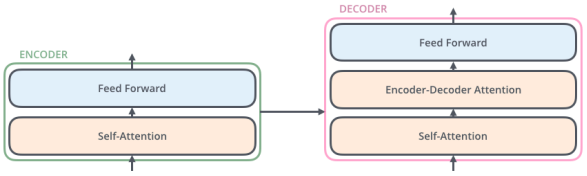
Transformer

일시: 2025년 8월 18일

제출자: 1반 윤소현

목차

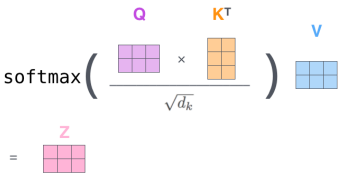
#1 Encoder와 Decoder



- Transformer는 Encoder, Decoder로 나뉩니다.
- Input과 Output
  - Encoder: 문장 길이 n, 임베딩 차원 d일때 input은 `word embedding 벡터(size [n × d])`이며 output은 `문맥 표현 벡터(size [n × d])`입니다.
  - Decoder: 현재까지 생성된 단어 k, 모델이 다룰 수 있는 전체 후보 단어의 개수 v일때 input은 `encoder output(문맥 표현 벡터, size [n × d])`과 `이미 생성된 단어 임베딩(지금까지의 출력 토큰 시퀀스, size [k × d])`이며 output은 `다음 단어 후보들에 대한 확률 분포(size [1 × v])`입니다.
- Encoder는 여러 개의 층을 쌓아 올린 구조이고 각 층은 Self-Attention Layer와 Feed-Forward Neural Network 두 부분으로 나뉩니다.
  - Self-Attention Layer는 문장 안의 단어들이 서로 어떤 관련을 가지는지를 계산합니다. 모든 단어 쌍을 비교해 중요한 연결을 찾아내고 그 결과 각 단어 벡터는 원래의 의미에 더해 다른 단어들과의 맥락 정보가 함께 반영되도록 갱신됩니다.
  - 그다음 Feed-Forward Network는 이렇게 맥락이 반영된 단어 정보를 비선형적으로 가공해서 단어의 표현을 더 높은 차원의 패턴으로 바꾸는 역할을 합니다. 이렇게 Encoder의 각 층을 지나며 문맥에 맞는 표현으로 가공(문맥 표현 벡터가 생성)됩니다.
- Decoder도 Self-Attention Layer와 Feed-Forward Network가 있고 그 사이에 Encoder-Decoder Attention 층이 추가됩니다.
  - Encoder-Decoder Attention 층은 디코더의 현재 hidden state가 인코더가 출력한 문맥 표현 벡터를 반영하여 어떤 위치(단어)에 주의를 기울일지를 알려주는 역할을 합니다(hidden state: 지금까지 생성된 단어 임베딩, 즉 출력 토큰 시퀀스가 디코더의 Self-Attention → Cross-Attention → FFN을 거쳐 생성된 현재 디코더 출력 벡터).

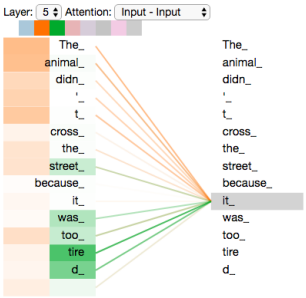
#2 Transformer의 구성

- Self-Attention



  - Self-Attention에서는 각 단어가 문장 내의 다른 단어들과 어떤 연관성을 가지는지를 Query, Key, Value 세 가지 벡터로 계산합니다. Query와 Key를 내적해 유사도를 구하고 이를 Softmax로 확률처럼 정규화하고 Value와 곱해 가중합 Z을 계산하면 각 단어에 다른 단어들과의 맥락 정보가 반영됩니다.

- Multi-Head Attention



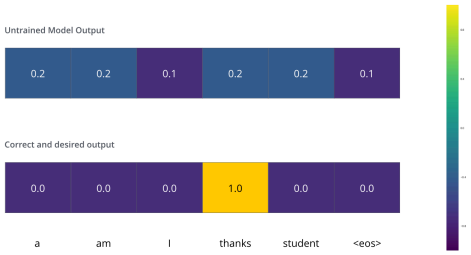
  - 특정 단어는 주변 단어들과의 다양한 의미적 맥락을 동시에 고려해야 하므로, 하나의 self-attention만으로는 부족하며 이에 여러 개의 Attention Head가 독립적으로 학습되는 Multi-Head Attention을 사용합니다. 예를 들어 우리가 “그것”을 인코딩할때 주황색 head는 “그 동물”에 집중하고 초록색 head는 “피곤”이라는 단어에 집중합니다. 이처럼 서로 다른 head들이 각기 다른 단어에 주목해서 모델은 “그것”의 표현 속에 “동물”과 “피곤” 두 단어의 의미를 동시에 반영할 수 있습니다. 마지막에는 이러한 여러 head의 출력을 합쳐 또 다른 가중치 행렬에 투영하고, 이로써 모델은 다양한 관점에서 얻은 정보를 통합해 더 풍부한 표현을 학습합니다.

- Positional Encoding
  - “동물이 길을 건너지 않았다”와 “길이 동물을 건너지 않았다” 같은 문장을 구분하려면 단어의 순서도 고려해야 하므로, 각 단어의 임베딩에 위치 정보를 담은 벡터를 더하는데 이 벡터는 단순히 1, 2, 3처럼 순서를 기록하는 것이 아니라, sine과 cosine 함수로 만들어진 주기적 패턴을 따르기 때문에 모델은 단어의 위치와 간격을 수학적 규칙으로 인식할 수 있습니다.
- Residual Connection과 Layer Normalization
  - Residual Connection은 입력(Raw 입력)과 출력(맥락을 반영하여 가공된 입력)을 더해 전달하는 방식으로 정보 손실을 줄이고 학습이 깊어질수록 발생하는 기울기 소실 문제를 완화합니다. Layer Normalization은 각 층의 출력을 정규화해 학습을 안정화합니다.
- Decoder의 Self-Attention
  - Decoder Self-Attention은 Encoder와 다르게 미래 단어를 보지 못하게 합니다. 이는 번역이 순서(좌→우)대로 이루어지기 때문인데 사람이 글을 쓸 때도 다음 단어를 미리 볼 수 없는 것처럼 모델 역시 아직 출력되지 않은 단어에 접근하지 못하는게 맞기 때문에, 이를 위해 마스킹이라는 기법을 사용하여 미래 단어의 정보를 차단시킴으로써 Decoder는 문장을 순서(좌→우)대로 생성하게 됩니다.
- Linear Layer와 Softmax
  - Decoder가 만든 벡터(= 인코더 문맥 벡터와 지금까지 생성된 출력 토큰 임베딩 벡터가 디코더의 self-attention과 cross-attention을 거쳐 생성된 hidden state)는 Linear Layer를 통해 모델이 학습한 전체 어휘 집합과 연결된 로짓 벡터가 됩니다. 만약 어휘가 10,000개라면, 로짓 벡터는 10,000차원의 벡터가 됩니다. 벡터의 각 원소는 해당 단어가 정답일 가능성을 점수 형태로 표현합니다. 로짓 벡터의 값들은 실수라 확률로 해석할 수 없으므로 Softmax 함수를 적용하여 모든 값의 합이 1이 되도록 만들고 변환된 결과는 “각 단어가 다음에 나올 확률 분포”가 됩니다. 이를 기반으로 가장 높은 확률을 가진 단어가 선택되어 실제 출력 단어가 됩니다. 이 과정이 반복되면서 한 단어씩 생성해가며 최종 문장을 만들어냅니다.

#3 Transformer의 학습

- 학습 목표
  - Output Vocabulary는 모델이 출력할 가능한 후보 단어들의 목록입니다.

    - 정답 벡터는 Output Vocabulary의 크기만큼 차원을 가진 벡터로 정답 단어 위치만 1이고 나머지는 0인 벡터입니다.
    - 모델 출력(예측)은 Linear + Softmax를 거쳐 나온 확률 분포로써 모든 값의 합이 1인 벡터입니다.
    - 정답은 “답인 한칸만 1인 벡터”, 예측은 “모든 값의 합이 1인 분포(벡터)”입니다. 두 벡터를 얼마나 가깝게 만들 것인가가 학습 목표입니다.
- Cross-Entropy + 역전파
    - 두 벡터를 얼마나 가깝게 만들 것인가가 학습 목표인데 “가깝다/멀다”의 정도를 Cross-Entropy로 측정합니다. Cross-Entropy는 확률 분포에서 정답 칸의 확률값 크기에 따라 손실을 부여합니다. 정답 칸의 확률을 p라고 하면 손실은 −log p가 됩니다. 모델이 정답칸 확률을 0.9로 주면 −log(0.9)라는 작은 손실을 받고, 0.1로 주면 −log(0.1)이라는 큰 손실을 받습니다. 이 손실은 역전파 과정에서 모델의 파라미터에 대해 손실 함수의 기울기를 계산하는 형태로 전파되고 정답 클래스의 확률이 커지도록 가중치를 조정합니다.



#4 Transformer의 추론(디코딩)

- 학습이 끝나고 모델을 실제로 쓰는 단계에서는 정답이 없으므로 디코더가 스스로 다음 단어를 선택해가며 문장을 생성합니다. Decoder가 생성한 벡터(= 인코더 문맥 벡터와 지금까지 생성된 출력 토큰 임베딩 벡터가 디코더의 self-attention과 cross-attention을 거쳐 생성된 hidden state)는 Linear Layer를 거쳐 ‘해당 단어가 정답일 가능성’ 벡터(로짓 벡터)가 되고 Softmax 함수를 통해 ‘다음 단어 후보들에 대해 얼마나 가능성이 높다고 판단하는지 정보’인 확률 분포가 됩니다. 여기서 확률이 가장 큰 단어를 선택해 출력하는 것이 Greedy Decoding입니다.

#5 Beam Search

- 모델이 문장을 만들기위한 첫 단어를 뽑을 때 Greedy Decoding이 확률이 가장 큰 단어 1개를 선택할 때 Beam Search에서는 확률이 높은 단어 k개를 후보로 두고, 각 후보로 시작하는 k개 시퀀스(후보 문장)를 선택합니다. 두 번째 단어를 뽑을 땐 k개 시퀀스마다 두 번째 단어를 추가하고 각 시퀀스의 누적 로그 확률을 계산, 확률이 높은 k개의 시퀀스만 남기고 나머지는 제거하여 후보는 k개를 유지합니다. 이러한 절차를 반복하여 최종적으로 완성된 시퀀스들 중에서 누적 확률이 가장 높은 시퀀스를 최종 출력으로 선택합니다. 빔 크기 k를 키우면 더 많은 후보를 선택함으로써 더 다양한 문장 경로를 탐색할 수 있지만 계산량이 늘어납니다. 빔 크기를 1로 줄이면 그리디 방식이 됩니다.

#6 학습과 추론

- 학습과 추론의 차이
    - 학습 때는 정답이 주어지므로 모델의 출력 확률 분포와 정답 벡터를 비교하여 오차를 계산할 수 있고, 추론 때는 정답이 없으므로 모델이 스스로 다음 단어를 선택해야 하는 환경입니다.
    - 학습 때는 출력인 예측 분포와 정답을 비교하여 Cross-Entropy를 계산하고 가중치를 수정하며 추론 때는 확률 분포에서 그리디 또는 Beam Search 등으로 다음 단어를 선택합니다.