

# XAI : BERT Paper Review

## Abstract

BERT 논문이 나타나기 이전에도 언어 모델을 사전 훈련한 후에 task에 대해서 feature-based 모델을 만들거나 fine-tuning하는 방식이 인기를 끌었다. BERT가 나타나기 이전에 나온 모델은 GPT-1으로 많은 양의 데이터를 학습된 pre-trained model이며 fine-tuning을 통해 성능을 보장했기에 상당한 인기를 얻었다.

그러나 기존 모델들은 LSTM이나 RNN처럼 결국에는 문제를 해결하기 위해서 **문장을 학습할 때에 순차적으로(Left to Right) 읽을 수 밖에 없다는 문제점**을 지니고 있다. 단어 임베딩의 경우 Transformer를 사용해서 Attention을 통해 관계성을 잘 파악하도록 만들어낼 수 있지만, **결국에 예측을 해야할 때는 단반향으로 읽어서 예측해야 하기에 이전 토큰만 참조할 수 있다는 큰 단점**이 있다.

이러한 문제는 다음 문장에 대한 예측이나 문장의 빈칸에 대한 예측을 할 때에 상당히 치명적으로 다가오게 된다. 하지만 이것을 해결하기 위해 양방향으로 그냥 읽게 되면, 정답을 보고 예측하게 되므로 효과적으로 학습시킬 수 없다.

이것을 **해결한 방식이 BERT의 MLM 방식**이고 이를 통해 **Bidirectional Transformer 이용이 가능**해졌기에 줄여서 BERT라고 불리는 것이다.

이러한 BERT는 NLP의 11분야에서 모두 state-of-art 성능을 보였으며, NLP의 한 획을 그은 모델이 되었다.

## Feature based + Fine-tuning

BERT는 기본적으로 feature-based를 지양하고 fine-tuning을 기반으로 하는 모델이다.

fine-tuning : 모든 것을(그러나 조금은 미세하게) 업데이트 시키기, 임베딩까지 모두 업데이트하는 기법

feature-based : 임베딩은 그대로 두고 그 위에 레이어만 학습 하는 방법

Fine-tuning approach에는 BERT를 예시로 들 수 있으며, **어떤 특정 task에 대한 parameter를 최소화하여 범용적인 pre-trained 모델을 사용합니다.** 그리고 특정 task를 수행할 경우 **그에 맞게 fine-tuning을 적용**한다.

BERT는 기본적으로 대용량의 unlabeled data로 pre-training을 진행하고 특정 task에 대해 BERT를 여러 대로 복사해 transfer learning을 진행하게 됩니다. 즉, pre-trained 모델이 기반이 되는 것이다.

language에 대해 학습된 값으로 initialised된 상태에서 task specific한 layer를 추가한 뒤 fine-tuning 시작. 이 때 pre-trained된 parameters들과 task specific layer의 parameter들이 모두 학습된다.

대표적으로 feature-based approach에는 ELMO가 포함된다. Feature-based의 핵심은 **어떠한 특정 task를 해결하기 위한 architecture를 task specific하게 구성하고 거기에 pre-trained representations(즉, embedding layer)를 부가적인 feature로 사용하는 것이다.**

ex) task-specific한 model이 pre-trained된 model을 feature로 사용한다. pre-trained된 feature를 concat해서 model의 input으로 사용하는 방식 등이 있다.

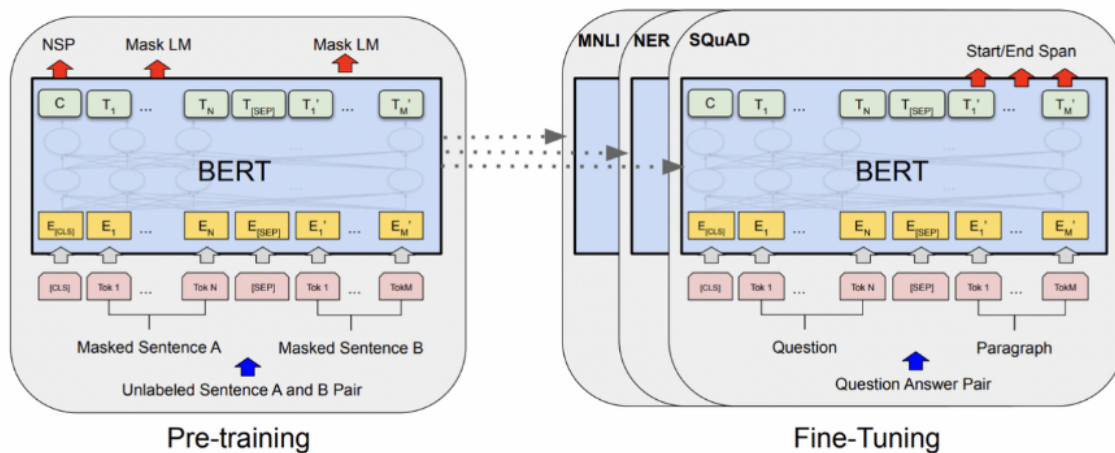
## BERT

BERT는 크게 pre-training 단계와 fine-tuning 단계 두가지 단계로 구분한다.

BERT가 높은 성능을 얻을 수 있었던 것은, 레이블이 없는 방대한 데이터로 사전 훈련된 모델을 만든 후, 레이블이 있는 다른 작업(Task)에서 추가 훈련과 함께 하이퍼파라미터를 재조정하여 이 모델을 사용하기 때문이다. 기존 사례에서도 해당 기법을 사용하면 상당히 좋은 성능이 발휘되는 것이 입증 됐다.

실제 task에서 사용하는 모델은 초기에 동일한 파라미터로 시작하지만, 최종적으로는 서로 다른 fine-tuned된 모델을 보유하게 된다. BERT는 pre-trained된 모델과 fine-tuned된 모델 사이의 구조적 차이가 거의 없게 된다. 그러나 미세하게 차이가 생기는 모델 여러 개를 생성하게 된다.

## Question Answer task



BERT의 구별된 특징은 다양한 작업에 걸친 통일된 구조라는 것이다. 사전 훈련된 구조와 마지막 다운 스트림 구조 사이의 최소한의 차이만 있다.

### Model Architecture

BERT의 모델 구조는 트랜스포머의 인코더만을 다중으로 쌓은 다중 레이어 양방향 트랜스포머 인코더이다. 즉, 기존의 Transformer와 달리 앞과 뒤 양방향에서 Attention을 사용하는 모델이라는 뜻이다.

레이어의 개수를 L, 은닉의 크기는 H, 셀프 어텐션 헤드의 수는 A로 표시한다. 저자들은 BERTBASE(L=12, H=768, A=12, Total Parameters=110M)과 BERTLARGE(L=24, H=1024, A=16, Total Parameters=340M) 두 모델 사이즈로 결과를 보여준다.

### Input / Output Representations

다운스트림 과제 중에 단일 문장이 아닌 문장의 쌍에 대한 학습이 필요한 경우가 있다. 이러한 경우 문장 쌍을 토큰을 통해 순서와 쌍을 만들어줄 수 있다.

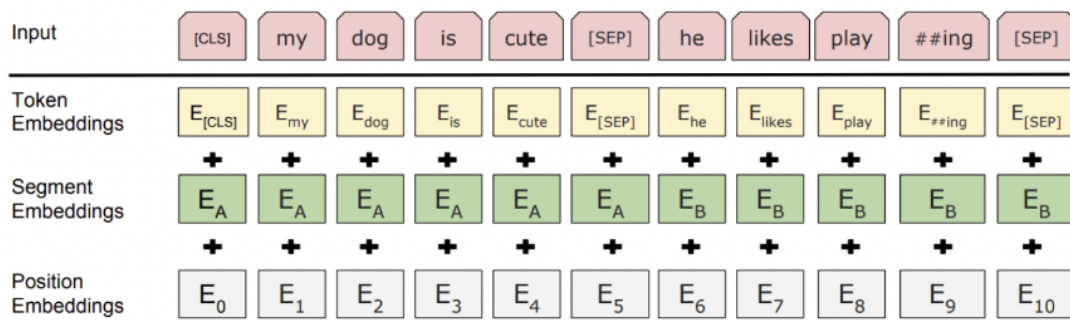
++

이 작업 동안 문장(Sentence)는 실제 언어의 문장 대신에 유사한 텍스트의 임의 범위가 될 수 있다.

그리고 논문에서 나타는 시퀀스(Sequence)는 BERT에 대한 입력 토큰 시퀀스를 말하며 두 개의 문장을 함께 패킹한 것일 수 있다.

문장의 쌍은 하나의 시퀀스로 함께 묶인다. 그리고 이 쌍을 두 가지 방법으로 문장을 구별한다. 먼저, 토큰 ([SEP], [SEP])을 통해 기존의 문장들의 분리를 보여준다. 두 번째로 우리는 모든 토큰에 이것이 문장 A인지 B인지 표시하는 학습된 임베딩을 추가한다.

BERT에서 사용하는 tokenizer는 WordPiece 임베딩 방식이다. 모든 문장의 첫 번째 토큰은 항상 [CLS]이다. 이 토큰을 통해 마지막의 분류를 해줄 때에 분류에 대한 값이 이 토큰의 연산 결과로 나타나게 된다.



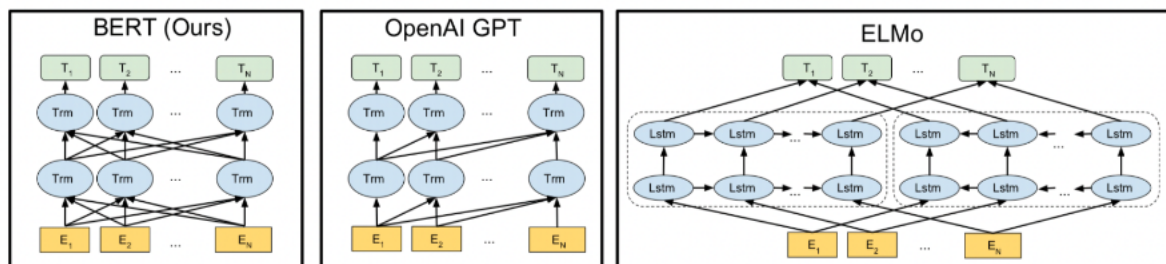
아래 사진을 보면, 각각의 임베딩 layer을 통해 나타나는 embedding을 E로 표시하고 각 단어의 tokenizer embedding, Segment Embedding, 그리고 Position Embedding을 해준다. 결론 적으로 BERT는 총 3개의 임베딩 층이 사용된다.

WordPiece Embedding : 실질적인 입력이 되는 워드 임베딩. 임베딩 벡터의 종류는 단어 집합의 크기로 30,522개. = 실제 word에 대한 embedding

Position Embedding : 위치 정보를 학습하기 위한 임베딩. 임베딩 벡터의 종류는 문장의 최대 길이인 512개. = input에서 몇번째 word인지에 대한 embedding

Segment Embedding : 두 개의 문장을 구분하기 위한 임베딩. 임베딩 벡터의 종류는 문장의 최대 개수인 2개. = 몇번째 sentece에 포함되는지 embedding

## Pre-training BERT



BERT는 기존의 방식들과 달리 두 가지 방향에서 학습을 진행했으며, ,두 가지 비지도 학습을 통해 훈련 시켰다.

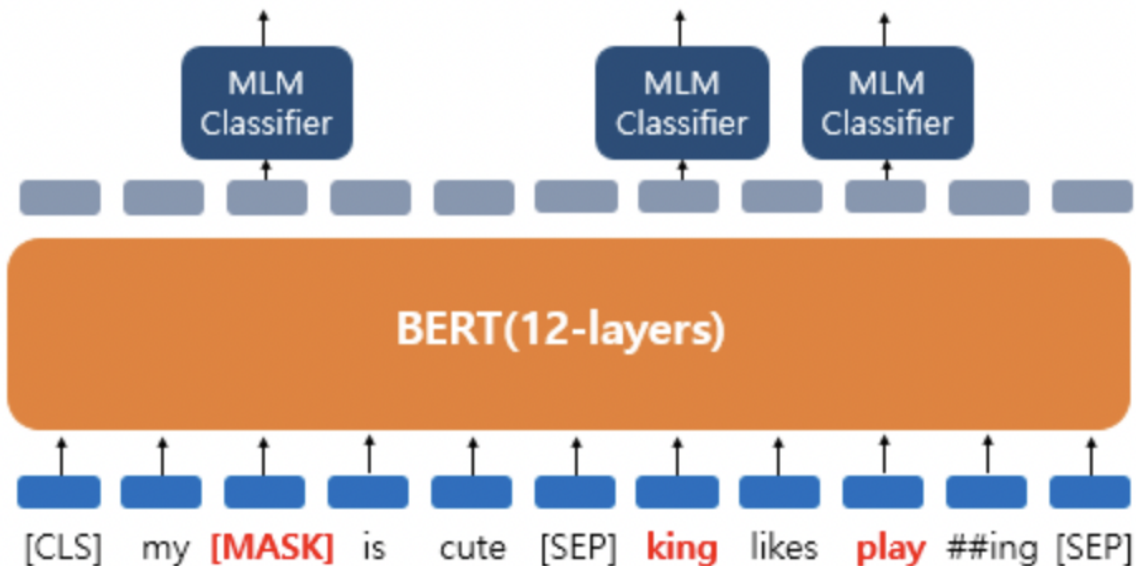
## Masked LM

언어 모델이 아무런 제약조건 없이 **Bidirectional하게 학습을 하게 되면** 간접적으로 예측하려는 단어를 참조하게 되고, **multi-layered 구조에서 해당 단어를 예측할 수 있기 때문이다.** 즉 제대로 학습을 시키는 것이 매우 어려워진다.

좀 더 자세히 설명하자면, 기존의 Model들이 unidirectional model의 결과들을 concat해서 사용한 이유는, bidirectional model은 자기 자신을 masking하더라도 다층 Layer에서는 간접적으로 자기 자신에 대한 정보를 알 수 있기에 제대로 된 학습이 불가능했다. Bidirectional Model 에서 다층 Layer일 경우에는 이전 Layer의 모든 output에서 모든 Token에 대한 정보를 담게 되기 때문에 특정 Token Masking했다고 하더라도 다음 Layer에서는 자기 자신에 대한 정보를 간접적으로 참조할 수 있게 된다.

이를 해결하기 위해 BERT 팀은, 다음 단어가 무엇이 오는지 예측하는 학습이 아니라, **문장 내에서 무작위로 입력 토큰의 일정 비율을 마스킹**하고, 그 다음 **마스킹된 토큰들을 예측**한다. 이를 줄여서 MLM이라 부른다.

이 경우에, **마스킹 토큰에 해당하는 마지막 hidden vector**는 표준 LM에서와 같이 **어휘를 통해 출력 소프트맥스**로 주어지고 단어를 예측하게 된다.



BERT는 **WordPiece 토큰의 15%**를 무작위로 각 시퀀스에서 마스킹한다. 이때 BERT는 문장을 복구하는 것이 목표가 아닌 해당 단어의 예측이 목표가 된다.

이를 통해 양방향으로의 학습이 가능해지지만, **fine-tuning 중에 [MASK] 토큰이 나타나지 않기 때문에(빈칸 단어 예측은 그냥 빈칸 단어로 주어지기 때문)**, 사전 훈련과 fine-tuning 사이에 불일치를 만들어내는 단점이 있다.

**이를 완화하기 위해 "마스킹된" 단어를 실제 [MASK] 토큰으로 항상 교체하지는 않습니다**

훈련 데이터를 생성할 때에 예측을 위해 무작위로 토큰 위치의 15%를 선택한다. 만약  $i$  번째 토큰이 선택된다면,

$i$  번째 토큰을 (1) 80%는 [MASK] 토큰으로 교체하거나 (2) 10%는 다른 토큰(단어)으로 교체하거나, (3) 10%는 변경되지 않은  $i$  번째 토큰을 사용한다.

결과적으로 모델은 변경이 된 단어에 대해 적합한지를 판단해 예측을 하게 된다.

전체 Word중에서 15%만 수행했으므로, 실제로 MASK Token으로 변경되는 비율은 12%밖에 되지 않는다. 이러한 과정을 통해 얻을 수 있는 이점은 Model이 모든 Token에 대해서 실제로 맞는 Token인지 의심을 할 수 밖에 없게 되기에 제대로 된 학습을 이뤄낼 수 있다.

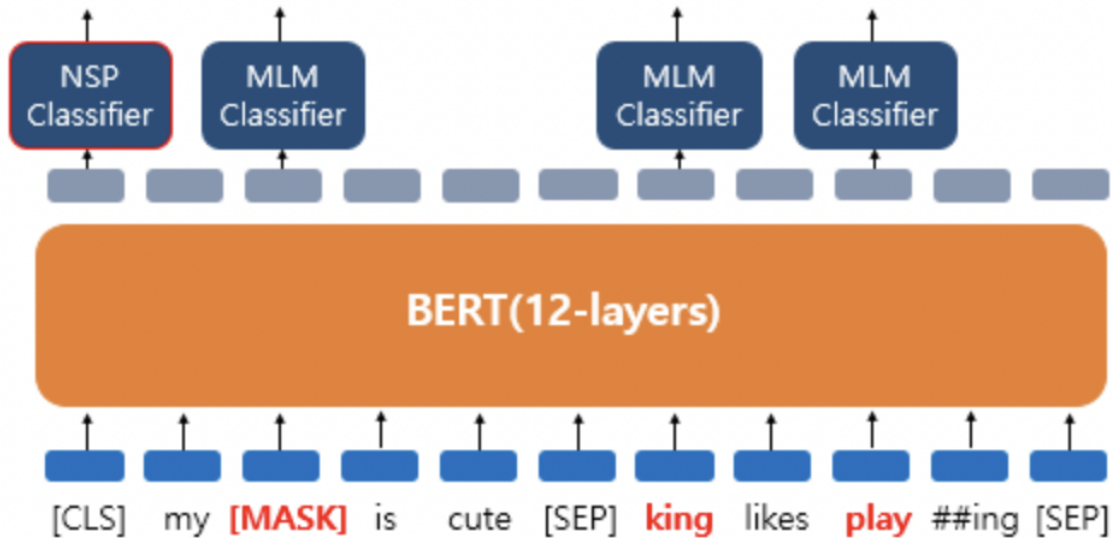
## Next Sentence Prediction(NSP)

NLI와 같은 task는 token단위보다 sentence 단위의 관계가 더 중요하다.

위의 MLM만으로 pre-training을 하게 될 경우 token level의 정보만을 학습하기 때문에 NSP를 통해 sentence level의 정보도 담을 필요가 있다.

위의 학습과 동시에 BERT는 Question Answering에서도 사용되기 위해 두개의 문장을 제공하고 문장들이 이어지는 문장인지 분류하는 훈련도 거치게 된다. 이때 문장들은 1/2 확률로 이어지는 문장이다. (isNext, NotNext의 Binary classification)

그리고 여기서 [SEP] 토큰이 사용된다. 각 문장이 다른 문장임을 보여주기 위해서 문장과 문장 사이에 [SEP] 토큰을 넣게 된다. 그리고 가장 앞에 있는 [CLS] 토큰에 이 문장이 이어지는 문장인지에 대한 예측을 하게 된다.



논문의 Ablation Study에서는 해당 NSP 훈련을 하지 않을 경우 모델의 성능이 많이 감소한다고 명시했다.

### pre-training

실제 Pre-training 단계는 다음과 같이 진행된다.

corpus에서 문장들을 뽑아내 두 문장의 sequence로 만들고, 각각 A,B를 Segment Embedding으로 부여한다. 50%의 확률로 B문장은 실제로 A문장에 이어지는 문장이고, 50%확률로 A문장에 이어지지 않는 문장이다. 이후 Word Piece Tokenization을 한 뒤, Masking을 수행한다.

Hyperparameters는 다음과 같다.

batch size = 256 sequences

sequence size = 512 tokens

#epoch = 40

Optimizer: Adam (learning rate = 1e-4, B<sub>1</sub> = 0.9, B<sub>2</sub> = 0.999, L2 weight decay = 0.01)

dropout: 0.1 in all layers

activation function: gelu

loss function: sum of the mean MLM likelihood + mean NSP likelihood

Pre-training에 BERT\_BASE는 16개의 TPU로 4일, BERT\_LARGE는 64개의 TPU로 4일이 소요됐다.

### Fine-tuning BERT

최종적으로 사전 학습된 BERT를 우리가 풀고자 하는 Task의 데이터를 통해 추가적으로 학습시켜 검증과 테스트를 거치게 된다. BERT 모델 자체가 Transformer를 통해 Attention Encoding을 하게 되고 그 모델을 다시 사용해서 문제를 해결하도록 fine-tuning 하기 때문에 큰 조작을 가하지 않는다.

Fine-tuning은 Pre-training에 비해 매우 빠른 시간 내에 완료된다. Fine-tuning에서는 각각의 task에 specific하게 input size, output size 등을 조정해야 한다. 또한 token-level task일 경우에는 모든 token들을 사용하고, sentence-level task일 경우에는 CLS token을 사용한다.

Hyperparameters는 대부분은 pre-training과 동일하게 진행하는 것이 좋지만, batch size, learning rate, #epochs는 task-specific하게 결정해야 한다. 하지만 다음의 값들에 대해서는 대체적으로 좋은 성능을 보였다.

batch size: 16, 32

learning rate: 5e-5, 3e-5, 2e-5

#epochs: 2, 3, 4

dataset의 크기가 클 수록 Hyperparameter의 영향이 줄어들었으며, 대부분의 경우 Fine-tuning은 매우 빠른 시간 내에 완료되기 때문에 많은 parameters에 대해 테스트를 진행할 수 있었다.

문제에 따라 Fine-tuning 하는 방식이 달라지게 되고 논문에서는 네 개의 학습을 보여주었다.

### 1) paraphrasing

### 2) hypothesis-premise pairs in entailment

- 이론과 가설 쌍의 함의 관계에 대한 분류. 즉, 두 문장에 대한 이론과 가설 관계를 가지는지 분류해주는 것

### 3) question answering

### 4) tagging, text pair classification

- 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 된다.
- text pair classification의 경우 문장 간의 관계를 예측하는 것이다.

이 외에도 다양하게 많은 fine-tuning이 존재한다.

## Experiment and Result

BERT는 총 11개의 NLP 분야에서 학습을 시켰고 이들을 종합해서 평가하고 있다. 직전에 나온 모델이 ELMO와 GPT-1이기 때문에 둘과의 비교가 상당히 많다.

### GLUE

GLUE는 **General Language Understanding Evaluation**의 약자로 다양한 분야의 general language understanding task를 포함하고 이를 평가하고 있다. GLUE에 맞추어 fine-tuning하기 위해 추가해준 것은 분류에 있어서 결과 개수에 따른 가중치 행렬 변화 밖에 없다.

32개의 batch size를 사용했고 3epoch을 통해 학습했다. lr rate는 5e-5, 4e-5, 3e-5, and 2e-5로 설정해주었다.

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
BERT <sub>BASE</sub>	84.6/83.4	71.2	90.5	93.5	52.1	85.8	88.9	66.4	79.6
BERT <sub>LARGE</sub>	<b>86.7/85.9</b>	<b>72.1</b>	<b>92.7</b>	<b>94.9</b>	<b>60.5</b>	<b>86.5</b>	<b>89.3</b>	<b>70.1</b>	<b>82.1</b>

실험 결과 BERT Base, Large 모두 기존의 방법보다 좋은 성능을 보이고 있다.

### SQuAD 1.1

System	Dev		Test	
	EM	F1	EM	F1
Top Leaderboard Systems (Dec 10th, 2018)				
Human	-	-	82.3	91.2
#1 Ensemble - nlnet	-	-	86.0	91.7
#2 Ensemble - QANet	-	-	84.5	90.5
Published				
BiDAF+ELMo (Single)	-	85.6	-	85.8
R.M. Reader (Ensemble)	81.2	87.9	82.3	88.5
Ours				
BERT <sub>BASE</sub> (Single)	80.8	88.5	-	-
BERT <sub>LARGE</sub> (Single)	84.1	90.9	-	-
BERT <sub>LARGE</sub> (Ensemble)	85.8	91.8	-	-
BERT <sub>LARGE</sub> (Sgl.+TriviaQA)	<b>84.2</b>	<b>91.1</b>	<b>85.1</b>	<b>91.8</b>
BERT <sub>LARGE</sub> (Ens.+TriviaQA)	<b>86.2</b>	<b>92.2</b>	<b>87.4</b>	<b>93.2</b>

SQuAD의 경우, Question, Answer로 이루어져 있는 데이터셋 Pair가 학습을 진행하게 된다. 위의 pre-train처럼 각 문장쌍을 하나의 Sequence로 넣게 되었고 answer가 시작할 때 S token과 마지막에 E token을 넣어주었다. 그리고 [SEP] token을 주지 않게 되는데 **모델은 어디서부터 어디까지가 answer의 영역인지** 구한다.

마지막의 hidden state에서 특정 단어 토큰  $T_i$ 이 start token이 될 score는 S와  $T_i$ 의 dot product 결과로 나타나게 된다. End token 역시 E와  $T_i$ 의 dot product 연산을 진행한다.

그리고 softmax를 구해주어 가장 높은 값을 찾는다.

start token S 로 부터 end token E 까지가 **answer 의 score 는  $S \cdot T_i + E \cdot T_j$  로 계산된다.** 그리고  $j \geq i$  인 후보 중에서, 가장 값이 큰  $\langle i, j \rangle$  쌍을 answer 의 영역으로 predict 한다.

$$P_i = \frac{e^{(S \cdot E) \cdot T_i}}{\sum_j e^{(S \cdot E) \cdot T_j}}$$

위의 score를 이용해서  $S \cdot T_i$ 와  $E \cdot T_j$ 를 더한 값이 가장 큰  $\langle i, j \rangle$ 쌍 (단,  $j \geq i$ )을 최종 Answer 영역으로 정한다.

$$\hat{s}_{i,j} = \max_{j \geq i} (S \cdot T_i + E \cdot T_j)$$

Loss 는 올바른 start 와 end position에 대한 sum of log-likelihoods 를 사용한다.google은 3epoch, learning rate 5e-5, batch size 32 를 사용했다 한다.



System	Dev		Test	
	EM	F1	EM	F1
Top Leaderboard Systems (Dec 10th, 2018)				
Human	-	-	82.3	91.2
#1 Ensemble - nlnet	-	-	86.0	91.7
#2 Ensemble - QANet	-	-	84.5	90.5
Published				
BiDAF+ELMo (Single)	-	85.6	-	85.8
R.M. Reader (Ensemble)	81.2	87.9	82.3	88.5
Ours				
BERT <sub>BASE</sub> (Single)	80.8	88.5	-	-
BERT <sub>LARGE</sub> (Single)	84.1	90.9	-	-
BERT <sub>LARGE</sub> (Ensemble)	85.8	91.8	-	-
BERT <sub>LARGE</sub> (Sgl.+TriviaQA)	<b>84.2</b>	<b>91.1</b>	<b>85.1</b>	<b>91.8</b>
BERT <sub>LARGE</sub> (Ens.+TriviaQA)	<b>86.2</b>	<b>92.2</b>	<b>87.4</b>	<b>93.2</b>

성능은 역시나 최고의 수준을 보이고 있다.

## SQuAD 2.0

세번째 실험은 SQuAD 2.0 dataset을 통해 진행했다. SQuAD 2.0은 답이 질문에 없는 즉, 대답할 수 없는 경우를 포함하고 있어 SQuAD보다는 조금 까다로운 Dataset이다. 그리고 그런 경우에는 CLS token에 결과가 나타나게 된다.

인간의 능력에는 미치지 못하였지만, 기존의 baseline에 비해서는 매우 우수한 성능을 확인할 수 있습니다. 이것을 예측하는 수식은 **답변이 있는 문장이라면, 기존 수식과 동일하게 진행한다.**

대답 불가능한 경우에는  $S_{ij}$  가  $s_{null}$  보다 작게 되는데 이를 통해 예측하게 된다. 이때 수식을 그대로 사용하는 것이 아니라 **상수  $r$  이 추가된다.** 결과적으로 이는  $s_{null}$ 과 결합해 threshold로 활용된다.

대답이 불가능할 경우의 Score

$$s_{null} = S \cdot C + E \cdot C$$

대답이 가능한 경우의 Score는 SQuAD v1.1과 동일하다. 대답 가능 여부는 두 Score를 비교해 판단하게 된다. 이 때 threshold값  $r$ 이 사용된다.

$$\hat{s}_{i,j} > s_{null} + r$$



System	Dev		Test	
	EM	F1	EM	F1
Top Leaderboard Systems (Dec 10th, 2018)				
Human	86.3	89.0	86.9	89.5
#1 Single - MIR-MRC (F-Net)	-	-	74.8	78.0
#2 Single - nlnet	-	-	74.2	77.1
Published				
unet (Ensemble)	-	-	71.4	74.9
SLQA+ (Single)	-	-	71.4	74.4
Ours				
BERT <sub>LARGE</sub> (Single)	78.7	81.9	80.0	83.1

성능은 인간의 능력에는 조금 부족하지만 전문가에 필적하는 성능을 보여준다.

## SWAG

**The Situations With Adversarial Generations (SWAG)**이라고 불리는 방식은 앞 문장이 주어지고, 보기로 4 문장이 주어진다. 그 주어진 문장중에서 가장 잘 어울리는 문장을 찾는 **sentence pair inference task** 이다.

Fine tuning 하기 위해, 앞 뒤 문장중 가능한 경우의 수의 문장들을 묶어 하나의 데이터로 만든다. 이때 앞 문장을 **embedding A**, 뒤 문장을 **embedding B**로

그 이후 GLUE 를 학습할때와 동일하게 **CLS token** 에 대응하는 **token C** 와 A 문장 이후에 나타나는 문장의 **token**의 **dot product** 한다. 이를 score 로 삼고, softmax 로 normalize 한다. 결과적으로 softmax 가 만든 확률로 classification을 진행해 가장 어울리는 문장을 찾게 된다.

System	Dev	Test
ESIM+GloVe	51.9	52.7
ESIM+ELMo	59.1	59.2
OpenAI GPT	-	78.0
BERT <sub>BASE</sub>	81.6	-
BERT <sub>LARGE</sub>	<b>86.6</b>	<b>86.3</b>
Human (expert) <sup>†</sup>	-	85.0
Human (5 annotations) <sup>†</sup>	-	88.0

## Ablation Study

인공지능, 특히 머신러닝 분야에서, ablation이란 학습이 사용되는데 AI 시스템의 일부를 제거한 것이다. 이를 통해 제거한 부분이 전체적인 시스템의 성능에 기여하는 바를 연구하는 것이다.

좀 더 직관적으로 말하면 **제안한 요소가 모델에 어떠한 영향을 미치는지 확인하고 싶을 때, 이 요소를 포함한 모델과 포함하지 않은 모델을 비교하는 것**을 말한다. 이는 딥러닝 연구에서 매우 중요한 의미를 지니는데, 시스템의 인과관계(causality)를 간단히 알아볼 수 있기 때문이다.

여기서는 NSP(Next Sentence Prediction), model size, feature-based Approach with Bert 이렇게 세가지로 나눠서 Ablation Study를 진행했다.

여기서 눈여겨 볼 점은 fine-tuning이 아닌 feature-based BERT 모델에 대한 것이다. Feature-based로 진행했을 때는 아래와 같은 실험으로 진행되었다.

#### Feature-based approach의 장점

1. Transformer Encoder의 Output에 몇몇 Layer를 추가하는 간단한 작업만으로는 해결할 수 없는 task들이 존재한다.
2. 매우 큰 pre-training model의 경우 pre-trained된 features를 계속 update하지 않고 고정된 값으로 사용함으로써 연산량을 획기적으로 줄일 수 있다.

Feature-based approach는 아래의 경우를 모두 고려하여 실험을 진행했다

- 1)Embedding만 사용
- 2)두번째 부터 마지막 Hidden을 사용
- 3)마지막 Hidden 만을 사용
- 4)마지막 4개의 Hidden을 가중합
- 5)마지막 4개의 Hidden을 concat
- 6)모든 12개의 층의 값을 가중합

System	Dev F1	Test F1
ELMo ( <a href="#">Peters et al., 2018a</a> )	95.7	92.2
CVT ( <a href="#">Clark et al., 2018</a> )	-	92.6
CSE ( <a href="#">Akbik et al., 2018</a> )	-	<b>93.1</b>
Fine-tuning approach		
BERT <sub>LARGE</sub>	96.6	92.8
BERT <sub>BASE</sub>	96.4	92.4
Feature-based approach (BERT <sub>BASE</sub> )		
Embeddings	91.0	-
Second-to-Last Hidden	95.6	-
Last Hidden	94.9	-
Weighted Sum Last Four Hidden	95.9	-
Concat Last Four Hidden	96.1	-
Weighted Sum All 12 Layers	95.5	-

결과를 살펴보면, 전체 layer를 가중합 하는 것 보다 마지막 4개만을 layer를 concatenate하는 방법이 가장 성능이 좋았다.(미세한 차이이기는 하지만) 또한 **fine-tuning의 성능과도 거의 차이가 없었다**는 점이 인상깊다.

하지만 최근의 트렌드가 점점 fine-tuning으로 변화하는 이유는 그것이 성능이 더 좋기 때문도 있지만 학습에 대한 cost가 낮기 때문이다. feature-based는 위치럼 모델 자체를 task에 맞게 변형시켜주고 학습시켜야 하기 때문에 부담이 크다.

## Effect of Pre-training Tasks

Tasks	Dev Set				
	MNLI-m (Acc)	QNLI (Acc)	MRPC (Acc)	SST-2 (Acc)	SQuAD (F1)
BERT <sub>BASE</sub>	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 BERT<sub>BASE</sub> 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.

No NSP는 pre-training 단계에서 MLM만 수행하고, NSP는 수행하지 않은 model이다. LTR & No NSP는 NSP 는 수행하지 않고, MLM 대신 Left to Right의 Unidirectional attention을 적용한 model이다.

BERT와 No NSP를 비교함으로써 NSP Pre-training이 성능 향상에 영향을 끼친다는 것을 알 수 있다. 한편, No NSP와 LTR & No NSP를 비교함으로써 Bidirectional Attention(MLM)이 성능 향상에 매우 큰 영향을 준다는 것을 알 수 있다. Token Level에서 Right to Left Context 정보를 얻기 위해 MLM이 아닌 BiLSTM을 추가했다. 해당 Model은 LTR & No NSP에 비해 SQuAD와 같은 task에서 매우 큰 성능 향상을 보였지만, 여타 task에서는 오히려 성능 하락을 보였다.

물론 Bidirectional Context를 담기 위해 LTR과 RTL을 각각 학습시킨 뒤 두 token을 concatenation하는 방법도 있지만, 이는 비용이 2배나 높고, QA와 같은 RTL 학습이 불가능한 task에서는 적용이 불가능하다는 점, 결론적으로 MLM과 같은 Deep Bidirectional Model에 비해 성능이 낮다는 점에서 비효율적이다.

## Effect of Model Size

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.**

GLUE task 중 3개를 뽑아 Model Size에 따라 성능을 측정했다. Model Size가 증가할수록 성능이 높아지는 경향을 확인할 수 있다. 특히 MRPC task는 pre-training task와 차이가 큰 task이면서 3600개의 적은 labeled training data를 사용했음에도 불구하고 Model Size가 증가함에 따라 성능도 향상됐다. 이를 통해 Model Size의 증가는 번역과 Language Modeling과 같은 큰 scale의 task에서도 성능 향상에 기여함은 물론, 충분한 pre-training이 있었다는 전제 하에 작은 scale의 task에서도 성능 향상에 기여함을 알 수 있다.

## 번외. RoBERTa

기존 BERT에 더 많은 데이터와 더 나은 hyper-param을 통해 BERT 이후의 PLM보다 더 나은 성능을 얻을 수 있음을 보임

### Different

- 기존 bert대비 10배 가량의 데이터를 더 오래 학습
- 실험을 통해 NSP가 불필요함을 보임 → NSP 안써도 성능이 좋다.-
- 최대 입력 길이(512)개에 맞춰 입력을 구성
- Feed-foweward 때마다 masking을 새롭게 구성

### Wrap-up

2018년 BERT의 등장 이후, 크기의 증가와 다양한 알고리즘을 통해 더 나은 PLM에 대한 연구가 많음


RoBERTa는 다음의 방법을 통해 BERT의 성능을 한층 더 끌어올림

BERT 이후 많은 PLM들이 제안 되었지만 결국 RoBERTa를 애용

### 참고

[NLP | 논문리뷰] BERT : Pre-training of Deep Bidirectional Transformers for Language Understanding 하편

상편을 쓰고 귀찮아하는 제 표정 절대 아닙니다. 앞의 WordPiece와 BERT 상편에서 이어지니 보고 오시는 것을 개인적으로 추천드립니다. BERT는 총 11개의 NLP 분야에서 학습을 시켰고 이들을 종합해서 평가하고 있다. 직전에 나온 모델이 ELMO와 GPT-1이기 때문에 둘과의 비교가 상당히 많다. GLUE GLUE는 General Language Understanding Evaluation의 약자 로 다양한 분

 <https://velog.io/@xuio/NLP-%EB%85%BC%EB%AC%B8%EB%A6%AC%EB%B7%B0-BERT-Pre-training-of-Deep-Bidirectional-Transformers-for-Language-Understanding-%ED%95%98%ED%8E%B8>

