Long-Short Term Memory

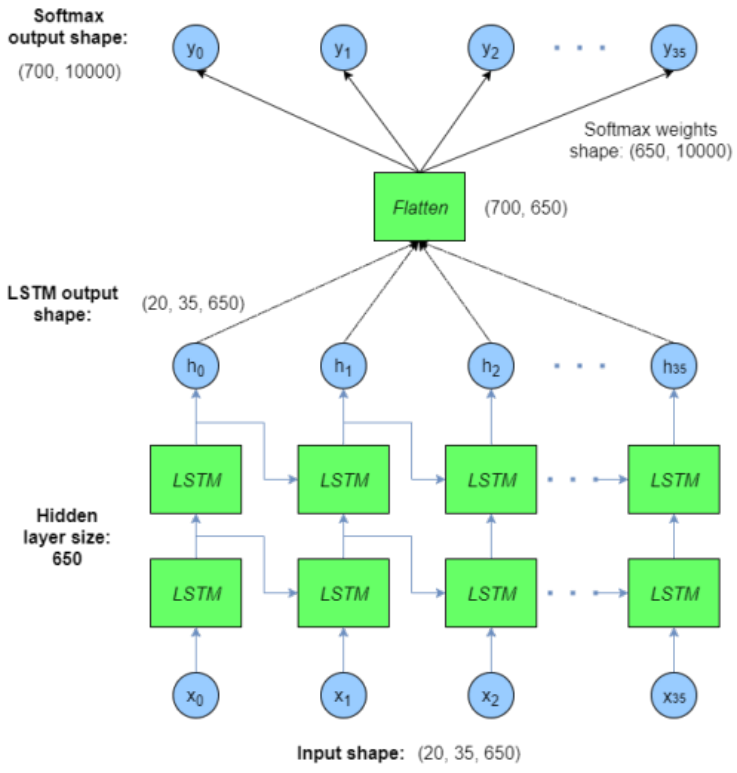
Long Short-Term Memory (LSTM) networks are an extension for recurrent neural networks. The units of an LSTM are used as building units for the layers of a RNN which is then often called an LSTM network.

LSTM ’s and GRU’s were created as the solution to short-term memory. They have internal mechanisms called gates that can regulate the flow of information.

LSTM basically extend the memory and enable RNN’s to remember their inputs over a long period of time. Because of that they are capable of learning long-term dependencies. Therefore, it is well suited to learn from important experiences that have very long time lags in between.

Regular RNNs just the hidden state and no cell state. It turns out that RNNs have difficulty of accessing information from a long time ago. LSTMs have two different states passed between the cells which are the cell state and the hidden state which carry the long and short-term memory respectively and thus face no problem in accessing information from a long time ago.

LSTM’s contain their information in a memory that is much like the memory of a computer and they can read, write and delete information from its memory. This memory can be seen as a gated cell where gated means that the cell decides whether or not to store or delete information (e.g. if it opens the gates or not) based on the importance it assigns to the information. The assigning of importance happens through weights, which are also learned by the algorithm. This simply means that it learns over time which information is important and which not. There are three gates - input, forget and output gate. These gates determine whether or not to let new input in (input gate), delete the information because it isn’t important (forget gate) or to let it impact the output at the current time step (output gate). The gates in a LSTM are analog, in the form of sigmoid, meaning that they range from 0 to 1. The fact that they are analog, enables them to do backpropagation with it.


Softmax output shape:
(700, 10000)
y0
y1
y2
y35
Softmax weights shape: (650, 10000)
Flatten
(700, 650)
LSTM output shape:
(20, 35, 650)
h0
h1
h2
h35
LSTM
Hidden layer size: 650
x0
x1
x2
x35
Input shape: (20, 35, 650)

LSTM TYPES

- Vanilla LSTM

A Vanilla LSTM is an LSTM model that has a single hidden layer of LSTM units and an output layer used to make a prediction.

- Stacked LSTM

Multiple hidden LSTM layers can be stacked one on top of another in what is referred to as a Stacked LSTM model.

- Bidirectional LSTM

On some sequence prediction problems, it can be beneficial to allow the LSTM model to learn the input sequence both forward and backwards and concatenate both interpretations. So in bidirectional LSTM's input sequences are presented and learned both forward and backward

- CNN LSTM

A CNN LSTM is a type of neural network developed for working with two-dimensional image data. A CNN model can be used in a hybrid model with an LSTM backend where the CNN is used to interpret subsequences of input that together are provided as a sequence to an LSTM model to interpret.

The CNN can be very effective at automatically extracting and learning features from one-dimensional sequence data such as univariate time series data. The first step is to split the input sequences into subsequences that can be processed by the CNN model. For example, we can first split our univariate time series data into input/output samples with four steps as input and one as output. Each sample can then be split into two sub-samples, each with two time steps. The CNN can interpret each subsequence of two time steps and provide a time series of interpretations of the subsequences to the LSTM model to process as input. The CNN model first has a convolutional layer for reading across the subsequence that requires a number of filters and a kernel size to be specified. The number of filters is the number of reads or interpretations of the input sequence. The kernel size is the number of time steps included of each 'read' operation of the input sequence. The convolution layer is followed by a max pooling layer that distills the filter maps down to 1/2 of their size that includes the most salient features. These structures are then flattened down to a single one-dimensional vector to be used as a single input time step to the LSTM layer.

- ConvLSTM

A type of LSTM related to the CNN-LSTM is the ConvLSTM where the convolutional reading of input is built directly into each LSTM unit. The ConvLSTM was developed for reading two-dimensional spatial-temporal data but can be adapted for use with univariate time series forecasting.

LSTM's VS RNN

- LSTM's and GRU's overcome the vanishing gradient problem which RNN's usually face.

- LSTM 's and GRU's can learn long-term dependencies that normal RNNs fundamentally can't. They were created as the solution to short-term memory. They can remember information for a long period of time. This is possible as they have internal mechanisms called gates that can regulate the flow of information. These gates can learn which data in a sequence is important to keep or throw away. By doing that, it can pass relevant information down the long chain of sequences to make predictions at the other end.

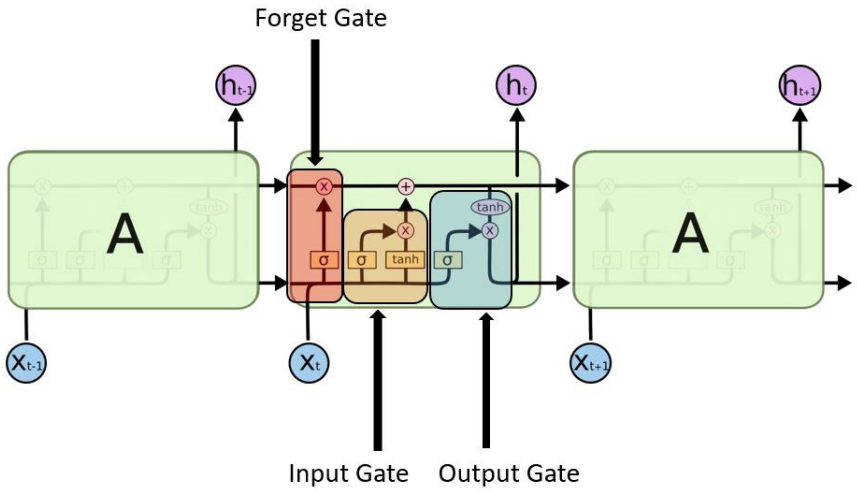
How it works

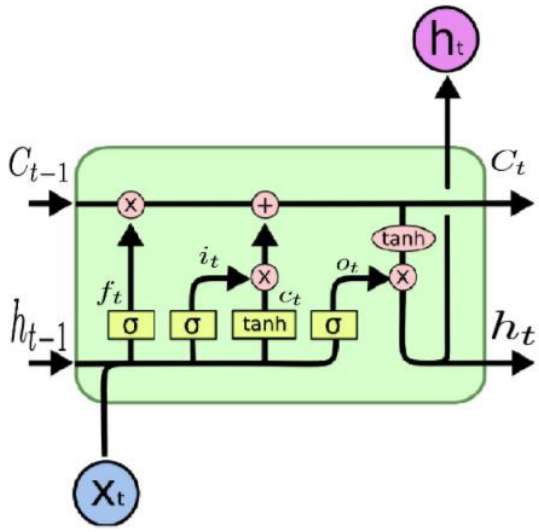
LSTM network is made up of cell state, forget gate, input gate and the output gate. The cell state act as a transport highway that transfers relative information all the way down the sequence chain. You can think of it as the memory of the network. So information from the earlier time steps can make its way to later time steps reducing the problem of short-term memory. As the cell state goes on its journey, information gets added or removed to the cell state via gates. The gates regulate the flow of information throughout the network.

How cell state is different from hidden state?

Cell state is a memory of LSTM cell while hidden state is an output of this cell. Hidden state and cell input are used to control what to do with memory i.e. whether to forget or to write new information. We decide what to do with memory knowing about previous output (hidden state) and current input.

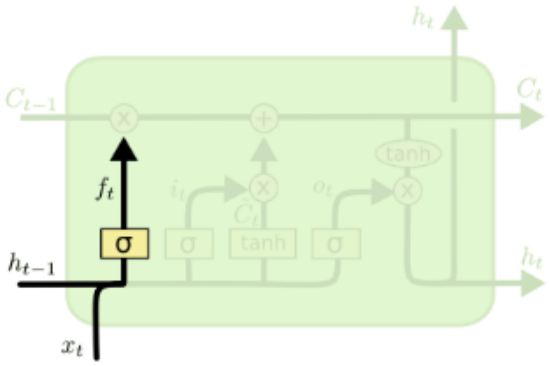
The flow of LSTM cell is shown below





Forget Gate - First we have the forget gate. The forget gate controls what information in the cell state to forget.

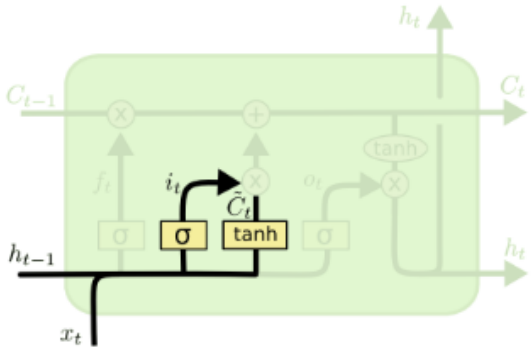
After getting the output of previous state h(t-1), forget gate helps us to take decisions about what must be removed from h(t-1) state i.e. what must be removed from C(t-1) cell state and thus keeping only relevant stuff. Forget gate is surrounded by a sigmoid function which helps to crush the input between [0,1]. The closer to 0 means to forget and the closer to 1 means to keep. We multiply forget gate with previous cell state to forget the unnecessary stuff from previous cell state which is not needed anymore



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

Input Gate - To update the cell state, we have the input gate. The input gate controls what new information will be added into the cell state. In the input gate we decide to add new stuff from the present input to our present cell state scaled by how much we wish to add them

First, we pass the previous hidden state and current input into a sigmoid function. That decides which values will be updated by transforming the values to be between 0 and 1. The closer to 0 means not important and the closer to 1 means important. You also pass the hidden state and current input into the tanh function to squish values between -1 and 1 to help regulate the network. Then you multiply the tanh output with the sigmoid output. The sigmoid output will decide which information is important to keep from the tanh output. In the below photo, sigmoid layer decides which values to be updated and tanh layer creates a vector for new candidates to be added to present cell state



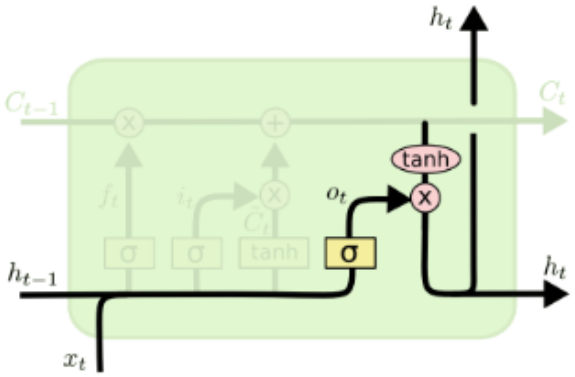
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

Cell state - Now we should have enough information to calculate the cell state. First, the cell state gets pointwise multiplied by the forget vector. This removes the information that we don't want to keep and keeps only the information that we want to take forward. Then we take the output from the input gate which is the relevant information from present state input and do a pointwise addition which updates the cell state to new values that the neural network finds relevant. That gives us our new cell state.

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

Output Gate - The output gate decides what the next hidden state should be and also controls what information in the cell state is sent to the network as input in the following time step.

First we pass the previous hidden state and the current input into a sigmoid function. Then we pass the newly modified cell state to the tanh function. We multiply the tanh output with the sigmoid output to decide what information the hidden state should carry. The output is the hidden state. The new cell state and the new hidden state is then carried over to the next time step.



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

$$h_t = o_t * \tanh\left(C_t\right)$$