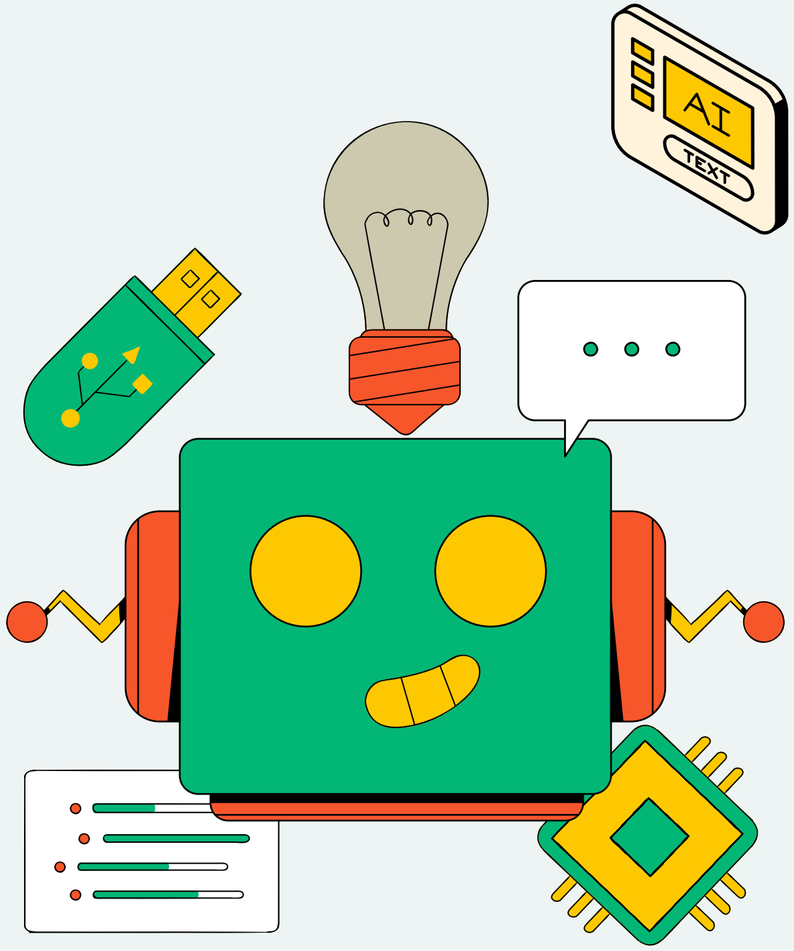
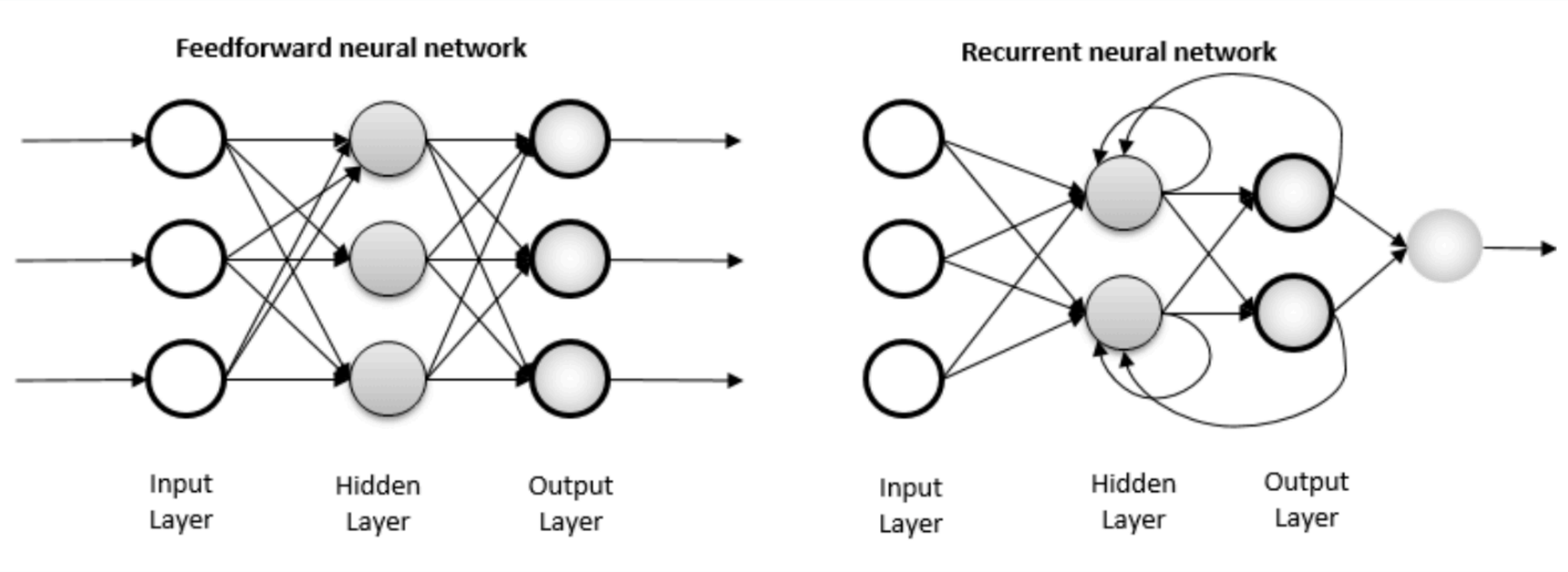


RECURRENT NEURAL NETWORKS

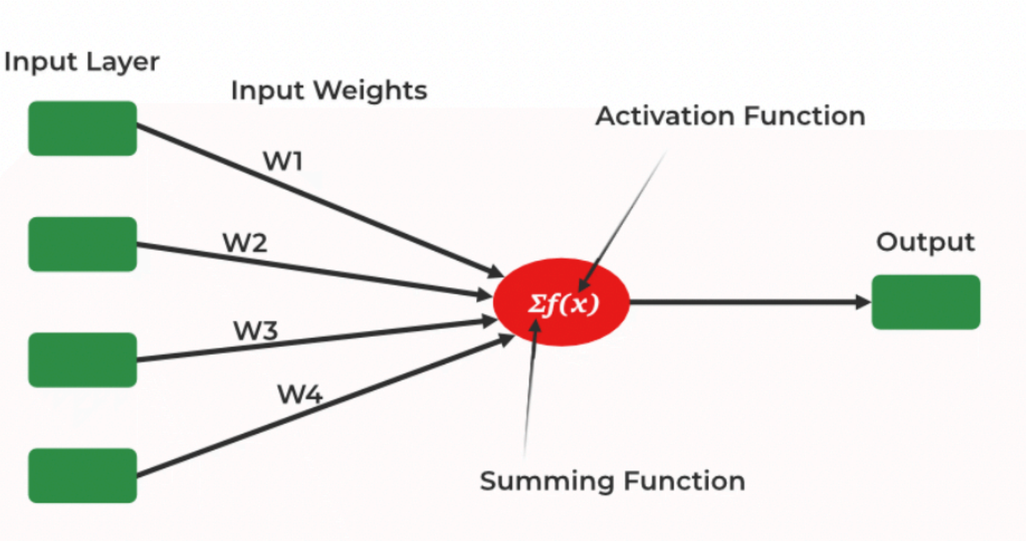
DIFFERENT TYPES OF NEURAL NETWORKS



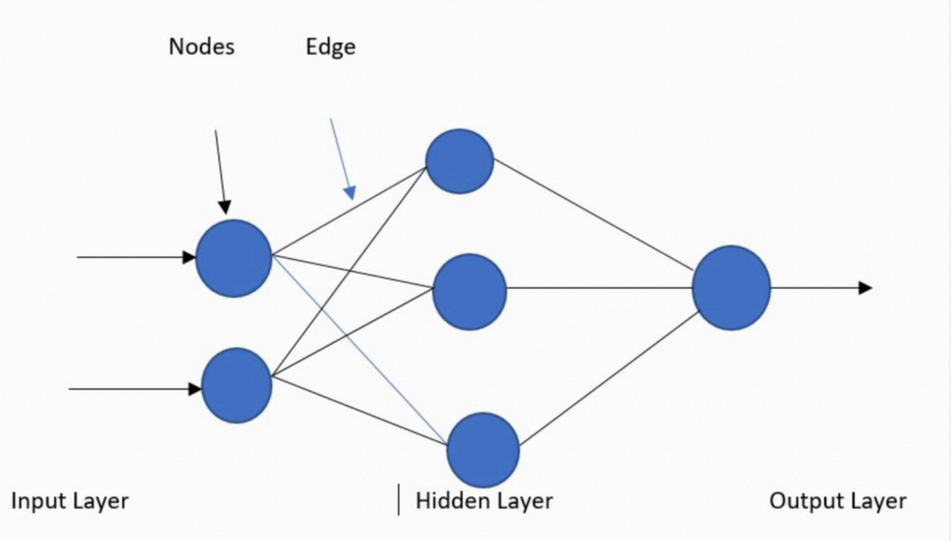
FEED-FORWARD NEURAL NETWORKS

Here the information flows in one direction only. This does not mean that there will be no back-propagation, it means that one input is not going to be used when wokring on another input – the model has no memory!
some types include–

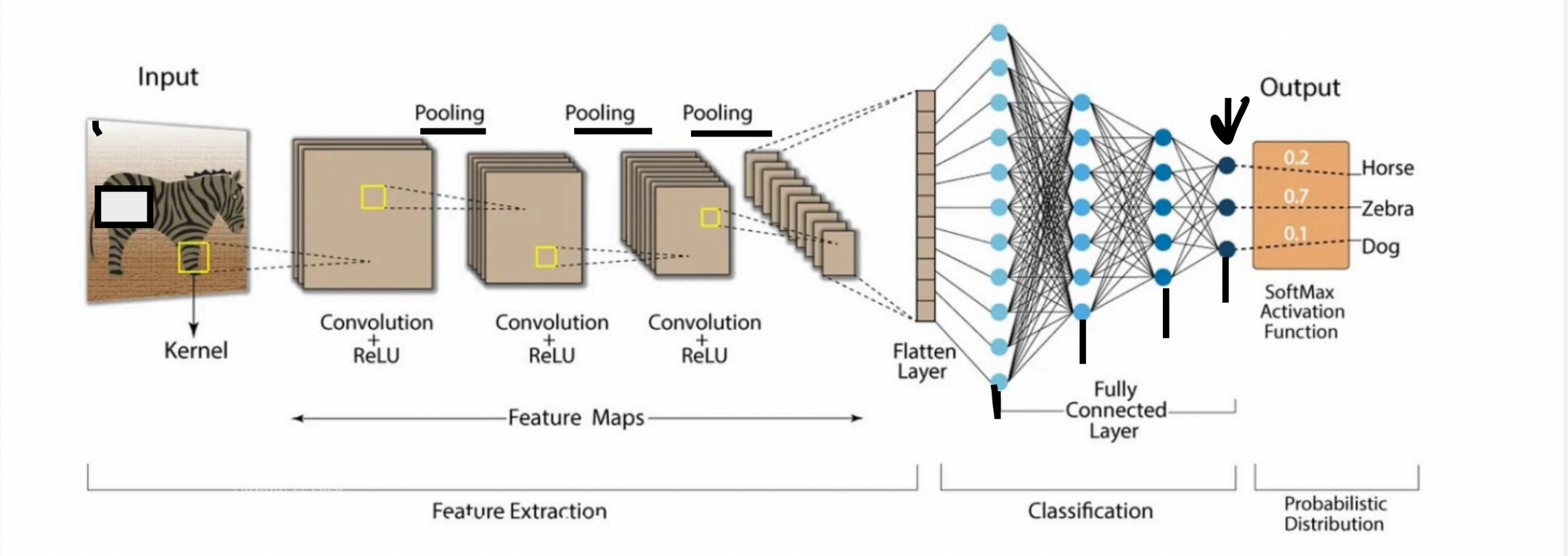
SINGLE-LAYER PERCEPTRON



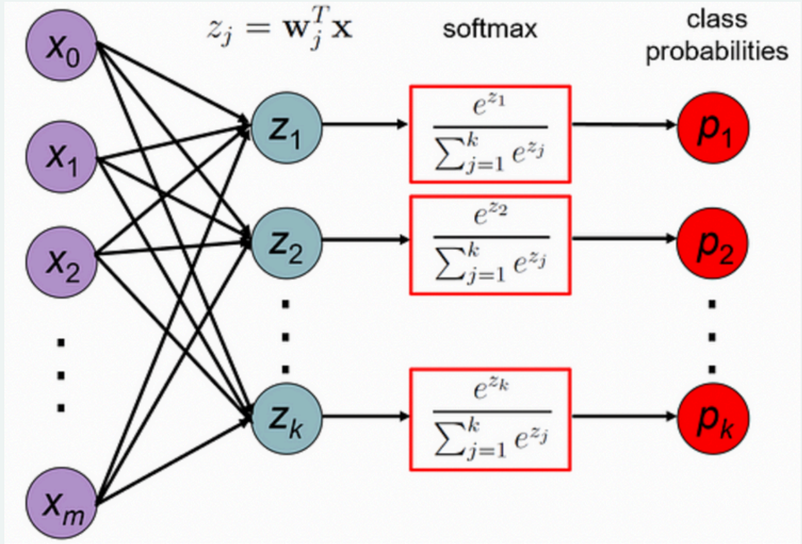
MULTI-LAYER PERCEPTRON



CONVOLUTIONAL NEURAL NETWORKS



THE SOFTMAX ACTIVATION FUNCTION

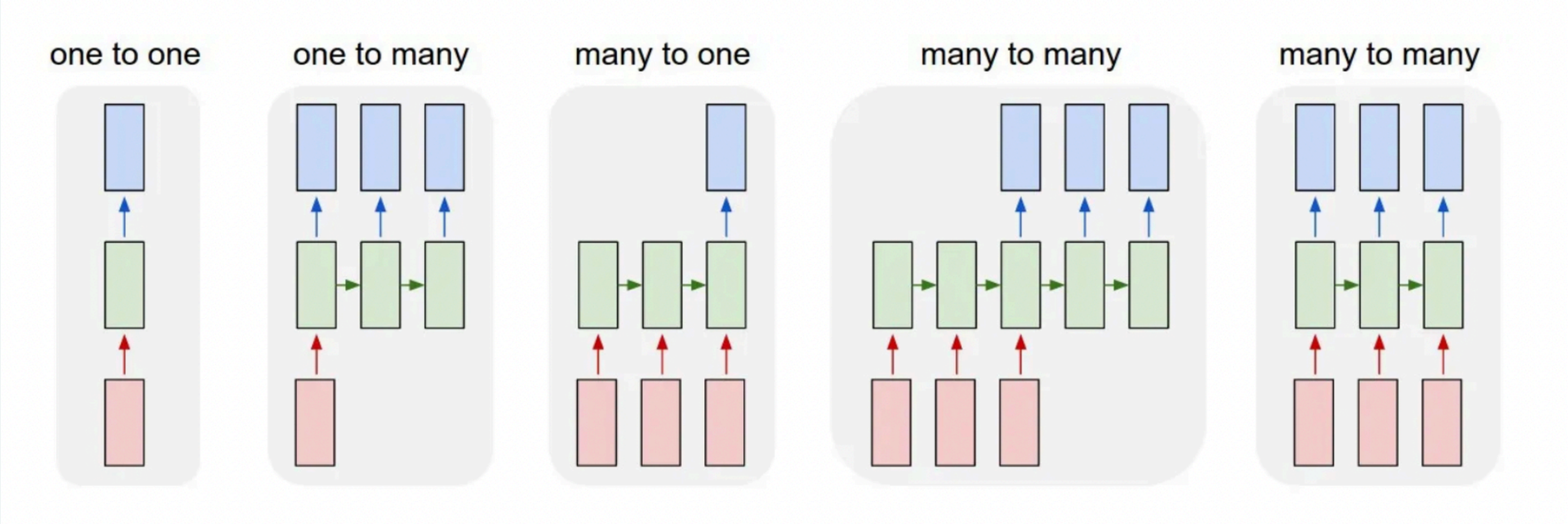


- consider a neural network being used for multi-class classification. The final layer will contain as many nodes as there are classes in the dataset.
- Essentially, each node of this layer is going to learn a linear combination of nodes in the previous layer.
- The ‘softmax’ or the ‘categorical activation function’ is used to translate the values in this layer to the relative probability that the given input belongs to a particular class.
- Now, the sum of values in the last node will be 1, agreeing with the rules of a probability distribution function.
- Predicted class isgenerally the class with max. probability.

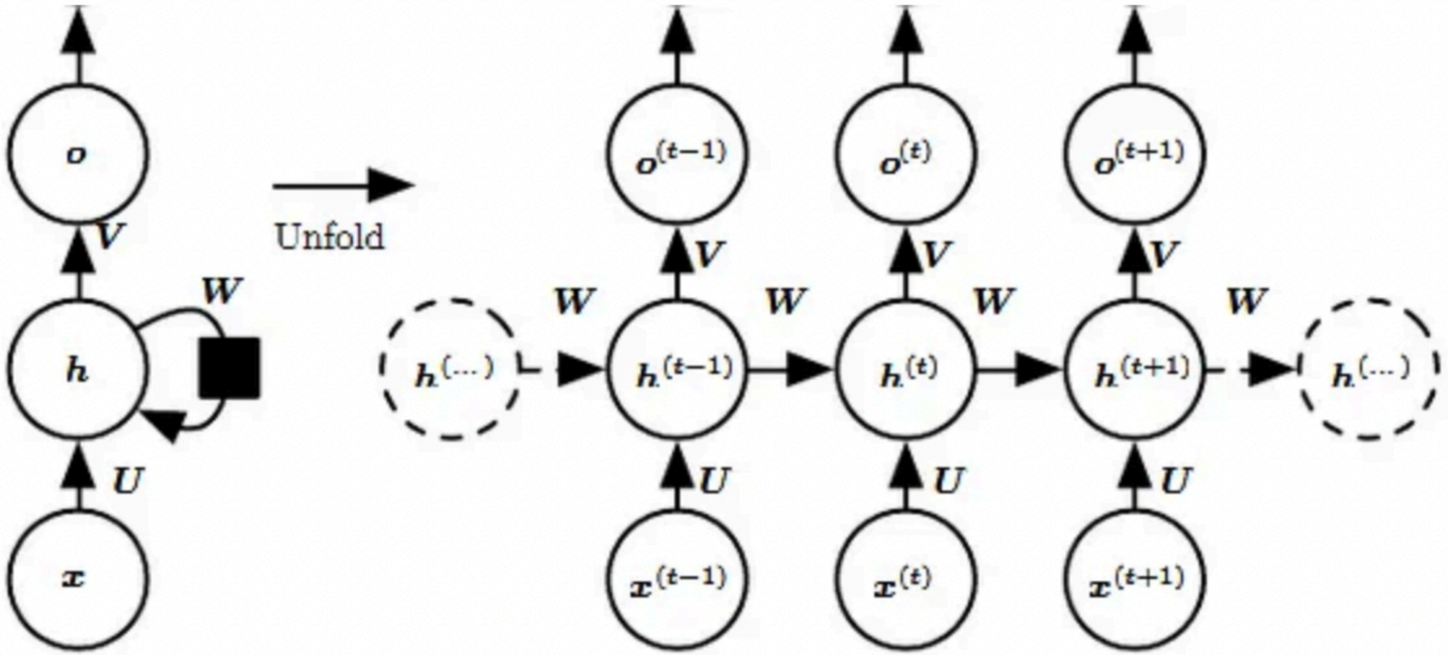
FEEDBACK/RECURRENT NEURAL NETWORK

- Feedforward neural networks allow the passage of data in one direction only.
- However, when the data is of sequential nature, we need to look at previous inputs and steps to better process the current input. This sequential nature is captured using a recurrent/ feedback neural network.
- A feedback neural network will essentially have “loops” which allow us to store processed values as a “hidden state”.
- A good example–
  - This stock is enjoying a good run.
  - I run daily.
- These two sentences clearly use the word ‘run’ in very different contexts. Sequential analysis is required to differentiate between the two.
- Thus, in feedforward neural networks we are working on fixed sized input and producing fixed sized output, while RNNs are able to adapt to inputs of varying sizes.
- applications of RNN–
  - sentence semantics detection
  - image captioning
  - image/ text emotion detection
  - time series analysis for stock prediction

DIFFERENT PARADIGMS OF RNN



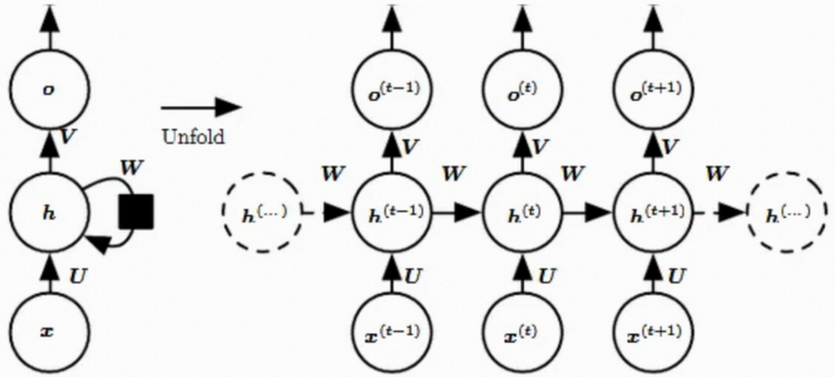
LETS BUILD A BASIC RNN!



Reference

- This RNN contains a single unit in the hidden layer, single unit input and output being produced in every time unit.
- In every time step, the following is done–
  - the value of input unit is combined with the value of the hidden state unit of the previous time unit using a linear function.
  - The result of previous computation is made to pass through an activation function.This value will then become the new hidden state of the RNN.
  - This is then used as the final input for prediction, where it is processed using weights and then activated using an activation function.
- This is the ‘forward pass’ of the RNN.

$$
\begin{aligned}
\boldsymbol{a}^{(t)} &= \boldsymbol{b} + \boldsymbol{W}\boldsymbol{h}^{(t-1)} + \boldsymbol{U}\boldsymbol{x}^{(t)} \\
\boldsymbol{h}^{(t)} &= \tanh(\boldsymbol{a}^{(t)}) \\
\boldsymbol{o}^{(t)} &= \boldsymbol{c} + \boldsymbol{V}\boldsymbol{h}^{(t)} \\
\hat{\boldsymbol{y}}^{(t)} &= \operatorname{softmax}(\boldsymbol{o}^{(t)})
\end{aligned}
$$



Reference

BACKPROPAGATION THROUGH TIME IN RNN

- when back-propagation is carried out in an RNN, we need to take into consideration the effect that previous time steps are having on the current value of error.
- Here Y3 is the the value obtained after actiavtion function is applied on the transformation of Wy S3
- Error function E3 will be a function of Y3 and hence a function of Wy

ADJUSTING WY:

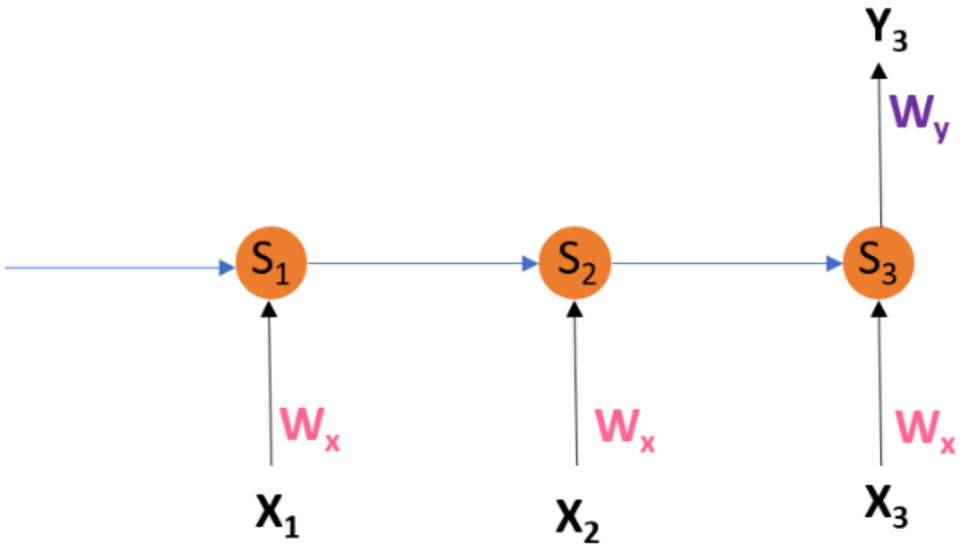
$$\frac{\partial E_3}{\partial W_y} = \frac{\partial E_3}{\partial Y_3} \cdot \frac{\partial Y_3}{\partial W_Y}$$

ADJUSTING WS:

$$\frac{\partial E_3}{\partial W_S} = \left(\frac{\partial E_3}{\partial Y_3} \cdot \frac{\partial Y_3}{\partial S_3} \cdot \frac{\partial S_3}{\partial W_S}\right) +$$

$$\left(\frac{\partial E_3}{\partial Y_3} \cdot \frac{\partial Y_3}{\partial S_3} \cdot \frac{\partial S_3}{\partial S_2} \cdot \frac{\partial S_2}{\partial W_S}\right) +$$

$$\left(\frac{\partial E_3}{\partial Y_3} \cdot \frac{\partial Y_3}{\partial S_3} \cdot \frac{\partial S_3}{\partial S_2} \cdot \frac{\partial S_2}{\partial S_1} \cdot \frac{\partial S_1}{\partial W_S}\right)$$



ADJUSTING WH:

$$\frac{\partial E_3}{\partial W_X} = (\frac{\partial E_3}{\partial Y_3} \cdot \frac{\partial Y_3}{\partial S_3} \cdot \frac{\partial S_3}{\partial W_X}) +$$

$$(\frac{\partial E_3}{\partial Y_3} \cdot \frac{\partial Y_3}{\partial S_3} \cdot \frac{\partial S_3}{\partial S_2} \cdot \frac{\partial S_2}{\partial W_X}) +$$

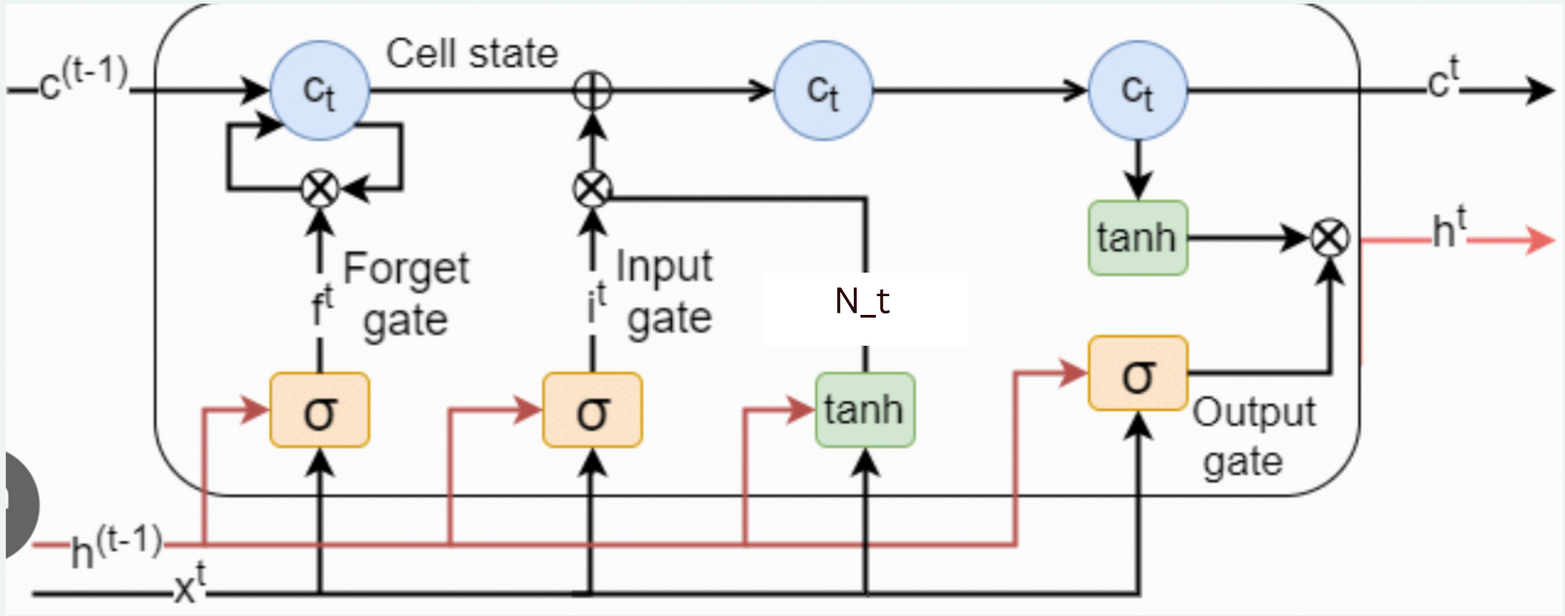
$$(\frac{\partial E_3}{\partial Y_3} \cdot \frac{\partial Y_3}{\partial S_3} \cdot \frac{\partial S_3}{\partial S_2} \cdot \frac{\partial S_2}{\partial S_1} \cdot \frac{\partial S_1}{\partial W_X})$$

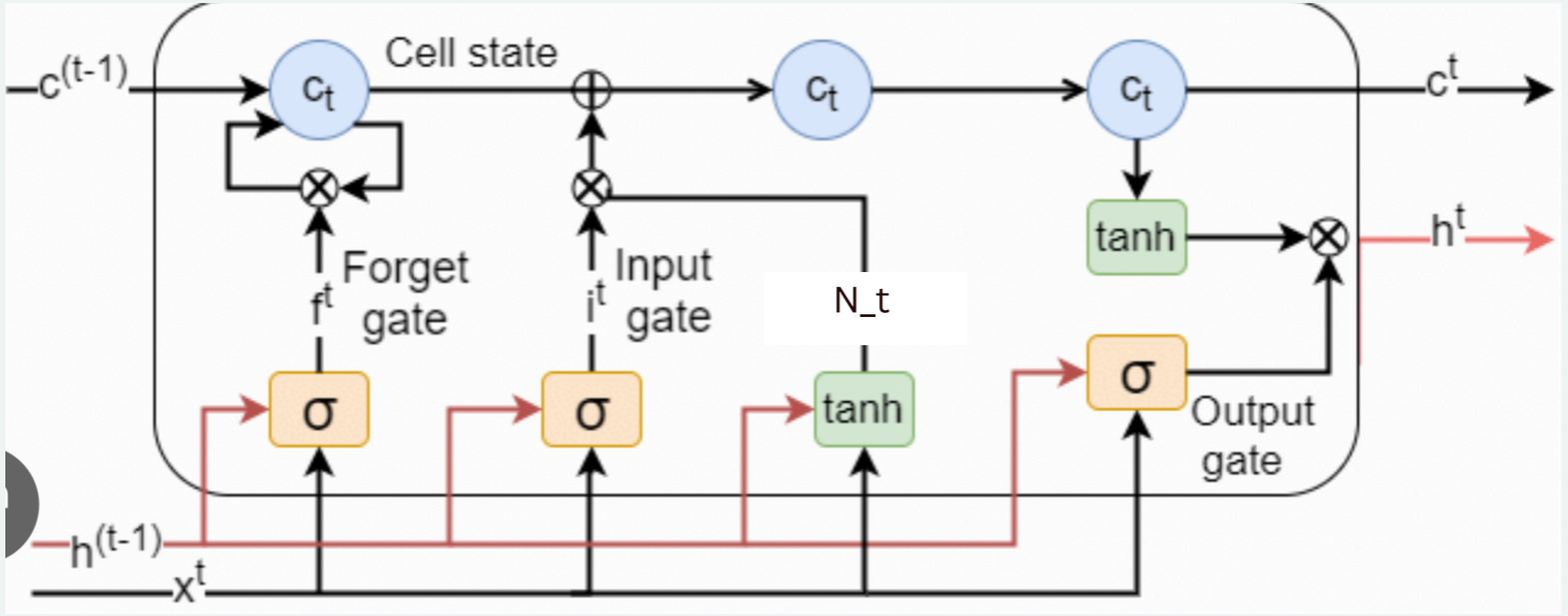
https://www.geeksforgeeks.org/ml-back-propagation-through-time/

- As is clearly visible, updating every weight leads to a product of many gradients.
- if long time intervals are to be considered, a large no of gradients will get accumulated, leading to extremely small gradients and the model will then not get updated much, regardless of the no. of training epochs.
- This is referred to as the **“vanishing gradient problem”** and is the main hinderance in the training of RNNs that have long-term memory.
- sometimes when the weights are initialised with extremely large values, then the gradients can get accumulated over time and become extremely large, as a result of which model is not fruitfully updated. This is the **“exploding gradient problem”**. This can be rectified by limiting the value of the gradient to a predefined maximum.

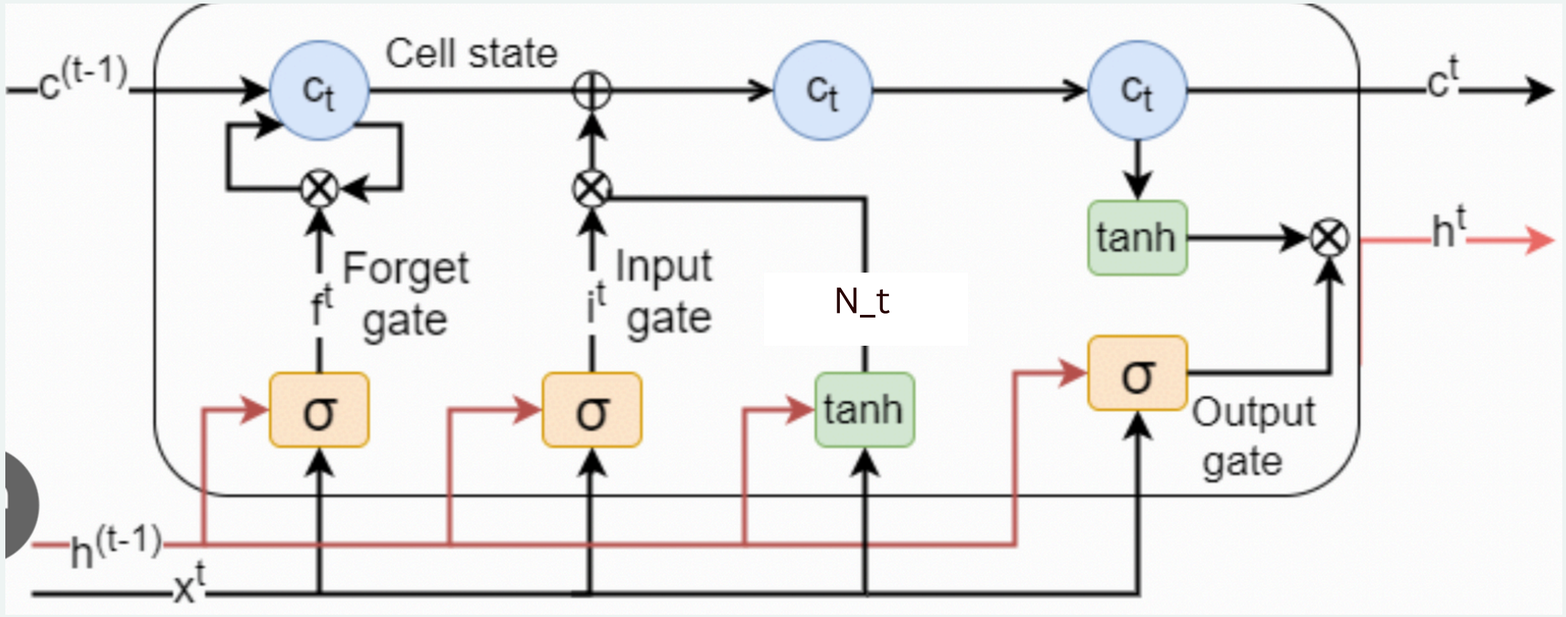
LONG-SHORT TERM MEMORY

- LSTMs were designed to overcome the issues that are commonly faced due to the vanishing gradient problem in traditional RNNs.
- LSTMs do this by selectively storing important information over long periods of time.
- an LSTM cell will differ from a traditional RNN cell by not only storing the pervious step's **"short term hidden state"**, but also a **"long term hidden state".**





- The LSTM cell will be divided into three parts or gates.
- **Forget gate**
  - this is responsible for checking if the long-term memory is relevant on the basis of the short-term memory and the current input.
- **Input Gate**
  - This is responsible for producing the new long-term memory taking into account the relevance of the previous long-term memory(as calculated by the forget gate), and combining it with the short-term memory as well as the current input.
- **output gate**
  - this is responsible for using the computed long-term memory, the short-term memory as well as the current input to produce the new short-term memory. An actiavtion of this newly computed short-term memory is used as output for the LSTM cell.



Forget Gate:

- $f_t = \sigma(x_t * U_f + H_{t-1} * W_f)$

Input Gate:

- $i_t = \sigma(x_t * U_i + H_{t-1} * W_i)$

Output Gate:

- $o_t = \sigma(x_t * U_o + H_{t-1} * W_o)$

- $N_t = \tanh(x_t * U_c + H_{t-1} * W_c)$ (new information)

$H_t = o_t * \tanh(C_t)$

$C_t = f_t * C_{t-1} + i_t * N_t$ (updating cell state)

$\text{Output} = \text{Softmax}(H_t)$

HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/2021/03/INTRODUCTION-TO-LONG-SHORT-TERM-MEMORY-LSTM/