

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



ctrl+alt+t 를 누르면 한 번에 여닫기를 할 수 있습니다.

개인적으로 정리하고 있던 자료와 합치다 보니, 다소 내용이 많은 점 참고 부탁 드립니다.
그리고 화이트 모드로 보는 것을 권장드립니다.



< 목차 >

- 0. 요약
 - 전반적으로 이런 내용을 다뤄요
- 1. BERT의 기본 컨셉
 - 1-1. BERT란
 - 1-2. BERT's Usefulness
- 2. BERT의 동작 방식 및 구조
 - 2-1. BERT의 이름 알아보기
 - 2-2. 동작 방식
 - 2-3. 구조
- 3. BERT의 사전 학습
 - 3-1. BERT의 입력 표현
 - 3-2. 사전 학습 전략
 - 3-3. 사전 학습 절차
- 4. 하위 단어 토큰화 알고리즘
 - 4-1. 하위 단어 토큰화를 하는 이유
 - 4-2. 바이트 쌍 인코딩(BPE)
 - 4-3. 바이트 수준 바이트 쌍 인코딩 (BBPE)
 - 4-4. 워드 피스
- 5. Code Review
 - Transformer(직접 구현)
 - BERT
- 6. 논문 리뷰
 - 1. Introduction (서론)
 - 2. Related Work (관련 연구)
 - 3. BERT (모델 설명)
 - 4. Experimental Results (실험 결과)
 - 5. Ablation Studies (성능 분석)

0. 요약

▼ 전반적으로 이런 내용을 다둬요

- 가장 널리 사용되는 고성능 텍스트 임베딩 모델인 BERT를 소개한다.
- BERT가 무엇이며 다른 임베딩 모델과 어떻게 다른지 이해하는데서부터 시작한 다음 BERT의 동작 방식과 구조를 살펴본다.
- 마스크 언어모델링(MLM)과 다음 문장 예측(NSP)이라는 두 가지 테스크 기반 BERT 모델이 어떻게 사전 학습을 진행하는지 알아본다.
- 하위단어 토큰화 알고리즘에 대해 알아본다.
- 앞 선 내용들을 이해하고, 논문을 다시 본다.

1. BERT의 기본 컨셉

▼ 1-1. BERT란

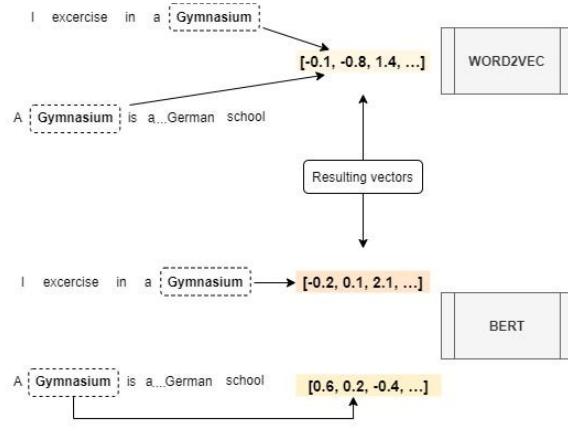
BERT(Bidirectional Encoder Representation from Transformer)는 구글에서 발표한 텍스트 임베딩 모델입니다.

NLP 분야에서 많은 기여를 해왔으며, BERT를 기반으로 한 다양한 모델들이 존재합니다.

▼ 1-2. BERT's Usefulness

이전의 **word2vec**과 같은 문맥 독립(context-free) 임베딩 모델은 해당 단어의 의미를 파악하기 힘들었습니다.

하지만 **BERT**는 모든 단어의 문맥 기반(context-based) 임베딩 모델로 **문맥 정보**를 고려할 수 있습니다.



6. Conclusion (결론)
7. 추후 방향성
8. 참고 링크
9. BERT 이전에 참고 할 만한 논문
10. BERT 이후 관련 논문
11. 한국어 관련 LLM 모음
12. 리뷰 후기

위의 그림을 보면 이해가 쉬울 것 같지만 아래의 예시를 통해 BERT를 조금 더 자세하게 컨셉을 이해해보도록 하겠습니다.

▼ 추가 예시

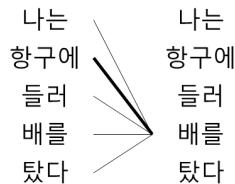
아래와 같이 두 문장이 있다고 하자.

A: 나는 항구에 들러 배를 탔다.

B: 나는 사과와 배를 먹었다.

두 문장에서 '배'라는 단어의 의미가 서로 다르다는 것을 알 수 있다.

BERT는 모든 단어의 문맥상 의미를 이해하기 위해 아래의 그림과 같이, 문장의 각 단어를 문장의 다른 모든 단어와 연결 시켜 이해합니다.



A 케이스의 경우, 항구와 함께 언급되었기에 해당 '배'의 의미는 ship이라는 의미로 이해할 수 있습니다.

또한 B케이스에서는 아마도 사과나 먹었다와 관련성을 가지고 pear라는 의미로 이해할 수 있습니다.

이렇게 문맥을 고려하면 **다의어** 및 **동음이의어**를 구분할 수 있게 됩니다.

2. BERT의 동작 방식 및 구조

▼ 2-1. BERT의 이름 알아보기

▼ B — Bidirectional

- RNN을 공부해보셨다면 양방향 RNN을 떠올리시면 쉽습니다.
- 트랜스포머 인코더는 원래 양방향으로 문장을 읽을 수 있습니다.
- 따라서 BERT는 기본적으로 트랜스포머에서 얻은 양방향 인코더 표현입니다.

▼ ER — Encoder + Representation

BERT는 이름에서 알 수 있듯이 트랜스포머 모델을 기반으로 하며, 인코더-디코더가 있는 트랜스포머 모델과 달리 인코더만 사용합니다.

문장을 트랜스포머 인코더에 입력하고, 문장의 각 단어에 대한 표현 벡터를 출력으로 반환하는 것을 확인하였습니다.

즉, 트랜스포머의 인코더는 BERT의 **표현 벡터**입니다.

- from

▼ T — Transformer

▼ Introduction

▼ Since 2016,

- In 2016, Google published their neural machine translation system(GNMT), which outperforms previous traditional MT system.

Google's Neural Machine Translation System.pdf

Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi
 yonghui,schuster,zhifengc,qvl,mnorouzi@google.com

Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Łukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, Jeffrey Dean

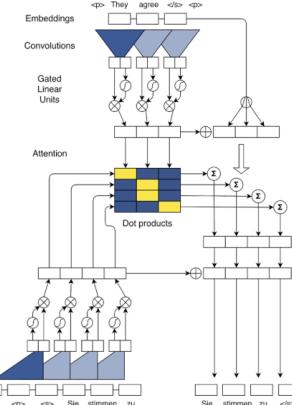
- However, RNN based Sequence-to-sequence reveals its limitations.

Table 10: Mean of side-by-side scores on production data

	PBMT	GNMT	Human	Relative Improvement
English → Spanish	4.885	5.428	5.504	87%
English → French	4.932	5.295	5.496	64%
English → Chinese	4.035	4.594	4.987	58%
Spanish → English	4.872	5.187	5.372	63%
French → English	5.046	5.343	5.404	83%
Chinese → English	3.694	4.263	4.636	60%

▼ Fully Convolutional Seq2Seq[Gehring et al.2017p

Convolutional Sequence to Sequence Learning.pdf



WMT'16 English-Romanian	BLEU
Sennrich et al. (2016b) GRU (BPE 90K)	28.1
ConvS2S (Word 80K)	29.45
ConvS2S (BPE 40K)	30.02

WMT'14 English-German	BLEU
Luong et al. (2015) LSTM (Word 50K)	20.9
Kalchbrenner et al. (2016) ByteNet (Char)	23.75
Wu et al. (2016) GNMT (Word 80K)	23.12
Wu et al. (2016) GNMT (Word pieces)	24.61
ConvS2S (BPE 40K)	25.16

WMT'14 English-French	BLEU
Wu et al. (2016) GNMT (Word 80K)	37.90
Wu et al. (2016) GNMT (Word pieces)	38.95
Wu et al. (2016) GNMT (Word pieces) + RL	39.92
ConvS2S (BPE 40K)	40.51

Table 1. Accuracy on WMT tasks compared to previous work. ConvS2S and GNMT results are averaged over several runs.

▼ Attention is all you need

Attention Is All You Need.pdf

▼ Transformer

- Encoder + decoder

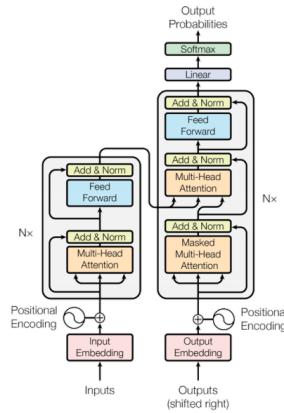


Figure 1: The Transformer - model architecture.

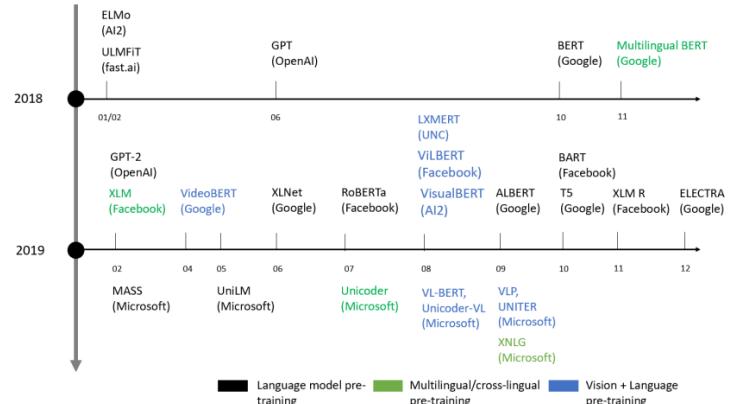
- 성능과 속도 모두 기준 모델을 압도

Table 2: The Transformer achieves better BLEU scores than previous state-of-the-art models on the English-to-German and English-to-French newstest2014 tests at a fraction of the training cost.

Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [18]	23.75			
Deep-Att + PosUnk [39]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [38]	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S [9]	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE [32]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble [39]		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble [38]	26.30	41.16	$1.8 \cdot 10^{20}$	$1.1 \cdot 10^{21}$
ConvS2S Ensemble [9]	26.36	41.29	$7.7 \cdot 10^{19}$	$1.2 \cdot 10^{21}$
Transformer (base model)	27.3	38.1		$3.3 \cdot 10^{18}$
Transformer (big)	28.4	41.8		$2.3 \cdot 10^{19}$

▼ Transformer and MLP

- Pretraining and finetuning (Transfer Learning) with Big-LM.



▼ Multi-head Attention

▼ Attention: Query Generation

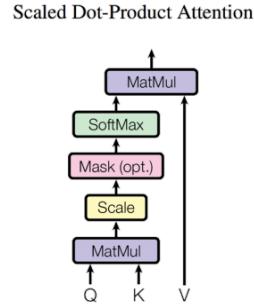
- Example



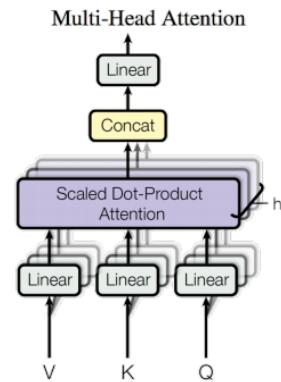
- 마음의 상태(state)를 잘 반영하면서 좋은 검색 결과를 이끌어내는 쿼리를 얻기 위함
 - 만약 검색을 다양하게 할 수 있다면?

▼ Transformer & Attention

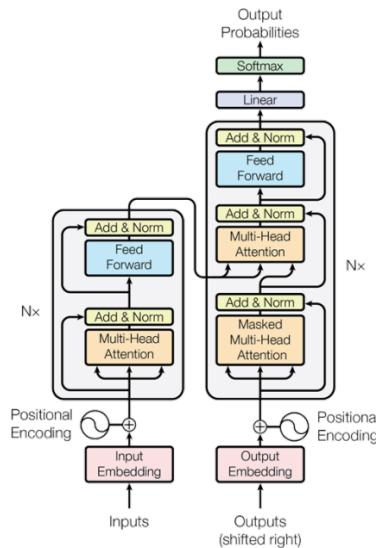
▼ Scaled Dot-product Attention



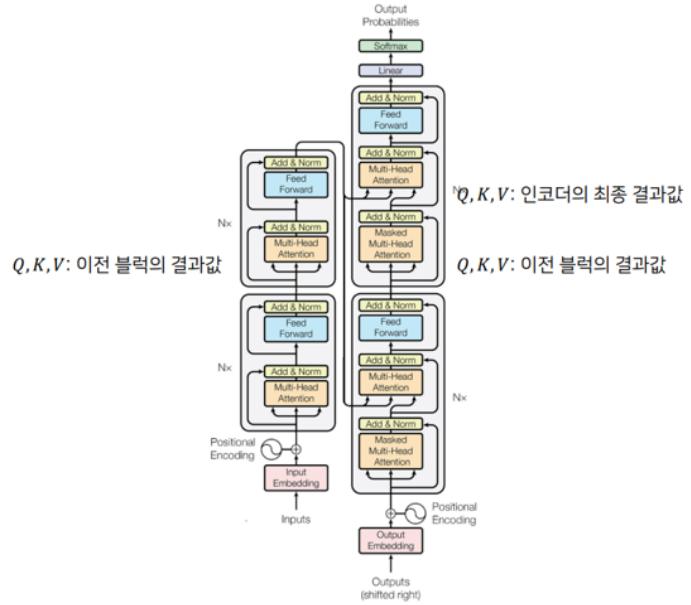
▼ Multi-Head Attention



▼ Transformer



▼ Encoder&Decoder가 여러 층인 것을 굳이 그려보자면



▼ Equations

[Multihead_Attention.pdf](#)

In case of Q , K and V come from same origin,
 $|Q| = |K| = |V| = (\text{batch_size}, n \text{ or } m, \text{hidden_size})$.

In case of Q and K , V come from different origin,
 $|Q| = (\text{batch_size}, n, \text{hidden_size})$
 $|K| = |V| = (\text{batch_size}, m, \text{hidden_size})$.

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{Q \cdot K^\top}{\sqrt{d_{\text{head}}}}\right) \cdot V$$

$$\text{MultiHead}(Q, K, V) = [\text{head}_1; \dots; \text{head}_h] \cdot W^O$$

where $\text{head}_i = \text{Attention}(Q \cdot W_i^Q, K \cdot W_i^K, V \cdot W_i^V)$,
and $W_i^Q \in \mathbb{R}^{d_{\text{model}} \times d_{\text{head}}}$, $W_i^K \in \mathbb{R}^{d_{\text{model}} \times d_{\text{head}}}$, $W_i^V \in \mathbb{R}^{d_{\text{model}} \times d_{\text{head}}}$, $W^O \in \mathbb{R}^{(h \times d_{\text{head}}) \times d_{\text{model}}}$.

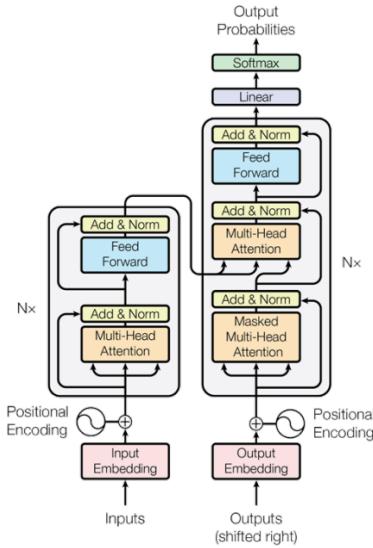
$$d_{\text{head}} = d_{\text{model}}/h = 64 \\ h = 8, d_{\text{model}} = 512$$

▼ Summary

- Previous method: attention in sequence to sequence
 - Query를 잘 만들어 key-value를 잘 matching시키자
- Multi-head Attention
 - 여러개의 Query를 만들어 다양한 정보를 잘 얻어오자
- Attention 자체로도 정보의 encoding과 decoding이 가능함을 보여줌

▼ Encoder block

▼ Transformer



▼ Equations

▼ Q,K and V are from previous alayer:

- Residual connections and Layer Normalizations are used.

[Deep Residual Learning for Image Recognition.pdf](#)

[Layer Normalization.pdf](#)

$$h_{0,1:m} = \text{emb}(x_{1:m}) + \text{pos}(1, m)$$

$$\tilde{h}_{i,1:m}^{\text{enc}} = \text{LayerNorm}(\text{Multihead}_i(Q, K, V) + h_{i-1,1:m}^{\text{enc}}),$$

where $Q = K = V = h_{i-1,1:m}^{\text{enc}}$.

$$\text{FFN}(h_{i,t}) = \text{ReLU}(h_{i,t} \cdot W_i^1) \cdot W_i^2$$

where $W_i^1 \in \mathbb{R}^{d_{\text{model}} \times d_{\text{ff}}}$ and $W_i^2 \in \mathbb{R}^{d_{\text{ff}} \times d_{\text{model}}}$.

$$h_{i,1:m}^{\text{enc}} = \text{LayerNorm}([\text{FFN}(\tilde{h}_{i,1}^{\text{enc}}); \dots; \text{FFN}(\tilde{h}_{i,m}^{\text{enc}})] + \tilde{h}_{i,1:m})$$

▼ Encoder is stack of encoder blocks:

$$h_{\ell_{\text{enc}},1:m}^{\text{enc}} = \text{Block}_{\text{enc}}(h_{\ell_{\text{enc}}-1,1:m}^{\text{enc}})$$

...

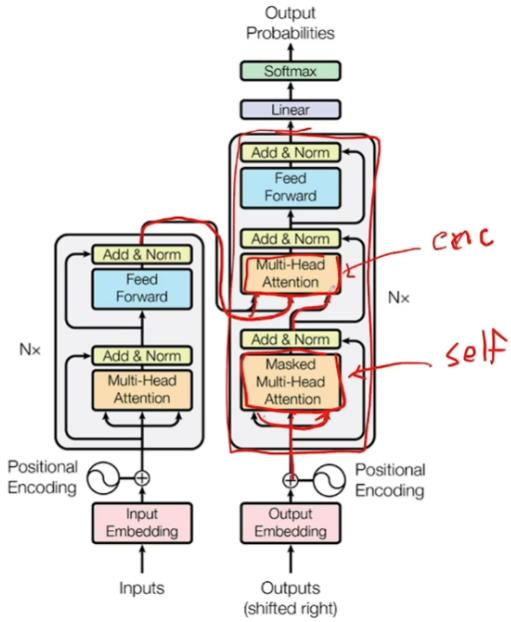
$$h_{1,1:m}^{\text{enc}} = \text{Block}_{\text{enc}}(h_{0,1:m}^{\text{enc}})$$

▼ Summary

- Encoder는 self-attention으로 구성되어 있음
 - Q,K,V는 이전 레이어의 출력값 - 즉, 같은 값
- Seq2Seq의 Attention과 달리, Q도 모든 time-step을 동시에 연산
 - 빠르지만 메모리를 많이 먹게 됨
- Residual connection으로 인해 깊은 네트워크 구성 가능
 - Big LM의 토대 마련

▼ Decoder with Masking

▼ Transformer



▼ Equations

▼ Given Dataset.

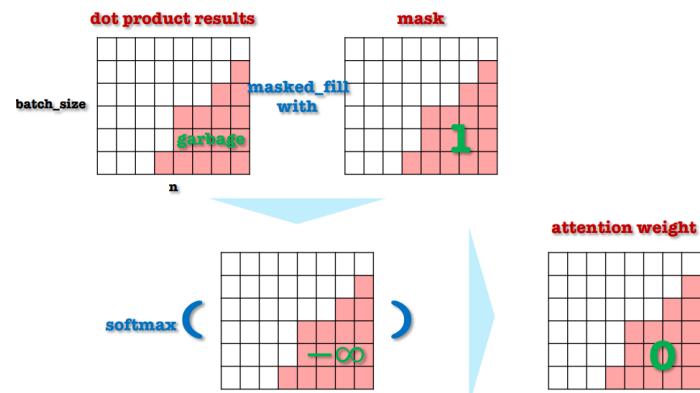
$$\begin{aligned} \mathcal{D} &= \{x^i, y^i\}_{i=1}^N \\ x^i &= \{x_1^i, \dots, x_m^i\} \text{ and } y^i = \{y_0^i, y_1^i, \dots, y_n^i\}, \\ \text{where } y_0 &= \langle \text{BOS} \rangle \text{ and } y_n = \langle \text{EOS} \rangle. \end{aligned}$$

▼ What we want is

$$\hat{y}_{1:n} = f(x_{1:m} : \theta)$$

▼ Before we start

- Using mask, assign $-\infty$ to make 0s for softmax results..



▼ Decoder Self-attention with mask

- 모든 attention에는 $\langle \text{pad} \rangle$ 에 마스킹이 들어간다.

- 단, 디코더에서는 AutoRegressive한 특성으로 인해, 다음스텝을 보는 것을 방지하는 마스킹을 함께 해줘야 한다.

$$h_{0,1:n} = \text{emb}(y_{0:n-1}) + \text{pos}(0, n-1)$$

$$\tilde{h}_{i,1:n}^{\text{dec}} = \text{LayerNorm}(\text{Multihead}_i(Q, K, V) + h_{i-1,1:n}^{\text{dec}}),$$

where $Q = K = V = h_{i-1,1:n}^{\text{dec}}$.

▼ Attention from encoder with mask for <pad>

$$\tilde{h}_{i,1:n}^{\text{dec}} = \text{LayerNorm}(\text{Multihead}_i(Q, K, V) + h_{i-1,1:n}^{\text{dec}}),$$

where $Q = \tilde{h}_{i,1:n}^{\text{dec}}$ and $K = V = h_{\ell,1:m}^{\text{dec}}$.

▼ FC layers

$$\text{FFN}(h_{i,t}) = \text{ReLU}(h_{i,t} \cdot W_i^1) \cdot W_i^2$$

where $W_i^1 \in \mathbb{R}^{d_{\text{model}} \times d_{\text{ff}}}$ and $W_i^2 \in \mathbb{R}^{d_{\text{ff}} \times d_{\text{model}}}$.

$$h_{i,1:m}^{\text{dec}} = \text{LayerNorm}([\text{FFN}(\tilde{h}_{i,1}^{\text{dec}}); \dots; \text{FFN}(\tilde{h}_{i,m}^{\text{dec}})] + \tilde{h}_{i,1:m}^{\text{dec}})$$

▼ Decoder is stack of decoder blocks.

$$h_{\ell_{\text{dec}},1:m}^{\text{dec}} = \text{Block}_{\text{dec}}(h_{\ell_{\text{dec}}-1,1:m}^{\text{dec}})$$

...

$$h_{1,1:m}^{\text{dec}} = \text{Block}_{\text{dec}}(h_{0,1:m}^{\text{dec}})$$

▼ Generator

$$\hat{y}_{1:n} = \text{softmax}(h_{\ell_{\text{dec}},1:m}^{\text{dec}} \cdot W_{\text{gen}}),$$

where $h_{\ell_{\text{dec}},1:m}^{\text{dec}} \in \mathbb{R}^{\text{batch_size} \times n \times \text{hidden_size}}$ and $W_{\text{gen}} \in \mathbb{R}^{\text{hidden_size} \times |V|}$.

▼ Summary

- Decoder는 2가지의 Attention으로 구성됨
 - Attention from encoder:
 - K와 V는 encoder의 최종 출력 값, Q는 이전 레이어의 출력 값
 - Self-Attention with mask:
 - Q,K,V는 이전 레이어의 출력 값
 - Attention weight 계산 시, softmax 연산 이전에 masking을 통해 음의 무한대를 주어, 미래 time-step을 보는 것을 방지
- 추론 때에는 self-attention의 mask는 필요 없으나, 모든 layer의 t 시점 이전의 모든 time-step(<t>)의 hidden_state가 필요

▼ Positional Encoding

▼ Unlike RNN,

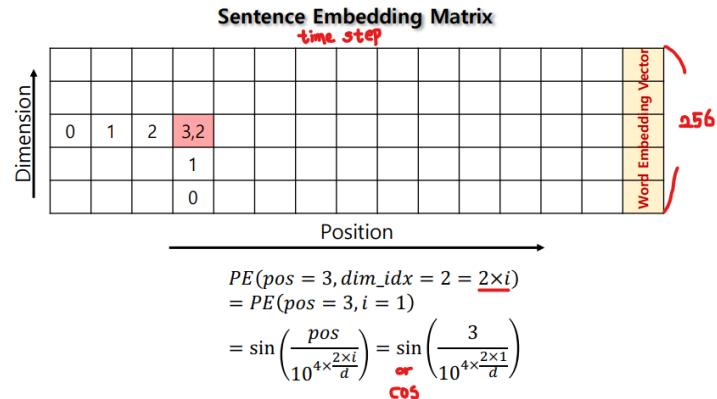
- Transformer는 위치 정보를 스스로 처리하지 않음(Conv2S도 마찬가지)
 - $h_t = f(x_t, h_{t-1}) \rightarrow$ RNN 계열은 위치 정보를 스스로 처리했음
 - 마치 FC layer의 입력 feature 순서를 바꿔 학습해도 성능이 똑같은 것과 같음

- 입력 순서를 바꿔 넣으면 출력도 순서가 바뀐 채 같은 값이 나올 것

- 따라서 위치순서 정보를 따로 인코딩해서 넣어줘야 함

▼ Positional Encoding

- 기존의 word embedding 값에 position encoding 값을 더해줌



▼ vs Positional Embedding → 학습도 가능

- 사실 위치 정보도 integer 값이므로 embedding layer를 통해 임베딩 할 수 있음
- BERT와 같은 모델은 positional encoding 대신에 positional embedding을 사용하기 도함

▼ Summary

- RNN과 달리, 순서(위치) 정보를 encoding해주는 작업이 필요
 - 학습이 아닌 단순 계산 후 encoding
- 학습에 의해 달라지는 값이 아니므로, 한번만 계산해 놓으면 됨

<https://www.blossominkyung.com/deeplearning/transformer-positional-encoding>

▼ Learning rate warm-up and linear decay

▼ Previous Method

SGD+Gradient

Clipping

- 가장 기본적

인 방법

$$\theta \leftarrow \theta - \gamma \nabla_{\theta} L(\theta)$$

- Learning rate에 따른 성능 변화

- 학습 후반부에 LR decay 해주기도

Adam

- Adaptive하게 LR을 조절

Algorithm 1: Generic adaptive optimization method setup. All operations are element-wise.

Input: $\{\alpha_t\}_{t=1}^T$; step size; $\{\phi_t, \psi_t\}_{t=1}^T$; function to calculate momentum and adaptive rate,

θ_0 ; initial parameter, $f(\theta)$; stochastic objective function.

Output: θ_T ; resulting parameters

while $t = 1$ to T **do**

$g_t \leftarrow \nabla f(\theta_{t-1})$ (Calculate gradients w.r.t. stochastic objective at timestep t)

$m_t \leftarrow \phi_t(g_1, \dots, g_t)$ (Calculate momentum)

$l_t \leftarrow \psi_t(g_1, \dots, g_t)$ (Calculate adaptive learning rate)

$\theta_t \leftarrow \theta_{t-1} - \alpha_t m_t l_t$ (Update parameters)

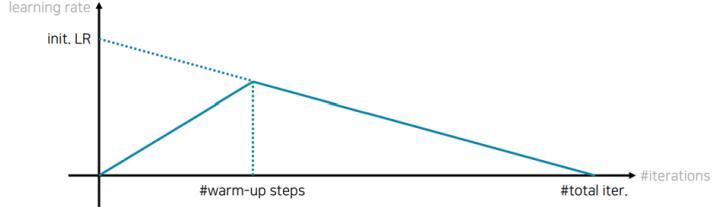
return θ_T

출처: [Liu et al., 2020]

▼ Warm-up and Linear Decay(Noam Decay)

- Heuristic Methods

- Control learning rate for Adam with hyper-params
- 학습 초기 불안정한 gradient를 통해 잘못된 momentum을 갖는 것을 방지
 - Residual Connection을 하는 과정에서 발생??
 - 대체로 5% 근처



- 결국 Trial&Error방식으로 Hyper-parameter 튜닝을 해야함
 - 가장 핵심은 #warm-up steps와 #total iterations.
 - 이외에도 다양한 hyper-parameters: init LR, batch size
- 심지어 튜닝에 따라 SGD+Gradient Clipping이 더 나은 결과를 얻기도 함

▼ Rectified Adam[Liu et al., 2020]

[On The Variance of the Adaptive Learning Rate and Beyond.pdf](#)

- Adam이 잘 동작하지 않는 이유(가설)
- Due to the lack of samples in the early stage, the adaptive learning rate has an undesirably large variance, which leads to suspicious/bad local optima. – [Liu et al., 2020]

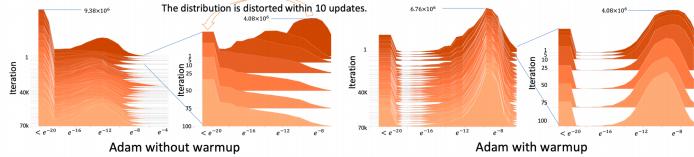


Figure 2: The absolute gradient histogram of the Transformers on the De-En IWSLT’ 14 dataset during the training (stacked along the y-axis). X-axis is absolute value in the log scale and the height is the frequency. Without warmup, the gradient distribution is distorted in the first 10 steps.

- Pytorch 구현

<https://github.com/LiyuanLucasLiu/RAdam>

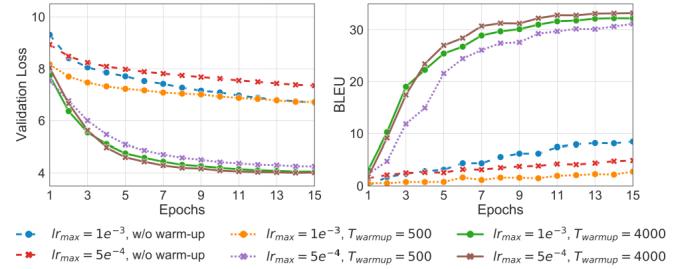
◦ `$ pip install torch-optimizer`

▼ Appendix: Beyond the paper

▼ Transformer의 단점

▼ 학습이 까다롭다.

- Bad local optima에 빠지기 매우 쉬움
- 그런데 paper에서 이것을 언급하지 않음
 - #warm-up step, learning rate



(a) Loss/BLEU on the IWSLT14 De-En task (Adam)

▼ 오죽하면,

[Training Tips for the Transformer Model.pdf](#)

[Transformers without Tears_ Improving the Normalization of Self-Attention.pdf](#)

[On the Variance of the Adaptive Learning Rate and Beyond.pdf](#)

▼ On Layer Normalization in Transformer Architecture[Xiong et al., 2020]

[On Layer Normalization in Transformer Architecture.pdf](#)

▼ Previous Work:

- Use Noam decay(warm-up and linear decay)
- Rectified Adam(RAdam)

▼ Propose:

- Layer Norm의 위치에 따라 학습이 수월해짐
 - LN이 gradient를 평坦하게 바꾸는 효과

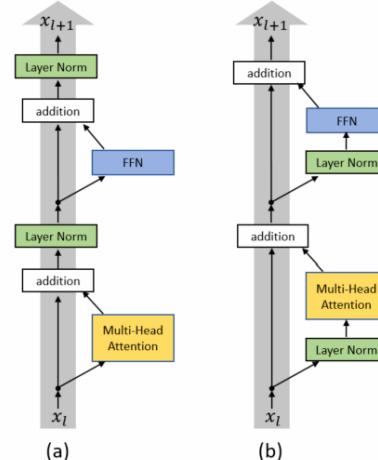
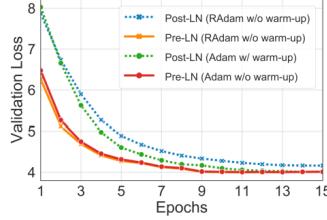


Figure 1: (a) Post-LN Transformer layer; (b) Pre-LN Transformer layer.

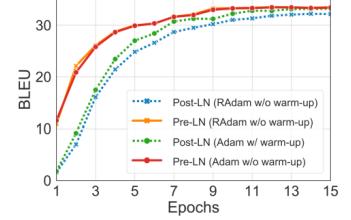
Table 1: Post-LN Transformer v.s. Pre-LN Transformer

Post-LN Transformer	Pre-LN Transformer
$x_{l,i}^{post,1} = \text{MultiHeadAtt}(x_{l,i}^{post}, [x_{l,1}^{post}, \dots, x_{l,n}^{post}])$	$x_{l,i}^{pre,1} = \text{LayerNorm}(x_{l,i}^{pre})$
$x_{l,i}^{post,2} = x_{l,i}^{post} + x_{l,i}^{post,1}$	$x_{l,i}^{pre,2} = \text{MultiHeadAtt}(x_{l,i}^{pre,1}, [x_{l,1}^{pre,1}, \dots, x_{l,n}^{pre,1}])$
$x_{l,i}^{post,3} = \text{LayerNorm}(x_{l,i}^{post,2})$	$x_{l,i}^{pre,3} = x_{l,i}^{pre} + x_{l,i}^{pre,2}$
$x_{l,i}^{post,4} = \text{ReLU}(x_{l,i}^{post,3}W^{1,l} + b^{1,l})W^{2,l} + b^{2,l}$	$x_{l,i}^{pre,4} = \text{LayerNorm}(x_{l,i}^{pre,3})$
$x_{l,i}^{post,5} = x_{l,i}^{post,3} + x_{l,i}^{post,4}$	$x_{l,i}^{pre,5} = \text{ReLU}(x_{l,i}^{pre,4}W^{1,l} + b^{1,l})W^{2,l} + b^{2,l}$
$x_{l,i}^{post} = \text{LayerNorm}(x_{l,i}^{post,5})$	$x_{l,i}^{pre} = x_{l,i}^{pre,5} + x_{l,i}^{pre,3}$
$\text{Final LayerNorm: } x_{Final,i}^{pre} \leftarrow \text{LayerNorm}(x_{l+1,i}^{pre})$	

▼ Evaluation Results



(a) Validation Loss (IWSLT)



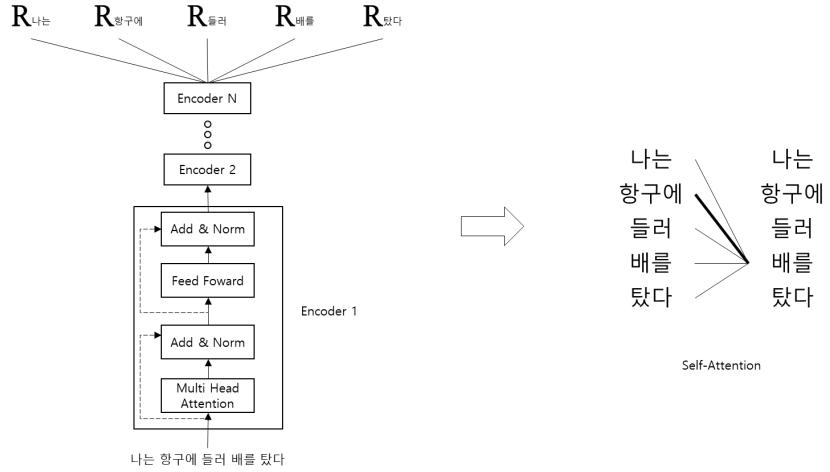
(b) BLEU (IWSLT)

▼ Summary

- Pre-Norm 방식을 통해 warm-up 및 LR 튜닝 제거 가능
 - LR decay는 여전히 필요
- 그 밖에도 Layer Norm을 대체하거나, weight initialization을 활용하여 좀 더 나은 성능을 확보할 수 있음

▼ 2-2. 동작 방식

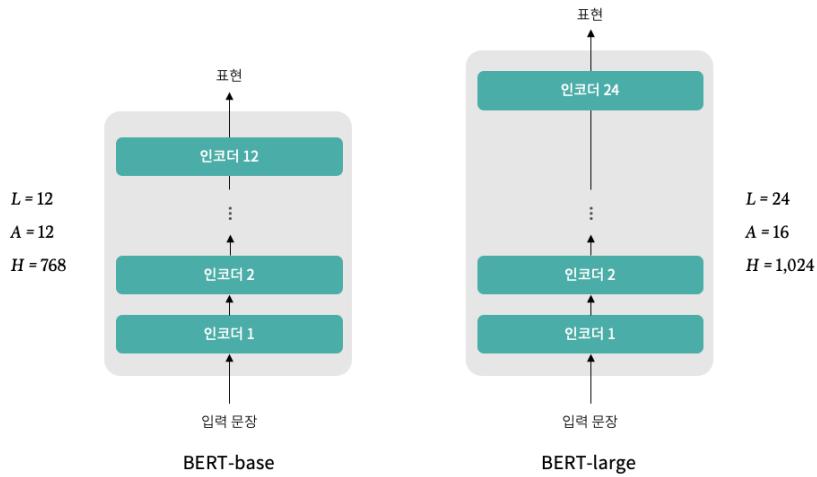
- 이전 예시 나는 항구에 들러 배를 탔다라는 A 문장을 트랜스 포머의 인코더에 입력으로 제공하고 문장의 각 단어에 대한 임베딩을 출력으로 가져온다.
- 인코더에 문장을 입력하면 인코더는 멀티헤드 어텐션 메커니즘을 사용해 문장의 각 단어의 문맥을 이해해 문장에 있는 각 단어의 문맥 표현을 출력으로 반환한다.



▼ 2-3. 구조

BERT는 크기에 따라 아래의 두 모델로 나뉜다.

- Bert-base : OpenAI GPT와 동일한 하이퍼파라미터를 가짐. GPT와의 성능 비교를 위해 설계됨
- Bert-large : BERT의 최대 성능을 보여주기 위해 만들어짐



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
BERT _{BASE}	84.6/83.4	71.2	90.1	93.5	52.1	85.8	88.9	66.4	79.6
BERT _{LARGE}	86.7/85.9	72.1	91.1	94.9	60.5	86.5	89.3	70.1	81.9

- 모든 task에 대해 SOTA 달성
- BERT-large가 일반적으로 base 모델보다 성능이 뛰어남
- 사전학습 덕분에 데이터셋의 크기가 작아도 모델의 크기가 클수록 정확도가 상승
 - 사전학습을 한다는 것은 General하게 사람들의 언어체계를 학습한다고 이해하면 쉽습니다.
 - 즉, 문맥과 표현을 사전학습을 통해 배워 놓고, Down stream task를 목적으로 따라 파인 투닝하는 것입니다.
- 그 밖의 여러 BERT 기본 구조

BERT-tiny	$L=2, A=2, H=128$
BERT-mini	$L=4, A=4, H=256$
BERT-small	$L=4, A=8, H=512$
BERT-medium	$L=8, A=8, H=512$

3. BERT의 사전 학습

▼ 3-1. BERT의 입력 표현

이제 우리는 BERT에 데이터를 입력하기 전에 임베딩 레이어를 기반으로 입력 데이터를 임베딩으로 변환해야 합니다.

그리고 아래와 같이 3가지 방법이 있습니다.

▼ Token Embedding (2)

- Sentence Pair는 합쳐져서 단일 Sequence로 입력되고, Pair는 한 개 혹은 2개의 문장으로 이루어져 있다.
 - e.g. Translation
 - 1: My dog is cute, he likes playing.
 - 2: 나의 강아지는 귀엽고, 노는 것을 좋아한다.
- 문장의 시작 부분에는 **[CLS]**라는 토큰을 추가합니다.

- [CLS] 토큰의 경우, 분류 테스크에서만 사용되지만 다른 테스크에서도 반드시 추가 해줘야 합니다.
- 문장의 끝에는 [SEP]라는 토큰을 추가합니다.
- 예시

Input	[CLS]	my	dog	is	cute	[SEP]	he	likes	play	# #ing	[SEP]
Token Embeddings	$E_{[CLS]}$	E_{my}	E_{dog}	E_{is}	E_{cute}	$E_{[\text{SEP}]}$	E_{he}	E_{likes}	E_{play}	$E_{\#\#\text{ing}}$	$E_{[\text{SEP}]}$

- 기존 토큰: `token = [my, dog, is, cute, he, likes, playing]`
- 토큰 임베딩을 거칠 경우: `token_embedding=[[CLS], my, dog, is, cute, [SEP], he, likes, playing , [SEP]]`

▼ Segment Embedding (3)

- Segment Embedding은 주어진 두 문장을 구별하는데 사용됩니다.
 - 토큰 임베딩이 진행되었다고 가정합니다.
 - `token_embedding=[[CLS], my, dog, is, cute, [SEP], he, likes, playing , [SEP]]`
 - 예시 문장1: My dog is cute
 - 예시 문장2: He likes playing
- Segment Embedding Layer는 입력에 대한 출력으로 E_A 와 E_B 만 반환합니다.

Input	[CLS]	my	dog	is	cute	[SEP]	he	likes	play	# #ing	[SEP]
Token Embeddings	$E_{[CLS]}$	E_{my}	E_{dog}	E_{is}	E_{cute}	$E_{[\text{SEP}]}$	E_{he}	E_{likes}	E_{play}	$E_{\#\#\text{ing}}$	$E_{[\text{SEP}]}$
Segment Embeddings	E_A	E_A	E_A	E_A	E_A	E_A	E_B	E_B	E_B	E_B	E_B

- 1번 문장에 속할 경우 E_A 를 반환
- 2번 문장에 속할 경우 E_B 를 반환

▼ Position Embedding (4)

- 주의 하실 점은 Positional Encoding과는 조금 다른 Positional Embedding은 다른 개념입니다.
- Transformer의 메커니즘을 이해한다면, 어떤 반복 메커니즘을 사용하지 않고 모든 단어를 병렬 처리함을 알고 있을 것입니다.
- 이에 따라, 단어의 순서가 중요하므로 위치에 대한 정보를 제공한다.

Input	[CLS]	my	dog	is	cute	[SEP]	he	likes	play	# #ing	[SEP]
Token Embeddings	$E_{[CLS]}$	E_{my}	E_{dog}	E_{is}	E_{cute}	$E_{[\text{SEP}]}$	E_{he}	E_{likes}	E_{play}	$E_{\#\#\text{ing}}$	$E_{[\text{SEP}]}$
Segment Embeddings	E_A	E_A	E_A	E_A	E_A	E_A	E_B	E_B	E_B	E_B	E_B
Position Embeddings	E_0	E_1	E_2	E_3	E_4	E_5	E_6	E_7	E_8	E_9	E_{10}

- 위 그림이 최종적으로 BERT 모델에 들어가는 INPUT 값이 됩니다.

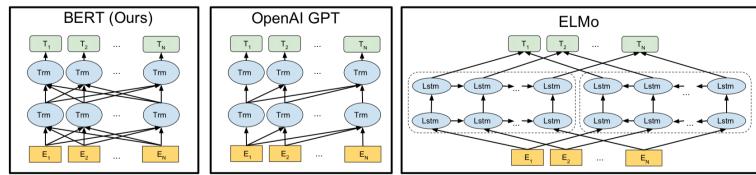
▼ WordPeice Tokenizer (1)

- 그런데 Token Embedding층에 들어갈 Input으로 사용될 토큰이 어떻게 쪼개지는지 알아야겠죠?
- BERT는 하위 단어 토큰화 알고리즘()을 기반으로 작동합니다.

- 한글을 기준으로 형태소를 분석하듯이, 영어의 경우 `pretraining`이라는 단어를 토큰화 해보겠습니다.
 - $\text{pre} + \text{train} + \text{ing}$
 - `tokens = [pre, ##train, ##ing]`
- 그렇다면 왜 하위 단어로 쪼개는 것일까요?
 - 하위 단어로 쪼갤 경우, 어휘 사전 이외(OOV, Out-Of-Vocabulary)의 단어를 처리하는데 효과적입니다.
 - 기본적으로 BERT의 어휘 사전은 3만 토큰이므로, 웬만한 어휘들은 토큰화가 가능합니다.
 - 만약 존재하지 않는 토큰이라면 어떻게 되는지는 에서 추가적으로 다뤄보겠습니다.

▼ 3-2. 사전 학습 전략

▼ 기존의 사전 학습 방법론



- 전통적인 언어 모델링(Language Modeling): **n-gram**, 앞의 N-1개의 단어로 뒤에 올 단어를 예측하는 모델
- 필연적으로 단방향일수 밖에 없고, BiLM을 사용하는 ELMo더라도 순방향, 순방향의 언어 모델을 둘 다 학습해 활용하지만,
- 단방향 언어 모델의 출력을 concat하여 사용하는 정도이므로 제한적인 양방향성을 가짐

NEW

▼ 마스크 언어 모델링(MLM, Masked Language Modeling)

- MLM은 일반적으로 임의의 문장이 주어지고 단어를 순서대로 보면서 다음 단어를 예측하도록 모델을 학습 시키는 것입니다.
- MLM의 종류 2가지

▼ 자귀 회귀 언어 모델링 — (단방향)

- 앞 → 끝 방향으로 예측(전방 예측)
- 끝 → 앞 방향으로 예측(후방 예측)
- 예시

Paris is a beautiful city. I love Paris.라는 두 문장이 있습니다.

- 처음에는 city라는 단어에 공백을 추가한다.

a. Paris is a beautiful __. I love Paris.

- 이제 모델은 공백을 예측한다.

a. 전방 예측은 Paris부터 문장을 읽는다.

i. (→) Paris is a beautiful __.

b. 후방 예측은 끝의 Paris부터 문장을 읽는다.

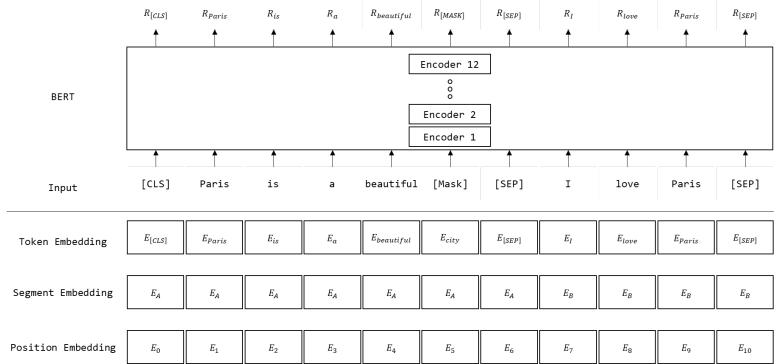
i. (←) __. I love Paris.

- 자동 회귀 언어 모델은 원래 단방향이므로 한 방향으로만 문장을 읽습니다.

▼ 자동 인코딩 언어 모델링 — (양방향)

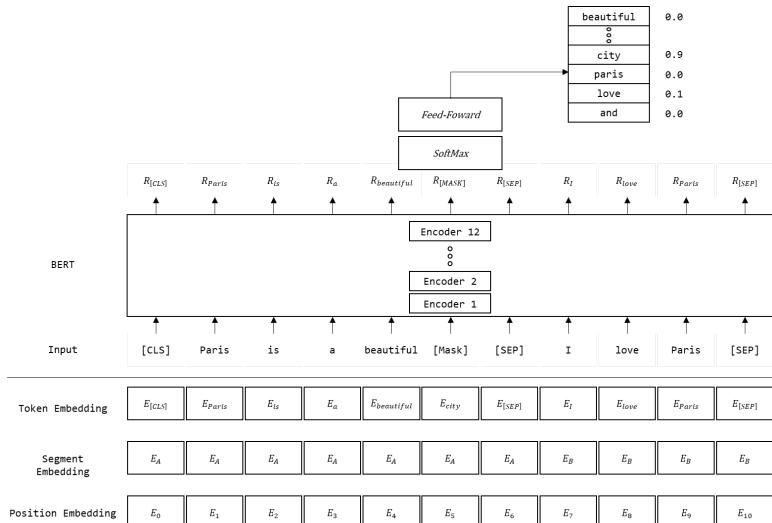
- 이 방식의 경우, 전방 및 후방 예측을 모두 활용합니다.

- 즉, 양방향으로 문장을 읽습니다.
 - (\rightarrow) Paris is a beautiful __. I love Paris. (\leftarrow)
 - 당연히 단방향 보다 문장 이해 측면에서 나으므로, 더 정확한 결과를 제공합니다.
 - BERT는 자동 인코딩 언어 모델로, 예측을 위해 양방향으로 문장을 읽습니다.
- ▼ 주어진 문장에서 전체 단어의 15%를 무작위로 마스킹하고, 마스크된 단어를 예측하도록 모델을 학습하는 것입니다.
- Paris is a beautiful city. I love Paris.
 - 위의 예시에서 들었던 문장을 토큰화 하고 마스킹하도록 하겠습니다.
- ```
tokens = [[CLS], Paris, is, a, beatiful, [MASK], [SEP], I, love, Paris,
 [SEP]]
```
- 하지만 위와 같이 토큰화를 하게 될 경우, 사전학습과 파인튜닝 사이에서 불일치가 발생합니다. [MASK] 토큰이 없기 때문입니다.
- ▼ 이 문제를 극복하기 위해 **80-10-10%** 규칙을 사용합니다.
- 기준 15% 중 **80%**의 토큰을 [MASK] 토큰으로 교체한다.
- ```
tokens = [ [CLS], Paris, is, a, beatiful, [MASK], [SEP], I, love, Paris,
          [SEP] ]
```
- 15% 중 **10%**의 토큰을 임의의 토큰으로 교체한다.
- ```
tokens = [[CLS], Paris, is, a, beatiful, love, [SEP], I, love, Paris,
 [SEP]]
```
- 15% 중 나머지 10%의 토큰은 어떤 변경도 하지 않는다.
- ```
tokens = [ [CLS], Paris, is, a, beatiful, city, [SEP], I, love, Paris, [SEP]
          ]
```
- ▼ 이후, 앞서 언급된 (Token, Segment, Position) Embedding 층을 거쳐 입력 임베딩(토큰의 표현 벡터)를 반환합니다.
- $R_{[CLS]}$ 는 [CLS] 토큰의 표현 벡터를 의미하고, $R_{[Paris]}$ 는 Paris의 표현 벡터를 의미합니다.



▼ 이제 우리는 토큰의 표현 벡터를 얻었으므로, 마스크된 토큰을 예측해야 한다.

- BERT에서 반환된 마스크된 토큰 $R_{[MASK]}$ 의 표현을 (softmax 활성화 + feed-forward) 네트워크에 입력한다.
- 이후, 우리는 해당 마스크 위치의 단어가 될 확률을 얻을 수 있게 된다.



▼ 전체 단어 마스킹(WWM, Whole Word Masking)

- MLM에서 조금 더 나아가 심화 내용인 전체 단어 마스킹에 대해서 알아보자.
 - Let us start pretraining the model이라는 문장을 워드피스 토크나이저를 사용해 문장을 토큰화하면 다음과 같은 토큰을 얻을 수 있다.
 - `tokens = [let, us, start, pre, ##train, ##ing, the, model]`
 - `tokens = [[CLS], let, us, start, pre, ##train, ##ing, the, model, [SEP]]`
 - `tokens = [[CLS], [MASK], us, start, pre, [MASK], ##ing, the, model, [SEP]]`
- WWM 방법에서는 하위 단어가 마스킹 되면 관련된 모든 단어가 마스킹된다.
 - `tokens = [[CLS], [MASK], us, start, [MASK], [MASK], the, model, [SEP]]`
 - 마스크 비율(15%)를 초과하면 다른 단어의 마스크를 무시한다. 이 때는 let을 무시한다.
 - `tokens = [[CLS], let, us, start, [MASK], [MASK], the, model, [SEP]]`
 - 이후는 동일하게 마스크 된 토큰을 학습하도록 한다.

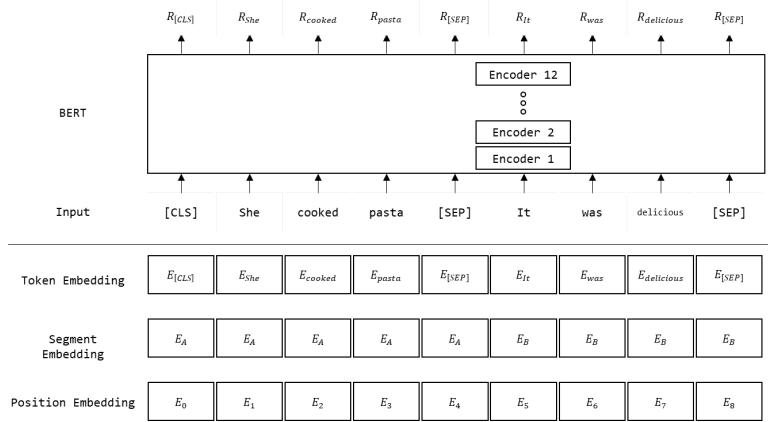
▼ 다음 문장 예측(NSP, Next Sentence Prediction)

- NSP는 BERT 학습에서 사용되는 다른 테스크로, 이진 분류 테스트다.

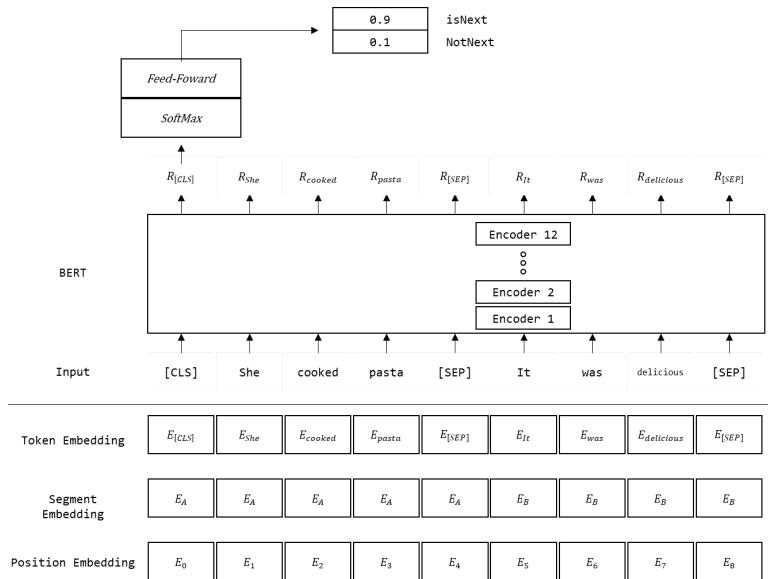
- isNext
- NotNext

문장 쌍	레이블
She cooked pasta It was delicious	isNext
Birds fly in the sky. He was reading	NotNext

- NSP는 두 문장 사이의 관계를 파악하며, 질문-응답 및 유사문장탐지와 같은 다운 스트림 테스크에서 유용하다.
- 예시를 통해 살펴보자.



- BERT는 [CLS] 토큰만 가져와 분류 작업한다.
 - [CLS] 토큰은 기본적으로 모든 토큰의 집계 표현을 보유하고 있으므로 문장 전체에 대한 표현을 담고 있다.
 - 학습 초기에는 물론 피드포워드 네트워크 및 인코더 계층의 가중치가 최적이 아니라 올바른 확률을 반환하지 못하지만,
 - 역전파를 기반으로 반복 학습을 통해 최적의 가중치를 찾게 되면 아래의 그림과 같은 반환 값을 내놓게 된다.



▼ 3-3. 사전 학습 절차

1. 말뭉치에서 두 문장 A, B를 샘플링한다.
 - A와 B의 총 토큰 수의 합은 512보다 작거나 같아야 한다.
 - 전체의 50%는 B 문장이 A 문장과 이어지는 문장(**isNext**)이 되도록 샘플링하고, 나머지 50%는 B 문장이 A 문장의 후속 문장이 아닌 것(**NotNext**)으로 샘플링한다.
2. 워드피스 토크나이저로 문장을 토큰화하고, 토큰 임베딩-세그먼트 임베딩-위치 임베딩 레이어를 거친다.
 - 시작 부분에 **[CLS]** 토큰을, 문장 끝에 **[SEP]** 토큰을 추가한다.
 - **80-10-10%** 규칙에 따라 토큰의 15%를 무작위 마스킹한다.
3. BERT에 토큰을 입력하고, MLM과 NSP 태스크를 동시에 수행한다
 - WarmUp Step(= 1만): 초기 1만 스텝은 학습률이 0에서 $1e - 4$ 로 선형 증가, 1만 스텝 이후 선형 감소

- DropOut(0.1)
- **GELU Activation Func** : 음수에 대해서도 미분이 가능해 약간의 그래디언트를 전달할 수 있음
 - GELU는 가우시안 오차 선형 유닛(Gaussian Error Linear Unit)을 사용한다고 합니다.
 - $GELU(x) = x\Phi(x)$
 - Φ 은 표준 가우시안 누적 분포이며, GELU함수는 다음 수식의 근사치라고 합니다.
 - $GELU(x) = 0.5x \left(1 + \tanh \left[\sqrt{\frac{2}{\pi}} (x + 0.044715x^3) \right] \right)$

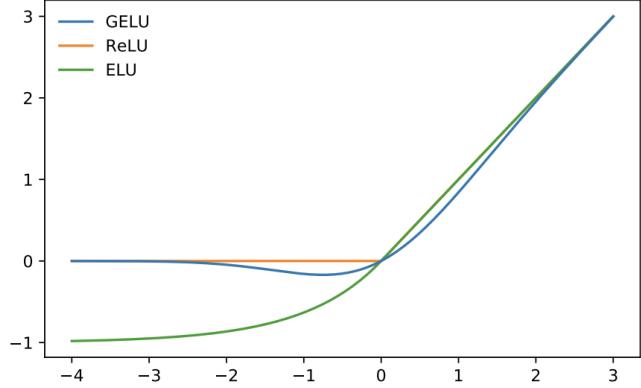


Figure 1: The GELU ($\mu = 0, \sigma = 1$), ReLU, and ELU ($\alpha = 1$).

4. 하위 단어 토큰화 알고리즘

▼ 4-1. 하위 단어 토큰화를 하는 이유

- BERT 및 GPT-3를 포함한 많은 최신 LLM 모델에서는 하위 단어 토큰화를 사용합니다.
- 그 이유는 바로 OOV단어 처리에 매우 효과적입니다.
 - OoV(Out of Vocabulary): 단어 집합에 존재하지 않는 단어들이 생기는 상황 (Inference 시 TrainSet에 없던 단어가 있을 경우)
 - OoV가 발생한다면 어떻게 할까?
 - OoV 단어 발생 시 <UNK> 토큰으로 처리하고, <UNK> 토큰은 모델 학습에 있어 매우 치명적으로 작용합니다.
 - 하지만 일반적으로 Vocab 자체가 매우 크기 때문에 신조어가 아닌 이상 대부분의 단어는 하위 단어로 토큰화 할 수 있습니다.

▼ 4-2. 바이트 쌍 인코딩(BPE)

- BPE은 빈도수가 가장 높은 쌍의 문자나 문자열을 하나로 합치는 알고리즘입니다.
- 이 방법은 초기에 데이터 압축에 사용되었으나, 나중에 자연어 처리에서 텍스트 토큰화에도 적용되었습니다.
 1. Dataset에서 모든 단어를 빈도수와 함께 추출
 2. 모든 단어를 문자로 나누고 문자 시퀀스로 만든다.
 3. 어휘 사전 크기를 정의한다.
 4. 문자 시퀀스에 있는 모든 고유문자를 어휘 사전에 추가한다.
 5. 가장 빈도수가 큰 기호 쌍을 식별하고, 해당 쌍을 병합해서 어휘 사전에 추가한다.
 6. 어휘 사전 크기에 도달할 때 까지 5번 과정을 반복한다.

• 참고 링크

Byte pair encoding 설명 (BPE tokenizer, BPE 설명, BPE 예시)

Byte pair encoding (BPE)는 문장 혹은 단어 안에 있는 글자들을 적절한 단위로 나누는 subword tokenizer의 하나로, token들의 빈도를 기반으로 높은 빈도의 토큰들을 merge해가며 최종 token들을 만들어내는 방법이다.

<https://process-mining.tistory.com/189>

▼ 4-3. 바이트 수준 바이트 쌍 인코딩(BBPE)

- BPE와 동작 방식이 거의 유사하지만, 단어를 문자 시퀀스로 변환하지 않고 바이트 수준 시퀀스로 변환합니다.
 - 이를 통해, 다국어에 대해 어휘 사전을 공유할 수 있게 됩니다.
- Dataset에서 모든 단어를 빈도수와 함께 추출
 - 모든 단어를 문자로 나누고 문자 시퀀스로 만든다 → 모든 단어를 문자로 나누고 바이트 수준 시퀀스 만든다.
 - 어휘 사전 크기를 정의한다.
 - 문자 시퀀스에 있는 모든 고유문자를 어휘 사전에 추가한다.
 - 가장 빈도수가 큰 기호 쌍을 식별하고, 해당 쌍을 병합해서 어휘 사전에 추가한다.
 - 어휘 사전 크기에 도달할 때 까지 5번 과정을 반복한다.

▼ 4-4. 워드 피스

- 워드 피스는 BPE와 유사하게 동작하지만, 한 가지 차이점이 있다.
- BPE에서는 데이터셋에서 단어의 빈도를 추출하고 단어를 문자 시퀀스로 나눈다. 이후, 어휘 사전 크기에 도달할 때 까지 고빈도 기호 쌍을 병합한다.
- 하지만 워드피스는 심볼 쌍을 병합하지 않고, 가능도를 기준으로 병합한다.
- 심볼쌍의 가능도를 계산
 - $\text{argmax}(\frac{p(st)}{p(s)p(t)})$

5. Code Review

▼ Transformer(직접 구현)

https://github.com/dorae222/DeepLearning/blob/main/4.%20Lv4_NLG/simple_nmt/models/transformer.py

▼ BERT

- 코드를 자세하게 분석하고 작성하기에 시간이 부족하여, 모델 아키텍처와 코드를 매칭 시켜 이해한대로 편집하였습니다.
- 코드 링크

▼ Full Architecture

- 토큰화
- PositionalEmbedding
- Transformer Encoder
- BERT(MLM + NSP)

▼ 토큰화

Token	Embedding	$E_{[CLS]}$	E_{She}	E_{cooked}	E_{pasta}	$E_{[SEP]}$	E_{lt}	E_{was}	$E_{delicious}$	$E_{[SEP]}$
-------	-----------	-------------	-----------	--------------	-------------	-------------	----------	-----------	-----------------	-------------

```
tokenizer = BertWordpieceTokenizer(
    do_lower_case=True,
    handle_chinese_chars=False,
    strip_accents=False,
    lowercase=True
)

tokenizer.train(
    filepaths,
    vocab_size=30_000,
    min_frequency=1,
    limit_alphabet=1000,
    wordpieces_prefix="#",
    special_tokens=[ "[PAD]", "[CLS]", "[SEP]", "[MASK]", "[UNK]" ]
)

tokenizer = BertTokenizer.from_pretrained("./bert-it-vocab.txt", local_files_only=True)
```

▼ PositionalEmbedding

Input	[CLS]	She	cooked	parts	[SEP]	IT	was	delicious	[SEP]
Token Embedding									
	$E_{[CLS]}$	E_{She}	E_{cooked}	E_{parts}	$E_{[\text{SEP}]}$	E_{IT}	E_{was}	$E_{\text{delicious}}$	$E_{[\text{SEP}]}$
Segment Embedding									
	E_s	E_s	E_s	E_s	E_s	E_d	E_d	E_d	E_d
Position Embedding									
	E_1	E_2	E_3	E_4	E_5	E_6	E_7	E_8	E_9

```

class PostEmbedding(Embedding, metaclass=abc.ABCMeta):
    def __init__(self, vocab_size, max_len=10000, embed_dim=100, seed=1337, dropout=0.0):
        super().__init__(vocab_size, max_len=max_len)
        # Converts the position embeddings back to log space
        self._log_pos_embeddings = np.log(np.arange(max_len) + 1).reshape((1, max_len, 1))
        self._pos_embedding = None
        self._pos_embedding_fn = None
        self._pos_embedding_is_set = False

        for pos in range(max_len):
            for i in range(embed_dim):
                for j in range(10000):
                    if pos * 10000 + i == j:
                        self._pos_embeddings[i][j] = 1 / (10000 ** 0.5)

        self._pos_embeddings = self._pos_embeddings / np.linalg.norm(self._pos_embeddings, axis=1, keepdims=True)
        self._pos_embeddings = self._pos_embeddings.T

    @property
    def pos_embedding(self):
        if self._pos_embedding is None:
            self._pos_embedding = self._pos_embedding_fn()
        return self._pos_embedding

    @property
    def pos_embedding_fn(self):
        if self._pos_embedding_fn is None:
            self._pos_embedding_fn = self._pos_embedding_fn_fn()
        return self._pos_embedding_fn

    def _pos_embedding_fn_fn(self):
        raise NotImplementedError("PostEmbedding does not implement _pos_embedding_fn_fn")

    def _pos_embedding_fn(self):
        if self._pos_embedding_is_set:
            return self._pos_embedding
        else:
            self._pos_embedding_is_set = True
            return self._pos_embedding_fn_fn()

    def __repr__(self):
        return f'{self.__class__.__name__}(vocab_size={self.vocab_size}, max_len={self.max_len}, embed_dim={self.embed_dim}, seed={self.seed}, dropout={self.dropout})'

    def __init__(self, vocab_size, max_len, embed_dim, seed, emb_initializer='xavier', dropout=0.0):
        super().__init__(vocab_size, max_len=max_len)
        self._embed_dim = embed_dim
        self._seed = seed
        self._emb_initializer = emb_initializer
        self._dropout = dropout
        self._pos_embeddings = None
        self._pos_embedding_fn = None
        self._pos_embedding_is_set = False
        self._pos_embedding_fn_fn = None

        if self._emb_initializer == 'xavier':
            self._pos_embeddings = np.random.uniform(-1, 1, (embed_dim, max_len))
        else:
            self._pos_embeddings = np.zeros((embed_dim, max_len))

        self._pos_embeddings = self._pos_embeddings / np.linalg.norm(self._pos_embeddings, axis=1, keepdims=True)
        self._pos_embeddings = self._pos_embeddings.T

    def __init__(self, vocab_size, max_len, embed_dim, seed, emb_initializer='xavier', dropout=0.0):
        super().__init__(vocab_size, max_len=max_len)
        self._embed_dim = embed_dim
        self._seed = seed
        self._emb_initializer = emb_initializer
        self._dropout = dropout
        self._pos_embeddings = None
        self._pos_embedding_fn = None
        self._pos_embedding_is_set = False
        self._pos_embedding_fn_fn = None

        if self._emb_initializer == 'xavier':
            self._pos_embeddings = np.random.uniform(-1, 1, (embed_dim, max_len))
        else:
            self._pos_embeddings = np.zeros((embed_dim, max_len))

        self._pos_embeddings = self._pos_embeddings / np.linalg.norm(self._pos_embeddings, axis=1, keepdims=True)
        self._pos_embeddings = self._pos_embeddings.T

    def __init__(self, vocab_size, max_len, embed_dim, seed, emb_initializer='xavier', dropout=0.0):
        super().__init__(vocab_size, max_len=max_len)
        self._embed_dim = embed_dim
        self._seed = seed
        self._emb_initializer = emb_initializer
        self._dropout = dropout
        self._pos_embeddings = None
        self._pos_embedding_fn = None
        self._pos_embedding_is_set = False
        self._pos_embedding_fn_fn = None

        if self._emb_initializer == 'xavier':
            self._pos_embeddings = np.random.uniform(-1, 1, (embed_dim, max_len))
        else:
            self._pos_embeddings = np.zeros((embed_dim, max_len))

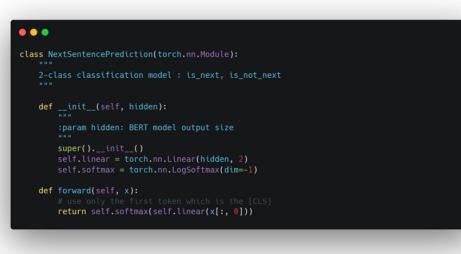
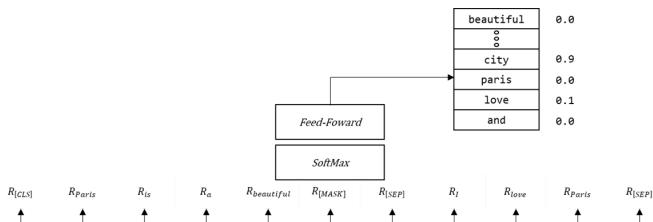
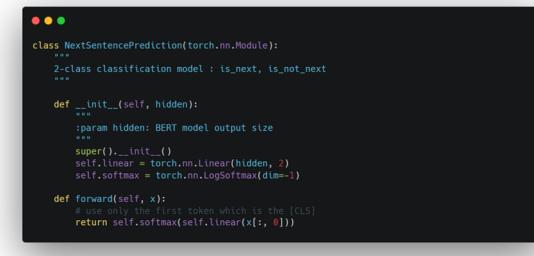
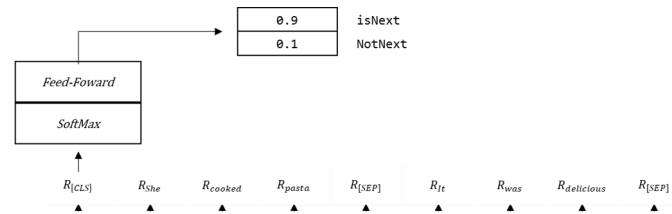
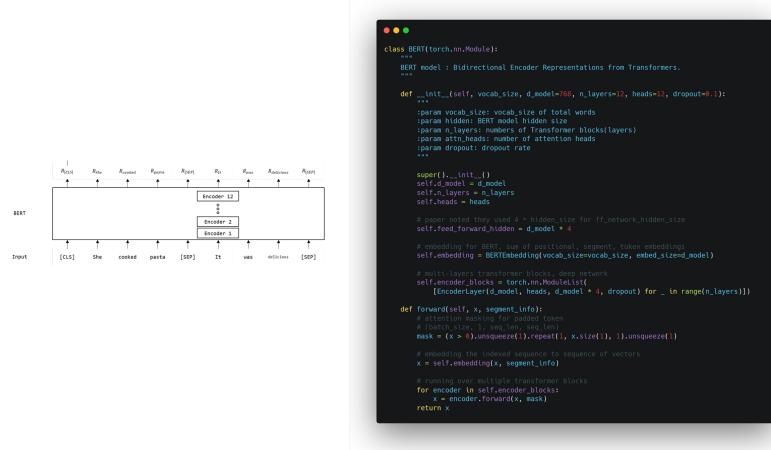
        self._pos_embeddings = self._pos_embeddings / np.linalg.norm(self._pos_embeddings, axis=1, keepdims=True)
        self._pos_embeddings = self._pos_embeddings.T

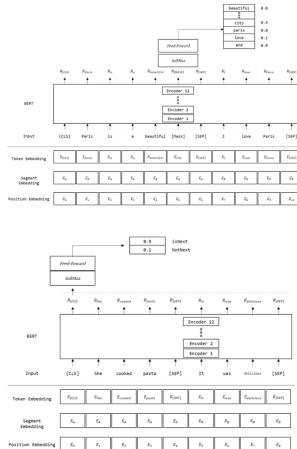
```

▼ Transformer Encoder

Encoder 12

▼ BERT(MLM + NSP)





- FineTuning 예시 코드(교내 동아리에서 진행된 방언 분류 코드입니다.)

https://github.com/dorae222/HAI_Kaggle_Competition/blob/main/v_1_hai_kaggle_summer.ipynb

6. 논문 리뷰



앞의 1~5까지 내용을 보시고 아래 내용을 보시면 아래 논문을 이해하기 쉬울 것 같습니다.

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

We introduce a new language representation model called BERT, which stands for Bidirectional Encoder Representations from Transformers. Unlike recent language representation models, BERT is...

 <https://arxiv.org/abs/1810.04805>

The arXiv logo consists of the word "arXiv" in a stylized font where the "X" is replaced by a red and grey crossed-out symbol.

▼ 1. Introduction (서론)

- BERT(Bidirectional Encoder Representations from Transformers)는 양방향 Transformer 아키텍처를 기반으로 한 모델입니다.
 - 사전 학습(pre-training)과 미세 조정(fine-tuning)의 두 단계로 학습되며, 다양한 NLP 작업에 적용할 수 있습니다.

▼ 2. Related Work (관련 연구)

2.1 Unsupervised Feature-based Approaches

- Word2Vec, GloVe와 같은 비지도 학습(unsupervised learning) 방법을 통해 단어 임베딩을 생성하는 방식이 언급됩니다.
 - 이러한 방법은 각 단어를 독립적으로 임베딩하는데, 문맥 정보가 제대로 반영되지 않는다는 한계가 있습니다.

- 단어 혹은 문장의 representation 학습

- non-neural method

Class-Based n-gram Models of Natural Language
Peter F. Brown, Vincent J. Della Pietra, Peter V. deSouza,
Jenifer C. Lai, Robert L. Mercer. Computational Linguistics,
Volume 18, Number 4, December 1992. 1992.
 <https://aclanthology.org/J92-4003/>

Peter F. Brown¹
Peter V. deSouza²
Robert L. Mercer³
IBM T.J. Watson Research Center

Vincent J. Della Pietra¹
Jenifer C. Lai¹

¹problems in predicting words in a sentence given context; ²an example of an application of this problem to speech recognition; ³an algorithm for estimating word probabilities based on the frequency of word pairs in a large corpus of text. We find that the class-based model produces better results than the standard n-gram model for predicting words in a sentence given context, especially for words that appear infrequently in the training corpus.

1. Introduction
In a number of natural language processing tasks, we face the problem of predicting a string of English words when it has been given by a person through a noisy channel. To tackle this problem, we want to make use of a large corpus of text, with which we can estimate the probability of each possible string of English words. This paper presents a new approach to this problem, called the class-based n-gram model, which is based on the observation that words in English tend to fall into well-defined semantic classes.

- neural method

One Billion Word Benchmark for Measuring Progress in Statistical...

We propose a new benchmark corpus to be used for measuring progress in statistical language modeling. With almost one billion words of training data, we hope this benchmark will be useful to...

 [X https://arxiv.org/abs/1312.3005](https://arxiv.org/abs/1312.3005)

<https://nlp.stanford.edu/pubs/glove.pdf>

2.2 Unsupervised Fine-tuning Approaches

- ELMo, GPT와 같이 미리 큰 데이터셋으로 사전 학습을 시킨 후, 특정 작업에 대해 미세 조정(fine-tuning)을 하는 방법을 다룹니다. GPT는 특히 Transformer의 디코더를 사용해 언어 모델을 학습시킵니다. 하지만 이는 주로 단방향(왼쪽에서 오른쪽 또는 그 반대) 정보만을 고려합니다.

2.3 Transfer Learning from Supervised Data

- 지도 학습(supervised learning)에서 얻은 지식을 다른 작업에 적용하는 전이 학습(transfer learning)에 관한 내용입니다. 예를 들어, 문장 분류 작업에서 학습된 모델을 다른 NLP 작업에도 적용할 수 있습니다. 그러나 이런 방법은 항상 레이블이 있는 데이터가 필요하다는 한계가 있습니다.

▼ 3. BERT (모델 설명)

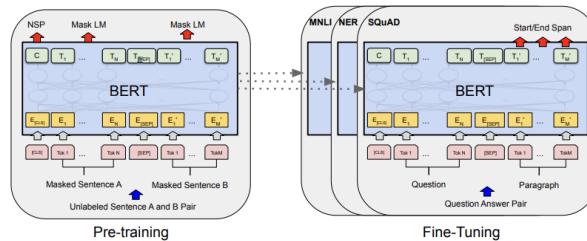


Figure 1: Overall pre-training and fine-tuning procedures for BERT. Apart from output layers, the same architectures are used in both pre-training and fine-tuning. The same pre-trained model parameters are used to initialize models for different downstream tasks. During fine-tuning, all parameters are 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).

3.1 Pre-training BERT

- **데이터 및 아키텍처:** BERT는 대규모의 언어 데이터(예: Wikipedia)를 사용하여 Transformer의 인코더 구조를 사전 학습합니다. 이때 아키텍처는 다양한 크기를 가질 수 있지만, 대표적으로 BERT-Base와 BERT-Large가 있습니다.
 - **Masked Language Model (MLM):** 전통적인 언어 모델은 단방향(왼쪽에서 오른쪽 또는 그 반대)만을 고려합니다. BERT는 양방향 정보를 고려하기 위해 MLM 작업을 사용합니다. 입력 문장에서 일부 단어를 무작위로 가린 뒤(hidden or 'masked') 이 가려진 단어를 예측하도록 학습됩니다.

- **Next Sentence Prediction (NSP)**: 두 문장이 주어졌을 때, 두 번째 문장이 첫 번째 문장 다음에 오는 문장인지를 판단하는 작업입니다. 이를 통해 모델은 문장 간의 관계를 더 잘 이해할 수 있습니다.

3.2 Fine-tuning BERT

- **작업 특화**: 사전 학습된 BERT 모델을 특정 NLP 작업에 맞게 미세 조정합니다. 여기에는 문장 분류, 개체 명명, 질문 응답 등이 포함될 수 있습니다.
- **데이터 및 학습**: 미세 조정은 일반적으로 작은 데이터셋에서도 효과적으로 이루어질 수 있습니다. BERT의 Transformer 인코더는 각 작업의 특성을 캡처할 수 있도록 학습됩니다.
- **양방향성의 이점**: 기존 모델들이 주로 단방향 정보만을 활용했다면, BERT는 양방향 정보를 활용하여 문맥을 더 정확하게 파악합니다. 이러한 문맥 정보는 특히 의미가 애매한 단어나 문장에서 더 정확한 결과를 도출하는 데 도움이 됩니다.

▼ 4. Experimental Results (실험 결과)

- 저도 정리를 하면서 찾은 내용인데, Downstream tasks에 관해 잘 정리된 내용이 있어 첨부합니다.

BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding
 논문 자료 : arXiv
<https://thebj.ai/bert/>

4.1 GLUE

- GLUE는 General Language Understanding Evaluation의 약자로 다양한 분야의 general language understanding task를 포함하고 이를 평가하고 있습니다
- 여기에는 문장 분류, 개체 명명 등 다양한 벤치마크가 존재합니다.

Dataset	Description	Data example	Metric
CoLA	Is the sentence grammatical or ungrammatical?	"This building is than that one." = Ungrammatical	Matthews
STS-2	Is the movie review positive, negative, or neutral?	"The movie is funny , smart , visually inventive , and most of all , alive ." = .93056 (Very Positive)	Accuracy
MRPC	Is the sentence B a paraphrase of sentence A?	A) "Yesterday , Taiwan reported 35 new infections , bringing the total number of cases to 418 ." B) "The island reported another 35 probable cases yesterday , taking its total to 418 ." = A Paraphrase	Accuracy / F1
STS-B	How similar are sentences A and B?	A) "Elephants are walking down a trail." B) "A herd of elephants are walking along a trail." = 4.6 (Very Similar)	Pearson / Spearman
QQP	Are the two questions similar?	A) "How can I increase the speed of my internet connection while using a VPN?" B) "How can Internet speed be increased by hacking through DNS?" = Not Similar	Accuracy / F1
MNLI-mm	Does sentence A entail or contradict sentence B?	A) "Tourist Information offices can be very helpful." B) "Tourist Information offices are never of any help." = Contradiction	Accuracy
QNLI	Does sentence B contain the answer to the question in sentence A?	A) "What is essential for the mating of the elements that create radio waves?" B) "Antennas are required by any radio receiver or transmitter to couple its electrical connection to the electromagnetic field." = Answerable	Accuracy
RTE	Does sentence A entail sentence B?	A) "In 2003, Yunus brought the microcredit revolution to the streets of Bangladesh to support more than 50,000 beggars, whom the Grameen Bank respectfully calls Struggling Members." B) "Yunus supported more than 50,000 Struggling Members." = Entailed	Accuracy
WNLI	Sentence B replaces sentence A's ambiguous pronoun with one of the nouns - is this the correct noun?	A) "Lily spoke to Donna, breaking her concentration." B) "Lily spoke to Donna, breaking Lily's concentration." = Incorrect Referent	Accuracy

• Table 1

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 _{BASE}	84.6/83.4	71.2	90.5	93.5	52.1	85.8	88.9	66.4	79.6
BERT _{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.

- 실험 결과 BERT Base,Large 모두 기존의 방법보다 좋은 성능을 보이고 있다.
- 혹시 벤치마크가 각 어떤 테스크인지 모르실 수도 있을 것 같아 첨부합니다.

4.2 SQuAD v1.1

- SQuAD v1.1(Stanford Question Answering Dataset)에서의 성능을 보여줍니다.
- 입력 : Context / question 쌍의 형태로 제공되며
- 출력 : Answer, 정수 쌍으로, Context 내에 포함된 답변 Text의 시작과 끝을 색인화 합니다.
- 주로 Exact Match(EM)과 F1 score 지표로 성능을 측정하며, BERT와 다른 모델들과의 성능 차이를 보여줍니다.

```
▼ paragraphs: [] 55 items
  ▼ 0:
    context: "Architecturally, the school has a Catholic character. Atop the Main Building's gold dome is a golden statue of the Virgin Mary. Immediately in front of the Main Building and facing it, is a copper statue of Christ with arms upraised with the legend "Venite Ad Me Omnes". Next to the Main Building is the Basilica of the Sacred Heart. Immediately behind the basilica is the Grotto, a Marian place of prayer and reflection. It is a replica of the grotto at Lourdes, France where the Virgin Mary reputedly appeared to Saint Bernadette Soubirous in 1858. At the end of the main drive (and in a direct line that connects through 3 statues and the Gold Dome), is a simple, modern stone statue of Mary."
  ▼ qas: [] 5 items
    ▼ 0:
      ▼ answers: [] 1 item
        ▼ 0:
          answer_start: 515
          text: "Saint Bernadette Soubirous"
          question: "To whom did the Virgin Mary allegedly appear in 1858 in Lourdes France?"
          id: "5733be284776f41900661182"
```

• Table 2

System	Dev		Test	
	EM	F1	EM	F1
Top Leaderboard Systems (Dec 10th, 2018)				
Human	-	-	82.3	91.2
#1 Ensemble - nlnet	-	-	86.9	90.5
#2 Ensemble - QANet	-	-	84.5	90.5
BIDAF+ELMo (Single)	85.6	-	85.8	-
R.M. Reader (Ensemble)	81.2	87.9	82.3	88.5
Ours	80.8	88.5	-	-
BERT _{BASE} (Single)	80.8	88.5	-	-
BERT _{MEDIUM} (Single)	84.1	90.9	-	-
BERT _{LARGE} (Single)	85.1	91.1	-	-
BERT _{LARGE} (Sgl+TrivialQA)	84.2	91.1	85.1	91.8
BERT _{LARGE} (En+TrivialQA)	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.

• Table 3

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	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.

4.3 SQuAD v2.0

- SQuAD는 여러 소스들에서 모인 질문/답변 쌍 데이터이다. 하나의 질문과, 그에 대한 답변 문단이 들어있고, 답변 문단 중에서 질문에 대한 답이 되는 구문을 찾는 것이 목적이다.



Figure 1: Training accuracies.

Problem Statement

The Stanford Question Answer Dataset (SQuAD) tackles the challenge of reading comprehension in machine learning.

The dataset itself comprises of questions posed over segments of articles sourced from Wikipedia, hereby referred to as “context.” For a given question, the dataset either provides the correct answer extracted from the context or sets the question as impossible if it does not have an answer.

The objective of this project was in two parts:

1. Predict an answer for each question;
2. Predict which questions do not have answers in the given context.

Overview

Input: A question and its corresponding context.
Output: The predicted answer within the context, or no answer for an impossible question.

- For training, a sampling rate of 50 was applied on the dataset to reduce the computational time. The data used was a subset of 2607 question/context/answer pairs.
- Testing was performed on a subset of 5213 question/context/answer pairs
- For each, the predicted answers were compared with the ground truth answers.

Approach

Model
The model used was a Bidirectional Encoder Representations from Transformers (BERT), pretrained for question answering.

Tokenizer

The tokenizer used on each question/context pair was a pre-trained BERT tokenizer from the bert-base-uncased vocabulary file.

Answer Prediction

The inputs were first tokenized and then processed through the BERT model. The model returns the likelihood for each index in the tokenized version of the input to be the start and the end of the answer segment. The argmax of the likelihoods for start and end indices is taken and used to index the answer from the context.

There may be some error in answer prediction due to the model not being able to run over arguments longer than

512 characters, which means that some inputs had to be reduced in length. If the answer were to be towards the end of the context there is a good chance it would get cut off from the input and thus impossible for the model to output the accurate answer.

Screening for impossible questions

The likelihoods of each index being the start or end was quantified from -10 to 10. The max usually ranged above 6 for questions with possible answers and below 6 for those without an answer. Values around 6 were used as cut-off between possible and impossible questions.

Results

The prediction accuracy was evaluated in the following ways:

- Exact match: prediction corresponds letter for letter to the ground truth answer.
- Partial match: prediction matches the distance between the prediction and the ground truth. Text distance measured using Jaro-Winkler.

For the training subset, the following exact match and partial match scores were obtained (Figure 3). Without screening for impossible answers, the accuracy obtained

were of 47.2% and 63.4% respectively. The highest accuracy was obtained when 6 was used as a baseline for either start or end index maximum likelihood.

```

Predictions: without any screening mechanism for impossible questions
Partial match: 0.4341516245036781
Exact match: 0.4720000000000001

Predictions: (i) torch.argmax_(index), (ii) torch.argmax_(index).item() > 4
Partial match: 0.4341516245036781
Exact match: 0.4720000000000001

Predictions: (i) torch.argmax_(index), (ii) torch.argmax_(index).item() > 4.5
Partial match: 0.4341516245036781
Exact match: 0.4720000000000001

Predictions: (i) torch.argmax_(index), (ii) torch.argmax_(index).item() > 4.5 OR torch.argmax_(index).item() > 4.5
Partial match: 0.4341516245036781
Exact match: 0.4720000000000001

Predictions: (i) torch.argmax_(index), (ii) torch.argmax_(index).item() > 4.5 OR torch.argmax_(index).item() > 4.5
Partial match: 0.4341516245036781
Exact match: 0.4720000000000001

```

Using 6 as a baseline, the model was tested on a larger subset and obtained similar accuracies of 59.6% for exact match and 74.0% for partial match.

Acknowledgements

I would like to express special thanks to the McGill AI Society its executives for their support throughout the entirety of the project.

References

1. Thomas Wolf and al. *HuggingFace’s Transformers: State-of-the-Art Natural Language Processing*, 2019, <https://huggingface.co/transformers>
2. Mohit Iyer et al. *String similarity — the basic know your algorithms guide* <https://treyv.io/string-similarity-the-basic-know-your-algorithms-guide-3de3d7246227>.
3. SQuAD2.0. <https://rajpurkar.github.io/SQuAD-explorer/>.

• Table 4

System	Dev	Test
ESIM+GloVe	51.9	52.7
ESIM+ELMo	59.1	59.2
OpenAI GPT	-	78.0
BERT _{BASE}	81.6	-
BERT _{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.

4.4 SWAG

- SWAG(Situation With Adversarial Generations)는 문맥을 이해하고 논리적 추론을 하는 능력을 테스트하는 데이터셋입니다.
- 이 테이블에서는 BERT가 얼마나 잘 추론을 하는지 다른 모델과 비교하여 보여줍니다.

On stage, a woman takes a seat at the piano. She
 a) sits on a bench as her sister plays with the doll.
 b) smiles with someone as the music plays.
 c) is in the crowd, watching the dancers.
 d) **nervously sets her fingers on the keys.**

A girl is going across a set of monkey bars. She
 a) jumps up across the monkey bars.
 b) struggles onto the monkey bars to grab her head.
 c) **gets to the end and stands on a wooden plank.**
 d) jumps up and does a back flip.

The woman is now blow drying the dog. The dog
 a) **is placed in the kennel next to a woman’s feet.**
 b) washes her face with the shampoo.
 c) walks into frame and walks towards the dog.
 d) tried to cut her face, so she is trying to do something very close to her face.

• Table 5

Tasks	Dev Set				
	MNLI-m (Acc)	QNLI (Acc)	MRPC (Acc)	SST-2 (Acc)	SQuAD (F1)
BERT _{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 BERT_{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.

- **No NSP:** MLM 사용 / NSP 미사용
- **LTR & No NSP:** MLM 대신 Left-to-Right 사용 / NLP 미사용
- NSP 태스크를 진행하지 않으면 자연어 추론 태스크(QNLI, MNLI)와 QA 태스크 (SQuAD)에서 큰 성능 하락이 있음
- MLM 대신 LTR이나 BiLSTM을 사용했을 때 MRPC와 SQuAD에서의 성능이 크게 하락함. MLM이 LTR과 BiLSTM보다 훨씬 깊은 양방향성을 뛴다.

▼ 5. Ablation Studies (성능 분석)

5.1 Effect of Pre-training Tasks

- 사전 훈련 작업의 종류가 모델 성능에 어떻게 영향을 미치는지를 분석합니다.
- 예를 들어, Masked Language Modeling(MLM)과 Next Sentence Prediction(NSP) 같은 다양한 작업을 이용해 어떤 것이 더 유용한지를 평가합니다.

5.2 Effect of Model Size

- 모델 크기 (예: 레이어 수, 파라미터 수 등)가 성능에 어떻게 영향을 미치는지 분석합니다.
- 이를 통해 모델 크기가 커질수록 성능이 얼마나 향상되는지, 그리고 언제 그 성능이 더 이상 향상되지 않는지를 확인할 수 있습니다.

5.3 Feature-based Approach with BERT

- BERT를 feature-based 방식으로 사용할 경우 성능이 어떻게 변하는지를 살펴봅니다.
- 예를 들어, BERT의 출력을 다른 모델의 입력으로 사용하는 경우와 BERT를 end-to-end로 훈련시키는 경우의 성능을 비교합니다.

• Table 6

• Table 7

Hyperparams				Dev Set Accuracy		
#L	#H	#A	LM (pp)	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 (pp)” is the masked LM perplexity of held-out training data.

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		
BERT _{LARGE}	96.6	92.8
BERT _{BASE}	96.4	92.4
Feature-based approach (BERT _{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.

▼ 6. Conclusion (결론)

- BERT는 사전 학습과 미세 조정을 통해 다양한 NLP 작업에서 뛰어난 성능을 보이며,
- 이를 통해 새로운 연구 및 애플리케이션의 가능성이 확대될 것이라는 결론을 내립니다.

7. 추후 방향성

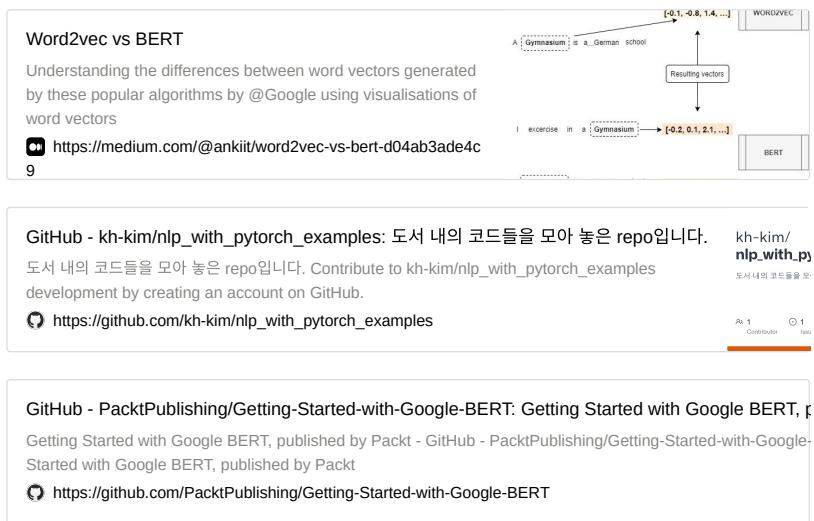
▼ 추가 정리 목표

- 사전 학습된 BERT 모델 추가 탐색()
- 사전 학습된 BERT에서 임베딩을 추출하는 방법
- BERT의 모든 인코더 레이어에서 임베딩을 추출하는 방법
- 다운스트림 태스크를 위한 BERT 파인 티닝 방법
- 다른 언어에 적용하는 법

8. 참고 링크

▼ 리스트

- 아래 링크들을 주요 내용으로 참고하여 내용을 작성하였습니다.



9. BERT 이전에 참고할 만한 논문

▼ 리스트

- Semi-supervised sequence tagging with bidirectional language models. : ELMo
- Attention is all you need. : Transformer
- Class-based n-gram models of natural language : non-neural word representation (1)
- A framework for learning predictive structures from multiple tasks and unlabeled data : non-neural word representation (2)
- Domain adaptation with structural correspondence learning : non-neural word representation (3)
- Distributed representations of words and phrases and their compositionality : neural word representation (1) (word2vec)
- Glove: Global vectors for word representation. : neural word representation (2) (GloVe)
- Semi-supervised sequence learning : fine-tuning representation (1)
- Universal language model fine-tuning for text classification : fine-tuning representation (2)
- Improving language understanding with Generative Pre-Training : fine-tuning representation (3) (GPT)

- Supervised Learning of Universal Sentence Representations from Natural Language Inference Data : NLI 를 이용해서 학습
- Learned in translation: Contextualized word vectors. : MT를 이용해서 학습 (CoVe)
- Google's neural machine translation system: Bridging the gap between human and machine translation. : WordPiece embedding

10. BERT 이후 관련 논문

▼ 파생 모델 1

- Multi-Task Deep Neural Networks for Natural Language Understanding
- RoBERTa: A Robustly Optimized BERT Pretraining Approach
- ALBERT: A LITE BERT FOR SELF-SUPERVISED LEARNING OF LANGUAGE REPRESENTATIONS
- ELECTRA: PRE-TRAINING TEXT ENCODERS AS DISCRIMINATORS RATHER THAN GENERATORS
- SpanBERT: Improving Pre-training by Representing and Predicting Spans

▼ 파생 모델 2(지식 증류 기반)

- DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter
- TinyBERT: Distilling BERT for Natural Language Understanding

11. 한국어 관련 LLM 모음

▼ 한국어 Task할 때 참고하면 좋아요

한국어 언어모델 (Korean Pre-trained Language Models) 둘아보기 (1)

최근 대규모 데이터를 이용하는 딥러닝 기반 자연어처리 연구가 활발합니다. 기업, 학계를 막론 ...

 <https://www.letr.ai/blog/tech-20220908>

한국어 언어모델 (Korean Pre-trained Language Models) 둘아보기 (2)

해외와 마찬가지로 한국어 역시 대량의 말뭉치를 통해 사전 학습된 Transformer를 기반으로 모델을 연구한 사례들이 많이 있습니다. KoBERT, KorBERT, HanBERT, KoELECTRA, KoGPT, Hyper CLOVA 등등 다양한 모델들이 발표되어왔죠. 이

 <https://www.letr.ai/blog/tech-20221124>

• Encoder-Centric Models: BERT 계열

모델	개발	학습 데이터	Tokenizer	Vocab	Params
KoBERT	ETRI	뉴스/책과시전 23GB	Morphology, WordPiece	30,349(Morphology) 30,791(WordPiece)	110M
KoBERT	SKT	위키피디아 50M	Sentence-Piece	8,002	92M
HanBERT	두글락 AI	일본/특허문서 70GB	Moran	54,000	128M
KoreALBERT	삼성 SDS	위키피디아/나무위키/뉴스/ 책 즐겨리 요약 등 43GB	Sentence-Piece	32,000	12M 18M
KLUE-BERT	Klue project	모두의영문자/ CC-100-Kor/ 나무위키/뉴스/청원 등 63GB	Morpheme-based subword	32,000	111M
KRBERT	서울대	위키피디아/뉴스	WordPiece	16,424(Character) 12,367(Subcharacter)	99M(Character) 96M(Subcharacter)
DistillKoBERT	개인(박장원)	위키피디아/나무위키/뉴스 등	Sentence-Piece	30,522	27.8M
KcBERT	개인(이준범)	네이버 뉴스의 댓글/대댓글	WordPiece	30,000	109M
KcELECTRA	개인(이준범)	네이버 뉴스의 댓글/대댓글	WordPiece	30,000	124M
KoBigBird	개인(박장원)	위키피디아/뉴스/모두의 말뭉치/Common Crawl 등	WordPiece	32,500	113.8M

• Decoder-Centric Models: GPT 계열

모델	개발	학습 데이터	Tokenizer	Vocab	Params
KoGPT2	SKT	위키피디어, 뉴스, 나무위키[10], 네이버 영화 리뷰, 한국어 commonCrawl 152M	Character BPE	51,200	125M
KoGPT-Trinity	SKT	Ko-DATA dataset 1.2B	-	51,200	1.2B
HyperCLOVA	NAVER	뉴스/카페/블로그/자신[1] 및 문서/댓글 등 네이버 수집 문서, 모두의 일봉지, 위키피디어 등 562B	Morpheme-aware byte-level BPE	-	82.0B
KoGPT	KakaoBrain	200B	-	64,512	6.0B

- Encoder-Decoder Models: Seq2seq 계열

모델	개발	학습 데이터	Tokenizer	Vocab	Params
KoBART	SKT	위키피디어/뉴스/책/ 모든 의약용 치/청와대 국민청원 등 0.27B	Character BPE	30,000	124M
KE-T5	KETI	한국어와 영어 데이터가 7.3B의 비율인 30GB 데이터	Sentence-Piece	64,000	247M
ET5	etri	위키피디어/뉴스/방송 대본/영화 드라마 대본 등 136GB	Sentence-Piece	45,100	60M
EXAONE	LG AI 연구원	말뭉치 600B, 이미지-텍스트 Pair 데이터 250M 이상	-	-	300B

▼ 금융권 한국어 BERT 기반 LM 모델

- 이번 학기에 KB국민은행과 산학 협력 프로젝트를 진행 중입니다.
- 금융권에서 언어모델을 어떻게 만들고 활용하는지 궁금하신 분들은 참고하시면 좋을 것 같습니다.

<https://prod-files-secure.s3.us-west-2.amazonaws.com/462d6414-91e6-4eeb-9ba6-f72fd930483f/ea661402-57cf-44ea-9ecf-e3a4c0c37a1c/KB-BERT.pdf>

12. 리뷰 후기



평소 NLP에 관심이 많아 BERT를 자주 마주치곤 했는데, 이번 기회에 자세하게 리뷰를 하면서 다시금 깊게 이해할 수 있어 좋은 경험이었습니다.
그리고 평소 전이 학습 된 모델을 주로 가져와서 쓰다 보니, 놓치는 점이 많았던 것 같았는데 이번 기회로 BERT 계열 모델들을 꾸준히 정리해보고자 합니다.