2-1. 자연어 처리 기본

목차

1. 워드 임베딩과 순환신경망 기반 모델(RNN & LSTM)

학습 목표

- 단어를 숫자로 표현하는 기본 아이디어들을 이해한다.
- 언어에서 ‘순서(문맥)’의 중요성을 설명할 수 있다.
- RNN의 기본 구조와 역할을 이해한다.
- LSTM의 핵심 아이디어를 설명할 수 있다.

학습 시작(Overview)

- 언어를 어떻게 숫자로 표현할까?
    - One-hot encoding, word embedding 등 기본 아이디어 이해
- 언어에서 ‘순서(문맥)’은 왜 중요하고 어떻게 다뤄야 할까?
    - 단어 순서와 의미의 연결성
- 순환 신경망(RNNs, LSTMs 등)은 어떻게 동작할까?
    - 기본 구조와 역할
    - 장기 의존성 문제와 LSTM의 핵심 아이디어

1. 단어를 숫자로 표현하기 : 워드임베딩

1-1. 원-핫 인코딩

- 규칙 기반 혹은 통계적 자연어처리 연구의 대다수는 단어를 원자적(쪼갤수 없는) 기호로 취급한다.
- 벡터 공간 관점에서 보면, 이는 한 원소만 1, 나머지는 모두 0인 벡터를 의미

    - ex) [0 0 0 0 1 0 0 0]
- 차원 수(=단어 사전크기) 는 대략 다음과 같다.
    - ex) 음성 데이터(2만), 코퍼스(5만), big vocab(50만)
- 이를 원-핫 표현이라고 부르고, 단어를 원-핫 표현으로 바꾸는 과정을 원-핫 인코딩이라 한다.

원-핫 인코딩의 문제점

- 조금만 달라져도 아예 다른 단어 취급 → 유사한 단어끼리의 유사성 측정 불가
- 차원의 저주 → 고차원의 희소벡터를 다루기 위해선 많은 메모리 필요, 차원이 커질 수록 데이터가 점점 더 희소(sparse) 해셔 활용이 어렵다.
- 의미적 정보 부족 → 비슷한 단어라도 유사한 벡터로 표시 안됨

1-2. 워드 임베딩

단어를 주변 단어들로 표현하면, 많은 의미를 담을 수 있다.
→ 현대 통계적 자연어 처리에서의 가장 성공적인 아이디어 중 하나

워드 임베딩 : 단어를 단어들 사이의 의미적 관계를 포착할 수 있는 밀집(dense)되고 연속적/분산적(distributed) 벡터 표현으로 나타내는 방법

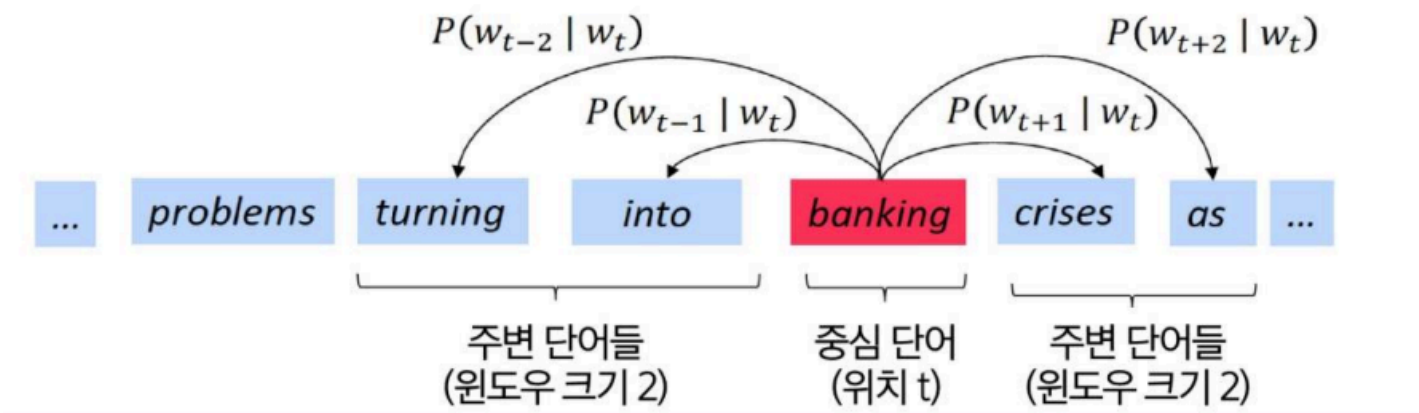
→ 유사한 단어끼리 공간상 서로 가깝게 위치하여, 의미적 유사성을 반영할 수 있다.

Word2Vec (2013, Google)

- 대표적인 워드 임베딩 기법
- 단어의 표현을 간단한 인공 신경망을 이용해 학습한다.
- 아이디어 : 각 단어와 그 주변 단어들 간의 관계를 예측
- 알고리즘
    1. Skip-grams (SG) 방식
        - 중심 단어를 통해 주변 단어들을 예측
        - 단어의 위치(앞/뒤)에 영향 받지 않음
    2. Continous Bag of Words(CBOW) 방식
        - 주변 단어들을 통해 중심 단어를 예측하는 방법
        - 문맥 단어들의 집합으로 중심 단어를 맞춘다.

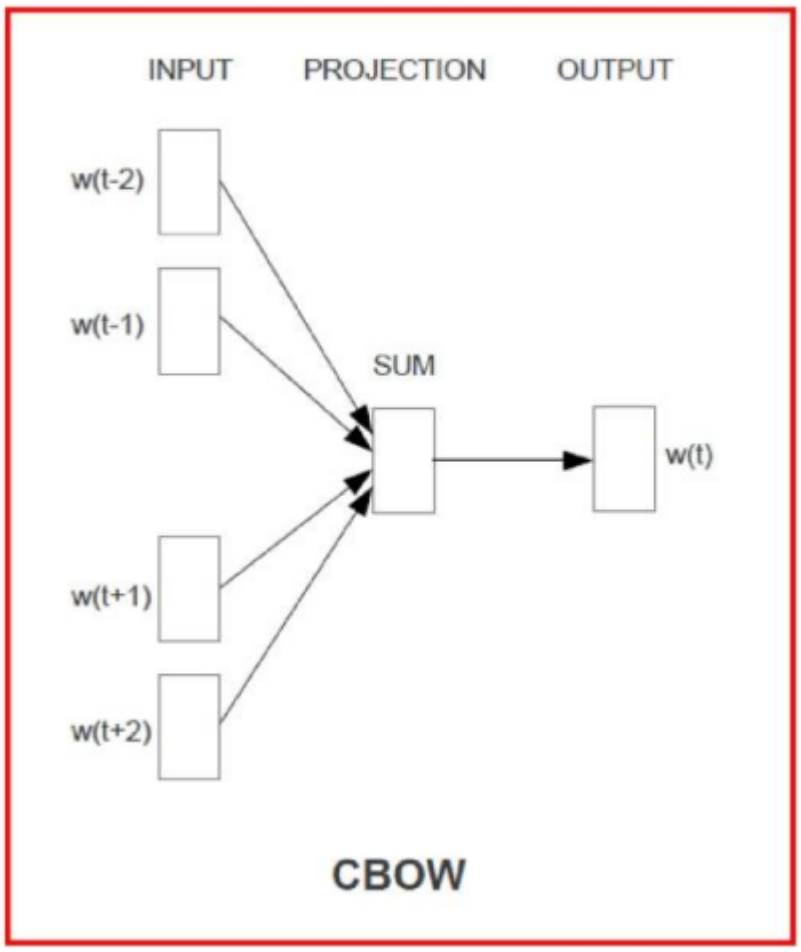
Skip-Grams(SG)

- 윈도우 크기(window size) = 중심 단어 주변 몇 개 단어를 문맥으로 볼 지 결정



CBOW

- 목표 : 주변 단어들의 집합이 주어졌을 때, 그 문맥과 함께 등장할 수 있는 단일 단어를 예측



모델	장점	한계
Skip-Gram	- 적은 데이터에도 잘 동작한다. - 희귀 단어나 구 표현에 강하다.	- 학습 속도가 느리다.
CBOW	- 학습 속도가 빠르다. - 자주 나오는 단어에 강하다.	- 희귀 단어 표현에 약하다.

2. 순차적 데이터

순차적 데이터 : 데이터가 입력되는 순서와 이 순서를 통해 입력되는 데이터들 사이의 관계가 중요한 데이터

- ex) 오디오/ 텍스트/비디오

순차적 데이터의 특징

1. 순서가 중요하다
    - 데이터의 순서가 바뀌면 의미가 달라진다.
2. 장기 의존성(Long-term dependency)
    - 멀리 떨어진 과거의 정보가 현재/미래에 영향을 준다.
3. 가변 길이(Variable length)
    - 순차 데이터의 길이는 일정하지 않고, 단어 수도 제각각이다.

순차적 데이터를 처리하기 위해서는 일반적인 모델(선형회귀, MLP)로 불가능

→ Sequential Models 가 필요 (RNN, LSTM, Transform 등)

3. RNN

전통적인 인공신경망인 MLP,CNN은 고정된 길이의 입력을 처리하기 때문에 가변 길이의 데이터를 처리하기에 적합하지 않다.

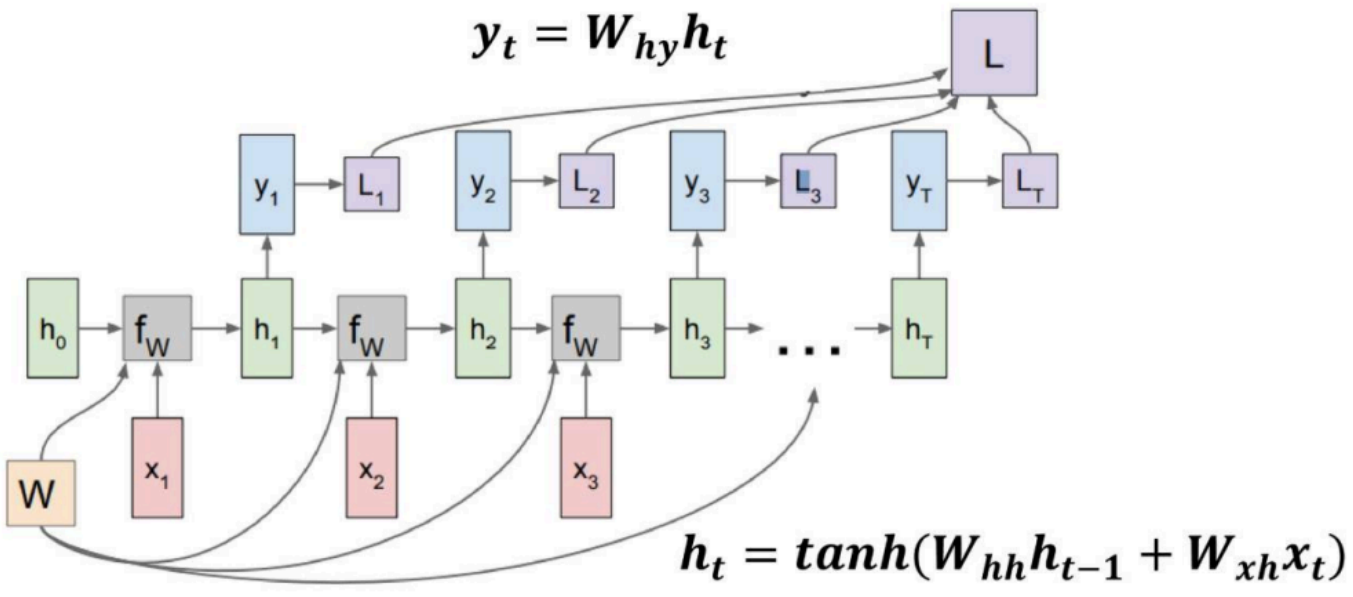
RNN

- 가변 길이의 입력을 받을 수 있고, 이전 입력을 기억할 수 있기 때문에 순차적 데이터 처리에 적합한 아키텍쳐이다.
  - ex) 감정 분류 (Sentiment Classification) , Image Captioning, Machine Translation

RNN 아키텍처 설명

- 전통적인 신경망(MLP, CNN)등과 달리, RNN은 이전 시점의 정보를 담은 hidden state가 존재
- 입력 시퀀스 벡터 x를 처리할 때마다, 각 시점마다 recurrence 수식을 적용하여 hidden state를 업데이트 한다.
  - ex) ht = fw(ht-1, xt)
    - ht : 새로운 hidden state
    - fw : 처리 함수(파라미터 W)
    - ht-1 : 이전 hidden state
    - xt : 입력 벡터
- RNN은 각 시점마다 동일한 가중치(weight)를 사용하기 때문에, DNN(Deep feedforward Neural Network) 보다 파라미터 효율적이다.

RNN 구조



RNN의 특징

- RNN은 한번에 하나의 요소를 처리하고, 정보를 앞으로 전달한다.
- 펼쳐서 보면, RNN은 각 층이 하나의 시점을 나타내는 깊은 신경망처럼 보인다
- RNN은 hidden state를 유지하면서 가변 길이 데이터를 처리할 수 있다.
- RNN의 출력은 과거 입력에 영향을 받는 다는 점에서 FeedForward 신경망과 다르다.

RNN의 한계

- 기울기 소실(Gradient Vanishig)
  - 역전파 과정에서 작은 값들이 계속 곱해져, 기울기가 0이 되는 현상
  - 장기 의존성 학습이 불가능해지고, 단기 의존성만 포착
- 기울기 폭발(Gradient Explosion)
  - 반대로 계속 큰 값이 곱해져 기울기가 터지는 현상

4. LSTM : 더 오래 기억하는 신경망

LSTM : RNN의 기울기 소실 문제를 해결하기 위해 제안된 RNN의 한 종류

LSTM의 특징

- Cell state의 존재
  - long-term information을 저장하는 cell state의 존재
  - LSTM은 cell state에서 정보를 읽고, 지우고, 기록할 수 있다.

  New cell content

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

- Gate의 존재
  - Forget gate : 이전 cell state에서 무엇을 버리고, 무엇을 유지할 지 결정

$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

  - Input gate : 새 정보 중 얼마나 cell state에 쓸지 결정

$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

  - Output gate : cell state 중 얼마나 hidden state로 내보낼지 결정

$$o_t = \sigma\left(W_o \; [h_{t-1}, x_t] \; + \; b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

  - 매 시점마다 게이트의 각 요소는 열림(1), 닫힘(0) 혹은 그 사이의 값으로 결정
  - 게이트는 동적으로 계산되며, 현재 입력과 hidden state 등 문맥에 따라 값이 정해진다.

2. 자연어 생성 모델 (Seq2Seq, Attention)

학습 목표

- 언어 모델의 개념과 역할을 설명할 수 있다
- 언어 모델이 문맥을 바탕으로 단어의 확률을 에측하는 방식을 이해한다.
- Seq2Seq 구조의 기본 아이디어(인코더-디코더)를 설명할 수 있다
- Attention 메커니즘의 개념과 필요성을 설명할 수 있다

학습 시작(오버뷰)

- 언어 모델이란 무엇일까?

    - 언어 모델의 정의와 역할
    - 단어 예측을 통한 문맥 이해
- Seq2Seq은 어떻게 동작할까?
    - 인코더-디코더 구조
    - 입력과 출력 길이가 다른 시퀀스 처리
- Attention은 왜 필요할까?
    - Bottleneck problem 해결
    - 중요한 단어에 집중하는 메커니즘

1. 언어 모델이란?

언어 모델 : 인간의 두뇌가 자연어를 생성하는 능력을 모방한 모델

- 단어 시퀀스 전체에 확률을 부여하여 문장의 자연스러움을 측정
- 한 문장의 확률은 각 단어의 조건부 확률들의 곱으로 표현할 수 있다

ex) 단어 자동완성 기능

N-gram 언어모델

- 대표적인 언어모델
- 텍스트에서 단어가 나올 확률을 계산할 때 앞선 N-1개의 단어를 고려해서 다음 단어의 확률을 예측하는 통계적 언어 모델

Statistical Machine Translation (SMT)

- 언어모델 예시
- 1990~2010년까지 기계번역을 통계학적으로 접근
    - ex) 한국어 → 영어 번역
    - 번역모델, 언어 모델로 나눔
    - 번역모델 : 단어와 구가 어떻게 번역되어야 하는 지 모델링, (한국어, 영어) 말뭉치로 학습
    - 언어모델 : 유창한 영어문장을 쓰는 방법을 모델링, 영어 데이터로 학습

SMT의 한계

- 구조적 복잡성
- 많은 수작업
- 언어 별 자원 구축 필요

→ 유지 및 확장에 어려움 존재

→ 기계 번역 연구는 SMT 에서 Neural Machine Translation(NMT)로 넘어감

2. Seq2Seq

Neural Machine Translation(NMT)

- 인공 신경망을 이용해 기계 번역을 수행하는 방법
- Seq2Seq 신경망 구조를 사용, 두 개의 RNN으로 이뤄짐
- 2014, Google에서 소개

번역 작업이 어려운 이유는 입력과 출력 길이가 다를 수 있기 때문

→ NMT에서는 길이가 다른 시퀀스 간의 매핑을 처리해야 했다

Seq2Seq 아이디어

- 2개의 LSTM을 이용
- LSTM1 : 입력 시퀀스를 한 타임스텝씩 읽어 고정된 차원의 큰 벡터 표현을 구한다. (Encoder)
- LMST2 : 앞에서 얻은 벡터로부터 출력 시퀀스를 생성 (Decoder)

Seq2Seq 아키텍처

- Encoder + Decoder
  - Encoder : 입력 문장에 담긴 정보를 인코딩한다.
  - Decoder : 인코딩 된 정보를 조건으로 하여 타겟 문장(출력)을 생성한다.

Seq2Seq의 다양한 적용

- 기계번역 이외에도 다양한 태스크에 적용 가능
- 요약 : 긴 길이의 문서를 읽고, 짧은 길이의 문장으로 요약
- 대화 : 사용자의 발화를 기반으로 맥락에 맞는 대답을 생성
- 코드 생성 : 자연어로 작성된 설명 혹은 명령어를 입력 받아 프로그래밍 코드 혹은 쿼리로 출력

Seq2Seq 학습 방법

- 인코더와 디코더는 하나의 통합 네트워크로 연결되어 있다.
  - 디코더에서 발생한 오차는 역전파를 통해 인코더까지 전달되어 전체 네트워크가 End2End로 동시에 최적화 된다.
- 학습 초반에는 모델의 예측 능력이 떨어지기 때문에 학습이 불안정하다
  - 모델이 스스로 예측한 단어 대신 정답 단어를 디코더 입력으로 강제로 넣는 Teacher Forcing 으로 훨씬 빠르고 안정적으로 수행한다.

Seq2Seq의 토큰 출력 방법

1. Greedy Inference

- 토큰을 출력하는 방법 중 하나, 각 단계에서 가장 확률이 높은 단어를 선택
- 단점 : 되돌리기 불가

2. Beam Search

- 매 단계마다 k개의 가장 유망한 후보 단어 유지
- 후보가 <EOS>에 도달하면, 완성된 문장으로 후보 문장 리스트 추가
- <EOS>문장이 충분히 모이면 탐색 종료
- 각 후보 문장들의 점수를 로그 확률의 합으로 구해 최종 선택

Seq2Seq의 한계

- The bottleneck Problem
  - 인코더는 입력 문장 전체를 하나의 벡터로 요약하는데, 마지막 hidden state에 문장의 모든 의미 정보가 담긴다.
  - 고정 길이 벡터 하나에 모든 문장의 의미를 압축하다보니 정보 손실이 발생 → bottleneck problem

3. Attention

Attention의 아이디어

- 디코더가 단어를 생서앟 때, 인코더 전체 hidden state 중 필요한 부분을 직접 참조할 수 있게 한다.
- 매 타입 스텝마다, 어떤 단어/구절에 집중할 지 가중치로 계산해, bottle neck 문제를 완화

Attention의 효과

- NMT 성능 향상
  - 디코더가 소스 문장 전체가 아닌, 필요한 부분에만 집중
- Bottleneck Problem 해결
  - 디코더가 인코더의 모든 hidden state에 직접 접근 가능
- Vanishing Gradient 문제 완화
  - Attention은 멀리 떨어진 단어도 직접 연결할 수 있음
- 해석 가능성 (Interpretability)
  - Attention 분포를 보면, 디코더가 어떤 단어를 생성할 때 어디에 집중했는 지 확인 가능

  → 모델의 의사결정 과정을 해석할 수 있는 단서
- 정렬 (Alignment)
  - 이전 기계번역에서는 단어 alignment 모델을 따로 학습해야 했으나 attention을 통해 단어와 단어 사이의 매핑 관계를 자연스럽게 학습한다.

Attention 요소

- Query와 Values
  - Seq2Seq에서 Attention을 사용할 때, 각 디코더의 hidden state와 모든 인코더의 hidden state 간의 관계를 query와 values 의 관계로 볼 수 있다.

Attention 과정

1. Query와 Values 사이 유사도 점수 계산
2. Softmax를 통해 확률 분포 얻기 (attention distribution)
3. 분포를 이용해 values를 가중합 → context vector (attention output)

3. Transformer

학습 목표

- Transformer의 등장 배경과 필요성을 설명할 수 있다
- Self-Attention의 기본 개념과 동작 원리를 이해한다.
- Transformer의 전체 구조(인코더,디코더 블록)을 설명할 수 있다

학습 시작(오버뷰)

- Transformer란 무엇이며, 왜 필요한가?
  - RNN의 한계와 Attention의 필요성
- Transformer는 어떻게 작동할까?
  - Self-Attention, Multi-head Attention, Feed-Forward 구조 등
- Transformer는 어떤 응용이 있을까?
  - 번역, 요약, 대화, 대형 언어 모델의 기반

1. Self-Attention

RNN이 꼭 필요한가?

- RNN은 정보를 hidden state 로 전달하며 순차적 의존성을 형성
- Attention은 필요한 순간, 입력 전체에서 직접 정보를 전달

→ Attention이 더 효율적이라면, 굳이 recurrence 가 필요할까?

RNN의 한계점

1. 장기 의존성
    - RNN은 왼쪽에서 오른쪽으로 순차 전개→ 먼 단어 쌍이 상호작용하려면 시퀀스 길이 만큼의 단계를 거쳐야 한다.
    - 길어진 길이만큼 기울기 소실/폭발 문제로 장기 의존성 학습이 어렵다.
    - 입력된 선형 순서를 강하게 반영하여 언어의 비선형적 의존성을 잡아내지 못함
2. 병렬화
    - Forward와 Backward pass 모두 시퀀스 길이 만큼 단계 필요
    - 순차적인 연산이 진행되어 병렬화 불가능
    - GPU 사용 불가, 대규모 데이터 학습에 비효율적

Attention

- 각 단어의 표현을 query로 두고, value 집합으로부터 필요한 정보를 직접 불러와 결합

→ 이 과정을 encoder-decoder 사이가 아니라 한 문장 내부에서 적용한다면?

→ Self-Attention

Self-Attention의 장점

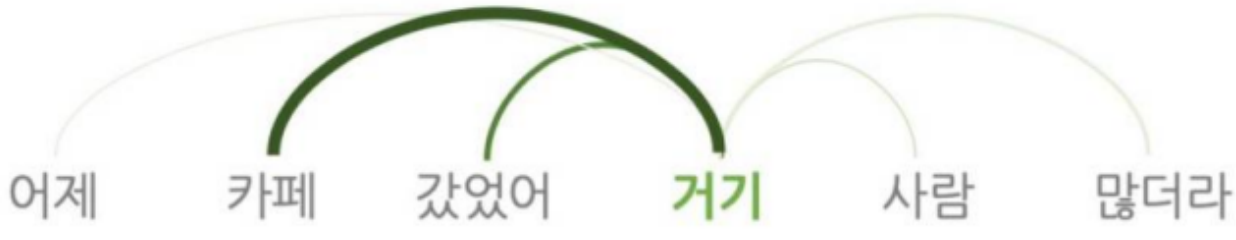
1. 순차적으로 처리해야하는 연산 수가 시퀀스 길이에 따라 증가하지 않음
2. 최대 상호작용 거리 ⇒ O(1) , 모든 단어가 각 층에서 직접 상호작용

Seq2Seq 에서의 Attention

- Attention(입력 문장 내의 단어들 | 출력 문장 내의 각 단어 'w' 가중치)

Self-Attention

- Attention(문장 내의 다른 단어들 | 문장 내의 각 단어'w' 가중치)



Self-Attention 의 요소

- Self-Attention에서 각 단어 i를 표현하는 Query, Key, Value 벡터 존재

1. 각 단어를 Query, Key, Value 벡터로 변환
    - Query 벡터 : 단어 i가 다른 단어로부터 어떤 정보를 찾을 지를 정의하는 벡터
    - Key 벡터 : 단어 i가 자신이 가진 정보의 특성을 표현하는 벡터
    - Value 벡터 : 실제로 참조되는 정보 내용을 담고 있는 벡터
2. Query, Keys 간의 유사도를 계산해, Softmax로 확률분포를 구한다.

$$e_{ij} = q_i^T k_j \qquad \alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{j'} \exp(e_{ij'})}$$

3. 각 단어의 출력을 Values 의 가중합으로 계산한다.

$$o_i = \sum_j \alpha_{ij} v_j$$

Self-Attention의 한계

1. 순서 정보 부재 → 단어간 유사도만 고려, 단어의 순서를 고려하지 않음
2. 비선형성 부족 → Attention은 선형 결합에 불과, 복잡한 패턴이나 깊은 표현력 담기 어려움
3. 미래 참조 문제 → 모든 단어를 동시에 참조하므로, 아직 생성되지 않아야 할 미래 단어를 참조한다.

Self-Attention의 한계 해결

1. Positional Encoding : 순서 정보 부재 문제를 해결하기 위해 도입
    - 각 단어 위치 i를 나타내는 위치 벡터를 정의해, 단어 임베딩 값에 더해 최종 입력으로 사용
    - 종류
        - Sinusoidal Position Encoding : 서로 다른 주기의 사인/코사인 함수를 합성해 위치 벡터 만들기
        - Learned Absolute Position Embedding : 위치 벡터를 모두 학습 파라미터로 설정해 학습 과정에서 데이터에 맞춰 최적화
2. Feed-Forward Network : 비선형성 부족 문제 해결을 위해 도입
    - 각 단어의 출력 벡터에 Feed-Forward Network(Fully Connected + RuLU)를 추가해 Self-Attention이 만든 표현을 깊고 선형적인 표현으로 확장
3. Masked Self-Attention : 미래 참조 문제 해결
    - Attention Score를 계산할 때, 미래 단어에 해당하는 항목을 -inf 로 설정해 계산을 수행할 때 반영되지 않게 한다.

Self-Attention 정리

- 문장 내 모든 단어가 서로 직접 상호작용하여 장거리 의존성을 효율적으로 포착, 병렬 처리가 가능하게 하는 메커니즘
- 한계 해결 방법
    1. 순서 정보 부재 → Positional Encoding
    2. 비선형성 부족 → Feed-Forward Network 추가
    3. 미래 참조 문제 → Masked Self-Attention

2. Transformer

Transformer (2017, Google에서 제안한 아키텍처)

- Self-Attention을 핵심 메커니즘으로 하는 신경망 구조
- Attention is All You Need

Transformer 는 encoder-decoder 구조로 설계된 신경망 모델이다.

- encoder : 입력 문장을 받아 의미적 표현으로 변환
- decoder : 인코더의 표현과 지금까지 생성한 단어들을 입력 받아, 다음 단어를 예측

  → decoder 가 언어 모델과 같은 방식으로 동작

Transformer의 구조

1. Multi-Headed Attention
   - 문장에서 같은 단어라도 여러 이유(문법적 관계, 의미적 맥락)로 다른 단어에 주목할 수 있다.

   → Self-Attention Head 하나로는 한 가지 관점에서 밖에 파악이 안됨

   → 여러 Attention Head를 두어 다양한 관점에서 동시에 정보를 파악

   ex)

   Attention Head1 : 단어의 문맥적 관계에 Attention

   Attention Head2 : 단어의 시제에 Attention

   Attention Head3 : 명사에 Attention

2. Scaled Dot Product

- Query와 Key의 차원이 커지면 두 벡터의 내적 값도 커짐

→ softmax함수의 출력이 뾰족해져 미세한 변화에도 큰 차이 발생

→ Gradient vanishing 발생

해결 : 내적 값을 그대로 사용하지 않고, 나눠서 스케일을 조정

→ 값이 안정적으로 분포되어 학습이 훨씬 빠르고, 안정적으로 진행

3. Residual Connection

- 깊은 신경망은 층이 깊어질수록 학습이 어려움 (기울기 소실/폭발)

→ 단순히 레이어 출력을 사용하면, 정보가 소실됨.

해결 : Layer가 전체를 예측하는 것이 아니라, 기존 입력과의 차이만 학습하도록 하는 Residual Connection 사용

4. Layer Normalization

- 층이 깊어질 수록 hidden vector 값들이 커졌다 작아졌다 하면서 학습이 불안해짐

→ 해결 : 각 레이어 단위에서 hidden vector 값을 정규화해서 안정적이고 빠른 학습을 돕는다.

5. Decoder

- Transformer의 디코더는 여러 개의 디코더 블록을 쌓아올 려 만든 구조
- 구성
  - Masked Self-Attention (Multi-Head)
    - 미래 단어를 보지 않도록 마스크를 씌움
  - Add & Norm (Residual Connection + Layer Normalization)
  - Feed-Forward Network
    - 각 위치에서 비선형 변환 수행

    - Add & Norm (Residual Connection + Layer Normalization)

→ 언어 모델처럼 단방향 문맥만 활용

6. Encoder
    - 양방향 문맥 모두 활용 가능
    - 입력 문장을 의미적 표현으로 변환
    - 각 단어가 양방향 문맥을 모두 반영한 벡터로 인코딩
    - 디코더와의 차이점 : Self-Attention 에서 Masking 제거

7. Encoder-Decoder
- 디코더는 단순 self-attention만 하는 것이 아니라, encoder의 출력 표현을 참조하는 cross-attention을 추가하여 입/출력을 연결

8. Cross-Attention
- Self-Attention과 다르게, Query는 decoder 에서, Key, Value는 encoder에서 가져온다.

Transformer의 결과
- NMT 에서 최고 성능
- 가장 효율적인 학습으로 비용 절감

Transformer와 사전학습(Pre-traning)
- Transformer의 등장으로 최신 모델들이 성능 향상을 위해 사전학습을 결합하게 됨
- 뛰어난 병렬 처리 능력 덕분에 NLP 표준 아키텍처가 됨

4. 사전 학습 기반 언어 모델

학습 목표
- 사전학습(Pre-training) 의 개념과 필요성을 설명할 수 있다
- Encoder 기반 모델(BERT)의 구조와 주요 활용 사례를 이해한다.
- Encoder-Decoder 기반 모델(T5)의 특징과 응용 분야를 설명할 수 있다.
- Decoder 기반 모델(GPT)의 구조와 강점을 이해한다.
- In-Context Learning(ICL)이 등장하게 된 배경과 그 의미를 설명할 수 있다.
- ICL 능력을 끌어올리기 위한 대표적인 프롬프팅 기법 (CoT, Zero-shot CoT)을 이해한다.

학습시작(오버뷰)
- 사전학습이란 무엇인가? 왜 중요한가?
    - 대규모 데이터 기반의 사전학습 개념과 필요성
- 대표적 모델 유형은 어떻게 구분되는가?
    - Encoder 기반, Encoder-Decoder 기반, Decoder 기반
- 각각 어떤 모델이 존재하는가?
    - BERT, T5, GPT 등 주요 모델 소개
- In-Context Learning과 발전된 Prompting 기법은 무엇인가?
    - ICL의 등장 배경과 의미

    - Chain-of-Thought, Zero-shot CoT 등 대표 기법

1. 사전 학습이란?

사전 학습(Pre-Training): 대규모 데이터 셋을 이용해, 모델이 데이터의 일반적인 특징과 표현을 학습하도록 하는 과정

ex) 언어모델 : 인터넷의 방대한 텍스트를 활용해 비지도학습 방식으로 학습, 일반적인 언어패턴, 지식, 문맥 이해능력을 습득

사전학습 관점에서의 워드 임베딩 vs 언어 모델

- 워드 임베딩의 경우 사전학습을 통해 단어의 의미를 학습하지만, 한계가 존재한다.

1. 다운스트림 태스크(ex, 텍스트 분류) 들에 적용하기엔 학습되는 데이터의 양이 적어 언어의 풍부한 문맥 정보를 충분히 학습할 수 없다.
2. 연결된 네트워크가 무작위 초기화 되어 학습 효율이 낮고, 많은 데이터와 시간이 필요

- 언어 모델의 경우 모델 전체를 사전학습을 통해 학습하기 때문에 강력한 NLP 구축에 이점이 있다.

1. 언어에 대한 표현 학습용으로 적합
2. 파라미터 초기화 관점에서 효과적
3. 언어에 대한 확률분포학습을 통해 샘플링, 생성에 사용 가능

언어 모델의 사전 학습

- 다음 단어의 확률 분포를 모델링하는 방법을 학습하여 사전학습 수행
- 인터넷의 대규모 텍스트 코퍼스에서 언어모델링 학습 수행
- 학습된 파라미터 저장해 다양한 다운스트림 태스크에 활용

사전학습 후 파인튜닝 하는 패러다임

- 사전학습을 통해 언어 패턴을 잘 학습한 파라미터로 초기화해 NLP 성능 향상

2. Encoder 모델

Encoder 모델의 사전학습

- Encoder 모델은 양방향 문맥을 모두 활용하여 전통적인 언어모델과의 차이점이 있다.
- 사전학습을 위해, 입력 단어의 일부를 [MASK] 토큰으로 치환한 뒤, 모델이 이 [MASK] 자리에 올 단어를 예측하는 학습 방법을 사용→ Masked Language Model
    - 대표적인 모델 : BERT

BERT(2018, Google)

- Masked LM 방법으로 사전학습을 수행하는 Transformer 기반 모델
- 학습 방법

1. Masked LM
    - 입력 토큰의 15%를 무작위 선택
    - [MASK] 토큰 치환 (80%), 랜덤 토큰 치환(10%), 그대로두기(10%)

    → 다양한 문맥 표현을 학습하여, 더 강건한 (robust) 표현을 학습하게 함

2. Next Sentence Prediction (NSP)

- 두번째 문장이 첫번째 문장의 실제 다음 문장인 지 여부를 예측 (NSP)
- NSP를 통해 문장 간 관계를 학습, 문맥적 추론 및 문장 수준 이해 태스크에 도움이 되도록 설계

ex) 자연어 추론, 문맥 예측, 질의응답

BERT의 다운스트림 태스크

- BERT는 이렇게 MLM과 NSP 두 가지 태스크를 동시에 학습
- [CLS] 토큰은 NSP 태스크용으로 학습되며, 다른 토큰들은 MLM 태스크용으로 학습

1. 문장 레벨
    - 두 문장 관계 분류 태스크
        - MNLI
        - QQP
    - 단일 문장 분류 태스크
        - SST2
2. 토큰 레벨
    - QA 태스크
        - SQuAD
    - 개체명 인식
        - CoNLL 2003 NER

BERT의 결과

- 다양한 태스크에 적용 가능한 범용성을 보여줌
- Fine-tuning을 통해 여러 NLP 과제에서 새로운 최첨단(SOTA) 성능을 이끌어냄
- Layer 수, hidden state의 크기, attention head의 수가 클 수록 성능이 향상

BERT의 한계

- 인코더 기반 모델인 BERT는 주어진 입력을 잘 이해하지만, 시퀀스를 생성하는 태스크에는 적합하지 않음
    - ex) 기계번역, 텍스트생성
- 생성 태스크에서는 autoregressive 하게, 해야 하지만, 자연스럽게 수행 못함
    - 생성 태스크에는 디코더 기반 모델 사용

3. Encoder-Decoder 모델

T5 (Text-to-Text Transfer Transformer, 2019 Google Research)

- Transformer Encoder-Decoder 구조 기반 모델
- 모든 태스크를 Text-to-Text 포맷으로 변환해 하나의 모델로 학습

T5의 학습 방법

Span Corruption : 인코더가 입력 문장을 모두 보고, 그 정보를 바탕으로 디코더가 출력 생성

- 과정
    - 입력 문장에서 연속된 토큰을 무작위로 선택해 제거
    - 제거되 부분을 특수 placeholder 토큰으로 치환 ex) <X>, <Y>
    - 디코더는 이 placeholder 에 해당하는 원래 span을 복원하도록 학습

    → 언어 모델링처럼 훈련 수행 가능

T5의 다운스트림 태스크

- T5는 NLU (GLUT, SuperGLUE), QA(SQuAD), 요약(CNNDM), 번역(En→De, En→Fr,En→Ro) 등의 태스크에서 모두 좋은 성능을 보임, 범용적으로 활용될 수 있는 모델

4. Decoder 모델

Fine-tuning Decoder

- Transformer의 Decoder 는 사전학습 단계에서 다음 단어 예측 (Next Token Prediction)을 학습한다.
- 생성 태스크에 활용할 때
  - 사전학습때와 동일하게 다음 단어 예측 방식으로 fine-tuning 한다.
  - 디코더는 대화나 요약 태스크 등 출력이 시퀀스인 태스크에 자연스럽게 적합
- 분류 태스크에 활용할 때
  - 마지막 hidden state 위에 새로운 linear layer를 연결해 clissifier 로 사용
  - linear layer는 아예 처음부터 다시 학습해야 하며, fine-tuning 시 기울기가 decoder 전체로 전파된다.

GPT-1 (2018, OpenAI)

- Transformer 기반 디코더 모델
- Autoregressive LM(왼쪽→오른쪽 단어예측) 방식으로 사전 학습 됨

GPT-1의 fine-tuning 방법 (분류 태스크)

- GPT-1은 다음 단어 예측이라는 언어 모델 학습 목표를 최대한 유지하면서 fine-tuning을 수행
- NLI(두문장을 입력 받아 관계 함의(entaliment)/모순(contradiction)/중립(neutral) 중 하나로 분류)
- 입력 토큰에 특수한 토큰([START],[DELIM],[EXTRACT])을 붙여 분류 문제 처리, [EXTRACT]위치의 hidden state에 classifier 를 연결해 사용

GPT-1의 결과

- 생성 태스크를 잘 했고, 태스크별 fine-tuning을 통해 분류/추론 등 이해 중심 태스크에서도 우수

GPT-2(2019, OpenAI)

- GPT-1 의 확장 버전
- 더 많은 데이터, 더 큰 parameter size로 학습
- 더 자연스러운 텍스트 생성 능력

5. In-Context Learning

GPT-3(2020, OpenAI)

- GPT-2에서 모델의 parameter size를 키워, 별도의 파인튜닝 없이 컨텍스트 안의 예시만 보고도 새로운 태스크를 수행 할 수 있게 됨

  → In-Context Learning

GPT-3 의 In-Context Learning

- 모델에 예시와 함께 어떤 태스크를 할 지 지정해주면, 모델이 그 패턴을 따라가는 식으로 동작
  - 예시 개수에 따라 분류

    Zero-shot : 예시 0개

    One-shot : 예시 1개

    Few-shot : 예시 몇 개

- 모델의 크기(Parameter size)가 커질수록 더 강력해진다.
- Shot의 수가 많을 수록 태스크 수행 능력이 향상

Chain-of-Thought Prompting

- In-Context Learning의 발견으로 Prompt의 중요성이 대두
- Few-shot Prompting 만으론 복잡한 문제 해결이 어려움
    - Chain-of-Thought(CoT) prompting방식의 등장
- CoT prompting 은 모델이 문제 해결 과정에서 논리적인 사고 단계를 거쳐 최종 답을 도출하도록 유도하는 prompting 기법이다.
- CoT Prompting 은 일반 프로프팅 방식보다 훨씬 우수한 성능을 보이며 새로운 SOTA 성과 달성
    - 심지어 fine-tuning 한 모델보다 성능이 우수

Zero-shot Chain-of-Thought prompting

- 기존 CoT 프롬프팅은 few-shot 예시가 필요하며, 예시가 없으면 추론 과정이 나타나지 않아 성능 저하가 발생

→ 기적의 문장 “Step by Step”의 등장, 모델이 스스로 추론 단계를 생성하도록 유도

→ Zero-shot CoT 프롬프팅