CHAPTER 4

BERT Algorithms Explained

This chapter takes a deep dive into the BERT algorithm for sentence embedding along with various training strategies, including MLM and NSP. We will also see an implementation of a text classification system using BERT.

How Does BERT Work?

BERT makes use of a transformer to learn contextual relations between words in a text. A transformer has two mechanisms—an encoder and a decoder—but BERT only requires the encoder mechanism. BERT uses a bidirectional approach and reads the text input sequentially, which allows the model to learn the context of a word based on its surrounding words. The input to the encoder is a sequence of tokens that are embedded into vectors. The vectors are then passed into the neural network and an output sequence of vectors is then generated corresponding to the input. The output vector for a word is dependent on the context in which it occurs. For example, the vector for the word "like" in the sentence "He likes to play cricket" would be different than the vector for the same word in the sentence "His face turned red like a tomato."

N. Sabharwal and A. Agrawal, *Hands-on Question Answering Systems with BERT*,
https://doi.org/10.1007/978-1-4842-6664-9_4

This procedure involves text processing steps before even starting the model-building phase. The next section discusses the text processing steps used in BERT.

Text Processing

There is a specific set of rules for representing the input text for the BERT model. This, too, is responsible for better functioning of the model. If we look into the embeddings, the input embedding in BERT is a combination of the following three types of embeddings.

- **Position embeddings:** Positional embeddings are used to learn the information of order in the embeddings. As in transformers the information related to order is missed, positional embeddings are used to recover it. For each of the positions in the input sequence, BERT learns a unique positional embedding. With the help of these positional embeddings, BERT is able to express the position of words in a sentence as it captures this sequence or order information.
- **Segment embeddings:** BERT also learns unique embedding for the first and second sentences to help the model distinguish between them. It can also take sentence pairs as inputs for tasks like question answering.
- **Token embeddings:** For every token in the WordPiece token vocabulary, token embeddings are learned. The WordPiece token vocabulary contains subwords of words in the corpus. As an example, for the word "Question," this vocabulary set will include all possible subwords of "Question," such as ["Questio", "Questi"…], and so on.

Figure 4-1 shows an example of sequences of embeddings in BERT.

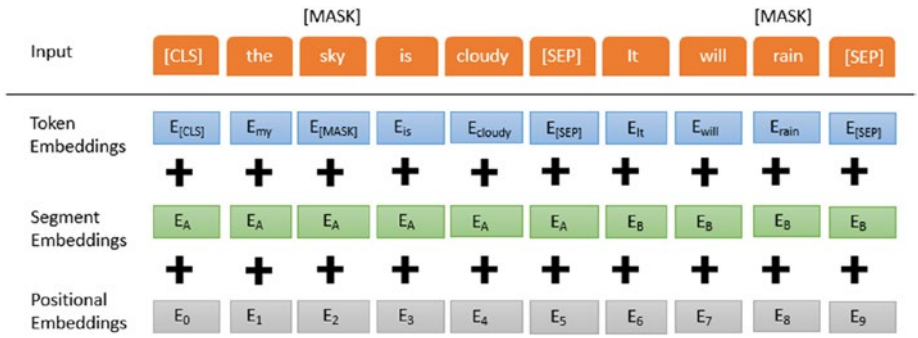


Figure 4-1. *BERT embeddings*

The input representation of a given token is constructed by summing the token, segment, and position embeddings. This makes it a comprehensive embedding scheme that contains a lot of useful information for the model.

For an NLP task where the job is to predict next word in a sentence, if we go with a directional approach, it has some limitations. However, BERT provides two strategies to learn contextual information: MLM and NSP. During training in BERT, both of these tasks will be trained together. When using these two strategies, the model tries to achieve the goal of minimizing the combined loss function.

Masked Language Modeling

BERT is a deep bidirectional model that is more powerful than a left-to-right model or the shallow concatenation of a left-to-right and a right-to-left model. The BERT network can effectively capture information from both the right and left context of a token. This goes from the first layer itself and all the way through to the last layer. Previously, language models were trained on left-to-right context, which made them susceptible to information less. Even though the ELMo model greatly improved on the

existing techniques using the shallow concatenating of the two LSTM language models, that wasn't enough. BERT has proven to be more significant than the existing techniques where MLM plays a crucial role.

In a masked language task, some of the words in text are randomly masked. The context words surrounding a [MASK] token are used to predict the [MASK] word. When word sequences are being fed into BERT, 15% of the words in each sequence are replaced with a [MASK] token. These 15% of words are randomly selected. Of these, 80% are masked, 10% are replaced with a random word, and 10% are retained. This is done because if 100% of the masked words were used then the model wouldn't necessarily produce good token representations for nonmasked words. The model performance is improved, as too much focusing on a particular position or tokens has been prevented. On the basis of the context provided by the nonmasked words in the sequence, the model tries to predict the original value of the masked words.

These processes that need to be followed for generation of word embedding using BERT:

- Addition of a classification layer on top of the encoder output.
- Multiplication of the output vectors by the embedding matrix, thus transforming them into the vocabulary dimension.
- Calculation of the probability of each word in the vocabulary with softmax.

The loss function in BERT only considers the prediction of the masked values; the prediction of the nonmasked words is ignored. This makes the model converge slower than directional ones. As an example, for the sentence "The birds are flying in the clear blue sky," if we are training the bidirectional model instead of predicting the next word in the sequence, a model can be built to predict the missing word from within

the sequence itself. Now, consider a token "flying" and mask it so that it can be considered missing. The model would now need to be trained in such a way that it can predict the value of this missing or masked token in the sentence "The birds are [MASK] in the clear blue sky." This is the essence of MLM, which enables the model to understand the relationships between words in a sentence.

Next Sentence Prediction

The NSP task is similar to next word prediction in a sentence. NSP predicts the next sentence in document, whereas the latter works for prediction of missing words in a sentence. BERT is also trained on the NSP task. This is required so that our model is able to understand how different sentences in a text corpus are related to each other. During the training of the BERT model, the sentence pairs are taken as input. It then predicts if the second sentence in the pair is the subsequent sentence in the original document. To achieve this, 50% of inputs are taken such that the second sentence is the subsequent sentence as in the original document, whereas the other 50% comprises the pair where the second sentence is chosen randomly from the document. It is assumed that the random second sentence is disconnected from the first sentence.

As an example, consider two different instances of training data for Sentence A and Sentence B:

```
Instance 1
Sentence A – I saw a bird flying in the sky.
Sentence B – It was a blue sparrow.
Label – IsNextSentence
Instance 2
Sentence A – I saw a bird flying in the sky.
Sentence B – The dog is barking.
Label – NotNextSentence
```

As we can see, for Instance 1 Sentence B is logically subsequent to Sentence A, but the same is not true for Instance 2, which is quite clear from the labels `IsNextSentence` and `NotNextSentence`, respectively.

These inputs are being processed even before the training process starts to differentiate between two sentences. The procedure is outlined here.

1. Two tokens are inserted in a sentence pair. One of the tokens [CLS] is inserted at the beginning of the first sentence and other token [SEP] is inserted at the end of each sentence. The two sentences are both tokenized and separated from each other by the separation token and then fed as a single input sequence into the model.
2. For each of the token sentences, embedding is added that indicates whether it is Sentence A or Sentence B. These sentence embeddings are basically similar in concept to token embeddings with a vocabulary of 2.
3. Along with the sentence embeddings, positional embeddings are also added to each of the tokens, which helps to indicate the position of the token in the sequence.

Now, the following steps are performed to predict if the second sentence is actually connected to the first.

1. The whole input sequence is passed though the transformer model.
2. With the help of the simple classification layer, the output of the [CLS] token is transformed into a 2X1 shaped vector.

3. Thereby, the probability of `IsNextSentence` is computed with the help of softmax.

As we know, BERT is used for variety of NLP tasks such as document summarization, question answering systems, document or sentence classification, and so on. Now, let's see how BERT can be used for classification of sentences.

Text Classification Using BERT

BERT can be used for a variety of language tasks. A small layer added to the core model allows use of BERT for tasks like classification, question answering, named entity recognition, and so on. The BERT model is fine tuned for this purpose. For classification tasks, a classification layer is added on top of the transformer output for the [CLS] token, similar to NSP. Most of the hyperparameters stay the same as in BERT training, but some of them require tuning to achieve state-of-the art-results for text classification tasks. Figure 4-2 gives an example of determining whether a given tweet is hate speech or not.

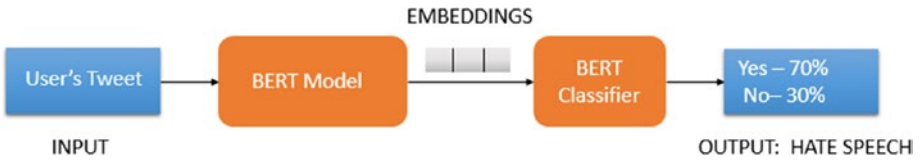


Figure 4-2. *An example of classification using BERT*

Similar types of tasks such as such as document classification, sentiment analytics, and so on, can also be achieved using BERT.

Next, we will see how a pretrained model of text classification can be configured in your system. Follow the steps listed here to configure or install the necessary prerequisites.

1. Make sure Python is installed on your system. Open a command prompt and run the following command to determine if Python is installed, as shown in Figure 4-3.

   ```
   Python
   ```

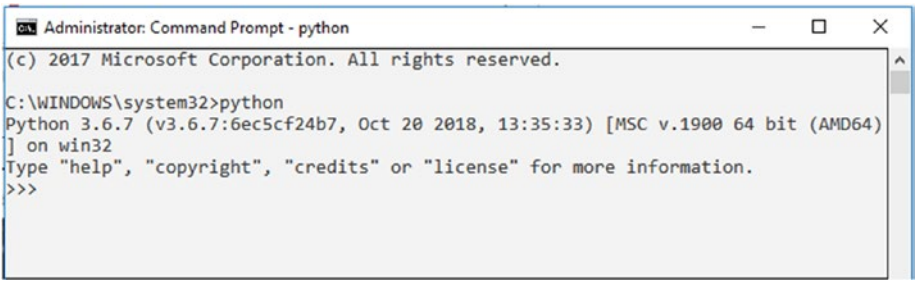


Figure 4-3. *Python console*

   This will start the Python console at the command prompt. If Python is not installed on your system, download and install Python as per your operating system using this link: https://www.python.org/downloads/

2. Next, install Jupyter Notebook. Open a command prompt and run the following command.

   ```
   pip install notebook
   ```

3. Open a command prompt and run the following command to run Jupyter Notebook.

   ```
   jupyter notebook
   ```

   The notebook will start in your default browser with localhost as the host address and port number as 8888, along with a unique token ID, as shown in Figure 4-4.

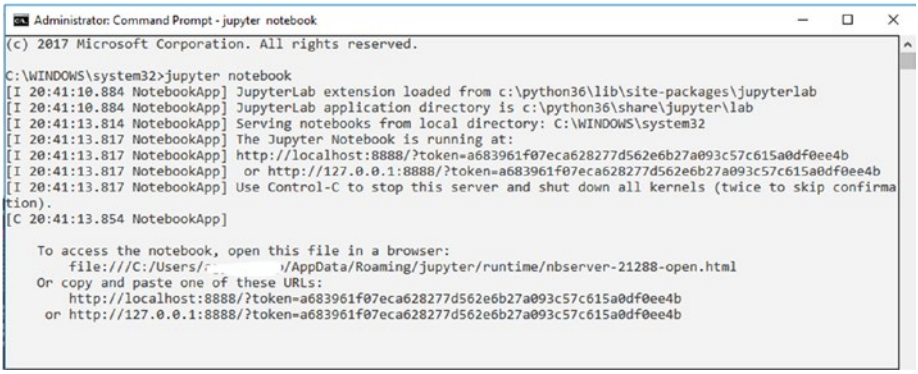


Figure 4-4. *Jupyter Notebook console*

4. You can also use Google Colab Notebook for the same purpose. It provides a fast and free environment to run your Python code in case your system doesn't have sufficient resources available. You can also use the graphics processing units (GPUs) and Tensor Processing Units (TPUs) for free, but for a limited time (12 hours) in Google Colab. You just need a Google account to log in to Google Colab Notebook. For this book, we will be using Google Colab Notebook to demonstrate text classification using BERT. Log in to your Google account and click `https://colab.research.google.com`. You will see the screen shown in Figure 4-5.

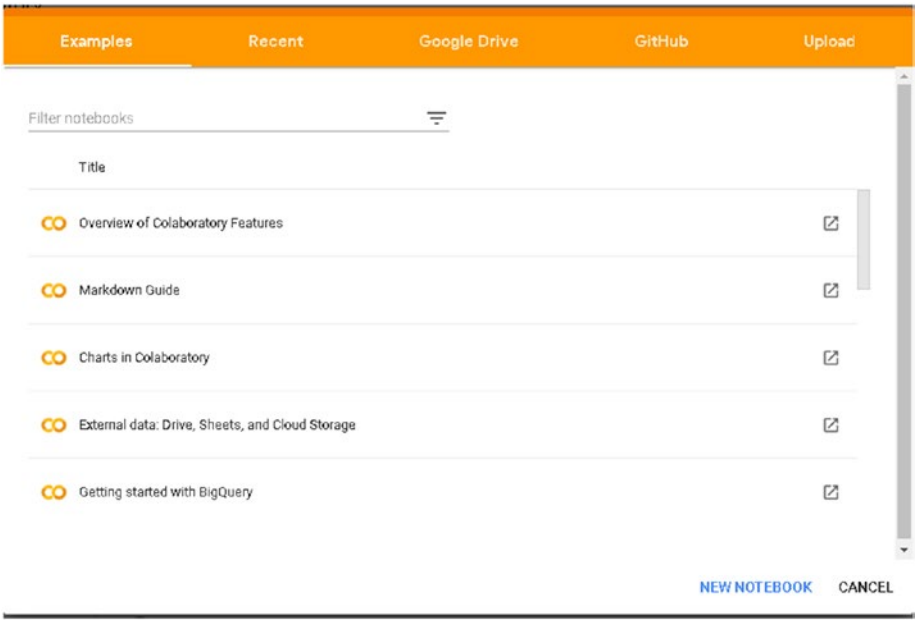


Figure 4-5. *Google Colab interface to create or import a notebook*

5. To create a new Colab notebook, click New Notebook in the bottom right corner as shown in Figure 4-5.

6. Install TensorFlow. Run the following command in your Jupyter Notebook or Colab Notebook.

```
pip install tensorflow
```

We have now installed all prerequisites for this exercise. Please follow the steps listed next to configure a pretrained model for text classification using BERT.

1. BERT model files and required code can be downloaded from the GitHub repository. Open the command prompt and clone the GitHub repository (i.e., `https:/github.com/google-research/bert.git`) onto the system by typing the following command:

   ```
   git clone https://github.com/google-research/bert.git
   ```

2. Downloaded model files containing the weights and other necessary BERT files. Depending on your requirements, a BERT pretrained model needs to be selected from this list.

   - BERT Base, Uncased
   - BERT Large, Uncased
   - BERT Base, Cased
   - BERT Large, Cased

3. If you have an access to a cloud TPU, BERT Large can be used; otherwise, the BERT Base model should be used. Further selection can be made from the cased and uncased models.

4. The data for fine-tuning the BERT model are expected to be in the format that BERT understands. Data have to be divided into three parts: train, dev, and test. As a rule of thumb, train should contain 80% of data and the remaining 20% will be divided into dev and test. You need to make a folder containing three separate files: `train.tsv`, `dev.tsv`, and `test.tsv`. The `train.tsv` file will be used for training the model, `dev.tsv` will be used for

developing the system, and `test.csv` will be used for evaluating model performance over unseen data. Both `train.tsv` and `dev.tsv` should not have headers and should have four columns as shown below.

1	1	a	text example belonging to class 1
2	1	a	text example belonging to class 1
3	2	a	text example belonging to class 2
4	0	a	text example belonging to class 0

Here are details of the columns used.

a. **First Column**: Represents IDs of sample.

b. **Second Column**: Classification labels corresponding to examples.

c. **Third Column**: Throw-away column.

d. **Fourth Column**: This represents the actual textual sentence that needs to be classified.

5. The `test.tsv` file should have a header line, unlike the other two files, and should appear as shown here.

id	sentence
1.	first test example
2.	second test example
3.	third test example

6. To train the model, you need to navigate to the directory into which the model has been cloned. Afterward, enter the following command at the command prompt:

```
python bert/run_classifier.py \
--task_name=cola \
--do_train=true \
--do_eval=true \
--data_dir=./data \
--vocab_file=$BERT_BASE_DIR/vocab.txt \
--bert_config_file=$BERT_BASE_DIR/bert_config.json \
--init_checkpoint=$BERT_BASE_DIR/bert_model.ckpt \
--max_seq_length=128 \
--train_batch_size=32 \
--learning_rate=2e-5 \
--num_train_epochs=3.0 \
--output_dir=./bert_output/
```

   If length of your training data text is longer than 128 words then the value for `max_seq_length` can be increased to 512. If you are training the model over a CPU system, then you can reduce the training size to avoid an out-of-memory error.

   When the training is finished, the reports get stored in the `bert_output` directory.

7. This trained BERT model is now ready to use for prediction purposes. If we have to make a prediction for new data, then data need to be stored in `test.tsv`. Go to the directory where the trained model files have been stored. Please refer to the highest-

number (latest model file) `model.ckpt` file seen in the `bert_output` directory. These files contain the weights of the model trained. Now run the following commands at the command prompt to obtain the classification result, which will be stored in `test_results.tsv` in the `bert_output` directory location.

```
python bert/run_classifier.py \
--task_name=cola \
--do_predict=true \
--data_dir=./data \
--vocab_file=$BERT_BASE_DIR/vocab.txt \
--bert_config_file=$BERT_BASE_DIR/bert_config.json \
--init_checkpoint=$TRAINED_CLASSIFIER \
--max_seq_length=128 \
--output_dir=./bert_output/
```

Please note that the value for the `max_seq_length` parameters should be the same as what was used during the training process.

For this book, we will demonstrate implementation of a question classification dataset where questions will be classified into their respective categories. There are mainly two types of questions, factoid (nondescriptive) and non-factoid questions. As an example, "What is the temperature in Delhi?" is a factoid question, as it is looking for an answer based on some facts. "What is temperature?" is a non-factoid question, as it is looking for text snippets about temperature. For this implementation, please refer to the dataset at https://cogcomp.seas.upenn.edu/Data/QA/QC/.

Now we will see how a question classification system can be implemented using BERT.

1. For this implementation, we will download the BERT base-cased model from GitHub as described previously.

2. The question classification dataset is already in the format required for training the BERT model. The data are split into `train.tsv`, `dev.tsv`, and `test.tsv` sets. In `train.tsv` and `dev.tsv`, we do not have any headers. The following is a description of the columns in the file.

   - **First Column**: Index for data point.
   - **Second Column**: Classification label (i.e., factoid or non-factoid). In this dataset, factoid is represented by 0 and 1 for non-factoid.
   - **Third Column**: Throwaway column with value a.
   - **Fourth Column**: Actual question text.

   Then we create data folder and save these files in the folder. Please refer Figures 4-6 through 4-8 for some examples from training files.

1	0	1	a	How did serfdom develop in and then leave Russia ?
2	1	0	a	What films featured the character Popeye Doyle ?
3	2	1	a	How can I find a list of celebrities' real names ?
4	3	0	a	What fowl grabs the spotlight after the Chinese Year of the Monkey ?
5	4	0	a	What is the full form of .com ?
6	5	0	a	What contemptible scoundrel stole the cork from my lunch ?
7	6	0	a	What team did baseball's St. Louis Browns become ?
8	7	0	a	What is the oldest profession ?
9	8	1	a	What are liver enzymes ?

Figure 4-6. *Snapshot of* `Dev.tsv`

```
55  1   a   What is the nature of learning ?
56  0   a   What's the only color Johnny Cash wears on stage ?
57  0   a   What's the term for a young fox ?
58  0   a   What U.S. state lived under six flags ?
59  1   a   What is the Kashmir issue ?
60  0   a   Where is the Loop ?
```

Figure 4-7. *Snapshot of* `train.tsv`

```
User_ID Description
1   I have a mac OS and I want to upgrade it to latest. Please help
2   My OS X is 10.8 , how to upgrade it ?
3   What is the process to update Mac OS?
4   I am no more able to see my deleted email, can you help?
5   I accidently deleted my emails, is there a way I can i recover them
```

Figure 4-8. *Snapshot of* `test.tsv`

3. Now, navigate to the directory where the downloaded BERT model exists.

4. As mentioned earlier, execute the command for training at the command prompt. The model output after completion of training gets stored in the location that has been defined under the `bert_output` parameter, as shown in Figure 4-9.

```
python run_classifier.py --task_name=cola --do_
train=true --do_eval=true --data_dir=$BERT_
BASE_DIR/data --vocab_file=$BERT_BASE_DIR/
bert_output/cased_L-12_H-768_A-12/vocab.
txt --bert_config_file=$BERT_BASE_DIR/bert_
output/ cased_L-12_H-768_A-12/bert_config.
json --init_checkpoint=$BERT_BASE_DIR/bert_
output/ model.ckpt-2023 --max_seq_length=
128 --train_batch_size=32 --learning_
rate=2e-5 --num_train_epochs=3.0 --output_
dir=$BERT_BASE_DIR/bert_output/
```

$BERT_BASE_DIR is a directory where you must have downloaded code from GitHub.

```
C:\BERT_base\bert>python run_classifier.py --task_name=cola --do_train=true --do_eval=true --data_dir=C:/BERT_base/base/
data --vocab_file=C:/BERT_base/base/bert_output/cased_L-12_H-768_A-12/vocab.txt --bert_config_file=C:/BERT_base/base/ber
t_output/ cased_L-12_H-768_A-12/bert_config.json --init_checkpoint=C:/BERT_base/base/bert_output/ model.ckpt-2023 --max_
seq_length=128 --train_batch_size=32 --learning_rate=2e-5 --num_train_epochs=3.0 --output_dir=C:/BERT_base/base/bert_out
put/
```

Figure 4-9. *Command to train BERT model*

5. After completion of training, we can classify the test data using the trained model. Run the following command at the command prompt to get a prediction for questions present in the `test.tsv` file as shown in Figure 4-10.

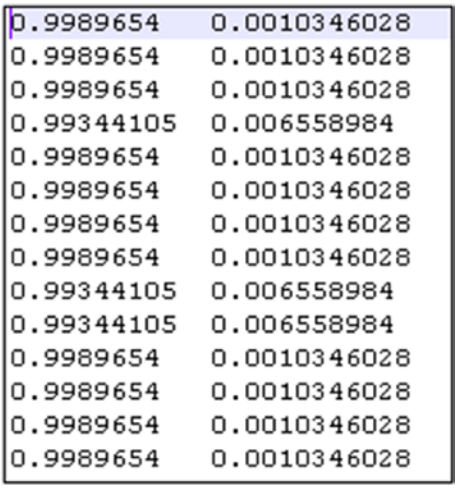
   ```
   python bert/run_classifier.py --task_name=cola --do_
   predict=true --data_dir=$BERT_BASE_DIR/data
   vocab_file=$BERT_BASE_DIR/bert_output/cased_L-
   12_H-768_A-12/vocab.txt --bert_config_file=$BERT_
   BASE_DIR/bert_output/ cased_L-12_H-768_A-12/
   bert_config.json --init_checkpoint=$TRAINED_
   CLASSIFIER --max_seq_length=128 --output_
   dir=$BERT_BASE_DIR/bert_output/
   ```

   $BERT_BASE_DIR is a directory where you must have downloaded code from GitHub.

```
C:\BERT_base\bert>python bert/run_classifier.py --task_name=cola --do_predict=true --data_dir=C:/BERT_base/base/data voc
ab_file=C:/BERT_base/base/bert_output/cased_L-12_H-768_A-12/vocab.txt --bert_config_file=C:/BERT_base/base/bert_output/
cased_L-12_H-768_A-12/bert_config.json --init_checkpoint=$TRAINED_CLASSIFIER --max_seq_length=128 --output_dir=C:/BERT_b
ase/base/bert_output/
```

Figure 4-10. *Command for prediction*

6. The results of the classification are stored in the location that has been defined as the value for the `bert_output` parameter in the `test_results.tsv` file. As we can see in Figure 4-11, the result of classification is a probability distribution of a question to two classes. The class with the higher score will be considered the relevant one.



0.9989654	0.0010346028
0.9989654	0.0010346028
0.9989654	0.0010346028
0.99344105	0.006558984
0.9989654	0.0010346028
0.9989654	0.0010346028
0.9989654	0.0010346028
0.9989654	0.0010346028
0.99344105	0.006558984
0.99344105	0.006558984
0.9989654	0.0010346028
0.9989654	0.0010346028
0.9989654	0.0010346028
0.9989654	0.0010346028

Figure 4-11. *Prediction results snapshot*

The first column corresponds to the label 0 (Factoid) and the second column corresponds to the label 1 (non-factoid). From this generated `.csv` we can see whether the questions in the test data are Factoid or non-factoid questions.

This question type classification system is quite useful in a conversational system where a query or question entered by an end user needs to be classified to retrieve relevant results.

Benchmarks for BERT Model

BERT embedding model performance and accuracy have been continuously evaluated over different types of datasets for various NLP tasks. This is being done to check if BERT is able to achieve benchmark values already set up for these datasets by some other methods. These benchmarks are datasets that evaluate the working of specific aspects of a model. There exist many such benchmarks and some of them are discussed next.

GLUE Benchmark

General Language Understanding Evaluation (GLUE) is a collection of datasets that can be used to train, evaluate, and analyze NLP models. These different models are compared with each other over the GLUE dataset. To test a model's language understanding, the GLUE benchmark includes nine diverse task datasets. To evaluate a model, first it is trained over a dataset provided by GLUE and then it is scored on all nine tasks. The final performance score is the average of all nine tasks.

$$Final\ GLUE\ Score = \sum Individual\ Task\ Score$$

The model is required to have representation of its input and output changed so as to accommodate the task. For instance, during the pretraining of BERT, few words are masked when sentences are given as input. Because the input representation layer in BERT accommodates all of the GLUE tasks, there is no need to change this layer. However, the pretraining classification layer has to be removed. This layer is replaced with the one that accommodates the output of each GLUE task. The BERT model scores a state-of-the-art result on the GLUE benchmark, with a score of 80.5%.

SQuAD Dataset

The Stanford Question Answering Dataset (SQuAD) is a reading comprehension dataset, consisting of questions asked on a set of Wikipedia articles. The answer to each of the questions is either a text segment or a span from the passage, respectively. There are two versions of the SQuAD dataset.

- SQuAD 1.1
- SQuAD 2.0

SQuAD2.0 has 100,000 questions in addition to SQuAD 1.1, which contains 50,000 unanswered questions, but are similar to questions that were answerable. This was done so that SQuAD2.0 can do well in situations where no answers to questions are supported by a paragraph for a question.

BERT is able to achieve state-of-the-art results on the SQuAD dataset with minor modifications. It requires semicomplex preprocessing of data and postprocessing to deal with the variable-length nature of SQuAD context paragraphs and the character-level answer annotations used for SQuAD training. The BERT model was able to achieve an F1 score of 93.2 and 83.1 for SQuAD 1.0 and SQuAD v2.0 over test dataset, respectively.

IMDB Reviews Dataset

The IMDB dataset is an extensive movie review dataset that has been used for classification of viewer sentiments about films. This dataset consists of 25,000 highly polar movie reviews for training and 25,000 reviews for testing. In addition to the training and testing data, there are also additional unlabeled data. This dataset has also been used to evaluate BERT in a sentiment classification task.

RACE Benchmark

RACE is a large-scale reading comprehension dataset from examinations. The RACE dataset is used to evaluate models on a reading comprehension task. This dataset was collected from English examinations of Chinese students. It consists of nearly 28,000 passages and 100,000 questions generated by human experts. The number of questions is much larger in RACE as compared to other benchmark datasets. The BERT large model achieves a score of 73.8% on the RACE benchmark dataset.

Types of BERT-Based Models

BERT is a ground-breaking natural language model and its introduction in the ML world has led to development of various models that are based on it. Variants of the BERT model have been developed to cater to different types of NLP-based systems. Here are a few of the major variants of BERT:

- ALBERT
- RoBERTa
- DistilBERT
- StructBERT
- $BERT_{joint}$ for Natural Questions

ALBERT

ALBERT is a much smaller version of BERT that was introduced jointly by Google Research and the Toyota Technological Institute. It is a smarter, "lite" BERT and is also considered a natural successor to BERT. It can also be used to implement state-of-the-art NLP tasks. This is all possible with less computation power compared to BERT, but you need to

compromise on accuracy a little bit. ALBERT was basically created to make improvements in architecture and training methods so that better results are delivered with fewer required computation resources.

ALBERT has a BERT-like core architecture. It has a transformer encoder architecture and a vocabulary of 30,000 words, which is the same as BERT. However, there are substantial architectural improvements in ALBERT for efficient parameter usage.

1. **Factorized embedding parameterization:** In the case of BERT, the WordPiece embeddings size (E) is directly tied to the hidden layer size (H). It was observed that WordPiece embeddings are designed to learn context-independent representations, whereas the hidden layer embeddings are designed to learn context-dependent representations. In BERT we try to learn context-dependent representations through the hidden layers only.

   When H and E are tied together, we end up with a model with billions of parameters that are rarely updated in training. This happens as the embedding matrix, which is V*E where V is the large vocabulary, must scale with the H (hidden layers). This actually results in inefficient parameters, as these two items work for different purposes.

   In ALBERT, to make it more efficient we untie the two parameters and embedding parameters are split into two smaller matrices. Now the one-hot vectors are not directly projected into H; rather, they are projected into a smaller, lower dimension matrix E, and then E is projected into the hidden layers. Thus, the parameters get reduced from O (V*H) to Θ(V*E+E*H).

2. **Cross-layer parameter sharing:** ALBERT has a smoother transition from layer to layer in comparison to BERT and the parameter efficiency is improved by sharing of all the parameters across all layers. The feed-forward and attention parameters are all shared. This weight sharing is helpful in stabilizing the network parameters.

3. **Training changes: Sentence order prediction:** Similar to BERT, ALBERT also uses MLM but does not use NSP. Instead of NSP, ALBERT uses its own newly developed training method called sentence order prediction (SOP).

   The NSP loss used in BERT was not found to be a very effective training mechanism in subsequent studies. Hence, it was leveraged to develop SOP as NSP was unreliable.

   In ALBERT SOP, loss is used to model intersentence coherence. SOP was mainly created to focus on intersentence coherence loss instead of topic prediction, whereas BERT combines topic prediction with coherence prediction. Hence, ALBERT is able to learn finer grained intersentence cohesion by avoiding issues of topic prediction.

ALBERT, even though it has fewer parameters than BERT, gets results in less time. In the language benchmark tests SQuAD1.1, SQuAD2.0, MNLI SST-2, and RACE, ALBERT has significantly outperformed BERT, as we can see in the comparison in Table 4-1.

Table 4-1. *Comparison Between BERT and ALBERT Models*

Model	Parameters	SQuAD1.1	SQuAD2.0	MNLI	SST-2	RACE	Avg	Speedup
BERT base	108M	90.5/83.3	80.3/77.3	84.1	91.7	68.3	82.1	17.7x
BERT large	334M	92.4/85.8	83.9/80.8	85.8	92.2	73.8	85.1	3.8x
BERT xlarge	1270M	86.3/77.9	73.8/70.5	80.5	87.8	39.7	76.7	1.0
ALBERT base	12M	89.3/82.1	79.1/76.1	81.9	89.4	63.5	80.1	21.1x
ALBERT large	18M	90.9/84.1	82.1/79.0	83.8	90.6	68.4	82.4	6.5x
ALBERT xlarge	59M	93.0/86.5	85.9/83.1	85.4	91.9	73.9	85.5	2.4x
ALBERT xxlarge	233M	94.1/88.3	88.1/85.1	88.0	95.2	82.3	88.7	1.2x

RoBERTa

RoBERTa is an optimized method for pretraining NLP systems. RoBERTa (Robustly optimized BERT) was developed by the Facebook AI team and based on Google's BERT model. RoBERTa reimplemented the neural network architecture of BERT with additional pretraining improvements that achieve state-of-the-art results on several benchmarks.

RoBERTa and BERT share several configurations, but there are some model settings that differentiate the two models.

- **Reserved token:** BERT uses [CLS] and [SEP] as starting token and separator token, respectively, whereas RoBERTa uses <s> and </s> to convert sentences.
- **Size of sub-word:** BERT has about 30,000 sub-words, whereas in RoBERTa there are about 50,000 sub-words.

In addition, there are specific modifications and adjustments that help RoBERTa to perform better than BERT.

- **More training data:** During reimplementation of BERT, several changes were made to the hyperparameters of the BERT model and training was done with a higher magnitude of data with more iterations. RoBERTa uses more training data. It uses BookCorpus (16G), CC-NEWS (76G), OpenWebText (38G), and Stories (31G) data, whereas BERT uses only BookCorpus as training data.
- **Dynamic masking:** When BERT was being ported to create RoBERTa, the creators modified the word masking strategy. BERT mainly uses static masking, in which the words are masked from sentences during preprocessing. RoBERTa makes use of dynamic masking. Here, a new masking pattern is generated whenever a sentence is fed into training. RoBERTa duplicates training data 10 times and masks those data differently. It is experimentally observed that the dynamic masking improves performance and gives a better result than static masking.
- **Different training objective:** BERT captures the relationships between the sentences by training on NSP. Some training approaches without application of NSP provided better results, proving the ineffectiveness of NSP. Experiments were done to compare models trained with segment-pair with NSP, sentence-pair with NSP, full sentences without NSP, and doc-sentences without NSP. The models trained without NSP performed better on SQuAD1.1/2.0, MNLI-m, SST-2, and RACE.

- **Training on longer sequences:** Better results have been achieved when a model was trained on longer sequences. BERT base is trained with a batch size of 256 sequences via 1 million steps, but training on 2,000 sequences and 31,000 steps shows improvement in performance.

With the implementation of the design changes, the RoBERTa model delivered state-of-the-art performance on the MNLI, QNLI, RTE, and RACE tasks. It also realized a sizable performance improvement on the GLUE benchmark with a score of 88.5.

RoBERTa demonstrates that the tuning the BERT training procedure can result in performance improvement on a variety of NLP tasks. This highlights the importance of exploring the design choices in BERT training for better performance output.

DistilBERT

DistilBERT was introduced for knowledge distillation. This knowledge distillation was required to address the drawbacks of computation of large numbers of parameters. The NLP models that have been developed recently show an increase in parameter count, now reaching parameter counts as high as in the tens of billions. Even though higher parameter count ensures optimal performance, it prevents model training and serving when computational resources are limited.

Knowledge distillation revolves around the idea that a larger model acts as a teacher for a smaller one that tries to replicate its outputs and sublayer activation for a given set of inputs. This is sometimes also known as teacher–student learning. It is a compression technique where the behavior of larger models is reproduced by the smaller ones. The output distribution from the teacher can be used for all possible targets, which helps in creation of a student with generalizability. For example, in the

sentence "The sky is [mask]" a teacher might assign high probabilities to words like "cloudy" and "clear." There are also chances that a high probability is assigned to the word "blue." This is helpful for the student model, so that it is able to generalize rather than only learn the correct target. This information is captured through the loss function that is being used to train the student. This loss function comprises a linear combination of three factors.

Distillation Loss

Distillation loss takes into consideration combination of the output probabilities of the teacher (t) and the student (s) as shown in the following equation.

$$L_{ce} = \sum_i t_i \log(s_i)$$

Distillation Loss

$$t_i = \exp(z_i/T) / \sum_j \exp(z_j/T)$$

Temperature Softmax

The teacher probabilities are calculated through temperature softmax. This is basically a modification to the softmax so that more granularities are obtained from the teacher model output distribution. This gives a smoother output distribution, as the size of larger probabilities are decreased and the smaller ones are increased. This helps to minimize the distillation loss.

Cosine Embedding Loss

Cosine embedding loss is a distance measure between the hidden representations for teacher and student. This helps in creation of a better model as it enables the student to imitate the teacher not just in the output layer, but in other layers, too.

Masked Language Modeling Loss

This is the same loss as used in training of the BERT model to predict the correct token value for the masked token in the sequence.

Architectural Modifications

The DistilBERT network architecture is also a transformer encoder model similar to BERT base, but it has half the number of layers. The hidden representations, though, are kept the same. This affects the parameter count, with a 66 million parameter count in the case of DistilBERT, whereas there are 110 million parameters in the teacher model. The reduction in the model size through the number of layers helps to achieve the drastic reduction in computation complexity. The reduction in the size of the vectors or the hidden state representations have also reduced the model size.

After the knowledge distillation, DistilBERT is able to achieve 97% of BERT base's score on the GLUE benchmark. This knowledge distillation has helped to condense the larger models or ensembles of models into a smaller student network. This has proven to be helpful in situations where the computational environment is limited.

StructBERT

StructBERT is a model based on BERT that incorporates language structures into BERT pretraining. The two linearization strategies help to incorporate language structure into BERT. Word-level ordering and sentence-level ordering are the two structural information sets that are leveraged in StructBERT. StructBERT achieves better generalizability and adaptability due to the incorporation of this structural pretraining. The dependency between the words as well as sentences is encoded in StructBERT.

Structural Pretraining in StructBERT

Similar to all the other BERT-based models, StructBERT also builds on the BERT architecture. The original BERT performs two unsupervised pretraining tasks, MLM and NSP. StructBERT is able to increase the ability of the MLM task. It shuffles a certain number of tokens after masking of words and predicts the right order. StructBERT is also able to understand the relationship between sentences in a better way. This is achieved by random swapping of the sentence order. This new BERT-based model captures the fine-grained word structure in every sentence.

After pretraining of the StructBERT it can be fine-tuned on task-specific data for a wide range of downstream tasks such as document summarization.

Pretraining Objectives

The pretraining objectives of the original BERT are extended in the case of StructBERT to fully utilize rich inner-sentence and intersentence structures in language. This is done in two ways.

1. **Word structural objective:** The BERT model fails to model sequential order and high-order word dependency in natural language explicitly. A good language model should be able to reconstruct a sentence from a given sentence that has randomly ordered words. StructBERT is able to implement this idea by supplementing BERT's training objectives with a new word structural objective. This new model objective gives the model the ability to restructure the sentence to have correct ordering of the randomly shuffled word tokens. This objective is trained together with the MLM objective from BERT.

2. **Sentence structural objective:** The original BERT model objective of NSP is extended in StructBERT by predicting both the next sentence as well as the previous sentence. This makes the pretrained model learn the sequential ordering of the sentences in a bidirectional manner.

These two auxiliary objectives are pretrained together with the original MLM objective to exploit inherent language structures.

$BERT_{joint}$ for Natural Questions

$BERT_{joint}$ is a BERT-based model for Natural Questions. The $BERT_{joint}$ model predicts short and long answers in a single model only instead of a pipeline approach. In this model each document is split into multiple instances of training with the help of overlapping token windows. This approach is used to create a balanced training set and is being followed by down sampling instances without an answer (null instances). The [CLS] token is used during training to predict null instances, and spans at inference time are ranked by the difference between the span score and the [CLS] score.

The model uses the Natural Questions (NQ) dataset that is a question answering dataset of 307,373 training examples, 7,830 development examples, and 7,842 test examples. For every example, a query is entered by the user over the Google search engine and the corresponding Wikipedia page that contains an answer. The Wikipedia page is annotated as an answer to the question.

The $BERT_{joint}$ model was initialized from the original BERT model that trained on the SQuAD 1.1 dataset. Afterward, this model was fine-tuned on Natural Questions training instances. It has used the Adam optimizer to minimize the loss. The $BERT_{joint}$ model for Natural Questions gives

dramatically better results than the baseline NQ systems. This variation of BERT offers a new way to design a question-answering system.

Conclusion

This chapter looked deeper into BERT, along with its two algorithms, MLM and NSP. We also discussed a sample text classification model developed using BERT. We also examined the behavior of BERT over different benchmark datasets, along with multiple variations of BERT. In the next chapter, we discuss the design of a question answering system using BERT.