Week11 Pre-Report: Fundamentals of Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) Network

Euijin Hong[1]
[1]Department of Electrical and Electronic Engineering, Yonsei University.

1 Introduction

Recurrent Neural Network (RNN) [2, 3] is a sequence model that processes an input and an output as a sequential data, such as natural language text. RNN is one of the most effective sequence model networks. Long Short-Term Memory (LSTM) Network [1] can be specified as RNNs, but shows more powerful performance in terms of processing long sequential data. This is because the 'vanilla RNN' has its limitation in dealing with long sequences, which is also known as *the problem of Long-Term Dependencies*, where the previous information is not delivered properly as time step goes on due to gradient vanishing and exploding. We can see that LSTM solved the limitation of RNN by simply adding structures and algorithms that enables the network to select which data to forget and which of them to maintain. In the following sections, we will discover the theory and network configuration of RNN and LSTM Network.

2 Theory and Configuration of RNN

In 'Feed Forward Neural Networks', the values that passed the activation function in hidden layers headed towards the output layers. However, RNN passes through the result of the activation functions in the hidden layer to the output layer, but also to the input of the computation of the next hidden layer.

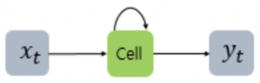

Figure 1: The diagram of a single cell of RNN.

As shown in Figure 1, where x is an input vector of an input layer and y is an output vector of an output layer, the node that exports the output through the activation function in the hidden layer is called a *cell*. The cell can also be referred to as memory cell or RNN cell, since it 'memorizes' the computation of the previous values.

The cell of a hidden layer shows its recurrent behaviour where it uses the hidden layer value of its previous time step as its input for each time step. This also means that the hidden layer value of current time step t is influenced by the values of the past cells.

The value that a cell produces and exports to the next time step $t+1$ is defined as a *hidden state*. The flow of each time step and its hidden state can be plotted as in Figure 2. The RNN cells can be connected to form the model with various length of input and output. For example, in tasks such as image captioning, multiple outputs come from a single input, thus require a one-to-many structure model. However, in tasks such as sentiment classification or spam mail detection, a single output comes from multiple input, hence require a many-to-one structure model.

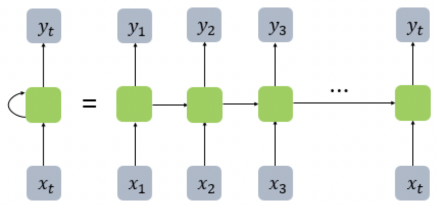

Figure 2: A diagram of an RNN that describes the flow of each hidden step in terms of time steps. The input and output x_t and y_t represents the input and output of each time step. We can see that the cell output of the previous time step serves as an input of the current time step cell.

Equations for RNN Let the hidden state in the current time step be given as h_t, the weight for the input layer W_x, and the weight W_h for the hidden state of the previous time step h_{t-1}. Then the output of the hidden layer and the output layer can be expressed as the following equations, where f is a non-linear activation function:

$$\begin{aligned} h_t &= tanh(W_x x_t + W_h h_{t-1} + b) \\ y_t &= f(W_y h_t + b) \end{aligned} \tag{1}$$

While computing h_t, *tanh* is normally used as an activation function. The weight parameters W_x, W_h, W_y shares the identical value among all time step in the same layer. But, when there are two or more hidden layers, the values of the weight parameters are different with each other.

The computation of the equation can also be implemented as a vetor and matrix computation. When the input vector is given as d and the size of the hidden state as D_h, then the size of each vector and matrix are as follows:

$$\begin{aligned} x_t &: (d \times 1) \\ W_x &: (D_h \times d) \\ W_h &: (D_h \times D_h) \\ h_{t-1} &: (D_h \times 1) \\ b &: (D_h \times 1) \end{aligned} \tag{2}$$

When we suppose the batch size to be 1, d and D_h to be 4, then we can depict the computation in hidden layer of RNN as in Figure 3.

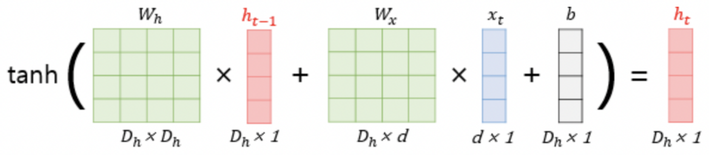

Figure 3: The matrix expression of hidden layer computation in RNN.

The output layer uses its activation function depending on the type of the task given. For example, in case when the network has to perform a binary classification, the output layer can make use of a sigmoid function based on logistic regression, and in multiple classification, the output layer can use sigmoid regression from a sigmoid function.

Backward Propagation As in other neural networks, RNN learns itself by back propagating and updating the gradients to improve its accuracy and reduce the loss. The overall process of back propagation can be depicted as in Figure 4.

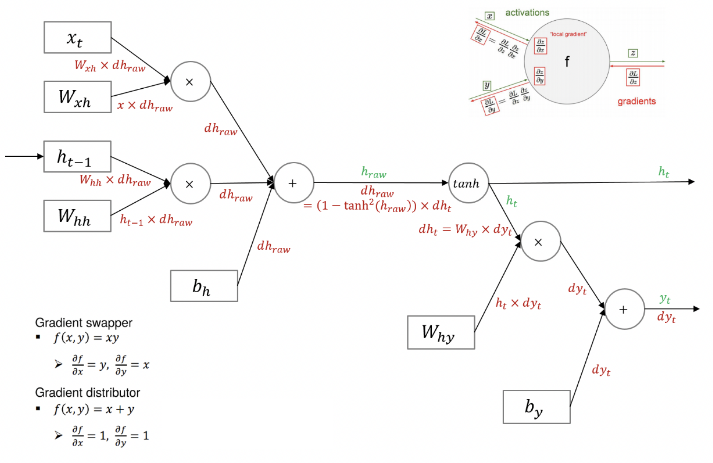

Figure 4: An overall backward propagation of an RNN cell.

As we have seen before, the output obtained through forward pass is y_t. Thus, dL/dy_t, the gradient of y_t with respect to the final loss L, comes first in the computation of backward propagation. Then, this calculation result is distributed to both sides of the addition block. dW_{hy} can be obtained by multiplying the gradient dL/dy_t and h_t, while dh_t is obtained by multiplying

This is an extended abstract. The full paper is available at the Fundamentals of Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) Network.

dL/dy_t and W_{hy}. Furthermore, dh_{raw} can be computed by multiplying the inflow gradient dh_t and the local gradient $1 - tanh^2(h_{raw})$. The rest of the back propagation process can be explained in an identical way. The noteworthy fact is that since RNN is a recurrent model in terms of its hidden nodes, h_{t-1} is reflected in obtaining h_t. In other words, the gradients from h_{t-1} are also added and reflected in back propagation process.

3 Theory and Configuration of Long Short-Term Memory(LSTM)

As mentioned before, there existed a problem of vanilla RNN known as *the problem of Long-Term Dependencies*. Literally, the problem occurs especially in processing long sequential data, where the gradient vanishes or explodes depending on the weight values multiplied in back propagation. To overcome the problem, the *Long Short-Term Memory (LSTM)* is introduced. LSTM decides what information to forget or to maintain, through adding the input gate, forget gate, and the output gate in its model configuration. In its algorithm, a new value called 'cell state' is introduced, whereas C_t is a cell state in a current time step. The cell state of the previous time step is also used s an input to obtain the cell state of the current time step. Figure 5 shows the overall structure of a LSTM network.

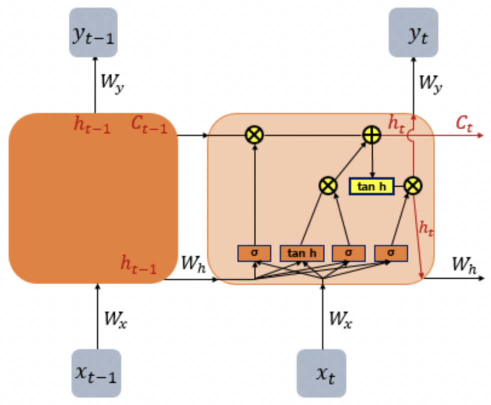

Figure 5: An overall structure of a LSTM network. The upper flow from c_{t-1} to C_t shows an update of a cell state. The input gate, forget gate, and the output gate are also described in the figure.

To calculate the values of the hidden state and the cell state, three new gates are used. These gates are called the forget gate, the input gate, and the output gate. All three gates have a common sigmoid function. Passing through the sigmoid function results in values between 0 and 1, which are used to control the gates. Let's look at each gate in more detail, referring to the following content.

- In the following equations, σ represents the sigmoid function.
- In the following equations, tanh represents the hyperbolic tangent function.
- $W_{xi}, W_{xg}, W_{xf}, W_{xo}$ are the four weights used with x_t in each gate.
- $W_{hi}, W_{hg}, W_{hf}, W_{ho}$ are the four weights used with h_{t-1} in each gate.
- b_i, b_g, b_f, b_o are the four biases used in each gate.

Input Gate is a gate for remembering the current information. First, we multiply the current x value at time step t by the weight W_{zi}, which connects to the input gate, and add it to the product of the previous hidden state at time step $t-1$ multiplied by the weight W_{hi}, which connects to the input gate. This sum passes through the sigmoid function, resulting in the value i_t. Next, we multiply the current x value at time step t by the weight W_{xg}, which connects to the input gate, and add it to the product of the previous hidden state at time step $t-1$ multiplied by the weight W_{hg}, which connects to the input gate. This sum passes through the hyperbolic tangent function, resulting in the value g_t. The sigmoid function provides a value between 0 and 1 for i_t, while the hyperbolic tangent function provides a value between -1 and 1 for g_t. These two values determine the amount of information to be memorized in this step. The specific mechanism for determining this will be explained in the cell state equation below.

$$\begin{aligned} i_t &= \sigma(W_{x_i}x_t + W_{h_i}h_{t-1} + b_i) \\ g_t &= tanh(W_{x_g}x_t + W_{h_g}h_{t-1} + b_g) \end{aligned} \tag{3}$$

Forget Gate is a gate for removing the memory. The current x value at time step t and the previous hidden state at time step $t-1$ pass through the sigmoid function. The sigmoid function produces a value between 0 and 1, which represents the amount of information that has undergone the forget process. A value closer to 0 indicates that more information has been discarded, while a value closer to 1 indicates that the information has been fully maintained. With this value, the cell state is determined, and the specific cell state equation, which will be explained below, is referenced.

$$f_t = \sigma(W_{x_f}x_t + W_{h_f}h_{t-1} + b_f) \tag{4}$$

Cell State can be obtained by the following process. Currently, the cell state C_t lost certain amount of its memory.

We perform an element-wise multiplication (entry-wise product) on the two values i_t and g_t obtained from the input gate. In other words, it means multiplying the corresponding elements of two matrices of the same size. In this context, it is denoted as $\odot$. This result represents the information selected to be remembered in this step.

We add the selected memory from the input gate to the result of the forget gate. This value is referred to as the cell state at the current time step t, and it is passed on to the next LSTM cell at time step $t+1$.

Let's understand the influence of the forget gate and the input gate. If the output value f_t of the forget gate becomes 0, the influence of the previous time step's cell state value C_{t-1} on the current time step's cell state value C_t becomes 0. In this case, only the result of the input gate can determine the value of the current time step's cell state C_t. This signifies that the forget gate is completely closed, and only the input gate is open. Conversely, if we assume that the value of the input gate i_t is 0, the value of the current time step's cell state C_t depends solely on the value of the previous time step's cell state C_{t-1}. This indicates that the input gate is completely closed, and only the forget gate is open. Ultimately, the forget gate determines how much of the previous input to consider, while the input gate determines how much of the current input to consider.

$$C_t = f_t \odot C_{t-1} + i_t \odot g_t \tag{5}$$

Output Gate and Hidden State can be obtained as follows.

$$\begin{aligned} o_t &= \sigma(W_{x_o}x_t + W_{h_o}h_{t-1} + b_o) \\ h_t &= o_t \odot tanh(c_t) \end{aligned} \tag{6}$$

The output gate is the result of passing the current x value at time step t and the previous hidden state at time step $t-1$ through the sigmoid function. This value is used in determining the current hidden state at time step t. The value of the cell state passes through the hyperbolic tangent function, becoming a value between -1 and 1. This value is then multiplied by the output gate's value, resulting in a filtering effect that produces the hidden state. The hidden state also propagates to the output layer.

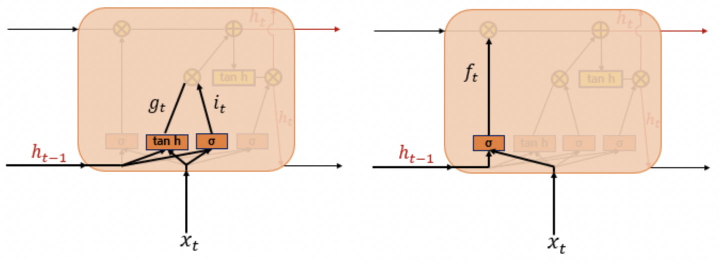

Figure 6: (left): Configuration of Input Gate, (right): Configuration of Forget Gate.

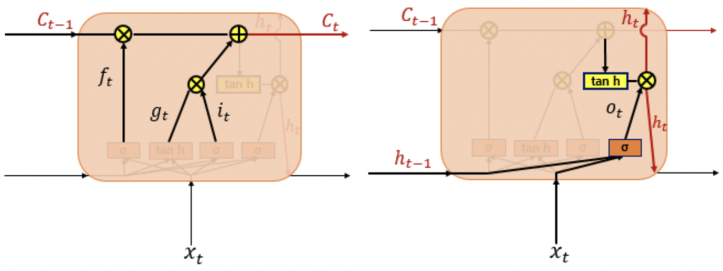

Figure 7: (left): Configuration of Cell State, (right): Configuration of Output Gate and Hidden State.

[1] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[2] MI Jordan. Serial order: a parallel distributed processing approach. technical report, june 1985-march 1986. Technical report, California Univ., San Diego, La Jolla (USA). Inst. for Cognitive Science, 1986.

[3] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning internal representations by error propagation. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science, 1985.