BPTT for RNN and LSTM

Songbin Liu

songbinliu@hotmail.com

March 4th, 2018

Abstract

The training process of RNN/LSTM is not easy to be comprehended. This essay presents a easy way to understand this process. Firstly, it will explain the BPTT process when training RNN, and the gradient vanishing problem of RNN. Secondly, it will explain the BPTT process for LSTM, and how LSTM can handle the gradient vanishing problem better.

1 Introduction

Recurrent Neural Networks (RNN), especially its variants, such as Long Short Term Memory networks (LSTM) and GRU, are powerful neural network models. With today's more advanced computation resources, larger volumes of training data, and better optimization techniques, these models are proved very effective to solve application problems. There are many platforms/tools have implemented them, and make them handy for use.

However, it is still useful to understand these models' internal math dynamic process, and how they can be implemented, so that we can take better use of these models: choose the right model for the application problem, prepare the training data well, and know how to tune the hyper-parameters.

In this documentation, Section-2 describes the training process of vanilla RNN with some math explanation; in addition, the gradient vanishing and exploding problem of vanilla RNN is also explained. Section-2 describes how to train the LSTM with math explanation too, and points out how LSTM overcomes the drawback of vanilla RNN.

Note: Because of my humble mathematical background, I will try to explain RNN/LSTM in an intuitive fashion, emphasizing concepts rather than mathematical details. In fact, after labeling the layers the unfolded RNN/LSTM properly, the training process is almost the same as training an ordinary multi-layer feed-forward neural networks.

2 training RNN with BPTT

In this section, the structure of the vanilla RNN is recapped first; second, the process of BPTT to train this RNN is demonstrated via a simplified example; third, the difficulty of training this vanilla RNN is explained through an example.

2.1 Vanilla RNN

Figure-1 shows the structure of a vanilla RNN.

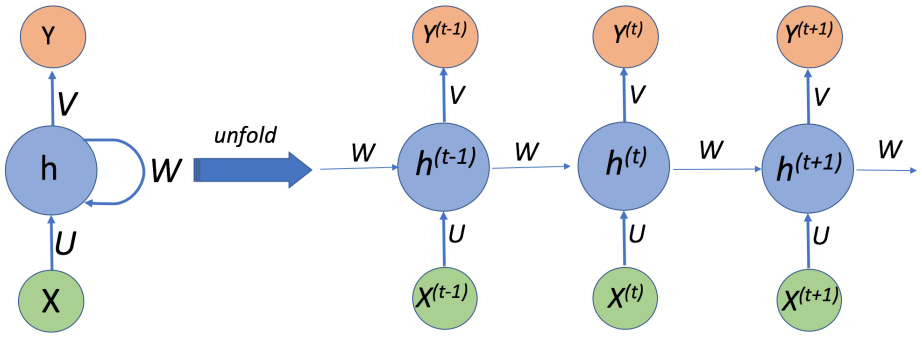


Figure 1: Vanilla RNN

$$h^{(t)} = \tanh(Ux^{(t)} + Wh^{(t-1)}) \tag{1}$$

$$y^{(t)} = softmax(Vh^{(t)}) \tag{2}$$

In this simple RNN, there is an input $x^{(t)}$ and an output $y^{(t)}$ at each time step t. $h^{(t)}$ is the hidden state of the RNN at (the end of) time step t. The output $y^{(t)}$ depends on current hidden state. While the current hidden state $h^{(t)}$ depends not only on current input $x^{(t)}$, but also depends on previous hidden state $h^{(t-1)}$. The training process will try to learn the parameters: U, W, and V (and also bias b, which is not shown).

2.2 Training RNN with BPTT

RNN is trained with the *Backpropagation Through Time (BPTT)* method, which is a variant of the normal *Backpropagation* method for feed-forward neural networks. When trained with the BPTT method, for each input sequence, RNN is unfolded into a (deep) feed-forward neural network in time dimension (as shown in 1), the gradients of parameters U, W and V are calculated at each layer. There are two main differences from the normal *Backpropagation*:

First, since the parameters U, W and V are shared across the steps/layers, these parameters are updated by the sum of the gradients in each step, like the how the shared parameters are updated in Convolutional neural network(CNN).

Second, usually there is an output at each timestep, so there is one training example at each time step. The error of each output is backpropageted to previous steps. In this way, a sequence of length T, will correspond to T overlapped feed-forward neural networks, and the depth of these neural networks is one layer more than previous one.

2.2.1 Step 1: Unfold the RNN

For the easy of understanding, the parameters U, W and V are labeled with the time step t. In this way, the unfolded RNN looks like a normal feed-forwad neural network. Figure-2 shows the resulting neural network of an unfolded RNN of 4 timesteps in total. In Figure 2, $W^{(t)}$ means the parameter W at timestep (or layer) t. $E^{(i)}$ is the error of time step i. Note that even though the parameters U, W and V are distinguished by label t, they are treated as one respectively when updating them.

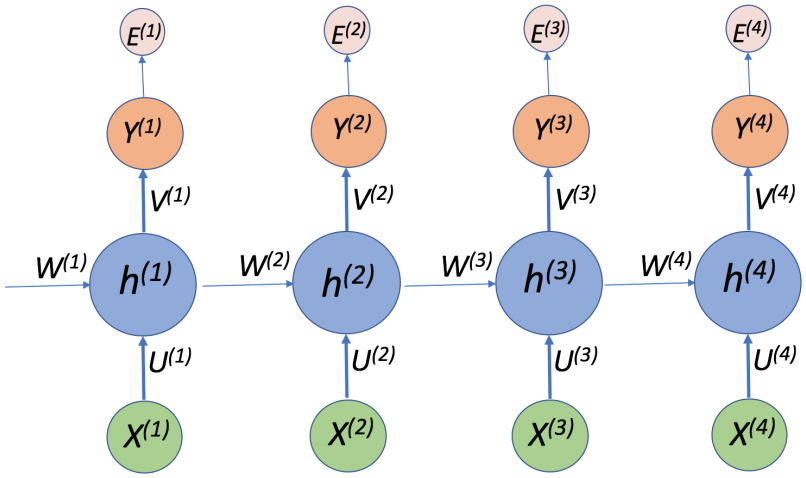


Figure 2: unfolded RNN of 4 time steps

2.2.2 Step 2: Error Backpropagation for each output

Calculate the derivates of the parameters for each output error. The output and input of each timestep is treated as one training example. For this training example, the unfolded RNN is treated as one feed-forward neural network with only one output: the current output, and current output is treated as the last layer of the network. For example, for timestep 3, timestep 4 is ignored, and the previous outputs are also ignored, as is shown in Figure-3. We only have to calculate the derivates with regard to $V^{(3)}$, $W^{(1,2,3)}$, and $U^{(1,2,3)}$ for error $E^{(3)}$. Next I will explain how to calculate these derivates for the example in Figure-3.

Firstly, define $\frac{\partial E^{(3)}}{\partial W^{(j)}}$, $\frac{\partial E^{(3)}}{\partial U^{(j)}}$, and $\frac{\partial E^{(3)}}{\partial V^{(j)}}$ as the derivates with regard to $W^{(j)}$, $U^{(j)}$ and $V^{(j)}$ for error $E^{(3)}$, where $j \in [1, 3]$.

Secondly, define $\Delta W^{(3)}$, $\Delta U^{(3)}$, and $\Delta V^{(3)}$ as the derivates with regard to W, U and V for error $E^{(3)}$. They are the sum of the derivates of each time step, as is shown in Equation-3.

$$\begin{cases} \Delta W^{(3)} &= \frac{\partial E^{(3)}}{\partial W} = \sum_{j=1}^{3} \frac{\partial E^{(3)}}{\partial W^{(j)}} \\ \Delta U^{(3)} &= \frac{\partial E^{(3)}}{\partial U^{(j)}} = \sum_{j=1}^{3} \frac{\partial E^{(3)}}{\partial U^{(j)}} \\ \Delta V^{(3)} &= \frac{\partial E^{(3)}}{\partial V^{(j)}} = \frac{\partial E^{(3)}}{\partial V^{(j)}} \end{cases} \tag{3}$$

To calculate $\Delta W^{(3)}$ and $\Delta U^{(3)}$, we use the similar tricks of *Backpropagation*. Define two auxiliary variables $z^{(t)}$, and $\delta^{(t)}$ as shown in Equation-4.

$$\begin{cases} z^{(t)} &= U^{(t)} x^{(t)} + W^{(t)} h^{(t-1)} \\ \delta^{(t)} &= \frac{\partial E^{(3)}}{\partial z^{(t)}} \end{cases} \tag{4}$$

With equation-4, $\frac{\partial E^{(3)}}{\partial W^{(j)}}$, $\frac{\partial E^{(3)}}{\partial U^{(j)}}$ can be calculated by Equation-5.

$$\begin{cases} \frac{\partial E^{(3)}}{\partial W^{(j)}} &= \frac{\partial E^{(3)}}{\partial z^{(j)}} \frac{\partial z^{(j)}}{\partial W^{(j)}} = \delta^{(j)} h^{(j-1)} \\ \frac{\partial E^{(3)}}{\partial U^{(j)}} &= \frac{\partial E^{(3)}}{\partial z^{(j)}} \frac{\partial z^{(j)}}{\partial U^{(j)}} = \delta^{(j)} x^{(j)} \end{cases} \tag{5}$$

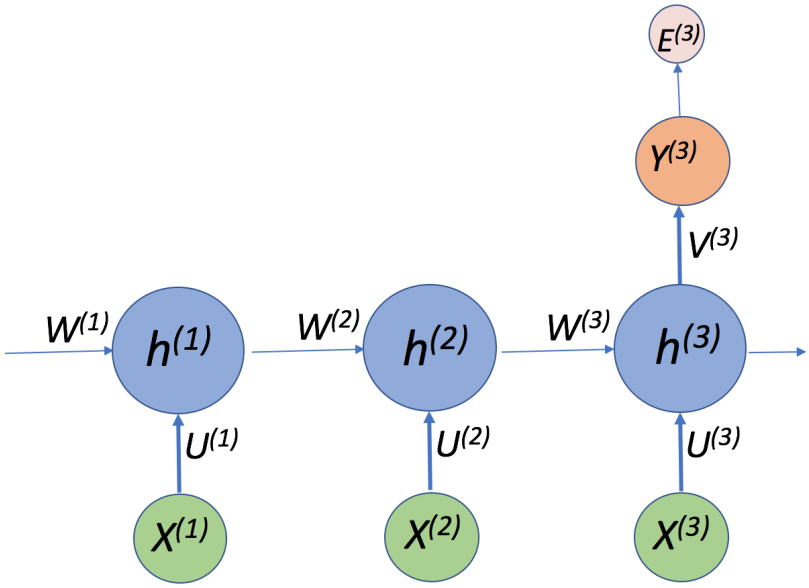


Figure 3: Calculate the derivates for the output error of timestep 3

While $\delta^{(j)}$ can be calculated by the chain rule, as is shown in Equation-6.

$$\delta^{(j)} = \frac{\partial E^{(3)}}{\partial z^{(j+1)}} \frac{\partial z^{(j+1)}}{\partial z^{(j)}} = \delta^{(j+1)} W^{(j+1)} tanh'(z^{(j)}) \tag{6}$$

In summary, we can **use the following backpropagation process to calculate the derivates for error** $E^{(3)}$, which is shown in Equation-7.

$$\begin{cases} \delta^{(j)} & = \delta^{(j+1)} W^{(j+1)} tanh'(z^{(j)}) \\ \Delta W^{(3)} & = \sum_{j=1}^{3} \delta^{(j)} \frac{\partial z^{(j)}}{\partial W^{(j)}} = \sum_{j=1}^{3} \delta^{(j)} h^{(j-1)} \\ \Delta U^{(3)} & = \sum_{j=1}^{3} \delta^{(j)} \frac{\partial z^{(j)}}{\partial W^{(j)}} = \sum_{j=1}^{3} \delta^{(j)} x^{(j)} \end{cases} \tag{7}$$

2.2.3 Step 3: Average up the derivatives for the sequence

As mentioned earlier, each timestep is a training example, so there are T training examples for a training sequence of length $T+1$. These training examples can be treated as a mini-batch. Thus, the derivates of W, U and V for one sequence (or several sequences) are the average of the derivates for each output, as is shown in Equation-8.

$$\begin{cases} \Delta W & = \frac{1}{T} \sum_{i=1}^{T} \Delta W^{(i)} \\ \Delta U & = \frac{1}{T} \sum_{i=1}^{T} \Delta U^{(i)} \\ \Delta V & = \frac{1}{T} \sum_{i=1}^{T} \frac{\partial E^{(i)}}{\partial V^{(j)}} \end{cases} \tag{8}$$

Where, $T+1$ is the length of the input training sequence, $E^{(i)}$ is the error of time step i.

2.2.4 Step 4: update the parameters

After all the derivates of the parameters of current input sequence (or sequence batch) have been calculated, we can update the parameters with Equations-9.

$$\begin{cases} W & = W - \lambda \Delta W \\ U & = U - \lambda \Delta U \\ V & = V - \lambda \Delta V \end{cases} \tag{9}$$

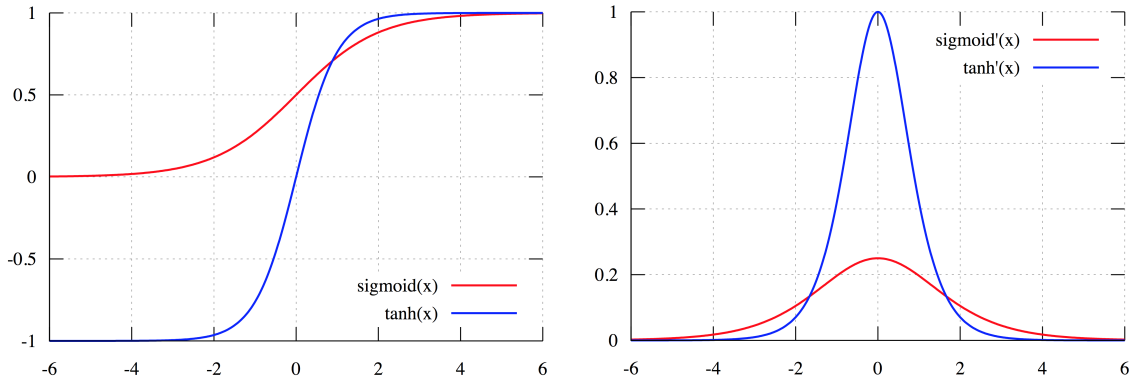
where λ is the learning rate. There are various ways to set a proper learning rate for each weight.

2.3 Gradient vanishing and exploding of RNN

The vanilla RNN model looks very promising for sequential problems, but it did not work well in application. This is because of the gradient vanishing and exploding problem during the training process, it is very difficult to train the RNN model in practice.

$$\delta^{(1)} = \delta^{(4)} W^{(4)} tanh'(z^{(3)}) W^{(3)} tanh'(z^{(2)}) W^{(2)} tanh'(z^{(1)}) \tag{10}$$

How does the gradient vanishing or exploding problem happen? Here is an example. Suppose to backpropagate the output error of timestep 4 to timestep 1. As explained in section-2.2.2, to calculate $\frac{\partial E^{(4)}}{\partial W^{(1)}}$, $\frac{\partial E^{(4)}}{\partial U^{(1)}}$, we need to get $\delta^{(1)}$ according to equations-5. And according to equation-6, $\delta^{(1)}$ can be obtained by equation-10. If the $W^{(j)} tanh'(z^{(j-1)})$ terms are greater than 1, then gradient exploding can happen, and will cause numerical overflow. If **the $W^{(j)} tanh'(z^{(j-1)})$ terms are smaller than 1**, then the gradient tends to be vanishing, which means the error can only be backpropagated to nearby steps (it can only have very short memory of very nearby inputs).



(a) Activation functions

(b) Derivatives of the activation functions

Figure 4: *sigmoid* and *tanh* functions and their derivatives

Usually, the gradient exploding problem can be handled by gradient clipping, and won't be a big problem for training RNN. The gradient vanishing problem is the major obstacle when training RNN. The gradient vanishing can happen very likely. As shown in Figure-4b, the derivatives of the

two commonly used activation functions *sigmoid* and *tanh* are very near to zero. Their derivatives are only significantly larger than zero in a very small interval (for example, $\tanh'(-3) \approx 0.01$, $sigmoid'(-3) \approx 0.047$). Even with very carefully set of the initial weights, the gradient still can get decreased to zero in exponential order with the number of layers increase.

3 LSTM

Long Short Term Memory network (LSTM) is a special kind of RNN, and is capable of learning long-term dependencies. It can handle the gradient vanishing problem much better, and are widely used in practice. There are several variants of LSTM, but in this section, we focus on the simplest one. It has a cell state, and a hidden state, three gates: forget gates, input gates, and output gates. These gates are controlled by current input and the hidden state(no peephole connection from the cell state).

3.1 Structure of LSTM

Define $x^{(t)}$ is the input of timestep t, $C^{(t)}$ is the cell state, $h^{(t)}$ is the hidden state, $y^{(t)}$ is the output. The forward process to update these values is shown in Equations-11–17.

$$\begin{cases} i^{(t)} = \sigma(W_i \cdot [h^{(t-1)}, x^{(t)}]) & (11) \\ A^{(t)} = \tanh(W_a \cdot [h^{(t-1)}, x^{(t)}]) & (12) \\ f^{(t)} = \sigma(W_f \cdot [h^{(t-1)}, x^{(t)}]) & (13) \\ C^{(t)} = f^{(t)} \cdot C^{(t-1)} + i^{(t)} \cdot A^{(t)} & (14) \\ o^{(t)} = \sigma(W_o \cdot [h^{(t-1)}, x^{(t)}]) & (15) \\ h^{(t)} = o^{(t)} \cdot \tanh(C^{(t)}) & (16) \\ y^{(t)} = softmax(V \cdot h^{(t)}) & (17) \end{cases}$$

Firstly,update the cell state(Equation-14) with the input gate(Equation-11), input value(Equation-12), forget gate(Equation-13), and previous cell state;

Secondly, update hidden state(Equation-16) with the new cell state and output gate(Equation-15);

Lastly, calculate the output based on the new hidden state(Equation-17);

The training process will try to learn the parameters W_i, W_a, W_f, W_o, and V (and also the biases). As a comparison, the vanilla RNN only has W_a and V.

3.2 Training LSTM

BPTT is also used to train LSTM neural networks. The process of training LSTM is similar to the process for a vanilla RNN as is shown in Section 2.2

Step 1: Unfold the input training sequence into a feed-forward neural network, as shown in Figure 5;

Step 2: For each output, treat the output and all the steps (or layers) before that output as a feed-forward neural network, and calculate the derivates for the parameters with regard to this output error. Sum up the derivates for each parameter;

Step 3: Average the derivatives for the all the outputs for the input sequence;

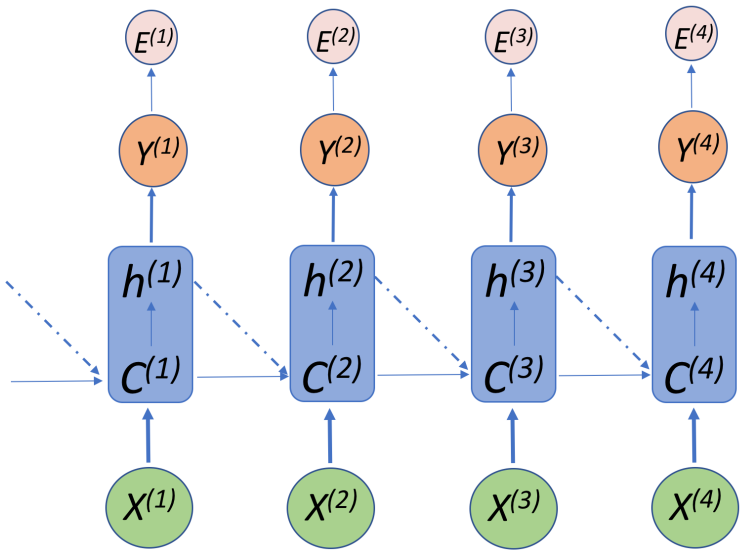


Figure 5: Unfolded LSTM with length of 4

Step 4: Update the parameters with the averaged derivative for each parameter with a proper learning rate.

The only difference from the vanilla RNN is in **Step 2**, which will be discussed in detail in following subsection.

3.2.1 Step 1: Unfold the LSTM

Similar to RNN, the unfolded LSTM looks like a normal feed-forward neural network, and the parameters are also distinguished with timestep t, as shown in Figure 5. However, LSTM has more parameters, and the internal process is much more complicated than that of RNN.

3.2.2 Step 2: Calculate the derivates for one output error

The output and input of each timestep of LSTM is also treated as one training example. For each training example, the unfolded LSTM is treated as one feed-forward neural network with only one output: the current output, and current output is treated as the last layer of the network. For example, when calculate the derivatives for the output error of timestep 3, timestep 4 is ignored, and the previous outputs are also ignored, as is shown in Figure-6. We only have to calculate the derivates with regard to $W_f^{(1,2,3)}$, $W_i^{(1,2,3)}$, $W_a^{(1,2,3)}$, $W_o^{(1,2,3)}$, and $V^{(3)}$ for error $E^{(3)}$. Next will explain how to calculate these derivates with the example in Figure-6.

Firstly, define $\frac{\partial E^{(3)}}{\partial W_f^{(j)}}$, $\frac{\partial E^{(3)}}{\partial W_i^{(j)}}$, $\frac{\partial E^{(3)}}{\partial W_a^{(j)}}$, $\frac{\partial E^{(3)}}{\partial W_o^{(j)}}$, and $\frac{\partial E^{(3)}}{\partial V^{(j)}}$ as the derivates with regard to $W_f^{(1,2,3)}$, $W_i^{(1,2,3)}$, $W_a^{(1,2,3)}$, $W_o^{(1,2,3)}$, and $V^{(3)}$ for error $E^{(3)}$.

Secondly, define $\Delta W_f^{(3)}$, $\Delta W_i^{(3)}$, $\Delta W_a^{(3)}$, $\Delta W_o^{(3)}$, and $\Delta V^{(3)}$ as the derivates with regard to W_f, W_i, W_a, W_o, and V for error $E^{(3)}$. They are the sum of the derivates of each time step, as is shown in Equation-18.

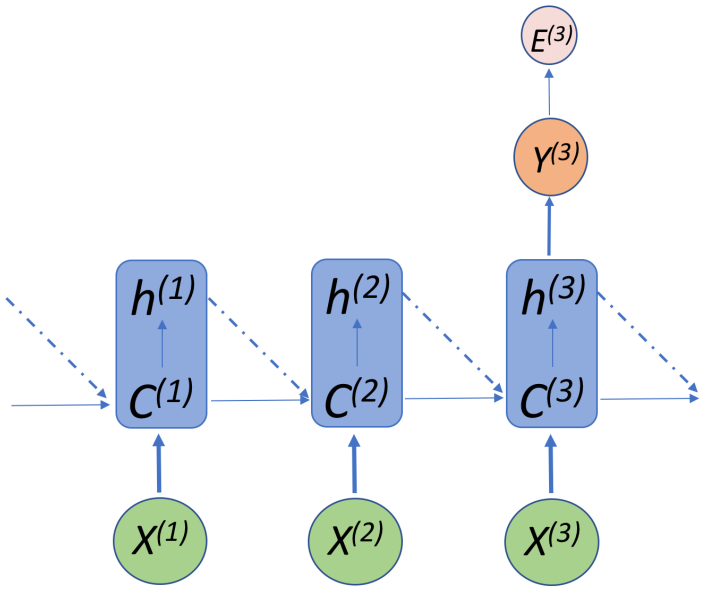


Figure 6: Calculate the derivates for the output error of timestep 3 in LSTM

$$\begin{cases} \Delta W_f^{(3)} &= \frac{\partial E^{(3)}}{\partial W_f} = \sum_{j=1}^{3} \frac{\partial E^{(3)}}{\partial W_f^{(j)}} \\ \Delta W_i^{(3)} &= \frac{\partial E^{(3)}}{\partial W_i} = \sum_{j=1}^{3} \frac{\partial E^{(3)}}{\partial W_i^{(j)}} \\ \Delta W_a^{(3)} &= \frac{\partial E^{(3)}}{\partial W_a} = \sum_{j=1}^{3} \frac{\partial E^{(3)}}{\partial W_a^{(j)}} \\ \Delta W_o^{(3)} &= \frac{\partial E^{(3)}}{\partial W_o} = \sum_{j=1}^{3} \frac{\partial E^{(3)}}{\partial W_o^{(j)}} \\ \Delta V^{(3)} &= \frac{\partial E^{(3)}}{\partial V^{(j)}} = \frac{\partial E^{(3)}}{\partial V^{(j)}} \end{cases} \tag{18}$$

Backpropagation method is used to calculate $\Delta W_f^{(3)}$, $\Delta W_i^{(3)}$, $\Delta W_a^{(3)}$, and $\Delta W_o^{(3)}$. Here define one auxiliary variable $\delta^{(j)}$ as the derivative with regard to $C^{(j)}$, as is shown in Equation-19.

$$\delta^{(j)} = \frac{\partial E^{(3)}}{\partial C^{(j)}} \tag{19}$$

Suppose, the value of $\delta^{(j)}$ is available, then $\frac{\partial E^{(3)}}{\partial W_f^{(j)}}$ can be calculate via Equation-20, according to Equation-14 and 13.

$$\begin{aligned}\frac{\partial E^{(3)}}{\partial W_f^{(j)}} &= \frac{\partial E^{(3)}}{\partial C_f^{(j)}} \cdot \frac{\partial C^{(j)}}{\partial W_f^{(j)}} \\ &= \delta^{(j)} \cdot \frac{\partial C^{(j)}}{\partial W_f^{(j)}} \\ &= \delta^{(j)} \cdot \sigma'(W_f \cdot [h^{(j-1)}, x^{(j)}]) \cdot ([h^{(j-1)}, x^{(j)}]) \cdot C^{(j-1)}\end{aligned} \tag{20}$$

Similarly, $\frac{\partial E^{(3)}}{\partial W_i^{(j)}}$ and $\frac{\partial E^{(3)}}{\partial W_a^{(j)}}$ can be calculate with $\delta^{(j)}$. However, it is $C^{(j+1)}$, not $C^{(j)}$ depends on $W_o^{(j)}$ (via $h^{(j)}$), so the computation of $\frac{\partial E^{(3)}}{\partial W_o^{(j)}}$ is kind of more complicated, and will be addressed at the end of this section.

$$\tau^{(j)} = \frac{\partial h^{(j)}}{\partial C^{(j)}} = o^{(j)} \cdot \tanh'(C^{(j)}) \tag{21}$$

Next, we will give the backward recurrence formula to compute $\delta^{(j)}$ based on $\delta^{(j+1)}$ for LSTM. It should be noted that in Equation-14, $f^{(j)}$, $i^{(j)}$ and $A^{(j)}$ also depend on $C^{(j-1)}$ via variable $h^{(j-1)}$. For the easy of notation, another auxiliary variable $\tau^{(j)}$ is defined as the derivative of $h^{(j)}$ with regard to $C^{(j)}$. Based on Equation-16, $\tau^{(j)}$ can be calculate with Equation-21.

$$\begin{aligned}\delta^{(j)} &= \frac{\partial E^{(3)}}{\partial C^{(j)}} \\ &= \frac{\partial E^{(3)}}{\partial C^{(j+1)}} \frac{\partial C^{(j+1)}}{\partial C^{(j)}} \\ &= \delta^{(j+1)} \cdot \frac{\partial [f^{(j+1)} \cdot C^{(j)} + i^{(j+1)} \cdot A^{(j+1)}]}{\partial C^{(j)}} \\ &= \delta^{(j+1)} \cdot (f^{(j+1)} + \frac{\partial f^{(j+1)}}{\partial h^{(j)}} \frac{\partial h^{(j)}}{\partial C^{(j)}} \cdot C^{(j)} + \frac{\partial i^{(j+1)}}{\partial h^{(j)}} \frac{\partial h^{(j)}}{\partial C^{(j)}} \cdot A^{(j+1)} + i^{(j+1)} \cdot \frac{\partial A^{(j+1)}}{\partial h^{(j)}} \frac{\partial h^{(j)}}{\partial C^{(j)}}) \\ &= \boldsymbol{\delta^{(j+1)} \cdot f^{(j+1)}} + \delta^{(j+1)} \tau^{(j)} \cdot (\frac{\partial f^{(j+1)}}{\partial h^{(j)}} \cdot C^{(j)} + \frac{\partial i^{(j+1)}}{\partial h^{(j)}} \cdot A^{(j+1)} + i^{(j+1)} \cdot \frac{\partial A^{(j+1)}}{\partial h^{(j)}})\end{aligned} \tag{22}$$

In Equation-22, the computation of $\frac{\partial f^{(j+1)}}{\partial h^{(j)}}$, $\frac{\partial i^{(j+1)}}{\partial h^{(j)}}$, and $\frac{\partial A^{(j+1)}}{\partial h^{(j)}}$ are very simple. For example, based on Equation-11, $\frac{\partial f^{(j+1)}}{\partial h^{(j)}}$ can be calculated with Equation-23.

$$\frac{\partial f^{(j+1)}}{\partial h^{(j)}} = \sigma'(W_i \cdot [h^{(j)}, x^{(j+1)}]) \cdot W_{i,h} \tag{23}$$

where $W_{i,h}$ is the part which acts on hidden state $h^{(j)}$ of the whole input gate weight W_i .

In summary, with Equation-22, $\delta^{(j)}$ can be computed; with Equation-20 and Equations-18, $\Delta W_f^{(3)}$, $\Delta W_i^{(3)}$, $\Delta W_a^{(3)}$ can be computed from $\delta^{(j)}$. (The computation of $\Delta W_o^{(3)}$ is described in next subsection).

3.2.3 Step 3: Average up the derivatives for the sequence

Similar to RNN, each timestep is a training example, so there are T training examples for a training sequence of length $T + 1$. These training examples can be treated as a mini-batch. Thus, the

derivatives of W, U and V for one sequence (or several sequences) are the average of the derivates for each output, as is shown in Equation-24.

$$\begin{cases} \Delta W_i &= \frac{1}{T}\sum_{i=1}^{T} \Delta W_i^{(i)} \\ \Delta W_a &= \frac{1}{T}\sum_{i=1}^{T} \Delta W_a^{(i)} \\ \Delta W_f &= \frac{1}{T}\sum_{i=1}^{T} \Delta W_f^{(i)} \\ \Delta W_o &= \frac{1}{T}\sum_{i=1}^{T} \Delta W_o^{(i)} \\ \Delta V &= \frac{1}{T}\sum_{i=1}^{T} \frac{\partial E^{(i)}}{\partial V^{(j)}} \end{cases} \tag{24}$$

Where, $T+1$ is the length of the input training sequence, $E^{(i)}$ is the error of time step i.

3.2.4 Step 4: update the parameters

After all the derivates of the parameters of current input sequence (or sequence batch) have been calculated, we can update the parameters with Equations-25.

$$\begin{cases} W_i &= W_i - \lambda \Delta W_i \\ W_a &= W_a - \lambda \Delta W_a \\ W_f &= W_f - \lambda \Delta W_f \\ W_o &= W_o - \lambda \Delta W_o \\ V &= V - \lambda \Delta V \end{cases} \tag{25}$$

where λ is the learning rate.

3.2.5 Note: calculate the derivative for W_o

TODO

3.3 How LSTM handles the gradient vanishing problem

As mentioned earlier, LSTM can handle the gradient vanishing problem much better than vanilla RNN. For the easy of explanation, let's simplify Equation-22 further. In the end of Equation-22, $\delta^{(j)}$ is the sum of two parts. According to Equation-23, and Equation-21, the later part of Equation-22 contains a product of $tanh'(C) \cdot sigmoid'(z)$. Based on the analysis of the Figure-4b, $tanh'(C) \cdot sigmoid'(z)$ is likely two orders smaller than 1. So for here, for analysis we can ignore the later part of Equation-22, and get $\delta^{(j)}$ for LSTM approximately via Equation-26.

$$\delta_{lstm}^{(j)} \approx \delta_{lstm}^{(j+1)} \cdot f^{(j+1)} = \delta_{lstm}^{(j+1)} \cdot \sigma(z_f) \tag{26}$$

As comparison, the $\delta^{(j)}$ for RNN of Equation-6 can be re-written in Equation-27.

$$\delta_{rnn}^{(j)} = \delta_{rnn}^{(j+1)} \cdot W^{(j+1)} \cdot tanh'(z^{(j)}) \tag{27}$$

The main reason that $\delta_{rnn}^{(j)}$ will be shrinking very fast because of the $tanh'(z^{(j)})$ term in Equation-27; however, $\delta_{lstm}^{(j)}$ only has the $\sigma(z_f)$ term, which is much better. As is shown in Figure-7, $tanh'(x)$

can be significantly larger than zero only in the interval $[-4, 4]$; on the other hand, $\sigma(x)$ can be significantly larger than zero in a much larger interval $[-4, +\infty]$. Thus $\delta_{lstm}^{(j)}$ will shrink much slower than $\delta_{rnn}^{(j)}$. In other words, the cell state of LSTM can have much longer memory for previous inputs.

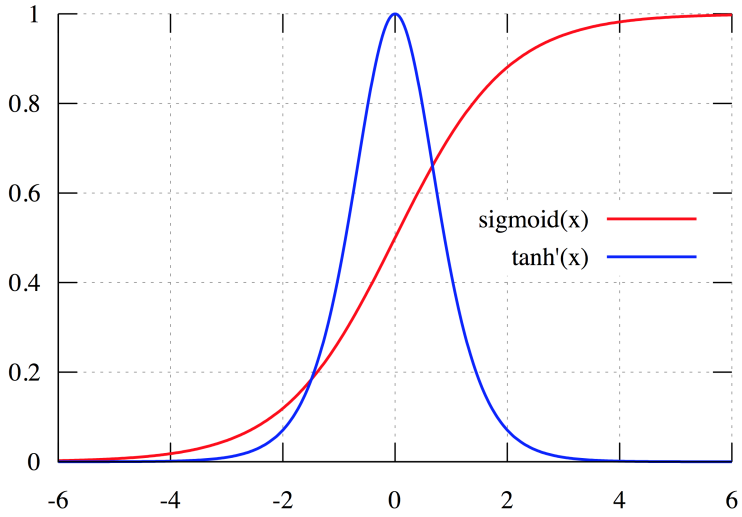


Figure 7: Compare the distribution of $sigmoid(x)$ and $tanh'(x)$

4 Implementation

A prototype of RNN layer has been implemented in my github (**https://github.com/beekbin/simpleRNN**). In this implementation, to be able to be stacked by other sequence layer (for example, another RNN layer, or Pooling layer), the output of the RNN layer is $Vh^{(t)}$, instead of $softmax(Vh^{(t)})$. The $Softmax$ is implemented as a separate layer.

5 Conclusion

This essay has explained:

- The structure of RNN and LSTM;
- How to train RNN and LSTM with BPTT method;
- The gradient vanishing problem of RNN;
- How LSTM handles the gradient vanishing problem better than RNN.