AMRITA VISHWA VIDYAPEETHAM
DEEMED TO BE UNIVERSITY

Introduction to Deep Learning

BERT

Bidirectional Encoder Representations from Transformers is a transformer-based machine learning technique for natural language processing pre-training developed by Google. BERT was created and published in 2018 by Jacob Devlin and his colleagues from Google

Courtesy:, analytics vidya,fast.ai, coursera: AndrewNG, http://jalammar.github.io/

In 2018, Google introduced and open-sourced BERT (11 NLP tasks).

In December 2019, BERT was applied to more than 70 different languages

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

Jacob Devlin **Ming-Wei Chang** **Kenton Lee** **Kristina Toutanova**
Google AI Language
{jacobdevlin,mingweichang,kentonl,kristout}@google.com

Abstract

We introduce a new language representation model called **BERT**, which stands for **B**idirectional **E**ncoder **R**epresentations from **T**ransformers. Unlike recent language representation models (Peters et al., 2018a; Radford et al., 2018), BERT is designed to pre-train deep bidirectional representations from unlabeled text by jointly conditioning on both left and right context in all layers. As a result, the pre-trained BERT model can be fine-tuned with just one additional output layer to create state-of-the-art models for a wide range of tasks, such as question answering and language inference, without substantial task-specific architecture modifications.

BERT is conceptually simple and empirically powerful. It obtains new state-of-the-art results on eleven natural language processing tasks, including pushing the GLUE score to 80.5% (7.7% point absolute improvement), MultiNLI accuracy to 86.7% (4.6% absolute improvement), SQuAD v1.1 question answering Test F1 to 93.2 (1.5 point absolute improvement) and SQuAD v2.0 Test F1 to 83.1 (5.1 point absolute improvement).

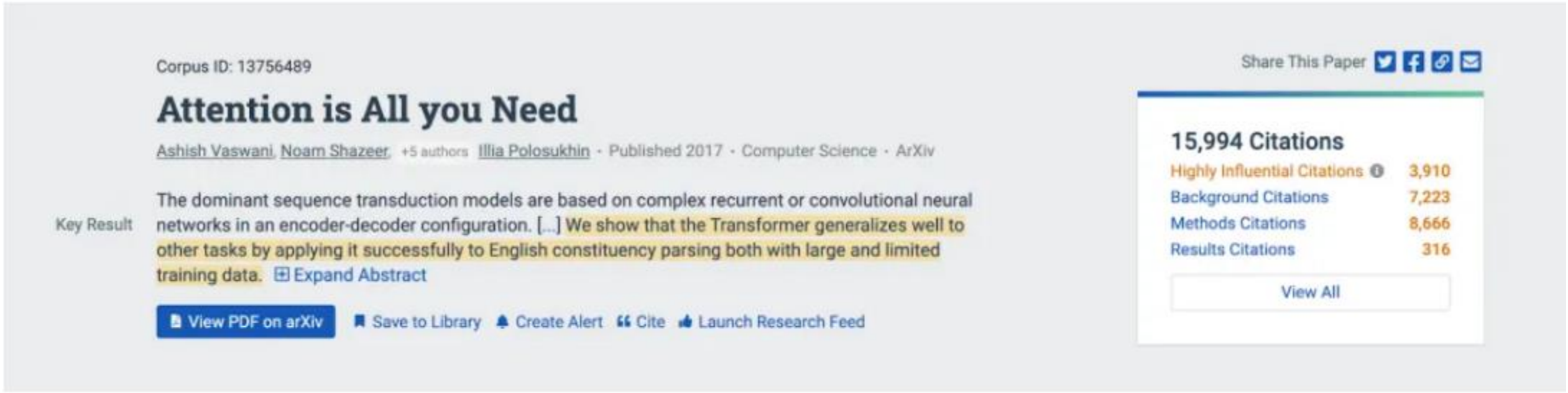
1 Introduction

There are two existing strategies for applying pre-trained language representations to downstream tasks: *feature-based* and *fine-tuning*. The feature-based approach, such as ELMo (Peters et al., 2018a), uses task-specific architectures that include the pre-trained representations as additional features. The fine-tuning approach, such as the Generative Pre-trained Transformer (OpenAI GPT) (Radford et al., 2018), introduces minimal task-specific parameters, and is trained on the downstream tasks by simply fine-tuning *all* pre-trained parameters. The two approaches share the same objective function during pre-training, where they use unidirectional language models to learn general language representations.

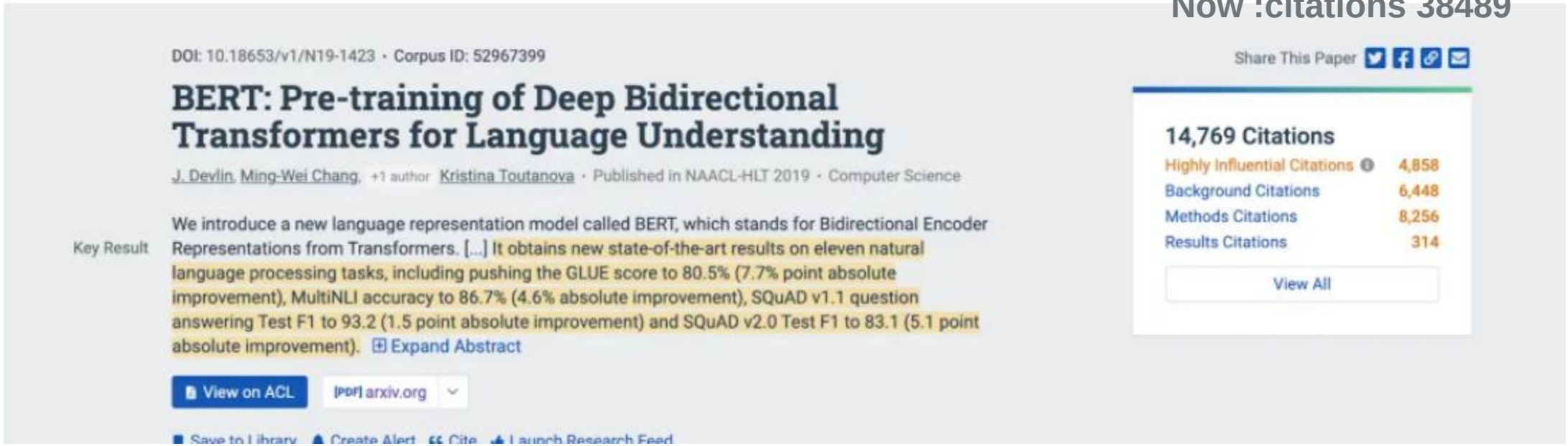
We argue that current techniques restrict the power of the pre-trained representations, especially for the fine-tuning approaches. The major limitation is that standard language models are unidirectional, and this limits the choice of architectures that can be used during pre-training. For example, in OpenAI GPT, the authors use a left-to-right architecture, where every token can only attend to previous tokens in the self-attention layers of the Transformer (Vaswani et al., 2017). Such re-

2020 status

Now : Citations 41978



Now :citations 38489



NLP's ImageNet moment

BERT basically uses the concept of pre-training the model on a very large dataset in an unsupervised manner for language modeling. A pre-trained model on a very large dataset has the capability to better understand the context of the input sentence. After pre-training, the model can be fine-tuned on the task-specific supervised dataset to achieve good results.

BERT is pre-trained from unlabeled data extracted from BooksCorpus (800M words) **and English Wikipedia** (2,500M words)

- Sequence-to-sequence based language generation tasks such as:
  - Question answering
  - Abstract summarization
  - Sentence prediction
  - Conversational response generation
- Natural language understanding tasks such as:
  - Polysemy and Coreference (words that sound or look the same but have different meanings) resolution
  - Word sense disambiguation
  - Natural language inference
  - Sentiment classification

One Hot Encoding, TF-IDF is a statistical measure used to determine the mathematical significance of words in documents. Term frequency and Inverse Document Frequency

- .

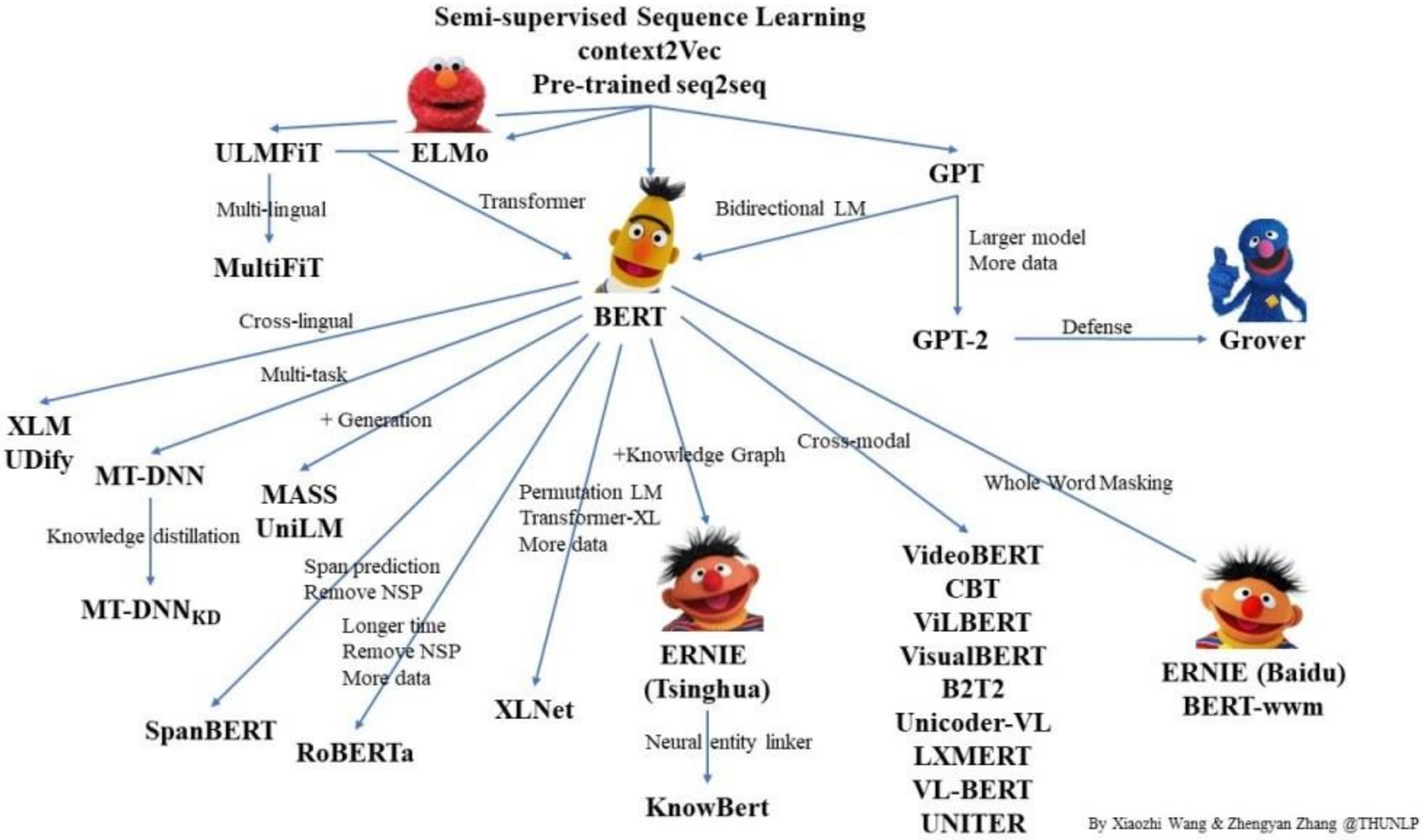
Word2Vec and **Glove** Word-embeddings showed that we can use a vector (a list of numbers) to properly represent words in a way that captures *semantic* or meaning-related relationships : Skip Gram and Common Bag Of Words (CBOW)

ELMo: Context Matters: ELMo looks at the entire sentence before assigning each word in it an embedding. It uses a bi-directional LSTM trained on a specific task to be able to create those embeddings

Universal Language Model FIne-Tuning(ULMFIT) is a transfer learning technique which can help in various NLP tasks : involves a 3-layer AWD-LSTM- Weight-Dropped LSTM

OpenAI Transformer: Pre-training a Transformer Decoder for Language Modeling

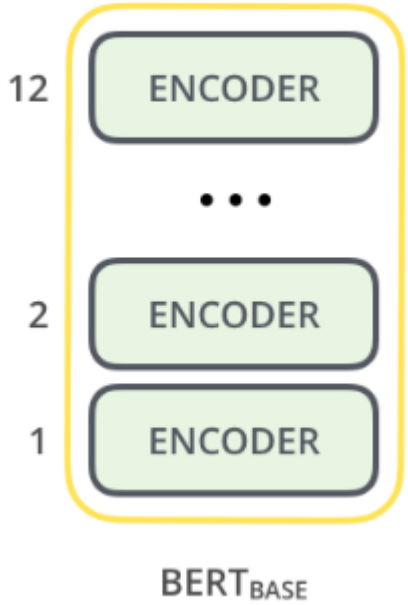
BERT: The openAI transformer gave us a fine-tunable pre-trained model based on the Transformer. But something went missing in this transition from LSTMs to Transformers. ELMo's language model was bi-directional, but the openAI transformer only trains a forward language model. Could we build a transformer-based model whose language model looks both forward and backwards - BERT


Semi-supervised Sequence Learning
context2Vec
Pre-trained seq2seq
ELMo
ULMFiT
Multi-lingual
MultiFiT
Transformer
BERT
Bidirectional LM
GPT
Larger model
More data
GPT-2
Defense
Grover
Cross-lingual
XLM
UDify
Multi-task
MT-DNN
Knowledge distillation
MT-DNN$_{KD}$
+ Generation
MASS
UniLM
Span prediction
Remove NSP
SpanBERT
Longer time
Remove NSP
More data
RoBERTa
Permutation LM
Transformer-XL
More data
XLNet
+Knowledge Graph
ERNIE
(Tsinghua)
Neural entity linker
KnowBert
Cross-modal
VideoBERT
CBT
ViLBERT
VisualBERT
B2T2
Unicoder-VL
LXMERT
VL-BERT
UNITER
Whole Word Masking
ERNIE (Baidu)
BERT-wwm
By Xiaozhi Wang & Zhengyan Zhang @THUNLP

Research groups and separate factions of Google are fine-tuning the BERT model architecture with supervised training to either optimize it for efficiency (modifying the learning rate, for example) or specialize it for certain tasks by pre-training it with certain contextual representations. Some examples include:

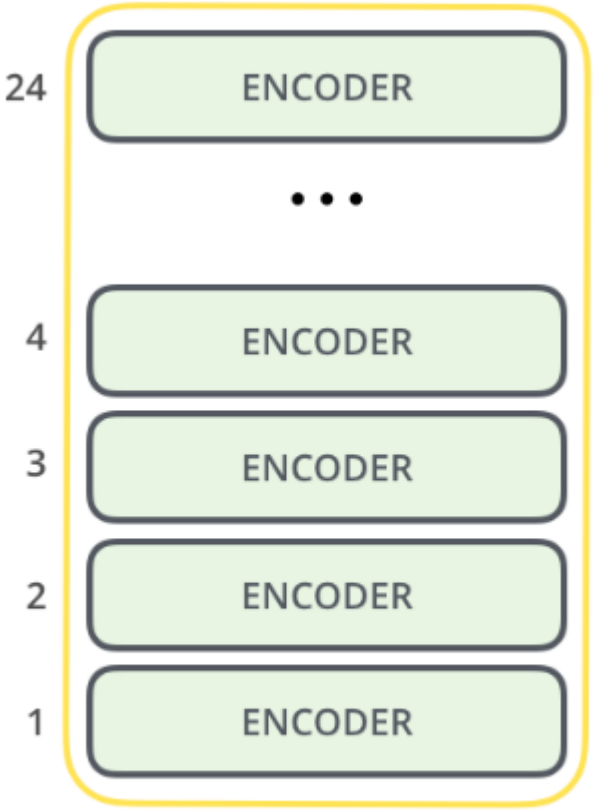
- **patentBERT** - a BERT model fine-tuned to perform patent classification.
- **docBERT** - a BERT model fine-tuned for document classification.
- **bioBERT** - a pre-trained biomedical language representation model for biomedical text mining.
- **VideoBERT** - a joint visual-linguistic model for process unsupervised learning of an abundance of unlabeled data on Youtube.
- **SciBERT** - a pretrained BERT model for scientific text
- **G-BERT** - a BERT model pretrained using medical codes with hierarchical representations using graph neural networks (GNN) and then fine-tuned for making medical recommendations.
- **TinyBERT** by Huawei - a smaller, "student" BERT that learns from the original "teacher" BERT, performing transformer distillation to improve efficiency. TinyBERT produced promising results in comparison to BERT-base while being 7.5 times smaller and 9.4 times faster at inference.
- **DistilBERT** by HuggingFace - a supposedly smaller, faster, cheaper version of BERT that is trained from BERT, and then certain architectural aspects are removed for the sake of efficiency.

BERT is basically a trained Transformer Encoder stack.



$BERT_{BASE}$

Both BERT model sizes have a large number of encoder layers (which the paper calls Transformer Blocks) – twelve for the Base version, and twenty four for the Large version. These also have larger feedforward-networks (768 and 1024 hidden units respectively), and more attention heads (12 and 16 respectively)
Than
the default configuration in the reference implementation of the Transformer in the initial paper (6 encoder layers, 512 hidden units, and 8 attention heads).



$BERT_{LARGE}$

1 - Semi-supervised training on large amounts of text (books, wikipedia..etc).

The model is trained on a certain task that enables it to grasp patterns in language. By the end of the training process, BERT has language-processing abilities capable of empowering many models we later need to build and train in a supervised way.

Semi-supervised Learning Step

Model:

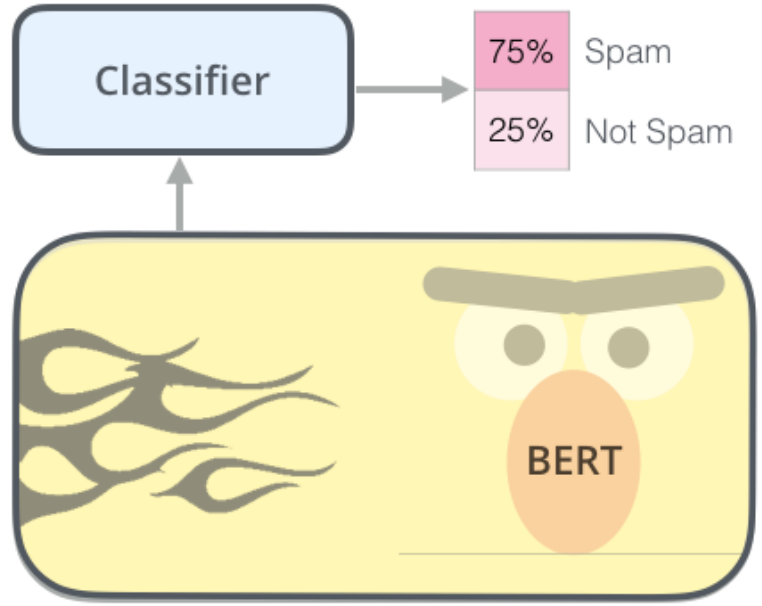
Dataset:

Objective: Predict the masked word (langauge modeling)

2 - Supervised training on a specific task with a labeled dataset.

Supervised Learning Step

Model: (pre-trained in step #1)



Dataset:

Email message	Class
Buy these pills	Spam
Win cash prizes	Spam
Dear Mr. Atreides, please find attached…	Not Spam

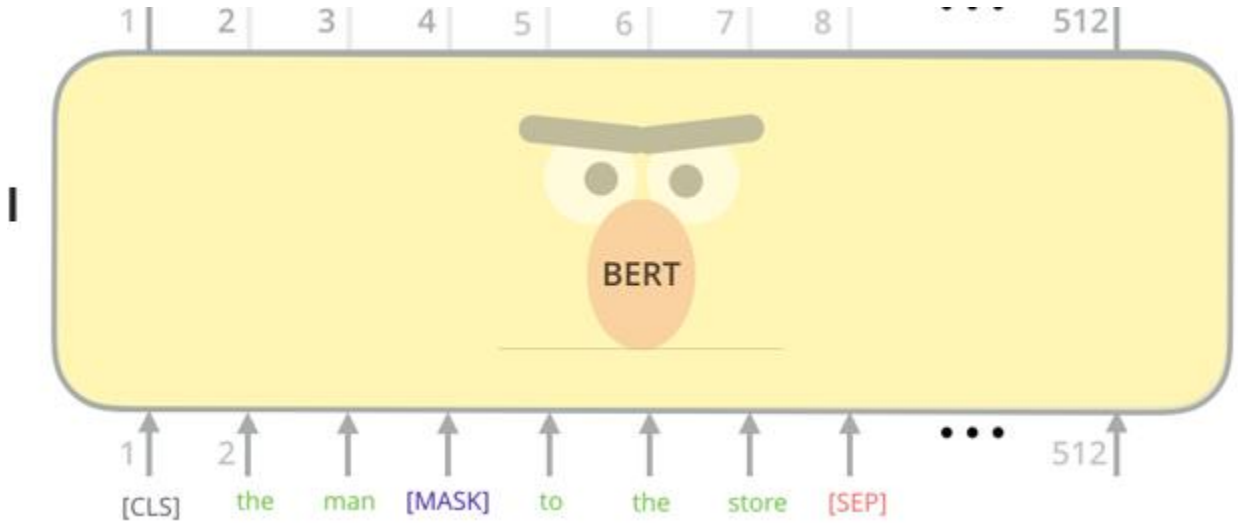
BERT uses two unsupervised strategies: Masked Language Model(MLM) and Next Sentence prediction(NSP) as part of pre-training.

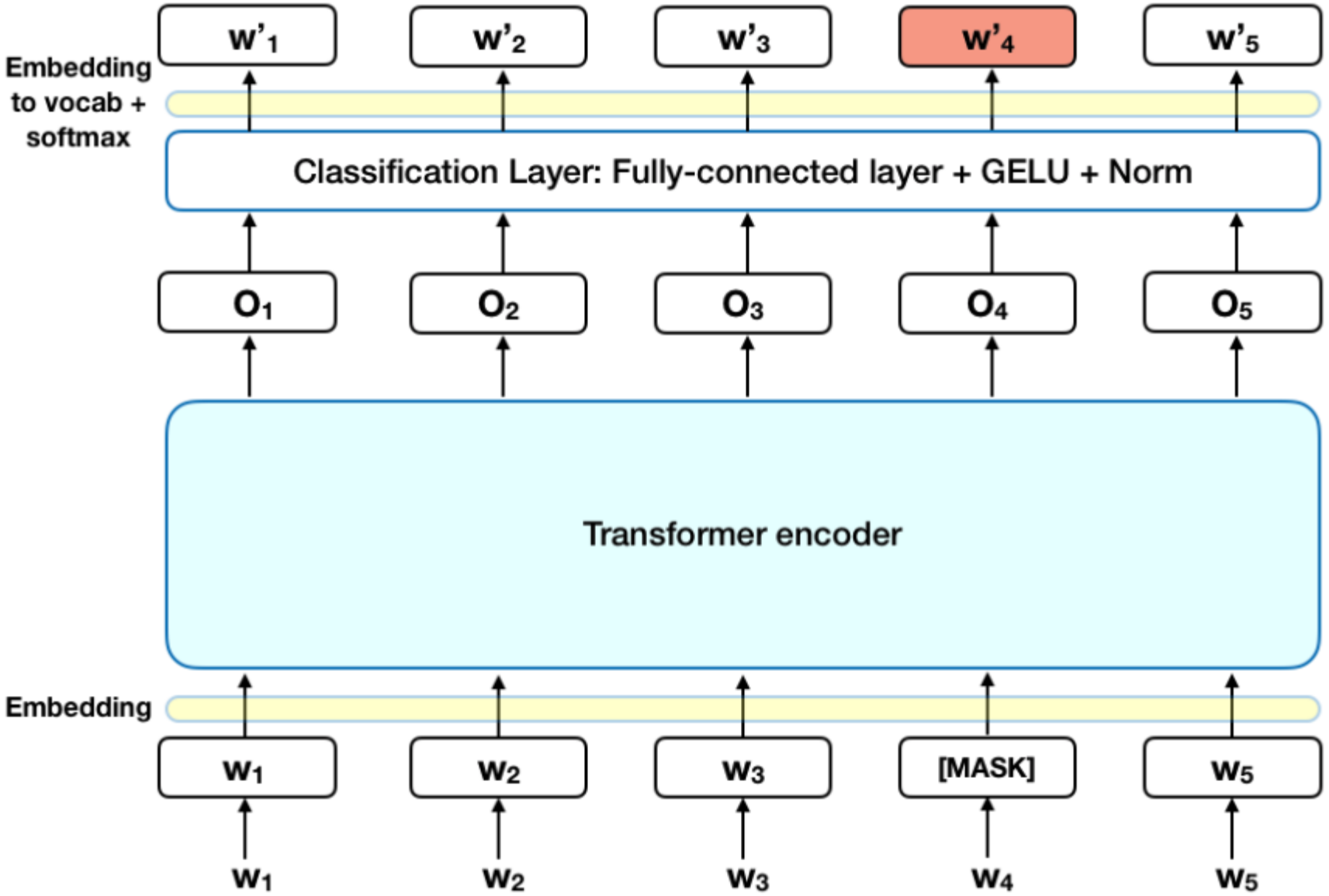
Masked LM (MLM)
Before feeding word sequences into BERT, 15% of the words in each sequence are replaced with a [MASK] token. The model then attempts to predict the original value of the masked words, based on the context provided by the other, non-masked, words in the sequence.

Next Sentence Prediction (NSP)
In the BERT training process, the model receives pairs of sentences as input and learns to predict if the second sentence in the pair is the subsequent sentence in the original document. During training, 50% of the inputs are a pair in which the second sentence is the subsequent sentence in the original document, while in the other 50% a random sentence from the corpus is chosen as the second sentence. The assumption is that the random sentence will be disconnected from the first sentence.

[CLS] and [SEP]

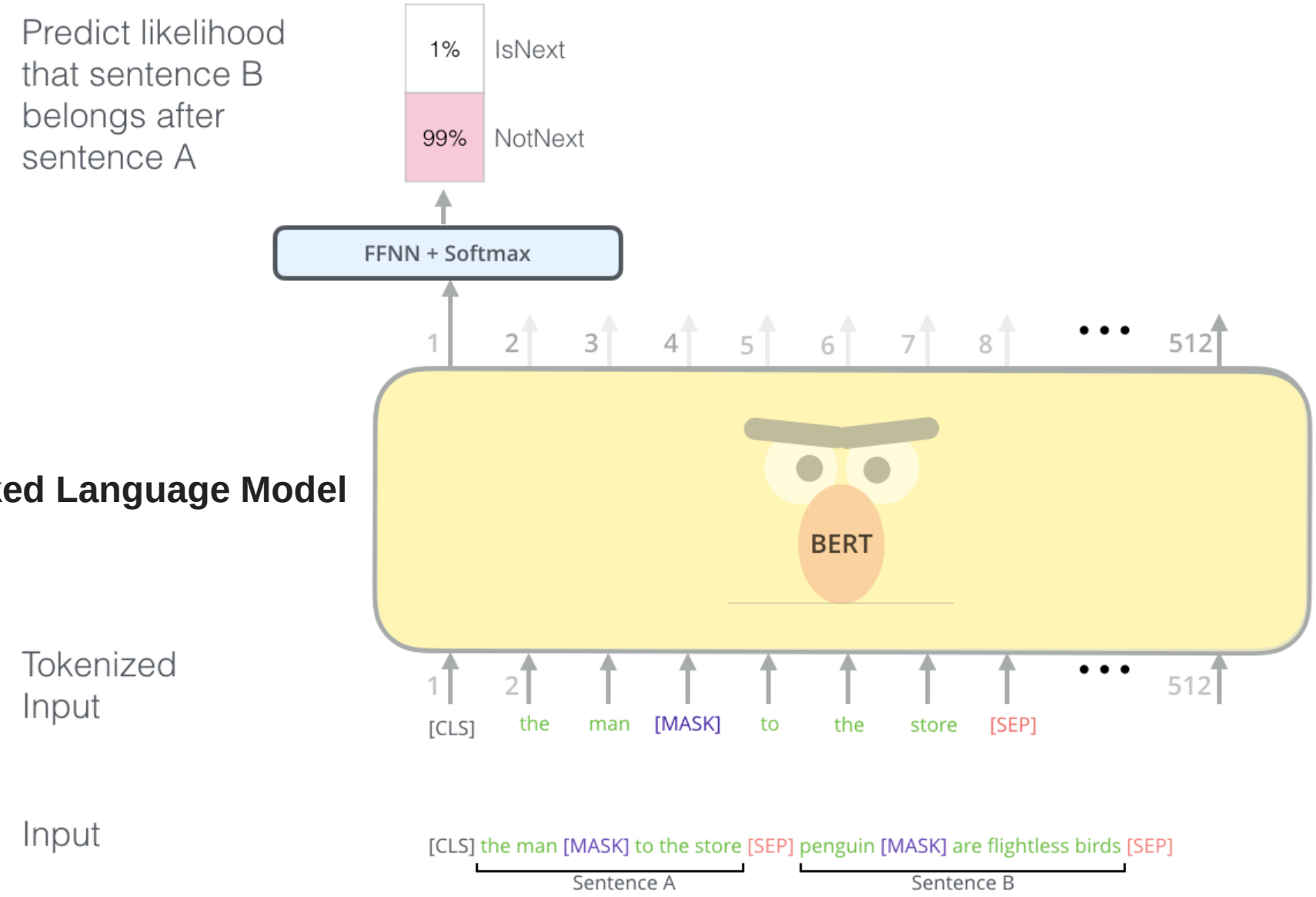
The whole input to the BERT has to be given a single sequence. BERT uses special tokens [CLS] and [SEP] to understand input properly. [SEP] token has to be inserted at the end of a single input. When a task requires more than one input such as NLI and Q-A tasks, [SEP] token helps the model to understand the end of one input and the start of another input in the same sequence input. [CLS] is a special classification token and the last hidden state of BERT corresponding to this token ($h_{[CLS]}$) is used for classification tasks




w'1
w'2
w'3
w'4
w'5
Embedding to vocab + softmax
Classification Layer: Fully-connected layer + GELU + Norm
O1
O2
O3
O4
O5
Transformer encoder
Embedding
w1
w2
w3
[MASK]
w5
w1
w2
w3
w4
w5

BERT uses two unsupervised strategies: Masked Language Model(MLM) and Next Sentence prediction(NSP) as part of pre-training.

Masked Language Model

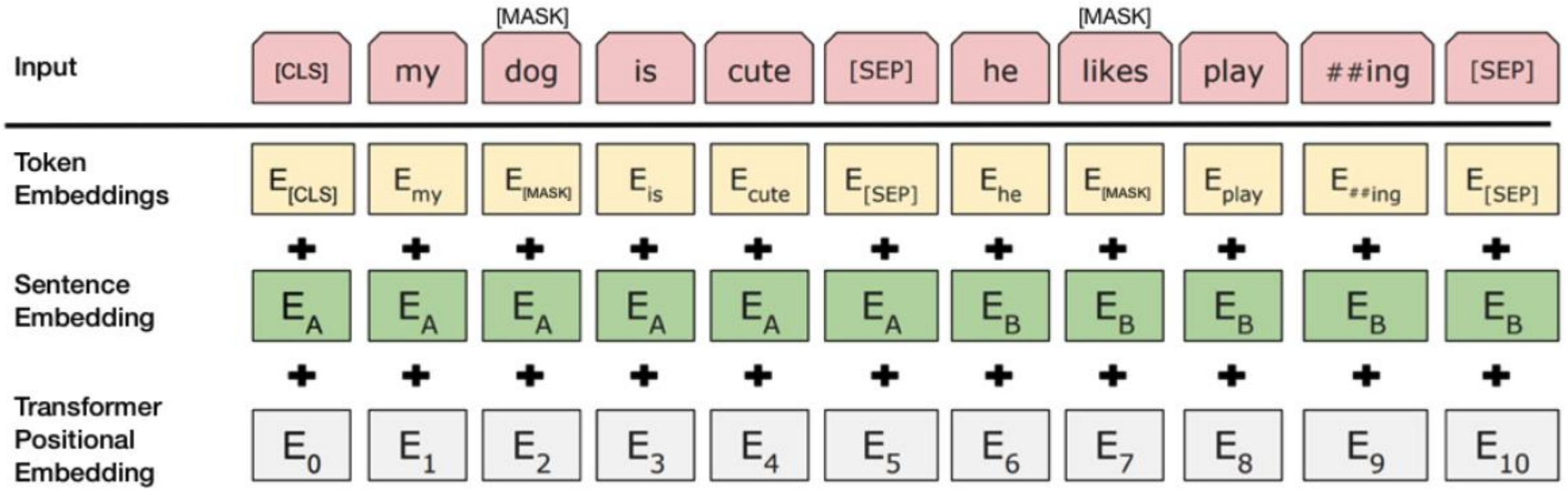


To help the model distinguish between the two sentences in training, the input is processed in the following way before entering the model:

1.A [CLS] token is inserted at the beginning of the first sentence and a [SEP] token is inserted at the end of each sentence.

2.A sentence embedding indicating Sentence A or Sentence B is added to each token. Sentence embeddings are similar in concept to token embeddings with a vocabulary of 2.

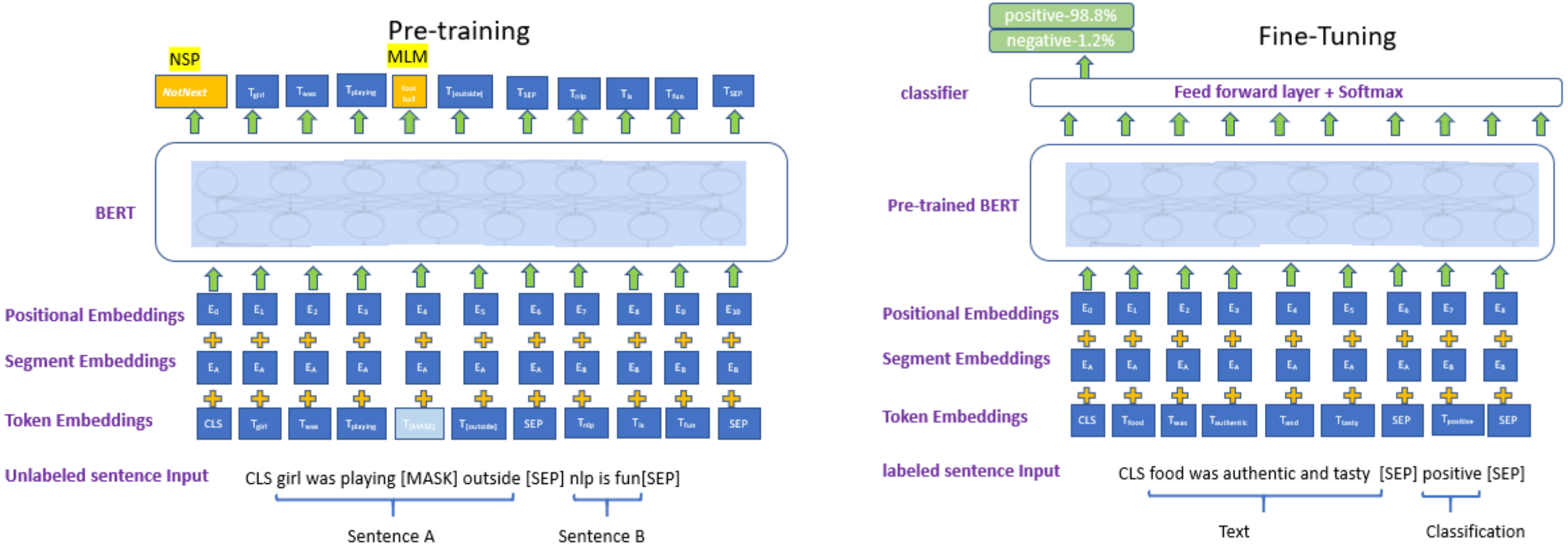
3.A positional embedding is added to each token to indicate its position in the sequence. The concept and implementation of positional embedding are presented in the Transformer paper.



To predict if the second sentence is indeed connected to the first, the following steps are performed:

1.The entire input sequence goes through the Transformer model.
2.The output of the [CLS] token is transformed into a 2×1 shaped vector, using a simple classification layer (learned matrices of weights and biases).
3.Calculating the probability of IsNextSequence with softmax.
When training the BERT model, Masked LM and Next Sentence Prediction are trained together, with the goal of minimizing the combined loss function of the two strategies.

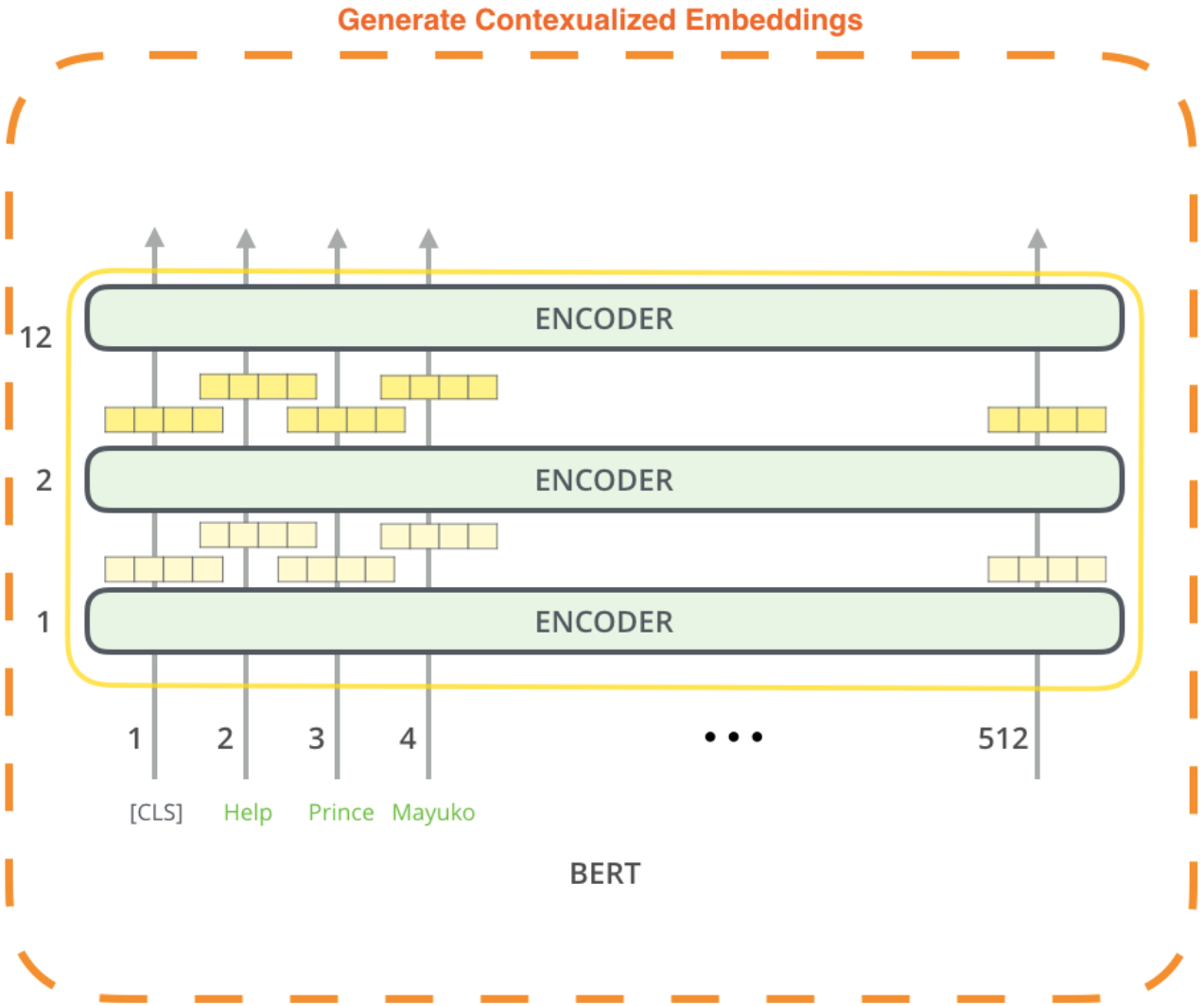
For fine-tuning the BERT model, we first initialize with the pre-trained parameters, and then all of the parameters are fine-tuned using labeled data from the downstream tasks.



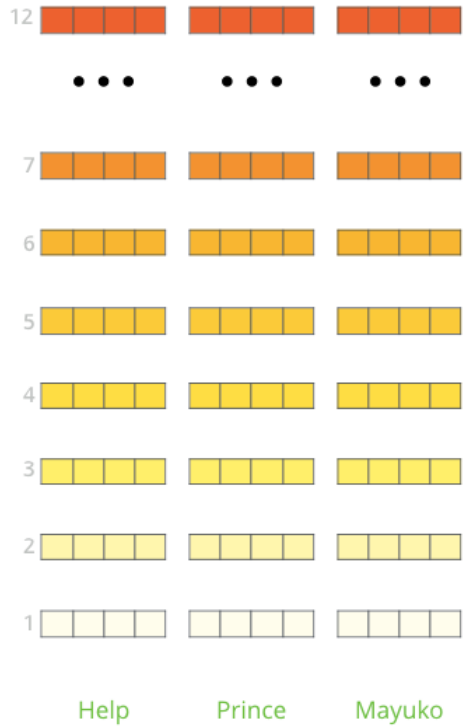
Fine-tuning is adding a layer of untrained neurons as a feedforward layer on top of the pre-trained BERT.

BERT for feature extraction

The fine-tuning approach isn't the only way to use BERT. Just like ELMo, you can use the pre-trained BERT to create contextualized word embeddings. Then you can feed these embeddings to your existing model



The output of each encoder layer along each token's path can be used as a feature representing that token.



But which one should we use?

Thank You

Namah Shivaya