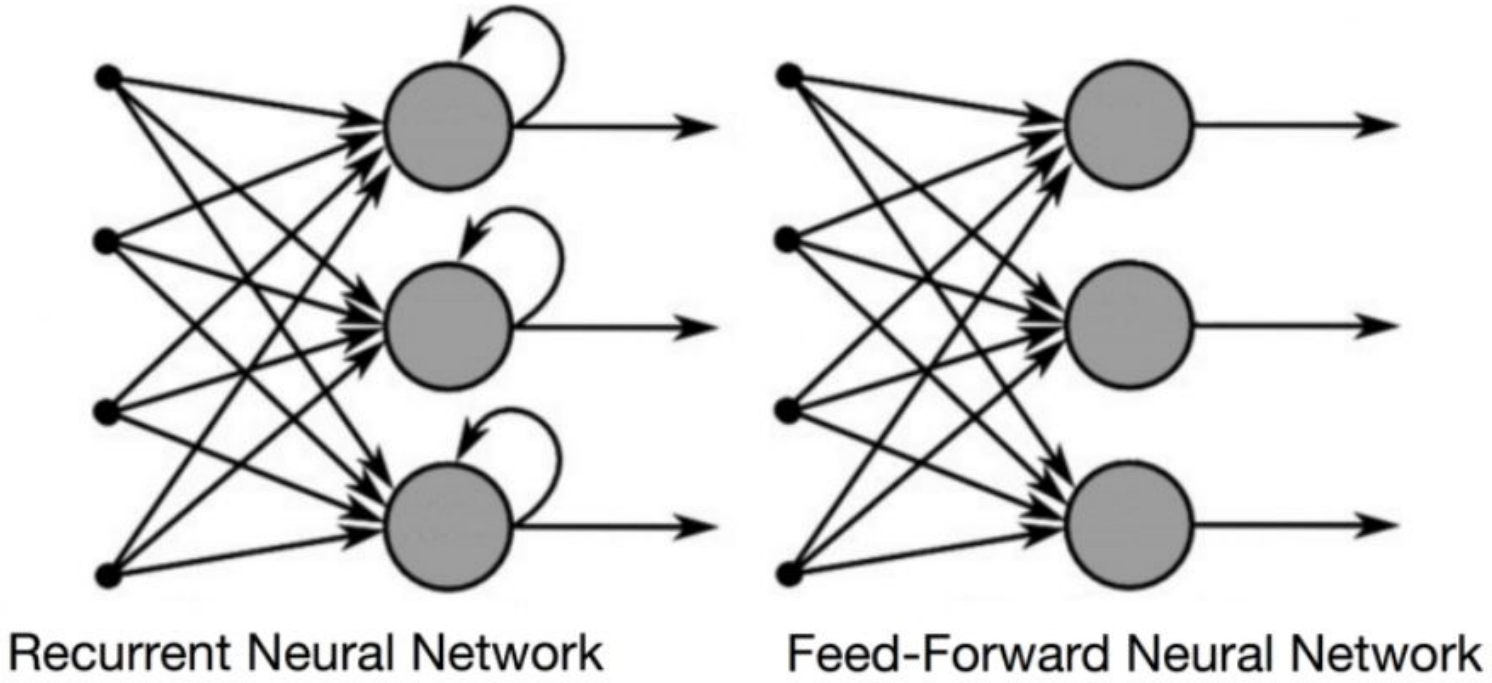
Recurrent Neural Networks (RNN)

Recurrent neural networks

- Recurrent Neural Networks(RNN) are a type of Neural Network where the output from the previous step is fed as input to the current step.
- For processing sequential data; time series, speech, text, audio, video etc.
- Remembers inputs due to an internal memory which helps in predicting what comes next
- The information cycles through a loop.
- While making decision, it considers the current input and also what it has learned from the past inputs, i.e., RNN adds the immediate past to the present.

Recurrent v/s Feed-forward Network



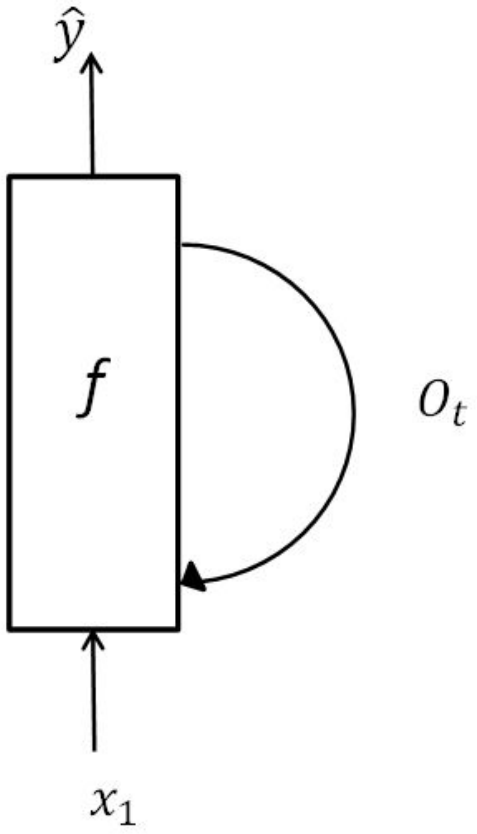
Applications

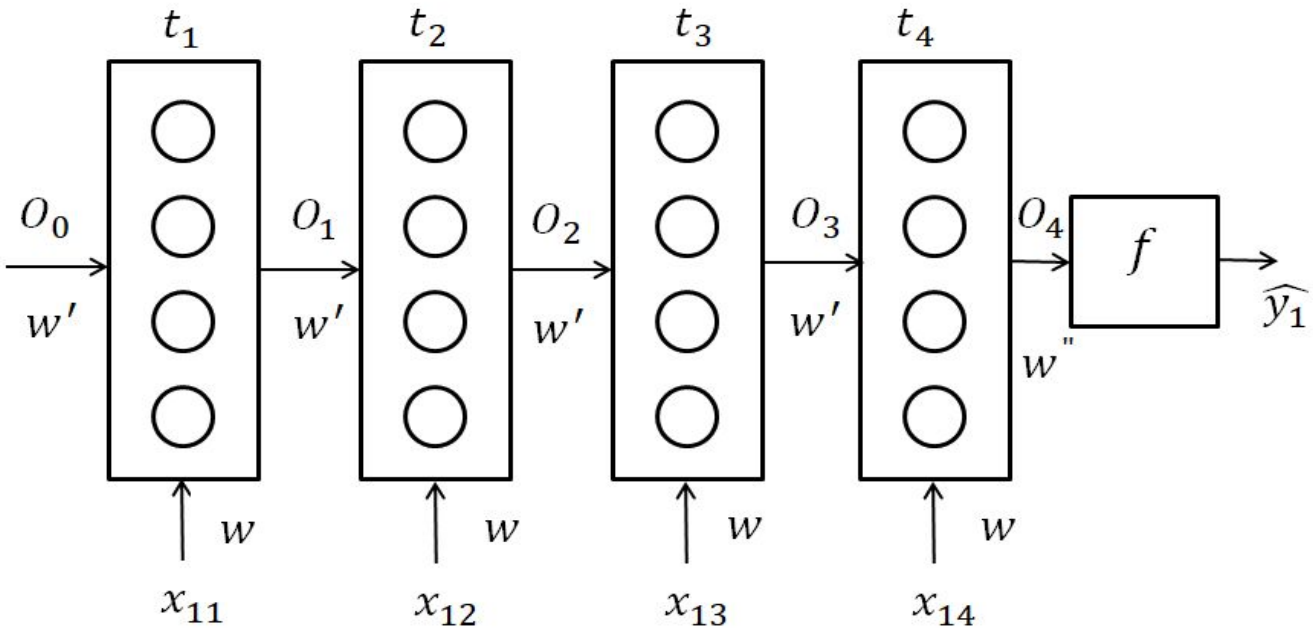
- Spam classification
- Sentiment analysis
- Language translation
- Time series data processing eg: Sales forecasting

Architecture

General representation

$$x_1 = \langle x_{11}, x_{12}, x_{13}, x_{14} \rangle$$

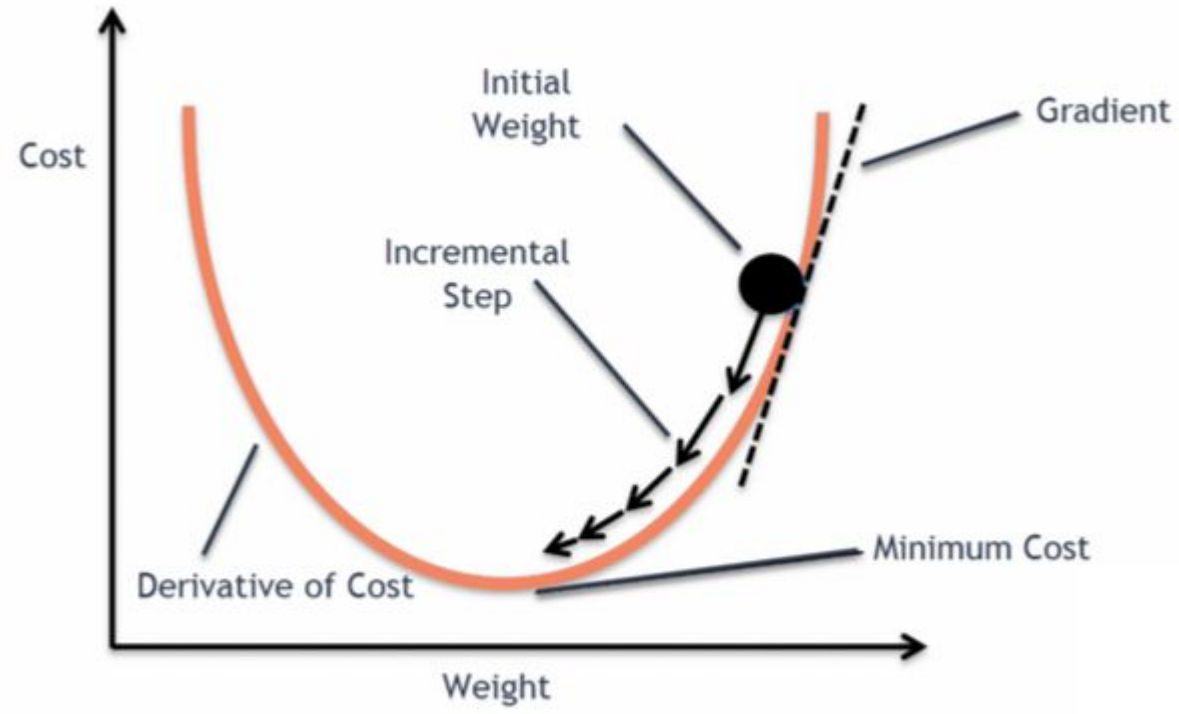




$$O_1 = f(x_{11}w + O_0w')$$
$$O_2 = f(x_{12}w + O_1w')$$
$$O_3 = f(x_{13}w + O_2w')$$
$$O_4 = f(x_{14}w + O_3w')$$

Now the loss, $\hat{y}$ - y, is calculated and is back propagated which needs to be reduced. An optimizer is used to reduce the loss so that $\hat{y}$ matches with y.

w'' is updated using $w'' = w'' - learning\ rate\ \frac{\partial loss}{\partial w''}$



When slope is negative, weight gets increased and vice-versa.

Learning rate decides the rate at which the weight gets updated. If the loss is not converging, it means there is some problem with the learning rate or the derivative term.

Chain rule in updating the weights

Updation of w'

$$w_{new}{}' = w_{old}{}' - \eta \frac{\partial L}{\partial w_{old}{}'}$$

$$\frac{\partial L}{\partial w'} = \left(\frac{\partial L}{\partial \widehat{y_1}} \frac{\partial \widehat{y_1}}{\partial O_4} \frac{\partial O_4}{\partial w'}\right) + \left(\frac{\partial L}{\partial \widehat{y_1}} \frac{\partial \widehat{y_1}}{\partial O_4} \frac{\partial O_4}{\partial O_3} \frac{\partial O_3}{\partial w'}\right) + \left(\frac{\partial L}{\partial \widehat{y_1}} \frac{\partial \widehat{y_1}}{\partial O_4} \frac{\partial O_4}{\partial O_3} \frac{\partial O_3}{\partial O_2} \frac{\partial O_2}{\partial w'}\right) +$$

$$\left(\frac{\partial L}{\partial \widehat{y_1}} \frac{\partial \widehat{y_1}}{\partial O_4} \frac{\partial O_4}{\partial O_3} \frac{\partial O_3}{\partial O_2} \frac{\partial O_2}{\partial O_1} \frac{\partial O_1}{\partial w'}\right)$$

Updation of w

$$w_{new} = w_{old} - \eta \frac{\partial L}{\partial w_{old}}$$

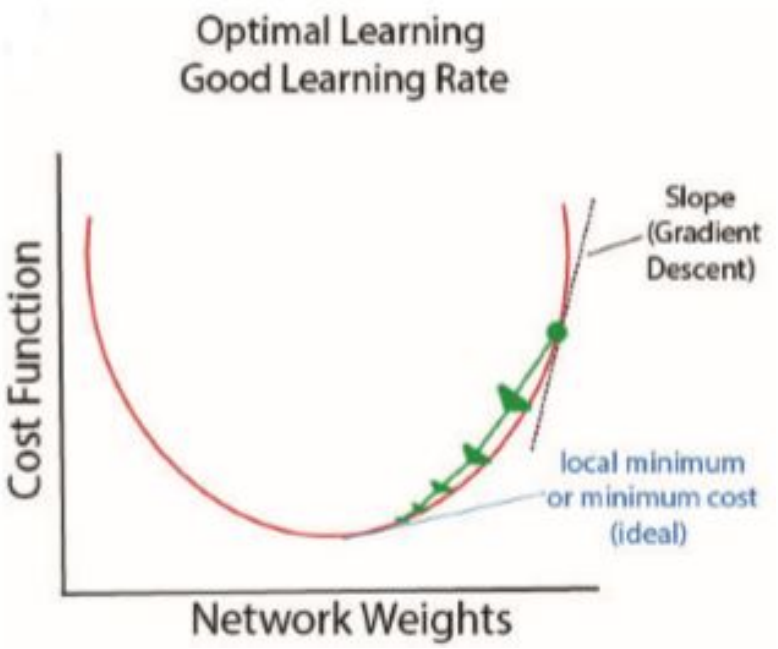
$$\frac{\partial L}{\partial w} = \left(\frac{\partial L}{\partial \widehat{y_1}} \frac{\partial \widehat{y_1}}{\partial O_4} \frac{\partial O_4}{\partial w}\right) + \left(\frac{\partial L}{\partial \widehat{y_1}} \frac{\partial \widehat{y_1}}{\partial O_4} \frac{\partial O_4}{\partial O_3} \frac{\partial O_3}{\partial w}\right) +$$

$$\left(\frac{\partial L}{\partial \widehat{y_1}} \frac{\partial \widehat{y_1}}{\partial O_4} \frac{\partial O_4}{\partial O_3} \frac{\partial O_3}{\partial O_2} \frac{\partial O_2}{\partial w}\right) + \left(\frac{\partial L}{\partial \widehat{y_1}} \frac{\partial \widehat{y_1}}{\partial O_4} \frac{\partial O_4}{\partial O_3} \frac{\partial O_3}{\partial O_2} \frac{\partial O_2}{\partial O_1} \frac{\partial O_1}{\partial w}\right)$$

Problems in simple RNN

- Vanishing gradient problem

The gradient will be vanishingly small, effectively preventing the weight from changing its value while using sigmoid activation function



- 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.

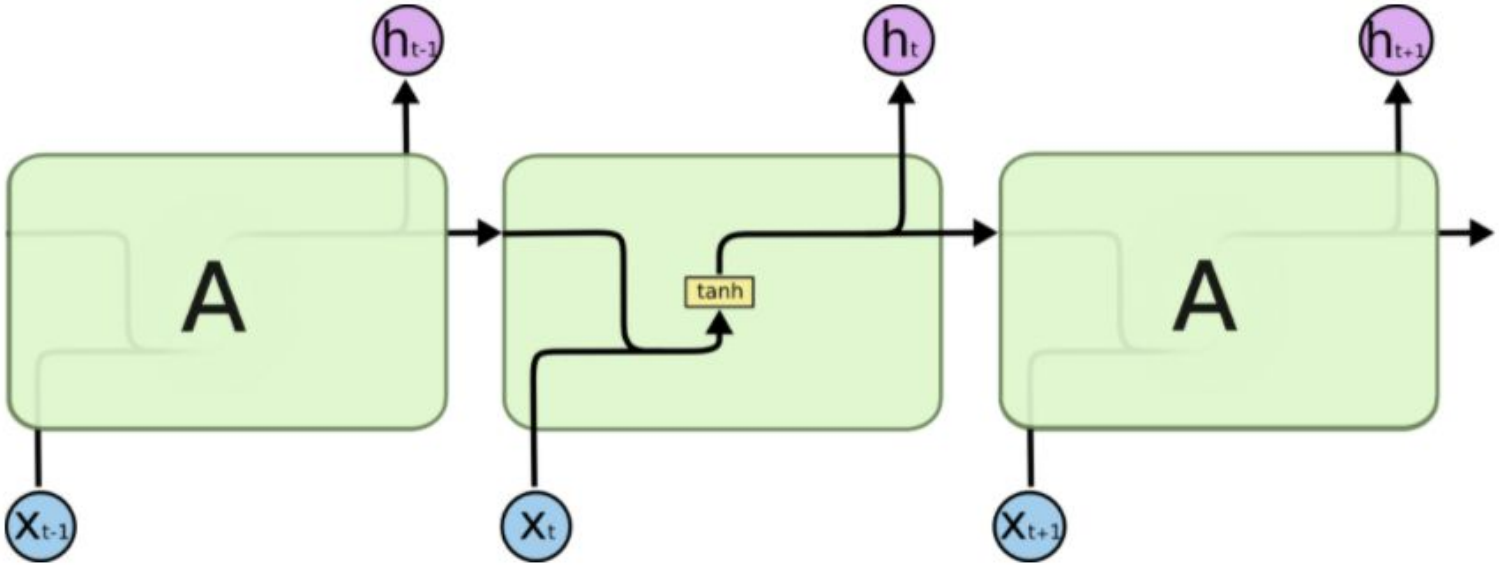
LSTM (Long Short Term Memory) Network

Problem of long term dependencies

RNNs can connect previous information to the present task. It is fine when the gap between the relevant information and the place where it is needed is small. But it is possible that the gap between the relevant information and the point where it is needed is large. As the gap grows, RNN becomes unable to learn to connect the information.

LSTM

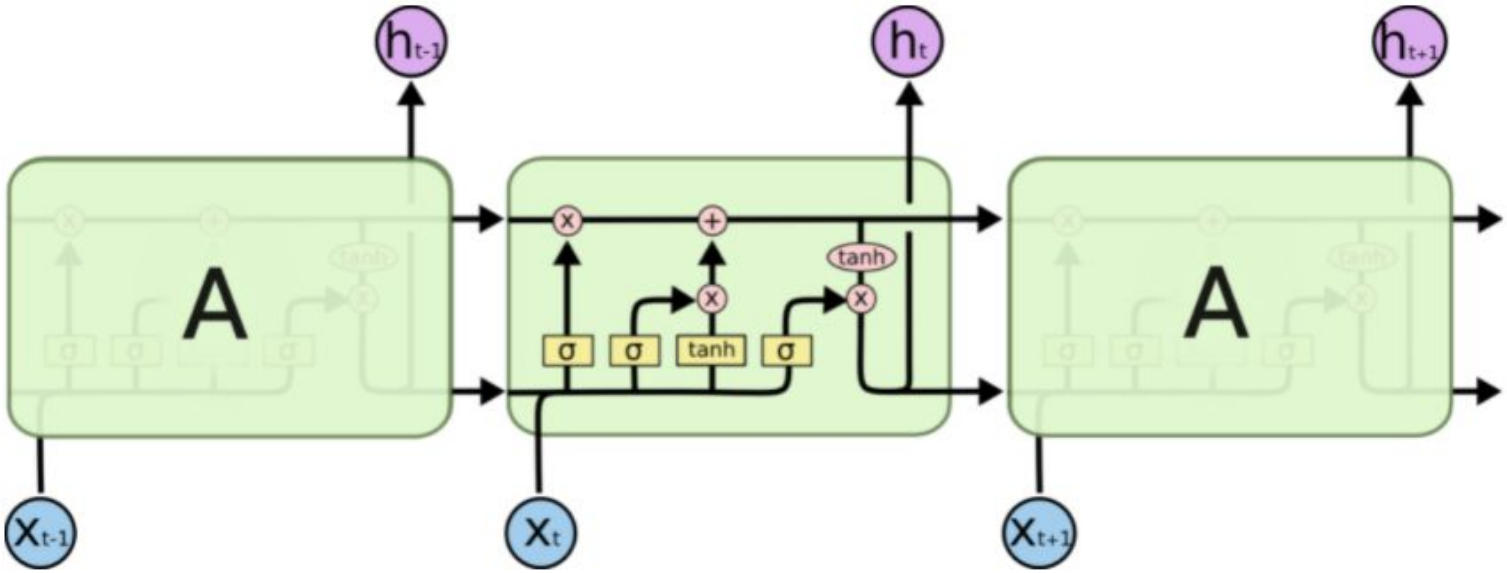
It is a special kind of RNN capable of learning long term dependencies.



The repeating module in a standard RNN contains a single layer.

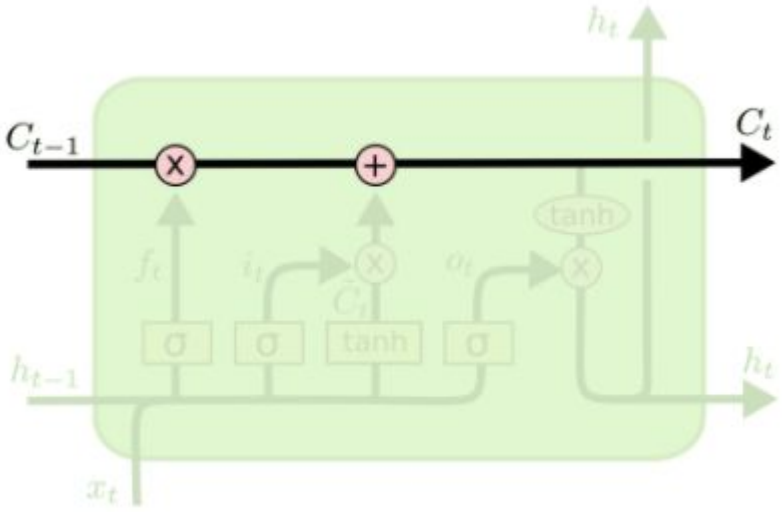
Instead of a single neural network layer, there are 4 interacting layers: a **cell state** (which is the memory of the LSTM cell) and **3 gates** that update and control the cell states.

The gates use tanh and sigmoid activation functions.

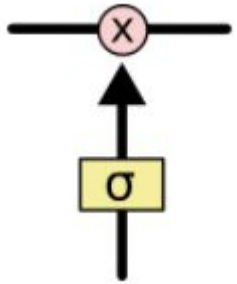


The repeating module in an LSTM contains four interacting layers.

The horizontal line running through the top is the **cell state** which runs straight down the entire chain, with some minor linear interactions.



LSTM can add or remove information to the cell state through structures called **gates**. They optionally let information through. They consists of a sigmoid neural net layer and a point wise multiplication operation.

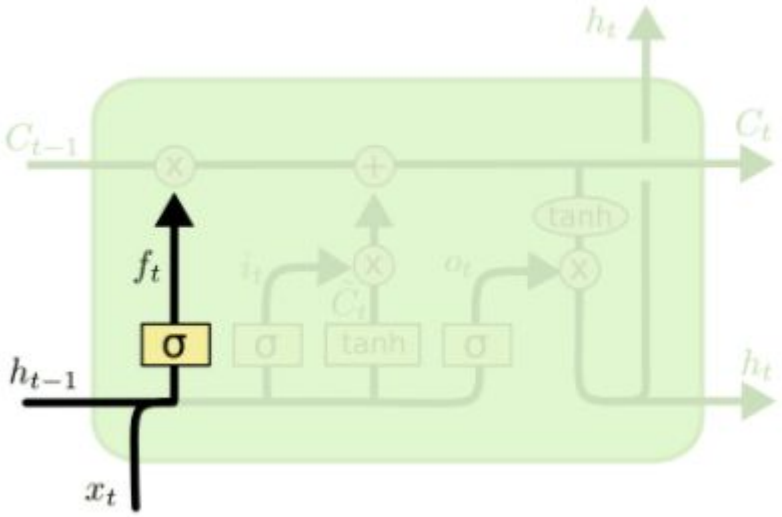


The output of sigmoid layer is values between 0 and 1; 0 meaning ‘let nothing’ and 1 meaning ‘let everything’.

Gate layers in LSTM and steps in processing

Step1

Forget gate layer: A sigmoid layer that decides what information needs to be kept and what needs to be thrown out. It controls what information in the cell state needs to be forgot.



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

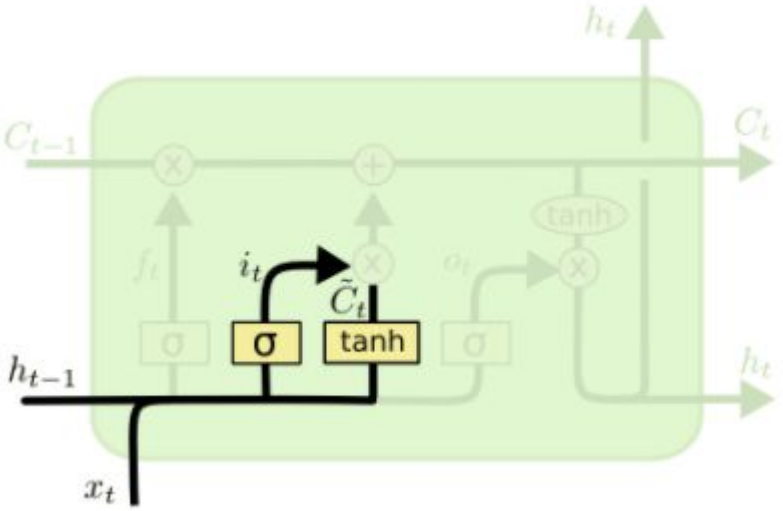
Step2

Decide what new information is going to be stored in the cell state.

It has 2 parts:

A sigmoid layer called **input gate layer** decides which values we will update.

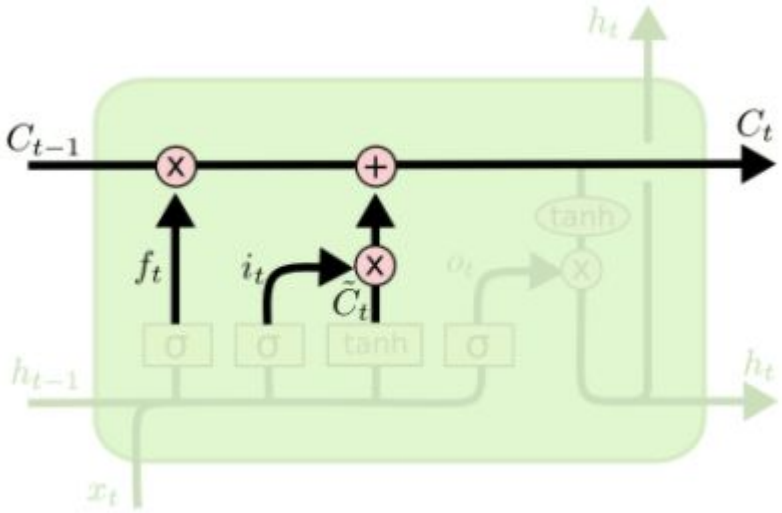
A **tanh layer** creates a vector of new candidate values, $\widetilde{C}_t$, that could be added to the state.



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

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

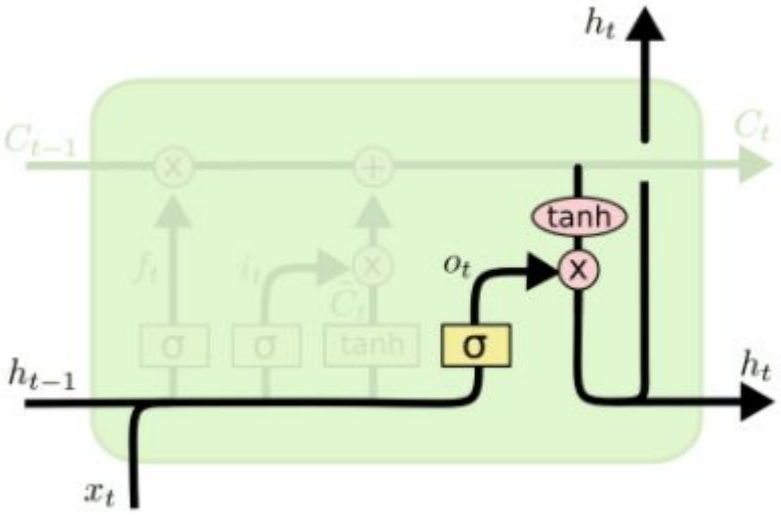
Now update the old cell state C_{t-1} by new cell state C_t
Multiply the old state by f_t, forgetting things decided earlier. Then $i_t * \widetilde{C}_t$ is added to it. This is the new candidate values scaled by how much we decided to update each state value.



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

Step 3

Finally the **output gate** decides what is going to the output. This is based on the cell state. First the sigmoid layer decides what parts of the cell state goes out. Then the cell state is passed through the tanh layer. This multiplies by the output of sigmoid layer gives the final output.

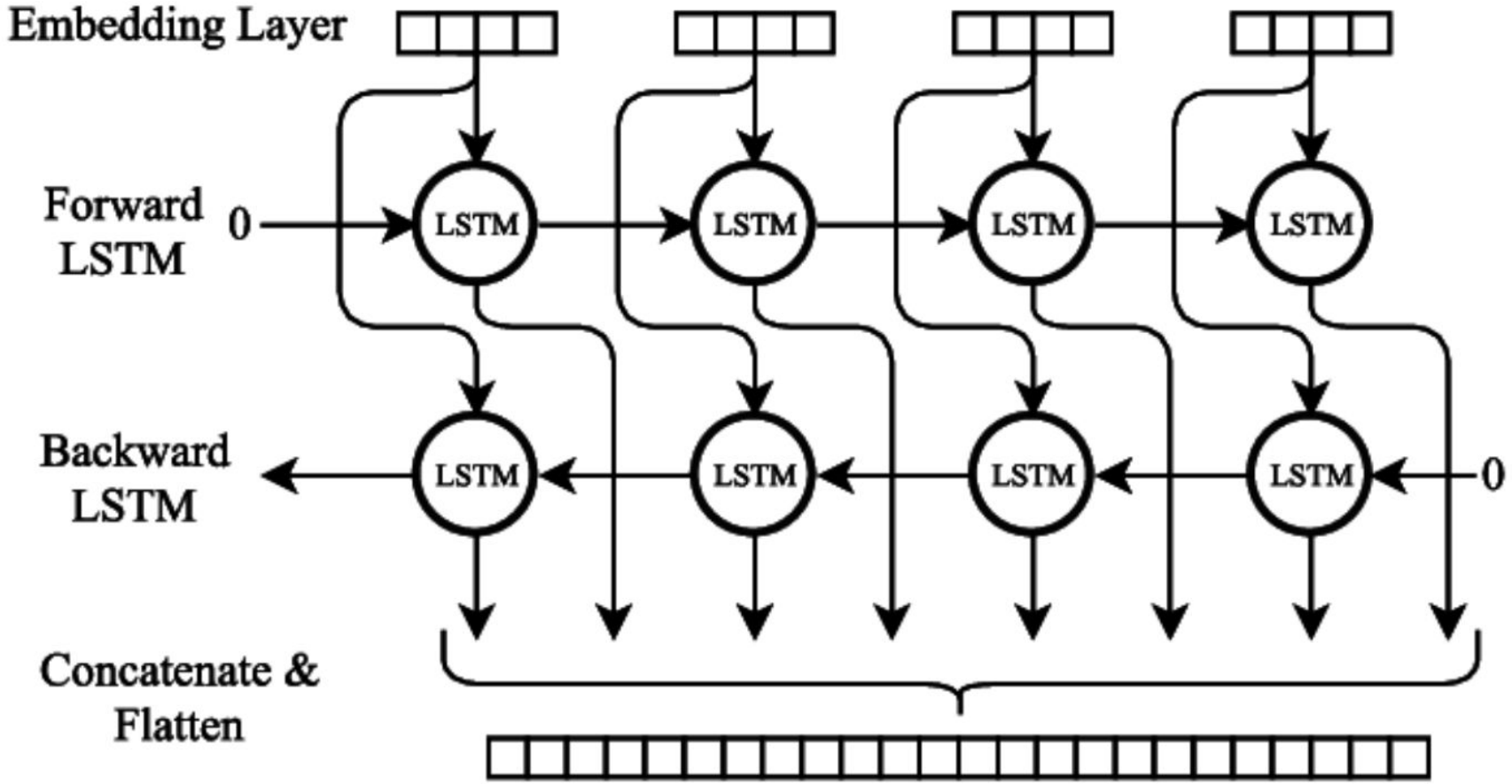


$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$

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

Bidirectional LSTM (Bi-LSTM)

- Making a neural network to have the sequence information in both directions; backwards (future to past) or forward (past to future).
- Here, the input flows in 2 directions to preserve the future and the past information.
- It is a powerful tool for modeling the sequential dependencies between words and phrases in both directions of the sequence.
- BiLSTM simply contains one more LSTM layer which reverses the direction of information flow.
- The outputs of both LSTMs are then combined.


Embedding Layer
Forward LSTM
0
LSTM
LSTM
LSTM
LSTM
Backward LSTM
LSTM
LSTM
LSTM
LSTM
0
Concatenate & Flatten