RNN, LSTM and GRU

Final Project for Introduction to Digital Speech Processing

Yu-Siang Wang (Principles), Yi-Ting Lee (Implementation)
ID: B03202047, B03202017
National Taiwan University

January 17, 2017

Abstract

In machine learning, RNN can deal with sequential information by using the memory cells. However, RNN has gradient vanishing problem and the widely used solution is to use LSTM or GRU unit to replace the RNN unit. In the study, we will introduce the principles and the properties of these three models, and then implement the models to compare their performances.

1 Introduction

Traditional Deep Neural Network assumes that all the inputs in the sequence are independent with each other, which means that the weights are not related with others. However, inputs often have strong relation with others in sequence-related task especially in NLP task. Take POS tagging for example, a word's POS tag is strongly related to its words which came before it, but traditional NN can't utilize this feature.

For RNN, it can use information of previous words so that we can have huge improvement on sequence-related task compared with traditional NN.

2 Principles in Neural Networks

In this section, we'll introduce three different types of recurrent neural networks: Simple Recurrent Neural Network, Long Short-Term Memory and Gated Recurrent Unit.

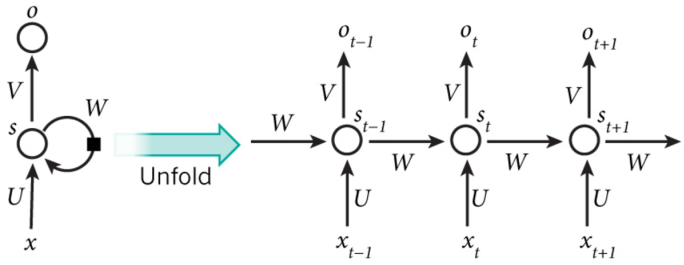

Figure 1: Recurrent Neural Network

2.1 Simple Recurrent Neural Network (RNN)

Simple Recurrent Neural Network is the simplest RNN architecture, and it is also known as recurrent neural network. RNN is often used to tackle sequence-related task.

The simplest RNN is look like Figure 1.

During going through this sequence, the weights will be "reused" in RNN, and the hidden states will be delivered to the next layers as features. The computation in RNN can be shown as follows:

Step 1: Features x_t, x_t can be word embedding.

Step 2: Hidden state h_t, h_t can be calculated by the following equations:

$$h_t = \sigma(Uh_{t-1} + Wx_t) \tag{1}$$

where σ usually is tanh or Relu activation function

Step 3: Output y_t. In our case, activation function in output layer is softmax. Therefore,y_t can be calculated as following equation

$$y_t = softmax(V_t h_t) \tag{2}$$

Step 4: Define loss function E. If we are tackling the classification problem, E is cross entropy and can be computed by

$$E_t(y_t, Y) = -y_t logY \tag{3}$$

Here, y_t is the correct answer and Y is our prediction

Step 5: Update the weights and bias. Repeat Step 1 to Step 4 until reaching the end of the sequence.

The above steps are the most common used computation when we implement RNN. In the next part, we are going to introduce a common problem we'll face when using RNN – Gradient Vanishing.

2.2 Gradient Vanishing Problem

In reality, RNN have difficulties learning long sequences or long range dependencies. Take a t words sentence for example. We call the loss of the last word E_t, then its gradient equals to

$$\frac{\partial E_t}{\partial W} = \sum_{k=0}^{t} \frac{\partial E_t}{\partial Y} \frac{\partial Y}{\partial s_t} \frac{\partial s_s}{\partial s_k} \frac{\partial s_k}{\partial W} \tag{4}$$

which also equals to

$$\frac{\partial E_t}{\partial W} = \sum_{k=0}^{t} \frac{\partial E_t}{\partial Y} \frac{\partial Y}{\partial s_t} \prod_{j=k+1}^{t} \frac{\partial s_j}{\partial s_{j-1}} \frac{\partial s_k}{\partial W} \tag{5}$$

R. Pascanu [2] proved that the upper bound of 2-norm Jacobian matrix of $\frac{\partial s_j}{\partial s_{j-1}}$ is 1 if the activation function is tanh. Therefore, we can have following equation

$$||\frac{\partial s_j}{\partial s_{j-1}}|| \leq \eta < 1 \tag{6}$$

Then we have

$$||\frac{\partial E_t}{\partial s_t} \prod_{j=k+1}^{t} \frac{\partial s_j}{\partial s_{j-1}}|| \leq \eta^{t-k} ||\frac{\partial E_t}{\partial s_t}|| \tag{7}$$

We can see that if t is large, then we can hardly learn from the "far away" steps since $\eta^{t-k} \ll 1$. If we are tackling NLP task, t is the number of words in one sentence, so t can be 10 or 20 or even longer, which makes the gradient vanishing problem more severe. Not only RNN will face gradient vanishing problem, but also DNN with very deep layers will also face this difficulty.

Nowadays, we have some new models which can combat this problem such as LSTM(1997) and GRU(2014), which we will introduce in the following sections.

2.3 Long Short-Term Memory (LSTM)

The Architecture of LSTM [3] is shown in Figure 2.

In one LSTM unit, there are three gates and one memory cell in it. In addition, the output is related to 4 inputs, which means much more parameters than parameters in one RNN unit. The hidden state h_t of LSTM can be computed by the following steps:

Step 1: Calculate the value controlled by input gate. This value will determine whether the previous hidden state can pass through into LSTM or not.

$$i = \sigma(x_t U^i + h_{t-1} W^i) \tag{8}$$

Step 2: Calculate the value controlled by forget gate. This value will determine whether using the value stored in the memory cell or not.

$$f = \sigma(x_t U^f + h_{t-1} W^f) \tag{9}$$

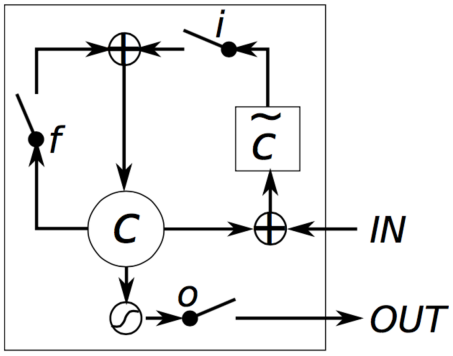

Figure 2: Long Short-Term Memory

Step 3: Calculate "candidate" hidden state. This value looks the same as the hidden state which we computed in RNN. However, we won't directly output this hidden state, so we name this as "candidate" hidden state.

$$g = tanh(x_t U^g + h_{t-1} W^g) \tag{10}$$

Step 4: Calculate new value of the memory cell. In LSTM, there is a memory cell storing the value c_t, and it can be computed by

$$c_t = c_{t-1} \circ f + g \circ i \tag{11}$$

Step 5: Calculate the value controlled by output gate. This value will determine whether the hidden state of this LSTM unit is zero or not.

$$o = \sigma(x_t U^o + h_{t-1} W^o) \tag{12}$$

Step 6: Now we can calculate the hidden state of the LSTM unit.

$$h_t = tanh(c_t) \circ o \tag{13}$$

Notice that $\circ$ means element wise multiplication.

There are other variations based on basic LSTM, such as peepholes version LSTM. The gates of this model not only depend on h_{t-1} but also c_{t-1}.

2.4 Gated Recurrent Unit (GRU)

The architecture of GRU is shown in Figure 3.

In GRU [1], there are two gates while LSTM have three.

Step 1: Calculate the value controlled by update gate z. This gate is a combination of input gate and forget gate in LSTM. It decides how much of the previous hidden state h_{t-1} to use.

$$z = \sigma(x_t U^z + s_{t-1} W^z) \tag{14}$$

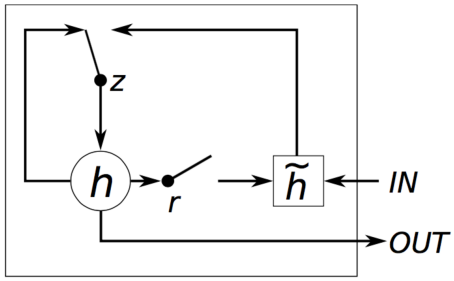

Figure 3: Gated Recurrent Unit

Step 2: Calculate the value controlled by reset gate r. The reset gate determines how to combine the previous hidden state and the new input.

$$r = \sigma(x_t U^r + s_{t-1} W^r) \tag{15}$$

Step 3: Calculate the "candidate" hidden state h.

$$h = tanh(x_t U^h + (h_{t-1} \circ r) W_h) \tag{16}$$

Step 4: Calculate the hidden state of the GRU.

$$h_t = (1 - z) \circ h + z \circ h_{t-1} \tag{17}$$

2.5 Solution to Gradient Vanishing

The reasons for both LSTM or GRU can solve gradient vanishing are very similar, and We will briefly explain why LSTM can solve gradient vanishing.

In RNN, this term $\prod_{j=k+1}^{t} \frac{\partial s_j}{\partial s_{j-1}}$ cause the gradient vanishing. However, the hidden state does not relate to the previous hidden state when the forget gate forget" (which means f=0). Therefore, our new term will be much larger than $||\eta||^{t-k}$ since the value of the power will be much smaller than $t - k$. Due to this reason, our gradient vanishing problem can be solved by LSTM or GRU.

3 Implementation

3.1 Task

Problem Definition

Use each type of recurrent neural networks model
to classify 17500 sentences into 7 subjects.

There are 7 subjects in the data set, which are "biology", "cooking", "crypto", "diy", "physics", "robotics", and "travel". Each of them contains 2500 sentences, so there are 17500 sentences totally. The length of longest sentence is 15 words. In addition, 10000 sentences among them are divided as training data, and 7500 sentences are divided as testing data to evaluate the performance.

Since all of the data have been labeled as the corresponding subject, it is a supervised learning problem.

The data set is downloaded from Kaggle.
(https://www.kaggle.com/c/transfer-learning-on-stack-exchange-tags/data)

3.2 Method

We use Python, Keras (a deep learning library) [4], and Gensim (a NLP library) [5] to implement these three models and compare their performances and convergence speed.
Step 1: Use all the sentences to train the word-to-vector model by Gensim.
Step 2: Embedding all the sentences into 300-dimensional vectors.
Step 3: Construct the training data. Pad zero in the end of the sentences if the length of the sentences < 15. Thus, 1 data will be presented as a 15×300 matrix.
Step 4: Train the model with regularization and dropout.
Step 5: Evaluating the performance on the testing data.

The code and data set can be viewed at `https://github.com/tiffany70072/RNN-LSTM-and-GRU`.

3.3 Model Structure

The structure of each model is shown in Table 1. (Take 2 hidden layers for example).

Since the number of hidden layers, number of neurons of each hidden layer and some other constants may cause the performance be different, We will discuss the result of each case in the next part. In addition, all of the models run for 100 epochs with batch size = 500.

	RNN	LSTM	GRU
Hidden Layer 1	SimpleRNN	LSTM	GRU
Regularization Constant	w, b, u = 0.01	w, b, u = 0.01	w, b, u = 0.01
Activation Function	tanh	tanh	tanh
Dropout	0.25	0.25	0.25
Hidden Layer 2	SimpleRNN	LSTM	GRU
Regularization Constant	w, b, u = 0.01	w, b, u = 0.01	w, b, u = 0.01
Activation Function	tanh	tanh	tanh
Dropout	0.25	0.25	0.25
Output Layer	Dense	Dense	Dense
Number of Neurons in the Output Layer	7	7	7
Activation Function	softmax	softmax	softmax

Table 1: Model Structure

4 Results and Discussion

4.1 Performance

The results are shown in Table 2 to 4.
The performance is defined as:

$$\frac{Number\ of\ sentences\ classified\ to\ the\ correct\ subjects}{Number\ of\ total\ sentences\ (=17500)}$$

In case 1, the performance of LSTM is the highest one, and GRU is the second one. This is because that LSTM model is more complex than GRU model. However, the performance of each model is not much different actually, since our problems are not very complicated. Thus, all of the model can do well on this problems.

In case 2, only the performance of RNN model is higher than the one in case 1, while the performances of LSTM and GRU model in case 2 are lower than the ones in case 1. Perhaps it is because that RNN model is more suitable for 1 layer.

In case 3, with a smaller regularization constant, all of the performances of training data get higher, while the performances of testing data get lower. This is because of the overfitting problem.

	Model	RNN	LSTM	GRU
Number of hidden layers = 2 Neurons of each layers = 100 Regularization constant = 0.01 Dropout = 0.25				
1	Training Data	0.8549	0.8632	0.8552
	Testing Data	0.8240	0.8451	0.8324
2	Training Data	0.8670	0.8667	0.8593
	Testing Data	0.8380	0.8483	0.8377
3	Training Data	0.8639	0.8679	0.8545
	Testing Data	0.8385	0.8455	0.8313
Average	Training Data	0.8619	0.8659	0.8563
	Testing Data	0.8335	0.8463	0.8338

Table 2: Performance for Case 1

Model	RNN	LSTM	GRU
Number of hidden layers = 1 Neurons of each layers = 100 Regularization constant = 0.01 Dropout = 0.25			
Training Data	0.8721	0.8615	0.8577
Testing Data	0.8408	0.8415	0.8333

Table 3: Performance for Case 2

Model	RNN	LSTM	GRU
Number of hidden layers = 2 Neurons of each layers = 100 Regularization constant = 0.003 Dropout = 0.25			
Training Data	0.9552	0.9403	0.9358
Testing Data	0.8261	0.8441	0.8436

Table 4: Performance for Case 3

4.2 Convergence Speed

	RNN	LSTM	GRU
Training time for 1 epoch for case 1 in sec	3-4	14-17	10-12
Convergence Speed, defined as the number of epoch that loss function is lower than 4.0	6	18	10

Table 5: Training Time of Each Model

Convergence speed of each model is shown in Table 5. Since training time depends on the CPU situation and results in inaccuracy, we can only record and compare the approximate order of magnitude.

It is easy to discover that the convergence speed of RNN is much faster than two other models with slightly worse performance in this task. This is because that parameters of LSTM and GRU models are more than simple RNN model, and they also need more training data to train. However, we cannot judge that RNN is a better model than two others, since each model is suitable for different task.

4.3 DNN model for this task

Actually, we have found that any of recurrent neural network model, that is, RNN, LSTM, or GRU, is not the most suitable one for our task this time.

For the simplest method in deep learning, if we want to simulate the influence of five continuous words with each word represented in 300-dim space, then we have to construct 1500 neurons in the input layer. With the recurrent model, we only need a input layer with 300 neurons. Besides, the influence of the former words are slighter than the latter ones. However, that is this property results that the performance of DNN is better than of RNN for our task, since all of the words in the sentences are indeed equally important for classification problems.

Thus, we have also tried DNN model this time. To avoid too much neurons in input layer, the training data is defined as a 300-dim vector which is the average vector of all of the words in one sentence, and the model structure and the performance are shown in Table 6.

The performance of testing data, 0.8777, is higher than other models mentioned above. Since the 1 layer DNN model is simple, it barely has overfitting problem so that we don't need to add dropout layers. In addition, the convergence speed is much faster than any recurrent models.

DNN	
Number of hidden layers	1
Neurons of each layers	100
Regularization constant	0.01
Dropout	0.00
Training Data	0.8813
Testing Data	0.8777
Training Time for 1 epoch	< 1 sec
Convergence Speed, defined as the number of epoch that loss function is lower than 4.0	1

Table 6: DNN Model

5 Conclusion

We learned that the RNN can do better in the sequence-related task since DNN will not use the information from the previous unit.However, when we proved that not only RNN but also DNN will face gradient vanishing problem, which will become more severe when the sequence is very long.

In DNN, we tackle the problem by using Relu as activation function. In RNN, we have Long short-term memory(LSTM) and Gate recurrent unit (GRU) to tackle this problem. Next, we want to compare these three various types RNN on subject classification task.

From the result, we can see that this task is more suitable for DNN model. However, we can still use it to compare the performance of each kind of recurrent neuron networks model.

Indeed, in the field of machine learning, each model has its pros and cons for different data set. It never exist a model which is the absolutely best one.

References

[1] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, Yoshua Bengio: Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling. NIPS Deep Learning and Representation Learning Workshop, (2014)

[2] Razvan Pascanu , Tomas Mikolov and Yoshua Bengio: On the difficulty of training recurrent neural networks. Arxiv (2012)

[3] S Hochreiter, J Schmidhuber: Long short-term memory. Neural computation, (1997)

[4] Francois Chollet: Keras Documentation , keras.io (2016)

[5] Rehurek Radim , Scalability of Semantic Analysis in Natural Language Processing, (2011)

[6] Hung-Yi Lee, Machine Learing Course, National Taiwan University (2016)

[7] Denny Britz, Recurrent Neural Networks Tutorial, WILDML blog (2015)