SENTIMENTAL ANALYSIS

June 2, 2019

C74 Jinyi Hu
2017011479

Exprimental Design

- **Goal**

  Given a set of Chinese with remarks by the reader for their sentiment to this news, we are aimed to trained a sentence classifier. There are 8 kind of emotions in total and we hope to train two different model using CNN and RNN neural network model, respectively.

- **Corpus**

  The corpus for training the model is from sina news

- **Model: TextCNN**

  TextCNN is a kind of algorithm for text classification task based on the CNN(Convolution Neural Network). It is designed by Yoon Kim in (Kim, 2014). Although CNN was first introduced in computer vision field, CNN shows great performance in multiple NLP tasks, especially in classification, which is similar to the image classification. The theory of Textcnn can be shown in the Figure 1.

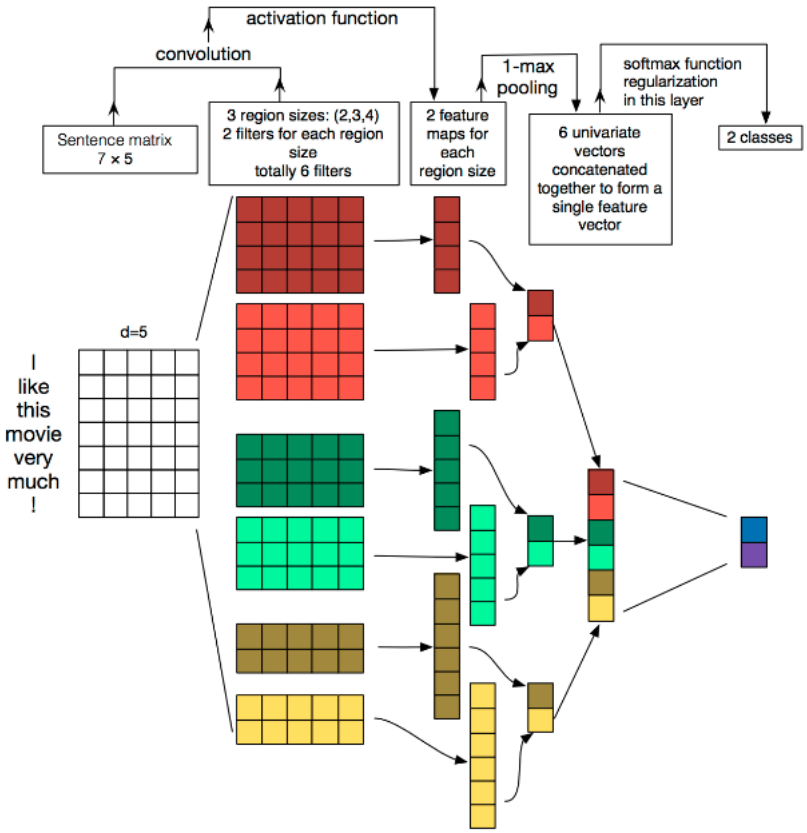

Figure 1: TextCNN

Here are the steps to train the neural network:

- Generate word vector
  Here, I used vector pre-trained based on sougou news with Word + Character + Ngram model. The dimension of each vector is 300. With these vectors, each sentence can be transformed into a matrix, which will be sent into the neural network. During the training, I set the embedding matrix as static.
- Convolution
  In this step, I use three different kernel with size of 3,4,5 as filters to extract features from the matrix. The number of each kind of kernel is 100. Then after activation function, we can get a vector which represents semantic information.
- Max Pooling
  Then we extract the max element of each vector and concatenates them together to form a single feature vector.
- Linear Layer
  We project the feature vector get from last into a vector with 8 dimension. After softmaxing, this vector can be the distribution of each vector.

The main parameters in this model: voca_size, embed_size, class_num, kernel_num, kernel_size, dropout, embed_weights(pre-trained matrix), lr, batch_size, epoch.

- **Model: LSTM**
  LSTM(Long Short-Term Memory) is a kind of time recurrent network. Due to the fact that typical is prone to gradient disappearance or gradient explosion, LSTM improves it by introducing forgetting gate. Figure 2 shows the internal structure of LSTM.

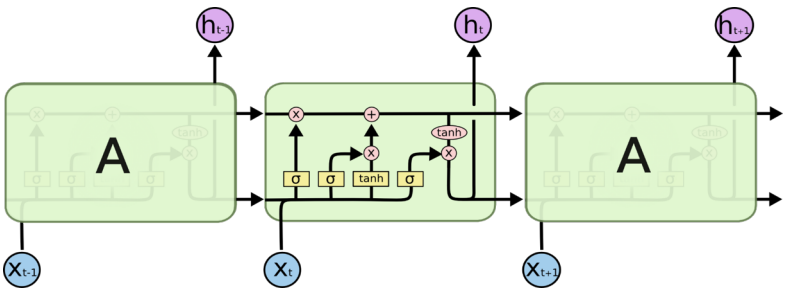

Figure 2: LSTM

Here are the steps to train the neural network:

- Read the tokenized sentence vector through the word embedding matrix which is load from the pre-trained word2vector. Then we can get the sentence matrix.

- Send the sentence matrix into a masked layer, which will mask the padding to avoid its negative influence on the trainning.

- Then the masked matrix will be sent into the LSTM layer. The final output of LSTM is what we want.
- Send the output of LSTM into a linear layer and we can get a 8-dimension vector. It's the output of the network.

The main parameters in this model: voca_size, embed_size, class_num, hidden_size, n_layers(the number of the layers of LSTM), dropout, embed_weights(pre-trained matrix), lr, batch_size, epoch.

Expriment Result

Here are the figure accuracy.

- **CNN classify**

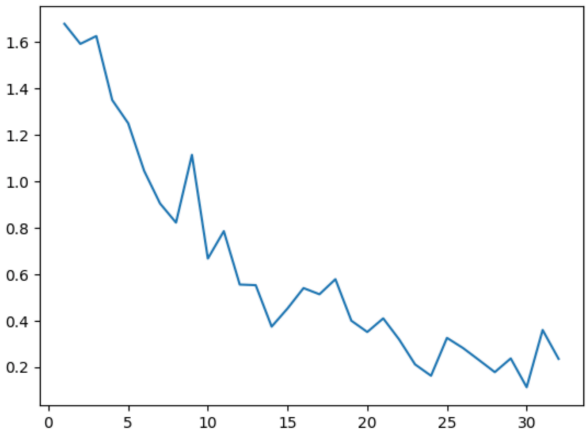

Figure 1: loss

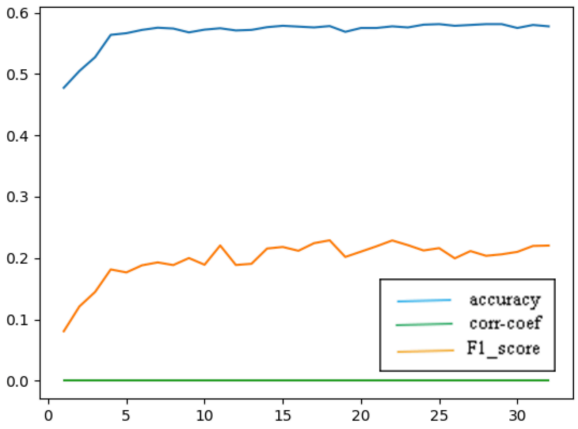

Figure 2: Evaluation index

Highest Accuracy: 59.1%
Highest F1_score: 0.21

- **CNN regression**

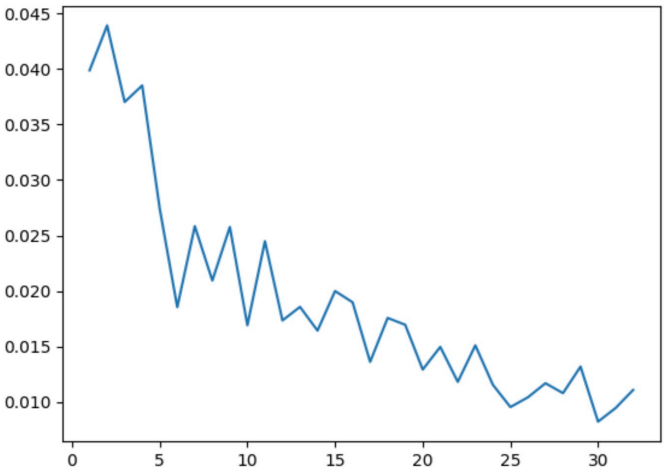

Figure 1: loss

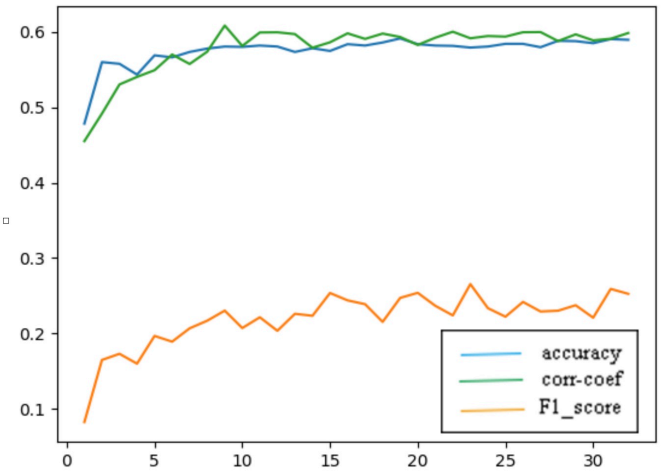

Figure 2: Evaluation index

Highest Accuracy: 58.9%
Highest F1_score: 0.26
Correlation Coefficient: 0.61

- **LSTM classify**

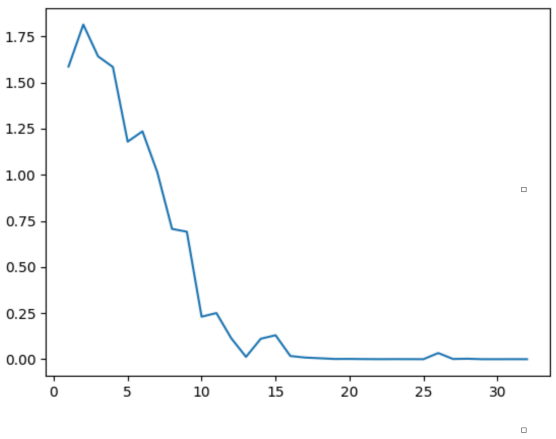

Figure 1: loss

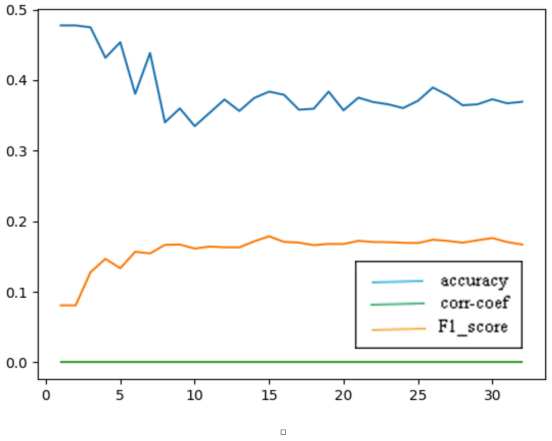

Figure 2: Evaluation index

Highest Accuracy: 49,1%
Highest F1_score: 0.27

- **LSTM regression**

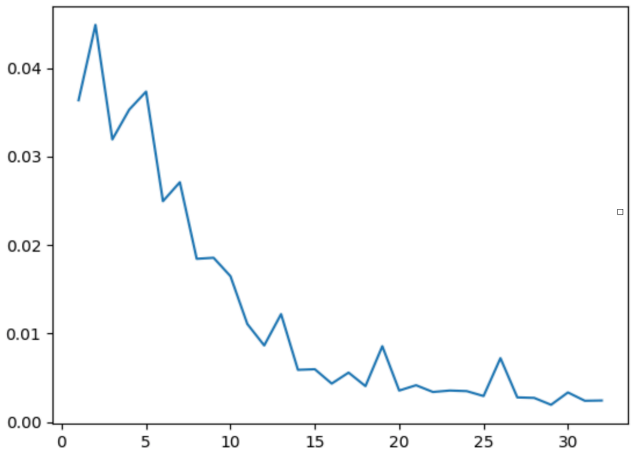

Figure 1: loss

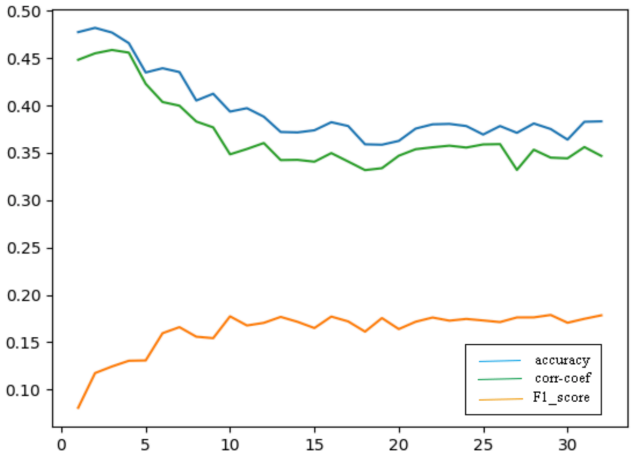

Figure 2: Evaluation index

Highest Accuracy: 48.7%
Highest F1_score: 0.25
Correlation Coefficient: 0.46

- **Result Analysis**

  Comparing the classifying and regression, the accuracy has no big difference. However, the loss of regression method is very small and regression method goes convergent faster.

  Comparing the LSTM and CNN, CNN has better performance than LSTM. The reasons will be discussed in the following part.

  Besides, we can discover some interesting result. Fox example, the figure of correlation coefficient and accuracy is quite similar.

Problem Analysis

- **Hyper Parameters**
  - **learning rate**

    The learning rate can be one of the most important parameters. Learning rate can directly influence the speed of adjustment of the model. In this experiment, I have try two ways to set the learning rate. The first is to set it static as 0.001 and the second is to make it half every 7 epoch. From the loss figure, we can see that fixed static will lead to large fluctuation with epoch increases, which is not beneficial to the convergence. So, it's a good way to use a relatively large learning rate at the beginning of the training and then decrease it with the processing of the training.

    However, in this experiment, static and non-static learning rate do not have distinctive affect on the accuracy test on the test set.
  - **batch size**

    Batch size is another important parameters for the training of the neural network. Relatively small batch size of data can increase the times of adjusting the network. For example, if we cut the total data into 100 batch, the network will be adjusted 100 times in one epoch. If we use a large batch size, then the total batch will be less than 100, and the adjusting times will be also less than 100. So if we want to increase the batch size, epoch should also be increased. However, too small batch size will lead to large fluctuation of the gradient. Extremely, if we set the batch size as 1, i.e., the gradient will go up and down, hardly go convergent. So we should avoid too small batch size. If equipment is available to support large memory, large batch size and large epoch is a good choice.
  - **CNN parameters: kernel_size**

    The kernel size can be thought to be Conceptually equivalent to the 'n' in n-gram language model. So, in general, we can choose it less than 5, because two words with distance larger than 5 can hardly have connection in semantic between each other. So in the experiment, I choose (3,4,5) three different kernels to extract features. And large kernel size, for example (5,6,7) has bad performance comparing to the (3,4,5)
  - **CNN parameters: kernel_num LSTM parameters: hidden_size**

    Although these two parameters are in different model, they play the same role

in the model: the extent to fit the training data. If they are too small, the model probably cannot fit the data. If the loss cannot go convergent, it means these two parameters are too small. It they are too large, the model can so perfectly fit the training set, in other words, over-fit the training data. It the loss can go convergent but the accuracy in test set is very low, it means these two parameters are too large.

- **Epoch**

  Epoch should adjust according to the training result. In general, we can set a large epoch and train it until the network go convergent. However, going convergent may lead to over-fitting. So we can have a test and save the model parameters after one epoch of training has done and then select the one with best result.

- **Parameters Initialization**

  - **not zero**

    Actually, not zero is just a special case. In fact, we should try to initial the parameters in every layers differently. If all the parameters in every layers are the same, then the 'deep' is meaningless because the feature extracted by each layers can be the same. So we should make them differently.

  - **random initialize**

    It's a good way to initial the parameters as a number close to 0. Although they are small, non-identical can break the symmetry of the network if initialing with 0. However, the parameters can not be too small. Otherwise, the gradient will also be very small, which is not beneficial to backword.

  - **He et al. initialization** The main problem of the random initialization is the increasing variance with the growth of the number of parameters. So we can divide a scale coefficient to normalize the variance. That is to divide the sqrt of the total number of all parameters. This initial method was introduced by He et al(He, Zhang, Ren, & Sun, 2015).

- **Over-fitting problem**

  Over-fitting problem is always a tricky problem. During training, the test accuracy often increase first and then go down. The increasing is due to the model's learning. However, if the model continuously learn, it will over fit the training set and lead to a low accuracy in test set. Also, too large size of the model may also over fit the

training data. So we can not set the hidden_size too large. Here are some method to avoid over fit during the training of the neural network.

- **Early stopping**
  Arguably, the simplest technique to avoid overfitting is to watch a validation curve while training and stop updating the weights once your validation error starts increasing.
- **Regularization**
  The simplest and perhaps most common regularization method is to add a penalty to the loss function in proportion to the size of the weights in the model. One method, called weight regularization (weight decay), is to penalize the model during training based on the magnitude of the weights. This will encourage the model to map the inputs to the outputs of the training dataset in such a way that the weights of the model are kept small.
- **Dropout**
  The key idea is to randomly drop units (along with their connections) from the neural network during training. This prevents units from co-adapting too much. During training, dropout samples from an exponential number of different "thinned" networks. At test time, it is easy to approximate the effect of averaging the predictions of all these thinned networks by simply using a single unthinned network that has smaller weights. This significantly reduces overfitting and gives major improvements over other regularization methods.

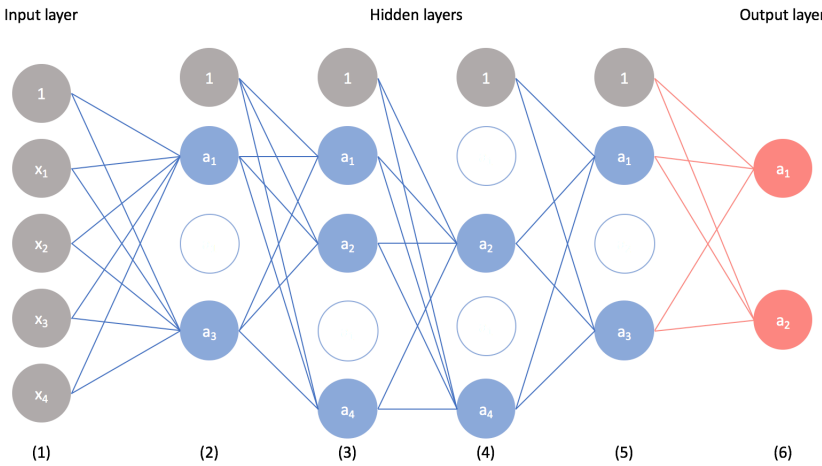

Figure 3: Dropout

- **Comparison between CNN and LSTM**
  - **Theory analysis**
    LSTM is a typical method to solve the series problem. Therefore, for a long time, LSTM and other RNN methods have been seen as the best to solve NLP task for its powerful memory capability. CNN is widely used in computer vision for its feature extraction ability. Transforming into NLP task, it can extract the feature in the sentence matrix. Comparing these two model, LSTM is better to solve mask which is rely on long-distance dependence, such as reading comprehension, machine translation, while CNN can extract more information about local features from the sentence.
  - **Accuracy**
    It's hard to directly distinguish which is better between LSTM and CNN. In this data set, CNN are far better than LSTM. One way to explain is that the emotion of news can probably be decided by one or two words. In this situation, CNN will have a better performance. However, when the emotion of the sentence is decided by the entire sentence, LSTM will be better.
  - **Running Speed**
    Comparing running speed, CNN is much faster than LSTM, for the reason that CNN can parallel operate on a sentence, while LSTM has to recurrent along the word sentence. That's also why Transformer Model has already replaced LSTM.

Discussion: How to Improve

- **Fine-tuning**

  From the figure of LSTM, the accuracy has been constantly dropped with the increase of epoch. The season for that is probably due to the non proper hyper parameters. I think the hidden size is so large that the model over fit the data at the beginning. So LSTM neea to be fine-tuned

- **Attention Mechanism**

  Attention mechanism has been widely used in NLP task. In this LSTM model, if we put a weight to the series, the model can learning to pay more attention to the word that may have great influence on the emotion of this sentence.

- **Bert**

  The state-of-art pre-trained model Bert has been considered the most powerful language model. The transformer architecture are more effective than both CNN and LSTM. So it's worth exciting to use this pre-model on this problem.

Appendix

- **Honor Code**

  In this project, I had dicussion with 陈果, 梁念宁, 林晓恩, 曹鼎原.The discussion is only on the thinking level without any code copying.

References

He, K., Zhang, X., Ren, S., & Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *CoRR*, *abs/1502.01852*. Retrieved from http://arxiv.org/abs/1502.01852

Kim, Y. (2014). Convolutional neural networks for sentence classification. *Eprint Arxiv*.