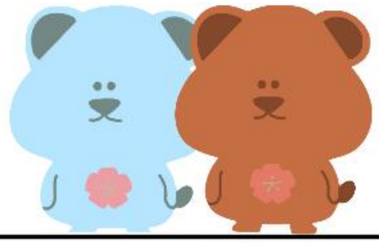
Transformers and Pretraining

송혜준, 김경민

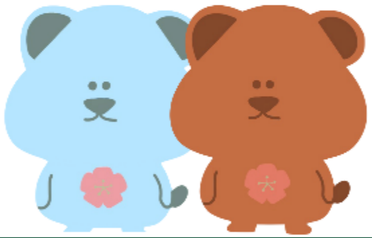
EWHA EURON

목차



EWHA EURON

#01 subword modeling



EWHA EURON

#01 subword modeling

#1 Word structure and subword models

Assumption: Word2vec 훈련에서 training set이 n개의 고정된 어휘를 가지고 있다

훈련된 Word2Vec에서는 training set에 있는 단어를 Embedding Vector로 만들지만 training set에 없는 새로운 단어는 **UNK** 토큰에 매핑된다.

주어진 문제를 풀 때 모르는 단어가 있으면 해결하기 어려움 => 단어를 subword로 나누어서 의미를 이해하자

Subword Segmenation

: Byte-Pair Encoding(BPE)을 사용하여 하나의 단어를 여러 subword로 분리하고 단어를 Embedding하는 것이다.

=> **Pretraining**에서 subwords가 input으로 들어간다.

#01 subword modeling

#2 The byte-pair encoding(BPE) algorithm

BPE는 서브워드 분리(subword segmentation) 알고리즘이다.

BPE을 요약하면, 글자(charcter) 단위에서 점차적으로 단어 집합(vocabulary)을 만들어 내는 Bottom up 방식의 접근을 사용한다.

우선 훈련 데이터에 있는 단어들을 **모든 글자(chracters)** or 유니코드(unicode) 단위로 단어 집합(vocabulary)를 만들고,

가장 많이 등장하는 **유니그램**을 하나의 유니그램으로 통합한다.

✓ Byte-Pair-Encoding

1. 단어를 문자 단위로 분해	……	connect
2. 전체 코퍼스에서 가장 자주 등장하는 연속된 두 문자열 병합	……	c## o## n## n## e## c## t
3. 병합된 토큰을 기반으로 분해 결과 조합	……	n## n## -> nn##
4. 종료 조건(병합 횟수, 토큰 개수 등)을 만족할 때 까지 1 ~ 3 반복	……	c## o## nn## e## c## t

#01 subword modeling

#2 The byte-pair encoding(BPE) algorithm: Example

어떤 훈련 데이터로부터 각 단어들의 빈도수를 카운트했다고 해보겠습니다. 그리고 각 단어와 각 단어의 빈도수가 기록되어져 있는 해당 결과는 임의로 딕셔너리(dictionary)란 이름을 붙였습니다.

```
# dictionary
# 훈련 데이터에 있는 단어와 등장 빈도수
low : 5, lower : 2, newest : 6, widest : 3
```

이 훈련 데이터에는 'low'란 단어가 5회 등장하였고, 'lower'란 단어는 2회 등장하였으며, 'newest'란 단어는 6회, 'widest'란 단어는 3회 등장하였다는 의미입니다. 그렇다면 딕셔너리로부터 이 훈련 데이터의 단어 집합(vocabulary)을 얻는 것은 간단합니다.

위의 딕셔너리에 BPE를 적용해봅시다. 우선 딕셔너리의 모든 단어들을 글자(chracter) 단위로 분리합니다. 이 경우 딕셔너리는 아래와 같습니다. 이제부터 딕셔너리는 자신 또한 업데이트되며 앞으로 단어 집합을 업데이트하기 위해 지속적으로 참고되는 참고 자료의 역할을 합니다.

```
# dictionary
l o w : 5,  l o w e r : 2,  n e w e s t : 6,  w i d e s t : 3
```

딕셔너리를 참고로 한 초기 단어 집합(vocabulary)을 아래와 같습니다. 간단히 말해 **초기 구성은 글자 단위로 분리된 상태입니다.**

```
# vocabulary
l, o, w, e, r, n, w, s, t, i, d
```

BPE의 특징은 알고리즘의 동작을 몇 회 반복(iteration)할 것인지를 사용자가 정한다는 점입니다. 여기서는 총 10회를 수행한다고 가정합니다. 다시 말해 **가장 빈도수가 높은 유니그램의 쌍을 하나의 유니그램으로 통합**하는 과정을 총 10회 반복합니다. 위의 딕셔너리에 따르면 빈도수가 현재 가장 높은 유니그램의 쌍은 (e, s)입니다.

1회 - 딕셔너리를 참고로 하였을 때 빈도수가 9로 가장 높은 (e, s)의 쌍을 es로 통합합니다.

```
# dictionary update!
l o w : 5,
l o w e r : 2,
n e w es t : 6,
w i d es t : 3
```

```
# vocabulary update!
l, o, w, e, r, n, w, s, t, i, d, es
```

2회 - 빈도수가 9로 가장 높은 (es, t)의 쌍을 est로 통합합니다.

```
# dictionary update!
l o w : 5,
l o w e r : 2,
n e w est : 6,
w i d est : 3
```

```
# vocabulary update!
l, o, w, e, r, n, w, s, t, i, d, es, est
```

#01 subword modeling

#2 The byte-pair encoding(BPE) algorithm: Example

3회 - 빈도수가 7로 가장 높은 (l, o)의 쌍을 lo로 통합합니다.

```
# dictionary update!
lo w : 5,
lo w e r : 2,
n e w est : 6,
w i d est : 3
```

```
# vocabulary update!
l, o, w, e, r, n, w, s, t, i, d, es, est, lo
```

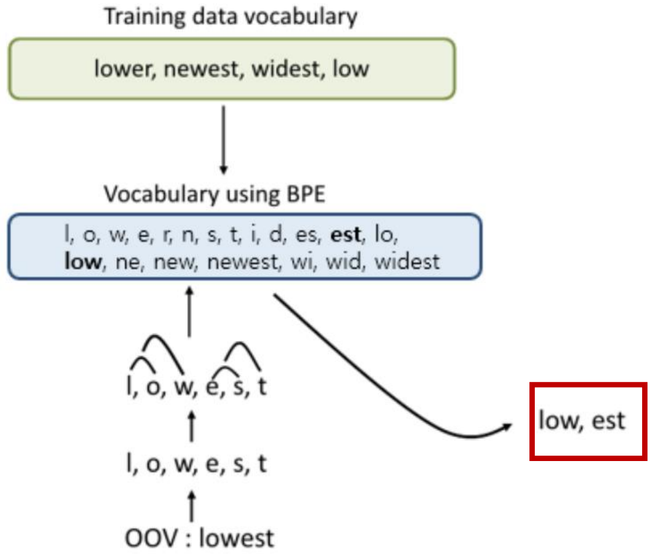
이와 같은 방식으로 총 10회 반복하였을 때 얻은 딕셔너리와 단어 집합은 아래와 같습니다.

```
# dictionary update!
low : 5,
low e r : 2,
newest : 6,
widest : 3
```

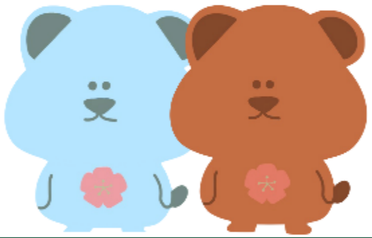
```
# vocabulary update!
l, o, w, e, r, n, w, s, t, i, d, es, est, lo, low, ne, new, newest, wi, wid, widest
```

기존 dictionary에는 없는 단어 “lowest ”

이 경우 테스트 과정에서 'lowest'란 단어가 등장한다면, 기존에는 OOV에 해당되는 단어가 되었겠지만 BPE 알고리즘을 사용한 위의 단어 집합에서는 더 이상 'lowest'는 OOV가 아닙니다. 기계는 우선 'lowest'를 전부 글자 단위로 분할합니다. 즉, 'l, o, w, e, s, t'가 됩니다. 그리고 기계는 위의 단어 집합을 참고로 하여 'low'와 'est'를 찾아냅니다. 즉, 'lowest'를 기계는 'low'와 'est' 두 단어로 인코딩합니다. 그리고 이 두 단어는 둘 다 단어 집합에 있는 단어이므로 OOV가 아닙니다.



#02 model pretraining from word embeddings



#02 model pretraining from word embeddings

#1 Problems with Word2Vec

Word2Vec : 각 단어에 대해 사전에 학습된 Embedding vector 부여 (fixed sized)

문맥을 고려하지 못한 Embedding 방식
: Word2Vec로 어떤 단어를 삽입할 때 그 단어 주위의 있는 단어들을 고려하지 않음

⇒다른 문맥임에도 동일 vector가 매핑
"나는 기록을 한다"라는 예시 문장에서 앞의 record와 뒤의 record는 "기록하다"와 기록으로 다른 의미이지만 같은 Word2Vec Embedding 으로 매핑됨

I **record** the **record**

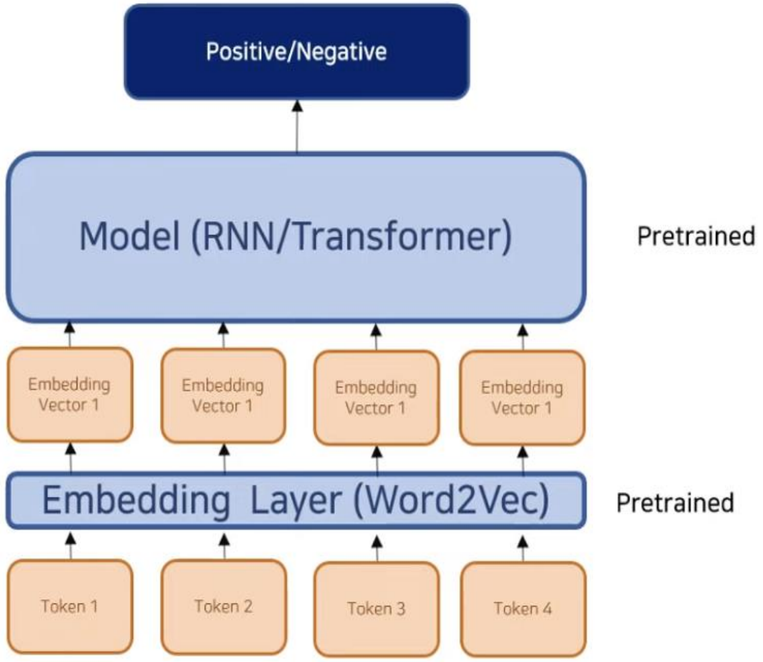
=> 그래서 **개념적**으로 Word2vec를 훈련시키는 것이 중요하다.

#02 model pretraining from word embeddings

#2 pretraining word embedding

- Pretrain(사전학습)
  - Input의 일부를 훼손
  - 모델이 input을 복원
  - 의미적/문법적 구조 학습

- Downstram Task
  - Pretrained model의 word Embedding을 가지고 실제 수행하고자 하는 Task
    - QA/Sentiment Analysis/NLI(Natural language inference)
  - 학습 방식: Pretrained Model의 weight를 initial weight로 삼아 fine-tuning 진행



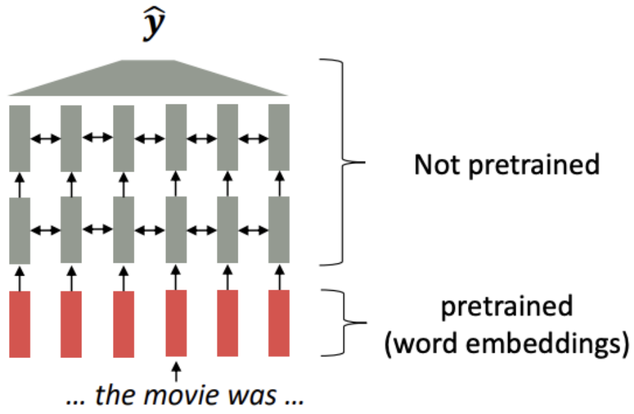
#02 model pretraining from word embeddings

#2 pretraining word embedding : Before&Now

Pretrained word Embedding으로 시작했다.(no context!)
Task에 대한 Training을 할 때 LSTM/Transformer에 문맥을 통합하는 방법에 대해 배우는 것 중요하다.

문제점:
- Downstream Task (ex QA)에 대한 훈련 데이터는 언어의 모든 상황적 측면을 가르치기에 충분해야 한다.
- 네트워크의 대부분의 paramters는 randomly initialized

[Recall, *movie* gets the same word embedding, no matter what sentence it shows up in]

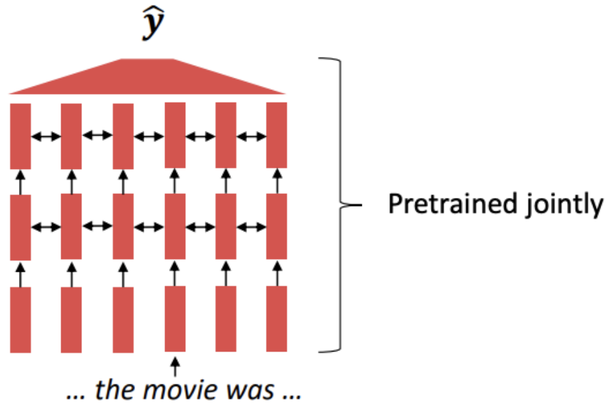


NLP 네트워크의 모든(또는 거의 모든) parameter는 Pretraining을 통해 초기화됩니다.

Pretraining 방법은 모델에서 입력의 일부를 숨기고 훈련시켜 해당 부분을 재구성한다.

장점:
- 언어의 표현

[This model has learned how to represent entire sentences through pretraining]



#02 model pretraining from word embeddings

#3 What can we learn from reconstructing the input?

Pretrained task : 문장의 일부를 변형하고 해당 부분의 단어를 예측한다.

- 이화여자대학교는 서울시 ___에 위치해 있다. [단편적 사실]
- 나는 어제 강아지와 놀__다. [문법 정보]
- 아이유는 노래를 잘 부른다. ___는 목 관리를 위해 노력한다. [상호 참조]
- 동생이 강아지 밥을 주려고 거실에 갔다. 동생 옆에 있던 강아지는 잠이와서 ___를 떠났다. [추론]
- 이 수열을 봐봐. 1, 1, 2, 3, 5,__,13 … [단순 계산]

Q. Pretraining을 하는 구체적인 방식은 무엇인지 궁금해요!

EWHA EURON

#02 model pretraining from word embeddings

#4 Transformer

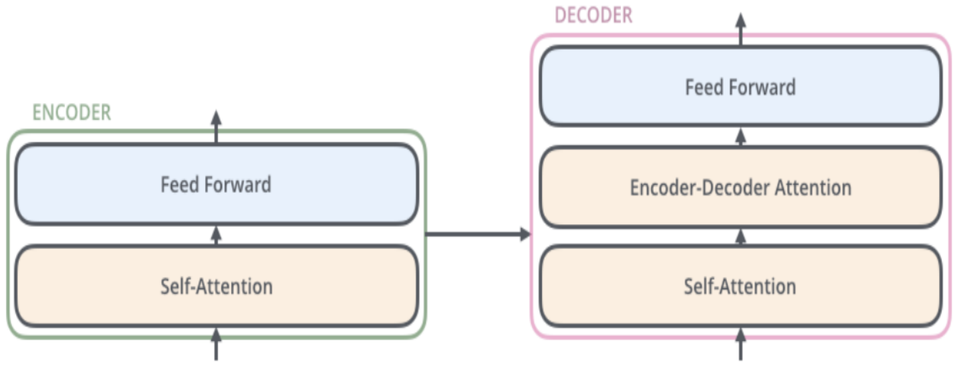
✓ **Encoder : input이 연속적으로 레이어 통과**

- Output : Contextual Embedding
  (batch, sequence length, embed dim)
- 입력 문장에서 맥락/의미/문법 구조 등을 파악

✓ **Decoder : input과 encoder output이 연속적으로 레이어 통과**

- Output : 조건부 문장 생성
  (batch, sequence length, vocab dim)
- 입력 문장과 이전 생성 문장을 바탕으로 문장 생성

Transformer 모델은 6개의 encoders와 6개의 decoders로 이루어져 있으며, 각각 동일한 구조이며 독립적인 모듈로써 존재한다. 위 그림에서 보다시피 입력은 encoder에 의해 임베딩되고, 임베딩된 context는 각각의 decoder에 입력으로 들어간다.



➡ **인코더와 디코더가 수행하는 역할이 다름**

#02 model pretraining from word embeddings

#4 Transformer: Encoder

Step 1: Input Embedding
- Input은 sequence of tokens
- 각각의 token들은 pre-trained embedding으로 표현된다.

Positional encoding
- RNN과 다르게 token은 순서의 개념이 없다
- 순서 정보를 넣기 위해서 Transformer는 input에 positional encoding을 더한다.

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{\text{model}}})$$
$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{\text{model}}})$$

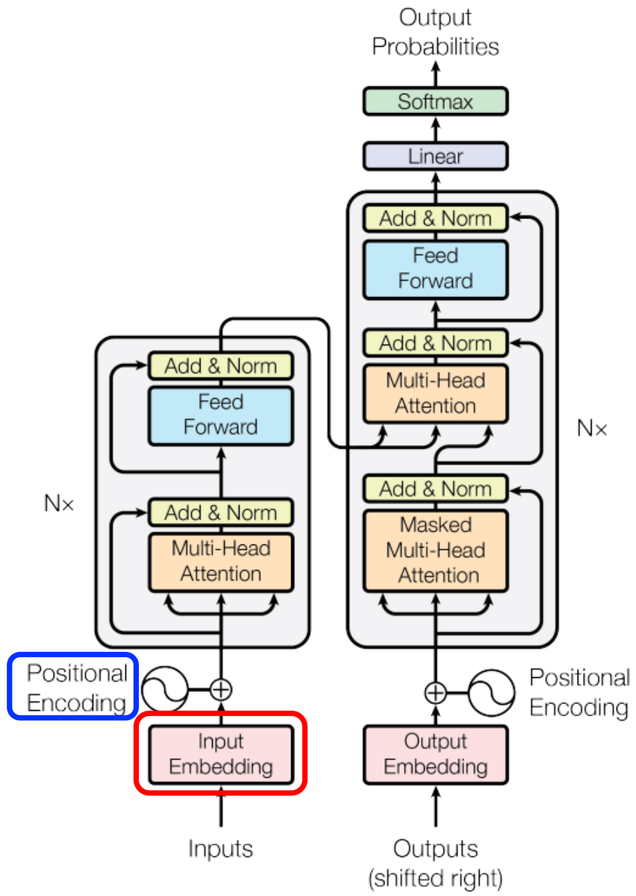


Figure 1: The Transformer - model architecture.

#02 model pretraining from word embeddings

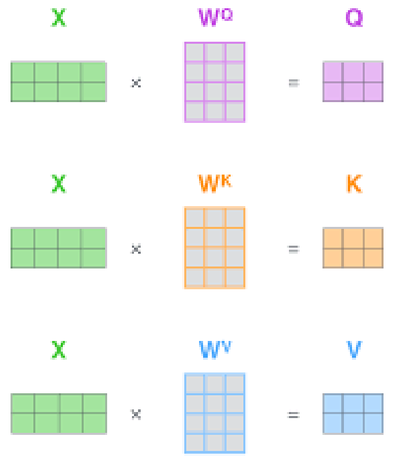
#4 Transformer: Encoder

Step 2: Contextualizing the word Embedding

- Self-attention:
  - 각각의 단어 Embedding들이 자기 자신에 대한 Q, K, V Embedding 를 갖는다.
  - X로 표현되는 input Embedding이 WQ, WK, WV를 곱해서(input->output 차원을 맞추기 위한 Linear transformation) Q, K, V를 구한다.

  - Input sequence에 들어온 문장전체(자기 자신)에서 K, Q, V가 다 나온다.
  - Q는 모든 단어마다 한번씩 될 수 있고 K는 자기 자신을 포함한 모든 단어이고, 모든 단어에 대해서 Attention을 한다.

  - Attention : Q를 기준으로 V의 weighted sum을 구하는 것이고 weighted sum은 Q와 K의 similarity를 구한다. (by dot product)
    -> attention value (Z)는 contextualized new word Embedding of same size



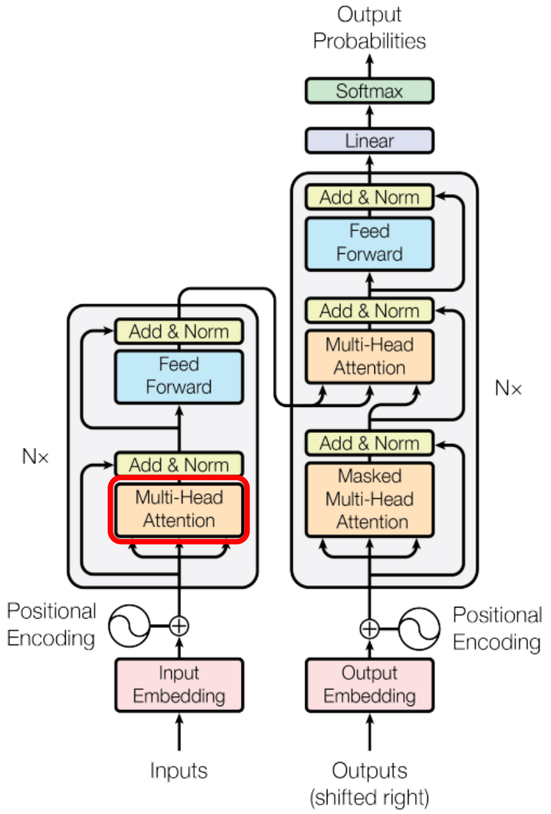


Figure 1: The Transformer - model architecture.

#02 model pretraining from word embeddings

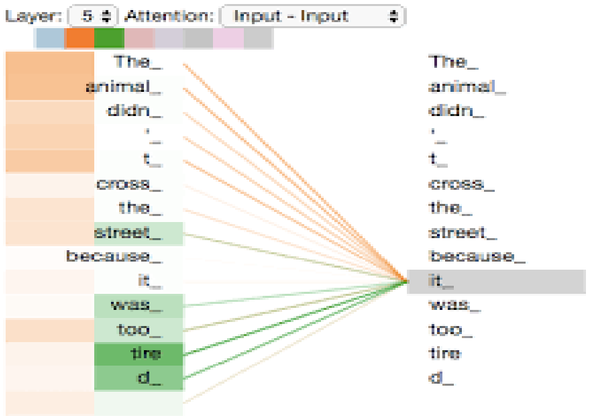
#4 Transformer: Encoder

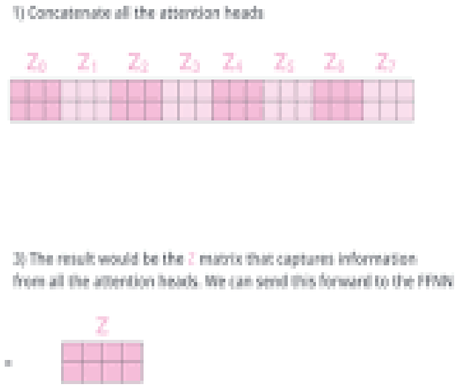
Step 2: Contextualizing the word Embedding

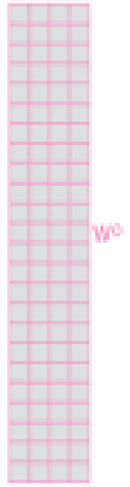
- **Multi-head** Self-Attention:

중의적인 의미를 가지는 단어가 존재하기 때문에 모델이 여러 곳을 attention 할 수 있도록 여러 번 attention 수행한다

- Attention을 n번 반복해서 나온 모든 attention value들을 concatenate한 뒤 linear transformation을 통해서 (Wo 곱함) input과 같은 크기로 되돌린다
- Input과 output Embedding의 크기는 변하지 않고 값만 바뀐다







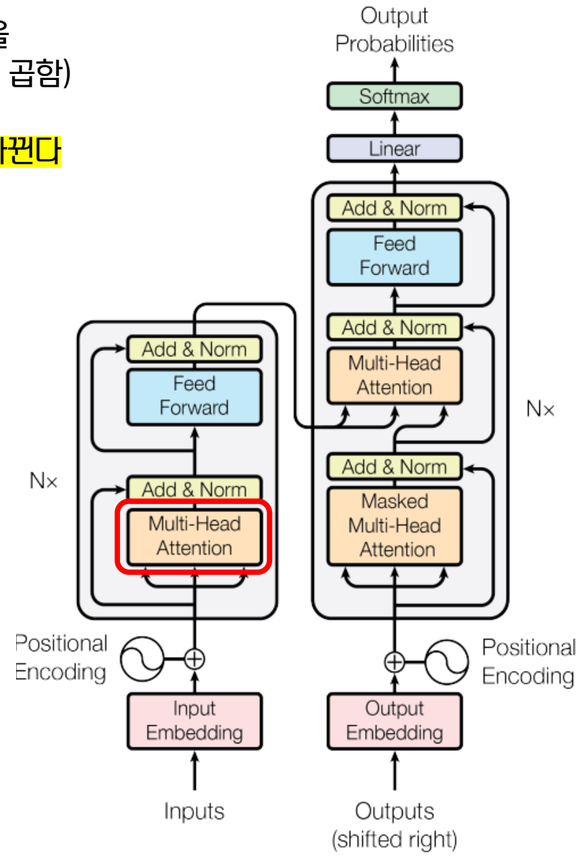


Figure 1: The Transformer - model architecture.

#02 model pretraining from word embeddings

1) This is our input sentence*

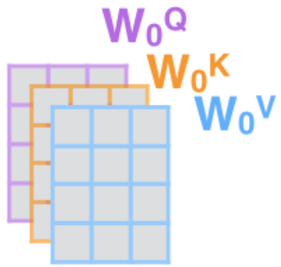
2) We embed each word*

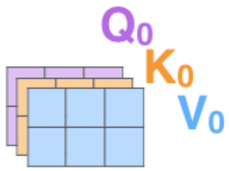
3) Split into 8 heads. We multiply X or R with weight matrices

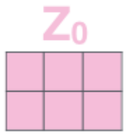
4) Calculate attention using the resulting Q/K/V matrices

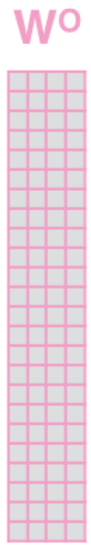
5) Concatenate the resulting Z matrices, then multiply with weight matrix W^O to produce the output of the layer

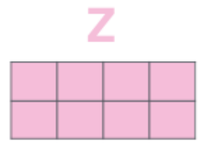
Thinking
Machines

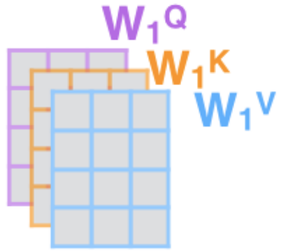


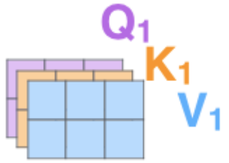


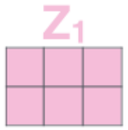




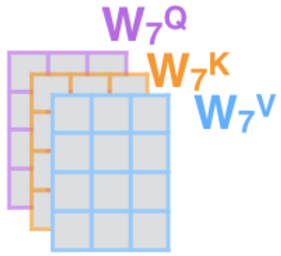


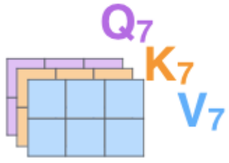


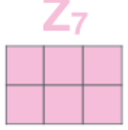




* In all encoders other than #0, we don't need embedding. We start directly with the output of the encoder right below this one







#02 model pretraining from word embeddings

#4 Transformer: Encoder

Step 3: Feed-forward layer

- New, contextualized word embedding 이 추가적인 FC layer를 통과한다.
- Output은 여전히 같은 크기의 contextualized token embedding이다.

Residual connection & layer normalization이 multi-head self attention과 FC layer의 끝에 추가된다.

$$\mathrm{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2$$

Stacked Self-attention Blocks

- Multiple (N) self-attention block 들이 쌓여 있다.
  - Lowest block은 input으로 pretrained embedding을 갖는다.
- 마지막 layer의 output은 같은 길이의 변형된 토큰이고 각각은 같은 크기의 contextualized embedding이다.

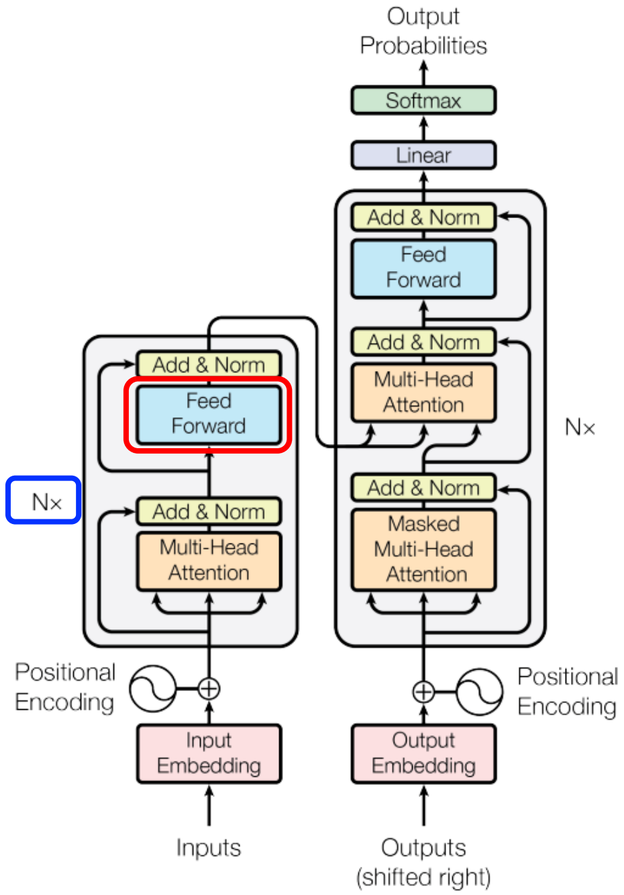


Figure 1: The Transformer - model architecture.

#02 model pretraining from word embeddings

#4 Transformer: Decoder

Step 4: Decoder input

- Given the encoder output Z = {z1,⋯,zn}, 하나의 output sequence를 auto-regressive하게 생성한다.
  - 전에 생성된 output을 다음 output을 생성하는데 추가적인 input으로 사용한다.
- Positional encoding은 encoder에서와 같은 방식으로 적용된다.

Step 5: Masked Multi-head self attention

- Q가 특정 단어일 때 K가 모든 단어를 보여주면 미리 결과를 보여주는 것이기 때문에 특정 단어 이후의 sequence는 K, V에서 모두 가려야 한다.
- Q가 참조로 하는 K와 V 중에서 아직 만들지 않은 sequence는 제외한다.

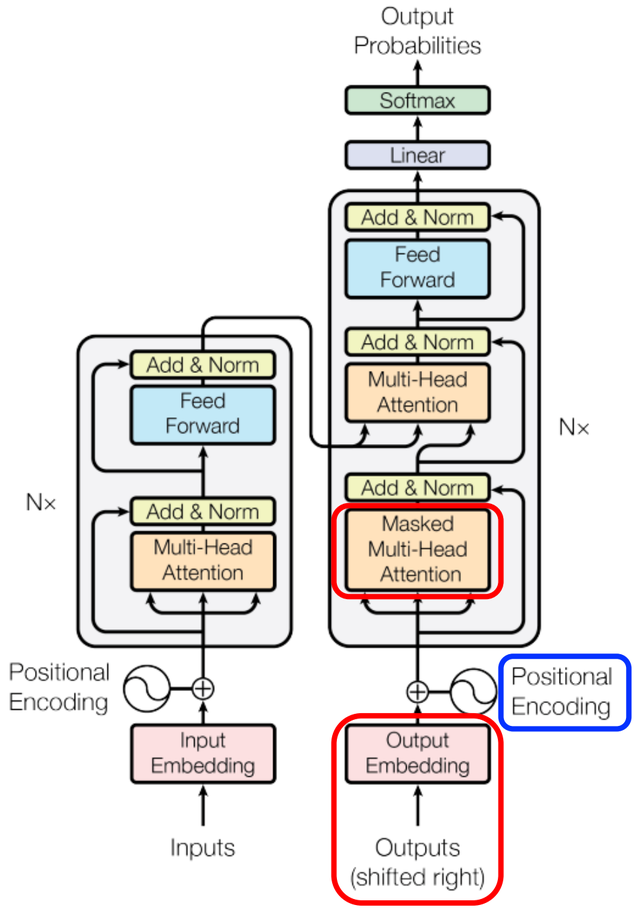


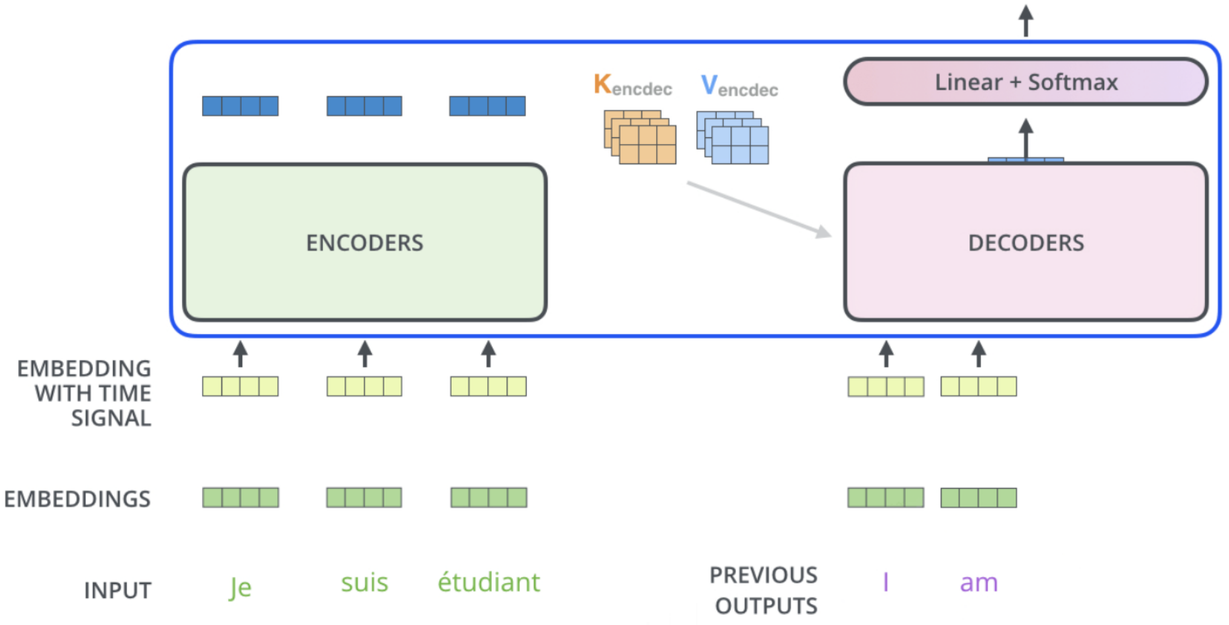
Figure 1: The Transformer - model architecture.

#02 model pretraining from word embeddings

#4 Transformer: Decoder

Step 6: Encoder Decoder attention

- Q : the query from decoder
- K, V : the key and value from encoder
- 이것을 제외한 나머지는 multi-head self attention layer와 동일하다
- 여기서는 masking없고 encoded sequence의 전체를 볼 수 있다.



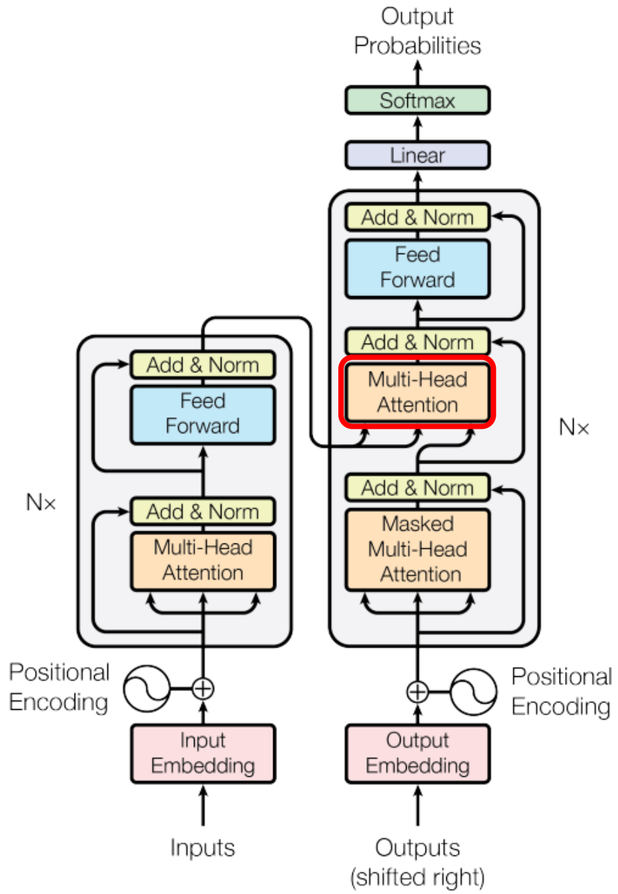


Figure 1: The Transformer - model architecture.

#02 model pretraining from word embeddings

#4 Transformer: Decoder

Step 7: Feed-forward Network

Step 8: Linear layer

Step 9 : Softmax layer

- Class score를 probability로 만든다.

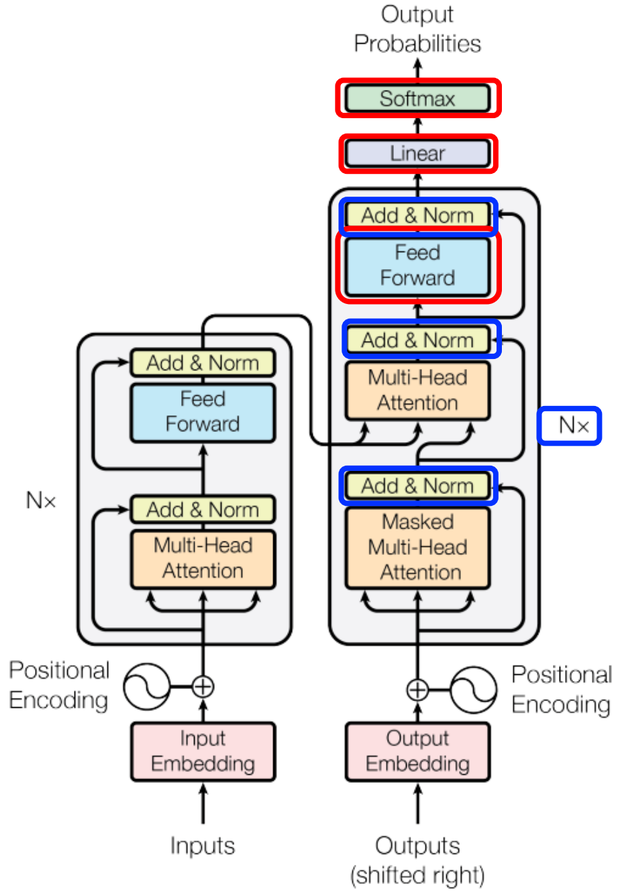


Figure 1: The Transformer - model architecture.

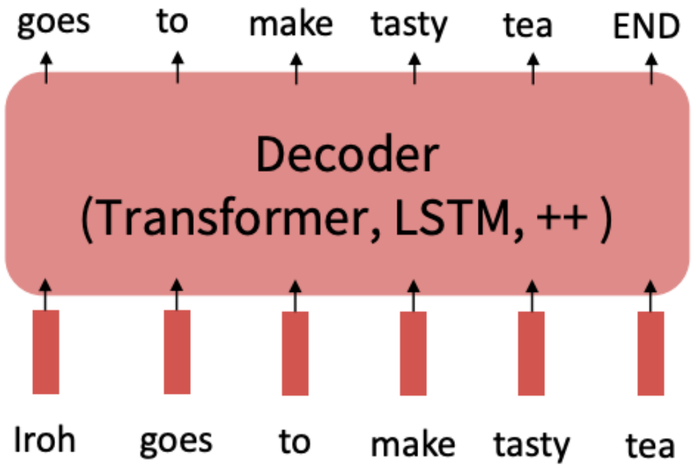
#02 model pretraining from word embeddings

#5 The Pretraining / Finetuning Paradigm

Pretraining can improve NLP applications by serving as parameter initialization.

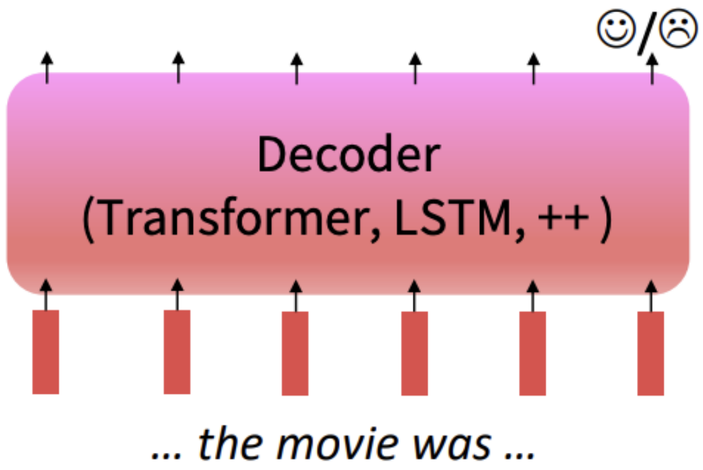
Step 1: Pretrain (on language modeling)

Lots of text; learn general things!



Step 2: Finetune (on your task)

Not many labels; adapt to the task!

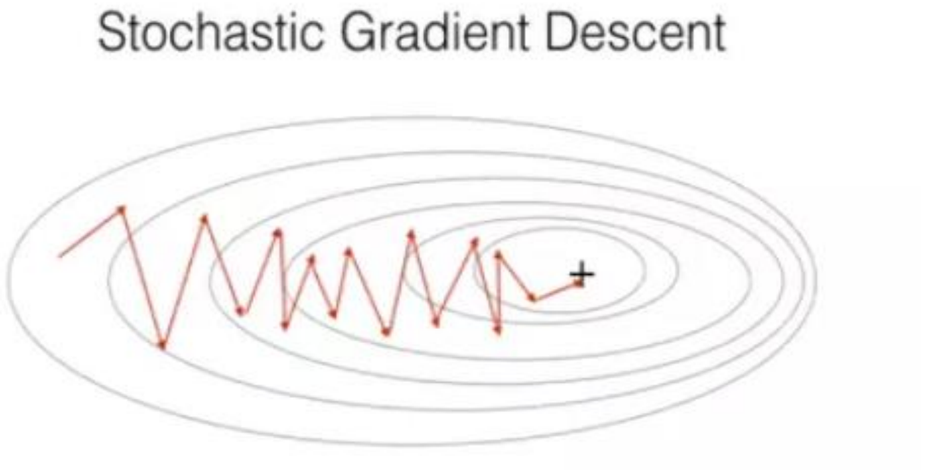


#02 model pretraining from word embeddings

#6 Stochastic gradient descent and pretrain/finetune

확률적 경사 하강법 (Stochastic Gradient Descent, SGD)

: 데이터를 미니배치로 무작위로 선정하여 경사 하강법으로 매개변수를 갱신하는 방법으러 추출된 데이터 한개에 대해서 그래디언트를 계산하고, 경사 하강 알고리즘을 적용합니다.



전체 데이터를 사용하는 것이 아닌 랜덤하게 추출한 일부 데이터를 사용한 것이다. 따라서 학습 과정에서 결과의 진폭이 크고 불안정하며 속도가 매우 빠르다.

SGD 장점
- shooting이 일어나기 때문에 local optimal에 빠질 리스크가 적다
- step에 걸리는 시간이 짧아서 수렴속도가 상대적으로 빠르다

SGD 단점
- global optimal을 찾지 못할 가능성이 있다(오차율이 크다)

EWHA EURON

#02 model pretraining from word embeddings

#6 Stochastic gradient descent and pretrain/finetune

SGD로 신경망을 훈련하는 관점에서 pretraining과 finetuning은 도움이 된다.

- $\hat{\theta} = \min_{\theta} L_{pretrain}(\theta)$
- $\min_{\theta} L_{finetune}(\theta)\ (initial\ weight = \hat{\theta})$

pretrain model의 loss를 최소값에 근사하면서 θ.hat 을 얻을 수 있고

θ.hat에서 시작해서 **Finetune model**의 loss를 줄일 수 있다.

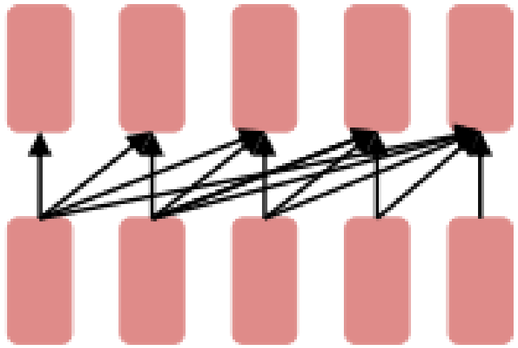
- Random initializing θ에 비해 $\hat{\theta}$ 가 global optimum에 가까울 수 있다.
- -> 적은 데이터로도 잘 학습, 쉽게 수렴

#03 Model Pretraining 3 Ways

Pretraining for 3 types of Architectures

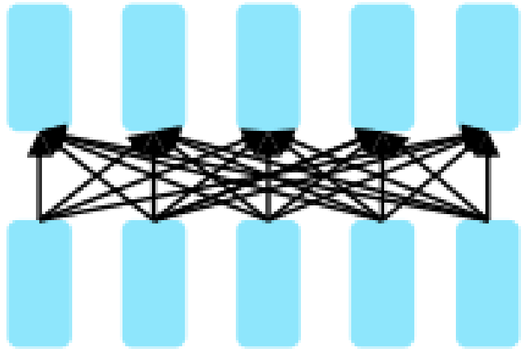
Encoder

- Language model
- 이전 시점의 토큰을 조건으로 하는 생성모델



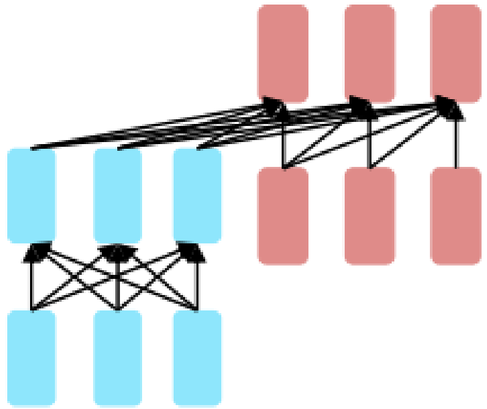
Decoder

- Bidirectional Context
- Language model로 pretrain 불가



Encoder-Decoder

- Good parts of decoders and encoders
- Best way to pretrain?

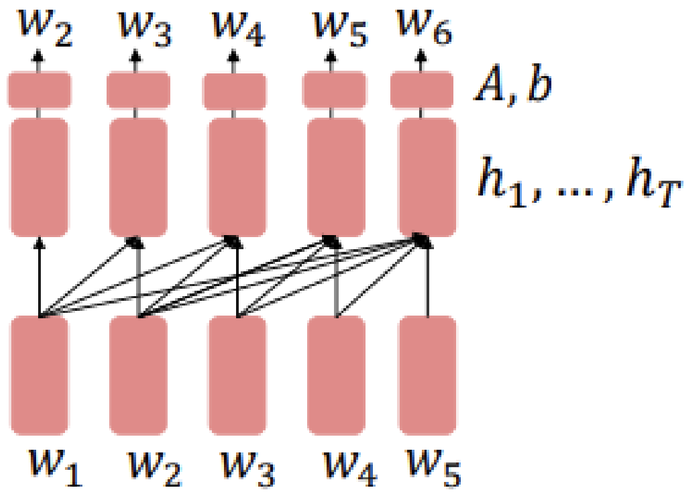


#03 Model Pretraining 3 Ways

Decoder

Pretraining decoder

- Input과 encoder output0이 연속적으로 레이어 통과
- Output : 조건부 문장 생성 (sequence)
- 입력된 문장과 이전 생성 문장을 바탕으로 그 다음 문장 생성



#03 Model Pretraining 3 Ways

Decoder : GPT-1

Generative Pretrained Transformer

- Transformer decoder (w/ 12 layers)
- 단방향 모델
- Train data : BooksCorpus

Finetuning task

- Natural Language Inference: Label pairs of sentences as entailing/contradictory/neutral

Method	MNLI-m	MNLI-mm	SNLI	SciTail	QNLI	RTE
ESIM + ELMo [44] (5x)	-	-	89.3	-	-	-
CAFE [58] (5x)	80.2	79.0	89.3	-	-	-
Stochastic Answer Network [35] (3x)	80.6	80.1	-	-	-	-
CAFE [58]	78.7	77.9	88.5	83.3		
GenSen [64]	71.4	71.3	-	-	82.3	59.2
Multi-task BiLSTM + Attn [64]	72.2	72.1	-	-	82.1	**61.7**
Finetuned Transformer LM (ours)	**82.1**	**81.4**	**89.9**	**88.3**	**88.1**	56.0

#03 Model Pretraining 3 Ways

Decoder : GPT-2

A larger version of GPT trained on more data

- Produce relatively convincing samples of natural language

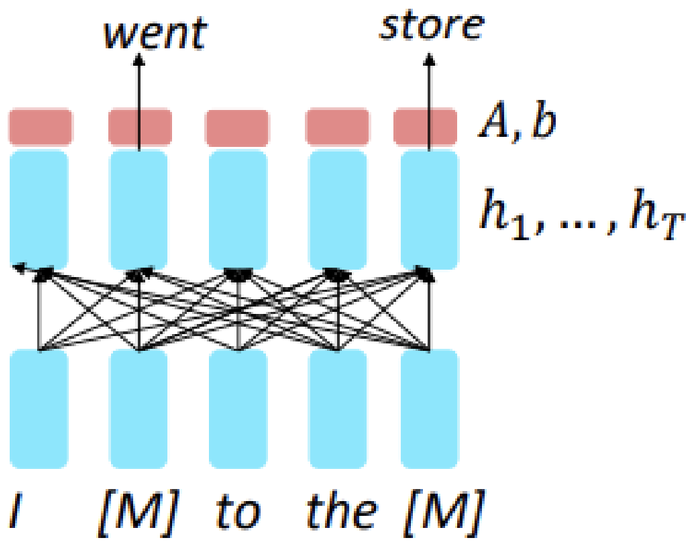
Context (human-written): In a shocking finding, scientist discovered a herd of unicorns living in a remote, previously unexplored valley, in the Andes Mountains. Even more surprising to the researchers was the fact that the unicorns spoke perfect English.
GPT-2: The scientist named the population, after 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.

#03 Model Pretraining 3 Ways

Encoder

Pretraining encoder

- Input이 연속적으로 레이어 통과
- Output : Contextual Embedding
- 입력된 문장에서 맥락/의미/문법 구조 등 파악
- Masked LM



#03 Model Pretraining 3 Ways

Encoder : BERT

Bidirectional Encoder Representations from Transformers

- The pretraining input to BERT was two separate contiguous chunks of text
- Trained to predict whether one chunk follows the other or is randomly sampled
- 양방향 모델
- Train data : BooksCorpus, English Wikipedia
  - Pretraining is expensive and impractical on a single GPU

Finetuning task

- Finetuning is practical and common on a single GPU

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**

#03 Model Pretraining 3 Ways

Encoder : Limitations of pretrained encoders & Extensions of BERT

Limitation

- BERT and other pretrained encoders don't naturally lead to nice autoregressive (1-word-at-a-time) generation methods

Extensions of BERT

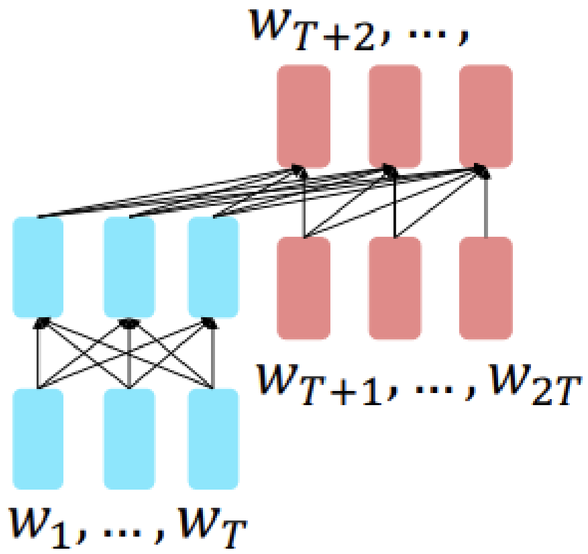
- RoBERTa : mainly just train BERT for longer and remove next sentence prediction
  - More compute, more data can improve pretraining even when not changing the underlying Transformer encoder
- SpanBERT: masking contiguous spans of words makes a harder, more useful pretraining task

#03 Model Pretraining 3 Ways

Encoder-decoder

Like language modeling

- Encoder portion benefits from bidirectional context
- Decoder portion is used to train the whole model through language modeling

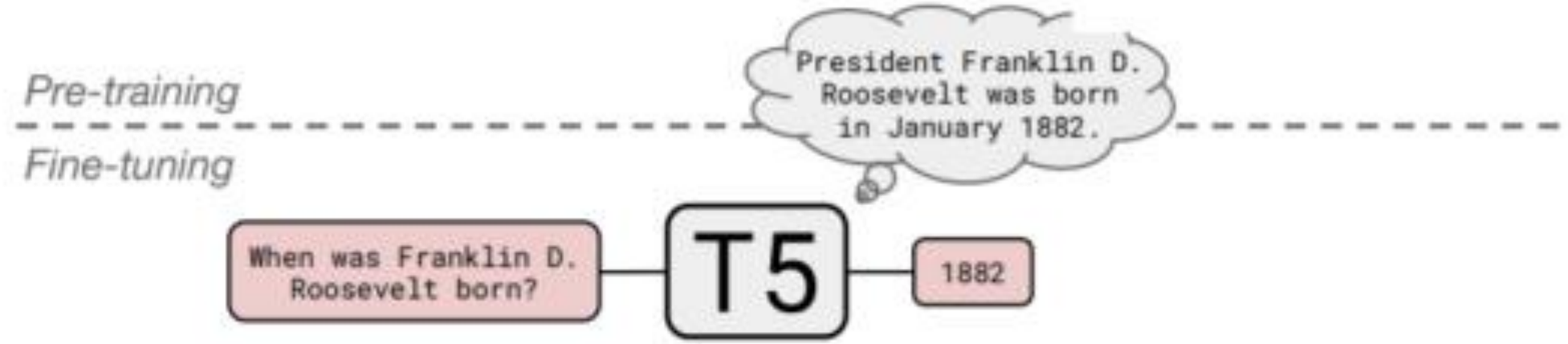


Architecture	Objective	Params	Cost	GLUE	CNNDM	SQuAD	SGLUE	EnDe	EnFr	EnRo
★ Encoder-decoder	Denoising	$2P$	M	**83.28**	**19.24**	**80.88**	**71.36**	**26.98**	**39.82**	**27.65**
Enc-dec, shared	Denoising	P	M	82.81	18.78	**80.63**	**70.73**	26.72	39.03	**27.46**
Enc-dec, 6 layers	Denoising	P	$M/2$	80.88	18.97	77.59	68.42	26.38	38.40	26.95
Language model	Denoising	P	M	74.70	17.93	61.14	55.02	25.09	35.28	25.86
Prefix LM	Denoising	P	M	81.82	18.61	78.94	68.11	26.43	37.98	27.39
Encoder-decoder	LM	$2P$	M	79.56	18.59	76.02	64.29	26.27	39.17	26.86
Enc-dec, shared	LM	P	M	79.60	18.13	76.35	63.50	26.62	39.17	27.05
Enc-dec, 6 layers	LM	P	$M/2$	78.67	18.26	75.32	64.06	26.13	38.42	26.89
Language model	LM	P	M	73.78	17.54	53.81	56.51	25.23	34.31	25.38
Prefix LM	LM	P	M	79.68	17.84	76.87	64.86	26.28	37.51	26.76

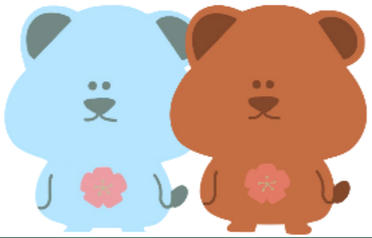
#03 Model Pretraining 3 Ways

Encoder-decoder : T5

Can be finetuned to answer a wide range of questions, retrieving knowledge from its parameters



Interlude : What does pretraining learn?



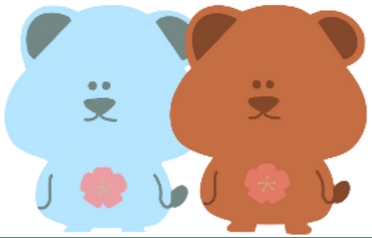
#04 Interlude : What does pretraining learn?

What kinds of things does pretraining learn?

There's increasing evidence that pretrained models learn a wide variety of things about the statistical properties of language. Taking our examples from the start of class:

- *Stanford University is located in _____, California.* [Trivia]
- *I put ___ fork down on the table.* [syntax]
- *The woman walked across the street, checking for traffic over ___ shoulder.* [coreference]
- *I went to the ocean to see the fish, turtles, seals, and _____.* [lexical semantics/topic]
- *Overall, the value I got from the two hours watching it was the sum total of the popcorn and the drink. The movie was ___.* [sentiment]
- Iroh went into the kitchen to make some tea. Standing next to Iroh, Zuko pondered his destiny. Zuko left the _____. [some reasoning – this is harder]
- I was thinking about the sequence that goes 1, 1, 2, 3, 5, 8, 13, 21, ____ [some basic arithmetic; they don't learn the Fibonnaci sequence]
- Models also learn – and can exacerbate racism, sexism, all manner of bad biases.
- More on all this in the interpretability lecture!

Very Large Models & In-context learning



#05 Very Large Models & In-context learning

GPT-3

Very large language models seem to perform some kind of learning ‘without gradient’

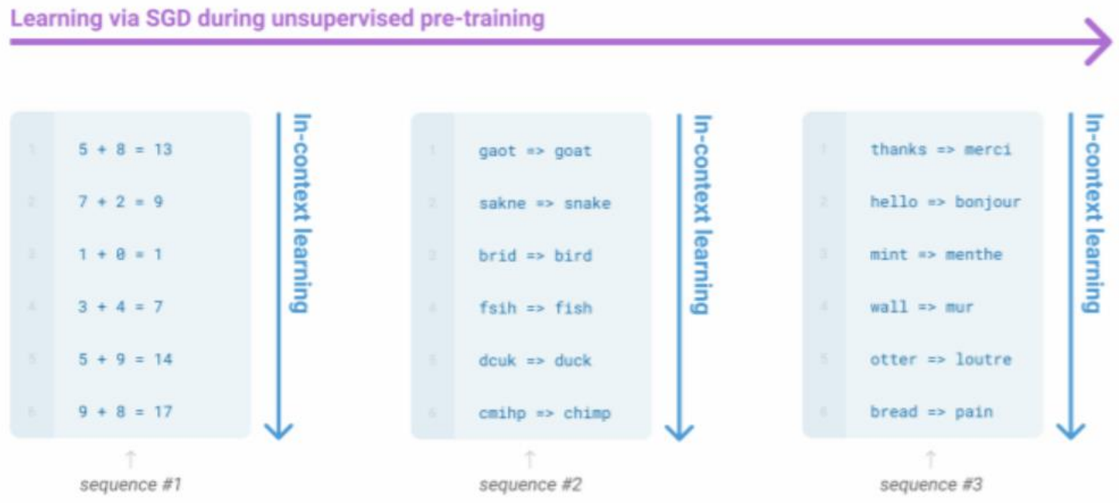
- GPT-3 has 175 billion parameters

Input (prefix within a single Transformer decoder context):

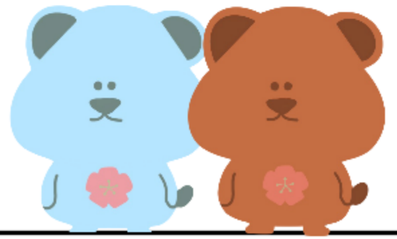
“ thanks -> merci
hello -> bonjour
mint -> menthe
otter -> ”

Output (conditional generations):

loutre…”



THANK YOU



EWHA EURON