בס"ד

דו"ח תרגיל בית 3 ביולוגיה חישובית

נתנאל לנדסמן 315873588

גל לוי 208540872

במשימה זו קיבלנו 2 קבצים, בכל קובץ יש 20,000 מחרוזות בינאריות, כך שכל מחרוזת היא 16 ספרות. לאחר כל מחרוזת יש את הסיפרה 0 או 1 כך שכל המחרוזות שיש אחריהם את הסיפרה 1 מתאימות לחוקיות מסוימת. מטרתנו הייתה למצוא את אותה חוקיות ע"י בניית רשת נוירונים, אך בשונה מרשת נוירונים רגילה שעדכון המשקולות נעשה ע"י שיטת ה-back propagation , בתרגיל זה נדרשנו לבצע את העדכון בעזרת אלגוריתם גנטי.

הוראות הפעלה:

קבצי ההרצה נמצאים בגיט בקישור הבא: https://github.com/landsboy/Computetional-Biology-Ex3

לאחר הורדת ה-repo תקבלו 3 תיקיות:

1) **תיקיית nn0** - המכילה 2 קבצי הרצה (exe) אחד בשם buildnet0.exe שלאחר שנלחץ עליו לחיצה כפולה (ונמתין כ- 10 שניות עד שהיא תאתחל את התוכנית) נצטרך להזין אליו את הpath של קובץ **כל** הדגימות. נשים לב כי התוכנית בעצמה תחלק אותה ל2 קבצי שונים של test & train [1]. נציין שאנחנו כבר הרצנו את הקובץ ונוצר wnet0.txt וזה כי נאמר שאתם לא תריצו את הקובץ הזה.
לאחר הרצת התוכנית לאותה תיקייה ייווצר קובץ בשם went0.txt (שנפרט בהמשך על הפורמט שלו) המכיל את המשקולות שעברו אימון.
כעת, נריץ את runnet0.exe (שוב, נמתין כ- 10 שניות עד שהיא תאתחל את התוכנית) ונזין לה קובץ חיזוי. היא תייצר קובץ פרדיקציה בשם predic_results0.txt עם תיוג בכל שורה, בהתאמה לכל שורה בקובץ חיזוי שהזנתם לו.
2) **תיקיית nn1** – הזהה לחלוטין בתוכן שלה לתיקייה nn0 רק שאצלה לכל קובץ יש 1 בסוף במקום 0.
3) **ass_code**- הקוד של התרגיל. (שנכתב בשפת פייתון)

הקובץ שנוצר לאחר הרצת buildnet.exe זה הקובץ went.txt והפורמט שלו הוא כדלקמן:

```
weights: [[0.07674224968367091], [0.07581074742999283], [0.08637774445965676], [0.07351861802214693], [0.07853171832693413], [0.06772737997762322],
biases: -0.557590160326378
model: [16, 1]
```

כפי שניתן לראות כל שורה מייצגת תיאור במבנה המודל.. בשורה הראשונה יש לנו את גודל המשקולות שבכל שכבה (במקרה שלנו הואיל ויש רק שכבה אחת אזי נקבל מערך משקולות בודד) בשורה השנייה אנחנו מציגים את ה-bias שעוזר לנו לתהליך הלמידה. ובשורה האחרונה את הארכיטקטורה המדויקת של המודל - כל מערך מייצג שכבה ברשת הנוירונים ובכל מערך יש 2 מספרים: מספר שכבות הinput לשכבה, ומספר שכבות הoutput של אותה שכבה. במקרה שלנו הואיל והמודל מורכב משכבה אחת יש לנו פה רק מערך אחת עם כניסה ל16 מספרים (הואיל והinput הוא מספר עם 16 ביטים) ויציאה למספר אחד שנכנס לפונ' אקטיבציה.

מאפייני האלגוריתם הגנטי:

בתרגיל בית 2 הגענו למסקנה שהמודל הכי טוב עבור פיתרון הקוד המוצפן הוא המודל הקלאסי (beta =1) אשר נתן העדפה לטובת סלקציה של האינדיבידואלים הכשירים ביותר לטובת העמדת צאצאים בתור הבא, אך עם זה גם נתן סיכוי עבור שאר הפריטים באוכלוסייה להיבחר ולהעמיד צאצאים. בתרגיל זה החלטנו להשתמש במודל הלמארקי (beta = 0), ראשית כי במשימה הזו הוא נתן לנו את התוצאות ואחוזי דיוק הכי גבוהים כפי שנתאר בהמשך. ושנית, גם בתרגיל הקודם אמנם המודל הקלאסי נתן תוצאות הכי טובות אך שני לו היה המודל הלמארקי ולא בפער גדול.. אז בהחלט היה ניתן לצפות שבמשימות אחרות המודל הלמארקי אף יעקוף את המודל הקלאסי וייתן תוצאות טובות יותר.

---

[1] ההוראות לא היו כל כך ברורות האם התוכנית צריכה לקבל מהמשתמש קובץ אחד של כל הדגימות והקוד מחלק אותה לשניים, או לחלופין שהמשתמש צריך להזין 2 קבצים שונים של train&test. שמנו לב שזה לא היה ברור גם לשאר חברי הקורס לכן נשמח להתחשבות במידה ומה שעשינו זה לא נכון או לא ברור.

בס"ד

<u>הפרמטרים של האלגוריתם הגנטי:</u>

1.pop_size - גודל האוכלוסיה (מספר האינדיבידואלים) בה אנו משתמשים. במקרה זה ניתן לראות כי בחרנו גודל אוכלוסייה של 100 הואיל וכפי שנראה בהמשך זה הגודל שנתן לנו את הביצועים הכי טובים, מדוייקים ומהירים.

2. GENE –מספר הדורות המקסימלי שהאלגוריתם יבצע לצורך חיפוש פתרון אופטימלי הוא 500. מספר 500 נבחר מהסיבה שאנו לא רוצים לרוץ לנצח אלא רוצים להגביל את מספר הדורות באופן שבו גם אם לאחר מספר מסוים של דורות לא הגענו להתכנסות, נעצור את פעולת האלגוריתם (בפועל לרוב מופעל תנאי התכנסות מוקדם יותר עליו יפורט במהשך)

3. sel_rate - כאמור, מאופן פעולתו של האלגוריתם הגנטי, לאחר כל דור מתבצעת סלקציה של האינדיבידואלים באוכלוסיה. בכל דור יש הכפלה של מספר האינדיבידואלים וכמו כן פעולות קרוסאובר ומוטציות לצורך קבלה של אינדיבידואלים בעלי שונות מסוימת מהאינדיבידואלים הקודמים ובכך הגדלת האקספרסיביטי של האוכלוסיה לטובת גילוי של פתרונות טובים יותר. כאמור, אנו הגדרנו עבור פרמטר זה ערך של 0.5 וזה אומר שעבור כל דור לאחר שביצענו הכפלה של האוכלוסיה וקבלת כמות צאצאים כפולה, בשלב הסלקציה נצמצם פי 0.5 (מחלק ב2) את כמות הצאצאים באופן בו אנו מדרגים את כל האינדיבידואלים לפי ערך הפיטנס שלהם כך שרק הצאצאים הכשירים ביותר ממשיכים לדור הבא

4. mut_rate - כפי שצוין, בכל דור ודור מתבצעות מוטציות עבור האינדיבדואלים באוכלוסיה. ערך שיעור המוטציות שבחרנו מוגדר להיות 0.01 וזה אומר שבהעדפה נמוכה יותר נרצה לבצע מוטציות באוכלוסיה. כלומר, 1 ל100 פעמים אכן באמת נבצע מוטציה. הסיבה שבחרנו בערך זה עבור שיעור המוטציות מקורה בהסתמכות על ביצוע הקרוסאובר בין האינדיבידואלים באופן שבו השונות שנוצרת באוכלוסיה בין הדורות מקורה בשיחלופים של ערכים קיימים (שככל הנראה טובים באופן יחסי בהנחה שהם שרדו סלקציה) על פני הוספת ערכי מוטציה שרירותיים. עם זאת, קיימת חשיבות לביצוע מוטציות על מנת להגדיל את השונות באוכלוסיה ומסיבה זו לא ויתרנו בכלל על יצירת המוטציות

5. MINIMA - פרמטר זה למעשה מכתיב את מספר הדורות בהם לא היה שינוי כלשהו בערך הפיטנס ולפיכך זה סממן עבור התכנסות האלגוריתם. במקרה שלנו קבענו את ערך הפרמטר ל20 כלומר במידה ובמשך 20 דורות לא משתנה ערך הפיטנס עבור האינדיבידואל המוביל באוכלוסיה אנו יודעים שככל הנראה הגענו למינימום (מקומי או גלובאלי) ולכן זה סימן לעצירת פעולת האלגוריתם ולהתכנסות.

נציין כי הפרמטרים הנ"ל זהים הן ל-buildnet0 והן ל-buildnet1 .

מבנה הרשת:

כאשר רצינו לבחור את הרשת שאיתה נפתור את התרגיל, ראשית ניסינו עם המודל הכי פשוט מתוך מחשבה שהוא גם יהיה הכי מהיר עקב הפשטות שלו. לכן בהתחלה הרשת שבנינו הייתה **פרספטרון** שהיא רשת הנוירונים הכי פשוטה – בעלת שכבה אחת בלבד. לשכבה נכנס input של 16 פרמטרים (הואיל וכל מחרוזת בדגימות שלנו יש בה 16 תווים כפי שתיארנו לעיל) והרשת מבצעת מכפלת וקטורים של 16 הפרמטרים עם וקטור המשקולות שלנו (שהולך ומשתנה במהלך ריצת התוכנית)[2] ומוציא לנו כ-output ערך בודד שעליו מפעלים פונקציית אקטיבציה שבחרנו אותה להיות sign (שוב, הלכנו בהתחלה על פשטות) פונקציית ה-sign מה שהיא עושה היא מחזירה 1+ במידה והערך שנכנס אליה חיובי ואילו 1- במידה ונכנס אליה ערך שלילי.

לאחר הרצת התוכנית, הופתענו לגלות שקיבלנו תוצאות טובות עם אחוזי דיוק גבוהים. וניסינו להריץ ולרואת מה קורה אם נריץ על רשת בעלת מספר שכבות או ע"י פונקציות אקטיבציה שונות וראינו שעבור רשתות עם מספר שכבות נרבל יסית את אותם תוצאות וכן העדפנו להמשיך עם פרספטרון הואיל והוא יותר מהיר בהרצה שלו (עקב הפשטות שלו) ועבור פונקציות אקטיבציה שונות (כמו RELU,sigmoid וכו) קיבלנו אחוזי דיוק אפילו יותר נמוכים מה-sign, וכמובן שזמן ההרצה גם היה יותר גבוה.

[2] נציין, שבמהלך ה-dot product השתמשנו גם בערך bias שהשתנה במהלך ריצת התוכנית.

בס"ד

תוצאות:

- <u>עבור nn0:</u>
בהתחלה הרצנו על גודל אוכלוסייה של 100 פריטים וקיבלנו אחוזי דיוק גבוהים ומספר דורות נמוך, כלומר האלגוריתם הגנטי הגיע לאופטימליות לאחר מספר דורות יחסית קטן.
לאחר מכן הרצנו על גודל של 50 וקיבלנו אחוזי דיוק יותר נמוכים מההרצה הקודמת ואלגוריתם עצר לאחר יותר דורות, כדי להבין האם באמת יש פה מגמה של ירידה באחוז הדיוק וזמן הריצה כפונקציה של גודל אוכלוסייה - הרצנו גם בגודל של 20 פריטים ואכן קיבלנו את הנתונים הבאים:

גודל אוכלוסייה	מספר דורות[3]	אחוז דיוק על ה-test	אחוז דיוק על ה-train
100	151	0.99	0.9905
50	314	0.9796	0.9826
20	343	0.9674	0.9778

דבר זה מאמת את הנקודה שכבר הזכרנו בתרגיל הקודם: ככל שנעלה את מספר הפריטים באוכלוסייה כך המהירות שנגיע לאופטימליות תהיה יותר מהירה, וכמובן שזה גם משפר את האופטימליות עצמה. דבר זה קורה הואיל וגודל אוכלוסיה גדול יותר מאפשר לפתרונות מגוונים יותר להיות נוכחים באוכלוסייה, דבר אשר יכול לסייע לנו בלחקור מרחב חיפוש גדול יותר ולהימנע מהתכנסות מוקדמת. (ובעצם להיתקע בפתרון לא אופטימלי)

- <u>עבור nn1:</u>
בהוראות התרגיל נאמר שהחוקיות בקובץ nn1 היא יותר קשה ולכן ציפינו להגיע לאחוי דיוק נמוכים יותר ובמספר דורות גדול יותר, אך להפתעתינו קיבלנו את התוצאות הבאות:

גודל אוכלוסייה	מספר דורות[2]	אחוז דיוק על ה-test	אחוז דיוק על ה-train
100	123	0.996	1.0
50	215	0.991	0.997
20	310	0.989	0.991

לא רק שקיבלנו אחוזי דיוק יותר טובים גם הגענו אליהם במספר דורות יותר נמוך! דבר זה גרם לנו לחשוד שמא נוצר בלבול בכתיבת ההוראות, ושאכן החוקיות בnn1 היא יותר קלה. דבר זה קיבל תוקף לאחר שמצאנו את החוקיות שבשני הקבצים כפי שנתאר בהמשך הדוח. (ונראה שאכן החוקיות של nn1 יותר פשוטה משל nn0)
ראוי לציין שגם פה אנחנו רואים שככל שנעלה את מספר הפריטים באוכלוסייה כך האלגוריתם הגנטי יתכנס לפתרון אופטימלי בצורה יותר מהירה וכפי שהסברנו לעיל.

- לסיום, בשני הקבצים הגענו למסקנה שעל מנת למצוא את התבניות המדוייקות שלהם עלינו להריץ **פרספטרון** על גודל אוכלוסייה של **100** וככה נגיע לאחוזי דיוק גבוהים ובפרדיקציה טובה של התבנית.

[3] מספר הדורות המובא כאן הוא ממוצע הדורות שיצאו לנו בtrain וב-test עבור כל גודל אוכלוסייה

בס"ד

מציאת התבניות של הקבצים:

- <u>עבור nn0:</u>

לאחר שהרצנו את המודל שלנו 10 פעמים עם גודל אוכלוסייה של 100 שמנו לב שיש איזושהי חוקיות בערכי המשקולות וערך הbias. ועל מנת להסביר ניתן לדוגמא את אחת התוצאות שיצאה לנו:

```
nn0 > wnet0.txt
weights: [[0.10066443166350857], [0.11097735981797252], [0.11116641961053343], [0.09215403313342058], [0.12141431617603944], [0.10463072541450413], [0.11164330464248025],
biases: -0.808259543975432
model: [16, 1]
```

כפי שניתן לראות כל ערכי המשקולות הם פחות או יותר אותו הדבר (במקרה הזה כולם באזור 0.1) ואילו ה-bias תמיד יצא לנו **שלילי** עם ערך של <u>חצי</u> מסכום כל המשקולות (כי: $\frac{16*0.1}{2} = 0.8$) ולכן הואיל ואנו החלטנו להגדיר מספר כ-1 או 0 לפי הסימן שלו – האם הוא חיובי או שלילי (פונ' *sign*) אז נוכל לומר שהתבנית של *nn0* היא כל המספרים שחצי מהם שווים ל1 כלומר שיש לפחות 8 ספרות מתוך 16 שהן 1. כי נגיד לפי הדוגמא לעיל כאשר נחבר 8 ספרות שערכן 1 לאחר שנכפיל אותם במשקולות שלהם שזה 0.1 נקבל קצת יותר מ- 0.8 ואז שנחסיר מהם את ה*bias* השלילי נקבל מספר **חיובי** ולכן ניתן לו תווית של 1 ונכניס אותו כשייך לחוקיות, וזה נכון גם להפך – כל מחרוזת בינארית עם פחות מ8 ספרות של 1 נקבל בחיבור שלהם לאחר הכפלת המשקולות מספר שקטן מ-0.8 ואז נקבל ערך **שלילי** עם ה-*bias* ולכן הוא יוגדר כ-0 ולא יהיה שייך לחוקיות.

שמנו לב במהלך הריצות הנ"ל שאחוזי הדיוק שלנו על ה*test* הם לא 100 אחוז (כפי שיהיה לנו ב*nn1*) לכן הרצנו שוב על מנת למצוא את אותם מחרוזות שגויות. ושמנו לב שכל המחרוזות השגויות קיבלו ערך של 0.4 ומעלה בפונקציית האקטיבציה (והואיל והם חיוביים הם סומנו כ-1) דבר זה גרם לנו לחשוב שה-*data* שלנו הוא לא פריד לינארית ואכן יש 2 קווי גבול:

1) כל המחרוזות שיש בהם פחות מ8 אחדות לא שייכים לחוקיות וכל המחרוזות שיש להם מעל 8 ופחות מ12 כן שייכות לחוקיות
2) כל המחרוזות שיש להם מעל 12 אחדות לא שייכות לחוקיות. (כי אותם מחרוזות בעצם מקבלים ערך שגדול שווה ל-0.4 וכמו שראינו זה כל אותם המחרוזות שבטעות סיווגנו אותם כ-1 למרות שהם 0)

לכן לסיכום - התבנית של *nn0* היא כל המחרוזות שיש להם מספר אחדות שבין 8 ל-12.

- <u>עבור nn1:</u>

גם פה הרצנו את המודל 10 פעמים וקיבלנו תוצאות די שוות, ושוב ניתן פה דוגמת הרצה אחת:

```
nn1 > wnet1.txt
weights: [[-0.11444460068685212], [-0.11883880391910295], [-0.11949095063288927], [-0.13105376406020203], [-0.1127123399070965], [-0.11614571664568354], [-0.1146595542709
biases: 0.8595871653657795
model: [16, 1]
```

קיבלנו בדיוק הפוך מהמודל הקודם! ערכי משקולות שווים בערכם אך **שליליים** (כולם באזור ה: *-0.1*) ודווקא ה-*bias* **חיובי** ושווה למחצית סכום ערכי המשקולות. דבר זה גרם לנו לחשוב שהתבנית ב-*nn1* היא כל המחרוזות שמספר האחדות בהם **קטן** ממש מ-8. שהרי לאותם מחרוזות נקבל שסכום המשקולות (לאחר מכפלתם ב-*0.1*) יהיה פחות מ: *-0.8* ואז כאשר נוסיף להם את ה-*bias* החיובי נקבל ערך שהוא חיובי. כלומר סה"כ נקבל מחרוזת ששייכת לחוקיות!

והואיל וקיבלנו בכל 10 הריצות <u>100</u> אחוזי דיוק, שום דבר לא גרם לנו לפקפוק בהבנת התבנית הנ"ל ולמחשבה שיש כאן *data* שהוא לא פריד לינארית כמו ב-*nn0*.

לכן לסיכום - התבנית של *nn1* היא כל המחרוזות שיש להם מספר אחדות לכל היותר 7!