JIE WANG
CSCI 4900 Independent Study
04/18/2019

Sentiment analysis based on Bert

Bidirectional transformer application

1 Introduction

This semester, I have finished a sentiment analysis based on BERT which released by Google research in 2017. BERT stands for **B**idirectional **E**ncoder **R**epresentations from **T**ransformers[1]. This model is using fine tuning to solve the pre-trained language representation which is totally different with word2vec[2] and Elmo[3] model. Word2vec is usually talking about transferring word into vector-format. And Elmo[3] is transferring a context-independent static vector to be a context-dependent dynamic vector. BERT is transferring word-level vector to be sentence-level vector so that when we do some downstream tasks, we can call easily. And the most difference between Elmo[3] and BERT is its encoding part which Elmo uses LSTM and BERT uses Transformer. That's why I think Bert will make my sentiment analysis more efficient than other models.

2 Background

Bert = pre-trained part + fine tuning part

2.1 pre-trained part:

How to train Bidirectional Model? In "Attention is all you need" by Google, they use two novel "unsupervised" prediction task. [1]

Task1: Masked language model

e.g: Input: The man went to the [mask1], he bought a [mask2] of milk

Label: [Mask1] = store

[Mask2] = gallon

According to paper by Google, they randomly masked 15% in corpus(Wiki + Book corpus), and then they passed final hidden vectors which is the output of masked token to softmax layer, to predict the masked token.

Task2: Next sentience prediction

e.g: Sentence A : the man went to the store

Sentence B : he bought a gallon of milk

Label: Isnextsentence

Sentence A: the man went to the store

Sentence B: penguins are flightless

Label: Notnextsentence

According to paper by Google, authors think that many NLP tasks like QA and NLI need to understand the relationship between two sentences. [1] And language model can't show that kind of relation. So they just add this task to make it more suitable to the specifics tasks.

And their training data BooksCorpus includes 800 million word and English WiKi includes 2500 million word. In my project, I use BERT-Base uncased 12-layer, 768-hidden, 12-heads, 110M parameters to do next step, Fine Tuning. Because Google has already finished the expensive part, pre-trained. And I am going to modify the "cheap" part, fine tuning, to make it suitable with my project.

2.2 fine tuning part

Fine tuning is more focusing on the specifics task. In the Google Bert code, there are two files doing responsible for fine tuning, run_classifier.py and run_squad.py.[5] Because my task is about sentence classifier, I use run_classifier.py as my base code.

2.21 preparing for fine tuning

According to main function code:

```
if __name__ == "__main__":
    flags.mark_flag_as_required("data_dir")
    flags.mark_flag_as_required("task_name")
    flags.mark_flag_as_required("vocab_file")
    flags.mark_flag_as_required("bert_config_file")
    flags.mark_flag_as_required(“output_dir")
        tf.app.run()
```

we need 5 things to do train or do eval, data_dir, task_name, vocab_file, bert_config_file and output_dir.

2.21.1 Data_dir

In this directory, we need 4 file according to Code which Google given.

dev.csv: This file contains 28 dataset when we eval and verify.

test_sentiment.txt: This file contains 132 dataset which has already labeled from 0 - 4 witch represent ‘heart’ ‘game’ ‘smile’ ‘disappointed’ ‘neutral’

train_sentiment.txt: This file contains 183 dataset which labeled from 0 - 4 and the same format with test_sentiment.txt.

train.csv: This file contains 28 dataset which is different with dev.csv to do train.

*all data are from Andrew Ng Sequence Models on Coursera [4]

https://www.coursera.org/learn/nlp-sequence-models/home/welcome

2.21.2 Task_Name

```
processors = {
        "cola": ColaProcessor,
        "mnli": MnliProcessor,
        "mrpc": MrpcProcessor,
        "xnli": XnliProcessor,
        "sim": SimProcessor,
}
```

When we are doing any training and prediction of model, we have to get an explicit input. And the processor is responsible for processing the input.

In this sentiment analysis task, we should do cross training which means we need to do eval after we do train for several times to get a better result. So I define a Sim processor to reload the label and get the input of get_train_examples, get_dev_examples and get_test_example. And take get_train_examples as an example. We return InputExample class as output. test_a and test_b are two different strings. When we use those in InputExample, it will pass text_a[SEP]text_b[SEP] as format.

Also we have 5 different labels to distinguish different sentiments.

```
def get_labels(self):
        return ['0', '1', ‘2','3','4']
```

2.21.3 vocab_file and bert_config_file

These two files are from uncased_L-12_H-768_A-12 which is the directory of pretrain dataset.

2.21.4 output_dir

This is the directory which show the output file.

2.22 do fine_tuning.

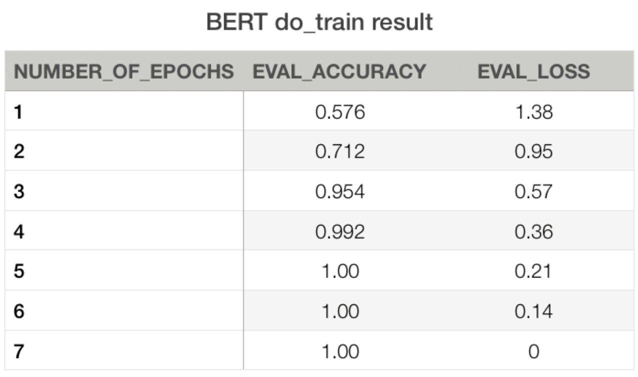
The command is :

```
python3 run_classifier.py \
  --data_dir=data \
  --task_name=sim \
  --vocab_file=uncased_L-12_H-768_A-12/vocab.txt \
  --bert_config_file=uncased_L-12_H-768_A-12/bert_config.json \
  --output_dir=sim_model \
  --do_train=true \
  --do_eval=true \
  --init_checkpoint=uncased_L-12_H-768_A-12/bert_model.ckpt \
  --max_seq_length=70 \
  --train_batch_size=32 \
  --learning_rate=5e-5 \
  --num_train_epochs=1.0
```

Because the number of training epochs is the number of time we do cross training. In my opinion, maybe there is some relationship existed between epochs and eval accuracy. (eventually convergence) So I did an experiment on this thought.

2.22.1 Experiment

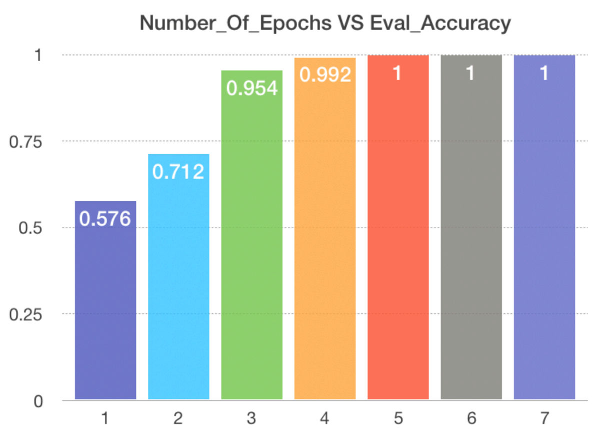
In experiment, I use number of training epochs as input and get eval accuracy and eval loss as output. Finally, find the relationship between those.



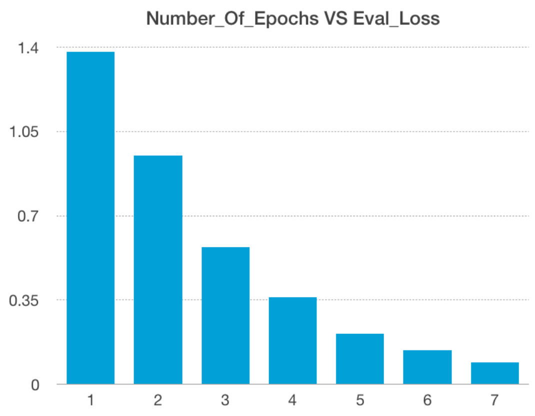
BERT do_train result

NUMBER_OF_EPOCHS	EVAL_ACCURACY	EVAL_LOSS
1	0.576	1.38
2	0.712	0.95
3	0.954	0.57
4	0.992	0.36
5	1.00	0.21
6	1.00	0.14
7	1.00	0

In this matrix chart, I record the number of epochs, eval accuracy and eval loss.



According to matrix chart, this is a column chart to show the relationship between the number of epochs and accuracy.

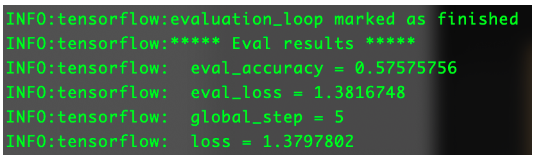


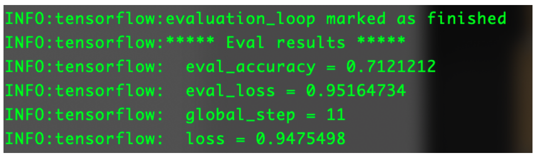
According to matrix chart, this is a column chart to show the relationship between the number of epochs and eval accuracy.

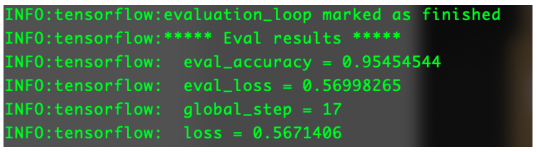
2.22.2 experiment result:

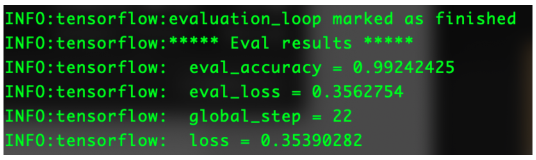
According to what I see from the result chart, more number of epochs, more eval accuracy we get. And more number of epochs, less eval loss we get.

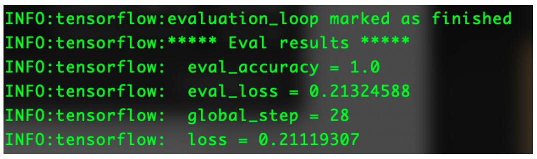
data_source image:

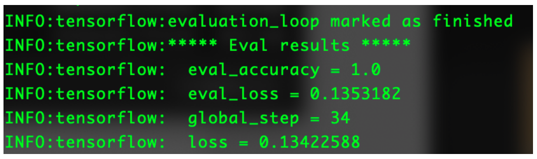

```
INFO:tensorflow:evaluation_loop marked as finished
INFO:tensorflow:***** Eval results *****
INFO:tensorflow:  eval_accuracy = 0.57575756
INFO:tensorflow:  eval_loss = 1.3816748
INFO:tensorflow:  global_step = 5
INFO:tensorflow:  loss = 1.3797802
```

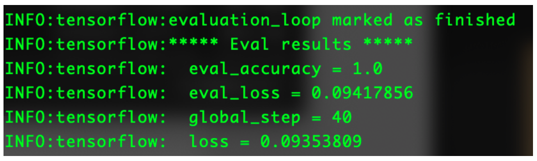

```
INFO:tensorflow:evaluation_loop marked as finished
INFO:tensorflow:***** Eval results *****
INFO:tensorflow:  eval_accuracy = 0.7121212
INFO:tensorflow:  eval_loss = 0.95164734
INFO:tensorflow:  global_step = 11
INFO:tensorflow:  loss = 0.9475498
```


```
INFO:tensorflow:evaluation_loop marked as finished
INFO:tensorflow:***** Eval results *****
INFO:tensorflow:  eval_accuracy = 0.95454544
INFO:tensorflow:  eval_loss = 0.56998265
INFO:tensorflow:  global_step = 17
INFO:tensorflow:  loss = 0.5671406
```

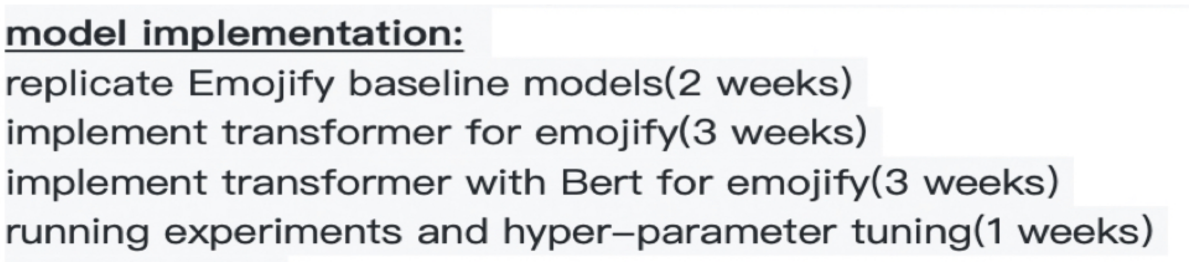

```
INFO:tensorflow:evaluation_loop marked as finished
INFO:tensorflow:***** Eval results *****
INFO:tensorflow:  eval_accuracy = 0.99242425
INFO:tensorflow:  eval_loss = 0.3562754
INFO:tensorflow:  global_step = 22
INFO:tensorflow:  loss = 0.35390282
```


```
INFO:tensorflow:evaluation_loop marked as finished
INFO:tensorflow:***** Eval results *****
INFO:tensorflow:  eval_accuracy = 1.0
INFO:tensorflow:  eval_loss = 0.21324588
INFO:tensorflow:  global_step = 28
INFO:tensorflow:  loss = 0.21119307
```


```
INFO:tensorflow:evaluation_loop marked as finished
INFO:tensorflow:***** Eval results *****
INFO:tensorflow:  eval_accuracy = 1.0
INFO:tensorflow:  eval_loss = 0.1353182
INFO:tensorflow:  global_step = 34
INFO:tensorflow:  loss = 0.13422588
```


```
INFO:tensorflow:evaluation_loop marked as finished
INFO:tensorflow:***** Eval results *****
INFO:tensorflow:  eval_accuracy = 1.0
INFO:tensorflow:  eval_loss = 0.09417856
INFO:tensorflow:  global_step = 40
INFO:tensorflow:  loss = 0.09353809
```

3 Conclusion


model implementation:
replicate Emojify baseline models(2 weeks)
implement transformer for emojify(3 weeks)
implement transformer with Bert for emojify(3 weeks)
running experiments and hyper–parameter tuning(1 weeks)

According to my plan, at first I learn Sequence models online to get the basic knowledge for RNN which includes vanishing gradient, Gated Recurrent Unit(GRU), LSTM, Bidirectional RNN and so on. And during spare time, I went to Professor James Martin NLP class to audit and learn word embedding method, word2vec[2], sentiment classification and so on. Learning basic knowledge takes me 3 weeks

And then I try to rebuild the Emojify code my own computer because LSTM source code used to run on Coursera Jupyter Notebook. That takes me 3-4 weeks to implement transformer for emojify.

Next, I read the material and paper about the mechanism of BERT and the code which was given by Google. The main paper I read is "Attention is all you need". And coding part is from https://github.com/google-research/bert. After that, I try to run that source code on my computer. Those things takes me 4 weeks to implement transformer with BERT.

Finally, I run experiment and hyper-parameter tuning to test my thought. And get "more number of epochs, more eval accuracy we get. And more number of epochs, less eval loss we get." as result.

Acknowledgements I am grateful to Professor James Martin and Phd Jon Z. Can for their suggestion and inspiration.

References:

[1] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, Illia Polosukhin. Attention Is All You Need. arXiv:1706.03762 [cs.CL]

[2]Tomas Mikolov, Kai Chen, Greg Corrado, Jeffrey Dean. Efficient Estimation of Word Representations in Vector Space. arXiv:1301,3781

[3] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, Luke Zettlemoyer. Deep contextualized word representations. arXiv: 1802.05365[cs.CL]

[4] Andrew Ng Wu. Machine Learning. AI to Everyone. Coursera

[5] run_classifier BERT https://github.com/google-research/bert