

A 'layer' in an LSTM consists of N cells, and these cells operate on every timestep of the input sequence. Each cell processes the entire sequence, one timestep at a time. So, while a layer has N cells, these cells are used at every timestep, not just a single timestep.

An LSTM can have multiple layers. these layers function together much like a normal neural net. Each layer consists of a number of cells (units). The output of one layer at each timestep serves as the input to the next layer at the same timestep.

All the cells in one layer operate simultaneously at each timestep, processing the input from the previous layer (or the initial input for the first layer) and producing an output that becomes the input for the next layer. This continues for each layer in the network.And then we continue on to the next timestep/.

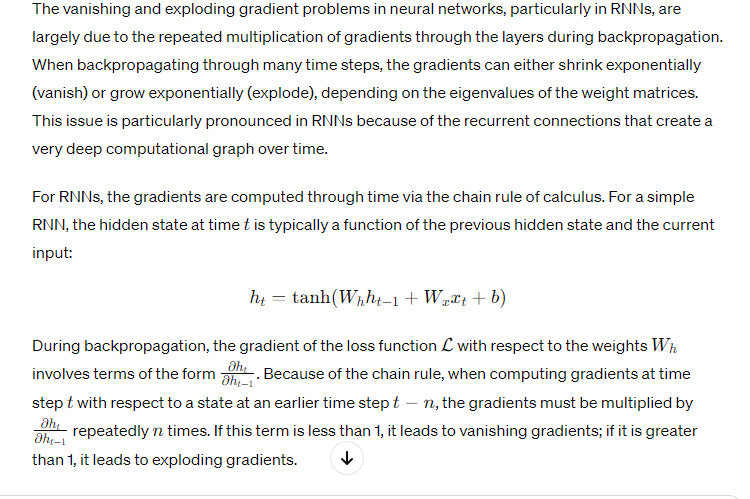
In a single LSTM layer, each cell operates independently in terms of processing the input and its previous hidden state; there is no direct passing of information between the cells within the same layer at the same timestep. Each cell contributes to a component of the output hidden state vector of that layer, but they do not communicate directly with each other.

LSTMs, like other types of RNNs, use the same set of weights across all timesteps. This is known as weight sharing and it's a key feature of recurrent neural networks. It means that the same weights are applied to the input and hidden state at every timestep, which greatly reduces the number of parameters compared to having unique weights for each timestep.

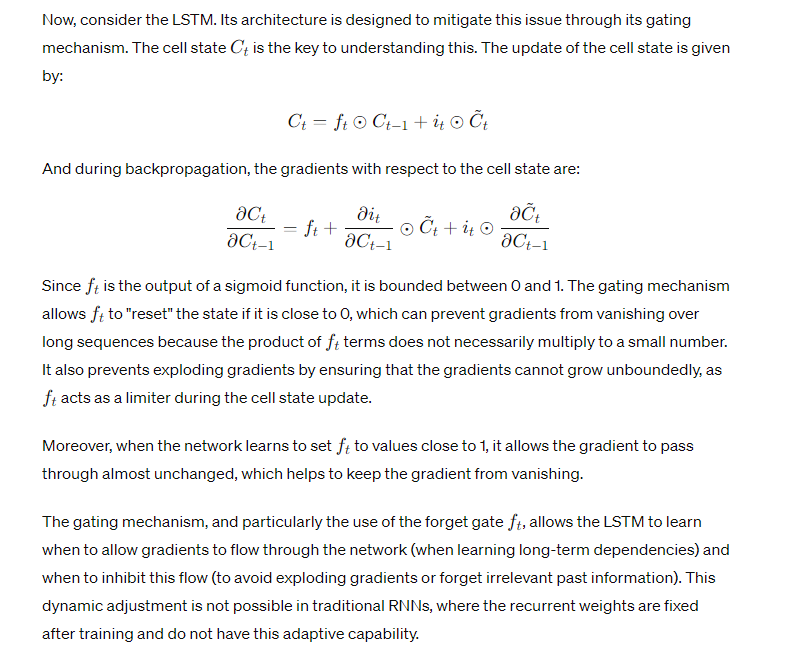
okay. now, I understand how each of these gates in theory does what it claims to do. what I don't understand is how the network actually learns to use these gates effectively. I know that the functions are differentiable and hence gradients flow through them. M

Each of these functions is differentiable and thus gradients flow through them. If gradients flow through them, then the weights will be updated in such a way that the loss will decrease as a result of the weight updates being applied. It doesn’t seem possible that this network could learn to use the gates. But it can.

The network will learn to set 𝑓𝑡*ft*​ values close to 1 for information that is important to keep through time. During backpropagation, if the gradient indicates that retaining certain information would lead to a lower loss, the weights and biases controlling 𝑓𝑡*ft*​ will be updated to produce a value closer to 1. The loss function, through backpropagation, implicitly guides the network to learn these long-term dependencies, as keeping this information will contribute to making better predictions and thus minimizing the loss.



Read this back to front



By having a pathway through which the gradient can flow that involves only element-wise multiplications (no matrix multiplications), and by controlling the flow with gates that the network learns to open and close, LSTMs provide a way to maintain more stable gradients over time. This results in the mitigation of the vanishing and exploding gradient problems that are common in standard RNNs.

I understand how all the computations occur in an lstm cell and how the loss will teach the thing to operate the gates in such a way that minimizes loss

(a meta way of thinking about this is that the best way to operate the gates is going to be the way that prevents vanishing/exploding gradients from messing up the performance of the network. Retaining useful information, discarding useless information, and making sure the gradients stay within reasonable bounds (i.e near 1) for info that needs to be propagated forward. Therefore, if you believe it is possible to use these gates to prevent vanishing/exploding gradients and believe that doing so would result in a network with better performance (lower loss) then you know how the gates will operate in such a way that minimizes loss

Q: I see how the gradients will be kept in check for C\_t, but what about h\_t?

