Explore The Power of Transfer Learning: BERT for Text Classification

Haoran Wang
University of Oregon
`hwang8@cs.uoregon.edu`

Abstract

BERT is a language model that was created and published in 2018 by Jacob Delvin and Ming-Wei Chang from Google [3]. It has taken the NLP landscape by a storm. BERT has been able to achieve state-of-the-art results in a wide variety of NLP tasks. Many researchers have been working on improving BERT in the past two years. As of now, we have many variations of the original BERT, such as ALBERT, RoBERTa, DistillBERT, and XLM-RoBERTa, etc. Transfer learning on those models have proven to achieve really good results on many different NLP tasks. In this paper, I have found that transfer learning on BERT and XLM-RoBERTa achieve really good result on text classification in comparison to building a simple model like CNN from scratch. My best performed model is a fine-tuned XLM-RoBERTa cross lingual model, which achieves 0.9459 on the test set. As a comparison, the current number one entry on Kaggle is 0.9555. In order to do training on a very large model like BERT, we will need the help with Google's hardware TPU (tensor processing units). I have found that TPU is about four times faster than GPU.

1 Introduction

For this project, I am competing in Jigsaw Multi-lingual Toxic Comment Classification competition hosted by Jigsaw and Google. Jigsaw, formerly known as Google ideas is a unit within Google that tackles emerging threats from online activities to make internet a better place for all. With the increasing usage of social media, cyber bullying has become a real threat, especially to teenagers. Therefore, Jigsaw and Google have dedicated $50,000 to host this Kaggle competition to explore how different NLP techniques can be used to classify toxic comments.

Text classification is like the "hello world" program in NLP. To make it interesting, I will apply transfer learning on pre-trained BERT as well XLM-RoBERTa and compare the result against using BERT as word embeddings and feed them into a CNN to do text classification.

In addition to fine-tuning BERT models, I will be using TPUs (tensor processing units) for training. TPUs are developed by Google and they only support Tensorflow natively. To enable PyTorch on TPU, I will need PyTorch/XLA package.

2 Background

This section will review the architecture of BERT and XLM-RoBERTa, as well as the concept of Transfer Learning.

2.1 BERT

BERT, short for Bidirectional Encoder Representation from Transformers has two pre-trained models: BERT base and BERT large. BERT base has 12 layers, and around 110 million parameters. BERT large has 24 layers, and around 340 million parameters [3]. It is a very large model indeed. Therefore, I will be using BERT base for this project because of limited resources.

To understand BERT, we need to first understand how Transformers work. BERT is basically layers of bidirectional Transformer encoders.

Transformers consists a number of encoders and decoders. Each encoder and decoder has two layers: a multi-head attention layer and a fully connected feed forward neural network.

Multi-head attention is to compute self-attention multiple times in parallel and independently.

The outputs are then concatenated and linearly transformed.

Now, let us examine what self-attention is. "Self-attention, sometimes called intra-attention, is an attention mechanism relating different positions of a single sequence in order to compute a representation of the sequence. [7]" Therefore, it helps the encoder to look at other words in the input sentence when encode each word.

To calculate self-attention, the encoder will create a query vector, a key vector, and a value vector for the embeddings of different words in the sequence. Next, we will calculate the dot product of the query vector with the key vector for each word of the input sequence against the other words in the sequence. Then, we will divide the dot product by the square root of the dimension of the key vectors, following by a softmax function. Finally we multiply each value vector by the softmax score and sum up the weighted value vectors.

Transformers also add positional encoding when encoding each word. BERT's structure makes BERT different from context-free word embeddings such as word2vec and GloVe, which just generate a word embedding (vector) for each word regardless of its context. Instead, BERT can generate different word embeddings based on its context. This makes BERT a very powerful language model.
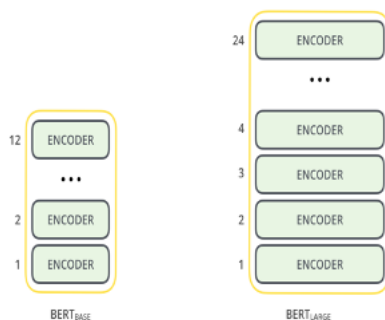

Figure 1: BERT base and BERT large [3]

2.2 Transfer Learning

Because of how well BERT models performed, transfer learning has come in handy when using BERT to solve different NLP tasks. Instead of
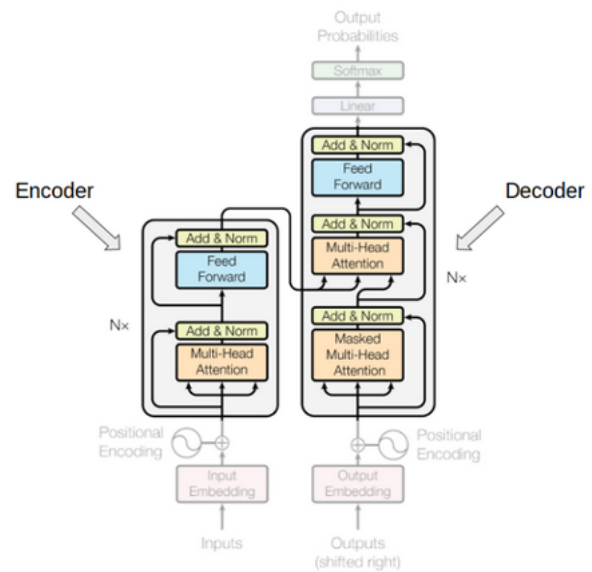

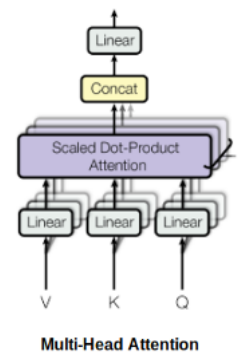Figure 2: Transformer Architecture [7]

Figure 3: Multi-Head Attention [7]

building a model from scratch, we can use a pre-trained model like BERT to solve our own spe-cific problem with a little bit fine-tuning to adapt the model to our problem. This saves a lot of resources because our model does not need to learn every parameter from scratch. In the case of pre-trained BERT base, we do not need to learn the first 12 layers of parameters. We can just add a layer on top of them to solve specific problems.

So, what tasks are BERT trained on? BERT is pre-trained on two tasks: Masked Language Modeling (MLM) and Next Sentence Prediction (NSP). For MLM, before feeding text sequences into BERT, some words in the sequence are masked with [MASK] token. BERT then tries to predict the original value of the masked words based on the context of the sequence. For NSP, BERT receives pairs of text sequences as input,

and BERT tries to predict if the second sentence is the subsequent sentence in the original document. When BERT model was trained, the two tasks were trained together, and the goal was to minimize the combined loss function of both tasks.

2.3 XLM-RoBERTa

BERT is trained entirely on English data. Many researchers who are working on non-English data have then come up with language specific models such as Finish BERT, French BERT and German BERT. However, research on a multilingual model has never stopped. In November, 2019, Facebook AI team resleased XLM-RoBERT [2], an update to their XLM-100 model. They used RoBERTa [5] model (A Robustly Optimized BERT Pretraining Approach) to improve XLM-100 and hence produce XLM-RoBERTa.

XLM-RoBERTa is trained on 100 languages, total of 2.5TB of text. The pre-trained XLM-RoBERTa has 550 million parameters, compared to 355 million of RoBERTa.

2.4 TPU

Tensor Processing Unit is developed by Google to specialize in deep learning. Models that pre-viously take days to train on GPU can be done in hours on TPU. However, in order to use PyTorch on TPU, we have to use a package called XLA, and there are no stable releases yet.

3 Data

This Kaggle competition has provided training set, validation set, and test set. The training set's comments are entirely in English. The test set's comments are composed of multiple non-English languages. I have combined the training set from previous competition to make a large training set of 2.1 million comments. Since my resource is limited, I only used the first 200,000 data as my training set.

I will use BERT and XLM-RoBERTa pre-trained tokenizer to tokenize and pad the training data. As for the test set, I will use this test set shared by one of the competitors [1] that translated the entire test set into English. When I evaluate my XLM-RoBERTa model, I can do it directly without translating the test set because it is a cross-lingual model.

Since this is a kernel only competition, inferencing is done separately on Kaggle notebook.

4 Models

This section investigates three different models.

4.1 BERT for Text Classification Directly

To adapt pre-trained BERT base model to do text classification, we just need to add a classifier layer on top of the 12 BERT layers. Then, we can fine-tune all the hyper-parameters to improve performance.

We will add drop out for BERT base to prevent overfitting. BERT base model produces two outputs: last hidden state and pooler output. After BERT base, we will do a mean pooling and max pooling on the last hidden state from BERT base. Then, we will concatenate them and send it to a linear (output) layer.

This linear (output) layer has size of (768*2,1). 768 is the hidden size of BERT base, which means it has 768 output features, we times it by 2 because we did both max pooling and mean pooling before. Since it is a binary classification problem, we put 1 to make the output value between 0 and 1.

We will use binary cross entropy with logits loss as our loss function, and Adam with decoupled weight decay[6] as our optimizer.

4.2 BERT as word Embeddings + KimCNN

We will use BERT to tokenize and pad words into word embeddings. Then we will feed those word embeddings to a CNN introduced in this paper [4] by Yoon Kim from New York Univer-sity in 2014.

Here are the steps of how KimCNN works:

(1) After we get word embeddings from BERT, which are of size [192, 768]. The max sequence length is 192 and the embedding size is 768.
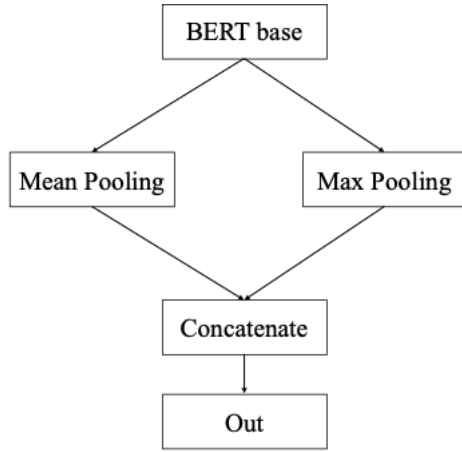
Figure 4: BERT for Text Classification

Figure 5: KimCNN [4]

(2) The model applies convolution on embeddings that uses three different filters of size [2, 768], [3, 768], and [4, 768]. These filters corre-sponds to capturing the bi-gram, tri-gram, and 4-gram relations.

(3) Apply ReLU to add ability to model nonline-ar problems.

(4) Apply 1-max pooling to down-sample the input representation and to help prevent overfit-ting.

(5) Concatenate vectors after 1-max pooling to a single vector.

(6) A dropout layer to help overfitting.

(7) Apply sigmoid function.

4.3 Cross-lingual model: XLM-RoBERTa

This is very similar to BERT. The pre-trained XLM-RoBERTa has two outputs, last hidden state and pooler output. The pooler output is feed into a linear layer of size (1024, 1). 1024 is the hidden size of XLM-RoBERTa.

5 Experiments

I have fine-tuned BERT and XLM-RoBERTa to compare against CNN. Also, I have trained BERT both on GPU and TPU to compare the performances.
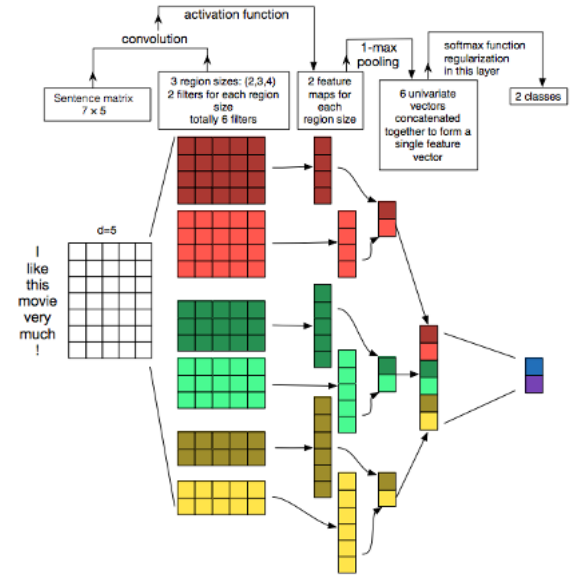
5.1 Fine-tune BERT on GPU

For fine-tuning, we will tune these hyperparameters: learning rate, batch size and number of training epochs.

Since our training dataset is rather large and training on GPU takes a long time, we will tune the hyperparameters on the first 50,000 data.

The BERT paper had a list of hyperparameters that generally work well across all tasks, which we can use as our starting point to fine-tune [3]: Batch size: 16, 32; Learning rate: 5e-5, 3e-5, 2e-5; Number of epochs: 2, 3, 4.

Learning Rate: Learning rate is perhaps the most critical one of all hyperparameters. To tune learning, rate we fix the training batch size to be 64 and the validation batch size to be 8. Training on one epoch. The training and validation loss plot every 100 batches to make the plot less clustered. We use ROC-AUC score to measure how well our model performs on validation set.

Learning Rate	ROC-AUC
5e-3	0.53
5e-5	0.82
2e-5	0.84
5e-6	0.79

Table 1: Table 1. Learning Rate vs. ROC-AUC
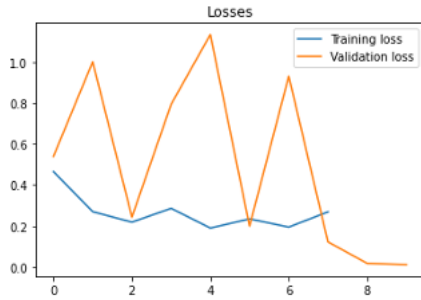
Figure 6: Figure 6. lr = 5e-3
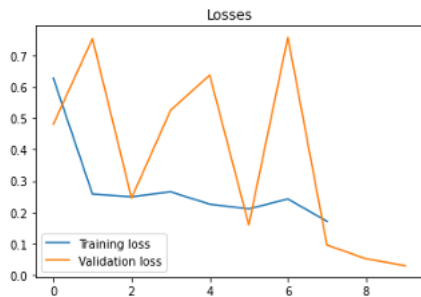

Figure 7: Figure 7. lr = 5e-5
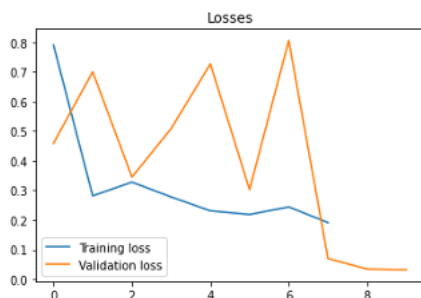

Figure 8: Figure 8. lr = 2e-5

Figure 9: Figure 9. lr = 5e-6

As we can see, with an aggressive learning rate of 5e-3, the model did poorly. We find that ideal learning rate should be around 2e-5, which is suggested by the paper.

Batch Size: We will set learning rate to be 1e-5, training on 3 epochs to tune batch sizes.

Generally, too large of a batch size will lead to poor generalization, while too small of a batch size will not guarantee to converge to the global optima. It will bounce around the global optima instead. [10]

I have found that the max training batch size is 64. I tried training batch size of 128 and 256, which will blow up GPU memory.

Training	Validation	ROC-AUC
16	2	0.79
32	4	0.81
64	8	0.82

Table 2: Table 2. Batch Size vs. ROC-AUC

As we can see training batch size of 64 is ideal for this model.

Number of Epochs: After trained on 6 epochs with learning rate of 1e-6, training batch size of 64, validation batch size of 8. We get an ROC-AUC score of 0.83 at the 6th epoch. As the originial paper suggested, BERT is far less sensitive to hyperparameter tuning on large data sets (100k+)[3]. As we can see from the results, once the hyperparameters are set to around the optimal values, it does not affect the model much.

My fine-tuned BERT base model achieved the accuracy of 0.9012 on the test set on Kaggle. This model takes 4.7 hours to train on GPU.

Max Sequence Length	192
Training Batch Size	64
Validation Batch Size	8
Epochs	5
Learning Rate	1e-6

Table 3: Table 3. Best Performed BERT

5.2 Fine-tune BERT on TPU

I trained my best BERT model on TPU. I used XLA package, which enables PyTorch on Google TPU. To make the training process even faster, I utilized all 8 cores on TPU to parallel processing. I have trained for one epochs to see how fast the

XLA : 1	12m 16s
XLA : 2	12m 12s
XLA : 3	12m 15s
XLA : 4	12m 13s
XLA : 5	12m 18s
XLA : 6	12m 17s
XLA : 7	12m 28s
XLA : 8	12m 19s

Table 4: Table 4. Training Time on TPU

speed up on TPU is.

As we can see, it only takes 12 minutes to train for one epoch on TPU vs. 56 minutes to train for one epoch on GPU.

5.3 BERT + CNN

Based on my experiments, this model does not perform well on the dataset. The training loss failed to converge to a low rate.

5.4 Fine-tune XLM-RoBERTa

After fine-tuning XLM-RoBERTa, my best performed model achieves 0.9459 on test set on Kaggle.

Max Sequence Length	192
Training Batch Size	16
Validation Batch Size	4
Epochs	2
Learning Rate	5e-6

Table 5: Table 5. Best Performed XLM-RoBERTa

6 Conclusion

This paper has shown that transfer learning on powerful models like BERT achieves much better result than creating a model from scratch. This is because models like BERT takes a lot of resources to train and it generalize very well on different tasks.

Also, I have found that multi-lingual models performs better than mono-lingual models.

Lastly, TPU is powerful especially comes to training a very large model like BERT.

References

[1] Camaro. test-en-df, Mar 2020.

[2] Alexis Conneau, Kartikay Khandelwal, Naman Goyal, Vishrav Chaudhary, Guillaume Wenzek, Francisco Guzmán, Edouard Grave, Myle Ott, Luke Zettlemoyer, and Veselin Stoyanov. Unsupervised cross-lingual representation learning at scale. 2019.

[3] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018.

[4] Yoon Kim. Convolutional neural networks for sentence classification. *CoRR*, abs/1408.5882, 2014.

[5] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized BERT pretraining approach. *CoRR*, abs/1907.11692, 2019.

[6] Ilya Loshchilov and Frank Hutter. Fixing weight decay regularization in adam. *CoRR*, abs/1711.05101, 2017.

[7] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017.