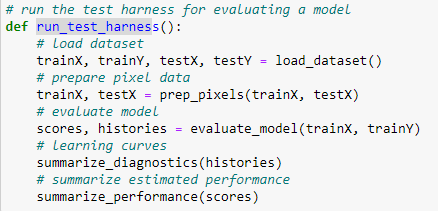
פונקציה נוספת:



בדומה לפונקציה הקודמת זאתי נועדה כדי לנרמל את התוצאות משלב האימון כך שנקבל דיוק ממוצע וסטיית תקן בצורה נוחה.

למשל:

פונקציה נוספת:



מעין פונקציית main שנועדה להקל על הרצה ואימון של המודל.

חלקת קוד נוספת:

Text, letter

Description automatically generated

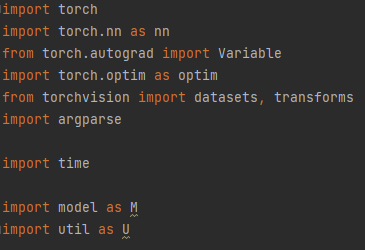


במקום להשתמש בפונקציה שכתבנו run\_test\_harness כתבנו מעין main שם נוכל לשלוט על כמות התמונות (דוגמאות) שבהם נשתמש לסט האימון

* Timing\_pred1/2 – הינם משתנים שמכילים דוגמא אחת כל אחד למטרת חישוב הזמן שלוקח למודל לייצא פרדיקציה.

רשת טרנרית

את הרשת מימשנו גם הפעם באמצעות python אך במקום לעבוד בjupyter השתמשנו בסביבת עבודה של pycharm ואת הרשת מימשנו במקום בtensorflow בpytorch. הייתרון בשימוש בpytorch הוא החופש בבניית הרשת כרצוני והגדרת פונקציות שלי ללא הכרח בשימוש בפונקציות שנתונות בספריה כמו בkeras. בנוסף pytorch מממש רשת נוירונים בצורה של תכנות מונחה עצמים עם ירושה מה שעוזר עם התפעול של הקוד.



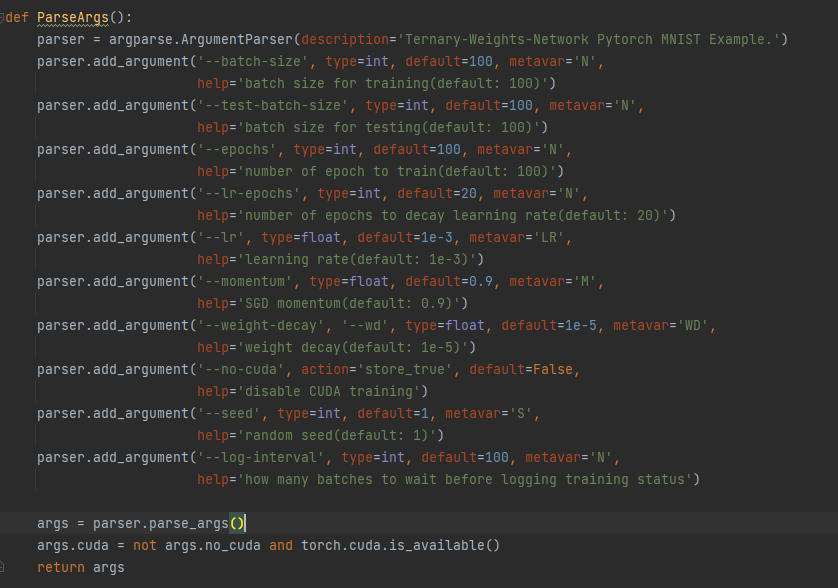
חבילות model וutil הם שלנו ומכילות את המודל ופונקציות עזר למשל שמירת ערכי המשקולות.

פונקציה נוספת:

לצורך הרצת הקוד בצורה יעילה הוספנו parser שמעביר פרמטרים שונים בזמן ריצת הקוד כדי שלא נצטרך לשנות כל פעם את הערכים שלהם במקום המתאים מה שנקרא hard coded.

חלק מהפרמטרים שאפשר לשלוט בהם:

* קצת למידה.
* מספר מחזורי אימון.
* האם להפעיל CUDA (להריץ על חומרה של כרטיסי מסך של NVIDIA).
* גודל הbatch.



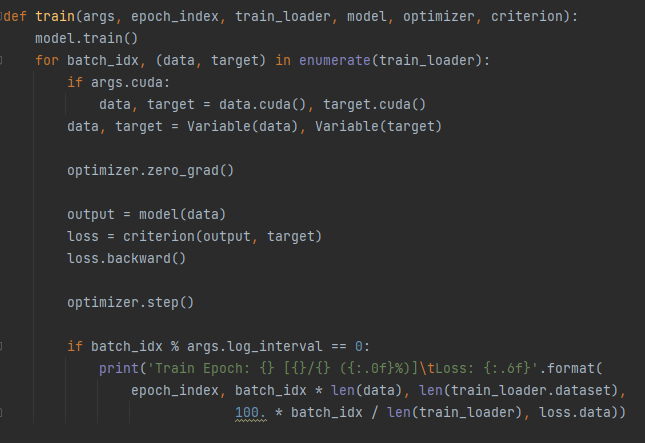
פונקציה נוספת:

מטרתה לבצע את שלב האימון חישוב המשקולות וביצוע forward propagation וbackwards propagation.

נשים לב לשורה model.train(), שלמעשה מציין לpytorch שאנחנו כרגע בתהליך האימון ולא פרדיקציה או טסט.

Optimizer.zero\_grad מאפס את המשתנים הפניימים שמייצגים את הנגזרות לצורך חישוב המשקולות.

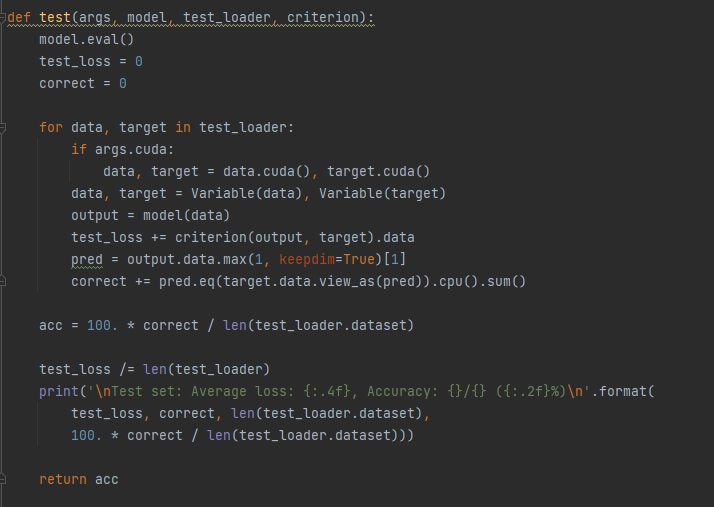
ולבסוף אנו מדפיסים למסך את קצת ההתקדמות כל פעם שהאינדקס של הbatch הוא כפולה של 100(כברירת מחדל).



פונקציה נוספת:

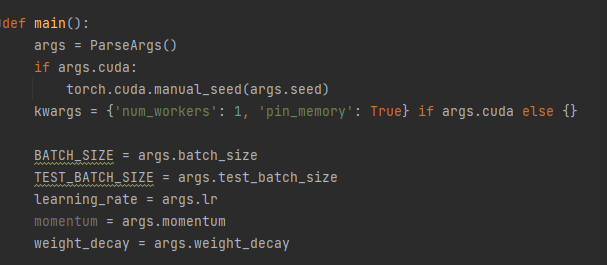
כמו בפונקציה הקודמת model.eval מסמל לpytourch שאנחנו בשלב של אבלואציה.

כאן מחושב הדיוק עבור הepoch הנוכחי כמובן לפי פונקציית הלוס והאופטימייזר שהגדרנו.



הmain:

הפונקציה לוקחת את הקלט של המשתמש ומגדירה את הערכים הפנימיים בהתאם.



Text

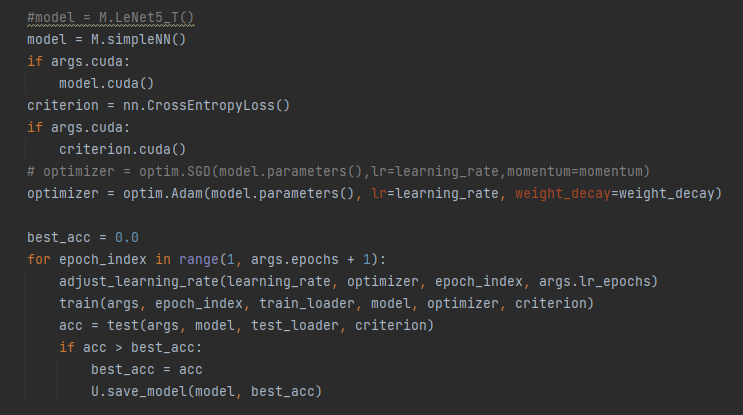
Description automatically generatedאנו משתמשים בפונקציות הבאות כדי לטעון את סט האימון והטסט לתוך מבנה נתונים שpytorch יודע לעבוד איתו.

Text

Description automatically generated with medium confidence

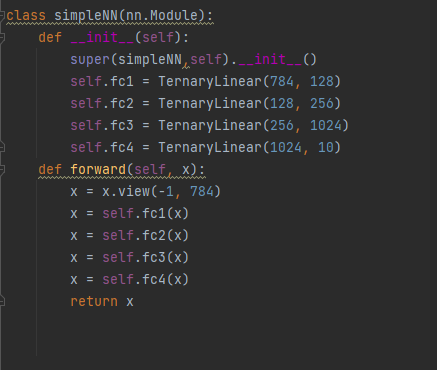
בשלב הבא אנו מייצרים את המודל שלנו ובנוסף מגדירים את הפונקציה לפיה נחשב את ה-loss.

אנו משתמשים באותו אופטימייזר כמו ברשת הבינרית (ADAM)



וכפי שניתן לראות לאחר ההגדרות אנו מתחילים את התהליך, טוענים את המידע קוראים לtrain בודקים איך היה בtest ומדפיסים החוצה דיוק ולוס נוכחיים.

הרשת עצמה:



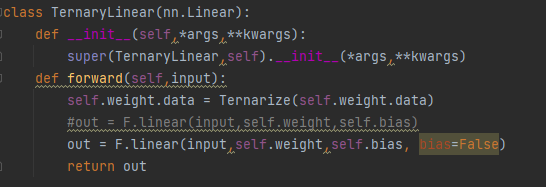
הרשת הנוכחית בנויה מ5 שכבות, 1 קלט, 1 פלט, ועוד 3 שכבות נסתרות בגודל 128,256,1024 בהתאמה.

\*כפי שכבר שהצגנו מספר הנוירונים, מספר השכבות, האופן שבוא המידע עובר (forward propagation) ועוד יכולים להשתנות במהלך הפרויקט.

בנוסף הגדרנו את הforward propagation מפורשות(בעוד שבבינרית לא היינו צריכים לעשות זאת)

ניתן לראות שהפלט של כל שכבה עוברת ישירות לבאה בתור ללא ביצוע שינוי כלשהו.

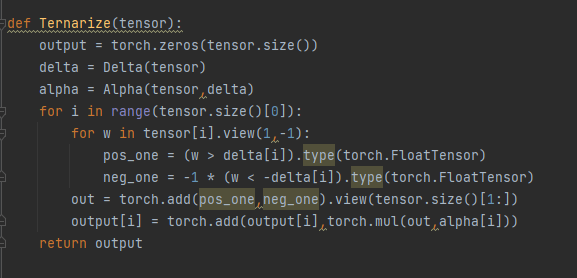
Ternary Dense Layer:



כמו ברשת הבינארית השתמשנו בשכבות Dense אך כיוון שבחרנו להשתמש בpytourch היינו צריכים לממש את השכבה בעצמנו כפי שהחבילה מצפה.

אנו מגדירים את השכבה ואת פונקציית המעבר שלה בהתבסס על פונקציה קיימת linear אבל לפני אנו ממירים את המשקולות לפורמט טרנרי.

Ternarization:



באמצעות פונקציה זו אנו ממירים את המשקולות לפורמט טרנרי באופן הבא:

מחשיבים delta ו – alpha בהם נשתמש כדי לקבוע את ערך הסף מעליו נקבע "1", מתחתיו נקבע

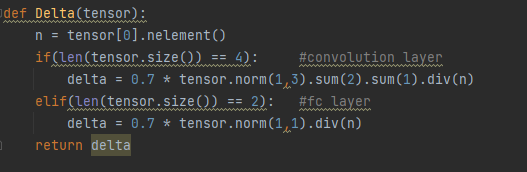
"1-" ובין לבין נקבע 0.

Delta – זהו ערך הסף.

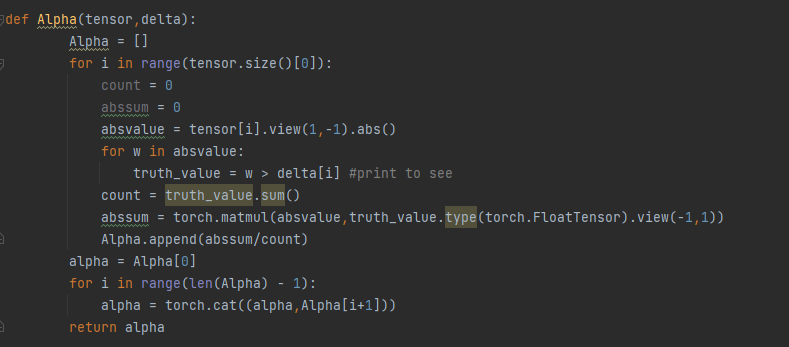
Alpha – זהו אותו ערך שאם נחלק בו נקבל את המשקולת בפורמט טרנרי כלומר, במהלך האימון המשקולות הם: .

העובדה שבשלב האימון אנו לא משתמשים במשקולות טרנריות, לא משפיעה על התוצאה הסופית משום שהמודל לומד על היחס והדפוסים במידע ולאו דווקא את המידע עצמו.

פונקציית Delta:



\*ישנו חלק גם של שכבת קונבולוציה עליו לא נדבר ולא נשתמש במהלך הפרוייקט.

פונקציית Alpha:

פונקציה זו עושה שני דברים:

1. מגלה את אותו ערך שאם נחלק בו נקבל את הפורמט הטרנרי.
2. כל הערכים שנמצאים בין a ל-a ימופו כדי שלאחר מכן נוכל לאפס אותם הרי ערך המשקולת שלהם שווה ל-"0".

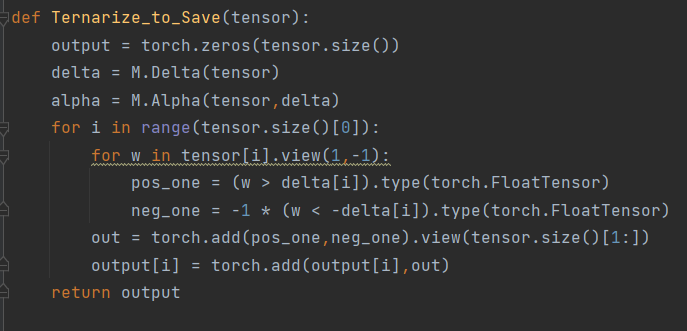
**מימוש Forward Propagation בC**

על מנת להשתמש במשקולות שמתקבלות כתוצאה מתהליך אימון הרשתות כתבנו קוד שמדמה חישוב ברשת נוירונים בשפת C.

הוספנו קוד לשלב שבו תהליך האימון קורה (עוד בשלב שמריצים את המודל בפייתון) על מנת לשמור את ערכי המשקולות לקבצי טקסט בודדים ושונים זה מזה.

לאחר מכן בעזרת סקריפט פורמטינג שכתבנו בפייתון אנו עוברים על קבצי הטקסט ומסירים תווים לא רצויים שC לא יודע להתמודד איתם.

לבסוף אנו מגיעים סוף סוף לקוד הC שמכיל פונקציות לטעינה של משקולות למערכים המתאימים להם ובנוסף כאלה שמדמות ריצה וחישוב של רשת שלבסוף מניבה פרדיקציה.

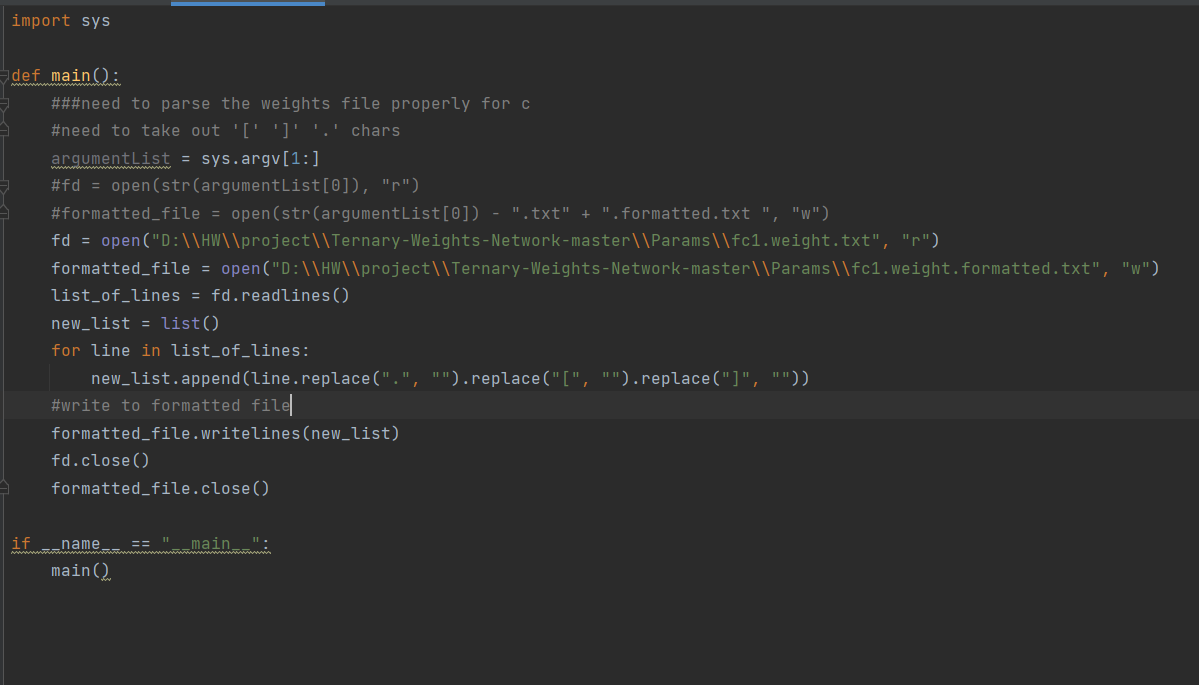


פונקציה זו ממירה את ערכי המשקולות ששמורים במודל מתצורה של (0,a,-a) ל(0,1,-1)



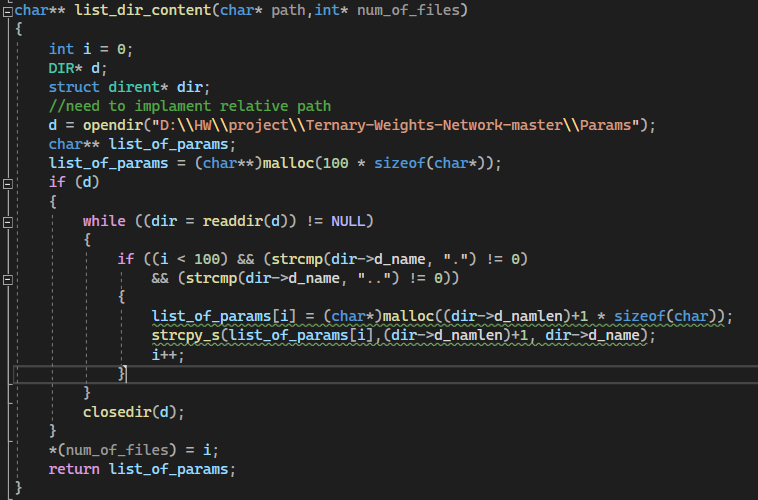
פונקציה זו פותחת תקייה חדשה בשם Params בה נשמרים כל קבצי המשקולות

ושומרת עבור כל שכבה את ערכי המשקולות הרלוונטיים אליה בקובץ נפרד.



פונקציה זו מבצעת את הסרת התווים שאינם רצויים מתוך קובץ משקולות בודד.

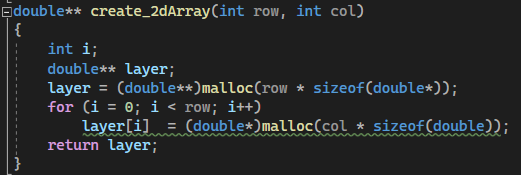
יש להריץ את הסקריפט עבור כל מסמך שנמצא בתיקייה Parmas.



פונקציה זו מייצאת מערך שמכיל בכל תא שם של קובץ אחד מתוך התקייה Params.

דוגמא לשימוש:

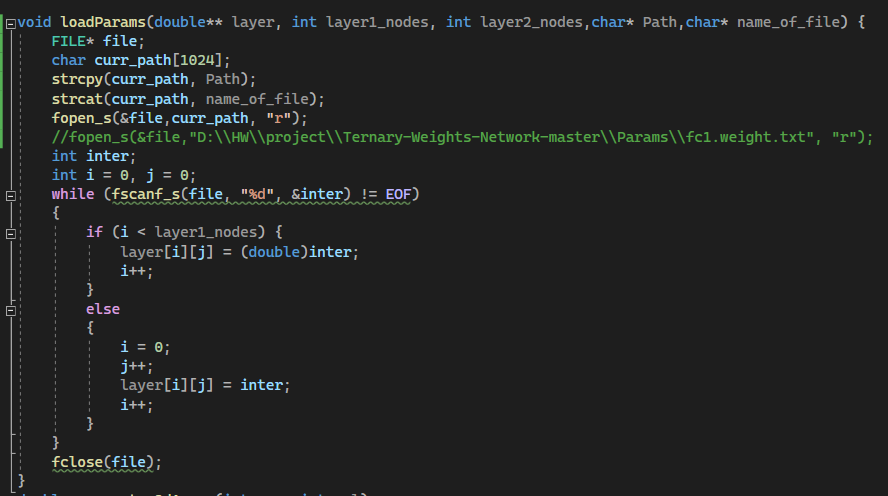




פונקציה זו יוצרת מערך דו מימדי מגודל שאנו רוצים לצורך שמירת המשקולות.

דוגמא לשימוש:





פונקצייה זו טוענת לתוך מערך שכבה בודד את המשקולות עבורו.

דוגמא לשימוש:



Text

Description automatically generated

פונקציה שמדמה חישוב בודד בין שכבה לשכבה לדוגמא הפונקציה יכולה לחשב מה קורה למידע במעבר דרך השכבה הראשונה כלומר ביצוע אינטרקציה בין ערכי הכניסה לבין ערכי משקולות הרלוונטיים.

דוגמא לשימוש:

Text

Description automatically generated

פונקציה זו מחזירה את האינדקס של הנוירון בעל תוצאת חישוב מקסימלית.

עבורנו ברשת הטרנרית, ערך המקסימום הוא מבציע על תוצאת הפרדיקציה כלומר

במידה ונוירון 5 בעל ערך מקסימלי נאמר שהניחוש של המודל הוא 5(התמונה מתאימה לתמונה של הסיפרה 5)

תוצאות:

בכדי להשוות בין הרשת הבינארית לטרנרית שינינו את מספר האיטרציות, כמות השכבות, מספר הנוירונים בכל שכבה וכמות המידע שעליו המודל מתאמן.

מדד ההשוואה שבו השתמשנו היה אחוז הדיוק של המודל כתלות בפרמטר אותו אנו משנים ובנוסף זמן הריצה לביצוע חישוב\פרדיקציה בודד\ת.

ראשית, שנינו את מספר השכבות במודל ומדדנו את אחוז ההצלחה לפרדיקציה נכונה:

ניתן לראות בברור כי ביצועי הרשת הטרנרית עולים על זו של הרשת הבינארית ובפרט נמצאים במגמת עלייה ככל שמספר השכבות גדל אך עם כמות שכבות גדולה מזהים האטה בקצב הגדילה ושאיפה לערך מקסימלי.

נשים לב כי עבור הרשת הבינארית ירד ערך הדיוק עבור כמות שכבות שנמוכה מ5 ולאחר מכן מתקנת כלפי מעלה והדיוק עולה מעט מעל הערך ההתחלתי.

שנית, שנינו את כמות הנוירונים הכוללת במודל ומדדנו את אחוז ההצלחה לפרדיקציה נכונה:

ניתן לראות בברור כי עבור עליה בכמות הנוירונים הכוללת, עלה אחוז הדיוק עד התייצבות על ערך סף(65% עבור BNN ו85% עבור TNN)

\*חשוב לזכור כי זהו ערך הסף עבור הקונסטרוקציה המסוימת עבורה שינינו רק את כמות הנוירונים הכוללת. בכל מקרה אחר ייתכן וערך הסף היה משתנה אך מתנהג בצורה זהה.

בנוסף אפשר לראות מיד כי הרשת הטרנרית עלתה בביצועה על הרשת הבינארית אפילו עבור אותם הפרמטרים אבל העלייה בביצועים תתבצע באופן זהה עבור שתי הרשתות

שלישית, מדדנו את הזמן הדרוש למודל כדי לבצע חישוב בודד(פרדיקציה) ושינינו את כמות הנוירונים הכוללת במודל:

כפי שניתן לראות מן הגרף זמן החישוב של הרשת הבינארית היה משמעותית נמוך יותר משל הרשת הטרנרית, למעשה כדי להציג את התוצאות המשותפות נדרשנו לשנות את סקלת הזמן של הרשת הבינארית(פי 10).

\*חשוב לציין כמה דברים שמשפיעים על התוצאות שלנו:

א. החבילה שבה מימשנו את הרשת הבינארית, מותאמת לרשתות בינאריות בפרט ולכן כל הפעולות והפונקציות שבאים לידי שימוש בקוד מתוך החבילה כבר עברו אופטימיזציה ולכן החישובים יעילים ומהירים יותר.

ב. אילו תוצאות משוערות בעולם התוכנה שעלולות להשתנות במידה והיינו בודקים בפן החומרתי עם רכיבים ייעודים לכל רשת(אפשר לבצע סימולציות למשל בורילוג).

מסקנות

כפי שהצגנו בניתוח התוצאות ניתן היה לראות שינויים בביצועי שתי הרשתות הן בפן השטח, הספק וזמן.

ראינו התנהגות זהה ברמת התנהגות הגרפים ושאיפתם לערך מסוים.

כפי שראינו GAIN CELL של TNN אמנם מורכב מכמות קטנה יותר של טרנזיסטורים אל מול השיטה הרווחת(5T ,6T) אך מוסיף מורכבת לניתוח המידה שכולל ערך נוסף אפשרי(-1 ,0 ,1).

ובמידה השנייה נוכל לזהות את זמן החישוב הקטן של BNN אל מול TNN.

מסקנה חשובה נוספת היא שתמיד ניתן לספק לרשת עוד מקרים ודוגמאות ולאמן אותה בצורה יותר יסודית וכך להביא לביצועים טובים יותר.