CIS 6930 Topics in Computing for Data Science
Week 9: Pre-trained Language Models (2)

10/28/2021

Yoshihiko (Yoshi) Suhara

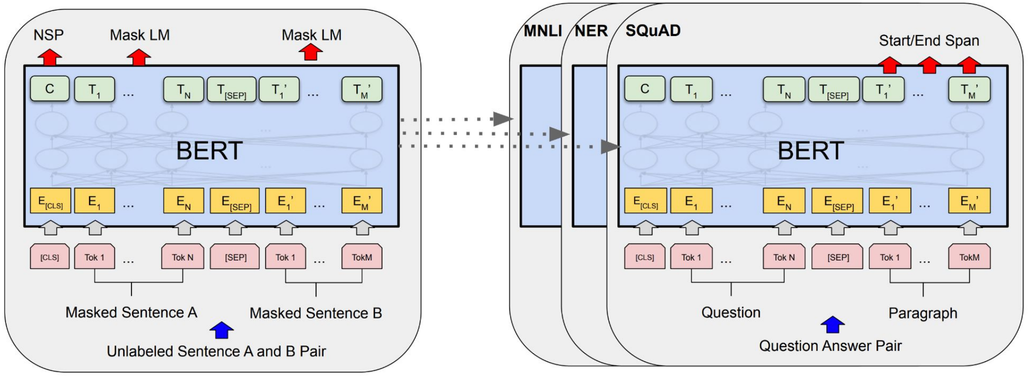
2pm-3:20pm

Week 9: Pre-trained Language Models

- ~~**Week 8: Transformers**~~
- **Week 9: Pre-trained Language Models**
- Week 10: More Machine Learning Techniques
- Week 11: More Deep Learning Techniques for NLP (Text generation, Text summarization, Information Extraction etc.)
- Week 12: Advanced Techniques and Challenges
- Week 13: Final project presentations

Today’s Goal

- Pre-trained Language Models
- BERT
  - Basic Architecture
  - Pre-training
  - Fine-tuning
- Hands-on session: Fine-tuning BERT for text classification



Pre-training Fine-Tuning

Recap: Pre-training & Fine-tuning Framework + BERT Basic Architecture

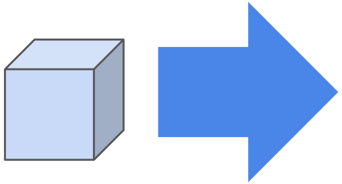
Pre-training & Fine-tuning: Intuition

Pre-training
(Basic training)

Pre-training Task 1

Pre-training Task 2

Fine-tuning
(Skill training)

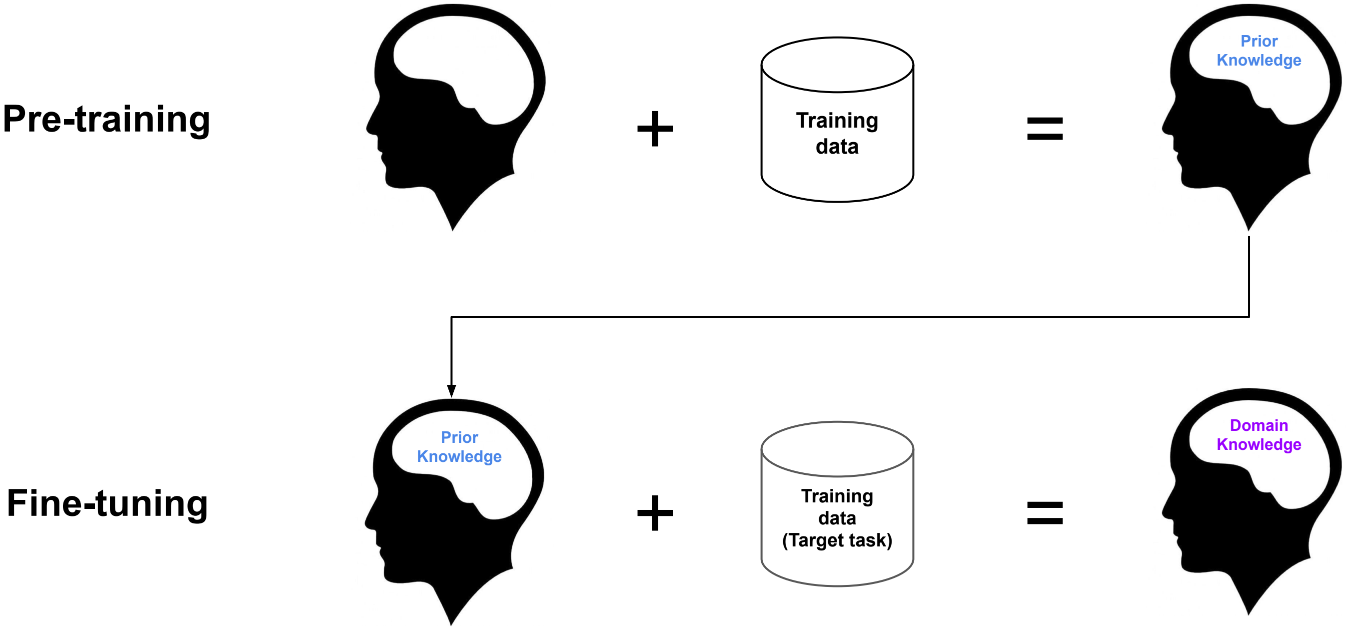


(e.g., BERT)

Downstream tasks
(Sports)

Pre-training & Fine-tuning Framework

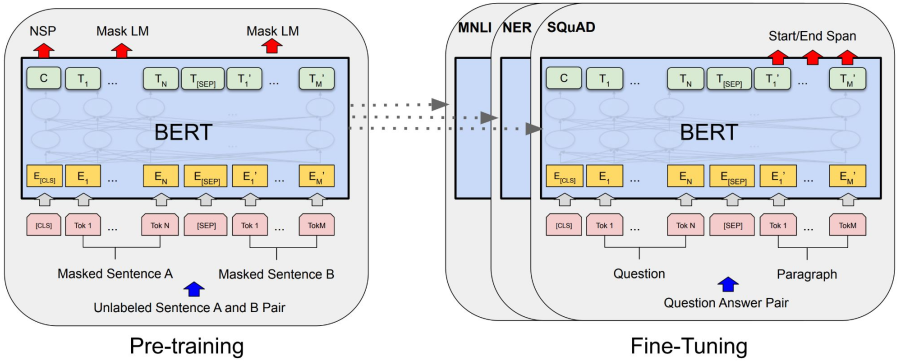
-



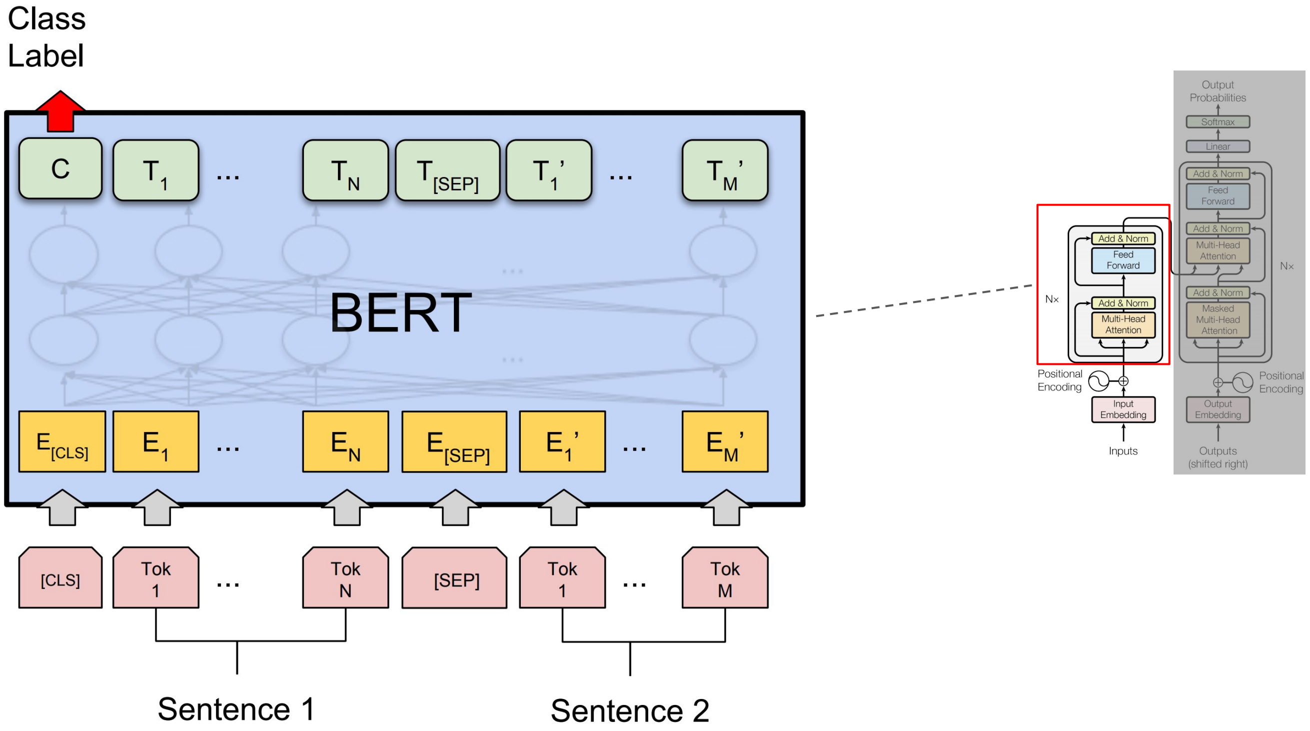
What is BERT? Why BERT?

- **BERT: Bidirectional Encoder Representations from Transformers**
  - The most popular pre-trained language model in NLP

- A pre-trained **Transformer Encoder** model, which can be fine-tuned for a variety of NLP tasks

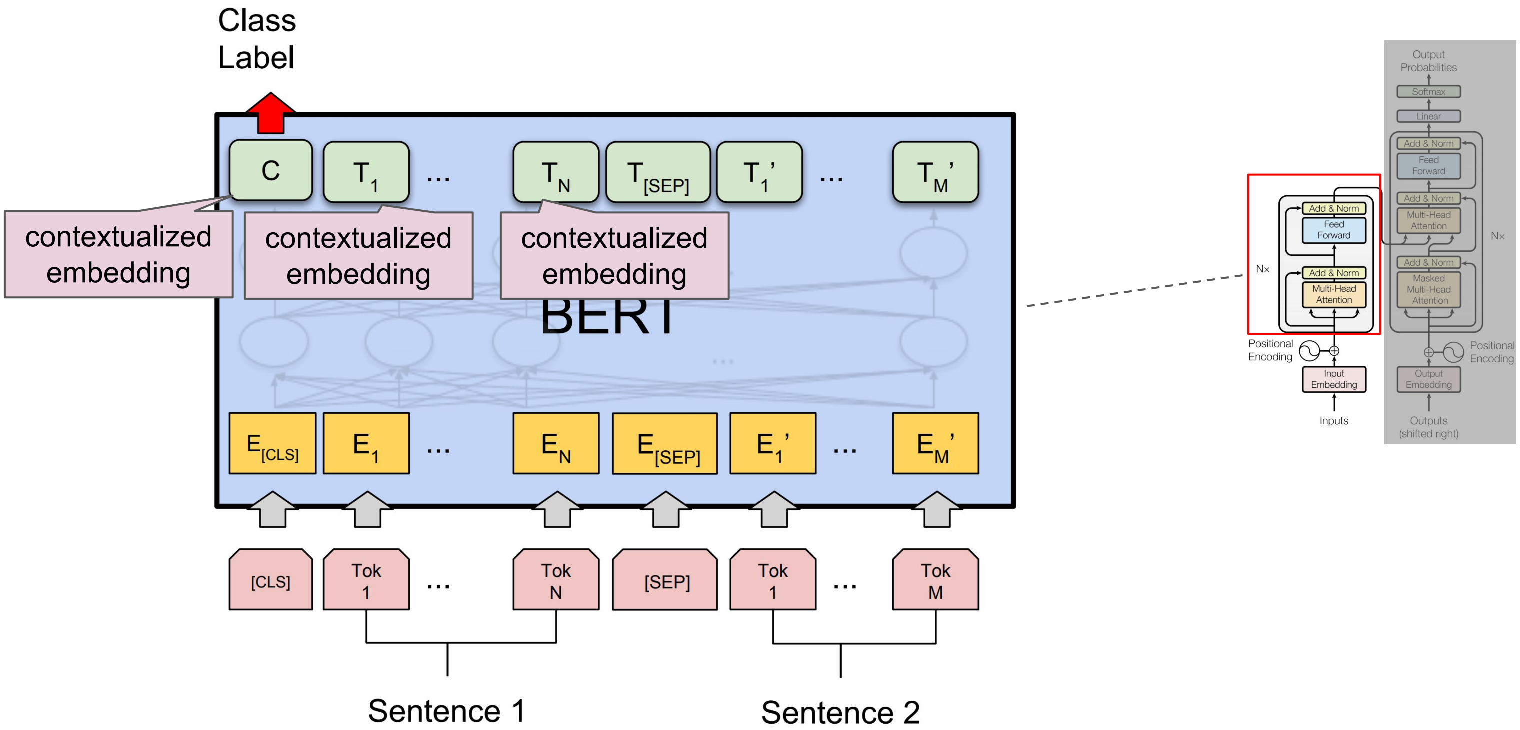


BERT = (pre-trained) Transformer Encoder
(= Stacked Transformer Encoder Blocks)



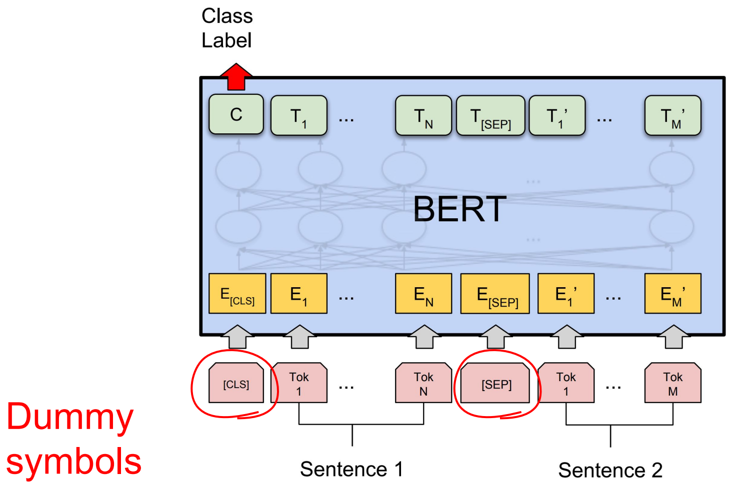
Key Point: Self-Attention

- Any input tokens attend **all input tokens** (i.e., “full” attention)



Dummy Symbols: [CLS] & [SEP]

- [CLS]: A class label
- [SEP]: A separator between two sequences



(*) In the vocabulary set, each dummy symbol has its own token ID

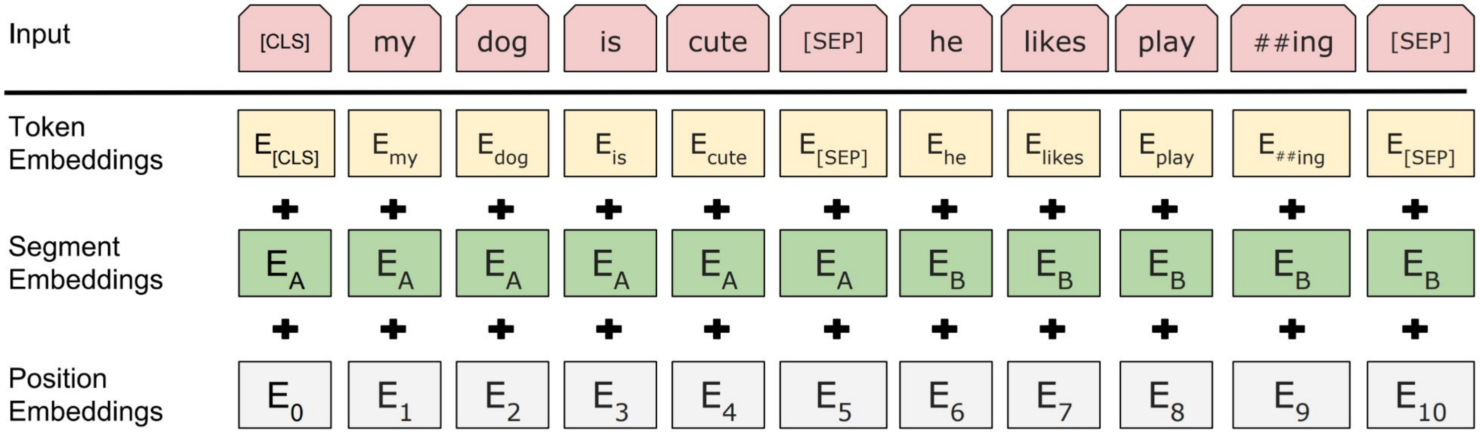
BERT Input Embeddings

- Summation of **3 different embeddings**

$\mathbb{R}^{|V|} \rightarrow \mathbb{R}^d$

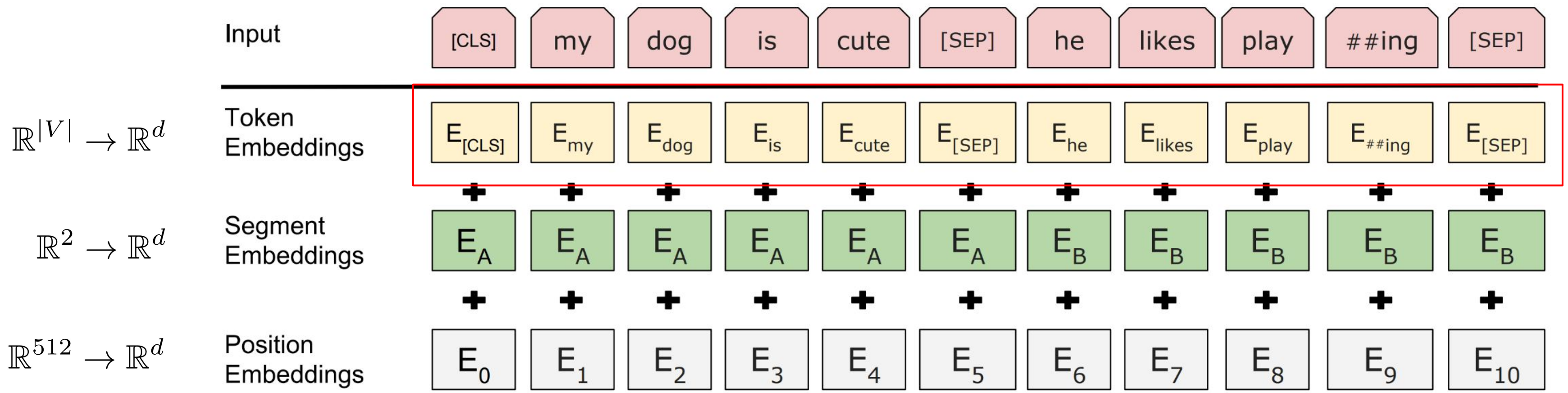
$\mathbb{R}^2 \rightarrow \mathbb{R}^d$

$\mathbb{R}^{512} \rightarrow \mathbb{R}^d$



BERT Input Embeddings: (1) Token Embeddings

- Token IDs → Dense vectors



```python
from transformers import AutoTokenizer
import torch.nn as nn

tokenizer = AutoTokenizer.from_pretrained("bert-base-uncased")
```

```python
embedding_layer = nn.Embedding(num_embeddings=len(tokenizer.vocab),
                               embedding_dim=512)
embedding_layer
```

```
Embedding(30522, 512)
```

[4]

```python
token_ids = tokenizer.encode("Hello, world!", return_tensors="pt")
token_ids
```

```
tensor([[ 101, 7592, 1010, 2088,  999,  102]])
```

Theoretically, one-hot vectors

[5]

```python
embedding_layer(token_ids)
```

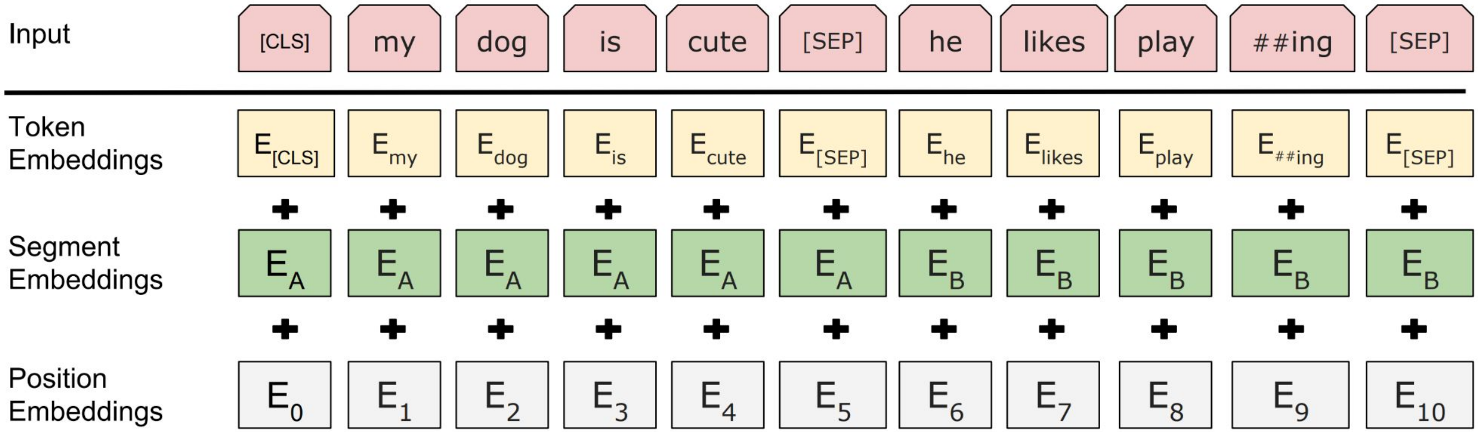
```
tensor([[[-0.0346, -0.4729,  0.1617,  ..., -0.2492, -2.0112,  0.8898],
         [ 1.5803,  0.6044, -1.5162,  ...,  1.0122,  0.7147,  0.1837],
         [ 0.3189,  0.3362, -0.2465,  ..., -0.6314,  0.3010,  1.9631],
         [ 0.0568,  1.3271,  1.1248,  ..., -0.1465,  0.0732, -0.7537],
         [-0.2657,  0.4876, -0.8247,  ..., -0.5379, -1.6721,  1.8140],
         [-1.0884, -0.4764,  0.3577,  ...,  1.5978, -0.4875, -0.1534]]],
       grad_fn=<EmbeddingBackward>)
```

```python
embedding_layer(token_ids).shape
```

```
torch.Size([1, 6, 512])
```

Recap: BERT Input Embeddings

- Summation of **3 different embeddings**



$\mathbb{R}^{|V|} \rightarrow \mathbb{R}^d$

$\mathbb{R}^2 \rightarrow \mathbb{R}^d$

$\mathbb{R}^{512} \rightarrow \mathbb{R}^d$

Recap: BERT Input Embeddings

- Summation of **3 different embeddings**

	Input
$\mathbb{R}^{\lvert V \rvert} \to \mathbb{R}^d$	Token Embeddings
$\mathbb{R}^2 \to \mathbb{R}^d$	Segment Embeddings
$\mathbb{R}^{512} \to \mathbb{R}^d$	Position Embeddings

```python
token_embedding_layer = nn.Embedding(num_embeddings=len(tokenizer.vocab),
                                     embedding_dim=512)
segment_embedding_layer = nn.Embedding(num_embeddings=2,
                                       embedding_dim=512)
position_embedding_layer = nn.Embedding(num_embeddings=512,
                                        embedding_dim=512)

token_emb = token_embedding_layer(token_ids)
segment_emb = segment_embedding_layer(torch.ones_like(token_ids))
position_emb = position_embedding_layer(torch.arange(token_ids.shape[1]).unsqueeze(0))

token_emb + segment_emb + position_emb
```

```
tensor([[[ 4.5849,  1.8874, -2.1406,  ...,  3.8009,  4.4695,  0.6101],
         [-0.7155,  3.2652,  0.9677,  ..., -0.2787,  2.3750, -1.5067],
         [ 0.9926, -1.4188, -2.0189,  ...,  1.3811,  0.0056,  0.2420],
         [-1.4637,  0.6328,  0.6636,  ...,  3.0259,  1.0696,  0.7773],
         [ 3.0965,  0.0137, -0.2175,  ..., -0.4382,  1.5621,  0.1593],
         [-0.1593,  0.0458, -1.4924,  ...,  0.2568, -1.0519, -1.3819]]],
       grad_fn=<AddBackward0>)
```

BERT Tokenizer: WordPiece

- A **subword tokenizer** that initializes the vocabulary with **individual characters** and then **iteratively adds the most frequent combinations** of symbols in the vocabulary
  - No out-of-vocabulary issue!
  - Basically the same as **Byte Pair Encoding (BPE)**

- **Word**: Jet makers feud over seat width with big orders at stake
- **wordpieces**: _J et _makers _fe ud _over _seat _width _with _big _orders _at _stake

(*) A whitespace is now considered one character

Tokenizers for BERT (Pre-trained LMs): Takeaway

- A subword tokenizer (i.e., No out-of-vocabulary issue)
- [important!] Need to be “trained” on text data

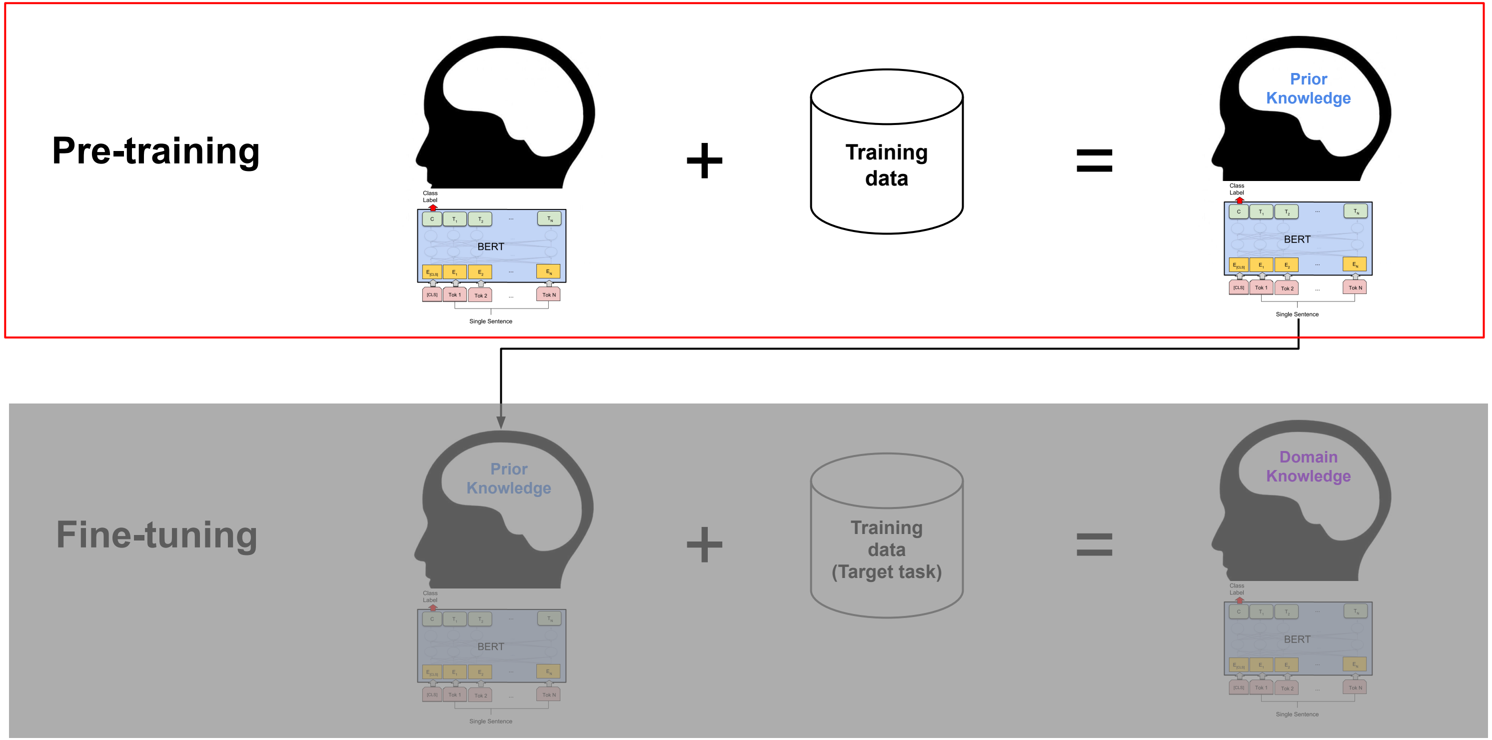
→ Pre-trained language model & **pre-trained** tokenizer <u>model</u> are coupled

```
>>> from transformers import AutoTokenizer, AutoModel

>>> tokenizer = AutoTokenizer.from_pretrained("bert-base-uncased")
>>> model = AutoModel.from_pretrained("bert-base-uncased")

>>> inputs = tokenizer("Hello world!", return_tensors="pt")
>>> outputs = model(**inputs)
```

Questions?

Pre-training BERT

Recap: Pre-training & Fine-tuning



Pre-training = (A) Pre-training Method(s) + (B) Corpus

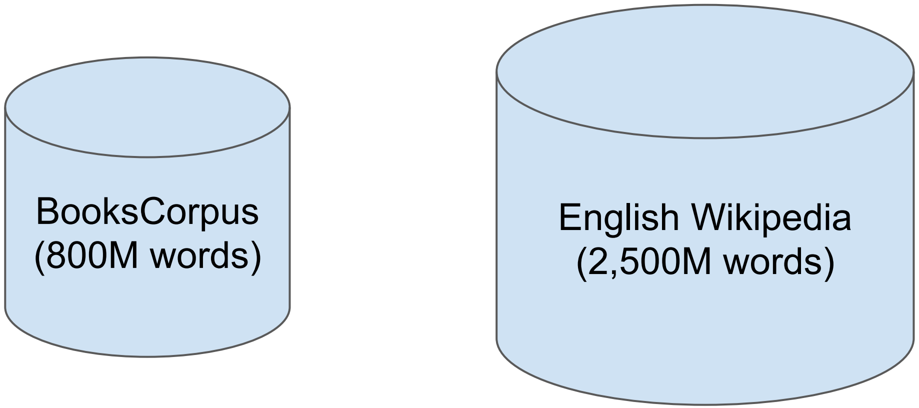
- (A) **How to learn**: Pre-training method(s)
- (B) **What to learn from**: Textual corpora

Pre-training

BERT Pre-training

- Pre-training Methods
  - 1) Masked Language Model
  - 2) Next Sentence Prediction

- Pre-training corpus



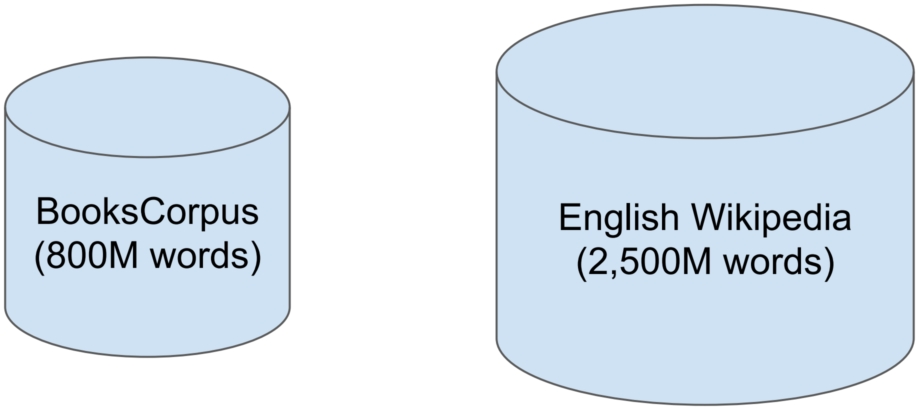
BERT Pre-training

- Pre-training Methods
  - 1) Masked Language Model
  - 2) Next Sentence Prediction

Why those two?

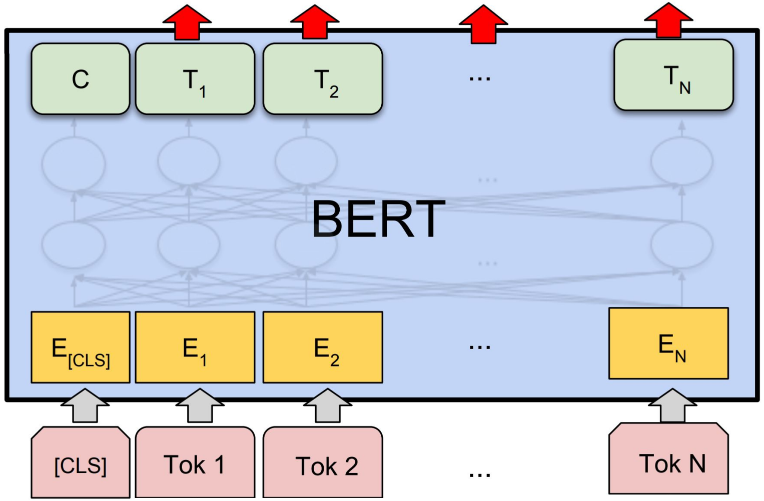
- Pre-training corpus

Why those two?



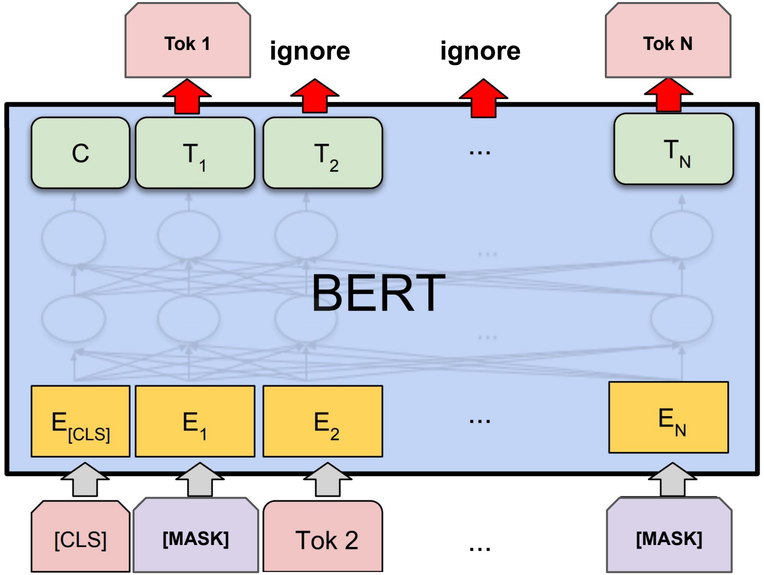
Masked Language Model (MLM)

- Step 1: Replace **randomly selected tokens** with **[MASK] tokens**
- Step 2: Train the model to **reconstruct the original token**



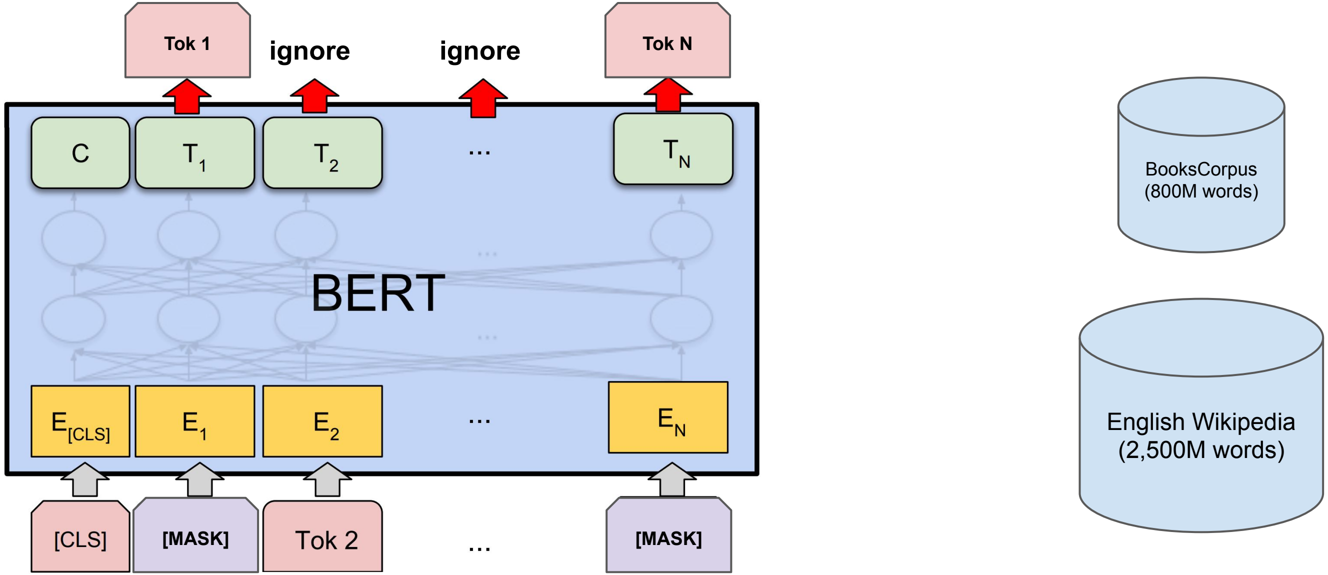
Masked Language Model (MLM)

- Step 1: Replace **randomly selected tokens** with **[MASK] tokens**
- Step 2: Train the model to **reconstruct the original token**



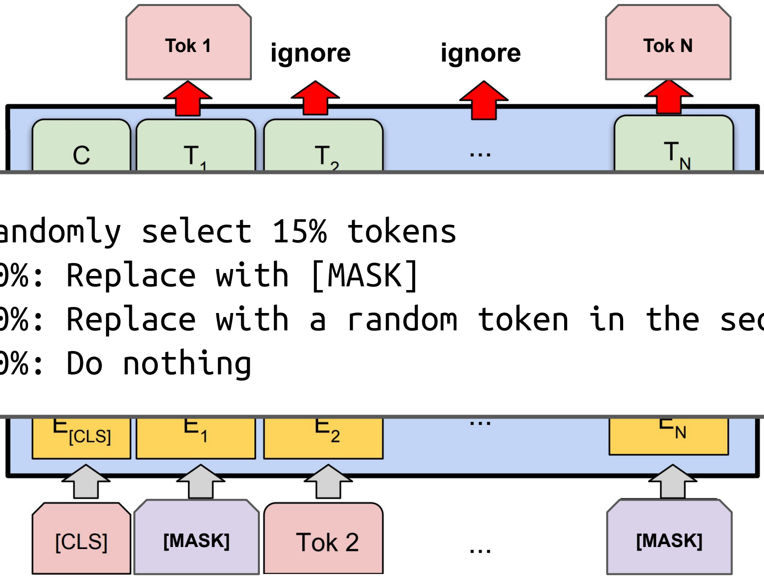
Masked Language Model (MLM)

- Step 1: Replace **randomly selected tokens** with **[MASK] tokens**
- Step 2: Train the model to **reconstruct the original token**

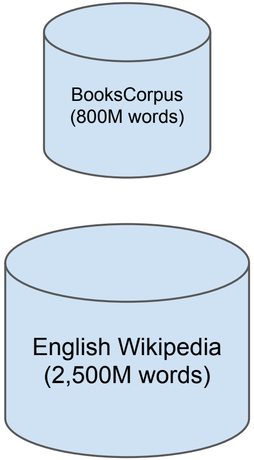


Masked Language Model (MLM)

- Step 1: Replace **randomly selected tokens** with **[MASK] tokens**
- Step 2: Train the model to **reconstruct the original token**



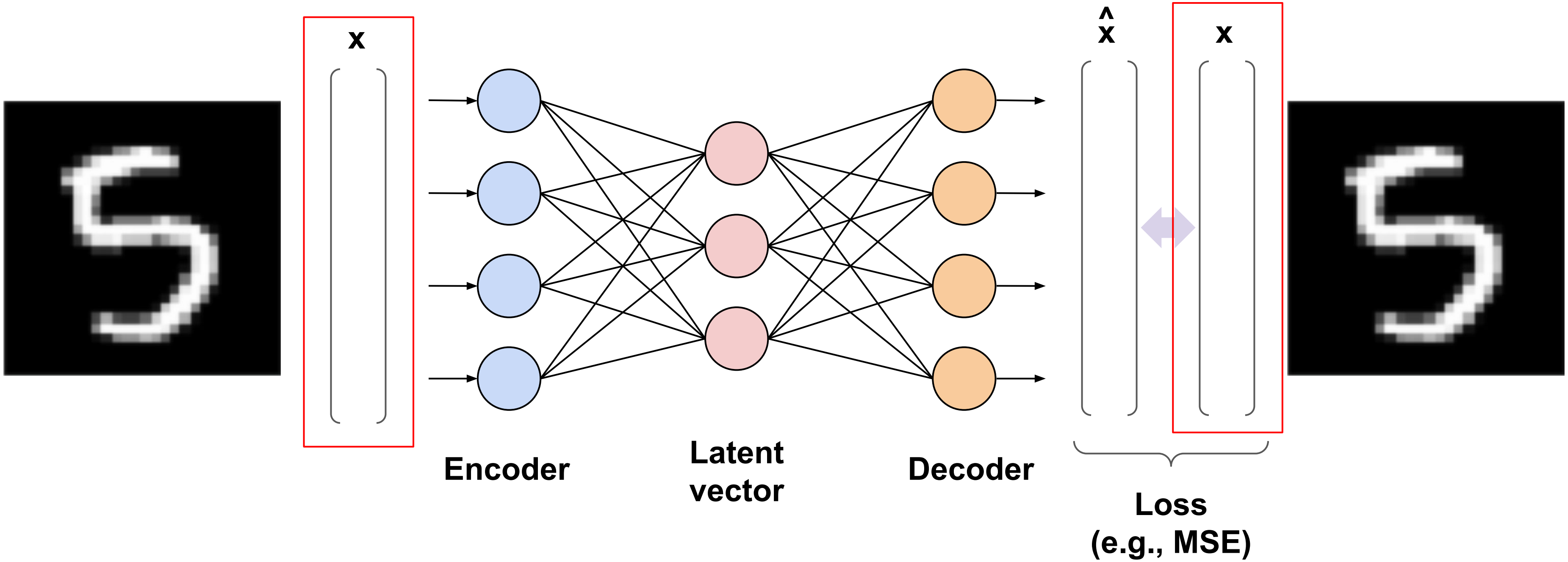

```
Step 1-1) Randomly select 15% tokens
Step 1-2) 80%: Replace with [MASK]
          10%: Replace with a random token in the sequence
          10%: Do nothing
```



Recap

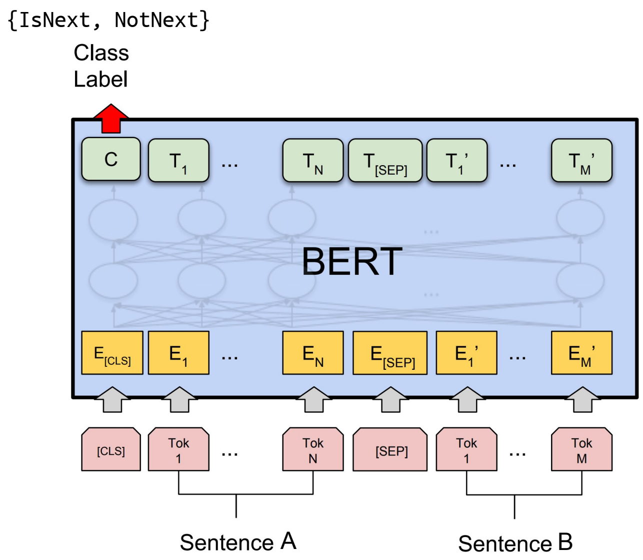
Autoencoders: Training (self-reconstruction)

- Simply use input data as the target data



Next Sentence Prediction (NSP)

- Step 1) Sample sentences A and B such that
    - 50%: B is **actual next sentence** that follows A
    - 50%: random sentence from the corpus
- Step 2) Train the model to predict the correct label



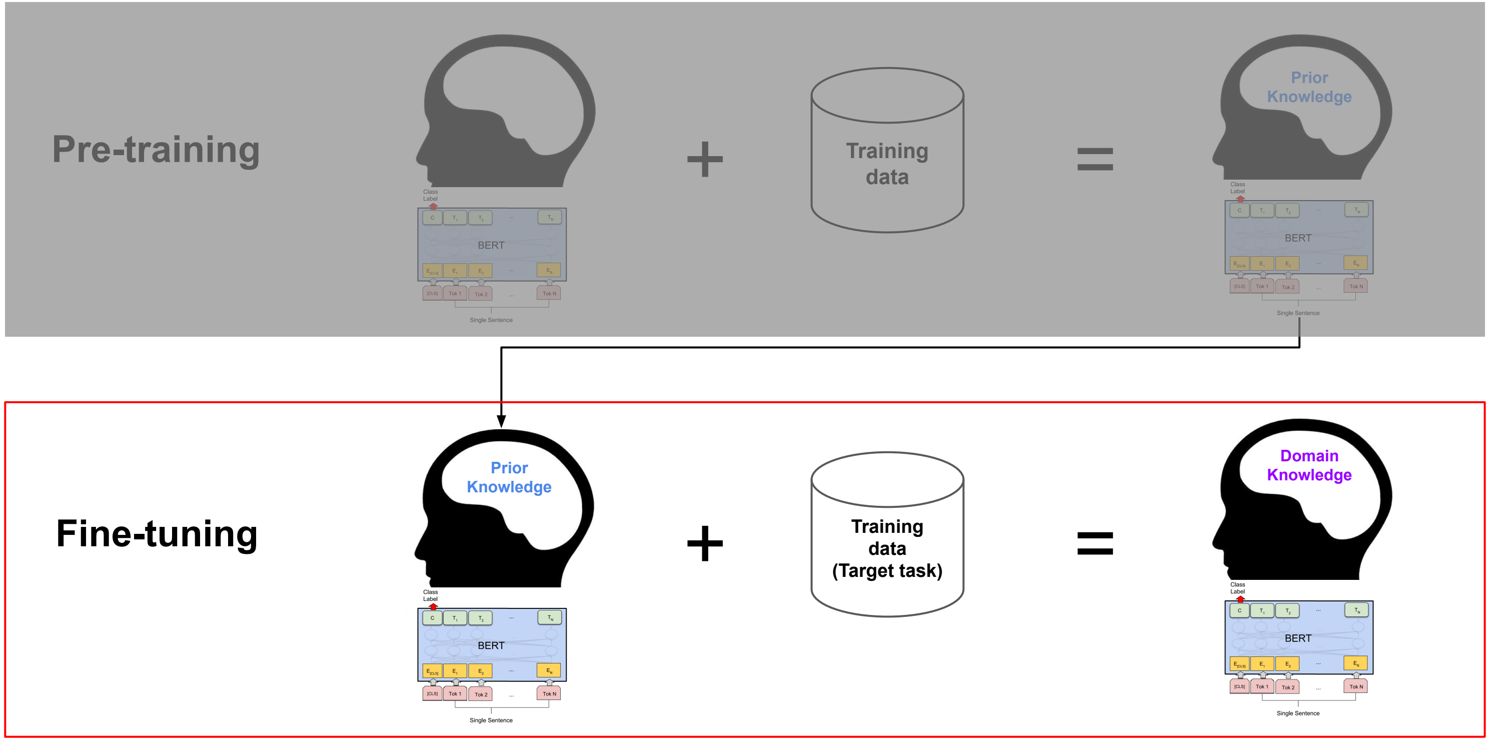
Self-supervised Learning for Pre-training

- Both Masked Language Model and Next Sentence Prediction **do NOT require** any additional labels
- This type of learning process is often called **self-supervised learning**

- BERT variants essentially
  - (1) use different (self-supervised) pre-training methods **and/or**
  - (2) use different pre-training corpora **and/or**
  - (3) use different model architectures
  - … and claim that they are better than BERT 😅

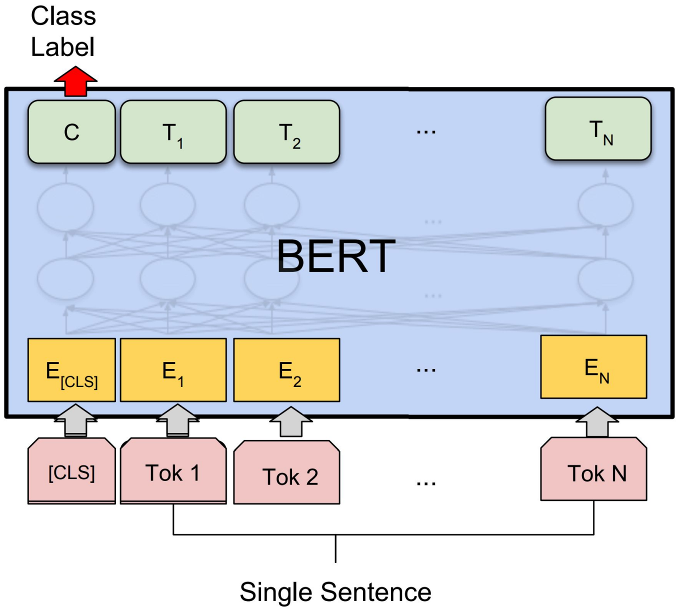
Fine-tuning BERT

Recap: Pre-training & Fine-tuning



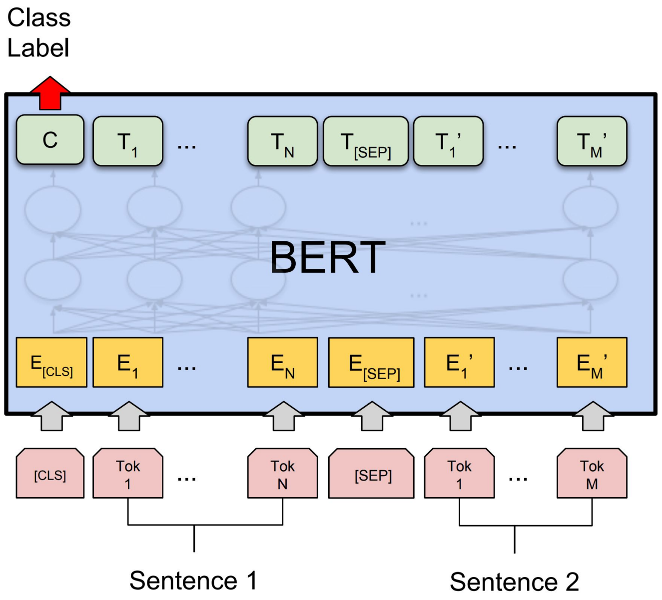
Single-sentence Classification

- Text classification tasks



Sentence-pair Classification

- Textual entailment recognition and duplicate question detection tasks



Recognizing Textual Entailment (RTE) aka Natural Language Inference

- Give two texts, judge if one text is **entailed** by the other

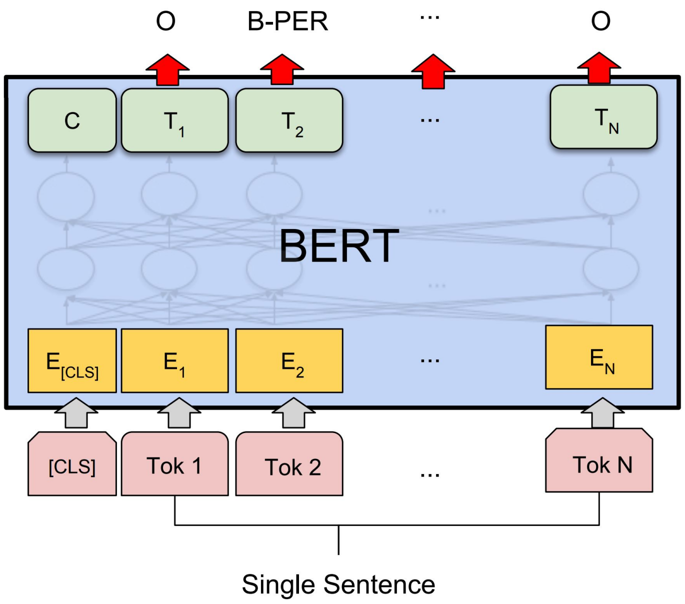
Gold-standard label

Text (premise)	Judgments	Hypothesis
A man inspects the uniform of a figure in some East Asian country.	contradiction C C C C C	The man is sleeping
An older and younger man smiling.	neutral N N E N N	Two men are smiling and laughing at the cats playing on the floor.
A black race car starts up in front of a crowd of people.	contradiction C C C C C	A man is driving down a lonely road.
A soccer game with multiple males playing.	entailment E E E E E	Some men are playing a sport.
A smiling costumed woman is holding an umbrella.	neutral N N E C N	A happy woman in a fairy costume holds an umbrella.

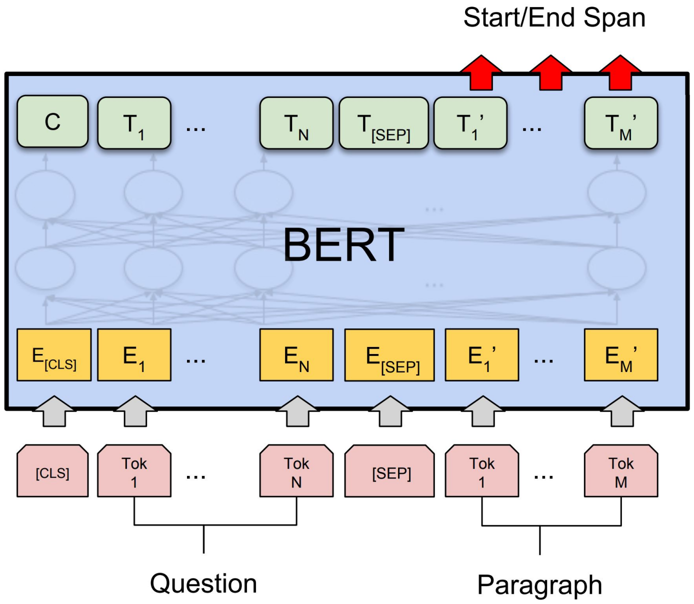
The Stanford NLI dataset https://nlp.stanford.edu/projects/snli/

Token Classification

- Sequential tagging problems (e.g., Named Entity Recognition)



Question Answering



(Extractive) Question Answering

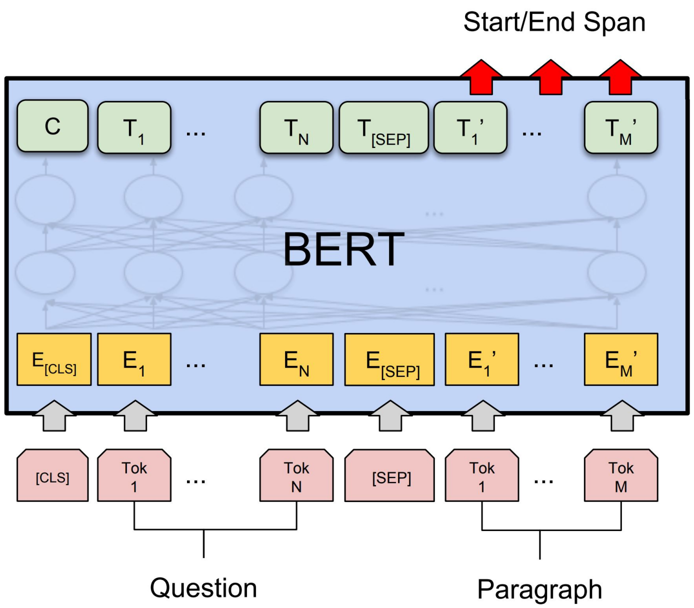
- Example: SQuAD v1.1
  - Input: Question & Passage
  - Output: Answer span

Question: Which team won Super Bowl 50?

Passage

Super Bowl 50 was an American football game to determine the champion of the National Football League (NFL) for the 2015 season. The American Football Conference (AFC) champion Denver Broncos defeated the National Football Conference (NFC) champion Carolina Panthers 24–10 to earn their third Super Bowl title. The game was played on February 7, 2016, at Levi's Stadium in the San Francisco Bay Area at Santa Clara, California.

Question Answering

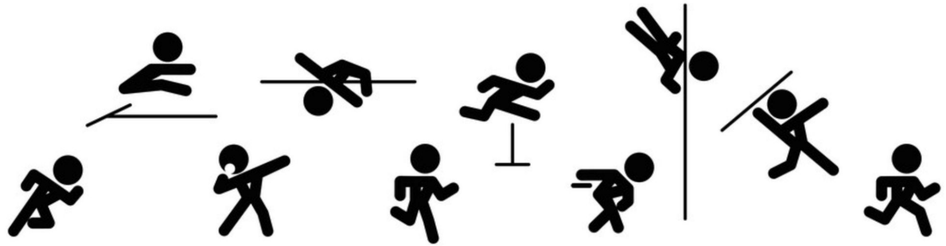


the B embedding. We only introduce a start vector $S \in \mathbb{R}^H$ and an end vector $E \in \mathbb{R}^H$ during fine-tuning. The probability of word i being the start of the answer span is computed as a dot product between T_i and S followed by a softmax over all of the words in the paragraph: $P_i = \frac{e^{S \cdot T_i}}{\sum_j e^{S \cdot T_j}}$. The analogous formula is used for the end of the answer span. The score of a candidate span from position i to position j is defined as $S \cdot T_i + E \cdot T_j$, and the maximum scoring span where $j \geq i$ is used as a prediction. The training objective is the

(* The task could be solved as a sequential tagging problem, which may be overkill)

BERT Established New SoTA Performance on the GLUE Benchmark

System	MNLI-(m/mm)	QQP	QNLI	SST-2	CoLA	STS-B	MRPC	RTE	**Average**
	392k	363k	108k	67k	8.5k	5.7k	3.5k	2.5k	-
Pre-OpenAI SOTA	80.6/80.1	66.1	82.3	93.2	35.0	81.0	86.0	61.7	74.0
BiLSTM+ELMo+Attn	76.4/76.1	64.8	79.8	90.4	36.0	73.3	84.9	56.8	71.0
OpenAI GPT	82.1/81.4	70.3	87.4	91.3	45.4	80.0	82.3	56.0	75.1
$\text{BERT}_{\text{BASE}}$	84.6/83.4	71.2	90.5	93.5	52.1	85.8	88.9	66.4	79.6
$\text{BERT}_{\text{LARGE}}$	**86.7/85.9**	**72.1**	**92.7**	**94.9**	**60.5**	**86.5**	**89.3**	**70.1**	**82.1**



GLUE is a “decathlon-style” benchmark, which consists of multiple NLP tasks

GLUE Leaderboard

https://gluebenchmark.com/

GLUE SuperGLUE Paper </> Code Tasks Leaderboard FAQ Diagnostics Submit Login

Rank	Name	Model	URL	Score	CoLA	SST-2	MRPC	STS-B	QQP	MNLI-m	MNLI-mm	QNLI	RTE	WNLI	AX
1	ERNIE Team - Baidu	ERNIE		91.1	75.5	97.8	93.9/91.8	93.0/92.6	75.2/90.9	92.3	91.7	97.3	92.6	95.9	51.7
2	AliceMind & DIRL	StructBERT + CLEVER		91.0	75.3	97.7	93.9/91.9	93.5/93.1	75.6/90.8	91.7	91.5	97.4	92.5	95.2	49.1
3	DeBERTa Team - Microsoft	DeBERTa / TuringNLRv4		90.8	71.5	97.5	94.0/92.0	92.9/92.6	76.2/90.8	91.9	91.6	99.2	93.2	94.5	53.2
4	liangzhu ge	DeBERTa + CLEVER		90.8	73.4	97.5	92.8/90.4	93.2/92.9	76.3/90.8	92.1	91.7	96.5	92.8	96.6	35.2
5	HFL iFLYTEK	MacALBERT + DKM		90.7	74.8	97.0	94.5/92.6	92.8/92.6	74.7/90.6	91.3	91.1	97.8	92.0	94.5	52.6
6	PING-AN Omni-Sinitic	ALBERT + DAAF + NAS		90.6	73.5	97.2	94.0/92.0	93.0/92.4	76.1/91.0	91.6	91.3	97.5	91.7	94.5	51.2
7	T5 Team - Google	T5		90.3	71.6	97.5	92.8/90.4	93.1/92.8	75.1/90.6	92.2	91.9	96.9	92.8	94.5	53.1
8	Microsoft D365 AI & MSR AI & GATECH	MT-DNN-SMART		89.9	69.5	97.5	93.7/91.6	92.9/92.5	73.9/90.2	91.0	90.8	99.2	89.7	94.5	50.2
9	Huawei Noah's Ark Lab	NEZHA-Large		89.8	71.7	97.3	93.3/91.0	92.4/91.9	75.2/90.7	91.5	91.3	96.2	90.3	94.5	47.9
10	Zihang Dai	Funnel-Transformer (Ensemble B10-10-10H1024)		89.7	70.5	97.5	93.4/91.2	92.6/92.3	75.4/90.7	91.4	91.1	95.8	90.0	94.5	51.6
11	ELECTRA Team	ELECTRA-Large + Standard Tricks		89.4	71.7	97.1	93.1/90.7	92.9/92.5	75.6/90.8	91.3	90.8	95.8	89.8	91.8	50.7
12	Microsoft D365 AI & UMD	FreeLB-RoBERTa (ensemble)		88.4	68.0	96.8	93.1/90.8	92.3/92.1	74.8/90.3	91.1	90.7	95.6	88.7	89.0	50.1
13	Junjie Yang	HIRE-RoBERTa		88.3	68.6	97.1	93.0/90.7	92.4/92.0	74.3/90.2	90.7	90.4	95.5	87.9	89.0	49.3
14	Facebook AI	RoBERTa		88.1	67.8	96.7	92.3/89.8	92.2/91.9	74.3/90.2	90.8	90.2	95.4	88.2	89.0	48.7
15	Microsoft D365 AI & MSR AI	MT-DNN-ensemble		87.6	68.4	96.5	92.7/90.3	91.1/90.7	73.7/89.9	87.9	87.4	96.0	86.3	89.0	42.8

Hugging Face’s Transformers Library
https://github.com/huggingface/transformers

- You can use a variety of pre-trained language models just with a few lines of Python code

```
>>> from transformers import AutoTokenizer, AutoModel

>>> tokenizer = AutoTokenizer.from_pretrained("bert-base-uncased")
>>> model = AutoModel.from_pretrained("bert-base-uncased")

>>> inputs = tokenizer("Hello world!", return_tensors="pt")
>>> outputs = model(**inputs)
```

Hands-on: BERT Fine-tuning Example (20 min)

- Google Colab

Assignment 4 (due Friday 11/5)

- https://colab.research.google.com/github/suhara/cis6930-fall2021/blob/main/notebooks/cis6930_week9b_pretrained_lm_assignment.ipynb

Summary

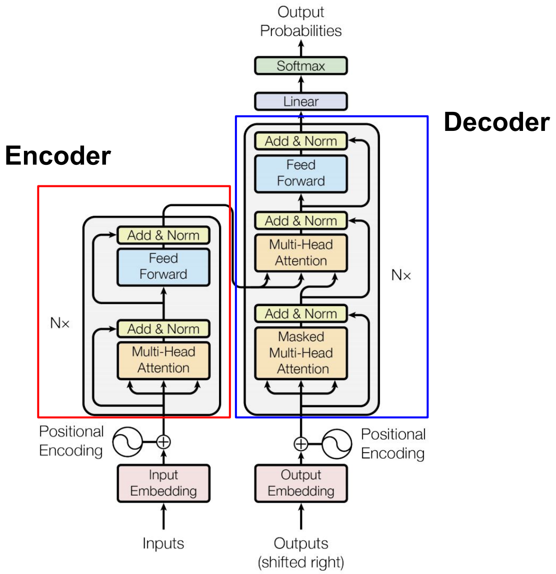
- Pre-training & Fine-tuning framework
- Pre-trained language models
    - =~ Pre-trained Transformer models (in NLP)
- BERT
    - A pre-trained Transformer Encoder-only model
    - Dummy symbols: [CLS], [SEP]
    - Input embeddings
    - **4 Fine-tuning patterns**
        - Single-sentence classification
        - Sequence-pair classification
        - Token classification (i.e., sequential labeling)
        - Question Answering
    - Pre-training methods
        - Masked Language Model
        - Next Sentence Prediction

Next session

Pre-trained Language Models: Categorization

- Most pre-trained LMs can be categorized into the following 3 patterns
  - 1) Transformer **Encoder-only** Models (e.g., BERT, RoBERTa, ALBERT)
  - 2) Transformer **Decoder-only** Models (e.g., GPT-2/3)
  - 3) Transformer **Encoder-Decoder** Models (e.g., BART, T5)

Next →



Week 10: More Transformers!

- ~~**Week 8: Transformers**~~
- Week 9: Pre-trained Language Models
- **Week 10: Pre-trained Language Models**
  - **More pre-trained Language Models (Decoder-only models, Encoder-decoder models)**
  - **Transformers for Computer Vision**
- Week 11: More Deep Learning Techniques for NLP (Text generation, Text summarization, Information Extraction etc.)
- Week 12: Advanced Techniques and Challenges
- Week 13: Final project presentations