1. [Home](https://www.analyticsvidhya.com/blog/)
2. [Advanced](https://www.analyticsvidhya.com/blog/category/advanced/)
3. Transfer Learning for NLP: Fine-Tuning BERT for Text Classification

**Transfer Learning for NLP: Fine-Tuning BERT for Text Classification**

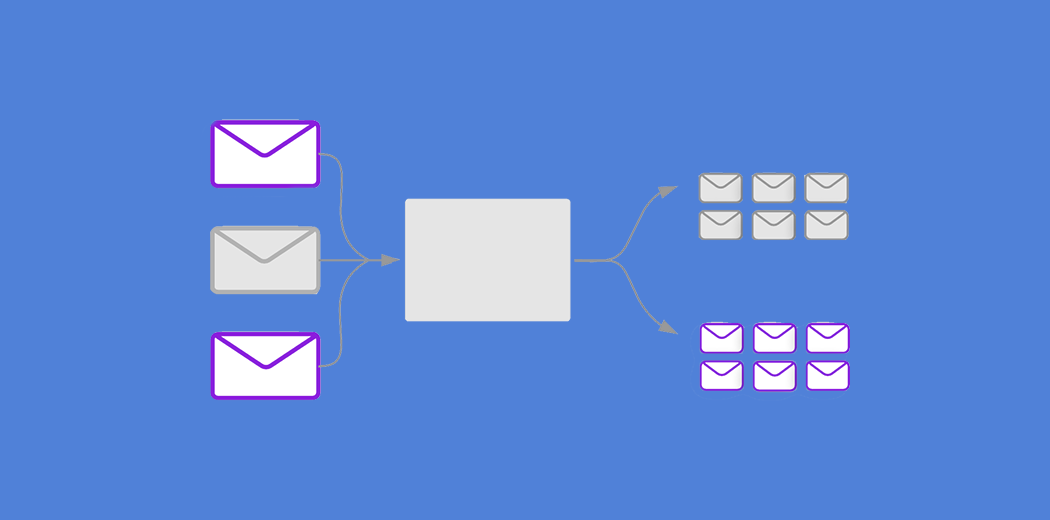
[](https://www.analyticsvidhya.com/blog/2020/07/transfer-learning-for-nlp-fine-tuning-bert-for-text-classification/)

[Prateek Joshi](https://www.analyticsvidhya.com/blog/author/pjoshi15/)20 Jun, 2022 • 10 min read

Introduction

With the advancement in deep learning, neural network architectures like recurrent neural networks (RNN and LSTM) and convolutional neural networks (CNN) have shown a decent improvement in performance in solving several Natural Language Processing (NLP) tasks like text classification, language modeling, machine translation, etc.

However, this performance of deep learning models in NLP pales in comparison to the performance of deep learning in Computer Vision.



One of the main reasons for this slow progress could be the lack of large labeled text datasets. Most of the labeled text datasets are not big enough to train deep neural networks because these networks have a huge number of parameters and training such networks on small datasets will cause overfitting.

Another quite important reason for NLP lagging behind computer vision was the lack of transfer learning in NLP. Transfer learning has been instrumental in the success of deep learning in computer vision. This happened due to the availability of huge labeled datasets like Imagenet on which deep CNN based models were trained and later they were used as pre-trained models for a wide range of computer vision tasks.

That was not the case with NLP until 2018 when **the transformer model was introduced by Google**. Ever since the transfer learning in NLP is helping in solving many tasks with state of the art performance.

[](https://community.analyticsvidhya.com/c/datahour/build-your-own-private-chatbot-using-genai-rag-b2be90)

In this article, I explain how do we fine-tune BERT for text classification.

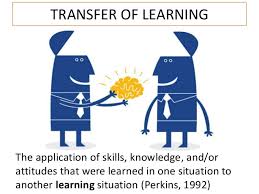
If you want to learn NLP from scratch, check out our course – [Natural Language Processing (NLP) Using Python](https://courses.analyticsvidhya.com/courses/natural-language-processing-nlp?utm_source=blog&utm_medium=fine_tune_BERT)

Table of Contents

1. Transfer Learning in NLP
2. What is Model Fine-Tuning?
3. Overview of BERT
4. Fine-Tune BERT for Spam Classification

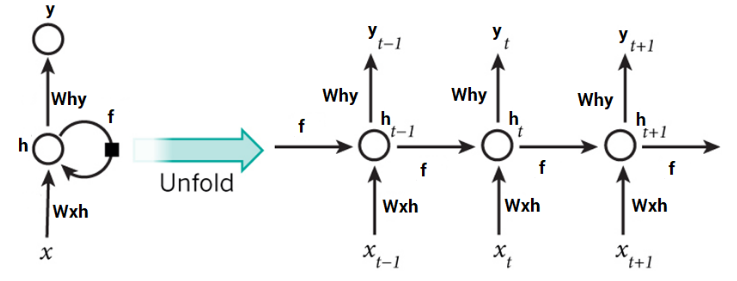
Transfer Learning in NLP

Transfer learning is a technique where a deep learning model trained on a large dataset is used to perform similar tasks on another dataset. We call such a deep learning model a pre-trained model. The most renowned examples of pre-trained models are the computer vision deep learning models trained on the [ImageNet dataset](https://www.analyticsvidhya.com/blog/2020/02/learn-image-classification-cnn-convolutional-neural-networks-3-datasets/?utm_source=blog&utm_medium=fine_tune_BERT). So, it is better to use a pre-trained model as a starting point to solve a problem rather than building a model from scratch.



This breakthrough of transfer learning in computer vision occurred in the year 2012-13. However, with recent advances in NLP, transfer learning has become a viable option in this NLP as well.

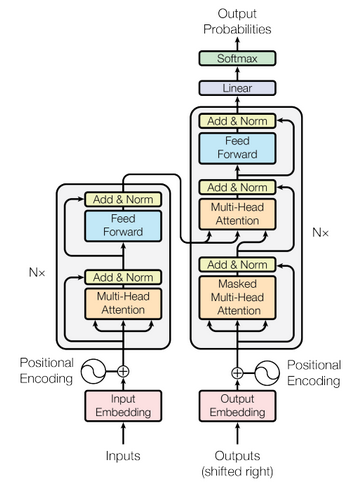
Most of the tasks in NLP such as text classification, language modeling, machine translation, etc. are sequence modeling tasks. The traditional machine learning models and neural networks cannot capture the sequential information present in the text. Therefore, people started using recurrent neural networks (RNN and LSTM) because these architectures can model sequential information present in the text.



A typical RNN

However, these recurrent neural networks have their own set of problems. One major issue is that RNNs can not be parallelized because they take one input at a time. In the case of a text sequence, an RNN or LSTM would take one token at a time as input. So, it will pass through the sequence token by token. Hence, training such a model on a big dataset will take a lot of time.

So, the need for transfer learning in NLP was at an all-time high. In 2018, the transformer was introduced by Google in the paper “[Attention is All You Need](https://arxiv.org/abs/1706.03762)” which turned out to be a groundbreaking milestone in NLP.



The Transformer – Model Architecture  
(Source: https://arxiv.org/abs/1706.03762)

Soon a wide range of transformer-based models started coming up for different NLP tasks. There are multiple advantages of using transformer-based models, but the most important ones are:

* First Benefit

These models do not process an input sequence token by token rather they take the entire sequence as input in one go which is a big improvement over RNN based models because now the model can be accelerated by the GPUs.

* 2nd Benefit

We don’t need labeled data to pre-train these models. It means that we have to just provide a huge amount of unlabeled text data to train a transformer-based model. We can use this trained model for other NLP tasks like text classification, named entity recognition, text generation, etc. This is how transfer learning works in NLP.

BERT and GPT-2 are the most popular transformer-based models and in this article, we will focus on BERT and learn how we can use a pre-trained BERT model to perform text classification.

What is Model Fine-Tuning?

BERT (Bidirectional Encoder Representations from Transformers) is a big neural network architecture, with a huge number of parameters, that can range from 100 million to over 300 million. So, training a BERT model from scratch on a small dataset would result in overfitting.

So, it is better to use a pre-trained BERT model that was trained on a huge dataset, as a starting point. We can then further train the model on our relatively smaller dataset and this process is known as model fine-tuning.

Different Fine-Tuning Techniques

* **Train the entire architecture** – We can further train the entire pre-trained model on our dataset and feed the output to a softmax layer. In this case, the error is back-propagated through the entire architecture and the pre-trained weights of the model are updated based on the new dataset.
* **Train some layers while freezing others** – Another way to use a pre-trained model is to train it partially. What we can do is keep the weights of initial layers of the model frozen while we retrain only the higher layers. We can try and test as to how many layers to be frozen and how many to be trained.
* **Freeze the entire architecture** – We can even freeze all the layers of the model and attach a few neural network layers of our own and train this new model. Note that the weights of only the attached layers will be updated during model training.

In this tutorial, we will use the third approach. We will freeze all the layers of BERT during fine-tuning and append a dense layer and a softmax layer to the architecture.

Overview of BERT

You’ve heard about BERT, you’ve read about how incredible it is, and how it’s potentially changing the NLP landscape. But what is BERT in the first place?

Here’s how the research team behind BERT describes the NLP framework:

“BERT stands for Bidirectional Encoder Representations from Transformers. It is designed to pre-train deep bidirectional representations from unlabeled text by jointly conditioning on both left and right context. As a result, the pre-trained BERT model can be fine-tuned with just one additional output layer to create state-of-the-art models for a wide range of NLP tasks.”

That sounds way too complex as a starting point. But it does summarize what BERT does pretty well so let’s break it down.

Firstly, BERT stands for **B**idirectional **E**ncoder **R**epresentations from **T**ransformers. Each word here has a meaning to it and we will encounter that one by one in this article. For now, the key takeaway from this line is – BERT is based on the Transformer architecture. Secondly, BERT is pre-trained on a large corpus of unlabelled text including the entire Wikipedia (that’s 2,500 million words!) and Book Corpus (800 million words).

This pre-training step is half the magic behind BERT’s success. This is because as we train a model on a large text corpus, our model starts to pick up the deeper and intimate understandings of how the language works. This knowledge is the swiss army knife that is useful for almost any NLP task.

Third, BERT is a **“deep bidirectional”** model. Bidirectional means that BERT learns information from both the left and the right side of a token’s context during the training phase.

To learn more about the BERT architecture and its pre-training tasks, then you may like to read the below article:

* [Demystifying BERT: A Comprehensive Guide to the Groundbreaking NLP Framework](https://www.analyticsvidhya.com/blog/2019/09/demystifying-bert-groundbreaking-nlp-framework/?utm_source=blog&utm_medium=fine_tune_BERT)

Fine-Tune BERT for Spam Classification

Now we will fine-tune a BERT model to perform text classification with the help of the Transformers library. You should have a basic understanding of defining, training, and evaluating neural network models in PyTorch. If you want a quick refresher on PyTorch then you can go through the article below:

* [A Beginner-Friendly Guide to PyTorch and How it Works from Scratch](https://www.analyticsvidhya.com/blog/2019/09/introduction-to-pytorch-from-scratch/?utm_source=blog&utm_medium=fine_tune_BERT)

**Link to [Colab Notebook](https://github.com/prateekjoshi565/Fine-Tuning-BERT/blob/master/Fine_Tuning_BERT_for_Spam_Classification.ipynb" \t "_blank)**

Problem Statement

We have a collection of SMS messages. Some of these messages are spam and the rest are genuine. Our task is to build a system that would automatically detect whether a message is spam or not.

The dataset that we will be using for this use case can be downloaded from [**here**](https://github.com/prateekjoshi565/Fine-Tuning-BERT/raw/master/spamdata_v2.csv) (right-click and click on “Save link as…”).

I suggest you use Google Colab to perform this task so that you can use the GPU. Firstly, activate the GPU runtime on Colab by clicking on *Runtime -> Change runtime type -> Select GPU*.

Install Transformers Library

We will then install Huggingface’s transformers library. This library lets you import a wide range of transformer-based pre-trained models. Just execute the code below to install the library.

!pip install transformers

Import Libraries

|  |  |
| --- | --- |
|  | import numpy as np |
|  | import pandas as pd |
|  | import torch |
|  | import torch.nn as nn |
|  | from sklearn.model\_selection import train\_test\_split |
|  | from sklearn.metrics import classification\_report |
|  | import transformers |
|  | from transformers import AutoModel, BertTokenizerFast |
|  |  |
|  | # specify GPU |
|  | device = torch.device("cuda") |

[view raw](https://gist.github.com/prateekjoshi565/48e2a18eb402cd98ee7b916145b52d67/raw/b87a394b7778d9d9875db3455b17df8e02c1eb15/import_lib_bert.py)[import\_lib\_bert.py](https://gist.github.com/prateekjoshi565/48e2a18eb402cd98ee7b916145b52d67#file-import_lib_bert-py)hosted with  by [GitHub](https://github.com/)

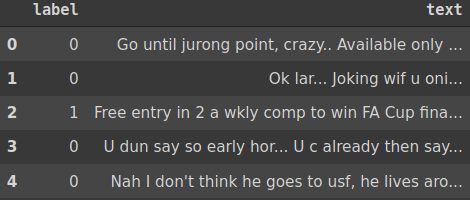
Load Dataset

You would have to upload the downloaded spam dataset to your Colab runtime. Then read it into a pandas dataframe.

|  |  |
| --- | --- |
|  | df = pd.read\_csv("spamdata\_v2.csv") |
|  | df.head() |

[view raw](https://gist.github.com/prateekjoshi565/5eeb347bc965f927d7c561ef20e64485/raw/d5fb4ec7c530817392e6d720934eaba5d9ad387a/load_data_bert.py)[load\_data\_bert.py](https://gist.github.com/prateekjoshi565/5eeb347bc965f927d7c561ef20e64485#file-load_data_bert-py)hosted with  by [GitHub](https://github.com/)

**Output:**



The dataset consists of two columns – “label” and “text”. The column “text” contains the message body and the “label” is a binary variable where 1 means spam and 0 means the message is not a spam.

Now we will split this dataset into three sets – train, validation, and test.

|  |  |
| --- | --- |
|  | # split train dataset into train, validation and test sets |
|  | train\_text, temp\_text, train\_labels, temp\_labels = train\_test\_split(df['text'], df['label'], |
|  | random\_state=2018, |
|  | test\_size=0.3, |
|  | stratify=df['label']) |
|  |  |
|  |  |
|  | val\_text, test\_text, val\_labels, test\_labels = train\_test\_split(temp\_text, temp\_labels, |
|  | random\_state=2018, |
|  | test\_size=0.5, |
|  | stratify=temp\_labels) |

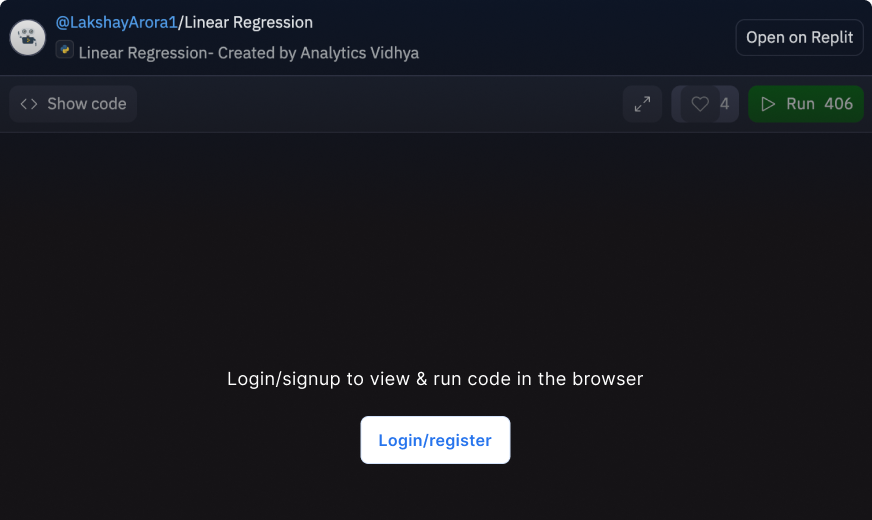
[view raw](https://gist.github.com/prateekjoshi565/a6ba56d3e5f96acd0d7a78c03c106699/raw/256ce80e36b96fd25d78aa367c9298a24c02ba9e/data_split_bert.py)[data\_split\_bert.py](https://gist.github.com/prateekjoshi565/a6ba56d3e5f96acd0d7a78c03c106699#file-data_split_bert-py)hosted with  by [GitHub](https://github.com/)

We will fine-tune the model using the train set and the validation set, and make predictions for the test set.

Import BERT Model and BERT Tokenizer

We will import the BERT-base model that has 110 million parameters. There is an even bigger BERT model called BERT-large that has 345 million parameters.

**Python Code:**

[](https://id.analyticsvidhya.com/auth/login/?next=https://www.analyticsvidhya.com/blog/2020/07/transfer-learning-for-nlp-fine-tuning-bert-for-text-classification/?&utm_source=coding-window-blog&source=coding-window-blog)

Let’s see how this BERT tokenizer works. We will try to encode a couple of sentences using the tokenizer.

|  |  |
| --- | --- |
|  | # sample data |
|  | text = ["this is a bert model tutorial", "we will fine-tune a bert model"] |
|  |  |
|  | # encode text |
|  | sent\_id = tokenizer.batch\_encode\_plus(text, padding=True) |
|  |  |
|  | # output |
|  | print(sent\_id) |

[view raw](https://gist.github.com/prateekjoshi565/48bc03ad0e2a66cc3c1b065cecc373db/raw/7151a41494cdaacf6d41df9941a3057312a580b7/testing_tokenizer_bert.py)[testing\_tokenizer\_bert.py](https://gist.github.com/prateekjoshi565/48bc03ad0e2a66cc3c1b065cecc373db#file-testing_tokenizer_bert-py)hosted with  by [GitHub](https://github.com/)

**Output:**

{‘input\_ids’: [[101, 2023, 2003, 1037, 14324, 2944, 14924, 4818, 102, 0],  
[101, 2057, 2097, 2986, 1011, 8694, 1037, 14324, 2944, 102]],

‘attention\_mask’: [[1, 1, 1, 1, 1, 1, 1, 1, 1, 0], [1, 1, 1, 1, 1, 1, 1, 1, 1, 1]]}

As you can see the output is a dictionary of two items.

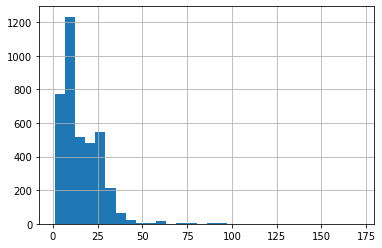
* ‘input\_ids’ contains the integer sequences of the input sentences. The integers 101 and 102 are special tokens. We add them to both the sequences, and 0 represents the padding token.
* ‘attention\_mask’ contains 1’s and 0’s. It tells the model to pay attention to the tokens corresponding to the mask value of 1 and ignore the rest.

Tokenize the Sentences

Since the messages (text) in the dataset are of varying length, therefore we will use padding to make all the messages have the same length. We can use the maximum sequence length to pad the messages. However, we can also have a look at the distribution of the sequence lengths in the train set to find the right padding length.

|  |  |
| --- | --- |
|  | # get length of all the messages in the train set |
|  | seq\_len = [len(i.split()) for i in train\_text] |
|  |  |
|  | pd.Series(seq\_len).hist(bins = 30) |

[view raw](https://gist.github.com/prateekjoshi565/b1e8894f881a4db7acee4434c0b3712a/raw/979039d89a181a75782f0c2aeb19baefcc9d4eb5/seq_len_bert.py)[seq\_len\_bert.py](https://gist.github.com/prateekjoshi565/b1e8894f881a4db7acee4434c0b3712a#file-seq_len_bert-py)hosted with  by [GitHub](https://github.com/)



We can clearly see that most of the messages have a length of 25 words or less. Whereas the maximum length is 175. So, if we select 175 as the padding length then all the input sequences will have length 175 and most of the tokens in those sequences will be padding tokens which are not going to help the model learn anything useful and on top of that, it will make the training slower.

Therefore, we will set 25 as the padding length.

|  |  |
| --- | --- |
|  | # tokenize and encode sequences in the training set |
|  | tokens\_train = tokenizer.batch\_encode\_plus( |
|  | train\_text.tolist(), |
|  | max\_length = 25, |
|  | pad\_to\_max\_length=True, |
|  | truncation=True |
|  | ) |
|  |  |
|  | # tokenize and encode sequences in the validation set |
|  | tokens\_val = tokenizer.batch\_encode\_plus( |
|  | val\_text.tolist(), |
|  | max\_length = 25, |
|  | pad\_to\_max\_length=True, |
|  | truncation=True |
|  | ) |
|  |  |
|  | # tokenize and encode sequences in the test set |
|  | tokens\_test = tokenizer.batch\_encode\_plus( |
|  | test\_text.tolist(), |
|  | max\_length = 25, |
|  | pad\_to\_max\_length=True, |
|  | truncation=True |
|  | ) |

[view raw](https://gist.github.com/prateekjoshi565/9a6d3138e1c43cef3db98909800a0377/raw/c9b29e3babd4fe043310aaa4bbeb388aea9d9541/tokenize_bert.py)[tokenize\_bert.py](https://gist.github.com/prateekjoshi565/9a6d3138e1c43cef3db98909800a0377#file-tokenize_bert-py)hosted with  by [GitHub](https://github.com/)

So, we have now converted the messages in train, validation, and test set to integer sequences of length 25 tokens each.

Next, we will convert the integer sequences to tensors.

|  |  |
| --- | --- |
|  | ## convert lists to tensors |
|  |  |
|  | train\_seq = torch.tensor(tokens\_train['input\_ids']) |
|  | train\_mask = torch.tensor(tokens\_train['attention\_mask']) |
|  | train\_y = torch.tensor(train\_labels.tolist()) |
|  |  |
|  | val\_seq = torch.tensor(tokens\_val['input\_ids']) |
|  | val\_mask = torch.tensor(tokens\_val['attention\_mask']) |
|  | val\_y = torch.tensor(val\_labels.tolist()) |
|  |  |
|  | test\_seq = torch.tensor(tokens\_test['input\_ids']) |
|  | test\_mask = torch.tensor(tokens\_test['attention\_mask']) |
|  | test\_y = torch.tensor(test\_labels.tolist()) |

[view raw](https://gist.github.com/prateekjoshi565/f8dde394ac17632499a9a68080ec7a7f/raw/e7b469f665039884610b1038282362251cd882bd/seq_to_tensor_bert.py)[seq\_to\_tensor\_bert.py](https://gist.github.com/prateekjoshi565/f8dde394ac17632499a9a68080ec7a7f#file-seq_to_tensor_bert-py)hosted with  by [GitHub](https://github.com/)

Now we will create dataloaders for both train and validation set. These dataloaders will pass batches of train data and validation data as input to the model during the training phase.

|  |  |
| --- | --- |
|  | from torch.utils.data import TensorDataset, DataLoader, RandomSampler, SequentialSampler |
|  |  |
|  | #define a batch size |
|  | batch\_size = 32 |
|  |  |
|  | # wrap tensors |
|  | train\_data = TensorDataset(train\_seq, train\_mask, train\_y) |
|  |  |
|  | # sampler for sampling the data during training |
|  | train\_sampler = RandomSampler(train\_data) |
|  |  |
|  | # dataLoader for train set |
|  | train\_dataloader = DataLoader(train\_data, sampler=train\_sampler, batch\_size=batch\_size) |
|  |  |
|  | # wrap tensors |
|  | val\_data = TensorDataset(val\_seq, val\_mask, val\_y) |
|  |  |
|  | # sampler for sampling the data during training |
|  | val\_sampler = SequentialSampler(val\_data) |
|  |  |
|  | # dataLoader for validation set |
|  | val\_dataloader = DataLoader(val\_data, sampler = val\_sampler, batch\_size=batch\_size) |

[view raw](https://gist.github.com/prateekjoshi565/3182eb33c4cbff9ed4c2b05636c1377c/raw/4237e2b7dee765808801896e485b8912697901f8/dataloaders_bert.py)[dataloaders\_bert.py](https://gist.github.com/prateekjoshi565/3182eb33c4cbff9ed4c2b05636c1377c#file-dataloaders_bert-py)hosted with  by [GitHub](https://github.com/)

Define Model Architecture

If you can recall, earlier I mentioned in this article that I would freeze all the layers of the model before fine-tuning it. So, let’s do it first.

|  |  |
| --- | --- |
|  | # freeze all the parameters |
|  | for param in bert.parameters(): |
|  | param.requires\_grad = False |

[view raw](https://gist.github.com/prateekjoshi565/f0bc5fdf47bae4d5dc5065bc24b86c57/raw/daeaa82a468b8181cffc070c782cd97f96c87df3/freeze_model_bert.py)[freeze\_model\_bert.py](https://gist.github.com/prateekjoshi565/f0bc5fdf47bae4d5dc5065bc24b86c57#file-freeze_model_bert-py)hosted with  by [GitHub](https://github.com/)

This will prevent updating of model weights during fine-tuning. If you wish to fine-tune even the pre-trained weights of the BERT model then you should not execute the code above.

Moving on we will now let’s define our model architecture.

|  |  |
| --- | --- |
|  | class BERT\_Arch(nn.Module): |
|  |  |
|  | def \_\_init\_\_(self, bert): |
|  |  |
|  | super(BERT\_Arch, self).\_\_init\_\_() |
|  |  |
|  | self.bert = bert |
|  |  |
|  | # dropout layer |
|  | self.dropout = nn.Dropout(0.1) |
|  |  |
|  | # relu activation function |
|  | self.relu = nn.ReLU() |
|  |  |
|  | # dense layer 1 |
|  | self.fc1 = nn.Linear(768,512) |
|  |  |
|  | # dense layer 2 (Output layer) |
|  | self.fc2 = nn.Linear(512,2) |
|  |  |
|  | #softmax activation function |
|  | self.softmax = nn.LogSoftmax(dim=1) |
|  |  |
|  | #define the forward pass |
|  | def forward(self, sent\_id, mask): |
|  |  |
|  | #pass the inputs to the model |
|  | \_, cls\_hs = self.bert(sent\_id, attention\_mask=mask) |
|  |  |
|  | x = self.fc1(cls\_hs) |
|  |  |
|  | x = self.relu(x) |
|  |  |
|  | x = self.dropout(x) |
|  |  |
|  | # output layer |
|  | x = self.fc2(x) |
|  |  |
|  | # apply softmax activation |
|  | x = self.softmax(x) |
|  |  |
|  | return x |

[view raw](https://gist.github.com/prateekjoshi565/9f98897cc5ad1574b729c8cfcdfd2e53/raw/c24baa0bef34a7872beb51465d2fac325aaa0a7c/architecture_bert.py)[architecture\_bert.py](https://gist.github.com/prateekjoshi565/9f98897cc5ad1574b729c8cfcdfd2e53" \l "file-architecture_bert-py)hosted with  by [GitHub](https://github.com/)

|  |  |
| --- | --- |
|  | # pass the pre-trained BERT to our define architecture |
|  | model = BERT\_Arch(bert) |
|  |  |
|  | # push the model to GPU |
|  | model = model.to(device) |

[view raw](https://gist.github.com/prateekjoshi565/e41850d47c00803f66b8c1dfe94f21f0/raw/474ba9784de467259e78da65a6f328f7aea44508/model_start_bert.py)[model\_start\_bert.py](https://gist.github.com/prateekjoshi565/e41850d47c00803f66b8c1dfe94f21f0#file-model_start_bert-py)hosted with  by [GitHub](https://github.com/)

We will use AdamW as our optimizer. It is an improved version of the Adam optimizer. To learn more about it do check out this [paper](https://arxiv.org/abs/1711.05101).

|  |  |
| --- | --- |
|  | # optimizer from hugging face transformers |
|  | from transformers import AdamW |
|  |  |
|  | # define the optimizer |
|  | optimizer = AdamW(model.parameters(), |
|  | lr = 1e-5) # learning rate |

[view raw](https://gist.github.com/prateekjoshi565/ddd5a70a00d77a28ca013a70d9732ad3/raw/2994ca76c6f73e66f41b4720460575a60c75892d/optm_bert.py)[optm\_bert.py](https://gist.github.com/prateekjoshi565/ddd5a70a00d77a28ca013a70d9732ad3#file-optm_bert-py)hosted with  by [GitHub](https://github.com/)

There is a class imbalance in our dataset. The majority of the observations are not spam. So, we will first compute class weights for the labels in the train set and then pass these weights to the loss function so that it takes care of the class imbalance.

|  |  |
| --- | --- |
|  | from sklearn.utils.class\_weight import compute\_class\_weight |
|  |  |
|  | #compute the class weights |
|  | class\_weights = compute\_class\_weight('balanced', np.unique(train\_labels), train\_labels) |
|  |  |
|  | print("Class Weights:",class\_weights) |

[view raw](https://gist.github.com/prateekjoshi565/f2526a6f5957ea3e56b39b475e1dcb76/raw/5061807f6d343aebb753926c74892b614b044cc1/compute_class_wt_bert.py)[compute\_class\_wt\_bert.py](https://gist.github.com/prateekjoshi565/f2526a6f5957ea3e56b39b475e1dcb76#file-compute_class_wt_bert-py)hosted with  by [GitHub](https://github.com/)

**Output:** [0.57743559 3.72848948]

|  |  |
| --- | --- |
|  | # converting list of class weights to a tensor |
|  | weights= torch.tensor(class\_weights,dtype=torch.float) |
|  |  |
|  | # push to GPU |
|  | weights = weights.to(device) |
|  |  |
|  | # define the loss function |
|  | cross\_entropy = nn.NLLLoss(weight=weights) |
|  |  |
|  | # number of training epochs |
|  | epochs = 10 |

[view raw](https://gist.github.com/prateekjoshi565/786cb83e5afb71b10332d061c9f18d6b/raw/3aed753141ec685fa89d16c3e9c18300203ae47f/define_loss_bert.py)[define\_loss\_bert.py](https://gist.github.com/prateekjoshi565/786cb83e5afb71b10332d061c9f18d6b#file-define_loss_bert-py)hosted with  by [GitHub](https://github.com/)

Fine-Tune BERT

So, till now we have defined the model architecture, we have specified the optimizer and the loss function, and our dataloaders are also ready. Now we have to define a couple of functions to train (fine-tune) and evaluate the model, respectively.

|  |  |
| --- | --- |
|  | # function to train the model |
|  | def train(): |
|  |  |
|  | model.train() |
|  |  |
|  | total\_loss, total\_accuracy = 0, 0 |
|  |  |
|  | # empty list to save model predictions |
|  | total\_preds=[] |
|  |  |
|  | # iterate over batches |
|  | for step,batch in enumerate(train\_dataloader): |
|  |  |
|  | # progress update after every 50 batches. |
|  | if step % 50 == 0 and not step == 0: |
|  | print(' Batch {:>5,} of {:>5,}.'.format(step, len(train\_dataloader))) |
|  |  |
|  | # push the batch to gpu |
|  | batch = [r.to(device) for r in batch] |
|  |  |
|  | sent\_id, mask, labels = batch |
|  |  |
|  | # clear previously calculated gradients |
|  | model.zero\_grad() |
|  |  |
|  | # get model predictions for the current batch |
|  | preds = model(sent\_id, mask) |
|  |  |
|  | # compute the loss between actual and predicted values |
|  | loss = cross\_entropy(preds, labels) |
|  |  |
|  | # add on to the total loss |
|  | total\_loss = total\_loss + loss.item() |
|  |  |
|  | # backward pass to calculate the gradients |
|  | loss.backward() |
|  |  |
|  | # clip the the gradients to 1.0. It helps in preventing the exploding gradient problem |
|  | torch.nn.utils.clip\_grad\_norm\_(model.parameters(), 1.0) |
|  |  |
|  | # update parameters |
|  | optimizer.step() |
|  |  |
|  | # model predictions are stored on GPU. So, push it to CPU |
|  | preds=preds.detach().cpu().numpy() |
|  |  |
|  | # append the model predictions |
|  | total\_preds.append(preds) |
|  |  |
|  | # compute the training loss of the epoch |
|  | avg\_loss = total\_loss / len(train\_dataloader) |
|  |  |
|  | # predictions are in the form of (no. of batches, size of batch, no. of classes). |
|  | # reshape the predictions in form of (number of samples, no. of classes) |
|  | total\_preds = np.concatenate(total\_preds, axis=0) |
|  |  |
|  | #returns the loss and predictions |
|  | return avg\_loss, total\_preds |

[view raw](https://gist.github.com/prateekjoshi565/d25b41492fedae8686660d3a5320a0d6/raw/c9b8bf6a8a60e992be246c9fa13b7f18fd6c0ee2/train_bert.py)[train\_bert.py](https://gist.github.com/prateekjoshi565/d25b41492fedae8686660d3a5320a0d6#file-train_bert-py)hosted with  by [GitHub](https://github.com/)

We will use the following function to evaluate the model. It will use the validation set data.

|  |  |
| --- | --- |
|  | # function for evaluating the model |
|  | def evaluate(): |
|  |  |
|  | print("\nEvaluating...") |
|  |  |
|  | # deactivate dropout layers |
|  | model.eval() |
|  |  |
|  | total\_loss, total\_accuracy = 0, 0 |
|  |  |
|  | # empty list to save the model predictions |
|  | total\_preds = [] |
|  |  |
|  | # iterate over batches |
|  | for step,batch in enumerate(val\_dataloader): |
|  |  |
|  | # Progress update every 50 batches. |
|  | if step % 50 == 0 and not step == 0: |
|  |  |
|  | # Calculate elapsed time in minutes. |
|  | elapsed = format\_time(time.time() - t0) |
|  |  |
|  | # Report progress. |
|  | print(' Batch {:>5,} of {:>5,}.'.format(step, len(val\_dataloader))) |
|  |  |
|  | # push the batch to gpu |
|  | batch = [t.to(device) for t in batch] |
|  |  |
|  | sent\_id, mask, labels = batch |
|  |  |
|  | # deactivate autograd |
|  | with torch.no\_grad(): |
|  |  |
|  | # model predictions |
|  | preds = model(sent\_id, mask) |
|  |  |
|  | # compute the validation loss between actual and predicted values |
|  | loss = cross\_entropy(preds,labels) |
|  |  |
|  | total\_loss = total\_loss + loss.item() |
|  |  |
|  | preds = preds.detach().cpu().numpy() |
|  |  |
|  | total\_preds.append(preds) |
|  |  |
|  | # compute the validation loss of the epoch |
|  | avg\_loss = total\_loss / len(val\_dataloader) |
|  |  |
|  | # reshape the predictions in form of (number of samples, no. of classes) |
|  | total\_preds = np.concatenate(total\_preds, axis=0) |
|  |  |
|  | return avg\_loss, total\_preds |

[view raw](https://gist.github.com/prateekjoshi565/30a61a1cf77ba5c51fc663b88c9b5cee/raw/b178fa724b564a8dc117bac88ad8208acada4ae8/evaluate_bert.py)[evaluate\_bert.py](https://gist.github.com/prateekjoshi565/30a61a1cf77ba5c51fc663b88c9b5cee#file-evaluate_bert-py)hosted with  by [GitHub](https://github.com/)

Now we will finally start fine-tuning of the model.

|  |  |
| --- | --- |
|  | # set initial loss to infinite |
|  | best\_valid\_loss = float('inf') |
|  |  |
|  | # empty lists to store training and validation loss of each epoch |
|  | train\_losses=[] |
|  | valid\_losses=[] |
|  |  |
|  | #for each epoch |
|  | for epoch in range(epochs): |
|  |  |
|  | print('\n Epoch {:} / {:}'.format(epoch + 1, epochs)) |
|  |  |
|  | #train model |
|  | train\_loss, \_ = train() |
|  |  |
|  | #evaluate model |
|  | valid\_loss, \_ = evaluate() |
|  |  |
|  | #save the best model |
|  | if valid\_loss < best\_valid\_loss: |
|  | best\_valid\_loss = valid\_loss |
|  | torch.save(model.state\_dict(), 'saved\_weights.pt') |
|  |  |
|  | # append training and validation loss |
|  | train\_losses.append(train\_loss) |
|  | valid\_losses.append(valid\_loss) |
|  |  |
|  | print(f'\nTraining Loss: {train\_loss:.3f}') |
|  | print(f'Validation Loss: {valid\_loss:.3f}') |

[view raw](https://gist.github.com/prateekjoshi565/8e7a15125ac8f2402577aa1935000258/raw/45c5efabf00a13af2762ee7e392259ffb1409e04/start_training_bert.py)[start\_training\_bert.py](https://gist.github.com/prateekjoshi565/8e7a15125ac8f2402577aa1935000258#file-start_training_bert-py)hosted with  by [GitHub](https://github.com/)

**Output:**

Training Loss: 0.592

Validation Loss: 0.567

Epoch 5 / 10

Batch 50 of 122.

Batch 100 of 122.

Evaluating...

Training Loss: 0.566

Validation Loss: 0.543

Epoch 6 / 10

Batch 50 of 122.

Batch 100 of 122.

Evaluating...

Training Loss: 0.552

Validation Loss: 0.525

Epoch 7 / 10

Batch 50 of 122.

Batch 100 of 122.

Evaluating...

Training Loss: 0.525

Validation Loss: 0.498

Epoch 8 / 10

Batch 50 of 122.

Batch 100 of 122.

Evaluating...

Training Loss: 0.507

Validation Loss: 0.477

Epoch 9 / 10

Batch 50 of 122.

Batch 100 of 122.

Evaluating...

Training Loss: 0.488

Validation Loss: 0.461

Epoch 10 / 10

Batch 50 of 122.

Batch 100 of 122.

Evaluating...

Training Loss: 0.474

Validation Loss: 0.454

You can see that the validation loss is still decreasing at the end of the 10th epoch. So, you may try a higher number of epochs. Now let’s see how well it performs on the test dataset.

Make Predictions

To make predictions, we will first of all load the best model weights which were saved during the training process.

|  |  |
| --- | --- |
|  | #load weights of best model |
|  | path = 'saved\_weights.pt' |
|  | model.load\_state\_dict(torch.load(path)) |

[view raw](https://gist.github.com/prateekjoshi565/87a958e0bc42e175408fbabb49cd8842/raw/2c99153cb92390fd9e7736584dec7c143e6d93b3/load_weights_bert.py)[load\_weights\_bert.py](https://gist.github.com/prateekjoshi565/87a958e0bc42e175408fbabb49cd8842#file-load_weights_bert-py)hosted with  by [GitHub](https://github.com/)

Once the weights are loaded, we can use the fine-tuned model to make predictions on the test set.

|  |  |
| --- | --- |
|  | # get predictions for test data |
|  | with torch.no\_grad(): |
|  | preds = model(test\_seq.to(device), test\_mask.to(device)) |
|  | preds = preds.detach().cpu().numpy() |

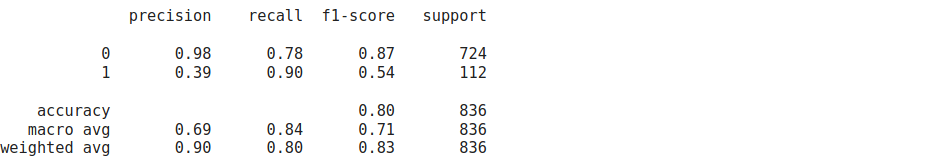
[view raw](https://gist.github.com/prateekjoshi565/ed8efd3f1210436c4733ab0a7db4d85f/raw/43785ea9f9488e81ad9560c2b5c6e0dfe5da0e67/prediction_bert.py)[prediction\_bert.py](https://gist.github.com/prateekjoshi565/ed8efd3f1210436c4733ab0a7db4d85f#file-prediction_bert-py)hosted with  by [GitHub](https://github.com/)

Let’s check out the model’s performance.

|  |  |
| --- | --- |
|  | preds = np.argmax(preds, axis = 1) |
|  | print(classification\_report(test\_y, preds)) |

[view raw](https://gist.github.com/prateekjoshi565/e579818bd95b493d5cd4b8e0280fc1ab/raw/4ff7b176a7b27b03edf41f9d2e7253d4129db79b/performance_bert.py)[performance\_bert.py](https://gist.github.com/prateekjoshi565/e579818bd95b493d5cd4b8e0280fc1ab#file-performance_bert-py)hosted with  by [GitHub](https://github.com/)

**Output:**



Both recall and precision for class 1 are quite high which means that the model predicts this class pretty well. However, our objective was to detect spam messages, so misclassifying class 1 (spam) samples is a bigger concern than misclassifying class 0 samples. If you look at the recall for class 1, it is 0.90 which means that the model was able to correctly classify 90% of the spam messages. However, precision is a bit on the lower side for class 1. It means that the model misclassifies some of the class 0 messages (not spam) as spam.

**Link to [Colab Notebook](https://github.com/prateekjoshi565/Fine-Tuning-BERT/blob/master/Fine_Tuning_BERT_for_Spam_Classification.ipynb" \t "_blank)**

End Notes

To summarize,  in this article, we fine-tuned a pre-trained BERT model to perform text classification on a very small dataset. I urge you to fine-tune BERT on a different dataset and see how it performs. You can even perform multiclass or multi-label classification with the help of BERT. In addition to that, you can even train the entire BERT architecture as well if you have a bigger dataset.

In case you are looking for a roadmap to becoming an expert in NLP read the following article-

* [A Comprehensive Learning Path to Understand and Master NLP in 2020](https://www.analyticsvidhya.com/blog/2020/01/learning-path-nlp-2020/?utm_source=blog&utm_medium=fine_tune_BERT)

You may use the comment section in case you have any thoughts to share or have any doubts.

[BERT](https://www.analyticsvidhya.com/blog/tag/bert/)[fine tuning](https://www.analyticsvidhya.com/blog/tag/fine-tuning/)[message spam classifier](https://www.analyticsvidhya.com/blog/tag/message-spam-classifier/)[Natural language processing](https://www.analyticsvidhya.com/blog/tag/natural-language-processing/)[pytorch](https://www.analyticsvidhya.com/blog/tag/pytorch/)[text classification](https://www.analyticsvidhya.com/blog/tag/text-classification/)[transfer learning](https://www.analyticsvidhya.com/blog/tag/transfer-learning/)[transformers](https://www.analyticsvidhya.com/blog/tag/transformers/)

[](https://www.analyticsvidhya.com/blog/author/pjoshi15/)

[Prateek Joshi](https://www.analyticsvidhya.com/blog/author/pjoshi15/)20 Jun 2022

Data Scientist at Analytics Vidhya with multidisciplinary academic background. Experienced in machine learning, NLP, graphs & networks. Passionate about learning and applying data science to solve real world problems.

[Advanced](https://www.analyticsvidhya.com/blog/category/advanced/)[Deep Learning](https://www.analyticsvidhya.com/blog/category/deep-learning/)[NLP](https://www.analyticsvidhya.com/blog/category/nlp/)[Project](https://www.analyticsvidhya.com/blog/category/project/)[Python](https://www.analyticsvidhya.com/blog/category/python-2/)

Responses From Readers

Top of Form



Submit reply

Bottom of Form



Dinesh Chauhan21 Jul, 2020

Hi, Thanks a lot for very detailed explanation. I have some doubt around precision/recall inference you made. You mentioned recall for class 1 is high (0.90) so model will correctly identify spam 90% of time , however doesn't that metric should be accuracy ? Also you mentioned precision is on lower side (0.39) which means we would misclassify non-spam as spam , however how do you interpret high precision for class 0 in same context ? if possible please expand little more on precision/recall for both spam (1) & ham(class 0). Thanks & Regards

 1Show 1 reply



John ODonovan23 Jul, 2020

Hi Dinesh, Nice tutorial, thanks! I got similar results to you. ( I used my own GPU box instead of Colab) precision recall f1-score support 0 0.97 0.87 0.91 724 1 0.48 0.81 0.61 112 accuracy 0.86 836 macro avg 0.73 0.84 0.76 836 weighted avg 0.90 0.86 0.87 836



Tegene28 Jul, 2020

Hi Prateek ! Thank you very much for your hands on explanation on such comple concept. I am enjoying your tutorials as well. Currently, am working a project on transfer learning for netx word prediction for one of my local languages. the language uses latin letters(english letters) and it uses a very long suffixes when it is inflected, it is also a low resource language. so which method would you recommend? if you can please put the steps for me. thanks in advance

 1Show 1 reply



ali07 Aug, 2020

Hi Can we save the model and then use it for prediction? How do we do?

 1Show 1 reply



Jim13 Aug, 2020

Hi Dinesh, I have a dataset with 3 classes and it is very imbalanced. However when I run the last piece of code where start fine-tuning of the model I got this error: RuntimeError: weight tensor should be defined either for all 2 classes or no classes but got weight tensor of shape: [3] at /pytorch/aten/src/THCUNN/generic/ClassNLLCriterion.cu:47 Can you help me?

 1Show 1 reply



Susanna20 Aug, 2020

Hi Prateek! Thank you so much for the explanation! It's very helpful! However, in the fine-tuning part, I'm getting the following error. RuntimeError: Expected object of scalar type Long but got scalar type Float for argument #2 'target' in call to \_thnn\_nll\_loss\_forward Would you happen to know how to fix this? I would be very grateful!

 1Show 1 reply



Stephen20 Aug, 2020

Hi Prateek, Great work! Can we specifically find out which text in the datasets was miss-classified? And also is there a way to add a few more lines so the code can take in sentences and then predict if it is spam or ham?

 1Show 1 reply



aly mostafa28 Aug, 2020

why did you choose '768' in self.fc1 = nn.Linear(768,512) in class BERT\_Arch(nn.Module): thanks in advance



Aman Kabra06 Sep, 2020

Thanks for the article, Prateek. Much appreciated! The following error is thrown while running the code: .... Batch 3,800 of 3,906. Batch 3,850 of 3,906. Batch 3,900 of 3,906. Evaluating... --------------------------------------------------------------------------- NameError Traceback (most recent call last) in () 15 16 #evaluate model ---> 17 valid\_loss, \_ = evaluate() 18 19 #save the best model in evaluate() 19 20 # Calculate elapsed time in minutes. ---> 21 elapsed = format\_time(time.time() - t0) 22 23 # Report progress. NameError: name 'format\_time' is not defined Could you help out?



yasmyne28 Sep, 2020

Hi, thank s for this great tutorial I want to apply this for a multi label text classification problem. My labels are of this format tensor([[0, 0, 0, ..., 0, 0, 0], [0, 0, 0, ..., 0, 0, 0], [0, 0, 0, ..., 0, 0, 0], ..., [0, 0, 0, ..., 0, 0, 0], [0, 0, 0, ..., 0, 0, 0], [0, 0, 0, ..., 0, 0, 0]]) I changed the softmax function in the bert model by the sigmoid function but when I tried to train the model I got this error multi-target not supported at /pytorch/aten/src/THCUNN/generic/ClassNLLCriterion.cu:18 Could u help plz thank u



Murtuza Dahodwala30 Sep, 2020

Hello Prateek, Very nice article to read!! However i want to use BERT to map correct word in my vocabulary with it's wrong spelling. I have a dataset with 20lakh food item names and another dataset with all the correct words. For eg: IN: "Cheeze Piza 2" OUT: "Cheese Pizza" I am aware that i can use BERT or GPT for this task but I'm not sure where to start. Can you please guide?



Ken03 Oct, 2020

Hi Thanks for you great explanations. I just wonder whether we can build knowledge graph with BERT from scratch? Or BERT is only good at the part of text classification?



Fareen09 Oct, 2020

I want to generate sentence embeddings using this fine-tuned pytorch model. How can I do this?



Mehul10 Oct, 2020

Hi Prateek, Thanks a lot for this great blog. I just have one question. How do we came to know that we have to unpack the output of the bert in forward pass? \_, cls\_hs = self.bert(sent\_id, attention\_mask=mask) Where can I find this detail in the Transformer documentation? Thanks again. : )



Peter24 Oct, 2020

Thank you for the tutorial, I had success training 500K tweets for Sentiment Analysis (positive/negative), when loading the model to evaluate, section "# get predictions for test data" i am getting the following error: Traceback (most recent call last): File "C:/Users/Botros/PycharmProjects/mlwork/test.py", line 201, in preds = model(test\_seq.to(device), test\_mask.to(device)) File "C:\Users\Botros\PycharmProjects\mlwork\venv\lib\site-packages\torch\nn\modules\module.py", line 722, in \_call\_impl result = self.forward(\*input, \*\*kwargs) File "C:/Users/Botros/PycharmProjects/mlwork/test.py", line 166, in forward \_, cls\_hs = self.bert(sent\_id, attention\_mask=mask) File "C:\Users\Botros\PycharmProjects\mlwork\venv\lib\site-packages\torch\nn\modules\module.py", line 722, in \_call\_impl result = self.forward(\*input, \*\*kwargs) File "C:\Users\Botros\PycharmProjects\mlwork\venv\lib\site-packages\transformers\modeling\_bert.py", line 830, in forward embedding\_output = self.embeddings( File "C:\Users\Botros\PycharmProjects\mlwork\venv\lib\site-packages\torch\nn\modules\module.py", line 722, in \_call\_impl result = self.forward(\*input, \*\*kwargs) File "C:\Users\Botros\PycharmProjects\mlwork\venv\lib\site-packages\transformers\modeling\_bert.py", line 198, in forward position\_embeddings = self.position\_embeddings(position\_ids) File "C:\Users\Botros\PycharmProjects\mlwork\venv\lib\site-packages\torch\nn\modules\module.py", line 722, in \_call\_impl result = self.forward(\*input, \*\*kwargs) File "C:\Users\Botros\PycharmProjects\mlwork\venv\lib\site-packages\torch\nn\modules\sparse.py", line 124, in forward return F.embedding( File "C:\Users\Botros\PycharmProjects\mlwork\venv\lib\site-packages\torch\nn\functional.py", line 1814, in embedding return torch.embedding(weight, input, padding\_idx, scale\_grad\_by\_freq, sparse) IndexError: index out of range in self i was able to create another model with 100k and run it fine, can you help?



Francesco05 Nov, 2020

Hello, very nice tutorial. in my dataset i have 5 class, and i get this error: "RuntimeError: weight tensor should be defined either for all 2 classes or no classes but got weight tensor of shape: [5] at /pytorch/aten/src/THCUNN/generic/ClassNLLCriterion.cu:44" I have to change the number of output units from 2 to 5 in the model architecture, but exatly where in the code? And how?



Mertay21 Nov, 2020

Hey Prateek, Thanks for the tutorial it was great. I have a question. How do I make predictions using the model. Not necessarily on the test but I want to come up with my input and see the results. Can you help me to get one example? Thank you.



Shreyash01 Dec, 2020

Hi! Really nice explanation. My code was running perfectly well yesterday and today it seems to give an error out of nowhere. I even tried your exact same code with the same dataset still it is giving the error. /usr/local/lib/python3.6/dist-packages/torch/nn/modules/linear.py in forward(self, input) 91 92 def forward(self, input: Tensor) -> Tensor: ---> 93 return F.linear(input, self.weight, self.bias) 94 95 def extra\_repr(self) -> str: /usr/local/lib/python3.6/dist-packages/torch/nn/functional.py in linear(input, weight, bias) 1686 if any([type(t) is not Tensor for t in tens\_ops]) and has\_torch\_function(tens\_ops): 1687 return handle\_torch\_function(linear, tens\_ops, input, weight, bias=bias) -> 1688 if input.dim() == 2 and bias is not None: 1689 # fused op is marginally faster 1690 ret = torch.addmm(bias, input, weight.t()) AttributeError: 'str' object has no attribute 'dim' Please let me know where is it going wrong?



Kieran10 Dec, 2020

Hello Prateek, thank you very much for this amazing article was really informative and you were able to explain such a complex thing in an easy way. Keep up your good work. cheers!!!



paulo gamalho16 Dec, 2020

Thanks, Prateek! Nice work! I have two questions: How can add my own test text? and how can I print a new csv file provided with the labels predicted by the system? Thanks!



Fatima Zulfiqar06 Jan, 2021

Hi I have successfully train the model however I am getting following error as: RuntimeError: CUDA out of memory. Tried to allocate 5.04 GiB (GPU 0; 14.73 GiB total capacity; 9.24 GiB already allocated; 1.88 GiB free; 11.80 GiB reserved in total by PyTorch) Can you help me out? I am using Google Colab



Abhishek08 Feb, 2021

Hi, Prateek. This is a beautiful explaination. Can we use it for multi label classification? If yes, what are the changes that we need to make?



Maram11 Feb, 2021

Hi Dinesh, I want to make the BERT model predict the classes for multi-classes classification, for a giving text for example for "Wish you a great day" the model will give me prediction of the class "Positive" I tried several ways to do it but none of them working "), do I have to save my fine-tuned model before making prediction? Can you kindly give me advice, hint of the way to do it. I would be very grateful!



Simone13 Feb, 2021

Hi, Thanks fro sharing, this is a great article! One question, would still be relevant utilising `class\_weights = compute\_class\_weight('balanced', np.unique(train\_labels), train\_labels)` in your cross entropy loss function even if you have more than two classes to predict?



Johnny Stone02 Dec, 2021

How would one go about to make a prediction for a custom text using the fine tuned model?



Nemo06 Jan, 2022

RuntimeError: "nll\_loss\_forward\_reduce\_cuda\_kernel\_2d" not implemented for 'Long' Hi, I cannot fix this this problem. Can you please help me?



Alice23 Feb, 2022

how can we save this model using save.pretrained(dir)? I get BERT\_Arch object has no attribute 'save\_pretrained'



Rifat16 Nov, 2023

Assalomu alaykum , Prateek Joshi, Thanks you for all , it is great instruction. I have a question that I can use BERT model for other language such as Uzbek.?

[Write, Shine, Succeed](https://datahack.analyticsvidhya.com/blogathon/)

[Write, captivate, and earn accolades and rewards for your work](https://datahack.analyticsvidhya.com/blogathon/)

* [Reach a Global Audience](https://www.analyticsvidhya.com/blog/2020/07/transfer-learning-for-nlp-fine-tuning-bert-for-text-classification/)
* [Get Expert Feedback](https://www.analyticsvidhya.com/blog/2020/07/transfer-learning-for-nlp-fine-tuning-bert-for-text-classification/)
* [Build Your Brand & Audience](https://www.analyticsvidhya.com/blog/2020/07/transfer-learning-for-nlp-fine-tuning-bert-for-text-classification/)
* [Cash In on Your Knowledge](https://www.analyticsvidhya.com/blog/2020/07/transfer-learning-for-nlp-fine-tuning-bert-for-text-classification/)
* [Join a Thriving Community](https://www.analyticsvidhya.com/blog/2020/07/transfer-learning-for-nlp-fine-tuning-bert-for-text-classification/)
* [Level Up Your Data Science Game](https://www.analyticsvidhya.com/blog/2020/07/transfer-learning-for-nlp-fine-tuning-bert-for-text-classification/)

[[](https://www.analyticsvidhya.com/blog/author/rahul105/)img](https://www.analyticsvidhya.com/blog/author/rahul105/)

[Rahul Shah](https://www.analyticsvidhya.com/blog/author/rahul105/)

[27](https://www.analyticsvidhya.com/blog/author/rahul105/)

[[](https://www.analyticsvidhya.com/blog/author/sion/)img](https://www.analyticsvidhya.com/blog/author/sion/)

[Sion Chakrabarti](https://www.analyticsvidhya.com/blog/author/sion/)

[16](https://www.analyticsvidhya.com/blog/author/sion/)

[[](https://www.analyticsvidhya.com/blog/author/chirag676/)img](https://www.analyticsvidhya.com/blog/author/chirag676/)

[CHIRAG GOYAL](https://www.analyticsvidhya.com/blog/author/chirag676/)

[87](https://www.analyticsvidhya.com/blog/author/chirag676/)

[[](https://www.analyticsvidhya.com/blog/author/barney6/)img](https://www.analyticsvidhya.com/blog/author/barney6/)

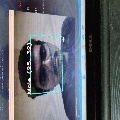
[Barney Darlington](https://www.analyticsvidhya.com/blog/author/barney6/)

[5](https://www.analyticsvidhya.com/blog/author/barney6/)

[[](https://www.analyticsvidhya.com/blog/author/suvojit3/)img](https://www.analyticsvidhya.com/blog/author/suvojit3/)

[Suvojit Hore](https://www.analyticsvidhya.com/blog/author/suvojit3/)

[9](https://www.analyticsvidhya.com/blog/author/suvojit3/)

[[](https://www.analyticsvidhya.com/blog/author/arnab1408/)img](https://www.analyticsvidhya.com/blog/author/arnab1408/)

[Arnab Mondal](https://www.analyticsvidhya.com/blog/author/arnab1408/)

[15](https://www.analyticsvidhya.com/blog/author/arnab1408/)

[[](https://www.analyticsvidhya.com/blog/author/prateekmaj21/)img](https://www.analyticsvidhya.com/blog/author/prateekmaj21/)

[Prateek Majumder](https://www.analyticsvidhya.com/blog/author/prateekmaj21/)

[68](https://www.analyticsvidhya.com/blog/author/prateekmaj21/)

* 34

**Company**

* [About Us](https://www.analyticsvidhya.com/about/)
* [Contact Us](https://www.analyticsvidhya.com/contact/)
* [Careers](https://www.analyticsvidhya.com/careers/)

**Discover**

* [Blogs](https://www.analyticsvidhya.com/blog/?utm_source=blog&utm_medium=footer)
* [Expert session](https://datahack.analyticsvidhya.com/datahour/?utm_source=blog&utm_medium=footer)
* [Podcasts](https://community.analyticsvidhya.com/c/leading-with-data?utm_source=blog&utm_medium=footer)
* [Comprehensive Guides](https://www.analyticsvidhya.com/category/guide/?utm_source=blog&utm_medium=footer)

**Learn**

* [Free courses](https://courses.analyticsvidhya.com/pages/all-free-courses?utm_source=blog&utm_medium=footer)
* [Learning path](http://courses.analyticsvidhya.com/courses/a-comprehensive-learning-path-to-become-a-data-scientist-in-2023?utm_source=blog&utm_medium=footer)
* [BlackBelt program](https://blackbelt.analyticsvidhya.com/plus?utm_source=blog&utm_medium=footer)
* [Gen AI](https://pinnacle.analyticsvidhya.com/bundles/gen-ai?utm_source=blog&utm_medium=footer)

**Engage**

* [Community](https://community.analyticsvidhya.com/?utm_source=blog&utm_medium=footer)
* [Hackathons](https://datahack.analyticsvidhya.com/?utm_source=blog&utm_medium=footer)
* [Events](https://datahack.analyticsvidhya.com/datahour/?utm_source=blog&utm_medium=footer)
* [Daily challenges](https://www.analyticsvidhya.com/coming-soon?utm_source=blog&utm_medium=footer)

**Contribute**

* [Contribute & win](https://datahack.analyticsvidhya.com/blogathon/?utm_source=blog&utm_medium=foote)
* [Become a speaker](https://www.analyticsvidhya.com/coming-soon?utm_source=blog&utm_medium=footer)
* [Become a mentor](https://www.analyticsvidhya.com/coming-soon?utm_source=blog&utm_medium=footer)
* [Become an instructor](https://www.analyticsvidhya.com/coming-soon?utm_source=blog&utm_medium=footer)

**Enterprise**

* [Our offerings](https://enterprise.analyticsvidhya.com/?utm_source=blog&utm_medium=foote)
* [Case studies](https://www.analyticsvidhya.com/coming-soon?utm_source=blog&utm_medium=footer)
* [Industry report](https://www.analyticsvidhya.com/coming-soon?utm_source=blog&utm_medium=footer)
* [quexto.ai](https://www.analyticsvidhya.com/coming-soon?utm_source=blog&utm_medium=footer)

Download App

[Terms & conditions](https://www.analyticsvidhya.com/terms/)  [Refund Policy](https://www.analyticsvidhya.com/refund-policy/)  [Privacy Policy](https://www.analyticsvidhya.com/privacy-policy/)  [Cookies Policy](https://www.analyticsvidhya.com/blog/2020/07/transfer-learning-for-nlp-fine-tuning-bert-for-text-classification/) © Analytics Vidhya 2023.All rights reserved.

We use cookies on Analytics Vidhya websites to deliver our services, analyze web traffic, and improve your experience on the site. By using Analytics Vidhya, you agree to our [Privacy Policy](https://www.analyticsvidhya.com/privacy-policy/) and [Terms of Use](https://www.analyticsvidhya.com/terms/).Accept