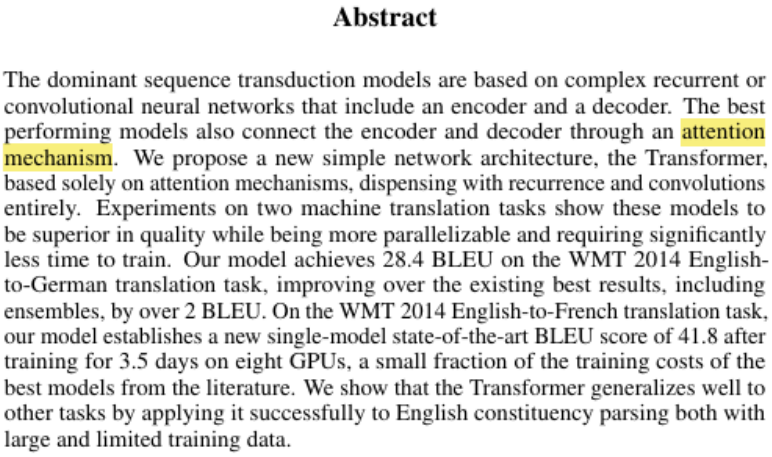
✅

Attention Is All You Need(논문뜯기)



Abstract

The dominant sequence transduction models are based on complex recurrent or convolutional neural networks that include an encoder and a decoder. The best performing models also connect the encoder and decoder through an attention mechanism. We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely. Experiments on two machine translation tasks show these models to be superior in quality while being more parallelizable and requiring significantly less time to train. Our model achieves 28.4 BLEU on the WMT 2014 English-to-German translation task, improving over the existing best results, including ensembles, by over 2 BLEU. On the WMT 2014 English-to-French translation task, our model establishes a new single-model state-of-the-art BLEU score of 41.8 after training for 3.5 days on eight GPUs, a small fraction of the training costs of the best models from the literature. We show that the Transformer generalizes well to other tasks by applying it successfully to English constituency parsing both with large and limited training data.

▼ 해석본

✅ 지배적인 시퀀스 변환 모델은 인코더와 디코더를 포함하는 복잡한 순환 신경망(RNN) 또는 합성곱 신경망(CNN)에 기반을 두고 있습니다. 최고의 성능을 내는 모델들은 인코더와 디코더를 어텐션 메커니즘을 통해 연결합니다. 우리는 순환이나 합성곱을 완전히 배제하고 오직 어텐션 메커니즘만을 사용하는 새로운 간단한 네트워크 아키텍처인 Transformer를 제안합니다. 두 가지 기계 번역 작업에 대한 실험 결과, 이 모델들은 품질 면에서 우수하며 더 병렬화가 가능하고 훈련 시간이 상당히 단축됩니다. 우리의 모델은 WMT 2014 영어-독일어 번역 작업에서 BLEU 점수 28.4를 기록하며, 기존의 최고 성능을 내는 결과들(앙상블 포함)을 2점 이상 초과했습니다. WMT 2014 영어-프랑스어 번역 작업에서도, 우리의 모델은 8개의 GPU로 3.5일 동안 훈련하여 BLEU 점수 41.8의 새로운 단일 모델 상태를 달성했습니다. 이는 문헌에서 보고된 최고의 모델들보다 훨씬 적은 훈련 비용을 요구합니다. 우리는 Transformer가 큰 데이터 세트와 제한된 데이터 세트를 모두 사용하는 영어 구문 분석 작업에서도 성공적으로 적용될 수 있음을 보여줍니다.

1 Introduction

Recurrent neural networks, long short-term memory [13] and gated recurrent [7] neural networks in particular, have been firmly established as state of the art approaches in sequence modeling and transduction problems such as language modeling and machine translation [35, 2, 5]. Numerous efforts have since continued to push the boundaries of recurrent language models and encoder-decoder architectures [38, 24, 15].

Recurrent models typically factor computation along the symbol positions of the input and output sequences. Aligning the positions to steps in computation time, they generate a sequence of hidden states h_t, as a function of the previous hidden state h_{t-1} and the input for position t. This inherently sequential nature precludes parallelization within training examples, which becomes critical at longer sequence lengths, as memory constraints limit batching across examples. Recent work has achieved significant improvements in computational efficiency through factorization tricks [21] and conditional computation [32], while also improving model performance in case of the latter. The fundamental constraint of sequential computation, however, remains.

Attention mechanisms have become an integral part of compelling sequence modeling and transduction models in various tasks, allowing modeling of dependencies without regard to their distance in the input or output sequences [2, 19]. In all but a few cases [27], however, such attention mechanisms are used in conjunction with a recurrent network.

In this work we propose the Transformer, a model architecture eschewing recurrence and instead relying entirely on an attention mechanism to draw global dependencies between input and output. The Transformer allows for significantly more parallelization and can reach a new state of the art in translation quality after being trained for as little as twelve hours on eight P100 GPUs.

▼ 해석본

✅ 1. Introduction

순환 신경망(RNN), 장단기 기억 네트워크(LSTM), 게이트 순환 유닛(GRU) 신경망은 언어 모델링 및 기계 번역과 같은 시퀀스 모델링 및 변환 문제에서 최첨단 방법으로 확립되었습니다. 수많은 연구가 이러한 순환 언어 모델 및 인코더-디코더 아키텍처의 한계를 넘어서는 노력을 계속하고 있습니다.

순환 모델은 일반적으로 입력 및 출력 시퀀스의 기호 위치를 시간 계산 단계로 정렬합니다. 그 결과, t 위치의 입력에 대한 이전 은닉 상태 $ht - 1h_{t-1}ht - 1$의 함수로서 은닉 상태 hth_tht의 시퀀스를 생성합니다. 이러한 순차적 성격은 훈련 예제 내에서 병렬 처리를 방해하여 메모리 제약으로 인해 예시 간의 배치 처리가 제한됩니다. 최근 연구에서는 인수 분해 기법과 조건부 계산을 통해 계산 효율성을 크게 향상시켰습니다. 그러나 순차 계산의 근본적인 제약은 여전히 남아 있습니다.

어텐션 메커니즘은 다양한 작업에서 시퀀스 모델링 및 변환 모델의 중요한 요소가 되었습니다. 이는 입력이나 출력 시퀀스에서 거리에 상관없이 의존성을 모델링할 수 있게 합니다. 대부분의 경우 어텐션 메커니즘은 순환 신경망과 함께 사용됩니다.

이 논문에서는 순환을 배제하고 입력과 출력 간의 전역 의존성을 도출하는 데 전적으로 어텐션 메커니즘을 사용하는 Transformer라는 모델 아키텍처를 제안합니다. Transformer는 병렬 처리를 크게 향상시키며, 8개의 P100 GPU에서 단 12시간 만에 훈련되어 번역 품질에서 새로운 최첨단 성능을 달성할 수 있습니다.

- 과거 순환신경망은, 순차적 성격때문에 병렬 처리를 방해하여 메모리 제약으로 인해 배치 처리가 제한되었습니다.
- “어텐션 메커니즘”은 거리에 상관없이 의존성을 모델링할 수 있고, 병렬 처리를 크게 향상시켜 시간단축에 큰 변화를 이루어냈습니다.

2 Background

The goal of reducing sequential computation also forms the foundation of the Extended Neural GPU [16], ByteNet [18] and ConvS2S [9], all of which use convolutional neural networks as basic building block, computing hidden representations in parallel for all input and output positions. In these models, the number of operations required to relate signals from two arbitrary input or output positions grows in the distance between positions, linearly for ConvS2S and logarithmically for ByteNet. This makes it more difficult to learn dependencies between distant positions [12]. In the Transformer this is reduced to a constant number of operations, albeit at the cost of reduced effective resolution due to averaging attention-weighted positions, an effect we counteract with Multi-Head Attention as described in section 3.2.

Self-attention, sometimes called intra-attention is an attention mechanism relating different positions of a single sequence in order to compute a representation of the sequence. Self-attention has been used successfully in a variety of tasks including reading comprehension, abstractive summarization, textual entailment and learning task-independent sentence representations [4, 27, 28, 22].

End-to-end memory networks are based on a recurrent attention mechanism instead of sequence-aligned recurrence and have been shown to perform well on simple-language question answering and language modeling tasks [34].

To the best of our knowledge, however, the Transformer is the first transduction model relying entirely on self-attention to compute representations of its input and output without using sequence-aligned RNNs or convolution. In the following sections, we will describe the Transformer, motivate self-attention and discuss its advantages over models such as [17, 18] and [9].

▼ 해석본

✅ 2. Background

순차적 계산을 줄이려는 목표는 Extended Neural GPU, ByteNet, ConvS2S와 같은 합성곱 신경망(CNN)을 기본 구성 요소로 사용하는 모델의 기반을 형성합니다. 이러한 모델은 입력과 출력 위치에서 은닉 표현을 병렬로 계산합니다. 하지만 두 임의의 입력 또는 출력 위치 간의 신호 관계를 연관시키는 데 필요한 연산 수는 거리에 비례하여 증가합니다. ConvS2S는 선형적으로, ByteNet은 로그 비례적으로 증가합니다. 이는 먼 위치들 간의 의존성을 학습하는 데 어려움을 줍니다. Transformer에서는 상수의 연산 수로 줄일 수 있지만, 어텐션 가중치 위치의 평균화로 인해 해상도가 줄어드는 문제가 발생합니다. 이를 다중 헤드 어텐션(Multi-Head Attention)으로 보완합니다.

자기 어텐션(Self-attention)은 시퀀스 내의 서로 다른 위치를 연관시켜 시퀀스의 표현을 계산하는 메커니즘으로, 다양한 작업에서 성공적으로 사용되었습니다. 대표적인 작업으로는 독해, 추상적 요약, 텍스트 간 의미 관계 학습 등이 있습니다.

엔드 투 엔드 메모리 네트워크는 시퀀스 정렬된 순환이 아닌 반복 어텐션 메커니즘을 기반으로 하며, 간단한 언어 질문 응답 및 언어 모델링 작업에서 잘 작동하는 것으로 나타났습니다.

우리의 지식에 따르면 Transformer는 시퀀스 정렬된 RNN이나 합성곱 없이 입력과 출력의 표현을 계산하기 위해 전적으로 자기 어텐션에 의존하는 첫 번째 변환 모델입니다. 이후 섹션에서는 Transformer를 설명하고, 자기 어텐션을 사용하는 이유 및 기존 모델과의 비교를 다룰 것입니다.

- Extended Neural GPU, ByteNet, ConvS2S 모델들은 CNN을 기본구성으로 하여 거리가 멀어질 수록 의존성이 떨어지게 됩니다.
- Transformer→ 의존성 문제를 상수의 연산으로 줄일 수 있지만, 어텐션 가중치 위치 평균화로 인해 해상도가 줄어드는 문제가 발생한다. 이를 Multi-Head Attention으로 해결합니다.
- Self-Attetion
  - Self Attention에서는 Query, Key, Value 라는 3가지 변수가 존재한. 직역하자면 Query는 "의문", Key는 "열쇠", Value는 "값"을 의미합니다.
  - Self Attention의 에서 가장 중요한 개념은 Query, Key, Value의 시작 값이 동일하다는 점입니다. 때문에 Self를 붙이는 것 입니다. 그러나 이 말이 Query, Key, Value이 동일하다는 말은 아닙니다. 그 이유는 위 그림과 같이 중간에 학습 weight

W값에 의해 최종적인 Query, Key, Value값은 서로 다르게 됩니다. 다시 말해 "Query, Key, Value의 기원은 같으나 최종 값은 학습을 통해 달라진다."입니다.

- “I am a studen”임베딩 후 계산해보기

  Self Attetion계산 예제

3 Model Architecture

Most competitive neural sequence transduction models have an encoder-decoder structure [5, 2, 35]. Here, the encoder maps an input sequence of symbol representations $(x_1, ..., x_n)$ to a sequence of continuous representations $\mathbf{z} = (z_1, ..., z_n)$. Given $\mathbf{z}$, the decoder then generates an output sequence $(y_1, ..., y_m)$ of symbols one element at a time. At each step the model is auto-regressive [10], consuming the previously generated symbols as additional input when generating the next.

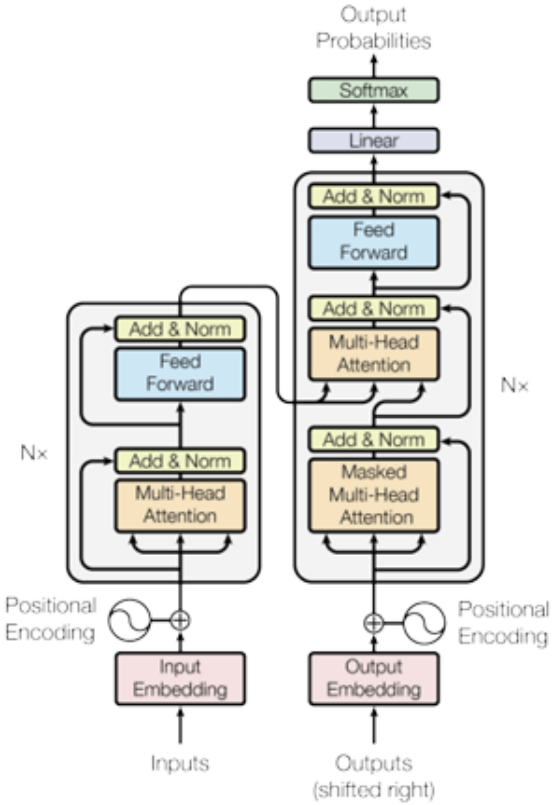
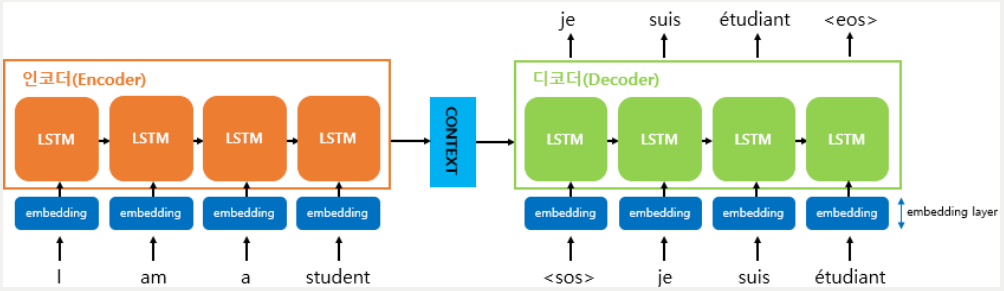


Figure 1: The Transformer - model architecture.

The Transformer follows this overall architecture using stacked self-attention and point-wise, fully connected layers for both the encoder and decoder, shown in the left and right halves of Figure 1, respectively.

▼ 해석본

✅ 대부분의 sequence trasduction model은 아래와 같은 encoder-decoder구조를 가진다 (아래는 seq2seq 예시이다). 이런 모델의 각 단계는 auto regressive이며, 이전에 생성된 output을 다음 단계에서 사용한다 **(이전 단계가 완료가 된 후에 다음 단계를 수행할 수 있다는 말 = 순차적으로 진행 = 병렬적으로 처리가 불가능함)**.



3.1 Encoder and Decoder Stacks

Encoder: The encoder is composed of a stack of $N = 6$ identical layers. Each layer has two sub-layers. The first is a multi-head self-attention mechanism, and the second is a simple, position-wise fully connected feed-forward network. We employ a residual connection [11] around each of the two sub-layers, followed by layer normalization [1]. That is, the output of each sub-layer is $\mathrm{LayerNorm}(x + \mathrm{Sublayer}(x))$, where $\mathrm{Sublayer}(x)$ is the function implemented by the sub-layer itself. To facilitate these residual connections, all sub-layers in the model, as well as the embedding layers, produce outputs of dimension $d_{\text{model}} = 512$.

Decoder: The decoder is also composed of a stack of $N = 6$ identical layers. In addition to the two sub-layers in each encoder layer, the decoder inserts a third sub-layer, which performs multi-head attention over the output of the encoder stack. Similar to the encoder, we employ residual connections 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. This masking, combined with fact that 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. Model Architecture + 3.1 Encoder and Decoder Stack

Figure1의 왼쪽의 회색 큰 박스는 Encoder 오른쪽의 회색 큰 박스 Decoder의 구조이고, N개를 쌓아 올려 레이어층을 깊게 만드는 구조입니다. 각각 **Input Embedding**과 **Positional Encoding**을 거쳐 입력되며 Encoder, Decoder에 **총 3가지 Attention** 이 있습니다.

Encoder: 첫 번째 주황색 Sub-Layer는 **Multi-Head Encoder Self-Attention**, 파란색 Sub-Layer는 단순한 Feed-Forward입니다. 두 Sub-Layer 각각 Residual Connection(잔차연결)을 사용하고 Layer Normalization을 사용합니다. N개의 동일한 층이 쌓이고 마지막 Layer가 Decoder의 **Multi-Head Encoder-Decoder Attention**으로 입력됩니다. 본 논문에서는 N=6으로 동일한 6개의 층을 쌓아 올렸고 출력 차원은 512로 설정하였습니다. 처음 Encoder에 입력부터 출력까지 총 512차원으로 고정됩니다.

Decoder: 첫 번째 주황색 Sub-Layer는 **Masked Multi-Head Decoder Self-Attention,** 두 번째 주황색 Sub-Layer는 Encoder에서의 출력으로 입력된 **Multi-Head Encoder-Decoder Attention**, 파란색 Sub-Layer는 마찬가지로 단순한 Feed-Forward이고 Residual Connection과 Layer Normalization으로 동일하게 구성됩니다.

3.2 Attention

An attention function can be described as mapping a query and a set of key-value pairs to an output, where the query, keys, values, and output are all vectors. The output is computed as a weighted sum

▼ 해석본

왼쪽 그림은 한 개의 Attention을 표현한 것이고, 이러한 Attention이 여러 개 모여서 만들어진 것이 오른쪽에 Multi-Head Attention입니다.

<Attention>

- 같은 문장 내에서 단어들 간의 관계를 나타내는 것입니다. 이 관계를 Query, Key, Value 3가지로 표현하여 하나의 출력으로 나타냅니다.
- Attention function은 query와 key-value 집합 쌍을 query, keys, values, output이 모두 벡터인 output에 매핑하는 것으로 설명할 수 있습니다. Output은 values의 weighted sum으로 계산되며, 여기서 각 value에 할당된 weight는 query와 해당 key의 compatibility function에 의해 계산됩니다.

Q: 영향을 받는 벡터

K: 영향을 주는 벡터

V: 주는 영향의 가중치 벡터

<Scaled Dot-Product Attention>

3.2.1 Scaled Dot-Product Attention

We call our particular attention "Scaled Dot-Product Attention" (Figure 2). The input consists of queries and keys of dimension d_k, and values of dimension d_v. We compute the dot products of the query with all keys, divide each by $\sqrt{d_k}$, and apply a softmax function to obtain the weights on the values.

In practice, we compute the attention function on a set of queries simultaneously, packed together into a matrix Q. The keys and values are also packed together into matrices K and V. We compute the matrix of outputs as:

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V \quad (1)$$

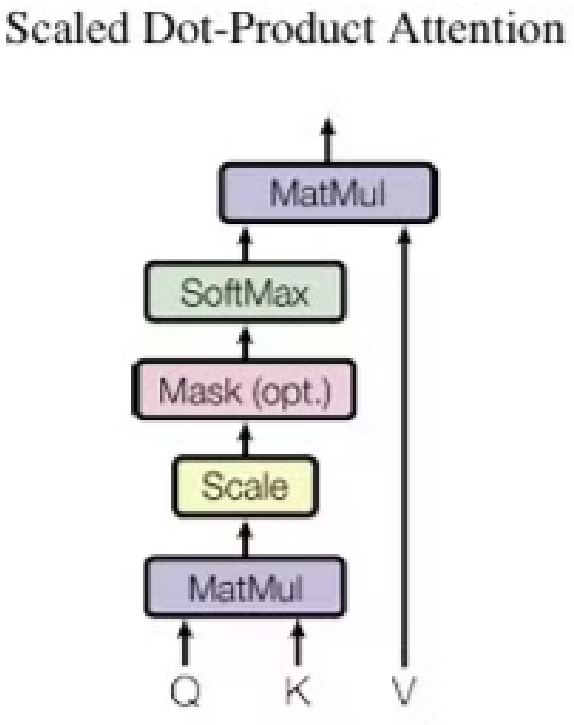
The two most commonly used attention functions are additive attention [2], and dot-product (multiplicative) attention. Dot-product attention is identical to our algorithm, except for the scaling factor of $\frac{1}{\sqrt{d_k}}$. Additive attention computes the compatibility function using a feed-forward network with a single hidden layer. While the two are similar in theoretical complexity, dot-product attention is much faster and more space-efficient in practice, since it can be implemented using highly optimized matrix multiplication code.

While for small values of d_k the two mechanisms perform similarly, additive attention outperforms dot product attention without scaling for larger values of d_k [3]. We suspect that for large values of d_k, the dot products grow large in magnitude, pushing the softmax function into regions where it has extremely small gradients [4]. To counteract this effect, we scale the dot products by $\frac{1}{\sqrt{d_k}}$.

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V \quad (1)$$

- Q와 K로 Attention Score를 구하고 Q, K, V Dimension(d_k)(여기서 k는 head의 개수)의 루트만큼 나누어 softmax를 취해주고 이 확률값에 V를 곱하여 구한다. →분자식 관련
- dot - product attention은matrix를 통해 최적화된 연산을 구현할 수 있기 때문에 훨씬 빠르고 공간 효율적이다. 하지만, Attention function의 다른방법인 Additive attention은single hidden layer가 있는 FFN을 사용하여 compatibility function을 계산한다. (덜 효율적이다)
- d_k가 값이 작은 경우에는 dot-product와 scaled dot-product가 유사하게 수행되지만, 값이 큰 경우에는 후자가 우수하다. d_k값이 클 때, dot-product의 size가 커지면서

softmax를 극도로 작은 gradient를 갖게끔 한다. 이를 개선하기 위해 $1/\sqrt{(d_k)}$**만큼 스케일링**한다. →분모식 관련



<Multi-Head Attention>

3.2.2 Multi-Head Attention

Instead of performing a single attention function with d_{model}-dimensional keys, values and queries, we found it beneficial to linearly project the queries, keys and values h times with different, learned linear projections to d_k, d_k and d_v dimensions, respectively. On each of these projected versions of queries, keys and values we then perform the attention function in parallel, yielding d_v-dimensional

[4]To illustrate why the dot products get large, assume that the components of q and k are independent random variables with mean 0 and variance 1. Then their dot product, $q \cdot k = \sum_{i=1}^{d_k} q_i k_i$, has mean 0 and variance d_k.

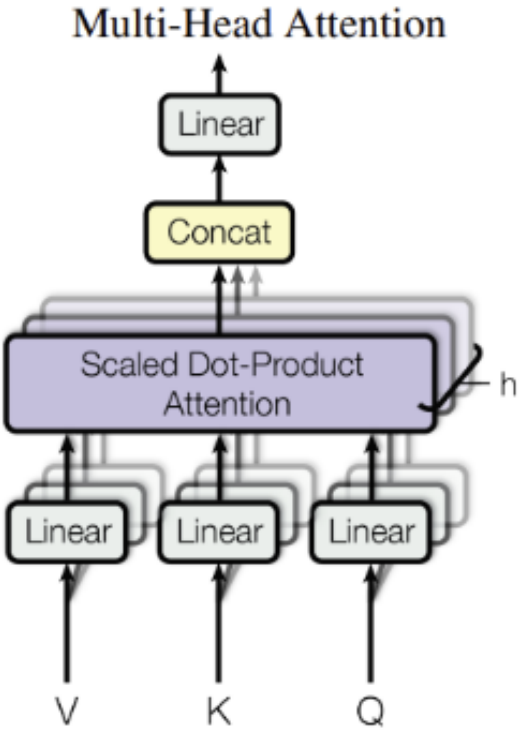
output values. These are concatenated and once again projected, resulting in the final values, as depicted in Figure 2.

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

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_\text{h})W^O$$
$$\text{where head}_\text{i} = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

Where the projections are parameter matrices $W_i^Q \in \mathbb{R}^{d_{\text{model}} \times d_k}$, $W_i^K \in \mathbb{R}^{d_{\text{model}} \times d_k}$, $W_i^V \in \mathbb{R}^{d_{\text{model}} \times d_v}$ and $W^O \in \mathbb{R}^{hd_v \times d_{\text{model}}}$.

In this work we employ $h = 8$ parallel attention layers, or heads. For each of these we use $d_k = d_v = d_{\text{model}}/h = 64$. Due to the reduced dimension of each head, the total computational cost is similar to that of single-head attention with full dimensionality.



~~d_model 차원의 query, key, value를 사용하여 single attention을 수행하는 대신,~~ 각각 d_k, d_k, d_v 차원에 대해 학습된 서로 다른 linear projection을 사용하여 query, key, value를 h회 linear projection하는 것이 유익하다는 것입니다.

Multi-Head Attention은각각의 동일한 query, key, value vector에 관해 동시에 병렬적으로 attention을 구하는 방법이며, 앞서 설명한 Scaled Dot-Product Attention을 여러 개의 head로 나누어 수행하는 것입니다.

이러한 query, key, value의 각 projection version에서 attention funtction을 **병렬로 수행하여 d_v차원 output을 생성하고, 이를 concat하여 다시 d_model 차원의 output이 생성되었습니다.**

> 💡CONCAT 함수는 둘 이상의 문자열을 입력한 순서대로 합쳐서 반환해주는 함수이다.

3.2.3 Applications of Attention in our Model

The Transformer uses multi-head attention in three different ways:

- 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. This allows every position in the decoder to attend over all positions in the input sequence. This mimics the typical encoder-decoder attention mechanisms in sequence-to-sequence models such as [38, 2, 9].
- The encoder contains self-attention layers. In a self-attention layer all of the keys, values and queries come from the same place, in this case, the output of the previous layer in the encoder. Each position in the encoder can attend to all positions in the previous layer of the encoder.
- Similarly, self-attention layers in the decoder allow each position in the decoder to attend to all positions in the decoder up to and including that position. We need to prevent leftward information flow in the decoder to preserve the auto-regressive property. We implement this inside of scaled dot-product attention by masking out (setting to $-\infty$) all values in the input of the softmax which correspond to illegal connections. See Figure 2.

▼ 해석본

✅ 3.2.3 모델에서의 어텐션 응용

트랜스포머는 멀티-헤드 어텐션을 세 가지 방식으로 사용한다:

- "인코더-디코더 어텐션" 레이어에서는 쿼리가 이전 디코더 레이어에서 오고, 메모리 키와 값은 인코더의 출력에서 온다. 이를 통해 디코더의 각 위치가 입력 시퀀스의 모든 위치에 주목(attend)할 수 있다. 이는 시퀀스-투-시퀀스 모델에서의 전형적인 인코더-디코더 어텐션 메커니즘을 모방한다.
- 인코더에는 셀프-어텐션 레이어가 포함된다. 셀프-어텐션 레이어에서는 모든 키, 값, 쿼리가 같은 곳에서 오는데, 이 경우는 인코더의 이전 레이어 출력이다. 인코더의 각 위치는 인코더의 이전 레이어에서 모든 위치에 주목할 수 있다.
- 마찬가지로, 디코더의 셀프-어텐션 레이어는 디코더의 각 위치가 해당 위치까지의 모든 위치에 주목할 수 있도록 한다. 우리는 디코더에서 왼쪽 방향으로 정보가 흐르는 것을 방지하여 자기 회귀적(auto-regressive) 특성을 유지해야 한다. 이는 스케일된 닷 프로덕트 어텐션(scaled dot-product attention) 내부에서 소프트맥스 입력의 불법 연결에 해당하는 모든 값을 $-\infty$로 설정하여 마스킹함으로써 구현된다.

3.3 Position-wise Feed-Forward Networks

In addition to 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. This consists of two linear transformations with a ReLU activation in between.

$$\mathrm{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2 \tag{2}$$

While the linear transformations are the same across different positions, they use different parameters from layer to layer. Another way of describing this is as two convolutions with kernel size 1. The dimensionality of input and output is $d_{\text{model}} = 512$, and the inner-layer has dimensionality $d_{ff} = 2048$.

▼ 해석본

✅ attention sub-layer 외에도 encoder와 decoder의 각 layer에는 FFN(feed-forward-network)가 포함되어 있으며, 이는 각 position에 개별적으로 동일하게 적용된다. 이것은 중간에 ReLU activation이 있는 두 가지 linear transformation으로 구성된다.

$$\mathrm{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2 \tag{2}$$

linear transformation은 여러 position에서 동일하지만 layer마다 다른 파라미터를 사용한다. 이것을 설명하는 또 다른 방법은 커널 크기가 1인 두 개의 convolution을 사용하는 것이다. Input과 output의 차원은 d_model=512이고, 내부 layer의 차원은 d_ff=2048이다.

- FFN은 입력 데이터를 처리하는 작은 신경망과 비슷합니다 :

1. **첫 번째 선형 변환**은 입력을 512 차원에서 **2048 차원**으로 확장 .
2. 그 후 **ReLU**가 적용되어 비선형성을 추가 .
3. **두 번째 선형 변환**은 다시 차원을 2048에서 **512 차원**으로 줄임.

이 과정을 거치면 각 위치에서 좀 더 복잡한 특성을 추출할 수 있게 되는데, 이 FFN은 레이어마다 파라미터가 달라져서 각 레이어가 다른 방식으로 입력을 처리할 수 있습니다.

또 다른 방식으로 설명하면, 이것은 커널 크기가 1인 두 개의 **컨볼루션**처럼 작동하는데, 이는 입력의 모든 위치를 독립적으로 처리하지만 레이어마다 다른 파라미터를 사용한다는 것을 의미합니다.

3.4 Embeddings and Softmax

Similarly to other sequence transduction models, we use learned embeddings to convert the input tokens and output tokens to vectors of dimension d_{model}. We also use the usual learned linear transformation and softmax function to convert the decoder output to predicted next-token probabilities. In our model, we share the same weight matrix between the two embedding layers and the pre-softmax linear transformation, similar to [30]. In the embedding layers, we multiply those weights by $\sqrt{d_{\text{model}}}$.

Table 1: Maximum path lengths, per-layer complexity and minimum number of sequential operations for different layer types. n is the sequence length, d is the representation dimension, k is the kernel size of convolutions and r the size of the neighborhood in restricted self-attention.

Layer Type	Complexity per Layer	Sequential Operations	Maximum Path Length
Self-Attention	$O(n^2 \cdot d)$	$O(1)$	$O(1)$
Recurrent	$O(n \cdot d^2)$	$O(n)$	$O(n)$
Convolutional	$O(k \cdot n \cdot d^2)$	$O(1)$	$O(log_k(n))$
Self-Attention (restricted)	$O(r \cdot n \cdot d)$	$O(1)$	$O(n/r)$

▼ 해석본

✅ 3.4 임베딩(Embeddings)과 소프트맥스(Softmax)

다른 시퀀스 변환 모델들과 마찬가지로, 우리는 학습된 임베딩을 사용하여 입력 토큰과 출력 토큰을 **d_model** 차원의 벡터로 변환합니다. 또한, 디코더의 출력을 예측된 다음 토큰의 확률로 변환하기 위해 **학습된 선형 변환**과 **소프트맥스 함수**를 사용합니다.

우리 모델에서는 두 개의 임베딩 레이어와 소프트맥스 전에 사용되는 선형 변환에서 **동일한 가중치 행렬**을 공유하는데, 이는 [30]에서와 유사한 방식입니다. 임베딩 레이어에서는 이 가중치를 **√d_model**로 곱합니다.
(30번 논문 첨부):

Ofir Press and Lior Wolf. Using the output embedding to improve language models. arXivpreprint arXiv:1608.05859, 2016.

이 방식은 모델이 입력과 출력 토큰을 처리할 때 **효율성을 높이고** 계산을 단순하게 할 수 있도록 도와줍니다.

- 다른 sequence trasduction 모델과 유사하게, 학습된 embedding을 사용하여 input token과 output token을 d_model차원의 벡터로 변환합니다. 또한, 일반적으로 학습된 linear transformation과 softmax 함수를 사용하여 decoder output을 예측된 다음 token 확률로 변환합니다. 본 모델에서, 두 embedding layer와 softmax 이전 linear transformation 사이에서 동일한 weight matrix를 공유합니다. Inner layer에서 이러한 weight를 가중치에 √(d_model)을 곱합니다.

<Table1 해석>

- **n**: 시퀀스 길이
- **d**: 표현(특징) 차원
- **k**: 컨볼루션의 커널 크기
- **r**: 제한된 셀프 어텐션에서의 이웃 크기

레이어 유형	레이어 당 복잡도	순차 연산 수	최대 경로 길이
Self-Attention	$O(n^2 \cdot d)$	$O(1)$	$O(1)$
Recurrent(RNN)	$O(n \cdot d^2)$	$O(n)$	$O(n)$
Convolutional	$O(k \cdot n \cdot d^2)$	$O(1)$	$O(\log_k(n))$
Self-Attention (제한된)	$O(r \cdot n \cdot d)$	$O(1)$	$O(n/r)$

1. 레이어 당 복잡도

- **Self-Attention**: 복잡도는 $O(n^2 \cdot d)$로, 시퀀스 길이(n)의 제곱에 비례하고 특징 차원(d)에 비례합니다. 셀프 어텐션은 전체 시퀀스의 모든 위치가 상호작용하므로 복잡도가 높습니다.
- **Recurrent**: 복잡도는 $O(n \cdot d^2)$로, 시퀀스 길이에 비례하고 특징 차원의 제곱에 비례합니다. 순차적인 처리가 필요하기 때문에 복잡도가 큽니다.
- **Convolutional**: 복잡도는 $O(k \cdot n \cdot d^2)$로, 커널 크기(k)에 비례하며 시퀀스 길이와 특징 차원의 제곱에 비례합니다.
- **제한된 셀프 어텐션**: 복잡도는 $O(r \cdot n \cdot d)$로, 시퀀스 길이와 특징 차원에 비례하지만 제한된 범위(r) 안에서만 계산이 이루어져 일반적인 셀프 어텐션보다 복잡도가 낮습니다.

2. 순차 연산 수

- **Self-Attention**, **Convolutional**, **제한된 셀프 어텐션**은 순차 연산 수가 $O(1)$입니다. 즉, 병렬 처리가 가능하여 한 번의 계산으로 여러 위치를 동시에 처리할 수 있습니다.
- **Recurrent**는 $O(n)$으로, 순차적으로 한 위치씩 처리해야 하므로 병렬 처리가 어렵고 느립니다.

3. 최대 경로 길이

- **Self-Attention**은 $O(1)$로, 모든 위치가 한 번에 서로 상호작용할 수 있어 경로 길이가 짧습니다.
- **Recurrent**는 $O(n)$으로, 시퀀스의 처음부터 끝까지 순차적으로 연결되므로 경로가 깁니다.
- **Convolutional**은 $O(\log_k(n))$으로, 커널 크기에 따라 경로 길이가 감소합니다.
- **제한된 셀프 어텐션**은 $O(n/r)$로, 제한된 범위 내에서만 상호작용하므로 경로 길이가 줄어듭니다.

요약:

- **셀프 어텐션**은 병렬 처리에 유리하지만 복잡도가 높습니다.
- **Recurrent(기존 RNN방식)**는 순차 처리가 필요해 경로가 길고, 병렬화가 어려운 단점이 있습니다.
- **컨볼루션**은 커널 크기에 따라 경로 길이가 짧아지고, 병렬화가 가능하지만 복잡도는 중간 정도입니다.
- **제한된 셀프 어텐션**은 셀프 어텐션의 이점을 살리면서도 복잡도를 낮춘 방식입니다.

3.5 Positional Encoding

Since our model contains no recurrence and no convolution, in order for the model to make use of the order of the sequence, we must 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. The positional encodings have the same dimension d_{model} as the embeddings, so that the two can be summed. There are many choices of positional encodings, learned and fixed [9].

In this work, we use sine and cosine functions of different frequencies:

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

where pos is the position and i is the dimension. That is, each dimension of the positional encoding corresponds to a sinusoid. The wavelengths form a geometric progression from 2π to $10000 \cdot 2\pi$. We chose this function because we hypothesized it would allow the model to easily learn to attend by relative positions, since for any fixed offset k, PE_{pos+k} can be represented as a linear function of PE_{pos}.

We also experimented with using learned positional embeddings [9] instead, and found that the two versions produced nearly identical results (see Table 3 row (E)). We chose the sinusoidal version because it may allow the model to extrapolate to sequence lengths longer than the ones encountered during training.

- Transformer는 순차적인 특성이 없고 이에 따라 **sequence의 위치 정보가 없기 때문에 positional 정보를 추가**해줘야한다. 이를 위해, **encoder와 decoder input embedding에 "positional encoding'을 추가**한다. Positional encoding은 embedding과 동일한 차원을 가진다.
- 본 연구에서는 다음과 같이 sin, cos 함수를 사용한다.

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

여기서 *pos*는 token의 위치를 의미하고, i는 차원을 의미한다. 즉, positional encoding의 각 차원은 사인파에 해당한다. geometric progression wavelengths(**기하학적 진행 파장)** 은 2π에서 10000까지이다. $PE_(pos + k)$**가** $PE_(pos)$**의 선형 함수로 표현**될 수 있기에 모델이 상대적인 위치를 쉽게 학습할 수 있을 것이라 가정했기 때문에 이 함수를 사용한다.

- 학습가능한 positional embedding을 사용해 봤지만 동일한 성능을 보이는 것을 확인할 수 있었고, 더 긴 시퀀스에서도 추론이 가능한 사인파 버전을 사용한다.

→정리

- 이 사인과 코사인 함수는 주기가 다르게 설정되며, 이로 인해 서로 다른 위치에서 벡터의 값들이 **주기적으로 변**합니다. 이를 통해 모델이 **토큰의 위치 정보**를 학습할 수 있도록 도와줍니다.
- **d_model**은 임베딩 벡터의 차원을 나타내며, 차원이 커질수록 더 느린 주기를 가진 사인/코사인 값이 할당됩니다.

이 방식의 장점은 위치에 대한 정보를 벡터에 자연스럽게 포함시킬 수 있으며, 모델이 위치에 따른 규칙을 학습하게 만들어 줍니다.

---

4 Why Self-Attention

In this section we compare various aspects of self-attention layers to the recurrent and convolutional layers commonly used for mapping one variable-length sequence of symbol representations $(x_1, ..., x_n)$ to another sequence of equal length $(z_1, ..., z_n)$, with $x_i, z_i \in \mathbb{R}^d$, such as a hidden layer in a typical sequence transduction encoder or decoder. Motivating our use of self-attention we consider three desiderata.

One is the total computational complexity per layer. Another is the amount of computation that can be parallelized, as measured by the minimum number of sequential operations required.

The third is the path length between long-range dependencies in the network. Learning long-range dependencies is a key challenge in many sequence transduction tasks. One key factor affecting the ability to learn such dependencies is the length of the paths forward and backward signals have to traverse in the network. The shorter these paths between any combination of positions in the input and output sequences, the easier it is to learn long-range dependencies [12]. Hence we also compare the maximum path length between any two input and output positions in networks composed of the different layer types.

As noted in Table 1, a self-attention layer connects all positions with a constant number of sequentially executed operations, whereas a recurrent layer requires $O(n)$ sequential operations. In terms of computational complexity, self-attention layers are faster than recurrent layers when the sequence

length n is smaller than the representation dimensionality d, which is most often the case with sentence representations used by state-of-the-art models in machine translations, such as word-piece [38] and byte-pair [31] representations. To improve computational performance for tasks involving very long sequences, self-attention could be restricted to considering only a neighborhood of size r in the input sequence centered around the respective output position. This would increase the maximum path length to $O(n/r)$. We plan to investigate this approach further in future work.

A single convolutional layer with kernel width $k < n$ does not connect all pairs of input and output positions. Doing so requires a stack of $O(n/k)$ convolutional layers in the case of contiguous kernels, or $O(log_k(n))$ in the case of dilated convolutions [18], increasing the length of the longest paths between any two positions in the network. Convolutional layers are generally more expensive than recurrent layers, by a factor of k. Separable convolutions [6], however, decrease the complexity considerably, to $O(k \cdot n \cdot d + n \cdot d^2)$. Even with $k = n$, however, the complexity of a separable convolution is equal to the combination of a self-attention layer and a point-wise feed-forward layer, the approach we take in our model.

As side benefit, self-attention could yield more interpretable models. We inspect attention distributions from our models and present and discuss examples in the appendix. Not only do individual attention heads clearly learn to perform different tasks, many appear to exhibit behavior related to the syntactic and semantic structure of the sentences.

▼ 해석본

✅ 4. 셀프 어텐션(Self-Attention)

이 섹션에서는, $(x_1,...,x_n)$과 같은 하나의 가변 길이 시퀀스의 기호 표현을 $(z_1,...,z_n)$으로 변환하는 데 일반적으로 사용되는 순환 레이어 및 컨볼루션 레이어를 셀프 어텐션 레이어와 비교합니다. 여기서 $x_1, z_1 \in R^d$이며, 이는 전형적인 시퀀스 변환 인코더 또는 디코더의 숨겨진 레이어에서 발생합니다. 셀프 어텐션을 사용하는 동기를 설명하기 위해 우리는 세 가지 요구 사항을 고려합니다.

첫 번째는 레이어 당 총 계산 복잡도입니다. 두 번째는 병렬화할 수 있는 계산량으로, 필요한 최소 순차 연산 수로 측정됩니다. 세 번째는 네트워크에서 **장거리 의존성(long-range dependencies)** 간의 경로 길이입니다. 장거리 의존성을 학습하는 것은 많은 시퀀스 변환 작업에서 중요한 과제입니다. 이러한 의존성을 학습할 수 있는 능력에 영향을 미치는 주요 요인 중 하나는 네트워크에서 **전방 및 후방 신호**가 이동해야 하는 경로의 길이입니다. 입력과 출력 시퀀스의 모든 위치 간의 경로가 짧을수록, 장거리 의존성을 학습하기가 더 쉽습니다【12】. 따라서 우리는 서로 다른 레이어 유형으로 구성된 네트워크에서 입력과 출력 간의 최대 경로 길이를 비교합니다.

표 1에서 설명한 바와 같이, 셀프 어텐션 레이어는 일정한 수의 순차적으로 실행되는 연산으로 모든 위치를 연결하는 반면, 순환 레이어는 **O(n)**의 순차 연산을 요구합니다. 계산 복잡도의 측면에서, 시퀀스 길이 **n**이 표현 차원 **d**보다 작은 경우 셀프 어텐션 레이어는 순환 레이어보다 빠릅니다. 이는 최신 번역 모델에서 사용하는 **단어 조각(word-piece)【38】** 및 **바이트 쌍(byte-pair)【31】** 표현에서 자주 발생하는 현상입니다. 매우 긴 시퀀스가 포함된 작업에서 계산 성능을 개선하기 위해, 셀프 어텐션을 입력 시퀀스에서 해당 출력 위치를 중심으로 크기 **r**인 이웃만 고려하도록 제한할 수 있습니다. 이 경우 최대 경로 길이는 **O(n/r)**로 증가합니다. 우리는 앞으로 이 접근 방식을 더 연구할 계획입니다.

커널 너비가 **k < n**인 단일 컨볼루션 레이어는 모든 입력 및 출력 위치 쌍을 연결하지 않습니다. 이렇게 하려면 연속 커널의 경우 **O(n/k)** 개의 컨볼루션 레이어를 쌓아야 하고, 확장된 컨볼루션의 경우 **$O(\log_k(n))$** 개가 필요하며【18】, 이는 네트워크에서 모든 두 위치 간의 가장 긴 경로 길이를 증가시킵니다. 일반적으로 컨볼루션 레이어는 **k** 배 더 비쌉니다. 그러나 **분리형 컨볼루션(Separable Convolution)【6】**은 복잡도를 상당히 줄여 **$O(k \cdot n \cdot d + n \cdot d^2)$**로 만듭니다. 하지만 **k = n**일 때에도, 분리형 컨볼루션의 복잡도는 셀프 어텐션 레이어와 포인트-와이즈 피드-포워드 레이어를 결합한 복잡도와 동일합니다.

> 부수적인 이점으로, 셀프 어텐션은 **해석 가능성이 더 높은 모델**을 제공할 수 있습니다. 우리는 모델에서 어텐션 분포를 조사하여 부록에서 예제를 제시하고 논의합니다. 개별 어텐션 헤드가 명확히 서로 다른 작업을 학습할 뿐만 아니라, 많은 경우 문장의 구문 및 의미 구조와 관련된 행동을 보이는 것처럼 보입니다.

4. 왜 셀프 어텐션(Self-Attention)을 사용하는가

셀프 어텐션 레이어와 일반적으로 사용되는 **순환(Recurrent)** 및 **컨볼루션(Convolutional)** 레이어를 비교하여 셀프 어텐션의 장점을 설명합니다. 특히, 시퀀스 변환 작업에서 하나의 가변 길이 시퀀스를 동일한 길이의 다른 시퀀스로 매핑하는 문제를 다룹니다.

<셀프 어텐션을 사용하게 된 동기>

1. **레이어 당 총 계산 복잡도**:
   - 각 레이어가 데이터를 처리하는 데 얼마나 많은 계산이 필요한지를 평가합니다.
2. **병렬 처리 가능성**:
   - 병렬 처리 가능한 계산량을 평가하는데, 최소한으로 필요한 **순차적 연산 수**를 기준으로 병렬화 정도를 측정합니다.
3. **장거리 의존성(Path Length)**:
   - 시퀀스에서 서로 떨어져 있는 입력과 출력 간의 **경로 길이**를 평가합니다. 긴 시퀀스에서 **장거리 의존성**을 학습하는 것이 주요 도전 과제이며, 네트워크에서 신호가 전달되기 위해 거쳐야 하는 경로가 짧을수록, 장거리 의존성을 쉽게 학습할 수 있습니다【12】.

비교 결과:

1. **셀프 어텐션**은 모든 위치가 일정한 순차 연산 수로 연결되기 때문에 병렬 처리에 유리합니다. 반면, 순환 레이어(RNN)는 O(n)의 순차적 연산이 필요합니다.
2. 계산 복잡도 측면에서, 시퀀스 길이 **n**이 표현 차원 **d**보다 작을 때 셀프 어텐션 레이어는 순환 레이어보다 더 빠릅니다. 일반적으로 문장 표현에서는 시퀀스 길이가 표현 차원보다 짧기 때문에, 셀프 어텐션이 유리합니다.
3. 매우 긴 시퀀스를 다룰 때는, 성능 향상을 위해 셀프 어텐션을 **제한된 범위 내**에서만 동작하도록 할 수 있습니다. 이 경우 최대 경로 길이는 O(n/r)로 증가합니다. 이는 앞으로 추가 연구할 계획입니다.

4. **컨볼루션 레이어**의 경우, 커널 너비 **k**가 시퀀스 길이보다 작은 단일 컨볼루션 레이어는 모든 입력-출력 쌍을 연결할 수 없습니다. 이를 해결하려면 O(n/k)개의 컨볼루션 레이어를 쌓아야 하며, 커널을 확장한 **확장된 컨볼루션**을 사용하면 **$O(\log_k(n))$** 레이어가 필요합니다. 하지만 이 경우 경로 길이가 길어집니다.
    - 일반적인 컨볼루션 레이어는 순환 레이어보다 **비용이 더 많이 들지만**, 분리형 컨볼루션(Separable Convolution)을 사용하면 복잡도가 크게 줄어듭니다【6】.
5. **분리형 컨볼루션**의 복잡도는 $O(k \cdot n \cdot d + n \cdot d^2)$로, 셀프 어텐션 레이어와 피드-포워드 레이어의 복잡도와 비슷합니다. 하지만 **k = n**일 때도, 여전히 셀프 어텐션과 피드-포워드 네트워크를 조합한 모델의 복잡도와 같습니다.

- 셀프 어텐션은 **모델 해석 가능성**을 높일 수 있습니다. 어텐션 분포를 통해 모델이 각 단어 간의 관계를 어떻게 학습하는지 명확하게 볼 수 있으며, 개별 어텐션 헤드가 문장의 구문 및 의미 구조와 관련된 작업을 수행하는 것을 관찰할 수 있습니다.
- 셀프 어텐션 레이어는 병렬 처리가 용이하고 장거리 의존성 학습에 유리하며, 계산 복잡도에서도 효율적입니다. 따라서 순차적 연산이 필요한 순환 레이어나 경로 길이가 길어질 수 있는 컨볼루션 레이어보다 여러 면에서 더 나은 성능을 보일 수 있습니다. 추가로, 셀프 어텐션을 통해 모델이 더 쉽게 해석 가능하다는 장점도 있습니다.

5 Training

This section describes the training regime for our models.

5.1 Training Data and Batching

We trained on the standard WMT 2014 English-German dataset consisting of about 4.5 million sentence pairs. Sentences were encoded using byte-pair encoding [3], which has a shared source-target vocabulary of about 37000 tokens. For English-French, we used the significantly larger WMT 2014 English-French dataset consisting of 36M sentences and split tokens into a 32000 word-piece vocabulary [38]. Sentence pairs were batched together by approximate sequence length. Each training batch contained a set of sentence pairs containing approximately 25000 source tokens and 25000 target tokens.

5.2 Hardware and Schedule

We trained our models on one machine with 8 NVIDIA P100 GPUs. For our base models using the hyperparameters described throughout the paper, each training step took about 0.4 seconds. We trained the base models for a total of 100,000 steps or 12 hours. For our big models,(described on the bottom line of table 3), step time was 1.0 seconds. The big models were trained for 300,000 steps (3.5 days).

5.3 Optimizer

We used the Adam optimizer [20] with $\beta_1 = 0.9$, $\beta_2 = 0.98$ and $\epsilon = 10^{-9}$. We varied the learning rate over the course of training, according to the formula:

$$lrate = d_{\text{model}}^{-0.5} \cdot \min(step\_num^{-0.5}, step\_num \cdot warmup\_steps^{-1.5}) \tag{3}$$

This corresponds to increasing the learning rate linearly for the first $warmup\_steps$ training steps, and decreasing it thereafter proportionally to the inverse square root of the step number. We used $warmup\_steps = 4000$.

5.4 Regularization

We employ three types of regularization during training:

▼ 해석본

✅ 5. 훈련(Training)

이 섹션에서는 모델 훈련 방식에 대해 설명합니다.

5.1 훈련 데이터 및 배칭(Training Data and Batching)

모델은 **WMT 2014 영어-독일어 데이터셋**을 사용해 훈련되었으며, 이 데이터셋은 약 450만 개의 문장 쌍으로 구성되어 있습니다. 문장은 바이트 쌍 인코딩(byte-pair encoding)을 사용하여 인코딩되었으며, 약 37,000개의 공유된 소스 및 타겟 단어 집합을 가집니다【3】. 영어-프랑스어의 경우에는 더 큰 **WMT 2014 영어-프랑스어 데이터셋**을 사용하였으며, 3,600만 개의 문장으로 구성되어 있고, 32,000개의 단어 조각(word-piece) 어휘 집합으로 나누어졌습니다【38】. 문장 쌍들은 대략적인 시퀀스 길이에 따라 배치되었습니다. 각 훈련 배치에는 약 25만 개의 소스 토큰과 25만 개의 타겟 토큰을 포함한 문장 쌍이 포함되었습니다.

5.2 하드웨어 및 스케줄(Hardware and Schedule)

모델은 8개의 **NVIDIA P100 GPU**를 사용한 한 대의 기계에서 훈련되었습니다. 논문 전체에서 설명된 하이퍼파라미터를 사용하여, 각 훈련 단계는 약 **0.4초**가 걸렸습니다. 기본(base) 모델은 **100,000 단계**, 즉 약 **12시간** 동안 훈련되었습니다. 큰(big) 모델의 경우, **300,000 단계** 동안 훈련되었으며, 이때 각 단계는 **1.0초**가 걸려 총 **3.5일**이 소요되었습니다.

5.3 옵티마이저(Optimizer)

- *Adam 옵티마이저【20】**를 사용하였으며, 하이퍼파라미터는 $\beta_1 = 0.9$, $\beta_2 = 0.98$, $\varepsilon = 10^{-9}$로 설정되었습니다. 훈련 과정에서 학습률(learning rate)은 아래 공식에 따라 변화시켰습니다:
- (3) 이 공식은 **warmup_steps** 동안에는 학습률을 선형적으로 증가시키고, 이후에는 단계 수(step_num)가 증가함에 따라 학습률을 감소시키는 방식입니다. **warmup_steps**는 **4,000**으로 설정되었습니다.

Table 2: The Transformer achieves better BLEU scores than previous state-of-the-art models on the English-to-German and English-to-French newstest2014 tests at a fraction of the training cost.

Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [18]	23.75			
Deep-Att + PosUnk [39]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [38]	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S [9]	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE [32]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble [39]		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble [38]	26.30	41.16	$1.8 \cdot 10^{20}$	$1.1 \cdot 10^{21}$
ConvS2S Ensemble [9]	26.36	**41.29**	$7.7 \cdot 10^{19}$	$1.2 \cdot 10^{21}$
Transformer (base model)	27.3	38.1	**$3.3 \cdot 10^{18}$**	
Transformer (big)	**28.4**	**41.8**	$2.3 \cdot 10^{19}$	

Residual Dropout We apply dropout [33] to the output of each sub-layer, before it is added to the sub-layer input and normalized. In addition, we apply dropout to the sums of the embeddings and the positional encodings in both the encoder and decoder stacks. For the base model, we use a rate of $P_{drop} = 0.1$.

Label Smoothing During training, we employed label smoothing of value $\epsilon_{ls} = 0.1$ [36]. This hurts perplexity, as the model learns to be more unsure, but improves accuracy and BLEU score.

- **Transformer (big)** 모델은 EN-DE에서 BLEU 점수 28.4, EN-FR에서 41.8로 이전 모델들보다 높은 성능을 기록했습니다.
- **훈련 비용(FLOPs)** 측면에서도 Transformer는 다른 모델들보다 훨씬 적은 비용으로 훈련이 가능했습니다. 예를 들어, **ConvS2S**는 EN-DE에서 9.6 × 10^{19} FLOPs가 필요한 반면, **Transformer (base model)**은 3.3 × 10^{18} FLOPs로 훈련 비용이 훨씬 낮습니다.
- **Residual Dropout**: 서브레이어 출력에 드롭아웃을 적용한 후, 이를 입력에 더하고 정규화합니다. 임베딩과 포지셔널 인코딩 합에도 드롭아웃을 적용하며, 기본 모델에서는 드롭아웃 비율 **P_{eorp} = 0.1**을 사용합니다.
- **Label Smoothing**: 훈련 중 **레이블 스무딩(label smoothing)** 값을 **ε_{ls} = 0.1**로 적용했습니다. 이는 모델이 더 불확실한 예측을 하도록 학습시켜, 정확성과 BLEU 점수를 향상시킵니다.

트랜스포머는 BLEU 점수에서 다른 모델들보다 우수한 성능을 보여주며, 훈련 비용도 더 효율적입니다. 특히 **base model**은 훨씬 적은 계산량으로 더 나은 성능을 제공합니다.

6 Results

6.1 Machine Translation

On the WMT 2014 English-to-German translation task, the big transformer model (Transformer (big) in Table 2) outperforms the best previously reported models (including ensembles) by more than 2.0 BLEU, establishing a new state-of-the-art BLEU score of 28.4. The configuration of this model is listed in the bottom line of Table 3. Training took 3.5 days on 8 P100 GPUs. Even our base model surpasses all previously published models and ensembles, at a fraction of the training cost of any of the competitive models.

On the WMT 2014 English-to-French translation task, our big model achieves a BLEU score of 41.0, outperforming all of the previously published single models, at less than $1/4$ the training cost of the previous state-of-the-art model. The Transformer (big) model trained for English-to-French used dropout rate $P_{drop} = 0.1$, instead of 0.3.

For the base models, we used a single model obtained by averaging the last 5 checkpoints, which were written at 10-minute intervals. For the big models, we averaged the last 20 checkpoints. We used beam search with a beam size of 4 and length penalty $\alpha = 0.6$ [38]. These hyperparameters were chosen after experimentation on the development set. We set the maximum output length during inference to input length + 50, but terminate early when possible [38].

Table 2 summarizes our results and compares our translation quality and training costs to other model architectures from the literature. We estimate the number of floating point operations used to train a model by multiplying the training time, the number of GPUs used, and an estimate of the sustained single-precision floating-point capacity of each GPU [5].

6.2 Model Variations

To evaluate the importance of different components of the Transformer, we varied our base model in different ways, measuring the change in performance on English-to-German translation on the

Table 3: Variations on the Transformer architecture. Unlisted values are identical to those of the base model. All metrics are on the English-to-German translation development set, newstest2013. Listed perplexities are per-wordpiece, according to our byte-pair encoding, and should not be compared to per-word perplexities.

	N	d_{model}	d_{ff}	h	d_k	d_v	P_{drop}	ϵ_{ls}	train steps	PPL (dev)	BLEU (dev)	params $\times 10^6$
base	6	512	2048	8	64	64	0.1	0.1	100K	4.92	25.8	65
(A)				1	512	512				5.29	24.9	
				4	128	128				5.00	25.5	
				16	32	32				4.91	25.8	
				32	16	16				5.01	25.4	
(B)					16					5.16	25.1	58
					32					5.01	25.4	60
(C)	2									6.11	23.7	36
	4									5.19	25.3	50
	8									4.88	25.5	80
		256			32	32				5.75	24.5	28
		1024			128	128				4.66	26.0	168
			1024							5.12	25.4	53
			4096							4.75	26.2	90
(D)							0.0			5.77	24.6	
							0.2			4.95	25.5	
								0.0		4.67	25.3	
								0.2		5.47	25.7	
(E)	positional embedding instead of sinusoids									4.92	25.7	
big	6	1024	4096	16			0.3		300K	**4.33**	**26.4**	213

development set, newstest2013. We used beam search as described in the previous section, but no checkpoint averaging. We present these results in Table 3.

In Table 3 rows (A), we vary the number of attention heads and the attention key and value dimensions, keeping the amount of computation constant, as described in Section 3.2.2. While single-head attention is 0.9 BLEU worse than the best setting, quality also drops off with too many heads.

In Table 3 rows (B), we observe that reducing the attention key size d_k hurts model quality. This suggests that determining compatibility is not easy and that a more sophisticated compatibility function than dot product may be beneficial. We further observe in rows (C) and (D) that, as expected, bigger models are better, and dropout is very helpful in avoiding over-fitting. In row (E) we replace our sinusoidal positional encoding with learned positional embeddings [9], and observe nearly identical results to the base model.

Table 4: The Transformer generalizes well to English constituency parsing (Results are on Section 23 of WSJ)

Parser	Training	WSJ 23 F1
Vinyals & Kaiser el al. (2014) [37]	WSJ only, discriminative	88.3
Petrov et al. (2006) [29]	WSJ only, discriminative	90.4
Zhu et al. (2013) [40]	WSJ only, discriminative	90.4
Dyer et al. (2016) [8]	WSJ only, discriminative	91.7
Transformer (4 layers)	WSJ only, discriminative	91.3
Zhu et al. (2013) [40]	semi-supervised	91.3
Huang & Harper (2009) [14]	semi-supervised	91.3
McClosky et al. (2006) [26]	semi-supervised	92.1
Vinyals & Kaiser el al. (2014) [37]	semi-supervised	92.1
Transformer (4 layers)	semi-supervised	92.7
Luong et al. (2015) [23]	multi-task	93.0
Dyer et al. (2016) [8]	generative	93.3

increased the maximum output length to input length + 300. We used a beam size of 21 and $\alpha = 0.3$ for both WSJ only and the semi-supervised setting.

Our results in Table 4 show that despite the lack of task-specific tuning our model performs surprisingly well, yielding better results than all previously reported models with the exception of the Recurrent Neural Network Grammar [8].

In contrast to RNN sequence-to-sequence models [37], the Transformer outperforms the Berkeley-Parser [29] even when training only on the WSJ training set of 40K sentences.

▼ 해석본

✅ 6. 결과

6.1 기계 번역

WMT 2014 영어-독일어 번역 작업에서, 트랜스포머 big 모델(Table 2의 Transformer (big))은 이전에 보고된 최고 성능 모델들(앙상블 포함)을 BLEU 점수에서 2.0 이상 능가하며, 28.4의 새로운 최고 BLEU 점수를 기록했습니다. 이 모델의 구성은 Table 3의 하단에 나와 있습니다. 훈련은 8개의 P100 GPU에서 3.5일 동안 진행되었습니다. 심지어 기본 모델(base model)조차도 이전에 발표된 모든 모델 및 앙상블을 능가하면서, 경쟁 모델 중 가장 적은 훈련 비용만으로도 이 성과를 달성했습니다.

WMT 2014 영어-프랑스어 번역 작업에서, 트랜스포머 big 모델은 BLEU 점수 41.0을 기록했으며, 이는 이전에 발표된 단일 모델 모두를 능가하는 성과입니다. 트랜스포머 big 모델은 이전 최고 성능 모델의 훈련 비용의 1/4 미만으로 훈련되었습니다. 영어-프랑스어 작업에서 트랜스포머 big 모델은 드롭아웃 비율 **P_drop = 0.1**을 사용했으며, 기존 모델에서는 0.3을 사용했습니다.

기본 모델(base model)에서는 10분 간격으로 기록된 마지막 5개의 체크포인트를 평균한 단일 모델을 사용했습니다. big 모델의 경우 마지막 20개의 체크포인트를 평균했습니다. 우리는 빔 서치에서 **빔 크기 4**와 **길이 패널티 α = 0.6**을 사용했습니다【38】. 이러한 하이퍼파라미터는 개발 셋에서의 실험을 통해 선택되었습니다. 우리는 입력 길이에 50을 더한 만큼 최대 출력 길이를 설정했고, 가능한 경우 조기 종료를 적용했습니다【38】.

Table 2는 우리의 결과를 요약하고, 문헌에 있는 다른 모델들과의 번역 품질과 훈련 비용을 비교합니다. 우리는 훈련 시간을 모델의 FLOPs 수로 나눈 후, 사용한 GPU의 수 및 각 GPU의 단정밀도 부동소수점 연산량을 바탕으로 FLOPs 수를 추정했습니다.

<Table3>

- **기본 모델(base)**: 6개의 레이어, 512 차원, 피드-포워드 차원 2048, 8개의 어텐션 헤드, 각 어텐션 헤드의 크기 64, 드롭아웃 0.1, 레이블 스무딩 0.1, 100K 학습 단계에서 BLEU 점수 25.8, 파라미터 수 65M.
- **변형 (A)**: 어텐션 헤드 수(h)와 어텐션 크기(d_k, d_v)를 다양하게 변경. 단일 헤드 설정은 성능이 가장 낮고, 최적의 설정에서는 기본 모델과 유사한 성능을 보임.

- **변형 (B)**: 어텐션 크기를 줄임으로써 성능이 떨어짐. 이는 호환성을 결정하는 것이 쉽지 않으며, 더 정교한 호환성 함수가 필요함을 시사.
- **변형 (C)**: 더 큰 모델이 더 좋은 성능을 보이며, 드롭아웃이 과적합을 방지하는 데 매우 유용함.
- **변형 (D)**: 드롭아웃 비율을 다양하게 설정한 실험. 드롭아웃 비율이 너무 낮거나 높을 때 성능이 떨어짐.
- **변형 (E)**: 기존의 사인 곡선 기반 포지셔널 인코딩 대신 학습된 포지셔널 임베딩을 사용한 결과, 성능이 거의 동일함.
- **big 모델**: 더 큰 차원(1024, 4096)을 가진 트랜스포머 big 모델은 300K 학습 단계에서 BLEU 점수 26.4를 기록하며, 기본 모델보다 더 나은 성능을 보임.

해석:

- Table 3의 (A)에서 어텐션 헤드 수와 어텐션 키, 값의 차원을 변화시키며 계산량은 동일하게 유지했습니다. 단일 어텐션 헤드의 성능은 최적의 설정보다 BLEU 점수가 0.9 낮았고, 너무 많은 헤드를 사용하면 품질이 저하되었습니다.
- (B)에서 어텐션 키 크기를 줄이는 것은 모델 품질을 악화시켰습니다. 이는 호환성을 결정하는 것이 쉽지 않으며, 더 정교한 호환성 함수가 필요하다는 것을 시사합니다.
- (C)와 (D)에서는 더 큰 모델이 더 나은 성능을 보였으며, 드롭아웃은 과적합을 방지하는 데 매우 유용함을 알 수 있었습니다.

트랜스포머 모델은 출력 길이를 **입력 길이 + 300**으로 설정하여 출력의 최대 길이를 늘렸습니다. **빔 크기 21**과 **길이 패널티 α = 0.3**을 WSJ만 사용한 경우와 반지도 학습(semi-supervised) 설정 모두에서 사용했습니다.

표 4에서의 결과는 특정 작업에 맞춘 튜닝이 없었음에도 불구하고 트랜스포머 모델이 놀랍도록 좋은 성능을 보여주며, 거의 모든 이전에 보고된 모델보다 더 나은 결과를 제공합니다. 단, **Recurrent Neural Network Grammar** 모델을 제외한 모든 모델보다 우수합니다【8】.

RNN 기반 시퀀스-투-시퀀스 모델과 대조적으로, 트랜스포머는 **Berkeley Parser**보다 더 나은 성능을 보였습니다【29】. 이는 WSJ 훈련 셋에서 40,000개의 문장으로만 훈련했음에도 불구하고 성취한 결과입니다.

7 Conclusion

In this work, we presented the Transformer, the first sequence transduction model based entirely on attention, replacing the recurrent layers most commonly used in encoder-decoder architectures with multi-headed self-attention.

For translation tasks, the Transformer can be trained significantly faster than architectures based on recurrent or convolutional layers. On both WMT 2014 English-to-German and WMT 2014 English-to-French translation tasks, we achieve a new state of the art. In the former task our best model outperforms even all previously reported ensembles.

We are excited about the future of attention-based models and plan to apply them to other tasks. We plan to extend the Transformer to problems involving input and output modalities other than text and to investigate local, restricted attention mechanisms to efficiently handle large inputs and outputs such as images, audio and video. Making generation less sequential is another research goals of ours.

The code we used to train and evaluate our models is available at `https://github.com/tensorflow/tensor2tensor`.

Acknowledgements We are grateful to Nal Kalchbrenner and Stephan Gouws for their fruitful comments, corrections and inspiration.

▼ 해석본

✅

7. 결론 (Conclusion)

이 연구에서, 우리는 트랜스포머를 소개했습니다. 트랜스포머는 오직 어텐션에만 기반한 첫 번째 시퀀스 변환 모델로, 기존 인코더-디코더 구조에서 가장 흔하게 사용되던 순환 레이어를 멀티-헤드 셀프 어텐션으로 대체한 모델입니다.

번역 작업에서, 트랜스포머는 순환 또는 컨볼루션 레이어 기반 아키텍처보다 훨씬 빠르게 훈련될 수 있습니다. **WMT 2014 영어-독일어**와 **WMT 2014 영어-프랑스어 번역 작업**에서, 우리는 새로운 최고 성능을 달성했으며, 영어-독일어 작업에서는 이전에 보고된 모든 앙상블 모델을 능가하는 성과를 보였습니다.

우리는 어텐션 기반 모델의 미래에 대해 기대하고 있으며, 이를 다른 작업에도 적용할 계획입니다. 또한 텍스트 외의 입력 및 출력 모달리티를 다루는 문제로 트랜스포머를 확장하고, 이미지, 오디오, 비디오와 같은 대형 입력과 출력을 효율적으로 처리할 수 있도록 **제한된 로컬 어텐션 메커니즘**을 연구할 계획입니다. 생성 작업을 더 비순차적(non-sequential)으로 만드는 것도 우리의 또 다른 연구 목표입니다.