Week3 학습정리

Week3-1

1.Sparse Embedding

One-hot encoding : 단어를 벡터로 만들자

예) 커피 = [1 0 0 0]
최소 = [0 1 0 0]
3 = [0 0 1 0]
잔 = [0 0 0 1]

단점

- 고차원 (차원의 수 == vocab 사이즈)
  - 차원의 저주(curse of dimensionality)
- 단어 확장성 ↓
- 단어의 의미를 표현하지 못함
  - 모든 단어의 거리가 동일

```
x = [
    [1,0,0,0],
    [0,1,0,0],
    [0,0,1,0],
    [0,0,0,1],
]
```

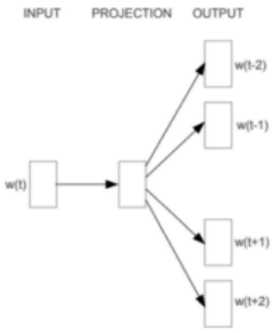
```
print(euclidean(x[0],x[1]))
print(euclidean(x[0],x[2]))
print(euclidean(x[0],x[3]))
```

```
1.4142135623730951
1.4142135623730951
1.4142135623730951
1.4142135623730951
```

    - 예) 유클리디안 거리 (l2 distance)

2. Skip-gram

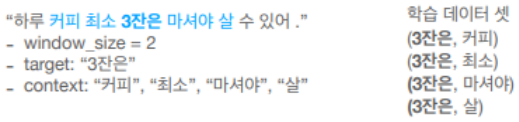
특징 : 나를 보고 주변 단어를 맞춰봐



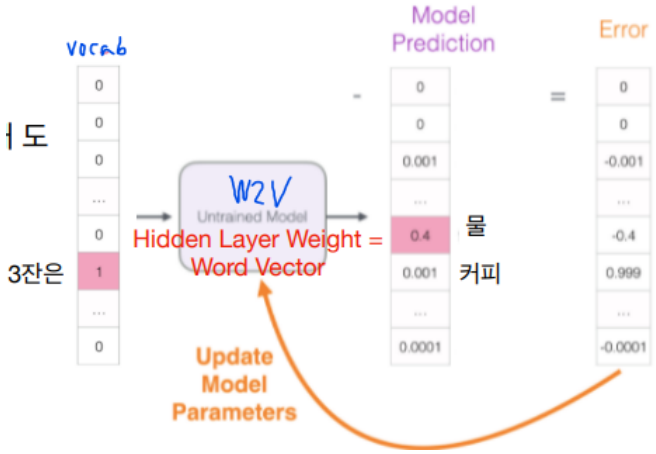
Skip-gram

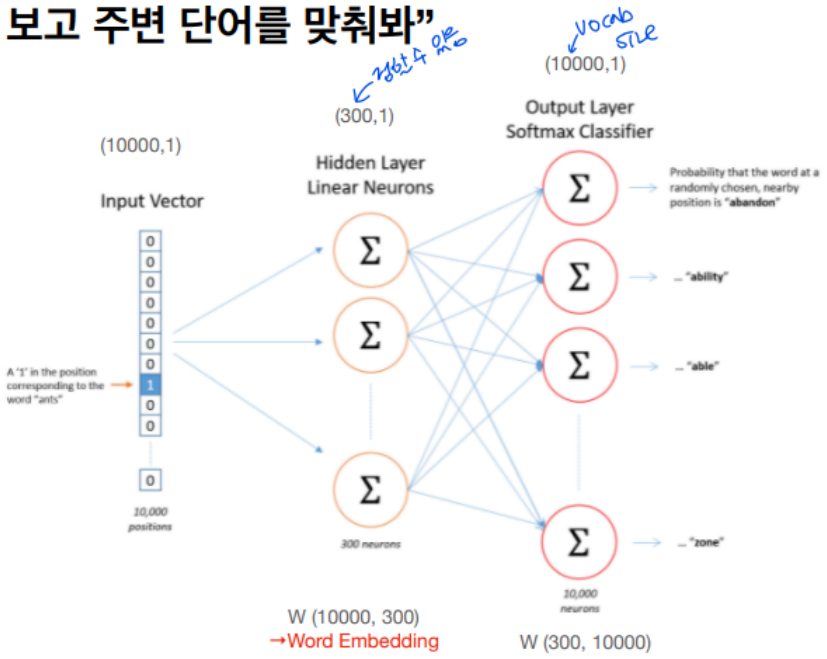
- [참고]

- context = 주변 단어
- target = 중심 단어
- window_size = (한 방향의) 주변 단어 개수
  - context 개수 = window_size * 2



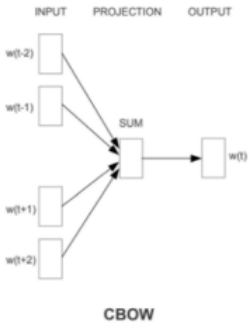
- 특정 단어(target)가 입력 됐을 때 그 주변 단어(context)를 맞추는 문제
- **주변 단어의 순서는 고려하지 않음 ⇒ 거리 개념 학습 안됨**
- 장점: 빈도 낮은 단어의 의미도 잘 잡아냄. 학습 데이터가 적을 때 도





3. CBOW

특징 : 주변 단어를 보고 나를 맞춰봐



CBOW

- 주변 단어(context)를 보고 특정 단어(target)를 예측
- 주변 단어의 순서는 고려하지 않음
- 장점: 빈도 높은 단어 의미 더 정확. 학습 속도 빠름

"하루 커피 최소 **3잔은** 마셔야 살 수 있어 ."
- window_size = 2
- target: "3잔은"
- context: "커피", "최소", "마셔야", "살"

학습 데이터 셋
(커피,3잔은)
(최소, 3잔은)
(마셔야, 3잔은)
(살, 3잔은)

- Ouput shape : (10000, 4)

4. Word2vec

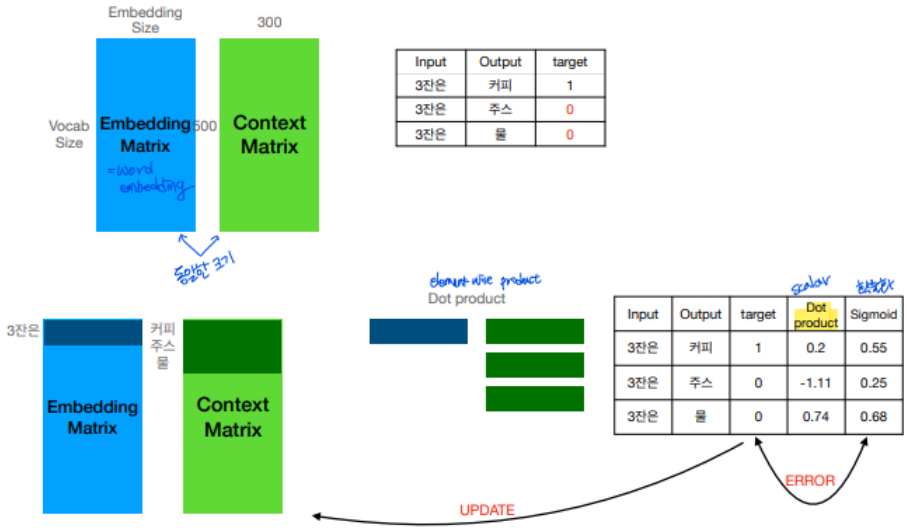
특징 : 간단한 신경 모델로 단어 embedding 학습하자

- Skip-gram 모델의 단점
  - weight의 크기가 큼 (이전 예제에서만 300*100002* = 6백만)
  - 데이터 셋이 큼
  - 따라서 학습 시간 오래 걸림
- 제안1. Subsampling Frequent Words
  - 자주 등장하는 단어를 코퍼스에서 제거함. 따라서 학습 데이터셋에 단어 등장 빈도를 줄임.
- 제안2. Negative Sampling (SKip-gram Negative Sampling; SKNS)
  - Logistic Regression task로 변형 (softmax -> sigmoid) + negative sampling (노이즈 추가)해 복잡도 감소

Input	Output	target
3잔은	커피	1
3잔은	최소	1
3잔은	마셔야	1
3잔은	살	1

Input	Output	target
3잔은	커피	1
3잔은	주스	0
3잔은	물	0

- 학습 과정에서 Embedding Matrix, Context Matrix를 업데이트
- Embedding Matrix를 word embedding으로 사용함
- Hyperparameter
  - Window size (context: (window_size-1)/2)
  - # negative samples
    - 데이터 셋 규모 ↓ : 5~10 / 데이터 셋 규모 ↑ : 2~5
  - Embedding size



Input	Output	target
3잔은	커피	1
3잔은	주스	0
3잔은	물	0

Input	Output	target	Dot product	Sigmoid
3잔은	커피	1	0.2	0.55
3잔은	주스	0	-1.11	0.25
3잔은	물	0	0.74	0.68

5. GloVe

특징 : 통계 정보로 단어의 embedding을 만들자

- Local context window 방식(skip-gram)은 문서 전체의 통계 정보를 반영하지 못한다는 단점이 있음. 따라서 단점을 보완하고자 **전체 통계 정보를 반영하는 방법론**.
- Global matrix factorization
  - Latent Semantic Analysis (LSA)를 발전시킴
    - Latent Semantic Analysis (LSA) : 토픽 모델링에서 사용됨
  - 단어가 동시에 등장하는 빈도수를 계산한 행렬(word-word co-occurrence statistics)을 생성
  - P(k|i) 특정 단어 i가 등장했을 때 주변 단어 k가 등장한 확률 계산
  - 중심 단어와 주변 단어 벡터 내적 == **동시 등장 확률**

n=1
"모닝 커피 좋아"

	모닝	커피	좋아
모닝	0	1	0
커피	1	0	1
조아	0	1	0

Probability and Ratio	$k = solid$	$k = gas$	$k = water$	$k = fashion$
$P(k\|ice)$	1.9×10^{-4}	6.6×10^{-5}	3.0×10^{-3}	1.7×10^{-5}
$P(k\|steam)$	2.2×10^{-5}	7.8×10^{-4}	2.2×10^{-3}	1.8×10^{-5}
$P(k\|ice)/P(k\|steam)$	8.9	8.5×10^{-2}	1.36	0.96

6. fastText

특징 : 단어 embedding이 아니라 **subword embedding을 만들자**

- 단어 → bag of character n-gram(단어를 character 단위로 쪼갠다)
- 단어 벡터를 학습하는 대신, n-gram character 벡터를 학습함
- 단어는 n-gram character 벡터의 합으로 표현
- **Skip-gram 모델을 사용**
- 장점1: 학습 데이터에 등장하지 않았거나 빈도가 낮은 단어도 표현 가능 (OOV 문제 해결)
- 장점2: 접두사, 접미사 의미 학습 가능

Word: artificial n=3 <ar, art, rti, tif, ifi, fic, ici, ial, al>

7. Reference

- Illustrated Word2vec ⭐⭐⭐
- [Skip-gram architecture explained](http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skipgram-model/) ⭐⭐⭐
- [skip-gram&CBOW explain](https://towardsdatascience.com/nlp-101-word2vec-skip-gram-andcbow-93512ee24314) ⭐
- [word2vec explained](http://mccormickml.com/2017/01/11/word2vec-tutorial-part-2-negativesampling/) ⭐
- skip-gram, CBOW 논문 ⭐⭐⭐
- GloVe 한국어 설명 ⭐⭐⭐
- word2vec 논문 ⭐⭐⭐
- GloVe 논문 ⭐
- fastText 논문 ⭐

Week3-2

Subword (segmentation) Tokenizer

Subword segmentation은 단어 기반 토크나이즈 방식이 갖는 OOV(Out Of Vocabulary) 문제를 해결하는 토크나이즈 방식입니다. 단어 보다 더 작은 단위인 “subword” 단위로 토큰을 분리하는데 이러한 방식은 신조어나 학습 코퍼스에 없던 새로운 단어도 사전에 있는 subword의 조합으로 표현할 수 있다는 장점이 있습니다. 뿐만 아니라 영어는 prefix, suffix가 의미를 가지고 있는 경우가 많습니다. 예를 들면, “fearless”란 단어는 “fear” + “less” subword로 이루어져 있는데 각각의 subword가 의미를 가지고 있습니다. 이렇게 subword tokenizer는 prefix 및 suffix의 의미를 살릴 수 있다는 장점이 있습니다.

한국어 예시를 들어보겠습니다. “킹받네"라는 말은 요즘 유행하는 신조어입니다. 보통 언어 모델은 대량의 코퍼스로 학습하기 때문에 과거 수집된 문서 위주로 학습될 가능성이 큽니다. 만약 단어 기반 토크나이즈를 사용하면 “킹받네"와 같은 단어는 사전에 없기 때문에 단어의 벡터 값을 얻을 수 없습니다 (OOV). 하지만 subword 토크나이저(Sentence-piece 사용)는 해당 단어를 ['킹', '##받', '##네'] 로 분리하기 때문에 subword의 조합으로 신조어를 표현할 수 있습니다. 따라서 OOV 비율이 줄어드는 효과를 볼 수 있습니다.

BPE (Byte Pair Encoding)

BPE는 다음의 프로세스를 n번 반복해 토크나이징을 합니다. 1. 문서의 모든 단어를 character 단위로 분리

2. 등장 빈도가 가장 높은 character pair(2쌍의 character)를 찾아 하나의 character로 병합

- BPE 논문
- 한국어 BPE 설명 및 학습 코드 사이트

WordPiece

구글은 WordPiece 토크나이저 코드를 공개하지 않았습니다. 대신 HuggingFace Tokenizer 라이브러리에서 WordPiece 토크나이저 클래스를 공개했기 때문에 해당 클래스로 학습할 수 있습니다.

- WordPiece 논문

SentencePiece

SentencePiece는 end-to-end 텍스트 토크나이저로 별도의 텍스트 전처리 작업을 할 필요 없이 텍스트를 입력값으로 받으면 사전에 지정된 vocab size만큼의 사전을 구축합니다. BPE과 Unigram Language Model 알고리즘이 구현되어 있습니다.

BERT 모델은 WordPiece Tokenizer를 사용했지만 해당 tokenizer 코드가 공개되지 않아 많은 모델들은 WordPiece 대신 SentencePiece를 사용하기도 합니다.

- SentencePiece 논문
- SentencePiece Github repository

Must Read

- HuggingFace Tokenizer Library - Tokenizer Summary
    - Tokenizer 라이브러리는 rust 언어로 구현되어 있어 python으로 구현된 Transformer 라이브러리보다 학습 속도가 빠릅니다.

Week3-3. NLP models - Why and how to read papers

NLP Researcher & Engineer가 논문을 읽어야만 하는 이유

NLP Researcher/Engineer == 응용 과학자

- 새로 나온 기술을 가장 먼저 내가 풀고자 하는 문제를 풀기 위한 도구로 적용하는 사람

기술 전파 속도

- 논문, 기업 기술 블로그, 트위터(최신 지식) >>>>> 학교 강의, 책, 강좌(보편타당 지식 위주)

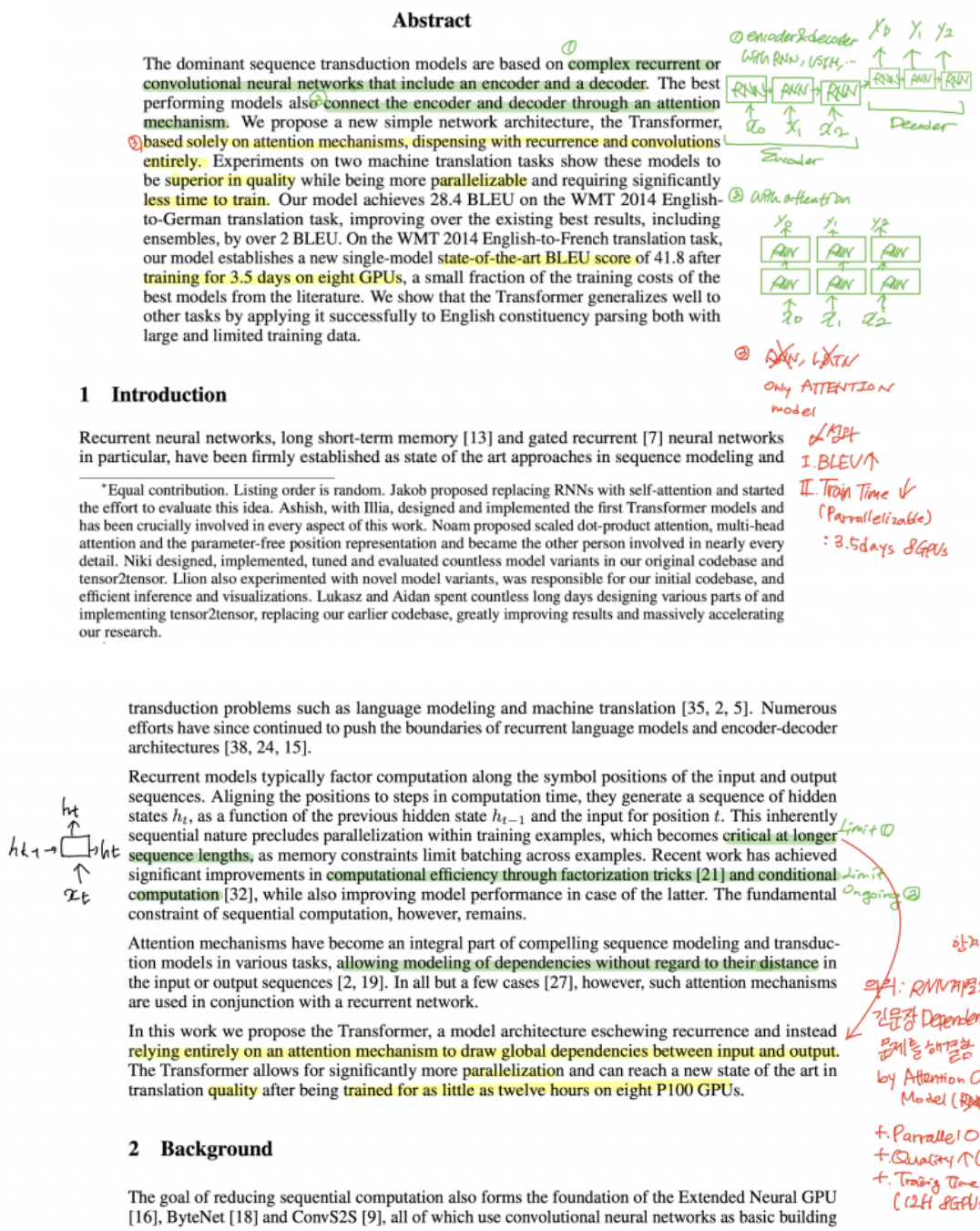
프로세스

- 새로운 모델의 등장 → 논문 읽기 + (오픈 소스 코드 확인) → 내가 가지고 있는 데이터셋 or 목적 함수에 맞게 변형 또는 적용
- 위 프로세스를 무한 반복

논문 읽는 방법 소개

[Basic] 이분법적으로 접근하기

- 문제점(노랑) ↔ 해결점(초록)
- 과거(노랑) ↔ 현재(초록)



Abstract

The dominant sequence transduction models are based on complex recurrent or convolutional neural networks that include an encoder and a decoder. The best performing models also connect the encoder and decoder through an attention mechanism. We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely. Experiments on two machine translation tasks show these models to be superior in quality while being more parallelizable and requiring significantly less time to train. Our model achieves 28.4 BLEU on the WMT 2014 English-to-German translation task, improving over the existing best results, including ensembles, by over 2 BLEU. On the WMT 2014 English-to-French translation task, our model establishes a new single-model state-of-the-art BLEU score of 41.8 after training for 3.5 days on eight GPUs, a small fraction of the training costs of the best models from the literature. We show that the Transformer generalizes well to other tasks by applying it successfully to English constituency parsing both with large and limited training data.

1 Introduction

Recurrent neural networks, long short-term memory [13] and gated recurrent [7] neural networks in particular, have been firmly established as state of the art approaches in sequence modeling and transduction problems such as language modeling and machine translation [35, 2, 5]. Numerous efforts have since continued to push the boundaries of recurrent language models and encoder-decoder architectures [38, 24, 15].

*Equal contribution. Listing order is random. Jakob proposed replacing RNNs with self-attention and started the effort to evaluate this idea. Ashish, with Illia, designed and implemented the first Transformer models and has been crucially involved in every aspect of this work. Noam proposed scaled dot-product attention, multi-head attention and the parameter-free position representation and became the other person involved in nearly every detail. Niki designed, implemented, tuned and evaluated countless model variants in our original codebase and tensor2tensor. Llion also experimented with novel model variants, was responsible for our initial codebase, and efficient inference and visualizations. Lukasz and Aidan spent countless long days designing various parts of and implementing tensor2tensor, replacing our earlier codebase, greatly improving results and massively accelerating our research.

Recurrent models typically factor computation along the symbol positions of the input and output sequences. Aligning the positions to steps in computation time, they generate a sequence of hidden states h_t, as a function of the previous hidden state h_{t-1} and the input for position t. This inherently sequential nature precludes parallelization within training examples, which becomes critical at longer sequence lengths, as memory constraints limit batching across examples. Recent work has achieved significant improvements in computational efficiency through factorization tricks [21] and conditional computation [32], while also improving model performance in case of the latter. The fundamental constraint of sequential computation, however, remains.

Attention mechanisms have become an integral part of compelling sequence modeling and transduction models in various tasks, allowing modeling of dependencies without regard to their distance in the input or output sequences [2, 19]. In all but a few cases [27], however, such attention mechanisms are used in conjunction with a recurrent network.

In this work we propose the Transformer, a model architecture eschewing recurrence and instead relying entirely on an attention mechanism to draw global dependencies between input and output. The Transformer allows for significantly more parallelization and can reach a new state of the art in translation quality after being trained for as little as twelve hours on eight P100 GPUs.

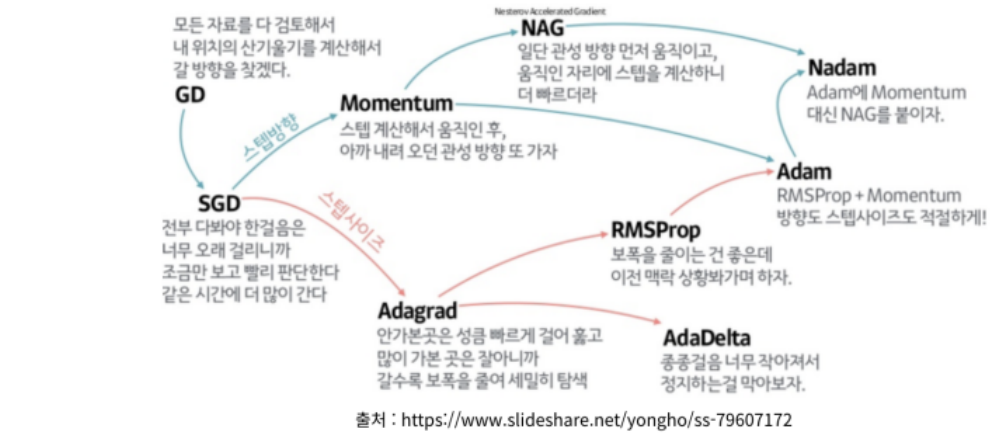
2 Background

The goal of reducing sequential computation also forms the foundation of the Extended Neural GPU [16], ByteNet [18] and ConvS2S [9], all of which use convolutional neural networks as basic building

[Advanced] 큰 흐름을 이해하고 해당 논문을 범주화 하기

- 방향성을 가지 치기
  - 예 : 옵티마이저의 종류

<Optimizer의 종류>



출처 : https://www.slideshare.net/yongho/ss-79607172

Reference

- blog on reading nlp papers : https://towardsdatascience.com/how-to-learn-deep-learning-by-reading-papers-c51b6025f226 ⭐⭐

Week3-3. NLP models - Transformer, Pre-trained Language Models (PLM)

목표 : Contextual Embedding이 무엇인지 설명할 수 있고 transformer와 bert의 모델 구조를 이해한다.

Contextual Embedding Models

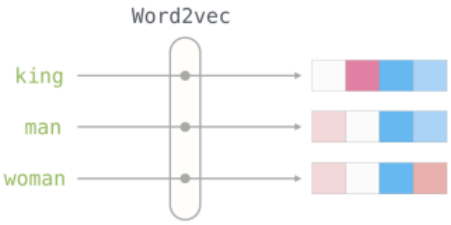
특징 : 모델이 문맥에 맞는 vector를 구해줄거야

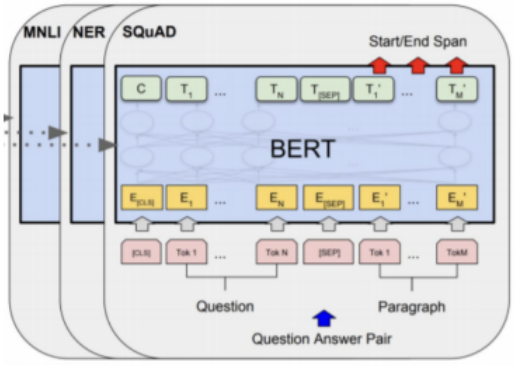
Word2vec 모델의 단점

- 단어의 의미를 vector화 할 수 있지만, 한 단어가 가지고 있는 여러 의미를 vector로 표현할 수 없음 → Fixed Embedding
- Input: Words (BOW) ex. “눈”
- Output: Fixed Word Embedding

Contextual Embedding 등장

- 하나의 단어가 **문맥에 따라 여러가지 vector로 표현**
- Input: Sentence (word1, word2, word3, …)
  - ex. “너 참 눈이 예쁘다” vs “눈이 내린다” : 눈의 뜻 다름
- Output: Contextualized Word Embedding
  - By: Model Weight





ELMO(Embedding from Language MOdels)

NER 문제를 해결할 때 LSTM 많이 쓰이기도 함

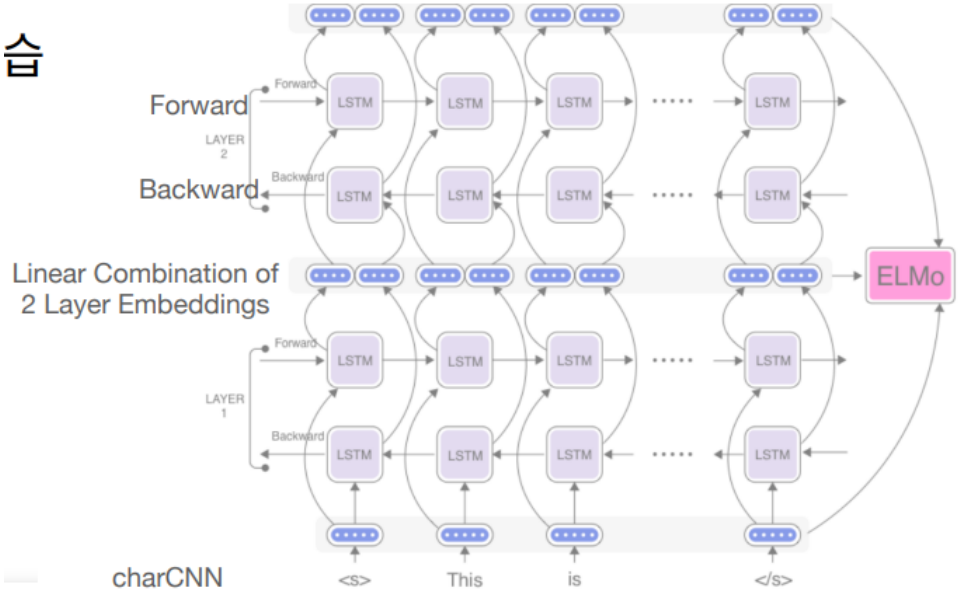
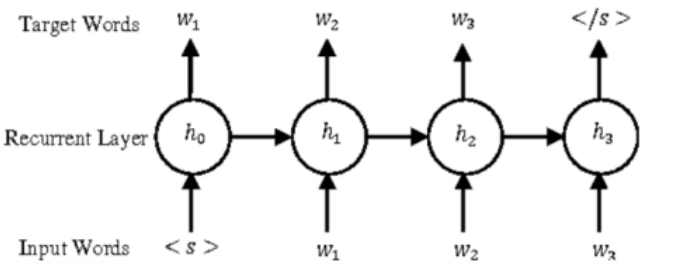


Figure: ELMo uses internal representations of multi-layer biLM

특징

- biLSTM
- Deep representation (linear combination of 2 layers)
  - 1번째 레이어 특징 : syntactic → 품사 태깅, 개체명 인식에서 유용
  - 2번째 레이어 특징 : semantic → 문장 유사도에서 유용
  - 1, 2번째 레이어 조합으로 좋은 임베딩 획득 가능
- Pre-trained Language Model : ELMO로 LM을 풀면서 Contextual embedding 학습
  - 30M (30,000,000) 문장 학습
  - **L**anguage **M**odel (un-supervised)
    - 라벨링 없이 코퍼스만 사용해 다음 단어 예측 문제



Transformer

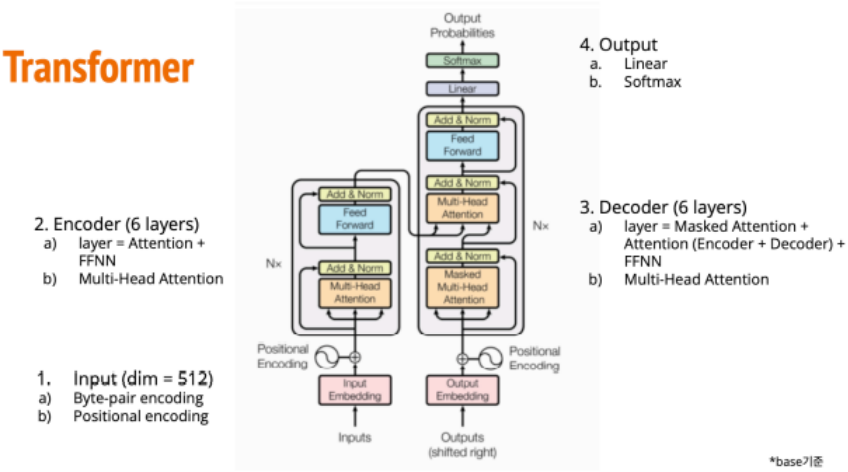
Attention is all you need 논문에서 제안한 구조

LSTM 단점 (recurrent model의 단점)

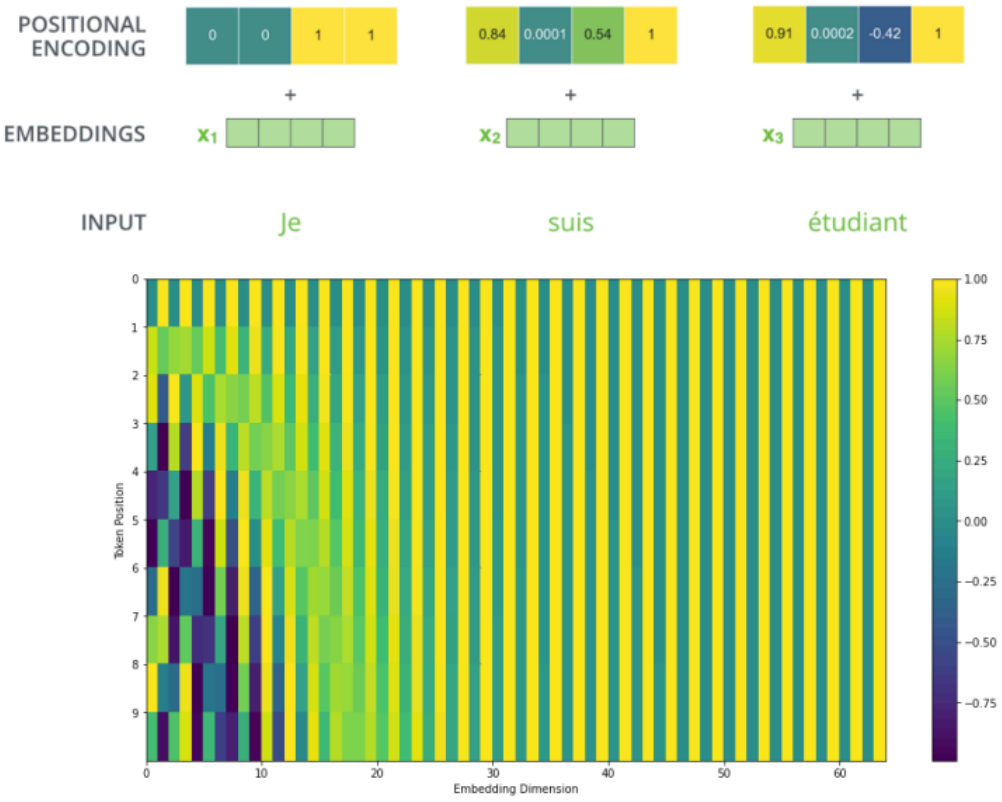
- Long term dependency : 문장 길수록 역전파에 따른 기울기가 소실됨, 첫번째 문장 토큰의 영향력이 상실됨
- 병렬 학습이 불가능 : weight가 순차적으로 전달되기 때문에

Transformer는 LSTM 단점 극복 : 문장과 문장의 pair-wise 곱으로 attention 연산(시각화 가능)으로 장기 의존성 문제 해소

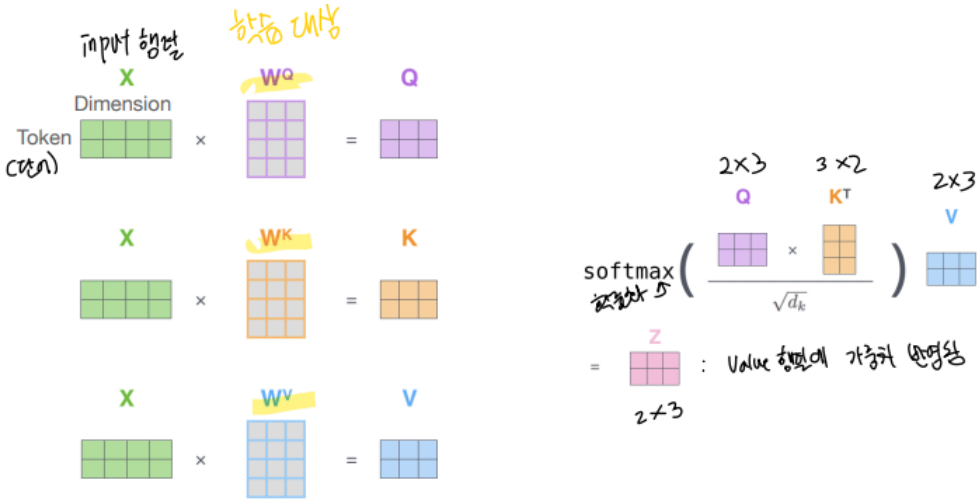
- Encoder + Decoder → 기계 독해
- Encoder → BERT (MaskedLM 사용해 bi-directional 가능케), NLU
- Decoder → GPT, NLG



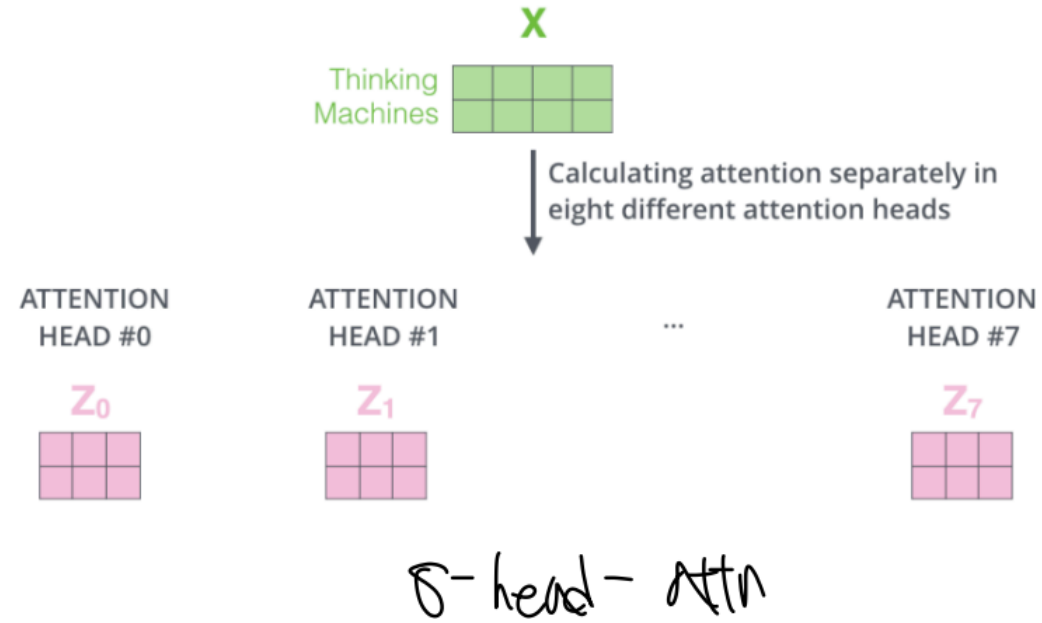
- Positional encoding : 단어의 순서를 주기 위해서 사인 함수 사용
  - 첫 단어의 위치 인코딩 시각화



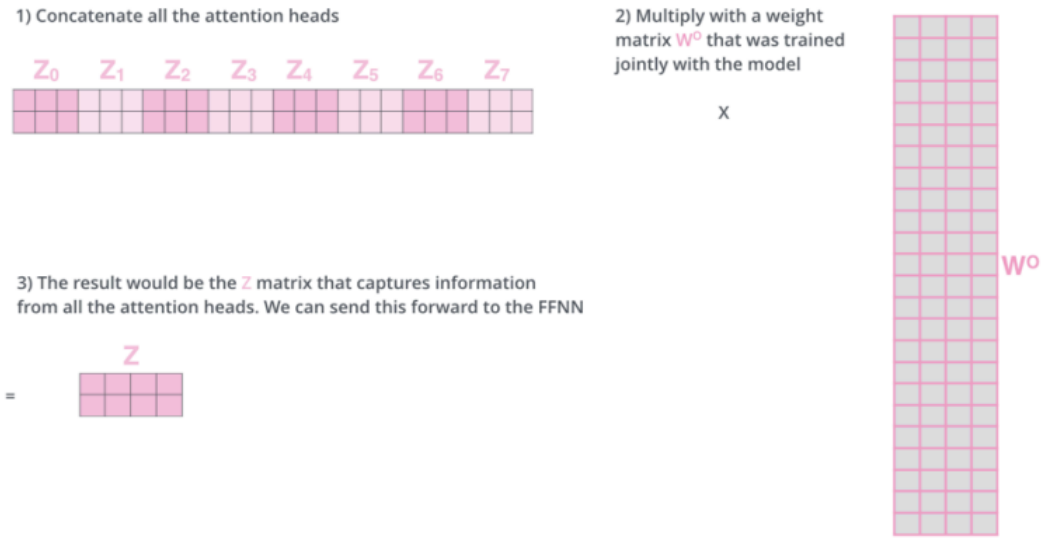
- Multi-head self-attention
  1. sequence의 Key, Query, Value matrix 생성
  2. Key * Query를 weight 삼아 Value와 곱해 attention matrix 생성



  3. Self attention을 여러번 계산 해 8개 (base 기준)의 attention matrix 생성



4. 8개의 attention matrix를 concat한 후 weight matrix와 곱해 차원을 맞춰줌



BERT

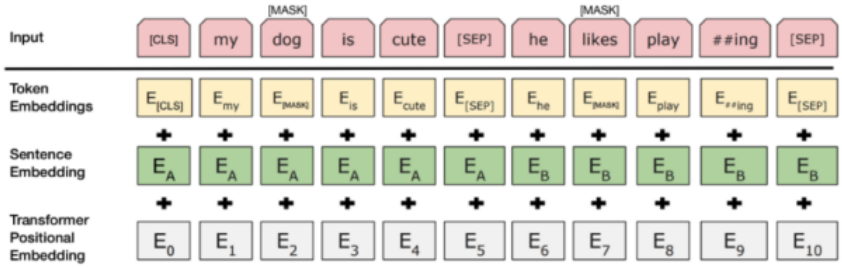
특징 : Attention으로 양방향

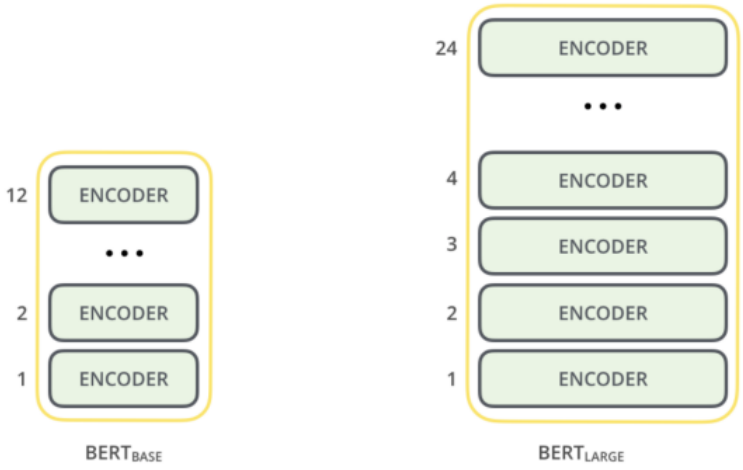
모델 구조

- Input Layer ← Token Embedding + Sentence Embedding +Positional Embedding(≠positional encoding)
  - 입력 : 2 문장 (QA 특화)

- 토큰 임베딩 : 토큰에 대한 벡터 매핑
- 문장 임베딩 : 단어의 소속 문장 표기
- 트랜스포머 위치 임베딩 : 순서를 표기

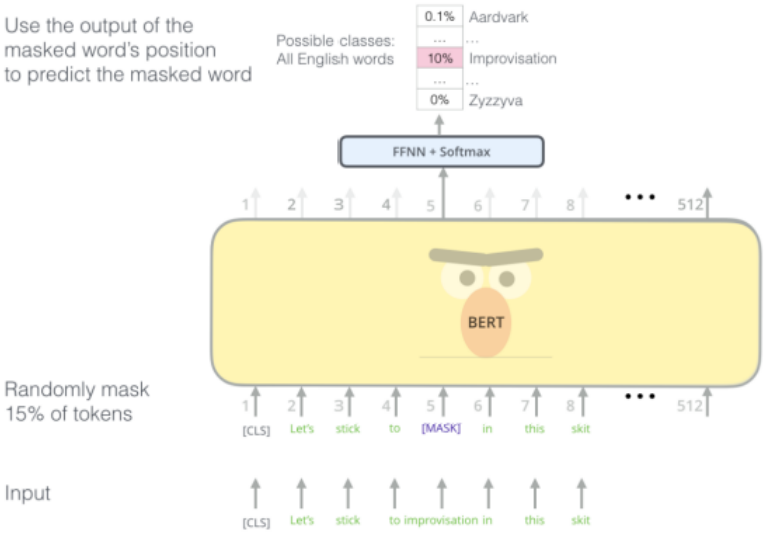
- 12 Encoder Layers



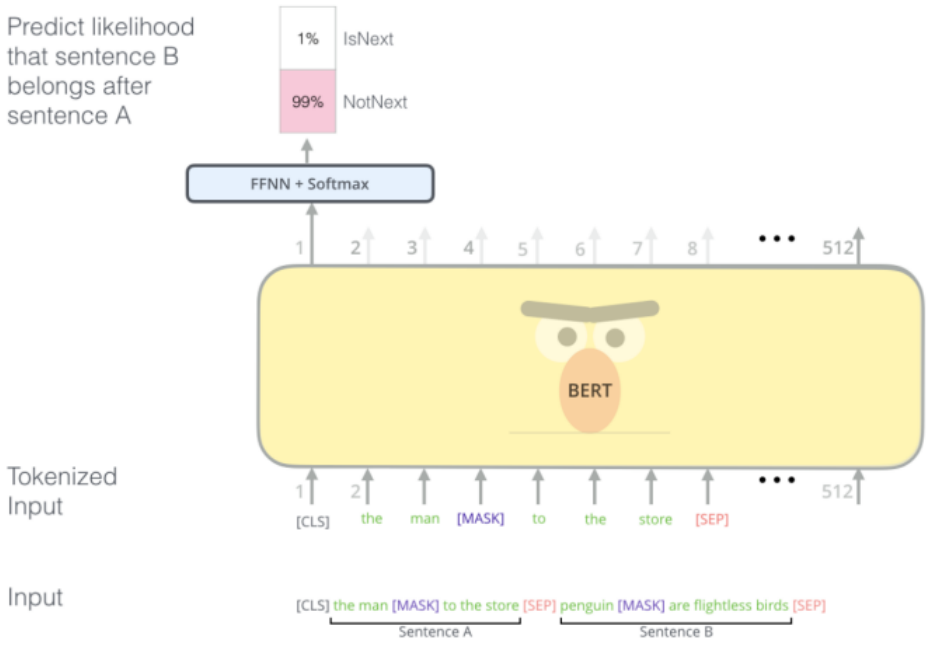


Pre-training 2 tasks

1. Language Model → Masked LM (15% 토큰에 [MASK])



2. Next Sentence Prediction (50% : 50%) : 의미론적 문장 간 관계를 학습



자주 사용되는 parameters

class transformers.BertConfig <source>

```
( vocab_size = 30522, hidden_size = 768, num_hidden_layers = 12, num_attention_heads = 12,
intermediate_size = 3072, hidden_act = 'gelu', hidden_dropout_prob = 0.1,
attention_probs_dropout_prob = 0.1, max_position_embeddings = 512, type_vocab_size = 2,
initializer_range = 0.02, layer_norm_eps = 1e-12, pad_token_id = 0, position_embedding_type =
'absolute', use_cache = True, classifier_dropout = None, **kwargs )
```

- vocab_size = 토크나이저 학습으로 만든 token lookup table
  - 자주 나오는 단어 서브워드 토크나이징, 빈도 수 적은 단어는 그대로 사용
  - vocab size는 Masked LM에서 클래스 개수(30522개)가 됨

Corpus: “나는 코딩이 좋아요! 꺍꺍꺍”
{“나” : 0,
“##는” : 1,
“코딩” : 2,
“##이” : 3,
“좋” : 4,
“##아” : 5,
“##요” : 6,
“##!” : 7,
“꺍꺍꺍” : 8}

- hidden_size = 토큰 벡터의 차원
  - 하나의 토큰이 768차원으로 표현
- num_hidden_layers = # 인코더
- num_attention_heads = 각 인코더 레이어의 # attention head

- max_position_embedding = 최대 입력 토큰의 개수 == 모델에 입력되는 토큰의 최대 개수

Reference

- Illustrated Transformer ⭐⭐⭐
- Illustrated ELMo & BERT ⭐⭐⭐
- Annotated Transformer ⭐⭐
- HuggingFace ⭐⭐
- BERT embeddings ⭐
- Attention is all you need ⭐⭐⭐
- BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding ⭐⭐⭐

Week3-4 Contextual Embedding (2) GPT, ALBERT, RoBERTa

목표 : Auto-regression을 설명할 수 있고 GPT 모델 및 BERT 이후에 나온 모델의 특징을 설명할 수 있다.

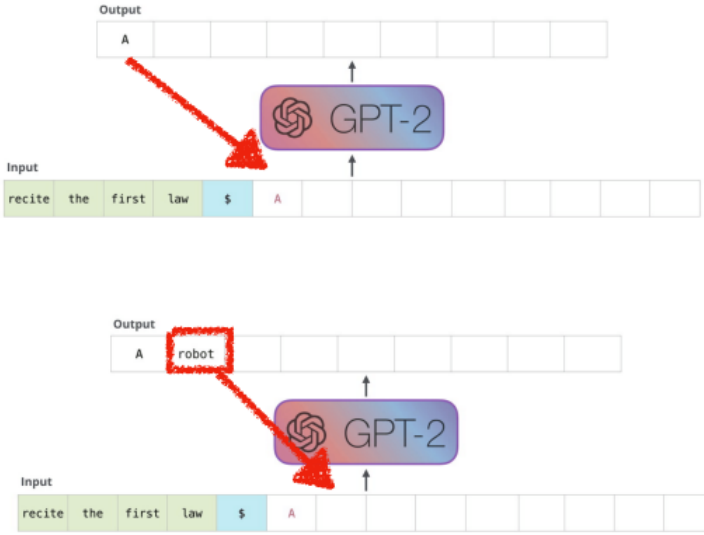
Overview

- Auto regression : 디코더, 문장 생성
  - GPT2
  - XLNet
  - …
- Encoder : NLU에서 사용
  - BERT
  - ALBERT
  - RoBERTa
  - Electra : BERT의 태스크를 바꿔 봄
  - … : 어텐션 방식을 바꿔보자 등

GPT(Generative Pre-trained Transformer)

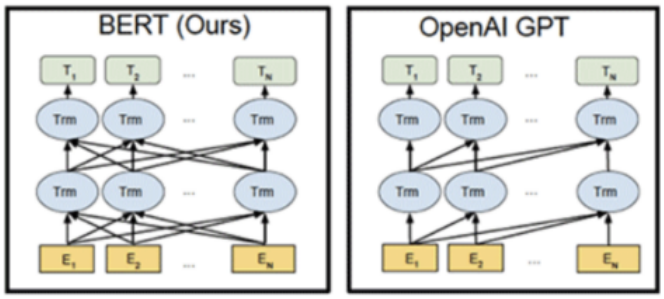
디코딩할 수 있는 사전학습된 트랜스포머

- Transformer Decoder
- Auto-regressive model : 생성된 값이 다음 스텝의 입력값이 됨.
  - Left-to-right transformers



BERT : Bidirectional transformers, 모든 토큰을 어텐션함.

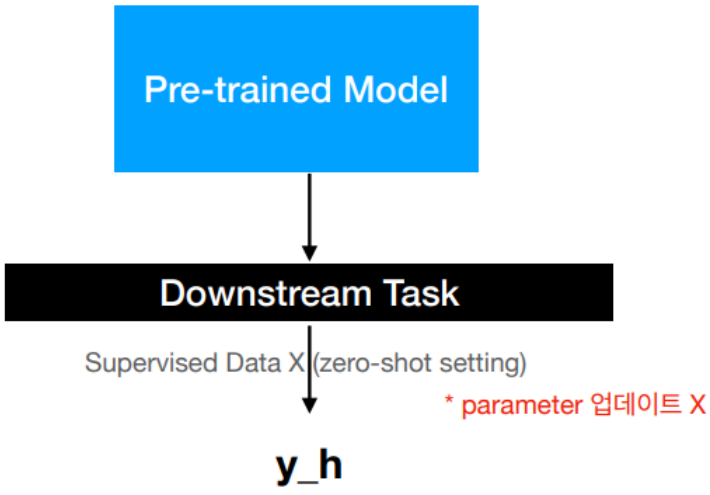
GPT : 이전 단어 보고 다음 단어 예측, 다음 단어가 사용되지 않음.



GPT2

논문 : Language Models Are Unsupervised Multitask Learners

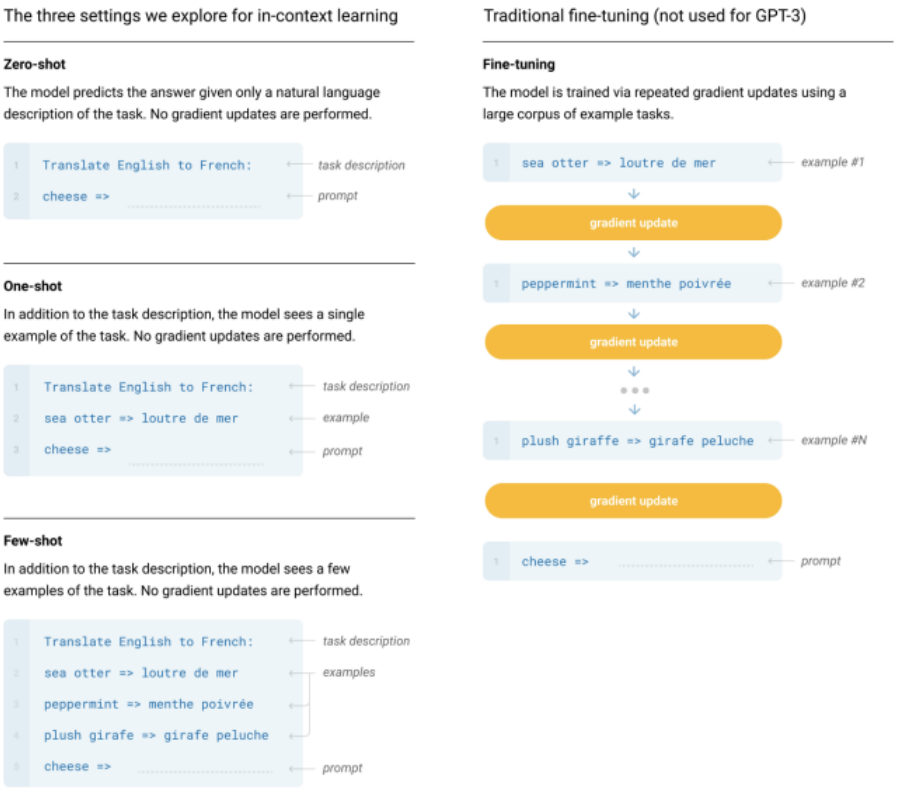
- (파인튜닝하지 않고) 적은 데이터로 문제를 풀 수 있게 거대한 모델을 만들자!



- Meta Learning : 모델이 여러 태스크 학습, 학습하는 방법을 모델이 학습함.
- Multi-task Learning
- GPT 원 모델 아키텍처 유사
  - GPT보다 레이어 개수와 단어 수가 매우 많음
  - 가중치 초기화 방법도 다름
- BPE (Byte Pair Encoding) : 임베딩 방법
- Application
  - QA
  - Machine Translation
  - Reading Comprehension
  - Summarization

Few-shot Learning

- 현업에서 데이터 구하기 힘들어서 유망함. 좋은 연구 주제.



ALBERT

A Lite BERT for Self-supervised Learning of Language Representations

기존 문제점

- 모델 규모가 커지면 자원의 한계로 상용화가 어려움
- 모델의 파라미터 수 증가가 성능과 완전 비례하지 않음

Parameter Reduction (건강한 파라미터 수 다이어트⇒파라미터 수 줄여도 SOTA 기록하기 때문)

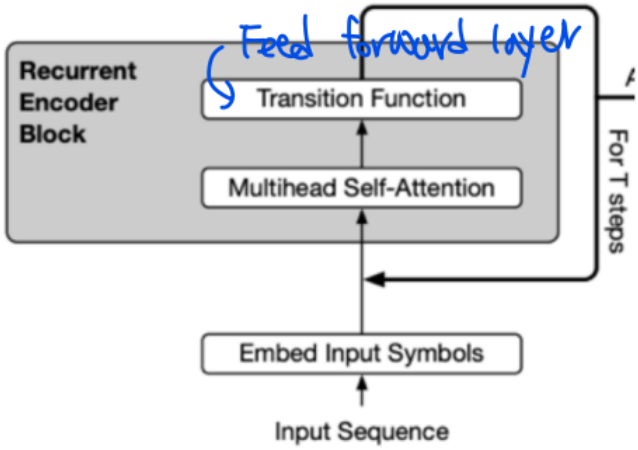
- 수단

- Factorized embedding parameterization

```
AlbertModel(
  (embeddings): AlbertEmbeddings(
    (word_embeddings): Embedding(30000, 128, padding_idx
    (position_embeddings): Embedding(512, 128)
    (token_type_embeddings): Embedding(2, 128)
    (LayerNorm): LayerNorm((128,), eps=1e-12, elementwis
    (dropout): Dropout(p=0, inplace=False)
  )
  (encoder): AlbertTransformer(
    (embedding_hidden_mapping_in): Linear(in_features=12
    (albert_layer_groups): ModuleList(
      (0): AlbertLayerGroup(
        (albert_layers): ModuleList(
          (0): AlbertLayer(
            (full_layer_layer_norm): LayerNorm((768,), e
            (attention): AlbertAttention(
              (query): Linear(in_features=768, out_featu
              (key): Linear(in_features=768, out_feature
              (value): Linear(in_features=768, out_featu
              (attention_dropout): Dropout(p=0, inplace=
              (output_dropout): Dropout(p=0, inplace=Fal
              (dense): Linear(in_features=768, out_featu
              (LayerNorm): LayerNorm((768,), eps=1e-12,
            )
            (ffn): Linear(in_features=768, out_features=
            (ffn_output): Linear(in_features=3072, out_f
            (dropout): Dropout(p=0, inplace=False)
          )
        )
      )
```

size

  - ALBERT : Input embedding dimension(128) < hidden size(768)
  - BERT : Input embedding dimension(768) == hidden size(768)
- Cross layer parameter sharing : 레이어 간 파라미터를 공유함



  - BERT : 12개 레이어의 Attn 레이어 파라미터 다름
  - ALBERT : 12개 레이어의 Attn 레이어 파라미터 공유
- [성능 향상을 위해] Next Sentence Prediction 목적 함수 → Sentence Order Prediction(
  - Masked LM은 두 모델 모두 사용
  - BERT : **Next Sentence Prediction**

    (쉬움) A 문장을 고르고 B 문장을 전체 문서에서 랜덤하게 하나 고름

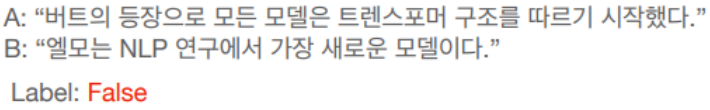
    A: “버트의 등장으로 모든 모델은 트렌스포머 구조를 따르기 시작했다.”
    B: “오늘 저녁은 탕수육을 먹을 것이다.”

    Label: False

  - ALBERT : **Sentence Order Prediction(어려움:)**

    랜덤하게 뽑는 거 의미 없음 주장 → SOP 하자.

(어려움) 두 문장의 위치를 바꾼 다음에 문장의 순서를 학습



이점

- 파라미터 공유로 인한 파라미터 개수 감소로 Memory 사용량 ↓
- 학습할 파라미터 개수가 줄어서 Train Time ↓

따라서 실제 서비스할 때 ALBERT가 많이 사용된다.

RoBERTa

A Robustly Optimized BERT Pretraining Approach

BERT 학습을 최적화 : 기존 BERT 덜 학습된 거 같다.

- 수단
  - 학습 시간 증가, 배치 사이즈 증가, 학습 데이터 10배 증가, 데이터 길이 증가
  - Next Sentence Prediction task 제거
    - 임베딩하는데 별로 도움 안 되는 듯해서 제거
  - Masking Pattern 변화 : Dynamic masking
    - 기존 문제: 문장 내 토큰 중 15%를 대치 (80%는 [MASK]로 대체, 10%는 그대로, 10%는 임의 토큰으로 대체) → 사전에 마스킹을 정의해놓기 때문에 매 epoch마다 문장의 패턴이 그대로 유지됨

- 이점
  - NLU에 강점 : 문장 유사도 측정에 강함.

Reference

- 2021 SOTA NLP 모델 설명 ⭐⭐⭐
- Illustrated GPT 2 ⭐⭐⭐
- ALBERT 설명 블로그 ⭐⭐⭐
- GPT ⭐⭐⭐
- GPT 2 ⭐
- GPT 3 ⭐
- ALBERT ⭐⭐⭐
- RoBERTa ⭐⭐⭐