Recurrent Neural Network

Application to Stock Market Forecasting

Bassem Ben Hamed

National School of Electronics and Telecommunications of Sfax

July 27, 2021

Plan

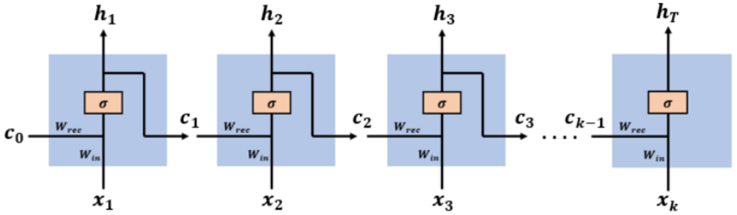
RNNs enable modelling time-dependent and sequential data tasks, such as stock market prediction, machine translation, text generation and many more.

However, RNNs suffer from the problem of vanishing gradients, which hampers learning of long data sequences.

Two main questions arise:

1. Why do RNNs suffer from vanishing and exploding gradients?
2. How do LSTMs keep the gradients from vanishing or explode?

Let's have a short reminder of how RNNs look like. The network looks like this:



The network has an input sequence of vectors $(x(1), x(2), \ldots, x(k))$, at time step t the network has an input vector $x(t)$. Past information and learned knowledge is encoded in the network state vectors $(c(1), c(2), \ldots, c(k-1))$, at time step t the network has an input state vector $c(t-1)$. The input vector $x(t)$ and the state vector $c(t-1)$ are concatenated to comprise the complete input vector at time step t, $[c(t-1), x(t)]$.

The network has two weight matrices: W_{rec} and W_{in} connecting $c(t-1)$ and $x(t)$, the two parts of the input vector $[c(t-1), x(t)]$, to the hidden layer. For simplicity, we leave out the bias vectors in our computations, and denote $W = [W_{rec}, W_{in}]$.

The sigmoid function is used as the activation function in the hidden layer.

The network outputs a single vector at the last time step (RNNs can output a vector on each time step, but well use this simpler model).

Plan

After the RNN outputs the prediction vector $h(k)$, we compute the prediction error $E(k)$ and use the Back Propagation Through time algorithm to compute the gradient

$$\frac{\partial E}{\partial W} = \sum_{t=1}^{T} \frac{\partial E_t}{\partial W}$$

The gradient is used to update the model parameters by:

$$W := W - \alpha \frac{\partial E}{\partial W}$$

Say we have learning task that includes T time steps, the gradient of the error on the k time step is given by:

$$\frac{\partial E_k}{\partial W} = \frac{\partial E_k}{\partial h_k}\frac{\partial h_k}{\partial c_k}\left(\prod_{t=2}^{k}\frac{\partial c_t}{\partial c_{t-1}}\right)\frac{\partial c_1}{\partial W} \quad (1)$$

Notice that since $W = [W_{rec}, W_{in}]$, $c(t)$ can be written as:

$$c_t = \sigma\left(W_{rec}.c_{t-1} + W_{in}.x_t\right)$$

Compute the derivative of $c(t)$ and get:

$$\frac{\partial c_t}{\partial c_{t-1}} = \sigma'\left(W_{rec}.c_{t-1} + W_{in}.x_t\right).W_{rec} \quad (2)$$

Plug (2) into (1) and get our backpropagated gradient

$$\frac{\partial E_k}{\partial W} = \frac{\partial E_k}{\partial h_k}\frac{\partial h_k}{\partial c_k}\left(\prod_{t=2}^{k}\sigma'\left(W_{rec}.c_{t-1} + W_{in}.x_t\right).W_{rec}\right)\frac{\partial c_1}{\partial W}$$

The last expression **tends to vanish when k is large**, this is due to the derivative of the sigmoid activation function which is smaller than 1.

Remark

The **product of derivatives can also explode** if the weights Wrec are large enough to overpower the smaller tanh derivative, this is known as the exploding gradient problem.

We have:

$$\prod_{t=2}^{k} \sigma' \left(W_{rec}.c_{t-1} + W_{in}.x_t \right).W_{rec} \to 0$$

So for some time step k:

$$\frac{\partial E_k}{\partial W} \to 0$$

And our complete error gradient will vanish

$$\frac{\partial E}{\partial W} = \sum_{t=1}^{T} \frac{\partial E_k}{\partial W} \to 0$$

The network's weights update will be:

$$W := W - \alpha \frac{\partial E}{\partial W} \simeq W$$

And no significant learning will be done in reasonable time.

Plan

An LSTM network has an input vector $[h(t-1), x(t)]$ at time step t. The network cell state is denoted by $c(t)$. The output vectors passed through the network between consecutive time steps t, $t+1$ are denoted by $h(t)$.

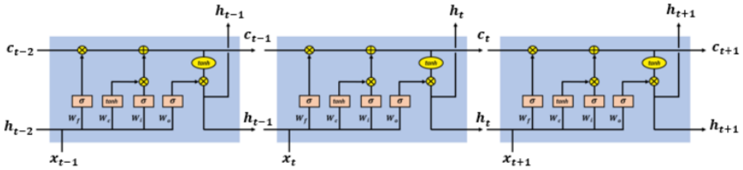


Figure: LSTM network cells at time steps $t-1$,t, $t+1$

An LSTM network has three gates that update and control the cell states, these are the forget gate, input gate and output gate. The gates use hyperbolic tangent and sigmoid activation functions.

The forget gate controls what information in the cell state to forget, given new information than entered the network.

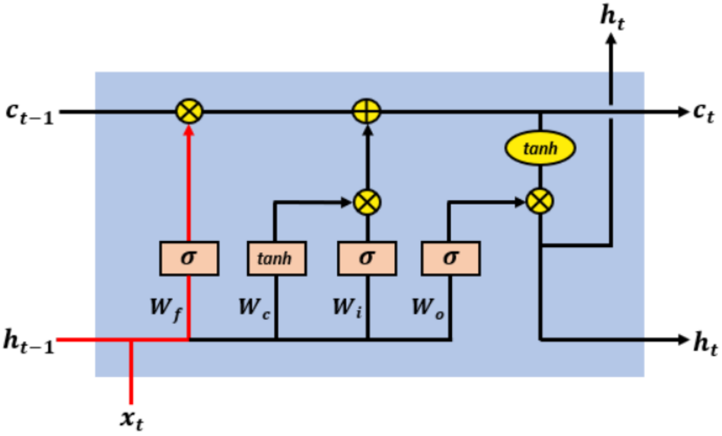


Figure: the LSTM forget gate update of the cell state

The forget gate's output is given by: $f_t = \sigma\left(Wf.[h_{t-1}, x_t]\right)$

The input gate controls what new information will be encoded into the cell state, given the new input information.

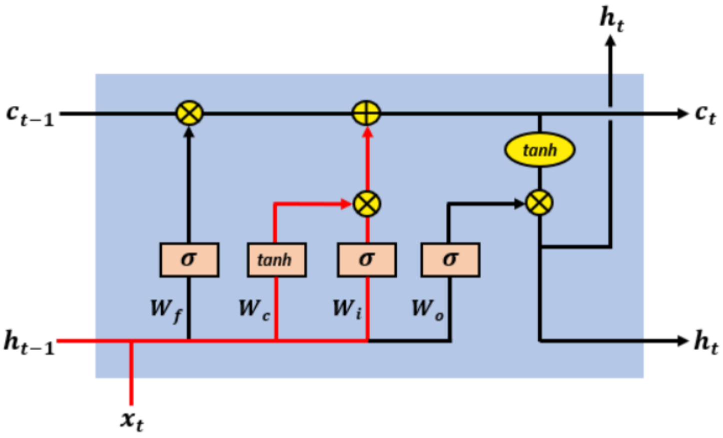


Figure: the LSTM input gate update of the cell state

The input gate's output has the form:

$$\tanh\left(W_c.\left[h_{t-1}, x_t\right]\right) \otimes \sigma\left(W_i.\left[h_{t-1}, x_t\right]\right)$$

and is equal to the element-wise product of the outputs of the two fully connected layers:

$$\widetilde{c}_t = \tanh\left(W_c.\left[h_{t-1}, x_t\right]\right)$$
$$i_t = \sigma\left(W_i.\left[h_{t-1}, x_t\right]\right)$$

The output gate controls what information encoded in the cell state is sent to the network as input in the following time step, this is done via the output vector $h(t)$.

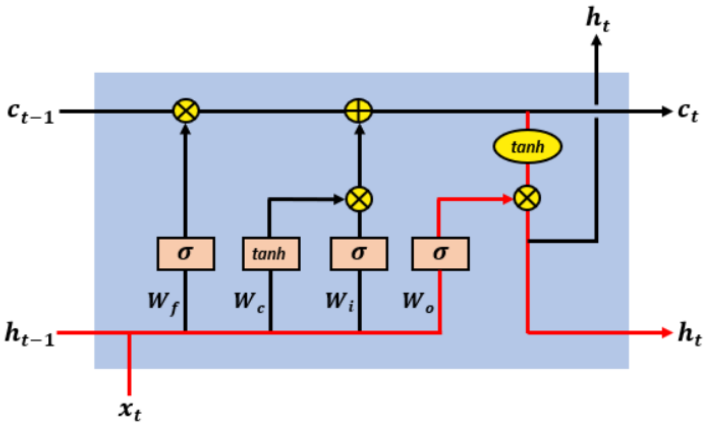


Figure: the LSTM outpute gate update of the cell state

The output gate's activations are given by: $o_t = \sigma\left(W_0 . \left[h_{t-1}, x_t\right]\right)$

and the cell's output vector is given by: $h_t = o_t \otimes \tanh\left(c_t\right)$

The LSTM cell state

The long term dependencies and relations are encoded in the cell state vectors and it's the cell state derivative that can prevent the LSTM gradients from vanishing. The LSTM cell state has the form:

$$c_t = c_{t-1} \otimes f_t \oplus \widetilde{c}_t \otimes i_t \tag{3}$$

Then, the derivate of the cell state is given by:

$$\frac{\partial c_t}{\partial c_{t-1}} = A_t + B_t + C_t + D_t$$

where

$$\begin{aligned}
A_t &= \sigma'\left(W_f.\left[h_{t-1}, x_t\right]\right).W_f.o_{t-1} \otimes \tanh'\left(c_{t-1}\right).c_{t-1} \\
B_t &= f_t \\
C_t &= \sigma'\left(W_i.\left[h_{t-1}, x_t\right]\right).W_i.o_{t-1} \otimes \tanh'\left(c_{t-1}\right).\widetilde{c}_t \\
D_t &= \sigma'\left(W_c.\left[h_{t-1}, x_t\right]\right).W_c.o_{t-1} \otimes \tanh'\left(c_{t-1}\right).i_t
\end{aligned}$$

Then, we can find a suitable parameter update of the forget gate at time step $k+1$ such that:

$$\frac{\partial E_{k+1}}{\partial W} \nrightarrow 0$$