**דו"ח מטלה מספר 3 – רשתות נוירונים**

**מגישות:** נגה בן ארי 208304220, שחר אורון 322807231

קישור לגיט:

<https://github.com/Shachar-Oron/Neural-Network-Genetic-algorithm>

**הרצת התרגיל:**

To run the app, follow these steps:

Navigate to the dist folder.

To run buildnet0.exe, enter the following command:

.\ buildnet0.exe

Then the app will ask you to input your test and train files for buildnet0

To run buildnet1.exe, enter the following command:

.\ buildnet1.exe

Then the app will ask you to input your test and train files for buildnet1

Both apps will create ‘wnet’ files as outputs that hold the best neural network structure for each runnet .exe files.

To run the ‘runnet0.exe’ program, enter the following command:

.\ runnet0.exe

Then the app will ask you to input your file for runnet0

Now do the same for runnet1.exe with the relevant input file.

The final results will be in the files: results0.txt & results1.txt

בתרגיל הנוכחי התבקשנו לבנות רשתות נוירונים באמצעות אלגוריתם גנטי שילמדו תבניות של קבצים שניתנו לנו ויוכלו לנבא האם מחרוזת מסוימת מתאימה לתבנית או לא.

התוכנית buildnetמחולקת ל4 חלקים:

1. קריאת הדטא שניתן כקלט מהמשתמש וההשמה שלו כמשתנים הרלוונטיים.
2. הclass של Individual המייצג רשת ניורונים, פרט באוכלוסיה. ה class מכיל את פונקציית הforward והprediction הנצרכות לforward propagation.
3. הclass של האלגוריתם הגנטי המנסה למצוא את הרשת עם המשקולות המניבים את התוצאה הטובה ביותר באמצעות פונקציית הfitness.
4. הפעלת פונקציית הpredict והforward על סט המבחן עם רשת הנוירונים בעלת הפיטנס הגבוה ביותר שקבלנו מהאלגוריתם הגנטי על סט האימון.
5. יצירת קובץ went המכיל את המבנה של רשת הניורונים הטובה ביותר שסופקה מהאלגוריתם הגנטי.

התוכנית runnetמחולקת ל2 חלקים:

1. קריאת קובץ הtest שהמשתמש מספק וקריאת קובץ הwnet
2. הפעלת פונקציית predict עם הרשת שקבלנו מ wnetעל הinputs המתקבלים מקובץ הtest.
3. יצירת קובץ results המכיל את הlabels שחושבו

|  |  |  |
| --- | --- | --- |
| **היפרפרמטר** | **Buildnet0** | **Buildnet1** |
| גודל אוכלוסיה | 150 | 150 |
| מספר דורות | 500 | 500 |
| תדירות מוטציות- mutation rate | 0.3 | 0.3 |
| גודל שכבת הinput | 16 | 16 |
| גודל שכבה אחת | 32 | 32 |
| גודל שכבה שתיים | 16 | 16 |
| גודל שכבת הoutput | 1 | 1 |
| פונקציית אקטיבציה | leaky\_relu | sigmoid |
| Selection rate | 0.1 | 0.1 |
| npairs | 3 | 3 |
| untouched | 0.3 | 0.3 |

\* **npairs** - 3 מספר המוטציות שפונקצית הלמארק מבצעת בכל אינדיבידואל בהתאם לmutation rate.

\* **untouched** אחוז פרטים שנשמרים וממשיכים מדור לדור.

**\* פונקצית אקטיבציה-** הגדרנו 3 פונקציות אקטיבציה שאיתן הרשת יכולה לפעול(ניסינו את האפשרויות השונות כדי למקסם ביצועים): sigmoid, relu, leaky relu.

בbuildnet1 לאורך הרשת פונקציית האקטיבציה היא leaky-relu ובשכבה האחרונה היא sigmoid בbuildnet0 פונקציית האקטיבציה ברשת היא רק leaky-relu.

**מבנה הרשתות:** סה"כ כל רשת (פרט באוכלוסיה) בנויה מ4 שכבות שכוללות שכבת input, כיוון שכל מחרוזת היא באורך 16 הגדרנו אותה כך ב2 הקבצים. לאחר מכן עוד 2 hidden layers ושכבת output שחייבת להיות בגודל 1.

**ייצוג הדטה**:

* ייצוג הדטה שחלצנו מהקבצים שניתנו כקלט ע"י המשתמש (קובץ אימון וקובץ מבחן) מיוצגים בתור מערכים של labels וexampels עבור סט האימון וסט המבחן.
* ייצוג רשתות הנירונים הוא באמצעות ndarray המכיל 3 שכבות כאשר כל שכבה היא מערך של משקלים.

**תיאור התוכנית buildnet :**

1. **קריאת הדטא:**

לאחר שאתחול האוכלוסיה מתבצע, מתבצעת קריאה של הדטא הנתון והשמה שלו במשתנים lables,examples גם לאימון וגם למבחן.

1. **Neural network classes- Individual class:**

* **add\_weight**: הוספת משקלים לרשת על ידי יצירת מטריצת משקל אקראית בצורה של (פנימה, החוצה)- כלומר, משקולות עבור השכבה הקודמת והשכבה הבאה. את המשקל החדש הוא מחשב בצורה הבאה:

*np.random.randn(inside, outside) \* np.sqrt(1 / inside)*

לאחר מכן הוא מוסיף את מטריצת המשקל הזו כשכבה חדשה לרשת.

* **Forward**: המתודה לוקחת דוגמאות קלט, שהן מערך NumPy, ומעבירה אותן דרך כל שכבה של הרשת באמצעות כפל מטריצה ​​(np.dot). הפלט של כל שכבה מועבר דרך פונקציית אקטיבציה לפני שהוא מוזן לשכבה הבאה. התוצאה של הפעלת השכבה האחרונה מוחזרת כפלט של הforward.
* **Predict:** המתודה מנבאת את הפלט באמצעות פונקציית forward על גבי סט הדוגמאות הנתון. לאחר מכן היא קובעת סף של 0.5 כדי להמיר את הפלטים לחיזויים בינארים ומוסיפה אותם למערך שחוזר כערך המוחזר של הפונקציה.
* **accuracy**: הפונקציה יוצרת מערך בוליאני ובו כל רכיב הוא true אם החיזוי שחושב דומה לlabel האמיתי של סט האימון או false אחרת. לאחר מכן, היא ממירה את המערך הבוליאני כמערך של אפסים ואחדות ( false הופך ל0 וtrue ל 1). בשלב הבא היא מחשבת את הממוצע של המערך וזהו ה accuracy.

1. **תיאור האלגוריתם הגנטי:**

1. **פונקציית selection** – השתמשנו ב2 פונקציות selection.

* **פונקצית selection רגילה**: לקחנו את האוכלוסיה מיינו אותה לפי הfitness score בסדר יורד. לאחר מכן, חילצנו מהרשימה הממויינת את האחוז המסויים של הפריטים הכי טובים באוכלוסיה לפי הselection rate שהגדרנו. לבסוף הפונקציה מחזירה 2 רשימות- את החלק של האוכלוסיה עם הפיטנס הטוב ביותר ואת החלק הפחות טוב שנותר מהאוכלוסיה.
* **Rank selection**: את ההורים של הדור הבא נבחר מהאוכלוסיה הפחות טובה (האוכלוסיה שחזרה בselection הקודם כאוכלוסיה הפחות טובה) באמצעות הrank selection. מה ששונה בselection הזה זה שנותנים הסתברויות שונות לכל פריט באוכלוסיה להיבחר ל"אליטה" כאשר ההסתברות גדלה כשהfitness score של אותו פריט יותר גבוה. את האוכלוסיה הטובה לא הכנסנו לrank selection וממנה שמרנו בצד אחוז מסויים לפי פרמטר ה untouched. החלק הזה באוכלוסיה לא יעבור מוטציות וקרוס אובר והוא בעצם יישמר עד הדור הבא.

2.**פונקציית** **crossove**r – הרעיון של הפונקציה הוא לבחור שני פריטים כאשר אחד מהם יהיה מהtop individuals שהגיעו מפונקציית הselection וההורה השני מגיע מהselected\_parents שהגיעו מהrank selection. בכל אחד מההורים היא בוחרת ניורון רנדומלי ולאחר מכן מתבצעת החלפה בין שני הניורונים שנבחרו יחד עם המשקולות המחוברות אליהם מהשכבה הקודמת ומהשכבה הבאה. איך זה נעשה? הפונקציה מאתחלת שני אינדיבידואלים חדשים שהם יהיו הצאצאים של פעולת הcrossover, לאחר מכן מתבצע מעבר מקביל בלולאה על השכבות של שני ההורים כאשר בכל פעם יש בחירה אקראית של ערך בוליאני (true או false) עבור כל עמודה במטריצת המשקולות של השכבה הנוכחית – העמודות שהן true יהיו שייכות לניורון שנבחר ויבוצע בהן החלפה, אלה שהן false ישארו אותן העמוד בילדים שייווצרו. לאחר מכן מתבצעת פעולת הcrossover עבור השכבה הנוכחית, נוצרת לנו מטריצת משקלים חדשה c1w שבה הערכים מ-w1 משמשים עבור עמודות שבהן עמודות היא True, והערכים מ-w2 משמשים עבור עמודות שבהן העמודות היא False. זה יוצר שילוב של משקלים משתי רשתות האם. לאחר מכן מתבצעת פעולה הcrossover עבור הילד השני ונוצרת מטריצת משקלים חדשה c2w שבה הערכים מ-w2 משמשים עבור עמודות שבהן עמודות היא True, והערכים מ-w1 משמשים עבור עמודות שבהן העמודות היא False.

לבסוף מתבצעת הוספה של מטריצות המשקלים החדשות לשכבות של הילדים בהתאמה, והחזרה של 2 הילדים.

3**.פונקציית mutation** – מגרילה מספר רנדומלי בין 0 ל1 ובודקת אם הוא קטן מתדירות המוטציה שהגדרנו כמשתנה גלובלי. לאחר מכן בוחרת באקראי אינדקס מטווח מספר השכבות ברשת הנוירונים – מה שקובע איזו שכבה תעבור מוטציה.  
בהמשך בוחרת אינדקס אקראי בטווח של מספר המשקולות בשכבה הנבחרת – מה שקובע איזה משקל בשכבה יעבור מוטציה. הפונקציה יוצרת משקל אקראי ומחליפה באמצעותו את המשקל שנבחר לעבור מוטציה.

4. **פונקצית למארק:** האלגוריתם יקרא לפונקציה כאשר ערך הbest\_fitness נשאר אותו הדבר במשך יותר מ3 דורות. במקרה זה, הפונקציה תחולל מוטציות בכל אינדיבידואל באוכלוסיה (אם כאשר אנו מגרילים ערך רנדומי- הוא גבוה יותר מהmutation rate, ככה בעצם אנו שומרים על הmutation rate שיהיה 30%) כמס' npairs(החפר פרמטר שהגרדנו מראש) פעמים. היא תחשב את הפיטנס החדש שנוצר בעקבות המוטציה ורק אם הוא יותר גבוה מהפיטנס שהיה לאותו האינדיבידואל לפני המוטציה- היא תוסיף אותו לאוכלוסיה של הדור הבא. אחרת, היא תכניס לדור הבא את האינדיבידואל שהיה מלפני המוטציה.

5.**פונקציית fitness** – הפונקציה מחשבת את הpredictions על גבי סט האימון באמצעות פונקציית הpredict שיש לכל אינדיבידואל באוכלוסיה ומחזירה את הערך המוחזר מפונקציית הaccuracy כאשר שולחים אליה את הpredictions הנ"ל.

**תמונה שמכילה טקסט, תרשים, קו, עלילה

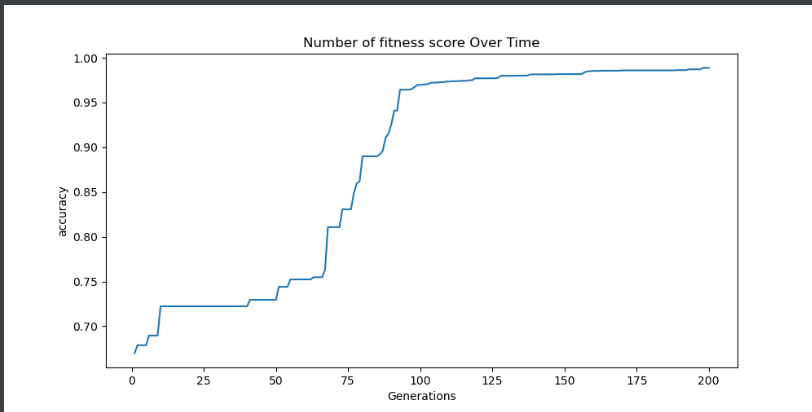
התיאור נוצר באופן אוטומטיהצגת plots עבור שימוש בפונקציות אקטיבציה שונות:**

**Leaky relu**

**Buildnet0**

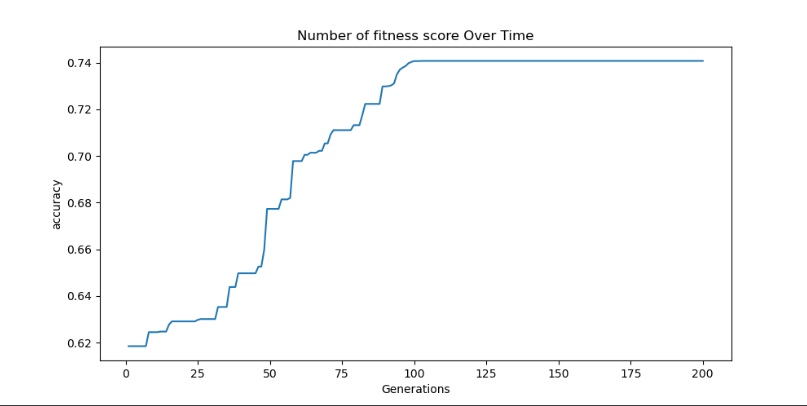
לאחר 200 דורות של ריצת האלגוריתם, רשת הנוירונים הטובה ביותר בעלת דיוק של 0.98281 על קבוצת האימון ודיוק של **0.97975** על קבוצת המבחן.

**relu**



לאחר 200 דורות של ריצת האלגוריתם, רשת הנוירונים הטובה ביותר בעלת דיוק של 0.98869 על קבוצת האימון ודיוק של **0.9885** על קבוצת המבחן.

**sigmoid**

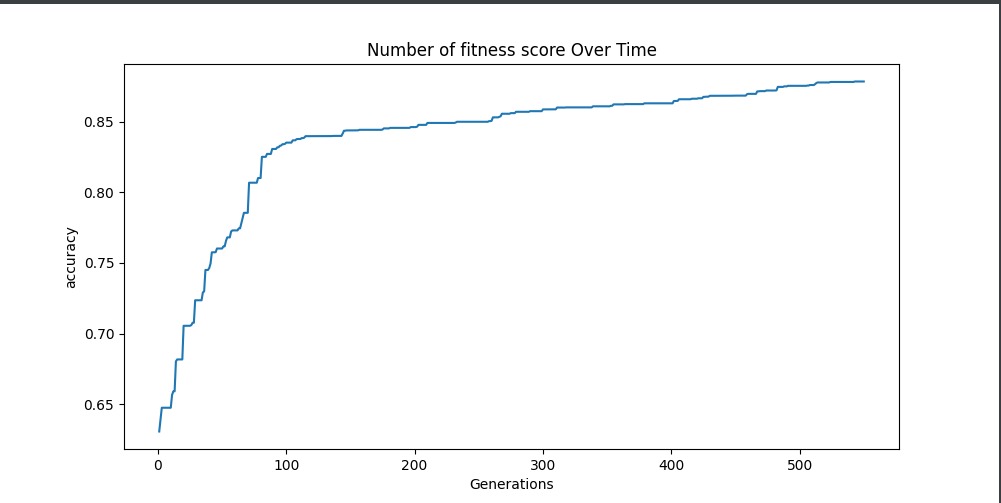


לאחר 200 דורות של ריצת האלגוריתם, רשת הנוירונים הטובה ביותר בעלת דיוק של 0.74075 על קבוצת האימון ודיוק של **0.74875** על קבוצת המבחן.

**relu**

**Leaky relu**

**Buildnet1**



לאחר 500 דורות של ריצת האלגוריתם, רשת הנוירונים הטובה ביותר בעלת דיוק של 0.8911 על קבוצת האימון ודיוק של **0.88** על קבוצת המבחן.

**Leaky relu**

תמונה שמכילה טקסט, קו, צילום מסך, תרשים

התיאור נוצר באופן אוטומטילאחר 500 דורות של ריצת האלגוריתם, רשת הנוירונים הטובה ביותר בעלת דיוק של 0.98475 על קבוצת האימון ודיוק של **0.98825** על קבוצת המבחן.

**sigmoid**

**relu**

**Leaky relu**

**relu**

תמונה שמכילה טקסט, קו, תרשים, עלילה

התיאור נוצר באופן אוטומטי

לאחר 500 דורות של ריצת האלגוריתם, רשת הנוירונים הטובה ביותר בעלת דיוק של 0.85431 על קבוצת האימון ודיוק של **0.8435** על קבוצת המבחן.

**מסקנה**: עבור buildnet0 פונקצית הleaky relu עבדה בצורה הטובה ביותר ואילו בbuildnet1 פונקצית האקטיבציה העדיפה היא sigmoid.

**חוקיות התבניות:**

Nn0: מס׳ האחדות צריך להיות בין 8-12

Nn1: פחות מ8 אחדות

אנו סבורות כי השוני בין התבניות מתבטא באופן השונה בו בנינו את רשתות הנוירונים עבור הקבצים, ב buildnet0 פונקצית הleaky relu עבדה בצורה הטובה ביותר, ידוע כי פונקציה זו מאפשרת לערכים שליליים קטנים לעבור, ומספקת פלט רציף. על ידי שימוש ב- leaky relu, אפשרנו לרשת ללכוד קשרים לא ליניאריים בנתונים בצורה יעילה. בנוסף, פונקציה זו מונעת את בעיית ה dying ReLU, שבה נוירונים יכולים להפוך ללא פעילים במהלך האימון. היא משמשת לעתים קרובות ברשתות ניורונים עמוקות ויכולה להיות מועילה כאשר מתמודדים עם דפוסים ונתונים מורכבים. ואכן נשמע כי התבנית של nn0 מורכבת מזו של nn1. בנוסף ראינו כי פונקציית הsigmoid אכן לא עבדה מספיק טוב לתבנית מורכבת זו.

בbuildnet1 נשתמש בפונקצית הsigmoid בשכבה האחרונה של הרשת (בעוד שבשאר השכבות השתמשנו בפונקציית האקטיבציה leaky relu), הפונקציה ממפה את הקלט לטווח שבין 0 ל-1. היא מתאימה לבעיות סיווג בינארי שבהן יש לפרש את הפלט כהסתברות. על ידי שימוש בsigmoid, רשת הניורונים יכולה ללמוד להוציא ערכים קרובים ל-1 כאשר מספר ה-1 נמצא מתחת ל8(הtreshold בתבנית(, מה שמצביע על נוכחות התבנית הרצויה. זה עוזר להפוך את הרשת לרגישה להבדלים קטנים והסתברויות בנתוני הקלט. הפונקציה מתאימה לבעיות שבהן גבול ההחלטה אינו מורכב במיוחד.

לסיכום, Leaky ReLU יכולה להתמודד עם מגוון רחב יותר של תבניות, מה שמאפשר גמישות רבה יותר, בעוד ש-sigmoid מתאים למשימות סיווג בינארי ולכידת הסתברויות.