https://yashuseth.blog/2019/06/12/bert-explained-faqs-understand-bert-working/

BERT Explained – A list of Frequently Asked Questions

Posted on June 12, 2019 by Yashu Seth

What is BERT?

BERT is a deep learning model that has given state-of-the-art results on a wide variety of natural language processing tasks. It stands for **Bidirectional Encoder Representations for Transformers**. It has been pre-trained on Wikipedia and BooksCorpus and requires task-specific fine-tuning.

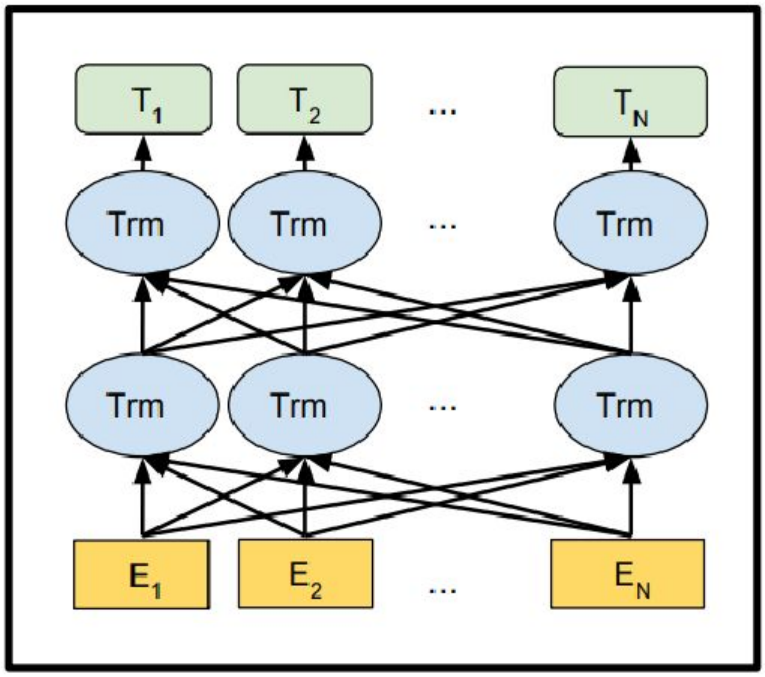
What is the model architecture of BERT?

BERT is a multi-layer bidirectional Transformer encoder. There are two models introduced in the paper.

- BERT base – 12 layers (transformer blocks), 12 attention heads, and 110 million parameters.
- BERT Large – 24 layers, 16 attention heads and, 340 million parameters.

For an in-depth understanding of the building blocks of BERT (aka Transformers), you should definitely check this awesome post – The Illustrated Transformers.

What is the flow of information of a word in BERT?



BERT ARCHITECTURE

(Source)

A word starts with its embedding representation from the embedding layer. Every layer does some multi-headed attention computation on the word representation of the previous layer to create a new intermediate representation. All these intermediate representations are of the same size. In the figure above, **E1** is the embedding representation, **T1** is the final output and **Trm** are the intermediate representations of the same token. In a 12-layers BERT model a token will have 12 intermediate representations.

What are the tasks BERT has been pre-trained on?

Masked Language Modeling and **Next Sentence Prediction**.

What is Masked Language Modeling?

Language Modeling is the task of predicting the next word given a sequence of words. In **masked language modeling** instead of predicting every next token, a percentage of input tokens is masked at random and only those masked tokens are predicted.

Why use masked language modeling over standard language modeling?

Bi-directional models are more powerful than uni-directional language models. But in a multi-layered model bi-directional models do not work because the lower layers leak information and allow a token to see itself in later layers.

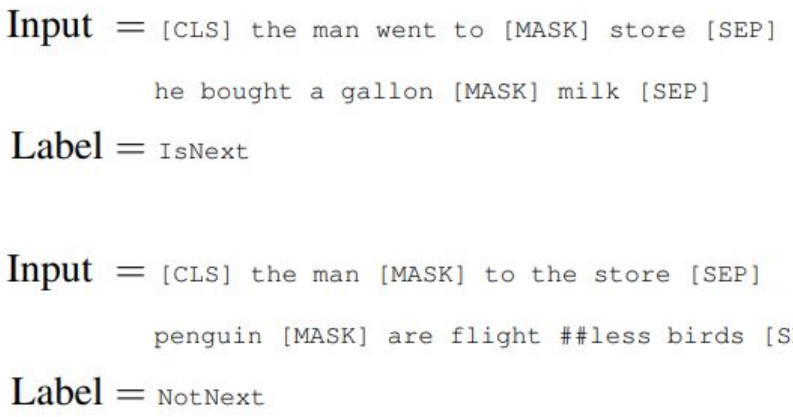
How is masked language modeling implemented in BERT?

The masked words are not always replaced with the masked token – **[MASK]** because then the masked tokens would never be seen before fine-tuning. Therefore, 15% of the tokens are chosen at random and –

- 80% of the time tokens are actually replaced with the token [MASK].
- 10% of the time tokens are replaced with a random token.
- 10% of the time tokens are left unchanged.

What is Next Sentence Prediction?

Next sentence prediction task is a binary classification task in which, given a pair of sentences, it is predicted if the second sentence is the actual next sentence of the first sentence.



Next Sentence Prediction

(Source)

This task can be easily generated from any monolingual corpus. It is helpful because many downstream tasks such as Question and Answering and Natural Language Inference require an understanding of the relationship between two sentences.

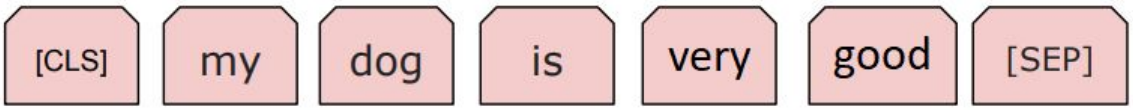
What downstream tasks can BERT be used for?

BERT can be used for a wide variety of tasks. The two pre-training objectives allow it to be used on any **single sequence** and **sequence-pair** tasks without substantial task-specific architecture modifications.

How is the input text represented before feeding to BERT?

The input representation used by BERT is able to represent a single text sentence as well as a pair of sentences (eg., [Question, Answer]) in a single sequence of tokens.

- The first token of every input sequence is the special classification token – **[CLS]**. This token is used in classification tasks as an aggregate of the entire sequence representation. It is ignored in non-classification tasks.
- For single text sentence tasks, this **[CLS]** token is followed by the WordPiece tokens and the separator token – **[SEP].**



Single Sentence Input

- For sentence pair tasks, the WordPiece tokens of the two sentences are separated by another **[SEP]** token. This input sequence also ends with the **[SEP]** token.

Sentence Pair Input

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

- A positional embedding is also added to each token to indicate its position in the sequence.

Which Tokenization strategy is used by BERT?

BERT uses WordPiece tokenization. The vocabulary is initialized with all the individual characters in the language, and then the most frequent/likely combinations of the existing words in the vocabulary are iteratively added.

How does BERT handle OOV words?

Any word that does not occur in the vocabulary is broken down into sub-words greedily. For example, if **play**, **##ing**, and **##ed** are present in the vocabulary but **playing** and **played** are OOV words then they will be broken down into **play + ##ing** and **play + ##ed** respectively. (**##** is used to represent sub-words).

What is the maximum sequence length of the input?

512 tokens.

How many layers are frozen in the fine-tuning step?

No layers are frozen during fine-tuning. All the pre-trained layers along with the task-specific parameters are trained simultaneously.

Is discriminative fine-tuning used?

No. All the parameters are tuned with the same learning rate.

What are the optimal values of the hyperparameters used in fine-tuning?

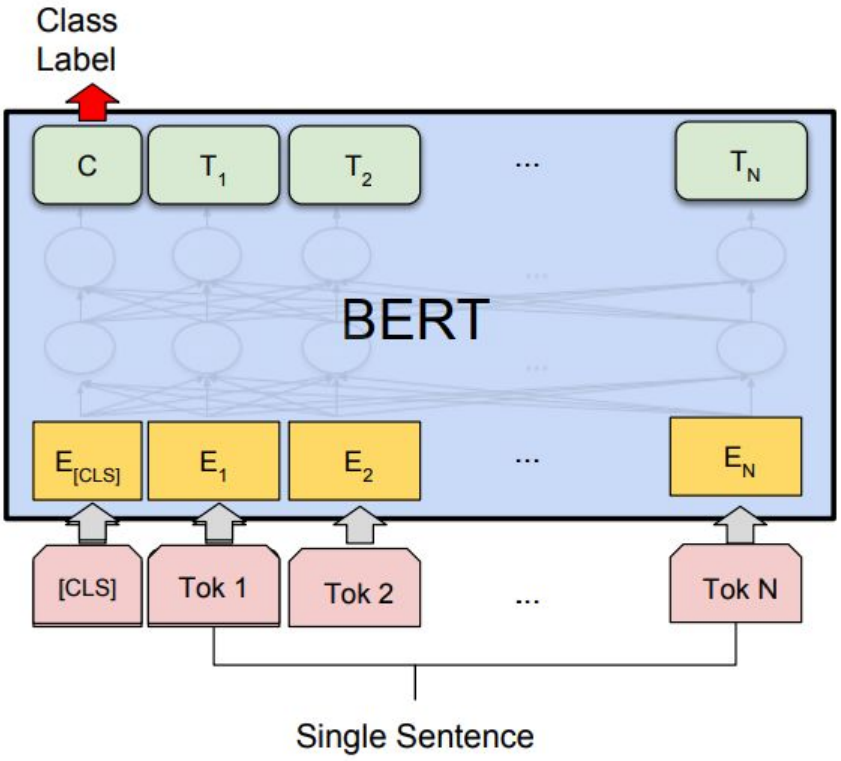
The optimal hyperparameter values are task-specific. But, the authors found that the following range of values works well across all tasks –

- **Dropout** – 0.1
- **Batch Size** – 16, 32
- **Learning Rate (Adam)** – 5e-5, 3e-5, 2e-5
- **Number of epochs** – 3, 4

The authors also observed that large datasets (> 100k labeled samples) are less sensitive to hyperparameter choice than smaller datasets.

What is the fine-tuning procedure for sequence classification tasks?

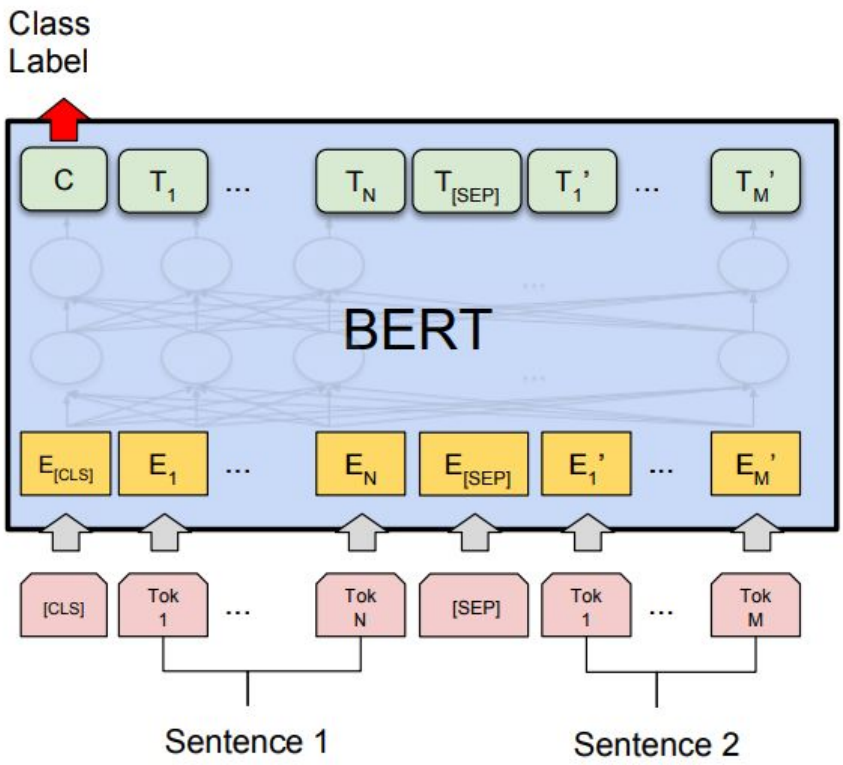
The final hidden state of the **[CLS]** token is taken as the fixed-dimensional pooled representation of the input sequence. This is fed to the classification layer. The classification layer is the only new parameter added and has a dimension of **K x H**, where **K** is the number of classifier labels and **H** is the size of the hidden state. The label probabilities are computed with a standard softmax.



Sequence Level Classification (Source)

What is the fine-tuning procedure for sentence pair classification tasks?

This procedure is exactly similar to the single sequence classification task. The only difference is in the input representation where the two sentences are concatenated together.



Sentence Pair Classification

(Source)

What is the fine-tuning procedure for question answering tasks?

Question answering is a prediction task. Given a question and a context paragraph, the model predicts a start and an end token from the paragraph that most likely answers the question.

- Input Question:

```
Where do water droplets collide with ice
crystals to form precipitation?
```

- Input Paragraph:

```
...  Precipitation forms as smaller droplets
coalesce via collision with other rain drops
or ice crystals within a cloud.  ...
```

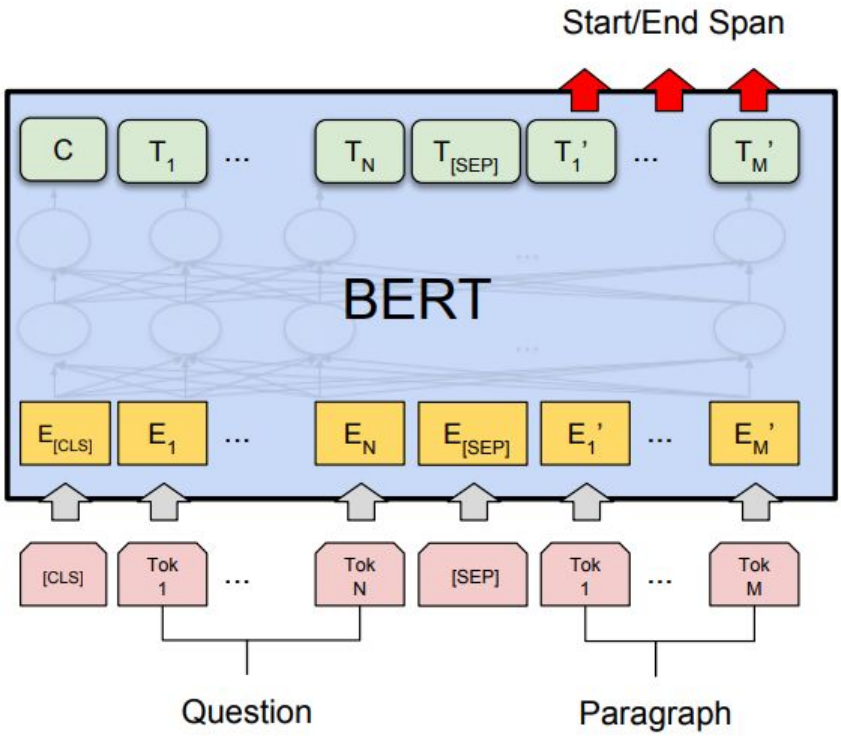
- Output Answer:

```
within a cloud
```

Question Answering Example

(Source)

Just like sentence pair tasks, the question becomes the first sentence and paragraph the second sentence in the input sequence. There are only two new parameters learned during fine-tuning a **start vector** and an **end vector** with size equal to the hidden shape size. The probability of token **i** being the start of the answer span is computed as – **softmax(S . K)**, where **S** is the start vector and **K** is the final transformer output of token **i**. The same applies to the end token.

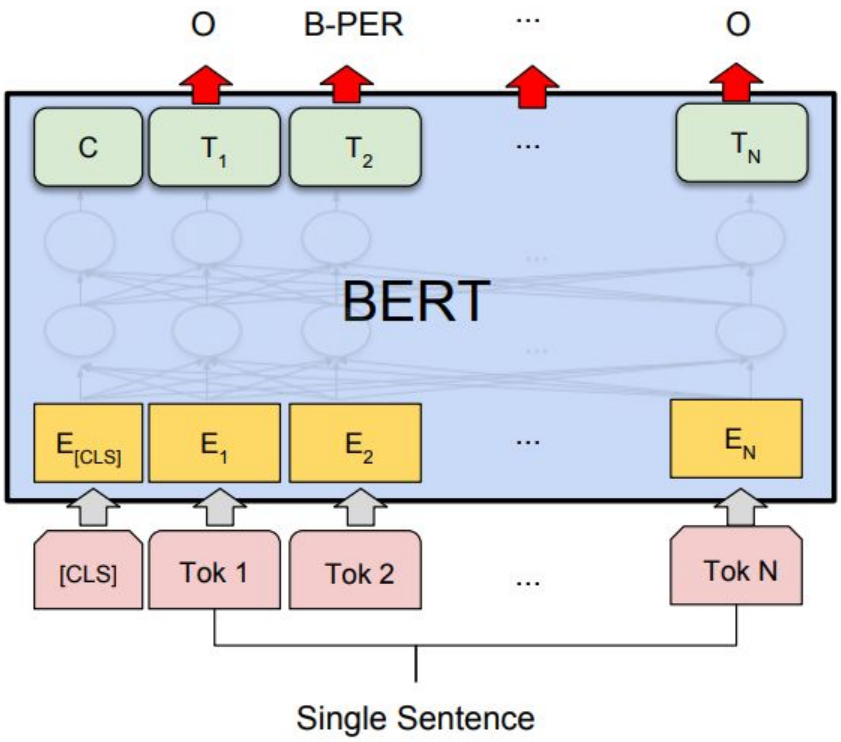


Question Answering

*(**Source**)*

What is the fine-tuning procedure for single sentence tagging tasks?

In single sentence tagging tasks such as named entity recognition, a tag must be predicted for every word in the input. The final hidden states (the transformer output) of every input token is fed to the classification layer to get a prediction for every token. Since WordPiece tokenizer breaks some words into sub-words, the prediction of only the first token of a word is considered.



*Single Sentence Tagging Task (**Source**)*

Can BERT be used with tensorflow?

Yes. The official open sourced code is in tensorflow (GitHub).

Can BERT be used with Pytorch?

Yes. Huggingface has open sourced the repository – **transformers**. It supports the op-to-op implementation of the official tensorflow code in PyTorch and many new models based on transformers.

Can BERT be used with Fastai?

As of now, fastai does not have official support for BERT yet. But, there are ways we can get around with it. This article demonstrates how BERT can be used with fastai.

Can BERT be used with Keras?

Yes. Check this out – BERT-keras.

How to use BERT as a sentence encoder?

The final hidden states (the transformer outputs) of the input tokens can be concatenated and / or pooled together to get the encoded representation of a sentence. bert-as-a-service is an open source project that provides BERT sentence embeddings optimized for production. I highly recommend this article – Serving Google BERT in Production using Tensorflow and ZeroMQ.

How does BERT perform when used as a sentence encoder with a task-specific architecture (similar to ELMO)?

BERT is effective for both fine-tuning and feature-based approaches. The authors did ablation studies on the CoNLL-2003 NER task, in which they took the output from one or more layers without fine-tuning and fed them as input to a randomly initialized two-layer 768 dimensional BiLSTM before the classification layer. The best performing model was the one that took representations from the top four hidden layers of the pre-trained transformer.

Is BERT available in languages other than english?

Yes, there is a **multilingual BERT** model available as well.

Is BERT available on domain specific pre-trained corpus?

Yes. I have come across Clinical BERT – BERT pre-trained on clinical notes corpus and sciBERT – Pre-Trained Contextualized Embeddings for Scientific Text.

How long does it take to pre-train BERT?

BERT-base was trained on 4 cloud TPUs for 4 days and BERT-large was trained on 16 TPUs for 4 days. There is a recent paper that talks about bringing down BERT pre-training time – **Large Batch Optimization for Deep Learning: Training BERT in 76 minutes**.

How long does it take to fine-tune BERT?

For all the fine-tuning tasks discussed in the paper it takes at most 1 hour on a single cloud TPU or a few hours on a GPU.

Thanks for reading. Please feel free to suggest more questions in the comment section. Your valuable suggestions are always welcome.