

עבוד שפה טבעית (20225211) - תרגיל 4 חלק 1 (5 נק')

שלום לכולם!

בחלק הזה של תרגיל 4 נצבור קצת נסיון מעשי בעבודה עם מודלי שפה מסוג רובוטריק (transformer). נתמקד במודל מסוג **אוטורגרסיבי**, כלומר חוזה-פני-עתידי, כלומר מתבונן-רק-שמאלה. ספציפית, נסתכל על המודל GPT-2, גרסה מ-2019 של המודל של OpenAI. מבחינה אלגוריתמית, אין הרבה הבדל בינו לבין GPT-4.5 הנוכחי אלא רק מבחינת סדר-הגודל של כמות הפרמטרים וכמות הדאטא עליו אומן. מה שכן, זה לא מודל שאומן למטרות שיח ובינו לבין ג'יפטופוט (ChatGPT) כן יש הבדלים מהותיים.

בסיום העבודה, הגישו **קובץ pdf** הכולל את כל הפלטים מכל התאים. דרך אפשרית אחת להשיג קובץ pdf היא על-ידי הדפסת הדף (מתוך colab, לא מהדפדפן) ולבחור "מדפסת" ישירה ל-pdf.

נא לוודא שכל המידע הדרוש מופיע בתדפיס ה-pdf --- לא יתקבלו ערעורים על-בסיס חיתוך. נא לא לחרוג ממבנה התאים ולא לענות במקומות שאינם מוגדרים לכך. אין למחוק הערות שמסמנות את מיקומי קטעי הקוד שלכם ואין לערוך קוד שנמצא מחוץ לתחומים המוגדרים. סעיפים שחורגים מהנחיה זו לא ייבדקו.

ניתן להגיש בצוותים של עד שלוש. סטודנטים. לא יהיו הנחות לצוותים קטנים. אפשר לחלק את פתרון התרגיל בין חברי.ות הצוות כרצונכם. אין להסתכל על קוד של צוותים אחרים ואין להקריא קוד לצוותים אחרים.

אין להשתמש ברכיב ג'מיני Gemini או כל סייען מתקדם אחר.

מועד ההגשה הוא דקה לפני תחילת השיעור ב-13/1. לא יינתנו הארכות.

בהצלחה,

-- יובל

התקנות

דבר ראשון, נתקין את החבילות הנדרשות. ל- datasets ו- tokenizers שהכרנו מתרגיל 2 מצטרפת transformers, גם היא מבית האגיגפייס, שבעזרתה נוכל לטעון את המודל המאומן ואת הטוקניזר שלו ולהשתמש בהם על טקסט חופשי.

```
!pip install datasets -q
!pip install tokenizers -q
!pip install transformers -q
```



480.6/480.6 kB	8.8 MB/s	eta 0:00:00
116.3/116.3 kB	7.1 MB/s	eta 0:00:00
179.3/179.3 kB	11.0 MB/s	eta 0:00:00
134.8/134.8 kB	7.6 MB/s	eta 0:00:00
194.1/194.1 kB	11.8 MB/s	eta 0:00:00

ERROR: pip's dependency resolver does not currently take into account all the package requirements of the package gcsfs 2024.10.0 requires fsspec==2024.10.0, but you have fsspec 2024.9.0 which is incompatible

נניבא חבילות ומודולים. הרובוטריקים של האגיגפייס יודעים לדבר עם torch וכיוון שנוח לנו נעבוד איתו לצרכי עיבוד נוסף של הפלטים.

```
import numpy as np
import datasets
from transformers import AutoTokenizer, AutoModelForCausalLM
from torch.nn.functional import softmax, log_softmax
import torch
```

נטען באמצעות datasets את אסופת סיפורי הילדים שעבדנו איתה כדי לאמן מודל שיכני מילים בשבוע 7. היום לא ממש נאמן כלום, אבל זה טוב שיש משפטים מן המוכן במקום להמציא.

```
stories = datasets.load_dataset("deven367/babylm-100M-children-stories")
```

→ /usr/local/lib/python3.10/dist-packages/huggingface_hub/utils/_auth.py:94: UserWarning: The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (<https://huggingface.co/settings/tokens>)
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public models.
warnings.warn(
README.md: 100% 661/661 [00:00<00:00, 31.1kB/s]
(...)-00000-of-00001- 10.7M/10.7M [00:00<00:00, 19.8MB/s]
13f0a33230d64dd9.parquet: 100%
(...)-00000-of-00001- 905k/905k [00:00<00:00, 9.67MB/s]
0177e6dbdee45eef.parquet: 100%
(...)-00000-of-00001- 1.10M/1.10M [00:00<00:00, 14.4MB/s]
0b94257b623049e5.parquet: 100%

כאמור, אנחנו נעבוד עם מודל [GPT-2](#) (הקישור מוביל לדף הסבר באתר האגיגפייס). צריך לטעון בנפרד את המודל ואת ה**טוקנייזר שלו** שאומן בשיטת קידוד זוגות בתים, או BPE, אותה למדנו בשבוע 8. שימו לב שבניגוד לתרגיל הקודם, הפעם לא נאמן כלום ולא נגריל כלום ולכן לא צריך לקבע זרע אקראיות.

```
tokenizer = AutoTokenizer.from_pretrained("openai-community/gpt2")
model = AutoModelForCausalLM.from_pretrained("openai-community/gpt2")
```



tokenizer_config.json: 100%

26.0/26.0 [00:00<00:00, 390B/s]

config.json: 100%

665/665 [00:00<00:00, 15.8kB/s]

vocab.json: 100%

1.04M/1.04M [00:00<00:00, 9.63MB/s]

merges.txt: 100%

456k/456k [00:00<00:00, 8.69MB/s]

tokenizer.json: 100%

1.36M/1.36M [00:00<00:00, 25.7MB/s]

model.safetensors: 100%

548M/548M [00:06<00:00, 80.3MB/s]

generation_config.json: 100%

124/124 [00:00<00:00, 2.00kB/s]

פונקציית הייצוג של מודל בחבילת transformers נותנת פרטים על הארכיטקטורה של המודל בצורה מקוננת ונוחה.

1. ענו על השאלות הבאות בהסתמך על הפלט של שני התאים הבאים (1 נק):

1. (0.25) מהו גודל אוצר המילים של המודל?

2. (0.25) ModuleList מתאר רשימת מודולים לפי סדר הפעלתם. מה שונה באופן סידור המודולים של בלוק השכבה ב-GPT-2 (כלומר, החלק שמשתכפל בהתאם למספר השכבות) מזה שלמדנו בכיתה? אין צורך להתבונן בתוך מרכיבי המודולים עצמם (כלומר ניתן להניח שמודול עם Attention בשם הוא בלוק צומי עצמי, למשל).

3. (0.5) שכבת ה-MLP של המודל משתמשת בפונקציית אקטיבציה שלא למדנו עליה. חקרו אודותיה מעט וענו מהי התכונה העיקרית שמבדילה אותה משלוש הפונקציות שאנחנו מכירים (בפחות מחמש מילים).

תשובות: (נא לערוך את התא הטקסטואלי)

1. 50257

2. ניתן לראות כי בתוך ModuleList נמצאים 12 בלוקים של GPT2 וכן בתוך כל אחד מהם יש שכבת נורמליזציה, שכבת צומי עצמי שכבת נורמליזציה ו-GPT2MLP אשר מורכב משתי שכבות קונבולוציה חד ממדיות אקטיבציה (GELU) ונישור (dropout) לכן ההבדל הינו ראש ה-MLP שלא נכח בראש הצומי העצמי וכן גם שכבות הנורמליזציה

3. GELU - גזירה ומוכלת בתחום $[0, \infty]$

model



```
GPT2LMHeadModel(
  (transformer): GPT2Model(
    (wte): Embedding(50257, 768)
    (wpe): Embedding(1024, 768)
    (drop): Dropout(p=0.1, inplace=False)
    (h): ModuleList(
      (0-11): 12 x GPT2Block(
        (ln_1): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
        (attn): GPT2SdpaAttention(
          (c_attn): Conv1D(nf=2304, nx=768)
          (c_proj): Conv1D(nf=768, nx=768)
```

```

        (attn_dropout): Dropout(p=0.1, inplace=False)
        (resid_dropout): Dropout(p=0.1, inplace=False)
    )
    (ln_2): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
    (mlp): GPT2MLP(
      (c_fc): Conv1D(nf=3072, nx=768)
      (c_proj): Conv1D(nf=768, nx=3072)
      (act): NewGELUActivation()
      (dropout): Dropout(p=0.1, inplace=False)
    )
  )
  (ln_f): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
)
(lm_head): Linear(in_features=768, out_features=50257, bias=False)
)

```

tokenizer

```

GPT2TokenizerFast(name_or_path='openai-community/gpt2', vocab_size=50257,
model_max_length=1024, is_fast=True, padding_side='right', truncation_side='right',
special_tokens={'bos_token': '<|endoftext|>', 'eos_token': '<|endoftext|>',
'unl_token': '<|endoftext|>'}, clean_up_tokenization_spaces=False,
added_tokens_decoder={
  50256: AddedToken("<|endoftext|>", rstrip=False, lstrip=False,
single_word=False, normalized=True, special=True),
})

```

נרענ את זכרוננו לגבי איך מחלצים את הטקסט מתוך הדאטא.

```

story1 = list(stories['train'])[1]['text']
print(story1)

```

```

HIGH above the city, on a tall column, stood the statue of the Happy Prince. He was

```

ועכשיו נבדוק איך הטוקנייזר מייצג אותו. הפונקציה הראשונה שנקרא לה תיצור עבורנו אצווה (batch) בפורמט פייטורץ (pt) שתכיל, בין היתר, את האלמנטים הבאים:

- קריאה לפונקציית tokens() תחזיר את המחרוזות אליהן התפצל הטקסט כדי להיכנס לאוצר המילים של המודל. BPE הוא טוקנייזר "תת-מילי" subword, כלומר יש מרכיבים שאינם מהווים מילים שלמות וזאת כדי שגם למילים נדירות או לא-מוכרות יהיה ייצוג. שימו לב שיש תו מיוחד שמוצג כאן כ-G עם נקודה מעל, תפקידו לסמן רוח והוא לרוב יתחבר למילה שאחריו.
- input_ids הם הטוקנים שיצר הטוקנייזר בצורתם הגולמית כאינדקסים מתוך אוצר המילים (כך, למשל, ניגש לשיכונים שלהם). אינטואיטיבית, נסו להעריך אם יש משמעות כלשהי לאינדקסים.

2. ענו על השאלות הבאות (1):

1. (0.5) מה מסמנים האלמנטים שמחזירה הפונקציה word_ids(), ובאיזה סיטואציה במיוחד הם חשובים? חישבו על הבעיות שלמדנו עד היום בקורס.

2. (0.5) בחרו עוד אלמנט בפלט של הטוקנייזר והסבירו את תפקידו. אפשר להדגים במקום המיועד לכך בתא הבא.

תשובות: (נא לערוך את התא הטקסטואלי)

1. האלמנטים שמוחזרים הינם מיקומי השיכונים ביחס למילה מהם הם הגיעו / נוצרו, ביחס למיקום המילה במשפט. מיקומים אלו יכולים לעזור בסיטואציות בהן נרצה ליצור positional embeddings או להתחשב במיקום השיכון ביחס למשפט.

2. input_ids הינם אלמנטים המייצגים את מיקום המילה / שיכון ביחס לאוצר המילים של הטוקנייזר, מיקומים אלו יהיו האינפוטים לטרנספורמר (או יותר נכון embeddor) ולכן נצטרך אותם על מנת להמיר מילים באצווה לשיכונים עצמם.

```
input_toks = tokenizer.batch_encode_plus([story1], return_tensors='pt')
tok_texts = input_toks.tokens()
print(' '.join(tok_texts[:10]))
print(len(input_toks.input_ids[0]))
print(input_toks.input_ids[0][:10])
print(input_toks.word_ids()[:10])
### 2.2 One more element: ###
#print(dir(input_toks), input_toks)
```

```
# for k,v in tokenizer.vocab.items():
#     if (v < 322 and v > 318) or v < 20 or v==50256:
#         print(f"{k} id is {v}")
```



H IGH Ġabove Ġthe Ġcity , Ġon Ġa Ġtall Ġcolumn

58

```
tensor([ 39, 18060, 2029, 262, 1748, 11, 319, 257, 7331, 5721])
[0, 0, 1, 2, 3, 4, 5, 6, 7, 8]
```

4 id is 19

, id is 11

1 id is 16

" id is 1

) id is 8

% id is 4

im id is 320

\$ id is 3

3 id is 18

. id is 13

2 id is 17

& id is 5

! id is 0

* id is 9

+ id is 10

<|endoftext|> id is 50256

id is 2

' id is 6

0 id is 15

Ġon id is 319

/ id is 14

- id is 12

(id is 7

am id is 321

כעת נפעיל את המודל עצמו באמצעות קריאה ישירה אליו על המבנה המפורק (שתי כוכביות פייתוניות) של פלט הטוקנייזר. הקריאה הזו תעביר את הטוקנים דרך כל הרובוטריק, מהשיכונים ועד שכבת החיזוי, ותיתן לנו את תוצרי כל אחד משלבי החיזוי של GPT-2 כמודל שפה.

הפלט מגדיר מבנה די מסובך (אפשר לראות תיעוד מלא באתר הדוקומנטציה או באמצעות קריאה למודל עם סימן שאלה אחריו בתא קוד במחברת). ניכנס ישר למה שמעניין אותנו - הלוג'יטים (logits), שהם הציונים הגולמיים של כל הטוקנים באוצר המילים בכל אחד משלבי החיזוי, כלומר על כל תחילית של הקלט. כך האיבר הרביעי במבנה הלוג'יטים ייתן את כל התחזיות עבור כל הטוקנים האפשריים בהינתן ארבעת הטוקנים הראשונים של הטקסט, כלומר את מרחב התחזיות לטוקן החמישי. לאחר שנעביר את הווקטור הזה דרך סופטמקס, נקבל את התחזיות האלה כהתפלגות.

```
generative_outputs = model(**input_toks)
print(generative_outputs[0].shape)
gen_out_logits = generative_outputs[0][0]
print(len(gen_out_logits))
print(len(gen_out_logits[3]))
print(float(gen_out_logits[3][1748]))
print(float(gen_out_logits[3][0]))
```

```
→ torch.Size([1, 58, 50257])
58
50257
-79.91033172607422
-87.89537811279297
```

```
softmaxed_outputs = softmax(gen_out_logits[3], dim=0) # specify a dimension or an annoyi
print(float(softmaxed_outputs[1748]))
print(torch.argmax(softmaxed_outputs), softmaxed_outputs[4417], [(k,v) for k,v in tokeniz
```

```
→ 0.005037951283156872
tensor(4417) tensor(0.0524, grad_fn=<SelectBackward0>) [('Ġcity', 1748), ('Ġsurface',
```

נראה שהמודל מעניק למילה האמיתית, city, הסתברות של כחצי אחוז להיות המילה הבאה. זה לא מעט בהינתן גודל אוצר המילים הכללי, אבל האם יש מילים שהוא מעדיף?

3. ענו (0.5):

1. (0.25) כמה מילים מעדיף המודל על-פני city?

2. (0.25) מהי המילה לה הוא מעניק את ההסתברות הגבוהה ביותר? (לא האינדקס: המילה עצמה)

תשובה - נמצאת בקוד הנמצא מטה אך מסוכמת גם כאן:

1. 13 מילים יותר סבירות מ city

2. "surface" עם הרחב בהתחלה

Answers for 3 (edit where necessary)

Answer for 3.1

```
city_prob = float(softmaxed_outputs[1748])
print(f"Number of words more probable than city: {len([prob for prob in softmaxed_outputs
```

Answer for 3.2

```
def best_token_id(sm_output) -> int:
    best_5_ids = torch.argsort(sm_output, descending=True)[:20]
    # print(f"{tokenizer.convert_ids_to_tokens(best_5_ids)=}")
    best_input_id = best_5_ids[0]
    print(f"{best_input_id=}")
    return best_input_id
```

```
def token_from_index(idx: int) -> str:
    ...

    Hint: look at the tokenizer
    ...

    return tokenizer.convert_ids_to_tokens([idx])[-1]
```

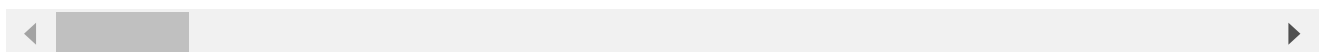
```
best_in_outputs = best_token_id(softmaxed_outputs)
print(f"Most probable word: {token_from_index(best_in_outputs)}")
```

```
➞ Number of words more probable than city: 13
    best_input_id=tensor(4417)
    Most probable word: Ġsurface
```

וכך נוכל לראות את ההסתברויות של כל המילים ברצף. שימו לב שיש הסתברויות מאוד גבוהות להמשכי המילים הנכונים כאשר הטוקנייזר חותך אותן, למשל בתוך המילה saphires:

```
word_probs = []
for loc, tok_idx in enumerate(input_toks.input_ids[0]):
    if loc < 1: continue # think why!
    sm_loc = softmax(gen_out_logits[loc - 1], dim = 0)
    prob = sm_loc[tok_idx]
    word_probs.append((tok_texts[loc], prob))
print('; '.join([f'{txt}: {prob:.4f}' for txt, prob in word_probs]))
```

```
➞ IGH: 0.0000; Ġabove: 0.0000; Ġthe: 0.3064; Ġcity: 0.0050; ,: 0.1424; Ġon: 0.0086; Ġa:
```



כמו שהזכרנו בכיתה, קשה למצוא מטריקות טובות להערכת חילול טקסט. מה שכן אפשר לעשות זה להעריך מודלים באמצעות מידת הסבירות הכוללת שהם נותנים לטקסט נתון. המטריקה, שגם מיתרגמת לפונקציית הפסד נוחה, היא **בלביות**, או בלעז **perplexity**. היא הגיעה אלינו מתורת האינפורמציה, והרעיון הכללי שבה הוא לחשב כמה המודל "מופתע" מהרצף שהוא נתקל בו, לעומת מה שהוא "רגיל" אליו.

להלן נוסחאתה:

$$PPL(\mathbf{x}) = \exp \left\{ -\frac{1}{N} \sum_i^N \log P(x_i | x_{<i}) \right\}$$

בכל נקודה ברצף, ככל שלוג ההסתברות גבוה יותר כך המודל "ציפה" יותר למילה האמיתית. לאחר השלילה וההעלאה מחדש בחזקה, מקבלים מדד שככל שהערך שלו גבוה יותר, המודל פחות טוב בזיהוי הרצף. באופן מדויק יותר, ככל שהבלבלות גבוהות כך הרצף פחות סביר לפי המודל.

4. ממשו את פונקציית הבלבלות וחשבו אותה עבור הרצף שלנו (1).

שימו לב שלוג ההסתברות אינו הלוג'יט! (מהו כן הלוג'יט? חכו לתרגיל 5.) תצטרכו לעשות כאן שימוש בפונקציה שייבאנו מבעוד מועד ביחד עם הסופטמקס (ר' לעיל) שמחשבת לוג הסתברות באופן ישיר ומהיר וללא בעיות נומריות.

כמו כן, שימו לב לשינוי הקטנטן שנאלצנו לעשות בתא הקודם כתוצאה מכך ש-GPT לא מתחיל רצפים עם טוקן מיוחד, והתאימו את הנוסחא בשאלה הזו.

```
def perplexity(logits, true_idx) -> float:
    assert len(logits) == len(true_idx)
    N = len(true_idx)

    ### 4. keep going ###
    words_probs = []
    for loc, tok_idx in enumerate(input_toks.input_ids[0]):
        if loc < 1: continue
        lsm_loc = log_softmax(gen_out_logits[loc - 1], dim = 0)
        prob = lsm_loc[tok_idx]
        words_probs.append(prob)
    ppl = torch.exp(-1*sum(words_probs)/(N-1))
    return ppl
```

```
print(perplexity(gen_out_logits, input_toks.input_ids[0]))
```

```
→ tensor(65.2570, grad_fn=<ExpBackward0>)
```

נסבר את האוזן עם חישוב אותה המטריקה עבור המקטע הבא של הסיפור. אפשר לראות אם המודל "ציפה" לו יותר או פחות מלמקטע הראשון.

ואפשר גם סתם על טקסט חופשי, כמו בתא שאחריו.

ולראות אם GPT-2 מכיר ביטויים מפורסמים באנגלית.

```
story2 = list(stories['train'])[2]['text']
input_toks_2 = tokenizer.batch_encode_plus([story2], return_tensors='pt')
print(len(input_toks_2.tokens()))
print(' '.join(input_toks_2.tokens()))
generative_outputs_2 = model(**input_toks_2)[0][0]
print(perplexity(generative_outputs_2, input_toks_2.input_ids[0]))
```

```
→ 68
```

```
He Ġwas Ġvery Ġmuch Ġadmired Ġindeed . Ġ ĠâĠ I He Ġis Ġas Ġbeautiful Ġas Ġa Ġweather
tensor(34.9774, grad_fn=<ExpBackward0>)
```



```
idiom = 'The rain in Spain stays mainly on the plain.'
idiom_toks = tokenizer.batch_encode_plus([idiom], return_tensors='pt')
print(' '.join(idiom_toks.tokens()))
gen_outs_idiom = model(**idiom_toks)[0][0]
perplexity(gen_outs_idiom, idiom_toks.input_ids[0])
```

```
→ The Grain Gin Spain stays mainly on the plain .
tensor(3.1091e+11, grad_fn=<ExpBackward0>)
```

```
true_idiom_last_tok = idiom_toks.input_ids[0][8]
print(int(true_idiom_last_tok))
idiom_sm = softmax(gen_outs_idiom[7], dim=0)
print(float(idiom_sm[true_idiom_last_tok]))
print(float(idiom_sm.max()))
```

```
most_probable_token = int(best_token_id(idiom_sm))
print(most_probable_token, token_from_index(most_probable_token))
```

```
→ 8631
0.00023369898553937674
0.11648979038000107
best_input_id=tensor(2323)
2323 ground
```

הפעם נבקש מ-GPT-2 להשלים רצפים בלי שאנחנו נותנים אותם כחלק מהקלט. לא יהיה כאן שום הבדל מבחינת ההתנהגות, הודות לסגירת המשולש העליון של מטריצת הצומי, אבל ככה נדמה יותר טוב את אופן הפעולה של המודל **בשלב היישום** (שפוגשים היום עם מודלי הצ'ט למיניהם): נותנים טקסט והמודל משלים.

5. עבור שלושת הרצפים הבאים, מצאו את המילה ש-GPT-2 ישלים תחת מנגנון חילול חמדן (כלומר, את הטוקן הכי סביר) (0.5):

```
sentences_to_complete = ['Why is the sky',
                          "You ain't seen nothing",
                          'Question: When was Barack Obama elected President of the USA? A
                          ]
```

```
def most_prob_continuation(sentence: str) -> int:

    ### Q5 - edit here ###
    input_tokens = tokenizer.batch_encode_plus([sentence], return_tensors='pt')
    outputs = model(**input_tokens)
    logits = outputs[0][0]
    sm_outputs = softmax(logits[-1], dim=0)
    return best_token_id(sm_outputs)

for s in sentences_to_complete:
    most_prob_cont = most_prob_continuation(s)
    print(int(most_prob_cont), token_from_index(most_prob_cont))

→ best_input_id=tensor(4171)
4171 Ġblue
best_input_id=tensor(1865)
1865 Ġyet
best_input_id=tensor(3269)
3269 ĠJanuary
```

[שאלות 6 ו-7 אינן תלויות זו בזו. אם מתקשים עם 6, אפשר לוותר על חצי נקודה ולדלג ל-7.]

6. התשובה לשאלה האחרונה היא לא מה שציפינו עבור תשובה בעלת טוקן אחד. מצאו החל מאיזה מקום בדירוג הטוקנים מקבלים תשובה מהסוג הרצוי (שנה) והאם זו השנה הנכונה. (0.5)

רמז: יש פונקציה מובנית לטנזורים של פייטורץ' שתקל על פתרון השאלה הזו.

**** אנחנו לא בטוחים אם הכוונה בשאלה מתי הוא נבחר או מתי היו הבחירות בחנו את שתי האופציות ****

7. עוד דרך לקבל תשובה נכונה היא להזין את החודש הנכון כהמשך הקלט ולראות מה הפלט הבא. מיצאו את ההסתברות שבהנתן הקלט המקורי (לעיל) המודל יוציא בדגימה את החודש הנכון ולאחריו את השנה הנכונה. הראו את החישוב. (0.5)

```
### Q6 + Q7 (more cells possible) ###
obama_sentence = 'Question: When was Barack Obama elected President of the USA? Answer:'
# The Answer should be January 20 2009 (elected) or November 08 2008 (elections)
def q6(sen, true_answer):
    input_tokens = tokenizer.batch_encode_plus([sen], return_tensors='pt')
    outputs = model(**input_tokens)
    logits = outputs[0][0]
    sm_outputs = softmax(logits[-1], dim=0)
    sorted_sm = torch.argsort(sm_outputs, descending=True)
    print(tokenizer.convert_ids_to_tokens(sorted_sm[:20]))

    for i, token in enumerate(tokenizer.convert_ids_to_tokens(sorted_sm)):
        if token[1:] == true_answer:
            print(i, token, sm_outputs[sorted_sm[i]])
            return i, sm_outputs[sorted_sm[i]]
    return -1, 0
# return best_token_id(sm_outputs)
```

```
def q7(sen):
    accumulated_prob = q6(sen, "January")[1] * q6(sen + " January", "20")[1] * q6(sen + " Ja
```

```

print(f"\n\nOption 1", accumulated_prob)
print("\n\n")
accumulated_prob = q6(sen, "November")[1] * q6(sen + " November", "08")[1] * q6(sen + "
print(f"\n\nOption 2", accumulated_prob)

print(q6(obama_sentence, "2009")[0])
print(q6(obama_sentence, "2008")[0])
print("\n\n")
q7(obama_sentence)

```

```

→ ['ĠJanuary', 'ĠNovember', 'ĠMarch', 'ĠFebruary', 'ĠSeptember', 'ĠOctober', 'ĠJuly', '
34 Ġ2009 tensor(0.0055, grad_fn=<SelectBackward0>)
34
['ĠJanuary', 'ĠNovember', 'ĠMarch', 'ĠFebruary', 'ĠSeptember', 'ĠOctober', 'ĠJuly', '
27 Ġ2008 tensor(0.0069, grad_fn=<SelectBackward0>)
27

```

```

['ĠJanuary', 'ĠNovember', 'ĠMarch', 'ĠFebruary', 'ĠSeptember', 'ĠOctober', 'ĠJuly', '
0 ĠJanuary tensor(0.0634, grad_fn=<SelectBackward0>)
['Ġ20', 'Ġ1', 'Ġ25', 'Ġ22', 'Ġ21', 'Ġ27', 'Ġ7', 'Ġ23', 'Ġ15', 'Ġ17', 'Ġ19', 'Ġ8', 'Ġ2
0 Ġ20 tensor(0.0780, grad_fn=<SelectBackward0>)
['', 'th', '-', '.', 'Ġ-', 'st', 'ĠĠĠ', 'Ġ', 'Ġ2009', 'Ġ2012', 'Ġof', 'Ġ2008', 'Ġ(
8 Ġ2009 tensor(0.0010, grad_fn=<SelectBackward0>)

```

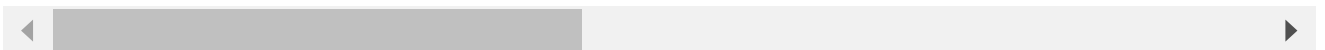
```
Option 1 tensor(4.7109e-06, grad_fn=<MulBackward0>)
```

```

['ĠJanuary', 'ĠNovember', 'ĠMarch', 'ĠFebruary', 'ĠSeptember', 'ĠOctober', 'ĠJuly', '
1 ĠNovember tensor(0.0531, grad_fn=<SelectBackward0>)
['Ġ8', 'Ġ7', 'Ġ9', 'Ġ6', 'Ġ20', 'Ġ1', 'Ġ22', 'Ġ25', 'Ġ5', 'Ġ4', 'Ġ17', 'Ġ19', 'Ġ2', '
60 Ġ08 tensor(0.0006, grad_fn=<SelectBackward0>)
['', 'th', 'Ġ2012', 'Ġ2008', 'Ġ2014', '.', 'Ġ2009', 'Ġ2010', 'Ġ2013', 'Ġ2011', 'Ġ201
3 Ġ2008 tensor(0.0037, grad_fn=<SelectBackward0>)

```

```
Option 2 tensor(1.0926e-07, grad_fn=<MulBackward0>)
```



אפשר לראות שבאופציה הראשונה - 20 בינואר 2009 ההסתברות לתשובה הזו הינה 0.00047% כאשר נבקש תחילה את החודש אחר כך את היום ולבסוף את השנה, אם נרצה רק את החודש והשנה נצטרך לוותר ולחשב מחדש עבור חודש ושנה

באופן דומה עבור - 08 בנובמבר 2008 ההסתברות לתשובה זו הינה - 0.000011%

עיבוד שפה טבעית (20225211) 2024-2025 - תרגיל 4 חלק 2 (4 נק')

שלום לכולם!

כעת נתבונן קצת יותר לעומק בשיטות ליצירת אוצרות מילים (vocabularies) מתחת לרמת המילה (subword). נאסוף קורפוס, נאמן טוקניזר לעברית שמתאים ל-BERT, ונבחן את תכונותיו.

בסיום העבודה, הגישו **קובץ pdf** הכולל את כל הפלטים מכל התאים. דרך אפשרית אחת להשיג קובץ pdf היא על-ידי הדפסת הדף (מתוך colab, לא מהדפדפן) ולבחור "מדפסת" ישירה ל-pdf.

שימו לב שכל המידע הדרוש מופיע בתדפיס ה-pdf --- לא יתקבלו ערעורים על-בסיס חיתוך. אין לחרוג ממבנה התאים ואין לענות במקומות שאינם מוגדרים לכך. אין למחוק הערות שמסמנות את מיקומי קטעי הקוד שלכם ואין לערוך קוד שנמצא מחוץ לתחומים המוגדרים. סעיפים שחורים מהנחיה זו לא ייבדקו.

ניתן להגיש בצוותים של עד שלושה סטודנטים. לא יהיו הנחות לצוותים קטנים. אפשר לחלק את פתרון התרגיל בין חברי.ות הצוות כרצונכם. אין להסתכל על קוד של צוותים אחרים ואין להקריא קוד לצוותים אחרים.

אין להשתמש ברכיב ג'מיני Gemini או כל סיוען מתקדם אחר.

מועד ההגשה הוא דקה לפני תחילת השיעור ב-13/1. לא יינתנו הארכות.

בהצלחה,

-- יובל

התקנות

דבר ראשון, נתקין את החבילות הנדרשות, כולן מבית האגינפייס.

```
!pip install datasets -q
!pip install tokenizers -q
!pip install transformers -q
```

נייבא חבילות. את os נצטרך כדי לכתוב את הדאטאסט שלנו לטובת האימון, וכדי לכתוב את אוצר המילים של הטוקניזר לאחר האימון כדי שיוכל להיטען לאובייקט הקורא אותו.

```
import os
import datasets
from tqdm.notebook import tqdm
```

החבילה datasets מכילה קורפוסים לא-מתויגים רבים, בדיוק מהסוג שאנחנו צריכים לטובת אימון טוקניזר תת-מילי. נשתמש ב-[OSCAR](#), פרויקט נקיון ותחזוקה של מידע ממאגר הגריפה הגדול commoncrawl התומך ב-166 שפות, וביניהן עברית.

נבקש ממנו את מחיצת האימון (היחידה שקיימת) לעברית ונספק את הפרמטר streaming כדי שנוכל לדגום חלק קטן ממנה בלי להוריד הכל לזכרון של קולאב (המון זמן). 200,000 מסמכים יהיו די והותר.

```
dataset = datasets.load_dataset(
    'oscar',
    'unshuffled_deduplicated_he',
    split='train',
    streaming=True,
    trust_remote_code=True)

sampled_ds = []
for i, doc in tqdm(enumerate(dataset)):
    if i > 200_000:
        break
    sampled_ds.append(doc)
```

```
git rm .gitignore
```

Start coding or [generate](#) with AI.

היעלה על הדעת שנוריד מידע ולא נתבונן בו? הבה נבחן את המסמכים האחרונים שהורדנו.

sampled_ds[-10:]

```
[[{'id': 199991,
  'text': 'ולכבוד הסרט גוגל בנו משחק קצרצר של מלחמת הכוכבים הפועל "Star Wars The Force Awakens" בלילה האחרון החלו הקרנות הבכורה בארץ ובעולם של הסרט החדש.
Stormעליכם יהיה להכנס לקישור עם הקוד המיוחד שיוצג לכם באתר במחשב מהפלאפון ולאחר ביצוע כיוול קצרצר ופשוט תוכלו להדוף יריות ולבסוף גם להלהם בn. בתוך דפדפן הכרום
תגיות: גוגל, דפדפן, כרום, n. לצערי המשקה מאוד מאוד קצר ומכיוון שמדובר במשחק דפדפן באיכות לא רעה ושליטה שניעשת ממכשיר אחר דרך הרשת יש מדי פעם קצת האטהn.
Trooper. Chrome, Lightsaber, Star Wars, Storm Trooper, The Force Awakens',
  {'id': 199992,
  'text': 'התרגלנו לחשוב שיש לנו סוג של בעלות על האדם האהוב עלינו. לפעמים מספיק שנראה את הבן זוג שלנו מסתכל על אישה אחרת, והקנאה מתעוררת. האמירה הבאה אולי':
  {'id': 199993,
  'text': 'כולנו מנהלים את עצמינו במספר מעגלי חיים: קריירה, משפחה, חברה, תחומי עניין אישיים. לא פעם אנו מוצאים את עצמינו "נקרעים" בין ה"עולמות" הנ"ל ולא פעם':
  {'id': 199994,
  'text': 'הנוסע המוסלמי אבן בטוטא (1304-1377) ביקר בטורקיהnדף הבית << הרצאות << טורקיה << סקר ארכיאולוגי\היסטורי של אתרים מרכזיים יווניים ורומיים במחוזות טורקיה':
  {'id': 199995,
  'text': 'אם זה לגובה או אם זה למרחב, בכל סוג של פרויקט מתקן החשמל דורש שיטות תכנון מתקדמות אשר התפתחו עם השנים מליבה יציבה ומוכחת של ידע, ניסיון ועבודת צוות.
  {'id': 199996,
  'text': 'אחד השיקולים החשובים ביותר בבחירת מקרר הוא בחירת הנפח המתאים. מקרר גדול מידי יגרם לבובזו אנרגיה ומקרר קטן מידי יהיה מוגבל ביכולת האחסון, 500-600 ליטרים':
  {'id': 199997,
  'text': 'סוויד - זמש הוא החומרים PU חומר עליוןאביב / סתיו נעלים ללא שרוכים לבן / שחור 6534228 2018 - $29.99 PU / בגדי ריקוד גברים נעלי נוחות סוויד':
  {'id': 199998,
  'text': 'כדי לסייע לכם להתאים את השיש לבחירות השונות שצריך לעשות במסגרת תהליך של עיצוב מטבחים, נועד מאמר זה. השתמשו בו בתבונה והשיגו ממנו את המרב. אתם בהחלט':
  {'id': 199999,
  'text': 'דעו כי בכל הנוגע לבחירת שישn, יכולים לקחת את מה שמועיל לכם ולהתעלם ממה שלא נראה לכם רלוונטי, אך דעו כי הבחירה הנכונה היום, היא שתאפשר לכם ליהנות מהשיש מחר ובכלל
  {'id': 200000,
  'text': 'נשאלת השאלה החשובה מתי נכון לבחור את השיש והאם לתקופה בה מתבצעת הבחירה יש משמעות מסוימת, התשובה על כךn. השימוש במשטה, בטעם האישי שלכם מבחינה עיצובית ועוד
```

לפי הסרט המוזכר במסמך 199,991, עושה רושם שהדאטאסט, או לפחות החלק הזה שלו, הוא מאמצע העשור הקודם.

שמירת דאטאסט מסודר למטרות אימון טוקנייזר

הוא הבא מכין את הדאטא עבור מנגנון האימון שכיר בחלק הבא.

CORPUS_DIR = './oscar_heb'

```
if not os.path.exists(CORPUS_DIR):
    os.mkdir(CORPUS_DIR)
```

```
text_data = []
file_count = 0
```

```
for sample in tqdm(sampled_ds):
```

```
# remove newline characters from each sample as we need to use exclusively as seperators
sample = sample['text'].replace('\n', '\s')
```


מי שמעוניינת מוזמנת לשמור את הקבצים בנפרד מהמחברת ולטעון לפי הצורך בשלבי פתרון התרגיל כדי לחסוך את זמן האימון עם כל ששן. עם זאת, ההגשה הסופית חייבת להיות תוצר של הרצה אחת נקייה מההתחלה ועד הסוף.

הבה נסתכל על כמה דוגמאות:

```
from transformers import BertTokenizer
```

```
MODEL_DIR = './wp-heb'
tokzr = BertTokenizer.from_pretrained(f'{MODEL_DIR}/wp-heb-vocab.txt')
```

```
print(f'There are {tokzr.vocab_size} tokens in our tokenizer\'s vocabulary.\n===')
print(tokzr.tokenize('hello world'))
print(tokzr.tokenize('שלום עולם'))
print(tokzr.tokenize('אם אין לי מי לי'))
```

```
c:\Users\Benzu\anaconda3\Lib\site-packages\transformers\tokenization_utils_base.py:1925: FutureWarning: Calling BertTokenizer.from_pretrained
warnings.warn(
There are 3000 tokens in our tokenizer's vocabulary.
===
['h', '##el', '##l', '##o', 'w', '##or', '##l', '##d']
['שלו', '##ם', 'עו', '##ם']
['אם', 'אין', 'אני', 'לי', 'מי', 'לי']
```

כמו שאנחנו רואים, הטוקנייזר שלנו בהחלט יודע לעבד עברית יותר טוב מאשר אנגלית, בהתאם לדאטא שקיבל לאימון.

סעיף 1 (0.5 נק): הצינו הסבר מדוע בכל-זאת "שלום" מפריד את האות הסופית אך "עולם" מפריד שתי אותיות אחרונות. ניתן להוסיף תא קוד להנמקה.

הסבר אפשרי הינו ש'שלו' הינו נפוץ יחסית ולכן קיבל טוקן משל עצמו, מאחר וזהו אלגוריתם גרידי הוא תחילית ארוכה יותר מ'של' ואילו 'עול' ככל הנראה לא נפוץ מספיק בשביל לקבל טוקן משל עצמו ולכן 'עו' קיבל את התחילית הארוכה ביותר (למטה ניתן לראות שעולים ועושים שניהם מתפרקים לטוקן 'עו')

```
print(tokzr.tokenize('שלום עולם'))

print(tokzr.tokenize('את לא כמו כולם'))
print(tokzr.tokenize('אין מקום בעולם בשבילם'))
print(tokzr.tokenize('סתם הולם'))

print(tokzr.tokenize('עולים לגובה עושים הרבה'))
print(tokzr.tokenize('שלחתי שליחים לשלום שלומית'))
```

```
['שלו', '##ם', 'עו', '##ם']
['את', 'לא', 'כמו', 'כול', '##ם']
['אין', 'מקום', 'בעולם', 'בש', '##ביל', '##ם']
['ס', '##תם', 'הל', '##ום']
['עו', '##לים', 'לג', '##ובה', 'עו', '##שים', 'הרבה']
['של', '##הת', '##י', 'שלי', '##חים', 'לש', '##לו', '##ם', 'שלו', '##מית']
```

כפי שאנו זוכרים מהשיעור, לרוב מאמנים טוקנייזר על אוצר מילים בסדר גודל של 30-50 אלף, ושלנו קטן פי עשרה. התוצאה היא הרבה פחות מילים וחלקי מילים שימושיים שזכו לקבל תמנית משלהן.

סעיף 2 (0.5 נק): מצאו מילה (רצף אותיות עבריות ללא רווחים) תקינה בעברית שמתפצלת ליותר מ-6 תמניות.

```
### 3 ###
MY_WORD = 'האינצקלופדיה'
### # ###

toks = tokzr.tokenize(MY_WORD)
print(len(toks), toks)

['האי', '##נ', '##צ', '##קל', '##יפ', '##די', '##ה']
```

כפי שהזכרנו בכיתה, אופן הפענוח (decoding, inference) של אלגוריתם וורדפיס הינו חמדני (greedy): בהינתן רצף תווים, וורדפיס יחפש את התמנית הארוכה ביותר שמתאימה לתחילתו, וימשיך מהנקודה בה נקטע.

סעיף 3 (1 נק): ממשו את אלגוריתם הפענוח באופן נכון אך לא בהכרח יעיל, באמצעות אובייקט הקורא את אוצר המילים ופועל חמדנית. הקוד בתא השני יבדוק את נכונותו.

```
from typing import Callable, List

class GreedyTokenizerFromFile(Callable):
```

```

def __init__(self, vocab_file: str) -> None:
    self.vocab = set()

    ### 5.1 (0.25 points) ###
    self.vocab |= set(open(vocab_file, 'r', encoding='utf-8').read().splitlines())

    print(f'initialized with {len(self.vocab)} pieces')

def __call__(self, s: str) -> List:

    ### 5.2 (0.75 points, conditioned on test passing) ###
    tokens = []
    word = s
    while word != '###':
        for i in range(len(word), 0, -1):
            if word[:i] in self.vocab:
                tokens.append(word[:i])
                word = '###' + word[i:]
                break
        elif i == 1:
            tokens.append(['UNK'])
            return tokens

    return tokens

### you may define any additional functions below ###

#####
### Do not modify this cell when submitting ###
#####

from tokenizers.pre_tokenizers import BertPreTokenizer

greedy_tok = GreedyTokenizerFromFile(f'{MODEL_DIR}/wp-heb-vocab.txt')

# Bert's PreTokenizer handles punctuation. It's called implicitly by tokzr
pretok = BertPreTokenizer()

# our "test set" is a random document from the dataset
for w, char_span in tqdm(pretok.pre_tokenize_str(sampled_ds[2023]['text'])):

    # ground truth - the automatically-learned tokenization
    trained_tok = tokzr.tokenize(w)

    # preparation - BertTokenizer actually has a preprocessing submodule called
    # "basic_tokenizer" that lowercases English and such
    grd_tok = greedy_tok(tokzr.basic_tokenizer.tokenize(w)[0])

    # Test - these asserts should not fail #
    assert trained_tok == grd_tok, f'{w}, {trained_tok}, {grd_tok}'

```

```

→ initialized with 3000 pieces
a%|      | a/25 [aa-aa<? >it+<1

```

משחק השבת

נניח שאנחנו מתכננים משימה שנוגעת לעיבוד טקסט ממסמכים הנוגעים לספורט. בואו נראה עד כמה המודל שלנו מותאם למילים מתוך התחום (domain) הספורטיבי:

```
' '.join(tokzr.tokenize('מכבי הפועל כדורגל כדורסל כדורעף טניס'))
```

```
→ 'מכ ##בי הפ ##ועל כד ##גל כד ##ור ##ור ##ס ##ל כד ##ור ##ע ##ט ##ני ##ס'
```

נראה שהמצב די רע. ננסה לאמן מודל טוקניזציה שיטפל במילים האלה יותר טוב רק באמצעות שינוי קורפוס האימון.

סעיף 4 (2 נק', אין ניקוד חלקי): צרו דאטאסט חדש מתוך הקורפוס שכבר בדינו (אל תייבאו קורפוס חדש, אל תפצלו מסמכים, ואל תכתבו יותר מ-200,000 מסמכים לדיסק) ואמנו עליו מודל WordPiece עם אותם היפר-פרמטרים כמקודם, שמפצל לכל היותר מילה אחת מהמילים לעיל ליותר משתי תמויות, ולא מפצל בכלל לפחות שתיים מהמילים. הראו את הפלט. תעדו היטב באמצעות הערות את כל השינויים שערכתם ביחס לקוד המקורי.

צוותים שיפנו בשעות הקבלה שלפני תחילת השיעור ב-7/1 עם פתרון חלקי יוכלו לקבל רמז. לאחר מועד זה לא תינתן עזרה על הסעיף.


```
import re

def clean_text(text: str) -> str:
    """
    Cleans the input text by removing characters that are not Hebrew letters, digits, spaces, or specified punctuation/symbols.
    Args:
        text (str): The input text to be cleaned.
    Returns:
        str: The cleaned text with only allowed characters and no extra spaces.
    """

    hebrew_digits_punct_regex = re.compile(r"^[^0-9א-ט\s.,!?:'\"~(){}[\]\-+*/@#$$%^&*(<>~|])")
    cleaned_text = hebrew_digits_punct_regex.sub("", text)
    cleaned_text = re.sub(r'\s+', ' ', cleaned_text).strip()
    return cleaned_text

text = "45 ()*8^%$#@ (: האם זה עובד? ,123 זהו מבחן abc <>"
cleaned_text = clean_text(text)
print(cleaned_text)
```

➡ 45 (<> שלום! זהו מבחן 123, האם זה עובד? :) *^%\$#@)

```
def download_corpus(keywords: List[str], n_docs: int) -> List[dict]:
    """
    Downloads a corpus of documents containing specified words.
    This is the same as before, just filtering the documents by specified keywords.
    Args:
        words (List[str]): A list of words to search for in the documents.
        n_docs (int): The number of documents to download.
    Returns:
        List[dict]: A list of dictionaries, each representing a document that contains any of the specified words.
    """
    i = 0
    sampled_sports_ds = []

    with tqdm(total=n_docs, desc="Finding sports documents") as pbar:
        for doc in dataset:
            if any([word in doc['text'] for word in keywords]):
                sampled_sports_ds.append(doc)
                i += 1
                pbar.update(1)
                if i >= n_docs:
                    break
    return sampled_sports_ds
```

```
import shutil
```

```
def save_corpus(corpus: List[dict], directory: str, clean=False) -> None:
    """
    Save a cleaned corpus to a specified directory in chunks of 5000 samples.

    Args:
        corpus (List[dict]): A list of dictionaries, each containing a 'text' key with the text sample.
        directory (str): The directory where the cleaned text files will be saved.

    Returns:
        None
    """

    if os.path.exists(directory):
        shutil.rmtree(directory)
    os.mkdir(directory)

    text_data = []
    file_count = 0

    for sample in tqdm(corpus, desc="Saving corpus"):

        # remove newline characters from each sample as we need to use exclusively as separators
        sample = sample['text'].replace('\n', '\s')

        text_data.append(sample)
        if len(text_data) == 5_000:

            # once we hit the 5K mark, save to file
```

```
with open(f'{directory}/text_{file_count}.txt', 'w', encoding='utf-8') as fp:
    text = clean_text('\n'.join(text_data)) if clean else '\n'.join(text_data)
    fp.write(text)
```

```
text_data = []
file_count += 1
```

```
# after saving in 5K chunks, we may have leftover samples, we save those now too
with open(f'{directory}/text_{file_count}.txt', 'w', encoding='utf-8') as fp:
    text = clean_text('\n'.join(text_data)) if clean else '\n'.join(text_data)
    fp.write(text)
```

```
>>:24: SyntaxWarning: invalid escape sequence '\s'
>>:24: SyntaxWarning: invalid escape sequence '\s'
C:\Users\Benzu\AppData\Local\Temp\ipykernel_35672\3057535395.py:24: SyntaxWarning: invalid escape sequence '\s'
sample = sample['text'].replace('\n', '\s')
```

```
SPORTS_WORDS = 'מכבי הפועל כדורגל כדורסל כדורעף טניס'.split()
SPORTS_CORPUS_DIR = './sports_adjacent-clean'
SPORTS_MODEL_DIR = './sp-heb'
MODEL_NAME = 'sports_adjacent-clean'
N_DOCS = 50_000
```

```
### 4 ###
```

```
# download the corpus with the sports keywords
sampled_sports_ds = download_corpus(SPORTS_WORDS, N_DOCS)
```

```
# clean and save the corpus
save_corpus(sampled_sports_ds, SPORTS_CORPUS_DIR, clean=True)
```

```
# All of this is the same as before, but with the new corpus
sports_tokzr = tokzr_sports_trainer = BertWordPieceTokenizer(clean_text=True)
sports_paths = sorted([str(x) for x in Path(SPORTS_CORPUS_DIR).glob('**/*.txt')])
tokzr_sports_trainer.train(files=sports_paths, vocab_size=3_000, min_frequency=2,
                           limit_alphabet=1_000, wordpieces_prefix='##',
                           special_tokens=[
                               '[PAD]', '[UNK]', '[CLS]', '[SEP]', '[MASK]'])
if not os.path.exists(MODEL_DIR):
    os.mkdir(MODEL_DIR)
```

```
tokzr_sports_trainer.save_model(MODEL_DIR, f'{MODEL_NAME}')
```

```
Finding sports documents: 0%|          | 0/50000 [00:00<?, ?it/s]
Saving corpus: 0%|          | 0/50000 [00:00<?, ?it/s]
['./wp-heb\sports_adjacent-clean-vocab.txt']
```

```
sports_tokzr = BertTokenizer.from_pretrained(f'{MODEL_DIR}/{MODEL_NAME}-vocab.txt')
```

```
### # ###
```

```
' '.join(sports_tokzr.tokenize('מכבי הפועל כדורגל כדורסל כדורעף טניס'))
```

```
c:\Users\Benzu\anaconda3\Lib\site-packages\transformers\tokenization_utils_base.py:1925: FutureWarning: Calling BertTokenizer.from_pretr
warnings.warn(
'מכבי הפועל כדורגל כדורסל כדורעף טניס'
### # ###
```

