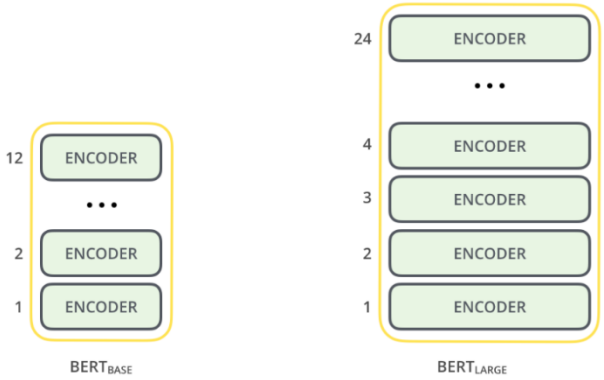
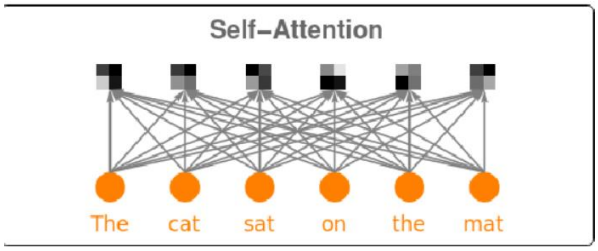
BERT 모델 encode, pretrain 구조 학습

동국대학교 컴퓨터공학 김관우 서혜민

Bert 모델은 Transformer Encoder 블록이 겹겹이 쌓인 형태로 구성되어 있다. base 모델은 인코더 블록이 12개이고, large 모델은 24개로 구성되어 있다. 이는 입력 시퀀스 전체의 의미를 N번 만큼 반복적으로 구축하는 것을 의미하고, 인코더의 블록 수가 많을수록 단어 사이에 보다 복잡한 관계를 더 잘 포착할 수 있다.

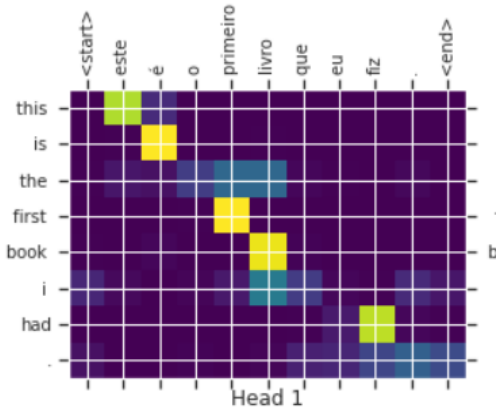


[Transformer Encoder]



위와 같은 인코더 블록은 task를 수행하기 위해 입력 문장의 모든 토큰이 사용되고 이 형태를 Self Attention 구조라고 한다. Self Attention 구조 내부에서는 내적 연산을 통해 Attention 벡터를 생성하게 되는데 Attention 벡터는 각각의 단어가 서로 어떤 연관성을 갖는지를 내포한다.

[attention 벡터의 예시 – 영어, 포트투갈 번역 모델]



위와 같은 Attention 맵에서 este를 보면 this라는 단어와 관련이 높은 것을 알 수 있는데 실제로 este는 This one이라는 뜻을 가지고 있다. 이처럼 각 단어의 연관성을 내포하는 Attention 벡터를 생성하는 역할을 transformer encoder 블록에서 수행한다.

인코더 블록의 가장 핵심적인 부분은 multi head attention이다. 이는 헤드가 여러 개인 어텐션을 의미하는데 서로 다른 가중치 행렬을 이용해 어텐션을 h번 계산한 다음 서로 연결한 결과를 가진다.

$$\begin{aligned}\text{MultiHead}(\mathbf{Q},\mathbf{K},\mathbf{V}) &= [\text{head}_1;\ldots;\text{head}_h]\mathbf{W}^O \\ \text{where head}_i &= \text{Attention}(\mathbf{Q}\mathbf{W}_i^Q,\mathbf{K}\mathbf{W}_i^K,\mathbf{V}\mathbf{W}_i^V)\end{aligned}$$ 4

BERT-base 모델의 경우 각각의 토큰 벡터 768차원을 헤드 수 만큼 12등분 하여 64개씩 12조각으로 차례대로 분리한다. 여기에 Scaled Dot-Product Attention을 적용하고 다시 786차원으로 합친다. 이렇게 수행하면 786차원 벡터는 각각 부위별로 12번 attention을 받은 결과가 된다.

➔ 텍스트의 다양한 면이 적용된 벡터가 나오게 된다.

[Scaled Dot-Product Attention]

** Self-Attention 알고리즘 => 한 단어와 나머지 다른 단어의 관계 정보를 처리할 수 있다.

1. 입력된 모든 토큰에 대해서 독립적으로 학습 가능한 Query, Key, Value 라는 벡터를 할당.
2. 한 토큰의 Query와 나머지 토큰의 Key 값을 모두 곱한다 => 이 값을 score라고 하며, 그 결과 각 토큰 마다 1개의 score가 나오게 된다.
3. 각 토큰의 2의 결과로 나온 score들을 8(dmodel/num_heads)로 나눈 뒤, softmax를 통해서 8로 나눠진 각 토큰의 score의 값의 합이 모두 1이 되도록 바꾼다.
   Softmax : 여러 값들이 존재할 때 각각의 값의 편차를 확대시켜 큰 값은 상대적으로 크

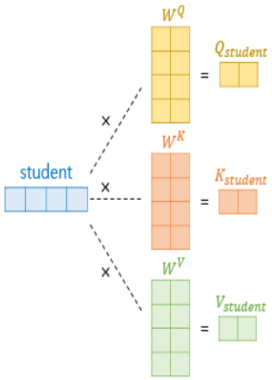
게, 작은 값은 작게 만든다음 정규화를 통해 이 값들의 합을 1로 만드는 함수.

4. 나온 값을 각각의 토큰의 Value값과 곱한 값이 attention이 된다.

위 과정을 토큰마다 반복하는 것이 1개의 head를 갖는 self-attention이 된다.

Muitihead Attention => Self-attention시, Query, Key, Value를 여러 개로 늘려서 계산.

** 단어들로부터 Q, K, V 벡터를 얻는 과정

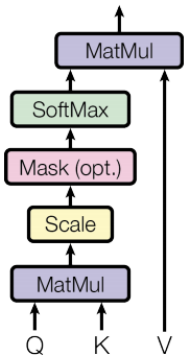


Q, K, V 벡터의 차원을 64(Dmodel/num_heads)의 값으로 결정한 뒤, 해당 벡터에 서로 다른 3개의 가중치 행렬(Wq, Wk, Wv)을 곱하여 64의 차원을 가지는 벡터를 얻어낸다.

모든 단어에 위 과정을 거치면 각각의 단어들의 Q,K,V벡터를 얻을 수 있다.

Attention(Q, K, V) – 어텐션 함수로 Q, K, V 세 개의 값을 입력 받는다. 기존의 Transformer에서 Q는 주로 디코더의 히든 스테이트, K는 주로 인코더의 히든 스테이트, V는 K에 에텐션을 부여 받은 Normalized Weights가 되며 초기 값은 V와 K가 동일하다. 하지만 BERT는 디코더를 사용하지 않기 때문에 Q, K, V의 초기 값이 모두 동일하다. BERT는 이처럼 동일한 토큰이 문장 내의 다른 토큰에 대한 self-Attention 효과를 가진다.

$$Attention(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$



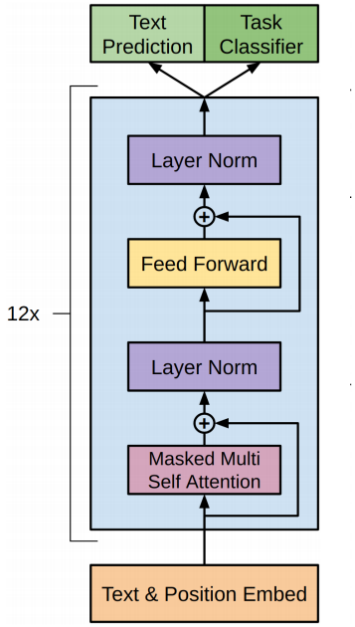
여기서 softmax는 활성화 함수 중 하나인데 주로 다중 분류에 사용되는 활성화 함수이다.

$$y_k = \frac{e^{a_k}}{\sum_{i=1}^{n} e^{a_i}}$$

e의 n승으로 계산하므로 변동 폭이 메우 크며, 작은 차이에도 쏠림이 두드러지게 나타난다. 결국 값이 큰 스칼라는 살아남고, 작은 쪽은 거의 0에 가까운 값을 곱하게 되어 배제되는 결과를 얻을 수 있다.

[전체 인코더 블록 구조]

위에서 구해진 멀티 헤더 어텐션의 결과는 position-wise feed forward network로 통과한다. feed forward는 입력 단어들에 대응하는 벡터 시퀀스를 벡터 각각을 feed-forward network에 들어가서 활성화 함수를 적용한다. 이를 통해 다음 인코더의 입력으로 들어가는 현재 인코더의 출력의 크기를 인코더에 들어왔을 때와 같게 하여 인코더간의 연결을 돕는다.



$$\mathrm{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2$$

FC2
ReLU
FC1

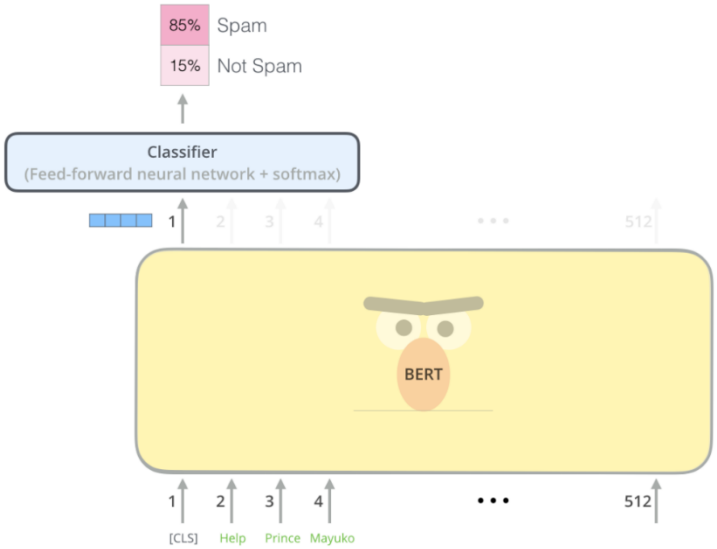
cf) add & norm : 각 sub layer마다 적용되어 잔차 연결(서브층의 입력과 출력을 더한 것 -> 정보의 손실과 모델의 학습 도움)과 층 정규화를 진행

BERT는 문장 표현을 학습하기 위해 pretraining으로 Masked Language Model과 Next Sentence Prediction으로 진행한다.

Masked Language Model: 문장 안의 랜덤한 곳을 mask로 가리고 그 안에 들어갈 단어를 예측하도록 한다. 추가적으로 다양한 표현을 학습하도록 80%는 [Mask]토큰으로 바꾸어 학습하고, 나머지 10%는 토큰을 random word로 바꾸고 마지막 10%는 원본 word를 그대로 사용한다.

Next Sentence Prediction 두 문장을 입력으로 주고 두번째 문장이 첫 문장 다음에 오는 문장인지 아닌지 모델이 예측하게 한다. 이는 두 문장의 관계를 학습할 수 있도록 한다. 보통 50%는 두 번째 문장이 첫 문장 다음에 오는 문장이고, 나머지 50%는 랜덤하게 뽑힌 두 문장으로 학습한다.

이렇게 학습된 BERT를 fine-tuning할 때 class label 개수만큼의 output을 가지는 Dense Layer를 붙여서 사용하게 된다.



[BERT의 sentence input]

BERT의 Input을 embedding하려면 특수문자와 대문자를 제거하고 [SEP] 토큰과 [CLS] 토큰을 삽입해 주어야 한다.

CLS: Special Classification token은 문장의 가장 첫 번째(문장의 시작) 토큰으로 삽입된다.

SEP: Special Separator token를 사용해 첫 번째 문장과 두 번째 문장을 구별한다

<Encode 실행 결과>

데이터 전체가 아닌 아래 2개의 데이터를 대상으로 했을 때의 결과이다.

```
'our deeds are the reason of this earthquake may allah forgive us all'
'forest fire near la ronge sask canada'
```

Token => Tokenizer를 통해서, 문장의 단어가 BERT의 단어 집합에 존재하지 않으면

이를 단어 집합에 있는 단어로 나눠서 쪼갠 뒤, 정수로 변환. 패딩은 0으로 지정

패딩의 크기(매개변수로 전달한 수 – 문장의 전체 토큰 길이) => 지정해준 최대길이보다 토큰의 길이가 짧을 경우 해당 토큰의 길이를 나타내기 위해 패딩이 쓰인다.

```
[101, 2256, 15616, 2024, 1996, 3114, 1997, 2023, 8372, 2089, 16455, 9641, 2149, 2035, 102,
[101, 3224, 2543, 2379, 2474, 6902, 3351, 21871, 2243, 2710, 102,
```

Mask_Pad => 나눠진 토큰들 전부를 1로, 토큰이 아닌 패딩을 0으로 지정하여 이 둘을 구분한다.

```
[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,
```

Segment_Id => 토큰 화된 텍스트의 토큰이 어떤 문장에 속하는지 0또는 1을 통해 지정을 하며, 단일 문장은 모두 0으로 지정을 한다.

```
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
```