Optimation in Deep Learning

- In Deep Learning, with the help of loss function, the performance of the model evaluated.
- This loss is used to train the network so that it performs Better.
- Essentially, we try to minimize the loss function.
- Lower loss means the model performs better
- The process of minimizing any mathematical function is called optimization.
- Optimers are algorithms used to change the features of neural network such as weight and learning rate so that the loss is reduced
- The Goal of an optimizer is to minimize the objective function.

Need for optimization

- ~~Presence~~ Prescence of Local minima reduces the model performance
- To minimize the loss value (Training error)
- To select appropriate weight values and other associated model parameters.

Types of optimization

1. Gradient descent

- it starts with some coefficients seen

- it moves towards lower weight and updates the values of coefficient and repeat until the local min is reach

Disadvantages

- Expensive to calculate a gradient if the size of the data is huge
- alot suitable for Non-Convex function

2. Stochastic Gradient Descent :

- Instead of taking the whole data set for each iteration randomly select batches of the data
- Select the initial parameter w and learning data
- Randomly shuffle the data and each iteration to reach the approximate minimum.

fast is than GD but the Computation

- Since only few batches are

Dis advantage :

3. Stochastic Gradient descent with Momentum :

→ Since SGD is a noisy path we are going for SGD with momentum

→ momentum helps in fast Convergence of the loss function

→ SGD oscillates b/w either direction of the gradient and update the weight.

→ By adding the fraction of the previous update to the current update will make the process a bit faster.

4. Mini batch gradient descent :

→ only a subset of the dataset is used for calculating the loss function

→ It takes only fewer iterations so faster than SGD

→ it is smoother than SGD

→ it has good balance b/w speed and accuracy

5. Adagrad Adaptive gradient descent

→ it uses different learning rates for each iteration

→ The change in the learning rate depends upon the difference in the parameters unit training.

and

Scanned with CamScanner

Recurrent Neural Networks

- RNNs are very powerful, because they combine two properties

1. Destributed hidden state that allows them to store a lot of information about the past efficiency
2. Non-linear dynamics that allows them to update their hidden state in complicated ways.

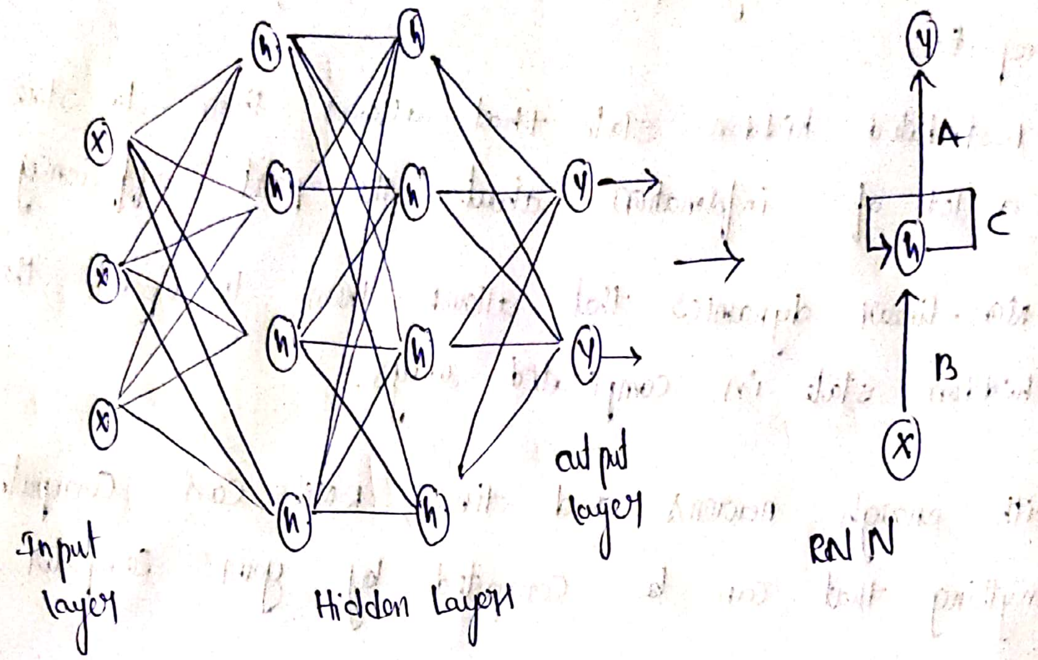
- with enough neurons and time, RNNs can compute anything that can be computed by your computer.

Need for RNN:

- Normal Networks cannot handle sequential data
- They considers only the current input
- Normal NN cannot memorize previous inputs

- RNN works on the principle of saving the output of a particular layer and feeding this back to the input in order to predict the output of the layer.

Converting a Full Network into Recurrent Network

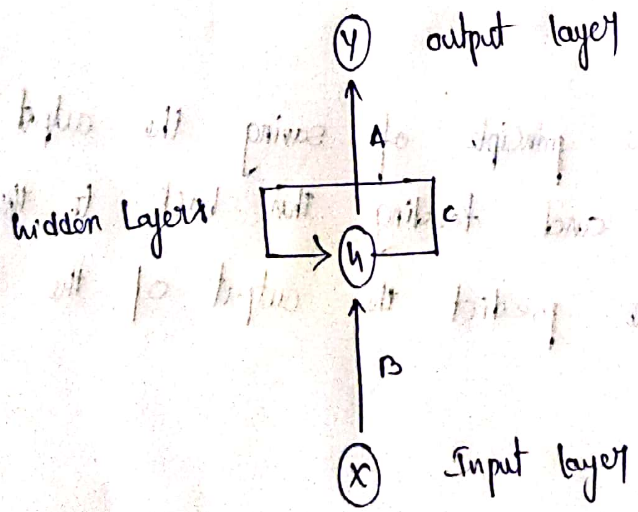


X - input layer

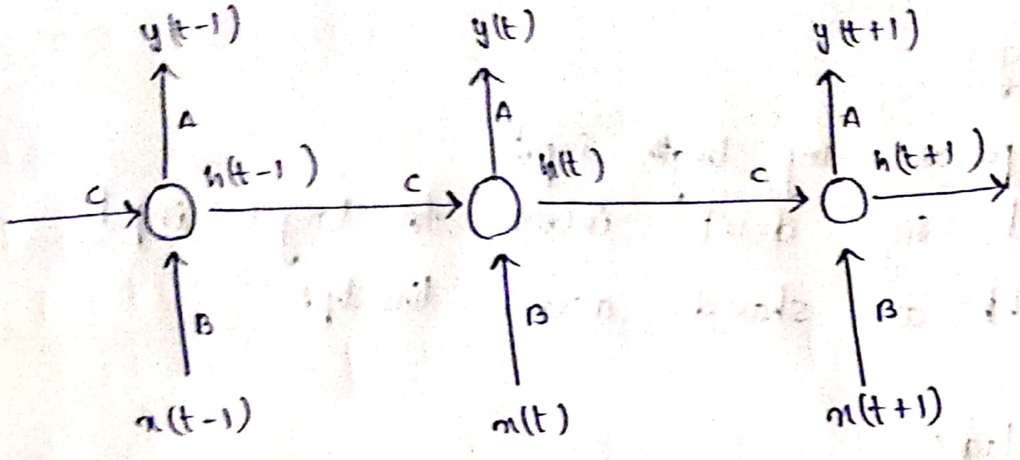
h - hidden layer

Y - output layer

A, B, C are the network parameters



A) Recurrent Network



B) Fully Connected RNN

$$h(t) = f_c(h(t-1), x(t))$$

$h(t)$ = new state

f_c = function with parameter c

$h(t-1)$ = old state

$x(t)$ = input vector at time step t

Providing Input to RNN

- Specify the initial states of all the units
- Specify the initial states of a subset of the units
- Specify the states of the same subset of the units at every time step.

Providing Target to RNN

- Specify desired final activities of all the units
- Specify good for learning attractors
- Specify the desired activity of a subset of the units.

Advantages

- Can process any length input
- Model size doesn't increase for longer input
- weights are shared accross timesteps

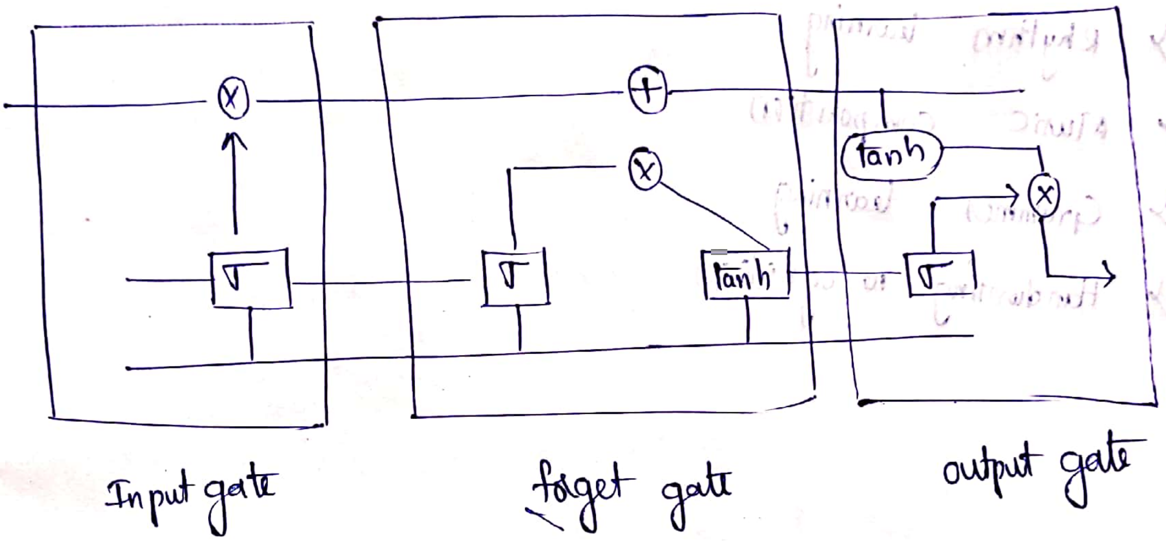
Disadvantages

- Recurrent computation is slow
- In practice, difficult to access information from many steps back.

Scanned with CamScanner

LSTMs – Long short Term Memory Network

* A type of RNN architecture that addresses the vanishing gradient problem and allow learning of long-term dependencies



σ – used to remaining unwanted data

tanh – used to add the additional information

<u>forget gate</u> : Controls what information to throw away from memory

$$f_t = \sigma(w_f[h_{t-1}, x_t] + b_f)$$

<u>Input gate</u> : controls what new information is added to cell state from current input

$$i_t = \sigma(w_i[h_{t-1}, x_t] + b_i)$$

$$\tilde{c}_t = \tanh(w_c[h_{t-1}, x_t] + b_c)$$

<u>Output gate</u> :

$$o_t = \sigma(w_o[h_{t-1}, x_t], b_o)$$

$$h_t = o_t * \tanh(c_t)$$

Applications of LSTM

- Robot Control
- Time series prediction
- speech recognition
- Rhythm learning
- Music composition
- Grammer learning
- Handwriting recognition