

NLP 과제_Attention is all you need

220210031 유현우

1. Encoder

A. Multi-head attention layer

- i. Multi-head attention은 아래 그림에서와 같이 scaled dot-product attention이 병렬적으로 수행되는 구조
- ii. Scale dot-product attention은 다음의 수식으로 설명될 수 있음

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

iii.

1. Input feature에 대해서 linear projection을 이용해서 query, key, value를 생성

```
Q = self.fc_q(query)
K = self.fc_k(key)
V = self.fc_v(value)
```

A. 코드상

2. 이때 query, key, value의 shape은 [batch size, seq length, hidden dimension]의 형태를 갖고 있음
3. 수식상에 transposed key가 나타내는 shape은 [batch size, hidden dimension, seq length]의 형태가 됨
4. 따라서 query와 transposed key간의 행렬곱으로 출력된 energy는 query와 key의 word seq간의 similarity를 의미
5. 여기에 normalize term으로 $\sqrt{d_k}$ 으로 energy를 나누어 줌. 이때 d_k 는 hidden dimension을 의미
6. Query와 key간의 similarity로 뽑은 energy를 softmax를 취하여 attention score를 얻음

A. 코드상

```
energy = torch.matmul(Q, K.permute(0, 1, 3, 2)) / self.scale
attention = torch.softmax(energy, dim = -1)
```

7. Attention score와 value와의 행렬곱을 통해 attention을 value에 적용하여 출

력

A. 코드상 `x = torch.matmul(self.dropout(attention), V)`

- iv. Multi-head attention은 scaled dot-product attention이 병렬적으로 수행되는 구

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \dots, \text{head}_h)W^O$$

조로 다음의 수식으로 설명 $\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$

- v. 위의 scale dot-product attention을 병렬적으로 수행해주기 위해 query, key, value의 matrix shape을 [batch size, seq length, hidden dimension]에서 [batch size, seq length, number head, head dimension]으로 변형시켜줌

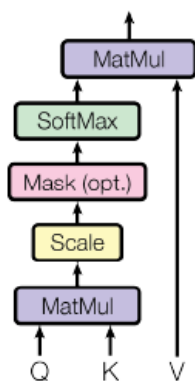
1. 이는 hidden dimension을 각 head 개수에 따라 head dimension으로 나누어 준 형태

- vi. 코드상

```
Q = Q.view(batch_size, -1, self.n_heads, self.head_dim).permute(0, 2, 1, 3)
K = K.view(batch_size, -1, self.n_heads, self.head_dim).permute(0, 2, 1, 3)
V = V.view(batch_size, -1, self.n_heads, self.head_dim).permute(0, 2, 1, 3)
```

- vii. 그리고 [batch size, number head, seq length, head dimension]으로 형태를 변환시켜 matrix상의 dim 3,4 ([seq length, head dimension])을 사용해 행렬곱을 수행하여 각 batch 및 number head에 대해 병렬적으로 attention 연산이 수행되도록 함
- viii. 최종적으로 한번 더 linear projection으로 embedding한 후 출력

Scaled Dot-Product Attention



Multi-Head Attention

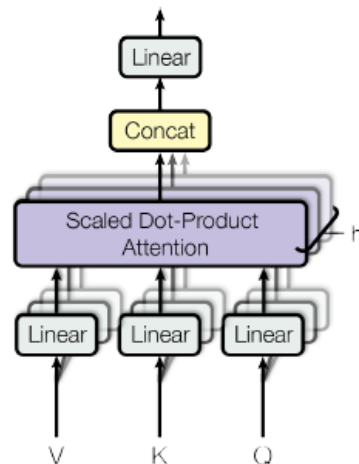


그림 1. scale dot-product attention 및 multi-head attention

B. Position wise feed forward layer

- i. Position wise feed forward layer는 network에서 자주 사용되는 layer임
- ii. 이는 단순히 각 seq를 linear projection으로 embedding 하고 relu activation을 취한 후 한번 더 linear projection을 이용해서 embedding 해주는 layer

1. 코드상

```
x = self.dropout(torch.relu(self.fc_1(x)))  
x = self.fc_2(x)
```

- iii. 즉, 그냥 각 seq 별로 두 번의 linear projection을 이용한 embedding을 수행해주는 것

C. Encoder layer

- i. Encoder layer는 위의 multi-head attention layer와 position wise feed forward layer로 이루어짐
- ii. Multi-head attention layer -> layer norm -> position wise feed forward -> layer norm으로 구성

```
_src, _ = self.self_attention(src, src, src, src_mask)  
src = self.self_attn_layer_norm(src + self.dropout(_src))  
_src = self.positionwise_feedforward(src)
```

- iii. 코드상 src = self.ff_layer_norm(src + self.dropout(_src))

D. Encoder

- i. Source word vector를 input으로 받음
- ii. Input된 word vector에 대해서 token embedding 및 position embedding을 수행해야 함
- iii. Token embedding은 linear projection 연산을 통해 이루어지고 position embedding은 각 위치에 해당하는 값을 더함

1. 코드상

```
pos = torch.arange(0, src_len).unsqueeze(0).repeat(batch_size, 1).to(self.device)
```

함수를 사용하는데 이는 src_len 길이의 1-d vectore를 생성해주는데 그 값은 index임 이것을 batch size만큼 이어붙여 해당 token과 더함

- iv. 코드상

- v.

```
src = self.dropout((self.tok_embedding(src) * self.scale) + self.pos_embedding(pos))
```
- vi. Token embedding과 position embedding을 수행한 token들을 encoder layer에 input 시켜 encoding 과정을 수행

2. Decoder

A. Decoder layer

- i. Decoder에서는 source word vector와 target word vector를 input으로 받음
- ii. Decoder layer는 앞서 다룬 encoder에서 사용한 multi-head attention layer 및 feed-forward layer를 마찬가지로 사용
- iii. 가장 먼저 embedding된 target token들을 input으로 받아 multi-head attention layer를 통과
 - 1. 이를 통해 token들의 similarity를 고려하여 embedding한 feature를 출력
 - 2. 코드상

```
_trg, _ = self.self_attention(trg, trg, trg, trg_mask)
```
 - 3. 그리고 layer norm 적용
- iv. 그리고 아래 그림의 transformer의 구조를 살펴보면 encoder에서 추출된 source token feature와 decoder에서 input된 target token feature를 multi-head attention의 input으로 받는 것을 알 수 있음
 - 1. 코드상

```
_trg, attention = self.encoder_attention(trg, enc_src, enc_src, src_mask)
```
 - 2. 이때 source token feature를 key, value로 embedding하고 target token feature를 query로 embedding 함
 - 3. 이를 통해 target token들이 source token들간의 관계를 고려하여 원하는 출력값을 도출할 수 있는 구조를 가짐
 - 4. 그리고 layer norm을 적용하여 출력
- v. 마지막으로 encoder block에서와 마찬가지로 feed-forward layer를 통과시킨 후 linear projection을 이용해 logit을 출력하고 이를 softmax를 적용하여 최종 prediction probabilities를 출력

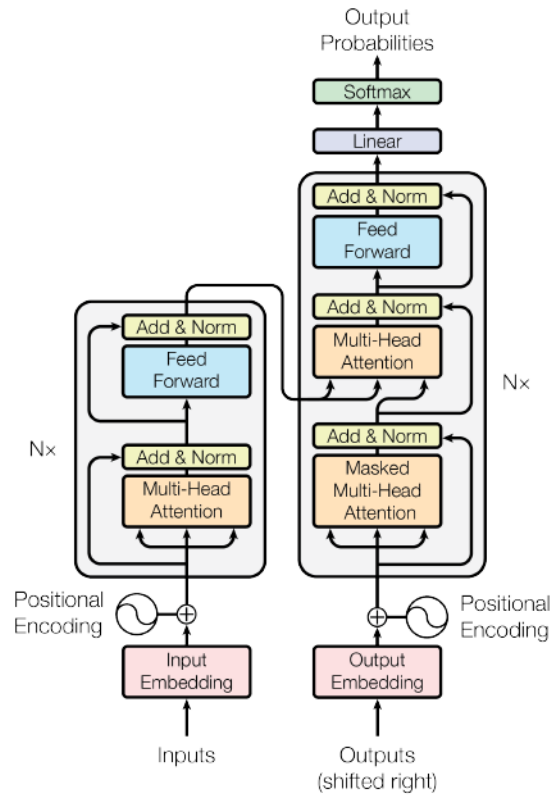


그림 2. Transformer architecture

3. Train

- A. 위에서 다룬 transformer encoder-decoder 구조를 이용해서 learning rate 0.0005, optimizer는 ADAM, loss는 cross entropy loss를 사용하여 10 epoch를 학습
- B. 학습 결과는 아래 그림과 같고 epoch에 따라 train loss 및 validation loss가 줄어드는 것을 확인할 수 있음

```
Epoch: 01 | Time: 0m 17s
      Train Loss: 4.238 | Train PPL: 69.251
      Val. Loss: 3.024 | Val. PPL: 20.566
Epoch: 02 | Time: 0m 16s
      Train Loss: 2.821 | Train PPL: 16.796
      Val. Loss: 2.312 | Val. PPL: 10.093
Epoch: 03 | Time: 0m 16s
      Train Loss: 2.235 | Train PPL: 9.347
      Val. Loss: 1.981 | Val. PPL: 7.253
Epoch: 04 | Time: 0m 17s
      Train Loss: 1.880 | Train PPL: 6.555
      Val. Loss: 1.802 | Val. PPL: 6.065
Epoch: 05 | Time: 0m 17s
      Train Loss: 1.635 | Train PPL: 5.129
      Val. Loss: 1.702 | Val. PPL: 5.485
Epoch: 06 | Time: 0m 17s
      Train Loss: 1.444 | Train PPL: 4.236
      Val. Loss: 1.649 | Val. PPL: 5.200
Epoch: 07 | Time: 0m 17s
      Train Loss: 1.293 | Train PPL: 3.643
      Val. Loss: 1.614 | Val. PPL: 5.024
Epoch: 08 | Time: 0m 17s
      Train Loss: 1.165 | Train PPL: 3.205
      Val. Loss: 1.618 | Val. PPL: 5.042
Epoch: 09 | Time: 0m 17s
      Train Loss: 1.055 | Train PPL: 2.872
      Val. Loss: 1.629 | Val. PPL: 5.099
Epoch: 10 | Time: 0m 17s
      Train Loss: 0.961 | Train PPL: 2.615
      Val. Loss: 1.642 | Val. PPL: 5.163
```

4. Inference

- A. Train 된 model weight를 불러와서 영어를 독일어로 번역 수행
- B. 실험 결과는 아래 그림에서와 같이