עמוד שער

תוכן עניינים

תוכן עיניינים

[**תוכן עניינים** 1](#_Toc114947731)

[**הקדמה אישית ותודות** 1](#_Toc114947732)

[**מבוא** 1](#_Toc114947733)

[**סקירת בינה מלאכותית** 1](#_Toc114947734)

[**מושגי בסיס:** 1](#_Toc114947735)

[**למידת מכונה (או למידה חישובית) מונחית**: 1](#_Toc114947736)

[**למידה בהנחיה עצמית:** 1](#_Toc114947737)

[**למידה לא מונחית:** 1](#_Toc114947738)

[**סקירת פייתון** 21](#_Toc114947739)

[**רשימת מבני נתונים בפייתון ומבני הנתונים המקבילים בשפות אחרות:** 21](#_Toc114947740)

[**רמזי סוג – Type Hints:** 21](#_Toc114947741)

[**רב צורתיות (פולימורפיזם):** 22](#_Toc114947742)

[**פעולות קסם – Magic/Dunder Methods:** 22](#_Toc114947743)

[**עצם קריא - Callable object:** 22](#_Toc114947744)

[**קשטן - :Decorator** 22](#_Toc114947745)

[**מחלקת בסיס אבסטרקטית:** 23](#_Toc114947746)

[**פונקציה אבסרקטית:** 23](#_Toc114947747)

[**פעולה סטטית:** 23](#_Toc114947748)

[**מחלקת נתונים – Data Class:** 24](#_Toc114947749)

[**פיתוח מודלי למידה עמוקה בעזרת tensorflow:** 24](#_Toc114947750)

[**המחלקה Module:** 24](#_Toc114947751)

[**המחלקה Layer:** 24](#_Toc114947752)

[**שכבות בנויות מראש:** 24](#_Toc114947753)

[**המחלקה Model:** 25](#_Toc114947754)

[**פיתוח התכונה** 26](#_Toc114947755)

[**האלגוריתם ליצירת טקסט בעזרת קבוצות של טוקנים:** 26](#_Toc114947756)

[**המחלקה :TextGenerator** 26](#_Toc114947757)

[**המחלקה TreeGenerator**: 27](#_Toc114947758)

[**המחלקה SampleGenerator:** 28](#_Toc114947759)

[**אפליקציית הווב** 29](#_Toc114947760)

[הקובץ \_\_init\_\_.py 29](#_Toc114947761)

[הקובץ database.py 29](#_Toc114947762)

[הקובץ **schema.sql** 30](#_Toc114947763)

[הקובץ **auth.py** 31](#_Toc114947764)

[הקובץ model.py 31](#_Toc114947765)

[מימוש הארכיטקטורה לטרנספורמר עם דיקודר בלבד: 32](#_Toc114947766)

[**אימון מודלים:** 33](#_Toc114947767)

[הורדת ויבוא חבילות: 34](#_Toc114947768)

[הגדרות: 34](#_Toc114947769)

[היפר פרמטרים: 34](#_Toc114947770)

[סט הנתונים: 35](#_Toc114947771)

[אוצר המילים: 35](#_Toc114947772)

[הטוקנייזר: 35](#_Toc114947773)

[חתיכת טקסטים ארוכים מדי: 36](#_Toc114947774)

[ריפוד: 36](#_Toc114947775)

[חלוקה לסט אימון, מבחן ווידוי (training, test and validation): 36](#_Toc114947776)

[ניקוי הזכרון: 36](#_Toc114947777)

[**הפרויקט הסופי:** 40](#_Toc114947778)

[**סיכום:** 40](#_Toc114947779)

[**ביבליוגרפיה:** 40](#_Toc114947780)

[**נספחים:** 40](#_Toc114947781)

[**חלקים נבחרים מהקוד:** 40](#_Toc114947782)

[**מימוש האלגוריתם לדגימה בקבוצות:** 40](#_Toc114947783)

[**המחלקה TextGenerator:** 40](#_Toc114947784)

[**המחלקה TreeGenerator:** 42](#_Toc114947785)

[**המחלקה TreeGenerator:** 44](#_Toc114947786)

[**קישורים:** 49](#_Toc114947787)

[העמוד של הפרויקט בגיטהאב 49](#_Toc114947788)

[**דוקומנטציות:** 49](#_Toc114947789)

[**קורסים של אוניברסיטת סטנפורד:** 49](#_Toc114947790)

[**מדריכים:** 49](#_Toc114947791)

הקדמה אישית ותודות

מבוא

סקירת בינה מלאכותית

מושגי בסיס:

למידת מכונה (או למידה חישובית) מונחית – Supervised Machine Learning:

שם כולל לאלגוריתמים שמקבלים נתונים בצורת צמדי קלט ופלט רצוי ומטרתם להגדיל או להקטין ערך של פונקציה מסוימת (פונקציית המטרה) המחשבת את הדמיון או השוני בין פלט האלגוריתם לפלט הרצוי.

לדוגמה: מודל שמסווג תמונות למספר קבוע של מחלקות ידועות.

למידה בהנחיה עצמית – Self Supervised Learning:

למידת מכונה בה האלגוריתם מקבל נתונים שאינם מחלוקים לצמדי קלט ופלט וחלקוה זו נעשית על ידי האלגוריתם עצמו.

לדוגמה – מודל שמקבל טקסט וחוזה את המילה הבאה.

למידה לא מונחית – Unsupervised Learning:

למידת מכונה בה אין פלט רצוי ופונקציית המטרה מחשבת מטרה אחרת.

לדוגמה – רכב אוטונומי שפונקציית המטרה שלו מחשבת את הסיכוי לתאונה.

ווקטור הסתברות:

במדעי הנתונים –הוא ווקטור בו P\_i מייצג את ההסתברות של מחלקה i. כל ההסתברויות בין אפס לאחד וסכום הווקטור 1.

ווקטור לוג'יט – Logit vector:

בלמידה עמוקה –הוא ווקטור בו מוצגות הסתברויות בטווח מינוס אינסןף עד אינסוף. פונקציית Softmax משמשת (בין היתר) להפוך ווקטור לוג'יט לווקטור הסתברות.

טנזור - Tensor:

מערך רב מימדי של מספרים. ניתן לחשוב על טנזור כעל מפה בין מספרים שלמים ואי שליליים לערכים מכל צורה.

לדוגמה:

נניח ויש לנו אתר בו כל פעם שמשתמש נכנס לאתר נוצרת טבלה (מטריצה) בתוך טנזור שמתארת את פעילותו באתר.

יתאר את התא בשורה ועמודה בטבלה שמתארת את הכניסה מספר של המשתמש מספר .

יתאראת העמודה ה בטבלה שמתארת את הכניסה מספר של המשתמש מספר

וככה הלאה.

הדרגה\רמה\מעלה של טנזור מתארת את מספר הפרמטרים המקסימלי של מספרים שאפשר להעביר אליו.

טנזור ברמה אפס הוא מספר (לדוגמה מינוס שבע עשרה) וזאת מכיוון שאי אפשר לגשת לאיבר מספר שלוש של מינוס שבע עשרה.

טנזור ברמה אחת הוא ווקטור או מערך של מספרים (לדוגמה ) וזאת מכיוון שאפשר לגשת לאיבר מספר שתיים במערך אך אי אפשר לגשת לאיבר כלשהו של המספר חמש עשרה.

נשים לב כי טנזור במרה אחת הוא מערך של טנזורים ברמה אפס.

טנזור ברמה שתיים הוא טבלה או מטריצה וזאת מכיוון שטבלה היא מערך של עמודות (או שורות) שהן בעעמן טנזורים מרמה אחת.

באופן כללי, נגדיר טנזור ברמה אפס כמספר וטנזור ברמה ( מספר טבעי) כמערך של טנזורים ברמה .

טוקניזציה – Tokenization:

בניגוד לתוכנות קלאסיות (שלא משתמשות בבינה מלאכותית ו\או למידת מכונה) מודלי שפה אינם מיצגים טקסט כרצף אותיות (מחרוזת) אלא כרצף של מילים, חלקי מילים או צירוף אותיות בעל משמעות (למשל הסיומת ים לציון רבים בעברית או הסיומת ing באנגלית).

כשיוצרים טוקנייזר, הוא מוצא את רצפי האותיות הכי נפוצים בסט האימון ונותן לכל אחד מהם מזהה (טוקן) בצורת מספר שלם ואי שלילי.

לטוקנייזר שתי פונקציות מרכזיות:

Encode: הטוקנייזר מקבל מחרוזת ומחזיר רצף של מזהיי טוקן לפי הסדר בהם הם מופיעים בטקסט.

Decode: הטוקנייזר מקבל רצף של מזהיי טוקן ומחזיר ומתרגם אותם לטקסט.

כל מודל שקולט ו\או מייצר שפה מאומן בהינתן טוקנייזר – הטוקנייזר מוגדר לפני תחילת אימון המודל ולא משתנה אף פעם.

אימון ראשוני למידול שפה סיבתי בעזרת יצירת שפה:

**Generative Pre-Training (GPT) for Causal Language Modeling**

היא שיטה לאימון מודלים שמטרתם ליצור טקסט בהינתן טקסט שמקורה במאמר improving language understanding by generative pre-training.

בהינתן רצף של טוקנים, המודל חוזה כל טוקן בהתבסס על הטוקנים הקודמים בטקסט.

המודל אומנם מקבל את כל הטוקנים אבל בעזרת מסכת הסתכלות קדימה (look ahead mask), כל טוקן תלוי בטוקנים שבאו לפניו בטקסט בלבד.

המודל לומד לחזות איזה טוקן נמצא בכל מקום (מתוך רשימה ידועה של טוקנים).

כאשר:

i המזהה של הטוקן שנמצא במקום ה i.

היא ההסתברות של הטוקן במקום ה t+1 להיות הטוקן שהמזהה שלו i.

seq len הוא אורך הרצף שהמודל מקבל.

המודל מצמם את ה categorical entropy הממוצע לרצף שאותו הוא מנסה לחזות.

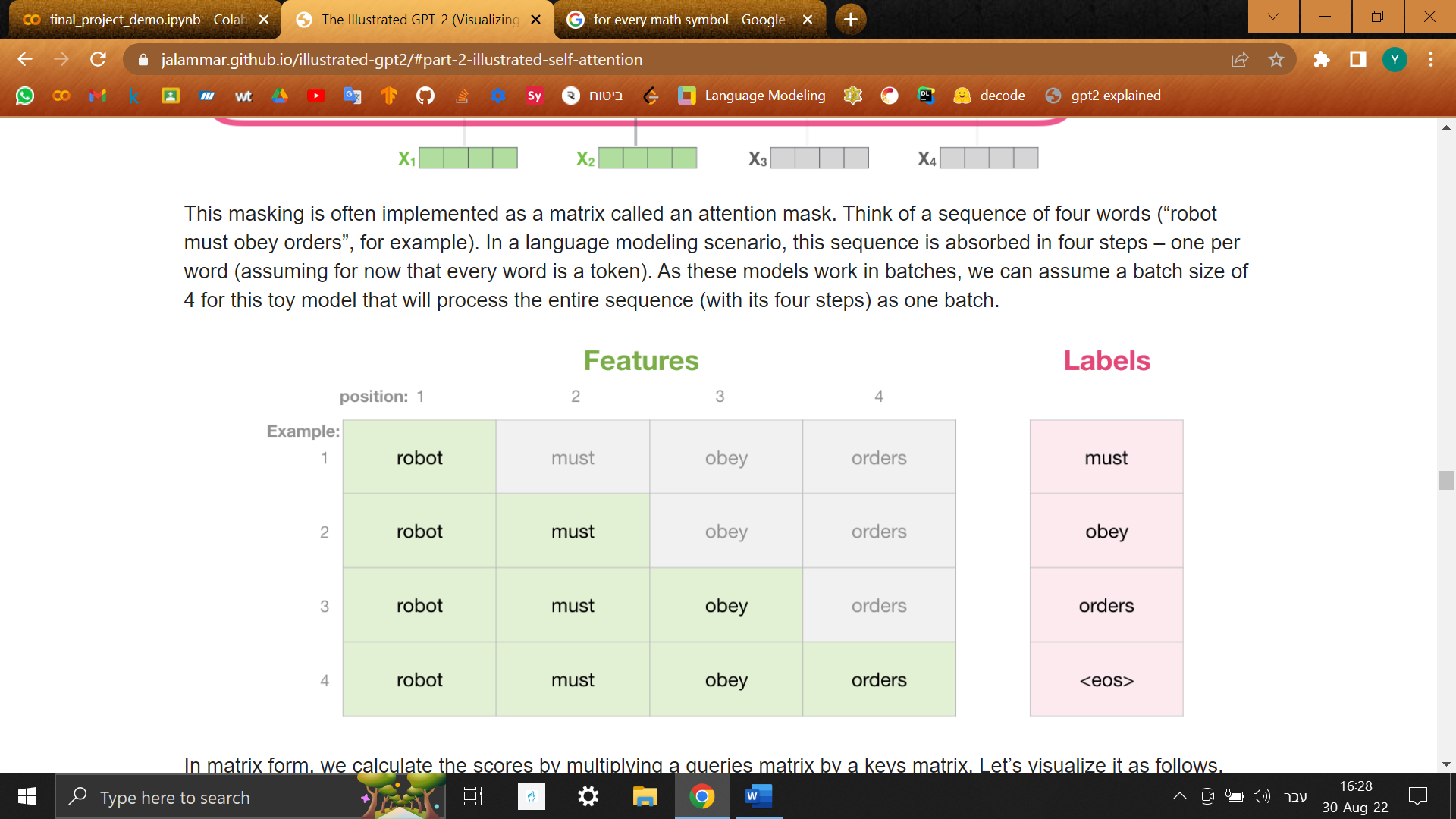
שימו לב שהמודל לא מתייחס לטוקן הראשון, זאת מכיוון שאין אפשרות לחזות את הטוקן הראשון בהתבסס על הטוקנים הקודמים (כי אין טוקנים שמגיעים לפני הטוקן הראשון).

משימה זו היא משימת לימוד בהנחיה עצמית:

המודל לומד לחזות או ליצור חלק אחד מהדוגמה בהינן חלק אחד מהדוגמה לכל דוגמה בסט הנתונים.

חלוקה של המשפט "robots must obey orders"

בכל שורה, המודל צריך לחזות מה יהיה הטוקן בתא האדום בהתבסס על הטוקנים בתאים הירוקים (הטוקנים בתאים האפורים ממוסכים).



למידה רב שלבית – Transfer Learning:

המשימה של חיזוי הטוקן הבא בטקסט איננה חשובה בפני עצמה. האימון למשימת מידול שפה טבעית הוא אימון ראשוני שאחריו מגיע אימון למשימה ספציפית (למשל: סיכום טקסט, תרגום בין שפות, מענה בשירות לקוחות ועוד משימות רבות).

הרעיון מאחורי אימון כללי שלאחריו אימון למשימה ספציפית (down stream task) הוא שהמודל לומד להבין שפה כללית וליצג טקסטים באופן כללי – מה שיעזור מאוד באימון למשימות ספציפיות עם פחות נתונים ופחות כוח חישוב.

גישה זו היא סטנדרטית באקדמיה ובתעשייה בשנים האחרונות.

האימון החוזר על משימה ספציפית נעשה כמו האימון הכללי – בלי התייחסות לקלט ופלט וזאת על מנת שהמודל ילמד לעבד את המידע שמופיע בקלט.

שימוש במודל ליצירת טקסט:

בהינן טקסט שנקלט מהמשתמש:

תחילה המודל מתורגם לרצף של טוקנים על ידי הטוקנייזר.

המודל תחילה מקבל את הקלט ולאחריו טוקן רנדומלי.

המודל חוזה את הטוקן שנמצא במקום האחרון ברצף בהתבסס על הטוקנים שלפניו ולכן זאת התחזית של המודל לטוקן הראשון בפלט.

הטוקן הבא נדגם (ראה חלק על שיטות דגימה) מההסתברות שהמודל יצר.

לאחר מכן המודל מקבל את הקלט ואת הטוקן הנגדם ומוצא את הטוקן הבא וכך הלאה.

מספר הטוקנים שנוצרים יכול להיקבע בכמה דרכים:

1. על ידי המשתמש.
2. מספר הטוקנים הנוצרים יהיה שווה למספר הטוקנים בקלט.
3. על ידי מודל שמקבל קלט וחוזה את אורך הפלט.
4. מתוך התפלגות של הטוקנים הפלטים בסט הנתונים.

One Hot representetion:

יצוג one hot ממיר אינדקס של מחלקה (במקרה שלנו טוקן) שהוא מספר שלם ואי שלילי לווקטור בינארי שאורכו מספר המחלקות בנתונים (במקרה שלנו מספר הטוקנים שהטוקנייזר שומר במילון).

יצוג one hot הוא בעצם ווקטור של הסתברות של תוצאה ידועה מראש ולכן אנחנו רוצים שההסתברות שהמודל יחזה תיהיה כמה שיותר קרובה ליצוג one hot .

|  |
| --- |
| def token\_to\_one\_hot(token\_id: int, num\_tokens: int) -> List[int]:  *"""Returns a one-hot list for the class class\_id with*  *num\_classes classes"""* ans: List[int] = [0] \* num\_tokens  ans[token\_id] = 1  return ans   def sequence\_to\_one\_hot(sequence: List[int],  num\_tokens: int) \  -> List[List[int]]:  *"""Creats a one-hot matrix  for the given sequence of tokens"""* return [token\_to\_one\_hot(token\_id, num\_tokens)  for token\_id in sequence] |

לדוגמה:

אם המילון של הטוקנייזר הוא {אני: 0, אוהב: 1, גלידה: 2} אז המחזרות "אני אוהב גלידה" תומר לרצף הטוקנים [2, 1, 0] ואז לייצוג one hot:

טרנספורמר עם דיקודר בלבד – Decoder Only Transformer:

היא הארכיטקטורה בה השתמשו כותבי המאמר

improving language understanding by generative pre-training

במודל טרנספורמר מבוסס דיקודר בלבד בלוק הדיקודר הראשון מקבל את האמבדינג והקידוד המקומי של הקלט. שאר בלוקי הדיקודר מקבלים את הפלט של בלוק הדיקודר שנמצא לפניהם במודל.

הפלט של בלוק הדיקודר האחרון מוכפל במטריצת אמבידנג משוחלפת ליצירת מטריצת לוג'יט.

על מטריצת הלוג'יט מופעלת פעולת שיוצרת את מטריצת ההסתברות שבה משתמשים על מנת ליצור רצף של טוקנים.

המודל מקבל מטריצה של ייצוג רצף הטוקנים בצורת one hot ומחזיר חיזוי בצורת לוג'יט של איזה טוקן יהיה בכל מקום ביחס בהתבסס על הטוקנים שבאים לפניו בטקסט.

בעבודה שלי אשתמש במושג מטריצת הסתברות לתאר מטריצה בה היא ההסתברות של הטוקן ה ברצף להיות ה הטוקן שהמזהה שלו הוא i בהתבסס על הטוקנים הקודמים (אפס עד t). ומטריצת לוג'יט לתאר מטריצה הבנויה מווקטורי לוג'יט כמו שמטריצת הסתברות בנויה מווקטורי הסתברות.

בלוק דיקודר Decoder Block:

בלוקי דיקודר הם שכבות עם תתי השכבות הבאות:

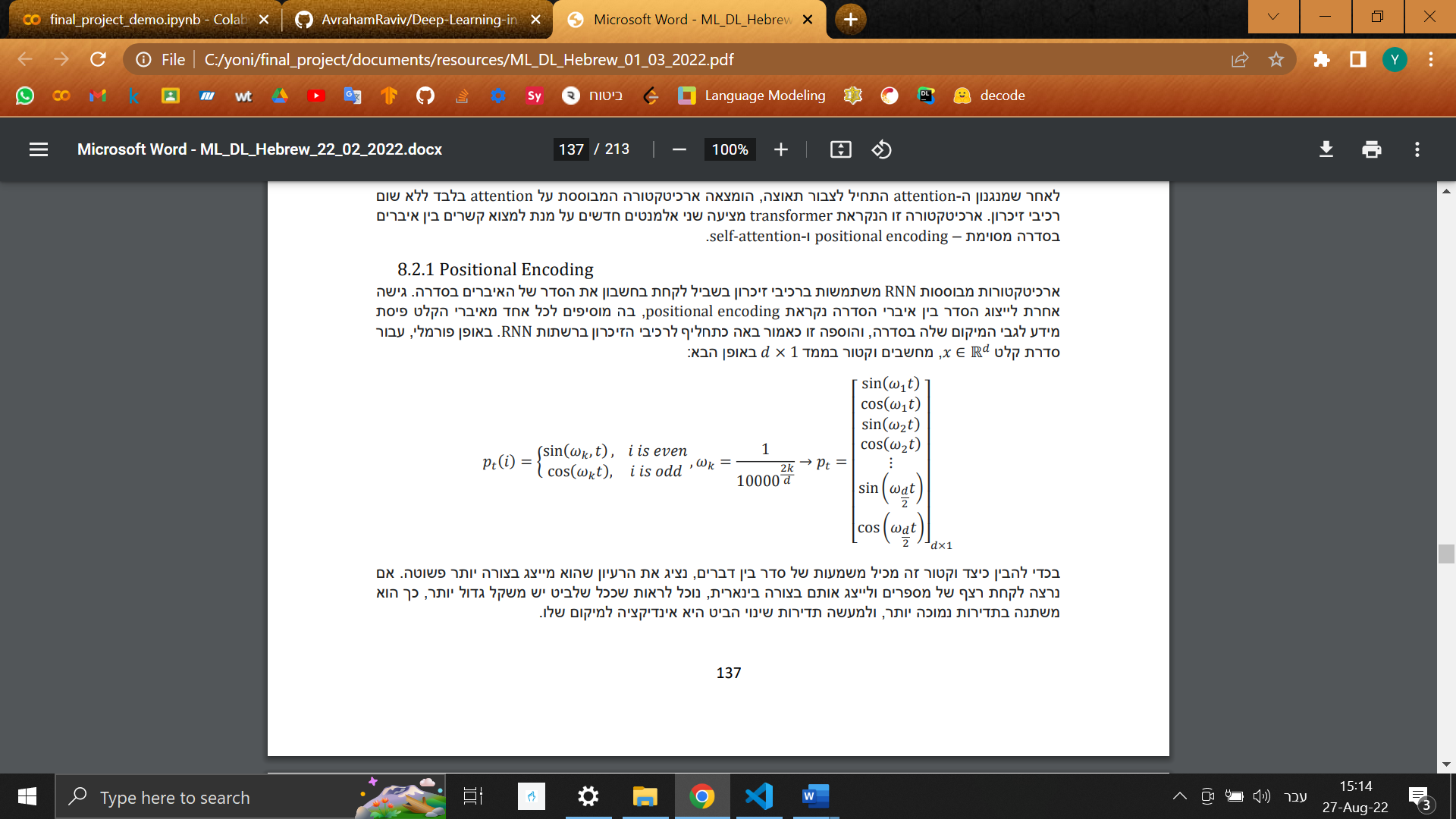
1. צומת לב עצמית רב ראשית עם מסכת הסתכלות קדימה
2. חיבור ונורמליזציה
3. רשת מחוברת לגמרי
4. חיבור ונורמליזציה

קידוד מיקומי – Positional Encoding:

מכיוון שהטרנספורמר אינו מתייחס באופן שונה לווקטורים במיקומים שונים בתוך רצף, יש צורך בהוספת מידע לכל טוקן בנוגע למיקומו במשפט.

שיטה בה מוסיפים לכל אחד מאיברי הקלט פיסת מידע (במקרה שלנו טוקן) לגבי המיקום שלה ברצף באופן פורמלי, עבור סדרת קלט

, מחשבים וקטור במימד x 1 באופן הבא:



אמבדינג – Embedding:

אמבדינג היא שכבה המקבלת רצף טוקנים ביצוג one hot וממירה כל יצוג one hot בווקטור בעל משעות סמנטית שאינו תלוי בטוקנים אחרים.

השכבה לומדת מטריצה בגודל מספר הטוקנים המוכרים למודל x שאפשר לדמיין אותה בתור טבלה בה כל שורה היא מילה שהמודל מכיר וכל עמודה היא תכונה סמנטית שיכולה להיות למילה או לצירוף אותיות בעל משמעות.

פעולת האמבדינג היא מכפלה של מטריצת הרצפים ביצוג one hot במטריצת האמבדינג.

קיימות מגוון שיטות ליצירת מטריצת אמבדינג שאינן בעזרת למידה עמוקה.

הכפלה במטריצת אמבדינג משוחלפת – Transposed Embedding:

*את התוצאה של הדיקודר אנחנו מכפילים במטריצת האמבדינג המשוחלפת.*

*נזכור כי הכפלה במטריצה משוחלפת היא הפעולה ההפוכה להכפלה במטריצה המקורית.*

אינטואיציה:

הדיקודר מחזיר את מטריצת אמבדינג של המילים במקום השני עד המקום seq len + 1 (לפני נורמליזציה).

הכפלה במטריצת אמבדינג היא תרגום של הסתברות לאמבדינג.

הכפלה במטריצת אמבדינג משוחלפת היא תרגות של אמבדינג להסתברות.

הפונקציה softmax:

*softmax() מוגדרת לפי הנוסחה:*

*תפקיד הפונקציה לקחת ווקטור עם ערכים בין מינוס אינסוף לאינסוף ולהפוך אותו לווקטור שערכיו בין אפס לאחד וסכומו אחד.*

*אחד השימושים שלה הוא חישוב ווקטור הסתברות מתוך ווקטור לוג'יט.*

*הווקטור שהפונקציה מחזירה הוא באותו גדול של הווקטור שהפונקציה מקבלת.*

*הפונקציה מחזירה ווקטור שסכומו אחד וכל איבריו בין אפס לאחד .*

*שימו לב גם ש .*

*פעולת softmax על מטריצה בציר מסוים היא פעולת softmax על כל ווקטור בציר.*

*כלומר: פעולת softmax בציר 1 היא פעולת softmax לכל שורה במטריצה.*

טמפרטורה - Temprature:

*טמפרטורה היא מספר חיובי שנבחר בעת יצירת טקסט ומטרתו להשפיע על התפלגות ווקטור ההסתברות שהמודל מיצר בדגימה שאיננה לפי הסתברות מקסימלית.*

*כל איבר במטריצת הלוג'יט מחולק בטמפרטורה לפני ככה ש:*

*אינטצואיציה:*

*הקטנת הטמפרטורה גורמת לאיזון ההסתברויות של הטוקנים ככה שלכל הטוקנים תיהיה הסתברות דומה יותר והגדלת הטמפרטורה גורמת לחוסר איזון בהסתברויות ככה שלטוקנים יהיו הסתברויות שונות יותר.*

*דוגמה: מציאת ווקטור ההסתברות של ווקטור הלוג'יט [1, 1-] עם טמפרטורה של אחד, שתיים וחצי.*

*טמפרטורה איינה שימושית במהלך אימון המודל מיכוון שאם נשנה את הטמפרטורה, המודל ילמד ליצר ווקטור לוג'יט בו כל ערך מוכפל באחד חלקי הטמפרטורה.*

מסכת הסתקלות קדימה look ahead mask:

*היא מטריצה בגודל seq\_len x seq\_len שמטרתה לגרום לכך שטוקנים לא יושפעו מהטוקנים שלפניהם.*

*כאשר seq\_len הוא אורך הרצף.*

*המסכה נוצרת לפי הקוד:*

|  |
| --- |
| def create\_look\_ahead\_mask(seq\_len: int) -> List[List[int]]:  *"""Returns a look ahead mask for the given length.  input: seq\_len: int  Returns: list of list of 0s and 1s"""* answer: List[List[int]] = [[0] \* seq\_len] \* seq\_len  for i in range(seq\_len):  for j in range(seq\_len):  if j > i:  answer[i, j] = 1  return answer |

*או לפי ההגדרה המתמטית לכל איבר:*

*מסכבת הסתכלות קדימה תמיד תיהיה מטריצה ריבועית בה האלכסון וכל האיברים מתחתיו אפס וכל האיברים מתחת לאלכסון 1.*

*דוגמה: מסכת הסתכלות קדימה לרצף באורך 3:*

*אפשר לחשוב על המסכה בטור טבלה בה האיבר בשורה j ועמודה i עונה על השאלה:*

*0 אם הטוקן במקום מושפע מהטוקן במקום ו - 1 אחרת.*

צומת לב עצמית ממוסכת בעזרת מכפלה סקלרית:

**Scaled Dot-Product** **Masked Self Attention**

המטרה של צומת הלב היא לקחת ייצוג של רצף של טוקנים ולתת לכל טוקן יצוג התלוי בטוקנים שמלפניו. כל טוקן בכל שלב במודל מיוצג על ידי ווקטור באורך (קיצור של attention dim).

נגדיר:

seq len = אורך הקלט לטרנספורמר (מספר חיובי ושלם)

*= המימד החבוי – תכונה של השכבה.*

*הן מטריצות פרמטרים הנלמדים על ידי המודל.*

*הפונקציה מקבלת מטרציה*

*הפעולה מחזירה מטריצה באותו גודל של המטריצה שהיא מקבלת.*

*הגדרה מתמטית לפעולת צומת הלב:*

***אינטואיציה:***

*המצב החבוי הוא מטריצה שמכילה רצף של ווקטורים בה כל ווקטור במקום i מיצג את המשמעות הסמנטית של הטוקן במקום i בקונטקסט של המשפט.*

*אם נדמיין כל מצב חבוי של טוקן (ווקטור בגודל ) כנקודה במרחב, הכפלתו במטריצת פרמטרים תשנה את מערכת הצירים בו הווקטור נמצא למערכת צירים שמייצגת בצורה יותר מדויקת את הקשרים שבין הטוקנים השונים. בשתי מערכות הצירים מימדים.*

*השאילתה (query) של טוקן היא ווקטור שקרוב לווקטורים שיכולה להיות להם השפעה על משמעות הטוקן.*

*המפתח (key) הוא ההשפעה של הטוקן על טוקנים אחרים.*

*הערך (value) הוא התוכן של הטוקן.*

*הפעולה יוצרת מטרציה (קיצור ל dot procut – המונח לערך שסלארי באנגלית) בגודל seq len x seq len בה הוא תוצאת המכפלה הסקאלרית בין הווקטור של שמיצג את הטוקן במקום ה i בשאילתה (לאחר הטרנספורמציה) לווקטור שמיצג את הטוקן במקום ה j במפתח (לאחר הטרנספורמציה) שמייצג את ההשפעה של הטוקן במקום i על הטוקן במקום j.*

*החלוקה של כל איבר מטריצה ב היא נורמליזציה ואינה הכרחית. אם לא נחלק הפעולה תתבצע באופן דומה מאוד והמודל יעבוד בצורה מאוד דומה. הנורמליזציה משפרת קלות את ביצועי המודל.*

*נזכור שמכפלה סקאלרית בין שני ווקטורים מייצגת את הדמיון ביניהם – ככל ששני ווקטורים יותר דומים – המכפלה הסקלארית שלהם יותר גדולה ולהפך. מכפלה סקלארית יכולה להיות חיובית או שלילית. מכפלה סקלארית לא יכולה להיות יותר גדולה מאורך הווקטור הארוך יותר בריבוע.*

*הפעולה :*

*המסכה מוכפלת במינוס מיליארד ככה שכל ערך שהיה אחד במסכה המקורית הוא מינוס מיליארד במסכה המוכפלת וכל ערך שהיה אפשר נשאר אפס ובאופן פורמלי:*

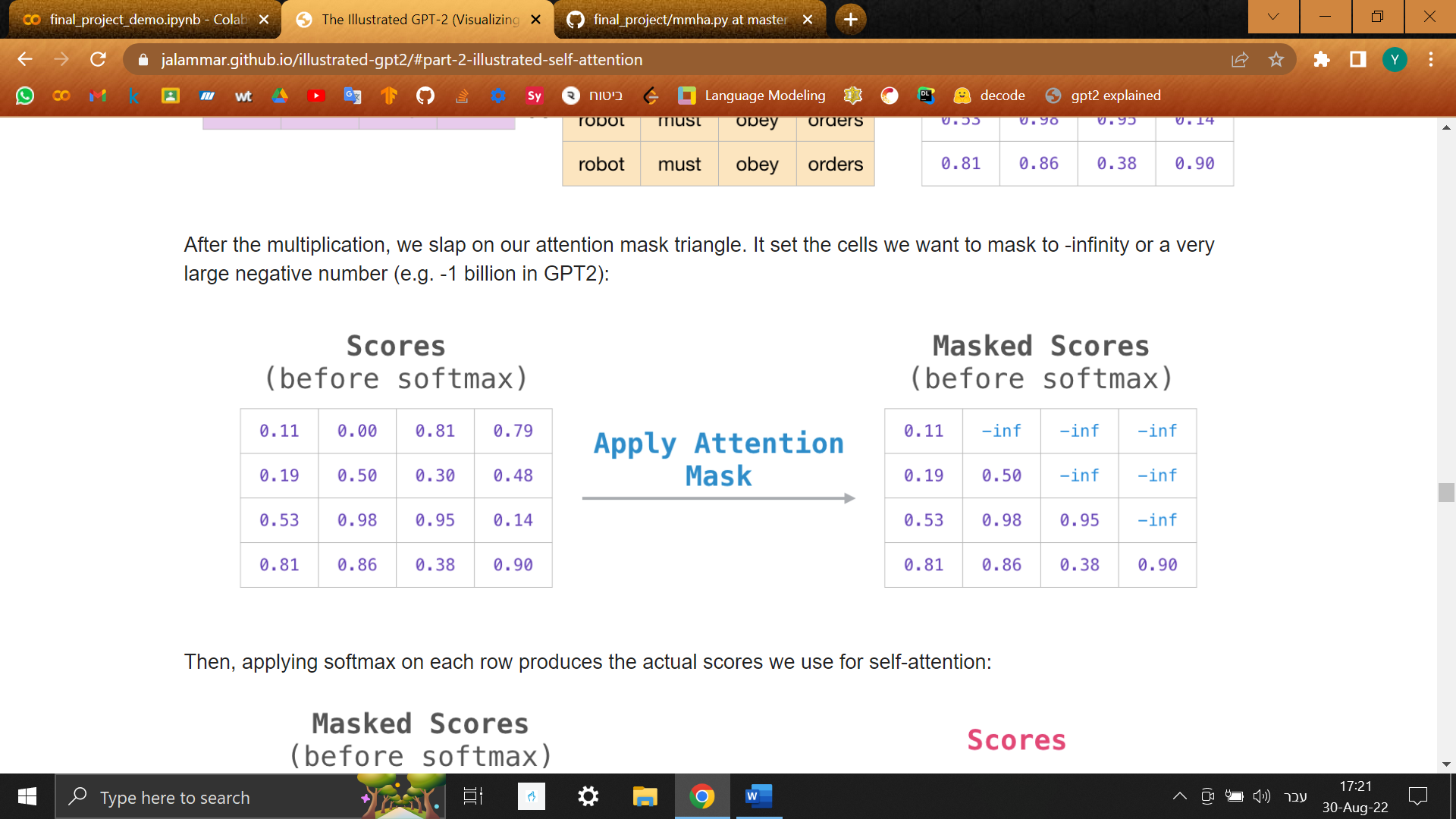
*את תוצאת המכפלה מחסרים ממטריצת הדמיון ככה שהדמיון בין*

*אינטואיציה:*

*במטריצה יש קשרים דו צדדיים בין כל הטוקנים כלומר כל טוקן משפיע על כל הטוקנים הסובבים אותו אבל זאת בעיה כי אנחנו רוצים לחזות כל טוקן מהתבסס על הטוקנים שקדמו לו בלבד.*

*אנחנו רוצים שההשפעה של טוקנים על טוקנים שבאים לפניהם ברצף תיהיה קטנה ככל הניתן.*

*המסכה גורמת להשפעה של טוקנים על הטוקנים שבאים לפניהם ברצף להיות קטנה מאוד – בערך מינוס מיליארד.*



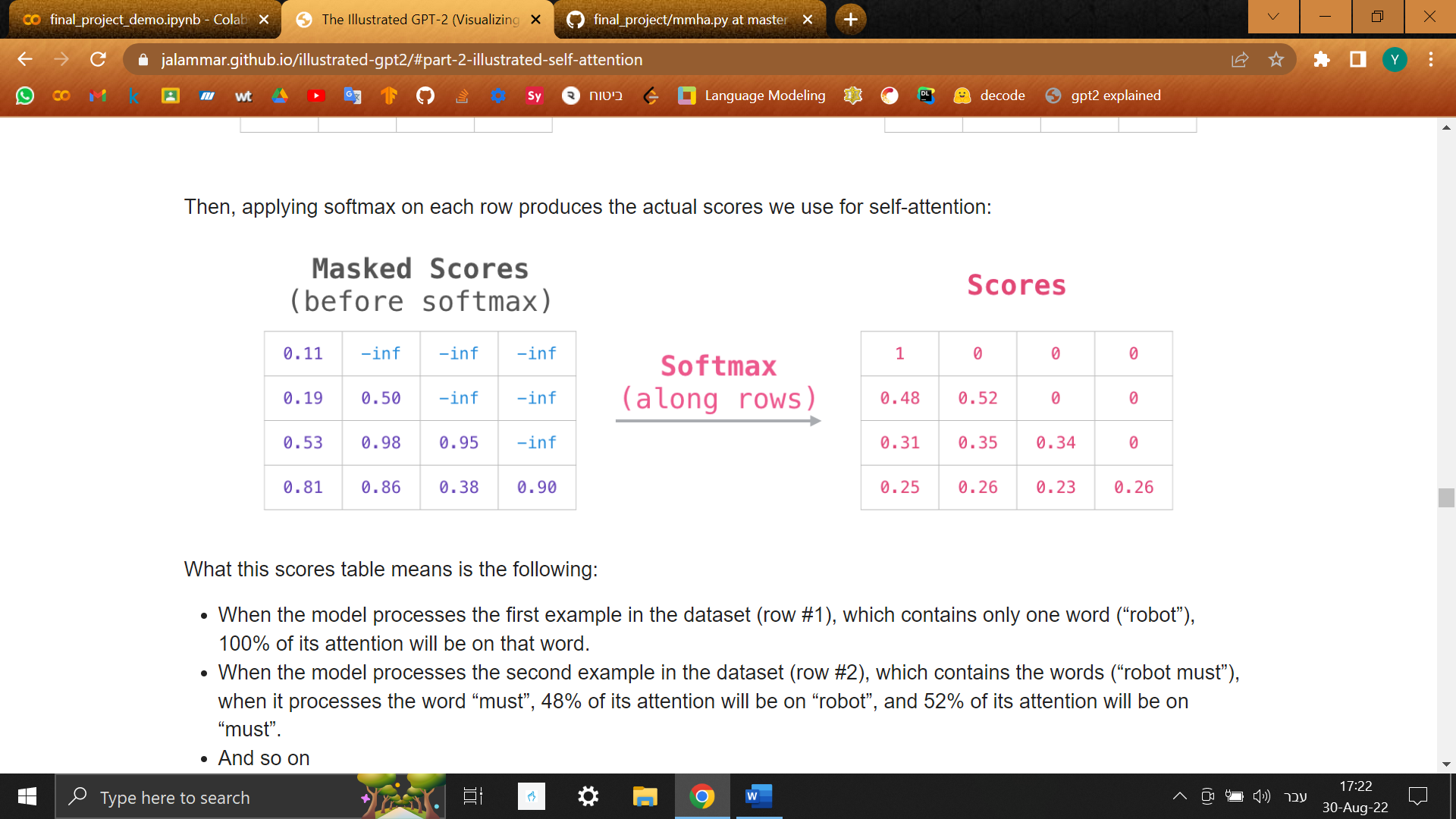
*הפעולה :*

*המטריצה ASM היא טבלה בה הוא האיבר בעמודה i ובשורה j המיצג את ההשפעה של הטוקן במקום ה i על הטוקן במקום ה j ופעולת softmax מנרמלת את הטבלה ככה שסכום כל שורה הוא 1.*

*אנחנו רוצים שההשפעה של כל טוקן על הטוקנים לפניו תיהיה קרובה מאוד ל 0.*

* *הערה: בדרך הכלל המספר הקרוב לאפס מתעגל לאפס.*

*וזאת מכיוון ש .*



*נגדיר את הערך הווקטורי של טוקן כערך שלו כפי שמיוצג במטריצה V.*

*הפעולה יוצרת מטריצת יצוג חבוי ככה ש כל טוקן מיוצג על ידי הממוצע המשוכלל של הערך הווקטורי של כל הטוקנים המשפיעים עליו.*

*אם נשתמש בדוגמה למעלה: הווקטור של הטוקן השני במטריצה יהיה 0.48 כפול הווקטור של הטוקן הראשון במטריצה ועוד 0.52 כפול הווקטור של הטוקן השני במטריצה .*

צומת לב רב-ראשית – Multi Head Attention:

*בהינתן אותם פרמטרים שמקבלת פעולת צומת לב ועוד פרמטר – מספר הראשים (num\_heads)*

*(המימד החבוי של המודל חייב להתחלק ב מספר הראשים).*

*הערך החבוי האחרון והערך לפני טרנספורמציה מפוצלים ככה ש המימדים שלהם משתנים מ :*

*Seq\_len x ל num\_heads x seq\_len x ()*

*ככה שלכל ראש יש ערך חבוי אחרון וערך לפני טרנספורמציה אחרים:*

*LHV\_of\_head\_i = LHV[i, :, :]*

*PTV\_of\_head\_i = PTV[i, :, :]*

*כל ראש מחשב את צומת הלב עם הערך החבוי האחרון והערך לפני טרנספורמציה שלו ולומד פרמטרים אחרים.*

*לפיצול לראשים שתי מטרות:*

1. *להקטין את זמן החישוב:*

*החישוב של צומת לב בין מטריצות קטנות לוקח פחות זמן והחישוב של כל הראשים מתבצע במקביל.*

1. *ללמוד דברים אחרים:*

*בפיצול המטריצות כל ווקטור המתאר טוקן מפוצל ככה ש:*

*vector\_size = d\_k / num\_heads*

*start\_index = i \* vector\_size*

*end\_index = (i + 1) \** *vector\_size*

*vector\_for\_head\_i = original\_vector[start\_index:end\_index]*

*נזכור שכל איבר בווקטור מייצג תכונה סמנטית של הטוקן ולכן כל ראש מתייחס לתכונות סמנטיות אחרות של הטוקנים.*

מסכת ריפוד - Pad/Padding Mask:

*מסכת ריפוד מאפשרת את ביצוע פעולת צומת הלב על קבוצה (batch) של רצפים באורכים שונים במקביל. לכל רצף נוסיף טוקן מיוחד - ריפוד מספר פעמים בסוף הטקסט ככה שאורך כל רצף יהיה כאורך הטוקן הארוך ברצף.*

*נגדיר = האורך של רצף הארוך ביותר, = מספר הרצפים בקבוצה (באטצ').*

*ניצור טנזור בינארי לפי הכלל:*

*. אם הטוקן במקום i+1 מושפע מהטוקן במקום j וגם הטוקן במקום j הוא אינו ריפוד ו - 1 אחרת.*

*לכל רצף במקום ה i, שכבת צומת הלב משתמשת ב במקום ב וככה נוצר מצב בו ערך כל טוקן תלוי בטוקנים שלפניו* ***וגם*** *אינם טוקן הריפוד.*

רשת מחוברת לגמרי – Fully Connected Feed Forward Network:

*רשת מחוברת לגמרי היא שכבה שמורכבת משלוש תת שכבות:*

*שכבה דחוסה עם גודל קלט*  *וגודל פלט feed forward depth (הייפר פרמטר של המודל)*

*אקטיבציית ReLU.*

*שכבה דחוסה עם גודל קלט feed forward depth וגודל פלט .*

שכבה דחוסה:

*לשכבה דחוסה שתי תכונות: גודל הקלט (n) וגודל הפלט (m).*

*היא לומדת מטריצת פרמטרים ו-ווקטור פרמטרים b בגודל m.*

*שכבה דחוסה מקבלת ווקטור שאורכו גודל הפלט ומבצעת עליה את הפעולה הלינארית:*

*הגדרה לכל איבר:*

*אינטואיציה:*

*נחשוב על הווקטור x בתור נקודה בתוך מערכת צירים, הכפלה של ווקטור במטריצה היא ייצוג של הווקטור במערכת צירים אחרת והוספה של הווקטור b היא הזזה של הווקטור בגודל וכיוון קבוע.*

פונקציית ReLU:

*הפונקציה פועלת על כל איבר בטנזור ונוחסתה:*

*הנגזרת של הפונקציה היא:*

ריפוד - Padding:

*בלמידה עמוקה אנחנו הרבה פעמי רוצים לשלוח למודל כמה דוגמאות בו זמנית.*

*עד עכשיו, הנחנו שהטרנספורמר מקבל רצף אחד אך קיימת שיטה לשלוח לטרנספורמר כמה דוגמאות בו זמנית וזאת על ידי ריפוד.*

*נוסיף לסוף כל רצף טוקן מיוחד הנקרא ריפוד ככה שכל הרצפים באותו אורך.*

*לכל רצף ניצור מסכת ריפוד – ווקטור בינארי המוגדר לפי:*

בלבול – Preplexity:

*בלבול היא שיטה להעריך כמה המודל בטוח בתחזיות שלו כשהוא יוצר רצף.*

*הגדרה פורמלית:*

*כאשר הוא אורך הרצף שנחזה, היא ההסתברות של הטוקן במקום i (בהינתן הטוקנים הקודמים אליו).*

דגימה – Sampling/Decoding:

*נניח ויש לנו מודל שמקבל רצף של טוקנים ומחזירה את ווקטור ההסתברות של הטוקן הבא, יש לנו רצף של טוקנים שאנחנו רוצים שהמודל ישלים ויש לנו אורך מסוים בו אנחנו רוצים שהטקסט המיוצר יהיה.*

*דגימה היא תהליך הוספת הטוקנים לרצף. ישנן שיטות שונות לדגימה:*

דגימה לפי הסתברות מקסימלית – argmax/greedy sampling:

*נתחילה נקבל את ווקטור ההסתברות של הטוקן הראשון על ידי הזנת הרצף המקורי למודל.*

*ניקח את הטוקן שההסתברות שלו הכי גבוהה ונוסיף אותו לרצף.*

*נחזור על התהליך עד שאורך הרצף הוא האורך הרצוי.*

*יתרונות:*

* *השיטה פשוטה וקלה לישום.*
* *אין את הסיכון של לדגום טוקן עם הסתברות ממש נמוכה.*

*החסרון:*

* *המטרה של רוב הטקסטים אינה להיות כמה שיותר צפוים אלא להעביר מידע.*

*אם כל מילה תיהיה המילה הצפויה ביותר בהינתן המילים שלפניה אז כמות המידע שעובר תיהיה קטנה מאוד.*

* *שיטה זו נוטה ליצר את הטוקנים שמופיעים הרבה בסט האימון של המודל (בדרך כלל מילות קישור).*
* *שיטה זו נוטה לייצר טוקנים שחוזרים על עצמם.*
* *אין שליטה: אין פרמטר שאנחנו יכולים לשנות אחרי אימון המודל על מנת לשנות את הטקסט שהמודל מייצר.*

דגימה רנדומלית מתוך הסתברות:

*ההבדל בין דגימה מתוך הסתברות לדגימה לפי הסתברות מקסימלית היא שבדגימה רנדומלית במקום לדגום את הטוקן שהסתברותו הכי גבוהה, נדגום ברנדומליות לפי ההסתברות שהמודל חזה.*

*היתרון:*

* *דגימה מגוונת יותר.*

*החסרונות של השיטה:*

* *דגימה של מילים עם הסתברות נמוכה מאוד.*
* *חוסר התאמה למטרה: המודל מאומן לחזות הסתברות כמה שיותר גבוהה לטוקן הנכון ובפונקציית המטרה אין התייחסות להסתברות של הטוקנים האחרים.*
* *תלות ברכיב רנדומלי.*

דגימה מתוך k הטוקנים שהסתברותם הכי גבוה – Top k Sampling:

*נבחר מספר שלם וחיובי k קטן או שווה לכמות הטוקנים שהמודל מכיר.*

*בהינתן ווקטור הסתברות:*

*נמצא את k הטוקנים שהתברותם הגדולה ביותר.*

*נקבע את ההסתברות של שאר הטוקנים לאפס.*

*נחלק את ההסתברות של כל טוקן בסכום של הווקטור החדש (על מנת לקבל ווקטור שסכומו אחד).*

*ונדגום מהווקטור שנוצר.*

*יתרונות של השיטה:*

*ברוב המקרים, כל ההסתברויות השונות מ – 0.*

*יתרונות:*

* *נותנת חסם מלמטה להסתברות של הטוקנים האופן שתלוי בווקטור ההסתברות.*
* *מאפשרת דגימה באופן שמתייחס לטוקנים שהסתברותם גבוהה בלבד.*

*חסרונות:*

* *לא מונעת לגמרי את האפשרות לדגום טוקנים שהסתברותם נמוכה.*
* *מגדירה טוקן עם הסתברות גבוהה כ: טוקן שיש פחות מ k טוקנים שהסתברותם גבוהה מהסתברותו – הגדרה שיכולה ליצור בעיות במקרי קצה.*

*הערה: דגימה מתוך k הטוקנים שהסתברותם הכי גבוה כאשר k שווה אחד היא דגימה לפי הסתברות מקסימלית.*

דגימה מתוך הטוקנים שסכום הסתברותם <= p – Top p Sampling:

*נבחר מספר p בין אפס ואחד.*

*אם p קטן או שווה להסתברות של הטוקן שהסתברותו הגדולה ביותר נבחר בו (בטוקן שהסתברותו הגודלה ביותר).*

*אחרת*

*ניצור ווקטור הסתברות חדש בו כל ההסתברויות 0.*

*נעבור על ווקטור ההסתברויות המקורי לפי סדר:*

*נוסיף לווקטור החדש טוקנים כל עוד סכום ההסתברות לא עולה על p.*

*ולאחר מכן נסכום את הווקטור החדש.*

*נעבור על הווקטור ונחלק את ההסתברות של כל טוקן בסכום של הווקטור החדש (על מנת לקבל ווקטור שסכומו 1).*

*ונדגום מהווקטור שנוצר באופן רנדומלי.*

*יתרונות (לעומת דגימה מתוך k הטוקנים שהסתברותם הכי גבוה):*

* *ההגדרה של הסתברות גבוהה יותר עמידה בפני מקרי קצה.*
* *השליטה בבחירת הטוקנים יותר טובה.*

*חסרון:*

* *אין חסם המנוע בחירת טוקנים בעלי הסתברות נמוכה.*

חיפוש עץ – :Beam Search

*נגדיר את ההסתברות של רצף T בו l טוקנים :*

*נבחר מספר חיובי שלם קטן או שווה לכמות הטוקנים שהמודל מכיר ונקרא לו רוחב העץ .*

*בכל שלב בחיפוש, נבחר את (רוחב העץ) הטוקנים שהסתברותם הכי גבוהה תוך מעקב על ההסתברות של הרצף.*

*לכל טוקן שבחרנו, נחזבחר את (רוחב העץ) הטוקנים שהסתברותם הכי גבוהה תוך .*

*נחזור על התהליך עד שנגיע למספר הטוקנים הרצוי.*

*לאחר מכן נבחר את הרצף שהסתברותו הגבוהה ביותר.*

*חסרון:*

*סיבוכיות זמן ריצה גדולה כשמייצריםרצפים ארוכים.*

*כאשר n הוא מספר הטוקנים שאנחנו רוצים לחזות.*

*אנחנו בעצם יוצרים עץ בו לכל הורה w ילדים ויצירה של ילדים נעשית באמצעות קיראה למודל ולכן אנחנו קוראים למודל פעמים וזמן הריצה של מודל מסוג טרנספורמר הוא ולכן זמן הריצה של הדגימה הוא בדיוק זהו זמן ריצה גדול יותר מזמן ריצה אקספנונציאלי.*

חיפוש עץ עם אילוץ על סכום ההסתברות:

*כמו חיפוש עץ רק שבמקום לבחור את w הטוקנים שהסתברותם הכי גדולה, בוחרים את הטוקנים שהסתברותם הכי גדולה ככה שסכום ההסתברויות לא עולה על p (בדומה ל דגימה מתוך הטוקנים שסכום הסתברותם <= p)****.***

*אם המודל חוזה הסתברות שווה לכל טוקן, יבחרו טוקנים כאשר v הוא מספר הטוקנים שהמודל מכיר ולכן פונקציית זמן הריצה היא ולכן זמן הריצה הוא אקספוננציאלי.*

סקירת פייתון

בחלק זה אסקור נושאים מתקדמים בתכנות ונושאים בשפת פייתון בהם השתמשתי בעבודה.

אני לא אסקור נושאים בסיסיים (כגון ירושה וייבוא) מכיוון שהם אינם נושא הפרויקט.

רשימת מבני נתונים בפייתון ומבני הנתונים המקבילים בשפות אחרות:

רשימה (list) – מערך דינמי. מושגים מקבילים: dynamic array, array list.

טאפל (tuple) – מערך שאינו ניתן לעריכה. מושג מקביל: imutable array.

מילון (dict) – מפה\מפת גיבוב. מושגים מקבילים: map, hash map.

סט (set) – קבוצה – רשימה בה אף איבר לא חוזר על עצמו. מושג מקביל: hash set.

רמזי סוג – Type Hints:

מערכת הטיפוסים בפייתון היא דינמית כלומר הטיפוסים בפייתון נקבעים בזמן הריצה ולא בזמן כתיבת התוכנית או הקומפילציה.

פעמים רבות נרצה לדעת מה הטיפוס של ביטוי משתנה או פונקציה בזמן קריאת הקוד ולכן יש צורך ברמזי סוג.

רמזי סוג הם הדרך המקובלת להוסיף מידע על סוג של נתונים (במקום הערות).

חשוב לציין כי רמזי סוג אינם מחייבים וכי השמת ערך במשתנה כאשר סוג הערך שונה מהרמז לסוג המשתנה אינו גורר שגיאה.

הספרייה הסטנדרטית typing מכילה כלי עזר לרמזי סוג.

רב צורתיות (פולימורפיזם):

עיקרון במדעי המחשב לפיו יש לקרוא למתודות אשר מבצעות את אותו תפקיד בשם זהה.

לא משנה איזה טיפוס נעביר לפונקציה, היא עושה את ההתאמות הנדרשות.

לדוגמה פונקציית הקסם str

פעולות קסם – Magic/Dunder Methods:

פעולות קסם הן פעולות שמורות בשפת פייתון שאפשר לקרוא להן בשתי דרכים:

magic(my\_object, args)

my\_object.\_\_magic\_\_( args)

פונקציות קסם ממשות את עקרון הרב צורתיות – מחלקות רבות ממשות של פונקצייית קסם.

דוגמה לכך היא הפונקציה str שהופכת כל עצם למחרוזת, בלי קשר לסוגו.

המחלקה Callable:

היא מחלקה המגדירה עצם קריא.

עצם קריא הוא עצם בעל פונקציית הקסם \_\_call\_\_ . הסינטקס:

My\_callable(args, kwargs)

מקביל לסינטקס:

My\_callable.\_\_call\_\_(args, kwargs)

הערה: פונקציות ועצמים קריאים הם מונחים מקבילים.

קשטן - :Decorator

קשטן היא פעולה שמקבלת פונקציה או מחלקה ומחיזרה פונקציה או מחלקה חדשה בהתאם.

הקשטן בדרך כלל מוסיף פונקציונליות חדשה לפונקציה או מחלקה בלי לשנות את הלוגיקה.

דוגמה: קשטן שמדפיס את הטיעונים בכל קריאה לפונקציה:

|  |
| --- |
| def printer(original\_function: Callable) -> Callable:  *"""A decorator that prints  the arguments of   the function it decorates  every time it is called."""* def new\_function(args):  print(args)  return original\_function(args)   return new\_function |

הסינטקסט לשימוש בקשטן:

|  |
| --- |
| @my\_decorator def original\_function():  # original function's code |

מקביל לסינטקס:

|  |
| --- |
| original\_function = my\_decorator(original\_function) |

מחלקת בסיס אבסטרקטית:

מחלקות אבסטרקטיות הינן מחלקות שלא ניתן לייצר מהן אובייקטים ומטרתן היחידה היא להוות מחלקת בסיס. במקרים בהם יש צורך להגדיר מחלקות בסיס לאובייקטים ממשיים ניתן להגדירם כמחלקות אבסטרקטיות.

לדוגמה: ניצור מחלקה אבסרקטית המייצגת צורה וממנה יורשות המחלקות ריבוע, משולש עיגול...

אנחנו רוצים ליצור עצמים ממחלקות המשנה (למשל ריבוע) ורוצים למנוע יצירת צורה כללית שאיננה אחת מהצורות הספציפיות (ריבוע, עיגול, משולש...).

על מנת ליצור מחלקה אבסטרקטית, אשתמש בקשטן abc.ABC.

פונקציה אבסרקטית:

היא פונקציית מחלקה (mathod) של מחלקת בסיס אבסטרקטית שממומשת בנפרד לכל אחת מהמחלקות היורשות.

לדוגמה – במחלקה צורה נרצה שלכל אחת מתת המחלקות תיהיה פונקציה המחשבת שטח ובעזרת נוחסה שונה לכל תת מחלקה.

על מנת ליצור פונקציה אבסטרקטית, אשתמש בקשטן abc.abstract\_method

פעולה סטטית:

פעולה סטטית היא פעולה ששייכת למחלקה עצמה ולא לעצם.

אחד השימושים הנפוצים של פעולות סטטיות הוא כפעולות עזר לפעולות לא סטטיות.

לדוגמה – במחלקה המייצגת רובוט שזורק חפצים, נרצה לממש פונקציית עזר שמקבלת מיקום של הרובוט, מסת החפץ הנזרק ומיקום רצוי של החפץ ומחשבת את זווית ומהירות הזריקה.

זוהי איננה פונקציה פנימית מכיוון שהיא לא תלויה בתכונות של הרובוט אך מקומה כן בתוך האובייקט מכיוון שהיא ממשת פעולה ש

על מנת ליצור פעולה סטטית, אשתמש בקשטן staticmethod.

מחלקת נתונים – Data Class:

הקשטן dataclasses.dataclass יוצר מחלקה ליצוג נתונים בעזרת שמות וסוגי המשתנים.

הקשטן יוצר פונקציות כגון: אתחול, יצוג כמחזורת ובדיקת שוויון.

למחלקת נתונים אפשר להוסיף פונקציות ואף לדרוס את הפונקציות שנוצרות על ידי הקשטן.

פיתוח מודלי למידה עמוקה בעזרת tensorflow:

הספרייה tensorflow מכילה כלים רבים ליצירה ואימון של מודלי למידה עמוקה באופן אמין ופשוט.

שניים מהכלים הם יצירת שכבות ומודלים בעזרת יצירת מחלקות היורשות מהמחלקה Layer ו Model בהתאם.

המחלקה Module:

נשתמש בה כשנרצה ליצור מחלקה היורשת מהמחלקה Callable שעובדים בצורה אופטימלית עם טנזורים. מחלקה היורשת באופן ישיר מ Module לא אמורה להשתמש במחלקות אחרות שיורשות באופן ישיר מ Module אחרים אלא רק בפונקציות הבנויות בספרייה.

היא מכילה אופטימיזציות רבות לפעולות על טנזורים.

הפונקציה call היא הפונקציה שנקראית באופן עקיף כשקוראים לאובייקט שמממש את המחלקה Module ובתוכה נכתוב את הלוגיקה של המחלקה.

המחלקה Layer:

יורשת מהמחלקה Module. נירש ממנה ישירות על מנת ליצור מחלקה המשתמשת במודולים ושכבות אחרות.

שכבות בנויות מראש:

בספרייה מומשות שכבות שימושיות כגון שכבה דחוסה, אמבדינג ורגורליזציה.

כל שכבה מומשה על ידי ירישה מהמחלקה Layer.

המחלקה Model:

הפונקציה compile מקבלת מודול והופכת אותו לפונקציה בשפת c++ ולאחר מכן מעבירה אותו קומפילציה.

הפונקציה fit של המחלקה מקבלת זוגות של קלט ופלט ומאמנת את המודל בגישת אימון מונחה ולכן לא אוכל להשתמש בה למידול שפה בגישת לימוד בהנחיה עצמית.

במידה ויש לנו מודול שמורכב מכמה שכבות\מודלים שונים, קריאה לפונקציה complie קוראת לפונקציה compile של תת השכבות והמודלים.

פיתוח התכונה

האלגוריתם ליצירת טקסט בעזרת קבוצות של טוקנים:

ביצירת טקסט בקבוצות, יצירת הטקסט נעשית כמו ביצירת טקסט "רגילה" רק שבמקום שהאלגוריתם יחשב את ההסתברות של הטוקן הבא – האו מחשב את ההסתברות של כמה מהטוקנים הבאים בהתבסס על הטוקנים שהוא כבר יצר.

**יש לקרוא את החלק התיאורטי לפני שקוראים את ההסבר על מימוש האלגוריתם.**

הקוד של האלגוריתם נמצא בקבצים: text\_generator.py, sampling\_generator.py, tree\_generator.py. הקבצים מופיעים כנספח בקובץ זה.

המחלקה :TextGenerator

היא מחלקת בסיס אבסטרקטית.

משתני המחלקה:

מודל, טוקנייזר, גודל הקבוצה, טוקני ריפוד, טמפרטורה.

המחלקה מייצגת אובייקט קריא (callable) שטוען מודל שפה סיבתי מ – [hugging face hub](https://huggingface.co/models?pipeline_tag=text-generation&sort=likes) ומשתמש בו ליצירת טקסט מתוך טקסט שנקלט מהמשתמש בעזרת שיטת הדגימה שפיתחתי.

הפונקציה האבסטרקטית \_\_call\_\_ מקבלת פרומפט ומספר טוקנים ומוסיפה לפרומפט את מספר הטוקנים הרצוי.

הפונקציה: create\_prob\_matמקבלת קלט למודל בצורת מחרוזת או רצף טוקנים.

ומחזירה רשימה דו מימדית (מטריצת ההסתברות) בה prob\_mat[a][b] היא ההסתברות של הטוקן שהמזהה שלו b להיות האיבר ה a בהמשך של הטקסט הנקלט.

זאת על ידי הוספה של (גודל הקבוצה פחות 1) טוקני ריפוד בסוף הרצף.

אינטואיציה:

LHM – look ahead mask – מסכת הסתכלות קדימה.

יצירת הטוקן הבא הבאה:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| next token: |  |  |  |  |  |
| Favorite | [LHM] | [LHM] | [LHM] | [LHM] | My |
| Food | [LHM] | [LHM] | [LHM] | favorite | My |
| Is | [LHM] | [LHM] | food | favorite | My |
| Ice | [LHM] | Is | food | favorite | My |
| Cream | ice | Is | food | favorite | My |

יצירת שני הטוקנים הבאים:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Next token |  |  |  |  |  |
| Food | [LHM] | [LHM] | [LHM] | favorite | My |
| Is | [LHM] | [LHM] | [PAD] | favorite | My |
| Ice | [LHM] | Is | food | favorite | My |
| Cream | [PAD] | Is | food | favorite | My |

דוגמה: אם הקלט הוא המילה "בוקר" ומזהה המילה "טוב" הוא 95 וההסתברות (לפי המודל) שהמילה "טוב" תופיע מיד אחרי המילה "בוקר" הוא 80% אז prob\_mat[0][95] יהיה 0.8 ואם מזהה המילה "לכולם" הוא 117 וההסתברות (לפי המודל) שהמילה "כולם" תופיע בצורה "בוקר" (מילה לא ידועה) "לכולם" היא 60% אז prob\_mat[1][117] יהיה 0.6.

המחלקה TreeGenerator:

יורשת מהמחלקה TextGenerator ומייצרת טקסט בעזרת חיפוש עץ בקבוצות:

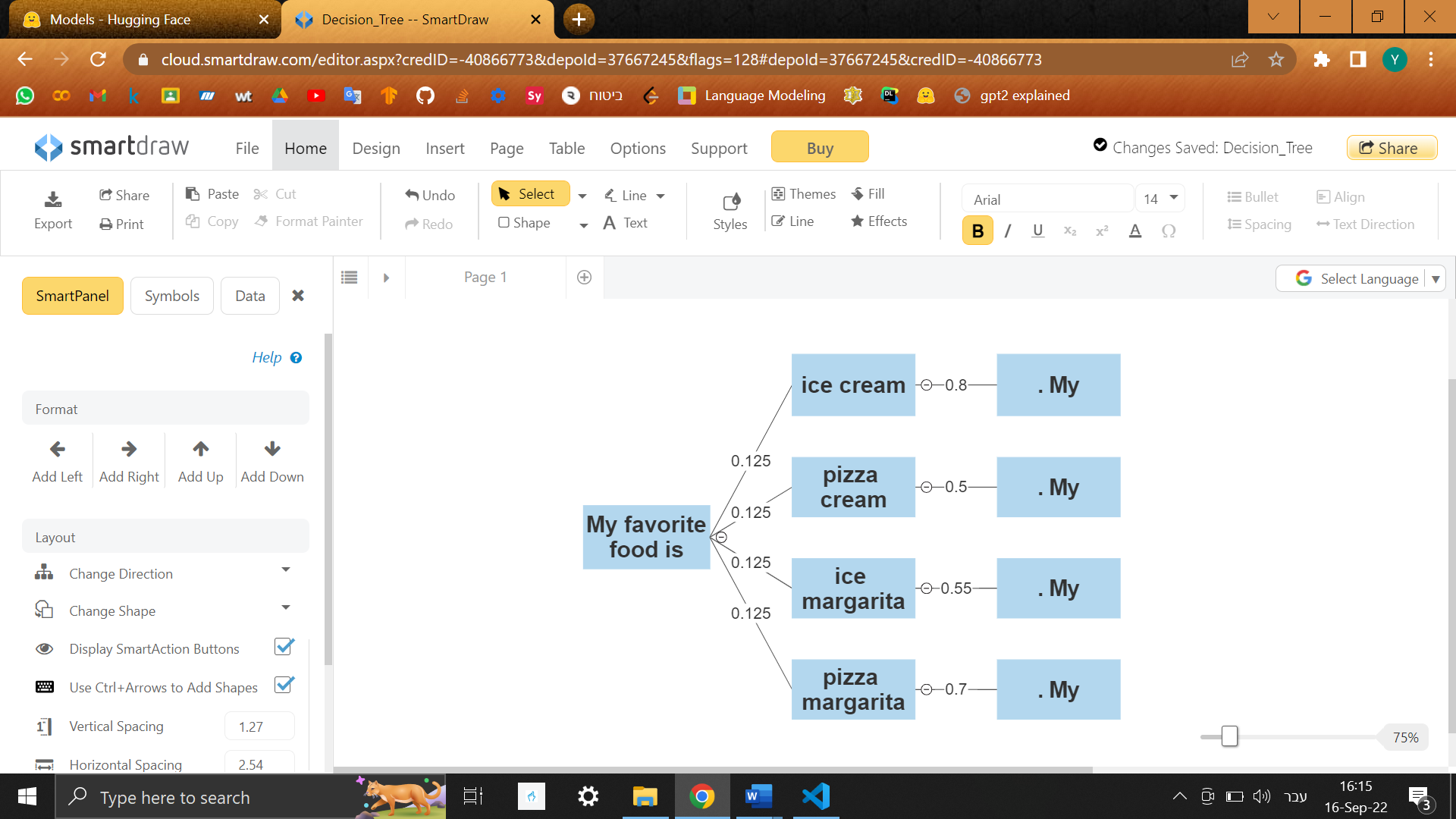
במקום לקחת את top\_k/top\_p הטוקנים שהסתברותם להיות הטוקן הבא הכי גבוהה בכל שלב - בוחרים לכל טוקן בקבוצה את top\_k/top\_p הטוקנים שהסתברותם הכי גבוהה ככה שאף טוקן לא מופיע יותר מפעם אחת – אם לטוקן מסוים ההסתברות גבוהה להיות במקום הראשון בקבוצה אז הסתברותו להיות במקומות אחרים בקבוצה נקבעת (באופן מלאכותי) לאפס.

ההסתברות של קבוצה מוגדרת במכפלת הסתברות הטוקנים בה. והסתברות של טקסט מוגדר כמפלת הסתברות הקבוצות.

דוגמה להמחשה – דגימה של 4 טוקנים כאשר גודל הקבוצה הוא 2 ו top\_pהוא 0.5 עבור הפורמפט "My favorite food is":

אם המזהה של הטוקנים pizza, ice, cream, margarita הם המספרים אפס עד שלוש בהתאם, מטריצת ההסתברות של הקבוצה הראשונה תיראה באופן הבא:

ובחירת הקבוצות תעשה באופן הבא:



הטקסט שיווצר הוא: "My favorite food is ice cream. My" מכיוון שהסתברותו הכי גבוהה.

המחלקה SampleGenerator:

יורשת מהמחלקה TextGenerator ומייצרת טקסט בעזרת דגימה בקבוצות. אסביר בעזרת דוגמה:

דגימה של 4 טוקנים כאשר גודל הקבוצה הוא 2 ו top\_pהוא 0.5 עבור הפורמפט

"My favorite food is":

אם המזהה של הטוקנים pizza, ice, cream, margarita הם המספרים אפס עד שלוש בהתאם, מטריצת ההסתברות של הקבוצה הראשונה תיראה באופן הבא:

נגריל את הטוקן הראשון לפי השורה הראשונה במטריצה ונקבל את הטוקן "pizza"

נקבע את ההסתברות של הטוקן "pizza" להיות אפס ונחלק כל איבר בווקטור ההסתברות בסכום הווקטור ונקבל שווקטור ההסתברות של הטוקן השני הוא:

*ואז נגריל את הטוקן "margarita".*

*לאחר מכן נתחיל את אותו התהליך עם הטקסט "My favorite food is pizza margarita" וככה הלאה...*

אפליקציית הווב:

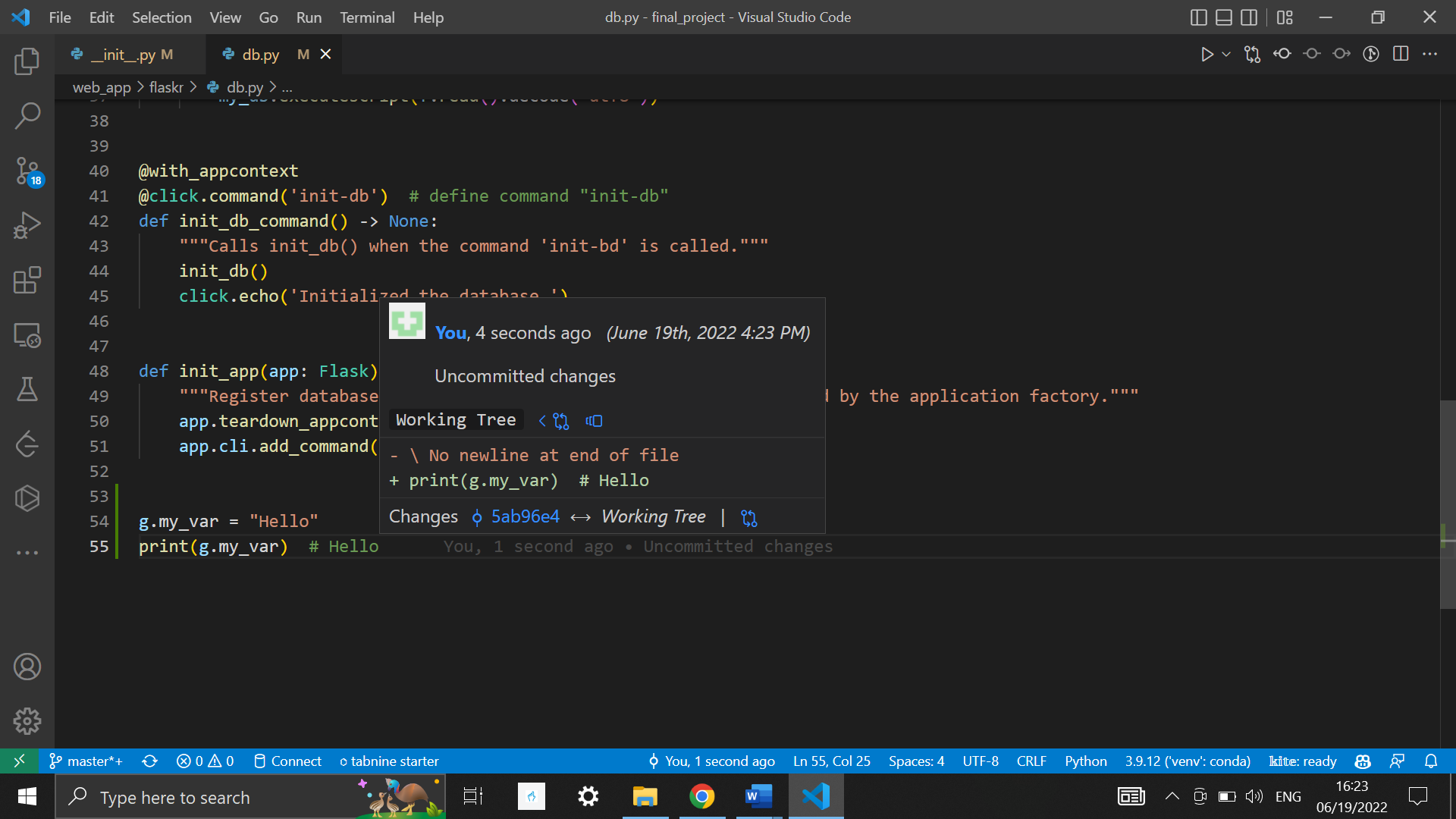
באפליקציית הווב המשתמשים יכולים להשתמש באלגוריתם לדגימה בקבוצות בעזרת מודלים מהאתר hugging face hub עם פרמטרים שונים במטרה להשוות בין שיטות דגימה, בין מודלים ובין פרמטרים של הדגימה (גודל הקבוצה, טמפרטורה, top\_p, top\_k...).

כל ההשלמות נשמרות בבסיס נתונים וכל משתמש יכול לראות את התוצאות של כל שאר המשתמשים.

הקובץ \_\_init\_\_.py:

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 מהסוג 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:



הקובץ database.py:

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 פקודות ה sql 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 מודלים עם העמודות:

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

user\_id – מספר סידורי של המשתמש הראשון שהשתמש במודל. (בין מודל למשתמש יש קשר רבים לרבים).

model\_name – שם הקובץ של המודל כפי שמופיע ב hugging face hub.

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

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 – הקלט לאלגוריתם.

answer - הפלט של האלגוריתם.

num\_tokens – מספר הטוקנים שיוצרו על ידי האלגוריתם.

Generation type – איזה אלגוריתם יצר את הטקסט? - (דגימה או עץ).

top\_p, top\_k, temperature – פרמטרים של האלגוריתם.

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 בהתאם.

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 עם הודעה לפיה עליו להירשם לפני שהוא משתמש באתר. 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 get\_model\_id מחזירה את המזהה של מודל מתוך תטבלת המודלים בבסיס הנתונים בהינתן השם שלו. אם המודל לא נמצא בבסיס הנתונים – הפונקציה מוסיפה אותו.

הקובץ complition.py מכיל את ה blueprint complition.

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 ופרמטרים ליצירת הטקסט. לאחר שהמשתמש לוחץ על הכפתור "Complete" המודל שהוא בחר משלים את הטקסט. ההשלמה נכנסת לבסיס הנתונים והמשתמש מועבר לעמוד הראשי.

השתמשתי ב Pytest בשביל לבדוק את כל 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.

מימוש הארכיטקטורה לטרנספורמר עם דיקודר בלבד:

ההסבר המלא על האלגוריתם של טרנספורמר עם דיקודר בלבד נמצא בסקירה על בינה מלאכותית.

בחלק זה אתמקד במימוש עצמו.

המחלקה Transformer:

כל המודלים שאצור על מנת לאמן הם עצמים מהמחלקה Transformer שיורשת מ Model (שיורשת גם מ tf.keras,layers.Layer) ויש לה 3 פונקציות (בנוסף לאלו שהיא יורשת):

אתחול "\_\_init\_\_": שתחילה קוראת לבנאי של המחלקה Model, יוצרת קידוד מיקומים המתאים להייפר הפרמטרים של המודל ומשמש להגדרת ה decoder ואז יוצרת את השכבות: אמבדינג, דיקודר ו-(הכפלה במטריצת אמבדינג משוחלפת ולאחריה פונקציית softmax).

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).

שאר המחלקות:

Decoder, DecoderBlock, PointWiseFeedForwardNetwork, MyMultiHeadAttention, ScaledDotProductAttention.

בנויות דומה למחלקה Tranformer: הן יורשות (באופן ישיר, בשונה מ \Tranformer) מ Layer ויש להן שתי פונקציות: אתחול וקריאה. פונקצית 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.

הפונקציה positional\_encoding:

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

הפונקציה create\_masks:

מקבלת את הרצף 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.

אימון מודלים:

בתחילת המחקר, ניסיתי לאמן בעצמי מודל שפה סיבתי (להתחיל ממשקלים רנדומליים).

ניסיון זה כשל בעקבות מגבלה על כוח המחשוב – היה ברשותי מעבד אחד בלבד וזכרון של כ- 12 GB ולכן יכולתי לאמן רק מודלים קטנים מאוד שאימונם לא הניב תוצאות משמעותיות.

המודלים שלי היו בעלי עשרות מליוני פרמטרים בעוד המודלים הגדולים ביותר בארכיטכטורה זו שאומנו עד היום (סוף 2022) הם בעלי

פונקציית המטרה שבחרתי הייתה categorical cross entropy וראיי כי ערכי פונקציית המטרה אינם מהווים שיפור לעומת חזייה של הסתברות שווה לכל טוקן ולכן החלטתי שהתוצאות שהגעתי אליהן אינן משמעותיות. בנוסף לכך, לא חל שיפור בערכי פונקציית המטרה במהלך האימון – זאת למרות נסיונות רבים לשינוי הייפר-פרמטרים.

המחברת create\_models.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.

הורדת ויבוא חבילות:

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

הגדרות:

המחברת משנה את הגרעין (seed) ל-0 על מנת לשמור על תוצאות קבועות ועל יכולת שחזור תוצאות. המחברת 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.

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.

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 אימון המודל.

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

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.

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.

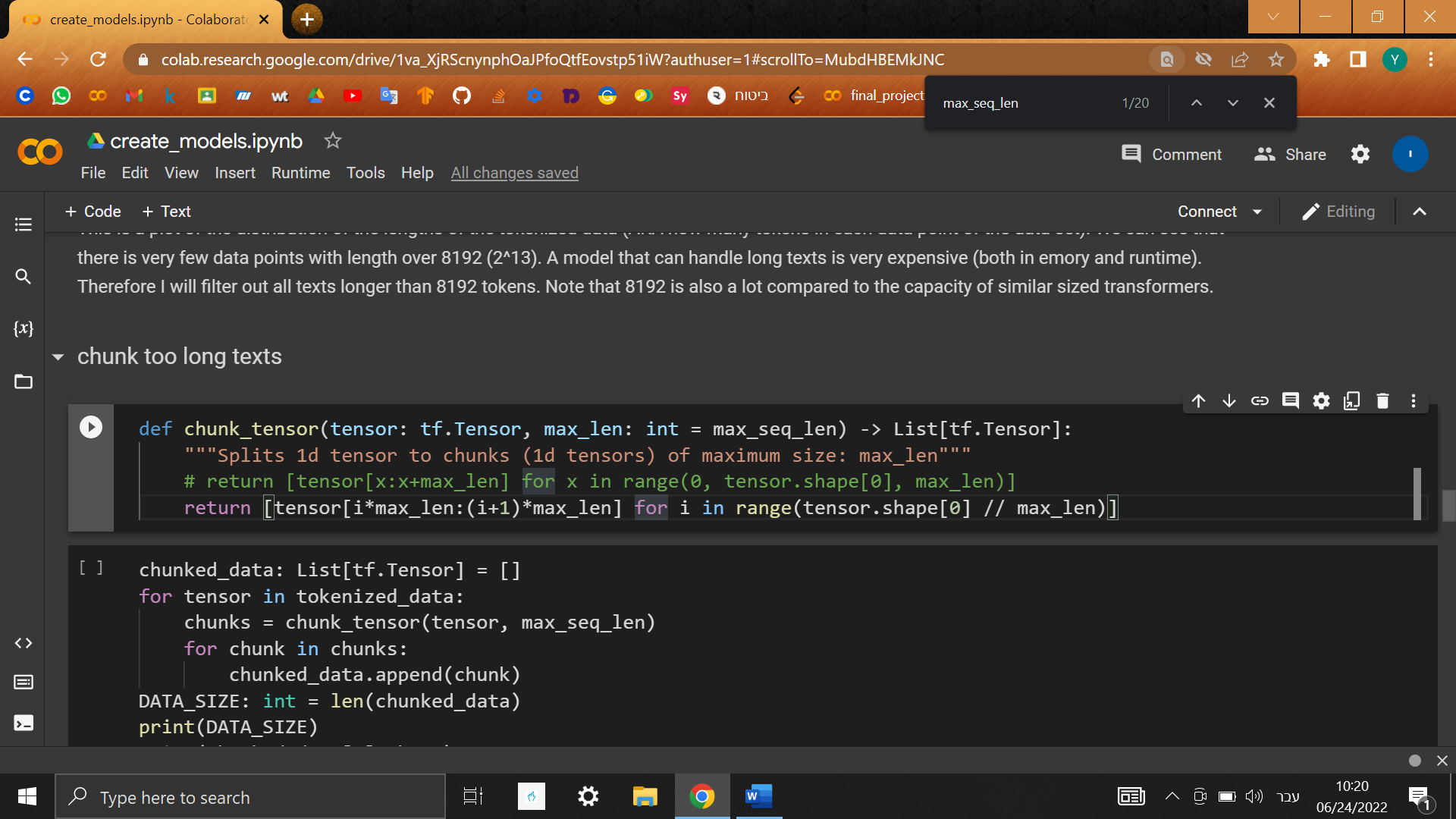
הפונקציה tokenize\_string מקבלת מחרוזת, קוראת לפונקציה tokenizer.Tokenize עליה, ממירה אותה מ- tf.RaggedTensor ל-tf.Tensor, משנה את טיפוס האיברים מ – int64 ל-int32 ומוסיפה טוקן התחלה וטוקן סוף.

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

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.

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.

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).

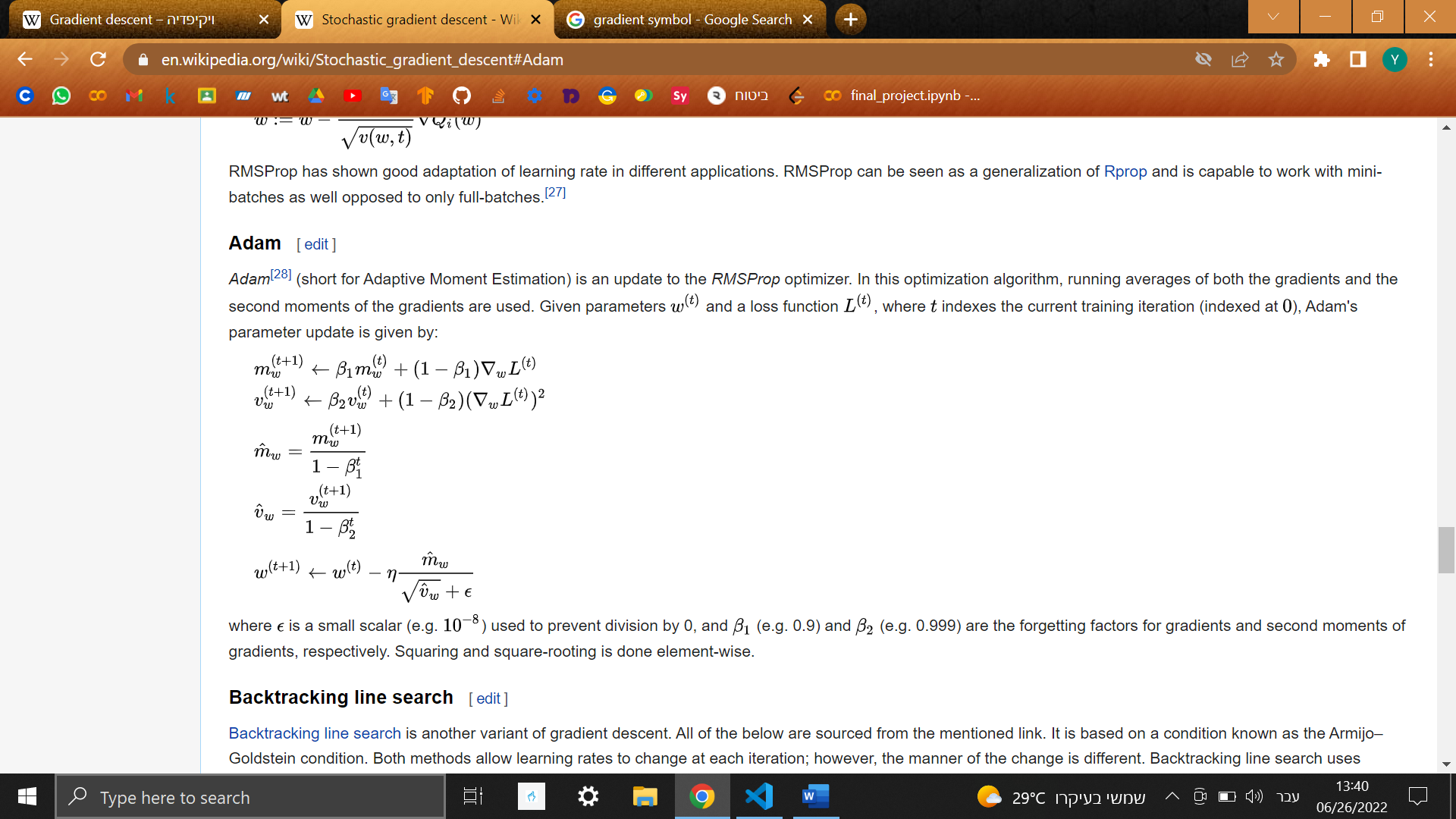
ניקוי הזכרון:

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.

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.



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.

הדמו:

הקובץ project\_demo.ipynb הוא מחברת ג'ופיטר אותה שאותה המשתמש פותח דרך colab .

במחברת המשתמש בוחר מודל מאומן מראש, ופרמטרים לדגימה (top k, top p, number of goups, group size) מכניס טקסט ורואה את הטקסט שהמודל חוזה בשיטת הדגימה שאני מציע בעבודה לעומת שיטות קודמות.

המודל המאומן מראש יכול להיות כל מודל שהועלו ל hugging face ויורש מהמחלקה [AutoModelForCausalLM](https://huggingface.co/docs/transformers/main/en/model_doc/auto#transformers.AutoModelForCausalLM). [בקישור הזה](https://huggingface.co/models?pipeline_tag=text-generation&sort=likes) אפשר למצוא רשימה מלאה של כל המודלים שמקיימים את הדרישות הללו.

הפרויקט הסופי:

סיכום:

ביבליוגרפיה:

נספחים:

חלקים נבחרים מהקוד:

מימוש האלגוריתם לדגימה בקבוצות:

המחלקה TextGenerator:

|  |
| --- |
| from abc import ABC, abstractmethod from collections.abc import Callable from typing import Optional, List  from torch import LongTensor, ones, no\_grad, Tensor from torch.nn import Softmax from transformers import (AutoTokenizer,  AutoModelForCausalLM,  AutoConfig,  PreTrainedTokenizer,  PreTrainedModel)   def get\_second\_item(sliceable):  return sliceable[1]   class TextGenerator(Callable, ABC):  *"""An abstract base class for  A callable object that given a prompt  and length of wanted answer,  generates text  the text generator has a model,  and some parameters  (Defined in the subclasses)"""* model\_name: str  tokenizer: PreTrainedTokenizer  model: PreTrainedModel  vocab\_size: int  temp: float  group\_size: int  padding\_tokens: List[int]  generation\_type: str   def \_\_init\_\_(self, model\_name: str, group\_size: int,  temp: float = 1.0):  *"""Model name: the name of the model  used for loading from hugging face hub  group size: int  the number of tokens to be predicted at each model call  temp: float  temperature parameter for the softmax function"""* self.model\_name = model\_name  self.tokenizer = AutoTokenizer.from\_pretrained(model\_name)  model = AutoModelForCausalLM.from\_pretrained(model\_name)  self.model = model.cuda()  config = AutoConfig.from\_pretrained(model\_name)  self.vocab\_size = config.vocab\_size  self.temp = temp  self.group\_size = group\_size  pad\_id = self.tokenizer.pad\_token\_id  self.padding\_tokens = [pad\_id] \* (self.group\_size - 1)   def get\_prob\_mat(self, prompt: Optional[str],  token\_list: Optional[List[int]])\  -> List[List[float]]:  *"""Returns the probability matrix  as a list of lists of floats"""* attention\_len = len(token\_list) + self.group\_size - 1  if token\_list is None:  tokenized\_prompt = self.tokenizer(prompt,  return\_tensors="pt")  if isinstance(tokenized\_prompt, list):  token\_list = tokenized\_prompt  elif isinstance(tokenized\_prompt, Tensor):  token\_list = tokenized\_prompt.tolist()  else:  token\_list = tokenized\_prompt["input\_ids"]   longer\_token\_list = token\_list + self.padding\_tokens  longer\_token\_tensor = LongTensor([longer\_token\_list])  attention\_mask = ones([1, attention\_len])  inputs = {"input\_ids": longer\_token\_tensor,  "attention\_mask": attention\_mask}  inputs = {name: tensor.cuda()  for name, tensor in inputs.items()}   if not isinstance(inputs, dict):  with no\_grad():  outputs = self.model(\*\*inputs)  elif "input\_ids" not in inputs.keys():  with no\_grad():  outputs = self.model(\*\*inputs)  else:  with no\_grad():  outputs = self.model(\*\*inputs,  labels=inputs["input\_ids"])   logits\_tensor = outputs.logits.squeeze(0) / self.temp   prob\_tensor = Softmax(dim=1)(logits\_tensor)  if self.group\_size <= prob\_tensor.shape[0]:  prob\_tensor = prob\_tensor[-self.group\_size:, :]  prob\_mat = [prob\_tensor[i, :].tolist()  for i in range(self.group\_size)]  else:  print("Warning: the group size is bigger than")  print("the length of the model's output")  print("If the length of the model input is n,")  print("n will be length of the model's output")  num\_tokens = self.group\_size - prob\_tensor.shape[0]  print(f"the predicted text will be \  {num\_tokens} tokens shorter")  prob\_mat = [prob\_tensor[i, :].tolist()  for i in range(prob\_tensor.shape[0])]   return prob\_mat   @abstractmethod  def \_\_call\_\_(self, prompt: str, num\_new\_tokens: int) -> str:  pass |

המחלקה TreeGenerator:

|  |
| --- |
| from random import choices, seed from copy import deepcopy from math import ceil from typing import Callable, List, Dict, Optional  from transformers import BatchEncoding  from text\_generator import TextGenerator, get\_second\_item   class SamplingGenerator(TextGenerator):  *"""A TextGenerator that generates text   using random sampling  with top-k or top-p filtering."""* top\_p: Optional[float] = None  top\_k: Optional[int] = None  filter\_tokens: Callable[[Dict[int, float]], Dict[int, float]]   generation\_type = "sampling"   def \_\_init\_\_(self, model\_name: str, group\_size: int,  temp: float = 1.0, top\_k: Optional[int] = None,  top\_p: Optional[float] = None):  super().\_\_init\_\_(model\_name, group\_size, temp)  seed(0)  if top\_p is None and top\_k is not None:  self.top\_k = top\_k  self.filter\_tokens = self.top\_k\_tokens  elif top\_k is None and top\_p is not None:  self.top\_p = top\_p  self.filter\_tokens = self.top\_p\_tokens  else:  raise ValueError("Either top\_k or top\_p \  should be set.")   def top\_p\_tokens(self,  sorted\_probs: Dict[int, float]) \  -> Dict[int, float]:  *"""Gets a token id: probability mapping  returns the tokens with the highest probability  such that their sum is <= top\_p.  or the token with the highest probability  if it's higher than top\_p."""* top\_p\_probs: Dict[int, float] = {}  prob\_sum: float = 0.0  for i, (curr\_token, curr\_prob) \  in enumerate(sorted\_probs.items()):  if i > 0 and prob\_sum + curr\_prob > self.top\_p:  break  prob\_sum += curr\_prob  top\_p\_probs[curr\_token] = curr\_prob  weighted\_probs = {k: v / prob\_sum  for k, v in top\_p\_probs.items()}  return weighted\_probs   def top\_k\_tokens(self, sorted\_probs: Dict[int, float])\  -> Dict[int, float]:  *"""Gets a token id: probability mapping  returns the top\_k tokens  with the highest probability."""* keys\_list = list(sorted\_probs.keys())  sorted\_top\_k\_keys = keys\_list[:self.top\_k]  top\_k\_probs: Dict[int, float]  top\_k\_probs = {k: sorted\_probs[k]  for k in sorted\_top\_k\_keys}  prob\_sum = sum(top\_k\_probs.values())  weighted\_probs = {k: v / prob\_sum  for k, v in top\_k\_probs.items()}  return weighted\_probs   def add\_group(self, prob\_mat: List[List[float]],  org\_used\_tokens: List[int]):  *"""Generates a group of tokens  using the choice\_function."""* used\_tokens = deepcopy(org\_used\_tokens)  answer = []  for curr\_token\_probs in prob\_mat:  for used\_token in used\_tokens:  curr\_token\_probs[used\_token] = 0.0   indexed\_prob: Dict[int, float]  indexed\_prob = {i: prob for i, prob  in enumerate(curr\_token\_probs)}  # O(vocab\_size)  items = indexed\_prob.items()  sorted\_probs: Dict[int, float]  # noinspection PyTypeChecker  sorted\_probs = dict(sorted(items,  key=get\_second\_item,  reverse=True))  weighted\_probs = self.filter\_tokens(sorted\_probs)  keys\_list = list(weighted\_probs.keys())  weights\_list = list(weighted\_probs.values())  sampled\_token = choices(keys\_list,  weights\_list,  k=1)[0]  answer.append(sampled\_token)  used\_tokens.append(sampled\_token)  return answer   def \_\_call\_\_(self, prompt: str, num\_new\_tokens: int) -> str:  num\_groups = ceil(num\_new\_tokens / self.group\_size)  tokenizer\_output = self.tokenizer(prompt,  return\_tensors="pt")  is\_dict = isinstance(tokenizer\_output, dict)  is\_batch\_encoding = isinstance(tokenizer\_output,  BatchEncoding)  if is\_dict or is\_batch\_encoding:  token\_tensor = tokenizer\_output["input\_ids"]  else:  token\_tensor = tokenizer\_output   curr\_token\_list = token\_tensor.squeeze().tolist()   for \_ in range(num\_groups):  prob\_mat = self.get\_prob\_mat(None,  curr\_token\_list)  new\_tokens = self.add\_group(prob\_mat,  curr\_token\_list)  curr\_token\_list.extend(new\_tokens)   prompt\_len = tokenizer\_output.shape[1]  final\_num\_tokens = prompt\_len + num\_new\_tokens  shorten\_token\_list = curr\_token\_list[:final\_num\_tokens]  final\_ans = self.tokenizer.decode(shorten\_token\_list)  return final\_ans |

המחלקה TreeGenerator:

|  |
| --- |
| from math import ceil from typing import List, Dict, Union, Tuple, Sequence, Any  from transformers import BatchEncoding  from text\_generator import TextGenerator, get\_second\_item   class TreeGenerator(TextGenerator):  *"""A TextGenerator that generates text  in a tree like fashion,  without random sampling."""* top\_p: float  top\_k: int  generation\_type = "tree"   def \_\_init\_\_(self, model\_name: str, group\_size: int,  top\_k: int, top\_p: float,  temp: float = 1.0):  super().\_\_init\_\_(model\_name, group\_size, temp)  self.top\_k = top\_k  self.top\_p = top\_p   @staticmethod  def no\_duplicates(my\_sequence: List[Any]) -> bool:  *"""Return if there isn't a repetition in the sequence  complexity: O(n) where n is the length of the sequence"""* return len(my\_sequence) == len(set(my\_sequence))   @staticmethod  def combinations(mat: Sequence[Sequence[Any]]) \  -> List[List[Any]]:  *"""Returns all the lists such that list[j] is in mat[j]  runtime function:  prod([len(mat[i]) for i in range(len(mat))])"""* if len(mat) == 1:  return [[mat[0][i]]  for i in range(len(mat[0]))]  res: List[List[Any]] = []  for i in mat[0]:  for j in TreeGenerator.combinations(mat[1:]):  res.append([i] + j)  filtered\_res: List[List[Any]]  filtered\_res = list(filter(  TreeGenerator.no\_duplicates,  res))  return filtered\_res   @staticmethod  def seq\_prob(tokens, prob\_mat,  org\_prompt\_prob) -> float:  *"""Given the probability matrix  and a list of tokens,  returns the probability  of the sequence.  prob\_mat[a][b] is the probability of  the token with id b the a-th  token in the sequence"""* probability = org\_prompt\_prob  sequence\_length = len(tokens)  for i in range(sequence\_length):  curr\_token = tokens[i]  probability \*= prob\_mat[i][curr\_token]  return probability   @staticmethod  def flatten(my\_list: Union[list, tuple]) -> List:  *"""Gets a list where some elements might be lists  and adds every item  in the inner list to the outer list.  example:  [1, [2, 3], 4, [[5]]] -> [1, 2, 3, 4, 5]  Complexity:  n - len(the flatten list)  m - the number of different lists  O(m + n)"""* new\_list = []  for item in my\_list:  if isinstance(item, list):  new\_list.extend(TreeGenerator.flatten(item))  else:  new\_list.append(item)  return new\_list   @staticmethod  def remove\_duplicates(completions: List[List[int]],  probs: List[float])\  -> Dict[Tuple[int], float]:  *"""Given a list of tokenized answers  and the probability of each completion,  removes every repeated completion  and every completion that have repeated tokens"""* filtered\_completions: Dict[Tuple[int], float]  filtered\_completions = dict()  for curr\_comp, curr\_prob in zip(completions, probs):  if len(curr\_comp) == len(set(curr\_comp)):  curr\_comp\_tuple = tuple(curr\_comp)  filtered\_completions[curr\_comp\_tuple] = curr\_prob  return filtered\_completions   def tree\_grouped\_sampling(self, prob\_mat: List[List[float]])\  -> List[List[int]]:  *"""given a matrix of probabilities,  returns a list of lists of tokens.  the matrix is of size group\_size x vocab\_size  where matrix[i, j] is  the probability of token j the i-th token in the group  samples the tokens such that  for each place in the group,  at most top\_k tokens are sampled  and at least one token is sampled  and the added probability of all the tokens is  less than or equal top\_p  returns a list of where every item is  a tuple of a sequence and probability  over all complexity of the function is  O(group\_size \* vocab\_size \* log(vocab\_size))"""* # prob\_tensor.shape is now  # (group\_size, vocab\_size)  possible\_tokens = []  already\_predicted = set()  for token\_prob in prob\_mat: # group\_size times  vocab\_size = len(token\_prob) # O(1)  indexed\_prob = list(zip(token\_prob,  range(vocab\_size)))  # O(vocab\_size)  sorted\_indexed\_prob = sorted(indexed\_prob,  key=get\_second\_item,  reverse=True)  # O(vocab\_size\*log(vocab\_size))  curr\_k = 0  total\_prob = 0  curr\_indices = []  for prob, token in sorted\_indexed\_prob:  # O(top\_k)  top\_p\_break = total\_prob + prob > self.top\_p  top\_k\_break = curr\_k == self.top\_k  if top\_p\_break or top\_k\_break:  break  if token not in already\_predicted:  already\_predicted.add(token)  curr\_k += 1  total\_prob += prob  curr\_indices.append(token)  possible\_tokens.append(curr\_indices) # O(1)  new\_sequences: List[List[int]]  new\_sequences = TreeGenerator.combinations(possible\_tokens)  # theta(prod(len(indices[i])  # for i in range(group\_size)))  # len(indices[i]) < min(top\_k, vocab\_size)  # therefore the complexity is  # O(min(top\_k, vocab\_size) \* group\_size)  return new\_sequences   def rec\_gen(self, org\_prompt, num\_tokens,  org\_prompt\_prob: float = 1.0) \  -> Dict[Tuple[int], float]:  *"""Recursively generates the next group of tokens  in a tree like behavior"""* num\_groups = ceil(num\_tokens / self.group\_size)  is\_list = isinstance(org\_prompt, list)  is\_tuple = isinstance(org\_prompt, tuple)  if is\_list or is\_tuple:  tokens\_list = TreeGenerator.flatten(org\_prompt)  str\_prompt = self.tokenizer.decode(tokens\_list)  else:  print(org\_prompt)  raise ValueError("org\_prompt must be a string \  or list of ints")   prob\_mat = self.get\_prob\_mat(str\_prompt, None)  tokenized\_ans\_list = self.tree\_grouped\_sampling(prob\_mat)  prob\_list: List[float]  prob\_list = [TreeGenerator.seq\_prob(seq,  prob\_mat,  org\_prompt\_prob)  for seq in tokenized\_ans\_list]  new\_prompts: List[List[int]]  new\_prompts = [TreeGenerator.flatten(tokens\_list + ans)  for ans in tokenized\_ans\_list]   completion\_probs: Dict[Tuple[int], float]  completion\_probs = TreeGenerator.remove\_duplicates(  new\_prompts, prob\_list)  items = completion\_probs.items()   if num\_groups == 1:  shorten\_completions = {k[:num\_tokens]: v  for k, v in items}  return shorten\_completions   new\_completions: Dict[Tuple[int], float] = dict()  for curr\_new\_prompt, curr\_new\_prompt\_prob in items:  curr\_completions: Dict[Tuple[int], float]  new\_number\_tokens = num\_tokens - self.group\_size  curr\_completions = self.rec\_gen(curr\_new\_prompt,  new\_number\_tokens,  curr\_new\_prompt\_prob)  tokens: Tuple[int]  prob: float  for tokens, prob in curr\_completions.items():  new\_completions[tokens] = prob  return new\_completions   def \_\_call\_\_(self, prompt: str, num\_new\_tokens: int) -> str:  *"""given a prompt and number of tokens to generate,  returns a string of the prompt + the generated tokens"""* tokenizer\_output = self.tokenizer(prompt,  return\_tensors="pt")  is\_dict = isinstance(tokenizer\_output, dict)  is\_batch\_encoding = isinstance(tokenizer\_output,  BatchEncoding)  if is\_dict or is\_batch\_encoding:  token\_tensor = tokenizer\_output["input\_ids"]  else:  token\_tensor = tokenizer\_output   final\_num\_tokens = token\_tensor.shape[1] + num\_new\_tokens   tokenized\_prompt: List[int]  tokenized\_prompt = token\_tensor.tolist()   seq\_prob\_dict: Dict[Tuple[int], float]  seq\_prob\_dict = self.rec\_gen(tokenized\_prompt,  num\_new\_tokens)   highest\_prob\_seq: Tuple[int]  highest\_prob\_seq = max(seq\_prob\_dict,  key=seq\_prob\_dict.get)  final\_token\_list: List[int]  final\_token\_list =list(highest\_prob\_seq[:final\_num\_tokens])  decoded\_prompt = self.tokenizer.decode(final\_token\_list)  return decoded\_prompt |

מימוש הארכיטקטורה של טרנספורמר עם דיקודר בלבד:

קישורים:

[העמוד של הפרויקט בגיטהאב](https://github.com/yonikremer/final_project)

[קולאב](https://colab.research.google.com/) – (Google collaboratoty)

דוקומנטציות:

[פייתון](https://docs.python.org/3/)

[פייתורץ'](https://pytorch.org/docs/stable/index.html)

[טנזורפלואוו](https://www.tensorflow.org/api_docs/python/tf/all_symbols)

[hugging face](https://huggingface.co/docs/transformers/main/en/model_doc/auto)

קורסים של אוניברסיטת סטנפורד:

[עיבוד שפה טבעית בעזרת למידה עמוקה](https://web.stanford.edu/class/cs224n/)

[הבנת שפה טבעית](https://web.stanford.edu/class/cs224u/)

מדריכים:

[NLP guide by PhD Lena Voita](https://lena-voita.github.io/nlp_course.html)

[data augmentation methods for text-blog](https://neptune.ai/blog/data-augmentation-nlp)

[Evaluating text generation blog post](https://towardsdatascience.com/how-to-evaluate-text-generation-models-metrics-for-automatic-evaluation-of-nlp-models-e1c251b04ec1)