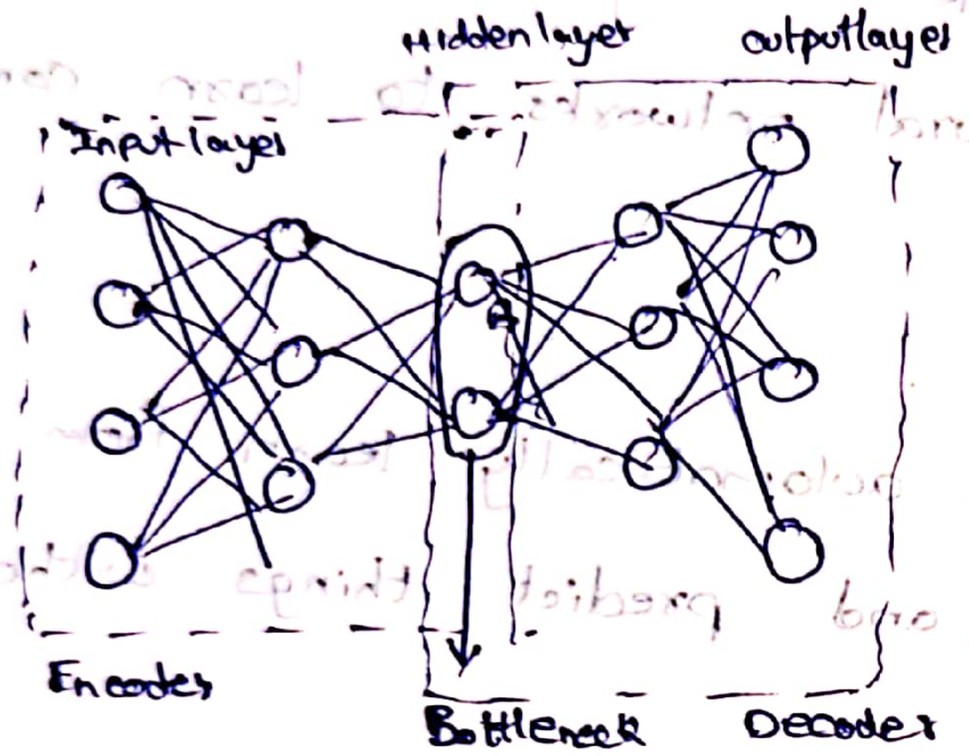
Autoencoder :-

An Auto encoder is a type of Artificial neural network that is used for unsupervised learning. The aim of an autoencoder is to learn a lower-dimensional representation (encoding) for a higher-dimensional data typically for dimensionality reduction, by training the network to capture the most important parts of the input image.



eg of autoencoder's

- → Dimensionality Reduction
- → Feature extraction
- → Image Denoising, Compression
- → Anomaly Detection

i) **Encoder** :- The encoder part of the autoencoder is responsible for the compressing the input data into lower-dimensional representation. The encoder typically consists of one or more hidden layers that use non-linear activation functions to transform the input data into a lower-dimensional space.

ii) **Bottleneck** :- The Bottleneck layer is most important of an autoencoder as it connects the encoder and decoder parts of an autoencoder. This layer typically has a lower dimensionality than the input and output layers and is used to create a compressed representation of the input data.

iii) **Decoder**:- The decoder part of the autoencoder is responsible for the reconstructing the original input data from the compressed representation created by the encoder. The output layer of decoder should have the same dimensionality as the input layer.

You need to set 4 hyperparameters before training an autoencoder:

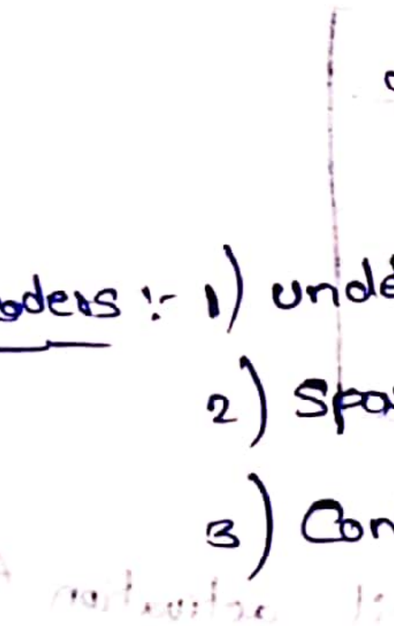
i) **Code size**: most imp hyperparameter used to tune the autoencoder. The bottleneck size decides how much the data has to be compressed.

ii) **Number of layers**:- This indicates depth of encoder & decoder. While a higher depth increases model complexity, a lower depth is faster to Process.

iii) **No. of nodes per layer**: defines the weights we use per layer. Typically the no of nodes in an encoder decrease till the bottleneck and increase in an decoder from the bottleneck.

iv) **Reconstruction Loss**:- The loss function we use to train the autoencoder is highly dependent on the type of input and output we want the autoencoder adapt to.

Eg of autoencoders :- 1) undercomplete autoencoder
2) Sparse autoencoder
3) Contractive autoencoder
4) Denoising autoencoder
5) Variational autoencoder



Variational autoencoder (VAE):- is a type of autoencoder that is designed to learn a probability distribution over the compressed representations. Instead of encoding the input data into a fixed point in a latent space, the VAE encodes the input data into probability distribution over the latent space. Used for data compression, synthetic data creation, etc...

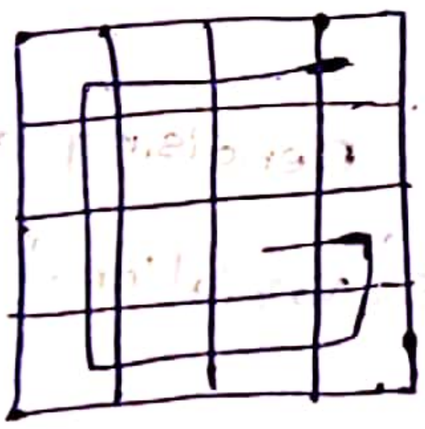
A Deep autoencoder is simply an autoencoder with multiple hidden lay in both the encoder and decoder. It is capable of learning more complex representations of input data compared to a shallow autoencoder with only one or two hidden layers.

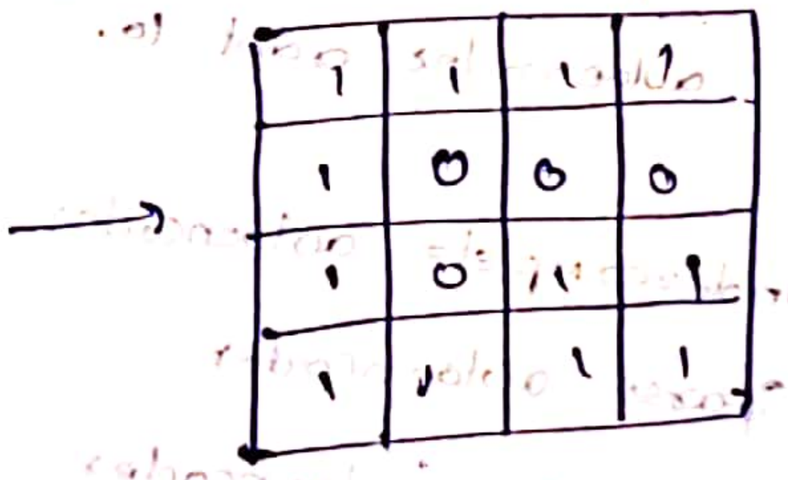
CNN :-

CNN stands for convolutional neural network, which is a type of AN Commonly used for image and video processing tasks, such as object recognition, scene reconstruction, and image classification.

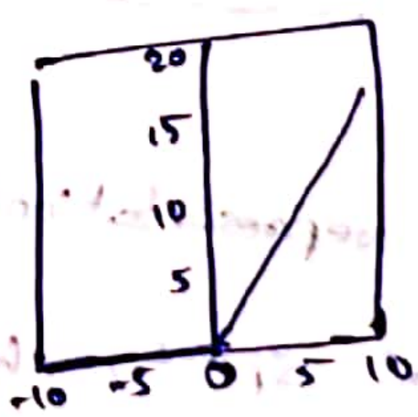
A CNN consists of multiple layers, each designed to perform a specific operation on the input data. The four import layers in CNN are:

i) Convolution layer :- This layer has multiple filters that perform the convolutio operation to extract valuable features from an image. Eve image is considered as matrix of pixel values, whose values is either 0(or) Each filter slides over the image matrix and perform element wise multiplication & sum up the results to produce single value in feature map.



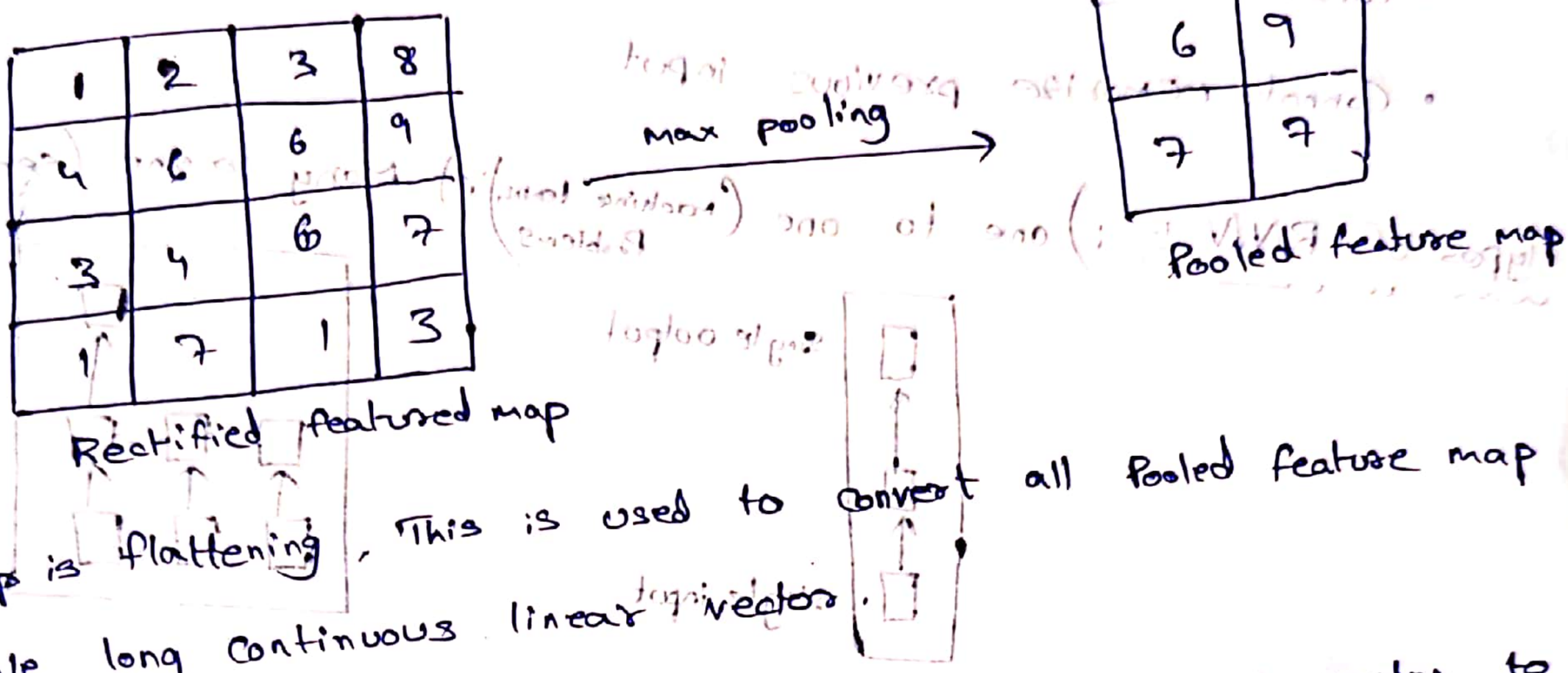


ii) RELU layer :- This layer applies rectified linear unit activation function to the output of the convolution layer. Relu sets all -ve values to 0 & +ve values remains unchanged.



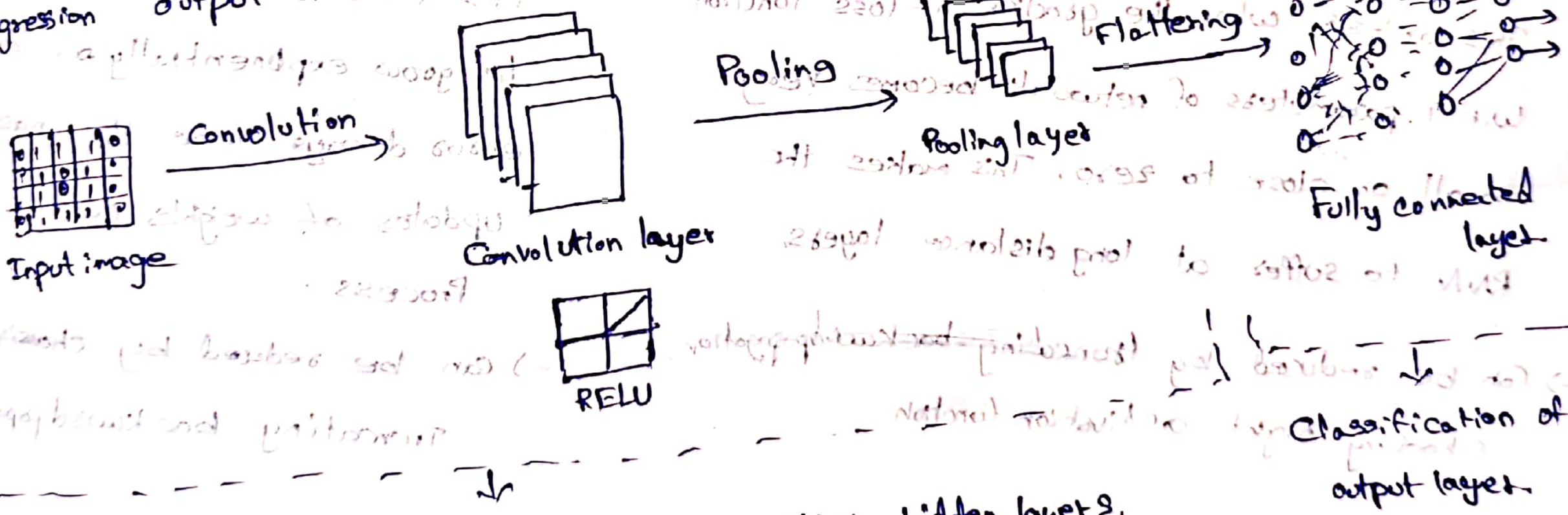
$R(z) = \max(0, z)$

iii) Pooling layer :- This layer reduce the spatial size of the featured map by applying a pooling function, such as max pooling, avg pooling to reduce dimensionality.



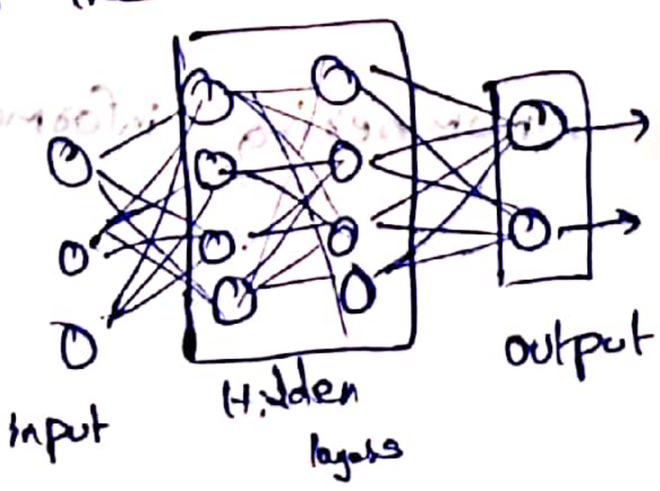
The next step is flattening. This is used to convert all Pooled feature map into a single long continuous linear vector.

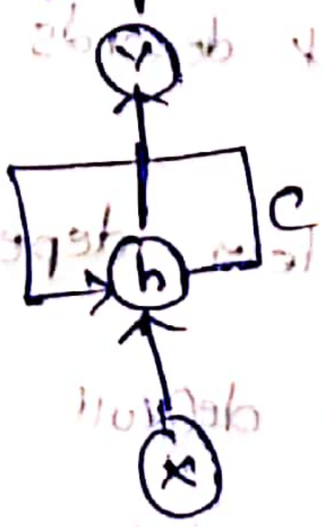
Fully connected layer :- This layer uses this single long linear vector to Connect with every single neuron of the next layer. The output of this generated NN is the final classification decision (or) regression output of the network.



Feature extraction in multiple hidden layers.

RNN :- This is a type of ANN in which the output of a network is feed back to the network as input in order to predict the output.

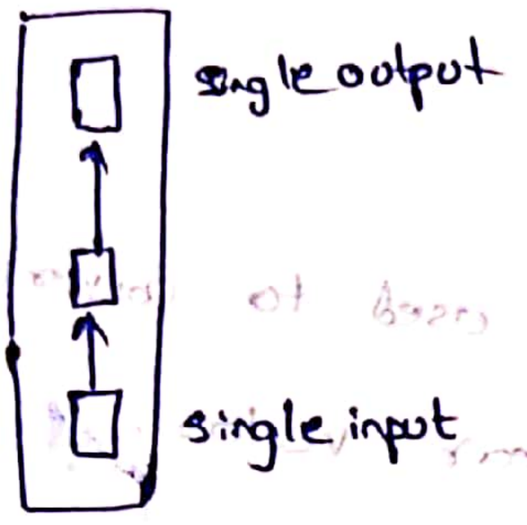


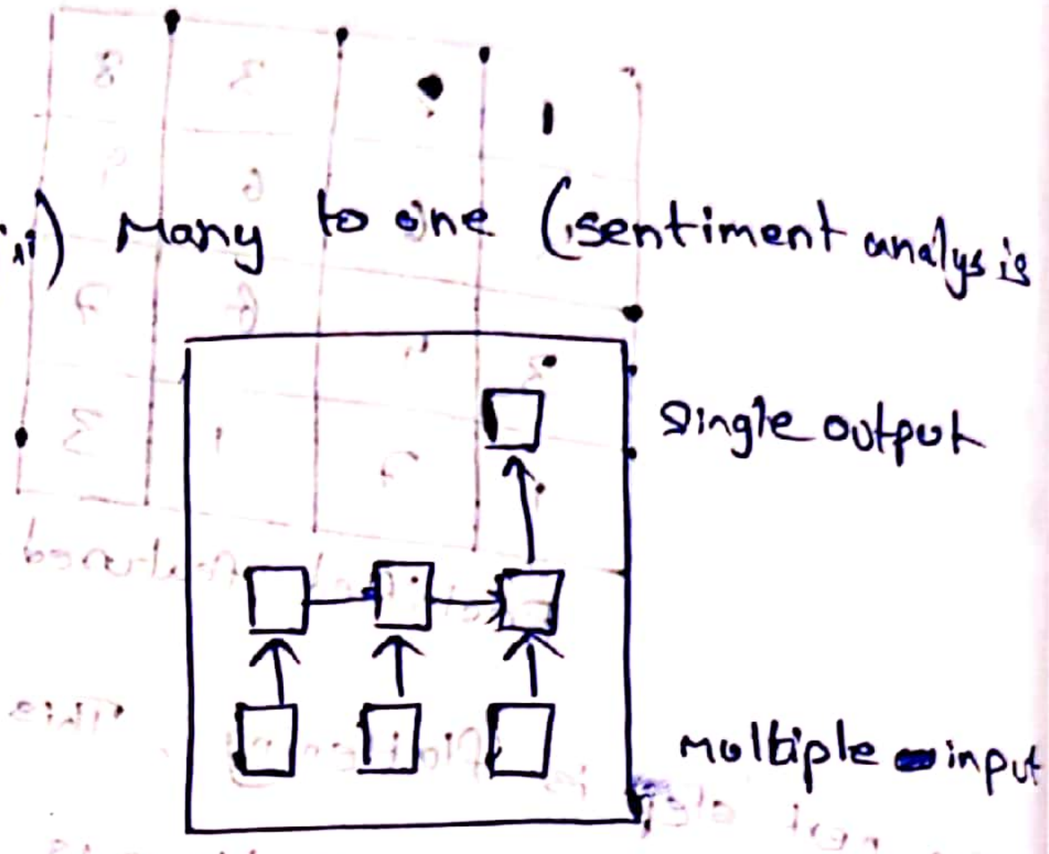


→ RNN were created because, feed-forward NN having following drawbacks

- Cannot handle sequential data
- Considers only the current input
- Cannot memorize previous input

Types of RNN :- i) one to one (machine learning problems) ii) Many to one (sentiment analysis





similarly iii) one to many (image caption) & iv) many to many. (machine translation)

Vanishing gradient problem

This arises when the gradient of loss function W.R.T parameters of network becomes very small or close to zero. This makes the RNN to suffer at long distance layers.

→ Can be reduced by ~~truncating backward~~ propagation.

Choosing right activation function

Exploding gradient problem

This arises when the slop tends to grow exponentially as large error decays cause the massive updates of weights during training Process.

→ Can be reduced by ~~choosing right~~ Truncating backward propagation

→ LSTMN → Long short-Term memory network.

eg:- I have been in Spain for last 10 years ... I can speak fluent ____

here, the prediction work depends on ~~for~~ information from previous words.

→ Capable of learning long-term dependencies by remembering information for long periods is the default behavior.

→ RNN is used for Text-to-speech conversions.

Bayes Rule :-

$$P(A|B) = \frac{P(B|A) \cdot P(A)}{P(B)} \quad \text{(or)} \quad \frac{P(B|A)\,P(A)}{P(B|A^-)\,P(A^-) + P(B|A^+)\,P(A^+)}$$

$P(A|B) \rightarrow$ probability of event A occuring, given event B has occurred

$P(B|A) \rightarrow$ " " " B given A "

$P(A) \rightarrow$ Probability of event A

" " B.

$P(B) \rightarrow$ " " B.

Gradient Descent algorithm :-

is a popular optamization algorithm. used in ML & DL to minimize the cost function. (linear regression)

- starts with inital set of parameters and updates them iteratly.
- It adjust the weights in the direction of -ve gradient of cost function.
- This is very easy to implement.

variants of gradient descent-

i) Stochastic gradient descent (SGD): only one training example is used to compute the gradient and update the parameter at each iteration. (may lead to noisy updates)

ii) Mini-batch GD : small batch of examples are used to compute the gradient and update the parameter at each iteration. (better than batch & SGD)

iii) Momentum-based GD : A momentum term is added to gradent update to smooth out the updation process.

iv) Adagrad : Adaptive learning rate for each parameter based on historical gradients

Generative Adversarial Network (GAN):-

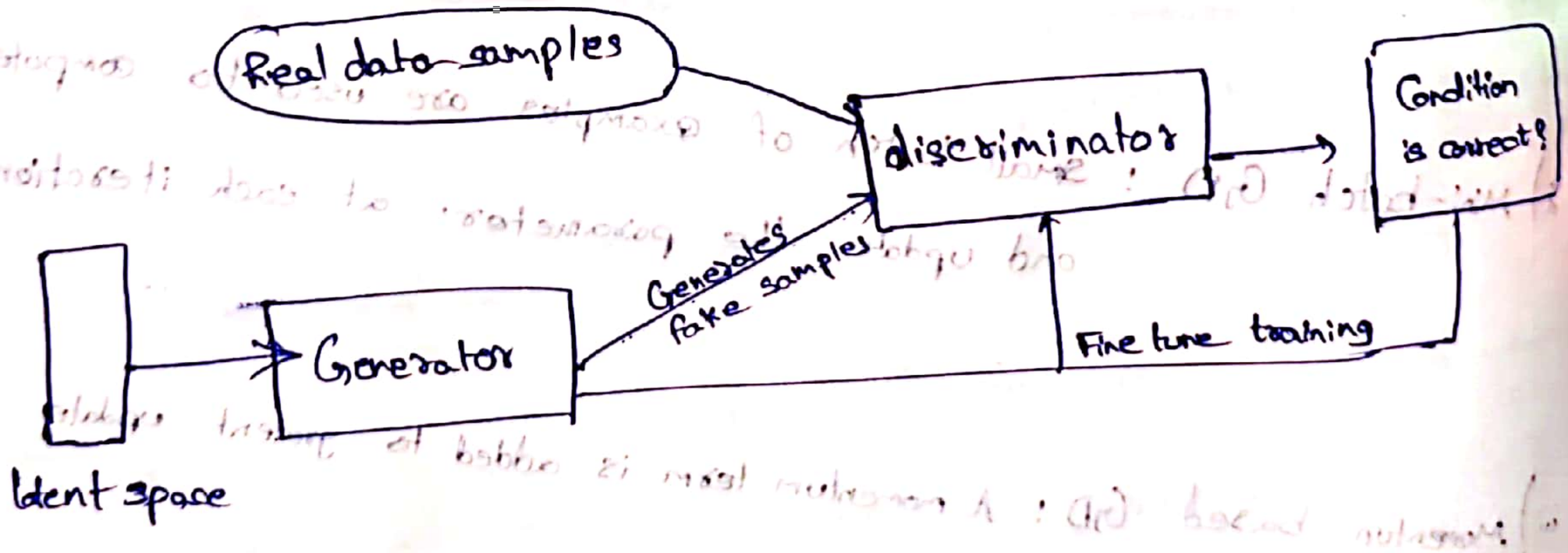
GAN is a deep learning model consisting of two neural networks: The generator and the discriminator.

Generator: The generator generates new data similar to the training data. it takes random noise vector as input and generates a new data sample as output.

Discriminator: The discriminator distinguishes between the real and generated data. It takes data sample as input and produce probability score as output, indicating wheather the input is real or generated.

The generator & discriminator are trained together in an adversarial manner, where the generator tries to generate realistic samples to fool the discriminator, and discriminator tries to correctly distinguish b/w the real & generated samples.

Applications of GAN :- image and video synthesis, data augmentation and anomaly detection.



et input

(a) Perceptron networks that implements AND function:-

the output "y" is calculated by calculating net input y_{in} and applying activation function to it.

$$y_{in} = b + \sum_{i=1}^{n} x_i w_i$$

$$y = f(y_{in}) = \begin{cases} 1 & \text{if } y_{in} > \theta \\ 0 & \text{if } -\theta \le y_{in} \le \theta \\ -1 & \text{if } y_{in} < -\theta \end{cases}$$

here θ is threshold.

$w(new) = w(old) + \alpha t x_i$ (α is learning rate),

Eg:- $w_1 = 1.2$; $w_2 = 0.6$, Threshold = 1 & learning Rate $\eta = 0.5$

A	B	A∧B
0	0	0
0	1	0
1	0	0
1	1	1

for Case i:- A=0, B=0 ; Target 0.

$w_i x_i = 0 \times 1.2 + 0 \times 0.6 = 0 = \text{Target}$

Case ii:- A=0, B=1 ; Target 0.

$w_i x_i = 0 \times 1.2 + 1 \times 0.6 = 0.6 = 0 = \text{Target}$

Case iii:- A=1; B=0; Target 0

$w_i x_i = 1 \times 1.2 + 0 \times 0.6 = 1.2 = 1 \neq 0 = \text{Target } 0$

Perceptron training rule :- $w_i = w_i + \eta (t - \text{actual output}) x_i$

$x_i = a$ in w_1

$x_i = b$ in w_2

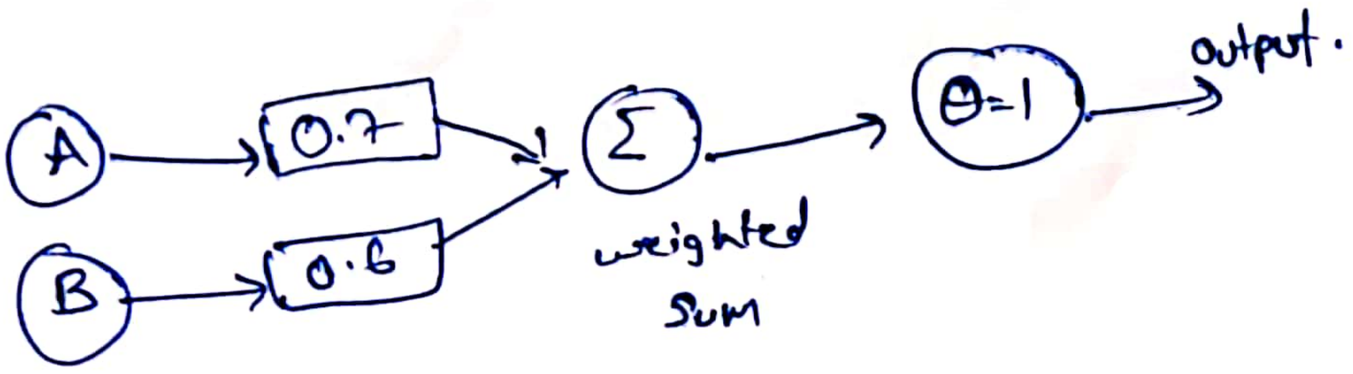
$w_1 = 1.2 + 0.5 (0-1) 1 = 0.7$

$w_2 = 0.6 + 0.5 (0-1) 0 = 0.6$

now $w_1 = 0.7$; $w_2 = 0.6$, Threshold = 1 & learning rate $\eta = 0.5$.

similar to above process we will go. Case-by-Case, in every case we will get the calculated output = actual output, so there is no weight updation

so. $w_1 = 0.7$ & $w_2 = 0.6$ are final weights.



Confusion matrix :- It is a matrix used to determine the performance of the classification model for a given set of test data.

Predicted values \ Actual values	Positive	Negative
Positive	TP	FP
Negative	FN	TN

$$\text{Accuracy} = \frac{TP+TN}{TP+FP+FN+TN}$$ (no. of correct predictions made by the classifier ratio with all other)

$$\text{Error Rate} = \frac{FP+FN}{TP+FP+FN+TN}$$ ("incorrect")

$$\text{Precision} = \frac{TP}{TP+FP}$$ (ratio of actual correct output provided by correct o/p predicted)

$$\text{Recall} = \frac{TP}{TP+FN}$$ (ratio of actual correct output by overall correct predicted)

$$\text{F-Score} = \frac{2 \times \text{Recall} \times \text{Precision}}{\text{Recall} + \text{Precision}}$$ (F.score is max. when Precision = Recall)

a) Underfitting and Overfitting in Machine Learning:-

* Underfitting:-

Underfitting occurs when a machine learning model does not learn the training data well enough. This can happen for a number of reasons, including:

- The model is too simple and cannot capture the underlying patters in data
- The model is not trained on enough data.
- The data is noisy and corrupted.

Underfitting can be identified by looking at the model's performance on the training data and the test data. If the model performs poorly on both training and testing data, then it is likely that the model is underfitting.

* Overfitting:-

Overfitting occurs when a machine learning model learns the training data too well. This can happen for a number of reasons, including:

- The model is too complex and learns the noise in the data as well as the underlying patterns.
- The model is trained on too much data.
- The data is not representative of the real world.

Overfitting can be identified by looking at the model's performance on training and testing data. If the model performs well on the training data but poorly on the test data, then it is likely that the model is overfitting.

* Preventing Underfitting and Overfitting:

There are a roll of things that can be done to prevent underfitting and overfitting, including:

- Selecting the right model: The complexity of the model should be chosen based on the amount of training data and the complexity of the problem.

- Ensuring that the training data is representative of the real world: The training data should be as large as possible and should be representative of the data that the model will be used on.
- Using regularization: Regularization is a technique that can help to prevent overfitting by adding a penalty to the model's loss function.
- Use cross-validation: is a technique that can be used to evaluate the performance of a model on unseen data. This can help to identify the problem with underfitting & overfitting.
- Start with a simple model: It is often better to start with a simple model and then add complexity as needed. This can help to prevent overfitting.
- Be patient: It may take some time to find a model that works well.

By carefully selecting the model, amount of training data and the regularization technique, it is possible to build a model that is both accurate and generalizable.

a)

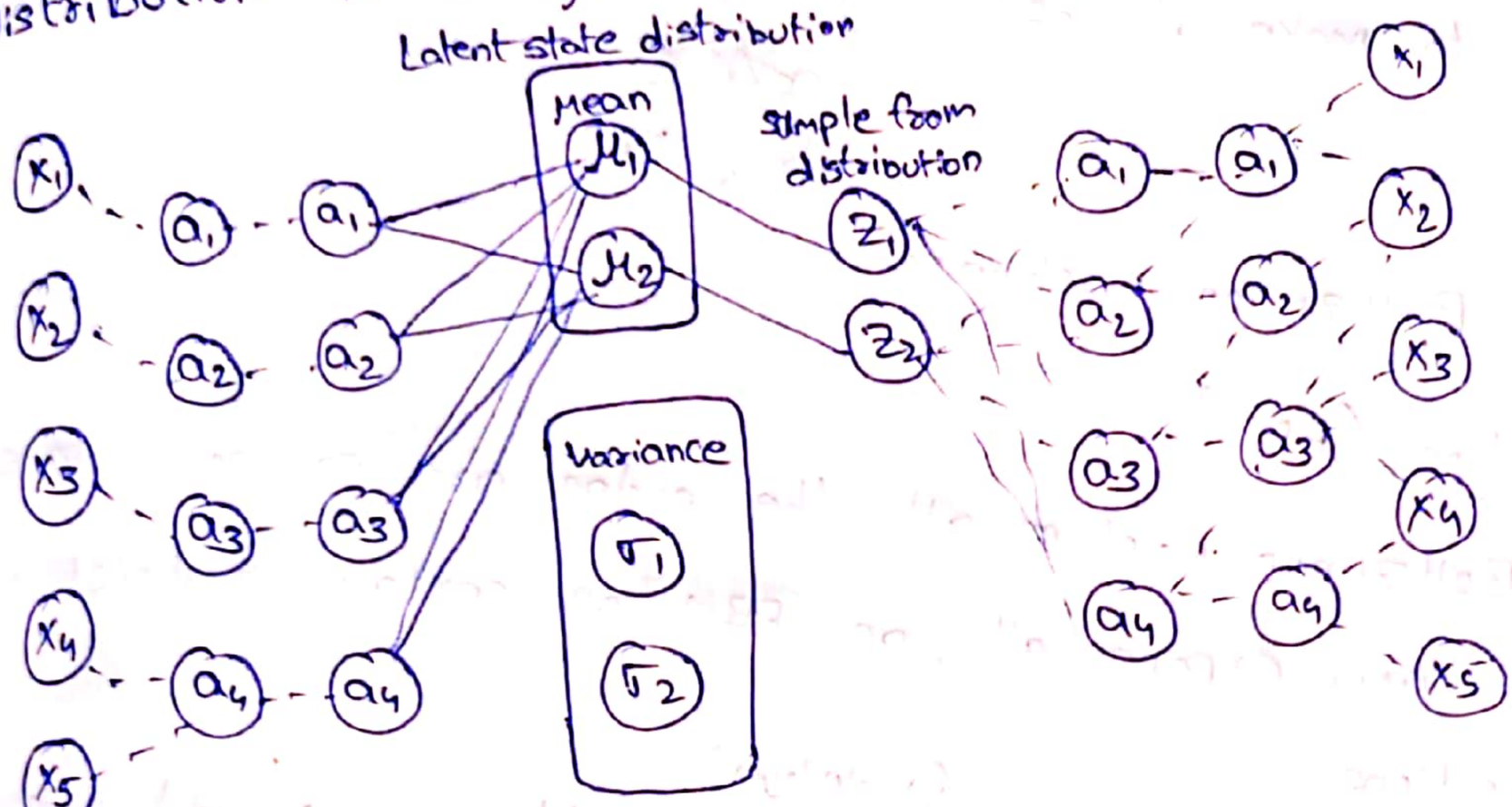
Feature	CNN	RNN
Definition	CNN stands for Convolution Neural Network	RNN stands for Recurrent Neural Network
Layers	Consists of convolution layers	Consists of Recurrent layers.
Tasks	Image classification, object detection	Speech recognition, NLP
Type of data	Spatial data	Temporal data
Strengths	good at extracting features from spatial data	good at learning long term dependencies
weaknesses	Not as good at processing temporal data	Not as good at extracting features from spatial data
Input/output	The network takes fixed-size inputs and generates fixed size outputs.	RNN can handle arbitrary input/output lengths.
Activation function	Rectified linear unit (ReLU)	tanh, sigmoid
Backpropagation	Backpropagation through time (BPTT)	Backpropagation.
Vanishing gradient Problem	Less prone to	More prone to
full definition	CNN is a type of feed-forward ANN with variations of multilayer perceptron designed to use minimal amount of preprocessing	RNN, unlike feed-forward neural network can use their internal memory to process arbitrary sequences of inputs.

Variational autoencoder: (VAE)

is a type of neural network that can learn to represent data in a latent space. The latent space is a lower-dimensional space that captures the most important features of the data. This makes it possible to generate new data that is similar to the training data, even if the new data has never been seen before.

VAEs are trained using a technique called variational inference

Variational inference is a way of finding the parameters of a probability distribution that best fit a set of data. In the case of VAEs, the probability distribution is a Gaussian mixture model.



VAE can be used for Data compression (helps in storing and transmitting data more efficiently); Data generation (Can be useful for creating synthetic data for training other ML models); Feature learning (Can be useful for tasks such as classification and regression).

Deep Autoencoder: A deep autoencoder is composed of two symmetrical deep-belief networks having four to five shallow layers. One of the networks represents the encoding half of the net and the second network makes up the decoding half. They have more layers than a simple autoencoder and thus are able to learn more complex features. The layers are restricted Boltzmann machines, the building blocks of deep-belief networks. These have same applicati as VAE, req large data set, difficult to train, computationally expensive.

Boltzmann machine:

is an unsupervised deep learning model in which every node is connected to every other node. It is a type of recurrent neural network, and the nodes make binary decisions with some level of bias. These machines are not deterministic deep learning models, they are stochastic (or) generative deep learning models. They are representations of a system which has visible nodes & Hidden nodes.

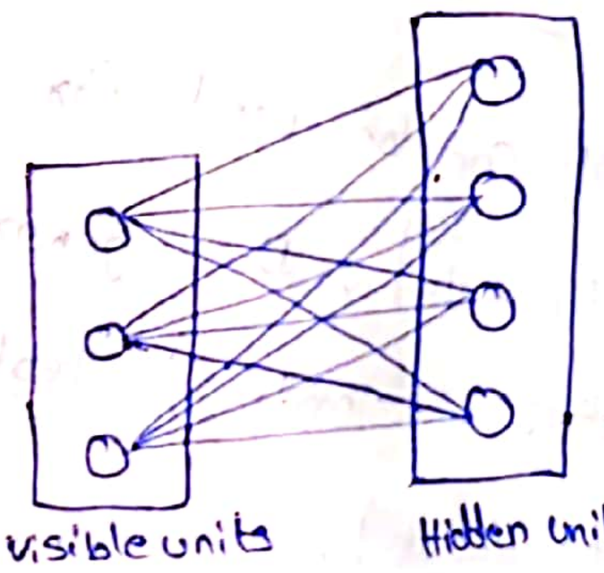
A Boltzmann machine is made up of a learning algorithm that enables it to discover interesting features in datasets composed of binary vectors. The learning algorithm tends to be slow in networks that have many layers of feature detections but it is possible to make it faster by implementing a learning layer of feature detectors.

These are three types of Boltzmann machines:-

i) Restricted Boltzmann Machine (RBM):-

while in a full Boltzmann machine all the nodes are connected to each other and the connections grow exponentially, an RBM has certain restrictions with respect to node connections.

RBM has two layers namely input layer (visible layer) & hidden layer and consists of nodes which are connected across different layers but no two nodes of the same layer are linked.



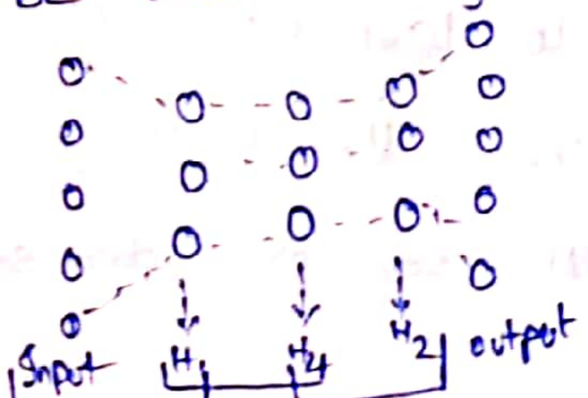
-> limited connection b/w nodes, makes it fast than BM; hidden layer's activations can be included in other models to boost performance.

-> CD-* algorithm is used instead of backpropagation algorithm; challenging to calculate the energy gradient function, tough to modify weights.

-> used in Collaborative filtering, Image & video processing, NLP, Anomaly detection.

ii) Deep Belief Network (DBNs):

In DBN, multiple RBMs are stacked, such that the outputs of the first RBM are the inputs of the subsequent RBN, and the final output is used for supervised learning tasks such as Classification or regression. The connections between layers are directed. A deep belief network can either be trained using a Greedy layer-wise Training Algorithm or a wake-sleep Algorithm.

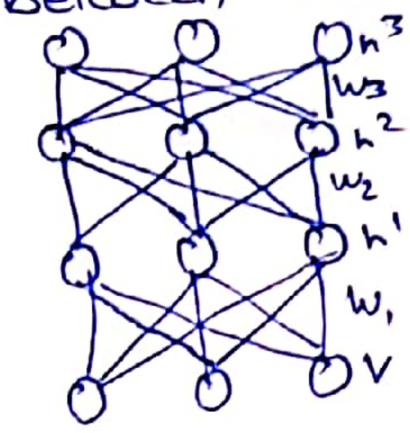


-> Provides high performance even with huge data, DBN avoids time taking techniques, It is flexible to new problems in future, and gain performance by adding layers.

-> requries huge data to perform better techniques, expensive to train because it has complex data models, requries hardware and skilled people & requries classifiers to grasp the output.

iii) Deep Boltzmann Machines (DBMs):

are very similar to Deep Belief networks. The diff b/w DBN & DBM is that while connection between layers in DBNs are directed, in DBMs, the connections within layers, as well as the connections between the layers, are all undirected.



-> Consists multiple layers of hidden units, which allows them to learn more complex problems than normal Boltzman machine.

-> Using multiple hidden layers, it learns hidden patterns in complex problems in very less time, easy of training for small data.

-> slow to train, especially if the network is deep, sensitivity to hyperparameters; computational complexity, if the network is deep.

-> used for image classification, NLP, speech recognition, Recommendation system.

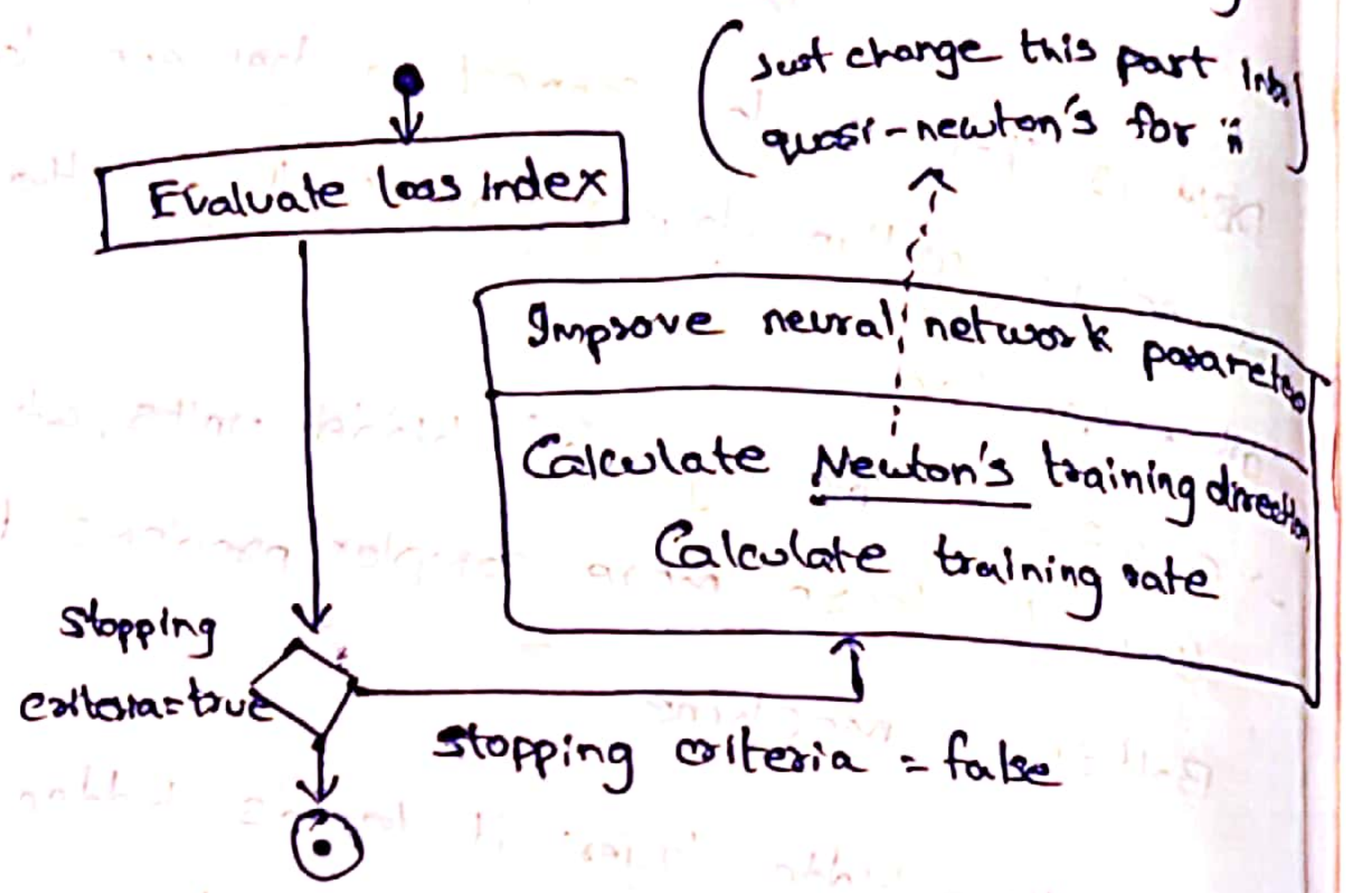
Algorithm	Pros	Cons
Gradient Descent	- simple to implement - can be used for any function - can be used for any no. of data points	- Can be slow to converge - can be sensitive to the choice of learning rate
Stochastic Gradient Descent	- Faster than gradient descent - Can be used for large datasets	- Can be less accurate than gradient descent - Can be more sensitive to noise in the data
Mini-Batch SGD	- Faster than stochastic gradient descent - More accurate than stochastic gradient descent - Less sensitive to noise in data	- More complex to implement than Stochastic gradient descent.

-> optimizers - are algorithms or methods used to change the attributes of the NN such as weights and learning rate to reduce the losses.

Approximate Second order Methods for Optimization:-

are algorithms that leverage second-order information to improve optimization performance. Two popular methods are Newton's method and quasi-Newton method:

i) Newton's Method:- Calculates the Hessian matrix and solves a system of equations to obtain the search direction. It provides rapid convergence by incorporating detailed curvature information. However, it is computationally expensive.



ii) Quasi-~~Network~~ Newton Methods: approximate the Hessian matrix using iterative updates, such as the BFGS algorithm. They strike a balance between computational cost and convergence speed. They provide efficient approximations without explicitly computing the Hessian.

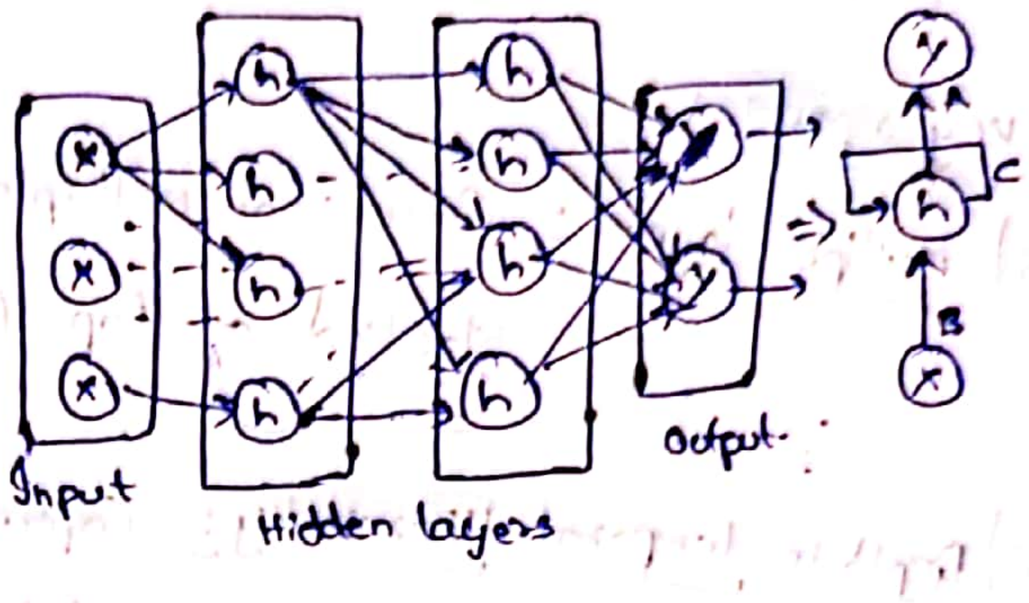
iii) Levenberg-Marquardt algorithm (LM): is designed to work specifically with loss functions which take the form of a sum of squared errors.

-> It works without computing the exact Hessian matrix.

-> Instead, it works with the gradient vector and the Jacobian matrix.

Recurrent Neural Networks (RNN) :-

are a class of NN designed to process sequential data. Unlike feedforward NN, which process input independently. In this, the output of the network is feedback to the network as input inorder to predict the output.



The main characteristics of RNN:

i) Sequential Processing:

Processes input sequence one element at a time, while maintaing a hidden state that captures information from previous inputs.

ii) Memory and Time Dependency:

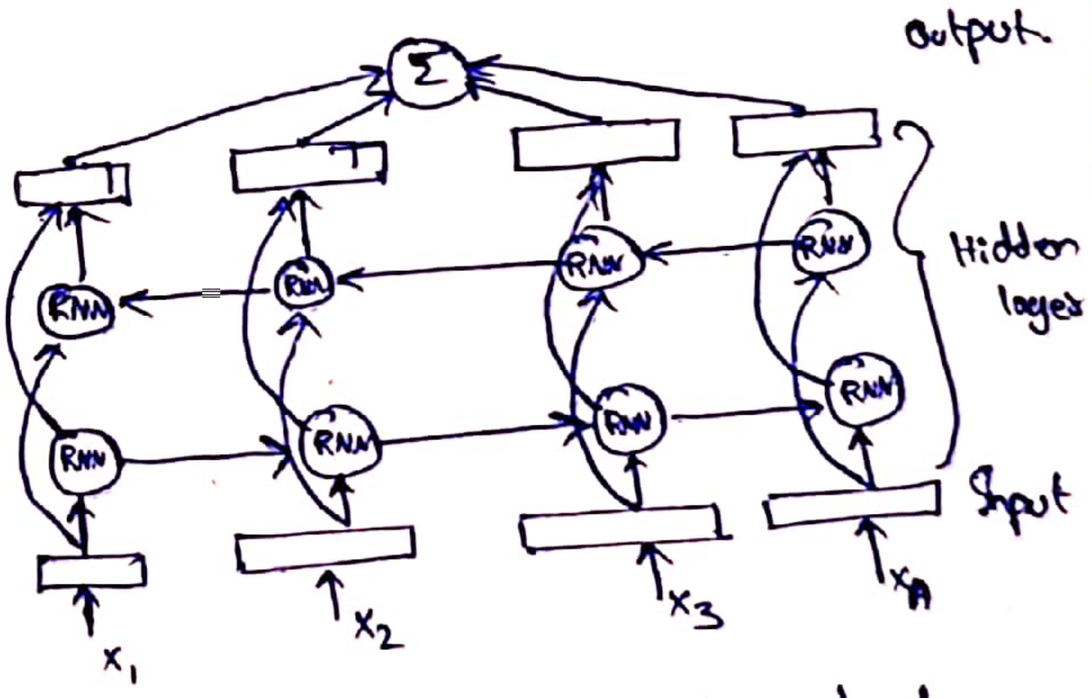
Can capture dependency over time, by considering past information through the hidden state, making them suitable for speech recognition, language model & machine translation.

iii) Parameter sharing:

RNN shares same set of ~~patterns~~ parameters across all time steps, allows them to learn patterns and dependencies across the entire sequence.

Bidirectional Recurrent Neural Networks (Bi-RNN) :-

enhance the capabilities of standard RNNs by considering information from both past and future context. RNN which is Bidirectional can process the sequence in reverse order.



Key aspects of Bi-dir RNN :-

i) Dual processing: Bi-dir RNN have two separate RNNs - one that processes the sequence from start and other from end.

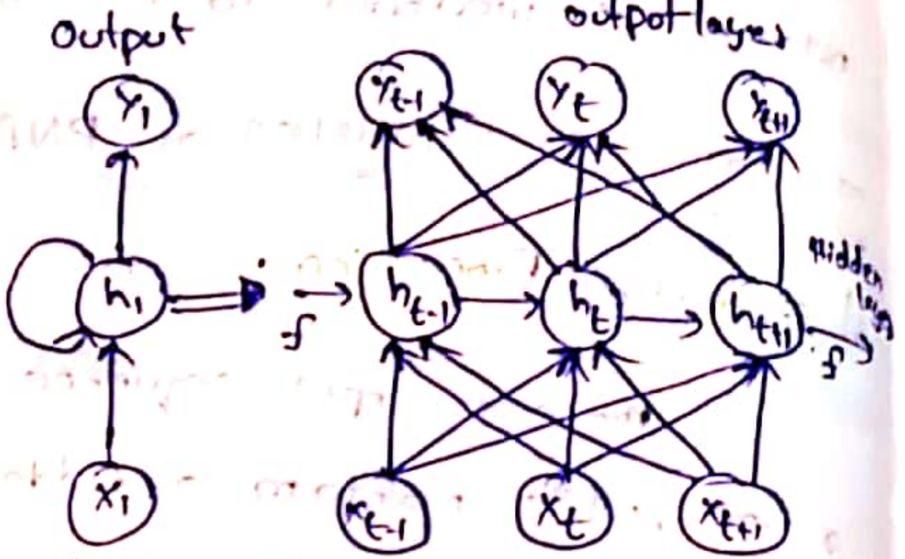
ii) Context Fusion: The outputs of the forward and backward RNNs are combined, to capture information from both past and future context.

iii) Enhanced Information Flow: By considering both past and future context, Bi-dir RNN can capture dependencies in both directions.

-) This is used in entity recognition, sentiment analysis, and speech recognition.

Deep Recurrent Neural networks (Deep RNN):

Refers to RNN architectures with multiple layers. These architectures enable the learning of hierarchical representations and complex dependencies in sequential data.



key aspects:

i) Multiple layers: Consists of multiple recurrent layers stacked on top of each other

ii) Depth in temporal Hierarchy: Captures both short-term and long-term patterns

iii) Enhanced representation learning: Deep RNN can learn more expressive representations by leveraging the hierarchical structure, enabling them to model complex sequential relationship.

→ Deep RNN is used in speech recognition, language modeling, and machine translation.