

דגימה בקבוצות – שימוש יעיל במודלי שפה סיבתיים

(Causal Language Models)

עבודות 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

2	תוכן עניינים
4	הקדמה אישית
4	תודות
5	מבוא
5	רקע:
5	שאלת המחקר:
5	השערת מחקר:
6	מטרת המחקר:
6	תיאור העבודה:
8	סקירת בינה מלאכותית
8	למידה בהנחיה עצמית – Self-Supervised Learning:
8	ווקטור הסתברות – Probability Vector:
8	ווקטור לוג'יט – Logit Vector:
8	טוקניזציה – Tokenization:
9	Dummy/One Hot Encoding:
10	Sparse Cross Entropy
10	שיכון טוקנים – Token Embedding:
10	הפונקציה SoftMax:
11	טמפרטורה - Temperature:
11	מודל שפה סיבתי – Causal Language Model:
12	טרנספורמר עם דיקודר בלבד – Decoder Only Transformer:
20	אימון ראשוני למידול שפה סיבתי בעזרת יצירת שפה:
21	דגימה – Sampling/Decoding:
25	הבעיות עם שיטות הדגימה הקיימות:
26	מדד ברט – BERT Score:
27	תצפיות על התנהגותם של מודלי שפה סיבתיים:
28	סקירת פייטון
28	רשימת מבני נתונים בפייטון ומבני הנתונים המקבילים בשפות אחרות:
28	רמזי סוג – Type Hints:
28	רב צורתיות (פולימורפיזם):
29	פעולות קסם – Magic/Dunder Methods:
29	קשטנים - Decorators:
30	מחלקת בסיס אבסטרקטית:

30.....	פונקציה אבסטרקטית:
30.....	פעולה סטטית:
31.....	מחלקת נתונים – Data Class:
31.....	המחלקה האבסטרקטית Callable:
31.....	Enum:
31.....	ערמת מקסימום\מינימום – Maximum/Minimum Heap:
32.....	פיתוח מודלי למידה עמוקה בעזרת TensorFlow:
34.....	הסבר האלגוריתם
36.....	פיתוח התכונה
43.....	אפליקציית הווב:
47.....	מימוש הארכיטקטורה לטרנספורמר עם דיקודר בלבד:
48.....	הדמו:
49.....	ניסויים:
74.....	הצגת התוצרים
74.....	אפליקציית הווב:
77.....	סיכום
77.....	המשך המחקר:
78.....	ביבליוגרפיה
82.....	נספחים
82.....	קישור לעמוד הגיטהאב של הפרויקט:
82.....	דיאגרמת בסיס הנתונים של אפליקציית הווב:
82.....	קוד האלגוריתם לדגימה בקבוצות:

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...). גלל 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 גודל הקבוצה לאורך הטקסט הטקסט שהאלגוריתם צריך ליצור אם גודל זה ידוע, ולחסם מלמעלה של אורך הטקסט אחרת. במקרה זה, ניצור טקסט בעזרת שימוש אחד בלבד במודל שפה סיבתי. חלקי העבודה:

- סקירת בינה מלאכותית – סקירה של נושאים בבינה מלאכותית שהבנתם קריטית על מנת להבין את הפתרון שפיתחתי.
- סקירת פייטון – סקירה של נושאים כללים במדעי המחשב וספציפיים לשפת פייטון שהשתמשתי בהם בפיתוח התוכנה. הבנת נושאים אלו עוזרת להבין את התוכנה 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 ואת הקוד המלא של העבודה. שימו לב שיחולו שינויים בקוד לאחר הגשת העבודה ולכן מצורפים לעבודה חלקים מרכזיים בקוד.

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 בהנחיה עצמית – Self-Supervised Learning:

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

לדוגמה – מודל שמקבל טקסט וחוזר את המילה החסרה, מודל שמוצא את החלק החסר בתמונה, מודל שמתקן טעויות בטבלה...

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 טוקניזר 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 בטוקניזר.

לטוקניזר שתי פונקציות מרכזיות:

Encode: הטוקניזר 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 למחרוזות.

כל מודל שקולט ואז 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 יכול לגרום לתוצאות חסרות משמעות.

:Dummy/One Hot Encoding (Suits)[22]

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

ייצוג one hot הוא בעצם וקטור 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 :

1	0	0
0	1	0
0	0	1

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:

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

שהסיבוכיות שלה היא $O(1)$ ולא בהגדרת Cross Entropy.

קוד:

```
def Sparse Categorical Cross Entropy(p: Tensor[n], q: OneHotVector[n]) -> float:
    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 $d_k \times \text{vocab size}$ 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]

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

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

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

אחד השימושים שלה הוא חישוב ווקטור הסתברות מתוך ווקטור לוגיטי.

הווקטור שהפונקציה מחזירה הוא באותו גודל של הווקטור שהפונקציה מקבלת.

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

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

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 הלוגיט מחולק בטמפרטורה לפני ככה ש:

$$\text{scaled logits}_i = \text{logits}_i / T$$

$$\sigma_T(\text{logits}) = \text{probability vector}_i = \text{softmax}(\text{scaled logits})_i = \frac{e^{\text{logits}_i / T}}{\sum_j e^{\text{logits}_j / T}}$$

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 הלוגיט $[-1, 1]$ עם הטמפרטורות: 0.5, 1, 2:

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

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

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

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 לוגיט L

בה L_i הוא ווקטור הלוגיט 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 הלוגיט של הטוקן שבה 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:

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

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 $\text{spare cross entropy}$.

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 (ישנם residual connections מעל לכל בלוק).

הפלט של בלוק הדיקודר האחרון הוא הפלט של הדיקודר והוא מוכפל במטריצת שיכון משוחלפת ליצירת מטריצת לוגיט.

על מטריצת הלוגיט מופעלת פעולת SoftMax שיוצרת את מטריצת ההסתברות שבה משתמשים על מנת ליצור רצף של טוקנים.

המודל מקבל מטריצה של ייצוג רצף הטוקנים במקום 1 עד n כאשר כל טוקן הוא ווקטור one hot ומחזיר מטריצה של רצף תחזיות לטוקנים במקום 2 עד $n+1$ כאשר כל תחזית היא ווקטור לוגיט בו הקטגוריות הם הטוקנים באוצר המילים של הטוקנייזר.

ישנם מודלים שמחזירים מטריצת הסתברות ולא מטריצת לוגיט.

בעבודה אשתמש במושג מטריצת הסתברות לתאר מטריצה בה $P_{i,t}$ היא ההסתברות של הטוקן ה- t ברצף להיות ה- t הטוקן שהמזהה שלו הוא i בהתבסס על הטוקנים הקודמים (אפס עד t לא כולל t). ומטריצת לוגייט לתאר מטריצה הבנויה מווקטורי לוגייט באותה הצורה.

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 היא ווקטור השיכון המקומי של הטוקן במקום i .

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

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

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.

$$\text{Logit matrix} = \text{embedding matrix}^T \cdot \text{decoder output}$$

אינטואיציה:

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

הכפלה במטריצת שיכון היא תרגום של הסתברות לשיכון.

הכפלה במטריצת שיכון משוחלפת היא תרגום של שיכון להסתברות.

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

מסכת הסתכלות קדימה look ahead mask: (Vaswani et al.) [24]

היא מטריצה בגודל $seq_len \times seq_len$ 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:

$$look_ahead_mask_{i,j} = \begin{cases} 1 & \text{if } j > i \\ 0 & \text{if } j \leq i \end{cases}$$

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:

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

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 מציג 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.

השאלתה (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 $seq\ len \times seq\ len$ בה $DP_{i,j}$ הוא

תוצאת המכפלה הסקאלרית בין הווקטור של שמיצג את הטוקן במקום i בשאלתה (לאחר הטרנספורמציה) לווקטור שמיצג את הטוקן במקום 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 במיליארד ככה שכל ערך שהיה אחד במסכה המקורית הוא מינוס מיליארד במסכה המוכפלת וכל ערך שהיה אפס נשאר אפס ובאופן פורמלי :

$$1,000,000,000 \cdot look_ahead_mask_{i,j} = \begin{cases} 1,000,000,000 & \text{if } j > i \\ 0 & \text{if } j \leq i \end{cases}$$

את תוצאת המכפלה מחסרים ממטריצת הדמיון ככה שהדמיון בין טוקן לטוקן שמגיעים אחריו ברצף הוא מינוס אינסוף.

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.

Scores (before softmax)					Masked Scores (before softmax)			
0.11	0.00	0.81	0.79	Apply Attention Mask →	0.11	-inf	-inf	-inf
0.19	0.50	0.30	0.48		0.19	0.50	-inf	-inf
0.53	0.98	0.95	0.14		0.53	0.98	0.95	-inf
0.81	0.86	0.38	0.90		0.81	0.86	0.38	0.90

הפעולה : $ASM = softmax(MDP)$

המטריצה ASM היא טבלה בה $ASM_{i,j}$ הוא האיבר בעמודה i ובשורה j המיצג את ההשפעה של הטוקן במקום i על הטוקן במקום j ופעולת SoftMax מגרמלת 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.

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

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

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: [24](Vaswani et al.)

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 :

$$\text{num_heads} \times \text{seq_len} \times \left(\frac{d_k}{\text{num_head}}\right) \text{ ל } \text{Seq_len} \times d_k$$

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

$$\text{LHV_of_head_i} = \text{LHV}[i, :, :]$$

$$\text{PTV_of_head_i} = \text{PTV}[i, :, :]$$

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 :

$$\text{vector_size} = d_k / \text{num_heads}$$

$$\text{start_index} = i * \text{vector_size}$$

$$\text{end_index} = (i + 1) * \text{vector_size}$$

$$\text{vector_for_head_i} = \text{original_vector}[\text{start_index}:\text{end_index}]$$

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 (m) 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 הקלט ומבצעת עליה את הפעולה הליניארית :

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

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

$$\frac{\partial \text{ReLU}(x)}{\partial x} = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{if } x < 0 \\ 0 \text{ or } 1 & \text{if } x = 0 \end{cases}$$

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-Training [18]

היא שיטה לאימון מודלים שמטרתם ליצור טקסט בהינתן טקסט. בשיטה זו, מאמנים מודל שפה סיבתי מתחיל בסט נתונים גדול עם טקסטים כלליים שעליו המודל מתאמן על מנת ליצור הבנה כללית של שפה אנושית כתובה.

משימה זו היא משימת לימוד בהנחיה עצמית:

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

חלוקה של המשפט "robots must obey orders"

¹ הנגזרת של פונקציית ReLU איננה מוגדרת מתמטית בנקודה בה $x=0$ אך ספריות למידת מכונה מגדירות אותה ל 1 או ל 0-תלוי בספרייה.

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

robot	must	obey	orders	must
robot	must	obey	orders	obey
robot	must	obey	orders	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 – Greedy Sampling/Argmax:

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 הלוגייט 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 חזרתיות.
- 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 גבוהה 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 w 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 אקספוננציאלי.

חיפוש עץ עם אילוף 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 $\leq p$).

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 הלוגייט (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 $\theta=1.2$.

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:

$$Pre - Normalized BERT Score = PNS = \sum_{x_i \in x} \max_{y_j \in y} (x_i^T y_j)$$

$$Pre Scaled BERT Recall = \frac{PNS}{|x|}$$

$$Pre Scaled BERT Precision = \frac{PNS}{|y|}$$

$$Pre Scaled BERT F1 = 2 \frac{BERT * Precision}{BERT + Precision} = 2PNS \frac{|y| + |x|}{|y||x|}$$

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:

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

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:

(Guido van Rossum and Talin)[9]

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:

(Guido van Rossum and Talin) [9]

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:

(Rossum, pt.2)[21]

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: `__init__` - אתחול, `__str__` - ייצוג כמחזורת - `__eq__` 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) וגדול (בערמת מקסימום) מבניו. כתוצאה מכך, השורש הוא החולייה בעלת הערך הקטן (בערמת

מינימום) וגדול (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 וגדולים ביותר ברשימה שאורכה n בסיבוכיות זמן ריצה $O(n + k \log_2 n)$ בעזרת הפונקציות `nleagest` ו `nsmallest` 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 $\text{group size} - 1$ 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 ל group 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 שמגיעים לפניהם בפלט.

בסוף הדיקודר, יש לדיקודר מידע על הטוקן הראשון בפלט (כרגיל) וכן על $\text{group size} - 1$ הטוקנים שאחריו. כך האלגוריתם מוצא את ווקטורי הלוגיט של group size הטוקנים הבאים באמצעות קריאה אחת בלבד למודל שפה סיבתי.

ווקטורי הלוגיט משמשים את האלגוריתם על מנת למצוא את הטוקנים עצמם.

אינטואיציה:

יצירת הטוקן הבא (בקבוצות של טוקן אחד):

טוקן 1	טוקן 2	טוקן 3	טוקן 4	טוקן 5	Next token
בוקר					טוב
בוקר	טוב				לכולם
בוקר	טוב	לכולם			,
בוקר	טוב	לכולם	,		היום
בוקר	טוב	לכולם	,	היום	אנחנו

יצירת שני הטוקנים הבאים (בקבוצות של שני טוקנים):

טוקן 1	טוקן 2	טוקן 3	טוקן 4	טוקן 5	Next token
בוקר	טוב				לכולם
בוקר	טוב	טוקן ריפוד			,
בוקר	טוב	לכולם	,		היום
בוקר	טוב	לכולם	,	טוקן ריפוד	אנחנו

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

פיתוח התכונה

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

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

עמוד האלגוריתם במאגר - <https://pypi.org/project/grouped-sampling/1.0.2/>

ניתן להוריד את החבילה באמצעות הפקודה:

```
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 אבסטרקטית `__call__` שמקבלת מטריצת לוגייט ואת רצף הטוקנים שהשתמשו בהם על מנת ליצור את המטריצה ומחזיר מטריצת לוגייט חדשה.

למחלקה שתי מחלקות בנות:

`LogitScalingRepetitionPenalty` מחלקת את ווקטורי הלוגייט שמייצגים טוקן שהופיע ברצף בפרמטר θ .

`NoRepetitionPenalty` לא משנה את מטריצת הלוגייט כלל. היא מקבילה למחלקה `LogitScalingRepetitionPenalty` כאשר θ שווה אחד.

`repetition_penalty_factory` מקבלת את הפרמטר θ ומחזירה עצם מסוג `RepetitionPenaltyStrategy` בהתאם לערכו של θ .

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 1 – group size 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 הלוגיט של הטוקן שמגיע במקום i+m+1. הווקטור במקום 0 הוא ווקטור הלוגיט של הטוקן במיקום m+1 ברצף (הטוקן שבא לאחר סוף הקלט), הווקטור במקום 1 הוא ווקטור הלוגיט של הטוקן במקום m+2 וככה הלאה...

הפונקציה משתמשת בפונקציה __call__ של מודל שפה סיבתי שהסיבוכיות 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 הלוגיט מהפונקציה get_logit_mat אך הווקטורים בה הם ווקטור הסתברות ולא ווקטורי לוגיט.

הפונקציה אחראית על הקטנת הלוגיטים של הטוקן המיוחד המעיד על סוף טקסט במקרה בו מספר הטוקנים הרצוי ידוע מראש, היא אחראית על הקטנת הלוגיט של טוקנים שהופיעו בקלט באמצעות אסטרטגיית הענשה (RepetitionPenaltyStrategy) ועל המרת ווקטורי לוגיט 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 :

אסטרטגיות עיבוד ראשוני וסופי (pre_processing_strategy, post_processing_strategy) מסוג PreProcessor ו PostProcessor בהתאם, שם מודל (מחזרות), מודל עטוף (ModelWrapper) וטוקניזר 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 ושיעורתי שזה יפגע ביכולת למדוד את איכות הטקסטים בעזרת מדד ברט.

הפונקציה __call__ מקבלת פרומפט אחד או יותר כמחרוזות, תוספות (תחילית וסופית) לפרומפט, מספר ההשלמות השונות לכל פרומפט, האורך המקסימלי של כל השלמה, אסטרטגיית קטיעות (לטוקניזר) 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:

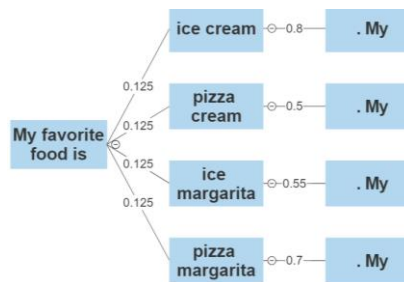
ההסתברות של קבוצה מוגדרת במכפלת ההסתברות הטוקנים בה. והסתברות של טקסט מוגדר כמכפלת ההסתברות הקבוצות.

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:

0.25	0.25	0.0001	0.0001	...
0.03	0.02	0.25	0.25	...

ובחירת הקבוצות תעשה באופן הבא:



הטקסט שיווצר הוא: "My favorite food is ice cream. My" מכיוון שהסתברותו הכי גבוהה. במימוש זה אין מגבלה הקובעת שם 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 גבוהה כאשר k הוא המשתנה top_k .
- top_p_sampling: משתמשת בדגימה 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 $1 + \frac{n}{g}$ 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:

0.25	0.25	0.0001	0.0001	...
0.04	0.02	0.25	0.25	...

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 0.02001 0.26 0.26 ...

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 הנתונים 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 : user, model, completion, 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 – המאחסנת 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 ה session 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 login_required (decorator).

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 login, register 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 completion.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 confest.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'ויפיטר 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 בתיקיה evaluation.

את התוצאות אספתי ריכוזי בעמוד <https://www.comet.com/yonikremer/grouped-sampling-evaluation/reports> באפליקציית הווב comet-ml שמשמשת לניהול ניסויים של מדעני נתונים.

תוצאות – חלק ראשון

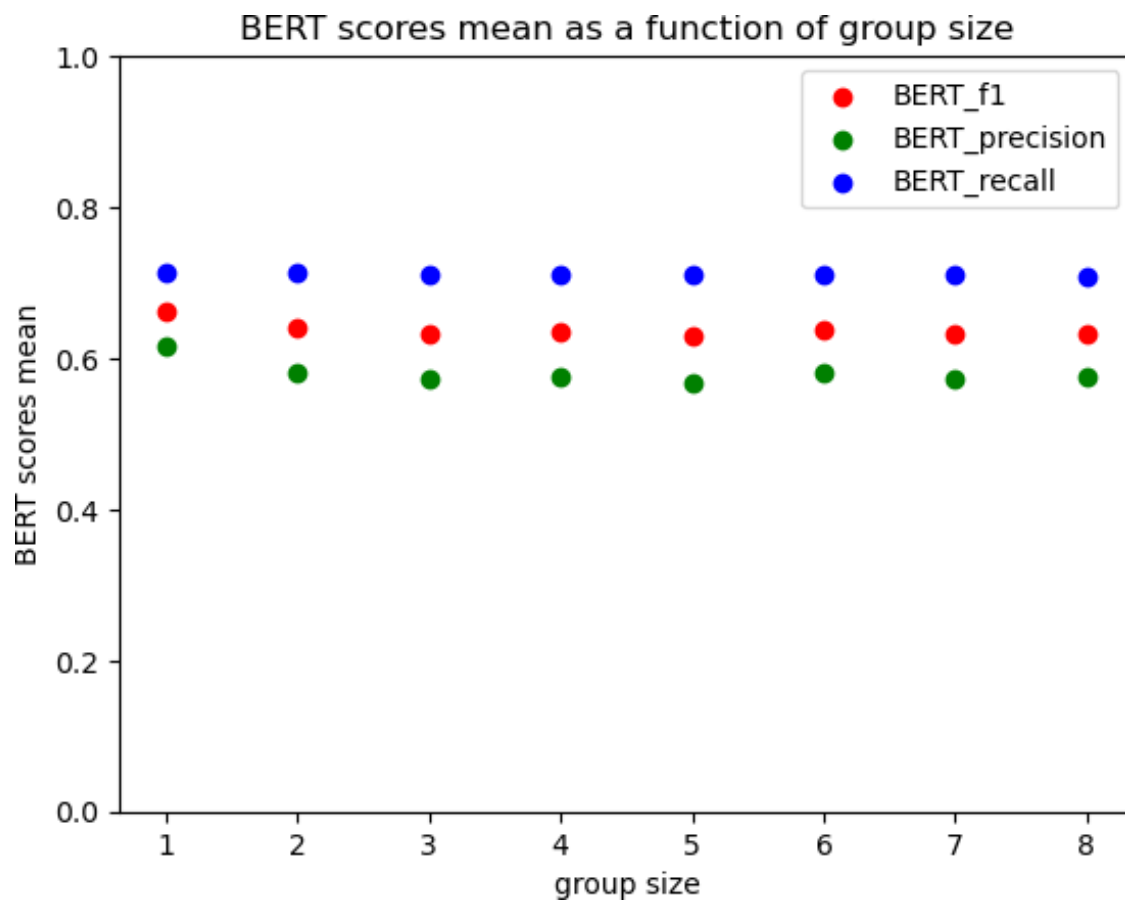
בניסויים הראשוניים הבנתי שבמשימת תרגום אני לא יודע מה יהיה אורך הפלט הרצוי אז שיניתי את הקוד של האלגוריתם ליצירת טקסט ככה שהטקסט יגמר כשהאלגוריתם יצור טוקן מיוחד שקיים בכל מודלי השפה שבא בסוף כל טקסט.

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 שניים מהקלט. אורך זה בא מתוך סט הנתונים שלי – היחס המקסימלי בין האורכים של צמד טקסטים הוא כמעט 2.

אלו התוצאות שקיבלתי:

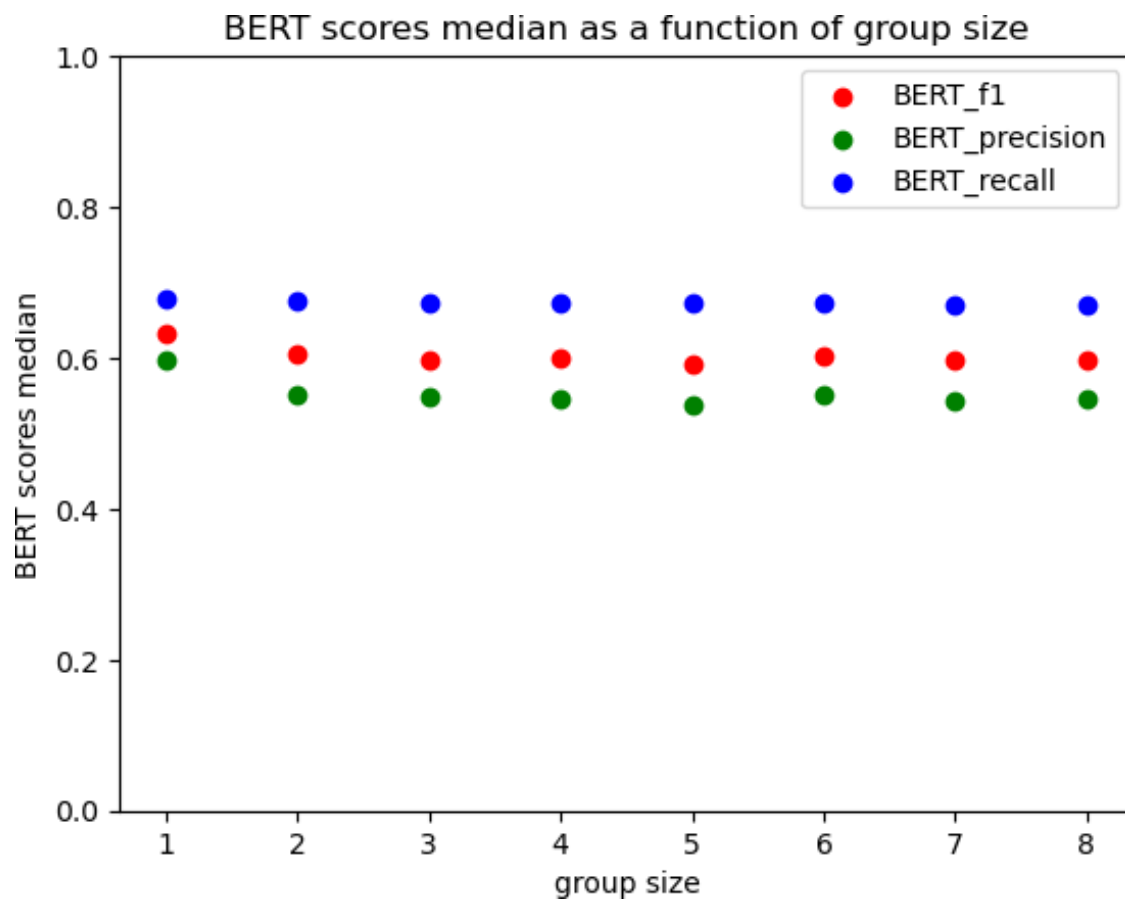
ממוצע:

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



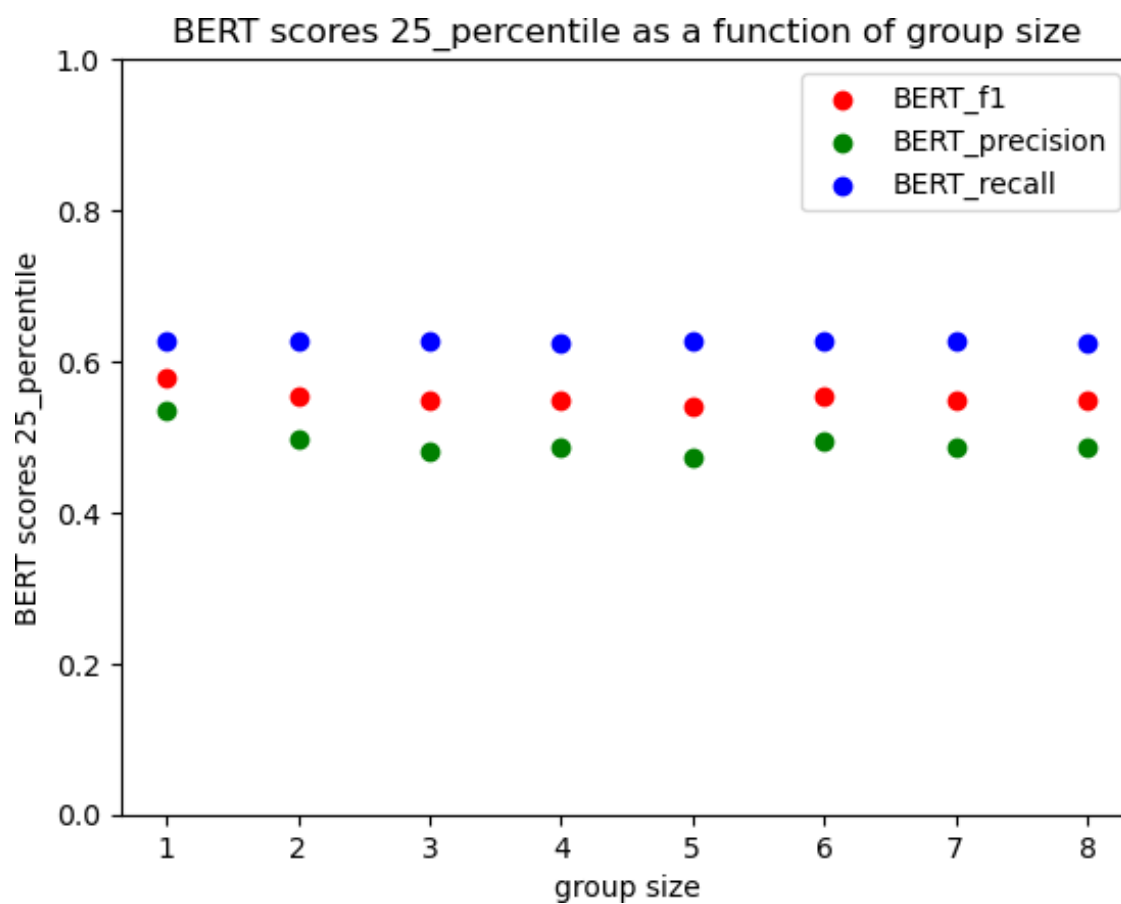
חציון:

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



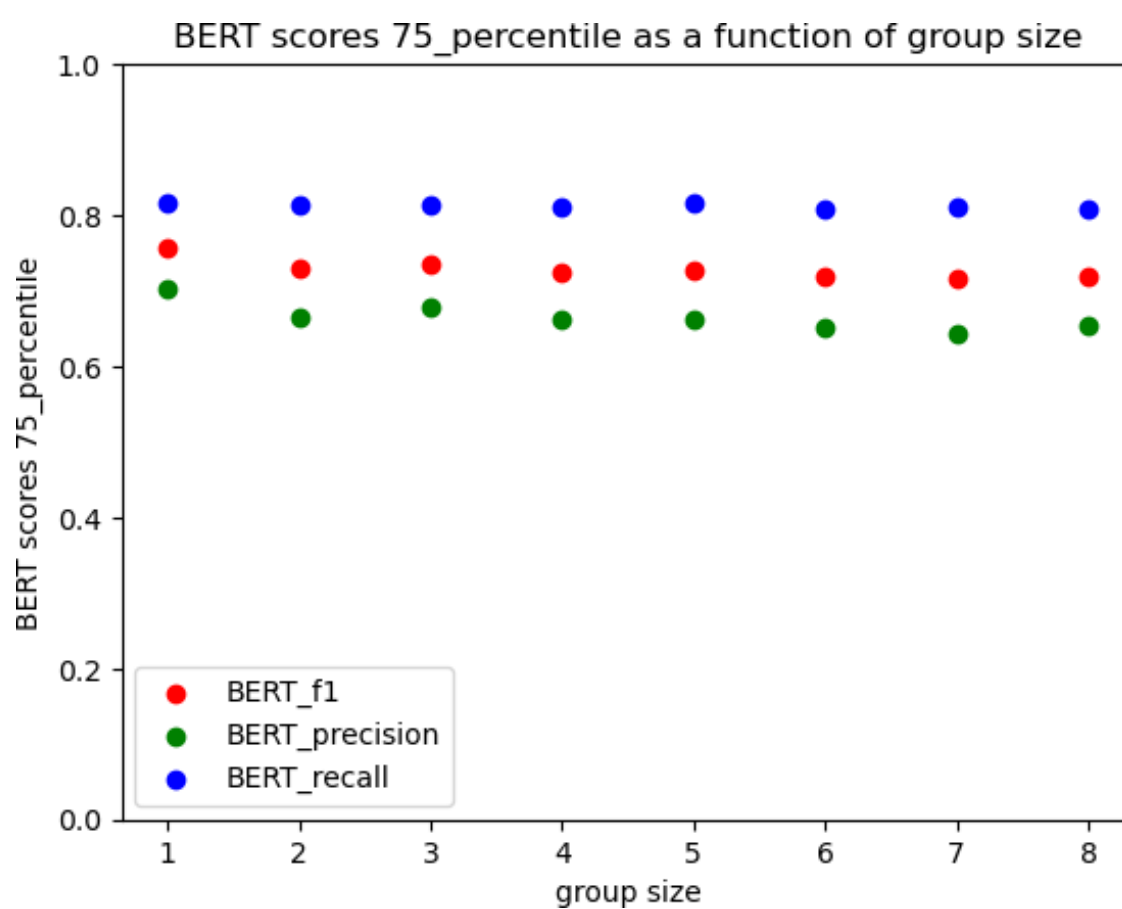
אחוזון 25 :

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

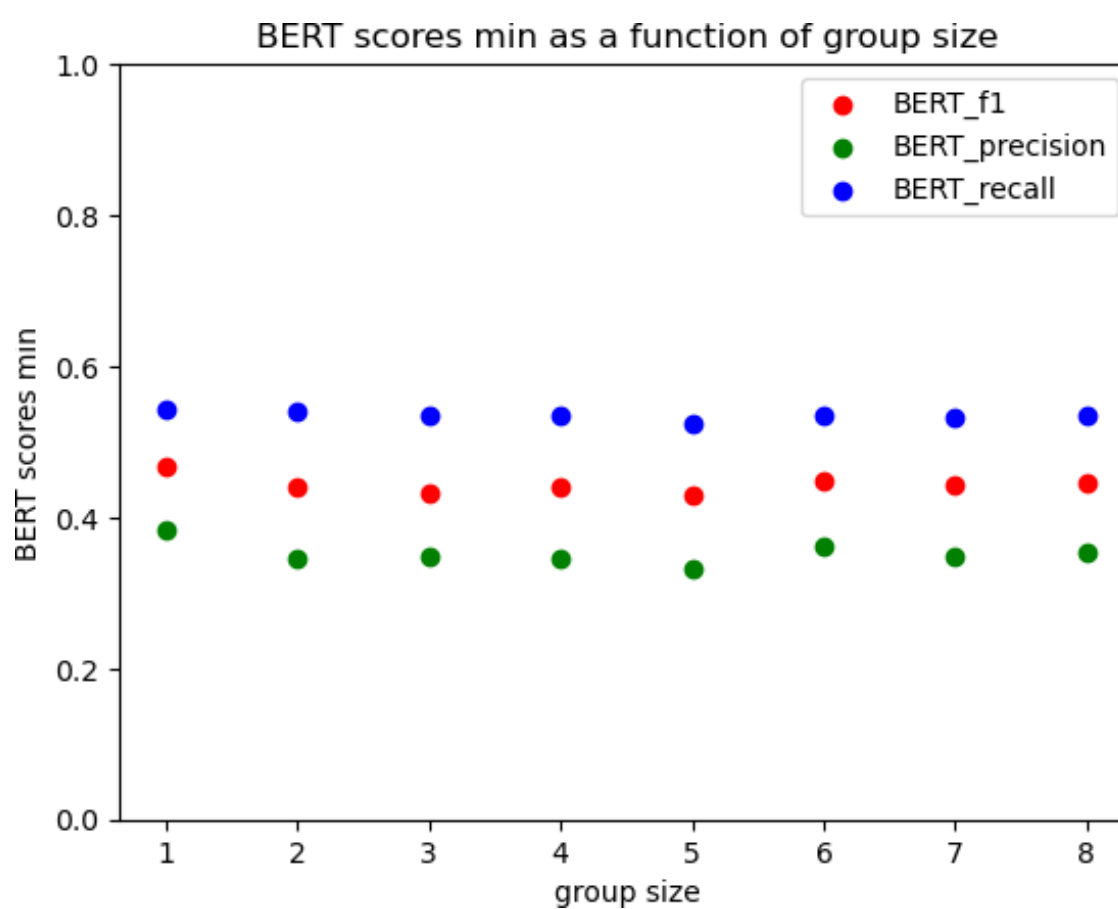


אחוזון 75 :

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

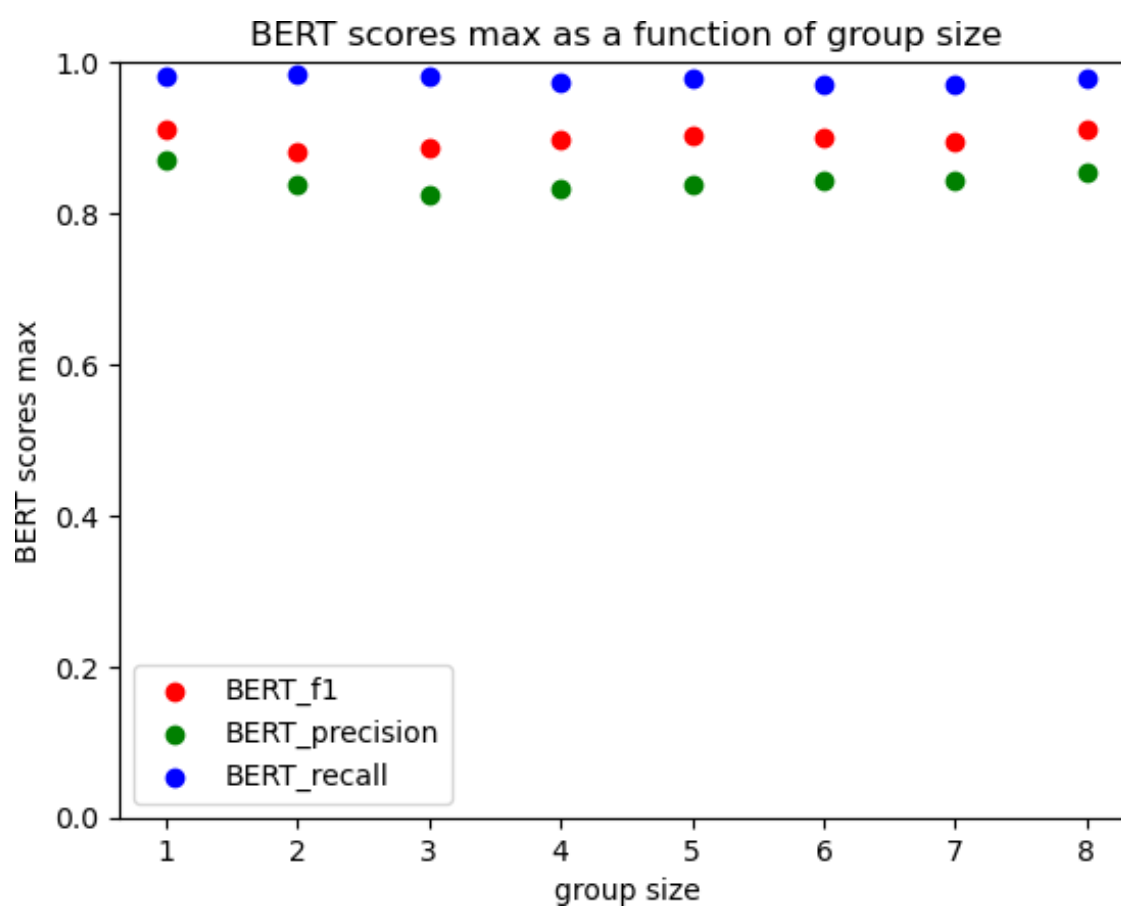


BERT_recall	BERT_precision	BERT_f1	group_size
0.542	0.383	0.468	1
0.541	0.346	0.44	2
0.536	0.348	0.433	3
0.535	0.345	0.44	4
0.525	0.331	0.43	5
0.535	0.362	0.449	6
0.532	0.348	0.444	7
0.536	0.353	0.445	8



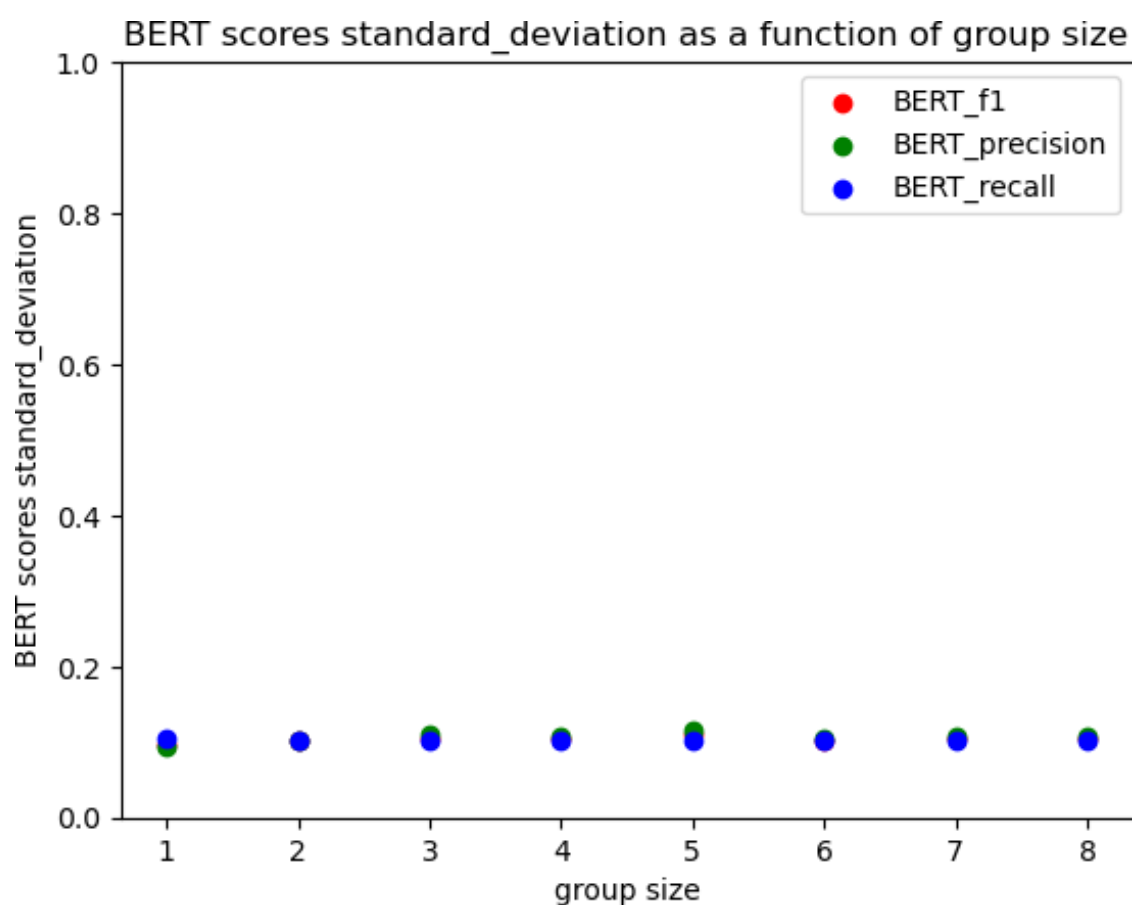
מקסימום :

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 :

BERT_recall	BERT_precision	BERT_f1	group_size
0.104	0.093	0.097	1
0.103	0.103	0.102	2
0.102	0.11	0.106	3
0.102	0.108	0.105	4
0.103	0.115	0.109	5
0.101	0.105	0.102	6
0.102	0.108	0.105	7
0.102	0.108	0.105	8



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 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.

$$Pre\ Scaled\ BERT\ Recall = \frac{PNS}{|x|}$$

$$Pre\ Scaled\ BERT\ Precision = \frac{PNS}{|y|}$$

ולכן אם $Recall > Precision$

נחלק את הנוסחאות של המדדים ב PNS (ידוע ש $PNS > 0$) 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 :

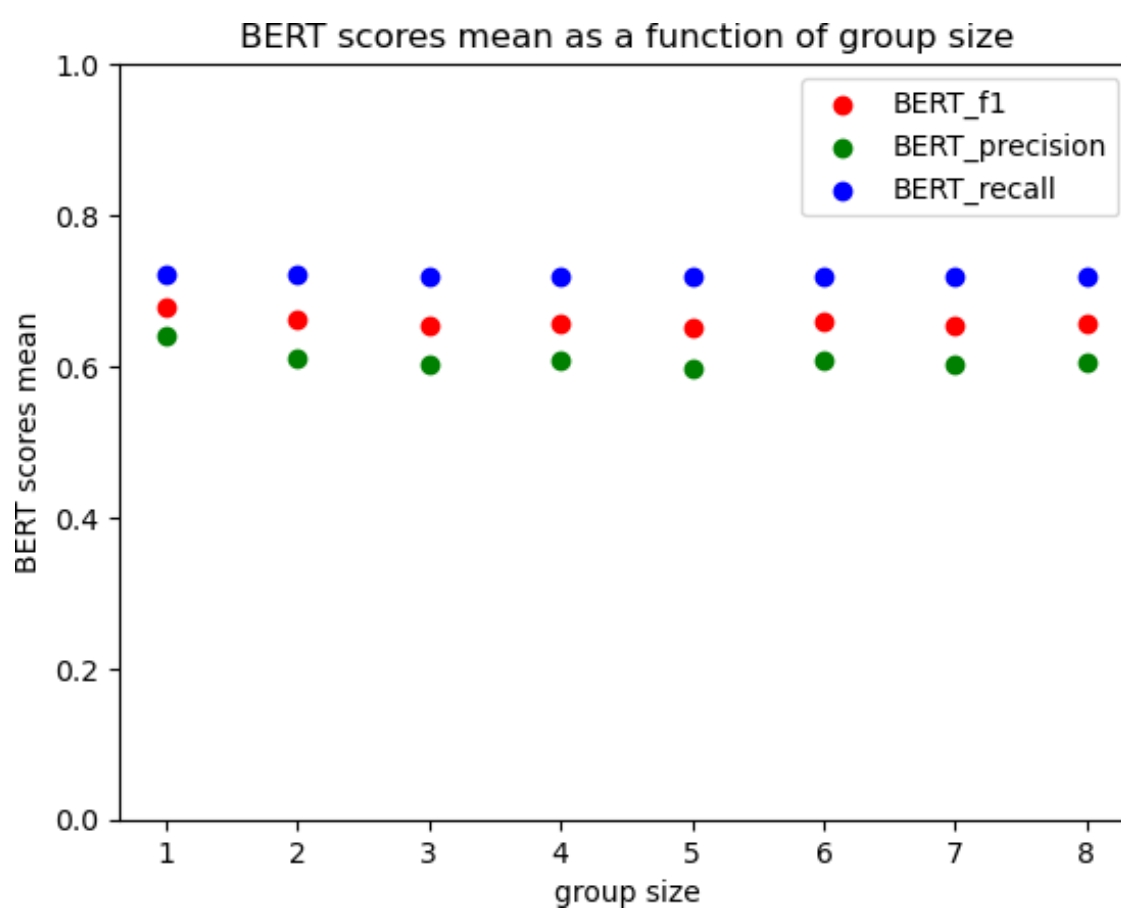
$|y| > |x|$ כלומר אורך הטקסט שהאלגוריתם יוצר גדול מאורך הטקסט הרצוי.

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.

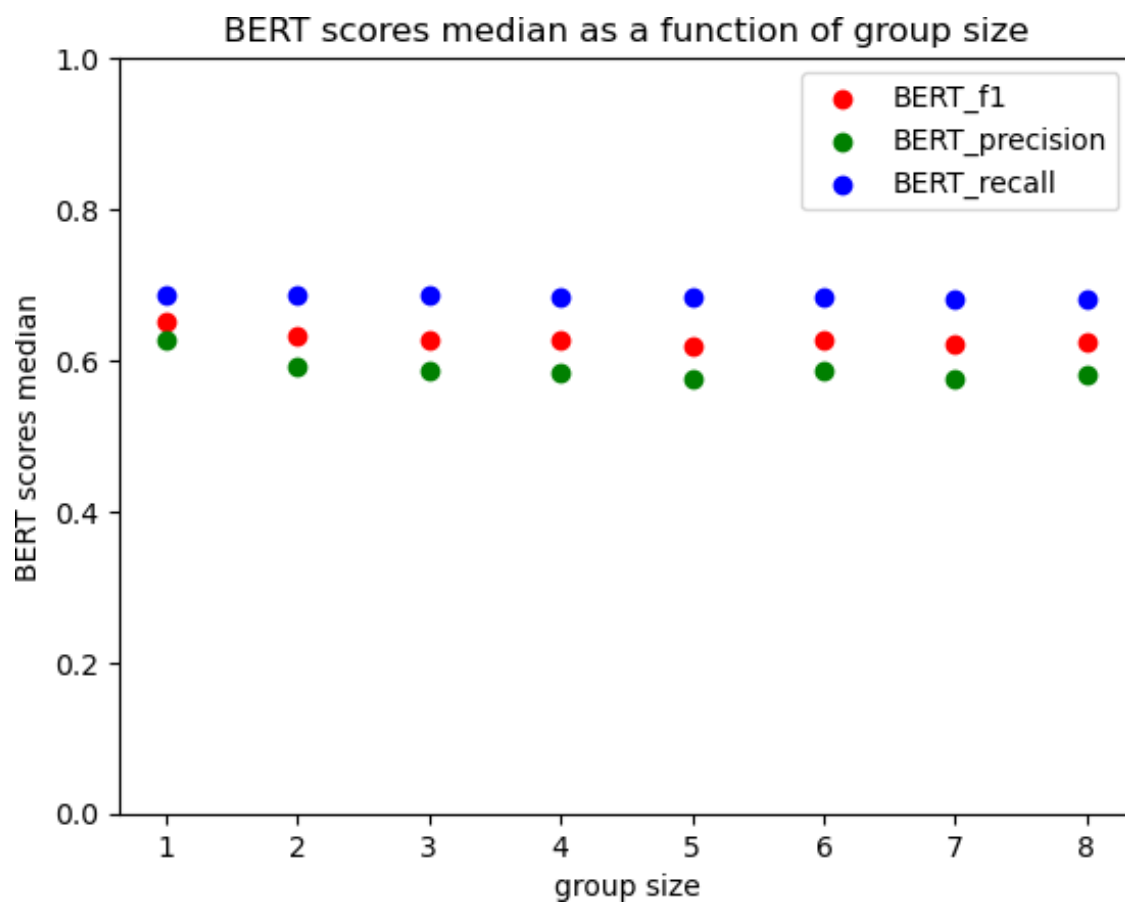
מאחר ששיניתי פרמטר שעשוי להשפיע על איכות הטקסטים, התחלתי סדרת ניסויים חדשה עם הפרמטר החדש.

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

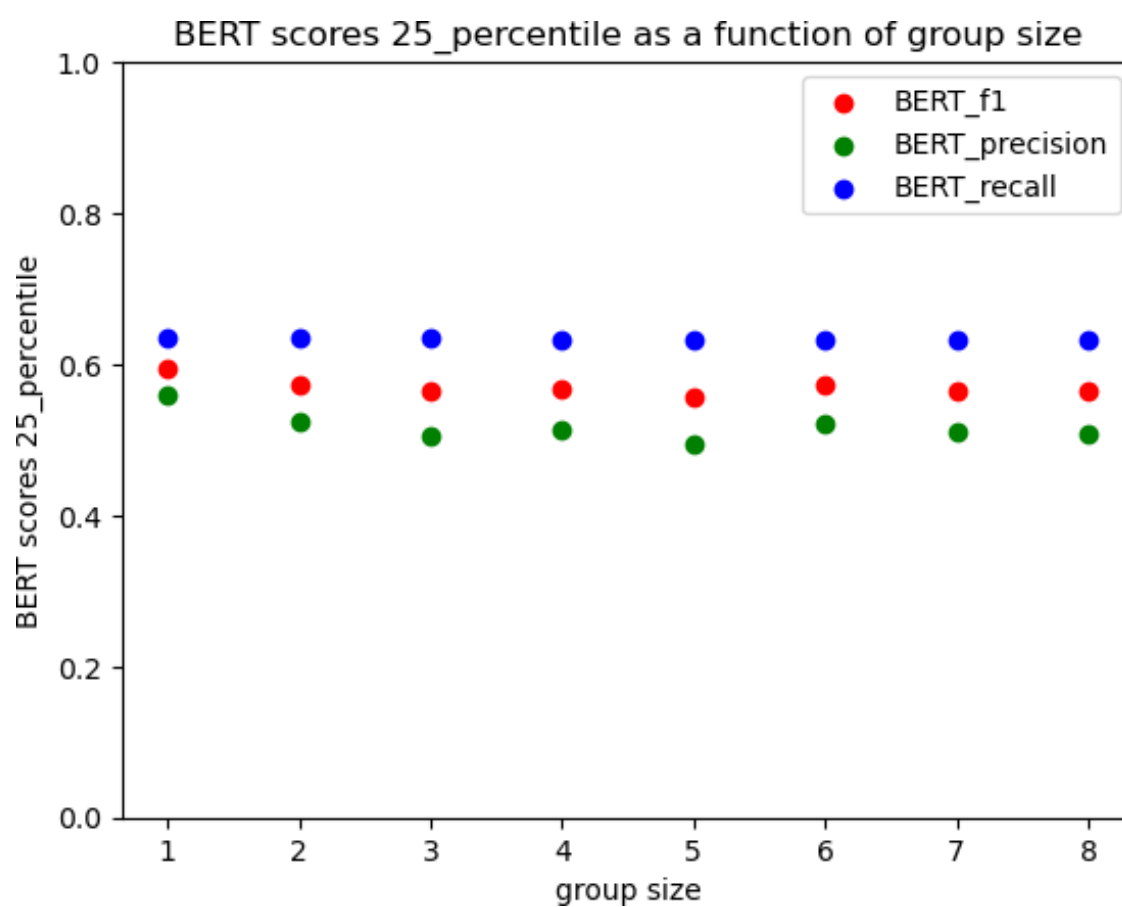


חציון:

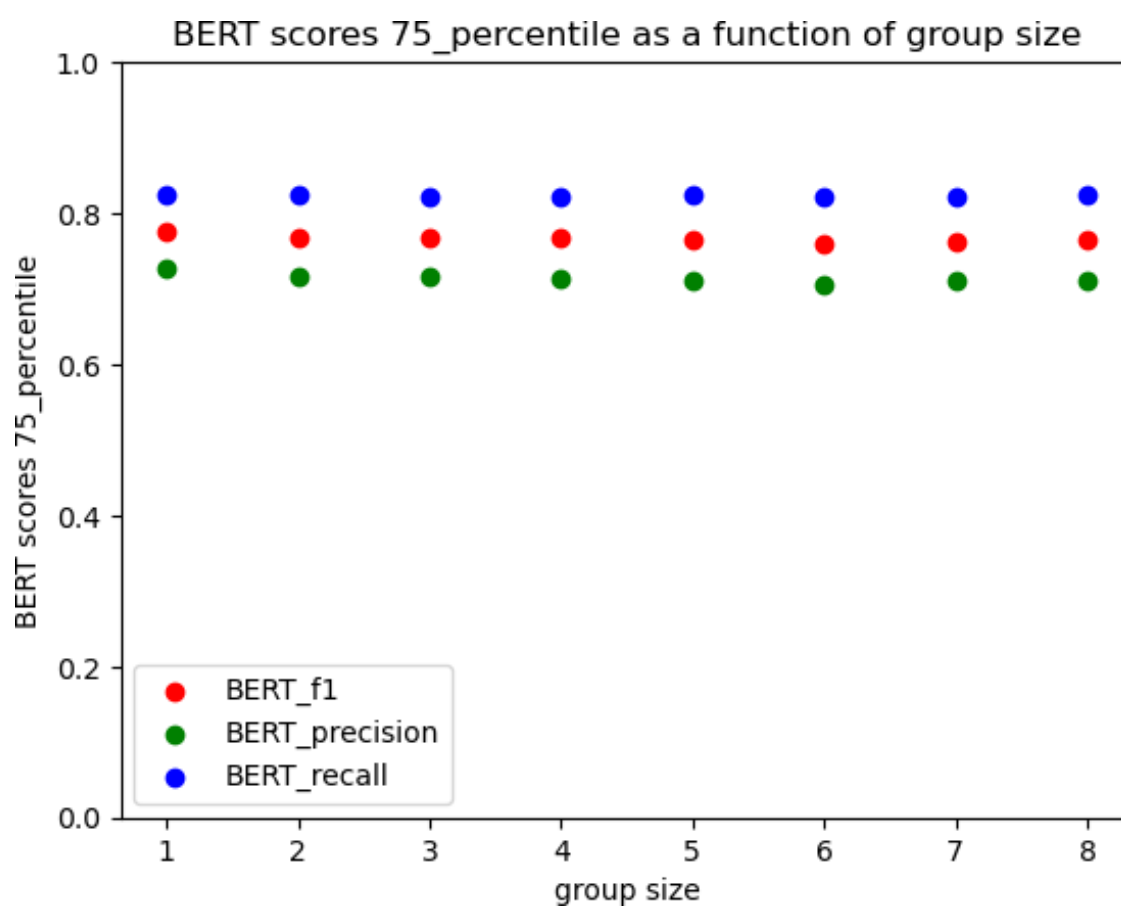
BERT_recall	BERT_precision	BERT_f1	group_size
0.687	0.627	0.652	1
0.687	0.591	0.633	2
0.685	0.587	0.627	3
0.683	0.583	0.628	4
0.683	0.576	0.62	5
0.683	0.586	0.628	6
0.682	0.575	0.621	7
0.682	0.581	0.624	8



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

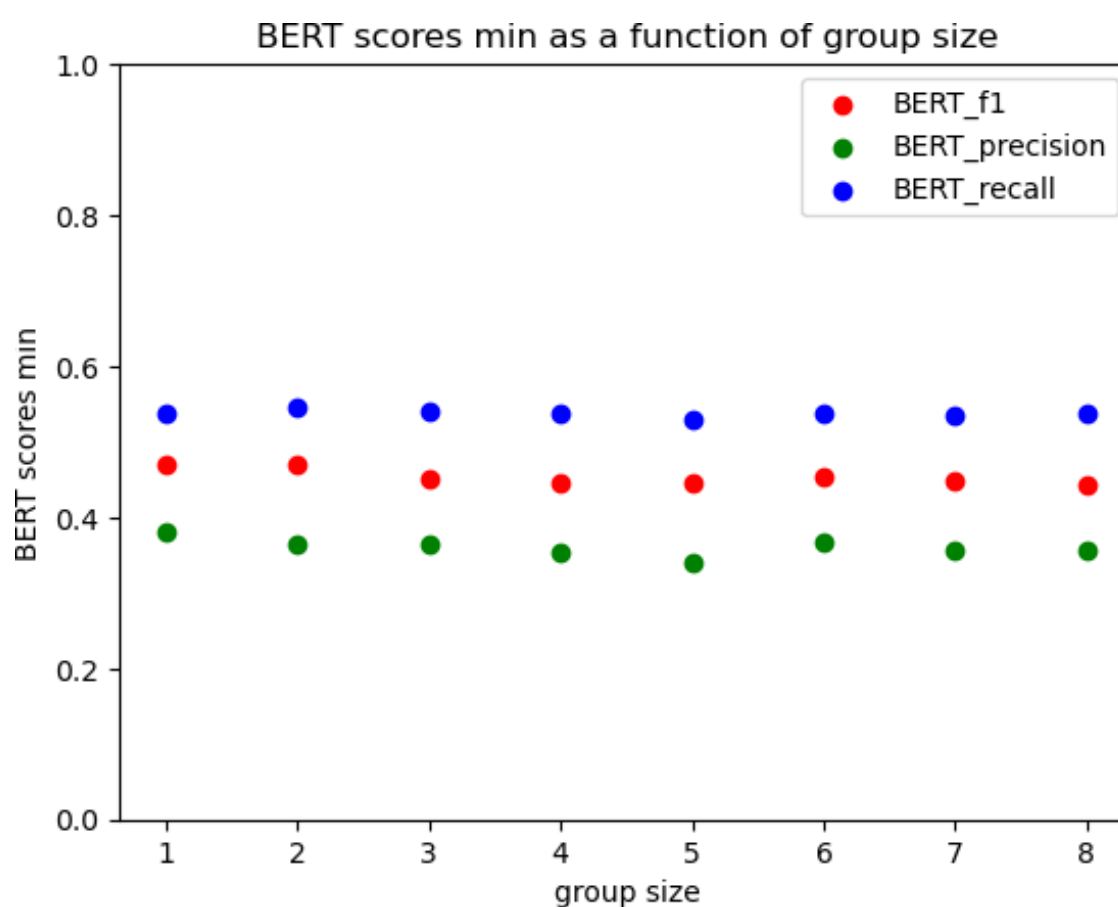


BERT_recall	BERT_precision	BERT_f1	group_size
0.824	0.728	0.774	1
0.824	0.715	0.767	2
0.822	0.717	0.767	3
0.822	0.715	0.767	4
0.824	0.712	0.764	5
0.821	0.706	0.76	6
0.822	0.71	0.762	7
0.824	0.71	0.764	8



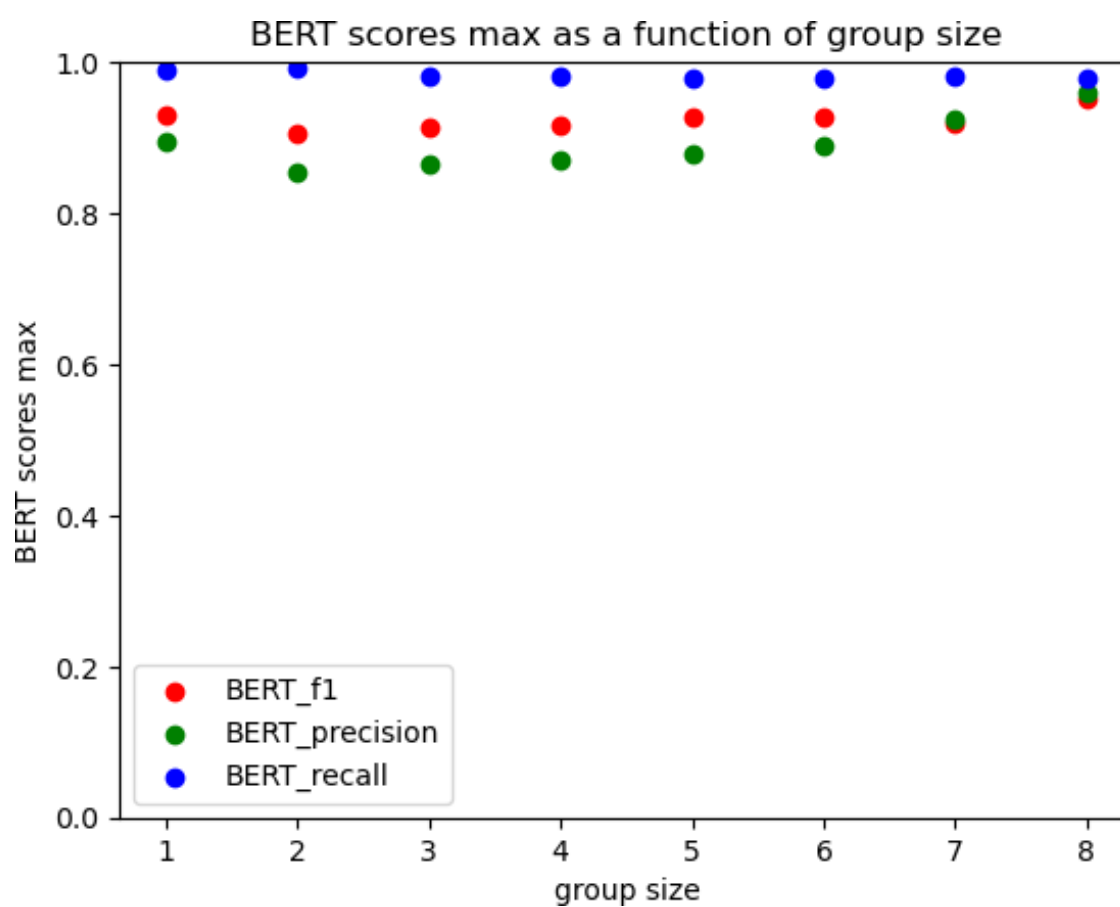
מינימום :

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



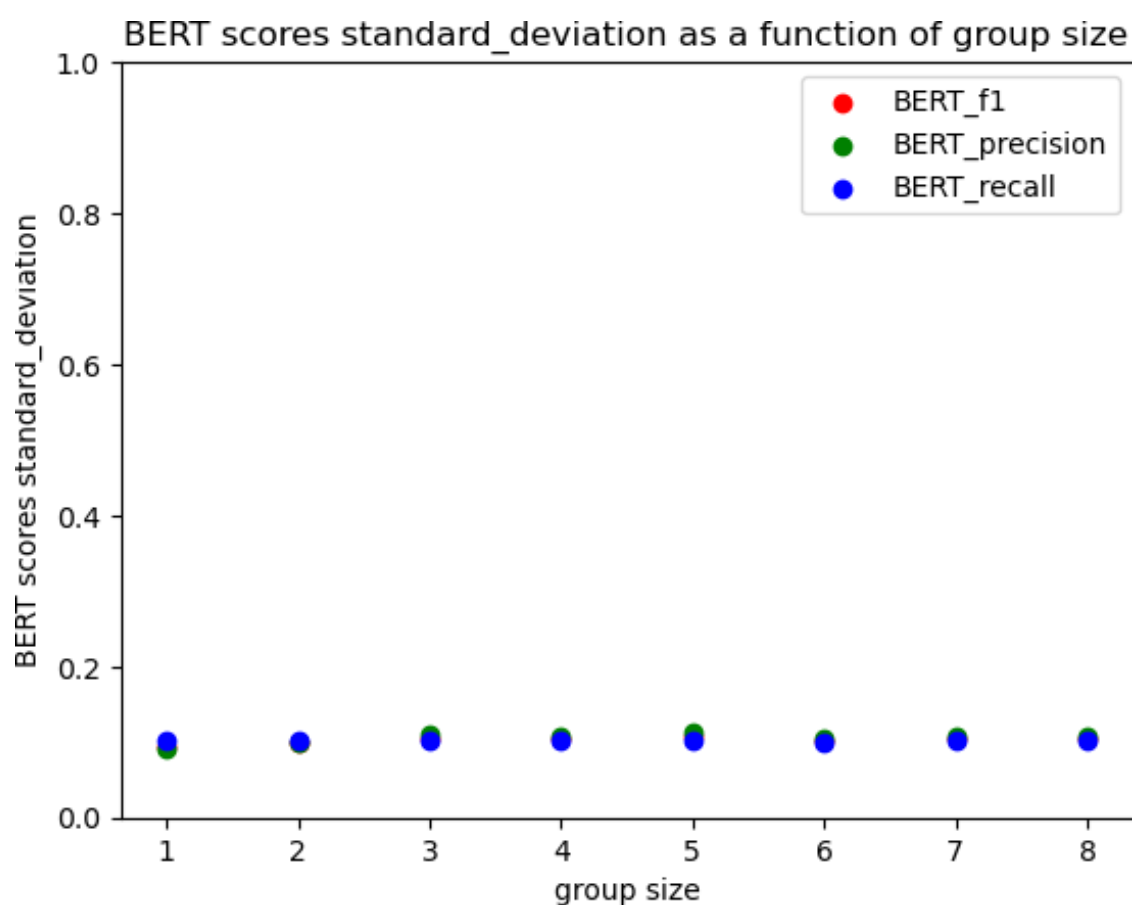
מקסימום :

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 שהסתברות של טוקן שכבר מופיע בטקסט תקבע מלאכותית ל 0 – השתמשתי בשיטת דגימה עם עונשים ובחרתי בפרמטר $\theta=1.2$ 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}).

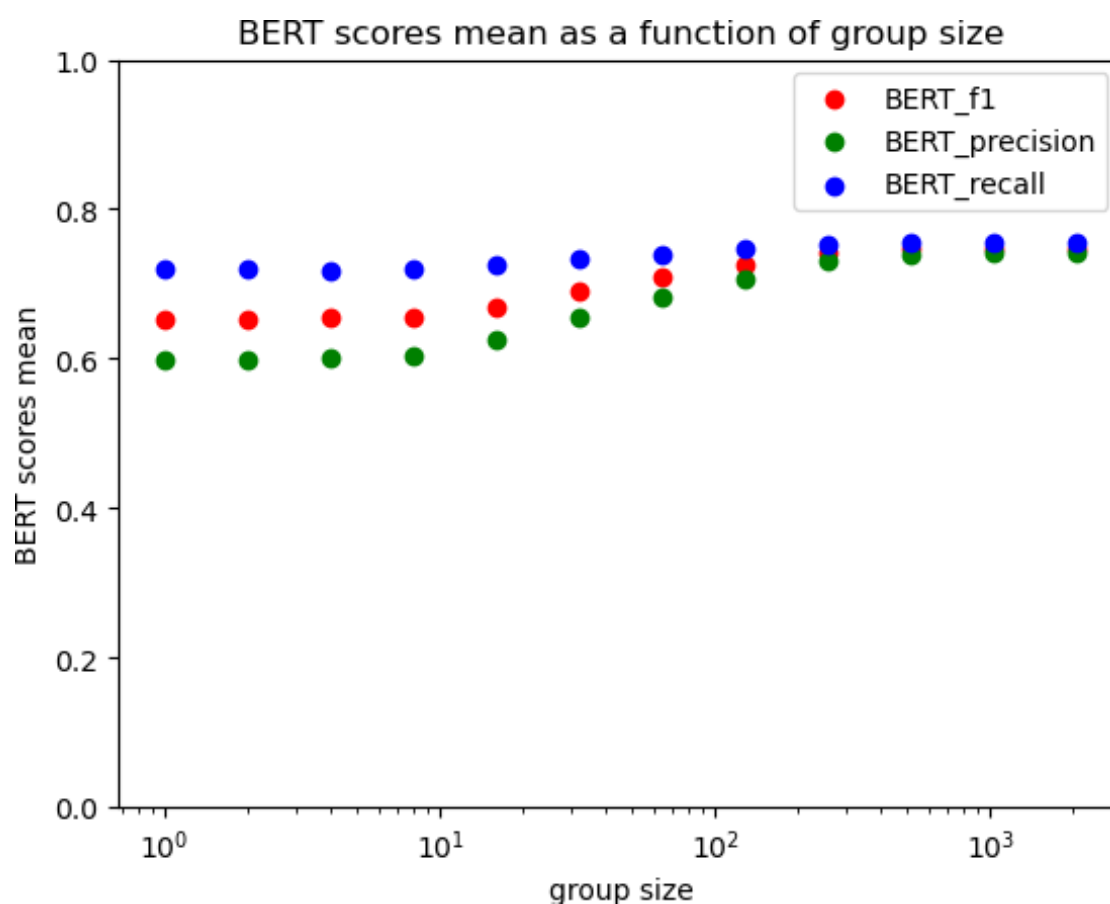
הפסקתי ב 2048 מכיוון שהניסוי עם גודל קבוצה של 4096 הגיע לשגיאת זיכרון – כלומר לא היה לי מספיק זיכרון במעבד הגרפי על מנת לבצע את הניסוי.

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

וצריכת הזיכרון במעבד הגרפי גדלה עם גודל הקלט למודל.

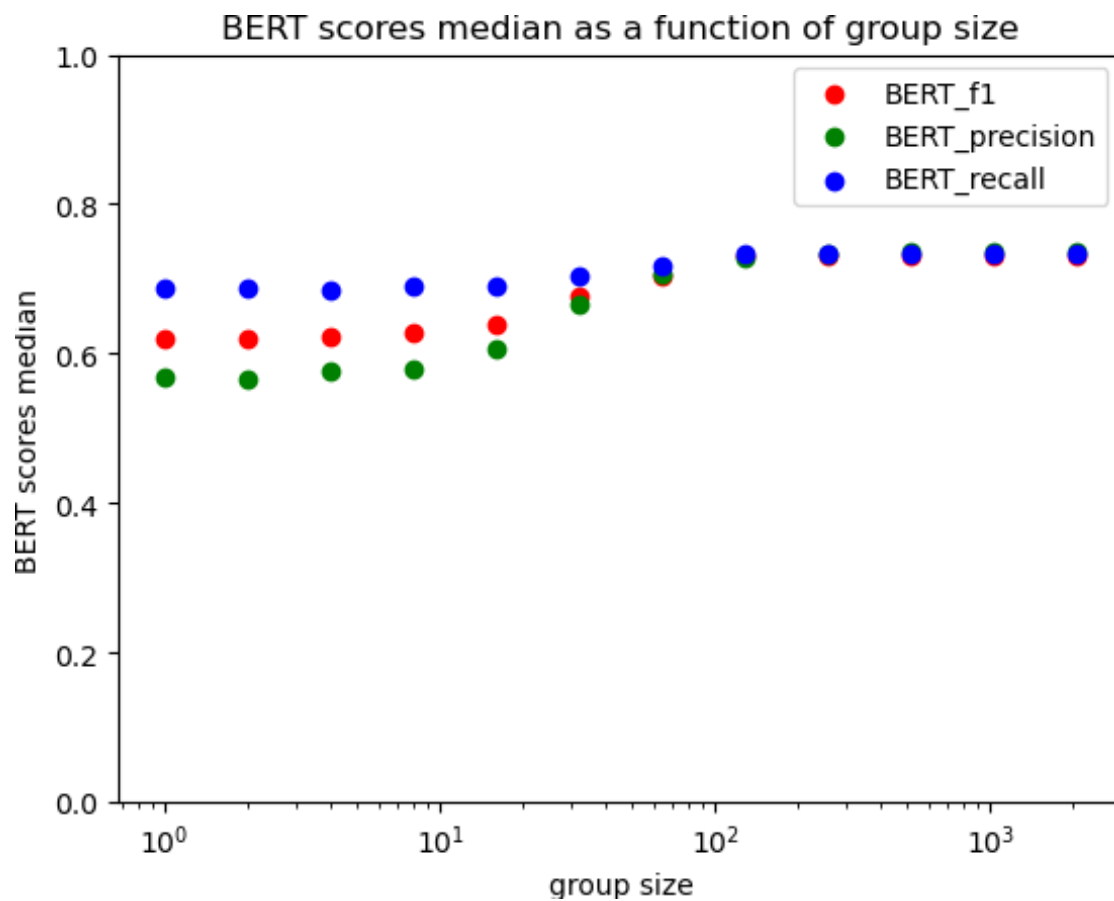
² • בגרפים בחלק השלישי גודל הקבוצה מוצג בכנה מידה לוגריתמית.

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

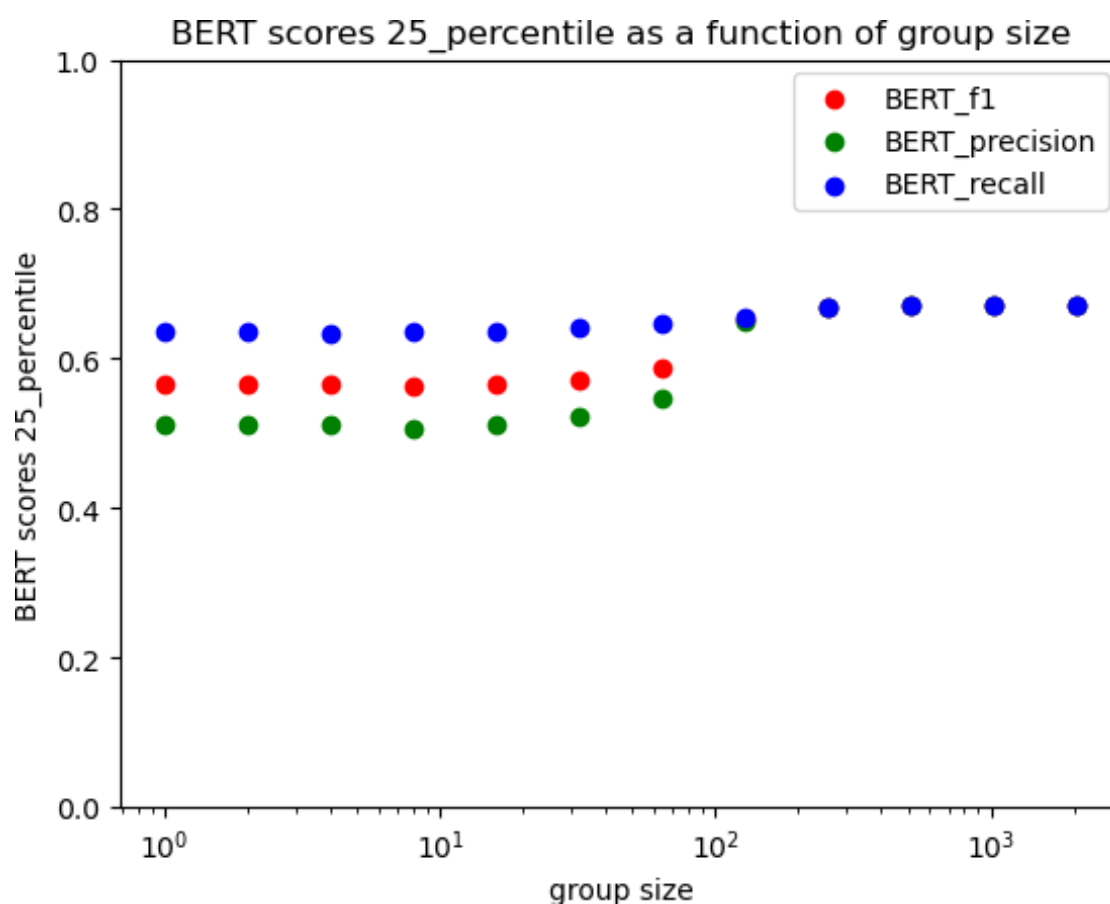


חציון:

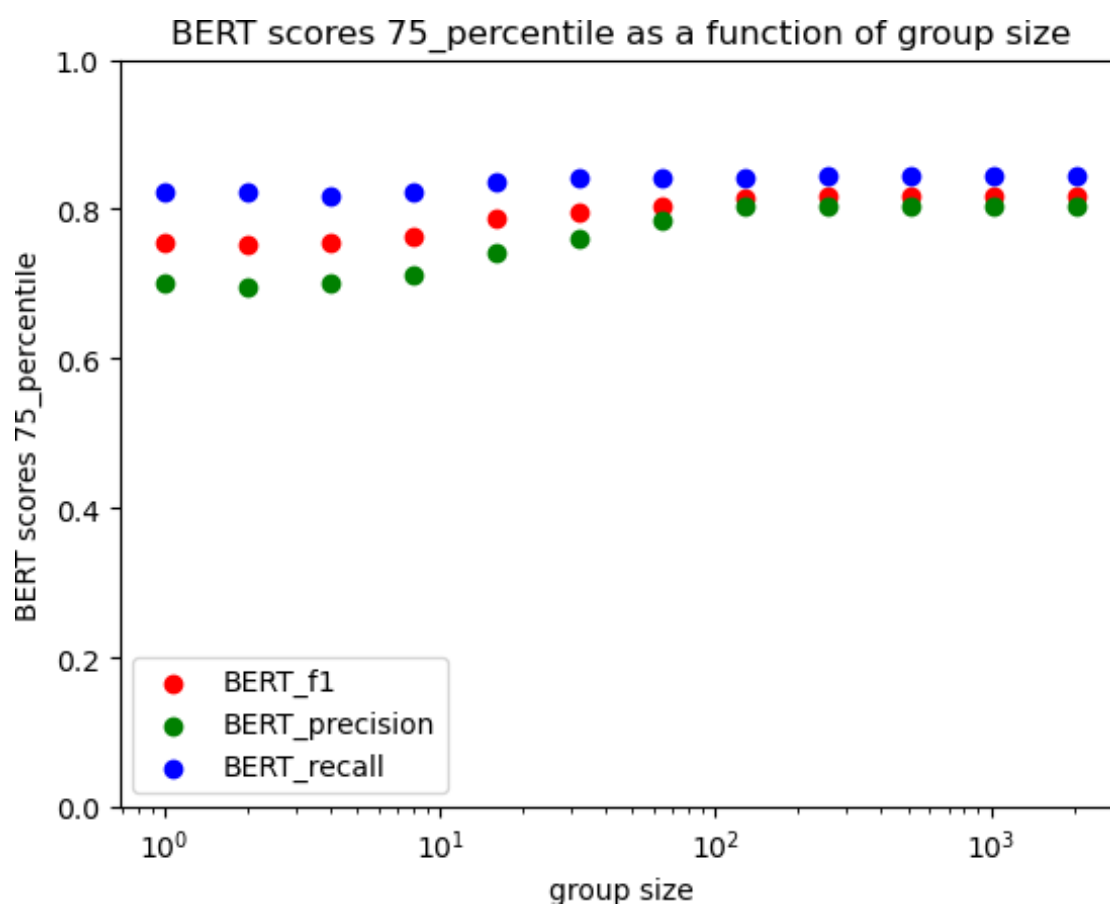
BERT_recall	BERT_precision	BERT_f1	group_size
0.688	0.567	0.621	1
0.687	0.566	0.619	2
0.685	0.577	0.624	4
0.69	0.581	0.629	8
0.691	0.606	0.64	16
0.704	0.665	0.677	32
0.718	0.707	0.704	64
0.733	0.729	0.731	128
0.734	0.734	0.732	256
0.734	0.735	0.732	512
0.734	0.735	0.732	1024
0.734	0.735	0.732	2048



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

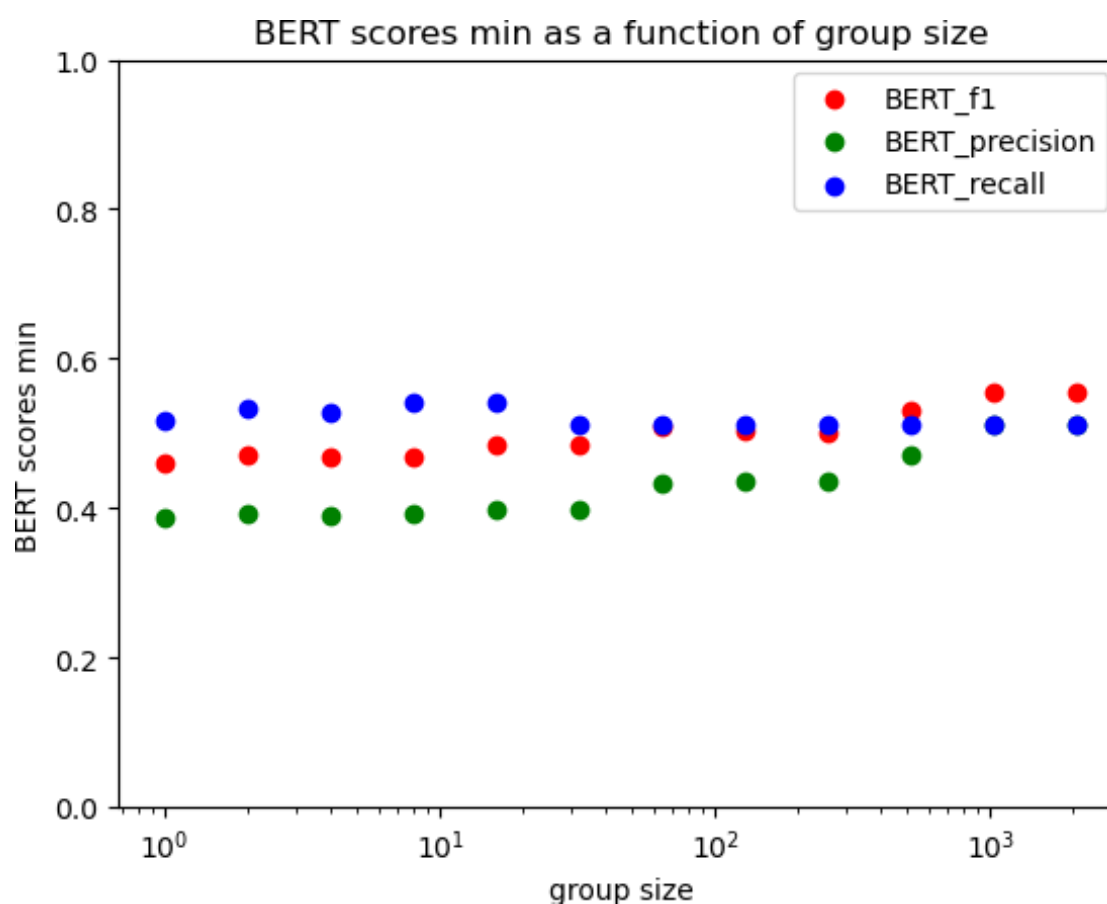


BERT_recall	BERT_precision	BERT_f1	group_size
0.822	0.7	0.755	1
0.823	0.695	0.753	2
0.818	0.702	0.755	4
0.822	0.711	0.763	8
0.836	0.741	0.787	16
0.841	0.76	0.797	32
0.842	0.784	0.805	64
0.842	0.803	0.813	128
0.845	0.803	0.817	256
0.845	0.803	0.817	512
0.845	0.803	0.817	1024
0.845	0.803	0.817	2048

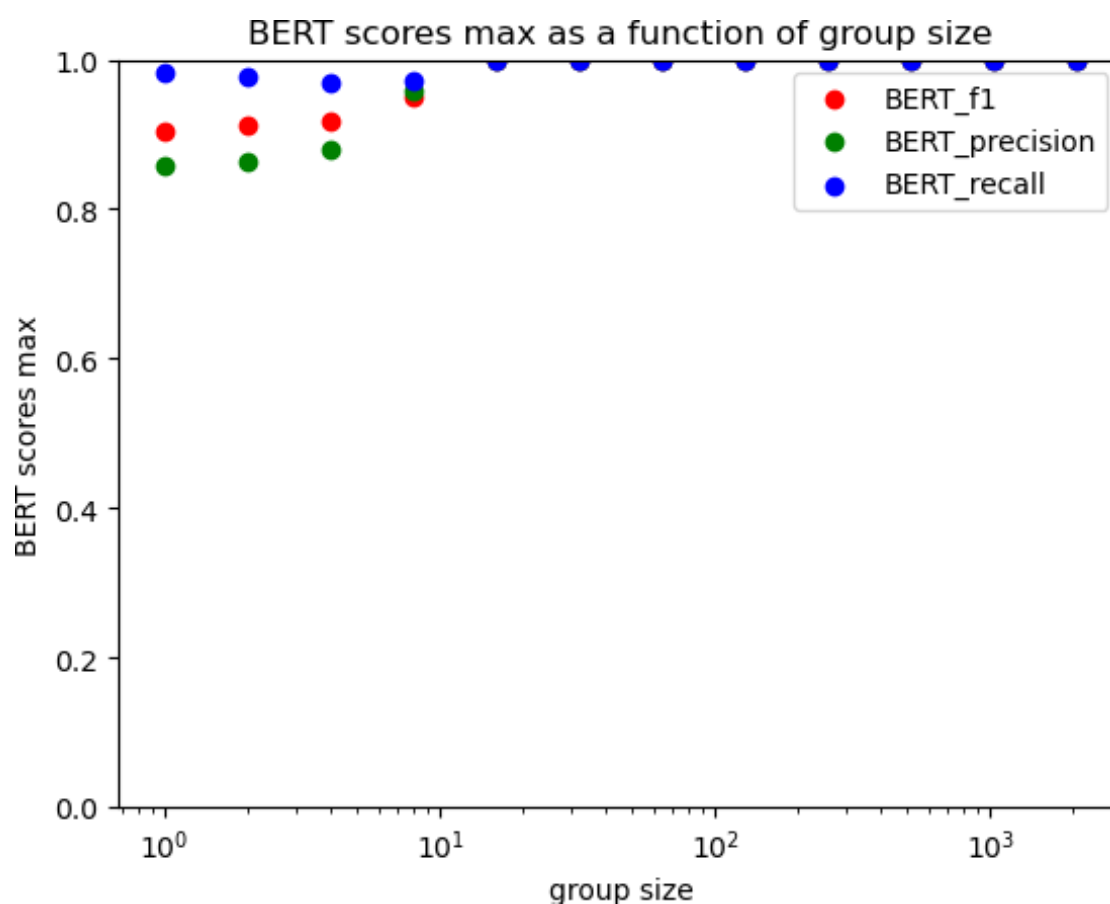


מינימום :

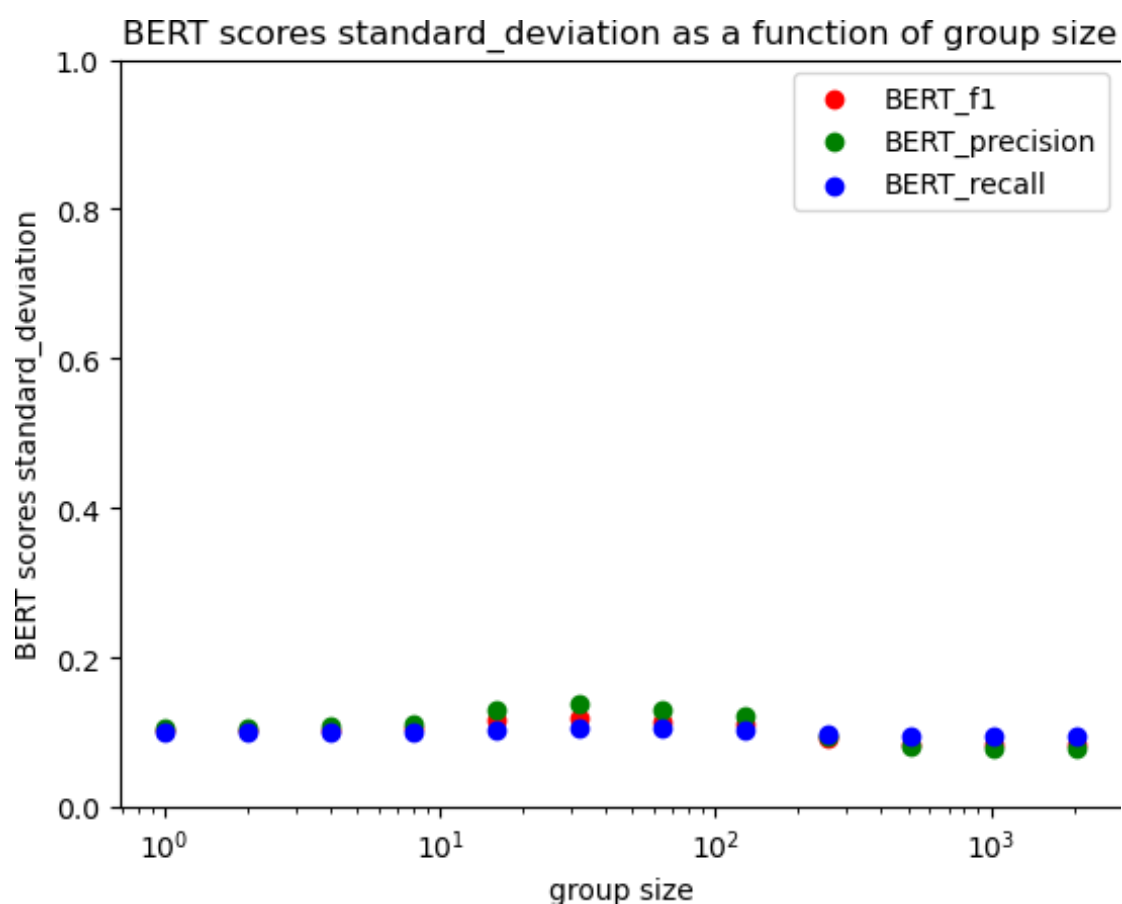
BERT_recall	BERT_precision	BERT_f1	group_size
0.518	0.388	0.46	1
0.534	0.394	0.472	2
0.529	0.391	0.469	4
0.542	0.393	0.469	8
0.541	0.398	0.484	16
0.511	0.399	0.484	32
0.511	0.432	0.508	64
0.511	0.437	0.502	128
0.511	0.436	0.502	256
0.511	0.471	0.529	512
0.511	0.511	0.554	1024
0.511	0.511	0.554	2048



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



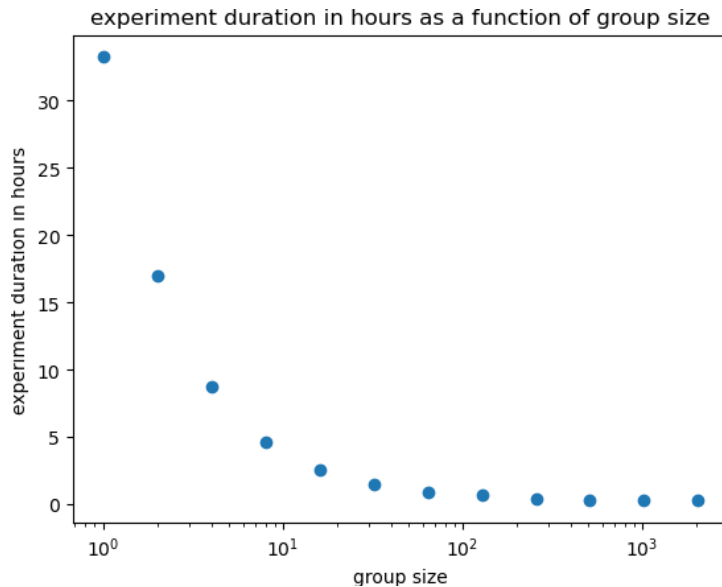
BERT_recall	BERT_precision	BERT_f1	group_size
0.101	0.105	0.102	1
0.101	0.106	0.103	2
0.101	0.107	0.104	4
0.1	0.11	0.105	8
0.103	0.129	0.116	16
0.107	0.137	0.121	32
0.105	0.129	0.115	64
0.103	0.122	0.11	128
0.097	0.096	0.093	256
0.095	0.082	0.085	512
0.095	0.079	0.084	1024
0.095	0.079	0.084	2048



- 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 ו 75 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 ל 2048 לא משפיעה על המדדים בכלל ומשפיעה לרעה על זמן הריצה.

בדיקה שערכתי הראתה שאורכי הטקסטים בסט הנתונים בו השתמשתי הם בין 6 ל 891 טוקנים. קבעתי שהאלגוריתם לא יוכל ליצור טקסטים ארוכים ביותר מ 25% 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:

The screenshot shows a web browser window with the title "Register - My Final Project". The address bar displays "127.0.0.1:5000/auth/register". The page content includes a header "My Final Project" and a sub-header "Register". Below these are two input fields: "Username" and "Password", each with a small eye icon for toggling visibility. A "Register" button is positioned at the bottom of the form. At the top right of the page, there are three links: "Register", "Log in", and "See all available modules". A footer at the bottom of the page contains the text "For any issues, please contact me at yoni.kremer@gmail.com" and a link to "Visit the project repository on GitHub". The Windows taskbar at the bottom shows the search bar, task view, and several application icons, with the system clock indicating 15:50 on 26-Dec-22.

עמוד ההתחברות

The screenshot shows a web browser window with the title "Log In - My Final Project". The address bar displays "127.0.0.1:5000/auth/login". The page content includes a header "My Final Project" and a sub-header "Log In". Below these are two input fields: "Username" and "Password", each with a small eye icon for toggling visibility. A "Log In" button is positioned at the bottom of the form. At the top right of the page, there are three links: "Register", "Log in", and "See all available modules". A footer at the bottom of the page contains the text "For any issues, please contact me at yoni.kremer@gmail.com" and a link to "Visit the project repository on GitHub". The Windows taskbar at the bottom shows the search bar, task view, and several application icons, with the system clock indicating 15:52 on 26-Dec-22.

עמוד השימוש באלגוריתם:

Create a Completion

See all available models

Enter the name of the model you choose
facebook/opt-125m

Enter the group size, positive integer
1

Enter the number of tokens of the completion, positive integer
4

Enter the number of different answers you want, positive integer
1

Enter top k, positive integer
1

Enter top p, $1 \geq \text{top p} \geq 0$
1

Enter a temperature, temperature > 0
1

Select generation algorithm>
True

Enter your prompt
I had so much fun in the

Complete
save

For any issues, please contact me at yoni.kremer@gmail.com
Visit the project repository on GitHub

עמוד הצפייה בטקסטים שהמודל יוצר:

Completions - My Final Project

See all past completion

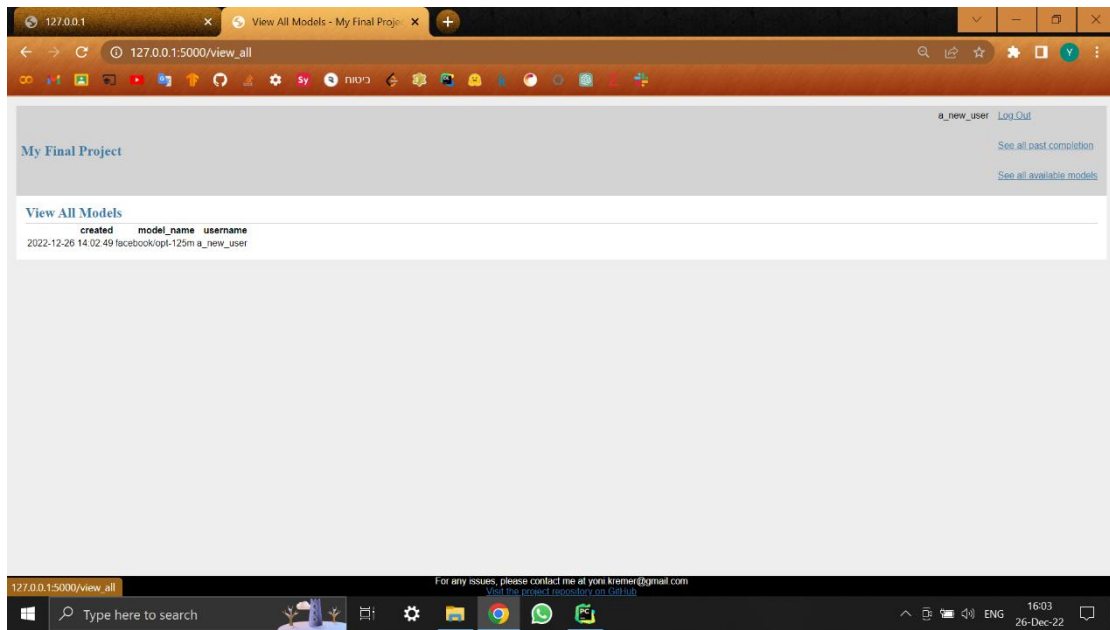
See all available models

Try it yourself

username	created	prompt	answer_num_tokens	model_name	group_size	generation_type	top_p	top_k	temperature
a_new_user	2022-12-26 14:02:49	I had so much fun in the	Dimensions 4	facebook/opt-125m	1	random	None	None	1.0

For any issues, please contact me at yoni.kremer@gmail.com
Visit the project repository on GitHub

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:

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 האלגוריתם שפיתחתי היא $O(n^2 + m^2)$.

סיבוכיות זמן הריצה של האלגוריתמים היעילים ביותר הקיימים כיום הוא $O(n^3 + nm^2)$.

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.

אני מתכנן להרחיב את הניסויים וכן לשנות את התכונה.

השינויים שאני מתכנן הם:

- הוספת האפשרות לקיבוץ (batching) – העברת מספר דוגמאות למודל במקביל.
- שימוש בגודל קבוצה שאינו קבוע.
- הרחבת התוכנה כך שתתמוך במודלי שפה סיבתיים נוספים.

אני מתכנן גם להוסיף את האלגוריתם לספריית הקוד הפתוח transformers.

³ בהנחה שסיבוכיות 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/0336dcbab05b9d5ad24f4333c7658a0e-Abstract.html>
5. Collections.Abc — Abstract Base Classes for Containers'. Python '
Documentation, The Python Software Foundation,
<https://docs.python.org/3/library/collections.abc.html>. Accessed 24 Dec. 2022
6. Eric V. Smith. 'PEP 557 – Data Classes'. Python Enhancement Proposals
(PEPs), <https://peps.python.org/pep-0557/>. Accessed 24 Dec. 2022
7. Fan, Angela, et al. 'Hierarchical Neural Story Generation'. Proceedings of the
56th Annual Meeting of the Association for Computational Linguistics
(Volume 1: Long Papers), Association for Computational Linguistics, 2018, pp.
.889–98. DOI.org (Crossref), <https://doi.org/10.18653/v1/P18-1082>

- Gehring, Jonas, et al. ‘Convolutional Sequence to Sequence Learning’. .8
 Proceedings of the 34th International Conference on Machine Learning,
 PMLR, 2017, pp. 1243–52. [proceedings.mlr.press,](https://proceedings.mlr.press/v70/gehring17a.html)
<https://proceedings.mlr.press/v70/gehring17a.html>
- Guido van Rossum and Talin. ‘PEP 3119 – Introducing Abstract Base Classes’. .9
 Python Enhancement Proposals (PEPs), <https://peps.python.org/pep-3119/>.
 .Accessed 24 Dec. 2022
- Heapq — Heap Queue Algorithm’. Python Documentation, The Python ‘ .10
 Software Foundation, <https://docs.python.org/3/library/heapq.html>. Accessed
 .24 Dec. 2022
- Holtzman, Ari, et al. The Curious Case of Neural Text Degeneration. arXiv, 14 .11
 .Feb. 2020. arXiv.org, <http://arxiv.org/abs/1904.09751>
- Keskar, Nitish Shirish, et al. CTRL: A Conditional Transformer Language .12
 Model for Controllable Generation. arXiv, 20 Sept. 2019. arXiv.org,
<https://doi.org/10.48550/arXiv.1909.05858>
- Kevin D. Smith, et al. ‘PEP 318 – Decorators for Functions and Methods’. .13
 Python Enhancement Proposals (PEPs), 2 Sept. 2004,
<https://peps.python.org/pep-0318>
- Liu, Peter J., et al. Generating Wikipedia by Summarizing Long Sequences. .14
 .arXiv, 30 Jan. 2018. arXiv.org, <https://doi.org/10.48550/arXiv.1801.10198>
- Mauro Cettolo, et al. WIT3 : Web Inventory of Transcribed and Translated .15
 .Talks. 2012, <https://aclanthology.org/2012.eamt-1.60.pdf>
- Mikolov, Tomas, et al. Efficient Estimation of Word Representations in Vector .16
 .Space. arXiv, 6 Sept. 2013. arXiv.org, <https://doi.org/10.48550/arXiv.1301.3781>

- nostalgebraist. Interpreting GPT: The Logit Lens. www.lesswrong.com,
<https://www.lesswrong.com/posts/AcKRB8wDpdAN6v6ru/interpreting-gpt-the-logit-lens>. Accessed 29 Dec. 2022 .17
- Papers with Code - Improving Language Understanding by Generative Pre-
Training. <https://paperswithcode.com/paper/improving-language-understanding-by>. Accessed 5 Jan. 2023 .18
- <https://paperswithcode.com/method/gpt>. Accessed 23 Dec. 2022 .--- .19
- 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> .20
- 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 .21
- 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> .22
- van Rossum, Guido, et al. ‘PEP 484 – Type Hints’. Python Enhancement
.Proposals (PEPs), <https://peps.python.org/pep-0484/>. Accessed 24 Dec. 2022 .23
- Vaswani, Ashish, et al. Attention Is All You Need. arXiv, 5 Dec. 2017. .24
.arXiv.org, <https://doi.org/10.48550/arXiv.1706.03762>
- 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 .25

- 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>

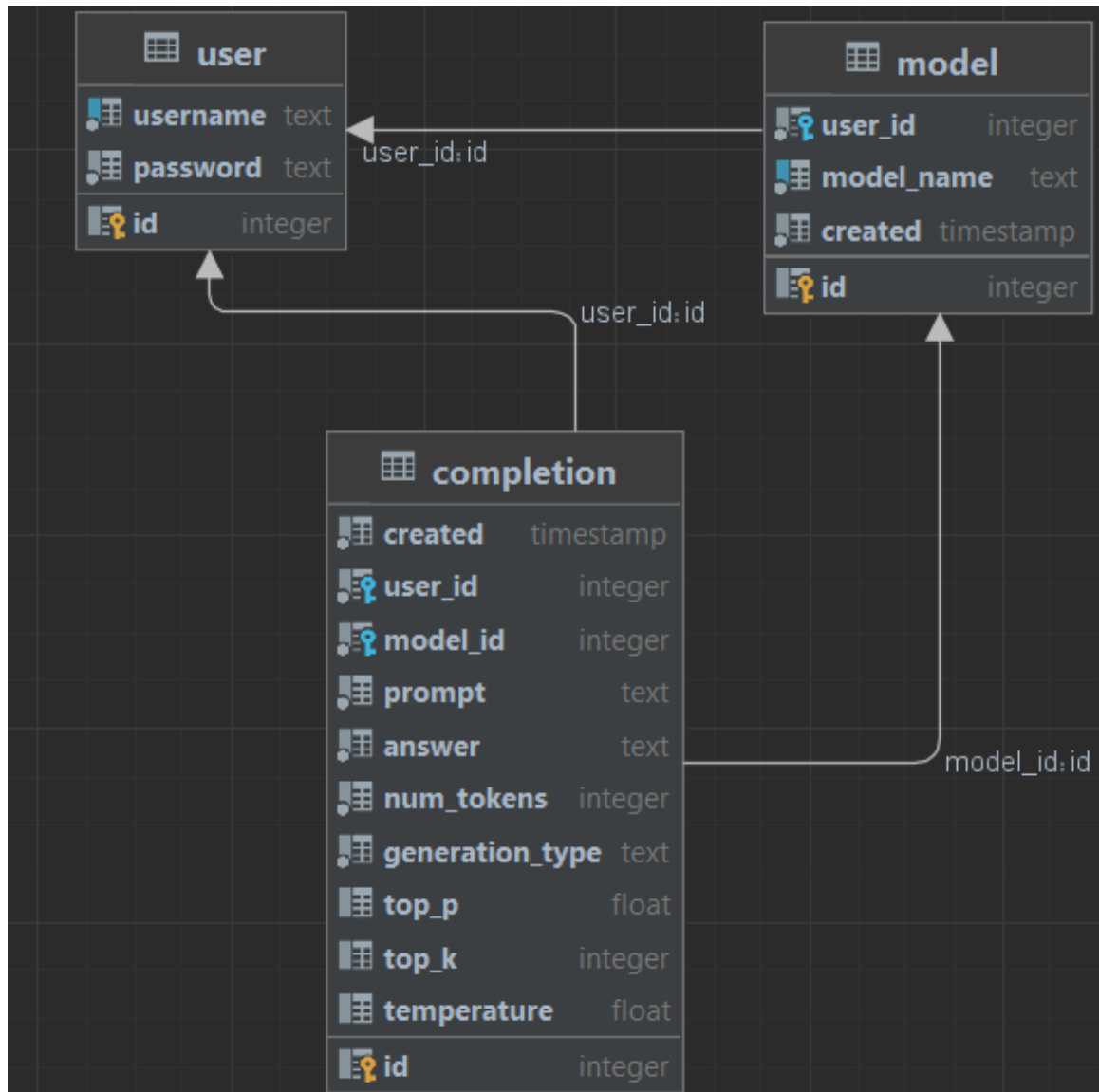
נספחים

קישור לעמוד הגיטהב של הפרויקט:

https://github.com/yonikremer/grouped_sampling

דיאגרמת בסיס הנתונים של אפליקציית הווב:

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



קוד האלגוריתם לדגימה בקבוצות: