Neural Networks 2019: Assignment 2

Abstract

There are two main parts in this report. One is a `proof-of-understanding` of three types of neural networks(`CNNs, RNNs, AutoEncoders`), another is an application and comparison of neural networks in **Sentiment Analysis of Short Texts**. Sentiment analysis of short texts is challenging because of the limited contextual information they usually contain. In recent years, deep learning models such as convolutional neural networks (CNNs) and recurrent neural networks (RNNs) have been applied to text sentiment analysis with comparatively remarkable results. In this project, we use `CNNs`, `RNNs` and a combination of `CNNs` and `RNNs` to classify the short texts. These neural networks show a good performance on experiments with SST1 benchmark corpora.

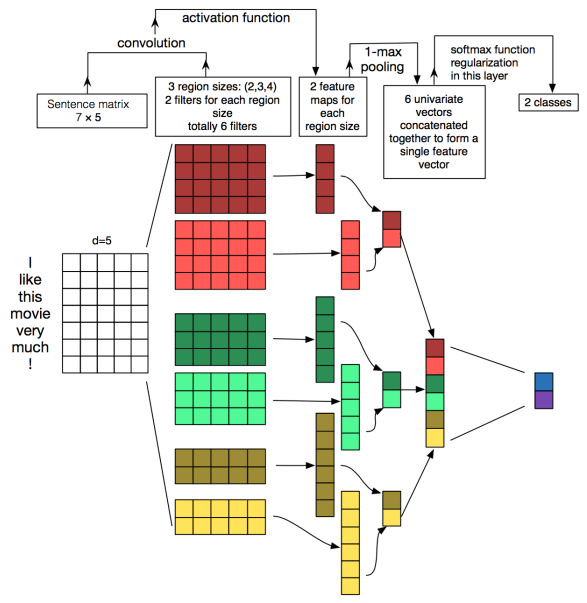
1. Proof of Understanding

1.1. CNNs

1.1.1. Introduction

When we talk about Convolutional Neural Networks(CNNs), we typically think of Computer Vision. CNNs were responsible for major breakthroughs in Image Classification and are the core of most Computer Vision systems today, from Facebooks automated photo tagging to self-driving cars.(2) More recently people also started to apply CNNs to problems in Natural Language Processing(NLP) and gotten some fairly nice performance.

For NLP, the inputs are not pixels but documents or sentences represented as a matrix. Each row of the matrix is a word which represented as a vector. People use `word2vec` or `GloVe` to index words into vectors. The workflow shows in Fig.1 (2). So if you have a 7-words sentence, the matrix will be $7 * n$ where n is the dimension you selected when map index into vectors. For this case, we use 3 different size filters, each of which has 2 filters. Every filter performs convolution on the sentence matrix and generates (variable-length) feature maps. Then 1-max pooling is performed over each map, i.e., the largest number from each feature map is recorded. Thus a univariate feature vector is generated from all six maps, and these 6 features are concatenated to form a feature vector for the penultimate layer. The final softmax layer then receives this feature vector as input and uses it to classify the sentence; here we assume binary classification and hence depict two possible output states.



activation function
convolution
1-max pooling
softmax function regularization in this layer
Sentence matrix 7 × 5
3 region sizes: (2,3,4) 2 filters for each region size totally 6 filters
2 feature maps for each region size
6 univariate vectors concatenated together to form a single feature vector
2 classes
d=5
I
like
this
movie
very
much
!

Figure 1. Illustration of a CNN architecture for sentence classification.

1.1.2. Experiments:Text Classification

The dataset we used is the **Moives Reviews from Rotten Tomatoes**, and it contains 10,662 example labeled review sentences, half positive and half negative. The dataset has around 20k vacabulary. We use 90% to train the model and use 10% as the validation dataset.

After 3,000 iterations, the results almost converged which shown as Fig.2. The accuracy on train dataset is around 1 and on validation dataset is around 0.74.

1.2. RNNs

1.2.1. Introduction

A recurrent neural network (RNN) is a class of artificial neural network where connections between nodes form a directed graph along a temporal sequence. This allows it to exhibit temporal dynamic behavior. Unlike feedforward neural networks, RNNs can use their internal state (memory)

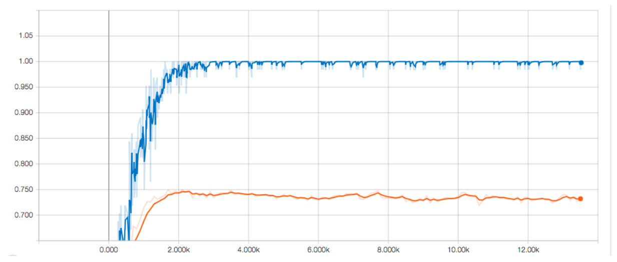


Figure 2. Accuracy with number of iterations.

to process sequences of inputs. This makes them applicable to tasks such as unsegmented, connected handwriting recognition or speech recognition.

The term `recurrent neural network` is used indiscriminately to refer to two broad classes of networks with a similar general structure, where one is finite impulse and the other is infinite impulse. Both classes of networks exhibit temporal dynamic behavior. A finite impulse recurrent network is a directed acyclic graph that can be unrolled and replaced with a strictly feedforward neural network, while an infinite impulse recurrent network is a directed cyclic graph that can not be unrolled.

Both finite impulse and infinite impulse recurrent networks can have additional stored state, and the storage can be under direct control by the neural network. The storage can also be replaced by another network or graph, if that incorporates time delays or has feedback loops. Such controlled states are referred to as gated state or gated memory, and are part of long short-term memory networks (LSTMs) and gated recurrent units.

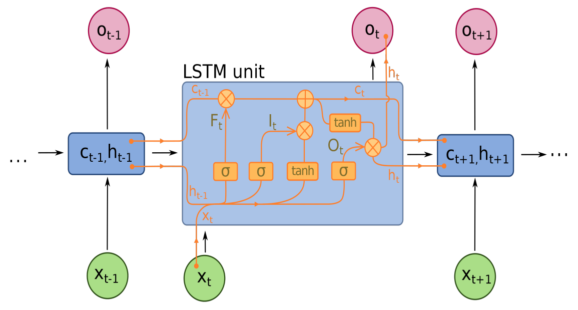


Figure 3. LSTM

1.2.2. Experiments:Sentiment Analysis

We rerun a recurrent neural network that performs sentiment analysis from github(5). RNN always performs good in this kind of task, because it includes information about sequence of words. The author uses a dataset of movie reviews, accompanied by labels. The size of the dataset is 32.1MB, which has 25000 reviews. The corresponding labels only consist of positive and negative labels, so it's actually a binary classification task. The architecture for this network is shown below:

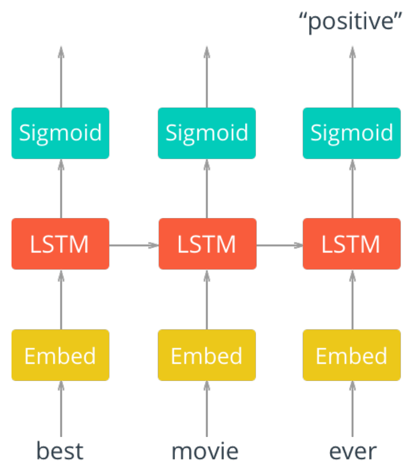


Figure 4. structure

At first, it pass the words into an embedding layer. Use word embedding rather than one-hot encoded vectors to represent the text is more efficient. The author use word2vec to train the embedding layer, which .

From the embedding layer, the new representations will be passed to LSTM cells. These will add recurrent connections to the network so we can include information about the sequence of words in the data. Finally, the LSTM cells will go to a sigmoid output layer here. We're using the sigmoid because we're trying to predict if this text has positive or negative sentiment. The output layer will just be a single unit then, with a sigmoid activation function.

We don't care about the sigmoid outputs except for the very last one, we can ignore the rest. We'll calculate the cost from the output of the last step and the training label.

This experiment only use accuracy as its evaluation method, and we got 0.81 in the reruning result of test dataset, which is 0.02 lower than the accuracy given in the github, the reason may be the random initialization function makes the initial weights are not always the same.

1.3. Autoencoders

1.3.1. Introduction

With the increasement of data size, it's more and more difficult for neural network to tackle big input data like a very clear picture with millions of pixels. Therefore, it's import to reduce the size of input data which can help neural network learn and predict more quickly, also called dimensionality reduction.

An autoencoder is a type of artificial neural network used to learn efficient data codings in an unsupervised manner. The aim of an autoencoder is to learn a representation (encoding) for a set of data, typically for dimensionality reduction, by training the network to ignore signal noise. Along with the reduction side, a reconstructing side is learnt, where the autoencoder tries to generate from the reduced encoding a representation as close as possible to its original input, hence its name. Recently, the autoencoder concept has become more widely used for learning generative models of data. Some of the most powerful AI in the 2010s have involved sparse autoencoders stacked inside of deep neural networks.

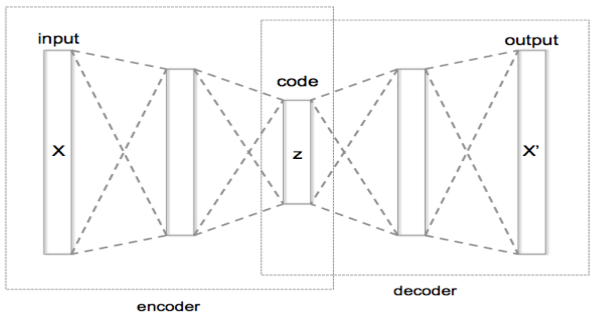


Figure 5. Autoencoder

In the encoder part, the original data is compressed to a smaller size of data. In the decoder part, the compressed data will be uncompressed to the same size data as original. It's easy to get the difference between the original data and uncompressed data by comparing them. The more likely they are, the more successful this encoder is. To increase the accuracy of representation of original data, we need to reduce the difference between original data and uncompressed data. Once the original data and uncompressed data are similar in someway, we can use the encoder part as our final autoencoder. So the decoder part is only useful in the training process, but encoder is what will be used in data compression.

1.3.2. Experiments:Place recognition with WiFi fingerprints

We rerun an project in (7) which use autoencoder to reduce dimensionality. This project use Tensorflow to implement model discussed in the paper *Place recognition with WiFi fingerprints using Autoencoders and Neural Networks* (3).

Using WiFi signals for indoor localization is the main localization modality of the existing personal indoor localization systems operating on mobile devices. WiFi fingerprinting is also used for mobile robots, as WiFi signals are usually available indoors and can provide rough initial position estimate or can be used together with other positioning systems. Currently, the best solutions rely on filtering, manual data analysis, and time-consuming parameter tuning to achieve reliable and accurate localization. In this paper, they propose to use deep neural networks to significantly lower the work-force burden of the localization system design, while still achieving satisfactory results. Assuming the state-of-the-art hierarchical approach, they employ the DNN system for building/floor classification. They show that stacked autoencoders allow to efficiently reduce the feature space in order to achieve robust and precise classification. Below is the architecture of their autoencoder model.

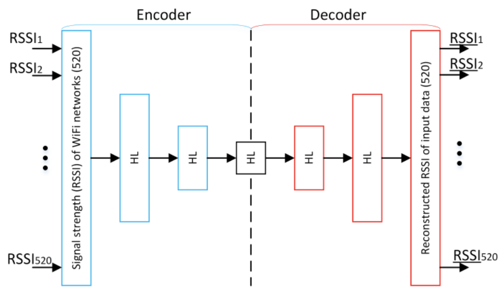

Encoder
Decoder
RSSI1
RSSI2
RSSI520
Signal strength (RSSI) of WiFi networks (520)
HL
HL
HL
HL
HL
Reconstructed RSSI of input data (520)
RSSI1
RSSI2
RSSI520

Figure 6. Autoencoder for classsification

Below is the architecture of combination of autoencoder and classifier:

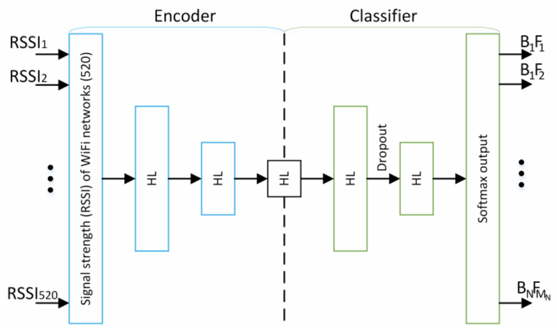


Figure 7. Autoencoder and classifier

They use tensorflow functions to construct encoder training and classifier training directly, and finally in reruning the script of their neural network, we got same accuracy in test set which is 0.92.

2. Sentiment Analysis of Short Texts

2.1. Introduction

Sentiment analysis refers to the use of natural language processing, text analysis, computational linguistics, and biometrics to systematically identify, extract, quantify, and study affective states and subjective information.

Sentiment analysis of short texts is challenging because of the limited contextual information and the sparse semantic information they normally contain. With the development of deep learning, typical deep learning models such as word embeddings, CNNs and RNNs have been applied to text sentiment analysis and gotten remarkable results.

At first, we implement the model proposed by the paper (8) which is a model that focus on solving sentiment problem and perform pretty well. They present a jointed CNN and RNN architecture that takes the local features extracted by CNN as input to RNN for sentiment analysis of short texts. They take the word embeddings as the input of CNN model in which windows of different length and various weight matrices are applied to generate a number of feature maps. After convolution and pooling operations, the encoded feature maps are taken as the input to the RNN model. The long-term dependencies learned by RNN can be viewed as the sentence-level representation. The sentence-level representation is taken to the fully connected network and the `softmax` output reveals the classification result.

After that, we implement single layer CNNs and single layer RNNs to try to solve the same problem. We use SST1 benchmark dataset to train and test these three different models. According to the experiment results, all three kinds of neural networks perform well, but the best model may differ in different evaluation methods.

2.2. Background

In this section, we will talk about some basic conceptions and techniques we used to build the models. We start from word embedding, it's the first step in building our models. Then, we introduce the optimization algorithm and loss functions used in our model, which play an import role in improving the performance of models.

2.2.1. Word Embedding

Word embedding is a neural network based distributed representation of word, it can be regarded as a task of word vectorization. Essentially, there are two main methods for word embedding, one is `Word2Vec` and another one is `GloVe`. `Word2Vec` is a "predictive" model, whereas `GloVe` is a "count-based" model.

When we control for all the training hyper-parameters, the embeddings generated using the two methods tend to perform very similarly in downstream NLP tasks. The additional benefits of `GloVe` over `word2vec` is that it is easier to parallelize the implementation which means it's easier to train over more data, which, with these models, is always a good thing.

In this project, we use `GloVe` pre-trained model [1] for all the experiments(4). In the pre-trained `GloVe` model we used, each word is represented by 100 dimensional vector, and there are 400,000 words in total which are trained based on Wikipedia.

2.2.2. Optimization Algorithm

Adam is an optimization algorithm that can be used instead of the classical stochastic gradient descent procedure to update network weights iterative based in training data. In fig.8, we can see Adam performs the best on **MNIST** dataset among these different kinds of algorithms. In this project, we used Adam as our optimization algorithm.

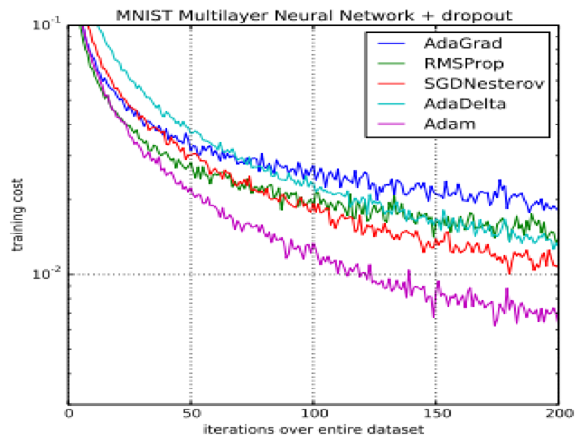


Figure 8. Comparison between different optimization algorithms

2.2.3. Loss Functions

For supervised learning, a loss function is required to be minimized in order to get the optimal values of the variables. MSE, MAE or Cross-entropy loss, etc are often choosed as loss functions. Here we use **Cross-entropy loss** as our loss function.

As for binary classification, where the number of classes M is 2, cross-entropy can be calculated as below:

$$Loss = -(y \log(p) + (1-y)\log(1-p)) \quad (1)$$

Here y is the label of a sample, it's either 1 or 0; and p is the probability of this sample to be predicted as positive. If $M > 2$ (i.e. multi-classification), we calculate the loss function for each sample as below:

$$Loss = -\sum_{c=1}^{M} y_{o,c} \log(p_{o,c}) \quad (2)$$

[1]The pre-trained model can be found on `https://github.com/stanfordnlp/GloVe`, here we use `glove.6B.100d.txt`.

where M is the number of classes, y is binary indicator (0 or 1) if class label c is the correct classification for observation o, p is predicted probability observation o is of class c.

2.3. Model

We implemented 3 different kinds of models to solve **Sentiment Analysis** problems. The main ideas of these 3 models are using `CNN`, `RNN` and a combination of `CNN` and `RNN` to receive short texts' word embeddings as input from word embedding layer, and then classify these vectors to different sentiment classes. In this section, we are going to dive into details to describe these 3 models and will also give a interpretation of the number of parameters in different models in the end of this section.

The input of a neural network should be a group of vectors. However, the dataset we are using is not vectors originally. We need somehow to convert these texts to vectors. So in the head of our models, we add a word embeddings layer to do this work for us. Here, we use the pre-trained `GloVe` model to map texts into vectors format. Below is an intuitional example of movie review word embedding matrix:

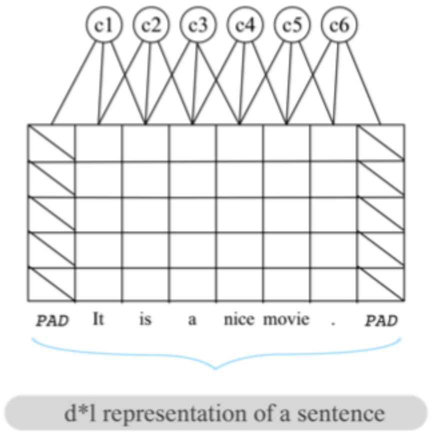


Figure 9. A sentence matrix with padding

From this example we can see that each word is represented as a 5-dimensional vector, but this example is just to illustrate the word embedding structure. In our Glove model, each embedded word is represented in a 100-dimensional vector. Now we will introduce the main parts of these three models.

2.3.1. GLOVE+CNN

The main structure of CNN model is given in Fig 10. After we get the word embedding result, we apply the CNN model to classify short texts. First, we use a convolutional layer to extract local features of texts, and then use a maxpooling layer to decrease the dimension of texts. After that, we use another convolutional layer to exact features from the output of max-pooling layer again. Finally, we flat it and connect it to a dense layer to make the final decision of classifications.

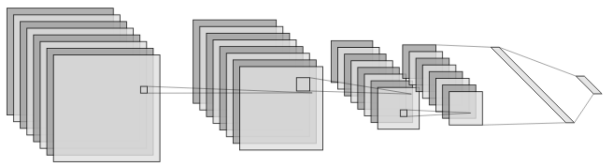


Figure 10. The structure and the number of parameters of CNN model

In fact, the word embeddings are not directly send to the convolutional layer. We feed these embedded vectors into a `Dropout` layer before convolution, which will drop some connections between neurons randomly to decrease the risk of over-fitting. Then, we use the output of `Dropout` layer as the input of our `Convolution` layer, whcih contains 200 filters and the size of each of them is 4x4, meanwhile, we use `Relu` as the activation function of this layer. Next, we add a `MaxPooling` layer on it, and the pool size is 2, which means it will cut down a half amount of the vectors. Below is an illustration of pairwise max pooling:

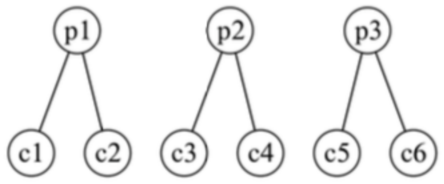


Figure 11. Pairwise max pooling operation on a scaling of 2

In this figure, the below nodes are the vectors generated from word embeddings after convolution action. As the figure shows, one bigger element is selected from every two elements to form the max-pooling layer. Therefore, the size of max-pooling layer will be half of the convolutional layer. Then, we use `DropOut` again. In the end, we flat the tensor and use a `Dense` layer with `softmax` to make the final classification.

2.3.2. GLOVE+RNN

The main structure of RNN model is given in Fig 12. The first step is also doing word embedding. After that, it's the same action as in CNN model, we use a `Dropout` layer to drop some connections between neurons randomly to decrease the risk of over-fitting. Different from CNN model, after Dropout layer we replace convolutional layer with a LSTM layer. The last part is also the same as CNN model, there is another Dropout layer and a final Dense layer to classify the text and output it to one of these classes.

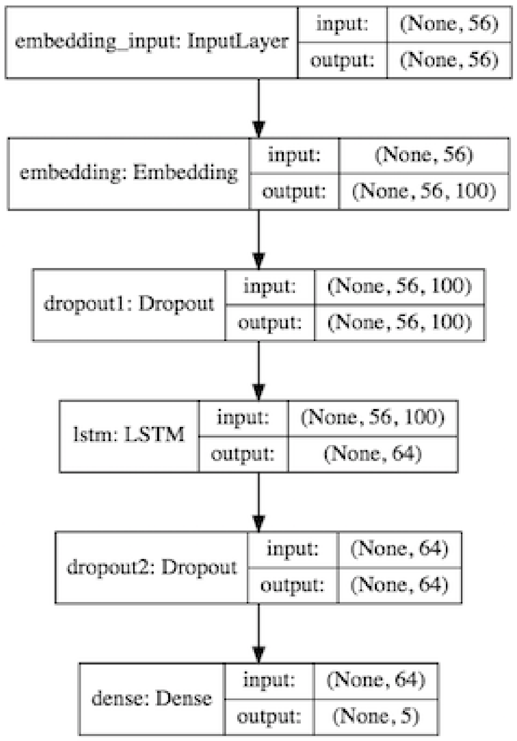


Figure 12. The structure and the number of parameters of RNN model

Here `None` is the number of short texts, `56` is the longest length of all the short texts. To convert words in our dataset into word embeddings, we need to use their one-hot representation as the input of word embedding layer, which requires the same size of the input. In the LSTM layer, every short text is input in 56 * 100 size, but reduced to a 64 * 1 size after the LSTM layer.

Finally, we use `DropOut` and `Dense` again to do the classification.

2.3.3. GLOVE+ Combination of CNN and RNN

The main structure of CNN+RNN model is given in Fig 13. After we get the word embeddings from word embedding layer, we implement CNN part at first. The bottom matrix in the figure is the embedding representation of a short text, and after convolution action we get the convolutional layer of this text. Then apply a max-pooling action as the previous CNN model do.

The green part in this structure is the connection layer which connects CNN part and RNN part. It receives the output of max-pooling layer from CNN part, and input them to the RNN input layer as a sequence.

Since recurrent neural network (RNN) can process sequential input and learn the long-term dependencies, we take these features as the input of the recurrent neural network. We apply LSTM and GRU that are mentioned in previous chapter and both get good results. The output of RNN is deemed as the encoding of the whole sentence. The features generated from RNN form the penultimate layer and are passed to a fully connected softmax layer whose output is the probability distribution over all the categories.

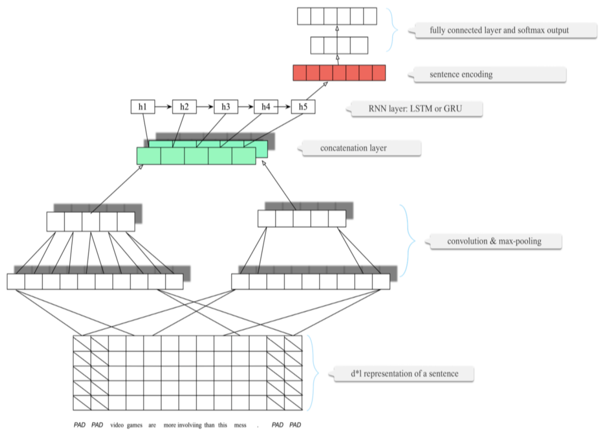

fully connected layer and softmax output
sentence encoding
h1
h2
h3
h4
h5
RNN layer: LSTM or GRU
concatenation layer
convolution & max-pooling
d*l representation of a sentence
PAD PAD video games are more involviing than this mess . PAD PAD

Figure 13. CNN-RNN Model architecture for an example sentence

2.3.4. The parameters of models

The detailed number of parameters are shown in tab.1. The number of Non-trainable parameters is 1,947,900 because we use a pre-trained model in the word embedding layer so we don't need to train any parameters on the word embedding layer. Here, each word is represented by 100 dimensional vectors and there are 19,479 distinct vocabularies in total, so it's $19,479 * 100 = 1,947,900$. For `CNN`, the size of filter of `Conv` layer is $4 * 4$ and there are 200 filters. So the number of parameters of the `Conv` layer should be: 80,200=200[the number of filters]*(1*4[the size of filter]*100[the dimension of each word]+1[bias]), and the number of parameters of the last `Dense` layer is 26005=26*20*5[the number of classes]+5[biases]. So the trainable parameters are $80,200 + 26,005 = 106,205$ in total. For the `RNN` and `CNN-RNN` models, it's the same story for calculating the number of parameters so we will not explain them one by one.

	CNN	RNN	CNN-RNN
Total Params	2,054,105	1,990,465	2,275,345
Trainable Params	106,205	42,565	327,445
Non-trainable Params	1,947,900	1,947,900	1,947,900

Table 1. The number of parameters

The non-trainable parameters are loaded from pre-trained `Glove` model, which provided us the detailed information of word embedding matrix. According to word embedding, every word in a movie review is represented as a fix sized vector. Here, in our glove model, the vector length of a word is 100. The longest length of movie review in this dataset is 56, which makes all reviews represented as a 100*56 word embedding matrix.

2.4. Dataset

There are many qualified public benchmarks for sentiment analysis. Also, researchers use universal dataset to test performances of their models in order to see if their models satisfy these benchmarks or not. One of the most famous datasets is from Stanford NLP team, which is called Stanford Sentiment Treebank 1(SST1(6))[2]. It is essentially an extension of MR(1) dataset. Technically, it's a dataset of movie reviews but provided with five kinds of labels, very negative, negative, neutral, positive and very positive. This dataset includes 153,725 reviews and the average sentence length of SSST1 is 18.

Semantic word spaces have been very useful but cannot express the meaning of longer phrases in a principled way. Further progress towards understanding compositionality in tasks such as sentiment detection requires richer supervised training and evaluation resources and more powerful models of composition. To remedy this, Standford NLP introduce this Sentiment Treebank.

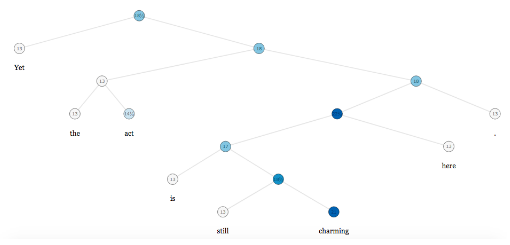


Figure 14. What the dataset looks like

[2]The dataset can be found on `https://nlp.stanford.edu/sentiment/`

2.5. Experimental

2.5.1. Setup

We use GPU on the **duranium** server, which is **GeForce GTX 980**, to run the code. It actually speeds up a lot compare to using CPU. The dataset is split into 2 parts, 80% of it is used to train the model and the rest of the dataset is used to test the model. Furthermore, we run each model 100 epochs and set batch size to 5,000.

2.5.2. Results

After 100 epochs, the details of final results are shown in the table 2.

	CNN	RNN	CNN-RNN
acc on training dataset	0.6383	0.6173	0.6610
acc on test dataset	0.5871	0.6036	0.5919
loss on training dataset	0.8954	0.9314	0.8249
loss on test dataset	0.9897	0.9515	0.9807
time	202s	401s	703s

Table 2. Results of different networks

As can be seen from the table, we use five evaluations to measure the performances of three models. First two performances are accuracy on training dataset and test dataset. As for the accuracy on training dataset, the combination of CNN-RNN model performs better than CNN and RNN model; however, RNN model is the best one in accuracy on test dataset. Therefore, more complex the model is, the effect of the model may not be better.

Corresponding to accuracy, loss function value is also an important indicator in both training dataset and test dataset. As for both training dataset and test dataset, the rank of three models' performances is the same as accuracy on training dataset.

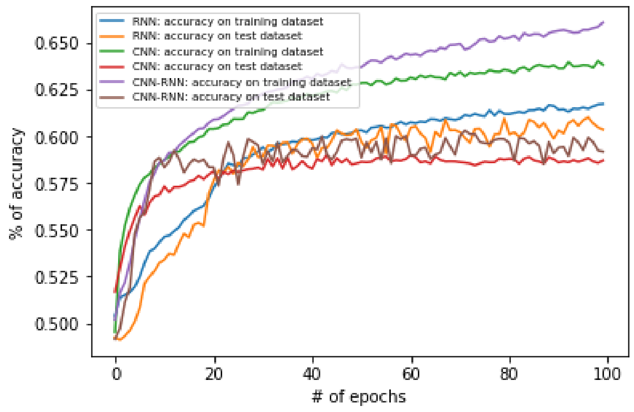


Figure 15. accuracy

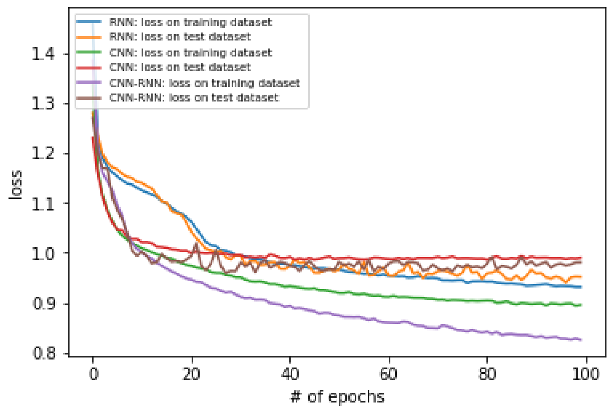


Figure 16. loss

3. Conclusion

References

[1] Lillian Lee Bo Pang. http://www.cs.cornell.edu/people/pabo/movie-review-data/, 2005.

[2] Denny Britz. Understanding Convolutional Neural Networks for NLP. http://www.wildml.com/2015/11/understanding-convolutional-neural-networks-for-nlp/.

[3] Michał Nowicki and Jan Wietrzykowski. Low-effort place recognition with wifi fingerprints using deep learning. In *International Conference Automation*, pages 575–584. Springer, 2017.

[4] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014.

[5] Vyom Sharma. Sentiment Analysis with an RNN. https://github.com/vyomshm/Sentiment-RNN/blob/master/Sentiment%20RNN%20Solution.ipynb.

[6] Socher. http://nlp.stanford.edu/sentiment/, 2013.

[7] Furkan Tektas. Place recognition with WiFi fingerprints using Autoencoders and Neural Networks. https://github.com/aqibsaeed/Place-Recognition-using-Autoencoders-and-NN.

[8] Xingyou Wang, Weijie Jiang, and Zhiyong Luo. Combination of convolutional and recurrent neural network for sentiment analysis of short texts. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*, pages 2428–2437, 2016.