דגימה בקבוצות – שימוש יעיל במודלי שפה סיבתיים

(Causal Language Models)

עבדות גמר מדעית בהיקף 5 יחידות לימודבמדעי המחשב

סמל שאלון: 899589

מגיש: יוני קרמר

תעודת זהות: 215005737

בית ספר: עירוני ד' על שם אהרון קציר, תל אביב

שם המנחה: עידו גודיס

רכזת עבודות גמר: לימור שיאון

הוגש בינואר 2023

משתתפת בתחרות מדענים ומפתחים צעירים בישראל 2023



תוכן עניינים

[תוכן עניינים 2](#_Toc124426009)

[הקדמה אישית 4](#_Toc124426010)

[תודות 4](#_Toc124426011)

[מבוא 5](#_Toc124426012)

[רקע: 5](#_Toc124426013)

[שאלת המחקר: 5](#_Toc124426014)

[השערת מחקר: 5](#_Toc124426015)

[מטרת המחקר: 6](#_Toc124426016)

[תיאור העבודה: 6](#_Toc124426017)

[סקירת בינה מלאכותית 8](#_Toc124426018)

[למידה מונחית – Supervised Learning: 8](#_Toc124426019)

[למידה בהנחיה עצמית – Self-Supervised Learning: 8](#_Toc124426020)

[ווקטור הסתברות – Probability Vector: 8](#_Toc124426021)

[ווקטור לוג'יט – Logit Vector: 8](#_Toc124426022)

[טוקניזציה – Tokenization: 8](#_Toc124426023)

[Dummy/One Hot Encoding: 9](#_Toc124426024)

[Sparse Cross Entropy 10](#_Toc124426025)

[שיכון טוקנים – Token Embedding: 10](#_Toc124426026)

[הפונקציה SoftMax: 10](#_Toc124426027)

[טמפרטורה - Temperature: 11](#_Toc124426028)

[מודל שפה סיבתי – Causal Language Model: 12](#_Toc124426029)

[טרנספורמר עם דיקודר בלבד – Decoder Only Transformer: 12](#_Toc124426030)

[**אימון ראשוני למידול שפה סיבתי בעזרת יצירת שפה:** 20](#_Toc124426031)

[**Generative Pre-Training (GPT) for Causal Language Modeling** 20](#_Toc124426032)

[דגימה – Sampling/Decoding: 21](#_Toc124426033)

[הבעיות עם שיטות הדגימה הקיימות: 26](#_Toc124426034)

[מדד ברט – BERT Score: 26](#_Toc124426035)

[תצפיות על התנהגותם של מודלי שפה סיבתיים: 27](#_Toc124426036)

[סקירת פייטון 28](#_Toc124426037)

[רשימת מבני נתונים בפייטון ומבני הנתונים המקבילים בשפות אחרות: 28](#_Toc124426038)

[רמזי סוג – Type Hints: 28](#_Toc124426039)

[רב צורתיות (פולימורפיזם): 28](#_Toc124426040)

[פעולות קסם – Magic/Dunder Methods: 29](#_Toc124426041)

[קשטנים - :Decorators 29](#_Toc124426042)

[מחלקת בסיס אבסטרקטית: 30](#_Toc124426043)

[פונקציה אבסטרקטית: 30](#_Toc124426044)

[פעולה סטטית: 30](#_Toc124426045)

[מחלקת נתונים – Data Class: 31](#_Toc124426046)

[המחלקה האבסטרקטית Callable: 31](#_Toc124426047)

[Enum: 31](#_Toc124426048)

[ערמת מקסימום\מינימום – Maximum/Minimum Heap: 31](#_Toc124426049)

[פיתוח מודלי למידה עמוקה בעזרת TensorFlow: 32](#_Toc124426050)

[הסבר האלגוריתם 34](#_Toc124426051)

[פיתוח התכונה 36](#_Toc124426052)

[אפליקציית הווב: 42](#_Toc124426053)

[מימוש הארכיטקטורה לטרנספורמר עם דיקודר בלבד: 46](#_Toc124426054)

[הדמו: 47](#_Toc124426055)

[ניסויים: 48](#_Toc124426056)

[הצגת התוצרים 68](#_Toc124426057)

[אפליקציית הווב: 68](#_Toc124426058)

[סיכום 71](#_Toc124426059)

[המשך המחקר: 71](#_Toc124426060)

[ביבליוגרפיה 72](#_Toc124426061)

[נספחים 76](#_Toc124426062)

[קישור לעמוד הגיטהאב של הפרויקט 76](#_Toc124426063)

[קוד האלגוריתם לדגימה בקבוצות: 76](#_Toc124426064)

[דיאגרמת בסיס הנתונים של אפליקציית הווב: 95](#_Toc124426065)

הקדמה אישית

את ההיכרות שלי עם למידת מכונה התחלתי כמו רבים בקורסים באינטרנט במהלך הקורונה.

התחום הזה ריתק אותי מהרגע שידעתי שהוא קיים. ברגע שהשתמשתי במודלים כמו gpt3 ידעתי שלבינה מלאכותית יש את היכולת לבצע המון משימות אנושיות – ואם אין בינה מלאכותית שמסוגלת לבצע את המשימות האלה היום, תהיה בינה מלאכותית שמסוגלת לבצע זאת בעתיד.

אני מאמין שבינה מלאכותית תוכל לבצע גם את המשימות שנחשבות הכי אנושיות בעתיד הנראה לעין. אני מאמין שבימי חיי אני אשתמש באותה תדירות שבה אני משתמש היום במחשבים מכל הסוגים (מחשב אישי, טלפון, טלוויזיה, מזגן...).

בגלל שאני משתמש קבוע במודלי שפה סיבתיים כגון OPT , chat-GPT ו codex, בחרתי לחקור אותם בעבודה זו.

תודות

אני רוצה להודות לעידו גודיס על הנחייה אקדמית, תמיכה מקיפה, זמינות תמידית ועזרה רבה בבית ספר.

תודה ללימור שיאון רכזת עבודות הגמר בביה"ס, על ליווי מעמיק ומסור וניהול מקצועי. בזכותך מספר כותבי עבודות הגמר בעירוני ד' גדול כל כך. אני רוצה להודות לך על כך ששכנעת אותי להתחיל את עבודת הגמר.

תודה לצוות תחרות מדענים צעירים בכלל ולחמוטל לוטן בפרט על חוויה של פעם בחיים, על ההזדמנות להציג את העבודה שלי למומחים בתחום המדעי הנתונים ולציבור הרחב ועל ההזדמנות לפגוש חוקרים צעירים כמוני ולשמוע על עבודות מרתקות במגוון תחומים.

תודה לליטל שיריין וצביאל למברגר על הליווי מטעם תחרות מדענים צעירים שכלל עזרה בכתיבת תקציר העבודה, עזרה בהכנת הפוסטר והסרטון.

מבוא

רקע:

מאז פרסום המודל GPT (שנקרא מאוחר יותר GPT1) בשנת 2018 ונכון להגשת העבודה (ינואר 2023), מודלי שפה סיבתיים היו מודלי הלמידה העמוקה המתקדמים ביותר למשימות בינה מלאכותית בהן גם הקלט וגם הפלט הם טקסטים. מודלי שפה סיבתיים מצליחים במשימות רבות שמודלים בטכנולוגיות קודמות לא הצליחו בהן כמו סיכום, כתיבת קוד, כתיבה יצירתית, ועוד.

כיום חברות רבות משתמשות במודלי שפה סיבתיים למשימות רבות:

Meta, google and Microsoft משתמשים בהם לתרגום,

Grammarly משתמשים בהם לתיקון שגיאות תחביריות,

AI21 Labs משתמשת בהם כדי לשפר ניסוח של טקסטים,

GitHub and Tabnine משתמשות בהם כדי לכתוב קוד.

החיסרון המרכזי של מודלים אלו הוא עלות המחשוב של השימוש בהם.

זאת הבעיה שפתרתי בעבודה זו.

שאלת המחקר:

האם ניתן ליצור טקסט באורך n טוקנים בעזרת פחות מ n שימושים מודל שפה סיבתי (causal language model)? אם כן, כיצד? ואיך שימוש באלגוריתם זה משפיע על הטקסט שנוצר (בהסתכלות על מדדי BERT Score בהשוואה לאלגוריתמים היוצרים n טוקנים על ידי n קריאות למודל)?.

השערת מחקר:

אני משער שניתן ליצור טקסט באורך n טוקנים בעזרת פחות מ n שימושים מודל שפה סיבתי. אני חושב שטקסטים אלה יהיו יותר קרובים לטקסטים אנושיים. זאת מכיוון שכשבני אדם כותבים טקסט או מדברים הם לא חושבים רק על מה תהיה המילה הבאה, הם חושבים מה הם רוצים להגיד באופן כללי. רק אז הם מנסחים כל פעם חלק מהטקסט כשכל חלק מורכב מכמה מילים.

מטרת המחקר:

לפתח אלגוריתם היוצר n טוקנים בעזרת כמות מינימלית של שימושים במודל שפה סיבתי.

האלגוריתם צריך לעבוד עם כמה שיותר מודלי שפה סיבתיים שונים ולהיות יעיל באופן משמעותי מכל אלגוריתם קיים.

קהל היעד של הפיתוח הם מדעני נתונים, חוקרי ומפתחי בינה מלאכותית היוצרים טקסטים באמצעות מודלי שפה סיבתיים.

תיאור העבודה:

בעובדה פיתחתי אלגוריתם הנקרא דגימה בקבוצות היוצר n מילים ב שימושים במודל שפה סיבתי, לאחר מכן הראיתי שהאלגוריתם שפיתחתי יותר יעיל ושהוא מצליח במשימת תרגום יותר מאלגוריתמים קיימים במשימת תרגום.

גיליתי שעל מנת לקבל זמן ריצה מינימלי ואיכות טקסט מקסימלית, יש לקבוע את גדול הקבוצה לאורך הטקסט הטקסט שהאלגוריתם צריך ליצור אם גודל זה ידוע, ולחסם מלמעלה של אורך הטקסט אחרת. במקרה זה, ניצור טקסט בעזרת שימוש אחד בלבד במודל שפה סיבתי.

חלקי העבודה:

* סקירת בינה מלאכותית – סקירה של נושאים בבינה מלאכותית שהבנתם קריטית על מנת להבין את הפתרון שפיתחתי.
* סקירת פייטון – סקירה של נושאים כללים במדעי המחשב וספציפיים לשפת פייטון שהשתמשתי בהם בפיתוח התוכנה. הבנת נושאים אלו עוזרת להבין את התכונה שפיתחתי ואת השיקולים שלי בפיתוח התוכנה.
* הסבר האלגוריתם – הסבר הרעיון שמאחורי האלגוריתם שפיתחתי.
* פיתוח התוכנה – תיאור מפורט של החלקים המרכזיים בתוכנה שפיתחתי, הכולל הסברים על ההחלטות שלקחתי במהלך הפיתוח.
* הצגת התוצרים – הצגת אפליקציית הווב שפיתחתי והצגת ניסויים שבהם בדקתי את הצלחת האלגוריתם שלי במשימת תרגום טקסטים.

בעמוד הגיטהאב (<https://github.com/yonikremer/grouped_sampling>) של הפרויקט ניתן למצוא חומרים נוספים על הפרויקט

אני מציע לקרואים להתחיל את קריאת העבודה בסקירת בינה מלאכותית על מנת להבין את כל התיאוריה מאחורי הפתרון, משם לעבור להסבר על האלגוריתם, לחלק של פיתוח התוכנה ולחלק של הניסויים והצגת התוצרים.

במקרה שאתם לא מכירים\לא מבינים מושג מסוים לבדוק אם הוא מופיע בסקירת פייטון.

רק לאחר שקראתם את החלק על פיתוח התוכנה כדאי לעבור להצגת התוצרים וזאת על מנת שתבינו את הסיבה לתוצאות.

סקירת בינה מלאכותית

למידה מונחית – Supervised 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, מודל שמוצא את החלק החסר בתמונה.

ווקטור הסתברות – Probability Vector:

במדעי הנתונים –הוא ווקטור בו מייצג את ההסתברות של מחלקה i. כל ההסתברויות בין אפס לאחד וסכום הווקטור 1 וגודל הווקטור הוא מספר המחלקות.

ווקטור לוג'יט – Logit Vector:

בלמידה עמוקה – ווקטור לוג'יט הוא ווקטור בו מוצגות הסתברויות בטווח מינוס אינסוף עד אינסוף. כאשר כלל שהלוג'יט של מחלקה יותר גבוהה, גם ההסתברות שלה יותר גבוהה. פונקציית SoftMax משמשת (בין היתר) להפוך ווקטור לוג'יט לווקטור הסתברות.

טוקניזציה – Tokenization:

בניגוד לתוכנות קלאסיות (שלא משתמשות בלמידת 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 וקובעים את גודל המילון **vocab size**, הוא מוצא את רצפי האותיות הכי נפוצים בסט האימון ונותן לכל אחד מהם מזהה (טוקן) בצורת מספר שלם ואי שלילי.

vocab size הוא מספר הטוקנים השונים בטוקנייזר.

לטוקנייזר שתי פונקציות מרכזיות:

Encode: הטוקנייזר מקבל מחרוזת ומחזיר רצף של מזהים של טוקנים לפי הסדר בהם הם מופיעים בטקסט כרשימה או כטנזור מסוג int.

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

כל מודל שקולט ו\או מייצר שפה מאומן בהינתן טוקנייזר – הטוקנייזר מוגדר לפני תחילת אימון המודל ולא משתנה אף פעם. שימוש במודל שפה בעזרת טוקנייזר לא מתאים או שינוי של הטוקנייזר גורם לתוצאות חסרות משמעות.

Dummy/One Hot Encoding:

[22](Suits)

ייצוג one hot ממיר אינדקס של מחלקה (במקרה שלנו טוקן) שהוא מספר שלם ואי שלילי לווקטור\מערך\רשימה בינארית שאורכה כמספר המחלקות בנתונים (במקרה שלנו מספר הטוקנים שהטוקנייזר שומר במילון).

ייצוג one hot הוא בעצם ווקטור של הסתברות של תוצאה ידועה מראש ולכן אנחנו רוצים שההסתברות שהמודל יחזה תהיה כמה שיותר קרובה לייצוג 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]]:  """Creates 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:

Sparse Cross Entropy

פונקציה המודדת את המרחק בין 2 ווקטורי הסתברות: .

אם הוא ווקטור one hot המיצג את המחלקה עם אינדקס , ניתן להשתמש בפונקציה השקולה:

שהסיבוכיות שלה היא ולא בהגדרת Cross Entropy.

סודו קוד:

float Sparse Categorical Cross Entropy(p: Tensor[n], q: OneHotVector[n])

c: int = q.getindex()

return -math.log(p[c])

שיכון טוקנים – Token Embedding:

[16](Mikolov et al.)

שיכון היא שכבה המקבלת רצף טוקנים בייצוג one hot וממירה כל ייצוג 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 מייצגת טוקן שהמודל מכיר וכל עמודה היא תכונה סמנטית שיכולה להיות לטוקן.

פעולת השיכון היא מכפלה של מטריצת הרצפים בייצוג one hotבמטריצת השיכון.

הפונקציה SoftMax:

[4]*(*Bridle*)*

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 זה.

לדוגמה: פעולת SoftMax בציר 1 היא פעולת SoftMax לכל שורה במטריצה.

טמפרטורה - Temperature:

[2]*(*Ackley et al*.* 5*)*

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:

אינטואיציה:

הקטנת הטמפרטורה גורמת לאיזון ההסתברויות של הטוקנים ככה שלכל הטוקנים תהיה 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 הטמפרטורות: 2, 1, 0.5:

מודל שפה סיבתי – Causal Language Model:

[14](Liu et al.)

מודל למידה עמוקה המקבל רצף של טוקנים  ומחזירה ומטריצת לוג'יט בה הוא ווקטור הלוג'יט שמייצג את החיזוי של המודל לטוקן במקום i+1 כאשר תלוי בטוקנים בלבד. הוא ווקטור הלוג'יט של הטוקן שבא לאחר סוף הרצף שהמודל מקבל – וככה אפשר להשתמש במודל שפה סיבתי על מנת לחזות את הטוקן הבא במשפט.

היתרון של מודלי שפה סיבתיים הוא שבהינתן רצף טוקנים שאיננו מחולק לקלט ופלט ניתן לאמן את המודל לחזות את הטוקן הבא בטקסט על ידי שימוש אחד בלבד במודל:

המודל מקבל מטריצה המייצגת רצף של טוקנים בצורת one-hot.

נוציא מהקלט את הטוקן במקום 0 כי המודל חוזה את הטוקנים החל מהמקום ה 1 ונוסיף טוקן מיוחד לסוף הקלט שמעיד על סוף הרצף. עכשיו קיבלנו מטריצה חדשה בה:

נשים לב ש מייצג את הטוקן שנמצא במקום i במטריצה M ולכן נאמן את המודל לצמצם את המרחק בין M ו P שנמדד באמצעות spare cross entropy.

טרנספורמר עם דיקודר בלבד – Decoder Only Transformer:

[14](Liu et al.)

ארכיטקטורה למימוש מודלי שפה סיבתיים בה המודל מקבל רצף של מספרים שלמים (מזהיי טוקן), הופך אותם לשיכונים ומוסיף להם שיכון מיקומי ואת התוצאה מעביר לדיקודר.

הדיקודר מורכב מכמה בלוקים הנקראים בלוקי דיקודר כאשר הבלוק הראשון מקבל את הפלט לדיקודר ושאר הבלוקים מקבלים את סכום הקלט והפלט של הבלוק שלפניהם.

הפלט של בלוק הדיקודר האחרון הוא הפלט של הדיקודר והוא מוכפל במטריצת שיכון 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 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 לא כולל t). ומטריצת לוג'יט לתאר מטריצה הבנויה מווקטורי לוג'יט באותה הצורה.

בלוק דיקודר Decoder Block:

[24](Vaswani et al., pt.3.1)

בלוקי דיקודר הם שכבות עם תתי השכבות הבאות:

תשומת לב עצמית רב ראשית עם מסכת הסתכלות קדימה

חיבור ונורמליזציה

רשת מחוברת לגמרי

חיבור ונורמליזציה

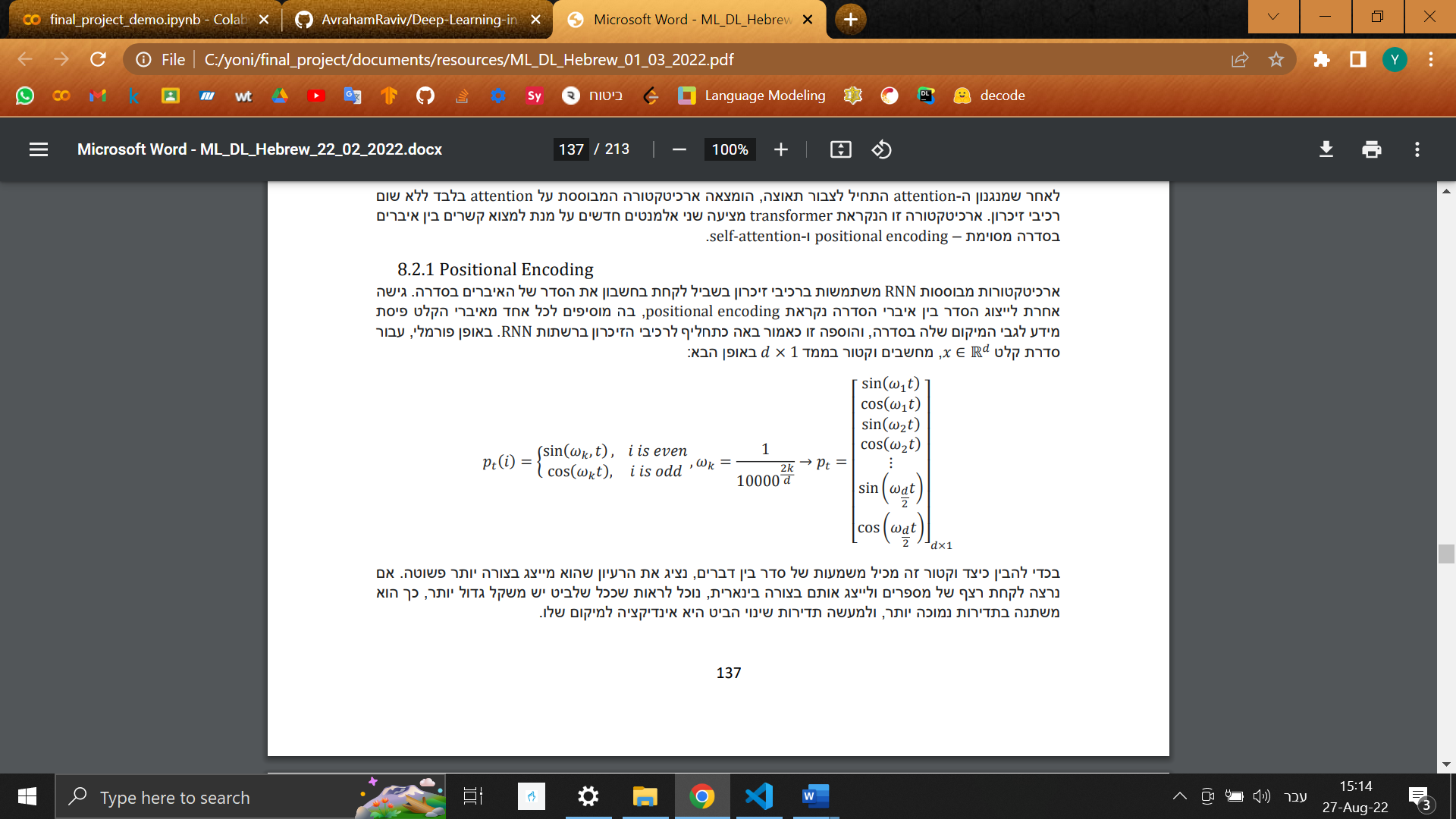
שיכון מיקומי – Positional Encoding\Embedding:

[24] (Vaswani et al., pt.3.5)

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 באורך ניצור מטריצה בגודל ( הוא המימד החבוי של המודל) בה העמודה ה i הוא ווקטור השיכון המקומי של הטוקן במקום i.

את השיכון המקומי אפשר ליצור לפי הנוסחה [24](Vaswani et al., pt.3.5):



או בשיטת שיכון מקומי נלמד [8](Gehring et al., pt.3.1) בה למודל של גודל רצף מקסימלי והוא לומד מטריצת שיכון מקומי בגודל וכשהמודל מקבל רצף בגודל , מטריצת השיכון המקומי שלו היא העמודות הראשונות ממטריצת השיכון המקומי.

הכפלה במטריצת שיכון משוחלפת – Transposed Embedding/Unembedding:

[20](Press and Wolf)

את התוצאה של הדיקודר אנחנו מכפילים במטריצת השיכון המשוחלפת.

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

אינטואיציה:

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

הכפלה במטריצת שיכון היא תרגום של הסתברות לשיכון.

הכפלה במטריצת שיכון משוחלפת היא תרגום של שיכון להסתברות.

*.*

מסכת הסתכלות קדימה look ahead mask:

[24] (Vaswani et al.)

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:

מסכת הסתכלות קדימה תמיד תהיה 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.

תשומת לב עצמית ממוסכת בעזרת מכפלה סקלרית:

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

[24] (Vaswani et al.)

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

נגדיר:

seq len - אורך הקלט לטרנספורמר (מספר חיובי ושלם)

- הממד החבוי – תכונה (היפר-פרמטר) של המודל.

- מטריצות פרמטרים הנלמדים על ידי המודל.

הפונקציה מקבלת מטריצה

הפעולה מחזירה מטריצה באותו גודל של המטריצה שהיא מקבלת.

הגדרה מתמטית לפעולת תשומת 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 מטריצה (קיצור ל dot product) 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.

הסיבוכיות של הפעולה הזו היא וזאת משום שאנחנו מכפילים מטריצות בגודל ומשום שאנחנו מחלקים את כל האיברים בתוצאה ( איברים) בקבוע.

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 סקלרית בין שני ווקטורים מייצגת את הדמיון ביניהם – ככל ששני ווקטורים יותר דומים – המכפלה הסקלרית שלהם יותר גדולה ולהפך. מכפלה סקלרית יכולה להיות חיובית או שלילית. מכפלה סקלרית 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:

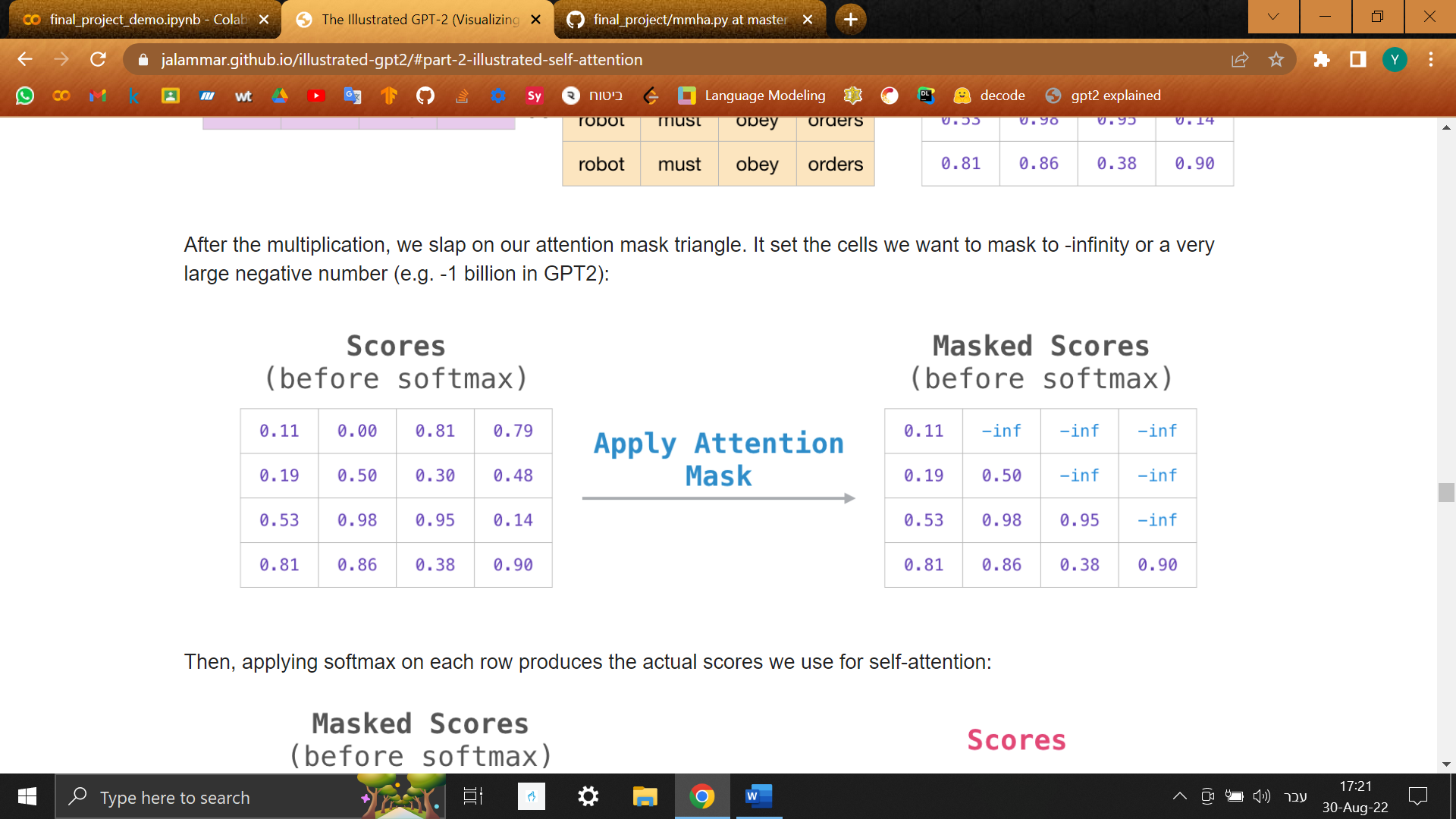
את תוצאת המכפלה מחסרים ממטריצת הדמיון ככה שהדמיון בין

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 תהיה קטנה ככל הניתן.

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



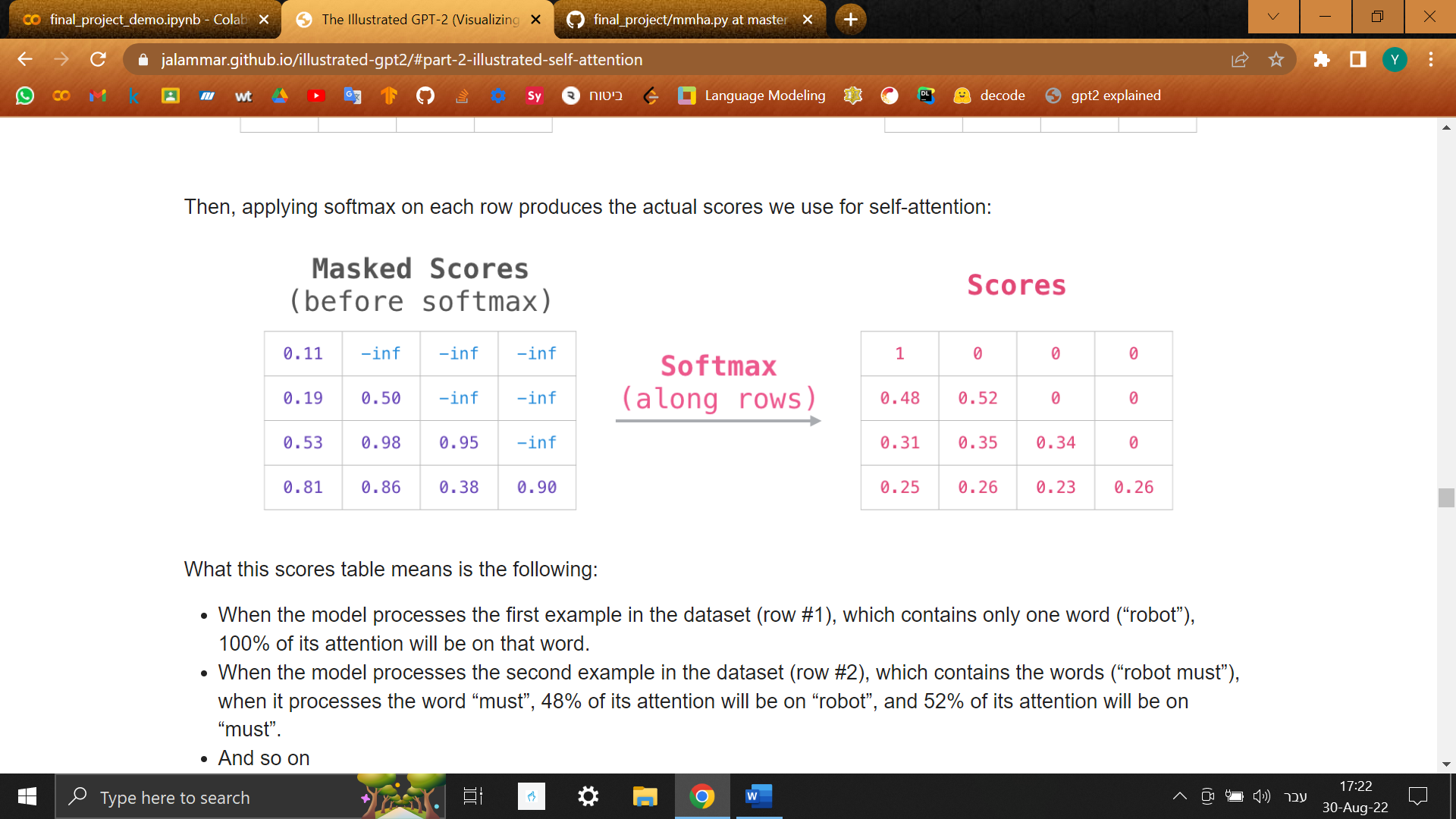
הפעולה :

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 תהיה 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 .

תשומת לב רב-ראשית – Multi Head Attention:

[24](Vaswani et al.)

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

(הממד החבוי של המודל חייב להתחלק ב מספר הראשים).

הערך החבוי האחרון והערך לפני טרנספורמציה מפוצלים ככה ש הממדים 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 תשומת הלב עם הערך החבוי האחרון והערך לפני טרנספורמציה שלו ולומד פרמטרים אחרים.

לפיצול לראשים שתי מטרות:

להקטין את זמן החישוב:

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

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.

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 (היפר-פרמטר של המודל)

אקטיבציית ReLU.

שכבה דחוסה עם גודל קלט feed forward depth וגודל פלט .

שכבה דחוסה\מחוברת לגמרי – Dense/Fully Connected Layer:

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:

[3](Agarap)

הפונקציה פועלת על כל איבר בטנזור ונוסחתה:

הנגזרת של הפונקציה (לפי המימוש בספריות למידת מכונה) היא:[[1]](#footnote-1)

ריפוד - Padding:

בלמידה עמוקה אנחנו הרבה פעמי רוצים לשלוח למודל כמה דוגמאות בו זמנית (לעשות batching).

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.

הטוקן שהמודל חוזה במקום שבו שמנו את טוקן הריפוד הוא הטוקן הבא בטקסט.

מסכת ריפוד מאפשרת את חישוב פונקציית המטרה כך שהמשקל של טוקני הריפוד הוא 0 והמודל לא לומד לחזות את טוקני הריפוד. זאת משום שטוקן ריפוד תמיד יבוא אחרי טוקן ריפוד או הטוקן המציין סוף טקסט וכי טוקן הריפוד הוא הטוקן הנפוץ ביותר בסט האימון בפער גדול.

**אימון ראשוני למידול שפה סיבתי בעזרת יצירת שפה:**

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

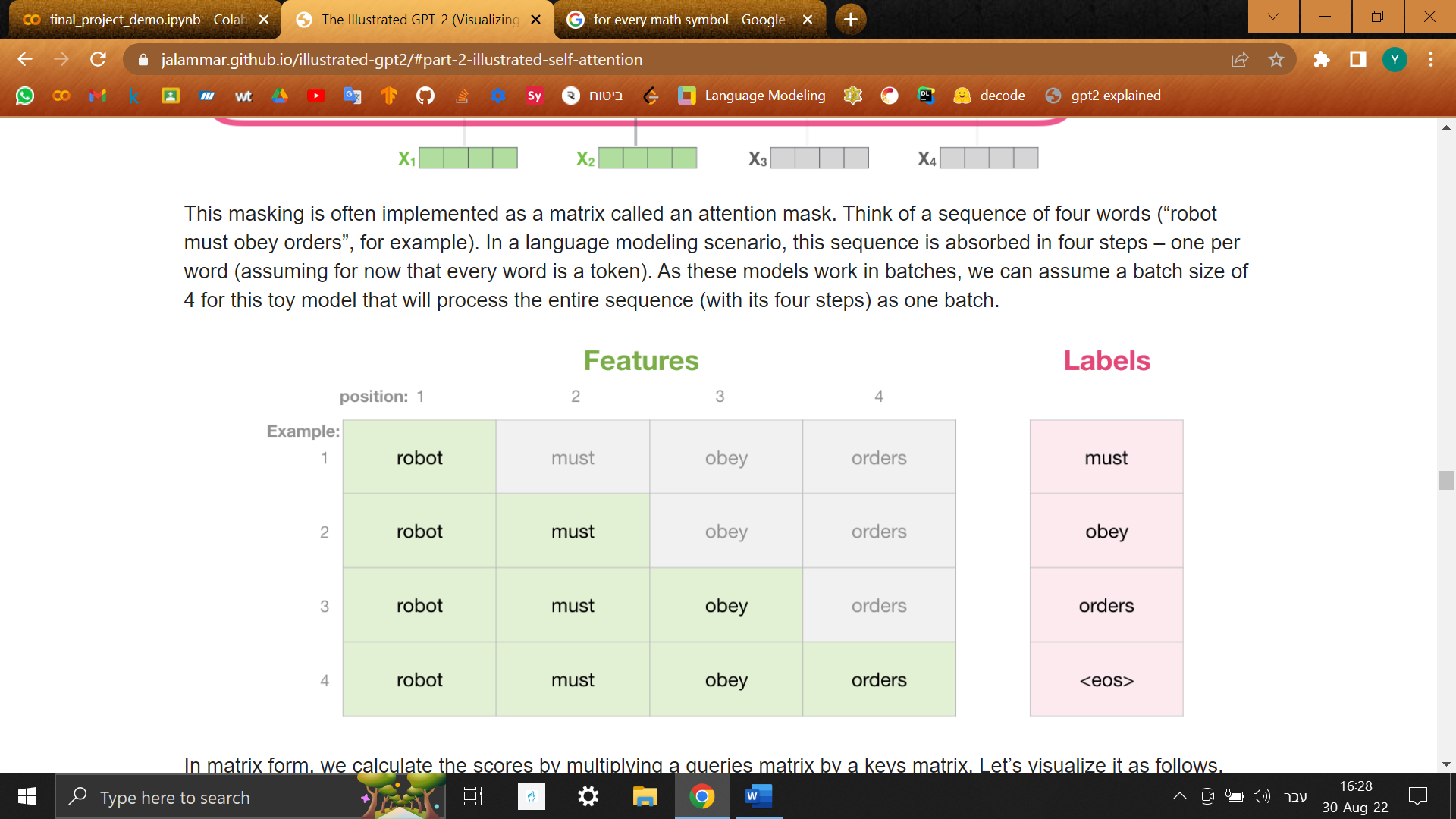
[18](Papers with Code - Improving Language Understanding by Generative Pre-Training) היא שיטה לאימון מודלים שמטרתם ליצור טקסט בהינתן טקסט. בשיטה זו, מאמנים מודל שפה סיבתי מתחיל בסט נתונים גדול עם טקסטים כלליים שעליו המודל מתאמן על מנת ליצור הבנה כללית של שפה אנושית כתובה.

משימה זו היא משימת לימוד בהנחיה עצמית:

המודל לומד לחזות או ליצור חלק אחד מהדוגמה בהינתן 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 בטקסטים כלליים 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 (downstream task) הוא שהמודל לומד להבין שפה כללית ולייצג 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 בדיוק כמו האימון הכללי.

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 תנאי עצירה.

תנאי העצירה יכול להיות הגעה לכמות מסוימת של טוקנים או בחירה של טוקן מיוחד המעיד על סוף הטקסט.

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

דגימה לפי הסתברות מקסימלית – Argmax/Greedy Sampling:

היא שיטת הדגימה הנאיבית. בהתחילה 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 ליישום.
* אין את הסיכון של לדגום טוקן עם הסתברות ממש נמוכה.
* אין צורך לבצע את פעולת ה SoftMax מכיוון שהטוקן עם ערך הלוג'יט הכי גדול בהכרח יהיה הטוקן עם ההסתברות הגבוהה ביותר.

חסרונות:

שיטה זו נוטה ליצר את הטוקנים שמופיעים הרבה בסט האימון של המודל (בדרך כלל מילות קישור) ביחס בלתי פרופורציונלי.

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 מתוך הסתברות – Pure Sampling:

נדגום ברנדומליות לפי ההסתברות שהמודל חזה בווקטור ההסתברות.

היתרונות:

* דגימה מגוונת יותר.
* פחות חזרתית.

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:

(Fan et al. 5)[7]

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

בהינתן ווקטור הסתברות:

נמצא את k הטוקנים שהסתברותם 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 כטוקן שיש פחות מ k טוקנים שהסתברותם גבוהה מהסתברותו – הגדרה שיכולה ליצור בעיות במקרי קצה.
* תלות ברכיב רנדומלי.
* סיבוכיות זמן ריצה של ולא כמו שיטות הדגימה הקודמות שהצגתי.

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:

[11]*(*Holtzman et al.*)*

נבחר מספר p בין אפס ואחד.

ניצור ווקטור הסתברות חדש בו כל ההסתברויות 0.

ונעקוב אחרי סכום ההסתברויות.

נעבור על ווקטור ההסתברויות המקורי לפי סדר:

נעבור על הטוקנים לפי סדר ההסתברות כל עוד סכום ההסתברויות קטן מ p:

בכל פעם נוסיף לווקטור החדש את הטוקן הנוכחי בצורה הבאה:

|  |
| --- |
| new\_probs[token\_id] = curr\_token\_prob.prob |

ונוסיף את ההסתברות לסכום ההסתברויות

נעבור על הווקטור ונחלק את ההסתברות של כל טוקן בסכום ההסתברויות (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.

חסרונות:

* אין חסם המנוע בחירת טוקנים בעלי הסתברות נמוכה.
* תלות ברכיב רנדומלי.
* צריך למיין את ווקטור ההסתברות.

חיפוש עץ – Tree Sampling:

נגדיר את ההסתברות של רצף T בו l טוקנים :

נבחר מספר ונקרא לו רוחב העץ.

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

לכל טוקן שבחרנו, נבחר 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 כשמייצרים רצפים ארוכים.

כאשר n הוא מספר הטוקנים שאנחנו רוצים לחזות.

אנחנו בעצם יוצרים עץ בו לכל הורה w ילדים ויצירה של ילדים נעשית באמצעות קריאה למודל ולכן אנחנו קוראים למודל פעמים וסיבוכיות זמן הריצה של המודל היא ולכן זמן הריצה של הדגימה הוא 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 סיבוכיות זמן הריצה היא ולכן זמן הריצה הוא אקספוננציאלי.

דגימה עם עונשים - penalized sampling:

[12](Keskar et al.)

על מנת למנוע טקסט שחוזר על עצמו, דגימה עם עונשים מקטינה את הלוג'יט (וכתוצאה מכך את ההסתברות) של טוקנים שהופיעו כבר ברצף שהטוקנים שהמודל יצר פי θ כאשר θ פרמטר גדול מ 1. במאמר [12] (Keskar et al.) שהציג את השיטה, החוקרים דיווחו על תוצאות מקסימליות כאשר θ=1.2.

הבעיות עם שיטות הדגימה הקיימות:

כל שיטות הדגימה הקיימות מבססות על תהליך דומה:

האלגוריתם מוצא את הטוקן הבא ברצף ולאחר מכן משתמש בו על מנת למצוא את הטוקן שבא אחריו וככה הלאה.

1. במידה והמודל טועה ביצירת אחד הטוקנים, כל הטוקנים שיבואו אחריו יכילו את הטעות הזאת.

לדוגמה: במידה ואני שואל את המודל שאלה, ואחד הטוקנים מכיל מידע שגוי, כל המשך התשובה תתבסס על אותה פיסת מידע שגוי.

יש לזכור שהמודל אומן בצורה שונה – המודל אומן כאשר כל הטוקנים שהוא קיבל הם הטוקנים שבאמת הופיעו בטקסט, המודל לא אומן להשלים את הטוקנים שהוא בעצמו יצר והוא לא אומן להשלים טקסטים שיש בהם טעויות שצריכות תיקון.

1. הבעיה השנייה עם שיטות דגימה אלו היא שעל מנת ליצור רצף של n טוקנים, צריך להשתמש במודל לפחות n פעמים – וידוע שכל שימוש במודל שפה סיבתי הוא פעולה עם עלות חישוב גבוהה מאוד.

מדד ברט – BERT Score:

[27](Zhang et al.)

מדד ברט מודד את הקרבה הסמנטית בין טקסט אחד מועמד לטקסט מטרה בעזרת מודל שפה מסוג BERT – מודל השפה מקבל טקסט ונותן לכל טוקן ווקטור שמייצג את המשמעות של הטוקן בתוך הקונטקסט של שאר המילים במשפט.

על מנת ליצור את מדד ברט, יוצרים לכל טקסט ייצוג כרצף ווקטורים.

לכל טוקן בטקסט המועמד מוצאים את הטוקן הכי קרוב עליו סמנטית בעזרת מכפלה סקלרית בין הווקטורים המייצגים את הטוקנים (הווקטורים מנורמלים כך שהמכפלה הסקלרית ביניהם שקולה לדמיון קוסינוס שלהם). והתוצאה של הטוקן היא המכפלה הסקלרית שלו עם הטוקן הכי קרוב. את התוצאות של הטוקנים סוכמים לסקלר.

כאשר x הוא המטריצה המתארת את טקסט המטרה ו y הוא הטקסט המועמד הנוסחה היא:

**b** הוא קו ההתחלה (baseline) שערכו הוא מדד ברט (precision recall or F1 בהתאמה) הממוצע לזוגות משפטים רנדומליים מסט נתונים גדול.

השקילה למדד נתון שנקרא לו PS הוא:

המדדים שאציג בתוצאות הם לאחר שקילה.

תצפיות על התנהגותם של מודלי שפה סיבתיים:

[17](nostalgebraist)

כבר בבלוק הדיקודר הראשון, הדיקודר יוצר תחזית חלקית על הטוקנים הבאים ברצף, החל מבלוק הדיקודר השני, הדיקודר מתחיל לדייק את תחזיות אלו באופן חזרתי עד שבבלוק הדיקודר האחרון הוא פולט את התחזיות אליהן הוא הגיע בשלב זה.

סקירת פייטון

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

מטרת הסקירה לאפשר לאנשים שלא מכירים את שפת פייטון להבין את פיתוח התכונה כמו גם להסביר מושגים במדעי המחשב שהקוראים לא בהכרח מכירים.

רשימת מבני נתונים בפייטון ומבני הנתונים המקבילים בשפות אחרות:

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

(tuple) – מערך שאינו ניתן לעריכה. מושג מקביל: immutable 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:

[23](van Rossum et al.)

מערכת הטיפוסים בפייטון היא דינמית כלומר הטיפוסים בפייטון נקבעים בזמן הריצה ולא בזמן 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: from\_dict, as\_dict, \_\_init\_\_, len, str…

פעולות קסם – Magic/Dunder Methods:

פעולות קסם הן פעולות שמורות בשפת פייטון שיש להן תפקיד מיוחד:

פונקציות קסם ממשות את עקרון הרב צורתיות – מחלקות רבות ממשות כל פונקציית קסם.

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

לדוגמה: הפונקציה הבונה\מאתחלת - \_\_init\_\_ שנקראת כשפותחים סוגריים בצמוד לשם המחלקה, \_\_call\_\_ שנקראת כשפותחים סוגריים בצמוד לשם של משתנה, \_\_del\_\_ שנקראת כשמוחקים עצם באמצעות המילה השמורה del, ועוד רבות.

קשטנים - :Decorators

(Smith)[13]

קשטן היא פעולה שמקבלת פונקציה או מחלקה ומחזירה 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 |

הסינטקס לשימוש בקשטן:

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

מקביל לסינטקס:

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

מחלקת בסיס אבסטרקטית:

[9](Guido van Rossum and Talin)

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 אבסטרקטית 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 אבסטרקטית:

[9] (Guido van Rossum and Talin)

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

לדוגמה – במחלקה המייצגת צורה דו ממדית נרצה שלכל אחת מתת המחלקות תהיה פונקציה המחשבת שטח ובעזרת נוסחה שונה לכל תת מחלקה. לשם כך ניצור פונקציה בשם calculate area שלא מקבלת פרמטרים ומחזירה את השטח במטרים רבועים כמספר עשרוני.

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

פעולה סטטית:

[21](Rossum, pt.2)

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 מממשת פעולה שקשורה למהות המחלקה.  
פונקציות בונות הן פונקציות סטטיות.

על מנת ליצור פעולה סטטית, אשתמש בקשטן staticmethod.

מחלקת נתונים – Data Class:

[6](Eric V. Smith)

הקשטן dataclasses.dataclass יוצר מחלקה לייצוג נתונים בעזרת שמות ורמזי הסוג של משתני המחלקה במחלקה שהוא מקבל.

הקשטן יוצר פונקציות כגון: אתחול - \_\_init\_\_, ייצוג כמחזורת - \_\_str\_\_ ובדיקת שוויון: \_\_eq\_\_.

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

המחלקה האבסטרקטית Callable:

[21](Guido van Rossum and Talin)

היא מחלקה המגדירה עצם קריא.

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

My\_callable(arg1, key\_word=arg2)

מקביל לסינטקס:

My\_callable.\_\_call\_\_(arg1, key\_word=arg2)

הערה: פונקציות ועצמים קריאים הם מונחים מקבילים – פונקציות הן עצמים קריאים ועצמים קריאים הם פונקציות.

Enum:

[25](Warsaw et al.)

מחלקה המייצגת סט קבוע של ערכים בעלי שם שנקבעים על ידי המתכנת.

ערמת מקסימום\מינימום – Maximum/Minimum Heap:

[10](‘Heapq — Heap Queue Algorithm’)

ערמת מינימום\מקסימום היא עץ בינארי כמעט שלם בו כל אב קטן (בערמת מינימום)\גדול (בערמת מקסימום) מבניו. כתוצאה מכך, השורש הוא החולייה בעלת הערך הקטן (בערמת מינימום)\גדול (בערמת מקסימום) ביותר בעץ וניתן להוציא אותה מהעץ בסיבוכיות זמן  ניתן להפוך רשימה לערמת מינימום בסיבוכיות זמן ריצה .

בעזרת ערימה ניתן למצוא את k הערכים בעלי הערכים הקטנים\גדולים ביותר ברשימה שאורכה n בסיבוכיות זמן ריצה בעזרת הפונקציות nleagest ו nsmallelest של הספרייה הסטנדרטית heapq.

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 (בלי קווים תחתונים) היא הפונקציה שנקראת באופן עקיף כשקוראים לפונקציה \_\_call\_\_ של אובייקט 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 ממומשות שכבות שימושיות כגון שכבה דחוסה, שיכון ורגורלריזציה.

כל שכבה מומשה על ידי ירושה מהמחלקה Layer.

המחלקה Model:

גם היא יורשת מ Module.

הפונקציה compile מקבלת מודל והופכת אותו לפונקציה בשפת c++ ולאחר מכן מעבירה אותו קומפילציה. היא מחזירה את הפונקציה לאחר הקומפילציה.

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 compile קוראת לפונקציה compile של תת השכבות והמודלים.

הסבר האלגוריתם

(יש לקרוא את סקירת הבינה המלאכותית לפני קריאת חלק זה)

בגלל שטוקן הריפוד מופיע רק לפני טוקני ריפוד אחרים במהלך האימון, והמשקל של טוקני הריפוד בחישוב פונקציית המטרה הוא 0 (כלומר המודל לא לומד לחזות טוקני ריפוד). פונקציית המטרה לא תלויה בטוקני ריפוד במהלך האימון ולכן גם לא בשיכון של טוקן הריפוד.

מסיבה זו, השיכון של טוקן הריפוד לא משתנה במהלך האימון והוא נשאר השיכון ההתחלתי.

השיכונים ההתחלתיים של כל הטוקנים מוגרלים מאותה התפלגות, המודל לומד את התפלגות זו ויודע לזהות אותה.

שימו לב שכל ווקטור שבלוק הדיקודר הראשון מקבל הוא הסכום של ווקטור השיכון עם ווקטור השיכון המקומי של הטוקן המסוים.

בעצם, המודל יודע לזהות מתי הוא מקבל ווקטור שיכון שהוגרל ולא שונה מאז הגרלתו ומתי הוא מקבל ווקטור שיכון שערכיו נקבעו במהלך האימון. ככה המודל יודע שיש לו מידע רק על מיקום הטוקן ולא על תוכנו.

כאשר אנחנו משלבים את טוקני הריפוד (או כל טוקן אחד שלא הופיע באימון) בקלט, המודל יודע שיש לו טוקן לא מזוהה במיקום מסוים ברצף ויודע לפעול בהתאם.

באלגוריתם שלי אני מוסיף לרצף הפלט, רצף באורך group size – 1 של טוקני ריפוד. ככה המודל יודע מה טוקני הקלט ושיש טוקנים לא ידועים במיקומים מסוימים והוא חוזה את הטוקן הראשון בקלט על סמך טוקני הפלט (כרגיל).

בבלוק הראשון, הדיקודר נותן תחזית ראשונית לgroup size הטוקנים הבאים בטקסט על סמך הטוקנים שנקלטו מהמשתמש בלבד. בשאר הבלוקים (הבלוק השני עד האחרון), הדיקודר משפר את התחזיות אלו. בבלוקים אלו הדיקודר משפר את התחזיות שלו לטוקנים בפלט על סמך הטוקנים בקלט ועל תחזית הביניים של הטוקנים שמגעים לפניהם בפלט.

בסוף הדיקודר, יש לדיקודר מידע על הטוקן הראשון בפלט (כרגיל) וכן על group size - 1 הטוקנים שאחריו. כך **האלגוריתם מוצא** את ווקטורי הלוג'יט של **group size הטוקנים הבאים באמצעות קריאה אחת בלבד למודל שפה סיבתי**.

ווקטורי הלוג'יט משמשים את האלגוריתם על מנת למצוא את הטוקנים עצמם.

אינטואיציה:

LHM – look ahead mask – מסכת הסתכלות קדימה.

יצירת הטוקן הבא הבא (בקבוצות של טוקן אחד):

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Next token |  |  |  |  |  |
| טוב | [LHM] | [LHM] | [LHM] | [LHM] | בוקר |
| לכולם | [LHM] | [LHM] | [LHM] | טוב | בוקר |
| , | [LHM] | [LHM] | לכולם | טוב | בוקר |
| היום | [LHM] | , | לכולם | טוב | בוקר |
| אנחנו | היום | , | לכולם | טוב | בוקר |

יצירת שני הטוקנים הבאים (בקבוצות של שני טוקנים):

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Next token |  |  |  |  |  |
| לכולם | [LHM] | [LHM] | [LHM] | טוב | בוקר |
| , | [LHM] | [LHM] | [PAD] | טוב | בוקר |
| היום | [LHM] | , | לכולם | טוב | בוקר |
| אנחנו | [PAD] | , | לכולם | טוב | בוקר |

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

פיתוח התכונה

בחלק זה של העבודה אעסוק בפירוט בתכונה שכתבתי ואסביר את חלקי הקוד ואת ההחלטות שלקחתי בכתיבת התוכנה.

את מימוש האלגוריתם עצמו, פרסמתי במאגר חבילות הקוד הפתוח של שפת פייטון – PyPi

ניתן להוריד את החבילה באמצעות הפקודה:

python -m pip install -q grouped-sampling

דוגמה בסיסית לשימוש באלגוריתם:

from grouped\_sampling import GroupedSamplingPipeLine

pipe = GroupedSamplingPipeLine(model\_name="facebook/opt-125m", group\_size=1024)

answer = pipe("this is an example prompt")["generated\_text"]

המחלקה RepetitionPenaltyStrategy:

מחלקה אבסטרקטית המגדירה פונקציה אבסרקטית \_\_call\_\_ שמקבלת מטריצת לוג'יט ואת רצף הטוקנים שהשתמשו בהם על מנת ליצור את המטריצה ומחזיר מטריצת לוג'יט חדשה.

למחלקה שתי מחלקות בנות:

LogitScalingRepetitionPenalty מחלקת את ווקטורי הלוג'יט שמייצגים טוקן שהופיע ברצף בפרמטר (theta).

NoRepetitionPenalty לא משנה את מטריצת הלוג'יט כלל. היא מקבילה למחלקה LogitScalingRepetitionPenalty כאשר theta שווה אחד.

repetition\_penalty\_factory מקבלת את הפרמטר theta ומחזירה עצם מסוג RepetitionPenaltyStrategy בהתאם לערכו של theta.

המחלקה GroupedGenerationUtils:

משתני המחלקה:

מודל (AutoModelForCausalLM), האם להשתמש ב softmax (bool), המזהה של הטוקן של סוף טקסט (int), אסטרטגיית הענשה לטוקנים שהופיעו בקלט (RepetitionPenaltyStrategy), גודל הקבוצה (int), גודל מקסימלי של הקלט למודל (int), המזהה של טוקן הריפוד (int), האם לעצור בטוקן של סוף הטקסט? (bool), טמפרטורה (float) וגודל מילון (int).

הפונקציה prepare\_model\_kwargs מקבלת רצף טוקנים ומחזירה את הקלט לפונקציה \_\_call\_\_ של המודל בפורמט המתאים. אחד הדברים שהיא עושה היא הוספת group size – 1 טוקני ריפוד לרצף הקלט.

הפונקציה get\_logit\_mat מקבלת רצף של m טוקנים ומחזירה מטריצה בגודל (group size, vocab size) בה logit\_mat[i] הוא ווקטור הלוג'יט של הטוקן שמגיע במקום m+1+i. הווקטור במקום 0 הוא ווקטור הלוג'יט של הטוקן במיקום m+1 ברצף (הטוקן שבא לאחר סוף הקלט), הווקטור במקום 1 הוא ווקטור הלוג'יט של הטוקן במקום m+2 וככה הלאה...

הפונקציה משתמשת בפונקציה \_\_call\_\_ של מודל שפה סיבתי שהסיבכויות שלה עם רצף בגודל m+group size-1 ולכן הסיבוכיות של הפונקציה היא .

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

ומחזירה מטריצת הסתברות בגודל (group size, vocab size) הבנויה כמו מטריצת הלוג'יט מהפונקציה get\_logit\_mat אך הווקטורים בה הם ווקטור הסתברות ולא ווקטורי לוג'יט.

הפונקציה אחראית על הקטנת הלוג'יטים של הטוקן המיוחד המעיד על סוף טקסט במקרה בו מספר הטוקנים הרצוי ידוע מראש, היא אחראית על הקטנת הלוג'יט של טוקנים שהופיעו בקלט באמצעות אסטרטגיית הענשה (RepetitionPenaltyStrategy) ועל המרת ווקטורי לוג'יט להסתברות.

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

המחלקה PostProcessor:

מחלקה המייצגת אובייקט קריא שתפקידו להמיר את רצף הטוקנים שהאלגוריתם יצר לפורמט הרצוי.

משתנה המחלקה: טוקנייזר (PreTrainedTokenizer).

הפונקציה \_\_call\_\_ מקבלת: את רצף הטוקנים שנוצר על ידי \_forward, המספר המקסימלי של הטוקנים הנוצרים, גודל הפרומפט והתוספות, האם להחזיר את הטקסט כמחרוזת, האם להחזיר את הטקסט כרצף טוקנים, האם להשאיר את התוספות כחלק מההשלמה, והאם לחבר טוקנים שונים למילה אחת.

הפונקציה מחזירה את הטקסט שנוצר על ידי האלגוריתם כמילון עם המפתחות:

"generated\_text", " generated\_token\_ids"

המחלקה PreProcessor:

מייצגת אובייקט קריא שתפקידו להמיר את הטקסט שנקלט מהמשתמש לרצף של מזהיי טוקנים.

משתני המחלקה:

טוקנייזר (PreTrainedTokenizer) וגודל קלט מקסימלי למודל (int).

הפונקציה preprocess מקבלת את הפרומפט, תוספות לפרומפט (תחילית וסופית) ואסטרטגיית קיטוע. היא מחזירה רצף טוקנים שמכיל את הטוקנים של הפרומפט והתוספות ואת האורך (בטוקנים) של הפרומפט והתוספות.

הפונקציה get\_token\_tensor היא פונקציית עזר של preprocess ופונקציית מעטפת לפונקציה \_\_call\_\_ של הטוקנייזר שהופכת מחרוזת לטנזור של שלמים המכיל רצף של מזהיי טוקנים.

המחלקה :GroupedGenerationPipeLine

היא מחלקת בסיס אבסטרקטית.

המחלקה מייצגת אובייקט קריא שמקבל טקסטים מהמשתמש ומחזיר טקסטים שנוצרו על ידי מודל שפה סיבתי.

משתני המחלקה המרכזיים:

אסרטגיות עיבוד ראשוני וסופי (pre\_processing\_strategy, post\_processing\_strategy) מסוג PostProcessor ו PreProcessor בהתאם, שם מודל (מחזרות), מודל עטוף (ModelWrapper) וטוקנייזר מסוג (transformers.PreTrainedTokenizer).

answer\_length\_multiplier (מספר עשרוני-float). מכפיל אורך התשובה משמש כחסם מקסימלי לאורך הטקסטים שהאלגוריתם מייצר. האלגוריתם יכול ליצור טקסטים שאורכם קטן או שווה לעיגול למטה של מכפלת אורך הפרומפט במכפיל אורך התשובה.

בחרתי להוסיף חסם מקסימלי לאורך הטקסט הנוצר מכיוון ששמתי לב שהאלגוריתם מייצר טקסטים ארוכים במיוחד ושיערתי שזה יפגע ביכולת למדוד את איכות הטסטים בעזרת מדד ברט.

הפונקציה \_\_call\_\_ מקבלת פרומפט אחד או יותר כמחרוזת, תוספות (תחילית וסופית) לפרומפט, מספר ההשלמות השונות לכל פרומפט, האורך המקסימלי של כל השלמה, אסטרטגיית קטיעות (לטוקנייזר) ופרמטרים בוליאניים שקובעים: האם להחזיר את ההשלמה כמחרוזת והאם להחזיר אותה כרצף טוקנים כרצף טוקנים והאם לחבר טוקנים שונים למילה אחת.

הפונקציה האבסטרקטית \_forward מקבלת רצף של הטוקנים בפרומפט ובתוספות (הרצף שנוצר בפונקציה preprocess), מספר הטוקנים המקסימלי ליצור ומספר הרצפים השונים ומחזירה רצף\רצפים של טוקנים הכולל את הטוקנים מהפרומפט והתוספות ואת הטוקנים שהאלגוריתם בחר.

המחלקה GroupedTreePipeLine:

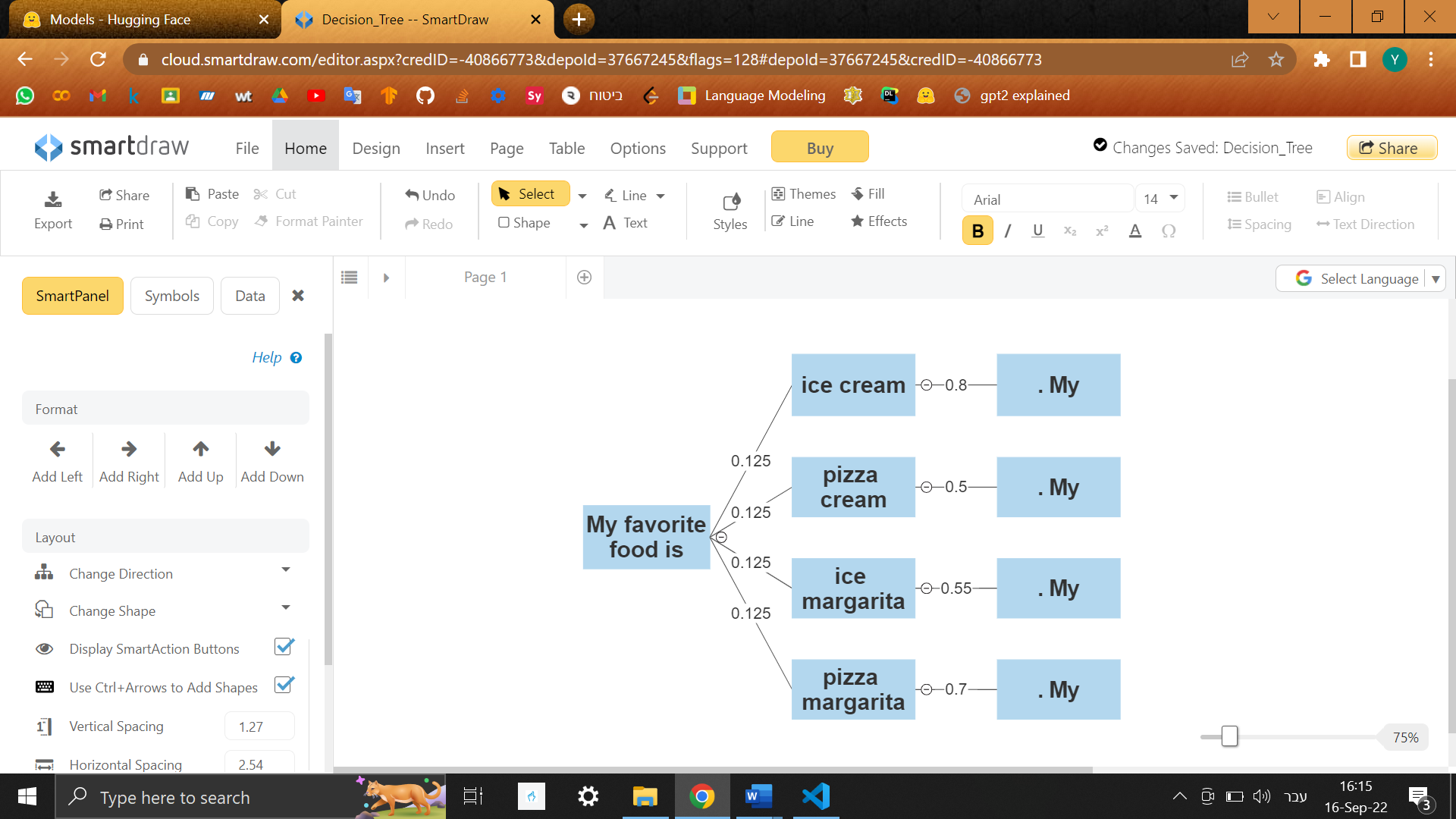
יורשת מהמחלקה GroupedGenerationPipeLine ומייצרת טקסט בעזרת חיפוש עץ בקבוצות:

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 הפרומפט "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 שייווצר הוא: "My favorite food is ice cream. My" מכיוון שהסתברותו הכי גבוהה.

במימוש זה אין מגבלה הקובעת שn טוקנים ייווצרו בעזרת פחות מ n קריאות למודל.

ולכן האלגוריתם הזה לא עונה על שאלת המחקר שלי וחיפשתי אלגוריתם חלופי.

המחלקה GroupedSamplingPipeLine:

יורשת מהמחלקה TextGenerator ומייצרת טקסט בעזרת **דגימה בקבוצות.**

בהתאם לפרמטרים של הגנרטור, הוא משתמש מאסטרטגיות הדגימה הבאות:

unfiltered\_sampling: משתמשת בדגימה מתוך הסתברות.

highest\_prob\_token: משתמשת בדגימת הסתברות מקסימלית.

:top\_k\_sampling דגימה מתוך k הטוקנים שהסתברותם הכי גבוה כאשר k הוא המשתנה top\_k.

:top\_p\_sampling משתמשת דגימה מתוך הטוקנים שסכום הסתברותם <= top\_p.

כל פונקציות הדגימה מקבלות ווקטור באורך קבוע (גודל המילון) ולכן הסיבוכיות שלהם .

הפונקציה generate\_group מקבלת מטריצת הסתברות ומחזירה קבוצה של טוקנים שאורכה הוא גודל הקבוצה.

הפונקציה עוברת בלולאה על כל ווקטורי ההסתברות במטריצת ההסתברות ודוגמת טוקן מכל ווקטור ומוסיפה אותו לקבוצה. סיבוכיות הפונקציה היא .

**מימוש הפונקציה \_forward:**

נגדיר: n – מספר הטוקנים שהפונקציה יוצרת, m – מספר הטוקנים בפרומפט, g-גודל הקבוצה

היא מגדירה את הרצף הנוכחי לטוקנים של הפרומפט.

ומתחילה לולאה:

בכל חזרה היא יוצרת מטריצת הסתברות בצורה [גודל הקבוצה, גודל המילון] בעזרת הפונקציה create\_prob\_mat כאשר הפונקציה מקבלת את הרצף הנוכחי שאורכו המקסימלי הוא: ולכן סיבוכיות השורה היא

היא משתמשת בפונקציה generate\_group על מנת ליצור קבוצת טוקנים ממטריצת ההסתברות בסיבוכיות

ומוסיפה את הטוקנים מהקבוצה לרצף הנוכחי בסיבוכיות .

בסך הכל, הסיבוכיות של כל חזרה היא

הלולאה ממשיכה עד הגעה לטוקן מיוחד שמעיד על סוף המשפט או עד שנוצר המספר הרצוי של טוקנים. בכל מקרה אנחנו יוצרים group size טוקנים בכל חזרה (למעט האחרונה במקרה שדגמנו את טוקן סיום הטקסט באמצע הקבוצה) ולכן יש לנו במקרה הרע חזרות.

לכן סיבוכיות המימוש של הפונקציה \_forward היא:

נזכיר שהפונקציה \_\_call\_\_ (כשהיא מקבלת פרומפט אחד) קוראת רק לפונקציות עם סיבוכיות לינארית ולכן סיבוכיות הפונקציה \_\_call\_\_ (כשהיא מקבלת פרומפט אחד) היא

.

יש לשים לב שכשאר גודל הקבוצה שווה למספר הטוקנים שהפונקציה יוצרת (), האלגוריתם קורא לפונקציה get\_prob\_mat רק פעם אחתולכן זמן הריצה יהיה מינימלי.

סיבוכיות זמן הריצה במקרה זה היא:

דוגמה:

דגימה של 4 טוקנים כאשר גודל הקבוצה הוא 2 עבור הפרומפט "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:

יהיו לנו 2 שורות במטריצה כי גודל הקבוצה שלנו הוא 2, השורה הראשונה חוזה מה יהיה הטוקן אחרי הטוקן is והשורה שאחריה חוזה מה יהיה הטוקן 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...).

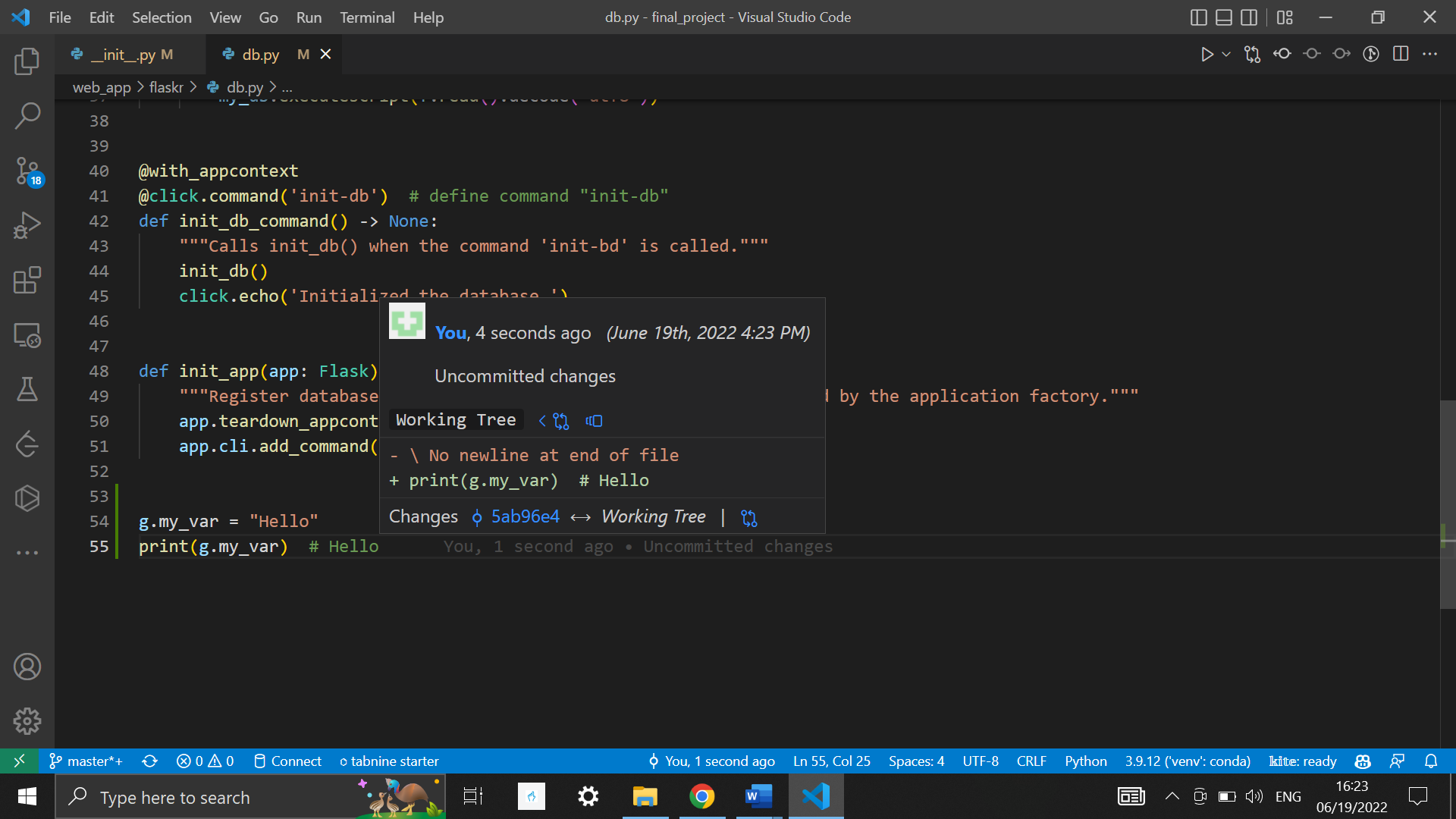
כל ההשלמות נשמרות בבסיס נתונים וכל משתמש יכול לראות את התוצאות של כל שאר המשתמשים.

המחלקה Blueprint:

נועדה על מנת לחלק אפליקציית Flask למספר חלקים, כל חלק תחת תת כתובת (subdomain) אחרת.

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 \_\_init\_\_.py:

כל אפליקציה שמפותחת באמצעות Flask חייבת לכלול קובץ ששמו: "\_\_init\_\_.py" ובו פונקציה הנקראת "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 תיקייה לבסיס הנתונים, יצירת בסיס הנותנים ושמירת blueprints.

הקובץ database.py:

בקובץ database.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 השתמשו לראשונה במודל.

completions – המאכסנת השלמות עם העמודות:

id – מזהה ההשלמה.

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?

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 הsession 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 (decorator) login\_required.

כשקוראים לפונקציה המקושתת בו, הוא בודק שיש משתמש במשתנה הגלובלי (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 ב session 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 טבלת המודלים בבסיס הנתונים בהינתן השם שלו. אם המודל לא נמצא בבסיס הנתונים – הפונקציה מוסיפה אותו.

הקובץ complition.py מכיל את ה blueprint completion.

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

הפונקציה create טוענת את העמוד בו המשתמש 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 התיקייה 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 שיכון מיקומים המתאים להיפר הפרמטרים של המודל ומשמש להגדרת ה decoder ואז יוצרת את השכבות: שיכון, דיקודר ו-(הכפלה במטריצת שיכון 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 Transformer: הן יורשות (באופן ישיר, בשונה מ Transformer) מ Layer ויש להן שתי פונקציות: אתחול וקריאה. פונקציית האתחול יוצרת את תתי השכבות ושומרות אותן במשתני מחלקה. בחלק מהמקרים היא שומרת גם היפר-פרמטרים שהיא מקבלת, חישובי עזר ואת שיכון המיקומים. מתודת הקריאה מקבלת בנוסף למידע שמתואר בחלק התיאורטי גם מסכות שנוצרות בכל קריאה למודל.

הלוגיקה של המחלקות מוסברת בפירוט רב בסקירת בינה מלאכותית.

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 נקראת מתוך מתודת הקריאה של המחלקה Transformer.

הדמו:

הקובץ colab\_demo.ipynb הוא מחברת ג'ופיטר אותה שאותה המשתמש פותח דרך Google Collaboratory בקישור <https://colab.research.google.com/github/yonikremer/grouped_sampling/blob/master/colab_demo.ipynb>

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

ניסויים:

על מנת לבדוק את איכות הטקסטים שהאלגוריתם יוצר, בחרתי את סט הנתונים ted\_talks\_iwslt [15](Mauro Cettolo et al.) המכיל מעל 22,000 תמלילים להרצאות TED שתורגמו על ידי מתרגמים מקצועיים.

נתתי לאלגוריתם שלי לתרגם את הטקסטים ובדקתי את הקרבה בין התרגום שהאלגוריתם יצר לבין התרגום שהמתרגם המקצועי יצר בעזרת מדד ברט.

בניסויים שלי השתמשתי בדגימה מתוך הסתברות ללא מגבלות, במודל opt-125M [26](Zhang et al..) ובטמפרטורה של אחת ושיניתי את גודל הקבוצה בלבד.

בחרתי בדגימה ללא מגבלות ובטמפרטורה של אחת על מנת לשמור על הערכי ברירת המחדל, במודל OPT [26](Zhang et al.) מכיוון שזהו מודל שפה סיבתי שאומן על תרגום טקסטים ארוכים וסט הנתונים שהשתמשתי בו לא הופיע בסט האימון שלו. בחרתי בגרסת 125 מיליון פרמטרים של המודל משיקולי חומרה (מגבלת זיכרון של המעבד הגרפי בו השתמשתי).

הקוד שבו השתמשתי על מנת להריץ את הניסויים נמצא בתיקייה evaluation.

את התוצאות אספתי ריכזתי [פה](https://www.comet.com/yonikremer/grouped-sampling-evaluation/reports).

תוצאות – חלק ראשון

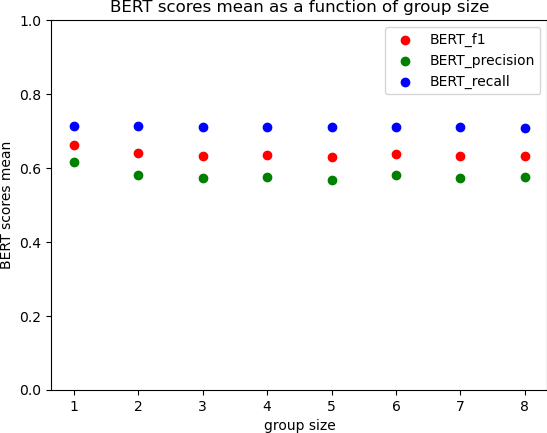
בניסויים הראשוניים הבנתי שבמשימת תרגום אני לא יודע מה יהיה אורך הפלט הרצוי אז שיניתי את הקוד של האלגוריתם ליצירת טקסט ככה שהטקסט יגמר כשהאלגוריתם יצור טוקן מיוחד שקיים בכל מודלי השפה שבא בסוף כל טקסט.

לאחר מכן שמתי לב שהרבה פעמים האלגוריתם יוצר טקסטים מאוד ארוכים (אלפי טוקנים) ולא מגיע לטוקן של סוף המשפט אז החלטתי שהאלגוריתם יעצור לאחר שהוא יוצר טקסט ארוך פי שתיים מהקלט. אורך זה בא מתוך סט הנתונים שלי – היחס המקסימלי בין האורכים של צמד טקסטים הוא כמעט 2.

אלו התוצאות שקיבלתי:

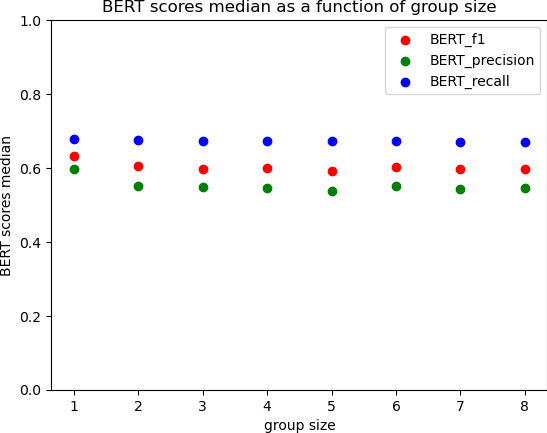
ממוצע:

|  |  |  |  |
| --- | --- | --- | --- |
| group\_size | BERT\_f1 | BERT\_precision | BERT\_recall |
| 1 | 0.662 | 0.617 | 0.714 |
| 2 | 0.639 | 0.581 | 0.713 |
| 3 | 0.634 | 0.573 | 0.711 |
| 4 | 0.635 | 0.576 | 0.71 |
| 5 | 0.629 | 0.567 | 0.711 |
| 6 | 0.637 | 0.58 | 0.71 |
| 7 | 0.633 | 0.572 | 0.71 |
| 8 | 0.634 | 0.574 | 0.709 |



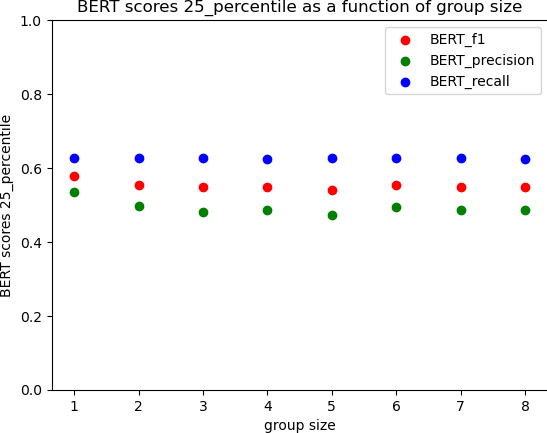
חציון:

|  |  |  |  |
| --- | --- | --- | --- |
| group\_size | BERT\_f1 | BERT\_precision | BERT\_recall |
| 1 | 0.632 | 0.596 | 0.677 |
| 2 | 0.604 | 0.552 | 0.676 |
| 3 | 0.598 | 0.549 | 0.673 |
| 4 | 0.6 | 0.547 | 0.672 |
| 5 | 0.592 | 0.537 | 0.672 |
| 6 | 0.603 | 0.551 | 0.673 |
| 7 | 0.597 | 0.543 | 0.671 |
| 8 | 0.598 | 0.546 | 0.671 |



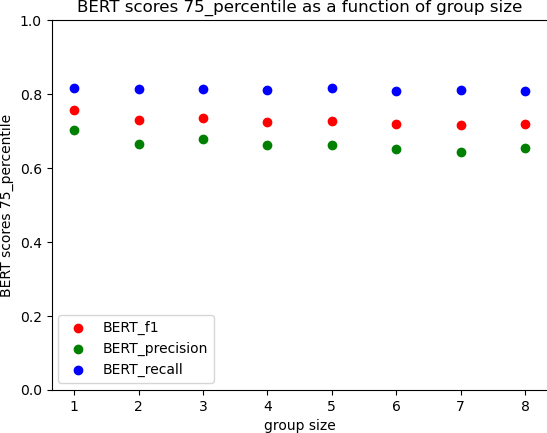
אחוזון 25:

|  |  |  |  |
| --- | --- | --- | --- |
| group\_size | BERT\_f1 | BERT\_precision | BERT\_recall |
| 1 | 0.578 | 0.536 | 0.628 |
| 2 | 0.554 | 0.496 | 0.627 |
| 3 | 0.548 | 0.481 | 0.627 |
| 4 | 0.548 | 0.486 | 0.625 |
| 5 | 0.542 | 0.473 | 0.626 |
| 6 | 0.555 | 0.494 | 0.627 |
| 7 | 0.548 | 0.485 | 0.626 |
| 8 | 0.548 | 0.486 | 0.626 |



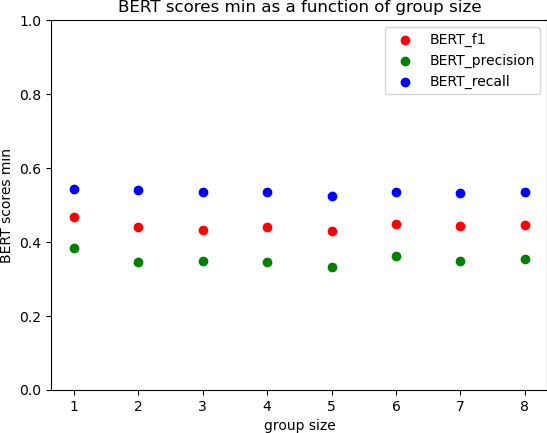
אחוזון 75:

|  |  |  |  |
| --- | --- | --- | --- |
| group\_size | BERT\_f1 | BERT\_precision | BERT\_recall |
| 1 | 0.757 | 0.703 | 0.816 |
| 2 | 0.729 | 0.666 | 0.815 |
| 3 | 0.736 | 0.679 | 0.812 |
| 4 | 0.725 | 0.662 | 0.811 |
| 5 | 0.727 | 0.662 | 0.817 |
| 6 | 0.719 | 0.652 | 0.809 |
| 7 | 0.715 | 0.643 | 0.811 |
| 8 | 0.719 | 0.653 | 0.81 |



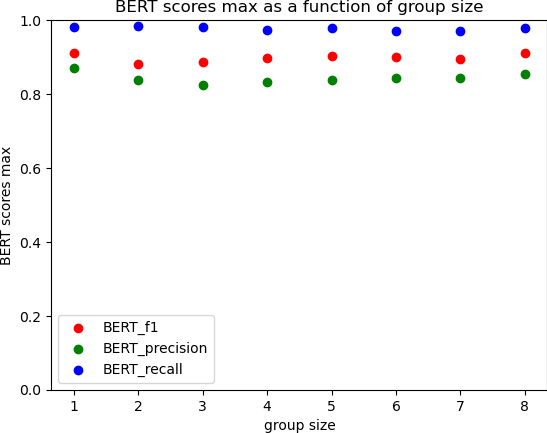
מינימום:

|  |  |  |  |
| --- | --- | --- | --- |
| group\_size | BERT\_f1 | BERT\_precision | BERT\_recall |
| 1 | 0.468 | 0.383 | 0.542 |
| 2 | 0.44 | 0.346 | 0.541 |
| 3 | 0.433 | 0.348 | 0.536 |
| 4 | 0.44 | 0.345 | 0.535 |
| 5 | 0.43 | 0.331 | 0.525 |
| 6 | 0.449 | 0.362 | 0.535 |
| 7 | 0.444 | 0.348 | 0.532 |
| 8 | 0.445 | 0.353 | 0.536 |



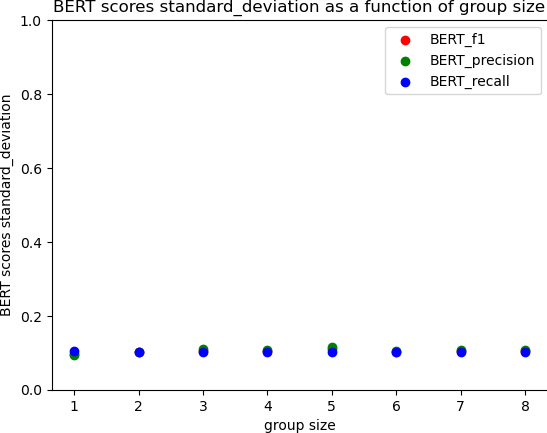
מקסימום:

|  |  |  |  |
| --- | --- | --- | --- |
| group\_size | BERT\_f1 | BERT\_precision | BERT\_recall |
| 1 | 0.912 | 0.87 | 0.982 |
| 2 | 0.881 | 0.838 | 0.983 |
| 3 | 0.888 | 0.824 | 0.981 |
| 4 | 0.897 | 0.833 | 0.974 |
| 5 | 0.902 | 0.837 | 0.98 |
| 6 | 0.9 | 0.845 | 0.971 |
| 7 | 0.895 | 0.844 | 0.971 |
| 8 | 0.912 | 0.855 | 0.978 |



סטיית תקן:

|  |  |  |  |
| --- | --- | --- | --- |
| group\_size | BERT\_f1 | BERT\_precision | BERT\_recall |
| 1 | 0.097 | 0.093 | 0.104 |
| 2 | 0.102 | 0.103 | 0.103 |
| 3 | 0.106 | 0.11 | 0.102 |
| 4 | 0.105 | 0.108 | 0.102 |
| 5 | 0.109 | 0.115 | 0.103 |
| 6 | 0.102 | 0.105 | 0.101 |
| 7 | 0.105 | 0.108 | 0.102 |
| 8 | 0.105 | 0.108 | 0.102 |



הדבר הראשון ששמתי לב אליו כמעט ואין השפעה על איכות הטקסטים. תוצאות אלו מראות על הצלחת המחקר בבירור – האלגוריתם משפר את זמן הריצה בלי לפגוע משמעותית באיכות התוצרים.

שמתי לב גם שה recall גדול משמעותית מה מה precision .

מתוך הנוסחאות (ראה סקירת בינה מלאכותית – מדד ברט):

כאשר x הוא המטריצה המתארת את טקסט המטרה ו y הוא הטקסט המועמד

ולכן הוא אורך הטקסט הרצוי ו הוא אורך הטקסט שהאלגוריתם יוצר.

ולכן אם

נחלק את הנוסחאות של המדדים ב (ידוע ש ) ונקבל

נכפיל ב (ידוע ששני האורכים חיוביים ולכן מכפלתם חיובית ואפשר להכפיל בה בלי לשנות סימנים) ונקבל:

כלומר אורך הטקסט שהאלגוריתם יוצר גדול מאורך הטקסט הרצוי.

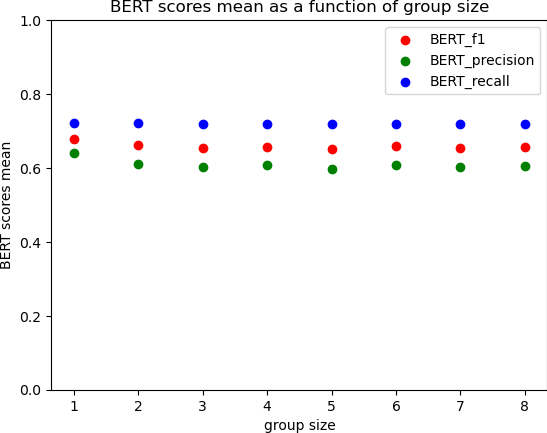
תוצאות החלק השני

על מנת להקטין את אורכי הטקסטים שהאלגוריתם יוצר, שיניתי את הפרמטר answer\_length\_multiplier מ 2 ל 1.25 ככה שהאלגוריתם לא יצור טקסטים ארוכים ביותר מ 25% מהטקסטים שהוא מקבל.

מאחר ששיניתי פרמטר שעשוי להשפיע על איכות הטקסטים, התחלתי סדרת ניסויים חדשה עם הפרמטר החדש.

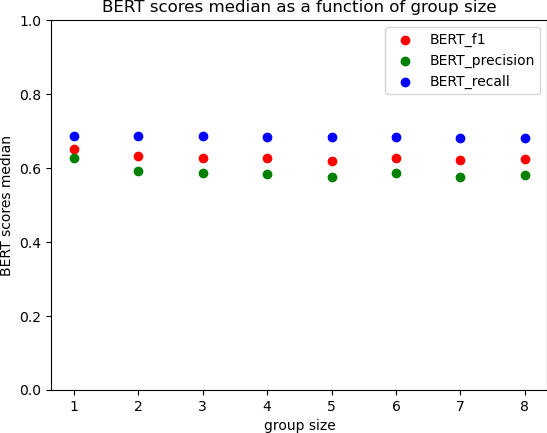
ממוצע:

|  |  |  |  |
| --- | --- | --- | --- |
| group\_size | BERT\_f1 | BERT\_precision | BERT\_recall |
| 1 | 0.678 | 0.64 | 0.722 |
| 2 | 0.662 | 0.612 | 0.721 |
| 3 | 0.655 | 0.603 | 0.72 |
| 4 | 0.657 | 0.607 | 0.718 |
| 5 | 0.651 | 0.596 | 0.719 |
| 6 | 0.658 | 0.609 | 0.718 |
| 7 | 0.654 | 0.602 | 0.718 |
| 8 | 0.656 | 0.605 | 0.718 |

:

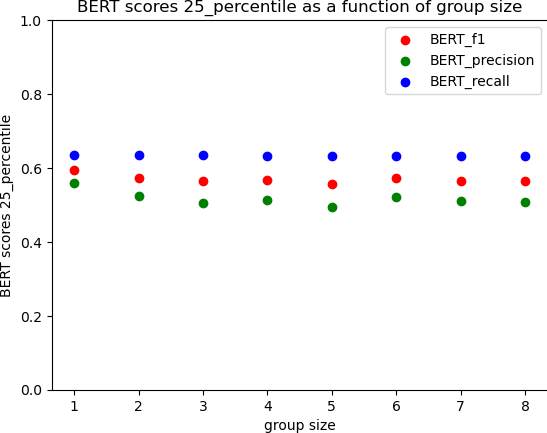
חציון:

|  |  |  |  |
| --- | --- | --- | --- |
| group\_size | BERT\_f1 | BERT\_precision | BERT\_recall |
| 1 | 0.652 | 0.627 | 0.687 |
| 2 | 0.633 | 0.591 | 0.687 |
| 3 | 0.627 | 0.587 | 0.685 |
| 4 | 0.628 | 0.583 | 0.683 |
| 5 | 0.62 | 0.576 | 0.683 |
| 6 | 0.628 | 0.586 | 0.683 |
| 7 | 0.621 | 0.575 | 0.682 |
| 8 | 0.624 | 0.581 | 0.682 |



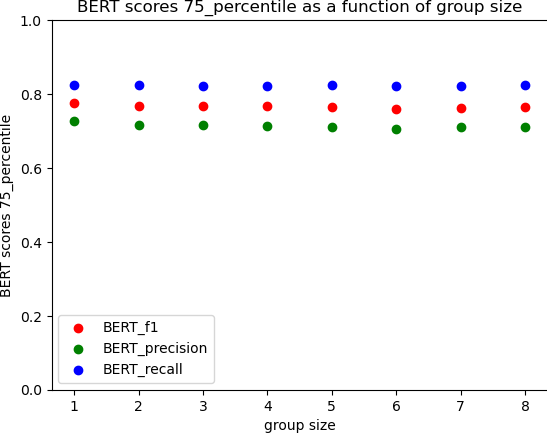
אחוזון 25:

|  |  |  |  |
| --- | --- | --- | --- |
| group\_size | BERT\_f1 | BERT\_precision | BERT\_recall |
| 1 | 0.593 | 0.559 | 0.634 |
| 2 | 0.573 | 0.524 | 0.635 |
| 3 | 0.566 | 0.506 | 0.635 |
| 4 | 0.567 | 0.514 | 0.633 |
| 5 | 0.558 | 0.494 | 0.633 |
| 6 | 0.572 | 0.521 | 0.634 |
| 7 | 0.565 | 0.51 | 0.633 |
| 8 | 0.564 | 0.509 | 0.633 |



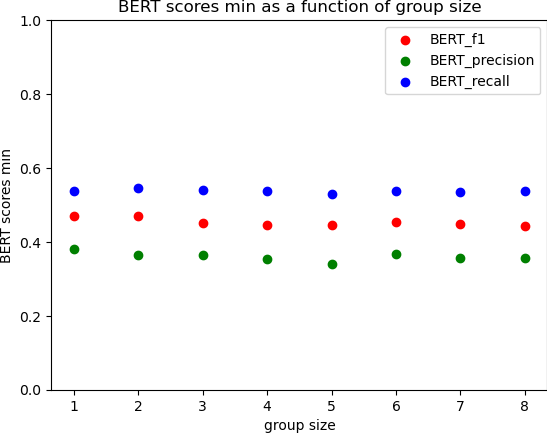
אחוזון 75:

|  |  |  |  |
| --- | --- | --- | --- |
| group\_size | BERT\_f1 | BERT\_precision | BERT\_recall |
| 1 | 0.774 | 0.728 | 0.824 |
| 2 | 0.767 | 0.715 | 0.824 |
| 3 | 0.767 | 0.717 | 0.822 |
| 4 | 0.767 | 0.715 | 0.822 |
| 5 | 0.764 | 0.712 | 0.824 |
| 6 | 0.76 | 0.706 | 0.821 |
| 7 | 0.762 | 0.71 | 0.822 |
| 8 | 0.764 | 0.71 | 0.824 |



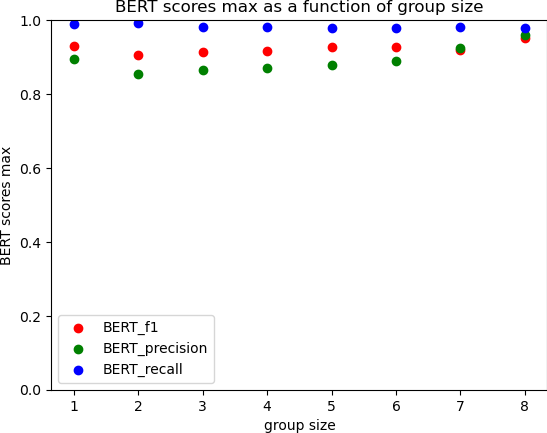
מינימום:

|  |  |  |  |
| --- | --- | --- | --- |
| group\_size | BERT\_f1 | BERT\_precision | BERT\_recall |
| 1 | 0.469 | 0.382 | 0.539 |
| 2 | 0.469 | 0.365 | 0.546 |
| 3 | 0.452 | 0.366 | 0.539 |
| 4 | 0.446 | 0.354 | 0.537 |
| 5 | 0.446 | 0.341 | 0.529 |
| 6 | 0.455 | 0.369 | 0.538 |
| 7 | 0.449 | 0.356 | 0.535 |
| 8 | 0.444 | 0.357 | 0.539 |



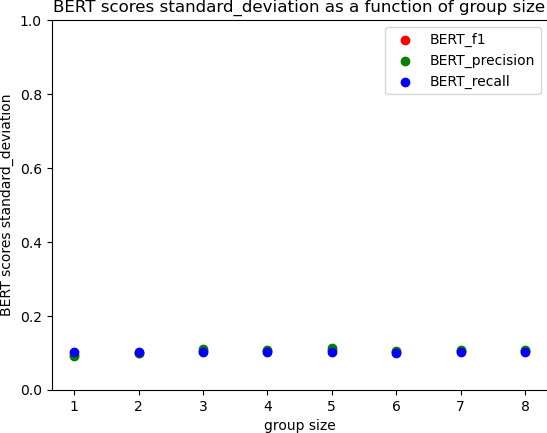
מקסימום:

|  |  |  |  |
| --- | --- | --- | --- |
| group\_size | BERT\_f1 | BERT\_precision | BERT\_recall |
| 1 | 0.93 | 0.896 | 0.989 |
| 2 | 0.905 | 0.854 | 0.992 |
| 3 | 0.914 | 0.864 | 0.98 |
| 4 | 0.916 | 0.871 | 0.982 |
| 5 | 0.927 | 0.879 | 0.979 |
| 6 | 0.927 | 0.89 | 0.978 |
| 7 | 0.92 | 0.924 | 0.982 |
| 8 | 0.951 | 0.959 | 0.978 |



סטיית תקן:

|  |  |  |  |
| --- | --- | --- | --- |
| group\_size | BERT\_f1 | BERT\_precision | BERT\_recall |
| 1 | 0.095 | 0.091 | 0.103 |
| 2 | 0.1 | 0.1 | 0.102 |
| 3 | 0.105 | 0.109 | 0.101 |
| 4 | 0.104 | 0.107 | 0.101 |
| 5 | 0.108 | 0.114 | 0.102 |
| 6 | 0.101 | 0.104 | 0.1 |
| 7 | 0.105 | 0.108 | 0.101 |
| 8 | 0.105 | 0.109 | 0.101 |



ראיתי שעדיין קיים פעם בין ה recall לבין ה precision אך הוא הצטמצם והחלטתי שהמגבלה ששמתי לאורך הטקסטים היא מספיק משמעותית ואין צורך להקטין את הפרמטר answer\_length\_multiplier לפחות מ 1.25.

תוצאות החלק השלישי:[[2]](#footnote-2)

בלי קשר לתוצאות הניסויים, שיניתי את מימוש האלגוריתם ככה ש:

במקום שההסתברות של טוקן שכבר מופיע בטקסט תקבע מלאכותית ל 0 – השתמשתי בשיטת דגימה עם עונשים ובחרתי בפרמטר θ=1.2 משום שזה הערך שהומלץ במאמר שהציג את השיטה.

בנוסף, מכיוון שהראיתי בחלקים הקודמים שהגדלת גודל הקבוצה מאחד לגדלים קטנים יחסית שתיים עד שמונה) אינה פוגעת באיכות הטקסטים אני רוצה לבדוק מה קורה כאשר מגדילים את גודל הקבוצה מאוד. לכן בניתי ניסוי בו הגדלתי את גודל הקבוצה מאחד () לשתיים () ואז לארבע () וככה הלאה עד אלפיים ארבעים ושמונה ().

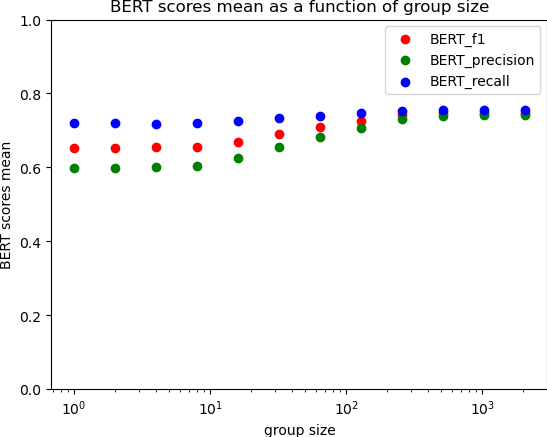
הפסקתי ב 2048 מכיוון שהניסוי עם גודל קבוצה של 4096 הגיע לשגיאת זיכרון – כלומר לא היה לי מספיק זיכרון במעבד הגרפי על מנת לבצע את הניסוי.

זאת משום שאני מכניס למודל שגודלו: 1 - אורך הפרומפט + גודל הקבוצה

וצריכת הזיכרון במעבד הגרפי גדלה עם גודל הקלט למודל.

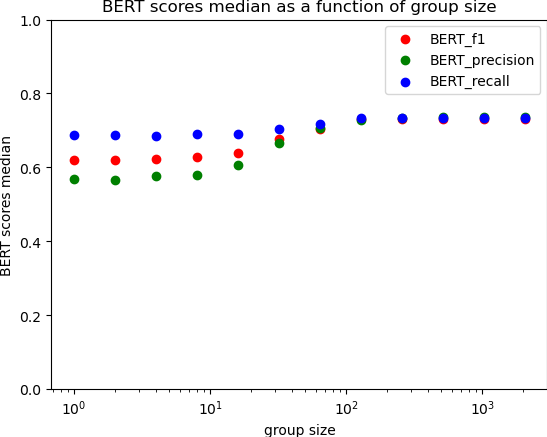
ממוצע:

|  |  |  |  |
| --- | --- | --- | --- |
| group\_size | BERT\_f1 | BERT\_precision | BERT\_recall |
| 1 | 0.653 | 0.599 | 0.72 |
| 2 | 0.652 | 0.598 | 0.719 |
| 4 | 0.654 | 0.602 | 0.718 |
| 8 | 0.656 | 0.603 | 0.72 |
| 16 | 0.67 | 0.626 | 0.724 |
| 32 | 0.689 | 0.654 | 0.732 |
| 64 | 0.708 | 0.682 | 0.74 |
| 128 | 0.725 | 0.707 | 0.747 |
| 256 | 0.741 | 0.731 | 0.754 |
| 512 | 0.746 | 0.739 | 0.755 |
| 1024 | 0.747 | 0.741 | 0.756 |
| 2048 | 0.747 | 0.741 | 0.756 |



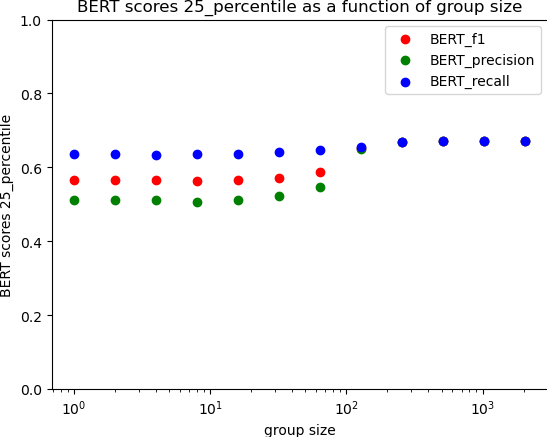
חציון:

|  |  |  |  |
| --- | --- | --- | --- |
| group\_size | BERT\_f1 | BERT\_precision | BERT\_recall |
| 1 | 0.621 | 0.567 | 0.688 |
| 2 | 0.619 | 0.566 | 0.687 |
| 4 | 0.624 | 0.577 | 0.685 |
| 8 | 0.629 | 0.581 | 0.69 |
| 16 | 0.64 | 0.606 | 0.691 |
| 32 | 0.677 | 0.665 | 0.704 |
| 64 | 0.704 | 0.707 | 0.718 |
| 128 | 0.731 | 0.729 | 0.733 |
| 256 | 0.732 | 0.734 | 0.734 |
| 512 | 0.732 | 0.735 | 0.734 |
| 1024 | 0.732 | 0.735 | 0.734 |
| 2048 | 0.732 | 0.735 | 0.734 |



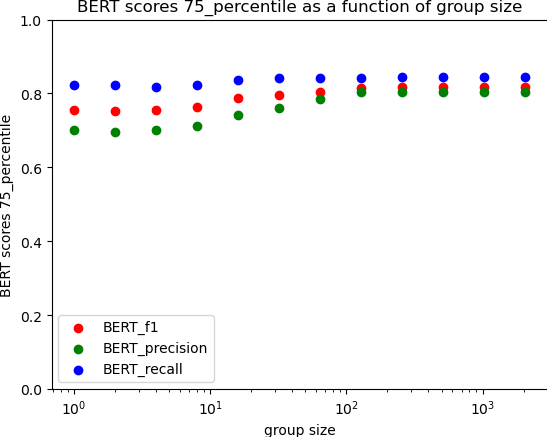
אחוזון 25:

|  |  |  |  |
| --- | --- | --- | --- |
| group\_size | BERT\_f1 | BERT\_precision | BERT\_recall |
| 1 | 0.566 | 0.512 | 0.635 |
| 2 | 0.565 | 0.51 | 0.635 |
| 4 | 0.565 | 0.511 | 0.634 |
| 8 | 0.564 | 0.508 | 0.637 |
| 16 | 0.565 | 0.512 | 0.637 |
| 32 | 0.572 | 0.522 | 0.641 |
| 64 | 0.588 | 0.546 | 0.646 |
| 128 | 0.651 | 0.65 | 0.655 |
| 256 | 0.669 | 0.669 | 0.668 |
| 512 | 0.671 | 0.67 | 0.671 |
| 1024 | 0.672 | 0.671 | 0.672 |
| 2048 | 0.672 | 0.671 | 0.672 |



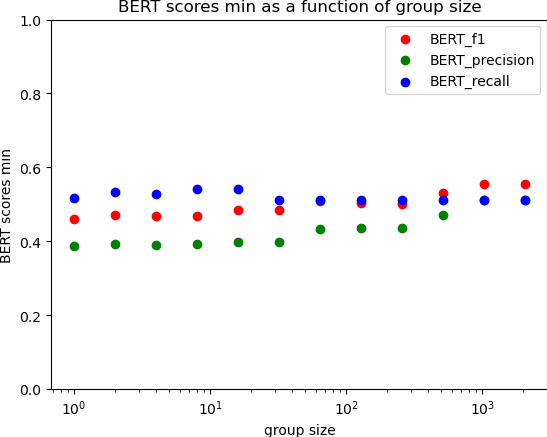
אחוזון 75:

|  |  |  |  |
| --- | --- | --- | --- |
| group\_size | BERT\_f1 | BERT\_precision | BERT\_recall |
| 1 | 0.755 | 0.7 | 0.822 |
| 2 | 0.753 | 0.695 | 0.823 |
| 4 | 0.755 | 0.702 | 0.818 |
| 8 | 0.763 | 0.711 | 0.822 |
| 16 | 0.787 | 0.741 | 0.836 |
| 32 | 0.797 | 0.76 | 0.841 |
| 64 | 0.805 | 0.784 | 0.842 |
| 128 | 0.813 | 0.803 | 0.842 |
| 256 | 0.817 | 0.803 | 0.845 |
| 512 | 0.817 | 0.803 | 0.845 |
| 1024 | 0.817 | 0.803 | 0.845 |
| 2048 | 0.817 | 0.803 | 0.845 |



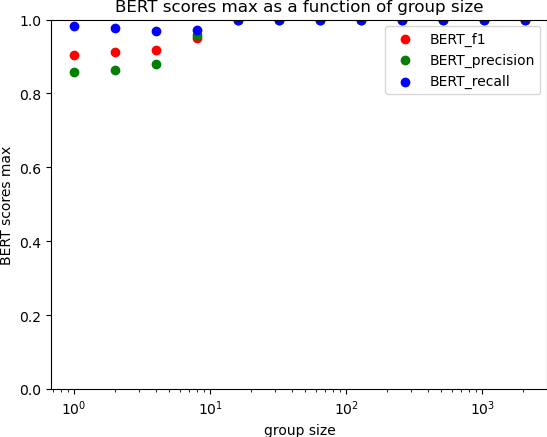
מינימום:

|  |  |  |  |
| --- | --- | --- | --- |
| group\_size | BERT\_f1 | BERT\_precision | BERT\_recall |
| 1 | 0.46 | 0.388 | 0.518 |
| 2 | 0.472 | 0.394 | 0.534 |
| 4 | 0.469 | 0.391 | 0.529 |
| 8 | 0.469 | 0.393 | 0.542 |
| 16 | 0.484 | 0.398 | 0.541 |
| 32 | 0.484 | 0.399 | 0.511 |
| 64 | 0.508 | 0.432 | 0.511 |
| 128 | 0.502 | 0.437 | 0.511 |
| 256 | 0.502 | 0.436 | 0.511 |
| 512 | 0.529 | 0.471 | 0.511 |
| 1024 | 0.554 | 0.511 | 0.511 |
| 2048 | 0.554 | 0.511 | 0.511 |



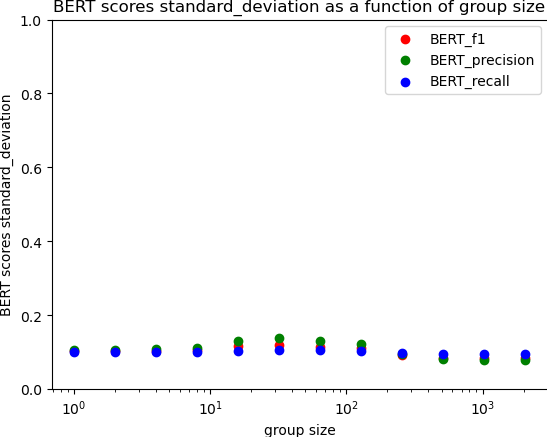
מקסימום:

|  |  |  |  |
| --- | --- | --- | --- |
| group\_size | BERT\_f1 | BERT\_precision | BERT\_recall |
| 1 | 0.905 | 0.859 | 0.983 |
| 2 | 0.912 | 0.863 | 0.977 |
| 4 | 0.918 | 0.88 | 0.969 |
| 8 | 0.951 | 0.959 | 0.971 |
| 16 | 1.0 | 1.0 | 1.0 |
| 32 | 1.0 | 1.0 | 1.0 |
| 64 | 1.0 | 1.0 | 1.0 |
| 128 | 1.0 | 1.0 | 1.0 |
| 256 | 1.0 | 1.0 | 1.0 |
| 512 | 1.0 | 1.0 | 1.0 |
| 1024 | 1.0 | 1.0 | 1.0 |
| 2048 | 1.0 | 1.0 | 1.0 |



סטיית תקן:

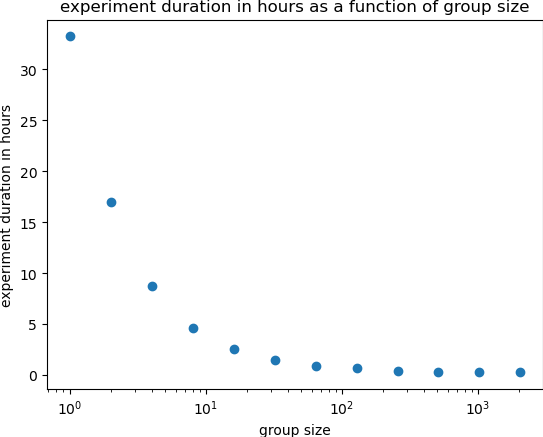
|  |  |  |  |
| --- | --- | --- | --- |
| group\_size | BERT\_f1 | BERT\_precision | BERT\_recall |
| 1 | 0.102 | 0.105 | 0.101 |
| 2 | 0.103 | 0.106 | 0.101 |
| 4 | 0.104 | 0.107 | 0.101 |
| 8 | 0.105 | 0.11 | 0.1 |
| 16 | 0.116 | 0.129 | 0.103 |
| 32 | 0.121 | 0.137 | 0.107 |
| 64 | 0.115 | 0.129 | 0.105 |
| 128 | 0.11 | 0.122 | 0.103 |
| 256 | 0.093 | 0.096 | 0.097 |
| 512 | 0.085 | 0.082 | 0.095 |
| 1024 | 0.084 | 0.079 | 0.095 |
| 2048 | 0.084 | 0.079 | 0.095 |



* כאשר באחת העמודות בגרפים מופיעה רק נקודה כחולה, הנקודה הכחולה והנקודה הירוקה נמצאות בדיוק באותו מקום כמו הנקודה הכחולה

ניתן לראות שככל שגודל הקבוצה גדל – הממוצע, החציון ואחוזונים 25 ו75 גדלים משמעותית.

זמן הריצה הכולל של הניסויים כפונקציה של גודל הקבוצה:



זמן הריצה של כל ניסוי מורכב משני מרכיבים עיקריים: חישוב מדד ברט ויצירת הטקסטים וממרכיבים שוליים כגון טעינת הנתונים, ושליחת הנתונים לאפליקציית ווב שהשתמשתי בה על מנת לעקוב אחר תוצאות הניסויים.

זמן הריצה של ממד ברט קבוע כי אורכי הטקסטים שהאלגוריתם יוצר לא אמורים להשתנות באופן משמעותי.

ראיתי שתוצאות הגדלת גודל הקבוצה מ 1024 ל2048 לא משפיעה על המדדים בכלל ומשפיעה לרעה על זמן הריצה.

בדיקה שערכתי הראתה שאורכי הטקסטים בסט הנתונים בו השתמשתי הם בין 891 ל6 טוקנים. קבעתי שהאלגוריתם לא יוכל ליצור טקסטים ארוכים ביותר מ25% מהטקסטים שהוא מקבל ולכן בניסויים לא נוצר טקסט שארוך יותר מ 1114 טוקנים. לכן אין צורך להגדיל את גודל הקבוצה מעבר ל 1114. בנוסף לכך, כשאנחנו מגדילים את גודל הקבוצה אנחנו מגדילים את הרצפים שנכנסים למודל ובכך מגדילים את זמן הריצה של כל שימוש במודל.

הצגת התוצרים

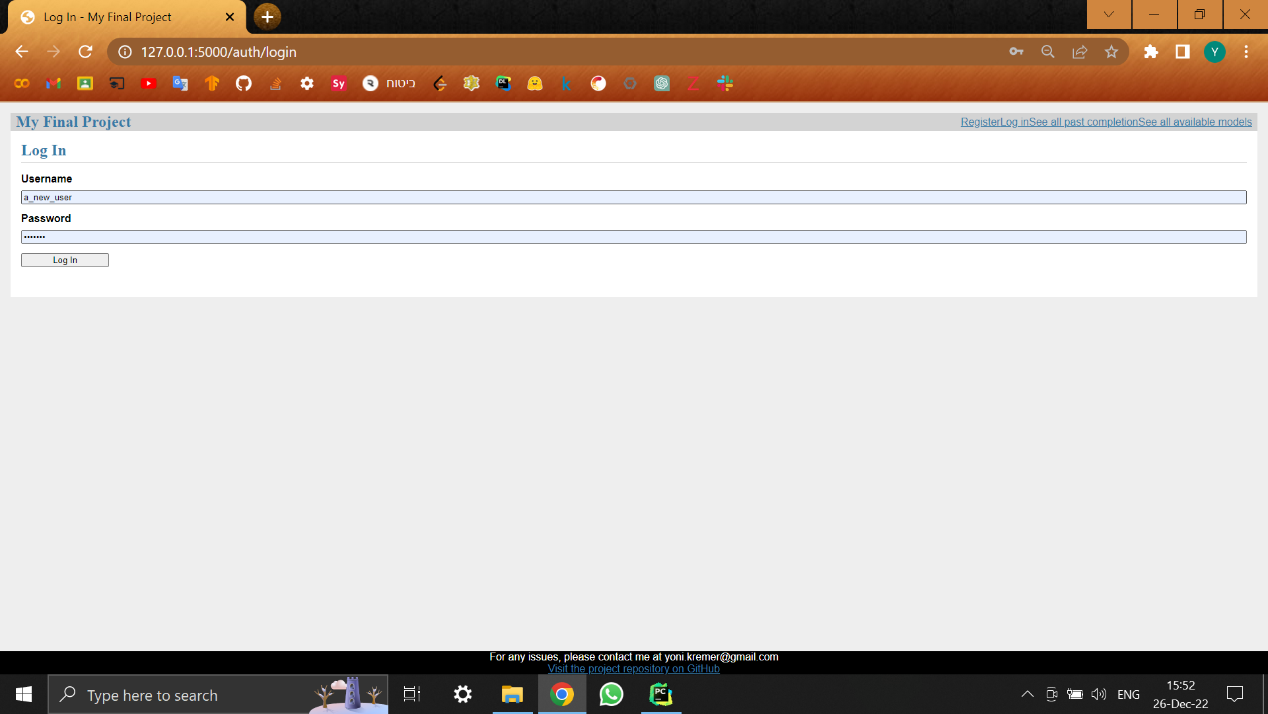
אפליקציית הווב:

עמוד ההרשמה:

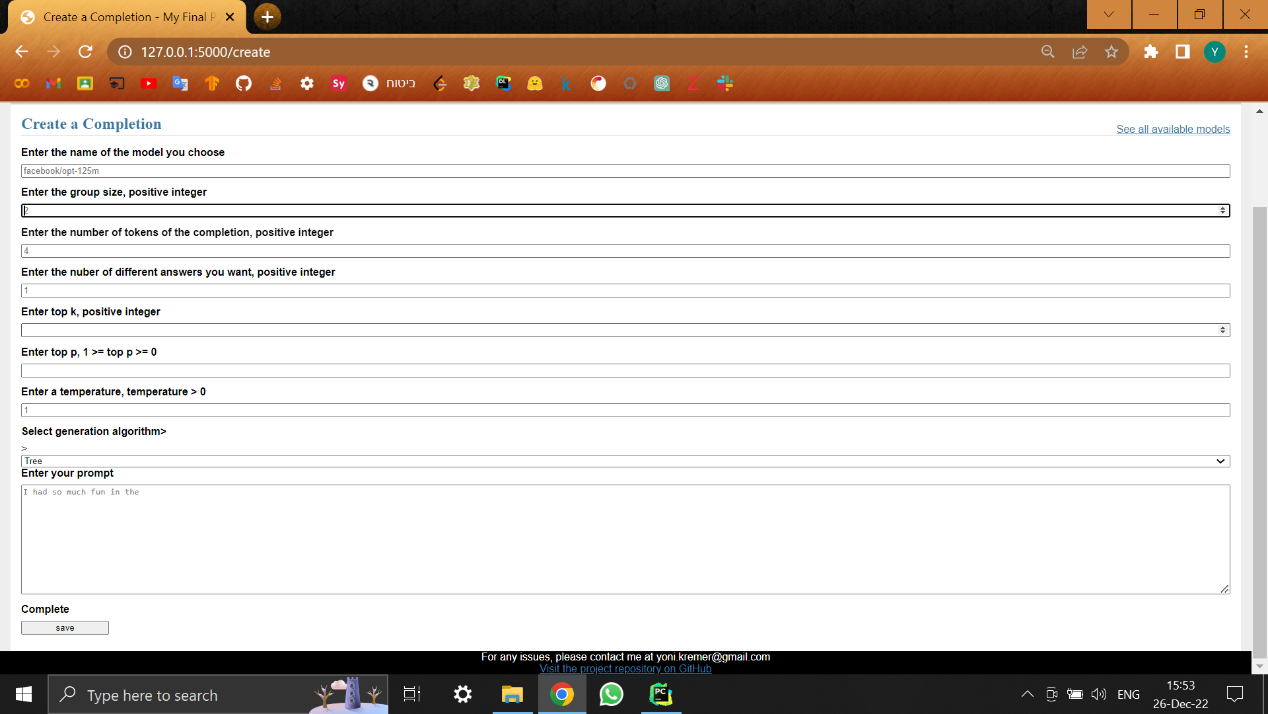
תמונה שמכילה טקסט, צילום מסך, צג, מקורה

התיאור נוצר באופן אוטומטי

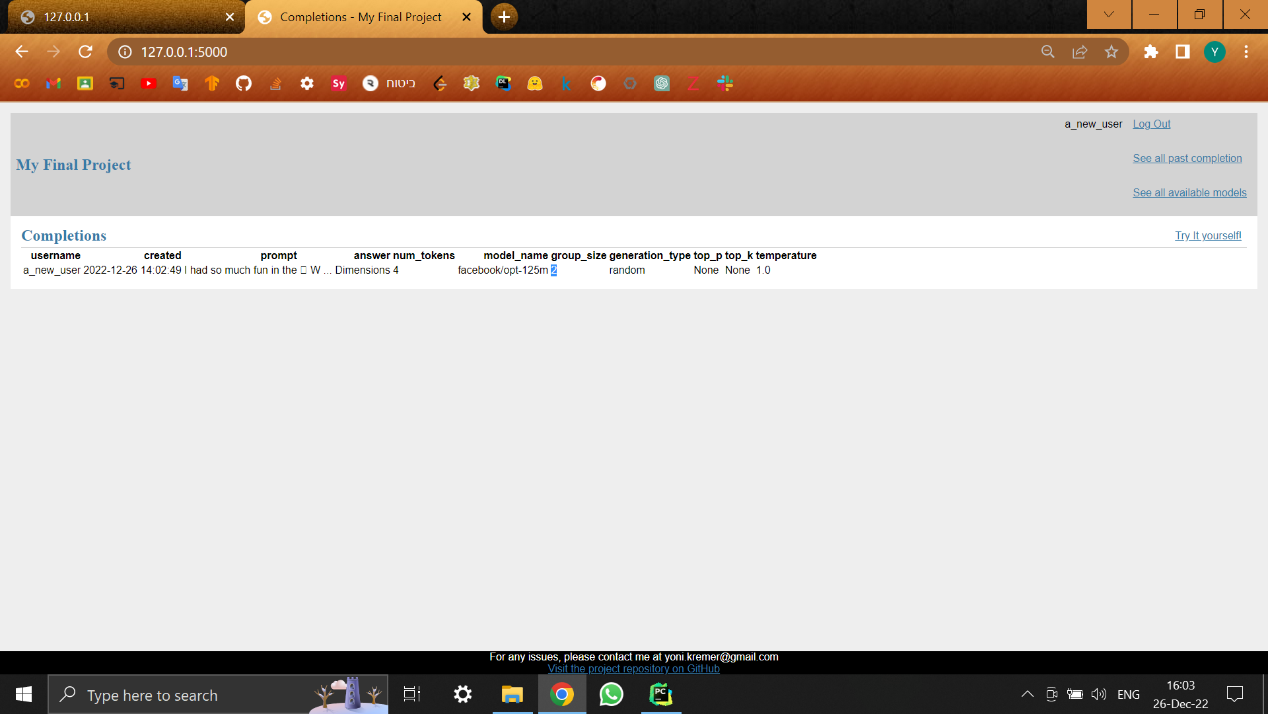
עמוד ההתחברות



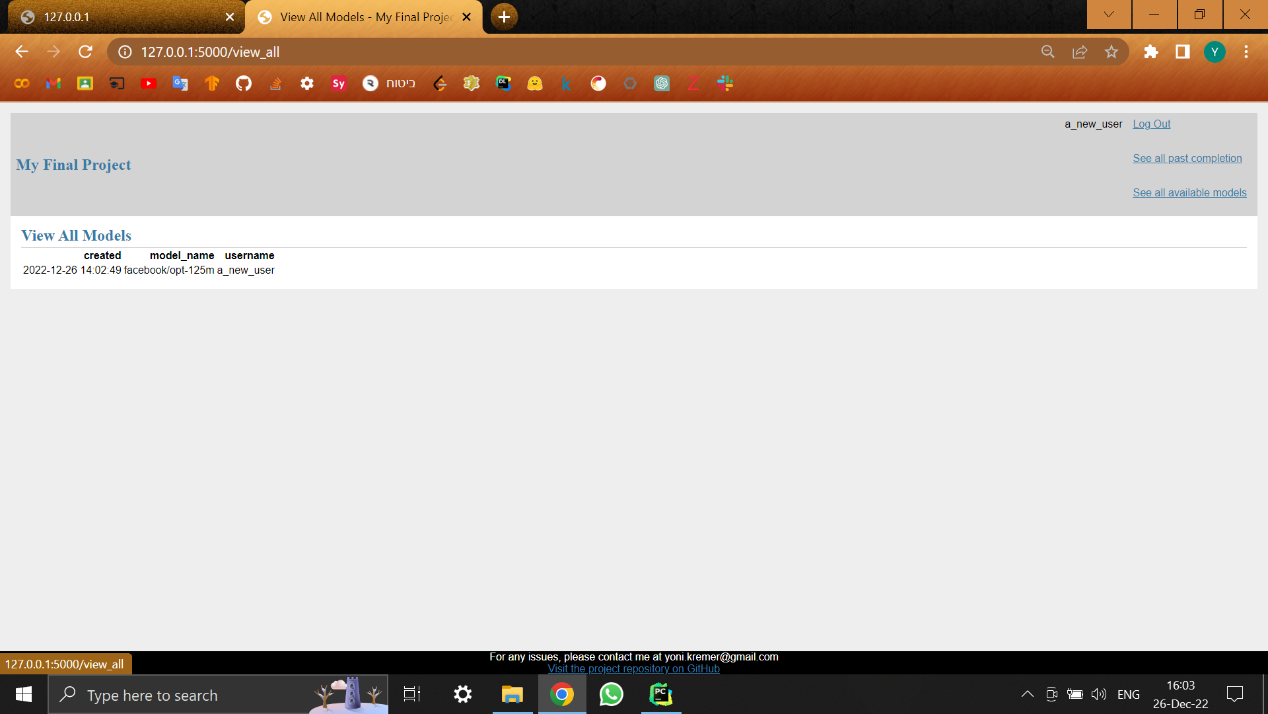
עמוד השימוש באלגוריתם:



עמוד הצפייה בטקסטים שהמודל יוצר:



עמוד הצפייה בכל המודלים השמורים בזיכרון של השרת:



סיכום

בעבודה זו, הצלחתי לפתח אלגוריתם היוצר טקסט באורך n טוקנים בעזרת פחות מ n שימושים במודל שפה סיבתי. האלגוריתם עובד ללא בעיות ועובר את כל הבדיקות בהצלחה.

האלגוריתם שלי יעיל משמעותית מכל אלגוריתם קיים הן מבחינת סיבוכיות:

סיבוכיות זמן הריצה של האלגוריתם שפיתחתי היא .

כאשר גודל הקבוצה שווה למספר הטוקנים שהאלגוריתם יוצר, כאשר n מייצג את כמות הטוקנים שהאלגוריתם יוצר ו m את כמות הטוקנים שהוא מקבל.

סיבוכיות זמן הריצה של האלגוריתמים היעילים ביותר הקיימים כיום הוא .

הראיתי גם שהאלגוריתם שפיתחתי מצליח יותר בתרגום טקסטים ארוכים כאשר מדדתי את מדדי ברט (BERT Score).

המחקר שלי מאפשר הורדה משמעותית מאוד של עלויות המחשוב וכן שיפור באיכות הטקסטים הנוצרים. לכל גוף שמשתמש במודלים אלו על מנת ליצור טקסט.

המשך המחקר:

אני מתכנן להמשיך לעבוד על האלגוריתם גם לאחר הגשת העבודה.

השינויים שאני מתכנן הם:

* הוספת האפשרות לקיבוץ (batching) – העברת מספר דוגמאות למודל במקביל.
* שימוש בגודל קבוצה שאינו קבוע.

ביבליוגרפיה

1. ‘3. Data Model’. Python Documentation, The Python Software Foundation, https://docs.python.org/3/reference/datamodel.html. Accessed 24 Dec. 2022.
2. Ackley, David H., et al. ‘A Learning Algorithm for Boltzmann Machines\*’. Cognitive Science, vol. 9, no. 1, Jan. 1985, pp. 147–69. DOI.org (Crossref), https://doi.org/10.1207/s15516709cog0901\_7.
3. Agarap, Abien Fred. Deep Learning Using Rectified Linear Units (ReLU). arXiv, 7 Feb. 2019. arXiv.org, https://doi.org/10.48550/arXiv.1803.08375.
4. Bridle, John. ‘Training Stochastic Model Recognition Algorithms as Networks Can Lead to Maximum Mutual Information Estimation of Parameters’. Advances in Neural Information Processing Systems, vol. 2, Morgan-Kaufmann, 1989. Neural Information Processing Systems, https://proceedings.neurips.cc/paper/1989/hash/0336dcbab05b9d5ad24f4333c7658a0e-Abstract.html.
5. ‘Collections.Abc — Abstract Base Classes for Containers’. Python Documentation, The Python Software Foundation, https://docs.python.org/3/library/collections.abc.html. Accessed 24 Dec. 2022.
6. Eric V. Smith. ‘PEP 557 – Data Classes’. Python Enhancement Proposals (PEPs), https://peps.python.org/pep-0557/. Accessed 24 Dec. 2022.
7. Fan, Angela, et al. ‘Hierarchical Neural Story Generation’. Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), Association for Computational Linguistics, 2018, pp. 889–98. DOI.org (Crossref), https://doi.org/10.18653/v1/P18-1082.
8. Gehring, Jonas, et al. ‘Convolutional Sequence to Sequence Learning’. Proceedings of the 34th International Conference on Machine Learning, PMLR, 2017, pp. 1243–52. proceedings.mlr.press, https://proceedings.mlr.press/v70/gehring17a.html.
9. Guido van Rossum and Talin. ‘PEP 3119 – Introducing Abstract Base Classes’. Python Enhancement Proposals (PEPs), https://peps.python.org/pep-3119/. Accessed 24 Dec. 2022.
10. ‘Heapq — Heap Queue Algorithm’. Python Documentation, The Python Software Foundation, https://docs.python.org/3/library/heapq.html. Accessed 24 Dec. 2022.
11. Holtzman, Ari, et al. The Curious Case of Neural Text Degeneration. arXiv, 14 Feb. 2020. arXiv.org, http://arxiv.org/abs/1904.09751.
12. Keskar, Nitish Shirish, et al. CTRL: A Conditional Transformer Language Model for Controllable Generation. arXiv, 20 Sept. 2019. arXiv.org, https://doi.org/10.48550/arXiv.1909.05858.
13. Kevin D. Smith, et al. ‘PEP 318 – Decorators for Functions and Methods’. Python Enhancement Proposals (PEPs), 2 Sept. 2004, https://peps.python.org/pep-0318/.
14. Liu, Peter J., et al. Generating Wikipedia by Summarizing Long Sequences. arXiv, 30 Jan. 2018. arXiv.org, https://doi.org/10.48550/arXiv.1801.10198.
15. Mauro Cettolo, et al. WIT3 : Web Inventory of Transcribed and Translated Talks. 2012, https://aclanthology.org/2012.eamt-1.60.pdf.
16. Mikolov, Tomas, et al. Efficient Estimation of Word Representations in Vector Space. arXiv, 6 Sept. 2013. arXiv.org, https://doi.org/10.48550/arXiv.1301.3781.
17. nostalgebraist. Interpreting GPT: The Logit Lens. www.lesswrong.com, https://www.lesswrong.com/posts/AcKRB8wDpdaN6v6ru/interpreting-gpt-the-logit-lens. Accessed 29 Dec. 2022.
18. Papers with Code - Improving Language Understanding by Generative Pre-Training. https://paperswithcode.com/paper/improving-language-understanding-by. Accessed 5 Jan. 2023..
19. ---. https://paperswithcode.com/method/gpt. Accessed 23 Dec. 2022.
20. Press, Ofir, and Lior Wolf. Using the Output Embedding to Improve Language Models. arXiv, 21 Feb. 2017. arXiv.org, https://doi.org/10.48550/arXiv.1608.05859.
21. Rossum, Guido van. ‘Unifying Types and Classes in Python 2.2’. Python, https://www.python.org/download/releases/2.2.3/descrintro/. Accessed 24 Dec. 2022.
22. Suits, Daniel B. ‘Use of Dummy Variables in Regression Equations’. Journal of the American Statistical Association, vol. 52, no. 280, Dec. 1957, pp. 548–51. DOI.org (Crossref), https://doi.org/10.1080/01621459.1957.10501412.
23. van Rossum, Guido, et al. ‘PEP 484 – Type Hints’. Python Enhancement Proposals (PEPs), https://peps.python.org/pep-0484/. Accessed 24 Dec. 2022.
24. Vaswani, Ashish, et al. Attention Is All You Need. arXiv, 5 Dec. 2017. arXiv.org, https://doi.org/10.48550/arXiv.1706.03762.
25. Warsaw, Barry, et al. ‘PEP 435 – Adding an Enum Type to the Python Standard Library’. Python Enhancement Proposals (PEPs), https://peps.python.org/pep-0435/. Accessed 24 Dec. 2022.
26. Zhang, Susan, et al. OPT: Open Pre-Trained Transformer Language Models. arXiv, 21 June 2022. arXiv.org, https://doi.org/10.48550/arXiv.2205.01068.
27. Zhang, Tianyi, et al. BERTScore: Evaluating Text Generation with BERT. arXiv, 24 Feb. 2020. arXiv.org, https://doi.org/10.48550/arXiv.1904.09675..

נספחים

קישור לעמוד הגיטהאב של הפרויקט:

<https://github.com/yonikremer/grouped_sampling>

קוד האלגוריתם לדגימה בקבוצות:

preprocessor.py

from typing import Tuple  
  
from torch import LongTensor, cat  
from transformers import PreTrainedTokenizer  
from transformers.tokenization\_utils\_base import TruncationStrategy, BatchEncoding  
  
  
class PreProcessor:  
 framework: str = "pt"  
  
 def \_\_init\_\_(  
 self,  
 tokenizer: PreTrainedTokenizer,  
 max\_input\_len: int,  
 ):  
 self.tokenizer: PreTrainedTokenizer = tokenizer  
 self.max\_input\_len: int = max\_input\_len  
  
 def get\_token\_tensor(  
 self,  
 text: str,  
 truncation: TruncationStrategy = TruncationStrategy.DO\_NOT\_TRUNCATE,  
 ) -> LongTensor:  
 *"""Complexity: O(n) where n is the number of characters in the text"""* if len(text) == 0:  
 return LongTensor([])  
 # tokenizing a string is O(n) where n is the length of the string  
 tokenized\_text = self.tokenizer(  
 text,  
 return\_tensors=self.framework,  
 padding=False,  
 add\_special\_tokens=False,  
 truncation=truncation,  
 max\_length=self.max\_input\_len,  
 )  
 if isinstance(tokenized\_text, dict) or isinstance(tokenized\_text, BatchEncoding):  
 token\_tensor: LongTensor = tokenized\_text["input\_ids"]  
 # O(1) because we are accessing a single element  
 # in a dictionary and saving the reference to it.  
 elif isinstance(tokenized\_text, LongTensor):  
 token\_tensor: LongTensor = tokenized\_text  
 # O(1) because we are saving the reference to the tensor  
 else:  
 raise TypeError(  
 "The tokenizer output is not one of:"  
 "dict, BatchEncoding, LongTensor"  
 )  
 token\_tensor = token\_tensor.squeeze()  
 # O(n) where n is the number of tokens in the text  
 # because we are copying n elements from one tensor to the other  
 # the number of tokens in the text  
 # is always less than the number of characters in the text  
  
 return token\_tensor  
  
 def \_\_call\_\_(  
 self,  
 prompt: str,  
 prefix: str = "",  
 truncation: TruncationStrategy = TruncationStrategy.DO\_NOT\_TRUNCATE,  
 postfix: str = "",  
 ) -> Tuple[LongTensor, int, int, int]:  
 *"""A helper method for \_\_call\_\_ that tokenize the prompt  
 all the arguments are sent directly from the \_\_call\_\_ method  
 Returns:  
 the tokenized prompt as a list of ints,  
 the length of the prompt,  
 the length of the prefix,  
 the length of the postfix  
  
 Complexity: O(a + b + c) where:  
 'a' is the number of characters in the prefix  
 'b' is the number of characters in the prompt  
 'c' is the number of characters in the postfix"""* prefix\_tokens: LongTensor = self.get\_token\_tensor(prefix, truncation)  
 # O(A) where A is the number of characters in the prefix.  
 postfix\_tokens: LongTensor = self.get\_token\_tensor(postfix, truncation)  
 # O(B) where B is the number of characters in the postfix.  
 prompt\_tokens: LongTensor = self.get\_token\_tensor(prompt, truncation)  
 # O(C) where C is the number of characters in the prompt.  
 token\_tensor: LongTensor = cat((prefix\_tokens, prompt\_tokens, postfix\_tokens))  
 # O( + b + c) where 'a' is the number of tokens in the prefix.  
 # 'b' is the number of tokens in the prompt.  
 # 'c' is the number of tokens in the postfix.  
 # we know that the number of tokens is less than the number of characters  
 return token\_tensor, len(prefix\_tokens), len(prompt\_tokens), len(postfix\_tokens)

postprocessor.py

from typing import Optional  
  
from torch import tensor  
from transformers import PreTrainedTokenizer  
  
from src.grouped\_sampling import TokenIDS  
  
  
class PostProcessor:  
 def \_\_init\_\_(  
 self,  
 tokenizer: PreTrainedTokenizer,  
 ):  
 self.tokenizer: PreTrainedTokenizer = tokenizer  
  
 def \_\_call\_\_(  
 self,  
 token\_ids: TokenIDS,  
 num\_new\_tokens: Optional[int],  
 prompt\_len: int,  
 return\_text: bool,  
 return\_tensors: bool,  
 return\_full\_text: bool,  
 clean\_up\_tokenization\_spaces: bool,  
 prefix\_len: int = 0,  
 postfix\_len: int = 0,  
 ):  
 *"""A helper method for \_\_call\_\_  
 that converts the token ids to dictionary  
 token\_ids - the token ids from the \_forward method  
 prompt\_len - the length of the tokenized prompt  
 the rest of the arguments are the arguments  
 from the \_\_call\_\_ method  
 look up the documentation of the \_\_call\_\_ method for more info  
 Complexity: O(n) where n is the number of tokens in token\_ids  
 """* # define n as the length of the token\_ids  
 full\_prompt\_len = prefix\_len + prompt\_len + postfix\_len  
 if num\_new\_tokens is None:  
 shorten\_token\_list = token\_ids  
 # O(1)  
 else:  
 final\_num\_tokens = full\_prompt\_len + num\_new\_tokens  
 # O(1)  
 if len(token\_ids) > final\_num\_tokens:  
 shorten\_token\_list = token\_ids[:final\_num\_tokens]  
 # O(final\_num\_tokens)  
 # because we are copying final\_num\_tokens  
 # elements from one list to the other  
 else:  
 shorten\_token\_list = token\_ids  
 # O(1)  
  
 generated\_tokens = shorten\_token\_list[full\_prompt\_len:]  
 # O(num\_new\_tokens) because we are copying  
 # num\_new\_tokens elements from one list to the other  
 if return\_full\_text:  
 prompt\_tokens = shorten\_token\_list[prefix\_len:prefix\_len + prompt\_len]  
 # Without prefix and postfix  
 # O(prompt\_len) because we are copying prompt\_len  
 # elements from one list to the other  
 final\_token\_list = prompt\_tokens + generated\_tokens  
 # O(prompt\_len + num\_new\_tokens)  
 # because we are copying elements from one list to the other  
 else:  
 final\_token\_list = generated\_tokens  
 # O(1) because we are saving the reference to the list  
 final\_ans = {}  
 if return\_tensors:  
 final\_ans["generated\_token\_ids"] = tensor(final\_token\_list)  
 # O(prompt\_len + num\_new\_tokens)  
 # because we are copying at maximum  
 # prompt\_len + num\_new\_tokens elements from a list to a tensor  
 if return\_text:  
 final\_ans["generated\_text"] = self.tokenizer.decode(  
 final\_token\_list,  
 skip\_special\_tokens=True,  
 clean\_up\_tokenization\_spaces=clean\_up\_tokenization\_spaces  
 )  
 # decoding is O(m)  
 # where m is the number of tokens in the text  
 # so the complexity of this line is O(n)  
 # because final\_token\_list could be at most n tokens long  
 return final\_ans

repetition\_penalty.py

from abc import ABC, abstractmethod  
from typing import Set, Optional  
  
from torch import Tensor  
  
from .token\_ids import TokenIDS  
  
  
class RepetitionPenaltyStrategy(ABC):  
 theta: float  
  
 @staticmethod  
 def get\_already\_generated\_tokens(  
 tokens: TokenIDS,  
 generation\_start: int  
 ) -> Set[int]:  
 return set(tokens[generation\_start:])  
  
 @abstractmethod  
 def \_\_call\_\_(  
 self,  
 logits: Tensor,  
 tokens: TokenIDS,  
 generation\_start: int  
 ) -> Tensor:  
 raise NotImplementedError  
  
 def \_\_str\_\_(self) -> str:  
 return f"{self.\_\_class\_\_.\_\_name\_\_}(theta={self.theta})"  
  
 def \_\_repr\_\_(self) -> str:  
 return str(self)  
  
  
class LogitScalingRepetitionPenalty(  
 RepetitionPenaltyStrategy  
):  
 def \_\_init\_\_(self, theta: float):  
 if theta <= 1:  
 raise ValueError("Theta must be > 1")  
 self.theta = theta  
  
 def \_\_call\_\_(  
 self,  
 logits: Tensor,  
 tokens: TokenIDS,  
 generation\_start: int  
 ) -> Tensor:  
 *"""applies repetition penalty,  
 using the repetition\_penalty\_theta parameter defined in the class  
 the formula from the paper is:  
 softmax(original\_logits, T) =  
 exp(original\_logits\_i/(T · h(i))  
 / sum(exp(original\_logits\_i/(T · h(i)))  
 which is equivalent to:  
 pi = exp((xi/h(i)) / T / sum(exp(xj/(T · h(j)))  
 and to:  
 penalized\_logits = original\_logits / h(i)  
 pi = softmax(original\_logits / h(i), T)  
 where:  
 h(i) = θ if i in generated\_tokens else 1  
 T is the temperature parameter of softmax  
 Args:  
 logits: the logits matrix of shape (group\_size, vocab\_size)  
 tokens: the sequence of tokes from the prompt and the generated  
 generation\_start: The index of the first  
 Returns:  
 original\_logits / h(i)  
 complexity: O(group\_size \* n)  
 where n is the number of tokens generated by the algorithm  
 """* generated\_tokens = self.get\_already\_generated\_tokens(  
 tokens,  
 generation\_start  
 )  
 for token\_id in generated\_tokens:  
 # len(generated\_tokens) < max(n, vocab\_size)  
 logits[:, token\_id] /= self.theta  
 # O(group\_size) because we are taking a slice of size group\_size  
 return logits  
  
  
class NoRepetitionPenalty(  
 RepetitionPenaltyStrategy  
):  
 theta = 1.0  
  
 def \_\_call\_\_(  
 self,  
 logits: Tensor,  
 tokens: TokenIDS,  
 generation\_start: int  
 ) -> Tensor:  
 return logits  
  
  
DEFAULT\_REPETITION\_PENALTY = LogitScalingRepetitionPenalty(1.2)  
  
  
def repetition\_penalty\_factory(  
 theta: Optional[float]  
) -> RepetitionPenaltyStrategy:  
 if theta is None:  
 return DEFAULT\_REPETITION\_PENALTY  
 if theta == 1:  
 return NoRepetitionPenalty()  
 elif theta > 1:  
 return LogitScalingRepetitionPenalty(theta)  
 else:  
 raise ValueError("theta must be greater than 1")

generation\_type.py

from enum import Enum  
  
  
class GenerationType(Enum):  
 *"""The type of generation to use"""* GREEDY = "greedy"  
 TOP\_K = "top\_k"  
 TOP\_P = "top\_p"  
 TREE = "tree"  
 RANDOM = "random"  
  
 def requires\_softmax(self) -> bool:  
 *"""Whether the generation type requires a softmax"""* return self in (self.TOP\_K, self.TOP\_P, self.TREE, self.RANDOM)

generation\_utils.py

from typing import Dict  
from warnings import warn  
  
from torch import cuda, LongTensor, ones, long, Tensor, cat, no\_grad  
from transformers import AutoModelForCausalLM  
from torch.nn import Softmax  
  
from .token\_ids import TokenIDS  
from .repetition\_penalty import RepetitionPenaltyStrategy  
  
  
class GroupedGenerationUtils:  
 descriptive\_attrs = (  
 "group\_size",  
 "end\_of\_sentence\_stop",  
 "temp",  
 "repetition\_penalty\_strategy",  
 )  
  
 def \_\_init\_\_(  
 self,  
 model\_name: str,  
 group\_size: int,  
 max\_input\_len: int,  
 end\_of\_sentence\_id: int,  
 padding\_id: int,  
 vocab\_size: int,  
 repetition\_penalty\_strategy: RepetitionPenaltyStrategy,  
 end\_of\_sentence\_stop: bool = True,  
 temp: float = 1.0,  
 use\_softmax: bool = True,  
 \*\*kwargs  
 ):  
 *"""initializes the model wrapper  
 Args:  
 model\_name: the name of the model to be used  
 repetition\_penalty\_strategy:  
 the strategy to be used for the repetition penalty  
 group\_size: the number of next tokens to be generated  
 max\_input\_len: the maximum length of the input to the model  
 padding\_id: the id of the padding token  
 vocab\_size: the size of the vocabulary  
 end\_of\_sentence\_stop:  
 whether to stop when the end of sentence token is generated  
 end\_of\_sentence\_id:  
 the id of the end of sequence token  
 use\_softmax:  
 true if the model should use softmax,  
 false if it should return the logits  
 \*\*kwargs: the arguments to be passed to the model  
 Complexity: O(1)  
 """* self.use\_softmax: bool = use\_softmax  
 self.end\_of\_sentence\_id: int = end\_of\_sentence\_id  
 self.repetition\_penalty\_strategy: RepetitionPenaltyStrategy = repetition\_penalty\_strategy  
 self.group\_size: int = group\_size  
 self.max\_input\_len: int = max\_input\_len  
 self.padding\_id: int = padding\_id  
 self.end\_of\_sentence\_stop: bool = end\_of\_sentence\_stop  
 self.model = AutoModelForCausalLM.from\_pretrained(  
 model\_name, \*\*kwargs  
 )  
 self.temp: float = temp  
 self.vocab\_size: int = vocab\_size  
 if cuda.is\_available():  
 self.model = self.model.cuda()  
  
 @property  
 def padding\_tokens(self) -> LongTensor:  
 cpu\_tokens = ones(self.group\_size, dtype=long) \* self.padding\_id  
 if cuda.is\_available():  
 return cpu\_tokens.cuda()  
 return cpu\_tokens  
  
 def prepare\_model\_kwargs(  
 self, tokens: TokenIDS  
 ) -> Dict[str, LongTensor]:  
 *"""preparing the arguments for the model call  
 Args:  
 tokens: the tokens to be sent to the model  
 Returns:  
 a dictionary of the arguments for the model call  
 Complexity: O(group\_size + n) where n is the number of tokens  
 """* if not isinstance(tokens, Tensor):  
 tokens = LongTensor(tokens) # O(n)  
 padded\_tokens: LongTensor = cat(  
 (tokens, self.padding\_tokens), dim=0  
 ).unsqueeze(0)  
 # the length of padded\_tokens is n + group\_size - 1  
 # so creating it is O(n + group\_size)  
 attention\_len = padded\_tokens.shape[1] # n + group\_size - 1  
 if attention\_len > self.max\_input\_len:  
 padded\_tokens = padded\_tokens[:, -self.max\_input\_len:]  
 # O(self.max\_input\_len) which is constant so O(1)  
 attention\_len = self.max\_input\_len  
 attention\_mask: LongTensor = ones(  
 [1, attention\_len], dtype=long  
 )  
 # O(attention\_len) so O(n + group\_size)  
 if cuda.is\_available():  
 padded\_tokens = padded\_tokens.cuda() # O(n + group\_size)  
 attention\_mask = attention\_mask.cuda() # O(n + group\_size)  
 else:  
 warn("CUDA is not available, using CPU")  
 return {  
 "input\_ids": padded\_tokens,  
 "attention\_mask": attention\_mask,  
 }  
  
 def get\_logits\_matrix(self, tokens: TokenIDS) -> Tensor:  
 *"""Given a sequence of tokens,  
 returns the logits matrix of shape (group\_size, vocab\_size)  
 where logits[i] is the logits vector of the i-th next token  
 complexity: O(n^2 + group\_size^2) where n is the length of the tokens  
 notice that the logits are not divided by the temperature in this function."""* # define n as the number of tokens in tokens  
 model\_kwargs = self.prepare\_model\_kwargs(tokens) # O(n + group\_size)  
 with no\_grad():  
 # The time complexity of causal language model`s \_\_call\_\_ function  
 # is O(n^2) where n is the length of the inputs  
 outputs = self.model(  
 \*\*model\_kwargs  
 )  
 # the length of all the inputs is n + group\_size - 1  
 # so the complexity of this line is O((n + group\_size - 1)^2)  
 # which is O(n^2 + group\_size^2 + group\_size \* n)  
 # we now that if a > b and a, b > 1 then a^2 > ab  
 # so the complexity is O(n^2 + group\_size^2)  
 unscaled\_relevant\_logits: Tensor  
 unscaled\_relevant\_logits = outputs.logits[0, -self.group\_size:, :self.vocab\_size]  
 # The shape of unscaled\_relevant\_logits is (group\_size, vocab\_size)  
 # So the complexity of this line should be  
 # O(group\_size) because we are coping group\_size \* vocab\_size  
 # elements from one tensor to the another  
 return unscaled\_relevant\_logits  
  
 def get\_prob\_mat(  
 self, tokens: TokenIDS, generation\_start: int  
 ) -> Tensor:  
 *"""Returns the probability matrix  
 as a list of lists of floats  
 Time complexity: O(n^2 + group\_size^2)  
 where n is the number of tokens"""* unscaled\_relevant\_logits = self.get\_logits\_matrix(tokens)  
 # O(n^2 + group\_size^2)  
 # unscaled\_relevant\_logits is a tensor of shape (group\_size, vocab\_size)  
 if not self.end\_of\_sentence\_stop:  
 unscaled\_relevant\_logits[:, self.end\_of\_sentence\_id] = -float('inf')  
 # setting a vector of size vocab\_size  
 # so the complexity is O(group\_size)  
 # setting the logits to -inf so the probability will be 0  
 penalized\_logits = self.repetition\_penalty\_strategy(  
 unscaled\_relevant\_logits, tokens, generation\_start  
 )  
 # O(group\_size \* n)  
 # where n is the number of tokens generated by the algorithm  
 prob\_tensor = self.logits\_to\_probs(penalized\_logits) # O(group\_size)  
 # We are doing a softmax operator  
 # of group\_size different vectors of size vocab\_size  
 # The complexity of the softmax for each vector is  
 # O(1) because the size of the vector size is constant  
 # the complexity of this line is O(group\_size)  
 # because we are doing group\_size softmax operations  
 return prob\_tensor  
  
 def logits\_to\_probs(self, penalized\_logits: Tensor) -> Tensor:  
 *"""Gets the logits matrix and returns the probability matrix  
 Time complexity: O(group\_size)"""* if self.use\_softmax:  
 # if the generation type  
 # is not greedy then we need to apply softmax  
 # to the penalized logits  
 if self.temp != 1.0:  
 penalized\_logits /= self.temp  
 # O(group\_size \* vocab\_size)  
 # because we are dividing a matrix of size (group\_size, vocab\_size)  
 return Softmax(dim=1)(penalized\_logits)  
 # O(group\_size \* vocab\_size) so the complexity is O(group\_size)  
 return penalized\_logits  
  
 def \_\_str\_\_(self):  
 return f"GroupedGenerationUtils({self.as\_dict()})"  
  
 def \_\_repr\_\_(self):  
 return str(self)  
  
 def as\_dict(self):  
 return {attr\_name: getattr(self, attr\_name)  
 for attr\_name in self.descriptive\_attrs}

base\_pipeline.py

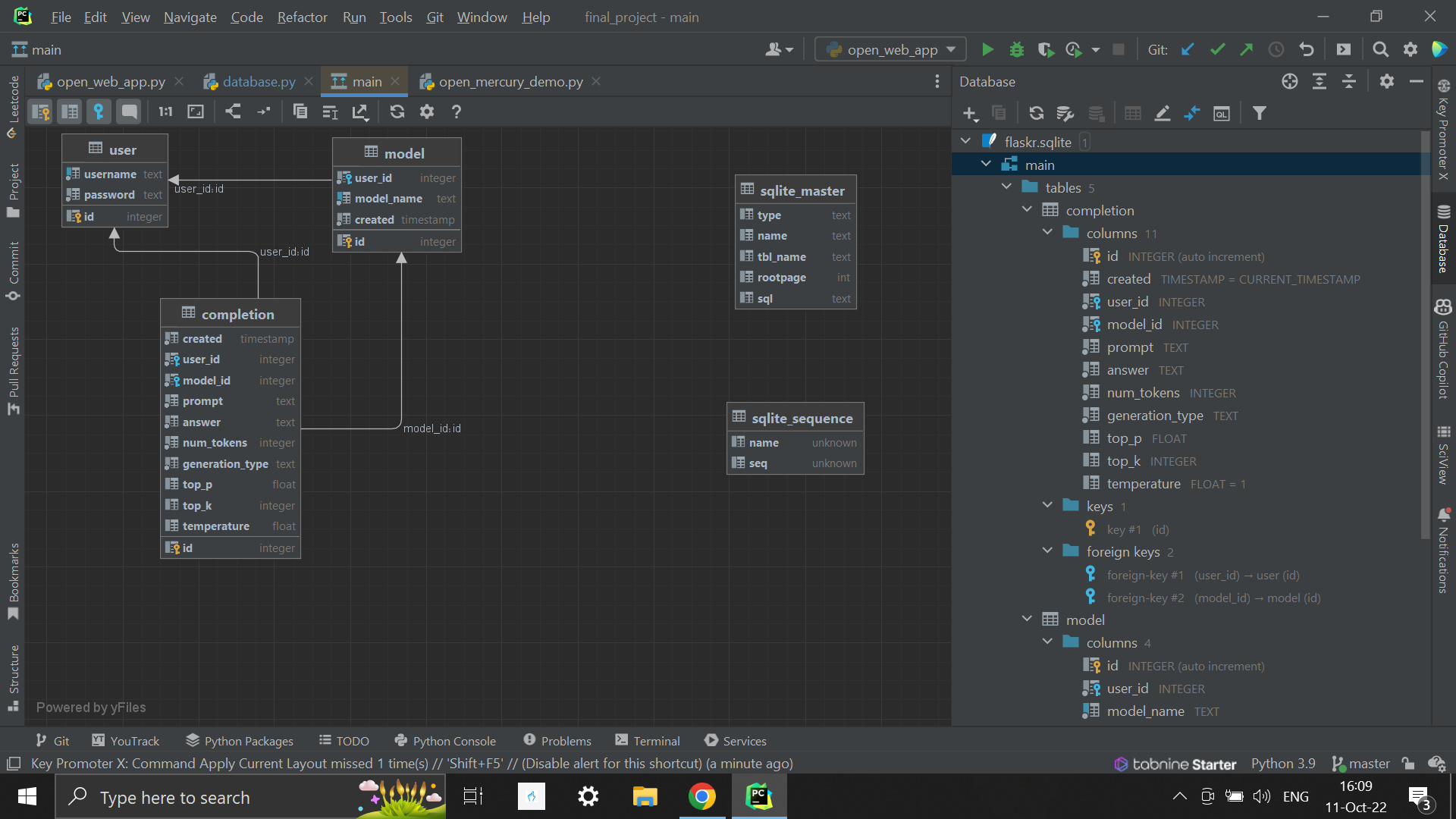
from \_\_future\_\_ import annotations  
  
from abc import ABC, abstractmethod  
from collections.abc import Callable  
from typing import Optional, List, Union, Dict, Any  
  
from torch import LongTensor  
from transformers import (  
 AutoTokenizer,  
 AutoConfig,  
 PreTrainedTokenizer,  
)  
from transformers.tokenization\_utils\_base import TruncationStrategy  
  
from .generation\_type import GenerationType  
from .generation\_utils import GroupedGenerationUtils  
from .postprocessor import PostProcessor  
from .preprocessor import PreProcessor  
from .repetition\_penalty import RepetitionPenaltyStrategy, DEFAULT\_REPETITION\_PENALTY  
from .completion\_dict import CompletionDict  
from .token\_ids import TokenIDS  
  
MAX\_MODEL\_INPUT\_SIZE = 32768  
  
  
def remove\_nones(d: Dict[str, Any]) -> Dict[str, Any]:  
 *"""Returns a copy of a dictionary with all the not None values"""* return {key: d[key] for key in d.keys() if d[key] is not None}  
  
  
def get\_padding\_id(tokenizer: PreTrainedTokenizer):  
 padding\_id = tokenizer.pad\_token\_id  
 if not isinstance(padding\_id, int):  
 padding\_id = tokenizer.unk\_token\_id  
 if not isinstance(padding\_id, int):  
 padding\_id = tokenizer.mask\_token\_id  
 if not isinstance(padding\_id, int):  
 raise RuntimeError(f"padding\_id is {padding\_id} and its type is {type(padding\_id)}")  
 return padding\_id  
  
  
class GroupedGenerationPipeLine(Callable, ABC):  
 *"""An abstract base class for  
 A callable object that given a func\_prompt  
 and length of wanted answer,  
 generates text  
 the text generator has a model,  
 and some parameters  
 (Defined in the subclasses)"""* framework: str = "pt"  
 descriptive\_attrs = (  
 "model\_name",  
 "generation\_type",  
 "answer\_length\_multiplier",  
 "wrapped\_model",  
 )  
  
 def \_\_init\_\_(  
 self,  
 model\_name: str,  
 group\_size: int,  
 temp: Optional[float] = None,  
 end\_of\_sentence\_stop: Optional[bool] = None,  
 repetition\_penalty\_strategy: RepetitionPenaltyStrategy = DEFAULT\_REPETITION\_PENALTY,  
 answer\_length\_multiplier: float = 16,  
 ):  
 *"""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  
 answer\_length\_multiplier: int  
 if the answer length is not given,  
 the maximum answer length is set to:  
 the length of the prompt \* answer\_length\_multiplier  
 repetition\_penalty\_strategy: RepetitionPenaltyStrategy  
 The strategy for the repetition penalty  
 """* self.model\_name: str = model\_name  
 tokenizer = AutoTokenizer.from\_pretrained(model\_name)  
 end\_of\_sentence\_id = tokenizer.eos\_token\_id  
 end\_of\_sentence\_stop = end\_of\_sentence\_stop and end\_of\_sentence\_id is not None  
 max\_input\_len = tokenizer.model\_max\_length  
 max\_len\_is\_huge = max\_input\_len > MAX\_MODEL\_INPUT\_SIZE  
 if max\_len\_is\_huge or max\_input\_len is None:  
 config = AutoConfig.from\_pretrained(model\_name)  
 max\_input\_len = config.max\_position\_embeddings  
 max\_len\_is\_still\_huge = max\_input\_len > MAX\_MODEL\_INPUT\_SIZE  
 if max\_len\_is\_still\_huge or max\_input\_len is None:  
 raise ValueError(  
 "The maximum length of the model is too big"  
 )  
 self.pre\_processing\_strategy: PreProcessor = PreProcessor(  
 tokenizer=tokenizer,  
 max\_input\_len=max\_input\_len,  
 )  
 self.post\_processing\_strategy: PostProcessor = PostProcessor(  
 tokenizer=tokenizer,  
 )  
 wrapped\_model\_kwargs: Dict[str, Any] = {  
 "model\_name": model\_name,  
 "group\_size": group\_size,  
 "max\_input\_len": max\_input\_len,  
 "end\_of\_sentence\_id": end\_of\_sentence\_id,  
 "end\_of\_sentence\_stop": end\_of\_sentence\_stop,  
 "repetition\_penalty\_strategy": repetition\_penalty\_strategy,  
 "padding\_id": get\_padding\_id(tokenizer),  
 "temp": temp,  
 "use\_softmax": self.generation\_type.requires\_softmax(),  
 "vocab\_size": tokenizer.vocab\_size,  
 }  
 self.wrapped\_model: GroupedGenerationUtils = GroupedGenerationUtils(\*\*remove\_nones(wrapped\_model\_kwargs))  
 self.answer\_length\_multiplier: float = answer\_length\_multiplier  
  
 @property  
 @abstractmethod  
 def generation\_type(self) -> GenerationType:  
 *"""A method that chooses the generation type  
 Returns:  
 a GenerationType object"""* raise NotImplementedError  
  
 @abstractmethod  
 def \_forward(  
 self,  
 tokenized\_prompt: LongTensor,  
 num\_new\_tokens: Optional[int] = None,  
 num\_return\_sequences: int = 1,  
 ) -> List[TokenIDS]:  
 *"""A helper method for \_\_call\_\_ that generates the new tokens  
 Has a unique implementation for each subclass  
 Args:  
 tokenized\_prompt: List[int]  
 the tokenized prompt from the preprocess method  
 num\_new\_tokens: int - the number of new tokens to generate  
 from the \_\_call\_\_ method  
 Returns:  
 the prompt + generated text as a list/tuple of ints"""* pass  
  
 def \_\_call\_\_(  
 self,  
 prompt\_s: Union[str, List[str]],  
 max\_new\_tokens: Optional[int] = None,  
 return\_tensors: bool = False,  
 return\_text: bool = True,  
 return\_full\_text: bool = True,  
 clean\_up\_tokenization\_spaces: bool = False,  
 prefix: str = "",  
 num\_return\_sequences: int = 1,  
 truncation: TruncationStrategy = TruncationStrategy.DO\_NOT\_TRUNCATE,  
 postfix: str = "",  
 ) -> CompletionDict | List[CompletionDict] | List[List[CompletionDict]]:  
 *"""The function that outside code should call to generate text  
 Args:  
 prompt\_s: str or list of str - the prompt(s) to start the generation from  
 (the text given by the user)  
 if many prompts are given as a list,  
 the function process each one independently and returns them as a list.  
 (the same as calling [\_\_call\_\_(prompt, \*args, \*\*kwargs)  
 for prompt in prompts])])  
 max\_new\_tokens: Optional[int] > 0  
 - the number of tokens to generate  
 if None, the function will generate tokens  
 until one of them is the end of sentence token  
 return\_tensors: bool - whether to return the generated token ids  
 return\_text: bool - whether to return the generated string  
 return\_full\_text: bool - whether to return the full text  
 (prompt + generated text)  
 (if false, it will return only the generated text)  
 clean\_up\_tokenization\_spaces: bool  
 - whether to clean up tokenization spaces  
 This parameter is forwarded to the decode function of the AutoTokenizer class  
 prefix (`str`, defaults to an empty string):  
 Prefix added to prompt.  
 num\_return\_sequences (`int`, defaults to 1):  
 The number of independently generated answers to return for each prompt.  
 For GroupedSamplingPipeLine:  
 each answer will be generated with different seed.  
 For GroupedTreePipeLine:  
 the num\_return\_sequences with the highest scores will be returned.  
 truncation: TruncationStrategy  
 - whether to truncate the prompt  
 postfix: str  
 - a postfix to add to the prompt  
 Returns:  
 Each result comes as a dictionary with the following keys:  
 - "generated\_text"  
 (`str`, present when `return\_text=True`)  
 -- The generated text.  
 - "generated\_token\_ids" (  
 `torch.tensor`, present when `return\_tensors=True`)  
 -- The token  
 ids of the generated text.  
 """* if max\_new\_tokens is None and \  
 not self.wrapped\_model.end\_of\_sentence\_stop:  
 raise ValueError(  
 "max\_new\_tokens must be given if end\_of\_sentence\_stop is False"  
 )  
 if isinstance(prompt\_s, list):  
 return [self.\_\_call\_\_(  
 prompt\_s=prompt,  
 max\_new\_tokens=max\_new\_tokens,  
 return\_text=return\_text,  
 return\_tensors=return\_tensors,  
 return\_full\_text=return\_full\_text,  
 clean\_up\_tokenization\_spaces=clean\_up\_tokenization\_spaces,  
 truncation=truncation,  
 postfix=postfix,  
 ) for prompt in prompt\_s]  
 tokens: LongTensor  
 prefix\_len: int  
 postfix\_len: int  
 prompt\_len: int  
  
 tokens, prefix\_len, prompt\_len, postfix\_len = self.pre\_processing\_strategy(  
 prompt=prompt\_s,  
 prefix=prefix,  
 truncation=truncation,  
 postfix=postfix  
 )  
 # O(len(prompt) + len(prefix) + len(postfix))  
  
 if max\_new\_tokens is None:  
 max\_new\_tokens = int(prompt\_len \* self.answer\_length\_multiplier)  
 # O(1)  
 tokenized\_answers: List[TokenIDS]  
 tokenized\_answers = self.\_forward(  
 tokens,  
 max\_new\_tokens,  
 num\_return\_sequences  
 )  
  
 if num\_return\_sequences > 1:  
 # O(sum(len(tokenized\_answer) for tokenized\_answer in tokenized\_answers))  
 return [self.post\_processing\_strategy(  
 token\_ids=tokenized\_answer,  
 num\_new\_tokens=max\_new\_tokens,  
 prompt\_len=prompt\_len,  
 return\_text=return\_text,  
 return\_tensors=return\_tensors,  
 return\_full\_text=return\_full\_text,  
 clean\_up\_tokenization\_spaces=clean\_up\_tokenization\_spaces,  
 prefix\_len=prefix\_len,  
 postfix\_len=postfix\_len,  
 ) for tokenized\_answer in tokenized\_answers]  
 else:  
 # O(len(tokenized\_answers[0]))  
 return self.post\_processing\_strategy(  
 token\_ids=tokenized\_answers[0],  
 num\_new\_tokens=max\_new\_tokens,  
 prompt\_len=prompt\_len,  
 return\_text=return\_text,  
 return\_tensors=return\_tensors,  
 return\_full\_text=return\_full\_text,  
 clean\_up\_tokenization\_spaces=clean\_up\_tokenization\_spaces,  
 prefix\_len=prefix\_len,  
 postfix\_len=postfix\_len,  
 )  
  
 def \_\_repr\_\_(self):  
 attrs\_description = ", ".join(  
 f"{attr}={getattr(self, attr)}" for attr in self.descriptive\_attrs  
 )  
 return f"{self.\_\_class\_\_.\_\_name\_\_}: " + attrs\_description  
  
 def \_\_str\_\_(self):  
 return repr(self)  
  
 def as\_dict(self) -> Dict[str, Any]:  
 *"""Returns a dictionary representation  
 of the generator  
 such that it can be saved and loaded  
 using the from\_dict method"""* return {  
 key: getattr(self, key)  
 for key in self.descriptive\_attrs  
 }  
  
 @classmethod  
 def from\_dict(cls, my\_dict: Dict[str, Any]):  
 *"""Creates an GroupedGenerationPipeLine from a dictionary  
 The dictionary should have the same format  
 as the dictionary returned by the as\_dict method"""* if "generation\_type" in my\_dict.keys():  
 my\_dict.pop("generation\_type")  
 wrapped\_model: GroupedGenerationUtils = my\_dict.pop("wrapped\_model")  
 wrapped\_model\_dict = wrapped\_model.as\_dict()  
 my\_dict.update(wrapped\_model\_dict)  
 return cls(\*\*my\_dict)

sampling\_pipeline.py

import heapq  
from collections.abc import Iterator  
from random import seed  
from typing import Callable, List, Dict, Optional, Any  
  
from torch import Tensor, zeros, argmax, multinomial, manual\_seed  
  
from .generation\_type import GenerationType  
from .base\_pipeline import GroupedGenerationPipeLine  
  
  
class ChangingSeed(Iterator):  
 *"""Context manager for changing the seed of the random module.  
 How to use:  
 with ChangingSeed(first\_seed, number\_of\_different\_seeds) as changing\_seed:  
 for \_ in changing\_seed:  
 # do something with random module"""* def \_\_init\_\_(self, default\_seed: int, max\_num\_calls: int):  
 self.default\_seed: int = default\_seed  
 self.curr\_seed: int = self.default\_seed  
 self.max\_num\_calls: int = max\_num\_calls  
 self.curr\_num\_calls: int = 0  
  
 def \_\_enter\_\_(self):  
 self.curr\_num\_calls = 0  
 self.curr\_seed = self.default\_seed  
 return self  
  
 def \_\_exit\_\_(self, \*args):  
 self.curr\_seed = self.default\_seed  
 seed(self.default\_seed)  
 manual\_seed(self.default\_seed)  
  
 def \_\_iter\_\_(self):  
 self.curr\_seed = self.default\_seed  
 return self  
  
 def \_\_next\_\_(self):  
 self.curr\_seed += 1  
 seed(self.curr\_seed)  
 manual\_seed(self.curr\_seed)  
 self.curr\_num\_calls += 1  
 if self.curr\_num\_calls > self.max\_num\_calls:  
 raise StopIteration  
  
  
class TokenProb:  
 *"""Class for storing the probability of a token and the token itself.  
 Used to store the probabilities of the next tokens in the sampling generator.  
 Is useful because it supports the < and > operators, which are used in the  
 heapq module  
 The < and > are the opposite of each other because the heapq module is only supporting minimum heaps  
 and I need a maximum heap"""* \_\_slots\_\_ = ['token\_id', 'prob']  
  
 def \_\_init\_\_(self, token\_id: int, prob: Tensor):  
 self.token\_id: int = token\_id  
 self.prob: Tensor = prob  
  
 def \_\_lt\_\_(self, other: "TokenProb"):  
 *"""Overrides the < operator  
 Comparison is done by the probability"""* return self.prob > other.prob  
  
 def \_\_gt\_\_(self, other: "TokenProb"):  
 *"""Overrides the > operator  
 Comparison is done by the probability"""* return self.prob < other.prob  
  
  
class GroupedSamplingPipeLine(GroupedGenerationPipeLine):  
 *"""A GroupedGenerationPipeLine that generates text  
 using random sampling  
 with top-k or top-p filtering."""* default\_seed: int = 0  
 seed(default\_seed)  
 manual\_seed(default\_seed)  
 unique\_attrs = "top\_k", "top\_p"  
  
 def \_\_init\_\_(self, top\_k: Optional[int] = None,  
 top\_p: Optional[float] = None, \*args, \*\*kwargs):  
 self.top\_p: Optional[float] = top\_p  
 self.top\_k: Optional[int] = top\_k  
 super().\_\_init\_\_(\*args, \*\*kwargs)  
  
 def \_\_setattr\_\_(self, key, value):  
 super().\_\_setattr\_\_(key, value)  
 if key == "default\_seed":  
 seed(value)  
 manual\_seed(value)  
  
 @property  
 def generation\_type(self) -> GenerationType:  
 if self.top\_k is None and self.top\_p is None:  
 return GenerationType.RANDOM  
 if self.top\_k == 1 or self.top\_p == 0.0:  
 return GenerationType.GREEDY  
 if self.top\_k is not None:  
 return GenerationType.TOP\_K  
 if self.top\_p is not None:  
 if self.top\_p < 1.0:  
 return GenerationType.TOP\_P  
 return GenerationType.RANDOM  
 raise RuntimeError("Uncovered case in generation\_type property")  
  
 @property  
 def sampling\_func(self) -> Callable[[Tensor], int]:  
 gen\_type\_to\_filter\_method: Dict[GenerationType, Callable[[Tensor, ], int]] = {  
 GenerationType.TOP\_K: self.top\_k\_sampling,  
 GenerationType.TOP\_P: self.top\_p\_sampling,  
 GenerationType.GREEDY: GroupedSamplingPipeLine.highest\_prob\_token,  
 GenerationType.RANDOM: GroupedSamplingPipeLine.unfiltered\_sampling,  
 }  
 return gen\_type\_to\_filter\_method[self.generation\_type]  
  
 @staticmethod  
 def unfiltered\_sampling(prob\_vec: Tensor) -> int:  
 *"""A sampling function that doesn't filter any tokens.  
 returns a random token id sampled from the probability vector"""* return multinomial(prob\_vec, 1).item()  
  
 @staticmethod  
 def highest\_prob\_token(prob\_vec: Tensor) -> int:  
 *"""Gets a probability vector of shape (vocab\_size,)  
 returns the token id with the highest probability"""* return argmax(prob\_vec).item()  
  
 def top\_p\_sampling(self, prob\_vec: Tensor) -> int:  
 *"""Gets a probability vector of shape (vocab\_size,)  
 computes a probability vector with the top p tokens  
 such that their sum in the original vector is <= self.top\_p.  
 and samples from that vector.  
 If token with the highest probability  
 have a probability higher than top\_p, it will be sampled"""* prob\_sum: float = 0.0  
 converted\_probs = [  
 TokenProb(i, prob) for i, prob in enumerate(prob\_vec)  
 ]  
 heapq.heapify(converted\_probs)  
 new\_probs = zeros(prob\_vec.shape, dtype=float)  
 while prob\_sum < self.top\_p and len(converted\_probs) > 0:  
 curr\_token\_prob: TokenProb = heapq.heappop(converted\_probs)  
 token\_id = curr\_token\_prob.token\_id  
 if curr\_token\_prob.prob <= 0.0:  
 break  
 if curr\_token\_prob.prob > 1:  
 raise ValueError(  
 f"Probability of token {token\_id} "  
 f"in the vector {prob\_vec} "  
 f"is {curr\_token\_prob.prob}"  
 f" which is higher than 1")  
 prob\_sum += curr\_token\_prob.prob  
 new\_probs[token\_id] = curr\_token\_prob.prob  
 if prob\_sum == 0.0:  
 return converted\_probs[0].token\_id  
 return GroupedSamplingPipeLine.unfiltered\_sampling(new\_probs)  
  
 def top\_k\_sampling(self, prob\_vec: Tensor) -> int:  
 *"""Gets a token id: probability mapping  
 returns the TOP\_K tokens  
 with the highest probability.  
 this is the bottleneck of the sampling generator."""* top\_k\_keys: List[int] = heapq.nlargest(  
 self.top\_k,  
 range(prob\_vec.shape[0]),  
 key=lambda x: prob\_vec[x]  
 )  
 prob\_sum = sum(prob\_vec[token\_id] for token\_id in top\_k\_keys)  
 new\_probs = zeros(prob\_vec.shape, dtype=float)  
 for token\_id in top\_k\_keys:  
 new\_probs[token\_id] = prob\_vec[token\_id] / prob\_sum  
 return GroupedSamplingPipeLine.unfiltered\_sampling(new\_probs)  
  
 def generate\_group(self, prob\_mat: Tensor) -> List[int]:  
 *"""Generates a group of tokens  
 using the choice\_function.  
 Complexity: O(group\_size)"""* prob\_mat.cpu()  
 # coping a tensor of size (group\_size, vocab\_size)  
 # so the complexity is O(group\_size)  
 # (vocab\_size is constant)  
 new\_group: List[int] = [  
 self.sampling\_func(prob\_vec)  
 for prob\_vec in prob\_mat  
 ]  
 # the complexity of the loop is O(group\_size)  
 # because self.sampling\_func gets a tensor  
 # of constant size (vocab\_size,)  
 # and therefore must be O(1) in complexity  
 # and the loop has group\_size iterations.  
 del prob\_mat  
 for i, token\_id in enumerate(new\_group):  
 if token\_id == self.wrapped\_model.end\_of\_sentence\_id:  
 return new\_group[:i + 1]  
 # return the group until the end of sentence token included  
 # the complexity of this line is O(group\_size)  
 # because it is coping a list with maximum size of group\_size  
 return new\_group  
  
 def \_forward(  
 self,  
 tokenized\_prompt: Tensor,  
 num\_new\_tokens: Optional[int] = None,  
 num\_return\_sequences: int = 1,  
 ) -> List[List[int]]:  
 *"""Complexity:  
 O(num\_return\_sequences \* (  
 ((n ^ 3) / group\_size) +  
 ((n \* l ^ 2) / group\_size) +  
 group\_size +  
 n)  
 )  
 where l is the number of tokens in the prompt  
 and n is the number of new tokens to generate"""* # let's define l = len(tokenized\_prompt), n = num\_new\_tokens  
 answers: List[List[int]] = []  
 curr\_token\_list: List[int] = tokenized\_prompt.tolist()  
 # coping a tensor of size lso O(l)  
 if num\_new\_tokens is None:  
 raise RuntimeError("num\_new\_tokens is None")  
 for \_ in ChangingSeed(  
 default\_seed=self.default\_seed,  
 max\_num\_calls=num\_return\_sequences):  
 # num\_return\_sequences iterations  
 for \_ in range(num\_new\_tokens // self.wrapped\_model.group\_size):  
 # and each iteration is  
 # O(n ^ 2 + l ^ 2 + group\_size ^ 2 + group\_size)  
 # so the complexity of the loop is  
 # O((n ^ 3) / group\_size + (n \* l ^ 2) / group\_size + group\_size + n)  
 prob\_mat: Tensor = self.wrapped\_model.get\_prob\_mat(  
 curr\_token\_list, len(tokenized\_prompt)  
 )  
 # complexity: O(group\_size ^ 2 + len(curr\_token\_list) ^ 2)  
 # len(curr\_token\_list) <= n + l  
 # so the complexity is  
 # O(group\_size ^ 2 + (n + l) ^ 2) is equals to  
 # O(n ^ 2 + nl + l ^ 2 + group\_size ^ 2)  
 # but nl <= max(n^2, l^2) so the complexity  
 # is O(n ^ 2 + l ^ 2 + group\_size ^ 2)  
 new\_tokens = self.generate\_group(prob\_mat)  
 # complexity: O(group\_size)  
 # len(curr\_token\_list) <= n + l  
 # so the complexity is O(group\_size \* (n + l + group\_size))  
 # len(new\_tokens) = group\_size  
 if self.wrapped\_model.end\_of\_sentence\_id in new\_tokens:  
 # the check is O(group\_size)  
 end\_of\_sentence\_index = new\_tokens.index(  
 self.wrapped\_model.end\_of\_sentence\_id)  
 # O(group\_size) because len(new\_tokens) <= group\_size  
 new\_tokens = new\_tokens[:end\_of\_sentence\_index]  
 # O(group\_size) because end\_of\_sentence\_index < group\_size  
 curr\_token\_list.extend(new\_tokens)  
 # O(group\_size) because len(new\_tokens) <= group\_size  
 answers.append(curr\_token\_list)  
 # O(1)  
 return answers  
  
 def \_\_repr\_\_(self):  
 super\_representation = super().\_\_repr\_\_()  
 unique\_representation = '/n'.join(  
 f"{unique\_attr\_name}={getattr(self, unique\_attr\_name)}"  
 for unique\_attr\_name in self.unique\_attrs)  
 return super\_representation + unique\_representation  
  
 def as\_dict(self) -> Dict[str, Any]:  
 super\_dict = super(GroupedSamplingPipeLine, self).as\_dict()  
 super\_dict.update(  
 {unique\_attr: self.\_\_getattribute\_\_(unique\_attr)  
 for unique\_attr in self.unique\_attrs}  
 )  
 return super\_dict

דיאגרמת בסיס הנתונים של אפליקציית הווב:

בדיאגרמה הזאת ניתן לראות את פרטי בסיס הנתונים של אפליקציית הווב.



1. הנגזרת של פונקציית ReLU איננה מוגדרת מתמטית בנקודה בה x=0 אך ספריות למידת מכונה מגדירות אותה ל 1. [↑](#footnote-ref-1)
2. • בגרפים בחלק השלישי גודל הקבוצה מוצג בכנה מידה לוגריתמי. [↑](#footnote-ref-2)