BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding

원문 링크는 다음과 같다.

BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding

Introduction

PLM(Pre-trained Language Model)은 많은 NLP task에서 효과적임이 증명되어왔다. 이러한 pre-trained language model을 downstream task에 적용하는 방법에는 크게 두 가지 방법이 있는데, feature-based와 fine-tuning이다.

Feature-based approach의 경우 ELMo에서 사용한 방식으로, pre-trained representation을 architecture의 추가적인 feature로 사용하는 방법론이다.

반면에, fine-tuning approach의 경우 GPT에서 사용한 방식이며, pre-trained parameter 전체를 가지고 downstream task에서 학습하며 feature-based approach와는 다르게 task-specific parameter를 최소화하는 방법이다.

이러한 두 방식은 위와 같은 차이점이 있지만, pre-training시에 같은 objective function(목적 함수)를 가지며, unidirectional language model(단방향 언어 모델)을 사용했다는 공통점이 있다.

논문에서는 unidirectional language model을 사용하는 2018년 당시의 기술이 pre-trained representation의 성능을 제한하다고 지적한다. 그러면서 unidirectional language model은 pre-training 중에 사용될 수 있는 architecture의 선택을 제한한다고 덧붙인다.

이러한 unidirectional language model의 한계는 sentence-level task에서도 차선책이며, Question Answering과 같이 양방향에서의 맥락을 통합하는 것이 중요한 task에서는 좋지 않은 결과를 도출할 수 있다고 주장한다.

저자들은 이러한 한계를 넘어설 수 있는 새로운 architecture인 BERT(Bidirectional Encoder Representation from Tranformer)를 소개한다. BERT는 위에서 언급한 unidirectional한 architecture의 한계를 **MLM(Masked Language Model)을 pre-training object로 사용하면서 해결하였다고 한다.**

MLM(Masked Language Model)에 대해 간략하게 설명하자면, input으로부터의 token들 중 몇 개를 random하게 mask 하고, 해당 MLM의 objective는 masked word의 original vocabulary를 context만 가진 상태로 예측하게끔 설정하는 architecture이다. 보다 자세한 내용에 대해서는 하단에 기술하도록 하겠다.

논문에서는 이러한 MLM은 기존 left-to-right language model(unidirectional language model)과는 다르게, **pre-trained representation이 양방향의 context(문맥)을 융합할 수 있게 하고, 이러한 특성 덕분에 deep bidirectional transformer architecture를 pre-train 할 수 있다고 한다.**

또한, **MLM과 더불어 next sentence prediction(NSP)를 통해 text-pair representation을 pre-train한다고 말한다.**

BERT

BERT를 학습시키는 과정에는 두 단계가 있는데, 바로 pre-training과 fine-tuning이다.

Pre-training 단계에서는 model이 unlabeled data를 통해 학습되며, fine-tuning 단계에서는 pre-training으로 초기화된 parameter를 가지고 labeled data를 이용하여 downstream task에 대한 paramter를 학습하게 된다.

이때, 각각의 downstream task에 대한 pre-trained parameter가 같더라도 각 task는 각각의 fine-tuned model을 가진다.

또한, **pre-trained architecture와 fine-tuned 된 downstream architecture에는 큰 구조적 차이가 없으며,** 이는 BERT의 독특한 특징인 **다양한 task에 걸친 unified architecture**라는 점에서 기인한다.

그렇다면, 이러한 BERT의 architecture에 대해 알아보자.

Model Architecture

BERT의 architecture는 multi-layer bidirectional transformer encoder로 구성되어 있다.

Architecture에 대해 자세히 살펴보기 전에, 본 논문에서 정의하는 기호에 대해 정의하고 넘어가도록 하겠다.

본 논문에서, L은 layer의 개수(transformer block의 개수), H는 hidden size, A는 self-attention의 head 개수를 의미한다.

논문에서는 두 가지의 모델을 제시한다. 바로 BERT-BASE와 BERT-LARGE 이다.

BERT-BASE는 $L = 12$, $H = 768$, $A = 12$, Total Parameters = 110M의 size를 가진 model이며, BERT-LARGE의 경우에는 $L = 24$, $H = 1024$, $A = 16$, Total Parameters = 340M의 size를 가진 model이다. 이때, BERT-BASE의 경우 OpenAI의 GPT와의 비교를 위해 같은 model size로 설계되었다고 한다.

Input/Output Representation

BERT를 다양한 down-stream task에 활용하기 위해, BERT의 input representation은 single sentence와 pair of sentence(<Question, Answer>)를 하나의 token sequence로 표현한다.

본 논문에서는 WordPiece embedding을 사용하였으며, 30000개의 token vocabulary를 사용한다.

(본격적으로 이에 대해 다루기 전에 짚고 넘어가야 할 것이 있다. 논문에서는 "sentence"가 실제 언어적인 문장이 아닌, contiguous text(연속된 텍스트)의 임의적인 범위가 될 수 있으며, "sequence"의 경우 BERT에 대한 input token sequence를 의미하며, 하나의 "sentence"혹은 두 개의 "sentence"가 함께 packed 될 수 있다고 명시한다.)

우선, 모든 sequence의 시작 부분에 [CLS] token를 추가한다. 해당 [CLS] token은 transformer 전체 층을 다 거치고 나면 token sequence의 결합된 의미를 가지게 되는데, 여기에 **classifier(feed-forward network와 softmax 함수를 이용)를 붙이면 단일 문장, 또는 연속된 문장의 classification을 할 수 있다**. 이러한 특성 때문에 이는 곧 classification task에서 사용된다.

앞서 우리는 **pair of sentence도 하나의 token sequence로 표현된다고 했다. 이를 위해, [SEP] token을 활용한다. 모든 sentence의 끝에 [SEP***] token을 추가**해주는데 예를 들면, (<Question, Answer>)와 같은 pair of sentence는 ([CLS], Question, [SEP], Answer, [SEP])과 같은 sequence로 변환하는 것이다. 그런 다음 각 token에 대해 embedding을 진행한다.

아래의 그림을 통해 한번 더 살펴보자

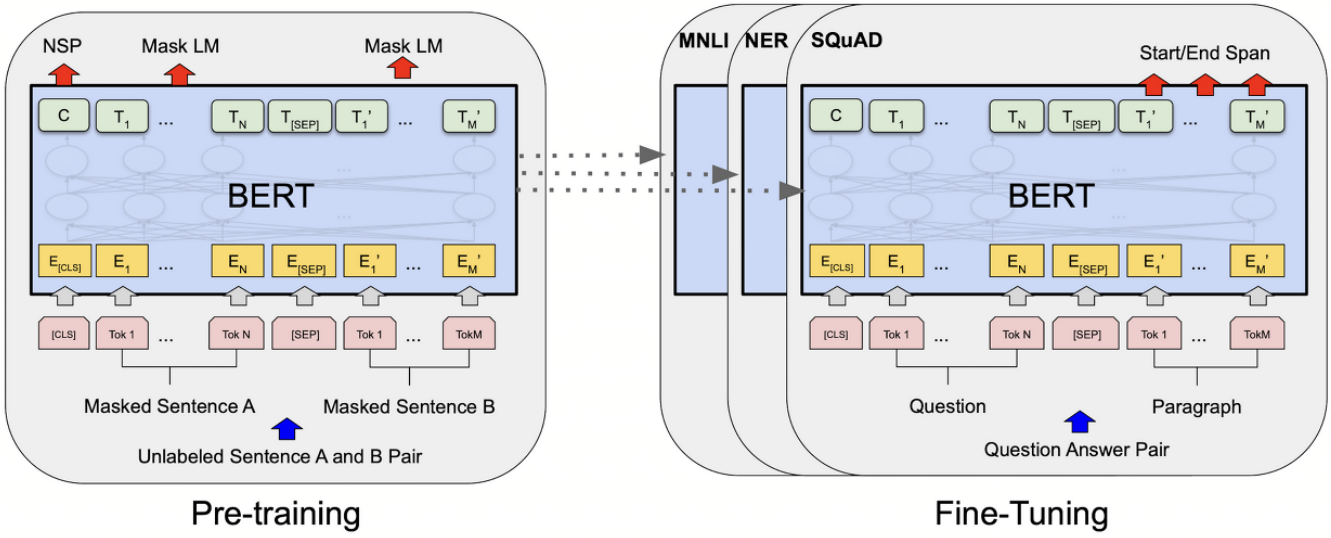


Figure 1: Overall pre-training and fine-tuning procedures for BERT. Apart from output layers, the same architectures are used in both pre-training and fine-tuning. The same pre-trained model parameters are used to initialize models for different down-stream tasks. During fine-tuning, all parameters are fine-tuned. `[CLS]` is a special symbol added in front of every input example, and `[SEP]` is a special separator token (e.g. separating questions/answers).

우선, sequence의 시작 부분에 [CLS] token이 추가된 것과 문장의 마지막에 [SEP] token이 추가된 것을 확인할 수 있다.

(위 자료에서는 마지막 문장의 마지막 부분에는 [SEP] token이 추가되지 않았다.)

이후, 기존 token들과 추가된 [CLS], [SEP] token에 대해 embedding을 거친 뒤, 모든 token에 대해 해당 token이 어느 sentence에 포함되어있는지를 나타내 주는 embedding을 더해준다.

추가적으로, BERT는 transformer 기반이기 때문에 input representation에 위치 정보를 담아줘야 한다. 따라서 위치 정보를 담고 있는 embedding도 추가적으로 더해준다.

정리하자면, BERT의 input representation은 token에 대한 embedding, 해당 token이 어느 sentence에 포함되어있는지에 대한 embedding, 위치 정보를 담고 있는 embedding의 합으로 이루어지며, 이를 각각 **Token embedding, Segment embedding, Position embedding이라고** 한다. (하단 그림 참조)

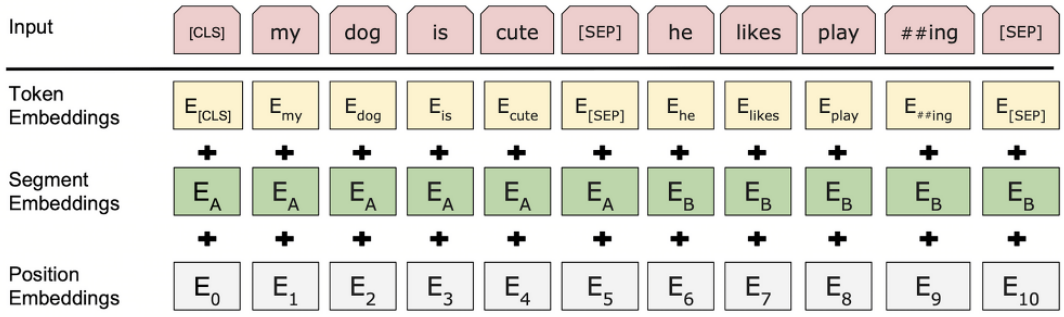


Figure 2: BERT input representation. The input embeddings are the sum of the token embeddings, the segmentation embeddings and the position embeddings.

Pre-training BERT

위에서 잠깐 언급한 것처럼, BERT는 ELMo와 GPT와는 달리 전통적인 left-to-right 혹은 right-to-left language model을 통해 pre-train하지 않는다. 대신 2개의 unsupervised task, MLM(Masked Language Model)와 NSP(next sentence prediction)를 통해 BERT를 pre-training 한다.

Masked LM

기존의 Language model을 bidirectional 하게 학습할 경우, 해당 timestep에서 예측하고자 하는 단어를 간접적으로 참조할 수 있기 때문에, 예측이 무의미해질 수 있다.

MLM(Masked Language Model)은 이러한 문제를 해결하고 **d**eep bidirectional representation을 학습하기 위해 도입된 개념**이다.

우선, MLM에서는 각각의 sequence에 존재하는 WordPiece token들 중에서 15%를 random 하게 masking 한다. 이러한 방법은 bidirectional pre-trained model을 얻을 수 있지만, pre-training에 존재하는 [MASK] token이 fine-tuning에는 존재하지 않기 때문에 pre-training과 fine-tuning 사이에 불일치를 일으킬 수 있다는 문제가 있다.

이러한 문제를 해결하고자, 본 연구에서는 masking 되는 sequence tokens의 15%를 전부 [MASK] token으로 바꾸지 않는다.

해당 15%의 token들 중에서, 80%만 [MASK] token으로 치환하고, 10%는 random 하게 임의의 token으로 치환, 나머지 10%의 token에 대해서는 변경하지 않고 그대로 사용한다.

이후, 15%의 random하게 masking 된 token들에 대해서, original token을 예측하게끔 학습된다.

(이때, 80%의 token과 10%의 임의로 변경된 token, 10%의 원본 token 모두 포함하여 예측의 대상이 된다.)

이 과정을 보다 자세하게 이해하기 위해, 예제와 함께 살펴보도록 하겠다.

다음과 같은 문장이 있다고 가정해보자. (편의를 위해 각 단어가 token이라고 가정)

> my dog is hairy

우선, 15%의 token을 random 하게 masking한다. 여기서는 hairy를 masking한다고 해보자.

만약, 해당 token을 로 치환한다고 하면

> my dog is

와 같은 형태가 될 것이며, 임의의 token으로 치환하면

> my dog is apple

과 같은 형태가 된다. 마지막으로, 변경하지 않고 그대로 사용하는 것은 원본과 같이

> my dog is hairy

와 같은 형태가 된다.

이러한 Masking rule은 transformer의 encoder가

또한, random하게 임의의 token으로 치환된 비율은 전체 sequence에서는 1.5%밖에 되지 않기에, 모델이 잘못 학습될 가능성도 낮다.

저자들은 이러한 Masking rule을 다양하게 변경하여 실험을 진행하였는데, 그 결과는 다음과 같다.

Masking Rates			Dev Set Results		
MASK	SAME	RND	MNLI	NER	
			Fine-tune	Fine-tune	Feature-based
80%	10%	10%	84.2	95.4	94.9
100%	0%	0%	84.3	94.9	94.0
80%	0%	20%	84.1	95.2	94.6
80%	20%	0%	84.4	95.2	94.7
0%	20%	80%	83.7	94.8	94.6
0%	0%	100%	83.6	94.9	94.6

Table 8: Ablation over different masking strategies.

해당 결과를 통해, 논문에서 제안된 80/10/10의 masking rule이 가장 높은 성능을 내는 것을 확인할 수 있다. 또한, 전반적으로 fine-tuning 된 model이 feature based model보다 성능이 높음을 확인할 수 있다.

저자들은 MLM이 각각의 batch에서 15%의 token만을 사용하여 prediction 하기 때문에 기존의 모든 단어에 대해 predict 하는 left-to-right model보다 converge 속도가 느리다는 것을 밝힌다. 그러나, 성능 측면에서의 개선이 증가된 training cost를 훨씬 능가한다고 덧붙인다.

Next Sentence Prediction (NSP)

Question Answering(QA)이나 Natural Language Inferenced(NLI)와 같은 task는 두 문장 사이의 관계를 이해해야 하는 task지만, 일반적인 language modeling으로는 두 문장 사이의 관계를 직접적으로 담는 것이 불가능하다.

논문에서는 이러한 문장 사이의 관계를 학습하기 위해 corpus에서 두 문장을 이어 붙여 해당 sequence가 원래의 corpus에서도 이어 붙여 저 있던 문장인지를 맞추는 binary next sentence-prediction을 수행한다.

먼저, 각각의 pre-training example에서 문장 A, B를 고른다.

이후 해당 pair of sentence의 50%는 문장 A 다음에 오는 문장 B를 그대로 유지한 다음 IsNext라고 labeling 하고, 나머지 50%에 대해서는 corpus의 문장 중에서 random 하게 치환한 다음 NotNext라고 labeling 한다. (위에서 binary next sentence-prediction라고 언급한 이유는 이렇게 IsNext와 NotNext를 분류하는 이진 분류 task이기 때문이다)

이후 [CLS] token의 final hidden vector C를 이용하여 next sentence prediction을 수행한다. 이렇게만 이야기하면 이해가 잘 안 될 수 있으니 예시와 함께 설명하도록 하겠다.

다음과 같은 pair of sentence가 주어진다고 가정해보자.

> She cooked pasta
> It was delicious

해당 pair of sentence를 그대로 유지한다면, IsNext로 labeling 한다. 그러나 만약 다음과 같이 random 하게 바뀐다면

> She cooked pasta
> He was reading

이 경우에는 NotNext로 labeling 하는 것이다.

이후에는 input representation으로 변환해주는 작업을 거친다.

우선 앞서 살펴봤던 것처럼 token으로 바꿔준다. (편의를 위해 각 단어가 token이라고 가정)

> [CLS], she, cooked pasta, [SEP], it, was, delicious. [SEP]

이후 해당 token들에 대해 token embedding과 segment embedding, position embedding을 더하여 input representation을 만들어준 뒤, [CLS] token의 final hidden vector C를 feed-forward network에 넣은 뒤 softmax를 취해 isNext와 NotNext에 대한 확률 값을 계산한다. 위에서 잠깐 언급했듯이, **[CLS] token은 transformer 전체층을 다 거치고 나면 token sequence의 결합된 의미를 가지고 있기 때문에 downstream task가 classification task일 때 혹은 이러한 NSP에 사용된다.**

Pre-training data

본 연구에서는 pre-training을 위해 800M 개의 words로 구성된 BookCorpus dataset과 2500개의 words로 이루어진 English Wikipedia dataset에서 text passage 만 사용했고, 목록이나 표 등은 제외하여 사용하였다.

또한, 긴 문맥을 학습하기 위해 Billion Word Benchmark와 같이 섞인 문장으로 구성된 데이터는 사용하지 않았다고 한다.

Fine-tuning BERT

Fine-tuning에서도 pre-training과 마찬가지로 single sentence와 pair of sentence(<Question, Answer>)를 하나의 token sequence로 바꿔주는 작업을 진행한다.

이후, 이렇게 만들어진 BERT의 input representation을 통해 parameter를 각각의 downstream task에 맞게 end-to-end로 fine-tune 한다.

이때, Sequence tagging이나 question answering 같이 **token-level task 들의 경우, 마지막 transformer layer의 token들의 hidden state로부터 나오는 output을 이용하여 fine-tuning 하고,**

Sentence Classification, sentiment analysis 등의 **sentence-level classification task 들은 마지막 layer의 [CLS] token의 hidden state로부터 나오는 output을 통해 fine-tuning을 진행**한다.

다음은 이러한 각 task별 fine-tuning에 대해 시각화한 자료이다.

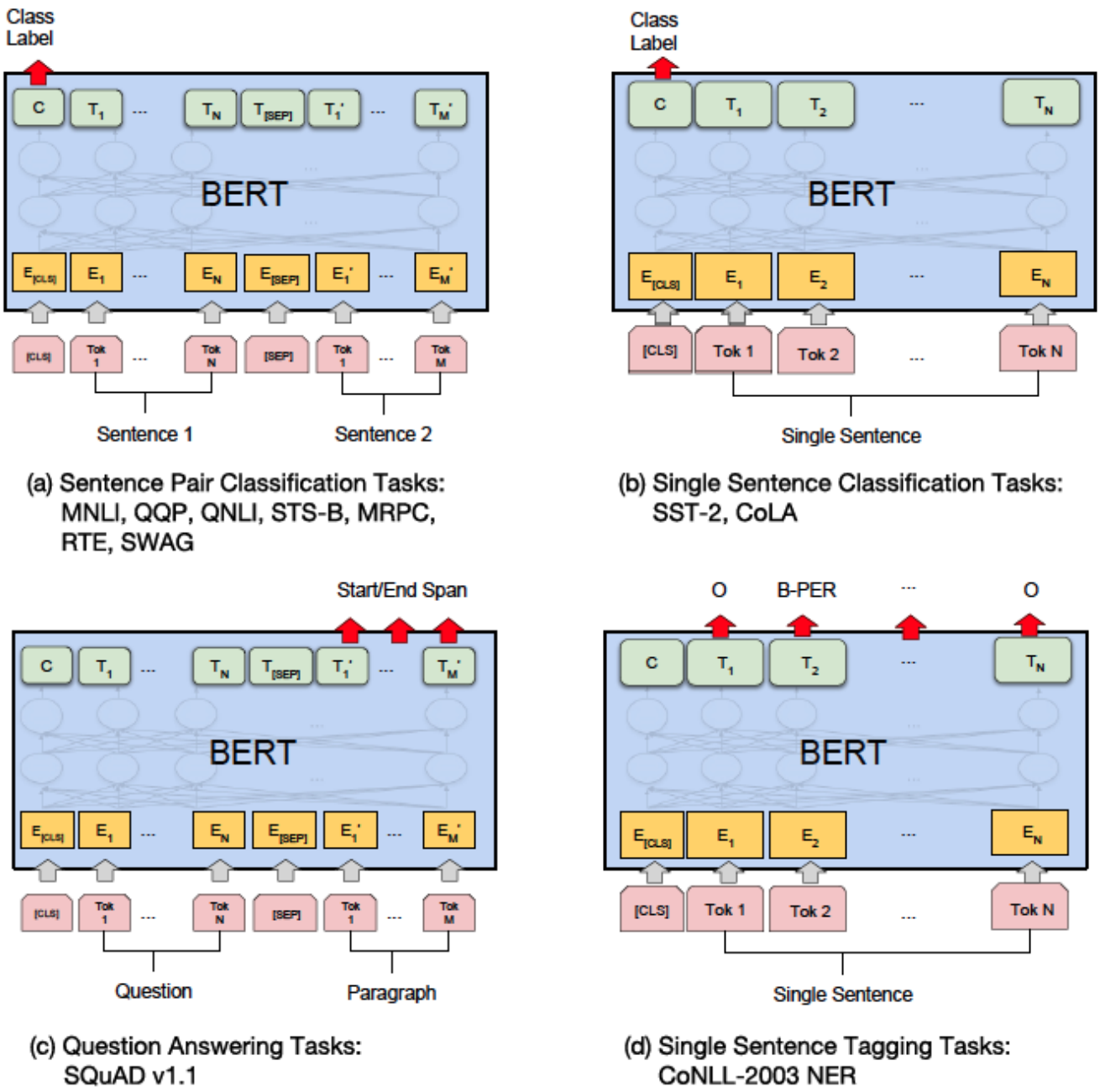


Figure 4: Illustrations of Fine-tuning BERT on Different Tasks.

마지막으로, Pre-training과 비교했을 때, fine-tuning 은 빠르게 학습이 가능하다.

Experiments

논문에서는 11개의 NLP task에 대해 BERT-fine tuning test를 진행하였다고 한다.

(GLUE의 8개 task(MNLI, QQP, QNLI, STS-B, MRPC, RTE, SST-2, CoLA)와 SQuAD v1.1, v2.0, SWAG)

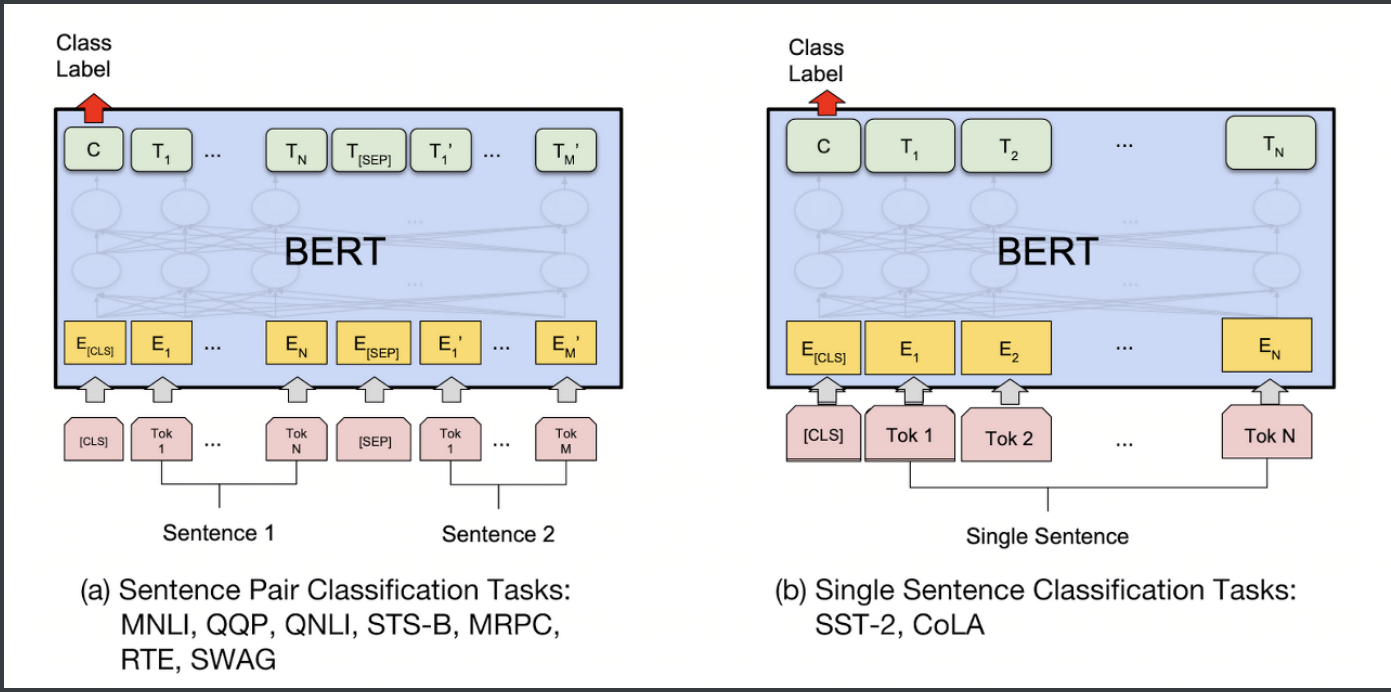
GLUE

저자들은 GLUE(General Language Understanding Evaluation) benchmark를 통해 fine-tuning 된 BERT model의 성능을 측정해보았다.

GLUE에 관련된 내용은 아래의 링크에 기술되어있다.

(논문 리뷰) GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding - GLUE

GLUE task에 대한 fine tuning 방법론에 대한 간략한 설명은 다음과 같으며



(a) Sentence Pair Classification Tasks: MNLI, QQP, QNLI, STS-B, MRPC, RTE, SWAG

(b) Single Sentence Classification Tasks: SST-2, CoLA

결과는 다음과 같다.

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.

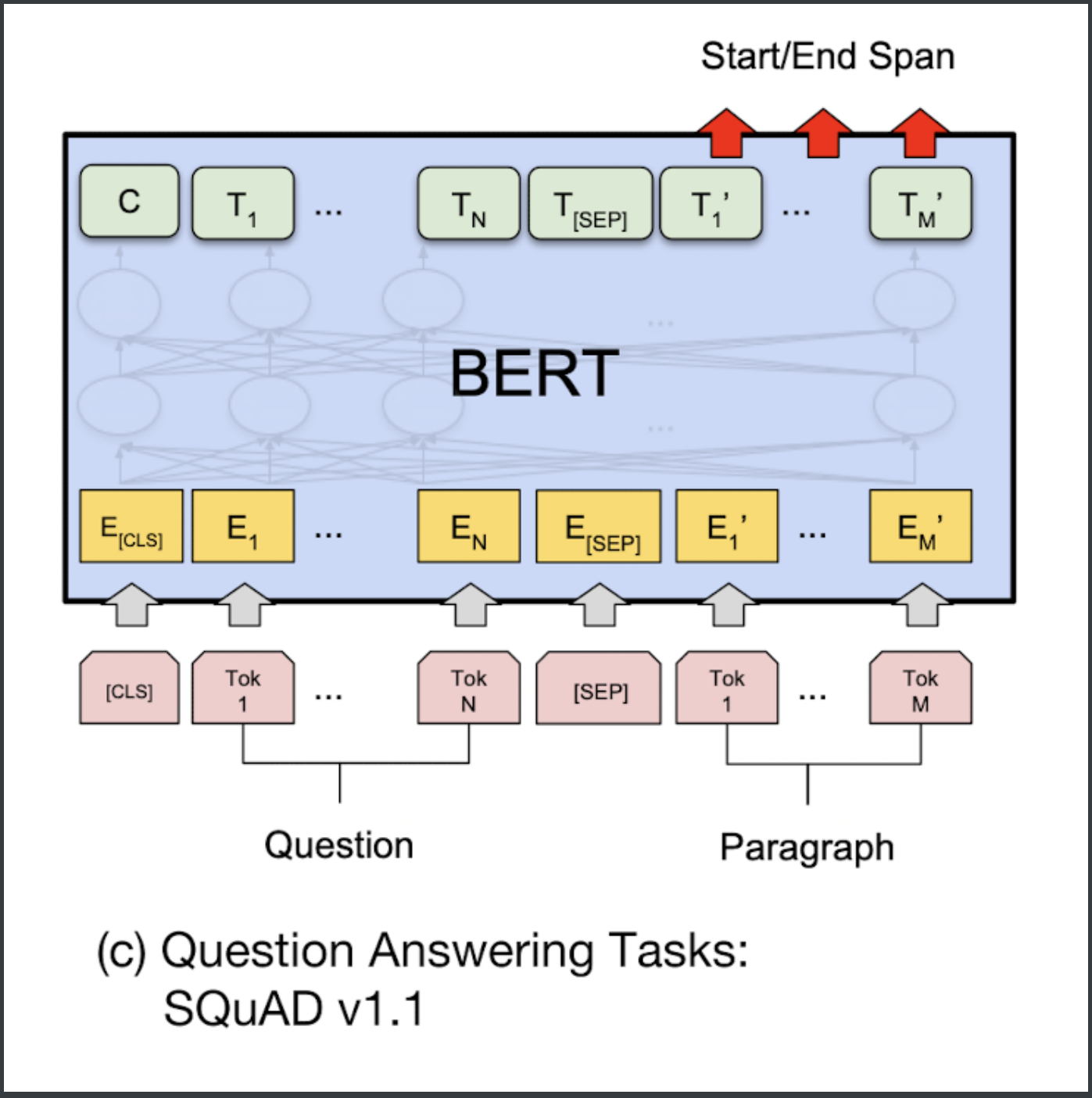
GLUE의 모든 task에서 SOTA를 달성했음을 확인할 수 있다.

또한, BERT-LARGE가 BERT-BASE에 비해 훨씬 좋은 성능을 내는 것을 확인할 수 있다. 특히, dataset의 크기가 작아도 BERT-LARGE가 더 좋은 성능을 내는 것을 확인할 수 있다.

SQuAD v1.1

SQuAD(Stanford Question Answering Dataset) dataset은 question과 answer를 담고 있는 passage를 제공하며, 이를 통해 passage 속에 있는 answer를 예측하는 Question Answering task를 진행한다.

이러한 Question Answering task에 대한 fine-tuning 과정을 살펴보자



(c) Question Answering Tasks: SQuAD v1.1

우선, question과 passage를 single packed sequence로 나타낸다. 이후 start vector $S \in \mathbb{R}^H$와 end vector $E \in \mathbb{R}^H$ 를 추가적으로 사용하게 된다. 이 start vector와 end vector가 무슨 역할을 하는지 알아보기 위해 QA task에서 passage로부터 answer을 예측하는 과정을 살펴볼 필요가 있다.

다음과 같은 question과 passage가 있다고 가정해보자.

> Question : 기계식 키보드는 무엇입니까?
> Passage : 기계식 키보드는 컴퓨터 자판의 한 종류로서 각각의 키가 축, 스프링 등으로 이루어진 스위치를 가지고 있는 키보드이다. 키보드 중에서 스프링과 기타 부품을 이용하여 만든 스위치들이 각각의 자판의 버튼을 구성하고, 스위치를 누르면 스위치가 장착된 기판에서 입력을 인식하는 방식이다. 내구도 면에서 대중적으로 사용되는 멤브레인 입력 방식의 키보드보다 우월하지만, 가격대가 최저 3만 원 중반부터 시작하므로 멤브레인 입력 방식의 키보드보다 비싼 편이다. 멤브레인 키보드가 하나의 키마다 1천만 번의 입력이 가능한 데 비하여, 대표적으로 기계식 키보드의 스위치와 키보드를 제작하는 독일의 ZF Friedrichsafen AG의 브랜드인 체리는 자사의 스위치가 스위치 당 5천만 번의 입력이 가능하다고 한다

여기서, QA task를 위해서는 passage에서 answer를 추출해내야 한다. 위의 예제에서는

"기계식 키보드는 컴퓨터 자판의 한 종류로서 각각의 키가 축, 스프링 등으로 이루어진 스위치를 가지고 있는 키보드이다."

가 answer이 되며, 이때 passage에서 첫 번째 인덱스인 "기계식"부터 15번째 인덱스인 "키보드이다" 까지가 answer의 범위이다.

즉, 우리는 QA task를 수행하는 model로 하여금 question에 대한 answer가 passage에서 어디부터 어디까지인지를 학습시키는 것이다.

그렇다면, passage에서 어디부터 어디까지가 answer인지는 어떻게 학습시킬 수 있을까?

이는 start vector S와 end vector E를 이용하여, passage 내의 token이 answer의 시작과 끝 token이 될 확률을 구함으로써 해결된다.

먼저, passage의 각각의 token이 answer의 시작이 될 확률을 구해보자.

passage 내부의 각각의 token T_i와 start vector S의 내적을 계산한다. 이후 해당 내적 값에 대해 softmax를 취해 각 token이 시작 token이 될 확률을 얻는데, 이는 다음의 수식과 같다.

$$P_i = \frac{e^{S \cdot R_i}}{\sum_j e^{S \cdot R_j}}$$

이후, 시간 token이 될 확률이 가장 높은 token의 index를 선택하여 시작 index를 계산한다.

passage의 각각의 token이 answer의 끝이 될 확률도 같다. 다만 start vector S가 end vector E가 되는 차이뿐이다.

$$P_i = \frac{e^{E \cdot R_i}}{\sum_j e^{E \cdot R_j}}$$

이렇게 시작 Index와 끝 index를 사용하여 passage에서 answer을 포함하는 범위를 선택할 수 있게 된다.

위와 같은 과정으로 얻어진 시작 index를 i, 끝 index를 j라고 할 때 $S \cdot T_i + E \cdot T_j$를 통해 선택된 span의 score가 산출되며, j가 i보다 크거나 같은 상황에서 가장 높은 score를 가진 span이 prediction으로 사용된다.

이때, training object는 correct start and end position의 log-likelihood의 합을 최대화하는 것을 목적으로 하여 training 한다.

이러한 과정을 거친 결과를 살펴보도록 하겠다.

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				
$\text{BERT}_{\text{BASE}}$ (Single)	80.8	88.5	-	-
$\text{BERT}_{\text{LARGE}}$ (Single)	84.1	90.9	-	-
$\text{BERT}_{\text{LARGE}}$ (Ensemble)	85.8	91.8	-	-
$\text{BERT}_{\text{LARGE}}$ (Sgl.+TriviaQA)	**84.2**	**91.1**	**85.1**	**91.8**
$\text{BERT}_{\text{LARGE}}$ (Ens.+TriviaQA)	**86.2**	**92.2**	**87.4**	**93.2**

Table 2: SQuAD 1.1 results. The BERT ensemble is 7x systems which use different pre-training checkpoints and fine-tuning seeds.

이러한 QA task에서도, BERT가 SOTA를 달성함을 확인할 수 있다.

SQuAD v2.0

SQuAD v2.0은 제공된 passage에 answer이 존재하지 않을 가능성을 허용함으로써 SQuAD v1.1에서의 문제 정의를 확장한 버전이다. 이를 BERT에 적용하기 위해, answer가 존재하는지에 대한 여부를 CLS token을 이용해 분류하는 과정을 추가한다.

Table 2: SQuAD 1.1 results. The BERT ensemble is 7x systems which use different pre-training checkpoints and fine-tuning seeds.

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				
$\text{BERT}_{\text{LARGE}}$ (Single)	78.7	81.9	80.0	83.1

Table 3: SQuAD 2.0 results. We exclude entries that use BERT as one of their components.

이러한 SQuAD v2.0에서도 BERT의 성능이 가장 높음을 확인할 수 있다.

SWAG

The Situations With Adversarial Generations (SWAG) dataset은 앞 문장이 주어졌을 때, 보기로 주어진 4 문장 중 가장 잘 어울리는 문장을 찾는 task이다.

System	Dev	Test
ESIM+GloVe	51.9	52.7
ESIM+ELMo	59.1	59.2
OpenAI GPT	-	78.0
$\text{BERT}_{\text{BASE}}$	81.6	-
$\text{BERT}_{\text{LARGE}}$	**86.6**	**86.3**
Human (expert)†	-	85.0
Human (5 annotations)†	-	88.0

Table 4: SWAG Dev and Test accuracies. †Human performance is measured with 100 samples, as reported in the SWAG paper.

여기서도 BERT는 압도적인 성능을 보여주며, 사람의 능력과 비슷한 성능을 보여주기까지 하였다.

Ablation studies

논문에서는 이전까지 핵심 요소들이라고 설명하였던 부분들을 하나씩 제거해가며 Ablation study를 진행하였다. 이를 통해 논문에서 제시한 요소들의 중요성을 역설한다.

Effect of Pre-training Tasks

저자들은 BERT의 deep bidirectionality의 중요성을 역설하기 위해 BERT의 핵심 요소들이었던 MLM과 NSP를 하나씩 제거하면서 실험을 진행하였다. BERT-BASE를 기반으로 한 두 가지 model을 추가적으로 실험하는데 다음과 같다.

- **MLM은 사용, NSP 미사용 (No NSP)**
- **MLM을 사용하지 않고 대신 left-to-right을 사용, NSP 미사용 (LTR & No NSP)**

그 결과는 다음과 같다.

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.

결과에서 확인할 수 있듯이, MLM과 NSP가 제거될 때마다 성능이 하락하는 것을 확인할 수 있다.

자세히 살펴보면, 먼저 NSP를 미적용한 model의 경우 NLI task(MNLI, QNLI, SQuAD)에서의 성능이 BERT-BASE에 비해 매우 저조함을 확인할 수 있다. 즉, **NSP가 두 문장 사이의 관계를 매우 잘 이해하도록 도와주는 것을 알 수 있다.**

이후, No NSP와 LTR & No NSP를 비교하여 MLM이 model에 미치는 영향을 확인해보자. **MLM 대신 LTR을 사용하게 되면 전반적으로 성능이 크게 하락하는 것을 확인할 수 있다.**

특이한 점은 randomly initialized Bidirectional LSTM을 model의 맨 위에 추가했을 때이다. 오히려 추가하기 전보다 성능이 하락했지만, 예외적으로 SQuAD 결과에서는 성능을 회복하였다. 그러나 회복한 성능도 MLM을 사용하였을 때보다 좋지 않으며, 결과적으로 Bidirectional LSTM의 경우에는 GLUE task에서의 성능에 악영향을 미치는 것을 확인할 수 있다.

Effect of Model Size

이어서, 저자들은 모델의 크기를 다르게 하면서 성능을 측정해보았다. 해당 결과는 다음과 같다.

(해당 결과는 각각 5번씩 random restart of fine-tuning 한 이후, 5개의 Dev set accuracy의 평균이다)

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.

모든 task에 대해 model size가 커질수록, 성능도 함께 향상되는 것을 확인할 수 있다. 또한, MRPC dataset과 같이, **downstream task를 수행하는 dataset의 크기가 작고 pre-training task와 상당히 다른 downstream task에서도 충분히 좋은 성능을 내는 것을 확인할 수 있다.**

논문에서는 기존에 제시된 model들에 비해 상당히 큰 model에서 성능 발전이 이루어진 것에 대해 이야기한다.

그 당시에도 machine translation이나 language modeling같이 large-scale task에서는 model size의 향상이 성능 향상을 이끈다는 것은 알려져 왔었다. 그러나, 저자들은 본 연구가 실제로 model이 충분한 pre-training을 거쳤을 경우 매우 큰 model size로 확장하는 것이 small scale task에서도 효과적이라는 것을 입증하는 첫 사례라고 주장한다.

그러면서 선행 연구들의 경우를 제시한다. 선행되었던 연구들에서는 특정 수준 이상의 model size부터 성능 향상이 크게 이루어지지 않았다. 논문에서는 이러한 선행 연구들의 경우에는 feature-based approach이며, 본 연구에서는 이와는 다르게 **model이 downstream task에 직접적으로 fine-tuned 되며, randomly initialized additional parameter를 최소한으로 할 때, task specific model은 downstream task data가 매우 작은 상황에서도 더 larger 하고 더 expressive 한 pre-trained representation로부터 이득**을 볼 수 있을 것이라고 가정한다고 밝힌다.

Feature-based Approach with BERT

지금까지 제시된 BERT의 결과들은 pre-training 된 model에 simple classification layer를 추가하고, downstream task에서 모든 parameter가 공동으로 fine-tuned 되는 fine-tuning approach를 사용하였다.

그러나, feature-based approach에도 명확한 장점이 있다.

먼저, transformer의 encoder로 쉽게 표현되지 않는 task가 존재하기에, 이런 경우에는 task-specific architecure가 추가되어야만 한다. 또한, **feature-based approach는 computational benefit이 존재**한다. training data의 expensive representation을 사전에 구해놓은 뒤, 이러한 representation 위에서 cheaper model을 통해 여러 번 실험을 진행할 수 있기 때문이다.

그래서, 논문에서는 BERT에서 두 방법론을 비교하고자 Named Entity Recognition(NER) task에 fine-tuning approach와 feature-based approach 모두 적용하여 실험을 진행하였다. feature-based approach의 경우, fine-tuning을 거치지 않은 BERT의 hidden layer들 중, 추출하는 layer의 개수를 다르게 하여 bidirectional LSTM의 input embedding으로 사용하였다.

결과는 다음과 같다.

System	Dev F1	Test F1
ELMo (Peters et al., 2018a)	95.7	92.2
CVT (Clark et al., 2018)	-	92.6
CSE (Akbik et al., 2018)	-	**93.1**
Fine-tuning approach		
$\text{BERT}_{\text{LARGE}}$	96.6	92.8
$\text{BERT}_{\text{BASE}}$	96.4	92.4
Feature-based approach ($\text{BERT}_{\text{BASE}}$)		
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	-

Table 7: CoNLL-2003 Named Entity Recognition results. Hyperparameters were selected using the Dev set. The reported Dev and Test scores are averaged over 5 random restarts using those hyperparameters.

당연하게도, fine-tuning approach를 적용했을 때가 성능이 좋지만, pre-training 된 BERT의 마지막 4개의 hidden layer의 concat을 feature-based approach로 사용했을 때의 성능이 Fine-tuning approach를 적용했을 때와 큰 차이가 없음을 확인할 수 있다.

이러한 결과를 통해, **BERT는 fine-tuning approach와 feature-based approach 모두 효과적임을 알 수 있다.**