CS5984 - HW2

Abhinav Sethi (PID: abhinavsethi)

October 23, 2021

1 Introduction

This document provides a brief overview of the approaches used in the Relation Classification of NYT dataset task as part of homework 2 for CS 5984 - Natural Language Processing.

2 Data

The problem statement requires building a model for relations classification of NYT dataset between the 29 defined relation types. The dataset is divided into three parts training, validation and testing set having over 63k, 7k and 4k samples respectively. Each sample consists of a string of text and two entity words with a one or more relations as labels.

3 Model

In this assignment, we have incorporated the approach used for single relation classification proposed in [1] and adapted it for multi label relation classification. This approach is based on the state-of-the-art model BERT (Bidirectional Encoder Representations from Transformers). Its key innovation is that it applies bidirectional training of Transformers, to language modelling [2]. Which is different from the previous models which looked at a text sequence either from left to right or both left-to-right and right-to-left training. The model showed that a bidirectionally trained model can have a deeper sense of language context and flow than a directional language model. For the BERT model, we have used 'bert-base-uncased' pre-trained model that was trained using a stack of 12 transformer encoders.

Figure 2 shows the architecture of the approach used in [1]. It uses special tokens of special token '$' and '#' at the start and end of the two entities so that the BERT model can capture the location information of the entities. Also a special token of '[CLS]' is added to the beginning of each sentence. The final hidden state output of the BERT model for entity $e1$ and $e2$ is then averaged to get a vector representation for the two target entities. Following this, a *tanh*

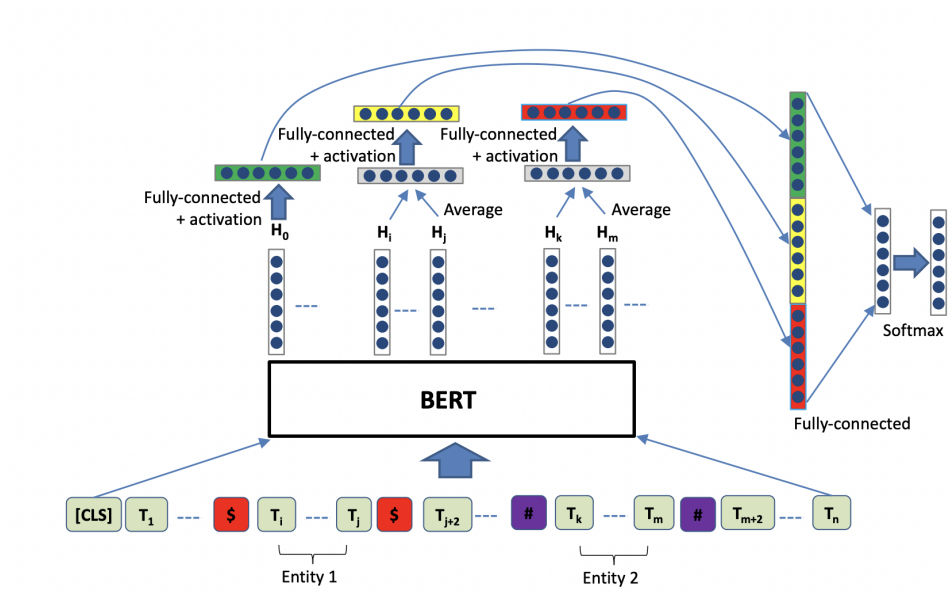

Figure 1: Model architecture [1]

activation is applied and a fully connected layer is added. Apart from the entity representation, *tanh* activation operation and a fully connected layer is added on the final hidden state of the '[CLS]' token as well. The three final vector representations are then concatenated and a fully connected layer with sigmoid activation. Here, the use of Sigmoid function is different from the paper [1] where softmax was used as this is a multi label classification problem. In each of the fully connected layer highlighted above the inputs are first flowed through a drop out layer for regularization that essentially zeros out the input values with a probability of p. Here p is a hyper-parameter its value was selected as 0.2, however it can be further tuned.

4 Pre-processing

The training for BERT model requires tokenization of the input text such that it has special tokens for unknown words, sentence separation, attention masks and in this case all has entities masks. Also, the input tokens need to be of fixed length (we have set 128) thus requires padding or truncation. All this processing is done using a custom method built on top of BERTTokenizer class. To find a good balance in the max sequence length (as with a large value we can run into memory issues and with low value a lot of information will be truncated), we plotted the token counts from the training data as given in figure 1. From the figure it is evident that the peak density is 50 tokens so we have picked 128

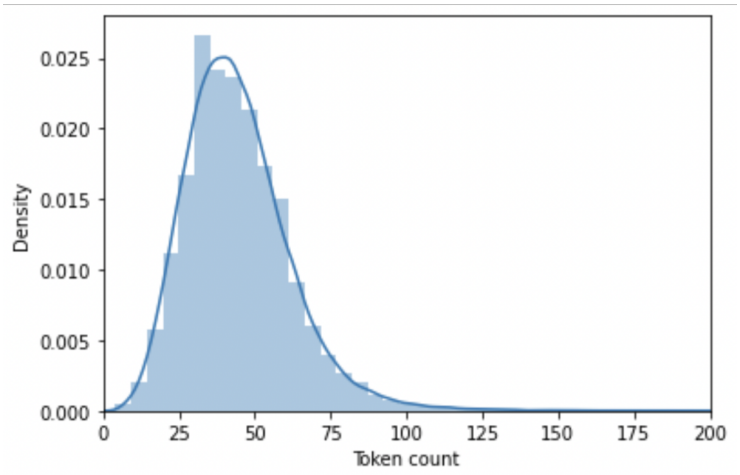

Figure 2: Encoded tokens count vs density plot

as the max length as it covers 99% of all samples. For batching of the dataset we have used the torch Dataset and DataLoader class by passing in the data along with a batch size. For this model, the batch size is set as 32. Any further increase can lead to memory issues.

We have been provided the entities starting and ending index and we use this information to add special tokens like $\{< e1 >, < e2 >, < /e1 >, < /e2 >\}$ at those locations such that we don't lose the relative positions after passing through the tokeniser. After tokenization, these tokens are replaced with '$' or '#' tokens for first and second entity respectively.

Since each sentence can have different pairs of entities, we need to add all possible permutations of those entities during pre-processing. For cases where the entity combination is not present in the dataset, we label it as 'other' and add it to the dataset. For example, if there are e1, e2, and e3 entities in a sentence with e1 - e2 pair having label_1 and e2 - e3 pair having label_2, we add 4 more pairs like e2 - e1, e3 - e2, e1 - e3, e3 - e1 and label them as 'other'.

5 Optimization

For optimization, Adam optimizer with weight decaying (AdamW) is used. It has been shown experimentally that AdamW yields better training loss and that the models generalize much better than models trained with Adam [3]. The learning rate is a hyper-parameter and was set to be $2e^{-5}$ after some limited tuning, however further tuning may be required to improve performance.

We have also incorporated a linear scheduler to further manipulate the learning rates during the batch training process. The learning rate selected above is the initial rate for a linear scheduler with warm-up that essentially first increases the learning rate and then starts decreasing linearly. The number of

Parameter	Current model's value
Batch Size	32
Number of Warm-up steps	500
Drop out probability	0.2
Learning rate	$2e^{-5}$
Max Length	128

Table 1: Model Parameters Used

warm up steps was set to 500 as it equals to about 10% of the training steps (as with 32 batch size we have total 4600 steps per epoch). Here, the number of warm up steps can also be further tuned. Also, for the loss function we have used BCELoss. Table 2 shows a list of all the hyperparameter values.

As an outline of the training process, for each batch, first all gradients are set to zero. Then the batch data is fed to the model then the current gradients are stored using the loss.backward() method call. Following this the gradients are clipped to the max value of 1 to prevent exploding gradients using clip_norm method from the pytorch utils. After the optimizer and the scheduler steps are iterated over before the next batch is trained.

6 Results

Since this is a multi label prediction task, we have applied a threshold value over the output vector to get a binary vector output. Such that, any value greater than threshold is replace with 1 and rest with 0. This final output is then used for F1 score computation. Also, any sample with true label as 'other' is ignored during F1 score computation.

We ran the model for 3 epochs separately (google collab requires a consistent connection, otherwise it automatically disconnects) as each epoch requires a training time of 95 minutes (approximately).

The performance statistics were produced using sklearn library's classification_report method. The final F1 micro average score on the test set was 89% after the three epochs. The overall summary of the results after the final epoch is shown in table 3. The validation set F1 score is found to be 96% which is quite close to the training F1 score of 97%. It is possible that the two datasets may have similar contents due to which the scores is found to be similar. As the F1 score has only dropped 8 points when compared with the training accuracy (96%), it shows that the model has generalized seemingly well.

7 Analysis

The total training time for the current model is $3 * 95 = 285$ minutes. Which has essentially hindered the proper tuning of the model. The F1 score can further be improved by running the model for more number of epochs. Apart from this,

Type	Raw Samples	Processed	Prediction	F1 (micro)
Train	63306	146496	78926	97%
Validation	7033	16352	8765	96%
Test	4006	9408	5851	89%

Table 2: Results after final epoch

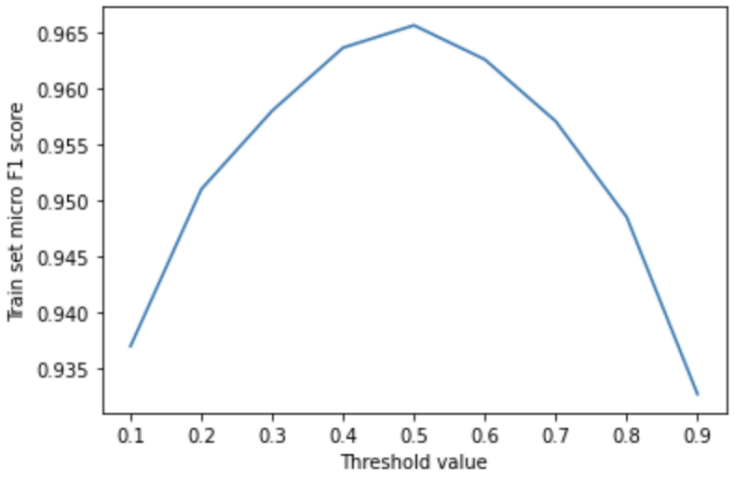

Figure 3: Threshold value applied to output vector vs training F1 score

hyper-parameters such as number of warm up steps, drop out probability, and learning rate can further be tuned to improve testing accuracy.

The model outputs a vector of size equal to the number of labels and with values ranging from 0 to 1. To get the multiple labels as output, we need to apply a threshold value to ascertain which labels are the outputs. For this, we performs a search by varying the threshold values from 0.1 to 1 in steps of 0.1 and measured the resulting F1 score on the training set. Figure 3 clearly shows that threshold value at 0.5 gives the best prediction.

In the classification report we notice that about 10 labels out of 30 have very poor prediction accuracy, this can be attributed to the fact that the training data for those label type is very limited. As a result the model does not have sufficient information about them. This issue can be addressed by adding more training samples for those labels.

Although we only used a generic BERT base model for our task, yet it has proved to be highly effective and has performed quite well in this Relation classification task leading to good generalization for prediction on new texts.

References

[1] Enriching Pre-trained Language Model with Entity Information for Relation Classification (2019), Shanchan Wu, Yifan He [arXiv:1905.08284v1]

[2] BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding (2019), NAACL, Jacob Devlin and Ming-Wei Chang and Kenton Lee and Kristina Toutanova [arXiv:1810.04805v2]

[3] Decoupled Weight Decay Regularization (2019), ICLR, Ilya Loshchilov, Frank Hutter [arXiv:1711.05101v3]