Analytics Club IIT M x Shaastra

2023

ML in Finance

A workshop teaching how Machine Learning principles are used in the world of markets. Enjoy

www.analyticsclubiitm.com

Introducing Sequential Data

Stocks are not discrete chunks of data but rather a sequence of numbers that have some correlation between them. So far, we have only dealt with discrete independent data points. Now we will learn to process sequential data , and to create predictive models that help us analyse them.

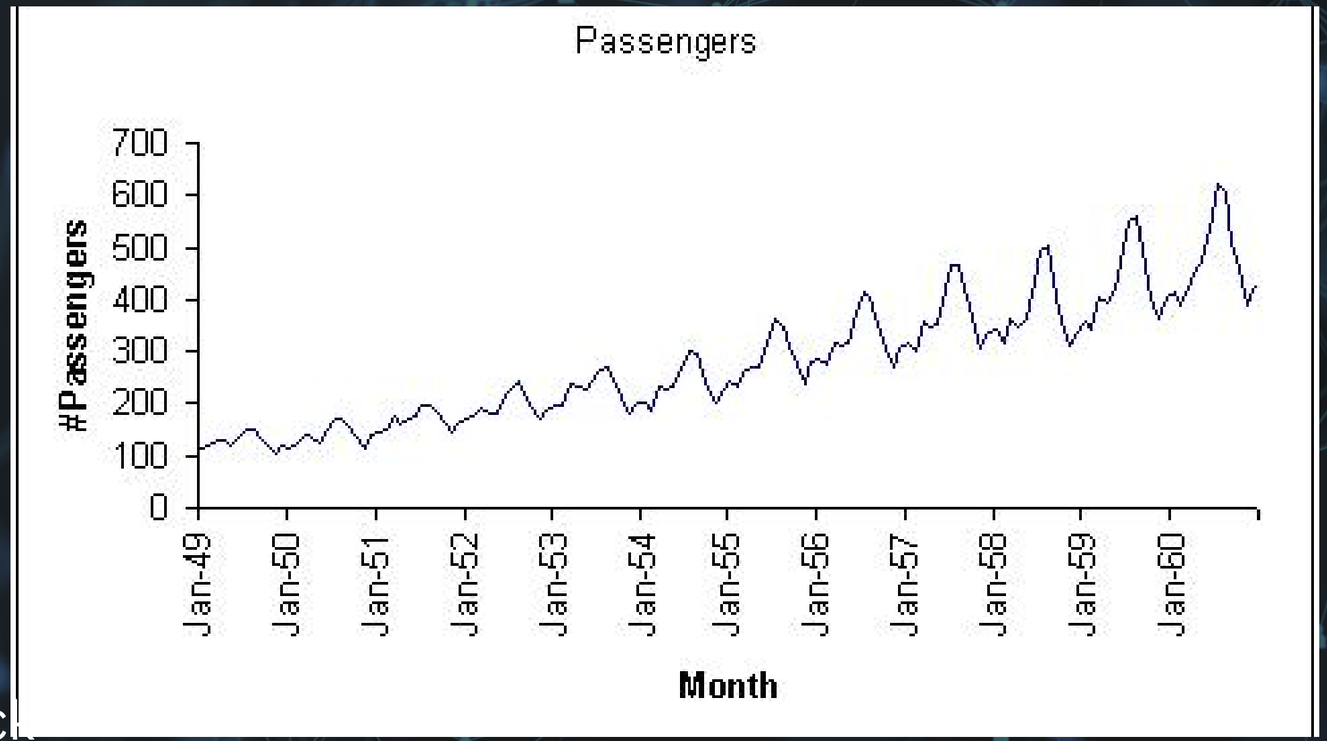
Why Sequential and not independent??

Say we have market data for some days and we want to predict the stock prices for the next 4 days. It would make sense to analyse the previous four days' stock prices in an orderly fashion(chronologically) rather than a randomized collection of numbers.

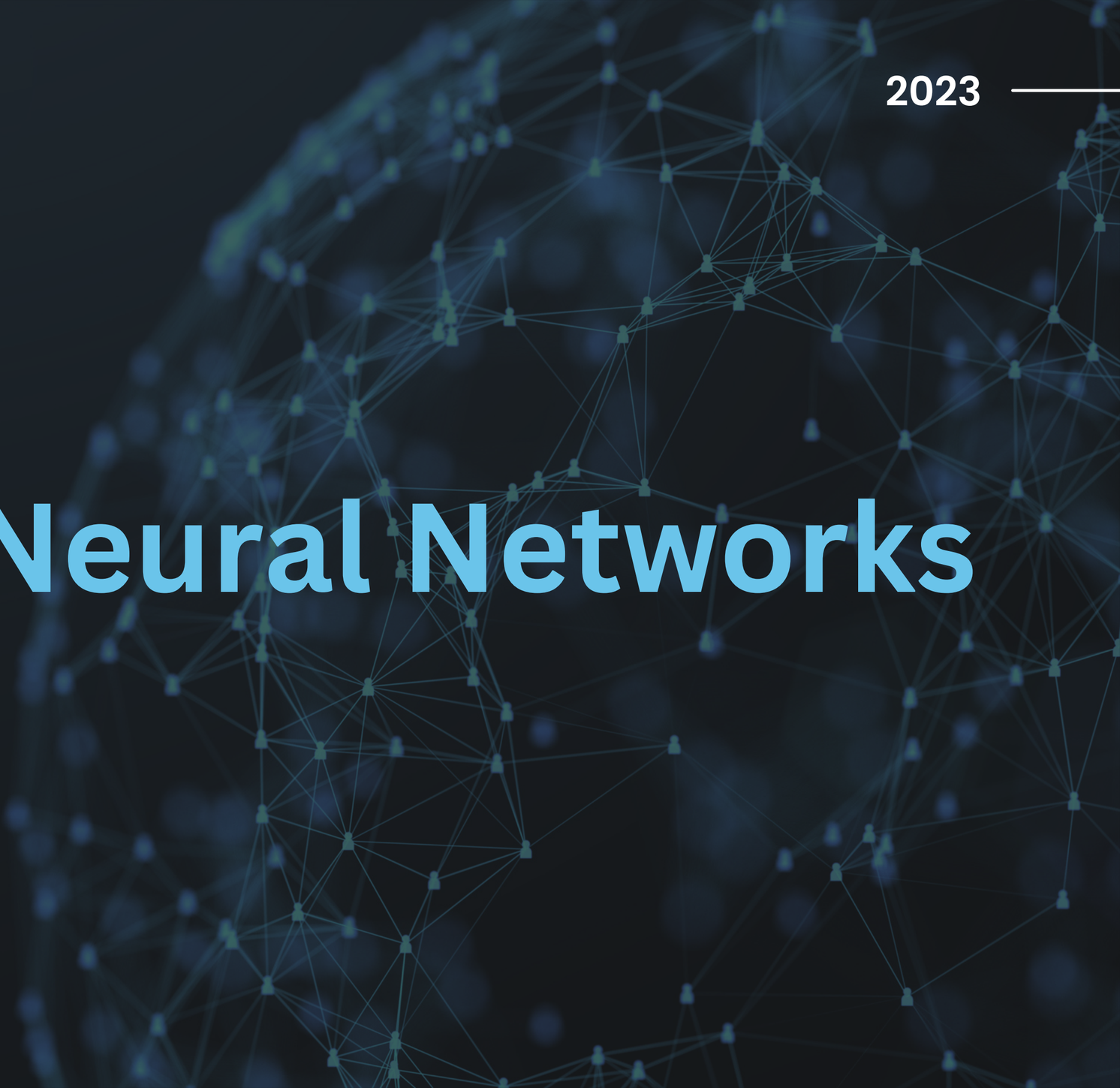
Time Series Data

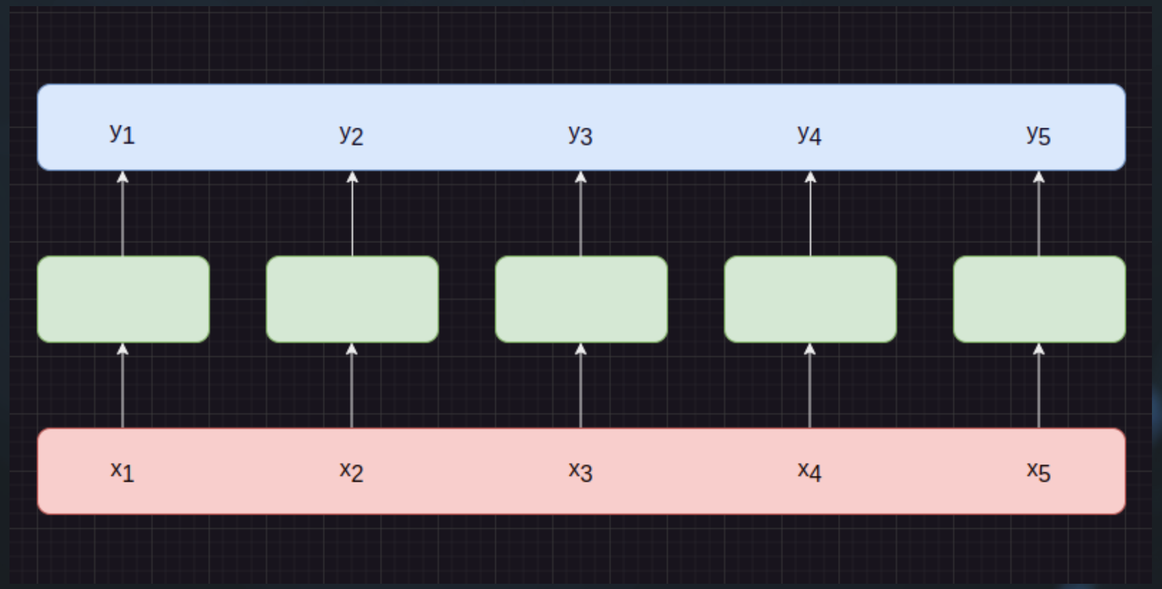
A time series data
Passengers vs Month

- Time series data is a special form of Sequential Data.
- Samples are taken at successive intervals
- called 'time-stamps' and the data is ordered as per the same.
- Not all Sequential Data is
- a Time series model. For example:
- DNA Sequences are ordered , dependent
- data points but are not dependent with
- each other in the dimension of time.
- Examples of Time series data include Stock prices,voice records,
- speech records and all data which was collected with time as a reference



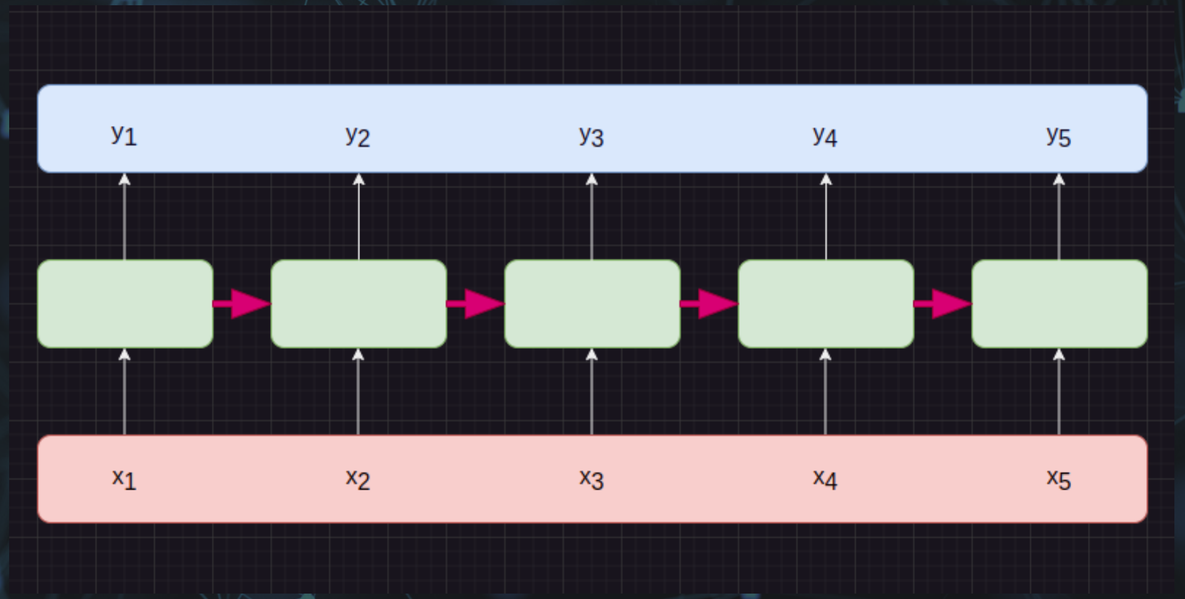
Recurrent Neural Networks

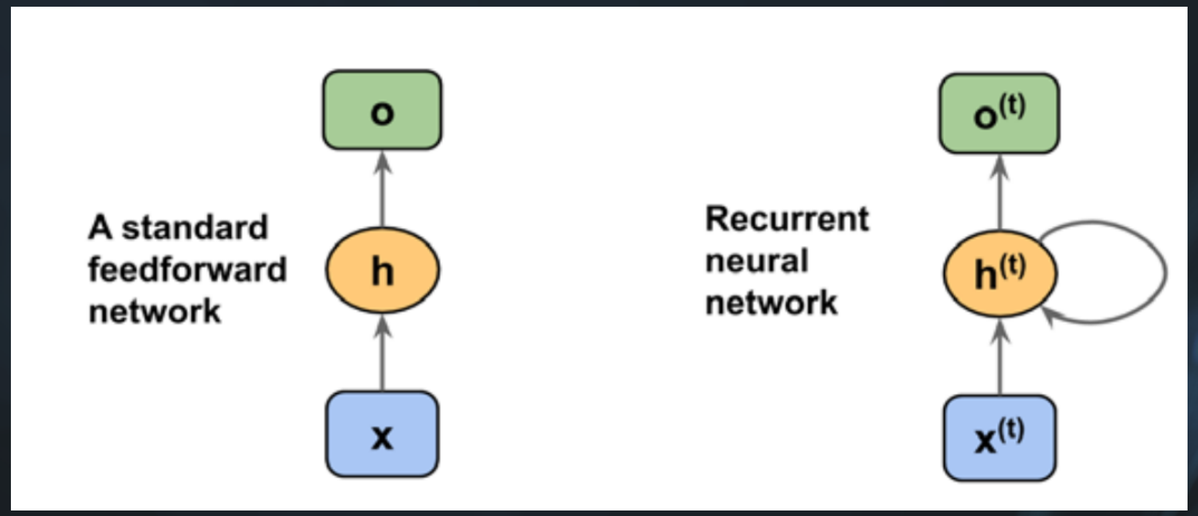




Regular Models: Processing cells are independent of each other

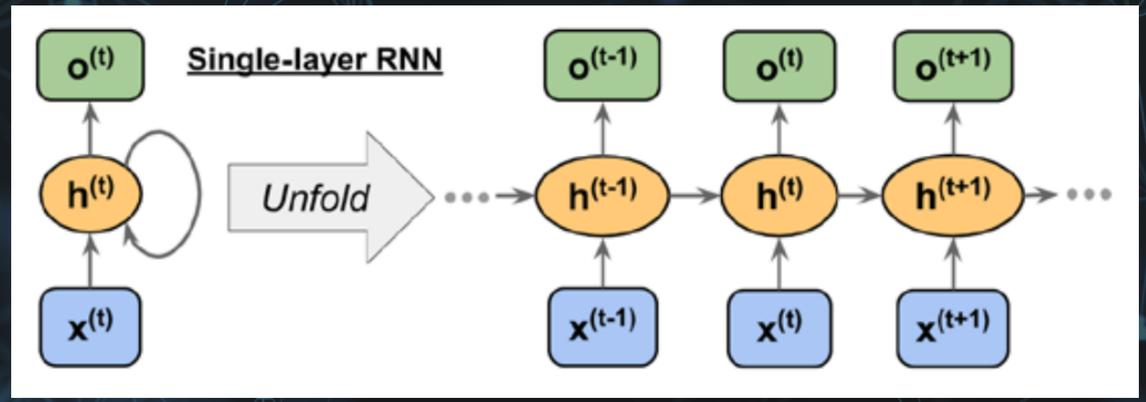
In RNNs, the blocks carry information from the previous steps to predict values





A simple comparison between an RNN and a simple Neural Network.
We can see how there is a recursive relation in the hidden layer of the RNN.

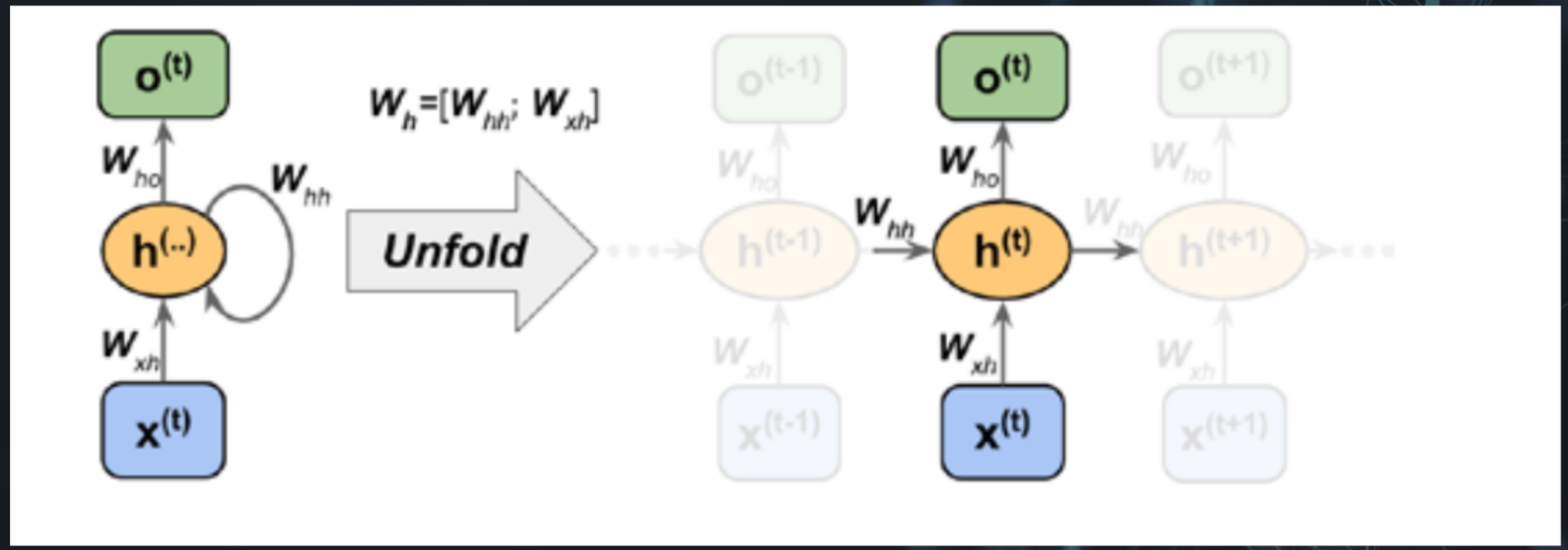
- In a standard feedforward network, information flows from the input to the hidden layer, and then fromthe hidden layer to the output layer.
- On the other hand, in an RNN, the hidden layer receives its input from both the input layer of the current time step and the hidden layer from the previous time step.



Let us consider a simple single layered RNN as an example for calculating the activation at the hidden layer.

These three matrices will decide the output at time (t)

- W_X_h: The weight matrix between the input, x(t), and the hidden layer, h
- W_h_h: The weight matrix associated with the recurrent edge
- W_h_o: The weight matrix between the hidden layer and output layer



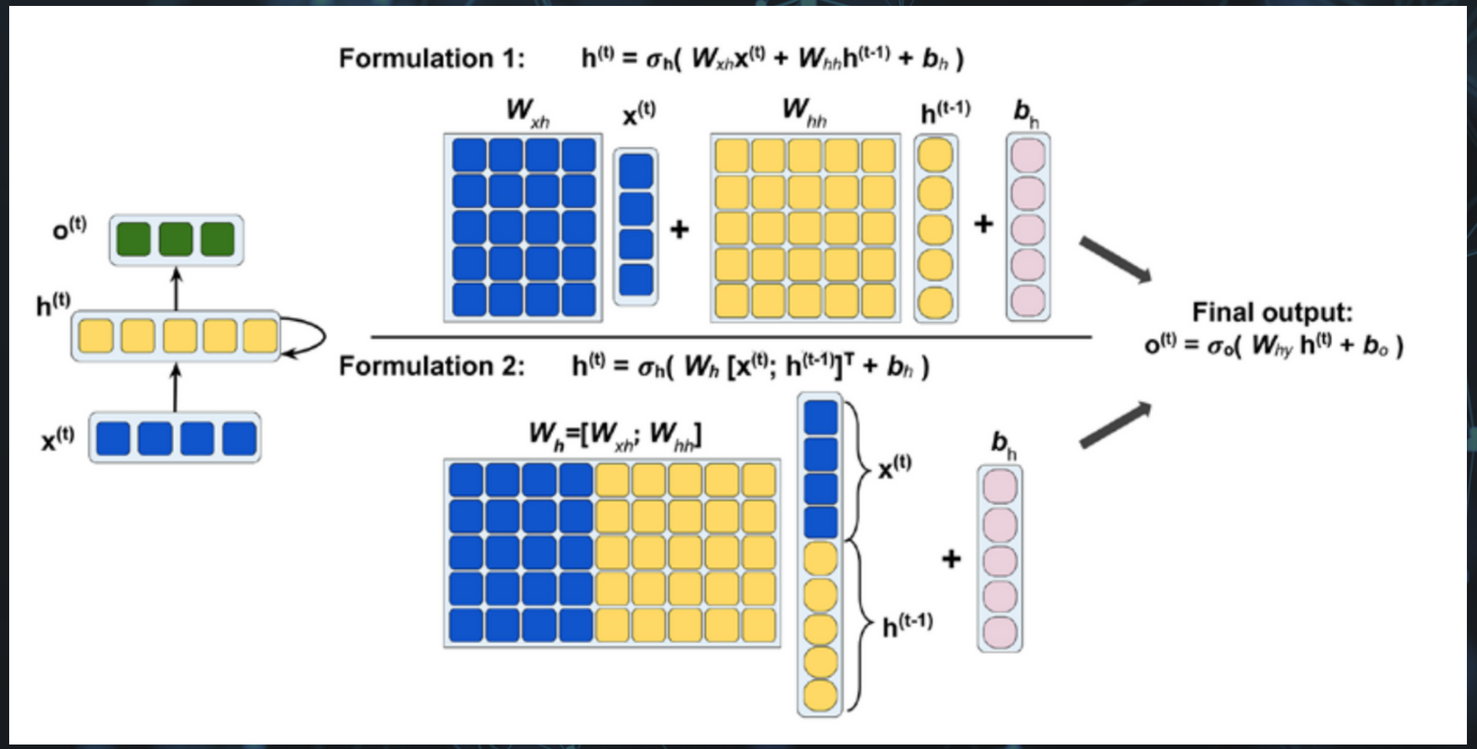
$$Z_h^t = W_{xh}x^t + W_{hh}h^{t-1} + b_h$$

Equation 1: Output w/o activation

$$Z_h^t = \sigma(W_{xh}x^t + W_{hh}h^{t-1} + b_h)$$

Equation 2: Output post activation

A very brief behind the mathematics of RNNs
We can simplify the sum into a single matrix product as shown in the figure



Some note points

- RNNs have connections between layers of different time steps
- There can be a recurrence from outputs to hidden layers as well instead of just hidden layers
- Recurrent Neural Networks are generally hard to train because of the dense data and the multi-directional movement of signals unlike simple feedforward neural networks.
- Because the features of the previous time-step have been incorporated in the hidden layers of the subsequent cells, a time series function can be created that can predict stock prices(our case) , by taking historical data into consideration as well!!!

Challenges faces by RNNs → Need for LSTMs

- RNNs struggle with something known as long-term dependencies.
- Sometimes in text or time-series data, the dependency of the current data is a bit far into the past, and hence RNNs may not be able to capture over such a long range
- Thankfully LSTMs are an architecture that can capture long range dependencies and can correlate between time-series data with ease.

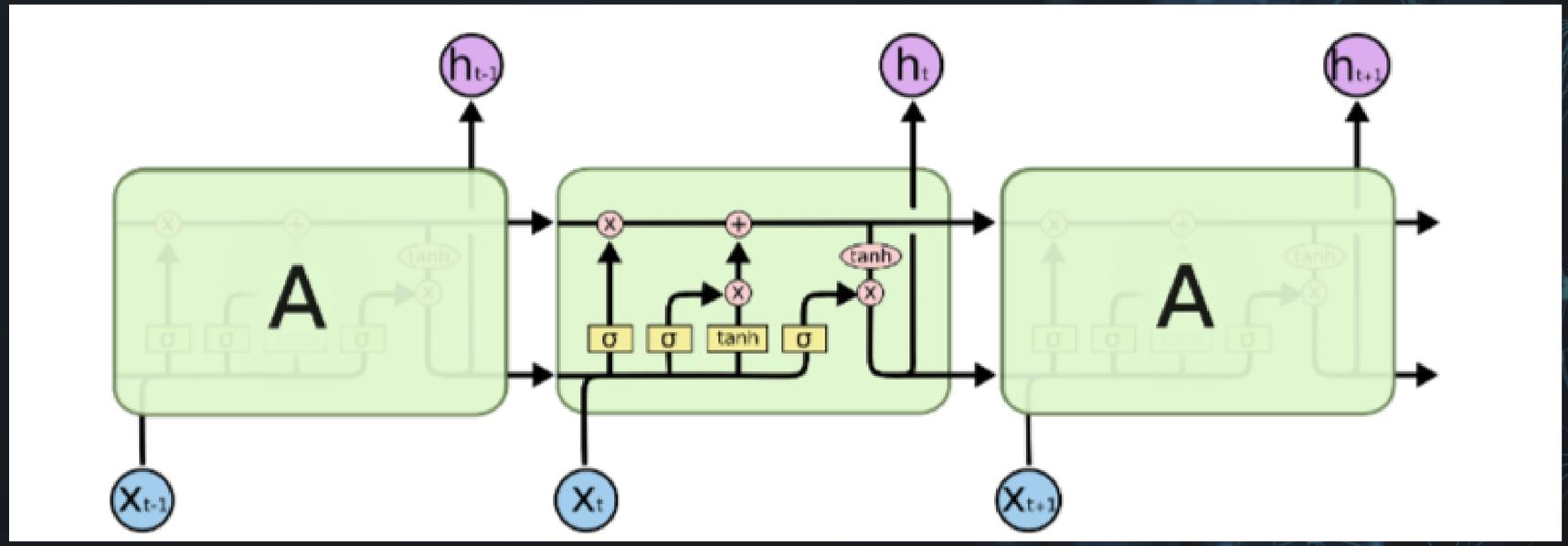
LSTMs

Intro to LSTMs

- Long-Short-Term-Memory Networks, commonly known as LSTMs, are a special type of RNNs that can learn long-term dependencies.
- Remembering information for a longer period of time is their natural state and can be done easily.
- As we saw in the previous slides, RNNs have a chain like structure of repeating modules of a Neural Network. LSTM has this too, just a bit more complex and suited for it's purpose

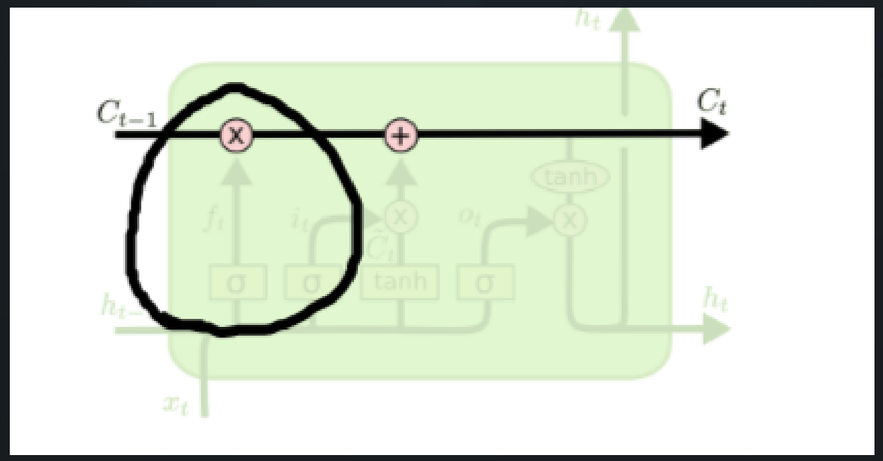
Structure of LSTMs

The repeating cell of RNN consists of a single layer. However in an LSTM, we have 4 layers, interacting in a unique way

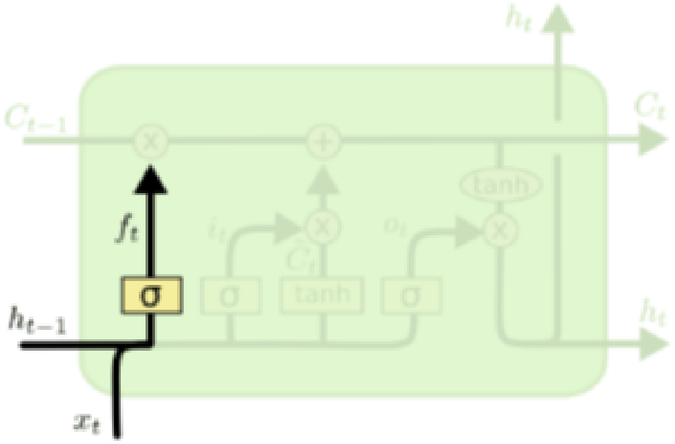


- The lines represent 1-D vectors
- Pink circles are simple pointwise operations such as addition and element-wise product. Yellow boxes are trained neural network layers.

- The cell state is mainly carried through the pipeline at the top of the cell
- The LSTM networks can make changes to this pipeline in a regulated manner
- The first sigmoid layer is called the forget layer which returns a value from 0-1 .



The value 0 means all the information entering the cell from the previous cell will be lost. 1 means everything is retained.

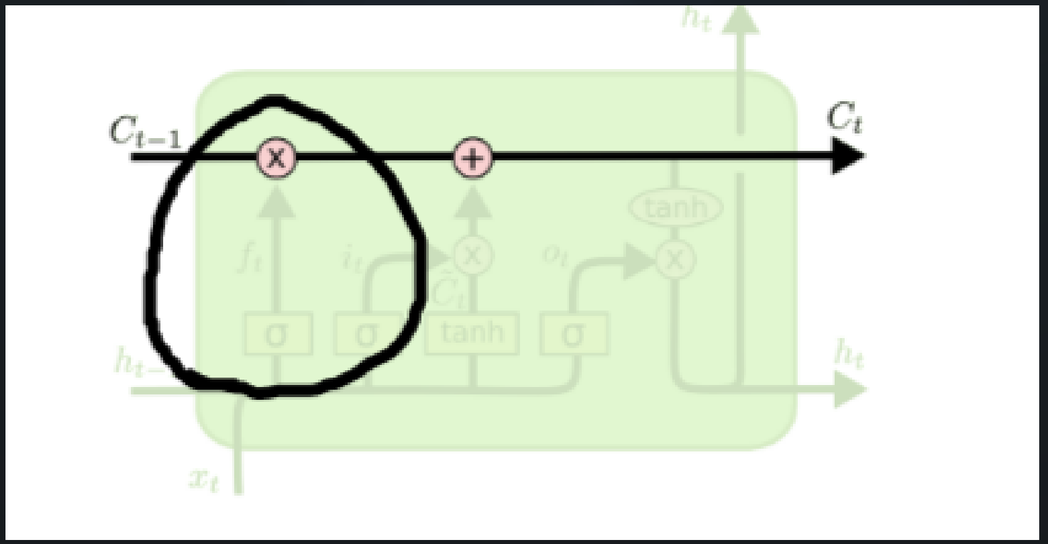


$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

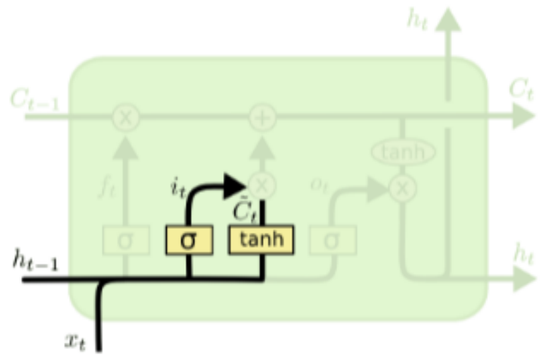
Working of the forget layer.

This is the first layer of the cell, the second layer which we will look at has two parts and is a bit more complicated.

- The cell state is mainly carried through the pipeline at the top of the cell
- The LSTM networks can make changes to this pipeline in a regulated manner
- The first sigmoid layer is called the forget layer which returns a value from 0-1 .



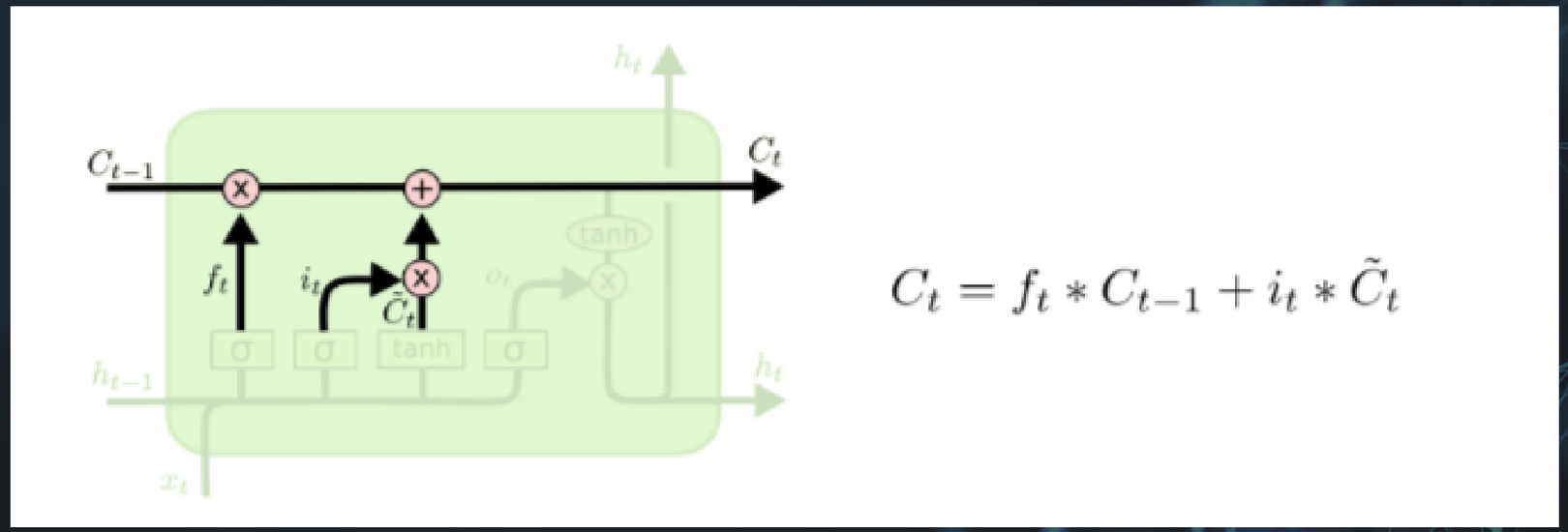
The value 0 means all the information entering the cell from the previous cell will be lost. 1 means everything is retained.



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

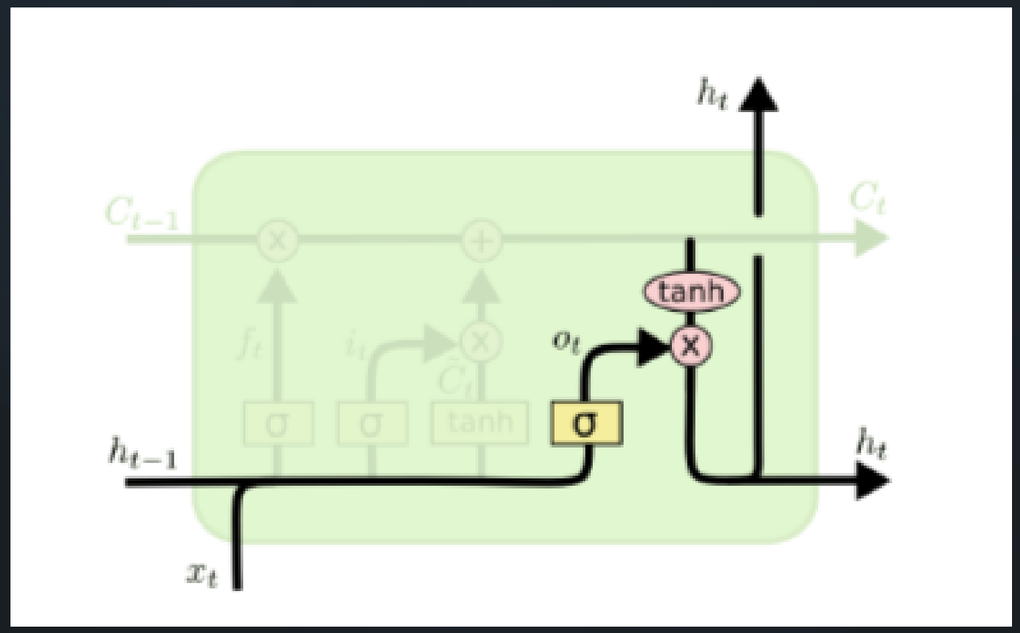
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

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, Ct in the figure, that could be added to the state. In the next step, we’ll combine these two to create an update to the state.



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

- It’s now time to update the old cell state, Ct−1, into the new cell state Ct. The previous steps already decided what to do, we just need to actually do it.
- We multiply the old state by ft(forget state), forgetting the things we decided to forget earlier. Then we add it Ct to it. This is the new candidate values, scaled by how much we decided to update each state value.

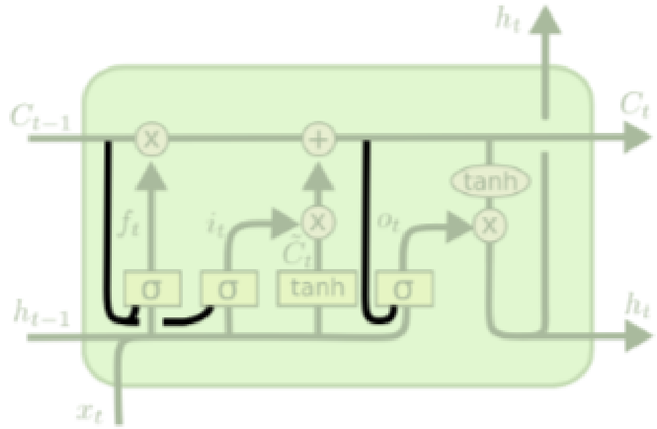


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

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

- 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 tanh (to push the values to be between −1 and 1)
- Finallymultiply it by the output of the sigmoid gate, so that we only output the parts we decided to.

There are some other variants of LSTM.
The differences are minor, but it’s worth mentioning some of them.



$$f_t = \sigma\left(W_f \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \; + \; b_f\right)$$
$$i_t = \sigma\left(W_i \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \; + \; b_i\right)$$
$$o_t = \sigma\left(W_o \cdot [\boldsymbol{C_t}, h_{t-1}, x_t] \; + \; b_o\right)$$

One such variant here above is the "peephole" LSTM where we let gate layers look at cell states .

Conv LSTM

- ConvLSTM is a Convolutional-LSTM model, in which the image passes through the convolutions layers and its result is a set flattened to a 1D array with the obtained features. When repeating this process to all images in the time set, the result is a set of features over time, and this is the LSTM layer input.
- ConvLSTM layer is a Recurrent layer, just like the LSTM, but internal matrix multiplications are exchanged with convolution operations.

Conv LSTM layer input

- The input of a ConvLSTM is a set of images over time as a 5D tensor with shape (samples, time_steps, channels, rows, cols).

Conv LSTM layer output

- The ConvLSTM layer output is a combination of a Convolution and a LSTM output. Just like the LSTM, if return_sequences = True, then it returns a sequence as a 5D tensor with shape (samples, time_steps, filters, rows, cols). On the other hand, if return_sequences = False, then it returns only the last value of the sequence as a 4D tensor with shape (samples, filters, rows, cols).

Applications of LSTM

- Sentiment Analaysis : categorizing opinion in order to determine whether the writer's attitude towards a particular topic is positive, negative, or neutral.
- Language modeling : It is the used to determine the probability of a given sequence of words occurring in a sentence.
- Speech recognition : It is the ability of a machine or program to identify words spoken aloud and convert them into readable text.

CNN LSTM

- The CNN LSTM architecture involves using Convolutional Neural Network (CNN) layers for feature extraction on input data combined with LSTMs to support sequence prediction.

- CNN LSTMs were developed for visual time series prediction problems and the application of generating textual descriptions from sequences of images (e.g. videos).

This architecture is used specifically for problems of

- Activity Recognition: Generating a textual description of an activity demonstrated in a sequence of images.

- Image Description: Generating a textual description of a single image.

- Video Description: Generating a textual description of a sequence of images.

CNN LSTM vs LSTM

- CNN LSTM has much higher about 50 times training and prediction speed as compared to normal LSTM.
- This is true even when CNN LSTM has more training parameters than normal LSTM.
- CNN leads to a variety of different complexity reductions by concentrating on the key features. The use of convolution layers leads to a reduction in the size of tensors.
- Also the use of pooling leads to a further reduction. And last but not least the ReLu layer reduces the complexity.
- Because of this the training time decreases