דגימה בקבוצות – שימוש יעיל במודלי שפה סיבתיים

(Causal Language Models)

עבדות גמר מדעית בהיקף 5 יחידות לימוד במדעי 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

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 $\frac{n}{\text{גודל הקבוצה (פרמטר)}}$ 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 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 P_i 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:

(Suits)[[22

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 [0, 1, 2] ואז לייצוג one hot:

$$\begin{matrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{matrix}$$

Sparse Cross Entropy

פונקציה המודדת את המרחק בין 2 ווקטורי הסתברות: p, q.

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

$$Sparse\ Categorical\ Cross\ Entropy(p, c) = -log(p_c)$$

שהסיבוכיות שלה היא $O(1)$ 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:

(Mikolov et al.)[16]

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 d_k 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 hot במטריצת השיכון.

הפונקציה SoftMax:

(Bridle) [4]

$\mathrm{SoftMax}(z \in \mathbb{R}^k)$ מוגדרת לפי הנוסחה:

$$softmax(z)_i = \frac{e^{z_i}}{\sum_j e^{z_j}}$$

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'ייט.

הווקטור שהפונקציה מחזירה הוא באותו גדול של הווקטור שהפונקציה מקבלת.

הפונקציה מחזירה ווקטור שסכומו אחד וכל איבריו בין אפס לאחד $1 > \sigma(z)_i > 0$.

שימו לב גם ש $\lim_{z_i \to -\infty} softmax(z)_i = 0$.

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:

(Ackley et al. 5) [2]

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

כל איבר במטריצת הלוג'ייט מחולק בטמפרטורה לפני ככה ש:

$$scaled\ logits_i = logits_i / T$$

$$\sigma_T(logits) = probability\ vector_i = softmax(scaled\ logits)_i = \frac{e^{logits_i/T}}{\sum e^{logits_j/T}}$$

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'ייט [1, 1-] עם הטמפרטורות: 2, 1, 0.5:

$$\sigma_2(-1,1) = \left(\frac{e^{-0.5}}{e^{-0.5}+e^{0.5}}, \frac{e^{0.5}}{e^{-0.5}+e^{0.5}}\right) \approx (0.26894, 0.73105)$$

$$\sigma_1(-1,1) = \left(\frac{e^{-1}}{e^{-1}+e^{1}}, \frac{e^{1}}{e^{-0.5}+e^{0.5}}\right) \approx (0.11920, 0.88079)$$

$$_{0.5}(-1,1) = \left(\frac{e^{-2}}{e^{-2}+e^{2}}, \frac{e^{2}}{e^{-2}+e^{2}}\right) \approx (0.01798, 0.98201)$$

מודל שפה סיבתי – Causal Language Model:

(Liu et al.) [14]

מודל למידה עמוקה המקבל רצף של טוקנים $T_0, T_1 \dots T_i \dots T_{n-1}$ ומחזירה ומטריצת לוג׳יט L בה L_i הוא ווקטור הלוג׳יט שמייצג את החיזוי של המודל לטוקן במקום i+1 כאשר L_i תלוי בטוקנים $T_0, T_1 \dots T_{i-1}, T_i$ בלבד. L_n 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 T המייצגת רצף של טוקנים $T_0, T_1 \dots T_i \dots T_n$ 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 M בה:

$$\begin{cases} M_i = onehot(end\ of\ text\ token)\ if\ i = n \\ M_i = T_{i-1}\ else \end{cases}$$

נשים לב ש P_i 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:

(Liu et al.)[14]

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 $Softmax$ 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 $P_{i,t}$ היא ההסתברות של הטוקן ה t 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:

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

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:

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

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 n ניצור מטריצה בגודל $d \times n$ (d 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):

$$p_t(i) = \begin{cases} \sin(\omega_k, t), & i \text{ is even} \\ \cos(\omega_k t), & i \text{ is odd} \end{cases}, \omega_k = \frac{1}{10000^{\frac{2k}{d}}} \rightarrow p_t = \begin{bmatrix} \sin(\omega_1 t) \\ \cos(\omega_1 t) \\ \sin(\omega_2 t) \\ \cos(\omega_2 t) \\ \vdots \\ \sin\left(\omega_{\frac{d}{2}} t\right) \\ \cos\left(\omega_{\frac{d}{2}} t\right) \end{bmatrix}_{d \times 1}$$

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

הכפלה במטריצת שיכון משוחלפת – Transposed Embedding/Unembedding:

(Press and Wolf)[20]

את התוצאה של הדיקודר אנחנו מכפילים במטריצת השיכון המשוחלפת.

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

$$Logit\ matrix = embedding\ matrix^T \cdot decoder\ output$$

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:

(Vaswani et al.) [24]

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:

$$ook\_ahead\_mask_{i,j} = \begin{cases} 1 \; if \; j > i \\ 0 \; if \; j \leq i \end{cases}$$

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:

$$\begin{matrix} 0 & 1 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{matrix}$$

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

0 אם הטוקן במקום $i + 1$ מושפע מהטוקן במקום j ו - 1 אחרת.

תשומת לב עצמית ממוסכת בעזרת מכפלה סקלרית:
Scaled Dot-Product Masked Self Attention

(Vaswani et al.) [24]

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 ad (קיצור של attention dimension).

נגדיר:

seq len - אורך הקלט לטרנספורמר (מספר חיובי ושלם)

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

$W_K, W_Q, W_V \in \mathbb{R}^{ad \times ad}$ - מטריצות פרמטרים הנלמדים על ידי המודל.

הפונקציה מקבלת מטריצה $X \in \mathbb{R}^{seq\ len \times ad}$

הפעולה מחזירה מטריצה באותו גודל של המטריצה שהיא מקבלת.

הגדרה מתמטית לפעולת תשומת הלב:

$$Q\ (query) \in \mathbb{R}^{ad \times seq\ len} = W_q X$$

$$K\ (key) \in \mathbb{R}^{ad \times seq\ len} = W_K X$$

$$V\ (value) \in \mathbb{R}^{ad \times seq\ len} = W_V X$$

$$DP\ (dot\ product) \in \mathbb{R}^{seq\ len \times seq\ len} = \frac{QK^T}{\sqrt{ad}}$$

$$MDP\ (masked\ dot\ product) \in \mathbb{R}^{seq\ len \times seq\ len} = DP - 1{,}000{,}000{,}000 \cdot Mask$$

$$ASM\ (after\ softmax) \in \mathbb{R}^{seq\ len \times seq\ len} = softmax(MDP, axis = 1)$$

$$A\ (attention) \in \mathbb{R}^{seq\ len \times ad} = ASM \cdot V$$

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 ad) 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 ad 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 $\frac{QK^T}{\sqrt{d_k}}$ יוצרת מטריצה DP (קיצור ל dot product) בגודל seq len x seq len בה $DP_{i,j}$ 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 $O(seq\ len^2)$ וזאת משום שאנחנו מכפילים מטריצות בגודל $seq\ len \times ad, ad \times seq\ len$ ומשום שאנחנו מחלקים את כל האיברים בתוצאה ($seq\ len^2$ איברים) בקבוע.

החלוקה של כל איבר מטריצה ב $\sqrt{ad}$ 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 $MDP = DP - 1{,}000{,}000{,}000 \cdot look\_ahead\_mask$:

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

$$1{,}000{,}000{,}000 \cdot look\_ahead\_mask_{i,j} = \begin{cases} 1{,}000{,}000{,}000 \; if \; j > i \\ 0 \; if \; j \leq i \end{cases}$$

את תוצאת המכפלה מחסרים ממטריצת הדמיון ככה שהדמיון בין

אינטואיציה :

במטריצה DP 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 $ASM = softmax(MDP)$:

המטריצה ASM היא טבלה בה $ASM_{i,j}$ 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.

$$ASM_{i,j} \approx 0 \; if \; j > i$$

וזאת מכיוון ש $\lim_{z_i \to -\infty} softmax(z)_i = 0$.

Masked Scores
(before softmax)

0.11	-inf	-inf	-inf
0.19	0.50	-inf	-inf
0.53	0.98	0.95	-inf
0.81	0.86	0.38	0.90

Softmax
(along rows)

Scores

1	0	0	0
0.48	0.52	0	0
0.31	0.35	0.34	0
0.25	0.26	0.23	0.26

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

הפעולה $A = ASM \cdot V$ 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 A יהיה 0.48 כפול הווקטור של הטוקן הראשון במטריצה V ועוד 0.52 כפול הווקטור של הטוקן השני במטריצה V.

תשומת לב רב-ראשית – Multi Head Attention:

(Vaswani et al.)[24]

בהינתן אותם פרמטרים שמקבלת פעולת תשומת לב ועוד פרמטר – מספר הראשים (num_heads)

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

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

num_heads x seq_len x $(\frac{d_k}{num\_head})$ ל Seq_len x d_k

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 d_k וגודל פלט feed forward depth (היפר-פרמטר של המודל)

אקטיבציית ReLU.

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

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

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

היא לומדת מטריצת פרמטרים $W \in \mathbb{R}^{m \times n}$ ו-ווקטור פרמטרים b בגודל m.

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

$$Dense(x) = Wx + b$$

הגדרה לכל איבר:

$$Dense(x)_i = \sum_{j=1}^{n} x_j \cdot W_{i,j} + b_i$$

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 Wx בגודל וכיוון קבוע.

פונקציית ReLU:

(Agarap) [3]

הפונקציה פועלת על כל איבר בטנזור ונוסחתה:

$$ReLU(x) = max\ (0, x)$$

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

$$\frac{\partial ReLU(x)}{\partial x} = \begin{cases} 1\ if\ x \geq 0 \\ 0\ if\ x < 0 \end{cases}$$

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

Papers with Code - Improving Language Understanding by Generative Pre-[18](Training) היא שיטה לאימון מודלים שמטרתם ליצור טקסט בהינתן טקסט. בשיטה זו, מאמנים

[1] הנגזרת של פונקציית ReLU איננה מוגדרת מתמטית בנקודה בה x=0 אך ספריות למידת מכונה מגדירות אותה ל 1.

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).

robot	must	obey	orders
robot	must	obey	orders
robot	must	obey	orders

must
obey
orders

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 $O(vocab\_size + top\_k * log_2 vocab\_size)$ ולא $O(vocab\_size)$ 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:

(Holtzman et al.)[11]

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 $t_1, t_2 \dots t_l$:

$$p(T) = \prod_{i=1}^{l} p(t_i | t_1, t_2 \dots t_{i-1})$$

נבחר מספר $1 \leq w \leq vocab\_size$ 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 $\sum_{i=0}^{n-1} w^i$ פעמים וסיבוכיות זמן הריצה של המודל היא $O(n^2)$ ולכן זמן הריצה של הדגימה הוא $O(\sum_{i=0}^{n-1} w^i \cdot i^2)$ 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 $p \cdot v$ טוקנים כאשר v הוא מספר הטוקנים שהמודל מכיר ולכן סיבוכיות זמן הריצה היא $O(\sum_{i=0}^{n-1}(p \cdot v)^i \cdot i^2)$ ולכן זמן הריצה הוא אקספוננציאלי.

דגימה עם עונשים - penalized sampling:

(Keskar et al.)[12]

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.

2. 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:

(Zhang et al.)[27]

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:

$$\boldsymbol{P}re - \boldsymbol{N}ormalized\ BERT\ \boldsymbol{S}core = \boldsymbol{PNS} = \sum_{x_i \in x} max_{y_j \in y}\left(x_i^T y_j\right)$$

$$\boldsymbol{P}re\ \boldsymbol{S}caled\ BERT\ Recall = \frac{\boldsymbol{PNS}}{|x|}$$

$$\boldsymbol{P}re\ \boldsymbol{S}caled\ BERT\ Precision = \frac{\boldsymbol{PNS}}{|y|}$$

$$\boldsymbol{P}re\ \boldsymbol{S}caled\ BERT\ F1 = 2\frac{BERT * Precision}{BERT +\ Precision} = 2\boldsymbol{PNS}\frac{|y| + |x|}{|y||x|}$$

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:

$$Scaled\ BERT\ Score = \frac{\boldsymbol{PS} - \boldsymbol{b}}{1 - \boldsymbol{b}}$$

המדדים שאציג בתוצאות הם לאחר שקילה.

תצפיות על התנהגותם של מודלי שפה סיבתיים:

(nostalgebraist)[17]

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:

(van Rossum et al.)[23]

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:

(Eric V. Smith)[6]

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:

(Guido van Rossum and Talin)[21]

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:

(Warsaw et al.)[25]

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

ערמת מקסימום\מינימום – Maximum/Minimum Heap:

(’Heapq — Heap Queue Algorithm‘)[10]

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 $O(log_2 n)$ ניתן להפוך רשימה לערמת מינימום בסיבוכיות זמן ריצה $O(n)$.

בעזרת ערימה ניתן למצוא את k הערכים בעלי הערכים הקטנים\גדולים ביותר ברשימה שאורכה n בסיבוכיות זמן ריצה $O(n + klog_2 n)$ 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'ייט של הטוקן שמגיע במקום m+1+i. הווקטור במקום 0 הוא ווקטור הלוג'ייט של הטוקן במיקום m+1 ברצף (הטוקן שבא לאחר סוף הקלט), הווקטור במקום 1 הוא ווקטור הלוג'ייט של הטוקן במקום m+2 וככה הלאה...

הפונקציה משתמשת בפונקציה __call__ של מודל שפה סיבתי שהסיבכויות שלה $O(n^2)$ עם רצף בגודל m+group size-1 ולכן הסיבוכיות של הפונקציה היא $O(m^2 + group\_size^2)$.

הפונקציה: create_prob_mat מקבלת קלט למודל בצורת רצף טוקנים.

ומחזירה מטריצת הסתברות בגודל (group size, vocab size) הבנויה כמו מטריצת הלוג'ייט מהפונקציה get_logit_mat אך הווקטורים בה הם ווקטור הסתברות ולא ווקטורי לוג'ייט.

הפונקציה אחראית על הקטנת הלוג'ייטים של הטוקן המיוחד המעיד על סוף טקסט במקרה בו מספר הטוקנים הרצוי ידוע מראש, היא אחראית על הקטנת הלוג'ייט של טוקנים שהופיעו בקלט באמצעות אסטרטגיית הענשה (RepetitionPenaltyStrategy) ועל המרת ווקטורי לוג'ייט 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 PreProcessor ו PostProcessor 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:

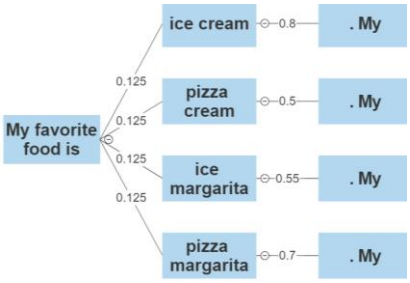
ההסתברות של קבוצה מוגדרת במכפלת הסתברות הטוקנים בה. והסתברות של טקסט מוגדר כמפלת הסתברות הקבוצות.

דוגמה להמחשה – דגימה של 4 טוקנים כאשר גודל הקבוצה הוא 2 ו top_p הוא 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:

$$\begin{matrix} 0.25 & 0.25 & 0.0001 & 0.0001 & \dots \\ 0.03 & 0.02 & 0.25 & 0.25 & \dots \end{matrix}$$

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 $O(1)$.

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 $O(group\ size)$.

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: $n + m - g$ ולכן סיבוכיות השורה היא $O(g^2 + n^2 + m^2)$

היא משתמשת בפונקציה generate_group על מנת ליצור קבוצת טוקנים ממטריצת ההסתברות בסיבוכיות $O(g)$

ומוסיפה את הטוקנים מהקבוצה לרצף הנוכחי בסיבוכיות $O(g)$.

בסך הכל, הסיבוכיות של כל חזרה היא $O(g^2 + n^2 + m^2)$

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 $\frac{n}{g}+1$ חזרות.

לכן סיבוכיות המימוש של הפונקציה forward_ היא: $O(ng+n^3/g+nm^2/g)$

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

$O(ng+\frac{n^3}{g}+\frac{nm^2}{g})$.

יש לשים לב שכשאר גודל הקבוצה שווה למספר הטוקנים שהפונקציה יוצרת ($g=n$), האלגוריתם קורא לפונקציה get_prob_mat רק פעם אחת ולכן זמן הריצה יהיה מינימלי.

סיבוכיות זמן הריצה במקרה זה היא:

$$O(n^2+m^2)$$

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:

$$\begin{matrix} 0.25 & 0.25 & 0.0001 & 0.0001 & \dots \\ 0.04 & 0.02 & 0.25 & 0.25 & \dots \end{matrix}$$

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:

$$0 \quad 0.02001 \quad 0.26 \quad 0.26 \quad \dots$$

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_k, top_p...).

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:

```
g.my_var = "Hello"
print(g.my_var)  # Hello
```

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 auth blueprint 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 model blueprint

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 completion blueprint.

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 בPyTest 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.

תוצאות – חלק ראשון

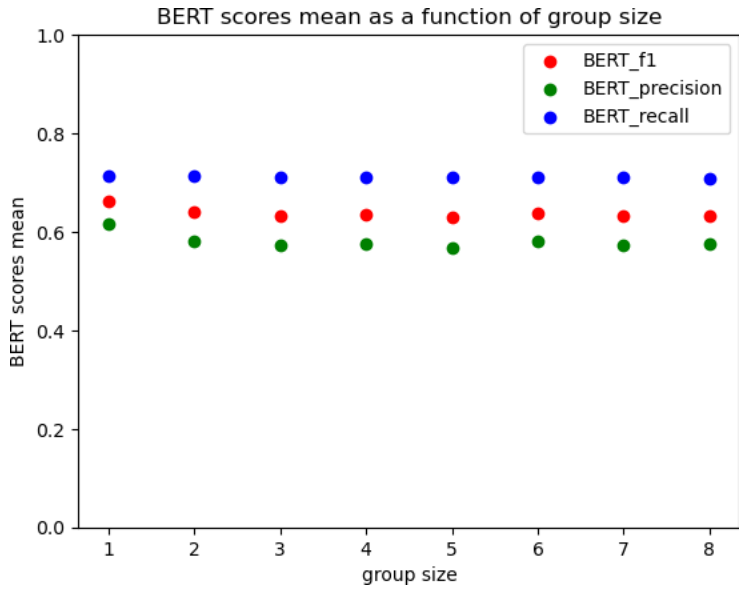
בניסויים הראשוניים הבנתי שבמשימת תרגום אני לא יודע מה יהיה אורך הפלט הרצוי אז שיניתי את הקוד של האלגוריתם ליצירת טקסט ככה שהטקסט יגמר כשהאלגוריתם יצור טוקן מיוחד שקיים בכל מודלי השפה שבא בסוף כל טקסט.

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:

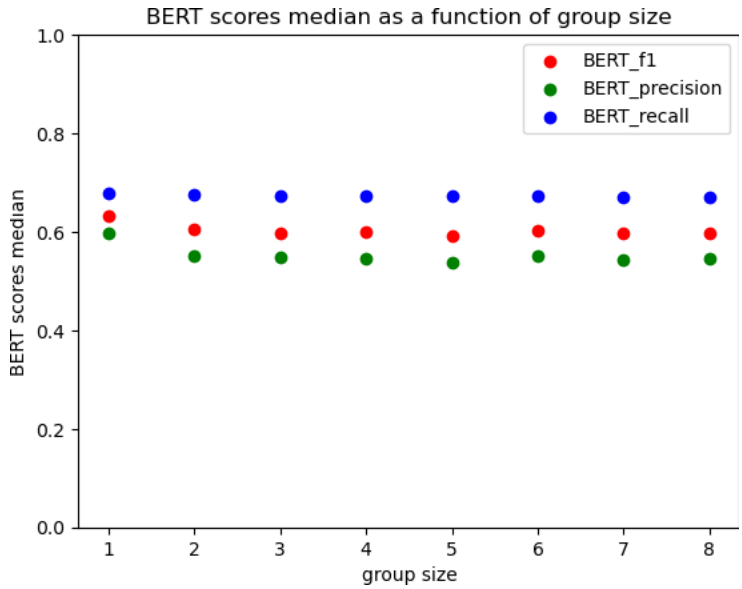
ממוצע:

BERT_recall	BERT_precision	BERT_f1	group_size
0.714	0.617	0.662	1
0.713	0.581	0.639	2
0.711	0.573	0.634	3
0.71	0.576	0.635	4
0.711	0.567	0.629	5
0.71	0.58	0.637	6
0.71	0.572	0.633	7
0.709	0.574	0.634	8



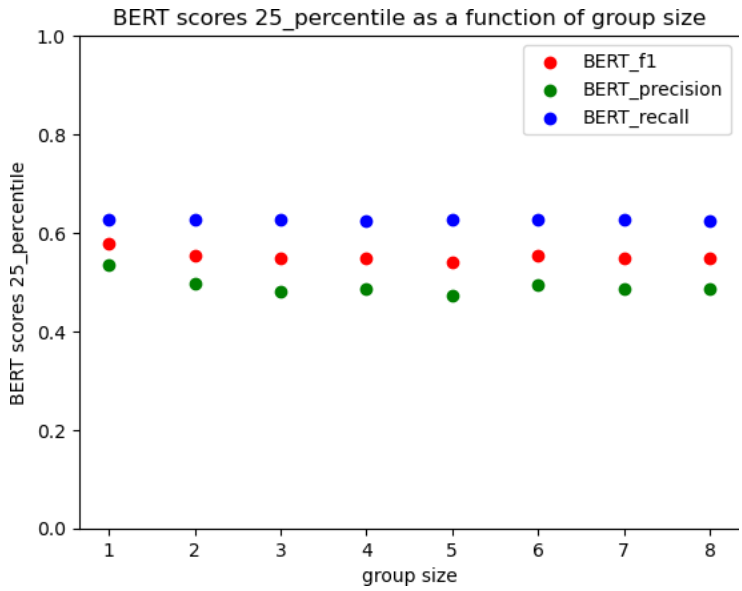
חציון:

BERT_recall	BERT_precision	BERT_f1	group_size
0.677	0.596	0.632	1
0.676	0.552	0.604	2
0.673	0.549	0.598	3
0.672	0.547	0.6	4
0.672	0.537	0.592	5
0.673	0.551	0.603	6
0.671	0.543	0.597	7
0.671	0.546	0.598	8



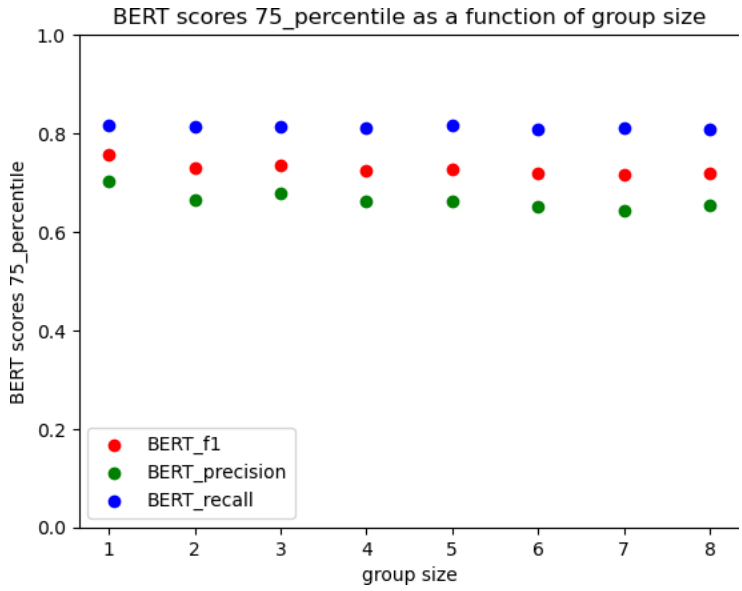
אחוזון 25:

BERT_recall	BERT_precision	BERT_f1	group_size
0.628	0.536	0.578	1
0.627	0.496	0.554	2
0.627	0.481	0.548	3
0.625	0.486	0.548	4
0.626	0.473	0.542	5
0.627	0.494	0.555	6
0.626	0.485	0.548	7
0.626	0.486	0.548	8



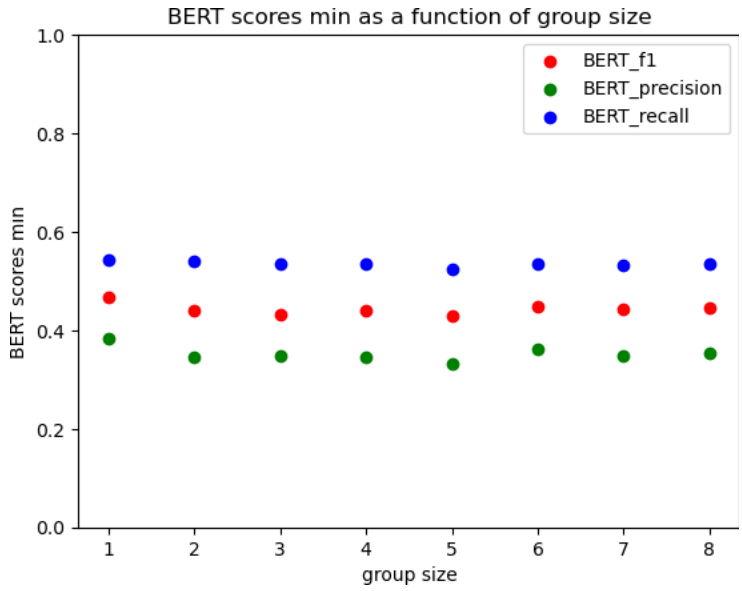
אחוזון 75:

BERT_recall	BERT_precision	BERT_f1	group_size
0.816	0.703	0.757	1
0.815	0.666	0.729	2
0.812	0.679	0.736	3
0.811	0.662	0.725	4
0.817	0.662	0.727	5
0.809	0.652	0.719	6
0.811	0.643	0.715	7
0.81	0.653	0.719	8



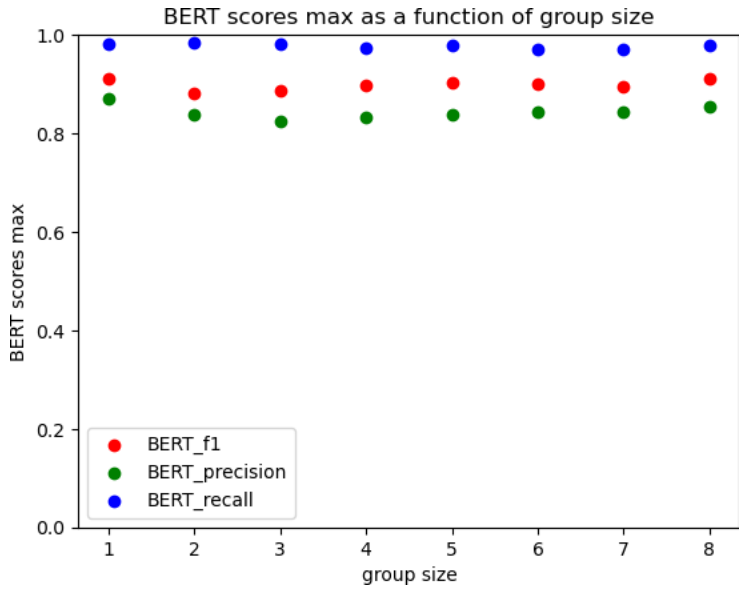
מינימום:

BERT_recall	BERT_precision	BERT_f1	group_size
0.542	0.383	0.468	1
0.541	0.346	0.44	2
0.536	0.348	0.433	3
0.535	0.345	0.44	4
0.525	0.331	0.43	5
0.535	0.362	0.449	6
0.532	0.348	0.444	7
0.536	0.353	0.445	8



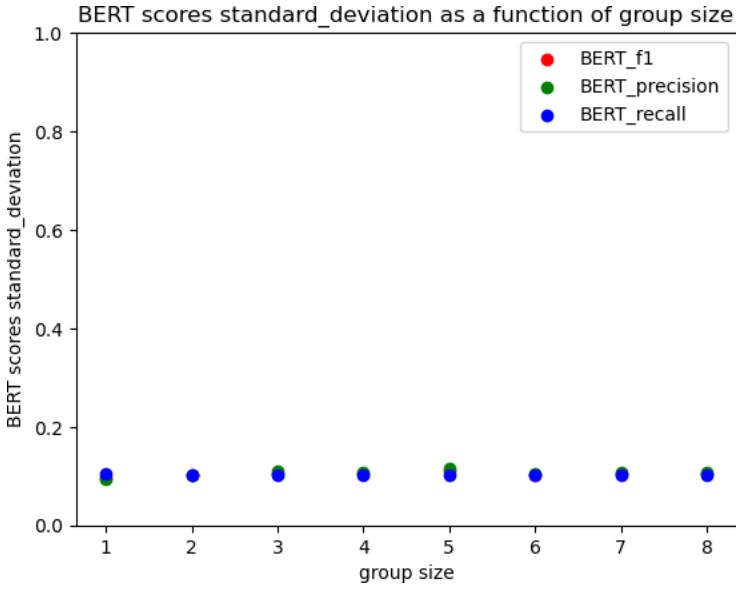
מקסימום :

BERT_recall	BERT_precision	BERT_f1	group_size
0.982	0.87	0.912	1
0.983	0.838	0.881	2
0.981	0.824	0.888	3
0.974	0.833	0.897	4
0.98	0.837	0.902	5
0.971	0.845	0.9	6
0.971	0.844	0.895	7
0.978	0.855	0.912	8



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 $|x|$ הוא אורך הטקסט הרצוי ו $|y|$ הוא אורך הטקסט שהאלגוריתם יוצר.

$$\boldsymbol{P}re\ \boldsymbol{S}caled\ BERT\ Recall = \frac{\boldsymbol{PNS}}{|x|}$$

$$\boldsymbol{P}re\ \boldsymbol{S}caled\ BERT\ Precision = \frac{\boldsymbol{PNS}}{|y|}$$

ולכן אם $Recall > Precision$

נחלק את הנוסחאות של המדדים ב $\boldsymbol{PNS}$ (ידוע ש $\boldsymbol{PNS} > 0$) ונקבל

$$\frac{1}{|x|} > \frac{1}{|y|}$$

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

$|y| > |x|$ כלומר אורך הטקסט שהאלגוריתם יוצר גדול מאורך הטקסט הרצוי.

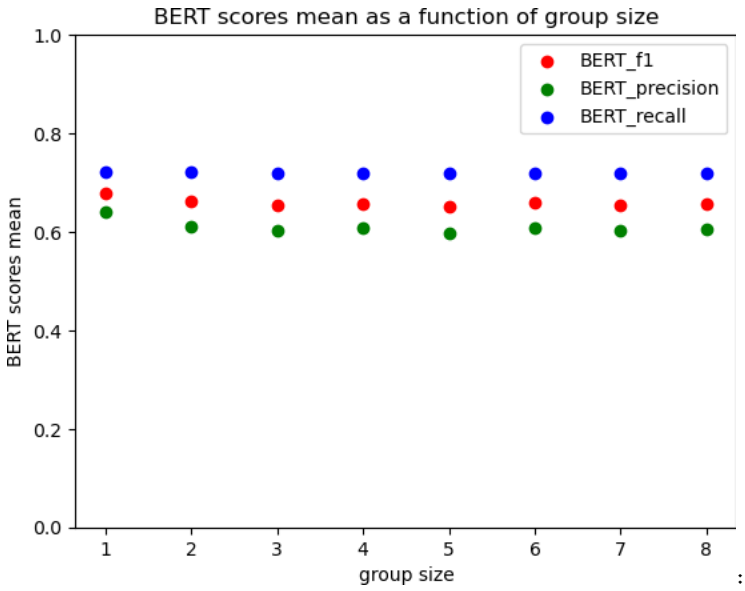
תוצאות החלק השני

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:

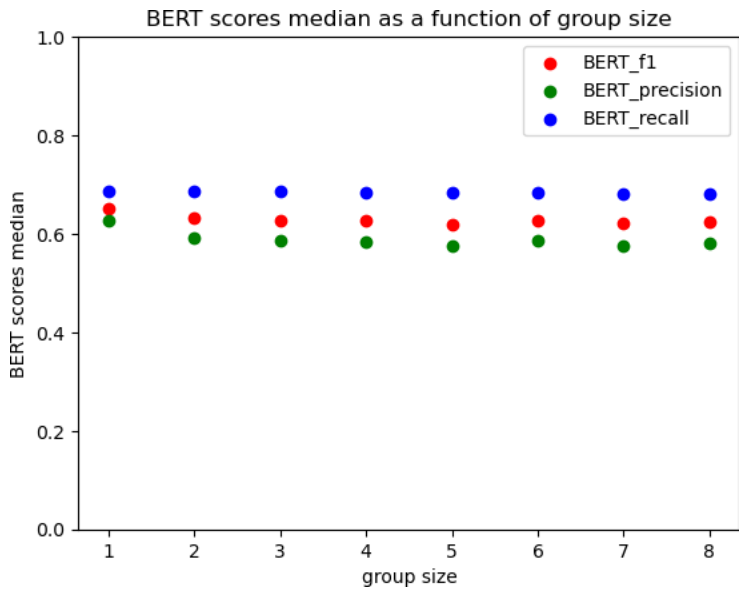
group_size	BERT_f1	BERT_precision	BERT_recall
1	0.678	0.64	0.722
2	0.662	0.612	0.721
3	0.655	0.603	0.72
4	0.657	0.607	0.718
5	0.651	0.596	0.719
6	0.658	0.609	0.718
7	0.654	0.602	0.718
8	0.656	0.605	0.718



:

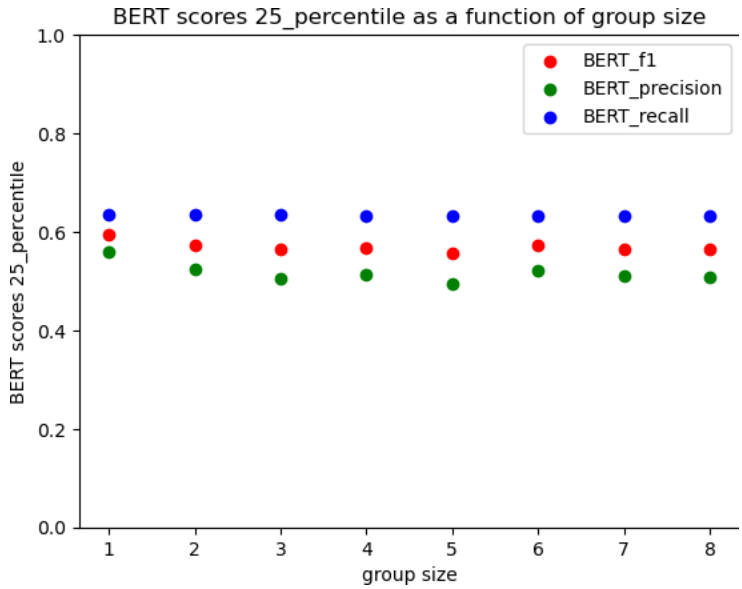
חציון :

BERT_recall	BERT_precision	BERT_f1	group_size
0.687	0.627	0.652	1
0.687	0.591	0.633	2
0.685	0.587	0.627	3
0.683	0.583	0.628	4
0.683	0.576	0.62	5
0.683	0.586	0.628	6
0.682	0.575	0.621	7
0.682	0.581	0.624	8



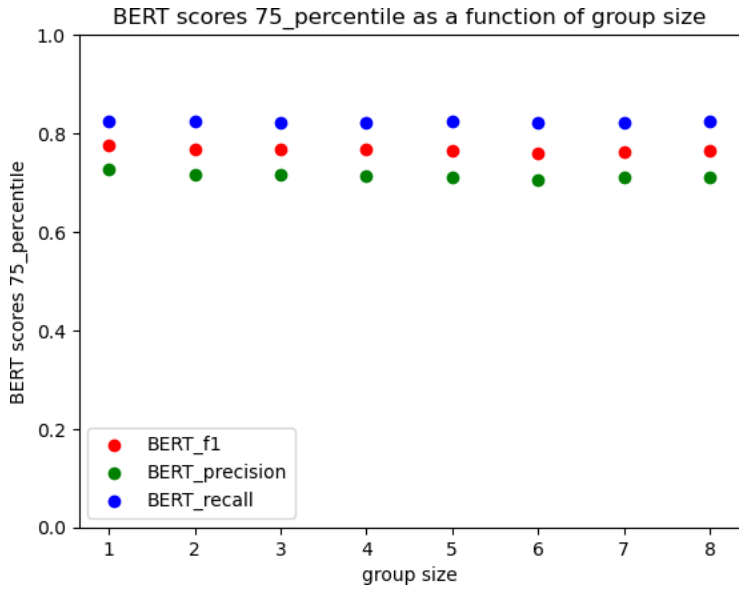
אחוזון 25:

BERT_recall	BERT_precision	BERT_f1	group_size
0.634	0.559	0.593	1
0.635	0.524	0.573	2
0.635	0.506	0.566	3
0.633	0.514	0.567	4
0.633	0.494	0.558	5
0.634	0.521	0.572	6
0.633	0.51	0.565	7
0.633	0.509	0.564	8



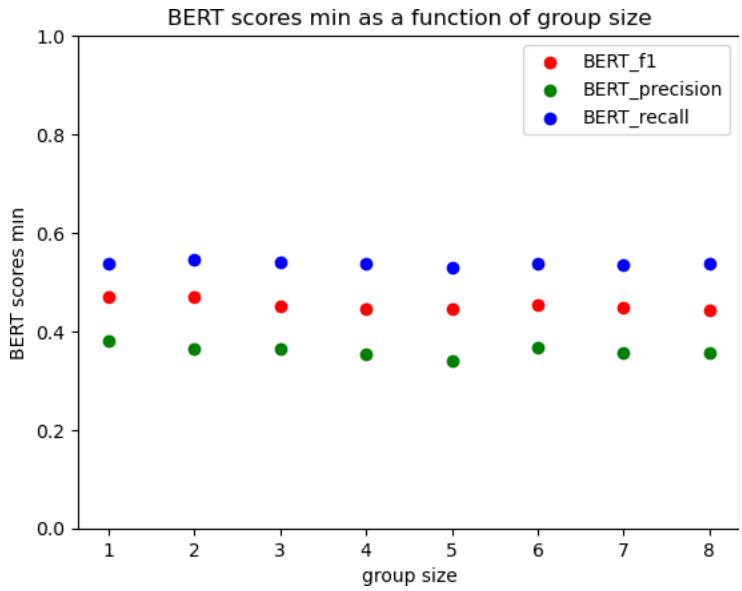
אחוזון 75:

BERT_recall	BERT_precision	BERT_f1	group_size
0.824	0.728	0.774	1
0.824	0.715	0.767	2
0.822	0.717	0.767	3
0.822	0.715	0.767	4
0.824	0.712	0.764	5
0.821	0.706	0.76	6
0.822	0.71	0.762	7
0.824	0.71	0.764	8



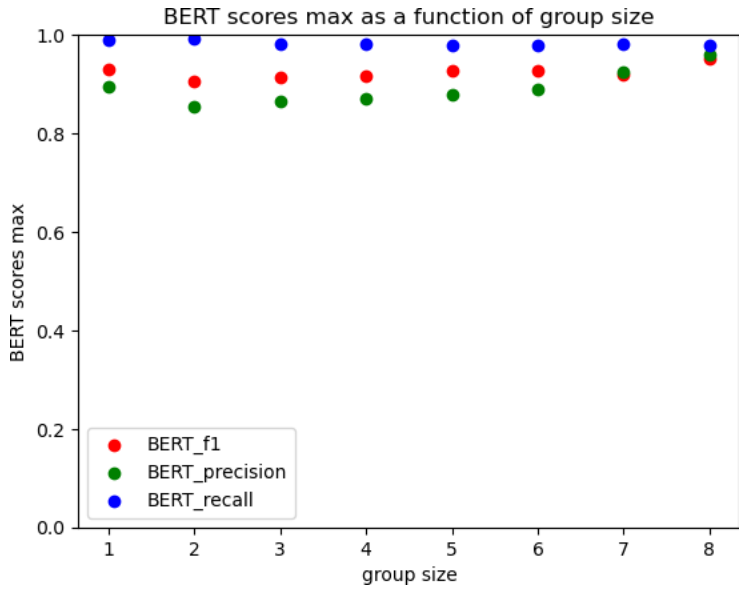
מינימום :

BERT_recall	BERT_precision	BERT_f1	group_size
0.539	0.382	0.469	1
0.546	0.365	0.469	2
0.539	0.366	0.452	3
0.537	0.354	0.446	4
0.529	0.341	0.446	5
0.538	0.369	0.455	6
0.535	0.356	0.449	7
0.539	0.357	0.444	8



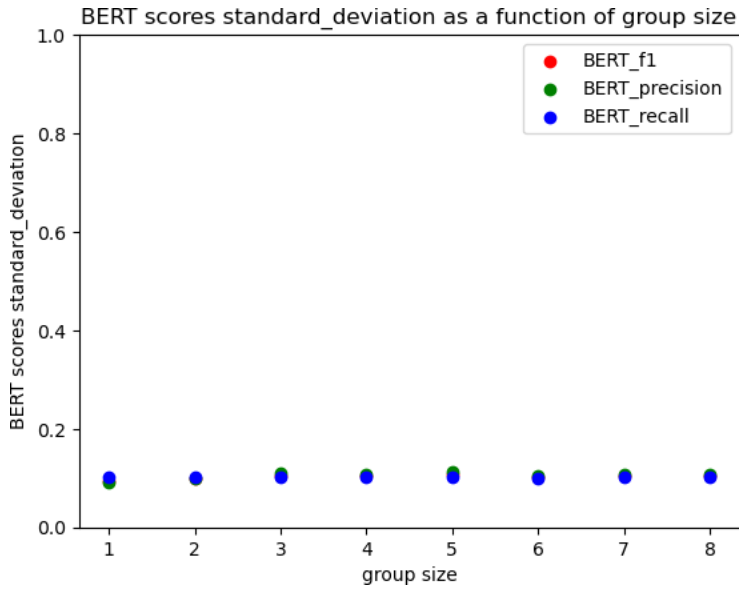
מקסימום :

BERT_recall	BERT_precision	BERT_f1	group_size
0.989	0.896	0.93	1
0.992	0.854	0.905	2
0.98	0.864	0.914	3
0.982	0.871	0.916	4
0.979	0.879	0.927	5
0.978	0.89	0.927	6
0.982	0.924	0.92	7
0.978	0.959	0.951	8



סטיית תקן :

BERT_recall	BERT_precision	BERT_f1	group_size
0.103	0.091	0.095	1
0.102	0.1	0.1	2
0.101	0.109	0.105	3
0.101	0.107	0.104	4
0.102	0.114	0.108	5
0.1	0.104	0.101	6
0.101	0.108	0.105	7
0.101	0.109	0.105	8



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]

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 (2^0) לשתיים (2^1) ואז לארבע (2^2) וככה הלאה עד אלפיים ארבעים ושמונה (2^{11}).

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.

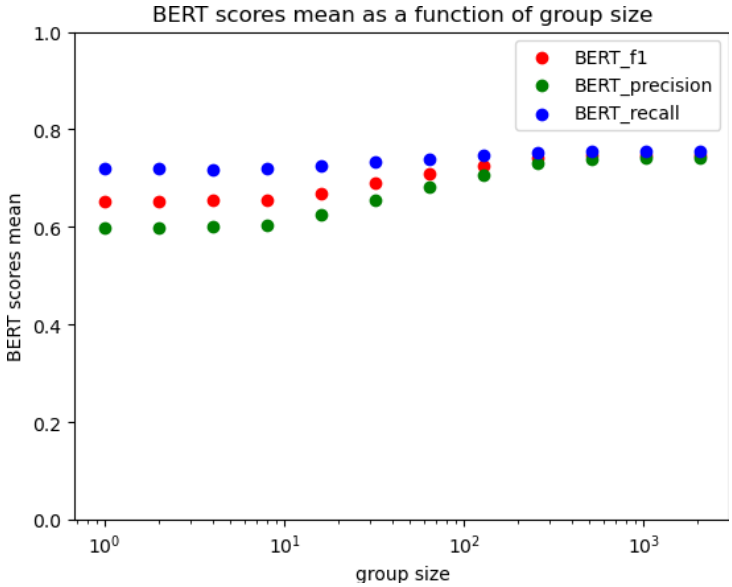
זאת משום שאני מכניס למודל שגודלו : 1 - אורך הפרומפט + גודל הקבוצה

וצריכת הזיכרון במעבד הגרפי גדלה עם גודל הקלט למודל.

[2] • בגרפים בחלק השלישי גודל הקבוצה מוצג בכנה מידה לוגריתמי.

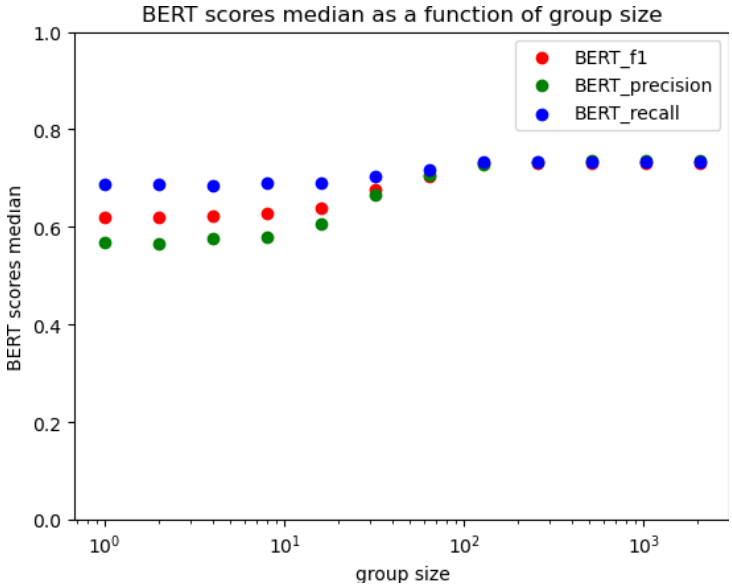
ממוצע:

BERT_recall	BERT_precision	BERT_f1	group_size
0.72	0.599	0.653	1
0.719	0.598	0.652	2
0.718	0.602	0.654	4
0.72	0.603	0.656	8
0.724	0.626	0.67	16
0.732	0.654	0.689	32
0.74	0.682	0.708	64
0.747	0.707	0.725	128
0.754	0.731	0.741	256
0.755	0.739	0.746	512
0.756	0.741	0.747	1024
0.756	0.741	0.747	2048



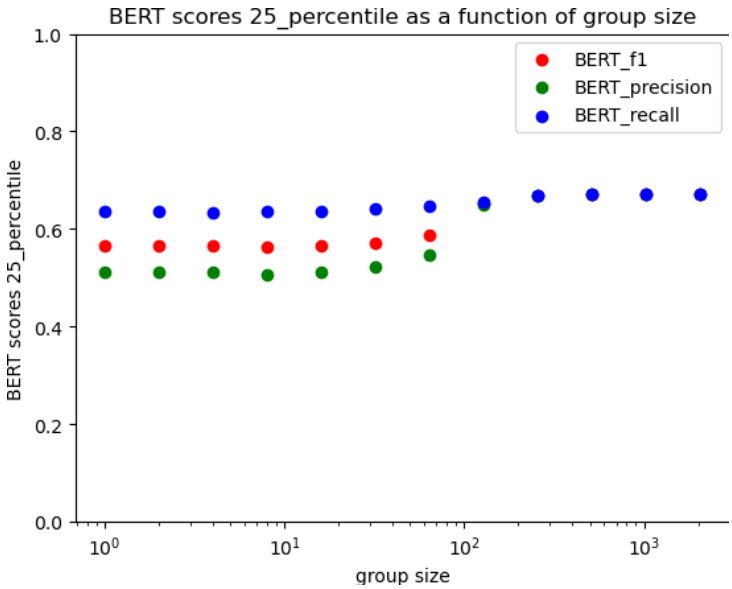
חציון:

BERT_recall	BERT_precision	BERT_f1	group_size
0.688	0.567	0.621	1
0.687	0.566	0.619	2
0.685	0.577	0.624	4
0.69	0.581	0.629	8
0.691	0.606	0.64	16
0.704	0.665	0.677	32
0.718	0.707	0.704	64
0.733	0.729	0.731	128
0.734	0.734	0.732	256
0.734	0.735	0.732	512
0.734	0.735	0.732	1024
0.734	0.735	0.732	2048



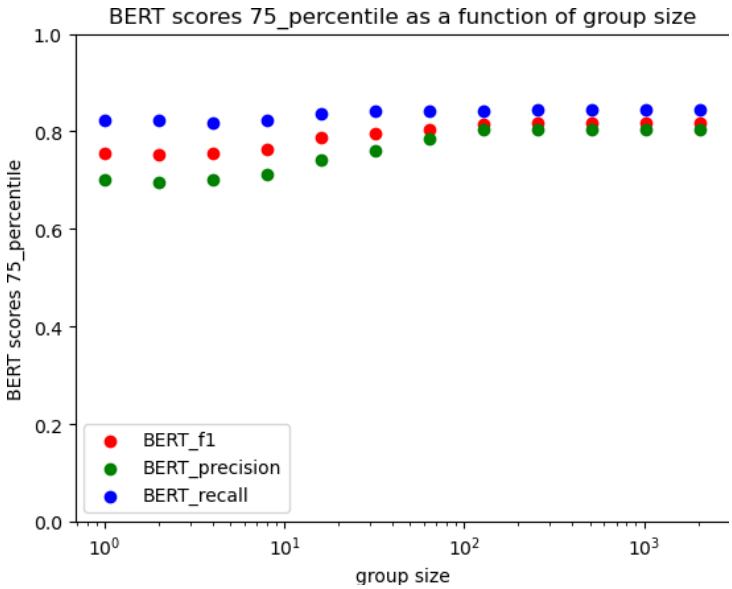
אחוזון 25 :

BERT_recall	BERT_precision	BERT_f1	group_size
0.635	0.512	0.566	1
0.635	0.51	0.565	2
0.634	0.511	0.565	4
0.637	0.508	0.564	8
0.637	0.512	0.565	16
0.641	0.522	0.572	32
0.646	0.546	0.588	64
0.655	0.65	0.651	128
0.668	0.669	0.669	256
0.671	0.67	0.671	512
0.672	0.671	0.672	1024
0.672	0.671	0.672	2048



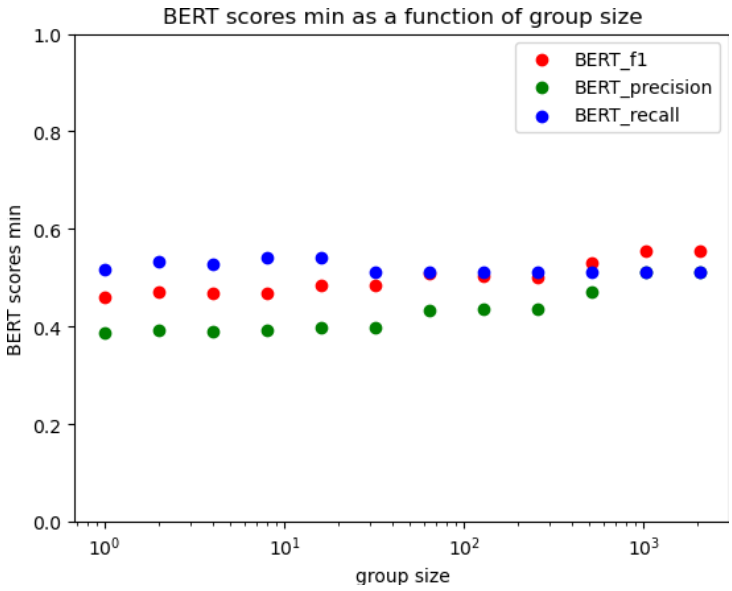
אחוזון 75:

BERT_recall	BERT_precision	BERT_f1	group_size
0.822	0.7	0.755	1
0.823	0.695	0.753	2
0.818	0.702	0.755	4
0.822	0.711	0.763	8
0.836	0.741	0.787	16
0.841	0.76	0.797	32
0.842	0.784	0.805	64
0.842	0.803	0.813	128
0.845	0.803	0.817	256
0.845	0.803	0.817	512
0.845	0.803	0.817	1024
0.845	0.803	0.817	2048



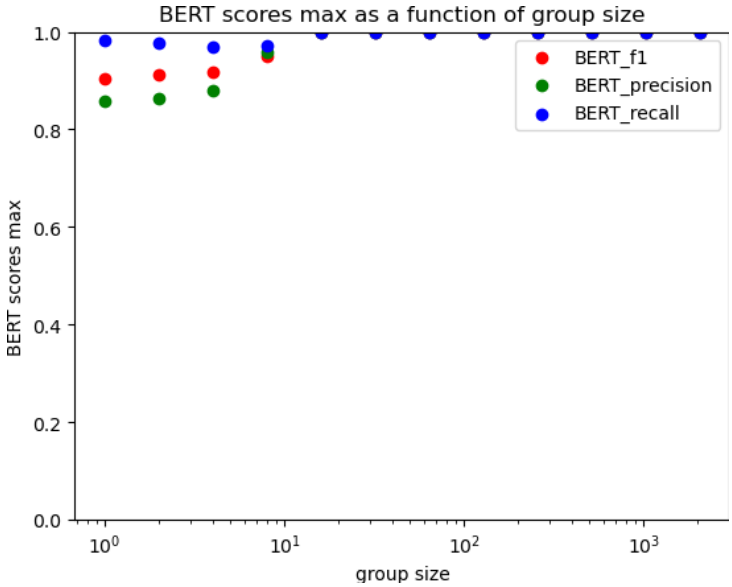
מינימום :

BERT_recall	BERT_precision	BERT_f1	group_size
0.518	0.388	0.46	1
0.534	0.394	0.472	2
0.529	0.391	0.469	4
0.542	0.393	0.469	8
0.541	0.398	0.484	16
0.511	0.399	0.484	32
0.511	0.432	0.508	64
0.511	0.437	0.502	128
0.511	0.436	0.502	256
0.511	0.471	0.529	512
0.511	0.511	0.554	1024
0.511	0.511	0.554	2048



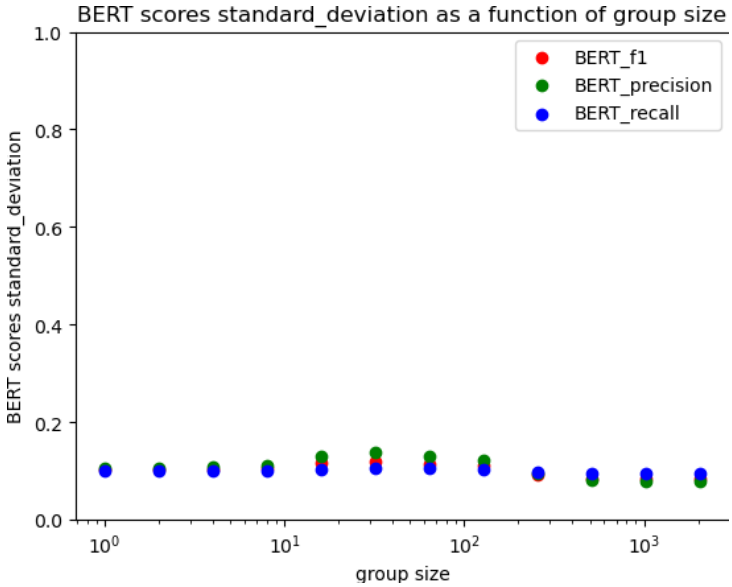
מקסימום :

BERT_recall	BERT_precision	BERT_f1	group_size
0.983	0.859	0.905	1
0.977	0.863	0.912	2
0.969	0.88	0.918	4
0.971	0.959	0.951	8
1.0	1.0	1.0	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



סטיית תקן:

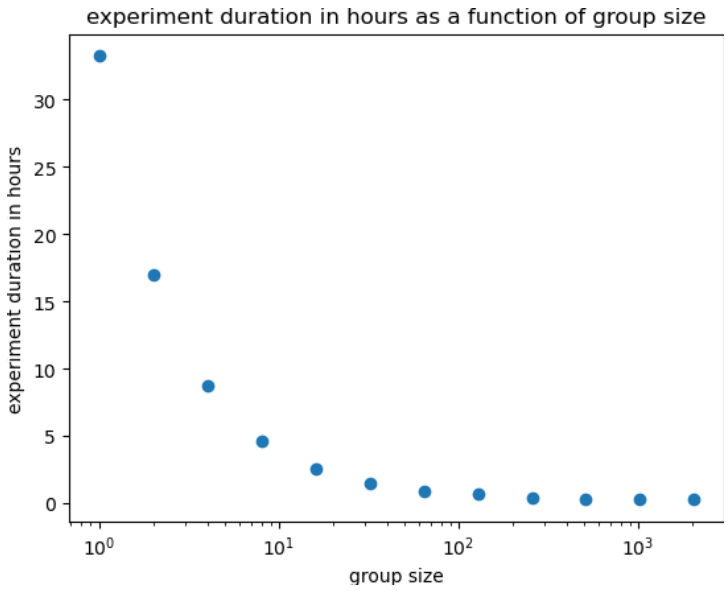
group_size	BERT_f1	BERT_precision	BERT_recall
1	0.102	0.105	0.101
2	0.103	0.106	0.101
4	0.104	0.107	0.101
8	0.105	0.11	0.1
16	0.116	0.129	0.103
32	0.121	0.137	0.107
64	0.115	0.129	0.105
128	0.11	0.122	0.103
256	0.093	0.096	0.097
512	0.085	0.082	0.095
1024	0.084	0.079	0.095
2048	0.084	0.079	0.095



- 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.

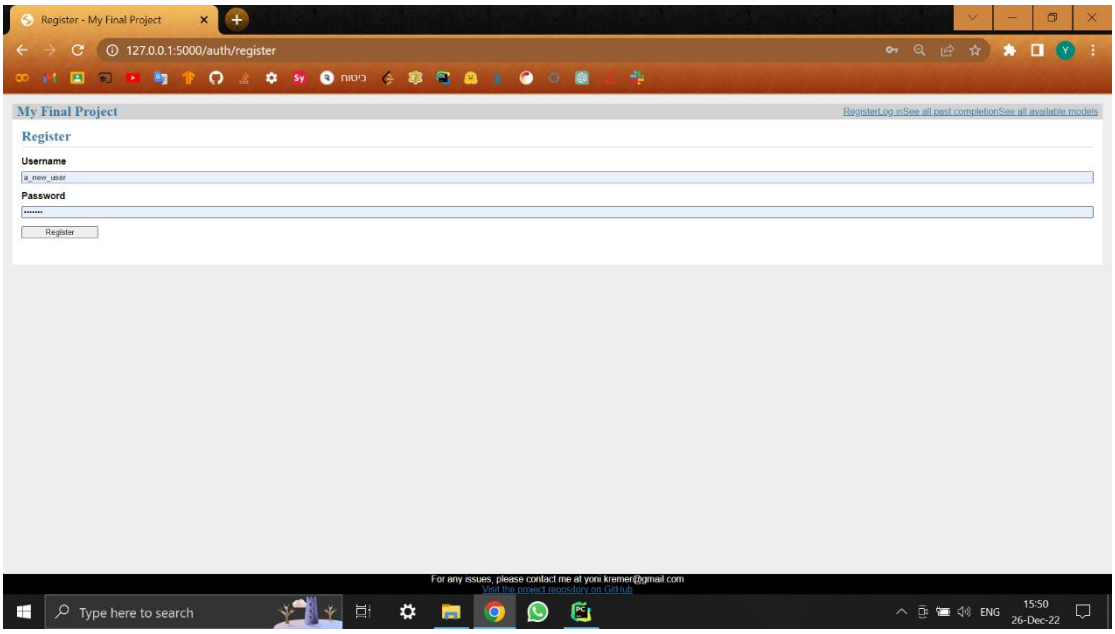
ראיתי שתוצאות הגדלת גודל הקבוצה מ 1024 ל2048 לא משפיעה על המדדים בכלל ומשפיעה לרעה על זמן הריצה.

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.

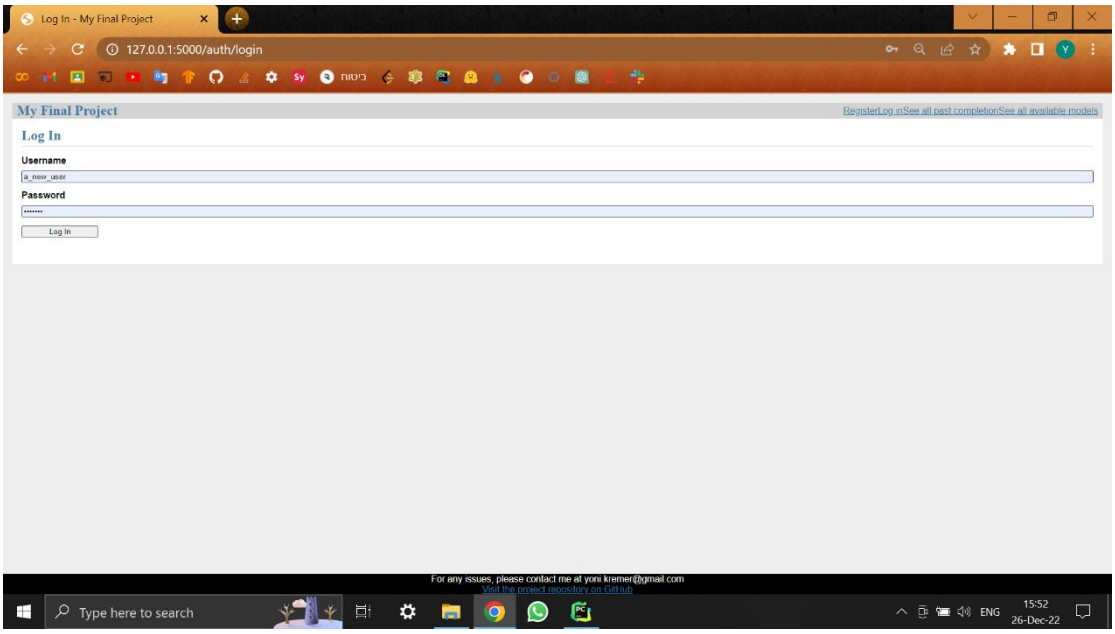
הצגת התוצרים

אפליקציית הווב:

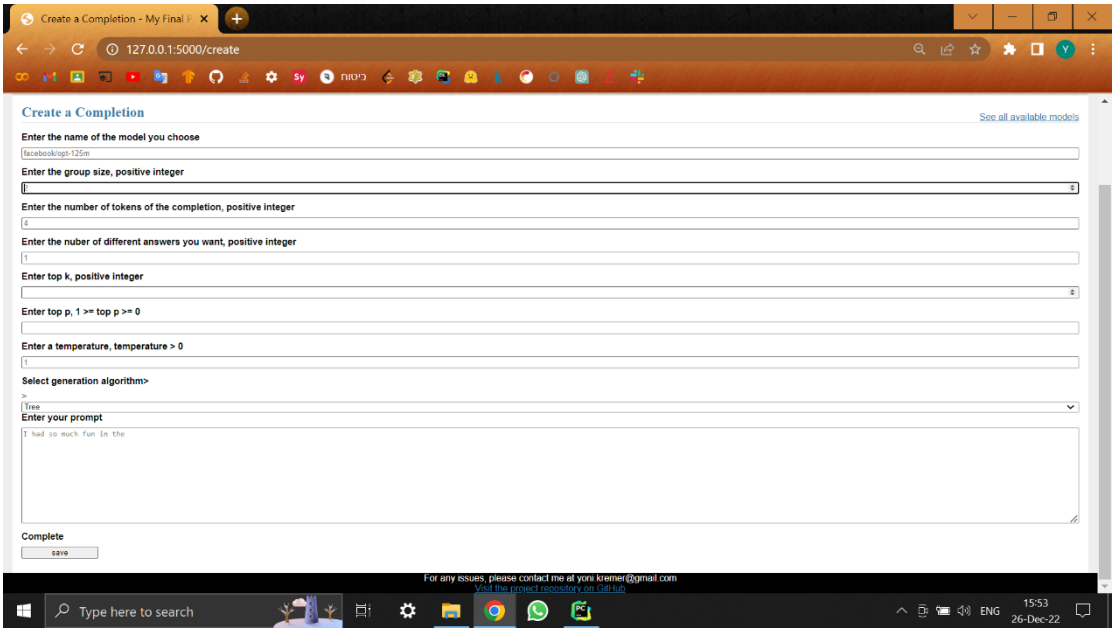
עמוד ההרשמה :



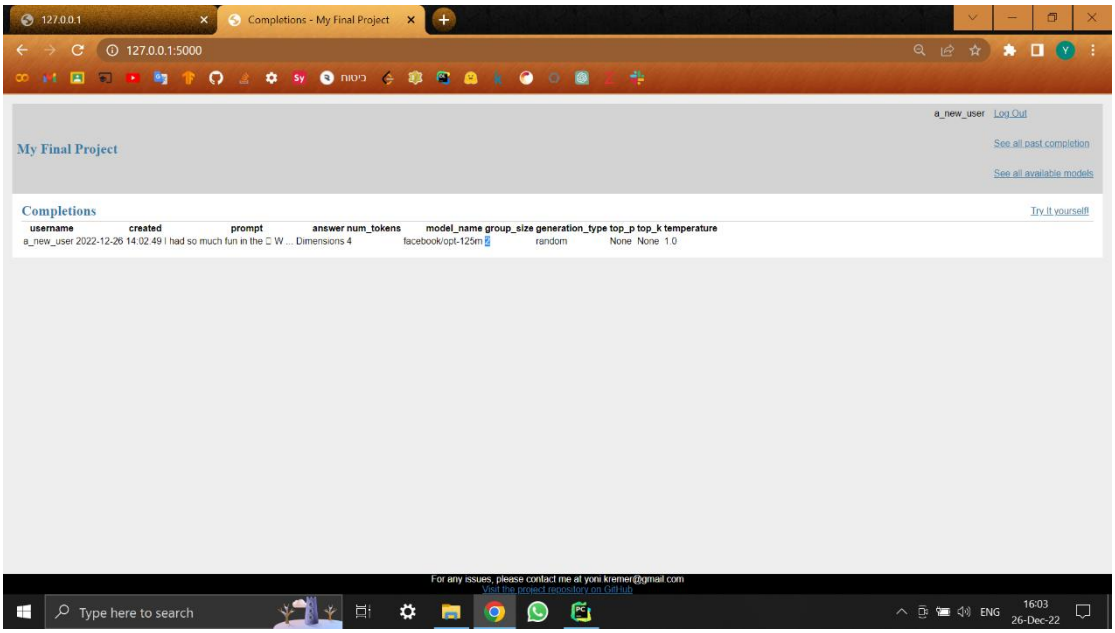
עמוד ההתחברות



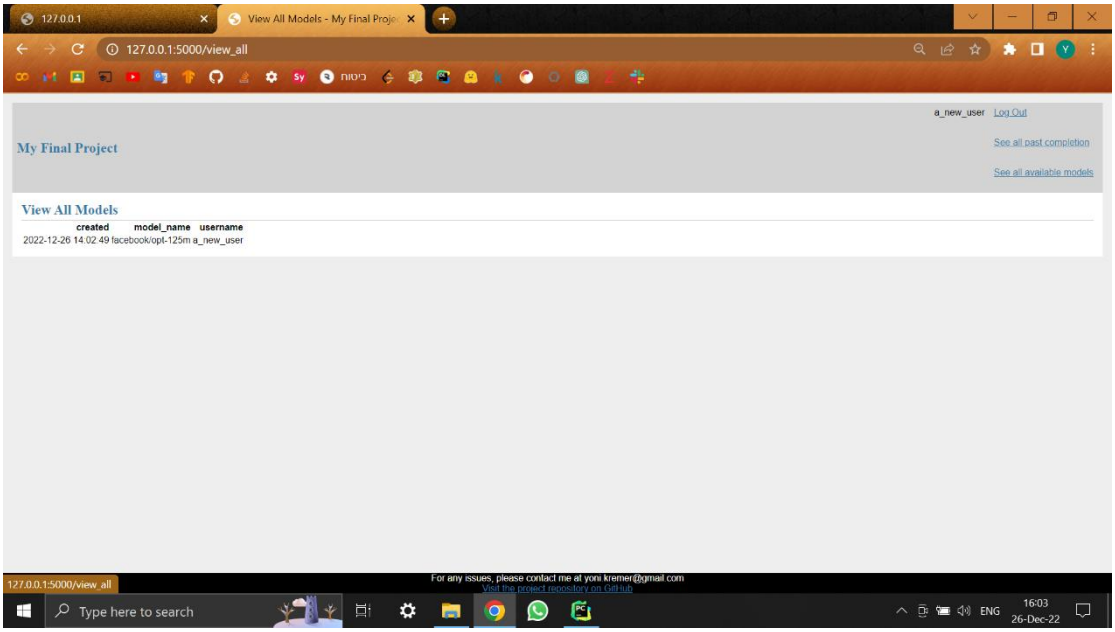
עמוד השימוש באלגוריתם:



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 $O(n^2 + m^2)$.

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 $O(n^3 + nm^2)$[3].

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.

[3] בהנחה שסיבכויות מודל השפה היא $O(n^2)$ כאשר n מייצג את מספר הטוקנים בקלט.

ביבליוגרפיה

.1 Data Model'. Python Documentation, The Python Software Foundation, .3'
.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/0336dcbab05b9d5ad24f4333c76
.58a0e-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

Zhang, Susan, et al. OPT: Open Pre-Trained Transformer Language Models. .26
.arXiv, 21 June 2022. arXiv.org, https://doi.org/10.48550/arXiv.2205.01068

Zhang, Tianyi, et al. BERTScore: Evaluating Text Generation with BERT. .27
..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

```python
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
```

```python
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
```

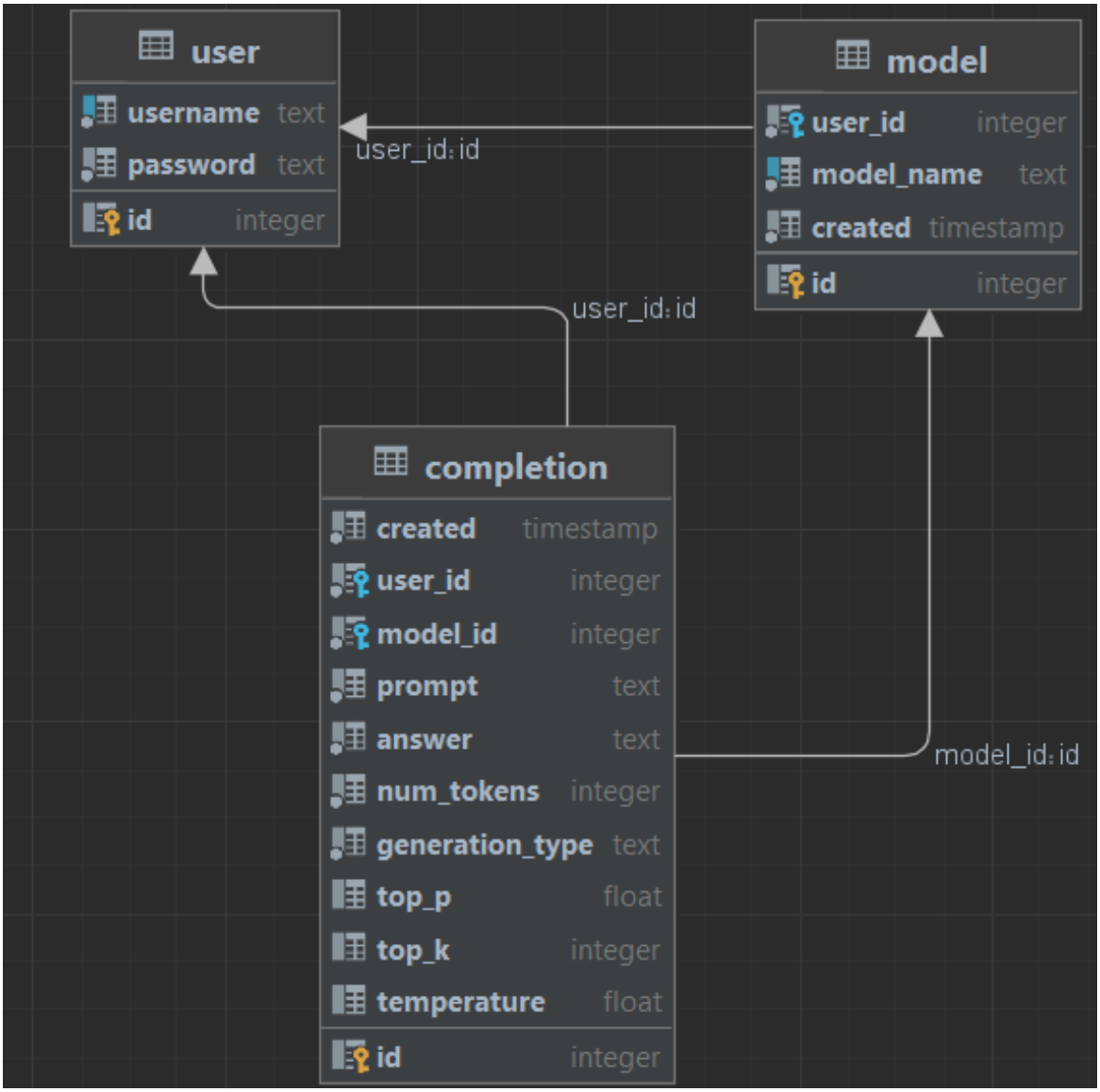
```
                # O(group_size ^ 2 + (n + l) ^ 2) is equals to
                # O(n ^ 2 + nl + l ^ 2 + group_size ^ 2)
                # but nl <= max(n^2, l^2) so the complexity
                # is O(n ^ 2 + l ^ 2 + group_size ^ 2)
                new_tokens = self.generate_group(prob_mat)
                # 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.


user
username text
password text
id integer
model
user_id integer
model_name text
created timestamp
id integer
user_id:id
user_id:id
completion
created timestamp
user_id integer
model_id integer
prompt text
answer text
num_tokens integer
generation_type text
top_p float
top_k integer
temperature float
id integer
model_id:id