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

Abstract

- BERT : Bidirectional Encoder Representations from Transformer
    - 논문의 제목에서 볼 수 있듯이, 본 논문은 *“Attention is all you need(Vaswani et al., 2017)”*(arxiv)에서 소개한 **Transformer** 구조를 활용한 **Language Representation**에 관한 논문이다.
    - **Transformer**에 대해서 간단한 리뷰
- BERT는 기본적으로, wiki나 book data와 같은 대용량 **unlabeled data**로 모델을 미리 학습 시킨 후, 특정 *task*를 가지고 있는 *labeled data*로 **transfer learning**을 하는 모델이다.
- BERT 이전에 이러한 접근 방법을 가지는 모델이 몇 가지 있었다. 대용량 unlabeld corpus를 통해 **language model**을 학습하고, 이를 토대로 뒤쪽에 특정 *task*를 처리하는 network를 붙이는 방식(ELMo, OpenAI GPT…)

- 하지만 BERT 논문에서는 이전의 모델의 접근 방식은 **shallow bidirectional** 또는 **unidirectional**하므로 language representation이 부족하다고 표현했다 .
- 게다가 BERT는 특정 task를 처리하기 위해 새로운 network를 붙일 필요 없이, BERT 모델 자체의 **fine-tuning**을 통해 해당 task의 *state-of-the-art(SOTA)*를 달성했다고 한다.

1. Introduction

- Introduction에서는 BERT와 비슷한 접근 방식을 가지고 있는 기존 model에 대한 개략적인 소개를 한다.

- Language model pre-training은 여러 NLP task의 성능을 향상시키는데에 탁월한 효과가 있다고 알려져 있다. (Dai and Le, 2015; Peters et al., 2018, 2018; Radford et al., 2018; …)
- 이러한 NLP task는 token-level task인 Named Entity Recognition(NER)에서부터 SQuAD question answering task와 같은 task까지 광범위한 부분을 커버한다.
- 이런 **pre-trained language representation**을 적용하는 방식은 크게 두 가지 방식이 있다. 하나는 **feature-based** 또 다른 하나는 **fine-tuning** 방식이다.
    - **feature-based approach** : 특정 task를 수행하는 network에 pre-trained language representation을 추가적인 feature로 제공. 즉, 두 개의 network를 붙여서 사용한다고 보면 됩니다. 대표적인 모델 : ELMo(Peters et al., 2018)
    - **fine-tuning approach** : task-specific한 parameter를 최대한 줄이고, pre-trained된 parameter들을 downstream task 학습을 통해 조금만 바꿔주는(fine-tuning) 방식. 대표적인 모델 : Generative Pre-trained Transformer(OpenAI GPT) (Radford et al., 2018)
- 앞에 소개한 ELMo, OpenAI GPT는 pre-training시에 동일한 **objective funtion**으로 학습을 수행, 하지만 **BERT**는 새로운 방식으로 **pre-trained Language Representation**을 학습했고 이것은 매우 효과적이었다.

BERT의 pre-training 방법론

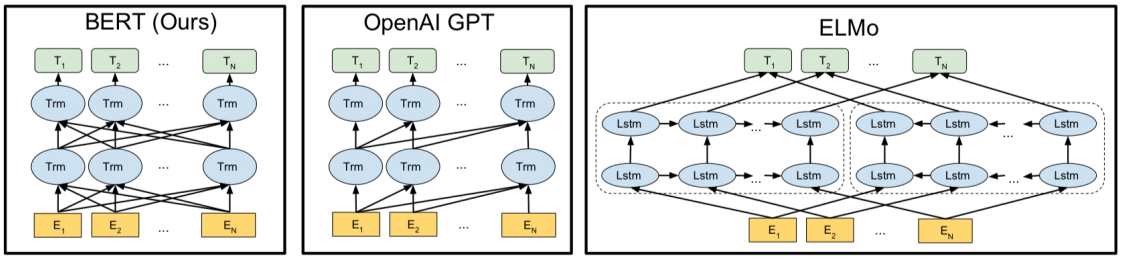


그림1. BERT, GPT, ELMo (출처 : BERT 논문)

- **BERT** pre-training의 새로운 방법론은 크게 2가지로 나눌 수 있다. 하나는 **Masked Language Model(MLM)**, 또 다른 하나는 N**ext Sentence Prediction(NSP)**이다.
  - 기존 방법론 : 앞에 소개한 ELMo, OpenAI GPT는 일반적인 language model을 사용하였다. 일반적인 language model이란, 앞의 n 개의 단어를 가지고 뒤의 단어를 예측하는 모델을 만드는 것이다(n-gram). 하지만 이는 필연적으로 **unidirectional**할 수 밖에 없고, 이러한 단점을 극복하기 위해 **ELMo**에서는 **Bi-LSTM**으로 양방향성을 가지려고 노력하지만, 굉장히 **shallow**한 양방향성 (단방향 concat 단방향)만을 가질 수 밖에 없었다(*그림1*).
  - **Masked Language Model(MLM)** : MLM은 input에서 무작위하게 몇 개의 token을 mask 시킨다. 그리고 이를 **Transformer** 구조에 넣어서 주변 단어의 context만을 보고 mask된 단어를 예측하는 모델이다. **OpenAI GPT**도 **Transformer** 구조를 사용하지만, 앞의 단어들만 보고 뒷 단어를 예측하는 **Transformer decoder**구조를 사용한다(*그림1*). 이와 달리 **BERT**에서는 input 전체와 mask된 token을 한 번에 **Transformer encoder**에 넣고 원래 token 값을 예측하므로(*그림1*) **deep bidirectional** 하다고 할 수 있다. BERT의 MLM에 대해서는 뒷장의 Pre-training Tasks에서 더 자세히 설명하겠다.
  - N**ext Sentence Prediction(NSP)** : 이것은 간단하게, 두 문장을 pre-training시에 같이 넣어줘서 두 문장이 이어지는 문장인지 아닌지 맞추는 것이다. pre-training 시에는 50:50 비율로 실제로 이어지는 두 문장과 랜덤하게 추출된 두 문장을 넣어줘서 BERT가 맞추게 시킨다. 이러한 task는 실제 Natural Language Inference와 같은 task를 수행할 때 도움이 된다.

2. Related Work

- ELMo, OpenAI GPT와 같은 모델이 존재하고, 앞에서 충분히 소개하였기 때문에 생략. 자세한 내용에서는 BERT 논문을 참고하셈.

3. BERT

- BERT의 아키텍처는 **Attention is all you need**에서 소개된 **Transformer**를 사용하지만, pre-training과 fine-tuning시의 아키텍처를 조금 다르게 하여 **Transfer Learning**을 용이하게 만드는 것이 핵심!!!!!.

3.1 Model Architecture

- **BERT**는 *transformer* 중에서도 **encoder** 부분만을 사용한다. 이에 대한 자세한 내용은 Vaswani et al (2017) 또는 tensor2tensor의 transformer를 참고하셈.
- **BERT**는 모델의 크기에 따라 *base* 모델과 *large* 모델을 제공한다.
  - **BERT_base** : L=12, H=768, A=12, Total Parameters = 110M
  - **BERT_large** : L=24, H=1024, A=16, Total Parameters = 340M
  - L : transformer block의 layer 수, H : hidden size, A : self-attention heads 수, feed-forward/filter size = 4H
- 여기서 **BERT_base** 모델의 경우, **OpenAI GPT**모델과 *hyper parameter*가 **동일하다**. 여기서 BERT의 저자가 의도한 바는, 모델의 하이퍼 파라미터가 동일하더라도, **pre-training concept**를 바꾸어 주는 것 만으로 훨씬 높은 성능을 낼 수 있다는 것을 보여주고자 하는 것 같다고 생각한다(내 생각).
- **OpenAI GPT**모델의 경우 *그림1*에서 볼 수 있듯, next token 만을 맞추어내는 기본적인 *language model* 방식을 사용하였고, 그를 위해 **transformer decoder**를 사용한다. 하지만 **BERT**는 **MLM**과 **NSP**를 위해 **self-attention**을 수행하는 **transformer encoder**구조를 사용했음을 알 수 있다.
- 실제로 대부분의 **NLP task SOTA**는 **BERT_large**모델로 이루어 냈다.

3.2 Input Representation

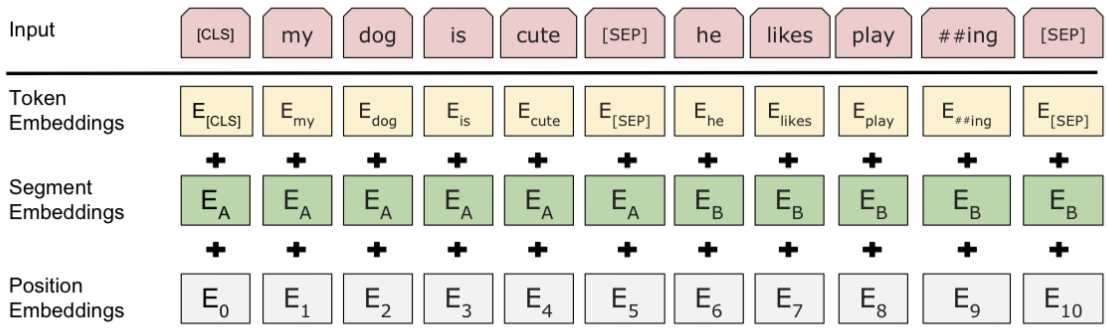


그림2. bert input representation (출처: BERT 논문)

- BERT의 input은 *그림 2*와 같이 3가지 embedding 값의 합으로 이루어져 있다.

- **WordPiece embedding**을 사용한다. BERT english의 경우 30000개의 token을 사용하였다.[추가설명]
- *그림 2*에서 볼 수 있듯이, *Position embedding*을 사용한다. 이는 **Transformer**에서 사용한 방식과 같으며, jalammer의 블로그 글을 참고하면 position embedding 뿐만 아니라 transformer의 전체적인 구조를 이해하는데 도움이 될 수 있다.
- 모든 sentence의 첫번째 token은 언제나 `[CLS]`(special classification token) 이다. 이 `[CLS]` token은 transformer 전체층을 다 거치고 나면 token sequence의 결합된 의미를 가지게 되는데, 여기에 **간단한 classifier를 붙이면 단일 문장, 또는 연속된 문장의 classification을 쉽게 할 수 있게 된다**. 만약 classification task가 아니라면 이 token은 무시하면 된다.
- **Sentence pair는 합쳐져서 single sequence**로 입력되게 된다. 각각의 Sentence는 실제로는 수 개의 sentence로 이루어져 있을 수 있다(eg. QA task의 경우 `[Question, Paragraph]`에서 Paragraph가 여러 개의 문장). 그래서 두 개의 문장을 구분하기 위해, 첫째로는 `[SEP]` **token 사용**, 둘째로는 **Segment embedding**을 사용하여 앞의 문장에는 `sentence A embedding`, 뒤의 문장에는 `sentence B embedding`을 더해준다.(모두 고정된 값)
- 만약 문장이 하나만 들어간다면 `sentence A embedding`만을 사용.

3.3 Pre-training Tasks

- 기존의 **ELMO**나 **GPT**는 **left to right** or **right to left** Language Model을 사용하여 **pre-training**을 하지만, **BERT**는 이와 다르게 2가지의 새로운 **unsupervised prediction task**로 **pre-training**을 수행한다.

3.3.1 Task #1: Masked LM

- Introduction의 pre-training 방법론에서 설명한 내용과 동일한 내용.
- 이번 장에서는 **MLM**이 구체적으로 어떤 식으로 수행되는 지에 대해서 설명하겠음.

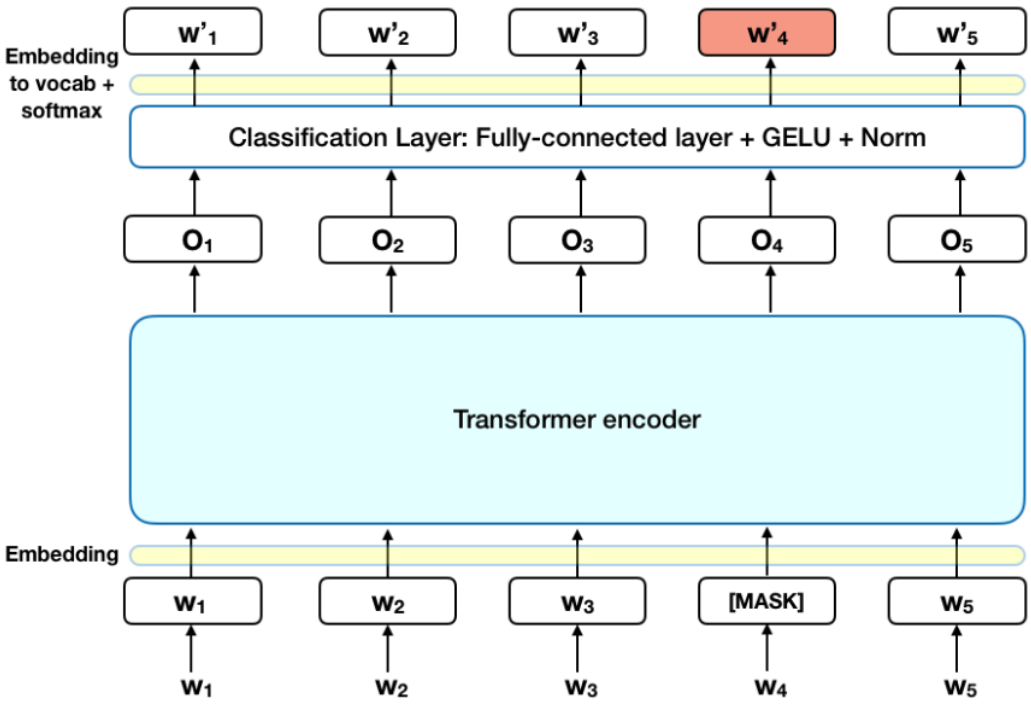


그림3. BERT Masked Language Model (출처: rani horev's blog : BERT explained)

- 그림 3에서 볼 수 있듯, 일단 **단어 중의 일부**를 `[MASK]` token 으로 바꾸어 준다. 바꾸어 주는 비율은 **15%** 이다.
- 그리고 **plain text**를 **tokenization**하는 방법은 input representation에서 설명한 바와 같이 WordPiece(Wu et al., 2016)를 사용한다.
- 이를 통하여 **LM**의 **left-to-right** (혹은 r2l)을 통하여 문장 전체를 **predict**하는 방법론과는 달리, `[MASK]` token 만을 **predict**하는 **pre-training task**를 수행한다.
- 이 `[MASK]` token은 **pre-training**에만 사용되고, **fine-tuning**시에는 사용되지 않는다. 해당 token을 맞춰 내는 task를 수행하면서, **BERT**는 문맥을 파악하는 능력을 길러내게 된다.

- 15%의 `[MASK]` token을 만들어 낼 때, 몇 가지 추가적인 처리를 더 해주게 된다. 그것은 다음과 같다.
  - 80%의 경우 : token을 `[MASK]`로 바꾼다. eg., `my dog is hairy -> my dog is [MASK]`
  - 10%의 경우 : token을 random word로 바꾼다. eg., `my dog is hariy -> my dog is apple`
  - 10%의 경우 : token을 원래의 단어로 그대로 놔둔다. 이는 실제 관측된 단어에 대한 표상을 bias해주기 위해 실시한다.

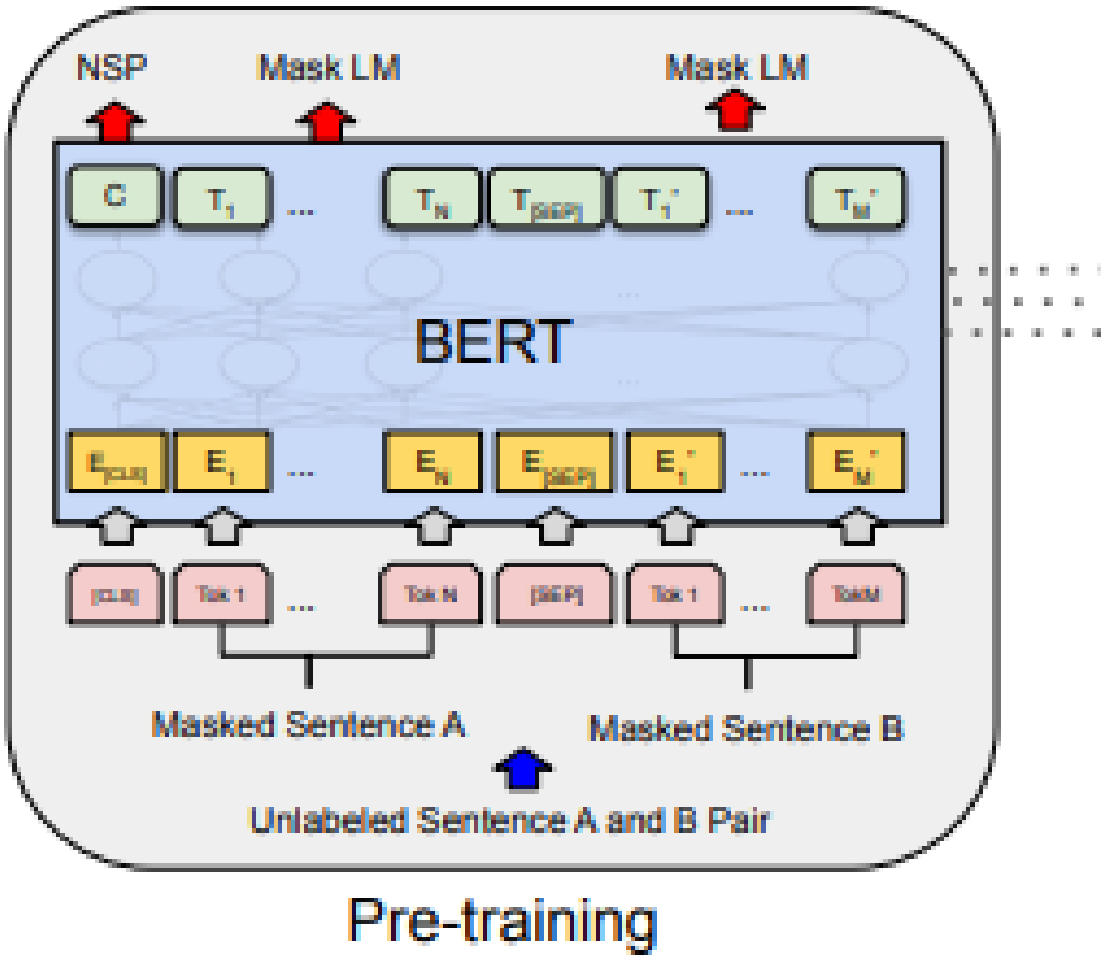
- **pre-trained** 되는 **Transformer encoder**의 입장에서는 어떤 단어를 **predict**하라고 하는 건지, 혹은 **random word**로 바뀌었는지 알 수 없다. **Transformer encoder**는 그냥 모든 token에 대해서 **distributional contextual representation**을 유지하도록 강제한다.
- 또한 **random word**로 바꾸는 것 때문에 모델의 **language understanding**능력에 해를 끼친다고 생각 할 수 있지만, 바뀌는 부분이 1.5%(15%의 10%)에 불과하므로, 해를 끼치지 않는다.
- 또한 **MLM**은 보통의 **LM** 보다 **converge**하는 데에 많은 training step이 필요하지만, **empirical**하게는 **LM**보다 훨씬 빠르게 좋은 성능을 낸다. 이는 Section 5.3에서 자세히 설명하겠다.

3.3.2 Task #2: Next Sentence Prediction

- 이 task 또한 Introduction의 pre-training 방법론에서 설명한 내용.
- 이 **pre-training task** 수행하는 이유는, 여러 중요한 **NLP task**중에 **QA**나 **Natural Language Inference**(**NLI**)와 같이 두 문장 사이의 관계를 이해하는 것이 중요한 것이 있기 때문이다. 이들은 **language modeling**에서 **capture**되지 않는다.
- 그래서 **BERT**에서는 corpus에서 두 문장을 이어 붙여 이것이 원래의 corpus에서 바로 이어 붙여져 있던 문장인지를 맞추는 **binarized next sentence prediction task**를 수행한다.
    - 50% : sentence A, B가 실제 next sentence
    - 50% : sentence A, B가 corpus에서 random으로 뽑힌(관계가 없는) 두 문장
    - 예를 들어

    > Input = [CLS] the man went to [MASK] store [SEP] he bought a gallon [MASK] milk [SEP] LABEL = IsNext
    >
    > Input = `[CLS] the man [MASK] to the store [SEP] penguin [MASK] are flight ##less birds [SEP]` Label = `NotNext`

- **pre-training**이 완료되면, 이 task는 97~98%의 accuracy를 달성했다. 이러한 간단한 task를 부여해도, **QA와 NLI**에 굉장히 의미 있는 성능 향상을 이루어 냈다. 이는 section5. Ablation Studies에서 자세히 설명이 되어있다.



Pre-training

3.4 Pre-training Procedure

- **pre-training**의 기본적인 절차는 **LM**에서 수행하는 것과 같다.
- **BERT_english**의 경우 BookCorpus (Zhu et al., 2015) (800M words)와 English Wikipedia (2,500M words)를 사용하였다. Wikipedia 데이터에서는 **text passage**만 추출하여 사용했다고 한다. 이유는, **long contiguous sequence**만을 학습 시키고자 해서 이다.

- **input pre-processing**
  - 먼저, **NSP**를 위해 sentence를 뽑아서 **embedding A, B**를 먹여준다. 물론 50%는 진짜 next sentence, 나머지는 random sentence를 사용한다.
  - 이 모든 토큰이 합쳐진 길이는 512개 이하여야 한다. (OOM 때문)
  - 이후 **Masking 작업**을 해준다.
- **pre-training 시의 Hyper Parameters**

- **batch size** : 256 sequences (256 sequences * 512 tokens = **128,000 tokens/batch**) for 1,000,000 steps -> 3.3 billion word corpus의 40 epochs
- **Adam optimizer**, learning rate : 1e-4, L2 weight decay of 0.01, learning rate warmup over the first 10,000 steps, linear decay of the learning rate

  β1=0.9

  β1=0.9

  β2=0.999

  β2=0.999
- **Dropout prob**: 0.1 for all layers
- using **gelu** activation Hendrycks and Gimpel, 2016
- **BERT_base** - 4 TPUs, **BERT_large** - 16 TPUs를 사용하여 4일 동안 학습

3.5 Fine-tuning Procedure

- **sequence-level classification tasks**에 대해서는 **BERT fine-tuning**과정이 매우 straightforward하다.
  - input sequence에 대해서 일정한 차원 수의 representation 결과를 얻고 싶기 때문에, `[CLS]` token의 **Transformer output**값을 사용한다.
  - `[CLS]` token의 벡터는 H차원을 가짐.

$$C \in \mathbb{R}^H$$

  - 여기서 **classify하고 싶은 갯수(K)에 따라 classification layer**를 붙여줌. classification layer :

$$W \in \mathbb{R}^{K \times H}$$

  - label probabilities는 **standard softmax**로 계산 된다.

$$P = softmax(CW^T)$$

  - W matrix와 BERT의 모든 파라미터가 같이 fine-tuning 된다.
- **span-level, token-level prediction tasks**의 경우에는, 위의 과정에서 약간 변형시켜 fine-tuning이 진행된다. 자세한 내용은 Section 4에서 설명하도록 하겠다.

- **fine-tuning시의 Hyper-Parameter**
  - 몇 가지를 제외하고는 pre-training때의 hyper parameter와 대부분 동일하다.
  - 다른 점은 **batch size, learning rate, trainig epochs 수** 이다.
  - **optimal hyperparameter**는 **task마다 달라지지만**, 다음에 제시하는 것을 사용하면 대부분 잘 학습된다고 한다.

    > **Batch size**: 16, 32 / **Learning rate(Adam)**: 5e-5, 3e-5, 2e-5 / **Number of epochs** : 3, 4

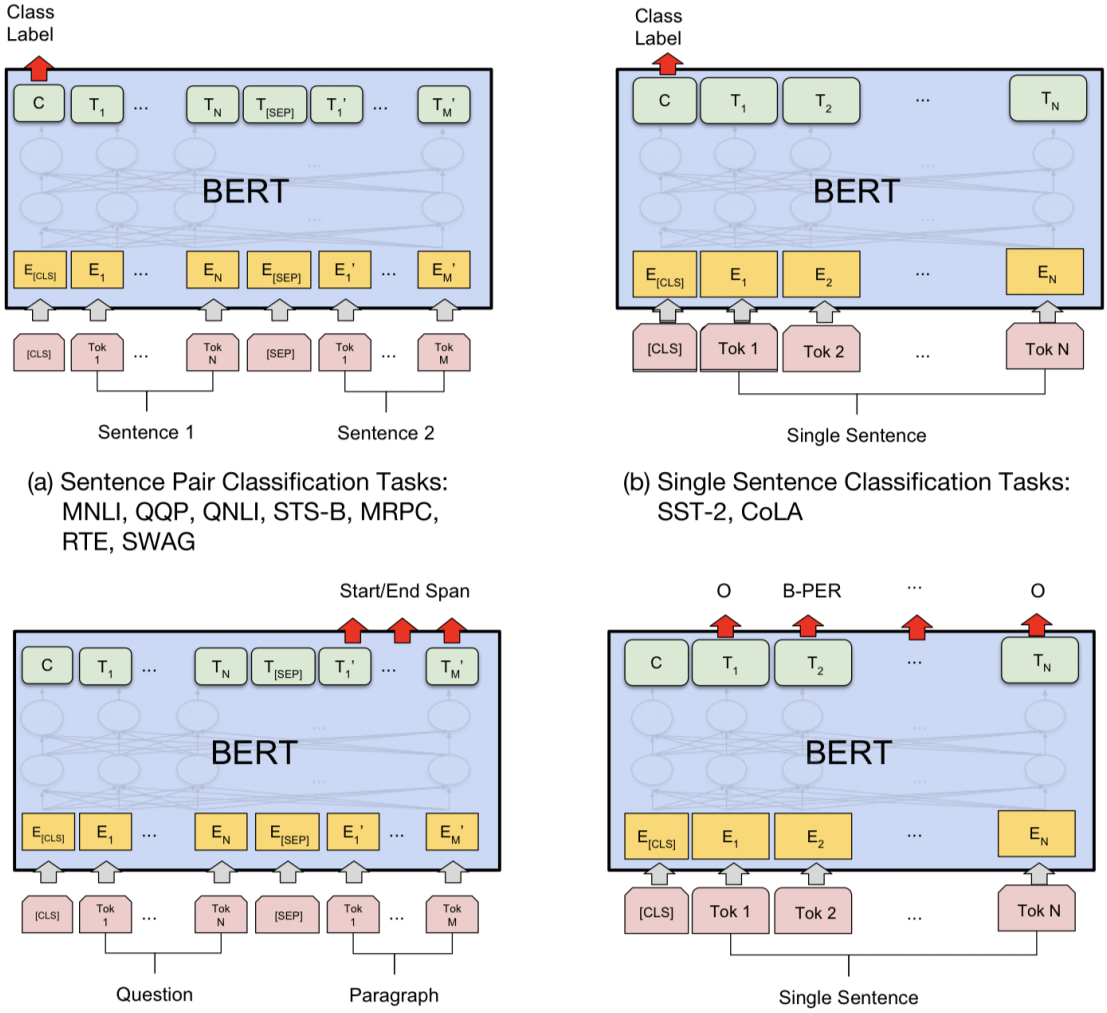
- fine-tuning 시의 **dataset의 크기가 클수록** hyperparameter에 **영향을 덜 받고 잘 training 됨**을 관측할 수 있었다고 한다.
- **Fine-tuning**은 굉장히 빠르게 학습되며(pre-training에 비해) 이로 인해 최적의 hyperparameter 탐색을 exhaustive search(노가다)로 찾아내도 무방하다.

3.6 Comparison of BERT and OpenAI GPT

- **OpenAI GPT**와 **BERT**는 **transformer**구조를 사용한다는 점에 있어서 공통점을 갖지만, **BERT**가 훨씬 좋은 성능을 갖는다. 이 차이는 앞에 설명한 MLM task, NSP task를 수행하는 것과 별개로 또 더 다른 점을 갖기에 생긴다.
  - **GPT**의 경우 **BookCorpus(800M words)**만을 pre-training에 사용; **BERT**는 거기에 + **Wikipedia(2500M words)** 사용
  - **GPT**는 `[SEP]`과 `[CLS]` token을 fine-tuning시에만 추가하여 학습; **BERT**는 pre-training시에도 학습 **(NSP task가 존재하기 때문에 가능)**
  - **GPT** : 32,000 words/batch for 1M steps ; **BERT** : 128,000 words/batch for 1M steps
  - **GPT**는 모든 fine-tuning을 수행할 때 learning rate : 5e-5를 사용; **BERT**는 **task-specific**하게 조절하여 사용
- 위에 나열된 차이점에 대한 효과는 **ablation experiments**가 수행된 Section 5.1에 설명되어 있다.

4. Experiments

- 이번 섹션에서는, **11개의 NLP tasks**에 대한 **BERT fine-tuning**에 대해서 설명하겠다.



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

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

(c) Question Answering Tasks: SQuAD v1.1

(d) Single Sentence Tagging Tasks: CoNLL-2003 NER

그림 4. BERT experiments result (출처: BERT 논문)

- **그림 4**를 참고하면, 각 fine-tuning 유형마다 어떻게 학습하는지를 알아 볼 수 있다.
- (a), (b)는 sequence-level task, (c)와 (d)는 token-level task이다. sequence-level task의 경우는 3.5 Fine-tuning Procedure에서 설명을 했었다.
- (c)의 QA task경우는, **Question에 정답이 되는 Paragraph의 substring을 뽑아내는 것**이므로, `[SEP]` token 이후의 token들에서 **Start/End Span**을 찾아내는 task를 수행한다.
- (d)의 경우는 **Named Entity Recognition(NER)**이나 **형태소 분석**과 같이 single sentence에서 각 토큰이 어떤 class를 갖는지 **모두 classifier** 적용하여 정답을 찾아낸다.

4.1 GLUE Datasets

- 해당 섹션은 The General Language Understanding Evaluation(GLUE) benchmark (Wang et al., 2018)에 대한 설명이 이어져 있다.

4.1.1 GLUE Results

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	79.9	90.4	36.0	73.3	84.9	56.8	71.0
OpenAI GPT	82.1/81.4	70.3	88.1	91.3	45.4	80.0	82.3	56.0	75.2
$\text{BERT}_{\text{BASE}}$	84.6/83.4	71.2	90.1	93.5	52.1	85.8	88.9	66.4	79.6
$\text{BERT}_{\text{LARGE}}$	**86.7/85.9**	**72.1**	**91.1**	**94.9**	**60.5**	**86.5**	**89.3**	**70.1**	**81.9**

그림5. GLUE results (출처: BERT 논문)

- 쉽게 말해, **모든 task에 대해 SOTA를 달성한다**.
- 특히 인상적인 것은 **데이터 크기가 작아도 fine-tuning 때는 좋은 성능을 낼 수 있다**는 것이다.
- 그리고 **BERT_large**가 **BERT_base**에 비해 훨씬 좋은 성능을 낸다.(dataset이 작은 task의 경우에도!) **BERT**의 모델 크기에 따른 성능은 Section 5.2에서 살펴보겠다.

4.2 SQuAD v1.1

- 기존 **SQuAD** dataset의 경우에는 **GLUE** dataset을 **BERT**에 **fine-tuning**할 때와 방식이 좀 다르다. GLUE dataset의 경우에는 task가 **sequence classification**이지만, SQuAD는 **질문**과 **지문**이 주어지고, 그 중 **substring인 정답**을 맞추는 task이다.
- 이러한 차이를 **BERT**는 아주 간단한 방법론으로 극복한다.
  - 일단 **질문을** `A embedding`**, 지문을** `B embedding` 으로 처리하고, **지문에서 정답이 되는 substring의 처음과 끝을 찾는 task**로 문제를 치환한다.
  - `start vector` (S) 와 `end vector` (E)를 **fine-tuning**중에 학습하여, **지문의 각 token**들과 **dot product**하여 **substring**을 찾아낸다. (그림 4 (c) 참조)

$$S \in \mathbb{R}^H$$

$$E \in \mathbb{R}^H$$

- 이런 접근법으로 **BERT_large**의 경우 기존의 모든 시스템을 **wide margin**을 두고 최고 성능 달성.

4.3 Named Entity Recognition

- **token tagging task**를 평가해보기 위해서 **CoNLL 2003 Named Entity Task**를 fine-tuning 해봤다.
  - 해당 데이터셋은 200k개의 words로 이루어져 있고, 각각의 단어들은 `Person, Organization, Location, Miscellaneous, Other`로 태깅되어 있다.
- 각 토큰마다 **classifier**를 붙여서 위의 제시된 **class**에 대해 판별을 한다.
- 각각의 **prediction**은 주위의 **prediction**에 영향을 받지 않는다. (CRF나 autoregressive를 사용하지 않음!)
- 이 task또한 **SOTA**를 달성.

4.4 SWAG

- The Situations With Adversarial Generations(SWAG) 데이터셋은 113k sentence-pair로 이루어져 있으며, **grounded common-sense inference**를 측정하기 위해 사용했다.
- 앞 문장이 주어지고, 보기로 주어지는 4 문장 중에 가장 잘 이어지는 문장을 찾는 task 이다.

> A girl is going across a set of monkey bars. She (i) jumps up across the monkey bars. (ii) struggles onto the bars to grab her head. (iii) gets to the end and stands on a wooden plank. (iv) jumps up and does a back flip.

- 이 데이터 셋을 **BERT**에 적용하는 것은 **GLUE** dataset 적용 법과 비슷하다. 4개의 가능한 input sequences를 구성한다. 이는 given sentence(`sentence A`)와 possible continuation(`sentence B`)을 `concat`한 것들이다.
- 해당 task specific한 벡터(V)를 학습시키고, input sequences의 값들을 합산한 벡터(Ci)를 **dot product**하여 softmax 한다.

$$V \in \mathbb{R}^H$$

$$C_i \in \mathbb{R}^H.$$

- 이 또한 사람을 능가하는 **SOTA** 달성.

5. Ablation Studies

- 해당 챕터에서는 이전 챕터에서 **중요한 요소**라고 설명했던 부분을 **하나씩 제거**하며 요소들의 중요함을 파악해 보고 있다.
- **논문에서 가장 중요한 챕터**라고 볼 수 있음(내 생각).

5.1 Effect of Pre-training Tasks

- 5.1에서는 이전 3.3 Pre-training Tasks에서 소개한 2가지 task를 하나씩 제거하면서 각각의 task의 효과를 알아본다.
- BERT_base와 동일한 hyperparameter로 실험을 진행하지만 **ablation**한 두 가지 다른 모델로 실험을 진행한다.
  - **No NSP**: MLM은 사용하지만, 다음 문장 예측 (**NSP**)를 없앤 모델
  - **LTR & No NSP** : **MLM** 대신 **Left-to-Right (LTR)** 을 사용하고, **NSP**도 없앤 모델, 이는 **OpenAI GPT**모델과 완전히 동일하지만, 더 많은 트레이닝 데이터를 사용했다.

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

그림6. pre-training task ablation result (출처:BERT 논문)

- 표를 보면, pre-training task를 하나라도 제거하면 성능이 굉장히 떨어지는 것을 볼 수 있다.
- **No NSP**의 경우에는 **NLI**계열의 task에서 **성능이 많이 하락**하게 되는데, 이는 **NSP task**가 문장 간의 논리적인 구조 파악에 중요한 역할을 하고 있음을 알 수 있다.
- MLM대신 LTR을 쓰게 되면 성능 하락은 더욱 더 심해지게 된다. **BiLSTM을 더 붙여도**, MLM을 쓸 때보다 성능이 하락하는 것으로 보아, **MLM task**가 더 **Deep Bidirectional**한 것임을 알 수 있다.

5.2 Effect of Model Size

- 간단하게 말해서, 측정한 데이터 셋에서는 모두 **모델이 커질수록, 정확도가 상승함**을 볼 수 있다.

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

그림7. Ablation over BERT model size (출처:BERT 논문)

- 수 년동안 알려져 왔듯, 번역 task나 language modeling과 같은 large-scale task는 모델 사이즈가 클 수록 성능은 계속 상승한다.
- 특히 BERT의 경우에는, downstream task를 수행하는 dataset의 크기가 작아도, pre-training덕분에, model의 크기가 클 수록 정확도는 상승함을 볼 수 있다.

5.3 Effect of Number of Training Steps

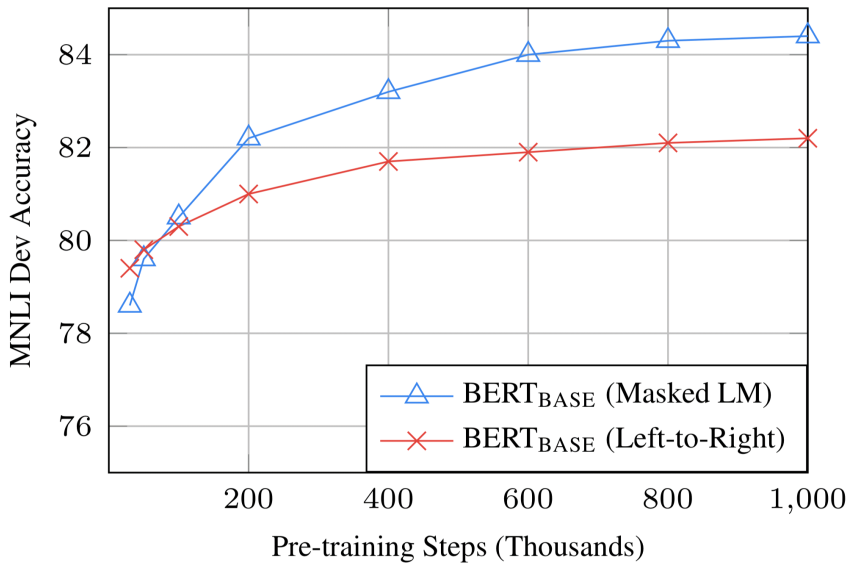


그림8. Ablation over number of training steps (출처:BERT 논문)

- 그림 8에서는 **MNLI** Dev accuracy를 pre-training step에 따라 정리했다. 세부사항은 문답형식으로 진행하겠다.

- Question 1. fine-tuning 단계에서 높은 정확도를 얻으려면, pre-training 단계에서 많은 training step이 필요합니까?
  - A : 그렇다. 0.5M step에 비해 1M step때 accuracy가 거의 1.0% 상승함을 볼 수 있다.
- Question 2. 3.3.1 Task #1: Masked LM의 마지막 부분에서 시사한 바와 같이, **MLM**으로 학습하면 15%의 단어만 맞추는 것으로 학습을 진행하기 때문에, LTR보다 **수렴속도가 훨씬 느리지** 않습니까?
  - A : **수렴속도가 조금 느린 것은 사실**이지만, **LTR보다 훨씬 먼저 out-perform**성능이 나오게 된다.

5.4 Feature-based Approach with BERT

- 지금까지 BERT는 pre-training을 진행한 후, downstream task를 학습 할 때, 간단한 classifier를 부착해서 모든 layer를 다시 학습시키는 fine-tuning 방법을 사용하는 것만 설명했다.
- 하지만 BERT를 ELMO와 같이 **feature based approach**로도 사용할 수 있다.
- **Feature-based Approach**는 몇 가지 이점이 존재한다.
  - Transformer encoder는 모든 NLP task를 represent하지는 못하므로, 특정 NLP task를 수행할 수 있는 Network를 부착하여 쓸 수 있다.
  - Computational benefit을 얻을 수 있다.
- 해당 section에서는 BERT를 ELMO와 같이 마지막 레이어에 Bi-LSTM을 부착시켜, 해당 레이어만 학습 시키는 방법론을 사용했다.

Layers	Dev F1
Finetune All	96.4
First Layer (Embeddings)	91.0
Second-to-Last Hidden	95.6
Last Hidden	94.9
Sum Last Four Hidden	95.9
Concat Last Four Hidden	96.1
Sum All 12 Layers	95.5

그림9. Ablation using BERT with a feature-based approach (출처:BERT 논문)

- 그림 9에서 볼 수 있듯이, **Concat Last Four Hiddem**값을 사용하면, **Finetune All**과 단지 0.3 F1 score차이밖에 나지 않는다.
- 이를 통해, BERT는 **Feature-based Approach**에서도 효과적이라고 할 수 있다.

6. Conclusion

- 읽으면서 계속해서 느꼈지만, 굉장히 놀라운 논문이라고 생각한다. 특히 MLM과 NSP task를 생각해 낸 것이 굉장히 놀라웠다. 이러한 간단한 **intuition**을 가지고 NLP의 전 분야를 아우르는 SOTA를 달성해 낸 것에 대해서 놀랍고, 역시 나는 먼지와 같은 존재구나라는 것을 다시 한 번 깨달았다. 앞서 나왔던 트랜스포머의 디코더를 이용한 GPT의 단점을 빠르게 파악하고 트랜스포머의 인코더를 이용한 BERT를 만들어낸 것이 정말 대단하다고 생각한다.