SATHYABAMA
INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)
Accredited "A" Grade by NAAC | 12B Status by UGC | Approved by AICTE
www.sathyabama.ac.in

SCHOOL OF COMPUTING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

UNIT – II – DEEP LEARNING – SCSA3015

UNIT II INTRODUCTION TO DEEP LEARNING

History of Deep Learning- A Probabilistic Theory of Deep Learning- Backpropagation and regularization, batch normalization- VC Dimension and Neural Nets-Deep Vs Shallow Networks Convolutional Networks- Generative Adversarial Networks (GAN), Semi-supervised Learning

2.1 History of Deep Learning [DL]:

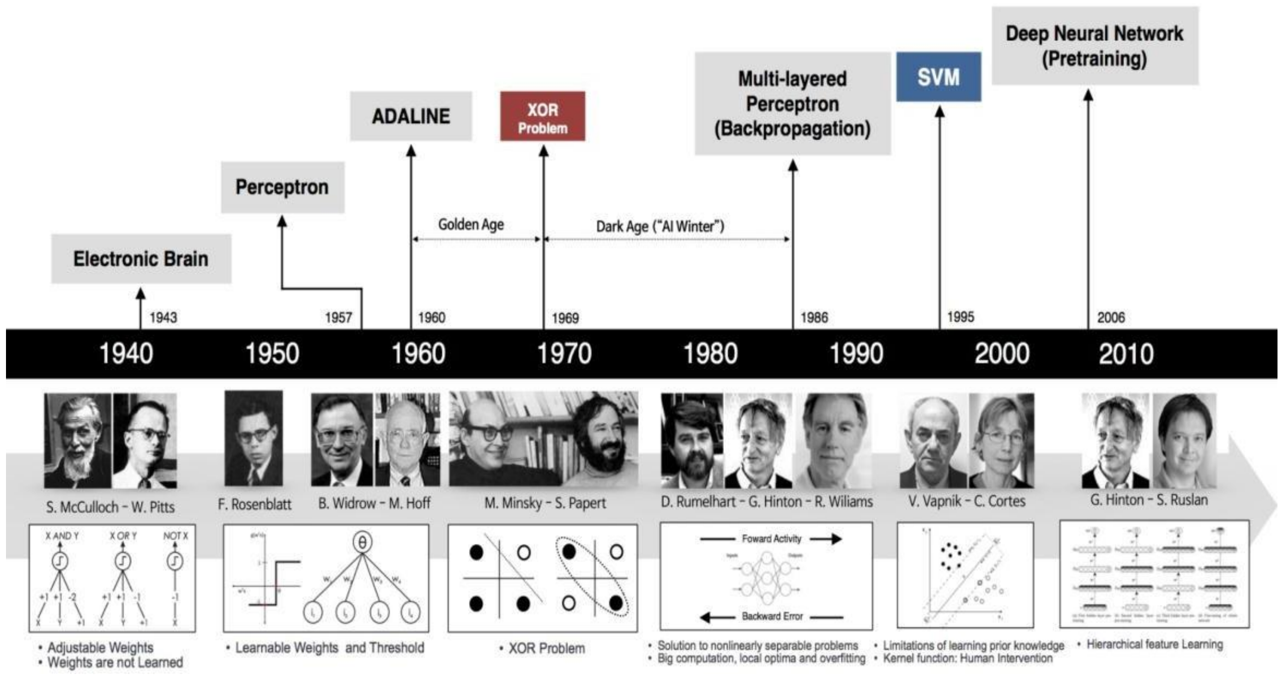
- ❑ The chain rule that underlies the back-propagation algorithm was invented in the seventeenth century (Leibniz, 1676; L'Hôpital, 1696)
- ❑ Beginning in the 1940s, the function approximation techniques were used to motivate machine learning models such as the perceptron
- ❑ The earliest models were based on linear models. Critics including Marvin Minskypointed out several of the flaws of the linear model family, such as its inability to learn the XOR function, which led to a backlash against the entire neural network approach
- ❑ Efficient applications of the chain rule based on dynamic programming began to appearin the 1960s and 1970s
- ❑ Werbos (1981) proposed applying chain rule techniques for training artificial neural networks. The idea was finally developed in practice after being independently rediscovered in different ways (LeCun, 1985; Parker, 1985; Rumelhart et al., 1986a)
- ❑ Following the success of back-propagation, neural network research gained popularity and reached a peak in the early 1990s. Afterwards, other machine learning techniques became more popular until the modern deep learning renaissance that began in 2006
- ❑ The core ideas behind modern feedforward networks have not changed substantially since the 1980s. The same back-propagation algorithm and the same approaches to gradient descent are still in use.

Most of the improvement in neural network performance from 1986 to 2015 can be attributed to two factors. First, larger datasets have reduced the degree to which statistical generalization is a challenge for neural networks. Second, neural networks have become much larger, because of more powerful computers and better software infrastructure. A smallnumber of algorithmic changes have also improved the performance of neural networks noticeably. One of these algorithmic changes was the replacement of mean squared error withthe cross-entropy family of loss functions. Mean squared error was popular in the 1980s and1990s but was gradually replaced by cross-entropy losses and the principle of maximum likelihood as ideas spread between the statistics community and the machine learningcommunity.

The other major algorithmic change that has greatly improved the performance of feedforward networks was the replacement of sigmoid hidden units with piecewise linear hidden units, such as rectified linear units. Rectification using the max{0, z} function was introduced in early neural network models and dates back at least as far as the Cognitron andNeo-Cognitron (Fukushima, 1975, 1980).

For small datasets, Jarrett et al. (2009) observed that using rectifying nonlinearities iseven more important than learning the weights of the hidden layers. Random weights are sufficient to propagate useful information through a rectified linear network, enabling the classifier layer at the top to learn how to map different feature vectors to class identities. Whenmore data is available, learning begins to extract enough useful knowledge to exceed the performance of randomly chosen parameters. Glorot et al. (2011a) showed that learning is fareasier in deep rectified linear networks than in deep networks that have curvature or two-sidedsaturation in their activation functions.

When the modern resurgence of deep learning began in 2006, feedforward networks continued to have a bad reputation. From about 2006 to 2012, it was widely believed that feedforward networks would not perform well unless they were assisted by other models, such as probabilistic models. Today, it is now known that with the right resources and engineering practices, feedforward networks perform very well. Today, gradient-based learning in feedforward networks is used as a tool to develop probabilistic models. Feedforward networks continue to have unfulfilled potential. In the future, we expect they will be applied to many more tasks, and that advances in optimization algorithms and modeldesign will improve their performance even further.


Electronic Brain
Perceptron
ADALINE
XOR Problem
Multi-layered Perceptron (Backpropagation)
SVM
Deep Neural Network (Pretraining)
Golden Age
Dark Age ("AI Winter")
1943
1957
1960
1969
1986
1995
2006
1940
1950
1960
1970
1980
1990
2000
2010
S. McCulloch - W. Pitts
F. Rosenblatt
B. Widrow - M. Hoff
M. Minsky - S. Papert
D. Rumelhart - G. Hinton - R. Wiliams
V. Vapnik - C. Cortes
G. Hinton - S. Ruslan
• Adjustable Weights
• Weights are not Learned
• Learnable Weights and Threshold
• XOR Problem
• Solution to nonlinearly separable problems
• Big computation, local optima and overfitting
• Limitations of learning prior knowledge
• Kernel function: Human Intervention
• Hierarchical feature Learning

2.2 A Probabilistic Theory of Deep Learning

Probability is the science of quantifying uncertain things. Most of machine learning and deep learning systems utilize a lot of data to learn about patterns in the data. Whenever data is utilized in a system rather than sole logic, uncertainty grows up and whenever uncertainty grows up, probability becomes relevant.

By introducing probability to a deep learning system, we introduce common sense to the system. In deep learning, several models like Bayesian models, probabilistic graphical models, Hidden Markov models are used. They depend entirely on probability concepts.

Probabilistic Theory - ACTING UNDER UNCERTAINTY

Uncertainty is a situation which involves imperfect and/or unknown information. Uncertainty arises in partially observable and/or stochastic environments. A state of having limited knowledge where it is impossible to exactly describe the existing state, a future outcome, or more than one possible outcome.

Let action A_t = leave for airport t minutes before flight.

Will A_t get me there on time?

Problems:

- partial observability (road state, other drivers' plans, etc.)
- noisy sensors (traffic reports)
- uncertainty in action outcomes (flat tire, etc.)
- immense complexity of modeling and predicting traffic

Example 1 :

Flip a coin. What is the chance of it landing heads side up? There are 2 sides, heads and tails. The heads side is one of them, thus

(Heads) = 1/(1+1) = ½ = .50 = 50%

Example 2:

Throw a die. What is the probability of getting a 3.

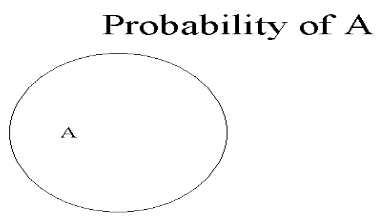
$$P(\text{die} = 3) = \frac{\text{1 side with a 3}}{\text{6 sides total}}$$

P(die = 3) = 1/6 = .167 = 16.7%

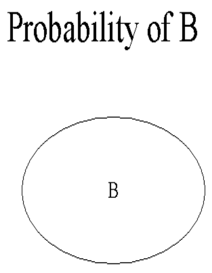
Conditional Probability

Events A and B are events that are not mutually exclusive, but occur conditionally on the occurrence of one another.

The probability that event A will occur: p(A)



The probability that event B will occur: p(B)

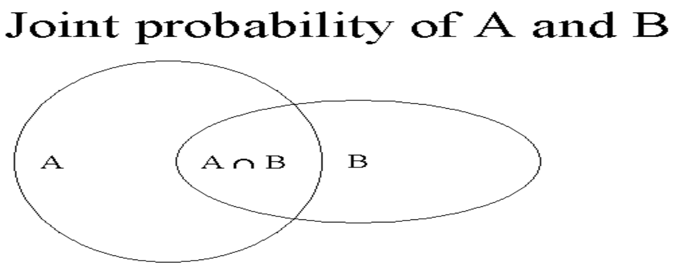


The number of times that both A and B occur or the probability that both events A and B will occur is called the **joint probability.**

Mathematically joint probability is defined as:

$p(A \cap B)$

i.e. the probability that both A and B will occur



The probability that event A will occur if event B occurs is called **conditional probability.**

$$P(A|B) = \frac{\text{the number of times A and B can occur}}{\text{the number of times B can occur}}$$

or

$$P(A|B) = \frac{P(A \cap B)}{P(B)}$$

Lets take an example.

Suppose we have 2 dice and we want to know what the probability of getting an 8 is. Normally if we roll both at the same time the probability is 5/36.

1,1	1,2	1,3	1,4	1,5	1,6
2,1	2,2	2,3	2,4	2,5	2,6
3,1	3,2	3,3	3,4	3,5	3,6
4,1	4,2	4,3	4,4	4,5	4,6
5,1	5,2	5,3	5,4	5,5	5,6
6,1	6,2	6,3	6,4	6,5	6,6

But what happens if we roll the first die and get a 5, now what is the probability of getting an 8?

There is only one way to get an 8 after a 5 has been rolled. You have to roll a 3.

Looking at the formula:

$$P(A|B) = \frac{P(A \cap B)}{P(B)}$$

Lets rephrase it for our problem:

$$P(\text{getting an 8 using 2 die} \mid \text{given that we roll a 5 with first dice}) = \frac{P(\text{rolling 5 and 3})}{P(\text{rolling a 5})}$$

P(A|B) = (1/36) / (1/6) = (1/36) * (6/1) = 6/36 = 1/6

BAYES'S RULE

The below equation is Bayes rule:

$$P(B|A) = \frac{P(A|B)\ P(B)}{P(A)}$$

Bayes rule allows unknown probabilities to be computed from known conditional probabilities.

In the given bayes rule equation,

- Events A, B – Hypothesis
- P(B|A) - **Posterior probability** (updated probability after the evidence is considered)
- P(A|B) – **Conditional Probability** or **Likelihood** (probability of the evidence, given the belief is true)
- P(B) – **Prior probability** (the probability before the evidence is considered)
- P(A) - **Marginal probability** (probability of the evidence, under any circumstance)

Example 1:

A bag I contains 4 white and 6 black balls while another Bag II contains 4 white and 3 black balls. One ball is drawn at random from one of the bags, and it is found to be black. Find the probability that it was drawn from Bag I.

Solution:

Let E1 - the event of choosing bag I,
E2 - the event of choosing bag II
A - the event of drawing a black ball.

By using Bayes' theorem, the probability of drawing a black ball from bag I out of two bags,

$$\mathbf{P(E1/A) = \frac{P(A/E1)\ P(E1)}{P(A)}}$$

P(E1) = P(E2) = ½
P(A/E1) = P(drawing a black ball from Bag I) = 6/10 = 3/5
P(A/E2) = P(drawing a black ball from Bag II) = 3/7
P(A) = P(E1) P(A/E1) + P(E2) P(A/E2) = 18/35

$$\mathbf{P(E1/A) = \frac{P(A/E1)\ P(E1)}{P(A)} = 7/12}$$

Naive Bayes Classifier

- Naïve Bayes algorithm is a supervised learning algorithm, which is based on **Bayes theorem** and used for solving classification problems.
- It is mainly used in text classification that includes a high-dimensional training dataset.
- Naïve Bayes Classifier is one of the simple and most effective Classification algorithms which helps in building the fast machine learning models that can make quick predictions.
- It is a probabilistic classifier, which means it predicts on the basis of the probability of an object.
- Some popular examples of Naïve Bayes Algorithm are spam filtration, Sentimental analysis, and classifying articles.

Advantages of Naïve Bayes Classifier:

- Naïve Bayes is one of the fast and easy ML algorithms to predict a class of datasets.
- It can be used for Binary as well as Multi-class Classifications.
- It is the most popular choice for text classification problems.

Disadvantages of Naïve Bayes Classifier:

- Naive Bayes assumes that all features are independent or unrelated, so it cannot learn the relationship between features.

Applications of Naïve Bayes Classifier:

- It is used for Credit Scoring**.**
- It is used in medical data classification**.**
- It can be used in real-time predictions**.**
- It is used in Text classification such as Spam filtering**.**

Example : We want to use Bayes theorem to find out the likelihood of playing tennis for a given set weather attributes. The attribute values (Outlook, Temperature, Humidity, and Wind). To determine our answer (if we are going to play tennis given a certain set of conditions) we make an expression that determines the probability based on our training examples from the table.Given a = (Outlook = sunny, Temperature = cool, Humidity = high, Wind = strong) to be selected.

Day	**Outlook**	**Temperature**	**Humidity**	**Wind**	**Play Tennis**
1	Sunny	Hot	High	Weak	No
2	Sunny	Hot	High	Strong	No
3	Overcast	Hot	High	Weak	Yes
4	Rain	Mild	High	Weak	Yes
5	Rain	Cool	Normal	Weak	Yes
6	Rain	Cool	Normal	Strong	No
7	Overcast	Cool	Normal	Strong	Yes
8	Sunny	Mild	High	Weak	No
9	Sunny	Cool	Normal	Weak	Yes
10	Rain	Mild	Normal	Weak	Yes
11	Sunny	Mild	Normal	Strong	Yes
12	Overcast	Mild	High	Strong	Yes
13	Overcast	Hot	Normal	Weak	Yes
14	Rain	Mild	High	Strong	No

$$P(\text{Play Tennis} \mid \text{Attributes}) = \frac{P(\text{Attributes} \mid \text{Play Tennis}) * P(\text{Play Tennis})}{P(\text{Attributes})}$$

P(a|v) = P(Outlook, Temperature, Humidity, Wind | Play tennis, Don’t Play tennis)

First we estimate the probability of playing tennis:

P(Play Tennis = Yes) = 9/14 = .64

P(Play Tennis = No) = 5/14 = .36

Then we estimate the conditional probabilities of the individual attributes

Outlook:

P(Outlook = Sunny | Play Tennis = Yes) = 2/9 = 0.22

P(Outlook = Sunny | Play Tennis = No) = 3/5 = 0.6

P(Outlook = Overcast | Play Tennis = Yes) = 4/9 = 0.44

P(Outlook = Overcast | Play Tennis = No) = 0/5 = 0

P(Outlook = Rain | Play Tennis = Yes) = 3/9 = 0.33

P(Outlook = Rain | Play Tennis = No) = 2/5 = 0.4

Temperature

P(Temperature = Hot | Play Tennis = Yes) = 2/9 = 0.22

P(Temperature = Hot | Play Tennis = No) = 2/5 = 0.40

P(Temperature = Mild | Play Tennis = Yes) = 4/9 = 0.44

P(Temperature = Mild | Play Tennis = No) = 2/5 = 0.40

P(Temperature = Cool | Play Tennis = Yes) = 3/9 = 0.33

P(Temperature = Cool | Play Tennis = No) = 1/5 = 0.20

Humidity

P(Humidity = High | Play Tennis = Yes) = 3/9 = 0.33

P(Humidity = High | Play Tennis = No) = 4/5 = 0.80

P(Humidity = Normal | Play Tennis = Yes) = 6/9 = 0.66

P(Humidity = Normal | Play Tennis = No) = 1/5 = 0.20

Wind

P(Wind = Weak | Play Tennis = Yes) = 6/9 = 0.66

P(Wind = Weak | Play Tennis = No) = 2/5 = 0.40

P(Wind = Strong | Play Tennis = Yes) = 3/9 = 0.33

P(Wind = Strong | Play Tennis = No) = 3/5 = 0.60

a = (Outlook = sunny, Temperature = cool, Humidity = high, Wind = strong)

P(Playtennis = No | (Outlook = sunny, Temperature = cool, Humidity = high, Wind = strong))

$$= \frac{P(sunny|Yes)*P(cool|Yes)*P(high|Yes)*P(strong|Yes) * P(yes)}{P((sunny, cool, high, strong) | Yes) + P((sunny, cool, high, strong) | No)}$$

$$= \frac{(.22 * .33 * .33 * .33) * .64}{(.22 * .33 * .33 * .33)*64 + (.6 * .2 * .8 * .6)*.36}$$

$$= \frac{.0051}{.0051 + .0207}$$

= 0 .1977

$$P(No|(sunny, cool, high, strong)) = \frac{P((sunny, cool, high, strong) | No) * P(No)}{P(sunny, cool, high, strong)}$$

$$= \frac{.0207}{.0051 + .0207}$$

= 0.8023

As we can see, the Bayes Naïve classifier gives a value of just about 20% for playing tennis in the described conditions, and value of 80% for not playing tennis in these conditions, therefore the prediction is that no tennis will be played if the day is like these conditions.

2.3 Back Propagation Networks (BPN)

Need for Multilayer Networks

- Single Layer networks cannot used to solve Linear Inseparable problems &can only be used to solve linear separable problems
- Single layer networks cannot solve complex problems
- Single layer networks cannot be used when large input-output data set isavailable
- Single layer networks cannot capture the complex information's available inthe training pairs

Hence to overcome the above said Limitations we use Multi-Layer Networks.

Multi-Layer Networks

- Any neural network which has at least one layer in between input and outputlayers is called Multi-Layer Networks
- Layers present in between the input and out layers are called Hidden Layers
- Input layer neural unit just collects the inputs and forwards them to the nexthigher layer
- Hidden layer and output layer neural units process the information's feed tothem and produce an appropriate output
- Multi -layer networks provide optimal solution for arbitrary classificationproblems
- Multi -layer networks use linear discriminants, where the inputs are nonlinear

Back Propagation Networks (BPN)

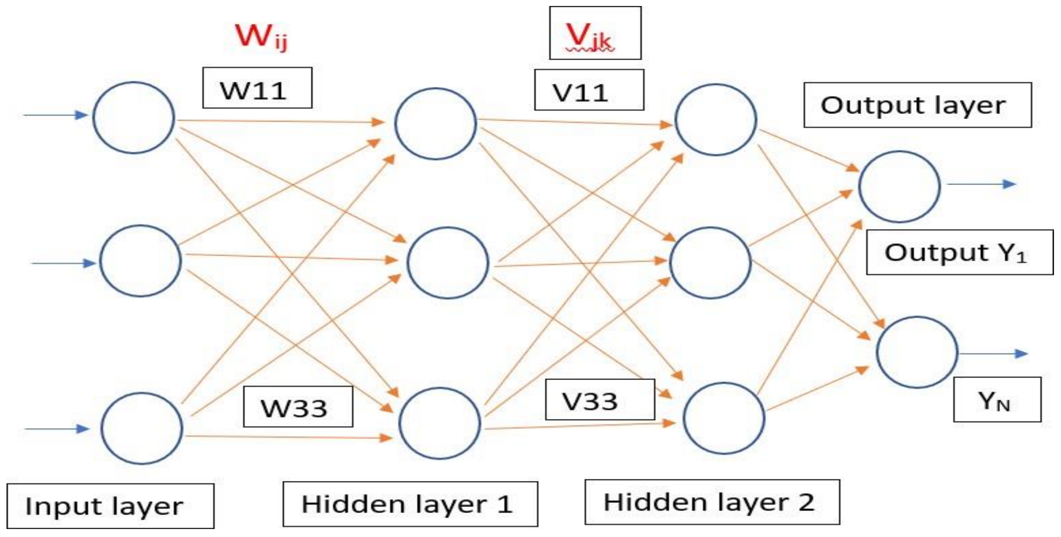
Introduced by Rumelhart, Hinton, & Williams in 1986.

BPN is a Multi-layer Feedforward Network but error is back propagated, Hence the name Back Propagation Network (BPN). It uses Supervised Training process; it has a systematic procedure for training the network and is used in Error Detection and Correction.

Generalized Delta Law /Continuous Perceptron Law/ Gradient DescentLaw is used in this network. Generalized Delta rule minimizes the mean squared error of the output calculated from the output. Delta law has faster convergence ratewhen compared with Perceptron Law.

The Training process used in backpropagation involves three stages, which are listed as below:

1. Feedforward of input training pair
2. Calculation and backpropagation of associated error
3. Adjustments of weights



The algorithm for BPN is as classified int four major steps as follows:

1. Initialization of Bias, Weights
2. Feedforward process
3. Back Propagation of Errors
4. Updating of weights & biases

Algorithm:

I. Initialization of weights:

Step 1: Initialize the weights to small random values near zero

Step 2: While stop condition is false , Do steps 3 to 10

Step 3: For each training pair do steps 4 to 9

II. Feed forward of inputs

Step 4: Each input xi is received and forwarded to higher layers (nexthidden)

Step 5: Hidden unit sums its weighted inputs as follows$Z_{inj} = W_{oj} + \Sigma x_i w_{ij}$

Applying Activation function$Z_j = f(Z_{inj})$

This value is passed to the output layerStep 6: Output unit sums it’s weighted inputs

$$y_{ink} = V_{oj} + \Sigma Z_j V_{jk}$$

Applying Activation function

$$Y_k = f(y_{ink})$$

III. Backpropagation of Errors

Step 7: $\delta_k = (t_k - Y_k) f(y_{ink})$Step 8: $\delta_{inj} = \Sigma \delta_j V_{jk}$

IV. Updating of Weights & Biases

Step 8: Weight correction is

$$\Delta w_{ij} = \alpha \delta_k Z_j$$

bias Correction is

$$\Delta w_{oj} = \alpha \delta_k$$

V. Updating of Weights & Biases

Step 9: continued:

New Weight is

$$W_{ij(new)} = W_{ij(old)} + \Delta w_{ij} \; V_{jk(new)} = V_{jk(old)} + \Delta V_{jk}$$

New bias is

$$W_{oj(new)} = W_{oj(old)} + \Delta w_{oj} V_{ok(new)} = V_{ok(old)} + \Delta V_{ok}$$

Step 10: Test for Stop Condition

<u>Advantages of BPN</u>

- Has smooth effect on weight correction
- Computing time is less if weight's are small
- Faster than perceptron model
- Has a systematic weight updating procedure

<u>Disadvantages of BPN</u>

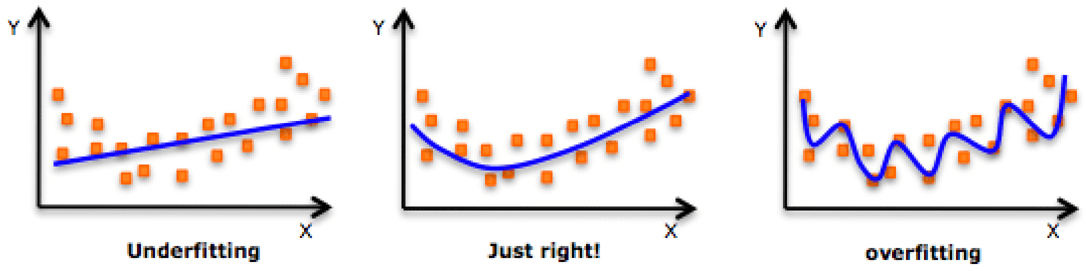
- Learning phase requires intensive calculations
- Selection of number of Hidden layer neurons is an issue
- Selection of number of Hidden layers is also an issue
- Network gets trapped in Local Minima
- Training time is more for Complex problems

2.4 Regularization

A fundamental problem in machine learning is how to make an algorithm thatwill perform well not just on the training data, but also on new inputs. Many strategies used in machine learning are explicitly designed to reduce the test error, possibly at the expense of increased training error. These strategies are known collectively as regularization.

Regularization is a technique that helps reduce overfitting or reduce variance in our network by penalizing for complexity.

In Overfitting, the model tries to learn too many details in the training data along with the noise from the training data. As a result, the model performance is very poor on unseen or test datasets. So overfitting problem can be defined as data perform well in training dada and fails or test data.



- **Underfit Model**. A model that fails to sufficiently learn the problem and performs poorly on a training dataset and does not perform well on a test sample.
- **Good Fit Model**. A model that suitably learns the training dataset and generalizes well to the test dataset.
- **Overfit Model**. A model that learns the training dataset too well, performing well on the training dataset but does not perform well on a test sample.

<u>Reasons for Overfitting</u>

1. Distribution of weights
2. Number of hidden layers

Different Regularization Techniques

L2 & L1 regularization

L1 and L2 are the most common types of regularization used to solve overfitting problem caused by distribution of weights. These update the general cost function by adding another term known as the regularization term.

Cost function = Loss (say, binary cross entropy) + Regularization term

Due to the addition of this regularization term, the values of weight matrices decrease because it assumes that a neural network with smaller weight matrices leads to simpler models. Therefore, it will also reduce overfitting to quite an extent.

However, this regularization term differs in L1 and L2.

In L2, we have:

$$Cost\,function = Loss + \frac{\lambda}{2m} * \sum \|w\|^2$$

Here, λ is the regularization parameter. It is the hyperparameter whose value is optimized for better results. L2 regularization is also known as weight decay as it forces the weights to decay towards zero (but not exactly zero).

In L1, we have:

$$Cost\,function = Loss + \frac{\lambda}{2m} * \sum \|w\|$$

In this, we penalize the absolute value of the weights. Unlike L2, the weights may be reduced to zero here. Hence, it is very useful when we are trying to compress our model. Otherwise, we usually prefer L2 over it.

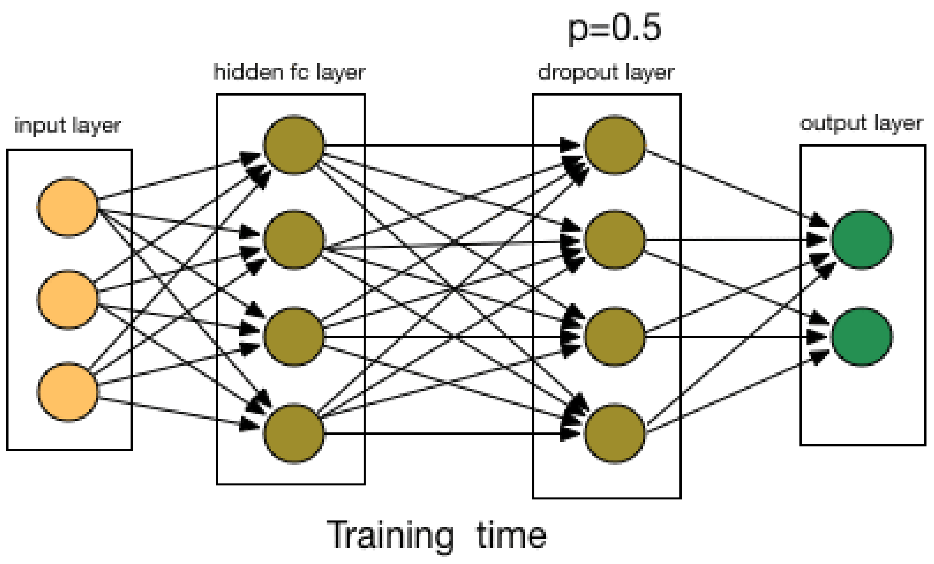
Dropout

This is the one of the most interesting types of regularization techniques. It also produces very good results and is consequently the most frequently used regularization technique in the field of deep learning.

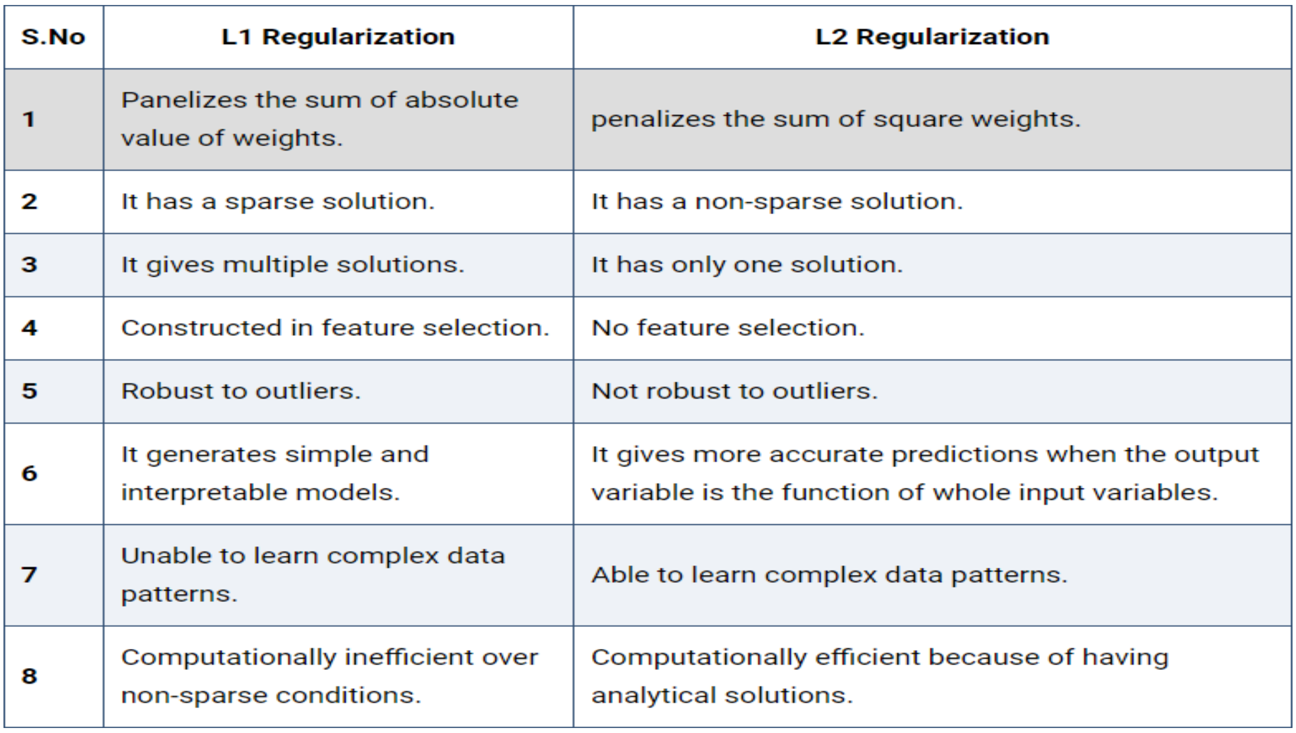
Dropout is a technique where randomly selected neurons are ignored during training. They are "dropped-out" randomly. This means that their contribution to the activation of downstream neurons is temporally removed on the forward pass and any weight updates are not applied to the neuron on the backward pass.

Dropout may be implemented on any or all hidden layers in the network as well as the visible or input layer. It is not used on the output layer. Dropout regularization is a generic approach. The default interpretation of the dropout hyperparameter is the probability of training a given node in a layer, where 1.0 means no dropout, and 0.0 means no outputs from the layer. A good value for dropout in a hidden layer is between 0.5 and 0.8. Input layers use a larger dropout rate, such as of 0.8.

Dropout is a regularization technique for reducing overfitting in neural networks by preventing complex co-adaptations on training data. It is a very efficient way of performing model averaging with neural networks. The term "dropout" refers to dropping out units (both hidden and visible) in a neural network.



L1 Regularization Vs L2 Regularization



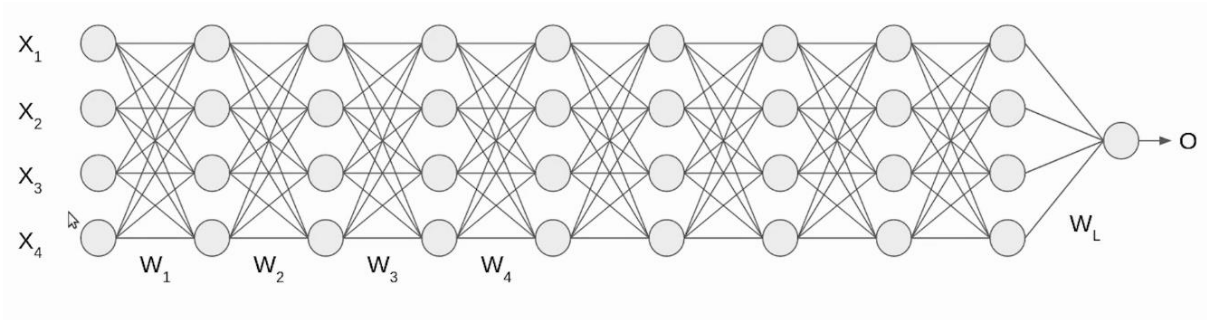
S.No	L1 Regularization	L2 Regularization
1	Panelizes the sum of absolute value of weights.	penalizes the sum of square weights.
2	It has a sparse solution.	It has a non-sparse solution.
3	It gives multiple solutions.	It has only one solution.
4	Constructed in feature selection.	No feature selection.
5	Robust to outliers.	Not robust to outliers.
6	It generates simple and interpretable models.	It gives more accurate predictions when the output variable is the function of whole input variables.
7	Unable to learn complex data patterns.	Able to learn complex data patterns.
8	Computationally inefficient over non-sparse conditions.	Computationally efficient because of having analytical solutions.

2.5 Batch Normalization:

It is a method of adaptive reparameterization, motivated by the difficulty of training very deep models. In Deep networks, the weights are updated for each layer. So, the output will no longer be on the same scale as the input (even though input is normalized).

Normalization - is a data pre-processing tool used to bring the numerical data toa common scale without distorting its shape. When we input the data to a machine or deep learning algorithm, we tend to change the values to a balanced scale because, we ensure thatour model can generalize appropriately.

A typical neural network is trained using a collected set of input data called **batch**. Similarly, the normalizing process in batch normalization takes place in batches, not as a single input.

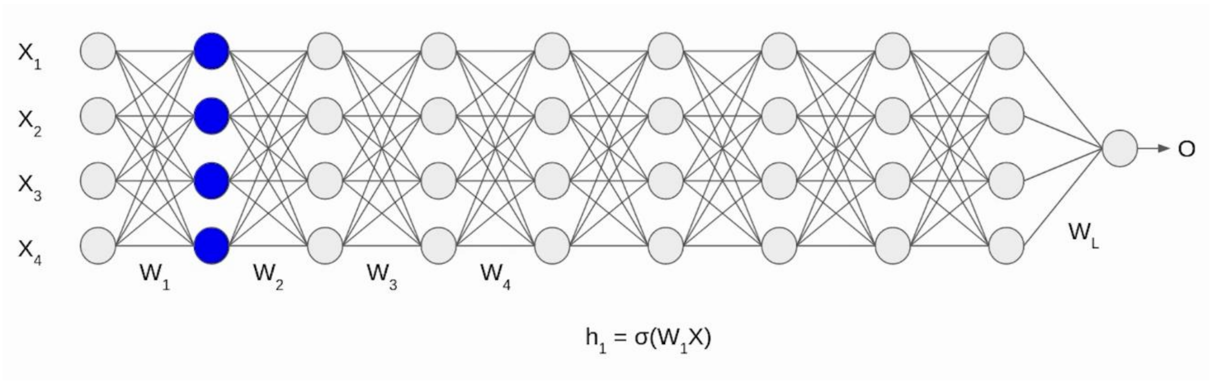


L = Number of layers

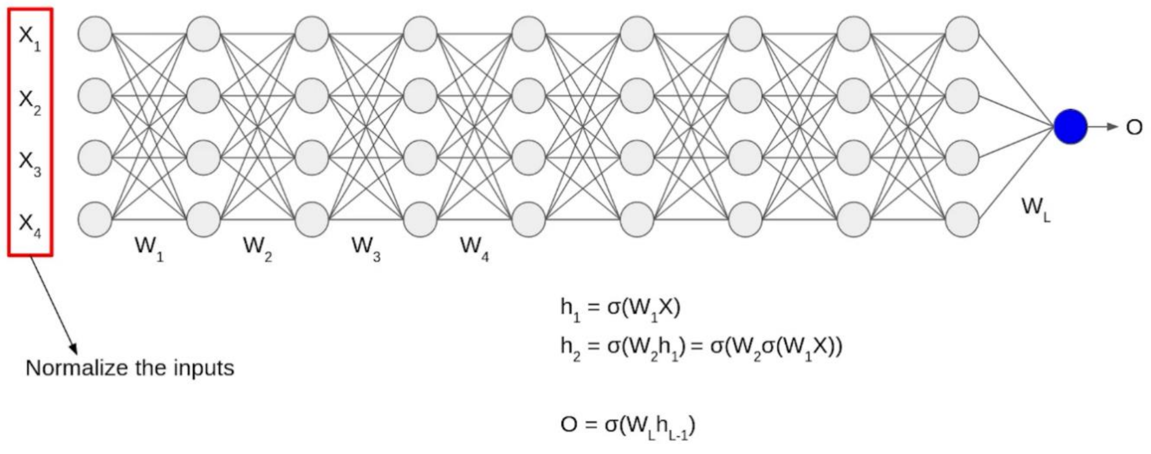
Bias = 0

Activation Function: Sigmoid

Initially, the inputs X1, X2, X3, X4 are in normalized form as they are coming from the pre-processing stage. When the input passes through the first layer, it transforms, as a sigmoid function applied over the dot product of input X and the weight matrix W.



Similarly, this transformation will take place for the second layer and go till the last layer L as shown in the following image.



Although, our input X was normalized with time the output will no longer be on the same scale. As the data go through multiple layers of the neural network and L activation functions are applied, it leads to an internal co-variate shift in the data.

Batch Normalization **is** a two-step process. First, the input is normalized, and later rescaling and offsetting is performed.

Normalization is the process of transforming the data to have a mean zero and standard deviation one. In this step we have our batch input from layer h, first, we need to calculate the mean of this hidden activation.

$$\mu = \frac{1}{m} \sum h_i$$

Here, m is the number of neurons at layer h.

Once we have meant at our end, the next step is to calculate the standard deviation of the hidden activations.

$$\sigma = \left[\frac{1}{m} \sum (h_i - \mu)^2 \right]^{1/2}$$

Further, as we have the mean and the standard deviation ready. We will normalize the hidden activations using these values. For this, we will subtract the mean from each input and divide the whole value with the sum of standard deviation and the smoothing term (ε).

The smoothing term(ε) assures numerical stability within the operation by stopping a division by a zero value.

$$h_{i(norm)} = \frac{(h_i - \mu)}{\sigma + \varepsilon}$$

Rescaling of Offsetting : In the final operation, the re-scaling and offsetting of the input take place. Here two components of the BN algorithm come into the picture, γ(gamma) and β (beta). These parameters are used for re-scaling (γ) and shifting(β) of the vector containing values from the previous operations.

$$h_i = \gamma\, h_{i(norm)} + \beta$$

These two are learnable parameters, during the training neural network ensures the optimal values of γ and β are used. That will enable the accurate normalization of each batch.

Advantages of Batch Normalization

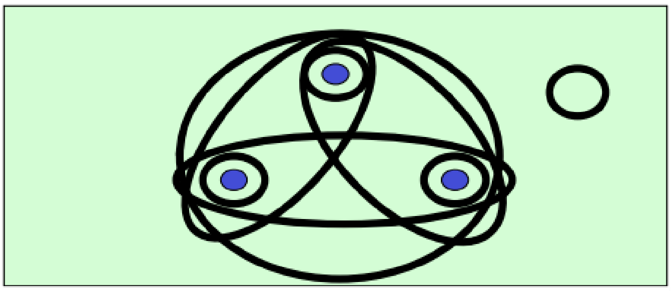
- Speed Up the Training
- Handles internal covariate shift

2.6 VC Dimension

The **Vapnik-Chervonenkis dimension**, more commonly known as the VC dimension, is a model capacity measurement used in statistics and machine learning. It is termed informally as a measure of a model's capacity. It is used frequently to guide the model selection process while developing machine learning applications.

- Def.1: (***set shattering***): a subset S of instances of a set X is shattered by a collection of function F if ∀ S'⊆ S there is a function $f \in F$ such data:

$$f(x) = \begin{cases} 1 & x \in S' \\ 0 & x \in S - S' \end{cases}$$

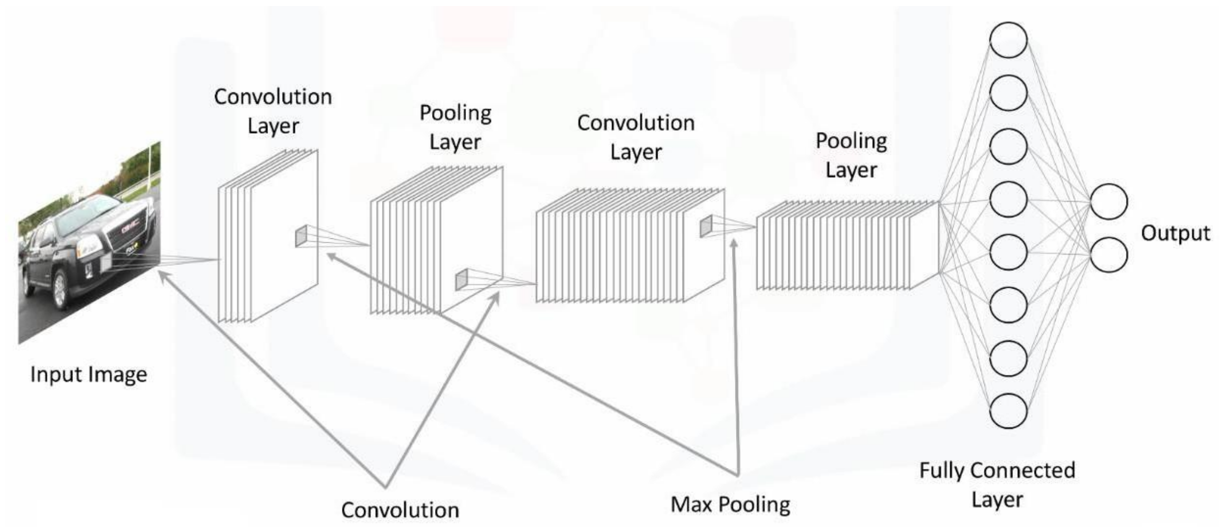


2.7 Difference Between a Shallow Net & Deep Learning Net:

Sl.No	Shallow Net's	Deep Learning Net's
1	One Hidden layer (or very less no. ofHidden Layers)	Deep Net's has many layers of Hidden layers with more no. of neurons in each layers
2	Takes input only as VECTORS	DL can have raw data like image, textas inputs
3	Shallow net's needs more parametersto have better fit	DL can fit functions better with less parameters than a shallow network
4	Shallow networks with one Hidden layer (same no of neurons as DL) cannot place complex functions overthe input space	DL can compactly express highly complex functions over input space
5	The number of units in a shallow network grows exponentially withtask complexity.	DL don't need to increase it size(neurons) for complex problems
6	Shallow network is more difficult to train with our current algorithms (e.g.it has issues of local minima etc)	Training in DL is easy and no issue of local minima in DL

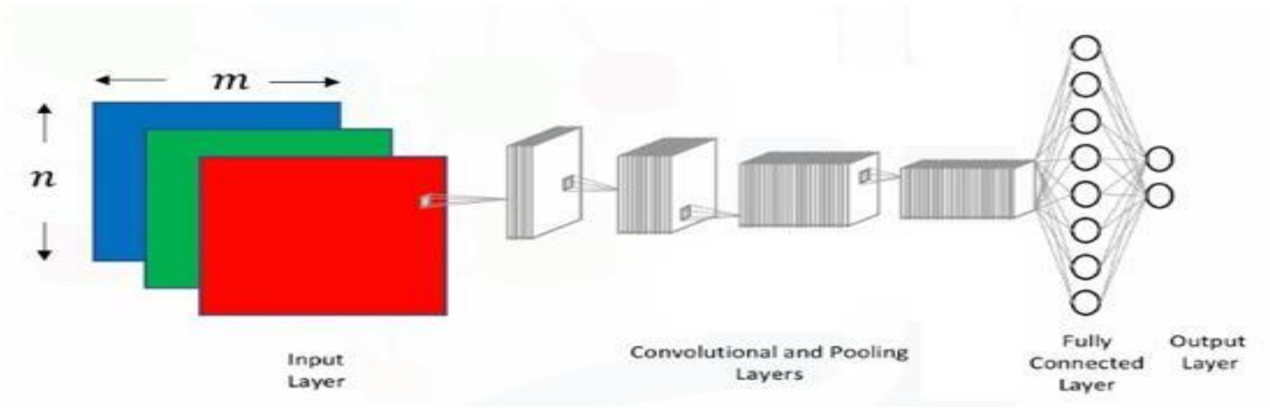
2.8 CONVOLUTIONAL NEURAL NETWORK

A **Convolutional neural network (CNN)** is a neural network that has one or more convolutional layers and is used mainly for image processing, classification, segmentation.



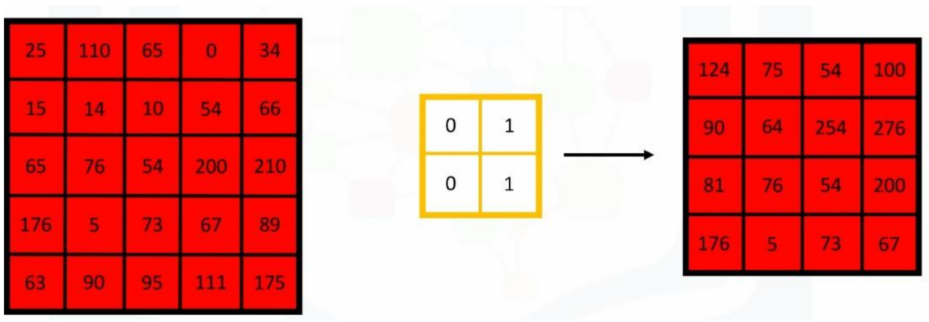
Input layer:

The input to a CNN, is mostly an image (nxmx1-gray scale image and nxmx3-colored image)



Convolution layer:

Here, we basically define filters and we compute the convolution between the defined filters and each of the 3 images.



25	110	65	0	34
15	14	10	54	66
65	76	54	200	210
176	5	73	67	89
63	90	95	111	175

0	1
0	1

124	75	54	100
90	64	254	276
81	76	54	200
176	5	73	67

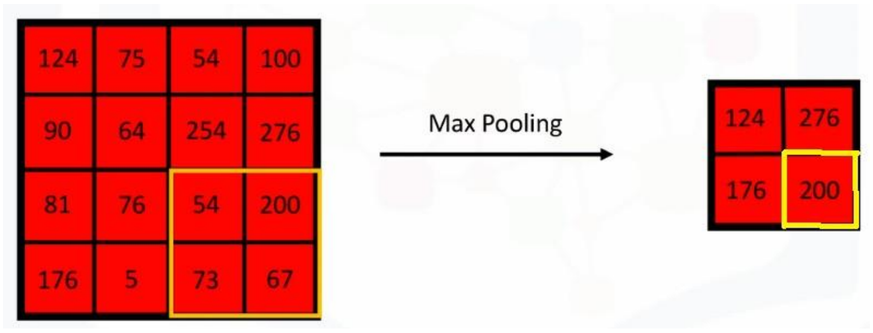
In the same way we apply to remaining (above is for red image, then we do same for green and blue) images. We can apply more than one filter. More filters we use, we can preserve spatial dimensions better. We use convolution instead of considering flatten image as input as we will end up with a massive number of parameters that will need to be optimized and computationally expensive.

It is worth to have ReLU activation function in convolution layer which passed only positive values and make negative values to zeros.

Pooling layer:

Pooling layer objective is to reduce the spatial dimensions of the data propagating through the network.

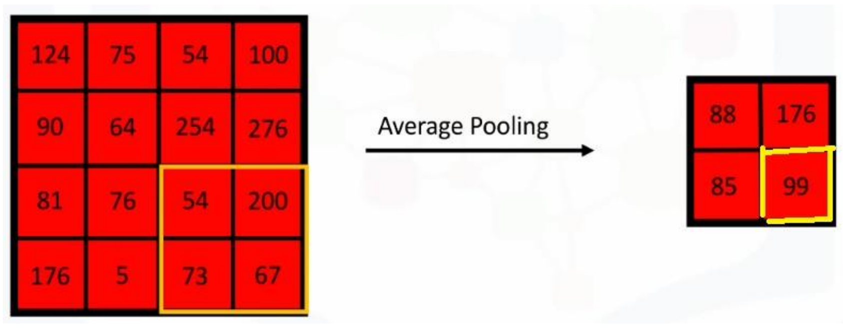
1. **Max Pooling** is the most common, for each section of theimage we scan and keep the highest value.



Max Pooling with stride = 2

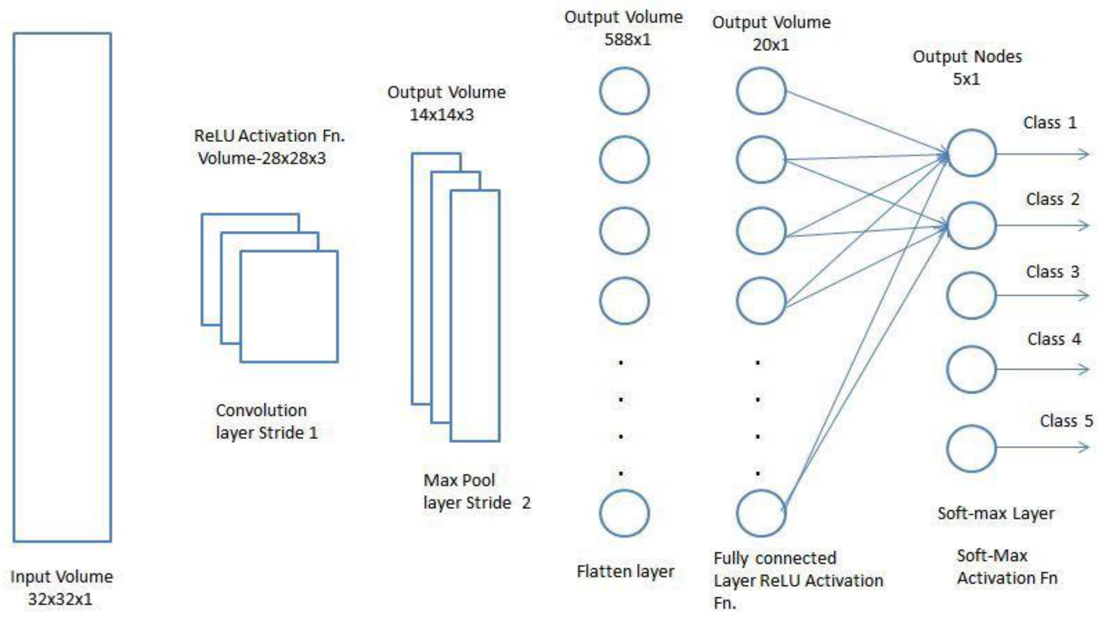
Max. pooling provides spatial variance which enables the neural network to recognize objects in an image even if the object does not exactly resemble the original object.

2. **Average Pooling:** Here, we take average of area we scan.



Average Pooling with stride = 2

Fully Connected Layer:



Here, we flatten the output of the last convolutional layerand connect every node of the current layer with everyother node of the next layer.

Adding a Fully-Connected layer is a (usually) cheap way of learning non-linear combinations of the high-level features as represented by the output of the convolutional layer. The Fully-Connected layer is learning a possibly non-linear function in that space.

Convert input image into a suitable form for our Multi-Level Perceptron, flatten the image into a column vector. The flattened output is fed to a feed-forward neural network and backpropagation applied to every iteration of training. Over a series of epochs, the model is able to distinguish between dominating and certain low-level features in images and classify them using the **Softmax Classification** technique.

There are various architectures of CNNs as,

1. LeNet
2. AlexNet
3. VGGNet
4. GoogLeNet
5. ResNet
6. ZFNet

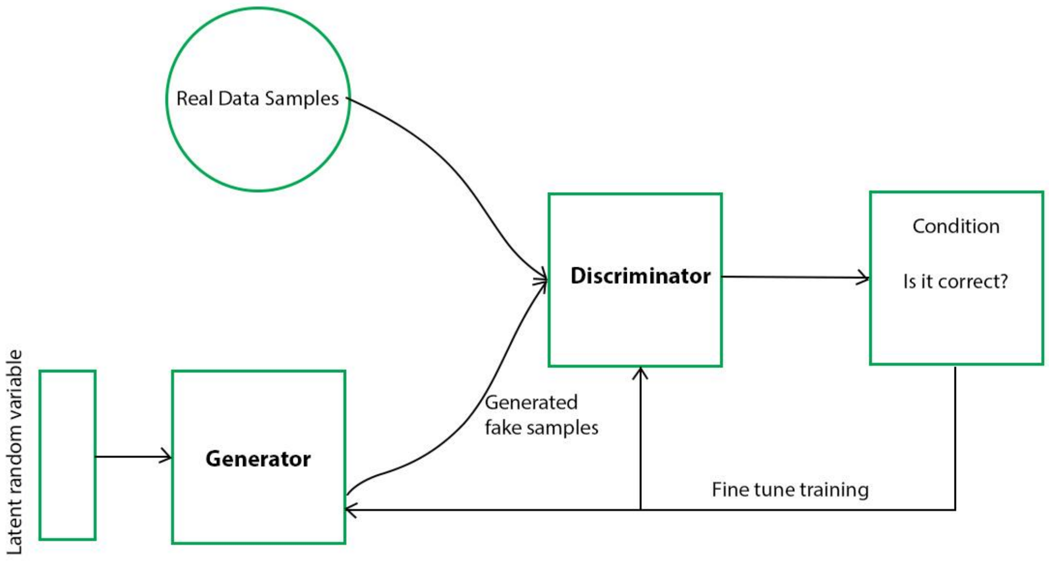
2.9 Generative Adversarial Networks (GANs)

Generative Adversarial Networks (GANs) are a powerful class of neural networks that are used for unsupervised learning. It was developed and introduced by Ian J. Goodfellow in 2014. GANs are basically made up of a system of two competing neural network models which compete with each other and are able to analyze, capture and copy the variations within a dataset.

Generative Adversarial Networks (GANs) can be broken down into three parts:

- **Generative:** To learn a generative model, which describes how data is generated in terms of a probabilistic model.
- **Adversarial:** The training of a model is done in an adversarial setting.
- **Networks:** Use deep neural networks as the artificial intelligence (AI) algorithms for training purpose.

In GANs, there is a **generator** and a **discriminator**. The Generator generates fake samples of data(be it an image, audio, etc.) and tries to fool the Discriminator. The Discriminator, on the other hand, tries to distinguish between the real and fake samples. The Generator and the Discriminator are both Neural Networks and they both run in competition with each other in the training phase. The steps are repeated several times and in this, the Generator and Discriminator get better and better in their respective jobs after each repetition.



Here, the generative model captures the distribution of data and is trained in such a manner that it tries to maximize the probability of the Discriminator in making a mistake. The Discriminator, on the other hand, is based on a model that estimates the probability that the sample that it got is received from the training data and not from the Generator.

The GANs are formulated as a minimax game, where the Discriminator is trying to minimize its reward **V(D, G)** and the Generator is trying to minimize the Discriminator's reward or in other words, maximize its loss. It can be mathematically described by the formula below:

$$\min_{G} \max_{D} V(D, G)$$

$$V(D, G) = \mathbb{E}_{x \sim p_{data}(x)}[\log D(x)] + \mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z))]$$

where,
G = Generator

D = Discriminator

Pdata(x) = distribution of real data

P(z) = distribution of generator

x = sample from Pdata(x)

z = sample from P(z)

D(x) = Discriminator network

G(z) = Generator network

So, basically, training a GAN has two parts:

Part 1: The Discriminator is trained while the Generator is idle. In this phase, the network is only forward propagated and no back-propagation is done. The Discriminator is trained on real data for n epochs, and see if it can correctly predict them as real. Also, in this phase, the Discriminator is also trained on the fake generated data from the Generator and see if it can correctly predict them as fake.

Part 2: The Generator is trained while the Discriminator is idle. After the Discriminator is trained by the generated fake data of the Generator, we can get its predictions and use the results for training the Generator and get better from the previous state to try and fool the Discriminator.

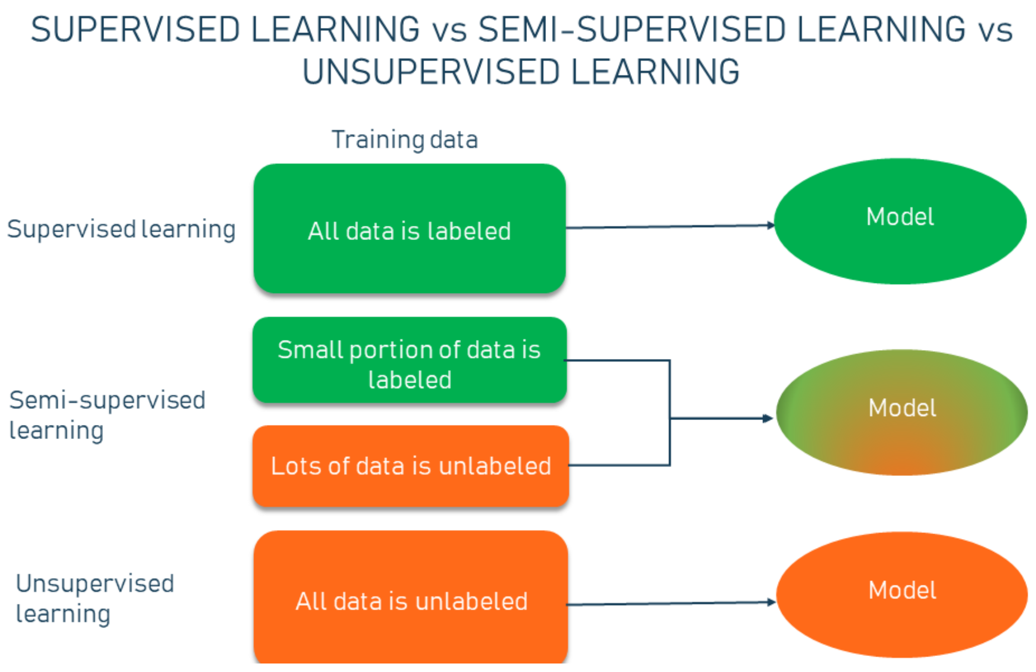
The above method is repeated for a few epochs and then manually check the fake data if it seems genuine. If it seems acceptable, then the training is stopped, otherwise, its allowed to continue for few more epochs.

Different types of GANs:

1. **Vanilla GAN:** This is the simplest type GAN. Here, the Generator and the Discriminator are simple multi-layer perceptrons. In vanilla GAN, the algorithm is really simple, it tries to optimize the mathematical equation using stochastic gradient descent.
2. **Conditional GAN (CGAN):** CGAN can be described as a deep learning method in which some conditional parameters are put into place. In CGAN, an additional parameter 'y' is added to the Generator for generating the corresponding data.
3. **Deep Convolutional GAN (DCGAN):** DCGAN is one of the most popular also the most successful implementation of GAN. It is composed of ConvNets in place of multi-layer perceptrons.
4. **Laplacian Pyramid GAN (LAPGAN):** The Laplacian pyramid is a linear invertible image representation consisting of a set of band-pass images, spaced an octave apart, plus a low-frequency residual.
5. **Super Resolution GAN (SRGAN):** SRGAN as the name suggests is a way of designing a GAN in which a deep neural network is used along with an adversarial network in order to produce higher resolution images. This type of GAN is particularly useful in optimally up-scaling native low-resolution images to enhance its details minimizing errors while doing so.

2.10 SEMI-SUPERVISED LEARNING

Semi-supervised learning (SSL) is a machine learning technique that uses a small portion of labeled data and lots of unlabeled data to train a predictive model.



Supervised learning is training a machine learning model using the labeled dataset. Organic labels are often available in data, but the process may involve a human expert that adds tags to raw data to show a model the target attributes (answers). In simple terms, a label is basically a description showing a model what it is expected to predict.

Supervised learning has a few limitations as,

- *slow* (it requires human experts to manually label training examples one by one) and
- *costly* (a model should be trained on the large volumes of hand-labeled data to provide accurate predictions).

Unsupervised learning, on the other hand, is when a model tries to mine hidden patterns, differences, and similarities in unlabeled data by itself, without human supervision. Hence the name. Within this method, data points are grouped into clusters based on similarities.

While unsupervised learning is a cheaper way to perform training tasks, it isn't a silver bullet. Commonly it is,

- has a *limited area of applications* (mostly for clustering purposes) and
- provides *less accurate results*.

Semi-supervised learning bridges supervised learning and unsupervised learning techniques to solve their key challenges. With it, you train an initial model on a few labeled samples and then iteratively apply it to the greater number of unlabeled data.

- Unlike unsupervised learning, SSL works for a variety of problems from classification and regression to clustering and association.
- Unlike supervised learning, the method uses small amounts of labeled data and also large amounts of unlabeled data, which reduces expenses on manual annotation and cuts data preparation time.

<u>Assumptions followed by Semi-Supervised Learning</u>

To work with the unlabeled dataset, there must be a relationship between the objects. To understand this, semi-supervised learning uses any of the following assumptions:

- **Continuity Assumption:** As per the continuity assumption, the objects near each other tend to share the same group or label. This assumption is also used in supervised learning, and the datasets are separated by the decision boundaries. But in semi-supervised, the decision boundaries are added with the smoothness assumption in low-density boundaries.
- **Cluster assumptions-** In this assumption, data are divided into different discrete clusters. Further, the points in the same cluster share the output label.
- **Manifold assumptions-** This assumption helps to use distances and densities, and this data lie on a manifold of fewer dimensions than input space.
- The dimensional data are created by a process that has less degree of freedom and may be hard to model directly. **(This assumption becomes practical if high).**

Working of Semi-Supervised Learning

Semi-supervised learning uses pseudo labeling to train the model with less labeled training data than supervised learning. The process can combine various neural network models and training ways.

- Firstly, it trains the model with less amount of training data similar to the supervised learning models. The training continues until the model gives accurate results.

- The algorithms use the unlabeled dataset with pseudo labels in the next step, and now the result may not be accurate.
- Now, the labels from labeled training data and pseudo labels data are linked together.
- The input data in labeled training data and unlabeled training data are also linked.
- In the end, again train the model with the new combined input as did in the first step. It will reduce errors and improve the accuracy of the model.

Semi-supervised learning models applications

- Speech Analysis
- Web content classification
- Protein sequence classification
- Text document classifier

Questions to revise:

Part A

1. Define uncertainty. Give a solution for this.
2. Flip a coin. What is the chance of it landing heads side up?
3. Give Baye's theorem with defined terms.
4. Define conditional probability with the help of a venn diagram.
5. Two dice are thrown at a time. What the probability of getting an 8 as its sum.
6. Differentiate posterior probability and prior probability.
7. What is the significance of naïve Bayesian classifier?
8. What is mean by overfitting? How can we overcome it?
9. What is meant by regularization? List out its types?
10. Differentiate between shallow network and deep network.
11. What is meant by batch normalization?
12. Define VC dimension.
13. What are the layers in a CNN architecture?
14. Define GAN. List out its types?
15. What are the types of activation functions in a neural network?
16. Briefly explain the working of a semi supervised learning algorithm.

Part B

1. What is mean by overfitting? Explain in detail about the techniques to solve this problem.
2. With a neat sketch, explain the algorithm steps of a back propagation network.
3. Define batch normalization. Explain in detail about its working.
4. Define CNN. Explain its elements and layers in detail with neat sketches.
5. How can a deep learning network generate images?
6. Discuss the pros and cons of supervised and unsupervised learning algorithms. Relate the significance of semi-supervised learning algorithm with its working principle.
7. Find out the probability of playing tennis for a set of attributes in the table under the given condition (Outlook = sunny, Temperature = cool, Humidity = high, Wind = strong)

Day	**Outlook**	**Temperature**	**Humidity**	**Wind**	**Play Tennis**
1	Sunny	Hot	High	Weak	No
2	Sunny	Hot	High	Strong	No
3	Overcast	Hot	High	Weak	Yes
4	Rain	Mild	High	Weak	Yes
5	Rain	Cool	Normal	Weak	Yes
6	Rain	Cool	Normal	Strong	No
7	Overcast	Cool	Normal	Strong	Yes
8	Sunny	Mild	High	Weak	No
9	Sunny	Cool	Normal	Weak	Yes
10	Rain	Mild	Normal	Weak	Yes
11	Sunny	Mild	Normal	Strong	Yes
12	Overcast	Mild	High	Strong	Yes
13	Overcast	Hot	Normal	Weak	Yes
14	Rain	Mild	High	Strong	No