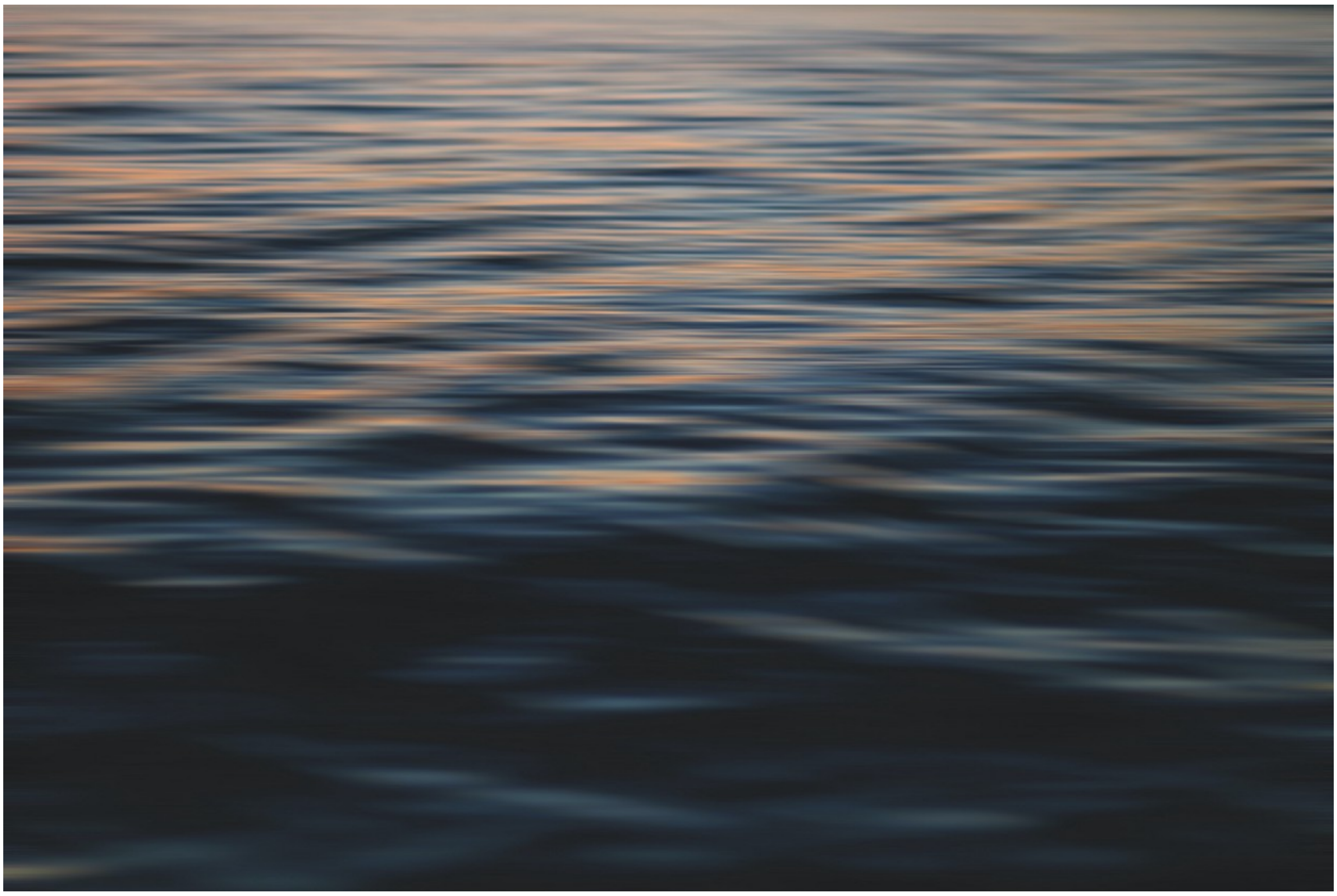


Constant flow. Photo by Jeremy Bishop on Unsplash

HANDS-ON TUTORIALS

How the LSTM improves the RNN

Tiago Miguel 5 days ago · 8 min read ★

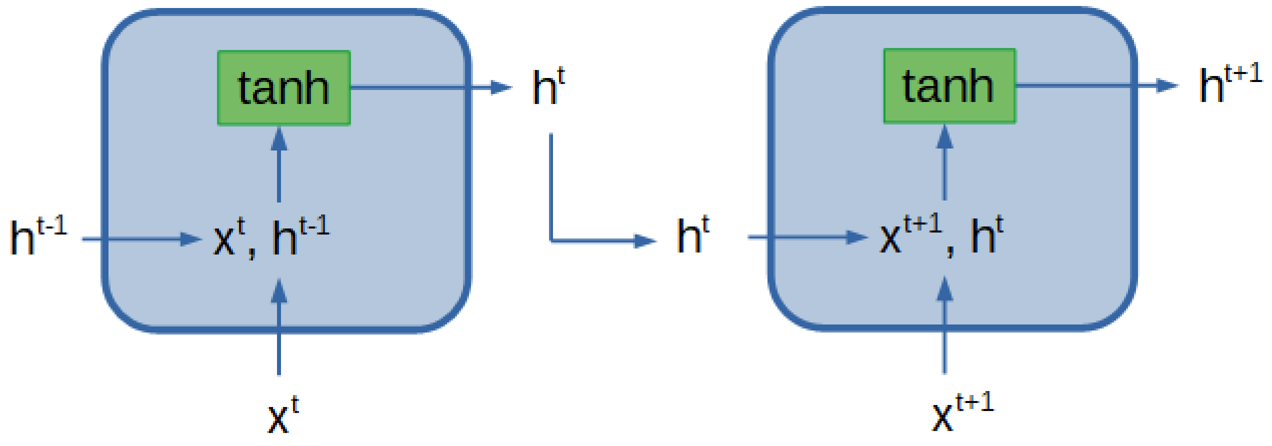
The advantage of the Long Short-Term Memory (LSTM) network over other recurrent networks back in 1997 came from an improved method of back propagating the error. Hochreiter and Schmidhuber called it “constant error back propagation” [1].

But what does it mean to be “constant”? We’ll go through the architecture of the LSTM and understand how it forward and back propagates to answer the question. We will make some comparisons to the Recurrent Neural Network (RNN) along the way. If you are not familiar with the RNN, you may want to read about it here.

However, we should first understand what is the issue of the RNN that demanded for the solution the LSTM presents. That issue is the **exploding** and **vanishing** of the gradients that comes from the backward propagation step.

Vanishing and exploding gradients

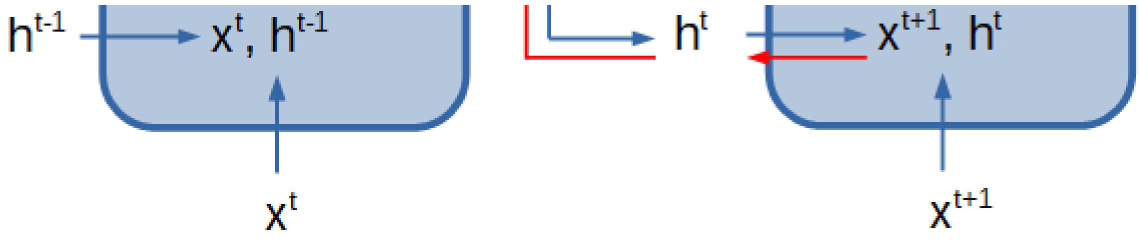
Back propagation is the propagation of the error from its prediction up until the weights and biases. In recurrent networks like the RNN and the LSTM this term was also coined **Back Propgation Through Time** (BPTT) since it propagates through all time steps even though the weight and bias matrices are always the same.



Portion of a typical RNN with two time steps. Figure by author.

The figure above depicts a portion of a typical RNN with two inputs. The green rectangle represents the feed forward calculation of net inputs and their hidden state activations using an hyperbolic tangent (tanh) function. The feed forward calculations use the same set of parameters (weight and bias) in all time steps.

tanh
h^t
tanh
h^{t+1}



Forward propagation path (blue) and back propagation path (red) of a portion of a typical RNN. Figure by author.

In red we see the BPTT path. For large sequences one can see that the calculations stack. This is important because it creates an exponential factor that depends greatly on the values of our weights. Everytime we go back a time step, we need to make an inner product between our current gradient and the weight matrix.

We can imagine our weight matrix to be a scalar and let's say that the absolute scalar is either around 0.9 or 1.1. Also, we have a sequence as big as 100 time steps. The exponential factor created by multiplying these values one hundred times would raise a **vanishing gradient** issue for 0.9:

> ***0.9^{100} = 0.000017(…)***

and an **exploding gradient** issue for 1.1:

> ***1.1^{100} = 13780.61(…)***

Essencially, the BPTT calculation at the last time step would be similar to the following:

$$\frac{dL}{dW^1} = \frac{dL}{d\hat{y}} \frac{d\hat{y}}{dh^n} \frac{dh^n}{dh^{n-1}} (\cdots) \frac{dh^3}{dh^2} \frac{dh^2}{dh^1} \frac{dh^1}{dW^1}$$

$$\frac{dL}{dW^1} = \frac{dL}{d\hat{y}} (W_h)^T (W_n)^T (\cdots) (W_3)^T (W_2)^T \frac{dh^1}{dW^1}$$

Note that although the representation is not completely accurate, it gives a good idea of the exponential stacking of the weight matrices in the BPTT of an RNN with n inputs. *W_h* is the weight matrix of the last linear layer of the RNN.

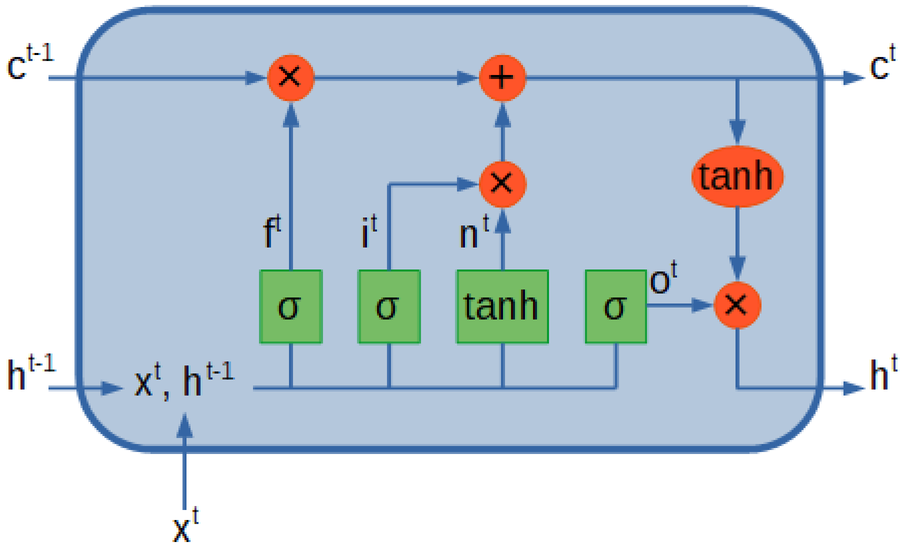
$$W_{new} = W_{old} - \eta \frac{dL}{dW}$$

Next, we would be adding a portion of these values to the weight and bias matrices. You can see that we either barely improve the parameters, or try to improve so much that it backfires.

Now that we understand these concepts of vanishing and exploding gradients, we can move on to learn the LSTM. Let's start by its forward pass.

LSTM forward propagation

Despite the differences that make the LSTM a more powerful network than RNN, there are still some similarities. It mantains the input and output configurations of one-to-one, many-to-one, one-to-many and many-to many. Also, one may choose to use a stacked configuration.



Representation of an LSTM cell. Figure by author.

Above we can see the forward propagation inside an LSTM cell. It is considerably more complicated than the simple RNN. It contains four networks activated by either the sigmoid function (σ) or the tanh function, all with their own different set of parameters.

Each of these networks, also refered to as gates, have a different purpose. They will transform the cell state for time step t ($c\hat{}t$) with the relevant information that should be passed to the next time step. The orange circles/elipse are element-wise transformations of the matrices that preceed them. Here's what the gates do:

Forget gate layer (f): Decides which information to forget from the cell state using a σ function that modulates the information between 0 and 1. It forgets everything that is 0, remembers all that is 1 and everything in the middle are possible candidates.

Input gate layer (i): This could also be a remember gate. It decides which of the new candidates are relevant for this time step also with the help of a σ function.

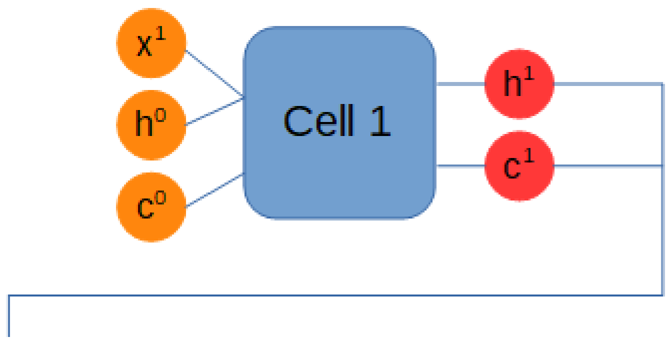
New candidate gate layer (n): Creates a new set of candidates to be stored in the cell state. The relevancy of these new candidates will be modulated by the element-wise multiplication with the input gate layer.

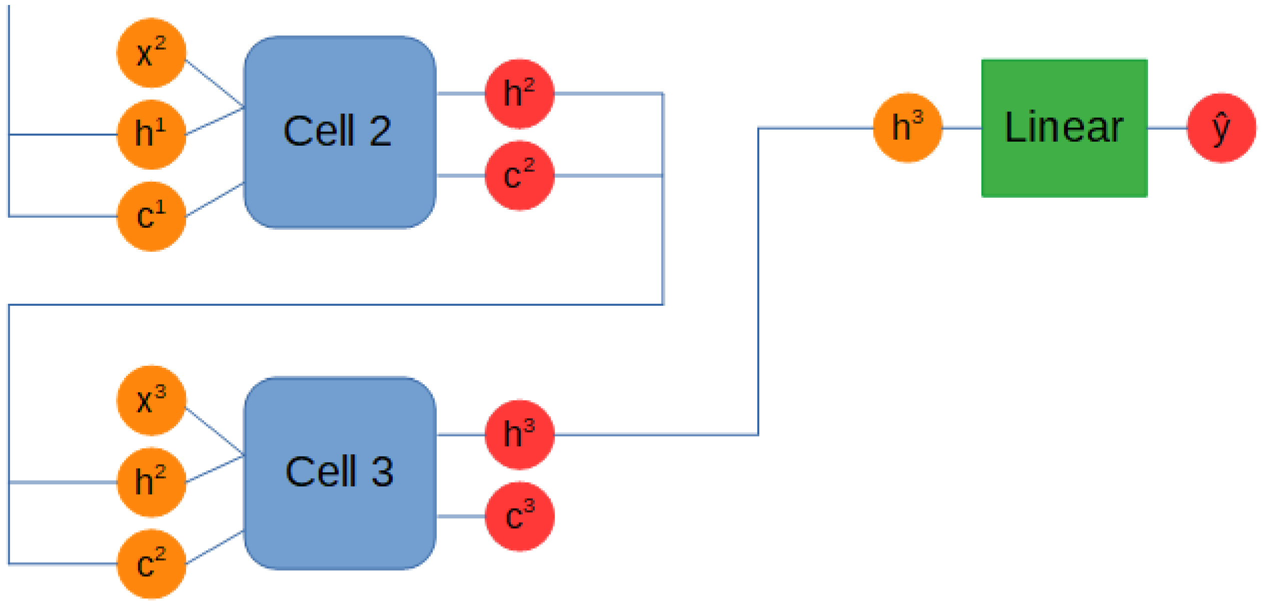
Output gate layer (o): Determines which parts of the cell state are output. The cell state is normalized through a tanh function and is multiplied element-wise by the output gate that decides which relevant new candidate should be output by the hidden state.

$$
\begin{gathered}
f^t = \sigma\left(\left(X^t\right)^T \cdot W_f + B_f\right) \\
i^t = \sigma\left(\left(X^t\right)^T \cdot W_i + B_i\right) \\
n^t = \tanh\left(\left(X^t\right)^T \cdot W_n + B_n\right) \\
o^t = \sigma\left(\left(X^t\right)^T \cdot W_o + B_o\right) \\
c^t = \left(f^t \times c^{t-1}\right) + \left(i^t \times n^t\right) \\
h^t = o^t \times \tanh\left(c^t\right) \\
\hat{y} = h^t \cdot W_h + B_h \\
L = Loss\left(\hat{y}, y\right)
\end{gathered}
$$

On the left you can see the calculations performed inside an LSTM cell. The last two calculations are an external feed forward layer to obtain a prediction and some loss function that takes the prediction and the true value.

The entire LSTM network's architecture is built to deal with a three time step input sequence and to forecast a time step into the future like shown in the following figure:





Representation of an LSTM with three inputs and one output. Figure by author.

Putting the inputs and parameters into vector and matrix form may help understand the dimansionality of the calculations. Note that we are using four weight and bias matrices with their own values.

$$X^t = \begin{bmatrix} x^t \\ h_1^{t-1} \\ h_2^{t-1} \end{bmatrix} W_{(o,n,i,f)} = \begin{bmatrix} w_{11}^x & w_{12}^x \\ w_{11}^h & w_{12}^h \\ w_{21}^h & w_{22}^h \end{bmatrix} B_{(o,n,i,f)} = \begin{bmatrix} b_{11} & b_{12} \end{bmatrix}$$

This is the forward propagation of the LSTM. Now it is time to understand how the network back propagates and how it shines compared to the RNN.

LSTM back propagation

The improved learning of the LSTM allows the user to train models using sequences with several hundreds of time steps, something the RNN struggles to do.

Something that wasn't mentioned when explaining the gates is that it is their job to decide the relevancy of information that is stored in the cell and hidden states so that, when back propagating from cell to cell, the passed error is as close to 1 as possible. This ensures that there is no vanishing or exploding of gradients.

Another simpler way of understanding the process is that the cell state connects the layers inside the cell with information that stabilizes the propagation of error somewhat like a ResNet does.

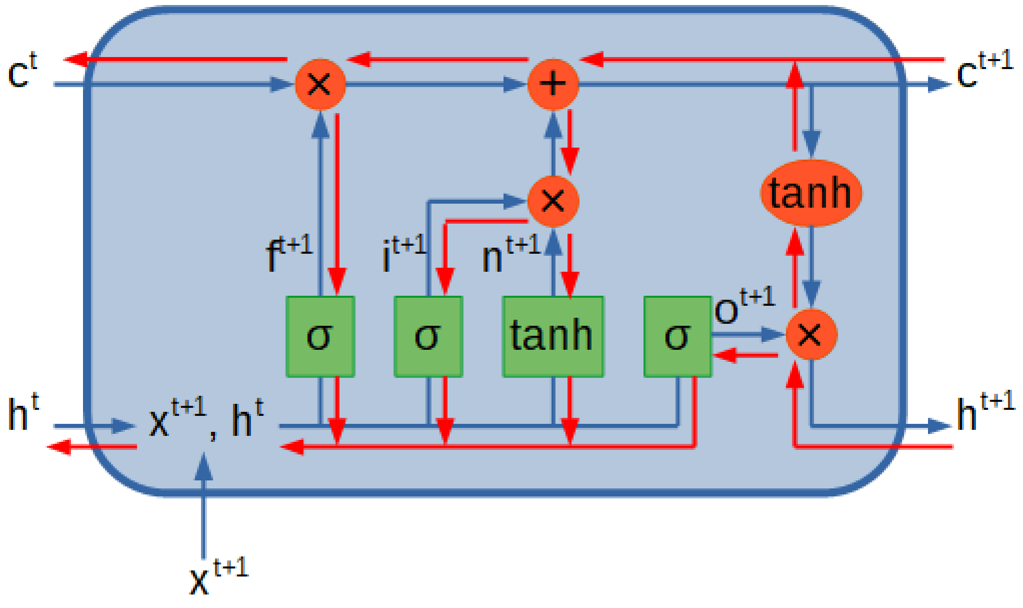
Let's see how the error is kept constant by going through the back propagation calculations. We'll start with the linear output layer.

$$\frac{dL}{d\hat{y}} = y - \hat{y}$$

$$\frac{dL}{dW_{\hat{y}}} = \frac{dL}{d\hat{y}}\frac{d\hat{y}}{dW_{\hat{y}}} = \left(\frac{dL}{d\hat{y}} \cdot (h^3)^T\right)^T$$

$$\frac{dL}{dB_{\hat{y}}} = \frac{dL}{d\hat{y}}\frac{d\hat{y}}{dB_{\hat{y}}} = \frac{dL}{d\hat{y}} \cdot 1$$

Now we'll go about the LSTM cell's back propagation. But first let's get a visual of the path we must take within a cell.



Full back propagation (red arrows) inside an LSTM cell. Figure by author.

As you can see, the path is quite complicated, which makes for computationally heavier operations than the RNN.

Bellow you can see the back propagation of both outputs of an LSTM cell, the cell state and the hidden state. You can refer to the equations I showed above for the forward pass to get a better understanding of which equations we are going through.

$$\frac{dL}{dh^3} = \frac{dL}{d\hat{y}}\frac{d\hat{y}}{dh^3} = \left(\frac{dL}{d\hat{y}} \cdot (W_{\hat{y}})^T\right)^T$$

$$\frac{dL}{dc^3}=\frac{dL}{dh^3}\frac{dh^3}{dc^3}=\frac{dL}{dh^3}\times o^3\times \tanh'(c^3)$$

You can see that the information that travelled forward through the cell state is now going backwards modulated by the tanh'. Note that the prime (') in σ' and tanh' represent the first derivative of both these functions.

In the next steps, we are going back to the parameters in each gate.

Output gate:

$$\frac{dL}{do^3}=\frac{dL}{dh^3}\frac{dh^3}{do^3}=\frac{dL}{dh^3}\times \tanh(c^3)$$

$$\frac{dL}{dW_o}=\frac{dL}{do^3}\frac{do^3}{dW_o}=\left(\frac{dL}{do^3}\times\sigma'(o^3)\cdot(X^3)^T\right)^T$$

$$\frac{dL}{dB_o}=\frac{dL}{do^3}\frac{do^3}{dB_o}=\frac{dL}{do^3}\times\sigma'(o^3)$$

New candidate gate:

$$\frac{dL}{dn^3}=\frac{dL}{dc^3}\frac{dc^3}{dn^3}=\frac{dL}{dc^3}\times i^3$$

$$\frac{dL}{dW_n}=\frac{dL}{dc^3}\frac{dc^3}{dW_n}=\left(\frac{dL}{dc^3}\times\tanh'(n^3)\cdot(X^3)^T\right)^T$$

$$\frac{dL}{dB_n}=\frac{dL}{dc^3}\frac{dc^3}{dB_n}=\frac{dL}{dc^3}\times\tanh'(n^3)$$

Input gate:

$$\frac{dL}{di^3}=\frac{dL}{dc^3}\frac{dc^3}{di^3}=\frac{dL}{dc^3}\times n^2$$

$$dL \qquad dL \quad dc^3 \qquad (\; dL \qquad \qquad (\;^3) \cdot (\;^3)^T)^T$$

$$\frac{}{dW_i}=\frac{}{dc^3}\frac{}{dW_i}=(\frac{}{dc^3}\times\tanh'(i^3)\cdot(X^3)^T)$$

$$\frac{dL}{dB_i}=\frac{dL}{dc^3}\frac{dc^3}{dB_i}=\frac{dL}{dc^3}\times\tanh'(i^3)$$

Forget gate:

$$\frac{dL}{df^3}=\frac{dL}{dc^3}\frac{dc^3}{df^3}=\frac{dL}{dc^3}\times c^2$$

$$\frac{dL}{dW_f}=\frac{dL}{dc^3}\frac{dc^3}{dW_f}=(\frac{dL}{dc^3}\times\tanh'(f^3)\cdot(X^3)^T)^T$$

$$\frac{dL}{dB_f}=\frac{dL}{dc^3}\frac{dc^3}{dB_f}=\frac{dL}{dc^3}\times\tanh'(f^3)$$

We have calculated the gradient for all the parameters inside the cell. However, we need to keep back propagating until the last cell. Let's see the last steps:

$$\frac{dL}{dc^2}=\frac{dL}{dc^3}\frac{dc^3}{dc^2}=\frac{dL}{dc^3}\times f^3$$

$$\frac{dL}{dX^3}=\frac{dL}{do^3}\frac{do^3}{dX^3}+\frac{dL}{dn^3}\frac{dn^3}{dX^3}+\frac{dL}{di^3}\frac{di^3}{dX^3}+\frac{dL}{df^3}\frac{df^3}{dX^3}$$

$$\frac{dL}{dX^3}=((\frac{dL}{do^3})^T W_o^T)^T+((\frac{dL}{do^3})^T W_n^T)^T+((\frac{dL}{do^3})^T W_i^T)^T+((\frac{dL}{do^3})^T W_f^T)^T$$

You may see that the information that travels from cell state c^3 to c^2 largelly depends on the outputs of the output gate and the forget gate. At the same time, the output and forget gradients depend on the information that was previously stored in the cell states. These interactions should provide the constant error back propagation.

Going further back into the global input (X^3), we add what is coming from all four gates together.

$$\left[\frac{dL}{}\right]$$

$$\frac{dL}{dX^3} = \begin{bmatrix} dx^3 \\ \frac{dL}{dh_1^2} \\ \frac{dL}{dh_2^2} \end{bmatrix} \rightarrow \begin{matrix} \frac{dL}{dx^3} = \frac{dL}{dX^3}[1] \\ \frac{dL}{dh^2} = \frac{dL}{dX^3}[2,3] \end{matrix}$$

$$\frac{dL}{dW_x} = \frac{dL}{dW_x^1} + \frac{dL}{dW_x^2} + \frac{dL}{dW_x^3}$$

$$\frac{dL}{dB_x} = \frac{dL}{dB_x^1} + \frac{dL}{dB_x^2} + \frac{dL}{dB_x^3}$$

Finally we deconcatenate the hidden state from the global input vector, go through the remaining cells and add all the gradients with respect to the parameters from all cells together.

Closing thoughts

This story's goal was to understand why the LSTM is capable of dealing with more complex problems than the RNN by keeping a constant flow of error throughout the backpropagation from cell to cell.

We explored the resulting issues from the poor handling of complex sequences from the RNN giving raise to the exploding and vanishing gradients.

Then we saw how these issues come to happen by exploring the flow of gradients in the RNN.

Finally we introduced the LSTM, its forward pass and, by deconstructing its backward pass, we understood that the cell state is influenced by two gate units that are responsible for ensuring a constant back flow of the error.

It is important to mention that as more experiments were performed with the LSTM there is a certain degree of complexity where this network stops being able to learn. Generally it goes to the thousand time steps before it happens which is already pretty good.

This is leading to a gradual phase out of the LSTM as problems become more ambitious in favour of a newer network called the Transformer or BERT. You may have also heard of the GTP-3 for Natural Language Processing. These are very powerful networks with a great potential.

However, the LSTM sure had its impact and was created with ingenuity and still is usefull today.

Thanks for reading!

References

[1] Sepp Hochreiter and Jürgen Schmidhuber, "Long Short-Term Memory" in Neural Computation, 1997, DOI: 10.1162/neco.1997.9.8.1735.

Long Short Term Memory Lstm Rnn Backpropagation Hands On Tutorials