BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding

CS221.N11.KHTN

Group 6:

1. Tran Huu Khoa
2. Nguyen Duy Dat
3. Huynh Hoang Vu
4. Le The Viet

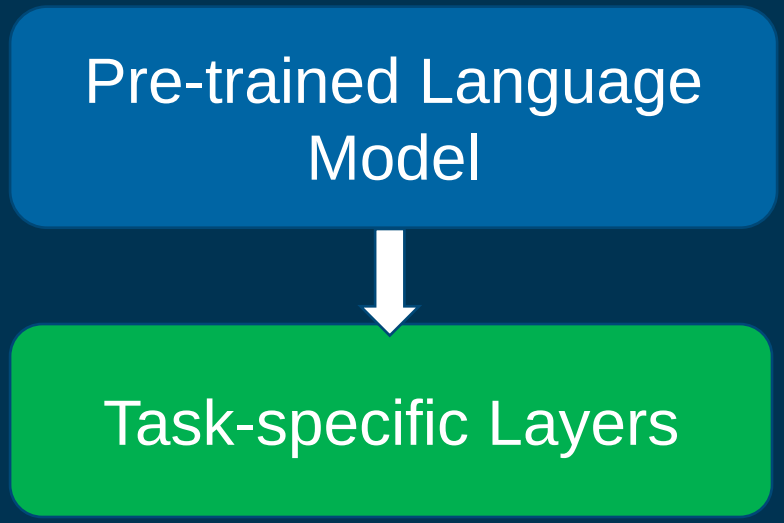
Table of Contents

Applying pre-trained models

Applying pre-trained models

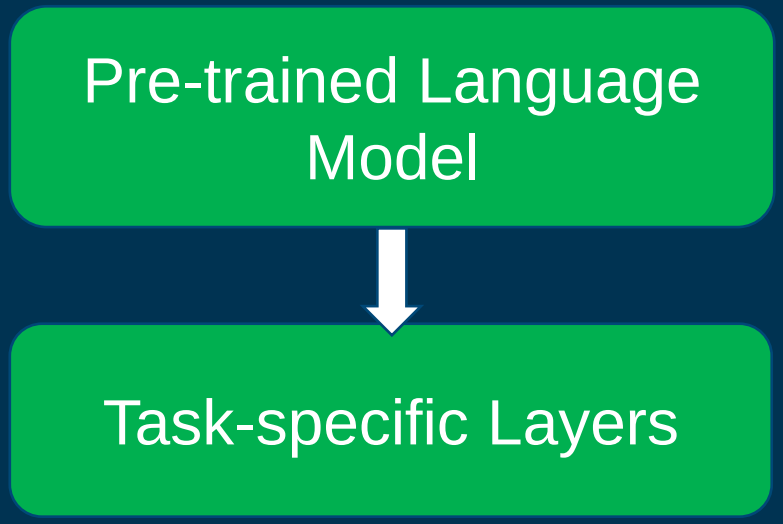
- There are two main approaches for applying pre-trained language models to downstream tasks:
  - The feature-based approach
  - The fine-tuning approach

Applying pre-trained models



Feature-based

- ***Only*** fine-tune task-specific parameters



Fine-tuning

- fine-tune ***all*** parameters

Applying pre-trained models

Feature-based

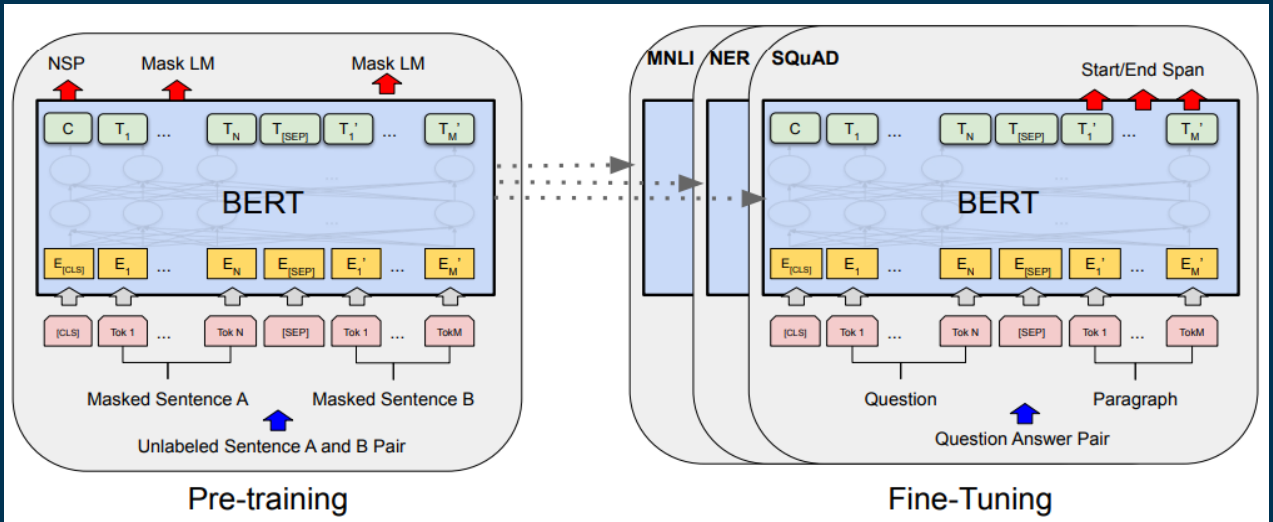
- ***Only*** fine-tune task-specific parameters
- Require less computational resources
- Preferred with small task-specific training data

Fine-tuning

- fine-tune ***all*** parameters
- Better performance
- Preferred with large task-specific training data

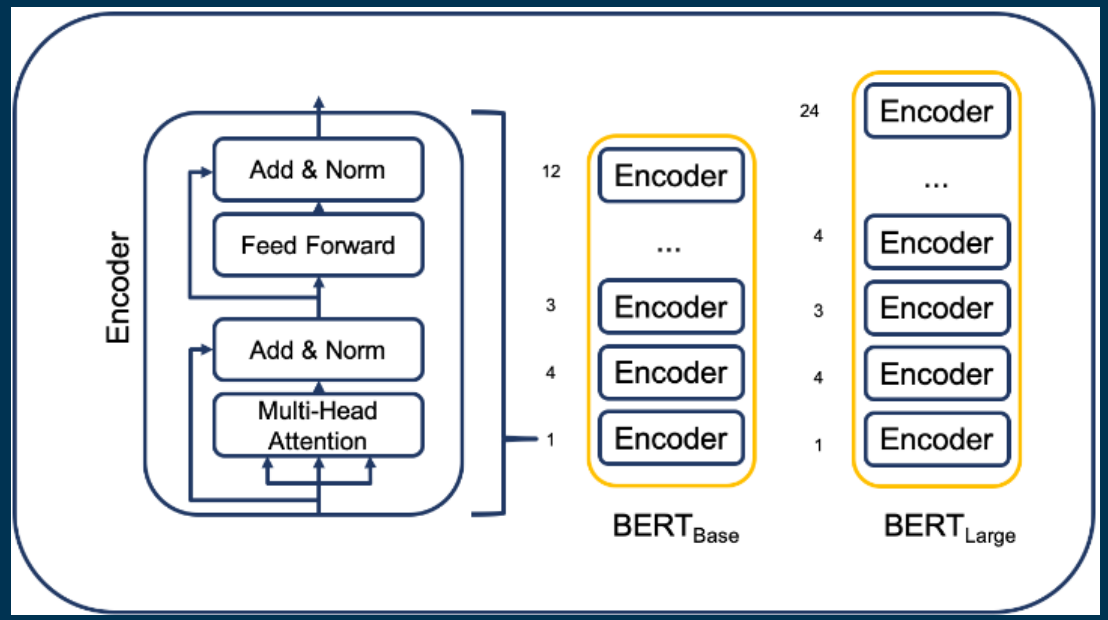
BERT

BERT



- There are two steps in BERT framework: pre-training and fine-tuning.
- Pre-training: trained on unlabeled data over different pre-training tasks.
- Fine-tuning: parameters are fine-tuned using labeled data from the downstream tasks(ner, sequence labeling, …)
- Unified architecture across different tasks.

Model Architecture



- Multi-layer bidirectional Transformer encoder
- **BERTBASE** (L=12, H=768, A=12, Total Parameters=110M)
- **BERTLARGE** (L=24, H=1024, A=16, Total Parameters=340M).

BERTBASE same model size as OpenAI GPT but uses **bidirectional self-attention**

L : number of layers | H : hidden size | A : number of self-attention heads.

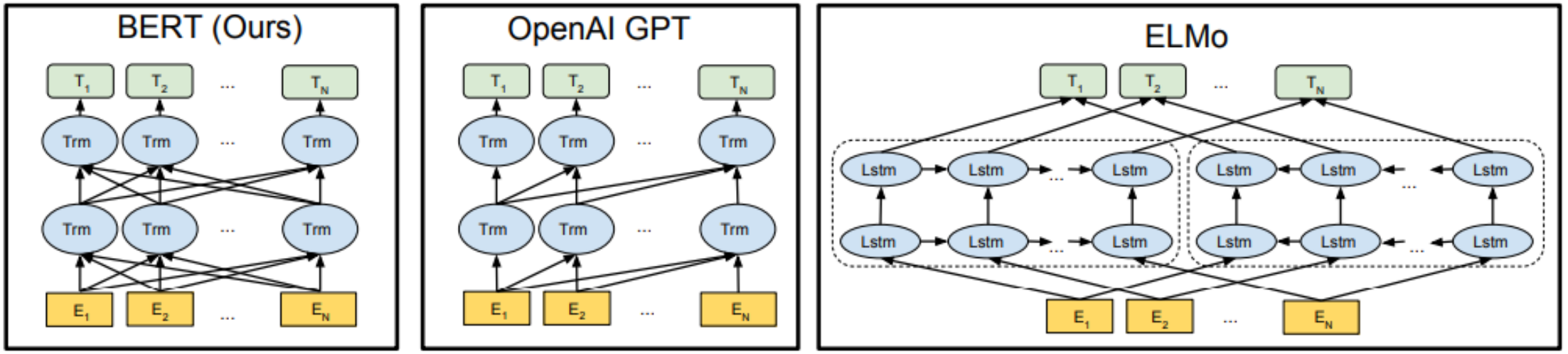


Figure 3: Differences in pre-training model architectures. BERT uses a bidirectional Transformer. OpenAI GPT uses a left-to-right Transformer. ELMo uses the concatenation of independently trained left-to-right and right-to-left LSTMs to generate features for downstream tasks. Among the three, only BERT representations are jointly conditioned on both left and right context in all layers. In addition to the architecture differences, BERT and OpenAI GPT are fine-tuning approaches, while ELMo is a feature-based approach.

Input Representations

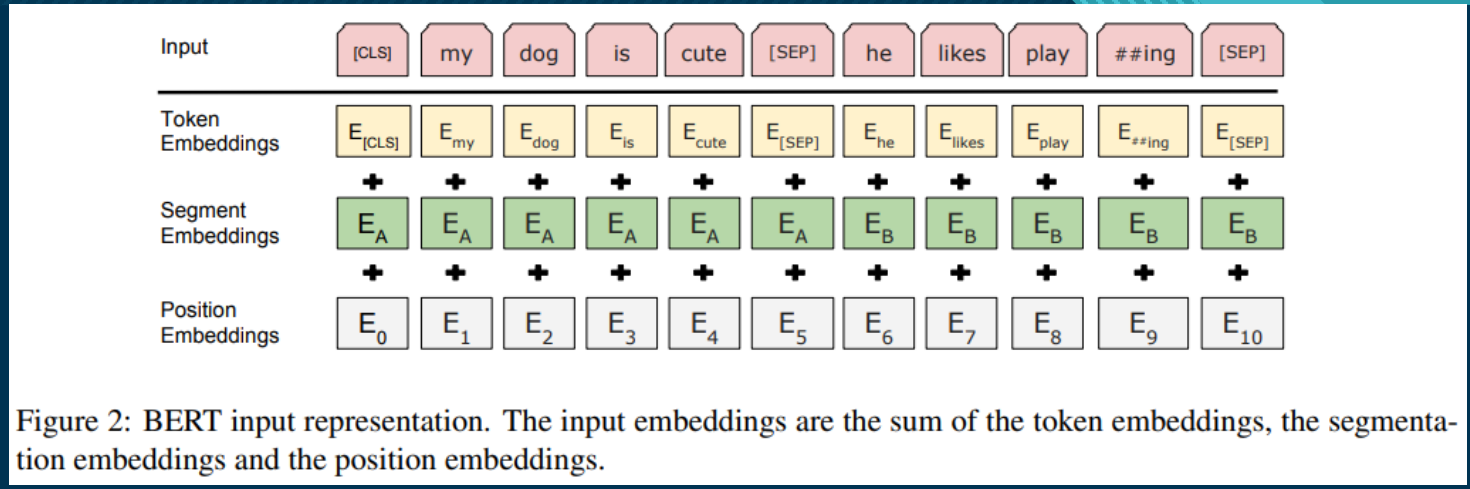


Figure 2: BERT input representation. The input embeddings are the sum of the token embeddings, the segmentation embeddings and the position embeddings.

Input : To make BERT handle a variety of down-stream tasks, always representing an input as a long sequence.

- A “sequence” refers to the input token sequence to BERT which may be a single sentence or two sentences packed together (e.g., h Question, Answer).
- The first token of every sequence is always a special classification token ([CLS]).

Input Representations

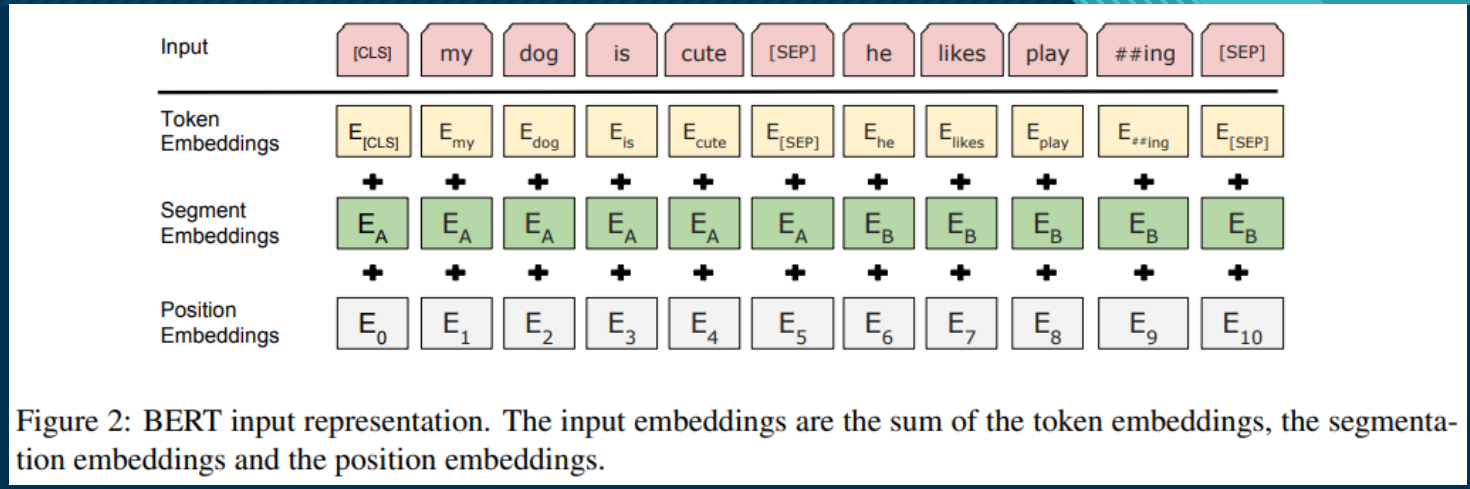


Figure 2: BERT input representation. The input embeddings are the sum of the token embeddings, the segmentation embeddings and the position embeddings.

Token Embeddings : WordPiece embeddings

Segment Embeddings : Separate the sentences in sequence.

- Token ([SEP])
- Add a learned embedding E_A, E_B

Position Embeddings : Sinusoid or any PE.

Input = Token Embeddings + Segment Embeddings + Position Embeddings

Pre-training data

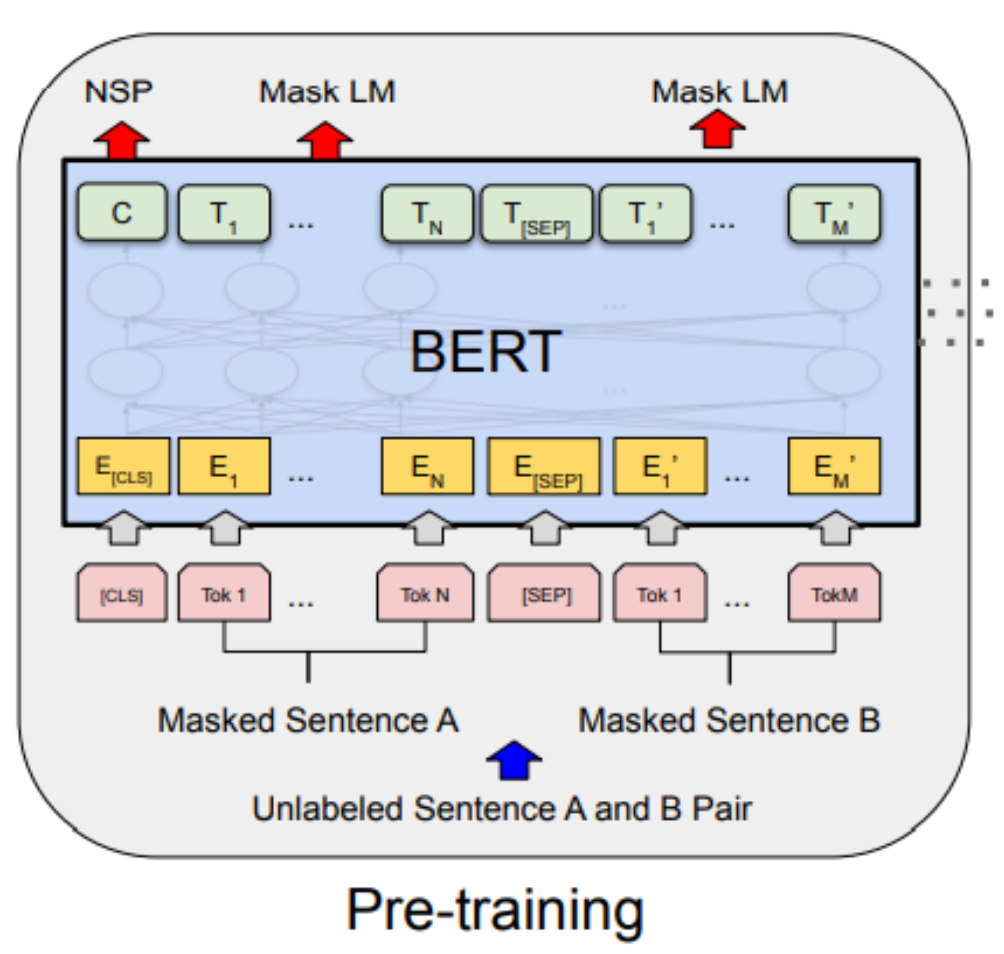
Pre-training corpus :

- BooksCorpus (800M words) (Zhu et al., 2015)
- English Wikipedia (2,500M words, only the text passages and ignore lists, tables, and headers).

- Use a document-level corpus rather than sentence-level (Billion Word Benchmark) to extract long contiguous sequences.

Pre-training BERT

Pre-training BERT



- Using two unsupervised tasks
  - Task #1: Masked LM

```
my dog is hairy → my dog is [MASK]
```

  - Task #2: Next Sentence Prediction (NSP)

```
Input = [CLS] the man went to [MASK] store [SEP]
        he bought a gallon [MASK] milk [SEP]
Label = IsNext
```

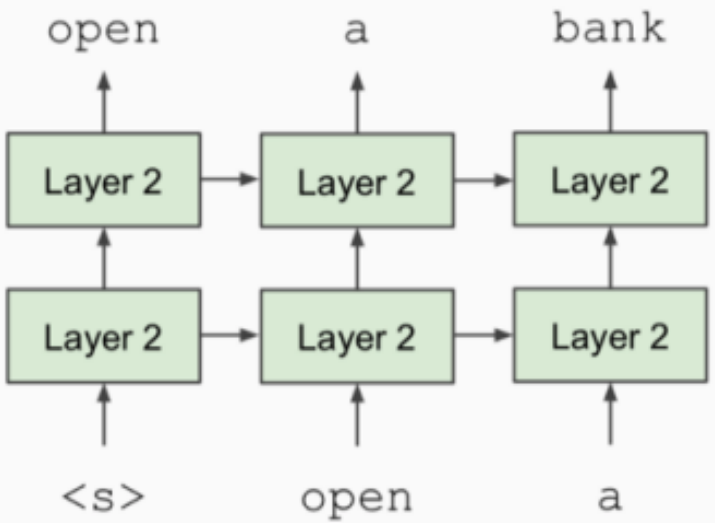
Task #1: Masked LM

- Deep bidirectional model is strictly more powerful than either left-to-right model or right-to-left model

- But standard conditional language models can only be trained left-to-right or right-to-left.
- Reason : Words can "see themselves" in bidirectional context

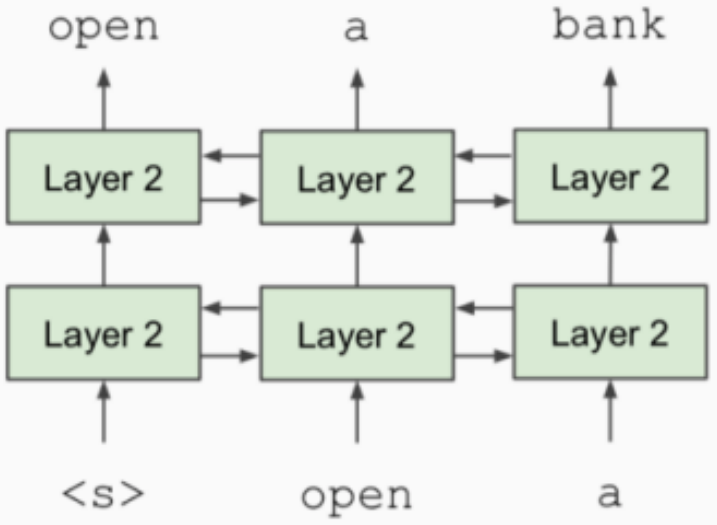
Unidirectional context

Build representation incrementally



Bidirectional context

Words can “see themselves”



Task #1: Masked LM

- Solution : Mask 15% input tokens at random, and then predict masked.

```
my dog is hairy → my dog is [MASK]
```

- Too little masking: too expensive to train
- Too much masking: not enough context
- Mismatch between pre-training and fine-tuning : [MASK] token does not appear during fine-tuning.

Solution

Don't always replace “masked” words with the actua l [MASK] token. Instead:

- 80% of the time: Replace the word with th e [MASK] token

  =>*my dog is hairy → my dog is [MASK]*

- 10% of the time: Replace the word with a random word

  => *my dog is hairy → my dog is apple*

- 10% of the time: Keep the word unchang ed

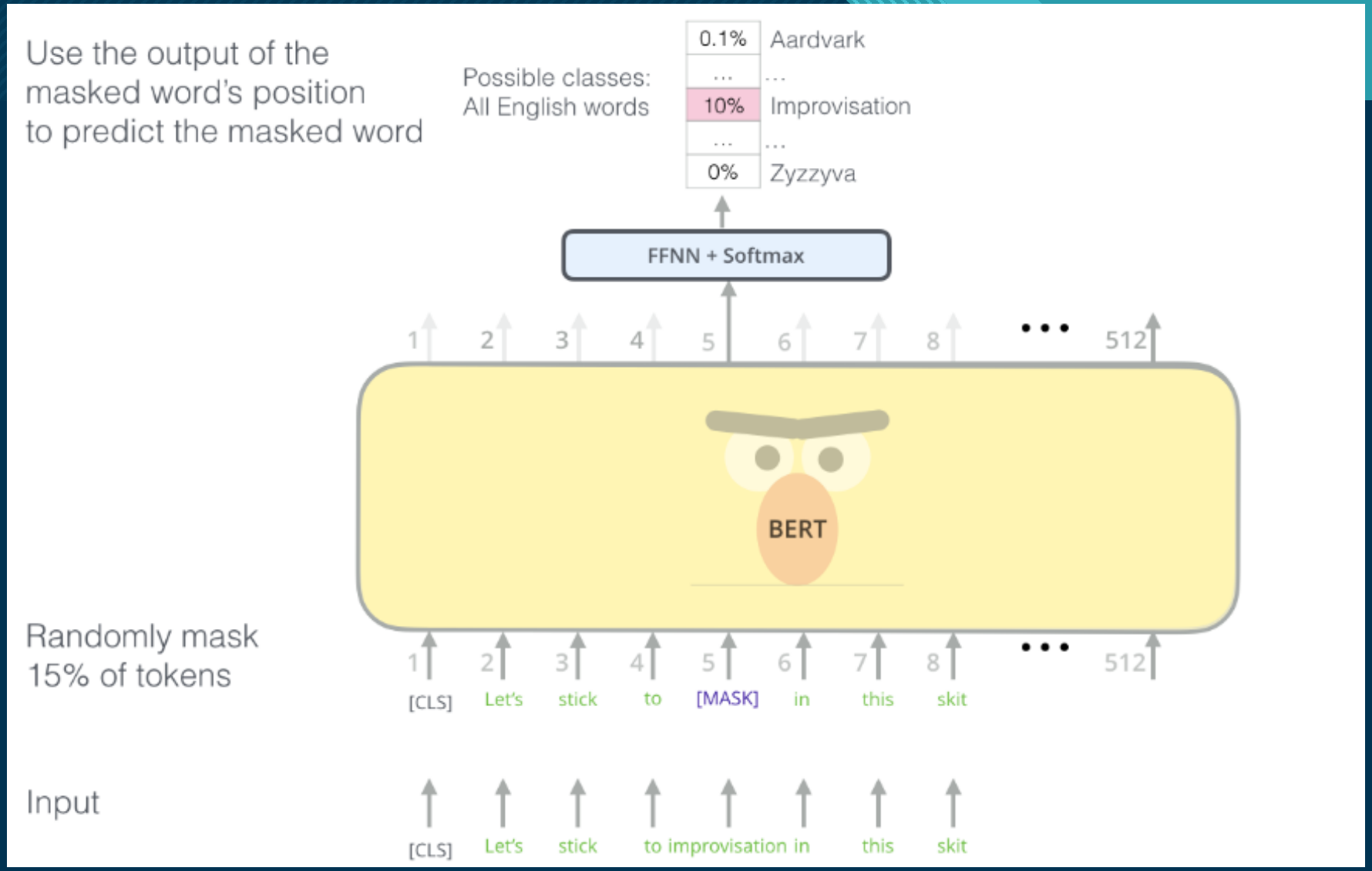
  => *my dog is hairy → my dog is hairy*

Advantage

=> Encoder not know which words it will be asked to predict or which have been replaced by random words => forced to keep a distributional contextual representation of every input token.

=> Random replacement only occurs for 1.5% of all tokens (i.e., 10% of 15%), this does not seem to harm the model’s language understanding capability.

Task #1: Masked LM - Overview



https://jalammar.github.io/illustrated-bert/

Task #2: Next Sentence Prediction (NSP)

- Many important downstream tasks based on understanding the *relationship* between two sentences, which is not directly captured by language modeling

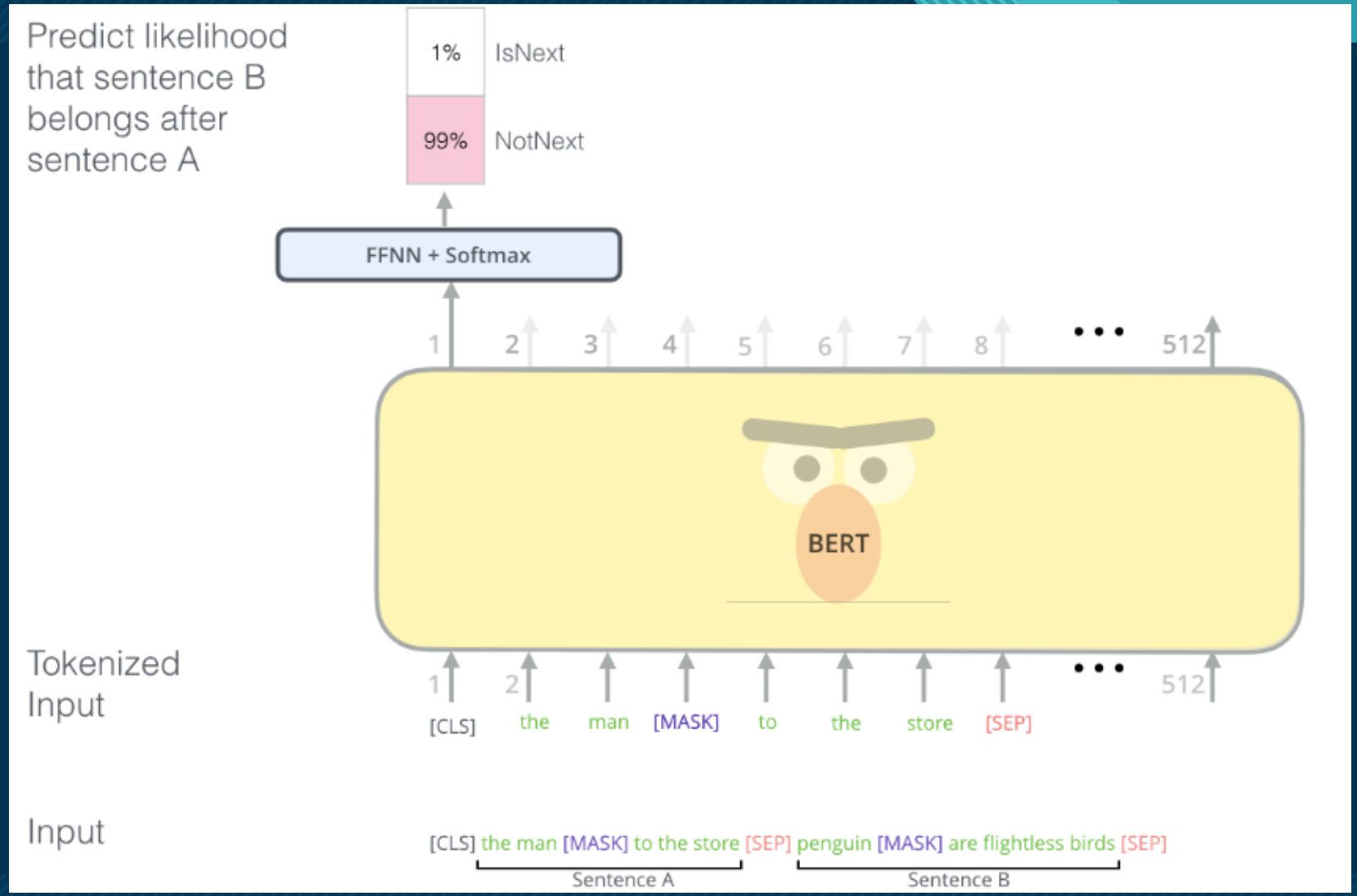
=> binarized next sentence prediction task

- When choosing the sentences A and B for each pretraining example:
  - 50% of the time B is the actual next sentence that follows A (labeled as IsNext)
  - 50% of the time it is a random sentence from the corpus (labeled as NotNext).

Input = `[CLS] the man went to [MASK] store [SEP]`
`he bought a gallon [MASK] milk [SEP]`
Label = `IsNext`

Input = `[CLS] the man [MASK] to the store [SEP]`
`penguin [MASK] are flight ##less birds [SEP]`
Label = `NotNext`

Task #2: Next Sentence Prediction (NSP) - Overview



https://jalammar.github.io/illustrated-bert/

Pre-training BERT

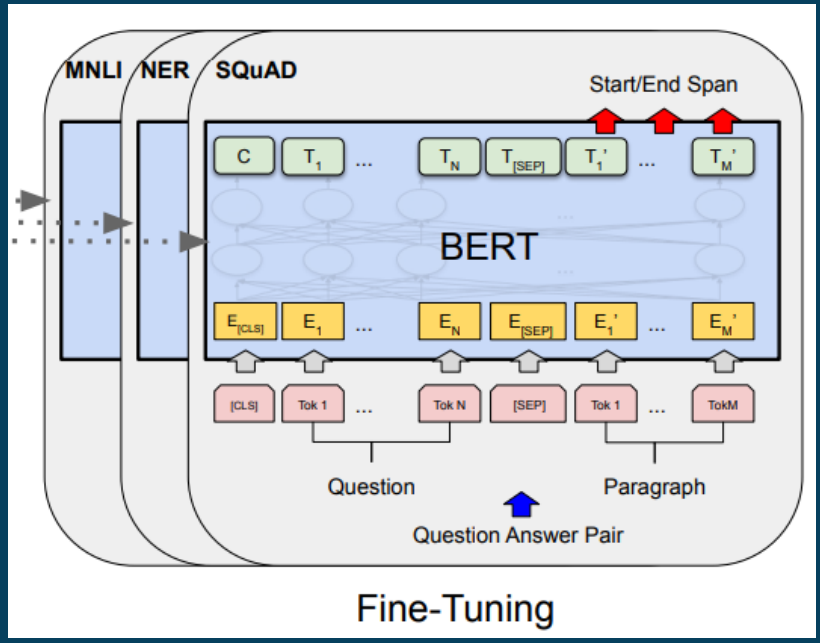
- After Pre-training step, BERT can understand language.
- More specifically, we will obtain **the representation of each token in the given input** after finishing Pre-training step.

- To solve downstream tasks, we need **fine-tuning BERT**

Fine-tuning BERT

Special feature

- A special feature of BERT that previous embedding models have never had is that the training results can be fine-tuned.
- We will add to the model architecture an output layer to customize the training task.
- During the fine-tuning process, all the parameters of the transfer learning layers will be fine-tuned.



Advantages of Fine-tuning

- **Quicker development**: The pre-trained BERT model weights already encode a lot of information about our language. As a result, it takes much less time to train our fine-tuned model

- **Less data:** In addition and perhaps just as important, because of the pre-trained weights this method allows us to fine-tune our task on a much smaller dataset than would be required in a model that is built from scratch. A major drawback of NLP models built from scratch is that we often need a prohibitively large dataset in order to train our network to reasonable accuracy, meaning a lot of time and energy had to be put into dataset creation.

- **Better results**: This simple fine-tuning procedure (typically adding one fully-connected layer on top of BERT and training for a few epochs) was shown to achieve state of the art results with minimal task-specific adjustments for a wide variety of tasks: classification, language inference, semantic similarity, question answering, etc.

Fine-tuning steps

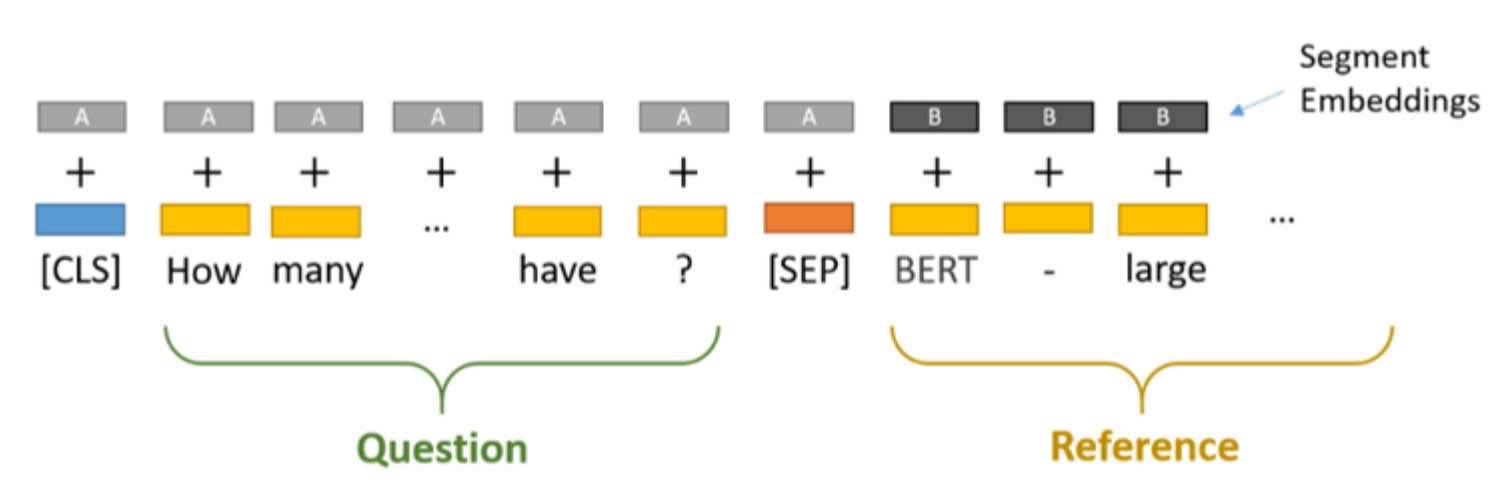
Step 1: Embedding all the tokens. [CLS] and [SEP] tokens to mark the beginning of the question and the space between the two sentences. These 2 tokens determine the Start/End parts of the output sentence.

Step 2: The vector embeddings are then fed into the multi-head attention architecture with multiple blocks of code.We obtain an output vector at the encoder.

Step 3: We will pass to decoder the encoder's output vector and the decoder's embedding input vector to compute the encoder-decoder attention. Using the liner layer and softmax to obtain the probability distribution.

Step 4: We will fix the result of the Question so that it matches the Question in the input. The remaining positions will be the Start/End Span extension corresponding to the answer found from the input sentence

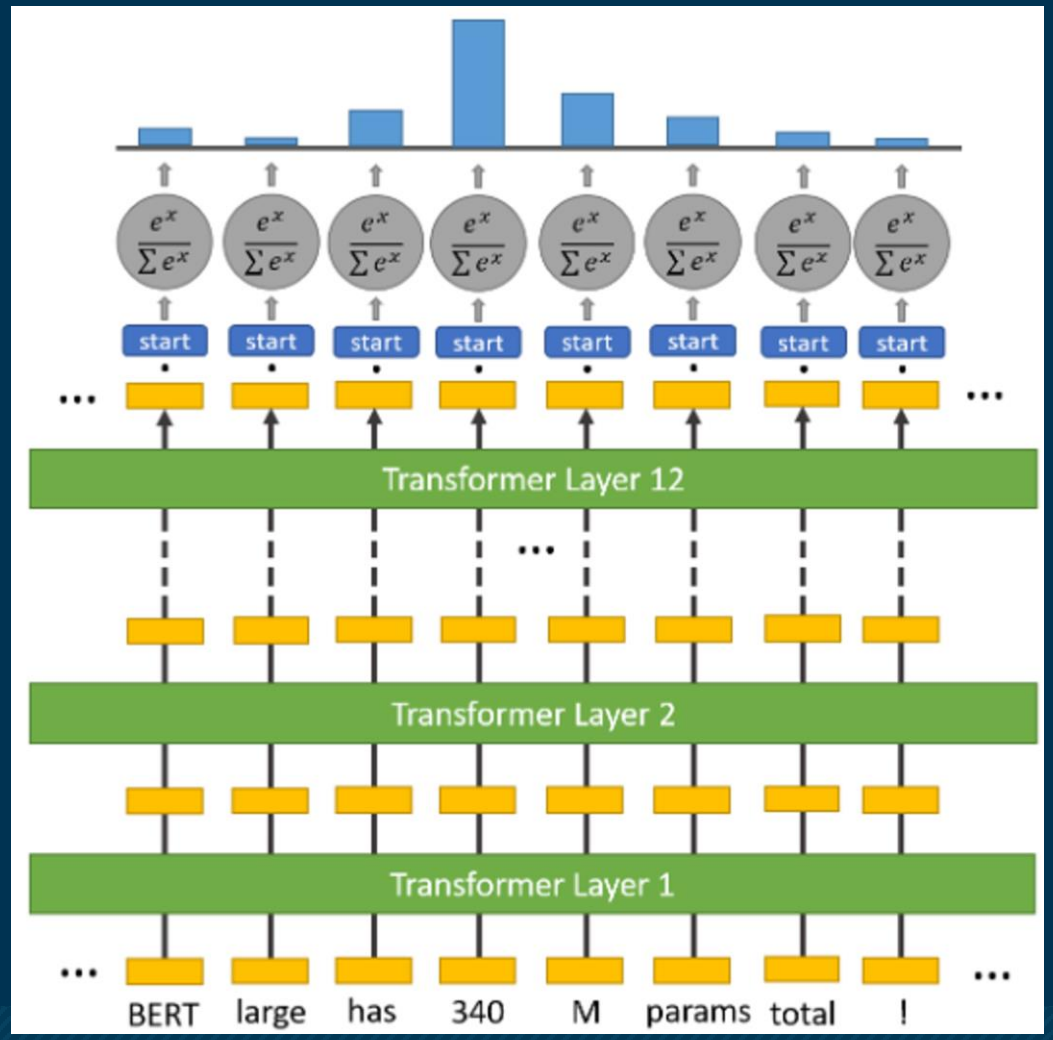
Example for QA task

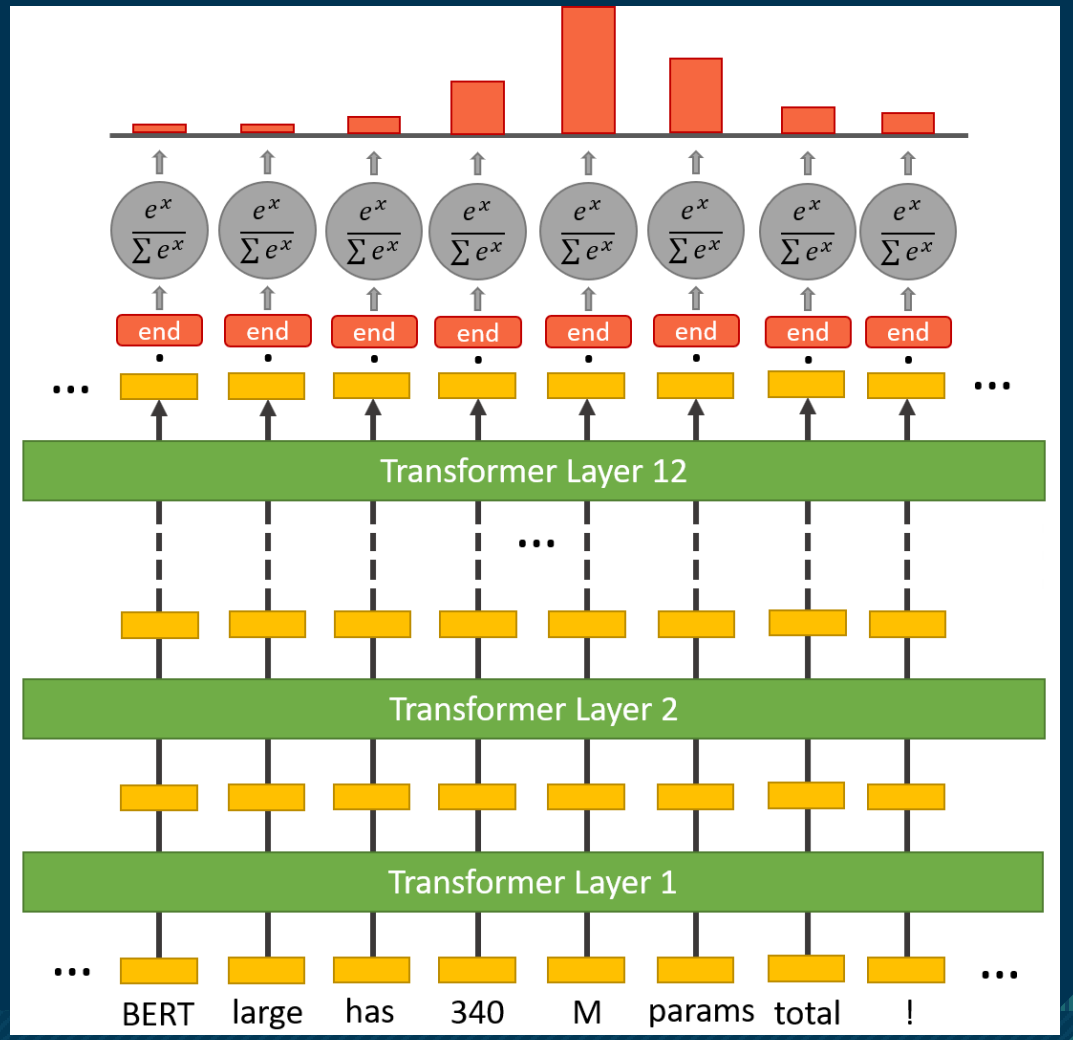


Question: How many parameters does BERT-large have?

Reference: BERT-large is really big… it has 24 layers and an embedding size of 1024, for a total of 340M parameters! Altogether it is 1.34GB, so expect it to take a couple minutes to download to your Colab instance.

Example - Start & End Token Classifiers





Thank You!