



# NLP The Recent Trend

**Minho Ryu**

MetaFact CTO/ IDS Lab

강의목표: 이번 강의를 통해 딥러닝을 이용한 자연어 처리에 대한 최신 트렌드와 기술에 대해 이해하고 나아가 응용할 수 있도록 도움을 드리고자 합니다.

- 1교시 : Vector embedding of word (token)
- 2교시 : Various deep learning based embedding methods
- 3교시 : Deep learning algorithm to encode word and sentence
- 4교시 : Deep Contextual Encoding Models Before BERT
- 5교시 : BERT (Bidirectional Encoder Representation from Transformer)

# 자연어 처리 정의

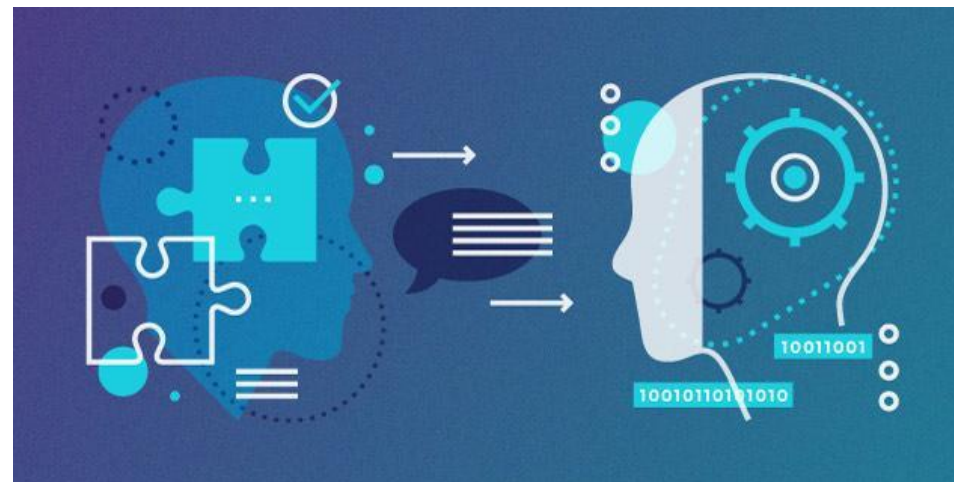
## 자연 언어 처리

인간의 언어 현상을 컴퓨터와 같은 기계를 이용해서 모사할 수 있도록 연구하고 이를 구현하는 인공지능의 주요 분야 중 하나

1. 자연 언어 이해: 컴퓨터가 입력으로 들어오는 자연 언어를 이해할 수 있도록 하는 것 (인코딩)
2. 자연 언어 생성: 컴퓨터가 자연 언어로써 출력을 내놓도록 하는 것 (디코딩)

## 자연어 처리 응용 분야 예시

- 정보검색
- QA 시스템
- 문서 자동 분류
- 신문기사 클러스터링
- 대화형 Agent 등



# 자연어 처리 기술 수준

## 기계 독해 분야: SQuAD 2.0과 1.1

Rank	Model	EM	F1
	Human Performance Stanford University (Rajpurkar & Jia et al. '18)	86.831	89.452
1 Jan 15, 2019	BERT + MMFT + ADA (ensemble) Microsoft Research Asia	85.082	87.615
2 Jan 10, 2019	BERT + Synthetic Self-Training (ensemble) Google AI Language <a href="https://github.com/google-research/bert">https://github.com/google-research/bert</a>	84.292	86.967
3 Dec 13, 2018	BERT finetune baseline (ensemble) Anonymous	83.536	86.096
4 Dec 16, 2018	Lunet + Verifier + BERT (ensemble) Layer 6 AI NLP Team	83.469	86.043
4 Dec 21, 2018	PAML+BERT (ensemble model) PINGAN GammaLab	83.457	86.122
5 Dec 15, 2018	Lunet + Verifier + BERT (single model) Layer 6 AI NLP Team	82.995	86.035
5 Jan 14, 2019	BERT + MMFT + ADA (single model) Microsoft Research Asia	83.040	85.892
6 Jan 10, 2019	BERT + Synthetic Self-Training (single model) Google AI Language <a href="https://github.com/google-research/bert">https://github.com/google-research/bert</a>	82.972	85.810
7 Dec 16, 2018	PAML+BERT (single model) PINGAN GammaLab	82.577	85.603
8 Nov 16, 2018	AoA + DA + BERT (ensemble) Joint Laboratory of HIT and iFLYTEK Research	82.374	85.310

Rank	Model	EM	F1
	Human Performance Stanford University (Rajpurkar et al. '16)	82.304	91.221
1 Oct 05, 2018	BERT (ensemble) Google AI Language <a href="https://arxiv.org/abs/1810.04805">https://arxiv.org/abs/1810.04805</a>	87.433	93.160
2 Oct 05, 2018	BERT (single model) Google AI Language <a href="https://arxiv.org/abs/1810.04805">https://arxiv.org/abs/1810.04805</a>	85.083	91.835
2 Sep 09, 2018	nlnet (ensemble) Microsoft Research Asia	85.356	91.202
2 Sep 26, 2018	nlnet (ensemble) Microsoft Research Asia	85.954	91.677
3 Jul 11, 2018	QANet (ensemble) Google Brain & CMU	84.454	90.490
4 Jul 08, 2018	r-net (ensemble) Microsoft Research Asia	84.003	90.147
5 Mar 19, 2018	QANet (ensemble) Google Brain & CMU	83.877	89.737
5 Sep 09, 2018	nlnet (single model) Microsoft Research Asia	83.468	90.133
5 Jun 20, 2018	MARS (ensemble) YUANFUDAO research NLP	83.982	89.796

## “구글, 사람처럼 전화 통화하는 인공지능 공개”

- 구글 인공지능과 실제 헤어샵, 레스토랑 직원 사이 대화를 녹음
- 실제 사람과 구분이 가지 않을 정도로 매끄러운 목소리로 대화
- 예약 전화를 받고 있는 헤어샵 직원은 상대방이 인공지능인 것을 전혀 눈치 채지 못함



<https://www.youtube.com/watch?v=pci3IMxFk4M>

# 1교시: Vector embedding of word (token)

- 원핫 인코딩 (one-hot encoding)
- TF-IDF 인코딩 (TF-IDF encoding)
- 분산표현 (distributed representation)

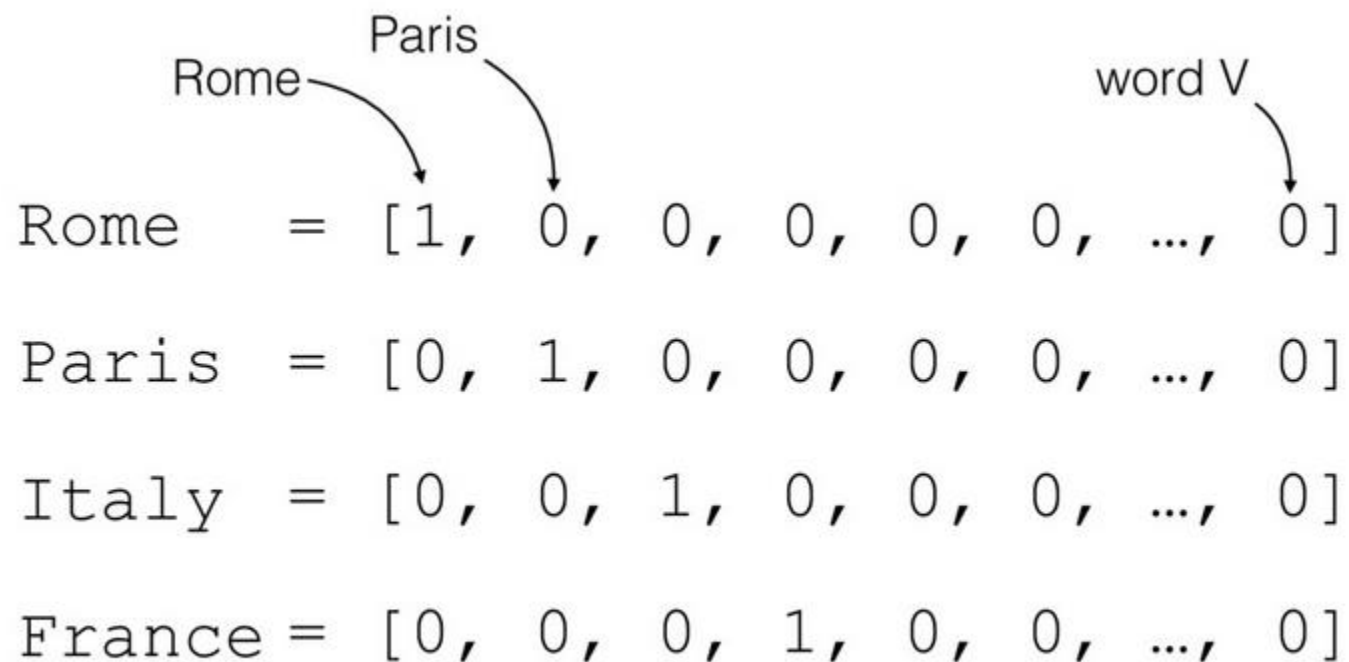
One-hot

TF-IDF

Word Embeddings

# 원핫 인코딩 (one-hot encoding)

- 원핫 인코딩: 단어를 범주형 변수로 변환한 것으로써 이진 벡터로 표현한 것
- 벡터의 각 차원이 단어 하나를 나타냄



# 원핫 인코딩의 bag of words

- 원핫 인코딩 을 이용하여 문장(문서)을 **bag of words** 형태로 나타낼 수 있다.
- 아래와 같이 각 단어의 출현 빈도를 사용할 수 있으며, 필요에 따라 출현 여부 만을 나타내도록 이진표현(0-1)으로 나타낼 수 있다.

doc\_1 = [32, 14, 1, 0, ..., 6]

doc\_2 = [ 2, 12, 0, 28, ..., 12]

...

doc\_N = [13, 0, 6, 2, ..., 0]



## 원핫 인코딩의 문제점

- 벡터 표현의 크기가 단어 코퍼스 크기에 따라 증가하므로 큰 단어 코퍼스에 대해 확장하기 어렵다.  
예를 들어, 코퍼스의 크기가 5000만 개일 경우, 각 단어의 벡터표현은 5000만 개의 float value가 필요로 된다.
- 각 단어의 벡터표현이 서로 같은 거리를 갖는다. 즉, 단어 간의 유사성 또는 차이와 같은 상관관계를 알 수 없다.
- 원핫 인코딩 bag of words의 경우 각 단어를 각 문장에만 의존하여 벡터를 구성하므로, 불용어에 대한 처리가 별도로 필요하게 된다.

# TF-IDF 인코딩 (TF-IDF encoding)

- 상대빈도분석 (Term Frequency Inverse Document Frequency)
- 문장의 벡터표현 시, 단어의 문장(문서) 내 빈도 및 전체 문장(문서) 내 빈도를 고려하여 단어를 표현

1. Term Frequency: 해당 문서 내 Token 발생 빈도

2. (문서 내 Token 출현 빈도) / (문서 전체 Token 개수)

3. Inverse Document Frequency: 문서 빈도의 역

4.  $\log(\text{전체 문서 수} / \text{Token 포함 문서 수})$

1. TF는 해당 문서만 있으면 바로 연산이 가능하지만

2. IDF는 모집단의 Token별 통계 데이터가 필요함

$$w_{x,y} = \text{tf}_{x,y} \times \log\left(\frac{N}{\text{df}_x}\right)$$

**TF-IDF**

Term  $x$  within document  $y$

$\text{tf}_{x,y}$  = frequency of  $x$  in  $y$

$\text{df}_x$  = number of documents containing  $x$

$N$  = total number of documents

# TF-IDF 인코딩 (TF-IDF encoding)

- 예를 들어 100개의 단어로 이루어진 문서에 단어 '고양이'가 12번 등장 했다면,
- $TF(\text{고양이}) = 12/100 = 0.12$
- 또한 전체 문서 개수 1,000,000개 중 50,000개의 문서에서 단어 '고양이'가 등장 했다면,
- $IDF(\text{고양이}) = \log(1000000/50000) = 1.30$
- 따라서 단어 '고양이'의 TF-IDF 값은
- $TF-IDF(\text{고양이}) = TF(\text{고양이}) * IDF(\text{고양이}) = 0.12 * 1.30 = 0.156$

$$w_{x,y} = tf_{x,y} \times \log \left( \frac{N}{df_x} \right)$$

**TF-IDF**

Term  $x$  within document  $y$

$tf_{x,y}$  = frequency of  $x$  in  $y$

$df_x$  = number of documents containing  $x$

$N$  = total number of documents


## TF-IDF 인코딩의 문제점

- IDF term으로 인해 불용어 처리에 대한 전처리를 피할 수 있지만 원핫 인코딩과 마찬가지로 벡터 표현의 크기가 단어 코퍼스 크기에 따라 증가하므로 큰 단어 코퍼스에 대해 확장하기 어렵다.
- TF-IDF 인코딩을 이용한 bag of words 표현을 이용하면 문장 간의 유사도를 구하기에는 용이할 수 있지만 단어 간의 상관관계를 구할 수 없다.
- 하지만 원핫 인코딩과 TF-IDF 인코딩은 사용하기 쉽고 단순한 자연어 문제에 대해서는 상당히 효과적인 성과를 보여주어 여전히 많이 사용되고 있다.

# 분산 표현 (distributed representation)

- 분산 표현: 단어를 다차원의 밀집된 실수 벡터로 나타내는 표현 방법
- 유사한 단어는 유사한 벡터로 투영할 수 있고 단순한 벡터 연산으로 단어 간의 상관관계를 표현할 수 있다.
- 단어의 벡터 표현의 차원이 코퍼스 크기보다 훨씬 작다.
- 분산 표현을 배우기 위한 대표적인 알고리즘으로 word2vec, FastText, GloVe 등이 있다.

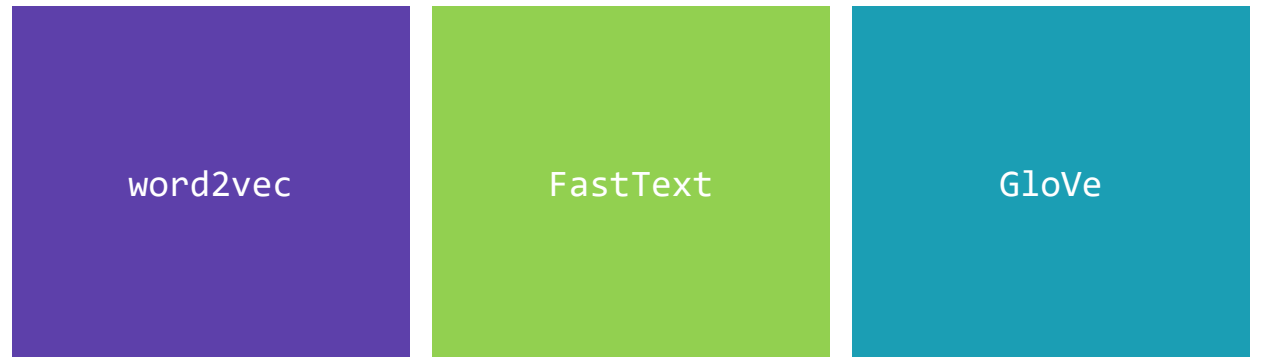
**n. dimensions << vocabulary size**



Rome = [0.91, 0.83, 0.17, ..., 0.41]  
Paris = [0.92, 0.82, 0.17, ..., 0.98]  
Italy = [0.32, 0.77, 0.67, ..., 0.42]  
France = [0.33, 0.78, 0.66, ..., 0.97]

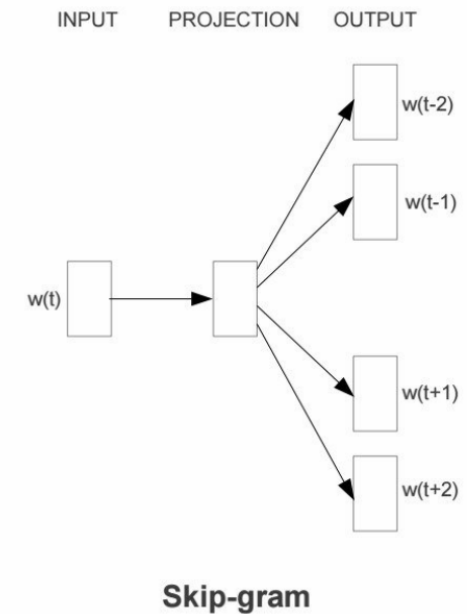
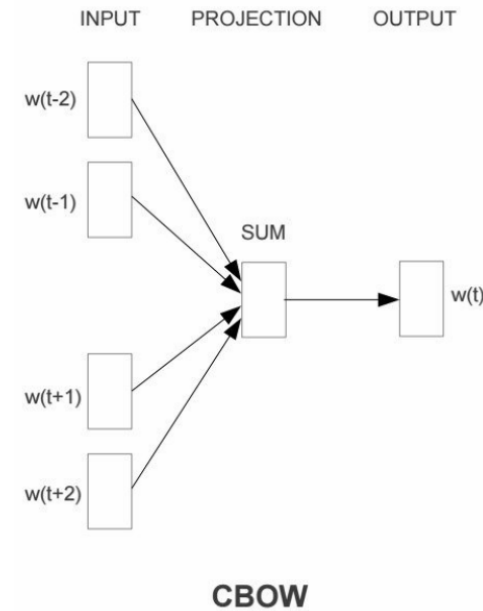
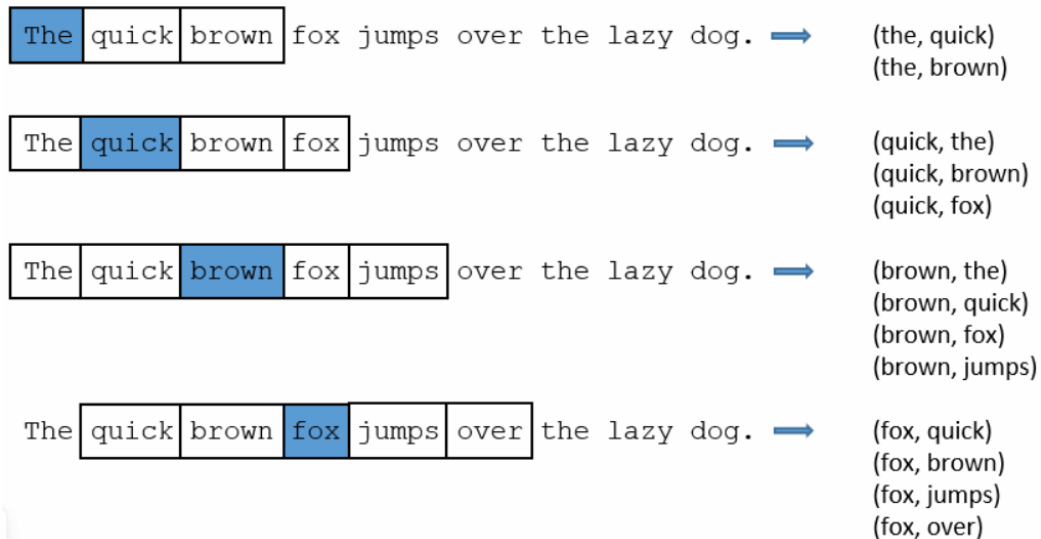
## 2교시: Various deep learning based embedding methods

- word2vec
- FastText
- GloVe



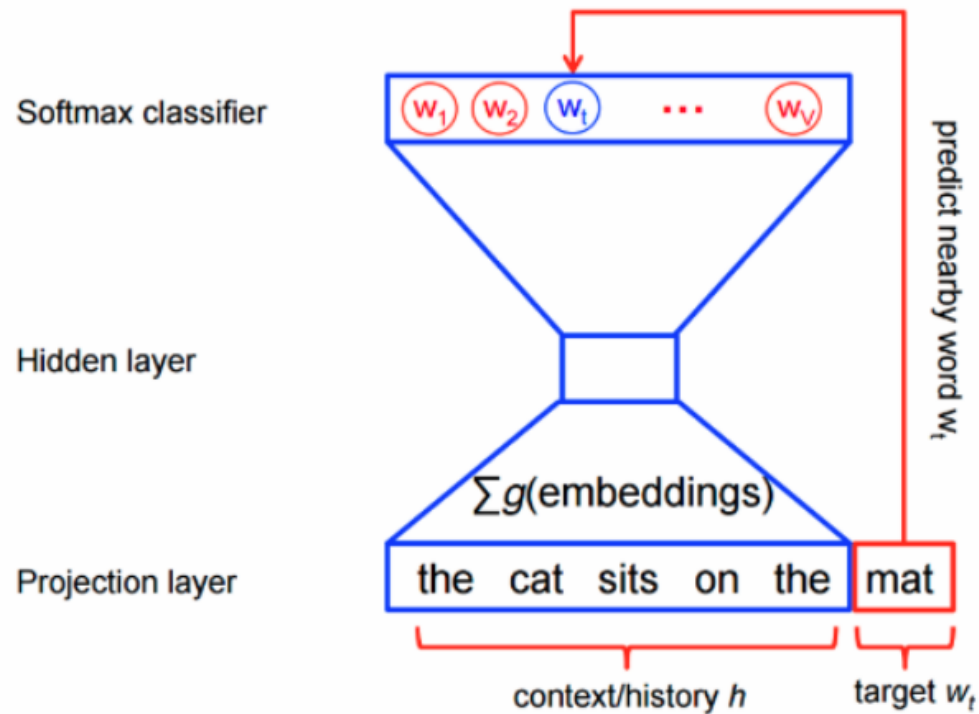
# word2vec

- word2vec 알고리즘은 cbow 또는 skip-gram 모델을 이용하여 아이디어는 간단하다.
- cbow 모델은 타겟의 주변 단어들의 워드 임베딩 벡터의 합을 이용하여 타겟단어를 예측하는 것이며,
- skip-gram 모델은 타겟의 워드 임베딩을 이용하여 주변 단어들을 예측하는 것이다.



# word2vec's brilliant contributions

- word2vec 알고리즘의 기발함은 모델의 간결함도 있지만 비싼 softmax 계산을 negative sampling이라는 기법을 제시하여 계산 비용을 획기적으로 줄인 것에 있다.
- negative sampling 은 실제로 예측해야 하는 단어를 제외한 단어들을 랜덤으로 선택하여 softmax cross entropy loss 를 사용하는 대신 binary logistic loss 를 이용한다.





- FastText 는 word2vec의 skip-gram 모델과 동일한 방법을 사용한다.
- 한 가지 차이점은 FastText 에서는 rare word 를 더 잘 처리하기 위해 단어의 벡터를 그 단어의 subword 의 합으로 나타낸다는 점이다. 예를 들어, subword의 길이를 3으로 정했을 때, 단어 where의 경우 다음과 같이 subword로 쪼갤 수 있으며 <wh, her, ere, re> 각각의 subword의 임베딩 벡터의 합으로 단어 where의 임베딩 벡터를 구할 수 있다.
- 그래서 논문 제목도 “Enriching Word Vectors with Subword Information”

The logo for FastText, with the word 'fast' in a red, italicized sans-serif font and the word 'Text' in a blue, bold sans-serif font.

- word2vec 알고리즘의 경우 단어 간의 동시 등장 확률은 고려하지 않는다.
- GloVe 에서는 co-occurrence matrix를 이용하여 동시 등장확률을 예측하도록 개선하였다.
- 본 논문에서는 word2vec보다 상당히 좋아졌다는 결과를 리포트 하였으나 다른 후속 연구 결과에서 hyper-parameter 를 잘 조정하면 word2vec 과 별로 차이가 나지 않는다고 한다.
- word2vec보다 데이터 병렬처리 하기가 간편하여 더 쉽게 빨리 학습할 수 있다.

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

## 3교시: Deep learning algorithm to encode word or sentence

- Feed Forward Neural Network (FFNN)
- Recurrent Neural Network (RNN)
- Long Short Term Memory (LSTM)
- Convolutional Neural Network (CNN)

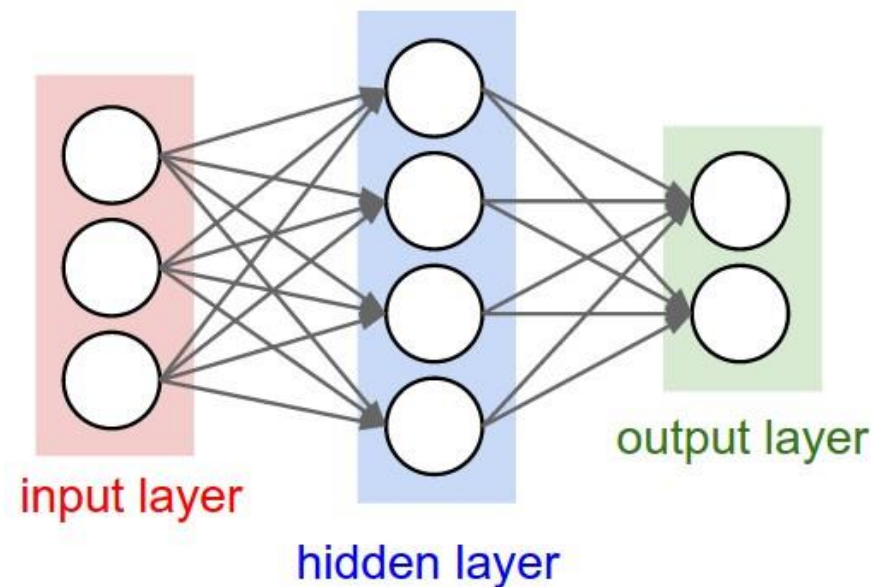


# Feed Forward Neural Network (FFNN)

- FFNN은 자연어처리 문제에서 one-hot 또는 tf-idf 인코딩을 이용한 bag of words 표현을 입력으로 사용하거나 FastText classification 처럼 averaged embeddings of words 를 입력으로 사용

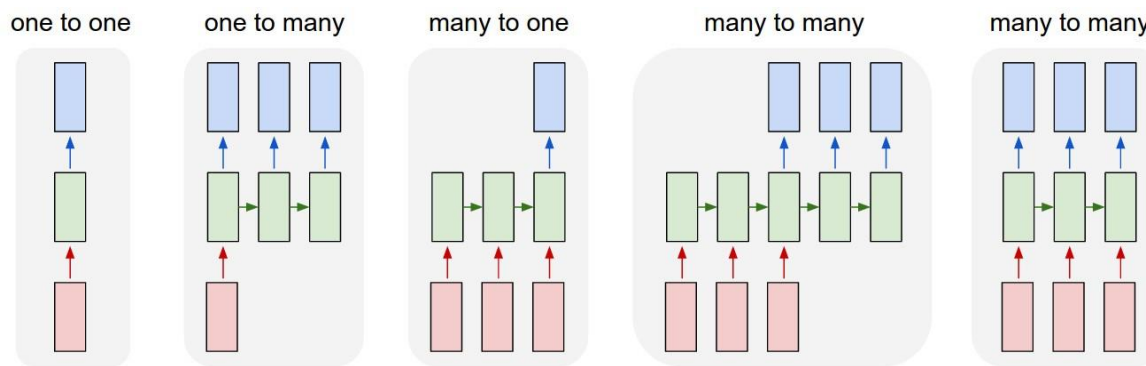
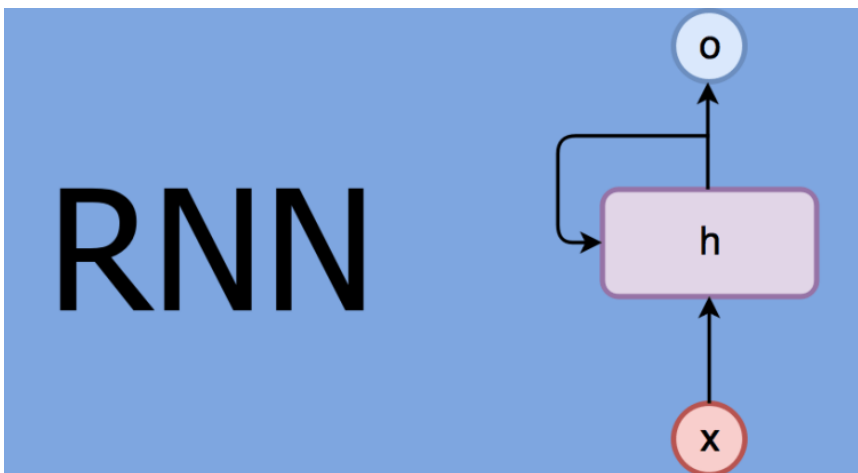
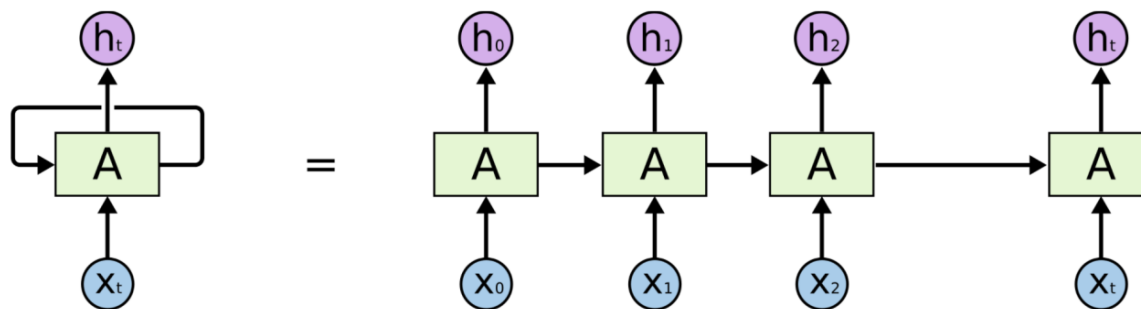
```
with tf.variable_scope("placeholder"):
    self.input_x = tf.placeholder(tf.float32, shape=(None, self.vocab_size))
    self.input_y = tf.placeholder(tf.int32, shape=(None,))

with tf.variable_scope("output", reuse=tf.AUTO_REUSE):
    W1 = tf.get_variable("W1", dtype=tf.float32,
                        initializer=tf.truncated_normal(
                            (self.vocab_size, self.hidden_size)))
    b1 = tf.get_variable("b1", dtype=tf.float32,
                        initializer=tf.constant(0.1,
                            shape=(self.hidden_size,)))
    W2 = tf.get_variable("W2", dtype=tf.float32,
                        initializer=tf.truncated_normal(
                            (self.hidden_size, self.n_class)))
    b2 = tf.get_variable("b2", dtype=tf.float32,
                        initializer=tf.constant(0.1, shape=(self.n_class,)))
    h = tf.nn.relu(tf.nn.xw_plus_b(self.input_x, W1, b1))
    logits = tf.nn.xw_plus_b(h, W2, b2)
    self.prob = tf.reduce_max(tf.nn.softmax(logits), axis=1)
    self.prediction = tf.cast(tf.argmax(logits, axis=1), tf.int32)
```



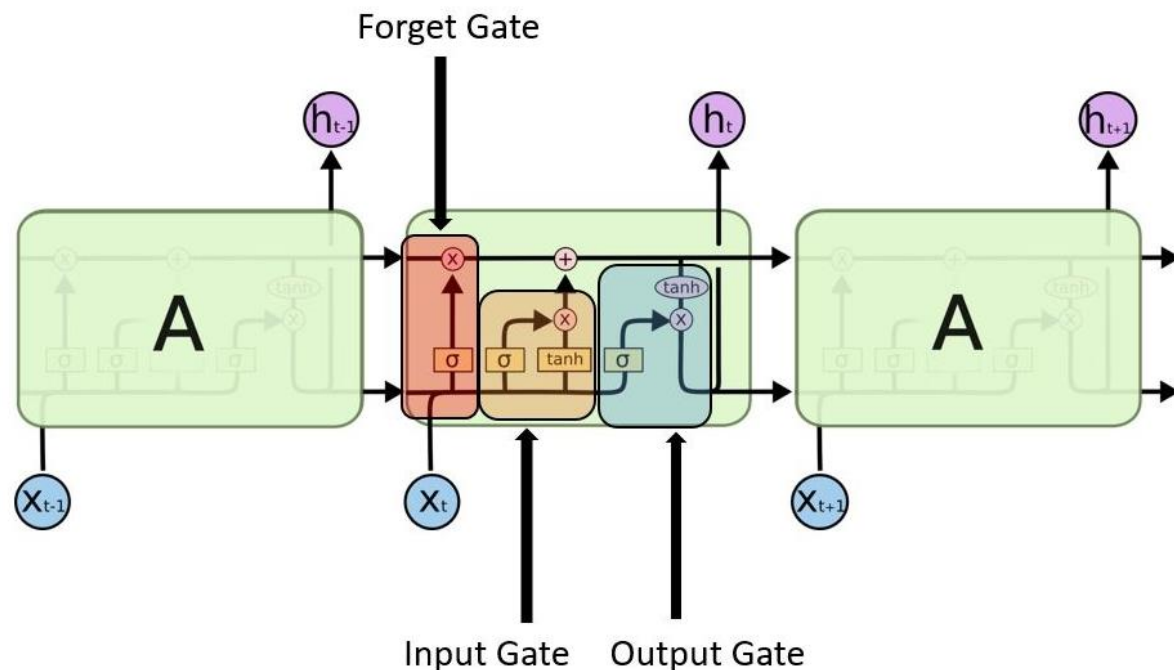
# Recurrent Neural Network (RNN)

- RNN 은 입력을 순차적으로 받으며, 받은 입력들의 정보를 저장하는 Hidden State 를 사용한다.
- 기본 RNN 의 경우 입력의 길이가 조금만 길어질 경우, 학습이 잘 되지 않는다.
- 이를 해결하기 위해서 나온 것이 LSTM



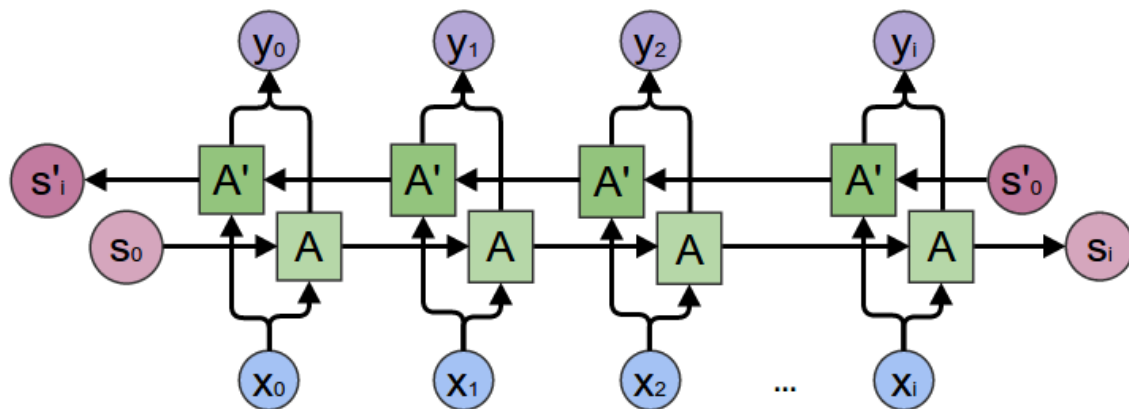
# Long Short Term Memory (LSTM)

- LSTM 은 ‘게이트’ 구조를 활용하여 삭제할 정보와 추가할 정보를 결정한다.
- Forget gate: 새로 들어온 정보(입력)와 hidden state 를 이용하여 이전 정보에서 어떤 정보를 얼마나 버릴 지 결정한다.
- Input gate: 새로 들어온 정보(입력)와 hidden state 를 이용하여 새로운 정보에서 어떤 정보를 얼마나 추가할 지 결정한다.
- Output gate: 새로 들어온 정보(입력)와 hidden state 를 이용하여 결정된 새로운 정보에서 출력할 정보를 결정한다.



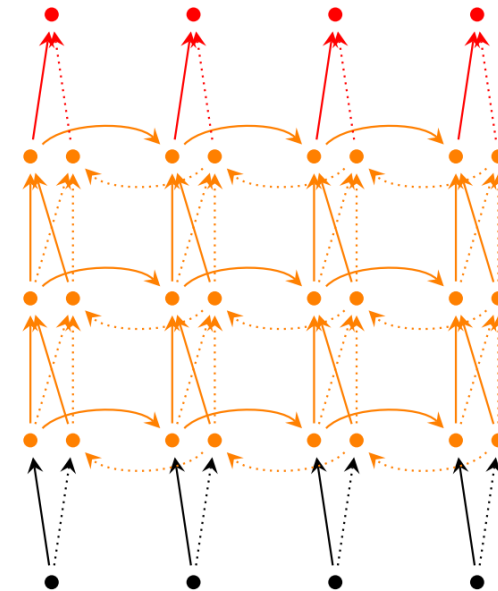
## Bidirectional RNN

- 입력이 길어질수록 앞 부분의 입력 정보가 뒷부분의 입력정보에 의해 희석이 되는 문제가 발생하는데 이를 보완하고자 입력의 순서를 역순으로도 넣어 정보를 얻어내는 방법



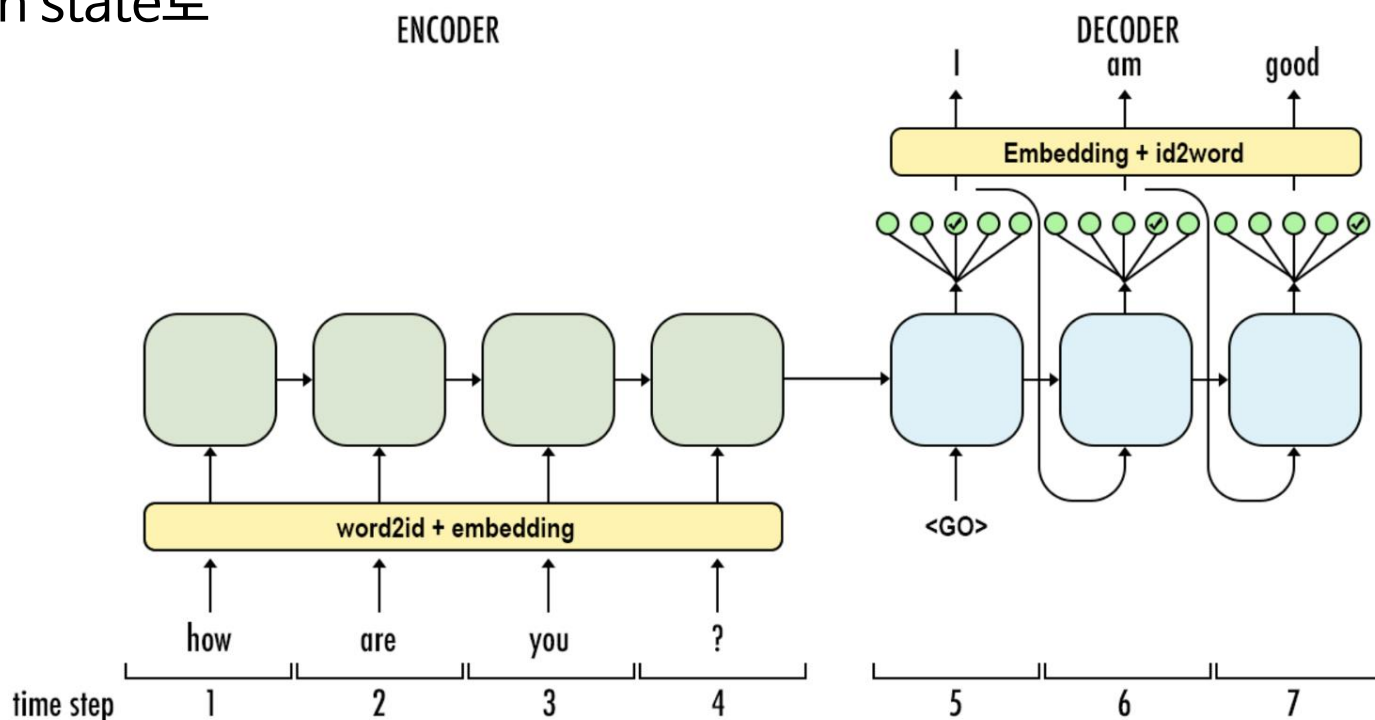
## Deep Bidirectional RNN

- 복잡한 문제일수록 더 많은 층을 쌓는 것이 도움이 된다는 경험적 교훈을 RNN 구조에 적용한 방법



# Sequence to Sequence (Seq2Seq)

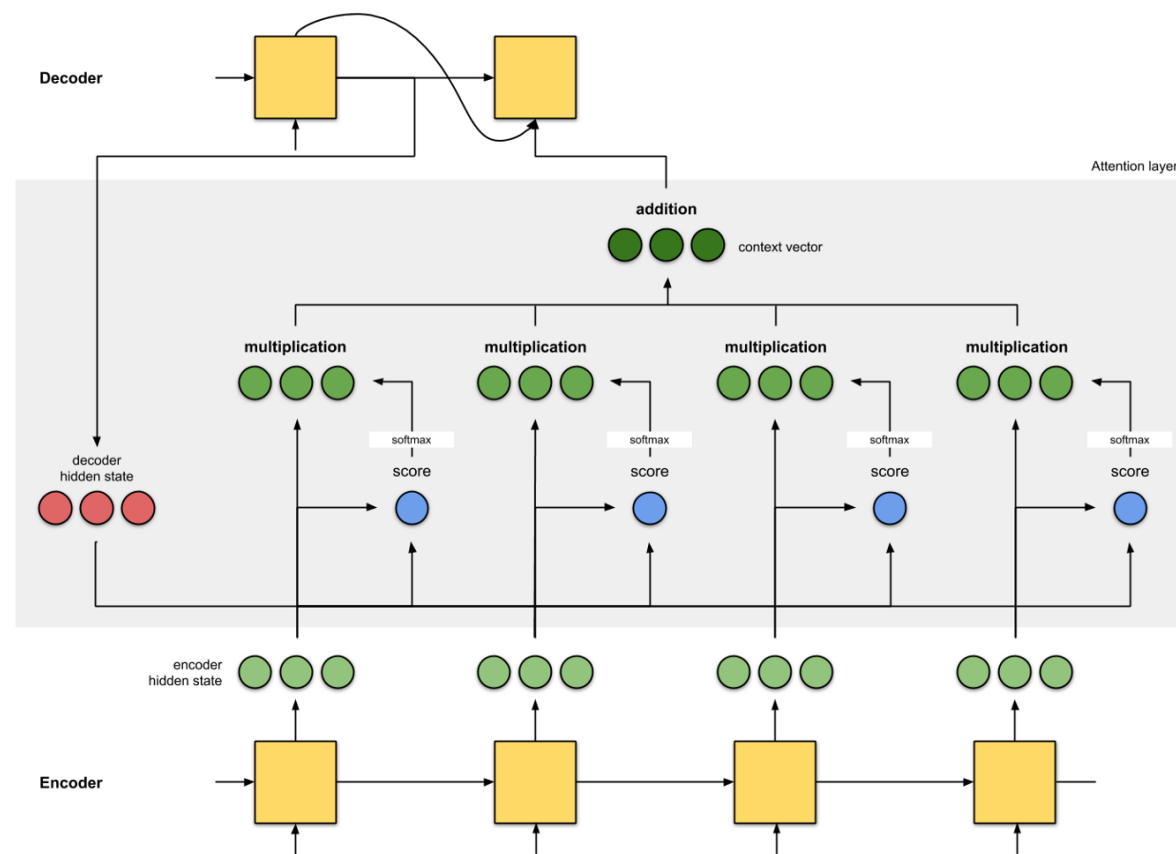
- Seq2Seq 모델(인코더-디코더 모델이라고도 함)은 두 개의 RNN 모듈을 이용한다.
- 첫 번째 RNN은 문장을 하나의 벡터로 변환(인코딩)하는 역할을 하고,
- 두 번째 RNN은 인코딩된 벡터를 첫 hidden state로 이용하여 디코딩을 진행한다.
- 이 구조를 이용하여 문장 번역, 문서 요약, 챗봇과 같이 문장을 받아 문장을 출력하는 작업을 수행할 수 있다.
- 문제점은?





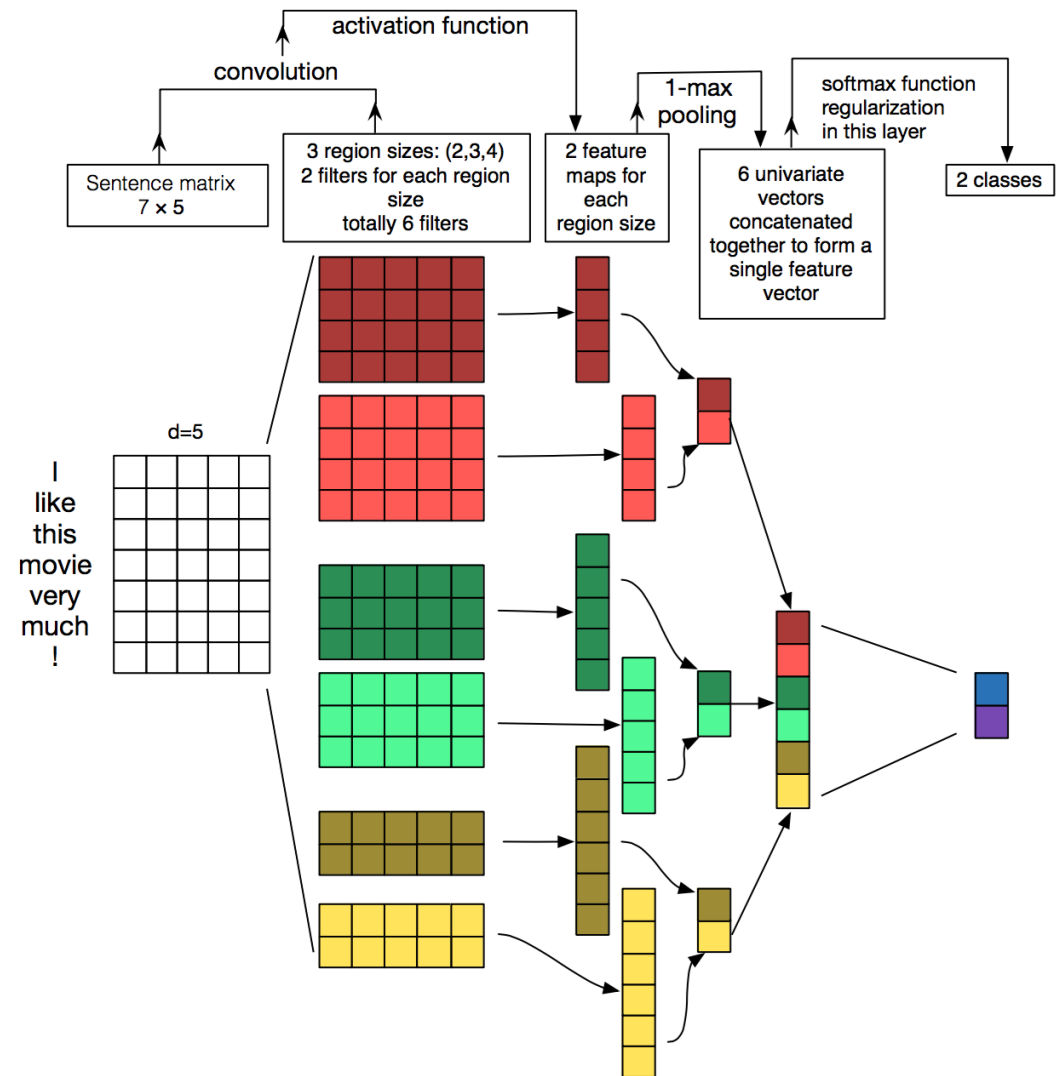
# Seq2Seq with attention

- Seq2Seq 모델은 입력이 길어질 경우, 모든 정보를 하나의 벡터에 담을 수 없다.
- 디코딩을 진행할 때, 인코더의 마지막 hidden state 만을 이용하기보다 각각의 인코딩된 단어의 hidden state를 이용하면 더 많은 정보를 활용할 수 있다.
- 각 인코딩된 단어들 중 어디에 집중할 것인지 결정하는 것을 attention mechanism 이라 한다.



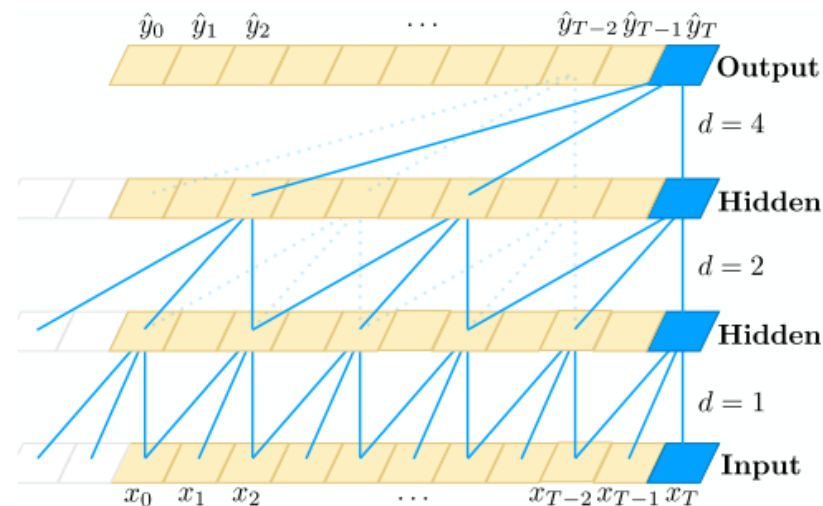
# Convolutional Neural Network (CNN)

- CNN은 n-gram을 한 번에 고려할 수 있다.  
(1d conv with filter size  $n$ )
- RNN 계열의 모델의 경우 입력 데이터가 모델에 순차적으로 들어가므로 병렬처리를 할 수 없는 단점이 있는 반면, CNN은 데이터 병렬처리에 유리하므로 훈련속도 및 실행속도를 큰 폭으로 개선할 수 있다.

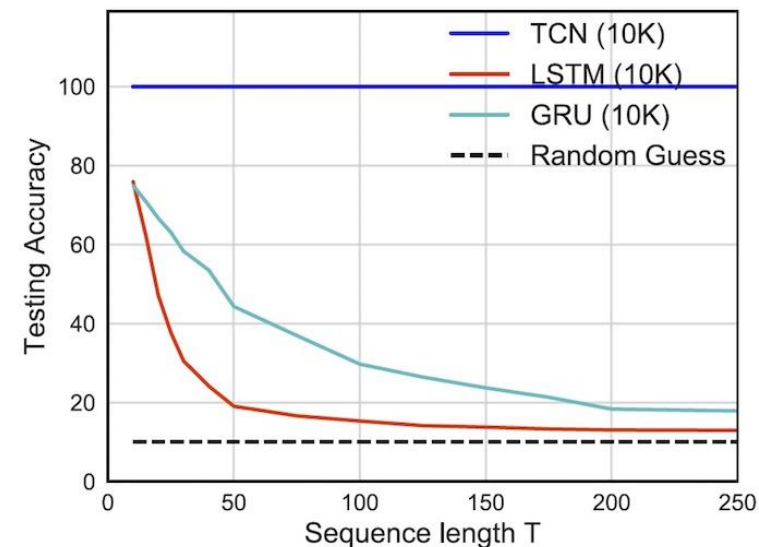


# Temporal Convolutional Network (TCN)

- TCN 은 CNN으로 순차 모델을 학습할 수 있도록 causal + dilated convolution 을 이용한 모델이다.
- Causal convolution 은  $t$  시점의 인코딩을 구할 때  $t$  시점을 포함한 이전 시점 입력만을 이용하는 convolution 방법이고,
- Dilated convolution 은 CNN 계층구조의 위로 올라갈수록 더 넓은 receptive field를 갖도록 도와준다.
- 입력이 길어질 때, 정보를 유지하는 데에 RNN계열보다 유리하다.

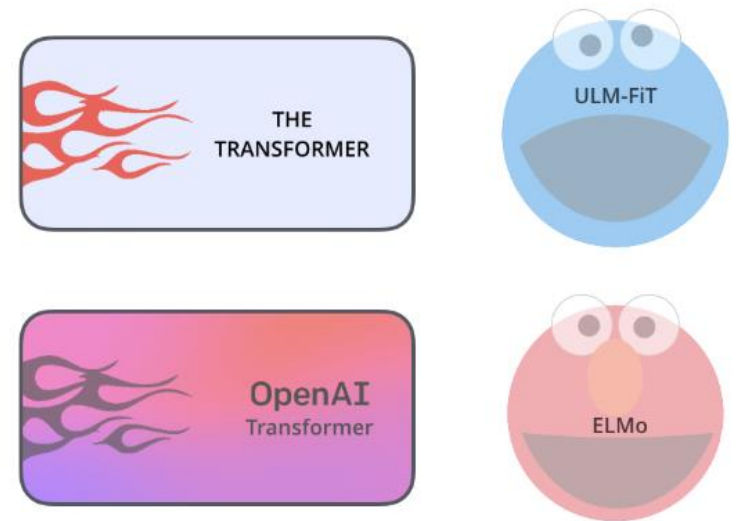


Causal + dilated convolution



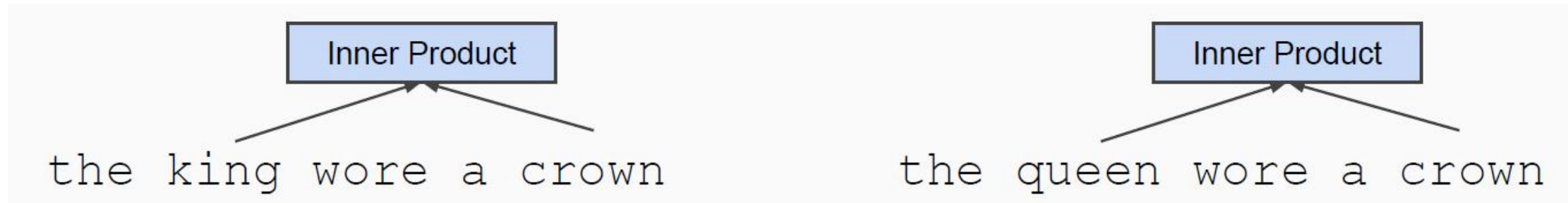
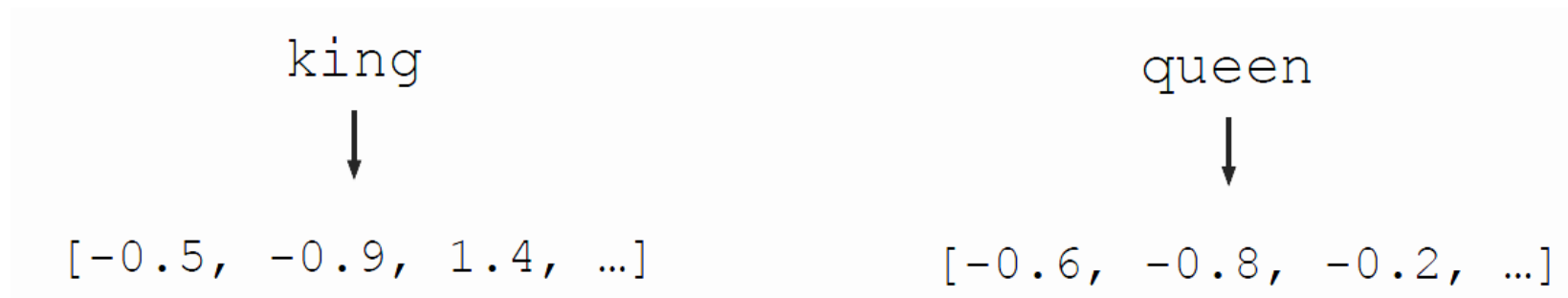
# 4교시: Deep Contextual Encoding Models Before BERT

- Contextual Representation
- History of Contextual Representation
- Problem with Previous Methods



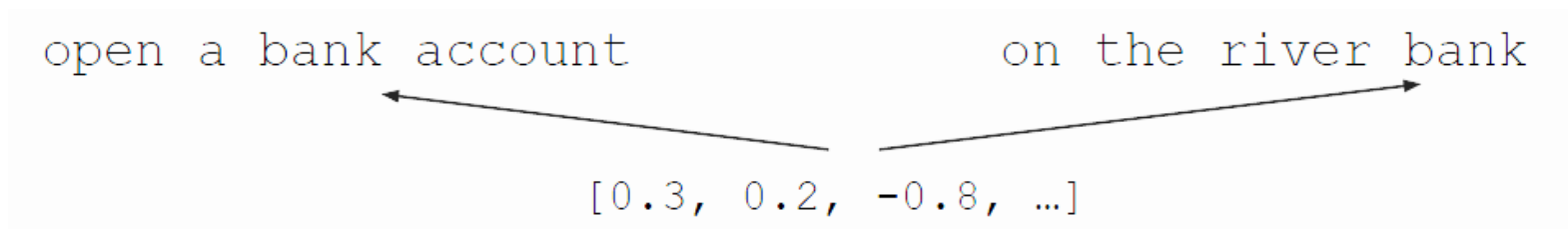
# Pre-Training in NLP

- 워드 임베딩 기법은 딥러닝을 이용한 자연어 처리 기술의 기초
- 주로 word2vec 또는 GloVe 같은 알고리즘을 이용하여 엄청 큰 코퍼스에 사전 훈련되어 사용
- 얇은 학습 (Shallow Learning)

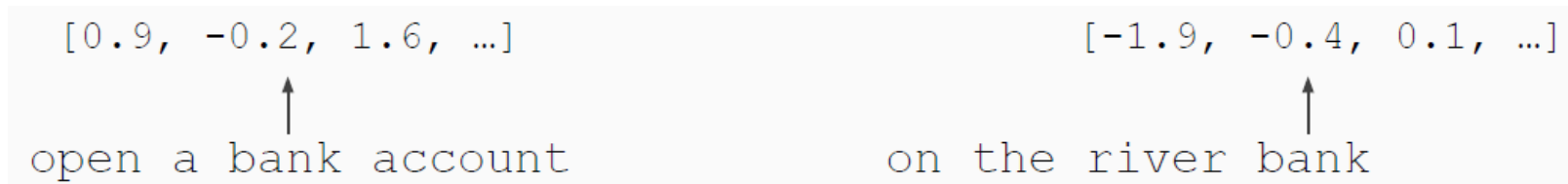


# Contextual Representation

- 워드 임베딩의 경우 context와 무관하게 적용된다는 문제점이 있다.

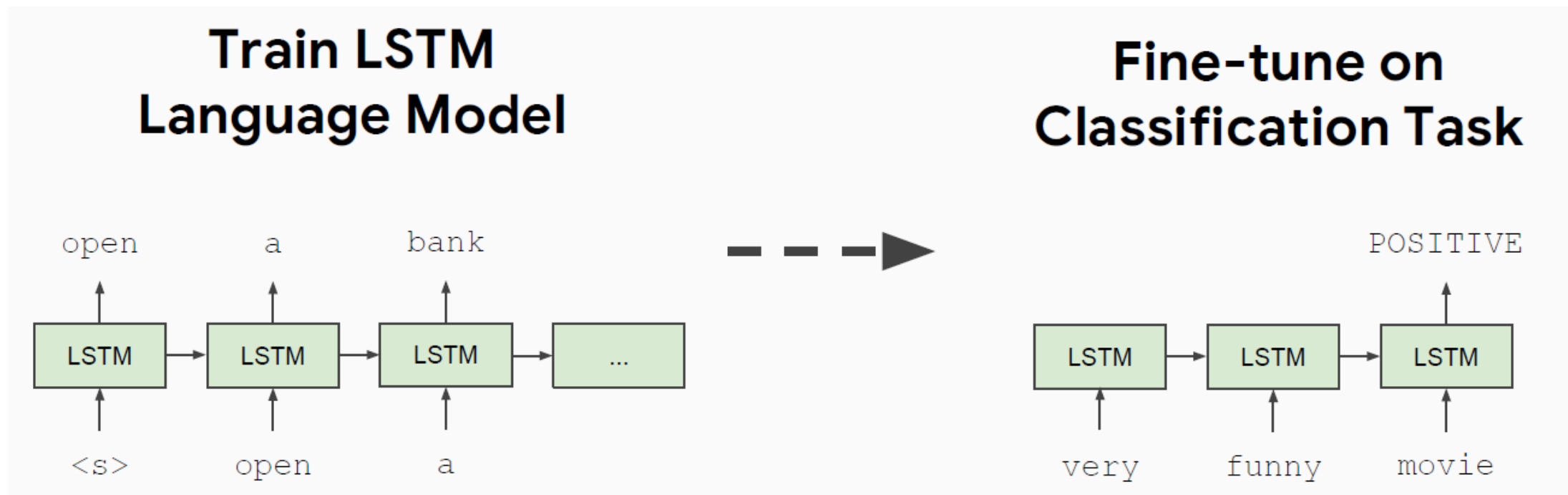


- 텍스트 코퍼스에 대해 contextual representation 을 학습함으로써 위 문제를 해결할 수 있다.
- 깊은 학습 (Deep Contextualized Word Representations Learning)



# History of Contextual Representation

- Semi-Supervised Sequence Learning, Google, 2015

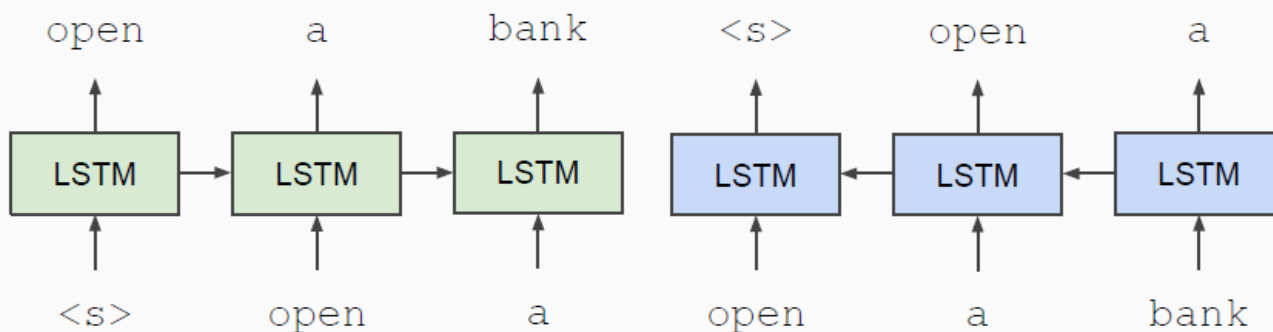


- LSTM으로 언어모델링을 학습한 뒤 Task 데이터 셋에 Fine-Tuning

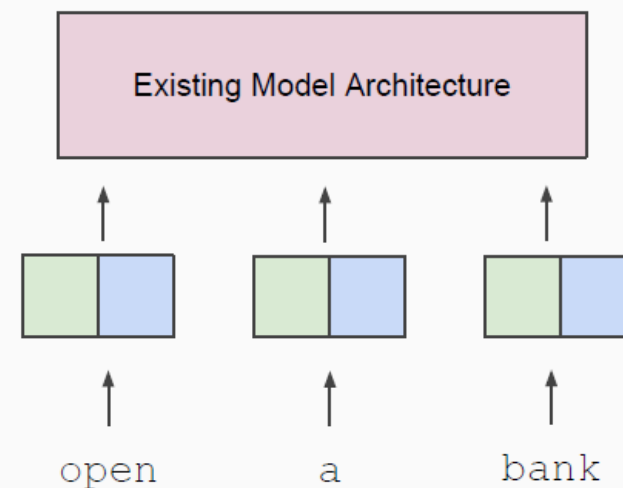
# History of Contextual Representation

- ELMo: Deep Contextual Word Embeddings, AI2 & University of Washington, 2017

## Train Separate Left-to-Right and Right-to-Left LMs



## Apply as “Pre-trained Embeddings”

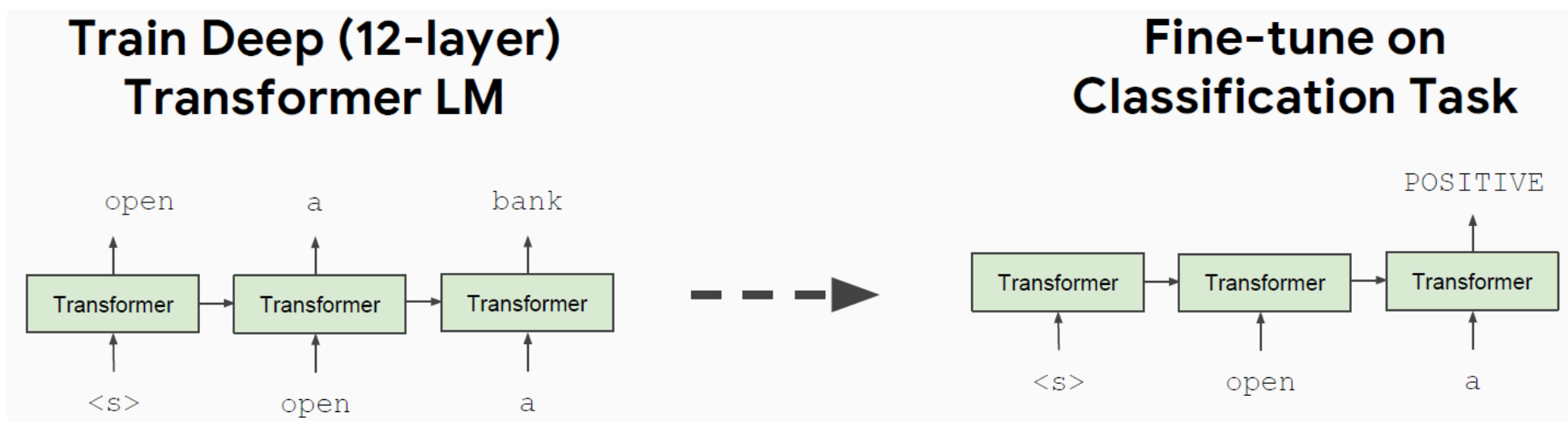


- Deep Bidirectional LSTM 을 이용하여 언어모델링을 학습한 뒤, 양방향 hidden state를 concat하여 Contextual Pre-trained Word Embeddings 로 사용



# History of Contextual Representation

- Improving Language Understanding by Generative Pre-Training, OpenAI, 2018  
(the model is simply called OpenAI GPT)



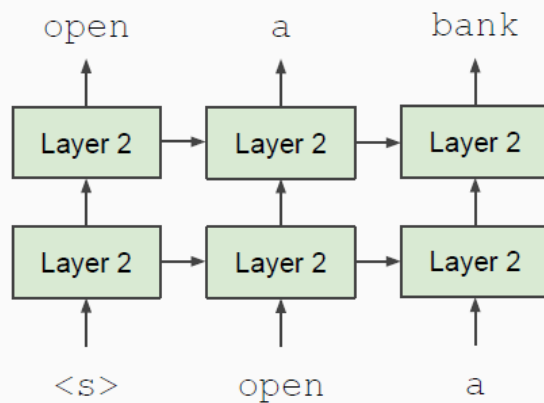
- Transformer Decoder 를 깊게 쌓은 모델로 언어모델링을 학습한 뒤 Task 데이터 셋에 Fine-Tuning

# Problem with Previous Methods

- 이전 모델들의 문제점은 언어의 이해는 양방향으로 이루어져야 하는데 언어모델을 학습할 때 왼쪽 context 또는 오른쪽 context 만을 이용하는 것이다.
- 그러나 양방향 인코더의 경우, 예측해야 하는 단어를 모델이 예측 전에 볼 수 있다는 문제가 있다.

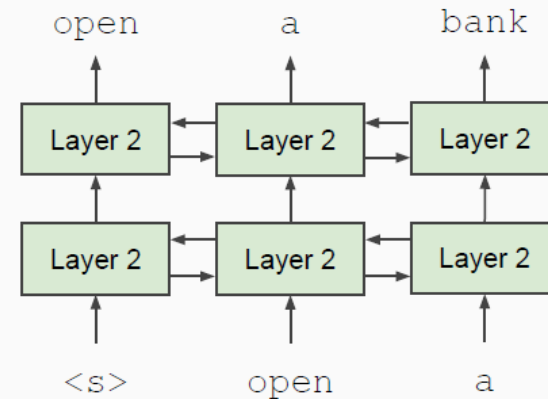
## Unidirectional context

Build representation incrementally



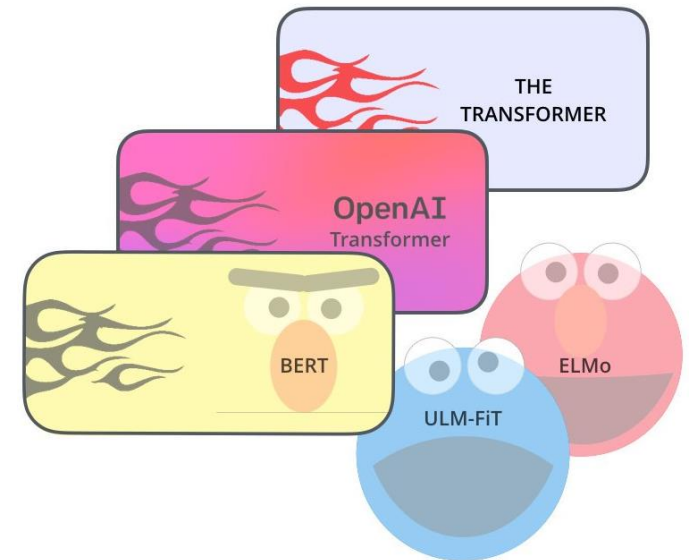
## Bidirectional context

Words can “see themselves”



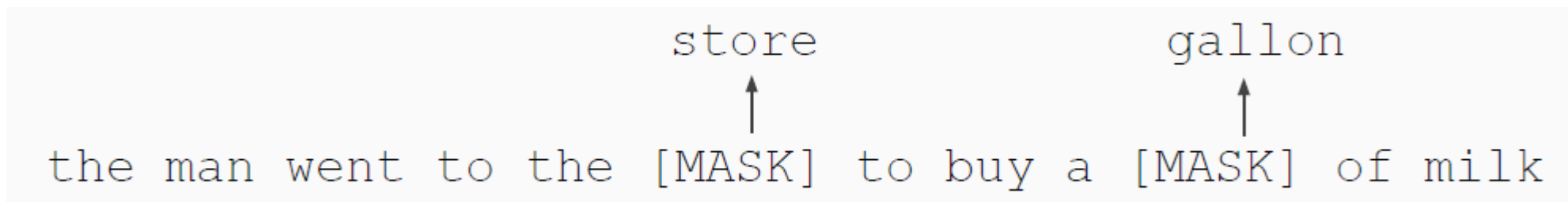
# 5교시: BERT (Bidirectional Encoder Representation from Transformer)

- Pretraining Task
- Input Representation
- Model Architecture
- Fine-Tuning Procedure



# Pretraining Task – Masked LM

- BERT 는 양방향 인코더를 사용하면서 see themselves 문제를 해결하기 위해 [MASK] 토큰을 도입한다.
- 전체 입력 단어들 중 15%를 [MASK] 토큰으로 대체하고 가려진 단어들을 예측하는 Task를 학습한다.



store                      gallon

↑                                      ↑

the man went to the [MASK] to buy a [MASK] of milk

- 단어를 너무 많이 [MASK] 토큰으로 대체할 경우 훈련하는 비용이 너무 비싸고,
- 너무 적게 대체할 경우 배울 수 있는 context 가 충분하지 않을 수 있다.

# Pretraining Task – Masked LM

- Masked LM 훈련의 문제점은 [MASK] 토큰이 fine-tuning 시에 등장하지 않는다는 것
- 이 문제에 대한 해결책으로 전체 예측할 15%의 단어들 중, 전부 [MASK] 토큰으로 대체하는 것이 아니라

- 80% 는 원래대로 [MASK] 토큰으로 대체하고,

went to the store → went to the [MASK]

- 10% 는 단어를 무작위로 선택된 단어로 대체하며,

went to the store → went to the running

- 10% 는 대체하지 않고 원래 단어를 사용한다.

went to the store → went to the store

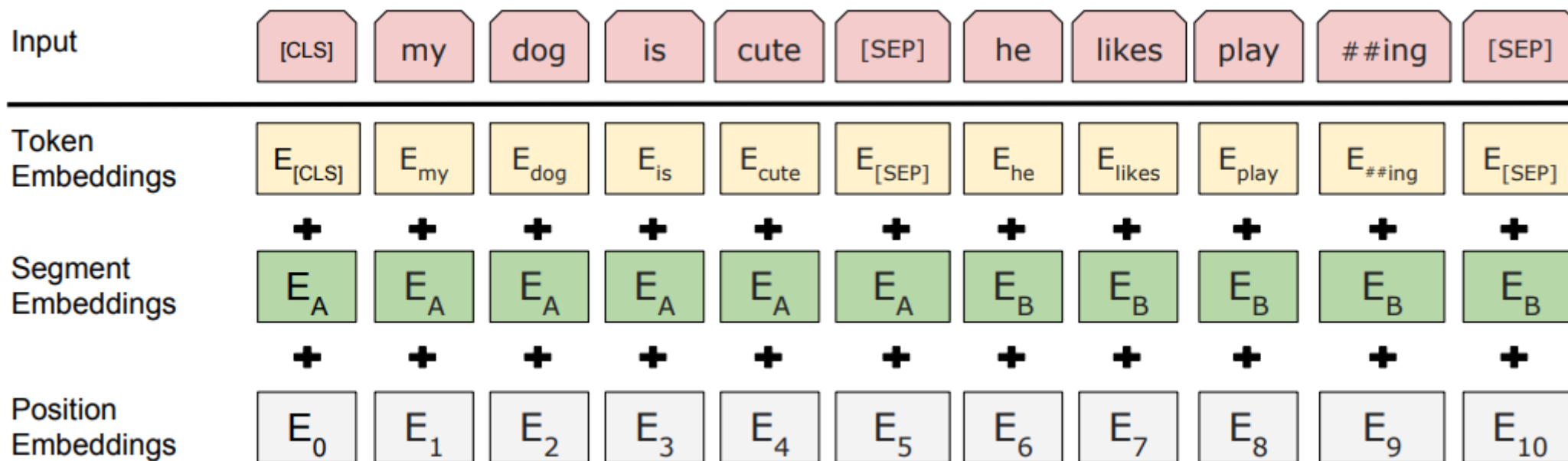
# Pretraining Task – Next Sentence Prediction

- BERT 는 Masked LM 과 더불어 문장 간의 관계를 배우기 위해서 두 문장이 주어졌을 때,
- 뒷문장이 실제로 앞문장 뒤에 오는 문장인지 아니면 무작위로 뽑힌 문장인지 구별하는 Task를 학습한다.
- 데이터 셋 구성 시 50%는 실제 이어진 문장을 사용하고 50%는 무작위로 선택된 문장을 사용한다.

**Sentence A** = The man went to the store.  
**Sentence B** = He bought a gallon of milk.  
**Label** = IsNextSentence

**Sentence A** = The man went to the store.  
**Sentence B** = Penguins are flightless.  
**Label** = NotNextSentence

# Input Representation



- 각 토큰은 Token, Segment, Position 임베딩의 합으로 이루어지며 각 임베딩은 학습된다.
- 모든 입력의 첫 토큰은 [CLS] 토큰을 사용하고, 문장 두 개를 구분하기 위해 [SEP] 토큰을 사용한다.
- 30,000개의 WordPiece vocabulary 를 사용

# WordPiece Model

- WordPiece Model 은 Byte Pair Encoding (BPE) 이라는 방법을 응용한다.
  - 단어를 문자(또는 음절) 단위로 쪼갠 뒤 빈도 수에 따라 문자를 병합하여 subword를 구성한다. 병합을 통해 생성된 subword 들은 모두 단어장에 포함된다.
  - 이 방법을 이용하면 각 언어에 대한 언어학적 지식 없이도 각 단어들을 효과적인 의미 단위로 쪼갤 수 있는 subword 를 텍스트 코퍼스로부터 학습할 수 있다.
- 
- **Word** : Jet makers feud over seat width with big orders at stake
  - **Wordpieces**: \_J et \_makers \_fe ud \_over \_seat \_width \_with \_big \_orders \_at \_stake



# WordPiece Tokenization

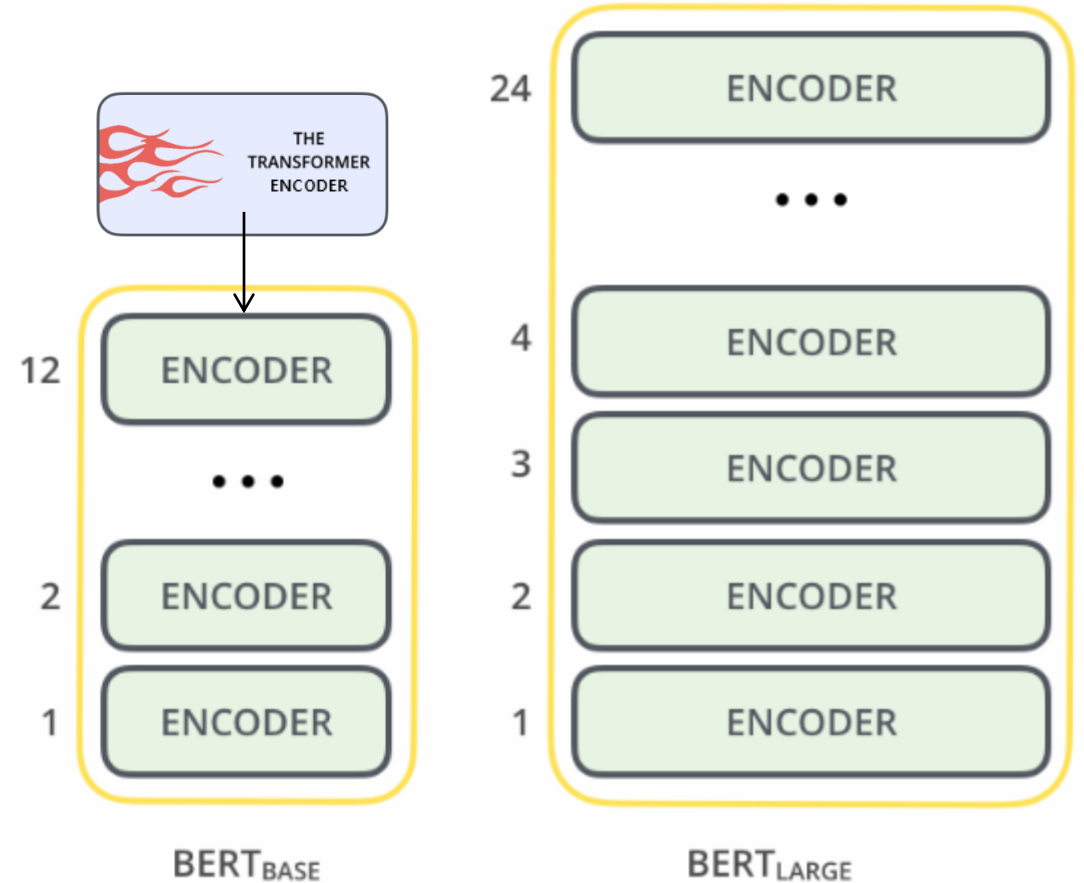
- WordPiece Model 을 통해 구성된 단어장(vocabulary)이 주어졌을 때, 이 단어장을 기준으로 각 단어를 문자 단위로 쪼개 뒤 단어장에 존재하는 가장 긴 subword로 tokenize 한다.
- 예를 들어 단어장이 {'play', '##ing', 'running', 'kor', '##ea', '##e', '##a'} 로 이루어졌다고 할 때, “playing running korea” 라는 문장은 ['play', '##ing', 'running', 'kor', '##ea'] 로 토큰화 된다.
- BERT 에서는 원래 주어진 단어가 subword로 쪼개질 경우, 위 예시와 같이 쪼개지는 뒷부분 앞에 '##'을 붙인다.

Jim Henson was a puppeteer

→ ['Jim' 'Hen' '##son' 'was' 'a' 'puppet' '##eer']

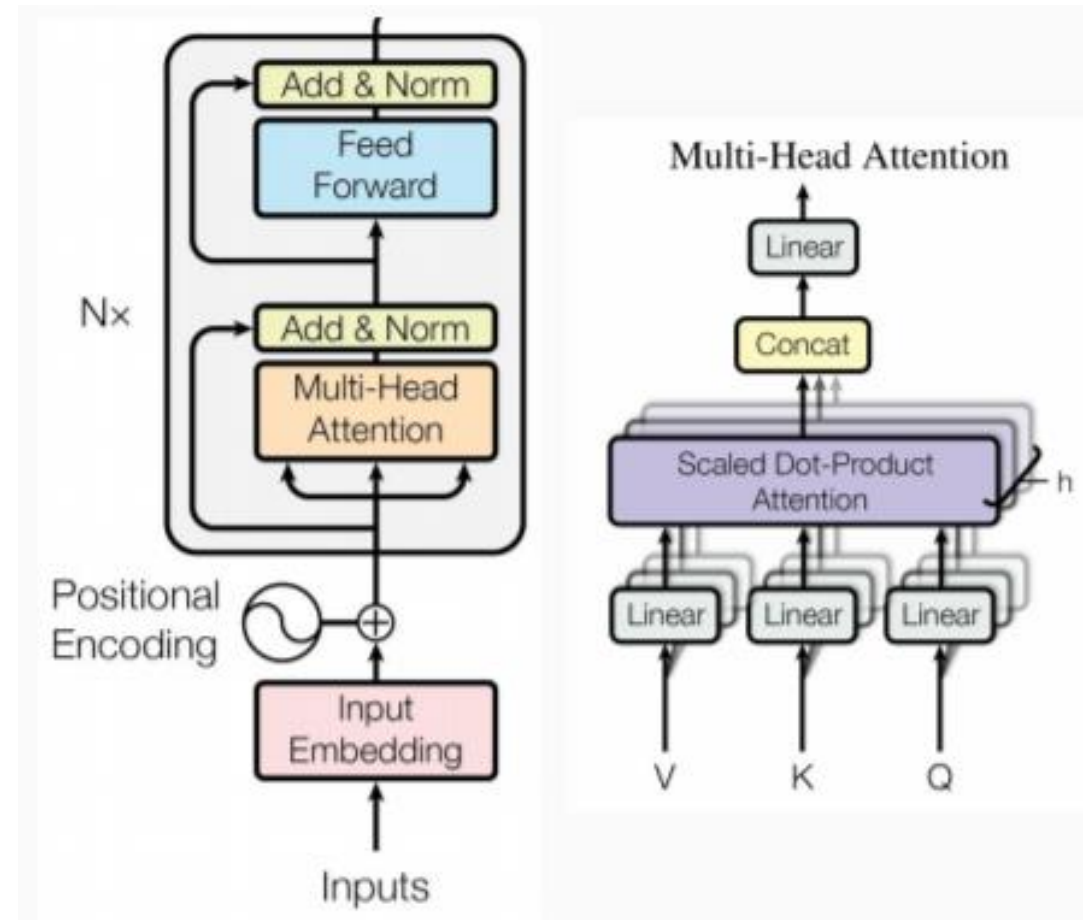
# Model Architecture

- BERT 는 Transformer Encoder 를 깊게 쌓은 모델
- BERT 는 베이스 모델이 110M 파라미터가 넘고 라지 모델은 340M 파라미터가 넘는 거대 인코더로 구성되어 있다.
- 이 거대 인코더를 입력 문장들을 인코딩하여 언어 모델을 학습시킨 뒤 파인 튜닝하면 여러 자연어 처리 Task 를 수행할 수 있다.



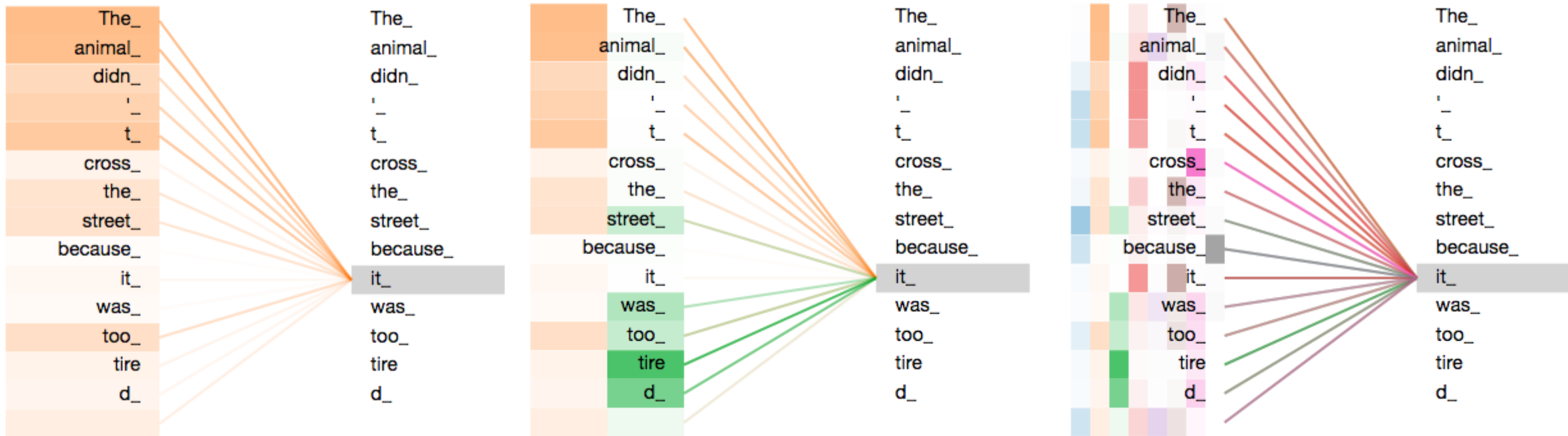
# Transformer Encoder

- Transformer Encoder 의 주요 모듈
- **Multi-headed self attention**
  - 모델의 컨텍스트를 파악
- **Feed-Forward layers**
  - 비선형 계층적 features 를 계산
- **Layer norm and residual connections**
  - 네트워크 훈련을 용이하게 함
- **Positioning Embedding**
  - 모델이 상대적인 포지셔닝을 배울 수 있도록 함



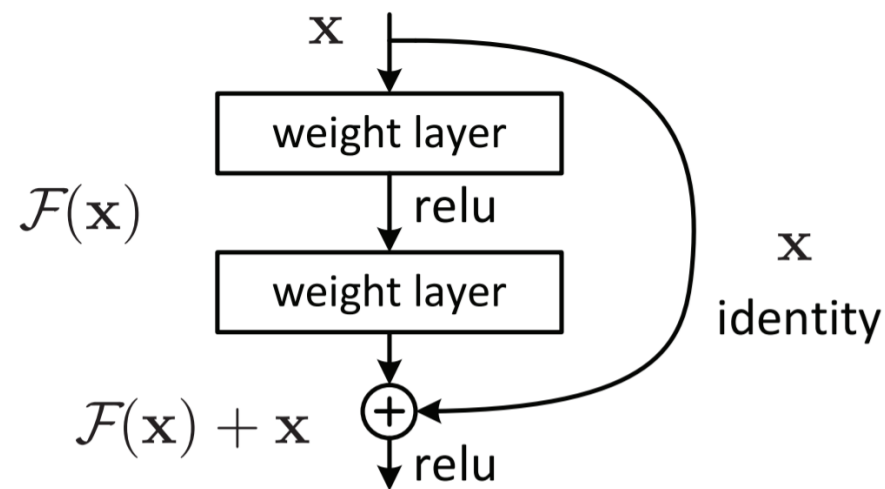
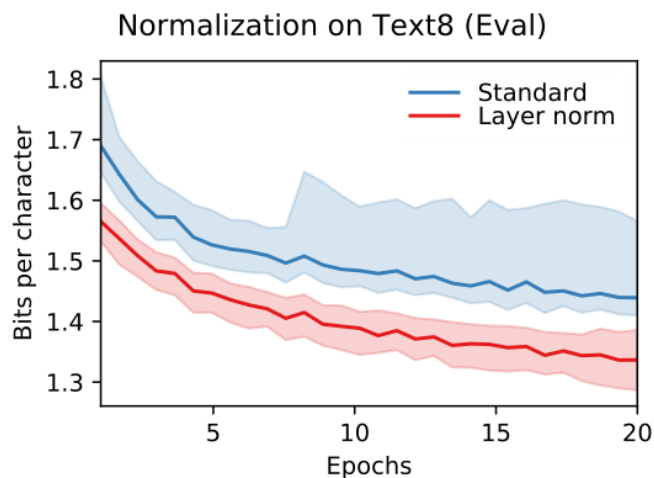
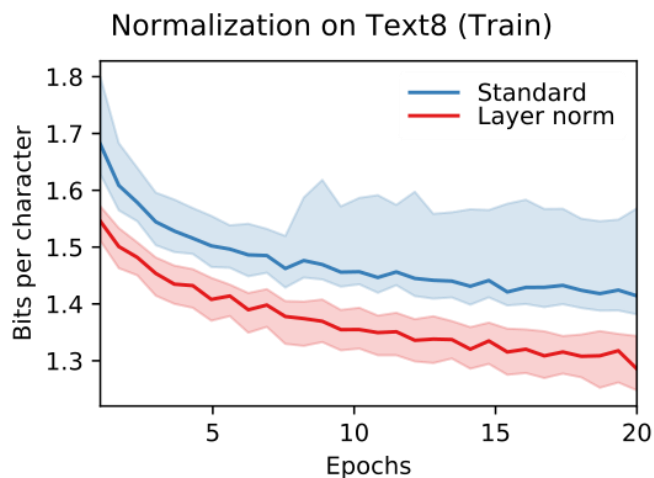
# Multi-headed self attention

- Self attention 은 언어 모델링을 통해 학습되는 것으로, 입력 시퀀스가 주어졌을 때, 주어진 시퀀스 내 단어 간의 연관 관계 정보를 얻을 수 있는 방법이다. 단어의 특정 정보를 가지고 있는 features 를 여러 개의 조각으로 나누어 self attention 을 수행하면 좀 더 효과적으로 단어 간의 연관 관계 정보를 얻을 수 있다.



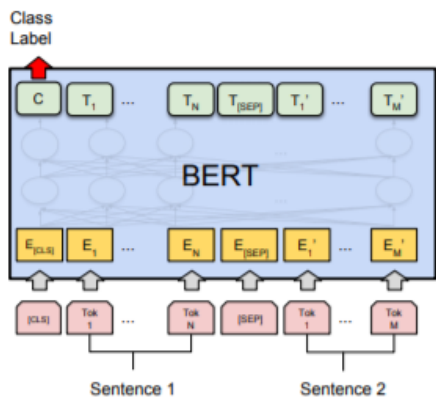
# Layer norm and residual connections

- Layer norm 은 각 features 의 값들을 한 layer 의 내의 값의 통계량으로 정규화하는 방법으로 학습을 안정적이고 더 빠르게 할 수 있다.
- Residual connections 은 특정 연산 전 features 를 연산 후 features 에 더하여 전파하는 방법으로 정보를 더욱 잘 보존할 수 있게 해주며, 훈련 시 gradient 소실 문제를 많은 부분 해결한다.

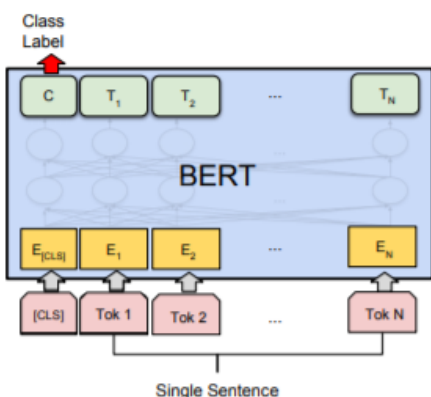


# Fine-Tuning Procedure

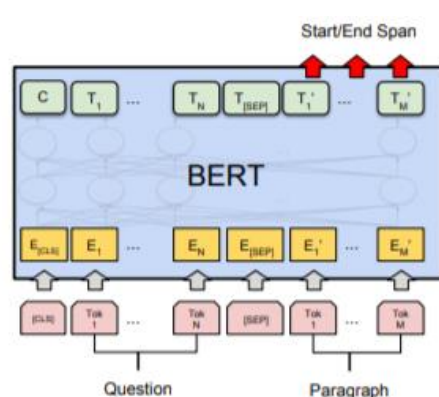
- Task에 상관없이 모든 입력의 첫 토큰은 [CLS]로 시작한다.
- 입력 문장이 두 개인 Task는 문장 사이에 [SEP] 토큰을 넣어 아래와 같이 입력한다.
- 입력 문장이 한 개인 Task는 [SEP] 없이 아래와 같이 입력한다.
- 분류 문제의 경우 [CLS] 토큰의 최종 임베딩을 이용하고, SQuAD 같이 정답 구간을 예측하는 문제는 시작 벡터( $\in \mathbb{R}^H$ )와 끝 벡터( $\in \mathbb{R}^H$ )를 도입하여 사용하며, 다중 분류 문제의 경우 각 해당 토큰의 최종 임베딩을 이용한다.



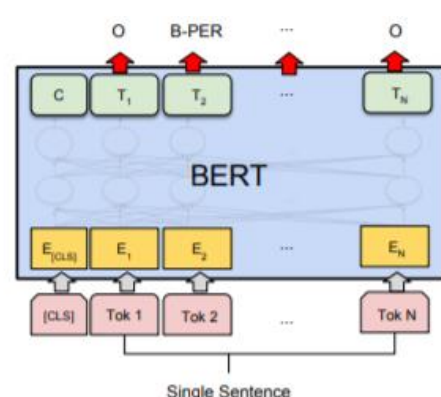
(a) Sentence Pair Classification Tasks:  
MNLI, QQP, QNLI, STS-B, MRPC,  
RTE, SWAG



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



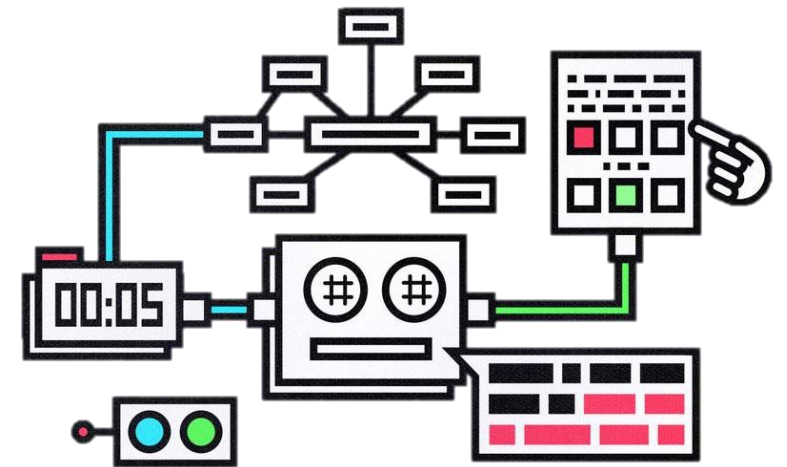
(c) Question Answering Tasks:  
SQuAD v1.1



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

# 별첨1: Language Models are Unsupervised Multitask Learners

- OpenAI GPT-2
- Model Architecture
- Generated Samples



# OpenAI GPT-2

- OpenAI 에서 Generative Pretrained Transformer (GPT)를 확장한 모델
- 논문 제목에서 알 수 있듯이 별도의 fine-tuning 과정 없이 언어 모델링을 학습시킨 것만으로 Reading Comprehension, Translation, QA 등 다양한 task에 대해 학습이 가능하다는 결과를 보여주었다.
- 15억 개의 파라미터를 가진 GPT-2 모델을 약 8백만 개의 문서에 대해 학습한 뒤 추가 학습없이 8개의 언어 모델링 데이터셋에 대해 테스트했을 때, 7개의 데이터 셋에 대해 SOTA를 달성했다.

	LAMBADA (PPL)	LAMBADA (ACC)	CBT-CN (ACC)	CBT-NE (ACC)	WikiText2 (PPL)	PTB (PPL)	enwik8 (BPP)	text8 (BPP)	WikiText103 (PPL)	1BW (PPL)
SOTA	99.8	59.23	85.7	82.3	39.14	46.54	0.99	1.08	18.3	<b>21.8</b>
117M	<b>35.13</b>	45.99	<b>87.65</b>	<b>83.4</b>	<b>29.41</b>	65.85	1.16	1.17	37.50	75.20
345M	<b>15.60</b>	55.48	<b>92.35</b>	<b>87.1</b>	<b>22.76</b>	47.33	1.01	<b>1.06</b>	26.37	55.72
762M	<b>10.87</b>	<b>60.12</b>	<b>93.45</b>	<b>88.0</b>	<b>19.93</b>	<b>40.31</b>	<b>0.97</b>	<b>1.02</b>	22.05	44.575
1542M	<b>8.63</b>	<b>63.24</b>	<b>93.30</b>	<b>89.05</b>	<b>18.34</b>	<b>35.76</b>	<b>0.93</b>	<b>0.98</b>	<b>17.48</b>	42.16



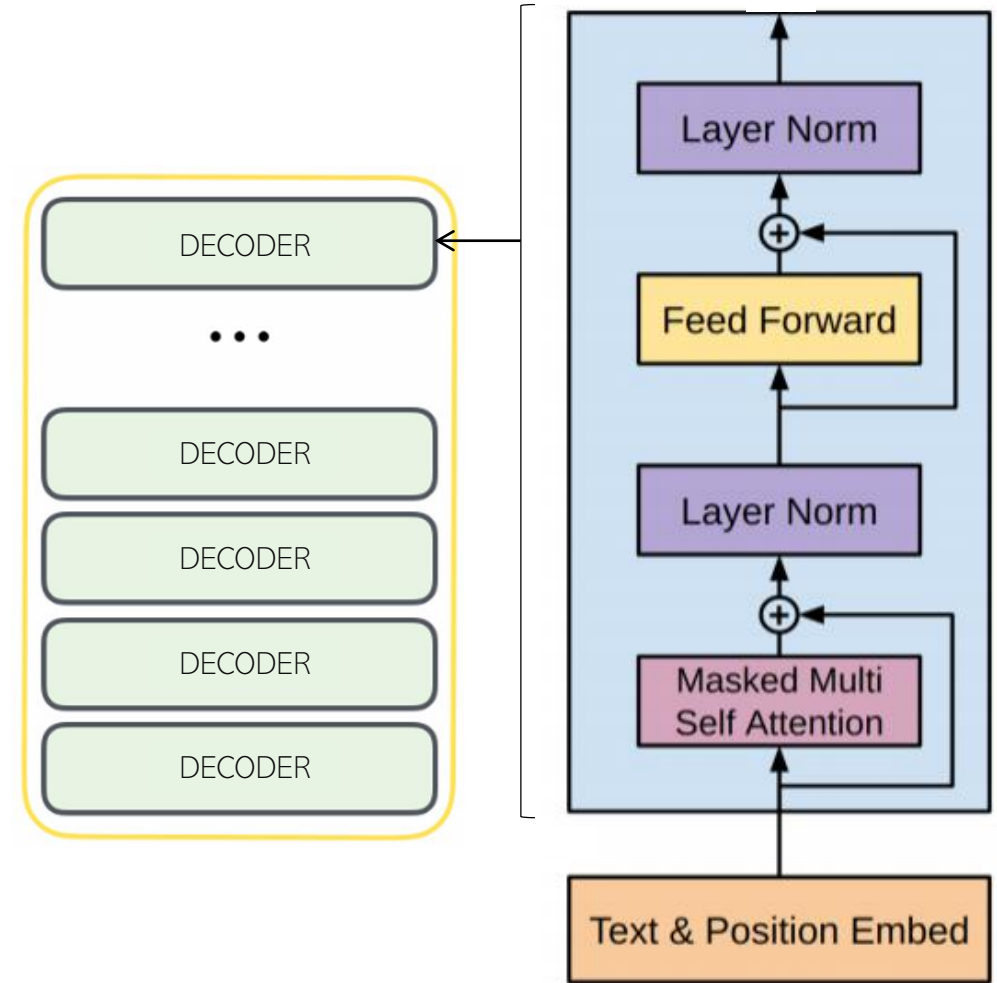
# Model Architecture

- OpenAI GPT 는 언어 모델링에 맞게 변형한 Transformer Decoder 를 깊게 쌓은 모델

- 총 4가지 크기의 모델에 대해 테스트

Parameters	Layers	$d_{model}$
117M	12	768
345M	24	1024
762M	36	1280
1542M	48	1600

- OpenAI GPT-2 는 OpenAI GPT의 Transformer 모듈을 조금 수정



# Generated Samples

## **Human-Written:**

*In a shocking finding, scientist discovered a herd of unicorns living in a remote, previously unexplored valley, in the Andes Mountains. Even more surprising to the researchers was the fact that the unicorns spoke perfect English.*

## **Machine-Written:**

The scientist named the population, after their distinctive horn, Ovid's Unicorn. These four-horned, silver-white unicorns were previously unknown to science.

Now, after almost two centuries, the mystery of what sparked this odd phenomenon is finally solved.

Dr. Jorge Pérez, an evolutionary biologist from the University of La Paz, and several companions, were exploring the Andes Mountains when they found a small valley, with no other animals or humans. Pérez noticed that the valley had what appeared to be a natural fountain, surrounded by two peaks of rock and silver snow.

Pérez and the others then ventured further into the valley. "By the time we reached the top of one peak, the water looked blue, with some crystals on top," said Pérez.

Pérez and his friends were astonished to see the unicorn herd. These creatures could be seen from the air without having to move too much to see them - they were so close they could touch their horns.

While examining these bizarre creatures the scientists discovered that the creatures also spoke some fairly regular English. Pérez stated, "We can see, for example, that they have a common 'language,' something like a dialect or dialectic."

Dr. Pérez believes that the unicorns may have originated in Argentina, where the animals were believed to be descendants of a lost race of people who lived there before the arrival of humans in those parts of South America.

While their origins are still unclear, some believe that perhaps the creatures were created when a human and a unicorn met each other in a time before human civilization. According to Pérez, "In South America, such incidents seem to be quite common."

However, Pérez also pointed out that it is likely that the only way of knowing for sure if unicorns are indeed the descendants of a lost alien race is through DNA. "But they seem to be able to communicate in English quite well, which I believe is a sign of evolution, or at least a change in social organization," said the scientist.

# Generated Samples

## **Human-Written:**

*We've trained a large language model called GPT-2 that generates realistic paragraphs of text, while also exhibiting zero shot generalization on tasks like machine translation, question answering, reading comprehension, and summarization - problems usually approached by using training datasets and models designed explicitly for these tasks.*

*A typical approach to language modeling is to learn the following task: predict the next word, given all of the previous words within some text. Last year, OpenAI's Generative Pre-trained Transformer (GPT) showed that language models trained on large amounts of data can be fine-tuned to specific tasks to achieve high performance. GPT-2 shows that much larger language models trained on a more diverse dataset derived from the internet begin to learn these NLP tasks without needing task-specific training data, instead learning from examples the system derives from the raw text. These systems also display a substantial qualitative jump in the realism and coherence of generated text.*

## **Machine-Written:**

Here you can see the most recent progress with Generative Pre-trained Transformer:

Figure 1: Generative Pre-trained Transformer training on several texts.

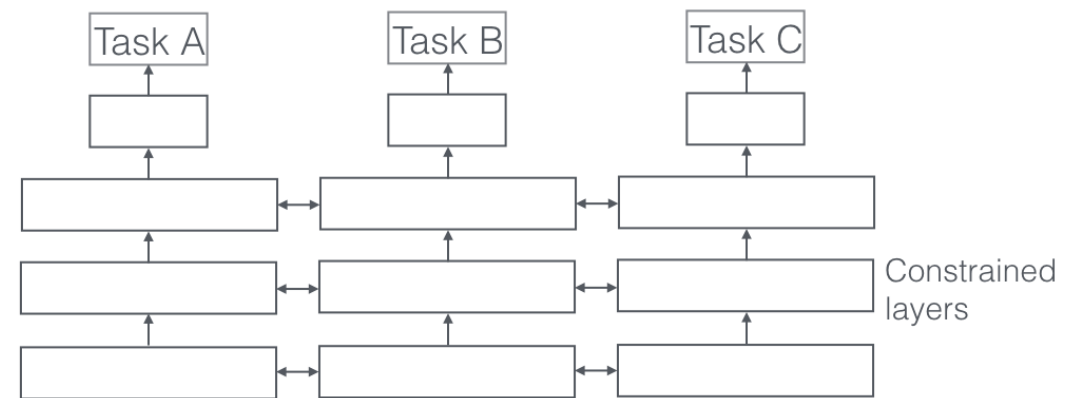
We are now preparing a collection of datasets for translation and machine translation in our language model. We will be using one of the large number of text samples provided by The New York Times.

We believe this project is the first step in the direction of developing large NLP systems without task-specific training data. That is, we are developing a machine language system in the generative style with no explicit rules for producing text.

We hope for future collaborations between computer scientists, linguists, and machine learning researchers.

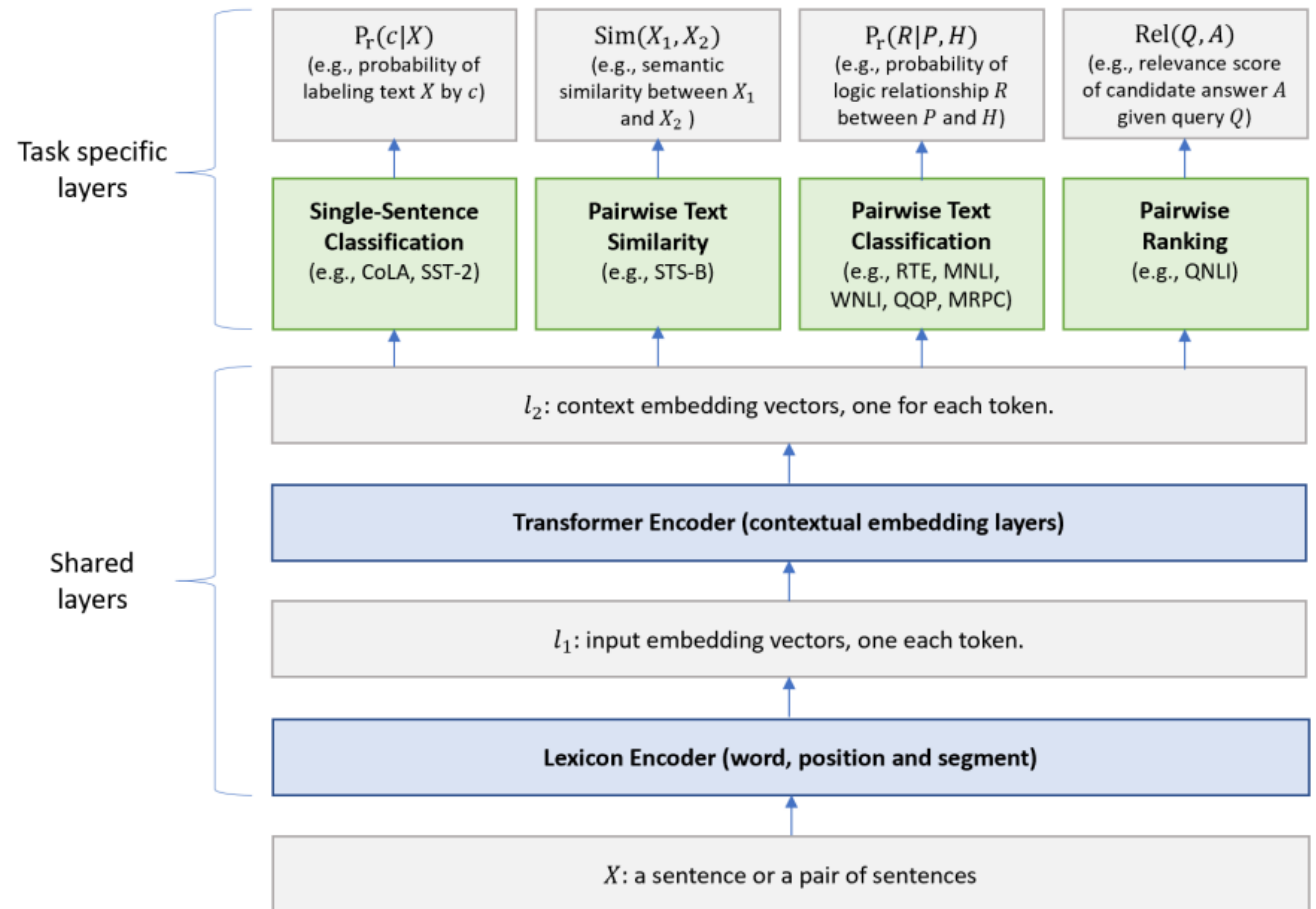
## 별첨2: Multi-Task Deep Neural Networks for NLU

- Multi-Task Learning
- Model Architecture
- Experimental Results



# Model Architecture

- BERT를 shared layers 로 사용
- Shared model 위로 각 Task 별 완전연결층을 연결



# Multi-Task Learning

- Multi-Task Learning 은 공유하는 파라미터를 이용하여 여러 Task 에 대해 동시에 학습시키는 것을 의미한다.
- 본 논문에서는 GLUE dataset 의 네가지 다른 Task (classification, regression, similarity scoring, relevance ranking) 에 맞게 loss function 을 정의하여 모든 9가지 dataset 을 합쳐서 한 번에 훈련을 진행한다.

---

**Algorithm 1:** Training a MT-DNN model.

---

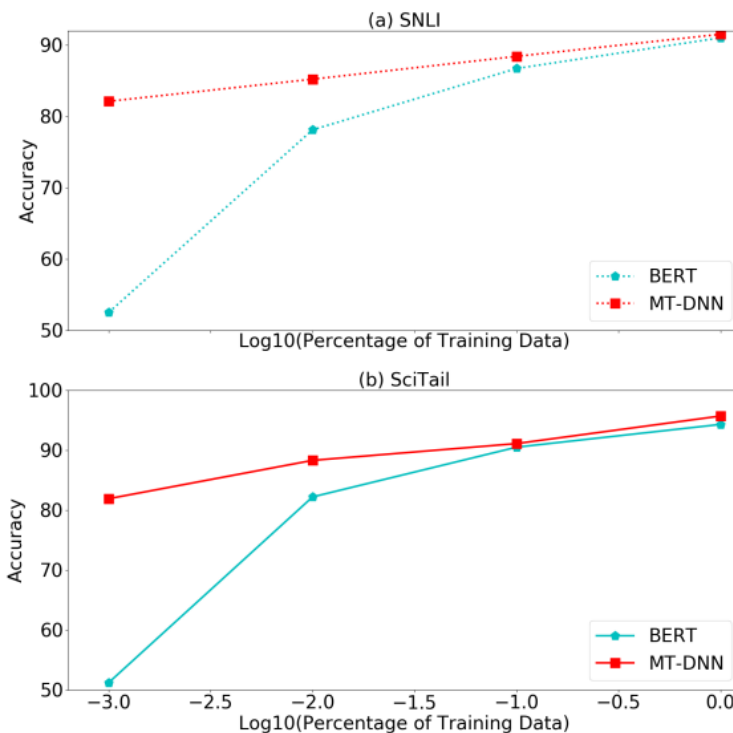
```
Initialize model parameters  $\Theta$  randomly.  
Pre-train the shared layers (i.e., the lexicon  
encoder and the transformer encoder).  
Set the max number of epoch:  $epoch_{max}$ .  
//Prepare the data for  $T$  tasks.  
for  $t$  in  $1, 2, \dots, T$  do  
| Pack the dataset  $t$  into mini-batch:  $D_t$ .  
end  
for  $epoch$  in  $1, 2, \dots, epoch_{max}$  do  
| 1. Merge all the datasets:  
|    $D = D_1 \cup D_2 \dots \cup D_T$   
| 2. Shuffle  $D$   
| for  $b_t$  in  $D$  do  
| | // $b_t$  is a mini-batch of task  $t$ .  
| | 3. Compute loss :  $L(\Theta)$   
| |    $L(\Theta)$  = Eq. 6 for classification  
| |    $L(\Theta)$  = Eq. 7 for regression  
| |    $L(\Theta)$  = Eq. 8 for ranking  
| | 4. Compute gradient:  $\nabla(\Theta)$   
| | 5. Update model:  $\Theta = \Theta - \epsilon \nabla(\Theta)$   
| end  
end
```

---

# Experimental Results

Model	CoLA 8.5k	SST-2 67k	MRPC 3.7k	STS-B 7k	QQP 364k	MNLI-m/mm 393k	QNLI 108k	RTE 2.5k	WNLI 634	AX	Score
BiLSTM+ELMo+Attn	36.0	90.4	84.9/77.9	75.1/73.3	64.8/84.7	76.4/76.1	79.9	56.8	65.1	26.5	70.5
Singletask Pretrain Transformer	45.4	91.3	82.3/75.7	82.0/80.0	70.3/88.5	82.1/81.4	88.1	56.0	53.4	29.8	72.8
GPT on STILTs	47.2	93.1	87.7/83.7	85.3/84.8	70.1/88.1	80.8/80.6	87.2	69.1	65.1	29.4	76.9
BERT <sub>LARGE</sub>	60.5	94.9	89.3/85.4	87.6/86.5	72.1/89.3	86.7/85.9	91.1	70.1	65.1	39.6	80.4
MT-DNN	<b>61.5</b>	<b>95.6</b>	<b>90.0/86.7</b>	<b>88.3/87.7</b>	<b>72.4/89.6</b>	<b>86.7/86.0</b>	<b>98.0</b>	<b>75.5</b>	<b>65.1</b>	40.3	<b>82.2</b>

- MT-DNN 은 단일 Task에 대해 학습하는 것보다 좋은 성능을 보여준다.
- Domain adaptation 실험 또한 진행하였는데 이는 각 해당 Task 를 제외하고 Multi-Task Learning 을 진행한 뒤 해당 Task 에 fine-tuning 하였다. 실험 결과, 적은 데이터로도 안정적인 성능을 보여주는 것을 볼 수 있다.





감사합니다.

