אמיר וולפנזון 300339785, אור שריר 201314796 , אדם ברטאש 206321440, ישי שפירא 203016217

פרויקט סקרנות - חלק ב'

למידה וסקרנות באמצעות רשת נוירונים מלאכותית

הקדמה (חוב קטן מחלק א'):

הצגת כלל הפיצ'רים בדאטה ותיאורם:

פיצ'ר	תיאור	תחום ערכים
xG	מטריקה לגולים צפויים שהקבוצה תכבוש במשחק, כימות סטטיסטי בין מספר מצבי ההבקעה שמוערך שתממש (כלומר יבשילו לכיבוש) לעומת כלל מצבי ההבקעה המשוערים של הקבוצה	0-6.63 תחום רציף
xGA	מטריקה לגולים צפויים שהקבוצה תספוג במשחק, כימות סטטיסטי בין ממוצע מצבי ההבקעה של יריבותיה = מימוש לעומת כלל מצבי ההבקעה של יריבותיה	0-6.63 תחום רציף
npxG	גולים צפויים שהקבוצה תכבוש במשחק, לא כולל פנדלים ושערים עצמיים	0-6.61 תחום רציף
npxGA	צפי לספיגה לקבוצה במשחק, לא כולל פנדלים ושערים עצמיים	0-6.61 תחום רציף
npxGD	ההפרש בין צפי להבקעות לעומת ספיגות, לא כולל פנדלים ושערים עצמיים	(-6) - 6 תחום רציף
ppda_att	מספר המסירות ההתקפיות המוצלחות* שנעשו ע"י הקבוצה	0-764 תחום בדיד
ppda_def	מספר המסירות ההגנתיות המוצלחות (בין שחקני ההגנה או מסירות אחורה) שנעשו ע"י הקבוצה	0-65 תחום בדיד
oppda_att	מספר המסירות ההתקפיות המוצלחות* שנעשו ע"י הקבוצה היריבה	0-764 תחום בדיד
oppda_def	מספר המסירות ההגנתיות המוצלחות* שנעשו ע"י הקבוצה היריבה	0-65 תחום בדיד
ppda_coef	עוצמת הלחץ שהקבוצה מפעילה בחצי המגרש של היריבה. כלומר, היחס בין מס' המסירות המוצלחות* שהיא מבצעת בחצי המגרש של היריבה לבין מס' הפעולות ההגנתיות של היריבה - חילוץ, פאול, חטיפת הכדור וכו'	0-152 תחום רציף
oppda_coef	עוצמת הלחץ שהקבוצה היריבה מפעילה בחצי המגרש של הקבוצה השניה.כלומר, היחס בין מס' המסירות המוצלחות* שהיריבה מבצעת בחצי המגרש של הקבוצה השניה לבין מס' הפעולות ההגנתיות של הקבוצה השניה	0-152 תחום רציף
deep	מסירות מוצלחות* שהושלמו במרחק של פחות מ-18 מטר משער היריבה (מסירות בתוך רחבת שער היריבה)	0-42 תחום בדיד
deep_allowed	מסירות מוצלחות* שהיריבה השלימה במרחק של פחות מ-18 מטר משער הקבוצה השניה	0-42 תחום בדיד

*מסירה מוצלחת - מוגדרת כמוצלחת כאשר הכדור הועבר בין שני שחקנים באותה הקבוצה, ללא קטיעה (חטיפה ע"י שחקן יריב, עבירה שבגינה המשחק הופסק, נבדל או שהכדור יצא מתחומי המגרש)

אמיר וולפנזון 300339785, אור שריר 201314796 , אדם ברטאש 206321440, ישי שפירא 203016217

פיצ'רים בהם בחרנו שלא להשתמש (הורדנו בתהליך "ניקוי" הדאטה):

פיצ'ר	תיאור	תחום ערכים	סיבת הניפוי
xpts	נקודות צפויות לקבוצה, למשחק	0-3 תחום רציף	מגלה למודל בהסתברות גבוהה מדי (>90% מהמקרים) מה תהיה התוצאה בפועל. החלטנו כי הדבר פוגע בלמידה
xG_diff	ההפרש בין גולים שהובקעו בפועל ומספר הגולים הצפוי, לקבוצה	(-5.9) - 5.9 תחום רציף	הסיבה שבחרנו להוריד את הפיצ'רים האלה היא משום שהם "מגלים" למודל מה התרחש בהסתברות = 1 (לפי ההפרשים ניתן לדעת בדיוק מה קרה במשחק- ניסינו וזה נתן לנו מההתחלה val_acc=1
xGA_diff	ההפרש בין הגולים שהקבוצה ספגה בפועל במשחק לעומת הצפי לספיגה של הקבוצה במשחק	(-5.9) - 5.9 תחום רציף	
Xpts_diff	ההפרש בין הנקודות שקיבלה הקבוצה בפועל ומספר הנקודות הצפוי שתקבל	(-2.9) - 2.9 תחום רציף	
Scored	כמה הקבוצה הבקיעה	0-10 תחום בדיד	הסיבה שבחרנו להוריד את הפיצ'רים האלה היא משום שהם "מגלים" למודל מה התרחש בהסתברות = 1
missed	כמה הקבוצה היריבה הבקיעה	0-10 תחום בדיד	
result	התוצאה הסופית-האם הקבוצה ניצחה או הפסידה	{W,D,L} קטגוריאלי	
wins	האם בפועל הקבוצה ניצחה?	{0,1} תחום בינארי	
draws	האם בפועל הקבוצה סיימה בתיקו?	{0,1} תחום בינארי	
loses	האם בפועל הקבוצה הפסידה?	{0,1} תחום בינארי	
Year	שנת המשחק הספציפי (שנת תיעוד השורה בדאטה)	2014-2019 תחום בדיד	לא רלוונטי לעולם אותו אנו חוקרים
League	שם הליגה בה המשחק התרחש	{בונדסליגה (גרמניה), פרמיירליג (אנגליה), לה-ליגה (ספרד), סרייה-א (איטליה), ליגה1 (צרפת), ארפיאל (רוסיה) } קטגוריאלי	משום שבעולם הנחקר קבוצות יתחרו תמיד מול קבוצות באותה הליגה- להערכתנו, לא רלוונטי לעולם הספציפי אותו אנו חוקרים
h_a	Home_Away - האם הקבוצה מארחת בביתה או מתארחת	בית או חוץ קטגוריאלי	לא רלוונטי לעולם אותו אנו חוקרים

team	שם הקבוצה	קטגוריאלי	לא רלוונטי לעולם אותו אנו חוקרים

4. פירמול הבעיה בעזרת רשת נוירונים

בהמשך למה שכתבנו בחלק א', נתונים לנו קבוצות מ-6 ליגות באירופה (ליגה ספרדית, ליגה אנגלית, ליגה איטלקית, ליגה צרפתית, ליגה גרמנית וליגה הרוסית), והתוצאות של הקבוצות מהליגות האלה בין השנים 2014 ל-2019.

מה שאנו רוצים ללמוד הוא האם קבוצה, במשחק מסוים, תנצח, תסיים בתיקו או תפסיד, הפלט מהרשת יהיה ההסתברות לכל אחת מהאפשרויות כאשר ניקח את האפשרות עם ההסתברות הגבוהה ביותר כחיזוי שלנו, לדוגמא, עבור רקורד ספציפי של נתונים הרשת תפלוט שההסתברויות לניצחון, תיקו והפסד הן 50%, 35% ו-15% בהתאמה אז החיזוי שלנו הוא ניצחון.

את הדאטה שלנו חילקנו באופן הבא: 15% ישמש לטסט, 15% ישמש לווליידישן, והשאר לסט אימון.

נירמלנו את כל הפיצ'רים לטווח ערכים בין 0 ל-1, הסיבה שעשינו זאת היא כדי שהמשקלים שיש בניורונים יהיו פחות או יותר באותו סדר גודל, אם לפיצ'ר מסוים טווח הערכים הוא באלפים ולפיצ'ר אחר הוא במספרים עשרוניים קטנים אז המשקלים יהיו בסקלות שונות ואת זה רצינו למנוע.

<u>הגדרת מבנה הרשת ואלגוריתם הלמידה (ההתחלתי):</u>

השתמשנו ברשת ANN מסוג sequential בעלת קלט, 3 שכבות מוסתרות ושכבת פלט, בחרנו במודל זה בגלל שקל מאוד לשחק עם מבנה הרשת, דבר שיעזור לנו בשלב ניתוח הרגישות.

שכבות 1-3: שכבות מסוג Dense (כל הנוירונים מחוברים לכל הנוירונים בשכבה הבאה), בכל שכבה יש 16 נוירונים.

ישנם שני מקדמי רגולריזציה, ערך המקדם הראשון הוא 0.0001 והשני הוא 0.001, הגדרנו מקדמי רגולריזציה בשביל להימנע מ overfitting, אבל נתנו להם ערכים קטנים כי המטרה שלנו היא ללמוד ולא נרצה שהם יפגעו התהליך זה.

פונקציית האקטיבציה בכל שכבה היא relu שלומדת את הסיכוי לנצח/להפסיד מכל תוצאה.

שכבת הפלט: שכבה בעלת 3 נוירונים אשר ממירה את תחום ערכי הנוירונים בשכבה הקודמת לערכי הסתברות משלימים בין 0 ל-1, כאן הלמידה תתבצע (לנצח, להפסיד או לסיים בתיקו, על סמך תוצאות התיקו), השתמשנו בפונקציית האקטיבציה softmax בשביל להמיר את הערכים מהשכבה הקודמת לערכי ההסתברויות לכל תוצאה - ניצחון, תיקו או הפסד.

שיעור הלמידה: נקבע ע"י שיטת האופטימייזייר Adagrad. זוהי שיטת gradient descent. שיטה זו מתאימה את שיעור הלמידה בהתאם לתדירות הפרמטרים, בניתוח הרגישות נבדוק אופטימייזרים נוספים.

פונקציית ה-**loss**: מכיוון שהפרמטר אותו אנו מנסים לחזות הוא קטגוריאלי השתמשנו בפונקציית loss categorical_crossentropy.

בנוסף לשינוי בפונקציית ה-loss רצינו גם לראות את את הדיוק של המודל אז השתמשנו במדד CategoricalAccuracy.

5. <u>תכנות הבעיה והפתרון:</u>

נעשה בקובץ jupyter notebook המצורף.

6. <u>ניתוח פרמטרי של הרשת:</u>

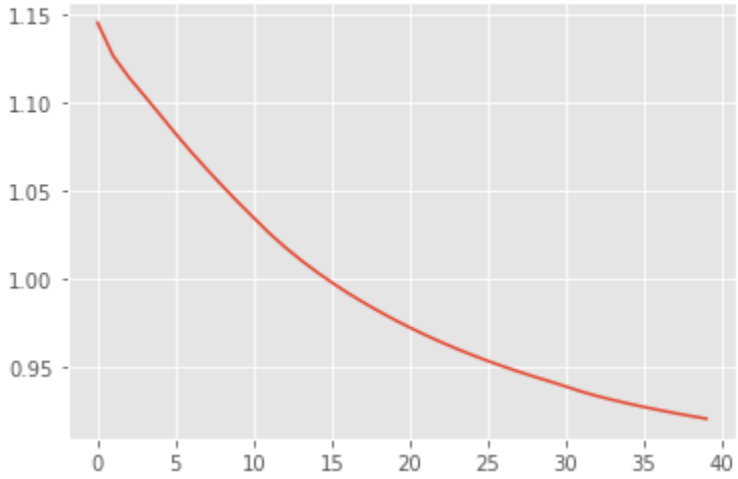
כמו שנאמר בסעיפים קודמים, איתחלנו את הרשת שלנו לרוץ עם 3 שכבות בעלות 16 נוירונים ופונקציית אקטיבציה relu ושכבת פלט בעלת 3 נוירונים עם אקטיבציה softmax כאשר האופטימייזר הוא Adagrad. ניתן לראות כי שגיאת ה MSE של רשת זו לפי נתוני המבחן (ורק בחישוב השגיאה השתמשנו בהם) היא 1,668.

```
In [16]: prediction_matrix = first_run_model.predict(x=x_test, batch_size=32, verbose=2)
         prediction_result = np.argmax(prediction_matrix, axis=-1)
         first_run_error = get_error(y_test, prediction_result)
         print('The error of the first model is: {}'.format(first_run_error))

         3687/3687 - 0s
         The error of the first model is: 1668.0
```

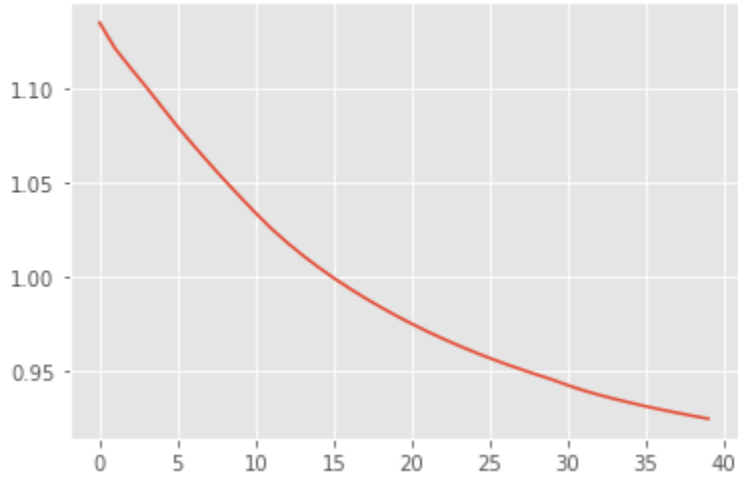
ה-loss שלנו על נתוני האימון יורד לאורך ה-epochs בצורה מונוטונית בקצב יורד.

```
In [17]: plt.plot(first_run_history.history['loss'])
Out[17]: [<matplotlib.lines.Line2D at 0x1f410eee9c8>]
```



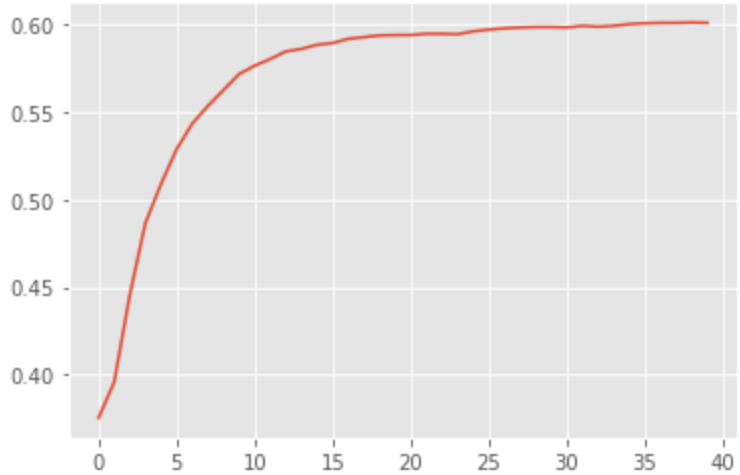
ניתן לראות שגם ה-validation loss שלנו יורד לאורך ה-epochs בצורה מונוטונית בקצב יורד .

```
In [18]: plt.plot(first_run_history.history['val_loss'])
Out[18]: [<matplotlib.lines.Line2D at 0x1f410f69a08>]
```



הדיוק של נתוני הוולידציה עולה בצורה מונוטונית למדי ומתכנס לאזור ה-60%.

```
In [19]: plt.plot(first_run_history.history['val_categorical_accuracy'])
Out[19]: [<matplotlib.lines.Line2D at 0x1f410f0b1c8>]
```



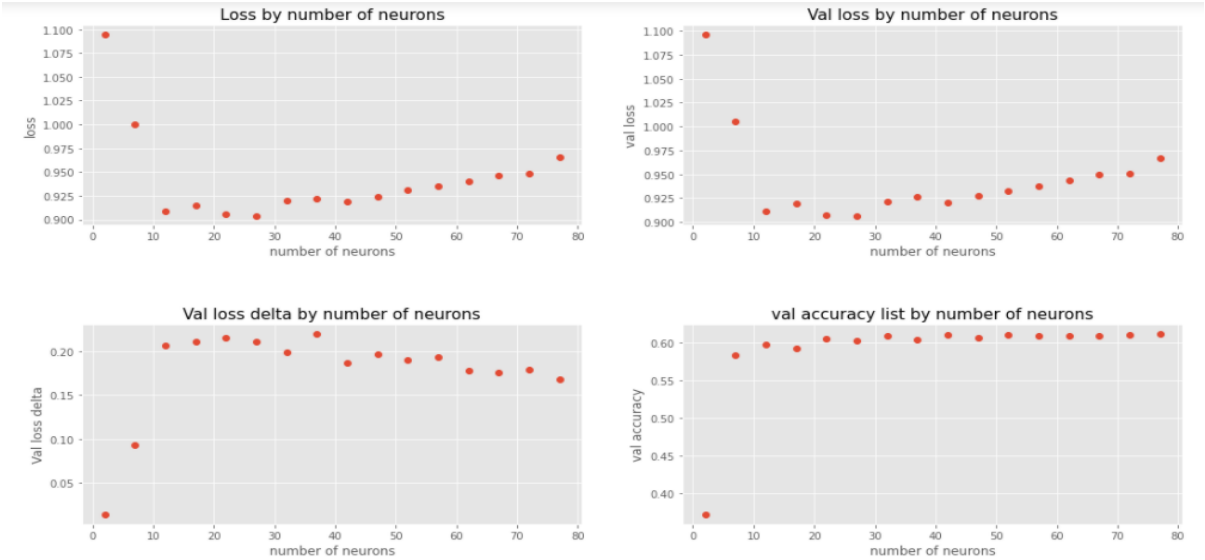
כעת לאחר קבלת רשת ראשונית עם תוצאות ניסינו בעצם לנתח את רגישות הרשת שלנו, בחרנו להתמקד בשינוי הפרמטרים הבאים:

- מספר הנוירונים בשכבה
- מספר השכבות
- שינוי פונקציית האקטיבציה
- שינוי האופטימייזר

ניתחנו את ההשפעה על ה Loss של נתוני האימון והווליד‌ציה, על הדיוק של נתוני האימון ועל השינוי ב Loss שקיבלנו (ערך הval loss ההתחלתי פחות ערך הval loss הסופי) על נתוני הווליד‌ציה.

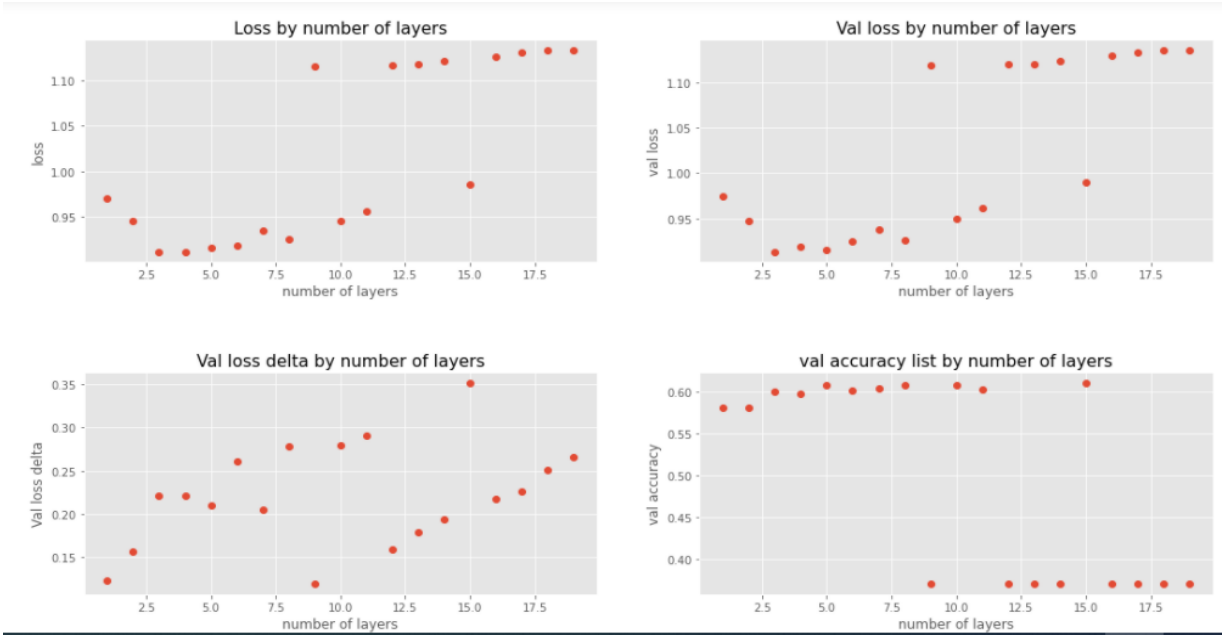
מספר הנוירונים בשכבה

ניתן לראות לגבי מספר הנוירונים בשכבות שהחל מכמות נוירונים של 12 הרשת שלנו הופכת להיות די אדישה (מבחינת כל המדדים) לכל הגדלה ואם נגדיל לדוגמא פי 8 את כמות הנוירונים (מ 10 ל 80) נקבל שיפור ב Val Loss של 7%. כמו כן יש לציין שעד לכמות הזו של 12 נוירונים הרשת שלנו אכן רגישה לשינויים.



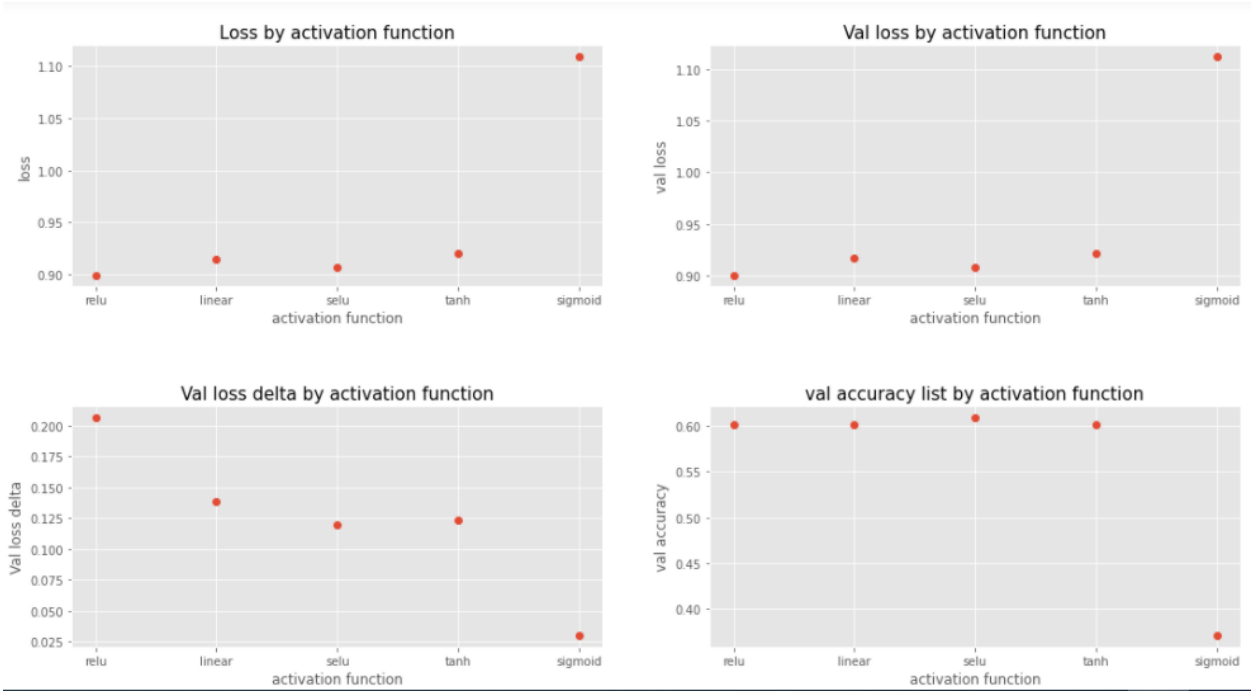
מספר השכבות

ישנו טווח של מספר שכבות שהרשת שלנו תהיה פחות או יותר אדישה אליו מבחינת רוב המדדים והוא 3-10, ניתן לראות שהרשת שלנו רגישה לשינויים מחוץ לטווח זה ומעבר שכזה יגרום לשינויים יחסית גדולים במדדי הרשת שלנו.



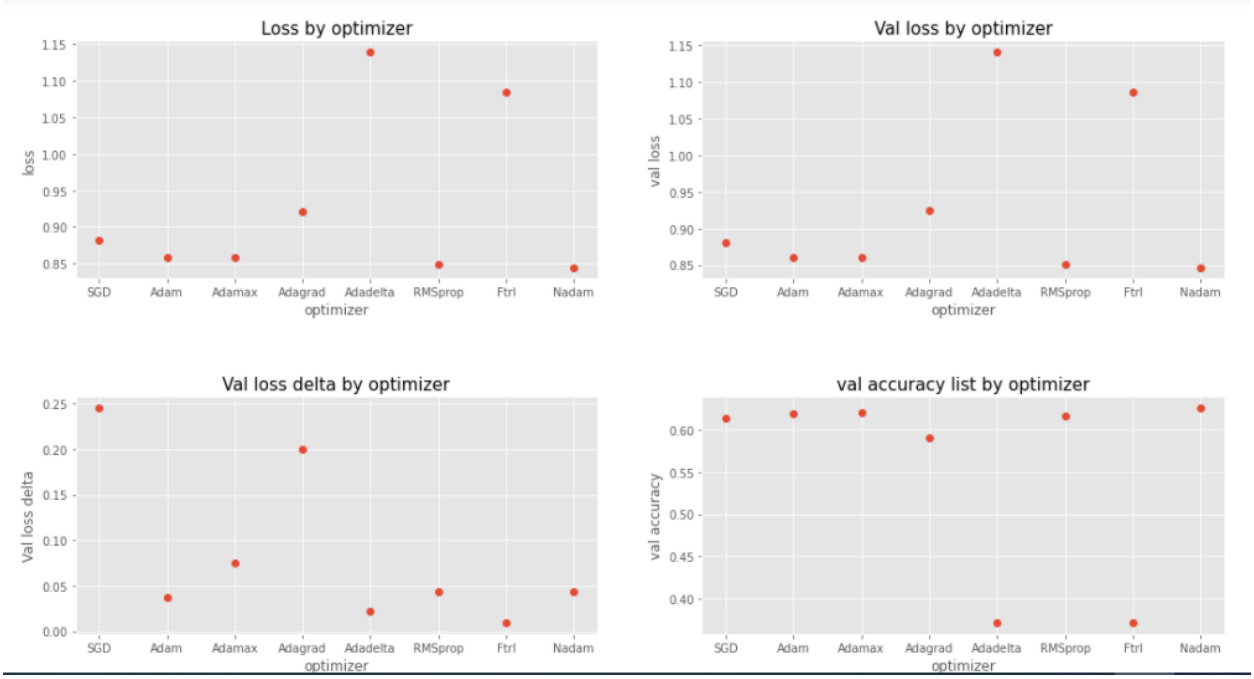
פונקציית האקטיבציה

בחנו את רגישות הרשת שלנו לפונקציית האקטיבציה ע"י הרצתה עם פונקציות האקטיבציה relu,linear, selu, tanh, sigmoid וניתן לראות כי הרשת שלנו לא רגישה מדי לשינויי פונקציות האקטיבציה מלבד רגישות לשימוש בפונקציות אקטיבציה sigmoid אשר תשפיע לרעה בצורה ניכרת על מדדי הרשת שנבחנו. אפשר גם לראות val loss delta - ההפרש בין הערך הראשון לאחרון בלמידה של ה val loss יש יתרון קטן לפונקציית relu.



האופטימייזר

ניתן לראות שאחד הפרמטרים אליהם הרשת שלנו היתה הכי רגישה הינו האופטימייזר אשר שינויים בו גורמים לשינויים משמעותיים בתוצאות הרשת. בנוסף קל להבחין כי RMSprop ו Adamax מביאים לתוצאות מיטביות (דיוק גבוה תוך Loss בנתוני הוולידציה נמוך).



Random Search

כעת לשם אופטימיזציה הרצנו random search עם 30 איטרציות שבכל פעם בוחר באופן אקראי את האופטימייזר, פונקציית האקטיבציה, מספר שכבות וכמות הנוירונים בכל שכבה ומריץ למשך 40 epochs רשת נוירונים עם הנתונים הללו שה random search בחר לה ושכבת פלט בעלת פונקציית אקטיבציה 'softmax' עם 3 נוירונים (ככמות ערכי שדה המטרה שלנו).

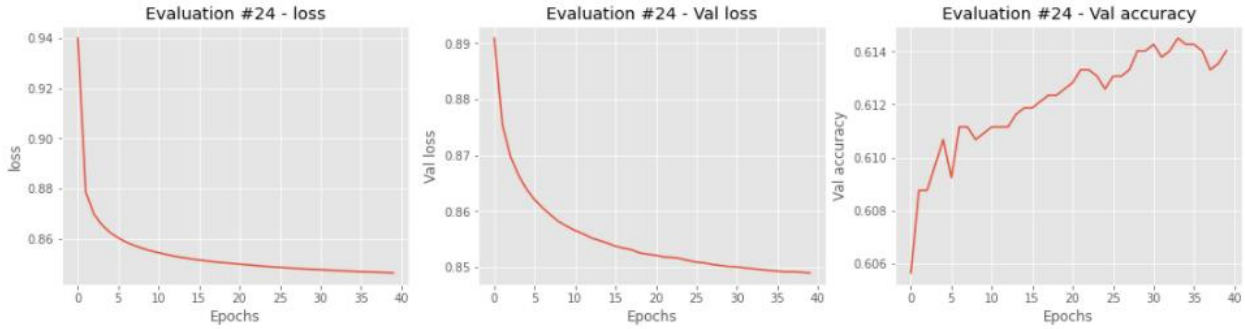
הסיבה שרצינו לבחון את המודל גם על ידי random search למרות שקשה ככה להצביע על מה גורם למה היא שלפעמים לשינוי כל הפרמטרים בבת אחת ישנה השפעה שונה ואולי שילוב של מספר שינויים במקביל יביאו לנו את המודל האופטימלי.

ניתחנו את ה accuracy, train/validation loss שהיו בכל אחת מהאיטרציות וראינו כי נתוני הרשת שהובילו ל val loss ושגיאת MSE מינימליים הינם - רשת בעלת שכבה 1 אשר לה 39 נוירונים, עם פונקציית אקטיבציה selu ואופטימייזר RMSprop (הרצה מספר 24 ב random search).

בצילום המסך שמצורף למטה ניתן לראות השוואה בין כל 30 הרצות ה-random search אשר ממויינות לפי ה val loss.

	Evaluation	# of layers	# of units per layer	Activation function	Optimizer	Val loss	Val loss Delta	MSE	# of params	BIC	AIC
23	24	1	39	selu	RMSprop	0.848908	0.0420387	5666.5	666	150924	145780
22	23	1	23	tanh	Adamax	0.851393	0.141645	6155	394	149661	146618
11	12	2	26	linear	RMSprop	0.853623	0.0554514	5776	1147	155921	147062
20	21	12	48	selu	Adam	0.854794	0.699716	6160	26691	405387	199226
24	25	6	6	relu	RMSprop	0.860408	0.0436813	5979.5	315	148410	145976
10	11	6	39	linear	RMSprop	0.864038	0.157862	6820.5	8466	229869	164478
6	7	4	22	linear	Adamax	0.866443	0.104116	6222	1895	164438	149801
4	5	6	21	linear	RMSprop	0.868082	0.119987	5691	2670	170483	149860
15	16	9	5	tanh	Adamax	0.873684	0.0880409	6931.5	328	151005	148472
2	3	11	30	linear	RMSprop	0.874477	0.201438	6848.5	9813	243036	167240
19	20	15	43	linear	RMSprop	0.883409	0.232199	6893.5	27222	412431	202168
12	13	14	28	tanh	Adamax	0.886453	0.4237	6913.5	11035	255077	169842
28	29	16	26	tanh	Adamax	0.889992	0.440092	6899.5	10975	254459	169688
3	4	4	27	tanh	SGD	0.915632	0.116868	5985.5	2730	171910	150823
13	14	2	46	linear	Adagrad	0.930939	0.108404	6726.5	2947	175971	153208
9	10	7	19	relu	SGD	0.933356	0.291277	5968.5	2606	170656	150528
5	6	14	32	relu	Adagrad	0.994521	0.53373	6289.5	14275	285001	174741
29	30	16	21	selu	SGD	1.04397	0.305874	5582.5	7290	215086	158778
18	19	8	39	tanh	SGD	1.04601	0.27352	5998	11586	258060	168570

נציג כעת את הרצה מספר 24 ב random search אשר הביאה לשגיאה ול val loss הנמוכות ביותר (כמו שאמרנו קודם, מדבר ברשת בעלת שכבה 1 אשר לה 39 נוירונים, עם פונקציית אקטיבציה selu ואופטימייזר RMSprop)-



ניתן לראות שהאופטימיזציה אכן הביאה אותנו לדיוק גבוה יותר מ 60% בערכי הווליידציה אשר קיבלנו מהרשת ההתחלתית ול val loss נמוך יותר מהרשת ההתחלתית שבה הגענו לקצת מתחת ל 0.95 ופה אנו מגיעים ל 0.85.

כעת לאחר שבחרנו את הרשת האופטימלית שלנו אשר הוצגה, אימנו אותה ובדקנו אותה על נתוני הווליידציה נציג את הביצועים שלה על נתוני המבחן.

השגיאה שלנו במודל האופטימלי על נתוני המבחן (ורק בחישוב השגיאה השתמשנו בהם) הינה 1,257.5 וזוהי שגיאה נמוכה יותר מזו שהושגה על נתוני הוולידציה במודל הראשוני.

```
In [75]: final_prediction_matrix = final_run_model.predict(x=x_test, batch_size=32, verbose=2)
         final_prediction_result = np.argmax(final_prediction_matrix, axis=-1)
         final_model_error = get_error(y_test, final_prediction_result)
         print('The error of the final model is: {}'.format(final_model_error))

         3687/3687 - 0s
         The error of the final model is: 1257.5
```

<u>מסקנות</u>

- עלה מניתוח הרגישות כי פרמטר האופטמייזר הינו הפרמטר אשר הרשת הכי רגישה אליו.
- עוד עלה מניתוח הרגישות כי כמות הנוירונים בשכבות ומספר השכבות יחסית לא משפיעים על הרשת.
- מניתוח הרגישות קיבלנו טווחי ערכים לפרמטרים אשר הרשת רגישה יותר.פחות אליהם ולמדנו מכך המון אך כאשר ניגשנו לאופטימיזציה של המודל נכחנו לראות שדווקא קומבינציות של שינויים (ולא שינויים בצורה של ניתוח רגישות שכל פעם מקבעים את כל הפרמטרים ורק משנים פרמטר מסויים) יכולים להשפיע על הרשת מאוד ואכן ראינו שבניתוח הרגישות היה נראה שטווח ערכים מסויים הוא אופטימלי בזמן שבתהליך האופטימיזציה של ה random search קיבלנו ערכים מעט שונים אך זה מוסבר בכך שהקומבינציה של הערכים האלו היא זו שמשפיעה בעצם.
- נשים לב כי לא נוצר מצב של overfitting לאורך ה epochs.

<u>שימוש ב-Features Selection</u>:

הבהרה: חלק זה מופיע לאחר הרצת המודל האופטימלי משום שבסופו של התהליך בחרנו שלא להשתמש בבחירת פיצ'רים, כלומר למודל הסופי שבנינו מוזנים כל 13 הפיצ'רים שתוארו לעיל.

בנוסף לבחירת מבנה הרשת הרצוי ואופטימיזציה שלו כפי שתיארנו בחלק הקודם, ובנוסף לכך שכבר הורדנו פיצ'רים שרמזו את התוצאה בפועל ואחרים (ראה טבלה בעמ' 2) - בדקנו האם ניתן לשפר עוד את המודל שלנו וזאת ע"י הרצת המודל עם מספר מצומצם ונבחר של פיצ'רים.

החשיבות של בחירת פיצ'רים באה לידי ביטוי בעיקר בדאטה סטים בהם יש מספר גדול מאוד של משתנים (High dimensional dataset), מכיוון שהמימד הגבוה של הפיצ'רים מעלה משמעותית את זמן אימון המודל, וכשמדובר ב-NN, הדבר יכול לגרום ליצירת מודל מסובך מאוד שיוביל ל-Overfitting.

שיטה 1- ניפוי מושכל של פיצ'רים:

בהסתכלות ראשונה על רשימת הפיצ'רים, חיפשנו את אותם פיצ'רים מיותרים, כלומר אלה שהם בהכרח איזושהי "הרחבה" לפיצ'רים חיוניים אחרים. ההנחה שלנו היא שפיצ'רים יתירים אלו בהכרח לא יתרמו משמעותית ללמידה. מתוך 13 הפיצ'רים בחרנו באלו:

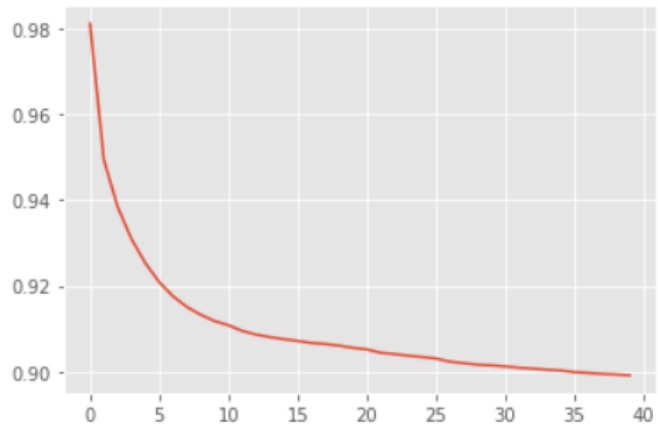
נשארו	נופו	הסבר
xG, xGA	**nxG , nxGA, npxGD**	המשתנים xG ו- xGA כבר מספקים מידע מדוייק יותר לכמות הכיבוש והספיגה של הקבוצה משום שהם כן סופרים פנדלים ושערים עצמיים, וכנראה ניתן ללמוד מכך יותר מאשר אם לא היינו סופרים פנדלים ושערים עצמיים. ובנוסף npxGD אמור להיות ההפרש בין הבקעות לספיגות צפויות, אותו ניתן לחשב על סמך שני המשתנים המשתנים nxG ו- nxGA (ראה טבלת פיצ'רים בעמ' 1 למידע נוסף)
ppda_att **ppda_def** **oppda_att** **oppda_def**	**ppda_coef,** **oppda_coef**	הפיצ'רים שבחרנו לנפות הינם "הרחבה" לפיצ'רים שנשארו כי הם למעשה נוסחה המורכבת ממשתנים אלו. היחס בין מס' המסירות המוצלחות של הקבוצה/היריבה בחצי המגרש של השניה לבין מס' הפעולות ההגנתיות של השניה (ראה טבלת פיצ'רים בעמ' 1 למידע נוסף)
,deep **deep allowed**		בחרנו להשאיר פיצ'רים אלה משום והמידע האצור בהם אינו נמצא באף משתנה אחר, ולכן בוודאות לא ניתן לוותר עליהם

נריץ את מודל ה-NN הסופי עם המשתנים הנבחרים לפי שיטה זו ונקבל כי שגיאת ה MSE של רשת זו לפי נתוני המבחן (ורק בחישוב השגיאה השתמשנו בהם) היא 1,548.5.

ניתן לראות גם מבחינת הפרש ה-Val_loss קיבלנו הפרש של 0.9811-0.8992=0.0819:

```
plt.plot(final_fsel_manual_run_history.history['val_loss'])
```

```
[<matplotlib.lines.Line2D at 0x1adefa3d848>]
```



אומנם הורדנו את מספר הפיצ'רים בכ-40% (מ-13 ל-8) אך ניתן לראות כי קיבלנו שגיאה גבוהה יותר בכ-23% מהמודל הסופי ללא בחירת הפיצ'רים (1548 לעומת 1,257 במודל הסופי). אומנם ציפינו שכאשר נוריד את מספר הפיצ'רים הדבר כן ייפגע במידה מסויימת בלמידהת אך לדעתנו- מכיוון שמראש צמצמנו את מס' הפיצרים מימד הדאטה אינו מספיק גדול, כלומר אינו "High dimensional dataset" מספיק כדי להצדיק פגיעה גבוהה ביכולת הלמידה של המודל. לכן, במקרה זה נעדיף שלא להשתמש בשיטה זו.

שיטה 2- **Filtering**: כמו בשיטה הקודמת, מתחילים כשברשותנו כל 13 הפיצ'רים. את הורדת הפיצ'רים נעשה בשיטת פילטרינג. בשיטה זו מבצעים מבחן סטטיסטי על הקורולציה בין המשתנים לבין ה- (Output target). בחרנו במבחן סטטיסטי חי-בריבוע משום שזה מבחן שמתאים למצב בו המשתנים רציפים והקלאסיפיקציה קטגורית. המבחן נותן "ציון" לכל פיצ'ר.

נריץ את השיטה ונקבל:

```python
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2

#Performing filter feature selection

# Feature extraction
test_chi2 = SelectKBest(score_func=chi2)
fit_chi2 = test_chi2.fit(x_train_arr, y_train_arr)
print(y_train_arr.shape)

# Summarize scores
np.set_printoptions(precision=0)
print(fit_chi2.scores_)
print(set(data))
```

```
(16714, 3)
[299. 308. 258. 268.  53.  52. 109.   5.  10.   1.   6.  10.   2.]
{'npxGD', 'oppda_att', 'npxGA', 'oppda_coef', 'npxG', 'deep', 'xG', 'xGA', 'ppda_def', 'oppda_def', 'deep_allowed', 'ppda_coe
f', 'ppda_att'}
```

ניתן לראות כי ניתן לנפות את המשתנים : xGA, ppda_att, npxG, xG, ppda_def ו- deep מכיוון שקיבלו את הציונים הנמוכים ביותר במבחן שלנו.

נריץ את מודל ה-NN הסופי עם המשתנים הנבחרים לפי שיטה זו ונקבל כי שגיאת ה MSE של רשת זו לפי נתוני המבחן (ורק בחישוב השגיאה השתמשנו בהם) היא 1,507.

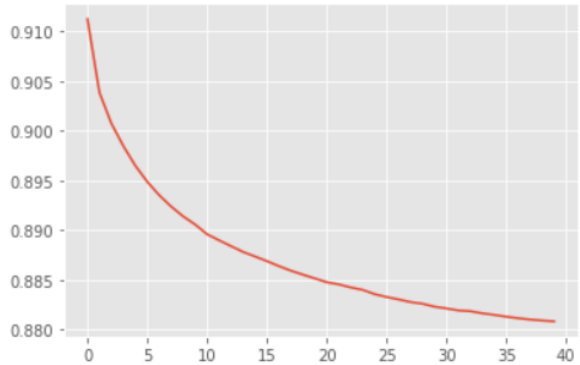
```python
prediction_matrix_filter = final_fsel_filter_run_model.predict(x=x_test_filter, batch_size=32, verbose=2)
prediction_result_filter = np.argmax(prediction_matrix_filter, axis=-1)
fsel_filter_error = get_error(y_test, prediction_result_filter)
print('The error using the features selection using "filter"  model is: {}'.format(fsel_filter_error))
```

```
3687/3687 - 0s
The error using the features selection using "filter"  model is: 1507.0
```

ניתן לראות גם כי מבחינת הפרש ה-Val_loss קיבלנו הפרש של 0.9112-0.8808=0.0304:

```python
plt.plot(final_fsel_filter_run_history.history['val_loss'])
```

```
[<matplotlib.lines.Line2D at 0x1ade4879348>]
```



שוב, אומנם הורדנו את מספר הפיצ'רים בכ-47% (מ-13 ל-7) אך ניתן לראות כי קיבלנו שגיאה גבוהה יותר בכ-20% מהמודל הסופי ללא בחירת הפיצ'רים (1,507 לעומת 1,257 במודל הסופי). וזוהי, לטעמנו, עדיין פגיעה גבוהה מדי בדיוק וביכולת למידת המודל בכדי להצדיק את הורדת הפיצ'רים.

שיטה 3- **Wrapper:** בגישה אחרת, נשתמש בשיטה מסוג "wrapper" כדי לחקור את אפשרויות השילובים של הפיצ'רים הנותרים:

בשיטת Wrapper (יצירת עטיפות של פיצ'רים) משתמשים במודל ML יחיד (כאן בחרנו שרירותית להשתמש ברגרסיה לוגיסטית - רוב המודלים דומים בתוצריהם) ומשתמשים בפלט שלו כקריטריון ההערכה. אנחנו השתמשנו במימוש השיטה ע"י תהליך (Recursive Feature Elimination RFE) בו מבצעים מעין Greed-Search למציאת סט הפיצ'רים הטוב ביותר. הוא עושה זאת ע"י יצירה איטרטיבית של מודל הרגרסיה וזיהוי הפיצ'ר הטוב או הגרוע ביותר של אותה איטרציה, ואז מוציא אותם מהרשימה ומריץ את מודל הרגרסיה וחוזר חלילה. לבסוף, בשיטה זו מדרגים את הפיצ'רים לפי סדר הוצאתם (Kהטובים ביותר שנבחרו ראשונים מקבל את הדירוג הגבוה ביותר (1) והגרועים ביותר את הדירוג הנמוך ביותר (לפי מספר האיטרציות). למודל יש להזין את מספר הפיצ'רים שנרצה להוציא בכל איטרציה- במקרה שלנו, לאחר ניסוי וטעייה בחרנו ב-K=6 (מחצית ממספר הפיצ'רים הכולל) . מכיוון שבדאטה שלנו השארנו 13 פיצ'רים, ה-Greed-search יקח לכל היותר 2^{13} איטרציות.

נריץ את השיטה ונקבל:

```python
from sklearn.feature_selection import RFE
from sklearn.linear_model import LogisticRegression

# Feature extraction
model_wrap = LogisticRegression()
rfe = RFE(model_wrap)
# Creating 1-D y Array
y_RFE=np.zeros([len(y_train_arr)])
for i in range(len(y_train_arr)):
    if(y_train_arr[i,1]==1):
        y_RFE[i]=1
    elif(y_train_arr[i,2]==1):
        y_RFE[i]=2
    else:
        y_RFE[i]=3
#Trainig the Wrap Model and prinint results
fit_wrap = rfe.fit(x_train_arr, y_RFE)
print("Num Features: %s" % (fit_wrap.n_features_))
print("Selected Features: %s" % (fit_wrap.support_))
print("Feature Ranking: %s" % (fit_wrap.ranking_))
print(set(data))
```

```
Num Features: 6
Selected Features: [ True  True False False False  True  True  True False False  True False
 False]
Feature Ranking: [1 1 3 2 4 1 1 1 5 7 1 6 8]
{'npxGD', 'oppda_att', 'npxGA', 'oppda_coef', 'npxG', 'deep', 'xG', 'xGA', 'ppda_def', 'oppda_def', 'deep_allowed', 'ppda_coe
f', 'ppda_att'}
```

כלומר על פי המפתח שהשיטה החזירה, הפיצ'רים שקיבלו את הציון הגבוה ביותר (Ranking), הם:
oppda_att, deep, xG, xGA, deep_allowed ו- npxGD

נריץ את מודל ה-NN הסופי עם המשתנים הנבחרים לפי שיטה זו ונקבל כי שגיאת ה MSE של רשת זו לפי נתוני המבחן (ורק בחישוב השגיאה השתמשנו בהם) היא 1,438

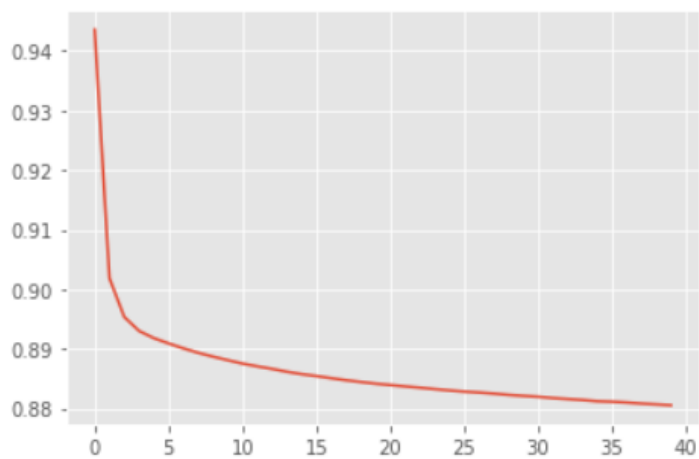
```
prediction_matrix_wrap = final_fsel_wrap_run_model.predict(x=x_test_wrap, batch_size=32, verbose=2)
prediction_result_wrap = np.argmax(prediction_matrix_wrap, axis=-1)
fsel_wrap_error = get_error(y_test, prediction_result_wrap)
print('The error of the features selected wrapper model is: {}'.format(fsel_wrap_error))
```

```
3687/3687 - 0s
The error of the features selected wrapper model is: 1438.0
```

ניתן לראות כי מבחינת הפרש ה-Val loss קיבלנו הפרש של 0.9438-0.8805=0.0633:

```
plt.plot(final_fsel_wrap_run_history.history['val_loss'])
```

```
[<matplotlib.lines.Line2D at 0x1adee3cda48>]
```



גם בשיטה זו, אומנם השארנו 6 פיצ'רים (הורדה של כ-54%) אך הדבר לטעמנו אינו מצדיק את הטרייד-אוף שנוצר בו השגיאה, גם במקרה כזה, גבוהה בכ- 15%.

<u>לסיכום נושא בחירת המשתנים</u>- בתיאוריה, צמצום מס' המשתנים אולי יקטין את זמן האימון של מודל מרובה ממדים, אך מכיוון שלא ניתן להתייחס לדאטהסט שלנו כ-מרובה ממדים, כלומר זמן האימון אינו ישתנה דרמטית במעבר בין 13 לפחות פיצ'רים אך כן יפגע ביכולת הלמידה של המודל ואף בדיוק שלו (השגיאה של המודל הסופי נמוכה במידה ניכרת). מבחינת למידה, נעדיף לא לצמצם את מספר המשתנים

ולתת למודל שלנו ללמוד על העולם כמה שיותר. ניתן לראות כי גם אם ישנם מספר פיצ'רים שהם הרחבה של משתנים אחרים, עדיין נראה כי המודל שלנו צריך אותם, ומצליח לדלות מהחלק שאינו יתיר בהם עוד חלקי מידע כדי לבצע למידה טובה יותר, ובסופו של דבר- להגיע לתובנות מדויקות יותר.

לכן, למודל הסופי נכניס את כל 13 הפיצ'רים ולא נוריד חלק מהם.

7. בונוס - מציאת מבנה הרשת ו/או הפרמטרים האופטימלי ללימוד הבעיה

ברור לנו שככל שיהיו יותר נתונים וככל שהרשת שלנו תהיה יותר עמוקה מבחינת שכבות ונוירונים אז נלמד יותר וגם השגיאה תהיה קטנה יותר, אבל האם זה "משתלם" לנו ללמוד עוד כמות מסוימת של מידע ב"עלות" של עוד שכבה/ X נוירונים/ X נתונים נוספים, השאלה היא כמה תוספת זו תלמד אותנו, אפשר לאמוד זאת בעזרת המדדים AIC ו-BIC.

בזמן שהרצנו את random search חישבנו לכל מודל את ערכי ה-BIC וה-AIC לפי הנוסחאות שלמדנו בהרצאה על ידי הפרמטרים הבאים:

- k - כמות הפרמטרים ברשת
- n - כמות הרשומות שיש לנו בסט האימון
- e - השגיאה לפי חישוב MSE לפי סט האימון (ולא לפי הטסט!)

ב-2 מדדים אלו המודל היותר טוב הוא המודל עם הערך הנמוך יותר.

נציג את כל המודלים ממוינים לפי מדד AIC:

```
# AIC
results.sort_values('AIC', ascending = True, inplace = True)
display(results)
```

	Evaluation	# of layers	# of units per layer	Activation function	Optimizer	Val loss	Val loss Delta	MSE	# of params	BIC	AIC
23	24	1	39	selu	RMSprop	0.848908	0.0420387	5666.5	666	150924	145780
24	25	6	6	relu	RMSprop	0.860408	0.0436813	5979.5	315	148410	145976
22	23	1	23	tanh	Adamax	0.851393	0.141645	6155	394	149661	146618
11	12	2	26	linear	RMSprop	0.853623	0.0554514	5776	1147	155921	147062
15	16	9	5	tanh	Adamax	0.873684	0.0880409	6931.5	328	151005	148472
6	7	4	22	linear	Adamax	0.866443	0.104116	6222	1895	164438	149801
4	5	6	21	linear	RMSprop	0.868082	0.119987	5691	2670	170483	149860
9	10	7	19	relu	SGD	0.933356	0.291277	5968.5	2606	170656	150528
3	4	4	27	tanh	SGD	0.915632	0.116868	5985.5	2730	171910	150823
13	14	2	46	linear	Adagrad	0.930939	0.108404	6726.5	2947	175971	153208
26	27	12	8	linear	Adadelta	1.17264	0.0943228	9719.5	931	162519	155328
29	30	16	21	selu	SGD	1.04397	0.305874	5582.5	7290	215086	158778
1	2	6	36	selu	Adagrad	1.11328	0.0442239	6072	7275	216345	160153
16	17	14	1	linear	Ftrl	1.08643	0.00895718	14533.5	46	160638	160283
14	15	11	4	tanh	Ftrl	1.08643	0.00896634	14533.5	271	162826	160733
17	18	5	41	selu	Adagrad	1.09318	0.0397006	6174.5	7588	219669	161059
10	11	6	39	linear	RMSprop	0.864038	0.157862	6820.5	8466	229869	164478
2	3	11	30	linear	RMSprop	0.874477	0.201438	6848.5	9813	243036	167240
18	19	8	39	tanh	SGD	1.04601	0.27352	5998	11586	258060	168570
28	29	16	26	tanh	Adamax	0.889992	0.440092	6899.5	10975	254459	169688

נציג את כל המודלים ממוינים לפי מדד BIC:

```
# BIC
results.sort_values('BIC', ascending = True, inplace = True)
display(results)
```

	Evaluation	# of layers	# of units per layer	Activation function	Optimizer	Val loss	Val loss Delta	MSE	# of params	BIC	AIC
24	25	6	6	relu	RMSprop	0.860408	0.0436813	5979.5	315	148410	145976
22	23	1	23	tanh	Adamax	0.851393	0.141645	6155	394	149661	146618
23	24	1	39	selu	RMSprop	0.848908	0.0420387	5666.5	666	150924	145780
15	16	9	5	tanh	Adamax	0.873684	0.0880409	6931.5	328	151005	148472
11	12	2	26	linear	RMSprop	0.853623	0.0554514	5776	1147	155921	147062
16	17	14	1	linear	Ftrl	1.08643	0.00895718	14533.5	46	160638	160283
26	27	12	8	linear	Adadelta	1.17264	0.0943228	9719.5	931	162519	155328
14	15	11	4	tanh	Ftrl	1.08643	0.00896634	14533.5	271	162826	160733
6	7	4	22	linear	Adamax	0.866443	0.104116	6222	1895	164438	149801
4	5	6	21	linear	RMSprop	0.868082	0.119987	5691	2670	170483	149860
9	10	7	19	relu	SGD	0.933356	0.291277	5968.5	2606	170656	150528
3	4	4	27	tanh	SGD	0.915632	0.116868	5985.5	2730	171910	150823
13	14	2	46	linear	Adagrad	0.930939	0.108404	6726.5	2947	175971	153208
29	30	16	21	selu	SGD	1.04397	0.305874	5582.5	7290	215086	158778
1	2	6	36	selu	Adagrad	1.11328	0.0442239	6072	7275	216345	160153
17	18	5	41	selu	Adagrad	1.09318	0.0397006	6174.5	7588	219669	161059
10	11	6	39	linear	RMSprop	0.864038	0.157862	6820.5	8466	229869	164478
2	3	11	30	linear	RMSprop	0.874477	0.201438	6848.5	9813	243036	167240
28	29	16	26	tanh	Adamax	0.889992	0.440092	6899.5	10975	254459	169688

$$AIC = n\ ln(E) + 2k$$

$$BIC = n\ ln(E) + k\ ln(n)$$

<u>מסקנות</u>

- שני המדדים שונים אבל לא בצורה משמעותית, שלושת המודלים המובילים לפי כל מדד הינם 23, 24 ו-25, המסקנה שלנו היא שפחות קריטי באיזה מדד מבין השניים להשתמש.
- בכל זאת יש ההבדל בין המדדים ואנו רוצים לציין אותו, ההבדל הוא המקדם של k בחלק השני של הנוסחאות, ב-AIC זה 2 לעומת ln של n, מכיוון שיש לנו אלפי רשומות בסט האימון אז יש יותר ערך לחלק זה ב-BIC לעומת AIC ולכן לפי מדד BIC יש יותר חשיבות לכמות הפרמטרים ברשת ובאמת המודל האופטימלי לפי BIC הוא 24 בו יש הכי פחות פרמטרים משלושת המודלים המובילים, לעומת זאת המודל המוביל לפי AIC הוא 23 ויש לו יותר פרמטרים מ-2 המודלים הבאים בטבלה.
- שניים מתוך שלושת המודלים המובילים (24 ו-23) הם גם המודלים המובילים לפי ניתוח הרגישות שעשינו, זה הפתיע אותנו כי חשבנו שיהיה "טרייד אוף" בין דיוק וכמות הלמידה של מודל לבין ה"יעילות" שלו, נראה לפחות לפי המקרה שלנו שזה הולך ביחד לא רע.