All You Need Is Attention

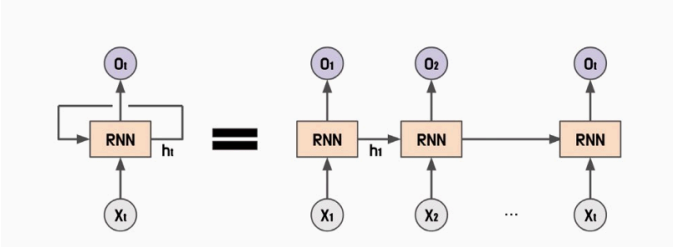
순환 신경망 (RNN)

연속적인 데이터 (Sequence data) : 각 시점 (time step) 의 데이터가 이전 시점의 데이터와 독립적이지 않음

- 자연어 또한 연속적인 데이터의 일종

순환 신경망 (RNN)

- 연속형 데이터를 순서대로 입력받아 처리하며 각 시점마다 은닉 상태 (Hidden State) 의 형태로 저장
- 셀 (Cell) : 각 시점의 데이터를 입력으로 받아 은닉 상태와 출력값을 계산하는 노드



- 은닉 상태 (Hidden State) :

수식 6.11 순환 신경망의 은닉 상태

$$h_t = \sigma_h(h_{t-1}, x_t)$$
$$h_t = \sigma_h(W_{hh}h_{t-1} + W_{xh}x_t + b_h)$$

- 출력값

수식 6.12 순환 신경망의 출력값

$$y_t = \sigma_y(h_t)$$
$$y_t = \sigma_y(W_{hy}h_t + b_y)$$

일대다 구조(One-to-Many)

하나의 입력 시퀀스에 대해 여러 개의 출력값을 생성하는 출력 신경망 구조

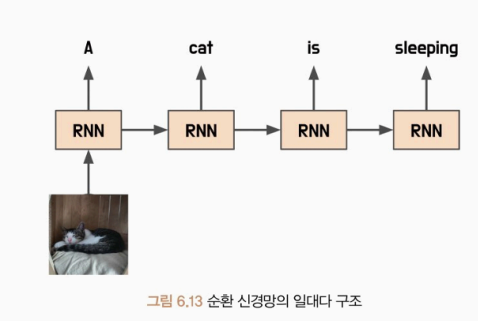


그림 6.13 순환 신경망의 일대다 구조

Ex) 이미지 캡셔닝 모델

일대다 구조를 구현하기 위해서는 출력 시퀀스의 길이를 미리 알고 있어야 하는데, 이를 위해 입력 시퀀스를 처리하면서 시퀀스의 길이를 예측하는 모델을 함께 구현해야 함.

다대일 구조 (Many-to-One)

여러 개의 입력 시퀀스에 대해 하나의 출력값을 생성하는 순환 신경망

- 감성 분류 (Sentiment Analysis)
  - 입력 시퀀스 (문장) → 출력값 (해당 문장의 감정)
- 자연어 추론 (Natural Language Inference)

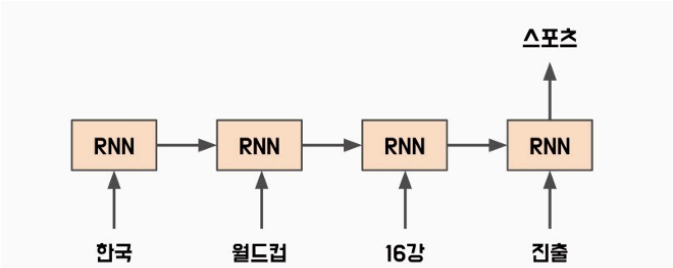


그림 6.14 순환 신경망의 다대일 구조

다대다 구조 (Many to Many)

입력 시퀀스와 출력 시퀀스의 길이가 여러 개인 경우 사용

- 입력 시퀀스와 출력 시퀀스의 길이가 서로 다른 경우 패딩을 추가하거나 잘라내는 전처리 과정이 필요
- Seq2Seq 구조 (인코더-디코더 구조)

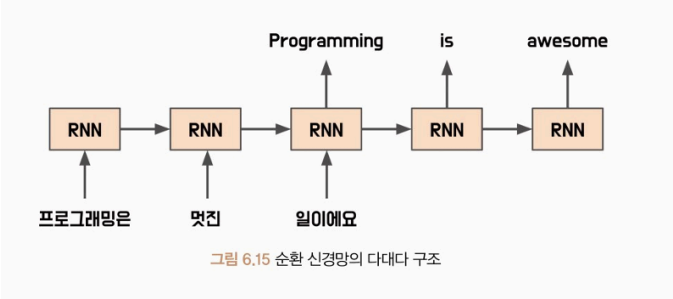


그림 6.15 순환 신경망의 다대다 구조

양방향 순환 신경망 (BiRNN)

기본적인 순환 신경망에서 시간 방향을 양방향으로 처리할 수 있도록 고안

- 이전 시점 (t-1) 과 이후 시점 (t+1) 의 은닉 상태도 함께 이용

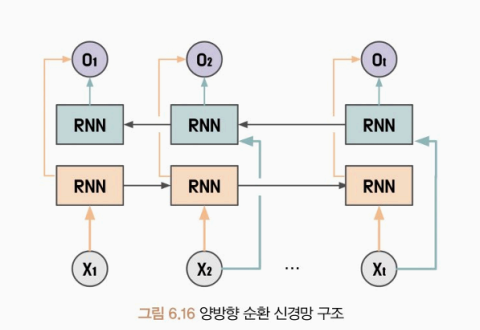


그림 6.16 양방향 순환 신경망 구조

다중 순환 신경망 (Stacked Recurrent Neural Network)

여러 개의 순환 신경망을 연결하여 구성한 모델로 각 순환 신경망이 서로 다른 정보를 처리하도록 설계

- 다층 퍼셉트론에서 퍼셉트론을 여러개 쌓는 것처럼 순환 신경망 여러 층을 쌓아 사용 → 더 복잡한 패턴을 학습 가능하지만 학습 시간이 오래 걸리고 기울기 소실 문제 발생

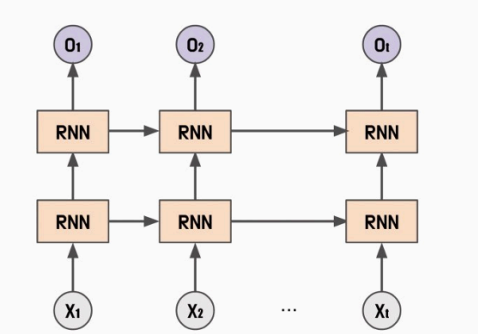


그림 6.17 다중 순환 신경망 구조

순환 신경망 클래스

- 입력값 (input) 과 초기 입력 상태 (h_0) 으로 순방향 연산을 수행
- 출력값 (output) 과 최종 은닉 상태(hiddes) 반환
- 최종 은닉 상태는 초기 은닉 상태와 동일한 차원

장단기 메모리 (LSTM)

- 장기 의존성 문제 : 어떤 문제에서는 다음 시점의 데이터를 유추하기 위해 훨씬 먼 시점의 데이터를 참고해야 하는데, 기존 모델에서는 불가능
- 활성화 함수로 tanh 나 ReLU 가 사용된느데, 역전파 과정에서 기울기 소실/폭주가 발생 가능

⇒ 메모리 셀(Memory Cell) 과 게이트 구조로 문제 해결

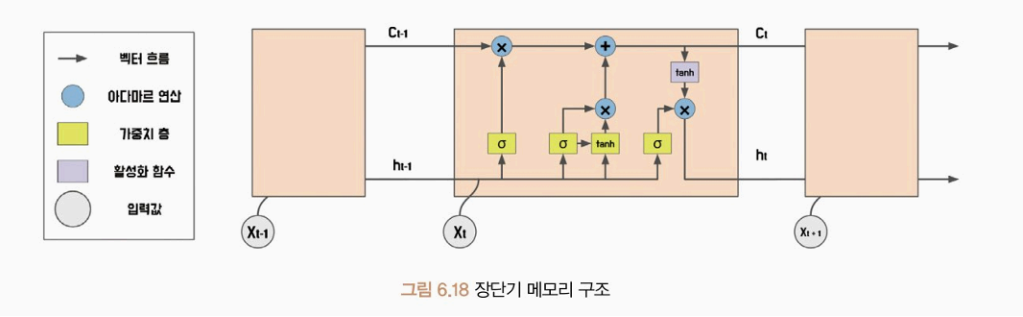


그림 6.18 장단기 메모리 구조

- 셀 상태 - 정보를 저장하고 유지하는 메모리 역할

수식 6.15 메모리 셀

$$c_t = f_t \odot c_{t-1} + g_i \odot i_t$$

C_t-1 : 이전 시점의 메모리 셀 값

- 망각 게이트 - 이전 셀 상태에서 어떠한 정보를 삭제할지 결정. 현재 입력과 이전 셀 상태를 입력으로 받아 어떤 정보를 삭제할지 결정

  → 결정된 대로 메모리 셀 업데이트

수식 6.13 망각 게이트

$$f_t = \sigma(W_x^{(f)} x_t + W_h^{(f)} h_{t-1} + b^{(f)})$$

Wx/h(입력값과 은닉 상태를 위한 가중치), b^(f) (망각 게이트의 편향)
→ x_t 와 h_t-1 을 이용해 출력값 계산

- 입력/기억 게이트 - 새로운 정보를 어떤 부분에 추가할지 결정. 현재 입력과 이전 셀 상태를 입력으로 받아 어떤 정보를 추가할지 결정 → 어디다 추가한다는거지

  → 시그모이드 결과(입력값의 중요도) X Tanh 결과 (변환된 입력값) = 새로운 기억 값 - 이 점수에 따라 어떤 정보를 얼마나 추가할지 결정

수식 6.14 기억 게이트

$$g_i = tanh(W_x^{(g)} x_t + W_x^{(g)} h_{t-1} + b^{(g)})$$
$$i_i = sigmoid(W_x^{(i)} x_t + W_x^{(i)} h_{t-1} + b^{(i)})$$

- 출력 게이트 - 셀 상태의 정보 중 어떤 부분을 출력할지 결정. 현재 입력과 이전 셀 상태, 새로 추가된 정보를 입력으로 받아 출력할 정보 저장

수식 6.16 출력 게이트

$$o_t = \sigma(W_x^{(o)} x_t + W_h^{(o)} h_{t-1} + b^{(o)})$$

- 은닉 상태 갱신

수식 6.17 은닉 상태 갱신

$$h_t = o_t \odot tanh(c_t)$$

장단기 메모리 구조

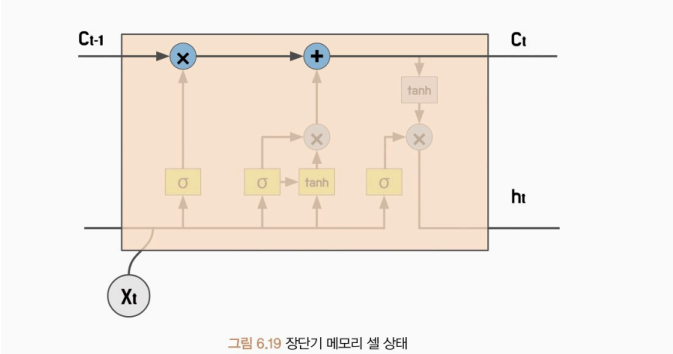


그림 6.19 장단기 메모리 셀 상태

- 장단기 메모리의 메모리 셀은 출력값 계산에 직접 사용되지는 않음
- 세 가지 게이트는 활성화 함수로 시그모이드 사용
- 입력값에 대한 가중치를 동적으로 조절하면서 적절한 정보만을 기억하고 유지할 수 있게 함

트랜스포머

어텐션 매커니즘만을 사용하여 시퀀스 임베딩 표현

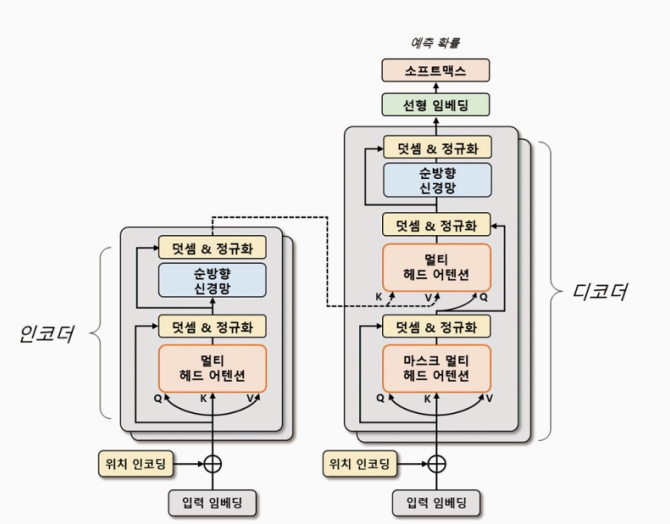


그림 7.2 트랜스포머 모델 구조(Q: 쿼리, K: 키, V: 값)

- 트랜스포머 블록 (Transformer Block) 으로 구성
  - 멀티 헤드 어텐션 (Multi-Head Attention) 과 순방향 신경망으로 이루어져 있음
  - 멀티 헤드 어텐션

    입력 시퀀스에서 쿼리, 키, 값 벡터를 정의해 입력 시퀀스들의 관계를 셀프 어텐션 하는 벡터 표현 방법

    → 쿼리와 각 키의 유사도를 계산하고 해당 유사도를 가중치로 사용하여 값 벡터 합산

    → 어텐션 행렬은 입력 시퀀스 각 단어의 임베딩 벡터를 대체
  - 입력 시퀀스 데이터 = 소스 + 타깃

    ex) 영어 → 한글 번역 : 소스(영어), 타깃(한글)
  - 인코더 : 소스 시퀀스 데이터를 위치 인코딩 (Positional Encoding) 된 입력 임베딩으로 표현
  - 디코더 : 인코더와 유사하게 트랜스포머 블록으로 구성되어 있지만 마스크 멀티 헤드 어텐션을 사용해 타깃 시퀀스 데이터를 순차적으로 생성
    - 마스크 멀티 헤드 어텐션을 사용해 타깃 시퀀스 데이터 순차 생성
- 입력 임베딩
- 위치 인코딩

  트랜스포머 모델은 순환 신경망과 달리 입력 시퀀스를 병렬 구조로 처리해 단어의 순서 정보를 제공하지 않는데, 위치 정보를 임베딩 벡터에 추가해 단어의 순서 정보를 모델에 반영

하기 위해 사용

임베딩 벡터에 각 단어의 위치 정보를 나타내는 **위치 임베딩 벡터**를 더함

- 위치 인코딩 벡터

  sin 과 cos 함수를 사용해 생성, 임베딩 벡터와 위치 정보가 결합된 최종 입력 벡터를 생성

  각 토큰의 위치를 각도로 표현해 sin 함수와 cos 함수로 위치 인코딩 벡터 계산

특수 토큰

입력 시퀀스의 시작과 끝을 나타내거나 마스킹 영역으로 사용

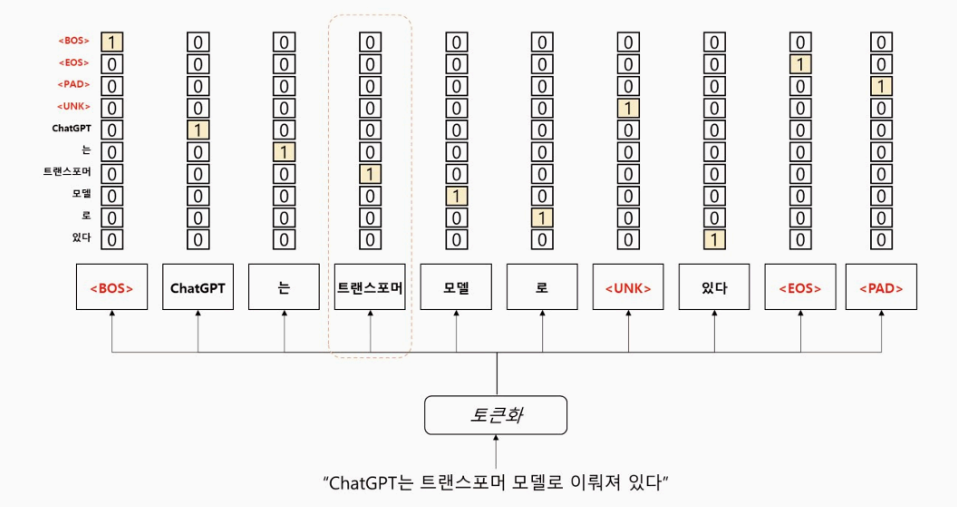


그림 7.3 특수 토큰을 활용한 문장 토큰 배열

- `BOS` : 문장의 시작
- `EOS` : 문장의 종료
- `UNK` : 어휘 사전에 없는 단어
- `PAD` : 모든 문장을 일정한 길이로 맞추기 위해 사용

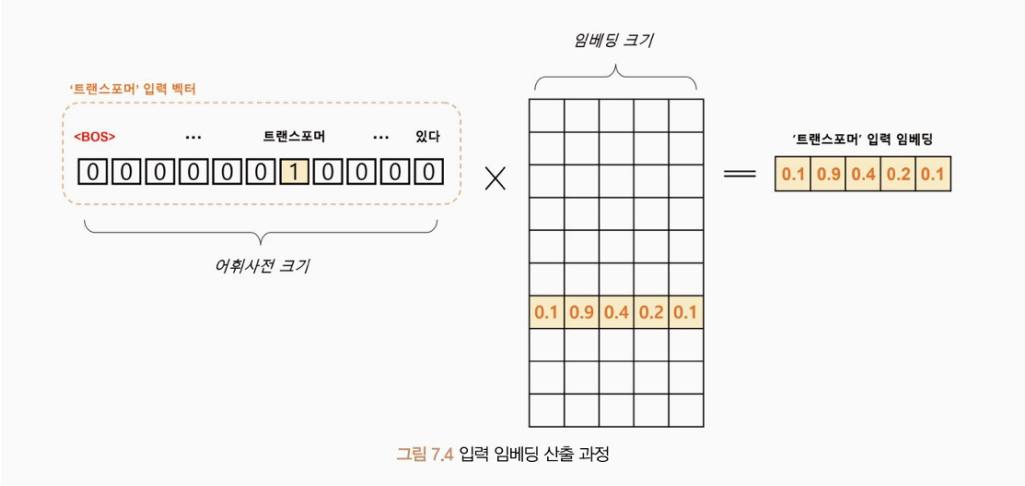


그림 7.4 입력 임베딩 산출 과정

→ 문장 토큰 배열을 어휘 사전에 등장하는 위치에 원-핫 인코딩으로 표현

트랜스포머 인코더

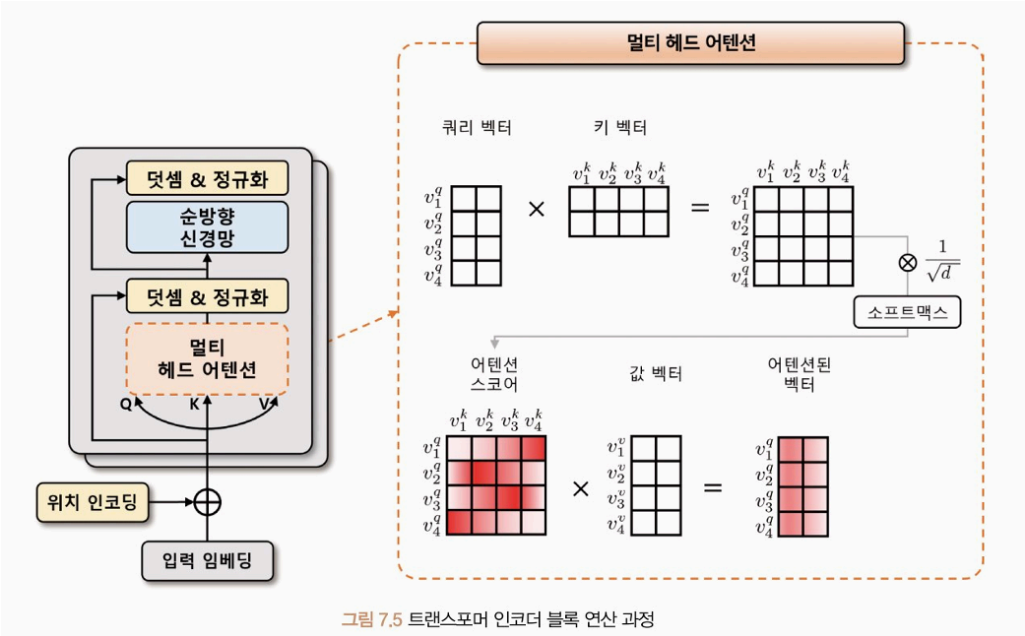


그림 7.5 트랜스포머 인코더 블록 연산 과정

- 인코더 계층
  - 멀티 헤드 어텐션

    [N, S, d] 차원의 입력 텐서

    → 쿼리(Q), 키(K), 값(V) 벡터 생성

- [N, S, S] 어텐션 스코어 맵 : 각 토큰이 문장 내 모든 토큰을 얼마나 참고하는지

수식 7.2 어텐션 스코어 계산식

$$score(v^q, v^k) = \text{softmax}\left(\frac{(v^q)^T \cdot v^k}{\sqrt{d}}\right)$$

셀프 어텐션이면 [N, S, S]

크로스 어텐션 (키와 쿼리의 길이 다름) 이면 [N, S_q, S_k]

멀티헤드면 [N, H, S, S]

- 어텐션 스코어 맵 A X 값 벡터 = 셀프 **어텐션된 벡터 [N, S, d]**
- 멀티 헤드 어텐션의 어텐션 스코어 맵 [N, k, S, d/k] X 값 벡터 = [N, k, S, d/k]

  → 임베딩 축 방향으로 concatenation

  → **어텐션된 벡터 [N, S, d]**

  - 순방향 신경망
    - 선형 임베딩 + ReLU 또는 1차원 합성곱
  - 덧셈과 정규화

    멀티 헤드 어텐션을 통과하기 전의 입력값 [N, S, d] 텐서와 통과한 이후의 출력값 [N, S, d] 텐서를 더함으로써 기울기 소실 완화

    임베딩 차원 축으로 정규화하는 계층 정규화 적용

- 입력 벡터 + 위치 임베딩 벡터

트랜스포머 디코더

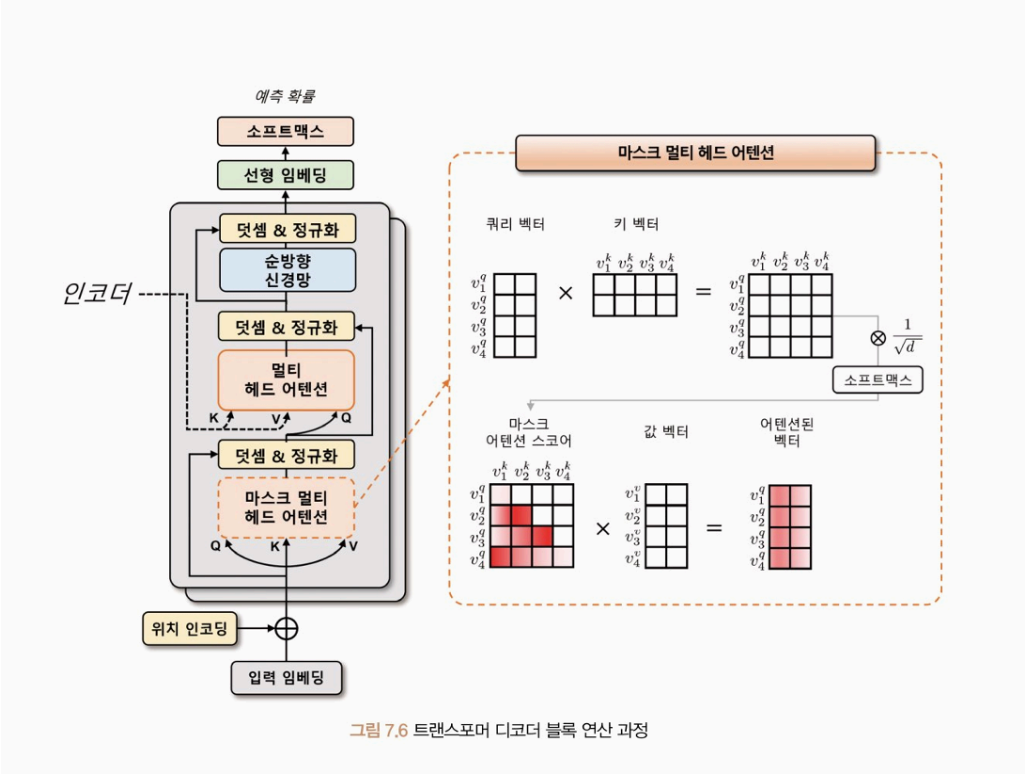
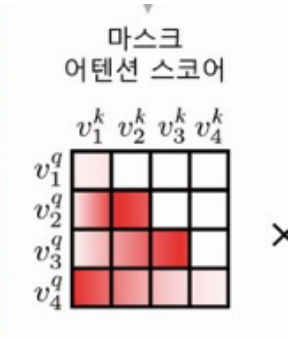


그림 7.6 트랜스포머 디코더 블록 연산 과정

- 타깃 데이터의 입력 임베딩 + 위치 인코딩
- **마스크 멀티 헤드 어텐션** : 어텐션 스코어 맵을 계산할 때 쿼리 벡터가 본인 순서 이하의 키 벡터만을 바라보게 함



- 여러 개의 트랜스포머 디코더 블록을 통과해 출력된 텐서 [N, S, d]

  → 선형 변환/소프트맥스

  → 타깃 시퀀스 위치마다 예측 확률 계산

- 타깃 데이터를 추론할 때 토큰 또는 단어를 순차적으로 생성

  → PAD 토큰 사용

표 7.2 인과성을 고려한 디코더 입력과 출력 예시

추론 순서	디코더 입력	디코더 출력
1	[BOS], [PAD], [PAD], [PAD], [PAD], [PAD], [PAD], [PAD], [PAD]	[BOS], 'ChatGPT', [PAD], [PAD], [PAD], [PAD], [PAD], [PAD], [PAD]
2	[BOS], 'ChatGPT', [PAD], [PAD], [PAD], [PAD], [PAD], [PAD], [PAD]	[BOS], 'ChatGPT', '는', [PAD], [PAD], [PAD], [PAD], [PAD], [PAD]
3	[BOS], 'ChatGPT', '는', [PAD], [PAD], [PAD], [PAD], [PAD], [PAD]	[BOS], 'ChatGPT', '는', '트랜스포머', [PAD], [PAD], [PAD], [PAD], [PAD]
4	[BOS], 'ChatGPT', '는', '트랜스포머', [PAD], [PAD], [PAD], [PAD], [PAD]	[BOS], 'ChatGPT', '는', '트랜스포머', '모델로', [PAD], [PAD], [PAD], [PAD]
5	[BOS], 'ChatGPT', '는', '트랜스포머', '모델로', [PAD], [PAD], [PAD], [PAD]	[BOS], 'ChatGPT', '는', '트랜스포머', '모델로', '이뤄져', [PAD], [PAD], [PAD]
6	[BOS], 'ChatGPT', '는', '트랜스포머', '모델로', '이뤄져', [PAD], [PAD], [PAD]	[BOS], 'ChatGPT', '는', '트랜스포머', '모델로', '이뤄져', '있다', [PAD], [PAD]
7	[BOS], 'ChatGPT', '는', '트랜스포머', '모델로', '이뤄져', '있다', [PAD], [PAD]	[BOS], 'ChatGPT', '는', '트랜스포머', '모델로', '이뤄져', '있다', [EOS], [PAD]

Attention Is All You Need

Abstract

기존 dominant sequence transdution 모델들이 인코더와 디코더를 포함하는 RNN 이나 CNN 에 기반. 그중에서도 가장 성능이 좋은 모델들은 인코더와 디코더를 attention 매커니즘을 통해 연결함.

- 새로운 네트워크 아키텍처 Transformer 제시
  - Transformer 는 순환, 합성곱 없이 attention 매커니즘에만 기반함
  - 두 가지 machine traslation 과제 실험에서 이 모델이 품질 면에서 우수하고, 더 parallelizable 하며, 학습에 훨씬 적은 시간이 필요하다는 것을 보여줌
    - WMT 2014 영어-독일어 번역 과제에서 28.4 BLEU 를 달성해 기존 결과보다 2BLEU 이상 향상
    - WMT 2014 영어-프랑스어 번역 과제에서는 새로운 단일모델 state-of-the-art BLEU 를 달성함.

    - 대규모 데이터와 제한된 데이터 모두에서 영어 구문 분석에 성공적으로 적용됨으써 다른 과에도 잘 일반화됨을 보여줌

1. Introduction

RNN, LSTM, gated recurrent neural networks 등은 sequence modeling, transduction 문제들, 특히 언어 모델링이나 기계 번역과 같은 문제에서 state of the art 한 접근 방식으로 확립되어 있음.

- Recurrent Model 의 특성
  입력과 출력 시퀀스의 심볼 위치에 따라 연산을 분해. 이 위치들을 연산 시간 안의 단계들에 정렬하여 이전 hidden state h_t-1 와 위치 t 의 입력의 함수로 하는 hidden state h_t 의 시퀀스를 형성
    - 순차적인 특성 때문에 병렬화 불가능
      -> 시퀀스 길이가 길어짐에 따라 메모리 제약이 batching 을 제한함.
    - 최근 연구의 factorization trick, conditional computation 등의 방법을 통해 개선을 이룸
- Attention 매커니즘
  입력이나 출력 시퀀스에서의 거리와 상관없이 modeling of dependencies 를 가능하게 함
  하지만 많은 경우 Recurrent Model 과 함께 사용됨
- Transformer 제안
  순환을 배제하고 입력과 출력 사이의 global dependencies 를 학습하기 위해 attention 매커니즘만 사용하는 모델 아키텍처
  훨씬 더 높은 병렬화를 가능하게 함.

2. Background

- 기존 합성곱 신경망의 문제점
    - Sequential computation을 줄이기 위해 개발된 Extended Neural GPU, ByteNet, ConvS2S 등 모델
    - 이러한 모델들에서는 **두 임의의 입력 또는 출력 위치 간의 신호를 연결히기 위해 필요한 연산의 수**가 포지션 간의 거리에 따라 증가.

ex) ConvS2S - 선형적으로, ByteNet - 로그로 증가
-> 멀리 떨어진 위치 간의 종속성(dependencies) 를 학습하는 것을 더 어렵게 함

- Transformer
  - **두 임의의 입력 또는 출력 위치 간의 신호를 연결하기 위해 필요한 연산의 수** -> 일정한 수의 연산으로 줄어들음.
  - 하지만 Transformer 가 sequence-aligned RNN 또는 합성곱을 사용하지 않고 오직 self attention 만 사용하는 첫 번째 변환 모델
- Self Attention(=intra attention)
  단일 시퀀스 내의 서로 다른 위치를 연결하여 그 시퀀스의 표현을 계산하는 Attention 매커니즘
- End to end memory network: sequence-alined recurrence 대신 recurrent attention mechanism 에 기반, 단순 언어 과제와 모델링 과제에서 좋은 성능을 보임

3. Model Architecture

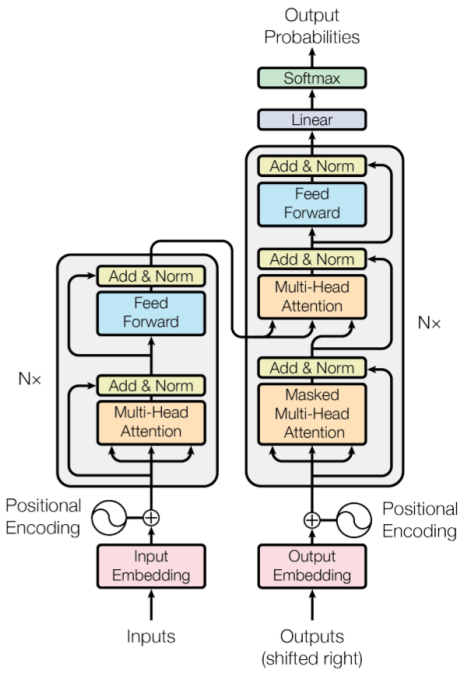


Figure 1: The Transformer - model architecture.

- encoder : input sequence (x1, ... xn) → continuous representations z = (z1, ... zn)
- decoder : z → output sequence (y1, ... , yn)

3.1 Encoder and Decoder Stacks

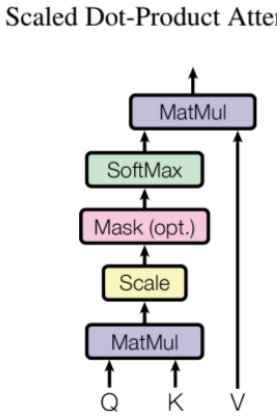
- Encoder :
  - N = 6
  - Sublayer 1 - 멀티 헤드 어텐션 + Add&Norm
  - Sublayer 2 - 순방향 네트워크 + Add&Norm
  - 각 서브레이어에 residual connection 이 적용

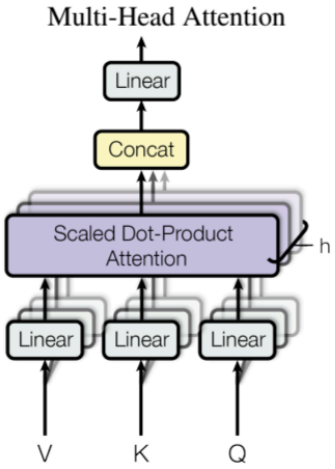
    → 각 서브레이어의 출력 : LayerNorm (x + Sublayer(x))

    모든 서브레이어는 d_model = 512 의 출력 생성
- Decoder :
  - N = 6
  - Sublayer 1 - Masked 멀티 헤드 어텐션 + Add&Norm
  - Sublayer 2 - 멀티 헤드 어텐션 + Add&Norm
  - Sublayer 3 - 순방향 네트워크 + Add&Norm

3.2 Attention

- Attention : mapping a query and a set of key-value pairs to an output, where the query, keys, and output are all vectors





1. Scaled Dot-Product Attention
2. Multi-Head Attention

   linearly project the queries, keys and values h times(헤드 수) with different, learned linear projectionsd to d_k, d_k, and d_v dimentions.

   → `d_v` 차원 output value

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

- head_i =

$$\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V) = \text{softmax}\left(\frac{Q_i K_i^\top}{\sqrt{d_k}}\right)V_i$$

  가중합 결과의 차원은 d_v 이므로 각 head 의 출력은 S X d_v

- Concat(~)W^O

  Concat 결과 : `S X (h·dv)`

  ✅

$$W^O \in \mathbb{R}^{(hd_v)\times d_\text{model}}$$

  → 에서, 최종 출력의 차원은 `S x d_model`

우리의 실험에서 h=8, d_k = d_v = d_model(512)/8 = 64 를 사용할 때 총 계산량은 single-head attentions 을 사용했을 때와 비슷하다.

3.2.3 Applications of Attention in our Model

트랜스포머가 멀티 헤드 어텐션을 사용하는 세 가지 방식

- encoder - decoder attention 층 : 쿼리는 이전 디코더 층에서 나오고, 메모리 키와 값은 인코더의 출력에서 오므로 디코더의 모든 포지션이 입력 시퀀스의 모든 포지션에 접근할 수 있게 함.
- encoder 의 self-attention 층 : 키, 값, 쿼리가 모든 같은 곳 (인코더의 이전 계층의 output) 에서 나오고, encoder 의 각 포지션이 이전 계츠이 모든 포지션에 접근할 수 있게 함.
- decoder 의 masked self-attention 층 : 자동 회귀 성질을 보존하려면 디코더에서 왼쪽으로 정보 흐름이 일어나지 않도록 하야 하는데, scaled dot-product attention 내부에서 허용되지 않는 연결에 해당하는 값들을 마스킹하는 방식으로 이를 구현

3.3 Position-wise Feed Forward Networks

인코더와 디코더의 각 층에 FC feed-forward network 가 포함되어 있어 각 위치에 대해 독립적이고 동일하게 사용됨.

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

- 식은 같지만 층마다 다른 파라미터를 사용하게 됨
- 커널 크기 1의 합성곱 두 번으로도 표현될 수 있음
- d_model = 512, 가운데 층의 차원은 d_ff = 2048

3.4 Embeddings and Softmax

- input tokens / output tokens → d_model 차원 vector 변환에 learned embedding 사용
- decoder output → predicted next-token probabilites 변환에 usual linear transformation & softmax 함수 사용
- 두 개의 임베딩 층과 소프트맥스 이전의 선형 변환 사이에 동일한 가중치 행렬을 공유하고 , 임베딩 층에서는 그 가중치를 d_model 의 제곱근 값으로 스케일하여 사용

3.5 Positional Encoding

시퀀스 내 토큰의 상대적 또는 절대적 위치에 관한 정보를 주입해서 시퀀스의 순서 정보를 활용하기 위해 인코더와 디코더 스택의 입력 임베딩 하단에 포지셔널 인코딩을 더함

- 임베딩과 같은 차원 d_model 을 가짐
- sine , cosine 함수 사용
  - each dimension of the positional encoding corresponds to a sinusoid.

✅ **Positional Encoding**

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

위치 pos 와 차원 인덱스 i 에 대해
* W_i = 10000^(-2i/d_model) - 차원마다 주파수 다름

- 짝수 차원: $PE_{(pos,2i)} = \sin(\omega_i \, pos)$
- 홀수 차원: $PE_{(pos,2i+1)} = \cos(\omega_i \, pos)$

→ 각 차원 값이 하나의 '사인/코사인파' 값이므로 각 차원이 하나의 사인파에 대응한다.

  - since for any fixed offset k, PE_pos+k can be represented as a linear function of PE_pos

✅ **Positional Encoding**

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

각 주파수 W 에 대해:

$$\sin(\omega(pos+k)) = \sin(\omega pos)\cos(\omega k) + \cos(\omega pos)\sin(\omega k)$$
$$\cos(\omega(pos+k)) = \cos(\omega pos)\cos(\omega k) - \sin(\omega pos)\sin(\omega k)$$

백터 형태로 쓰면

$$\begin{bmatrix}\sin(\omega(pos+k)) \\ \cos(\omega(pos+k))\end{bmatrix} = \underbrace{\begin{bmatrix}\cos(\omega k) & \sin(\omega k) \\ -\sin(\omega k) & \cos(\omega k)\end{bmatrix}}_{\text{상수}(pos\text{와 무관})} \begin{bmatrix}\sin(\omega pos) \\ \cos(\omega pos)\end{bmatrix}$$

→ 고정된 k 에 대해 PE_pos+k 는 PE_pos 에 상수 행렬을 곱한 선형 변환으로 표현
⇒ self attention 이 상대 위치 (거리 k) 정보를 선형 결합만으로 쉽게 다룰 수 있게 됨.

- learned positional embedding 사용 실험 → 두 버전이 거의 비슷한 결과를 나타냄을 관찰

4. Why Self-Attention

- Self Attention VS CNN & RNN
  1. 층당 계산 복잡도
  2. 병렬화할 수 있는 계산의 양
  3. 네트워크에서 장거리 의존성 사이의 경로 길이

- 더 해석 가능한 모델 제공

  모델의 어텐션 분포를 점검하고 많은 헤드가 문장의 통사적 (syntactic) , 의미적 (semantic) 구조와 관련된 행동을 보이는 것처럼 보인다.

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)	$O(r \cdot n \cdot d)$	$O(1)$	$O(n/r)$

서로 다른 레이어 유형의 최대 경로 길이, 층당 계산 복잡도, 필요한 최소 순차 연산 수

5. Training

5.1 Training Data and Batching

- WMT 2014 영어-독일어 데이터셋
  - 4.5M 문장 쌍 포함
  - 문장들은 byte-pair encoding 으로 인코딩됨
- WMT 2014 영어-프랑스어 데이터셋
  - 36M 문장 포함
- 각 문장 쌍은 대략적인 sequence 길이에 따라 함께 배치
- 각 학습 배치는 대략 25000 개의 소스 토큰 , 25000 개의 타깃 토큰을 포함하는 문장 쌍들로 구성

5.2 Hardware and Schedule

- NVIDIA P100 GPU 8개가 장착된 하나의 머신에서 학습
- 학습 스텝당 약 0.4초 소요
- base model 은 총 100,000 스텝 (약 12시간) 정도 학습시킴
- 가장 큰 모델은 총 300,000 스텝 (약 3.5일) 동안 학습

5.3 Optimizer

- Adam Optimizer 사용.
- Learning Rate

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

  warmup step 동안은 lr 을 선형적으로 증가스키고 그 이후에는 스텝 수의 inverse square root 에 비례해 감소시킴

  - warmup step = 4000

5.4 Regularization

- Residual Dropout : 각 sub-layer 출력에 dropout, 임베딩/포지셔널 인코딩의 합에도 드롭아웃 적용 (p=0.1)
- Label Smoothing : εls=0.1

  ✅ Label Smoothing?

  퍼플렉서티는 조금 나빠지지만 정확도/BLEU 개선

- (+) Attention Dropout

6. Results

6.1 Machine Traslation

Transformer 는 훈련 비용의 극히 일부만으로 이전 최첨단 모델들보다 더 나은 BLEU 점수 달성

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**	$2.3 \cdot 10^{19}$	

- WMT 영어-독일어 번역 : 이전 모델들보다 BLEU 2.0 이상 앞서며 BLEU 28.4 의 새로운 SOTA 확립.

  → P100 에서 3.5일 학습

- WMT 영어-프랑스어 번역 과제 : BLEU 41.0 를 달성하며, 이전에 발표된 모든 단일 모델들을 능가.
  - SOTA 모델 학습 비용의 1/4 미만으로 이를 달성
  - 드롭아웃 비율을 0.3 대신 0.1 로 수정함

6.2 Model Variations

	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

→ Base 모델을 여러 가지로 변형시켜 실험

- (A) 행 : 계산량은 일정하게, 어텐션 헤드 수와 key/value 차원 변화

  헤드가 너무 많거나 적은 경우에 품질이 떨어짐

- (B) 행 : 어텐션 key 차원 d_k 를 줄이면 모델 품질이 악화

  compatibility을 정하는 일이 쉽지 않으며 dot product 보다 더 정교한 함수가 유리할 수 있음

- (C), (D) 행 : 더 큰 모델이 더 낫고, 드롭아웃이 과적합 방지에 도움이 된다는 것을 확인 가능

- (E) 행 : sinusoidal positional encoding 을 learned positional embedding 으로 교체했는데, 베이스 모델과 거의 동일한 결과

6.3 English Constituency Parsing

Transformer 가 다른 과제에도 일반화되는지 실험. 출력이 강한 구조적 제약을 따르고, 입력보다 출력 시퀀스가 훨씬 길다는 점에서 까다로움.

- 실험 설정

- 모델 : d_model = 1024, 4층 Transformer
- 데이터 : Penn Treebank 의 WSJ 구간 (약 40000 문장) / high-confidence 와 BerkleyParser 말뭉치 (약 1700만 문장) 사용 반지도 학습
- 어휘 : WSJ 는 16K, 반지도는 32K 토큰
- 튜닝 : 드롭아웃, lr, beam size 만 소규모로 하고 다른 부분은 영어-독일어 번역 base 모델과 동일

- 결과

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

task-specific tuning 이 거의 없이도 이전 모델들을 능가하는 결과를 보임.

7. Conclusion

- Transformer 모델을 제시 - 어텐션에 기반한 최초의 sequence model 로 , 인코더-디코더 구조에서 흔히 쓰이던 을 멀티헤드 자기어텐션으로 대체
- 학습 속도도 훨씬 빠름