AI Application Specialist

대규모 언어모델 이해 및 파인튜닝 (PART-1)

2025.8.18-19

사내교수 이재원 / 첨단기술아카데미

SEU
Samsung Electronics University

Schedule

시간	1 일차	2 일차
08:30 ~ 09:20	• 과정 OT: 과정 소개 / AVD 소개	• LLM Fine-Tuning: PEFT (LoRA, …)
09:30 ~ 10:20	• [개요] Tokenizer	• [실습-4] PEFT/LoRA (40m)
10:30 ~ 11:20	• [실습-1] Text Tokenizer (30m)	• LLM Fine-Tuning: Preference Tuning (PPO, DPO)
11:30 ~ 12:20	• [개요] Word Embedding / Language Modeling	• [실습-5] Preference Tuning - DPO (40m)
12:30 ~ 13:30	중식	
13:30 ~ 14:20	• Transformers Library / DataSet	• LLM Evaluation
14:30 ~ 15:20	• [실습-2] Text Generation API (30m)	• [실습-6] LLM Evaluation (30m)
15:30 ~ 16:20	• LLM Fine-Tuning: SFT	• Transformer Architecture 및 최근 동향
16:30 ~ 17:20	• [실습-3] LLM Fine-Tuning - SFT (40m)	• 과정 정리 및 QA

Overview

대규모 언어 모델 개발 과정

- Pre-Training Corpus
  - IF Corpus
  - Data Augmentation
- Data Vectorization
  - Tokenizer
  - Word Embedding

- Model Architecture
  - Transformer
- Language Modeling
  - MLM, CLM

- Pre-Training
  - Foundation Model
- Fine-Tuning
  - Supervised Fine-Tuning
  - PEFT
  - Preference Tuning

- Text Generation
- In-Context Learning
- RAG

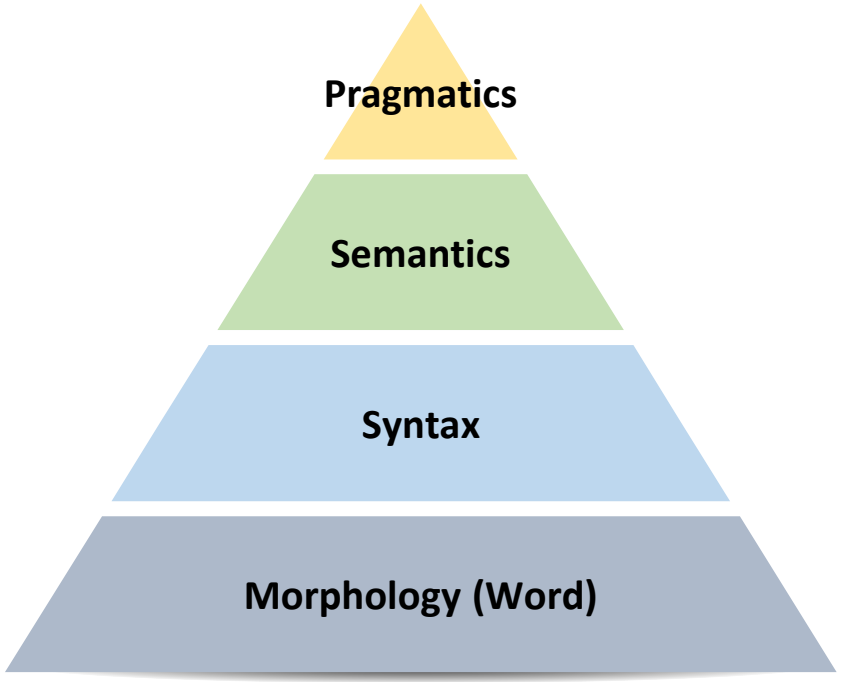
- Benchmark Dataset
- Evaluation Metric

CONTENTS

SEU
Samsung Electronics University

NLP Basics

- **WORD**
  - Word Representation
  - Morphological Analysis
  - Part-Of-Speech, Tagging, NER
- **SYNTAX**
  - Syntactic Parsing & Grammars
  - Dependency Parsing
- **SEMANTICS**
  - Word-level: Word Sense Disambiguation
  - Sentence-level: Semantic Role Labeling
- **PRAGMATICS**
  - Discourse Structures
  - Coherence / Anaphora Resolution



Natural Language Processing Pyramid

1.1 Preprocessing – Text Tokenizer

토큰화 (Tokenizing)

- 문장을 토큰으로 분할하는 과정으로 NLP 핵심 구성 요소
  1) Word-based Tokenization: Vocabulary Size, Unknown Word 이슈
  2) Character-based Tokenization: Semantic Meaning, Sequence Length 이슈 (Chinese Character)
  3) Subword Tokenization: Rare words → Subwords (Agglutinative Language)
- 주요 Subword Tokenization 방법: BPE, WordPiece, Unigram
- 자연어처리 모델 성능에 중요한 영향을 미치며, OOV or UNK(Unknown Token) 이슈와 밀접한 관련

* out of vocabulary

"시간이 화살처럼 지나간다"

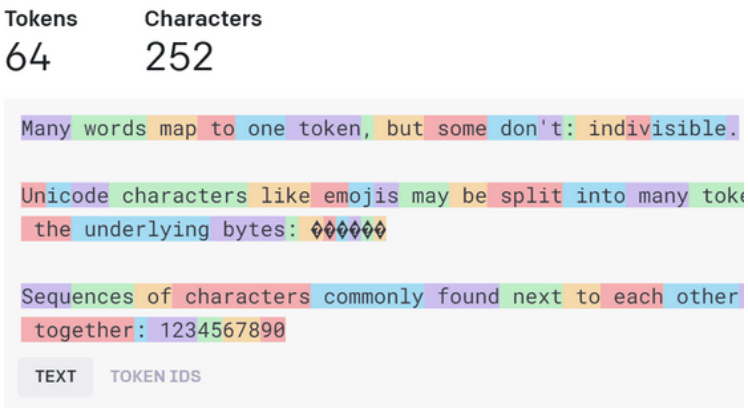
시간이	화살처럼	지나간다	Word-based Tokenization

시	간	이	화	살	처	럼	지	나	간	다	Character-based Tokenization

시간	이	화살	처럼	지나	간다	Subword Tokenization

1.1 Preprocessing – Text Tokenizer

- **BPE (Byte Pair Encoding)**
  - 1994년 제안된 데이터 압축 알고리즘
  - 자연어처리에서 서브워드로 분리하기 위한 목적으로 활용
  - 기본단위 (단일문자)의 리스트로 어휘 사전을 만들고, 가장 많이 등장하는 글자의 쌍을 하나의 토큰으로 병합
  - 점진적으로 새 토큰 만드는 과정을 반복하여 원하는 어휘 사전 크기 확보
  - https://arxiv.org/pdf/1508.07909.pdf (Univ. of Edinburgh, 2016)
  - 적용 사례: GPT, GPT-2, RoBERTa, BART, …



1.1 Preprocessing – Text Tokenizer

BPE (Byte Pair Encoding)

Algorithm 1 Learn BPE operations

```
import re, collections

def get_stats(vocab):
  pairs = collections.defaultdict(int)
  for word, freq in vocab.items():
    symbols = word.split()
    for i in range(len(symbols)-1):
      pairs[symbols[i],symbols[i+1]] += freq
  return pairs

def merge_vocab(pair, v_in):
  v_out = {}
  bigram = re.escape(' '.join(pair))
  p = re.compile(r'(?<!\S)' + bigram + r'(?!\S)')
  for word in v_in:
    w_out = p.sub(''.join(pair), word)
    v_out[w_out] = v_in[word]
  return v_out

vocab = {'l o w </w>' : 5, 'l o w e r </w>' : 2,
         'n e w e s t </w>':6, 'w i d e s t </w>':3}
num_merges = 10
for i in range(num_merges):
  pairs = get_stats(vocab)
  best = max(pairs, key=pairs.get)
  vocab = merge_vocab(best, vocab)
  print(best)
```

예제

```
0) Vocab: [l, o, w, e, r, n, s, t, i, d]
```

```
1) Vocab: [l, o, w, e, r, n, s, t, i, d, es]
```

```
2) Vocab: [l, o, w, e, r, n, s, t, i, d, es, est]
```

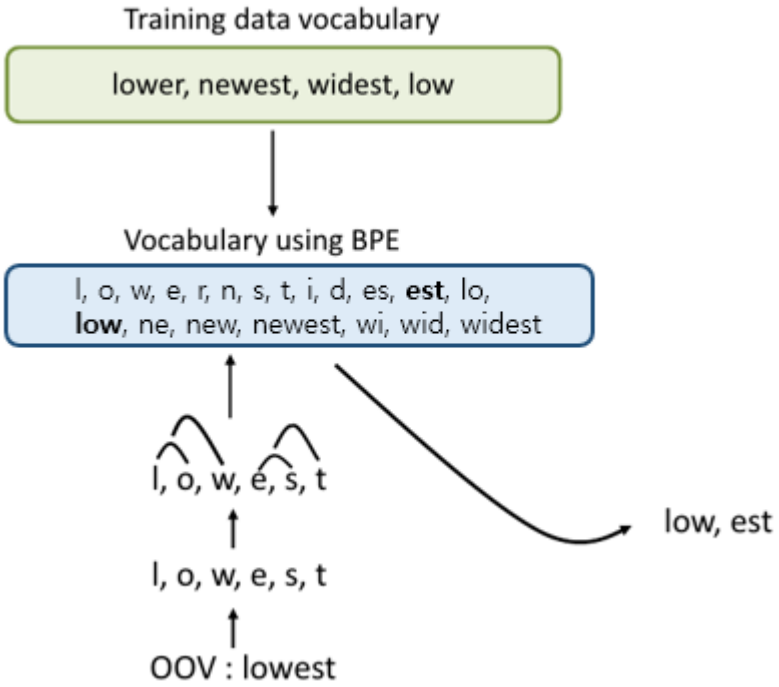
```
3) Vocab: [l, o, w, e, r, n, s, t, i, d, es, est, lo]
```

...

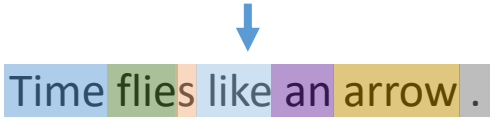
```
10) Vocab: [l, o, w, e, r, n, s, t, i, d, es, est,
            lo, low, ne, new, newest, wi, wid, widest]
```

1.1 Preprocessing – Text Tokenizer

BPE (Byte Pair Encoding)



Time flies like an arrow .



[1074, 315, 3293, 359, 2786, 3761, 150]

Token	Token ID	Dim-1	Dim-2	Dim-3	…	Dim-n
“Time”	1074	0.17	0.92	0.62	…	0.23
“flie”	315	0.02	0.34	0.51	…	0.39
“s”	3293	0.41	0.23	0.13	…	0.98
…	…	…	…	…	…	…

Ref. https://wikidocs.net/22592

1.1 Preprocessing – Text Tokenizer

WordPiece Tokenizer

- Google이 BERT 사전학습을 위해 개발
- BPE와 동일하게 특수 토큰과 알파벳을 포함한 작은 Vocabulary 로 시작
- 접두사(BERT ##)를 추가하여 하위단어 식별
- Likelihood 값이 가장 높은 쌍을 병합하는 방법

$$score = \frac{freq_{pair}}{(freq_{1st\_element} \times freq_{2nd\_element})}$$

예제

```
("hug", 10), ("pug", 5), ("pun", 12), ("bun", 4),
("hugs", 5)
```

```
("h" "##u" "##g", 10), ("p" "##u" "##g", 5), ("p" "##u"
"##n", 12), ("b" "##u" "##n", 4), ("h" "##u" "##g"
"##s", 5)
```

```
("h" "##u" "##g", 10), ("p" "##u" "##g", 5), ("p" "##u"
"##n", 12), ("b" "##u" "##n", 4), ("h" "##u" "##gs", 5)
```

```
("hu" "##g", 10), ("p" "##u" "##g", 5), ("p" "##u"
"##n", 12), ("b" "##u" "##n", 4), ("hu" "##gs", 5)
```

```
("hug", 10), ("p" "##u" "##g", 5), ("p" "##u" "##n",
12), ("b" "##u" "##n", 4), ("hu" "##gs", 5)
```

1.1 Preprocessing – Text Tokenizer

Unigram Tokenizer

- 빈도수가 아닌 Subword에 대한 확률 모델 사용
- 손실(Loss)은 Subword가 어휘 사전에서 제거되었을 경우 코퍼스의 Likelihood 값의 감소 정도
- 전체 손실에 영향을 적게 미치는 10% ~ 20% 토큰을 제거, 원하는 사전 사이즈 될 때까지 반복
- https://arxiv.org/pdf/1804.10959.pdf (2018)
- 적용 사례: ALBERT, T5, mBART, XLNet

예제

```
("hug", 10), ("pug", 5), ("pun", 12), ("bun", 4),
("hugs", 5)
```

```
("h", 15) ("u", 36) ("g", 20) ("hu", 15) ("ug", 20)
("p", 17) ("pu", 17) ("n", 16) ("un", 16) ("b", 4)
("bu", 4) ("s", 5) ("hug", 15) ("gs", 5) ("ugs", 5) …
```

```
“pug”
[“p”, “u”, “g”] : score 0.00132
[“p”, “ug”] : score 0.0077
[“pu”, “g”] : score 0.0077
```

```
"hug“ : ["hug"] (score 0.071428)
"pug“ : ["pu", "g"] (score 0.007710)
"pun“ : ["pu", "n"] (score 0.006168)
"bun“ : ["bu", "n"] (score 0.001451)
"hugs": ["hug", "s"] (score 0.001701)
```

$-\log(P(word))$

[참고] SentencePiece (https://github.com/google/sentencepiece)

1.1 Preprocessing – Text Tokenizer

토큰화 알고리즘 비교

Model	BPE	WordPiece	Unigram
Training	Starts from a small vocabulary and learns rules to merge tokens	Starts from a small vocabulary and learns rules to merge tokens	Starts from a large vocabulary and learns rules to remove tokens
Training Step	Merges the tokens corresponding to **the most common pair**	Merges the tokens corresponding to **the pair with the best score based on the frequency of the pair**, privileging pairs where each individual token is less frequent	**Removes all the tokens in the vocabulary that will minimize the loss** computed on the whole corpus
Learns	**Merge rules and a vocabulary**	**Just a vocabulary**	**A vocabulary with a score for each token**
Encoding	Splits a word into characters and applies the merges learned during training	Finds the longest subword starting from the beginning that is in the vocabulary, then does the same for the rest of the word	Finds the most likely split into tokens, using the scores learned during training

Ref. https://huggingface.co/learn/nlp-course/chapter6

1.1 Preprocessing – Text Tokenizer

텍스트 처리 Tool

- **NLTK: Python-based Natural Language Toolkit (https://www.nltk.org/)**
- **KoNLPy: Korean NLP in Python (https://konlpy.org/ko/latest/index.html)**
  - ✓ 형태소 분석기: Okt(Open Korea Text), Mecab, Komoran, Hannanum, Kkma 지원
  - ✓ 형태소(Morpheme) 추출, 품사(POS: Part-Of-Speech) 태깅, 명사추출 등

아버지가 방에 들어가신다 → 아버지/Noun + 가/Josa + 방/Noun + 에/Josa + 들어가/Verb + 시ㄴ다/Eomi

```
!pip install konlpy

from konlpy.tag import Okt
tokenizer = Okt()

tokenizer.morphs(input_sentence)
tokenizer.pos(input_sentence)
tokenizer.nouns(input_sentence)
```

[실습] Text Tokenizer (1/2)

[실습 과제] Sub-Word Tokenizer 활용 실습!!

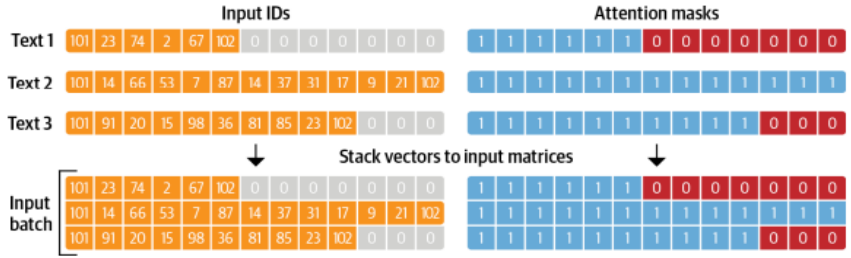
(aas_text_tokenizer.ipynb)

- BPE Algorithm
- Pretrained Tokenizer (WordPiece)
- Tokenizer Training (Python Code)

[실습] Text Tokenizer (2/2)

Pre-Trained Tokenizer

- AutoTokenizer.from_pretrained(model_ckpt)
- tokenizer.convert_ids_to_tokens()
- tokenizer.convert_tokens_to_string()
- tokenizer.vocab_size
- tokenizer.model_max_length
- tokenizer.model_input_names



Tokenizer Training

- Vocabulary Size
- DataSet: Tokenizer Training
- Text Normalization: tokenizer.backend_tokenizer.normalizer
- Iterator
- tokenizer.train_new_from_iterator(iterator, vocab_size, initial_alphabet)

- BertWordPieceTokenizer
- CharBPETokenizer
- ByteLevelBPETokenizer
- SentencePieceBPETokenizer

Ref. https://huggingface.co/docs/transformers/main_classes/tokenizer

1.2 NLP – Word Embedding

- 자연어처리에서,

  단어를 어떻게 표현할 것인가?

  문장을 어떻게 표현할 것인가?

  문서를 어떻게 표현할 것인가?

- 딥러닝 모델에서는 모든 것을 벡터로 표현

0.2	0.1	-0.3	0.03	1.4	-2.2	3.2	0.45	0.01	0.0

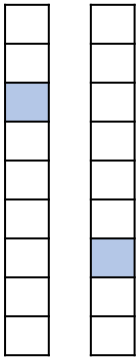
1.2 NLP – Word Embedding

Sparse Representation: One-Hot Encoding

- Vocabulary Size를 벡터의 차원으로 설정
- 벡터 중 한개 값만 1 로 표시, 나머지 값은 모두 Zero (N dimension vector for N words)
- 단어가 어떤 의미를 가지고 있는지를 표현하기에는 부적합
- Curse of Dimensionality

Time flies like an arrow

time	1	0	0	0	0
flies	0	1	0	0	0
like	0	0	1	0	0
an	0	0	0	1	0
arrow	0	0	0	0	1



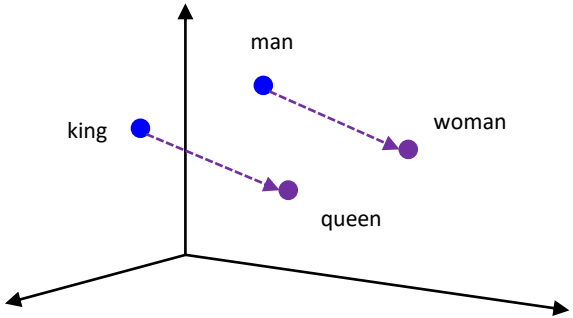
"N-개의 단어는 N-차원 벡터로 표현"

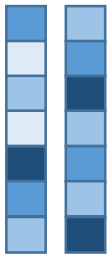
1.2 NLP – Word Embedding

Dense Representation

- 밀집된 차원의 벡터로 표현되며, 벡터값은 실수값을 가지게 됨
- 단어의 의미를 여러 차원에 분산하여 표현
- 분산표현은 "비슷한 문맥에서 등장하는 단어들은 비슷한 의미를 가진다"는 가정 하에 표현
- 단어 벡터 간의 유의미한 유사도 계산이 가능

※ 두 벡터간의 거리는 그들의 관련성을 측정 (가까운 거리는 높은 관련성, 먼 거리는 낮은 관련성)





"단어가 가지는 의미를 다차원 공간에 벡터화"

	On-Hot Vector	Embedding Vector
차 원	고차원 (사전 크기)	저차원 (특징)
Sparse/Dense	Sparse	Dense
표현 방법	수동	훈련 데이터 학습
벡터 값	0 or 1	실수

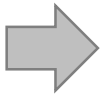
1.2 NLP – Word Embedding

- **Word Embedding:** 단어를 밀집 벡터의 형태로 표현하는 방법
  - Count Based: LSA, HAL
  - Prediction Based: CBOW, Skip-gram (Word2Vec, FastText)
  - Co-occurrence Statistics (GloVe)

$$x_{embedding} = x_{one-hot} * W_{embedding}$$

(vocabulary size X embedding_dimension)

word	one-hot (vocabulary size)					
time	1	0	0	0	0	…
flies	0	1	0	0	0	…
like	0	0	1	0	0	…
an	0	0	0	1	0	…
arrow	0	0	0	0	1	…



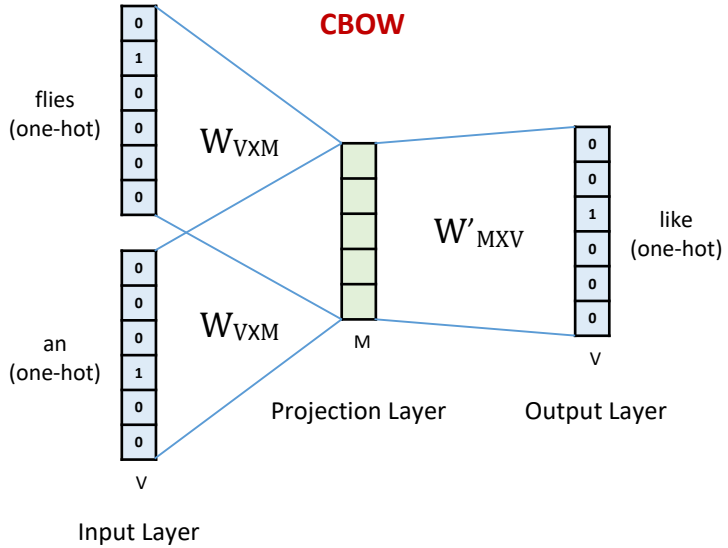
word	embedding dimension					
time	0.20	0.48	-0.01	0.19	0.33	…
flies	0.76	0.12	0.38	-0.36	0.11	…
like	0.29	0.41	0.08	0.15	-0.53	…
an	0.03	0.01	0.63	0.29	0.02	…
arrow	0.45	0.17	-0.21	0.36	0.12	…

* LSA(Latent Semantic Analysis), HAL(Hyperspace Analogue to Language)

1.2 NLP – Word Embedding

사전 훈련된 단어 임베딩

- Word2Vec (https://code.google.com/archive/p/word2vec/)
- FastText (https://github.com/facebookresearch/fastText)
- GloVe (https://github.com/stanfordnlp/GloVe)



1. CBOW (Continuous Bag Of Word)

- 주변 단어를 통해 주어진 단어를 예측
- 예) time flies __ an arrow

2. Skip-Gram

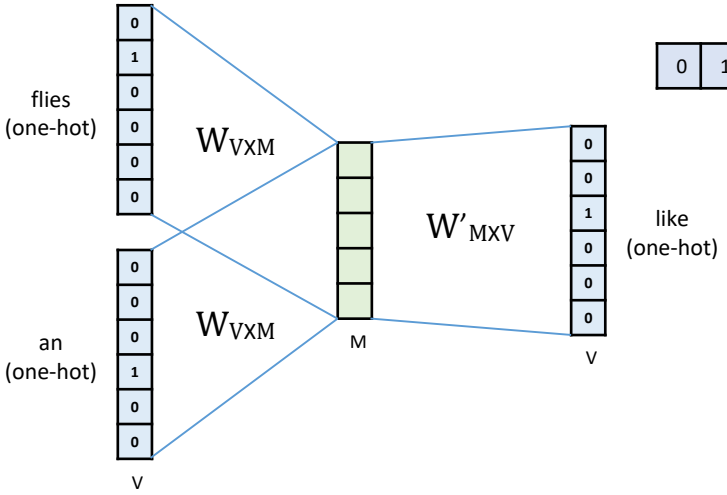
- 중심 단어에서 주변 단어를 예측하는 방식
- 예) __ __ like __ __

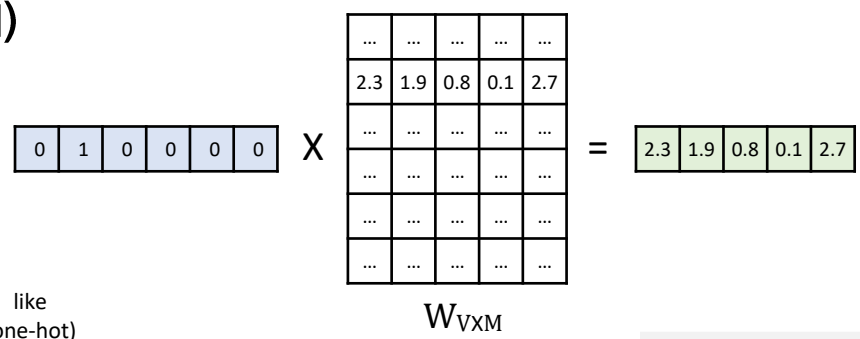
3. Co-occurrence

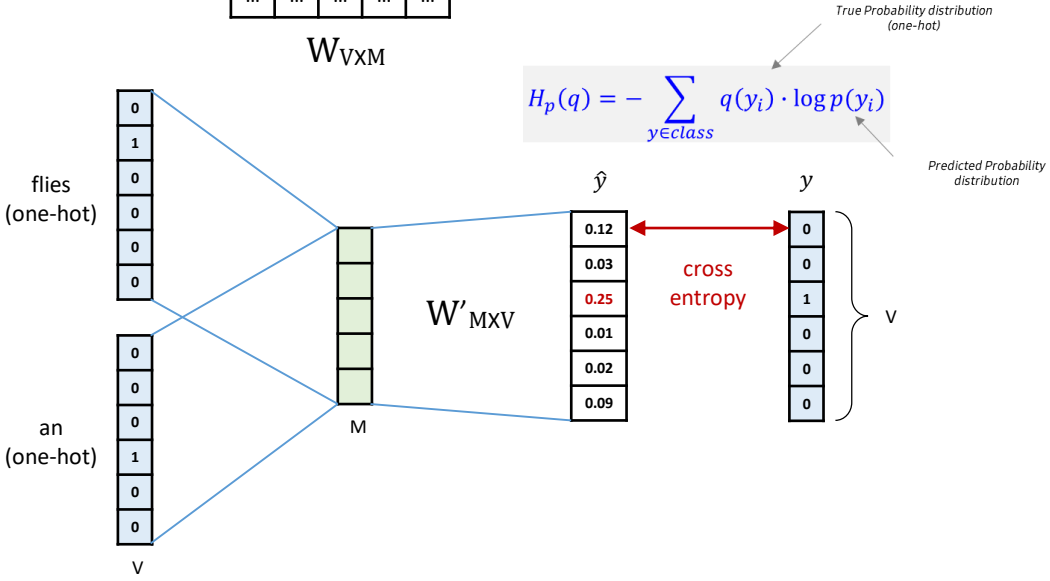
- 단어-단어의 동시 출연 빈도를 이용한 학습

1.2 NLP – Word Embedding

CBOW (Continuous Bag Of Word)





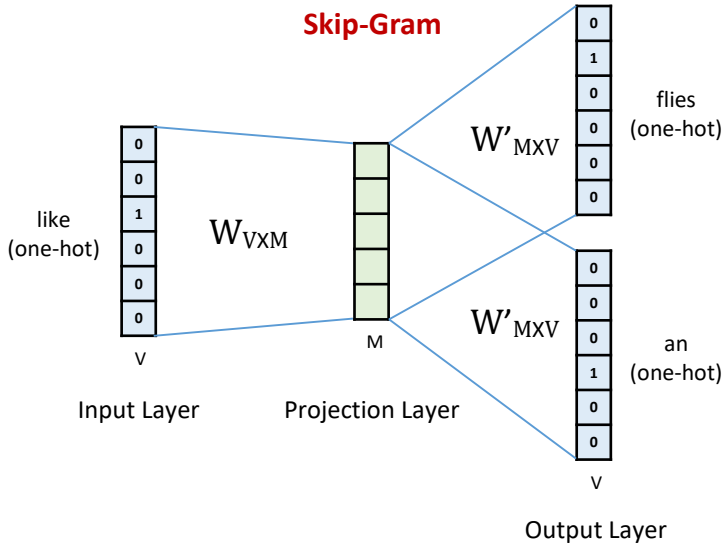


※ Prediction 시 좌우 컨텍스트 단어에 W Matrix 곱한 후 윈도우 사이즈로 나눈 평균값 사용

1.2 NLP – Word Embedding

사전 훈련된 단어 임베딩

- Word2Vec (https://code.google.com/archive/p/word2vec/)
- FastText (https://github.com/facebookresearch/fastText)
- GloVe (https://github.com/stanfordnlp/GloVe)



1. CBOW (Continuous Bag Of Word)

   - 주변 단어를 통해 주어진 단어를 예측
   - 예) time flies __ an arrow

2. Skip-Gram

   - 중심 단어에서 주변 단어를 예측하는 방식
   - 예) __ __ like __ __

3. Co-occurrence

   - 단어-단어의 동시 출연 빈도를 이용한 학습

1.2 NLP – Word Embedding

사전 훈련된 단어 임베딩

- Word2Vec (https://code.google.com/archive/p/word2vec/)
- FastText (https://github.com/facebookresearch/fastText)
- GloVe (https://github.com/stanfordnlp/GloVe)

✓ 동시 등장 확률 P(k|i) 이용

✓ 임베딩된 중심 단어와 주변 단어 벡터의 내적이 전체 코퍼스에서의 동시 등장 확률이 되도록 만드는 방법

$$dot\ product(w_i\ \tilde{w}_k) \approx log\ P(k \mid i) = log\ P_{ik}$$

Cooccurrence Probability

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

1. CBOW (Continuous Bag Of Word)
   - 주변 단어를 통해 주어진 단어를 예측
   - 예) time flies __ an arrow
2. Skip-Gram
   - 중심 단어에서 주변 단어를 예측하는 방식
   - 예) __ __ like __ __
3. Co-occurrence
   - 단어-단어의 동시 출연 빈도를 이용한 학습

1.2 NLP – Word Embedding

Word Embedding 구현 (with Gensim Python Library)

- 자연어 처리를 위한 오픈 소스 Python 라이브러리
- 텍스트에서 새로운 단어 벡터를 학습하기위한 Word2Vec 단어 임베딩 구현을 지원

```
class gensim.models.word2vec.Word2Vec(sentences=None, corpus_file=None,
          vector_size=100, alpha=0.025, window=5, min_count=5, max_vocab_size=None, sample=0.001,
          seed=1, workers=3, min_alpha=0.0001, sg=0, hs=0, negative=5,  ns_exponent=0.75,
          cbow_mean=1, hashfxn=<built-in function hash>, epochs=5, null_word=0,
          trim_rule=None, sorted_vocab=1, batch_words=10000, compute_loss=False,
          callbacks=(), comment=None, max_final_vocab=None, shrink_windows=True)
```

```
class gensim.models.fasttext.FastText(sentences=None, corpus_file=None,
          vector_size=100, alpha=0.025, window=5, min_count=5, max_vocab_size=None, sample=0.001,
          seed=1, workers=3, word_ngrams=1, min_alpha=0.0001, sg=0, hs=0, negative=5, ns_exponent=0.75,
          cbow_mean=1, hashfxn=<built-in function hash>, epochs=5, null_word=0, min_n=3, max_n=6,
          sorted_vocab=1, bucket=2000000, trim_rule=None, batch_words=10000, callbacks=(),
          max_final_vocab=None, shrink_windows=True)
```

sg (*{1, 0}, optional*) – Training algorithm: skip-gram if *sg=1*, otherwise CBOW.

1.2 NLP – Word Embedding

LLM Embeddings

Traditional Vector Representation
On-Hot Encoding, BoW (Bag-of-Words)

Word Embeddings
Word2Vec, FastText, GloVe

Contextual Embeddings
Transformer-based Models (ELMo, BERT, GPT)

1.3 Language Modeling

언어 모델 (Language Model)

Probability distribution over sequences of words

* 단어들의 조합이 얼마나 적절한지, 해당 문장이 얼마나 적합한지를 알려주는 모델

예) 이전 단어들이 주어졌을 때 다음 단어를 예측하는 언어 모델

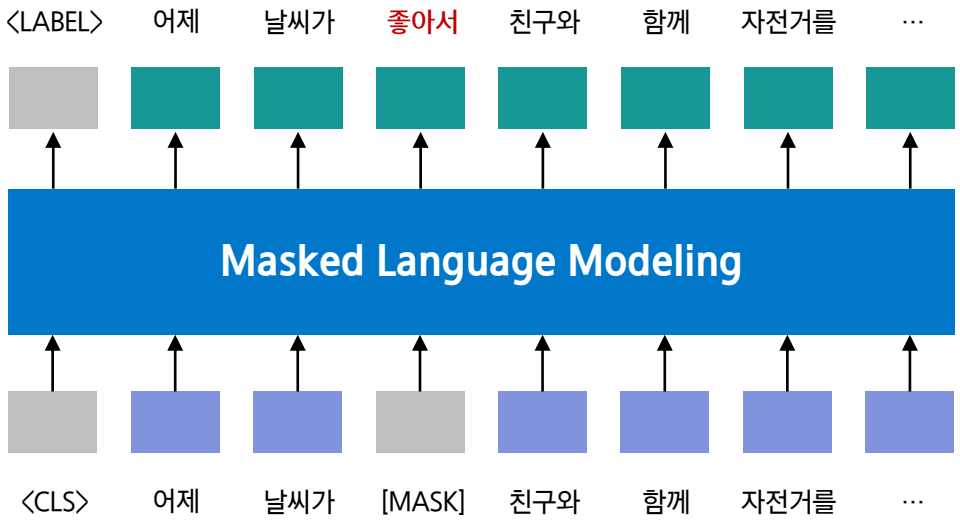
"어제 날씨가 좋아서 친구와 함께 자전거를 _____"

탔다(0.8)　　사러(0.03)　　오렌지(0.000002)　　먹으러(0.000001)

Language Model을 어떻게 만들 수 있을까?

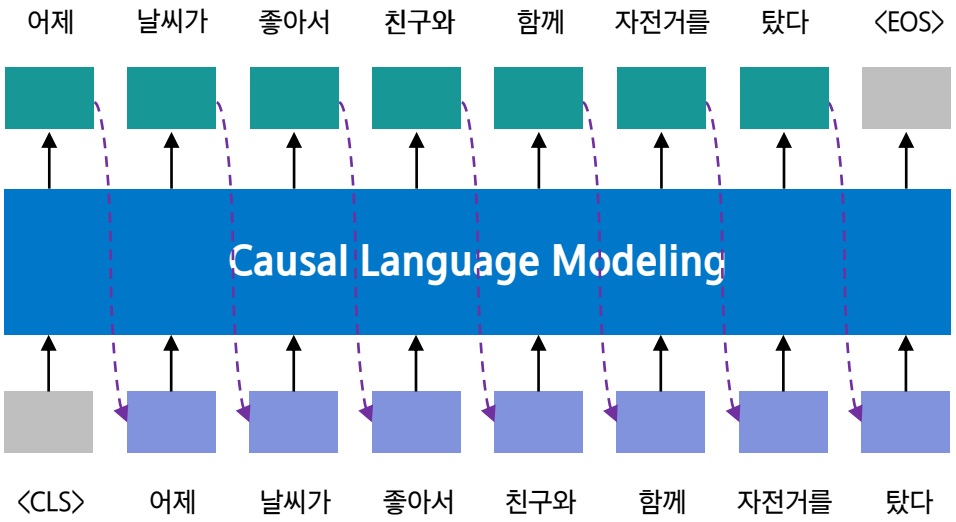
1.3 Language Modeling

MLM (Masked Language Model)



1.3 Language Modeling

CLM (Causal Language Model)

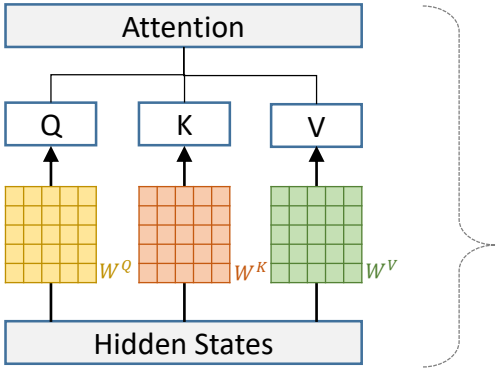


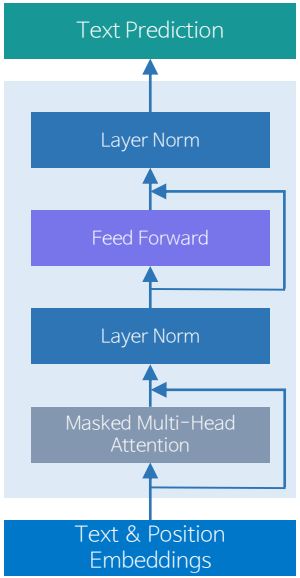
1.3 Language Modeling

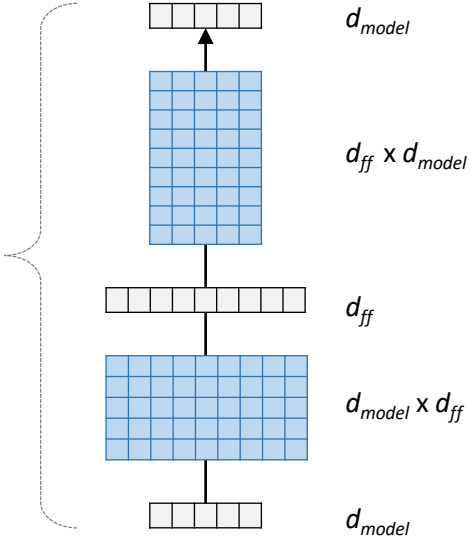
Transformer Architecture

(1) Multi-Head Attention

$$Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_K}}\right)V$$



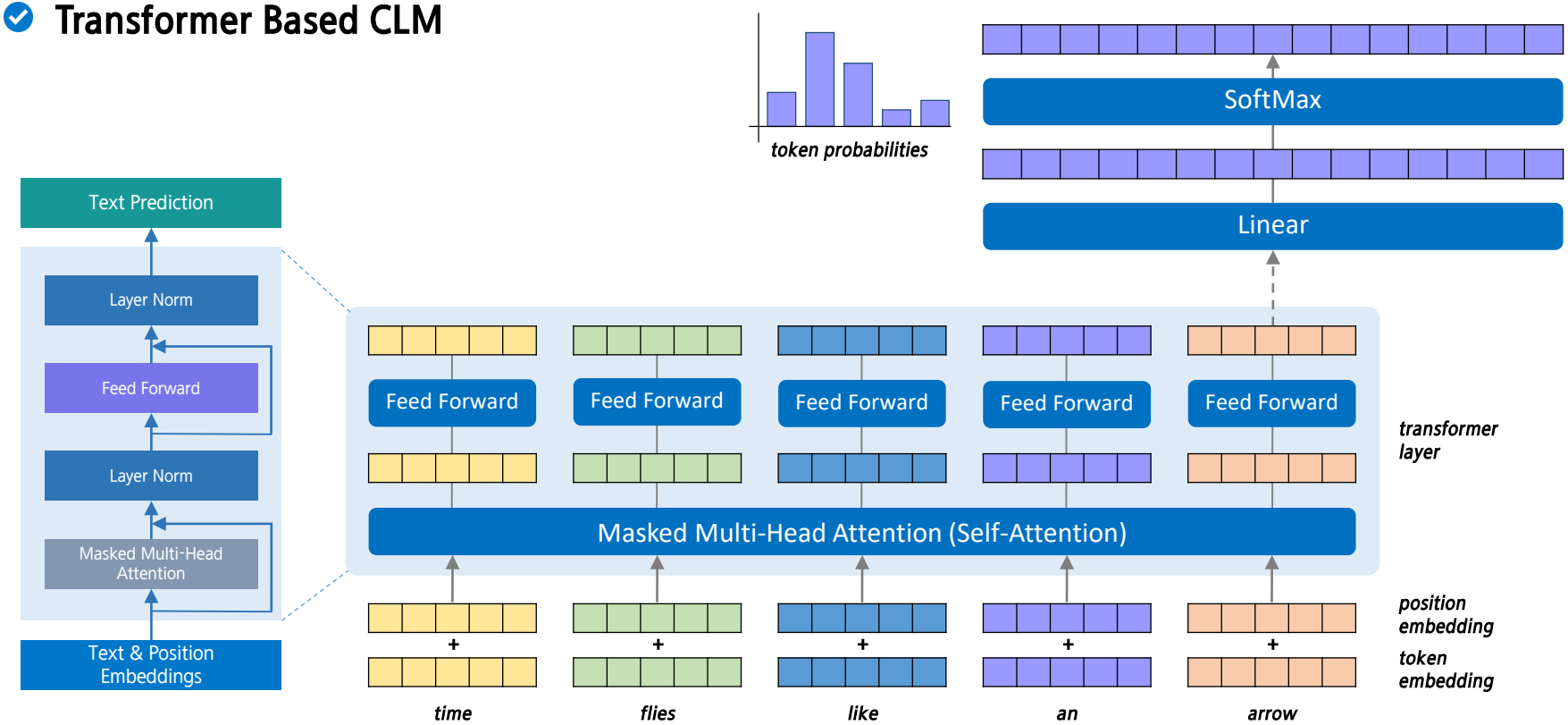




(2) Feed Forward

1.3 Language Modeling

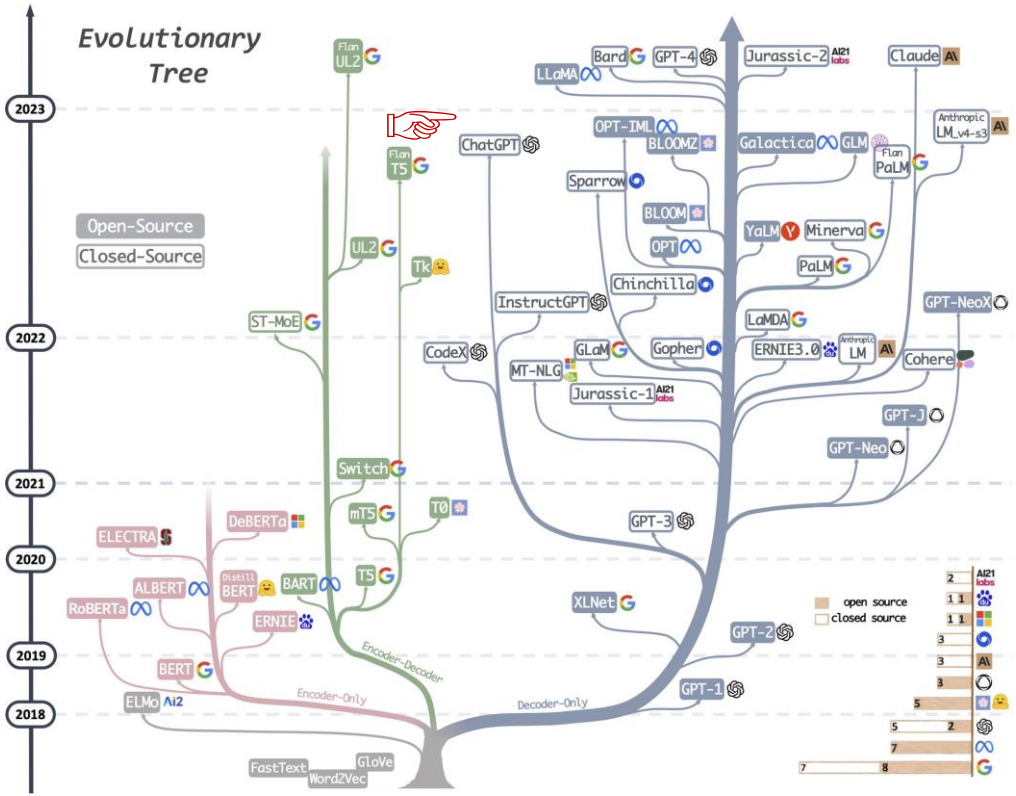
Transformer Based CLM



1.3 Language Modeling

TYPE	MODEL
Encoder Only	**BERT**
	DistilBERT
	RoBERTa
	ALBERT
	DeBERTa
Decoder Only	**GPT**
	GPT-2
	GPT-3
	LLaMA
	Bard
Encoder-Decoder	T5
	BART

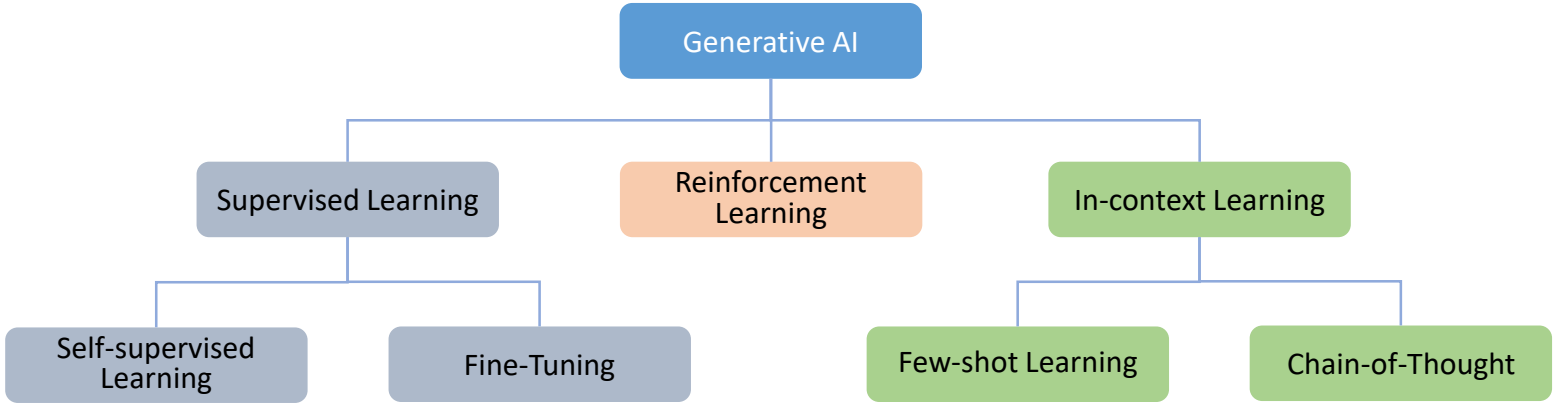
Global Large Language Models in 2024-2025



Ref. Yann LeCun, LLMs evolutionary tree.

1.3 Generative AI

주요 학습 기술



요약

- Tokenizer: BPE, WordPiece, Unigram
- Word Embedding: CBOW, Skip-Gram, Co-occurrence
- Language Modeling: MLM, CLM
- Transformer Based CLM

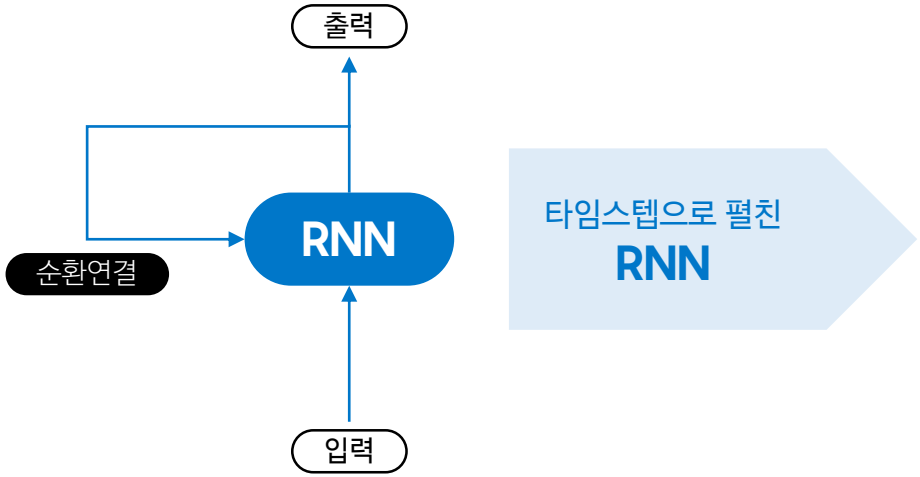
CONTENTS

SEU
Samsung Electronics University

2.0 RNN Attention

RNN (Recurrent Neural Network)

- 순환적인 구조를 특징으로 하는 신경망의 한 종류
- 문장과 같은 시퀀스 데이터나 시계열 데이터를 모델링하는데 적합
- Vanilla RNN을 선택적으로 정보를 기억할 수 있도록 개선: LSTM, GRU

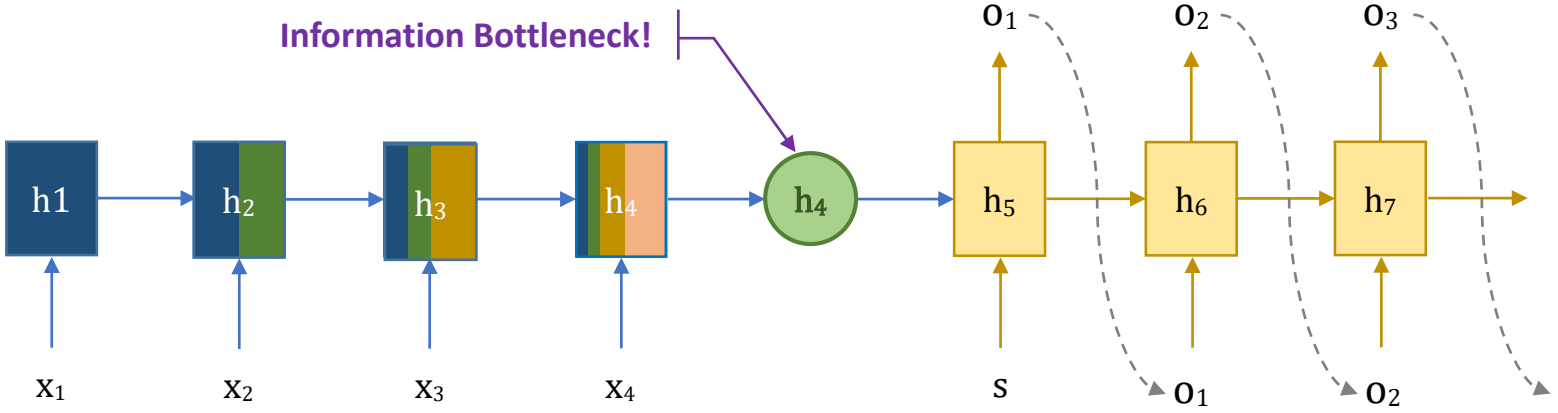


Structure of Perceptron

2.0 RNN Attention

RNN Encoder-Decoder

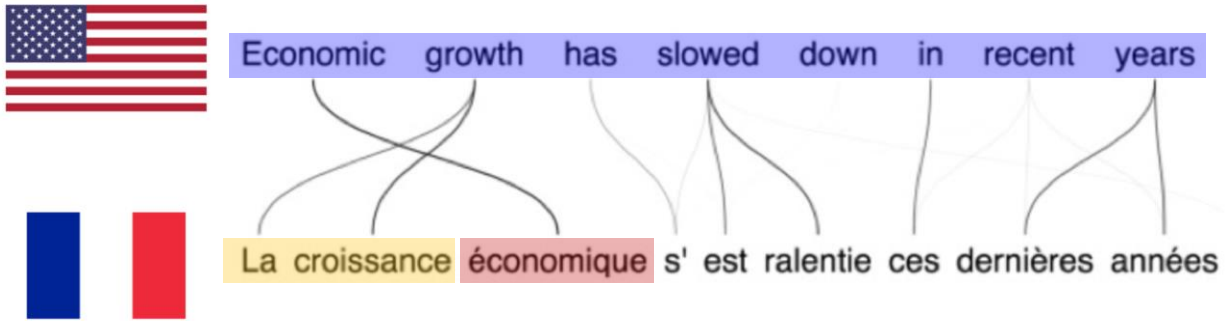
- What if the input sentence is long, can a single vector from the encoder hold all the relevant information to provide to the decoder?
- Is it possible to focus on a few relevant words in a sentence when predicting the target word rather than a single vector holding the information about the entire sentence?



2.0 RNN Attention

Observation

- At every step, all the inputs are not equally useful
- Inputs relevant to the context may be more useful



Attention Model

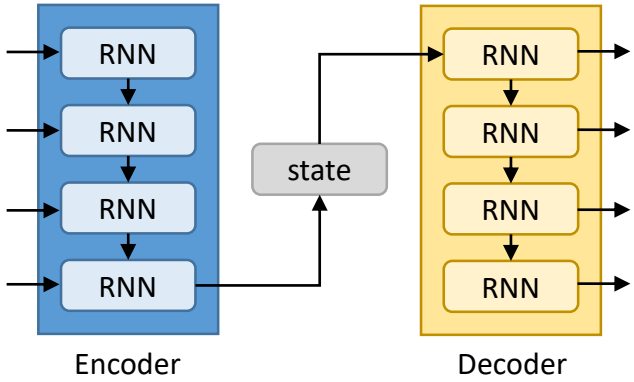
- The basic idea is to avoid attempting to learn a single vector representation for each sentence, instead, it pays attention to specific input vectors of the input sequence based on the attention weights

[출처] Kyunghyun Cho, "Introduction to Neural Machine Translation with GPUs" (2015)

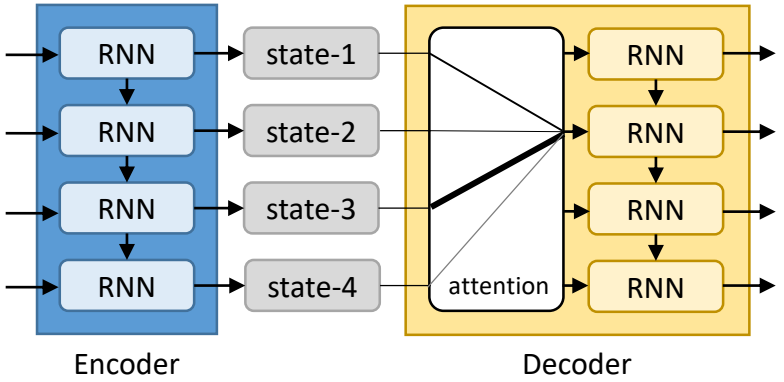
2.0 RNN Attention

RNN Attention

- The basic idea is to avoid attempting to learn a single vector representation for each sentence, instead, it pays attention to specific input vectors of the input sequence based on the attention weights
- Create a separate context vector for each output



RNN based Encoder-Decoder

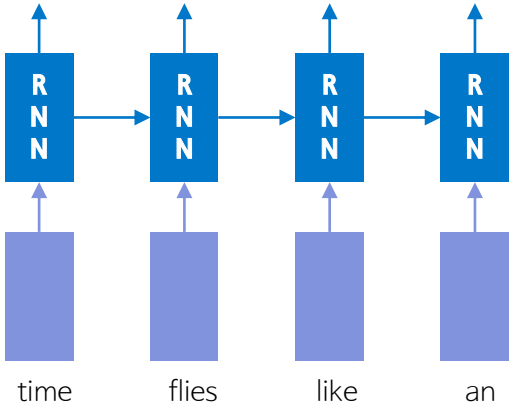


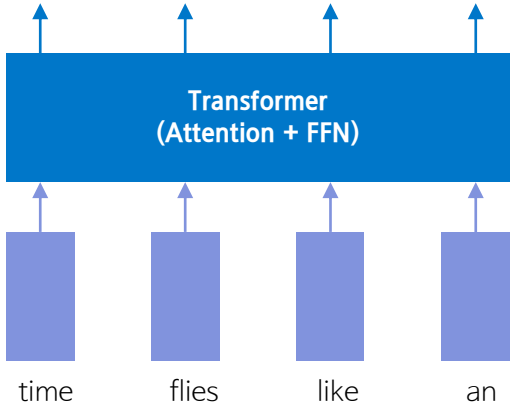
RNN-Attention based Encoder-Decoder

Ref. 트랜스포머를 활용한 자연어처리

2.1 Transformer Basics

- 트랜스포머의 핵심은 어텐션(Attention)과 병렬화 가능한 아키텍처





RNN is hard to parallelize and not time-efficient !

2.1 Transformer Basics

대규모 언어모델의 핵심이 되는 논문 - Transformer (NIPS 2017)

Attention Is All You Need

Ashish Vaswani* Google Brain avaswani@google.com

Noam Shazeer* Google Brain noam@google.com

Niki Parmar* Google Research nikip@google.com

Jakob Uszkoreit* Google Research usz@google.com

Llion Jones* Google Research llion@google.com

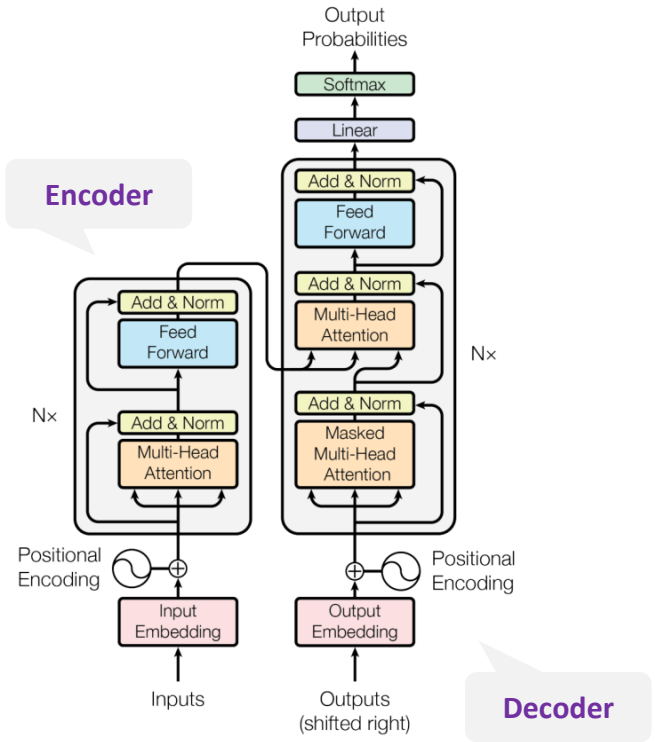
Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu

Łukasz Kaiser* Google Brain lukaszkaiser@google.com

Illia Polosukhin* ‡ illia.polosukhin@gmail.com

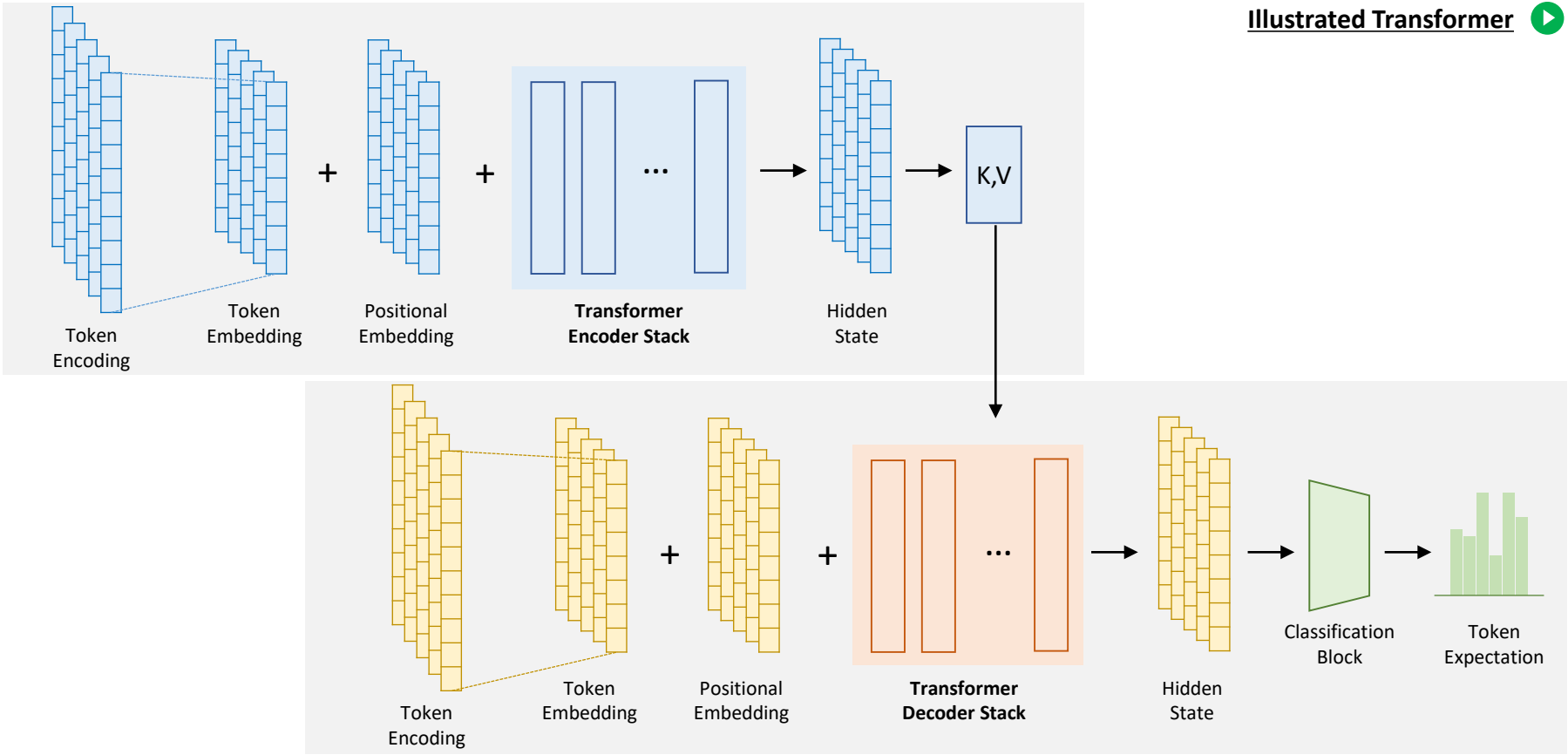
Abstract

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 41.0 after training for 3.5 days on eight GPUs, a small fraction of the training costs of the best models from the literature.



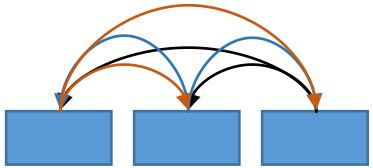
2.1 Transformer Encoder-Decoder Architecture

Illustrated Transformer

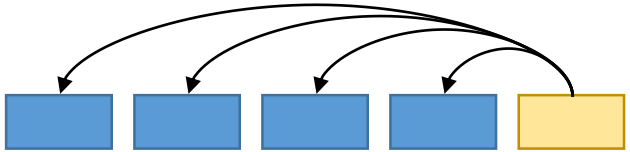


2.1 Attention

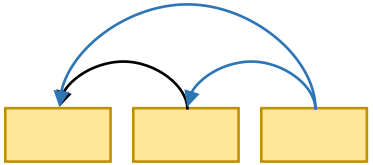
- 문장내 단어들간의 연관성과 중요성에 대해 효과적으로 모델링할 수 있는 프레임워크
- 단어 간의 관계를 측정한 값을 Attention Score, 하나의 테이블로 만든 것을 Attention Map
- Encoder Self-Attention, Encoder-Decoder Attention, Decoder Self-Attention



Encoder **Self-Attention**
(Bidirectional Attention)



Encoder-Decoder Attention

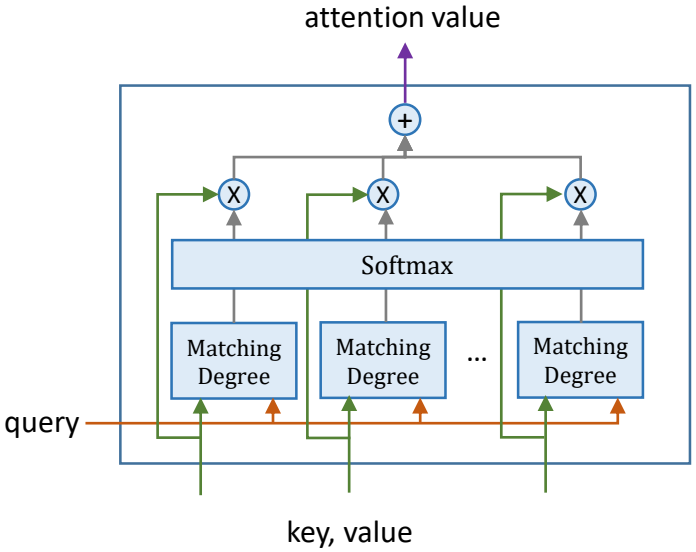


Decoder **Self-Attention**
(Causal/Autoregressive Attention)

2.1 Attention Mechanism (1/3)

Attention(Query, Key, Value) 개념 설명

$$Z = Attention(Q, K, V) = softmax(QK^T)V$$

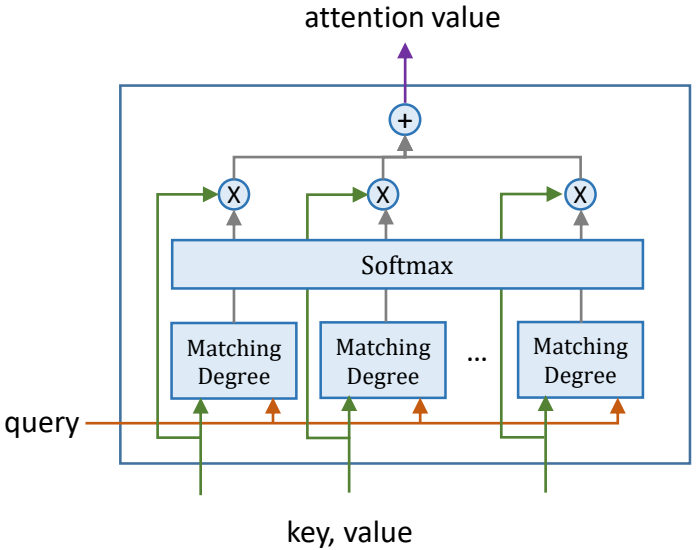


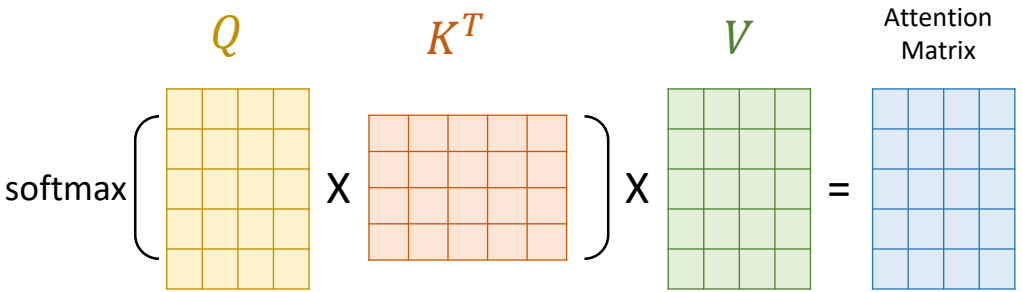
	k_0	k_1	k_2	k_3	k_4
q_0					
q_1					
q_2					
q_3					
q_4					
q_5					

2.1 Attention Mechanism (2/3)

Basic Dot Product Attention

$$Z = Attention(Q, K, V) = softmax(QK^T)V$$





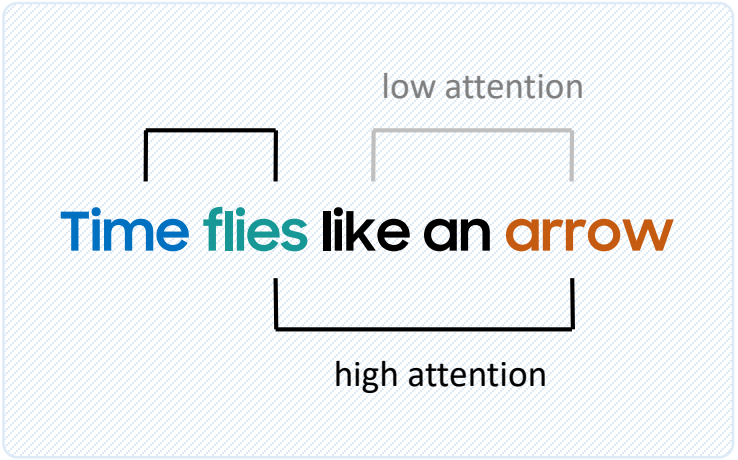
* $Q, K, V : seq\_len \; X \; d_{model}$

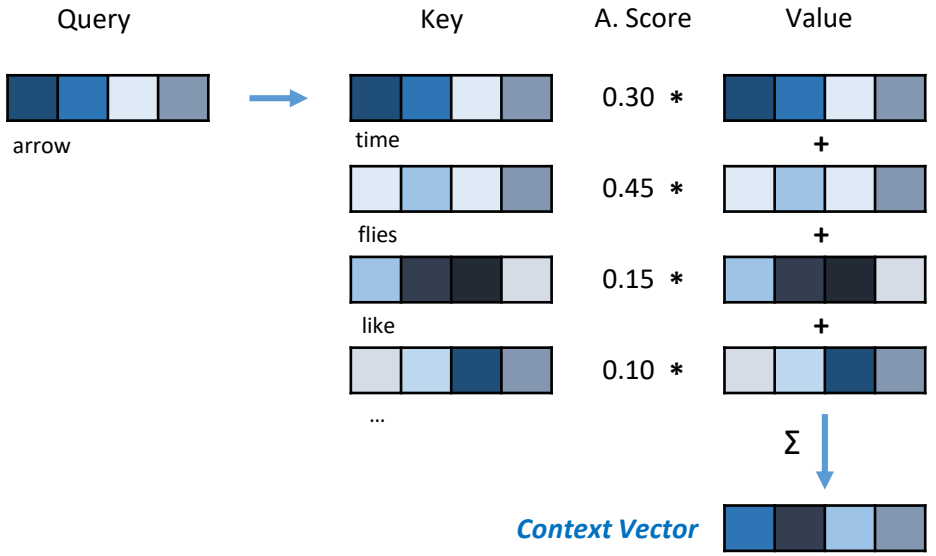
2.1 Attention Mechanism (3/3)

Basic Dot-Product Attention

Scaled Dot-Product Attention

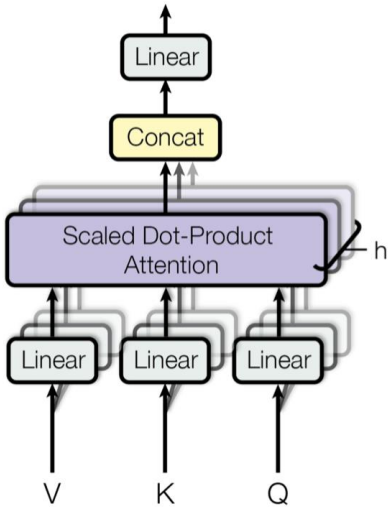
$$Z = Attention(Q, K, V) = softmax(QK^T)V$$

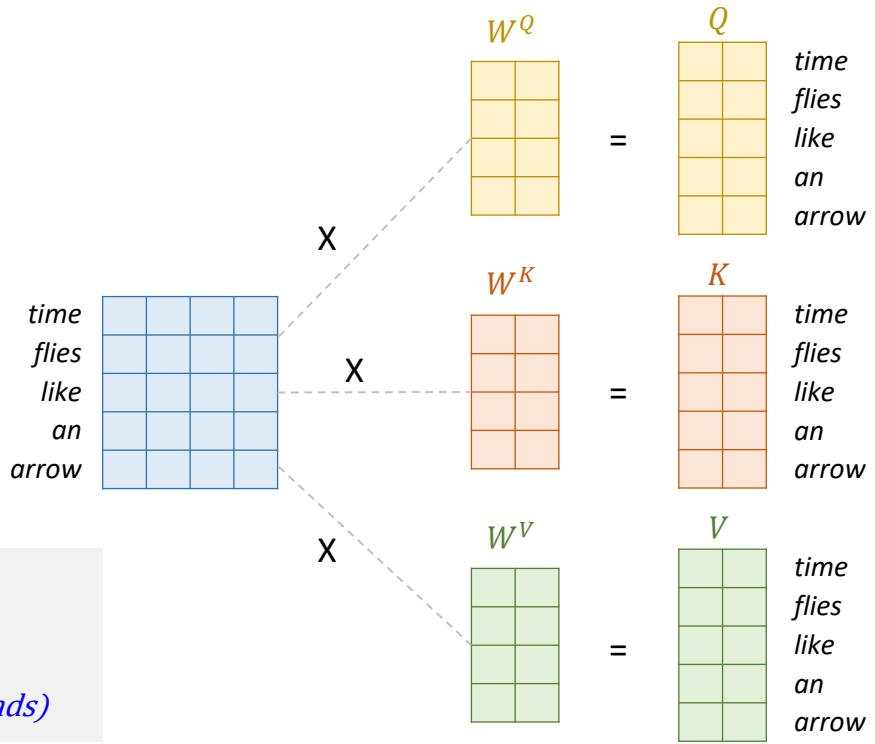




2.1 Multi-Head Attention (1/3)

Q, K, V 벡터 생성: 임베딩에 독립적인 선형변환 적용 (CNN Filter 개념과 유사*)



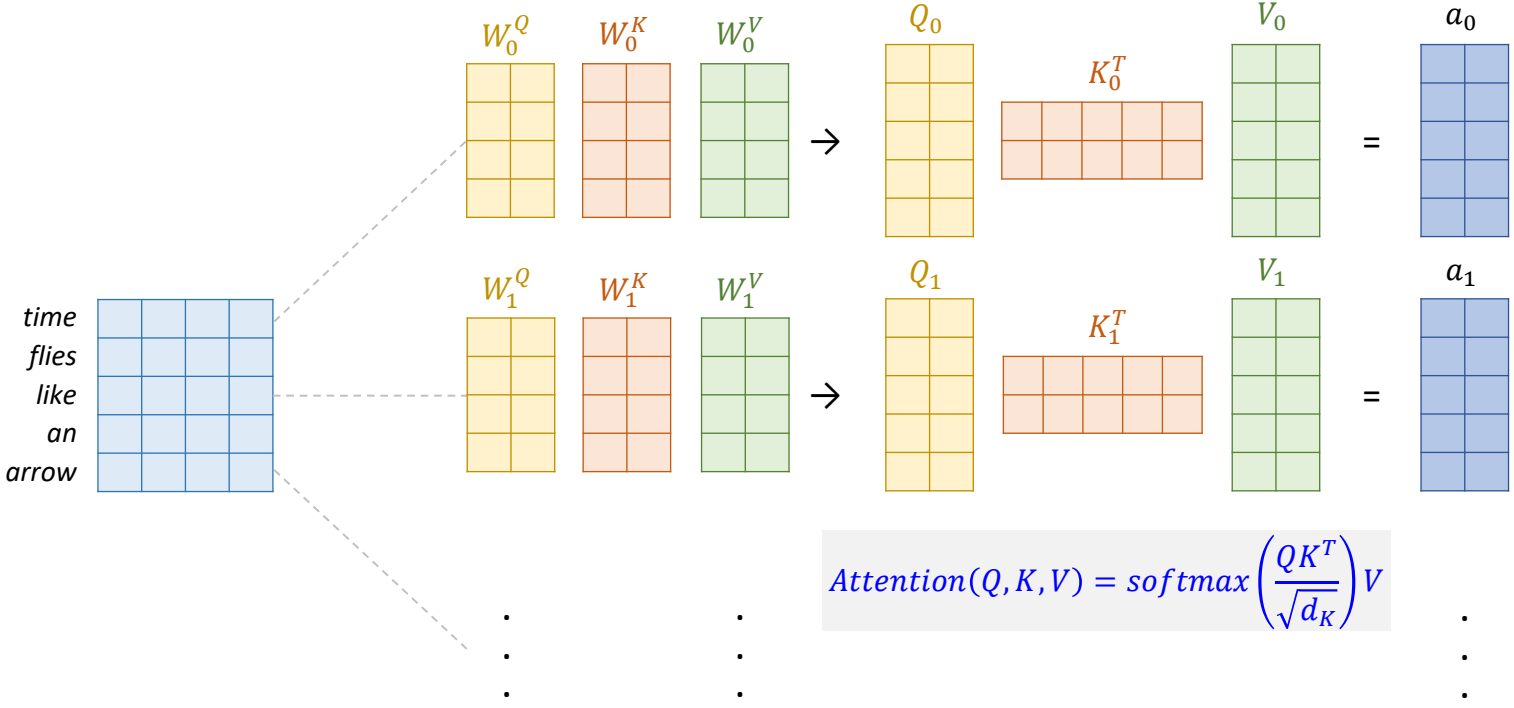


- d_{model}
- num_heads
- d_{model} X (d_{model}/num_heads)

* 멀티헤드를 통해 모델이 동시에 여러 측면에 초점

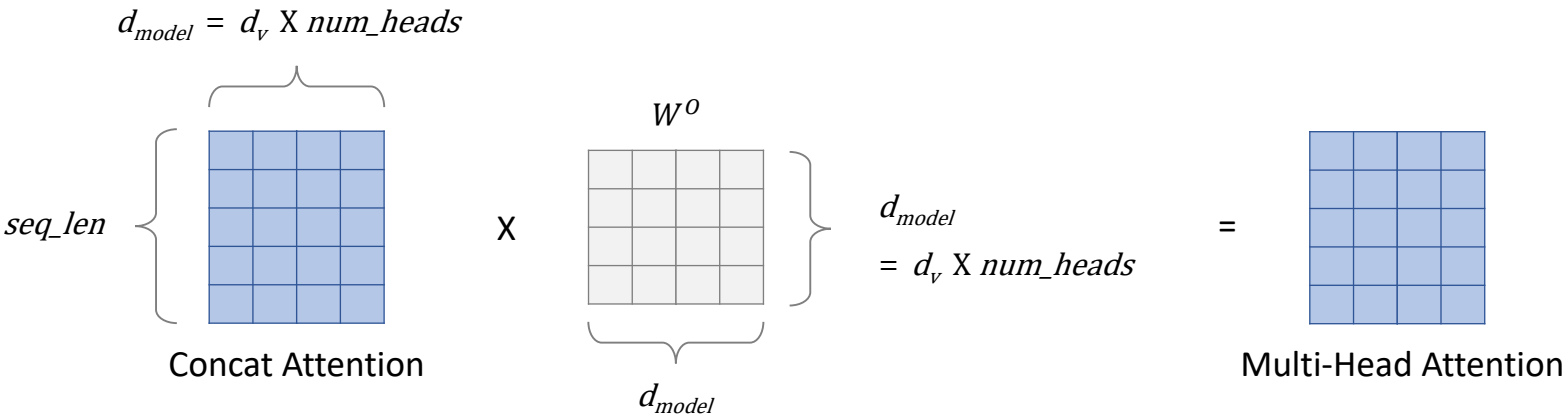
2.1 Multi-Head Attention (2/3)

Scaled Dot-Product Attention & Concat



2.1 Multi-Head Attention (3/3)

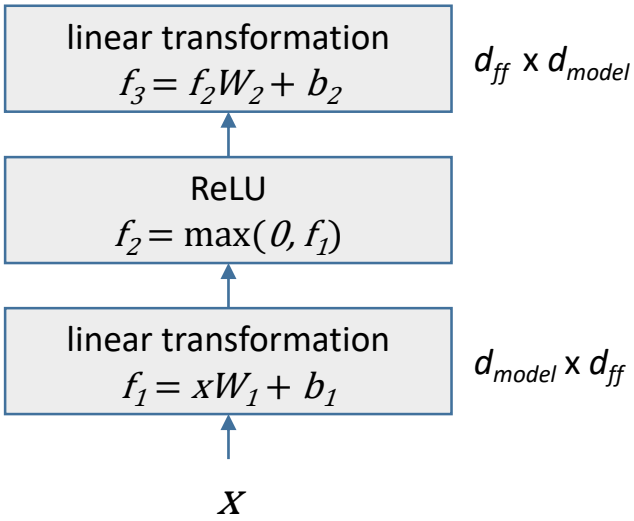
- 선형변환 (Linear): W^O

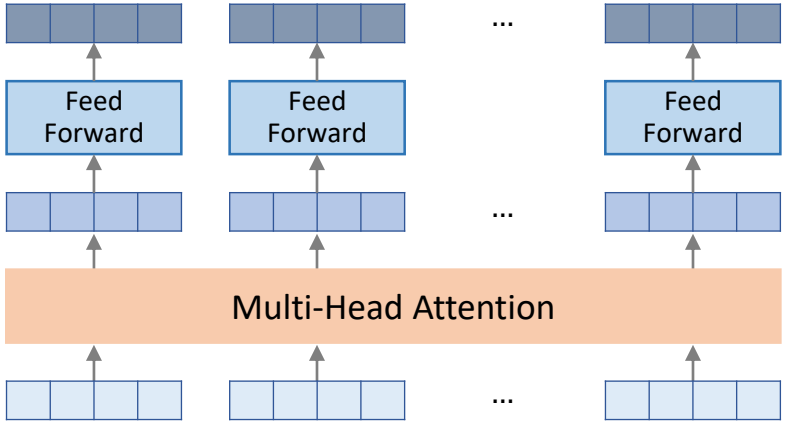


2.1 Feed Forward

Position-wise Feedforward Network

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

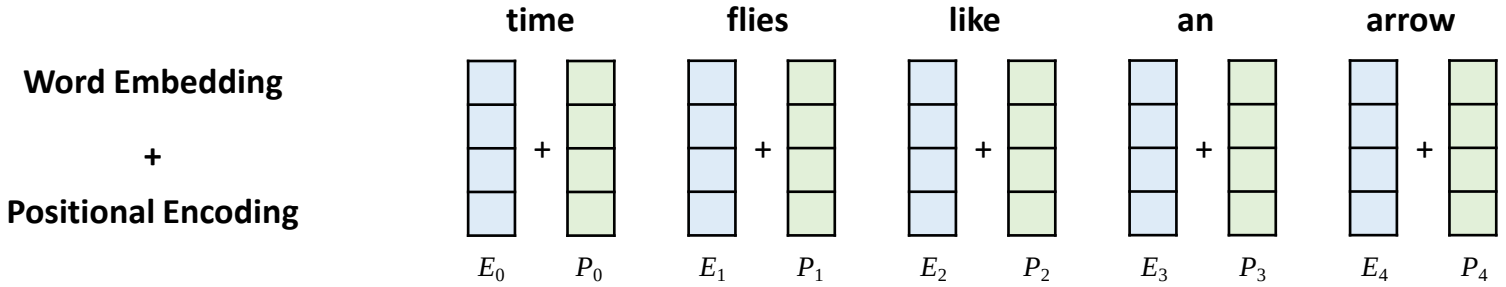




2.1 Positional Encoding (1/2)

Positional Encoding의 필요성

- Transformer 모델은 입력 데이터를 병렬 처리하면서 입력된 순서에 대한 정보가 사라지는 문제 발생
- 자연어의 경우 단어의 위치에 따라 의미가 달라지기 때문에 입력에 위치 정보를 추가하는 방법이 필요



Positional Encoding의 필요조건

- 각 위치값은 시퀀스의 길이나 입력값에 관계없이 동일한 위치값을 가져야 함
- 모든 위치값이 입력값에 비해 너무 크면 안됨 (데이터가 가지는 의미를 잃어버릴 수 있음)
- 위치에 따라 서로 다른 값을 가져야 함

2.1 Positional Encoding (2/2)

Sine & Cosine 함수 기반 Positional Encoding

RoPE

- -1 ~ 1 사이를 반복하는 주기 함수
- 서로 다른 주기의 Sine & Cosine 함수를 결합하여 사용

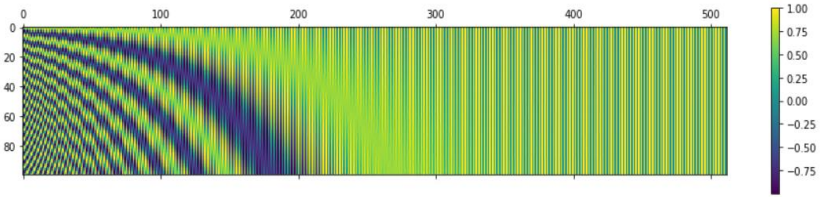
$$\mathrm{PE}_{(\mathrm{pos},2i)} = \sin\left(\frac{\mathrm{pos}}{10000^{2i/d_{\mathrm{model}}}}\right)$$

$$\mathrm{PE}_{(\mathrm{pos},2i+1)} = \cos\left(\frac{\mathrm{pos}}{10000^{2i/d_{\mathrm{model}}}}\right)$$

$$0 \le i \le d_{\mathrm{model}}/2$$

$$\mathrm{PE} = \begin{bmatrix} \mathrm{PE}_{(0,0)} & \mathrm{PE}_{(0,1)} & \cdots \mathrm{PE}_{(0,d_{\mathrm{model}}-1)} \\ \vdots & \ddots & \vdots \\ \mathrm{PE}_{(n-1,0)} & \mathrm{PE}_{(n-1,1)} & \cdots \mathrm{PE}_{(n-1,d_{\mathrm{model}}-1)} \end{bmatrix}$$

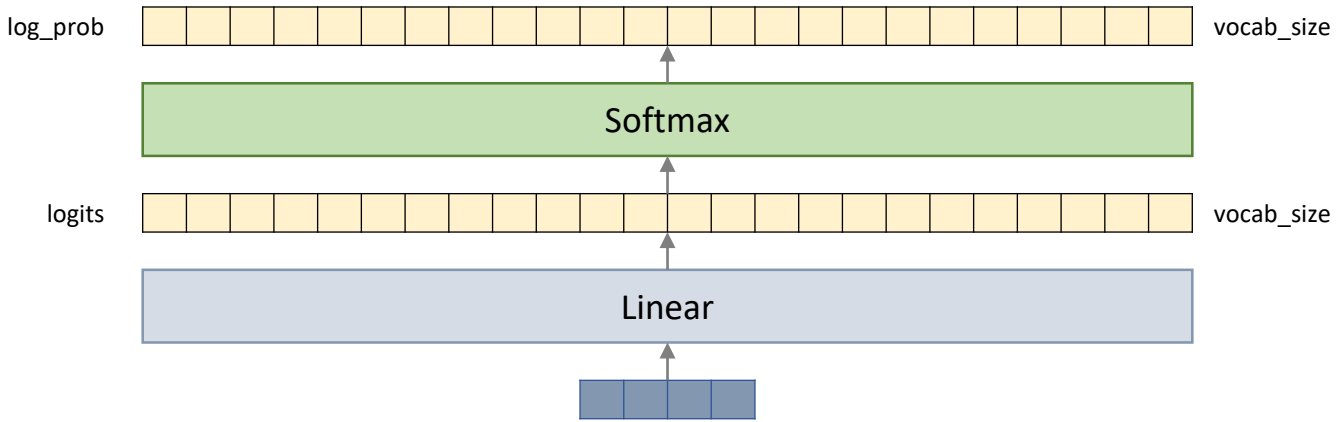
```python
import numpy as np
import matplotlib.pyplot as plt

def getPositionEncoding(seq_len, d, n=10000):
    P = np.zeros((seq_len, d))
    for k in range(seq_len):
        for i in np.arange(int(d/2)):
            denominator = np.power(n, 2*i/d)
            P[k, 2*i] = np.sin(k/denominator)
            P[k, 2*i+1] = np.cos(k/denominator)
    return P

P = getPositionEncoding(seq_len=100, d=512, n=10000)
cax = plt.matshow(P)
plt.gcf().colorbar(cax)
```



2.1 Linear & Softmax Layer

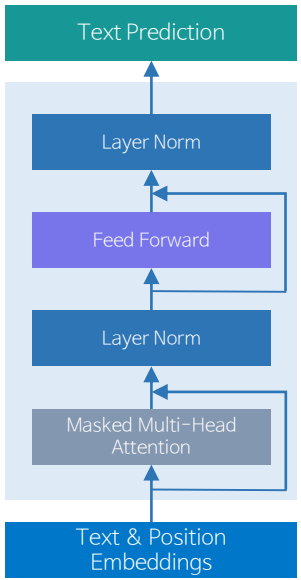
- **Linear Layer**
  - Fully connected neural network
  - projects the vector produced by the stack of decoders into a logits vector
- **Softmax Layer**
  - turns those scores into probabilities

$$softmax(y_i) = \frac{e^{y_i}}{\sum_j e^{y_i}}$$



2.1 Transformer Based CLM

Transformer Based CLM



1. Text & Position Embeddings

$$h^{(0)} = Embed(x) + PosEnc(x)$$

2. Masked Multi-Head Attention

$$MaskedAttention(Q,K,V) = softmax\left(\frac{QK^T}{\sqrt{d_K}} + M\right)V$$

$$Head_i = MaskedAttention(QW_i^Q, KW_i^K, VW_i^V)$$

$$MaskedMultiHead(Q,K,V) = Concat(Head_i, \dots, Head_h)W^o$$

3. Residual + Layer Norm

$$h^{(1)} = LayerNorm(x + MaskedMultiHead(x,x,x))$$

4. Position-wise Feedforward Network

$$FFN(x) = ReLU(xW_1 + b_1)W_2 + b_2$$

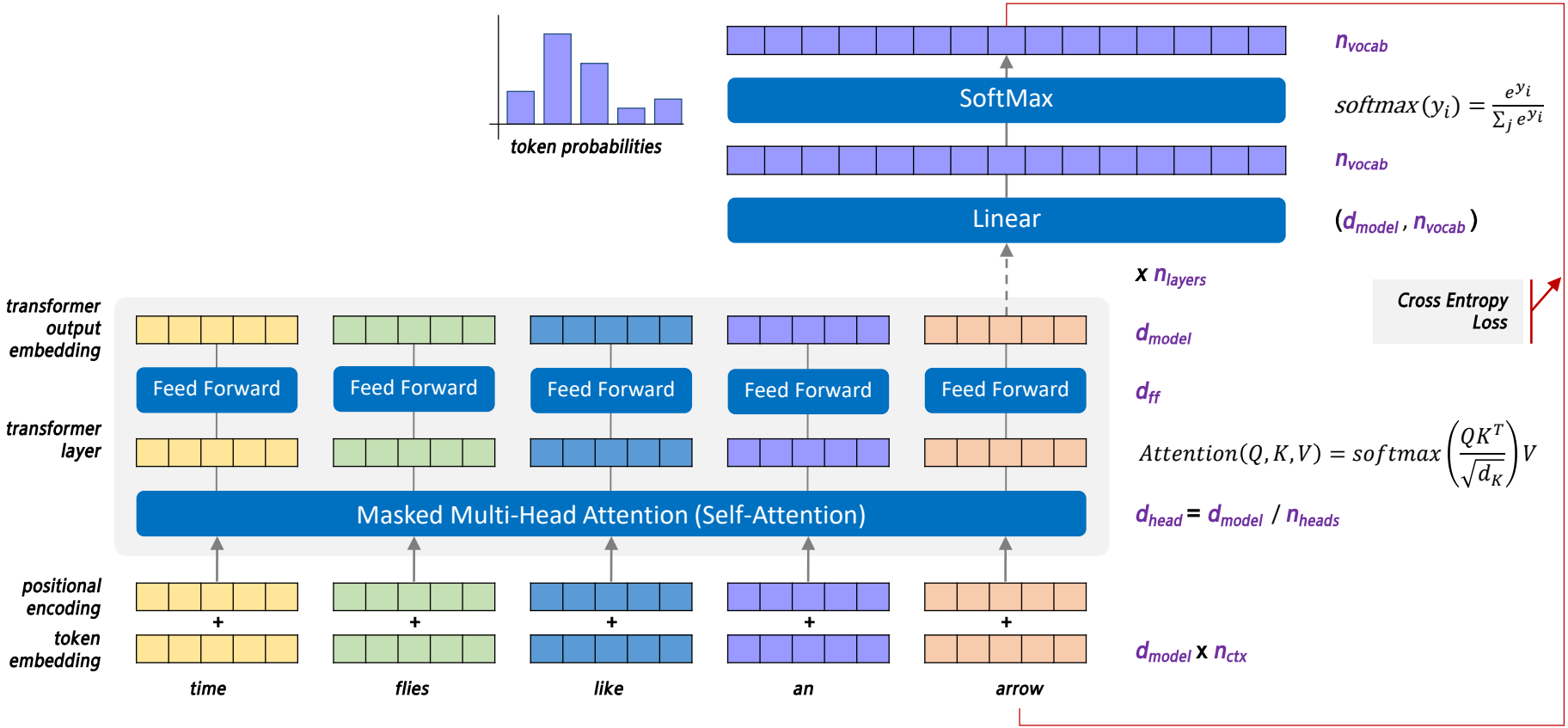
5. Residual + Layer Norm

6. Logits & Softmax (Text Prediction)

$$logit\ s_t = h_t^{(L)}W_E^T + b$$

$$P(x_{t+1}|x_{\le t}) = softmax(logit\ s_t)$$

2.1 Transformer Based CLM



※ Layer Normalization & Skip Connection: Multi-Head Attention과 Feed Forward의 전/후에 배치

[실습] Transformer Architecture (1/2)

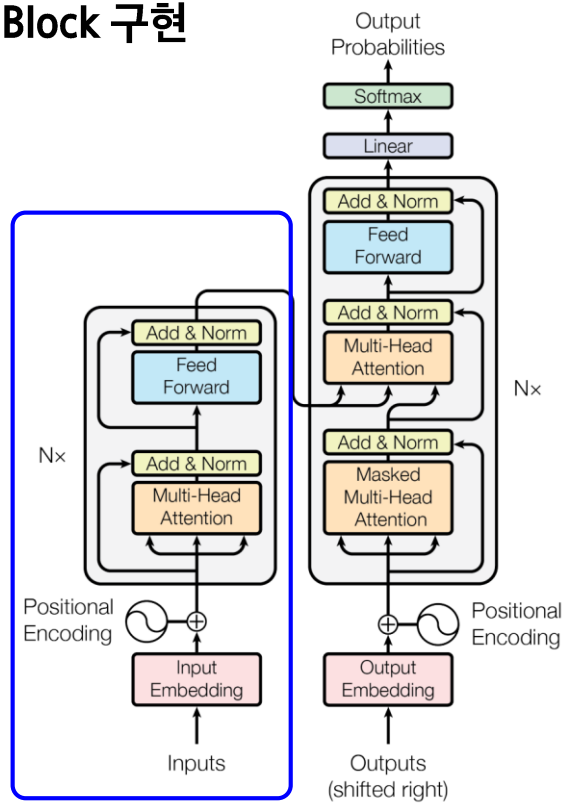
[실습 과제] Transformer Encoder 및 Sequence Classification 구축!!

(aas_transformer_architecture.ipynb)

- Transformer Attention Viewer
- Transformer Encoder Block
  (Multi-Head Attention, Token/Position Embedding, Feed Forward)
- Sequence Classification Task Head

[실습] Transformer Architecture (2/2)

Encoder Block 구현

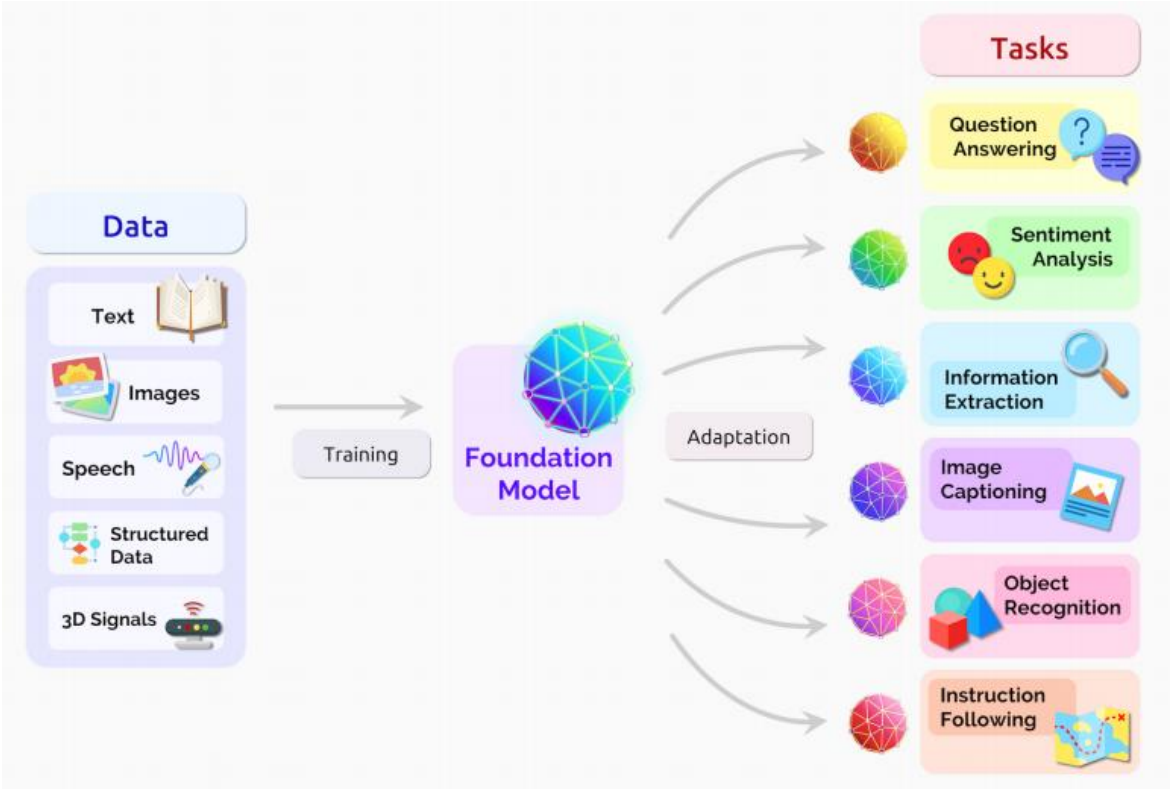


주요 Block

- Input Embedding
- Position Embedding (Positional Encoding)
- Scaled Dot Product Attention
- Multi-Head Attention
- Feed Forward
- Layer Normalization
- Skip Connection
- Transformer Encoder Layer
- Transformer Encoder

2.2 LLM (Large Language Model)

- Transformer
- Foundation Model
- Self-supervised Learning
- Pre-trained Model



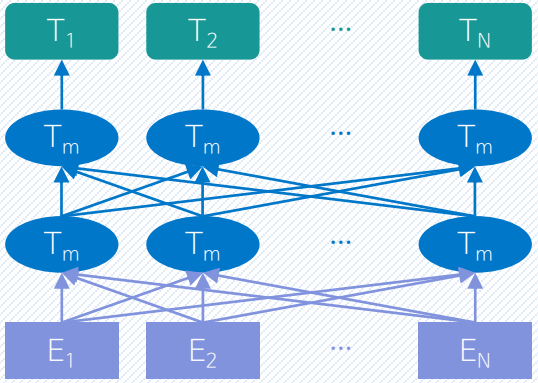
Ref. On the Opportunities and Risk of Foundation Models (2022)

2.2 Pre-trained LLM: BERT, GPT

대표적인 사전학습 모델

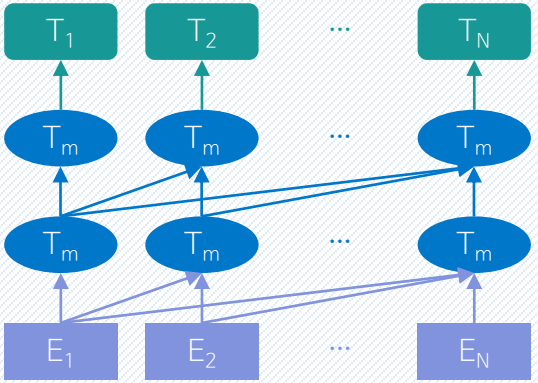
- Google, BERT (Bidirectional Encoder Representations from Transformers, 2018)
- OpenAI, GPT (Generative Pre-trained Transformer, 2018)

BERT



- Transformer Encoder, Bidirectional
- Pre-training : 1) 마스크 언어 모델,
  2) 다음 문장 예측

GPT



- Transformer Decoder, Unidirectional
- Pre-training : 다음 단어 예측

2.2 Pre-trained LLM: BERT, GPT

Pre-training: Self-supervised Learning

- BERT: Masked Language Model (MLM)
- GPT: Causal Language Model (CLM) - Next Word Prediction

BERT

- 빈칸 채우기 문제
- 양쪽 문맥 모두 참조 가능

인재개발원은 _____ 에 있습니다

나는 이번 학기에 _____ 강의를 수강합니다

BERT는 트랜스포머의 _____ 구조만을 사용합니다

조성진은 한국이 낳은 훌륭한 _____ 입니다

피보나치 수열은 1, 1, 2, 3, 5, 8, 13, 21, _____ 와 같다

GPT

- 다음 단어 맞추기 문제
- 앞쪽 문맥만 참조 가능

인재개발원은 _____

나는 이번 학기에 딥러닝 강의를 _____

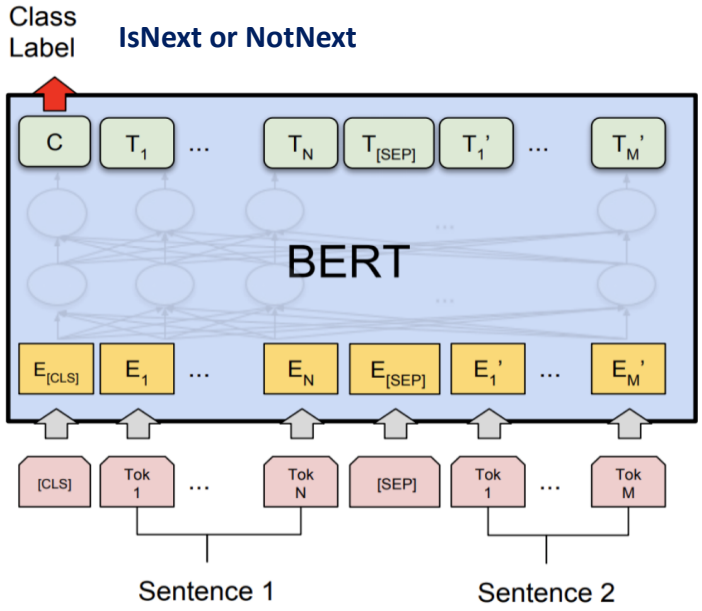
GPT는 트랜스포머의 _____

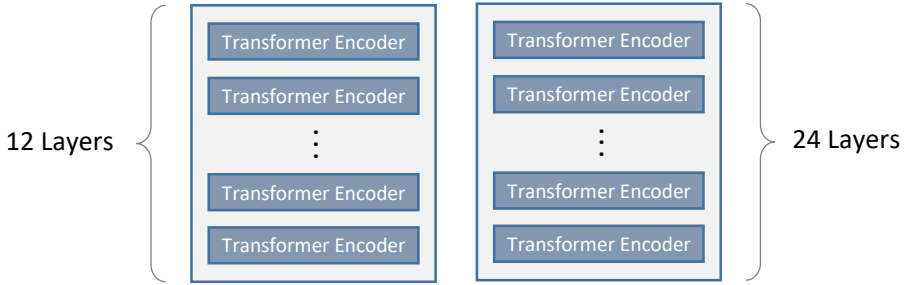
조성진은 한국이 낳은 훌륭한 _____

피보나치 수열은 1, 1, 2, 3, 5, 8, 13, 21, _____

2.2 BERT Basics

- BERT는 트랜스포머의 Encoder 구조만 이용하여 모델링
- 비지도 학습 : 마스크 언어모델 및 다음 문장 예측하도록 방대한 데이터를 사전학습

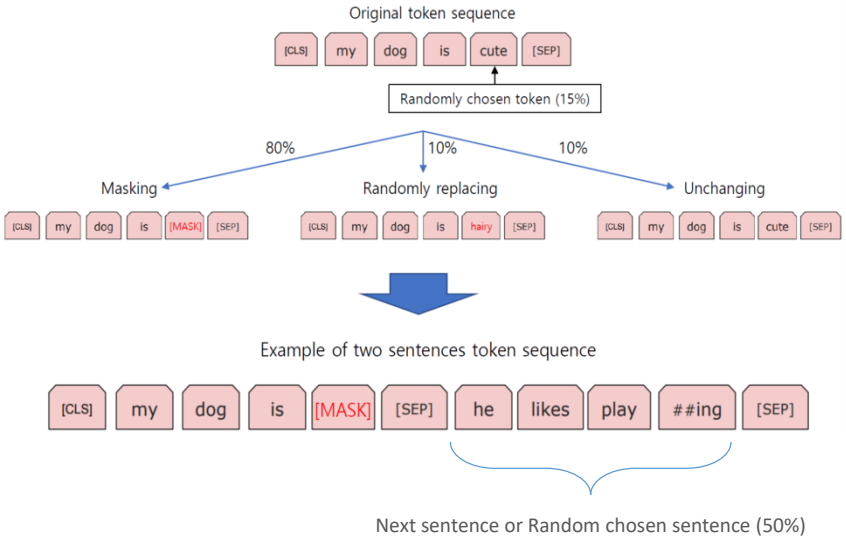




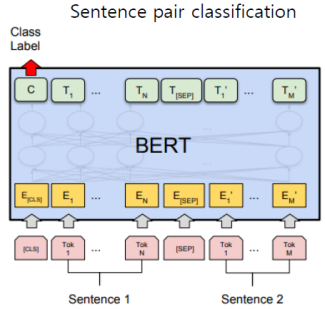
	Layers	*d_models*	*num_heads*	*Parameter*
BERT-Base	12	768	12	110M
BERT-Large	24	1,024	16	340M

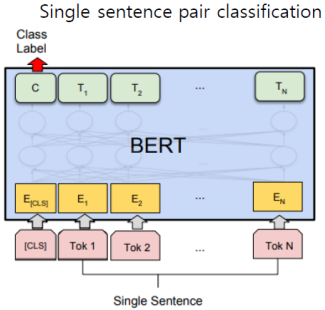
2.2 BERT Basics

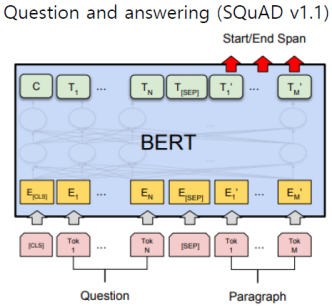
- BERT Pre-Training: MLM (Masked Language Model), NSP (Next Sentence Prediction)
- BERT Fine-Tuning: 다양한 NLP Task 적용

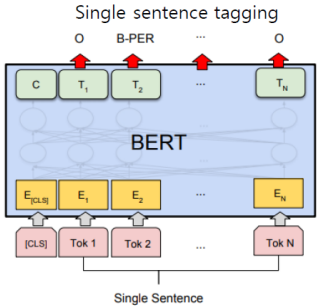


MLM, NSP Pre-Training Data





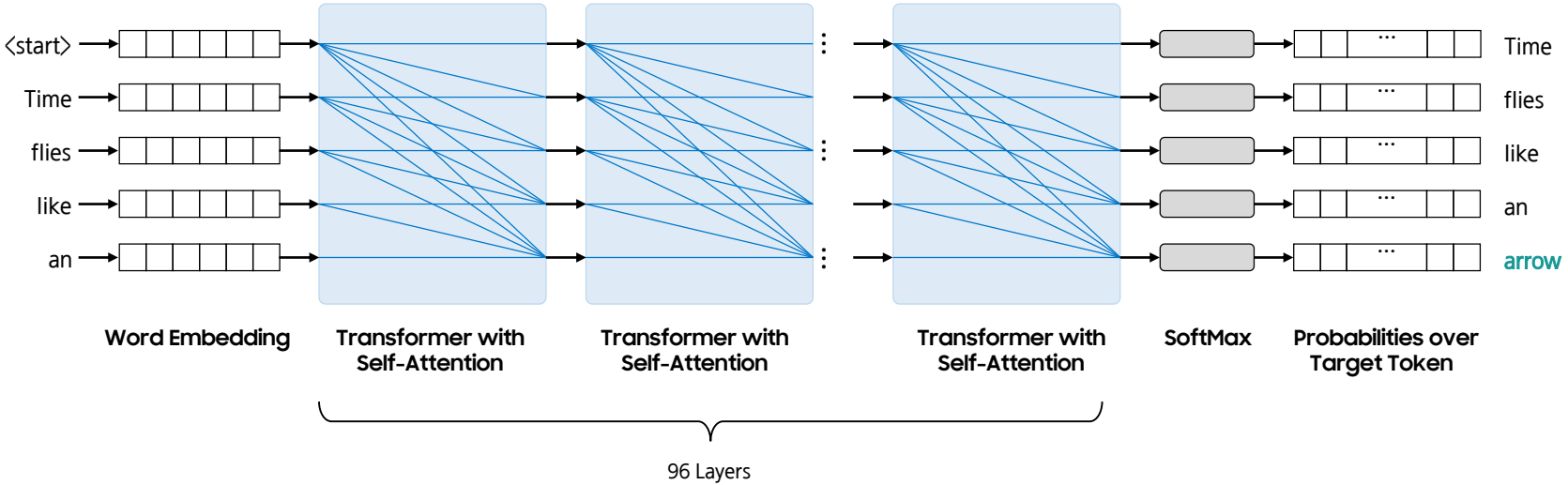




2.2 GPT Basics

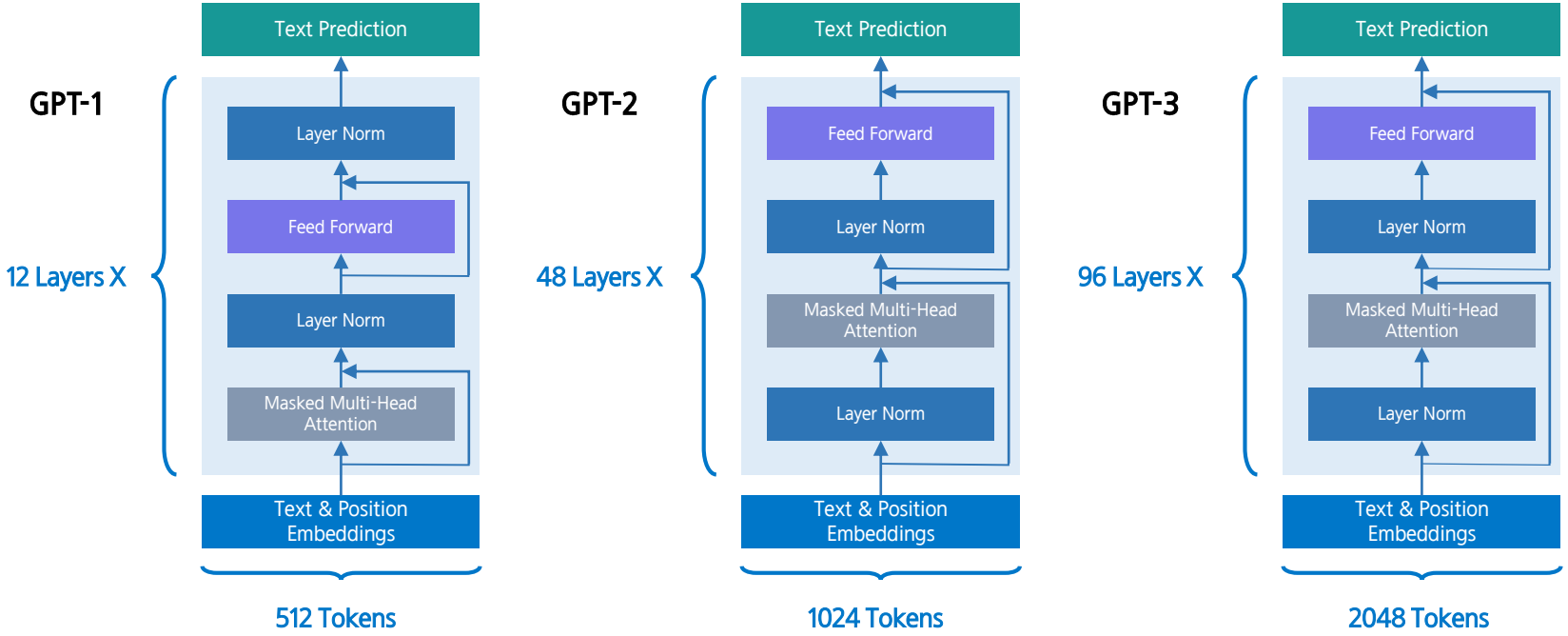
- GPT(Generative Pre-trained Transformer)는 트랜스포머의 Decoder 구조만 이용하여 모델링
- 비지도 학습 : 문장에서 이전 단어들을 활용하여 다음 단어를 예측하도록 방대한 데이터를 사전학습

언어 모델링 : Time flies like an [?]
문맥 (이전 단어) 예측 단어



2.2 GPT Basics

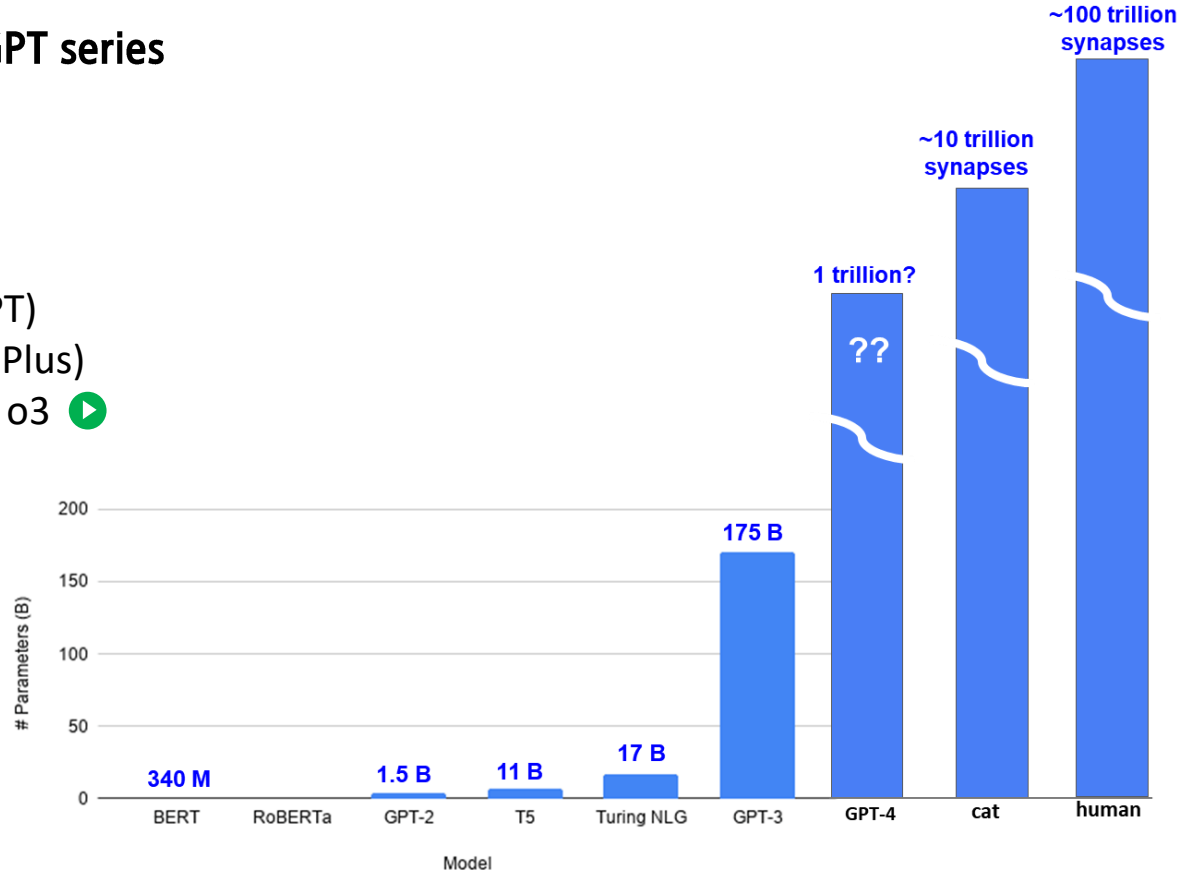
- GPT-3: 175 Billion Parameters (96 Layers, 2048 Tokens)
- GPT-4: ? (8K / 32K 토큰)



2.2 GPT Basics

Comparisons between GPT series

- 2018 GPT-1
- 2019 GPT-2
- 2020 GPT-3
- 2022 GPT-3.5 (ChatGPT)
- 2023 GPT-4 (ChatGPT Plus)
- 2024 GPT-4o, GPT o1, o3
- 2025 GPT-4.5
- 2025 GPT-5?



요약

- Transformer Basics: Attention + Feed-Forward
- Attention Mechanism, Self-Attention, Multi-Head Attention
- LLM (Large Language Model)
- Pre-Trained Model: BERT, GPT

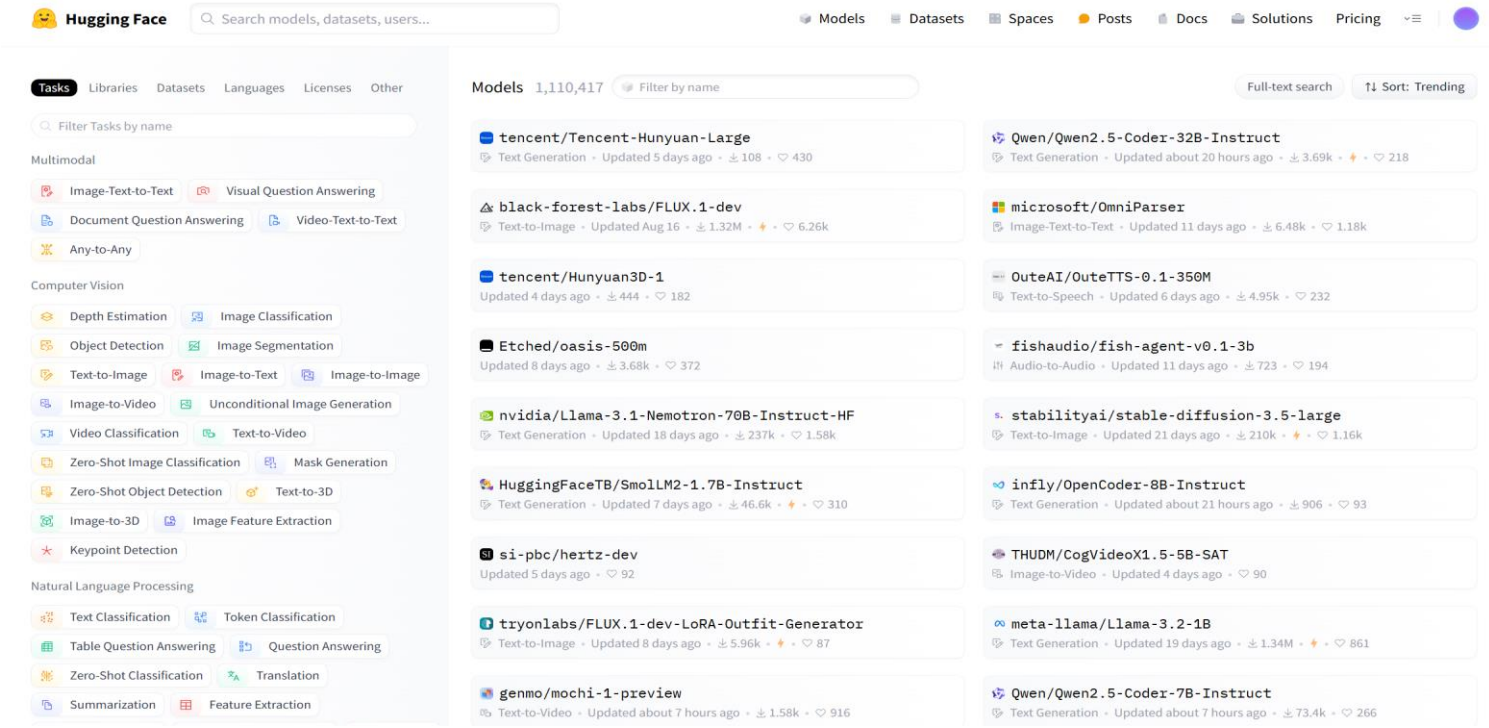
CONTENTS

SEU
Samsung Electronics University

3.1 Hugging Face Hub

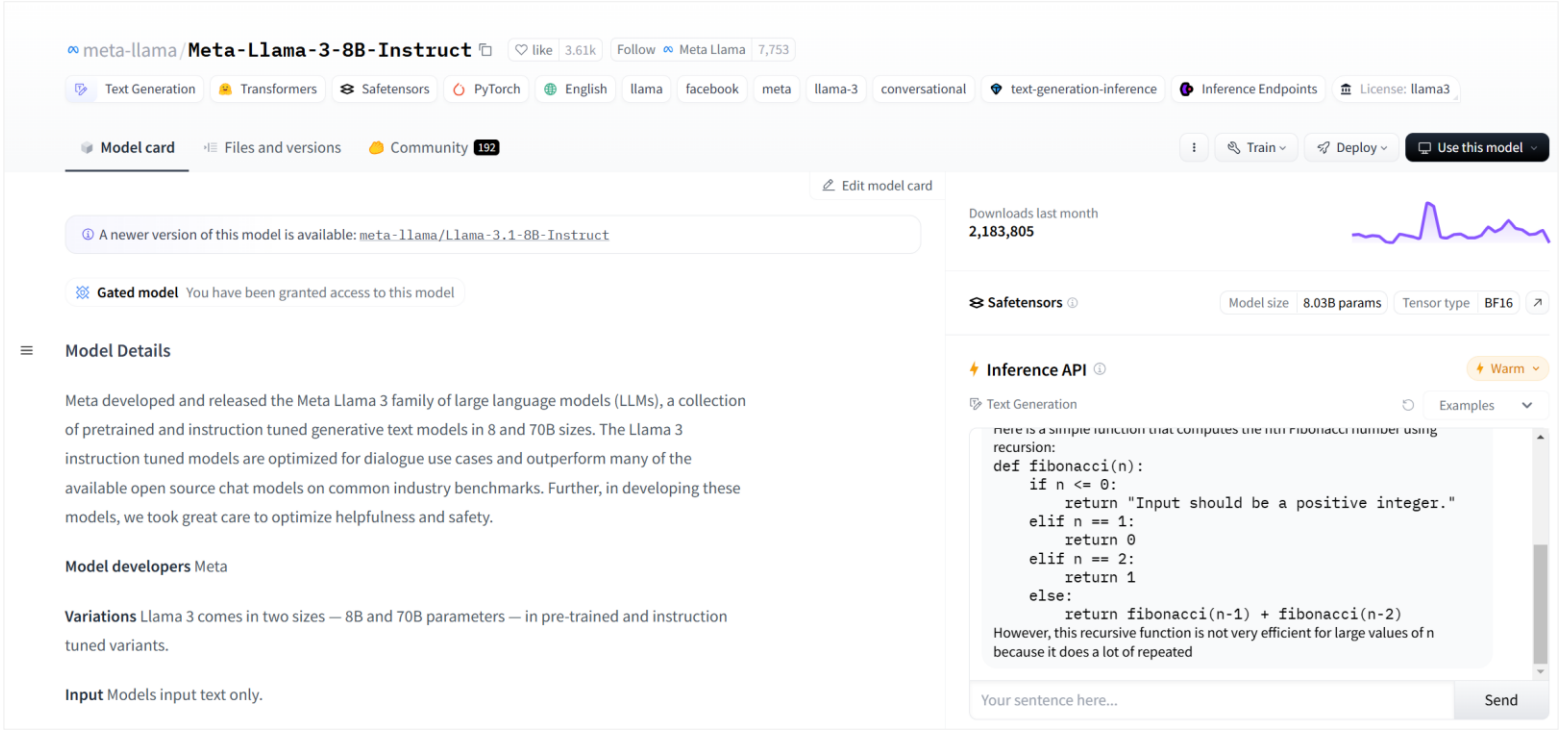
허깅페이스 허브 (https://huggingface.co/)

Huggingface Access Token



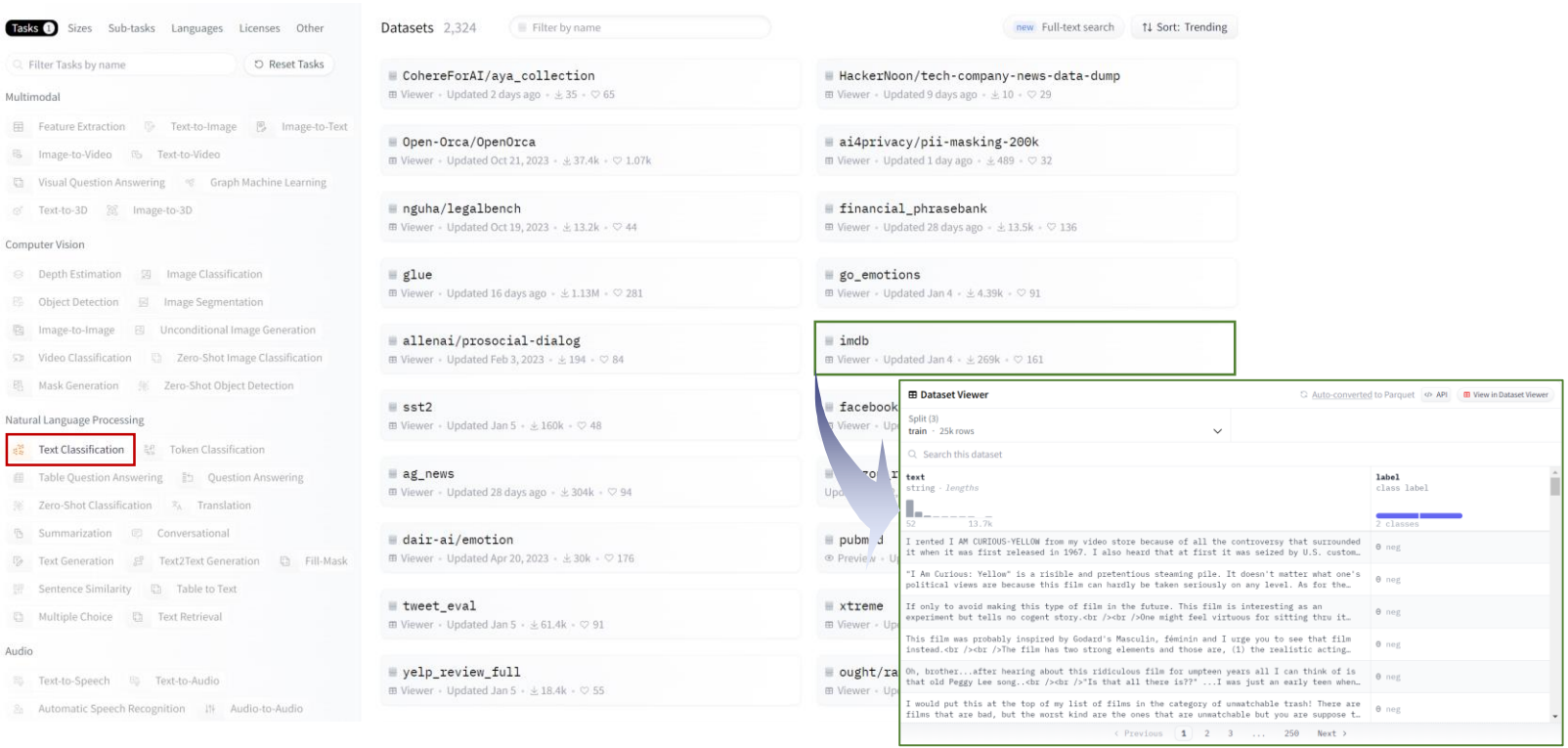
3.1 Hugging Face Hub

모델: 무료 공개된 1,900,000여개 모델 호스팅 (`25.7 기준)



3.1 Hugging Face Hub

- 데이터셋: 현재 Hub에는 460,000개 이상의 데이터셋 공유 (`25.7 기준)



3.2 Hugging Face Library

PreTrainedConfig Class

- 모델에 사용되는 하이퍼파라미터 정보 이용

PreTrainedModel Class

- Config Class를 이용하여 모델 구성

PreTrainedTokenizer Class

- 사전 학습된 모델의 토크나이저 사용

```
from transformers import AutoConfig, AutoTokenizer,
                         AutoModelForCausalLM, Trainer

model_ckpt = “gpt2-xl"

tokenizer = AutoTokenizer.from_pretrained(model_ckpt)
config = AutoConfig.from_pretrained(model_ckpt)
model = AutoModelForCausalLM.from_config(config)
```

Auto Class

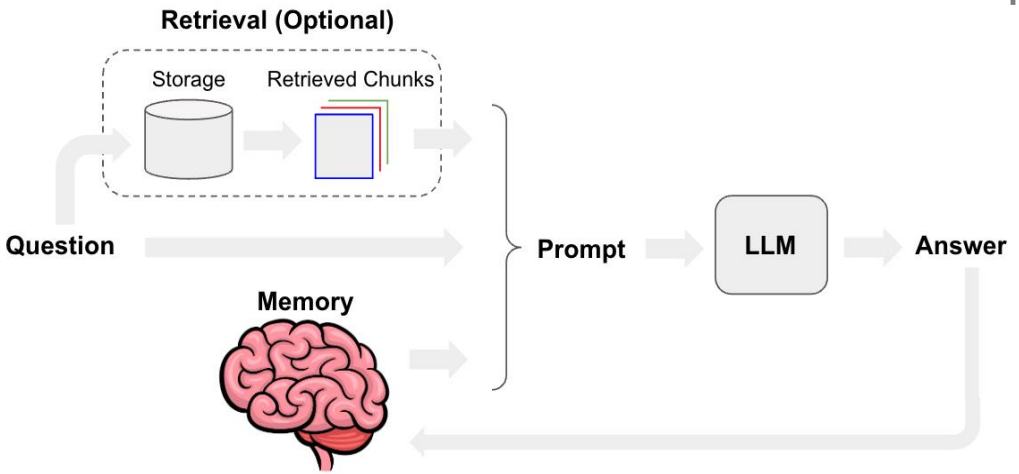
클래스	유형
AutoModelForSequenceClassification	입력 텍스트 분류
AutoModelForTokenClassification	입력 토큰 분류
AutoModelForCausalLM	생성형 언어모델
AutoModelForMaskedLM	마스크 언어모델
AutoModelForSeq2SeqLM	인코더-디코더 모델

Trainer Class

```
trainer = Trainer(
    model=None,              # The model to train, evaluate
    args=None,               # TrainingArguments Class
    data_collator=None,      # Data Collator
    train_dataset=None,      # The dataset for training
    eval_dataset=None,       # The dataset for evaluation
    compute_metrics=None,    # Metrics at evaluation
    optimizers=None          # Optimizer, Default: AdamW
)
```

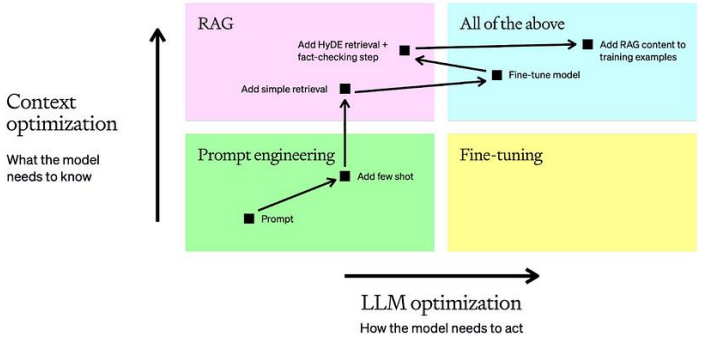
3.3 LLM Application

LLM Fine-Tuning v.s. Prompt v.s. RAG



Optimizing LLMs for Accuracy

- LLM Fine-Tuning (Ch.5)
- Response Generation (Next Word Prediction)
- Prompt Engineering
- Retrieval Augmented Generation (RAG)

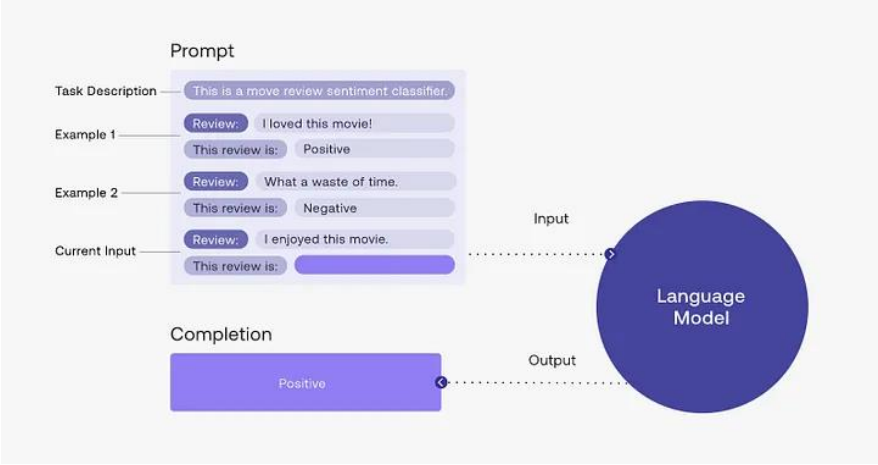


* LLM (Large Language Model), RAG (Retrieval Augmented Generation)

3.3 LLM Application – Prompt

Elements of a Prompt

- Instruction
- Context
- Input Data
- Output Indicator



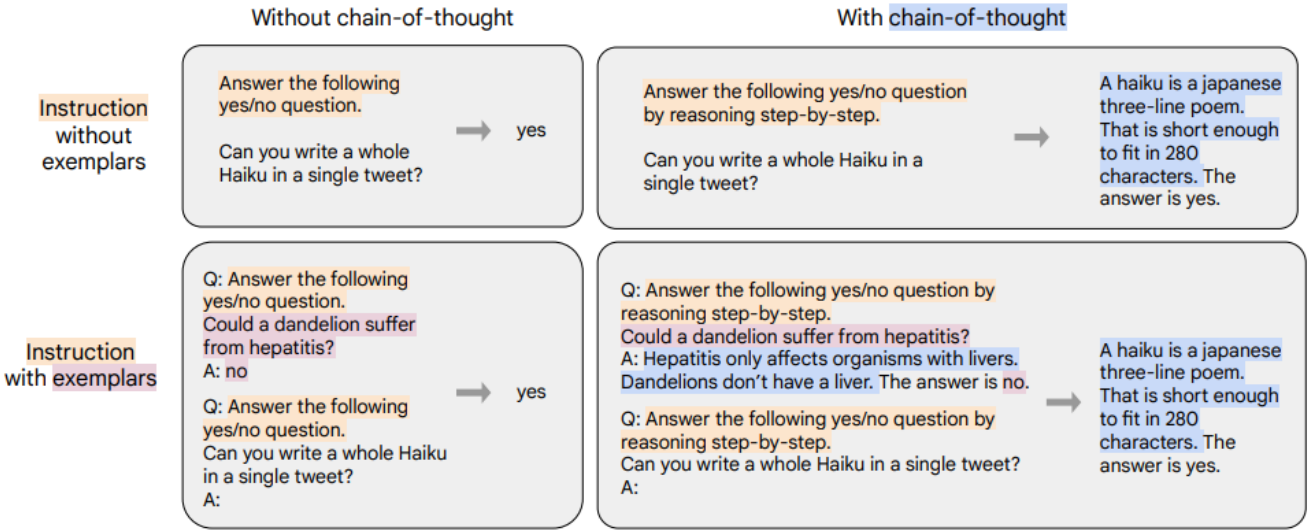
Prompt Engineering Techniques

- Zero-Shot/Few-Shot Prompting
- Chain-of-Thought Prompting
- Self-Consistency
- Tree of Thoughts
- Retrieval Augmented Generation
- Automatic Reasoning and Tool-use
- …

Ref. Prompt Engineering (cohere.ai) / A Prompt Pattern Catalog to Enhance Prompt Engineering with ChatGPT

3.3 LLM Application – Prompt

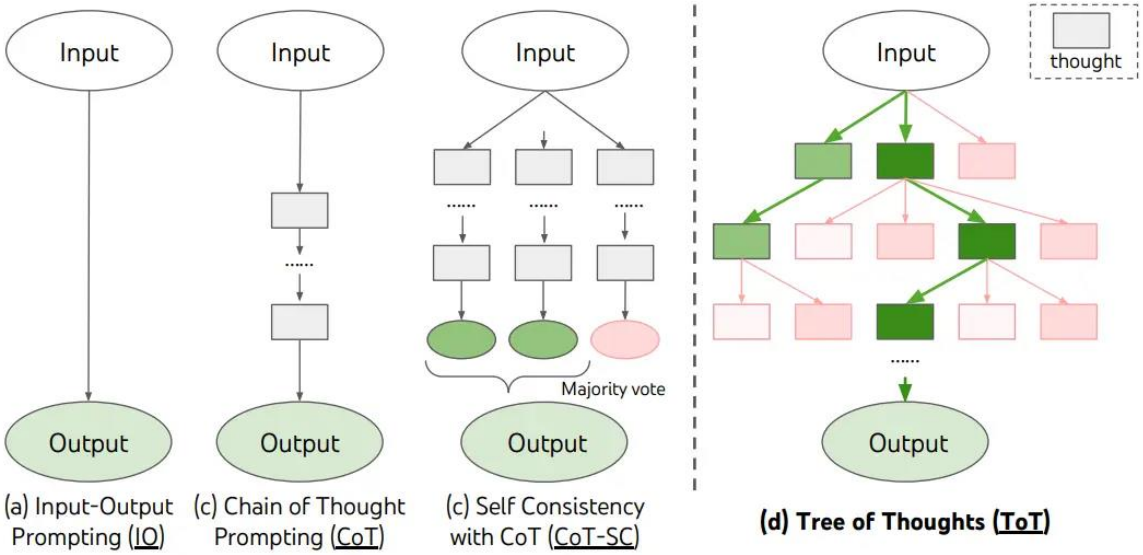
- **Prompt Engineering: Zero-Shot/Few-Shot Prompting, Chain-of-Thought**
  - In-context Learning 으로 프롬프트 내 패턴을 활용하여 예측
  - Few-Shot 등 예를 들어 설명하거나, 결론에 대한 이유를 설명하도록 요청 (CoT)



Chung, HW et al., Scaling Instruction-Finetuned Language Models (2022.12)

3.3 LLM Application – Prompt

- **Prompt Engineering: Tree of Thought (ToT)**
  - Tree 형태로 복수의 Reasoning Branch 진행
  - 1) Thought Decomposition, 2) Thought Generator, 3) State Evaluator, 4) Search Algorithm



Ref. https://arxiv.org/abs/2305.10601 (Tree of Thoughts: Deliberate Problem Solving with Large Language Models, 2023)

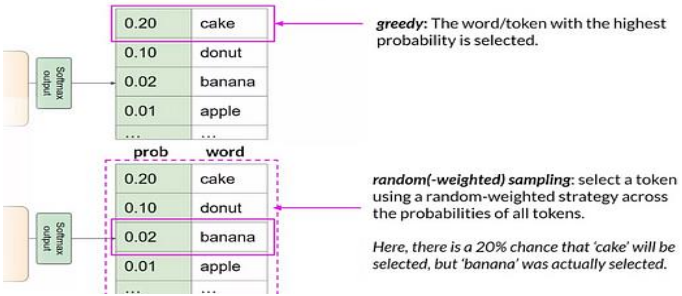
3.3 LLM Application – Generation

Text Generation: 언어 모델을 활용한 다음 단어 예측

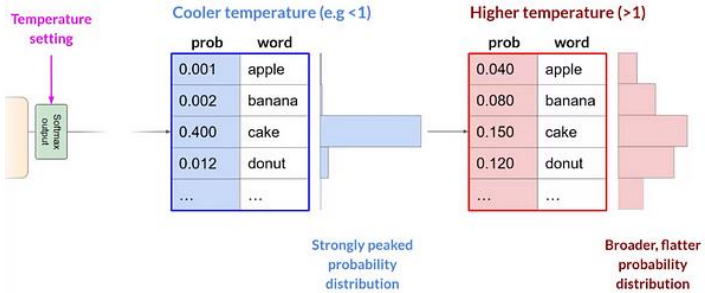
Greedy Search v.s Beam Search

- 고정된 어휘 리스트 확률 정보를 기반으로 다음 단어를 순차적으로 선택 (이전 시퀀스 기반 다음 단어 선택)
- 반복적인 고빈도 어휘의 선택 낮추고, 발화 자유도를 높이기 위해 다양한 샘플링 방식 존재

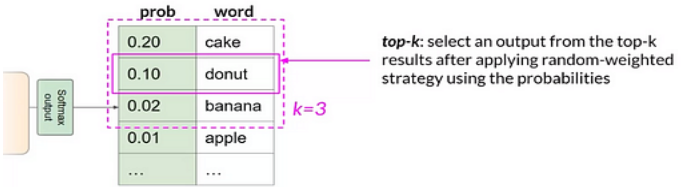
Generative config - greedy vs. random sampling



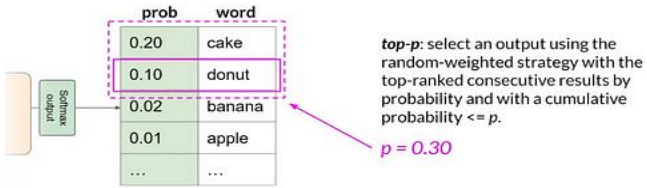
Generative config - temperature



Generative config - top-k sampling



Generative config - top-p sampling



Ref. https://www.coursera.org/learn/generative-ai-with-llms

3.3 LLM Application – Generation

- An increasingly common use case for LLMs is chat. In a chat context, the model instead continues a conversation that consists of one or more messages, each of which includes a role, like “user” or “assistant”, as well as message text.
- Chat Templates specify how to convert conversations, represented as lists of messages, into a single tokenizable string in the format that the model expects.

Instruction Following Format

```
messages = [
        { "role": "system",
          "content": "You are a friendly chatbot who always responds in the style of a pirate",
        },
        { "role": "user",
          "content": "How many helicopters can a human eat in one sitting?"},
]

tokenized_chat = tokenizer.apply_chat_template(messages, tokenize=True, add_generation_prompt=True,
                                               return_tensors="pt")
```

Ref. https://huggingface.co/docs/transformers/main/chat_templating

[실습] Text Generation

[실습 과제] 사전 학습된 CLM을 이용하여 자연어 텍스트 생성하기!!

(aas_clm_decoding_gpt.ipynb, aas_clm_decoding_llama.ipynb)

- Task: Text Generation (Greedy/Beam Search, Random/Top-K/Top-P Sampling, Chat Template)
- Pretrained LLM: GPT2-XL (Causal LM), Meta Llama-3-8B-Instruct (Chat Model)
- API: model.generate(), tokenizer.apply_chat_template()

[실습] Text Generation

Text Generation API

```
from transformers import AutoTokenizer, AutoModelForCausalLM

tokenizer = AutoTokenizer.from_pretrained(model_ckpt)
model = AutoModelForCausalLM.from_pretrained(model_ckpt)
encoded_input = tokenizer(input_txt, return_tensors="pt")

# Greedy Search Decoding
output_greedy = model.generate(**encoded_input, max_length=max_length, do_sample=False)

# Beam Search Decoding
output_beam = model.generate(**encoded_input, max_length=max_length, num_beams=5, do_sample=False)

# Random Sampling
output_temp = model.generate(**encoded_input, max_length=max_length, do_sample=True, temperature=2.0)

# Top-k Sampling
output_topk = model.generate(**encoded_input, max_length=max_length, do_sample=True, top_k=50)

# Top-p Sampling
output_topp = model.generate(**encoded_input, max_length=max_length, do_sample=True, top_p=0.90)
```

* Huggingface Pipeline("text-generation"): 모델 로딩, 토크나이징, 텍스트 생성, 디코딩의 과정을 한번에 처리

요약

- Hugging Face Hub
- Open Source Models, Datasets, etc.
- LLM Application: Prompt Engineering, RAG
- CLM Based Text Generation, Chat Templates

Reference

- Deep Learning with Python
- Deep Learning for NLP and Speech Recognition
- Natural Language Processing with Transformers, O'REILLY
- Natural Language Processing with PyTorch, O'REILLY
- Deep Learning Bible, https://wikidocs.net/book/8809
- The Hugging Face Course, https://huggingface.co/course/
- 딥러닝을 이용한 자연어 처리 입문, https://wikidocs.net/book/2155
- Transformers (신경망 언어모델 라이브러리) 강좌, https://wikidocs.net/book/8056
- 한권으로 끝내는 실전 LLM 파인튜닝, 위키북스

Thank You!

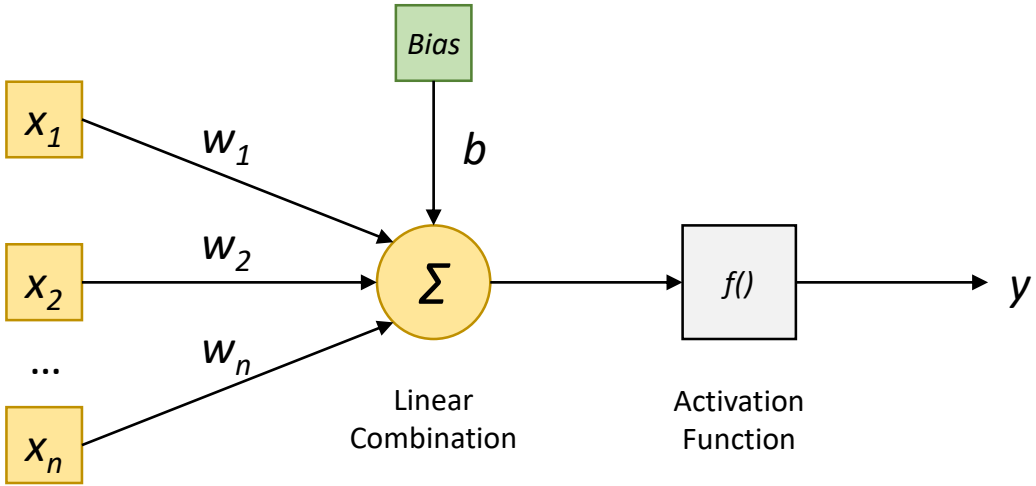
SEU
Samsung Electronics University

CONTENTS

Appendix

Samsung Electronics University

[Appendix] Structure of Perceptron



[Appendix] Distributional Semantics

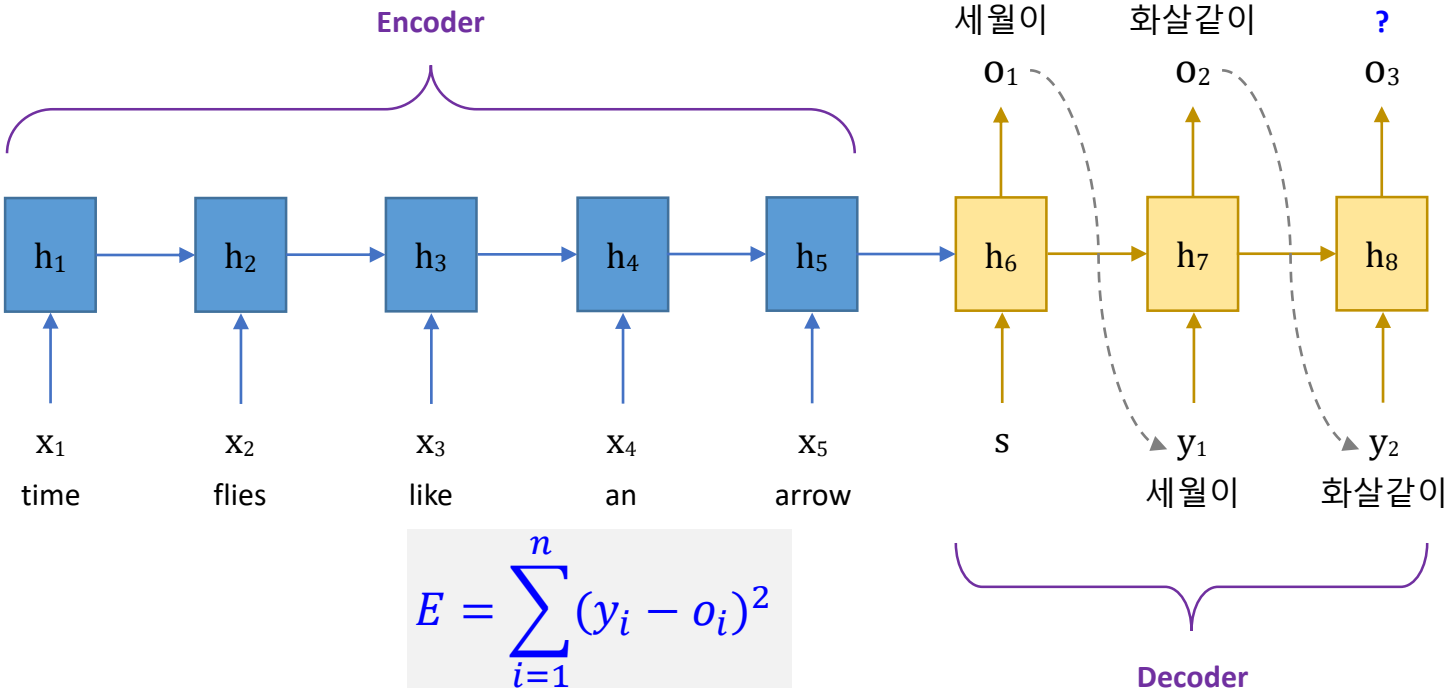
"You shall know a word by the company it keeps"

(John Rupert Firth 1957)

- A word's meaning is given by the words that frequently appear close-by
- One of the most successful ideas of modern statistical NLP!
- Words that occur in similar contexts tend to have similar meanings.

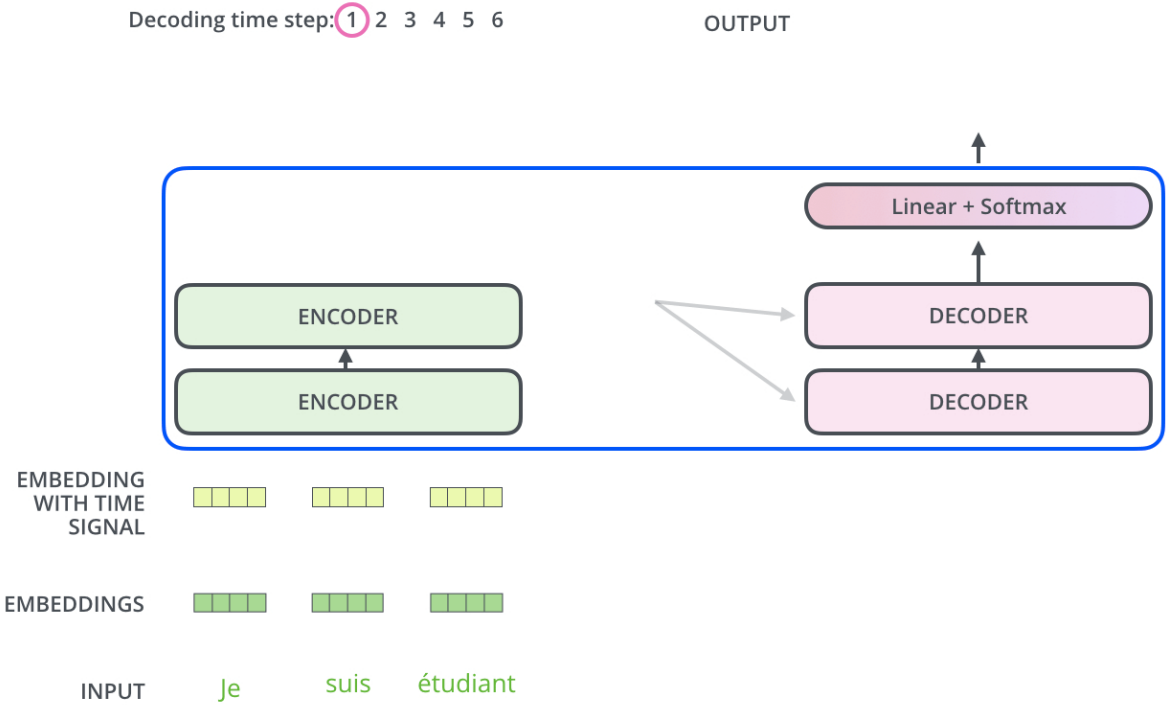
[Appendix] Seq2Seq: Many-to-Many

Encoder-Decoder 구조



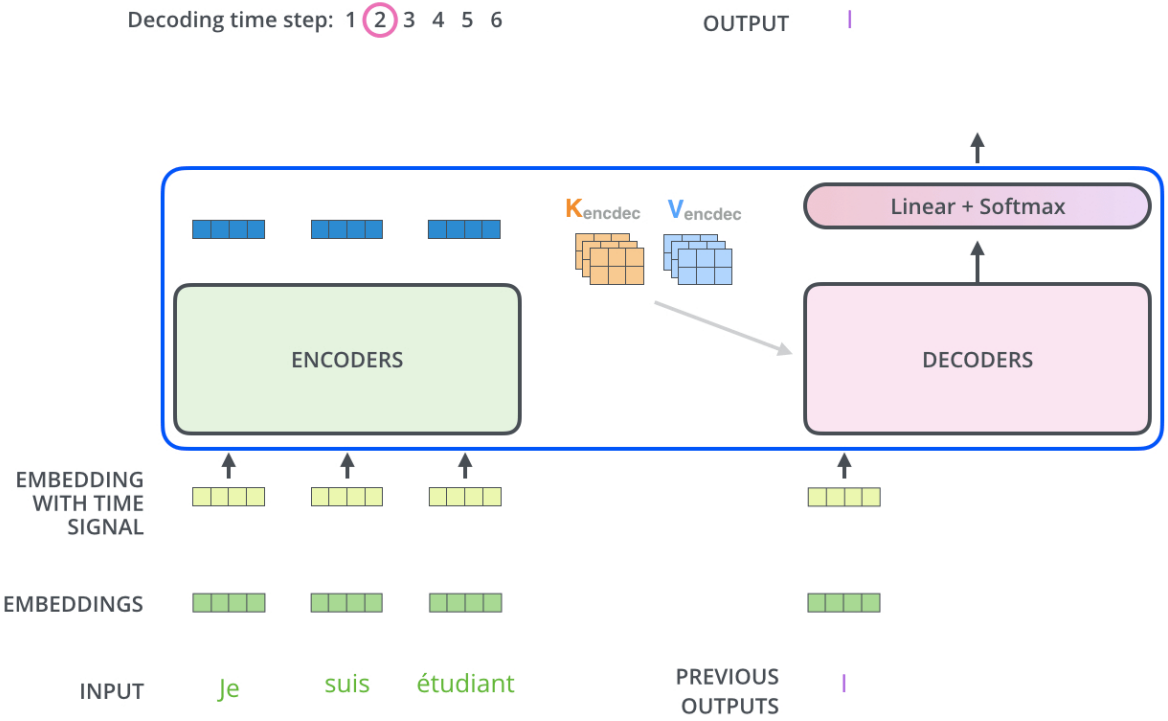
$$E = \sum_{i=1}^{n} (y_i - o_i)^2$$

[Appendix] Transformer Encoder



Ref. The Illustrated Transformer (https://jalammar.github.io/illustrated-transformer/)

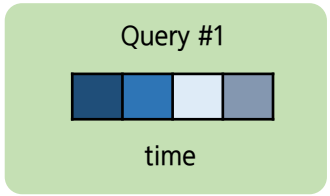
[Appendix] Transformer Decoder

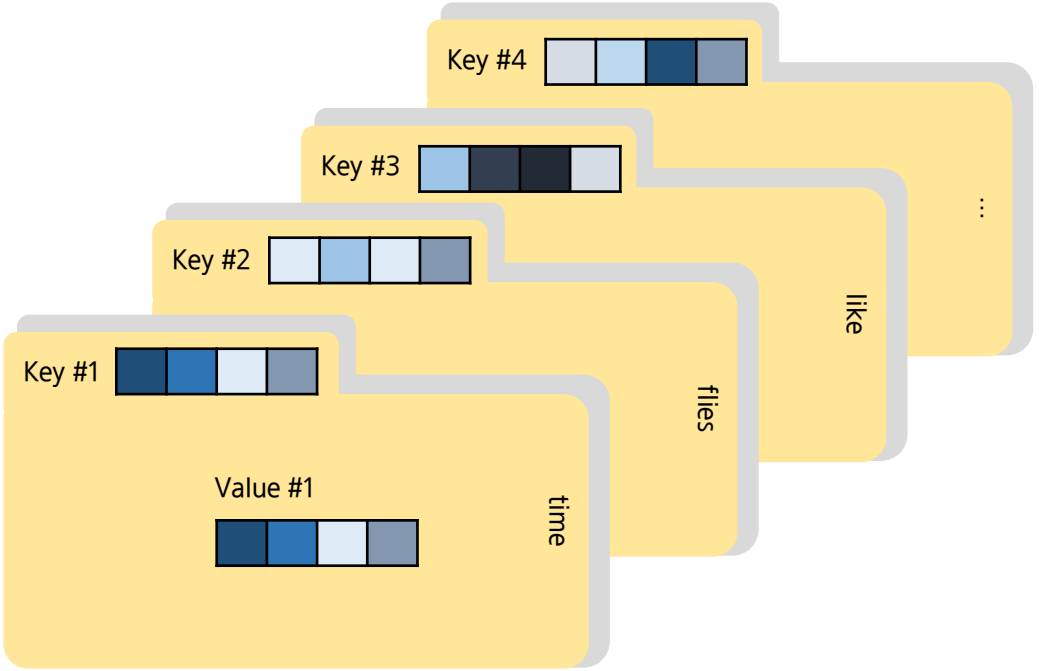


Ref. The Illustrated Transformer (https://jalammar.github.io/illustrated-transformer/)

[Appendix] Attention Mechanism

- **Query, Key, Value** 벡터 개념은 정보검색 시스템에서 유래





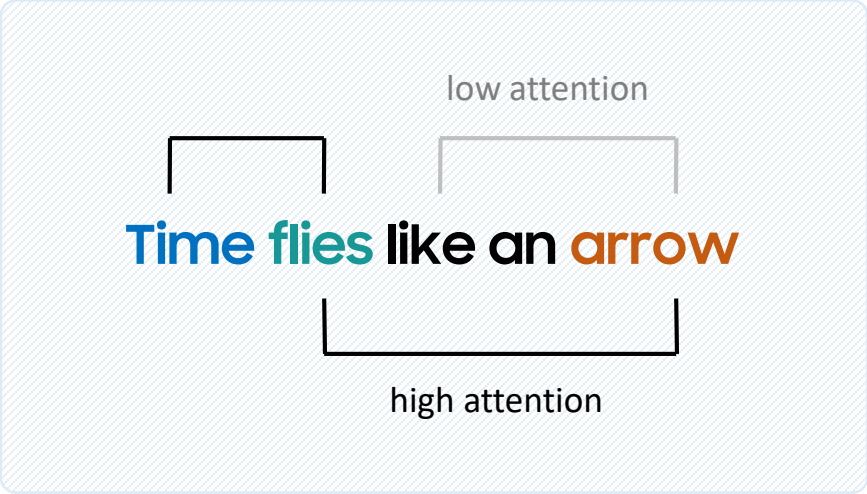
[Appendix] Self-Attention

- 같은 문장 (Sequence) 내 토큰들간의 관계에 의한 자기자신과의 특수 형태 Attention

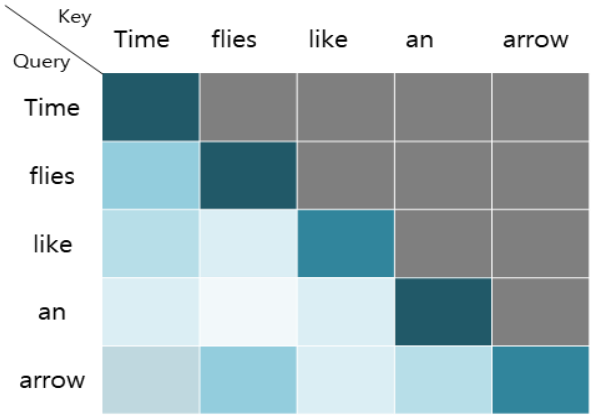
Definition:

$$A(x_i, X, X) = s_{i1}x_1 + s_{i2}x_2 + \ \ldots + s_{in}x_n = x_i'$$

s_{in} : $similarity\ of\ x_i\ to\ x_n$



Attention Map:



[Appendix] Attention

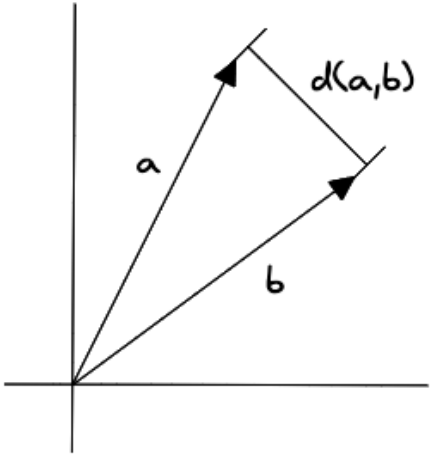
다양한 종류의 어텐션(Attention)

이름	스코어 함수	Defined by
Dot	$score(s_t, h_i) = s_t^T h_i$	Loung et al. (2015)
Scaled dot	$score(s_t, h_i) = \frac{s_t^T h_i}{\sqrt{n}}$	Vaswani et al. (2017)
General	$score(s_t, h_i) = s_t^T W_a h_i$ # W_a는 학습 가능한 가중치 행렬	Loung at al. (2015)
Concat	$score(s_t, h_i) = W_a^T tanh(W_b[s_t;h_i])$ $score(s_t, h_i) = W_a^T tanh(W_b s_t + W_c h_i])$	Bahdanau et al. (2015)
Location-base	$\propto_t = softmax(W_a s_t)$ # $\propto_t$ 산출시에만 s_t 만 사용하는 방법	Loung et al. (2015)

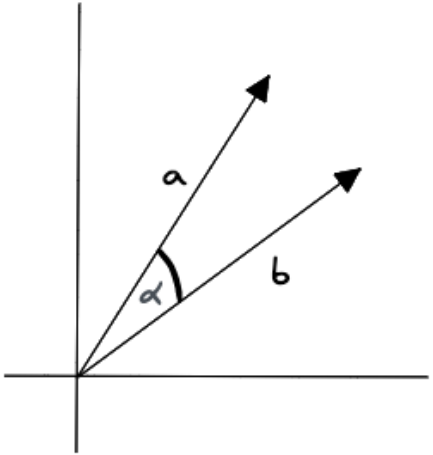
* s_t: Query, h_i: Keys, W: Weights

[Appendix] Similarity Metrics

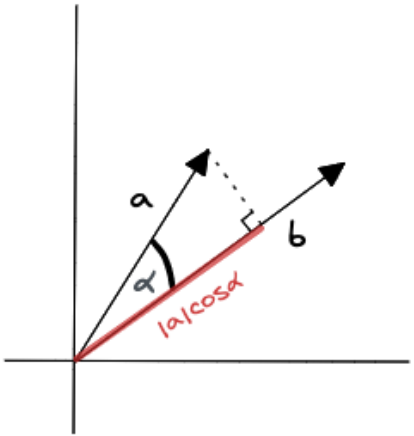
- Euclidean: 벡터간의 물리적인 거리를 측정하며, 이미지/소리 등의 유사성 측정에 사용
- Cosine: 벡터 방향의 유사성을 측정하며, 문서/단어 등의 유사성 측정에 사용
- Dot Product: 벡터의 방향과 크기에 따른 유사성을 측정하며, 문서/단어 등의 유사성 측정에 사용



Euclidean Distance



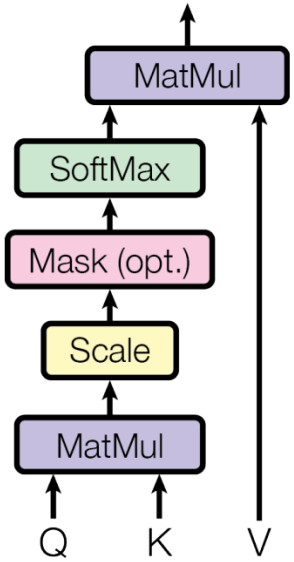
Cosine Similarity



Dot Product

[Appendix] Attention Mechanism

Scaled Dot-Product Attention



$$Z = Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_K}}\right)V$$

```python
def scaled_dot_product_attention(query, key, value, mask):
    matmul_qk = tf.matmul(query, key, transpose_b=True)

    dk = tf.cast(tf.shape(key)[-1], tf.float32)
    scaled_attention_logits = matmul_qk / tf.math.sqrt(dk)

    if mask is not None:
       scaled_attention_logits += (mask * -1e9)

    attention_weights = tf.nn.softmax(scaled_attention_logits, axis=-1)
    output = tf.matmul(attention_weights, value)

    return output, attention_weights
```

[Appendix] Cross Entropy Loss

Next Token Prediction: Cross Entropy Loss

0.01
0.15
0.09
0.58
0.02
0.03
0.14
0.07

True Probability distribution (one-hot)

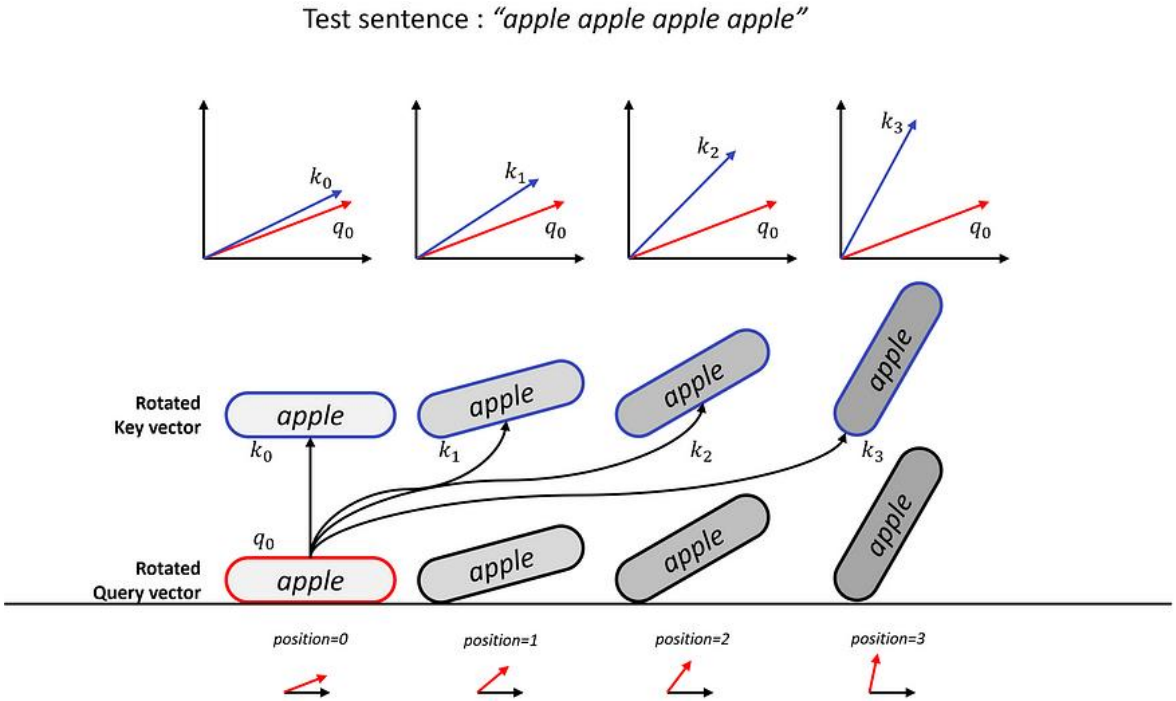
$$H_p(q) = -\sum_{x \in class} q(x_i) \cdot \log p(x_i)$$

Predicted Probability distribution

0
0
0
1
0
0
0
0

[Appendix] RoPE

Example

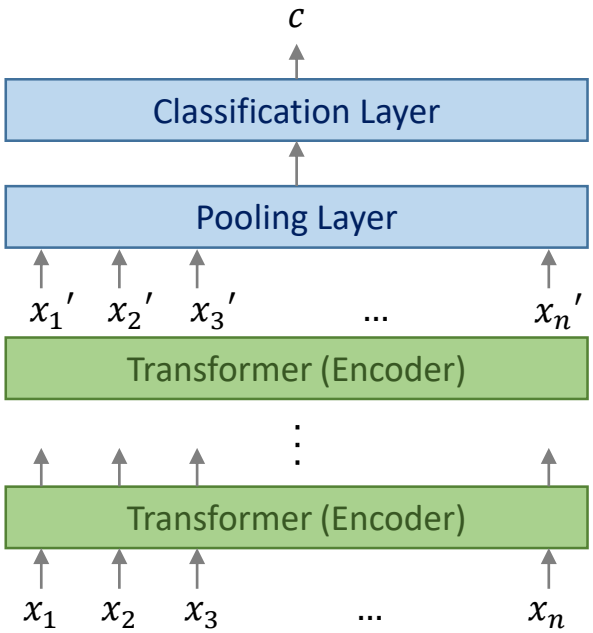


Ref. ROPEformer: Enhanced Transformer with Rotary Position Embedding (https://arxiv.org/pdf/2104.09864)

[Appendix] Sequence Classification Task

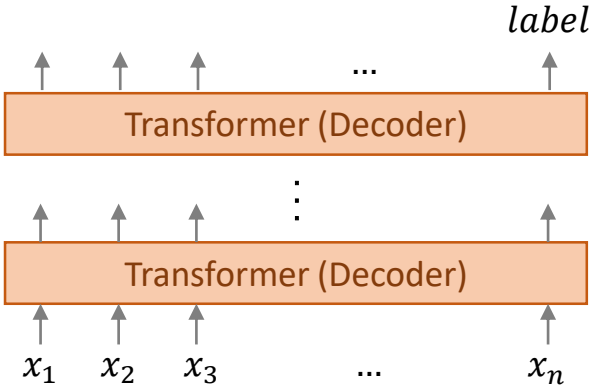
Encoder Model

- Input: Token Sequence
- Output: Label



Decoder Model (CLM)

- Input: Token Sequence + Label
- Output: Next Word Prediction



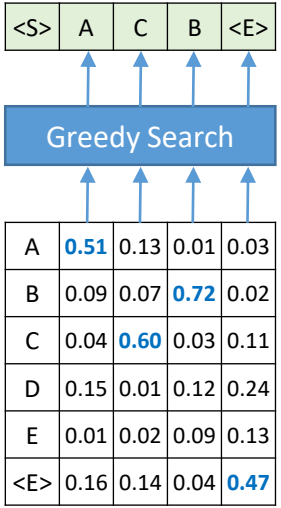
Tweet text : sentence … label :

[Appendix] Decoding

Greedy Search Decoding

- 각 타임스텝에서 확률이 가장 높은 토큰을 선택

$$\hat{y}_t = \underset{y_t}{\operatorname{argmax}}\, P(y_t \mid y_{<t}, x)$$

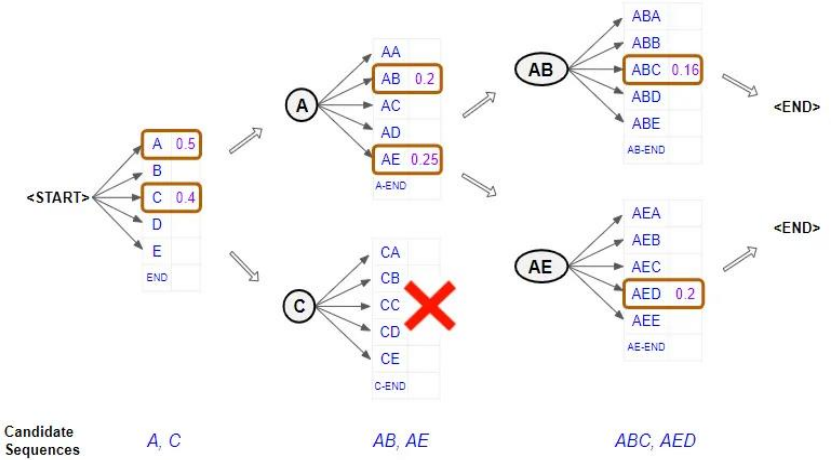


A	**0.51**	0.13	0.01	0.03
B	0.09	0.07	**0.72**	0.02
C	0.04	**0.60**	0.03	0.11
D	0.15	0.01	0.12	0.24
E	0.01	0.02	0.09	0.13
<E>	0.16	0.14	0.04	**0.47**

Beam Search Decoding

- 각 타임스텝에서 확률 높은 상위 B개 토큰을 추적

$$\log P(y_1, \ldots, y_t \mid x) = \sum_{t=1}^{N} \log P(y_t \mid y_{<t}, x)$$



[Appendix] Chat GPT

Instruction Training

- GPT-3의 경우 자연어 생성 태스크 (다음 단어 예측)에 대해 뛰어난 성능을 보임
- 반면, Prompt를 따르도록 학습된 것은 아니기 때문에 Instruction Following 성능이 미흡
- Prompt-Response 데이터를 이용한 Fine-tuning으로 성능 개선한 Instruct GPT 발표

Step-1

예제 데이터 수집 후 Supervised Learning 기반의 초기 모델 완성
(Demonstration Dataset: 13K Prompts)

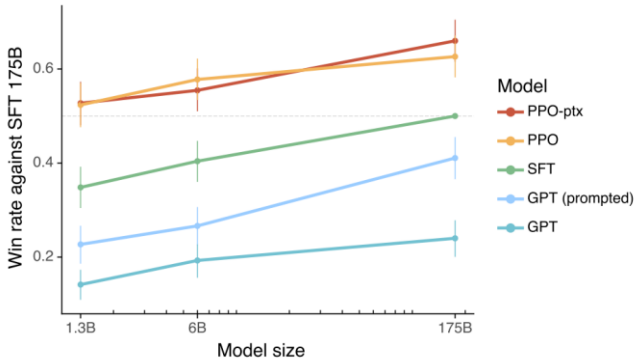
[Prompt] *Explain the moon landing to a 6 year old*
[Response] *Some people went to the moon ...*

Step-2

결과물에 대한 사람의 선호도를 학습하여 Reward Model 확보
* 사람의 Feedback을 모사

Step-3

강화학습을 통한 Reward Model 최적화로 Fine-tuning



Ref. Long Ouyang, et. al., "Training language models to follow instructions with human feedback", OpenAI, 2022

[Appendix] Chat GPT

In-context Learning: Zero-shot, One-shot, Few-shot

- 원하는 태스크에 대한 간단한 설명을 함께 입력하여 주는 방법 (원하는 태스크에 빠르게 적응할 수 있도록 학습)

Zero-shot

The model predicts the answer given only a natural language description of the task. No gradient updates are performed.

```
Translate English to French:        ← task description
cheese =>                           ← prompt
```

One-shot

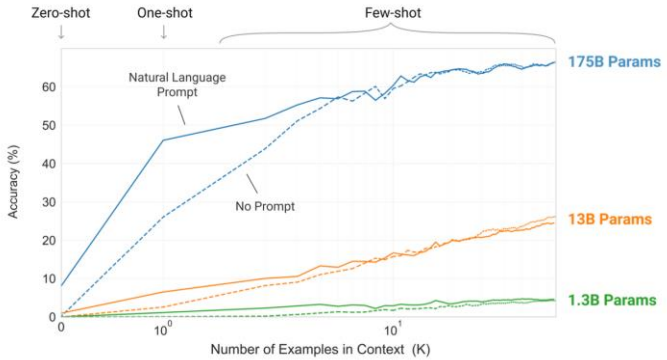
In addition to the task description, the model sees a single example of the task. No gradient updates are performed.

```
Translate English to French:        ← task description
sea otter => loutre de mer          ← example
cheese =>                           ← prompt
```

Few-shot

In addition to the task description, the model sees a few examples of the task. No gradient updates are performed.

```
Translate English to French:        ← task description
sea otter => loutre de mer          ← examples
peppermint => menthe poivrée
plush girafe => girafe peluche
cheese =>                           ← prompt
```



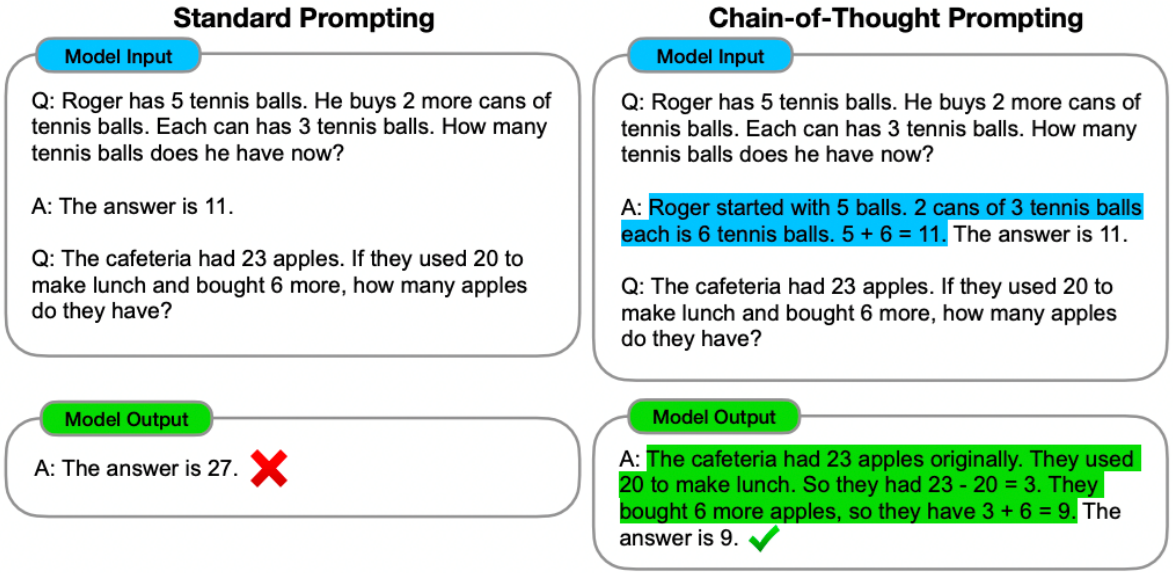
다양한 Function(Task) Approximation 가능성!

Ref. Tom B. Brown. Benjamin Mann, Nick Ryder, Melanie Subbiah, et.al., Language Models are Few-Shot Learners", OpenAI, 2020

[Appendix] CoT

In-context Learning: CoT (Chain of Thought)

- **Step-by-step Reasoning**

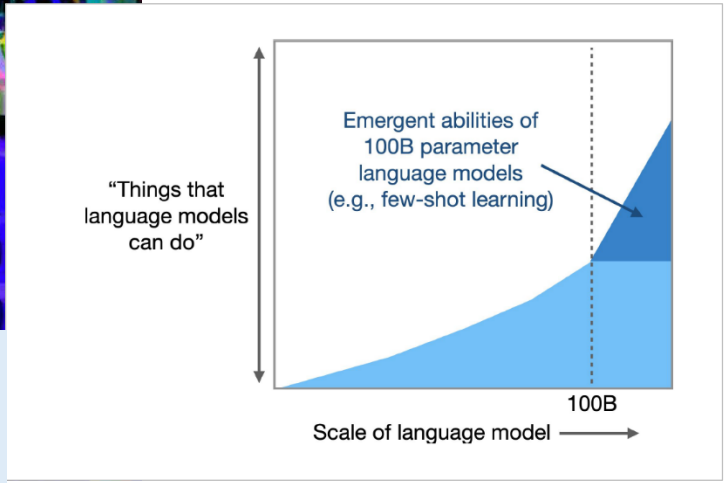


Ref. https://arxiv.org/pdf/2201.11903 (Chain-of-Thought Prompting Elicits Reasoning in Large Language Models, Google)

[Appendix] Generative AI

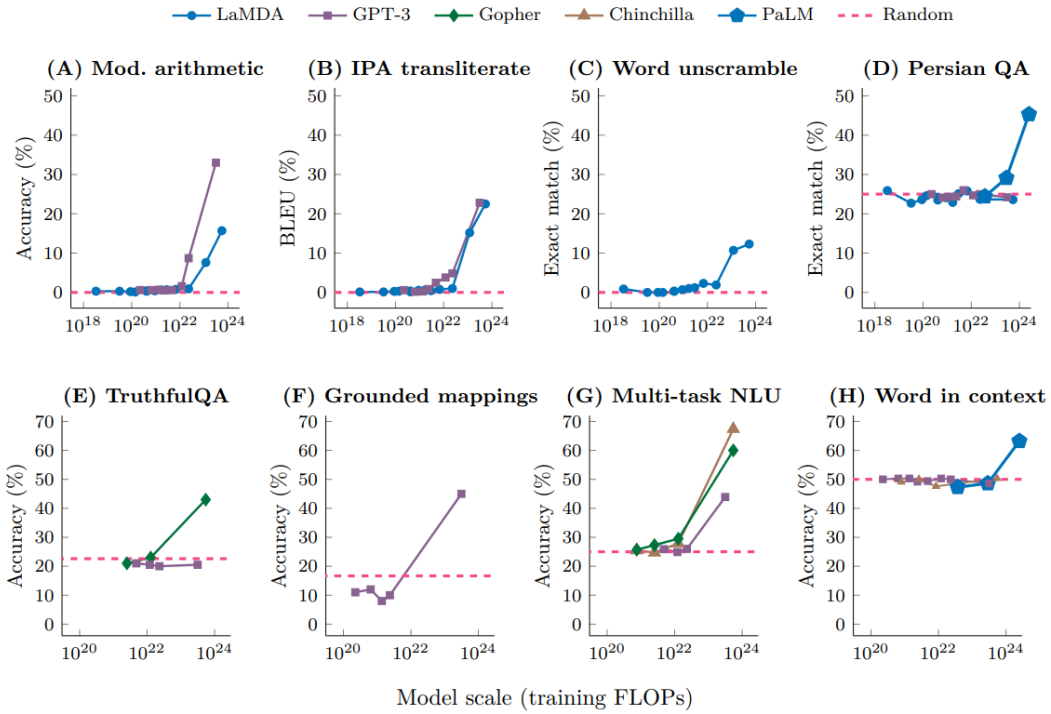
- **Wei, Jason, et al. "Emergent abilities of large language models."**
  (https://arxiv.org/abs/2206.07682)

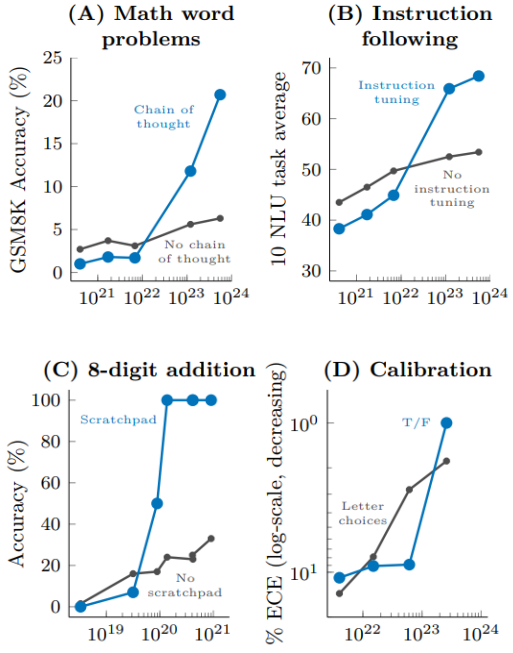
LLM 모델 학습 과정에서 특정 임계치를 넘으면 기존의
작은 모델에서는 나타나지 않았던 새로운 능력이 발현됨



Ref. Examining Emergent Abilities in Large Language Models, Stanford University (2022)

[Appendix] Generative AI

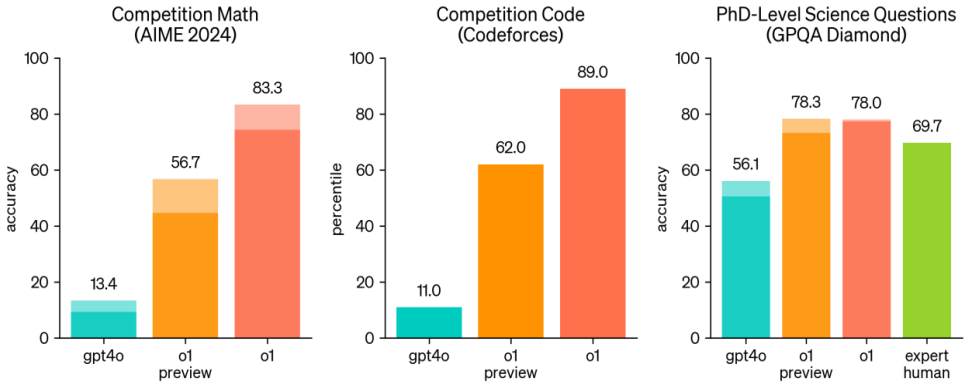


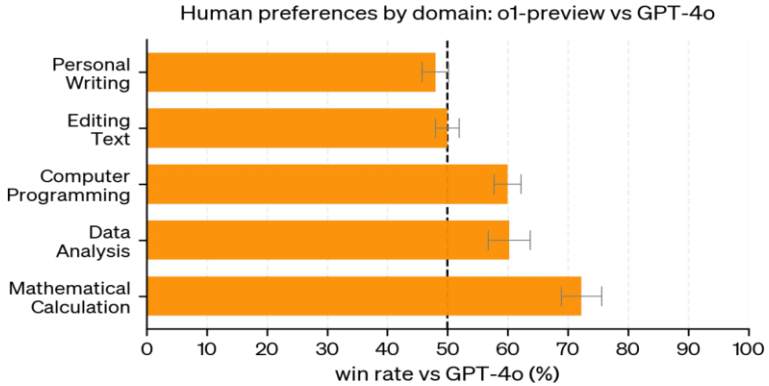


Ref. Examining Emergent Abilities in Large Language Models, Stanford University (2022)

[Appendix] BigTech: OpenAI – GPT

GPT-4o v.s. GPT-o1





Ref. OpenAI

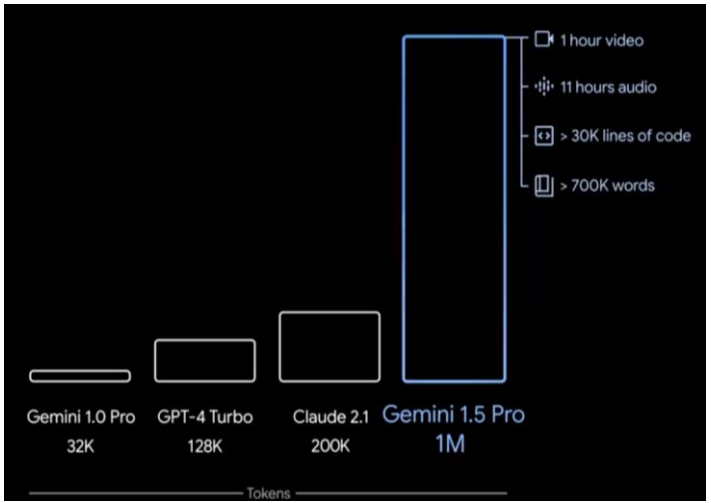
[Appendix] BigTech: Google – Gemini

트랜스포머 모델 발표 (2017년) 이후 BERT, LaMDA, PaLM, Gemini 등 지속 발표

- 2023년 2월 LaMDA 언어모델 기반 Bard, 5월 Google I/O에서 PaLM2 언어모델 기반 Bard 발표
- 차세대 LLM 'Gemini' (`23.12): 텍스트, 이미지, 오디오, 동영상 등 멀티모달 정보를 인식/이해/생성
- 2024년 5월 Google I/O 2024: Gemini Pro 1.5 / Flash / Nano 등 업데이트

* Context Window: 2M (1.4M 개 단어, 2 hr 분량 비디오, 22 hr 분량 오디오)

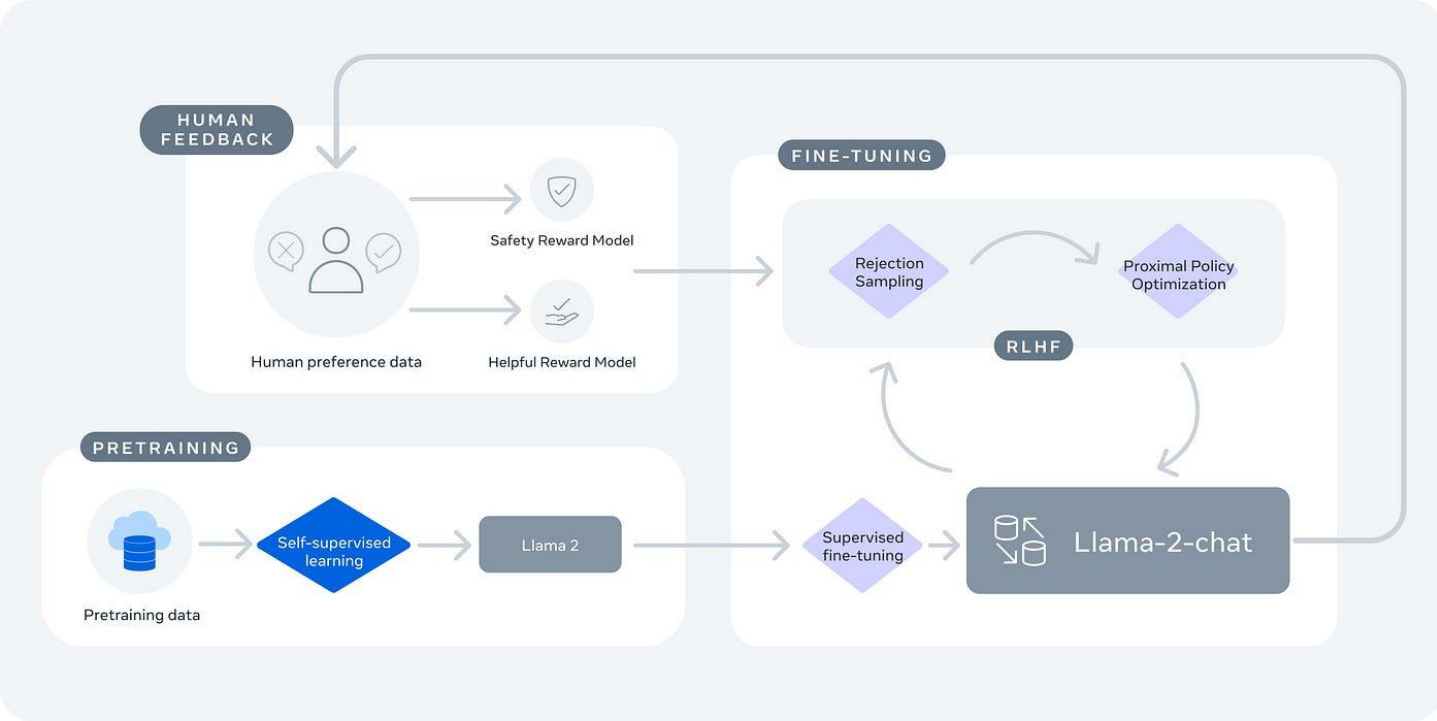
모델	시기	파라미터 수
BERT	2018	110 M (Base) 340 M (Large)
T5	2019	11 B (22GB)
LaMDA (Bard)	2021	137 B (274 GB)
PaLM	2022	540 B
Gemini	2023	1.75 T (Ultra)



Ref. https://blog.google/technology/ai/google-gemini-ai/

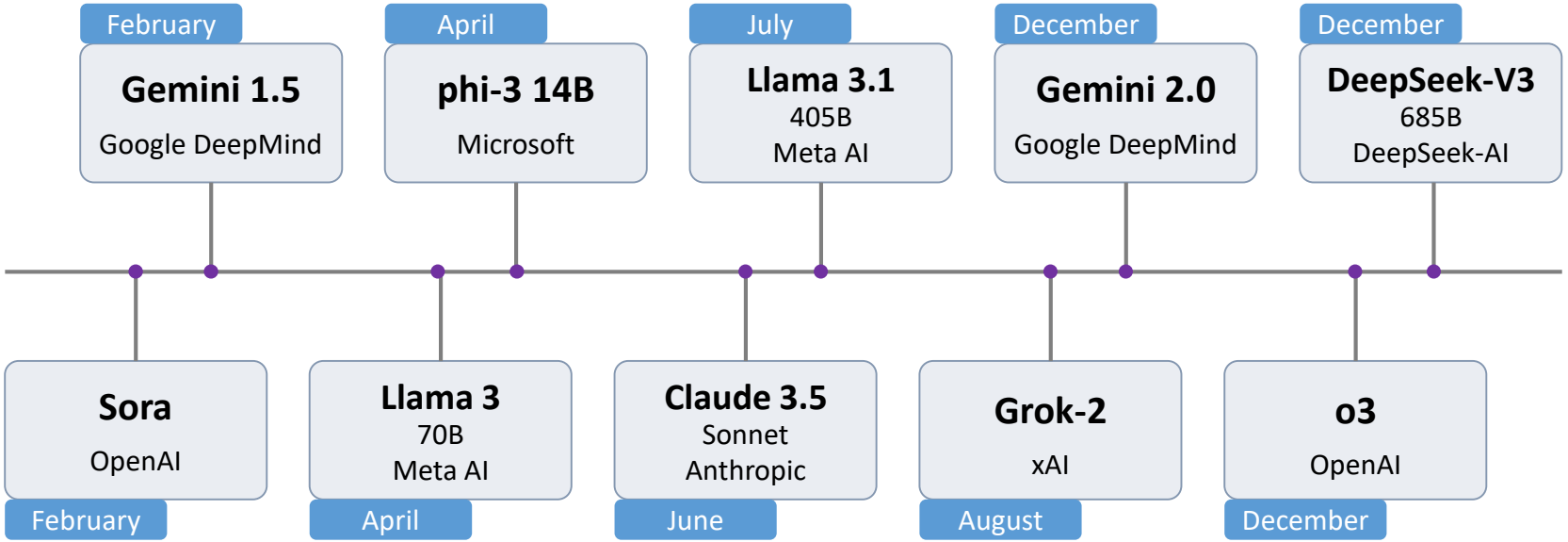
[Appendix] Opensource LLM: Llama-2

- **Llama 2 is intended for commercial and research use in English.**

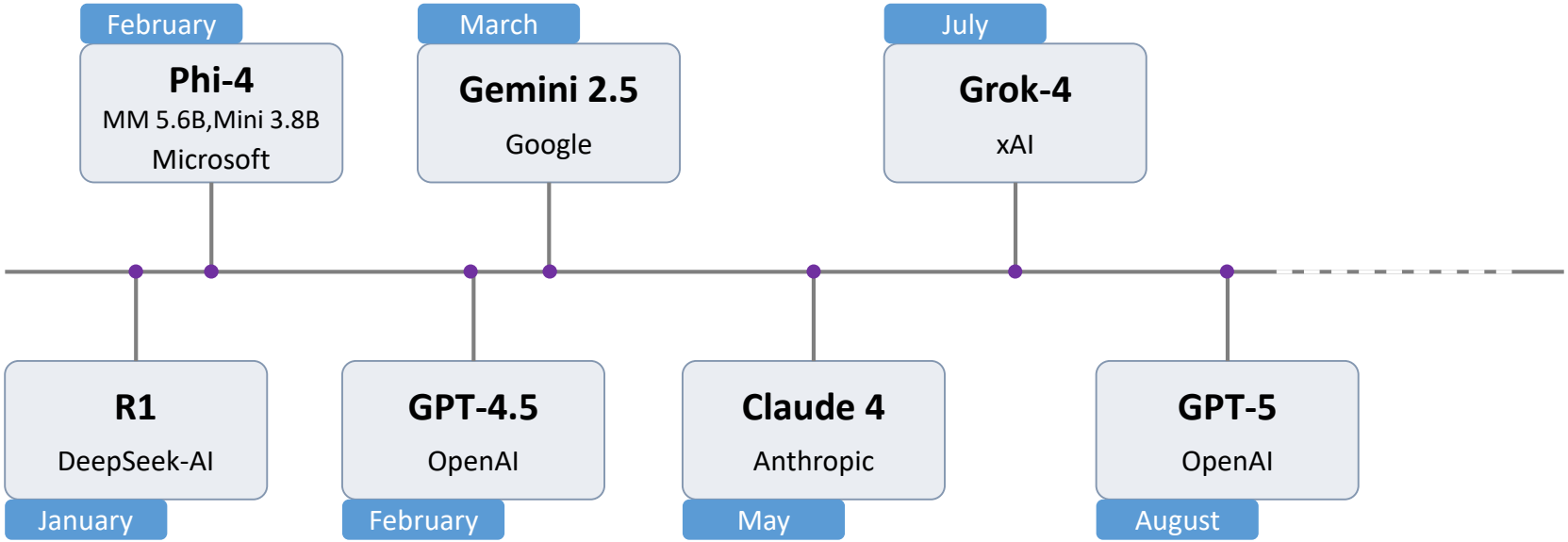


Ref. Llama 2: Open Foundation and Fine-Tuned Chat Models (arXiv:2307.09288)

[Appendix] GenAI Model Timeline 2024

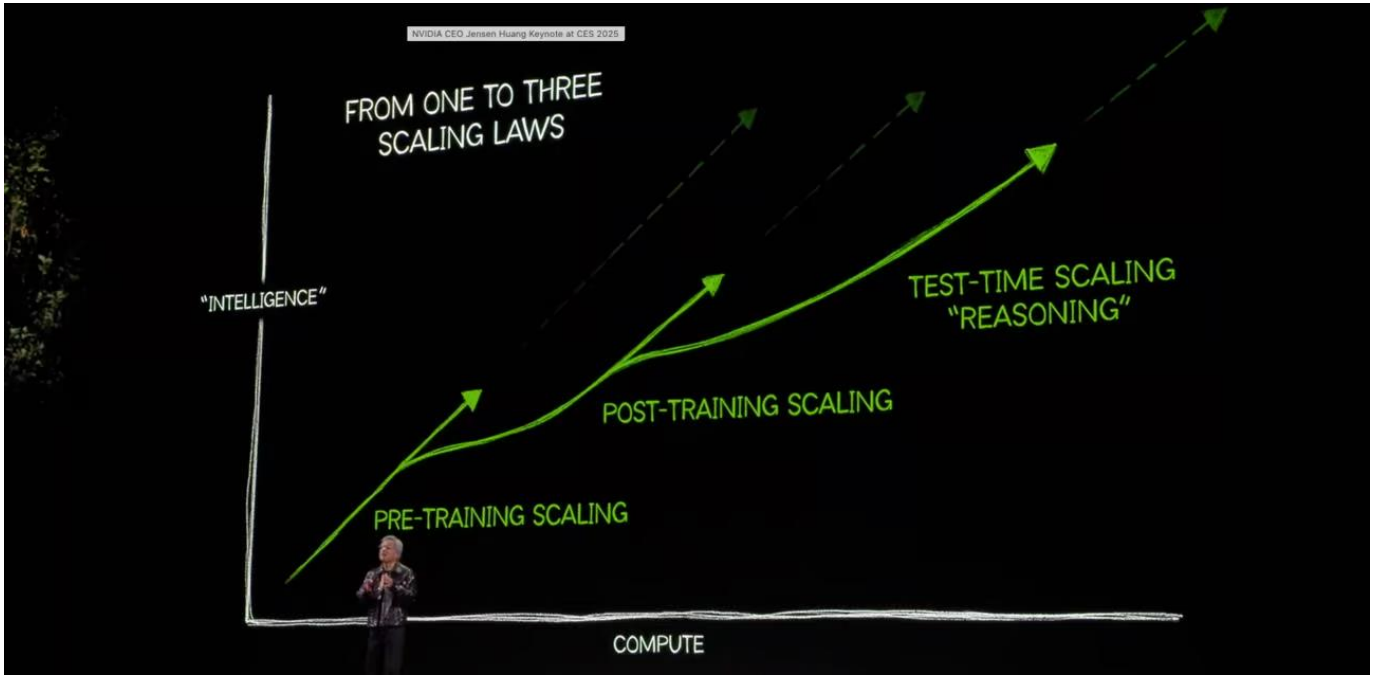


[Appendix] GenAI Model Timeline 2025



[Appendix] NVIDIA GTC 2025

- **PRE-TRAINING SCALING → POST-TRAINING SCALING → TEST-TIME SCALING**



[Appendix] Prompt Engineering

LLM Settings (Text Generation)

- Temperature
- Top P
- Max Length
- Stop Sequences
- Frequency Penalty
- Presence Penalty

$$softmax(y_i) = \frac{e^{\frac{y_i}{T}}}{\sum_j e^{\frac{y_j}{T}}}$$

Temperature

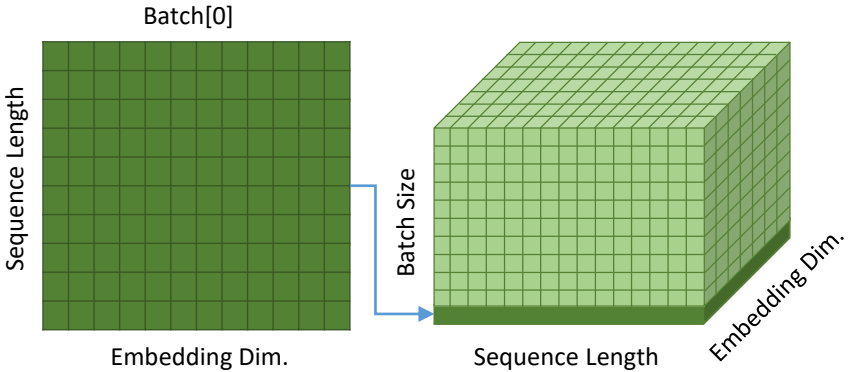
Elements of a Prompt

- Instruction
- Context
- Input Data
- Output Indicator

[Appendix] Transformer

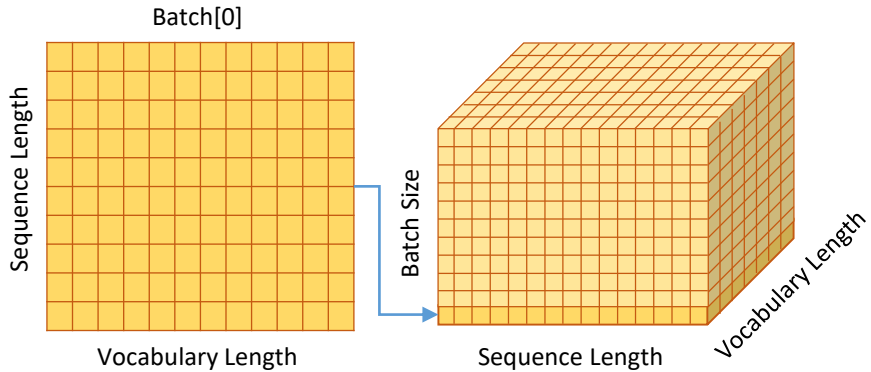
Source Input Tensor

- Batch Size
- Sequence Length
- Embedding Dimension



Output Probabilities Tensor

- Batch Size
- Sequence Length
- Vocabulary Length.



[Appedix] Instruction Following Format

Meta Llama 3

System prompt and multiple turn conversation between the user and assistant

```
<|begin_of_text|><|start_header_id|>system<|end_header_id|>

{{ system_prompt }}<|eot_id|><|start_header_id|>user<|end_header_id|>

{{ user_message_1 }}<|eot_id|>
<|start_header_id|>assistant<|end_header_id|>

{{ model_answer_1 }}<|eot_id|><|start_header_id|>user<|end_header_id|>

{{ user_message_2 }}<|eot_id|>
<|start_header_id|>assistant<|end_header_id|>
```

Alpaca

We used the following prompts for fine-tuning the Alpaca model:

- for examples with a non-empty input field:

```
Below is an instruction that describes a task, paired with an input that provides further context. Write a

### Instruction:
{instruction}

### Input:
{input}

### Response:
```

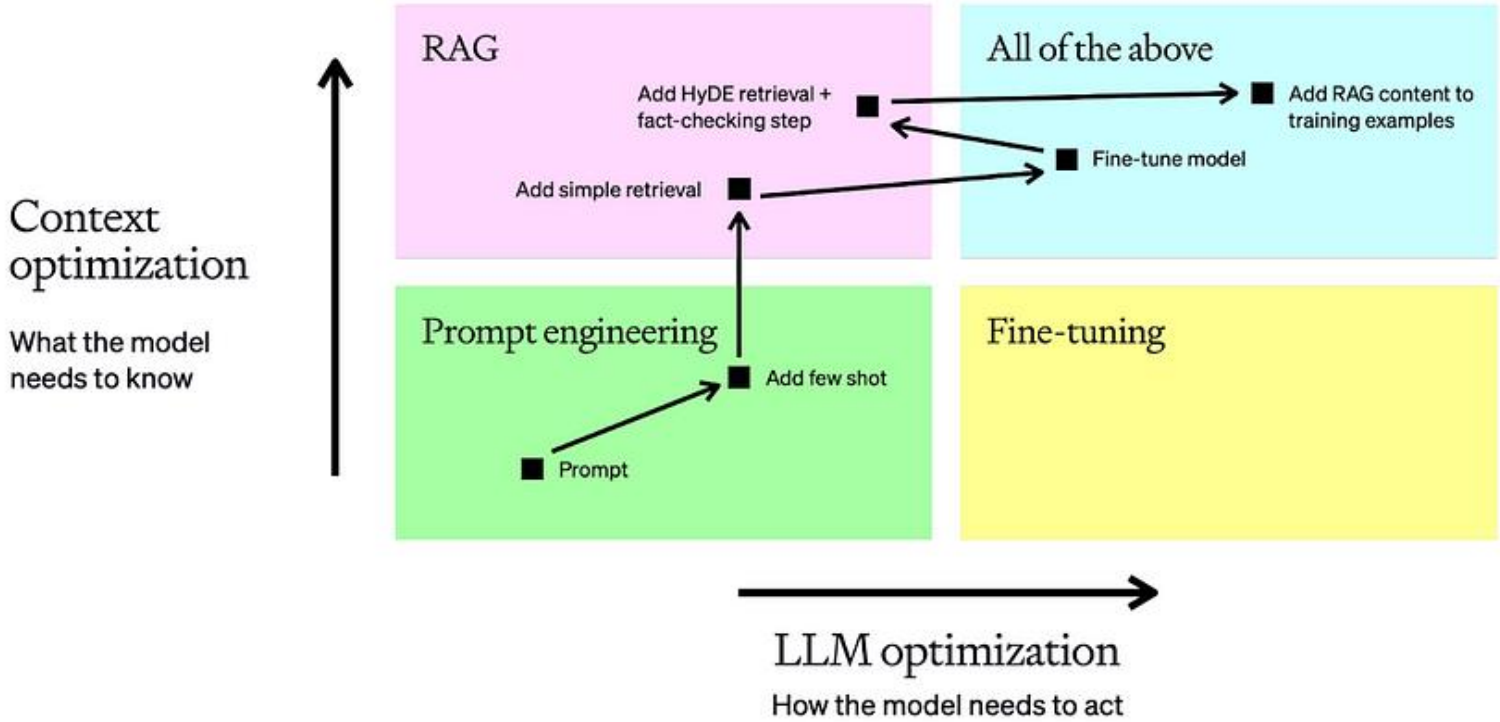
- for examples with an empty input field:

```
Below is an instruction that describes a task. Write a response that appropriately completes the request.

### Instruction:
{instruction}

### Response:
```

During inference (eg for the web demo), we use the user instruction with an empty input field (second option).

```
<|start_header_id|>system<|end_header_id|>
You are a helpful AI assistant.<|eot_id|><|start_header_id|>user<|end_header_id|>
Hi!<|eot_id|><|start_header_id|>assistant<|end_header_id|>
Hello there! It's great to meet you!<|eot_id|><|start_header_id|>user<|end_header_id|>
What are some recipes for making chocolate chip cookies?<|eot_id|><|start_header_id|>assistant<|end_header_id|>
```

[Appendix] Optimizing LLMs for Accuracy

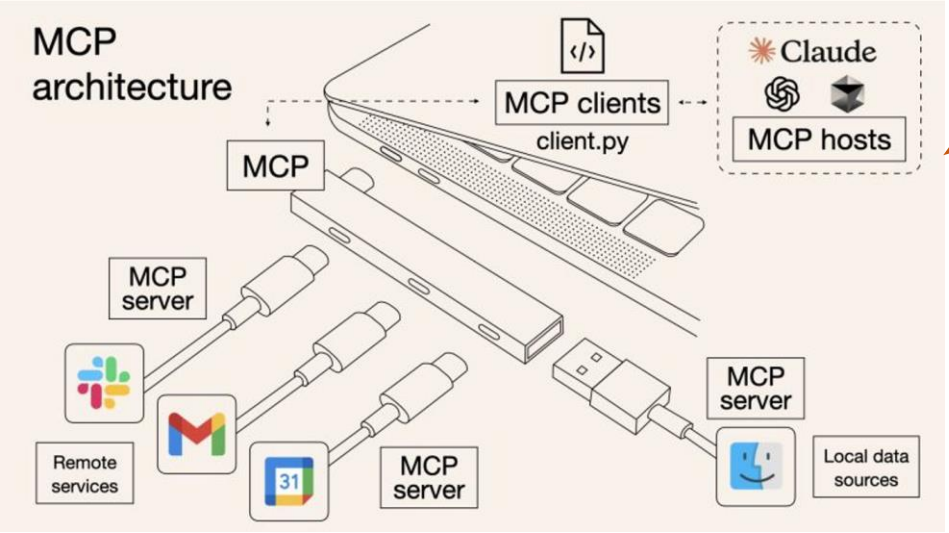


Ref. OpenAI

[Appendix] MCP (1/2)

MCP (Model Context Protocol) for Tools and Resources

- LLM 기반 AI 시스템과 외부 데이터 또는 도구를 안전하고 유연하게 연결하는 개방형 표준 프로토콜
- 오픈소스 공개 (2024.11, Anthropic)
- GitHub, OpenAI, Google, Microsoft, Cursor AI 등 MCP 지원, AI 커뮤니티의 공동작업으로 발전



AI 모델이 다양한 외부 어플리케이션, API, 데이터 소스와 표준화된 방식으로 상호 작용할 수 있도록 하는 지원하는 **Agentic AI** 핵심 인프라 요소

[Appendix] MCP (2/2)

github.com/modelcontextprotocol

Model Context Protocol

Model Context Protocol

A protocol for seamless integration between LLM applications and external data sources

Documentation | Specification | Discussions

The Model Context Protocol (MCP) is an open protocol that enables seamless integration between LLM applications and external data sources and tools. Whether you're building an AI-powered IDE, enhancing a chat interface, or creating custom AI workflows, MCP provides a standardized way to connect LLMs with the context they need.

Getting Started

- 📖 Read the Documentation for guides and tutorials
- 🔍 Review the Specification for protocol details
- 💻 Use our SDKs to start building:
  - TypeScript SDK
  - Python SDK
  - Java SDK
  - Kotlin SDK
  - C# SDK

Project Structure

- specification - Protocol specification and documentation
- typescript-sdk - TypeScript implementation
- python-sdk - Python implementation
- java-sdk - Java implementation
- kotlin-sdk - Kotlin implementation

Model Context Protocol servers

This repository is a collection of *reference implementations* for the Model Context Protocol (MCP), as well as references to community built servers and additional resources.

The servers in this repository showcase the versatility and extensibility of MCP, demonstrating how it can be used to give Large Language Models (LLMs) secure, controlled access to tools and data sources. Each MCP server is implemented with either the Typescript MCP SDK or Python MCP SDK.

> Note: Lists in this README are maintained in alphabetical order to minimize merge conflicts when adding new items.

Reference Servers

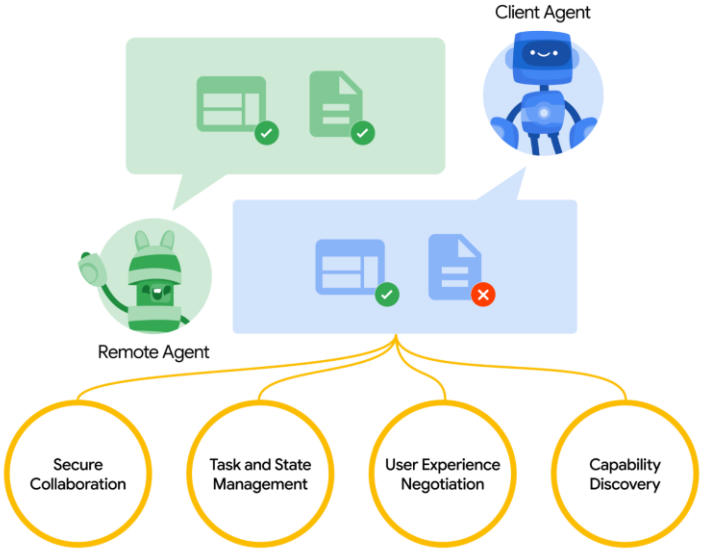
These servers aim to demonstrate MCP features and the TypeScript and Python SDKs.

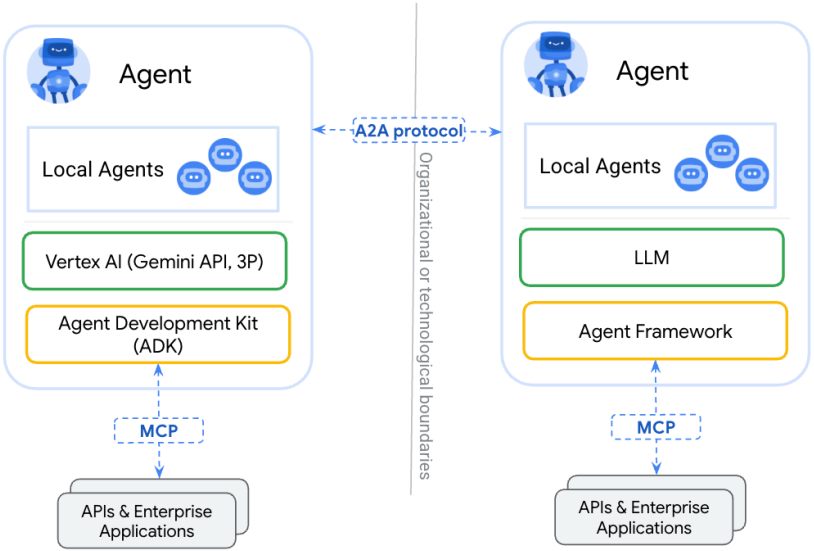
- AWS KB Retrieval - Retrieval from AWS Knowledge Base using Bedrock Agent Runtime
- Brave Search - Web and local search using Brave's Search API
- EverArt - AI image generation using various models
- Everything - Reference / test server with prompts, resources, and tools
- Fetch - Web content fetching and conversion for efficient LLM usage
- Filesystem - Secure file operations with configurable access controls
- Git - Tools to read, search, and manipulate Git repositories
- GitHub - Repository management, file operations, and GitHub API integration
- GitLab - GitLab API, enabling project management
- Google Drive - File access and search capabilities for Google Drive
- Google Maps - Location services, directions, and place details
- Memory - Knowledge graph-based persistent memory system

[Appendix] ADK / A2A (1/2)

A2A (Agent2Agent) for Agent-Agent Collaboration

- Dynamic, multimodal communication between different agents
- Available using Google ADK, LangGraph, Crew.AI



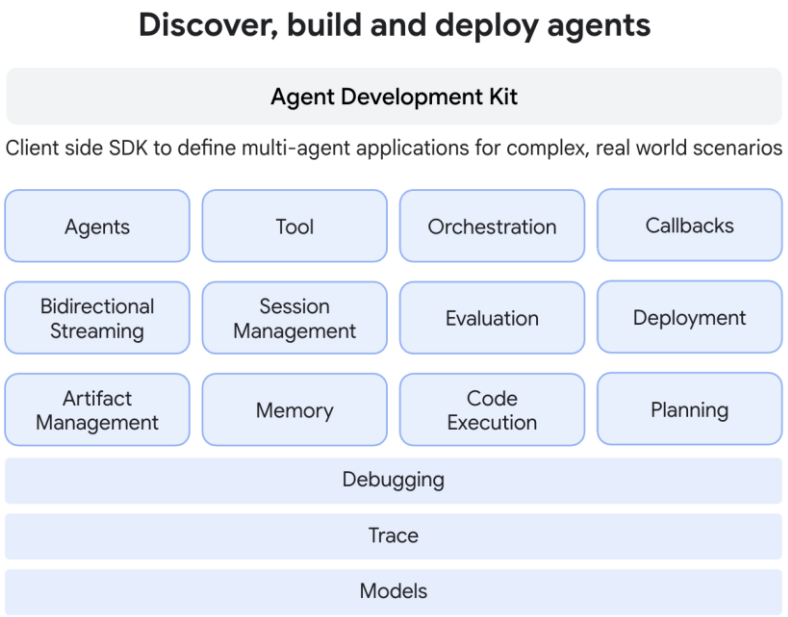


Ref. https://developers.googleblog.com/en/a2a-a-new-era-of-agent-interoperability/

[Appendix] ADK / A2A (2/2)

ADK (Agent Development Kit)

- https://google.github.io/adk-docs/



Agent2Agent

- https://github.com/google/A2A

An open protocol enabling communication and interoperability between opaque agentic applications.

- Agent2Agent Protocol A2A
  - Getting Started
  - Contributing
  - What's next
  - About

One of the biggest challenges in enterprise AI adoption is getting agents built on different frameworks and vendors to work together. That's why we created an open *Agent2Agent (A2A) protocol*, a collaborative way to help agents across different ecosystems communicate with each other. Google is driving this open protocol initiative for the industry because we believe this protocol will be **critical to support multi-agent communication by giving your agents a common language – irrespective of the framework or vendor they are built on**. With *A2A*, agents can show each other their capabilities and negotiate how they will interact with users (via text, forms, or bidirectional audio/video) – all while working securely together.

See A2A in Action

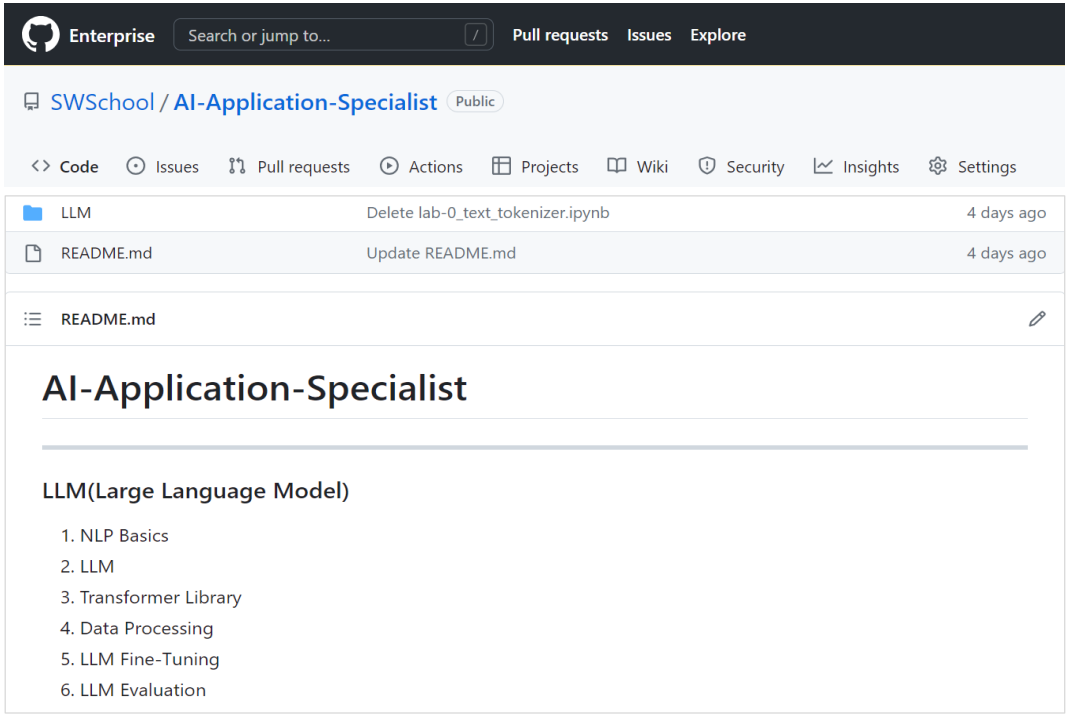
Watch this demo video to see how A2A enables seamless communication between different agent frameworks.

Conceptual Overview

The Agent2Agent (A2A) protocol facilitates communication between independent AI agents. Here are the core concepts:

[Appendix] 실습 준비

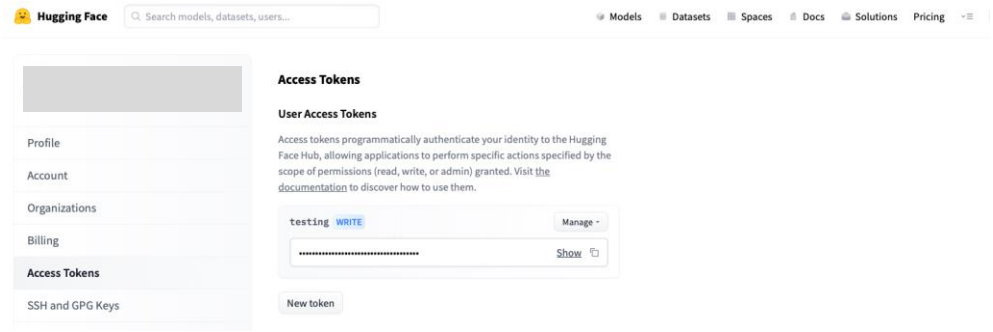
- https://github.sec.samsung.net/SWSchool/AI-Application-Specialist



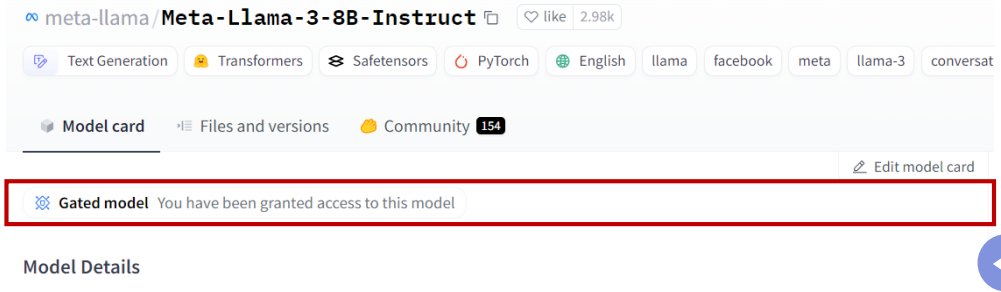
- aas_text_tokenizer.ipynb
- aas_clm_decoding_gpt.ipynb
- aas_clm_decoding_llama.ipynb
- aas_sft_sentiment_classification.ipynb
- aas_gemma-2b+LoRA.ipynb
- aas_gemma-2b-it+LoRA+DPO.ipynb
- aas_transformer_architecture.ipynb
- aas_llama-3-8B-instruct_evaluation.ipynb
- aas_gemma-3-4b-it_evaluation.ipynb

[Appendix] 실습 준비

- **Hugging Face 계정 생성**
  (https://huggingface.co/)
- **Access Token 생성***
  - Settings > Access Tokens



- **Google Gemma, Meta Llama 사용 신청**
  - https://huggingface.co/google/gemma-1.1-2b-it
  - https://huggingface.co/meta-llama/Meta-Llama-3-8B-Instruct



※ Access Key 생성시 Repositories Permissions에 Model 선택 추가