# Explore The Power of Transfer Learning: BERT for Multilingual Toxic Comments 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 [5]. 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 has achieved state-of-the-art results when applied to different NLP tasks. In this paper, I show that transfer learning utilizing BERT and XLM-RoBERTa achieves near-top quality results on text classification in comparison to existing, simpler models such as CNN. Fine-tuned XLM-RoBERTa (cross lingual model), achieves an accuracy 0.9459 on the Jigsaw test set. In comparison, the top entry on Kaggle for Jigsaw scores 0.9555. Training was accelerated using Google's hardware TPU (tensor processing units) in conjunction with PyTorch-XLA. This results in an 4x speed increase over GPU training.

## 1 Introduction

For this project, I competed in the Jigsaw Multilingual Toxic Comment Classification competition hosted on Kaggle by Jigsaw and Google [6]. Jigsaw [1], formerly known as Google Ideas, is a unit within Google that tackles emerging online threats to make the internet a better place for all. With the increasing prevalence of social media, cyber bullying has become a real threat, especially to teenagers. Jigsaw and Google have therefore dedicated $50,000 to host this Kaggle competition to explore how different NLP techniques can be used to classify toxic comments.
z
Text classification is one of the principal tasks encountered in NLP. I applied transfer learning on pre-trained BERT and XLM-RoBERTa, and compared the results against CNN, which uses BERT word embeddings fed into a CNN.

In addition to fine-tuning BERT models, I used TPUs (tensor processing units) for training acceleration. TPUs are developed by Google, and only support Tensorflow natively. To enable PyTorch on TPU, I utilized the PyTorch/XLA package [10].

## 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 **B**idirectional **E**ncoder **R**epresentation from **T**ransformers 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 [5]. This is quite large. Therefore, I used BERT base for this project due to my limited computational resources.

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

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

Multi-headed attention computes self-attention multiple times in parallel and independently. The outputs are then concatenated and linearly transformed.

"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. [12]" Therefore, it helps the encoder to look at other words in the input sentence when encoding each word. Thus, it makes the encoder take the contexts of the sentence into consideration.

To calculate self-attention, the encoder creates a query vector, a key vector, and a value vector for the embeddings of different words in the sequence. Next, it calculates 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, it divides the dot product by the square root of the dimension of the key vectors, followed by an application of the softmax function. Finally it multiplies each value vector by the softmax score and sums 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 simply generate word embedding (vectors) for each word regardless of that word's context. Instead, BERT generates different word embeddings based on word context. This makes BERT a very powerful language model.

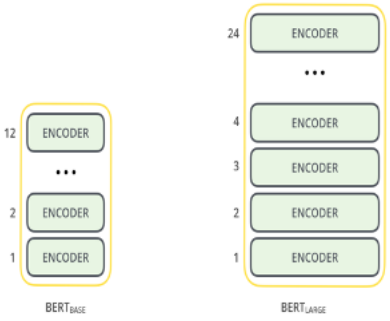


Figure 1: BERT base and BERT large [5]

## 2.2 Transfer Learning

Because of how well the BERT models performs, BERT is often combined with transfer learning to solve different NLP tasks. Instead of building a model from scratch, we can use a pre-trained model like BERT to solve our own specific

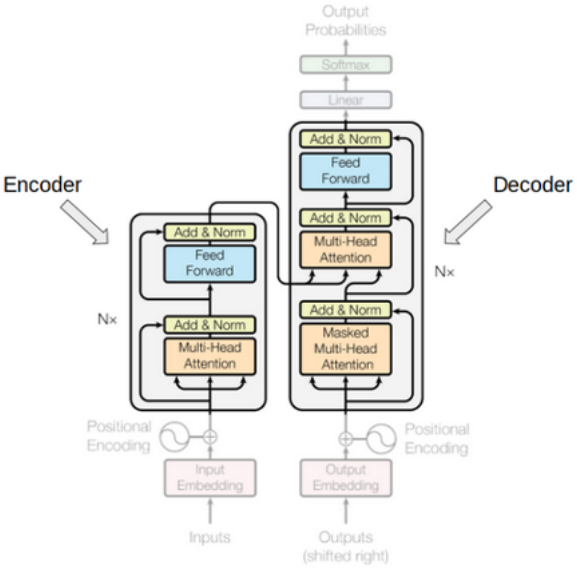


Figure 2: Transformer Architecture [12]

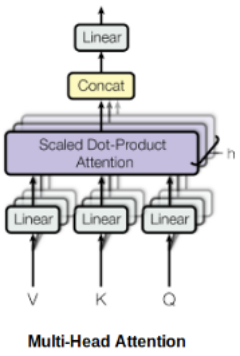


Figure 3: Multi-Head Attention [12]

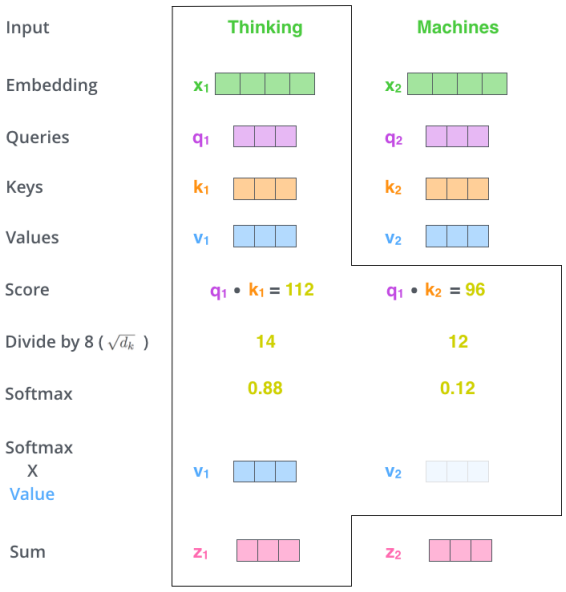


Figure 4: Calculate Self-Attention [2]

problems, with a little bit fine-tuning to adapt the model. 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.

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 the [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 the BERT model was trained, both MSM and NSP were trained together, and the goal was to minimize the combined loss function of both.

### 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 Finnish BERT, French BERT and German BERT. Research on a multilingual model continues to this day. In November 2019, the Facebook AI team released XLM-RoBERT [4], an update to their XLM-100 model. They used the RoBERTa [8] model (A Robustly Optimized BERT Pretraining Approach) to improve XLM-100 and hence produce XLM-RoBERTa.

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

### 2.4 TPU

The Tensor Processing Unit is an ASIC developed by Google to specialize in deep learning. Models that previously took days to train on GPU can be trained in hours on TPU. However, in order to use PyTorch on TPU, we have to use a package called Pytorch/XLA. There are currently no stable releases of this platform.

## 3 Data

The Jigsaw Kaggle competition [6] has provided a training set, a validation set, and a test set. The training set's comments are entirely in English. The validation and test sets' comments are composed of multiple non-English languages, such as, Spanish, French, Italian, Russian, etc. I have combined the training set from the previous competition to make a large training set of 2.1 million comments. Since my computational resources are limited, I only used the first 200,000 datapoints.

I used the BERT and XLM-RoBERTa pre-trained tokenizer to tokenize and pad the training data.

The validation set has 8,000 datapoints. I tested my trained model directly on the validation set without translating it into English because I want to see how my model is adapting to non-English languages.

The test set has 63,812 datapoints. I used this test set shared by one of the competitors [3] which translates the entire test set into English. I used this translated test set to evaluate my BERT and BERT+CNN models because I want to achieve the best score possible on test set. However, I evaluated my XLM-RoBERTa model directly on the test set because it is a cross-lingual model.

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

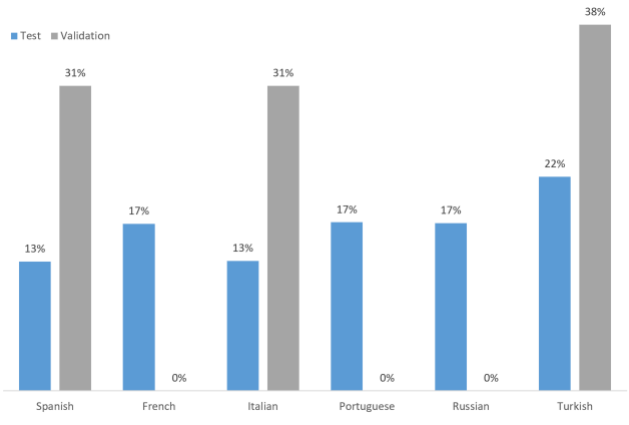


Figure 5: Languages used in Validation and Test set

## 4 Models

This section investigates three different models using both GPU and TPU.

### 4.1 BERT for Text Classification Directly

To adapt pre-trained BERT (base) model to classify toxic comments, I simply added a classifier layer on top of the 12 BERT layers. Then, I fine-tuned the hyperparameters to improve performance.

First, I added drop out for BERT output to prevent overfitting. The BERT model produces two outputs: last hidden state and pooled output. I applied mean pooling and max pooling on the last hidden state from BERT output. Then, I concatenated them and sent it to a linear (output) layer.

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

I used binary cross entropy with logits loss as the loss function, and Adam with decoupled weight decay [9] as the optimizer. To evaluate the performance on validation set, I used ROC-AUC score [11], which is a desirable way to measure classification performance. It has the following advantages: (1) It measures how well predictions are ranked, rather than their absolute values. (2) It measures the quality of the model's prediction irrespective of what classification threshold is.

### 4.2 BERT as word Embeddings + KimCNN

I used BERT tokenizer to tokenize and pad words into word embeddings. Then I fed them to a CNN introduced in this paper [7] by Yoon Kim from New York University in 2014.

Here are the steps of how KimCNN works in details [7]:

(1) The word embeddings from BERT are of size [192, 768]. This is because the max sequence length is 192 and the embedding size is 768.

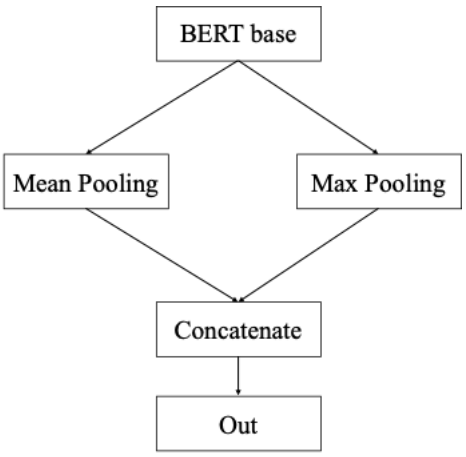


Figure 6: BERT for Text Classification

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

(3) Apply ReLU to add the ability to model nonlinear problems.

(4) Apply 1-max pooling to down-sample the output representation of the convolutional layer and to help prevent overfitting.

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

(6) Dropout (to prevent 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 pooled output. The pooled output is fed into a linear (output) layer of size (1024, 1). This is because the hidden size of XLM-RoBERTa is 1024 and the output values should be between 0 and 1.

## 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.

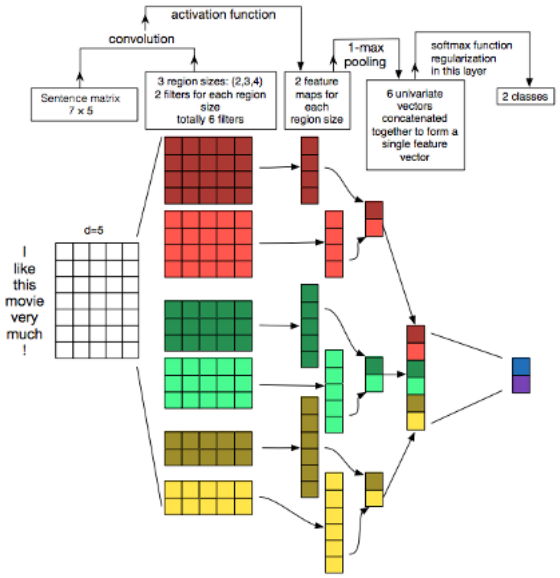


Figure 7: KimCNN [7]

### 5.1 Fine-tune BERT on GPU

For fine-tuning, I tuned these hyperparameters: learning rate, batch size and number of training epochs.

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

The BERT paper had a list of hyperparameters that generally work well across all tasks, which I used as my starting point to fine-tune [5]: 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 the hyperparameters. To tune learning rate, I fixed the training batch size to be 64 and the validation batch size to be 8. Training on one epoch. The training and validation losses are plotted every 100 batches to make the plots less clustered. I used ROC-AUC score to measure how well my model performed on validation set. Please note that the y-axes of figures below are at different scales. For reference, there are 781 training batches and 1,000 validation batches.

As shown by Table 1, with an aggressive learning rate of 5e-3, the model did poorly on the validation set. I found that a good learning rate for this model should be around 2e-5, which is suggested


Figure 8: lr = 5e-3

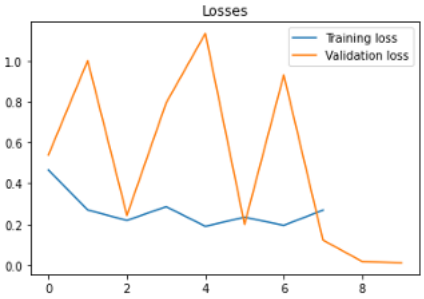


Figure 9: lr = 5e-5

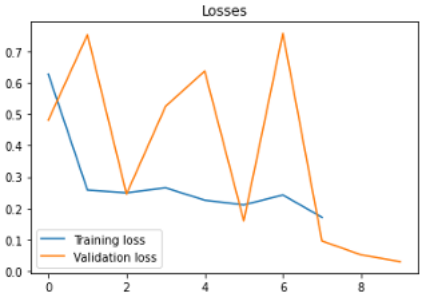


Figure 10: lr = 2e-5

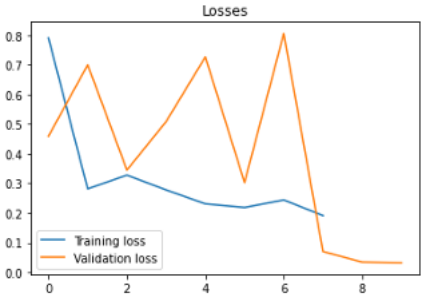


Figure 11: lr = 5e-6

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

Table 1: Learning Rate vs. ROC-AUC

by the paper.

**Batch Size:** I set learning rate to be 1e-5, and trained 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 convergence to the global optimum. It will bounce around the global optima instead.

I have found that the max training batch size I can use for this model is 64. I tried training batch sizes of 128 and 256, which will blow up the GPU memory on Google Colab notebook.

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

Table 2: Batch Size vs. ROC-AUC

This table shows that training batch size of 64 is ideal for this model given my constraints.

**Number of Epochs:** After trained on 6 epochs with learning rate of 1e-6, training batch size of 64, and validation batch size of 8. I got an ROC-AUC score of 0.83 at the 6th epoch.

As the originial paper suggests, BERT is far less sensitive to hyperparameter tuning on large data sets (100k+)[5]. The results show that once the hyperparameters are set near the optimal values, it does not affect the performance of the model much.

My fine-tuned BERT (base) model achieved the accuracy of 0.9012 on the test set on Kaggle. This model took 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: 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 epoch to see how fast the speed up on TPU is.

| | |
|---|---|
| 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: Training Time on TPU

As shown by Table 4, it only took 12 minutes to train for one epoch on TPU vs. 56 minutes to train for one epoch on GPU.

### 5.3 BERT + CNN

I started with a relatively large batch size and learning rate training for 5 epochs. Then I gradually tuned down the batch size and learning rate while training for more epochs.

However, this model does not perform well on the dataset. The training loss failed to converge to a low loss, and the ROC-AUC score is quite low.

| Training Batch Size | Learning Rate | ROC-AUC |
|---|---|---|
| 128 | 1e-3 | 0.51 |
| 128 | 1e-5 | 0.44 |
| 64 | 5e-6 | 0.46 |
| 32 | 1e-6 | 0.44 |

Table 5: BERT+CNN

### 5.4 Fine-tune XLM-RoBERTa

I did similar fine-tuning procedures as the BERT model described above. I found that the max training batch size I can use on Colab TPU is 16.
After fine-tuned XLM-RoBERTa, my best performed model achieved 0.9459 on test set on Kaggle.

| Learning Rate | ROC-AUC | Time |
|---|---|---|
| 5e-6 | 0.9394 | 46m 7s |
| 1e-6 | 0.9552 | 43m 36s |

Table 6: Learning Rate vs. ROC-AUC

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

Table 7: Best Performed XLM-RoBERTa

## 6 Conclusion

I have found that transfer learning utilizing BERT is very effective compared to using a simple model such as CNN. A large language model like BERT takes a lot of resources to train and it generalizes very well on many NLP tasks, including text classification. On the other hand, the CNN model I tried is very simple and did not have enough data to be trained on.

Transfer learning also depends on the similarity between the task the model was trained on and the task that needs to be solved. Since XLM-RoBERTa was trained to be a cross-lingual model, it generalizes well on the test set than BERT, which is a mono-lingual model.

Therefore, transfer learning utilizing BERT to do text classification will achieve a high performance without a lot of effort compared to building a specific model from scratch, which is quite challenging.

Lastly, TPU is powerful especially when it comes to training a very large model like BERT. Models like BERT and XLM-RoBERTa take a lot of computational resources to train. TPU dramatically increases the speed of training those models.

## References


[1] Jigsaw llc. https://jigsaw.google.com/.

[2] Jay Alammar. The illustrated transformer. http://jalammar.github.io/illustrated-transformer/.

[3] Camaro. test-en-df. https://www.kaggle.com/bamps53/test-en-df.

[4] 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.

[5] 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.

[6] Jigsaw. Jigsaw multilingual toxic comment classification, Mar 2020.

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

[8] 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.

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

[10] PyTorch. Pytorch on xla devices. https://pytorch.org/xla/release/1.5/index.html#.

[11] sklearn. Roc-auc score. https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_auc_score.html.

[12] 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.