How LSTM networks solve the problem of vanishing gradients

A simple, straightforward mathematical explanation

Nir Arbel Follow
Dec 21, 2018 · 10 min read

Greenport New York

When diving into the theory behind Recurrent Neural Networks (RNNs) and Long Short-Term Memory (LSTM) networks, 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?

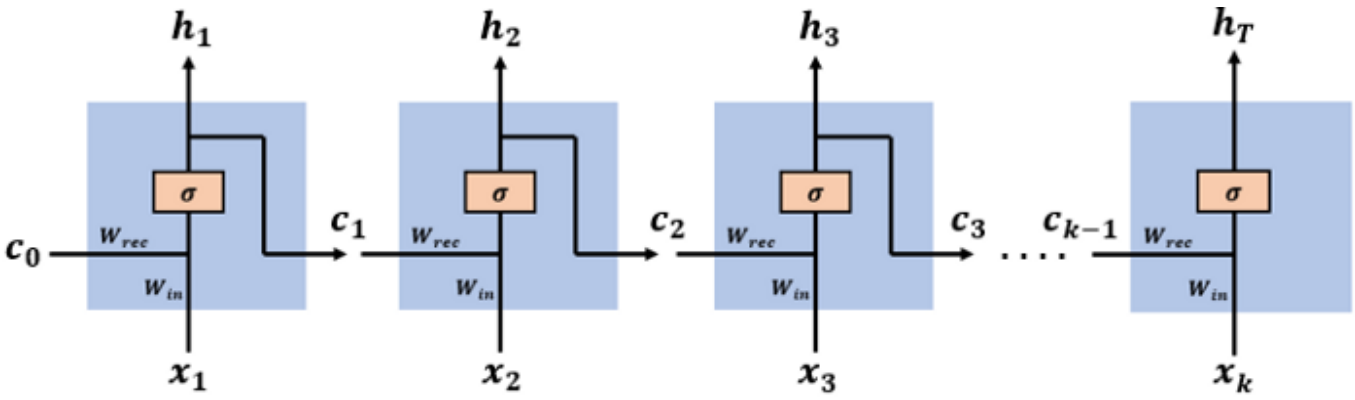
When I tried answering these questions, I searched for a mathematical explanation to get a better understanding of how these networks work. I had a hard time finding proofs that were understandable and clear enough for me. After reading the recommended papers and known blogs dealing with these questions I wrote an explanation that worked for me and made me feel I better understand the problem and solution.

RNNs and vanishing gradients

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. The gradients carry information used in the RNN parameter update and when the gradient becomes smaller and smaller, the parameter updates become insignificant which means no real learning is done.

Let's have a short reminder of how RNNs look like. We will work with a simple single hidden layer RNN with a single output sequence. The network looks like this:



The network has an input sequence of vectors [x(1), x(2),…, 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),…, 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: Wrec and Win 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 = [Wrec, Win].

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 we'll use this simpler model).

Backpropagation through time (BPTT) in RNNs

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 of the error term in an RNN

The gradient is used to update the model parameters by:

$$W \leftarrow W - \alpha \frac{\partial E}{\partial W}$$

And we continue the learning process using the Gradient Descent (GD) algorithm (we use the basic version of the GD in this work).

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

$$\frac{\partial E_k}{\partial W} = \frac{\partial E_k}{\partial h_k}\frac{\partial h_k}{\partial c_k} \cdots \frac{\partial c_2}{\partial c_1}\frac{\partial c_1}{\partial W}$$

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

Notice that since W=[Wrec, Win], c(t) can be written as:

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

Compute the derivative of c(t) and get:

$$\frac{\partial C_t}{\partial C_{t-1}} = \sigma'(W_{rec} \cdot c_{t-1} + W_{in} \cdot x_t) \cdot \frac{\partial}{\partial C_{t-1}}[W_{rec} \cdot c_{t-1} + W_{in} \cdot x_t]$$

$$= \sigma'(W_{rec} \cdot c_{t-1} + W_{in} \cdot x_t) \cdot W_{rec} \qquad (2)$$

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

$$\frac{\partial E_k}{\partial W} = \frac{\partial E_k}{\partial h_k}\frac{\partial h_k}{\partial C_k}\left(\prod_{t=2}^{k} \sigma'(W_{rec} \cdot c_{t-1} + W_{in} \cdot x_t) \cdot W_{rec}\right)\frac{\partial C_1}{\partial W}$$

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

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:

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

So for some time step k:

$$\frac{\partial E_k}{}$$

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

And our complete error gradient will vanish

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

The network's weights update will be:

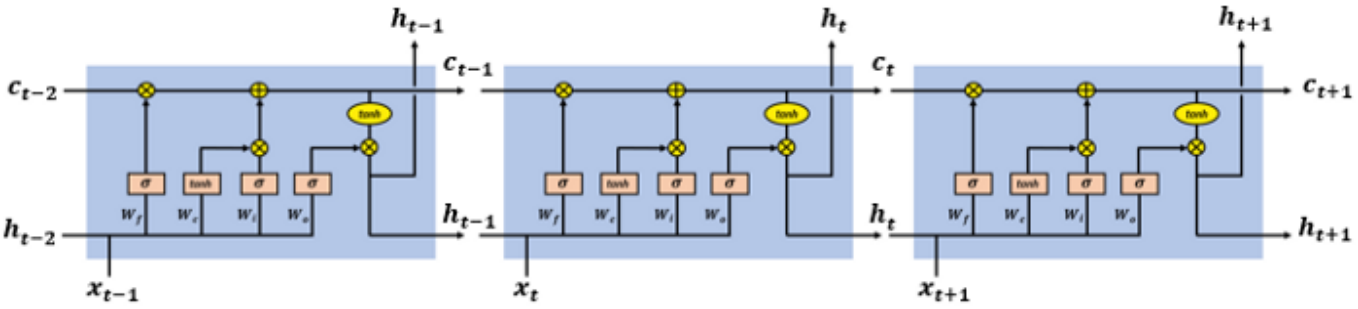
$$W \leftarrow W - \alpha \frac{\partial E}{\partial W} \approx W$$

And no significant learning will be done in reasonable time.

How LSTMs solve this?

I recommend reading Colah's blog for an in-depth review of LSTMs since we are only going to have a short reminder here.

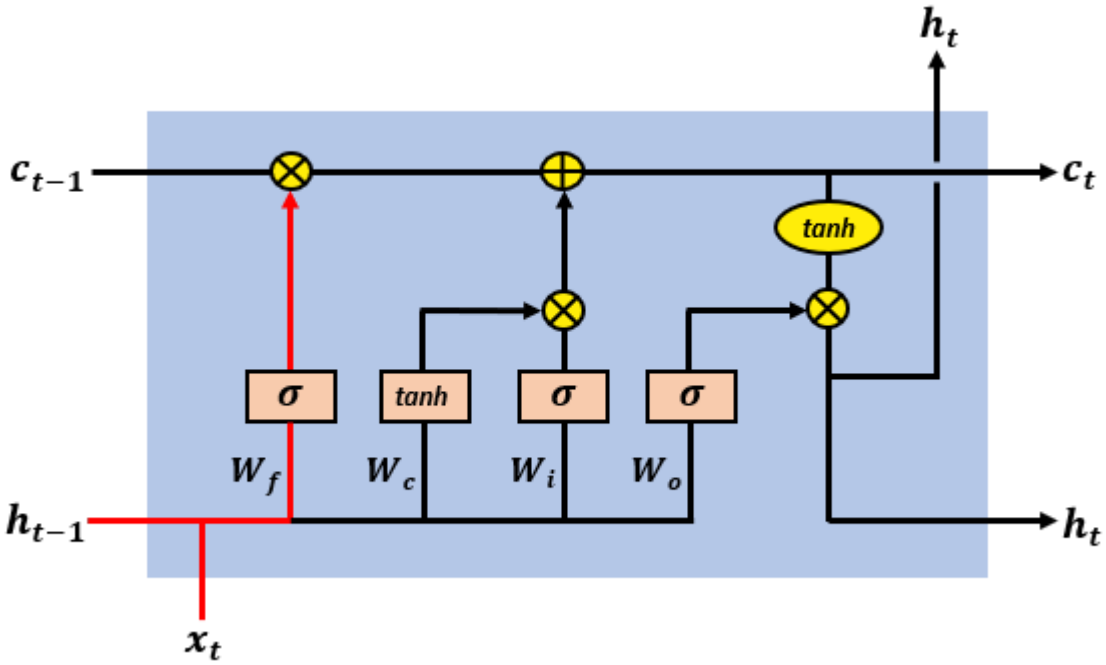
An LSTM network has an input vector [h(t-1),x(t)] at time step t. The network cell state is denoted by c(t). The output vectors passed through the network between consecutive time steps t, t+1 are denoted by h(t).



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.

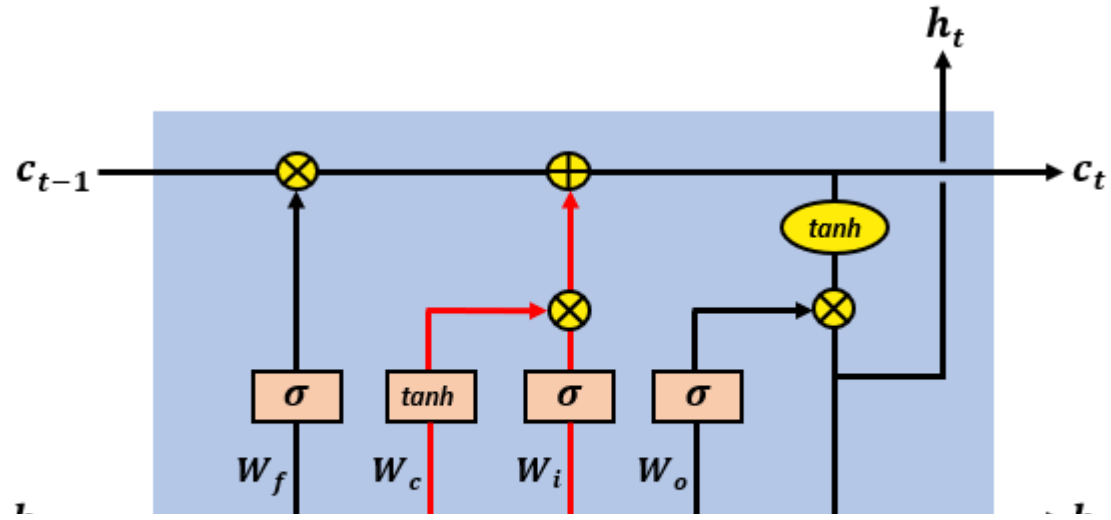


The LSTM forget gate update of the cell state

The forget gate's output is given by:

$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t]\right)$$

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



The LSTM input gate update of the cell state

The input gate's output has the form:

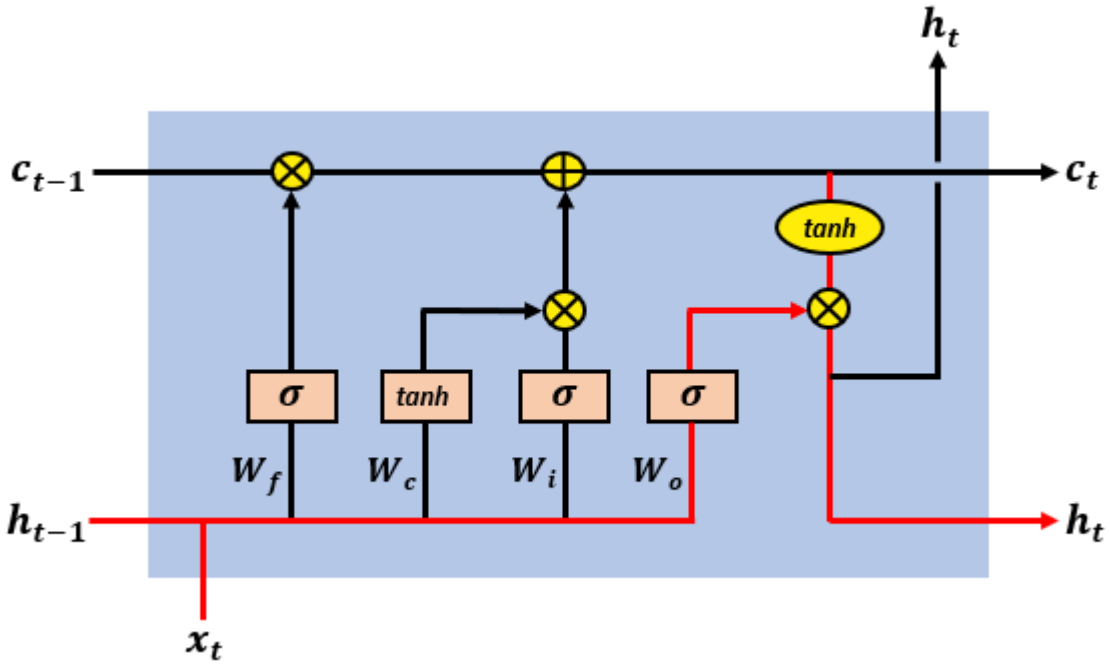
$$tanh(W_c \cdot [h_{t-1}, x_t]) \otimes \sigma(W_i \cdot [h_{t-1}, x_t])$$

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

$$\tilde{c}_t = tanh\ (W_c \cdot [h_{t-1}, x_t])$$

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t])$$

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).



The LSTM output gate's action on the cell state

The output gate's activations are given by:

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t])$$

and the cell's output vector is given by:

$$h_t = o_t \otimes tanh(c_t)$$

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:

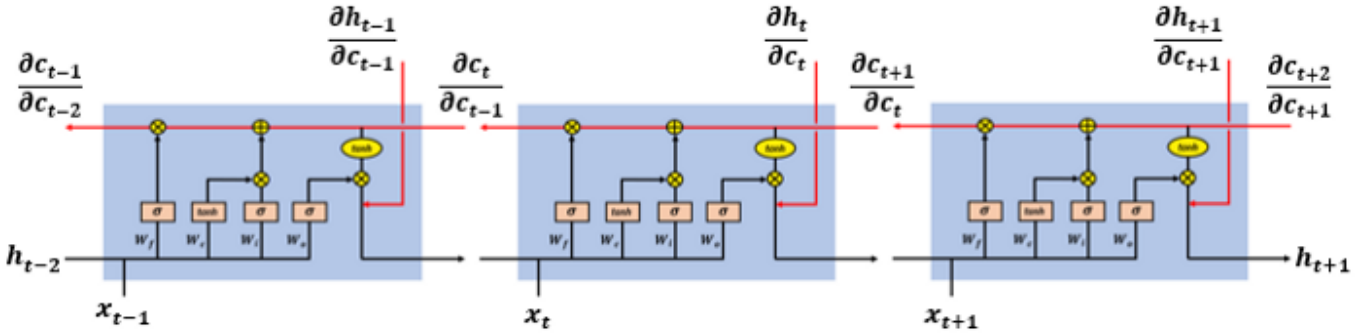
$$c_t = c_{t-1} \otimes f_t \oplus \tilde{c}_t \otimes i_t$$

Backpropagation through time in LSTMs

As in the RNN model, our LSTM network outputs a prediction vector h(k) on the k-th time step. The knowledge encoded in the state vectors c(t) captures long-term dependencies and relations in the sequential data.

The length of the data sequences can be hundreds and even thousands of time steps, making it extremely difficult to learn using a basic RNN.

We compute the gradient used to update the network parameters, the computation is done over T time steps.



Backpropagating through time for gradient computation

As in RNNs, the error term gradient is given by the following sum of T gradients:

$$\frac{\partial E}{\partial W} = \sum_{t=1}^{T} \frac{\partial E_t}{\partial W} \qquad (3)$$

The gradient of the error in an LSTM

For the complete error gradient to vanish, all of these T sub gradients need to vanish. If we think of (3) as a series of functions, then by definition, this series converges to zero if the sequence of its partial sums tends to zero, so

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

if the series of partial sums

$$(S_1, S_2, S_3, \ldots)$$

where

$$S_n = \sum_{t=1}^{n} \frac{\partial E_t}{\partial W}$$

tends to zero.

So if we want (3) not to vanish, our network needs to increase the likelihood that at least some of these sub gradients will not vanish, in other words, make the series of sub gradients in (3) not converge to zero.

The error gradients in an LSTM network

The gradient of the error for some time step k has the form:

$$\frac{\partial E_k}{\partial W} = \frac{\partial E_k}{\partial h_k}\frac{\partial h_k}{\partial c_k}\cdots\frac{\partial c_2}{\partial c_1}\frac{\partial c_1}{\partial W}$$

$$= \frac{\partial E_k}{\partial h_k}\frac{\partial h_k}{\partial c_k}\left(\prod_{t=2}^{k}\frac{\partial c_t}{\partial c_{t-1}}\right)\frac{\partial c_1}{\partial W} \qquad (4)$$

As we have seen, the following product causes the gradients to vanish:

$$\prod_{t=2}^{k}\frac{\partial c_t}{\partial c_{t-1}}$$

In an LTSM, the state vector c(t), has the form:

$$c_t = c_{t-1} \otimes \sigma(W_f \cdot [h_{t-1}, x_t]) \oplus$$

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

which can be written compactly as

$$c_t = c_{t-1} \otimes f_t \oplus \tilde{c}_t \otimes i_t \qquad (5)$$

Notice that the state vector c(t) is a function of the following elements, which should be taken into account when computing the derivative during backpropagation:

$$c_{t-1},\ f_t,\ \tilde{c}_t, i_t$$

Compute the derivative of (5) and get:

$$\frac{\partial c_t}{\partial c_{t-1}} = \frac{\partial}{\partial c_{t-1}}[c_{t-1} \otimes f_t \oplus \tilde{c}_t \otimes i_t]$$

$$= \frac{\partial}{\partial c_{t-1}}[c_{t-1} \otimes f_t] + \frac{\partial}{\partial c_{t-1}}[\tilde{c}_t \otimes i_t]$$

$$= \frac{\partial}{\partial c_{t-1}}[o_{t-1} \otimes f_t] \cdot \frac{\partial}{\partial c_{t-1}}[o_t \otimes i_t]$$

$$= \frac{\partial f_t}{\partial c_{t-1}} \cdot c_{t-1} + \frac{\partial c_{t-1}}{\partial c_{t-1}} \cdot f_t + \frac{\partial i_t}{\partial c_{t-1}} \cdot \tilde{c}_t + \frac{\partial \tilde{c}_t}{\partial c_{t-1}} \cdot i_t$$

We compute (detailed computations are given in the end of the article) the four derivative terms and write:

$$\frac{\partial c_t}{\partial c_{t-1}} = \sigma'\left(W_f \cdot [h_{t-1}, x_t]\right) \cdot W_f \cdot o_{t-1} \otimes tanh'(c_{t-1}) \cdot c_{t-1}$$

$$+ f_t$$

$$+ \sigma'(W_i \cdot [h_{t-1}, x_t]) \cdot W_i \cdot o_{t-1} \otimes tanh'(c_{t-1}) \cdot \tilde{c}_t$$

$$+ \sigma'(W_c \cdot [h_{t-1}, x_t]) \cdot W_c \cdot o_{t-1} \otimes tanh'(c_{t-1}) \cdot i_t$$

Denote the four elements comprising the derivative of the cell state by:

$$A_t = \sigma'\left(W_f \cdot [h_{t-1}, x_t]\right) \cdot W_f \cdot o_{t-1} \otimes tanh'(c_{t-1}) \cdot c_{t-1}$$

$$B_t = f_t$$

$$C_t = \sigma'(W_i \cdot [h_{t-1}, x_t]) \cdot W_i \cdot o_{t-1} \otimes tanh'(c_{t-1}) \cdot \tilde{c}_t$$

$$D_t = \sigma'(W_c \cdot [h_{t-1}, x_t]) \cdot W_c \cdot o_{t-1} \otimes tanh'(c_{t-1}) \cdot i_t$$

We write the additive gradient as:

$$\frac{\partial c_t}{\partial c_{t-1}} = A_t + B_t + C_t + D_t \qquad (6)$$

Plug (6) into (4) and get the LSTM states gradient:

$$\frac{\partial E_k}{\partial W} = \frac{\partial E_k}{\partial h_k}\frac{\partial h_k}{\partial c_k}\left(\prod_{t=2}^{k}[A_t + B_t + C_t + D_t]\right)\frac{\partial c_1}{\partial W}$$

Preventing the error gradients from vanishing

Notice that the gradient contains the forget gate's vector of activations, which allows the network to better control the gradients values, at each time step, using suitable parameter updates of the forget gate. The presence of the forget gate's activations allows the LSTM to decide, at each time step, that certain information should not be forgotten and to update the model's parameters accordingly.

Let's go over how this property helps us. Say that for some time step k<T, we have that:

$$\sum_{t=1}^{k}\frac{\partial E_t}{\partial W} \to 0$$

Then for the gradient not to vanish, we can find a suitable parameter update of the forget gate at time step k+1 such that:

$$\frac{\partial E_{k+1}}{\partial W} \not\to 0$$

It is the presence of the forget gate's vector of activations in the gradient term along with additive structure which allows the LSTM to find such a parameter update at any time step, and this yields:

$$\sum_{t=1}^{k+1}\frac{\partial E_t}{\partial W} \not\to 0$$

and the gradient doesn't vanish.

Another important property to notice is that the cell state gradient is an additive function made up from four elements denoted A(t), B(t), C(t), D(t). This additive property enables better balancing of gradient values during backpropagation. The LSTM

updates and balances the values of the four components making it more likely the additive expression does not vanish.

For example, say that for every t in {2,3,…,k} we take the following four neighbourhoods of values as a balancing combination in our gradient:

$$A_t \approx \overrightarrow{0.1},\ B_t = f_t \approx \overrightarrow{0.7},\ \mathrm{C_t} \approx \overrightarrow{0.1},\ \mathrm{D_t} \approx \overrightarrow{0.1}$$

which yields:

$$\prod_{t=2}^{k}[A_t + B_t + C_t + D_t] \approx \prod_{t=2}^{k}\left[\overrightarrow{0.1} + \overrightarrow{0.7} + \overrightarrow{0.1} + \overrightarrow{0.1}\right]$$

$$\approx \prod_{t=2}^{k} \vec{1} \nrightarrow 0$$

and the product does not vanish.

This additive property is different from the RNN case where the gradient contained a single element inside the product. **In RNNs, the sum in (3) is made from expressions with a similar behaviour that are likely to all be in [0,1] which causes vanishing gradients.**

In LSTMs, however, the presence of the forget gate, along with the additive property of the cell state gradients, enables the network to update the parameter in such a way that the different sub gradients in (3) do not necessarily agree and behave in a similar manner, making it less likely that all of the T gradients in (3) will vanish, or in other words, the series of functions does not converge to zero:

$$\sum_{t=1}^{T} \frac{\partial E_t}{\partial W} \nrightarrow 0$$

and our gradients do not vanish.

As mentioned briefly, the RNN gradients can also explode if the sum in (3) is made up from expressions with a similar behaviour that are all significantly greater than 1.

Summing up, we have seen that RNNs suffer from vanishing gradients and caused by long series of multiplications of small values, diminishing the gradients and causing the learning process to become degenerate. In a analogues way, RNNs suffer from exploding gradients affected from large gradient values and hampering the learning process.

LSTMs solve the problem using a unique additive gradient structure that includes direct access to the forget gate's activations, enabling the network to encourage desired behaviour from the error gradient using frequent gates update on every time step of the learning process.

References

[1] LONG SHORT TERM MEMORY, Sepp Hochreiter, Jurgen Schmidhuber

[2] On the difficulty of training recurrent neural networks, Razvan Pascanu, Tomas Mikolov, Yoshua Bengio

[3] Long Short-Term Memory in Recurrent Neural Networks, Felix Gers

[4] Cristopher Olah's Blog

[5] Noah Webber's blog (weberna's blog)

Additional computations needed for the cell state gradient computation:

$$\frac{\partial f_t}{\partial c_{t-1}} \cdot c_{t-1} = \frac{\partial}{\partial c_{t-1}} \left[\sigma\left(W_f \cdot [h_{t-1}, x_t]\right)\right] \cdot c_{t-1}$$

$$= \sigma'\left(W_f \cdot [h_{t-1}, x_t]\right) \cdot W_f \cdot \frac{\partial h_t}{\partial c_{t-1}} \cdot c_{t-1}$$

$$= \sigma'\left(W_f \cdot [h_{t-1}, x_t]\right) \cdot W_f \cdot o_{t-1} \otimes tanh'(c_{t-1}) \cdot c_{t-1}$$

$$\frac{\partial i_t}{\partial c_{t-1}} \cdot \tilde{c}_t = \frac{\partial}{\partial c_{t-1}} \left[\sigma(W_i \cdot [h_{t-1}, x_t])\right] \cdot \tilde{c}_t$$

$$= \sigma'(W_c \cdot [h_{t-1}, x_t]) \cdot W_c \cdot \frac{\partial H_t}{\partial c_{t-1}} \cdot \tilde{c}_t$$

$$= \sigma'(W_c \cdot [h_{t-1}, x_t]) \cdot W_c \cdot o_{t-1} \otimes tanh'(c_{t-1}) \cdot \tilde{c}_t$$

$$\frac{\partial \tilde{c}_t}{\partial c_{t-1}} \cdot i_t = \frac{\partial}{\partial c_{t-1}} [\sigma(W_c \cdot [h_{t-1}, x_t])] \cdot i_t =$$

$$= \sigma'(W_c \cdot [h_{t-1}, x_t]) \cdot W_c \cdot \frac{\partial H_t}{\partial c_{t-1}} \cdot i_t =$$

$$= \sigma'(W_c \cdot [h_{t-1}, x_t]) \cdot W_c \cdot o_{t-1} \otimes tanh'(c_{t-1}) \cdot i_t$$