Assignment3 Report: Recurrent Neural Networks

CS5242 Neural Network and Deep Learning

Meng-Jiun Chiou
mengjiun.chiou@u.nus.edu

November 21, 2017

1 Introduction

Recurrent neural networks have been a popular approach to the natural language processing (NLP) problems, e.g. Language Modeling, Machine Translation, Speech Recognition and Question Answering. To realize how recurrent neural network became successful, we have to fully understand the architecture of it. The aim of this assignment is to let students implement the modules of a vanilla recurrent neural network. In addition, this assignment introduces and implements the bidirectional recurrent neural network [1].

2 Modules

This reports introduces the basics to implement essential in (bi-directional) recurrent neural networks.

2.1 RNN Step Forward

The basic module of forward process is the function **rnn step forward**. Given previous hidden state h_{t-1}, input of this timestep x_t, respective weights W_h and W_x and bias b, the output hidden state h_t is:

$$a_t = W_h h_{t-1} + W_x x_t + b \tag{1}$$

$$h_t = tanh(a_t) \tag{2}$$

2.2 RNN Step Backward

The basic module of forward process is the function **rnn step backward**. The backpropagation of recurrent neural network backprop the gradient with respect to loss to the parameters of current time step. Let the gradient of the hidden state in the next time step $\frac{\partial L}{\partial h_t}$ to be d_{h_t}, then the gradient of a_t of current time step is:

$$d_{a_t} = d_{h_t}(1 - h_t^2) \tag{3}$$

The gradient of hiddent state h_{t-1} of previous time step is:

$$d_{h_{t-1}} = d_{a_t} W_h^T \tag{4}$$

The gradient of input x of current time step is:

$$d_{x_t} = d_{a_t} W_x^T \tag{5}$$

The gradient of weight corresponding to hidden state W_h is:

$$d_{W_h} = W_h^T d_{a_t} \tag{6}$$

The gradient of weight corresponding to current input W_x is:

$$d_{W_x} = W_x^T d_{a_t} \tag{7}$$

The gradient of bias of current time step is:

$$d_b = d_{a_t} \tag{8}$$

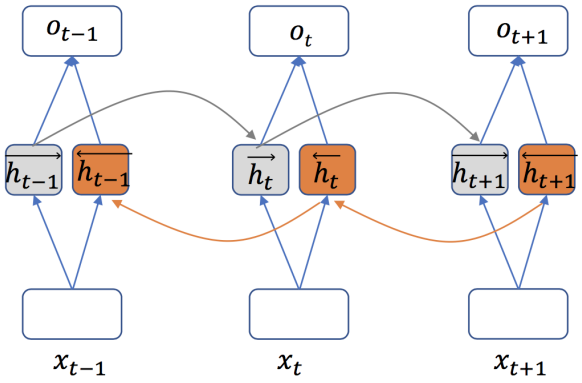

Figure 1: Bi-directional RNN

2.3 RNN Forward

The **rnn forward** function runs a vanilla RNN forward on an entire sequence of data and output the hidden states for the entire time series.

For each time step, we run a **rnn step forward** function to calculate hidden states. For the first iteration, we use a all zero vector as our initial hidden state h_0.

2.4 RNN Backward

The **rnn backward** function runs the backward pass for a vanilla RNN over an entire sequence of data. This function output the gradients of inputs for each time steps and also gradients of weights W_x, W_h and bias b.

Note that the gradients of weights W_x, W_h and bias b is the sum over the entire sequence of data (all time steps), but not independent gradients for each time step.

2.5 Bi-directional RNN Concatenation: Forward and Backward

The **bi-directional rnn concatenation forward** function runs the forward pass for concatenating hidden vectors obtained from a RNN trained on normal sentences and a RNN trained on reversed sentences at each time step. See figure 1 for reference.

Given the current hidden states $\overrightarrow{h_t}$ and $\overleftarrow{h_t}$, the two hidden states are simply masked out (set values to zero for which the positions exceed the length of sentence) and concatenated together.

$$o_t = V[\overrightarrow{h_t}, \overleftarrow{h_t}] + c \tag{9}$$

The backward process is simply backprop the respective part of concatenated gradient d_{ho} to $\overrightarrow{h_t}$ and $\overleftarrow{h_t}$.

3 RNN for Sentiment Analysis

With the modules we have implemented so far, we can put them together to form a recurrent neural networks for sentiment analysis task.

3.1 Architecture

The network consists of multiple layers:

Input - RNN Layer - Temporal Affine Layer - Average Layer - Affine Layer - Output (Softmax)

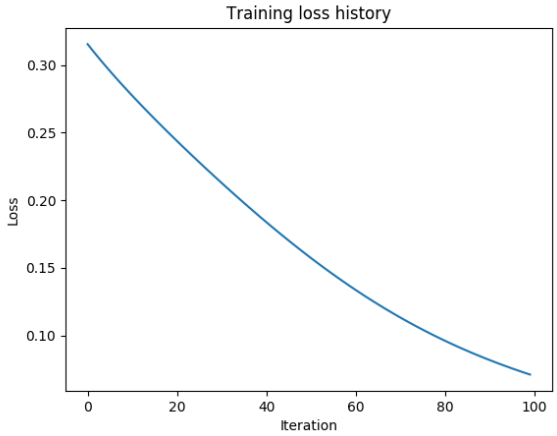

Figure 2: Training loss history

What we need to do here is to stack all the layers needed together, feeding previous output into the next layer. The loss we get from the last layer (softmax) is used to calculating the gradients to do backpropagation.

3.2 Training and Inference

After training, we have got the parameters needed. We then simply use forward pass only to generate the probability distribution (scores) over all sentiment classes, which have only two classes in this example.

Figure 2 shows the result of training on the small data (100 samples) of 100 iterations. The loss decreases from 0.32 to 0.07 quickly, showing this is the correct training procedure.

4 Discuss

This assignment aims at letting students understand the basics of recurrent neural network by implementing vanilla recurrent neural network and bi-directional layer. However, there are still many tricks can be used to train a better network. For example, the initial state h_0 is set to be a all-zero vector, while in fact h_0 can be learned like a parameter through gradient descent update. Other examples include adaptive learning rate (Adam, RMSProp), adding gated units, layers normalization (L2 normalization) and truncated Backpropagation Through Time (BPTT) for large sentences.

References

[1] M. Schuster and K. K. Paliwal, "Bidirectional recurrent neural networks," *IEEE Transactions on Signal Processing*, vol. 45, no. 11, pp. 2673–2681, 1997.