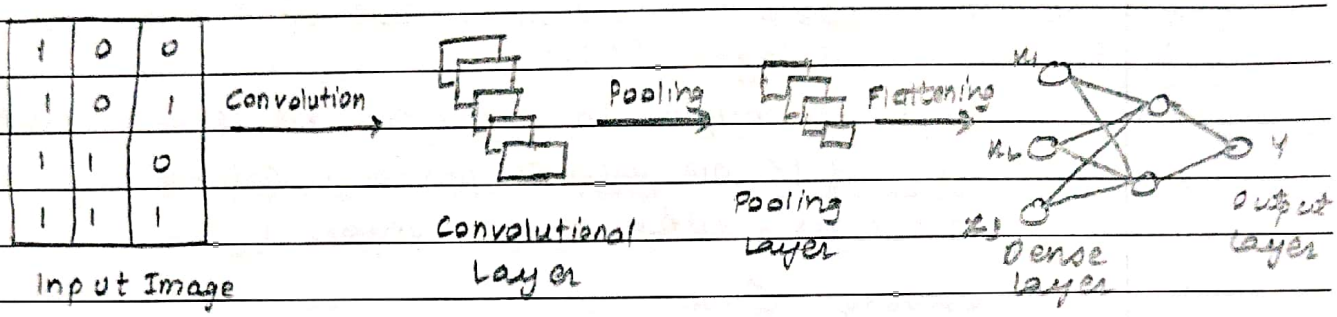
RAJIV GANDHI INSTITUTE OF TECHNOLOGY, MUMBAI

JUHU VERSOVA LINK ROAD, VERSOVA, ANDHERI (WEST), MUMBAI - 53.

DEEP LEARNING

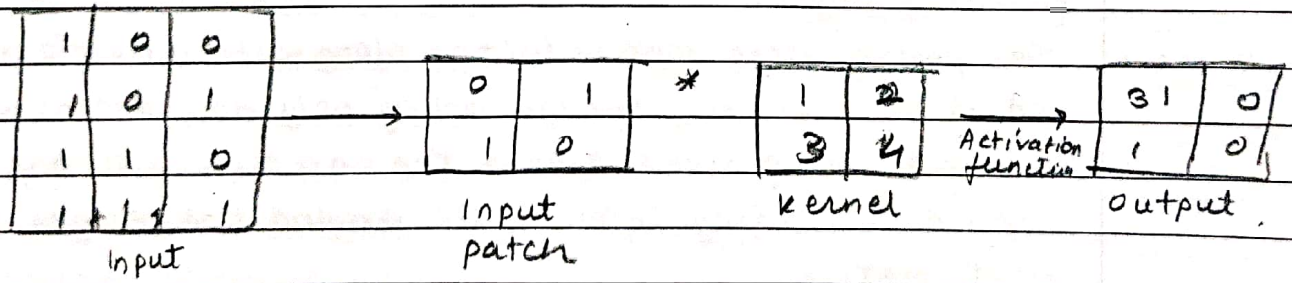
CHAPTER 3 :- CONVOLUTION NEURAL NETWORK

* CNN Architecture :-



1 Convolutional Layer :-

- Foundation of CNN, responsible for executing convolution operations.
- Kernel / Filter component performs convolution.
- Until whole image is scanned, kernel makes adjustments horizontally and vertically until according to its stride rate. Stride rate is the number of steps taken by a kernel.



2. Pooling Layer :-

- Helps reduce dimensionality, saves computational complexity.
- Pooling can be max pooling or average pooling.
- Max pooling :- Takes max value from a kernel.
- Average pooling :- Takes average of all values from a kernel.

MANJARA CHARITABLE TRUST
RAJIV GANDHI INSTITUTE OF TECHNOLOGY, MUMBAI
JUHU VERSOVA LINK ROAD, VERSOVA, ANDHERI (WEST), MUMBAI - 53.

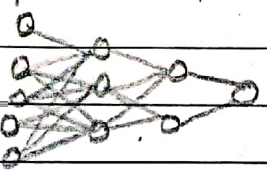
5	6	1	2
7	8	3	4
12	6	5	6
3	2	8	10

Max Pooling →

8	4
12	10

3. Fully connected layer (FC):-

- Every input is coupled to a neuron, this is called flattened input.
- Next layers of FC are used to perform calculations on input data, the classification process ~~began~~ begins.



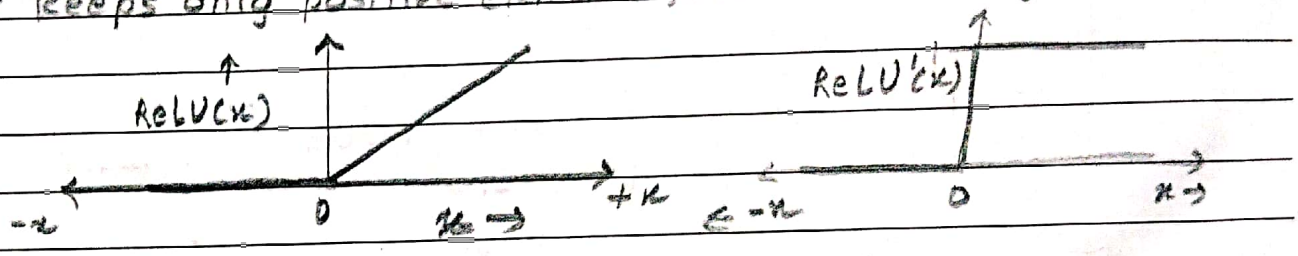
- Additional terms :-
- Last FCL's activation function is always distinct from others.
- Each activity selects appropriate activation function.
- Dropout layers :-
- It is a mask that nullifies some neuron's contribution to the following layer.
- Max pooling does noise reduction along with dimensionality reduction, whereas average pooling only does dimensionality.
- To avoid losing pixels around the corners we simply do padding which is adding extra pixels around the edges of the input matrix.
- The amount of sliding we do across our input tensor is called as stride.

* ReLU :-
- It is activation function that decides whether a neuron

should be fired or not by the weighted sum & input bias.
- Due to its simplity and performance ReLU is used alot.
- Stands for rectified Linear unit, provides non-linear transformation. Given an element x, ReLU is defined as:

$$ReLU = \max(x, 0)$$

- It keeps only positive elements & discards all negative elements.



- ReLU avoids vanishing gradient descent problem.
- Dropout layers help avoid overfitting on the training data.

* Interleaving :-
- Convolution, Pooling & ReLU layers are interleaved in order to express power.
- Mostly, ReLU follows Convolutional layers.
- Interleaved means they are alternated in between.
- Let C be Conv., P be Pooling, R be ReLU :-
  One interleaving can be :- CRCRP or CRCRCRP

* Local Response Normalization :-
- Unbounded activation functions require normalization.
- Normalization is applied before activation functions.
- Local response normalization is used in CNN for normalization.
- Was first introduced in AlexNet
- Introduces lateral inhibition increases competition among neurons.
- This helps to normalize response and scale them for a particular neuron based on the activities of neighbouring neurons.

UNIT 4 :- Recurrent & Recursive Nets

* Unfolding computational Graphs :-
- Computational Graphs are a way to formalize structure of a set of computations.
- We unfold a recursive or a recurrent function that has a repetitive structure.
- Example of unfolding recurrent equation :-

$$s(t) = f(s(t-1); \theta) \quad \text{or} \quad s^{(t)} = f(s^{(t-1)}; \theta)$$

It is recurrent as current state $s^{(t)}$ depends on previous $s^{(t-1)}$.
- $s^{(t)}$ is called state of the system.
- For a finite numbes of steps τ the graph can be unfolded by applying the definition $\tau - 1$ times.

E.g :- For $\tau = 3$, we have

$$s^{(3)} = f(s^{(2)}; \theta) = f(f(s^{(1)}; \theta); \theta) \quad \ldots (i)$$

- $s^{(1)}$ is called ground state.
- A dynamical system given by $s^{(t)} = f(s^{(t-1)}; \theta)$ & $s^{(3)} = f(f(s^{(1)}; \theta); \theta)$ can be given by an acyclic unfolded computational graph as :-

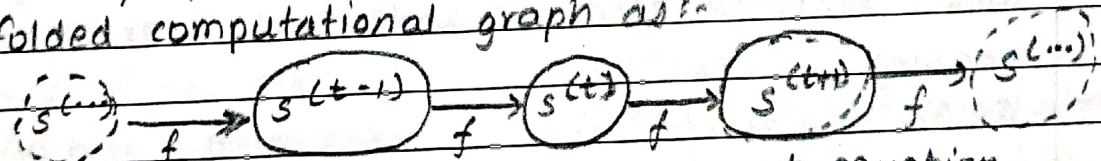
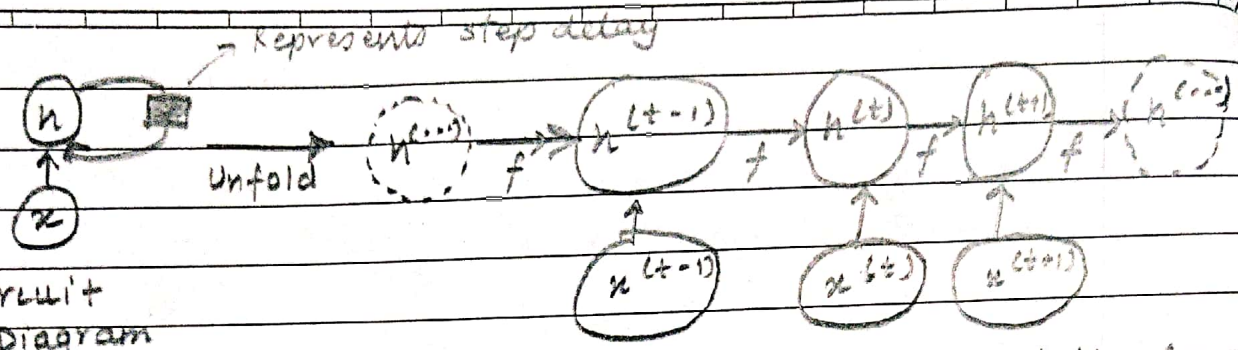


Fig : Unfolding recurrent equation,

- Each node represents a state at time, function f maps state at a time t to t+1, and θ is used as a parameter.
- We can add a dynamic system driven by external signal $x^{(t)}$ as $s^{(t)} = f(s^{(t-1)}, x^{(t)}; \theta) \quad \ldots (ii)$
- Because of $x^{(t)}$ state now contains whole past data.
- We use equation (ii) to define hidden units as :-

$$h^{(t)} = f(h^{(t-1)}; x^{(t)}; \theta)$$

MANJARA CHARITABLE TRUST
RAJIV GANDHI INSTITUTE OF TECHNOLOGY, MUMBA
JUHU VERSOVA LINK ROAD, VERSOVA, ANDHERI (WEST), MUMBAI - 53.

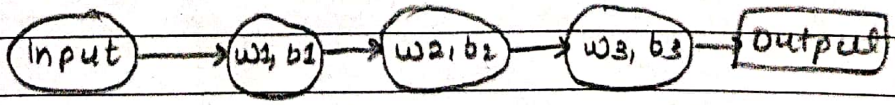


• Advantages of unfolding:-

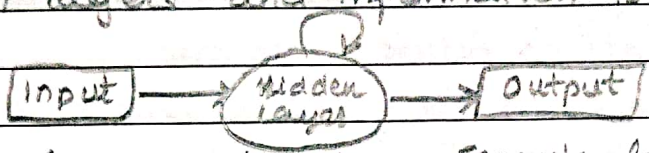
1) Regardless of sequence length, learned model has same input size.
- This is because it is specified in terms of variable length transition from one state to another and not in variable length history of state.
2) Possible to use same function with same parameters.

* Recurrent Neural Network:-
- A type of neural network in which results of one step are feeded into next step's computation.
- RNNs remember remember previous inputs which is sometimes required to generate output for example in NLP the whole sentence needs to understood in order to get the meaning.
- CNNs are useful when working with a grid of input (Images) but they cannot work on temporal data which RNNs overcome.
- So RNNs work on sequence of data $x^{(1)}, \ldots, x^{(t)}$

• Working of RNN:-
- Consider neural network with 3 hidden layers, one i/o layers.



- All three layers have different weights and biases.
- Each layer is independent of others and does not retain info.
- Now, RNN gives same weights & biases creating dependency between layers and information is saved.



Formula for current state:- $h^{(t)} = f(h^{(t-1)}, x^{(t)})$

$h^{(t)}$ → Current

$h^{(t-1)}$ → Previous

$x^{(t)}$ → Input state

Formula for Activation function $h^{(t)} = \tanh(W_{hh} h^{(t-1)} + W_{xh} x^{(t)})$

W_{xh} → Weight of input layer

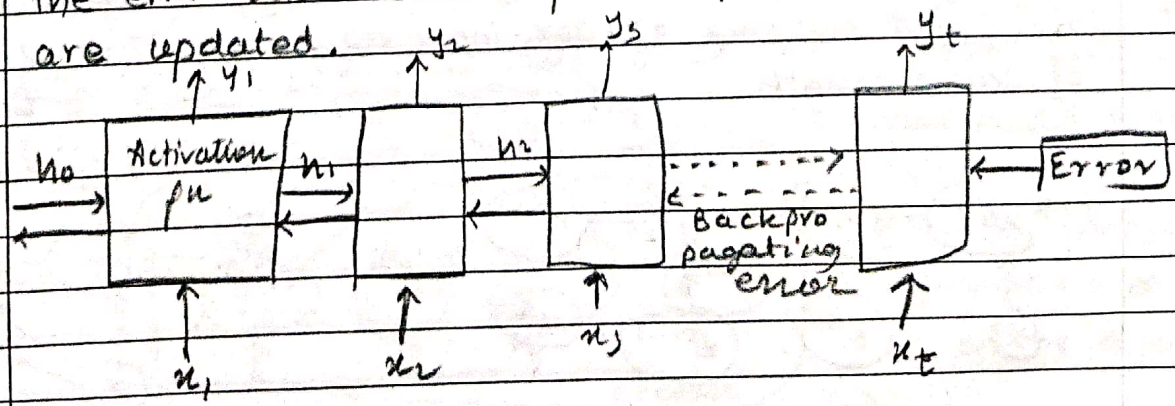
W_{hh} → —"— Hidden —"—

Output :- $y^{(t)} = W_{yh} \cdot h^{(t)}$

$y^{(t)}$ → Output W_{yh} → Weight at output layer

Training RNN :-

- Input is given in sequence.
- Using ~~current~~ prior state and current input, calculate present input.
- Error Final state gives the output.
- Difference between output & actual output is calculated.
- The error value is used for backpropagation & weights are updated.

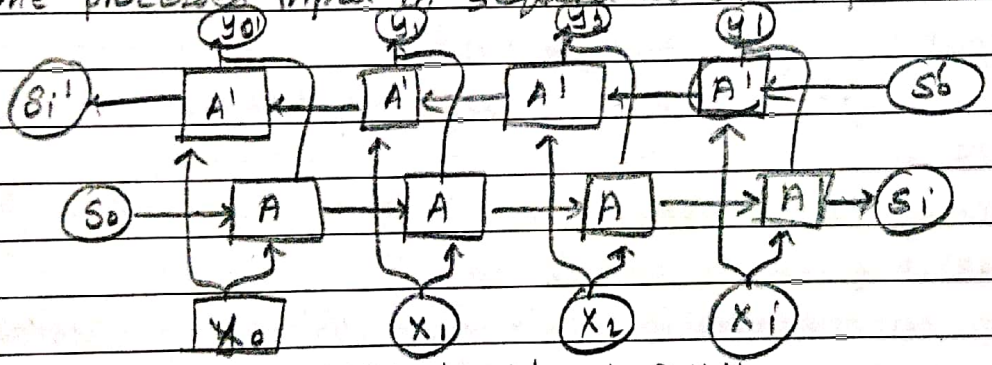


MANJARA CHARITABLE TRUST
RAJIV GANDHI INSTITUTE OF TECHNOLOGY, MUMBA
JUHU VERSOVA LINK ROAD, VERSOVA, ANDHERI (WEST), MUMBAI - 53.

- RNNs face the problems of vanishing & exploding gradients.

* Bidirectional RNNs :-

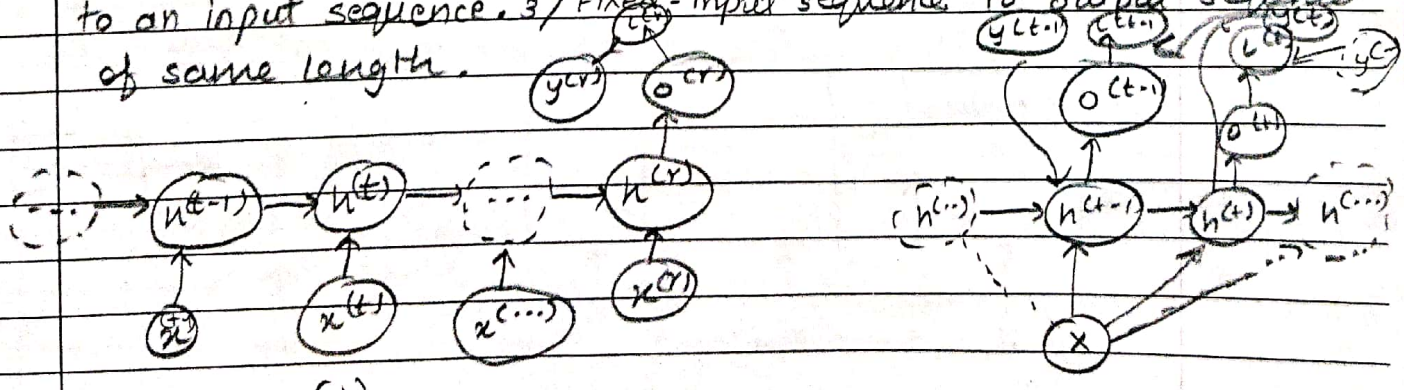
- In speech recognition, correct interpretation may depend on past as well as future phonemes.
- Handwritting recognition also requires to know next letters.
- Combine RNN that moves forward through time starting from the beginning and another that starts from the end towards the beginning.
- Bidirectional RNNs consists of such two RNNs stacked upon each other.
- One processes input in sequential order & other in reverse.



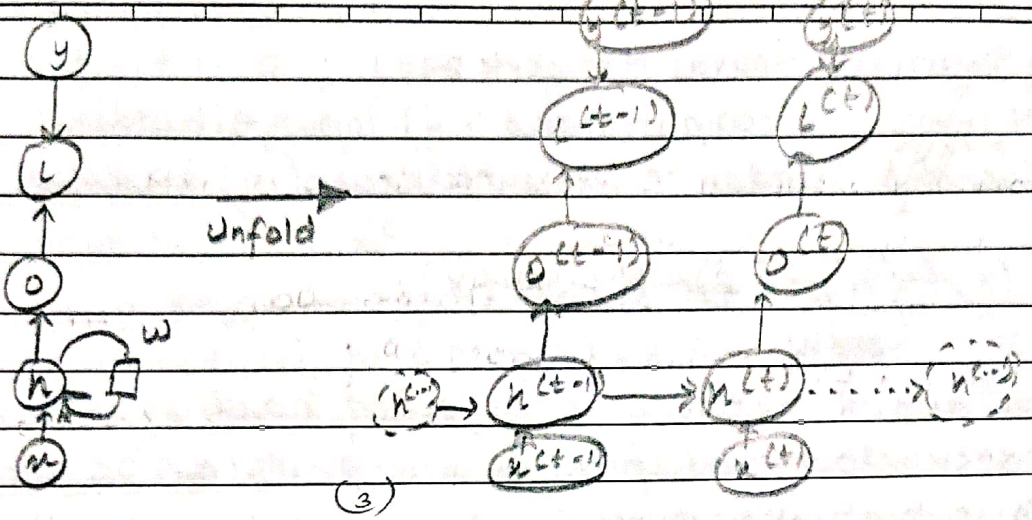
Bidirectional RNN

- There are 3 designs depending on input/output vector lengths:

1) Input sequence to a fixed-size vector. 2) Fixed-size vector to an input sequence. 3) Fixed- input sequence to output sequence of same length.



(1)



(3)

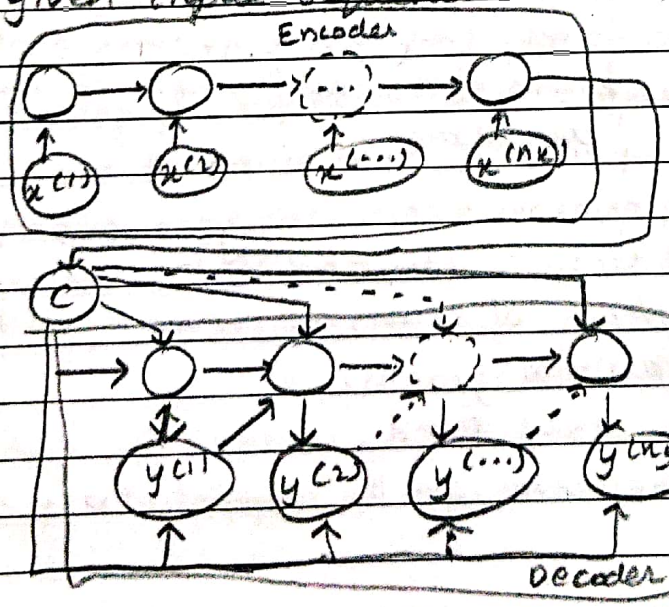
(1) Draw sequence of x, last hidden layer goes to output, output and y both go to loss.

(2) Fixed size min single x goes to all hidden, all hidden have their loss, all y connect to prev. loss & current hidden.

(3) Typical RNN.

* Encoder-Decoder Model or Sequence-to-Sequence RNNs.

- Used when input & outputs are not of the same length.
- Learns to generate output sequence :- $(y^{(1)}, \ldots, y^{(n_y)})$ given input sequence :- $(x^{(1)}, \ldots, x^{(n_x)})$.

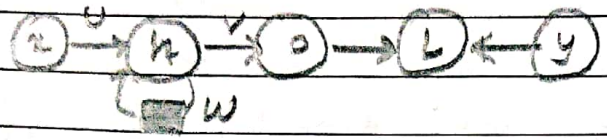


- Encoder :- Reads input seq.
- Decoder :- Generates output seq. or probability of a given output sequence.
- Final hidden state computes context C.
- C holds semantic information of input.
- It can be a parse tree.

MANJARA CHARITABLE TRUST
RAJIV GANDHI INSTITUTE OF TECHNOLOGY, MUMBA
JUHU VERSOVA LINK ROAD, VERSOVA, ANDHERI (WEST), MUMBAI - 53.

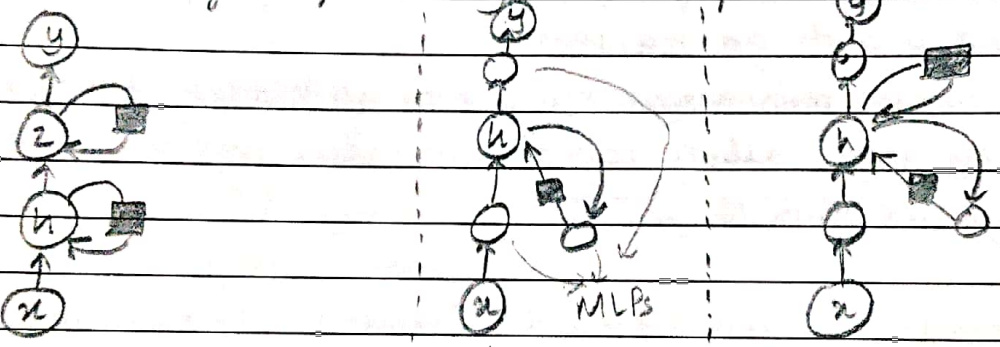
* Deep Recurrent Neural Network :-

- RNN involves 3 computations :- i) Input → hidden, ii) ~~Current~~ Previous hidden → Current hidden, (iii) hidden → Output.



- When above network is unfolded each of it corresponds to a shallow transformations i.e it can be represented as a single layer in MLP.

- Three ways of Making RNN Deep :-



- Hidden recurrent states broken down in groups organized hierarchially	-Add more computation in input-hidden, hidden-hidden, hidden-output.	-Lengthen path by adding skip connections.
- Increases complexity of input representation	- We add deep MLPs after each computation.	- Between any two variables we have doubled the length of shortest path.

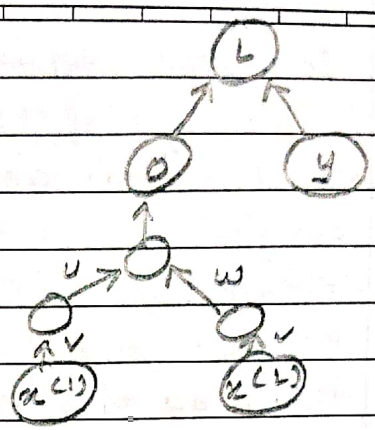
* Recursive Neural Networks :- [RVNN]

- Another generalization of recurrent network with different computational graphs.
- Structured as a deep tree.
- Variable input sequence can be mapped to a fixed size output.

RAJIV GANDHI INSTITUTE OF TECHNOLOGY, MUMBAI

JUHU VERSOVA LINK ROAD, VERSOVA, ANDHERI (WEST), MUMBAI - 53.

- Output :- O ; Loss :- L ;
  Actual Output :- y.
  Parameters :- V, W, U
  Input :- $x^{(1)}$; $x^{(2)}$



- For a sequence of length T the depth reduces to O(log T), helps in long-term dependencies.
- ~~Recurrent~~ RNNs are more suited to time-series data as they don't consider linguistic structure aside from word order.
- RvNNs on other hand were born to model NLP by syntatic parse trees.
- RvNNs don't take tokens sequentially, recursive models combine neighbors.
- RvNNs AKA Tree Nets.

* Long-Term dependencies :-

- Optimization become difficult when computational graphs become deep, like deep feed-forward networks or RNNs working on long ~~time~~ temporal sequence.
- Gradients propagated may vanish or explode.
- Vanishing gradients make it difficult to know which direction to move to reduce cost.
- Exploding gradients make learning unstable.

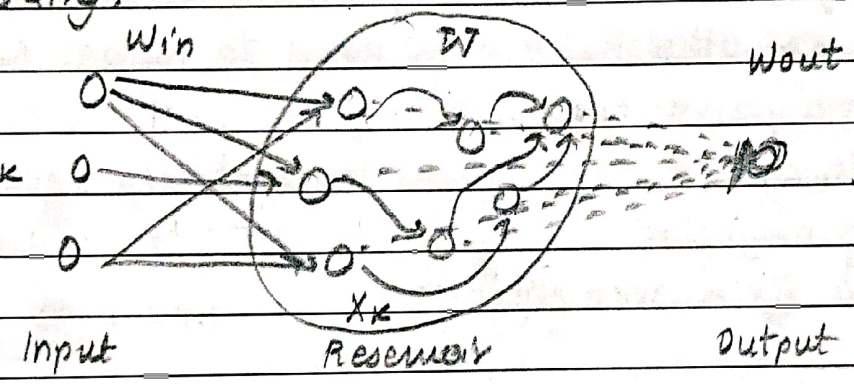
Consider a computational graph consisting of multiplying by W after t steps we have W^t. Its eigen decomposition is

$$W^t = (V \, diag(\lambda) \, V^{-1})^t = V \, diag(\lambda)^t V^{-1}$$

Any eigenvalues λ, that are not close to absolute value of 1 will explode if > 1 and vanish if < 1.

MANJARA CHARITABLE TRUST
RAJIV GANDHI INSTITUTE OF TECHNOLOGY, MUMBA
JUHU VERSOVA LINK ROAD, VERSOVA, ANDHERI (WEST), MUMBAI - 53.

* Echo State Networks :-
- It is a type of RNN associated with reservoir computing.
- Input-Hidden and Hidden weights are non-trainable & randomly assigned, weights of output neuron are trainable.
- Hidden layer or Reservoir is sparsely connected.
- Reservoir computing is an extension of NNs in which input is connected to a non-trainable & random dynamic system called the reservoir, which creates high-dimension display called embedding.

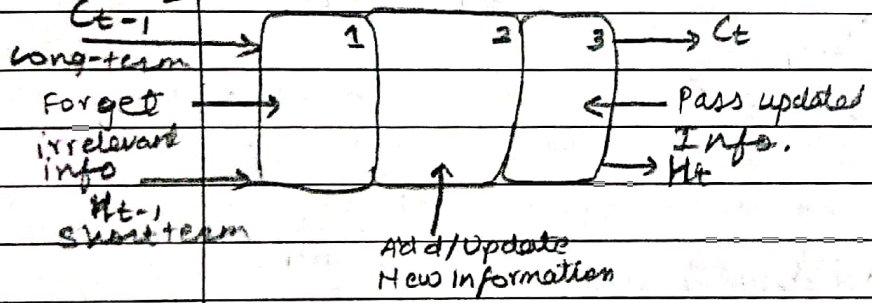


* Leaky Units & strategies for multiple time scales.
. Multiple time scales :-
- To design a model that deals with long-term dependencies :-
i) Design it to deal with fine-grained time so for some parts scales to handle details (ii) other parts deal with coarse time scales.
- so some operate at fine & others at coarse level.
- Strategy to do so :-
1) Adding skip connection through time :-
- Gradients explode wrt time steps (T)
- Introduce time delay (d) now gradient diminish as a function of T/d rather than T.
- This helps capture long term dependencies.

2) Leaky units & spectrum of time scales :-
- Don't use integer skip of d instead use real-valued α.
- Consider $u^{(t)}$ as running average of $v^{(t)}$ as $u^{(t)} \leftarrow \alpha u^{(t)} + (1-\alpha) v^{(t)}$
- This is called linear self-connection corrections.
- Hidden layers with linear self-connections are leaky units.

3) Removing Connections :-
- Create Create states of RNN at multiple time scales.
- Create longer connection remove length one connections.

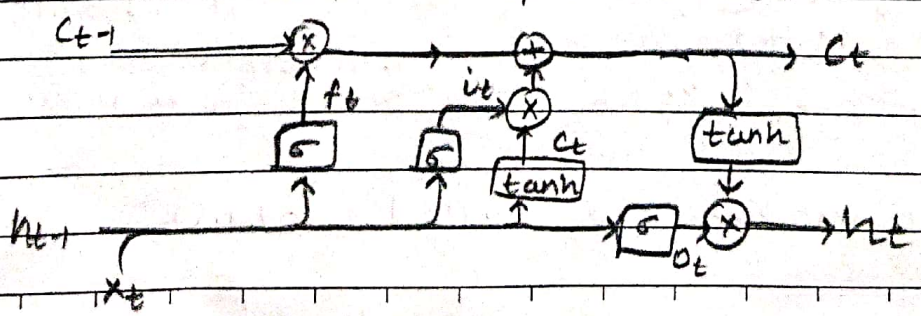
* LSTM & Other Gated RNNs :-
- These solve vanishing & exploding gradients problem.



• Forget Gate :
$f_t = \sigma(x_t \times U_f + H_{t-1} W_f)$
Input, Input wt., Prev. hidden state, Hidden weights.
If $C_{t-1} \times f_t = 0$ forget if 1 then rem.

• Input Gate :-
$i_t = \sigma(x_t \times U_i + H_{t-1} W_i)$
Input weight
New Info : $\tanh(i_t)$
$H_t = \tanh(x_t \times U_c + H_{t-1} \times W_c)$
$\therefore C_t = (f_t \times C_{t-1} + i_t \times H_t)$

• Output Gate :-
$O_t = \sigma(x_t \times U_o + H_{t-1} \times W_o)$; $H_t = O_t \times \tanh(C_t)$
Output = Softmax (H_t)



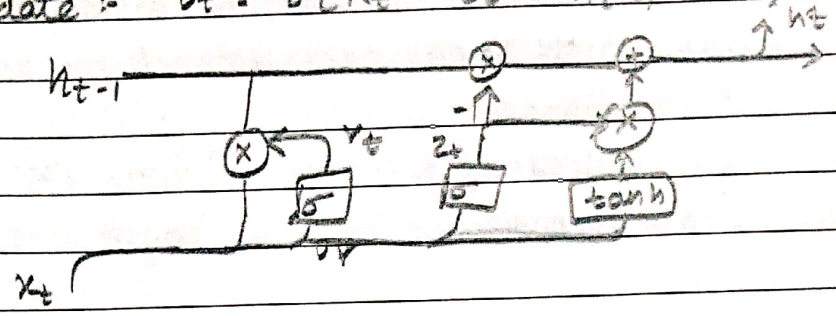
MANJARA CHARITABLE TRUST
RAJIV GANDHI INSTITUTE OF TECHNOLOGY, MUMBAI
JUHU VERSOVA LINK ROAD, VERSOVA, ANDHERI (WEST), MUMBAI - 53.

* GRUs :-

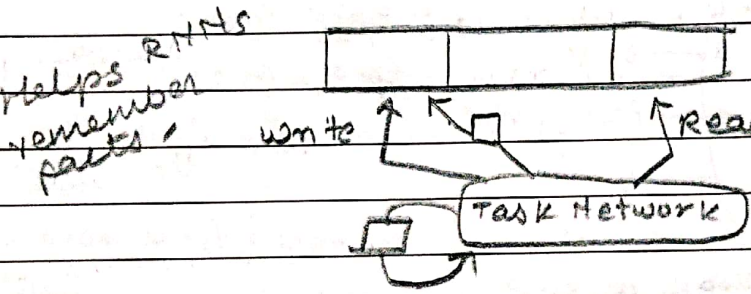
Gated Recurrent Units, has only two gates :-

i) Reset :- $r_t = \sigma(x_t * U_r + H_{t-1} * W_f)$ forgets

ii) Update :- $U_t = \sigma(x_t * U_u + H_{t-1} * W_U)$ Update



* Explicit Memory :-



Explicit Memory.
Neural Turing Machine.

* Performance Metrics :-

- ~~Error~~ Regression :- Squared error, RMS

Squared error = $\frac{\Sigma(y - \hat{y})^2}{n}$ RMS = $\sqrt{\frac{\Sigma(y - \hat{y})^2}{n}}$

- Classification :- Accuracy = $\frac{\text{Number of correct}}{\text{Total correct.}}$

Precision = $\frac{TP}{TP + FP}$ Recall = $\frac{TP}{TP + FN}$ = $\frac{\text{Predicted Positives}}{\text{Actual Positives}}$

P = $\frac{\text{Predicted Positives}}{\text{Total Positives predicted}}$

- Density estimations :- $KL(p||q) = -\int p(x) \ln\left\{\frac{p(x)}{q(x)}\right\} dx$

↓ Difference in probability distributions.

- Information Retrieval :- Precision and Recall.

$F = \frac{2PR}{P+R}$ Harmonic Mean [F1 - Score]

- Metric for Image Segmentation :-

Dice Coefficient :- $dice(X, Y) = \frac{2 \times X \cap Y}{X + Y}$

Overlap between predicted & true positive region

* Baseline Approaches :-

- To build an end-to-end machine learning system we must choose architecture, optimizations and regularizations.

- Architecture depends on input data :-

1) Supervised learning with fixed size vector = Feedforward Neural Network
2) Topological structure e.g images = CNN
3) Input / Output is sequenced = LSTM or GRU

- Choose optimization algorithm :-

1) Stochastic Gradient Descent with momentum + decaying learning rate.
2) Adam batch normalization = dramatic effects.
3) We Resonable to omit batch normalization.

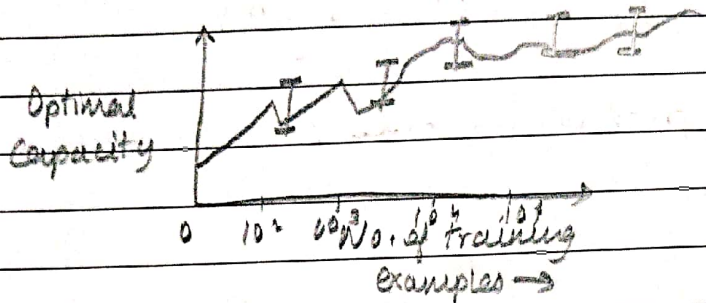
- Regularization :-

1) If input not > 10 million include regularization.
2) Early stopping Mostly used.
3) Dropout easy to use & must be implemented.

* Whether to gather more data

- After end-to-end system is established work on how to improve it.
- ~~sometimes~~ Often better to get more data than to improve algo.

- If performance on training data is poor, more data is needed.
- or if no data, then increase model size or improve learning algorithm.
- If performance on test data is acceptable, nothing else required.



* Selecting Hyperparameters :-
* Manually :- Requires understanding
* Automatic :- Computationally expensive
* Learning rate is the MVP in hyperparameters.
* To do automatic hyperparams. selection :-

1) Grid Search :-
- Exhaustive combos of hyper params. using grid.
- Creates cartesian product of all hyper params.

2) Random Search :-
- Samples values from pre-defined distributions.

* Choosing list of parameters :-
- Choose the smallest and largest values for defined range, so we cover better range.
- For example learning rate can be : [0.1, 0.01, 0.001, 0.0001]
  Here test with 0.1 & 0.0001 to cover wide range.

UNIT 5 :- Deep Generative Models

* Introduction :-

- Generative modelling refers to building a model $p(x)$, from which we can sample from.

• Why are they important :-

1) Data efficeny :- Helps remove or reduce dimensions of data
2) Model checking by sampling. 3) Understanding latent variables

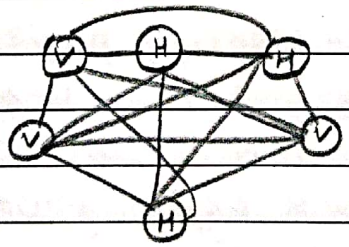
- So our goal is to take input samples from some distribution & learn a model that represents those distribution.
- It can be used for density estimation & sample generation.
- It can help in debaising & outlier detection.

* Boltzmann Machines (BMs)

- Unsupervised DL model; every node is connected every other.
- Connections are bidirectional, Boltzmanns are stochastic models.

• Two nodes in BMs are :-

1) Visible nodes :- Nodes we can measure.
2) Hidden nodes :- Nodes that cannot be or we don't measure.



A Boltzmann Machine

• Mathematics :-

BMs are defined over a d-dimensional binary vector $x \in \{0,1\}^d$. We define its probability using energy function.

$$P(x) = \frac{\exp(-E(x))}{Z}$$

$E(x)$ is energy function.

Z is partition function that helps ensure $\sum P(x) = 1$

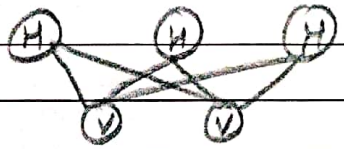
$E(x) = -x^T Ux - b^T x$

where U is weight & b is vector of bias parameters.

- Boltzmann distribution specifies different states of system and BMs create such states.

* Restricted Boltzmann Machines:- [RBMs]

- In BMs every node is connected to every other, and the network grows exponentially, this is where RBMs come in.
- In RBMs hidden nodes cannot be connected to each other as well as visible nodes.



RBMs are bipartite graphs.

- Its energy function: ~~E(v,h)~~ $E(v,h) = -b^T v - c^T h - v^T W h$

b, c, W are learnable parameters, V & H are visible & hidden vecs.

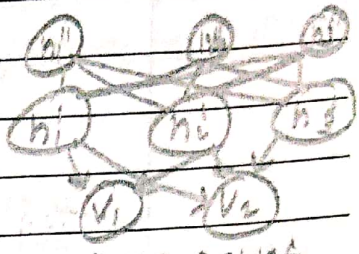
- Its joint probability:- $P(v,h) = \frac{Exp(-E(v,h))}{Z}$

- RBMs are used to build Recommender Systems.
- Feed Forward Pass :- Identify positive associations; i.e a link between V node & H node is a match. We also track negative associations.
- Feed Backward Pass :- Adjusting weights; Understanding weights that activate a link.

* Deep Belief Networks :-

- A hybrid graphical model involving ~~no~~ both directed & undirected connections with no intralayer connections like RBMs.

MANJARA CHARITABLE TRUST
RAJIV GANDHI INSTITUTE OF TECHNOLOGY, MUMBAI
JUHU VERSOVA LINK ROAD, VERSOVA, ANDHERI (WEST), MUMBAI - 53.



Deep Belief Network

- Has multiple hidden layers.
- It was the first non-convolutional deep architecture.
- DBNs are generative models with several layers of latent variables [Hidden variables]
- In DBNs, connection between top two layers are undirected while rest are directed.
- A DBN with only one hidden layer is an RBN.
- DBN = Stacks of RBMs + Fine tuning [Gradient Descent]

• Working of DBNs :-

1) Gibbs sampling :-
- It is done on top two layers of DBNs.
- Gibbs sampling is iterative process, it extracts samples from hidden units given other units.
- After Multiple iterations DBN has generated samples, the dependencies between top two hidden layer does not exist now.

2) Ancestral sampling :-
- After Gibbs, ancestral sampling is done on remaining layers.
- It generates samples from visible units.

3) Greedy pretraining :-
- Used to determine values of hidden/latent variables.
- Done layer by layer.

• Main advantage of DBN :- Learn features in unsupervised manner.
- DBNs resistant to overfitting.

* Generative Adversarial Network [GAN] :-
- Way to make generative models by having two neural networks compete with each other.

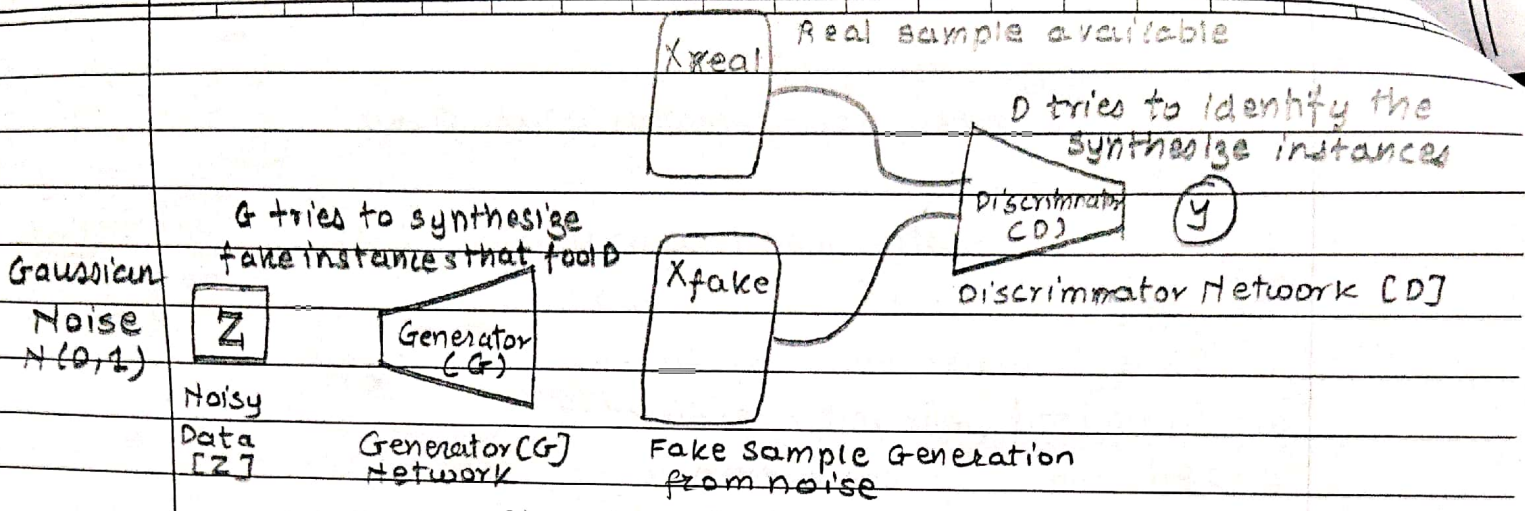
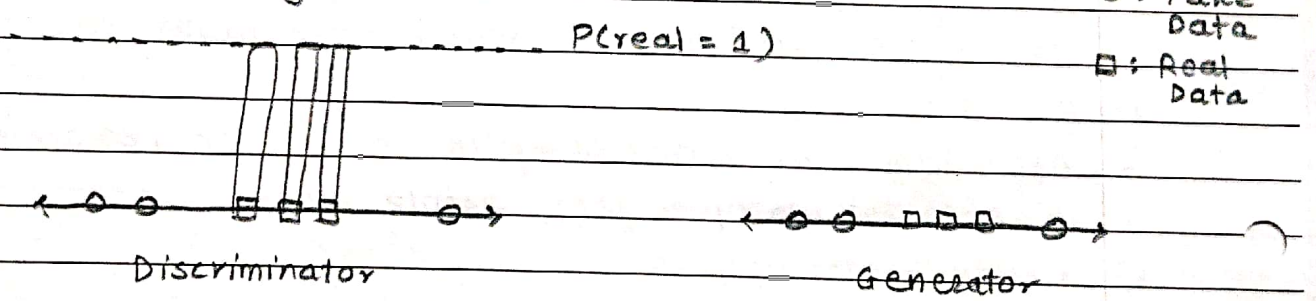
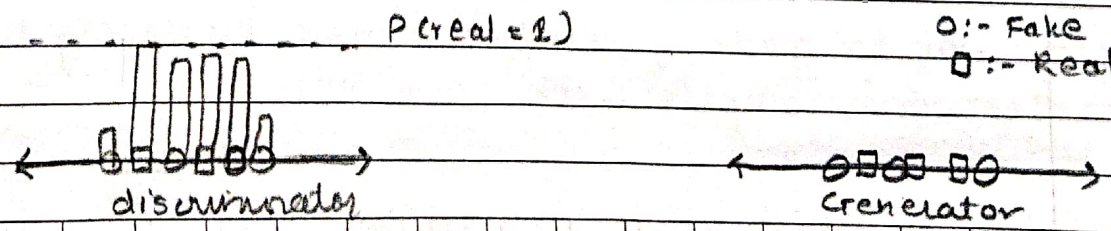


Fig :- Architecture of GAN

- In GAN, two neural networks are at war with each other to help generate the best samples.
- Generator :- It turns noise into imitation of the data to try and trick discriminator.
- Discriminator :- Identifies real data from fakes created by Generato
- Understanding G & D :-



1) Generator starts from noise (indicated by o).
2) It passes these data to discriminator.
3) Discriminator predicts what's fake and what's real.
4) Generator will now try to improve its data such that it becomes similar to real data.



After training generator for meultiple iterations.

- Training :- Adversorial (Against each other) objectives for G & D.
- Global optimum :- G produces true data distribution.
- Loss Functions :-

Loss Functions are used for training they are defined for G & D as follows :-

1) ~~Discrimna~~ Discriminator Network : [D] :-

$$\underset{D}{\text{argmax}} \; E_{z,x}\left[\underbrace{\log D(G(z))}_{\text{Fake}} + \underbrace{\log(1 - D(x))}_{\text{Real}}\right]$$

- argmax is used as D wants to maximize its probability that real is classified as real & fake as fake.
- G(z) is the generated output.
- log D(G(z)) is the log-likelihood of that output being fake.
- D(x) is the estimate that a real data from training sample x is fake, and 1 - D(x) tells if a real data is real.

2) Generator Network [G] :-

$$\underset{G}{\text{argmin}} \; E_{z,x}\left[\log D(G(z)) + \log(1 - D(x))\right]$$

- G simply wants to minimize the probability that generated data is identified as fake.
- Hence total loss function is :-

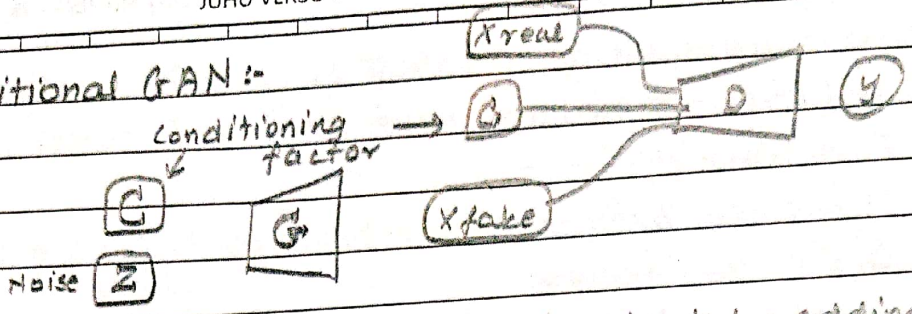
$$\underset{G}{\arg\min} \; \underset{D}{\max} \; E_{z,x}\left[\log D(G(z)) + \log(1 - D(x))\right]$$

After training generator network creates new data that is never seen before.

* Types of GAN :-

MANJARA CHARITABLE TRUST
RAJIV GANDHI INSTITUTE OF TECHNOLOGY, MUMBAI
JUHU VERSOVA LINK ROAD, VERSOVA, ANDHERI (WEST), MUMBAI - 53.

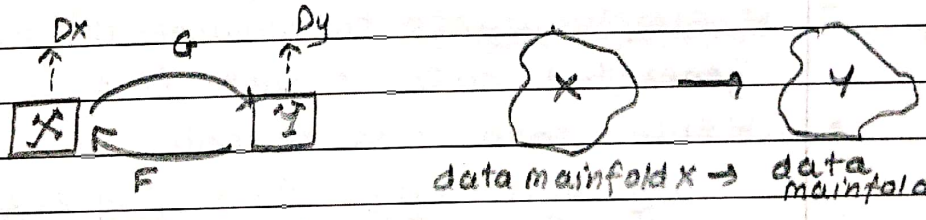
1) Conditional GAN :-



- Condition factor control nature of output by adding conditioning on label.
- For examples, Google Maps converts aerial view to street views.
- In aerial view the images of the area is seen.
- In street view only the borderlines of area is seen.
- So we design a GAN such that our G takes ~~aerial~~ street views to generate an aerial view and our D will compare between generated aerial view and the real street, to see if they form a pair i.e. they are correct.
- Hence a condition is added to compare pairs.

2) Cycle GAN :- Domain Transformation

- These transform across domains with unpaired data.



- Model takes input images from one domain,
- It may not have images from target domain but is trained to create images in the target domain.
- This can not only tranform images but also speech.
- This is how deepfake audios are generated also vids.
- This is now used to create what it would feel like if a ~~ee~~ singer sings an indian songs.

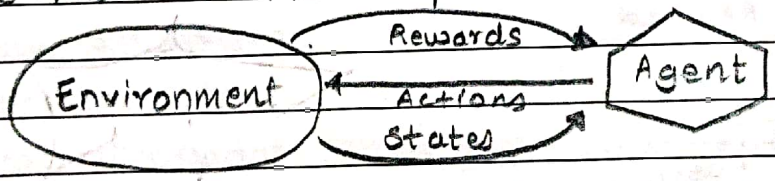
3) Vanilla GAN :-
- The GAN that we discussed earlier was vanilla GAN.
- Simplest form of GAN.

* Applications of GAN Networks :-
1) Create Augmented data [Add more features to add]
- Given a set of facial pictures add sunglasses to everyone.
⇒ Create different versions of same images.
2) Generate fake images.
3) Image-to-image translation [Conditional GANs]
4) Text-to-image translation.
5) Photograph editing.
6) Photo Blending (Mix Photos)

UNIT 6 :- REINFORCEMENT LEARNING

* Introduction :-
- Deep Reinforcement Learning = Reinforcement + Deep learning.
- Reinforcement learning [RL] consists of an agent that learns by trial & Error.
- Deep RL makes agent take decisions from unstructured input without manually creating the state space.
- Used in robotics, video games, computer vision & more.
- Google made an agent play Atari breakout.
- DeepMind created Alpha Go an agent that plays game of Go.

* Markov Decision Process :-
- It is used to formalize a reinforcement learning problem.
- If the environment is completely observable its dynamics are known as markov process.



Markov Decision Process

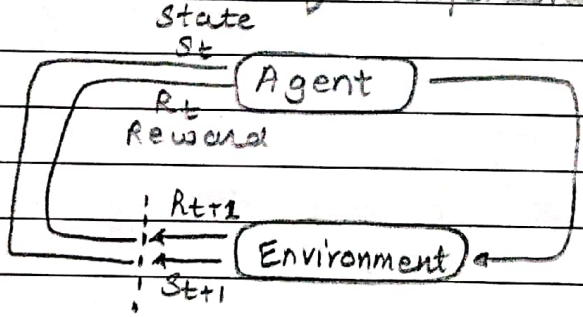
- In MDP, agent constrantly iteracts with environment to get in a new state and ~~fetch~~ be gifted rewards.
- MDP contains :-

1) Set of possible states (S) :-
- Every state that agent can be in

2) Model [$T(S, a, S') \sim P(S' | S, a)$] :-
- A Transition Model is defined as ~~T(S, a, S')~~ where T is function of current state S, action taken a, and

next state reached S'.
- For stochastic actions we define it as P(S' | S, a).
- Actions [A(s)] :-
- A(s) denotes set of actions that can be performed for a set of states s.
- Reward [R(s, a, s')] :-
- Real-valued function.
- Policy [π(s)]
- Given set of ~~set~~ states it decides the best action to choose from.
- $\pi^*(s) = \underset{a}{\operatorname{argmax}}\ \varphi(s, a)$
- Markov property :-
- Current state transition does not depend on past states/actions

* Basic Framework of Reinforcement Learning :-



- Agent recieves first state S_0.
- Based S_0, agent performs action A_0.
- Based on action agent goes to state S_1.
- Agent also receives reward R_1.
- Agents goal is to maximize the reward.

$$R_t = \sum_{i=t}^{\infty} \gamma^i r_i = \gamma^t r_t + \gamma^{t+1} r_{t+1} + \ldots + \gamma^{\infty} r_{\infty}$$

γ represents the discount factor for the i^{th} environment.

- S is state of game & o is partially observable state.
- In a discreate space actions are finite. [Mario]
- In a continuous space actions are infinite [Dr. Driving].

* Challenges of Reinforcement learning :-
1) Sample efficeny [learning with less samples].
2) Reproducibility issues [Difficult to replicate an RL Model]
3) Performing in Real-life scenarios [self-driving car].
4) Sparse reward. [Reward Techniques fail]
5) Offline reinforcement [Agent stops collecting experiences].

⇔

* Dynamic programming algorithms for reinforcement learning.
- Given an MDP, dynamic programming can find optimal policy (π).
- They are only useful when environment dynamics are known and can be modeled accurately.
- Methods :-

1) Value Iteration :-
- Helps compute optimal value function and policy for an MDP.
- Iteratively update value function until convergence.
- Algorithm :-

i) Intialization :-
- Intialize value function for all states as zero or arbitary.

ii) Iteration :-
- Repeat until convergence :
  For each state update value f^n using Bellman's eq,

$$V^*(s) \leftarrow \sum_{a \in A} R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a V^*(s')$$

- We iteratively apply bellman's eq^n until value convergence to optimum.

- After convergence, value function represents return from each state.

2) Policy iterations :-

- Alternates between policy evaluation & policy improvement.
- Algorithm :-

i) Intialization :

- Intialize policy $\pi(a|s)$ arbitarly.

ii) Iteration :-

- Repeat until convergence :-
  - Policy Evaluation :-
    - For each state s, evaluate value function for each policy using Bellman's expectation equation :-

$$V_{k+1}(s) = \sum_{a \in A} \pi(a|s) \left[R_s^a + \gamma \sum_{s \in S} P_{ss'}^a V_k(s') \right]$$

  - Policy improvement :-
    For each ~~policy~~ state s, update the policy by choosing action that maximizes returns :-

$$\pi' = \text{greedy}(V_\pi)$$

- This algorithm interatively peforms policy evaluation and policy improvement steps until policy converges to optimal policy.

* Q-Learning :-

- A RL learning policy that finds the best next action.

<Draw Agent / Environment diagram>

- It is model-free, it does not use reward system to learn, but uses trial & error.
- $Q(s_t, a_t) = E[R_t | s_t, a_t]$

Q-Function gives expected total future reward for an agent in state s by performing action a_t.

- Q - Function simply gives the expected reward, the action to choose from given rewards is done by ~~Q~~ policy $\pi(s)$.
- $\pi^*(s) = \arg\max_a Q(s_t, a_t)$
- We use iterative approximation to learn the Q function:-

$$Q(s,a) = r(s,a) + \gamma \max_{a'} Q(\delta(s,a), a')$$

This is the Bellman's equation, where r() is reward, γ is discount & δ is resulting state.

- The Bellman's equation is a recursive definition for Q that helps us best approximate Q iteratively.
- The algorithm for Q-learning is as follows:-

1) Intialize all $\hat{Q}(s,a)$ to zero. ($\hat{Q}$ is learner's estimate of Q)
2) Observe current state s.
3) ~~Do forever~~ Repeat
   - Select action a & execute it.
   - Receive reward r.
   - Observe new state s'.
   - Update table entry for $\hat{Q}(s,a)$ as

$$\hat{Q}(s,a) \leftarrow r + \gamma \max_{a'} \hat{Q}(s',a')$$

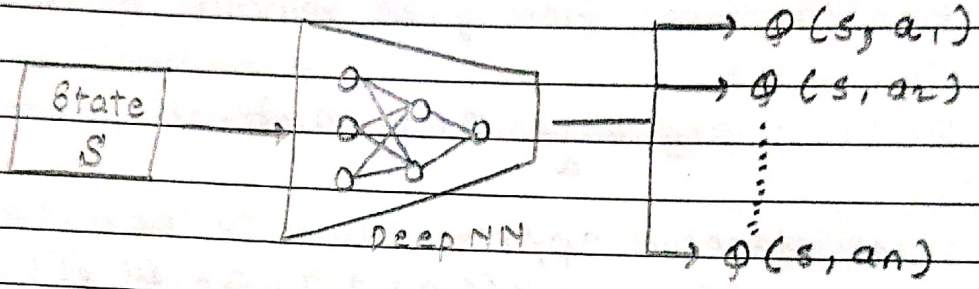
   $s \leftarrow s'$

- Q-learning algo converges on 3 conditions:

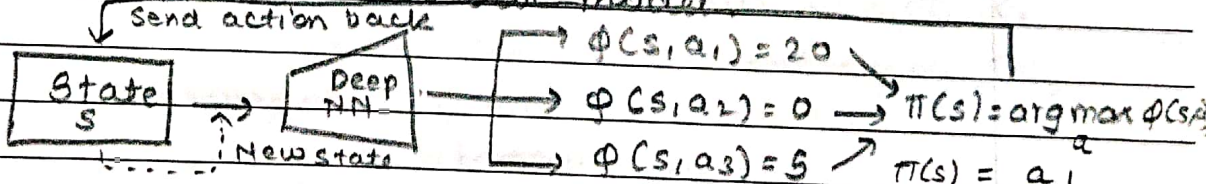
1) Deterministic MDP.
2) Immediate reward values are bounded $r(s,a) < c$
3) Agent visits every state-action pair possible.

MANJARA CHARITABLE TRUST
RAJIV GANDHI INSTITUTE OF TECHNOLOGY, MUMBAI
JUHU VERSOVA LINK ROAD, VERSOVA, ANDHERI (WEST), MUMBAI - 53.

* Deep - ϕ - Network



- ϕ-learning is only practical for small environments.
- Deep - ϕ - Networks uses DNN to approximate ϕ - values and ϕ-functions, rather than table of values.
- $L = E\left[\| \underbrace{(r + \gamma \max_{a'} \phi(s', a'))}_{\text{Target}} - \underbrace{\phi(s, a)}_{\text{Predicted}} \|^2 \right]$
- Above equation is called as ϕ - loss
- ϕ-loss is used to train our model



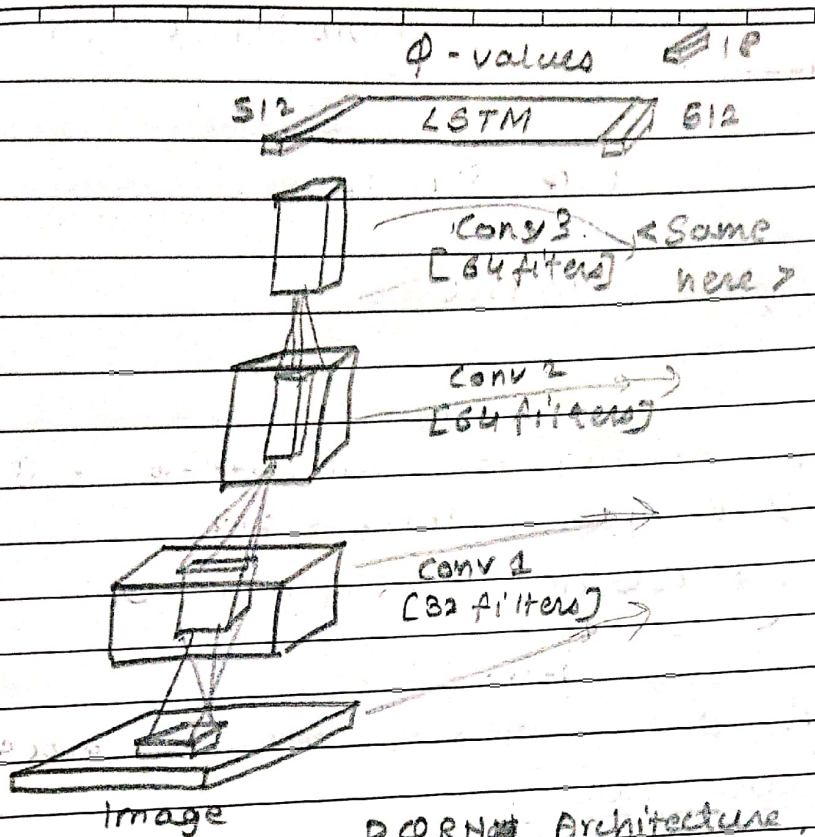
- Optimal policy function chooses the best action possible based on expected returns.

* Deep ϕ Recurrent Networks :-

- Deep ϕ-Recurrent networks extend the DQN architecture by adding recurrent layers allowing the network to capture sequential & temporal information.
- DQRNs use two networks in their architecture.

i) CNN :- Used to extract observations [Observation model]

ii) LSTM :- Used to get better approximations of states using ~~CNN~~ from observations [Transition Model].



DQRN Architecture.

- < Explain Architecture >
- DQR is mostly used to train agents for games.

* Reinforcement learning for tic-tac-toe :-

- Let us define some parameters for tic-tac-toe game :-
  - State space :- Possible configs of grids, combinations of x and o on the grid.
  - Actions :- Action correspond to placing X or O.
  - Reward function :- +1 for W, 0 for Draw.
  - Q-function :- Expected return for particular action.
  - ~~Policy~~ :-
  - Tic-Tac-Toe now can be implemented as follows :-

I) Intialize :-
- Intialize $\phi(s,a)$ $\forall$ $s \in S$ as arbitary values.
- Intialize ϵ to control exploration & exploitation rate.

II) Training :-
- Choose action based on current state.
- Observe next state & reward.
- Update ϕ :- $\phi(s,a) = \phi(s,a) + \alpha[r + \gamma \max_{a'} \phi(s',a') - \phi(s,a)]$

III) Exploration - Exploitation Strategy :-
- ϵ gradually is decreased and focus is moved to exploitation.
- Exploration : Trying out different actions.
- Exploitation : Selecting actions only based on learned knowledge.

IV) Evaluation :-
- Training agent against trained opponents such as rule-based agents.
- Test the agents ability to win.