Transfomer-총정리

Seq2seq

Seq2seq 기법은 encoder가 문맥을 고려한 sequence를 받아 context vector를 만들고 이 context vector로 decoder에서 sequence를 만드는, 다음 state를 예측하는 기법이다.

Seq2seq 의 단점은

1. sequence가 길면 처음 정보가 미미해진다.
2. Fixed size context vector로 긴 sequence의 정보를 압축하기 때문에 잘 압축되지 않는다.
3. 모든 token이 영향을 미치는 것 때문에 중요하지 않은 token의 영향을 받는다.

1의 단점을 보완하기 위해 만들어진 기법이 attention이다.

Attention Mechanism

RNN에서는 어떤 input의 output은 러닝 용도로만 활용되고 context vector로 가게 되는데, attention method에서는 1개의 input 도 활용하여 attention 하는데 활용한다.

어텐션의 기본 아이디어는 디코더에서 출력 단어를 예측하는 매 시점(time step)마다, 인코더에서의 전체 입력 문장을 다시 한 번 참고한다는 것이다. 단, 전체 입력 문장을 전부 다 동일한 비율로 참고하는 것이 아니라, 해당 시점에서 예측해야할 단어와 연관이 있는 입력 단어 부분을 좀 더 집중(attention)해서 보게 된다. 따라서 RNN에서 있었던 입력 시퀀스가 길어졌을 때 발생하는 문제를 보정할 수 있게 된다.

Attention Mechanism에서 필요한 paramater에는 Query, Keys, Values가 있다. 각각 의미하는 바는 다음과 같다.

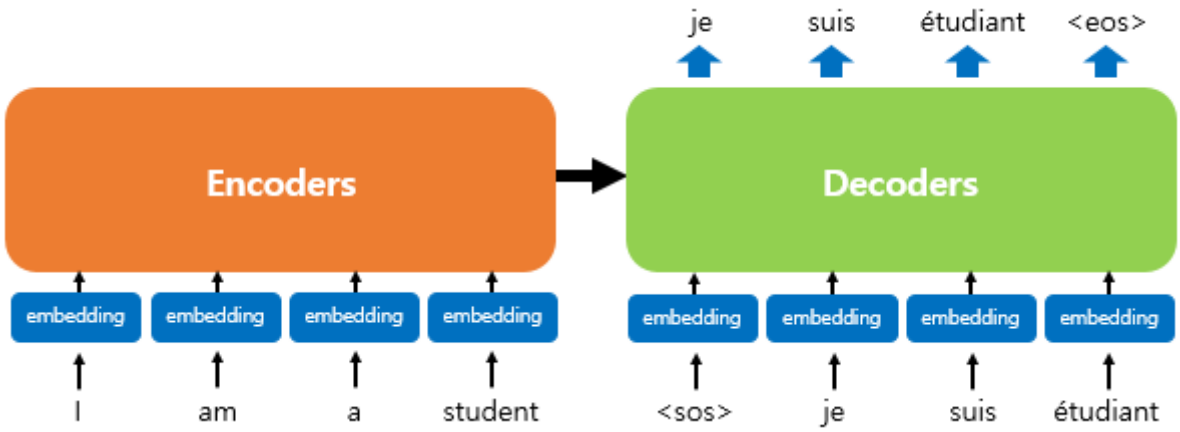
```
Q = Query : t 시점의 디코더 셀에서의 은닉 상태
K = Keys : 모든 시점의 인코더 셀의 은닉 상태들
V = Values : 모든 시점의 인코더 셀의 은닉 상태들
```

이 attention을 이용하여 RNN+attention을 적용한 모델이 나왔지만, 여전히 RNN을 이용한 학습은 오랜시간이 걸리고 위의 단점들을 보완하고, attention의 다른 방식을 사용한 모델이 Transformer이다.

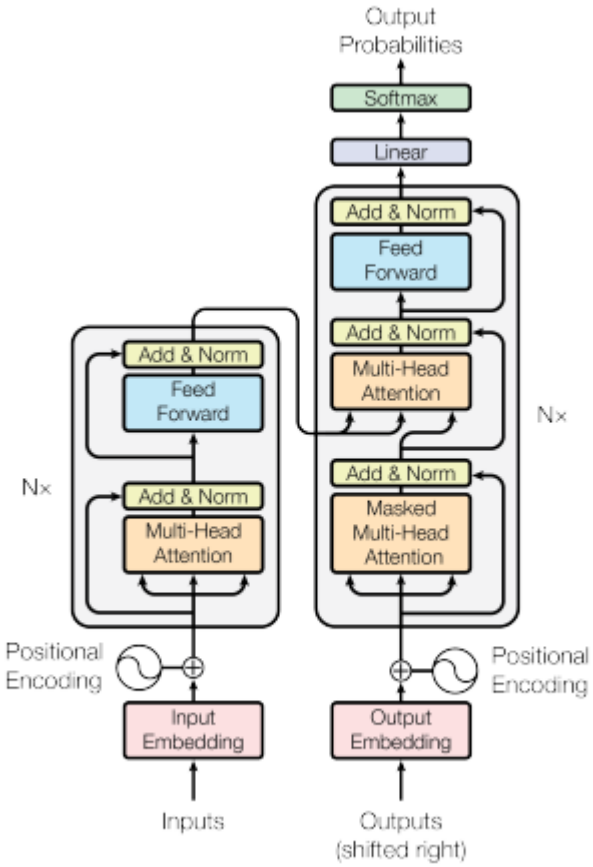
Transformer

이 모델은 RNN을 사용하지 않고, attention method만 이용해서 인코더-디코더 구조를 설계하였음에도 성능도 RNN보다 우수하다는 특징을 갖고있다.

Seq2seq와 인코더, 디코더를 같이 쓴다는 점에서 구조적으로 같지만 다른 점은 인코더와 디코더라는 단위가 N개가 존재할 수 있다는 점이다. seq2sesq 모델에서는 인코더나 디코더 내에 있는 RNN 셀이 시점별로 이동하고 t개의 시점을 가지는 반면에, transformer에서는 단위가 N개로 구성되어 있다.



RNN+ attention methods는 decoder가 해석하기에 적합한 weight을 찾으려고 학습을 시켰다면, transformer는 self-attention을 사용하여 encoder가 출력값을 내기 적합한 weight를 찾으려고 학습을 시켰다.

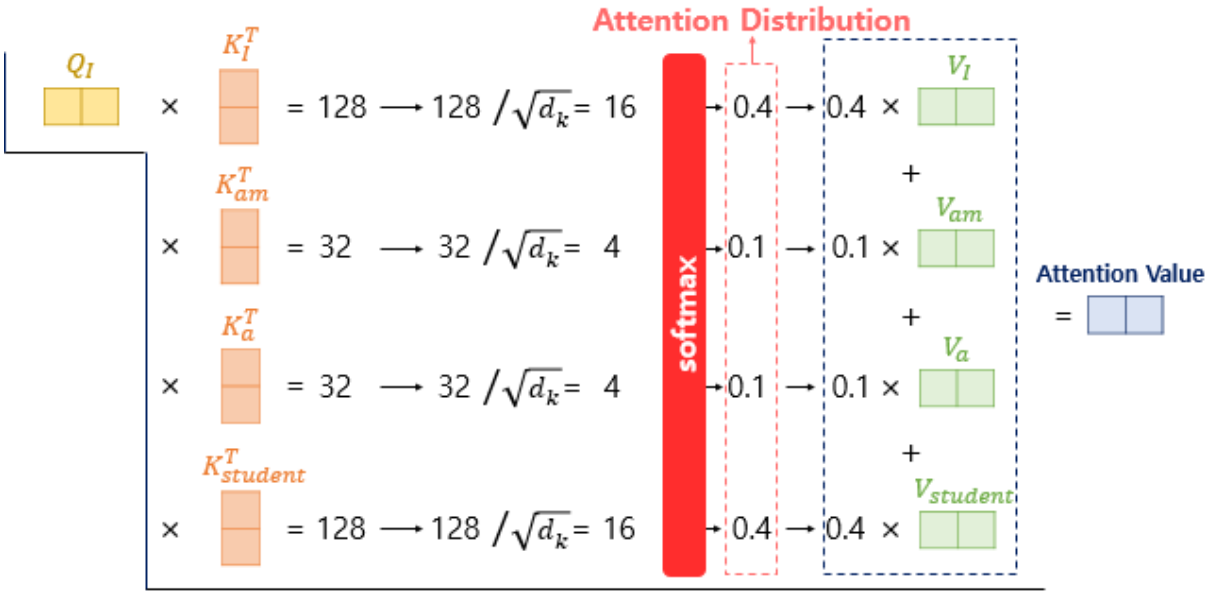


Self Attention

Attention 은 query 값과 key값의 유사도를 구해 value와 곱하고 가중합 해서 리턴하는 방식이다. Self attention은 encoder가 해석하기에 적합한 weight를 찾는 것이기 때문에 Q(query),K(keys),V(values)가 모두 encoder안에서 나오는데 이는 decoder가 해석하기에 적합한 weight를 찾기 때문에 K,V 는 encoder에서, Q 는 decoder에서 나오는 attention과는 다르다.

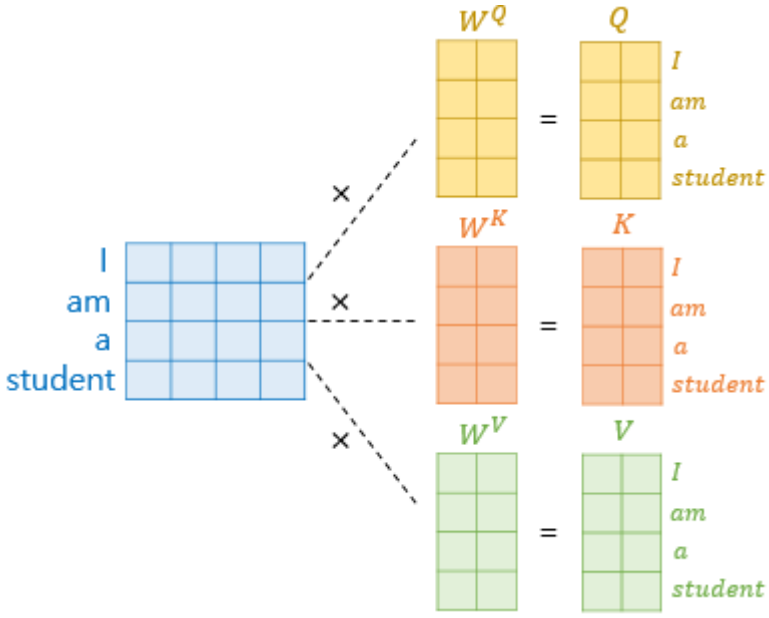
각각 Q,K,V 벡터는 처음에 무작위로 initialize 한 가중치 벡터 W^Q W^K W^V 와 embedded 된 input을 곱한 값으로 얻을 수 있다.

Q 벡터와 K 벡터를 내적하면 attention score가 나오고 attention method와 마찬가지로 softmax() 함수를 이용해 attention distribution을 얻는다. 이것과 V 벡터와 가중합하면 attention value를 얻을 수 있다.

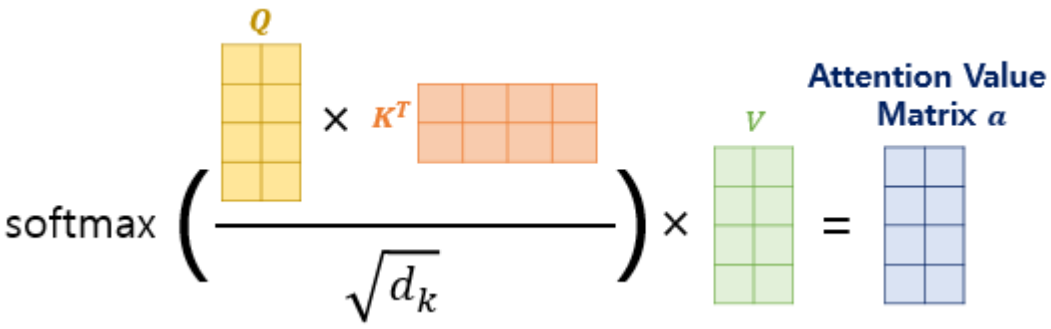


행렬 연산으로 일괄 처리

Input data 하나하나를 위와 같이 계산하는 것이 아니라 실제로는 input 시퀀스 모두를 행렬 연산으로 한번에 처리하는데 그림으로 표현하면 아래와 같다.



하나하나 할때와 마찬가지의 과정을 거치면, 다음과 같은 값을 얻을 수 있다.

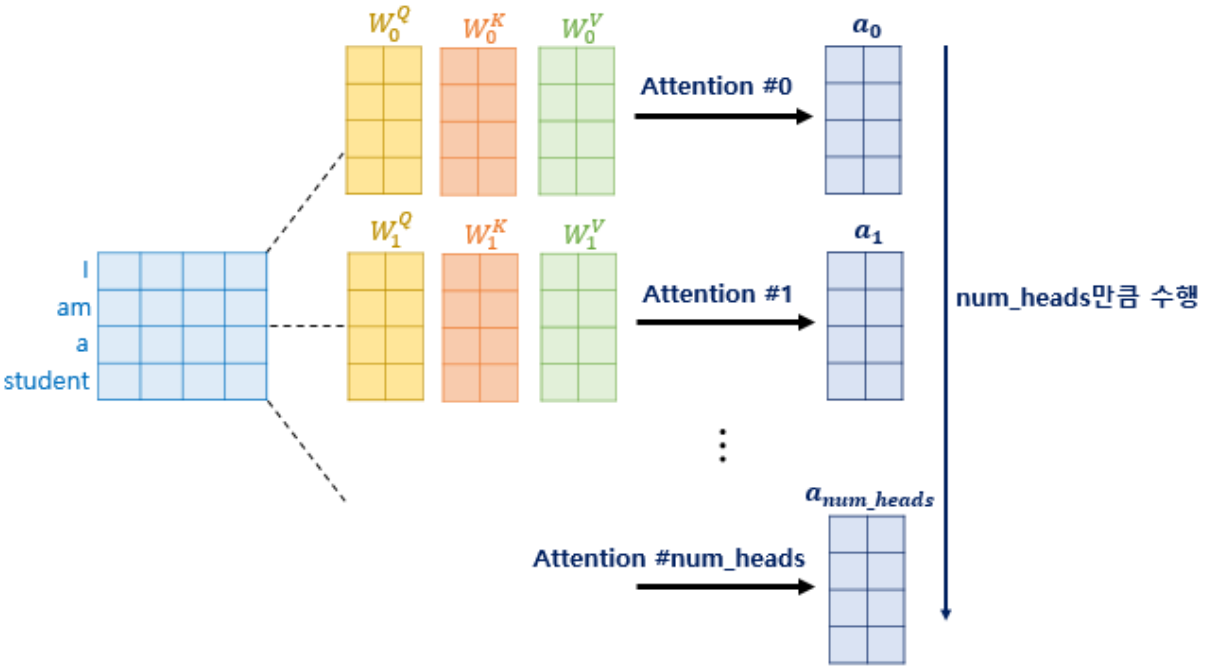


여기서 $\sqrt{d_k}$를 Q와 K의 내적 값을 나눠주는데 이 이유는 Q,K의 dimension이 너무 크게 되면 softmax함수가 매우 작은 값으로 되기 때문에 이것을 줄이기 위해서 $\sqrt{d_k}$로 나눠줬다고 한다.

(위 그림은 multi-head attention 일때 하나의 head 결과를 보여준 것으로 실제 one-head attention은 Q,K,V벡터의 크기가 다르기 때문에 attention value가 input과 같은 크기로 나온다. 단순히 행렬로 곱셈을 한번에 어떻게 하는지에 대한 간략한 이해만 하는게 좋다.)

one-head self attention 이면 Wv가 그림에선 4x4가 되는것이 맞다.

Multi-head attention

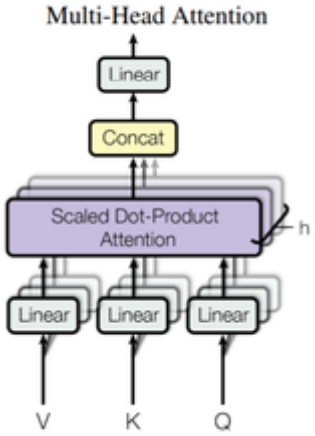


Multi-head attention은 attention을 여러번을 하는 것인데 이렇게 하는 이유는

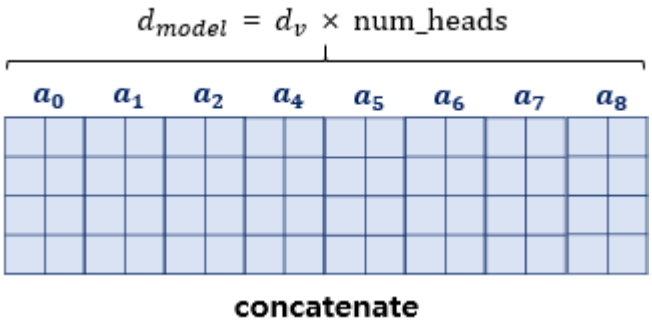
1. Multi-head로 하면 한 주제에 대해 여러가지 관점으로 해석할 수 있으므로 더 정확성이 올라간다.
2. Multi-head로 계산하더라도 병렬로 처리되고, linear과정을 통해 차원을 축소시키기 때문에 계산 시간은 똑같다.

RNN은 병렬 처리를 못하는데 Transfromer는 attention으로만 구성되어 있기 때문에 가능한 일이다.

논문에서는 여러번의 attention을 병렬로 하는게 더 효과적이라고 생각해서 d_{model} 을 num_heads로 나누어서 output value를 얻는다.

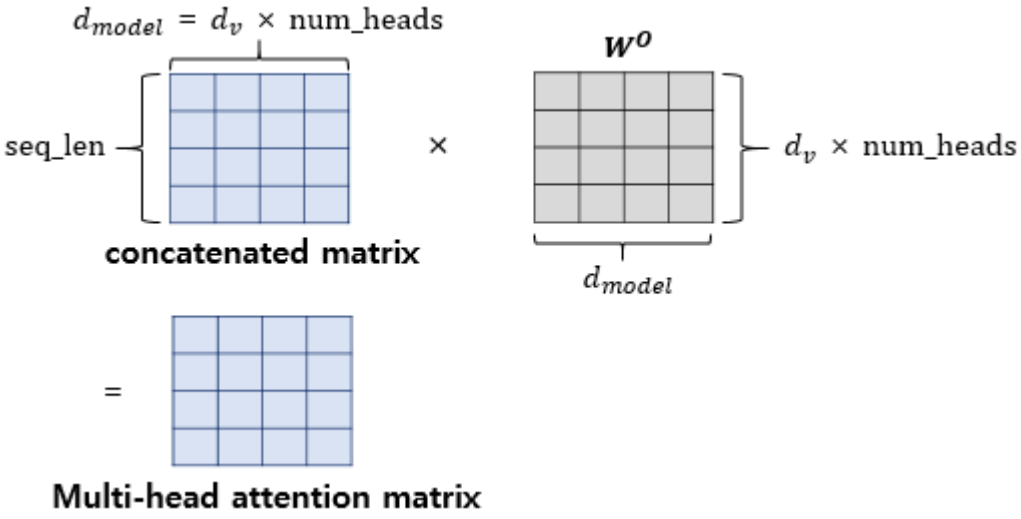


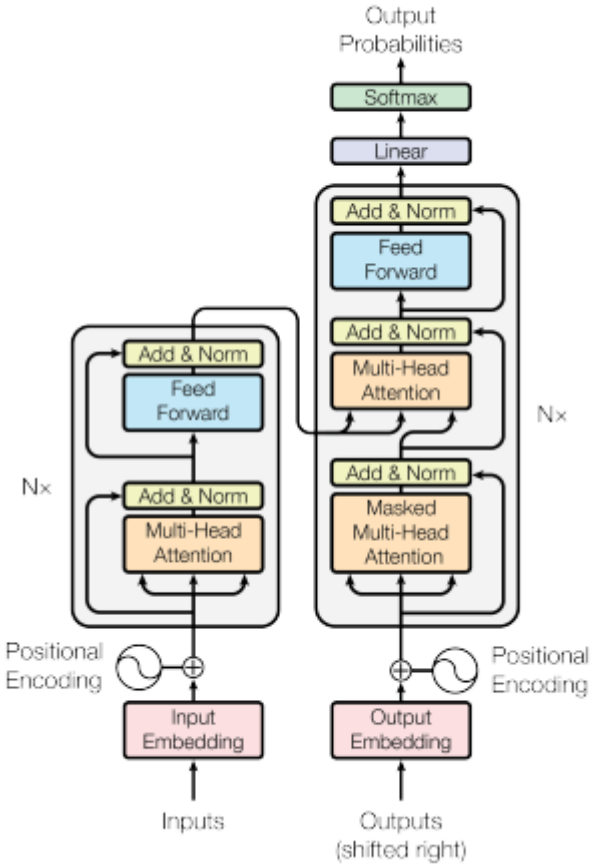
각각의 가중치의 차원을 num_heads로 나눠 줘서 계산을 진행하고 (위 그림에서 아랫부분 Linear) concatenate가 끝나고 다른 가중치 W를 써서 Linear과정을 한번더 거치고 input과 같은 차원을 만들어 준다.

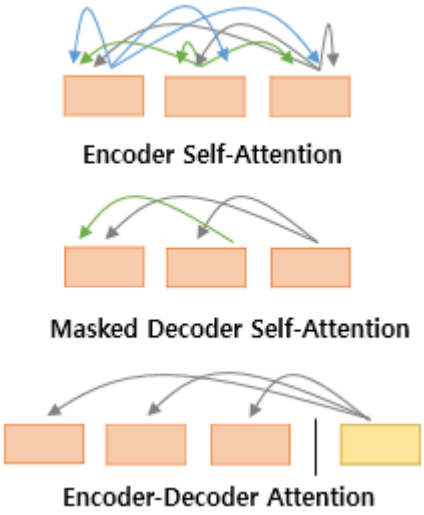


마지막에 각각의 attention head 값을 연결 시켜서 가중치를 곱하면 multi-head attention matrix 가 나온다.

설계할때 Q,K,V의 벡터 사이즈를 num_heads값에 맞추서 설계해서 결국 d_{model} 사이즈의 multi-head attention matrix가 나오기 때문에 전체 computational cost는 single-head attention 과 비슷하다.


$d_{model} = d_v \times$ num_heads
seq_len
concatenated matrix
×
W^O
$d_v \times$ num_heads
d_{model}
=
Multi-head attention matrix


Output Probabilities
Softmax
Linear
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Add & Norm
Masked Multi-Head Attention
N×
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
N×
Positional Encoding
Positional Encoding
Input Embedding
Output Embedding
Inputs
Outputs (shifted right)



Position-wise Feed-Forward Networks

multi head attention의 결과로 나온 값 x를 가지고 가중치 행렬 W_1과W_2로 연산을 해주는데 식은 $FFNN(x) = MAX(0, xW_1 + b_1)W_2 + b_2$ 이다. 한 encoder layer에서는 같은 값을 쓰지만 다른 layer에서는 다른 값을 쓴다.

이 부분의 의미는 앞에 값이 수식을 지나고 원래의 값의 색깔을 너무 잃어버리지 않도록 한 번 더해주는 것이다. input_data + multi-head self attention의 결과를 더해준것이라 생각해도 된다.

multi-head self attention 의 각 값들을 한쪽에 치우쳐지지 않도록 균등하게 섞는 역활도 한다.

Decoder

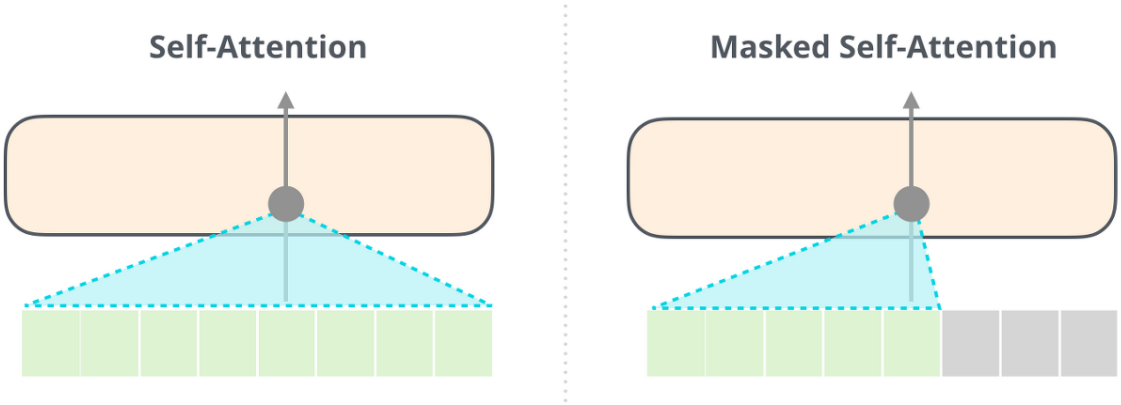
N = 6인 identical layers의 stack으로 이루어져 있다.

3개의 sub-layer가 있다. Decoder스스로 하는 masked-self attention, 그리고 원래 encoder-decoder 구조에 필요한 attention과 Feed forward layer이 있다. masked decoder self-attention은 decoder stack에 position이 subsequent position에 attending 하는 걸 막기 위해 사용된다.

Positional Encoding

transformer의 입력으로 사용되기 전에 positional encoding값이 더해진다. encoder와 decoder stack 바닥에 있다.

sin 과 cos 으로 위치값을 encoding하는 것으로 논문에서는 sin 을 활용했다.



BERT vs GPT

BERT와 GPT 는 Transformer의 인코더 부분과 디코더 부분을 떼서 따로 쓰는 모델이다. 셋은 비슷하지만 다른데 BERT 와 GPT는 언어 모델을 사전 학습시키는것이 주 목적인것에 반해, Transformer의 주 목적은 번역이었기 때문에 encoder와 decoder 두 파트를 둘다 썼던것이다.

BERT

Transformer의 encoder부분을 따서 구현되었으며 위키피디아(25억 단어)와 BooksCorpus(8억 단어)와 같은 레이블이 없는 텍스트 데이터로 사전 훈련된 언어 모델이다.

레이블이 없는 방대한 데이터로 사전 훈련된 모델을 가지고, 레이블이 있는 다른 작업(Task)에서 추가 훈련과 함께 하이퍼파라미터를 재조정하여 이 모델을 사용하면 성능이 높게 나오는 기존의 사례들을 참고하였기 때문이다. 다른 작업에 대해서 파라미터 재조정을 위한 추가 훈련 과정을 파인 튜닝(Fine-tuning)이라고 한다.

Transformer 인코더를 12개(BERT_Base, GPT-1과 비교하기 위해 만들어짐) 또는 24개(BERT_Large) 쌓아올린 구조로 Transformer 보다 큰 네트워크이다.

Pre-training, Fine-tuning

BERT는 pre-training을 먼저 진행한 후 에 fine-tuning을 진행하는 방식으로 이루어져있다. pre-training은 unlabeled data로 모델을 training시키는 것이고, fine-tuning은 pre-trained된 parameter로 init값을 쓰고 labeled data로 fine-tuning을 진행한다.

Wordpiece tokenization

BERT에서는 embedding에서 Wordpiece를 사용했는데 Wordpiece는 word embedding 과 character embedding을 합친 방식이다.

Wordpiece는 먼저 character 단위로 분리를 하고 자주 나온 character들을 병합하여 하나의 토큰으로 만들 수 있다.

Pre-training

BERT의 학습 방식은 bidirectional하다. 다음 단어를 예측하는 Statistical language model은 bidirctional하게 구축 할 수 없기 때문에 이 문제를 두가지 학습으로 해결하였고 이 두가지 방식이 MLM 과 NSP이다.

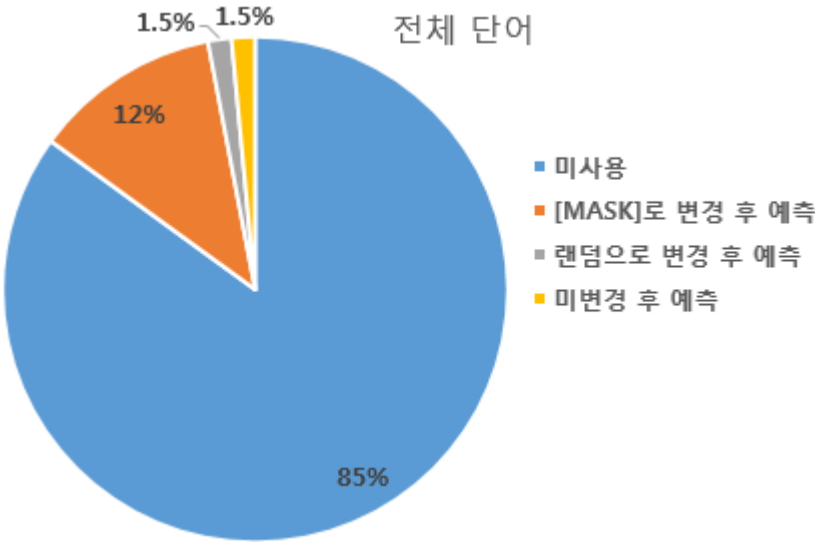
Pre-training-Masked Language Model(MLM)

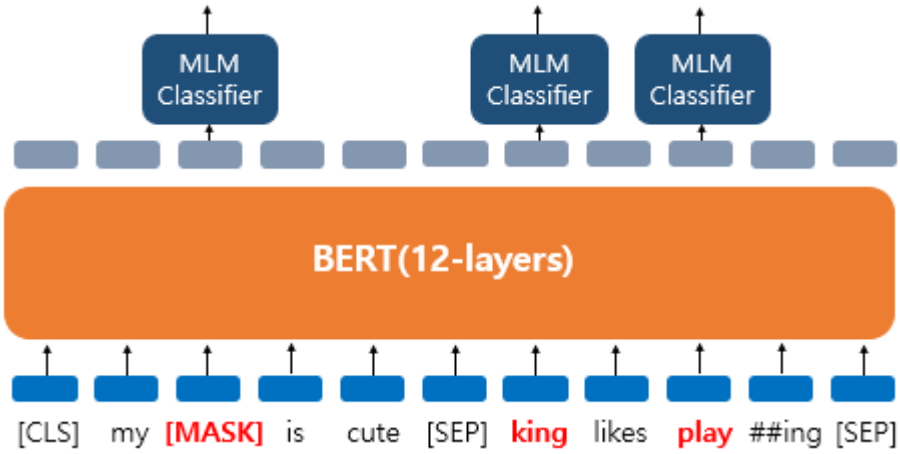
BERT는 사전 훈련을 위해서 인공 신경망의 입력으로 들어가는 입력 텍스트의 15%의 단어를 랜덤으로 마스킹(Masking)한다. 그리고 인공 신경망에게 이 가려진 단어들을(Masked words) 예측하도록 한다.

중간에 단어들에 구멍을 뚫어놓고, 구멍에 들어갈 단어들을 예측하게 하는 식이다. 예를 들어 '나는 [MASK]에 가서 그곳에서 빵과 [MASK]를 샀다'를 주고 '슈퍼'와 '우유'를 맞추게 한다.

15%의 단어들중 80%는 [MASK]로, 10%는 랜덤 단어로 바뀌고, 10%단어는 그대로 둔다. 이와 같이 mask를 하는 비율중에 또 비율을 나눈 이유는 fine-tuning시 들어오는 data는 mask가 없기 때문에 아무것도 예측할 필요가 없다고 생각할 수 있어 mask 토큰이 없어도 예측할 수 있도록 한 것이다.

밑의 그래프는 비율을 가시화한것이다.

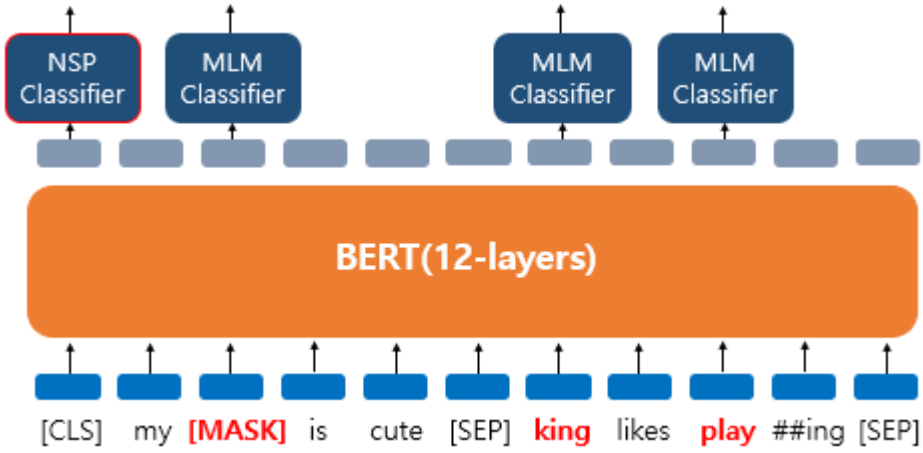




MLM bidirectional하다. 주변 단어의 context만 보고 mask된 단어를 예측하는 모델이다. Bidirectional한 모델을 사용할 수 있는 이유는 transformer의 encoder 부분을 썼기 때문인데 encoder 부분은 self-attention 을 통해 모두가 모두를 참조하여 attention하는데 decoder 부분은 masked self-attention으로 전의 position만 참조하기 때문에 앞의 단어들만 보고 뒷 단어를 예측한다.

Pre-training-Next Sentence Prediction(NSP)

BERT는 두 개의 문장을 준 후에 이 문장이 이어지는 문장인지 아닌지를 맞추는 방식으로 훈련시킨다. 이를 위해서 50:50 비율로 실제 이어지는 두 개의 문장과 랜덤으로 이어붙인 두 개의 문장을 주고 훈련시킨다.



[CLS]와 [SEP] 토큰은 BERT가 분류 문제를 풀기 위한 특별 토큰이다. 위 그림과 같이 CLS 토큰 위치의 출력층에서 NSP Classifier가 들어가 이어진 문장인지 아닌지 이진 분류를 한다. [SEP] 토큰은 두 문장을 구별하기 위한 토큰이다. 첫문장의 끝과 두번째 문장의 끝에 붙는다.

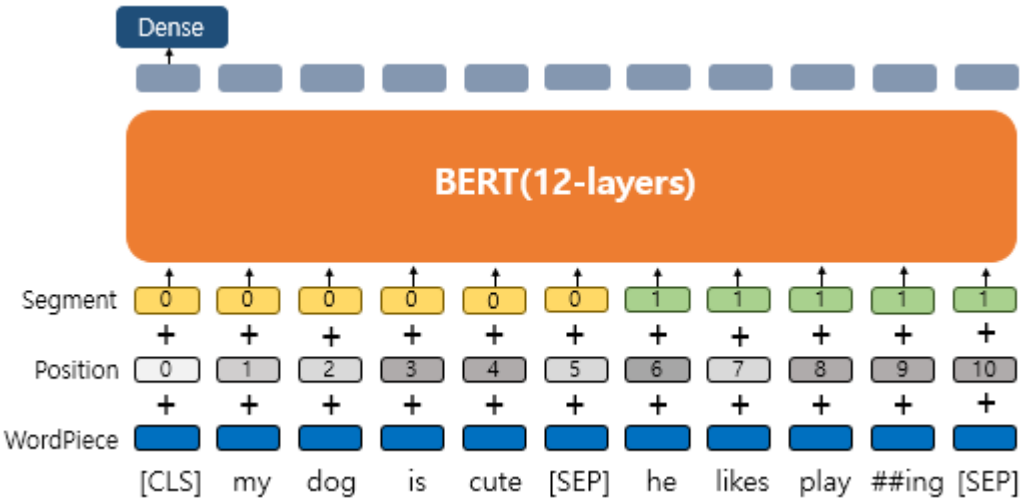
주로 QA 나 Natural Language Inference등의 task에서 MLM을 학습한것 만으로는 충분하지 않기 때문에 추가했다..

Embedding

BERT에는 3개의 임베딩 층이 사용된다. BERT의 장점중 하나가 dynamic embedding이다. 같은 단어라 할지라도 문맥에 따라 다른 벡터를 가진다.

- WordPiece Embedding : 실질적인 입력이 되는 워드 임베딩. 임베딩 벡터의 종류는 단어 집합의 크기로 30,522개.
- Position Embedding : 위치 정보를 학습하기 위한 임베딩. 임베딩 벡터의 종류는 문장의 최대 길이인 512개. (Transformer는 positional encoding이다.)
- Segment Embedding : 두 개의 문장을 구분하기 위한 임베딩. 임베딩 벡터의 종류는 문장의 최대 개수인 2개.

Embedding 층을 표현한 그림은 다음과 같다.



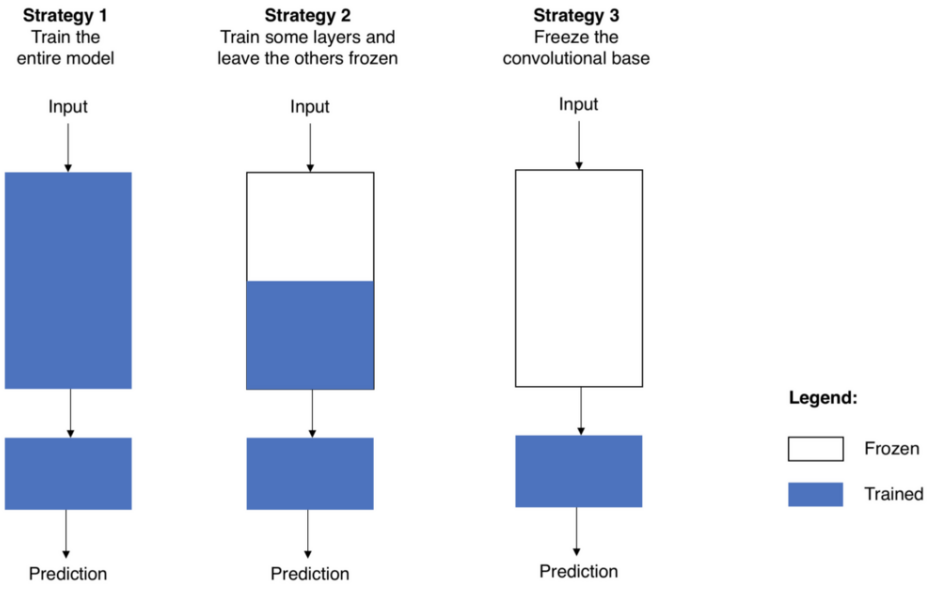
Segment Embedding에서 두 개의 문장이라고 설명 되어 있지만 사실상 두개의 문서 혹은 두 종류의 텍스트라고 할 수 있다. 예를 들어, QA문제를 푸는 경우에는 Question 부분과 Paragraph 부분이 나뉘는데 본문이 여러개의 문장으로 구성 될 수 있기 때문이다. 반대로 입력이 하나의 문장일 때는 Segment 가 0으로 embedding 함으로써 해결 할 수 있다.

Fine-tuning

Pre-tuning이 끝나고 난 뒤, 우리가 원하는 downstream task에 관해서 fine-tuning을 진행하게 된다.

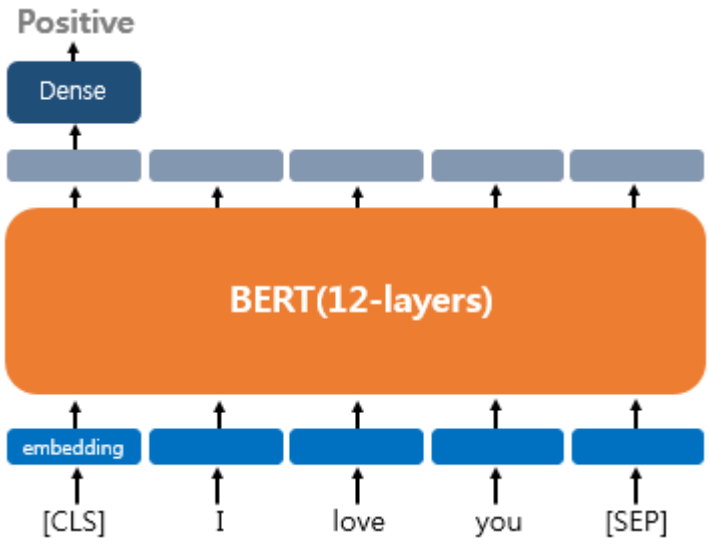
Fine-tunning에는 세가지 전략이 있다.

1. 전체 모델을 새로 학습시키기
   사전학습 모델의 구조만 사용하고 전부 새로 학습시키는것이다.
2. Convolutional base의 일부분은 고정시키고 나머지 계층과 classifier를 새로 학습시키기
   낮은 레벨(어떤 문제를 푸느냐에 상관없이 독립적인 특징)의 특징과 높은 레벨의 특징을 추출해서 어느 정도까지 재학습시킬지를 정할 수 있다고 한다.
3. Convloutional base는 고정시키고, classifier만 새로 학습시키기
   컴퓨팅 연산 능력이 부족하거나 데이터 셋이 너무 작을때, 내가 풀고자하는 문제가 이미 학습한 데이터셋과 매우 비슷할때 사용한다.



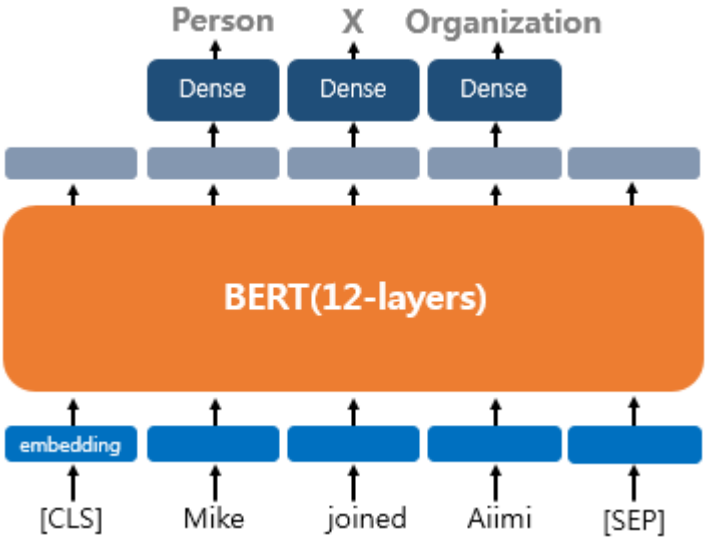
Fine-tuning의 종류에는 여러가지가 있는데

a. 하나의 텍스트에 대한 텍스트 분류 유형(Single Text Classification)



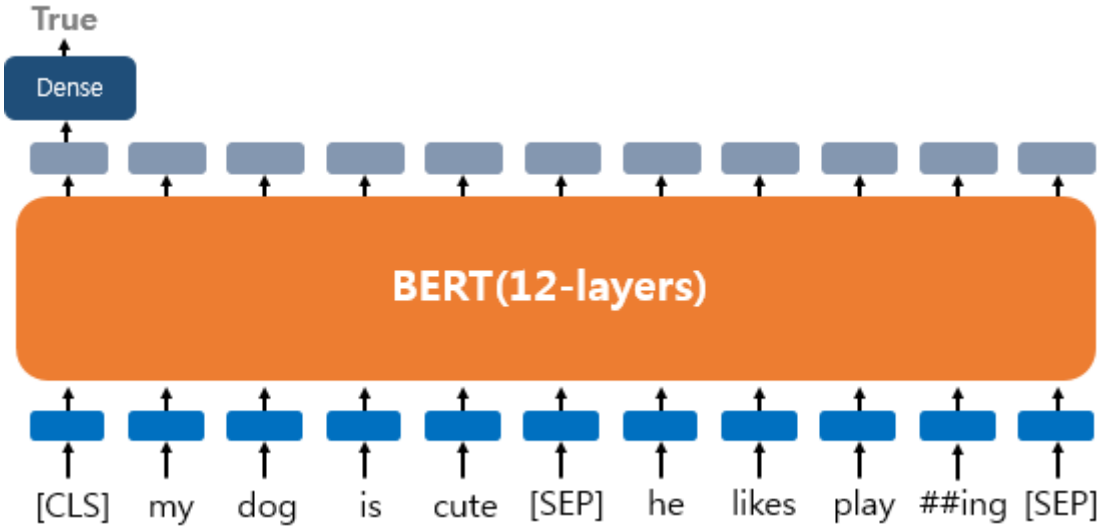
영화 리뷰 감성 분류, 로이터 뉴스 분류 등과 같이 입력된 문서에 대해서 분류를 하는 유형으로 문서의 시작에 [CLS] 라는 토큰을 입력한다. 그 다음 [CLS] 토큰의 위치의 출력층에서 밀집층(Dense layer) 또는 같은 이름으로는 완전 연결층(fully-connected layer)이라고 불리는 층들을 추가하여 분류에 대한 예측을 하게된다.

b. 하나의 텍스트에 대한 태깅 작업(Tagging)



RNN 계열의 신경망들을 이용해서 풀었던 task와 같다. 대표적으로 문장의 각 단어에 품사를 태깅하는 품사 태깅 작업과 개체를 태깅하는 개체명 인식 작업이 있다. 출력층에서는 입력 텍스트의 각 토큰의 위치에 밀집층을 사용하여 분류에 대한 예측을 하게 됩니다.

c. 텍스트 쌍에 대한 부류 혹은 회귀 문제(Text Pair Classification or Regression)

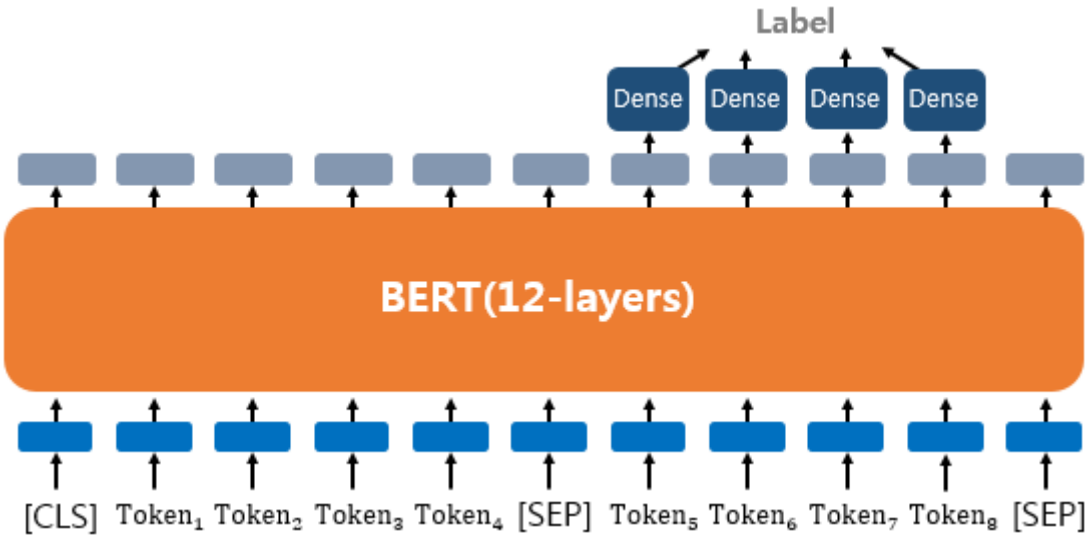


두 문장이 주어졌을 때, 하나의 문장이 다른 문장과 논리적으로 어떤 관계에 있는지를 분류하는 task이다. 유형으로는 모순 관계(contradiction), 함의 관계(entailment), 중립 관계(neutral)가 있다.

텍스트의 쌍을 입력받는 이러한 태스크의 경우에는 입력 텍스트가 1개가 아니므로, 텍스트 사이에 [SEP] 토큰을 집어넣고, Sentence 0 임베딩과 Sentence 1 임베딩이라는 두 종류의

세그먼트 임베딩을 모두 사용하여 문서를 구분한다.

d. 질의 응답(Question Answering)



텍스트의 쌍을 입력으로 받는 또 다른 task이다. 대표적인 데이터셋으로 SQuAD(Stanford Question Answering Dataset)이 있다.

ALBERT

일반적으로 BERT 같은 pre-trained language representation 모델은 모델의 크기가 커지면 성능이 향상된다. 하지만 너무 커지면 memory limitation 과 memory degradation 때문에 오히려 모델 성능이 감소한다.

이를 해결하기 위해 나온 모델이 ALBERT이다. 모델의 크기는 줄이고 더 좋은 성능을 얻었다.

1. Factorized embedding parameterization
   Input layer의 parameter 수를 줄였다.

   일반적으로 Input Token embedding size와 hidden size는 같은데 hidden 은 context 정보도 포함하고 있으니 정보량이 더 많다고 할 수 있다. 따라서 Input Token embedding size를 줄여도 된다. 하지만 이렇게 했을 때 transformer layer에서 차원이 맞지 않게 된다. 이것을 해결하기 위해 V가 vocabulary size 라면 VxH Matrix를 이용해

Token embedding을 했다고 한다.

2. Cross-layer parameter sharing
   Transformer의 각 Layer 간에 같은 parameter를 공유했다.
   Layer간의 Parameter를 공유해도 성능은 떨어지지 않았지만 FFN까지 같이 공유했을 때는 성능이 떨어지는것을 확인할 수 있었다. 그래도 크게 성능이 떨어지지 않기 때문에 model size를 낮추는데 더 의미가 있다고 한다.
3. Sentence order prediction
   NSP 대신 두 문장간 순서를 맞추는 방식으로 학습했다. NSP보다 더 나은 학습 방식으로 제안된 것인데, NSP는 한 문장이 주어지고 임의로 뽑은 문장과 연관관계를 보는것인데 임의로 뽑은 문장이 다른 Topic일 가능성이 커서 topic prediction에 가깝다.
   하지만 SOP는 실제 연속인 두 문장을 순서만 앞뒤로 바꾼것으로 학습이 진행된다. 이렇게 한 결과로 SOP가 대부분의 downstream task에서 잘 된것을 볼 수 있다. 또한 NSP로 학습한 것으로 SOP의 성능은 낮지만 SOP로 학습한 것으로 NSP의 성능은 괜찮다.

BERT와 비교하자면

1. BERT large- 334mb, ALBERT large - 18mb(memory size)
2. Training time (ALBERT large가 BERT large보다 2배빠르다)
   이 이유는 communication과 computation이 훨씬 줄기 때문이다.

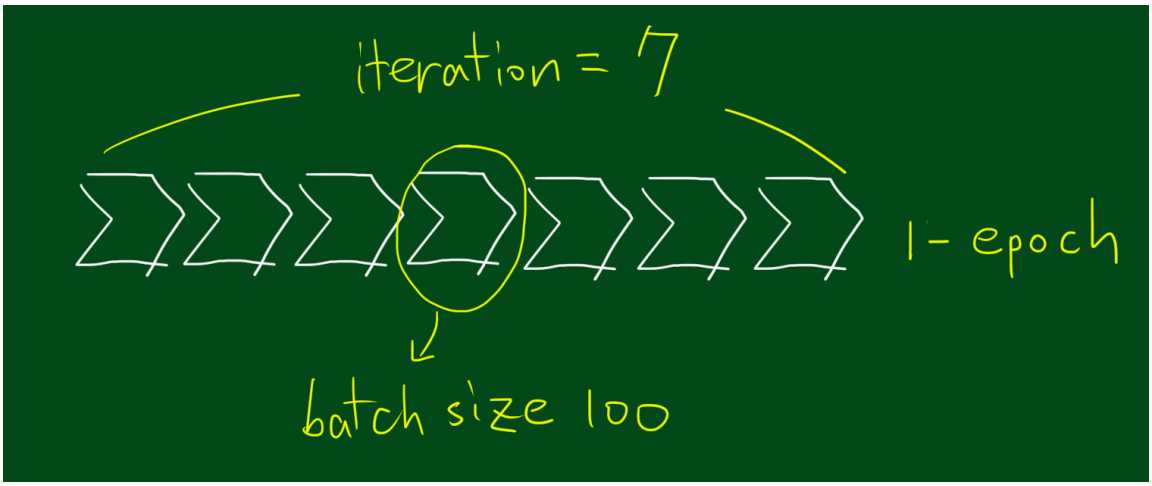
RoBERTa

BERT를 발표할 때 모델들에서 hyper parameter들에 대한 명확한 정보가 없었고, BERT의 학습 시간이 길어, hyper parameter들을 조정하면서 실험하는게 어려웠다. Pre-training 단계의 hyper parameter에 관한 정보를 얻기 위해 실험한 것이 RoBERTa이다.

1. Dynamic Masking
   기존의 BERT가 무작위로 token에 mask를 씌우고 그것을 예측하는 방식이었는데, 학습을 시작하기 전에 mask를 씌웠다. RoBERTa에서는 매 epoch마다 mask를 새로 씌우는 dynamic masking을 사용했다.
   새로 masking을 해서 학습시간은 늘어나지만, 메모리를 아낄 수 있고, static masking보다 더 좋은 결과가 나왔다고 한다.

2. Input format / NSP
   NSP를 사용했을 때 아무 두 문장을 연결해서 매우 짧은 input이 만들어지고, token의 수가 512에 턱없이 모자라는 경우가 있지만 NSP를 빼고 난 후에는 512안에 최대한 가깝게 문장들을 이었기 때문에 더 효율적이라 할 수 있다. 이 부분에서도 BERT보다 나은 성능을 보였다고 한다.

3. Batch size
   RoBERTa 는 batch의 영향력을 확인하기 위한 실험을 진행했는데, 실험 결과는 batch size가 크면 클수록 성능이 좋았다고 한다. 비례하게 커지진 않지만 parallelize에 유리하다.



4. Tokenizer
   BERT와 다르게 GTP-2에 사용한 tokenizer를 사용했다..

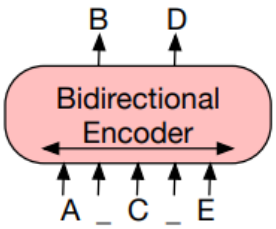
5. Data
   BERT는 16GB으로 pre-train 했지만 RoBERTa 는 160GB로 했고, 성능이 더 좋았다. 학습시간을 길게할수록 성능이 올라가는것을 확인했다.

이러한 RoBERTa의 실험들은 새로운 deeplearning 모델을 제시하는 것도 중요하지만 안의 hyperparameter의 값을 조정하는 것도 좋은 성능의 모델을 얻는데 중요하다는 것을 알려준다.

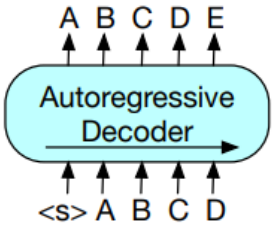
BART(Bidirectional Auto-Regressive Transformer)

BART의 핵심은 임의의 noise function에 의해 변환된 원본 텍스트를 가지고, 이 원본 텍스트를 복구하도록 학습한다는 것이다.

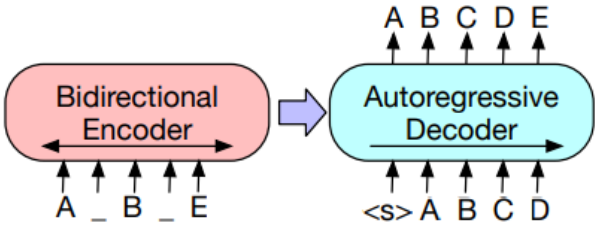
이렇게 미리 학습했을 때 장점은 text에 noise가 있을 때 복구하는 성능이 좋다는 것이다. Text generation과 reading comprehension task에서 효과적이었고, abstractive dialogue summarization, QA, summarization에서 SOTA를 달성했다고 한다.



(a) BERT: Random tokens are replaced with masks, and the document is encoded bidirectionally. Missing tokens are predicted independently, so BERT cannot easily be used for generation.



(b) GPT: Tokens are predicted auto-regressively, meaning GPT can be used for generation. However words can only condition on leftward context, so it cannot learn bidirectional interactions.



BART는 각각 transformer의 인코더, 디코더 부분만 사용하는 BERT, GPT와는 다르게 모두 사용한다.

Machine translation을 위해 word embedding을 대체하는 추가적인 encoder를 넣었다.

기본적으로 구조는 Seq2seq transformer와 같지만 ReLU activation function을 GeLUs로 바꿨다.

Activation function

입력 신호의 총합을 출력신호로 변환하는 함수를 activation function이라 한다.

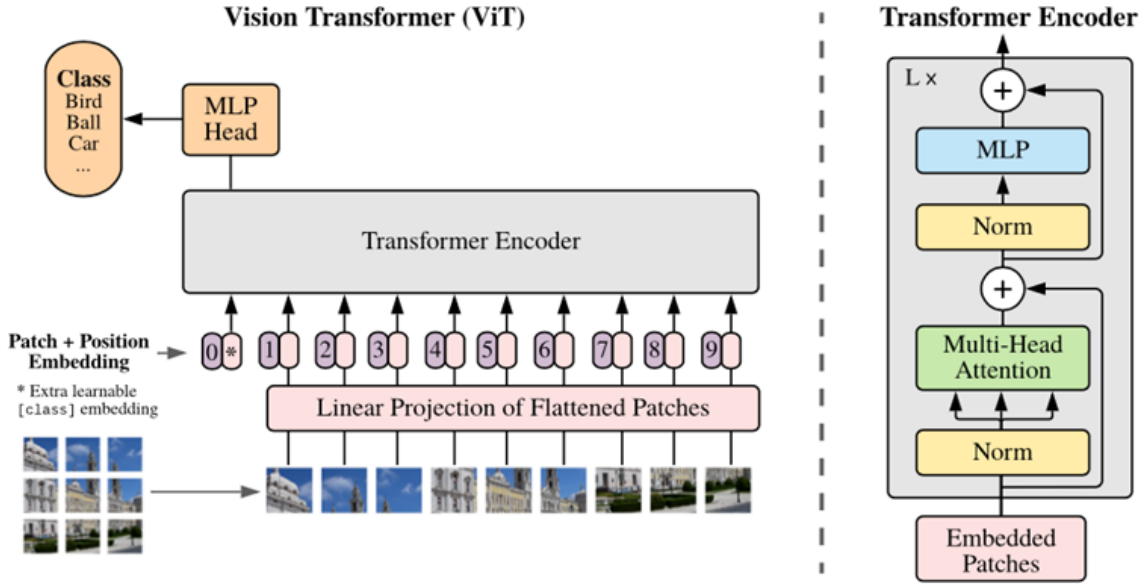
Sigmoid fucntion을 원래 썼다가 ReLu로 대체하게 된 이유 중 가장 큰 것이 gradient vanishing 문제이다. Simgoid function은 0에서 1사이의 값을 가지는데 gradient descent를 사용해 수행시 layer를 지나면서 gradient를 계속 곱하므로 gradient는 0으로 수렴하게 된다. 따라서 layer가 많아지면 잘 작동하지 않게 된다.

ReLU는 max(0, x) 함수이다. 계산 효율이 좋고 gradient vanishing 문제가 없다.

GELU는 x*sigma(1.702x)이다.

ViT

BERT, GTP 는 모두 text를 처리하는 언어 모델이고, ViT는 image를 처리하는 모델이다. Vision Transformer BERT와 거의 같은 architecture를 가진다. image를 패치단위로 쪼개서 단어 넣듯이 넣어준다. image는 2D이기 때문에 1자로 펴서 넣어준다.



GPT

Generative Pre-training 의 약자로 gernerative가 붙은 이유는 다음에 올 적절한 토큰을 생성하는 언어 모델이기 때문이다.

GPT는 transformer의 decoder 부분을 잘라서 사용한다.

decoder 부분 맨 아래에 있는 masked self-attention 부분 때문인데 masked self - attention은 input에 대해서 각 input이 전의 것만 참조 할 수 있도록 만든 것이다. 예를 들어, i am a boy 가 input으로 들어왔다고 가정하면 am은 참조할 때 i 밖에 참조를 못하는 것이다.

이러한 부분 때문에 GPT는 전반적으로 bidirectionality가 필요한 task에서 안좋은 성능을 가진다. 예를 들어 빈칸채우기나 전체 글을 읽고 짧은 답을 내는 task가 있다.

GPT1

GPT1에서는 다양한 special token을 활용해, fine-tuning의 성능을 극대화 시킨 모델이다. 문장 끝에 넣어준 토큰을 활용해 원래 원하는 downstream task의 query 벡터로 활용 된다. 같은 transformer 구조를 별도의 학습없이 여러 task에서 활용할 수 있다는 장점이 있다. 같은 맥락으로 수행하고자 하는 task에 대한 데이터가 얼마 없을때 pre-training 된 데이터를 fine tuning 부분에 자연스럽게 전이 학습 시킬 수 있다.

처음에 unsupervised pre-training을 하고, supervised fine-tuning을 한다.

GPT2

Webtest라고 불리는 수백만의 webpage로 외부적인 감독 없이 QA, 기계 번역, 문맥읽기, 요약 같은 걸 하도록 만들었다.

머신 러닝 은 자신들이 배운것으로 일을 잘하도록 설계되었지만 data distribution이 약간만 바뀌면 불안정해진다. 현재 system은 좁은 일정부분에 초점이 맞춰져 있고, GTP-2는 더 general한 system을 만들었다.

log-linear 방식의 task에서 performance를 높이기 위해서는 capacitiy of language model을 늘리는게 중요하다.

GPT-2는 1.5B parameter Transformer이다. zero-shot setting하는 dataset에서 7/8의 결과를 얻었다.

Masked language system을 만드는 주된 방법은 필요한 업무에 맞는 dataset을 모으는것이다. 하지만 model caption, reading comprehension, image classifier 같은 것들은 다양한 input 때문에 단점이 명확히 드러났다.

General 하게 만들려는 시도가 전에도 있었는데 GLUE,decaNLP가 있다.

이때까지 best는 pre-training과 supervised fine-tunning을 같이 쓰는것이었다.

GPT-2와 GPT-1의 차이점

GPT1은 단일 도메인의 데이터 셋, GPT2는 여러 도메인의 데이터셋이다.

GPT2는 pre-training만 사용해 zero-shot learning을 했다.

대용량 데이터 set WebText에 학습시켰고, task를 조건화한 unsupervised multitask learning으로 여러 도메인에서 활용할 수 있도록 했다.

Layer normalization이 sub-block의 input으로 옮겨졌고, 추가적인 layer normalization이 마지막 self- attention block에 생겼다. (archiecture)

Model depth에 residual path 쌓는걸 책임지는 modified initialization 이 사용되었다.

residual layer 을 $1/\sqrt{N}$으로 scale 해줬다. N은 residual layers 숫자다. 이것을 해서 깊은 layer일 수록 weight parameter를 작게 해서 깊은 layer의 역할이 줄어들도록 구성했다. Vocabulary 는 40000에서 50257이 됐고 context size 는 512에서 1024 batchsize는32에서 512로 바뀌었다.

Task 조건화

언어는 자연적인 순서를 가지고 있기 때문에 $p(s_n|s_1, ..., s_{n-1})$로 symbol들의 조건부 확률을 곱해서 구한다. p(x)를 이렇게 구하면 장점이 $p(s_{n-k}, ..., s_n|s_1, ..., s_{n-k-1})$ 같은 어떠한 형태의 확률도 다 구할 수 있다. self-attention architecture가 이런 확률을 계산하는데 좋다.

어떤 하나의 task를 배우는 rh 학습하는 것을 p(output | input) 으로 표현 가능하다.
general system은 많은 다른 tasks를 수행할 수 있어야 하기 때문에 p(output | input,task)로 modeling 되어야 한다.

언어는 task, input, output을 symbols의 sequence로 모두 유연하게 특정할 수 있다. 예로 (translate to french, english test, french text), (answer the question, document, question, answer) MQAN(multi QA network) 가 이런 형식의 포맷을 가지고 있는 예로 여러가지 다른 task를 수행했다. (In-text-learning)

또한 다양한 NLP task를 QA task로 통합했다.

위에서 말한 task를 조건화한 unsupervised multitask learning 이다.

Training Dataset

대부분 전의 연구는 text의 single domain만 받고 일을 했는데 이 연구팀은 다양한 domain과 context를 담은 다양한 dataset을 만들려고 했다. 사람들에 의해 엄선된 자료만 모았다. 다양한 사이트에서 3개이상 추천을 받은 링크만 썼다. Dragnet, Newspaper content extractors를 결합해 text를 추출했다. 그중에 wikipedia 자료는 common data source이기 때문에 overlapping 하는 것을 막기 위해서 모두 없앴다고 한다.

Input (BPE,Byte Pair Encoding)

Unicode strings을 UTF-8 bytes의 sequence로 쪼개는 것은 큰 dataset에 word-level LM 보다 좋지 않다.

Byte Pair Encoding(BPE)는 원래 character와 word level 사이의 실용적인 모델인데 종종 reference BPE implementation이 byte sequence로 안되어 있고 unicode로 되어있다. 이런 uni code symbol로 string을 만드려면 13만 이상의 base vocabulary가 있어야 한다. Byte-level version은 base vocabulary가 256이면 된다.

하지만 byte-level version으로 하면 다른 빈도수 많은 단어들 dog같은 경우 dog. dog! dog? 같은 쓸모없는 용량차지를 하기도 하는데 이걸 피하기 위해서 BPE를 character category를 넘는 merging을 떨어뜨려 놨다(dog만 저장). 이걸로 byte-level 접근의 generality로 word-level LM의 장점을 결합하게 해줬고, Unicode string에 확률을 배정할 수 있기 때문에 어떤 dataset이라도 GP-2를 평가할 수 있게 되었다.

Input 부분에 task에 대한 설명도 같이 들어간다.

결과

Summarization, translation에서 좋지 않은 성능이 나왔지만 나머지 성능은 괜찮고 QA, Language Modeling에서 좋은 성능이 나왔다.

Translation은 용량 부족으로 다른 언어의 corpus가 작았는데 그것을 감안하면 나쁘지 않다.

Generalization VS Memorization

Dataset에서 train set 과 test set이 데이터에서 겹치는 경우가 있어서 그것이 memorization이 아닌가에 대한 걱정이 있다. overlap을 제거하는 방법을 써야한다.

GPT3

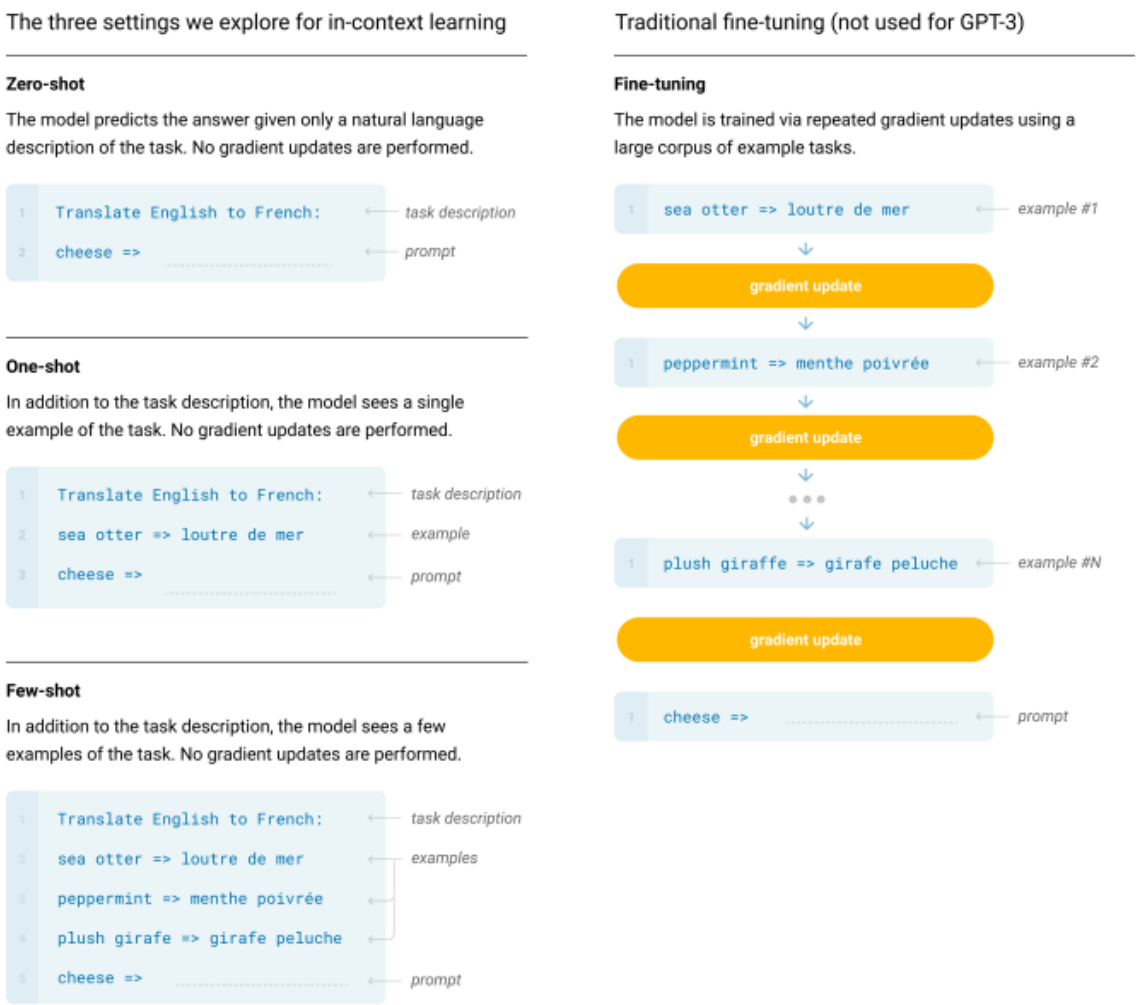
사람은 많은 supervised dataset이 필요없다. GPT2에서는 zero-shot을 이용했지만 GPT3는 사람이 어떤 일을 받을 때 예시가 1개나 몇개가 필요하다는 점을 빌어 one-shot 과 few-shot을 이용해 마찬가지로 성능이 증가할 수 있다는 것을 보여주려 했다.

하지만 여전히 fine-tunning없이 하는 방법은 있는것보다 많이 성능이 떨어지긴한다.

최근 transformer LM의 용량이 170억 parameter까지 올라왔다.

이런 증가가 downstream task에서 좋은 성능을 보였고, log loss(오차 log씌운거)는 scale이 증가하면 smooth해진다는 증거가 나왔다.

GPT3는 1750억 parameter로 train 됐다.



fine-tuning은 benchmark한 부분은 좋은 성능을 보이지만 모든 task마다 새롭고 큰 dataset이 필요하고 general하지 않다.

few shot은 model에 약간의 demonstration이 있다. weight를 update하진 않는다. few-shot 의 few 의 범위는 이 논문내에서는 10~100 사이의 값으로 나왔는데 최대 2048까지 가능하지만 늘린다고 좋은것만은 아니라고 한다.

zero-shot 은 model size에 맞게 steadily 좋아지는데 few-shot은 더 가파르게 좋아진다.

Arichitecture

architecture는 GPT2와 거의 같다.

차이점은 transformer의 층에서 dense attention 과 locally banded sparse attention pattern을 번갈아 썼다는 것이다.

원래 attention이 n*n matrix를 써야해서 메모리를 많이 먹었는데 각각의 출력 중에서 입력들의 일부 서브셋만 계산하는 것으로 attention pattern을 만들 수 있다고 한다. 그 방법으로 메모리 사용을 좀 줄인것이다.

Total compute 부분에서 GPT가 크기대비 가장 좋은 petaflop/s-days 를 받았다. (petaflop → 1초당 1000조번의 수학 연산처리)

Training dataset

GPT3 도 2와 유사하게 training dataset을 손봤는데 CommonCrawl에 있는 데이터를 high-quality reference로 한번 filtering 하고, 중복되는 부분을 없애고, 다른 high-quality refererence corpora를 추가했다.

Training mixed weight를 썼는데 본인들이 중요하다고 생각하는 dataset(higher quality dataset)에 더 weight를 많이 두고 섞었다.

Memorization

실험에서는 test에 쓸 downstream task가 contamination 당하지 않도록 미리 데이터에서 지우는 과정을 거쳤다. 나중에 질문할 내용들이 그대로 dataset에 있으면 learning했다고 보기 어렵고 memorization한 것이기 때문이다.

결과

GTP2 에서는 용량 문제 때문에 multilingual collection을 할때 거의 영어만 추출해서 썼었고 french text는 10mb만 training했었음에도 불구하고 번역을 하는 가능성을 보여줬었다.

GTP3에서는 용량을 많이 늘렸기 때문에 여전히 영어가 93% 다른 언어가 7%이지만 그나마 더 많은 다른 language에 대해 training이 가능했고, 실험을 할 때 GTP3의 가능성을 보기 위해 German와 Romanian을 추가했다고 한다. 하지만 역시 다른 languages의 비율이 부족하고, paired example을 많이 만들지 못했기 때문에 zero,one,few shot 들의 증가 비율이 크게 다르지 않았다고 한다.

QA 관련해서 fine-tuned SOTA와 비교했을 때 one-shot과 few-shot이 더 높은 정확성을 가졌다.

Commonsense reasoning에서는 한가지 dataset에서는 SOTA를 갱신했지만 나머지 부분에서는 낮게 나왔다.

Reading Comprehension에서는 전반적으로 모두 낮게 나왔다.

한계

GPT는 기본적으로 masked self attention을 사용하는 구조이기 때문에 unidirectional하다 따라서 전체적으로 bidirectional한 능력이 필요한 곳에서 약한 모습을 보였다. 빈칸 채우기나 전체 글을 읽고 짧은 답을 내는 task들이 그렇다.

GPT2와 비교했을때 질적, 양적으로 모두 발전했지만 여전히 몇몇 task에는 좋지 못하다.

문자 합성에서 전반적인 질은 높지만 가끔 스스로 반복하고, 연관성을 잃고, 서로 모순되는 등의 문제가 발견된다.

common sense physics에 약하다. “If I put cheese into the fridge, will it melt?” 같은 질문에 약하다고 하고, 독해 부분에서도 약하고 comparison task에서는 context learning 성능이 suite benchmark보다 잘 안나온다.

GPT3도 fine-tuning이 가능하지만 큰 bidirectional model보다는 안좋다.

더 기본적인 한계는 pretraining objective이다.

큰 pretrained language model은 다른 domain에서 쓸 수 없다. (video, real-world interaction)

따라서 self-supervised prediction 말고 새로운 approach가 필요하다.

pretraining 하는데 sample efficiency가 매우 안좋다.

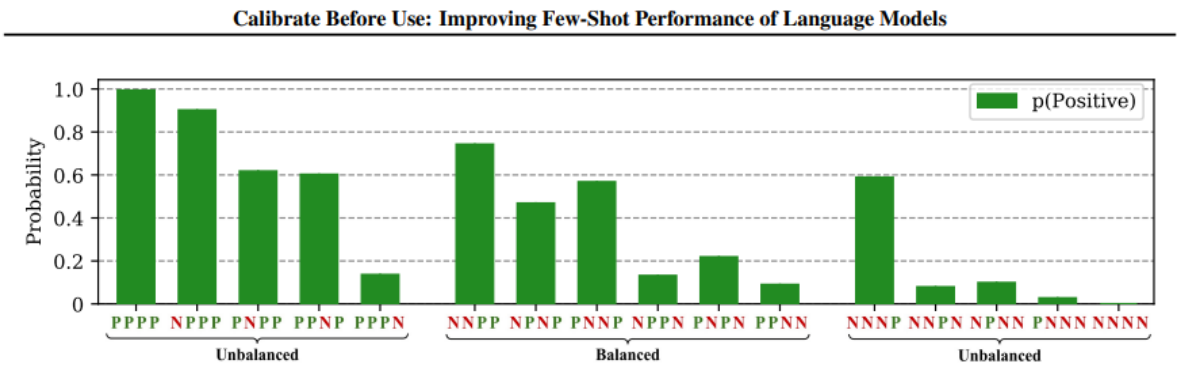
모든 deep learning system의 한계인 결정을 쉽게 해석할 수 없다.

Biased data를 계속 유지한다.

Calibrate Before Use: Improving Few-Shot Performance of Language Models

이 논문에서는 사용전 calibrate 해서 GPT-3의 성능을 30%가량 향상 시켰다고 한다.

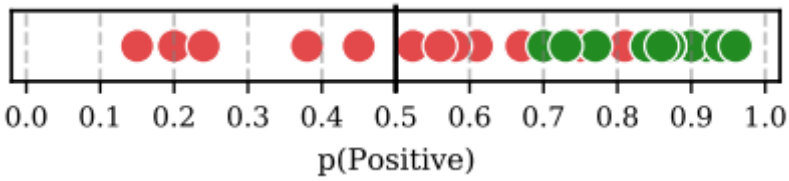
GPT-3를 few-shot learning을 시키는데 이 few-shot learning이 불안정하다는 것이다. few-shot learning이 포맷이나, training example, 그리고 training examples의 순서가 정확도의 영향을 끼친다는 것이다. 예를 들어 감정 분석에서 순서를 바꾸는 것이 정확도가 54%~ 93% 차이가 심하다고 한다.



Recency bias, common token bias등이 있는데 이름에서 유추할 수 있듯이, prompt가 어떤 것으로 끝나느냐에 따라서 결과 값이 바뀌고, pre-training data에 어떤 단어가 더 자주 나왔느냐에 따라서 결과 값이 바뀐다고 한다.

Majority label bias는 prompt에 자주나오는 답으로 답을 내도록 편향된다는 것이다.

이런 bias를 calibrate하는 방법은 마지막을 N/A값을 넣는것이다. 이 값을 넣었을때 모든 답이 같은 답이 나오도록 content-free하게 만들어주는 것이다.



random하게 값을 넣었을때 전체적으로 positive한 결과에 치우쳐져있는것을 알 수 있다.

$$\hat{\mathbf{q}} = \text{softmax}(\mathbf{W}\hat{\mathbf{p}} + \mathbf{b}),$$

로 새로운 확률 q를 구할 수 있는데 여기서 W는 weight matrix이고, p는 원래 확률, b는 bias vector이다.

p는 각각 label 이름과 연관되어 있다. 일반적으로 하기 위해서 p를 첫번째 token의 확률의 집합으로 만들었다. W를 diagonal한 matrix를 썼다고 한다.

W,b를 구하기 위해서 쓰는 방법이 위에 잠깐 언급했던 마지막에 N/A값을 넣는것이다.

Input: Subpar acting. Sentiment: Negative
Input: Beautiful film. Sentiment: Positive
Input: N/A Sentiment:

이런 데이터가 들어오면, 원래는 P와N이 50%확률로 나와야 하지만 model의 bias 때문에 실제로는 61.8%로 P가 나온다고 한다. 이것을 이용해 content-free input의 확률로 p_{cf} 로 설정해 W를 diag$(p_{cf})^{-1}$로 설정하고 b를 all zero 벡터로 설정했다고 한다.