Attention 메커니즘 (2015)

- RNN (1986) : RNN 첫 시작
- LSTM (1997) : 다양한 시퀀스 정보 모델링 가능 - 주가예측, 주기함수 예측
- **Seq2Seq (NIPS 2014)**

▷ LSTM 활용해서 딥러닝 기반 기술로 탄생. 현대의 딥러닝 기술들이 나오던 시점에 탄생.

▷ LSTM 을 활용해서 고정된 크기의 context vector를 사용하는 방식으로 번역 수행.

⇢ context vector는 인코더가 입력 시퀀스의 모든 정보를 요약해 압축한 벡터로, 디코더가 각 단어를 생성할 때 참조하는 역할을 합니다.

⇢그러나 고정된 크기의 context vector에는 한계가 있습니다.

특히 입력 시퀀스가 길어질수록 모든 정보를 담기에 벡터 크기가 부족해질 수 있고, 이로 인해 성능이 떨어지거나 정보 손실이 발생할 수 있습니다. 이를 해결하기 위해 어텐션 메커니즘이 도입되었습니다. 어텐션 메커니즘은 디코더가 필요할 때마다 인코더의 각 시점별 hidden state를 동적으로 참조하여 중요한 정보를 선택적으로 활용할 수 있게 해줍니다. 이를 통해 길이가 긴 시퀀스에서도 더 높은 성능을 낼 수 있게 되었습니다.

- **Attention (ICLR 2015)** : Seq2Seq 모델에 어텐션 기법 적용하여 성능 더 올릴 수 있었음
- **Transformer (NIPS 2017)**

▷ RNN 자체를 사용할 필요가 없다고 제안

▷ 오직 어텐션 기법에 의존하는 아키텍쳐를 설계했더니 성능이 엄청 향상.

▷ 트랜스포머를 기점으로 자연어 처리 기법으로 RNN 더이상 사용하지 않고 어텐션 메커니즘을 더욱 더 많이 사용

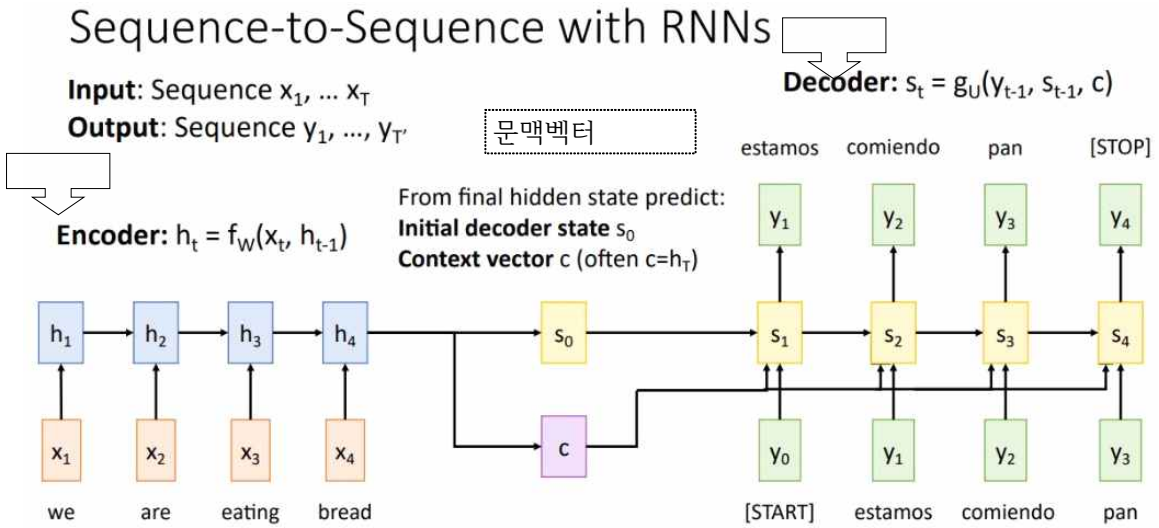
▷ 어텐션 메커니즘 등장 이후 입력 시퀀스 전체에서 정보를 추출하는 방향으로 연구 방향 발전.

▷ 어텐션 기법을 활용하는 트랜스포머 아키텍쳐를 따르는 방식으로 다양한 고성능 모델들이 제안되고 있음

- GPT : Transformer의 '디코더(decoder)' 아키텍처 활용
- BERT : Transformer의 '인코더(encoder)' 아키텍처 활용

Seq2Seq with RNN 모델 (2014) 설명

- Seq2Seq(Sequence to Sequence) 모델은 주로 자연어 처리 분야에서 사용되며, 입력 시퀀스를 다른 시퀀스로 변환하는 작업에 적합한 모델입니다.
- 이 모델은 특히 기계 번역, 텍스트 요약, 질문 답변과 같은 작업에서 널리 사용됩니다. Seq2Seq 모델은 RNN(Recurrent Neural Network) 아키텍처를 사용하여 시퀀스 데이터를 처리하며, 기본적으로 두 가지 주요 구성 요소로 나눌 수 있습니다:



예시) 나는 철수를 사랑해

▷ 위 그림처럼 Encoder에서는 input sequence를 받아 t만큼의 step을 거쳐 h t hidden state를 생산하고 이를 decoder의 첫번째 hidden state s 0 와 context vector로 사용한다.

▷ Decoder에서는 input인 START token을 시작으로 이전 state(첫번째에선 s 0)과 context vector를 통해 hidden state s t 를 생산하고 이를 통해 output y t 를 계산한다.

▷ 이 output을 다시 input으로 넣어 STOP token을 output으로 출력할 때 까지 step을 반복한다.

이때 context vector는 input sequence를 요약하여 decoder의 매 step에서 사용되며 encoded sequence와 decoded sequece사이에서 정보를 전달하는 중요한 역할을 한다.
하지만 실제 환경에선 위 그림의 예제처럼 input이 simple sequence가 아니고 text book처럼 input sequence가 매우 클 경우 문제가 발생한다.
(문재점)

▶ 첫째, 하나의 context vector에 모든 input 정보를 압축하려고 하니 정보 손실이 발생한다 (Bottleneck 현상 or Long-Term Dependencies problem).[1)]

▶ 둘째, backprop시 gradient vanishing 문제가 발생한다.

위와같은 문제를 해결하기 위해 single context vector를 사용하는 것이 아니라 decoder의 매 step별로 context vector를 새로 생성하는 Attention개념을 도입하였다.

1. 인코더(Encoder)

- 인코더는 입력 시퀀스를 받아 이를 고정된 길이의 벡터, 즉 컨텍스트 벡터(Context Vector)로 변환하는 역할을 합니다.
- 이 과정은 주로 RNN, LSTM(Long Short-Term Memory), GRU(Gated Recurrent Unit) 같은 순환 신경망으로 처리됩니다.
- 인코더의 RNN은 입력 시퀀스의 각 단어를 하나씩 처리하면서, 이전에 처리한 단어들의 정보를 기억하고 이를 사용해 다음 단어의 정보를 갱신합니다.
- 시퀀스의 마지막 RNN 셀이 생성한 최종 은닉 상태(hidden state)가 컨텍스트 벡터로 사용되며, 이는 디코더로 전달되어 시퀀스 변환의 핵심 정보가 됩니다.

2. 디코더(Decoder)

- 디코더는 인코더에서 전달받은 컨텍스트 벡터를 사용하여 새로운 시퀀스를 생성하는 역할을 합니다.
- 디코더 역시 RNN, LSTM, GRU 같은 순환 신경망을 사용할 수 있으며, 인코더와 구조가 유사합니다.
- 디코더는 첫 번째 시점에 컨텍스트 벡터와 함께 시작 토큰을 받아 다음 단어를 예측하고, 그 다음 시점부터는 이전 시점에 예측한 단어를 입력으로 받아 계속해서 단어를 생성합니다.
- 이 과정은 전체 시퀀스를 예측할 때까지 반복됩니다.

3. Seq2Seq 모델의 동작 과정

1) 인코더 단계

- 입력 시퀀스(예: "I am a student")가 인코더에 전달됩니다.
- 인코더는 입력 시퀀스를 시간 축을 따라 하나씩 처리하여 각 시점마다 새로운 은닉 상태를 생성하고, 마지막 시점에서 최종 은닉 상태(컨텍스트 벡터)를 생성합니다.
- 컨텍스트 벡터 : 인코더의 마지막 은닉 상태는 입력 시퀀스의 정보를 압축하여 나타낸 컨텍스트 벡터입니다. 이 벡터는 입력 시퀀스의 모든 중요한 정보를 요약한 것으로, 디코더가 새로운 시퀀스를 생성할 때 참조하는 주요 정보입니다.

2) 디코더 단계

- 디코더는 인코더로부터 받은 컨텍스트 벡터와 시작 토큰(예: "<start>")을 입력으로 받아, 첫 번째 단어를 예측합니다.
- 예측된 단어는 다음 시점의 입력으로 사용되어 디코더가 다음 단어를 예측하도록 하며, 이 과정을 시퀀스가 끝날 때까지 반복합니다.

4. 장점과 한계

장점:

---

1)

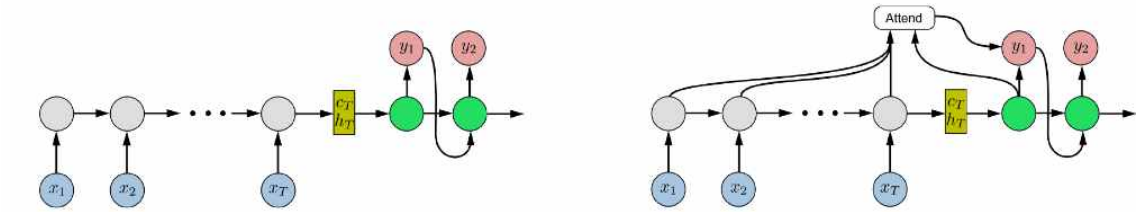
- 다양한 길이의 시퀀스 처리: 입력 시퀀스와 출력 시퀀스의 길이가 달라도 유연하게 대응할 수 있습니다.
- 모듈화: 인코더와 디코더는 독립적으로 작동하므로, 다양한 RNN 구조를 적용할 수 있습니다.

한계

- 정보 압축 문제: Seq2Seq 모델에서 컨텍스트 벡터가 입력 시퀀스 전체 정보를 담고 있기 때문에, 긴 시퀀스의 경우 중요한 정보가 손실될 수 있습니다. 이는 기계 번역과 같은 긴 문장을 처리할 때 문제를 일으킬 수 있습니다.
- 장기 의존성 문제: 기본 RNN 구조는 긴 시퀀스에서 앞부분의 정보를 잃어버릴 수 있는 문제를 가지고 있습니다. LSTM과 GRU 같은 개선된 RNN 구조가 이를 완화하지만, 여전히 한계가 존재합니다.

5. Attention 메커니즘의 등장

- Attention 메커니즘은 Seq2Seq 모델의 한계를 극복하기 위해 등장한 기법입니다.
- 기본 Seq2Seq 모델은 인코더의 마지막 은닉 상태 하나만을 컨텍스트 벡터로 사용하여 정보를 압축하지만, **Attention 메커니즘은 디코더가 출력을 생성할 때 입력 시퀀스의 모든 은닉 상태를 참조할 수 있도록 합니다.**
- 이를 통해 디코더는 시퀀스의 특정 부분에 더 집중하여 정보를 처리할 수 있어 성능이 향상됩니다.



(a) Seq2Seq (b) Seq2Seq with attention mechanism

Attention 메커니즘 (2015)

- 딥러닝 분야에서 특히 자연어 처리(NLP)에서 중요한 혁신을 가져왔습니다.
- Attention 메커니즘은 RNN(Recurrent Neural Network) 및 LSTM(Long Short-Term Memory) 같은 기존 모델이 가지는 한계, 특히 긴 문장이나 시퀀스 데이터를 처리할 때 중요한 정보가 소실되는 문제를 해결하는 데 중요한 역할을 했습니다.
- 2015년 ICLR에서 발표된 "Neural Machine Translation by Jointly Learning to Align and Translate"로, 저자들은 이 논문에서 처음으로 Attention 메커니즘을 제안했습니다.
- 이 논문에서는 **seq2seq 모델에 Attention을 적용**하였습니다.

배경

- 전통적인 RNN 기반의 자연어 처리 모델들은 입력 시퀀스가 길어질수록 학습 과정에서의 정보 손실 문제가 발생합니다.

- 예를 들어, 문장이 길어질수록 초반의 정보가 뒤로 갈수록 소실되거나 희미해지는 장기 의존성 문제(Long-term Dependency)를 겪습니다. 이 때문에, 긴 문장을 번역하거나 분석하는 데에 어려움이 있었습니다.

Seq2Seq 한계점

- 소스 문장을 대표하는 하나의 context vector 를 만들어야 하는데 입력 문장이 짧을 수도 있고 길 수 도 있다.
- 이런 다양한 경우의 수가 있는데 항상 고정된 크기의 context vector 를 만들어야 하는건 병목 현상의 원인이 될 수 있습니다.

2015년의 Attention 메커니즘은 이러한 문제를 해결하기 위해 등장했습니다.
기본 아이디어는 모델이 시퀀스의 각 부분에 주의를 기울여, 번역이나 예측 과정에서 특정 시점의 중요한 정보를 더 잘 파악할 수 있도록 하는 것입니다.
이를 통해 모든 입력 단어를 동일한 방식으로 처리하는 대신, 각 입력 단어가 예측에 기여하는 정도를 가중치를 통해 차등적으로 반영할 수 있게 되었습니다.

Attention 메커니즘의 주요 아이디어
Attention의 핵심은 모델이 출력 단어를 예측할 때 입력 시퀀스 전체를 참조할 수 있도록 하며, 그중 어떤 부분이 더 중요한지 가중치를 부여한다는 것입니다.

Attention 메커니즘의 핵심 아이디어
1. 동적 집중(Dynamic Focus)

모델이 입력 데이터의 다양한 부분에 가변적인 집중도(focus)를 적용할 수 있도록 하는 기법입니다.
이 개념은 특히 자연어 처리(NLP)와 컴퓨터 비전(Computer Vision)에서 입력의 중요한 요소를 동적으로 파악하고 가중치를 할당하는 데 사용

- 기존의 Seq2Seq 모델에서는 입력 시퀀스 전체를 하나의 고정된 컨텍스트 벡터로 압축하여 처리했기 때문에, 특히 긴 시퀀스에서 중요한 정보가 소실되는 문제(Bottleneck 현상)가 있었습니다.
- Attention 메커니즘은 이를 해결하기 위해, 출력 시퀀스의 각 단어를 예측할 때 입력 시퀀스의 모든 요소를 가중치를 기반으로 참조하게 합니다.
- 이로 인해, 출력 시점에서 모델이 중요한 입력 정보에 더 집중할 수 있습니다.

가변성(Flexibility)
동적 집중은 고정된 패턴이 아닌, 입력 데이터에 따라 집중할 부분이 동적으로 변경됩니다. 모델은 각 입력마다 다른 패턴을 학습하며, 입력의 특정 부분에 더 큰 가중치를 부여 (가중치 합=1)할 수 있습니다.
어텐션 메커니즘의 연장선:
동적 집중은 어텐션 메커니즘의 일종입니다. 어텐션은 Query, Key, Value를 사용해 입력의 중요한 부분을 강조하며, 동적 집중은 이 과정을 통해 입력의 의미적 관계를 더욱 정확하게

포착합니다.
예시
번역 모델에서 동적 집중은 문장 구조나 단어의 중요도에 따라 단어 간의 관계를 동적으로 학습할 수 있도록 합니다.
이미지 인식에서는 특정 객체나 주요 영역에 집중하여, 더 정확한 특징 추출이 가능합니다.

2. 동적 집중의 구현과 수식 : Self-Attention (자기-어텐션) 메커니즘, Query, Key, Value

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

- Q (Query): 현재 입력이 무엇을 찾고 있는지 나타냅니다.
- K (Key): 각 입력의 특징을 나타냅니다.
- V (Value): 실제 입력의 정보입니다.
- 1/ k : 스케일링 인자로, 큰 값으로 인해 발생할 수 있는 기울기 소실 문제를 완화합니다.
- softmax: 정규화 함수로, 각 입력에 가중치를 부여합니다.
- 이 수식은 입력 벡터 간의 유사도를 계산하고, 이 유사도를 기반으로 집중할 위치를 동적으로 결정합니다.

Attention 메커니즘의 구성 요소
기본적으로 Query, Key, Value라는 세 가지 벡터와 이들 간의 상호작용으로 이루어집니다.
1) Query (쿼리, 질의)
- 현재 타겟 단어가 입력 시퀀스의 어느 부분에 주목해야 하는지를 결정합니다.
- Query는 "어떤 정보가 필요한가?"를 나타내는 벡터입니다. Query는 검색어처럼, 입력 데이터에서 중요한 정보를 찾기 위한 기준입니다.
- 예를 들어, 번역 작업에서 다음에 나올 단어를 예측할 때, Query는 현재 모델이 예측하려는 단어의 정보로 표현됩니다.
- Query는 주로 디코더에서 나오는 벡터로, 해당 시점에서 어떤 입력 부분이 중요한지 찾는 역할을 합니다.

2) Key (키, 특징)
- 인코더에서 생성된 hidden states로, Query가 참조하는 대상의 "특성"을 정의합니다.
- 각 Key는 입력 시퀀스의 한 시점에 해당하는 **정보**를 포함하고 있습니다.
- Key는 "어떤 정보가 존재하는가?"를 나타내는 벡터입니다.
- Key는 주로 인코더에서 나오는 벡터입니다. 예를 들어, 기계 번역에서 입력 문장의 각 단어를 나타낼 수 있습니다.

3) Value (값, 응답)
- 최종적으로 반환될 "정보"를 담고 있는 벡터입니다. Value는 입력 데이터의 실제 정보입니다.

- Key와 같이 인코더의 hidden states로부터 생성되며, 실제로 Query가 주목할 특정 정보를 포함하고 있습니다.
- Attention 메커니즘은 **각 Value를 가중합하여 최종 context를 계산하는 데 사용**합니다.
- Value는 "어떤 정보를 제공할 것인가?"**를 나타내는 벡터입니다.
- Key와 Query가 상호작용한 결과로 도출된 중요도에 따라 실제로 사용할 정보를 담고 있습니다.
- Value 역시 입력 데이터의 각 부분을 나타내며, Key와 함께 제공됩니다.

검색 엔진을 비유하자면:
- Query는 사용자가 입력한 검색어
- Key는 웹 페이지의 메타데이터
- Value는 웹 페이지의 내용입니다.

검색 엔진은 사용자의 Query와 Key를 비교하여, 가장 관련성이 높은 Value(웹 페이지)를 반환합니다.

3. 가중치 계산 **(Attention Score)**

▷ 모델은 입력 시퀀스의 각 부분이 현재 예측에 얼마나 중요한지를 결정하기 위해 가중치를 계산합니다. 이는 입력과 출력 사이의 정렬(alignment) 문제를 해결하는 방법입니다.

▷ Query와 Key 사이의 유사도를 기반으로 가중치(스코어)를 계산합니다. 가중치 합=1

▷ 이 스코어는 두 벡터 간의 내적(dot product)으로 구할 수 있으며, 이 스코어는 Query가 특정 Key에 얼마나 집중할지를 나타냅니다.

▷ 이 스코어는 이후에 Softmax 함수를 통해 확률 값으로 변환되며, 입력 시퀀스의 각 요소에 대한 중요도를 가중치로 부여합니다.

(과정)

1) Query와 Key의 Dot product

- 각 Query와 Key에 대해 dot product(내적)을 계산합니다.
- 내적 값은 두 벡터가 얼마나 유사한지, 즉 Query가 특정 Key에 주목할 정도를 나타냅니다.
- 내적 결과가 크면 Query와 Key 간 유사도가 높아 해당 Key에 더 집중해야 한다는 의미로 해석됩니다.

2) Scaling (스케일링)

내적으로 얻어진 값에

$\sqrt{d_k}$

로 나누어 줍니다. 여기서 d_k 는 Key의 차원 수입니다.

이렇게 하는 이유는 Key의 차원이 클수록 내적 값이 커질 가능성이 높기 때문에,

이를 안정화하여 너무 큰 값이나 작은 값으로 인한 학습 불안정을 방지합니다.

$$\text{Scaled Dot Product} = \frac{\text{Query} \cdot \text{Key}}{\sqrt{d_k}}$$

3) Softmax를 통한 가중치 계산

- 스케일링된 값을 Softmax 함수에 통과시켜 각 Key에 대한 가중치를 계산합니다.
- Softmax는 모든 가중치의 합이 1이 되도록 조정해 주며, 가장 중요한 Key의 가중치가 가장 높아지도록 합니다.
- 이 가중치 값들이 바로 **Attention Score**로, Query가 Key에 얼마만큼 주목해야 하는지를 의미합니다.

$$\text{Attention Weights} = \text{Softmax}\left(\frac{\text{Query} \cdot \text{Key}}{\sqrt{d_k}}\right)$$

$d\ k$ 는 Key의 차원을 의미하며, 이를 통해 스코어를 안정화합니다.

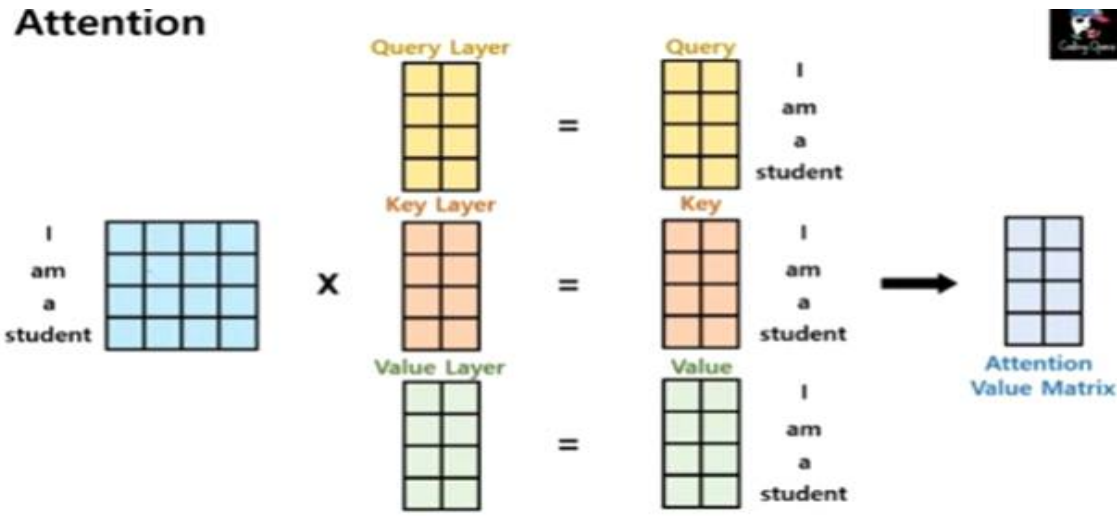
4. Weighted Sum (가중합)

▷ 마지막으로, 계산된 가중치(Attention Score)를 Value 벡터에 곱해주고, 이를 합산하여 최종 컨텍스트 벡터를 얻습니다.

▷ 이 벡터는 주어진 Query에 가장 적합한 정보로, 모델이 다음 예측을 할 때 사용하는 정보입니다.

$$\text{Attention Output} = \sum(\text{Attention Weights} \times \text{Value})$$

- 컨텍스트 벡터 생성: Attention 메커니즘은 가중치를 기반으로 중요한 정보에 더 많은 비중을 두어 입력 시퀀스의 정보를 합칩니다. 이로써, 현재 예측해야 할 단어와 관련성이 높은 입력 단어들에 집중할 수 있습니다.
- 출력 생성: 이렇게 가중치를 적용한 입력 정보와 이전에 예측된 정보를 종합하여 최종 출력을 생성합니다.



사례로 이해하기 : '나는 밥을 먹었다' 를 영어로 번역하는 사례

"나는 밥을 먹었다"라는 문장을 영어로 번역하는 과정을 Seq2Seq 모델의 Attention 메커니즘을 통해 설명해 보겠습니다. 영어 번역은 **"I ate rice"**가 될 텐데, 이 과정에서 Attention 메커니즘이 어떤 식으로 작동하는지 살펴보겠습니다.

사례 문장: "나는 밥을 먹었다" 번역하기

1. Query, Key, Value 생성

문장의 각 단어: "나는", "밥을", "먹었다"는 **인코더를 통해 벡터로 임베딩**됩니다.

이때, 디코더는 "I ate rice"를 생성하는 과정에서 각 단어가 번역될 때마다 필요한 정보를 효율적으로 찾기 위해 Attention을 적용합니다.

- Query: 디코더는 "ate"라는 단어를 생성할 차례에 해당하는 시점에 다음에 어떤 단어가 나와야 할지 결정해야 합니다. 이때, 디코더는 번역 과정에서 현재까지 생성된 정보를 바탕으로 현재 시점에 필요한 정보를 찾기 위한 "질문"을 던지게 되는데, 이 질문에 해당하는 것이 바로 Query입니다.

구체적으로

- 디코더의 현재 상태: 디코더는 지금까지 번역해온 단어들("I"까지 생성한 상태)을 반영하여 현재 시점에서 필요한 단어가 무엇인지를 파악합니다.
- Query의 역할: 이때 디코더의 현재 상태가 Query가 되며, Query는 "ate"라는 단어를 번역하는 데 필요한 정보를 찾는 데 사용됩니다.
- Query와 Key의 매칭: 이 Query는 인코더가 입력 문장("나는 밥을 먹었다")에서 생성한 Key들과 비교되어, 어느 Key가 "ate"라는 단어를 번역하는 데 가장 유용한지를 판단하게 됩니다.
- 어느 Key에 주목할지 결정: 예를 들어, "ate"라는 단어는 "먹었다"라는 단어와 의미상 유사하므로, "먹었다"에 해당하는 Key와의 유사도가 높을 것입니다. 이로 인해 디코더는 "ate"라는 단어를 생성할 때 "먹었다"에 가장 주목하게 됩니다.
- 따라서, 디코더의 현재 상태가 Query가 되며, 이 Query를 통해 어떤 Key에 주목할지 결정하여 번역에 필요한 정보를 찾게 됩니다.
- Key & Value: "나는", "밥을", "먹었다" 각각이 Key와 Value로 표현됩니다.
- Key는 각 단어가 가진 고유 특성을, Value는 실제 번역에 필요한 정보를 담고 있습니다.

2. **Query와 Key의 내적 (유사도 계산)**

Query(현재 번역할 "ate")와 각 Key("나는", "밥을", "먹었다") 간 dot product을 계산하여 유사도를 측정합니다.

이 예에서, "ate"라는 단어는 "먹었다"와 유사도가 높을 가능성이 큽니다.

반면, "나는"이나 "밥을"과의 유사도는 낮을 수 있습니다.

3. 스케일링과 Softmax

내적으로 얻은 값들을 Key의 차원 수로 나누어 값을 스케일링하여 학습의 안정성을 높입니다.

스케일링된 값을 Softmax 함수에 통과시켜 각 Key에 대한 가중치를 구합니다.

예를 들어, "ate"는 "먹었다"에 높은 가중치를 주고, "나는"과 "밥을"에는 상대적으로 낮은 가중치를 줄 것입니다.

4. Value의 가중합 계산 (최종 context vector)

Softmax로 얻은 가중치를 각 Value에 곱한 후 가중합을 계산하여 최종 context vector를 만듭니다.
이 context vector는 "ate"라는 단어를 생성할 때 필요한 정보("먹었다"라는 의미)가 담겨 있으며, 이를 통해 디코더가 "ate"라는 단어를 생성할 수 있게 됩니다.
이후 과정
마찬가지로, "rice"를 생성할 때는 "밥을"에 주목해야 하므로 "밥을"에 더 높은 가중치가 주어지고, context vector는 이를 반영하게 됩니다.
이러한 Attention 메커니즘 덕분에 Seq2Seq 모델은 문맥에 맞는 번역을 위해 각 단어가 주목해야 할 정보를 효과적으로 선택하여 사용할 수 있게 됩니다.

성과와 영향

- Attention 메커니즘은 머신 번역뿐만 아니라 텍스트 생성, 음성 인식, 이미지 캡셔닝 등 다양한 분야에서 성능을 획기적으로 향상시켰습니다.
- 특히 Transformer 모델(2017년 발표)과 BERT, GPT 등의 대형 언어 모델들에 이르기까지, Attention은 핵심적인 구성 요소로 자리 잡았습니다.
- 이 메커니즘 덕분에 자연어 처리 모델들은 시퀀스 길이에 대한 제한 없이 더 정확한 번역과 문맥 파악을 할 수 있게 되었고, 이후로도 딥러닝의 많은 분야에서 필수적인 요소로 사용되고 있습니다.

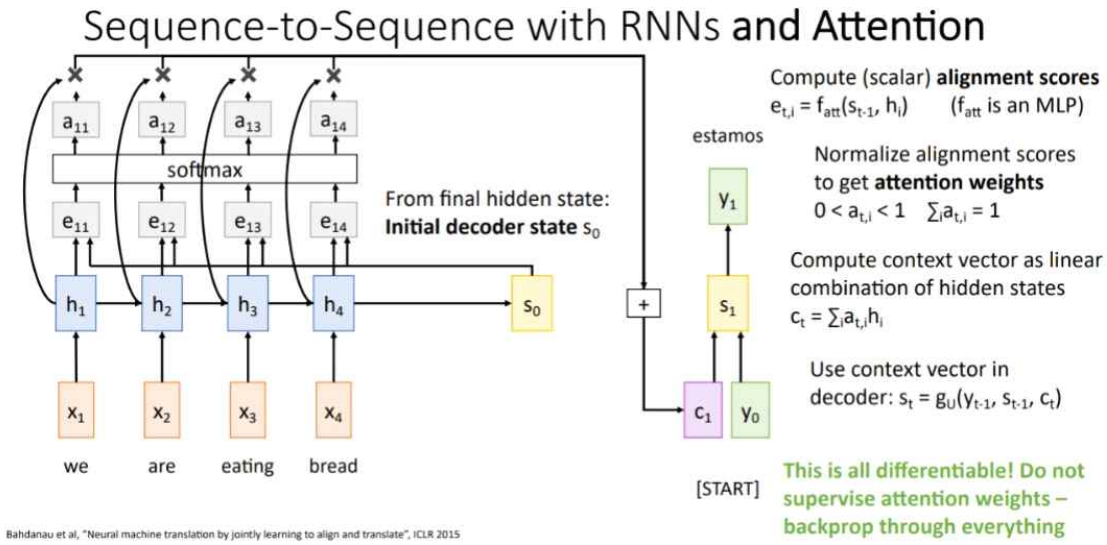
1)

2)

Seq2Seq with RNN and Attention (2015)

2)

Attention을 적용한 seq2seq의 Decoder의 매 step별 context vector를 재 생성하는 Attention mechanism



1)

우측 상단의 식처럼 **alingment function**을 거쳐 e t,i 의 (scalar) score를 뽑아내게 된다.

이는 s t 와 h i 를 input으로 받는 fully connected network이다.

Alignment Function이 Attention Mechanism에서 하는 일

- Alignment Function은 Query와 Key 간의 유사도를 계산하여, 특정 Query가 어느 Key에 집중할지 결정하는 함수입니다.
- Dot Product, Additive, Cosine Similarity 등 다양한 방식이 있으며, 이 값은 Softmax를 통해 Attention Score가 됩니다.
- 최종적으로, Alignment Function은 context vector가 번역 과정에서 필요한 정보를 효과적으로 반영하도록 돕습니다.

다양한 attention mechanism마다 alingment function은 다르지만 dot product attention은 다음과 같은 alingment function이 사용된다.

$$e_{t,i} = score(s_t, h_i) = s_t^T h_i$$

이때 output score는 decoder의 현재 t시점에서 encoder의 각 state에 집중하는 정도를 나타내게 된다(각 e t,i)

2)
이전의 output score (e t,i)를 softmax를 통해 0~1사이의 값을 갖는 probability distribution으로 나타내고 이러한 softmax의 output을 attention weights 라고 하고 이는 각 hidden state에 얼마나 가중치를 둘 것이냐를 나타낸다.

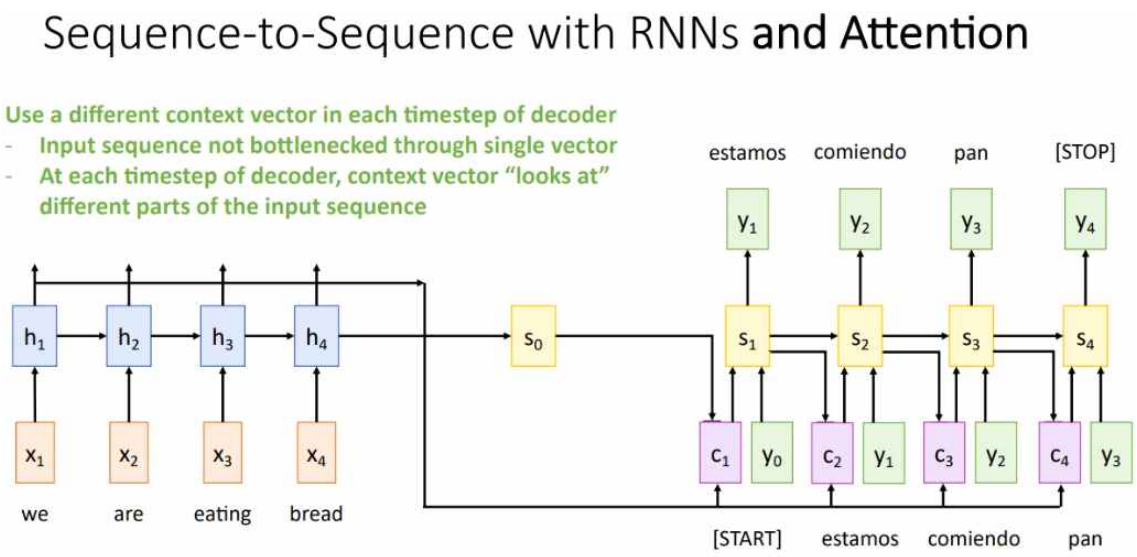
Attention weights와 각 hidden state를 weighted sum해주어 (∑ a t,i h i) t시점의 새로운 context vector c t 를 구해준다.
Decoder는 이러한 context vector c t 와 input y t−1 이전 state인 h t−1 을 사용하여 t시점의 state s t 를 생성하고 이를통해 predict된 output을 생성해준다.

위 그림의 computational graph에 나타나는 모든 연산들은 differentiable하기 때문에 backprop이 가능하고 이로 인해 network가 알아서 decoder의 time step별 input의 어느 state에 attention 해야 하는지 학습하게 된다.

이러한 과정을 decoder의 time step마다 반복해서 사용하고 기존의 seq2seq처럼 decoder가 STOP token을 출력하면 멈추게 된다.

Seq2Seq with RNN and Attention 모델과 2017년 트랜스포머(Transformer) 모델 비교

Seq2Seq with RNN and Attention 모델과 2017년 트랜스포머(Transformer) 모델은 모두 시퀀스 데이터 처리에서 강력한 성능을 발휘하는 모델들이지만, 그 구조와 처리 방식에서 중요한 차이점들이 있습니다.



1. 기본 아키텍처

1) Seq2Seq with RNN and Attention

- 인코더와 디코더 구조: Seq2Seq 모델은 전통적으로 RNN, LSTM 또는 GRU를 기반으로 한 인코더와 디코더로 구성됩니다. 인코더는 입력 시퀀스를 처리하여 은닉 상태(hidden state)를 만들고, 디코더는 이 은닉 상태를 받아 새로운 시퀀스를 생성합니다.
- Attention 메커니즘 추가: Attention 메커니즘이 추가되면서, 디코더가 인코더의 마지막 은닉 상태 하나만 참조하는 것이 아니라, 입력 시퀀스의 모든 은닉 상태들을 동적으로 참조할 수 있게 되었습니다. 이로 인해 디코더는 입력의 특정 부분에 집중할 수 있으며, 장기 의존성 문제와 정보 손실을 완화할 수 있습니다.

2) Transformer (2017)

- **RNN 제거:** Transformer 모델은 RNN, LSTM, GRU 같은 순환 구조를 완전히 제거했습니다. 대신 Self-Attention 메커니즘을 기반으로 데이터를 처리합니다.
- 완전히 병렬화된 처리: RNN 기반 모델은 데이터를 시간에 따라 순차적으로 처리해야 하는 특성이 있지만, Transformer는 시퀀스 내 모든 입력을 병렬로 처리할 수 있습니다. 이를 통해 훈련 속도와 효율성이 크게 향상됩니다.
- 레이어 구조: Transformer는 Multi-Head Attention과 피드포워드 신경망을 포함하는 여러 층을 통해 입력을 처리합니다. 각 층은 Self-Attention을 사용하여 모든 단어가 다른 단어들과의 관계를 학습할 수 있게 합니다.

2. Attention 메커니즘의 차이

1) Seq2Seq with Attention

- 인코더-디코더 Attention: 이 구조에서는 디코더가 출력을 생성할 때 인코더에서 나온 은닉 상태들(입력 시퀀스의 각 단어의 표현)에 가중치를 부여하여 어느 부분에 집중할지를 결정합니다.
- 디코더는 시간에 따라 이전 시점의 예측과 인코더의 정보를 참조하면서 출력을 생성하는데, Attention은 디코더가 모든 인코더의 출력(은닉 상태)을 참조하여 그 중 중요한 정보

를 더 잘 활용할 수 있게 해줍니다.

2) Transformer

- Self-Attention: Transformer에서는 입력 시퀀스의 각 단어가 모든 다른 단어들과의 관계를 학습합니다. 즉, Self-Attention을 통해 입력 시퀀스의 각 단어가 시퀀스 내의 다른 단어들과 얼마나 연관되어 있는지를 평가합니다.
- Multi-Head Attention: Self-Attention 메커니즘을 여러 번(보통 8번 또는 12번) 병렬로 수행하는 Multi-Head Attention이 사용됩니다. 각 헤드는 다른 부분에 집중할 수 있어 시퀀스 내에서 다양한 패턴을 학습할 수 있게 합니다.
- 인코더와 디코더의 Attention: Transformer의 디코더는 Self-Attention을 먼저 수행한 후, 인코더에서 나온 출력에 대해 인코더-디코더 Attention을 수행하여 입력 시퀀스에 집중합니다. 이 구조는 Seq2Seq 모델과 유사하지만, 모든 부분이 병렬로 처리된다는 점이 다릅니다.

3. 입력 처리 방식

1) Seq2Seq with RNN and Attention

- 순차적 처리: RNN 기반 Seq2Seq 모델은 입력을 시간 순서대로 처리합니다. 입력 시퀀스가 주어지면, 인코더는 한 단어씩 처리하여 은닉 상태를 갱신합니다. 이 순차적 처리 특성으로 인해 시퀀스가 길어지면 처리 시간이 오래 걸릴 수 있습니다.
- 상태 유지: RNN 구조는 각 시점에서 이전 시점의 정보를 유지해야 하므로, 모델이 시퀀스의 순서를 유지하며 데이터를 처리합니다.

2) Transformer

- 병렬 처리: Transformer는 모든 입력 시퀀스를 동시에 처리할 수 있습니다. 이는 Self-Attention 메커니즘 덕분인데, 각 입력 단어가 독립적으로 처리되기 때문에 모델 훈련 속도가 RNN 기반 모델보다 훨씬 빠릅니다.
- 포지셔널 인코딩: Transformer는 RNN처럼 순차적인 구조가 없기 때문에, 입력 시퀀스 내 단어들의 순서 정보를 포지셔널 인코딩(Positional Encoding)이라는 방식을 통해 추가합니다. 이를 통해 단어 간의 상대적 위치 정보를 모델에 제공할 수 있습니다.

4. 병렬화 및 효율성

1) Seq2Seq with RNN and Attention

- 병렬 처리 불가: RNN 기반 Seq2Seq 모델은 입력 시퀀스를 순차적으로 처리하기 때문에 병렬 처리가 어렵습니다. 이로 인해 긴 시퀀스를 처리하는 데 시간이 오래 걸리고, 훈련 속도가 느립니다.
- 훈련 및 추론 비용: 시간이 지남에 따라 각 단계에서 정보를 전달하고 업데이트하는 비용이 커질 수 있습니다.

2) Transformer

- 완벽한 병렬 처리 가능: Transformer는 Self-Attention을 사용해 입력 시퀀스를 병렬로 처리할 수 있으므로, 긴 시퀀스도 빠르게 처리할 수 있습니다. 이는 GPU 및 TPU 같은 병렬 처리가 가능한 하드웨어 환경에서 큰 성능 향상을 가져옵니다.
- 훈련 효율성: RNN 기반 모델과 비교해 훈련 시간이 크게 단축됩니다. 이는 특히 매우 긴 시퀀스나 대규모 데이터셋을 처리할 때 중요한 장점입니다.

5. 성능 차이

1) Seq2Seq with RNN and Attention

- 일정 수준의 성능: Attention 메커니즘 덕분에 RNN 기반 Seq2Seq 모델도 상당히 좋은 성능을 발휘하지만, 여전히 순차적 처리로 인한 제약이 남아 있습니다.
- 기억력 한계: RNN은 기본적으로 장기 의존성 문제를 가지고 있으며, Attention 메커니즘이 이를 완화하지만 완벽히 해결하지는 못합니다.

2) Transformer

- 뛰어난 성능: Transformer는 RNN의 장기 의존성 문제를 완전히 제거했으며, 특히 긴 시퀀스에서도 훨씬 안정적이고 강력한 성능을 발휘합니다. 자연어 처리(NLP) 및 기타 시퀀스 처리 작업에서 Transformer 기반 모델들이 현재 최상의 성능을 보이고 있습니다.
- 스케일링 가능성: 대규모 데이터와 모델을 사용해도 Transformer는 병렬 처리 덕분에 매우 효율적으로 확장 가능합니다. 이를 통해 GPT, BERT 같은 대형 언어 모델들이 탄생하게 되었습니다.

(주석 정리)

주1. Bottleneck 현상과 Long-Term Dependencies 문제

RNN(순환 신경망) 및 Seq2Seq 모델과 같은 시퀀스 처리 모델에서 발생하는 중요한 두 가지 문제로, 특히 긴 시퀀스를 처리할 때 그 성능에 영향을 미칩니다.

1. Bottleneck 현상 (병목 현상)

Bottleneck 현상은 시퀀스 데이터 처리 중에 발생하는 정보 압축 문제를 의미합니다.

이 문제는 특히 Seq2Seq 모델에서 인코더와 디코더 간에 정보를 주고받는 과정에서 발생합니다.

발생 원인

- Seq2Seq 모델의 기본 구조에서, 인코더는 입력 시퀀스를 처리하여 마지막 은닉 상태를 컨텍스트 벡터로 출력합니다.
- 이 컨텍스트 벡터는 입력 시퀀스의 모든 정보를 하나의 고정된 벡터로 압축하여 디코더에 전달하는 역할을 합니다.
- 하지만, 입력 시퀀스가 매우 길거나 복잡한 경우, 하나의 고정된 크기의 벡터에 전체 정보를 압축하는 것이 비효율적일 수 있습니다.
- 그 결과, 중요한 정보가 소실되거나 정확한 예측에 필요한 세부 정보들이 누락될 수 있습니다.

결과

- 컨텍스트 벡터가 충분히 정보를 담지 못하면, 디코더는 불완전한 정보를 기반으로 출력을 생성하게 됩니다.
- 이는 번역, 요약 등의 시퀀스 처리 작업에서 성능 저하로 이어질 수 있습니다.
- 긴 시퀀스나 복잡한 문장을 처리할 때 Bottleneck 현상이 두드러지며, 정확한 결과를 도출하기 어려워집니다.

2. Long-Term Dependencies 문제 (장기 의존성 문제)

Long-Term Dependencies 문제는 RNN 기반 모델이 긴 시퀀스에서 앞부분의 정보를 적절히 기억하지 못하는 문제를 말합니다.

이는 RNN이 시간에 따라 상태를 갱신할 때, 오래된 정보가 소실되거나 희미해지는 현상과 관련이 있습니다.

(발생 원인)

RNN은 시간에 따라 순차적으로 입력 데이터를 처리합니다. 이전 시간 단계에서의 은닉 상태가 다음 단계의 입력과 함께 갱신되며, 이 과정이 반복됩니다.

하지만, 시간이 지나면서 RNN은 기억해야 할 정보가 많아지게 되며, 특히 중요한 초반의 정보가 뒤로 갈수록 잊혀지기 시작합니다. 이는 기울기 소실 문제(Vanishing Gradient Problem)와도 관련이 있습니다.

따라서, 모델이 멀리 떨어진 시점에서의 중요한 정보를 기억하지 못하게 되어, 시퀀스 내에서의 장기 의존성을 학습하기 어렵습니다.

결과

- 예를 들어, 긴 문장이나 문서 내에서 앞부분의 정보가 뒷부분의 단어 예측에 중요한 경우, RNN이 이 정보를 잘 기억하지 못해 부정확한 예측이 발생할 수 있습니다.
- 기계 번역에서 첫 번째 문장의 주어가 문장 마지막의 동사와 연관되어 있을 때, 이 관계를 모델이 잘 파악하지 못하는 경우가 많습니다. 즉, 문맥 정보가 소실되면 결과가 왜곡됩니다.

해결 방법

- LSTM(Long Short-Term Memory)와 GRU(Gated Recurrent Unit):

▷ LSTM과 GRU는 RNN의 장기 의존성 문제를 해결하기 위해 개발된 모델들입니다.

▷ 이들 모델은 게이트 메커니즘을 사용하여 중요한 정보를 더 오래 유지하고, 필요 없어진 정보는 삭제합니다.

▷ LSTM은 Cell State를 통해 정보를 장기적으로 유지하고, 게이트를 통해 정보를 적절히 저장, 업데이트, 삭제하는 과정을 제어합니다.

▷ 이를 통해, 긴 시퀀스에서도 중요한 정보가 잘 보존됩니다.

- Attention 메커니즘:

▷ Attention 메커니즘은 Bottleneck 현상과 Long-Term Dependencies 문제를 모두 해결하는 데 도움이 됩니다.

▷ Attention은 디코더가 출력을 생성할 때 입력 시퀀스의 모든 은닉 상태에 집중할 수 있도록 하여, 각 시점에서 필요한 정보를 선택적으로 참조할 수 있게 합니다. 이를 통해, 입력 시퀀스 전체에서 중요한 부분에 집중할 수 있으며, 정보가 하나의 고정된 벡터로 압축되는 문제를 피할 수 있습니다.

▷ 특히 Transformer 모델에서는 RNN 대신 Self-Attention을 사용하여 입력 시퀀스의 모든 부분이 서로의 중요성을 평가할 수 있도록 합니다. 이를 통해 장기 의존성을 잘 처리할 수 있습니다.

파이껀 실습

어텐션 참조