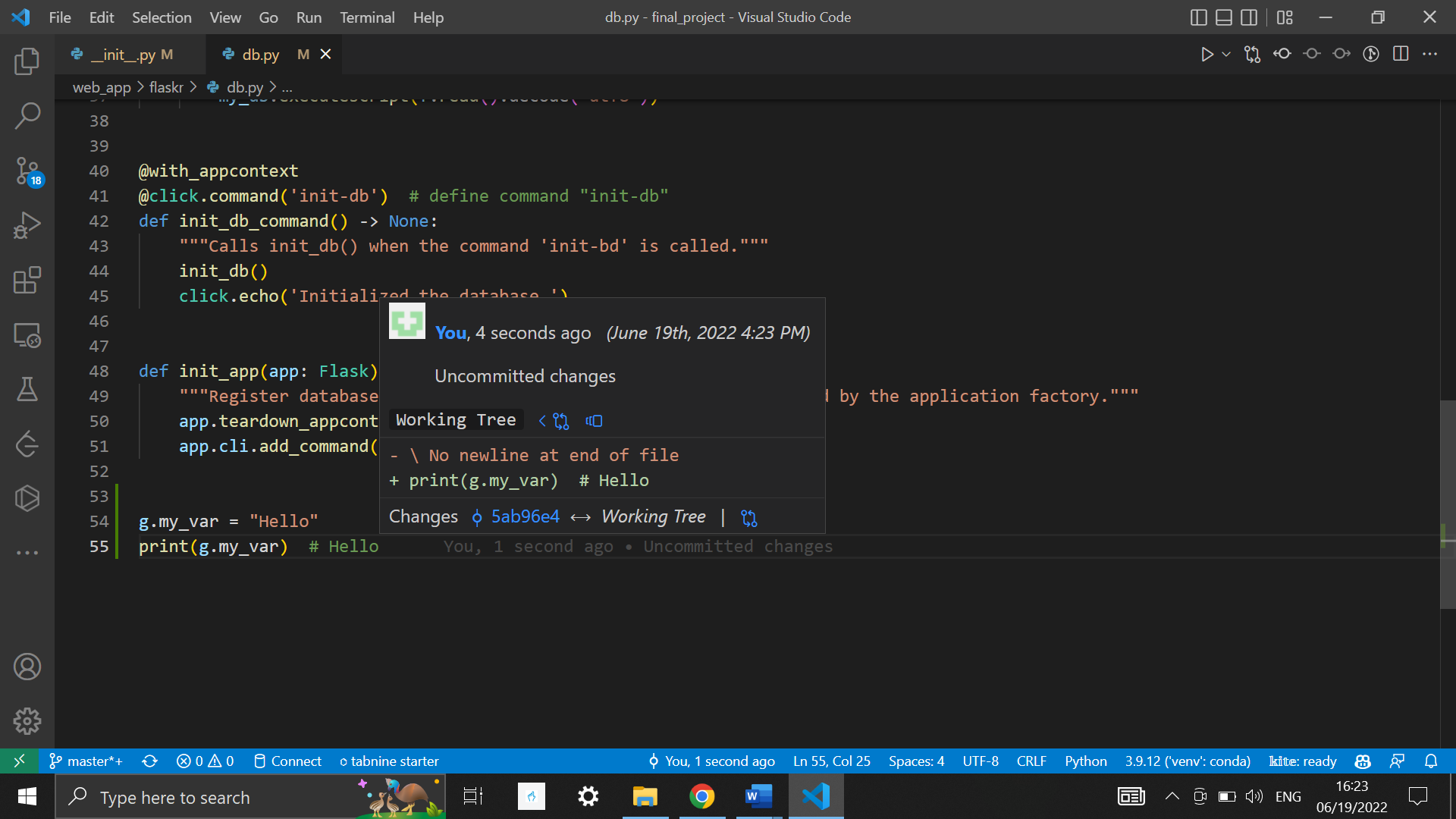
**החלק המחקרי**

**אפליקציית הווב**:

כל אפליקציה שמפותחת באמצעות Flask חייבת לכלול קובץ ששמו: "\_\_init\_\_.py" ובו פונקציה הנקראית "create\_app" בלבד. פונקציה זאת נקראת כשהשרת מתחיל להריץ את האפליקציה. הפונקציה מבצעת את הפעולות הבאות: יצירת עצם האפליקציה שיורש מהמחלקה Flask , הגדרת קונפיגורציה (למשל מיקום בסיס הנתונים), יצירת תקייה לבסיס הנתונים, יצירת בסיס הנותנים ושמירת blueprints.

המשתנה הגלובלי g מוגדר כשהמשתמש נכנס לאפליקציה. הוא ריק כל עוד לא מכניסים אליו שום דבר. אפשר לשמור בו משתנים (מכל סוג) לפי שם ולגשת בו לכל משתנה לפי שם:



בקובץ db.py נמצאות הפונקציות שאחראיות על ניהול בסיס הנתונים:

הפונקציה get\_db() בודקת אם קיים חיבור לבסיס הנתונים (עצם מהמחלקה sqlite3.Connection) במשתנה הגלובלי ואם לא, יוצרת אחד כזה ושומרת אותו ב g.my\_db ולאחר מכן (בלי קשר לתנאי הראשון) מחזירה את g.my\_db.

הפונקציה init\_db() מקבלת חיבור למסד הנתונים ומפעילה את פקודות ה SQL שבקובץ schema.sql. הקובץ schema.sql מכיל את הפקודות הבאות:

אם קיימות טבלאות בשמות: completion, model, user מחק אותן.

צור את הטבלאות הבאות (כל הטבלאות בבסיס הנתונים):

user (המאחסנת משתמשים) עם העמודות:

id - מספר סידורי: שלם של המשתמש שהוא המפתח הראשי של הטבלה.

username – שם משתמש: טקסט ומיוחד.

password – סיסמה מוצפנת: טקסט (הסיסמה מוצפנת לפני שהיא נכנסת לבסיס הנתונים).

model המאחסנת מודלי למידה עמוקה עם העמודות:

Id - מספר סידורי: שלם של המודל שהוא המפתח הראשי של הטבלה.

User\_id – מספר סידורי של המשתמש שהעלה את המודל לאתר (כל משתמש יכול להעלות מספר מודלים לאתר וככה לאפשר לכל המשתמשים להשתמש בהם) בין מודל למשתמש יש קשר רבים לרבים.

model\_name – שם הקובץ של המודל: טקסט.

Created – הזמן בו נוצר המודל. ברירת המחדל היא הזמן בו המודל נכנס לבסיס הנתונים.

בנוסף, לכל אחד מהיפר הפרמטרים של המודל יש עמודה בטבלה זו. שמות הייפר-הפרמטרים הם: set\_size, batch\_size, learning\_rate, num\_blocks, d\_model, dff, num\_heads, dropout\_rate.

completions – השלמות. כל משתמש באתר יכול לבחור מודל מהמודלים הקיימים ולהכניס טקסט ולראות איך המודל משלים את הטקסט. העמודות בטבלה הן:

User\_id - מספר משתמש, model\_id – מספר מודל, created – הזמן בו נוצרה ההשלמה, prompt – הטקסט שהמשתמש הכניס למודל ו answer – הטקסט שהמודל פלט.

הקובץ auth.py מכיל את ה blueprint auth שכולל את הפונקציה login ו register שמציגות את עמודי הכניסה והרשמה לאתר. כל אחת מהפונקציות קולטת את שם המשתמש והסיסמה מעמוד ה HTML. אם המשתמש נרשם בהצלחה הוא מועבר לעמוד ההתחברות ואם הוא התחבר בהצלחה הוא מועבר לעמוד completion.index.

הפונקציה logout מנקה את הסשין ומעבירה את המשתמש לעמוד completion.index.

הפונקציות login, logout, register מופעלות (ללא פרמטרים) כשהמשתמש נכנס לקישור auth/login, auth/logout, auth/register בהתאמה.

בנוסף, הקובץ מכיל את הגדרת הקשטן (decerator) login\_required כשקוראים לפונקציה המקושתת בו, הוא בודק שיש משתמש במשתנה הגולבאלי (g) ואם לא, הוא מעביר את המשתמש לעמוד ההרשמה. הוא מקשט את הפונקציות של העמודים הדורשים התחברות למערכת.

הקובץ מכיל גם את הפונקציה load\_logged\_in\_user שנקראת באופן אוטומטי כשמשתמש מגיע לעמודים register, login או logout הבודקת אם המשתמש שמור בסשין ואם כן שומרת או במשתנה הגלובאלי.

הקובץ model.py מכיל את ה blueprint model

הפונקציה view\_all טוענת את העמוד בו המשתמש רואה את כל המודלים שהועלו לאתר על כה. הפונקציה upload טוענת את העמוד בו המתמש מעלה מודלים לאתר.

הפונקציות allowed\_file, file\_errors, model\_errors הן פונקציות עזר ל upload.

הקובץ creation.py מכיל את ה blueprint creation.

הפונקציה index טוענת את העמוד הראשי בו מוצגות כל ההשלמות הקיימות בבסיס הנתונים.

הפונקציה create טוענת את העמוד בו המתמש מעלה כותב טקסט ובוחר מודל, לאחר שהמשתמש לוחץ על הכפתור complete המודל שהוא בחר משלים את הטקסט. ההשלמה נכנסת לבסיס הנתונים והמשתמש מועבר לעמוד הראשי.

הפונקציה complete היא פונקציית עזר ל create.

בקובץ tokenizer.py קיימות שתי פונקציות עזר: tokenize\_and\_preprocces היא פונקציית עזר לפונקציה create שמקבלת מחרוזת ומחזירה רצף של טוקנים כטנזור מסוג tf.Tensor.

Init\_tokenizer היא פונקציית עזר שיוצרת טוקנייזר מהסוג tf\_text.BertTokenizer הזהה לטוקנייזר ששימש ליצירת המודלים. פרט זה חשוב מאוד מכיוון שכל מודל לומד את יצוג המילים לפי טוקנייזר מסויים.

השתמשתי ב Pytest בשביל לבדוק את כל הקוד של האפליקציה. הפקודה pytest (ב command prompt מתוך התקייה final\_project/web\_app כאשר הסביבה הווירטואלית עובדת) מריצה את הקובץ conftest.py שמגדיר ומפעיל את כל הבדיקות.

**פיתוח המודלים:**

**א – הנתונים:**

המחברת create\_model.ipynb היא מחברת ג'ופיטר שכאשר מריצים את תאיה לפי הסדר היא יוצרת מודל, מאמנת אותו ושומרת אותו בגוגל דרייב. מומלץ להריץ את המחברת דרך google collaboratory (colab) עם האצת GPU.

המחברת לפי שלבים:

1) המחברת מורידה ומיבאת את כל הספריות ההכרחיות ומדפיסה מידע על המעבד ועל גרסאות פייתון והספריות Tensorflow, Tensorflow-Text.

2) המחברת משנה את הגרעין (seed) ל-0 על מנת לשמור על תוצאות קבועות ועל יכולת שחזור תוצאות. המחברץ מגדירה משתנה device מסוג מחרוזת שיכול להיות שווה "colab" כשמריצים את המחברת על colab ו "local" אחרת. משתנה זה ישמש אחר כך למציאת מיקומי קבצים. לא השתמשתי במשתנה בוליאני כדי לאפשר יצירת אופציה נוספת במידת הצורך.

3) קביעת היפר פרמטרים: היפר פרמטרים הם מספרים שנקבעים בתחילת כל אימון מודל והערכים שלהם משפיעים מאוד על תהליך האימון. הם מתחלקים לקבוצות:

תכונות של הליך הלמידה - מספרים שמשמשים לתהליך הלמידה עצמו אך אינם תכונות של המודל שנוצר. התכונות של תהליך הלמידה אצלי הן:learning\_rate, set\_size, batch\_size

תכונות של המודל הנוצר - התכונות של המודלים שלי הן: num\_blocks,

d\_model, dff, num\_heads, dropout\_rate

תכונה משותפת – תכונה של המודל שחייבים להשתמש בה גם בתהליך האימון אצלי זאת max\_seq\_len

שימו לב שיש היפר פרמטרים שאני משאיר קבוע והם מופיעים במקום בו אני משתמש בהם ולא מגדיר בתחילת המודל-זאת מכיוון שמקובל להשאיר אותם בגודל מסוים קבוע (מכיוון שזה הערך שבסבירות גבוהה מאפשר אימון מוצלח). לדוגמה הפרמטר beta1 של ה-optimizer Adam שהערך המקובל שלו הוא 0.9.

VOCAB\_SIZE היא תכונה משותפת של הטוקנייזר והמודל אך היא שנארת קבועה כי תהליך יצירת הטוקנייזר אינו משתנה בין מודל למודל ולכן זאת תכונה שאיננה הייפר פרמטר.

שימו לב גם שהמודל לומד לזהות טקסט שעבר טוקניזציה על ידי טוקנייזר מסויים. זאת מכיוון שכל טוקנייזר נותן ערך מספרי אחר לכל מילה.

4) טעינת סט הנתונים: [סט הנתונים בו השתמשתי](https://www.kaggle.com/datasets/urbanbricks/wikipedia-promotional-articles) הוא סט של מאמרים מוויקיפדיה באנגלית שתויגו על ידי הקהילה [כמאמרים טובים](https://en.wikipedia.org/wiki/Wikipedia:Good_articles). בחרתי בו בגלל: 1. הגודל שלו (כחצי מגהבייט) שמתאים לכמות כוח המחשוב שברשותי. 2. אורך הטקסטים-הסט מכיל טקסטים ארוכים שבהם שיערתי שלמודלים שלי יהיה יתרון. 3. בקרת איכות-טקסטים שסומנו כטובים מכילים פחות תופעות כגון דברי שנאה, הסטה ודברי שקר שלא הייתי רוצה שהמודלים שלי ילמדו. 4. שפה אחידה-תמיכה בשפה אחת מפשטת את כל הפרויקט.

סט הנתונים נטען ל DataFrame בעזרת ספריית pandas והעמודה טקסט מומרת לרשימה של מחרוזות הנקראית data\_list. מודפסים נתונים בסיסיים על הרשימה.

5) יצירת אוצר המילים: בהינתן סט נתונים של טקסטים ומספר מקסימלי של מילים באוצר מילים (במקרה שלי 8192) אוצר המילים הוא רשימה של תתי המילים הכי נפוצות בסט הנתונים. תתי מילים יכולות להיות מילים או תתי מילים נפוצות (לדוגמה הסיומת ים). אל רשימה זו נכנסות "מילים שמורות": [UNK] – מילה לא מוכרת (קיצור של unknown), [START] תחילת טקסט (שונה מ-start שמציינת את המילה עצמה), [END] – סוף טקסט, [PAD] – ממלא מקום בטקסט שקצר מאורך הטקסט המקסימלי, [MASK] – מילה מוסתרת – הסבר מפורט יותר בהמשך. אוצר המילים לא מבדיל בין אותיות גדולות וקטנות. אוצר המילים נשמר בקובץ vocab.txt. אם קובץ זה כבר קיים, האוצר מילים לא נבנה מחדש אלא נקרא מתוך הקובץ. לאחר מכן כל מילה באוצר המילים ממורת לטנזור מסוג מחרוזת.

6) יצירת הטוקנייזר: look\_up\_table הוא עצם מסוג tf.lookup.StaticVocabularyTable בעל פונקציה lookup שמקבלת טנזור של מחרוזות ומחזירה טנזור של שלמים עם אותה צורה ככה שלכל איבר בטנזור:

אם האיבר באוצר מילים: "מחליפים" אותו באינדקס של האיבר באוצר המילים.

אחרת: "מחליפים" אותו באינדקס של המילה השמורה [UNK]. ("מחליפים" במירכאות כי טנזור הוא בלתי ניתן לשינוי ובעל סוג קבוע).

Tokenizer הוא עצם מסוג tf\_text.BertTokenizer בעל שתי פונקציות שאשתמש בהן:

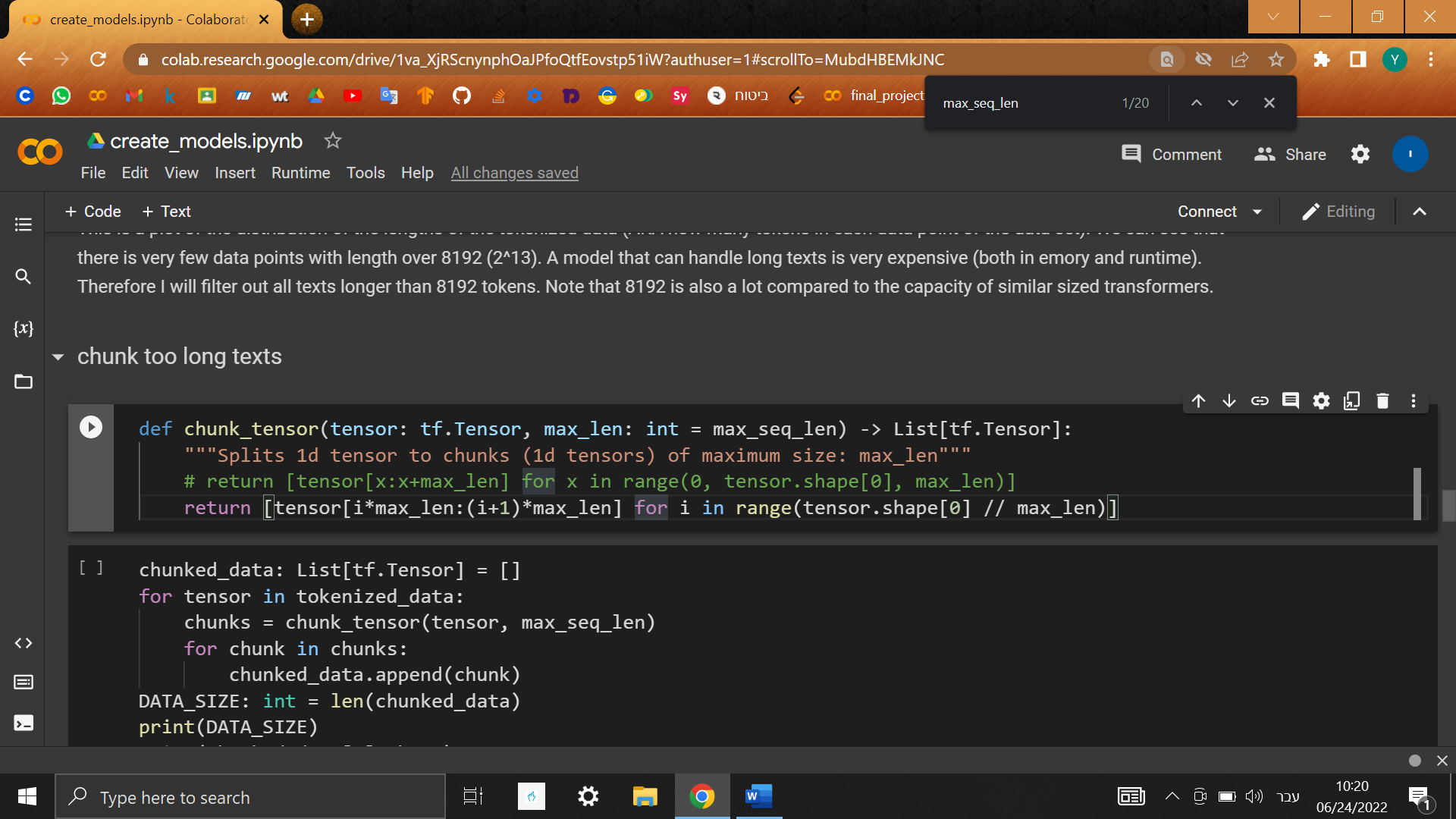
tokenize מקבלת מחרוזת, מחלקת אותה לטנזור (מדרגה 1 וסוג מחרוזת) של תתי מילים ומחזירה את הפלט של look\_up\_table.lookup() כשהיא מקבלת את הטנזור.

7) tokenization: הפונקציה tokenize\_string מקבלת מחרוזת, קוראת לפונקציה tokenizer.Tokenize עליה, ממירה אותה מ- tf.RaggedTensor ל-tf.Tensor, משנה את טיפוס האיברים מ – int64 ל-int32 ומוסיפה טוקן התחלה וטוקן סוף.

Tokenized\_data היא רשימה של טנזורים שנוצרת מקריאה ל tokenize\_string על כל איבר ברשימה data\_list. בקטע מודפס מידע על אורכי הטנזורים ברשימה.

8) חתיכת טקסטים ארוכים מדי: קבעתי את האורך המקסימלי לכל טקסט בחלק 3 של המחברת. הפונקציה chunk\_tensor מקבלת טנזור מדרגה 1 וסוג שלם ואורך מקסימלי מסוג שלם, היא חותכת את הטנזור לפי הכלל:

לכל i מ 0 עד העיגול למטה של אורך הטנזור חלקי האורך המקסימלי, החלק ה i מכיל את האיברים מ i כפוך האורך המקסימלי עד (i+1) כפול האורך המינימלי. הפונקציה מחזירה את החלקים כרשימה של טנזורים.



כל טנזור ברשימה Tokenized\_data נחתך על ידי הפונקציה והחלקים של כל הטנזורים נשמרים ברשימה של הטנזורים chunked\_data.

9) ריפוד: חלק מהטנזורים קצרים יותר מהאורך המקסימלי והמודל מקבל טנזורים שהאורך שלהם שווה לאורך המקסימלי בלבד. לכן, לכל טנזור ברשימה chunked\_data אצרף את התת מילה [PAD] שמיוצגת על ידי המספר 0.

10) חלוקה לסט אימון, מבחן ווידוי (training, test and validation): סט האימון הוא 80% מסט הנתונים, סט המבחן והוידוי שניהם 10% מסט הנתונים. כל סט מחולק ל batches ונשמר ברשימה של טנזורים בה כל טנזור הוא batch שצורתו [גודל הבאטצ', האורך המקסימלי של הטקסט] (שני הגדלים הם הייפר פרמטרים שנקבעו בחלק 3).

11) ניקוי הזכרון: בעבודה עם מחברות ג'ופיטר, המשתנים אינם נמחקים באופן אוטומטי לאחר השימוש האחרון בהם ולכן אני מוחק את כל המשתנים שאני לא אשתמש בהם בהמשך באמצעות המילה השמורה del.

**ב – המודל:**

הפרטים המלאים על חלקי המודל בחלק התיאורטי

כל המודלים שאצור הם עצמים מהמחלקה SeTransformer שיורשת מ tf.keras.Model (שיורשת גם מ tf.keras,layers.Layer) ויש לה 2 מתודות:

אתחול "\_\_init\_\_" שמתאחלת את tf.keras.Model, יוצרת קידוד מיקומים המתאים להייפר הפרמטרים של המודל ומשמש להגדרת ה encoder וה decoder ואז יוצרת את חלקי כל אחד מחלקי המודל-כולם עצמים שיורשים מהמחלקה tf.keras.layers.Layer.

קריאה: "call" שמקבלת רשימה בה שני טנזורים: פלט (inp) הוא רצף הטוקנים שהמודל צריך להשלים שאורכו כאורך הטקסט המקסימלי ומטרה (tar) הוא רצף של טוקנים באורך השווה לאורך הפלט הרצוי. ובנוסף מקבלת משתנה training שנכון כשהמודל מתאמן ושגוי אחרת. למתודה זו קוראים באמצעות המתודה predict (קריאה ישירה ל call מביאה לשגיאה).

למחלקה SeTransformer חמש תכונות שכולן יורשות בעקיפין מהמחלקה tf.keras.layers.Layer: decoder הוא עצם מהמחלקה Decoder, encoder הוא עצם מהמחלקה Encoder. embedding, dense and softmax הן שכבות הבנויות בתוך tensorflow.

המחלקות: Decoder, Encoder, EncoderBlock, DecoderBlock, PointWiseFeedForwardNetwork, MyMultiHeadAttention, ScaledDotProductAttention בנויות דומה למחלקה SeTranformer: הן יורשות (באופן ישיר, בשונה מ SeTranformer) מ tf.keras.layers.Layer ויש להן שתי מתודות: אתחול וקריאה. מתודת האתחול יוצרת את תתי השכבות ושומרות אותן במשתני מחלקה. בחלק מהמקרים היא שומרת גם הייפר-פרמטרים שהיא מקבלת, חישובי עזר ואת קידוד המיקומים. מתודת הקריאה מקבלת בנוסף למידע שמתואר בחלק התיאורטי גם מסכות שנוצרות בכל קריאה למודל.

קידוד מיקומים: בגלל שהטרנספורמר קולט מידע במקביל ולא כרצף, הוא צריך מידע נוסף על מיקומי כל תת מילה בטקסט שהוא מקבל. הפונקציה create\_positional\_encoding מקבלת את אורך הטקסט המקסימלי ואת d\_model שהוא אחד מההייפר-פרמטרים של המודל ומחזירה טנזור מהצורה # (1, , d\_modelאורך הטקסט המקסימלי) שכל אביריו בין 1 ל 1- ומסוג עשרוני (float). הפונקציה get\_angles מבצעת חישוב עזר לפונקציה positional\_encoding ולכן היא פונקציה פנימית של create\_positional\_encoding.

positional\_encoding נקראת רק מתוך מתודת האתחול של SeTransformer. התוצאות שלה נשמרות במשתנה מחלקה במחלקות Encoder ו Decoder והן שימושיות במתודת הקריאה של המחלקות הללו.

יצירת מסכות: כדי שהמודל לא יתייחס בחישוביו למטרה ולטוקן הריפוד ([PAD]), המסכות הן טנזור באותה צורה של הטנזור אותו מכסות ככה שערך כל איבר במסכה שתואם לאיבר שיש לכסות הוא 1 ושאר הערכים 0.

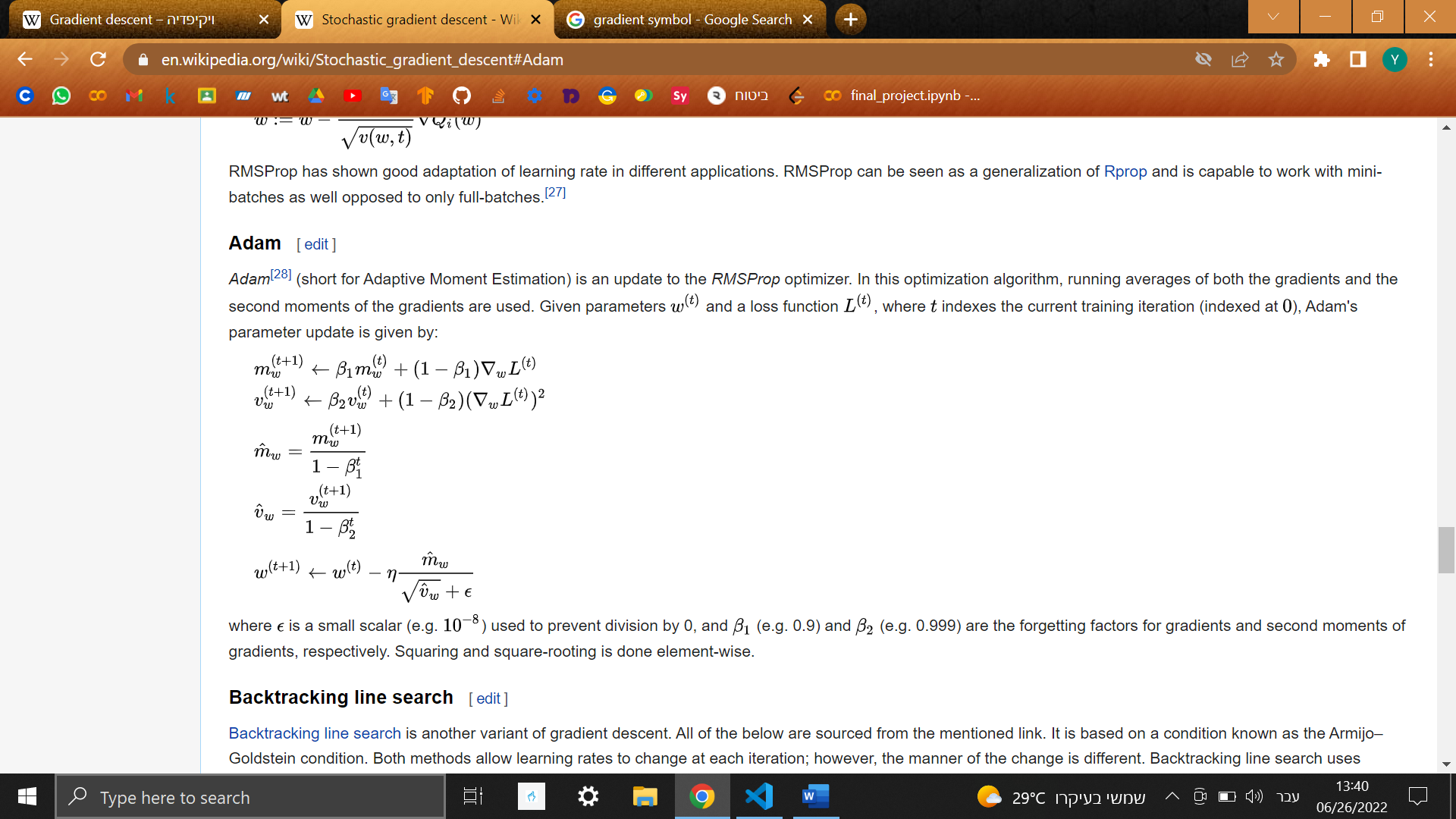
הפונקציה create\_masks מקבלת את הפלט, המטרה ואת הערך השלם שמייצג ריפוד של המודל ומחזירה מסכה אחת המכסה את המטרה ומסכה שניה המכסה את טוקן הריפוד. לפונקציה פונקציית עזר פנימית שנקראת create\_padding\_mask שמייצרת מסכה לכיסוי טוקן הריפוד שהיא גם חישוב עזר ליצירת מסכה למטרה. הפונקציה נקראית מתוך מתודת הקריאה של המחלקה SeTranformer.

**ג – האימון:**

נגדיר את המייעל להיות מייעל מסוג Adam (ראשי תיבות להערכת מומנטום הסתגלותית) עם קצב הלמידה שקבענו בחלק א 3, אפסילון ששווה לערך החיובי הקטן ביותר שמאפשר חישובים ללא סיכון של עיגול למטה ל 0 ועם בטא 1 ובטא 2 ברירת המחדל (0.9 ו- 0.999 בהתאם). למייעל מתודה apply\_gradients המקבלת 2 רשימות של טנזורים כאשר כל שני טנזורים מתאימים בעלי אותה צורה וכולם מסוג עשרוני. רשימה אחת של הפרמטרים הניתנים לאימון של המודל ורשימה של הנגזרות היחסיות של כל פרמטר ביחס לערך המתקבל מפונקציית המטרה (מספר עשרוני) ומחשבת פרמטרים חדשים למודל לפי הנוחסה (הנוסחה מחושבת לכל פרמטר בנפרד):

נקודה חשובה! כשיוצרים מודל, הפרמטרים מוגרלים באופן רנדומלי והם מספרים רנדומליים בין מינוס אחד לאחד.

כאשר: t הוא אינדקס של החזרה,  *הוא ערך פרמטר ניתן לאימון, הוא ערך המטרה (הערך שחושב בפונקציית המטרה), ו- הוא הנגזרת היחסית של הפרמטר היחס לערך המטרה. זאת הנוסחה:*



נשתמש בפונקציית המטרה: sparse categorical crossentrop.

הפונקציה מחשבת את השוני בין טנזור הסתברות בו האיבר במקום i מייצג את ההסתברות לתוצאה עם אינדקס i לבין האינדקס של התוצאה הנכונה. מטרת המודל שלנו היא לחזות הסתברות כמה שיותר גבוהה לתוצאה הרצויה. ערך גבוהה של הפונקציה שקול לחזייה לא מוצלחת ולהפך ולכן מטרת המודל לצמצם את ערך הפונקציה.

הפונקציה מוגדרת עבור הסתברות של התוצאה הרצויה p:

כשההסתברות שואפת ל 0, ערך הפונקציה שואף לפלוס אין-סוף וכשהיא 1, ערך הפונקציה 0. הפונקציה מתעלמת מההסתברות של כל תוצאה אחרת.

לאחר מכן, ניצור את המודל עם הייפר-הפרמטרים שבחרנו בחלק א 3.

הפונקציה train\_step ממשת את אלגוריתם ה Stochastic gradient descent עם מייעל הערכת מומנטום הסתגלותית.

כל מודל למידת מכונה הוא בעצם פונקציה בעלת הרבה פרמטרים (במקרה שלי עשרות מליונים) שניתן לגזור בכל נקודה. כאשר כל קלט תקין למודל הוא נקודה במרחב הקלט האפשרי. גם פונקציית המטרה היא פונקציה מתמטית גזירה ולכן אפשר למצוא את הנגזרת היחסית של כל פרמטר במודל ביחס לפלט. הנגזרת היחסית של פרמטר ביחס לפלט מייצגת את השפעת הפרמטר על הפלט (גודל וכיוון).

Tape הוא אובייקט מהמחלקה tf.GradientTape שעוקב אחרי הנגזרות של כל הפעולות בעל פונקציה gradient שמחשבת את הנגזרת היחסי בין כל שני משתנים.

הפונקציה train\_step מקבלת את הפלט והקלט הרצוי בצורת רצץ של טוקנים, קוראת למודל ומחשבת את פונקציית המטרה תוך כדי מעקב על הנגזרות, מחשבת את הנגזרת היחסית של ערך פונקציית המטרה ביחס לפרמטרים של המודל ואז משנה את הפרמטרים של המודל בעזרת המיעל. הפונקציה מחזירה את ערך פונקציית המטרה. היא מקושטת בעזרת הקשטן tf.function.

הפונקציה validate מקבלת באטצ' מסט הוולידציה או המבחן ומחשבת את ערך פונקציית המטרה הממוצע.

הפונקציה check\_points מקבלת מיקום רצוי מסוג מחרוזת, מודל מסוג SeTransformer וערך פונקציית המטרה הממוצע על סט האימון, הוולידציה והמבחן מסוג עשרוני. היא שומרת את המודל במיקום הרצוי ומוסיפה שורה לקובץ expirement.csv שכל שורה בו מייצגת מודל אחד שלו ובשורה יש מידע על היפר הפרמטרים של המודל כמו גם על ערך פונקציית המטרה שלו על סט האימון והוולידציה. אם קיים מודל שמור מאותו תאריך, הפונקציה מוחקת אותו וזאת מכיוון שהמודל הוא בהכרח גרסה מוקדמת יותר של המודל הקיים שביצועיה נמוכים.

לולאת האימון:

יוצרת רשימות ריקות של ערכי פונקציית המטרה על סט האימון והוולידציה בכל חזרה מסוג עשרוני.

לכל מספר חזרה בטווח (0 עד מספר גבוה) (עד שהמחברת תנתק מהאתר או שפונקציית המטרה תתחיל לעלות):

יוצרת רשימות ריקות של ערכי פונקציית המטרה על סט האימון והוולידציה בכל באטצ' מסוג עשרוני.

לכל באטצ' בסט האימון:

מחלקת את הבאטצ' לסטים, קוראת לפונקציה train\_step על כל סט ועוקבת אחרי ערך פונקציית המטרה. אם מספר הבאטצ' מתחלק ב 8 = כמות הבאטצ'ים בסט האימון-80% מסך הבאטצ'ים חלקי כמות הבאטצ'ים בסט הוולידציה-10% מסך הבאטצ'ים:

קוראת לפונקציה validate על הבאטצ' ועוקבת אחר ערך פונקציית המטרה.

ערך פונקציית המטרה של החזרה מוגדר לממוצע של ערכי פונקציית המטרה על כל באטצ'.

אם ערך פונקציית המטרה על סט הוולידציה לא משתפר:

האימון מופסק.

אחרת אם מספר החזרה מתחלק ב 10: שמור את המודל בעזרת הפונקציה

מוצג גרף של ערך המטרה לפי חזרות.

ערך פונקציית המטרה על סט המבחן מחושב על ידי הפונקציה validate והמודל נשמר.