Deep Learning

Sentence-level Sentiment Classification with RNN

Author: Yantian Luo

Institute: Electronic Engineering

ID: 2018310742

Update: May 12, 2019

Contents

Chapter 1 Introduction

In this homework, we focus on fine-grained sentence-level sentiment classification problem.

We use Stanford Sentiment Treebank (SST) dataset for experiments. This dataset contains 11,855 sentences, and has been splitted into the training / validation / test parts, containing 8,544 / 1,101 / 2,210 sentences respectively. Each sentence is categorized into one type of 5 sentiments. To preprocess the dataset, we use torchtext package. Before training the model, we need to tokenize words, building vocabulary, construct word embedding and intialize data iterator. torchtext has good wrappers for these operations and we can directly use dataloader in a similar way as before.

Chapter 2 Algorithm Design

In this homework, we need to complete the forward process of RNNCell, GRUCell and LSTMCell using PyTorch. Considering these three cells have a common forward API: def forward(self, input, state). We can design the following algorithms.

2.1 RNNCell

We can use Algorithm 1 to complete the forward process of RNNCell.

Algorithm 1 the forward algorithm of RNNCell

```
Input:  input: $x$, hidden state: $s = [h, h]$(where $h$ is hidden state vector)
Output:  new state: $s'$
1: $h' = \tanh(W_{ih}x + b_{ih} + W_{hh}h + b_{hh})$
2: return $s' = [h', h']$
```

2.2 GRUCell

We can use Algorithm 2 to complete the forward process of GRUCell.

Algorithm 2 the forward algorithm of GRUCell

```
Input:  input: $x$, hidden state: $s = [h, h]$(where $h$ is hidden state vector)
Output:  new state: $s'$
1: $r = \sigma(W_{ir}x + b_{ir} + W_{hr}h + b_{hr})$
2: $z = \sigma(W_{iz}x + b_{iz} + W_{hz}h + b_{hz})$
3: $n = \tanh(W_{in}x + b_{in} + r * (W_{hn}h + b_{hn}))$
4: $h' = (1 - z) * n + z * h$
5: return $s' = [h', h']$
```

2.3 LSTMCell

We can use Algorithm 3 to complete the forward process of LSTMCell.

Algorithm 3 the forward algorithm of LSTMCell

```
Input:   input: $x$, hidden state: $s = [h, h]$(where $h$ is hidden state vector)
Output:  new state: $s'$
1: $r = \sigma(W_{ir}x + b_{ir} + W_{hr}h + b_{hr})$
2: $z = \sigma(W_{iz}x + b_{iz} + W_{hz}h + b_{hz})$
3: $n = \tanh(W_{in}x + b_{in} + r * (W_{hn}h + b_{hn}))$
4: $h' = (1 - z) * n + z * h$
5: return $s' = [h', h']$
```

Chapter 3 Results

3.1 Plot the loss value and accuracy value of one-layer RNN with 3 rnncells against to every epoch during training

In this section, we use $lr = 0.01, epochs = 5$ with 3 rnncells to complete the experiments, and the loss value and accuracy value curves are as follow.

3.1.1 RNNCell

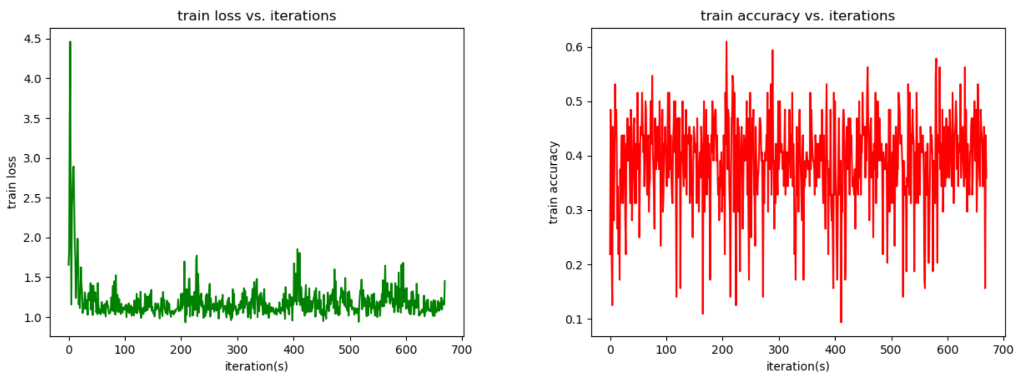

Figure 3.1: train loss curve and train accuracy curve using RNNCell against to every iteration

3.1.2 GRUCell

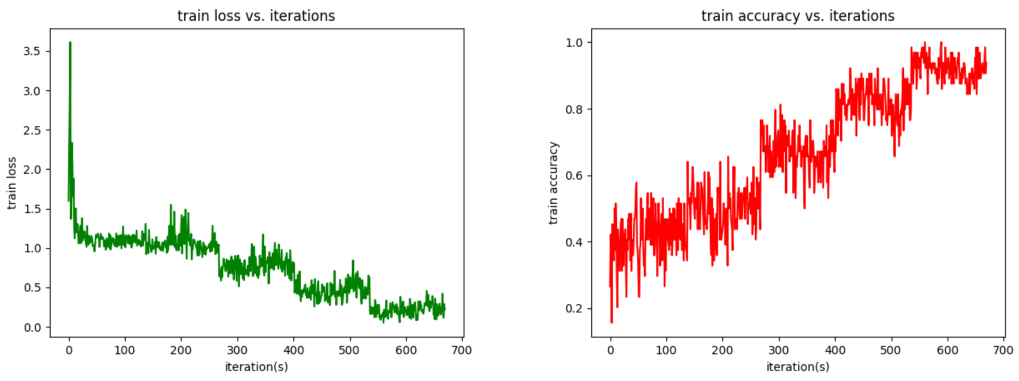

Figure 3.2: train loss curve and train accuracy curve using GRUCell against to every iteration

3.1.3 LSTMCell

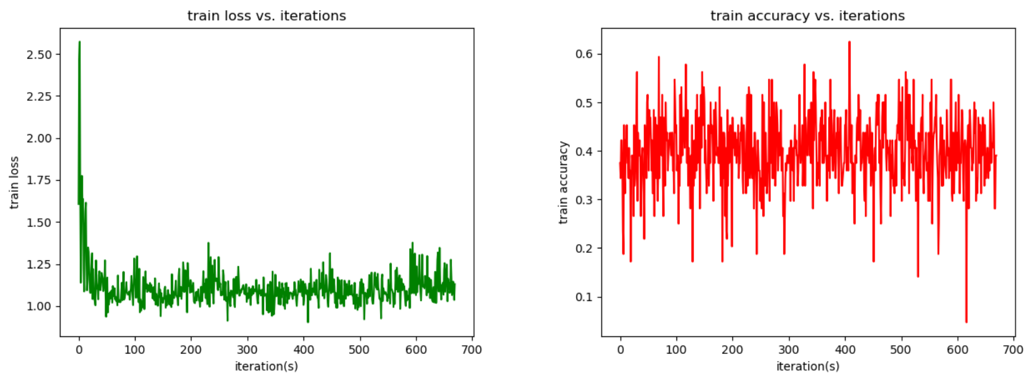

Figure 3.3: train loss curve and train accuracy curve using LSTMCell against to every iteration

3.2 Compare and analyze the performance of the 3 rnncells

We sum up the test results of 3 rnncells and list the results in Table 3.1. From the table and the Figure in last section, we can find that:

- Using RNNCell, the training speed is very fast but the test accuracy is not well;
- Using GRUCell, the training speed is also fast, and the test accuracy is better than RNNCell;
- Using LSTMCell, the training speed is very slow, and the test accuracy and test loss is better than RNNCell.

Therefore, we can find in this work, GRUCell is better than the other two rnncells.

Table 3.1: The comparison results of 3 rnncells

Cell Type	Test Accuracy	Test Loss
RNNCell	0.5350	1.0595
GRUCell	0.5825	1.2509
LSTMCell	0.5765	1.0741

3.3 Using 100 dimensional embedding vector

In this section, we use 100 dimensional embedding vector and GRUCell to do the experiment. We also plot the train loss curve and train accuracy curve against to every iteration in Figure 3.4. The best test accuracy is 0.5945 and the best test loss is 1.2753. We can find that the test accuracy is a little better than before and the train speed is a litter fast because the dimension is a smaller than before.

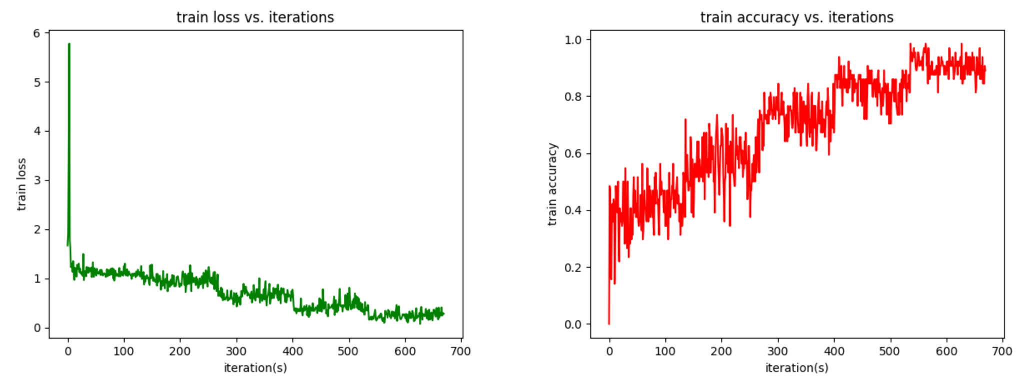

Figure 3.4: train loss curve and train accuracy curve using 100 dimensional embedding vector against to every iteration

3.4 Initialize embedding vectors without normal distribution

In this section, we try to set $unk\_init = None$ to do the experiment that initialize embedding vectors without normal distribution. We also plot the train loss curve and train accuracy curve against to every iteration in Figure 3.5. The best test accuracy is 0.6466 and the best test loss is 0.9108. We can find that the test accuracy and the test loss is much better than before.

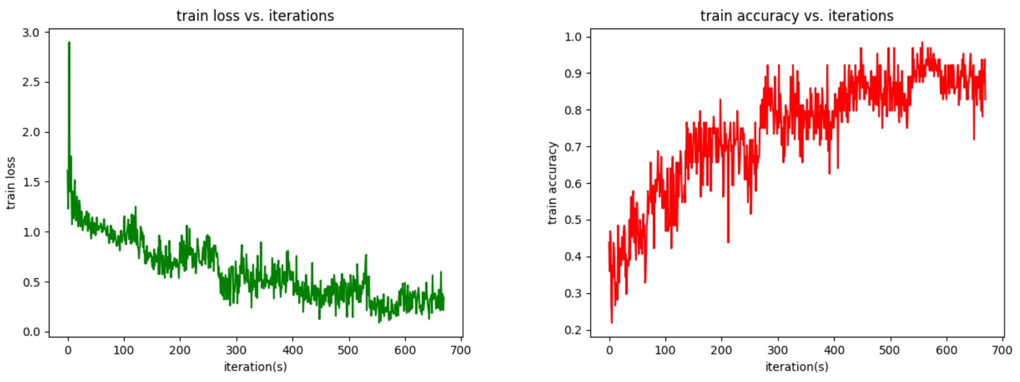

Figure 3.5: train loss curve and train accuracy curve using 300 dimensional embedding vector and GRUCell without normal distribution against to every iteration