The purpose of this article is to provide a step-by-step tutorial on how to use BERT for multi-classification task. BERT ( **B**idirectional **E**ncoder **R**epresentations from **T**ransformers), is a new method of pre-training language representation by Google that aimed to solve a wide range of Natural Language Processing tasks. This model is based on unsupervised, deeply bidirectional system and managed to achieve state-of-the-art results when it was first released to the public in 2018. If you would like to know more about it, you can find the academic paper in the following [link](https://arxiv.org/abs/1810.04805).

There are 5 sections in this tutorial:

1. Setup and installation
2. Dataset Preparation
3. Training model
4. Prediction
5. Conclusion

**[Section 1] Setup and installation**

In this tutorial, I will be using Ubuntu 18.04 paired with a single GeForce RTX 2080 Ti. Personally, I do not recommend training without GPU due to the extremely long training time as the base model is simply too large.

**Virtual Environment**

It is advisable to set up a virtual environment. If you are using Ubuntu for the first time, open up terminal and change the directory to your desired location. It will be the root folder of your environment. Run the following command to install pip:

sudo apt-get install python3-pip

Then, ran the following command to install virtualenv module:

sudo pip3 install virtualenv

You can now create your own virtual environment (replace bertenv with any name that you preferred):

virtualenv bertenv

If you prefer to do it without virtualenv module, there is another way to create virtual environment by using just python3

python3 -m venv bertenv

You should have created a **bertenv** folder. Check out the following [link](https://gist.github.com/Geoyi/d9fab4f609e9f75941946be45000632b) for more information. You can activate the virtual environment using the following command:

source bertenv/bin/activate

**Github**

Clone the repository from the following [link](https://github.com/google-research/bert). Once it is completed, extract the zip file and put it to a directory of your choice. You should have a **bert-master** folder. I put it alongside the virtual environment folder. Hence, in the root directory I have the following subfolders:

1. bertenv
2. bert-master

**Python modules**

BERT only requires tensorflow module. You have to install version that is equal or greater than 1.11.0. Make sure that you installed either the CPU version or the GPU but not both.

tensorflow >= 1.11.0 # CPU Version of TensorFlow. tensorflow-gpu >= 1.11.0 # GPU version of TensorFlow.

You can install it via pip or the [*requirement.txt*](https://github.com/google-research/bert/blob/master/requirements.txt) file that is located in the **bert-master** folder.

**BERT model**

We will need a base model for fine-tuning process. I will be using [**BERT-Base, Cased**](https://storage.googleapis.com/bert_models/2018_10_18/cased_L-12_H-768_A-12.zip) ( 12-layer, 768-hidden, 12-heads , 110M parameters) for this tutorial. If you would like to try **BERT-Large** ( 24-layer, 1024-hidden, 16-heads, 340M parameters), make sure that you have sufficient memory. 12GB GPU is not sufficient to run BERT-Large. Personally, I would recommend you to use 64GB GPU for BERT-Large. At the time of this writing, the team behind BERT also released other models such as Chinese, Multilingual and Whole Word Masking. Kindly check them out via the following [link](https://github.com/google-research/bert#pre-trained-models). Once you have downloaded the file, extract it and you should have the following files:

1. Three ckpt files
2. vocab.txt
3. bert\_config.json

Put them inside a **model**folder and move it to **bert-master** folder. Kindly proceed to to the next section on dataset preparation.

**[Section 2] Dataset Preparation**

Data preparation is a lot complicated for BERT as the official github link do not cover much on what kind of data is needed. First and foremost, there are 4 classes that can be used for sequence classification tasks:

1. Xnli (Cross-Lingual NLI)
2. Mnli (Multi-Genre Natural Language Inference)
3. Mrpc (Microsoft Research Paraphrase Corpus)
4. Cola (The Corpus of Linguistic Acceptability)

All of the classes are based on DataProcessor class (refer to [*run\_classifier.py*](https://github.com/google-research/bert/blob/master/run_classifier.py) file at line 177) that is used to extract data into the following:

1. **guid**: Unique id for the example.
2. **text\_a**: String data. The untokenized text of the first sequence. For single sequence tasks, only this sequence must be specified.
3. **text\_b**: (Optional) string data. The untokenized text of the second sequence. Only must be specified for sequence pair tasks.
4. **label**: String data. The label of the example. This should be specified for train and evaluate examples, but not for test examples.

In other words, we can either modify our dataset to mimic the schema and format to one the 4 classes or write our own class that extend the DataProcessor class to read our data. In this tutorial, I will convert the dataset into cola format as it is the simplest among all of them. Examples of other dataset can be found in the following [link](https://gluebenchmark.com/tasks) (GLUE version).

In the original version of BERT, [*run\_classifier.py*](https://github.com/google-research/bert/blob/master/run_classifier.py) is based on reading input from three tsv files:

1. train.tsv (no header)
2. dev.tsv (evaluation, no header)
3. test.tsv (header is required)

For *train.tsv* and *dev.tsv*, you should have the following format (no header):

a550d 1 a To clarify, I didn't delete these pages.  
kcd12 0 a Dear god this site is horrible.  
7379b 1 a I think this is not appropriate.  
cccfd 2 a The title is fine as it is.

1. **Column 1**: The guid for the example. It can be any unique identifier.
2. **Column 2**: The label for the example. It is string-based and can be in the form of text instead of just numbers. For simplicity, I will just use numbers here.
3. **Column 3**: The untokenized text of the second sequence. Only must be specified for sequence pair tasks. Since we are doing single sequence tasks, this is just a throwaway column. We will just fill it with ‘a’ for all the rows.
4. **Column 4**: The untokenized text of the first sequence. Fill it with the text for the example.

As for the *test.tsv*, you should have the following format (header is required):

guid text  
casd4 I am not going to buy this useless stuff.  
3ndf9 I wanna be the very best, like no one ever was

**Convert data from other sources into desired format**

If you have data that differs from the format given above, you can easily convert it using pandas module and sklearn module. Install the module via pip (make sure that the virtual environment is activated):

pip install pandas

Install sklearn as well if you intend to use train\_test\_split:

pip install sklearn

For example, if we have the following train dataset in csv format:

id,text,label  
sadcc,This is not what I want.,1  
cj1ne,He seriously have no idea what it is all about,0  
123nj,I don't think that we have any right to judge others,2

We can easily load our dataset and convert it into the respective format using the following code (modify the path accordingly):

**Create dataframe from csv file**

import pandas as pddf\_train = pd.read\_csv('dataset/train.csv')

**Create a new dataframe from existing dataframe**

df\_bert = pd.DataFrame({'guid': df\_train['id'],  
 'label': df\_train['label'],  
 **'alpha': ['a']\*df\_train.shape[0]**,  
 'text': df\_train['text']})

The part highlighted in bold means that we are going to fill the column alpha with string a based on the number of rows in df\_train dataframe. shape[0] refers to number of rows while shape[1] refers to number of columns.

**Output tsv file**

df\_bert\_train.to\_csv('dataset/train.tsv', **sep='\t'**, index=False, header=False)

Do not be surprise by the to\_csv function call as both tsv and csv have similar format except for the separators. In other words, we only need to provide the proper tab separator and it will become a tsv file (highlighted in bold).

Here are the full working code snippet to create all the required files (modify the path accordingly).

import pandas as pd  
from sklearn.model\_selection import train\_test\_split#read source data from csv file  
df\_train = pd.read\_csv('dataset/train.csv')  
df\_test = pd.read\_csv('dataset/test.csv')#create a new dataframe for train, dev data  
df\_bert = pd.DataFrame({'guid': df\_train['id'],  
 'label': df\_train['label'],  
 'alpha': ['a']\*df\_train.shape[0],  
 'text': df\_train['text']})#split into test, dev  
df\_bert\_train, df\_bert\_dev = train\_test\_split(df\_bert, test\_size=0.01)#create new dataframe for test data  
df\_bert\_test = pd.DataFrame({'guid': df\_test['id'],  
 'text': df\_test['text']})#output tsv file, no header for train and dev  
df\_bert\_train.to\_csv('dataset/train.tsv', sep='\t', index=False, header=False)  
df\_bert\_dev.to\_csv('dataset/dev.tsv', sep='\t', index=False, header=False)  
df\_bert\_test.to\_csv('dataset/test.tsv', sep='\t', index=False, header=True)

Once you have all the files required, move the **dataset**folder into **bert-master** folder. Let’s move on to the next section to fine-tune your model.

**[Section 3] Training model**

The easiest way to fine-tune BERT’s model is running the run\_classifier.py via the command line (terminal). Before that, we need to modify the python file based on our labels. The original version is meant for binary classification using 0 and 1 as labels. If you are doing multi-classification or binary classification with different labels, you need to change the*get\_labels()*function of **ColaProcessor**class (line 354, modify accordingly if you are using other **DataProcessor**class):

**Original code**

def get\_labels(self):  
 return ["0", "1"]

**5 labels multi-classification task**

def get\_labels(self):  
 return ["0", "1", "2", "3", "4"]

**Binary classification using different labels**

def get\_labels(self):  
 return ["POSITIVE", "NEGATIVE"]

If you encounter any KeyError such as follows, it means that your get\_labels function does not match your dataset:

label\_id = label\_map[example.label]  
KeyError: '2'`

We are now ready for training process. If you are using NVIDIA GPU, you can type the following in the terminal to check the status and CUDA version.

nvidia-smi

Change the directory to point to **bert-master**folder, make sure that you have the **dataset**folder and the required files in **bert-master** folder. It is advisable to run the training via the command line instead of using jupyter notebook due to the following reasons:

1. The official code uses 2 units indentation which differs from the defaults 4 units indentation in notebook.
2. Memory issues and additional code to configure which GPU to be used for training.

**Parameters**

Before that, let’s explore the parameters that can be fine-tuned for the training process:

1. **data\_dir**: The input directory that contains train.tsv, dev.tsv and test.tsv.
2. **bert\_config\_file**: The config json file corresponding to the pre-trained BERT model. This specifies the model architecture.
3. **task\_name**: The name of the task to train. 4 options are available (xnli, mrpc, mnli, cola).
4. **vocab\_file**: The vocabulary file that the BERT model was trained on.
5. **output\_dir**: The output directory where the model checkpoints will be written.
6. **init\_checkpoint**: Initial checkpoint (usually from a pre-trained BERT model).
7. **do\_lower\_case**: Whether to lower case the input text. Should be True for uncased and False for cased.
8. **max\_seq\_length**: The maximum total input sequence length after WordPiece tokenization. Sequences longer than this will be truncated, and sequences shorter will be padded. Default is 128.
9. **do\_train**: Whether to run training on train.tsv.
10. **do\_eval**: Whether to run evaluation on the dev.tsv.
11. **do\_predict**: Whether run the model in inference mode on test.tsv
12. **train\_batch\_size**: Total batch size for training. Default is 32.
13. **eval\_batch\_size**: Total batch size for evaluation. Default is 8
14. **predict\_batch\_size**: Total batch size for testing and prediction. Default is 8.
15. **learning\_rate**: Initial learning rate for Adam. Default is 5e-5.
16. **num\_train\_epochs**: Total number of training epochs to perform. Default is 3.0.
17. **warmup\_proportion**: Proportion of training to perform linear learning rate warmup for from 0 to 1. Default is 0.1 means 10%.
18. **save\_checkpoints\_steps**: Number steps interval to save the model checkpoint. Default is 1000.
19. **iterations\_per\_loop**: Number of steps interval for each estimator call. Default is 1000.
20. **use\_tpu**: Whether to use TPU.
21. **tpu\_name**: The Cloud TPU to use for training.
22. **tpu\_zone**: GCE zone where the Cloud TPU is located in.
23. **gcp\_project**: Project name for the Cloud TPU-enabled project
24. **master**: TensorFlow master URL.
25. **num\_tpu\_cores**: Only used if **use\_tpu**is True. Total number of TPU cores to use.

Do not be overwhelmed by the number of parameters as we are not going to specify each and everyone of them.

**Training explanation**

From the [official documentation](https://github.com/google-research/bert), it recommends to export the path as variable via the following command line call (replace export to set if you are using Windows OS):

export BERT\_BASE\_DIR=/path/to/bert/uncased\_L-12\_H-768\_A-12

In this tutorial, I will not be exporting the path as you still need to specify it in the command line. Just make sure that you organized your folders properly and you are good to go. To specify the GPU, you need to type it before the python call (example as follow, do not run it):

CUDA\_VISIBLE\_DEVICES=0 python script.py

0 refers to the order of GPU. Use the following command to check it:

nvidia-smi

Inside **bert-master** folder, create a output folder. I will just call it **bert\_output**. Make sure that you have the following folders and files in **bert-master** folder:

1. **dataset**folder(contains train.tsv, dev.tsv, test.tsv)
2. **model**folder (contains ckpt, vocab.txt, bert\_config.json)
3. **bert\_output**folder (empty)

**Training via command line**

Make sure that the terminal points to bert-master directory and virtual environment is activated. Modify the parameters based on your preference and run it. I made the following changes:

1. Reduced the **train\_batch\_size**to 2: If you have sufficient memory, feel free to increase it. This affects the training time. The higher it is, the shorter the training time.
2. Increased the **save\_checkpoint\_steps**to 10000. I do not want to have this many checkpoints as each checkpoint models is 3x the original size. Rest assured that this script only keeps 5 models at a time. Older models will be deleted automatically. Highly recommend to just keep it to 1000 (default).
3. Reduced the **max\_seq\_length**to 64. Since 99% of my dataset does not exceed 64 in length, setting it higher will be redundant. Modify this accordingly based on your dataset. Default is 128. (Sequnce length refers to the character length after word piece tokenization, kindly takes this into account).

CUDA\_VISIBLE\_DEVICES=0 python run\_classifier.py --task\_name=cola --do\_train=true --do\_eval=true --data\_dir=./dataset --vocab\_file=./model/vocab.txt --bert\_config\_file=./model/bert\_config.json --init\_checkpoint=./model/bert\_model.ckpt --max\_seq\_length=64 --train\_batch\_size=2 --learning\_rate=2e-5 --num\_train\_epochs=3.0 --output\_dir=./bert\_output/ --do\_lower\_case=False --save\_checkpoints\_steps 10000

The training should have started indicated the number of steps run/sec. Check the bert\_output folder and you should notice the following:

1. three ckpt files (the model)
2. tf\_record
3. checkpoints and events files (temp file, can be safely ignored and deleted after the training)
4. graph.pbtxt

It might take a while from a few hours to a few days depends on your dataset and configurations.

**Training completed**

Once the training is completed,you should have a *eval\_results.txt* that indicates the performance of your model.

eval\_accuracy = 0.96741855  
eval\_loss = 0.17597112  
global\_step = 236962  
loss = 0.17553209

Identify the highest number of steps of the model. If you are unsure about it, kindly open the checkpoint in a text editor, you should see the following:

model\_checkpoint\_path: "model.ckpt-236962"  
all\_model\_checkpoint\_paths: "model.ckpt-198000"  
all\_model\_checkpoint\_paths: "model.ckpt-208000"  
all\_model\_checkpoint\_paths: "model.ckpt-218000"  
all\_model\_checkpoint\_paths: "model.ckpt-228000"  
all\_model\_checkpoint\_paths: "model.ckpt-236962"

In this case, the highest steps is 236962. We can now use this model to predict the results.

**[Section 4] Prediction**

For making prediction, we will use the same *run\_classifier.py*. This time we will need to specify **do\_predict**as True and set the **init\_checkpoint**to the latest model that we have which is model.ckpt-236962 (modify accordingly based on the highest steps that you have). However, you need to make sure that the **max\_seq\_length**is the same as the one you used for training.

CUDA\_VISIBLE\_DEVICES=0 python run\_classifier.py --task\_name=cola --do\_predict=true --data\_dir=./dataset --vocab\_file=./model/vocab.txt --bert\_config\_file=./model/bert\_config.json --init\_checkpoint=./bert\_output/model.ckpt-236962 --max\_seq\_length=64 --output\_dir=./bert\_output/

Once the process has completed, you should have a test\_results.tsv in **bert\_output**folder (depends on what you have specified for **output\_dir**). If you open it with a text editor, you should see the following output:

1.4509245e-05 1.2467547e-05 0.99994636  
1.4016414e-05 0.99992466 1.5453812e-05  
1.1929651e-05 0.99995375 6.324972e-06  
3.1922486e-05 0.9999423 5.038059e-06  
1.9996814e-05 0.99989235 7.255715e-06  
4.146e-05 0.9999349 5.270801e-06

The number of columns depends on the number of labels that you have. Each column represent each label in the order that you have specified for get\_labels() function. The value represent the propabilities of the prediction. For example, the model predicted that the first example belongs to the 3rd class since it has the highest probability.

**Mapping results to the respective classes**

If you would like to map the results to calculate the accuracy, you can do so with the following code (modify accordingly):

import pandas as pd#read the original test data for the text and id  
df\_test = pd.read\_csv('dataset/test.tsv', sep='\t')  
#read the results data for the probabilities  
df\_result = pd.read\_csv('bert\_output/test\_results.tsv', sep='\t', header=None)#create a new dataframe  
df\_map\_result = pd.DataFrame({'guid': df\_test['guid'],  
 'text': df\_test['text'],  
 'label': df\_result.idxmax(axis=1)})#view sample rows of the newly created dataframe  
df\_map\_result.sample(10)

**idxmax**is a funtion used to return index of first occurrence of maximum over requested axis. NA/null values are excluded. In this case, we passed 1 as the axis to indicate for column instead of row.

**[Section 5] Conclusion**

In this tutorial, we have learnt to fine-tune BERT for multi-classification task. For your information, BERT can be used on other Natural Language Processing tasks instead of just classification. Personally, I have tested the BERT-Base Chinese for emotion analysis as well and the results are surprisingly good. Bear in mind that non-Latin language such as Chinese and Korean are character tokenized instead of word tokenized. Feel free to try out other models on different kind of datasets. Thanks for reading and have a great day ahead. See you again in the next article!

**BERT-related stories**

1. [3 Ways to Optimize and Export BERT Model for Online Serving](https://towardsdatascience.com/3-ways-to-optimize-and-export-bert-model-for-online-serving-8f49d774a501)

**Reference**

1. <https://arxiv.org/abs/1810.04805>
2. <https://github.com/google-research/bert>
3. <https://gist.github.com/Geoyi/d9fab4f609e9f75941946be45000632b>
4. <https://gluebenchmark.com/tasks>
5. <https://github.com/google-research/bert/blob/master/run_classifier.py>
6. <https://mc.ai/a-guide-to-simple-text-classification-with-bert/>
7. [https://appliedmachinelearning.blog/2019/03/04/state-of-the-art-text-classi](https://appliedmachinelearning.blog/2019/03/04/state-of-the-art-text-classification-using-bert-model-predict-the-happiness-hackerearth-challenge/)