BERT 논문 리뷰

윤세환

목차

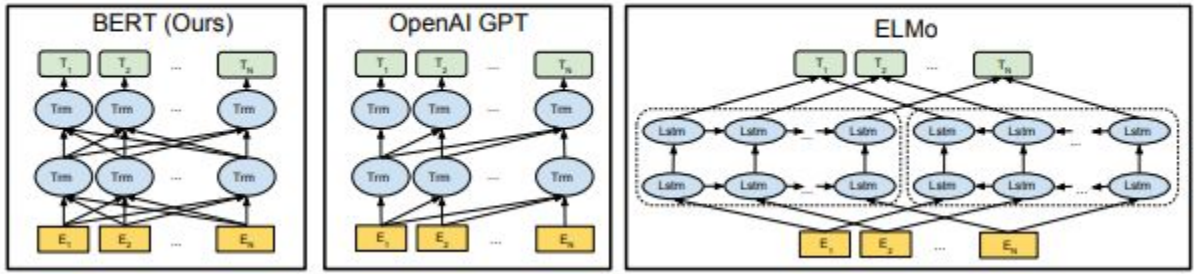
- 등장배경
- 타 모델과의 구조 간단 비교
- 전체적인 모델 구조 (input embedding, encoder)
- 핵심개념 (NSP, MLM)
- pre-train
- 성능비교

등장배경

- pre-train된 언어 모델을 fine-tuning해서 사용하는 구조가 효과적이었고, 이런 구조를 채택하는 모델인 EMLo와 GPT가 좋은 성능을 보여줌

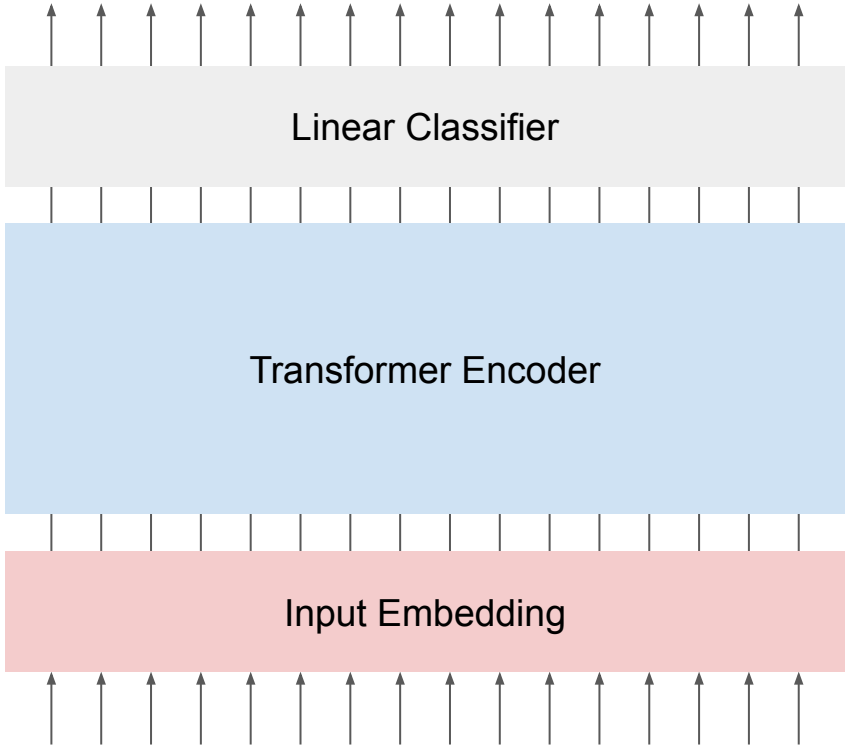
- 하지만 위 2개 모델의 경우 단방향 모델을 사용하는 방식이었고 이 방식은 모델이 문맥을 이해하는데 있어 한계가 있다고 주장하여 (특히 GPT), 양방향을 한번에 모두 고려하는 모델인 BERT(**Bidirectional** Encoder Representations from Transformers)을 제시함

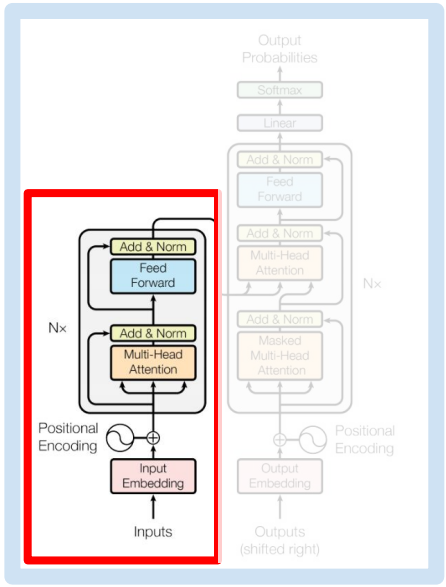
타 모델과의 구조 간단 비교

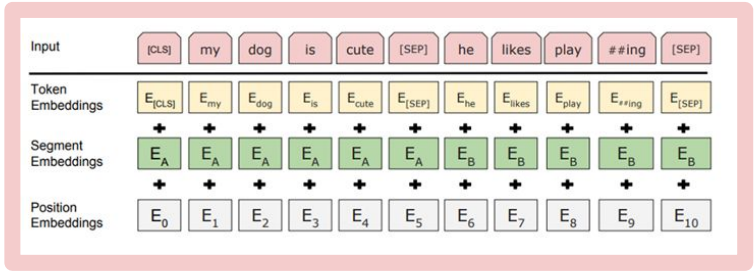


- 단방향을 고려하는 GPT,
  양방향을 모두 반영하긴 하지만 2개의 분리된 단방향 모델을 병합해서 사용하는 ELMo와는 달리,
  BERT의 경우 한번에 양방향의 데이터를 반영하고 있다.

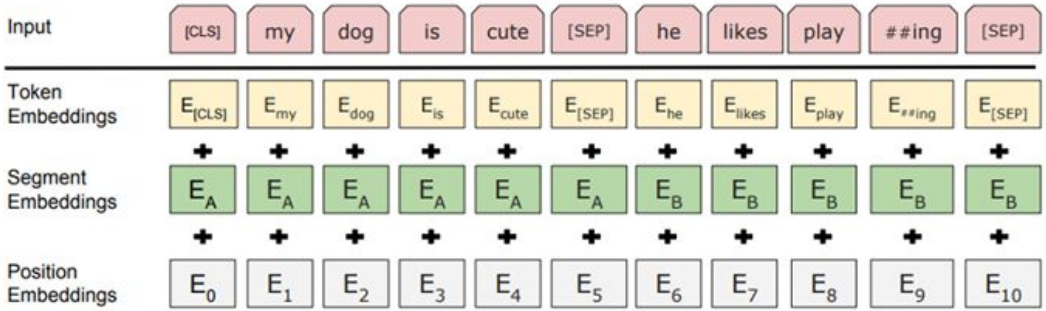
전체적인 모델 구조


Linear Classifier
Transformer Encoder
Input Embedding


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


Input
[CLS] my dog is cute [SEP] he likes play ##ing [SEP]
Token Embeddings
Segment Embeddings
Position Embeddings

Input Embedding



[CLS] 토큰 : 입력의 첫 번째 토큰

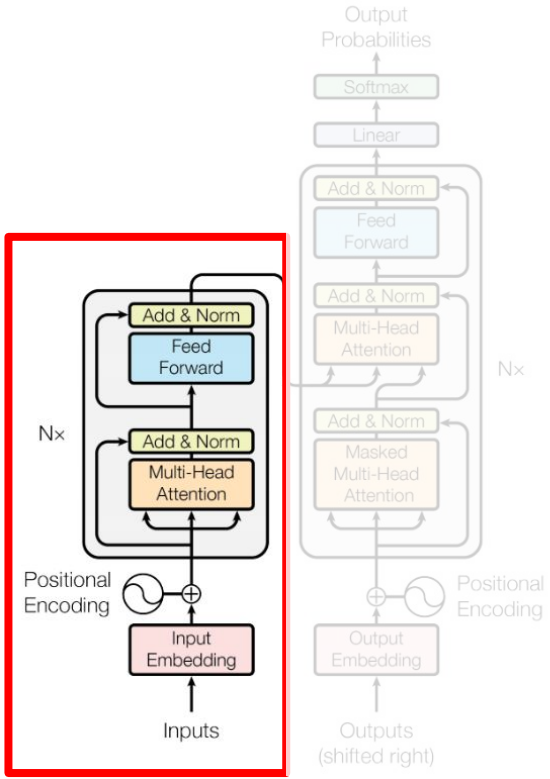
[SEP] 토큰 : 한 문장의 종료를 의미하는 토큰

- Token Embeddings (“my” -> [0.01, 0.51, 0.34, 0.4, …])

- Segment Embeddings
  BERT의 Pre-train과정에서 NSP(이후에 언급)을 위해서 2개의 문장을 받게 되는데, 이 2개의 문장을 구분하기 위해 수행

- Position Embeddings
  Position Encoding을 별도의 embedding layer를 사용해 수행

Encoder Block



- Transformer 구조 중, Encoder 부분만을 사용
- BERT-base
  - encoder의 개수 : 12
  - 히든 레이어 차원 : 784
  - self-attention의 multi-head 수 : 12
- BERT-large
  - encoder의 개수 : 24
  - 히든 레이어 차원 : 1024
  - self-attention의 multi-head 수 : 16

- BERT-base는 GPT와 매개변수 개수를 맞춘 모델

핵심 개념

- Next Sentence Prediction (NSP)
- Masked Language Model (MLM)

핵심 개념 - NSP

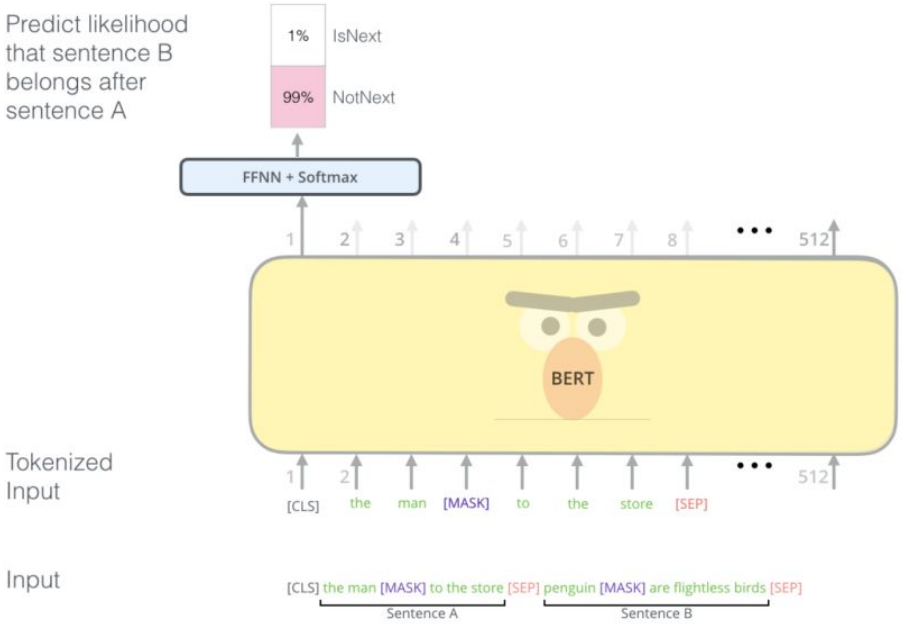
- 입력으로 받은 2 문장이 서로 이전/이후 문장으로 연결되는 관계인가?를 학습

  [예시1] 내일 팀배치를 위한 설문조사를 진행할 예정입니다. 신중하고 솔직한 답변으로 응해주시길 바랍니다. >> 두 문장이 자연스럽게 연결됨

  [예시2] 내일 팀배치를 위한 설문조사를 진행할 예정입니다. 원하는 어휘 크기에 도달할 때까지 이 단계가 반복됩니다. >> 두 문장이 서로 다른 맥락

- 2 문장이 서로 연결되면 [1, 0], 아니면 [0, 1]을 리턴하도록 학습 수행

핵심 개념 - NSP



- NSP를 학습할 때는, 문장의 첫 토큰인 [CLS]의 output값을 사용한다.

- 즉, [CLS]의 output에 적용하는 Linear Classifier의 최종 결과물의 차원은 2차원이며, 입력받은 2 문장이 서로 연결된다면 [1, 0], 아니라면 [0, 1]에 가깝게 결과값이 출력되도록 학습되도록 한다.

핵심 개념 - MLM (Masked Language Model)

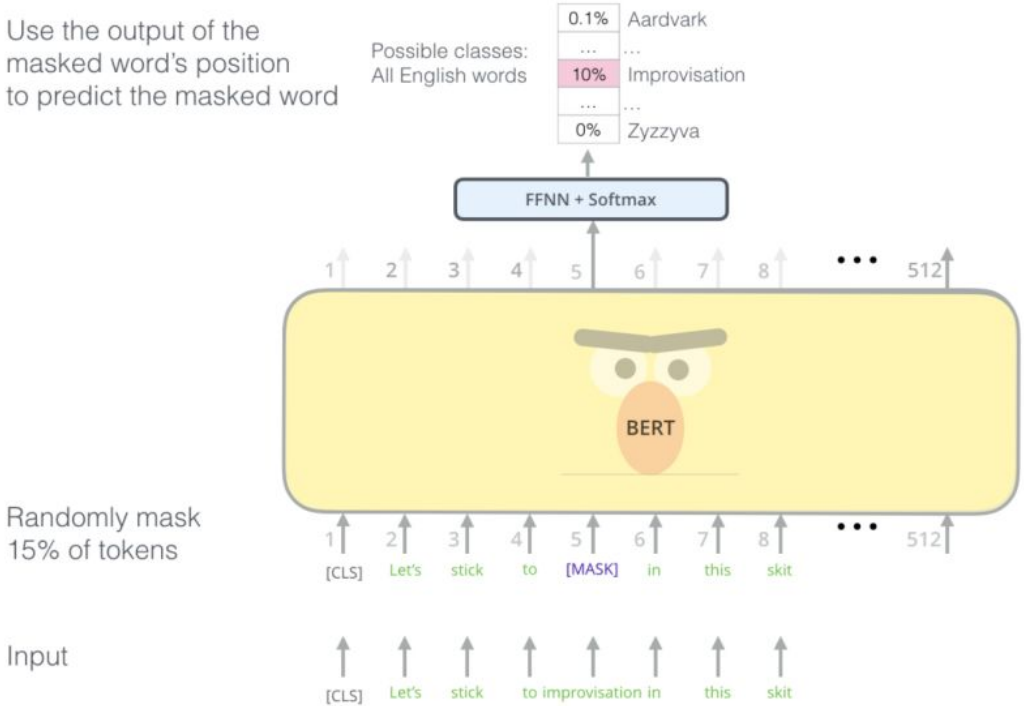
- 단일 방향 모델을 적용했을 때처럼, 다음 단어를 예측하는 방식은 양방향을 한번에 반영하는 모델에는 적용하기 어려움
- 따라서, 문장의 특정 단어를 마스킹하고, 해당 마스킹한 단어를 예측하는 방식으로 학습을 진행 (전체 중 15%)
- 단, 이 방식을 사용할 경우 pre-train과 fine-tuning간 학습 방식에 큰 차이가 발생할 수 있기에, 이를 완화하기 위해서 Masking을 다음과 같이 수행한다.
  - masking할 단어를 정했다면,
    80%의 확률로 해당 입력을 [MASK] 토큰을 마스킹한다.
    10%의 확률로는 아예 다른 단어로 대체하고
    10% 확률에는 단어를 수정하지 않는다.

핵심 개념 - MLM

[예시] “흥미로운 논문이 **하나** 있어서 가져와봤습니다.”에서 “하나"를 마스킹하는 경우

- 80% : 흥미로운 논문이 **[MASK]** 있어서 가져와봤습니다.
- 10% : 흥미로운 논문이 **사과** 있어서 가져와봤습니다.
- 10% : 흥미로운 논문이 **하나** 있어서 가져와봤습니다.

핵심 개념 - MLM


Use the output of the masked word's position to predict the masked word
Possible classes: All English words
0.1% Aardvark
... ...
10% Improvisation
... ...
0% Zyzzyva
FFNN + Softmax
1 2 3 4 5 6 7 8 ... 512
BERT
Randomly mask 15% of tokens
1 2 3 4 5 6 7 8 ... 512
[CLS] Let's stick to [MASK] in this skit
Input
[CLS] Let's stick to improvisation in this skit

Pre-training 과정 요약

- input Embedding을 사용하여 입력된 단어를 임베딩한다.
  (token embedding, segment embedding, positional embedding을 더한 값)

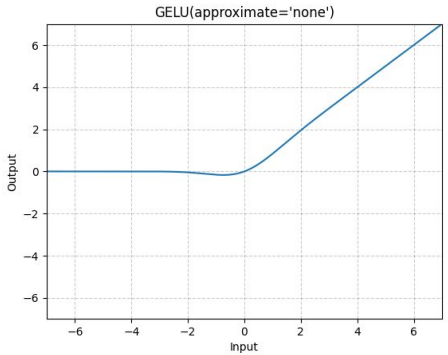
- input Embedding을 수행한 값을 Transformer의 encoder에 입력값으로 넣는다.

- 가장 첫 토큰인 [CLS]의 output의 경우 두 문장이 서로 연결되었는지 여부 (NSP) 학습을 위해 사용되고, 나머지 결과들 중 maksed된 위치의 결과값은 마스킹된 단어를 예측하는 MLM 학습을 위해 사용된다.

- 손실함수값은 NSP, MLM에서 발생한 값을 더해서 사용한다.

Pre-training 관련 정보

- 손실함수는 NSP, MLM 모두 Cross Entropy를 사용
- 모든 레이어에 p=0.1의 dropout을 적용
- 최적화 기법은 Adam을 사용했으며 하이퍼파라미터는 아래와 같음
  - learning rate : 1e-4
  - b1 = 0.9
  - b2 = 0.999
  - l2 weight decay = 0.01
  - 추가적으로 learning rate는 10000 step까지 천천히 1e-4까지 증가시켰다가 그 이후는 linear하게 감소시킴 (warm-up lear
- 활성화 함수는 gelu를 사용



성능 비교

System	MNLI-(m/mm) 392k	QQP 363k	QNLI 108k	SST-2 67k	CoLA 8.5k	STS-B 5.7k	MRPC 3.5k	RTE 2.5k	**Average** -
Pre-OpenAI SOTA	80.6/80.1	66.1	82.3	93.2	35.0	81.0	86.0	61.7	74.0
BiLSTM+ELMo+Attn	76.4/76.1	64.8	79.8	90.4	36.0	73.3	84.9	56.8	71.0
OpenAI GPT	82.1/81.4	70.3	87.4	91.3	45.4	80.0	82.3	56.0	75.1
$\text{BERT}_{\text{BASE}}$	84.6/83.4	71.2	90.5	93.5	52.1	85.8	88.9	66.4	79.6
$\text{BERT}_{\text{LARGE}}$	**86.7/85.9**	**72.1**	**92.7**	**94.9**	**60.5**	**86.5**	**89.3**	**70.1**	**82.1**

Table 1: GLUE Test results, scored by the evaluation server (https://gluebenchmark.com/leaderboard). The number below each task denotes the number of training examples. The "Average" column is slightly different than the official GLUE score, since we exclude the problematic WNLI set.[8] BERT and OpenAI GPT are single-model, single task. F1 scores are reported for QQP and MRPC, Spearman correlations are reported for STS-B, and accuracy scores are reported for the other tasks. We exclude entries that use BERT as one of their components.

성능비교 - MLM, NSP 적용 및 모델 크기

Tasks	Dev Set MNLI-m (Acc)	QNLI (Acc)	MRPC (Acc)	SST-2 (Acc)	SQuAD (F1)
$\text{BERT}_{\text{BASE}}$	84.4	88.4	86.7	92.7	88.5
No NSP	83.9	84.9	86.5	92.6	87.9
LTR & No NSP	82.1	84.3	77.5	92.1	77.8
+ BiLSTM	82.1	84.1	75.7	91.6	84.9

Table 5: Ablation over the pre-training tasks using the $\text{BERT}_{\text{BASE}}$ architecture. "No NSP" is trained without the next sentence prediction task. "LTR & No NSP" is trained as a left-to-right LM without the next sentence prediction, like OpenAI GPT. "+ BiLSTM" adds a randomly initialized BiLSTM on top of the "LTR + No NSP" model during fine-tuning.

Hyperparams				Dev Set Accuracy		
#L	#H	#A	LM (ppl)	MNLI-m	MRPC	SST-2
3	768	12	5.84	77.9	79.8	88.4
6	768	3	5.24	80.6	82.2	90.7
6	768	12	4.68	81.9	84.8	91.3
12	768	12	3.99	84.4	86.7	92.9
12	1024	16	3.54	85.7	86.9	93.3
24	1024	16	3.23	86.6	87.8	93.7

Table 6: Ablation over BERT model size. #L = the number of layers; #H = hidden size; #A = number of attention heads. "LM (ppl)" is the masked LM perplexity of held-out training data.

참고 링크

- bert 모델에 대한 시각 자료 : https://jalammar.github.io/illustrated-bert/
- positional encoding vs positional embedding : https://heekangpark.github.io/ml-shorts/positional-encoding-vs-positional-embedding
- bert input embedding : https://tinkerd.net/blog/machine-learning/bert-embeddings/
- WordPiece 토큰화 : https://wikidocs.net/166826
- 논문 링크 : [1810.04805] BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding