עיבוד שפה טבעית (20225211) 2024-2025 - תרגיל 4 חלק 1 (5 נק')

שלום לכולכן!

בחלק הזה של תרגיל 4 נצבור קצת נסיון מעשי בעבודה עם מודלי שפה מסוג רובוטריק (transformer). נתמקד במודל מסוג **אוטורגרסיבי**, כלומר חוזה-פני-עתיד, 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'יפיטפוט (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.ה סטודנטיות. 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 האגינגפייס, שבעזרתה נוכל לטעון את המודל המאומן ואת הטוקנייזר 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
gcsfs 2024.10.0 requires fsspec==2024.10.0, but you have fsspec 2024.9.0 which is inc
```

נייבא חבילות ומודולים. הרובוטריקים של האגינגפייס 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 שיכוני 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: UserWarnin
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
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 mo
  warnings.warn(
README.md: 100%  661/661 [00:00<00:00, 31.1kB/s]
(…)-00000-of-00001-13f0a33230d64dd9.parquet: 100%  10.7M/10.7M [00:00<00:00, 19.8MB/s]
(…)-00000-of-00001-0177e6dbdee45eef.parquet: 100%  905k/905k [00:00<00:00, 9.67MB/s]
(…)-00000-of-00001-0b94257b623049e5.parquet: 100%  1.10M/1.10M [00:00<00:00, 14.4MB/s]
```

כאמור, אנחנו נעבוד עם מודל GPT-2 (הקישור מוביל לדף הסבר באתר האגינגפייס). 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 וGPT2MLP אשר מורכב משתי שכבות קונבולציה חד ממדיות אקטיבציה (GELU) ונישור (dropout) לכן ההבדל הינו ראש הMLP 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|>',
'unk_token': '<|endoftext|>'}, clean_up_tokenization_spaces=False,
added_tokens_decoder={
        50256: AddedToken("<|endoftext|>", rstrip=False, lstrip=False,
single_word=False, normalized=True, special=True),
}
)
```

נרענן 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 לפונקצית `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 **רווח** 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 לembeddor) 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 כ**התפלגות**.

```
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 sapphires:

```
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 **בלבלות**, 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_idxs) -> float:
  assert len(logits) == len(true_idxs)
  N = len(true_idxs)

  ### 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 ĠâĢ Ĺ 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 Ġrain Ġin ĠSpain Ġstays Ġmainly Ġon Ġthe Ġplain .
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 Ġground
```

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 מתקשיסות 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):
  accumlated_prob = q6(sen, "January")[1] * q6(sen + " January", "20")[1] * q6(sen + " Ja
```

```
  print(f"\n\nOption 1", accumlated_prob)
  print("\n\n")
  accumlated_prob = q6(sen, "November")[1] * q6(sen + " November", "08")[1] * q6(sen + "
  print(f"\n\nOption 2", accumlated_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', 'ĠâĢĵ', 'Ċ', '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%