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

Introducing a new language representation model called BERT, which stands for Bidirectional Encoder Representations from Transformers. Unlike recent language representation models, 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).

Unlike traditional models that were trained for specific language tasks both BERT and GPT pretrain their models semi-supervised on large text datasets such as Wikipedia or BooksCorpus with over 3 billion words in total. BERT was then fine-tuned on labelled datasets for NLP tasks such as sentiment analysis, question answering or named-entity recognition and well surpassed previous SOTA-results in many well-known benchmarks.

What is great about the pre-training is that it allows the model to gain a deeper understanding of the language structure using an almost unlimited resource of data. In itself though the pre-training tasks are not very useful. Instead the purpose of both GPT and BERT is to be fine-tuned on specific language tasks.

Self-Supervised Learning Vs Semi-Supervised Learning: How They Differ

Self-supervised learning

In the case of supervised learning, the AI systems are fed with labelled data. But as we work with bigger models, it becomes difficult to label all the data. Additionally, there is just not enough labelled data for a few tasks, such as training translation systems for low-resource languages.

A self-supervised learning system aims at creating a data-efficient artificial intelligent system. It is generally referred to as extension or even improvement over unsupervised learning methods. However, as opposed to unsupervised learning, self-supervised learning does not focus on clustering and grouping.

It could even be seen as an autonomous form of supervised learning as it requires no human input in the form of data labelling. There are three significant advantages to self-supervised learning:

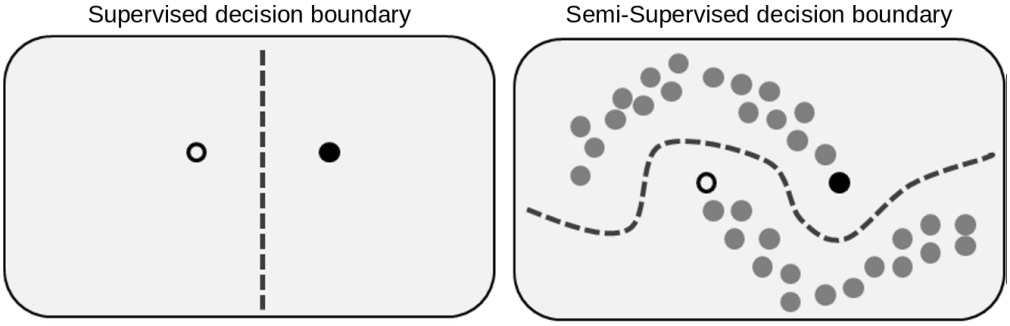
- **Scalability:** Supervised learning technique needs labelled data to predict the outcome for unknown data. However, it may need a large dataset to build models that make accurate predictions. Manual data labelling is time-consuming and often not practical. Here is where ***self-supervised learning*** helps as it automates the process even with large amounts of data.
- **Improved capabilities:** Self-supervised learning has significant applications in computer vision for performing tasks such as colourisation, 3D rotation, depth completion, and context filling. Speech recognition is another area where self-supervised learning thrives.
- **Human intervention:** Self-supervised learning automatically generates labels without human intervention.

Despite its various advantages, self-supervised learning suffers from **uncertainty.** In cases such as Google's BERT model, where variables are discrete, this technique works well. However, in the case of variables with continuous distribution (variables obtained only by measuring), this technique has failed to generate successful results.

Semi-supervised learning

Semi-supervised learning is a combination of supervised and unsupervised learning. It uses a small amount of labelled data with a larger share of unlabelled data. Semi-supervised learning technique typically involves the following steps:

- First, training the model with a small amount of labelled data (similar to what is done in supervised learning) until the model gives good results.
- Using the model with unlabelled training or pseudo label dataset to predict the output.
- Link the labels from the labelled training data with the pseudo labels and the data inputs from the labelled training data with the inputs in the unlabeled data.
- Train the model in the same way as one would in the case of the fully labelled dataset.



One popular semi-supervised learning technique is by combining clustering and classification algorithms. Clustering algorithms are unsupervised learning methods that group data based on their similarities. These algorithms help in finding the most relevant samples in the data set. The samples can then be labelled and used to train the supervised learning model for a classification task.

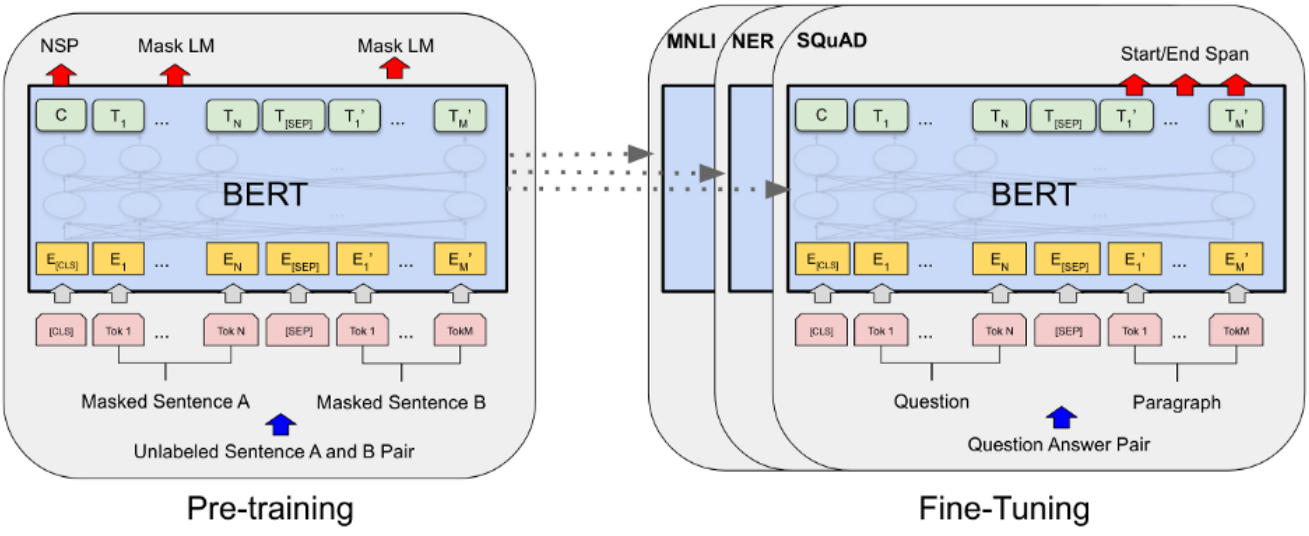
BERT General Overview and Logic

BERT leverages a fine-tuning based approach for applying pre-trained language models; i.e. a common architecture is trained for a relatively generic task, and then, it is fine-tuned on specific downstream tasks that are more or less similar to the pre-training task.
To achieve this, BET proposes **2 pre-training tasks:**

1. Masked Language Modeling (MLM)
2. Next Sentence Prediction (NSP)

and fine-tuning on downstream tasks such as:

1. Sequence Classification
2. Named Entity Recognition (NER)
3. Natural Language Inference (NLI) or Textual Entailment
4. Grounded Common Sense Inference
5. Question Answering (QnA)



To elaborate on the BERT-specific architecture, **we will compare the encoder and the decoder of the Transformer**:

- **The Transformer Encoder** is essentially a Bidirectional Self-Attentive Model, that uses all the tokens in a sequence to attend each token in that sequence

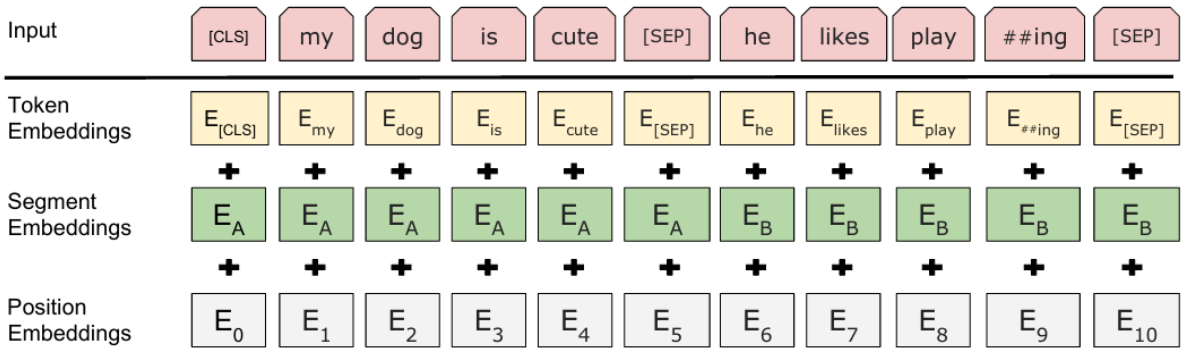
i.e. for a given word, the attention is computed using all the words in the sentence and not just the words preceding the given word in one of the left-to-right or right-to-left traversal order.

- **The Transformer Decoder**, is a Unidirectional Self-Attentive Model, that uses only the tokens preceding a given token in the sequence to attend that token

i.e. for a given word, the attention is computed using only the words preceding the given word in that sentence according to the traversal order, left-to-right or right-to left.

Therefore, the **Transformer Encoder gives BERT its Bidirectional Nature**, as it uses tokens from both the directions to attend a given token.

BERT Representations



Input/output representations used in BERT:

- **[CLS]:** This token is called as the '*classification*' token. It is used at the beginning of a sequence.

- **[SEP]:** This token indicates the separation of 2 sequences i.e. it acts as a delimiter.

- **[MASK]:** Used to indicate masked token in the MLM task.

- **Segment Embeddings** are used to indicate the sequence to which a token belongs **i.e.** if there are multiple sequences separated by a [SEP] token in the input, then along with the Positional Embeddings(from Transformers), these are added to the original Word Embeddings for the Model to identify the sequence of the token.

- The tokens fed to the BERT model are tokenized using **WordPiece embeddings**. The working of this tokenization algorithm is out of scope for this discussion, however, it basically is a tokenization technique between purely character-level encoding and complete word-level encoding to increase the coverage for most of the words in the vocabulary with an appreciable reduction in the vocabulary size (i.e. most of the out of vocabulary or OOV words are covered).

BERT Pre-Training:

As mentioned previously, BERT is trained for 2 pre-training tasks:

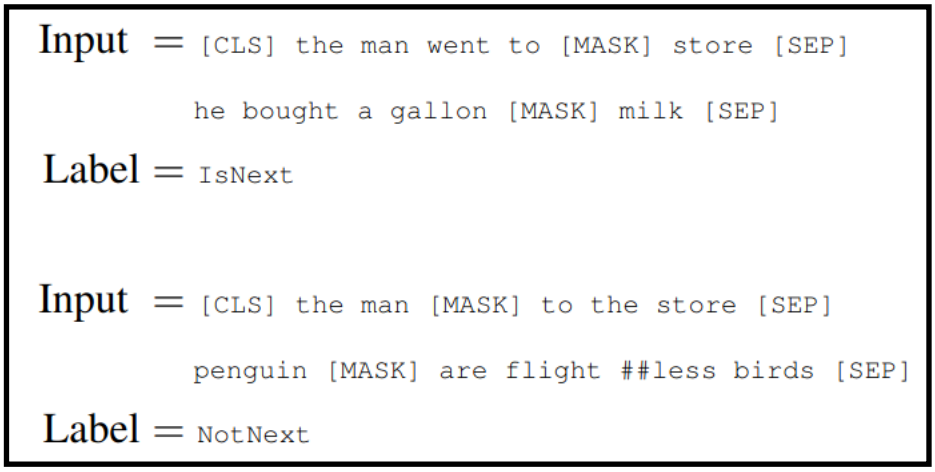
1. Masked Language Model (MLM)

In this task, 15% of the tokens from each sequence are randomly masked (replaced with the token **[MASK]**). The model is trained to predict these tokens using all the other tokens of the sequence.
However, the fine-tuning task is in no way going to see the [MASK] token in its input. So, for the model to adapt these cases, 80% of the time, the 15% tokens are masked; 10% of the time, 15% tokens are replaced with random tokens; and 10% of the time, they are kept as it is i.e. untouched.

- 80% of the time: Replace the word with the `[MASK]` token, e.g., `my dog is hairy` → `my dog is [MASK]`
- 10% of the time: Replace the word with a random word, e.g., `my dog is hairy` → `my dog is apple`
- 10% of the time: Keep the word unchanged, e.g., `my dog is hairy` → `my dog is hairy`. The purpose of this is to bias the representation towards the actual observed word.

2. Next Sentence Prediction (NSP)

This task is somewhat similar to the Textual Entailment task. There are two input sequences (separated using [SEP] token, and **Segment Embeddings** are used). It is a binary classification task involving prediction to tell if the second sentence succeeds the first sentence in the corpus.
For this, 50% of the time, the next sentence is correctly used as the next sentence, and 50% of the time, a random sentence is taken from the corpus for training. This ensures that the model adapts to training on multiple sequences (for tasks like question answering and natural language inference).



Fine-Tuning BERT

BERT achieved the state of the art on 11 *GLUE* (General Language Understanding Evaluation) benchmark tasks. We will discuss modeling on a few of them in this section:

- **Sequence Classification:** The pre-trained model is trained on a supervised dataset to predict the class of a given sequence. Since the **output of the BERT (Transformer Encoder) model is the hidden state for all the tokens in the sequence**, the output needs to be pooled to obtain only one label. The Classification token ([CLS] token) is used here. The output of this token is considered as the classifier pooled output and it is further put into a fully-connected classification layer for obtaining the labeled output.

- **Named Entity Recognition (NER):** The hidden state outputs are directly put into a classifier layer with the number of tags as the output units for each of the token. Then these logits are used to obtain the predicted class of each token using *argmax.*

- **Natural Language Inference (NLI) or Textual Entailment:** We train BERT the same as in the **NSP** task, with both the sentences i.e. the **text** and the **hypothesis** separated using [SEP] token, and are identified using the

Segment Embeddings. The [CLS] token is used to obtain the classification result as explained in the Sequence Classification part.

- **Grounded Common Sense Inference:** Given a sentence, the task is to choose the most plausible continuation among four choices. We take 4 input sequences, each containing the original sentence and the corresponding choice concatenated to it. Here too, we use the [CLS] token state and feed it to a fully-connected layer to obtain the scores for each of the choices which are then normalized using a softmax layer.

- **Question Answering:** A paragraph is used as one sequence and the question is used as another. **There are 2 cases in this task:**
- In the first case, the answer is expected to be found within the paragraph. Here, we intend to find the **Start** and the **End** token of the answer from the paragraph. For this, we take the dot product of **each token *T_i*** and the **start token *S*** to obtain the **probability of the token *i* to be the start of the answer**. Similarly, we obtain the probability of the end token *j.* The score of a candidate span from position *i* to position *j* is defined as ***S.T_i + E.T_j***, and the maximum scoring span where $j \geq i$ is used as a prediction.

- In the second case, **we consider the possibility that there may be no short answer to the question present in the paragraph**, which is a more realistic case. For this, we consider the probability scores for the **start and end of the answer at the [CLS] token itself**. We call this as *s_null.* We compare this *s_null* with the max score obtained at the best candidate span (i.e. the best score for the first case). If this obtained score is greater than *s_null* by a sufficient threshold τ, then we use the candidate best score as the answer. The threshold τ can be tuned to maximize the dev set ***F1-score***.