Attention is all you need

트랜스포머

- 기존 인코더 디코더 발전시킨 딥러닝 모델
- RNN을 사용하지 않음
- 기계번역에 있어서 기존 RNN기반 인코더 디코더보다 학습이빠르고 성능이 좋음
- 영어를 독일어와 프랑스어로 번역하는 과정에서 학습이 빠르고 성능 좋은것을 확인 가능

어떻게 더 빠른가?

병렬화(Parallelization): 일을 최대한 한번에 처리

- RNN이 순차적으로 계산하여 입력된단어를 인코딩 하는 반면, 트랜스포머는 한방에 처리
- RNN
  - 인코더: 순차적으로 상태값을 계산해 문맥 벡터로 사용
  - 디코더: 문맥벡터를 기반으로 입력된 문장을 번역.
    end signal까지 계속해서 반복
  - 문맥벡터가 고정된 크기라서 책과 같이 긴 문장이 있는 경우에는 번역결과가 엉터리가 되는 경우 많음
- Transformer이전
  - 인코더: 순차적으로 상태값 계산된 후 고정된 크기의 문맥벡터 사용하지 않고, 대신 단어를 하나씩 번역할때마다 동적으로 인코더 출력값에 어텐션 메커니즘을 수행해서 긴 문장의 번역 능력이 좋음.
  - 하지만 여전히 RNN셀을 순차적으로 수행해 느림
  - RNN을 대신할 빠르고 성능 좋은 방법이 고민

attention is all you need

- RNN의 순차적인 계산은 트랜스포머에서 행렬곱으로 한번에 처리됨
- 트랜스포머는 한번의 연산으로 모든 중요 정보를 인코딩
- 가장 큰 특징은 RNN을 성공적으로 인코더 디코더에서 제거한 것
- 디코더에 번역과정은 기존 인코더 디코더 방법과 동일하게 스타트~엔드 사인까지 번역을 함. 이 대목에서 트랜스포머는 기존 인코더 디코더의 컨셉을 간직하고 있다는 것을 확인할 수 있음.기존 인코더 디코더의 컨셉을 간직하되 RNN을 없애서 학습시간을 단축했고, 어텐션 뿐만아니라 더 많은 스마트한 기술들을 함께 제공하여 성능을 올림.
- 자연어 처리에서 문장을 처리할 때 실제 단어의 위치 및 순서는 상당히 중요함. RNN이 자연어 처리에 많이 활용된 이유도 단어의 위치와 순서 정보를 잘 활용하는데 있기 때문임. 그렇다면 RNN이 없는 트랜스포머는 어떻게 위치 및 순서정보를 활용할까?: positional encoding

positional encoding

- 인코더 및 디코더 입력값마다 상대적인 위치정보를 더해주는 기술
- 사인과 코사인 함수를 활용한 positional encdoing 사용
- 장점: 항상 positional encoding이 값이 -1~1, 학습 데이터 중 가장 긴 문장보다 더 긴 문장이 실제 운영중에 들어와도 포지셔널 인코딩이 에러없이 상대적인 인코딩인 값을 줄 수 있다는 것
- 각 단어의 word embeding에 positional embeding을 더해준 후에, 셀프 어텐션 연산을 해주어야함

self attention

- 인코드에서 이루어지는 어텐션 연산을 self attention이라고 함
- 쿼리, 키, 밸류 라는 개념, weight matrix로 딥러닝 학습과정을 통해서 최적화됨
- word embeding은 벡터이고 실제 한문장은 행렬이라고 할 수 있음
- 행렬은 행렬과 곱할 수 있으므로 각 문장에 있는 단어들의 쿼리, 키, 밸류는 행렬 곱을 통해 한 번에 구할 수 있음
- 쿼리, 키, 밸류는 벡터의 형태
- 쿼리(Wq): 어떤 단어와의 상관관계를 구할때 쿼리를 그 단어의 키값에 곱해줌. 쿼리와 키값을 곱한값을 attention score라고 함. 쿼리와 키가 둘다 벡터이므로 닷프로덕트 곱하면 숫자로나옴. 숫자가 높을수록 단어의 연관성이 높고, 낮을수록 연관성이 낮다라고 생각하면 됨. attention score를 0부터 1까지 확률개념으로 바꾸기 위해 softmax적용. 그 전에 score를 키벡터의 차원수의 루트 값으로 나누어줬는데, 논문에 따르면 키 벡터

의 차원이 늘어날수록 닷프러덕트 계산시 값이 증대되는 문제를 보완하기 위해 조치를 취했다고 함.
softmax의 결과값은 키값에 해당하는 단어가 현재 단어에 어느정도의 연관이 있는지 나타냄. 각 퍼센테이지를 각 벨류에 곱해줌. 곱해준 결과 연관성이 별로 없는 value는 희미해짐.
최종적으로 softmax*value를 모두 더해주면 단순한 단어가 아닌 문장 속에서의 단어가 가진 전체적인 의미를 지닌 벡터라고 간주할 수 있게 됨.
입력문장 전체는 행렬로 표시할 수 있음. 모든 단어에 대해 모든 attention 연산은 행렬곱으로 한번에 처리할 수 있음. RNN처리했다면 순차적으로 계산했어야 했는데..
⇒ 이것이 attention활용한 병렬처리의 가장 큰 장점

- 키(Wk)
- 밸류(Wv)
- 트랜스포머는 attention layer 8개를 병렬로 동시에 수행함.
  ⇒ 병렬 처리된 attention layer를 multi head attention이라고 함
  ⇒ 기계 번역에 큰 도움을 줌
  ⇒ 연관된 정보를 다른 관점에서 수집해서 한 단어에 포커스하는 점을 보완

인코더

- 단어를 word embeding으로 전환한 후에 positional encoding을 적용 → multi head attention에 입력 → 여러개의 결과값들을 모두 이어붙여 또다른 행렬과 곱해, 최초 word embeding과 동일한 차원을 갖는 벡터를 출력하도록 되어있음
- 출력벡터의 차원의 크기가 입력벡터와 동일함
- word embeding + positional encoding
  포지셔널 인코딩이 많이 손실될수 있음
  → residual connection으로 입력값을 다시한번 더해줌
- Encoder Layer의 출력벡터와 입력벡터의 차원의 크기가 동일
  ⇒ 인코더 레이어를 여러개 붙여서 사용할 수 있다는 의미
- 트랜스포머 인코더는 실제 인코더 레이어를 6개 연속으로 붙인 구조. 각각의 인코더 레이어는 서로의 모델 파라미터 즉 가중치를 공유하지 않고 따로 학습시킴. 트랜스포머 인코더의 최종 출력값은 6번째 인코더 레이어의 출력값이 됨.

디코더

- 인코더와 상당히 유사하게 생김. 인코더처럼 6개의 동일한 레이어로 구성
- 기존 인코더, 디코더의 작동방식과 같이 순차적으로 단어를 출력

- 디코더 역시 어텐션 병렬 처리를 적극 활용. 디코더에서 현재까지 출력된 값들에 어텐션을 적용하고 또한 인코더 최종출력값에서 어텐션이 적용됨
- 인코더와의 차이점
    1. masked multi head attention 디코더 레이어에서 지금까지 출력된 값들에만 어텐션을 적용하기 위해서 붙여진 이름. 출력되지 않은 미래의 단어에 어텐션을 적용할 수 없기 때문.
    2. multi heada attention layer 인코더처럼 키, 밸류, 쿼리로 연산을 하는데 인코더의 멀티 헤드 어텐션 레이어와 가장 큰 차이점은 디코더의 멀티 헤드 어텐션은 현재 디코더의 입력값을 쿼리로 사용하고 인코더의 최종 출력값을 키와 밸류로 사용한다는 것. 디코더의 현재 상태를 쿼리로 인코더에 질문하고, 인코더 출력값에서 중요한 정보를 키와 밸류로 획득해서 디코더에 다음 단어에 가장 적합한 단어를 출력하는 과정
    3. hidden forwoard layer : 최종 값을 벡터로 출력. 이 벡터를 어떻게 실제 단어로 출력할 수 있을까? 실제 단어로 출력하기 위해 디코더 최종 단에는 linear layer 와 softmax layer가 존재함. linear layer은 softmax의 입력값으로 들어갈 로짓을 생성. softmax는 모델이 알고있는 모든단어들에 대한 확률값을 출력, 가장 높은 확률을 지닌 값이 다음 단어가 됨.
       트랜스포머는 최종 단계에도 Label Smoothing 기술을 통해 모델의 퍼포먼스를 업그레이드 시킴. 0또는 1이 아닌, 정답은 1에 가까운값 오답은 0에 가까운 값으로 변화를 주는 기술. 모델 학습시 모델이 학습데이터에 치중하여 학습하지 못하도록 보완하는 기술.

       레이블이 같은 입력값인데 다른 출력값들이 학습데이터에 많은 경우(Noisy한 경우) label smoothing은 큰 도움이 됨

* BLUE 점수: 자동 번역 및 동일한 소스 문장에 대해 사람이 만든 참조 번역 간의 차이를 측정한 점수

* 원핫인코딩: 각 속성을 분리하여 어떤 특성을 가지고 있는지 아닌지를 1 0으로만 표현하는 방법

BERT 논문 분석

Introduction

[1] BERT의 정의

> 💡 **BERT**
> **: B**idirectional **E**ncoder **R**epresentations from **T**ransformer
>
> 이름에서부터 직관적으로 알 수 있듯이, BERT란, Transformer의 자식 모델로, Transformer에서 Encoder만을 사용해서 양측으로 representation하는 모델을 말한다.

[2] ELMo vs GPT vs BERT

▼ Feature-based:

specific-task network에 pre-trainded language representation을 additional feature로 제공하는 방식

ex) ELMO는 backward/ forward를 따로 각각 학습시켜서 나온 representation 모델을 결합하는 **biLM(Bidirectional Language Model) 이다.**

다만, 이는 shallow bidirectional이며 왼→오, 오→왼을 단순히 concat해 구성된다는 점에서 양방향 representation인 **BERT**의 **deep directional**과 차이가 있다.

▼ fine-tuning:

특정 task를 수행하는 parameter식를 최대한 줄이고, pretrained parameter를 downstream task 훈련을 통해 조금 바꾸는 방식

ex) GPT(Generative Pretrained Transformer)는 Transformer의 decoder방식만 사용한 언어모델로, decoder는 한쪽 방향만(왼→오) 사용하기 때문에 Unidirectional problem에 사용된다.

BERT의 경우 pretrain을 통해 **양방향 task 수행 기능**을 향상시켰다는 점에서 앞선 두 모델과 차이가 있다.

Contribution

BERT는 GPT와 마찬가지로 별도의 라벨링없이 학습되는 unsupervised model이다. 다만, 기존의 왼쪽에서 오른쪽으로 읽어가면서 현재 위치에서 다음 단어의 위치를 예측하는 GPT 방식과 달리 encoder방식을 사용하는 BERT는 **모든 토큰을 한방에** 계산해 단어를 예측한다는 점에서 차이가 있다.

BERT task

BERT는 두가지 task를 수행하는데, 다음과 같다.

- Masked language model(MLM)
- Next sentence prediction(NSP)

[1] MLM

sequential하게 나가는 데이터를 forward/backward와 무관하게 임의의 순서를 배정해 특정 위치 해당하는 부분에 masking을 하고 prediction하도록 하는 모델을 말한다. 가려진 단어는 mask token으로 불리며, 단순히 문장의 단어를 랜덤하게 가리고 가려진 문장을 예측할 수 있도록 학습하면 된다.

다시 말해, 앞선 단어와 뒤의 단어를 모두 사용하되, 일정 부분 masking을 해서 task를 수행하는 것을 말한다.

[2] NSP

특정한 두 쌍의 sentence가 들어왔을 때 해당하는 sentence가 문맥적으로 살펴봤을 때, 실질적으로 다음에 등장했던 sentence인지 아닌 지를 학습한다.

BERT의 우수성

BERT는 한 문장의 입력값을 받아 출력하는 자연어 task뿐만 아니라 2문장의 문장을 받아 처리하는 질의 및 응답도 수행할 수 있다.

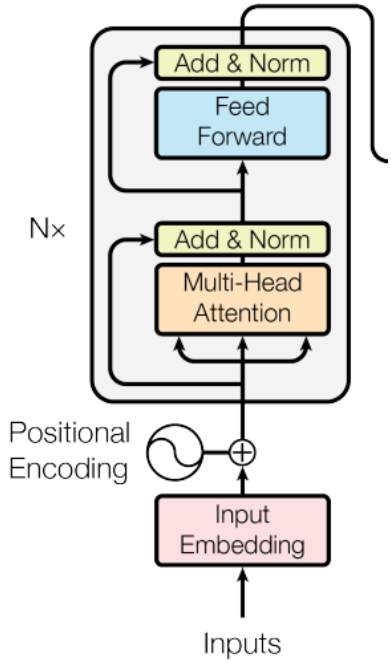
다시 말해, BERT는 2가지 방식으로 pretrained 모델을 가져다 하나의 단순한 layer를 쌓아서 SQuAD, question answering task, NER, Sentiment Anyalysis 의 task를 수행할 때,

BERT의 출시 기준으로 이전에 출시된 SOTA 모델을 모두 앞지르는 우수한 모델을 만들어 냈다는 점에서 우수성을 보인다.

→ 실제로 11개의 task에서 **state-of-the-art** 우수성을 입증했다.

BERT의 모델 architecture

muli-layer bidirectional transformer encoder



→ Only **encoder**

BERT: Input/Output

BERT가 다양한 downstream tasks(궁극적으로 BERT를 이용해 풀고자 하는 문제)를 다루기 위해,

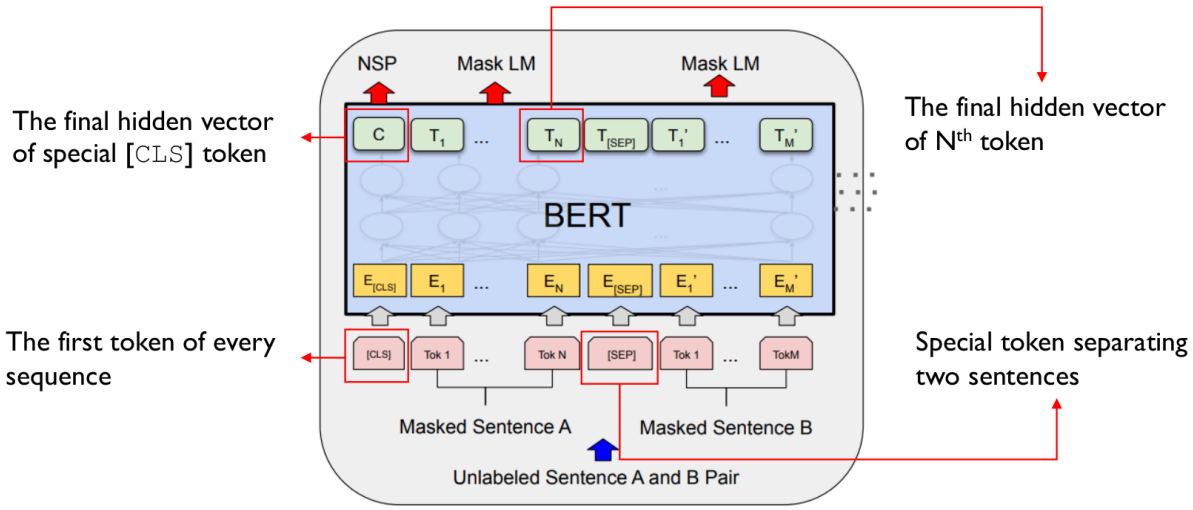
BERT의 Input representation은 두 가지 형태를 입력값으로 받을 수 있다.

하나의 토큰 sequence에서 하나의 문장과 QA와 같은 pair 문장를 분명하게 표현했다. 이 때, **Sequence**는 하나의 Sentence의 의미를 가질 수 있고, 두 개의 Sentence을 의미할 수 있는데, 일반적인 **문장**의 의미와 다른 의미를 지닌다. 앞선 sentence는 임의로 계속해서 이

어지는 문장을 말하며, 하나의 *Sentence* 안에 일반적인 의미의 문장이 하나만 있을 수도 여러개(packed)가 있을 수 있다.

최종적으로, 다음은 BERT가 입력값으로 받을 수 있는 두 가지 형태이다.

- Sentence can be an **arbitrary span of contiguous text(임의로 계속해서 이어지는 문장)**, rather than an actual linguistic sentence.
- Sequence refers to the input token sequence to BERT, which may be a single sentence or
  two sentences packed together

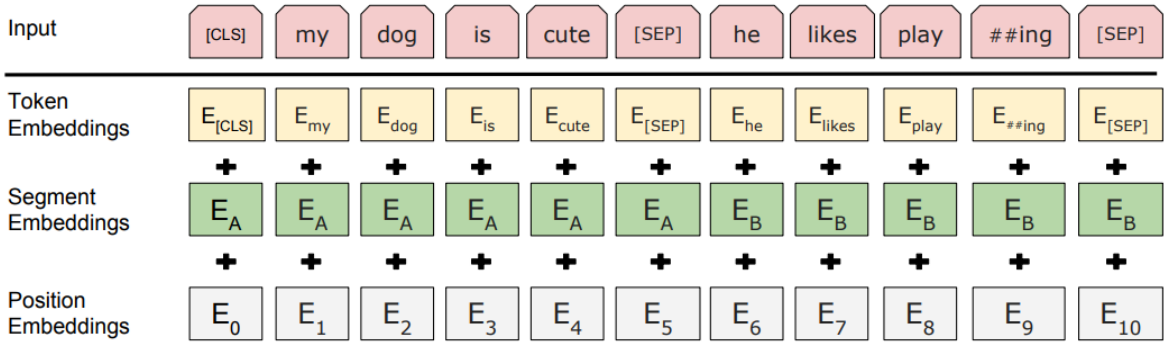


→ *BERT의 special token*

`[CLS]` : Classfication, 분류를 시작하기 전 선언으로 처음에 사용되는 토큰

`[sep]` : Seperation, 입력 데이터가 두 개인 경우(QA) 이 둘을 구분하기 위해 사용되는 토큰

→ 만일, 하나의 sentence로만 구성된 입력 데이터라면 맨 마지막에 위치함.



BERT에 입력되는 벡터는 세가지 embedding vector의 합으로 구성된다.

1. Token embeddings : 3만개의 단어로 이루어진 사전으로부터 가져온 단어 **토큰 임베딩 벡터**
2. Segment embeddings : `[sep]` 을 기준으로 Sentence의 위치를 표현하는 **세그먼트 임베딩 벡터**

   다시 말해, `[sep]` 을 기준으로 앞에 있는 sentence인지 뒤에 있는

   sentence인지 구분하는 역할을 함. 만약 하나 sentence로만 구성된

   입력 데이터라면 맨 마지막에 위치함.
3. Positional embeddings : 각 토큰이 어디에 위치하는 지 표시하는 **위치 임베딩 벡터**

Task #1 : Masked LM

단어 중 일부를 mask 토큰으로 바꾸며 그 양은 15%임.

15%에 해당하는 `[mask]` 토큰은 다음과 같이 치환됨.

- 80%에 해당하는 단어를 `[mask]` 토큰으로 바꿈
- 10%는 원래 단어로 둠
- 10%는 임의의 단어로 치환함

BERT는 이 mask 토큰을 맞추며 문맥을 파악하는 능력을 길러내게 됨.

Task #2 : NSP

QA의 성능을 높이려면, 문맥을 파악해 A,B sentence의 차이를 파악해야함.

즉, 어떤 문장이 끝났을 때, 언제 끝나서 구별되는 지 파악해야함.

BERT는 단순히 `[CLS]` 와 `[SEP]` 를 이용해 문장을 구별함. 구분된 문장을 하나로 이어준 뒤, 새로 시작한 `[CLS]` 인지 문장의 끝을 표현하는 `[SEP]` 인지를 파악해 문장을 구별함

학습은 방법은 다음과 같은 두 개 유형의 데이터 셋을 이용해 학습함

- 50% :sentence A,B가 실제 문장이 주고 받는 이어지는 sentence임
- 50% : sentence A,B가 corpus에서 random으로 뽑힌 이어지지않는, 무관한 두 문장

Pre-training BERT

→ pretraining data로 약 8억 단어로 구성된 Book corpus와 25억 단어로 구성된 wiki corpus 사용

최대 토큰 길이	512
배치 사이즈	256
Optimizer	Adam
learning rate	1e-4
beta1	0.9
beta2	0.999
L2 weight decay	0.01

- 매 layer마다 0.1의 dropout 사용
- activation function: gelu
- loss function: sum of the mean MLM likelihood + mean NSP likelihood
- BERT_BASE : 4days by 16 TPU ,
  BERT_LARGE : 4days by 64 TPU

L : # of layer, i.e transformer의 block을 얼마나 사용한 것인지

H : size of hidden node

A : # of Multi-head attention의 heads

[1] BERT_BASE

- L = 12, H = 768, A = 12
- Total parameters = 110M(1억천만)

→ ELMO가 나오고 GPT가 나왔을 때, GPT와의 성능을 비슷한 복잡도, 비슷한 parameter 상에서의 비교를 위해 위와 같이 parameter설정

같은 하이퍼파라미터상에서도 encoder방식으로 양방향 문맥을 파악하는 BERT를 사용하면 더 나은 성능을 보임을 입증하고자 사용한 것으로 보임.

[2] BERT_LARGE

- L = 24, H = 1,024 , A = 16
- Total parameters = 340M(3억 4천만)

Conclusion

BERT는 단반향의 architecture만이 사용 가능했다는 한계점을 보완해, Bidirectional architecture의 확장을 이뤘다는 점에서 의의를 지닌다.

BERT를 활용한 연구 예시

Downstream task : NER, RE

기존 Pretraining된 PubMedBERT 사용

- Pretraining (Masked Language Model)
  - Training data : 새로 수집한 cell line 관련 data
- Finetuning
  - NER : 기존 데이터셋 활용하여 training
  - RE : 새로 annotation하여 training

entity type : Protein, DNA, RNA, cell type, cell line, disease, drug
Relation type : regulates / interacts / covaries / downregulates …

참고자료

https://yngie-c.github.io/nlp/2020/07/04/nlp_bert/