به نام خدا

گزارش تمرین تبدیل فینگلیش به فارسی و بالعکس

سید محمد حسینی (۴۰۰۳۰۱۷۲۶)، صدرالدین باریک‌بین (۹۸۲۰۸۸۲۴)، امین بهجتی (۴۰۰۲۱۰۹۳۶)

مقدمه

در این تمرین هدف تبدیل نوشتار فارسی به متون فینگلیش و تبدیل نوشتار فینگلیش به فارسی است. برای جمع‌آوری داده‌ی آموزش از کراول کردن داده استفاده کردیم و از مدل ترجمه ماشینی مبتنی بر transformer در سطح حروف برای تبدیل نوشتار استفاده کرده‌ایم.

آماده‌سازی و پیش‌پردازش داده

داده‌های آموزشی ارائه شده کم بودند. برای همین ابتدا با استفاده از یک کتاب‌خانه nodeJS برای تبدیل نوشته فارسی به فینگلیش، تلاش کردیم با کراول کردن داده‌های ویکی‌پدیا، داده‌های یادگیری بسازیم. اما این کتاب‌خانه به خوبی تبدیل انجام نمی‌داد. برای همین به سراغ کراول کردن سایت واژه‌یاب با استفاده از scrapy رفتیم. در سایت واژه یاب همانطور که در زیر می‌بینید کنار هر کلمه نحوه تلفظ آن نیز نوشته شده است.


سلام /salām/

معنی

۱. واژه‌ای که در آغاز گفتگو به کار می‌رود.
۲. (فقه) سه جمله‌ای که با سلام آغاز می‌شود و از ارکان نماز است و نمازگزار در آخرین رکعت نماز بیان می‌کند.
۳. تحیت و درود.
۴. [قدیمی] پاکی رهایی از عیب و آفت.
۵. (نظامی) احترام مرد نظامی به حالت خبردار و بالا بردن دست راست و قرار دادن نوک انگشت وسط رو به شقیقه، به شکلی که کف دست راست رو به جلو است.
⟨ سلام کردن: (مصدر لازم) درود گفتن به کسی؛ سلام‌علیک یا سلام‌علیکم گفتن؛ سلام زدن.
⟨ سلام گفتن: (مصدر لازم) = ⟨ سلام کردن

فرهنگ فارسی عمید

پس از کراول کردن با یک پیش‌پردازش داده را تمیز کردیم و عبارت مربوط به تلفظ با استفاده از کتاب‌خانه re به عبارت فینگلیش تبدیل کردیم. مجموعا حدود ۴۰هزار کلمه متمایز کراول شد و با استفاده از این‌ها مدل را آموزش دادیم و خروجی بهتری نسبت به داده‌های آموزش قبلی گرفتیم. کدهای این بخش در فولدر crawled code آمده است.

شرح کلی مدل ترجمه مبتنی بر transformer

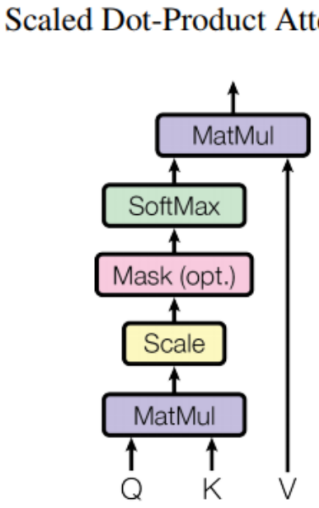
برای پیاده‌سازی این بخش که بخش اصلی کد است، از کد آماده‌ی tensorflow برای ترجمه ماشینی مبتنی بر transformer استفاده کردیم و متناسب با نیاز خود آن را تغییر دادیم و مدل یادگیری را ایجاد کردیم.

مراحل کلی این فرآیند به شرح زیر است:

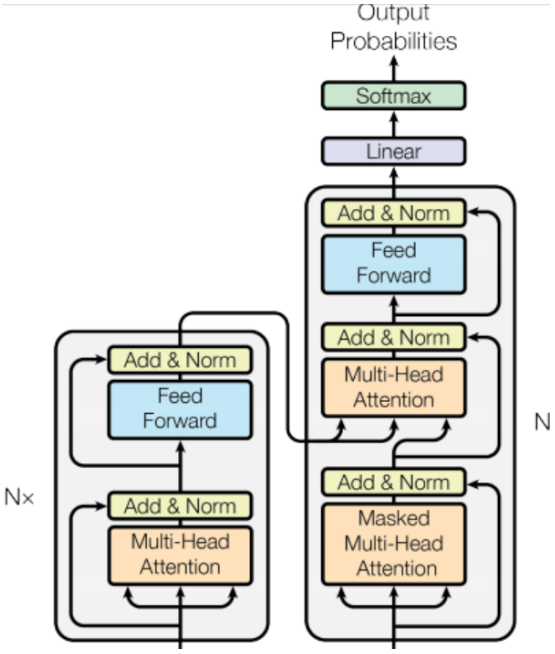
- ایجاد دیتاست برای داده‌ها: در این بخش داده‌ها را لود کردیم و با استفاده از tf.data.Dataset.from_tensor_slices، دیتاست‌های لازم برای آموزش و ارزیابی را ساختیم.
- تبدیل متن به دنباله‌ای از نویسه‌ها و ایجاد خط داده: با استفاده از tf.keras.layers.StringLookup و دادن لیست حروف به آن فرآیند تبدیل حرف به عدد را ایجاد کردیم. سپس داده‌ها را به batchهای به طول ۶۴ تقسیم کردیم تا batchهای آموزش و ارزیابی را بسازیم
- کدگذاری مکانی (Positional encoding): تابع پیاده‌سازی برای رسیدن به این embedding را پیاده می‌کنیم:

$$PE_{(pos,2i)} = \sin(pos/10000^{2i/d_{model}})$$
$$PE_{(pos,2i+1)} = \cos(pos/10000^{2i/d_{model}})$$

- ایجاد ماسک: حال تابع ایجاد padding mask و look ahead mask را داریم که موقع استفاده از attention کلیدها را محدود کنیم.
- ضرب داخلی scale شده برای attention: حال تابع پیاده‌سازی برای محاسبه‌ی وزن‌ها و خروجی attention را برحسب مدل ترنسفورمر پیاده می‌کنیم:



- تعریف کلاس برای MultiHeadAttention: حال تابعی برای چندسره کردن attention داریم که d_model را به تعدادی بخش افراز می‌کند و سپس پاسخ‌ها را با هم ترکیب می‌کند و خروجی attention را با استفاده از تابع بخش قبل محاسبه می‌کند.
- تابع feed forward برای شبکه: در این بخش از RelU برای activation استفاده می‌کنیم.
- ساخت لایه و کلاس Encoder و Decoder: در این بخش ابتدا با توجه به ساختار ترنسفورمر کلاسی برای لایه دیکودر و کلاسی برای لایه انکودر مطابق شکل زیر ایجاد می‌کنیم. همه بخش‌های آن در بخش‌های قبلی پیاده شده‌اند. سپس با اضافه کردن ورودی‌ها و اعمال PE و n لایه از انکودر و دیکودر در کنار هم، کلاس انکودر و دیکودر را می‌سازیم.



- ساخت Transformer: حال با ترکیب این ۲ کلاس، کلاس ترسنفورمر کامل می‌شود و تابع ایجاد ماسک و فراخوانی را برایش تعریف می‌کنیم.
- تنظیم Hyperparameterها: این مقادیر را به صورت زیر هستند:

```
num_layers = 2
d_model = 32
dff = 64
num_heads = 8
dropout_rate = 0.1
```

- نوشتن بهینه‌ساز: برای نرخ یادگیری از تابعی که در مقاله به آن اشاره شده استفاده می‌کنیم:

$$lrate = d_{model}^{-0.5} * \min(step\_num^{-0.5}, step\_num \cdot warmup\_steps^{-1.5})$$

- تعریف تابع loss و معیارهای ارزیابی: در این بخش تابع loss و ملاک‌های ارزیابی accurecy و bleu قرار دارد.
- چک‌پوینت‌گذاری و آموزش داده: در آخر نیز یک instance از ترنسفورمر می‌سازیم و از چک‌پوینت برای لود کردن مدلی که قبلا آموزش داده شده باشد استفاده می‌کنیم. سپس با قرار دادن EPOCH=6، مدل را آموزش می‌دهیم. در تبدیل فینگلیش به فارسی به این دقت در یادگیری می‌رسیم:

Loss 0.2486 Accuracy 0.9187

در فارسی به فینگلیش به این دقت می‌رسیم:

Loss 0.3283 Accuracy 0.8896

ارزیابی داده‌ها

از آنجا که داده تست داده‌ای برای ارزیابی نبود (زیرا یا فقط عبارت فارسی را داشت، یا فینگلیش) و ما از داده train داده شده استفاده نکردیم، آن را مبنای ارزیابی قرار دادیم. اما از آنجا که مدل ما برای تبدیل یک کلمه ساخته شده است باید کلمات این فایل را از جملات استخراج می‌کردیم. برای این‌کار اگر تعداد کلمات فارسی و فینگلیش برابر می‌بودند، پس از تمیز کردن داده، آن‌ها را به لیست داده‌های ارزیابی اضافه می‌کردیم. کد این پردازش در فولدر dataset وجود دارد و فایل train.csv را مورد پردازش قرار دادیم و نتیجه را در eval.csv ریختیم.

برای ارزیابی نیز از ملاک‌های bleu و accuracy استفاده کردیم. برای bleu از کتابخانه ignite استفاده کردیم و برای accurecy نیز کد آن را پیاده کردیم. سپس داده‌های ارزیابی نیز بخش بخش به مدل دادیم و امتیاز آن‌ها را حساب کردیم. برای فینگلیش به فارسی امتیاز اینطور شد:

Validation Accuracy 0.7885

Validation BLEU 0.5842

برای فارسی به فینگلیش نیز به این امتیازها رسیدیم:

Validation Accuracy 0.7291

Validation BLEU 0.4679

تبدیل جمله

در بخش پایانی کد، جمله‌های فایل train.csv را باید تبدیل می‌کردیم. یک کلاس Translator برای تبدیل کلمه ایجاد کردیم و از آن استفاده کردیم. این جملات را کلمه کلمه تبدیل کردیم و حاصل را در دو فایل test-pe و test-fi در فولدر dataset قرار دادیم.
چند نمونه از تبدیل کلمه‌ها را در زیر می‌بینید:

فارسی به فینگلیش:

اما ما خیلی بدجنس بودیم	amaa maa khili badjans budim
میشه کمکش کنیم بره خونه اش؟	mishe kamkesh kanim barre khune ash
این بهترین داستانیه که تابه حال شنیدم	in bahtarin daastaaniye kah taabe haal shanidam

فینگلیش به فارسی:

دیگه بع من حرف نمیزند	dige ba man harf nemizanad.
باید حواسش رع پرت کنیم	baayad havaasash ro part konim.
یک پرنده ندیدن	yek parande nadidan?
شرمنده حمه وگتشهن پره	sharmande hame vaghteshoon pore.