LSTM and GRU Lecture Notes (EE698R)

Parampreet Singh(21104268), Thishyan Raj T(20204420)

April 2022

1 Time Series Modelling

Time series data is recurrent. To model such data, we need the value of previous inputs. We use RNNs to model time series.

2 Recurrent Neural Network (RNN)

We can see RNN represented using normal NN with an additional delay block added to it as shown below:

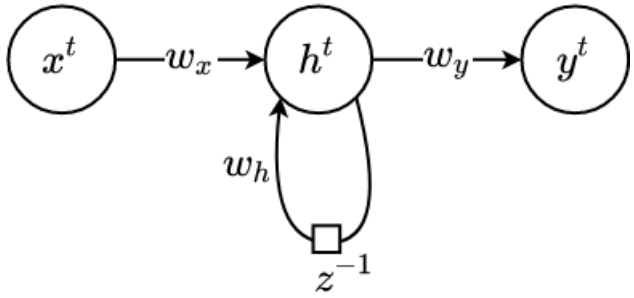

Figure 1: A graphical representation of a simple Recurrent Neural Network (RNN).

Here,

- x^t - represents the inputs
- w_x - represents the weights between the input layer and the hidden layers.
- h_t - represents the hidden layers.
- w_y - represents the weights between the hidden layers and the output layers.

- y_t - represents the output layer.
- z^{-1} - represents the delay block to feedback past outputs of the hidden layers to the current input.
- w_h - represents the scaling factor to feed the delayed output of the hidden layers.

Mathematically, the we can write the input output equations as :

$$h^t = f(w_x x^t + w_h h^{t-1})$$
$$y^t = f(w_y h^t)$$

Apart from graphical and mathemetical representation, we have other representation called the 'rolled out graph representation' as shown below:

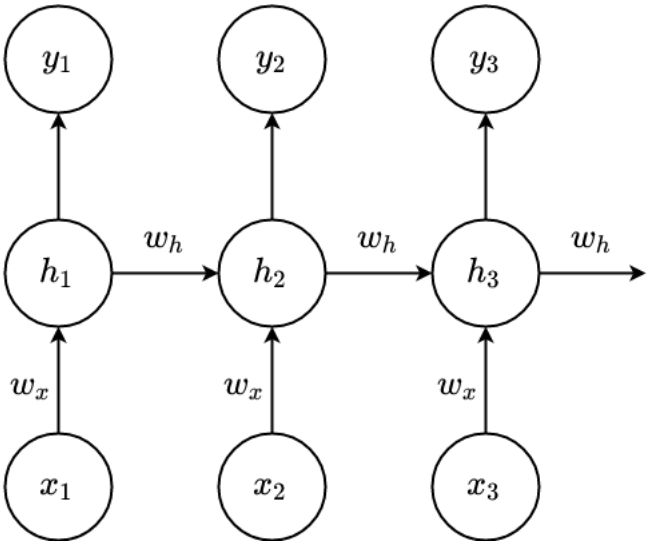

Figure 2: Representation of an RNN unfolded over time.

3 Training

We train the RNN using Back Propagation Through Time (BPTT). Consider the loss L^t at some instant t. Suppose we would like to find the parameter w_x that minimizes the loss L^3 at time instant $t = 3$, we get,

$$\begin{aligned}\frac{\partial L^3}{\partial w_x} &= \frac{\partial L^3}{\partial h^3}\frac{\partial h^3}{\partial w_x} + \frac{\partial L^3}{\partial h^3}\frac{\partial h^3}{\partial h^2}\frac{\partial h^2}{\partial w_x} + \frac{\partial L^3}{\partial h^3}\frac{\partial h^3}{\partial h^2}\frac{\partial h^2}{\partial h^1}\frac{\partial h^1}{\partial w_x} \\ &= \frac{\partial L^3}{\partial h^3}\left(\frac{\partial h^3}{\partial w_x} + \frac{\partial h^3}{\partial h^2}\left(\frac{\partial h^2}{\partial w_x} + \frac{\partial h^2}{\partial h^1}\frac{\partial h^1}{\partial w_x}\right)\right)\end{aligned}$$

This algorithm presents some challenges :

- It could become infinitely unfolded based on the number of time instants up to which past information needs to be stored. In such cases, we may have to limit the number of time instants that need to be remembered.
- But the main problem is that of vanishing gradients. If we have any terms in between gradients with very small values, it will make all the succeeding terms vanish.
- If w_h is very small, it will create a problem as it will be repeatedly multiplied and will make the gradients vanish.
- Consider the case where y^3 depends upon the previous input x^1. Now, it should depend upon the w_x related to x^1 and not the w_x from third input.
- Also, if we need long term dependencies as in video analysis (to detect hand waving) etc., then this algorithm will create a problem for the same reason as above.
- Another problem is that of Gradient explosion. The exploding gradient is the inverse of the vanishing gradient and occurs when large error gradients accumulate, resulting in extremely large updates to neural network model weights during training. As a result, the model is unstable and incapable of learning from your training data.

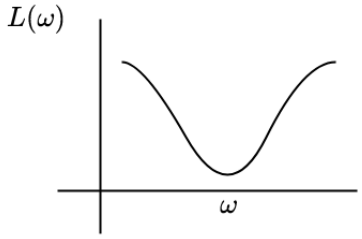

Figure 3: The problem of exploding gradients while training RNNs.

This can be easily countered if we clip the gradient. So, we take

$$\left|\frac{\partial L}{\partial w}\right| = \min\left(\left|\frac{\partial L}{\partial w}\right|, C\right)$$

where C is some constant, that puts an upper limit on the gradients.

In case of RNNs, gradients vanishing problem is difficult to handle. So to tackle this problem, we use memory units in between layers to save the required information. Now we can retrieve or forget the saved information in the memory units, according to need. By doing this, we avoid the same term being repeatedly multiplied. This method is called as LSTM.

4 Long Short Term Memory (LSTM)

LSTM is a type of RNN architecture, with feedback connections. A common LSTM unit is composed of a cell, an input gate, an output gate and a forget gate. The cell remembers values over arbitrary time intervals and the three gates regulate the flow of information into and out of the cell.

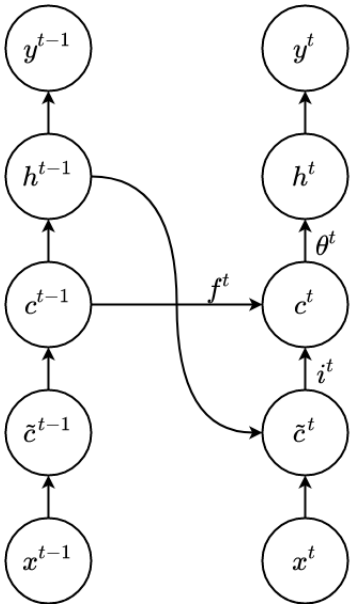

Figure 4: A typical LSTM unit architecture.

In a general LSTM architecture, we use memory units c^t and c^{t-1}, that depend upon $\tilde{c}^t$ and $\tilde{c}^{t-1}$ (dummy memory states). i^t is a vector, which represents the gate, and f^t is another vector, which represents the forget gate. The memory unit c^t receives both old and new information and the weightage for each of them is adjusted using i^t and f^t. o^t is called as output gate and can control the information flow from memory unit c^t to h^t. i^t, f^t, and o^t are functions of x^t and h^{t-1}.

5 Gated Recurrent Unit (GRU)

Gated recurrent unit is a simplified version of LSTM as shown below. The GRU is like a long short-term memory (LSTM) with a forget gate, but has fewer parameters than LSTM, as it lacks an output gate.

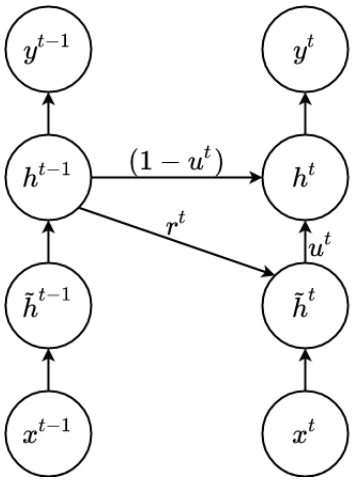

Figure 5: The GRU units, are a simplified version of LSTM.

Here, we have r^t as retention gate and u^t as update gate. For control of information from h^{t-1} to h^t, we use $(1-u^t)$ instead of creating a new variable to store that value, further reducing the number of variables. Now, if we compute $\frac{\partial L^3}{\partial w_x}$ for GRU, we won't have w_h, but we have gates instead. So, if the gate does not want to forget the previous state value, it will be multiplying a higher value and hence the gradient won't vanish. But when the gate has a smaller value, that means we are supposed to forget those previous state values as there is no dependency on those previous states. So vanishing gradients is actually aiding what we want by removing unwanted previous state values.