Attention is All You Need

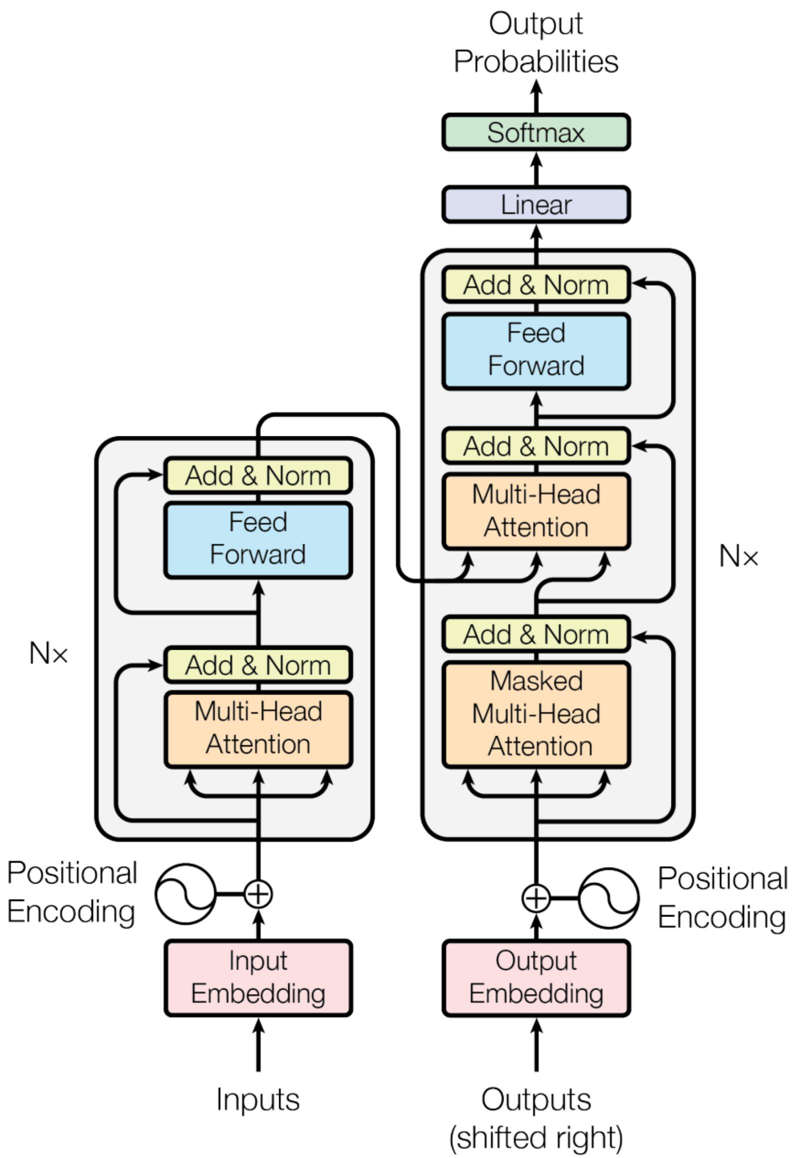
Project to implement the Transformer using Pytorch

단국대학교 모바일시스템공학과 양윤성

Contents

1-(1). Transformer

- No RNN and LSTM, No recurrence
- Still using encoder-decoder structure
- Only use attention method
- Repeat attention on multiple layers



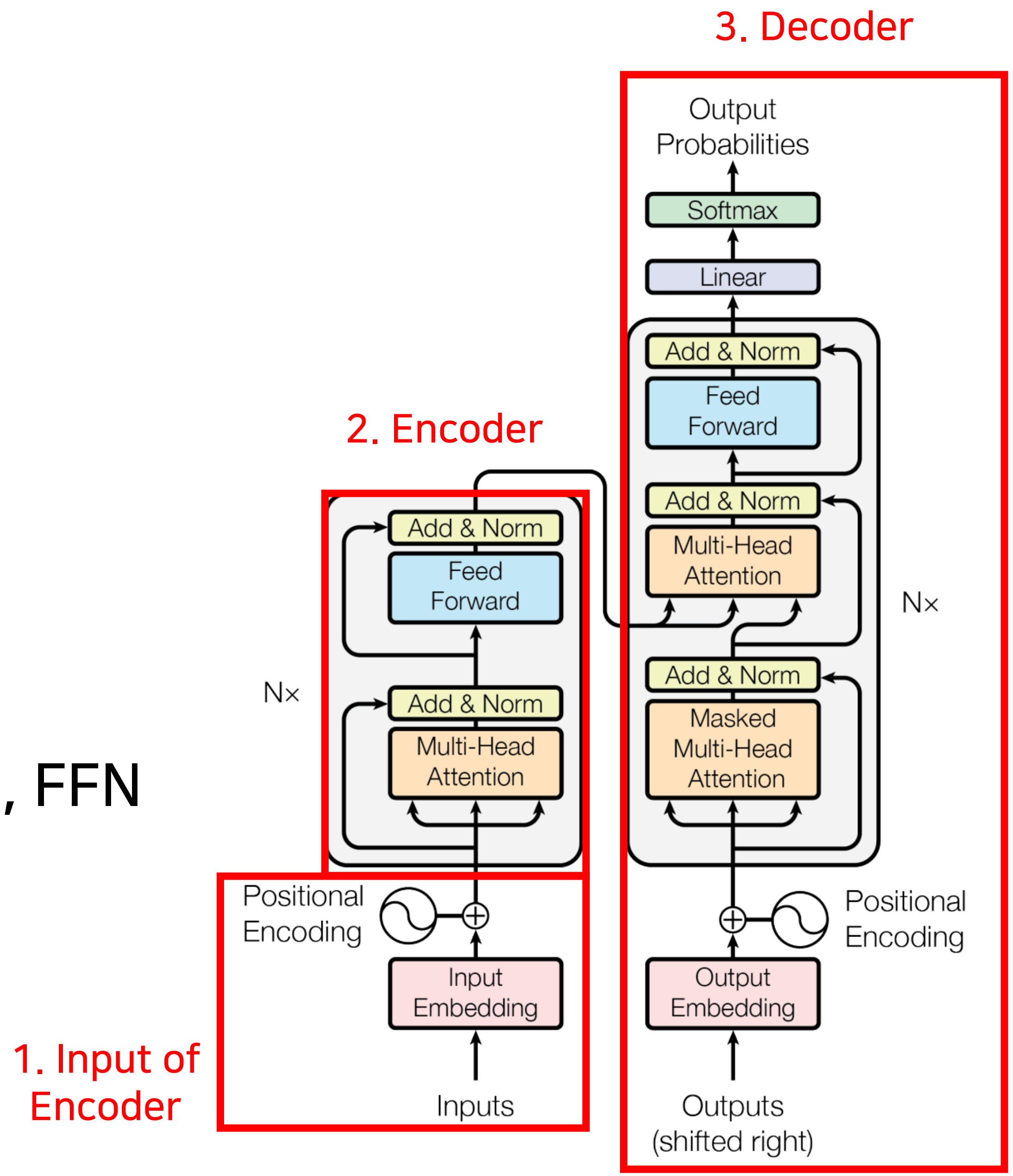
1-(2). Architecture

- Before Network : 2 Embedding

  (Input/Output, Positional)

- Encoder/Decoder :

  Multi-head attention, Add&Norm, FFN



1-(3). Implementation Reasons

- Studying deep-learning based NLP techniques
- Understanding how to user Pytorch
- Hands-on model training experience
- High utilization of Transformer architecture

Contents

2-(1). Overall Code

https://colab.research.google.com/drive/1thwDDMJ4

W3w0fZS82cZADZ8R8J_VGl4a?usp=sharing

2-(2). Prior Preparation

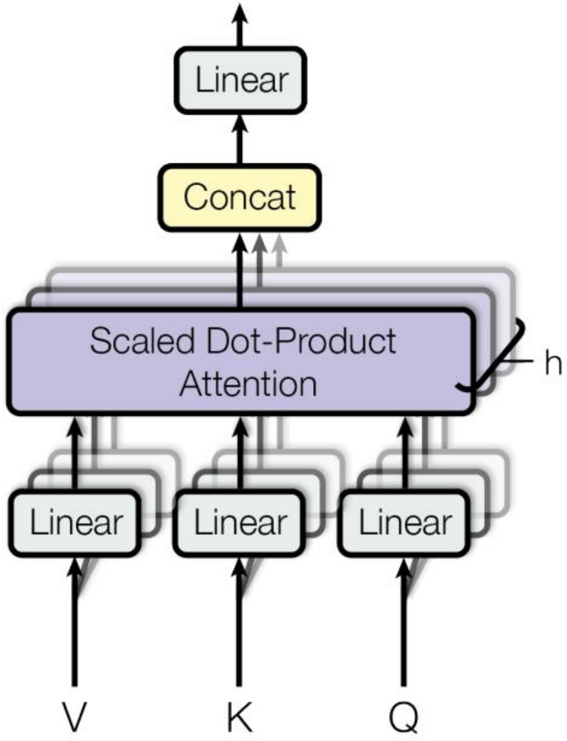
- Import Torchtext and spaCy
- Make tokenizer functions
- Specify data preprocessing to use Field library
- Load dataset (Multi30k) and make vocab
- Set batch size and iterator

2-(3). Multi-Head Attention

- Use Query, Key, Value
- In encoder, self-attention is used (Q, K, V are same)

```
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)
```

- Scaled Dot-Product Attention
- Concat results and convert it linearly → Final output



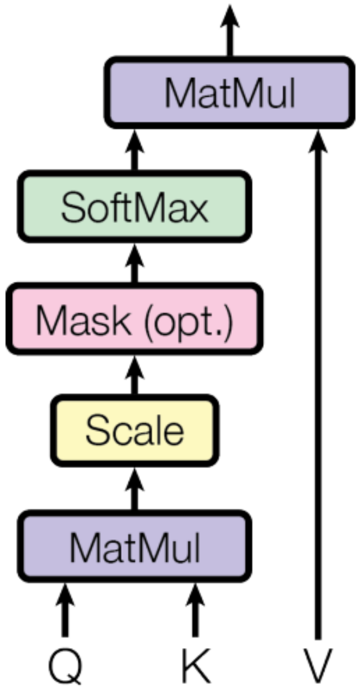
2-(3). Multi-Head Attention

1. Multiply query and key's inverse matrix
2. Divide scale factor (scaling)

→ Attention energies

$$\frac{QK^T}{\sqrt{d_k}}$$

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

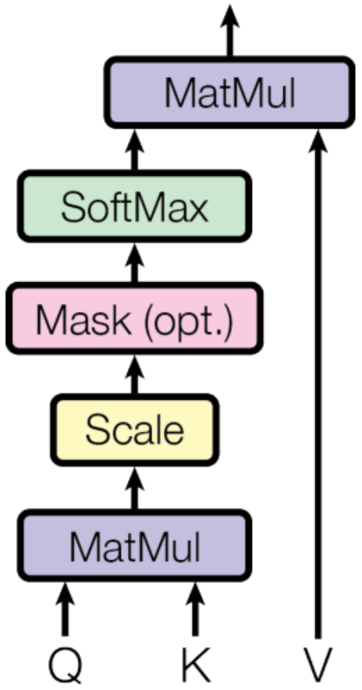


2-(3). Multi-Head Attention

3. Masking (optional)
4. Calculate attention score (probability)
5. Multiply attention score and value

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

```
attention = torch.softmax(energy, dim=-1)
x = torch.matmul(self.dropout(attention), V)
```



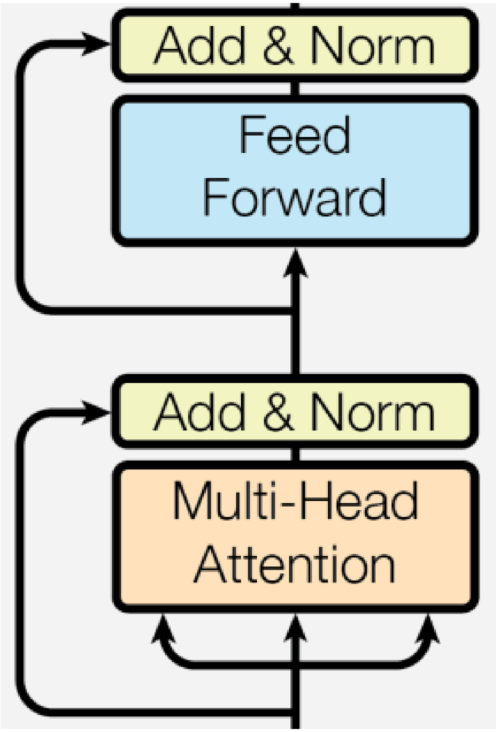
2-(4). Feed Forward Network

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

```
class PositionwiseFeedforwardLayer(nn.Module):
    def __init__(self, hidden_dim, pf_dim, dropout_ratio):
        super().__init__()

        self.fc_1 = nn.Linear(hidden_dim, pf_dim)
        self.fc_2 = nn.Linear(pf_dim, hidden_dim)

        self.dropout = nn.Dropout(dropout_ratio)

    def forward(self, x):
        x = self.dropout(torch.relu(self.fc_1(x)))
        x = self.fc_2(x)
        return x
```

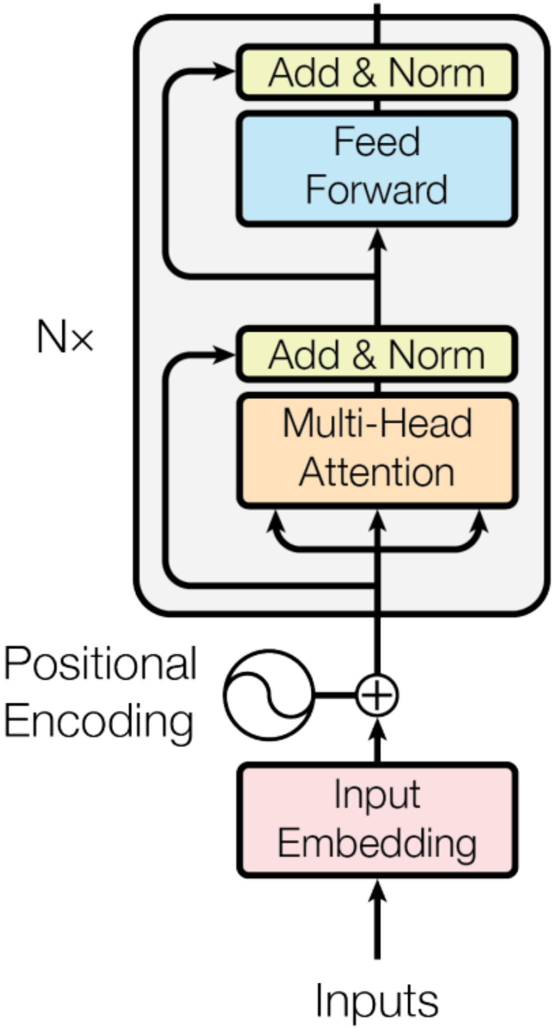
2-(5). Encoder

```
# self attention
# 필요한 경우 mask matrix로 attention을 할 단어들을 조정
_src, _ = self.self_attention(src, src, src, src_mask)

# Add & Norm
src = self.self_attn_layer_norm(src + self.dropout(_src))

# FFN
_src = self.positionwise_feedforward(src)

# Add & Norm
src = self.ff_layer_norm(src + self.dropout(_src))

return src
```



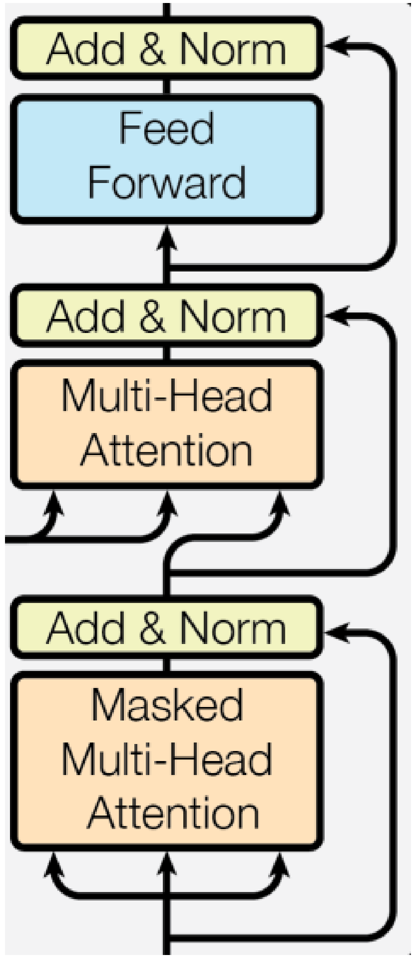
2-(5). Encoder

```
# Input embedding + Positional embedding
src = self.dropout((self.tok_embedding(src) * self.scale)
                   + self.pos_embedding(pos))

for layer in self.layers:
    src = layer(src, src_mask)

# 마지막 레이어 결과를 return
return src
```



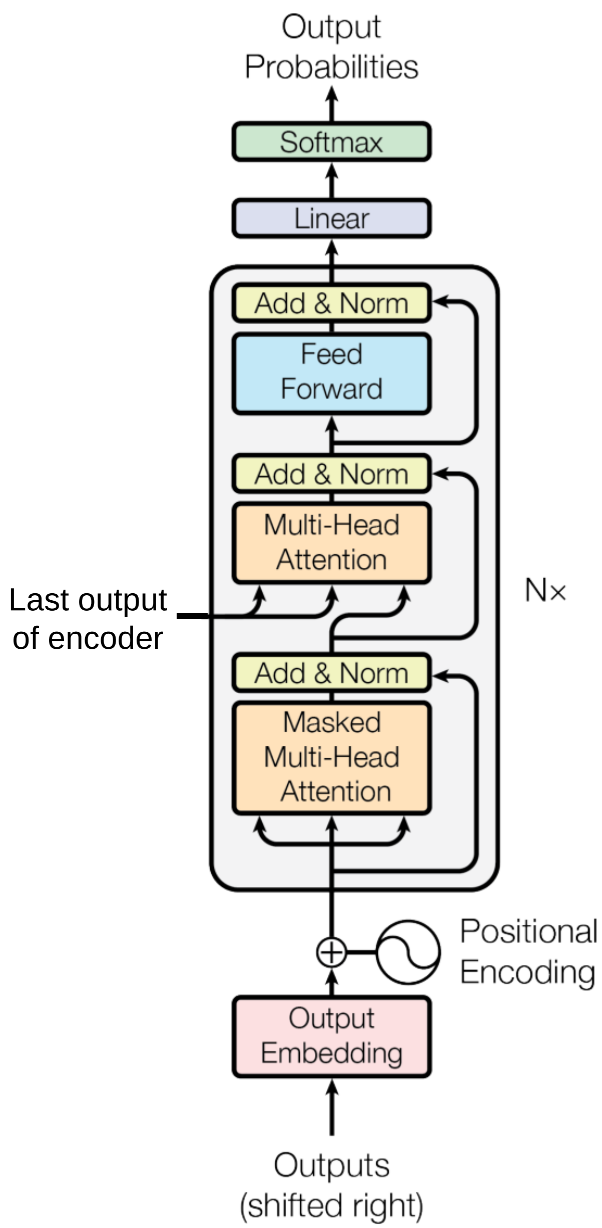
2-(6). Decoder

```
# self attention
_trg, _ = self.self_attention(trg, trg, trg, trg_mask)

# Add & Norm
trg = self.self_attn_layer_norm(trg + self.dropout(_trg))

# encoder attention
# 인코더의 출력 값(enc_src)을 attention하는 구조
# 디코더의 쿼리를 이용해 인코더를 attention
_trg, attention = self.encoder_attention(trg,enc_src,enc_src,src_mask)

# Add & Norm
trg = self.enc_attn_layer_norm(trg + self.dropout(_trg))

# FFN
_trg = self.positionwise_feedforward(trg)

# Add & Norm
trg = self.ff_layer_norm(trg + self.dropout(_trg))

return trg, attention
```



2-(6). Decoder

```
# Output embedding + Positional embedding
trg = self.dropout((self.tok_embedding(trg) * self.scale)
                   + self.pos_embedding(pos))

for layer in self.layers:
    trg, attention = layer(trg, enc_src, trg_mask, src_mask)

output = self.fc_out(trg)

# 마지막 레이어 결과(최종 번역 결과)를 return
return output, attention
```



2-(7). Transformer

- make_src_mask : Set mask value to 0 for padding token
- make_trg_mask : Set mask value to 0 for following words + make_src_mask

```
# masking
src_mask = self.make_src_mask(src)
trg_mask = self.make_trg_mask(trg)

# encoder
enc_src = self.encoder(src, src_mask)

# decoder
output, attention = self.decoder(trg, enc_src, trg_mask, src_mask)

return output, attention
```

Contents

3-(1). Parameters

```
INPUT_DIM = len(SRC.vocab)
OUTPUT_DIM = len(TRG.vocab)
HIDDEN_DIM = 256
ENC_LAYERS = 3
DEC_LAYERS = 3
ENC_HEADS = 8
DEC_HEADS = 8
ENC_PF_DIM = 512
DEC_PF_DIM = 512
ENC_DROPOUT = 0.1
DEC_DROPOUT = 0.1

SRC_PAD_IDX = SRC.vocab.stoi[SRC.pad_token]
TRG_PAD_IDX = TRG.vocab.stoi[TRG.pad_token]
```

3-(2). Key Point of Training

- Epochs : 10
- Optimize learning with Adam optimizer
- Set padding values to be ignored
- Set parameters to update only when validation loss is reduced

```python
if valid_loss < best_valid_loss:
    best_valid_loss = valid_loss
    torch.save(model.state_dict(), 'transformer_german_to_english.pt')
```

Contents

4. Translation and BLEU score

```
[100/1000]
예측: ['a', 'group', 'of', 'asian', 'children', 'are', 'sitting', 'down', 'chairs', 'in', 'blue', 'chairs', '.']
정답: ['a', 'group', 'of', 'mostly', 'asian', 'children', 'sitting', 'at', 'cubicles', 'in', 'blue', 'chairs', '.']
[200/1000]
예측: ['all', 'standing', 'in', 'the', 'group', 'of', 'people', 'standing', 'under', 'umbrellas', '.']
정답: ['the', 'group', 'of', 'people', 'are', 'all', 'covered', 'by', 'umbrellas', '.']
[300/1000]
예측: ['a', 'goalie', 'in', 'a', 'yellow', 'jersey', 'is', 'blowing', 'the', 'goal', '.']
정답: ['a', 'goalie', 'in', 'a', 'yellow', 'field', 'is', 'protecting', 'the', 'goal', '.']
[400/1000]
예측: ['two', 'young', 'children', 'on', 'the', 'sand', '.']
정답: ['two', 'young', 'children', 'are', 'on', 'sand', '.']
[500/1000]
예측: ['two', 'medium', 'sized', 'dogs', 'run', 'across', 'the', 'snow', '.']
정답: ['two', 'medium', 'sized', 'dogs', 'run', 'across', 'the', 'snow', '.']
```

```
Total BLEU Score = 35.41
Individual BLEU1 score = 67.79
Individual BLEU2 score = 43.23
Individual BLEU3 score = 28.23
Individual BLEU4 score = 19.01
Cumulative BLEU1 score = 67.79
Cumulative BLEU2 score = 54.14
Cumulative BLEU3 score = 43.57
Cumulative BLEU4 score = 35.41
```

Thank You!

Project to implement the Transformer using Pytorch