Week10_복습과제 - Attention Is All You Need 논문 정리

<abstract>
: 그간 지배적인 sequence transduction models는 encoder와 decoder를 포함한 복잡한 recurrent or convolutional neural networks를 기반으로 했으며, 가장 좋은 성능을 낸 모델은 attention mechanism을 이용한 encoder와 decoder를 모두 사용한 것이었음. 해당 논문에서는 새롭고 간단한 네트워크 구조인 Transformer를 소개하고자 하며, 이는 attention mechanism만을 기반으로하여 recurrence와 convolutions를 나누는 모델임.

1. Introduction
: 언어 모델링과 기계번역과 같은 sequence modeling& transduction problems를 다루는데 공고해진 모델들: RNN, LSTM, gated recurrent neural networks

Recurrent language models vs. encoder-decoder architectures
- Recurrent models: 주로 the symbol positions of input and output sequences를 계산하는데 많은 영향: aligning the positions to steps in computation time, generating a sequence of hidden states, the previous hidden state and the input for position -> 길이 길어질수록 문제 야기

Attention mechanisms: integral part of compelling sequence modeling and transduction models. input 또는 output 내 거리에 의존하지 않음. 그러나 단독으로 사용되기 보다는 recurrent network와 conjunction되어 사용됨.

Transformer: a model architecture eschewing recurrence and instead relying entirely on an attention mechanism

2. Background
: sequential computation을 줄이는 것을 목표로 Extended Neural GPU, ByteNet and ConvS2S이 개발됨. 이들은 basic block으로 convolutional neural networks를 사용하며, 모든 input과 output position에 수평적으로 hidden representations를 계산
the number of operations: relate signals from two arbitrary input of output positions grows in the distance between positions를 필요로 함 - ConvS2S는 linearly/ByteNet은 logarithmically
--> 이는 먼 거리 위치들 간 dependencies를 학습하는 것을 더 어렵게 만듦.

Transformer: constant number of operations를 줄임,(비록 averaging attention-weighted positions 때문에 the cost of reduced effective resolution이 발생해도)
Self-attention(intra-attention): single sequence의 각각 다른 positions를 연결하는 attention mechanism, -> reading comprehension, abstractive summarization, textual entailment and learning task-independent sentence representations 문제를 해결하는

데 성공적 결과를 보여옴.

End-to-end memory networks: sequence-aligned recurrence 대신 recurrent attention mechanism 기반, -> simple-language question answering and language modeling task에 좋은 성과 보여옴.

Transformer의 의의: the first transduction model relying entirely on self-attention to compute representations of its input and output without using sequence-aligned RNNs or convolution

3. Model Architecture

: 대부분의 neural sequence transduction models는 encoder-decoder 구조를 가짐.
input x -> z -> output y
transformer는 이러한 구조를 stacked self-attention과 point-wise하게 이용하며, input과 output 모두에 fully connected layers.

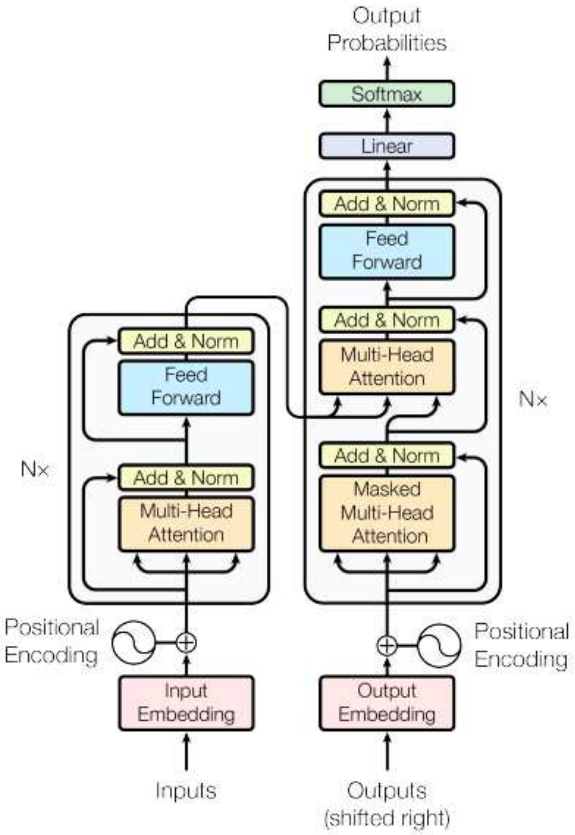

Figure 1: The Transformer - model architecture.

* Encoder: 6개의 동일한 layers의 stack 형태로 구성됨. 각각의 layer는 두 개의 sub-layers를 가짐. 첫 번째는 multi-head self-attention mechanism이고, 두 번째는

simple, position-wise fully connected fee-forward network임. -> residual connection around each of the two sub-layers, followed by layer normalization.

- the output of each sub-layer: $LayerNorm(x + Sublayer(x))$
- produce outputs of dimension $d_{model} = 512$(residual connections, sub-layers, embedding layers 포함)

* Decoder: 역시 6개의 동일한 stack 형태의 layers로 구성됨. encoder와 마찬가지로 각각 두 개씩의 sub-layers를 가지며 추가로 third sub-layer를 가짐. 이는 multi-head attention over the ouput of the encoder stack의 역할을 함. residual connection around each of the sub-layers, followed by layer normalization. -> we also modify the self-attention sub-layer in the decoder stack to prevent positions from attending to subsequent positions.

- the output embeddings are offset by one position, ensures that the predictions for position i can depend only on the known outputs at positions less than i.

3.2. Attention

: attention fuction은 mapping a query and a set of key-value pairs to an output, where the query, keys, values, and output are all vectors로 설명될 수 있음.
이 output은 값들의 가중합으로 계산됨. 각 value에 배치된 가중치는 key에 맞는 query의 compatibility function에 의해 계산됨.

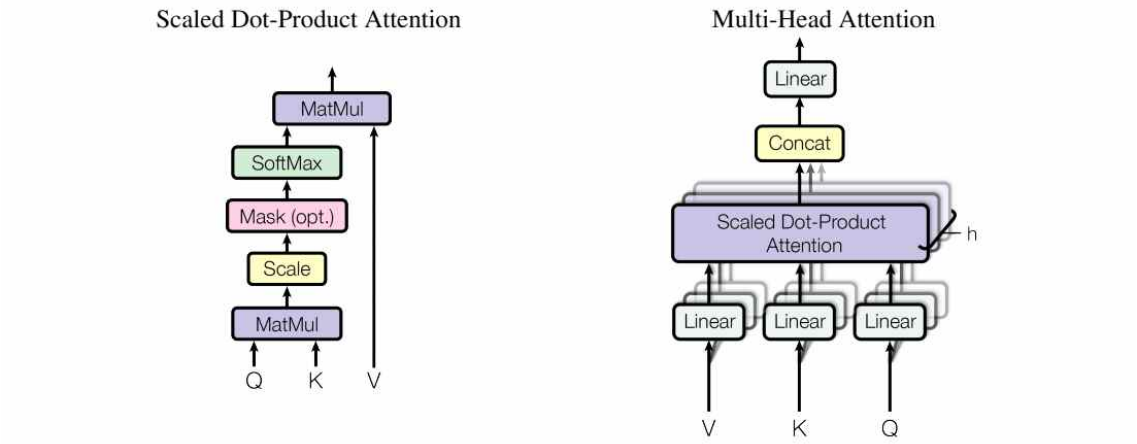

Figure 2: (left) Scaled Dot-Product Attention. (right) Multi-Head Attention consists of several attention layers running in parallel.

3.2.1. Scaled Dot-Product Attention

: d_k에서의 queries and keys, d_v에서의 values로 구성된 input -> “Scaled Dot-Product Attention” -> compute the dot products of the query with all keys, dived each by $\sqrt{d_k}$, and apply a softmax fuction to obtain the weights on the values.

: we compute the attention fuction on a set of queries simultaneously, packed together into a matrix Q.

- output of Attention: $Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V$
- The two most commonly used attention functions: additive attention, dot-product attention
- Dot product attention: identical to our algorithm, except for the scaling factor of $\frac{1}{\sqrt{d_k}}$
- Additive attention: computes the compatibility fuction using a fee-forward network with a single hidden layer.
- dot product attention is much faster and more space-efficient in practice

-d_k:small value -> two mechanisms perform similarly

3.2.2 Multi-Head Attention

: the model to jointly attend to information from different representation subspaces at different positions. With a single attention head, averaging inhibits this.

$$MultiHead(Q, K, V) = Concat(head_1, ..., head_p)W^O$$

where $head_i = Attention(QW_i^Q, KW_i^K, VW_i^V)$

--> reduced dimension of each head, the total computational cost is similar to that of single-head attention with full dimensionality.

3.2.3 Applications of Attention in our Model

- In “encoder-decoder attention” layers, the queries come from the previous decoder layer, and the memory keys and values come from the output of the encoder. -> every position in the decoder attend over all position in the input sequence.
- The encoder contains self-attention layers. In a self-attention layer all of the keys, values, queries come from the same place. -> each position in the encoder can attend to all positions in the previous layer of the encoder.
- each position in the decoder attend to all positions in the decoder up and including that position.

3.3 Position-wise Feed-Forward Networks

: attention sub-layers, each of the layers in our encoder and decoder contains a fully connected feed-forward network, which is applied to each position separately and identically. -> two linear transformations with a ReLU activation in between.

$$FFN(x) = \max(0, x W_1 + b_{1)} W_2 + b_2$$

● While the linear transformations are the same across different positions, they use different parameters from layer to layer.

3.4 Embeddings and Softmax

: use learned embeddings to convert the input tokens and output tokens to vectors of dimension d_{model}.
: use the usual learned linear transformation and softmax function to convert the decoder output to predicted next-token probabilities.

3.5 Positional Encoding

: In order model to make use of the order of the sequence, inject some information about the relative or absolute position of the tokens in the sequence.
: To this end, we add “positional encodings” to the input embeddings at the bottoms of the encoder and decoder stacks.

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

4. Why Self-Attention

- One is the total computational complexity per layer. : faster
- the amount of computation that can be parallelized, as measured by the minimum number of sequential operations required. : cheaper
- the path length beetween long-range dependencies in the network. : decrease complexity

6. Results
Machine Translation, Model variations, English Constituency Parsing에서 모두 convolutional neural networks보다 우월