Prof. Sangwoo Mo's AI Study

Multimodal Perceptron

: BERT & CLIP

20240505 Hyunseong Kong
20240600 Jian Choi

2025/10/29

Chapter 1: BERT

Pre-training of Deep Bidirectional Transformers for Language Understanding

[1] Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2018). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding.
[2] Sudharsan Ravichandiran. (2021). Getting Started with Google BERT. Packt Publishing.
[3] https://medium.com/@shaikhrayyan123/a-comprehensive-guide-to-understanding-bert-from-beginners-to-advanced-2379699e2b51

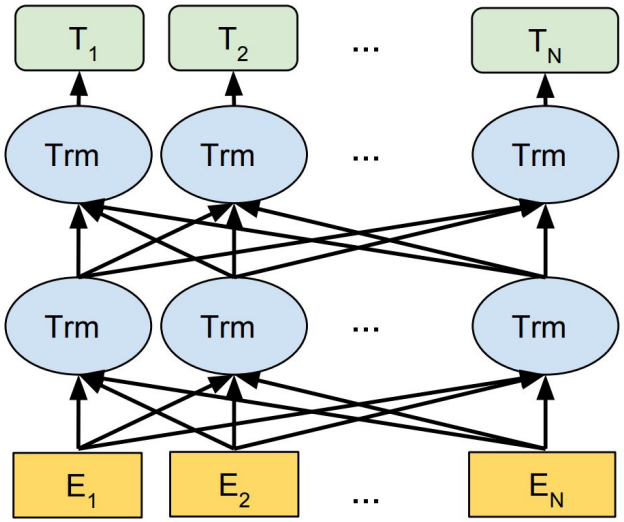
Contents of BERT

- Introduction: What is BERT?
- BERT's Preprocessing
- BERT's Training Strategies
- Common Challenges and Some Mitigations
- Recent Developments and Variants
- Practice: Implementing a Tiny-Headed BERT

Introduction: What is BERT?

BERT

- **BERT** is the unimodal, pretrained model for the purpose of interpreting human languages.

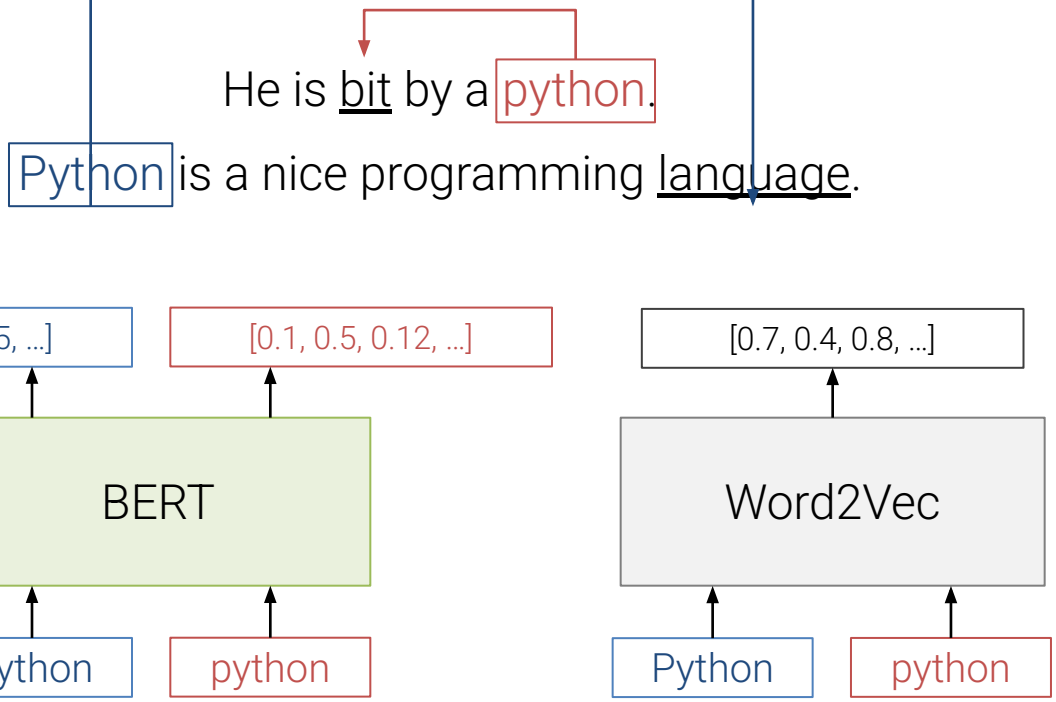


Although it is **not multimodal**, it can be extended to have an embedded image vector to its inputs. Ex) VisualBERT

- The main purpose of the BERT is to understand natural language (NLU). So, it can be varied wherever the language process is needed.

BERT vs Traditional Model

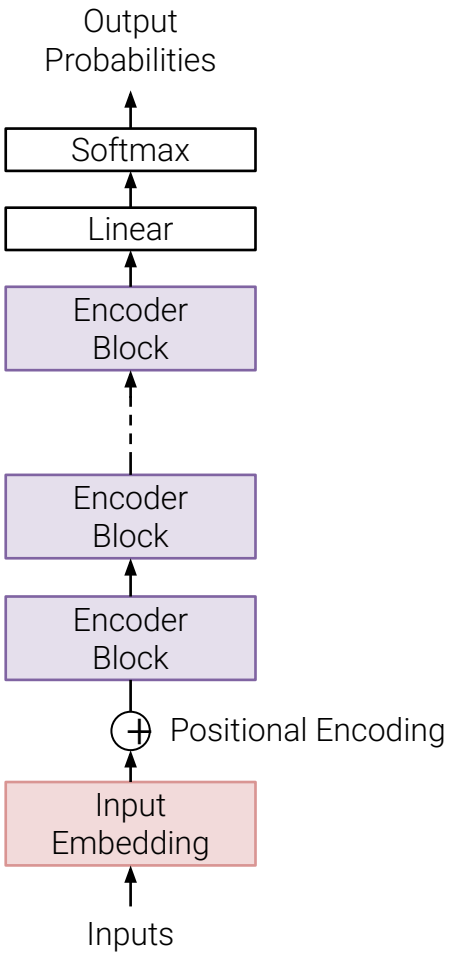
- BERT is context sensitive, while the traditional model like Word2Vec doesn't.



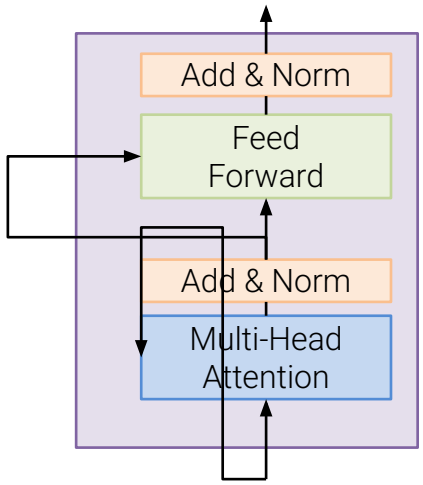
Those two words are textually the same but has different meaning because of the context.

How does BERT works?

- BERT is basically powered by the Transformer architecture.



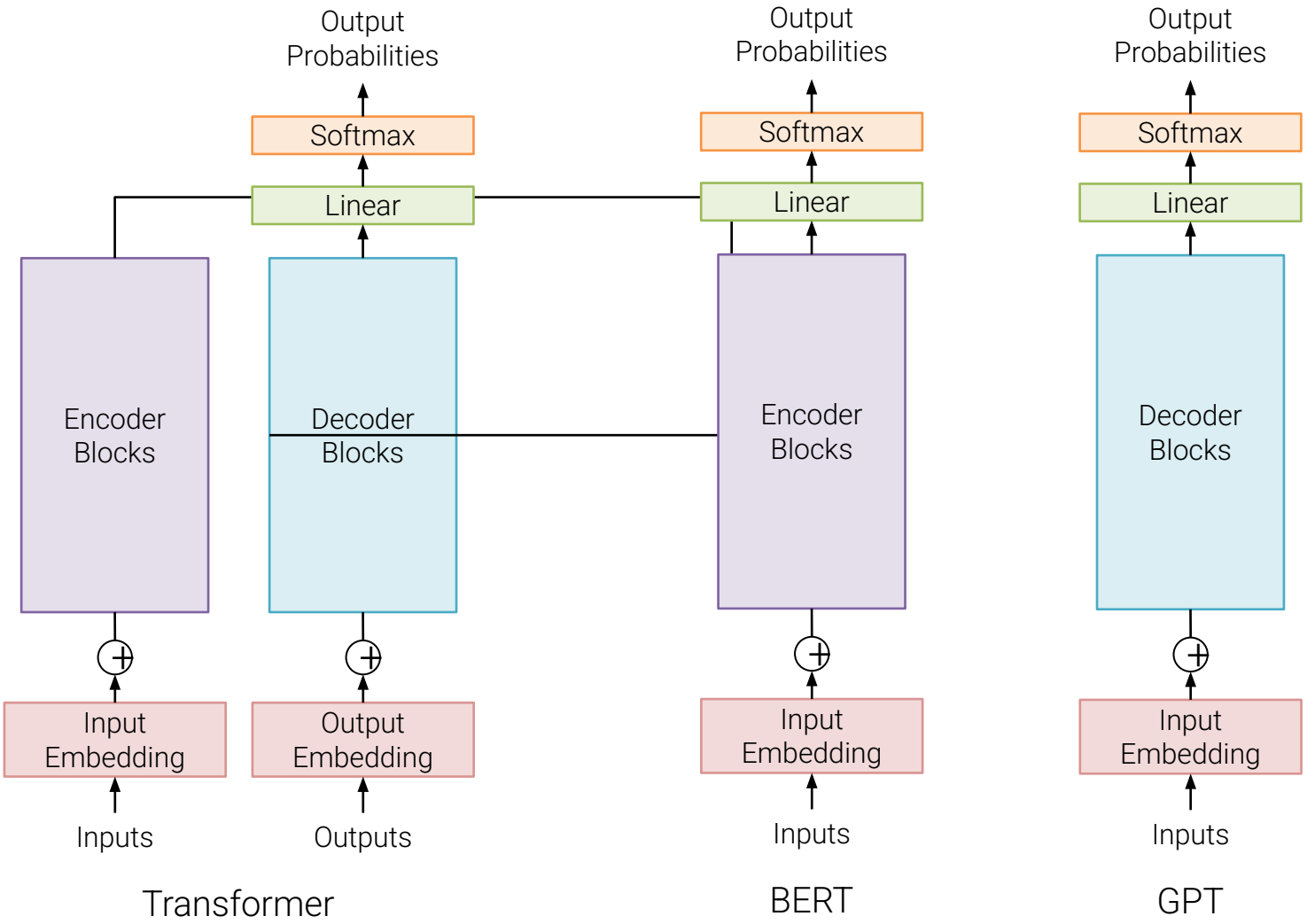
BERT
(Encoder-only)



Encoder Block

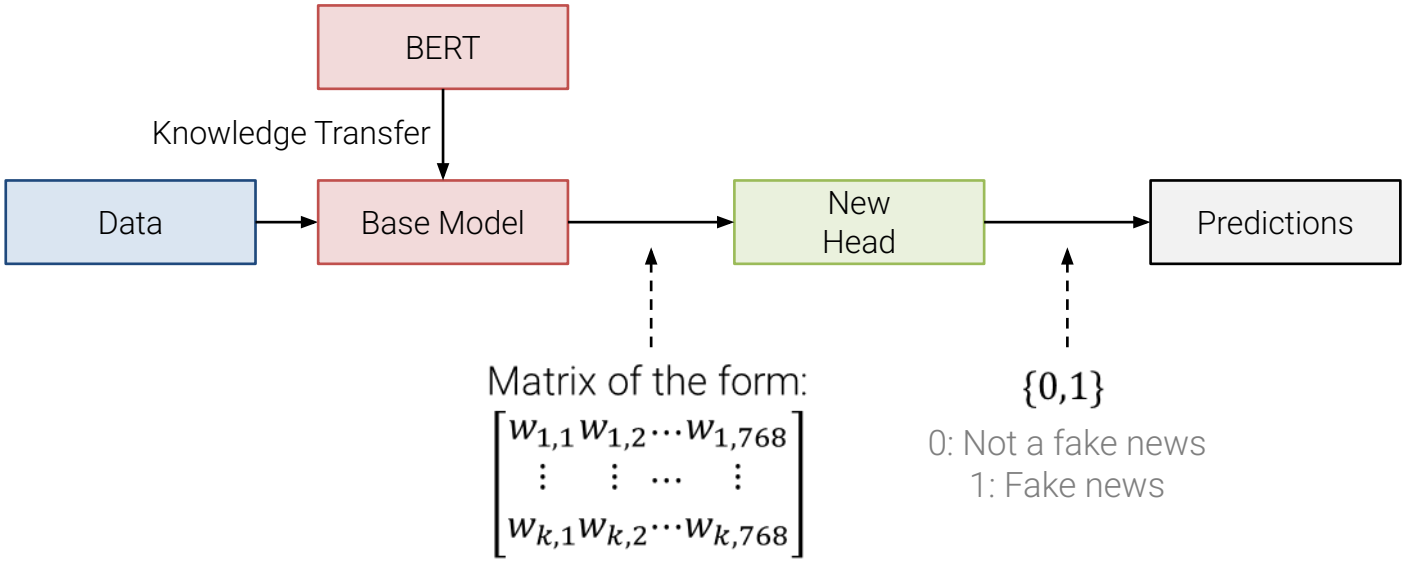
Recall: Transformer

- The Transformer architecture, a state-of-the-art model, includes both an encoder and a decoder.



Apply BERT to an NLP Use-Case

- You can apply BERT in any cases that needs natural language processing.



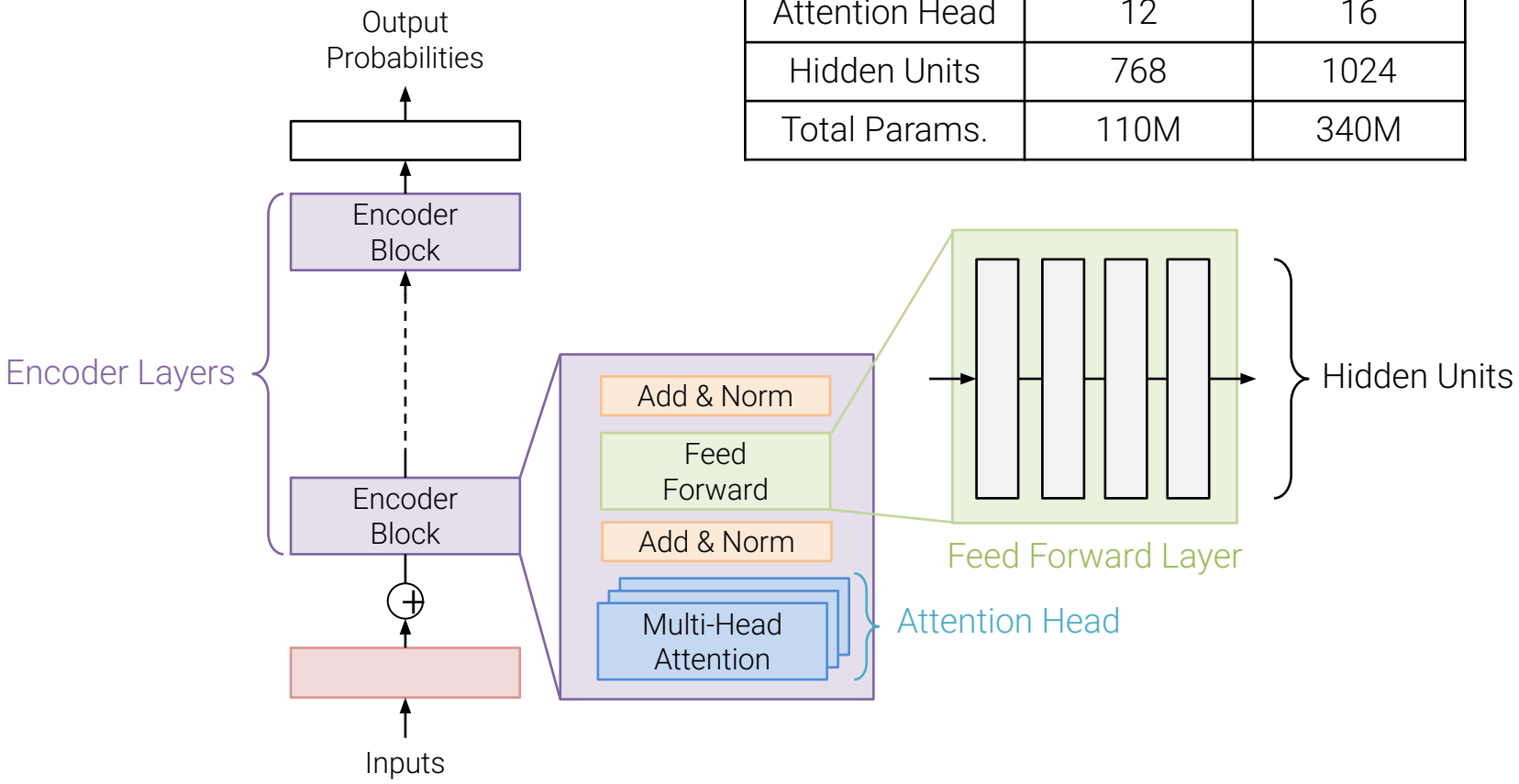
Fake News Detection using BERT

- The loss functions might be chosen appropriately based on the Structure of the Head.
- You may fix the parameters of the Base Model so the pre-trained BERT is unchanged. But it is recommended not to do it.

Configurations of BERT

- There are several models based on the # of parameters.

	BERT-base	BERT-large
Encoder Layers	12	24
Attention Head	12	16
Hidden Units	768	1024
Total Params.	110M	340M



BERT’s Preprocessing

Tokenization

- BERT uses WordPiece tokenization

Original Text
"ChatGPT is fascinating."

WordPiece Tokens
["Chat", "##G", "##PT", "is", "fascinating", "."]

Splits words into smaller pieces

This helps handle tricky words and ensures that BERT doesn't get lost in unfamiliar words.

- Also, there are some special tokens:

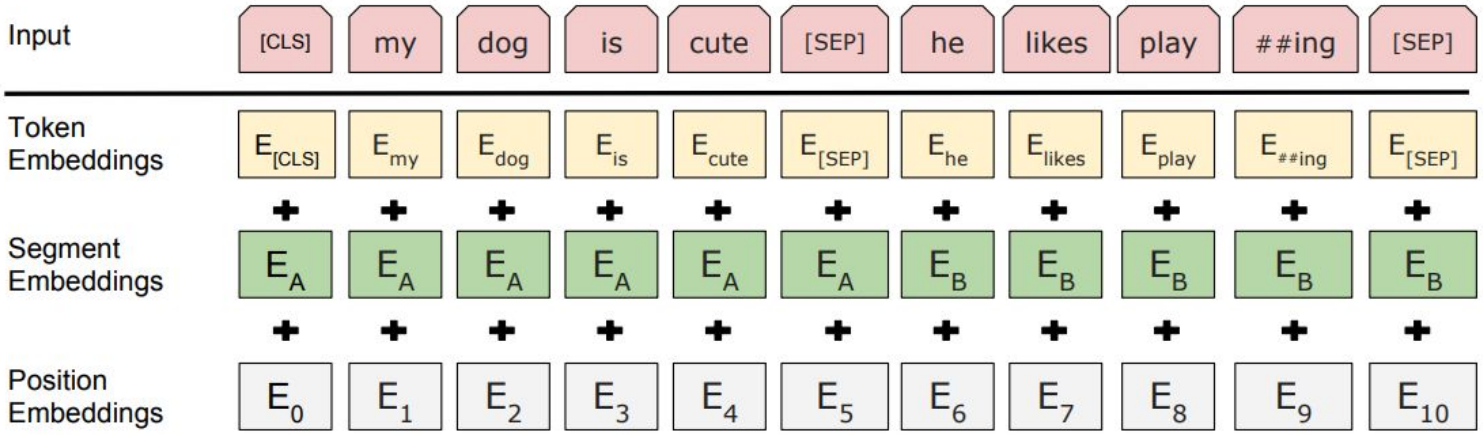
"[SEP]" : stands for separation
"[CLS]" : stands for classification

With Special Tokens
["[CLS]", "Chat", "##G", "##PT", "is", "fascinating", ".", "[SEP]"]

- Note that the [CLS] token is added only at the beginning of the first sentence, while the [SEP] token is added at the end of every sentence.

Other Embeddings

- BERT uses Position Embedding & Segment Embeddings



- Segment Embedding: To check if the token contains in the first sentence or the second sentence. (E_a or E_b)
- Position Embedding: Add the position information.
- After the embedding, the size of the input vector is fixed to $B \times L \times 768$ where $L \leq 512$ and B is the batch size.

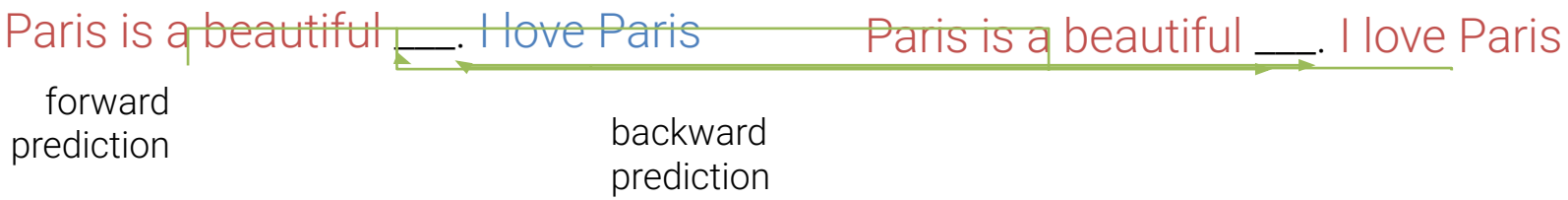
BERT’s Training Strategies

Language modeling

- We can categorize language modeling into two aspects:
  - Auto-regressive language modeling
  - Auto-encoding language modeling

- Auto-regressive
  - Unidirectional

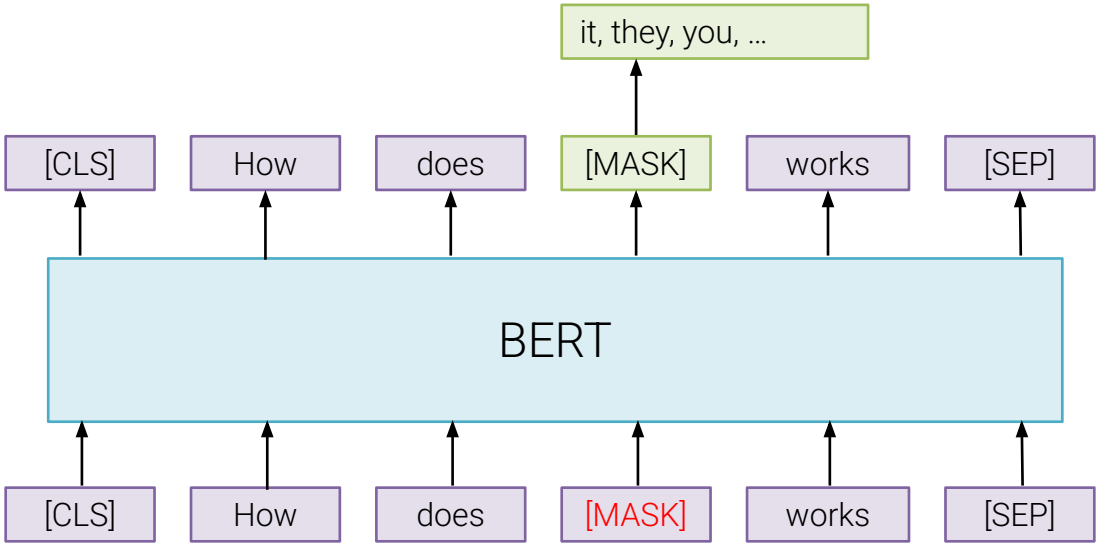
- Auto-encoding
  - Bidirectional



- BERT is auto-encoding language model.

Masked Language Model (MLM)

- During the training of BERT, 15% of the words are masked.
- BERT learns to predict the correct word for the masked position.

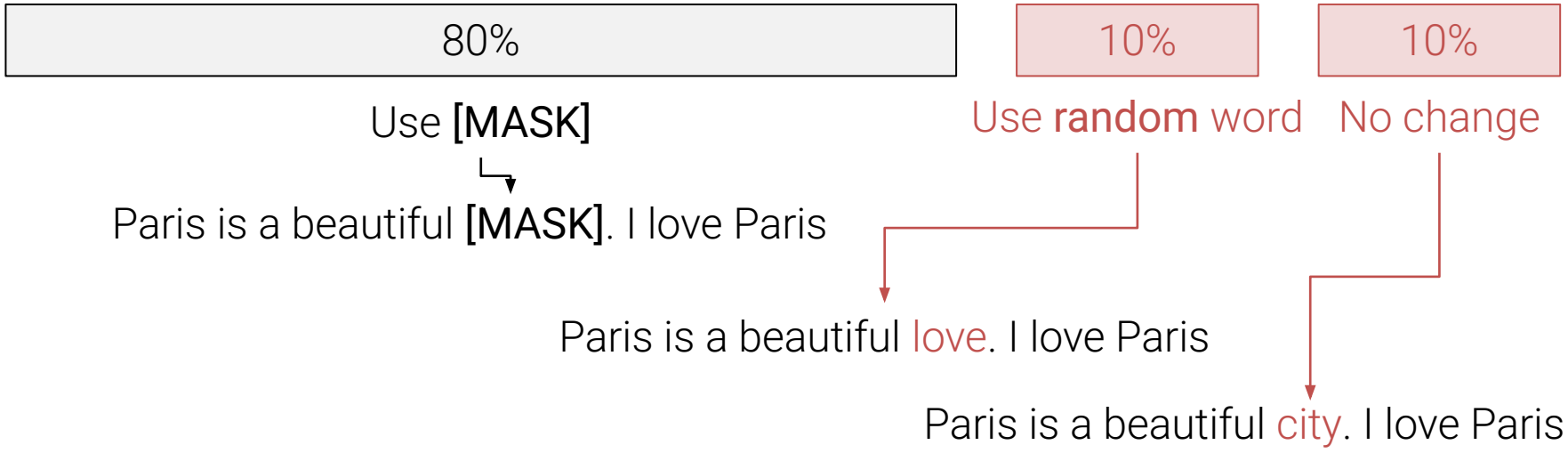


Masked Language Model (MLM)

Masked Language Model (MLM)

- When BERT is fine-tuned, the [MASK] token will not used at all.
  - Mismatch between pre-training & fine-tuning.

- 80-10-10% rule:



Masked Language Model (MLM)

- Dynamic Masking: Different masking positions on each epoch.
  - Static Masking: Same masking positions on each epoch. (Original BERT)

- Whole-Word/Span Masking

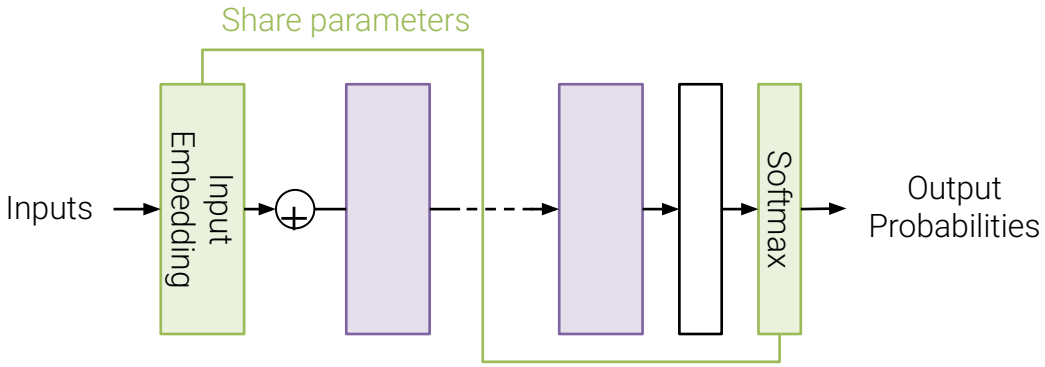
Basic Masking playing → play + ##ing → play + [MASK]

Might be affected

Whole-Word Masking playing → play + ##ing → [MASK] + [MASK]

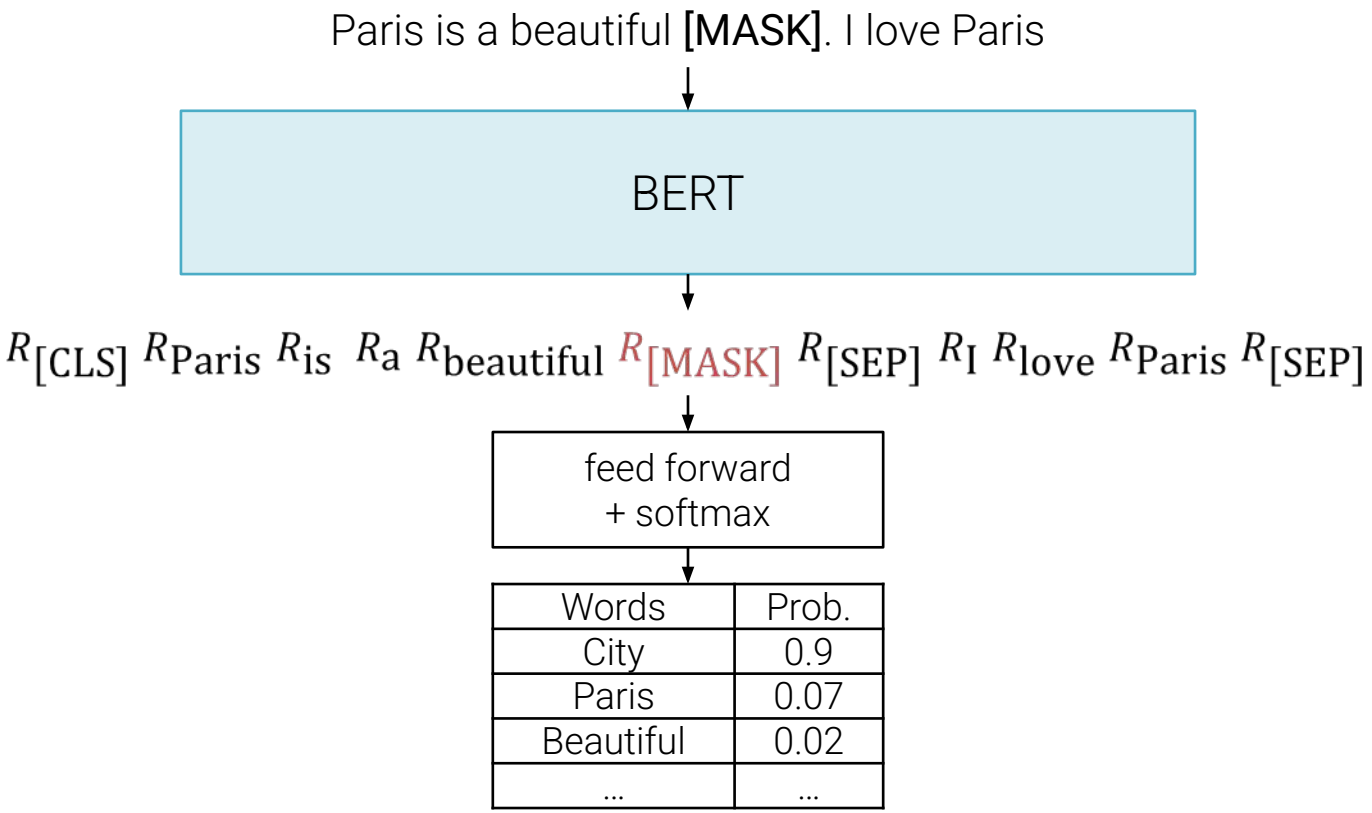
Span Masking: More general version. Mask the whole phrase.

- Embedding Weight Tying
  - Share parameters between "Input Token Embedding Layer" & "Output Softmax Layer"



Masked Language Model (MLM)

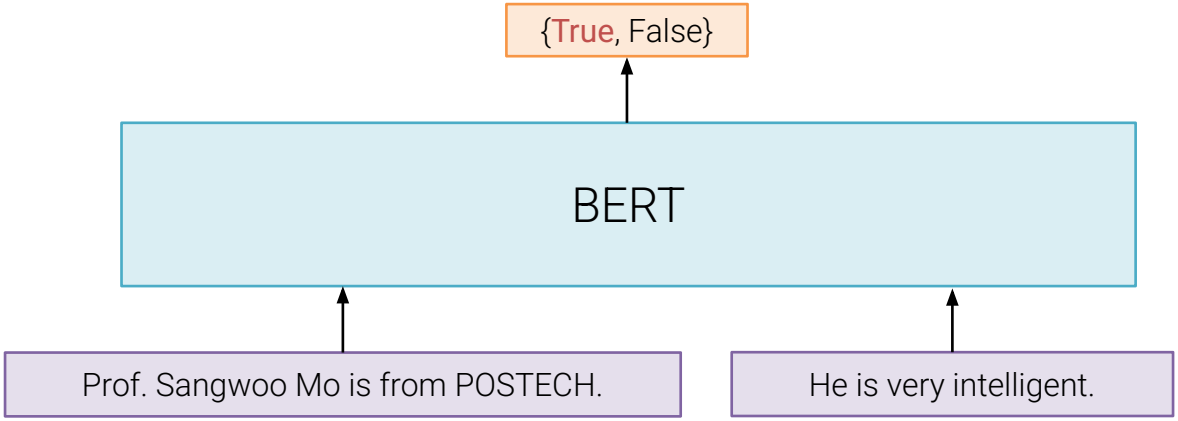
- How to predict the [MASK] token?
- Feed the representation of the masked token $R_{[\text{MASK}]}$ into a feedforward network with a softmax activation



- In the initial step, the model will not generate the appropriate probability distribution.
- After a series of iterations, with backpropagation, it will.

Next Sentence Prediction (NSP)

- Given two sentences, predict whether the two sentences are connected in the original content.

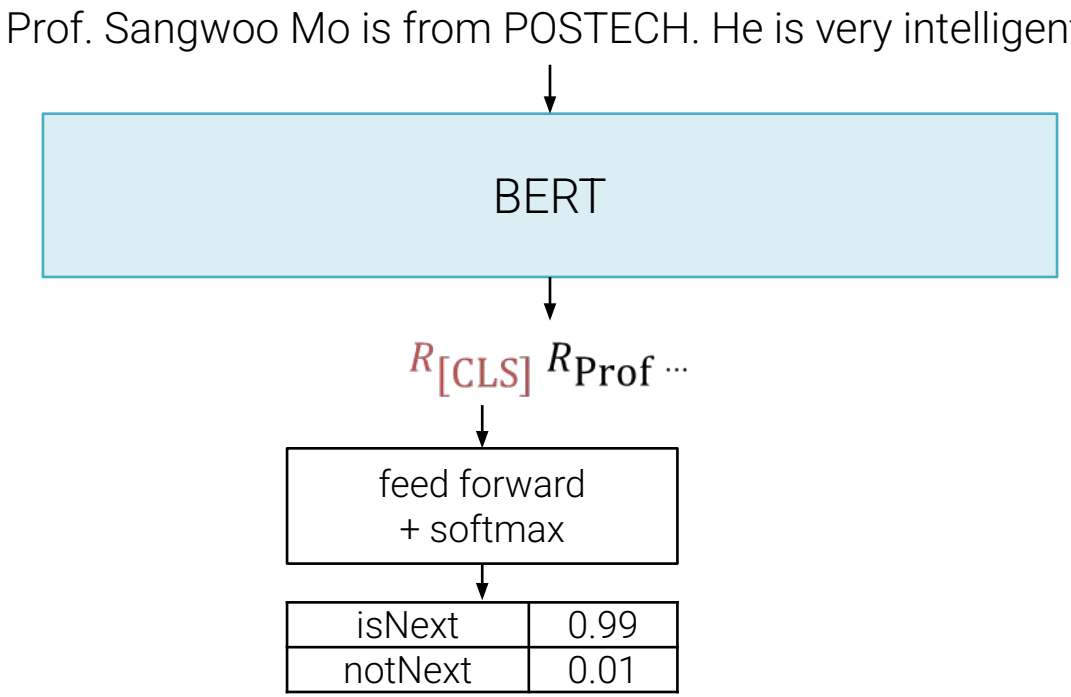


Next Sentence Prediction (NSP)

- Since this one is provided in the BERT paper, future research found that NSP didn't contribute to model performance. (Topic-shift)

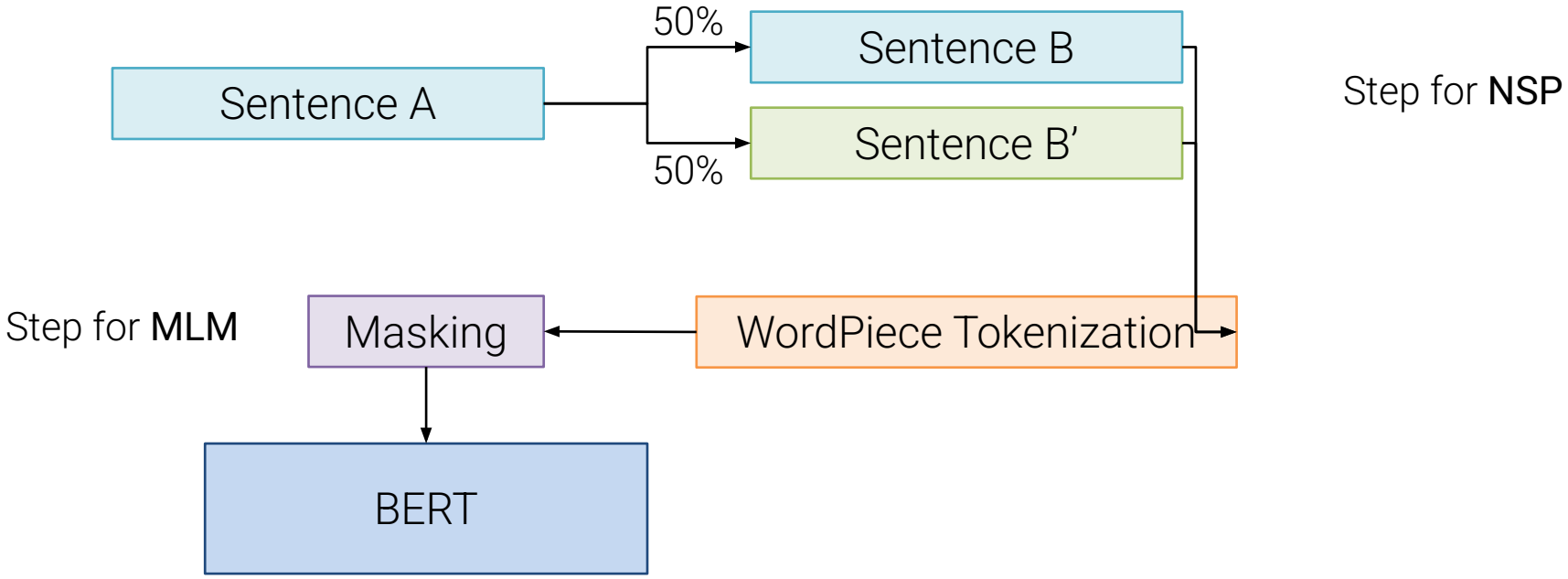
Next Sentence Prediction (NSP)

- How to classify if the two sentences are “isNext”? → Use [CLS]
  - The [CLS] token basically holds the aggregate representation of all the tokens.



Train Specifications

- BooksCorpus (800M) + English Wikipedia (2,500M) to train.

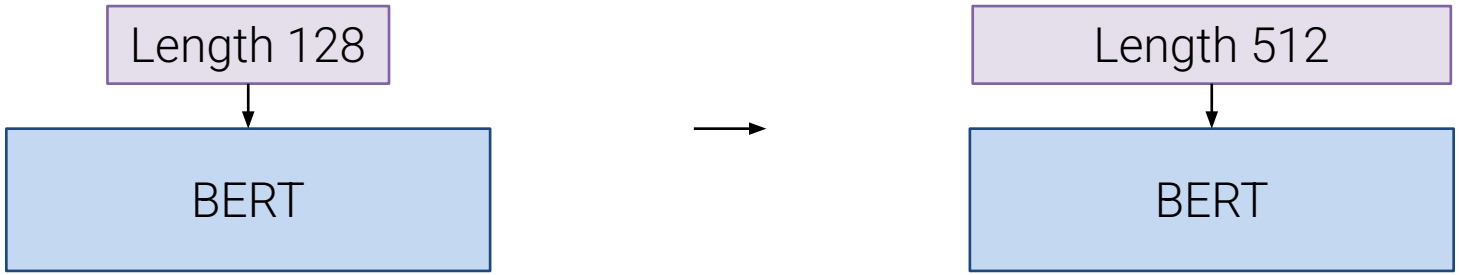


Data preparation for training BERT

- 256 sequences per batches of total 1,000,000 steps.
- Adam optimizer with **warm-up step** set to 10,000.
- 0.1 dropout applied to every layers.

Train Specifications

- Length Scheduling
  - Initial step → Use short length sentences($L = 128$)
  - Later step → Change the input sentences length $L = 512$



Learn the basic language pattern efficiently

Increase the model performance

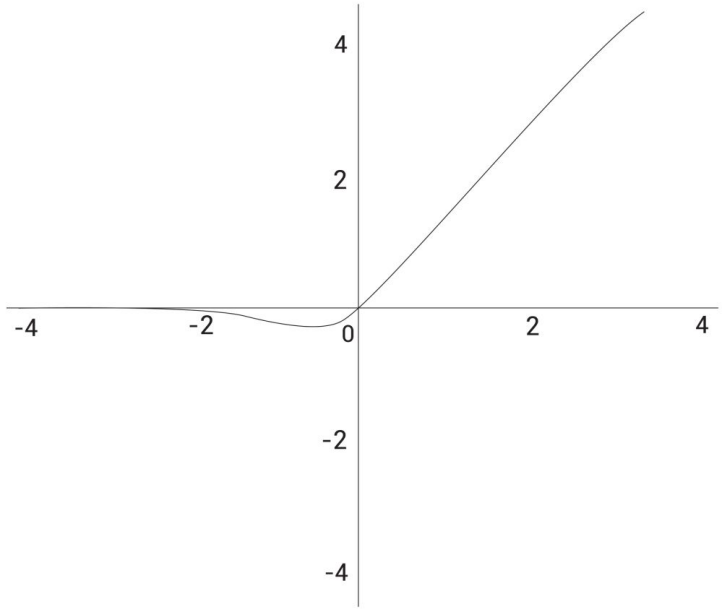
- Fine-tuning Guide
  - Use small learning rate
  - Use only 2 ~ 4 epochs to prevent overfitting

Train Specifications

- GELU(Gaussian Error Linear Unit) for activation function.

$$\mathrm{GELU}(x) = x\Phi(x)$$

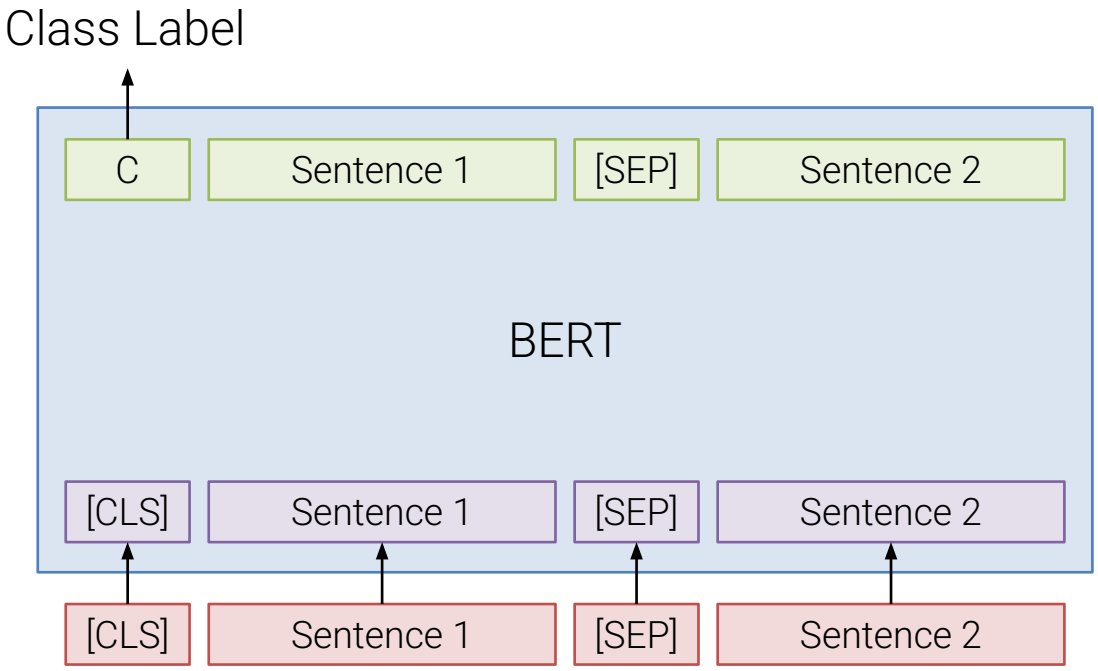
Here, $\Phi(x)$ is the standard Gaussian cumulative distribution function.



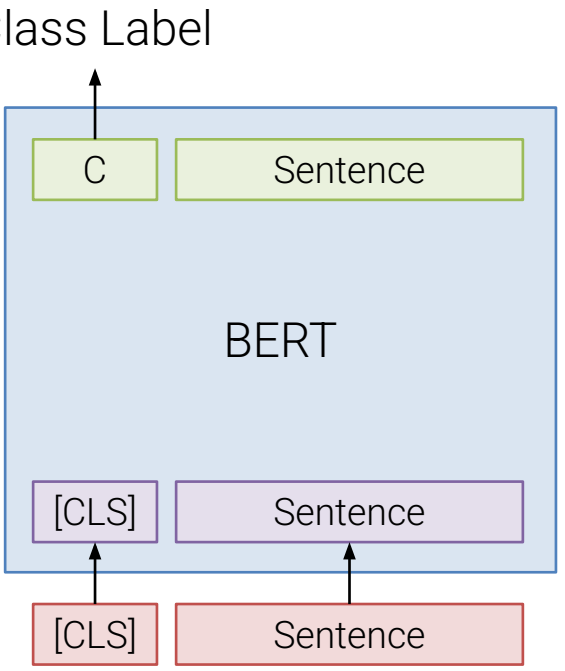
$$GELU(x) \approx 0.5x\left(1 + tanh\left(\sqrt{\frac{2}{\pi}}(x + 0.044715x^3)\right)\right)$$

Fine-tuning BERT

- The BERT paper outlines four methods for fine-tuning.
- One of these involves sharing (or freezing) the core BERT parameters and only modifying the structure of the output layer.



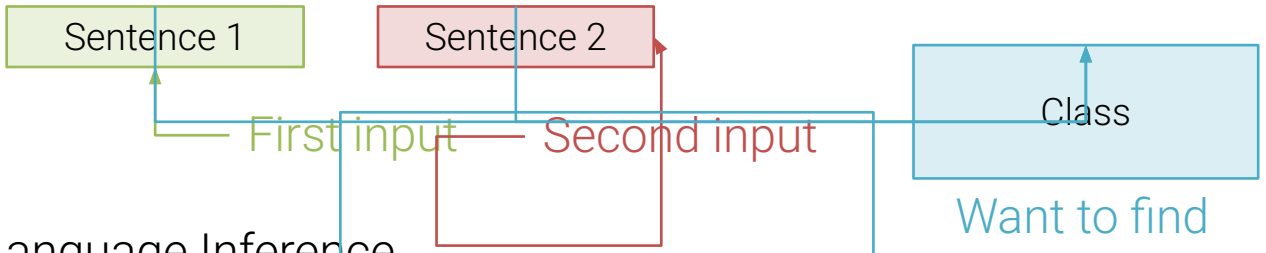
Sentence Pair Classification



Single Sentence Classification

Fine-tuning BERT

- Sentence Pair Classification
  - Given two sentences, classify the relation of the two sentences



(1) Natural Language Inference
(2) Semantic Textual Similarity
(3) Duplicate Question Detection

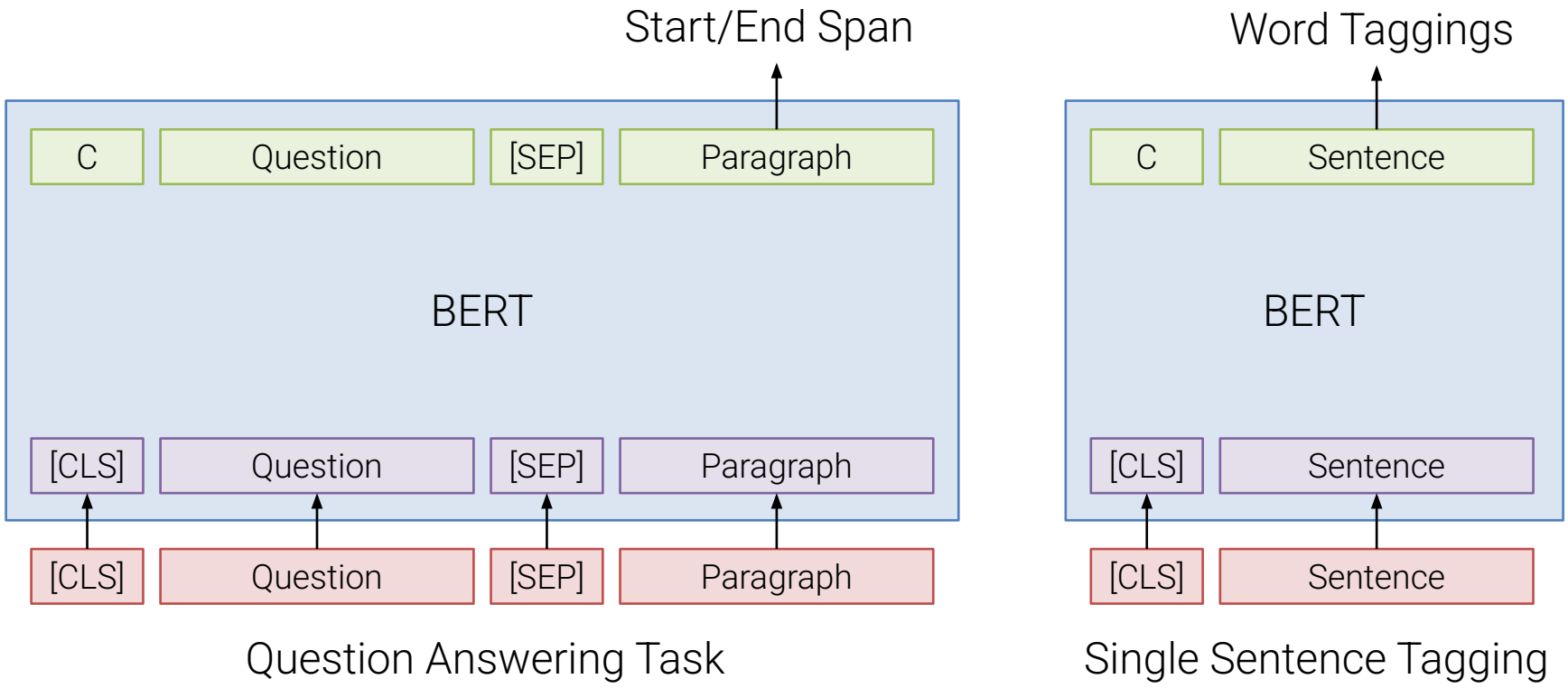
- Single Sentence Classification
  - Given one sentence, classify that sentence



(1) Sentiment Analysis
(2) News Category Classification
(3) Spam Detection

Fine-tuning BERT

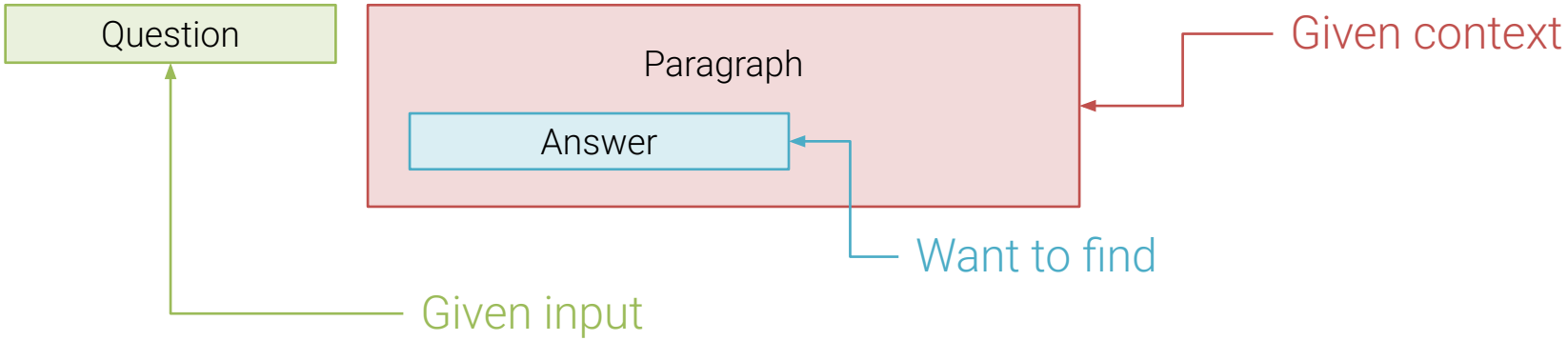
- Freeze the encoder's weights, design an additional Fully Connected Layer.



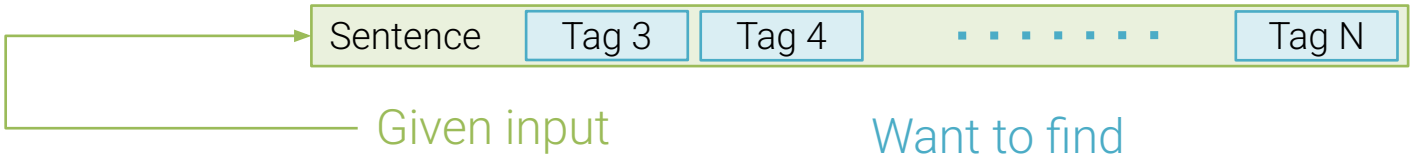
- In actual fine-tuning, it's more common not to completely freeze the BERT weights. Instead, the entire model is fine-tuned using a very small learning rate.

Fine-tuning BERT

- Question Answering Task
  - Given a question and a context, find the answer from the given context.



- Single Sentence Tagging
  - Given one sentence, tag each words

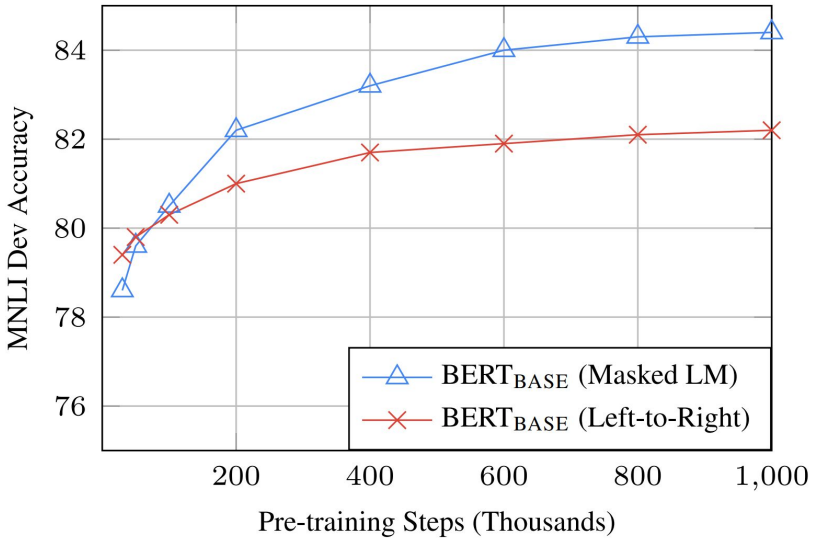


(1) Named Entity Recognition
(2) Part-of-Speech Tagging(POS)

Common Challenges and Some Mitigations

Bi-directional vs Left-to-Right

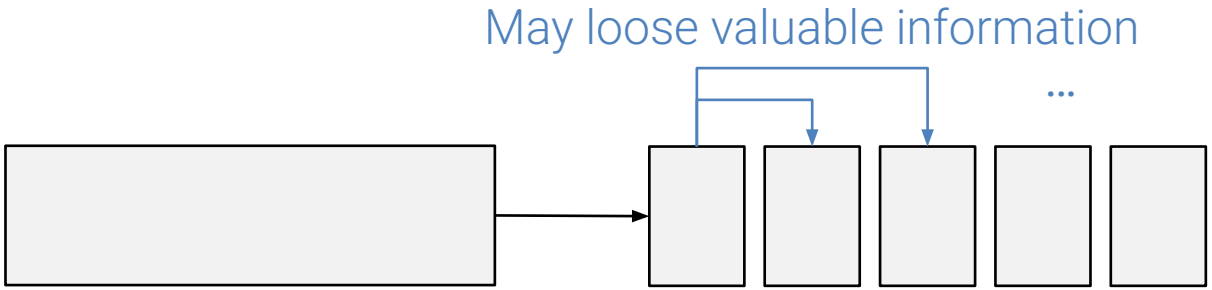
- In the paper, the bidirectional model BERT beat the unidirectional model GPT in performance.



- But, researchers today want to study GPT, not BERT.
  - Massively scaled decoder models exhibit "emergent abilities“, which represent the current frontier of AI research.
  - GPT's ability to perform new tasks with zero or few examples (few-shot learning) makes it a far more flexible and general-purpose platform than task-specific models like BERT.

Dealing with Long Texts

- BERT has a maximum token limit for input, and long texts can get cut off.
- Mitigation:

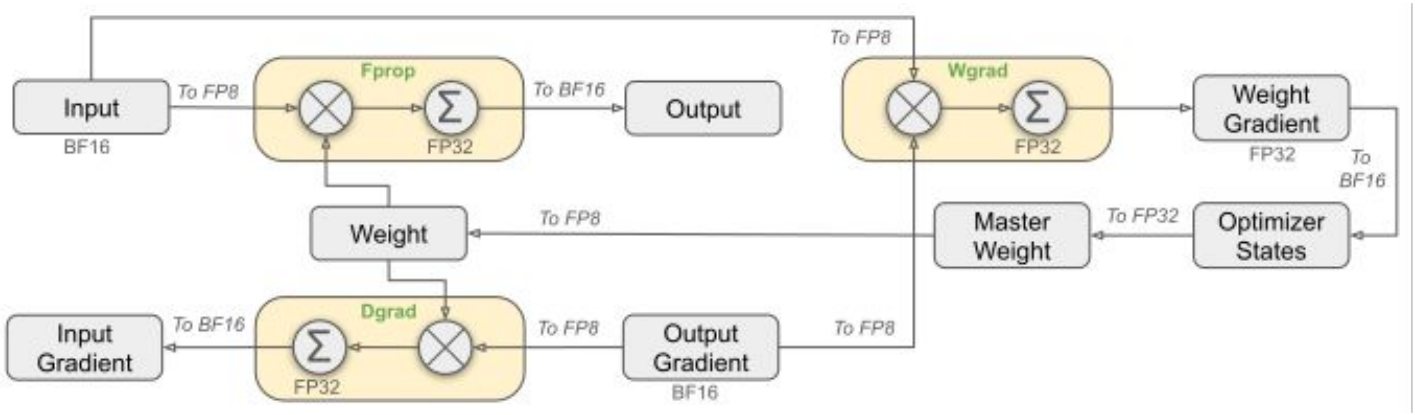


Split the text into manageable chunks.

- Need to carefully manage the context between these chunks to ensure meaningful results.

Resource Intensive Computation

- BERT models, especially the larger ones, can be computationally demanding.
- Mitigation:



mixed-precision training in DeepSeek V3

- Deep learning models store and process parameters in the FP32 format.
- Mixed-precision training switches a significant portion of this to be processed using FP16.

Domain Adaptation

- BERT might not perform optimally in certain domains.
- Mitigation: Fine-tune BERT on domain-specific data.
  - By exposing it to text from the target domain, BERT will learn to understand the nuances and terminology specific to that field.

Recent Developments and Variants

ALBERT

- ALBERT stands for “A Lite version of BERT”.
- Designed to be efficient, using parameter-sharing techniques to reduce memory consumption.

1. Factorized Embedding Parameterization
2. Cross-Layer Parameter Sharing

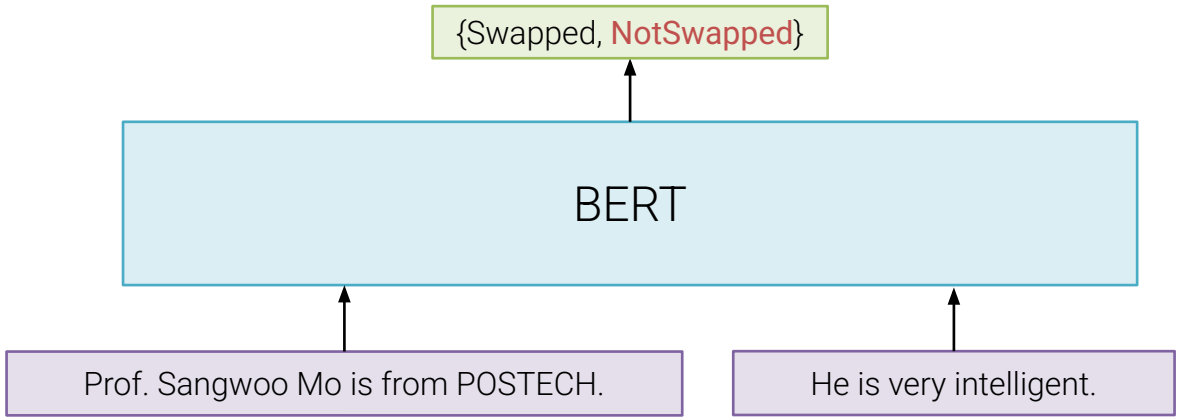
parameter optimization

3. **Sentence Order Prediction (SOP)**

learning optimization

Sentence Order Prediction (SOP)

- Given two sentences, predict whether the given sentence pair is swapped or not.



Sentence Order Prediction (SOP)

- In NSP, the model can easily distinguish negative examples by noticing the shift in topic between the two segments.
- SOP addresses this limitation by controlling for the topic.

Other BERT-family

- RoBERTa: Beyond BERT's Basics
  - BERT's clever sibling. It's trained with a more thorough recipe, involving larger batches, more data, and more training steps.
  - This enhanced training regimen results in even better language understanding and performance across various tasks.
- DistilBERT: Compact Yet Knowledgeable
  - Distilled version of BERT. It's trained to mimic BERT's behavior but with fewer parameters.
  - This makes DistilBERT lighter and faster while still retaining a good portion of BERT's performance.
- ELECTRA: Efficiently Learning from BERT
  - Introduces an interesting twist to training. Instead of predicting masked words, ELECTRA trains by detecting whether a replaced word is real or artificially generated.
  - This efficient method makes ELECTRA a promising approach for training large models without the full computational cost.
- VisualBERT: Multimodal embeddings by feeding [text, image] into the BERT.

Practice: Implementing a Tiny-Headed BERT

A Tiny-Headed BERT

- In this context, we will use the HuggingFace Transformers library to build a "toy head" for BERT and perform sentiment analysis.

```
!pip install transformers datasets accelerate
```

```
import torch
from datasets import load_dataset
from transformers import (
    AutoTokenizer,
    AutoModelForSequenceClassification,
    TrainingArguments,
    Trainer
)
import numpy as np
from sklearn.metrics import accuracy_score, f1_score
```

https://colab.research.google.com/drive/1xBfiG8JtEu3vKIU7PgL1G-sj11d3_MEo

A Tiny-Headed BERT

- Preprocessing

```
import csv
import re

def preprocess_text_simple(text):
    normalized_text = re.sub(r',{2,}', ',', text)
    normalized_text = re.sub(r'\.{4,}', '...', normalized_text)
    normalized_text = re.sub(r' {2,}', ' ', normalized_text)
    return normalized_text.strip()

# 입출력 파일 이름 설정
input_filename = 'ratings_train.txt'
output_filename = 'output_simple_processed.csv'

# 파일을 읽어 전처리 후 csv로 저장
try:
    with open(input_filename, 'r', encoding='utf-8') as infile, \
         open(output_filename, 'w', encoding='utf-8', newline='') as outfile:

        tsv_reader = csv.reader(infile, delimiter='\t')
        csv_writer = csv.writer(outfile)

        # 헤더 처리
        header = next(tsv_reader)
        csv_writer.writerow(['document', 'label'])

        print("전처리 시작")
        for row in tsv_reader:
            original_text = row[1]
            processed_text = preprocess_text_simple(original_text)
            csv_writer.writerow([processed_text, row[2]])
    print("파일 처리 완료")
```

Preprocess using regular expression

A Tiny-Headed BERT

- Load Datasets. Select 5000 of them. (Speed Issue)

```
loaded_dataset = load_dataset("csv", data_files="data.csv", split="train")
sampled_dataset = loaded_dataset.select(range(5000))

dataset = sampled_dataset.train_test_split(test_size=0.2)

train_dataset = dataset["train"]
eval_dataset = dataset["test"]
```

train/eval set separation

- Select pretrained BERT.

```
model_name = "klue/bert-base"
tokenizer = AutoTokenizer.from_pretrained(model_name)

# num_labels=2 (긍정/부정)로 이진 분류 헤드 추가
model = AutoModelForSequenceClassification.from_pretrained(model_name, num_labels=2)
```

Use [CLS] token. Single layer with size 768.

A Tiny-Headed BERT

- Dataset Tokenization

```
def tokenize_function(examples):
    texts = [str(text) for text in examples["document"]]
    return tokenizer(texts, padding="max_length", truncation=True)

train_dataset = train_dataset.map(tokenize_function, batched=True)
eval_dataset = eval_dataset.map(tokenize_function, batched=True)
```

A Tiny-Headed BERT

- Set a training hyperparameters.

```
def compute_metrics(p):
    preds = np.argmax(p.predictions, axis=1)
    return {
        "accuracy": accuracy_score(p.label_ids, preds),
        "f1": f1_score(p.label_ids, preds, average="weighted"),
    }

training_args = TrainingArguments(
    output_dir="./results_korean",
    num_train_epochs=2, # 에폭 수를 줄여서 빠르게 테스트하기
    per_device_train_batch_size=16,
    per_device_eval_batch_size=16,
    logging_dir="./logs_korean",
    logging_steps=10,
    eval_strategy="epoch"
)
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_dataset,
    eval_dataset=eval_dataset,
    compute_metrics=compute_metrics,
)
```

[default parameters]
learning_rate=5e-5 (0.00005)
weight_decay=0
lr_scheduler_type="linear"
warmup_steps=0

A Tiny-Headed BERT

- Training

```
# 모델 학습 시작
trainer.train()
```

[500/500 13:19, Epoch 2/2]

Epoch	Training Loss	Validation Loss	Accuracy	F1
1	0.354200	0.327351	0.866000	0.865847
2	0.228000	0.375380	0.878000	0.877919

```
TrainOutput(global_step=500,
training_loss=0.295131293177604 67, metrics=
{'train_runtime': 800.6326,
'train_samples_per_second': 9.992,
'train_steps_per_second': 0.625, 'total_flos':
2104888442880000.0, 'train_loss':
0.295131293177604 67, 'epoch': 2.0})
```

A Tiny-Headed BERT

- Test the results.

```
text = "이 영화 배우들 연기력이 정말 대단하네요. 몰입해서 봤어요!"
inputs = tokenizer(text, return_tensors="pt")

if torch.cuda.is_available():
    inputs = {k: v.to(model.device) for k, v in inputs.items()}

with torch.no_grad():
    outputs = model(**inputs)

logits = outputs.logits
predicted_class_id = torch.argmax(logits, dim=-1).item()

# NSMC 데이터셋의 레이블: 0 (부정), 1 (긍정)
labels = ["부정", "긍정"]
print(f"입력 텍스트: '{text}'")
print(f"예측 결과: {labels[predicted_class_id]}")
```

입력 텍스트:'이 영화 배우들 연기력이 정말 대단하네요. 몰입해서 봤어요!'
예측 결과: 긍정

창창한 미래

https://blog.google/products/search/search-language-understanding-bert/
구글 검색은 BERT가 담당하고 있음. 사용자 쿼리에서 다의어를 능숙하게 처리함.

Applying BERT models to Search

Last year, we introduced and open-sourced a neural network-based technique for natural language processing (NLP) pre-training called Bidirectional Encoder Representations from Transformers, or as we call it--BERT, for short. This technology enables anyone to train their own state-of-the-art question answering system.

https://huggingface.co/docs/transformers/main/en/model_doc/modernbert
ModernBERT: 2024년 12월 18일에 나온 BERT의 개량판

창창한 미래?

왜 요즘은 BERT를 잘 안쓸까?

(1) Zero-Shot/Few-Shot 학습
BERT는 작업에 특화되게 Fine-tuning 해야 됨.
GPT는 그냥 프롬프트 엔지니어링 하면 됨.

(2) BERT는 보조도구 느낌
인코더 아키텍처는 이제 BERT라는 이름보다, RAG(검색 증강 생성) 시스템의 검색 모듈이나 경량화 모델 분야로 세분화되어 연구가 진행됨. BERT 단일 모델의 효율을 개선하는 건 이제 안하고, LLM 기반 시스템에 통합하는 연구가 더 많아짐.

(3) GPT 계열의 LLM이 훨씬 할 거 많음. 그래서 연구가 거기에 집중됨
GPT-3 이후, 모델의 크기를 일정 수준 이상으로 키우면 예측하지 못했던 능력(복잡한 추론이나 단계별 문제 해결 등)이 갑자기 나타난다는 창발성(Emergence)이 연구됨. 일단 흥미로움.

(4) 최신 트렌드 멀티모달(Multi-modal) 연구랑 안맞음
GPT-4o, Gemini와 같은 최신 모델은 멀티모달 기능이 핵심. 인코더 전용 아키텍처는 이런 복합적인 생성 및 이해 작업을 수행하기 어려움.

Chapter 2: CLIP

Contrastive Language-Image Pre-training

[1] Alec Radford et al. (2021). Learning Transferable Visual Models From Natural Language Supervision.
[2] Alex Krizhevsky et al. (2012). ImageNet Classification with Deep Convolutional Neural Networks.
…

Contents of CLIP

- Introduction: What is CLIP? And Why CLIP?
- Contrastive Learning
- Model Architecture
- Utilizing CLIP
- Limitations of CLIP
- Practice: Semantic Image Search with CLIP
- Where can we use CLIP?
- Summary

Introduction: What is CLIP? And Why CLIP?

Traditional Model: ImageNet

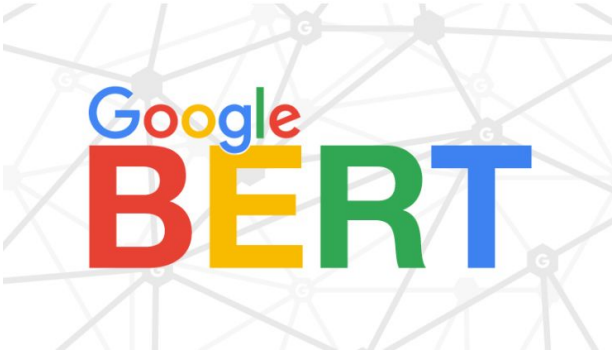
- AI Vision dataset
- ImageNet Classification with Deep Convolutional Neural Networks (2012)
- Labeled 14B dataset (high resolution, 22M Classes)
- Constructed for CNN

-> 1. Too expensive 2. Don’t know anything except the 22M classes



Revolution of BERT

- BERT from NLP using Transformer
- BERT uses self supervised learning



"If BERT could master language using just vast amounts of unlabeled text, could we master the visual world using vast amounts of unlabeled images?"

Problem!

- In NLP, we can make problems like ‘fill in the blank’. But we can’t make ‘blanks’ in images.

Natural Language Supervision

- BERT only looks at text
- CLIP looks at both images and text together, and learn the relationship between them

- CLIP teaches itself from hundreds of millions of (image, text) pairs on the internet

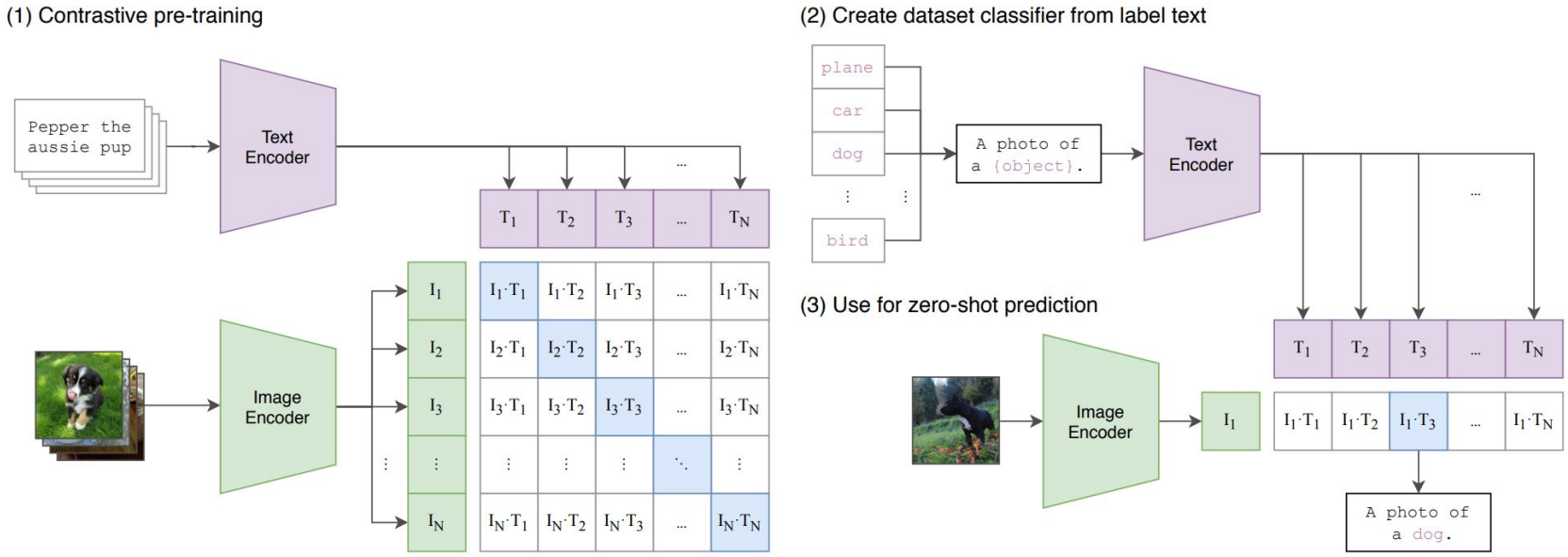
- Image:



- Text: Professor Mo volunteers to distribute briquettes

CLIP

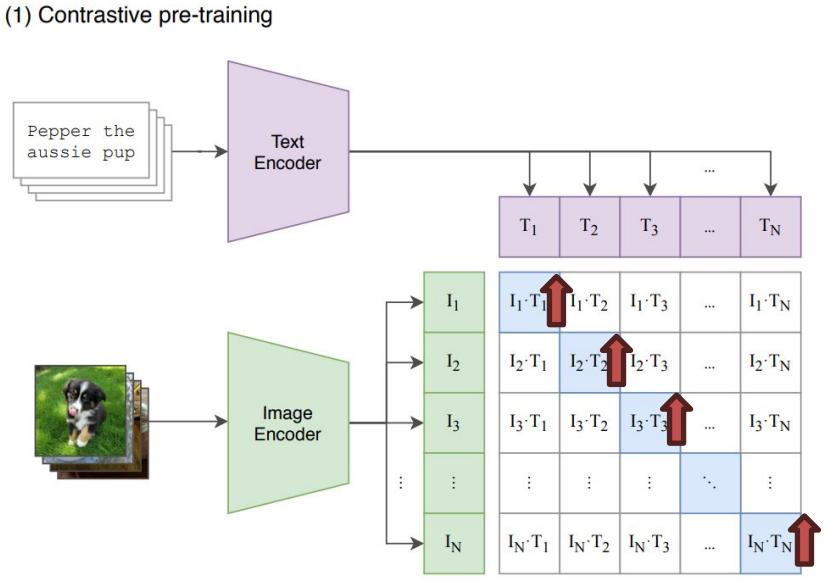
- Learning Transferable Visual Models From Natural Language Supervision (2023, OpenAI)
- Contrastive learning model (Not generative model)
- By training text encoders and image encoders using data from the Internet, they can guess what an unknown image is.
- Used 400B-sized dataset created by himself



Contrastive Learning

Contrastive Learning

- Method that model learns what is similar to each other and what is different.
- CLIP used **cosine similarity** and **cross entropy loss** for the loss function
- Not new! Idea from InfoNCE




```
# extract feature representations of each modality
I_f = image_encoder(I) #[n, d_i]
T_f = text_encoder(T)  #[n, d_t]

# joint multimodal embedding [n, d_e]
I_e = l2_normalize(np.dot(I_f, W_i), axis=1)
T_e = l2_normalize(np.dot(T_f, W_t), axis=1)

# scaled pairwise cosine similarities [n, n]
logits = np.dot(I_e, T_e.T) * np.exp(t)

# symmetric loss function
labels = np.arange(n)
loss_i = cross_entropy_loss(logits, labels, axis=0)
loss_t = cross_entropy_loss(logits, labels, axis=1)
loss   = (loss_i + loss_t)/2
```

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|}$$

Cosine similarity

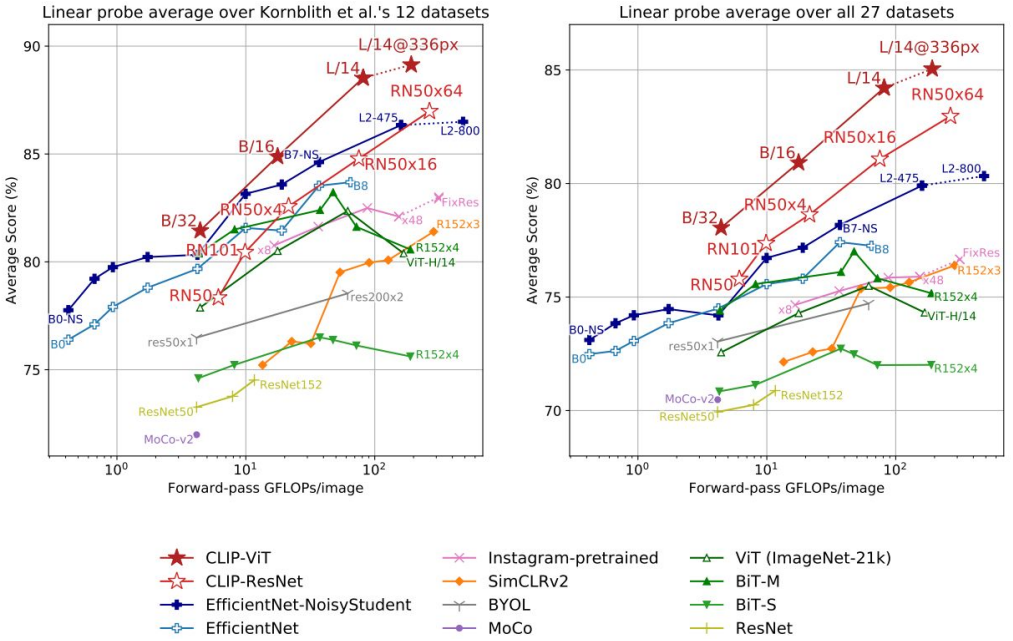
$$H = -\sum p(x) \log p(x)$$

Cross Entropy Loss

Model Architecture

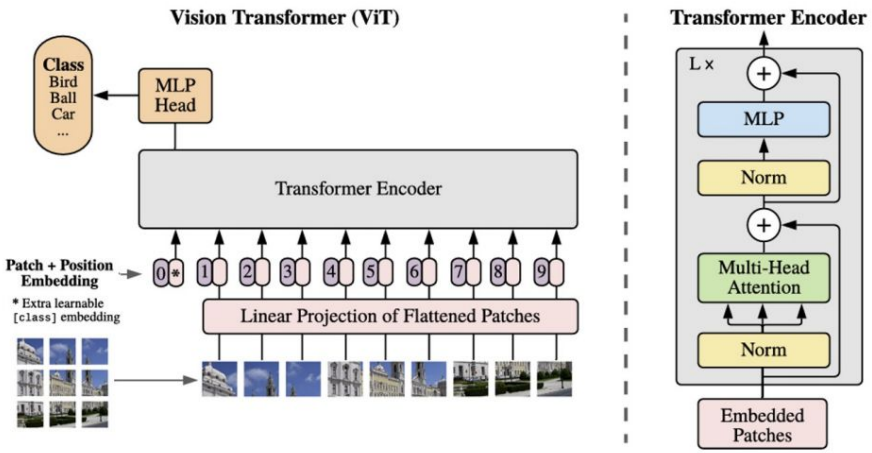
Image Encoder

- ResNet VS ViT -> ViT was 3 times more effective



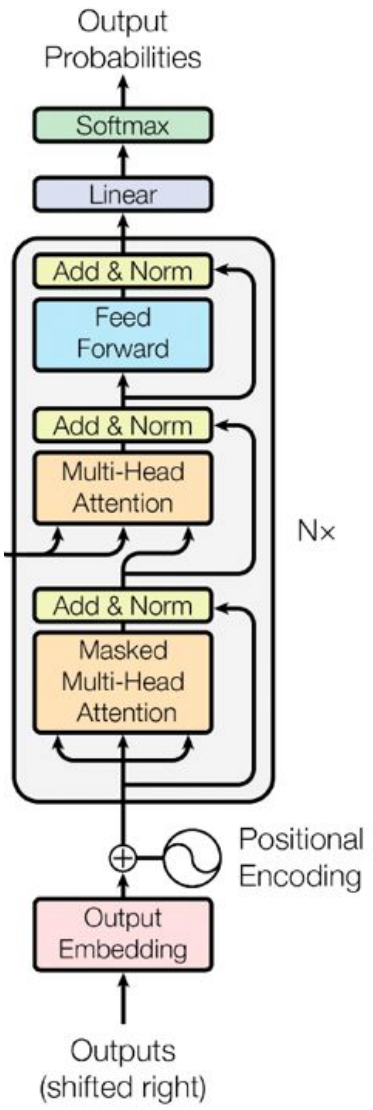
ViT Recall!

- The image is partitioned into small patches, processed via Patch and Position Embeddings, and then fed into the Transformer.



Text Encoder

- Transformer
- CLIP used the architecture introduced in the GPT2 paper, which is “decoder only”



Utilizing CLIP

Zero-Shot Prediction

1. Get all class you want to classify and use a prompt template like "A photo of a {object}." to make each class name a sentence
2. By inputting the completed sentences into the Text Encoder, vectors(T_1, T_2, T_3,..., T_N) are generated
3. An image to be classified is input into Image Encoder to extract a vector(I_1)
4. Calculate the cosine similarity of vectors and output it in the form of a probability distribution using the softmax function.

$$y_k = \frac{\exp(a_k)}{\sum_{i=1}^{n} \exp(a_i)}$$

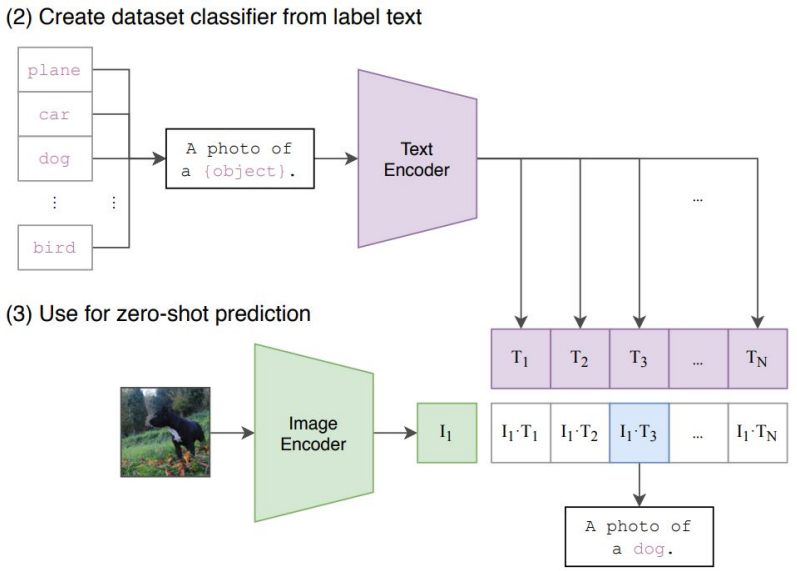
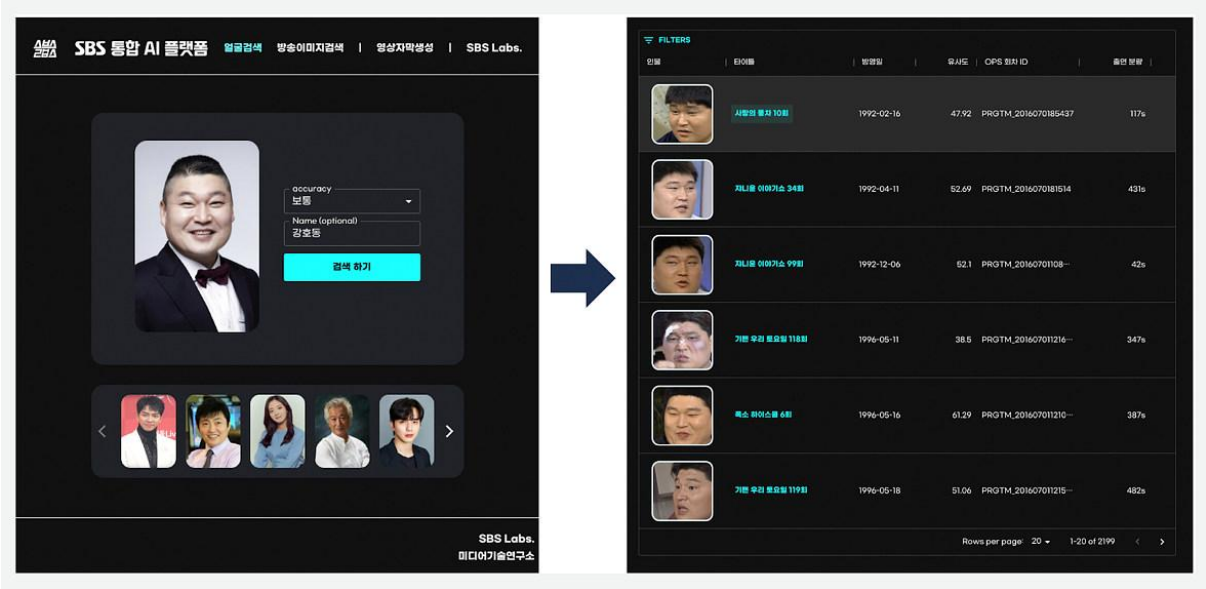


Image Database Indexing

1. Pass all images through the Image Encoder to extract individual image vectors and prepare these images in searchable database
2. The user enters text, such as “Professor Mo teaching students”, text passes through the Text Encoder and generate a text vector
3. Compare the text vector with the numerous image vectors and calculate a cosine similarity score
4. Sort the images by highest score and display them as search results



Limitations of CLIP

Limitations of CLIP

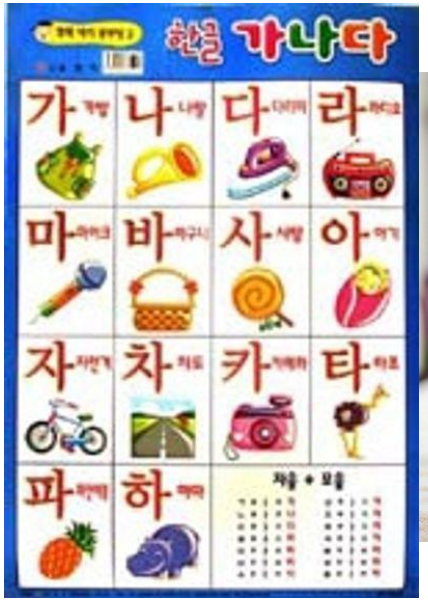
- Gap with SOTA(state of the art)
  - The researchers estimate that reaching SOTA level would require 1,000 times more computing resources.
- Shows poor performance on certain types of tasks
  - Fine-grained classification(flower's varieties, car models,…), abstract or systematic tasks(counting the number of objects in an image)



Limitations of CLIP

- Still vulnerable to data that is completely out of the training data distribution
  - For example, CLIP can't classify MNIST data (especially few shot)
- Still unable to solve the problem of low data efficiency in deep learning
  - It means DL has to learn many datas, but actually human don't need much data as DL

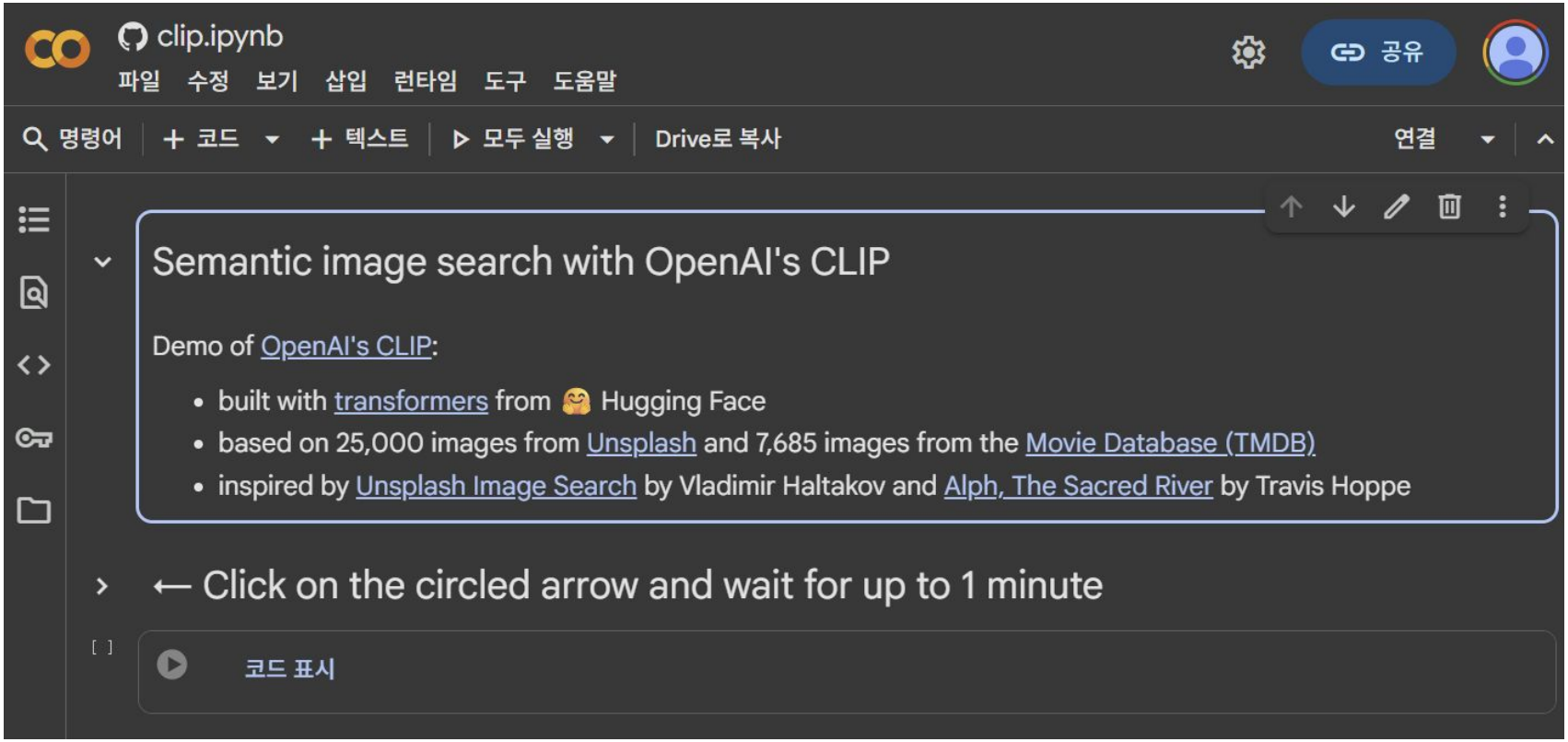






Practice: Semantic Image Search with CLIP

Let’s run CLIP code ~



Where can we use CLIP?

Guiding Image Generation

- CLIP can evaluate performance for models that convert text into images, such as DALL-E 2 and Stable Diffusion.

Clustering with Image Similarity

- CLIP can search or group images based on semantic similarity, without text. – Go start a business!

Content Moderation

- CLIP can identify harmful content using zero-shot classification capabilities. – Go start a business!

Examples of Industrial Use

- Example) FashionCLIP - A model specialized in the fashion industry that requires zero-shot classification, with a good understanding of specific styles, materials, patterns, and brands.

Summary

Summary of CLIP

- CLIP이란 이미지와 텍스트를 이해하고, 이 둘을 하나의 임베딩 공간에 배치하는 모델이다.
- 인터넷에 있는 4억 개의 (이미지, 텍스트) 쌍(WIT)을 가져와서 학습한다.
- Image Encoder는 ViT, Text Encoder는 Transformer Decoder를 사용한다.
- 실제 짝이 맞는 (이미지, 텍스트) 벡터는 서로 가까워지도록, 짝이 맞지 않는 벡터는 서로 멀어지도록 학습한다.
- Zero Shot 성능이 강력하다는 장점이 있다.
- Few Shot 성능이 약하고, 세분화/추상화된 개념에 약하다는 단점이 있다.

Any Questions?