CS410 Final Project Report

Text Classification Competition: Twitter Sarcasm Detection

Our Team:

Team name:
Salty Fish
Team captain:
Name: Jinning Li NetID: jinning4
Team member 1:
Name: Jialong Yin NetID: jialong2
Team member 2:
Name: Jianzhe Sun NetID: jianzhe2

Introduction

The final project we choose is the text classification competition. The main task of the competition is to do the sentiment analysis of a given tweet to determine whether it is sarcasm or not.

We first built a model using pre-trained LSTM to do the classification. We only used the response data to train the model and we did not beat the baseline. We did not make use of the context data, and we also did some research on the state-of-art network models of the sentiment analysis.

We then built a BERT model using a pre-trained BERT. We used BCEloss to finetune the parameters during training and added a fully connected layer to output the classification result. With this method we beat the baseline, and the best result we achieved is (0.6832, 0.8433, 0.7549), precision, recall and F1 score respectively.

LSTM Baseline

Introduction of LSTM

LSTM is the abbreviation of Long short-term memory. LSTM is a special kind of RNN, which solves the deficiency of gradient vanishing and exploding during long-term training that RNNs have.
LSTM consists of three gate structures:forget gate, input gate and output gate.

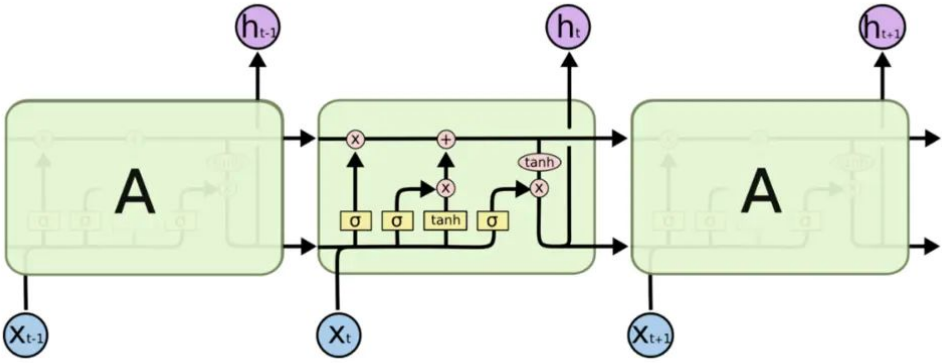

Fig. Overall structure of the LSTM

The forget gate layer is implemented by a sigmoid layer. It looks at h_{t-1} and x_t ,and outputs a value, denoted as f_t , between 0 and 1 for each number in the cell state C_{t-1} , where h_{t-1} is the output at timestamp t-1 and x_t is the input at timestamp t.

The input gate layer is used to update the information in cell state. This process has two steps. The first step is to use a sigmoid layer to decide which value to get updated, denoted as i_t , and the second step is to use a tanh layer to create a vector of new candidate values, C'_t . Then we can update the cell state using the previous output from the forget gate and the input gate.
The new cell state C_t is calculated as:

$$Ct = f_t * C_{t-1} + i_t * C'_t$$

The output gate layer is to decide the output value. First, we use a sigmoid layer to decide what parts of the cell state will be output. Then, we put the cell state through tanh and multiply by the output of the sigmoid layer, denoted as o_t , and then we can get the final output h_t .

Result of using LSTM

We first tried to use the pre-trained LSTM from pytorch and we added two fully connected layers to make the classification given the output of the LSTM. In this case we get two classifications, ‘SARCASM’ and ‘NOT_SARCASM’.

The best result we get using this method is:

Precision	Recall	F1 Score
0.6109979633401222	0.6666666666666666	0.6376195536663125

From the result we can see that the pre-trained LSTM model without fine-tuning can not beat the baseline, this is probably because LSTM does not perform well when dealing with long term dependency problems. So we decide to try another model, BERT, to get better performances. The LSTM model is implemented by Python 3.7.0, PyTorch 1.2.0 and trained on Intel x86 CPU devices with 8 cores. The code can be found in https://github.com/jinningli/CourseProject. Please refer to the Code Manual section for the usage of our code.

BERT Model and Logits Ensemble

BERT Model

BERT is a language representation model, which is pre-trained on unlabeled text data at first and fine-tuned with an additional output layer for different NLP tasks. BERT can learn a bidirectional representation for each word on both left and right context, compared to the static word embedding learned in word2vec.

BERT's model architecture is a multi-layer bidirectional Transformer encoder based on the original implementation described in Vaswani et al. (2017) as shown in Fig.

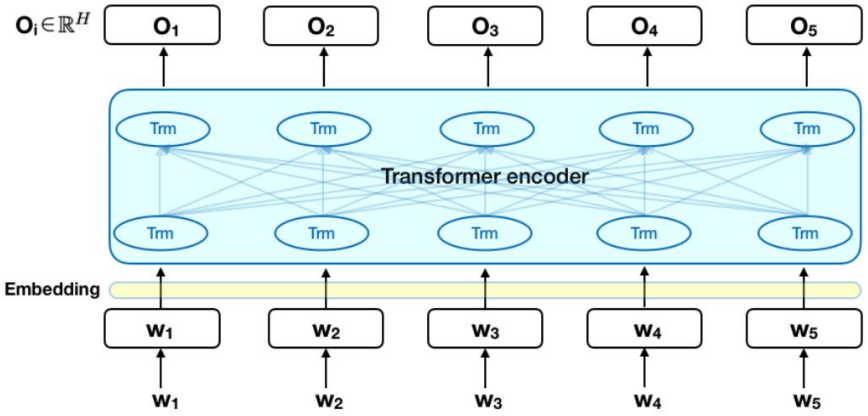

Fig. BERT architecture based on Transformer Encoder.

Two unsupervised tasks are used to pre-train BERT. The first task is Masked language model (MLM) where some percentage of the input tokens are masked at random, and the final hidden vectors corresponding to the mask tokens are fed into an output softmax over the vocabulary for prediction. The second task is Next Sentence Prediction (NSP) where the hidden vector at [CLS] token is used to classify whether the given two sentences are continuous sentences or not. Both tasks can be trained jointly as in Fig.

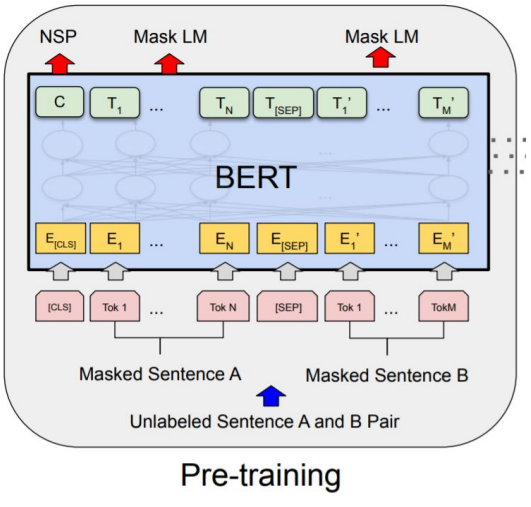

Fig. Two unsupervised tasks for BERT pre-training.

During fine-tuning, the BERT model is first initialized with the pre-trained parameters, and a fully connected layer is added as the output layer. All of the parameters are tuned using labeled data from the downstream tasks as shown in Fig.

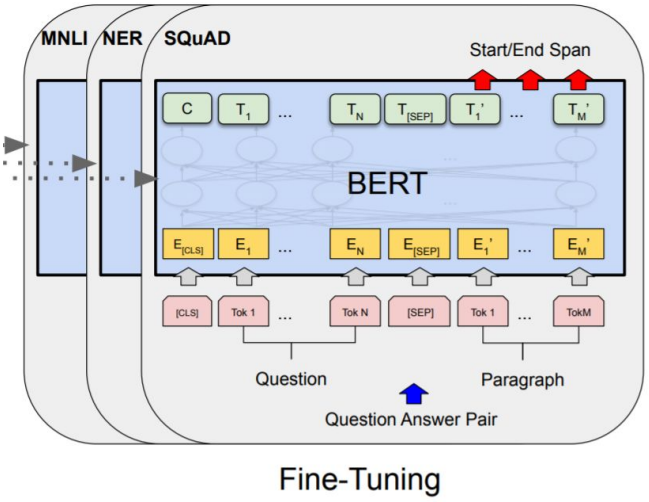

Fig. Procedure of BERT fine-tuning.

Since BERT obtained the state of the art result in eleven NLP tasks once published, variants of work are proposed based on the BERT model and push the performance of NLP tasks a big step forward such as XLNet and ALBERT.

Our method is based on the BERT model. We load the weights of BERT from PyTorch-Transformers which is pre-trained on BooksCorpus (800M words) and English Wikipedia (2,500M words). A fully connected layer is added as the output layer to classify Twitter response into “SARCASM” or not. All the parameters are tuned in the training process with BCELoss.

The Twitter dataset is splitted into training dataset with 4000 samples and validation dataset with 1000 samples. Validation dataset is used to tune the hyperparameters during training and find 2 epochs is the best since the validation loss starts to grow up after it probably due to the limited dataset. Finally, the model is trained on the whole dataset including both training dataset and validation dataset.

The model is implemented by Python 3.7.0, PyTorch 1.2.0. We train the model on NVIDIA TITAN V GPU, 16 cores Intel x86 CPU machine. The code can be found in https://github.com/jinningli/CourseProject. Please refer to the Code Manual section for the usage of our code.

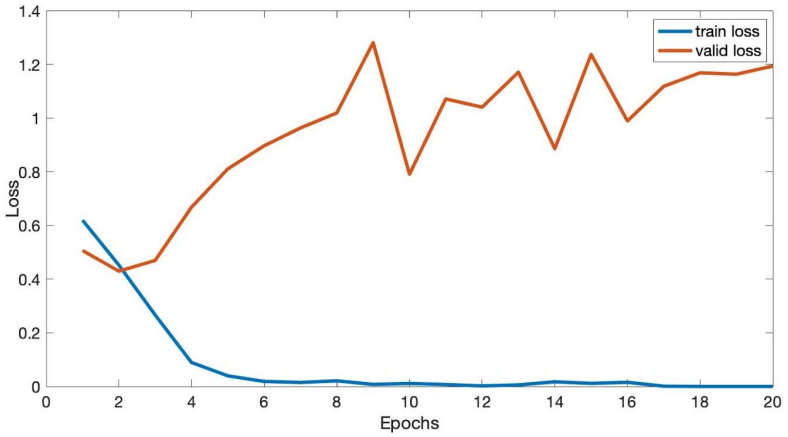

Fig. Train Loss and Validation Loss

From the figure above, we can see that the validation loss is relatively low around epoch 2-4. Then after epoch 4 the validation loss is high due to overfitting. This is mainly because the size of the training dataset is small, so in this case the model tries to overfit the data in the training dataset rather than generalize from patterns observed from the dataset. This results in the low training loss and high validation loss after epoch 4. Therefore, we only take the result of 2-4 epochs for further ensemble.

In our final submission, we used all the 5000 samples (without validation) in the training dataset to train the model to get better performances.

Logits Ensemble

In order to further improve the performance of our model on the testset, we tried to submit several results to the LiveDataLab OnlineJudge system and collect the Precision, Recall, and F1 Score response. The scores are shown in the figure below. We find that our epoch 3 model and epoch 4 model have the similar highest F1 Score. In addition, their precisions and recalls are varied. The ensemble method should be effective by taking the average of logits values (the output of neural network).

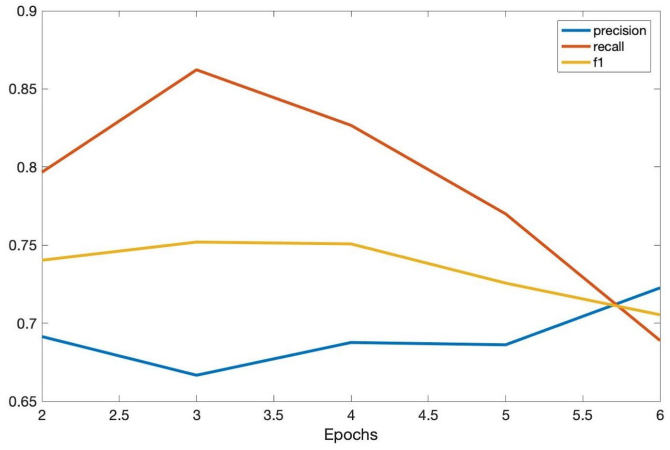

Fig. Precision, Recall, and F1 Score on Testset (Response after submission)

We ensemble the logits by taking the average

$$avglogit = \sum_{i}^{n} logit_i / n$$

The prediction is the argmax of the average logits

$$pred = argmax(avglogit)$$

The ensemble method further improves our F1 Score from 0.7519 to 0.7549

Model	Precision	Recall	F1 Score
BERT Epoch 3	0.6667	0.8622	0.7519
BERT Epoch 4	0.6876	0.8267	0.7508
BERT Epoch 3&4 Ensemble	0.6832	0.8433	0.7549

Table 1: Ensemble Improvement

Conclusion

In conclusion, we built two models, an LSTM model and a BERT model to accomplish the twitter sarcasm detection. The LSTM model with a pre-trained LSTM and two fully connected layers did not beat the baseline. The BERT model with a pre-trained BERT and a fully connected layer using BCEloss beats the baseline.

Code Manual

Evaluate Model and Ensemble

Download pre-trained model for our method

Download `checkpoint.zip` from https://drive.google.com/file/d/1nRucz1yDqyoYR8jLeP6fKZt8FpqJBBUA/view?usp=sharing

unzip checkpoint.zip
mkdir checkpoint; mkdir checkpoint/run0
mv checkpoint checkpoint/run0/checkpoint

Evaluate and Generating Predictions

python3 evaluate.py --run run0 --use_bert --device 3

Ensemble

```
# modify ensemble_paths in ensemble.py
python3 ensemble.py
```

All Parameters

`--run` Evaluating on runX. e.g. --run=run1

`--use_bert` Use bert model. If not, using LSTM model.

`--device` GPU device to use

Train Model

Start Training with Validation and Evaluation

```
python3 main.py --use_bert --device 2 --lr 2e-5 --epochs 20 --use_valid --eval
```

Start Training without Validation

```
python3 main.py --use_bert --device 2 --lr 2e-5 --epochs 20
```

Start Training using LSTM model

```
python3 main.py --device 2 --lr 2e-5 --epochs 20
```

All Parameters

`--epochs` Number of epochs for training

`--batch_size` Batch size

`--lr` Learning rate for optimizer

`--run` Continue training on runX. Eg. --run=run1

`--eval` Evaluate on training set

`--use_valid` Use valid dataset. If not, using the whole dataset for training

`--use_bert` Use bert model. If not, using the LSTM model

`--split_ratio` When using validation, percentage of trainset

`--device` GPU device to use

Appendix

Screenshot of our scores and rank:

Rank	Username	Submission Number	precision	recall	f1	completed
18	Li, Jinning	15	0.6831683168316832	0.8433333333333334	0.7548483341621084	1