Transformer

구글 브레인팀 : Attention is All You Need (2017)

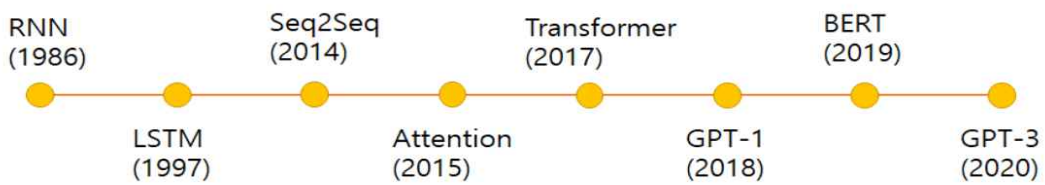
We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely. Experiments on two machine translation tasks show these models to be superior in quality while being more parallelizable and requiring significantly less time to train. Our model achieves 28.4 BLEU[1] 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.

1. 배경: 기존 RNN, LSTM 모델의 한계

기존의 RNN(Recurrent Neural Network)이나 LSTM (Long Short-Term Memory) 모델은 순차 데이터를 처리하기 위해 고안된 모델로, 입력 시퀀스를 순차적으로 처리하는 구조입니다. 하지만 이러한 모델들은 몇 가지 한계를 가지고 있었습니다.

- 연산 속도 문제: 시퀀스를 순차적으로 처리하기 때문에, 병렬 연산이 어렵고 시간이 오래 걸립니다.
- 장기 의존성 문제: 긴 시퀀스를 처리할 때, 과거의 정보가 멀어질수록 중요한 정보를 기억하기 어렵고, 기울기 소실 문제가 발생합니다.
- 병렬 처리 불가: 순차적 처리 방식으로 인해 긴 시퀀스에 대해 효율적으로 병렬 처리가 불가능했습니다.

구글의 Transformer 모델은 이러한 한계를 극복하기 위해 제안된 모델로, Attention 메커니즘만을 사용하여 시퀀스의 모든 요소를 한꺼번에 처리하고, 병렬 연산을 가능하게 하였습니다.

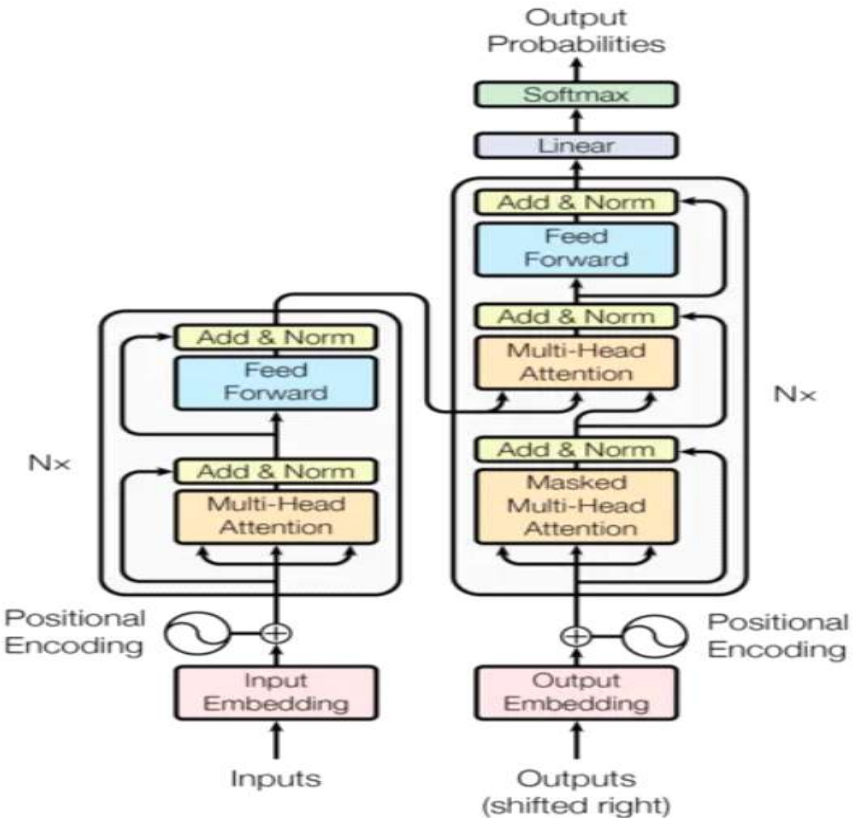


2. Transformer의 주요 개념

Transformer는 RNN이나 LSTM과 달리, 순환 구조를 사용하지 않고 Attention 메커니즘을

1)

기반으로 동작하는 모델입니다.
이는 입력 시퀀스의 모든 단어가 서로 어떤 관련이 있는지 파악하고, 중요한 정보를 찾아내어 가중치를 부여하는 방식으로 동작합니다.
Transformer 모델은 두 가지 주요 구성 요소로 나뉩니다.



▶ **Encoder (왼쪽)**

아나로그 --> 디지털

입력을 받아 입력의 표현(representation) 또는 feature을 만듭니다. :

입력 시퀀스를 처리하고, 입력의 의미를 이해하기 위해 중요한 정보를 추출합니다.

Encoder는 2개의 sublayer로 구성 :

⇨ **Multi-head attention layer와 Feed Forward layer**

▶ **Decoder (오른쪽)**

아나로그 --> 디지털

Encoder의 표현과 Decoder의 입력을 사용하여 target 시퀀스를 생성합니다.

Decoder는 3개의 sublayer로 구성 :

⇨ **Masked multi-head attention, Multi-head attention, Feed Forward layer**

Encoder와 마찬가지로 decoder에도 residual connection이 있다.

디코더(Decoder): 인코더의 출력과 함께 목표 시퀀스(target sequence)를 처리하여 최종 출력(예: 번역된 문장)을 생성합니다.

Transformer architecture

가장자리에 Nx라고 쓰여진 것을 볼 수 있는데 이것은 stack을 몇개의 구조로 쌓을지에 관한 정보이다. 그림엔 encoder와 decoder가 각각 1개씩 표현되어있지만 사실 Transformer는 이러한 구조가 여러 개가 쌓인 stack[2]이다.

이러한 stack 구조가 가능한 이유는 encoder에서 input vector와 output vector의 차원이 갖게 설계되어있기 때문이다. 따라서 이전 encoder의 output을 다음 encoder의 input으로 사용할 수 있는 것이다. decoder도 마찬가지이다. 원논문에선 stack의 갯수 N=6으로 설정하였다.

3. Transformer 모델의 구조(정리)

3.1 인코더(Encoder)

- 인코더는 트랜스포머 모델에서 입력 시퀀스(input sequence)를 처리하여 잠재 벡터(latent vector), 즉 내부 표현(Internal Representation)을 생성하는 역할을 합니다.
- 인코더는 여러 층으로 이루어진 네트워크로, 각 층이 입력 데이터를 어텐션 메커니즘(Self-Attention)을 통해 처리하여 의미 있는 표현을 추출합니다.

[인코더의 주요 역할]

- 입력 시퀀스를 이해하고, 이를 통해 문맥상 중요한 정보를 추출합니다.
- 단어 간의 상호 관계를 고려해 문장의 문맥 정보를 잘 파악합니다.
- 최종적으로 잠재 벡터로 표현된 정보를 디코더에 전달하여, 디코더가 이를 바탕으로 출력을 생성할 수 있도록 합니다.
- **아나로그 (자연어) → 디지털 (벡터화)**

▶ Embedding 레이어: 입력 텍스트를 벡터로 변환하여 단어의 의미를 임베딩.
▶ Positional Encoding: 입력 문장의 각 단어 위치에 대한 정보를 더해줍니다.
▶ Self-Attention: 각 단어가 문장 내 다른 단어들과의 관계를 학습하도록 합니다.
▶ Multi-Head Attention: 여러 가지 관점을 통해 단어의 문맥적 관계를 학습합니다.
▶ Feed-Forward Network: 각 단어 벡터에 독립적으로 변환을 적용합니다.
▶ Encoder Layer: Transformer의 기본 구성 요소인 인코더를 구현했습니다.

[인코더의 과정]

1) 입력 임베딩(Input Embedding)[3]

: 입력 단어를 고차원 벡터로 변환합니다.

2) 포지셔널 인코딩(Positional Encoding)[4]

- : Transformer는 순차적인 구조를 사용하지 않기 때문에, 입력 시퀀스에서 각 단어의 위치 정보를 추가적으로 제공해야 합니다.
- 이 위치 정보를 통해 단어 간의 순서를 모델이 이해할 수 있습니다.

---

2)
3)
4)

- Transformer 모델은 RNN처럼 순차적으로 데이터를 처리하지 않기 때문에, 입력 시퀀스 내에서 각 단어의 순서에 대한 정보를 추가적으로 제공해야 합니다.
- 이를 위해 포지셔널 인코딩(Positional Encoding)이 사용됩니다.
- 포지셔널 인코딩은 각 단어의 위치를 나타내는 벡터로, 이 벡터는 모델이 단어의 순서와 위치 정보를 이해하도록 돕습니다.
- 포지셔널 인코딩은 sine 함수와 코사인 함수를 사용해 계산되며, 각 위치에 고유한 패턴을 부여합니다.

3) Self-Attention

[Attention 메커니즘]

- Transformer 모델의 핵심은 Attention 메커니즘입니다.
- Attention은 입력 시퀀스의 각 단어가 다른 단어들과 얼마나 관련이 있는지를 측정하여, 더 중요한 단어에 가중치를 부여하는 방식입니다. --> 즉, 모델이 중요한 부분에 집중할 수 있도록 하는 방법입니다.

[어텐션의 주요 구성 요소]

- 쿼리(Query, Q): 현재 처리 중인 단어가 다른 단어들과 얼마나 관련이 있는지를 측정하는 기준 벡터입니다.
- 키(Key, K): 입력 시퀀스 내에서 각 단어의 정보(위치와 의미 등)를 나타내는 벡터입니다.
- 값(Value, V): 실제 단어의 정보를 담고 있는 벡터입니다.

어텐션은 쿼리 벡터와 키 벡터 간의 내적(dot product)을 계산하여, 해당 쿼리가 다른 단어들과 얼마나 관련 있는지를 측정하고, 이를 기반으로 가중치를 계산한 뒤, 해당 값 벡터에 가중치를 적용해 최종 출력을 생성합니다.

: Self-Attention 메커니즘을 통해 입력 문장 내에서 각 단어가 다른 단어와 어떻게 상호작용하는지 계산합니다. 각 단어의 문맥 정보를 다른 단어들과 비교해가며 강화하는 단계입니다.

****Self-Attention (자기 어텐션)****

Self-Attention은 입력 시퀀스의 각 단어가 ***동일한 시퀀스 내의 다른 단어들과 어떤 관계***를 가지는지를 측정하는 방법입니다.

Self-Attention을 사용하면 문맥을 고려한 중요한 정보가 반영됩니다.

[Self-Attention의 주요 아이디어]

- Self-Attention의 핵심은 시퀀스 내에서 각 단어가 동일한 시퀀스 내의 다른 단어들과의 관계를 스스로 학습한다는 점입니다. 이를 통해 문맥 정보를 더 잘 파악할 수 있습니다.

1) 자기 참조(Self-Referencing)

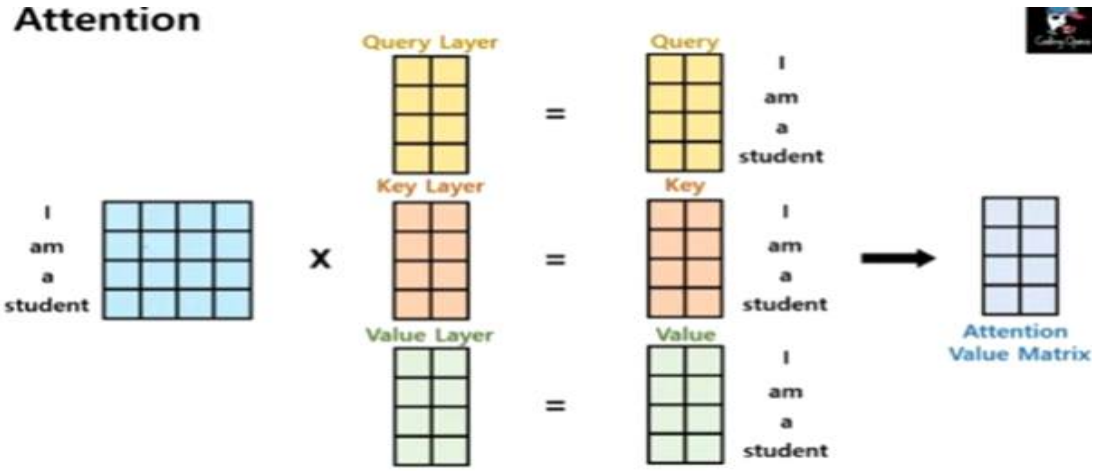
▷ Self-Attention에서 Query(Q), Key(K), Value(V)는 동일한 입력 시퀀스로부터 생성됩니다. 즉, 시퀀스의 각 단어는 시퀀스 내의 모든 다른 단어들과 상호작용하여, 각 단어 간의

관련성을 평가합니다.

▷ 예를 들어, 문장 내에서 어떤 단어가 다른 단어들과 얼마나 연관성이 있는지를 평가하는 것입니다. 이로써 모델은 시퀀스 내에서 중요한 관계(예: 명사와 관련된 형용사, 주어와 동사 등)를 학습할 수 있습니다.

2) 병렬 처리 가능

Self-Attention의 큰 장점은 RNN처럼 시퀀스를 순차적으로 처리할 필요가 없다는 것입니다. 모든 단어가 동시에 다른 단어들과의 관계를 계산할 수 있으므로 병렬 처리가 가능하며, 이로 인해 모델 훈련 속도가 크게 향상됩니다.



3) 위치 정보 추가(Positional Encoding)

▷ Self-Attention은 각 단어 간의 관계만을 학습하므로, 시퀀스의 순서 정보가 자연스럽게 포함되지 않습니다.

▷ 이를 해결하기 위해 Positional Encoding이라는 기법을 사용하여 시퀀스 내 단어들의 위치 정보를 추가로 반영합니다.

▷ Positional Encoding은 각 단어의 위치 정보를 고유한 벡터로 표현하여, 모델이 단어 간의 순서까지 학습할 수 있게 도와줍니다.

4) 다중 헤드(Multi-Head Attention, Self-Attention의 확장)

▷ Multi-Head Attention은 여러 개의 Self-Attention을 병렬로 수행하는 방식입니다.

▷ 이를 통해 다양한 패턴과 관계를 한 번에 학습할 수 있으며, 모델의 성능이 더욱 향상됩니다. 예를 들어, 어떤 헤드는 문맥 상의 명사와 동사의 관계를 학습하고, 다른 헤드는 형용사와 명사의 관계를 학습할 수 있습니다.

▶ Attention 메커니즘은 입력 시퀀스의 모든 부분을 동적으로 참조하여, 출력 시퀀스 생성 시 중요한 정보에 집중할 수 있도록 도와줍니다.

▶ Self-Attention은 동일한 시퀀스 내에서 단어들이 서로 간의 관계를 학습하며, 이를 병렬로 처리할 수 있어 매우 효율적입니다. 이는 특히 Transformer 모델에서 중요한 역할을 하며, 빠르고 강력한 성능을 가능하게 합니다.

다중 헤드(Self-Attention의 확장): Multi-Head Attention[5)]

	Attention	Self-Attention
작동 방식	인코더와 디코더 간의 상호작용에서 사용되며, 디코더가 인코더의 모든 출력에 집중하여 중요한 부분을 선택	시퀀스 내에서 각 단어가 다른 단어들과의 관계를 학습하여 중요한 정보를 스스로 선택
입력	Query는 디코더에서, Key와 Value는 인코더에서 생성	Query, Key, Value 모두 동일한 입력 시퀀스에서 생성
용도	주로 Seq2Seq 구조에서 입력과 출력 시퀀스 간의 관계를 학습할 때 사용	Transformer 모델에서 시퀀스 내 단어 간의 관계를 병렬로 학습
병렬 처리	순차적 처리 필요(RNN 기반일 때)	완전한 병렬 처리 가능
포지셔널 인코딩	필요 없음	필요 (시퀀스 내 순서 정보 보존을 위해)

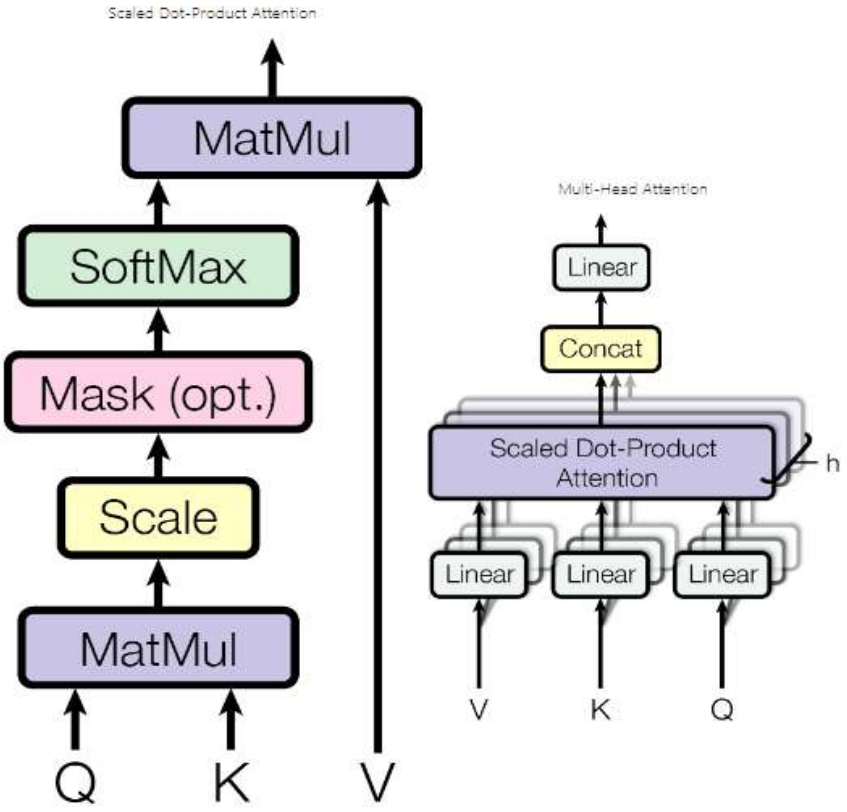


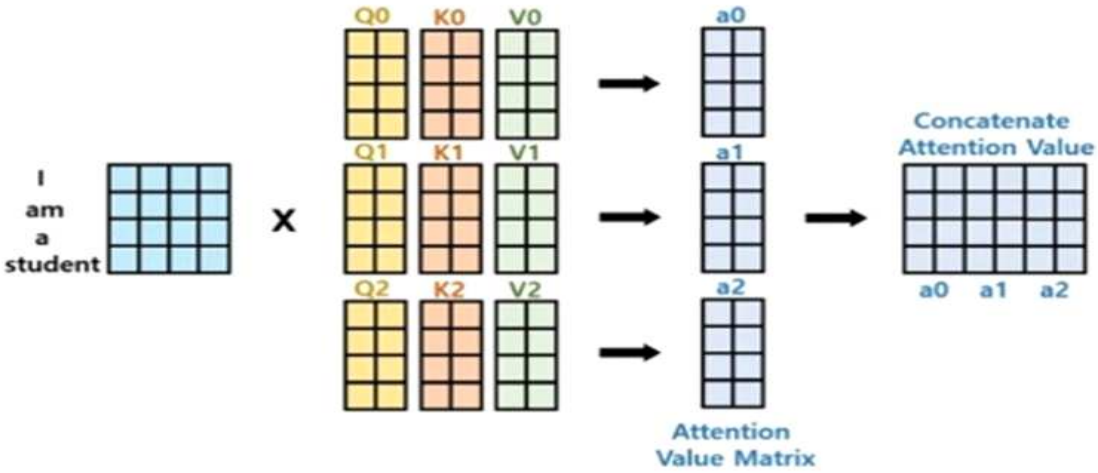
Figure 2: (left) Scaled Dot-Product Attention. (right) Multi-Head Attention consists of several attention layers running in parallel.

* 멀티헤드 어텐션(Multi-Head Attention)은 Attention 메커니즘의 확장된 개념으로, 단일 어텐션이 아닌 여러 개의 어텐션을 병렬로 사용하여 서로 다른 관점에서 단어 간의 관계를 학습합니다.

5)

예를 들어, 어떤 헤드는 문맥 상의 명사와 동사의 관계를 학습하고, 다른 헤드는 형용사와 명사의 관계를 학습할 수 있습니다.

- 각 헤드는 서로 다른 Query(Q), Key(K), 값을 학습하고, 이들이 병렬로 처리됩니다.
- 여러 개의 어텐션을 병합하여 더 풍부한 표현을 학습할 수 있습니다. 예를 들어, 한 헤드는 단어 간의 근접 관계를 학습하고, 다른 헤드는 문장의 전반적인 의미를 학습할 수 있습니다.
- 멀티헤드 어텐션을 통해 모델은 단어 간의 복잡한 관계를 더 잘 이해하고, 성능을 크게 향상시킬 수 있습니다.



ADD (잔차 연결) & Norm (정규화)[6)]

■ ADD : Residual Connection (잔차 연결)

: 잔차 연결은 모델이 학습을 더 쉽게 하고 기울기 소실(Vanishing Gradient) 문제를 완화하기 위해 도입되었습니다.→ 잔차연결(Residual Connection)과 관련하여 덧셈(Addition)

잔차 연결은 항상 작동하는 것이 아니라, 각 레이어의 출력을 계산할 때 특정 조건에서 적용됩니다.

잔차 연결의 목적

- 기울기 소실 문제 완화: 잔차 연결을 통해 역전파가 입력으로 더 잘 전달되므로, 네트워크가 깊어지더라도 기울기가 소실되지 않도록 도와줍니다.
- 더 쉬운 학습: 레이어가 학습할 때, 입력을 그대로 전달하면서 새로운 정보를 추가 학습하는 방식이므로, 학습이 더 쉬워지고 안정적으로 이루어집니다.
- 중요 정보 유지: 레이어가 지나치게 복잡해지더라도, 입력 자체를 그대로 유지하는 경로를 통해 중요한 정보가 손실되지 않고 전달됩니다.

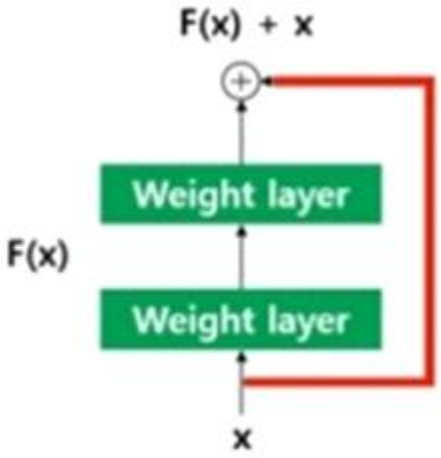
■ NORM(정규화, Normalization) 목적

신경망 학습을 안정화하고 효율성을 높이기 위해 입력 데이터나 층(layer)의 출력을 일정한 범위 내로 정규화하는 것입니다.

정규화 기법은 딥러닝 모델에서 주로 사용되며, 학습 속도 향상, 기울기 소실 방지, 모델의 일반화 능력 향상을 목표로 합니다.

---

6)



<Residual Connection>

- 학습 속도 향상 : 신경망에서 입력 데이터 또는 레이어의 출력값들이 너무 큰 값이나 작은 값으로 분포되면, 기울기 소실(Vanishing Gradient) 또는 기울기 폭발(Exploding Gradient) 문제가 발생할 수 있습니다. 이러한 문제는 신경망이 역전파 과정에서 가중치를 제대로 업데이트하지 못하게 하여 학습 속도를 저하시킵니다.

정규화(Normalization)를 통해 입력 또는 출력값을 일정한 범위로 조정하면, 신경망의 가중치 업데이트가 안정적으로 이루어져 학습 속도가 빨라집니다.
특히, Batch Normalization이나 Layer Normalization은 각 층에서 출력을 정규화하여, 학습 과정에서 더 빠르게 수렴할 수 있게 돕습니다.

- 기울기 소실 및 기울기 폭발 문제 완화

4) 피드포워드 네트워크(Feed-Forward Network):
어텐션을 통해 계산된 정보를 비선형 변환하여 더 복잡한 패턴을 학습합니다.
피드포워드 네트워크(Feed-Forward Network[7]): 어텐션 레이어 뒤에 각 위치별로 독립적인 피드포워드 신경망을 적용하여 더 복잡한 패턴을 학습할 수 있도록 합니다.

피드포워드 네트워크의 구조
Transformer에서 사용하는 피드포워드 네트워크는 두 개의 완전 연결층(fully connected layers)과 활성화 함수로 구성됩니다. Transformer에서 FFN은 시점별 위치마다 독립적으로 적용되며, 각 입력이 동일한 파라미터로 처리됩니다.

5) 출력
: 최종적으로 인코더는 입력 시퀀스의 내부 표현을 만들어내고, 이 값은 디코더로 전달됩니다.

6) 출력층(Output Layer): 디코더에서 최종 출력(예: 번역된 문장)을 생성합니다.

7)

[수학적 이해]

■ **인코더 과정**의 수학적 설명

인코더(Encoder)는 주로 Self-Attention과 피드포워드 네트워크를 사용해 입력 시퀀스의 문맥 정보를 추출합니다.

인코더는 여러 층으로 구성되며, 각 층은 다음과 같은 계산을 수행합니다.

(1~2) 입력 임베딩(Input Embedding) 및 포지셔널 인코딩(Positional Encoding)

각 단어 $x\ i$ 는 임베딩 벡터 $E(x\ i)$로 변환됩니다.

트랜스포머는 순차적인 정보를 유지하기 위해 포지셔널 인코딩 $P\ E(i)$를 추가합니다.

$$z_0 = E(x_i) + PE(i)$$

- E(x i) : 단어 $x\ i$ 의 임베딩 벡터
- $P\ E(i)$: 포지셔널 인코딩 벡터로, 각 단어의 위치 정보를 포함합니다.
- z 0 : 입력 임베딩에 포지셔널 인코딩이 더해진 값

(3) 어텐션 메커니즘(Self-Attention Mechanism)

인코더의 각 층에서 입력 데이터 간의 관계를 계산하는 Self-Attention 메커니즘을 적용합니다. Self-Attention은 입력 시퀀스의 모든 단어가 서로 어떻게 상호작용하는지를 계산합니다. 주된 계산 과정은 쿼리(Query), 키(Key), 값(Value)의 벡터를 사용하는 방식으로 이루어집니다.

- 쿼리, 키, 값 계산: 각 입력 벡터

$z\ 0$ 에 대해 쿼리 Q, 키 K, 값 V를 다음과 같이 계산합니다.

$$Q = z_0 W^Q, \quad K = z_0 W^K, \quad V = z_0 W^V$$

W^Q, W^K, W^V는 각각 쿼리, 키, 값의 **가중치 행렬**

- 어텐션 스코어(Attention Score) 계산: 쿼리와 키의 내적을 통해 어텐션 스코어를 계산한 후, 이를 정규화합니다.

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

d k : 키 벡터의 차원으로 소프트맥스 함수를 통해 어텐션 스코어를 정규화합니다.

이 계산을 통해 각 단어가 다른 단어와 얼마나 중요한 관계가 있는지 반영된 가중치 합이 계산됩니다.

(4) 피드포워드 네트워크(Feed-Forward Network)
Self-Attention을 통해 얻은 출력값을 피드포워드 네트워크에 통과시킵니다.
피드포워드 네트워크는 각 단어에 독립적으로 적용되며, 두 개의 선형 변환과 비선형 활성화 함수로 이루어져 있습니다.

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

- W 1 ,W 2 : 가중치 행렬
- b 1, b 2 : 편향 값
- max(0, x : ReLU 함수

(5) 인코더 출력
인코더의 각 층에서 Self-Attention과 피드포워드 네트워크를 거친 후, 잔차 연결(Residual Connection)[8]과 Layer Normalization을 적용합니다.

3.2 디코더(Decoder)

- 디코더는 인코더가 생성한 잠재 벡터를 바탕으로 출력 시퀀스(output sequence)를 생성하는 역할을 합니다.
- 디코더는 번역된 문장 또는 생성된 텍스트를 출력하는 작업에서 중요한 역할을 합니다.
- 디코더 역시 여러 층으로 구성되며, 인코더의 출력을 활용하면서도 자기 자신의 어텐션(Self-Attention)을 통해 이전에 생성된 단어들과 상호작용합니다.

[디코더의 과정]
1) 입력 임베딩(Output Embedding)
: 디코더는 출력 시퀀스(예: 번역된 문장의 단어)를 고차원 벡터로 변환합니다. 첫 시점에서는 보통 시작 토큰(start token)이 입력됩니다.
디코더는 시작 토큰을 입력받고, 출력 시퀀스의 각 단어에 대해 임베딩과 포지셔널 인코딩을 적용합니다.
2) 포지셔널 인코딩(Positional Encoding)
: 입력 단어 간의 순서 정보를 보존하기 위해 포지셔널 인코딩을 추가합니다.

$$y_0 = E(y_i) + PE(i)$$

- E(y i)는 출력 단어 y i 의 임베딩 벡터
- P E(i))는 포지셔널 인코딩

3) 마스크드 어텐션 (Masked Self-Attention)

---

8)

: 디코더는 이전에 생성된 단어들만 참조하여 현재 단어를 예측해야 하기 때문에, 마스크드(Self-Attention)를 사용하여 미래의 단어를 보지 않도록 설정합니다.
--> 디코더는 마스크드 어텐션을 사용하여 미래의 단어를 보지 않고, 현재까지 생성된 단어만 참조하여 출력을 생성합니다.

- 쿼리, 키, 값 계산: 디코더에서도 쿼리, 키, 값을 다음과 같이 계산합니다.

$$Q = y_0 W^Q, \quad K = y_0 W^K, \quad V = y_0 W^V$$

- 마스크 적용: 마스크를 적용하여 현재 시점 이후의 단어에 대한 정보를 보지 못하게 합니다. 마스크 행렬 M은 다음과 같이 적용됩니다.

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

* Masked multi-head attention[9)]
: 다음에 나올 단어를 미리 보지 못하게 하는 제약을 가해 시퀀스 생성을 더 효과적으로 만들기 위한 기법이다.
Masked Multi-Head Attention은 다음에 나올 단어를 미리 보지 못하게 하는 제약을 가해 시퀀스 생성을 더 효과적으로 만들기 위한 기법입니다.
이 기법은 주로 텍스트 생성 작업(예: 번역, 요약)에서 사용되며, 디코더가 현재 시점에서 앞으로 생성해야 할 단어를 보지 않고 학습하도록 하는 데 필수적입니다.
Masked Multi-Head Attention이 필요한 이유
Transformer의 디코더는 시퀀스 생성 모델로 작동하기 때문에, 예측할 때 미래의 정보를 참조해서는 안 됩니다. 예를 들어, 디코더가 문장의 첫 번째 단어를 예측할 때, 아직 예측하지 않은 두 번째, 세 번째 단어의 정보를 미리 알게 된다면, 그 과정이 학습에 부정적인 영향을 미칩니다. 따라서, 현재 시점까지의 정보만을 기반으로 단어를 생성해야 합니다.

4) 인코더-디코더 어텐션(Encoder-Decoder Attention)
: 디코더는 인코더에서 생성된 잠재 벡터와 현재의 디코더 상태를 결합하여 입력 시퀀스의 정보에 따라 출력 시퀀스를 생성합니다. 이 단계에서 인코더의 출력을 참조하여 현재 시점의 단어를 결정합니다.
디코더는 마스크드 어텐션을 수행한 후, 인코더의 출력을 참조하여 다음 단어를 생성합니다. 이때, 인코더의 출력을 참조하여 디코더의 출력을 조정합니다.

- 쿼리, 키, 값 계산: 쿼리는 디코더의 은닉 상태
$y\ l$ 에서 계산되고, 키와 값은 인코더의 출력에서 계산됩니다.

$$Q = y_l W^Q, \quad K = z_L W^K, \quad V = z_L W^V$$

- 인코더-디코더 어텐션 계산: 디코더의 쿼리와 인코더의 키를 통해 어텐션 스코어를 계산한

9)

후, 인코더의 값을 가중합하여 디코더의 출력에 반영합니다.

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

5) 피드포워드 네트워크

마스크드 어텐션과 인코더-디코더 어텐션의 출력을 피드포워드 네트워크에 통과시켜 최종 출력을 계산합니다.

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

6) 출력

- 최종적으로 디코더는 출력 시퀀스의 각 단어를 예측합니다.
- 디코더는 각 시점마다 하나의 단어를 생성하며, 이 과정은 종료 토큰(end token)이 나올 때까지 반복됩니다.
- 디코더는 최종적으로 각 시점마다 다음 단어를 예측하여 출력 시퀀스를 생성합니다. 이 과정도 마찬가지로 잔차 연결과 Layer Normalization이 포함됩니다.

[디코더의 주요 역할]

- 인코더의 출력을 바탕으로 입력 시퀀스의 의미를 이해하고, 이를 바탕으로 출력 시퀀스를 생성합니다.
- 이전 출력과 현재 입력 간의 상호작용을 통해 새로운 단어를 예측합니다.
- 출력 시퀀스를 단계적으로 생성하며, 이전에 생성된 단어를 기반으로 다음 단어를 예측합니다.
-

3.3. 인코더와 디코더의 비교 및 차이점

특징	인코더(Encoder)	디코더(Decoder)
입력 데이터	원본 입력 시퀀스(예: 번역할 문장).	이전에 생성된 단어(또는 시작 토큰)를 입력받음.
출력 데이터	입력 시퀀스의 잠재 벡터 표현(문장의 의미와 문맥 정보를 포함한 상태).	최종 출력 시퀀스(예: 번역된 문장 또는 생성된 텍스트).
포지셔널 인코딩	입력 시퀀스의 단어 순서를 보존하기 위해 사용.	출력 시퀀스의 단어 순서를 보존하기 위해 사용.
어텐션(Self-Attention)	입력 시퀀스 내의 각 단어들이 서로 어떻게 연결되어 있는지 계산.	**마스크드 어텐션**을 사용하여 현재 시점까지 생성된 단어만 참조.
인코더-디코더 어텐션	없음	**인코더의 출력**을 참조하여, 출력 시퀀스를 생성.
주요 역할	입력 시퀀스에서 **의미 있는 표현**을 추출하고, 이를 통해 입력 데이터를 **잠재 벡터**로 변환.	인코더의 출력과 **자신의 이전 출력**을 바탕으로 **출력 시퀀스**를 단계적으로 생성.

3-4 트랜스포머 인코더-디코더의 상호작용

- 인코더는 입력된 원본 문장의 문맥 정보를 추출하여 잠재 벡터로 변환합니다.

- 이 잠재 벡터는 입력 시퀀스 전체에 대한 의미를 포함하고 있으며, 디코더는 이 벡터를 기반으로 번역 작업이나 텍스트 생성 작업을 수행할 수 있습니다.
- 디코더는 인코더에서 생성된 정보를 참조하면서 단어를 하나씩 생성합니다. 이 과정에서 디코더는 자기 자신의 이전 출력과 인코더의 출력을 결합하여 출력 문장의 각 단어를 예측합니다.

3-5. 예시: 번역 작업에서 인코더와 디코더

- 인코더의 역할: 만약 "I am a student"라는 영어 문장을 입력으로 받았다면, 인코더는 이 문장을 분석하여 각 단어 간의 관계와 문장의 의미를 추출하고, 이를 잠재 벡터로 변환합니다.
- 디코더의 역할: 디코더는 이 잠재 벡터를 바탕으로 번역된 문장을 생성합니다. 첫 번째 단계에서는 "저는"과 같은 한국어 단어를 생성하고, 이 단어와 잠재 벡터를 결합하여 다음 단어를 예측하는 방식으로 문장을 완성해 나갑니다.

4. 장점 및 성능

1) 병렬 처리

RNN이나 LSTM과 달리, Transformer는 모든 시퀀스 데이터를 한꺼번에 처리할 수 있으므로 병렬 처리가 가능합니다. 이로 인해 학습 속도가 크게 향상됩니다.

2) 장기 의존성 문제 해결

Attention 메커니즘 덕분에, Transformer는 장기 의존성 문제를 해결하는 데 매우 효과적입니다. 각 단어 간의 관계를 직접 계산하기 때문에, 긴 문장에서도 중요한 정보를 쉽게 처리할 수 있습니다.

3) 성능 향상

Transformer는 기존 RNN 기반 모델들보다 더 높은 성능을 발휘하며, 번역뿐만 아니라 문장 생성, 텍스트 요약 등의 다양한 자연어 처리 작업에서 매우 뛰어난 결과를 보여줍니다.

5. Transformer의 혁신과 영향

"Attention is All You Need" (2017) 논문은 NLP 분야에서 큰 혁신을 일으켰습니다. Transformer 모델은 이후 자연어 처리에서 중요한 역할을 하며, 이를 기반으로 한 다양한 모델들이 개발되었습니다.

대표적인 예시로는 다음과 같은 모델들이 있습니다.

- BERT(Bidirectional Encoder Representations from Transformers): 양방향으로 문맥을 이해하는 사전 훈련된 언어 모델
- GPT(Generative Pre-trained Transformer): 대규모 텍스트 데이터를 사전 훈련하여, 문장 생성 및 텍스트 예측에 강력한 성능을 보이는 모델

Machine Translation

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	$\mathbf{3.3 \cdot 10^{18}}$	
Transformer (big)	**28.4**	**41.8**	$2.3 \cdot 10^{19}$	

Transformer의 encoder와 decoder의 각 부분은 task에 따라 독립적으로 사용될 수 있다.

■ Encoder 전용 모델

Encoder 모델은 Transformer의 encoder만 사용하는 모델로 각 stage에 어텐션 레이어가 초기 문장의 모든 단어에 접근할 수 있습니다.

Encoder 모델을 흔히 auto-encoding 모델이라고도 합니다.

이 모델을 pre-training하기 위해서는 일반적으로 주어진 문장 중 임의의 단어를 masking 한 후, 모델이 원래 문장을 찾거나 복원하는 작업을 수행하게 한다.

▷ 입력에 대한 이해가 필요한 task에 사용

▷ e.g., 문장 분류(sentence classification), 명명된 개체 인식(named entity recognition), 질문 답변(question answering)

Encoder 모델은 전체 문장의 이해를 요구하는 task에 가장 적합하며 대표적인 Encoder 모델은 BERT입니다.

■ **Decoder** 전용 모델

Decoder 모델은 Transformer의 decoder만 사용하며 auto-regressive 모델이라고 부르기도 한다.

Decoder 모델의 어텐션 레이어는 각 stage에서 문장 내에서 주어진 단어의 바로 앞 단어만을 접근할 수 있다.

Decoder 모델의 pre-training은 일반적으로 문장의 다음 단어 예측을 중심으로 이루어집니다.

따라서 텍스트 생성과 관련 task에 가장 적합g나 것으로 알려집니다.

▷ 생성 task에 적합함

▷ e.g., 텍스트 생성(text generation)

대표적인 Decoder 모델로 GPT 모델이 있습니다.

■ Encoder-Decoder 모델

▷ 입력이 필요한 생성 task에 적합함

▷ e.g., 번역(translation) 또는 요약(summarization)

Encoder-Decoder 모델(또는 Sequence-to-Sequence 모델)은 Transformer의 Encoder와 Decoder를 모두 사용한다.

각 stage에서 encoder의 어텐션 레이어는 초기 문장의 모든 단어에 접근할 수 있는 반면, decoder의 어텐션 레이어는 입력으로 주어진 단어 앞 단어에만 접근할 수 있다.

Encoder-Decoder 모델은 주어진 입력에 따라 새로운 문장을 생성하는 요약, 번역, 생성적 질문 답변에 가장 적합하다.

[주석 용어 정리]

주1. BLEU (Bilingual Evaluation Understudy)

■ BLEU(Bilingual Evaluation Understudy) 점수란?

기계 번역의 품질을 평가하기 위한 대표적인 지표입니다. 기계 번역 모델이 번역한 문장과 참조 번역(reference translation) 사이의 유사도를 측정하는 방식으로, 점수가 높을수록 번역의 품질이 높다고 평가됩니다. BLEU는 주로 기계 번역, 텍스트 요약 등의 자연어 처리 모델 성능을 평가하는 데 사용됩니다.

BLEU 점수의 핵심 개념

1. n-그램 매칭: BLEU는 기계 번역이 생성한 문장과 참조 번역 간의 n-그램(n-gram) 일치율을 계산합니다. 여기서 n-그램은 연속된 단어들의 집합으로, 예를 들어 "I am a student"이라는 문장에서는 2-그램이 "I am", "am a", "a student"입니다.

1-그램부터 4-그램까지 다양한 n-그램을 사용하여 기계 번역과 참조 번역 사이의 일치율을 측정합니다.

2. 정확도(Precision): 기계 번역된 문장에서 참조 번역과 일치하는 n-그램의 비율을 측정합니다. 즉, 기계 번역에서 사용된 단어들이 얼마나 참조 번역과 일치하는지를 나타냅니다.

3, 짧은 문장에 대한 페널티 (Brevity Penalty): BLEU 점수는 문장이 지나치게 짧아지는 것을 막기 위해, 번역된 문장이 참조 문장보다 짧을 경우 페널티를 적용합니다. 이를 통해 짧은 번역 문장이 점수를 인위적으로 높이는 것을 방지합니다.

4.. 0~1 사이의 값: BLEU 점수는 0에서 1 사이의 값으로 표현되며, 일반적으로 100점 만점으로 변환하여 해석합니다. 1에 가까울수록 번역의 질이 높다고 평가됩니다.

■ BLEU 점수 계산 방식

BLEU는 주로 다음 수식을 사용하여 계산됩니다.

$$\text{BLEU} = BP \cdot \exp\left(\sum_{n=1}^{N} w_n \log p_n\right)$$

▷ BP는 Brevity Penalty(문장이 짧을 때 부여되는 페널티)입니다.

▷ p_n은 n-그램 정확도를 나타내고, w_n은 각 n-그램에 가중치를 부여하는 요소입니다.

▷ 일반적으로 1-그램부터 4-그램까지의 가중치를 동일하게 부여합니다.

■ BLEU의 한계

BLEU는 단순한 단어 일치에 의존하기 때문에 문장의 문법적 구조나 문맥을 잘 반영하지 못하는 한계가 있습니다.

여러 번역 가능성을 고려하지 못하고, 하나의 참조 번역만을 기준으로 평가하므로, 동일한 의미를 다른 표현으로 번역한 경우 점수가 낮게 나올 수 있습니다.

주2. Transformer에서의 Stack

Transformer 모델에서 Stack은 주로 인코더(Encoder)와 디코더(Decoder) 블록을 여러 개 쌓은 구조를 의미합니다. 각각의 인코더와 디코더 블록은 어텐션 메커니즘과 피드포워드 네트워크로 구성되며, 이러한 블록이 여러 층 쌓여 있는 것이 Transformer의 Stack 구조입니다.
Transformer의 Stack 구조:
인코더(Encoder) Stack:
Transformer의 인코더는 N개의 인코더 블록이 쌓인 구조입니다. 각 인코더 블록은 멀티헤드 어텐션과 피드포워드 네트워크로 구성되어 있습니다.
인코더 블록들은 위에서 아래로 차례차례 쌓여 있으며, 각 블록은 이전 블록의 출력을 받아 다음 블록으로 전달합니다.
기본 Transformer에서는 인코더 블록을 6개 쌓아 사용합니다.
디코더(Decoder) Stack:
디코더 역시 N개의 디코더 블록이 쌓인 구조입니다. 각 디코더 블록은 자기 어텐션(Self-Attention), 인코더-디코더 어텐션 및 피드포워드 네트워크로 구성됩니다.
디코더 Stack도 인코더와 유사하게 블록들이 순차적으로 쌓여 있습니다.
인코더-디코더 구조:
Transformer는 인코더와 디코더로 나뉘어 있으며, 두 부분이 Stack으로 이루어져 있습니다. 인코더 Stack은 입력 시퀀스에서 중요한 정보(특징)를 추출하고, 디코더 Stack은 인코더에서 추출된 정보와 디코더의 입력을 결합하여 최종 출력을 생성합니다.
3. Stack의 역할
Stack 구조는 딥러닝 모델에서 중요한 역할을 합니다. 각 레이어가 이전 레이어에서 더 복잡한 정보 또는 패턴을 학습하면서, 계층적으로 점차적으로 복잡한 특징을 학습해 나가는 것이 Stack의 핵심입니다. 이 과정을 통해 **심층 학습(Deep Learning)**이 가능해지며, 텍스트 번역, 이미지 인식 같은 복잡한 문제를 해결할 수 있게 됩니다.
Stack의 장점:
복잡한 패턴 학습: Stack 구조는 데이터의 저차원 특징에서 시작해 고차원 특징을 점차적으로 학습하며, 복잡한 관계나 패턴을 모델이 이해할 수 있도록 합니다.
표현력 증가: 레이어를 쌓음으로써 모델의 표현력이 증가하여, 더 다양한 패턴을 학습할 수 있습니다.
모듈화: 여러 레이어를 쌓은 구조는 모듈화된 처리 과정을 통해 각 레이어에서 다양한 기능을 수행할 수 있습니다. 예를 들어, Transformer에서는 어텐션과 피드포워드 네트워크가 반복적으로 쌓이면서 다양한 수준의 정보를 처리합니다.

4. Stack의 장단점
장점:
복잡한 패턴 학습 가능: 깊은 네트워크를 통해 더 복잡한 패턴을 학습할 수 있습니다.
모듈화된 구조: 각 레이어가 모듈화되어 다양한 기능을 수행할 수 있습니다.
계층적 학습: 저차원 특징부터 고차원 특징까지 점진적으로 학습하는 방식이 가능합니다.
단점:
기울기 소실 문제: 깊은 네트워크는 기울기 소실(vanishing gradient) 문제에 취약할 수 있습

니다. 이를 해결하기 위해 Residual Connection이 Transformer에서 사용됩니다.
계산 비용 증가: 깊어질수록 계산 비용이 증가하므로, 모델의 복잡도와 계산 효율성을 균형 있게 고려해야 합니다.

주3. 입력 임베딩(Input Embedding)

입력 임베딩(Input Embedding)은 자연어 처리(NLP)에서 단어 또는 기호를 컴퓨터가 처리할 수 있는 고차원 벡터로 변환하는 중요한 단계입니다.
텍스트 데이터는 본질적으로 이산적인 데이터를 포함하고 있기 때문에, 이를 연속적인 수치형 벡터로 변환해야만 딥러닝 모델이 효과적으로 학습할 수 있습니다.
이 변환 과정에서 사용되는 것이 단어 임베딩(word embedding)이며, 이는 텍스트 데이터를 더 의미 있는 수치적 표현으로 바꿔주는 역할을 합니다.
1. 입력 임베딩의 필요성
딥러닝 모델은 숫자를 입력으로 받아들이기 때문에, 텍스트 데이터를 처리하기 위해서는 단어를 숫자로 변환해야 합니다. 하지만 단순히 고유한 숫자로 인코딩하는 방식(예: One-Hot 인코딩)은 매우 비효율적입니다. One-Hot 벡터는 고차원이지만 대부분의 값이 0이고, 단어 간의 의미적 관계를 나타내지 못하는 한계가 있습니다.
입력 임베딩(Input Embedding)은 이를 해결하기 위해 각 단어를 고차원 벡터 공간으로 변환하여, 단어 간의 유사성과 의미적 관계를 학습할 수 있도록 합니다. 이를 통해 같은 문맥에서 자주 등장하는 단어들이 유사한 벡터로 표현될 수 있습니다.

2. 입력 임베딩의 작동 원리
입력 임베딩(Input Embedding)은 일반적으로 단어를 고차원 벡터로 변환하기 위한 행렬 곱으로 수행됩니다.
변환된 벡터는 밀집 벡터(Dense Vector)로, 각 차원이 단어의 의미와 관련된 정보를 포함합니다. 입력 임베딩을 적용하는 방법은 크게 두 가지로 나눌 수 있습니다:
1) Pre-trained Embeddings (사전 학습된 임베딩)
사전 학습된 임베딩은 대규모 말뭉치(코퍼스)를 사용하여 미리 학습된 단어 벡터를 사용하는 방식입니다. 대표적인 예로 Word2Vec, GloVe, FastText 등이 있습니다.
이 방법을 사용하면 모델을 학습하기 전에 이미 단어 간의 유사성이 학습되어 있기 때문에, 초기 학습을 더 효과적으로 시작할 수 있습니다.
2) 학습 가능한 임베딩 (Learned Embeddings)
학습 가능한 임베딩은 모델이 학습하면서 단어 임베딩 행렬을 함께 학습하는 방식입니다. 모델이 입력 데이터를 처리하면서 임베딩 가중치도 업데이트되므로, 주어진 작업에 맞게 단어 간의 관계를 조정할 수 있습니다.
Transformer와 같은 모델에서는 임베딩 레이어가 포함되어 있으며, 학습 과정에서 이 레이어의 가중치가 업데이트되어 단어 벡터를 학습합니다.

임베딩 행렬의 작동 방식:
임베딩 행렬(Embedding Matrix)은 각 단어에 해당하는 고차원 벡터를 저장하는 테이블입니

다.
크기가 [$V \times d$]인 행렬로, 여기서 V는 어휘 크기(vocabulary size), d는 임베딩 차원수입니다.
각 단어는 이 임베딩 행렬의 한 행에 해당하는 벡터로 변환됩니다. 즉, 입력 단어를 임베딩 벡터로 변환하기 위해 단어 인덱스를 사용하여 이 행렬에서 해당 벡터를 가져옵니다.
예시:임베딩 행렬이 다음과 같이 정의된다고 가정합시다.

$$\text{임베딩 행렬} = \begin{pmatrix} \text{king} & \rightarrow & [0.3, 0.7, -0.1] \\ \text{queen} & \rightarrow & [0.31, 0.69, -0.09] \\ \text{man} & \rightarrow & [0.1, -0.8, 0.6] \\ \text{woman} & \rightarrow & [0.11, -0.79, 0.61] \end{pmatrix}$$

여기서 단어 "king"은 [0.3, 0.7, -0.1]라는 3차원 벡터로 임베딩됩니다. 이 벡터는 학습을 통해 단어의 의미적 유사성을 반영하도록 조정됩니다.
3. 입력 임베딩의 수학적 표현
임베딩은 수학적으로는 임베딩 행렬(Embedding Matrix)과 원-핫 벡터(One-Hot Vector)의 곱으로 표현됩니다. 원-핫 벡터는 각 단어를 표현하는 매우 큰 벡터로, 단어의 인덱스 위치에만 1이 있고 나머지는 0인 벡터입니다.

임베딩 행렬과 원-핫 벡터:
원-핫 벡터(One-Hot Vector): 단어를 고유 인덱스로 변환한 벡터로, 특정 단어에 해당하는 인덱스만 1이고 나머지는 0입니다.
예시:
단어 'king'이 어휘에서 1번째 인덱스를 차지한다고 가정하면, 원-핫 벡터는
[1,0,0,0][1,0,0,0]이 됩니다.
단어 'queen'이 2번째 인덱스라면, 원-핫 벡터는
[0,1,0,0][0,1,0,0]이 됩니다.
임베딩 행렬(Embedding Matrix): 어휘의 모든 단어가 고차원 벡터로 변환된 행렬입니다.
크기:
[$|V| \times d$] (어휘 크기 V, 임베딩 차원 d)
각 행은 단어의 임베딩 벡터를 나타냅니다.
임베딩 벡터 계산: 임베딩은 원-핫 벡터와 임베딩 행렬을 곱하여 계산됩니다.

$$e_i = W \cdot x_i$$

여기서,W는 임베딩 행렬 [$|V| \times d$]
x i 는 원-핫 벡터 [$|V|$]
e i 는 임베딩 벡터 [d]입니다.

원-핫 벡터에서 1이 있는 위치에 해당하는 행이 그대로 임베딩 벡터로 출력됩니다.

4. Transformer에서의 입력 임베딩
Transformer 모델에서 입력 단어는 먼저 임베딩 레이어를 통해 고차원 벡터로 변환됩니다. 이후 포지셔널 인코딩(Positional Encoding)을 통해 단어의 순서 정보가 더해집니다. Transformer는 시퀀스의 각 단어가 고유한 위치를 가지고 있기 때문에, 이를 명확히 하기 위해 단어 벡터에 위치 정보를 추가하는 방식으로 임베딩이 확장됩니다.
Transformer에서 입력 임베딩의 특징:
고차원 벡터 변환: 각 단어는 정수 인덱스로 표현되며, 임베딩 레이어를 거쳐 고차원 벡터로 변환됩니다.
포지셔널 인코딩 추가: Transformer는 RNN처럼 순차적인 처리 과정을 따르지 않으므로, 각 단어의 위치 정보를 명시적으로 추가해야 합니다. 이를 위해, 포지셔널 인코딩을 임베딩 벡터에 더해줍니다.
병렬 처리 가능: 모든 단어가 동시에 처리되므로, RNN이나 LSTM과 달리 순차적으로 처리할 필요가 없습니다. 따라서 병렬 처리가 가능합니다.

주4. 포지셔널 인코딩(Positional Encoding)
Transformer 모델에서 사용되는 중요한 개념으로, 입력 시퀀스의 위치 정보를 모델에 제공하는 방법입니다.
Transformer는 RNN이나 LSTM과 달리 순차적으로 데이터를 처리하지 않기 때문에 단어의 순서를 명시적으로 알려주지 않으면 모델이 입력 시퀀스의 순서를 인식할 수 없습니다. 따라서 포지셔널 인코딩은 시퀀스의 각 위치를 벡터로 인코딩하여, 단어의 순서와 위치 정보를 모델이 인식하도록 돕습니다.

1. 포지셔널 인코딩의 필요성
Transformer 모델은 입력 데이터를 병렬 처리하므로, 순차적으로 데이터를 처리하는 RNN과 달리 시퀀스 내에서 각 단어의 순서를 학습할 수 있는 구조가 없습니다. 예를 들어, "The cat is on the mat"과 "The mat is on the cat"이라는 문장은 단어의 순서는 다르지만 동일한 단어로 구성되어 있습니다. Transformer는 단어의 순서 정보를 고려하지 않으면 두 문장을 동일하게 처리할 수 있습니다.

기존 RNN과 같은 경우 sequential하게 처리하므로 이러한 word order에 대한 정보가 내재되어 있었다. 그러나 Transformer는 이러한 RNN 구조를 사용하지 않으므로 해당 정보를 얻을 수 없다. 그렇다면 Transformer는 이러한 word order 정보를 어떻게 알까?

그래서, 단어의 위치 정보를 임베딩 벡터에 추가하는 것이 필요합니다. 포지셔널 인코딩은 각 단어의 위치를 벡터로 인코딩하여 모델이 단어의 위치를 알 수 있도록 합니다.

positional encoding이다. 해당 word의 position에 대한 정보를 따로 만들어 기존의 input 데이터와 합치는 것이다. 본 논문에서는 positional encoding을 위해 sine, cosine과 같은

주기 함수를 사용하였다.

2. 포지셔널 인코딩의 구조

- 포지셔널 인코딩은 주로 사인(sin)과 코사인(cos) 함수에 기반하여 계산됩니다.
- 이 함수들은 주기적이기 때문에, 임베딩 벡터에 위치 정보를 부여할 수 있으며, 시퀀스의 길이가 길어지더라도 패턴을 유지할 수 있습니다.
- Transformer에서는 포지셔널 인코딩이 다음과 같은 공식을 사용하여 계산됩니다,

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

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

$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{\frac{2i}{d}}}\right)$$

$$PE_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{\frac{2i}{d}}}\right)$$

- pos: 시퀀스에서의 단어의 위치 (position, 0부터 시작).
- i: 벡터의 차원 (0, 1, 2, …).
- d: 임베딩 벡터의 차원 크기 (임베딩 차원의 총 길이).

포지셔널 인코딩 설명:

▷ 짝수 차원(2i): 해당 차원의 포지셔널 인코딩 값은 사인 함수를 통해 계산됩니다.

▷ 홀수 차원(2i+1): 해당 차원의 포지셔널 인코딩 값은 코사인 함수를 통해 계산됩니다.

이 방식으로, 시퀀스 내 각 단어의 위치에 따라 사인과 코사인 값을 다르게 부여하여 고유한 위치 정보를 생성합니다.

사인과 코사인 함수가 사용되는 이유

▷ 위치 정보의 주기적 성질:

사인과 코사인 함수는 주기적(periodic) 성질을 가지고 있어, 입력된 시퀀스에서 위치 정보를 일정한 주기로 표현할 수 있습니다. 즉, 시퀀스의 길이가 달라지더라도 각 위치는 고유한 주기적 표현을 가지게 됩니다. 이는 모델이 시퀀스의 상대적 위치 정보를 학습하는 데 매우 효과적입니다.

특히, 사인과 코사인의 파동 패턴이 다르기 때문에, 두 함수를 함께 사용하여 위치 정보를 다양하게 인코딩할 수 있습니다.

▷ 다양한 주파수 정보 제공

포지셔널 인코딩에서 각 위치 pos는 차원별로 다른 주파수(frequency)를 갖는 사인과 코사인

값으로 인코딩됩니다. 이 방식은 다른 차원에서 위치 정보를 더욱 풍부하게 제공합니다.
사인과 코사인은 주파수가 변할 때마다 각 위치에서 다양한 패턴의 인코딩을 가능하게 합니다. 이를 통해 모델이 멀리 떨어진 단어들 간의 관계뿐만 아니라, 가까운 단어들 간의 관계도 학습할 수 있습니다.

▷ 상대적 위치 정보

사인과 코사인 함수의 값은 각 위치 간의 상대적 거리에 따라 자연스럽게 변화합니다. 예를 들어, 가까운 단어들 간의 위치 정보는 유사하게 인코딩되며, 멀리 떨어진 단어들은 상이한 값을 가집니다. 이러한 특성은 모델이 단어 간의 상대적 위치를 인식할 수 있도록 도와줍니다.

▷ 연속적이며 부드러운 변화

사인과 코사인은 연속적이고 부드럽게 변화하는 함수이기 때문에, 단어 간의 순서가 조금씩 달라져도 인코딩된 값이 연속적으로 변합니다. 이를 통해 모델이 시퀀스 내의 순서 변화를 자연스럽게 인식할 수 있습니다.

Transformer 모델에서 각 단어는 512차원의 벡터로 변환되며, 이 차원은 임베딩 공간의 크기입니다. 즉, 각 단어는 512차원 공간에서 표현되며, 이 공간에서 단어 간의 관계를 학습하게 됩니다.

512의 역할

- 임베딩 차원수는 Transformer에서 단어 벡터의 크기입니다.
- Transformer는 각 단어를 단순한 정수 인덱스가 아닌, 512차원의 고차원 벡터로 변환한 후 모델에 입력합니다.
- 포지셔널 인코딩은 이 512차원 벡터 각각에 위치 정보를 부여하기 위해, 사인(sin)과 코사인(cos) 함수를 사용하여 각 차원별로 위치를 반영한 값을 더해줍니다.
- 여기서 pos는 단어의 위치, 2i는 임베딩 벡터의 짝수 차원, 2i+1는 홀수 차원, 그리고 d는 임베딩 벡터의 전체 차원수(512)를 의미합니다.

512차원의 선택 이유

- 512는 일반적으로 NLP 작업에서 임베딩 차원수로 많이 사용되는 값입니다.
- 너무 낮으면 충분히 복잡한 관계를 학습하기 어렵고, 너무 높으면 과적합(overfitting)의 위험이 있습니다.
- 실제로, BERT, GPT, Transformer 등의 최신 모델에서 자주 사용되는 임베딩 차원수는 256, 512, 1024 등이며, 512는 효율성과 성능을 모두 고려한 차원수입니다.
- 따라서, Transformer 모델의 포지셔널 인코딩에서는 512차원의 벡터에 각 단어의 위치 정보를 결합하여, 단어의 의미와 위치 정보를 모두 학습할 수 있게 해줍니다.

이 공식을 통해, 시퀀스 내의 각 위치에 대해 사인 함수와 코사인 함수의 값을 계산하여 포지셔널 인코딩 벡터를 생성합니다.

3. 포지셔널 인코딩의 특성

1) 주기적 패턴

사인과 코사인 함수는 주기적으로 변화하므로, 위치 정보가 주기적으로 변하게 됩니다. 이로

인해, 위치가 서로 가까운 단어들은 유사한 포지셔널 인코딩 값을 가집니다.
이러한 주기성 덕분에 모델은 시퀀스 내의 상대적 위치 정보를 효과적으로 인식할 수 있습니다.
2) 다양한 주파수
포지셔널 인코딩에서 각 차원은 서로 다른 주파수를 사용합니다. 높은 차원의 포지셔널 인코딩은 짧은 거리에서 더 세밀한 변화를 제공하고, 낮은 차원에서는 긴 시퀀스에 대해 더 긴 범위의 변화를 제공합니다.
예를 들어, 차원 i가 커질수록 위치의 변화가 더 느려지고, 차원 i가 작을수록 더 세밀하게 위치를 구분할 수 있습니다.
3) 연속성
포지셔널 인코딩은 연속적인 위치 벡터를 생성하므로, 입력 시퀀스의 길이에 상관없이 적용할 수 있습니다. 시퀀스의 길이가 늘어나더라도 포지셔널 인코딩 값은 계속해서 규칙적으로 변동하며, 그에 따른 순서 정보를 제공하게 됩니다.
4) 확장성
포지셔널 인코딩은 주기적인 패턴을 따르기 때문에, 시퀀스의 길이가 임의적으로 늘어나더라도 자연스럽게 새로운 위치에 대한 정보를 제공할 수 있습니다. 사전 정의된 위치만 처리할 수 있는 것이 아니라, 시퀀스 길이가 길어지면 그에 맞는 위치 정보를 계속 생성할 수 있습니다.

4. 포지셔널 인코딩의 계산 예시
예시:
Transformer 모델에서 임베딩 차원이 512인 경우, 시퀀스 내의 단어 위치에 대한 포지셔널 인코딩 값은 아래의 공식을 통해 계산됩니다.
첫 번째 단어(위치 pos = 0)에 대한 포지셔널 인코딩:

$$PE_{(0,2i)} = \sin\left(\frac{0}{10000^{\frac{2i}{512}}}\right) = 0$$

$$PE_{(0,2i+1)} = \cos\left(\frac{0}{10000^{\frac{2i}{512}}}\right) = 1$$

첫 번째 단어의 포지셔널 인코딩은 차원이 짝수일 때는 0, 홀수일 때는 1이 됩니다.
두 번째 단어(위치 pos = 1)에 대한 포지셔널 인코딩

$$PE_{(1,2i)} = \sin\left(\frac{1}{10000^{\frac{2i}{512}}}\right)$$

$$PE_{(1,2i+1)} = \cos\left(\frac{1}{10000^{\frac{2i}{512}}}\right)$$

두 번째 단어는 차원에 따라 조금씩 다른 값을 가지게 됩니다.
차원이 높을수록 값의 변화가 더 느리며, 차원이 낮을수록 값의 변화가 더 큽니다.
포지셔널 인코딩 벡터
예를 들어, 시퀀스 내 첫 번째 단어의 포지셔널 인코딩 벡터는 [0, 1, 0, 1, 0, 1, ...]과 같은 형태를 띠며,
시퀀스 내 두 번째 단어의 포지셔널 인코딩 벡터는 [0.841, 0.540, 0.841, 0.540, ...]과 같은 형태로 계산됩니다.
이 값들은 모델의 임베딩 벡터에 더해져, 단어의 의미 정보와 위치 정보를 결합한 최종 입력 벡터가 됩니다.
5. 포지셔널 인코딩의 적용 방법
포지셔널 인코딩은 임베딩 벡터에 단순히 더해져 사용됩니다.
Transformer의 입력 단계에서 각 단어는 입력 임베딩(Input Embedding)을 통해 고차원 벡터로 변환되고, 여기에 해당 위치의 포지셔널 인코딩이 더해집니다.
Final Embedding =Word Embedding + Positional Encoding

이를 통해 모델은 단어의 의미와 위치 정보를 모두 포함하는 입력 벡터를 얻을 수 있으며, 이를 바탕으로 시퀀스 내에서 단어 간의 관계를 학습할 수 있게 됩니다.

6. 포지셔널 인코딩의 장점
- 병렬 처리 가능: Transformer는 시퀀스의 모든 단어를 동시에 처리하기 때문에, 순차적으로 데이터를 처리할 필요가 없습니다. 포지셔널 인코딩을 사용하여 단어의 순서를 유지하면서도 병렬 처리가 가능하게 됩니다.
- 확장성: 시퀀스의 길이가 길어지더라도 주기적인 패턴을 통해 위치 정보를 제공할 수 있으므로, 다양한 길이의 시퀀스를 효과적으로 처리할 수 있습니다.

위치 정보 보강

주5. 멀티헤드 어텐션(Multi-Head Attention

멀티헤드 어텐션(Multi-Head Attention)**은 Transformer 모델의 핵심 구성 요소 중 하나로, 어텐션 메커니즘을 확장한 방식입니다. 이는 여러 개의 **어텐션(Aggregate Attention)**을 동시에 수행함으로써 모델이 입력 시퀀스 내에서 다양한 관계를 학습할 수 있게 해줍니다. 단일 어텐션보다 더 풍부한 표현을 학습할 수 있도록 하는 것이 멀티헤드 어텐

션의 주요 목적입니다.
Transformer의 성능을 크게 높인 혁신적인 기법 중 하나로, 입력 시퀀스의 각 요소(단어 등)가 서로 어떻게 관계하는지 여러 개의 다른 어텐션 메커니즘을 통해 다양한 관점에서 학습합니다.
1. 멀티헤드 어텐션의 주요 개념
멀티헤드 어텐션의 핵심 아이디어는 단일 어텐션 메커니즘을 사용하는 대신, 여러 개의 어텐션을 병렬로 실행하는 것입니다. 각 어텐션 "헤드"는 입력 시퀀스 내에서 다른 관점을 학습합니다. 그 결과, 여러 어텐션을 통해 학습된 정보를 종합하여 더 풍부하고 다양한 관계를 학습할 수 있습니다.
멀티헤드 어텐션의 주요 과정:
쿼리(Query), 키(Key), 값(Value) 벡터를 각 헤드마다 서로 다른 가중치로 계산합니다.
각 어텐션 헤드는 독립적으로 어텐션을 수행하여, 입력 시퀀스에서 서로 다른 패턴이나 관계를 학습합니다.
각 헤드의 출력을 병합하고, 다시 한번 선형 변환을 통해 최종 출력을 계산합니다.
단일 어텐션과의 차이:
단일 어텐션은 하나의 어텐션 스코어(가중치)만을 계산하는 반면, 멀티헤드 어텐션은 여러 개의 어텐션을 동시에 적용하여 서로 다른 관계를 학습할 수 있습니다.
이를 통해, 각 단어가 시퀀스 내의 다른 단어들과 어떤 식으로 상호작용하는지 다양한 방식으로 학습할 수 있습니다. 예를 들어, 한 어텐션 헤드는 가까운 단어들의 관계를 학습하고, 다른 어텐션 헤드는 멀리 있는 단어들 간의 관계를 학습할 수 있습니다.
2. 멀티헤드 어텐션의 구조
멀티헤드 어텐션은 크게 입력 벡터의 분할, 어텐션 수행, 출력 병합 세 단계로 구성됩니다. Transformer에서 사용되는 쿼리(Query), 키(Key), **값(Value)**는 입력 시퀀스에서 각각 생성됩니다. 이를 각각의 헤드로 나누고, 각 헤드는 독립적으로 어텐션 연산을 수행합니다.
1) Query, Key, Value의 생성
쿼리(Query), 키(Key), **값(Value)**는 입력 시퀀스의 벡터에서 선형 변환을 통해 생성됩니다. 입력 데이터의 각 단어(벡터)는 가중치 행렬을 통해 세 가지 벡터로 변환됩니다.

$$\text{Query} = XW_Q,\ \ \text{Key} = XW_K,\ \ \text{Value} = XW_V$$

여기서 X는 입력 벡터, W_Q, W_K, W_V는 각각 쿼리, 키, 값 벡터를 만들기 위한 가중치 행렬입니다.
2) 어텐션 연산
각 헤드에서 Query, Key, Value 벡터를 이용해 Scaled Dot-Product Attention을 계산합니다. 이는 각 단어 간의 유사도를 측정하여 중요도를 평가하고, 그에 따라 값(Value) 벡터를 가중합하는 방식입니다.
어텐션 스코어(가중치)를 계산하는 공식은 다음과 같습니다:

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

Q는 쿼리 벡터, K는 키 벡터, V는 값 벡터,
d_k는 키 벡터의 차원수입니다. 나눈 값은 값을 정규화(scaling)하여 기울기 소실 문제를 줄이고 안정적으로 학습할 수 있도록 합니다.

3) 병렬 어텐션 실행
여러 개의 어텐션 헤드에서 각각 독립적으로 어텐션 연산을 수행합니다. 예를 들어, 8개의 어텐션 헤드가 있다면, 각 헤드가 서로 다른 관점에서 시퀀스 간의 관계를 학습하게 됩니다.
각 어텐션 헤드의 결과는 서로 다른 정보를 반영하며, 이를 통해 다양한 패턴을 동시에 학습할 수 있습니다.
4) 출력 병합
각 헤드에서 계산된 어텐션 값은 병합되고, 다시 한번 선형 변환을 통해 최종 출력을 계산합니다.

여기서 Concat은 각 어텐션 헤드의 출력을 결합하는 연산이며, W_O는 이 결합된 출력을 다시 선형 변환하는 가중치 행렬입니다. 이를 통해 여러 헤드에서 얻은 정보를 하나의 결과로 종합합니다.
3. 멀티헤드 어텐션의 수학적 표현
멀티헤드 어텐션은 여러 개의 **스케일드 닷 프로덕트 어텐션(Scaled Dot-Product Attention)**을 병렬로 수행하는 방식입니다. 각 헤드는 입력을 서로 다른 가중치로 변환하여 다양한 시각에서 시퀀스 내의 관계를 학습합니다.
멀티헤드 어텐션 공식:

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)W_O$$

각 head_i는 개별 어텐션 연산을 수행한 결과입니다.

$$\text{head}_i = \text{Attention}(QW_{Q_i}, KW_{K_i}, VW_{V_i})$$

여기서, **W_{Q_i}, W_{K_i}, W_{V_i}**는 각각 i번째 헤드에서 사용하는 가중치 행렬입니다.

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

Q(쿼리), K(키), V(값) 벡터가 스케일된 닷 프로덕트 연산을 통해 각 단어 간의 중요도를 계산한 뒤, 값(Value)을 가중합하여 최종 출력을 생성합니다.
설명:
Query는 현재 단어가 다른 단어들과 얼마나 관련이 있는지 측정합니다.
Key는 각 단어가 가진 정보의 특징을 나타냅니다.
Value는 실제로 출력에 반영될 단어의 값을 나타냅니다.
이 계산을 h번 수행하여 서로 다른 관점에서 시퀀스의 관계를 학습한 후, 각 결과를 병합하여 최종 출력을 만듭니다.

4. 멀티헤드 어텐션의 장점
1) 다양한 관점에서 관계 학습
여러 어텐션 헤드를 병렬로 사용함으로써, 시퀀스 내에서 단어들 간의 관계를 다양한 관점에서 학습할 수 있습니다. 예를 들어, 한 헤드는 가까운 단어 간의 관계를 학습하고, 다른 헤드는 더 먼 단어 간의 관계를 학습할 수 있습니다.
2) 정보의 손실 최소화
단일 어텐션에서는 한 번의 연산으로만 중요한 정보를 추출하기 때문에 특정 정보가 누락될 수 있습니다. 하지만 멀티헤드 어텐션에서는 각 헤드가 다른 방식으로 정보를 처리하므로, 다양한 정보가 더 풍부하게 반영될 수 있습니다.
3) 병렬 처리 가능
Transformer의 멀티헤드 어텐션은 병렬 처리가 가능하여, RNN이나 LSTM과 달리 긴 시퀀스를 처리할 때도 학습 속도가 매우 빠릅니다.
4) 기울기 소실 문제 해결

Self-Attention 메커니즘은 Transformer 모델의 핵심 구성 요소로, 문장 내에서 단어들이 서로 간에 얼마나 관련이 있는지를 학습하여, 특정 단어가 다른 단어와의 관계에서 얼마나 중요한지를 판단합니다. 이를 통해 문맥을 이해하고, 번역이나 텍스트 생성 같은 작업에서 중요한 역할을 합니다.
Self-Attention의 개념
Self-Attention은 문장의 각 단어가 다른 단어들과 얼마나 관련 있는지를 계산하는 메커니즘입니다. 이를 통해 모델은 각 단어가 다른 단어들에 대해 "얼마나 집중해야 하는지"를 결정합니다. 단어들이 문맥에서 중요한지, 덜 중요한지에 따라 각 단어의 가중치가 달라집니다.
사례 문장:
"The cat sat on the mat."
이 문장에서 Self-Attention이 어떻게 작동하는지를 단계적으로 설명하겠습니다.
1. 단어 임베딩
먼저 각 단어를 벡터로 변환합니다. 이 벡터는 단어의 의미를 나타내며, 모델에 입력되면 각

각의 단어가 고유한 벡터로 표현됩니다.
"The" -> 벡터1
"cat" -> 벡터2
"sat" -> 벡터3
"on" -> 벡터4
"the" -> 벡터5
"mat" -> 벡터6
2. Query, Key, Value 벡터 생성
문장의 각 단어는 Query (Q), Key (K), **Value (V)**라는 세 가지 벡터로 변환됩니다. 이 세 벡터는 각 단어의 의미적 관계를 학습하는 데 중요한 역할을 합니다.
Query (Q): 다른 단어들과의 유사성을 찾기 위한 벡터.
Key (K): 특정 단어와 관련된 정보를 나타내는 벡터.
Value (V): 실제로 참조할 정보를 담고 있는 벡터.
3. Query와 Key 간의 유사도 계산
각 단어의 Query와 다른 단어들의 Key 사이의 유사도를 계산합니다. 이 유사도는 두 단어가 문장 내에서 얼마나 관련이 있는지를 나타냅니다. 예를 들어, "cat"이라는 단어는 "sat"와 "mat"이라는 단어들과 더 관련이 있습니다. 따라서, "cat"과 "sat"의 유사도는 높고, "cat"과 "on"의 유사도는 상대적으로 낮을 수 있습니다.
유사도는 두 벡터 간의 **내적(dot product)**을 통해 계산됩니다.
4. Softmax로 가중치 계산
각 Query와 Key 사이의 유사도를 기반으로 Softmax 함수를 사용해 각 단어에 대한 가중치를 계산합니다. 이 가중치는 해당 단어가 다른 단어에 대해 얼마나 집중해야 하는지를 나타냅니다.
예를 들어, "cat"의 Query는 "sat"의 Key와 높은 유사도를 가지므로 "sat"에 높은 가중치를 부여하고, "on"에는 낮은 가중치를 부여할 수 있습니다.
5. 가중합 계산
계산된 가중치를 각 단어의 Value 벡터에 곱해 더한 값이 최종 Self-Attention 값이 됩니다. 이 값은 각 단어가 문맥에서 얼마나 중요한지에 대한 정보를 담고 있으며, 모델은 이 값을 이용해 다음 예측을 수행합니다.
예시:
"cat"의 Self-Attention 결과:
"cat" -> "sat"에 높은 가중치 (주어와 동사 간의 연관성)
"cat" -> "on", "the"에 낮은 가중치 (연관성 약함)
"mat"의 Self-Attention 결과:
"mat" -> "on"에 높은 가중치 (위치 관계)
"mat" -> "cat", "sat"에 낮은 가중치
Self-Attention의 핵심
Self-Attention의 핵심은 문장 내에서 각 단어가 서로에게 얼마나 중요한지를 학습하는 것입니다. 이 과정은 문맥을 이해하는 데 매우 중요하며, 특히 문장의 길이가 길거나 단어 간의 관계가 복잡할 때 매우 유용합니다. Transformer와 같은 모델이 문장의 의미를 더 정확하게 파악할 수 있는 이유도 Self-Attention 덕분입니다.

요약:
Self-Attention은 문장 내 각 단어가 서로 어떻게 연결되는지 학습합니다.
각 단어는 Query, Key, Value 벡터로 변환되며, Query와 Key 사이의 유사도를 계산해 가중치를 부여합니다.
이 가중치를 바탕으로 각 단어의 문맥적 중요도를 계산하여 모델이 더 정확한 예측을 할 수 있게 합니다.
Self-Attention은 텍스트 생성, 번역, 요약 등 많은 NLP 작업에서 Transformer 모델의 핵심 기법으로 사용됩니다.

주6. ADD와 Norm , Residual Connection(잔차 연결)

Transformer 모델에서 어텐션 메커니즘과 함께 사용되는 중요한 컴포넌트입니다. 이들은 모델이 안정적으로 학습하고 성능을 향상시키는 데 중요한 역할을 합니다.
Transformer에서 자주 사용되는 Residual Connection(잔차 연결)과 Layer Normalization을 함께 사용하는 것이 핵심입니다.
이들은 모델이 학습 중 기울기 소실 문제를 줄이고, 모델의 성능을 개선하는 데 기여합니다.

1. ADD (Residual Connection, 잔차 연결)
Residual Connection(잔차 연결)은 Transformer에서 Add로 표시되는 부분입니다.
이 구조는 모델이 깊어짐에 따라 발생할 수 있는 기울기 소실 문제를 해결하고, 모델의 학습 안정성을 높이기 위해 사용됩니다. Residual Connection은 입력과 출력을 직접 더하는 방식으로, 잔차(residual)를 네트워크에 추가합니다.

1) Residual Connection을 사용하는가?
Transformer는 깊은 네트워크 구조를 가지고 있어, 레이어가 많아지면 기울기 소실이나 기울기 폭발 같은 문제가 발생할 수 있습니다.
Residual Connection은 각 레이어의 입력을 해당 레이어의 출력에 더하여 입력 정보를 다음 레이어로 직접 전달합니다. 이를 통해 기울기 흐름이 원활해지며, 더 깊은 네트워크에서도 안정적인 학습이 가능해집니다.
ADD의 계산 과정:
입력 x가 있을 때, 해당 레이어의 출력 **f(x)**와 x를 더하는 방식으로 계산됩니다.

$$\text{Output} = f(x) + x$$

여기서 f(x)는 해당 레이어에서 계산된 출력이고, x는 이전 레이어의 입력입니다.
이를 더함으로써 입력 정보가 유지되면서 출력 정보도 반영됩니다.

Residual Connection의 효과
기울기 소실 문제 방지: 입력이 그대로 다음 레이어로 전달되기 때문에, 기울기가 더 안정적

으로 흐르며 깊은 네트워크에서도 학습이 가능합니다.
학습 안정성: 출력과 입력을 더하여 입력 정보가 모델의 다음 단계로 직접 전달되므로, 학습 과정이 더 안정적입니다.

잔차 연결(Residual Connection)
딥러닝 모델, 특히 딥 레이어 구조에서 학습의 효율성을 높이고, 기울기 소실(vanishing gradient) 문제를 완화하기 위해 사용되는 기법입니다.
이 개념은 ResNet(Residual Network)에서 처음 도입되었으며, 트랜스포머(Transformer) 모델에서도 각 층에서 사용됩니다. 잔차 연결은 입력을 그대로 출력에 더해주는 방식으로, 스킵 연결(skip connection)"이라고도 합니다.
잔차 연결의 주요 목표는 신경망이 깊어질수록 발생하는 학습 문제를 완화하는 것입니다. 이를 통해 신경망이 더 깊어져도 성능이 떨어지지 않고, 오히려 효율적으로 학습할 수 있습니다.
잔차 연결의 기본 개념
잔차 연결은 입력 x를 레이어(혹은 함수 $F(x)$)를 거친 출력에 더해주는 방식으로 작동합니다.
이때, 모델은 입력 자체와 함수로 변환된 입력을 모두 활용하여 다음 레이어로 전달합니다. 이를 통해 중요한 정보를 직접 전달할 수 있으며, 모델이 더 잘 학습할 수 있도록 돕습니다.
잔차 연결 수식
잔차 연결은 다음과 같이 정의됩니다:

$$\text{Output} = F(x) + x$$

x는 입력값입니다.
$F(x)$는 입력값을 처리한 결과(레이어의 출력)입니다.
Output은 잔차 연결을 적용한 결과로, 레이어의 출력
$F(x)$에 입력 x를 더한 값입니다.
즉, 신경망은 입력을 그대로 전달하는 경로와, 입력을 변환한 값을 더하는 경로를 동시에 학습하게 됩니다.
잔차 연결의 동작 방식
잔차 연결은 네트워크의 각 레이어가 잔차(residual)를 학습하게 만듭니다.
즉, 모델은 직접적인 출력보다는, 입력과 출력 간의 차이(잔차)를 학습하는 데 집중합니다.
이로 인해 학습이 더 쉬워지며, 네트워크가 깊어져도 학습 성능이 유지됩니다.
잔차 연결이 적용되지 않은 경우:
Output= $F(x)$
즉, 네트워크는 입력
x에서 $F(x)$만 학습해야 하므로, 정보 손실이나 기울기 소실 문제가 발생할 수 있습니다.
잔차 연결이 적용된 경우:
Output= $F(x)+x$
이 경우, 네트워크는

$F(x)$와 x를 함께 활용하므로, 학습이 훨씬 더 안정적이며, 정보 손실이 줄어듭니다.

잔차 연결의 역할

- 기울기 소실 문제 완화: 딥러닝 모델이 깊어질수록 발생할 수 있는 기울기 소실 문제를 완화합니다. 잔차 연결을 통해 입력 x가 출력까지 직접 전달되므로, 역전파를 할 때 기울기가 소실되지 않고 전달될 수 있습니다.
- 학습 속도 향상: 잔차 연결은 네트워크가 학습하는 패턴을 간단하게 만들어, 모델이 빠르게 수렴하도록 돕습니다. 입력과 출력 간의 차이만 학습하게 되어, 각 층이 더 쉽게 학습할 수 있습니다.
- 정보 전달: 잔차 연결은 입력 정보를 그대로 다음 층으로 직접 전달하므로, 중간 층에서 발생하는 정보 손실을 줄입니다. 이를 통해 더 깊은 신경망에서도 중요한 정보가 보존됩니다.

트랜스포머에서의 잔차 연결

트랜스포머(Transformer) 모델에서도 잔차 연결이 각 층에 적용됩니다.
트랜스포머는 여러 층의 Self-Attention과 피드포워드 네트워크(FFN)로 구성되며, 각 층에서 잔차 연결이 다음과 같이 적용됩니다.
Self-Attention 출력에 대한 잔차 연결:
z'=LayerNorm (z +Self-Attention(z))
여기서
z는 입력이고, Self-Attention 출력과 더한 후 Layer Normalization이 적용됩니다.

피드포워드 네트워크 출력에 대한 잔차 연결:

$$z_{l+1} = \mathrm{LayerNorm}(z' + \mathrm{FFN}(z'))$$

여기서
z' 는 Self-Attention을 통과한 값이며, 이 값에 피드포워드 네트워크의 출력을 더한 후 다시 Layer Normalization이 적용됩니다.

이를 통해, 트랜스포머는 각 층에서 입력 정보를 유지하면서도, 출력에 대한 학습을 안정적으로 수행할 수 있습니다.
잔차 연결의 장점 요약
깊은 네트워크에서 발생하는 문제 완화: 네트워크가 깊어지면 학습이 어려워지거나 기울기 소실 문제가 발생할 수 있는데, 잔차 연결은 이를 해결합니다.
더 빠른 학습: 모델이 학습할 때 더 간단한 패턴을 학습하도록 하여 수렴 속도를 높입니다.
정보 손실 방지: 중요한 정보가 손실되지 않고 층을 통해 전달될 수 있습니다.

2. Norm (Layer Normalization)

Layer Normalization(레이어 정규화)는 ADD(Residual Connection) 이후에 사용되는 단계로, 네트워크가 더 빠르고 안정적으로 학습할 수 있도록 돕습니다.
Norm은 모델이 각 레이어에서 출력되는 값들을 정규화하여 훈련 과정에서 값이 지나치게 커지거나 작아지지 않도록 하는 역할을 합니다.

Layer Normalization의 주요 목표:
딥러닝에서 학습 속도를 개선하고, 모델의 안정성을 높이기 위해 사용하는 정규화 기법 중 하나입니다. 이는 신경망의 각 층에서 입력값의 분포를 정규화하여, 학습이 원활하게 이루어지도록 도와줍니다.

- 모델 학습 속도 개선: 정규화를 통해 각 레이어의 출력이 일정한 범위 내에 유지되므로, 학습이 더 원활하게 이루어집니다.
- 학습 안정성 향상: 입력 데이터의 분포가 바뀌는 것을 방지하여, 과적합이나 기울기 폭발 같은 문제를 줄여줍니다.

Layer Normalization의 주요 개념
Layer Normalization은 신경망의 각 층의 입력을 정규화하여, 각 층의 출력이 일정한 분포를 가지도록 하는 방법입니다. 이는 주로 배치 크기에 상관없이 신경망의 학습을 더 안정적으로 만들기 위해 사용됩니다.
Layer Normalization 과정
주어진 신경망에서, 한 층의 입력은 다수의 뉴런을 통해 처리됩니다. 이때, 각 뉴런에 대한 입력을 평균과 분산으로 정규화합니다. Layer Normalization은 입력 데이터의 **특징 차원(feature dimension)**에 대해 평균과 분산을 계산하여 정규화합니다.
정규화 수식:
Layer Normalization에서, 주어진 입력
$h = [h1, h2, \ldots, hN]$ (입력 벡터 N개의 뉴런)에 대해, 각 입력 벡터 $h\,i$ 는 다음과 같이 정규화됩니다.

$$\hat{h}_i = \frac{h_i - \mu}{\sqrt{\sigma^2 + \epsilon}}$$

h i 는 입력값입니다. μ는 특징 차원의 평균

$$\mu = \frac{1}{N}\sum_{i=1}^{N} h_i$$

$\sigma 2$ 특징 차원의 분산

$$\sigma^2 = \frac{1}{N}\sum_{i=1}^{N}(h_i - \mu)^2$$

ϵ은 수치적인 안정성을 위해 작은 값(보통 1e-5을 더해줍니다. 이는 분모가 0이 되는 것을 방지하기 위함입니다.
$h\hat{}i$ 는 정규화된 값으로, 평균이 0이고 분산이 1로 조정된 값입니다.

가중치 및 편향 조정:
Layer Normalization은 단순한 정규화뿐 아니라, 각 정규화된 값에 대해 학습 가능한 가중치(scale, γ)와 편향(shift, β)을 도입합니다.
이를 통해 모델이 필요에 따라 데이터를 다시 스케일링하고 이동할 수 있습니다:

결국 Layer Normalization의 수식:

$$y_i = \gamma\hat{h}_i + \beta$$

$$\mathrm{LayerNorm}(x) = \frac{x - \mu}{\sigma + \epsilon} \cdot \gamma + \beta$$

x: 입력 벡터.
μ: 해당 레이어에서의 평균(mean).
σ: 해당 레이어에서의 표준편차(standard deviation).
γ, β: 학습 가능한 파라미터로, 정규화된 값을 조정하여 네트워크가 더 유연하게 학습할 수 있도록 합니다.
ε: 작은 값으로, 0으로 나누는 오류(divide by zero)를 방지하기 위한 안정화 파라미터입니다.

Layer Normalization의 역할:

- 안정적인 출력: 정규화된 값을 사용하므로, 각 레이어에서의 출력이 너무 크거나 작아지는 문제를 방지하여 안정적인 학습이 가능해집니다.
- 학습 속도 증가: 정규화를 통해 학습이 더 빠르게 수렴되며, 네트워크의 수렴 속도가 향상됩니다.

Layer Normalization의 동작 방식

- Layer Normalization은 각 층의 모든 뉴런에 대해 정규화를 수행합니다. 즉, 각 뉴런의 출력을 평균이 0이고 분산이 1인 상태로 조정합니다.
- 이는 모델이 배치(batch) 크기와 무관하게 동작할 수 있도록 하며, 특히 RNN, LSTM, 트랜스포머와 같은 순차적 데이터나 비동일한 배치 크기에서도 효과적으로 작동합니다.
- Batch Normalization이 배치 차원에서 정규화를 수행하는 반면, Layer Normalization은 특징 차원에서 정규화를 수행합니다.

- 배치 크기에 의존하지 않으므로, 온라인 학습이나 미니 배치 크기가 작은 상황에서도 안정적인 학습이 가능합니다.

Layer Normalization과 Batch Normalization의 차이

- Batch Normalization은 배치 단위로 입력을 정규화하지만, Layer Normalization은 각 레이어별로 정규화를 수행합니다.
- Layer Normalization은 주로 RNN, Transformer와 같이 배치 크기에 영향을 덜 받는 모델에서 효과적입니다.

특징	Batch Normalization	Layer Normalization
정규화 범위	**배치 차원(batch dimension)**에서 정규화	**특징 차원(feature dimension)**에서 정규화
배치 크기에 의존	배치 크기가 클수록 효과적 (작은 배치에서는 불안정할 수 있음)	배치 크기에 상관없이 작동, 작은 배치에서도 안정적
순차 모델에서의 사용	RNN, LSTM과 같은 순차 모델에서는 잘 사용되지 않음	RNN, LSTM, 트랜스포머와 같은 순차 모델에서 주로 사용
계산 비용	배치 크기에 따라 다름	상대적으로 낮음
사용 분야	CNN, 대규모 배치 훈련에 적합	RNN, 트랜스포머, 온라인 학습에 적합

3. ADD & NORM의 동작 순서

Transformer 모델에서 ADD(Residual Connection)과 NORM(Layer Normalization)은 항상 어텐션 레이어와 피드포워드 레이어의 출력에 적용됩니다. 이 순서대로 동작합니다:

어텐션 레이어 또는 피드포워드 레이어에서 출력을 계산합니다.

이 출력에 Residual Connection을 적용하여, 입력을 출력에 더해줍니다.

Layer Normalization을 적용하여, 더한 결과를 정규화합니다.

이 과정을 통해 각 레이어는 입력 정보를 유지하면서도 안정적으로 학습을 진행할 수 있게 됩니다.

[계산 순서]

- 어텐션 레이어의 출력: Attention_output = f(x)
- 잔차 연결: Residual_output = f(x) + x
- 정규화: Normalized_output = LayerNorm(Residual_output)

4. ADD & NORM의 중요성 요약

ADD (Residual Connection): 입력과 출력을 더해 모델이 더 깊어져도 학습이 안정적으로 이루어지도록 합니다. 이를 통해 기울기 소실 문제를 완화하고 효율적인 학습을 가능하게 합니다.

NORM (Layer Normalization): 레이어의 출력을 정규화하여 학습을 더 안정적이고 빠르게 만듭니다. 각 레이어에서 출력되는 값들이 너무 크거나 작아지는 문제를 방지하여, 모델의 성능을 개선합니다.

이 두 컴포넌트는 Transformer 모델의 성능 향상과 안정적 학습을 보장하는 데 매우 중요한 역할을 합니다. Residual Connection을 통해 네트워크가 깊어도 학습을 잘 할 수 있도록 하고, Layer Normalization을 통해 각 층에서의 출력 값이 정규화되어 학습 속도가 빨라지고 안정성이 향상됩니다.
결론
ADD(Residual Connection)와 Norm(Layer Normalization)은 Transformer 모델에서 안정적인 학습을 보장하고 성능을 최적화하는 중요한 역할을 합니다. Residual Connection은 기울기 소실 문제를 해결하고, Layer Normalization은 출력 값을 정규화하여 학습을 더 빠르고 효과적으로 만듭니다. 이들은 Transformer의 각 레이어에서 어텐션 메커니즘과 피드포워드 네트워크의 결과를 더 안정적으로 학습할 수 있게 해주며, 모델의 성능을 높이는 데 기여합니다.

주7. 피드포워드 네트워크(Feed-Forward Network:

피드포워드 네트워크(Feed-Forward Network, FFN)는 인공 신경망(Artificial Neural Network)의 기본 구성 요소 중 하나로, 데이터를 순차적으로 입력에서 출력으로 전달하는 방식으로 작동하는 네트워크입니다. 특히 Transformer 모델에서 피드포워드 네트워크는 어텐션 메커니즘 뒤에 위치해, 어텐션 레이어가 처리한 정보를 비선형적으로 변환하여 더 복잡한 특징을 학습하는 역할을 합니다.
이를 통해 어텐션 메커니즘이 학습한 관계 정보를 바탕으로 더 깊이 있는 패턴을 학습하게 됩니다.
1. 피드포워드 네트워크의 구조
Transformer에서 사용하는 피드포워드 네트워크는 두 개의 완전 연결층(fully connected layers)과 활성화 함수로 구성됩니다. Transformer에서 FFN은 시점별 위치마다 독립적으로 적용되며, 각 입력이 동일한 파라미터로 처리됩니다.
FFN의 구성:
첫 번째 완전 연결층:
입력 벡터를 더 큰 차원의 벡터로 확장합니다. 입력 차원은 d_model(임베딩 차원)이고, 확장된 차원은 일반적으로 더 큽니다(예: 2048차원).
활성화 함수:
활성화 함수로는 주로 ReLU(Rectified Linear Unit)가 사용됩니다. ReLU는 비선형성을 추가하여 신경망이 더 복잡한 패턴을 학습할 수 있도록 돕습니다.
두 번째 완전 연결층:
확장된 벡터를 다시 원래 차원으로 축소합니다. 입력 벡터 차원 d_model로 돌아가게 됩니다.
피드포워드 네트워크의 계산:
여기서 x는 입력 벡터, W_1과 W_2는 가중치 행렬, b_1과 b_2는 편향(bias)입니다.
max(0, x)는 ReLU 활성화 함수로, 음수를 0으로 만들어 비선형성을 부여합니다.

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

2. 피드포워드 네트워크의 동작 원리

Transformer에서 피드포워드 네트워크(FFN)는 각 어텐션 레이어 뒤에 위치하여 어텐션 메커니즘이 학습한 정보에 대해 더 복잡한 변환을 수행합니다. 이 단계에서 비선형 변환을 통해 모델이 더 깊고 복잡한 패턴을 학습할 수 있게 됩니다.

FFN의 역할:

비선형 변환: 어텐션 레이어의 출력은 선형 변환만 포함되어 있어, 이를 보완하기 위해 FFN은 ReLU와 같은 활성화 함수를 적용하여 비선형성을 추가합니다.

특징 추출: FFN은 어텐션에서 학습한 단어 간 관계 정보를 더 정교하게 가공하여, 입력 시퀀스에서 더 복잡한 특징을 추출합니다.

레이어 간 깊이 추가: 두 개의 완전 연결층은 입력 데이터의 복잡한 패턴을 학습할 수 있도록 깊이를 더해줍니다. 이는 단순한 선형 연산 이상의 패턴을 학습하는 데 중요합니다.

3. 피드포워드 네트워크의 역할 및 중요성

Transformer에서 피드포워드 네트워크는 어텐션 메커니즘과 함께 단어 간 관계를 학습할 뿐만 아니라, 입력 데이터에서 더 높은 수준의 추상적인 패턴을 학습하는 데 중요한 역할을 합니다. 이 네트워크가 없다면 모델은 단순한 선형 관계만 학습할 수 있으며, 복잡한 데이터의 패턴을 이해하기 어려워집니다.

FFN의 주요 역할:

- 비선형성 추가:
- ReLU와 같은 활성화 함수는 비선형성을 추가하여 모델이 단순한 선형 패턴을 넘어, 더 복잡한 관계를 학습할 수 있게 합니다.
- 입력 변환:
- FFN은 입력 벡터를 변환하여, 어텐션 메커니즘에서 학습된 정보를 보완하고 더 정교한 패턴을 추출합니다. 즉, 어텐션이 학습한 관계를 기반으로 새로운 표현을 만들 수 있도록 합니다.
- 어텐션 출력 후처리:

어텐션 레이어는 입력 시퀀스의 각 요소 간의 관계를 학습하지만, 그 자체로는 단어 간의 상호작용만을 고려합니다. FFN은 이 상호작용을 더욱 복잡하게 가공하여 최종 출력에 반영합니다.

4. 피드포워드 네트워크의 적용 예시

Transformer 모델의 각 레이어는 멀티헤드 어텐션 뒤에 피드포워드 네트워크를 적용합니다. 예를 들어, 입력 문장이 주어지면, 각 단어는 어텐션 레이어를 통해 다른 단어들과의 관계를 학습한 뒤, FFN을 통해 복잡한 변환을 거쳐 최종적으로 모델 출력에 반영됩니다.

Transformer의 레이어 내 FFN 적용 흐름:

- 입력 임베딩: 입력 시퀀스가 모델로 전달되어 단어 벡터로 변환됩니다.
- 멀티헤드 어텐션: 입력 시퀀스 간의 관계를 학습합니다.
- 피드포워드 네트워크: 어텐션 레이어의 출력을 변환하고, 추가적으로 비선형성을 부여하여 복잡한 패턴을 학습합니다.
- 출력: 최종 출력으로 연결되며, 이는 예측이나 변환 작업에 사용됩니다.

5. 피드포워드 네트워크의 특징

- 위치 독립적: 피드포워드 네트워크는 입력 시퀀스 내 각 위치에 대해 독립적으로 적용되므

로, 각 단어가 별도로 처리되며 위치 정보에 의존하지 않습니다.

- 확장성: FFN은 단일 층이 아니라, 여러 층을 쌓아서 더 복잡한 패턴을 학습할 수 있습니다. Transformer에서는 기본적으로 두 개의 레이어로 구성되어 있지만, 필요에 따라 더 깊은 네트워크로 확장할 수 있습니다.
- 복잡한 관계 학습: 어텐션 메커니즘에서 학습된 관계를 더 복잡한 방식으로 변환할 수 있어, 단순한 어텐션 연산 이상의 패턴을 학습할 수 있습니다.

6. 피드포워드 네트워크의 장점

- 높은 유연성: 입력과 출력 간의 비선형 변환을 통해 더 유연하게 다양한 패턴을 학습할 수 있습니다.
- 병렬 처리 가능: FFN은 입력 시퀀스의 각 단어에 대해 독립적으로 처리되므로, 병렬 처리가 가능합니다. 이는 RNN 계열 모델과 비교해 학습 속도를 크게 향상시킵니다.
- 비선형성 제공: 활성화 함수인 ReLU가 비선형성을 추가하여, 단순한 선형 모델보다 더 복잡한 데이터 표현을 학습할 수 있게 도와줍니다.

주9. Masked Multi-Head Attention

Masked Multi-Head Attention은 Transformer 모델에서 디코더(Decoder)의 자기 어텐션(Self-Attention) 단계에서 사용되는 중요한 개념입니다. Masked Multi-Head Attention은 다음에 나올 단어를 미리 보지 못하게 하는 제약을 가해 시퀀스 생성을 더 효과적으로 만들기 위한 기법입니다.

이 기법은 주로 텍스트 생성 작업(예: 번역, 요약)에서 사용되며, 디코더가 현재 시점에서 앞으로 생성해야 할 단어를 보지 않고 학습하도록 하는 데 필수적입니다.

1. Masked Multi-Head Attention이 필요한 이유

Transformer의 디코더는 시퀀스 생성 모델로 작동하기 때문에, 예측할 때 미래의 정보를 참조해서는 안 됩니다.

예를 들어, 디코더가 문장의 첫 번째 단어를 예측할 때, 아직 예측하지 않은 두 번째, 세 번째 단어의 정보를 미리 알게 된다면, 그 과정이 학습에 부정적인 영향을 미칩니다.

따라서, 현재 시점까지의 정보만을 기반으로 단어를 생성해야 합니다.

Masked Multi-Head Attention의 역할:

디코더가 이전 단어들만을 보고 다음 단어를 예측하도록 제한합니다.

이를 통해 순차적인 단어 생성이 가능하게 되고, 미래 정보에 의존하지 않도록 합니다.

2. Masked Multi-Head Attention의 작동 원리

Masked Multi-Head Attention은 Self-Attention(자기 어텐션)의 일종입니다. 기본적인 Self-Attention에서는 입력 시퀀스 내의 모든 단어가 서로 어텐션 연산을 통해 관계를 학습합니다. 하지만 Masked Multi-Head Attention에서는 시퀀스의 각 시점에서 이후 시점에 있는 단어들에 대해 가중치 값을 0으로 처리하여 모델이 해당 정보를 참조하지 못하게 만듭니다.

기본 Self-Attention 연산:

쿼리(Query), 키(Key), 값(Value) 벡터가 입력 시퀀스의 각 단어로부터 생성됩니다.

쿼리와 키의 내적(dot product)을 통해 어텐션 스코어를 계산하고, 이를 **값(Value)**에 가

중합하여 최종 출력으로 사용합니다.
Masked Multi-Head Attention 연산:
Masked Multi-Head Attention에서는 미래 시점의 단어들을 참조하지 못하도록, 어텐션 스코어를 계산할 때 **마스크(Mask)**를 적용하여 미래 단어에 대한 가중치가 0이 되도록 설정합니다.
이를 통해 현재 시점의 단어는 자신보다 앞에 있는 단어들만 참조할 수 있습니다.
3. Masking의 작동 방식
마스킹(Masking)은 어텐션 스코어 행렬에 적용되어 특정 시점에서 참조할 수 없는 단어들에 대한 가중치를 0으로 만드는 과정입니다.
이 과정에서, 모델은 현재 시점과 이전 시점에 해당하는 정보만을 이용하고, 미래 시점의 정보는 무시하게 됩니다.
Masking 과정:
어텐션 스코어 행렬을 계산하는 과정에서, 각 단어의 쿼리 벡터와 모든 키 벡터 간의 내적을 구해 어텐션 스코어를 얻습니다.
마스킹 행렬(mask matrix)을 생성하여 미래 시점에 해당하는 요소를 무한대의 작은 값(일반적으로 매우 작은 음수 값, 예: -∞)으로 설정합니다. 이는 소프트맥스 함수에 적용했을 때 해당 값이 0에 가깝게 되어, 해당 단어들에 대한 가중치가 0으로 처리되도록 합니다.

Masking 적용 수식:
Masked Multi-Head Attention에서의 어텐션 스코어는 다음과 같이 계산됩니다:

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

여기서 M은 마스크 행렬로, 미래 시점에 해당하는 값들을 매우 작은 값(예: -∞)으로 설정하여 소프트맥스 함수에서 해당 값들이 0으로 변환되도록 합니다.
이렇게 함으로써, 현재 시점의 단어는 미래 시점의 단어에 대한 정보를 참조하지 않고, 이전 단어들만을 기반으로 어텐션을 수행합니다.
마스크 행렬(M):
마스크 행렬은 **상삼각 행렬(Upper Triangular Matrix)**로 구성됩니다. 이 행렬은 대각선 위쪽에 있는 값을 **무한대의 작은 값(-∞)**으로 설정하여, 해당 값들이 소프트맥스에서 무시되도록 합니다.
예를 들어, 길이가 5인 시퀀스에 대한 마스크 행렬은 다음과 같습니다:

이 마스크 행렬을 적용하면, 첫 번째 단어는 이후의 단어들을 볼 수 없고, 두 번째 단어는 첫 번째 단어만을 참조하며, 세 번째 단어는 첫 번째와 두 번째 단어만 참조하게 됩니다.

$$M = \begin{pmatrix} 0 & -\infty & -\infty & -\infty & -\infty \\ 0 & 0 & -\infty & -\infty & -\infty \\ 0 & 0 & 0 & -\infty & -\infty \\ 0 & 0 & 0 & 0 & -\infty \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

4. Masked Multi-Head Attention의 계산 과정

단계별 계산:

쿼리(Query), 키(Key), 값(Value) 벡터를 생성:

입력 시퀀스의 각 단어에 대해 쿼리, 키, 값 벡터가 생성됩니다.

이 벡터들은 가중치 행렬을 통해 계산되며, 각 벡터는 어텐션 계산에 사용됩니다.

어텐션 스코어 계산:

각 쿼리와 키 벡터 간의 내적을 통해 어텐션 스코어가 계산됩니다. 이때 마스크 행렬을 적용하여 미래 시점의 단어에 대한 가중치를 0으로 설정합니다.

소프트맥스 적용:

마스크된 어텐션 스코어 행렬에 소프트맥스 함수를 적용하여, 어텐션 가중치가 계산됩니다. 이때 마스크된 값은 매우 작은 값이므로, 소프트맥스에서 해당 값에 대한 가중치는 0이 됩니다.

값(Value) 벡터 가중합:

계산된 어텐션 가중치를 값(Value) 벡터에 곱하여 최종 출력이 생성됩니다.

5. Masked Multi-Head Attention의 역할

순차적 단어 생성: Masked Multi-Head Attention은 다음 단어를 예측하는 디코더에서 중요한 역할을 합니다. 각 단어를 예측할 때, 이전 시점의 단어들만을 참조하여 순차적으로 단어를 생성할 수 있게 합니다.

미래 정보 차단: 모델이 미래 단어에 접근하지 못하도록 하여, 학습 시 부적절한 정보 유출을 방지합니다.

올바른 시퀀스 예측: Masked Multi-Head Attention을 통해 모델이 올바른 순서로 단어를 예측할 수 있게 하여, 텍스트 생성 문제에서 일관된 예측이 가능하게 합니다.

6. Masked Multi-Head Attention의 예시

Transformer 모델에서 디코더의 Masked Multi-Head Attention은 다음과 같이 정의됩니다:

- 2017년 구글이 발표한 "Attention is All You Need" 논문 이후, Transformer 모델을 기반으로 다양한 연구가 진행되었고, 어텐션 메커니즘은 자연어 처리(NLP)를 비롯한 여러 분야에서 중요한 역할을 하게 되었습니다.
- 특히 Transformer의 발전은 딥러닝의 새로운 표준으로 자리 잡았으며, 이를 기반으로 한 많은 모델들이 발표되었고, 성능과 효율성을 향상시키기 위한 다양한 개선이 이루어졌습니다. 아래는 구글 어텐션 논문 이후의 주요 발전 사항입니다.

1. BERT (2018)

BERT (Bidirectional Encoder Representations from Transformers)는 구글이 2018년에 발표한 모델로, Transformer의 인코더 구조만을 활용한 모델입니다. BERT는 양방향으로 문맥을 이해할 수 있는 모델로, 이전의 순차적인 처리 방식과 달리 모든 단어를 양방향으로 학습합니다.

주요 특징:

- Masked Language Model(MLM): 입력 텍스트에서 일부 단어를 가리고, 이 가려진 단어들을 예측하는 방식으로 사전 훈련이 이루어집니다. 이를 통해 BERT는 양방향 문맥을 고려할 수 있습니다.
- Next Sentence Prediction(NSP): 두 문장이 주어졌을 때, 두 문장이 이어지는 문장인지 여부를 예측하는 방식으로 훈련됩니다. 이는 문장 간 관계를 학습하게 합니다.

영향:

BERT는 질의 응답, 텍스트 분류, 텍스트 요약 등 다양한 NLP 작업에서 뛰어난 성능을 보였습니다.

이후 BERT를 개선한 다양한 변형 모델들이 등장했습니다.

2. GPT 시리즈 (2018 - 2023)

- GPT (Generative Pre-trained Transformer) 시리즈는 OpenAI에서 개발한 Transformer 디코더 기반의 언어 생성 모델입니다.
- GPT는 텍스트 생성 작업에 뛰어난 성능을 보이며, 모델 크기를 계속해서 확장해 나가면서 자연어 생성 작업의 성능을 혁신적으로 높였습니다.

▷ GPT-1 (2018)

Transformer 디코더 아키텍처를 사용하여, 주로 언어 생성에 집중한 모델입니다.

사전 학습(pre-training)과 미세 조정(fine-tuning)으로 텍스트 생성과 다른 NLP 작업에서 우수한 성능을 보여줬습니다.

사전 학습(pre-training)과 미세 조정(fine-tuning)은 인공지능 모델, 특히 딥러닝 모델을 학습하는 데 사용하는 중요한 두 단계입니다. 이 두 과정은 모델이 더 나은 성능을 발휘하도록 돕는 전이 학습(transfer learning)[10]의 핵심 요소입니다.

****사전 학습 (Pre-training)**

- 사전 학습은 대규모의 일반 데이터셋을 사용하여 모델이 기본적인 패턴과 언어 구조를 학습하도록 하는 단계입니다.
- 주로 언어 모델이나 이미지 모델에서 이뤄지며, 이 과정에서 모델은 특정 작업에 최적화되기보다는 일반적인 지식을 학습합니다.

예: 텍스트 모델의 경우, 대규모 텍스트 데이터셋(예: 위키피디아, 뉴스 기사)에서 모델이 언어 구조, 어휘, 문법 등을 학습하는 방식입니다. 이미지 모델의 경우, 여러 객체를 포함한 이미지에서 사전 학습을 수행해 다양한 패턴을 인식하도록 합니다.

목적: 특정 작업에 맞춘 데이터가 부족할 때, 일반적인 지식을 학습함으로써 모델이 다양한 작업에 쉽게 적응할 수 있는 기초를 다지는 것입니다. 이렇게 학습된 모델은 여러 분야

10)

에 적용할 수 있습니다.
방법: 사전 학습 단계에서는 언어 모델의 경우 다음 단어 예측, 마스크된 단어 예측 등 자기지도학습(self-supervised learning)[11] 학습 방식이 사용됩니다.
이미지 모델의 경우는 객체 분류 등의 기본적인 이미지 분류 태스크로 학습합니다.
**** 미세 조정 (Fine-tuning)**
미세 조정은 사전 학습된 모델을 특정한 작업에 맞춰 재학습하는 과정입니다.
이 과정에서는 특정 작업에 맞는 소량의 레이블이 있는 데이터를 사용해 모델의 성능을 높입니다.
예: 사전 학습된 언어 모델이 특정 도메인(예: 의료, 법률)에서 문서를 분류해야 한다면, 해당 도메인의 데이터로 모델을 미세 조정합니다.
이미지 분류 모델이 특정 클래스 분류에 적합하도록 미세 조정하는 경우도 있습니다.
목적: 사전 학습된 모델을 특정 작업의 특성에 맞춰 성능을 높이기 위함입니다.
이를 통해 모델은 일반적인 지식에 더해, 특정 도메인의 세부적인 특징이나 요구 사항을 반영할 수 있습니다.
방법: 미세 조정 단계에서는 특정 작업에 대한 데이터(예: 감정 분류, 특정 객체 탐지)를 모델에 제공하여 모델이 해당 작업에 맞게 가중치를 조정하도록 합니다.
사전 학습으로 기본적인 지식은 이미 학습되었기 때문에, 비교적 적은 데이터로도 효율적인 학습이 가능합니다.
사전 학습과 미세 조정의 장점과 차이점

구분	사전 학습 (Pre-training)	미세 조정 (Fine-tuning)
목적	일반적인 패턴과 구조 학습	특정 작업의 특성 학습
데이터	대규모의 일반적인 데이터	소규모의 특정 작업 관련 데이터
학습 방식	자가 학습 방식으로 패턴 학습	지도 학습 방식으로 특정 작업에 최적화
주요 장점	다양한 작업에 적응할 수 있는 지식 제공	특정 작업에서의 성능 향상
주요 예시	BERT, GPT 등의 언어 모델, 이미지넷을 통한 이미지 모델	사전 학습된 BERT를 뉴스 감정 분석에 활용, 특정 이미지 분류

두 단계는 결합하여 모델의 성능을 높이고, 학습 시간을 줄이며, 데이터가 부족한 상황에서도 강력한 모델을 만들기 위한 중요한 방법으로 사용됩니다.

▷ GPT-2 (2019)
GPT-2는 크기와 성능이 크게 확장된 모델로, 15억 개의 매개변수를 가진 모델입니다.
텍스트 생성 작업에서 매우 긴 텍스트도 자연스럽게 생성할 수 있게 되면서 주목받았습니다.
제로 샷 학습을 통해 미리 정의된 작업에 대해 별도의 학습 없이도 좋은 성능을 보여줬습니다.
▷ GPT-3 (2020)
GPT-3는 1750억 개의 매개변수를 가진 대규모 모델로, NLP 분야에서 강력한 언어 생성 능력을 보여주며 큰 주목을 받았습니다.

11)

사전 학습된 모델을 Few-Shot, One-Shot, Zero-Shot 방식으로 사용할 수 있어, 특정 작업에 대한 학습 없이도 다양한 작업에서 뛰어난 성능을 발휘할 수 있었습니다.
GPT-4가 2023년 출시되었고, 더 나은 자연어 이해와 생성 능력을 보여주었습니다.

3. T5 (2019)
T5 (Text-to-Text Transfer Transformer)는 구글이 발표한 또 다른 혁신적인 모델로, 모든 NLP 문제를 텍스트 변환 문제로 처리하는 방식입니다. 즉, 입력과 출력 모두를 텍스트 형식으로 취급하여 다양한 NLP 작업을 통합적으로 다룰 수 있는 모델입니다.
주요 특징:
텍스트 입력과 텍스트 출력: 모든 작업을 텍스트 입력과 텍스트 출력으로 처리하여 다양한 NLP 작업을 통일된 형식으로 다룰 수 있습니다.
범용 NLP 모델: T5는 다양한 작업을 단일 모델로 처리할 수 있도록 설계되었으며, 번역, 질의 응답, 문장 완성 등 다양한 작업에서 우수한 성능을 보여줍니다.
영향:
다양한 NLP 작업을 단일 프레임워크에서 처리할 수 있게 했으며, 작업마다 별도의 모델을 설계할 필요 없이 통합된 방식으로 접근할 수 있게 했습니다.

4. ALBERT (2019)
ALBERT (A Lite BERT)는 BERT의 경량화 버전으로, 모델의 크기를 줄이면서 성능을 유지하는 데 중점을 둔 모델입니다.
주요 특징:
매개변수 공유: 각 레이어에서 동일한 가중치를 공유하여 모델의 크기를 줄였습니다.
사전 학습 개선: BERT의 사전 학습 방법을 개선하여, 더 적은 데이터로도 더 나은 성능을 발휘할 수 있도록 했습니다.
영향:
효율성과 성능을 균형 있게 유지하면서, 다양한 NLP 작업에서 BERT와 유사한 성능을 보여줍니다.

5. ELECTRA (2020)
ELECTRA는 사전 학습 모델로, 텍스트 내에서 잘못된 단어를 찾아내는 방식으로 학습하는 새로운 접근 방식을 제안했습니다. 기존의 Masked Language Model(MLM) 방식을 개선하여 더 효율적으로 학습할 수 있도록 설계되었습니다.
주요 특징:
생성기(Generator)와 판별기(Discriminator) 구조: ELECTRA는 입력 텍스트에서 일부 단어를 변경하고, 판별기가 이 변경된 단어를 찾아내도록 학습합니다.
효율적인 학습: ELECTRA는 전체 텍스트를 더 빠르게 학습할 수 있어, BERT와 비교해 더 적은 계산 비용으로 비슷한 성능을 발휘합니다.
영향:
기존의 Masked Language Model 기반 학습보다 효율적인 학습 방식을 제공하며, 적은 자원으로도 뛰어난 성능을 보여줍니다.

6. BIG-bench (2021)

BIG-bench는 구글에서 주도한 대규모 벤치마크 프로젝트로, 인공지능의 이해와 생성 능력을 측정하는 다양한 작업을 포함하고 있습니다. 이 프로젝트는 대규모 사전 학습된 언어 모델이 얼마나 다양한 작업에서 성능을 발휘하는지를 평가하기 위한 목표로 진행되었습니다.

주요 특징:

다양한 과제: 여러 학문 분야에서 AI가 얼마나 잘 학습하고 추론할 수 있는지 테스트할 수 있는 다양한 과제들을 포함하고 있습니다.

대규모 AI 모델 평가: GPT-3와 같은 대규모 언어 모델들의 능력을 측정하는 기준이 되었습니다.

7. GPT-4 (2023)

GPT-4는 OpenAI의 최신 언어 모델로, GPT-3의 성공을 기반으로 성능과 효율성이 대폭 향상된 버전입니다.

이 모델은 멀티모달 (Multimodal) 입력을 지원하며, 텍스트뿐만 아니라 이미지에 대해서도 처리할 수 있습니다.

주요 특징:

- 멀티모달 처리: GPT-4는 텍스트와 이미지 입력을 동시에 처리할 수 있는 기능을 도입하여 더 많은 종류의 데이터를 이해하고 생성할 수 있습니다.
- 성능 향상: GPT-4는 GPT-3보다 더 뛰어난 언어 이해와 생성 능력을 발휘하며, 다양한 작업에서 인간 수준의 성능을 보여줍니다.

멀티모달(Multimodal)

- 인공지능(AI)에서 텍스트, 이미지, 음성, 비디오 등의 여러 데이터 형태(모달리티)를 동시에 사용하여 모델이 더 풍부한 정보를 처리하고 이해할 수 있도록 하는 기술입니다.
- 멀티모달 AI는 서로 다른 모달리티 간의 상호작용을 학습하여, 개별 모달리티에서 얻을 수 있는 정보를 결합하여 더욱 강력하고 유연한 모델을 구축합니다.

멀티모달의 구성 요소와 예시

텍스트-이미지 결합

예: 이미지 캡셔닝(Image Captioning)

이미지를 보고 내용을 텍스트로 설명하는 AI입니다. 예를 들어, “강아지가 잔디밭 위에서 공을 물고 있다”와 같이 이미지 속 내용을 텍스트로 표현합니다.

활용: 자동 이미지 설명 생성, 시각장애인을 위한 정보 전달 서비스 등

텍스트-음성 결합

예: 음성 비서, 자동 자막 생성

텍스트 명령을 음성으로 변환하거나, 음성을 텍스트로 변환해 상황에 맞는 답변을 생성하는 방식입니다.

활용: 고객 서비스 봇, 실시간 번역, 음성 기반 검색

이미지-텍스트-음성 결합

예: 비디오 분석 및 설명 생성

비디오에 나타나는 장면과 음성을 결합해 내용을 분석하고, 특정 장면을 텍스트로 설명합니다.

활용: 비디오 캡셔닝, 영상 속 특정 이벤트 탐지
텍스트-이미지-음성-센서 데이터 결합
예: 자율 주행 차량
자율 주행에서는 카메라의 영상, 라이다 센서 데이터, 주변의 소리, 차량 내부와 외부에서 수집된 다양한 센서 데이터를 종합하여 주행을 결정합니다.
활용: 자율 주행, 로봇 공학, 지능형 IoT 장비
멀티모달 모델의 주요 기술
멀티모달 AI에서는 서로 다른 데이터 모달리티 간의 상관관계를 학습하기 위해 다음과 같은 기술이 활용됩니다.

▷ 멀티모달 임베딩: 각 모달리티의 데이터를 벡터 공간에 표현하여, 다른 모달리티 데이터와 상호작용할 수 있도록 합니다. 이를 통해 텍스트와 이미지가 같은 의미 공간에서 매핑됩니다.
▷ 어텐션 메커니즘: 이미지와 텍스트 등의 모달리티 간 중요한 정보를 선택적으로 반영하도록 도와줍니다. 예를 들어, 특정 이미지 부분이 어떤 텍스트와 관련이 있는지를 학습합니다.
▷ 트랜스포머 모델: 트랜스포머 기반 모델은 멀티모달 학습에 적합하며, 특히 텍스트와 이미지와 같은 모달리티의 상호작용을 효율적으로 학습할 수 있습니다. GPT-4, CLIP, DALL-E 같은 모델이 여기에 해당됩니다.

멀티모달 AI의 장점과 도전 과제
장점

- 복합적인 정보 파악으로 정교하고 직관적인 응답 생성이 가능합니다.
- 이미지나 텍스트 하나만으로는 불가능한 복잡한 작업(예: 영상 설명 생성)에서 강력한 성능을 발휘합니다.

도전 과제

- 데이터 처리 및 학습 비용이 크며, 다양한 모달리티 간의 정보 결합과 학습이 복잡합니다.
- 각 모달리티가 다른 성격과 구조를 가지기 때문에, 이를 공통된 표현으로 통합하는 것이 어렵습니다.
- 멀티모달 기술은 시각, 청각, 텍스트 기반 인식 및 생성에서의 가능성을 넓히며, AI가 사람처럼 다양한 정보를 통합해 인지하고 상호작용할 수 있게 하는 방향으로 발전하고 있습니다.

8. Vision Transformers (ViT, 2020)
Vision Transformers (ViT)는 Transformer 모델을 컴퓨터 비전 분야에 적용한 모델로, 이미지 데이터를 처리하기 위해 이미지를 패치로 나누어 Transformer를 적용한 방식입니다.
주요 특징:
이미지를 패치로 분할: 이미지의 각 패치를 시퀀스로 취급하여 Transformer로 처리하는 방식으로, 기존 CNN 기반의 이미지 처리 모델보다 더 효율적이고 유연한 접근 방식을 제시했습니다.
성능: ViT는 대규모 데이터에서 기존 CNN을 능가하는 성능을 보였으며, 이후 여러 비전 작

업에 Transformer가 응용되었습니다.

주8. Self-Attention 메커니즘

Self-Attention 메커니즘은 Transformer 모델의 핵심 구성 요소로, 문장 내 모든 단어가 서로 상호작용하고, 각 단어가 문맥에서 어떻게 중요한지 판단할 수 있도록 하는 방법입니다.
Self-Attention은 입력 시퀀스에서 각 단어가 다른 단어들과 어떻게 관련되는지를 학습하고, 이를 바탕으로 더 정확하게 텍스트를 처리합니다.
이 메커니즘은 문맥을 잘 이해하고, 병렬 처리가 가능하여 Transformer 모델에서 중요한 역할을 합니다.
Self-Attention은 특히 긴 문장이나 복잡한 문맥을 처리하는 데 매우 유용하며, 단순한 순차적 처리보다 병렬 처리가 가능하므로 효율적입니다.

1. Self-Attention의 기본 아이디어
Self-Attention의 기본 아이디어는, 문장의 각 단어가 자신 이외의 다른 단어들과의 관련성을 계산하고, 이를 바탕으로 중요한 단어에 더 집중할 수 있도록 가중치를 부여하는 것입니다.
(예시) "The cat sat on the mat"라는 문장에서 "sat"이라는 단어는 "cat"과 밀접한 관련이 있지만, "mat"과는 조금 더 적은 관련이 있을 수 있습니다.
이때, Self-Attention 메커니즘은 이런 단어 간의 중요도를 계산하여, 각 단어가 문맥 내에서 다른 단어들과 얼마나 중요한지를 반영할 수 있게 합니다.

2. Self-Attention의 계산 과정
Self-Attention 메커니즘은 크게 세 가지 벡터인 Query (질의), Key (키), Value (값) 벡터를 사용해 단어 간의 유사성을 계산합니다. 이 과정은 다음과 같이 이루어집니다.
1) Query, Key, Value 벡터 생성
입력된 각 단어는 세 가지 벡터로 변환됩니다:
- Query (질의): 해당 단어가 다른 단어에 대해 질문하는 벡터.
- Key (키): 해당 단어가 다른 단어에 답변하는 벡터.
- Value (값): 해당 단어 자체의 정보를 담은 벡터.

이 세 가지 벡터는 입력 벡터를 학습된 가중치 행렬로 변환하는 과정을 통해 얻어집니다.
입력 벡터
X가 주어지면:

$$Q = XW_Q, \quad K = XW_K, \quad V = XW_V$$

여기서 W_Q, W_K, W_V는 각각 Query, Key, Value에 대한 가중치 행렬입니다.

2) 유사도(Attention Score) 계산
각 단어의 Query 벡터와 문장 내 다른 단어의 Key 벡터 간의 유사도를 계산합니다. 이 유사도는 내적(dot product)을 사용하여 계산하며, 그 결과는 Attention Score라고 불립니다. 이 점수는 단어들 간의 연관성을 나타냅니다.

Attention Score

$$\text{Attention Score}(Q, K) = Q \cdot K^T$$

3) Softmax를 통해 가중치로 변환
계산된 유사도(Attention Score)는 Softmax 함수를 통해 가중치로 변환됩니다.
Softmax는 모든 유사도를 0과 1 사이의 값으로 정규화하여, 각 단어가 문장에서 얼마나 중요한지를 나타내는 확률값을 제공합니다.

$$\text{Attention Weights}(Q, K) = \text{Softmax}\left(\frac{Q \cdot K^T}{\sqrt{d_k}}\right)$$

d k 는 Key 벡터의 차원 수로, 이 값으로 나누어 안정적인 학습을 가능하게 합니다.

4) Value 벡터와의 가중합 계산
Softmax로 구한 가중치를 사용해, 각 단어의 Value 벡터에 가중치를 곱해주고, 그 결과들을 가중합하여 최종 출력을 계산합니다. 이 단계는 중요한 단어의 정보를 더 많이 반영하는 과정입니다.

$$\text{Self-Attention Output} = \sum(\text{Attention Weights} \cdot V)$$

3. Self-Attention의 예시

문장: "The cat sat on the mat."
각 단어는 Query, Key, Value 벡터로 변환됩니다.
각 단어의 Query와 다른 단어들의 Key 간의 유사도를 계산합니다.
예를 들어, "cat"의 Query와 "sat"의 Key 간의 유사도를 계산하여 그 연관성을 확인합니다.
Softmax 함수를 사용해 각 단어의 연관성에 대한 가중치를 계산합니다.
"cat"은 "sat"와 강한 연관성을 가지므로 높은 가중치를 부여받습니다.
각 단어의 Value 벡터에 가중치를 곱하여, "cat"과 관련된 중요한 정보를 최종적으로 반영합니다.

4. Self-Attention의 장점
1) 병렬 처리
Self-Attention은 입력 문장 내 모든 단어가 동시에 처리되기 때문에, RNN처럼 순차적으로 처리하지 않아도 됩니다. 이는 병렬 처리가 가능하므로, 효율적이고 빠른 처리가 가능합니다.
2) 긴 문맥 처리
Self-Attention은 멀리 떨어진 단어들 간의 관계도 쉽게 학습할 수 있습니다. 예를 들어, 문장의 앞부분에 있는 단어와 뒷부분에 있는 단어의 상호작용을 효율적으로 학습할 수 있습니

다. 이는 RNN이 가진 장기 의존성(Long-term dependency) 문제를 해결합니다.
3) 문맥을 더 잘 반영
각 단어가 문장 내에서 다른 단어들과 어떻게 연결되는지를 고려하여, 문맥적 의미를 더 잘 반영할 수 있습니다. 이를 통해 더 정확한 문맥적 이해와 텍스트 처리가 가능합니다.
5. Multi-Head Attention (멀티헤드 어텐션)
Self-Attention을 여러 번 수행하는 Multi-Head Attention은 다양한 관점에서 문맥을 이해할 수 있도록 도와줍니다. Multi-Head Attention에서는 여러 개의 Self-Attention 모듈을 병렬로 실행하여, 각 Attention Head가 다른 문맥적 정보를 학습하게 합니다.
여러 개의 Query, Key, Value 벡터 집합을 각각 다른 가중치로 변환하여 Self-Attention을 여러 번 적용합니다.
각 Attention Head의 출력을 합쳐서 더 풍부한 정보를 반영한 최종 결과를 도출합니다.

6. Self-Attention의 응용
1) 기계 번역
Self-Attention 메커니즘은 Transformer 모델에서 중요한 역할을 하며, 기계 번역에서 문맥을 이해하고, 원문과 번역문 간의 관계를 잘 학습할 수 있습니다.
2) 자연어 생성
ChatGPT와 같은 모델은 Self-Attention을 사용하여 주어진 문맥에서 자연스러운 텍스트를 생성합니다. 이를 통해 모델은 문맥을 잘 반영한 문장을 만들어낼 수 있습니다.
3) 문서 요약
긴 문서 내에서 중요한 부분을 파악하고 요약하는 작업에서 Self-Attention은 중요한 정보를 반영하는 데 사용됩니다.

주10. 전이 학습(Transfer Learning)

- 전이 학습(Transfer Learning)은 인공지능(AI)에서 한 작업(Task)에서 학습한 지식을 다른 관련 작업에 적용하는 방법입니다.
- 이 방법은 특히 새로운 작업을 수행할 때 데이터가 충분하지 않거나, 학습 시간이 오래 걸리는 상황에서 유용합니다.
- 전이 학습은 이미 학습된 모델을 재활용하여 모델 성능을 높이고, 학습 속도를 빠르게 하는 데 목적이 있습니다.

전이 학습의 기본 개념
전이 학습의 핵심은 사전 학습(pre-training)과 미세 조정(fine-tuning)입니다. 즉, 모델이 이미 대규모 데이터에서 학습한 지식을 새로운 데이터나 특정 도메인에 맞춰 조정하는 과정입니다. 전이 학습의 기본 아이디어는 두 작업이 어느 정도 관련성을 가지기 때문에 하나의 작업에서 학습한 지식이 다른 작업에 도움이 될 수 있다는 것입니다.
전이 학습이 필요한 이유

- 데이터 부족: 특정 작업에 대한 레이블이 있는 데이터가 적을 때, 다른 대규모 데이터로 학습한 모델을 재활용하여 성능을 향상시킬 수 있습니다.
- 학습 시간 절약: 처음부터 모델을 학습하는 데 걸리는 시간을 줄여, 이미 학습된 모델을 재사용함으로써 효율성을 높일 수 있습니다.

- 복잡한 문제 해결: 특정 도메인에서 학습 데이터를 얻기 어려운 경우, 다른 도메인에서 사전 학습한 모델을 이용해 특정한 작업을 해결할 수 있습니다.

전이 학습의 단계

- 사전 학습 (Pre-training): 먼저 대규모의 일반 데이터셋(예: 이미지, 텍스트, 오디오)을 이용하여 모델을 학습합니다. 이 단계에서 모델은 일반적인 패턴과 지식을 학습하게 됩니다.
- 미세 조정 (Fine-tuning): 사전 학습된 모델을 새로운 작업에 맞춰 조정합니다. 이때 특정 작업에 맞는 작은 데이터셋을 이용해 모델의 가중치를 조정하여 성능을 극대화합니다.

전이 학습의 주요 유형

- 특성 추출 (Feature Extraction): 사전 학습된 모델의 일부 계층을 고정하여, 그 계층들이 학습한 특성을 새로운 작업에 그대로 사용합니다. 일반적으로 사전 학습된 모델의 마지막 계층만 새로운 작업에 맞게 조정하는 방식입니다.
- 미세 조정 (Fine-tuning): 사전 학습된 모델 전체를 새로운 작업에 맞춰 조정하는 방식입니다. 이 경우 모델의 모든 계층이 새로운 작업에 맞게 조정됩니다.
- 고정 모델 사용 (Frozen Model): 사전 학습된 모델의 가중치를 고정한 채, 새로운 작업을 위한 추가 계층을 쌓아 학습합니다. 이 방법은 새로운 작업이 사전 학습한 작업과 크게 다르지 않을 때 유용합니다.

전이 학습의 사례

- 컴퓨터 비전: 이미지넷 같은 대규모 데이터셋에서 사전 학습된 모델(예: ResNet, VGG)은 다양한 이미지 분류 작업에 사용됩니다. 새로운 작업에서는 마지막 분류 계층만 조정하여, 특정한 이미지 분류 작업을 수행할 수 있습니다.
- 자연어 처리: BERT, GPT, T5 등과 같은 대규모 언어 모델이 일반 텍스트에서 사전 학습되며, 이를 감정 분석, 번역, 요약 등 다양한 NLP 작업에 적용할 수 있습니다. 예를 들어, BERT는 대규모 텍스트 데이터로 사전 학습된 후, 특정 도메인(의료 텍스트 등)으로 미세 조정하여 더 높은 성능을 발휘할 수 있습니다.
- 음성 인식: 일반적인 음성 데이터로 사전 학습된 모델을 특정 언어, 또는 특정 도메인(예: 법률, 의료)에서 음성 인식을 수행하도록 미세 조정하여 높은 성능을 달성할 수 있습니다.

전이 학습의 장점과 단점

장점:

- 학습 속도 향상: 이미 학습된 지식을 이용하므로 학습 시간이 단축됩니다.
- 데이터 효율성: 새로운 작업에 적은 양의 데이터로도 좋은 성능을 발휘할 수 있습니다.
- 다양한 작업에 활용 가능: 같은 사전 학습 모델을 다양한 작업에 맞게 조정하여 사용할 수 있습니다.

단점:

- 작업 불일치: 사전 학습된 모델과 새로운 작업이 너무 다르면 효과가 낮아질 수 있습니다.
- 메모리와 계산 비용: 대규모 사전 학습 모델을 미세 조정하거나 사용하는 데 많은 메모리와 계산 비용이 필요할 수 있습니다.

전이 학습은 기존 모델을 재사용함으로써 다양한 AI 작업에서 성능을 향상하고, 특히 특정 작업에 대한 데이터가 적을 때 강력한 해결책이 됩니다.

주11. 자기지도학습 (Self-supervised learning)

▷ 자기지도학습(Self-supervised learning)은 AI 모델이 레이블이 없는 데이터에서 학습할 수 있도록, 데이터를 스스로 생성한 레이블을 이용해 학습하는 방식입니다.

▷ 이 방식은 주로 비지도 학습(unsupervised learning)과 지도 학습(supervised learning)의 중간 형태로, 레이블이 없는 데이터에서 유용한 패턴을 찾아 특정 태스크를 수행하도록 학습됩니다.

Self-supervised learning의 기본 개념

Self-supervised learning에서는 데이터의 일부 정보를 이용해 나머지 정보를 예측하는 방식으로 레이블을 만들어 모델을 학습시킵니다.

예를 들어, 텍스트 데이터에서는 문장의 일부 단어를 가려두고, 모델이 이 단어들을 예측하도록 합니다. 이로 인해 많은 양의 레이블이 없는 데이터를 효율적으로 활용할 수 있습니다.

Self-supervised learning 과정

1) 자체 레이블 생성

- **Self-supervised learning)**의 핵심은 모델이 학습에 필요한 레이블을 스스로 생성한다는 것입니다.
- 모델은 원본 데이터에 특정 변형을 가하여, 그 변형 전후의 관계를 레이블로 사용합니다.

예: NLP에서는 마스킹(Masking) 기법을 사용하여 일부 단어나 문장을 지운 후, 이를 예측하도록 하여 학습 데이터를 생성합니다. 컴퓨터 비전에서는 이미지를 회전시키거나, 일부를 가려서 원본 상태를 복원하도록 학습할 수 있습니다.

2) 변형된 데이터를 학습 데이터로 사용

모델은 변형된 데이터를 입력으로 받고, 원래의 데이터를 예측하면서 학습합니다.

이 과정에서 모델은 데이터의 패턴이나 특성을 스스로 학습하게 됩니다.

예를 들어, 텍스트 데이터에서는 문장에서 특정 단어를 마스킹하고, 모델이 마스크된 단어를 예측하도록 합니다. 이를 통해 모델은 언어의 맥락과 의미를 이해하게 됩니다.

3) 지도 학습 방식으로 학습 진행

- self-supervised learning은 자율 학습과 달리 지도 학습과 동일한 방식으로 손실(loss)을 계산하여 학습합니다.
- 모델이 예측한 결과와 원본 레이블 간의 차이를 손실로 계산하고, 이를 통해 모델을 최적화합니다.

예를 들어, 마스크된 단어 예측의 경우, 예측한 단어와 실제 단어 간의 손실을 계산해, 모델이 더욱 정확하게 단어를 예측할 수 있도록 가중치를 조정합니다.

4) 반복 학습을 통한 특성 추출

이러한 과정을 반복함으로써 모델은 데이터의 중요한 특성을 점점 더 잘 파악하게 됩니다.

학습이 끝나면, 모델은 주어진 데이터를 이해하고, 필요한 정보를 효과적으로 추출하는 데 필요한 패턴을 학습하게 됩니다.

- **Self-supervised learning의 주요 방식**

◇ **마스킹(Masking) 방식**

텍스트 데이터에서 일부 단어를 가리고(마스킹) 모델이 이 단어를 예측하도록 학습합니다.
예: BERT 모델은 문장의 중간 단어 일부를 가려놓고 모델이 이를 예측하게 함으로써 언어 구조와 문맥을 이해하게 합니다.
◇ 다음 항목 예측
텍스트나 이미지에서 다음에 올 항목을 예측하도록 하는 방식입니다.
예: GPT 모델은 텍스트의 다음 단어를 예측하도록 학습하여 언어 생성 능력을 키웁니다.
예: 비디오 프레임에서 다음 프레임을 예측하는 방식도 자가 학습의 예가 됩니다.
◇ 대조 학습(Contrastive Learning)
이미지나 텍스트에서 유사한 항목을 가까이 두고, 다른 항목을 멀리 떨어뜨리는 방식으로 학습합니다.
예: 이미지 두 장이 같은 객체에 속하는지 여부를 학습하여, 유사한 이미지는 유사한 벡터로 표현되도록 합니다.
◇ 부분 복원(Prediction of Missing Parts):
이미지의 일부를 제거한 뒤, 제거된 부분을 모델이 복원하도록 학습합니다. 이를 통해 모델은 이미지의 패턴과 구조를 이해하게 됩니다.

Self-supervised learning 장점
- 대규모 비지도 데이터 활용 가능: 레이블이 없는 데이터에서 자체적으로 레이블을 만들어 학습할 수 있어, 레이블을 수집하기 어려운 데이터에 유용합니다.
- 다양한 도메인에 적용 가능: 텍스트, 이미지, 음성 등 여러 도메인에 자가 학습 방식을 활용할 수 있습니다.
- 데이터 패턴 학습: 모델이 데이터를 자율적으로 해석하고 패턴을 학습해, 다양한 작업에 대한 기초를 다질 수 있습니다.

단점
- 잘못된 레이블 생성 가능성: 변형한 데이터에서 잘못된 레이블이 생성될 경우 학습이 왜곡될 수 있습니다.
- 복잡한 모델 설계: 자가 학습 방식을 적용하기 위해 데이터 변형과 레이블 생성 방식을 잘 설계해야 합니다.

Self-supervised learning은 특히 사전 학습(Pre-training) 단계에서 유용하게 사용되며, 모델이 데이터의 기본 구조와 패턴을 이해하는 데 도움을 줍니다. 이를 통해 모델이 지도 학습 없이도 많은 데이터를 활용하여 초기 성능을 끌어올리고, 이후 미세 조정(Fine-tuning)에서 좋은 성과를 낼 수 있습니다.

Self-supervised learning 주요 사례
대표적으로 NLP(자연어 처리)와 컴퓨터 비전 분야에서 많이 사용되며, 특히 텍스트나 이미지 데이터를 대규모로 활용할 때 유용합니다.
- BERT: 텍스트 데이터에서 일부 단어를 마스킹하고 예측하는 방식으로 언어 구조를 학습합니다.BERT 모델은 문장 내에서 무작위로 일부 단어를 마스킹한 후, 모델이 마스크된 단어를 예측하도록 학습합니다.

예를 들어, “나는 [MASK]을 좋아한다”라는 문장이 주어지면 모델은 [MASK] 자리에 “음악”이

나 “운동”과 같은 단어를 예측하도록 학습합니다.
이를 통해 모델은 문맥을 이해하고, 단어 간의 관계를 학습할 수 있습니다.

- SimCLR, MoCo: 이미지 대조 학습을 사용하여 이미지의 유사성과 차이를 학습하고, 같은 이미지의 다른 변환이 비슷하게 인식되도록 합니다.컴퓨터 비전에서의 이미지 복원 (Image Inpainting)

자가 학습을 통해 모델이 가려진 이미지 부분을 복원하도록 할 수 있습니다. 이미지의 일부분을 지우고, 모델이 이 부분을 원래대로 복원하는 방식으로 학습합니다.
예를 들어, 모델은 인물의 얼굴 일부가 가려진 이미지를 보고, 원래 얼굴을 복원하도록 학습해 사람의 얼굴 구조에 대한 패턴을 학습할 수 있습니다.

- GPT: 텍스트의 다음 단어를 예측하여, 자연스러운 텍스트 생성을 위한 언어 모델을 구축합니다.

Self-supervised learning은 지도 학습처럼 대규모의 레이블 데이터에 의존하지 않고도 고성능 모델을 구축할 수 있게 해 주며, 특히 언어 모델, 이미지 모델 등에서 강력한 성능을 발휘합니다.

Self-supervised Learning 금융 사례

▷ **이상 거래 탐지**

금융 기관에서는 이상 거래를 감지하기 위해, 정상 거래와 비정상 거래의 패턴을 학습합니다.
Self-supervised 방법: 과거 정상 거래 데이터를 사용하여 자기학습 데이터를 구성합니다.
예를 들어, 일부 데이터 포인트를 의도적으로 변형시키거나, 일부 거래의 특정 속성을 마스킹한 뒤 원래 속성을 예측하도록 학습합니다.
이를 통해 모델은 정상적인 거래의 특성을 학습하고, 학습한 패턴에 벗어나는 이상 거래를 탐지할 수 있습니다.
응용 예시: 의심스러운 금액, 빈도 또는 비정상적인 거래 시간 등을 자동으로 감지하여 경고를 발송합니다.

▷ **신용 리스크 예측 및 평가**

Self-supervised 학습을 사용해 과거 신용 데이터를 기반으로 신용 리스크 평가 모델을 개발할 수 있습니다. 신용 점수, 신용 카드 거래 패턴 등의 데이터를 이용해 부도 가능성을 예측합니다.
Self-supervised 방법: 예를 들어, 고객의 신용 거래 내역 중 일부 정보를 마스킹하고, 원래의 정보를 예측하도록 학습시킵니다.
이렇게 하면 고객의 신용 기록과 관련된 패턴을 파악할 수 있으며, 새로운 고객에 대해서도 신용 평가를 수행할 수 있습니다.
응용 예시: 특정 패턴을 따르지 않는 고객이 나타나면 이를 리스크로 분류해, 부도 가능성 등의 신용 위험을 예측합니다.

▷ 포트폴리오 최적화

금융 포트폴리오 최적화에서 self-supervised 학습을 통해 개별 자산의 상관관계 및 시장의 변동성을 학습할 수 있습니다.
Self-supervised 방법: 과거 자산 가격 데이터를 사용해 일정 기간의 데이터를 마스킹하고, 그 기간 동안의 수익률을 예측하도록 학습합니다. 이를 통해 모델은 자산 간의 상관관계와 시장 상황 변동에 따른 영향을 파악하게 됩니다.

응용 예시: 포트폴리오를 구성할 때 자산 간의 연관성 및 변동성을 반영하여, 수익률을 최대화하고 위험을 최소화하는 전략을 세울 수 있습니다.

▷ 가격 예측 및 이상 변동 탐지

주식, 채권, 암호화폐 등의 자산 가격 변동을 self-supervised learning을 통해 예측할 수 있습니다.

Self-supervised 방법: 과거 가격 데이터의 일부 구간을 제거하고, 모델이 이 구간의 가격을 예측하게끔 학습합니다. 이렇게 하면 가격 패턴과 변동성을 파악할 수 있게 됩니다.

응용 예시: 주식 시장이나 암호화폐의 갑작스러운 변동을 예측해 리스크 관리에 활용합니다.

Self-supervised Learning 금융 사례의 장점

- 라벨 없이 패턴 학습 가능: 금융 데이터는 방대하고 라벨링이 어려운데, self-supervised learning을 통해 라벨 없이도 학습이 가능합니다.
- 데이터 효율성: 보유한 대규모 비정형 데이터를 최대한 활용해 학습할 수 있어 비용이 절감됩니다.
- 리스크 관리 강화: 금융 거래의 비정상 패턴이나 리스크를 조기에 감지하여 신속하게 대응할 수 있습니다.

파이썬 실습

Transformer 모델

Self-supervised learning