Research on Identification of Rumor Based on Different Models

Subject: Machine Learning

GROUP MEMBER

Leo 1930004003

Raven 1930026034

Justin 1930026081

1. Introduction

Background

The development of social media has accelerated the spread of information and also brought about the proliferation of rumor information, which often triggers many instability factors and has a huge impact on the economy and society. So identify false information by using the Machine Learning model becomes a necessary need. In this project, we are going to use 5 kinds of model and splits them into 3 parts, our target is to find the model with best accuracy and other discriminators, such as precision and recall.

Dataset and preprocessing

Our data set is the Chinese rumor data set from GitHub, each data is in JSON format, and they have been divided into rumor data set (number: 1538) and non-rumor data set (number: 1849).

Therefore, we first parse the data and label them as non-rumors or rumors, that is 0(non-rumor) and 1(rumor), respectively.

We then use the entire data set of rumors and non-rumors to generate a dictionary with a number for each word, which means each Chinese word gets a unique number. We use this dictionary to embed sentences to vectors which are unequal in length. As a result, for first 2 parts we have to complete them with 0's so that all the sentence vectors are of equal length, and them combine them into the matrices. For part 3, it uses another way.

For the division of training set and test set, rumor data and non-rumor data were first scrambled and combined to ensure the randomness of data, and then the index was used as the classification standard. We randomly choose 80% of the index number in mixture data set, this part is our training set and the remaining 20% is our testing set. So, we

have a total of 3,387 vectors of varying lengths, about 80% of the training set.

2. Normal Models

SVM

The first model we use is SVM, which is also used on classifying problem, by building the model and give it our training data, we got the prediction.

Model building

Base on the code we learned before, firstly we use loop to find the best C value and gamma value for this data set, after that we build SVM model based on function given by sklearn and train it with training set.

Result

By providing the testing set to the trained model, we have result below:

```
Confusion_matrix is:
 [[ 14 178]
 [  0 232]]
```

```
SVM:
Accuracy Score is: 0.5801886792452831
Precision Score is: 0.5658536585365853
Recall Score is: 1.0
F1 Score is: 0.7227414330218067
```

Confusion matrix and Score for SVM

Logistic Regression

Model building

Second one is logistic regression, which is also based on the code we have learned, here we define a regularized cost function with some other relative functions and use minimize function in optimize package to calculate minimum cost, at the same time, we use regularized gradient to update theta.

$$\text{sigmoid: } h_\theta(x) = \frac{1}{1 + e^{-\theta^T x}}$$

$$\text{gradient: } \theta_0 = \theta_0 - a\frac{1}{m}\sum_{i=1}^{m}[h_\theta(x^i) - y^i]x_0^i$$

$$\theta_j = \theta_j - a\frac{1}{m}\sum_{i=1}^{m}[h_\theta(x^i) - y^i]x_j^i + \frac{\lambda}{m}\theta_j$$

Result

By using the trained theta, we put it with testing set into predicting function and get the result below:

```
Confusion_matrix is:
 [[ 81 111]
 [ 78 154]]
```

```
Logistic Regression:
Accuracy Score is: 0.5542452830188679
Precision Score is: 0.5811320754716981
Recall Score is: 0.6637931034482759
F1 Score is: 0.619718309859155
```

Confusion matrix and Score for LR

Ensemble Learning

Lastly, we hope EL model can improve the result. Since we used two models before, here both of we'll do EL for both of them.

Model building

The EL type we choose is bootstrap, so firstly for each loop we choose index with replacement randomly, the size we choose is same with training set, then create new data set based on index and use this new data set to train model in this loop, then we store the result into a list. Totally we have 101 loops, so in the final step, we do vote by using these 101 results.

Result

For SVM`s bootstrap we have:

```
Confusion_matrix is:
 [[ 14 178]
 [  0 232]]
EL:
Accuracy Score is: 0.5801886792452831
Precision Score is: 0.5658536585365853
Recall Score is: 1.0
F1 Score is: 0.7227414330218067
```

Confusion matrix and Score for EL-SVM

For LR's bootstrap we have:

```
Confusion_matrix is:
 [[ 80 112]
 [ 75 157]]
EL:
Accuracy Score is: 0.5589622641509434
Precision Score is: 0.5836431226765799
Recall Score is: 0.6767241379310345
F1 Score is: 0.6267465069860279
```

Confusion matrix and Score for EL-LR

We find that it`s so strange that result of SVM is same as above, but by printing the result of each loop we can ensure that EL model works well, we guess it`s because the recall is 1 already, which means SVM always put non-rumors into rumors, so it`s hard to improve by voting. From the logistic regression we find that it`s result has been improved a little.

For the problem of SVM, possibly we can use AdaBoost to solve it.

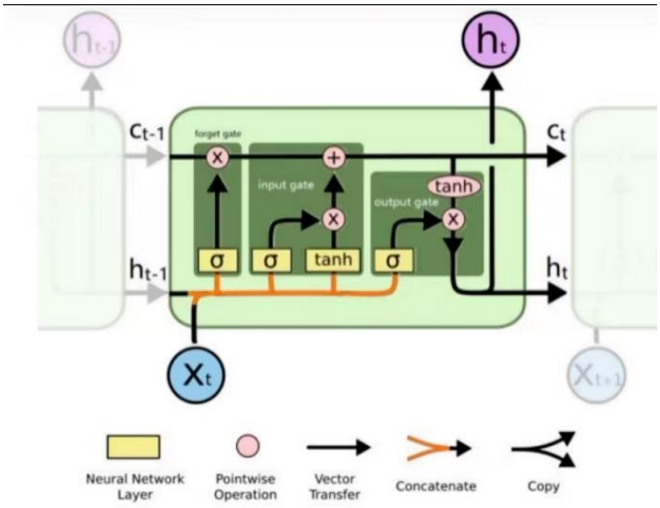
3. Long Short-Term Memory

Word Embedding

Since the length of each news is inconsistent, it cannot be directly combined into small batches. We define the preprocess function to convert the words in each text into word index through the dictionary, and then fix the length of each text by padding 0. However, since there are less than 5000 words in our dictionary, it may lead to over-fitting of the model and inaccurate prediction of the model in other data sets. For a higher accuracy, we decided to quote a pretrain word embedding model called "sgns.weibo.bigram" after insert a step of tokenization. The purpose of this step is to make the word vector transformation smoothly and accurately.

Modelling

Long short-term memory records long-term memory on the basis of recurrent neural network. In other words, when we calculate the hidden layer state, we need to use not only the current input and the hidden layer information of the previous hidden state, but also the long-term memory information of the previous state. Certainly, prior to this, a portion of the memory was considered not currently needed and would be forgotten by the forget gate before being used to calculate the current hidden layer and the current long-term memory layer.



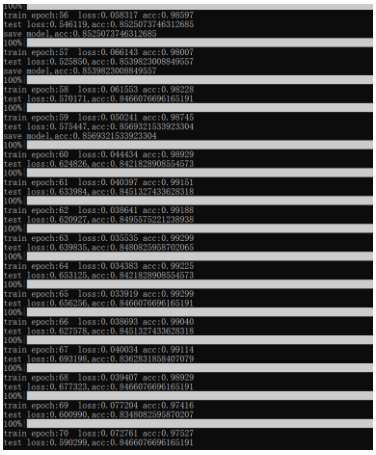
LSTM workflow in a moment

The structure of our model can be described as:

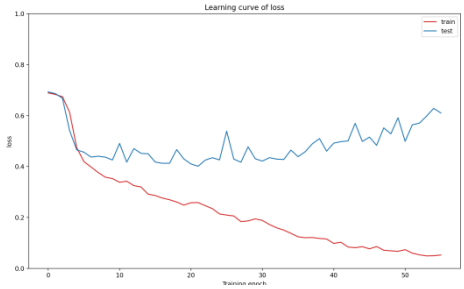
- An embedding layer: using the pretrained word embedding model to get the eigenvectors of the sentence.
- A 2-way LSTM neural network: with the embedding dimension of 300 due to the pretrained model, the hidden dimension of 128, and the number of LSTM layer of 2.
- Two fully connected neural network with the sizes of each are: [128*2, 128] and [128, 2]. As a result, the final output is 2.

Model Optimization

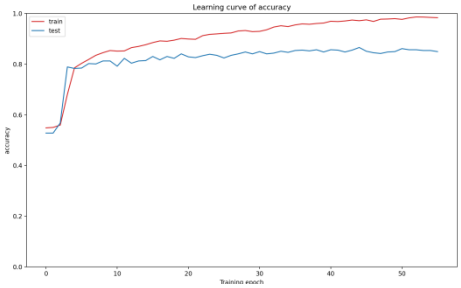
For the hyperparameters, we basically use the learning rate of 0.001, epoch number of 70 and batch size of 128. After simply defining the Cross-entropy loss function and accuracy, we train the model with the Adaptive Gradient Optimizer. In this process, once the accuracy rate exceeds 10 rounds without increasing, the program simply stops training and automatically saves the model. For each epoch, we print the training information and used the test set to calculate the current prediction loss and accuracy.



The training processing



The Learning curve of loss



The learning curve of accuracy

Model performance

We can see that after using the parameters defined by us, the accuracy of model training reaches to 0.999, and the prediction loss is reduced to 0.007. And the accuracy on the test set is 0.846. And some other indices are shown underlying. We should be able to reliably predict whether a sentence is a rumor or not. Here we can see we have successfully judge if a sentence is a rumor or not no matter it is a news or an entertainment.

```
data1 = np.int64(get_data("就没有潍坊人放不了的风筝"))
data2 = np.int64(get_data("""冰箱是保鲜小能手，但是有一些食物并不适合放进去。
                             如香蕉/火龙果/芒果等热带水果。茄子/青椒/番薯/黄瓜等蔬菜，在低温下储存
                             容易产生冻害或加速腐败。"""))
data3 = np.int64(get_data("""广州市卫生健康委副主任陈斌在发布会上将4月8日以来广州市发生的本地
                             新冠肺炎疫情的处置情况进行了通报，并指出广州本次疫情进入收尾阶段。"""))

data.append(data1)
data.append(data2)
data.append(data3)

# the shape of each sentence
base_shape = [[len(c) for c in data]]

# use the shape to generate the input data
tensor_words = fluid.create_lod_tensor(data, base_shape, place)
print(type(tensor_words))

# prediction
result = exe.run(program=infer_program,
                 feed={feeded_var_names[0]: tensor_words},
                 fetch_list=target_var)

# define the class
names = ['rumor', 'non-rumor']

# catch the label of the larger probability
for i in range(len(data)):
    lab = np.argsort(result)[0][i][-1]
    print('The label of the news is: %d.  The class is: %s.  The possibility is: %f' % (lab, names[lab],
```

```
<class 'paddle.fluid.core_avx.LoDTensor'>
The label of the news is: 1.  The class is: non-rumor
The label of the news is: 1.  The class is: non-rumor
The label of the news is: 1.  The class is: non-rumor
```

Prediction

	0	1
0	270	50
1	54	304

The confuse matrix

Improvement

As we have mentioned earlier, the embedding method for the input sentences is just transform each word to the corresponding number in the dictionary generated before, which make the whole sentence a vector. However, this means 2 obvious shortages.

Firstly, "0" makes sense in our dictionary, so supplementing vector length with 0 may distort the vector to some extent.

Secondly, the quoted word embedding model may only recognize textual similarity

rather than emotional similarity, and contextual associations may be eliminated. Examples: "完蛋了" and "芭比 Q 了".

Therefore, we decided to introduce the embedding method BERT.

4. BERT

After applying the method of the LSTM, we try to use BERT for solving this problem. The full name of the BERT is Bidirectional Encoder Representations from Transformers, which is the most successful pretraining model in NLP field. Comparing to the traditional autoregressive model which can only see the previous information, BERT applying bidirectional embeddings from language model (ELMo) and the self-attention mechanism to combine the information from globally, which is significant for better performance in the natural language processing.

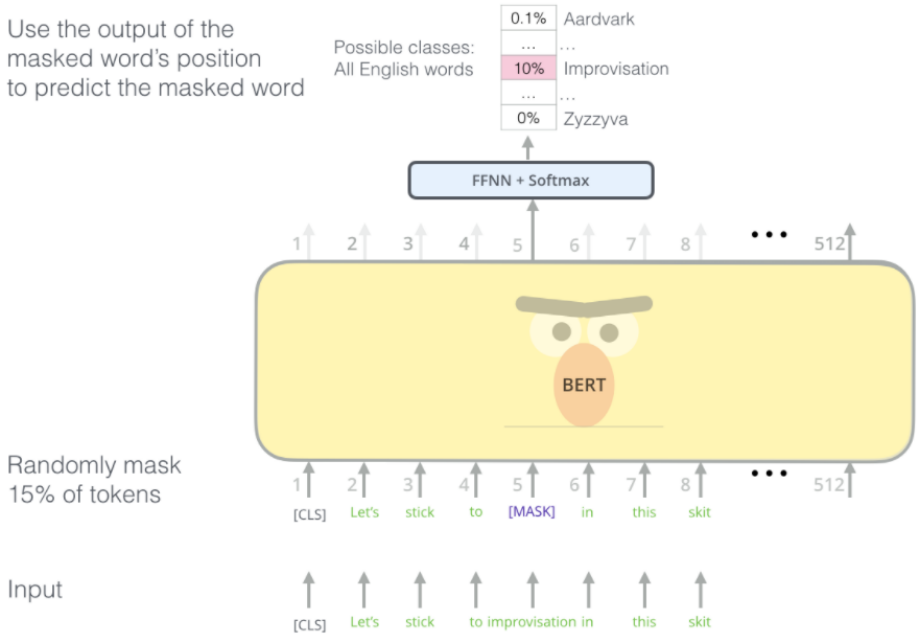
Idea

The Bert model can be separated by two parts: Pre-training and Fine-tuning

Pre-training

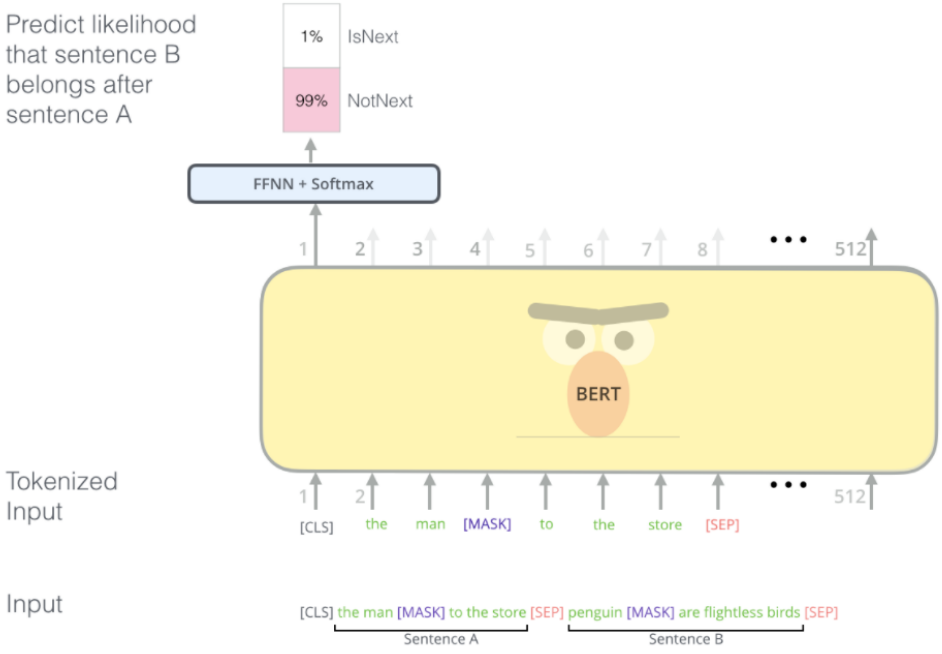
The goal of it is to generate pretrained language models. It consists of two sub-tasks in order to do that. Masked language modeling (MLM) and next sentence prediction (NSP), which are both trained by the unlabeled texts. As for the tokens, it uses [SEP] for separating the sentences. [CLS] for indicating the initial position of the sentences.

For the Masked language modeling (MLM). The model will be trained by the sentences which will selected 15% of tokens for processing. For these select token, 80% of it will be masked by the token [MASK], 10% of it will be replaced by another word and 10% of it will be kept the same. The aim for this model is to predict the mask word by Artificial Neural Network and SoftMax function.



Graph for MLM

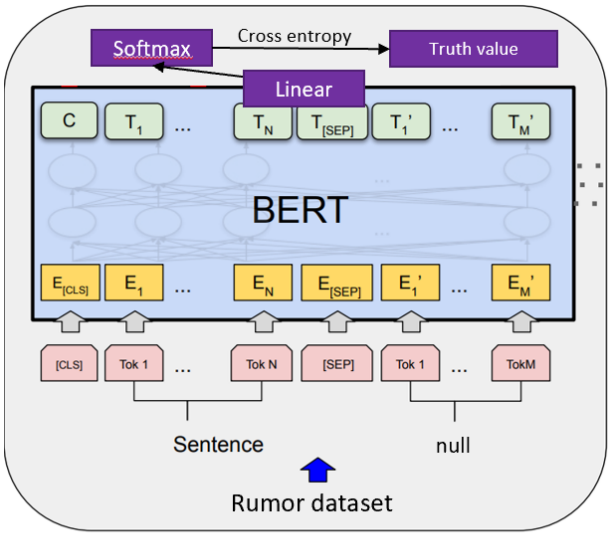
For the Next Sentence Prediction (NSP). The model will be trained by the sentences pair. For these sentences pair (A and B), fifty percent of B is the next sentence of A. And the aim for this model is to predict whether the sentence is the real next sentence. This task can let the model have a better understanding on the relationship of the sentences by doing a binary classification.



Graph for NSP

Fine-tuning

The fine-tuning step is used to train the model for the specific problem, which can also mean to let the model focus on the downstream task. Since our target is to predict whether the weibo message is a rumor. The training sentence will used as the input sentences. After embedding and the data go through dense layer, linear layer and SoftMax function. The model will give a Boolean result for the truthiness of the sentence.



Fine-tuning

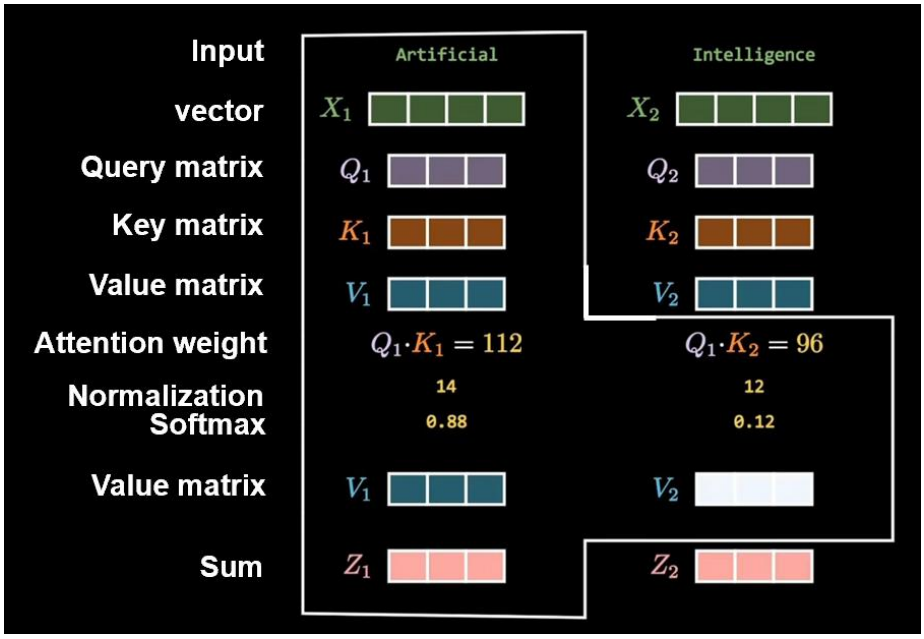
Important concept

Multi-Head Attention

Since the BERT is derived from the encoder part of the transformer. And the core of the encoder is the multi-head attention mechanism, which is consist for multiple operation of the self-attention.

Self-attention is aim to insert the relationship data between each word and position data to the original vector which we obtain from the input sentence. The procedure is firstly covert the sentence into the vector, after we insert the position data to the vector. For single vector X, we use it to make product with the Q, K, V matrix to get the

Q_X, K_X, V_X, which is the weighted matrix we need to train and update. We then get the attention weight by make the dot product for Q_X with every K_X generate by other vector. Then we normalize the attention weight by divided the square root of the length of data and process by SoftMax function and product with Value matrix. We can finally get the Z which is the same length as the input vector and insert the attention weight.

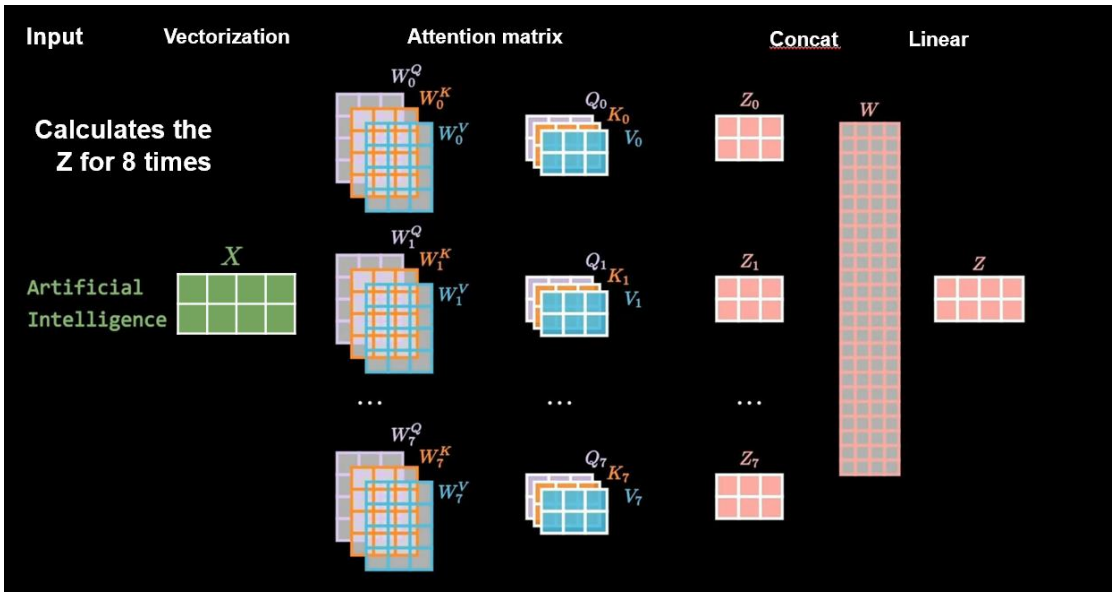


Graph for self-attention

The following are the calculation by applying matrix. which can be represented as:

$$softmax\left(\frac{Q_X \times K_X^T}{\sqrt{d_K}}\right)V_X = Z$$

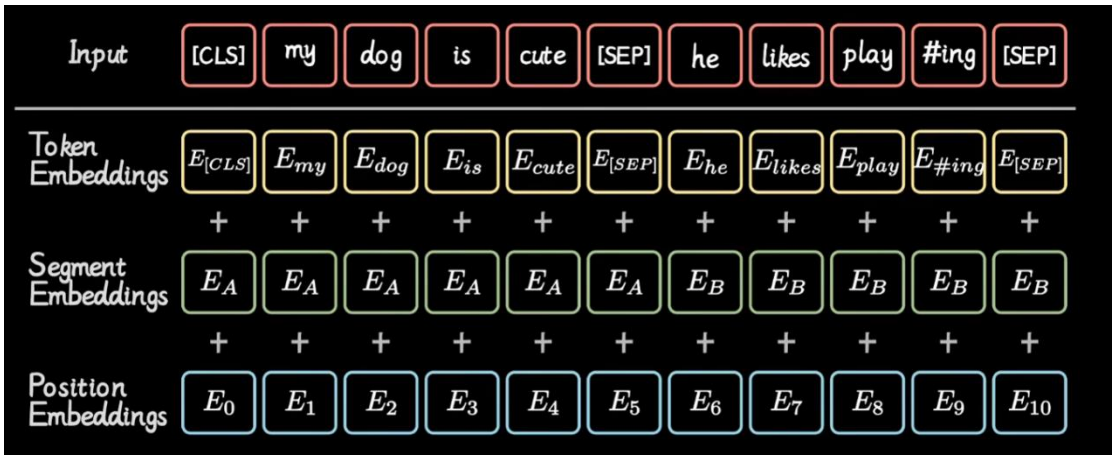
Base on the self-attention, what the multi-head attention do is calculating the Z for 8 times and then concatenate every Z matrix into a weighted sum Z, which can eliminate the influence of the initial value of Q, K, V matrix.



Graph for multi-head attention

Sentence Embedding

The embedding part is aim to changing the sentences after tokenization to the vector. The processing method is simple. Just adding the token embedding, segment embedding and position embedding together by applying broadcast. Which is shown as follows:



Graph of Sentence Embedding

The token embedding layer is to convert each word into a vector of fixed dimensions. In BERT, each word is converted into a 768-dimensional vector representation

Segment Embeddings imply whether the sentence is the first or the second one.

Position embeddings imply the position of each word.

Implementations

For the coding, we use the bert-base-chinese from Hugging Face, which is the package transforming the Chinese sentences into the vector by the method of one word to a corresponding vector.

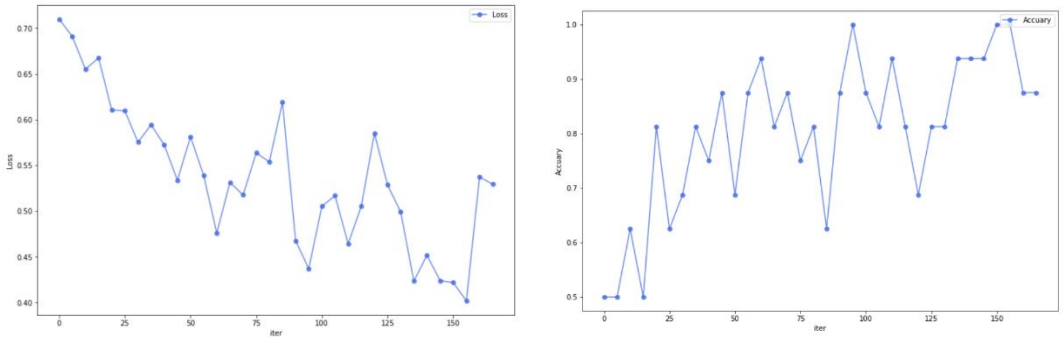
We set the batch size as 16, the max length of the vector as 500. And in this model, each Chinese word is converted to ta 768 dimensions.

After we transfer the sentence into the vector which inset the positional data as well as the meaning relationship between each word. We padding 0 or cutting information which exceed the max length to make the length of the vector be the same, and we get the attention mask which indicate the valid position of the word.

Then we apply the fully connected neural network for training. After training, we get the final model.

Performance

When we are training the model, we output the loss and the accuracy for each epoch, the graph is shown as follows, the left one is the loss, right one is the accuracy:



Line chart for loss and accuracy

The following is the performance of our model in the test set, including confusion matrix, accuracy, recall, precision, and F1-score. which is perform relatively satisfied.

```
Confusion_matrix is:
 [[242  63]
 [ 37 330]]
```

```
Accuracy Score is: 0.8511904761904762
Precision Score is: 0.8396946564885496
Recall Score is: 0.8991825613079019
F1 Score is: 0.8684210526315788
```

Confusion matrix and score for BERT

We can find that the line chart of loss and accuracy is not that smooth, it is possible that our training dataset is relatively small and not trained enough, so we can improve our model in these areas in the future.

5. Conclusion and Exploration

For this project, we apply the Linear regression, SVM, EL with SVM and Linear regression, LSTM and Bert for the weibo rumor detect, the LSTM and the Bert have better performance.

	SVM	LR	EL-SVM	EL-LR	LSTM	BERT
Accuracy	0.580	0.554	0.580	0.559	0.844	0.861
Precision	0.566	0.581	0.566	0.584	0.840	0.840
Recall	1.0	0.664	1.0	0.678	0.840	0.923
F1 Score	0.723	0.620	0.723	0.627	0.840	0.879

Score for each model

While there is still something we can make further improvements. For example, for the part of basic model, we can improve the embedding part by applying Bert instead of changing the word to number directly. Besides, we can use other advanced model after the word embedding by BERT. For the data itself, there is no enough data for us to have a better training, so we can also collect more rumor data used for training.

6. Reference

1. John Duchi. *Adaptive Subgradient Methods for Online Learning and Stochastic Optimization.* Journal of Machine Learning Research 12 (2011) 2121-2159

2. Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, & Christopher Potts. *Learning Word Vectors for Sentiment Analysis*. Stanford, CA 94305

3. Y. Bengio, R. Ducharme, P. Vincent, & C. Jauvin. 2003. *A neural probabilistic language model.* Journal of Machine Learning Research, 3:1137–1155, August

4. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). *Attention is all you need.* Paper presented at the, 2017- 5999-6009.

5. Jacob, D., Ming-Wei, C., Kenton, L., & Kristina T. (2018). *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding.*

6. Song, C., Tu, C., Yang, C., Liu, Z., & Sun, M.,(2018). *CED: Credible Early Detection of Social Media Rumors.*