1015 | 2-1. 자연어 처리 기본

1. 워드 임베딩과 순환신경망 기반 모델(RNN & LSTM)

1. 단어를 숫자로 표현

- 워드 임베딩(word embedding)
  - one-hot encoding
    - a. 규칙 기반 혹은 통계적 자연어 처리 연구의 대다수는 단어를 원자적(쪼갤 수 없는) 기호로 취급한다.
    - b. 벡터공간관점에서 보면, 이는 한 원소만 1이고 나머지는 모두 0인 벡터를 의미
    - c. 차원 수(= 단어 사전 크기)는 대략적으로 음성 데이터 2만개, Penn Treebank(PTB) 코퍼스(5만) - big vocab(50만) - Google 1T(1300만)
    - d. 이를 one-hot(원핫) 표현이라고 부르고 단어를 one-hot 표현으로 바꾸는 과정을 one-hot encoding이라고 한다.
  - one-hot encoding의 단점 및 문제점
    - a. 검색 query vector와 대상이 되는 문서 vector들이 서로 직교하게 되어, one-hot vector로는 유사도를 측정할 수 없다.
    - b. 차원의 저주(Curse of Dimenstionality)
      - (a) 고차원의 희소 벡터를 다루기 위해선 많은 메모리가 필요하다.
      - (b) 차원이 커질 수록 데이터가 점점 더 희소(sparse)해져 활용이 어렵다.
    - c. 의미적 정보 부족
      - (a) 비슷한 단어라도 유사한 벡터로 표현되지 않는다.
      - (b) 은행과 금융은 의미적으로 밀접하지만, one-hot encoding에서는 전혀 무관한 vector로 취급된다.(의미적으로 밀접해도 one-hot encoding에서는 각각을 무관하게 취급하거나 의미적으로 전혀 관계가 없어도 유관하게 취급할 수가 있다.)
  - word embedding이란 단어를 단어들 사이의 의미적 관계를 포착할 수 있는 밀집(dense)되고 연속적/분산적(distributed) vector 표현으로 나타내는 방법
    - a. one-hot encoding에선 은행과 금융이 완전히 독립적인(무관한) vector로 표현되었지만,
    - b. word embedding에선 두 단어의 vector가 공간상 서로 가깝게 위치하며, 이를 통해 의미적 유사성을 반영할 수 있다.
    - 대표적인 기법
      - (a) Word2Vec
        - 단어의 표현을 간단한 인공 신경망을 이용해서 학습
        - 주요 아이디어는 각 단어와 그 주변 단어들 간의 관계를 예측

        - 주요 알고리즘
            - Skip-grams(SG) 방식
                - 중심 단어를 통해 주변 단어들을 예측하는 방법
                - 단어의 위치(앞 또는 뒤)에 크게 구애 받지 않는다.
                - 윈도우 크기(window size) : 중심 단어 주변 몇개 단어를 문맥으로 볼 것인가?
                - 장점 : 적은 데이터에도 잘 동작하며, 희귀 단어나 구 표현에 강하다.
                - 단점 : 학습 속도가 느리다.
            - Continuous Bag of Words(CBOW) 방식
                - 주변 단어들을 통해 중심 단어를 예측하는 방법
                - 문맥 단어들의 집합으로 중심 단어를 맞춘다.
                - 주변 단어와 함께 묶어서 input data로 활용해서 예측
                - 주변 단어들의 집합이 주어졌을 때, 그 문맥과 함께 등장할 수 있는 단일 단어를 예측
                - 장점 : 학습 속도가 빠르며, 자주 나오는 단어에 강하다.
                - 단점 : 희귀 단어 표현에 약하다.

2. 언어는 순서가 중요함

- 순차적 데이터의 특징
  - 자연에 수많은 순차적(Sequential Data)가 존재
  - 데이터가 입력되는 순서와 이 순서를 통해 입력되는 데이터들 사이의 관계가 중요한 데이터
  - 대표적으로 오디어, 텍스트, 비디오 데이터 등이 있음.
  - 1) 순서가 중요하다. 데이터의 순서가 바뀌면 의미가 달라진다.
  - 2) 장기의존성(Long-term dependency) : 멀리 떨어진 과거의 정보가 현재/미래에 영향을 준다.
  - 3) 가변길이(Variable length) : 순차 데이터는 길이가 일정하지 않고, 단어 수도 제각각이다.
    - 처리 방법
        - 일반적인 모델인 선형회귀, MLP 등으로는 처리가 불가능하다.
        - 따라서, Sequential Models이 필요하다.(RNN, LSTM, Transformer 등)

3. RNN

- 문맥을 기억하는 신경망
  - 전통적인 인공신경망은 MLP, CNN 등이며, 이는 고정된 길이의 입력을 받기 때문에, 가변 길이의 데이터를 처리하기에 적합하지 않는다.
  - RNN은 가변 길이의 입력을 입력 받을 수 있고, 이전 입력을 기억할 수 있기 때문에, 순차적 데이터 처리에 적합한 architecture이다.
  - 대표적으로, one to one, many to one, one to many, many to many 등이 있다.

- Architecure 설명
  - 전통적인 인공신경망은 MLP, CNN과 달리, RNN은 이전 시점의 정보를 담는 hidden state를 가지고 있다.
  - 따라서, 입력 시퀀스 벡터 x를 처리할 때, 각 시점마다 recurrence 수식을 적용하며 hidden state를 업데이트한다.

$$h_t = f_W(h_{t-1}, x_t)$$

$$h_t = f_W(h_{t-1}, x_t)$$

$$h_t : 새로운 hiddenstate$$

$$f_W : 처리함수(파라미터 W)$$

$$h_{t-1} : 이전 hiddenstae$$

$$x_t : 입력 vector(Timestep)$$

- RNN은 각 시점마다 동일한 가중치(weight)를 사용하기 때문에, 깊은 feedward 신경망보

$$y_t = W_{hy}h_t$$

$$h_t = tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

- RNN의 특징
  - RNN은 한 번에 하나의 요소를 처리하고, 정보를 앞으로 전달한다.
  - RNN은 각 층이 하나의 시점을 나타내는 깊은 신경망처럼 보인다.
  - RNN은 hidden state를 유지하면서 가변 길이 데이터를 처리할 수 있다.
  - RNN의 출력은 과거 입력에 영향을 받는다는 점에서, feedward 신경망과 다르다.
- RNN의 단점
  - 기울기 소실(vanishing gradient) 문제
  - vanishing gradient 문제란?
    - 딥러닝에서 역전파 시 앞쪽 측의 기울기가 0에 가까워져서 장기 의존성 학습이 어려

    - 역전파 과정에서 작은 값들이 계속 곱해진다.
    - 과거 시점에서 온 오차 신호는 갈수록 더 작은 기울기를 갖게 된다.
    - 결국 파라미터들이 장기 의존성은 학습하지 못하고, 단기 의존성만 포착하게 된다.

4. LSTM

- LSTMs란 기울기 소실문제를 해결하기 위해, 1997년에 제안된 RNN의 한 종류
  - LSTMs의 특징

        - 시점 t에서 RNN은 길이가 n인 vector hidden state h_t와 Cell state C_t를 가진다.
        - Hidden state는 short-term information을 저장한다.
        - Cell state는 long-term information을 저장한다.
    - LSTMs cell state에서 정보를 읽고(read), 지우고(erase), 기록(write)할 수 있다.
    - LSTMs는 3가지 게이트를 통해 어떤 정보를 지우고 쓰고 읽을지를 결정한다.
        - Forget gate
            - 이전 cell state에서 무엇을 버리고 무엇을 유지할지 결정한다.

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

- Input gate
    - 새 정보 중 얼마나 cell state에 쓸지 결정한다.

$$i_t = sigma(W_i cdot[h_t, x_t] + b_i)$$

$$tildeC_t = tanh(W_C cdot[h_{t-1}, x_t] + b_C)$$

- New Cell Content
    - New Cell Content : Cell에 기록될 새로운 후보

$$C_t = f_t * C_{t-1} + i + t * tildeC_t$$

- Output gate
    - cell state 중 얼마나 hidden state로 내보낼지 결정한다.

$$o_t = sigma(W_o[h_{t-1}, x_t] + b_o)$$

$$h_t = o_t * tanh(C_t)$$

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

2. 자연어 생성 모델 (Seq2Seq, Attention)

1. 언어 모델이란?

1-1. 개념

- 인간의 두뇌가 자연어를 생성하는 능력을 모방한 모델
- 단어 시퀀즈 전체에 확률을 부여하여 문장의 자연스러움을 측정
- 한 문장의 확률은 각 단어의 조건부 확률들의 곱으로 표현할 수 있음

→ 언어 모델은 **문장 내 단어의 등장 확률 분초를 모델링하는 것**을 의미

→ 즉, 주어진 단어들의 연속이 나올 확률을 계산하여 **다음 단어를 예측**하거나

→ 문장의 자연스러움(유창성**)을 평가**하는 모델

1-2. 일상 속의 언어모델 ex. 텍스트 자동완성 기능

1-3. 대표적인 언어모델 : N-gram 언어모델

- n-gram : 연속된 n개의 단어 묶음
  - unigram: `"The"` , `"students"` , `"opened"` , `"their"`
  - bigram: `"The students"` , `"students opened"` , `"opened their"`
  - trigram: `"The students opened"` , `"students opened their"`
  - 4-gram: `"The students opened their"`
- 언어모델의 목표는 다음 단어의 조건부 확률을 추정하는 것
- 다른 예시 : 4-gram 언어 모델
- 예시 데이터:
  - `"students opened their"` : 1000회 등장
  - `"students opened their books"` : 400회 등장
    → P(books ㅣ students opened their)=0.4P(\text{books} | \text{students opened their}) = 0.4P(books ㅣ students opened their)=0.4
  - `"students opened their exams"` : 100회 등장
    → P(exams ㅣ students opened their)=0.1P(\text{exams} | \text{students opened their}) = 0.1P(exams ㅣ students opened their)=0.1

1-4. 언어모델 사용 예시 : Statistical Machine Translation, SMT

- 1990년대~2010년대 초반까지는 번역 문제를 **확률적 모델링**으로 접근
- 예시 : 한국어 → 영어
  - 목표: 주어진 한국어 문장 x 에 대해, 가장 적절한 영어 문장 y 를 찾는 것

$$argmax_y P(y \mid x)$$

- **Bayes 정리를 이용한 분해**P(y ∣ x)=P(x)P(x ∣ y)P(y)⇒argymaxP(x ∣ y)P(y)

$$= argmax_y P(x \mid y)P(y)$$

- 두 개의 구성요소로 분리:
  - **번역모델 P(x ∣ y)**
    - 단어와 구가 어떻게 번역되어야 하는지 모델링
    - (한국어, 영어) 말뭉치로 학습
      → 영어 문장 yyy가 주어졌을 때 한국어 문장 xxx가 생성될 확률
  - **언어모델 P(y)**
    - 유창한 영어 문장을 쓰는 방법을 모델링
    - 영어 데이터로 학습
- 한계점 : SMT의 문제
  - **구조적 복잡성**: 문장 구조를 반영하기 어렵다.
  - **수작업 의존도**: 문장 정렬, 단어 정렬, 구문 규칙 등의 수작업 필요.
  - **언어별 자원 구축 부담**: 각 언어쌍마다 별도의 말뭉치와 규칙이 필요.
    → 결과적으로 유지·확장성이 떨어짐
- 발전 방향 : **Neural Machine Translation (NMT)**
  - 위 한계를 극복하기 위해 **신경망 기반 번역모델(NMT)**로 전환됨.
  - NMT는 **하나의 신경망이 전체 번역 과정을 자동으로 학습**하여 번역모델과 언어모델을 통합적으로 처리함

2. Seq2Seq

2-1. Neural Machine Translation (NMT)

- 개념
  - Neural Machine Translation 이란 인공 신경망을 이용해 기계 번역을 수행하는 방법
  - 이 때 사용되는 신경망 구조를 sequence-to-sequence (Seq2Seq)이라 하며, 두 개의 RNNs으로 이루어짐
    - 2014년 Google의 "Sequence to Sequence Learning with Neural Networks"라는 논문에서 처음 소개
- Translation이 어려운 이유
  - 번역 문제는 입력과 출력의 길이가 다를 수 있음
    - 영어: the black cat drank milk (5개의 단어)
    - 프랑스어: le chat noir a bu du lait (7개의 단어)

  → 따라서 NMT에서는 길이가 다른 시퀀스 간의 매핑을 처리할 수 있어야함

- Seq2Seq의 아이디어
  - 2개의 LSTM을 이용하자
    - 한 LSTM은 입력 시퀀스를 한 타임스텝씩 읽어 고정된 차원의 큰 벡터 표현을 얻기 (Encoder)
    - 다른 LSTM은 앞에서 얻은 벡터로부터 출력 시퀀스를 생성하기 (Decoder)

2-2. Seq2Seq Architecture

- Seq2Seq의 구성
  - Encoder와 Decoder로 이루어진다.
    - Encoder는 입력 문장에 담긴 정보를 인코딩
    - Decoder는 인코딩된 정보를 조건으로 하여 타겟 문장(출력)을 생성
- Seq2Seq의 다양한 적용
  - Seq2Seq 구조는 기계번역 외에도 다양한 태스크에 적용 가능
    - 요약 : 긴 길이의 문서를 읽고, 짧은 길이의 문장으로 요약된 텍스트를 출력하는 태스크
    - 대화: 사용자의 발화를 기반으로, 맥락에 맞는 대답(출력 텍스트)을 생성하는 태스크
    - 코드 생성: 자연어로 작성된 설명 혹은 명령어를 입력 받아, 그에 대응하는 프로그래밍 코드 혹은 쿼리를 출력하는 태스크
- Seq2Seq의 학습 수행 (Teacher Forcing)
  - Seq2Seq 모델은 인코더와 디코더가 하나의 통합 네트워크로 연결되어 있음
  - 디코더에서 발생한 오차는 역전파 과정을 통해 입력을 처리한 인코더까지 전달되어 전체 네트워크가 End-to-End로 동시에 최적화된다.
  - 학습 초반에는 모델의 예측 능력이 떨어지기 때문에 학습이 불안정할 수 있음
  - Teacher Forcing이란?
    - 모델이 스스로 예측한 단어 대신 정답 단어를 디코더 입력으로 강제로 넣어줌으로써 훨씬 안정적이고 빠르게 학습을 수행하는 방법이다.
- Seq2Seq의 토큰 출력 방법 (Greedy Inference)
  - 토큰을 출력하는 방법 중 하나로, 각 단계에서 가장 확률이 높은 단어를 선택
  - 한계
    - 되돌리기가 불가능
    - 예시 : le _____ → le chien ____ → …
      chein이 오답이어도 돌아갈 방법이 없지
  - Beam Search
    a. 매 단계마다 k개의 가장 유망한 후보 유지
    b. 후보가 에 도달하면, 완성된 문장으로 리스트 추가
    c. 문장이 충분히 모이면 탐색 종료
    d. 각 후보들의 점수를 로그 확률의 합으로 구해 최종 선택

3. Attention

3-1. Seq2Seq의 한계 : the bottleneck problem

- Bottleneck problem 개념
  - 인코더는 입력 문장 전체를 하나의 벡터로 요약하는데, 마지막 hidden state에 문장의 모든 의미 정보가 담긴다
  - 고정 길이 벡터 하나에 모든 문장의 의미를 압축하다 보니 정보 손실이 생길 수 있는데, 이를 bottleneck problem이라고 한다.
- Attention의 인사이트
  - Attention은 디코더가 단어를 생성할 때, 인코더 전체 hidden state 중 필요한 부분을 직접 참조할 수 있도록 한다.
  - 즉, 매 타임스텝마다 "어떤 단어/구절에 집중할지"를 가중치로 계산해, bottleneck 문제를 완화했다.

3-2. Attention의 효과

- Attention mechanism은 많은 장점이 존재한다.
  i. NMT 성능 향상
    - 디코더가 소스 문장 전체가 아닌, 필요한 부분에만 집중할 수 있기 때문이다.
  ii. Bottleneck Problem 해결
    - 디코더가 인코더의 모든 hidden states에 직접 접근할 수 있다.
  iii. Vanishing Gradient Problem 완화
    - Attention은 멀리 떨어진 단어도 직접 연결할 수 있게 해준다.
- Attention의 효과: 해석 가능성(Interpretability)
  - Attention 분포를 보면, decoder가 어떤 단어를 생성할 때, 입력 문장의 어느 부분에 집중했는지 확인할 수 있다.
  - 즉, 모델이 내부적으로 참고한 근거를 사람이 파악할 수 있음
    → 모델의 의사결정 과정을 해석할 수 있는 단서
- Attention의 효과: 정렬(Alignment)
  - 기계번역에서는 전통적으로 단어 alignment 모델을 따로 학습해야 했다.
  - 하지만, attention은 통해 decoder가 필요한 입력 단어에 자동으로 집중하기 때문에, 단어와 단어 간의 매핑 관계를 자연스럽게 학습한다.
- Attention : Query와 Values
  - Seq2Seq에서 attention을 사용할 때, 각 decoder의 hidden state와 모든 encoder의 hidden states 간의 관계를 Query와 Values의 관계로 볼 수 있다.
  - 이 관점에서 Attention 과정을 정리해보면:
    a. Query와 Values 사이 유사도 점수(score) 계산 (예: dot-product, multiplication, additive 등)
    b. Softmax를 통해 확률 분포(attention distribution) 얻기
    c. 분포를 이용해 values를 가중합 → context vector (attention output)

3. Transformer

1. Self-Attention

1-1. RNN

- RNN의 필요성
  - RNN은 정보를 hidden state로 전달하며 순차적 의존성을 형성한다.
  - 그에 반해 Attention은 필요한 순간, 입력 전체에서 직접 정보를 전달한다.
    → RNN이 하던 "정보 전달"을 Attention이 더 효율적으로 수행할 수 있다면, 굳이 recurrence가 필요할까?

1-2. RNN의 한계점

1. 장기 의존성
   - RNN은 왼쪽에서 오른쪽으로 순차 전개되어, 먼 단어 쌍이 상호작용하려면 시퀀스 길이 만큼의 단계를 거쳐야 한다.
   - 따라서, 길어진 단계만큼 기울기 소실 혹은 폭발 문제가 발생해 장기 의존성을 학습하기 어렵다.
   - 또한, RNN은 입력된 선형 순서를 강하게 반영하기 때문에, 언어의 비선형적 의존성을 잘 잡아내지 못한다.
     - 예시: 그 책은 오랫동안 방치되어 먼지가 많이 쌓인 탓에… 매우 더러웠다.
2. 병렬화
   - Forward와 Backward pass 모두 시퀀스 길이 만큼의 단계가 필요하다.
   - 이처럼, 순차적인 연산이 진행되기 때문에, 병렬화가 불가능하다.
   - 이는, 병렬 연산에 강한 GPU 활용을 어렵게 만들며, 대규모 데이터 학습에 비효율적이다.

1-3. Attention은 어떨까 ?

- Attention은 각 단어의 표현을 query로 두고, value 집합으로부터 필요한 정보를 직접 불러와 결합한다.
- 이러한 Attention 메커니즘을 encoder-decoder 간이 아닌, 한 문장 내부에서 적용한다면 어떨까?
  → Self-Attention

1-4. Self-Attention의 장점은?

1. 순차적으로 처리해야 하는 연산 수가 시퀀스 길이에 따라 증가하지 않는다.
2. 최대 상호작용 거리 =O(1) → 모든 단어가 각 층에서 직접 상호작용한다.

1-5. Seq2Seq에서의 Attention

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

- 배경
  - 기존 Seq2Seq 모델은 입력 문장을 **하나의 고정된 벡터(Context Vector)** 로 압축한 뒤, 이를 디코더가 이용해 출력 문장을 생성
  - 하지만 문장이 길어질수록 이 벡터 하나에 모든 정보를 담기 어려워졌고, **Attention**이 도입되면서 이 문제를 해결하게
- 핵심 아이디어
  - 출력 문장(Decoder)의 각 단어를 생성할 때, 입력 문장(Encoder)의 **모든 단어들 중에서 "관련 있는 단어"에 더 집중(attend)** 하도록 하는 메커니즘.
  - **입력 문장(Encoder의 hidden states)** → Key, Value
  - **출력 문장(Decoder의 현재 hidden state)** → Query
- 예시
  - 입력 문장: `"I love cats"`
  - 출력 문장: `"나는 고양이를 좋아해"`
    - Decoder가 `"고양이를"` 을 생성할 때
      → Encoder의 단어들 중 `"cats"` 에 높은 가중치를 부여
      → `"I"` , `"love"` 에는 낮은 가중치
- 정리
  - Seq2Seq Attention은 **"출력 단어를 생성할 때 입력 문장의 어떤 단어에 집중할지"**를 결정하는 메커니즘이다.

1-6. Self-Attention

- Attention(문장 내의 다른 단어들 | 문장 내의 각 단어 "w")
- 배경
  - Self-Attention은 Seq2Seq처럼 "입력 → 출력" 관계가 아니라, **하나의 문장 내부 단어들끼리 서로 어떤 연관이 있는지**를 학습하는 메커니즘
- 핵심 아이디어
  - 하나의 단어를 표현할 때, 그 단어가 같은 문장 안의 다른 단어들과 어떤 관계를 맺는지를 반영
- 예시
  - 문장: `"The animal didn't cross the street because it was too tired."`
  - 단어 `"it"` 이 나올 때,
    → Self-Attention은 `"animal"` 과 `"it"` 이 같은 대상을 가리킨다는 관계를 학습한다.
    즉, `"it"` 을 표현할 때 `"animal"` 의 정보가 중요하게 반영된다.
- 정리
  - Self-Attention은 **"문장 내에서 각 단어가 다른 단어들과의 관계를 고려하여 표현을 만드는 메커니즘"**이다.
  - 모든 단어가 서로를 참조하면서 문맥적 의미를 포착한다.

1-7. Self-Attention - Key, Query, Value

- Attention에서 각 단어 "i"를 표현하는 Query와 Value 벡터가 있었듯이, Self-Attention에서는 각 단어 "i"를 표현하는 Query, Key, Value 벡터가 존재한다.
  i. 각 단어를 Query, Key, Value 벡터로 변환한다.
    - Query 벡터
      - 단어 i가 다른 단어로부터 어떤 정보를 찾을지를 정의하는 벡터
    - Key 벡터
      - 단어 i가 자신이 가진 정보의 특성을 표현하는 벡터
    - Value 벡터
      - 실제로 참조되는 정보 내용을 담고 있는 벡터
  ii. Query, Keys 간의 유사도를 계산해, softmax로 확률분포를 구한다.
    - 유사도 계산식

$$e_{ij} = q_i^T k_j$$

    - Softmax로 확률화

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{j'} \exp(e_{ij'})}$$

  iii. 각 단어의 출력을 Values의 가중합으로 계산

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

1-8. Self-Attention의 한계

- 이렇듯, Self-Attention은 단어 간 관계를 효율적으로 잡아내는 강력한 메커니즘이지만, 한계가 존재한다.
- 한계
  i. 순서 정보 부재 → 단어 간 유사도만 계산하기 때문에, 단어의 순서를 고려하지 않는다.
  ii. 비선형성 부족 → Attention 계산은 본질적으로 가중 평균 연산이라는 선형 결합에 불과하기 때문에, 복잡한 패턴이나 깊은 표현력을 담기 어렵다.
  iii. 미래 참조 문제 → 언어 모델은 시퀀스를 왼쪽에서 오른쪽으로 생성해야 하지만, Self-Attention은 모든 단어를 동시에 보기 때문에, 아직 생성되지 않아야 할 미래 단어를 참조한다.

1-9. Self-Attention의 한계 해결

1. Positional Encoding
   - 순서 정보 부재 문제를 해결하기 위해, Positional Encoding이라는 기법을 사용한다.
   - 각 단어 위치 i를 나타내는 위치 벡터를 정의해, 단어 임베딩 값에 더해 최종 입력으로 사용한다.
   - 두 가지 방법이 존재
     - Sinusoidal Position Encoding

        - 서로 다른 주기의 사인/코사인 함수를 합성해 위치 벡터를 만드는 방법
    - Learned Absolute Position Embedding
        - 위치 벡터를 모두 학습 파라미터로 설정해 학습 과정에서 데이터에 맞춰 최적화하는 방법
2. Feed-Forward Network 추가하기
    - Self-Attention 연산은 비선형 변환이 없어, 복잡한 패턴 학습에 한계가 존재한다.
    - 따라서, 각 단어 출력 벡터에 Feed-Forward Network (Fully Connected + ReLU 등)을 추가해 Self-Attention이 만든 표현을 깊고 비선형적인 표현으로 확장한다.
3. Masked Self-Attention
    - 단어를 생성할 때는 한 단어씩 순차적으로 미래 단어를 예측해야 하지만, Self-Attention은 기본적으로 모든 단어(미래 포함)를 동시에 참조한다.
    - 따라서, Attention Score를 계산할 때, 미래 단어에 해당하는 항목을 -∞로 설정해, 계산을 수행할 때 반영되지 않도록 한다.

1-10. Self - Attention 정리

- Self-Attention은 문장 내 모든 단어가 서로 직접 상호작용하여,
    i. 장거리 의존성을 효율적으로 포착하고
    ii. 병렬 처리를 가능하게 하는 메커니즘이다.
- 한계 해결 방법
    i. 순서 정보 부재 → Positional Encoding
    ii. 비선형성 부족 → Feed-Forward Network 추가
    iii. 미래 참조 문제 → Masked Self-Attention

epeat for number
f encoder blocks

2. Transformer

2-1. Transformer

- Transformer는 encoder-decoder 구조로 설계된 신경망 모델이다.
    - encoder: 입력 문장을 받아 의미적 표현으로 변환을 수행한다.
    - decoder: 인코더의 표현과 지금까지 생성한 단어들을 입력 받아, 다음 단어를 예측한다.
      → 이 중 decoder가 언어 모델과 같은 방식으로 동작한다.

2-2. Multi-Headed Attention

- 문장에서 같은 단어라도 여러 이유(문법적 관계, 의미적 맥락 등)로 다른 단어에 주목할 수 있다.
- 하지만, 단일 Self-Attention Head로는 한 가지 관점에서의 단어 간 관계 밖에 파악할 수 없다.
  → 따라서, 여러 Attention Head를 두어 다양한 관점에서 동시에 정보를 파악한다.

- Attention Head 1 - 단어의 문맥적 관계에 Attention → 문법적 의존성 관계 단어들에 주목
- Attention Head 2 - 단어의 시제에 Attention → 개체에 주목
- Attention Head 3 - 명사에 Attention

2-3. Scaled Dot Product

- Query와 Key의 차원이 커질수록, 두 벡터의 내적 값도 자연스럽게 커진다.
- 이 값이 너무 크면 softmax 함수가 출력이 지나치게 뾰족해져, 미세한 변화에도 큰 차이가 발생하고 gradient vanishing 문제가 생길 수 있다.

$$output_l = softmax(XQ_lK_l^TX^T) * XV_l$$

- 따라서, 내적 값을 그대로 사용하지 않고, 나눠주어 스케일을 조정한다.
- 이렇게 하면 감이 안정적으로 분포되어 학습이 훨씬 더 빠르고 안정적으로 진행되다.

2-3. Residual Connection

- 깊은 신경망은 층이 깊어질수록 학습이 어려워진다. (Gradient vanishing / exploding)
- 따라서, 단순히 Layer의 출력만 사용한다면 정보가 소실되어, layer가 전체를 예측하는 것이 아니라, 기존 입력과의 차이만 학습하도록 하는 residual connection을 사용한다.

2-4. Layer Normalization

- 층이 깊어질수록, hidden vector 값들이 커졌다 작아졌다 하면서 학습이 불안정하다.
- Layer Normalization은 각 레이어 단위에서 hidden vector 값을 정규화해, 안정적이고 빠른 학습을 돕는다.

2-5. Decoder

- Transformer의 decoder는 여러 개의 decoder 블록들을 쌓아 올려서 만든 구조이다.
- 각 블록은 다음으로 구성된다:
  - Masked Self-Attention (Multi-Head)
    - 미래 단어를 보지 않도록 마스크를 씌운 Multi-Head Self-Attention.
  - Add & Norm (Residual Connection + Layer Normalization)
  - Feed-Forward Network
    - 각 위치 별 비선형 변환을 수행한다.
  - Add & Norm (Residual Connection + Layer Normalization)

→ 언어 모델처럼 단방향 문맥만 활용

2-6. Encoder

- 그에 반해 Transformer의 encoder는 양방향 문맥을 모두 활용할 수 있다.

    - 입력 문장을 의미적 표현으로 변환할 수 있다.
    - 각 단어가양방향 문맥을 모두 반영한 벡터로 인코딩된다.
- Decoder와의 차이점은 Self-Attention에서 masking을 제거한 것 뿐이다.

2-7. Encoder-Decoder

- 기계 번역에서 Seq2Seq 모델을 사용했던 것처럼, Transformer에서도 이해를 위한 encoder와 생성을 위한 decoder로 이뤄진 encoder-decoder 구조를 채택한다.
- decoder는 단순 Self-Attention만 하는 것이 아니라, encoder의 출력 표현을 참조하는 Cross-Attention을 추가하여 입/출력을 연결한다.
- Cross-Attention
    - Cross-Attention을 수행할 때는 Self-Attention과는 다르게
    - Query는 decoder에서, Key와 Value는 encoder에서 가져온다.

2-8. Transformer

- 결과
    - Neural Machine Translation task에서 당시 최고 성능을 달성했을 뿐 아니라, 가장 효율적인 학습으로 비용까지 절감할 수 있었다.
- Transformer와 사전학습(pretraining)
    - Transformer의 등장은 대부분의 최신 모델들이 성능 향상을 위해 사전학습(pretraining)을 결합하도록 했다.
    - 또한, 뛰어난 병렬 처리 능력 덕분에 대규모 사전학습에 적합하여 NLP의 표준 아키텍처로 자리 잡았다.

4. 사전 학습 기반 언어 모델

1. 사전학습이란

- 정의
    - 대규모 데이터 셋을 이용해, 모델이 데이터의 일반적인 특징과 표현을 학습하도록 하는 과정
    - 특히, 언어모델은 인터넷의 방대한 텍스트(웹문서, 책, 뉴스 등)을 활용해 비지도학습 방식으로 학습되어, 일반적인 언어 패턴, 지식, 문맥 이해 능력을 습득한다.
- 사전학습의 관점에서 word embedding vs language model
    - word embedding의 경우 사전학습을 통해 단어의 의미를 학습하지만 한계가 존재
        - downstream task(ex. 텍스트 분류)들에 적용하기엔 학습되는 데이터의 양이 적어, 언어의 풍부한 문맥 정보를 충분히 학습할 수 없다.
        - 연결된 네트워크가 무작위 초기화되어 학습 효율이 낮고, 많인 데이터와 시간이 필요하다.

- Language Model의 경우 모델 전체를 사전학습을 통해 학습하기 때문에, 강력한 NLP 모델을 구축하는 데에 이점이 있다.
  - 언어에 대한 표현학습용으로 적합
  - parameter 초기화의 관점에서 효과적
  - 언어에 대한 확률 분포 학습을 통해 sampling, 생성에 사용할 수 있다.
- Language Model의 pre-train
  - 과거 단어들이 주어졌을 때, 다음 단어의 확률 분포를 모델링하는 방법을 배움으로서, 사전학습을 수행할 수 있다.
  - 인터넷의 대규모 텍스트 코퍼스에서 언어모델링 학습을 수행 후 학습된 network parameter를 저장해, 다양한 downstream task에 활용한다.
- 사전학습에서 fine-tuning으로의 패러다임 변화
  - 사전학습을 통해, 언어 패턴을 잘 학습한 parameter롤 초기화해 NLP application 성능을 향상시킬 수 있다.

2. Encoder 모델

- Encoders
  - 양방향 문맥을 활용하기 때문에, 전통적인 언어 모델과는 차이점이 있다.
  - 따라서, Encoder 모델의 사전 학습을 위해선 입력 단어의 일부를 [MASK] 토큰으로 치환해, 모델이 이 [MASK] 자리에 올 단어를 예측하도록 학습하는 방법
  - 이를 Masked Language Model(MLM)이라 하며, 대표적인 모델이 BERT이다.
    - transformer 기반의 모델로 Masked LM 방법으로 사전학습을 수행
    - 학습방식 1
      - Masked Language Model(MLM)
      - 입력 토큰의 15%를 무작위로 선택
      - [MASK] 토큰 치환(80%), 랜덤 토큰 치환(10%), 그대로 두기 (10%)
      - 모델이 masked된 단어에만 집중하지 않고 다양한 문맥표현을 학습해 더 강건한(robust) 표현을 학습할 수 있도록 했다.
    - 학습방식 2
      - Next Sentence Prediction(NSP)
      - BERT는 입력을 두 개의 연속된 텍스트로 받아, 두번째 문장이 첫번째 문장의 실제 다음 문장인지 여부를 예측하는 Next Sentence Prediction(NSP)을 수행
      - NSP를 통해 문장 간 관계를 학습하여 문맥적 추론 및 문장 수준 이해 task에 도움이 되도록 설계
      - ex.
        - 자연어 추론(Nature Language Inference)
        - Paraphrase detection
        - 질의응답(Quetions Answering, QA)

        - BERT는 이렇게 MLM과 NSP 두 가지 task를 동시에 학습한다.
        - [CLS] 토큰은 NSP task용으로 학습되며, 다른 토큰들은 MLM task용으로 학습된다.
    - BERT의 downstream task
        - Sentence Level
            - 두 문장 관계 분류 task
                - MNLI
                    - 전제(Premise)
                    - 가설(Hypothesis)
                    - 분류 : {Entailment, Contradiction, Neutral}
                - QQP
                    - Q1, Q2, ...을 제공
                    - 분류 : {Duplicate, Not Duplicate}
            - 단일 문장 분류 task
                - SST2
                    - 문장
                    - 분류 : {Positive, Negative}
        - Token Level
            - QA task
                - SQuAD
                    - 질문
                    - 문맥
                    - 정답
            - 개체명 인식(Named Entity Recognition,NER)
                - CoNLL 2003 NER
                    - 문장
                    - 라벨
    - BERT의 결과
        - 다양한 task에 적용가능한 범용성을 보여줬으며, fine-tuning을 통해 여러 NLP 과제에서 SOTA 성능을 이끌어 냄
        - Layer의 수, hidden state의 크기, attention head의 수가 클수록 성능이 향상되는 경향을 보였다.
    - BERT의 한계
        - Encoder 기반 model인 BERT는 주어진입력을 잘 이해하도록 학습되지만, sequence를 생성해야 하는 task에는 적합하지 않다.(ex. 기계번역, 텍스트 생성 등)
        - 생성 task에서는 autoregressive하게 즉 한 번에 한 단어씩 생성해야하는데, 이를 자연스럽게 수행하지 못하기 때문에, 생성 task엔 decoder 기반 모델을 주로 사용한다.

3. Encoder-Decoder 모델

- Encoder와 Decoder의 장점을 모두 결합
- T5(Text-to Text Transfer Transformer)
  - Transformer Encoder-Decoder 구조 기반의 모델
  - 모든 task를 Text-to-Text format으로 변환해 하나의 모델로 학습
- 학습 방법
  - Span Corruption
  - Encoder-Decoder 구조에서는 Encoder가 입력 문장을 모두 보고, 그 정보를 바탕으로 Decoder가 출력을 생성한다.
  - 따라서, 학습을 위해, Span Corruption이라는 과정을 수행
  - 과정
    - 입력문장에서 연속된 token을 무작위로 선택해 제거
    - 제거된 부분을 특수 placeholder token으로 치환
    - decoder는 이 placeholder에 해당하는 원래 span을 복원하도록 학습
    - 이를 통해 언어 모델처럼 훈련을 수행할 수 있다.
    - T5는 NLU(GLUE, SuperGLUE), QA(SQuAD), 요약(CNNDM), 번역(En->De, En->Fr,En->Ro) 등의 task에서 모두 좋은 성능을 보여 범용적으로 활용될 수 있는 모델임을 입증

4. Decoder 모델

- 전형적인 언어 모델 구조, 문장 생성에 유용(미래 단어 참조 불가)
- Finetunning Decoder
  - Transformer의 Decoder는 사전학습 단계에서 다음 단어 예측(Next Token Prediction)을 학습
  - 생성 task에 활용할 때
    - 사전학습 때와 동일하게 다음 단어 예측 방식으로 fine-tuning한다.
    - 따라서, decoder는 대화나 요약 task 등 출력이 sequence인 task에 자연스럽게 적합한다.
  - 분류 task에 활용할 때
    - 마지막 hidden state 위에 새로운 linear layer를 연결해 classifier로 사용
    - 이때, linear는 아예 처음부터 다시 학습해야하며, fine-tuning 시 gradient가 decoder가 전체로 전파된다.
- GPT-1
  - Transformer 기반의 Decoder 모델
  - Autoregressive LM(왼쪽->오른쪽 단어 예측) 방식으로 사전학습되었다.
- GPT-1의 fine-tuning방법(분류 task)
  - GPT-1은 다음 단어 예측이라는 언어모델의 학습 목표를 최대한 유지하면서, fine-tuning을 수행
  - NLI (두 문장을 입력받아 관계를 함의, 모순, 중립 중 하나로 분류) "
    - 전제, 가설 => Entailment

- 입력토큰에 특수한 토큰([START], [DELIM], [EXTRACT])을 붙여 분류 문제를 처리했고, [EXTRACT] 위치의 hidden state에 classifier를 연결해 사용

5. In-Context Learning

- GPT-3
  - 2020년 GPT-2에서 모델의 parameter size를 키워(1750억), 별도의 파인튜닝 없이 컨텍스트 안의 에시만 보고도 새로운 테스크를 수행할 수 있게 되었다.
  - 이런 능력을 In-Context Learning이라고 한다.
  - 모델에 에시와 함께 어떤 테스크를 할지 지정해주면, 모델이 그 패턴을 따라가는 식으로 동작하면서, 완벽하진 않지만 그럴듯하게 task를 수행하는 모습을 보인다.
  - 이러한 능력은 모델의 크기, 즉 parameter size가 커질수록 더 강력하게 나타났으며, zero-shot, one-shot, few-shot 모두에서 일관된 성능 개선이 관찰됨
  - 또한, 모든 parameter size의 모델에서 shot의 수가 많을 수록 task 수행이 향상
  - zero-shot < one-shot < few-shot
  - In-Context Learning의 발견으로 인해, Prompt의 중요성이 대두되었다.
  - 하지만 단순한 few-shot prompting만으로는 여러 스텝을 거쳐야 하는 문제들을 풀기 어려웠는데, 이를 해결하기 위해 Chain-of-Thought(CoT) prompting 방식이 등장
  - CoT prompting은 모델이 문제 해결 과정에서 논리적인 사고단계를 거쳐 최종 답을 도출하도록 유도하는 prompting 기법
  - CoT prompting은 일반적인 prompt 방식보다 훨씬 우수한 성능을 보이며 새로운 SOTA 성과를 달성
  - fine-tuning을 수행한 모델들보다 더 좋은 성능을 보임
- Zero-shot Chain-of-Thought prompting
  - 하지만 기존 CoT Prompting은 few-shot 예시가 필요
  - 예시가 없는 경우, 추론 과정이 나타나지 않아 성능 저하 발생
  - 이를 보완하기 위해, 질문 뒤에 "Let's think step by step"이라는 한 문장을 추가해 모델이 스스로 추론 단계를 생성하도록 유도하는 Zero-Shot prompting 방법 등장

1. CNN 살펴보기

   1-1. CNN vs. FCN

   *FCN(Fully-Connected Layer): 입력을받아서 출력으로 변환하는 신경망의 기본 모듈

   FCN 변환 예시:

   입력: 이미지가 32x32x3인 고차행렬이라고 가정 => 1x3072 1차원

   모델상수: 모델이 학습해야하는 파라미터 => 10 x 3072 weights

   출력: 10x1의 1차원 벡터 => 변환된 데이터를 10개 중 1개로 분류

   *CNN 모델

   기본 구조: 합성곱-> 활성화->풀링->완전연결층

   합성곱 레이어: 입력 이미지를 필터와 연산하여 특징 맵(feature map)을 뽑아내는 모듈

   :1차원 구조로 변환하는 FCN과 달리 3차원 구조를 그대로 보존하면서 연산

> 컨볼루션: 필터(3x5x5)를 이미지(3x32x32)상에서 이동시키면서 내적을 반복수행-> 내적으로 구한 모든 값을 출력 제공
> : 차원을 반드시 주의! 필터는 항상 입력의 깊이/채널 축과 동일한 차원을 가진다

특정위치에서 필터와 동일 사이즈의 이미지 영역 간 내적한 결과 값, 필요 연산은 (3*5*5 =75)차원 내적 곱 +bias 벡터
-> 필터의 이동경로를 따라 모든 연산 값을 모아 활성화 맵을 출력

합성곱 레이어의 예시: 박스필터

각 필터값과 대응되는 입력값의 곱을 모두 합산 -> 입력 가운데 위치의 출력값
합성곱 레이어의 특징
1. 출력 해상도는 입력 해상도 - 필터 해상도 + 1로 도출
2.만약 입출력 해상도를 유지하고 싶다면 -> 출력값 중 정의되지 않은 경우, 0 혹은 가장 가까운 출력값으로 대체한다(패딩)

2. CNN 모델구조

2-1. 중첩
모델 상수(파라미터)를 증가시키면? 어차피 Linear classifier
-> 비선형 블록(Conv[Linear] + ReLU[비선형 변환])과 함께하면 모델링 파워 향상

2-2. 필터
필터 시각화: 학습된 필터 시각화를 통해 각 모델(구조)가 학습한 정보를 이해가능하다.
cf. 선형모델의 필터: 각 카테고리의 템플렛(전형적모습) 역할
FCN의 필터: 다양한 템플렛 제공
CNN의 필터: 계층별로 이미지의 지역적 템플렛(예: 엣지, 색상 등)

2-3. 수용영역(리셉티브 필드)
CNN이 이미지를 처리하면서 한 번에 볼 수 있는 영역의 크기, 네트워크의 시야
일반적으로 네트워크가 깊어질 수록 수용역역도 넓어짐 -> 폭 넓은 맥락 이해가 가능하다
CNN 레이어는 이미지 작은 부분인 "지역 정보"를 추출하는데 유리하게 설계, 이미지 전체(예: 맥락)의미 파악이 필요
따라서 "지역 정보 + 이미지 전체의 맥락" 을 이해/활용하기 위한 설계가 필요하다.
->중첩 (2-1)과 수용영역(2-3)을 이용한다.
고해상도 이미지 처리 시, 많은 레이어를 통과해야한다. 이는 연산,비용,학습 가능성 고려할 때 비실용적임
실제 해법: 입력 사이즈를 줄여 모델에 입력함

2-4. 풀링
효율적 연산 및 위치 변화의 강건성 확보
지역적 특징을 더 압축된 형태로 전달하여, 합성곱 층이 더 넓은 맥락을 이해하는 데 기여한다,
입력 크기가 줄면 CNN의 연산량이 크게 줄어들기때문에 CNN 레이어의 출력을 줄여 연산 효율성을 확보해야한다.

> 위치 변화 강건성: 입력 내 객체의 위치가 다소 변해도 동일한 출력을 제공, 사실상 저해상도 정보에 근거하여 작업 수행하므로 몇 화소 이동은 무시됨

> 맥스풀링: 정해진 커널 사이즈로 이미지를 나누어 각 영역 내 가장 큰 값을 선택하는 연산 (모델 상수 학습 X)
> : 이렇게 하면 연산해야 할 데이터의 양이 줄어들어 학습과 추론 속도가 빨라지고, 메모리 사용량도 절감된다.

2-5. 스트라이드 합성곱
풀링의 한계 계선
효과: 풀링 처럼 입력 해상도를 줄임 (연산 효율과 위치 강건성)

> 일반 합성곱: 필터를 1칸씩 이동하면서 연산 수행(출력값 계산)
> 스트라이트 합성곱: 필터를 스트라이드 값만큼(S칸) 이동한 후 출력 연산
> : 풀링은 학습 상수가 없지만 스트라이드는 커널을 동시에 학습한다.
> :스트라이드 합성곱은 해상도 저하로 인한 정보 손실이 적고, 풀링 + 합성곱을 하나의 레이어로 대체함
> :고도화된 CNN에서는 스트라이드 합성곱을 풀링 대신 활용되는 경향이 있다.

3. CNN 기반 모델 변천사
3-1. AlexNet(2012), 딥러닝 CNN의 본격적인 시작점
5개의 합성곱 계층과 3개의 완전 연결 계층으로 구성된 8계층 CNN모델
3-2. VGGNet(2014),3x3 작은 필터를 깊게 쌓은 단순하면서도 강력한 구조
5개의 합성곱 블록 + 맥스 풀링 구조
:작고 단순한 필터를 깊게 쌓으면 성능이 올라간다.
:단순 설계로 모델 해설에 이상적이며, 특징 추출기, 전이 학습에 강력한 베이스라인을 가짐(파라피터가 많고 연산량 요구가 매우 크다, 거대한 모델
3-3. GoogLeNEt
3-3.ResNet(2015), 잔차 연결을 도입하고 기울기 소실 문제를 해결함
최종 요약: 깊은 모델 학습을 위한 중요한 전진(+100 레이어 학습 가능), 깊은 모델의 강력함을 다시 확인
:제안 당시 모든 벤치마크에서 최고 성능 달성, 지금도 CNN 기반 구조 중 가장 활발하게 활용
합성곱 블록(VGG 유사)과 잔차 블록
: 잔차 블록은 블록입력을 그대로 출력에 더해주는 지름길 연결이 특징. Inception모델에서 차원 축소로 활용된 1x1 합성곱을 적용, 연산 효율 개선

> 등장 배경
> 단순히 레이어를 깊게 하면 오히려 성능이 떨어짐
> -해법: 작은 모델의 성능은 "최소한" 누릴 수 있도록 작은 레이어의 추정값(입력값)을 그대로 후속 레이어에 제공하여
> 최소한 작은 레이어 수준의 성능은 보장
> 잔차블럭:
> 입력 X가 두 경로(변환경로, Shortcut 경로)로 나뉘어 전달됨 -> 마지막에 두 값을 더한다.
>
> > 일반 잔차 블록 vs 보틀넥 잔차 블록, 연산 효율을 위해 보틀넥 잔차구조 도입(더깊은 모델을, 더 적은 연산으로)

> Stem 구조
> 기존 모델의 효율성 레시피를 잘 활용
> (스템 구조를 모델 초기에 도입, 입구 데이터 해상도를 1/4로 줄임, 여러개의 FC레이어를 제거하고 전역 평균 풀링을 사용-> 마지막 1개 FC만 적용)

3-4. MobileNet, 모바일.임베디드 환경 최적화
기존 합성곱은 공간과 채널을 동시에 처리하여 과도한 연산량아 요구됨

해법: 공간과 채널을 두 단계로 분리하여 처리한다.
> 깊이별 합성곱:각 채널별 독립적 3x3 합성곱 수행

> 화소별 합성곱: 1x1 합성곱을 채널 방향으로 적용
> 효과: 연산량 기존 대비 9배 가량 절감, 모델 상수 대폭 감소

딥러닝 학습 안정성 및 모델 구성 요소 요약

1. 학습의 일반적인 문제

- 학습 불안정: 손실 폭발, 학습 멈춤(기울기 소실), 수렴 실패.
- 과적합: 훈련 데이터에 과도하게 맞춰져 검증/실전 성능이 저하되는 현상.
- 느린 수렴: 최적점에 도달하는 데 불필요하게 많은 에폭 소모.
- 좋은 모델 구조만으로는 성능을 보장할 수 없습니다.

2. 모델 구성 및 안정화 전략

2.1. 활성화 함수 (Activation Function)

- 목적: 신경망에 비선형성을 부여하여 복잡한 패턴 학습을 가능하게 합니다.
- Sigmoid (출력 [0, 1]):
  - 문제: 기울기 소실, 편향된 업데이트(출력 항상 양수), 계산 비용 높음.
- Tanh (출력 [-1, 1]):
  - 장점: 0 중심 대칭으로 학습 안정성 높음.
  - 문제: 기울기 소실 문제 여전, 계산 비용 높음.
- ReLU (f(x) = max(0, x)):
  - 장점: 양의 영역에서 기울기 소실 크게 감소, 계산 효율성 및 학습 속도 빠름.
  - 문제: 죽은 뉴런(Dead ReLU) 문제 발생, 0을 기준으로 비대칭.

- Leaky ReLU (f(x) = max(0.01x, x)):
  - 장점: ReLU 장점 수용, 죽은 뉴런 문제 해결.
- ELU:
  - 장점: ReLU 장점 수용, 평균 출력이 0 근처에 위치하여 학습 안정성 높음.
  - 문제: 계산 복잡성 증가, 하이퍼파라미터 알파(α) 설정 필요.

2.2. 데이터 전처리

- 목적: 학습/검증/테스트 데이터 간의 조건을 통일하고 모델 입력에 적합하게 조정합니다.
- 통일 요소: 이미지 해상도, 색상, 밝기, 정규화.
- 정규화 방식:
  - AlexNet: 학습 데이터 전체의 평균 이미지를 입력에서 뺌.
  - VGG: R, G, B 채널별 평균을 입력에서 뺌.
  - ResNet: 채널별 평균을 빼고 채널별 표준편차로 나눔 (표준화).

2.3. 모델 상수 초기화 (가중치 W, 편향 b)

- 0 초기화: 모든 뉴런의 출력 및 기울기가 동일해져 학습이 불가능합니다.
- 랜덤 초기화: 깊은 모델에서는 분산(상수 크기)에 따라 기울기 소실/폭발 위험이 있습니다.
- Xavier 초기화: 가중치 초기 분포의 분산을 입력 차원(Nin)에 맞춰 설계합니다. (입출력 대칭 분포 가정)
- He 초기화: 가중치 초기 분포의 분산을 2 * 입력 차원(2/Nin)에 맞춰 설계합니다. ReLU와 같은 비대칭 활성화 함수에 적합합니다.
- ResNet의 등장 배경: 잔차 연결(Skip Connection)을 사용해 깊은 신경망도 안정적으로 학습할 수 있으며, 초기에는 He 초기화 등을 통해 안정적인 학습 출발을 유도합니다.

2.4. 모델 정규화 (과적합 방지)

- 가중치 감소 (Weight Decay/Regularization): 큰 가중치에 패널티를 부여하여 과적합을 방지합니다.
  - L2 정규화 (Ridge): 가중치 제곱에 패널티. 모든 가중치가 골고루 작아집니다.
  - L1 정규화 (Lasso): 가중치 절대값에 패널티. 중요하지 않은 가중치를 완전히 0으로 만들어 희소한 모델을 만들고 특성 선택 효과가 있습니다.
  - Elastic Net (L1+L2): L1과 L2의 장점을 결합합니다.
- Dropout: 학습 과정에서 일부 뉴런을 확률적으로 끔(0으로 설정)으로써 과적합을 방지하고 일반화 성능을 높입니다 (앙상블 효과).
  - 단점: 학습 시간 증가, 배치 정규화가 적용된 모델에서는 비선호됨.
  - Inverted Dropout: 학습 시 살아남은 뉴런의 출력을 스케일링하여 테스트 시 추가 연산 없이 출력 크기를 유지합니다.

3. 학습 안정성 전략 (학습률 및 종료)

- 학습률(Learning Rate) 조정: 큰 값에서 시작하여 에폭이 지날수록 작은 값으로 조정합니다.
  - 계단식 변경: 일정 에폭마다 계단식으로 줄임.
  - 코사인 변경: 코사인 함수 곡선처럼 부드럽게 줄여나감.
  - 선형 변경: 직선으로 줄여나감 (사전학습/미세조정).
- 빠른 종료 (Early Stopping): 검증 손실을 모니터링하여 더 이상 성능이 좋아지지 않을 때 학습을 조기에 멈춥니다.

하이퍼파라미터 (Hyperparameter)

- 정의: 학습 전에 사용자가 정하는 값 (예: 학습률, 배치 사이즈, 드롭아웃 비율).
- 탐색 방식:
  - 그리드 탐색 (Grid Search): 후보 값들을 격자처럼 모든 조합으로 탐색.
  - 랜덤 탐색 (Random Search): 무작위로 값을 선택해 탐색.
- 체크리스트:
  i. 초기 손실 확인: CE 손실이라면 log(C)와 유사해야 정상입니다.
  ii. 작은 샘플 과적합: 작은 샘플로 고의로 과적합(정확도 100%)을 시도하여 모델의 학습 능력을 확인합니다.
  iii. 최적 학습률 탐색: 모든 데이터로 손실이 실제로 줄어드는 학습률을 우선 탐색합니다.
  iv. 좋은 조합 탐색: 후보 학습률과 가중치 변형 방식 중 초기 몇 에폭 내에 손실 감소 및 검증 성능이 좋은 조합을 선택합니다.
  v. 학습률에 따른 손실곡선 관찰: 초반 손실 유지 또는 손실 감소 후 정체 시 학습률 변경을 시도합니다.
  vi. 과적합/약한 모델 판단: 학습/검증 정확도가 크게 벌어지면 과적합, 결과 차이가 매우 적다면 모델의 힘이 약하다고 판단합니다.

1015 | 2-1. 자연어 처리 기본

1. 워드 임베딩과 순환신경망 기반 모델(RNN & LSTM)

1. 단어를 숫자로 표현

- 워드 임베딩(word embedding)
  - one-hot encoding
    - a. 규칙 기반 혹은 통계적 자연어 처리 연구의 대다수는 단어를 원자적(쪼갤 수 없는) 기호로 취급한다.
    - b. 벡터공간관점에서 보면, 이는 한 원소만 1이고 나머지는 모두 0인 벡터를 의미
    - c. 차원 수(= 단어 사전 크기)는 대략적으로 음성 데이터 2만개, Penn Treebank(PTB) 코퍼스(5만) - big vocab(50만) - Google 1T(1300만)
    - d. 이를 one-hot(원핫) 표현이라고 부르고 단어를 one-hot 표현으로 바꾸는 과정을 one-hot encoding이라고 한다.
  - one-hot encoding의 단점 및 문제점
    - a. 검색 query vector와 대상이 되는 문서 vector들이 서로 직교하게 되어, one-hot vector로는 유사도를 측정할 수 없다.
    - b. 차원의 저주(Curse of Dimenstionality)
      - (a) 고차원의 희소 벡터를 다루기 위해선 많은 메모리가 필요하다.
      - (b) 차원이 커질 수록 데이터가 점점 더 희소(sparse)해져 활용이 어렵다.
    - c. 의미적 정보 부족
      - (a) 비슷한 단어라도 유사한 벡터로 표현되지 않는다.
      - (b) 은행과 금융은 의미적으로 밀접하지만, one-hot encoding에서는 전혀 무관한 vector로 취급된다.(의미적으로 밀접해도 one-hot encoding에서는 각각을 무관하게 취급하거나 의미적으로 전혀 관계가 없어도 유관하게 취급할 수가 있다.)
  - word embedding이란 단어를 단어들 사이의 의미적 관계를 포착할 수 있는 밀집(dense)되고 연속적/분산적(distributed) vector 표현으로 나타내는 방법
    - a. one-hot encoding에선 은행과 금융이 완전히 독립적인(무관한) vector로 표현되었지만,
    - b. word embedding에선 두 단어의 vector가 공간상 서로 가깝게 위치하며, 이를 통해 의미적 유사성을 반영할 수 있다.
    - 대표적인 기법
      - (a) Word2Vec
        - 단어의 표현을 간단한 인공 신경망을 이용해서 학습
        - 주요 아이디어는 각 단어와 그 주변 단어들 간의 관계를 예측
        - 주요 알고리즘

        - Skip-grams(SG) 방식
            - 중심 단어를 통해 주변 단어들을 예측하는 방법
            - 단어의 위치(앞 또는 뒤)에 크게 구애 받지 않는다.
            - 윈도우 크기(window size) : 중심 단어 주변 몇개 단어를 문맥으로 볼 것인가?
            - 장점 : 적은 데이터에도 잘 동작하며, 희귀 단어나 구 표현에 강하다.
            - 단점 : 학습 속도가 느리다.
        - Continuous Bag of Words(CBOW) 방식
            - 주변 단어들을 통해 중심 단어를 예측하는 방법
            - 문맥 단어들의 집합으로 중심 단어를 맞춘다.
            - 주변 단어와 함께 묶어서 input data로 활용해서 예측
            - 주변 단어들의 집합이 주어졌을 때, 그 문맥과 함께 등장할 수 있는 단일 단어를 예측
            - 장점 : 학습 속도가 빠르며, 자주 나오는 단어에 강하다.
            - 단점 : 희귀 단어 표현에 약하다.

2. 언어는 순서가 중요함

- 순차적 데이터의 특징
  - 자연에 수많은 순차적(Sequential Data)가 존재
  - 데이터가 입력되는 순서와 이 순서를 통해 입력되는 데이터들 사이의 관계가 중요한 데이터
  - 대표적으로 오디어, 텍스트, 비디오 데이터 등이 있음.
  - 1) 순서가 중요하다. 데이터의 순서가 바뀌면 의미가 달라진다.
  - 2) 장기의존성(Long-term dependency) : 멀리 떨어진 과거의 정보가 현재/미래에 영향을 준다.
  - 3) 가변길이(Variable length) : 순차 데이터는 길이가 일정하지 않고, 단어 수도 제각각이다.
    - 처리 방법
        - 일반적인 모델인 선형회귀, MLP 등으로는 처리가 불가능하다.
        - 따라서, Sequential Models이 필요하다.(RNN, LSTM, Transformer 등)

3. RNN

- 문맥을 기억하는 신경망
  - 전통적인 인공신경망은 MLP, CNN 등이며, 이는 고정된 길이의 입력을 받기 때문에, 가변 길이의 데이터를 처리하기에 적합하지 않는다.
  - RNN은 가변 길이의 입력을 입력 받을 수 있고, 이전 입력을 기억할 수 있기 때문에, 순차적 데이터 처리에 적합한 architecture이다.
  - 대표적으로, one to one, many to one, one to many, many to many 등이 있다.
    - Architecure 설명

- 전통적인 인공신경망은 MLP, CNN과 달리, RNN은 이전 시점의 정보를 담는 hidden state를 가지고 있다.
- 따라서, 입력 시퀀스 벡터 x를 처리할 때, 각 시점마다 recurrence 수식을 적용하며 hidden state를 업데이트한다.

$$h_t = f_W(h_{t-1}, x_t)$$

$$h_t = f_W(h_{t-1}, x_t)$$

$$h_t : 새로운 hidden state$$

$$f_W : 처리함수(파라미터 W)$$

$$h_{t-1} : 이전 hiddenstae$$

$$x_t : 입력 vector(Timestep)$$

- RNN은 각 시점마다 동일한 가중치(weight)를 사용하기 때문에, 깊은 feedward 신경망보

$$y_t = W_{hy}h_t$$

$$h_t = tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

- RNN의 특징
  - RNN은 한 번에 하나의 요소를 처리하고, 정보를 앞으로 전달한다.
  - RNN은 각 층이 하나의 시점을 나타내는 깊은 신경망처럼 보인다.
  - RNN은 hidden state를 유지하면서 가변 길이 데이터를 처리할 수 있다.
  - RNN의 출력은 과거 입력에 영향을 받는다는 점에서, feedward 신경망과 다르다.
- RNN의 단점
  - 기울기 소실(vanishing gradient) 문제
  - vanishing gradient 문제란?
    - 딥러닝에서 역전파 시 앞쪽 측의 기울기가 0에 가까워져서 장기 의존성 학습이 어려

    - 역전파 과정에서 작은 값들이 계속 곱해진다.
    - 과거 시점에서 온 오차 신호는 갈수록 더 작은 기울기를 갖게 된다.
    - 결국 파라미터들이 장기 의존성은 학습하지 못하고, 단기 의존성만 포착하게 된다.

4. LSTM

- LSTMs란 기울기 소실문제를 해결하기 위해, 1997년에 제안된 RNN의 한 종류
  - LSTMs의 특징
    - 시점 t에서 RNN은 길이가 n인 vector hidden state h_t와 Cell state C_t를 가진다.

        - Hidden state는 short-term information을 저장한다.
        - Cell state는 long-term information을 저장한다.
    - LSTMs cell state에서 정보를 읽고(read), 지우고(erase), 기록(write)할 수 있다.
    - LSTMs는 3가지 게이트를 통해 어떤 정보를 지우고 쓰고 읽을지를 결정한다.
        - Forget gate
            - 이전 cell state에서 무엇을 버리고 무엇을 유지할지 결정한다.

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

- Input gate
    - 새 정보 중 얼마나 cell state에 쓸지 결정한다.

$$i_t = sigma(W_i cdot[h_t, x_t] + b_i)$$

$$tildeC_t = tanh(W_C cdot[h_{t-1}, x_t] + b_C)$$

- New Cell Content
    - New Cell Content : Cell에 기록될 새로운 후보

$$C_t = f_t * C_{t-1} + i + t * tildeC_t$$

- Output gate
    - cell state 중 얼마나 hidden state로 내보낼지 결정한다.

$$o_t = sigma(W_o[h_{t-1}, x_t] + b_o)$$

$$h_t = o_t * tanh(C_t)$$

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

2. 자연어 생성 모델 (Seq2Seq, Attention)

1. 언어 모델이란?

1-1. 개념

- 인간의 두뇌가 자연어를 생성하는 능력을 모방한 모델
- 단어 시퀀즈 전체에 확률을 부여하여 문장의 자연스러움을 측정
- 한 문장의 확률은 각 단어의 조건부 확률들의 곱으로 표현할 수 있음

→ 언어 모델은 **문장 내 단어의 등장 확률 분초를 모델링하는 것**을 의미

→ 즉, 주어진 단어들의 연속이 나올 확률을 계산하여 **다음 단어를 예측**하거나

→ 문장의 자연스러움(유창성**)을 평가**하는 모델

1-2. 일상 속의 언어모델 ex. 텍스트 자동완성 기능

1-3. 대표적인 언어모델 : N-gram 언어모델

- n-gram : 연속된 n개의 단어 묶음
  - unigram: `"The"` , `"students"` , `"opened"` , `"their"`
  - bigram: `"The students"` , `"students opened"` , `"opened their"`
  - trigram: `"The students opened"` , `"students opened their"`
  - 4-gram: `"The students opened their"`
- 언어모델의 목표는 다음 단어의 조건부 확률을 추정하는 것
- 다른 예시 : 4-gram 언어 모델
- 예시 데이터:
  - `"students opened their"` : 1000회 등장
  - `"students opened their books"` : 400회 등장
    → P(books ㅣ students opened their)=0.4P(\text{books} | \text{students opened their}) = 0.4P(books ㅣ students opened their)=0.4
  - `"students opened their exams"` : 100회 등장
    → P(exams ㅣ students opened their)=0.1P(\text{exams} | \text{students opened their}) = 0.1P(exams ㅣ students opened their)=0.1

1-4. 언어모델 사용 예시 : Statistical Machine Translation, SMT

- 1990년대~2010년대 초반까지는 번역 문제를 **확률적 모델링**으로 접근
- 예시 : 한국어 → 영어
  - 목표: 주어진 한국어 문장 x 에 대해, 가장 적절한 영어 문장 y 를 찾는 것

$$argmax_y P(y \mid x)$$

- **Bayes 정리를 이용한 분해**P(y ∣ x)=P(x)P(x ∣ y)P(y)⇒argymaxP(x ∣ y)P(y)

$$= argmax_y P(x \mid y)P(y)$$

- 두 개의 구성요소로 분리:
  - **번역모델 P(x ∣ y)**
    - 단어와 구가 어떻게 번역되어야 하는지 모델링
    - (한국어, 영어) 말뭉치로 학습
      → 영어 문장 yyy가 주어졌을 때 한국어 문장 xxx가 생성될 확률
  - **언어모델 P(y)**
    - 유창한 영어 문장을 쓰는 방법을 모델링
    - 영어 데이터로 학습
- 한계점 : SMT의 문제
  - **구조적 복잡성**: 문장 구조를 반영하기 어렵다.
  - **수작업 의존도**: 문장 정렬, 단어 정렬, 구문 규칙 등의 수작업 필요.
  - **언어별 자원 구축 부담**: 각 언어쌍마다 별도의 말뭉치와 규칙이 필요.
    → 결과적으로 유지·확장성이 떨어짐
- 발전 방향 : **Neural Machine Translation (NMT)**
  - 위 한계를 극복하기 위해 **신경망 기반 번역모델(NMT)**로 전환됨.
  - NMT는 **하나의 신경망이 전체 번역 과정을 자동으로 학습**하여 번역모델과 언어모델을 통합적으로 처리함

2. Seq2Seq

2-1. Neural Machine Translation (NMT)

- 개념
  - Neural Machine Translation 이란 인공 신경망을 이용해 기계 번역을 수행하는 방법
  - 이 때 사용되는 신경망 구조를 sequence-to-sequence (Seq2Seq)이라 하며, 두 개의 RNNs으로 이루어짐
    - 2014년 Google의 "Sequence to Sequence Learning with Neural Networks"라는 논문에서 처음 소개
- Translation이 어려운 이유
  - 번역 문제는 입력과 출력의 길이가 다를 수 있음
    - 영어: the black cat drank milk (5개의 단어)
    - 프랑스어: le chat noir a bu du lait (7개의 단어)

  → 따라서 NMT에서는 길이가 다른 시퀀스 간의 매핑을 처리할 수 있어야함

- Seq2Seq의 아이디어
  - 2개의 LSTM을 이용하자
    - 한 LSTM은 입력 시퀀스를 한 타임스텝씩 읽어 고정된 차원의 큰 벡터 표현을 얻기 (Encoder)
    - 다른 LSTM은 앞에서 얻은 벡터로부터 출력 시퀀스를 생성하기 (Decoder)

2-2. Seq2Seq Architecture

- Seq2Seq의 구성
  - Encoder와 Decoder로 이루어진다.
    - Encoder는 입력 문장에 담긴 정보를 인코딩
    - Decoder는 인코딩된 정보를 조건으로 하여 타겟 문장(출력)을 생성
- Seq2Seq의 다양한 적용
  - Seq2Seq 구조는 기계번역 외에도 다양한 태스크에 적용 가능
    - 요약 : 긴 길이의 문서를 읽고, 짧은 길이의 문장으로 요약된 텍스트를 출력하는 태스크
    - 대화: 사용자의 발화를 기반으로, 맥락에 맞는 대답(출력 텍스트)을 생성하는 태스크
    - 코드 생성: 자연어로 작성된 설명 혹은 명령어를 입력 받아, 그에 대응하는 프로그래밍 코드 혹은 쿼리를 출력하는 태스크
- Seq2Seq의 학습 수행 (Teacher Forcing)
  - Seq2Seq 모델은 인코더와 디코더가 하나의 통합 네트워크로 연결되어 있음
  - 디코더에서 발생한 오차는 역전파 과정을 통해 입력을 처리한 인코더까지 전달되어 전체 네트워크가 End-to-End로 동시에 최적화된다.
  - 학습 초반에는 모델의 예측 능력이 떨어지기 때문에 학습이 불안정할 수 있음
  - Teacher Forcing이란?
    - 모델이 스스로 예측한 단어 대신 정답 단어를 디코더 입력으로 강제로 넣어줌으로써 훨씬 안정적이고 빠르게 학습을 수행하는 방법이다.
- Seq2Seq의 토큰 출력 방법 (Greedy Inference)
  - 토큰을 출력하는 방법 중 하나로, 각 단계에서 가장 확률이 높은 단어를 선택
  - 한계
    - 되돌리기가 불가능
    - 예시 : le _____ → le chien ____ → …
      chein이 오답이어도 돌아갈 방법이 없지
  - Beam Search
    a. 매 단계마다 k개의 가장 유망한 후보 유지
    b. 후보가 에 도달하면, 완성된 문장으로 리스트 추가
    c. 문장이 충분히 모이면 탐색 종료
    d. 각 후보들의 점수를 로그 확률의 합으로 구해 최종 선택

3. Attention

3-1. Seq2Seq의 한계 : the bottleneck problem

- Bottleneck problem 개념
  - 인코더는 입력 문장 전체를 하나의 벡터로 요약하는데, 마지막 hidden state에 문장의 모든 의미 정보가 담긴다
  - 고정 길이 벡터 하나에 모든 문장의 의미를 압축하다 보니 정보 손실이 생길 수 있는데, 이를 bottleneck problem이라고 한다.
- Attention의 인사이트
  - Attention은 디코더가 단어를 생성할 때, 인코더 전체 hidden state 중 필요한 부분을 직접 참조할 수 있도록 한다.
  - 즉, 매 타임스텝마다 "어떤 단어/구절에 집중할지"를 가중치로 계산해, bottleneck 문제를 완화했다.

3-2. Attention의 효과

- Attention mechanism은 많은 장점이 존재한다.
  i. NMT 성능 향상
    - 디코더가 소스 문장 전체가 아닌, 필요한 부분에만 집중할 수 있기 때문이다.
  ii. Bottleneck Problem 해결
    - 디코더가 인코더의 모든 hidden states에 직접 접근할 수 있다.
  iii. Vanishing Gradient Problem 완화
    - Attention은 멀리 떨어진 단어도 직접 연결할 수 있게 해준다.
- Attention의 효과: 해석 가능성(Interpretability)
  - Attention 분포를 보면, decoder가 어떤 단어를 생성할 때, 입력 문장의 어느 부분에 집중했는지 확인할 수 있다.
  - 즉, 모델이 내부적으로 참고한 근거를 사람이 파악할 수 있음
    → 모델의 의사결정 과정을 해석할 수 있는 단서
- Attention의 효과: 정렬(Alignment)
  - 기계번역에서는 전통적으로 단어 alignment 모델을 따로 학습해야 했다.
  - 하지만, attention은 통해 decoder가 필요한 입력 단어에 자동으로 집중하기 때문에, 단어와 단어 간의 매핑 관계를 자연스럽게 학습한다.
- Attention : Query와 Values
  - Seq2Seq에서 attention을 사용할 때, 각 decoder의 hidden state와 모든 encoder의 hidden states 간의 관계를 Query와 Values의 관계로 볼 수 있다.
  - 이 관점에서 Attention 과정을 정리해보면:
    a. Query와 Values 사이 유사도 점수(score) 계산 (예: dot-product, multiplication, additive 등)
    b. Softmax를 통해 확률 분포(attention distribution) 얻기
    c. 분포를 이용해 values를 가중합 → context vector (attention output)

3. Transformer

1. Self-Attention

1-1. RNN

- RNN의 필요성
  - RNN은 정보를 hidden state로 전달하며 순차적 의존성을 형성한다.
  - 그에 반해 Attention은 필요한 순간, 입력 전체에서 직접 정보를 전달한다.
    → RNN이 하던 "정보 전달"을 Attention이 더 효율적으로 수행할 수 있다면, 굳이 recurrence가 필요할까?

1-2. RNN의 한계점

1. 장기 의존성
   - RNN은 왼쪽에서 오른쪽으로 순차 전개되어, 먼 단어 쌍이 상호작용하려면 시퀀스 길이 만큼의 단계를 거쳐야 한다.
   - 따라서, 길어진 단계만큼 기울기 소실 혹은 폭발 문제가 발생해 장기 의존성을 학습하기 어렵다.
   - 또한, RNN은 입력된 선형 순서를 강하게 반영하기 때문에, 언어의 비선형적 의존성을 잘 잡아내지 못한다.
     - 예시: 그 책은 오랫동안 방치되어 먼지가 많이 쌓인 탓에… 매우 더러웠다.
2. 병렬화
   - Forward와 Backward pass 모두 시퀀스 길이 만큼의 단계가 필요하다.
   - 이처럼, 순차적인 연산이 진행되기 때문에, 병렬화가 불가능하다.
   - 이는, 병렬 연산에 강한 GPU 활용을 어렵게 만들며, 대규모 데이터 학습에 비효율적이다.

1-3. Attention은 어떨까 ?

- Attention은 각 단어의 표현을 query로 두고, value 집합으로부터 필요한 정보를 직접 불러와 결합한다.
- 이러한 Attention 메커니즘을 encoder-decoder 간이 아닌, 한 문장 내부에서 적용한다면 어떨까?
  → Self-Attention

1-4. Self-Attention의 장점은?

1. 순차적으로 처리해야 하는 연산 수가 시퀀스 길이에 따라 증가하지 않는다.
2. 최대 상호작용 거리 =O(1) → 모든 단어가 각 층에서 직접 상호작용한다.

1-5. Seq2Seq에서의 Attention

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

- 배경
  - 기존 Seq2Seq 모델은 입력 문장을 **하나의 고정된 벡터(Context Vector)** 로 압축한 뒤, 이를 디코더가 이용해 출력 문장을 생성
  - 하지만 문장이 길어질수록 이 벡터 하나에 모든 정보를 담기 어려워졌고, **Attention**이 도입되면서 이 문제를 해결하게
- 핵심 아이디어
  - 출력 문장(Decoder)의 각 단어를 생성할 때, 입력 문장(Encoder)의 **모든 단어들 중에서 "관련 있는 단어"에 더 집중(attend)** 하도록 하는 메커니즘.
  - **입력 문장(Encoder의 hidden states)** → Key, Value
  - **출력 문장(Decoder의 현재 hidden state)** → Query
- 예시
  - 입력 문장: `"I love cats"`
  - 출력 문장: `"나는 고양이를 좋아해"`
    - Decoder가 `"고양이를"` 을 생성할 때
      → Encoder의 단어들 중 `"cats"` 에 높은 가중치를 부여
      → `"I"` , `"love"` 에는 낮은 가중치
- 정리
  - Seq2Seq Attention은 **"출력 단어를 생성할 때 입력 문장의 어떤 단어에 집중할지"**를 결정하는 메커니즘이다.

1-6. Self-Attention

- Attention(문장 내의 다른 단어들 | 문장 내의 각 단어 "w")
- 배경
  - Self-Attention은 Seq2Seq처럼 "입력 → 출력" 관계가 아니라, **하나의 문장 내부 단어들끼리 서로 어떤 연관이 있는지**를 학습하는 메커니즘
- 핵심 아이디어
  - 하나의 단어를 표현할 때, 그 단어가 같은 문장 안의 다른 단어들과 어떤 관계를 맺는지를 반영
- 예시
  - 문장: `"The animal didn't cross the street because it was too tired."`
  - 단어 `"it"` 이 나올 때,
    → Self-Attention은 `"animal"` 과 `"it"` 이 같은 대상을 가리킨다는 관계를 학습한다.
    즉, `"it"` 을 표현할 때 `"animal"` 의 정보가 중요하게 반영된다.
- 정리
  - Self-Attention은 **"문장 내에서 각 단어가 다른 단어들과의 관계를 고려하여 표현을 만드는 메커니즘"**이다.
  - 모든 단어가 서로를 참조하면서 문맥적 의미를 포착한다.

1-7. Self-Attention - Key, Query, Value

- Attention에서 각 단어 "i"를 표현하는 Query와 Value 벡터가 있었듯이, Self-Attention에서는 각 단어 "i"를 표현하는 Query, Key, Value 벡터가 존재한다.
  i. 각 단어를 Query, Key, Value 벡터로 변환한다.
    - Query 벡터
      - 단어 i가 다른 단어로부터 어떤 정보를 찾을지를 정의하는 벡터
    - Key 벡터
      - 단어 i가 자신이 가진 정보의 특성을 표현하는 벡터
    - Value 벡터
      - 실제로 참조되는 정보 내용을 담고 있는 벡터
  ii. Query, Keys 간의 유사도를 계산해, softmax로 확률분포를 구한다.
    - 유사도 계산식

$$e_{ij} = q_i^T k_j$$

    - Softmax로 확률화

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{j'} \exp(e_{ij'})}$$

  iii. 각 단어의 출력을 Values의 가중합으로 계산

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

1-8. Self-Attention의 한계

- 이렇듯, Self-Attention은 단어 간 관계를 효율적으로 잡아내는 강력한 메커니즘이지만, 한계가 존재한다.
- 한계
  i. 순서 정보 부재 → 단어 간 유사도만 계산하기 때문에, 단어의 순서를 고려하지 않는다.
  ii. 비선형성 부족 → Attention 계산은 본질적으로 가중 평균 연산이라는 선형 결합에 불과하기 때문에, 복잡한 패턴이나 깊은 표현력을 담기 어렵다.
  iii. 미래 참조 문제 → 언어 모델은 시퀀스를 왼쪽에서 오른쪽으로 생성해야 하지만, Self-Attention은 모든 단어를 동시에 보기 때문에, 아직 생성되지 않아야 할 미래 단어를 참조한다.

1-9. Self-Attention의 한계 해결

1. Positional Encoding
   - 순서 정보 부재 문제를 해결하기 위해, Positional Encoding이라는 기법을 사용한다.
   - 각 단어 위치 i를 나타내는 위치 벡터를 정의해, 단어 임베딩 값에 더해 최종 입력으로 사용한다.
   - 두 가지 방법이 존재
     - Sinusoidal Position Encoding

        - 서로 다른 주기의 사인/코사인 함수를 합성해 위치 벡터를 만드는 방법
    - Learned Absolute Position Embedding
        - 위치 벡터를 모두 학습 파라미터로 설정해 학습 과정에서 데이터에 맞춰 최적화하는 방법
2. Feed-Forward Network 추가하기
    - Self-Attention 연산은 비선형 변환이 없어, 복잡한 패턴 학습에 한계가 존재한다.
    - 따라서, 각 단어 출력 벡터에 Feed-Forward Network (Fully Connected + ReLU 등)을 추가해 Self-Attention이 만든 표현을 깊고 비선형적인 표현으로 확장한다.
3. Masked Self-Attention
    - 단어를 생성할 때는 한 단어씩 순차적으로 미래 단어를 예측해야 하지만, Self-Attention은 기본적으로 모든 단어(미래 포함)를 동시에 참조한다.
    - 따라서, Attention Score를 계산할 때, 미래 단어에 해당하는 항목을 -∞로 설정해, 계산을 수행할 때 반영되지 않도록 한다.

1-10. Self - Attention 정리

- Self-Attention은 문장 내 모든 단어가 서로 직접 상호작용하여,
    i. 장거리 의존성을 효율적으로 포착하고
    ii. 병렬 처리를 가능하게 하는 메커니즘이다.
- 한계 해결 방법
    i. 순서 정보 부재 → Positional Encoding
    ii. 비선형성 부족 → Feed-Forward Network 추가
    iii. 미래 참조 문제 → Masked Self-Attention

epeat for number
f encoder blocks

2. Transformer

2-1. Transformer

- Transformer는 encoder-decoder 구조로 설계된 신경망 모델이다.
    - encoder: 입력 문장을 받아 의미적 표현으로 변환을 수행한다.
    - decoder: 인코더의 표현과 지금까지 생성한 단어들을 입력 받아, 다음 단어를 예측한다.
      → 이 중 decoder가 언어 모델과 같은 방식으로 동작한다.

2-2. Multi-Headed Attention

- 문장에서 같은 단어라도 여러 이유(문법적 관계, 의미적 맥락 등)로 다른 단어에 주목할 수 있다.
- 하지만, 단일 Self-Attention Head로는 한 가지 관점에서의 단어 간 관계 밖에 파악할 수 없다.
  → 따라서, 여러 Attention Head를 두어 다양한 관점에서 동시에 정보를 파악한다.

- Attention Head 1 - 단어의 문맥적 관계에 Attention → 문법적 의존성 관계 단어들에 주목
- Attention Head 2 - 단어의 시제에 Attention → 개체에 주목
- Attention Head 3 - 명사에 Attention

2-3. Scaled Dot Product

- Query와 Key의 차원이 커질수록, 두 벡터의 내적 값도 자연스럽게 커진다.
- 이 값이 너무 크면 softmax 함수가 출력이 지나치게 뾰족해져, 미세한 변화에도 큰 차이가 발생하고 gradient vanishing 문제가 생길 수 있다.

$$output_l = softmax(XQ_lK_l^TX^T) * XV_l$$

- 따라서, 내적 값을 그대로 사용하지 않고, 나눠주어 스케일을 조정한다.
- 이렇게 하면 감이 안정적으로 분포되어 학습이 훨씬 더 빠르고 안정적으로 진행되다.

2-3. Residual Connection

- 깊은 신경망은 층이 깊어질수록 학습이 어려워진다. (Gradient vanishing / exploding)
- 따라서, 단순히 Layer의 출력만 사용한다면 정보가 소실되어, layer가 전체를 예측하는 것이 아니라, 기존 입력과의 차이만 학습하도록 하는 residual connection을 사용한다.

2-4. Layer Normalization

- 층이 깊어질수록, hidden vector 값들이 커졌다 작아졌다 하면서 학습이 불안정하다.
- Layer Normalization은 각 레이어 단위에서 hidden vector 값을 정규화해, 안정적이고 빠른 학습을 돕는다.

2-5. Decoder

- Transformer의 decoder는 여러 개의 decoder 블록들을 쌓아 올려서 만든 구조이다.
- 각 블록은 다음으로 구성된다:
  - Masked Self-Attention (Multi-Head)
    - 미래 단어를 보지 않도록 마스크를 씌운 Multi-Head Self-Attention.
  - Add & Norm (Residual Connection + Layer Normalization)
  - Feed-Forward Network
    - 각 위치 별 비선형 변환을 수행한다.
  - Add & Norm (Residual Connection + Layer Normalization)

→ 언어 모델처럼 단방향 문맥만 활용

2-6. Encoder

- 그에 반해 Transformer의 encoder는 양방향 문맥을 모두 활용할 수 있다.

    - 입력 문장을 의미적 표현으로 변환할 수 있다.
    - 각 단어가양방향 문맥을 모두 반영한 벡터로 인코딩된다.
- Decoder와의 차이점은 Self-Attention에서 masking을 제거한 것 뿐이다.

2-7. Encoder-Decoder

- 기계 번역에서 Seq2Seq 모델을 사용했던 것처럼, Transformer에서도 이해를 위한 encoder와 생성을 위한 decoder로 이뤄진 encoder-decoder 구조를 채택한다.
- decoder는 단순 Self-Attention만 하는 것이 아니라, encoder의 출력 표현을 참조하는 Cross-Attention을 추가하여 입/출력을 연결한다.
- Cross-Attention
    - Cross-Attention을 수행할 때는 Self-Attention과는 다르게
    - Query는 decoder에서, Key와 Value는 encoder에서 가져온다.

2-8. Transformer

- 결과
    - Neural Machine Translation task에서 당시 최고 성능을 달성했을 뿐 아니라, 가장 효율적인 학습으로 비용까지 절감할 수 있었다.
- Transformer와 사전학습(pretraining)
    - Transformer의 등장은 대부분의 최신 모델들이 성능 향상을 위해 사전학습(pretraining)을 결합하도록 했다.
    - 또한, 뛰어난 병렬 처리 능력 덕분에 대규모 사전학습에 적합하여 NLP의 표준 아키텍처로 자리 잡았다.

4. 사전 학습 기반 언어 모델

1. 사전학습이란

- 정의
    - 대규모 데이터 셋을 이용해, 모델이 데이터의 일반적인 특징과 표현을 학습하도록 하는 과정
    - 특히, 언어모델은 인터넷의 방대한 텍스트(웹문서, 책, 뉴스 등)을 활용해 비지도학습 방식으로 학습되어, 일반적인 언어 패턴, 지식, 문맥 이해 능력을 습득한다.
- 사전학습의 관점에서 word embedding vs language model
    - word embedding의 경우 사전학습을 통해 단어의 의미를 학습하지만 한계가 존재
        - downstream task(ex. 텍스트 분류)들에 적용하기엔 학습되는 데이터의 양이 적어, 언어의 풍부한 문맥 정보를 충분히 학습할 수 없다.
        - 연결된 네트워크가 무작위 초기화되어 학습 효율이 낮고, 많인 데이터와 시간이 필요하다.

- Language Model의 경우 모델 전체를 사전학습을 통해 학습하기 때문에, 강력한 NLP 모델을 구축하는 데에 이점이 있다.
  - 언어에 대한 표현학습용으로 적합
  - parameter 초기화의 관점에서 효과적
  - 언어에 대한 확률 분포 학습을 통해 sampling, 생성에 사용할 수 있다.
- Language Model의 pre-train
  - 과거 단어들이 주어졌을 때, 다음 단어의 확률 분포를 모델링하는 방법을 배움으로서, 사전학습을 수행할 수 있다.
  - 인터넷의 대규모 텍스트 코퍼스에서 언어모델링 학습을 수행 후 학습된 network parameter를 저장해, 다양한 downstream task에 활용한다.
- 사전학습에서 fine-tuning으로의 패러다임 변화
  - 사전학습을 통해, 언어 패턴을 잘 학습한 parameter롤 초기화해 NLP application 성능을 향상시킬 수 있다.

2. Encoder 모델

- Encoders
  - 양방향 문맥을 활용하기 때문에, 전통적인 언어 모델과는 차이점이 있다.
  - 따라서, Encoder 모델의 사전 학습을 위해선 입력 단어의 일부를 [MASK] 토큰으로 치환해, 모델이 이 [MASK] 자리에 올 단어를 예측하도록 학습하는 방법
  - 이를 Masked Language Model(MLM)이라 하며, 대표적인 모델이 BERT이다.
    - transformer 기반의 모델로 Masked LM 방법으로 사전학습을 수행
    - 학습방식 1
      - Masked Language Model(MLM)
      - 입력 토큰의 15%를 무작위로 선택
      - [MASK] 토큰 치환(80%), 랜덤 토큰 치환(10%), 그대로 두기 (10%)
      - 모델이 masked된 단어에만 집중하지 않고 다양한 문맥표현을 학습해 더 강건한(robust) 표현을 학습할 수 있도록 했다.
    - 학습방식 2
      - Next Sentence Prediction(NSP)
      - BERT는 입력을 두 개의 연속된 텍스트로 받아, 두번째 문장이 첫번째 문장의 실제 다음 문장인지 여부를 예측하는 Next Sentence Prediction(NSP)을 수행
      - NSP를 통해 문장 간 관계를 학습하여 문맥적 추론 및 문장 수준 이해 task에 도움이 되도록 설계
      - ex.
        - 자연어 추론(Nature Language Inference)
        - Paraphrase detection
        - 질의응답(Quetions Answering, QA)

        - BERT는 이렇게 MLM과 NSP 두 가지 task를 동시에 학습한다.
        - [CLS] 토큰은 NSP task용으로 학습되며, 다른 토큰들은 MLM task용으로 학습된다.
    - BERT의 downstream task
        - Sentence Level
            - 두 문장 관계 분류 task
                - MNLI
                    - 전제(Premise)
                    - 가설(Hypothesis)
                    - 분류 : {Entailment, Contradiction, Neutral}
                - QQP
                    - Q1, Q2, ...을 제공
                    - 분류 : {Duplicate, Not Duplicate}
            - 단일 문장 분류 task
                - SST2
                    - 문장
                    - 분류 : {Positive, Negative}
        - Token Level
            - QA task
                - SQuAD
                    - 질문
                    - 문맥
                    - 정답
            - 개체명 인식(Named Entity Recognition,NER)
                - CoNLL 2003 NER
                    - 문장
                    - 라벨
    - BERT의 결과
        - 다양한 task에 적용가능한 범용성을 보여줬으며, fine-tuning을 통해 여러 NLP 과제에서 SOTA 성능을 이끌어 냄
        - Layer의 수, hidden state의 크기, attention head의 수가 클수록 성능이 향상되는 경향을 보였다.
    - BERT의 한계
        - Encoder 기반 model인 BERT는 주어진입력을 잘 이해하도록 학습되지만, sequence를 생성해야 하는 task에는 적합하지 않다.(ex. 기계번역, 텍스트 생성 등)
        - 생성 task에서는 autoregressive하게 즉 한 번에 한 단어씩 생성해야하는데, 이를 자연스럽게 수행하지 못하기 때문에, 생성 task엔 decoder 기반 모델을 주로 사용한다.

3. Encoder-Decoder 모델

- Encoder와 Decoder의 장점을 모두 결합
- T5(Text-to Text Transfer Transformer)
  - Transformer Encoder-Decoder 구조 기반의 모델
  - 모든 task를 Text-to-Text format으로 변환해 하나의 모델로 학습
- 학습 방법
  - Span Corruption
  - Encoder-Decoder 구조에서는 Encoder가 입력 문장을 모두 보고, 그 정보를 바탕으로 Decoder가 출력을 생성한다.
  - 따라서, 학습을 위해, Span Corruption이라는 과정을 수행
  - 과정
    - 입력문장에서 연속된 token을 무작위로 선택해 제거
    - 제거된 부분을 특수 placeholder token으로 치환
    - decoder는 이 placeholder에 해당하는 원래 span을 복원하도록 학습
    - 이를 통해 언어 모델처럼 훈련을 수행할 수 있다.
    - T5는 NLU(GLUE, SuperGLUE), QA(SQuAD), 요약(CNNDM), 번역(En->De, En->Fr,En->Ro) 등의 task에서 모두 좋은 성능을 보여 범용적으로 활용될 수 있는 모델임을 입증

4. Decoder 모델

- 전형적인 언어 모델 구조, 문장 생성에 유용(미래 단어 참조 불가)
- Finetunning Decoder
  - Transformer의 Decoder는 사전학습 단계에서 다음 단어 예측(Next Token Prediction)을 학습
  - 생성 task에 활용할 때
    - 사전학습 때와 동일하게 다음 단어 예측 방식으로 fine-tuning한다.
    - 따라서, decoder는 대화나 요약 task 등 출력이 sequence인 task에 자연스럽게 적합한다.
  - 분류 task에 활용할 때
    - 마지막 hidden state 위에 새로운 linear layer를 연결해 classifier로 사용
    - 이때, linear는 아예 처음부터 다시 학습해야하며, fine-tuning 시 gradient가 decoder가 전체로 전파된다.
- GPT-1
  - Transformer 기반의 Decoder 모델
  - Autoregressive LM(왼쪽->오른쪽 단어 예측) 방식으로 사전학습되었다.
- GPT-1의 fine-tuning방법(분류 task)
  - GPT-1은 다음 단어 예측이라는 언어모델의 학습 목표를 최대한 유지하면서, fine-tuning을 수행
  - NLI (두 문장을 입력받아 관계를 함의, 모순, 중립 중 하나로 분류) "
    - 전제, 가설 => Entailment

- 입력토큰에 특수한 토큰([START], [DELIM], [EXTRACT])을 붙여 분류 문제를 처리했고, [EXTRACT] 위치의 hidden state에 classifier를 연결해 사용

5. In-Context Learning

- GPT-3
  - 2020년 GPT-2에서 모델의 parameter size를 키워(1750억), 별도의 파인튜닝 없이 컨텍스트 안의 에시만 보고도 새로운 테스크를 수행할 수 있게 되었다.
  - 이런 능력을 In-Context Learning이라고 한다.
  - 모델에 에시와 함께 어떤 테스크를 할지 지정해주면, 모델이 그 패턴을 따라가는 식으로 동작하면서, 완벽하진 않지만 그럴듯하게 task를 수행하는 모습을 보인다.
  - 이러한 능력은 모델의 크기, 즉 parameter size가 커질수록 더 강력하게 나타났으며, zero-shot, one-shot, few-shot 모두에서 일관된 성능 개선이 관찰됨
  - 또한, 모든 parameter size의 모델에서 shot의 수가 많을 수록 task 수행이 향상
  - zero-shot < one-shot < few-shot
  - In-Context Learning의 발견으로 인해, Prompt의 중요성이 대두되었다.
  - 하지만 단순한 few-shot prompting만으로는 여러 스텝을 거쳐야 하는 문제들을 풀기 어려웠는데, 이를 해결하기 위해 Chain-of-Thought(CoT) prompting 방식이 등장
  - CoT prompting은 모델이 문제 해결 과정에서 논리적인 사고단계를 거쳐 최종 답을 도출하도록 유도하는 prompting 기법
  - CoT prompting은 일반적인 prompt 방식보다 훨씬 우수한 성능을 보이며 새로운 SOTA 성과를 달성
  - fine-tuning을 수행한 모델들보다 더 좋은 성능을 보임
- Zero-shot Chain-of-Thought prompting
  - 하지만 기존 CoT Prompting은 few-shot 예시가 필요
  - 예시가 없는 경우, 추론 과정이 나타나지 않아 성능 저하 발생
  - 이를 보완하기 위해, 질문 뒤에 "Let's think step by step"이라는 한 문장을 추가해 모델이 스스로 추론 단계를 생성하도록 유도하는 Zero-Shot prompting 방법 등장

1. CNN 살펴보기

   1-1. CNN vs. FCN

   *FCN(Fully-Connected Layer): 입력을받아서 출력으로 변환하는 신경망의 기본 모듈

   FCN 변환 예시:

   입력: 이미지가 32x32x3인 고차행렬이라고 가정 => 1x3072 1차원

   모델상수: 모델이 학습해야하는 파라미터 => 10 x 3072 weights

   출력: 10x1의 1차원 벡터 => 변환된 데이터를 10개 중 1개로 분류

   *CNN 모델

   기본 구조: 합성곱-> 활성화->풀링->완전연결층

   합성곱 레이어: 입력 이미지를 필터와 연산하여 특징 맵(feature map)을 뽑아내는 모듈

   :1차원 구조로 변환하는 FCN과 달리 3차원 구조를 그대로 보존하면서 연산

> 컨볼루션: 필터(3x5x5)를 이미지(3x32x32)상에서 이동시키면서 내적을 반복수행-> 내적으로 구한 모든 값을 출력 제공
> : 차원을 반드시 주의! 필터는 항상 입력의 깊이/채널 축과 동일한 차원을 가진다

특정위치에서 필터와 동일 사이즈의 이미지 영역 간 내적한 결과 값, 필요 연산은 (3*5*5 =75)차원 내적 곱 +bias 벡터
-> 필터의 이동경로를 따라 모든 연산 값을 모아 활성화 맵을 출력

합성곱 레이어의 예시: 박스필터

각 필터값과 대응되는 입력값의 곱을 모두 합산 -> 입력 가운데 위치의 출력값
합성곱 레이어의 특징
1. 출력 해상도는 입력 해상도 - 필터 해상도 + 1로 도출
2.만약 입출력 해상도를 유지하고 싶다면 -> 출력값 중 정의되지 않은 경우, 0 혹은 가장 가까운 출력값으로 대체한다(패딩)

2. CNN 모델구조

2-1. 중첩
모델 상수(파라미터)를 증가시키면? 어차피 Linear classifier
-> 비선형 블록(Conv[Linear] + ReLU[비선형 변환])과 함께하면 모델링 파워 향상

2-2. 필터
필터 시각화: 학습된 필터 시각화를 통해 각 모델(구조)가 학습한 정보를 이해가능하다.
cf. 선형모델의 필터: 각 카테고리의 템플렛(전형적모습) 역할
FCN의 필터: 다양한 템플렛 제공
CNN의 필터: 계층별로 이미지의 지역적 템플렛(예: 엣지, 색상 등)

2-3. 수용영역(리셉티브 필드)
CNN이 이미지를 처리하면서 한 번에 볼 수 있는 영역의 크기, 네트워크의 시야
일반적으로 네트워크가 깊어질 수록 수용역역도 넓어짐 -> 폭 넓은 맥락 이해가 가능하다
CNN 레이어는 이미지 작은 부분인 "지역 정보"를 추출하는데 유리하게 설계, 이미지 전체(예: 맥락)의미 파악이 필요
따라서 "지역 정보 + 이미지 전체의 맥락" 을 이해/활용하기 위한 설계가 필요하다.
->중첩 (2-1)과 수용영역(2-3)을 이용한다.
고해상도 이미지 처리 시, 많은 레이어를 통과해야한다. 이는 연산,비용,학습 가능성 고려할 때 비실용적임
실제 해법: 입력 사이즈를 줄여 모델에 입력함

2-4. 풀링
효율적 연산 및 위치 변화의 강건성 확보
지역적 특징을 더 압축된 형태로 전달하여, 합성곱 층이 더 넓은 맥락을 이해하는 데 기여한다,
입력 크기가 줄면 CNN의 연산량이 크게 줄어들기때문에 CNN 레이어의 출력을 줄여 연산 효율성을 확보해야한다.

> 위치 변화 강건성: 입력 내 객체의 위치가 다소 변해도 동일한 출력을 제공, 사실상 저해상도 정보에 근거하여 작업 수행하므로 몇 화소 이동은 무시됨

> 맥스풀링: 정해진 커널 사이즈로 이미지를 나누어 각 영역 내 가장 큰 값을 선택하는 연산 (모델 상수 학습 X)
> : 이렇게 하면 연산해야 할 데이터의 양이 줄어들어 학습과 추론 속도가 빨라지고, 메모리 사용량도 절감된다.

2-5. 스트라이드 합성곱
풀링의 한계 계선
효과: 풀링 처럼 입력 해상도를 줄임 (연산 효율과 위치 강건성)

> 일반 합성곱: 필터를 1칸씩 이동하면서 연산 수행(출력값 계산)
> 스트라이트 합성곱: 필터를 스트라이드 값만큼(S칸) 이동한 후 출력 연산
> : 풀링은 학습 상수가 없지만 스트라이드는 커널을 동시에 학습한다.
> :스트라이드 합성곱은 해상도 저하로 인한 정보 손실이 적고, 풀링 + 합성곱을 하나의 레이어로 대체함
> :고도화된 CNN에서는 스트라이드 합성곱을 풀링 대신 활용되는 경향이 있다.

3. CNN 기반 모델 변천사
3-1. AlexNet(2012), 딥러닝 CNN의 본격적인 시작점
5개의 합성곱 계층과 3개의 완전 연결 계층으로 구성된 8계층 CNN모델
3-2. VGGNet(2014),3x3 작은 필터를 깊게 쌓은 단순하면서도 강력한 구조
5개의 합성곱 블록 + 맥스 풀링 구조
:작고 단순한 필터를 깊게 쌓으면 성능이 올라간다.
:단순 설계로 모델 해설에 이상적이며, 특징 추출기, 전이 학습에 강력한 베이스라인을 가짐(파라피터가 많고 연산량 요구가 매우 크다, 거대한 모델
3-3. GoogLeNEt
3-3.ResNet(2015), 잔차 연결을 도입하고 기울기 소실 문제를 해결함
최종 요약: 깊은 모델 학습을 위한 중요한 전진(+100 레이어 학습 가능), 깊은 모델의 강력함을 다시 확인
:제안 당시 모든 벤치마크에서 최고 성능 달성, 지금도 CNN 기반 구조 중 가장 활발하게 활용
합성곱 블록(VGG 유사)과 잔차 블록
: 잔차 블록은 블록입력을 그대로 출력에 더해주는 지름길 연결이 특징. Inception모델에서 차원 축소로 활용된 1x1 합성곱을 적용, 연산 효율 개선

> 등장 배경
> 단순히 레이어를 깊게 하면 오히려 성능이 떨어짐
> -해법: 작은 모델의 성능은 "최소한" 누릴 수 있도록 작은 레이어의 추정값(입력값)을 그대로 후속 레이어에 제공하여
> 최소한 작은 레이어 수준의 성능은 보장
> 잔차블럭:
> 입력 X가 두 경로(변환경로, Shortcut 경로)로 나뉘어 전달됨 -> 마지막에 두 값을 더한다.
>
> > 일반 잔차 블록 vs 보틀넥 잔차 블록, 연산 효율을 위해 보틀넥 잔차구조 도입(더깊은 모델을, 더 적은 연산으로)

> Stem 구조
> 기존 모델의 효율성 레시피를 잘 활용
> (스템 구조를 모델 초기에 도입, 입구 데이터 해상도를 1/4로 줄임, 여러개의 FC레이어를 제거하고 전역 평균 풀링을 사용-> 마지막 1개 FC만 적용)

3-4. MobileNet, 모바일.임베디드 환경 최적화
기존 합성곱은 공간과 채널을 동시에 처리하여 과도한 연산량아 요구됨

해법: 공간과 채널을 두 단계로 분리하여 처리한다.
> 깊이별 합성곱:각 채널별 독립적 3x3 합성곱 수행

> 화소별 합성곱: 1x1 합성곱을 채널 방향으로 적용
> 효과: 연산량 기존 대비 9배 가량 절감, 모델 상수 대폭 감소

딥러닝 학습 안정성 및 모델 구성 요소 요약

1. 학습의 일반적인 문제

- 학습 불안정: 손실 폭발, 학습 멈춤(기울기 소실), 수렴 실패.
- 과적합: 훈련 데이터에 과도하게 맞춰져 검증/실전 성능이 저하되는 현상.
- 느린 수렴: 최적점에 도달하는 데 불필요하게 많은 에폭 소모.
- 좋은 모델 구조만으로는 성능을 보장할 수 없습니다.

2. 모델 구성 및 안정화 전략

2.1. 활성화 함수 (Activation Function)

- 목적: 신경망에 비선형성을 부여하여 복잡한 패턴 학습을 가능하게 합니다.
- Sigmoid (출력 [0, 1]):
  - 문제: 기울기 소실, 편향된 업데이트(출력 항상 양수), 계산 비용 높음.
- Tanh (출력 [-1, 1]):
  - 장점: 0 중심 대칭으로 학습 안정성 높음.
  - 문제: 기울기 소실 문제 여전, 계산 비용 높음.
- ReLU (f(x) = max(0, x)):
  - 장점: 양의 영역에서 기울기 소실 크게 감소, 계산 효율성 및 학습 속도 빠름.
  - 문제: 죽은 뉴런(Dead ReLU) 문제 발생, 0을 기준으로 비대칭.

- Leaky ReLU (f(x) = max(0.01x, x)):
  - 장점: ReLU 장점 수용, 죽은 뉴런 문제 해결.
- ELU:
  - 장점: ReLU 장점 수용, 평균 출력이 0 근처에 위치하여 학습 안정성 높음.
  - 문제: 계산 복잡성 증가, 하이퍼파라미터 알파(α) 설정 필요.

2.2. 데이터 전처리

- 목적: 학습/검증/테스트 데이터 간의 조건을 통일하고 모델 입력에 적합하게 조정합니다.
- 통일 요소: 이미지 해상도, 색상, 밝기, 정규화.
- 정규화 방식:
  - AlexNet: 학습 데이터 전체의 평균 이미지를 입력에서 뺌.
  - VGG: R, G, B 채널별 평균을 입력에서 뺌.
  - ResNet: 채널별 평균을 빼고 채널별 표준편차로 나눔 (표준화).

2.3. 모델 상수 초기화 (가중치 W, 편향 b)

- 0 초기화: 모든 뉴런의 출력 및 기울기가 동일해져 학습이 불가능합니다.
- 랜덤 초기화: 깊은 모델에서는 분산(상수 크기)에 따라 기울기 소실/폭발 위험이 있습니다.
- Xavier 초기화: 가중치 초기 분포의 분산을 입력 차원(Nin)에 맞춰 설계합니다. (입출력 대칭 분포 가정)
- He 초기화: 가중치 초기 분포의 분산을 2 * 입력 차원(2/Nin)에 맞춰 설계합니다. ReLU와 같은 비대칭 활성화 함수에 적합합니다.
- ResNet의 등장 배경: 잔차 연결(Skip Connection)을 사용해 깊은 신경망도 안정적으로 학습할 수 있으며, 초기에는 He 초기화 등을 통해 안정적인 학습 출발을 유도합니다.

2.4. 모델 정규화 (과적합 방지)

- 가중치 감소 (Weight Decay/Regularization): 큰 가중치에 패널티를 부여하여 과적합을 방지합니다.
  - L2 정규화 (Ridge): 가중치 제곱에 패널티. 모든 가중치가 골고루 작아집니다.
  - L1 정규화 (Lasso): 가중치 절대값에 패널티. 중요하지 않은 가중치를 완전히 0으로 만들어 희소한 모델을 만들고 특성 선택 효과가 있습니다.
  - Elastic Net (L1+L2): L1과 L2의 장점을 결합합니다.
- Dropout: 학습 과정에서 일부 뉴런을 확률적으로 끔(0으로 설정)으로써 과적합을 방지하고 일반화 성능을 높입니다 (앙상블 효과).
  - 단점: 학습 시간 증가, 배치 정규화가 적용된 모델에서는 비선호됨.
  - Inverted Dropout: 학습 시 살아남은 뉴런의 출력을 스케일링하여 테스트 시 추가 연산 없이 출력 크기를 유지합니다.

3. 학습 안정성 전략 (학습률 및 종료)

1015 | 2-1. 자연어 처리 기본

1. 워드 임베딩과 순환신경망 기반 모델(RNN & LSTM)

1. 단어를 숫자로 표현

- 워드 임베딩(word embedding)
  - one-hot encoding
    - a. 규칙 기반 혹은 통계적 자연어 처리 연구의 대다수는 단어를 원자적(쪼갤 수 없는) 기호로 취급한다.
    - b. 벡터공간관점에서 보면, 이는 한 원소만 1이고 나머지는 모두 0인 벡터를 의미
    - c. 차원 수(= 단어 사전 크기)는 대략적으로 음성 데이터 2만개, Penn Treebank(PTB) 코퍼스(5만) - big vocab(50만) - Google 1T(1300만)
    - d. 이를 one-hot(원핫) 표현이라고 부르고 단어를 one-hot 표현으로 바꾸는 과정을 one-hot encoding이라고 한다.
  - one-hot encoding의 단점 및 문제점
    - a. 검색 query vector와 대상이 되는 문서 vector들이 서로 직교하게 되어, one-hot vector로는 유사도를 측정할 수 없다.
    - b. 차원의 저주(Curse of Dimenstionality)
      - (a) 고차원의 희소 벡터를 다루기 위해선 많은 메모리가 필요하다.
      - (b) 차원이 커질 수록 데이터가 점점 더 희소(sparse)해져 활용이 어렵다.
    - c. 의미적 정보 부족
      - (a) 비슷한 단어라도 유사한 벡터로 표현되지 않는다.
      - (b) 은행과 금융은 의미적으로 밀접하지만, one-hot encoding에서는 전혀 무관한 vector로 취급된다.(의미적으로 밀접해도 one-hot encoding에서는 각각을 무관하게 취급하거나 의미적으로 전혀 관계가 없어도 유관하게 취급할 수가 있다.)
  - word embedding이란 단어를 단어들 사이의 의미적 관계를 포착할 수 있는 밀집(dense)되고 연속적/분산적(distributed) vector 표현으로 나타내는 방법
    - a. one-hot encoding에선 은행과 금융이 완전히 독립적인(무관한) vector로 표현되었지만,
    - b. word embedding에선 두 단어의 vector가 공간상 서로 가깝게 위치하며, 이를 통해 의미적 유사성을 반영할 수 있다.
    - 대표적인 기법

- (a) Word2Vec
  - 단어의 표현을 간단한 인공 신경망을 이용해서 학습
  - 주요 아이디어는 각 단어와 그 주변 단어들 간의 관계를 예측
  - 주요 알고리즘
    - Skip-grams(SG) 방식
      - 중심 단어를 통해 주변 단어들을 예측하는 방법
      - 단어의 위치(앞 또는 뒤)에 크게 구애 받지 않는다.
      - 윈도우 크기(window size) : 중심 단어 주변 몇개 단어를 문맥으로 볼 것인가?
      - 장점 : 적은 데이터에도 잘 동작하며, 희귀 단어나 구 표현에 강하다.
      - 단점 : 학습 속도가 느리다.
    - Continuous Bag of Words(CBOW) 방식
      - 주변 단어들을 통해 중심 단어를 예측하는 방법
      - 문맥 단어들의 집합으로 중심 단어를 맞춘다.
      - 주변 단어와 함께 묶어서 input data로 활용해서 예측
      - 주변 단어들의 집합이 주어졌을 때, 그 문맥과 함께 등장할 수 있는 단일 단어를 예측
      - 장점 : 학습 속도가 빠르며, 자주 나오는 단어에 강하다.
      - 단점 : 희귀 단어 표현에 약하다.

2. 언어는 순서가 중요함

- 순차적 데이터의 특징
  - 자연에 수많은 순차적(Sequential Data)가 존재
  - 데이터가 입력되는 순서와 이 순서를 통해 입력되는 데이터들 사이의 관계가 중요한 데이터
  - 대표적으로 오디어, 텍스트, 비디오 데이터 등이 있음.
  - 1) 순서가 중요하다. 데이터의 순서가 바뀌면 의미가 달라진다.
  - 2) 장기의존성(Long-term dependency) : 멀리 떨어진 과거의 정보가 현재/미래에 영향을 준다.
  - 3) 가변길이(Variable length) : 순차 데이터는 길이가 일정하지 않고, 단어 수도 제각각이다.
    - 처리 방법
      - 일반적인 모델인 선형회귀, MLP 등으로는 처리가 불가능하다.
      - 따라서, Sequential Models이 필요하다.(RNN, LSTM, Transformer 등)

3. RNN

- 문맥을 기억하는 신경망
  - 전통적인 인공신경망은 MLP, CNN 등이며, 이는 고정된 길이의 입력을 받기 때문에, 가변 길이의 데이터를 처리하기에 적합하지 않는다.

- RNN은 가변 길이의 입력을 입력 받을 수 있고, 이전 입력을 기억할 수 있기 때문에, 순차적 데이터 처리에 적합한 architecture이다.
- 대표적으로, one to one, many to one, one to many, many to many 등이 있다.
  - Architecure 설명
    - 전통적인 인공신경망은 MLP, CNN과 달리, RNN은 이전 시점의 정보를 담는 hidden state를 가지고 있다.
    - 따라서, 입력 시퀀스 벡터 x를 처리할 때, 각 시점마다 recurrence 수식을 적용하며 hidden state를 업데이트한다.

$$h_t = f_W(h_{t-1}, x_t)$$

$$h_t = f_W(h_{t-1}, x_t)$$

$$h_t : 새로운 hidden state$$

$$f_W : 처리함수(파라미터 W)$$

$$h_{t-1} : 이전 hidden stae$$

$$x_t : 입력 vector(Timestep)$$

- RNN은 각 시점마다 동일한 가중치(weight)를 사용하기 때문에, 깊은 feedward 신경망보

$$y_t = W_{hy}h_t$$

$$h_t = tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

- RNN의 특징
  - RNN은 한 번에 하나의 요소를 처리하고, 정보를 앞으로 전달한다.
  - RNN은 각 층이 하나의 시점을 나타내는 깊은 신경망처럼 보인다.
  - RNN은 hidden state를 유지하면서 가변 길이 데이터를 처리할 수 있다.
  - RNN의 출력은 과거 입력에 영향을 받는다는 점에서, feedward 신경망과 다르다.
- RNN의 단점
  - 기울기 소실(vanishing gradient) 문제
  - vanishing gradient 문제란?
    - 딥러닝에서 역전파 시 앞쪽 측의 기울기가 0에 가까워져서 장기 의존성 학습이 어려

    - 역전파 과정에서 작은 값들이 계속 곱해진다.
    - 과거 시점에서 온 오차 신호는 갈수록 더 작은 기울기를 갖게 된다.
    - 결국 파라미터들이 장기 의존성은 학습하지 못하고, 단기 의존성만 포착하게 된다.

4. LSTM

- LSTMs란 기울기 소실문제를 해결하기 위해, 1997년에 제안된 RNN의 한 종류
  - LSTMs의 특징
    - 시점 t에서 RNN은 길이가 n인 vector hidden state h_t와 Cell state C_t를 가진다.
    - Hidden state는 short-term information을 저장한다.
    - Cell state는 long-term information을 저장한다.
  - LSTMs cell state에서 정보를 읽고(read), 지우고(erase), 기록(write)할 수 있다.
  - LSTMs는 3가지 게이트를 통해 어떤 정보를 지우고 쓰고 읽을지를 결정한다.
    - Forget gate
      - 이전 cell state에서 무엇을 버리고 무엇을 유지할지 결정한다.

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

- Input gate
  - 새 정보 중 얼마나 cell state에 쓸지 결정한다.

$$i_t = sigma(W_i cdot[h_t, x_t] + b_i)$$

$$tildeC_t = tanh(W_C cdot[h_{t-1}, x_t] + b_C)$$

- New Cell Content
  - New Cell Content : Cell에 기록될 새로운 후보

$$C_t = f_t * C_{t-1} + i + t * tildeC_t$$

- Output gate
  - cell state 중 얼마나 hidden state로 내보낼지 결정한다.

$$o_t = sigma(W_o[h_{t-1}, x_t] + b_o)$$

$$h_t = o_t * tanh(C_t)$$

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

2. 자연어 생성 모델 (Seq2Seq, Attention)

1. 언어 모델이란?

1-1. 개념

- 인간의 두뇌가 자연어를 생성하는 능력을 모방한 모델
- 단어 시퀀즈 전체에 확률을 부여하여 문장의 자연스러움을 측정
- 한 문장의 확률은 각 단어의 조건부 확률들의 곱으로 표현할 수 있음

→ 언어 모델은 **문장 내 단어의 등장 확률 분초를 모델링하는 것**을 의미

→ 즉, 주어진 단어들의 연속이 나올 확률을 계산하여 **다음 단어를 예측**하거나

→ 문장의 자연스러움(유창성**)을 평가**하는 모델

1-2. 일상 속의 언어모델 ex. 텍스트 자동완성 기능

1-3. 대표적인 언어모델 : N-gram 언어모델

- n-gram : 연속된 n개의 단어 묶음
    - unigram: `"The"` , `"students"` , `"opened"` , `"their"`
    - bigram: `"The students"` , `"students opened"` , `"opened their"`
    - trigram: `"The students opened"` , `"students opened their"`
    - 4-gram: `"The students opened their"`
- 언어모델의 목표는 다음 단어의 조건부 확률을 추정하는 것
- 다른 예시 : 4-gram 언어 모델
- 예시 데이터:
    - `"students opened their"` : 1000회 등장
    - `"students opened their books"` : 400회 등장
      → P(books ∣ students opened their)=0.4P(\text{books} | \text{students opened their}) = 0.4P(books ∣ students opened their)=0.4
    - `"students opened their exams"` : 100회 등장
      → P(exams ∣ students opened their)=0.1P(\text{exams} | \text{students opened their}) = 0.1P(exams ∣ students opened their)=0.1

1-4. 언어모델 사용 예시 : Statistical Machine Translation, SMT

- 1990년대~2010년대 초반까지는 번역 문제를 **확률적 모델링**으로 접근
- 예시 : 한국어 → 영어
  - 목표: 주어진 한국어 문장 x 에 대해, 가장 적절한 영어 문장 y 를 찾는 것

$$argmax_y P(y \mid x)$$

  - **Bayes 정리를 이용한 분해**P(y ∣ x)=P(x)P(x ∣ y)P(y)⇒argymaxP(x ∣ y)P(y)

$$= argmax_y P(x \mid y)P(y)$$

- 두 개의 구성요소로 분리:
  - **번역모델 P(x ∣ y)**
    - 단어와 구가 어떻게 번역되어야 하는지 모델링
    - (한국어, 영어) 말뭉치로 학습
      → 영어 문장 yyy가 주어졌을 때 한국어 문장 xxx가 생성될 확률
  - **언어모델 P(y)**
    - 유창한 영어 문장을 쓰는 방법을 모델링
    - 영어 데이터로 학습
- 한계점 : SMT의 문제
  - **구조적 복잡성**: 문장 구조를 반영하기 어렵다.
  - **수작업 의존도**: 문장 정렬, 단어 정렬, 구문 규칙 등의 수작업 필요.
  - **언어별 자원 구축 부담**: 각 언어쌍마다 별도의 말뭉치와 규칙이 필요.
    → 결과적으로 유지·확장성이 떨어짐
- 발전 방향 : **Neural Machine Translation (NMT)**
  - 위 한계를 극복하기 위해 **신경망 기반 번역모델(NMT)**로 전환됨.
  - NMT는 **하나의 신경망이 전체 번역 과정을 자동으로 학습**하여 번역모델과 언어모델을 통합적으로 처리함

2. Seq2Seq

2-1. Neural Machine Translation (NMT)

- 개념
  - Neural Machine Translation 이란 인공 신경망을 이용해 기계 번역을 수행하는 방법
  - 이 때 사용되는 신경망 구조를 sequence-to-sequence (Seq2Seq)이라 하며, 두 개의 RNNs으로 이루어짐

        - 2014년 Google의 "Sequence to Sequence Learning with Neural Networks"라는 논문에서 처음 소개
- Translation이 어려운 이유
    - 번역 문제는 입력과 출력의 길이가 다를 수 있음
        - 영어: the black cat drank milk (5개의 단어)
        - 프랑스어: le chat noir a bu du lait (7개의 단어)

      → 따라서 NMT에서는 길이가 다른 시퀀스 간의 매핑을 처리할 수 있어야함
- Seq2Seq의 아이디어
    - 2개의 LSTM을 이용하자
        - 한 LSTM은 입력 시퀀스를 한 타임스텝씩 읽어 고정된 차원의 큰 벡터 표현을 얻기 (Encoder)
        - 다른 LSTM은 앞에서 얻은 벡터로부터 출력 시퀀스를 생성하기 (Decoder)

2-2. Seq2Seq Architecture

- Seq2Seq의 구성
    - Encoder와 Decoder로 이루어진다.
        - Encoder는 입력 문장에 담긴 정보를 인코딩
        - Decoder는 인코딩된 정보를 조건으로 하여 타겟 문장(출력)을 생성
- Seq2Seq의 다양한 적용
    - Seq2Seq 구조는 기계번역 외에도 다양한 태스크에 적용 가능
        - 요약 : 긴 길이의 문서를 읽고, 짧은 길이의 문장으로 요약된 텍스트를 출력하는 태스크
        - 대화: 사용자의 발화를 기반으로, 맥락에 맞는 대답(출력 텍스트)을 생성하는 태스크
        - 코드 생성: 자연어로 작성된 설명 혹은 명령어를 입력 받아, 그에 대응하는 프로그래밍 코드 혹은 쿼리를 출력하는 태스크
- Seq2Seq의 학습 수행 (Teacher Forcing)
    - Seq2Seq 모델은 인코더와 디코더가 하나의 통합 네트워크로 연결되어 있음
    - 디코더에서 발생한 오차는 역전파 과정을 통해 입력을 처리한 인코더까지 전달되어 전체 네트워크가 End-to-End로 동시에 최적화된다.
    - 학습 초반에는 모델의 예측 능력이 떨어지기 때문에 학습이 불안정할 수 있음
    - Teacher Forcing이란?
        - 모델이 스스로 예측한 단어 대신 정답 단어를 디코더 입력으로 강제로 넣어줌으로써 훨씬 안정적이고 빠르게 학습을 수행하는 방법이다.
- Seq2Seq의 토큰 출력 방법 (Greedy Inference)
    - 토큰을 출력하는 방법 중 하나로, 각 단계에서 가장 확률이 높은 단어를 선택
    - 한계
        - 되돌리기가 불가능
        - 예시 : le _____ → le chien ____ → …
          chein이 오답이어도 돌아갈 방법이 없지
    - Beam Search

a. 매 단계마다 k개의 가장 유망한 후보 유지
b. 후보가 에 도달하면, 완성된 문장으로 리스트 추가
c. 문장이 충분히 모이면 탐색 종료
d. 각 후보들의 점수를 로그 확률의 합으로 구해 최종 선택

3. Attention

3-1. Seq2Seq의 한계 : the bottleneck problem

- Bottleneck problem 개념
  - 인코더는 입력 문장 전체를 하나의 벡터로 요약하는데, 마지막 hidden state에 문장의 모든 의미 정보가 담긴다
  - 고정 길이 벡터 하나에 모든 문장의 의미를 압축하다 보니 정보 손실이 생길 수 있는데, 이를 bottleneck problem이라고 한다.
- Attention의 인사이트
  - Attention은 디코더가 단어를 생성할 때, 인코더 전체 hidden state 중 필요한 부분을 직접 참조할 수 있도록 한다.
  - 즉, 매 타임스텝마다 "어떤 단어/구절에 집중할지"를 가중치로 계산해, bottleneck 문제를 완화했다.

3-2. Attention의 효과

- Attention mechanism은 많은 장점이 존재한다.
  i. NMT 성능 향상
    - 디코더가 소스 문장 전체가 아닌, 필요한 부분에만 집중할 수 있기 때문이다.
  ii. Bottleneck Problem 해결
    - 디코더가 인코더의 모든 hidden states에 직접 접근할 수 있다.
  iii. Vanishing Gradient Problem 완화
    - Attention은 멀리 떨어진 단어도 직접 연결할 수 있게 해준다.
- Attention의 효과: 해석 가능성(Interpretability)
  - Attention 분포를 보면, decoder가 어떤 단어를 생성할 때, 입력 문장의 어느 부분에 집중했는지 확인할 수 있다.
  - 즉, 모델이 내부적으로 참고한 근거를 사람이 파악할 수 있음
    → 모델의 의사결정 과정을 해석할 수 있는 단서
- Attention의 효과: 정렬(Alignment)
  - 기계번역에서는 전통적으로 단어 alignment 모델을 따로 학습해야 했다.
  - 하지만, attention은 통해 decoder가 필요한 입력 단어에 자동으로 집중하기 때문에, 단어와 단어 간의 매핑 관계를 자연스럽게 학습한다.
- Attention : Query와 Values

- Seq2Seq에서 attention을 사용할 때, 각 decoder의 hidden state와 모든 encoder의 hidden states 간의 관계를 Query와 Values의 관계로 볼 수 있다.
- 이 관점에서 Attention 과정을 정리해보면:
  a. Query와 Values 사이 유사도 점수(score) 계산 (예: dot-product, multiplication, additive 등)
  b. Softmax를 통해 확률 분포(attention distribution) 얻기
  c. 분포를 이용해 values를 가중합 → context vector (attention output)

3. Transformer

1. Self-Attention

1-1. RNN

- RNN의 필요성
  - RNN은 정보를 hidden state로 전달하며 순차적 의존성을 형성한다.
  - 그에 반해 Attention은 필요한 순간, 입력 전체에서 직접 정보를 전달한다.
    → RNN이 하던 "정보 전달"을 Attention이 더 효율적으로 수행할 수 있다면, 굳이 recurrence가 필요할까?

1-2. RNN의 한계점

1. 장기 의존성
   - RNN은 왼쪽에서 오른쪽으로 순차 전개되어, 먼 단어 쌍이 상호작용하려면 시퀀스 길이 만큼의 단계를 거쳐야 한다.
   - 따라서, 길어진 단계만큼 기울기 소실 혹은 폭발 문제가 발생해 장기 의존성을 학습하기 어렵다.
   - 또한, RNN은 입력된 선형 순서를 강하게 반영하기 때문에, 언어의 비선형적 의존성을 잘 잡아내지 못한다.
     - 예시: 그 책은 오랫동안 방치되어 먼지가 많이 쌓인 탓에… 매우 더러웠다.
2. 병렬화
   - Forward와 Backward pass 모두 시퀀스 길이 만큼의 단계가 필요하다.
   - 이처럼, 순차적인 연산이 진행되기 때문에, 병렬화가 불가능하다.
   - 이는, 병렬 연산에 강한 GPU 활용을 어렵게 만들며, 대규모 데이터 학습에 비효율적이다.

1-3. Attention은 어떨까 ?

- Attention은 각 단어의 표현을 query로 두고, value 집합으로부터 필요한 정보를 직접 불러와 결합한다.
- 이러한 Attention 메커니즘을 encoder-decoder 간이 아닌, 한 문장 내부에서 적용한다면 어떨까?
  → Self-Attention

1-4. Self-Attention의 장점은?

1. 순차적으로 처리해야 하는 연산 수가 시퀀스 길이에 따라 증가하지 않는다.
2. 최대 상호작용 거리 =O(1) → 모든 단어가 각 층에서 직접 상호작용한다.

1-5. Seq2Seq에서의 Attention

- Attention(입력 문장 내의 단어들 | 출력 문장 내의 각 단어 "w")
- 배경
  - 기존 Seq2Seq 모델은 입력 문장을 **하나의 고정된 벡터(Context Vector)** 로 압축한 뒤, 이를 디코더가 이용해 출력 문장을 생성
  - 하지만 문장이 길어질수록 이 벡터 하나에 모든 정보를 담기 어려워졌고, **Attention**이 도입되면서 이 문제를 해결하게
- 핵심 아이디어
  - 출력 문장(Decoder)의 각 단어를 생성할 때, 입력 문장(Encoder)의 **모든 단어들 중에서 "관련 있는 단어"에 더 집중(attend)** 하도록 하는 메커니즘.
  - **입력 문장(Encoder의 hidden states)** → Key, Value
  - **출력 문장(Decoder의 현재 hidden state)** → Query
- 예시
  - 입력 문장: `"I love cats"`
  - 출력 문장: `"나는 고양이를 좋아해"`
    - Decoder가 `"고양이를"` 을 생성할 때
      → Encoder의 단어들 중 `"cats"` 에 높은 가중치를 부여
      → `"I"` , `"love"` 에는 낮은 가중치
- 정리
  - Seq2Seq Attention은 **"출력 단어를 생성할 때 입력 문장의 어떤 단어에 집중할지"**를 결정하는 메커니즘이다.

1-6. Self-Attention

- Attention(문장 내의 다른 단어들 | 문장 내의 각 단어 "w")
- 배경
  - Self-Attention은 Seq2Seq처럼 "입력 → 출력" 관계가 아니라, **하나의 문장 내부 단어들끼리 서로 어떤 연관이 있는지**를 학습하는 메커니즘
- 핵심 아이디어
  - 하나의 단어를 표현할 때, 그 단어가 같은 문장 안의 다른 단어들과 어떤 관계를 맺는지를 반영
- 예시
  - 문장: `"The animal didn't cross the street because it was too tired."`
  - 단어 `"it"` 이 나올 때,
    → Self-Attention은 `"animal"` 과 `"it"` 이 같은 대상을 가리킨다는 관계를 학습한다.

즉, `"it"` 을 표현할 때 `"animal"` 의 정보가 중요하게 반영된다.

- 정리
  - Self-Attention은 **“문장 내에서 각 단어가 다른 단어들과의 관계를 고려하여 표현을 만드는 메커니즘”**이다.
  - 모든 단어가 서로를 참조하면서 문맥적 의미를 포착한다.

1-7. Self-Attention - Key, Query, Value

- Attention에서 각 단어 "i"를 표현하는 Query와 Value 벡터가 있었듯이, Self-Attention에서는 각 단어 "i"를 표현하는 Query, Key, Value 벡터가 존재한다.
  i. 각 단어를 Query, Key, Value 벡터로 변환한다.
    - Query 벡터
      - 단어 i가 다른 단어로부터 어떤 정보를 찾을지를 정의하는 벡터
    - Key 벡터
      - 단어 i가 자신이 가진 정보의 특성을 표현하는 벡터
    - Value 벡터
      - 실제로 참조되는 정보 내용을 담고 있는 벡터
  ii. Query, Keys 간의 유사도를 계산해, softmax로 확률분포를 구한다.
    - 유사도 계산식

$$e_{ij} = q_i^T k_j$$

    - Softmax로 확률화

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{j'} \exp(e_{ij'})}$$

  iii. 각 단어의 출력을 Values의 가중합으로 계산

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

1-8. Self-Attention의 한계

- 이렇듯, Self-Attention은 단어 간 관계를 효율적으로 잡아내는 강력한 메커니즘이지만, 한계가 존재한다.
- 한계
  i. 순서 정보 부재 → 단어 간 유사도만 계산하기 때문에, 단어의 순서를 고려하지 않는다.
  ii. 비선형성 부족 → Attention 계산은 본질적으로 가중 평균 연산이라는 선형 결합에 불과하기 때문에, 복잡한 패턴이나 깊은 표현력을 담기 어렵다.
  iii. 미래 참조 문제 → 언어 모델은 시퀀스를 왼쪽에서 오른쪽으로 생성해야 하지만, Self-Attention은 모든 단어를 동시에 보기 때문에, 아직 생성되지 않아야 할 미래 단어를 참조한다.

1-9. Self-Attention의 한계 해결

1. Positional Encoding
    - 순서 정보 부재 문제를 해결하기 위해, Positional Encoding이라는 기법을 사용한다.
    - 각 단어 위치 i를 나타내는 위치 벡터를 정의해, 단어 임베딩 값에 더해 최종 입력으로 사용한다.
    - 두 가지 방법이 존재
        - Sinusoidal Position Encoding
            - 서로 다른 주기의 사인/코사인 함수를 합성해 위치 벡터를 만드는 방법
        - Learned Absolute Position Embedding
            - 위치 벡터를 모두 학습 파라미터로 설정해 학습 과정에서 데이터에 맞춰 최적화하는 방법
2. Feed-Forward Network 추가하기
    - Self-Attention 연산은 비선형 변환이 없어, 복잡한 패턴 학습에 한계가 존재한다.
    - 따라서, 각 단어 출력 벡터에 Feed-Forward Network (Fully Connected + ReLU 등)을 추가해 Self-Attention이 만든 표현을 깊고 비선형적인 표현으로 확장한다.
3. Masked Self-Attention
    - 단어를 생성할 때는 한 단어씩 순차적으로 미래 단어를 예측해야 하지만, Self-Attention은 기본적으로 모든 단어(미래 포함)를 동시에 참조한다.
    - 따라서, Attention Score를 계산할 때, 미래 단어에 해당하는 항목을 -∞로 설정해, 계산을 수행할 때 반영되지 않도록 한다.

1-10. Self - Attention 정리

- Self-Attention은 문장 내 모든 단어가 서로 직접 상호작용하여,
    i. 장거리 의존성을 효율적으로 포착하고
    ii. 병렬 처리를 가능하게 하는 메커니즘이다.
- 한계 해결 방법
    i. 순서 정보 부재 → Positional Encoding
    ii. 비선형성 부족 → Feed-Forward Network 추가
    iii. 미래 참조 문제 → Masked Self-Attention

epeat for number
f encoder blocks

2. Transformer

2-1. Transformer

- Transformer는 encoder-decoder 구조로 설계된 신경망 모델이다.
    - encoder: 입력 문장을 받아 의미적 표현으로 변환을 수행한다.

- decoder: 인코더의 표현과 지금까지 생성한 단어들을 입력 받아, 다음 단어를 예측한다.
  → 이 중 decoder가 언어 모델과 같은 방식으로 동작한다.

2-2. Multi-Headed Attention

- 문장에서 같은 단어라도 여러 이유(문법적 관계, 의미적 맥락 등)로 다른 단어에 주목할 수 있다.
- 하지만, 단일 Self-Attention Head로는 한 가지 관점에서의 단어 간 관계 밖에 파악할 수 없다.
  → 따라서, 여러 Attention Head를 두어 다양한 관점에서 동시에 정보를 파악한다.
  - Attention Head 1 - 단어의 문맥적 관계에 Attention → 문법적 의존성 관계 단어들에 주목
  - Attention Head 2 - 단어의 시제에 Attention → 개체에 주목
  - Attention Head 3 - 명사에 Attention

2-3. Scaled Dot Product

- Query와 Key의 차원이 커질수록, 두 벡터의 내적 값도 자연스럽게 커진다.
- 이 값이 너무 크면 softmax 함수가 출력이 지나치게 뾰족해져, 미세한 변화에도 큰 차이가 발생하고 gradient vanishing 문제가 생길 수 있다.

$$output_l = softmax(XQ_lK_l^TX^T) * XV_l$$

- 따라서, 내적 값을 그대로 사용하지 않고, 나눠주어 스케일을 조정한다.
- 이렇게 하면 감이 안정적으로 분포되어 학습이 훨씬 더 빠르고 안정적으로 진행되다.

2-3. Residual Connection

- 깊은 신경망은 층이 깊어질수록 학습이 어려워진다. (Gradient vanishing / exploding)
- 따라서, 단순히 Layer의 출력만 사용한다면 정보가 소실되어, layer가 전체를 예측하는 것이 아니라, 기존 입력과의 차이만 학습하도록 하는 residual connection을 사용한다.

2-4. Layer Normalization

- 층이 깊어질수록, hidden vector 값들이 커졌다 작아졌다 하면서 학습이 불안정하다.
- Layer Normalization은 각 레이어 단위에서 hidden vector 값을 정규화해, 안정적이고 빠른 학습을 돕는다.

2-5. Decoder

- Transformer의 decoder는 여러 개의 decoder 블록들을 쌓아 올려서 만든 구조이다.
- 각 블록은 다음으로 구성된다:
  - Masked Self-Attention (Multi-Head)
    - 미래 단어를 보지 않도록 마스크를 씌운 Multi-Head Self-Attention.
  - Add & Norm (Residual Connection + Layer Normalization)

    - Feed-Forward Network
        - 각 위치 별 비선형 변환을 수행한다.
    - Add & Norm (Residual Connection + Layer Normalization)

→ 언어 모델처럼 단방향 문맥만 활용

2-6. Encoder

- 그에 반해 Transformer의 encoder는 양방향 문맥을 모두 활용할 수 있다.
    - 입력 문장을 의미적 표현으로 변환할 수 있다.
    - 각 단어가양방향 문맥을 모두 반영한 벡터로 인코딩된다.
- Decoder와의 차이점은 Self-Attention에서 masking을 제거한 것 뿐이다.

2-7. Encoder-Decoder

- 기계 번역에서 Seq2Seq 모델을 사용했던 것처럼, Transformer에서도 이해를 위한 encoder와 생성을 위한 decoder로 이뤄진 encoder-decoder 구조를 채택한다.
- decoder는 단순 Self-Attention만 하는 것이 아니라, encoder의 출력 표현을 참조하는 Cross-Attention을 추가하여 입/출력을 연결한다.
- Cross-Attention
    - Cross-Attention을 수행할 때는 Self-Attention과는 다르게
    - Query는 decoder에서, Key와 Value는 encoder에서 가져온다.

2-8. Transformer

- 결과
    - Neural Machine Translation task에서 당시 최고 성능을 달성했을 뿐 아니라, 가장 효율적인 학습으로 비용까지 절감할 수 있었다.
- Transformer와 사전학습(pretraining)
    - Transformer의 등장은 대부분의 최신 모델들이 성능 향상을 위해 사전학습(pretraining)을 결합하도록 했다.
    - 또한, 뛰어난 병렬 처리 능력 덕분에 대규모 사전학습에 적합하여 NLP의 표준 아키텍처로 자리 잡았다.

4. 사전 학습 기반 언어 모델

1. 사전학습이란

- 정의
    - 대규모 데이터 셋을 이용해, 모델이 데이터의 일반적인 특징과 표현을 학습하도록 하는 과정

    - 특히, 언어모델은 인터넷의 방대한 텍스트(웹문서, 책, 뉴스 등)을 활용해 비지도학습 방식으로 학습되어, 일반적인 언어 패턴, 지식, 문맥 이해 능력을 습득한다.
- 사전학습의 관점에서 word embedding vs language model
    - word embedding의 경우 사전학습을 통해 단어의 의미를 학습하지만 한계가 존재
        - downstream task(ex. 텍스트 분류)들에 적용하기엔 학습되는 데이터의 양이 적어, 언어의 풍부한 문맥 정보를 충분히 학습할 수 없다.
        - 연결된 네트워크가 무작위 초기회되어 학습 효율이 낮고, 많인 데이터와 시간이 필요하다.
    - Language Model의 경우 모델 전체를 사전학습을 통해 학습하기 때문에, 강력한 NLP 모델을 구축하는 데에 이점이 있다.
        - 언어에 대한 표현학습용으로 적합
        - parameter 초기화의 관점에서 효과적
        - 언어에 대한 확률 분포 학습을 통해 sampling, 생성에 사용할 수 있다.
    - Language Model의 pre-train
        - 과거 단어들이 주어졌을 때, 다음 단어의 확률 분포를 모델링하는 방법을 배움으로서, 사전학습을 수행할 수 있다.
        - 인터넷의 대규모 텍스트 코퍼스에서 언어모델링 학습을 수행 후 학습된 network parameter를 저장해, 다양한 downstream task에 활용한다.
    - 사전학습에서 fine-tuning으로의 패러다임 변화
        - 사전학습을 통해, 언어 패턴을 잘 학습한 parameter롤 초기화해 NLP application 성능을 향상시킬 수 있다.

2. Encoder 모델

- Encoders
    - 양방향 문맥을 활용하기 때문에, 전통적인 언어 모델과는 차이점이 있다.
    - 따라서, Encoder 모델의 사전 학습을 위해선 입력 단어의 일부를 [MASK] 토큰으로 치환해, 모델이 이 [MASK] 자리에 올 단어를 예측하도록 학습하는 방법
    - 이를 Masked Language Model(MLM)이라 하며, 대표적인 모델이 BERT이다.
        - transformer 기반의 모델로 Masked LM 방법으로 사전학습을 수행
        - 학습방식 1
            - Masked Language Model(MLM)
            - 입력 토큰의 15%를 무작위로 선택
            - [MASK] 토큰 치환(80%), 랜덤 토큰 치환(10%), 그대로 두기 (10%)
            - 모델이 masked된 단어에만 집중하지 않고 다양한 문맥표현을 학습해 더 강건한(robust) 표현을 학습할 수 있도록 했다.
        - 학습방식 2
            - Next Sentence Prediction(NSP)

        - BERT는 입력을 두 개의 연속된 텍스트로 받아, 두번째 문장이 첫번째 문장의 실제 다음 문장인지 여부를 예측하는 Next Sentence Prediction(NSP)을 수행
        - NSP를 통해 문장 간 관계를 학습하여 문맥적 추론 및 문장 수준 이해 task에 도움이 되도록 설계
        - ex.
            - 자연어 추론(Nature Language Inference)
            - Paraphrase detection
            - 질의응답(Quetions Answering, QA)
    - BERT는 이렇게 MLM과 NSP 두 가지 task를 동시에 학습한다.
    - [CLS] 토큰은 NSP task용으로 학습되며, 다른 토큰들은 MLM task용으로 학습된다.
- BERT의 downstream task
    - Sentence Level
        - 두 문장 관계 분류 task
            - MNLI
                - 전제(Premise)
                - 가설(Hypothesis)
                - 분류 : {Entailment, Contradiction, Neutral}
            - QQP
                - Q1, Q2, ...을 제공
                - 분류 : {Duplicate, Not Duplicate}
        - 단일 문장 분류 task
            - SST2
                - 문장
                - 분류 : {Positive, Negative}
    - Token Level
        - QA task
            - SQuAD
                - 질문
                - 문맥
                - 정답
        - 개체명 인식(Named Entity Recognition,NER)
            - CoNLL 2003 NER
                - 문장
                - 라벨
- BERT의 결과
    - 다양한 task에 적용가능한 범용성을 보여줬으며, fine-tuning을 통해 여러 NLP 과제에서 SOTA 성능을 이끌어 냄

        - Layer의 수, hidden state의 크기, attention head의 수가 클수록 성능이 향상되는 경향을 보였다.
    - BERT의 한계
        - Encoder 기반 model인 BERT는 주어진입력을 잘 이해하도록 학습되지만, sequence를 생성해야 하는 task에는 적합하지 않다.(ex. 기계번역, 텍스트 생성 등)
        - 생성 task에서는 autoregressive하게 즉 한 번에 한 단어씩 생성해야하는데, 이를 자연스럽게 수행하지 못하기 때문에, 생성 task엔 decoder 기반 모델을 주로 사용한다.

3. Encoder-Decoder 모델

- Encoder와 Decoder의 장점을 모두 결합
- T5(Text-to Text Transfer Transformer)
    - Transformer Encoder-Decoder 구조 기반의 모델
    - 모든 task를 Text-to-Text format으로 변환해 하나의 모델로 학습
- 학습 방법
    - Span Corruption
    - Encoder-Decoder 구조에서는 Encoder가 입력 문장을 모두 보고, 그 정보를 바탕으로 Decoder가 출력을 생성한다.
    - 따라서, 학습을 위해, Span Corruption이라는 과정을 수행
    - 과정
        - 입력문장에서 연속된 token을 무작위로 선택해 제거
        - 제거된 부분을 특수 placeholder token으로 치환
        - decoder는 이 placeholder에 해당하는 원래 span을 복원하도록 학습
        - 이를 통해 언어 모델처럼 훈련을 수행할 수 있다.
        - T5는 NLU(GLUE, SuperGLUE), QA(SQuAD), 요약(CNNDM), 번역(En->De, En->Fr,En->Ro) 등의 task에서 모두 좋은 성능을 보여 범용적으로 활용될 수 있는 모델임을 입증

4. Decoder 모델

- 전형적인 언어 모델 구조, 문장 생성에 유용(미래 단어 참조 불가)
- Finetunning Decoder
    - Transformer의 Decoder는 사전학습 단계에서 다음 단어 예측(Next Token Prediction)을 학습
    - 생성 task에 활용할 때
        - 사전학습 때와 동일하게 다음 단어 예측 방식으로 fine-tuning한다.
        - 따라서, decoder는 대화나 요약 task 등 출력이 sequence인 task에 자연스럽게 적합한다.
    - 분류 task에 활용할 때
        - 마지막 hidden state 위에 새로운 linear layer를 연결해 classifier로 사용

        - 이때, linear는 아예 처음부터 다시 학습해야하며, fine-tuning 시 gradient가 decoder가 전체로 전파된다.
- GPT-1
    - Transformer 기반의 Decoder 모델
    - Autoregressive LM(왼쪽->오른쪽 단어 예측) 방식으로 사전학습되었다.
- GPT-1의 fine-tuning방법(분류 task)
    - GPT-1은 다음 단어 예측이라는 언어모델의 학습 목표를 최대한 유지하면서, fine-tuning을 수행
    - NLI (두 문장을 입력받아 관계를 함의, 모순, 중립 중 하나로 분류) "
        - 전제, 가설 => Entailment
- 입력토큰에 특수한 토큰([START], [DELIM], [EXTRACT])을 붙여 분류 문제를 처리했고, [EXTRACT] 위치의 hidden state에 classifier를 연결해 사용

5. In-Context Learning

- GPT-3
    - 2020년 GPT-2에서 모델의 parameter size를 키워(1750억), 별도의 파인튜닝 없이 컨텍스트 안의 에시만 보고도 새로운 테스크를 수행할 수 있게 되었다.
    - 이런 능력을 In-Context Learning이라고 한다.
    - 모델에 에시와 함께 어떤 테스크를 할지 지정해주면, 모델이 그 패턴을 따라가는 식으로 동작하면서, 완벽하진 않지만 그럴듯하게 task를 수행하는 모습을 보인다.
    - 이러한 능력은 모델의 크기, 즉 parameter size가 커질수록 더 강력하게 나타났으며, zero-shot, one-shot, few-shot 모두에서 일관된 성능 개선이 관찰됨
    - 또한, 모든 parameter size의 모델에서 shot의 수가 많을 수록 task 수행이 향상
    - zero-shot < one-shot < few-shot
    - In-Context Learning의 발견으로 인해, Prompt의 중요성이 대두되었다.
    - 하지만 단순한 few-shot prompting만으로는 여러 스텝을 거쳐야 하는 문제들을 풀기 어려웠는데, 이를 해결하기 위해 Chain-of-Thought(CoT) prompting 방식이 등장
    - CoT prompting은 모델이 문제 해결 과정에서 논리적인 사고단계를 거쳐 최종 답을 도출하도록 유도하는 prompting 기법
    - CoT prompting은 일반적인 prompt 방식보다 훨씬 우수한 성능을 보이며 새로운 SOTA 성과를 달성
    - fine-tuning을 수행한 모델들보다 더 좋은 성능을 보임
- Zero-shot Chain-of-Thought prompting
    - 하지만 기존 CoT Prompting은 few-shot 예시가 필요
    - 예시가 없는 경우, 추론 과정이 나타나지 않아 성능 저하 발생
    - 이를 보완하기 위해, 질문 뒤에 "Let's think step by step"이라는 한 문장을 추가해 모델이 스스로 추론 단계를 생성하도록 유도하는 Zero-Shot prompting 방법 등장

1. CNN 살펴보기

    1-1. CNN vs. FCN

*FCN(Fully-Connected Layer): 입력을받아서 출력으로 변환하는 신경망의 기본 모듈

FCN 변환 예시:

입력: 이미지가 32x32x3인 고차행렬이라고 가정 => 1x3072 1차원

모델상수: 모델이 학습해야하는 파라미터 => 10 x 3072 weights

출력: 10x1의 1차원 벡터 => 변환된 데이터를 10개 중 1개로 분류

*CNN 모델

기본 구조: 합성곱-> 활성화->풀링->완전연결층

합성곱 레이어: 입력 이미지를 필터와 연산하여 특징 맵(feature map)을 뽑아내는 모듈

:1차원 구조로 변환하는 FCN과 달리 3차원 구조를 그대로 보존하면서 연산

> 컨볼루션: 필터(3x5x5)를 이미지(3x32x32)상에서 이동시키면서 내적을 반복수행-> 내적으로 구한 모든 값을 출력 제공
> : 차원을 반드시 주의! 필터는 항상 입력의 깊이/채널 축과 동일한 차원을 가진다

특정위치에서 필터와 동일 사이즈의 이미지 영역 간 내적한 결과 값, 필요 연산은 (3*5*5 =75)차원 내적 곱 +bias 벡터

-> 필터의 이동경로를 따라 모든 연산 값을 모아 활성화 맵을 출력

합성곱 레이어의 예시: 박스필터

각 필터값과 대응되는 입력값의 곱을 모두 합산 -> 입력 가운데 위치의 출력값

합성곱 레이어의 특징

1. 출력 해상도는 입력 해상도 - 필터 해상도 + 1로 도출

2.만약 입출력 해상도를 유지하고 싶다면 -> 출력값 중 정의되지 않은 경우, 0 혹은 가장 가까운 출력값으로 대체한다(패딩)

2. CNN 모델구조

2-1. 중첩

모델 상수(파라미터)를 증가시키면? 어차피 Linear classifier

-> 비선형 블록(Conv[Linear] + ReLU[비선형 변환])과 함께하면 모델링 파워 향상

2-2. 필터

필터 시각화: 학습된 필터 시각화를 통해 각 모델(구조)가 학습한 정보를 이해가능하다.

cf. 선형모델의 필터: 각 카테고리의 템플렛(전형적모습) 역할

FCN의 필터: 다양한 템플렛 제공

CNN의 필터: 계층별로 이미지의 지역적 템플렛(예: 엣지, 색상 등)

2-3. 수용영역(리셉티브 필드)

CNN이 이미지를 처리하면서 한 번에 볼 수 있는 영역의 크기, 네트워크의 시야

일반적으로 네트워크가 깊어질 수록 수용역역도 넓어짐 -> 폭 넓은 맥락 이해가 가능하다

CNN 레이어는 이미지 작은 부분인 "지역 정보"를 추출하는데 유리하게 설계, 이미지 전체(예: 맥락)의미 파악이 필요

따라서 "지역 정보 + 이미지 전체의 맥락" 을 이해/활용하기 위한 설계가 필요하다.

->중첩 (2-1)과 수용영역(2-3)을 이용한다.

고해상도 이미지 처리 시, 많은 레이어를 통과해야한다. 이는 연산,비용,학습 가능성 고려할 때 비실용적임
실제 해법: 입력 사이즈를 줄여 모델에 입력함

2-4. 풀링

효율적 연산 및 위치 변화의 강건성 확보
지역적 특징을 더 압축된 형태로 전달하여, 합성곱 층이 더 넓은 맥락을 이해하는 데 기여한다,
입력 크기가 줄면 CNN의 연산량이 크게 줄어들기때문에 CNN 레이어의 출력을 줄여 연산 효율성을 확보해야한다.

> 위치 변화 강건성: 입력 내 객체의 위치가 다소 변해도 동일한 출력을 제공, 사실상 저해상도 정보에 근거하여 작업 수행하므로 몇 화소 이동은 무시됨
> 맥스풀링: 정해진 커널 사이즈로 이미지를 나누어 각 영역 내 가장 큰 값을 선택하는 연산 (모델 상수 학습 X)
> : 이렇게 하면 연산해야 할 데이터의 양이 줄어들어 학습과 추론 속도가 빨라지고, 메모리 사용량도 절감된다.

2-5. 스트라이드 합성곱

풀링의 한계 계선
효과: 풀링 처럼 입력 해상도를 줄임 (연산 효율과 위치 강건성)

> 일반 합성곱: 필터를 1칸씩 이동하면서 연산 수행(출력값 계산)
> 스트라이트 합성곱: 필터를 스트라이드 값만큼(S칸) 이동한 후 출력 연산
> : 풀링은 학습 상수가 없지만 스트라이드는 커널을 동시에 학습한다.
> :스트라이드 합성곱은 해상도 저하로 인한 정보 손실이 적고, 풀링 + 합성곱을 하나의 레이어로 대체함
> :고도화된 CNN에서는 스트라이드 합성곱을 풀링 대신 활용되는 경향이 있다.

3. CNN 기반 모델 변천사

3-1. AlexNet(2012), 딥러닝 CNN의 본격적인 시작점
5개의 합성곱 계층과 3개의 완전 연결 계층으로 구성된 8계층 CNN모델
3-2. VGGNet(2014),3x3 작은 필터를 깊게 쌓은 단순하면서도 강력한 구조
5개의 합성곱 블록 + 맥스 풀링 구조
:작고 단순한 필터를 깊게 쌓으면 성능이 올라간다.
:단순 설계로 모델 해설에 이상적이며, 특징 추출기, 전이 학습에 강력한 베이스라인을 가짐(파라피터가 많고 연산량 요구가 매우 크다, 거대한 모델
3-3. GoogLeNEt
3-3.ResNet(2015), 잔차 연결을 도입하고 기울기 소실 문제를 해결함
최종 요약: 깊은 모델 학습을 위한 중요한 전진(+100 레이어 학습 가능), 깊은 모델의 강력함을 다시 확인
:제안 당시 모든 벤치마크에서 최고 성능 달성, 지금도 CNN 기반 구조 중 가장 활발하게 활용
합성곱 블록(VGG 유사)과 잔차 블록
: 잔차 블록은 블록입력을 그대로 출력에 더해주는 지름길 연결이 특징. Inception모델에서 차원 축소로 활용된 1x1 합성곱을 적용, 연산 효율 개선

> 등장 배경
> 단순히 레이어를 깊게 하면 오히려 성능이 떨어짐
> -해법: 작은 모델의 성능은 "최소한" 누릴 수 있도록 작은 레이어의 추정값(입력값)을 그대로 후속 레이어에 제공하여
> 최소한 작은 레이어 수준의 성능은 보장
> 잔차블럭:
> 입력 X가 두 경로(변환경로, Shortcut 경로)로 나뉘어 전달됨 -> 마지막에 두 값을 더한다.
>
> > 일반 잔차 블록 vs 보틀넥 잔차 블록, 연산 효율을 위해 보틀넥 잔차구조 도입(더깊은 모델을, 더 적은 연산으로)
> > Stem 구조
> > 기존 모델의 효율성 레시피를 잘 활용
> > (스템 구조를 모델 초기에 도입, 입구 데이터 해상도를 1/4로 줄임, 여러개의 FC레이어를 제거하고 전역 평균 풀링을 사용-> 마지막 1개 FC만 적용)

3-4. MobileNet, 모바일.임베디드 환경 최적화
기존 합성곱은 공간과 채널을 동시에 처리하여 과도한 연산량아 요구됨

해법: 공간과 채널을 두 단계로 분리하여 처리한다.
> 깊이별 합성곱:각 채널별 독립적 3x3 합성곱 수행

> 화소별 합성곱: 1x1 합성곱을 채널 방향으로 적용
> 효과: 연산량 기존 대비 9배 가량 절감, 모델 상수 대폭 감소

딥러닝 학습 안정성 및 모델 구성 요소 요약

1. 학습의 일반적인 문제

- 학습 불안정: 손실 폭발, 학습 멈춤(기울기 소실), 수렴 실패.
- 과적합: 훈련 데이터에 과도하게 맞춰져 검증/실전 성능이 저하되는 현상.
- 느린 수렴: 최적점에 도달하는 데 불필요하게 많은 에폭 소모.
- 좋은 모델 구조만으로는 성능을 보장할 수 없습니다.

2. 모델 구성 및 안정화 전략

2.1. 활성화 함수 (Activation Function)

- 목적: 신경망에 비선형성을 부여하여 복잡한 패턴 학습을 가능하게 합니다.
- Sigmoid (출력 [0, 1]):
  - 문제: 기울기 소실, 편향된 업데이트(출력 항상 양수), 계산 비용 높음.
- Tanh (출력 [-1, 1]):
  - 장점: 0 중심 대칭으로 학습 안정성 높음.
  - 문제: 기울기 소실 문제 여전, 계산 비용 높음.
- ReLU (f(x) = max(0, x)):
  - 장점: 양의 영역에서 기울기 소실 크게 감소, 계산 효율성 및 학습 속도 빠름.
  - 문제: 죽은 뉴런(Dead ReLU) 문제 발생, 0을 기준으로 비대칭.
- Leaky ReLU (f(x) = max(0.01x, x)):
  - 장점: ReLU 장점 수용, 죽은 뉴런 문제 해결.
- ELU:
  - 장점: ReLU 장점 수용, 평균 출력이 0 근처에 위치하여 학습 안정성 높음.
  - 문제: 계산 복잡성 증가, 하이퍼파라미터 알파(α) 설정 필요.

2.2. 데이터 전처리

- 목적: 학습/검증/테스트 데이터 간의 조건을 통일하고 모델 입력에 적합하게 조정합니다.
- 통일 요소: 이미지 해상도, 색상, 밝기, 정규화.
- 정규화 방식:
  - AlexNet: 학습 데이터 전체의 평균 이미지를 입력에서 뺌.
  - VGG: R, G, B 채널별 평균을 입력에서 뺌.
  - ResNet: 채널별 평균을 빼고 채널별 표준편차로 나눔 (표준화).

2.3. 모델 상수 초기화 (가중치 W, 편향 b)

- 0 초기화: 모든 뉴런의 출력 및 기울기가 동일해져 학습이 불가능합니다.
- 랜덤 초기화: 깊은 모델에서는 분산(상수 크기)에 따라 기울기 소실/폭발 위험이 있습니다.
- Xavier 초기화: 가중치 초기 분포의 분산을 입력 차원(Nin)에 맞춰 설계합니다. (입출력 대칭 분포 가정)
- He 초기화: 가중치 초기 분포의 분산을 2 * 입력 차원(2/Nin)에 맞춰 설계합니다. ReLU와 같은 비대칭 활성화 함수에 적합합니다.
- ResNet의 등장 배경: 잔차 연결(Skip Connection)을 사용해 깊은 신경망도 안정적으로 학습할 수 있으며, 초기에는 He 초기화 등을 통해 안정적인 학습 출발을 유도합니다.

2.4. 모델 정규화 (과적합 방지)

- 가중치 감소 (Weight Decay/Regularization): 큰 가중치에 패널티를 부여하여 과적합을 방지합니다.
  - L2 정규화 (Ridge): 가중치 제곱에 패널티. 모든 가중치가 골고루 작아집니다.
  - L1 정규화 (Lasso): 가중치 절대값에 패널티. 중요하지 않은 가중치를 완전히 0으로 만들어 희소한 모델을 만들고 특성 선택 효과가 있습니다.
  - Elastic Net (L1+L2): L1과 L2의 장점을 결합합니다.
- Dropout: 학습 과정에서 일부 뉴런을 확률적으로 끔(0으로 설정)으로써 과적합을 방지하고 일반화 성능을 높입니다 (앙상블 효과).
  - 단점: 학습 시간 증가, 배치 정규화가 적용된 모델에서는 비선호됨.
  - Inverted Dropout: 학습 시 살아남은 뉴런의 출력을 스케일링하여 테스트 시 추가 연산 없이 출력 크기를 유지합니다.

3. 학습 안정성 전략 (학습률 및 종료)

- 학습률(Learning Rate) 조정: 큰 값에서 시작하여 에폭이 지날수록 작은 값으로 조정합니다.
  - 계단식 변경: 일정 에폭마다 계단식으로 줄임.
  - 코사인 변경: 코사인 함수 곡선처럼 부드럽게 줄여나감.
  - 선형 변경: 직선으로 줄여나감 (사전학습/미세조정).
- 빠른 종료 (Early Stopping): 검증 손실을 모니터링하여 더 이상 성능이 좋아지지 않을 때 학습을 조기에 멈춥니다.

하이퍼파라미터 (Hyperparameter)

- 정의: 학습 전에 사용자가 정하는 값 (예: 학습률, 배치 사이즈, 드롭아웃 비율).
- 탐색 방식:
  - 그리드 탐색 (Grid Search): 후보 값들을 격자처럼 모든 조합으로 탐색.
  - 랜덤 탐색 (Random Search): 무작위로 값을 선택해 탐색.
- 체크리스트:
  i. 초기 손실 확인: CE 손실이라면 log(C)와 유사해야 정상입니다.
  ii. 작은 샘플 과적합: 작은 샘플로 고의로 과적합(정확도 100%)을 시도하여 모델의 학습 능력을 확인합니다.
  iii. 최적 학습률 탐색: 모든 데이터로 손실이 실제로 줄어드는 학습률을 우선 탐색합니다.
  iv. 좋은 조합 탐색: 후보 학습률과 가중치 변형 방식 중 초기 몇 에폭 내에 손실 감소 및 검증 성능이 좋은 조합을 선택합니다.
  v. 학습률에 따른 손실곡선 관찰: 초반 손실 유지 또는 손실 감소 후 정체 시 학습률 변경을 시도합니다.

vi. 과적합/약한 모델 판단: 학습/검증 정확도가 크게 벌어지면 과적합, 결과 차이가 매우 적다면 모델의 힘이 약하다고 판단합니다.

- 학습률(Learning Rate) 조정: 큰 값에서 시작하여 에폭이 지날수록 작은 값으로 조정합니다.
  - 계단식 변경: 일정 에폭마다 계단식으로 줄임.
  - 코사인 변경: 코사인 함수 곡선처럼 부드럽게 줄여나감.
  - 선형 변경: 직선으로 줄여나감 (사전학습/미세조정).
- 빠른 종료 (Early Stopping): 검증 손실을 모니터링하여 더 이상 성능이 좋아지지 않을 때 학습을 조기에 멈춥니다.

하이퍼파라미터 (Hyperparameter)

- 정의: 학습 전에 사용자가 정하는 값 (예: 학습률, 배치 사이즈, 드롭아웃 비율).
- 탐색 방식:
  - 그리드 탐색 (Grid Search): 후보 값들을 격자처럼 모든 조합으로 탐색.
  - 랜덤 탐색 (Random Search): 무작위로 값을 선택해 탐색.
- 체크리스트:
  i. 초기 손실 확인: CE 손실이라면 log(C)와 유사해야 정상입니다.
  ii. 작은 샘플 과적합: 작은 샘플로 고의로 과적합(정확도 100%)을 시도하여 모델의 학습 능력을 확인합니다.
  iii. 최적 학습률 탐색: 모든 데이터로 손실이 실제로 줄어드는 학습률을 우선 탐색합니다.
  iv. 좋은 조합 탐색: 후보 학습률과 가중치 변형 방식 중 초기 몇 에폭 내에 손실 감소 및 검증 성능이 좋은 조합을 선택합니다.
  v. 학습률에 따른 손실곡선 관찰: 초반 손실 유지 또는 손실 감소 후 정체 시 학습률 변경을 시도합니다.
  vi. 과적합/약한 모델 판단: 학습/검증 정확도가 크게 벌어지면 과적합, 결과 차이가 매우 적다면 모델의 힘이 약하다고 판단합니다.