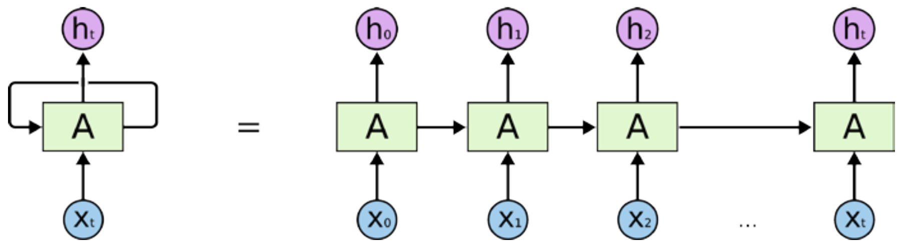
Introduction to LSTMs

So far, you have learnt to make asset price predictions using:

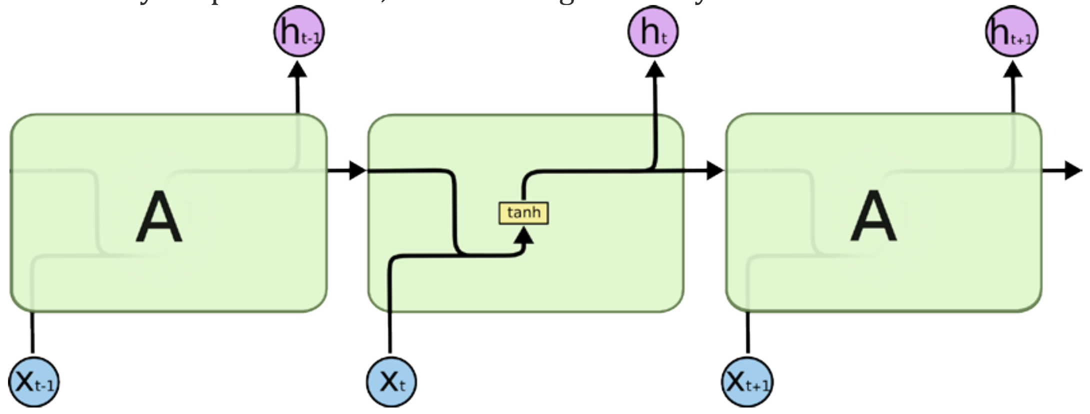
- Simple Moving Average/Exponential Moving Average
- Kalman Filter
- OLS Regression

Now, Long Short-Term Memory Models.

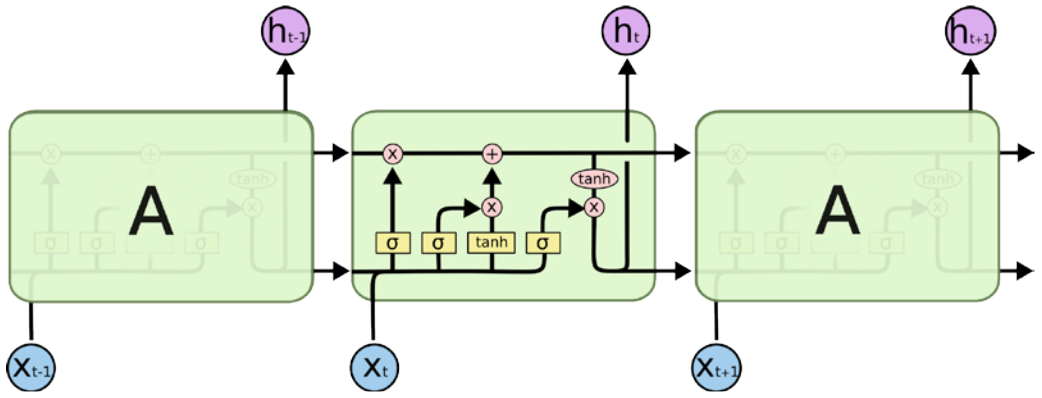
Recurrent Neural Networks



Long Short Term Memory networks – usually just called "LSTMs" – are a special kind of RNN, capable of learning long-term dependencies. All recurrent neural networks have the form of a chain of repeating modules of neural network. In standard RNNs, this repeating module will have a very simple structure, such as a single tanh layer.

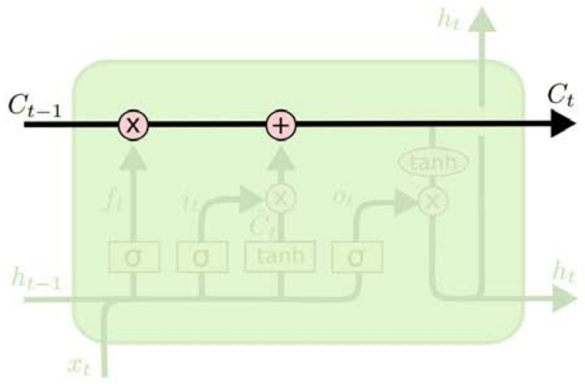


LSTMs also have this chain like structure, but the repeating module has a different structure. Instead of having a single neural network layer, there are four, interacting in a very special way.



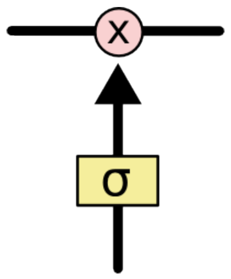
The Core Idea Behind LSTMs

The key to LSTMs is the cell state, the horizontal line running through the top of the diagram. The cell state is kind of like a conveyor belt. It runs straight down the entire chain, with only some minor linear interactions. It's very easy for information to just flow along it unchanged.



The LSTM does have the ability to remove or add information to the cell state, carefully regulated by structures called gates.

Gates are a way to optionally let information through. They are composed out of a sigmoid neural net layer and a point-wise multiplication operation.

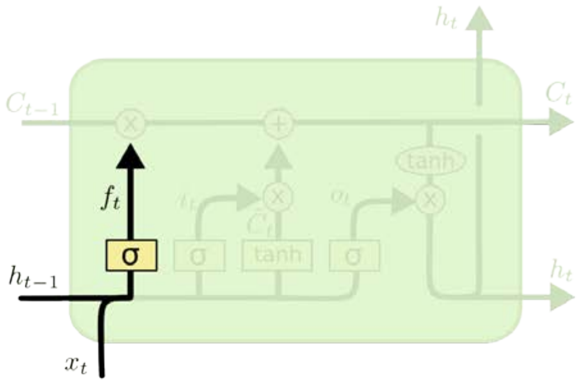


The sigmoid layer outputs numbers between zero and one, describing how much of each component should be let through. A value of zero means "let nothing through," while a value of one means "let everything through!"

An LSTM has three of these gates, to protect and control the cell state.

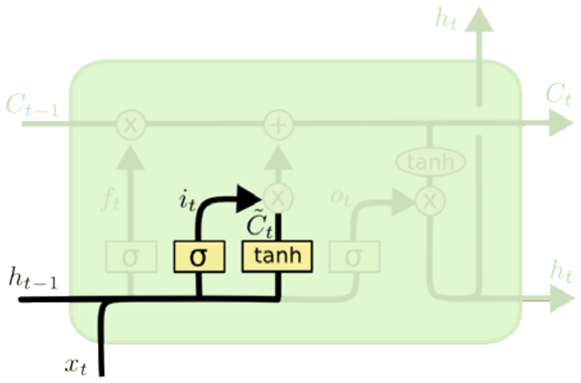
Step-by-Step LSTM Walk Through

The first step in our LSTM is to decide what information we're going to throw away from the cell state. This decision is made by a sigmoid layer called the "forget gate layer." It looks at h_{t-1} and x_t, and outputs a number between 0 and 1 for each number in the cell state C_{t-1}. A 1 represents "completely keep this" while a 0 represents "completely get rid of this."



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

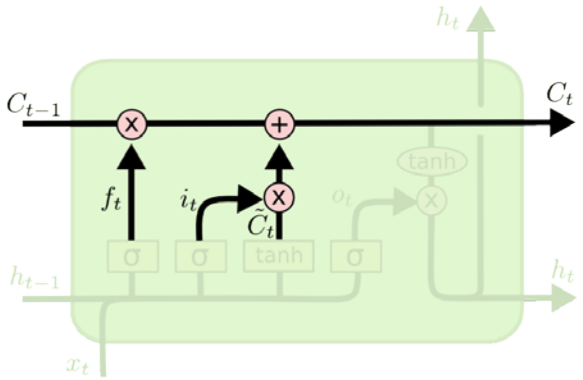
The next step is to decide what new information we're going to store in the cell state. This has two parts. First, a sigmoid layer called the "input gate layer" decides which values we'll update. Next, a tanh layer creates a vector of new candidate values, $\tilde{C}_t$, that could be added to the state. In the next step, we'll combine these two to create an update to the state.



$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)$$

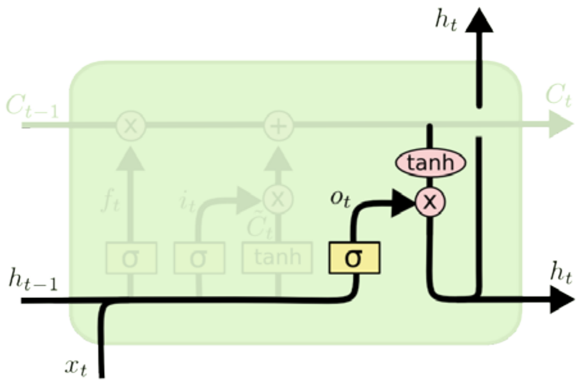
It’s now time to update the old cell state, C_{t-1}, into the new cell state C_t. The previous steps already decided what to do, we just need to actually do it.

We multiply the old state by f_t, forgetting the things we decided to forget earlier. Then we add , $i \cdot \widetilde{C}_t$. This is the new candidate values, scaled by how much we decided to update each state value.



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Finally, we need to decide what we’re going to output. This output will be based on our cell state, but will be a filtered version. First, we run a sigmoid layer which decides what parts of the cell state we’re going to output. Then, we put the cell state through tanhtanh (to push the values to be between −1−1 and 11) and multiply it by the output of the sigmoid gate, so that we only output the parts we decided to.



$$o_t = \sigma\left(W_o\ [h_{t-1}, x_t]\ +\ b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

Notes on Tanh and Sigmoid Function

The logistic sigmoid function, a.k.a. the inverse logit function, is

$$g(x) = \frac{e^x}{1 + e^x}$$

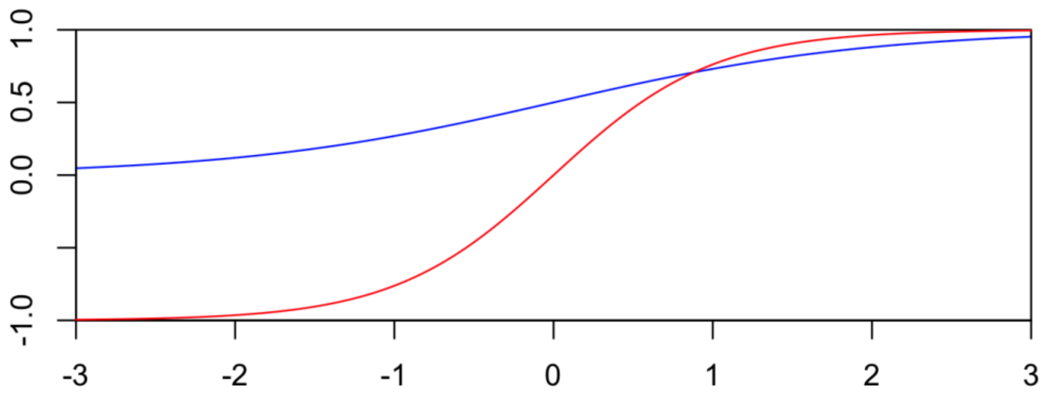
Its outputs range from 0 to 1, and are often interpreted as probabilities (in, say, logistic regression).

The tanh function, a.k.a. hyperbolic tangent function, is a rescaling of the logistic sigmoid, such that its outputs range from -1 to 1. (There's horizontal stretching as well.)

$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

The (-1,+1) output range tends to be more convenient for neural networks, so tanh functions show up there a lot.

The two functions are plotted below. Blue is the logistic function, and red is tanh.


1.0
0.5
0.0
-1.0
-3
-2
-1
0
1
2
3