UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft

Faculty for Computer Science, Electrical Engineering and Mathematics
Department of Computer Science
Research Group Data Science

Seminar Report

Submitted to the Data Science Research Group
in Partial Fullfilment of the Requirements for the Degree of

Master of Science

BERT Model for Language Representation

by
Sai Nikhil Menon

Seminar Supervisor:
Dr. Mohamed Sherif

Paderborn, February 5, 2021

Abstract. Last few years saw a rapid development in the field of Machine Learning and Natural Language Processing. With many new sophisticated learning models and architectures coming into picture, the systems are becoming more efficient in terms of accuracy and computational time. The development of the Transformers neural network architecture along with effective use of Transfer Learning is one reason behind this growth. In this report, we look at the BERT language model developed by Google which uses the Transformer architecture for capturing bidirectional contextual information and language understanding. The BERT model uses the Multi-headed Attention mechanism to evaluate the context of a token(word) based on the tokens that occur before and after it, by evaluating the effect these neighbouring tokens have on the token under consideration, for a given input sentence. The BERT model is first pre-trained on a large corpus of data to build the language representation and then fine tuned based on the downstream task, by adjusting the model weights accordingly. Experimental results show that the BERT model achieves state-of-the-art results in 11 language processing tasks.

Contents

0.1 Introduction

In the last few years, there has been a massive growth in the field of Natural Language Processing(NLP) and has been a hot topic of research in the data science fraternity for a while. Many multi-national companies in different domains are now utilising Natural Language Processing to develop products and user services that are smart and responsive, as well as to gain powerful insights to help them make business decisions. A large portion of this development can be attributed to the development in the field of Neural Networks such as the development of the Transformer architecture and growing use of concepts like Transfer Learning, which forms the core of the BERT model. Transfer Learning approach allows the developer to utilise pre-trained models for performing specific tasks on a different dataset based on their need. This helps in building models with a good overall performance in a much shorter amount time and using much lesser amount of data. In general, there are two approaches for realising tasks using Transfer Learning: *Fine tuning* and *Feature Based*. In the Fine tuning approach, a pre-trained model is tuned to perform the required task by updating the existing model parameters based on the dataset and the task in hand. The BERT model works on this principle, and is discussed in further detail later. Feature based approach on the other hand, utilises the pre-trained parameters as addition features along with the new features that are required for realising the specific task. ELMo [1] is one such model that works on the Feature based approach.

BERT has been undoubtedly one of the biggest game changer in the field of Natural Language Processing. The BERT model algorithm was developed by Google in 2018 to develop language models for performing a variety of language processing tasks. BERT works on the principle of bidirectional context analysis, meaning that it understands the context of a given word based on the words that occur before it as well as based on the words that occur after it. The contexts are captured using a mechanism knows as the Attention mechanism, which is derived from the Transformers architecture [2], a neural network based architecture that uses encoder and decoder stacks in different layers, and forms the core of the BERT model. The BERT model is trained in two phases: *Pre-training phase* and *Fine Tuning phase* . Pre-training BERT is a computationally intensive task, however there are various ready-to-use pre-trained BERT models available that can be directly fine-tuned for production purpose, thereby utilising the power of Transfer Learning. The ease of use combined with accurate results makes BERT a very powerful language model, and is being used for performing various tasks such as Question-Answering and Sentiment Analysis.
In this report, the presentation of ideas and results are based on the results presented in the original paper.[3]

0.2 Related Work

The BERT language model is undoubtedly a breakthrough in the field of Natural Language Processing. In order to understand the working of the BERT model, it is important to understand some of the predecessor systems of BERT, which pioneered the different ideas used while developing BERT. In this section, we look at a few such predecessor systems which laid the foundation for the BERT language model.

ELMo: Embeddings from Language Models The ELMo model [1] was one of the first models to introduce the concept of *contextualised embeddings*. These embeddings were based on the simple fact that a given word could mean different things in different contexts[4]. The embedding associated with a word is dependent on all the other words in the sentence as well.

The traditional methods for generating word embeddings [5] such as Word2Vec [6] and GloVe [7] were not capable of generating embeddings based on context. The ELMo model architecture utilises two layered bidirectional LSTM network for developing the language model. Each layer has a forward pass and backward pass. Forward pass is used to understand the context of a word based on the words that occur before it in the sequence. The backward pass on the other is used to incorporate the context aspect from the words that occur after the current word under consideration in the sentence. The two passes are then concatenated to build the overall model. The BERT model utilised these ideas of contextual embeddings to capture the context bidirectionally, and this was aided due the development of the Transformer architecture.

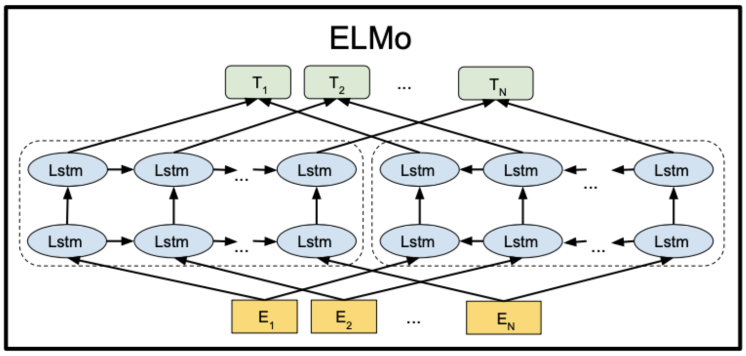


Figure 1: ELMo Language model [1]

Transformer Model Transformer model [2] is basically a neural network based architecture containing stacks of encoders and decoders. It was initially developed in order to tackle the task of Machine Translation and has since been used in different forms for different tasks. What makes the transformer model so attractive to the NLP practitioners is the concept of "Attention". The attention mechanism allows the context of a word to be generated bidirectionally. However, unlike the traditional LSTM based systems, the encoder-decoder architecture allowed multiple token to flow in simultaneously making it much faster in terms of computation.
The BERT model utilises the encoder part of the transformer to build the model, as we will see in later sections. Various other language models such as the different versions of the GPT model are based on the decoder aspect of the transformer architecture.

GPT: Generative Pre-trained Transformer The GPT language model [8] is also based on the Transformer architecture, however GPT utilises the decoder aspect for building the model. The decoder architecture is designed in such a way that it can accept only one input token at a time, unlike the encoder part and hence the GPT model like the traditional LSTM based systems also works sequentially uni-directionally. The GPT model is *auto-regressive* in nature, in the sense that each token that is predicted by looking at its preceding words is added as in input to predict the next word after.
GPT model was one of the first language model to utilise the transformer architecture and laid a stepping stone for BERT, while utilising the encoder stack instead of the decoder stack to build the model.

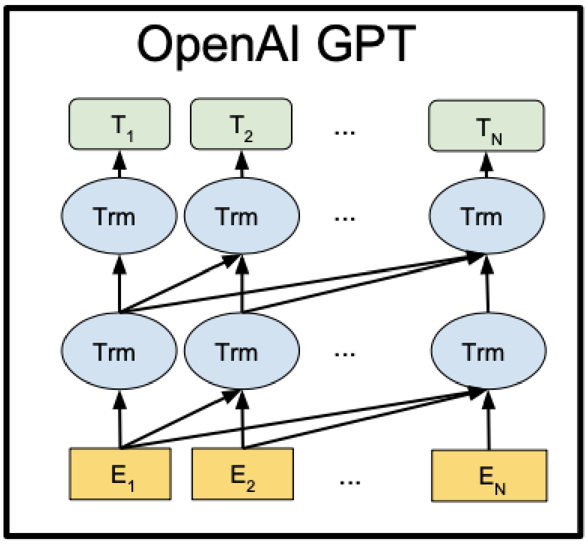


Figure 2: GPT Language model [8]

0.3 The BERT model

0.3.1 Architecture of BERT model

The architecture of the BERT model has the Transformer neural network architecture in its core. This enables BERT to leverage the Attention model[2] for understanding the context of the words with great efficiency. The Transformer in its vanilla form is a neural network model consisting of layers of encoder and decoders. The encoders are responsible for accepting input sentences and encoding them by using the multi-headed attention mechanism. This allows the model to understand the language from a contextual viewpoint, thereby building a powerful language representation. Decoder on the other hand, takes the encoder output as its input and trains itself to perform the required task such as machine translation. Transformers have been experimentally proven to outperform its counterparts such as Recurring Neural Networks and LSTMs in performing language processing tasks and Seq2Seq tasks in particular. This is mainly because of the fact that Transformer is capable of processing all the words in a sentence simultaneously, in one go. RNNs and LSTMs on the other hand follow a more sequential approach.

Since with BERT, the authors aim to build a language model that can be later tuned for performing specific tasks, the BERT architecture only consists of the encoder stack. The encoder stack, as the name suggests is layers of encoders stacked one above the other. The BERT model comes in two variants, *BERT-Base* and *BASE-Large*. The BERT-Base model consists of 12 encoder layers and is trained using 110 million parameters whereas the BERT-Large consists of 24 encoder layers and is trained using 340 million parameters. Figure 3 shows the high level architecture of the two BERT variants.

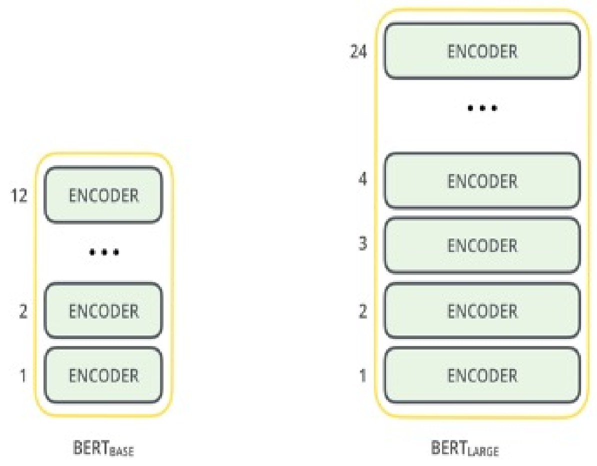


Figure 3: BERT Variants: BERT-Base and BERT-Large [1]

Each encoder layer consists of multiple Attention heads and a feed forward network. The attention heads acts on every word and creates an attention embedding for each word based on the words that occur before and after it. This captures the context related information for every word in the input sentence. The feed forward network is a simple linear layer consisting of weights to process the attention embedded words, thereby aiding the learning process. Figure 4 shows the two components of an encoder layer. We will look at the Attention model and the Feed forward network in further detail in next sections.

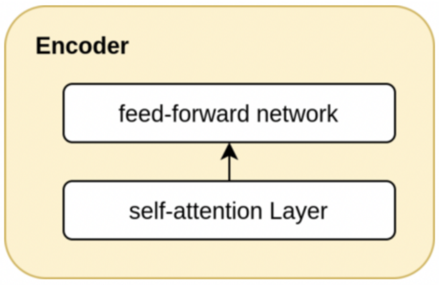


Figure 4: BERT Encoder [2]

An encoder layer in the BERT model, expects the input data point to be a two-dimensional array of shape $S*D$, where S is the length of the input sentence and D is the number of dimensions in the word embedding of each word in the input sentence, generated using the WordPiece embeddings. Since the BERT model, consists of a stack of encoders, one on top of another, the output of a particular encoder layer must also be of the shape $S*D$, since this data is given as the input to the encoder layer above it.

Multi-Head Self Attention layer

The input data point first goes into the Self Attention layer [9] [2]. The Self Attention layer of the encoder is where most of the contextual learning occurs. Most of the traditional methods that were employed for performing Natural Language Processing tasks, use non-contextual

[1]http://jalammar.github.io/illustrated-bert
[2]https://lionbridge.ai/articles/what-are-transformer-models-in-machine-learning

word embeddings during the feature engineering process. While this approach has a plethora of advantages and use cases, such an approach does not work well if the input data points are sentences, where the contextual information of a word is important and can have a negative impact on the accuracy if not handled correctly.
Consider the following sentences:

Tom reached the bank after crossing the road

Tom reached the bank after crossing the river

In the above sentences, the context of the word *bank* depends on whether the last word of the sentence is *road* or *river*. The traditional word embedding algorithms will fail to understand this context since it looks at only that particular word at a given instant, and the word embedding is generated independent of the words around it. The Attention layer on the other hand, looks at the neighbouring words in both directions of the word under consideration. The attention layer will be able to understand the meaning of the word *bank* by looking at its neighbouring words and will therefore generate two different embeddings for the word *bank* depending on the context in which it is used.

Figure 5 gives a very high level illustration of how the embedding of a word is generated in the Attention layer. As seen in the figure, the word representation of *bank* is calculated by looking at its neighbouring words. The word representation of *bank*, generated by the Attention layer will therefore be context-aware. The BERT model will therefore be able to tell that the *bank* here refers to the institution called bank and not the river bank.

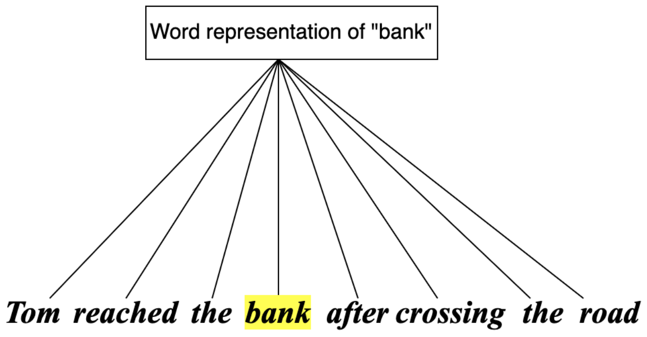

Word representation of "bank"
Tom reached the bank after crossing the road

Figure 5: Contextual Word Representation using Attention

As mentioned before, the input data point to an encoder layer in BERT is a matrix of shape *S*D*, where *S* is the length of the input sentence and *D* is the number of dimensions in the word embedding of each word in the input sentence, generated using the WordPiece embeddings. To put things into perspective, what this means is that an input sentence *Tom reached the bank after crossing the road* will be represented by a matrix of shape *8 * 512*, where *S* has been set to 8, since the sentence contains 8 words and *D* has been set to 512 by default while generating the WordPiece embeddings. This can however be tweaked based on the requirement.

1. The first step in the Self Attention layer is to generate *Query(Q)*, *Key(K)* and *Value(V)* vectors for each input word vector (embedding of a word). This is done by multiplying

the input vector with three different weight matrices, W_k, W_q and W_v for generating Q, K and V vectors respectively. These weight matrices are adjusted iteratively during the training process and have a dimension of $D*d$, where d is set to 64 by default, but can be changed based on need and architectural design requirement.

2. Once the Q, K and V vectors are generated for each word vector, the *attention score* is calculated for each word against every other word in the input sentence sequence. This is done by finding the dot product of the Query vector, Q of the word under consideration with the Key Vector, K of every other word in the input sentence (all neighbouring words). The attention score essentially signifies the similarity between two words.

3. The next step is to normalise these attention scores by dividing them by $\sqrt{d}$ and applying the *Softmax* operation on each of the score. As a result, the scores associated with a particular word under consideration against every other word in the input will be in the range of 0 to 1.

4. The normalised score obtained against each word in the input sentence is multiplied with the corresponding Value vector, V associated with that word. This is done in order to put a higher focus on neighbouring words that have a high attention score against the word under consideration and to suppress the effect of neighbouring words that have a low attention score, since words with a low attention score will probably not have a great impact on the word under consideration

5. The resulting vectors are summed up and multiplied with a weight matrix W_o that is trained during the training process, thus giving us the "Attention" associated with the word under consideration.

The sequence of events that occur in the Attention layer are illustrated in Figure 6. The illustration shows the attention mechanism at a single word level.
A two word input *Learning Data* has been taken as the input data point for simplicity and ease of understanding and the word *Learning* has been taken as the reference word for which the attention mechanism is illustrated.

1. The input word vector(embedding) for *Learning* is represented using vector X_1 and that of *Data* is represented as X_2.
   Their *Query*, *Key* and *Value* vectors are generated by multiplying the input vector with weight matrices W_k, W_q and W_v for generating Q_1, K_1 and V_1 vectors for the word *Learning* and Q_2, K_2 and V_2 vectors for the word *Data* respectively.

2. For the word *Learning* taken as the reference word, attention scores are calculated against every word in the input sentence, i.e against the word *Learning* itself and against the word *Data* by finding the dot product of Q_1 and K_1 and Q_1 and K_1 respectively.

3. The attention scores obtained for are then normalised. These attention scores essentially signify the similarity between the word under consideration. *Learning* and every other word in the input sentence.

4. Once the normalised scores are generated, the words with lesser focus are suppressed. The normalised scores are therefore multiplied with their respective value vectors, V_1 and V_2 in this case. Vectors Z_1 and Z_2 are generated. These are basically the attention embedded vectors for the word *Learning.* against the other words in the input.

5. The final output embedding Z for the word *Learning*, with the "Attention" aspect inculcated in it is obtained by finding the sum of vectors generated in previous step and multiplying the sum with a weight matrix W_o trained during the training process.

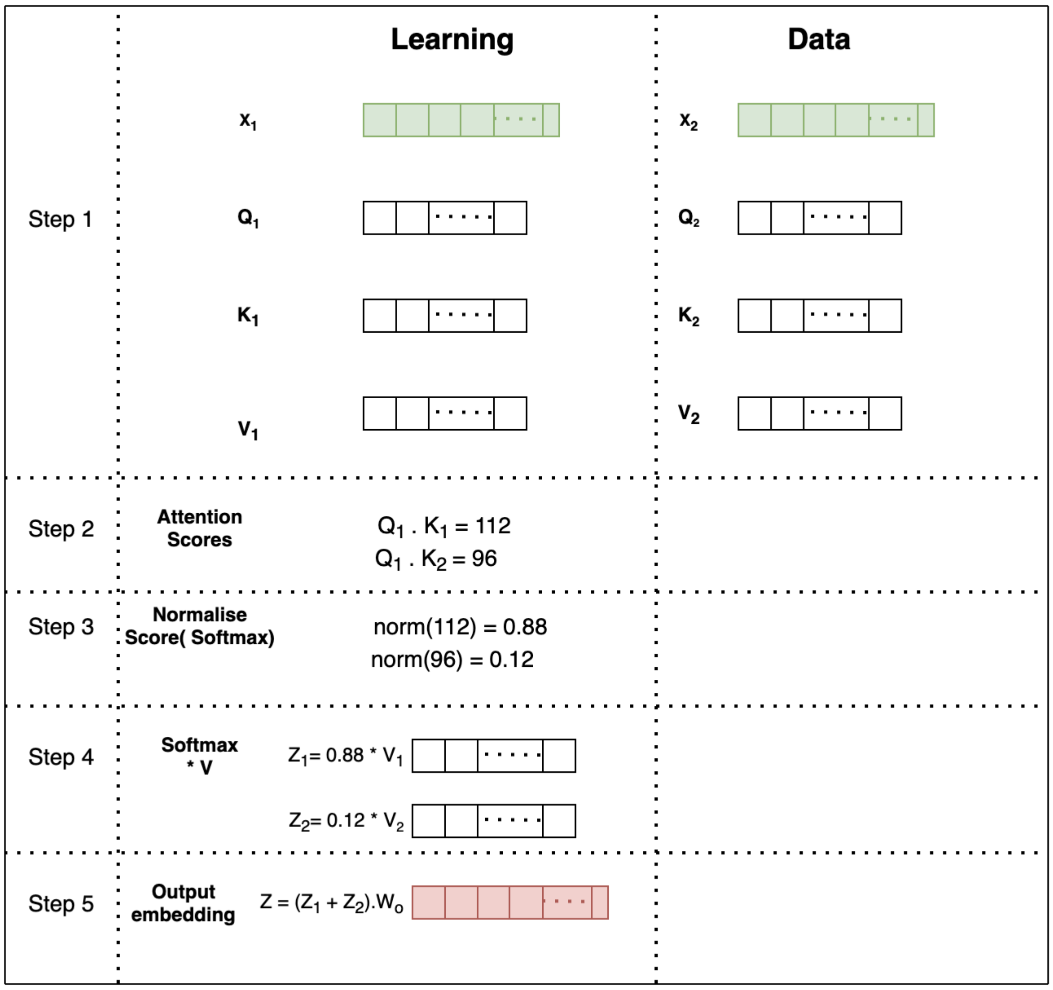


Figure 6: Illustration of Attention Mechanism at Single Word Level

The multi-head attention mechanism used in the BERT model is an extension of the vanilla attention mechanism, however the multi-head attention has multiple attention layers stacked parallel to one another within a each encoder. This allows the model to learn from different perspectives or "representation subspaces"(cite). By default, the BERT-Base and BERT-Large model are trained with 12 and 16 different attention layers respectively. This means that each encoder generates 12 different *Query*, *Key* and *Value* weight matrices in BERT-Base and 24 different matrices in BERT-Large. Hoever, the number of attention layers is a design choice that has to be made before the training phase. The multi-head attention mechanism is illustrated in the Figure 7. As seen in the figure, the input matrix X is fed into 8 different attention layers in parallel. The output from the 8 different attention layers, representing different "subspaces" are then concatenated and multiplied with the weight matrix W_o to generate the final attention embedded output matrix of shape $S * D$.

Another important point to note here is that in the practical implementation of BERT, all the calculations are done using matrix operations, i.e, in a vectorized form. This allows the data to be processed faster, since the modern day processors are equipped to handle matrix operations with much more ease than iterative loop statements.

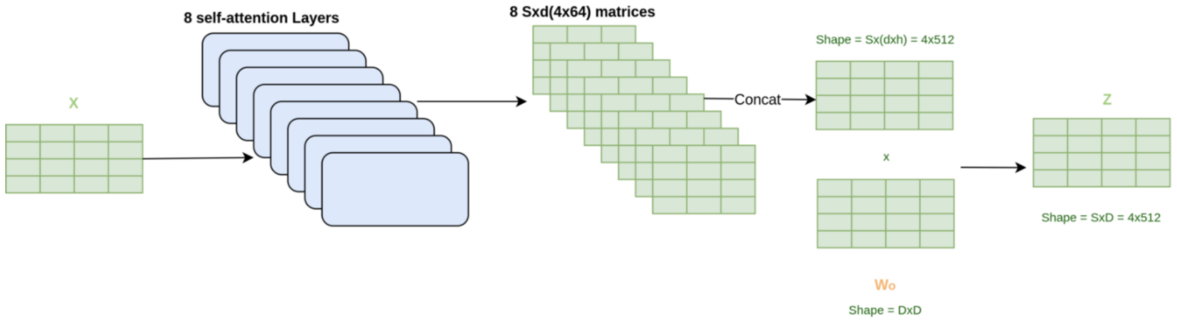


Figure 7: Multi-Head Attention Mechanism with Eight Attention Layers source[3]

Feed-Forward Network

The attention embedded output from the Multi-Head Attention layer is given as input to a feed-forward neural network. This network is *positional* in nature, in the sense that it is not one single network to which the embeddings associated with all the words in the input sentence are sent. Instead, it is a collection of many feed-forward networks, each of which is identical in design and receives the input embedding vector associated with a single word or "position" in the input sentence.

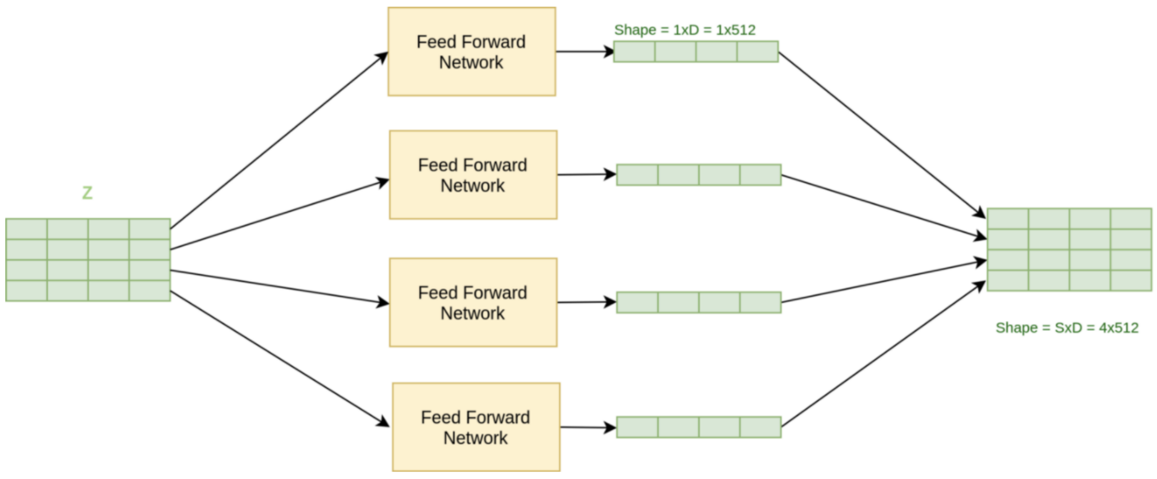


Figure 8: Feed-Forward Network [4]

Figure 8 shows the feed forward network in BERT model. As seen in the figure, each attention embedded word vector or "position" vector associated with a word in the input sentence is given as input to different feed-forward networks. A feed-forward network is essentially a simple

[3]https://lionbridge.ai/articles/what-are-transformer-models-in-machine-learning/
[4]https://lionbridge.ai/articles/what-are-transformer-models-in-machine-learning/

neural network that propagates data in one direction only. The design of these individual feed-forward networks, such as deciding the number of hidden layers and other technical aspects are implementation and use-case dependent. Each feed-forward network processes the input vector associated with exactly one "position", and the output layer of the network returns a vector of the shape as the input, i.e, *1 * D*. The output from all the different feed-forward network are then combined to form a data matrix of shape $S * D$, which is then given as input to the encoder layer above.

0.3.2 Implementation of BERT model

The implementation of the BERT model is a two step process : *Pre-training* and *Fine-tuning*. However, before going into the process involved in implementing BERT, it is important to know what the input data to the BERT model looks like.

Input Representations in BERT

One of the biggest advantages of BERT is the fact that multiple tokens can be processed simultaneously and this is due to the fact that the BERT model has an underlying Transformer architecture. Thus, the input to BERT model is a sequence of words. However, learning algorithms cannot understand textual data. The textual data has to be converted to numerical data for the algorithm to work. The input embedding to the BERT model contains three parts: *Token Embedding, Segment Embedding and Positional Embedding* [3]

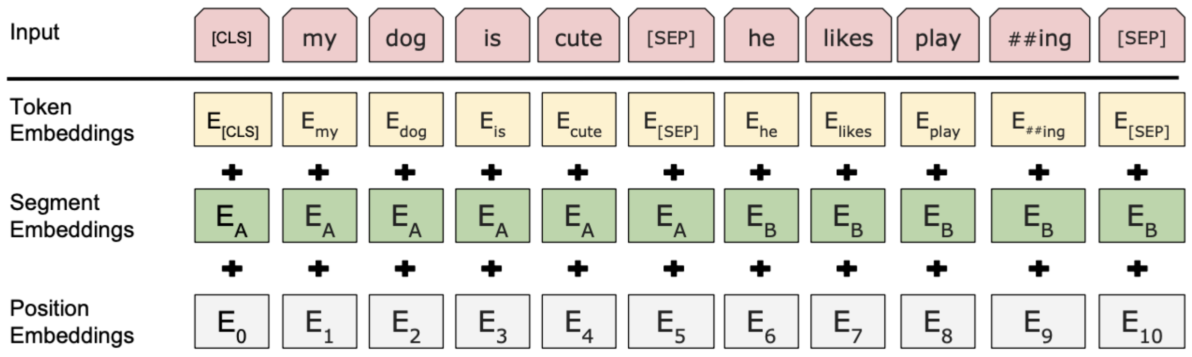

Input
[CLS]
my
dog
is
cute
[SEP]
he
likes
play
##ing
[SEP]
Token Embeddings
$E_{[CLS]}$ E_{my} E_{dog} E_{is} E_{cute} $E_{[SEP]}$ E_{he} E_{likes} E_{play} $E_{\#\#ing}$ $E_{[SEP]}$
Segment Embeddings
E_A E_A E_A E_A E_A E_A E_B E_B E_B E_B E_B
Position Embeddings
E_0 E_1 E_2 E_3 E_4 E_5 E_6 E_7 E_8 E_9 E_{10}

Figure 9: Input Representation in BERT [3]

Token Embedding The first step in generating input for the BERT model is to *tokenize* the input sequence. *Tokenization* is the process of breaking down a sequence of words into individual words or *tokens*. Each of these tokens are then converted to their respective vector representation based on the pre-defined *WordPiece embeddings*. The vector representation of a token is known as a *Token Embedding*. The token embedding associated with each word is a vector of dimension 768.
BERT architecture is designed to accept two sequences of words as input as seen in Figure 9. However, the architecture of BERT is flexible enough to accept a single sequence of words as input as well., depending on the kind of downstream task in hand. In Figure 9, two sequences *My dog is cute* and *He likes playing* are given as input. The words of the two input sequences are tokenized and converted to their respective token embeddings, such as E_{my}, E_{dog} and so on. During the tokenization process, two special tokens - [CLS] and [SEP] are added to the input. The purpose of the [SEP] token is to indicate the separation between two sequences given as

input or to indicate the end of the input. The [CLS] token is a special token that is used in classification tasks, by averaging the representation of the entire sequence in the final output layer of BERT. WordPiece embeddings contain the vector representation for these special tokens as well and hence their token embedding can be generated just like any other token.

Segment Embedding Once the token embeddings are generated for each token in the input, the next step is to determine the *Segment Embedding* of each token. Segment embeddings are basically used to incorporate information about which *segment* of the input, a particular token belongs to.
In Figure 9, all the tokens belonging to the first sequence *My dog is cute* have the segment embedding represented by E_A, whereas all the tokens belonging the second sequence *He likes playing* have the segment embedding represented by E_B. Here, A and B represent the two input sequences respectively. The segment embeddings associated with the two input segments are vectors of dimension 768. The sentence embedding of the special token [CLS] is taken as E_A, always, while that of [SEP] is taken as E_A when used as a separator between the two input segments, and is taken as E_B when used at the end of the input sequence. To summarise, these embeddings embed the information regarding which segment of the given input a particular token belongs to.

Position Embedding The third step in generating the input representation for BERT is to determine the *Position Embedding* of each token. These embeddings are used to incorporate the positional information of a token in the given input. This is necessary, since the BERT model works on the principle of Attention and often the same word used in different positions of an input sequence may have to be treated differently. Consider the following example:

The robber robbed the bank and went to the bank of the river Rhine

In the above sentence, the word *bank* at position 5 in Yellow and at position 10 in Blue have different meanings and adding their positional information will help BERT in making this distinction better. In Figure 9, the positional embeddings for each token in the input sequence is represented as E_0, E_1 and so on, depending on the position of the token. BERT model can accept input sequences containing upto 512 tokens. The embedding at each position is a vector of length 768.

Once the Token, Segment and Position Embeddings are generated for all the tokens in the input sequence, they are added to generate the final input representation that can be given as input to the BERT model.

Pre-training the BERT model

The BERT model is pre-trained on a large corpus of non-annotated data. The main idea here is to develop a model that is capable of language modelling. In other words, the model should be able to predict the probability of occurrence of a word in a sentence, given its neighbouring words by understanding the rules and linguistics behind the language. In order to cater to this requirement, two unsupervised tasks are performed: *Masked Language Modelling* and *Next Sentence Prediction.*

Masked Language Modelling As the name suggests, this task involved masking out certain words randomly from the input sequence before giving it as input to the BERT model for pre-training. The BERT model is then trained to predict the masked out word, analogous to a fill in the blank problem. The following example shows the masking of the input sequence for a basic sentence in English language, where the token *quick* is masked by replacing it with a [MASK]. This masked sequence is given as input to the BERT model for pre-training.

The quick brown fox →The [MASK] brown fox

This masked sequence in the vectorized form is given as input to the BERT model, and the task of the model is the predict what the masked word is. Figure 10 illustrated this process in pre-training of BERT.

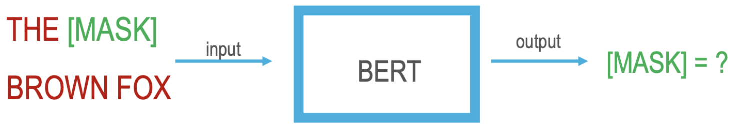


Figure 10: Masked Language Modelling task during BERT pre-training

In general, approximately 15% of the word in a given sequence are masked out randomly. However, this poses a problem since during the *Fine Tuning* task, the the expected input for specific downstream tasks does not contain masked words and this might cause the model to under-perform during the fine tuning phase. In order to control the damage and prevent performance loss, the following scheme has been devised [3]:
Out of 15% words selected for masking in a given sequence

1. 80% of them are masked with the [MASK] token.
2. 10% of the them are replaced with some random token
3. 10% of the them are replaced with the actual true token. This is done to add a little bit of bias to the model and help the model learn better.

Next Sentence Prediction The second unsupervised task performed during BERT pre-training is called Next Sentence Prediction. Next Sentence Prediction task involves passing two sequences to the BERT model. The BERT model is trained to predict the relationship between the two sentences, i.e, whether the second sentence follows the first sentence or not.

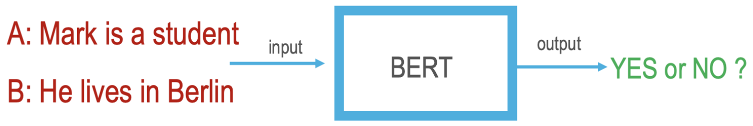


Figure 11: Next Sentence Prediction task during BERT pre-training

Figure 11 illustrated the Next Sentence Prediction task. As seen in the example, the BERT model takes two sequences *Mark is a student* and *He lives in Berlin* as input. The BERT model is trained to predict whether the second sentence follows the first sentence or not. This task can be viewed as a binary classification task, where the model outputs a *Yes* or a *No.*, depending on whether the sentences satisfy the relationship or not. During the pre-training process, half of the training examples are chosen such that the two sentences are actually next to one another and the other half are chosen such that they don't occur next to one another. Training for Next Sentence Prediction makes BERT more intelligent in performing downstream tasks such as Question-Answering.

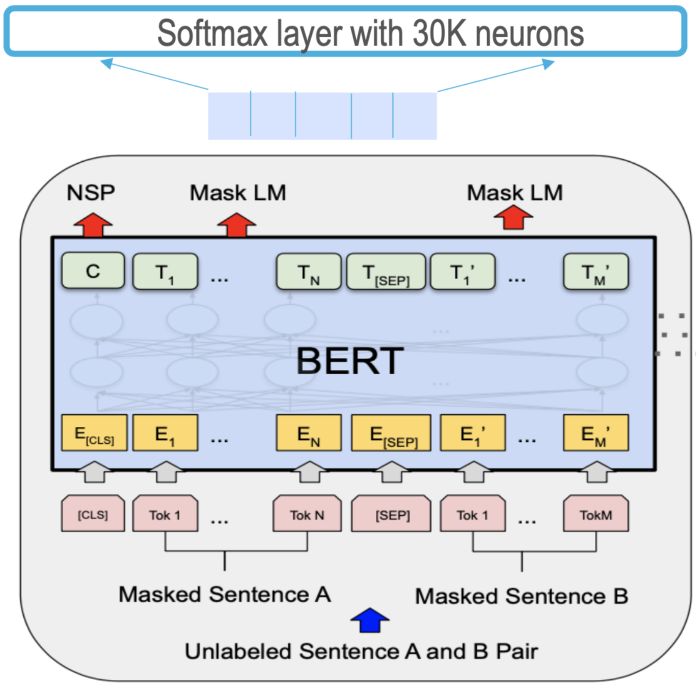


Figure 12: Pre-training the BERT model

What makes pre-training process of BERT all the more interesting is the fact that the two tasks, i.e, Masked Language Modelling and Next Sentence Prediction are performed simultaneously. The inputs are therefore two sequences of tokens with some of the tokens being randomly masked. Figure 12 illustrates the pre-training process in BERT. As shown in the figure, the input to the model are two masked sentences A and B, to enable the two tasks to be carried out in parallel. On the input end, the tokens of two masked sentences are converted to their input representation as discussed in Section 0.3.2, represented as $E_{[CLS]}$, E_1, E_2,..., $E_{M'}$. On the output front, the token C is generated from the the special input $E_{[CLS]}$ token and is used to extract the output of the Next Sentence Prediction task. For every input token that is given to BERT, a corresponding output vector is generated at the output end, represented as T_1, T_2,..., $T_{M'}$. The output vectors corresponding to only the masked tokens are then thrown to a *softmax* layer with approximately $30,000$ neurons(size of BERT vocabulary). The Softmax layer gives a probability distribution for each word in the BERT vocabulary and the word with the highest probability is predicted to be the value of the masked word.

Fine tuning the BERT model

Once the pre-trained BERT model is obtained, it can be utilised for performing specific downstream tasks such as Question-Answering, Machine Translation and Sentiment Analysis. This allows us to leverage the concept of *Transfer Learning*, thereby saving both time and computational effort. The biggest advantage of BERT lies in the fact the architecture leveraged for pre-training can be used for fine tuning as well by making a few modifications. In general, during the fine tuning tasks, the inputs and the outputs are modified to cater to the needs of the specific task in hand. The model parameters that were learnt during the pre-training phase are used as a starting point at the beginning of the fine tuning task and are simply adjusted during the training process based on the task in hand.The output of any fine tuning task is captured by adding an additional layer and sending the encoder output token representations to this additional layer, which then generates the output based on the task and input.

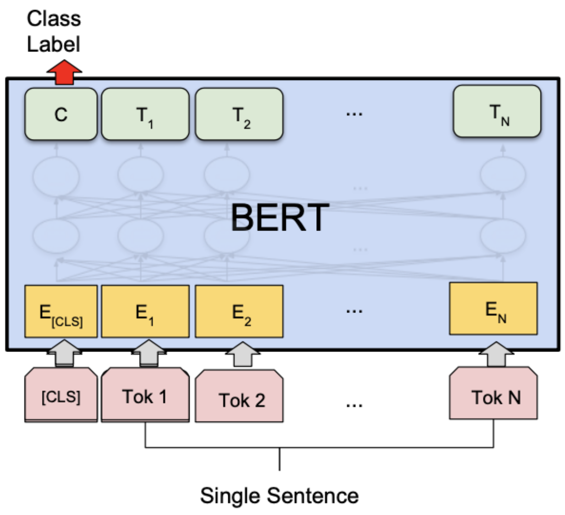
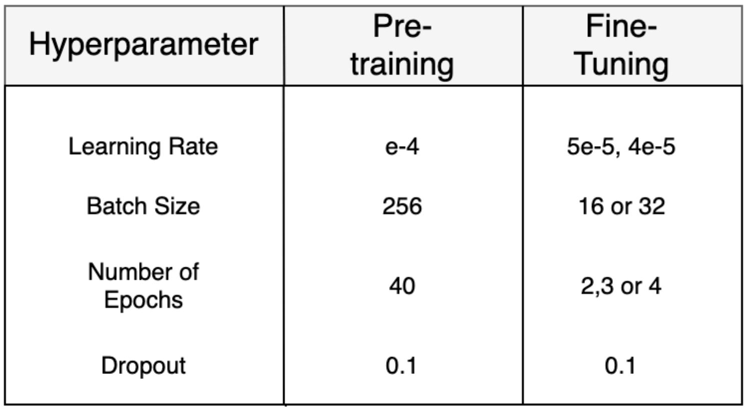


Figure 13: Fine tuning the BERT model for classification task [3]

Figure 13 represents the fine tuning process for a classification task for an input sentence. The task requires only a single sequence as input, and this is given to the BERT in the input layer. It is important to note that none of the tokens are masked unlike the pre-training phase. The tokens are converted to their corresponding input representations and given to the BERT model, which under the hood is essentially a stack of encoder layers. The input is processed through the different Attention layers and Feed forward networks, and the outputs are obtained in the output layer. Since the task in hand is a classification task, the output layer is modified to extract the output, which in this case will be a label, from the output C vector. All the other output vectors, i.e, $T_1, T_2, ..T_N$ corresponding to the different input tokens are not useful in this case. At the end of the fine-tuning process, the model parameters would have been adjusted in order minimise the losses.



Hyperparameter	Pre-training	Fine-Tuning
Learning Rate	e-4	5e-5, 4e-5
Batch Size	256	16 or 32
Number of Epochs	40	2,3 or 4
Dropout	0.1	0.1

Figure 14: Model hyper-parameters settings for BERT Pre-training and Fine tuning

Model Hyper-parameters During the pre-training and fine-tuning of the BERT model, model hyper-parameters have to be set in order to obtain models with high accuracy. This is an important step of the training process. The following table 14, shows the optimal hyper-parameter values that was used by Google in their experiments. These parameters can be tweaked during the experimental set up, based on the downstream task.

0.4 Experimental Results

The BERT model was fine-tuned for performing eleven different tasks in the field of Natural Language Processing, and the model achieved state-of-the-art results on each on those tasks [3]. The downstream tasks were designed in a diverse manner with different tasks involving Question-Answering, Sentiment Analysis, Classification and so on. Moreover, all the tasks were performed using both the variants of BERT,i.e, BERT-Base and BERT-Large. Out of the eleven tasks that were performed, eight of them were GLUE tasks, and are used as a benchmark for comparing the different models. Three other tasks were Question-Answer based tasks performed on two versions of the SQuAD(Stanford Question Answering Dataset) datasets and SWAG dataset. The following table summarises the results obtained for the different GLUE tasks.

System	MNLI-(m/mm)	QQP	QNLI	SST-2	CoLA	STS-B	MRPC	RTE
Pre-OpenAI SOTA	80.6/80.1	66.1	82.3	93.2	35.0	81.0	86.0	61.7
BiLSTM+ELMo+Attn	76.4/76.1	64.8	79.8	90.4	36.0	73.3	84.9	56.8
OpenAI GPT	82.1/81.4	70.3	87.4	91.3	45.4	80.0	82.3	56.0
BERT-Base	84.6/83.4	71.2	90.5	93.5	52.1	85.8	88.9	66.4
BERT-Large	86.7/85.9	72.1	92.7	94.9	60.5	86.5	89.3	70.1

Table 1: Results of GLUE tasks on different systems

As seen from 1, the BERT systems perform better than their counterparts on each task in the GLUE framework. For each of task, BERT was trained over 3 or 4 epochs with a training batch size of 32 and learning rate of $5e-5$, $4e-5$ or $3e-5$, whichever works better for the individual tasks. Another interesting observation is the fact that BERT-Large outperforms BERT-Base in each of the task. However, the computational time required for training BERT-Large was much more than that required for BERT-Base.
The metrics used for all the tasks except *QQP, MRPC* and *STS-B* are *accuracy scores* of GLUE framework. *QQP* and *MRPC* use the *F1 score* metric and *STS-B* uses *Spearman Correlation.*
The BERT model outperforms the other systems in SQuAD and SWAG tasks as well. All the hyper-parameters used for these tasks are the same as before, however the batch size for these tasks was taken as 16 instead of 32.

0.5 Discussion and Conclusion

As seen through the course of this report, BERT model has made monumental strides in the field of Natural Language Processing and has proven to have many advantages over its counterparts in realising downstream tasks. In this section, we discuss some of these advantages and the downfalls that has been observed with the BERT language model.

Some of the advantages of the BERT model are listed below:

- Simultaneous processing of tokens in the input sequence
- Bi-directional context awareness enabled due to the simultaneous processing of token

- Unified architecture for pre-training and fine tuning phase
- Ease in realising fine tuned downstream tasks

Some of the disadvantages of the BERT model are:

- Pre-training is computationally very expensive. It takes approximately 5 days to train BERT-Base on multiple GPUs.
- Slow convergence due to masking during pre-training. This is because the model is made to predict only the masked words of a sequence, i.e, only approximately 15% of the words in a given sequence of tokens.

In my opinion, one way to tackle the problem of slow convergence could be to increase the fraction of masked words in an input sequence. Further, the same input sequence could be permuted differently and masked randomly while maintaining the semantic correctness. This is will probably help in adding a little bias to the model and increase the rate of learning, thereby reducing the time taken for convergence. Whether this will work out or not is something that could be looked at as future work in this domain.

To summarise, the BERT model was tested with 11 Natural Language Processing tasks, i.e, 8 General Language Understanding(GLUE) tasks, a Squad v1.1 task that tests the system for Question-Answering, a Squad v2.0 task which is essentially an extension of the Squad v1.1 task for Question-Answering but excluding short answers from paragraphs in the dataset, and SWAG task for evaluating sentence pair continuation. BERT achieved state-of-the-art results for all 11 tasks in both the variants, i.e BERT-Base and BERT-Large, however, BERT-Large outperformed BERT-Base on all the 11 tasks.[3]

The pre trained-BERT model is now being fine tuned for performing a large number of natural language processing tasks in various domains, with great accuracy and optimal time utilisation. Some of the more famous variants that have developed out of the BERT model are RoBERTa, which is used by Facebook for content moderation and detecting fake news; bioBERT, which is used in the biomedical industry for biomedical text data mining and videoBERT, which is used by YouTube for generating video captions automatically. Effective utilisation of transfer learning makes BERT one of the most powerful language models today. With Natural Language Processing gaining more and more momentum in different industries, the contribution of BERT in dynamic business environment is something to look forward to.

Bibliography

[1] M. E. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer, "Deep contextualized word representations," 2018. cite arxiv:1802.05365Comment: NAACL 2018. Originally posted to openreview 27 Oct 2017. v2 updated for NAACL camera ready.

[2] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems* (I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, eds.), vol. 30, pp. 5998–6008, Curran Associates, Inc., 2017.

[3] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, (Minneapolis, Minnesota), pp. 4171–4186, Association for Computational Linguistics, June 2019.

[4] A. Miaschi and F. Dell'Orletta, "Contextual and non-contextual word embeddings: an in-depth linguistic investigation," in *Proceedings of the 5th Workshop on Representation Learning for NLP*, (Online), pp. 110–119, Association for Computational Linguistics, July 2020.

[5] M. Peters, M. eumann, L. Zettlemoyer, and W.-t. Yih, "Dissecting contextual word embeddings: Architecture and representation," in *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, 2018.

[6] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," 2013.

[7] J. Pennington, R. Socher, and C. Manning, "GloVe: Global vectors for word representation," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, (Doha, Qatar), pp. 1532–1543, Association for Computational Linguistics, Oct. 2014.

[8] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language models are unsupervised multitask learners," 2019.

[9] D. Hu, "An introductory survey on attention mechanisms in nlp problems," 2018.