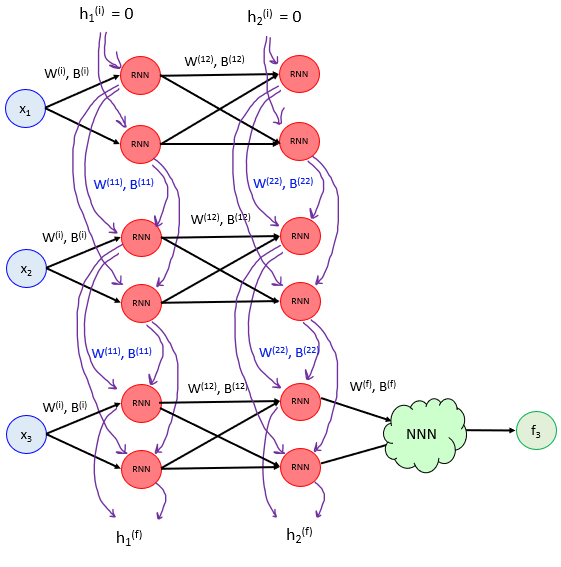
**LSTM/GRU Neural Networks**

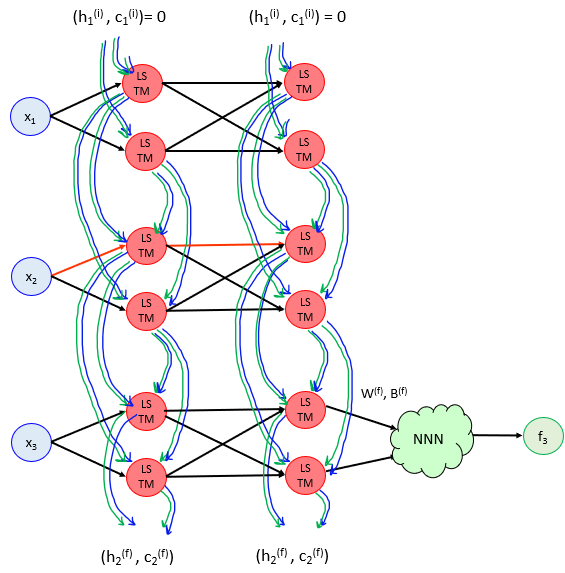
So we saw that RNN have the exploding/vanishing gradient problem. All the recursive connections W(ab), such as W(11), W(22), below, will cause the gradient to explode/vanish if they are greater/less than 1.



Stated another way, the information conveyed by those aformentioned connections either grows or diminishes (literally exponentially in this case) with each time-step (each successive input). In the case of a vanishing gradient (seemingly the most common scenario), this will mean that given a last term xn, the preceding terms xn-1, xn-2 will have much larger importance than, say, terms xn-6, xn-7, etc. This can be a problem in general. Specifically it’s a problem for language models where context is important. For instance, the correct conjugation of a verb would depend on a subject that was introduced a few words ago. And it can be a problem in general time-series learning problems where effects are dependent on causes far in the past (or, just not recent).

**LSTM Networks**

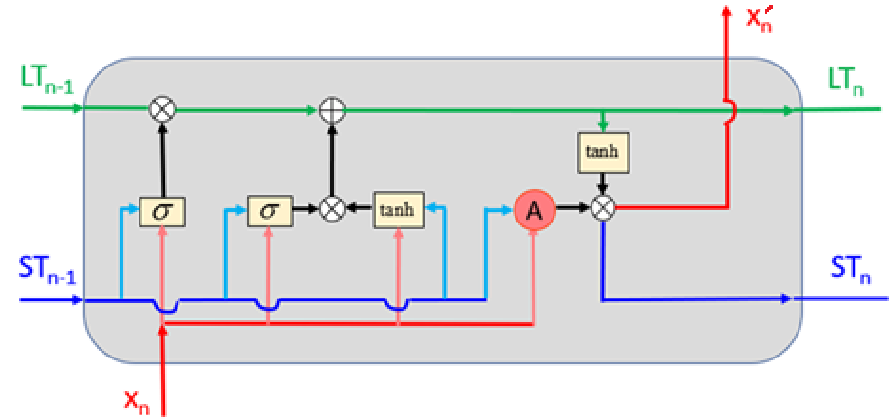
LSTM’s are designed to surmount this problem. (I think) they replace the single recursive connections and activation nodes, with two connections.



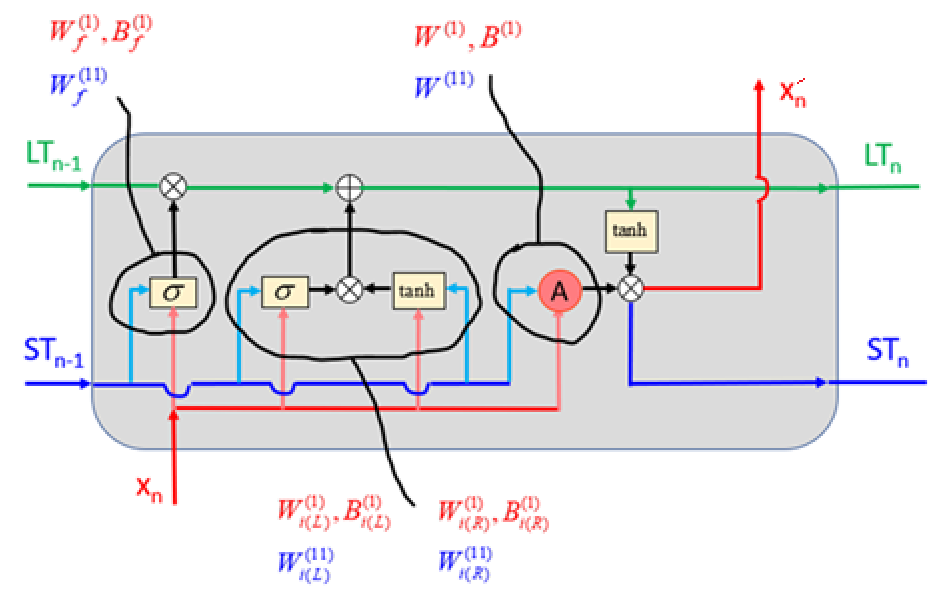
These are a **short term** and ***long term*** memory connection. These are also called the **hidden state** and ***cell state***. The short term/hidden state connection is kind of the same thing as the previous recursive connection discussed in the RNN network file. The long term/cell state connection is new. Still calling the initial sideways input and output h, though. Should emphasize that only the *recursively* connected activation nodes are replaced with LSTM’s. Want to emphasize that all lines coming out of the A’s carry the same values initially. These of course will be multiplied by the weight in the line. It's just easier to draw them coming out from different places on the A node, rather than the same place. Also, all lines terminating on an A will of course be added together (and a bias added).

**Inside the LSTM node**

Let’s focus on one of the middle activation nodes. Gonna draw the inner workings of the node, called an LSTM, below. I tried to match the color scheme, so look at the LSTM node above with red leads coming in and out.



So at roughest glance, we can see the xn input and x´n outputs to the (LSTM) node. These are as before. But the single recursive connection is replaced by two recursive connections, the Long Term (LT) and Short Term (ST) connections. Let’s talk about what’s going on, below:



So we have values flowing down short term memory (ST) and long term memory (LT) lines. The ST line is basically what our recursive connection was in the basic RNN’s context. The LT memory line is an addition, which allows previous cells to exert more influence on the given cell’s outcome (x´n), and ultimately output, f. The information line (xn, x´n) is just as it was before in the RNN context.

The rightmost black circle encloses what is basically the old RNN node: we have the xn and

STn-1 lines joining up at the activation function with the old bias and weights from before. Let’s talk about the input a little more. So xn is multiplied by a weight, W(1). And the short term memory guy STn-1 is a linear combination of the other two short term memory terms coming into the node (see the network up above). I’m vaguely denoting this incoming combination as W(11)STn-1. A bias is added to these terms so that the total input into the (A) is W(1)xn + W(11)STn-1 + B(1). The activation function does its job and so A(W(1)xn + W(11)STn-1 + B(1)) gets output from the activation function, and multiplied by tanh(LTn). This is where the LT line has its say on/influences the x´n output, and STn output. So the final output along these lines is:



Note these are the same, as was the case with the RNN too. If LTn is large, then tanh(LTn) ~ 1, and so our outputs would be as before, in the usual RNN. So we can see that only if LTn is *not* large will we get something different than the usual RNN output x´n and STn.

The other two black circles enclose where the ST and x lines have their say on what is carried/output by the LT line. They express their say by multiplying the incoming LTn-1 by a weight and a bias. So the first/leftmost circle encloses the so-called ***forget gate***. It multiplies the incoming LTn-1 memory by a *weight* given by σ(Wf(1)xn + Wf(11)STn-1 + Bf(1)), where σ is the eponymous sigmoid activation function. If the weight is 0, then this would wipe out the influence/memory of previous cells. One might want to do this in a language model if we’re changing subjects, starting a new sentence, or something. On the other hand, we’d want to keep the memory if it contained information of sorts about the subject of the sentence and we are about to conjugate the verb in the sentence. The second/middle circle encloses the so-called *input gate*. It adds a *bias* to LTn-1 given by σ(Wi(L)(1)·xn + Wi(L)(11)·STn-1 + Bi(L)(1)) · tanh(Wi(R)(1)·xn + Wi(R)(11)·STn-1 + Bi(R)(1)). So the net LT output is:



The input gate is where the cell can add its own information to the long term memory, for other cells down the line to use. So in summary, the x and ST lines function basically as before – each has the same output as the other, just like they did in the previous RNN thing. The LT is new, and it has a *different* output than x and ST (in the previous RNN stuff by contrast, *all* outputs from a node were the same – *before* they were multplied by weights and things further down their lines). And as the incoming x and ST lines input into A, they give a weight and bias to LT, which then multiplies the x and ST outputs of A by tanh(LT). Be aware that the weights in the LTSM node (all 12 of them) will be the same for all corresponding LSTM cells in the unrolled network, just as the single weight W(11) was before.

I think for most LSTM’s, the activation function A, is a σ (sigmoid). In Keras, all the σ’s (including the A) are actually called the ‘*recurrent activation*’ functions, and the tanh’s are just ‘*activation*’ functions. These can be changed. Switching out the *activation* function default tanh for relu, or some custom function is common for analyzing time series.

**GRU Networks**

GRU networks are designed to address the same issue LSTM’s do, but with a fairly different architecture. For one, it eliminates the separate long term/short term memory lines, but combines them into a single line, bringing us back to the single line that we had in regular RNN’s.

A computer screen shot of a computer

Description automatically generated

Well I don’t care. But suffice to say that the output yn (which I called x´n, above), and the hidden state output hn are both the same, just as with an RNN. Of course this is different than what we had for the LSTM. The general circuit would look something like this, for example:

