به نام خدا

دانشگاه تهران

دانشکده مهندسی برق و کامپیوتر

درس شبکه‌های عصبی و یادگیری عمیق

تمرین پنجم

نام و نام خانوادگی	حسام اسداله‌زاده – مسعود طهماسبی
شماره دانشجویی	810198346 – 810198429
تاریخ ارسال گزارش	۱۴۰۱.۱۰.۱۶

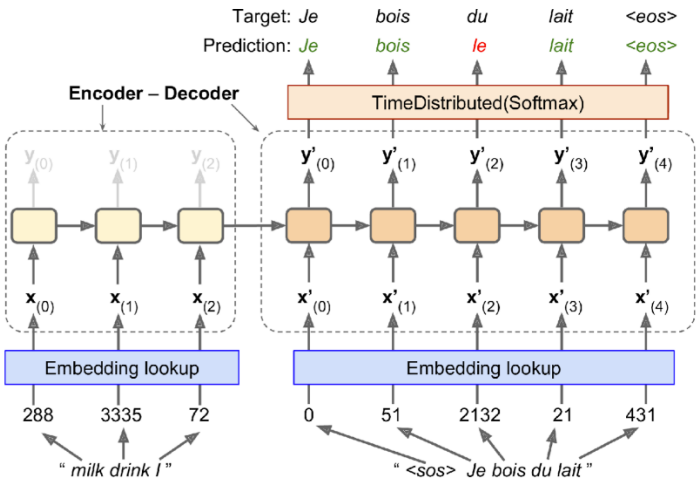
فهرست

شکل‌ها

پاسخ ۱ – آشنایی با مفهوم توجه[1] و پیاده سازی مدل BERT

۱-۱- پیاده‌سازی کدگذار[2]

1. **توضیح مختصری در مورد مفهوم توجه**: مکانیزم توجه یکی از مهم‌ترین مفاهیم و معماری‌هایی است که انقلابی در حوزه‌ی هوش مصنوعی و پردازش زبان طبیعی (NLP) محسوب می‌شود و بعدها این مکانیزم باعث شروع استفاده‌ی وسیع از ترنسفورمرها شد. همانطور که در درس دیدیم شبکه‌های بازگشتی RNN در کاربردهایی که sequence طولانی‌تر از معمول است، دقت خوبی ندارند. دلیل این موضوع نیز گرادیان‌های ناپایدار است که به دو شکل گردایان exploding یا vanishing ظاهر می‌شود. برای حل این مشکل انواع و اقسام long-term memories ارائه شد، یعنی حافظه‌هایی که به صورت طولانی‌مدت اطلاعات را نگه دارند که معروف‌ترین آنها نیز شبکه‌ی LSTM بود. از طرفی عمل ترجمه نیاز دارد که طول ورودی‌ها و خروجی‌ها متغیر و اختیاری باشند، برای برطرف کردن مشکل فراموشی تدریجی شبکه‌های فوق، مدل encoder-decoder ارائه شد:



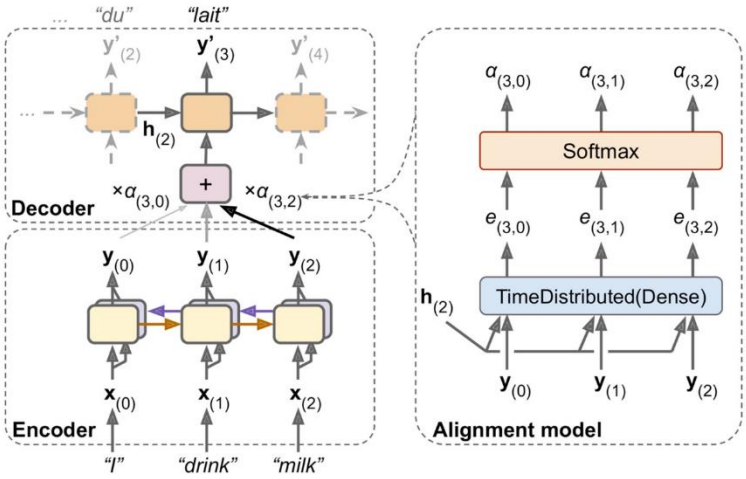
شکل 1. مدل **Encoder-Decoder**

برای آموزش این شبکه، جمله‌ی زبان مبدا به encoder داده می‌شود و جمله‌ی مقصد به decoder داده می‌شود (البته با یک شیفت زمانی). یعنی در هر مرحله کلمه‌ای را به decoder می‌دهیم که باید در مرحله قبل خروجی می‌داد و به عنوان اولین ورودی (قبل از اولین کلمه جمله) یک توکن به اسم SOS (توکنی

[1] Attention
[2] Encoder

برای start-of-sequence) به شبکه می‌دهیم. یک ایده هم این است که جمله مبدا را هم برعکس کنیم که اولین کلمه‌ی جمله، آخرین ورودی به encoder باشد. این نکته باعث می‌شود که کلمات اول جمله‌ی اول فاصله کم‌تری با کلمات اول جمله‌ی دوم داشته باشند. حالا این «فاصله‌ی کم‌تر» می‌تواند باعث تولید شدن بهتر کلمات اول جمله مقصد شود و احتمال اینکه در ادامه هم جمله مقصد بهتری تولید شود نیز افزایش می‌یابد. ایده‌ی بهتر برای حل این مسئله استفاده از مکانیزم **توجه** است.

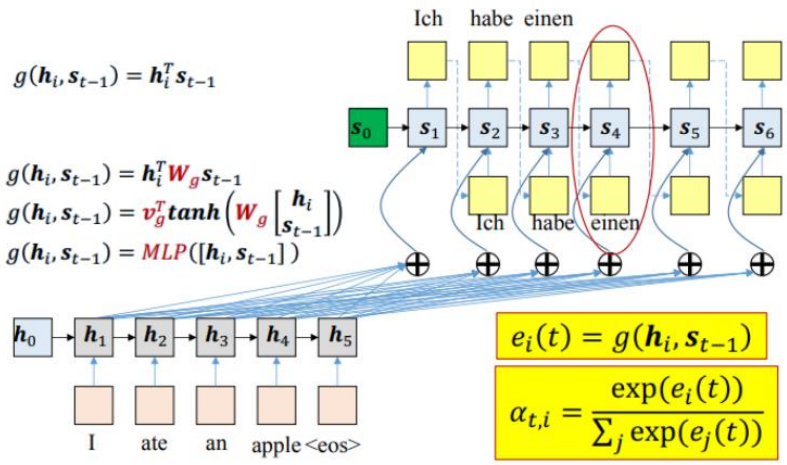
مکانیزم توجه تکنیکی است که ساختار کدگشا بتواند در هر قدم روی کلمات مناسب تمرکز و «توجه» کند. در این معماری علاوه بر state آخرین حالت نهان (final hidden state)، تمام خروجی‌های کدگذار را هم به کدگشا می‌فرستیم تا در هر گام، کدگشا یک جمع وزن‌دار بین این خروجی‌ها حساب کند و همین وزن‌ها هستند که مفهوم توجه را پیاده‌سازی می‌کنند؛ یعنی استیتی که وزن بیشتری دارد، سهم بیشتری در این بردار حاصل از جمع وزن‌دار دارد، پس در واقع توجه بیشتری به آن می‌شود.



شکل 2. مکانیزم توجه

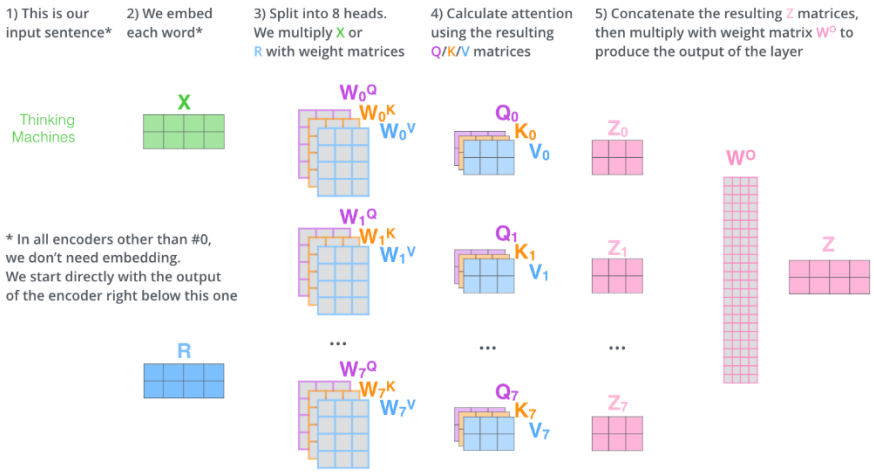
وزن‌های مذکور همان α ها در تصویر بالا هستند که همراه شبکه آموزش می‌بینند.

به طور خلاصه، ابتدا یک معماری دنباله به دنباله ساده پیشنهاد شد. این معماری ایده اولیه خوبی بود ولی مشکلاتی داشت. برای مثال یکی از عیب‌هایش این بود که اطلاعات کلماتی که اول جمله مبدا بودند به خوبی به قسمت کدگشا منتقل نمی‌شد و هنگام آموزش هم گرادیان به خوبی به عقب منتقل نمی‌شد. در واقع مشکل این بود که کل اطلاعات جمله مبدا باید در بردار آخرین حالت نهان کدگذار ذخیره می‌شدند که یک بردار با طول ثابت بود. این‌جا بود که ایده مکانیزم توجه خلق و ابداع شد. این ایده به این شکل بود که در قسمت کدگشا برای محاسبه حالت‌های نهان هر گام، یک بردار حاصل از جمع وزن‌دار بردارهای حالت نهان قسمت کدگذار محاسبه می‌شد که این وزن‌ها خود تابعی از میزان مشابهت حالت نهان فعلی کدگشا با حالت های نهان کدگذار هستند.



شکل 3. مدل مبتنی بر **Attention**

2. چرا در تبدیل کننده از Multi-Head Attention به جای Single-Head Attention استفاده می‌شود؟ فرض کنید در لایه‌ی Multi-Head ما h تا head مستقل و موازی داشته باشیم. این بدین معنی است که به جای اینکه یک representation داشته باشیم می‌توانیم چندتا subspace مختلف داشته باشیم (یعنی برای هر کلمه‌ی X، چندین query مختلف $(W_0^Q X, \dots, W_{h-1}^Q X)$ ساخته می‌شود و این کلمه با استفاده از h تا head مختلف به زیرفضاهای مختلفی map می‌شود). این موضوع برای key و value نیز دقیقا به همین صورت است و هر head از ماتریس تبدیل متفاوتی برای Q, K, V برای هر کلمه استفاده می‌کند. مزیت این موضوع این است که می‌توان گفت هر head مسئول توجه به بخش متفاوتی از جمله خواهد بود و در واقع کل حالات و توجه‌های مختلف محاسبه می‌شوند که کمک بسیاری به بهبود عملکرد شبکه می‌کند. در واقع این موضوع، توانایی مدل برای تمرکز بر موقعیت‌های مختلف را افزایش می‌دهد و قابلیت استفاده از چندین «زیرفضای بازنمایی» به لایه توجه می‌دهد. خلاصه‌ای از نحوه کار این لایه در ادامه آمده است:



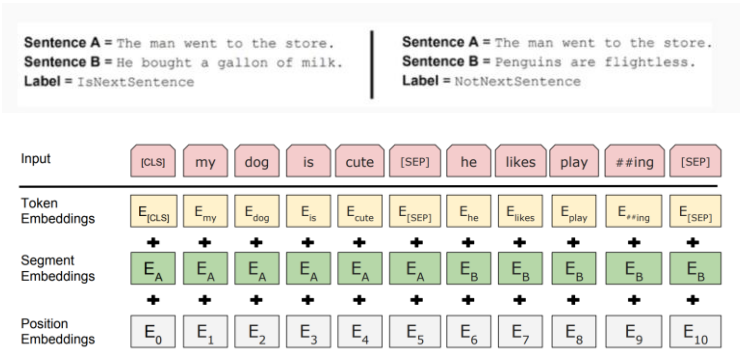
شکل 4. **Multi-Head Attention**

دلیل آورده شده در مقاله Attention Is All You Need به شرح زیر است:

Multi-head attention allows the model to jointly attend to information from different representation subspaces at different positions.

توضیح segmentation embedding

یکی از ابتکارهای BERT اضافه کردن توکن‌های خاص CLS و SEP است. در پیاده‌سازی برت، در هنگام ورودی گرفتن یک رشته، یک توکن CLS افزوده می‌شود که بردار بازنمایی که برت برای آن تولید می‌کند در واقع بازنمایی از کل دنباله است. برای مثال یک کاربرد این بازنمایی این است که با اضافه کردن یک لایه عصبی Softmax روی بازنمایی حاصل از CLS، در تسک تشخیص جمله بعدی، BERT قادر خواهد بود تا توزیع احتمال متوالی بودن جملات A و B را تخمین بزند (یعنی می‌تواند تشخیص دهد که دو جمله A و B واقعا متوالی هستند یا خیر). همچنین در پایان هر زیردنباله در دنباله ورودی نیز یک توکن SEP اضافه می‌شود که جای خاتمه جملات و مرز بین آن‌ها را نشان می‌دهد. علاوه بر این ابتکار، بردار segment embedding قابل یادگیری نیز به زیردنباله‌ها اعمال می‌شود، به بیان دیگر به ازای کلمات در زیردنباله‌های مختلف، بردارهای segmentation embedding متفاوتی بر آن‌ها اعمال می‌شوند. در نهایت BERT دارای سه مکانیزم تعبیه مکانی (positional)، قطعه‌ای (segmentation) و کلمه‌ای است که به ترتیب بر رشته ورودی اعمال می‌شوند:



گزارش نتیجه جمله‌ی "I liked this movie":

```
history = model.fit(
    x_train,
    y_train,
    batch_size=128,
    epochs=2,
    validation_data=(x_test, y_test)
)
```

```
Epoch 1/2
1970/1970 [==============================] - 832s 414ms/step - loss: 0.5228 - accuracy: 0.7315 - val_loss: 0.4691 - val_accuracy: 0.7775
Epoch 2/2
1970/1970 [==============================] - 807s 410ms/step - loss: 0.4244 - accuracy: 0.8000 - val_loss: 0.4587 - val_accuracy: 0.7873
```

```
model(np.array([encode_sentence("I liked this movie".lower(), MAXLEN)], dtype=np.int64))
```

```
<tf.Tensor: shape=(1, 1), dtype=float32, numpy=array([[0.95685554]], dtype=float32)>
```

در این تصویر مشخص است که مدل با احتمال ۹۶ درصد تشخیص داده که این جمله متعلق به کلاس مثبت (۱) می‌باشد.

پاسخ ۲ – آشنایی با کاربرد تبدیل کننده‌ها[1] در تصویر

۲-۱- آشنایی با مدل BEiT

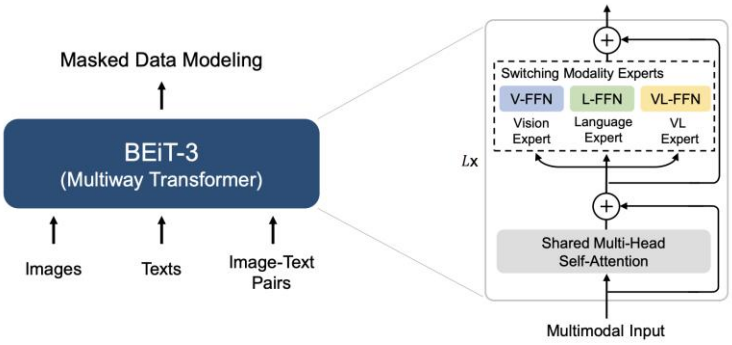
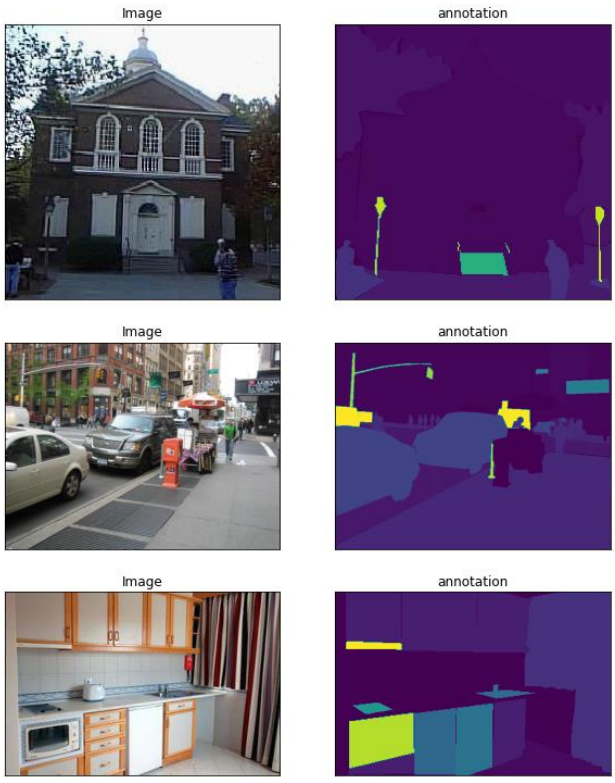


Figure 2: Overview of BEIT-3 pretraining. We perform masked data modeling on monomodal (i.e., images, and texts) and multimodal (i.e., image-text pairs) data with a shared Multiway Transformer as the backbone network.

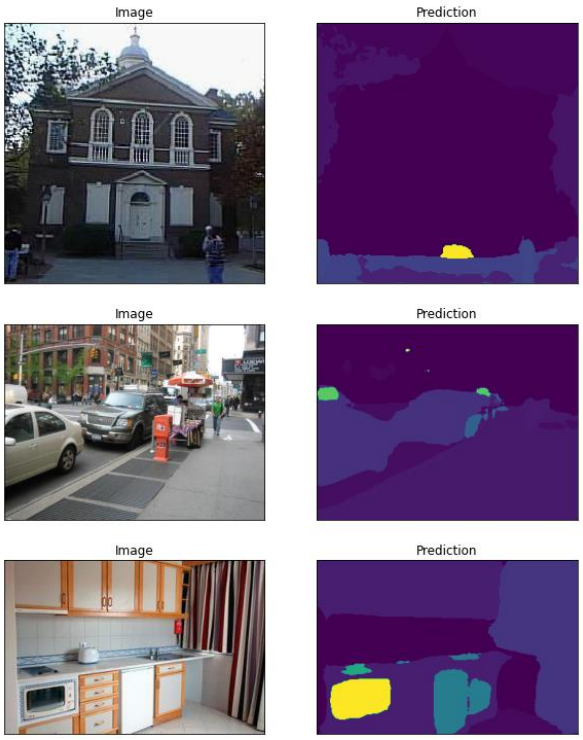
۲-۲- تقسیم‌بندی معنایی تصاویر

۳ نمونه تصویر (اصل تصویر و تقسیم‌بندی معنایی آن)



[1] Transformers

تقسیم‌بندی معنایی تصاویر فوق با استفاده از مدل بازآموزش یافته:



۲-۳- طبقه‌بندی تصاویر

نتایج MLP

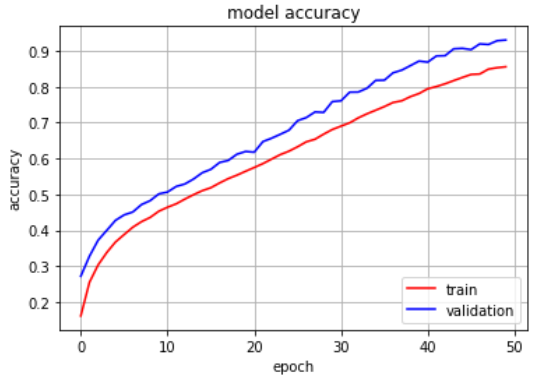
معماری شبکه (تعداد نورون‌های موجود در هر لایه‌ی FC):

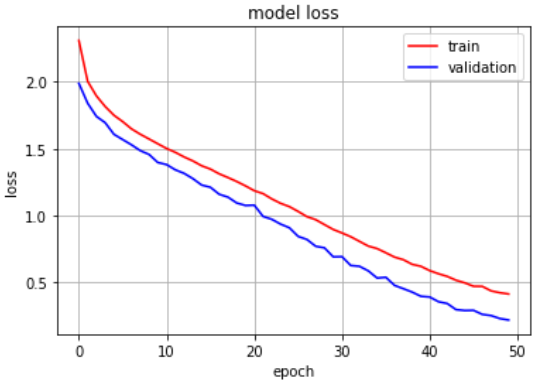
3072 – 4096 – 8192 – 4096 – 1024 – 256 – 10

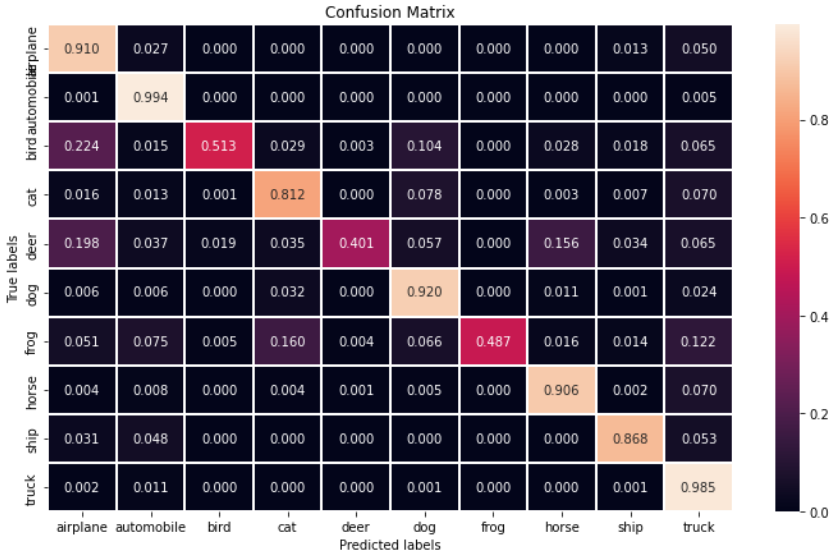
تعداد پارامترهای شبکه: 93,608,714

نتایج:

	precision	recall	f1-score	support
airplane	0.62	0.91	0.74	986
automobile	0.80	0.99	0.89	986
bird	0.96	0.51	0.67	1038
cat	0.76	0.81	0.78	989
deer	0.98	0.40	0.57	1007
dog	0.75	0.92	0.82	995
frog	1.00	0.49	0.66	987
horse	0.81	0.91	0.86	1012
ship	0.90	0.87	0.89	978
truck	0.66	0.99	0.79	1022
accuracy			0.78	10000
macro avg	0.82	0.78	0.77	10000
weighted avg	0.82	0.78	0.77	10000



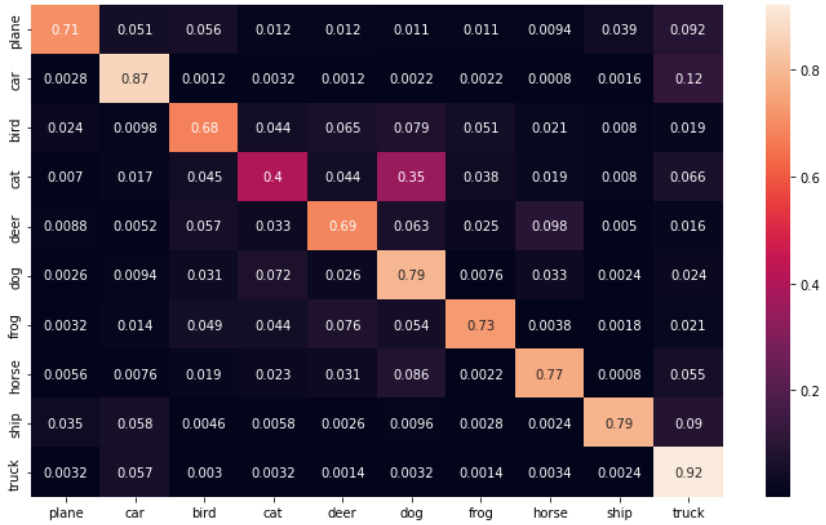




نتایج طبقه‌بند BeIT

از آنجایی که fine-tune (بازآموزش) این مدل بسیار زمان‌بر بوده (هر ایپاک تقریبا 30 دقیقه) و مدت زمان قابل استفاده از GPU در کولب محدود است، توانستیم به مدت 6 ایپاک این مدل را آموزش دهیم:

```
test loss : 0.742321598724303
test accuracy : 0.9503218531608582
test precision : 0.8155014514923096
```




```
Epoch: 0
        Loss: 1.7394749669719223
Epoch: 1
        Loss: 1.4057737993111004
Epoch: 2
        Loss: 1.2311617908993366
Epoch: 3
        Loss: 1.1053410907510146
Epoch: 4
        Loss: 0.9854935388380491
Epoch: 5
        Loss: 0.8862873429834118
Epoch: 6
```

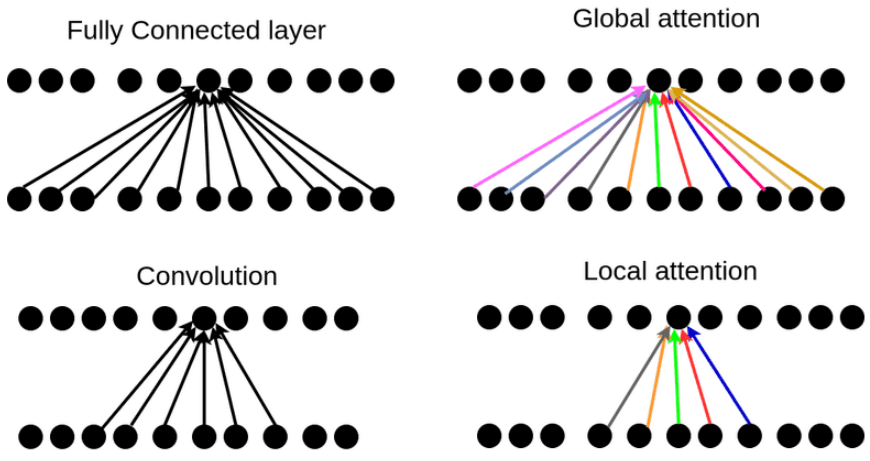
مدل BEiT ، در انتهای ایپاک ششم به Loss = 0.89 رسیده. این در حالی است که مدل MLP با تعداد پارامترهای تقریبا مشابه، به نتایج زیر رسیده:

```
Epoch 1/50
98/98 [==============================] - 11s 108ms/step - loss: 2.3086 - accuracy: 0.1614 - val_loss: 1.9883 - val_accuracy: 0.2720
Epoch 2/50
98/98 [==============================] - 10s 102ms/step - loss: 2.0038 - accuracy: 0.2551 - val_loss: 1.8401 - val_accuracy: 0.3278
Epoch 3/50
98/98 [==============================] - 10s 106ms/step - loss: 1.8946 - accuracy: 0.3036 - val_loss: 1.7414 - val_accuracy: 0.3725
Epoch 4/50
98/98 [==============================] - 10s 105ms/step - loss: 1.8151 - accuracy: 0.3382 - val_loss: 1.6942 - val_accuracy: 0.4000
Epoch 5/50
98/98 [==============================] - 10s 105ms/step - loss: 1.7497 - accuracy: 0.3674 - val_loss: 1.6073 - val_accuracy: 0.4277
```

این موضوع به وضوح قدرت شبکه‌ی BEiT را نسبت به شبکه‌های MLP نشان می‌دهد. البته بار محاسباتی شبکه‌ی BEiT قابل مقایسه با شبکه‌های MLP نیست و هم مرحله forward و هم backward آن بسیار زمان‌بر می‌باشد.

۲-۴- پرسش‌ها

1. در شبکه‌های CNN در کدام بخش مفهومی مانند مفهوم توجه اتفاق می‌افتد؟ همان‌گونه که اشاره شد، در مفهوم attention، به جای استفاده از Hidden State آخر، با وزن‌دهی مناسب، از تمام Hidden Stateها به صورت وزن‌دار استفاده می‌کنیم. در شبکه‌های کانولوشنی هنگام آموزش دیدن شبکه، وزن‌های مربوط به پنجره‌های کانوولوشن‌گیری (kernel ها) مقداردهی می‌شوند و ویژگی‌های مهم‌تر، تاثیر بیشتری در خروجی دارند. به نوعی توجه بیشتری به این ویژگی‌ها صورت می‌گیرد. این اتفاق مشابه مفهوم attention است. بنابراین عمل کانوولوشن‌گیری و وزن‌دار کردن پنجره‌ها، مشابه مفهوم annotation است. البته تفاوت کانولوشن با مفهوم توجه این است که هسته در کانولوشن ثابت بوده ولی در مکانیزم توجه به ازای هر بخش هر ورودی متفاوت می‌باشد.
2. در یک شبکه‌ی عصبی، در ارتباط یک لایه با لایه‌ی بعد، چه تفاوتی میان یک شبکه‌ی convolution با شبکه‌ی توجه همگانی و شبکه‌ی توجه محلی وجود دارد؟ تصویر زیر، به خوبی تفاوت میان شبکه‌های CNN، Fully Connected، توجه محلی و توجه همگانی را نشان می‌دهد. در شبکه‌های Fully Connected، تمام نورون‌ها به یکدیگر وصل هستند و ورودی هر نورون، از تمام نورون‌های لایه‌ی قبل، تاثیر می‌پذیرد. در حالیکه در شبکه‌های CNN، ورودی هر نورون تنها به **تعدادی** از نورون‌های لایه‌ی قبل وابسته است. بنابراین با توجه به مفهوم توجه محلی و توجه همگانی، به نوعی می‌توان گفت که شبکه‌های CNN، به نوعی از توجه محلی استفاده می‌کنند ولی شبکه‌های Fully Connected از توجه همگانی استفاده می‌کنند.

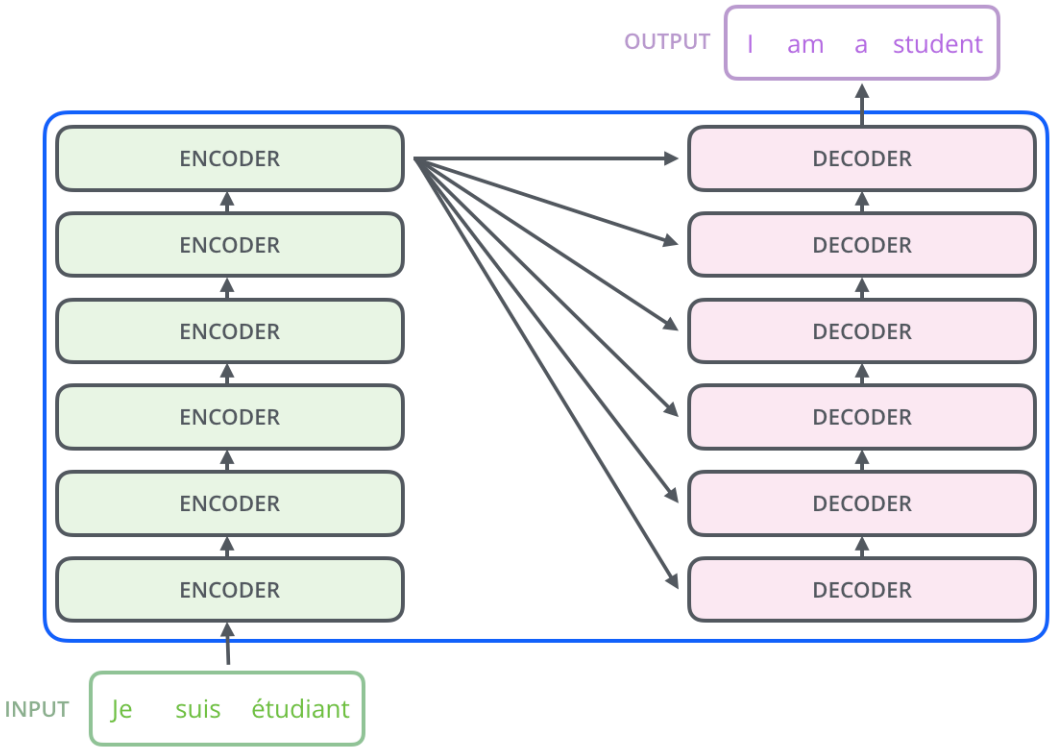


تفاوت دیگری که وجود دارد، بین شبکه‌های CNN و Fully Connected با شبکه‌های توجه محلی و همگانی است. در شبکه‌های CNN و Fully Connected اثری که هر نورون از یک نورونِ به خصوص که در لایه‌ی قبل قرار دارد، می‌پذیرد ثابت است؛ چراکه وزن‌های بین دو نورون پس از آموزش دیدن، ثابت

باقی می‌ماند. در حالیکه این تاثیرپذیری در شبکه‌های توجه محلی و همگانی می‌تواند متغیر باشد و لزوما ثابت نیست. به طور مثال اگر یک کرنل با بعد ۱ روی ورودی اعمال شود، مقدار این کرنل در شبکه‌ی کانولوشنی به ازای کل ورودی‌ها ثابت است ولی در مکانیزم توجه محلی با بعد ۱، چون توجه خروجی به هر کدام از ورودی‌ها متفاوت است، در نتیجه مقدار این کرنل نیز برای هر ورودی متفاوت خواهد بود.

درستی یا نادرستی جملات

- در بخشی از لایه‌های تبدیل کننده‌ی Vanilla از شبکه‌ی LSTM استفاده شده است: **نادرست**
  - همانگونه که در تصویر زیر مشخص است، در هیچ‌کدام از لایه‌های تبدیل‌کننده‌ی Vanilla، از شبکه‌ی LSTM استفاده نشده است. دلیل عدم وجود شبکه‌ی LSTM در لایه‌های تبدیل‌کننده‌ی Vanilla نیز واضح است، چراکه تبدیل‌کننده‌ها اصولا ساختار بازگشتی (Recurrent) ندارند در حالیکه شبکه‌ی LSTM زیرمجموعه‌ای از شبکه‌های بازگشتی است و وجود چنین شبکه‌ای داخل تبدیل‌کننده‌ها، با ذات این شبکه‌ها در تناقض است. شبکه‌های تبدیل‌کننده، قابلیت موازی‌سازی بسیار بیشتری نسبت به LSTM دارند چون به جای شبکه‌های بازگشتی از Self-attention استفاده می‌کنند.
- یک تبدیل کننده از چند بلوک رمزگذار و چند بلوک رمزگشا تشکیل شده است. **درست**
  - ساختار Vanilla Transformer به صورت زیر است:

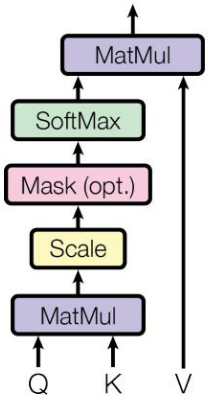


- Multi-head attention از یک بخش توجه و چند لایه‌ی تمام متصل موازی تشکیل شده است.

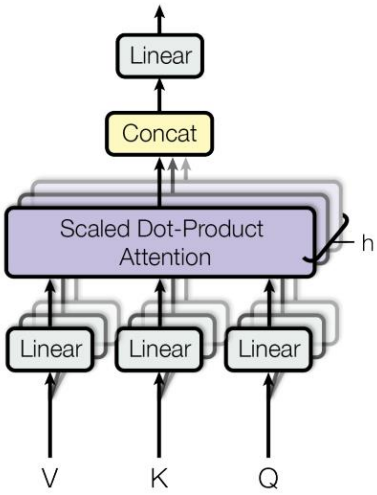
نادرست

  - همانگونه که در تصویر زیر مشخص است، در Multi-Head Attention از چندین ماژول Scaled Dot-Product Attention استفاده می‌شود نه یک بخش توجه. در شکل 4 نیز مشخص است، ابتدا چندین representation با head های مختلف به دست می‌آید (که داخل خود شامل attention و لایه‌ی تمام‌متصل می‌باشد) و در نهایت نیز با Concat کردن بازنمایی‌های مختلف و map کردن آن به سایز ورودی اصلی با استفاده از یک لایه‌ی تمام متصل، بازنمایی نهایی ساخته می‌شود.

Scaled Dot-Product Attention



Multi-Head Attention



- وجود Positional Encoding در ساختار یک تبدیل‌کننده حیاتی است و بدون آن شبکه از کار می‌افتد. **درست**

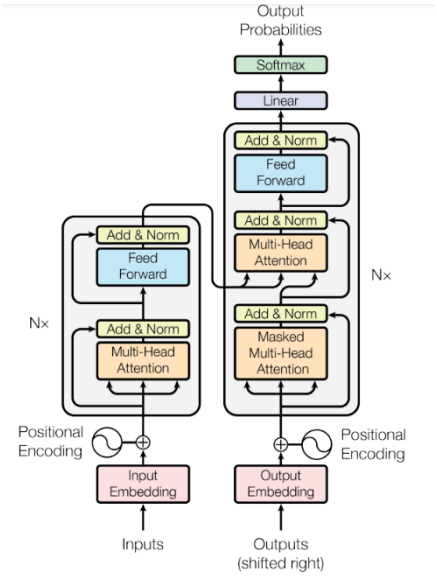


Figure 1: The Transformer - model architecture.

  - از آنجایی که تبدیل‌کننده‌ها حاوی هیچ کانولوشن و recurrence نیستند، برای اینکه مدل بتواند ترتیب کلمات دنباله را نیز لحاظ کند، باید اطلاعاتی در مورد موقعیت نسبی یا مطلق توکن‌ها در بازنمایی دنباله اضافه کنیم. برای این منظور، "رمزگذاری‌های موقعیتی" را به بازنمایی‌های ورودی در ابتدای رمزگذار و رمزگشا اضافه می کنیم.