Natural language inference via textCNN

A cute kid
The digital version of the manuscript.

Abstract

I propose a series of methods for the natural language inference task based on the model of textCNN(kim and Yoon, 2014), which integrates the BERT(Devlin, Jacob and Chang, Ming-Wei and Lee, Kenton and Toutanova, Kristina, 2019) model effectively, without a full pre-training process. Unlike recent textCNN-based models, I describe the textCNN in more detail, which has focused on 2D convolutional kernels and pooling layers. The model first utilizes BERT's tokenizer to split the sentences into tokens and sends them to BERT's embedding layer; transforms the features of tokens using different 2D convolution kernels; filters the main features with a pooling layer; updates the model parameters by back-propagation and a linear layer. Towards more efficiency, I completed the entire training process of the model through few-shot learning. I evaluate my models on three datasets and achieve a 100% accuracy in OCNLI. I also show an exhaustive analysis of the outputs of the models to prove the advantage of the method that I proposed.

1 Introduction

With the development of natural language processing, the natural language inference task(NLI) is becoming more and more important. Instead of the complex definition by the Stanford University Visiting Scholar from China, I interpret the natural language inference task simply by browsing the dataset and using a computer to reason about the relationship within sequential sentences. However, the natural language inference task is quite intricate in the real world. Figure 1 clearly shows that a nature research journal bot says the Chinese NLI is difficult. Since the sequence of the two sentences has changed, the meaning changes, and depending on the scenario, the same conversation between different people, the meaning expressed can arise differently.

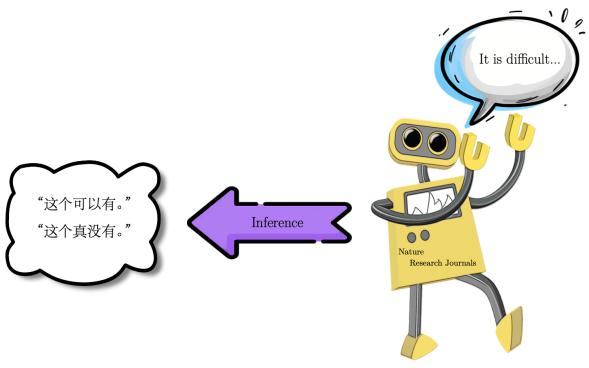

Figure 1: The Introduction of Natural Language Inference.

In recent years, BERT pretrained models have made great strides in NLI tasks. Nevertheless, the present textCNN methods are still vague. In this paper, I provide a clearer explanation of the textCNN approach and the contribution of my paper are followed:

- Directly invoke the basic BERT pre-trained model that have been trained to represent the text and combine it with textCNN.
- The training method uses few-shot learning strategy, which samples only a few amount of data from the training set and reduces the entropy loss value of the model to achieve convergence.
- Dimensionality reduction analysis using SVD in the test set.
- The distribution analysis of the model's output using violin plots.
- Overall discrepancy analysis of the model's output in the test set using scatterplots.
- Localized error case studies from the test set using bertviz.
- 2D Convolution kernel visualization.

2 Model

In this section, I decomposition the architecture of the model into four parts to describe, which are pretrained model, convolution neural network, pooling layer and linear layer.

2.1 Pretrained model

In this paper, invoking the BERT model involves two steps shown in figure 2. The first step is to invoke

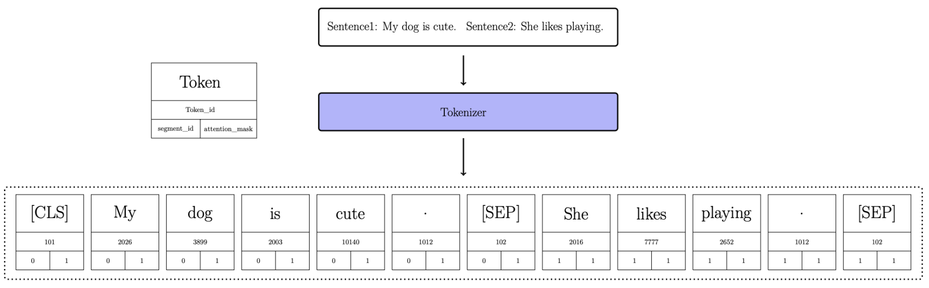

Figure 2: The BERT tokenizing process.

the tokenizer to tokening the given sequence. Then, send the tokens to the Bert model that has been trained to represent. In this paper, I utilize two methods to fine-tune the pretraining representation. One uses the "[CLS]" representations in all layers of BERT for fine-tuning, and the other uses the representations of all tokens of one sequence in the last layer of BERT for fine-tuning. For the BERT base model, the tensor shape output by "[CLS]" fine-tuning is [batch_size,layer_size,768], and the tensor shape output by the last layer fine-tuning is [batch_size,max_length,768].

2.2 2D Convolution kernel

Figure 3 shows the convolution process by a 2d convolution kernel. The overall process of convolution refers to continuously doing matrix dot product operations with local features of the sequence through the convolution kernel. In this paper, I utilize three category 2d convolution kernels to operate the embedding layer individually. There are four convolutional kernels of each kind, with the shapes of [3,768], [5,768], and [7,768]. To keep the shape of the output tensor consistent after operating, it can also pad the tensor timely according to the different sizes of the convolution kernels.

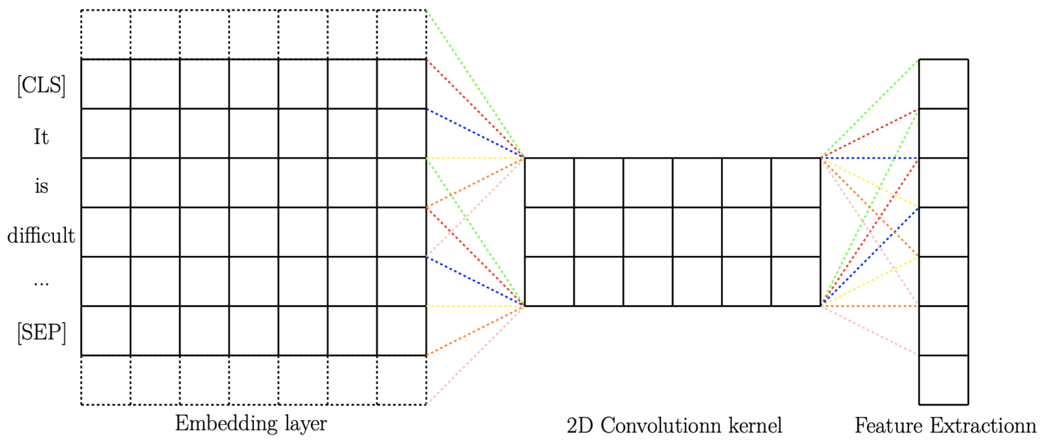

Figure 3: The convolution process.

2.3 Pooling layer

The figure 4 shows the pool process. First, the convolved output is fed into the GELU activation function to operate. Then, in this paper, the pooling's output is obtained by invoking the nn.Maxpool2d((2,1)).

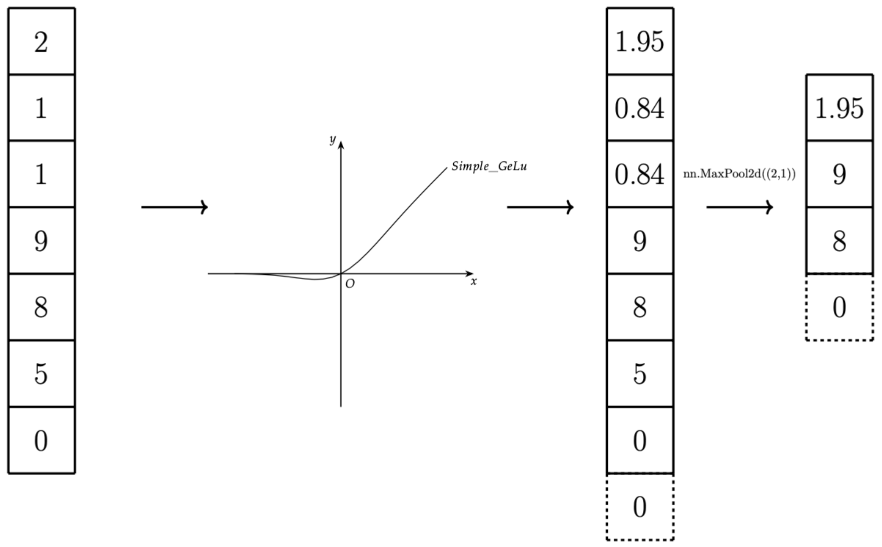

Figure 4: The pool process.

2.4 Linear layer

The linear layer refers to a matrix multiplication operation. The entire model updates its parameters from the bottom up using the back-propagation algorithm.

3 Datasets and Experimental Setup

I test my model on three datasets. Summary statistics of the datasets are followed:

- **MRPC(MSRParaphraseCorpus.msi, 2024):** A text file containing 5800 pairs of sentences which have been extracted from news sources on the web, along with human annotations indicating whether each pair captures a paraphrase/semantic equivalence relationship.
- **OCNLI(Hu, Hai and Richardson, 2020):** The first large-scale NLI dataset for Chinese called the Original Chinese Natural Lannguage Inference dataset.
- **SNLI(MacCartney, Bill and Manning, Christopher D., 2008):** A collection of sentence pairs labeled for entailment, contradiction, and semantic independence.

3.1 Hyperparameters and Few-shot Traininng

Table 1 shows the hyperparameters of the model. The training strategy used in this experiment is few-shot learning. Previously, training with few-shot learning typically aimed to achieve high performance using only a small number of samples.

Hyperparameters	Value
batch size	1
learning rate	1e-5
epoch	300
torch.mnual_seed	66

Table 1: Hyperparameters.

In this paper, I use a much smaller sample to train the model, i.e., 6 pieces of data (covering all the categories of the labels in the dataset). The aim is to demonstrate that my model can effectively reduce the loss value. Figure 5 shows the variation of epoch and loss values. The trend of the line in the figure

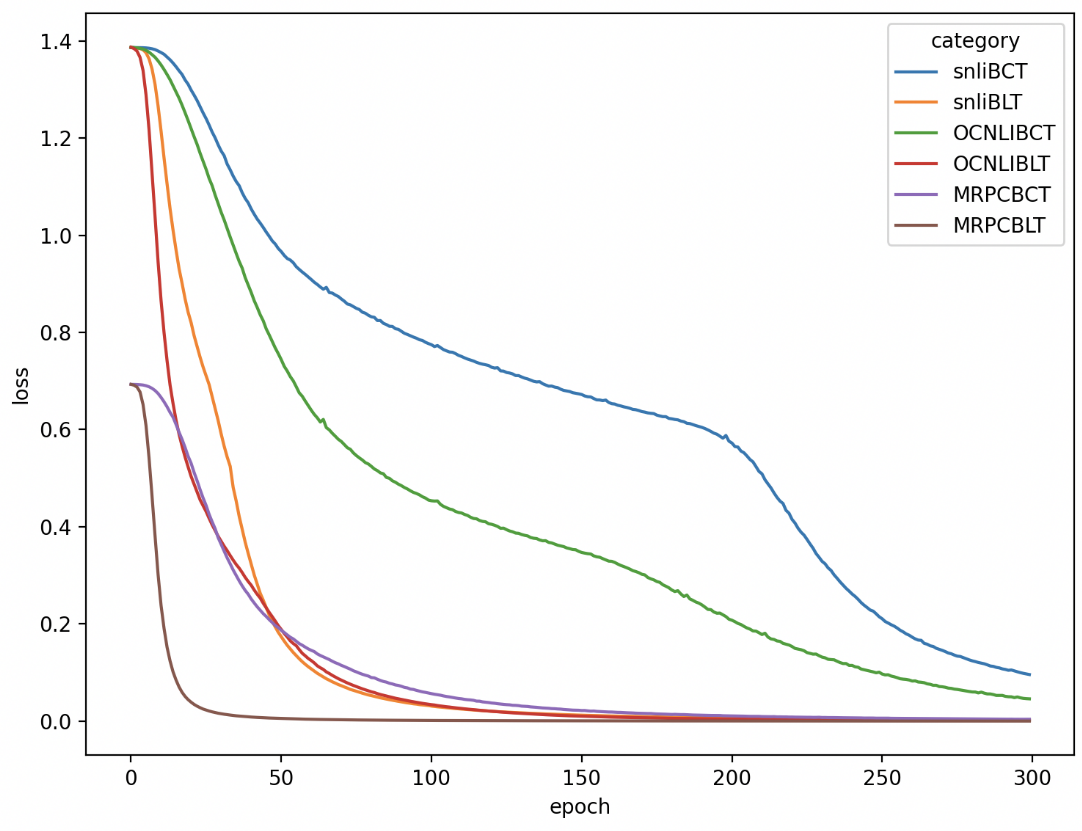

Figure 5: The loss value line graph.

5 also shows that the last layer fine-tuning method is easier for the model to reach convergence. Moreover, I tell you straight, to really achieve high performance, you need to train with the full amount of training data. And my model could easily be in the top 10 of the charts if I use the full amount of training data.

3.2 Model Results

Table 2 shows the results of the models in test datasets. For the OCNLI dataset, since no test set of the already labeled version could be found on the internet, I labeled the data in the test set directly with different models already trained on my computer separately.

Model	**test samples**	**Acc.**
snliBCT	10000	$\frac{2585}{10000}$
snliBLT	10000	$\frac{3606}{10000}$
MRPCBCT	1725	$\frac{1153}{1725}$
MRPCBLT	1725	$\frac{1152}{1725}$
OCNLIBCT	3000	$\frac{3000}{3000}$
OCNLIBLT	3000	$\frac{3000}{3000}$

Table 2: Results.

4 Dimensionality reduction analysis

In this section, I use SVD to reduce the dimension of the model's outputs. Before that, I utilize the softmax to preprocess the model's output. The Figure 6 shows the result after dimensionality reduction. The blue dots represent samples that were predicted correctly, while the yellow forks represent samples that were mispredicted.

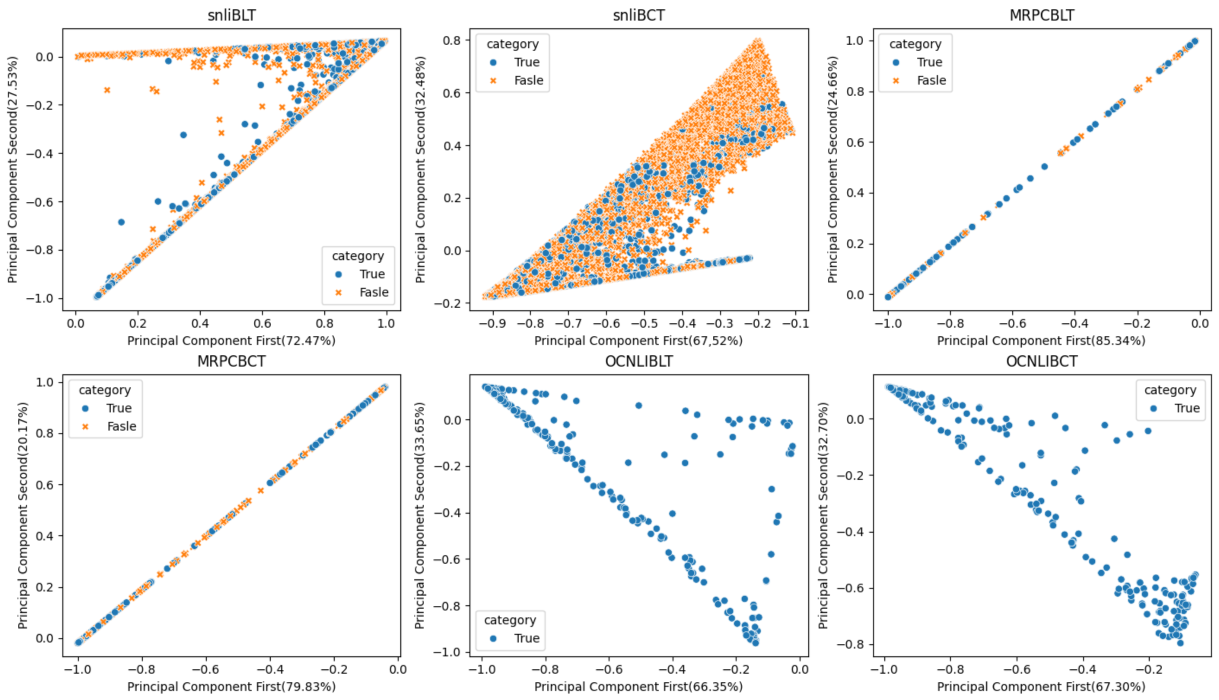

Figure 6: The SVD dimensionnality reduction analysis.

5 Distribution analysis

The figure 7 shows the model's output distribution. I first separated the model's outputs by those that were predicted correctly or incorrectly, and then displayed their distributions in the figure 7 according to the corresponding labels. The white horizontal bar in the figure 7 represents the median, and the black bolded portion represents the interquartile range.

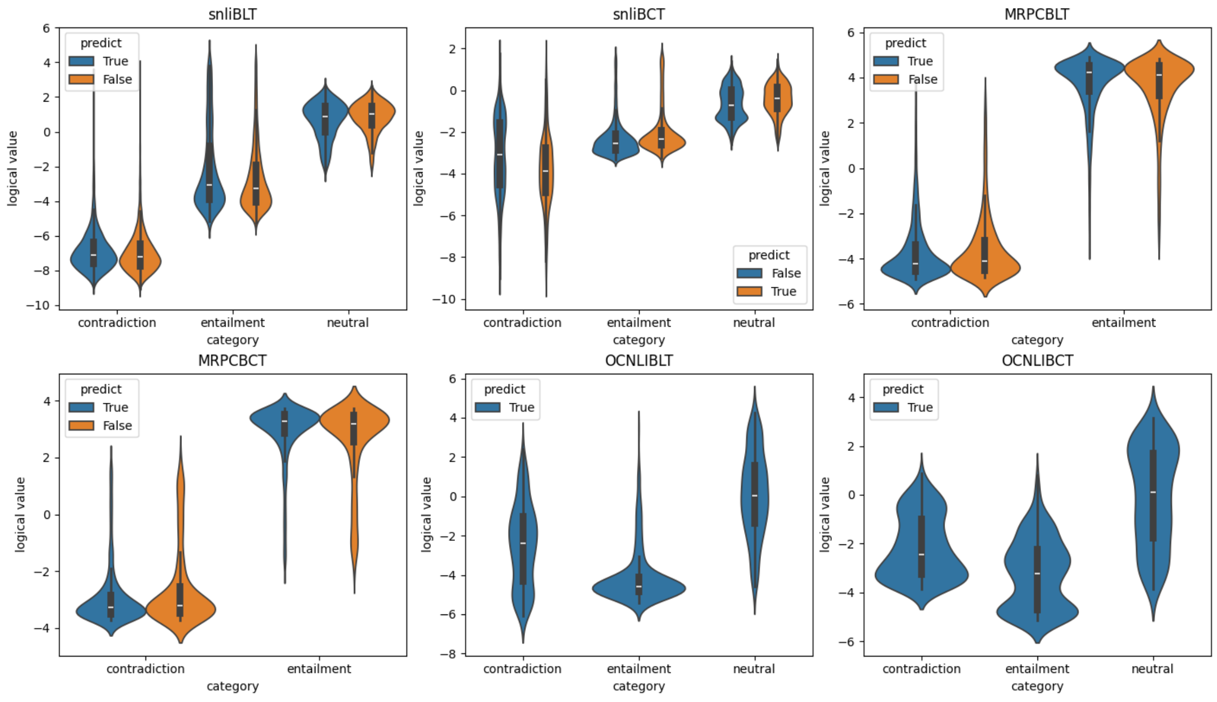

Figure 7: The distributionn analysis.

6 Discrepancy analysis

The figure 8 shows the results of the discrepancy analysis. I preprocess the model's output by softmax, and compute the predicted values of the model with the corresponding labeled values by fold change.

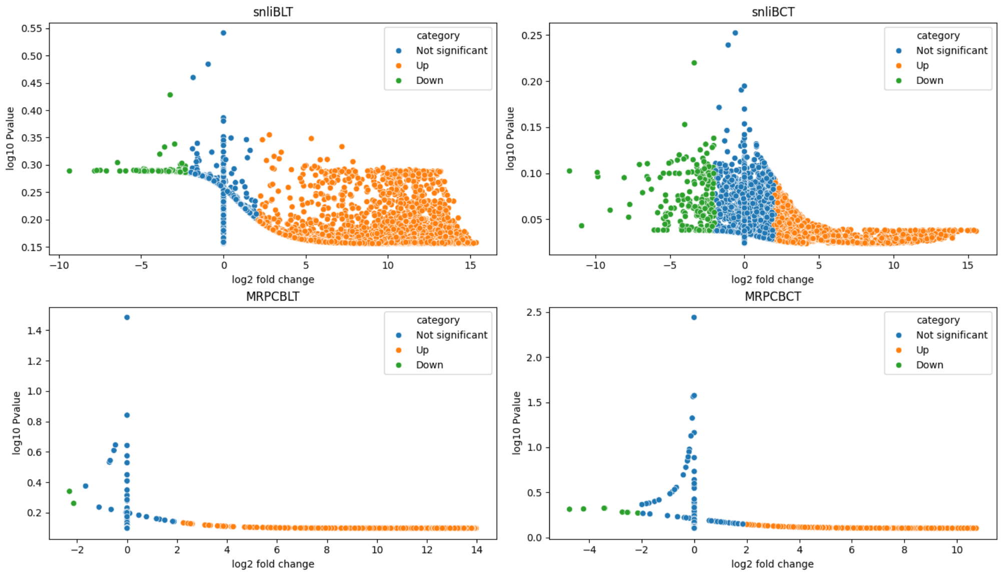

Figure 8: The discrepancy analysis.

Points marked in orange represent significant differences, green represents insignificant differences, and blue represents no differences. The vertical coordinate is the value calculated by the model through a statistically significant hypothesis test. Obviously, the reason for the high number of orange points with significant differences in the figure 8 is less training data.

7 Error case studies

In this section, I choose one error case from the snli test set and invoke bertviz to visualize it. The figure 9 shows the last layer attention-head view. More detailed aspects need to be explained through the

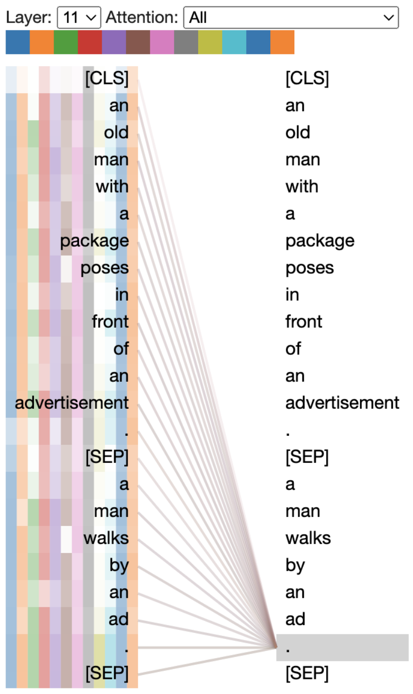

Figure 9: The attention head visualization.

source code of BertViz and the corresponding pre-trained model. The figure 10 shows a view of the last

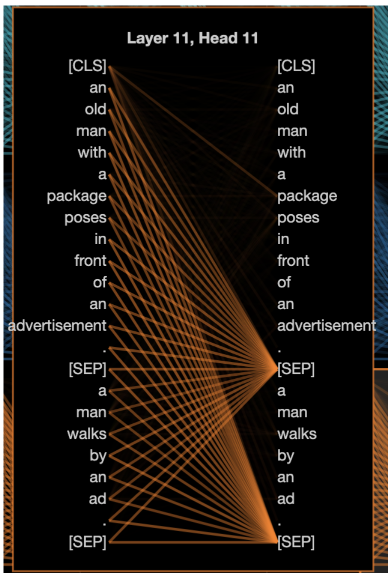

Figure 10: The visualization of last head in last layer.

head in the last layer. The figure 11 shows the neuron view visualization which include the value of the query q and the key k and the dot product of them. In summary, there are two reasons for the error in the

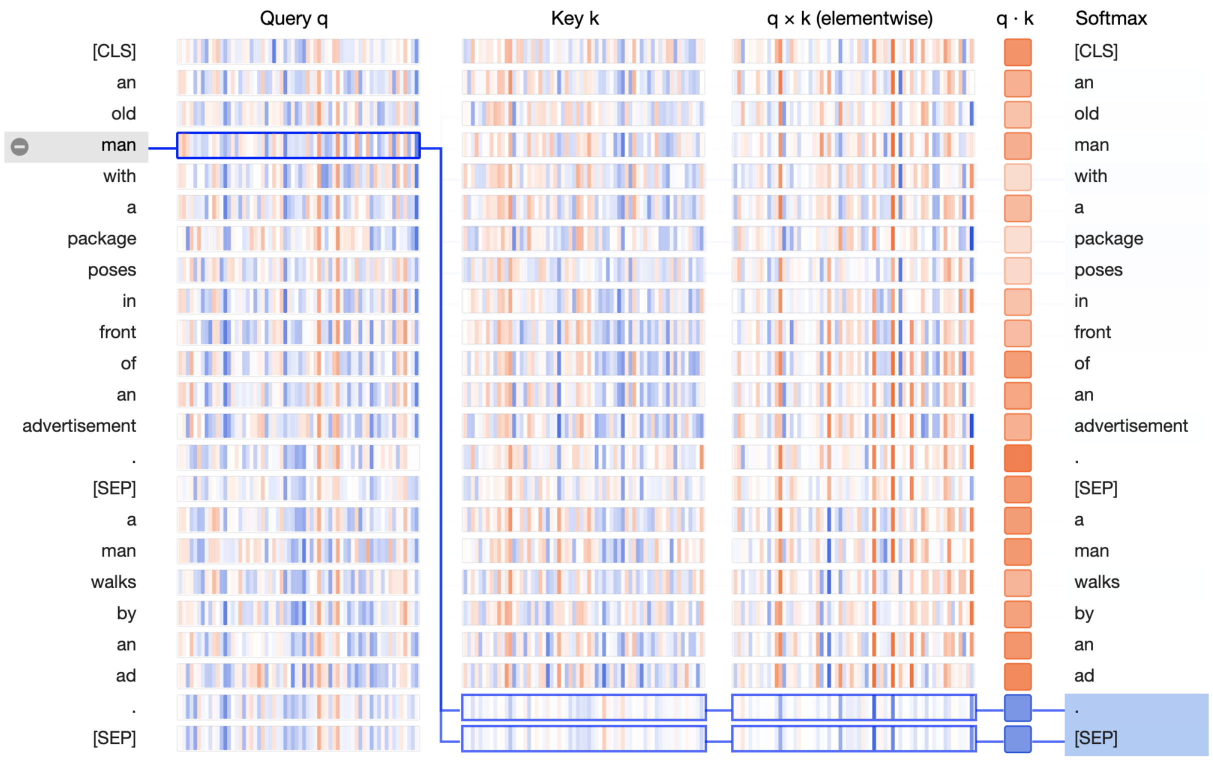

Figure 11: The neuron view visualization.

case. One is due to the effect of random numbers, which leads to incomplete training data coverage of the BERT model in pre-training. Another reason is that the Bert model's loss values are not decreasing thoroughly enough within the training step.

8 Convolution kernel visualization

The figure 12-17 show the visualization of the connvolution kernels for different models. Each convolution kernel under each model has different parameters. One of the reasons why the concept of convolution is difficult to understand is that the existing process of calculating convolution formulas is inconsistent with the mechanisms implemented in computers.

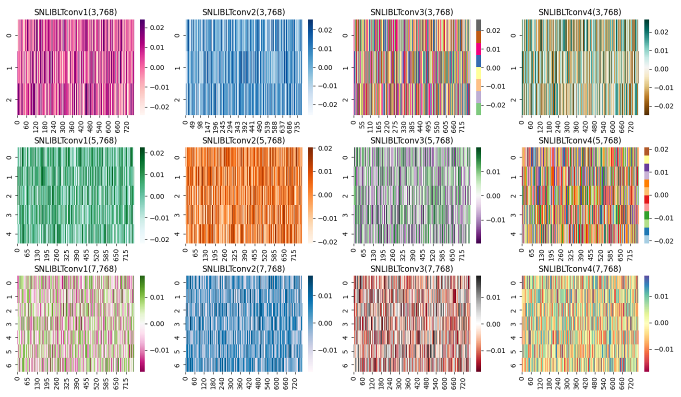

Figure 12: The snliBLT convolution kernel visualization.

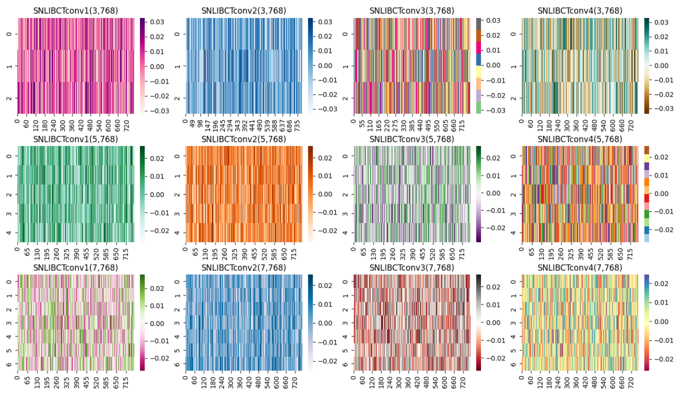

Figure 13: The snliBCT convolution kernel visualization.

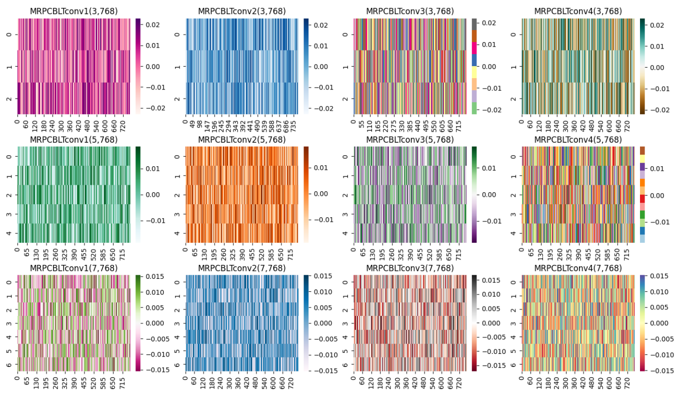

Figure 14: The MRPCBLT convolution kernel visualization.

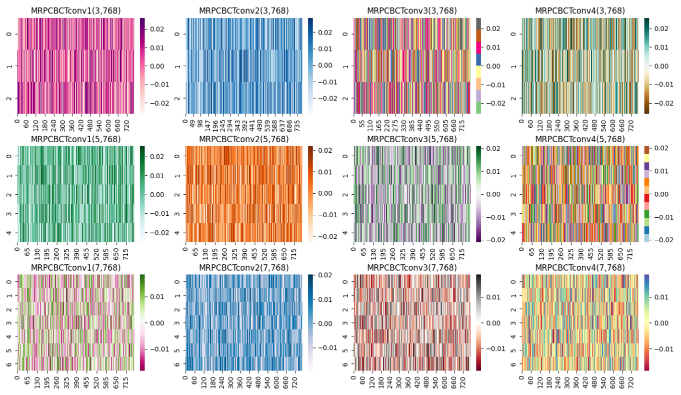

Figure 15: The MRPCBCT convolution kernel visualization.

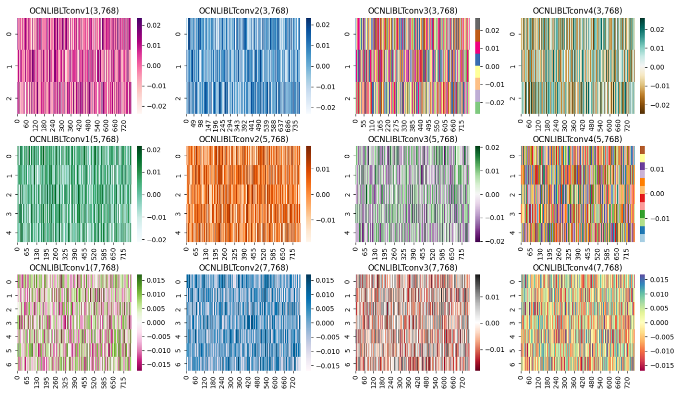

Figure 16: The OCNLIBLT convolution kernel visualization.

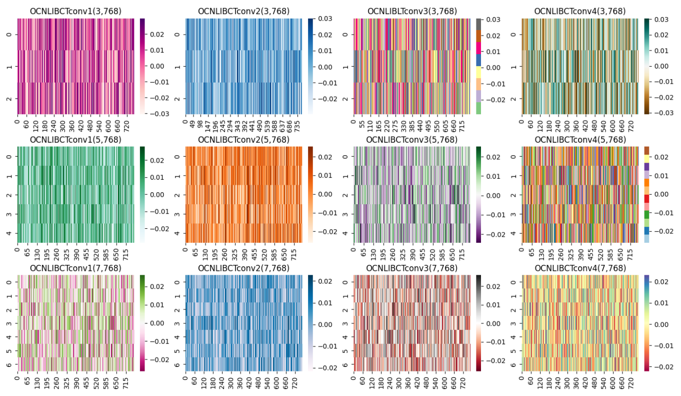

Figure 17: The OCNLIBCT convolution kernel visualization.

9 Conclusion

In the present work I have described a series of experiments with convolutional neural networks built on BERT. With just a simple few-shot training, bringing out the great detail in the CNN. My results add to the well-established evidence that the textCNN is an important ingredient in deep learning for NLP.

Acknowledgements

All experiments were done on a Macbook Pro. Thank for Microsoft and Apple. I use both Windows and Apple systems.

References

Kim, Yoon. 2014. *Convolutional neural networks for sentence classification*, preprint.

Devlin, Jacob and Chang, Ming-Wei and Lee, Kenton and Toutanova, Kristina. 2019. *Bert: Pre-training of deep bidirectional transformers for language understanding*, volume 1.

MSRParaphraseCorpus.msi. 2024. *Microsoft Research Paraphrase Corpus*, 2024.

Hu, Hai and Richardson, Kyle and Xu, Liang and Li, Lu and Kübler, Sandra and Moss, Lawrence S. 2020. *Ocnli: Original chinese natural language inference*, 2020.

MacCartney, Bill and Manning, Christopher D. 2008. *Modeling semantic containment and exclusion in natural language inference*. 521–528.