تمرین سوم

اعضای گروه:

علی درخشش
محراب مرادزاده
ابوالفضل ملک احمدی

بخش اول – مدل زبانی n-gram

ما در این بخش یک مدل بر پایه مدل زبانی n-gram آموزش دادیم ’ که این مدل برای پیش بینی کلمه بعد مورد استفاده و آموزش داده شده است .

در ابتدا با درخواست زدن به مخزن مورد نظر داده های مربوطه را جهت شروع کار استخراج میکنیم :

```
import requests
import json
from tqdm import tqdm

base_url = "https://github.com/language-ml/course-nlp-ir-1-text-exploring/raw/main/exploring-datasets/health/"  # Replace with your JSON URL
file_names = ['hidoctor-1.json', 'hidoctor-2.json', 'hidoctor-3.json', 'hidoctor-4.json', 'hidoctor-5.json',
              'namnak-1.json', 'namnak-2.json', 'namnak-3.json', 'namnak-4.json', 'namnak-5.json']
url_list = [base_url + file_name for file_name in file_names]
all_paragraphs = []

for url in tqdm(url_list):
    response = requests.get(url)
    data = json.loads(response.text)
    for dict_ in data:
        all_paragraphs += dict_['paragraphs']
```

پس از بارگیری مجموعه داده جملات هر پاراگراف را با استفاده از SentenceTokenizer موجود در کتابخانه ی hazm جدا می شوند و سپس جملات نرمال می شوند و نیز چون در داخل جملات یکسری علامت های نگارشی و سایر علائم (از جمله ایموجی و غیره) وجود دارد آنها را نیز حذف میکنیم :

```
def __preprocess(self, all_paragraphs):
    # Text preprocessing

    # Extract sentences from paragraphs :
    sentences_list = []
    for paragraph in tqdm(all_paragraphs, desc = 'Sentences tokenization'):
        sentences_list += sent_tokenize(paragraph)

    # Normalize + remove extra denotations like :
    normalized_sentences_list = []
    normalizer = Normalizer()
    for sentence in tqdm(sentences_list, desc = 'Normalization'):
        normalized_sentence = re.sub('[:,،.<>/!@#$%^~{}();»«…“”"؟:◊♦–\*\+_\^]', ' ', sentence)
        normalized_sentence = normalizer.normalize(normalized_sentence)
        normalized_sentences_list.append(normalized_sentence)

    # Extract tokens from normalized sentences
    self.all_tokens = [] # <s> and </s> tags are added
    for sentence in tqdm(normalized_sentences_list, desc = 'Tokenization'):
        self.all_tokens.append("<s>")
        temp_tokens = word_tokenize(sentence)
        # علامت های خیلی زیادی وجود داره و پیدا کردن همشون سخته فکر کنم تنها کلمه تک حرفی توی فارسی 'و' باشه
        # پس به غیر از 'و' همه توکن های تک حرفی رو حذف کنیم
        for token in temp_tokens:
            if len(token) > 1 or token == 'و':
                self.all_tokens.append(token)
        self.all_tokens.append("</s>")

    self.remaked_corpus = ' '.join(self.all_tokens) # use this for search and count
```

توجه یکی دیگر از کار های که در رشته کد بالا مورد توجه است توکن کردن است زیرا ما باید مرز جملات را مشخص کنیم به خاطر اینکه کلمه اخر یک جمله با کلمه اول جمله بعد را به عنوان یک bigram نگیرد و با هم احتمالش را حساب کند زیرا این دو با هم مرتبط نیست .
پ.ن: برای حذف علائم نگارشی و سایر علائم غیر کاربردی از انجایی که تنها تک حرف داخل فارسی «و» است پس انرا نگه داشته و مابقی را حذف میکنیم.

بخش اول – مدل زبانی n-gram

معرفی مدل

- تابع fit

در این قسمت ابتدا ما یک dataframe شامل ngram و count میسازیم که داخل این dataframe بسته به این که مقدار n برابر ۱ باشد تعداد 1-gramها اگر ۲ باشد تعداد 2-gram ها و اگر ۳ باشد تعداد 3-gram ها به همین صورت تا به بالا قرار میگیرد و تعداد هر کدام را شمرده و در ستون count قرار میدهد :
نکته: ۱. چون nltk.ngrams از روی توکن ها ngramهای تکراری نیز تولید میکند از set استفاده شده است .
۲. برای محاسبه تعداد هر ngram از تابع findall رجکس استفاده شده است .

```python
def fit(self, input_corpus):
    '''
    input_corpus is a list(each item should be a string) of paragraphs.
    '''

    self.__preprocess(input_corpus)

    df_ngrams = pd.DataFrame(columns = ['ngram', 'count'])
    df_indx = -1
    for ngram in tqdm(set(ngrams(self.all_tokens, self.n)), desc = f'{self.n}_grams calculation'): # use set to ignore repeated ones
        if '<s>' not in ngram and '</s>' not in ngram:
            str_ngram = ' '.join(ngram)
            try :
                pattern = r'\b' + str_ngram + r'\b' # it's so importatnt because of preventing sub simple words
                count = len(re.findall(pattern, self.remaked_corpus))
                if count > 0:
                    df_indx += 1
                    df_ngrams.loc[df_indx, 'ngram'] = str_ngram
                    df_ngrams.loc[df_indx, 'count'] = count
            except:
                print("Error at :")
                print(str_ngram)

    self.df_ngrams = df_ngrams
    self.df_ngrams.sort_values(by = 'count', ascending = False, inplace = True)
```

- تابع set_pre_probs

توجه این مدل فقط نیاز به یکبار آموزش دیدن دارد و در دفعات بعد کاربر بدون نیاز به آموزش مدل و تنها با داشتن جدول تشکیل شده در بالا میتواند از مدل استفاده کند :

```python
def set_pre_probs(self, ngram_file_name):
    '''
    set pre calculated ngrams from file.
    ngram_file is a .csv with "ngram", "count" as columns
    '''
    self.df_ngrams = pd.read_csv(ngram_file_name)
    self.df_ngrams.sort_values(by = 'count', ascending = False, inplace = True)
```

(parameter) top_n: Any

بخش اول – مدل زبانی n-gram

- تابع generate

در این تابع ابتدا جمله ورودی پیش پردازش میشود و از انجایی که نرمالایز hazm فاصله ها را حدف میکند و ما برای اینکه بفهمیم که کاربر کلمه اخر خود را کامل داده یا ناقص نیاز است که فاصله ها را داشته باشی زیرا ما با استفاده از فاصله ها تشخیص میدهیم که کلمه اخر کامل یا ناقص است. در نتیجه مقدار correct_spacing را false در نظر میگیریم و در مرحله بعد کامل بودن یا نبودن کلمه اخر را تشخیص می دهیم

```
def generate(self,input_text, top_n):
    '''
    input_text : user input text
    top_n : top n words to show
    '''
    input_text = re.sub('[:,،.<>/!@#$%^&*{}();؛»«…“”"؟:؛()♦–\*\+_\^]', ' ', input_text)
    normalizer = Normalizer(correct_spacing = False)
    input_text = normalizer.normalize(input_text)

    input_text_tokens = input_text.split()
    last_word = input_text_tokens[-1] # no worries with space --> split doesn't count last space

    last_incomplete = True
    if input_text[-1] == ' ': # So the last word is complete
        last_incomplete = False

    if not last_incomplete: #complete
        if (self.n -1 > len(input_text_tokens)):
            raise Exception(f"input text must be longer than {self.n} words")
    else: #incomplete
        if (self.n > len(input_text_tokens)):
            raise Exception(f"input text must be longer than {self.n} words")
```

حال اگر کلمه کامل باشد و n برابر ۱ باشد در اینجا باید بیشترین احتمال ها بدون در نظر گرفتن قبلی ها برگردانیم و کلمه ای که بیشترین احتمال را داشته بر میگردانیم و اگر n بیشتر از ۱ بود از آن ترکیب های محاسبه شده(به عنوان مثال ترکیب های دوتایی یا ترکیب های سه تایی محاسبه شده)استفاده میکنیم

```
if not last_incomplete: # last complete
    if self.n == 1:
        df_output['ngram'] = df_output['ngram'].str.split().str[-1]
        return df_output.head(top_n).values
    else:
        input_ngram_list = input_text_tokens[-(self.n-1):]
        str_input_ngram = ' '.join(input_ngram_list)
        df_output = df_output[df_output['ngram'].str.startswith(str_input_ngram + " ")]
        df_output['ngram'] = df_output['ngram'].str.split().str[-1]
        return df_output.head(top_n).values
```

برای اینکه کلمه بعد به صورت ناقص امده باشد روند بالا را طی میکنیم با این تفاوت که فاصله اضافه شده برای کلمات کامل را در این قسمت در نظر نمیگیریم چون میخواهیم ابتدا کلمه ناقص را پیش بینی کنیم

```
else: # incomplete
    if self.n == 1:
        df_output = df_output[df_output['ngram'].str.startswith(last_word)]
        df_output['ngram'] = df_output['ngram'].str.split().str[-1]
        return df_output.head(top_n).values
    else:
        input_ngram_list = input_text_tokens[-self.n:] # no need to subtract by 1 because last incomplete
        str_input_ngram = ' '.join(input_ngram_list)
        df_output = df_output[df_output['ngram'].str.startswith(str_input_ngram)]
        df_output['ngram'] = df_output['ngram'].str.split().str[-1]
        return df_output.head(top_n).values
```

حال برای استفاده از مدل به صورت زیر عمل میکنیم :

```
model = Generator_ngram(2)
model.set_pre_probs('2_gram_probs.csv') # read from trained & saved ngrams
text = input()
model.generate(text, 5)
```

ابتدا n مورد نظر برای مدل را مشخص کرده که در این جا مقدار ۲ در نظر گرفته شده سپس فایل پیش بینی شده از ترکیب ها که در اینجا ترکیب های دوتایی که محاسبه شده بود خوانده شده حال جمله مورد نظر را به آن میدهیم که نتایج بدست آمده به صورت زیر است :

N=1:

```
در انتظار تاک

array([['216 ,'تاکید'],
       ['113 ,'تاکنون'],
       ['14 ,'تاکسول'],
       ['12 ,'تاکیکاردی'],
       ['9 ,'تاکی']], dtype=object)

باید به آنجا

: array([['68488 ,'و'],
        ['48098 ,'به'],
        ['46933 ,'از'],
        ['46705 ,'در'],
        ['32546 ,'را']], dtype=object)
```

N=2:

```
تجویز قرص ضد

: array([['212 ,'التهابی'],
        ['131 ,'التهاب'],
        ['116 ,'عفونی'],
        ['87 ,'آفتاب'],
        ['82 ,'باکتری']], dtype=object)
```

N=3:

```
یکی از علائم این بیماری عفونت دستگاه

array([['9 ,'ادراری'],
       ['8 ,'گوارش'],
       ['4 ,'تنفسی'],
       ['4 ,'تناسلی']], dtype=object)
```

- اثبات چرایی استفاده از تعداد محاسبه شده به جایی احتمال :

$$P(x_1, x_2, x_3) = p(x_1)p(x_2|x_1)p(x_3|x_2, x_1)$$

$$p(x_3|x_2, x_1) = \frac{p(x_1, x_2, x_3)}{p(x_1)p(x_2|x_1)}$$

حالا با استفاده از روابط بالا در مثال زیر فرضیه خود را ثابت میکنیم

مثال :

x= افزایش نرخ تورم سال جاری----افزایش نرخ تورم سال جاری

برای محاسبه احتمال جمله به این صورت است که :

$P(x) = p(\text{افزایش})p(\text{نرخ}|\text{افزایش})p(\text{تورم}|\text{افزایش ,نرخ})p(\text{سال}|\text{افزایش ,نرخ ,تورم})p(\text{جاری}|\text{افزایش ,نرخ ,تورم ,سال})$

حالا فرض کنید جاری را نداشته حال برای اینکه این کلمه را پیش بینی کنیم باید کلمه ای را انتخاب کنیم که احتمال کل جمله را بیشینه کند

$$P(x) = p(\text{افزایش})p(\text{نرخ}|\text{افزایش})p(\text{تورم}|\text{افزایش ,نرخ})p(\text{سال}|\text{افزایش ,نرخ ,تورم})p(y|\text{افزایش ,نرخ ,تورم ,سال})$$

به طور مثال اگر به جای y سه کلمه داشته باشیم مثل جاری ،برای ،شیر حال ما احتمال کلمه جمله را برای تک تک این کلمات حساب میکنیم

$$P(x) = p(\text{افزایش})p(\text{نرخ}|\text{افزایش})p(\text{تورم}|\text{افزایش ,نرخ})p(\text{سال}|\text{افزایش ,نرخ ,تورم})p(\text{جاری}|\text{افزایش ,نرخ ,تورم ,سال})$$

$$P(x) = p(\text{افزایش})p(\text{نرخ}|\text{افزایش})p(\text{تورم}|\text{افزایش ,نرخ})p(\text{سال}|\text{افزایش ,نرخ ,تورم})p(\text{برای}|\text{افزایش ,نرخ ,تورم ,سال})$$

$$P(x) = p(\text{افزایش})p(\text{نرخ}|\text{افزایش})p(\text{تورم}|\text{افزایش ,نرخ})p(\text{سال}|\text{افزایش ,نرخ ,تورم})p(\text{شیر}|\text{افزایش ,نرخ ,تورم ,سال})$$

با توجه به اینکه ترم های قبل ترم اخر یکی است میتوان کل انها را نادیده گرفت و فقط ترم اخر را در نظر گرفت و برای ما مهم ست

$$P(x) = \cancel{p(\text{افزایش})p(\text{نرخ}|\text{افزایش})p(\text{تورم}|\text{افزایش ,نرخ})p(\text{سال}|\text{افزایش ,نرخ ,تورم})}p(\text{جاری}|\text{افزایش ,نرخ ,تورم ,سال})$$

$$P(x) = \cancel{p(\text{افزایش})p(\text{نرخ}|\text{افزایش})p(\text{تورم}|\text{افزایش ,نرخ})p(\text{سال}|\text{افزایش ,نرخ ,تورم})}p(\text{برای}|\text{افزایش ,نرخ ,تورم ,سال})$$

$$P(x) = \cancel{p(\text{افزایش})p(\text{نرخ}|\text{افزایش})p(\text{تورم}|\text{افزایش ,نرخ})p(\text{سال}|\text{افزایش ,نرخ ,تورم})}p(\text{شیر}|\text{افزایش ,نرخ ,تورم ,سال})$$

حال با استفاده از قضیه مارکوف می دانیم که به جای اینکه کل $p(x)$ رو در نظر بگیریم اگر $n=1$ باشد y مستقل از همه ی قبلی ها میشه و بنابراین در این ترم $p(y|\text{افزایش ,نرخ ,تورم ,سال})$ برابر خود $p(y)$ است پس کافی است مقدار زیر را محاسبه کنیم و کلمه ای را پیدا کنیم که بیشترین احتمال را داشته باشد :

$$p(y) \rightarrow argmax\ p(y)$$

حال اگر $n=2$ باشد یعنی مارکوف مرتبطه دوم که یعنی فقط به کلمه قبلی بستگی دارد پس کافیه ما برای مثال بالا عبارت زیر را محاسبه کنیم

$$P(x) = p(\text{جاری}|\text{سال}) = \frac{p(\text{جاری ,سال})}{p(\text{سال})}$$

$$P(x) = p(\text{برای}|\text{سال}) = \frac{p(\text{برای ,سال})}{p(\text{سال})}$$

$$P(x) = p(\text{شیر}|\text{سال}) = \frac{p(\text{شیر ,سال})}{p(\text{سال})}$$

که مخرج تمام این کسر ها برابر است که میتوان ان را نادیده گرفت و فقط ترم صورت کسر را باید بیشینه کنیم به صورت زیر :

$$P(x) = p(\text{سال}|\text{جاری}) = p(\text{جاری .سال}) \sim count(\text{جاری .سال})$$
$$P(x) = p(\text{سال}|\text{برای}) = p(\text{برای .سال}) \sim count(\text{برای .سال})$$
$$P(x) = p(\text{سال}|\text{شیر}) = p(\text{شیر .سال}) \sim count(\text{شیر .سال})$$

حال برای اینکه این احتمال ها را به دست بیاوریم باید تعداد مثلا جا های که به عنوان مثلا «سال» در کنار «جاری» امده را بشماریم و بر تعداد کل کلمات داخل جمله تقسیم کنیم که احتمال این ترم را به دست بیاوریم برای مابقی هم به همین صورت که از انجایی که باز هم مخرج سه ترم به یک صورت است فقط نیاز است تعداد جاهای که این کلمات کنار هم امده است را محاسبه کنیم .

بخش دوم –مدل زبانی عمیق

در این بخش مدل عمیق مورد بررسی قرار می‌گیرد.

برای اجرا این بخش ما از vps با مشخصات ۷۲ گیگابایت رم، یک کارت گرافیک Nvidia RTX ۳۰۹۰ و ۱۲هسته پردازشی از HPC دانشگاه شریف استفاده کردیم.

- تابع make dataset و make dataset v۲

برای استفاده از دیتاست خام و آماده کردن آن (توکنایز کردن و نرمال کردن) در ابتدا ما با استفاده از تابعی که دانشجویان ترم‌های گذشته برای این کار تدارک دیده بودند استفاده کردیم. زمان استفاده شده برای بر روی HPC دانشگاه ۱۸ ساعت بود.

در تابع make dataset v۲ بهبودهایی بر روی تابع اولیه شکل گرفت و مرتبه زمانی تابع از $O(n^3)$ به $O(n^2)$ کاهش یافت و زمان اجرا بر روی HPC به کمتر از ۳ دقیقه کاهش یافت. همچنین بهینه سازی‌های دیگری نیز در نرمال سازی جملات صورت گرفت مانند حذف علائم نگارشی و ... .

```python
def make_dataset_v2(file_path):
    all_paragraphs = []
    for file in glob.glob(file_path):
        with open(file, 'r') as f:
            print(f'====== file_name: {file} =======')
            if re.match(r'.*\.json$', file):
                data = json.load(f)
                for dict_ in data:
                    all_paragraphs += dict_['paragraphs']
            elif re.match(r'.*\.csv$', file):
                data = pd.read_csv(f)
                all_paragraphs += data['text'].tolist()
    print(len(all_paragraphs))
    # Extract sentences from paragraphs :
    sentences_list = []
    for paragraph in tqdm(all_paragraphs, desc = 'Sentences tokenization'):
        sentences_list += sent_tokenize(paragraph)
    # Normalize + remove extra denotations like :
    normalized_sentences_list = []
    normalizer = Normalizer()
    for sentence in tqdm(sentences_list, desc = 'Normalization'):
        normalized_sentence = re.sub('[:,،.<>/!@#$%~{}();»«…“”"؟:۰♦-\*\+_\^]', ' ', sentence)
        normalized_sentence = normalizer.normalize(normalized_sentence)
        normalized_sentences_list.append(normalized_sentence)
    print(len(normalized_sentences_list))
    with open("dataset_sentences_v2", "wb") as fp:
            pickle.dump(normalized_sentences_list, fp)
```

همچنین برای غنی‌تر کردن دیتاست‌های مورد استفاده، داده‌های سایت‌های العربیه و lastsecond کراول شده‌اند که کدهای کراوال در فایل‌های ژوپیتر lastsecond_scraper و DataCrawling موجود می‌باشد.

مدل پایه

بهترین مدل زبانی موجود در زبان فارسی در حال حاضر مدل GPT۲ بلبل‌زبان می‌باشد. همچنین میتوان از مدل‌های زبانی جدیدتر و متن بازی مانند LLaMa هاگینگ فیس و Alpaca نیز استفاده کرد اما به دلیل حجم بالای وزن ها و همچنین

محدودیت سخت افزاری موجود (حتی در صورت استفاده از کارت گرافیک‌های HPC دانشگاه) این امر امکان پذیر نیست. لذا مدل GPT۲ بهترین مدل در دسترس است.

آموزش مدل GPT۲ بر روی دیتاست پایه

در ابتدا ما مدل GPT۲ را بر روی دیتاست پایه‌ای که فقط شامل متون سایت های Hidoctor و namnak بود تنظیم دقیق کردیم. این امر در ۸ epoch انجام شد.
تنظیم دقیق مدل با استفاده از تابع Trainer و TrainingArguments کتابخانه Transformers انجام شد.
همچنین قابل ذکر است با توجه به استفاده از یک GPU ۳۰۹۰ ما قادر بودیم تا از تمامی جملات دیتاست به طور کامل استفاده کنیم.

```
# train Model
training_args = TrainingArguments(output_dir='./results', num_train_epochs=epochs, logging_steps=50, save_steps=3000,
                                  per_device_train_batch_size=batch_size, per_device_eval_batch_size=batch_size,
                                  warmup_steps=10, weight_decay=0.05, logging_dir='./logs', report_to = 'none')

trainer = Trainer(model=model,  args=training_args, train_dataset=train_dataset,
        eval_dataset=val_dataset, data_collator=lambda data: {'input_ids': torch.stack([f[0] for f in data]),
                                                              'attention_mask': torch.stack([f[1] for f in data]),
                                                              'labels': torch.stack([f[0] for f in data])}).train()

model.save_pretrained("./gpt2_namnak_hidoctor/model_base_on_namnak_hidoctor_dataset_v2")
```

در مرحله بعد مدل توسط دیتاست غنی‌تر شده شامل کلمات پایه و کلمات سایت‌های العربیه و lastsecond یک بار دیگر در ۶ epoch تنظیم دقیق شد.

روش های رمزگشایی و نتایج

برای بررسی نتایج همان‌طور که در این سایت اشاره شده بود چندین روش برای دیکودینگ وجود دارد که با استفاده از کتابخانه Transformers این روش‌ها به سادگی قابل پیاده‌سازی هستند.

```
def predict_next(model, tokenizer, text, kind, num, max_length):
    # Encode a text inputs
    generator = pipeline('text-generation', model=model, tokenizer=tokenizer,  pad_token_id=tokenizer.eos_token_id)
    if kind == 'greedy':
        outputs = generator(text, max_length=max_length, num_return_sequences=num)
    elif kind == 'beams':
        outputs = generator(text, max_length=max_length, num_beams=5, num_return_sequences=num)
    elif kind == 'random_sampling':
        outputs = generator(text, max_length=max_length, top_k=0, do_sample=True, temperature=0.7, num_return_sequences=num)
    elif kind == 'text_p_sampling':
        outputs = generator(text, max_length=max_length, top_k=0, top_p=0.92, do_sample=True, num_return_sequences=num)

    elif kind == 'text_k_sampling':
        outputs = generator(text, max_length=max_length, top_k=40, do_sample=True, num_return_sequences=num)
    return outputs
```

حال با استفاده از تابع predict_next می‌توانیم نتایج خروجی مدل‌های تنظیم دقیق شده را مشاهده کنیم.

```
tokenizer_base = AutoTokenizer.from_pretrained('bolbolzaban/gpt2-persian')
model_base = GPT2LMHeadModel.from_pretrained('bolbolzaban/gpt2-persian')

samples = [ 'احتمال خطر سکته های ',
          'احتمال خطر سکته های م',
          'بهترین روش برای غلبه بر استرس ',
          'بهترین روش برای غلبه بر استرس ن',
          'بهترین روش برای غلبه بر استرس نوشیدن ',
        ]
num = 5
kind = 'greedy'
max_length = 20
num_words = 2
for sample in samples:
    print('input : ', sample)
    s_len = len(sample.split(' '))
    predictions = predict_next(model_base, tokenizer_base, sample, kind , num, max_length)
    for p in predictions:
        preds = p['generated_text'].split()
        print(' '.join(preds[s_len-1:s_len-1+num_words]))
    print('................................')
```

در زیر خروجی مدل‌های پایه، تنظیم دقیق شده بر روی دیتاست پایه، و دیتاست غنی با استفاده از رمزگشایی greedy را مشاهده می‌کنید:

بخش دوم –مدل زبانی عمیق

```
input :  احتمال خطر سکته های
کوییک هم
کمتر از
فو در
فو، در
کویلر و
.....................................
input :  احتمال خطر سکته های م
مگالوبلاستیک در
مگالوبلاستیک در
مگالوبلاستیک در
مگالوبلاستیک بسیار
مقلبی در
.....................................
input :  بهترین روش برای غلبه بر استرس
و نگرانی
امتحان کنید
تعطیلات
،و اضطراب
در تعطیلات
.....................................
input :  بهترین روش برای غلبه بر استرس ن
ننوشیدن آب
ننوشیدن الکل
ننوشیدن الکل.
ننوشیدن الکل
ننوشیدن الکل
```

مدل تنظیم دقیق شده بر روی دیتاست پایه

```
input :  احتمال خطر سکته های
فو به
فو در
فو با
فوبیا در
فو سکته
.....................................
input :  احتمال خطر سکته های م
مگالوبلاستیک هم
مگالوبلاستیک در
ماحتمال و
مگالوبلاستیک کم
مگالوبلاستیک در
.....................................
input :  بهترین روش برای غلبه بر استرس
امتحان این
از جمله
و غلبه
و افسردگی
.....................................
input :  بهترین روش برای غلبه بر استرس ن
ننوشیدن آب
ننوشیدن آب
ننوشیدن قهوه
ننوشیدن قهوه
ننوشیدن آب
```

مدل پایه GPT۲

```
input :  احتمال خطر سکته های
ساله از
درصد بیشتر
قلبی -
درصد وجود
های قلبی
.....................................
input :  احتمال خطر سکته های م
مغیر قلبی
محدب مغزی
م ادر
مشد ید
متواتر پیش
.....................................
input :  بهترین روش برای غلبه بر استرس
و خستگی
و بهبود
است.
و اضطراب
و کاهش
.....................................
input :  بهترین روش برای غلبه بر استرس ن
ننشینید.
ننوشیدن آب
ننوشیدن آب
ننوشیدن آب
ننوشیدن آب
```

مدل تنظیم دقیق شده بر روی دیتاست غنی

تنظیم دقیق مدل با استفاده از فریز کردن لایه‌ها

همان‌طور که در مقاله آقای **Jeremy Howard** اشاره شده است با استفاد از فریز کردن تدریجی لایه‌ها و استفاده از نرخ یادگیری متفاوت برای هر لایه می‌توانیم مدل را تنظیم دقیق کنیم.
تابع splitter لایه‌های مدل را به چهار قسمت تقسیم می‌کند:

```
def splitter(model):
    "Split a GPT2 `model` in 3 groups for differential learning rates."

    # First layers group : decoder blocks from 0 to 3
    modules = []
    for i in range(4): modules.append(model.transformer.h[i])
    groups = [nn.Sequential(*modules)]

    # Second layers group : decoder blocks from 4 to 7
    modules = []
    for i in range(4,8,1): modules.append(model.transformer.h[i])
    groups = L(groups + [nn.Sequential(*modules)])

    # Third layers group : decoder blocks from 8 to 11
    modules = []
    for i in range(8,12,1): modules.append(model.transformer.h[i])
    groups = L(groups + [nn.Sequential(*modules)])

    groups = L(groups + [nn.Sequential(model.transformer.wte,model.transformer.wpe,model.transformer.ln_f)])

    return groups.map(params)
```

و نرخ یادگیری را برای هر لایه تنظیم میکنیم، برای لایه‌های ابتدایی نرخ یادگیری کم و برای لایه‌های انتهایی این نرخ افزایش می‌یابد.

```
param_groups = []
learning_rates = [1e-4,2e-4,4e-4,5e-4]
param_splitter = splitter(model)
for i in range(0, len(param_splitter)):
    parameters_of_group_layer = param_splitter[i]
    for parameter in parameters_of_group_layer:
        param_groups.append({'params': [parameter], 'lr': learning_rates[i]})
```

فریز کردن لایه‌ها

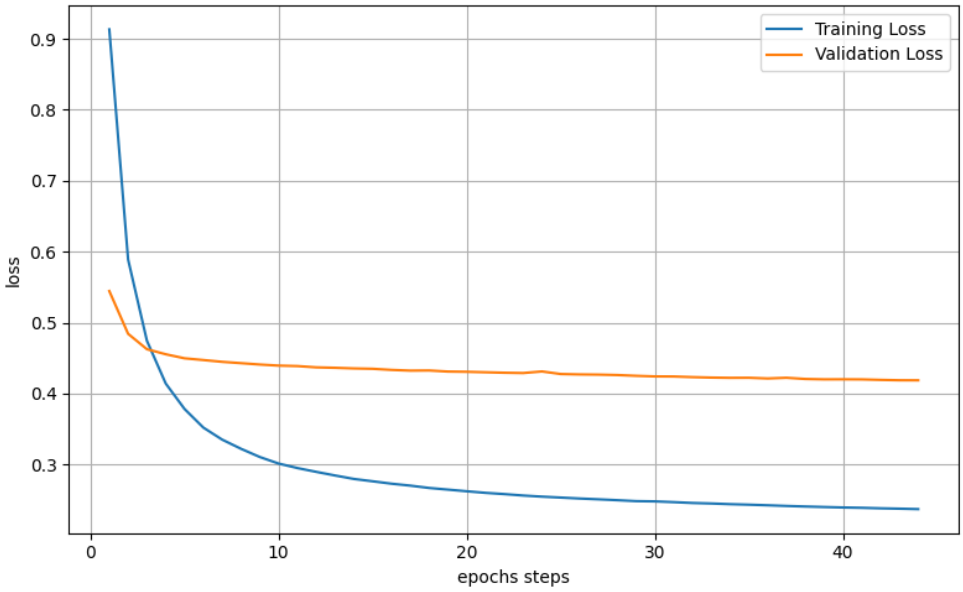
روش دیگری که در مقاله اشاره می‌شود فریز کردن مرحله به مرحله تعدادی از لایه‌ها در ایپاک‌های مختلف است. در ایپاک اول لایه‌های آغازین فریز میشوند و در ایپاک های بعدی لایه های بعدی فریز می‌گردند.

```
# Gradual Layer Freezing
if epoch_i == 0:
    param_splitter = splitter(model)
    parameters_layer2 = param_splitter[1]
    for param in parameters_layer2:
        param.requires_grad = False
    plot_loss(train_losses, val_losses, epoch_i)

if epoch_i == 1:
    param_splitter = splitter(model)
    parameters_layer3 = param_splitter[2]
    for param in parameters_layer3:
        param.requires_grad = False
    plot_loss(train_losses, val_losses, epoch_i)

if epoch_i == 2:
    param_splitter = splitter(model)
    parameters_layer4 = param_splitter[3]
    for param in parameters_layer4:
        param.requires_grad = False
    plot_loss(train_losses, val_losses, epoch_i)
```

در نهایت مدل با استفاده از دیتاست غنی در epoch ۳ تنظیم دقیق شد و نتایج به در زیر آمده است.

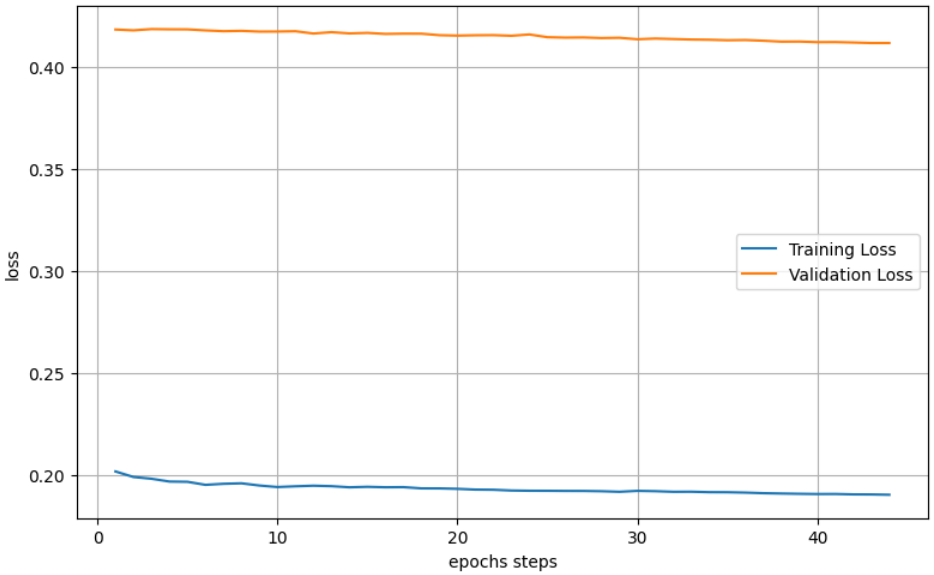
ایپاک اول:

Average training loss: ۰.۴۷
Perplexity: ۱.۶۱
Validation Loss : ۰.۴۲
Perplexity: ۱.۵۲



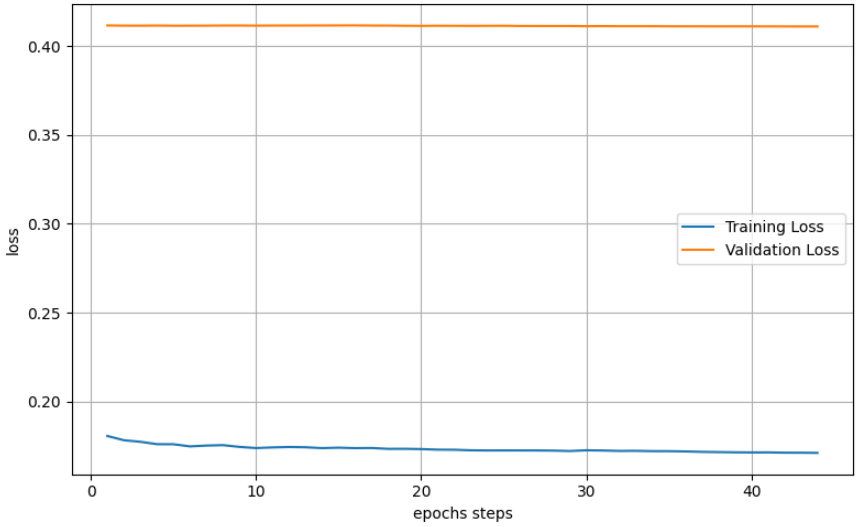
ایپاک دوم:

Average training loss: ۰.۳۸
Perplexity: ۱.۴۶
Validation Loss : ۰.۴۱
Perplexity: ۱.۵۱



ایپاک سوم:

Average training loss: ۰.۳۴
Perplexity: ۱.۴۱
Validation Loss : ۰.۴۱
Perplexity: ۱.۵۱



نتایج خروجی این مدل در زیر قابل مشاهده است:

```
input :  احتمال خطر سکته های
قلبی و
قلبی را
قلبی ناشی
- مغزی
پرتروميين هم
.....................................
input :  احتمال خطر سکته های م
مغذی برای
مغزی را
مگالوبلاستیک <<unk>sndof<unk>e
مگالوبلاستیک در
مغیر قلبی
.....................................
input :  بهترین روش برای غلبه بر استرس
خود چه
تان در
با جلوگیری
ی که
های روزانه
.....................................
input :  بهترین روش برای غلبه بر استرس ن
ننشینین و
ننشینین و
ننشینید <<unk>sndof<unk>e
ننوشیدن الکل
نپوشیدن لباسهای
.....................................
input :  بهترین روش برای غلبه بر استرس نوشیدن
یک فنجان
چای سیاه
```