به نام خدا

دانشگاه تهران

پردیس دانشکده‌های فنی

دانشکده برق وکامپیوتر

درس پردازش زبان های طبیعی

پروژه شماره : 5

نام و نام خانوادگی : محمدرضا بختیاری

شماره دانشجویی : 810197468

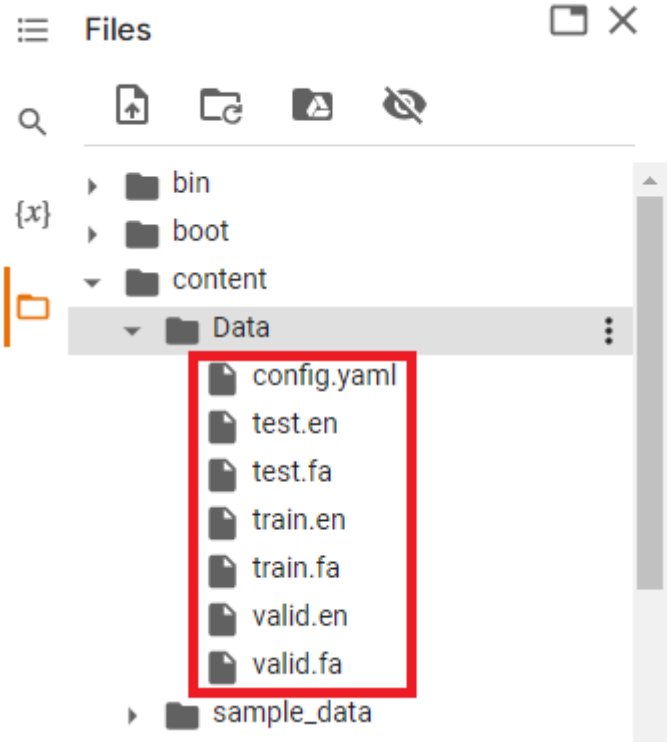
خرداد 1401

فهرست سوالات

1-Basic settings

پیش از شروع کار و اجرای کد ها لازم است برخی تنظیمات اولیه انجام شوند تا کد ها به درستی و بدون خطا اجرا شوند.

ابتدا لازم است در محیط google colab وارد شده و از طریق پوشه ی Files وارد پوشه ی content شده و یک پوشه جدید با نام Data ایجاد کرده و 6 فایل موجود در مجموعه دیتا[1] بعلاوه فایل config.yaml که شامل پیکر بندی های لازم است را در داخل این پوشه همانند تصویر زیر بارگذاری[2] کرده :



شکل 1-1 : محل آپلود فایل های نام برده در پوشه ی Data

[1] Dataset
[2] upload

2-Pre-Processing

پیش از تولید یک مدل زبانی , ابتدا نیاز داریم متناسب با هدف طبقه بند پیش پردازش های لازم را انجام دهیم.

برای شروع ابتدا نیاز است تنظیمات لازم را انجام دهیم و پکیج های مورد نیاز در طول سوال را نصب کنیم :

Setup

```
[1]  !pip install nltk
     !pip install random
     !pip install spellchecker
     !pip install pyspellchecker
     !pip install bpe
```

شکل 2-1 : نصب پکیج های مورد نیاز

در ادامه پکیج های مورد نیاز را وارد[1] می کنیم :

Importing important packages

```
[2]  import nltk
     nltk.download('punkt')
     from nltk import word_tokenize
     import random
     from spellchecker import SpellChecker
     import re
     from typing import Dict, Tuple
     from bpe import Encoder
```

شکل 2-2 : ایمپورت کردن پکیج های مورد نیاز

[1] Import

پس از آن نیاز است داده های آموزش[1] , آزمون[2] و اعتبارسنجی[3] را بارگیری کنیم :

Loading dataset

```
[3]  # Loading source dataset
     file_train_en = open("Data/train.en", "r")
     train_en = file_train_en.read()
     file_test_en = open("Data/test.en", "r")
     test_en = file_test_en.read()
     file_valid_en = open("Data/valid.en", "r")
     valid_en = file_valid_en.read()
     # Loading target dataset
     file_train_fa = open("Data/train.fa", "r")
     train_fa = file_train_fa.read()
     file_test_fa = open("Data/test.fa", "r")
     test_fa = file_test_fa.read()
     file_valid_fa = open("Data/valid.fa", "r")
     valid_fa = file_valid_fa.read()
```

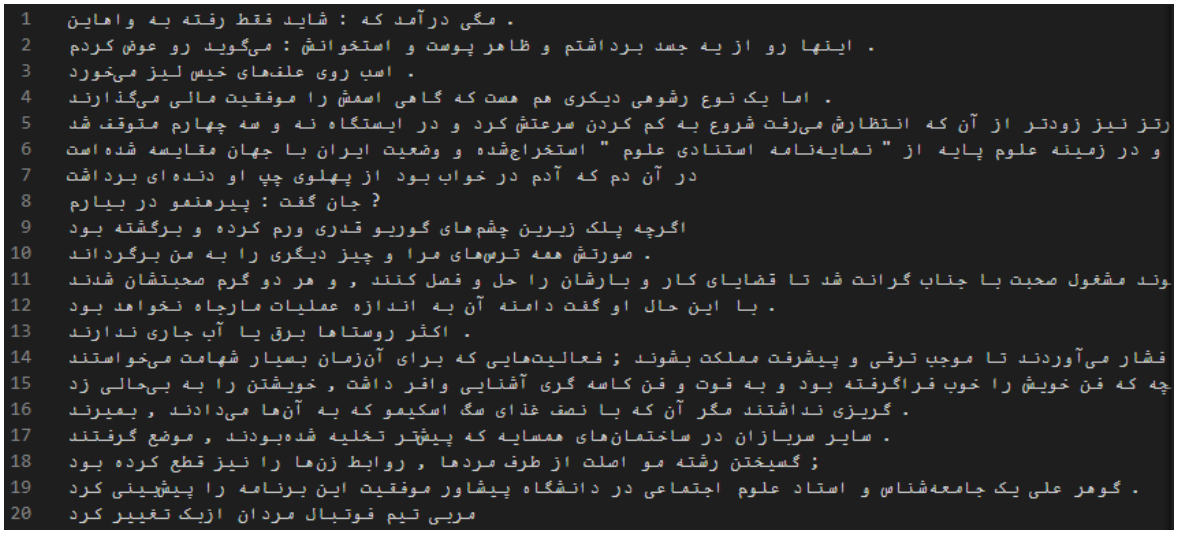
شکل 3-2 : بارگیری دادگان آموزش , آزمون و اعتبارسنجی

حال نگاهی گذرا به دادگان آموزش مبدا[4] (در اینجا دادگان انگلیسی) و مقصد[5] (در اینجا دادگان فارسی) می اندازیم تا پیش پردازش های مناسب را در ادامه انتخاب کنیم :

```
maybe hes just gone into wahine , meggie suggested .
i assembled this from a corpse , remolding its flesh and bones .
the horse slipped on the wet grass
but there is a pale shade of bribery which is sometimes called prosperity .
all too soon , the hogwarts express was pulling in at platform nine and three quarters .
for this purpose , scientific information of iranian origin in form of articles dealing with basic scienc
while he slept took a rib from his left side
take off my shirt ?
though goriot 's eyes seemed to have shrunk in their sockets
his face brought back all my fear and then some .
when edmund , being engaged apart in some matter of business with dr. grant , which seemed entirely to en
however , he said it would not be a marjah - like offensive .
most villages have no electricity or running water .
who urged governments and local businesses to adopt progressive ideas that were too daring for the time .
for the young rascal , being expert at a variety of feints and dodges
could do no less than die on half the ration of the husky .
other soldiers took positions on evacuated adjacent buildings .
the bond once broken on the side of the men , it was loosed between the women ;
sociologist gohar ali , a university of peshawar social sciences teacher , predicted success for the prog
uzbek men " s soccer team gets new coach
```

شکل 4-2 : بیست جمله اول از دادگان آموزش مبدا

[1] Train
[2] Test
[3] Validation
[4] Source
[5] Target



شکل 5-2 : بیست جمله اول از دادگان آموزش مقصد

حال نمونه هایی از پیش پردازش های مرسوم در پردازش زبان های طبیعی را مشاهده می کنیم و سپس متناسب با هدف نهایی که آموزش یک مدل ترجمه ماشینی است , پیش پردازش های مناسب را انتخاب و پیاده سازی می کنیم.

0- BPE[1]

الگوریتم BPE به دنبال پیدا کردن یک جفت کاراکتر در هر مرحله و ترکیب آن ها و اضافه کردن واژه جدید به مجموعه واژگان هست. این الگوریتم در هر مرحله جفت کاراکتری را انتخاب می کند که بیشترین تکرار را در متن داشته باشد[2].

پیاده سازی این الگوریتم را یک بار به صورت دستی و یک بار هم با استفاده از کتابخانه های موجود انجام می دهیم.

-پیاده سازی BPE به صورت دستی[3] :

در گام اول از روی مجموعه نوشته های[4] داده شده , لغات اولیه که شامل حروف یکتای به کار برده شده در متن است را استخراج می کنیم و مجموعه واژگان[5] را تشکیل می دهیم.

پیش از شروع کار به انتهای هر لغت , کاراکتر '_' را اضافه می کنیم تا مشخص کننده ی پایان هر لغت باشد.

[1] Byte Pair Encoding
[2] Most frequent pair
[3] Implement BPE manually
[4] Corpus
[5] Vocabulary

در هر مرحله دو جفت واژه ای که با بیشترین تکرار پشت سر هم می آیند را پیدا کرده و پس از ترکیب آن ها و مشخص کردن آن ها به عنوان یک واژه جدید , آن جفت را به مجموعه واژگان اضافه کرده و به همین ترتیب مراحل را ادامه می دهیم.

شرط پایان این فرایند را می توانیم به دو صورت در نظر بگیریم. می توانیم تعداد تکرار[1] مشخصی را در نظر بگیریم و پس از تکرار شدن حلقه به تعداد مورد نظر , فرایند را خاتمه دهیم. یا می توانیم از ابتدا شرط مشخصی بر روی تعداد واژه های موجود در مجموعه واژگان در نظر بگیریم.

A- *Implement BPE manually*

```
[19] # Find all unique letters in the corpus
     def Find_Vocabulary(Dictionary):
         Vocabulary = []
         for word in Dictionary:
             for string in word:
                 Vocabulary.append(string)
         Vocabulary = list(dict.fromkeys(Vocabulary))
         Vocabulary.remove(' ')
         return Vocabulary

     # Find all most frequent pairs in the corpus
     def Find_Most_Frequent_Pair(My_Dictionary: Dict[str, int]) -> Dict[Tuple[str, str], int]:
         Pair = {}
         for Vocab, Frequency in My_Dictionary.items():
             Letter = Vocab.split()
             for i in range(len(Letter) - 1):
                 pair = (Letter[i], Letter[i + 1])
                 New_Frequency = Pair.get(pair, 0)
                 Pair[pair] = New_Frequency + Frequency
         return Pair

     # Merge most frequent Consecutive letters into one word
     def Combine_Vocabulary(Most_Frequent_Pair: Tuple[str, str], Input_Vocabulary: Dict[str, int]) -> Dict[str, int]:
         Merged_Vocabulary = {}
         pattern = re.escape(' '.join(Most_Frequent_Pair))
         Substitution = ''.join(Most_Frequent_Pair)
         for Input_Word in Input_Vocabulary:
             Output_Word = re.sub(pattern, Substitution, Input_Word)
             Merged_Vocabulary[Output_Word] = Input_Vocabulary[Input_Word]
         return Merged_Vocabulary
```

شکل 6-2 : پیاده سازی الگوریتم BPE به صورت دستی

-پیاده سازی BPE با استفاده از کتابخانه :

یک بار هم پیاده سازی این الگوریتم را با استفاده از کتابخانه های موجود انجام می دهیم :

B- *Implement BPE using library*

```
[20] encoder = Encoder(200, pct_bpe=0.88)  # params chosen for demonstration purposes
     encoder.fit(train_en.split('\n'))
```

شکل 7-2 : پیاده سازی BPE با استفاده از کتابخانه (بر روی دادگان آموزش مبدا)

[1] Iteration

حال برای مثال یک جمله از زبان مبدا را به عنوان ورودی می دهیم و خروجی را مشاهده می کنیم :

```
[21] # Let's see an example of the first sentence in the train corpus
     example = "maybe hes just gone into wahine , meggie suggested ."
     print(encoder.tokenize(example))
     print(next(encoder.transform([example])))
     print(next(encoder.inverse_transform(encoder.transform([example]))))
```

شکل 8-2 : اعمال BPE برو روی یک جمله از دادگان مبدا

و خروجی زیر را به همراه تعداد تکرار و همچنین جمله ی ورودی پس از اعمال BPE مشاهده می کنیم:

['__sow', 'ma', 'y', 'be', '__eow', '__sow', 'he', 's', '__eow', '__sow', 'j', 'us', 't', '__eow', '__sow', 'go', 'ne', '__eow', '__sow', 'in', 'to', '__eow', '__sow', 'wa', 'hi', 'ne', '__eow', ',', '__sow', 'me', 'g', 'g', 'ie', '__eow', '__sow', 'su', 'g', 'ge', 'st', 'ed', '__eow', '.']
[25, 95, 42, 88, 24, 25, 14, 20, 24, 25, 169, 109, 28, 24, 25, 189, 78, 24, 25, 9, 6, 24, 25, 156, 71, 78, 24, 3, 25, 73, 40, 40, 137, 24, 25, 158, 40, 132, 59, 53, 24, 4]
“maybe hes just gone into wahine , meggie suggested .”

ما در این پروژه از پیاده سازی دوم یعنی پیاده سازی با استفاده از کتابخانه استفاده خواهیم کرد.

1- Lowercasing

یکی از مهم ترین پیش پردازش ها به خصوص برای تسک ترجمه ماشینی , کوچک کردن حروف بزرگ یا همان Lowercasing است.

در واقع حروف بزرگ حاوی اطلاعاتی هستند که برای تسک تشخیص هویت های اسمی[1] مفید هستند اما حاوی هیچ اطلاعاتی برای Stemming نیستند و در واقع حروف بزرگ با اثر Stemming مقابله می کنند , لازم است که این پیش پردازش انجام شود و حروف بزرگ را به کوچک تبدیل کنیم.

1- *Lowercasing*

```
[4]  # Lowercasing source dataset
     train_en = train_en.lower()
     test_en = test_en.lower()
     valid_en = valid_en.lower()
     # Lowercasing target dataset
     train_fa = train_fa.lower()
     test_fa = test_fa.lower()
     valid_fa = valid_fa.lower()
```

شکل 9-2 : کوچک کردن حروف بزرگ دادگان با استفاده از دستور lower().

[1] Named Entity Recognition

در واقع در زبان انگلیسی , مواردی مانند : اسامی خاص , کلمات ابتدای جمله , ضمیر اول شخص ‘I’ , ... همواره با حروف بزرگ آغاز می شوند و نیاز است که برای تسک ترجمه ماشینی این حروف به حروف کوچک تبدیل شوند.

اگر چه در فارسی نمایش های مختلف برای حروف داریم (مثل : غـ_ـغـ_ـغ_ـغ) اما حروف بزرگ و کوچک در فارسی معنایی ندارند و علت نمایش های مختلف یک حرف در کلمات , قرار گرفتن در جایگاه های مختلف در کلمه است.

پس این پیش پردازش (Lowercasing) در این بخش برای متون[1] فارسی جایگاهی نداشته و در متون انگلیسی از آن استفاده می کنیم.

توجه : البته با توجه به این که در دیتاست موجود Lowercasing از پیش رعایت شده است , دیگر نیازی به اعمال این پیش پردازش نخواهیم داشت.

2- Remove Extra Whitespaces

یکی دیگر از پیش پردازش های مورد توجه در ترجمه ماشینی , حذف فضاهای اضافی موجود است که در متون انگلیسی از آن به عنوان فضاهای سفید[2] (مثلا ”I go to school “ که با حذف فضای قبل از I به ”I go to school“ تبدیل می شود) و در فارسی از آن به عنوان فاصله / نیم فاصله یاد می شود.

2- *Remove Extra Whitespaces*

```
[5]  def remove_whitespace(text):
         return  " ".join(text.split())

     # Removing Extra Whitespaces in source dataset
     train_en = remove_whitespace(train_en)
     test_en = remove_whitespace(test_en)
     valid_en = remove_whitespace(valid_en)
     # Removing Extra Whitespaces in target dataset
     train_fa = remove_whitespace(train_fa)
     test_fa = remove_whitespace(test_fa)
     valid_fa = remove_whitespace(valid_fa)
```

شکل 10-2 : حذف کردن فاصله های اضافی دادگان

این فاصله ها هیچ اطلاعات اضافی به ما نمی دهند و فقط منجر به افزایش اندازه ی متن شده و برای جلوگیری از این امر , با استفاده از تابع نشان داده شده , این فضاهای اضافی را حذف می کنیم.

[1] Corpus
[2] Whitespaces

3- Tokenization

یکی دیگر از پیش پردازش های مهم و ضروری در پردازش زبان , توکنایز کردن متن ورودی است. به این صورت که متن داده شده را به قسمت های کوچک تری به نام توکن[1] تقسیم می کنیم.

3- *Tokenization*

```
[6]  # Tokenization of source dataset
     train_en_token = word_tokenize(train_en)
     test_en_token = word_tokenize(test_en)
     valid_en_token = word_tokenize(valid_en)
     # Tokenization of target dataset
     train_fa_token = word_tokenize(train_fa)
     test_fa_token = word_tokenize(test_fa)
     valid_fa_token = word_tokenize(valid_fa)
```

شکل 11-2 : توکنایز کردن مجموعه دادگان

حال نگاهی به توکن های تولید شده در زبان مبدا و مقصد می اندازیم :

```
[7]  # Let's take a look at some random tokens
     print("Some of the english tokens at the train data set :")
     print(random.choices(train_en_token, k=10))
     print("Some of the persian tokens at the train data set :")
     print(random.choices(train_fa_token, k=10))

     Some of the english tokens at the train data set :
     ['biting', 'even', 'recently', '2002', 'work', 'giorgio', 'with', 'nasty', 'black', 'technology']
     Some of the persian tokens at the train data set :
     ['را', 'بود', '.', 'منفی', 'از', '8', 'کودک', ',', 'ایدز', 'اتمی']
```

شکل 12-2 : 10 نمونه تصادفی از توکن های تولید شده در زبان مبدا و مقصد

ما در این قسمت بنا به نیاز و تسک موجود , از توکنایز کردن متن در سطح کلمه[2] استفاده کردیم. (در حالت کلی امکان توکنایز کردن در سطح جمله و کاراکتر نیز وجود دارد)

4- Spelling Correction

یکی دیگر از مهم ترین پیش پردازش ها در ترجمه ماشینی , تصحیح املای کلمات است. اشتباهات تایپی در دادگان متنی رایج است و ممکن است بخواهیم قبل از انجام تجزیه و تحلیل , اشتباهات املایی موجود را تصحیح کنیم که تضمین می کند ما نتایج بهتری را از مدل خود دریافت خواهیم کرد.

[1] Token
[2] Word-level tokenization

4- *Spelling Correction*

```
[9]  def spell_check(text):
         result = []
         spell = SpellChecker()
         for word in text:
             correct_word = spell.correction(word)
             result.append(correct_word)
         return result

     # Spelling Correction of source dataset
     train_en_split = train_en.split()
     train_en_Spelling_Correction = spell_check(train_en_split)
     test_en_split = test_en.split()
     test_en_Spelling_Correction = spell_check(test_en_split)
     valid_en_split = valid_en.split()
     valid_en_Spelling_Correction = spell_check(valid_en_split)
     # Spelling Correction of target dataset
     train_fa_split = train_fa.split()
     train_fa_Spelling_Correction = spell_check(train_fa_split)
     test_fa_split = test_fa.split()
     test_fa_Spelling_Correction = spell_check(test_fa_split)
     valid_fa_split = valid_fa.split()
     valid_fa_Spelling_Correction = spell_check(valid_fa_split)
```

شکل 13-2 : تصحیح املای کلمات با استفاده از دستور SpellChecker()

در ادامه یک نمونه از تصحیح املا را مشاهده می کنیم :

```
[6]  # Let's try an example of spelling correction
     text = "confuson matrx".split()
     spell_check(text)

     ['confusion', 'matrix']
```

شکل 13-2 : یک نمونه از تصحیح املای کلمات انگلیسی

برای انجام یک ترجمه ی خوب , نیازمند این هستیم که دادگان موجود دارای کمترین میزان خطا باشند تا مدل بتواند به خوبی آموزش ببیند و به دقت بالاتری دست پیدا کنیم , پس تشخیص و بعد از آن تصحیح خطاهای موجود , یک امر مهم و ضروری در تسک ترجمه ماشینی است.

5- Removal of Tags

اگر مجموعه دادگان ما به صورتی باشد که شامل تگ های متعدد (تگ های HTML , ...) باشد به نحوی که تگ های موجود بارمعنایی نداشته باشند و اطلاعات اضافه تری در اختیار ما قرار ندهند , بهتر

است که از این پیش پردازش استفاده شود و تگ های اضافی حذف شوند تا از بزرگ شدن متن هم جلوگیری شود.

5- *Removal of Tags*

```
[18] def remove_tag(text):
         text=' '.join(text)
         html_pattern = re.compile('<.*?>')
         return html_pattern.sub(r'', text)

     # Removal of Tags of source dataset
     train_en = remove_tag(train_en.split())
     test_en = remove_tag(test_en.split())
     valid_en = remove_tag(valid_en.split())
     # Removal of Tags of target dataset
     train_fa = remove_tag(train_fa.split())
     test_fa = remove_tag(test_fa.split())
     valid_fa = remove_tag(valid_fa.split())
```

شکل 2-14 : حذف برخی تگ های اضافی

حال یک نمونه از حذف تگ را مشاهده می کنیم :

```
[19] # Let's see an example of "Removal of Tags"
     text = "<HEAD> this is head tag </HEAD>"
     print("The input text is :")
     print(text,"\n")
     print("The output text after removing tags is :")
     print(remove_tag(text.split()))

     The input text is :
     <HEAD> this is head tag </HEAD>

     The output text after removing tags is :
      this is head tag
```

شکل 2-15 : یک نمونه متن ورودی به همراه خروجی آن پس از حذف تگ

در اینجا به علت این که متن ما به شکلی بود که شامل تگ های خاصی نبود , الزامی به استفاده از این پیش پردازش نخواهیم داشت.

6- Other Pre-processing

پیش پردازش های دیگری نظیر Removal of , Removing Punctuations , Stopwords Removal Removal of URLs , Lemmatization , Stemming , Frequent Words هم وجود دارد که بعضا از اهمیت کمتری برخوردار بوده اند و یا متناسب با هدف نهایی ما (ساخت مدل ترجمه ماشینی) نبوده و در مواردی حتی منجر به کاهش دقت مدل خواهند شد.

-اهمیت پیش پردازش های ذکر شده در زبان فارسی و انگلیسی و مقایسه ی آن ها :

در این قسمت با توجه به توضیحات مفصلی که در هر بخش برای هر پیش پردازش ارائه دادیم , به طور کوتاه به بررسی اهمیت هر کدام در زبان فارسی و انگلیسی می پردازیم.

0- BPE : از منظر این که زبان فارسی از نظر ریخت شناسی[1] قوی تر و از نظر ساخت واژی[2] نیز قوی تر و پیچیده تر از زبان انگلیسی است , استفاده از پیش پردازش BPE در زبان فارسی از جایگاه بالاتری برخوردار بوده و اهمیت بیشتری نسبت به زبان انگلیسی دارد.

1- Lowercasing : همان طور که پیش از این به طور کامل در خصوص این پیش پردازش توضیحاتی ارائه دادیم , در زبان انگلیسی , مواردی مانند : اسامی خاص , کلمات ابتدای جمله , ضمیر اول شخص 'I' , ... همواره با حروف بزرگ آغاز می شوند و نیاز است که برای تسک ترجمه ماشینی این حروف به حروف کوچک تبدیل شوند. اگر چه در فارسی نمایش های مختلف برای حروف داریم (مثل : غـ_ـغـ_ـغ_غ) اما حروف بزرگ و کوچک در فارسی معنایی ندارند و علت نمایش های مختلف یک حرف در کلمات , قرار گرفتن در جایگاه های مختلف در کلمه است. پس این پیش پردازش (Lowercasing) در این بخش برای متون فارسی جایگاهی نداشته و در متون انگلیسی از آن استفاده می کنیم.

پس با توجه به توضیحات ارائه شده , پیش پردازش Lowercasing در زبان انگلیسی حائز اهمیت بوده و در زبان فارسی جایگاهی ندارد.

2- Remove Extra Whitespaces : وجود فاصله و نیم فاصله در زبان فارسی به مراتب بیشتر از زبان انگلیسی بوده و از این حیث این دو با هم قابل مقایسه نیستند. (موارد و چالش های بسیاری در خصوص فاصله و نیم فاصله در زبان فارسی وجود دارد , مانند : [می روم – میروم] , [کتابخانه – کتاب خانه] , ...)

در نتیجه پیش پردازش Remove Extra Whitespaces در زبان فارسی نسبت به زبان انگلیسی از اهمیت بیشتری برخوردار است.

3- Tokenization : از این منظر که زبان فارسی از نظر ریخت شناسی بسیار قوی تر از زبان انگلیسی بوده , فلذا :

پیش پردازش توکنیزاسیون در زبان فارسی جایگاه بالاتری داشته و از اهمیت بیشتری هم برخوردار خواهد بود.

4- Spelling Correction : با توجه به این که زبان فارسی از نظر ساخت واژی قوی تر و پیچیده تر از زبان انگلیسی می باشد (به عنوان مثال نمایش مختلف حروف در زبان فارسی تنوع به مراتب بیشتری نسبت به زبان انگلیسی دارد , برای نمونه در فارسی برای نمایش حرف Z در انگلیسی از 4 حرف { ز , ذ ,

[1] Morphology
[2] Lexical

ض , ظ } استفاده می کنیم که همین تنوع حروف می تواند منجر به خطا های املایی شوند.) اگر چه در زبان انگلیسی هم اشتباهات رایج املایی اعم از جابجایی جفت حروف [e – i] , [a – i] , [u – o] , ... را داریم اما باتوجه به توضیحات داده شده :

خطاهای املایی در زبان فارسی از تنوع و تعداد بیشتری برخوردار بوده و در نتیجه پیش پردازش Spelling Correction اهمیت بیشتری در زبان فارسی خواهد داشت.

5- Removal of Tags : با توجه به فراخور زبان انگلیسی و متون انگلیسی استفاده از تگ نسبت به متون فارسی بیشتر رایج بوده درنتیجه :

در پیش پردازش Removal of Tags اهمیت زبان انگلیسی بیشتر از فارسی خواهد بود.

-پیش پردازش های انتخاب شده نهایی :

در نهایت با توجه به توضیحات مفصل برای هر کدام از پیش پردازش های ذکر شده و اهمیت آن ها در تسک ترجمه ماشینی , پیش پردازش های **BPE** , **Remove Extra Whitespaces** و **Spelling Correction** را انتخاب می کنیم.

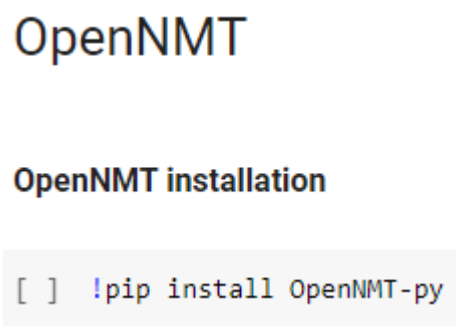
3-Tools

در ادامه از دو ابزار OpenNMT و FairSeq جهت آموزش مدل ترجمه ماشینی استفاده می کنیم.

OpenNMT

با استفاده از منابع آموزشی داده شده در صورت پروژه , به پیاده سازی این ابزار می پردازیم.

در ابتدا نیاز داریم تا مطابق شکل زیر این ابزار را بر روی کولب نصب کنیم :



شکل 3-1 : نصب ابزار OpenNMT بر روی گوگل کولب

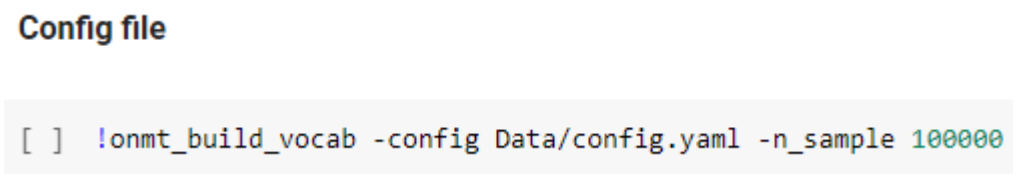
بعد از آن نوبت به پیکر بندی[1] و تنظیمات دقیق پارامتر های مدل می رسد.

پیکر بندی های گوناگون و به فراخور آن تنظیمات گوناگونی می توانیم برای پارامتر های مدل در نظر بگیریم , که در ادامه به یکی از آن ها می پردازیم.

لازم به ذکر است که برای اعمال پیکر بندی های لازم , به دو روش می توان اقدام کرد که فرایند هر دو روش را در ادامه خواهیم دید.

-پیکر بندی با استفاده از اعمال دستورات و مقداردهی پارامتر ها در فایل YAML.

در این روش اعمال دستورات (اعم از محل ذخیر مدل و ...) و مقداردهی پارامتر های مدل (اعم از میزان dropout یا اندازه ی هر batch و ...) را در یک فایل YAML. ذخیره کرده و در نهایت پیکربندی های لازم را اعمال می کنیم :



شکل 3-2 : پیکر بندی و تنظیمات پارامتر های مدل

توجه : پیکر بندی های انجام شده در قالب فایل config.yaml در پوشه ی Data موجود است.

[1] Configuration

-پیکر بندی و مقداردهی پارامتر های مدل به صورت مستقیم

در این روش تنظیمات لازم و مقداردهی پارامتر های مدل را به صورت مستقیم انجام می دهیم.

به عنوان نمونه یکی از موارد برای ست کردن پارامتر های مدل را مشاهده می کنیم :

```
!python OpenNMT-py/train.py -data OpenNMT-py/data/demo -save_model OpenNMT-py/data/model/model -layers 6 -rnn_size 512 -word_vec_size 512 -transformer_ff 2048 -heads 8 -encoder_type transformer -decoder_type transformer -position_encoding -train_steps 200000 -max_generator_batches 2 -dropout 0.1 -batch_size 4096 -batch_type tokens -normalization tokens -accum_count 2 -optim adam -adam_beta2 0.998 -decay_method noam -warmup_steps 8000 -learning_rate 2 -max_grad_norm 0 -param_init 0 -param_init_glorot -label_smoothing 0.1 -valid_steps 1000 -save_checkpoint_steps 1000 -world_size 1 -gpu_rank 0
```

در اینجا 20 پارامتر برای مدل مقدار دهی شده اند که به تفکیک 10 مورد از آن ها را توضیح می دهیم.

Model-layers : مشخص کننده ی تعداد لایه های مدل است. که این تعداد در این جا به صورت پیش فرض 6 انتخاب شده است.

Rnn-size : تعیین کننده ی اندازه ی شبکه ی عصبی ما (در اینجا شبکه عصبی بازگشتی[1]) می باشد که به صورت پیش فرض مقدار 512 برای آن در نظر گرفته شده است.

Train-steps : نشان دهنده ی تعداد گام های لازم در هنگام آموزش مدل است که در اینجا به صورت پیش فرض 200000 انتخاب شده است که در ادامه خواهیم دید با انتخاب مقادیر کمتر نیز به دقت خوبی بر روی دادگان آموزش خواهیم رسید. (با انتخاب تعداد گام 80000 به دقت حدود 97% بر روی دادگان آزمون خواهیم رسید که قابل قبول می باشد)

Dropout : تعیین کننده ی این است که چه تعداد نورون[2] در هر لایه به هنگام آموزش مدل خاموش می شود که این مقدار به صورت پیش فرض 0.1 فرض شده است.

Batch-size : می دانیم مدل , دادگان را به صورت دسته دسته استفاده می کند و این پارامتر مشخص کننده ی اندازه ی هر دسته می باشد که در اینجا 4096 در نظر گرفته شده است.

Learning-rate : در یادگیری ماشینی و آمار، نرخ یادگیری یک پارامتر تنظیم در یک الگوریتم بهینه‌سازی است که اندازه گام را در هر تکرار در حالی که به سمت حداقل یک تابع ضرر[3] حرکت می‌کند، تعیین می‌کند.

Valid-steps : این پارامتر نیز همانند پارامتر *train-steps* می باشد که پیش تر توضیح داده شد , با این تفاوت که تعداد گام بر روی دادگان اعتبار سنجی را مشخص می کند.

[1] Recurrent neural network
[2] Neuron
[3] Loss function

Save-checkpoint-steps : تعیین کننده ی این است که در طول آموزش مدل , بعد از هر چند گام مدل آموخته شده تا آن لحظه را ذخیر کنیم.

World-size & gpu-rank : در این جا *World-size* و *gpu-rank* ترتیب 1 و 0 انتخاب شده اند که نشان دهنده ی آموزش مدل بر روی یک GPU می باشد که می توان با تغییر این مقادیر تعداد GPU استفاده شده در حین آموزش مدل را نشان داد.

پس از آن به بررسی پارامتر هایی که در کیفیت خروجی یک مدل ترجمه ماشینی تاثیرگذار است ولی امکان تغییر یا تنظیم آن ها باتوجه به منابع موجود وجود ندارد می پردازیم.

با توجه به محدودیت های سخت افزاری در کل پارامتر هایی که منجر به بزرگ شدن مدل می شوند همانند : *Batch-size* , *Embedding-size* , ... می توانند منجر به بزرگ تر شدن مدل , طولانی تر شدن آموزش مدل , اشغال رم بیشتر و در کل اشغال حجم بیشتر شوند.

پس از ست کردن پارامتر های لازم و پیکر بندی های اعمال شده در قسمت قبل حال نوبت به آموزش مدل بر روی دادگان آموزش می رسد.

Training

```
[16] !onmt_train -config Data/config.yaml
```

شکل 3-3 : آموزش مدل

حال به صورت کامل مشخصات مدل ایجاد شده را مشاهده می کنیم :

```
[2022-06-07 07:14:56,206 INFO] Missing transforms field for corpus_1 data, set to default: [].
[2022-06-07 07:14:56,206 WARNING] Corpus corpus_1's weight should be given. We default it to 1 for you.
[2022-06-07 07:14:56,206 INFO] Missing transforms field for valid data, set to default: [].
[2022-06-07 07:14:56,206 INFO] Parsed 2 corpora from -data.
[2022-06-07 07:14:56,206 INFO] Get special vocabs from Transforms: {'src': set(), 'tgt': set()}.
[2022-06-07 07:14:56,206 INFO] Loading vocab from text file...
[2022-06-07 07:14:56,206 INFO] Loading src vocabulary from Data/run/example.vocab.src
[2022-06-07 07:14:56,256 INFO] Loaded src vocab has 33033 tokens.
[2022-06-07 07:14:56,271 INFO] Loading tgt vocabulary from Data/run/example.vocab.tgt
[2022-06-07 07:14:56,364 INFO] Loaded tgt vocab has 38965 tokens.
[2022-06-07 07:14:56,380 INFO] Building fields with vocab in counters...
[2022-06-07 07:14:56,428 INFO]  * tgt vocab size: 38969.
[2022-06-07 07:14:56,509 INFO]  * src vocab size: 33035.
[2022-06-07 07:14:56,512 INFO]  * src vocab size = 33035
[2022-06-07 07:14:56,512 INFO]  * tgt vocab size = 38969
[2022-06-07 07:14:56,516 INFO] Building model...
[2022-06-07 07:15:08,009 INFO] NMTModel(
  (encoder): RNNEncoder(
    (embeddings): Embeddings(
      (make_embedding): Sequential(
        (emb_luts): Elementwise(
          (0): Embedding(33035, 500, padding_idx=1)
        )
      )
    )
    (rnn): LSTM(500, 500, num_layers=2, dropout=0.3)
  )
  (decoder): InputFeedRNNDecoder(
    (embeddings): Embeddings(
      (make_embedding): Sequential(
        (emb_luts): Elementwise(
          (0): Embedding(38969, 500, padding_idx=1)
        )
      )
    )
    (dropout): Dropout(p=0.3, inplace=False)
    (rnn): StackedLSTM(
      (dropout): Dropout(p=0.3, inplace=False)
      (layers): ModuleList(
        (0): LSTMCell(1000, 500)
        (1): LSTMCell(500, 500)
      )
    )
    (attn): GlobalAttention(
      (linear_in): Linear(in_features=500, out_features=500, bias=False)
      (linear_out): Linear(in_features=1000, out_features=500, bias=False)
    )
  )
  (generator): Sequential(
    (0): Linear(in_features=500, out_features=38969, bias=True)
    (1): Cast()
    (2): LogSoftmax(dim=-1)
  )
)
[2022-06-07 07:15:08,010 INFO] encoder: 20525500
[2022-06-07 07:15:08,010 INFO] decoder: 44765969
[2022-06-07 07:15:08,010 INFO] * number of parameters: 65291469
[2022-06-07 07:15:08,011 INFO] Starting training on GPU: [0]
[2022-06-07 07:15:08,011 INFO] Start training loop and validate every 10000 steps...
[2022-06-07 07:15:08,011 INFO] corpus_1's transforms: TransformPipe()
[2022-06-07 07:15:08,012 INFO] Weighted corpora loaded so far:
```

شکل 4-3 : مشخصات کامل مدل ایجاد شده

آموزش مدل را بر روی 80000 گام[1] انجام می دهیم و در نهایت دقت را بر روی دادگان آموزش و اعتبارسنجی مشاهده می کنیم.

به عنوان مثال دقت مدل را در طی 400 گام ابتدایی بر روی دادگان آموزش مشاهده می کنیم :

[1] Step

```
[2022-06-09 05:23:29,998 INFO] Step 50/80000; acc:   3.69; ppl: 576887.83; xent: 13.27; lr: 1.00000; 7878/8255 tok/s;
[2022-06-09 05:23:36,632 INFO] Step 100/80000; acc:   4.05; ppl: 40801.12; xent: 10.62; lr: 1.00000; 8244/8659 tok/s;
[2022-06-09 05:23:43,855 INFO] Step 150/80000; acc:   5.05; ppl: 6200.69; xent: 8.73; lr: 1.00000; 7957/8365 tok/s;
[2022-06-09 05:23:50,757 INFO] Step 200/80000; acc:   6.60; ppl: 3636.94; xent: 8.20; lr: 1.00000; 7712/8164 tok/s;
[2022-06-09 05:23:57,802 INFO] Step 250/80000; acc:   6.88; ppl: 2612.12; xent: 7.87; lr: 1.00000; 7881/8276 tok/s;
[2022-06-09 05:24:04,612 INFO] Step 300/80000; acc:   7.84; ppl: 1930.21; xent: 7.57; lr: 1.00000; 7857/8301 tok/s;
[2022-06-09 05:24:11,685 INFO] Step 350/80000; acc:   8.12; ppl: 1708.15; xent: 7.44; lr: 1.00000; 7827/8196 tok/s;
[2022-06-09 05:24:18,864 INFO] Step 400/80000; acc:   8.77; ppl: 1474.62; xent: 7.30; lr: 1.00000; 7694/8065 tok/s;
```

شکل 5-3 : دقت مدل در 400 گام ابتدایی بر روی دادگان آموزش

پس از اتمام آموزش مدل , دقت نهایی را بر روی دادگان آموزش و اعتبارسنجی مشاهده می کنیم :

```
[2022-06-09 08:41:50,918 INFO] Step 79700/80000; acc:  97.51; ppl:  1.09; xent: 0.09; lr: 0.12500; 7280/7670 tok/s;
[2022-06-09 08:41:58,509 INFO] Step 79750/80000; acc:  96.85; ppl:  1.12; xent: 0.11; lr: 0.12500; 7412/7759 tok/s;
[2022-06-09 08:42:05,964 INFO] Step 79800/80000; acc:  97.11; ppl:  1.11; xent: 0.10; lr: 0.12500; 7747/8079 tok/s;
[2022-06-09 08:42:20,809 INFO] Step 79900/80000; acc:  96.97; ppl:  1.11; xent: 0.11; lr: 0.12500; 7274/7716 tok/s;
[2022-06-09 08:42:28,303 INFO] Step 79950/80000; acc:  97.17; ppl:  1.10; xent: 0.10; lr: 0.12500; 7352/7710 tok/s;
[2022-06-09 08:42:35,618 INFO] Step 80000/80000; acc:  96.95; ppl:  1.12; xent: 0.11; lr: 0.06250; 7229/7643 tok/s;
[2022-06-09 08:42:36,226 INFO] Validation perplexity: 6032.97
[2022-06-09 08:42:36,226 INFO] Validation accuracy: 29.7977
```

شکل 6-3 : دقت مدل در انتهای آموزش مدل بر روی دادگان آموزش و اعتبار سنجی

مشاهده می کنیم پس از 80000 گام , مدل به دقت خوبی بر روی دادگان آموزش دست پیدا کرده است (تقریبا % 97) و به نوعی انگار که مدل دادگان آموزش را حفظ کرده است.

مشاهده می کنیم مطابق با انتظار به دقت خوبی بر روی دادگان اعتبار سنجی دست پیدا نکرده ایم. (حدود 30%).

در انتها نوبت به تست مدل ساخته شده بر روی دادگان آزمون می رسد :

Testing a model

```
[18] !onmt_translate -model Data/run/model_step_80000.pt -src Data/test.en -output Data/pred_1000.txt -gpu 0 -verbose
```

شکل 7-3 : تست مدل ایجاد شده بر روی دادگان آزمون

بخشی از خروجی مدل هنگام اعمال آن بر روی دادگان آزمون را بررسی می کنیم :

SENT 424: ['the', 'vessel', 'took', 'off', 'from', 'russia', "'s", 'baikonur', 'cosmodrome', 'in', 'kazakhstan', 'at', '4', ':', '11', 'p.m.', '(', '6', ':', '11', 'a.m.', 'et', ')', 'and', 'reached', 'its', 'preliminary', 'orbit', 'in', 'nine', 'minutes', ',', 'the', 'u.s.', 'space', 'agency', 'nasa', 'said', '.']
PRED 424: داون آمریکایی اعلام کرد که این قایق از وسعت روسیه در 4 ام قزاقستان در 4 ساعت افزایش پیدا کرده‌است ; در ساعت 4 : 11 , پیروزی در 4 دقیقه به عنوان یک بالگرد در رده 4 : 30 دقیقه گذشته و به مقام رسیدند .
PRED SCORE: -20.2843

SENT 425: ['the', 'launch', 'follows', 'a', 'failed', 'launch', 'in', 'august', ',', 'when', 'another', 'progress', 'crashed', 'into', 'the', 'siberian', 'wilderness', '.']
PRED 425: این شروع به صدور یک بسته واحد در ماه اوت منجر به فوران کرد .
PRED SCORE: -8.1822

SENT 426: ['it', 'was', 'the', 'first', 'such', 'crash', 'in', 'the', 'craft', "'s", '30', '-', 'year', 'service', 'history', ',', 'but', 'space', 'officials', 'said', 'the', 'station', "'s", 'three', '-', 'member', 'crew', 'was', 'never', 'in', 'any', 'danger', 'of', 'running', 'out', 'of', 'food', 'or', 'water', '.']
PRED 426: در تاریخ اول دوره زیر سیستم‌ها , در یک تاریخ آزمایش فرعی ساله , اظهار داشت که گفته می‌شود مقامات این کاوشگر سه تن از اعضای دریاچه یا آب که هیچ خطری از آن خارج نشده است , هیچ خطری از این آب ندارد .
PRED SCORE: -22.5176

SENT 427: ['in', 'a', 'written', 'statement', ',', 'nasa', "'s", 'manned', 'space', 'flight', 'chief', ',', 'bill', 'gerstenmaier', ',', 'said', 'the', 'launch', 'sets', 'the', 'stage', 'for', 'the', 'next', 'scheduled', 'crew', 'rotation', 'aboard', 'the', 'iss', 'in', 'november', '.']
PRED 427: در یک بیانیه نوشته شده‌است که مرکز برنامه‌ریزی‌شده ای باید پرواز در میان پرواز بعدی , کاپیتان بیل , گفته‌است که این مرحله در یک بیانیه کتبی قرار می‌گیرد .
PRED SCORE: -15.2308

در اخر نیز میانگین امتیاز پیش بینی[1] نهایی را هنگام اعمال مدل بر روی دادگان تست مشاهده می کنیم:

PRED AVG SCORE: -0.4663, PRED PPL: 1.5941

همان گونه که مشاهده کردیم مدل تنوانست به دقت مطلوبی بر روی دادگان آزمون و اعتبار سنجی دست پیدا کند , علت این امر می تواند رخ دادن فرابرازش[2] بر روی دادگان آموزش ناشی از محدود بودن منابع (دادگان آموزش) باشد. همچنین ممکن است با تغییر پارامتر های اولیه ی مدل به نتایج بهتری دست پیدا کنیم.

FairSeq

در ادامه با استفاده از ابزار دیگری به نام FairSeq پیاده سازی خود را انجام می دهیم.

جهت جلوگیری از طولانی شدن گزارش , از آوردن کدهای این بخش خودداری می کنیم و فقط نتایج را بررسی می کنیم. (تمامی کد ها به همراه توضیحات لازم در فایل ipynb. موجود می باشد)

ابتدا پیکربندی های لازم به همراه برخی پارامتر های مدل را همراه با تفسیر[3] لازم برای هر کدام مشاهده می کنیم :

```
# cpu threads when fetching & processing data.
num_workers=2,
# batch size in terms of tokens. gradient accumulation increases the effective batchsize.
max_tokens=8192,
accum_steps=2,
# the lr s calculated from Noam lr scheduler. you can tune the maximum lr by this factor.
lr_factor=2.,
lr_warmup=4000,
# clipping gradient norm helps alleviate gradient exploding
clip_norm=1.0,
# maximum epochs for training
max_epoch=30,
start_epoch=1,
# beam size for beam search
beam=5,
# generate sequences of maximum length ax + b, where x is the source length
max_len_a=1.2,
max_len_b=10,
# when decoding, post process sentence by removing sentencepiece symbols and jieba tokenization.
```

[1] Pre Avg Score
[2] Overfitting
[3] Comment

```
post_process = "sentencepiece",
# checkpoints
keep_last_epochs=5,
resume=None, # if resume from checkpoint name (under config.savedir)
# logging
use_wandb=False,
```

با توجه به مشترک بودن برخی پارامتر های موجود در این ابزار با ابزار قبلی و همچنین توضیحات کافی که قبل از هر پارامتر مشخص شده , از توضیح مجدد پارامتر ها خودداری می کنیم.

در ادامه سایر پیکربندی های لازم را که پیش تر نیز توضیح داده شد مشاهده می کنیم :

Architecture Related Configuration

```
arch_args = Namespace(
    encoder_embed_dim=256,
    encoder_ffn_embed_dim=512,
    encoder_layers=1,
    decoder_embed_dim=256,
    decoder_ffn_embed_dim=1024,
    decoder_layers=1,
    share_decoder_input_output_embed=True,
    dropout=0.3,
)
```

شکل 8-3 : سایر پیکر بندی های لازم در ابزار FairSeq

در ادامه در طی 30 دوره[1] میزان خطا[2] را به همراه معیار BLUE مشاهده می کنیم :

```
2022-06-10 01:57:44 | INFO | hw5.seq2seq | training loss: 6.2331
2022-06-10 01:57:44 | INFO | hw5.seq2seq | begin validation
2022-06-10 01:57:47 | INFO | hw5.seq2seq | example source: but it 's going to modernize weapons
2022-06-10 01:57:47 | INFO | hw5.seq2seq | example hypothesis: می خواهمند ." :می گوید. "
2022-06-10 01:57:47 | INFO | hw5.seq2seq | example reference: اینمی رودکهشلیحانتظامیراکیهابهاینتیازنداریمرامدرنکند.
2022-06-10 01:57:47 | INFO | hw5.seq2seq | validation loss:     6.2497
2022-06-10 01:57:47 | INFO | hw5.seq2seq | BLEU = 0.10 6.8/0.3/0.0/0.0 (BP = 1.000 ratio = 3.808 hyp_len = 5179 ref_len = 1360)
2022-06-10 01:57:47 | INFO | hw5.seq2seq | saved epoch checkpoint: /content/checkpoints/rnn/checkpoint28.pt
2022-06-10 01:57:47 | INFO | hw5.seq2seq | end of epoch 28
2022-06-10 01:57:57 | INFO | hw5.seq2seq | training loss: 6.1806
2022-06-10 01:57:57 | INFO | hw5.seq2seq | begin validation
2022-06-10 01:57:59 | INFO | hw5.seq2seq | example source: we shall manage somehow , she said .
2022-06-10 01:57:59 | INFO | hw5.seq2seq | example hypothesis: می دونند " :می گوید. "
2022-06-10 01:57:59 | INFO | hw5.seq2seq | example reference: بالاخرهیککاریتمی کنیم , گفت:غصهنخور. "
2022-06-10 01:57:59 | INFO | hw5.seq2seq | validation loss:     6.2272
2022-06-10 01:57:59 | INFO | hw5.seq2seq | BLEU = 0.10 6.5/0.2/0.0/0.0 (BP = 1.000 ratio = 3.868 hyp_len = 5261 ref_len = 1360)
2022-06-10 01:58:00 | INFO | hw5.seq2seq | saved epoch checkpoint: /content/checkpoints/rnn/checkpoint29.pt
2022-06-10 01:58:00 | INFO | hw5.seq2seq | end of epoch 29
2022-06-10 01:58:10 | INFO | hw5.seq2seq | training loss: 6.1511
2022-06-10 01:58:10 | INFO | hw5.seq2seq | begin validation
2022-06-10 01:58:13 | INFO | hw5.seq2seq | example source: he wore old fashioned spectacles that
2022-06-10 01:58:13 | INFO | hw5.seq2seq | example hypothesis: می دونند " :می گوید , 
2022-06-10 01:58:13 | INFO | hw5.seq2seq | example reference: عینکیازمدافتادهبهچشم داشت
2022-06-10 01:58:13 | INFO | hw5.seq2seq | validation loss:     6.1989
2022-06-10 01:58:13 | INFO | hw5.seq2seq | BLEU = 0.09 5.8/0.2/0.0/0.0 (BP = 1.000 ratio = 3.953 hyp_len = 5376 ref_len = 1360)
2022-06-10 01:58:13 | INFO | hw5.seq2seq | saved epoch checkpoint: /content/checkpoints/rnn/checkpoint30.pt
2022-06-10 01:58:13 | INFO | hw5.seq2seq | end of epoch 30
```

شکل 9-3 : نمایش 3 ایپاک آخر به همراه معیار BLUE به انضمام یک مثال در هر ایپاک

[1] Epoch
[2] Loss

در نهایت به بررسی 3 معیار که برای ارزیابی یک ابزار تولید ماشین ترجمه مناسب هستند می پردازیم و بر این اساس به مقایسه ی دو ابزار انتخاب شده (OpenNMT و FairSeq) می پردازیم.

پس از بررسی کامل دو ابزار و آموزش مدل بر روی هر کدام , به 3 معیار مهم برای ارزیابی یک ابزار تولید ماشین ترجمه دست پیدا کردیم.

اولین معیار , سادگی کار با ابزار مورد نظر است. یکی از مهم ترین ویژگی ها برای یک ابزار مناسب برای هر تسکی (اعم از ترجمه ماشینی) , کاربرپسند[1] بودن ابزار مورد نظر است.

معیار دوم , تعداد پارامتر قابل تنظیم یک ابزار است. همانگونه که دیدیم تعداد پارامتر قابل تنظیم در ابزار OpenNMT به مراتب کمتر از ابزار FairSeq بود که هم می تواند در عین حال یک حسن محسوب شود هم یک عیب. حسن از این نظر که هر چقدر تعداد پارامتر ها زیاد شود تنظیم کردن آن ها سخت تر می شود (مثل FairSeq) و عیب از این نظر که هر چه تعداد پارامتر ها کمتر باشد قابلیت انعطاف مدل و بهره وری آن کمتر می شود.

در نهایت آخرین معیار مورد نظر , سرعت آموزش مدل می باشد. قطعا هر چه سرعت ترین شدن یک مدل بیشتر باشد مدل , مدل بهتر و قوی تری محسوب می شود که با توجه به مدت زمان ترین شدن در هر ابزار , دیدیم که سرعت FairSeq از OpenNMT بیشتر بوده و از این جهت از برتری نسبی برخوردار بوده است.

در آخر با توجه به معیار های ذکر شده و ارزیابی هر ابزار با استفاده از این معیار ها , اگر چه تنظیم پارامتر های مدل با استفاده از ابزار FairSeq کمی پیچیده تر بوده , اما انعطاف بالای مدل و مدت زمان ترین شدن مدل , به این نتیجه می رسیم که ابزار FairSeq انتخاب بهتری نسبت به OpenNMT بوده است.

[1] User friendly

Attachment

- تمامی خواسته ها و سوالات مطرح شده در صورت پروژه به طور کامل به تفکیک در هر بخش به صورت کامل پاسخ داده شده است. پیشاپیش از این که گزارش را با صبر و حوصله مطالعه می کنید , کمال تشکر را دارم.
- همچنین تمامی فایل های ipynb در پوشه ی Codes موجود می باشد. همچنین تمامی کد ها در محیط colab اجرا و تست شده اند.
- با توجه به شرایط و پیاده سازی های موجود در این پروژه و بعضا سنگین بودن حجم پردازشات , اجرای برخی سل ها زمان بر خواهد بود , لذا پیشاپیش از صبر و تحمل شما برای اجرا شدن کدها سپاسگزارم.
- همچنین پیش از اجرای کد ها , لازم است تنظیمات ذکر شده در ابتدای گزارش در بخش Basic settings اعمال شوند تا سل ها بدون خطا اجرا شوند.