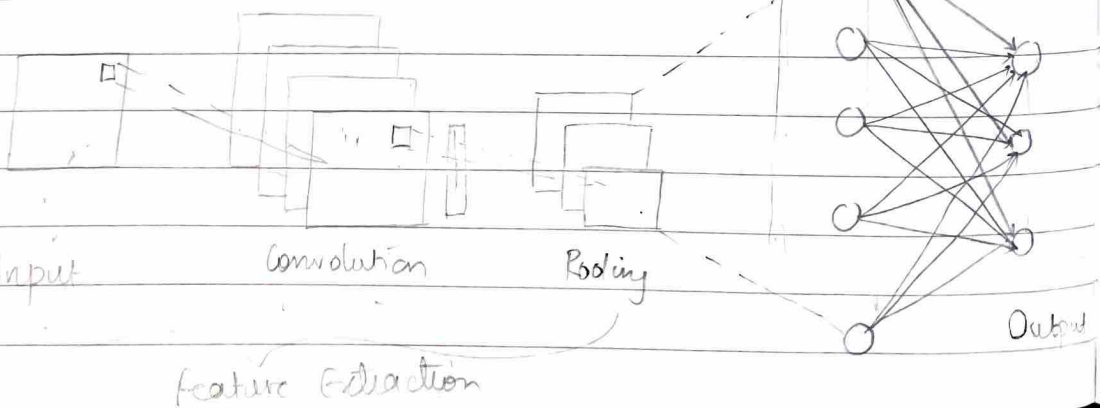
UNIT 3.
CONVOLUTIONAL NEURAL NETWORKS.

* Convolutional Neural Network (CNN).

- Special class of multilayer perceptrons
- designed to recognize 2-D shapes with high degree of accuracy to skewing, scaling & translation.
- type of ~~neural networks~~ deep learning algo. used to analyze visual data such as images & videos
- it automatically learns & extracts relevant features from the input data through convolutional layer.
- Input data is processed through multiple layers - convolut-ional, pooling & fully connected layers.
- convolutional layer - apply filters to input data & perform convolution operations which capture patterns & local dependencies.
- Pooling layer - downsample feature maps, reduce dimensionality while retaining important information.
- fully connected layer - connect high level features to final output layer which produces desired output.
- applications: image & video segmentation, image classification, medical image analysis, NLP brain computer interfaces

* Architecture of CNN.



1) Input layer
- first layer of CNN.
- recievers input data which is typically an image or set of images.
- dimensions correspond to size of input images.

2) Convolutional layer.
- responsible for extracting features from the input data.
- each layer applies a set of learnable ~~features~~ filters to the input, performing convolutional operations to produce feature maps.
- filters detect patterns & local dependencies in the input such as edges, textures or shapes.

3) Activation function.
- after each convolutional layer, a non-linear activation function is applied to the feature map.
- this introduces non-linearity allowing network to learn complex relationships.

4) Pooling layer.
- downsample the feature maps.
- reduce the dimensionality while retaining important information.
- common operations → max pooling & average pooling.
- Pooling helps in reducing the no. of parameters & controlling overfitting.

5) Fully connected layers.
- after convolutional & pooling layers, feature maps are flattened into vector & are fed into one or more fully connected layers.
- similar to traditional neural networks (one neuron connected to another)
- capture high-level relationships & produce final output

6) Output layer.
- produces the final predictions or classifications.
- no. of neurons present depend upon the specific task.
- eg. in image classification, output layer may have neurons corresponding to the no. of classes, each neuron represents the probability of a particular class.

* Padding

- technique used in CNNs to preserve spatial dimensions of the input data when applying convolution operations.
- involves adding extra border pixels around input data before performing convolutions.
- typically employed to address the issue of dimensionality reduction which occurs when convolutions are applied.
- spatial dimensions of output feature decrease after each convolution layer → loss of information.
- by adding padding → dimensions of O/P feature maps can be maintained or controlled.
- adds extra pixels with zero values around input data, creating a border that allows filters to access pixels at the edges of input.
-

Types.

Same Padding | Valid Padding | Causal padding.

1) Same Padding
- size of padding is adjusted so that output feature maps have the same dimensions as the input.
- done by calculating necessary amount of padding based on

32 - 512 filters in parallel

size of convolutional filter.
- used when goal is to maintain spatial dimensions / reso. of input through out the network
- padding layer have zero values in the outer fram of images

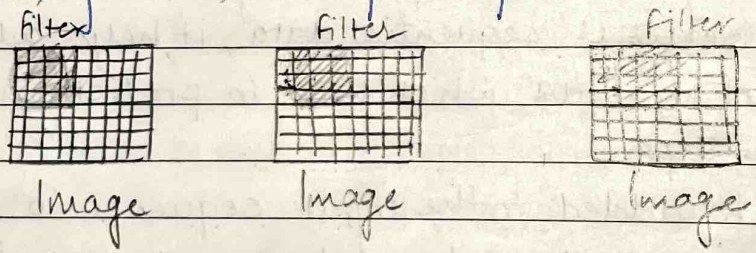
2) Valid Padding.
- no padding is added to the input.
- ∴ spatial dimensions of the output feature maps are smaller than the input.
- used when goal of is to reduce the spatial dimensions & extract high level features from the ~~output~~ input..
- used with Max-pooling layers.
- works on validation of pixel & not size of input.

3) Causal Padding.
- works with 1-D convolution layers.
- used majorly in time-series analysis.
- As time series is sequential data, it helps adding zeros at the start of data, which helps in predicting the values of previous steps.
- padding is added to the input sequence in such a way that no information from future elements is accessible to the current element during convolution operation.

Strided Convolution.

- used to downsample the dimensions of the feature maps.
- Unlike usual convolution operation which uses stride of 1, strided convolution involves moving filter across input with larger stride value → resulting in reduction in output size.

- stride → no. of steps conv. filter takes while sliding across the input data.
- stride of 1 ⇒ filter moves 1 pixel at a time → OP feature map size = input feature
- stride parameter → length of step in stride.
  ↗ always 1 in any framework
- ↑ stride to save time / cut calculation time.
- ✗ stride of 0 → no sliding at all.
- If we use convolution with a (2×2) stride, the step is 2 in both x & y direction followed by non-strided convolution stride 1, step 1.
- The output of a convolution with stride 2×2 halves the width & height of input
- Helps to capture broader contextual features & info while reducing the computational cost.
- helps to summarize the info in larger regions of input, resulting in more compact representations.



Left Image : Stride = 0
Middle Image : Stride = 1
Right Image : Stride = 2

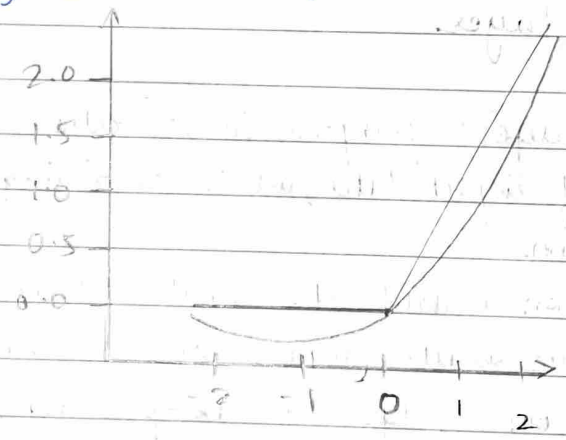
Rectified linear unit.

- widely used activation function that introduces non-linearity to the network.
- also known as ramp function.

- defined as

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

- sets all negative values in the I/P to zero & positive values are left unchanged.
- It clips -ve values at zero & produces / introduces a non-linear activation.
- overcomes the vanishing gradient problems, allowing models to learn faster & to perform better.
- it is default activation when developing multilayer perception & CNN.
- helps CNN to capture complex relationships & non-linear patterns in data.
- helps CNNs learn & represent wider range of functions.
- computationally efficient to compute.

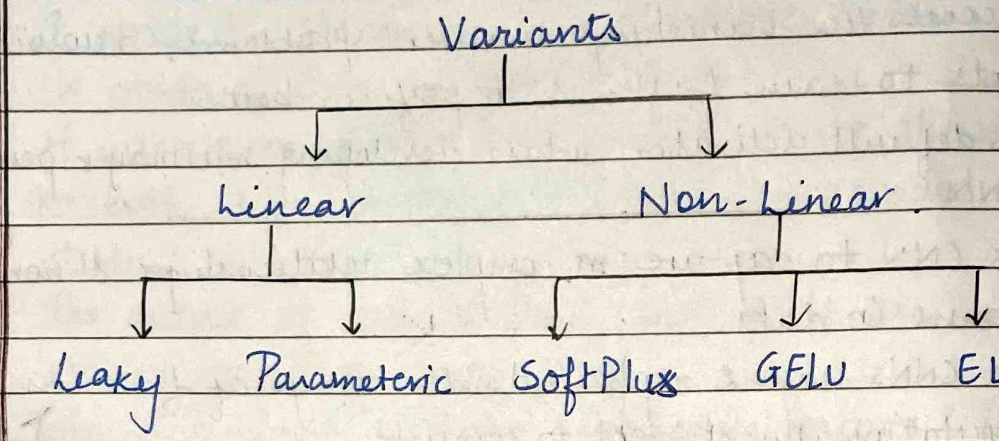


Advantages:

1) Gradient propogation is better
   Very few vanishing gradient problems, compared to sigmoidal activation function.
2) computation is fast & easy. → involves +, × & comparison
3) scale invariant.
4) allow effective training of deep neural architectures & complex database

Disadvantages.

1) Function is differentiable except at 0
   value of derivative at 0 is ~~ab~~ arbitrarily chosen as 0 & 1.
2) It is unbounded.
3) it is not zero centred.



Pooling Layer

- Pooling layer's purpose is to reduce the spatial dimensions (w & h) of input data, while retaining the most important information.
- operates on a grid-like input such as an image / feature map, divided into smaller regions called pooling windows / kernels.
- ~~the most common type → max pooling~~.
- dimensionality reduction → less computational power
- aids in proper training the model & extracting dominating characteristics.
- Types.

Max Pooling Average Pooling.

i) Max Pooling.

- max. value with in each pooling window is selected & passed

- a window / filter of 2×2 slides over input feature map
- At each pos. of window largest value is selected & retained
- others are discarded.
- preserves prominant features.

to the next layer, while other values are discarded.
- functions as noise-suppressant.
- does de-noising & dimensionality reduction.

ii) Average Pooling.
- average of all values from the area of the picture covered by the kernel is returned by average pooling.
- only carries out dimensionality reduction as noise-suppressing strategy.
- Max Pooling >> Average Pooling.

less prominent features can cause blurring.

12	20	30	0
8	12	2	0
34	70	37	4
112	100	25	12

max pooling →

20	30
112	37

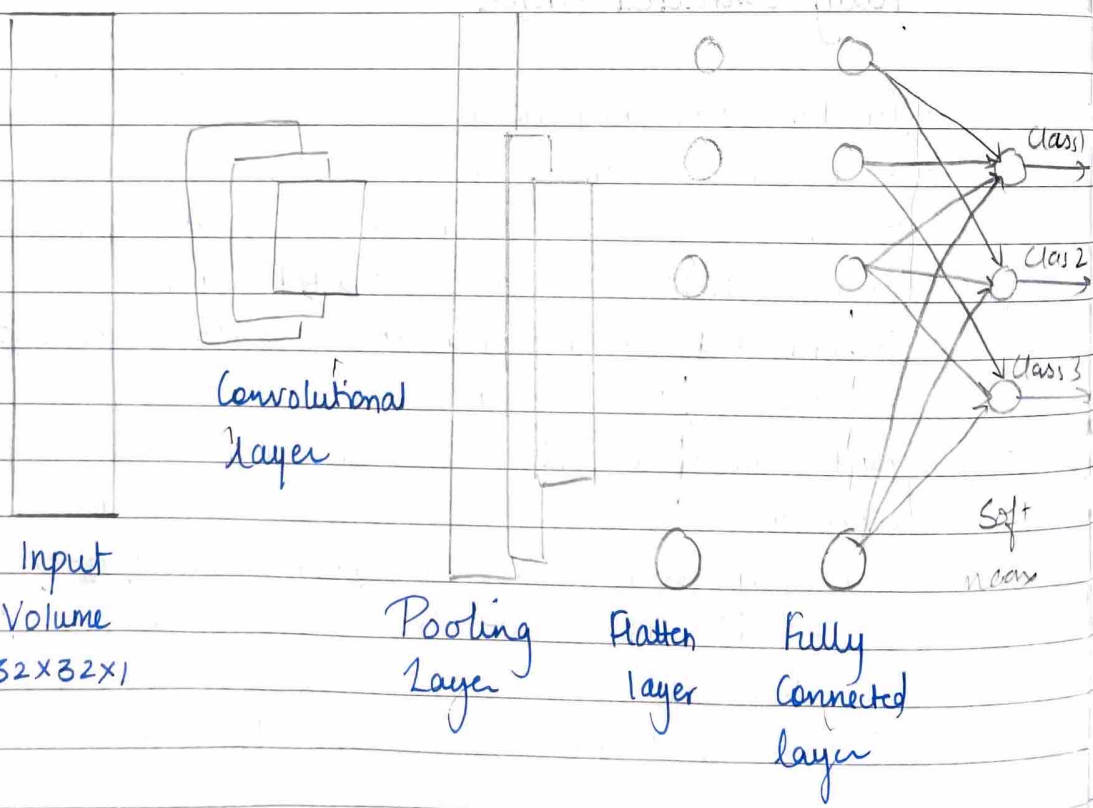
average pooling →

13	8
79	20

Fully Connected Layer

- fundamental component.
- ~~responsd~~ it follows the convolutional & pooling layers.
- responsible for learning complex relationships & making final predictions based on extracted features.
- it connected every neuron in the previous layer to every neuron in the current layer.
- each neuron in the fully connected layer recieves inputs from all the neurons in the previous layer.
- Eg- Suppose we have a feature map ~~of~~ after convolutional & pooling layers with 7×7×64, 7 columns, 7 rows and 64 channels.
- To feed the feature maps into a fully connected layer we ~~to~~ flatten it into vector

- Size of flattened vector = 3136.
- This flattened vector is then connected to a fully connected vector
- Each neuron fully connected layer → every neuron in flattened vectors through weighted connections.
- weights = parameters of fully connected layers.
- each neuron o in the fully connected layer computes the weighted sum of inputs it recieves & passes the output through activation function.
- activation function produces non-linearity & allows model to learn complex relationships bet^n features.
- Output of fully connected layer is fed into one or more subsequent fully connected layers.
- The final fully connected layer ~~often~~ the output layer is responsible for producing final predictions based on the learned representations.



LeNet 5 → CNN
7 layers. excluding input layer, IP → weights (trainable param)

1) Layer C1 → Conv layer → 6 feature maps, size of FM = 28×28
2) S2 → Sub sampling layer → " → " , size = 14×14
3) C3 → conv layer → 16 FM size = 10×10
4) S4 → sub sam - 16 FM size = 5×5
5) C5 → conv layer 120 F.M 1×1. 6) F6 → 84 units full con. to C5.

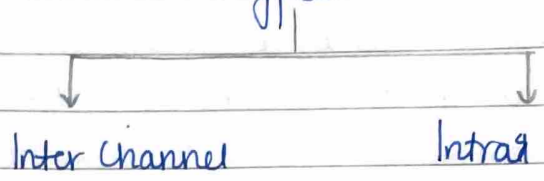
Interleaving between Layers.

- refers to the process of combining or integrating the output of different layers in a neural network.
- involves merging the feature maps or activations from multiple layers to create a unified representation of info. from various levels of abstraction
- used to enhance performance & effectiveness of network.
- allow flow directly from one layer to another.
- Eg. ResNet.
  In ResNet, the output from ~~previous~~ one layer is combined with the output of its previous layer by concatenation.
- This allows network to reuse & propogate information from earlier layers to deeper layers, addressing the problem of vanishing gradients.
- improves gradient flow
- helps in learning more accurate & robust features.

Local Response Normalization.

- technique used in CNN to enhance response of neurons. & promote competition among neighboring neurons.
- type of normalization applied to the output of convolutional layer
- purpose is to normalize responses of neurons within a receptive field / local neighborhood.
- LRN is typically applied independently to each feature map produced by the convolutional layer.
- effect of LRN is to increase response of neuron relative to its neighbors if it has large activation compared to them
- promotes competition between neurons.

- encourages the network to focus on most active neurons within local region.
- Types.



Batch normalization.

- supervised learning technique to transfer middle layer output of neural network to normal form / common form.
- speeds up learning technique because normalization prevents activation values from being too high or low
- allows each layer to learn independently of other layers.
- normalizing input reduces data loss between processing layers.
- batch normalisation influences output of the preceding activation layer.
- provides layer with 2 more trainable parameters: α & β.
- by adjusting other weights, data loss can be reduced & network stability is improved.
- Advantages:

1) Speeds up training process
2) Handles covariate shift, input for every layer is distributed around the same mean & S.D.
3) Smoothens the loss functions by optimizing model parameters.

Training a Convolutional Network.

1) Data Preparation.
- dataset is prepared by dividing it into training, validation

& test set.

2) Model Architecture.
- Design architecture of CNN.
- no. & type of layers, filter sizes & pooling operations are decided.
- AlexNet, LeNet can be used.

3) Initialization
- Initialize parameters of CNN randomly / with pretrained weights.

4) Loss function
- Choose appropriate loss function based on the problem
- Eg. Classification: softmax cross entropy & binary cross-entropy.

5) Optimizer
- Select optimization algo. such as stochastic Gradient Descent to update parameters of your CNN.

6) Mini-batch Training.
- train the CNN in mini-batches.
- done by feeding small batches of data (training) through the network.
- Compute loss & gradient using chosen loss function & backpropagate gradients to update parameters

7) Validation
helps to avoid overfitting.

8) Hyperparameter tuning.
Experiment with hyperparameters such as L.R, batch size, regularization techniques to improve CNN's performance

9) Testing
Once you are satisfied with the performance on validation set, evaluate CNN on test set.

10) Deployment.
- Deploy trained CNN to make predictions on new data.

UNIT IV.
RECURRENT & RECURSIVE NETS.

Unfolding Computational Graphs.

- Computational graph → directed acyclic graph
  - graphical rep. of algorithm
  - nodes → operations
  - edges → flow of data bet^n operations.
- recursive or recurrent computation into a graph having repetitive structure.
- Eg. Classical form of a dynamical system

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

s^t = state of system.

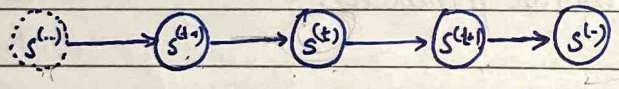
t = time.

- For a finite no. of steps 't', the graph can be unfolded by applying definition '(t-1)' times.

Eg $t = 3$ time steps

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

$s^{(1)}$ = ground state



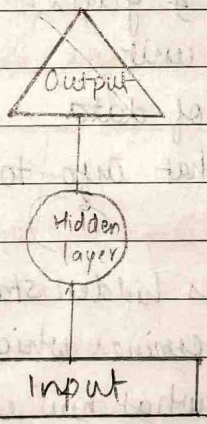
Each node represents state at some time t

f is function that maps state at time t to state at $t+1$.

Recurrent Neural Networks.

- type of artificial neural network
- designed to handle ~~the~~ sequential data.
- RNNs have a form of memory that allows them to retain information from previous steps & use it to make predictions

- basic idea is to introduce loops in the network.
- these loops allow information to be passed from one step to another/next.
- Each step recieves an input & produces an output, & the hidden state that captures information from previous steps.
- Hidden state acts as the memory of the network.
- helps to consider the context & dependencies between a sequential datapoints.
- each module takes an input, processes along with the hidden state from previous state/module & produces output & a new hidden state.
- repeated several times, allowing network to capture the temporal relationships between the input.
- uses the same parameters for each inputs as it performs the same tasks on all inputs & hidden layers.
- RNNs can use memory to process variable lengths sequences of inputs.
- RNNs are tusing complete.
- have an infinite impulse response & it is a directed cyclic graph



Types of RNNs.

- RNNs have additional stored states & the storage can be under direct control by the neural network.
- Such controls are refered to as gated state & they are ~~refered to as~~ a part of LSTM.

Types of RNNs.

1) ~~Binary~~ Simple RNN.

- basic & most straightforward form of RNN.
- propogates information from previous steps to current step.
- suffer from the vanishing gradient problem, where gradients diminish over time
- difficult to capture long-term dependencies.

2) Long Short Term Memory (LSTM)

- type of recurrent neural network (RNN).
- designed to address the vanishing gradient problem and capture long-term dependencies in sequential data.
- designed to overcome limitations of traditional RNNs.
- information is stored & retrieved in memory cells, showing the network to retain information for longer periods.
- key components : cells & gates.
- memory cell - storage unit
  gates - control flow of data

1) Forget gate : decides what info. to discard from previous memory cell.
   - takes previous hidden state & current input as inputs & output determines which info is retained.
2) Input gate : decides what new information should be stored in the current memory cell.
   - takes previous hidden states, current input as inputs applies an activation function & produces an output.
3) Output gate : decides what information should be outputted from current memory cell.

3) Gated Recurrent Unit (GRU) Networks.

- similar to LSTM.
- designed as a simplified versions
- use gating mechanism to control the flow of information
- but they have reduced no. of gates
- computationally efficient.
- it has a simpler structure than LSTM.
- GRU has two gates: Update Gate & Reset Gate
- Based on update & reset gate, the GRU computes the current hidden state & output.

4) Bi-directional RNNs

- sometimes, context & dependencies of both past and future time steps are important.
- bi-directional RNNs address this by having two separate RNNs - ~~one processing the sequence~~
- One processing the sequence in forward direction & the other in the reverse direction
- outputs from both directions are combined to capture both past & future information.

5) Heirarchial RNNs.

- ~~incorporates RNNs incorporates~~ -
- RNNs incorporates multiple layers of RNNs to capture heirarchial structure in sequential data.
- lower layers capture local dependencies.
- shigher layer capture long-term dependencies across larger sequences.
- structure helps to model complex dependencies in the data.

★ Bidirectional RNNs.

- In many sequence-based tasks, understanding the context from both past & future data can be beneficial.
- Bidirectional RNNs were designed to address this need by combining info from both directions, allowing the network to capture dependencies in both past & future data.
- Instead of processing the input sequence in a single direction a ~~bird~~ bidirectional RNN processes the sequence in both directions
- This allows the network to capture dependencies in both ~~past & future data~~ forward & backward direction simultaneously
- Working.

1) Forward Pass.
- the input sequence is fed into one instance of RNN, called forward RNN.
- forward RNN processes sequence from beginning to end.
- at each step, the forward RNN takes current input & its previous hidden state & produces a new hidden state & output.

2) Backward Pass
- Simultaneously, the input sequence is fed into another instance of an RNN, known as backward RNN.
- backward RNN processes the sequence in reverse order starting from end & moving towards the beginning.
- at each time step, the backward RNN takes the current input & its previous hidden state & produces an output & a hidden new state.

3) Combining Outputs.
- Once both the forward & backward RNNs have

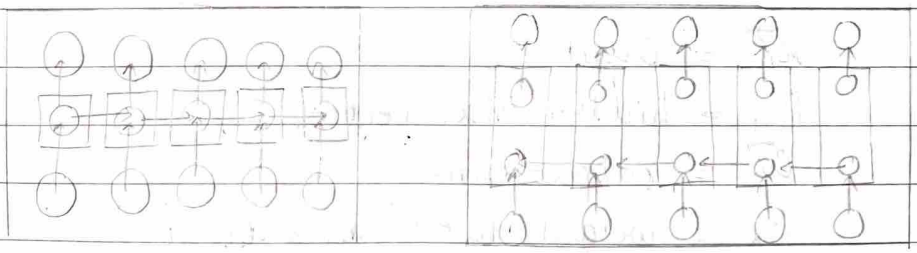
processed the entire sequence, the outputs from each direction are combined.

- involves concatenating the outputs or applying some operation to merge the information captured from both directions.

- The combined output represents the bidirectional representation of the input sequence, incorporating information from both past & future context of each time step.
- can be used for predictions.
- Eg. In NLP, understanding the meaning of a word often relies on its surrounding words both before & after it.
- Bi-directional RNNs can capture these dependencies effectively.
- require entire input sequence to be available at once.

Difference LSTM & BERT.

- LSTM preserves info. of past only as, inputs are seen only from past
Bi-directional - runs in both directions / two ways → past to future & future to past.
- BRNN splits the neurons of RNN in 2 directions - forward states & backward states.



RNN BRNN

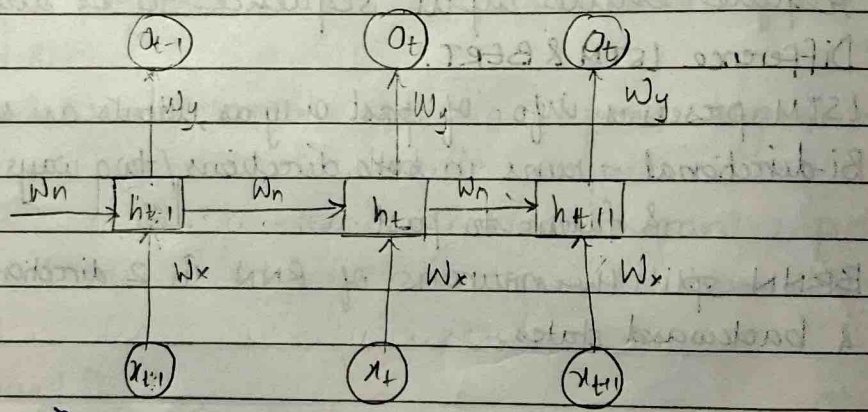
- When back propagation through time is applied, additional processes are needed as input & output layers cannot be done at once.
- Translation, Entity extraction, Pos tagging.

1) Encoder - made by stacking RNNs cells on top of each other.
- RNNs scan the vect input.
- Hidden state is updated at each time step with input
- Using initial input & preceding hidden state, first cell updates its own H.S.
- Every layer produces 2 OPs → hidden state & output.

Encoder-Decoder Architecture.

- framework commonly used in sequence-to-sequence tasks such as machine learning, text summarization etc
- Involves two main components: encoder & decoder.
- It is a system that takes an input sequence & produces output sentence/sequence.

1) Encoder.
- role is to process input sequence & capture its meaningful representation, known as context vector
- takes in input sequence, & produces a condensed rep. of the information contained in the sequence
- condensed rep → "thought vector" / "context vector".
contains imp. info about the sequence

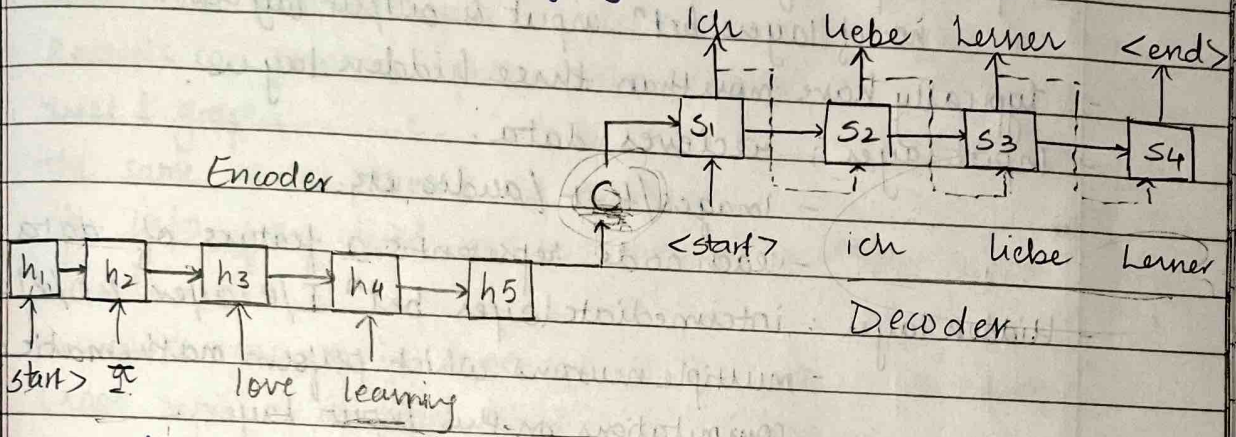


$\vec{x_i}$ = input vector
$\vec{h_{t-1}}$ = hidden state vector
$\vec{O_t}$ = output vector.
W_x = weights bet^n $\vec{x_i}$ & $\vec{h_t}$
W_h = weights bet^n $\vec{h}_{t-1}$ & $\vec{h}_t$

2) Decoder
- takes context vector produced by the encoder & uses it to generate the output sequence
- it recieves context vector as its hidden state & generates

the output sequence step by step.
- At each step, decoder produces output based on the previous state & hidden state
- output state/seq. is built iteratively, by context vector & generated outputs from previous steps.

- enables network to learn how to convert one sequence into another by leveraging context vector.
-



- encoder produces state C representing the sentence in source language (English / I love learning)
- then decoder unfolds the state C into target language Ich Liebe Lernen
- C is vectorized representation of whole sequence.

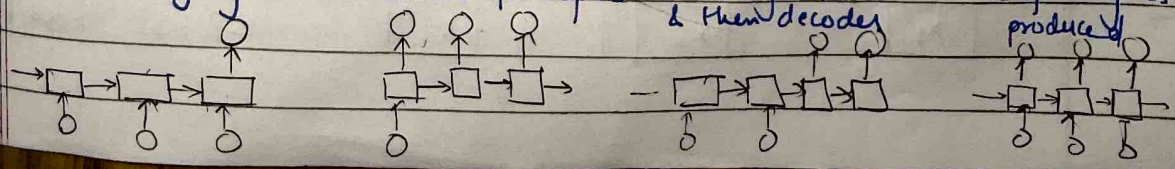
Types of E-D Arch.

Many to One
- classification & Senti analysis
- input = seq. of words
- O/P = category.

One-to many
- text generation
- I/P = topic
- O/P - seq. of words for topic

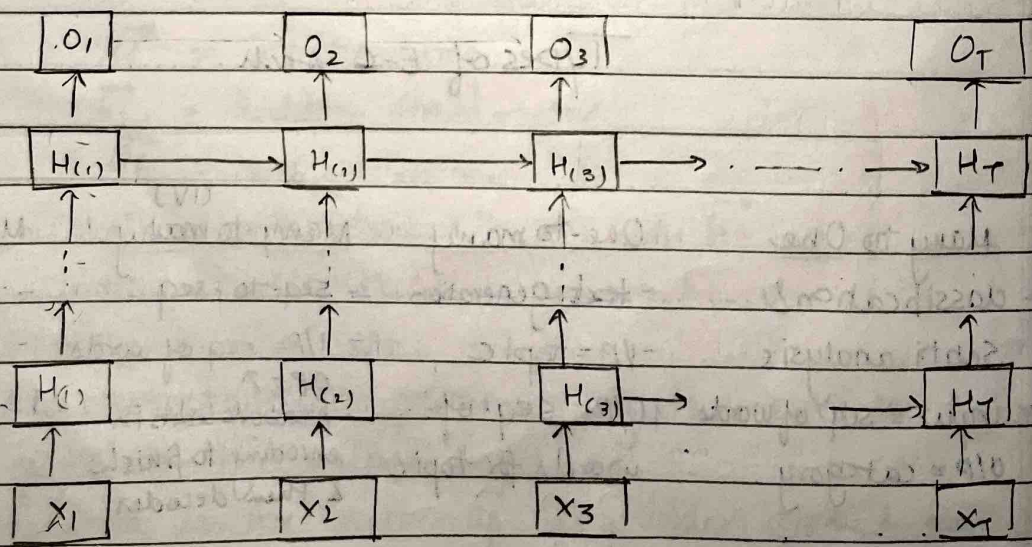
Many-to many (IV)
- seq-to-seq.
- I/P = seq. of words
- OP = ?
- network waits for encoding to finish & then decodes

Many to many (2V)
- pos tagging
- same time frames go & captioning is produced



Deep Recurrent Network.

- consist of multiple layers of interconnected / stacked RNNs on top of each other.
- the networks are called "deep" because they have significant no. of hidden layers, allowing them to learn hierarchical representations of data.
- deep = depth of networks:
  no. of layers bet^n input & output layers.
- typically have more than three hidden layers.
- Input layer: recieves data.
  - images/text/audio etc.
  - each node represents a feature of data.
- Hidden layer: intermediate layer bet^n I/P layer & O/P layer
  - multiple neurons which perform mathematical computations on the input layer.
  - each neuron takes input from previous layer, applies an activation function & produces output which is input for the next layer
- Output layer: produces final output.
  - no. of nodes depend on the task.



- DNNs are trained using backpropogation which involves iteratively adjusting the wts & bias of neurons to minimize difference between predicted output & desired output.
- training is done using large labelled datasets.

Recursive Neural Networks.

- type of neural network architecture designed to handle heirarchial relationships.
- RecNNs can process inputs with variable input-lengths like trees & graphs.
- the same neural network is recursively applied on all inputs
- the info. from child nodes is combined to compute rep. of parent nodes.
- recursive process is continued until single representation (root rep) is obtained for the whole structure.
- Each element is encoded into low-dimensional rep. using encoding.
- The RecNN iterativly applies the same neural network module to compute representations at each level of the structure.
- Once, the recursive processing is complete, the final representation of the entire structure is obtained.

The Challenge of long-term Dependencies.

- long-term dependencies : dependencies bet^n distant elements in a sequence, where info from previous steps is relevant for making predictions
- RNNs are designed to process sequential data by maintaining hidden states that capture info. about the past context.

- As the sequence length increases, RNNs struggle to effectively propogate & retain relevant info over long time steps.
- problem arises due to vanishing/~~grad~~ exploding gradient issue
- it affects the ability of RNNs ~~of~~ to learn & update their parameters based on info from time steps.
- during training, when backpropogating error gradients, the gradient can either become extremely small or large.
- solutions:
  LSTM → memory cells & gating mechanisms
  GRU → gating mechanisms
  Transformer based → Chatgpt / Bert

Echo state Networks.

- type of RNN which has three main components: input neurons, a reservoir & output neurons.
- input neurons recieve input data → a sequence of values
- reservoir is a large set of recurrently connected neurons that serve as hidden layer.
- these neurons are randomly initialized & have fixed connections among themselves
- output neurons take the information from the reservoir & produce final output.
- During training, only the connections from the input connections from the I/P neurons → reservoirs are learned.
- connections with reservoir & reservoir → output neurons are randomly set & remain fixed.
- The randomization helps/allows it to have complex dynamics making it possible to capture patterns in IP sequence
- The input is fed into the network → the reservoir neurons react to IP & pass info through connections.

→ dynamic state of reservoir achieved which represents the processed information from input sequence.
- O/P is generated by training connections from reservoir to output neurons using a learning algo like linear regression, ridge regression.
- training maps internal state of reservoir to desired output
- can efficiently handle long-term dependencies.
- computationally easy to train as compared to trad RNN.
- Working

1) Initialization:
Network is initialized by randomly assigning weights to the connections betⁿ the I/P neurons & reservoir, as well as within the reservoir.

2) Training
- set of input sequences with their corresp. desired output is presented to the network.
- network processes each input sequence & generates an internal state.

3) Reservoir dynamics
- input sequence is fed into I/P neurons, the values propogate through ther connection to the reservoir.
- reservoir neurons react to the I/P & pass info. through their Recurrent connections.
- dynamic behaviors helps to capture & process patterns in the I/P data

4) Output Generation.
- Internal state of reservoir is used as input to output neurons.
- output neurons transform internal state to desired output

5) Training output weights
- connections are trained using simple algo like L.R or RR.
- training process adjusts wts to find best mapping betⁿ internal state & desired output

5421
4316
1145

classmate
Date
Page

6) Prediction & Testing.

Keaksy Wriks

Long Short Term Memory.

- its solves the problems in RNN of Vanishing & exploding gradients.
- LSTM is a type of RNN.
- It is a more enhanced version of RNN.
- it permits information to endure
- it retains prior knowledge & apply it to process incoming data.
- long term dependency issues are prevented.
- output of previous step is input to the next step.
- addresses the issues of RNN long term dependencies, in which RNN is unable to predict words stored in long term memory.
- LSTM keeps information for a long time by default.
- can handle not only single data but also data streams.
- linked handwriting recoguition & speech recoguition.

Architecture of LSTM.

- LSTM consists of 4 NN & numerous ~~cell~~ memory blocks called cells in a chain structure.
- Conventional LSTM consists of input gate, forget gate & output gate.
- flow of information into & from the cells is controlled by these gates.
- Cell remembers these values over time intervals.
  ^ memory

X_t = i/p to current timestamp
U_f = wt. asso. with i/p
H_{t-1} = hidden state of prev.
W_f = Wt. matrix of hidden state.

memory cell has 3 gates.

memory
- Cells stores information
- Gates manipulate memory.

1) Input Gate.
- decides which input values should be used to change memory.
- Sigmoid func: determines whether 0 or 1 should be allowed through
- tanh func: assigns weight to data from -1 to 1.
- how much info. should be saved/stored from the current step in the memory cell.
- It takes into account the input state & hidden state

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

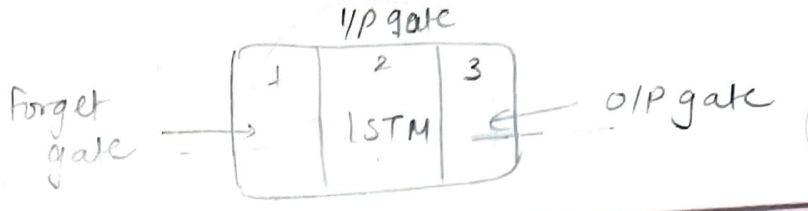
2) Forget gate.
- It decides which data should be discarded from the previous time step based on the previous hidden ~~step~~ state & current input.
- Sigmoid function is used to decide that.
- For each number in the cell state & input, it produces a no. between 0 (discard) & 1 (keep).

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

3) Output Gate.
- Controls how much information ~~the output gate~~ from the memory cell should be output to the next ~~cell~~ step/output.
- It combines current input + previous hidden state. = O/P.

$$O_t = \sigma(W_o \cdot [h_{t-1}, X_t] + b_o)$$
$$h_t = O_t \cdot \tanh(c_t)$$

4) Hidden State.
- carries info. across time steps.
- It passes info from one LSTM cell to other.



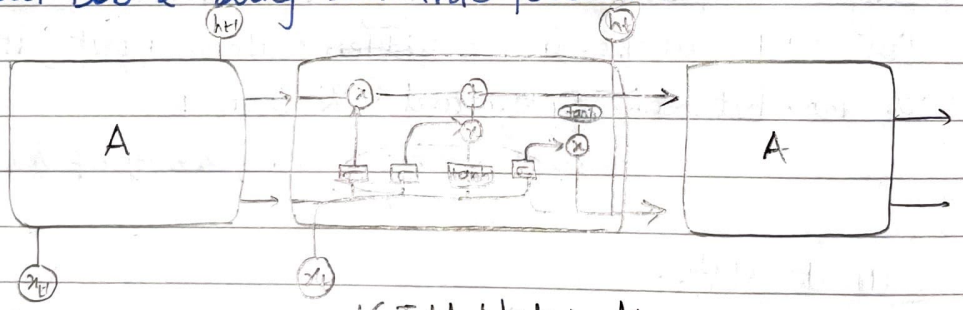
- acts like the memory of the cell/network.
- enables model to capture & propogate info over long sequences.

5) Cell-state.

- long term memory.
- responsible for carrying info over time.
- updated through IP & forget gates.
- this allows LSTM to selectively remember or forget info. from previous time steps.

Eg. Bob is a lovely person. Dan is nasty.

Here the first sentence clearly indicates that Bob is a lovely person & then switches to saying Dan is nasty. Now the LSTM system should be able to understand the switch between Bob & Dan. Here's where forget gate comes into picture. The forget gate enables it to forget ~~that~~ about Bob & bring Dan into focus.



LSTM Network

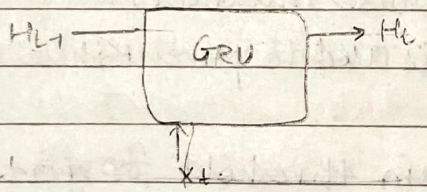
BIDIRECTIONAL LSTM.

- extension of LSTM architecture
- It introduces concept of processing sequential data in both backward & forward direction.
- BILSTM processes it both backward & forward simultaneously

- main idea is to capture data from both past & future contexts.
- LSTM can utilize info. from both past & future timestamps, allowing it to capture dependencies.
- Architecture has two separate LSTM networks
- one processes the sequence in forward direction & the other in backward.
- each network operates independently, has its own weights & hidden states.
- hidden states of both networks at each step are concatenated to find the output.

GATED RECURRENT UNITS.

- quite similar to LSTM.
- uses gates to regulate the flow of info like LSTM.
- quite new as compared to LSTM.
- similar architecture.
-



- It lacks distinct 'cell state' which is present in LSTM.
- quicker to train due to the architecture simplicity.
- accepts input (X_t) & hidden state (H_{t-1}) from ~~each~~ timestamp t-1 for each timestamp t.
- A new hidden state is output which is given to next step.
- Has 2 gates:

1) Reset Gate
- network's short term memory.
- also known as hidden state.

2) Update Gate.

- long term memory.

* GRU VS LSTM.

- GRU → 2 Gates, LSTM – 3.
- GRU → no internal memory, no output gate

Optimization for long Term Dependencies.

1) Clipping Gradient.

- used to mitigate the problem of exploding gradients but not vanishing gradients.
- When training a neural network, we calculate gradients which indicate how much each parameter should be adjusted to improve model's performance.
- Sometimes the gradients become too large & cause problems during training & cause instability.
- Gradient clipping is used to prevent these gradients from getting too big.
- We set a maximum threshold for gradient values.
- If any gradient exceeds this threshold, we scale it down to make sure it stays in the limit.
- This helps to keep training process stable.
- It also allows the model to learn more effectively.
- By controlling magnitude of gradients, gradient clipping ensures that the model does not get overwhelmed by extremely large updates during training.
- It is like putting a cap on the gradients to prevent them from ~~go~~ growing too much.

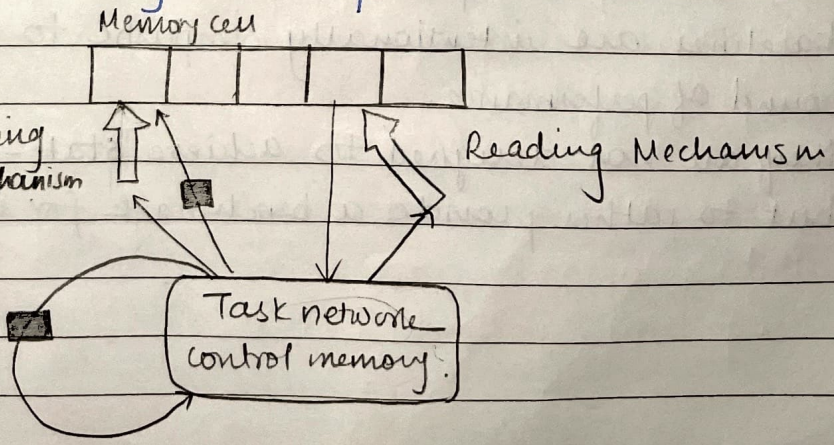
2) Regularizing.

- done to encourage information flow.
- it solves the problem of vanishing gradients which cannot be solved by gradient clipping.
- here the parameters are constrained to encourage info. flow.
- L1 or L2 regularization is used.
- These techniques add a regularization term to loss function during training.
- This penalizes large weights in the model.
- By penalizing the large weights, it forces the model to prioritize smaller weights, reducing the impact of less relevant features.
- This helps in encouraging info. flow by allowing the model to focus on the most relevant features.

EXPLICIT MEMORY.

- type of long term memory
- It is concerned with recollection of facts & events.
- also known as declarative memory.
- Information stored in memory cell may be ~~f~~ carried forward in ~~time due~~ & gradient carried bdckward in time without losing the content, if the content in memory cell is copied at many time steps.

(disapp
or
exploding)



- The above system is an RNN, despite the fact that task neural network may be feedforward or recurrent.
- Task network → select which memory address to read from or write to.
- Task network learns to control the memory, deciding where to read from & where to write within the memory.
- Due to explicit memory, models can learn tasks that standard RNNs or LSTM RNNs are unable to learn.

DEFAULT BASELINE MODELS.

- Baseline models are starting points or reference points for ~~start~~ evaluating the straightforward solution to a problem.
- It allows researchers to establish baseline level of performance which can be improved.
- choice of default baseline models depend on the specific problem domain & available data.
- eg.
  - Classification → (Binary) → Logistic Regression.
  - Regression → mean/median ~~for~~ of target value.
  - Clustering → K-means clustering

etc.

- these models provide a basic level of performance that can be used as a reference to evaluate effectiveness of more advanced techniques.
- baselines are intentionally simplistic to establish lower bound of performance.
- They are not designed to achieve state-of-art results but to rather provide a benchmark for improvement.

DETERMINING WHETHER TO GATHER MORE DATA.

- Determining whether to gather more data for an ML model involves assessing current data & considering factors that could indicate ~~that~~ the need of more data.

1) Evaluate model performance
   - assess performance of model on current dataset.
   - If performing well & meeting metrics, no new data reqd.
   - If performance is not good, it may ~~lack~~ indicate lack of data. or insufficient rep. of underlying patterns.

2) Analyze data set size
   - If dataset = small, gather more data.
   - dataset = large → provide more insights, no data reqd.

3) Assess data quality.
   - examine quality & diversity of dataset.
   - If imbalanced, or has outliers, missing values, gathering more data helps to overcome those adversities.

4) Cost & feasibility.
   - Assess practicality & cost with acquiring data.
   - factors- collection methods, time constraints, financial resources etc.

5) Domain complexity
   - Intricate tasks need more data to capture underlying trends.

Unit 5
Deep Generative Models.

Introduction

* - Deep Generative models.
- class of models which learn to generate new data that is similar to the training data it has been trained on.
- They learn the underlying probability distribution of training data to generate new samples.
- built using ~~a~~ DNN, which have multiple layers of interconnected units. which learn increasingly abstract representation of data.
- Types : Variational Autoencoders (VAEs)
  Generative Adverserial Networks (GANs)
- image synthesis, text generation, speech synthesis etc.

Boltzmann Machine.

- kind of recurrent neural network
- nodes make binary decisions & are present with certain biases.
- Many Boltzmann machines together → deep belief system.
- designed to model & learn probability distribution of ~~data~~ a set of binary valued input data.
- aim to find ~~stat~~ optimal state of network which maximizes the probability of generating observed data.
- Architecture consists of binary valued visible units & binary valued hidden units.
- Every unit is connected to other unit by symmetric weights.
- The network is trained using unsupervised algorithm called contrastive divergence.
- During training, machine learns to adjust weights to

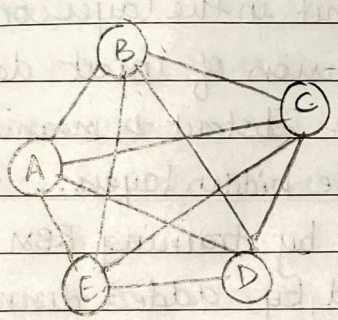
Visible units = input layer.
hidden units = hidden layer.

small updates.

capture the stat. dependencies & patterns in data.
- Does that by iteratively updating state of each unit based on the state of other units.
- Training has two phases: positive phase & negative phase

1) Positive phase
- In positive phase the the visible unit values are fixed to the values in the training data
- The states of the hidden units are updated.

2) Negative phase.
- In negative phase, states of both hidden & visible units are sampled from the known distribution to generate fantasy data.
- The learning algo. then adjusts weight according to the difference between the observed data & generated fantasy data.



* Features / Characteristics.

1) There are no connections bet^n layers.
2) They employ symmetric & recursive structures.
3) Algo. for its unsupervised learning.
4) In learning process, RBMs try to link low energy state with high one & so one.

Types

1) Deep Belief Networks
2) Restricted Boltzmann Machines : have restricted connectivity. pattern bet^n hidden & visible
3) Deep Boltzmann Machine.

Deep Belief Networks.

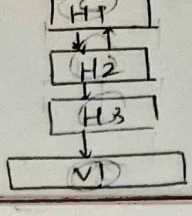
- It is a stack of Restricted Boltzmann Machines.
- powerful generative model.
- they are inspired by the human brain, designed to find patterns & make sense of complex data.
- One RBM takes input, works on it & produces ~~an~~ output which is further used as input to the next RBM in the sequence.
- As it is a generative model, it can be used in a supervised or unsupervised setting.
- Architecture:

i) Visible layer - represents input data. such as images text.
Each unit in the layer corresponds to a feature or dimension of input data.

ii) Hidden layers - capture abstract & meaningful features.
multiple hidden layers.
- created by training RBM.

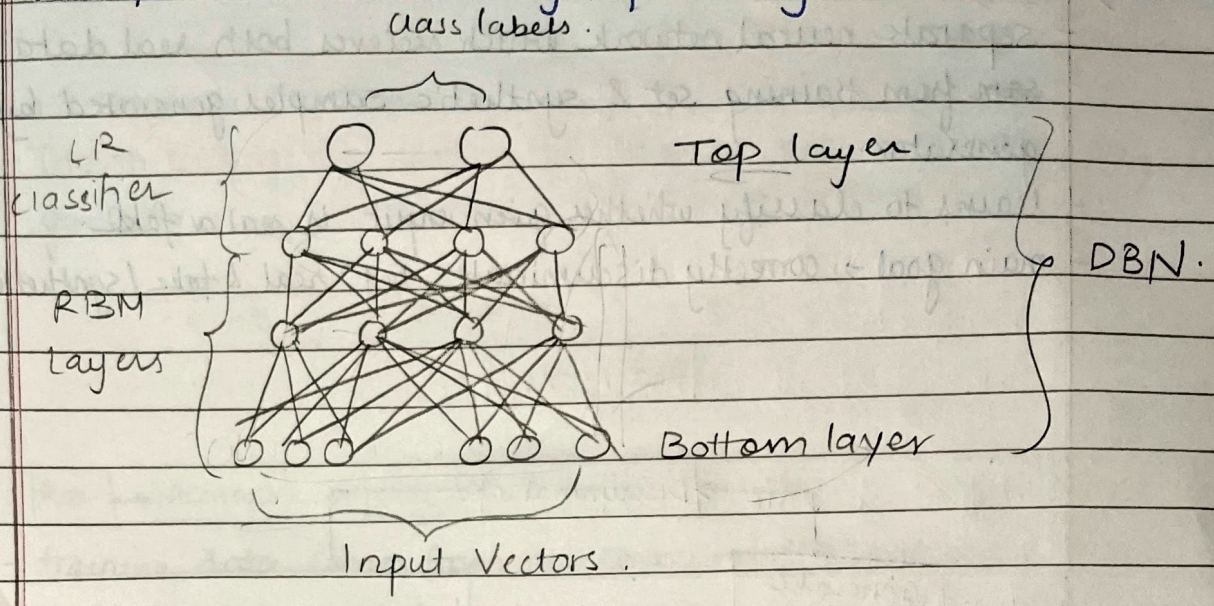
iii) Output layer - ~~created by~~ added during fine-tuning phase
- It is a softmax layer for classification.

- DBN learn by first pre-training each layer separately using method called RBM.
- Helps networks to uncover patterns & trends & extract useful information.
- After pre-training, the model is fined tuned where all layers are trained together using a supervised learning algorithm.
- Fine-tining adjusts the weights bet^n neurons making network better at ~~proce~~ predicting / classifying.



overview.

- used for tasks such as image, speech recognition.



Generative Adverserial Network.

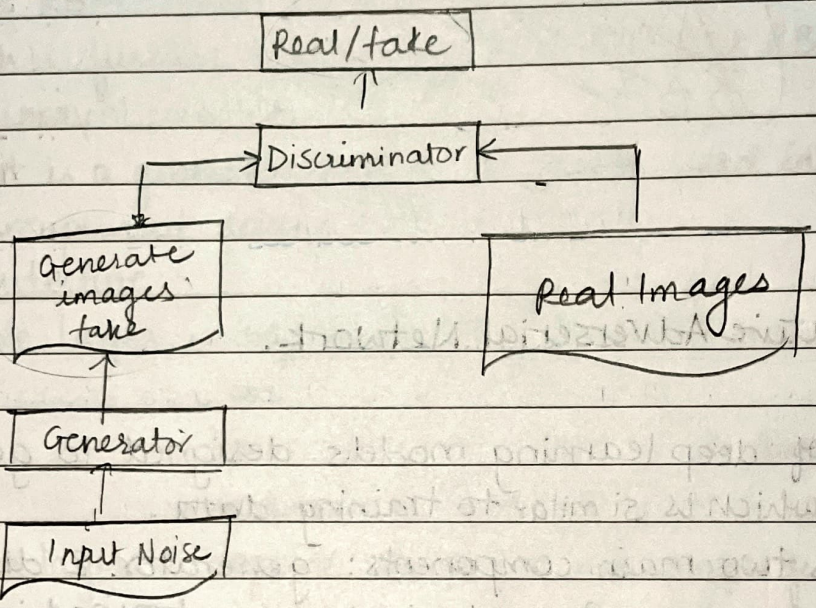
- class of deep learning models designed to generate new data which is similar to training data.
- It has two main components: generator & discriminator.
- The generator & discriminator are trained in a competitive manner.
- The generator aims to produce realistic samples to fool discrim. The discriminator improves its ability to differentiate between real & fake samples.
- Architecture.

1) Generator.

- takes random noise/latent input & produces/generates synthetic samples
- consists of one or more layers of NN which learn to transform the input noise to meaningful data representations.
- goal is to generate realistic data samples which look & are undistinguishable from real data.

2) Discriminator.

- separate neural network which recieves both real data & ~~some~~ from training set & synthetic samples generated by generator.
- learns to classify whether given input is real or fake
- main goal → correctly discriminate bet" real & fake / synthetic samples

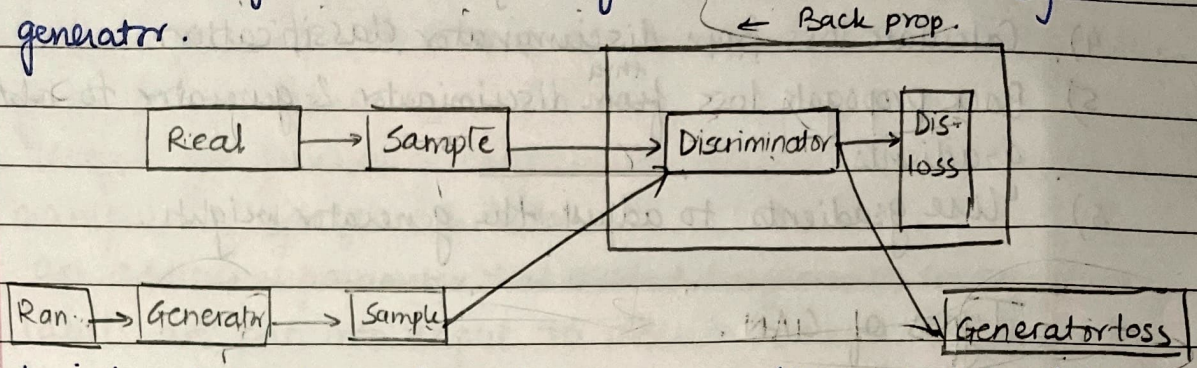


- During training the generator generates fake samples & discriminator evaluates & provides feedback on the authenticity.
- As the training progresses, generator gets better at generating fake samples (more realistic), while discriminator becomes more accurate at distinguishing them from the real ones.
- Applications:

1) realistic human face images
2) generate artwork
3) enhance image quality
4) text generation
5) Video generation

Discriminator Network

- is simply a classifier.
- Tries to distinguish real data from the data created by the generator.



- training data comes from two sources — the real data & the synthetic data created by the generator.
- The discriminator connects to 2 loss functions
- It ignores the generator loss & takes into account the discriminator loss.
- Generator loss is used in generator training.

1) Discriminator classifies both real & fake data
2) The dis· loss penalizes it when it misclassifies a real instance as fake or fake instance as real.
3) It updates weights through back propagation from loss func.

Generator Network.

- gen· part creates fake realistic data samples by incorporating feedback from discriminator.
- Generator training → tighter bond (intg.) bet^ generator & discriminator.
- Portion of gan training generator
  1) random input.
  2) generator network that converts input into data sample.
  3) discriminator network, classify data
  4) discriminator output.
  5) generator loss, which penalises generator for failing to fool discrim.

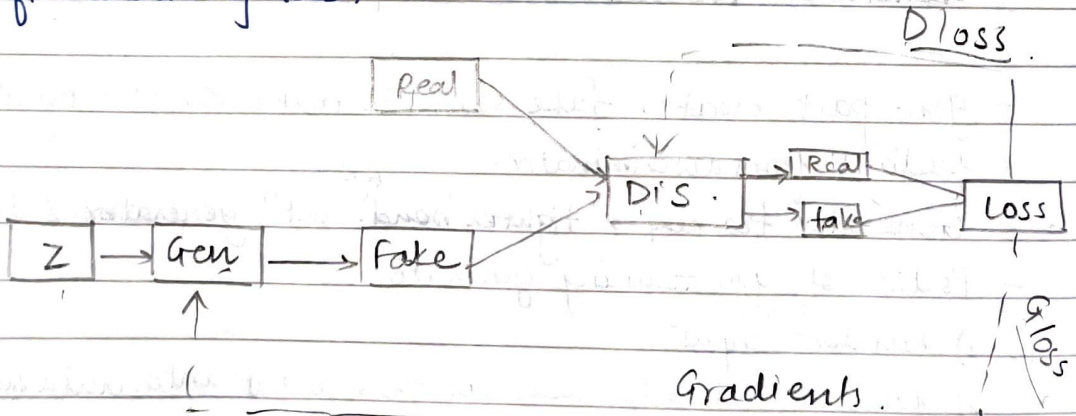
Training generator:

1) Sample random noise
2) Produce generator output
3) Get discriminator to classify whether 'Real' or 'Fake'.
4) Calculate loss from disciminator classification
5) Back propogate loss thru discriminator & generator to obtain gradients
6) Use gradients to adjust the generator weights.

Types of GAN.

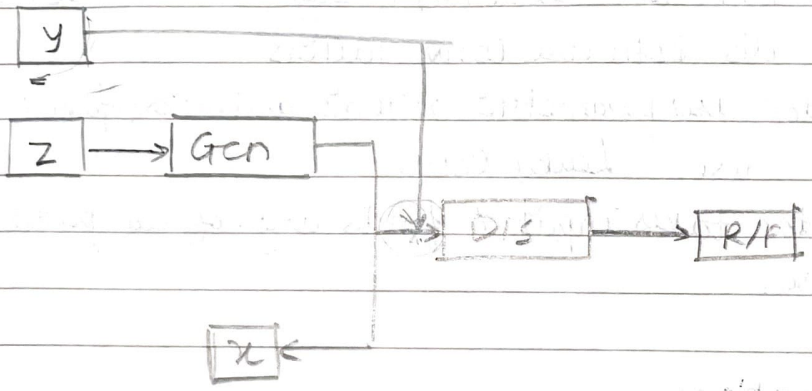
1) Vanilla GAN.

- simplest type of GAN.
- Generator & discriminator → simple multi-layer perceptrons
- the algo. is very simple.
- It tries to optimize mathematical eqn using stochastic gradient descent.
- generator : captures data distribution
- discriminator : predicts to which class does the input belong to
- feedback is sent to both generator & discriminator after calculating loss function.



2) Conditional GAN.

- adds extra information like class labels or attributes to guide generation process.
- allows generator to create specific samples based on the given ~~parab~~ labels & attributes.
- useful for generating samples with desired characteristics or manipulating specific attributes.
- an additional parameter Y is added to generate corresp. data.
- labels are put into input to discriminator in order for the discriminator to help distinguish real data from fake one.



3) Deep convolutional GAN.

- most popular & successful implementation of GAN.
- composed of conv. Nets in place of multilayer perceptrons.
- Nets are implemented without Max Pooling.
- Max pooling is replaced by conv stride.
- layers are not fully connected.
- commonly used for generating images as conv. layers can understand spatial rel^{n} in images.

4) Laplacian Pyramid GAN.

- linear invertible representation of image.

- LAPGAN uses multiple no. of generators & discriminators networks
- It also uses different layers of Laplacian pyramid.
- used as it creates high level images.
- first image is downsampled ~~or~~ at each layer & upscaled in backward pass.

5) Super Resolution (SRGAN)

- neural network + adversarial network
- produces higher resolution images.
- tranforms low resolution image to high resolution.
- Gen net & dis. both use conv. layers
- Generator uses parametric relu as activation function
- Disc. uses Leaky ReLU.
- Generator takes input & finds avg. of all possible sol? pixelwise.

Applications.

1) Healthcare

- By comparing patient's scans & images from dataset which have healthy organs, GAN can spot abnormalities in the patient's scan
- Rapid & precise malignant tumor analysis/diagnosis → GAN.

Drug research

- Molecular structures of drugs used to target & treat diseases can be created using GAN.

- To find the subs. that can aid in treatment need not be found by researchers manually in the database. Program helps to cut down time needed for med. research & development by automatically identifying such chemicals.

3) Generate examples for image datasets.
4) Generate photographs of Human faces
5) Generate realistic photos
6) Image-to image translation
7) Text-to-image
8) Semantic-image-to-photo.
9) Photos to Emoji
10) Face Aging
11)

X