עיבוד שפה טבעית (20225211) 2023-2024 - תרגיל 4

על ידי: פן אייל - 208722058, קורן אביטבול - 318796448

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
```

```
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 510.5/510.5 kB 6.0 MB/s eta 0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 116.3/116.3 kB 10.6 MB/s eta 0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 134.8/134.8 kB 12.8 MB/s eta 0:00:00
```

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
from torch import exp, topk
```

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_hul
The secret `HF_TOKEN` does not exist in your Colab sec
To authenticate with the Hugging Face Hub, create a tol
You will be able to reuse this secret in all of your n
Please note that authentication is recommended but sti
  warnings.warn(
Downloading readme: 100%          661/661 [00:00<00:00, 25.4kB/s]
Downloading data: 100%        10.7M/10.7M [00:01<00:00, 9.05MB/s]
Downloading data: 100%          905k/905k [00:00<00:00, 5.19MB/s]
Downloading data: 100%        1.10M/1.10M [00:00<00:00, 2.67MB/s]
Generating train split: 100%  76758/76758 [00:00<00:00, 254294.97
Generating valid split: 100%   5996/5996 [00:00<00:00, 87653.91 ex
```

כאמור, אנחנו נעבוד עם מודל GPT-2 (הקישור מוביל לדף הסבר באתר האגינגפייס). צריך לטעון בנפרד את המודל ואת ה**טוקנייזר שלו**, שהוא מודל-עזר שתפקידו להפוך טקסט גולמי לקלט מספרי עבור הרובוטריק. הטוקנייזר של GPT-2 אומן בשיטת קידוד זוגות בתים, או BPE, עליו תוכלו לקרוא בפרק 2.5.2 של סטנפורד, או בחומרי העזר של הקורס, או ממש לעומק בהסרטון הזה שיצא אך החודש.

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, 439B/s]
config.json: 100%                 665/665 [00:00<00:00, 15.8kB/s]
vocab.json: 100%              1.04M/1.04M [00:00<00:00, 7.25MB/s]
merges.txt: 100%                456k/456k [00:00<00:00, 5.89MB/s]
tokenizer.json: 100%          1.36M/1.36M [00:00<00:00, 9.02MB/s]
model.safetensors: 100%        548M/548M [00:06<00:00, 92.8MB/s]
generation_config.json: 100%      124/124 [00:00<00:00, 1.93kB/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 (2 נק'):

1. (0.25) מהו גודל אוצר המילים של המודל?
2. (0.75) `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. (1) שכבת ה-MLP של המודל משתמשת בפונקציית אקטיבציה שלא למדנו עליה. חקרו אודותיה מעט וענו מהי התכונה העיקרית שמבדילה אותה משלוש הפונקציות שאנחנו מכירים.

תשובות:

1. אפשר לראות לפי שכבת הembedding כי גודל אוצר המילים הוא **50,257**
2. נוספה שכבת נרמול לפני שכבת הצומי, ובנוסף שכבת הfeedforward המלאה שונתה להיות multi layer perceptron. כלומר הרחבנו את שכבת הfeedforward להיות כמה שכבות.
3. התכונה העיקרית שמבדילה את הNewGELU מהפונקציות שלמדנו (ReLU, sigmoid, tanh) היא האי-מונוטוניות של הפונקציה.

```
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): GPT2Attention(
          (c_attn): Conv1D()
          (c_proj): Conv1D()
          (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()
          (c_proj): Conv1D()
          (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=True),
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 g
```

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 (אין סיבה שזה יהיה דווקא כך).

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:

1. האלמנטים שמחזירה הפונקציה word_ids() מסמנים את אינדקס המילה במשפט אליה הtoken שייך. דבר זה יהיה חשוב בסיטואציה בה נרצה לזהות מילים או כמה מילים בטקסט, כגון במשימת זיהוי ישויות שבה המילים עצמן מעניינות אותנו, ולא הtokens שמייצגים את היישות.
2. הפלט של הטוקנייזר מכיל את האלמנט 'attention_mask' שהוא מערך בינרי בגודל שווה למספר הטוקנים בטקסט. עבור כל טוקן בטקסט, ערך של 1 ב-attention mask מציין שהטוקן אמור להשפיע על הפלט של המודל, בעוד שערך 0 מציין שהטוקן אינו משפיע.

```
input_toks = tokenizer.batch_encode_plus([story1], return_tensors='pt')
tok_texts = input_toks.tokens()
print(' '.join(tok_texts[:5]))
print(len(input_toks.input_ids[0]))
print(input_toks.input_ids[0][:5])
print(input_toks.word_ids()[:5])

### 2.2 One more element: ###
print(input_toks['attention_mask'])
```

```
H IGH Ġabove Ġthe Ġcity
58
tensor([   39, 18060,  2029,   262,  1748])
[0, 0, 1, 2, 3]
tensor([[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
```

```
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1]])
```

כעת נפעיל את המודל עצמו באמצעות קריאה ישירה אליו על המבנה המפורק (שתי כוכביות) 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)
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]))
```

```
58
50257
-79.91034698486328
```

```
softmaxed_outputs = softmax(generative_outputs[0][0][3], dim=0)  # specify a dimension or
float(softmaxed_outputs[1748])
```

```
0.005037946160882711
```

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 (1):

1. (0.5) כמה מילים מעדיף המודל על-פני city?
2. (0.5) מהי המילה לה הוא מעניק את ההסתברות הגבוהה ביותר? (**לא האינדקס: המילה עצמה**)

```
### Answers for 3 (edit where necessary) ###

### Answer for 3.1 ###

bigger_then_city = np.sum([1 for i in range(len(softmaxed_outputs)) if softmaxed_outputs[i
print("1. The model preffer", bigger_then_city, "words more than the word 'city'")

### Answer for 3.2 ###

def best_token_id(sm_output) -> int:
  return np.argmax(sm_output.detach().numpy())


def token_from_index(idx: int) -> str:
  '''
  Hint: look at the tokenizer
  '''
  vocab = tokenizer.get_vocab()
  return list(vocab.keys())[list(vocab.values()).index(idx)]


best_in_outputs = best_token_id(softmaxed_outputs)
print("2.",token_from_index(best_in_outputs))
```

```
1. The model preffer 13 words more than the word 'city'
2. 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 (2).

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 ###

  log_probs = log_softmax(logits, dim=-1)
  true_log_probs = log_probs[range(N), true_idxs]
  return exp(-true_log_probs.mean()).item()

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

```
12650.2314453125
```

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 Ï He Ġis Ġas Ġbeautiful Ġas Ġa Ġweather c
43444.23828125
```

```
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 .
11037.453125
```

```
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
0.00023369934933725744
0.11648908257484436
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 הסייענים 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) (1):

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

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

  ### Q5 - edit here ###
  sentence_toks = tokenizer.batch_encode_plus([sentence], return_tensors='pt')
  gen_outs = model(**sentence_toks)[0][0]
  sm = softmax(gen_outs[-1], dim=0)
  return best_token_id(sm)
```

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

```
4171 Ġblue
1865 Ġyet
3269 ĠJanuary
```

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. (1)

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

בונוס: 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 ###

def is_year(token):
  predicted_year = token
  if len(predicted_year) > 1 and predicted_year[0] == 'Ġ':  # Check for prefix
    predicted_year = predicted_year[1:]  # Remove prefix if present
  try:
    if len(predicted_year) == 4:
      # try to cast to int
      predicted_year = int(predicted_year)
      return token
  except ValueError:
    return False

def most_prob_continuation_year(sentence: str) -> int:

  ### Q5 - edit here ###
  sentence_toks = tokenizer.batch_encode_plus([sentence], return_tensors='pt')
  gen_outs = model(**sentence_toks)[0][0]
  sm = softmax(gen_outs[-1], dim=0)

  top_k = 100
  top_probs, top_inds = topk(sm, k=top_k)

  for i, idx in enumerate(top_inds):
      if (predicted_year := is_year(token_from_index(idx.item()))):
          print(f"Found token: {idx.item()}, {predicted_year}")
          return i
          break  # Exit loop if year is found

location = most_prob_continuation_year(sentences_to_complete[-1])
print(f"From location {location} we recieve a year type")
print("2008 is the correct answer")
```

```
Found token: 3648, Ġ2008
From location 27 we recieve a year type
2008 is the correct answer
```

נחשב:

P(next word in the sentence is November) * P(next word is 2008 | last word in the sentence is November)

```
### Bonus ###
def calc_correct_next_word_prob(sentence, word):

  sentence_toks = tokenizer.batch_encode_plus([sentence], return_tensors='pt')
  gen_outs = model(**sentence_toks)[0][0]
  sm = softmax(gen_outs[-1], dim=0)

  top_k = 100
  top_probs, top_inds = topk(sm, k=top_k)

  prob = 0
  # For correct next word, get the probability:
  for i, idx in enumerate(top_inds):
    if token_from_index(idx) == word:
      prob = top_probs[i].item()
      break

  return prob

sentence = sentences_to_complete[-1]
month = 'ĠNovember'
month_prob = calc_correct_next_word_prob(sentence, month)

sentence = sentences_to_complete[-1] + " November"
year = 'Ġ2008'
year_prob = calc_correct_next_word_prob(sentence, year)

# Overall the probability that given the original input the model will sample the correct month followed by the correct year is:
print("The overall probability is: ", month_prob * year_prob)
```

```
The overall probability is:  0.0009448270019450922
```