4페이지 – 트렌스 포머 모델의 소개

저는 Transformer 모델에 대해 이야기하려고 합니다. Transformer 모델은 2017년 Vaswani et al.의 논문 "Attention is All You Need"에서 소개된 자연어 처리(NLP) 모델로, 시퀀스 변환 작업에 혁신적인 접근을 제안했습니다. 기존의 순환 신경망(RNN)과 LSTM과는 다르게, Transformer는 시퀀스 데이터를 처리하기 위해 순차적인 접근을 배제하고 병렬 처리가 가능한 self-attention 메커니즘을 도입했습니다.

Transformer 모델이 왜 중요한지, NLP 및 기타 분야에서의 역할

Transformer 모델은 self-attention 메커니즘을 통해 자연어 처리의 다양한 문제에서 뛰어난 성능을 보였습니다. 기계 번역, 텍스트 요약, 질의 응답 시스템 등에서 우수한 성과를 기록했죠. 또한 BERT와 GPT 같은 모델을 통해 사전 학습된 변형 모델로 NLP의 패러다임을 바꾸었고, 이미지 처리, 음성 인식 등의 다양한 분야에도 응용되고 있습니다.

5페이지

인코더와 디코더 블록의 구조는 크게 차이나지 않아 보입니다. 인코더와 디코더의

차이점은 Masked Multi-Head Attention이 존재한다는 것과 인코더 블록의 Output이

디코더 블록의 Multi-Head Attention의 input으로 들어간다는 것입니다.

7페이지- Self-Attention

Self-Attention 메커니즘에 대해 알아보겠습니다. Self-Attention은 입력 시퀀스의 모든 단어들이 서로를 참조하여 가중치를 계산하는 방식입니다. 각 단어의 쿼리(queries), 키(keys), 값(values)를 계산하고, 쿼리와 키의 내적(dot product)을 통해 어텐션 스코어를 구합니다. 이 스코어는 소프트맥스 함수로 정규화된 후, 값 벡터에 가중합 되어 최종 출력이 됩니다. 이를 통해 문맥에 따른 의미를 효과적으로 확인 할 수 있습니다.

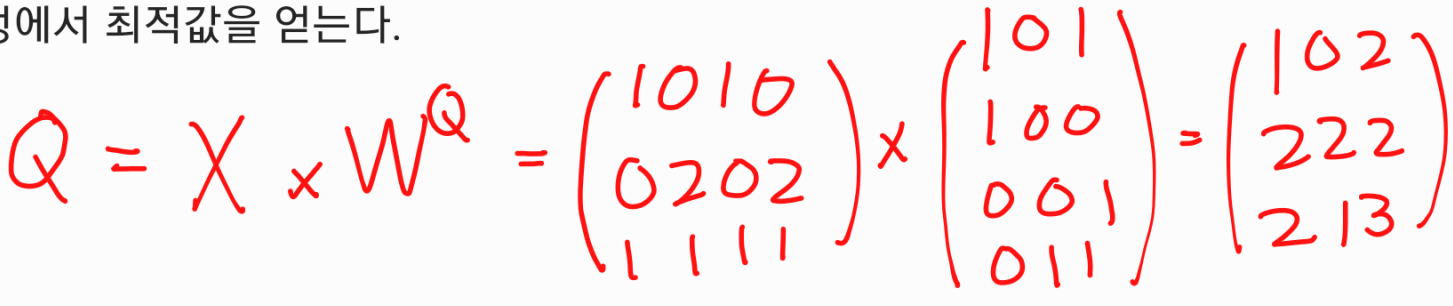
I 순서

- **임베딩 값을 input으로 받는다.**

임베당 값은 모델 학습 과정에서 같이 학습되는데, input값의 shape는(I * e)로 i는 문장의 길이(단어의 개수)이며 e는 임베딩 사이즈이다. 단어의 개수가 3개이고 임베딩 사이즈를 4로 가정한다.

- **Input값을 기준으로 쿼리(Q), 키(K), 벨류(V)를 생성한다.**

input값에 가중치를 곱해서 생성한다. 모두 동일한 사이즈로 (E * D)의 크기를 가지며 D는 임의의 차원이다.(여기서 3으로 가정한다) 처음엔 임의의 값을 가지며 학습 과정에서 최적값을 얻는다.



$$K = X \times W^K = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 2 & 0 & 2 \\ 1 & 1 & 1 & 1 \end{pmatrix} \times \begin{pmatrix} 0 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 0 \\ 1 & 1 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 1 & 1 \\ 4 & 4 & 0 \\ 2 & 3 & 1 \end{pmatrix}$$

$$V = X \times W^V = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 2 & 0 & 2 \\ 1 & 1 & 1 & 1 \end{pmatrix} \times \begin{pmatrix} 0 & 2 & 0 \\ 0 & 3 & 6 \\ 1 & 2 & 3 \\ 1 & 1 & 0 \end{pmatrix} = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 8 & 0 \\ 2 & 6 & 3 \end{pmatrix}$$

- **쿼리(Q) 행렬과 키(K) 행렬의 내적 연산을 수행한다.**

이 연산을 통해 문장의 각 단어들과 문장 내 다른 단어들과의 유사도를 계산하게 된다.

$$Q \times K^T = \begin{pmatrix} 1 & 0 & 2 \\ 2 & 2 & 2 \\ 2 & 1 & 3 \end{pmatrix} \times \begin{pmatrix} 0 & 4 & 2 \\ 1 & 4 & 3 \\ 1 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 2 & 4 & 4 \\ 4 & 16 & 12 \\ 4 & 12 & 10 \end{pmatrix}$$

- **3의 결과를 키 벡터 차원의 제곱근 값으로 나눈다**

이를 통해서 안정적인 경사값을 얻을 수 있게 된다.

$$\frac{QK^T}{\sqrt{d_K}} = \begin{bmatrix} \frac{2}{\sqrt{3}} & \frac{4}{\sqrt{3}} & \frac{4}{\sqrt{3}} \\ \frac{4}{\sqrt{3}} & \frac{16}{\sqrt{3}} & \frac{12}{\sqrt{3}} \\ \frac{4}{\sqrt{3}} & \frac{12}{\sqrt{3}} & \frac{10}{\sqrt{3}} \end{bmatrix}$$

.

- **Softmax function을 통해 정규화를 진행한다.**

이를 통해 나온 결과물을 스코어 행렬이라고 부르며 각 단어의 문장 내 다른 단어들과의 연관정도를 확인할 수 있다.

$$\mathrm{Softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right) = \mathrm{Softmax}\begin{bmatrix} \frac{2}{\sqrt{3}} & \frac{4}{\sqrt{3}} & \frac{4}{\sqrt{3}} \\ \frac{4}{\sqrt{3}} & \frac{16}{\sqrt{3}} & \frac{12}{\sqrt{3}} \\ \frac{4}{\sqrt{3}} & \frac{12}{\sqrt{3}} & \frac{10}{\sqrt{3}} \end{bmatrix}$$

$$= \begin{bmatrix} 0.13613 & 0.43194 & 0.43194 \\ 0.00084 & 0.90884 & 0.09027 \\ 0.00744 & 0.75471 & 0.23785 \end{bmatrix}$$

- **벨류 값과의 행렬곱 연산을 통해 어텐션 행렬(Z)를 구한다.**

어텐션 행렬은 벨류 벡터 값의 가중치 값이 합산된 결과로 문장 내 다른 단어들과의 연관성을 최종적으로 볼 수 있다. 이를 스케일드 닷 프로덕트 어텐션이라고도 불린다.

$$Z = \begin{bmatrix} \quad \end{bmatrix} \times \begin{bmatrix} 123 \\ 280 \\ 263 \end{bmatrix} = \begin{bmatrix} 1.8639 & 6.3194 & 1.7042 \\ 1.9991 & 17.8141 & 0.2135 \\ 1.9926 & 17.4196 & 0.7359 \end{bmatrix}$$

첫 행 [1.8639, 6.3194, 1.7402]는 단어 Messi가 문장 내 단어(messi, smiled, brightly)들과 갖는 연관성을 나타낸다. 단어 smlied와의 값이 6.3194로 가장 높으며 인느 메시가 단어 스마일드와 높은 연관성을 가짐을 알 수 있다. 단어 스마일드는 자기자신과 연관성이 높고, 브라이어틀리 역시 스마일드와 연관성이 높음을 알 수 있다.

Transformer 모델은 입력 시퀀스의 순서를 내재적으로 처리하지 않기 때문에 위치 정보를 명시적으로 추가해야 합니다. 이를 위해 위치 인코딩(Positional Encoding)을 사용합니다. 사인(sin)과 코사인(cosine) 함수의 주기를 이용해 각 위치에 대한 위치 벡터를 생성하고, 이를 입력 임베딩에 더해줌으로써 모델이 각 단어의 위치 정보를 알 수 있게 합니다.

10 페이지 – 인코더

Transformer 모델은 RNN과 달리 입력 시퀀스의 순서를 내재적으로 처리하지 않으므로, 위치 정보를 명시적으로 추가해야 합니다. 이를 위해 위치 인코딩(Positional Encoding)을 사용하는데요, 위치 인코딩은 시퀀스 내 각 위치에 고유한 값을 부여하여 모델이 순서를 학습할 수 있게 합니다. 주로 사인(sin)과 코사인(cosine) 함수의 주기를 이용해 각 위치에 대한 위치 벡터를 생성합니다. 이 벡터는 입력 임베딩에 더해져 모델이 각 단어의 위치 정보를 알 수 있도록 합니다.

11 페이지 – 디코더

디코더는 이전 디코더에서 생성한 값을 Output으로 사용하여 Input으로 받습니다. 디코더에서 입력되는 값은 Multi-Head Attention으로 가기 전에 Masked Multi-Head Attention으로 들어갑니다. 이 어텐션은 Self-Attention이 입력되는 단어에만 만드는 역할을 합니다. Self-Attention은 단어 하나하나마다 다른 단어들과의 관계를 학습합니다. 하지만 디코더는 이전 디코더 레이어에서 생성한 단어들에만 집중한 다음, 생성할 단어가 무엇일지 예측하는 것이 더욱 효율적입니다. 이를 위해 Self-Attention에서 Softmax 함수를 적용하기 전 모두 -무한대로 값을 바꿉니다.

쿼리는 타깃 문장을 표현하는 값이므로 Masked Multi-Head Attention의 결과값을 참조합니다. 키와 벨류는 입력 문장의 표현을 가지는 값이므로 인코더의 출력값을 참고하게 됩니다. Multi-Head Attention에서 쿼리와 키의 내적으로 타깃 단어와 입력 문장 단어들 간의 유사성을 구합니다. 이후 순서는 기존의 Multi-Head Attention과 동일하게 진행됩니다.

12페이지 Multi-Head Attention

셀프 어텐션이 하나만 존재한다고 가정할때, 문장 내 단어의 의미가 모호한 경우 의미가 맞지 않은 단어의 벡터값이 높으면 잘못 해석할 가능성이 높아진다.

만약 단 1개의 어텐션이 학습을 잘못한다면 성능이 떨어질 것이다. 이런 위험성을 낮추기 위해 어텐션을 여러 개 두어 오분류 위험성을 낮추는 것이 멀테-헤드 어텐션이다

13 페이지 - Feed-Forward Networks

각 인코더와 디코더 블록에는 self-attention 후에 피드포워드 네트워크가 위치합니다. 피드포워드 네트워크는 두 개의 선형 변환과 활성화 함수인 ReLU로 구성된 단순한 신경망입니다. 이 피드포워드 네트워크는 각 위치에서의 표현을 독립적으로 처리하여 비선형 변환을 적용하고, 모델의 표현력을 강화합니다.

이제 피드포워드 네트워크의 기본 개념을 설명하겠습니다. Logistic regression은 왼쪽 그림과 같이 하나의 unit으로 표현될 수 있습니다. 반면, 피드포워드 신경망 (FFNN)은 이러한 unit(혹은 neuron)이 층(layer) 단위로 연결된 형태의 네트워크를 말합니다. 간선은 인접한 layer의 unit 쌍끼리 이어져 있으며 한 방향으로만 향합니다.

Input layer는 데이터 상의 초기 input을 나타내는 unit들이 모인 layer이고, output layer는 신경망 모델이 예측한 최종 output을 나타내는 unit들이 모인 layer입니다. Hidden layer는 input layer와 output layer를 제외한 나머지 layer들로, 모델의 내부적이고 은닉된 층이라는 의미에서 hidden이라는 이름이 붙여졌습니다. Input layer는 통상적으로 0번째 layer라고 간주하며, 네트워크의 layer 개수에 포함하지 않습니다. 예를 들어, 오른쪽 그림의 FFNN은 3-layer 네트워크입니다.

14 페이지 - Residual Connections (잔차 연결)

1. 정보 손실 방지

깊은 신경망에서 정보 손실은 학습 과정에서 발생하는 일반적인 문제입니다. 각 레이어를 통과할 때마다 신호의 일부가 손실될 수 있으며 이는 모델의 성능 저하로 이어질 수 있습니다. 잔차 연결은 이러한 문제를 해결하는 데 도움이 됩니다.

왼쪽 그림은 잔차 연결을 사용한 트랜스포머 모델의 아키텍처를 보여줍니다. 입력 시퀀스는 인코더의 각 레이어를 통과하며 각 레이어는 입력에 정보를 추가합니다. 그런 다음 각 레이어의 출력은 입력에 추가되어 최종 출력이 생성됩니다. 이 추가 단계는 신호의 정보 손실을 방지하는 데 도움이 됩니다. 입력 시퀀스의 원본 정보는 각 레이어를 통과하여 최종 출력에 유지됩니다. 이는 모델이 더 긴 시퀀스를 학습하

고 더 나은 성능을 달성하는 데 도움이 됩니다.

2. 학습 속도 향상

"잔차 연결은 또한 트랜스포머 모델의 학습 속도를 향상시키는 데 도움이 됩니다. 잔차 연결을 사용하면 각 레이어의 출력이 입력에 직접 추가되므로 그라디언트가 더 쉽게 역전파될 수 있습니다. 이는 모델이 더 빠르게 학습하는 데 도움이 됩니다

3. 최적화 문제 해결

깊은 신경망은 학습하기 어려울 수 있으며 최적화 문제에 빠질 수 있습니다. 잔차 연결은 이러한 문제를 해결하는 데 도움이 될 수 있습니다. 잔차 연결을 사용하면 모델이 더 안정적이고 최적화하기 쉬워집니다

4. 모델 성능 향상

잔차 연결을 사용하면 트랜스포머 모델의 성능이 크게 향상됩니다. 잔차 연결을 사용한 트랜스포머 모델은 자연어 처리, 기계 번역, 텍스트 요약 등 다양한 작업에서 최첨단 결과를 달성했습니다.

15페이지 - **Layer Normalization(층 정규화)**

층 정규화(Layer Normalization)는 인공 신경망 학습 과정에서 각 레이어의 출력을 정규화하는 기술입니다. 이는 신경망 학습을 여러 측면에서 돕는데 다음과 같습니다..

1. 기울기 폭발 문제 방지

첫 번째로, 층 정규화는 기울기 폭발 문제를 방지하는 데 도움이 됩니다. 신경망 학습 과정에서 각 레이어의 가중치 업데이트가 너무 커져 학습이 불안정해지는 현상을 기울기 폭발(Gradient Explosion)이라고 합니다. 층 정규화는 각 레이어의 출력을 정규화하여 이 문제를 방지합니다.

왼쪽 그림은 층 정규화를 사용하지 않은 신경망에서 발생하는 기울기 폭발 문제를 보여줍니다. 각 레이어를 통과할수록 기울기가 점점 커져 학습이 불안정해집니다. 반면, 오른쪽 그림은 층 정규화를 사용한 신경망에서 각 레이어의 출력이 정규화되어 기울기 폭발 문제가 발생하지 않는 것을 보여줍니다.

2: 학습 속도 향상

두 번째로, 층 정규화는 학습 속도를 향상시키는 데 도움이 됩니다. 층 정규화를 사용하면 각 레이어의 출력 분포가 정규 분포에 가까워져 학습이 더 안정적이 됩니다. 이는 모델이 더 빠르게 학습하는 데 도움이 됩니다.

3: 과적합 방지

세 번째로, 층 정규화는 과적합(Overfitting)을 방지하는 데 도움이 됩니다. 과적합은 학습 데이터에만 맞는 모델이 만들어져 일반화 능력이 저하되는 문제입니다. 층 정규화를 사용하면 각 레이어의 출력 범위를 제한하여 모델이 학습 데이터에만 맞는 것이 아니라 일반화 능력을 향상시키는 데 도움이 됩니다.

4: 모델 학습 안정화

네 번째로, 층 정규화는 모델 학습을 안정화하는 데 도움이 됩니다. 층 정규화를 사용하면 학습 과정에서 발생하는 변동성을 줄여 모델이 더 안정적으로 학습하도록 합니다.

5: 하이퍼파라미터 설정 간편

마지막으로, 층 정규화는 하이퍼파라미터 설정을 간편하게 합니다. 층 정규화를 사용하면 학습률과 같은 다른 하이퍼파라미터에 대한 민감도가 감소하여 하이퍼파라미터 설정이 덜 중요해집니다.

슬라이드 7: 층 정규화의 작동 방식

층 정규화는 다음과 같은 단계로 수행됩니다:

- 평균 및 표준 편차 계산: 각 레이어의 출력에 대한 평균 및 표준 편차를 계산합니다.

- 정규화: 각 레이어의 출력을 평균에서 빼고 표준 편차로 나눕니다.

- 가중치 조정: 정규화된 출력에 가중치를 곱하고 편향을 더합니다.