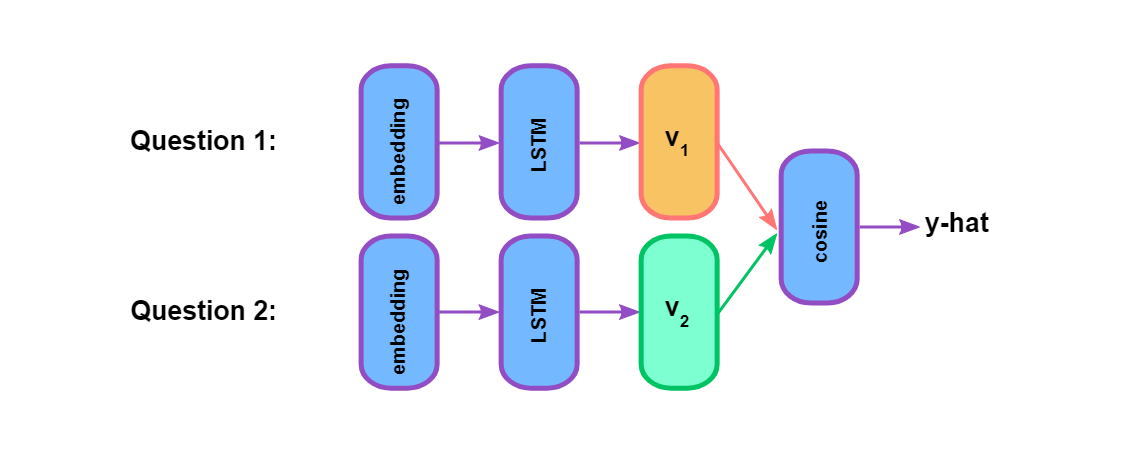
**עבודה מספר 4 סדנא בלמידה עמוקה- מידול שפה**

(ניב היה אחראי על רמת המילה (סעיף 2) וירין היה אחראי על רמת האותיות (סעיף 1) כאשר היה שיתוף קודם שיתוף מרוכז של טכניקת הפתרון והדוח נכתב ביחד)

בעבודה זו יצרנו מעיין בסיס למנוע חיפוש עבור מוצרי בית (של home depot). באופן גנרי, בהינתן קלט של שתי מחרוזות, עלינו להבין מה רמת הרלוונטיות של של מחרוזת אחת עבור מחרוזת שנייה. במקרה זה, בהינתן קלט של שאילתה, מה רמת הרלוונטיות של אותה שאילתה עבור תוצאה מסויימת. לשם ביצוע משימה זו בחנו 4 מודלים שכולם מתבססים על הארכיטקטורה של רשתות סיאמיות.



(תמונה לצורך המחשה)

בכדי לבצע את ה embedding עבור המחרוזות כדי שנוכל להשתמש בכלי Deep learning קיימים, השתמשנו בספריית word2vec.בתהליך, הסרנו stopwords ובהנחה שהקלט עבר גודל מסויים (לדוגמא עבור רמת המילה - גודל של שלושים מילים, עבור רמת האותיות גודל של 70 אותיות) חתכנו אותו, ואם הוא היה קטן מידי, ריפדנו ב padding. בנוסף, ביצענו נרמול של ערכי relevance לתחום שבין 0-1 כדי שנוכל לעבור עם ערכים אלו בצורה טובה יותר.

לאחר מכן, העברנו שני קלטים ברשת, ביצענו חישוב loss בין הפלטים שלהם (נקרא לפלט של Loss זה l1) ולאחר מכן חישוב loss בין l1 לבין relevance. את הloss הזה פעפענו ברשת ובכך אימנו אותה (רשת מבוססת ארכיטקטורת LSTM). כיוון שמדובר בבעיית רגרסיה השתמשנו בשני מדדים, RMSE,MAE לשם חישובים ההפסדים.

בניסויים הראשונים, לא היינו אגנוסטיים לגודל הקלט מה שגרם לשגיאות ולאימון לקוי. לאחר מכן, ביצענו שינויים ברשת כך שתדע להיות אגנוסטית לגודל הקלט ובכך הגענו לביצועים הגיוניים.

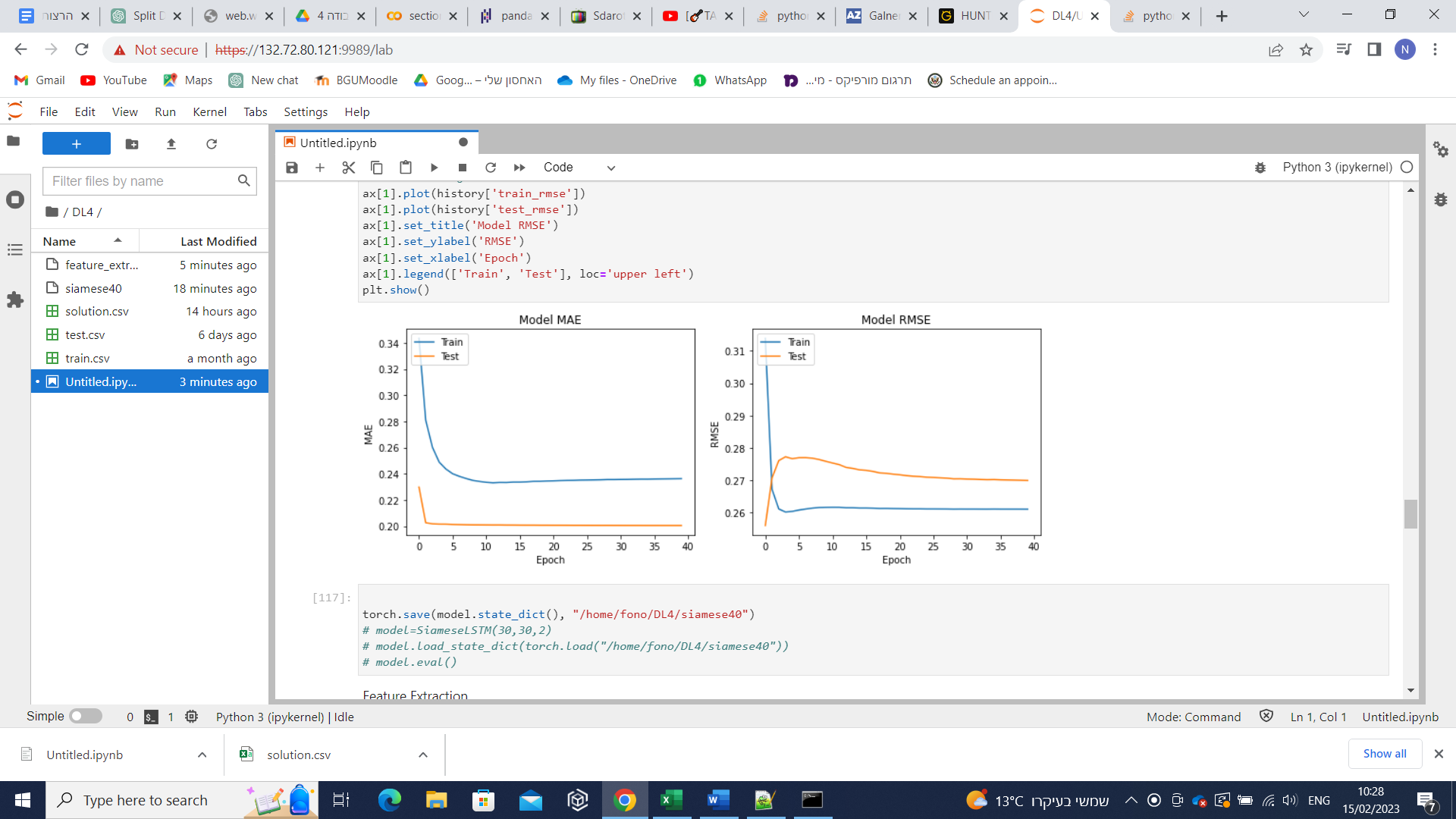
לאחר שביצענו זאת, השתמשנו בשני אלגוריתמים של למידת מכונה קלאסית לשם ביצוע בעיית הרגרסיה בהנתן הfeature extraction של הרשת. לאחר שהעברנו את שני הקלטים ברשת, שמרנו את שני הפלטים בטבלה והשתמשנו בטבלה עם ה relevance המתאים והשתמשנו בטבלה זו לכדי אימון המודלים. השתמשנו במודל random forest ומודל knn כאשר k=2. ביצענו זאת כיוון שמעניין לראות האם יש הבדל בין מודל שהוא eager למודל lazy בהקשר של feature extraction.

בנוסף השתמשנו באלגוריתם נאיבי ברמת האותיות כדי לבדוק שאכן האלגוריתמים שלנו יעילים. האלגוריתם משווה את ה cosine similarity בין הוקטור המייצג את האותיות בכותרת לבין הוקטור המייצג את האותיות במונח החיפוש. cosine similarity יתן לנו מספר בין 0 ל-1 המייצג את הדומי בין שתי הוקטורים מה שמתאים להצגת מידת רלוונטיות החיפוש.

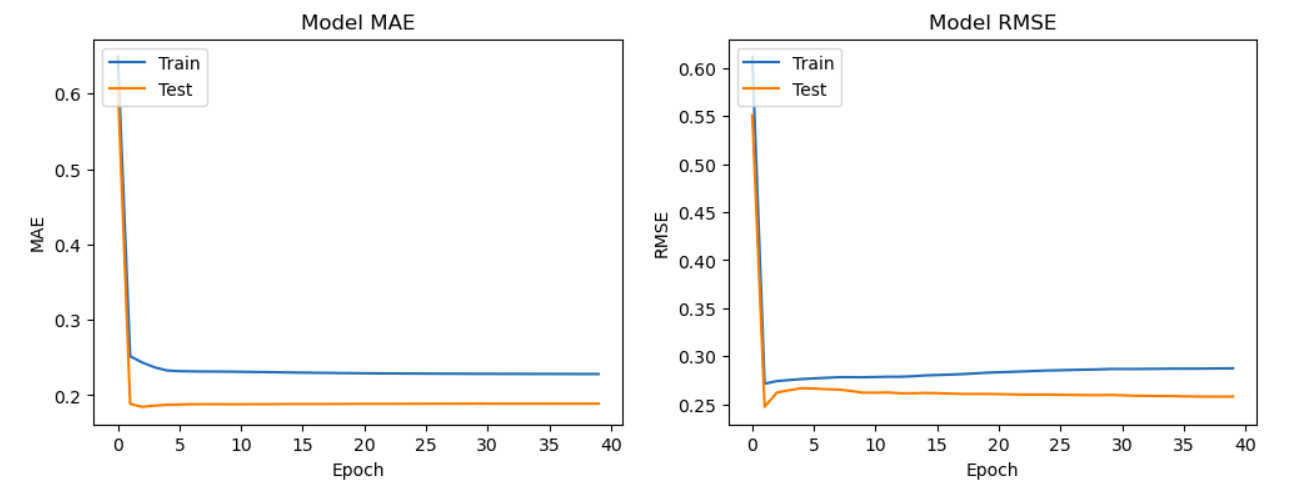
השתמשנו בטבלה מייצגת לניסויים השונים שלנו:

| test mae | val  mae | train  mae | test rmse | val rmse | train rmse | runtime | comments | model type |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0.226 | 0.22220 |  | 0.252 | 0.24823 | 429.803 | 5 epochs tial, no convergence- different network structure | word level LSTM |
|  | 0.216 | 0.19720 |  | 0.261 | 0.23662 | 1235.385 | 15 epochs, convergence from 0.7+ in rmse&mae | word level LSTM |
|  | 0.233 | 0.20287 |  | 0.271 | 0.26323 | 426 | changing loss functions to multiplication of outputs instead of loss of them, 4 epochs in colab | word level LSTM |
|  | 0.218 | 0.21571 |  | 0.248 | 0.26001 | 700 | same as above but with decreasinig lr every 2 epochs,mul by ¾. on the first iteration we got val mae of 0.214, rmse of 0.245 and it has only increased by the continue of the epochs | word level LSTM |
|  | 0.214 | 0.19330 |  | 0.265 | 0.23495 | 1336.34 | 40 epochs, same lr at all epochs, best model so far so we saved it and feature extracting it | word level LSTM |
|  | 0.356 | 0.25274 |  | 0.279 | 0.25387 | using ealy stopping so didnt count | 8 epochs, same lr at all epochs, using early stopping after increasing validation, different network using only lstm as backbone without fc layers as we did so far. shows better convergence  In addition in the previous tests our network wasnt agnostic to the input size now it is. | word level LSTM |
|  | 0.247 | 0.25205 |  | 0.28266 | 0.238 | 1751.742 | same network, 30 epochs, decreasing lr every 5 epochs mul by 0.75 | word level LSTM |
| 0.509 | 0.257 | 0.20735 | 0.467 | 0.273 | 0.22897 | 1898.303 | same network,40 epochs | word level LSTM |
|  | 0.22 | 0.221 |  | 0.266 | 0.233 | 1386.613 | vector size 30, limit to first 30 chars | character level LSTM |
|  | 0.223 | 0.252 |  | 0.294 | 0.237 | 2179.94 | vector size 30, limit to first 100 chars | character level LSTM |
|  | 0.263 | 0.267 |  | 0.264 | 0.235 | 2244.426 | vector size 20, limit to first 100 chars | character level LSTM |
| 0.586 | 0.231 | 0.206 | 0.868 | 0.264 | 0.233 | 1795.34 | vector size 20, limit to first 70 chars, best model so far so we saved it and feature extracting it | character level LSTM |
|  | 0.662 | 0.663 |  | 0.438 | 0.44 | 369.582 | cosine similarity between the embedding vector of the search term to the embedding vector of the title, vector size 30, limit to first 100 chars | naive model |
| test mae | val  mae | train  mae | test rmse | val rmse | train rmse | runtime | comments | model type |
|  | 0.2327 | 0.1388 |  | 0.0894 | 0.0339 | 0.015+1898.03 (of model) | knn is lazy evaluator so we will check the prediction time | KNN word level feature extraction |
| 0.356 | 0.1943 | 0.204 | 0.2 | 0.056 | 0.065 | 1898.03 (of model)+0.207 | RF is eager evaluator so we will check the training time | Random Forest  word level feature extraction |
| 0.367 | 0.251 | 0.139 | 0.238 | 0.1 | 0.033 | 0.025(of model)+ 0.107 |  | KNN character level feature extraction |
|  | 0.218 | 0.218 |  | 0.071 | 0.071 | 1.873(of model)+0.154 |  | Random Forest  character level feature extraction |

word-level LSTM



char-level LSTM



לסיכום, למדנו להשתמש בארכיטקטורת Siamese networks והבנו את השימוש הרב שלה במציאת רלוונטיות באחזור מידע. בחנו גם את ה word level וגם את ה character level כאשר קיבלנו את התוצאות הבאות- ברמת word level האלגוריתם הטוב ביותר הוא DNN+פיצ'ר אקסטרקשן לרנדום פורסט משמע אלגוריתם eager עדיף במקרה זה, מצד שני ברמת character level המודל הטוב ביותר הוא DNN+פיצ'ר אקסטרקשן ל- KNN משמע אלגוריתם lazy עדיף במקרה זה. יש לציין שהשוני בין שימוש באלגוריתם lazy לאלגוריתם eager אינו מובהק ויש צורך לקחת גורמים נוספים בחשבון בבחירת סוג האלגוריתם (כגון זמן ריצה בזמן אמת).  
לאחר שביצענו את אימוני האלגוריתמים בדקנו על נתוני ה test האמיתיים וקיבלנו שהאלגוריתם ברמת character level נותן תוצאות טובות יותר.