UNIT I INTRODUCTION TO DEEP LEARNING

Introduction to machine learning - Linear models (SVMs and Perceptron's, logistic regression)- Introduction to Neural Nets: What are a shallow network computes- Training a network: loss functions, back propagation and stochastic gradient descent- Neural networks as universal function approximates.

1 Definition of Machine Learning [ML]:

Well posed learning problem: "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E."(Tom Michel)

Machine Learning (ML) is an algorithm that works on consequently through experience and by the utilization of information. It is viewed as a piece of AI. ML calculations assemble a model dependent on example information (Data), known as "training Data or information", to settle on forecasts or choices without being unequivocally customized to do as such. AI calculations are utilized in a wide assortment of utilizations, for example, in medication, email sifting, discourse acknowledgment, and Computer vision, where it is troublesome or impractical to foster customary models to play out the needed tasks.

ML includes PCs finding how they can perform tasks without being expressly modified to do as such. It includes Systems that can perform tasks without being expressly modified to do as such. It includes models gaining data so they can do certain specific applications. For basic undertakings appointed to Models, it is feasible to models advising the machine how to execute all means needed to tackle the current issue; on the systems part, no learning is required. For further developed undertakings, it tends to be trying for a human to physically make the required calculations. For more advanced tasks, it can be challenging for a human to manually create the needed algorithms. In practice, it can turn out to be more effective to help the machine develop its own algorithm, rather than having human programmers specify every needed step.

The term *machine learning* was coined in 1959 by Arthur Samuel, an American IBMer and pioneer in the field of computer gaming and artificial intelligence.

1.1 Fundamentals of ANN

a specific application, such as pattern recognition or data classification, through a learning process. Learning in biological systems involves adjustments to the synaptic connections that exist between the neurons. This is true of ANNs as well.

1.2 The Biological Neuron

The human brain consists of a large number, more than a billion of neural cells that process information. Each cell works like a simple processor. The massive interaction between all cells and

their parallel processing only makes the brain's abilities possible. Figure 1 represents a human biological nervous unit. Various parts of biological neural network(BNN) is marked in Figure 1.

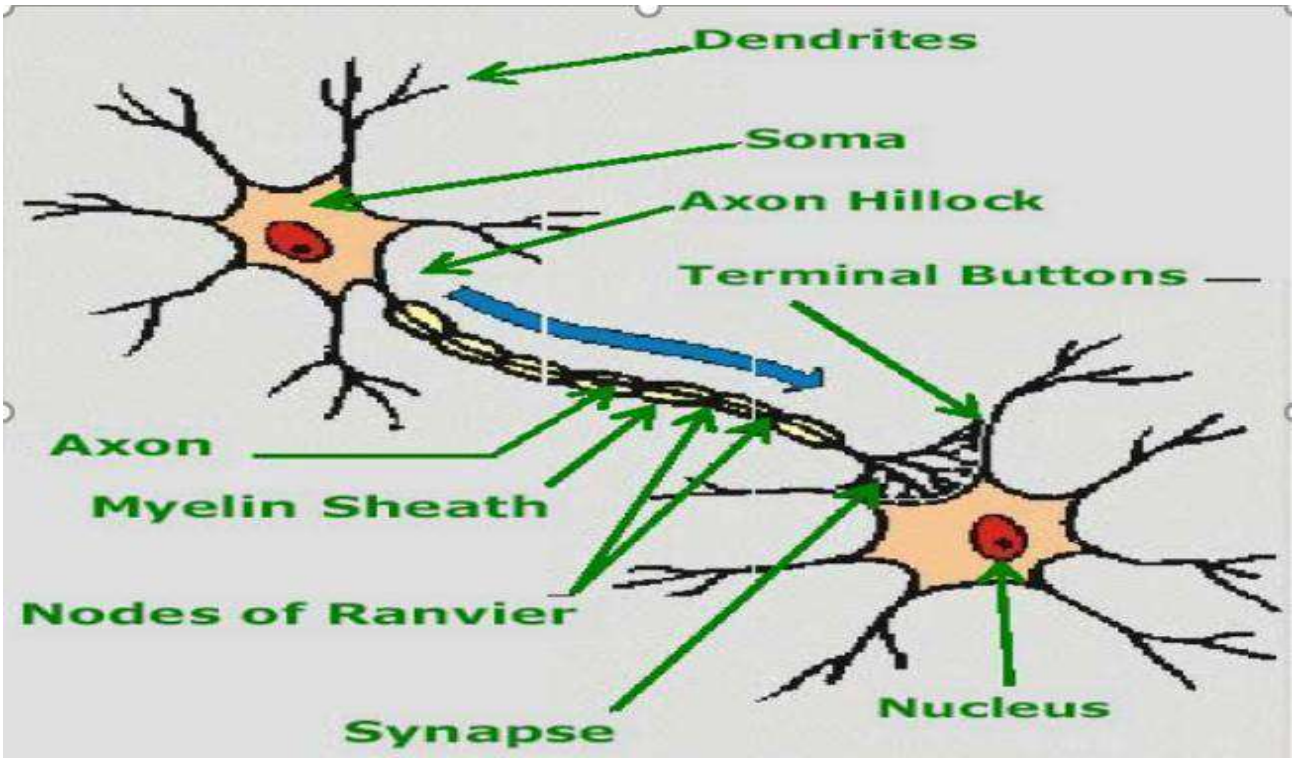


Figure 1: Biological Neural Network

Dendrites are branching fibres that extend from the cell body or soma.

Soma or cell body of a neuron contains the nucleus and other structures, support chemical processing and production of neurotransmitters.

Axon is a singular fiber carries information away from the soma to the synaptic sites of other neurons (dendrites ans somas), muscels, or glands.

Axon hillock is the site of summation for incoming information. At any moment, the collective influence of all neurons that conduct impulses to a given neuron will determine whether or n ot an action potential will be initiated at the axon hillock and propagated along the axon.

Myelin sheath consists of fat-containing cells that insulate the axon from electrical activity. This insulation acts to increase the rate of transmission of signals. A gap exists between each myelin sheath cell along the axon. Since fat inhibits the propagation of electricity, the signals jump from one gap to the next.

Nodes of Ranvier are the gaps (about 1 μm) between myelin sheath cells. Since fat serves as a good insulator, the myelin sheaths speed the rate of transmission of an electrical impulse along the axon.

Synapse is the point of connection between two neurons or a neuron and a muscle or a gland. Electrochemical communication between neurons take place at these junctions.

Terminal buttons of a neuron are the small knobs at the end of an axon that release chemicals called neurotransmitters.

Information flow in a neural cell

The input/output and the propagation of information are shown below.

1.3. Artificial neuron model

An artificial neuron is a mathematical function conceived as a simple model of a real (biological) neuron.

- The McCulloch-Pitts Neuron
  This is a simplified model of real neurons, known as a Threshold Logic Unit.
- A set of input connections brings in activations from other neuron.
- A processing unit sums the inputs, and then applies a non-linear activation function (i.e. squashing/transfer/threshold function).
- An output line transmits the result to other neurons.

1.3.1 Basic Elements of ANN:

Neuron consists of three basic components –weights, thresholds and a single activation function. An Artificial neural network(ANN) model based on the biological neural sytems is shown in figure 2.

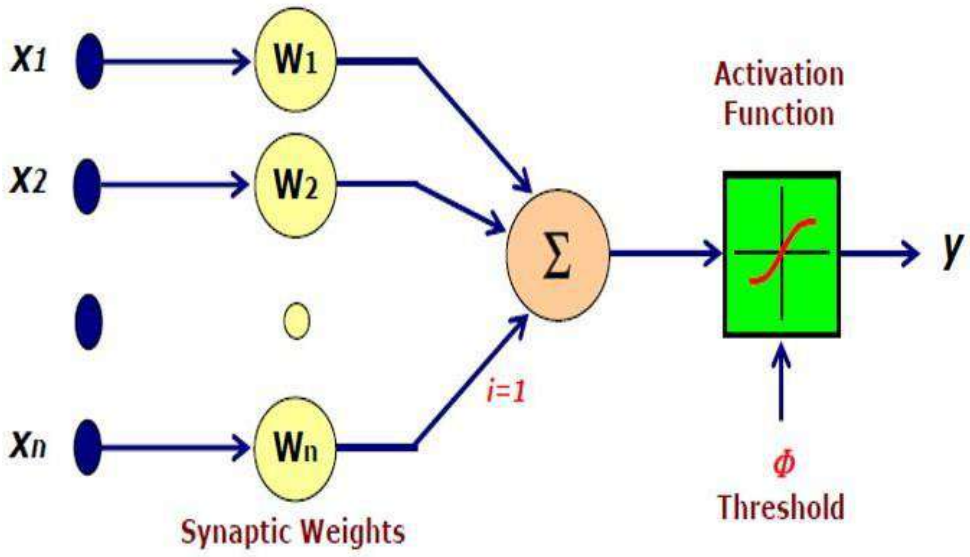


Figure 2: Basic Elements of Artificial Neural Network

1.4 Different Learning Rules

A brief classification of Different Learning algorithms is depicted in figure 3.

- **Training: It is the process in which the network is taught to change its weight and bias.**
- **Learning: It is the internal process of training where the artificial neural system learns to update/adapt the weights and biases.**

Different Training /Learning procedure available in ANN are

- **Supervised learning**
- **Unsupervised learning**
- **Reinforced learning**
- **Hebbian learning**
- **Gradient descent learning**
- **Competitive learning**
- **Stochastic learning**

1.4.1. Requirements of Learning Laws:

- **Learning Law should lead to convergence of weights**
- **Learning or training time should be less for capturing the information from the training pairs**
- **Learning should use the local information**
- **Learning process should able to capture the complex non linear mapping available between the input & output pairs**
- **Learning should able to capture as many as patterns as possible**
- **Storage of pattern information's gathered at the time of learning should be high for the given network**

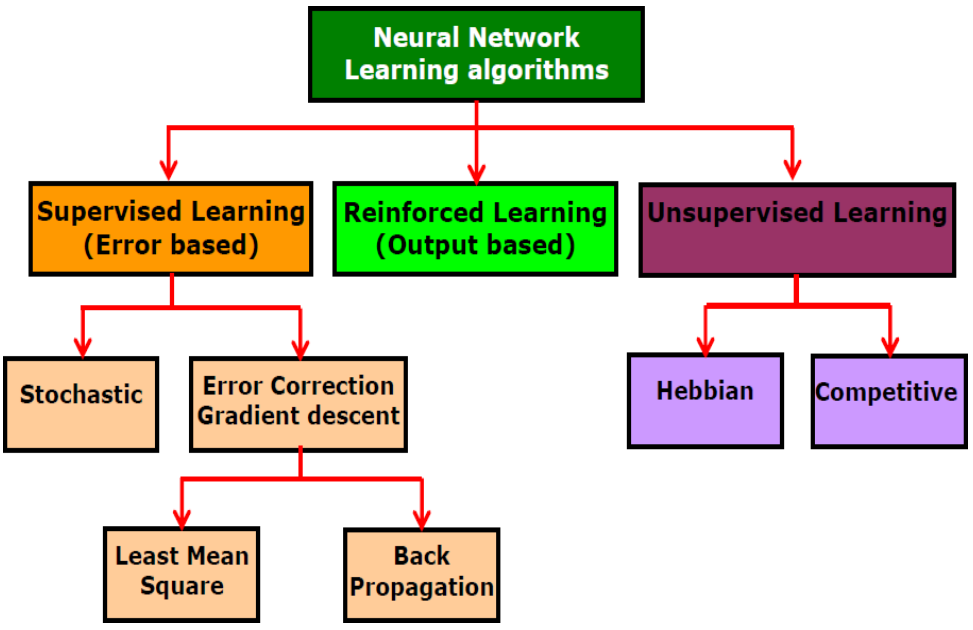


Figure 3: Different Training methods of Artificial Neural Network

1.4.1.1.Supervised learning :

Every input pattern that is used to train the network is associated with an output pattern which is the target or the desired pattern.

A teacher is assumed to be present during the training process, when a comparison is made between the network's computed output and the correct expected output, to determine the error.The error can then be used to change network parameters, which result in an improvement in performance.

1.4.1.2 Unsupervised learning:

In this learning method the target output is not presented to the network.It is as if there is no teacher to present the desired patterns and hence the system learns of its own by discovering and adapting to structural features in the input patterns.

1.4.1.3 Reinforced learning:

In this method, a teacher though available, doesnot present the expected answer but only indicates if the computed output correct or incorrect.The information provided helps the network in the learning process**.**

1.4.1.4 Hebbian learning:

This rule was proposed by Hebb and is based on correlative weight adjustment.This is the oldest learning mechanism inspired by biology.In this, the input-output pattern pairs (x_i, y_i) are associated by the weight matrix W, known as the correlation matrix.

It is computed as

$$W = \sum_{i=1}^{n} x_i y_i^T \qquad \text{---------- eq(1)}$$

Here y_i^T is the transposeof the associated output vector y_i.Numerous variants of the rule have been proposed.

1.4.1.5 Gradient descent learning:

This is based on the minimization of error E defined in terms of weights and activation function of the network.Also it is required that the activation function employed by the network is differentiable, as the weight update is dependent on the gradient of the error E.

Thus if Δw_{ij} is the weight update of the link connecting the i^{th} and j^{th} neuron of the two neighbouring layers, then Δw_{ij} is defined as,

$$\Delta w_{ij} = \eta \frac{\partial E}{\partial w_{ij}} \qquad \text{---------- eq(2)}$$

Where, η is the learning rate parameter and $\frac{\partial E}{\partial w_{ij}}$ is the error gradient with reference to the weight w_{ij}.

1.5 Perceptron Model

1.5.1 Simple Perceptron for Pattern Classification

Perceptron network is capable of performing pattern classification into two or more categories. The perceptron is trained using the perceptron learning rule. We will first consider classification into two categories and then the general multiclass classification later. For classification

into only two categories, all we need is a single output neuron. Here we will use bipolar neurons. The simplest architecture that could do the job consists of a layer of N input neurons, an output layer with a single output neuron, and no hidden layers. This is the same architecture as we saw before for Hebb learning. However, we will use a different transfer function here for the output neurons as given below in eq (7). Figure 7 represents a single layer perceptron network.

$$y = \begin{cases} 1 & \text{if } y\_in > \theta \\ 0 & \text{if } -\theta \le y\_in \le \theta \\ -1 & \text{if } y\_in < -\theta \end{cases}$$

.....eq (7)

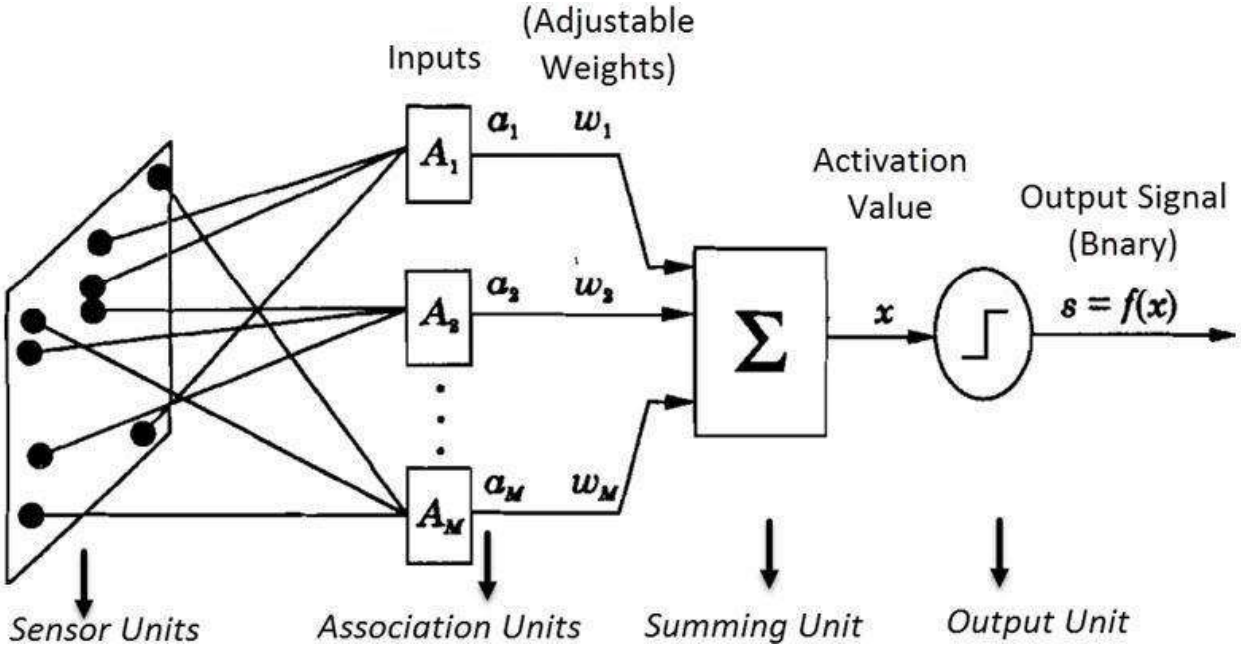


Figure 4: Single Layer Perceptron

Equation 7 gives the bipolar activation function which is the most common function used in the perceptron networks. Figure 7 represents a single layer perceptron network. The inputs arising from the problem space are collected by the sensors and they are fed to the aswociation units.Association units are the units which are responsible to associate the inputs based on their similarities. This unit groups the similar inputs hence the name association unit. A single input from each group is given to the summing unit.Weights are randomnly fixed intially and assigned to this inputs. The net value is calculate by using the expression

$$x = \Sigma w_i a_i - \theta$$ _____eq(8)

This value is given to the activation function unit to get the final output response.The actual output is compared with the Target or desired .If they are same then we can stop training else the weights haqs to be updated .It means there is error .Error is given as **δ = b-s** , where b is the desired

/ Target output and S is the actual outcome of the machinehere the weights are updated based on the perceptron Learning law as given in equation 9.

Weight change is given as $\Delta w = \eta \delta a_i$. So new weight is given as

$$W_{i\,(new)} = W_{i\,(old)} + \text{Change in weight vector } (\Delta w) \_\_\_\_\_ \text{eq(9)}$$

1.5.2. Perceptron Algorithm

Step 1: Initialize weights and bias.For simplicity, set weights and bias to zero.Set learning rate in the range of zero to one.

- Step 2: While stopping condition is false do steps 2-6
- Step 3: For each training pair s:t do steps 3-5
- Step 4: Set activations of input units $x_i = ai$
- Step 5: Calculate the summing part value Net = Σ aiwi-θ
- Step 6: Compute the response of output unit based on the activation functions
- Step 7: Update weights and bias if an error occurred for this pattern(if *yis not equal to t)*

  Weight (new) = w_i(old) + atx_i , & bias (new) = b(old) + at

  Else *w_i(new) = w_i(old) & b(new) = b(old)*

- Step 8: Test Stopping Condition

1.5.3. Limitations of single layer perceptrons:

- Uses only Binary Activation function
- Can be used only for Linear Networks
- Since uses Supervised Learning ,Optimal Solution is provided
- Training Time is More
- Cannot solve Linear In-separable Problem

1.5.4. Multi-Layer Perceptron Model:

Figure 8 is the general representation of Multi layer Perceptron network.Inbetween the input and output Layer there will be some more layers also known as Hidden layers.

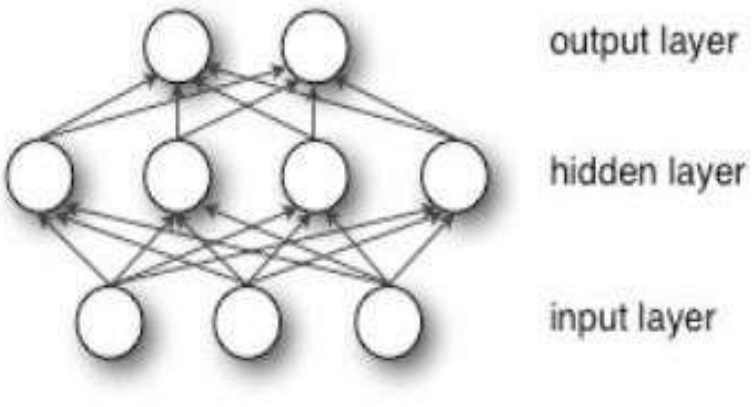


Figure 5: Multi-Layer Perceptron

1.5.5. Multi Layer Perceptron Algorithm

1. Initialize the weights (W_i) & Bias (B_0) to small random values near Zero
2. Set learning rate η or α in the range of "0" to "1"
3. Check for stop condition. If stop condition is false do steps 3 to 7
4. For each Training pairs do step 4 to 7
5. Set activations of Output units: $x_i = s_i$ for i=1 to N
6. Calculate the output Response

   $y_{in} = b_0 + \Sigma x_i w_i$
7. Activation function used is Bipolar sigmoidal or Bipolar Step functions

   For Multi Layer networks, based on the number of layers steps 6 & 7 are repeated
8. If the Targets is (not equal to) = to the actual output (Y), then update weights and bias based on Perceptron Learning Law

   $W_{i\,(new)} = W_{i\,(old)}$ + Change in weight vector

   Change in weight vector $= \eta t_i x_i$

   Where η = Learning Rate

   t_i = Target output of i^{th} unit

   x_i = i^{th} Input vector

   $b_{0(new)} = b_{0\,(old)}$ + Change in Bias

   Change in Bias $= \eta t_i$

   Else $W_{i\,(new)} = W_{i\,(old)}$

   $b_{0(new)} = b_{0\,(old)}$
9. Test for Stop condition

1.6. linearly seperable & Linear in separable tasks:

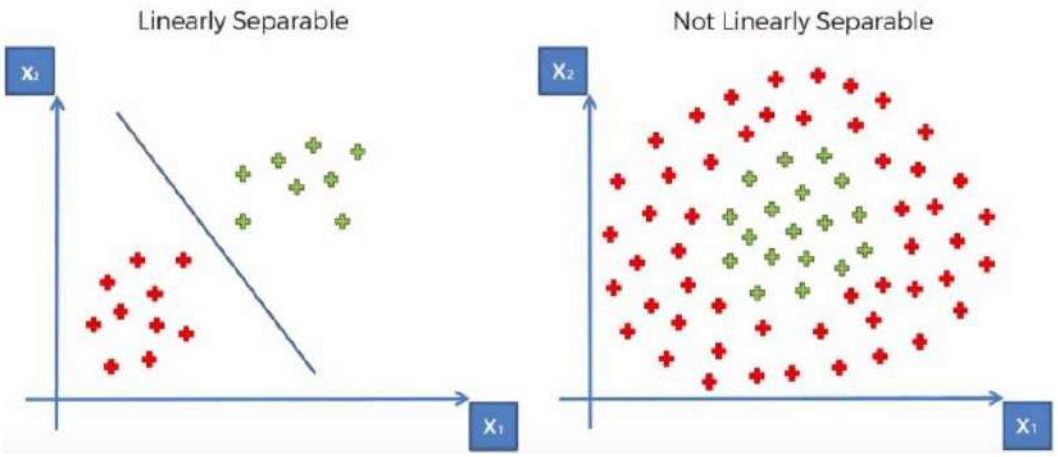


Figure 6: Representation of Linear seperable & Linear-in separable Tasks

Perceptron are successful only on problems with a linearly separable solution sapce.Figure 9 represents both linear separable as well as linear in seperable problem.Perceptron cannot handle, in particular, tasks which are not linearly separable.(Known as linear inseparable problem).Sets of points in two dimensional spaces are linearly separable if the sets can be seperated by a straight line.Generalizing, a set of points in n-dimentional space are that can be seperated by a straight line.is called Linear seperable as represented in figure 9.

Single layer perceptron can be used for linear separation.Example AND gate.But it cant be used for non linear ,inseparable problems.(Example XOR Gate).Consider figure 10.

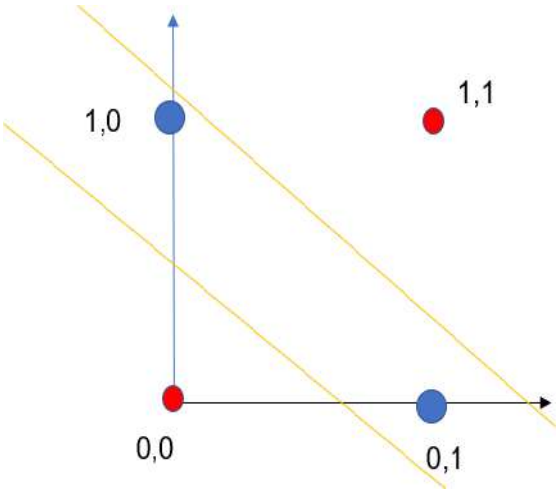


Figure 7: XOR representation (Linear-in separable Task)

Here a single decision line cannot separate the Zeros and Ones Linearly.At least Two lines are required to separate Zeros and Onesas shown in Figure 10. Hence single layer networks can not be used to solve inseparable problems. To over come this problem we go for creation of **convex regions**.

Convex regions can be created by multiple decision lines arising from multi layer networks.Single layer network cannot be used to solve inseparable problem.Hence we go for multilayer network there by creating convex regions which solves the inseparable problem.

1.6.1 Convex Region:

Select any Two points in a region and draw a straight line between these two points. If the points selected and the lines joining them both lie inside the region then that region is **known as convex regions.**

1.6.2. Types of convex regions

(a) Open Convex region (b) Closed Convex region

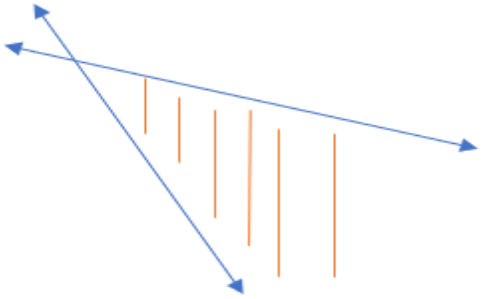


Figure 8: Open convex region

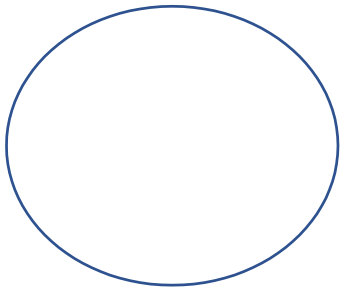


Figure 9 A: Circle - Closed convex region

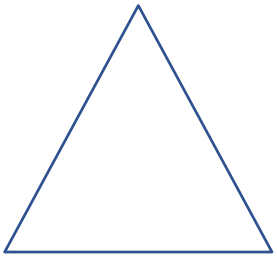


Figure 9 B: Triangle - Closed convex region

1.7. Logistic Regression

Logistic regression is a probabilistic model that organizes the instances in terms of probabilities. Because the classification is probabilistic, a natural method for optimizing the parameters is to ensure that the predicted probability of the observed class for each training occurrence is as large as possible. This goal is achieved by using the notion of maximumlikelihood estimation in order to learn the parameters of the model. The likelihood of the training data is defined as the product of the probabilities of the observed labels of each training instance. Clearly, larger values of this objective function are better. By using the negative logarithm of this value, one obtains a loss function in minimization form. Therefore, the output node uses the negative log-likelihood as a loss function. This loss function replaces the squared error used in the Widrow-Hoff method. The output layer can be formulated with the sigmoid activation function, which is very common in neural network design.

- Logistic regression is another supervised learning algorithm which is used to solve the classification problems. In classification problems, we have dependent variables in a binary or discrete format such as 0 or 1.

- Logistic regression algorithm works with the categorical variable such as 0 or 1, Yes or No, True or False, Spam or not spam, etc.

- It is a predictive analysis algorithm which works on the concept of probability.

- Logistic regression is a type of regression, but it is different from the linear regression algorithm in the term how they are used.

- Logistic regression uses sigmoid function or logistic function which is a complex cost function. This sigmoid function is used to model the data in logistic regression. The function can be represented as:

$$f(x)= \frac{1}{1+e^{-x}}$$

Where f(x)= Output between the 0 and 1 value.
x= input to the function
e= base of natural logarithm.

When we provide the input values (data) to the function, it gives the S-curve as follows: It uses the concept of threshold levels, values above the threshold level are rounded up to 1, and values below the threshold level are rounded up to 0.

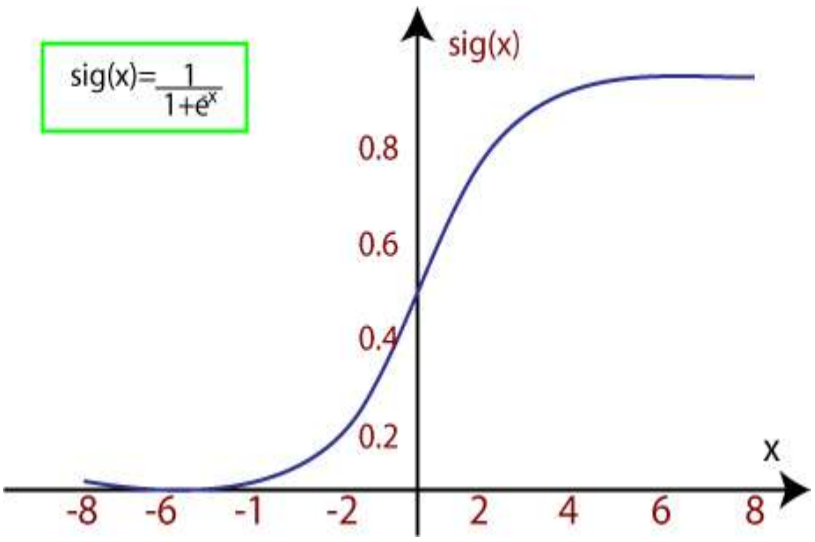


Figure 10: Circle – Logistic Function

1.8. Support Vector Machines

Support Vector Machine or SVM is one of the most popular Supervised Learning algorithms, which is used for Classification as well as Regression problems. However, primarily, it is used for Classification problems in Machine Learning. The goal of the SVM algorithm is to create the best line or decision boundary that can segregate n-dimensional space into classes so that we can easily put the new data point in the correct category in the future. This best decision boundary is called a hyperplane. SVM chooses the extreme points/vectors that help in creating the hyperplane. These extreme cases are called as support vectors, and hence algorithm is termed as Support Vector Machine. Consider the below diagram in which there are two different categories that are classified using a decision boundary or hyperplane:

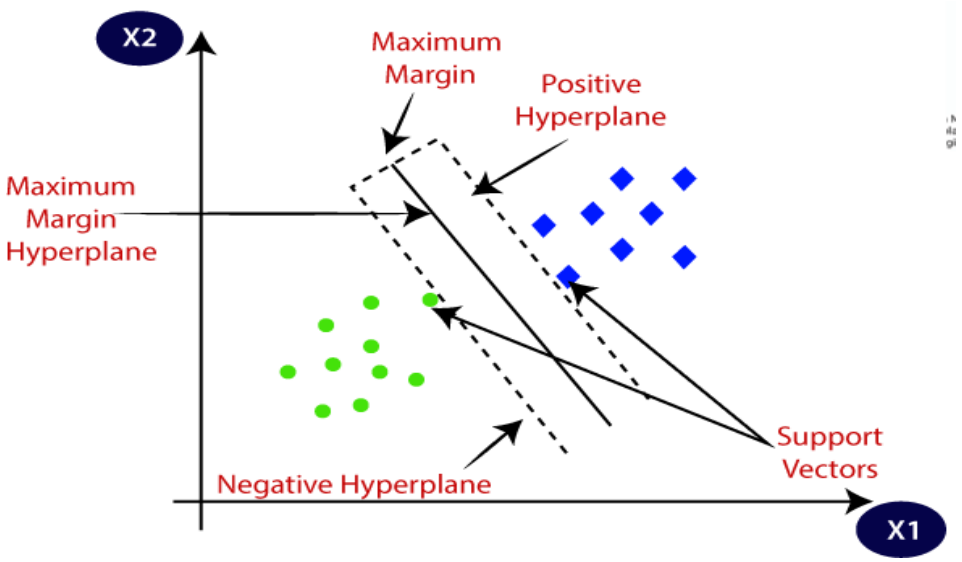


Figure 11: SVM – Classification

1.8.1. SVM can be of two types:

Linear SVM: Linear SVM is used for linearly separable data, which means if a dataset can be classified into two classes by using a single straight line, then such data is termed as linearly separable data, and classifier is used called as Linear SVM classifier.

Non-linear SVM: Non-Linear SVM is used for non-linearly separated data, which means if a dataset cannot be classified by using a straight line, then such data is termed as non-linear data and classifier used is called as Non-linear SVM classifier

Support Vectors:

The data points or vectors that are the closest to the hyperplane and which affect the position of the hyperplane are termed as Support Vector. Since these vectors support the hyperplane, hence called a Support vector.

1.8.2. Linear SVM:

The working of the SVM algorithm can be understood by using an example. Suppose we have a dataset that has two tags (green and blue), and the dataset has two features x1 and x2. We want a classifier that can classify the pair (x1, x2) of coordinates in either green or blue. Consider the below image figure11. It is 2-d space so by just using a straight line, we can easily separate these two classes. But there can be multiple lines that can separate these classes. Consider the below image:

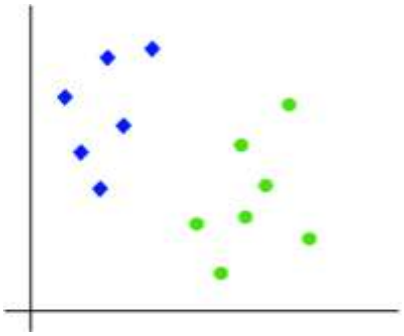


Figure 12A: SVM – Input Space

Hence, the SVM algorithm helps to find the best line or decision boundary; this best boundary or region is called as a hyperplane. SVM algorithm finds the closest point of the lines from both the classes. These points are called support vectors. The distance between the vectors and the hyperplane is called as margin. And the goal of SVM is to maximize this margin. The hyperplane with maximum margin is called the optimal hyperplane.

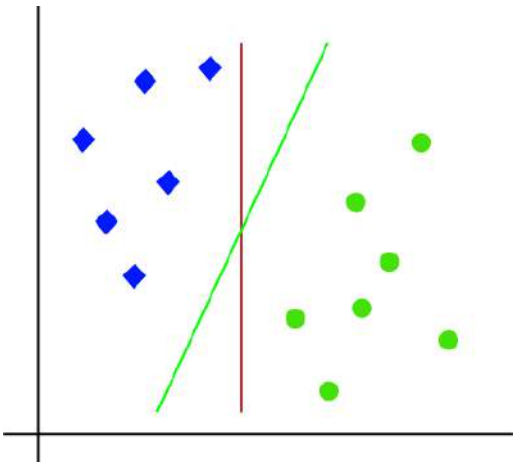


Figure 12B: SVM – Linear Classification

1.9. Gradient Descent:

Gradient Descent is a popular optimization technique in Machine Learning and Deep Learning, and it can be used with most, if not all, of the learning algorithms. A gradient is the slope of a function. It measures the degree of change of a variable in response to the changes of another variable. Mathematically, Gradient Descent is a convex function whose output is the partial derivative of a set of parameters of its inputs. The greater the gradient, the steeper the slope.Starting from an initial value, Gradient Descent is run iteratively to find the optimal values of the parameters to find the minimum possible value of the given cost function.

1.9.1. Types of Gradient Descent:

Typically, there are three types of Gradient Descent:

1. Batch Gradient Descent
2. Stochastic Gradient Descent
3. Mini-batch Gradient Descent

1.9.2. Stochastic Gradient Descent (SGD):

The word ‘stochastic‘ means a system or a process that is linked with a random probability. Hence, in Stochastic Gradient Descent, a few samples are selected randomly instead of the whole data set for each iteration. In Gradient Descent, there is a term called “batch” which denotes the total number of samples from a dataset that is used for calculating the gradient for each iteration. In typical Gradient Descent optimization, like Batch Gradient Descent, the batch is taken to be the whole dataset. Although, using the whole dataset is really useful for getting to the minima in a less noisy and less random manner, but the problem arises when our datasets gets big. Suppose, you have a million samples in your dataset, so if you use a typical Gradient Descent optimization technique, you will have to use all of the one million samples for completing one iteration while performing the Gradient Descent, and it has to be done for every iteration until the minima is reached. Hence, it becomes computationally very expensive to perform.

Reference Books:

1. B. Yegnanarayana, “Artificial Neural Networks” Prentice Hall Publications.
2. Simon Haykin, “Artificial Neural Networks”, Second Edition, Pearson Education.
3. Laurene Fausett, “Fundamentals of Neural Networks, Architectures, Algorithms and Applications”, Prentice Hall publications.
4. Cosma Rohilla Shalizi, Advanced Data Analysis from an Elementary Point of View, 2015.
5. 2. Deng & Yu, Deep Learning: Methods and Applications, Now Publishers, 2013.
6. 3. Ian Goodfellow, Yoshua Bengio, Aaron Courville, Deep Learning, MIT Press, 2016.
7. 4. Michael Nielsen, Neural Networks and Deep Learning, Determination Press, 2015.

Note: For further reference, kindly refer the class notes, PPTs, Video lectures available in the Learning Management System (Moodle)

********** ALL THE BEST **********

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 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 Minsky pointed 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 appear in 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.

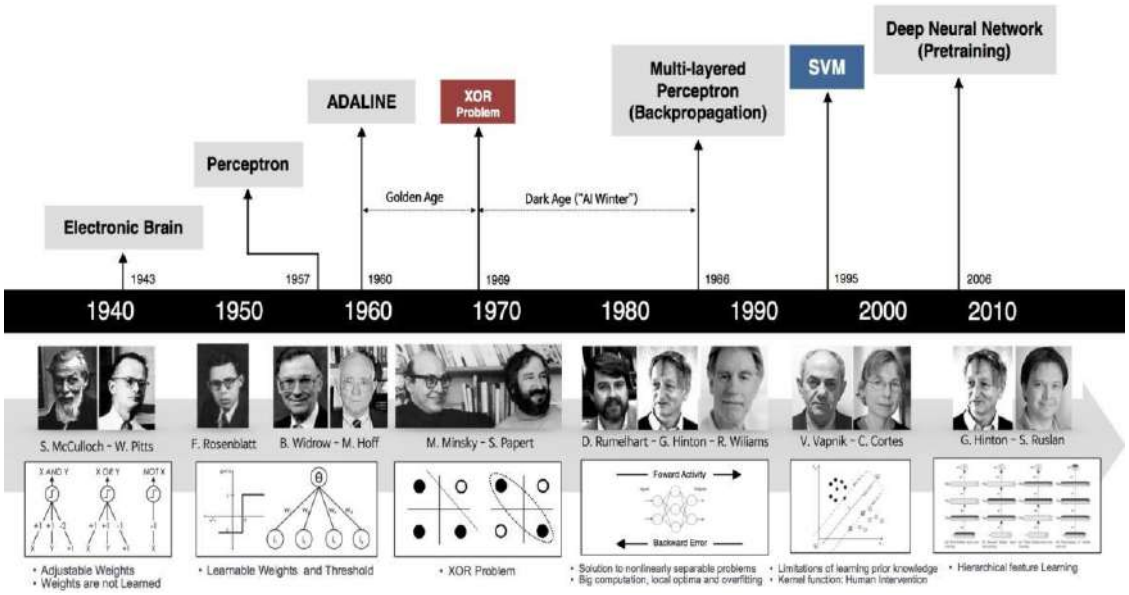
  Most of the improvement in neural network performance from 1986 to 2015 can be attributed to two factors. First, larger datasets have reduced the degree to which statistical generalization is a challenge for neural networks. Second, neural networks have become much larger, because of more powerful computers and better software infrastructure.A small number of algorithmic changes have also improved the performance of neural networks noticeably. One of these algorithmic changes was the replacement of mean squared error with the cross-entropy family of loss functions. Mean squared error was popular in the 1980s and 1990s but was gradually replaced by cross-entropy losses and the principle of maximum likelihood as ideas spread between the statistics community and the machine learning community.

  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 and Neo-Cognitron (Fukushima, 1975, 1980).

  For small datasets, Jarrett et al. (2009) observed that using rectifying nonlinearities is even 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. When more 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 far easier in deep rectified linear networks than in deep networks that have curvature or two-sided saturation 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 model design will improve their performance even further.



2.1 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.

Real world data is chaotic. Since deep learning systems utilize real world data, they require a tool to handle the chaoticness.

2.2 Back Propagation Networks (BPN)

2.2.1. 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 is available
- Single layer networks cannot capture the complex information's available in the training pairs

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

2.2.2. Multi-Layer Networks

- Any neural network which has at least one layer in between input and output layers 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 next higher layer
- Hidden layer and output layer neural units process the information's feed to them and produce an appropriate output
- Multi -layer networks provide optimal solution for arbitrary classification problems
- Multi -layer networks use linear discriminants, where the inputs are non linear

2.2.3. 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 Descent Law 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 rate when compared with Perceptron Law. It is the extended version of Perceptron Training Law. Limitations of this law is the Local minima problem. Due to this the convergence speed reduces, but it is better than perceptron's. Figure 1 represents a BPN network architecture. Even though Multi level perceptron's can be used they are flexible and efficient that BPN. In figure 1 the weights between input and the hidden portion is considered as Wij and the weight between first hidden to the next layer is considered as V_{jk}. This network is valid only for Differential Output functions. 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

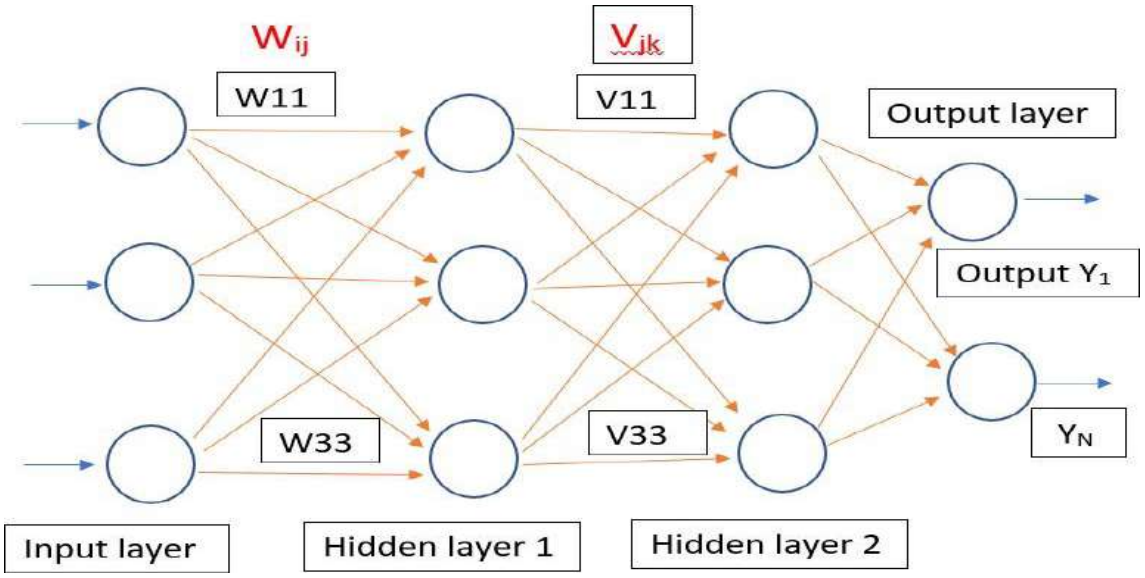


Figure 1: Back Propagation Network

2.2.4. BPN Algorithm

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 (next hidden)

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 layer

Step 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

2.2.5 Merits

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

2.2.6. Demerits

- 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
- Temporal Instability
- Network Paralysis
- Training time is more for Complex problems

2.3 Regularization

A fundamental problem in machine learning is how to make an algorithm that will 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.

Definition: - "any modification we make to a learning algorithm that is intended to reduce its generalization error but not its training error."

❖ In the context of deep learning, most regularization strategies are based on regularizing estimators.

❖ Regularization of an estimator works by trading increased bias for reduced variance.

An effective regularizer is one that makes a profitable trade, reducing variance significantly while not overly increasing the bias.

- Many regularization approaches are based on limiting the capacity of models, such as neural networks, linear regression, or logistic regression, by adding a parameter norm penalty Ω(θ) to the objective function J. We denote the regularized objective function by J˜

$$\tilde{J}(\theta; X, y) = J(\theta; X, y) + \alpha\Omega(\theta)$$

where $\alpha \in [0, \infty)$ is a hyperparameter that weights the relative contribution of the norm penalty term, Ω, relative to the standard objective function J. Setting α to 0 results in no regularization. Larger values of α correspond to more regularization.

- The parameter norm penalty Ω that penalizes only the weights of the affine transformation at each layer and leaves the biases unregularized.

2.3.1 L2 Regularization

One of the simplest and most common kind of parameter norm penalty is L2 parameter & it's also called commonly as weight decay. This regularization strategy drives the weights closer to the origin by adding a regularization term $\Omega(\theta) = \frac{1}{2}||w||_2^2$. L2 regularization is also known as ridge regression or Tikhonov regularization. To simplify, we assume no bias parameter, so θ is just w. Such a model has the following total objective function.

$$\tilde{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) = \frac{\alpha}{2}\boldsymbol{w}^\top \boldsymbol{w} + J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}),$$

with the corresponding parameter gradient

$$\nabla_{\boldsymbol{w}} \tilde{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) = \alpha\boldsymbol{w} + \nabla_{\boldsymbol{w}} J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}).$$

To take a single gradient step to update the weights, we perform this update

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \epsilon\left(\alpha\boldsymbol{w} + \nabla_{\boldsymbol{w}} J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y})\right).$$

Written another way, the update is

$$\boldsymbol{w} \leftarrow (1 - \epsilon\alpha)\boldsymbol{w} - \epsilon\nabla_{\boldsymbol{w}} J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}).$$

- We can see that the addition of the weight decay term has modified the learning rule to multiplicatively shrink the weight vector by a constant factor on each step, just before performing the usual gradient update. This describes what happens in a single step.
- The approximation ^J is Given by

$$\hat{J}(\boldsymbol{\theta}) = J(\boldsymbol{w}^*) + \frac{1}{2}(\boldsymbol{w} - \boldsymbol{w}^*)^\top \boldsymbol{H}(\boldsymbol{w} - \boldsymbol{w}^*),$$

Where H is the Hessian matrix of J with respect to w evaluated at w∗.

The minimum of ˆJ occurs where its gradient ∇wˆJ(w) = H(w − w∗) is equal to '0'

To study the eff ect of weight decay,

$$\alpha \tilde{\boldsymbol{w}} + \boldsymbol{H}(\tilde{\boldsymbol{w}} - \boldsymbol{w}^*) = 0$$
$$(\boldsymbol{H} + \alpha \boldsymbol{I})\tilde{\boldsymbol{w}} = \boldsymbol{H}\boldsymbol{w}^*$$
$$\tilde{\boldsymbol{w}} = (\boldsymbol{H} + \alpha \boldsymbol{I})^{-1}\boldsymbol{H}\boldsymbol{w}^*$$

- As α approaches 0, the regularized solution ˜w approaches w*. But what happens as α grows? Because H is real and symmetric, we can decompose it into a diagonal matrix Λ and an orthonormal basis of eigenvectors, Q, such that $H = Q\Lambda Q^T$. Applying Decomposition to the above equation, We Obtain

$$\begin{aligned}\tilde{\boldsymbol{w}} &= (\boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^\top + \alpha \boldsymbol{I})^{-1}\boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^\top \boldsymbol{w}^* \\ &= \left[\boldsymbol{Q}(\boldsymbol{\Lambda} + \alpha \boldsymbol{I})\boldsymbol{Q}^\top\right]^{-1}\boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^\top \boldsymbol{w}^* \\ &= \boldsymbol{Q}(\boldsymbol{\Lambda} + \alpha \boldsymbol{I})^{-1}\boldsymbol{\Lambda}\boldsymbol{Q}^\top \boldsymbol{w}^*.\end{aligned}$$

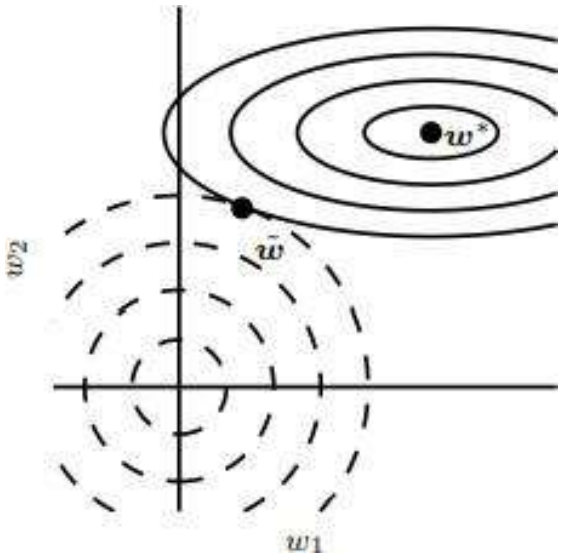


Figure 2: Weight updation effect

The solid ellipses represent contours of equal value of the unregularized objective. The dotted circles represent contours of equal value of the L 2 regularizer. At the point w˜, these competing objectives reach an equilibrium. In the first dimension, the eigenvalue of the Hessian of J is small. The objective function does not increase much when moving horizontally away from w∗ . Because the objective function does not express a strong preference along this direction, the regularizer has a strong effect on this axis. The regularizer pulls w1 close to zero. In the second dimension, the objective function is very sensitive to movements away from w∗ . The corresponding eigenvalue is large, indicating high curvature. As a result, weight decay affects the position of w2 relatively little.

2.3.2 L1 Regularization

While L2 weight decay is the most common form of weight decay, there are other ways to penalize the size of the model parameters. Another option is to use L1 regularization.

- L1 regularization on the model parameter w is defined as the sum of absolute values of the individual parameters.

$$\Omega(\boldsymbol{\theta}) = ||\boldsymbol{w}||_1 = \sum_i |w_i|,$$

L1 weight decay controls the strength of the regularization by scaling the penalty Ω using a positive hyperparameter α. Thus, the regularized objective function J˜(w; X, y) is given by

$$\tilde{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) = \alpha||\boldsymbol{w}||_1 + J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}),$$

with the corresponding gradient as

$$\nabla_{\boldsymbol{w}}\tilde{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) = \alpha \text{sign}(\boldsymbol{w}) + \nabla_{\boldsymbol{w}}J(\boldsymbol{X}, \boldsymbol{y}; \boldsymbol{w}), \longrightarrow \text{Eq-1}$$

By inspecting equation 1, we can see immediately that the effect of L 1 regularization is quite different from that of L 2 regularization. Specifically, we can see that the regularization contribution to the gradient no longer scales linearly with each wi ; instead it is a constant factor with a sign equal to sign(wi).

Quadratic approximation of the L 1 regularized objective function decomposes into a sum over the parameters

$$\hat{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) = J(\boldsymbol{w}^*; \boldsymbol{X}, \boldsymbol{y}) + \sum_i \left[\frac{1}{2}H_{i,i}(\boldsymbol{w}_i - \boldsymbol{w}_i^*)^2 + \alpha|w_i|\right].$$

The problem of minimizing this approximate cost function has an analytical solution with the following form:

$$w_i = \text{sign}(w_i^*) \max\left\{|w_i^*| - \frac{\alpha}{H_{i,i}}, 0\right\}.$$

Consider the situation where w ∗ i > 0 for all i. There are two possible outcomes:

1. The case where $w_i^* \leq \frac{\alpha}{H_{i,i}}$. Here the optimal value of w_i under the regularized objective is simply $w_i = 0$. This occurs because the contribution of $J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y})$ to the regularized objective $\tilde{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y})$ is overwhelmed—in direction i—by the L^1 regularization, which pushes the value of w_i to zero.

2. The case where $w_i^* > \frac{\alpha}{H_{i,i}}$. In this case, the regularization does not move the optimal value of w_i to zero but instead just shifts it in that direction by a distance equal to $\frac{\alpha}{H_{i,i}}$.

2.3.3 Difference between L1 & L2 Parameter Regularization

- L1 regularization attempts to estimate the median of data, L2 regularization makes estimation for the mean of the data in order to evade overfitting.

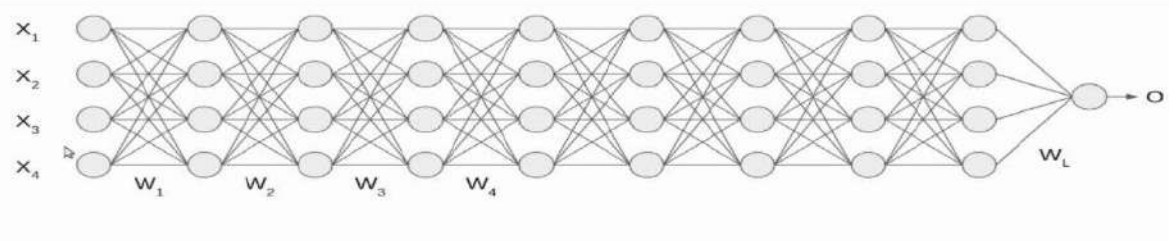
- L1 regularization can add the penalty term in cost function. But L2 regularization appends the squared value of weights in the cost function.
- L1 regularization can be helpful in features selection by eradicating the unimportant features, whereas, L2 regularization is not recommended for feature selection
- L1 doesn't have a closed form solution since it includes an absolute value and it is a non-differentiable function, while L2 has a solution in closed form as it's a square of a weight

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.

2.4 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 to a 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 that our model can generalize appropriately.(Normalization is used to bring the input into a balanced scale/ Range).

Let's understand this through an example, we have a deep neural network as shown in the following image.



Initially, our 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.

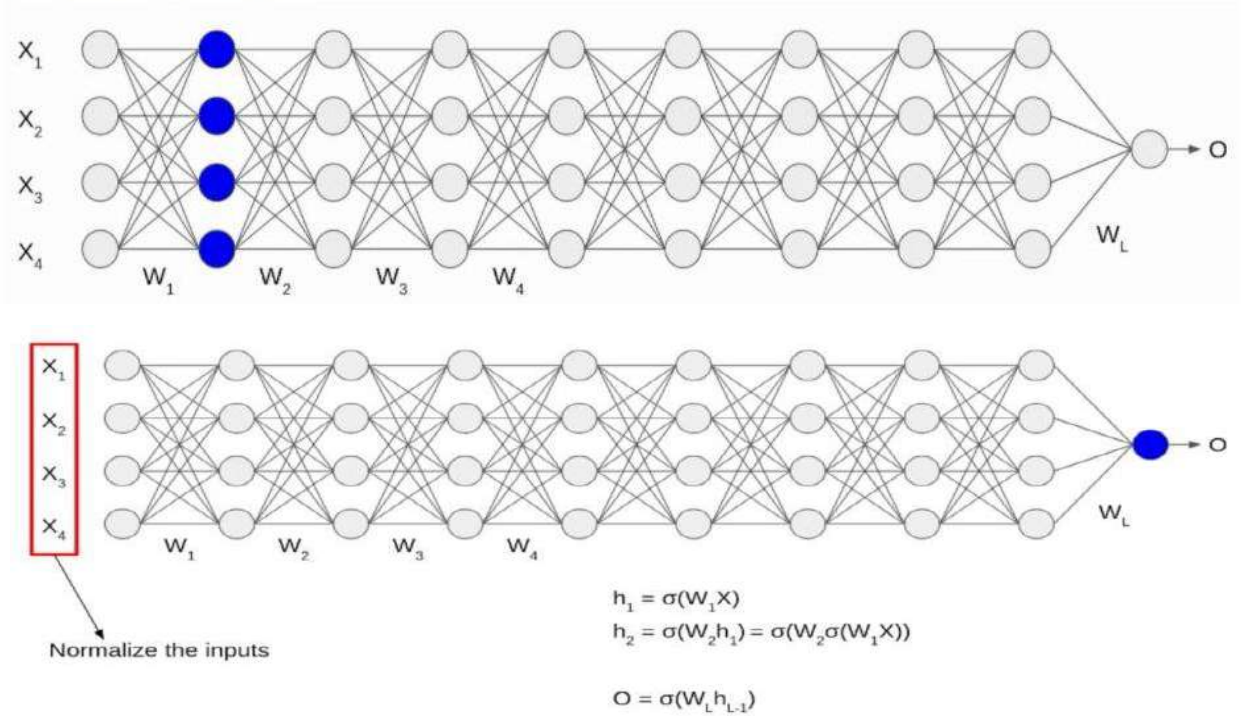


Image Source: https://www.analyticsvidhya.com/blog/2021/03/introduction-to-batch-normalization/

Even though the input X was normalized but the output is no longer on the same scale. The data passes through multiple layers of network with multiple times(sigmoidal) activation functions are applied, which leads to an internal co-variate shift in the data.

This motivates us to move towards Batch Normalization

Normalization is the process of altering the input data to have mean as zero and standard deviation value as one.

2.4.1 Procedure to do Batch Normalization:

(1) Consider the batch input from layer h, for this layer we need to calculate the mean of this hidden activation.
(2) After calculating the mean the next step is to calculate the standard deviation of the hidden activations.
(3) Now we normalize the hidden activations using these Mean & Standard Deviation values. To do this, we subtract the mean from each input and divide the whole value with the sum of standard deviation and the smoothing term (ε).
(4) As the final stage, the re-scaling and offsetting of the input is performed. Here two components of the BN algorithm is used, γ(gamma) and β (beta). These parameters are used for re-scaling (γ) and shifting(β) the vector contains values from the previous operations.

These two parameters are learnable parameters, Hence during the training of neural network, the optimal values of γ and β are obtained and used. Hence we get the accurate normalization of each batch.

2.5. Shallow Networks

Shallow neural networks give us basic idea about deep neural network which consist of only 1 or 2 hidden layers. Understanding a shallow neural network gives us an understanding into what exactly is going on inside a deep neural network A neural network is built using various hidden layers. Now that we know the computations that occur in a particular layer, let us understand how the whole neural network computes the output for a given input *X*. These can also be called the *forward-propagation* equations.

$$Z^{[1]} = W^{[1]T} X + b^{[1]}$$

$$A^{[1]} = \sigma\left(Z^{[1]}\right)$$

$$Z^{[2]} = W^{[2]T} A^{[1]} + b^{[2]}$$

$$\hat{y} = A^{[2]} = \sigma\left(Z^{[2]}\right)$$

1. The first equation calculates the intermediate output **Z[1]** of the first hidden layer.
2. The second equation calculates the final output **A[1]** of the first hidden layer.
3. The third equation calculates the intermediate output **Z[2]** of the output layer.
4. The fourth equation calculates the final output **A[2]** of the output layer which is also the final output of the whole neural network.

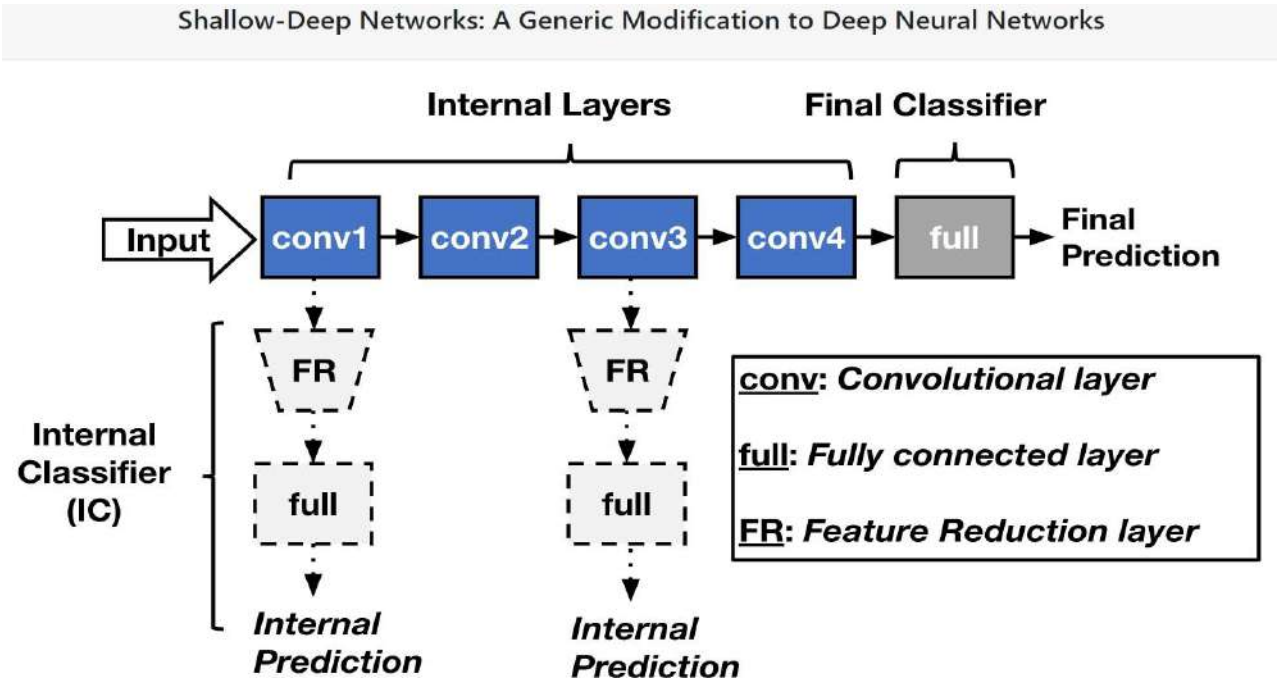


Figure 2:Shallow Networks – Generic Model

2.5.1 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. of Hidden 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, text as inputs
3	Shallow net's needs more parameters to 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 over the input space	DL can compactly express highly complex functions over input space
5	The number of units in a shallow network grows exponentially with task 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

Reference Books:

1. B. Yegnanarayana, "Artificial Neural Networks" Prentice Hall Publications.
2. Simon Haykin, "Artificial Neural Networks", Second Edition, Pearson Education.
3. Laurene Fausett, "Fundamentals of Neural Networks, Architectures, Algorithms and Applications", Prentice Hall publications.
4. Cosma Rohilla Shalizi, Advanced Data Analysis from an Elementary Point of View, 2015.
5. 2. Deng & Yu, Deep Learning: Methods and Applications, Now Publishers, 2013.
6. 3. Ian Goodfellow, Yoshua Bengio, Aaron Courville, Deep Learning, MIT Press, 2016.
7. 4. Michael Nielsen, Neural Networks and Deep Learning, Determination Press, 2015.

Note: For further reference, kindly refer the class notes, PPTs, Video lectures available in the Learning Management System (Moodle)

********** ALL THE BEST **********

UNIT III DIMENTIONALITY REDUCTION

Linear (PCA, LDA) and manifolds, metric learning - Auto encoders and dimensionality reduction in networks - Introduction to Convnet - Architectures – AlexNet, VGG, Inception, ResNet - Training a Convnet: weights initialization, batch normalization, hyperparameter optimization.

3.1 Linear Factor Models:

linear factor models are used as building blocks of mixture models of larger, deep probabilistic models. A linear factor model is defined by the use of a stochastic linear decoder function that generates x by adding noise to a linear transformation of h. It allows us to discover explanatory factors that have a simple joint distribution. A linear factor model describes the data-generation process as follows. (we sample the explanatory factors h from a distribution)

$$h \sim p(h)$$

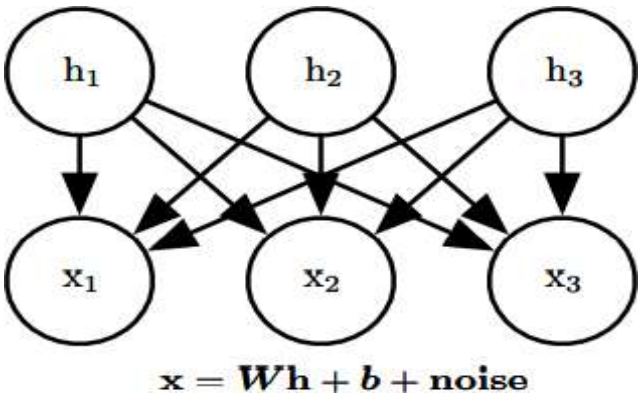


Figure:1 Linear Factor Model

3.2 Dimensionality Reduction:

- High dimensionality is challenging and redundant
- It is natural to try to reduce dimensionality
- We reduce the dimensionality by feature combination i.e., combine old features X to create new features Y as given below

$$x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{bmatrix} \rightarrow f\left(\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{bmatrix}\right) = \begin{bmatrix} y_1 \\ \vdots \\ y_k \end{bmatrix} = y \quad \text{with } k < d$$

Figure 2: Dimensionality Reduction

3.3 Principal Component Analysis (PCA):

Principal Component Analysis, or simply PCA, is a statistical procedure concerned with elucidating the covariance structure of a set of variables. In particular it allows us to identify the principal directions in which the data varies.

For example, in figure 1, suppose that the triangles represent a two variable data set which we have measured in the X-Y coordinate system. The principal direction in which the data varies is shown by the U axis and the second most important direction is the V axis orthogonal to it. If we place the U-V axis system at the mean of the data it gives us a compact representation. If we transform each (X, Y) coordinate into its corresponding (U, V) value, the data is de-correlated, meaning that the co-variance between the U and V variables is zero. For a given set of data, principal component analysis finds the axis system defined by the principal directions of variance (ie the U − V axis system in figure 3). The directions U and V are called the principal components

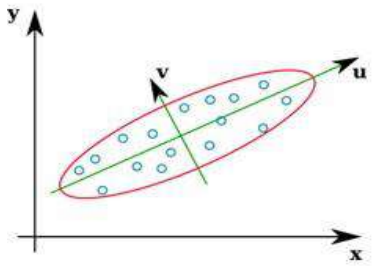


Figure 3A: PCA for Data Representation

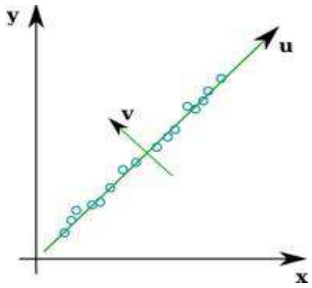


Figure 3B: PCA Dimension Reduction

If the variation in a data set is caused by some natural property, or is caused by random experimental error, then we may expect it to be normally distributed. In this case we show the nominal extent of the normal distribution by a hyper-ellipse (the two-dimensional ellipse in the example). The hyper ellipse encloses data points that are thought of as belonging to a class. It is drawn at a distance beyond which the probability of a point belonging to the class is low, and can be thought of as a class boundary.

If the variation in the data is caused by some other relationship, then PCA gives us a way of reducing the dimensionality of a data set. Consider two variables that are nearly related linearly as shown in figure 3B. As in figure 3A the principal direction in which the data varies is shown by the U axis, and the secondary direction by the V axis. However in this case all the V coordinates are all very close to zero. We may assume, for example, that they are only non zero because of experimental noise. Thus in the U V axis system we can represent the data set by one variable U and discard V . Thus we have reduced the dimensionality of the problem by 1Computing the Principal Components

3.3.1. Computing the Principal Components

In computational terms the principal components are found by calculating the eigenvectors and eigenvalues of the data covariance matrix. This process is equivalent to finding the axis system in which the co-variance matrix is diagonal. The eigenvector with the largest eigenvalue is the direction of greatest variation, the one with the second largest eigenvalue is the (orthogonal) direction with the next highest variation and so on. To see how the computation is done we will give a brief review on eigenvectors/eigenvalues.

Let A be a n × n matrix. The eigenvalues of A are defined as the roots of:

$$Determinant\ (\boldsymbol{A} - \lambda \boldsymbol{I}) = |(\boldsymbol{A} - \lambda \boldsymbol{I})| = 0$$

where I is the n n identity matrix. This equation is called the characteristic equation (or characteristic polynomial) and has n roots.
Let λ be an eigenvalue of A. Then there exists a vector x such that:

$$\boldsymbol{Ax} = \lambda \boldsymbol{x}$$

The vector x is called an eigenvector of A associated with the eigenvalue λ. Notice that there is no unique solution for x in the above equation. It is a direction vector only and can be scaled to any magnitude. To find a numerical solution for x we need to set one of its elements to an arbitrary value, say 1, which gives us a set of simultaneous equations to solve for the other elements. If there is no solution, we repeat the process with another element. Ordinarily we normalize the final values so that x has length one, that is $x \cdot x^T = 1$.
Suppose we have a 3 × 3 matrix A with eigenvectors x1, x2, x3, and eigenvalues λ1, λ2, λ3 so:

$$Ax1 = \lambda 1 x1 \qquad Ax2 = \lambda 2 x2 \qquad Ax3 = \lambda 3 x3$$

Putting the eigenvectors as the columns of a matrix gives:

$$\boldsymbol{A}\begin{bmatrix} \boldsymbol{x_1} & \boldsymbol{x_2} & \boldsymbol{x_3} \end{bmatrix} = \begin{bmatrix} \boldsymbol{x_1} & \boldsymbol{x_2} & \boldsymbol{x_3} \end{bmatrix}\begin{bmatrix} \lambda_1 & 0 & 0 \\ 0 & \lambda_2 & 0 \\ 0 & 0 & \lambda_3 \end{bmatrix}$$

writing:

$$\Phi = \begin{bmatrix} \boldsymbol{x_1} & \boldsymbol{x_2} & \boldsymbol{x_3} \end{bmatrix} \qquad \Lambda = \begin{bmatrix} \lambda_1 & 0 & 0 \\ 0 & \lambda_2 & 0 \\ 0 & 0 & \lambda_3 \end{bmatrix}$$

gives us the matrix equation:

$$\boldsymbol{A}\Phi = \Phi\Lambda$$

gives us the matrix equation: AΦ = ΦΛ We normalised the eigenvectors to unit magnitude, and they are orthogonal, so: ΦΦT = ΦT Φ = I, which means that: Φ T AΦ = Λ and: A = ΦΛΦT. Now let us consider how this applies to the covariance matrix in the PCA process. Let Σ be an n×n covariance matrix. There is an orthogonal n × n matrix Φ whose columns are eigenvectors of Σ and a diagonal matrix Λ whose diagonal elements are the eigenvalues of Σ, such that $\Phi^T \Sigma \Phi = \Lambda$ We can look on the matrix of eigenvectors Φ as a linear transformation which, in the example of figure 3A transforms data points in the [X, Y] axis system into the [U, V] axis system. In the general case the linear transformation given by Φ transforms the data points into a data set where the variables are uncorrelated. The correlation matrix of the data in the new coordinate system is Λ which has zeros in all the off-diagonal elements.

3.3.2 Steps involved in PCA:

- **Start with data for n observations on p variables**
- **Form a matrix of size n X p**
- **Calculate the Covariance Matrix**

- **Calculate the Eigen vectors and Eigen Values**
- **Choose Principal Component from Feature Vectors**
- **Derive the new Data Set**

3.3.3 PCA Advantages:

1. Removes Correlated Features:

   In a real-world scenario, it is very common that we get thousands of features in our dataset. You cannot run your algorithm on all the features as it will reduce the performance of your algorithm and it will not be easy to visualize that many features in any kind of graph. Hence the data set should be reduced. We need to find out the correlation among the features (correlated variables). Finding correlation manually in thousands of features is nearly impossible, frustrating and time-consuming. PCA performs this task effectively. After implementing the PCA on your dataset, all the Principal Components are independent of one another. There is no correlation among them.

2. Improves Algorithm Performance:

   With so many features, the performance of your algorithm will drastically degrade. PCA is a very common way to speed up your Machine Learning algorithm by getting rid of correlated variables which don't contribute in any decision making. The training time of the algorithms reduces significantly with a smaller number of features. So, if the input dimensions are too high, then using PCA to speed up the algorithm is a reasonable choice.

3. Reduces Overfitting:

   Overfitting mainly occurs when there are too many variables in the dataset. So, PCA helps in overcoming the overfitting issue by reducing the number of features.

4. Improves Visualization:

3.3.4. Disadvantages of PCA

1. **Independent variables become less interpretable:** After implementing PCA on the dataset, your original features will turn into Principal Components. Principal Components are the linear combination of your original features. Principal Components are not as readable and interpretable as original features.
2. **Data standardization is must before PCA:** You must standardize your data before implementing PCA, otherwise PCA will not be able to find the optimal Principal Components.
3. **Information Loss:** Although Principal Components try to cover maximum variance among the features in a dataset, if we don't select the number of Principal Components with care, it may miss some information as compared to the original list of features.

3.4 Linear Discrimination Analysis (LDA):

Linear Discriminant Analysis as its name suggests is a linear model for classification and dimensionality reduction. Most commonly used for feature extraction in pattern classification problems.

3.4.1 Need for LDA:

- Logistic Regression is perform well for binary classification but fails in the case of multiple classification problems with well-separated classes. While LDA handles these quite efficiently.
- LDA can also be used in data pre-processing to reduce the number of features just as PCA which reduces the computing cost significantly**.**

3.4.2. Limitations:

- Linear decision boundaries may not effectively separate non-linearly separable classes. More flexible boundaries are desired.
- In cases where the number of observations exceeds the number of features, LDA might not perform as desired. This is called Small Sample Size (SSS) problem. Regularization is required.

Linear Discriminant Analysis or **Normal Discriminant Analysis** or **Discriminant Function Analysis** is a dimensionality reduction technique that is commonly used for supervised classification problems. It is used for modeling differences in groups i.e. separating two or more classes. It is used to project the features in higher dimension space into a lower dimension space. For example, we have two classes and we need to separate them efficiently. Classes can have multiple features. Using only a single feature to classify them may result in some overlapping. So, we will keep on increasing the number of features for proper classification.

3.4.3 Steps involved in LDA:

There are the three key steps.

(i) Calculate the separability between different classes. This is also known as between-class variance and is defined as the distance between the mean of different classes.

(ii) Calculate the within-class variance. This is the distance between the mean and the sample of every class.

(iii) Construct the lower-dimensional space that maximizes Step1 (between-class variance) and minimizes Step 2(within-class variance).

3.4.4. Pros & Cons of LDA

Advantages of LDA:

1. Simple prototype classifier: Distance to the class mean is used, it's simple to interpret.
2. Decision boundary is linear: It's simple to implement and the classification is robust.
3. Dimension reduction: It provides informative low-dimensional view on the data, which is both useful for visualization and feature engineering.

Shortcomings of LDA:

1. Linear decision boundaries may not adequately separate the classes. Support for more general boundaries is desired.

2. In a high-dimensional setting, LDA uses too many parameters. A regularized version of LDA is desired.

3. Support for more complex prototype classification is desired.

3.5. Manifold Learnings:

- Manifold learning for dimensionality reduction has recently gained much attention to assist image processing tasks such as **segmentation, registration, tracking, recognition, and computational anatomy.**
- The drawbacks of PCA in handling dimensionality reduction problems for **non-linear weird** and **curved shaped surfaces** necessitated development of more advanced algorithms like **Manifold Learning**.
- There are different variant's of Manifold Learning that solves the problem of **reducing data dimensions** and feature-sets obtained from real world problems representing uneven weird surfaces by sub-optimal data representation.
- This kind of data representation **selectively chooses data points** from a low-dimensional manifold that is embedded in a high-dimensional space in an attempt to **generalize linear frameworks** like PCA.
- Manifolds give a look of flat and featureless space that behaves like Euclidean space. Manifold learning problems are **unsupervised** where it learns the **high-dimensional structure** of the data from the data itself, without the use of **predetermined classifications** and **loss of importance of information** regarding some characteristic of the original variables.
- The goal of the manifold-learning algorithms is to recover the o**riginal domain structure**, up to some **scaling** and rotation. The nonlinearity of these algorithms allows them to reveal the domain structure even when the manifold is not **linearly embedded**. It uses some **scaling** and **rotation** for this purpose.
- *Manifold learning algorithms are divided in to two categories:*
  - **Global methods:** Allows high-dimensional data to be mapped from high-dimensional to low-dimensional such that the global properties are preserved. Examples include **Multidimensional Scaling (MDS), Isomaps** covered in the following sections.
  - **Local methods:** Allows high-dimensional data to be mapped to low dimensional such that local properties are preserved. Examples are **Locally linear embedding (LLE), Laplacian eigenmap (LE), Local tangent space alignment (LSTA), Hessian Eigenmapping (HLLE)**
- **Three popular manifold learning algorithms:**
  - IsoMap (Isometric Mapping)

Isomap seeks a lower-dimensional representation that maintains 'geodesic distances' between the points. A geodesic distance is a generalization of distance for curved surfaces. Hence, instead of measuring distance in pure Euclidean distance with the Pythagorean theorem-derived distance formula, Isomap optimizes distances along a discovered manifold

- Locally Linear Embeddings

Locally Linear Embeddings use a variety of tangent linear patches (as demonstrated with the diagram above) to model a manifold. It can be thought of as performing a PCA on each of these neighborhoods locally, producing a linear hyperplane, then comparing the results globally to find the best nonlinear embedding. The goal of LLE is to 'unroll' or 'unpack' in distorted fashion the structure of the data, so often LLE will tend to have a high density in the center with extending rays

- t-SNE

t-SNE is one of the most popular choices for high-dimensional visualization, and stands for **t-distributed Stochastic Neighbor Embeddings**. The algorithm converts relationships in original space into t-distributions, or normal distributions with small sample sizes and relatively unknown standard deviations. This makes t-SNE very sensitive to the local structure, a common theme in manifold learning. It is considered to be the go-to visualization method because of many advantages it possesses.

3.6.Auto Encoders:

AutoEncoder is an **unsupervised Artificial Neural Network** that attempts to encode the data by compressing it into the lower dimensions (bottlenecklayer or code) and then decoding the datato reconstruct the original input.The bottleneck layer (or code) holds the compressed representation of theinputdata. In AutoEncoder the number of output units must be equal to the number ofinputunits since we're attempting to reconstruct theinput data.

AutoEncoders usually consist of an encoder and a decoder. The encoder encodes the provided data into a lower dimension which is the size of thebottleneck layer and the decoder decodes the compressed data into itsoriginalform.The number of neurons in the layers of the encoder will be decreasing as we move on with further layers, whereas the number of neurons in thelayers of thedecoderwillbeincreasingas we moveonwithfurtherlayers. Thereare three layersused in the encoder and decoder in the following example. The encoder contains 32, 16, and 7 units in each layer respectively and the decodercontains 7, 16, and 32 unitsineachlayer respectively. The code size/the number of neurons in bottle-neck must be less than the number of featuresin the data. Before feeding the data into the AutoEncoder the data must definitely be scaled between 0 and 1 using MinMaxScaler since we are going to use sigmoid

activation function in the output layer which outputs values between0 and 1.When we are using AutoEncoders for dimensionality reduction we'll beextracting the bottleneck layer and use it to reduce the dimensions. Thisprocess can be viewed as **feature extraction**.

The type of AutoEncoder that we're using is **Deep AutoEncoder**, where theencoder and the decoder are symmetrical. The Autoencoders don't necessarily have a symmetrical encoder and decoder but we can have the encoderanddecodernon-symmetricalaswell.

3.6.1. Types of AutoEncoders are,

- Deep Autoencoder
- Sparse Autoencoder
- Under complete Autoencoder
- Variational Autoencoder
- LSTM Autoencoder

3.6.2. Hyperparameters of an AutoEncoder

- ✶ Code size or the number of units in the bottleneck layer
- ✶ Input and output size, which is the number of features in the data
- ✶ Number of neurons or nodes per layer
- ✶ Number of layers in encoder and decoder.
- ✶ Activation function
- ✶ Optimization function

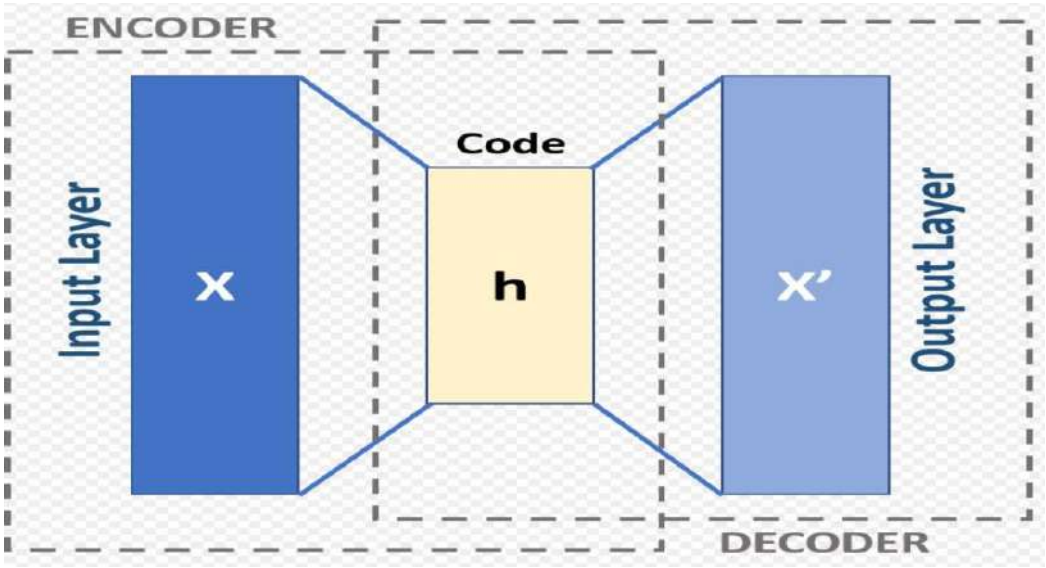


Figure 4: Auto Encoders

3.7. AlexNet:

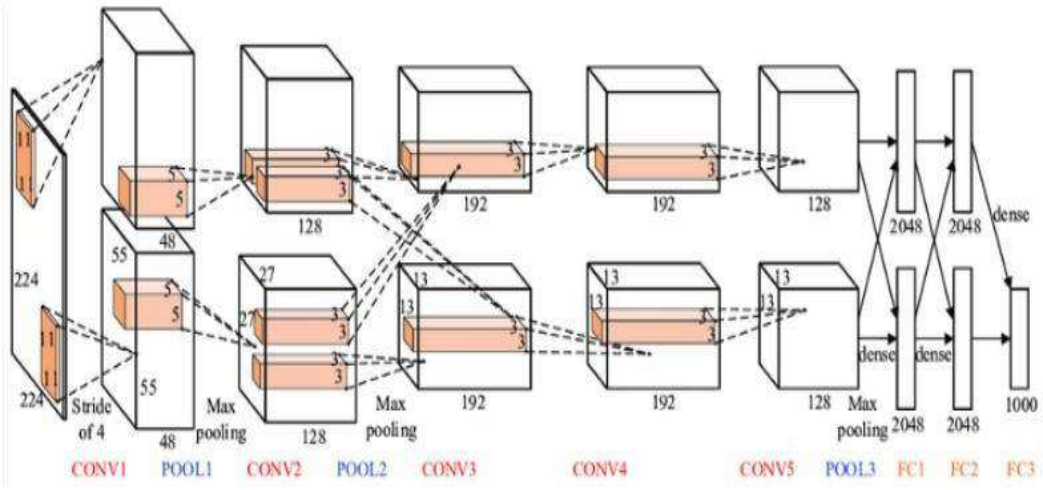


Figure 5: Alexnet Architecture

Alexnet model was proposed in 2012 in the research paper named Imagenet Classification with Deep Convolution Neural Network by Alex Krizhevsky and his colleagues

- The Alexnet has eight layers with learnable parameters
- The model has five layers with a combination of max pooling followed by 3 fully connected layers
- The fully connected layers use Relu activation except the output layer
- They found out that using the Relu as an activation function accelerated the speed of the training process by almost six times.
- They also used the dropout layers, which prevented the model from overfitting.
- The model is trained on the Imagenet dataset. The Imagenet dataset has arounf 14 million images across a 1000 classes.
- The input to this model is the images of size 227X227X3
- The first convolution layer with 96 filters of size 11X11 with stride 4
- The activation function used in this layer is relu. The output feature map is 55X55X96
- Next, we have the first Maxpooling layer, of size 3X3 and stride 2
- Next the filter size is reduced to 5X5 and 256 such filtersare added
  The stride value is 1 and padding 2. The activation function used is again relu. The output size we get is 27X27X256
- Next we have a max-pooling layer of size 3X3 with stride 2. The resulting feature map size is 13X13X256
- The third convolution operation with 384 filters of size 3X3 stride 1 and also padding 1is done next. In this stage the activation function used is relu. The output feature map is of shape 13X13X384

- Then the fourth convolution operation with 384 filters of size 3X3. The stride value along with the padding is 1.The output size remains unchanged as 13X13X384.
- After this, we have the final convolution layer of size 3X3 with 256 such filters. The stride and padding are set to 1,also the activation function is relu. The resulting feature map is of shape 13X13X256

  If we look at the architecture now, the number of filters is increasing as we are going deeper. Hence more features are extracted as we move deeper into the architecture. Also, the filter size is reducing, which means a decrease in the feature map shape.

3.8. VGG-16

- The major shortcoming of too many hyper-parameters of AlexNet was solved by VGG Net by replacing large kernel-sized filters (11 and 5 in the first and second convolution layer, respectively) with multiple 3×3 kernel-sized filters one after another.
- The architecture developed by Simonyan and Zisserman was the 1st runner up of the Visual Recognition Challenge of 2014.
- The architecture consist of 3*3 Convolutional filters, 2*2 Max Pooling layer with a stride of 1.
- Padding is kept same to preserve the dimension.
- There are 16 layers in the network where the input image is RGB format with dimension of 224*224*3, followed by 5 pairs of Convolution(filters: 64, 128, 256,512,512) and Max Pooling.
- The output of these layers is fed into three fully connected layers and a softmax function in the output layer.
- In total there are 138 Million parameters in VGG Net

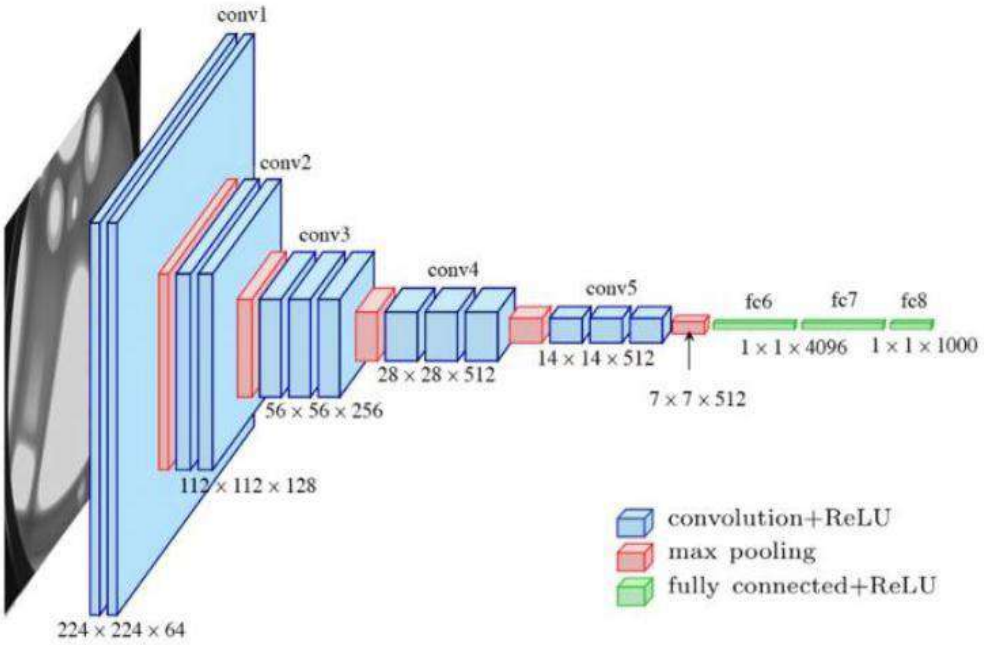


Figure6: VGG Architecture

3.9 ResNet:

ResNet, the winner of ILSVRC-2015 competition is a deep network with over 100 layers. Residual networks (ResNet) is similar to VGG nets however with a sequential approach they also use “Skip connections” and “batch normalization” that helps to train deep layers without hampering the performance. After VGG Nets, as CNNs were going deep, it was becoming hard to train them because of vanishing gradients problem that makes the derivate infinitely small. Therefore, the overall performance saturates or even degrades. The idea of skips connection came from highway network where gated shortcut connections were used

3.10 Inception Net:

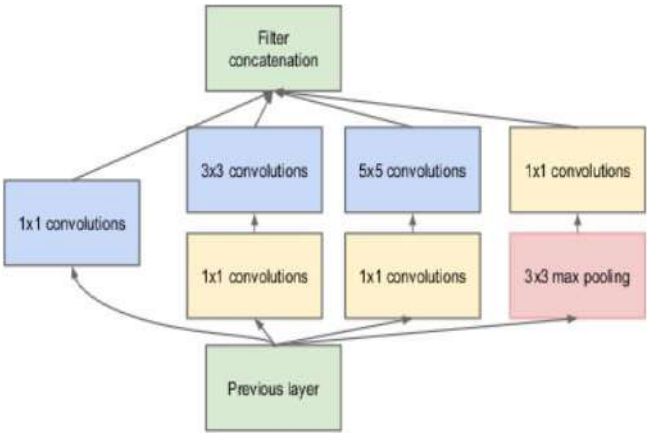


Figure 7: InceptionNet

Inception network also known as GoogleLe Net was proposed by developers at google in “Going Deeper with Convolutions” in 2014. The motivation of InceptionNet comes from the presence of sparse features Salient parts in the image that can have a large variation in size. Due to this, the selection of right kernel size becomes extremely difficult as big kernels are selected for global features and small kernels when the features are locally located. The InceptionNets resolves this by stacking multiple kernels at the same level. Typically it uses 5*5, 3*3 and 1*1 filters in one go.

3.11. Hyperparameter Optimization:

Hyperparameter optimization in machine learning intends to find the hyperparameters of a given machine learning algorithm that deliver the best performance as measured on a validation set. Hyperparameters, in contrast to model parameters, are set by the machine learning engineer before training. The number of trees in a random forest is a hyperparameter while the weights in a neural network are model parameters learned during training. Hyperparameter optimization finds a combination of hyperparameters that returns an optimal

model which reduces a predefined loss function and in turn increases the accuracy on given independent data

3.11.1 Hyperparameter Optimization methods

- ❑ Manual Hyperparameter Tuning
- ❑ Grid Search
- ❑ Random Search
- ❑ Bayesian Optimization
- ❑ Gradient-based Optimization

Reference Books:

1. B. Yegnanarayana, "Artificial Neural Networks" Prentice Hall Publications.
2. Simon Haykin, "Artificial Neural Networks", Second Edition, Pearson Education.
3. Laurene Fausett, "Fundamentals of Neural Networks, Architectures, Algorithms and Applications", Prentice Hall publications.
4. Cosma Rohilla Shalizi, Advanced Data Analysis from an Elementary Point of View, 2015.
5. 2. Deng & Yu, Deep Learning: Methods and Applications, Now Publishers, 2013.
6. 3. Ian Goodfellow, Yoshua Bengio, Aaron Courville, Deep Learning, MIT Press, 2016.
7. 4. Michael Nielsen, Neural Networks and Deep Learning, Determination Press, 2015.

Note: For further reference, kindly refer the class notes, PPTs, Video lectures available in the Learning Management System (Moodle)

********** ALL THE BEST **********

UNIT IV DIMENTIONALITY REDUCTION

Optimization in deep learning– Non-convex optimization for deep networks- Stochastic Optimization Generalization in neural networks- Spatial Transformer Networks- Recurrent networks, LSTM Recurrent Neural Network Language Models- Word-Level RNNs & Deep Reinforcement Learning - Computational & Artificial Neuroscience.

4.1 Optimization in Deep Learning:

In Deep Learning, with the help of loss function, the performance of the model is estimated/ evaluated. This loss is used to train the network so that it performs better. Essentially, we try to minimize the Loss function. Lower Loss means the model performs better. The Process of minimizing any mathematical function is called Optimization.

Optimizers are algorithms or methods used to change the features of the neural network such as weights and learning rate so that the loss is reduced. Optimizers are used to solve optimization problems by minimizing the function

The Goal of an Optimizer is to minimize the Objective Function(Loss Function based on the Training Data set). Simply Optimization is to minimize the Training Error.

4.1.1 Need for Optimization:

- Prescence of Local Minima reduces the model performance
- Prescence of Saddle Points which creates Vanishing Gradients or Exploding Gradient Issues
- To select appropriate weight values and other associated model parameters
- To minimize the loss value (Training error)

4.2. Convex Optimization:

Convex optimization is a kind of optimization which deals with the study of problem of minimizing convex functions. Here the optimization function is convex function.

All Linear functions are convex, so linear programming problems are convex problems. When we have a convex objective and a convex feasible region, then there can be only one optimal solution, which is globally optimal.

Definition: A set $C \subseteq Rn$ is convex if for $x, y \in C$ and any $\alpha \in [0, 1]$,

$$\alpha x + (1 - \alpha)y \in C$$

Convexity plays a vital role in the design of optimization algorithms. This is largely due to the fact that it is much easier to analyze and test algorithms in such a context.

Consider the given Figure 4.1 given below, select any to points in the region and join them by a straight Line. If the line and the selected points all lie inside the region then we call that region as Convex Region (as Shown in the diagram Figure 4.1)

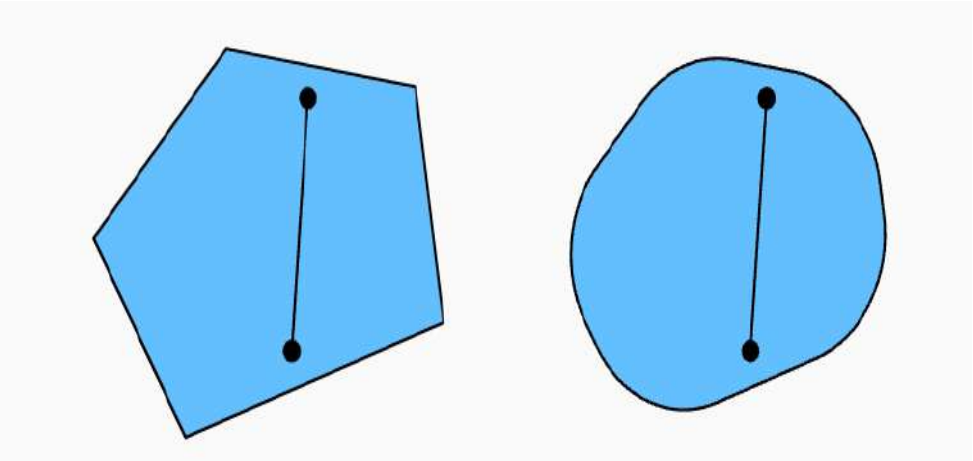


Figure 4.1: Convex Regions

A convex optimization problem is of the form:

$$\min_{x \in D} f(x)$$

subject to

$$g_i(x) \leq 0, \ i = 1, ..., m$$

$$h_j(x) = 0, \ j = 1, ..., r$$

where f and g_i are all convex , and h_j are affine. Any local minimizer of a convex optimization problem is a global minimizer.

4.3. Non-Convex Optimization:

- The Objective function is a non- convex function
- All non-linear problems can be modelled by using non convex functions. (Linear functions are convex)
- It has Multiple feasible regions and multiple locally optimal points.
- There can't be a general algorithm to solve it efficiently in all cases
- Neural networks are universal function approximators, to do this, they need to be able to approximate non-convex functions.
  Refer the figure 4.2 .It shows Non Convex Region

4.3.1. How to solve non-convex problems?

- ✶ Stochastic gradient descent
- ✶ Mini-batching
- ✶ SVRG
- ✶ Momentum

4.3.2. Reasons For Non-Convexity:

- Presence of many Local Minima
- Prescence of Saddle Points
- Very Flat Regions
- Varying Curvature

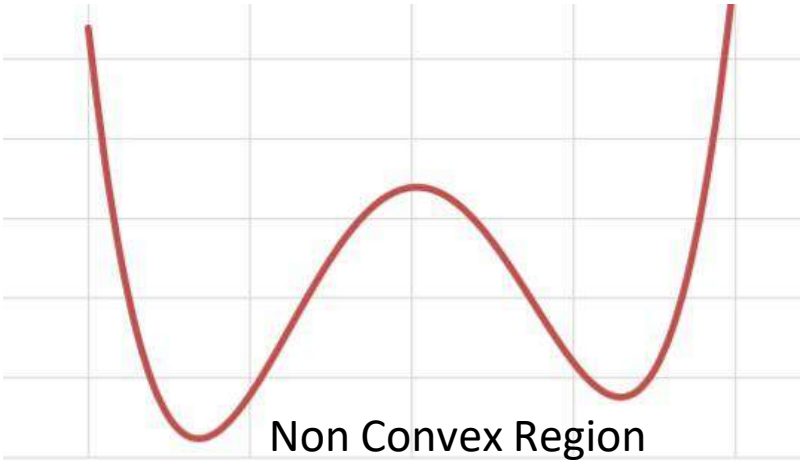


Figure 4.2: Convex Regions

4.4. Spatial Transform Network [STN]:

Spatial Transformer Network (SSTN) helps to crop out and scale-normalizes the appropriate region, which can simplify the subsequent classification task and lead to better classification performance. The Spatial Transformer Network contains three parts Namely, Localization, Grid Generator and Sampler. These Networks are used for performing Transformations such as Cropping, Rotation etc on the given input images.

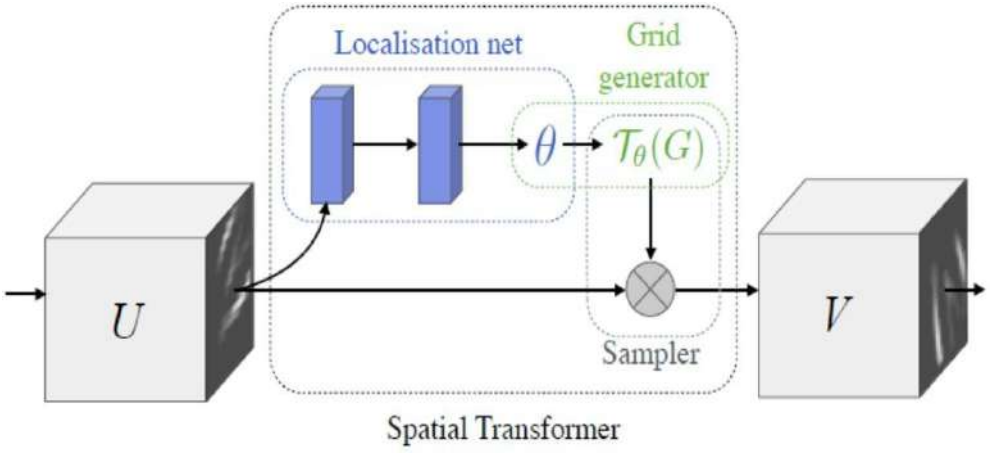


Figure 4.3: Convex Regions

Localisation Net:

With **input feature map *U***, with width *W*, height *H* and *C* channels, **outputs are *θ***, the parameters of transformation *Tθ*. It can be learnt as affine transform

Grid Generator:

Suppose we have a regular grid G, this G is a set of points with **target coordinates (xt_i, yt_i)**. Then we **apply transformation *T θ* on G**, i.e. *T θ*(*G*). After *Tθ*(*G*), a set of points with **destination coordinates (xt_i, yt_i) is outputted**. These points have been altered based on the transformation parameters. It can be Translation, Scale, Rotation or More Generic Warping depending on how we set *θ* as mentioned above.

Sampler:

Based on the new set of coordinates (xt_i, yt_i), we generate a transformed output feature map *V*. This *V* is translated, scaled, rotated, warped, projective transformed or affined, whatever. It is noted that STN can be applied to not only input image, but also intermediate feature maps.

- **STN is a mechanism that rotates or scales an input image or a feature map in order to focus on the target object and to remove rotational variance** .

- One of the most notable features of STNs is their modularity (the module can be injected into any part of the model) and their ability to be trained with a single backprop algorithm without modification of the initial model.

4.4.1. Advantages:

- Helps in learning explicit spatial transformations like translation, rotation, scaling, cropping, non-rigid deformations, etc. of features.
- Can be used in any networks and at any layer and learnt in an end-to-end trainable manner.
- Provides improvement in the performance of existing models.

4.5. Recurrent Neural Networks:

- RNNs are very powerful, because they combine two properties:
  - Distributed hidden state that allows them to store a lot of information about the past efficiently.
  - Non-linear dynamics that allows them to update their hidden state in complicated ways.
- With enough neurons and time, RNNs can compute anything that can be computed by your computer.

4.5.1. Need for RNN:

- Normal Networks cannot handle sequential data

- They considers only the current input
- Normal Neural networks cannot memorize previous inputs

The solution to these issues is the RNN

RNN works on the principle of saving the output of a particular layer and feeding this back to the input in order to predict the output of the layer. We can convert a Feed-Forward Neural Network into a Recurrent Neural Network as given below in figure 4.4.

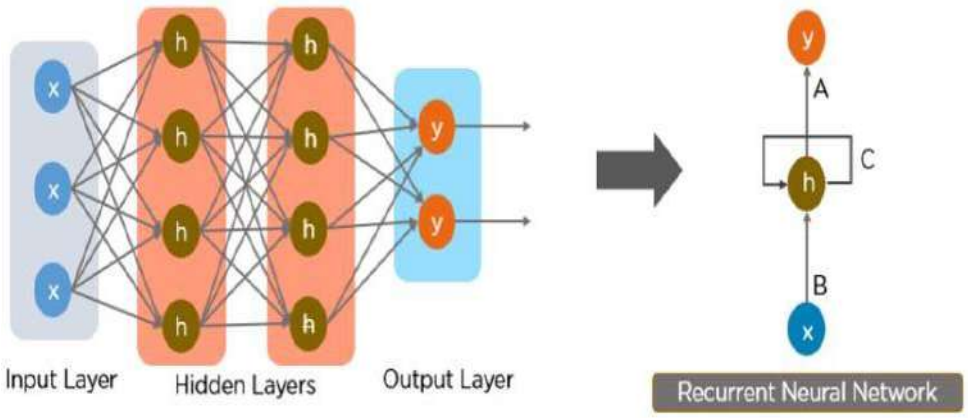


Figure 4.4: Converting a Full network into Recurrent Network

The nodes in different layers of the neural network are compressed to form a single layer of recurrent neural networks. A, B, and C are the parameters of the network. Here, "x" is the input layer, "h" is the hidden layer, and "y" is the output layer. A, B, and C are the network parameters used to improve the output of the model. At any given time t, the current input is a combination of input at x(t) and x(t-1). The output at any given time is fetched back to the network to improve on the output.(Refer Figures 5A and 5 B)

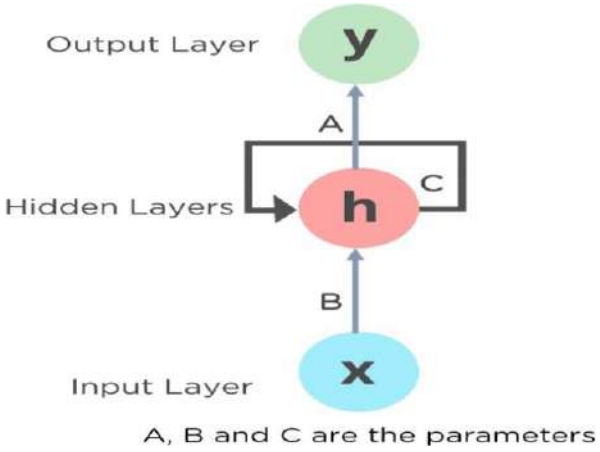


Figure 4.5 A: Recurrent Network

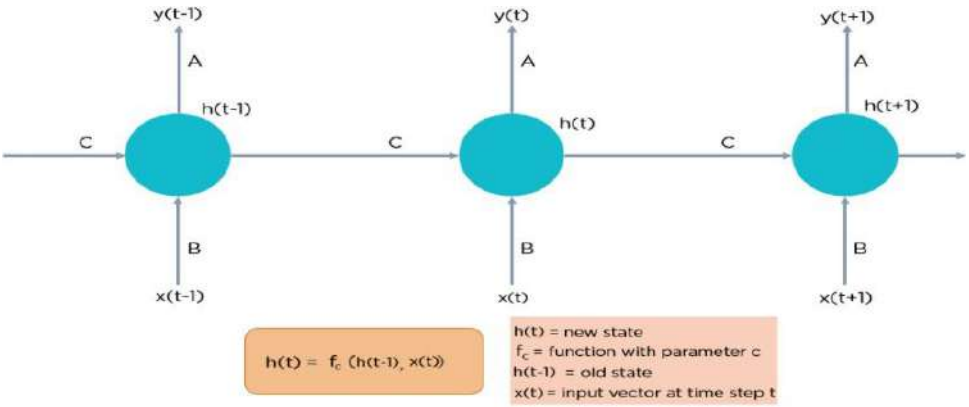


Fig: Fully connected Recurrent Neural Network

Figure 4.5 B: Fully Connected RNN

4.5.2. Providing Input to RNN:

We can specify inputs in several ways:

– Specify the initial states of all the units.

– Specify the initial states of a subset of the units.

– Specify the states of the same subset of the units at every time step.

4.5.3. providing Targets to RNN:

We can specify targets in several ways:

– Specify desired final activities of all the units

– Specify desired activities of all units for the last few steps

• Good for learning attractors

• It is easy to add in extra error derivatives as we backpropagate.

– Specify the desired activity of a subset of the units

4.6. Long Short Term Memory Network's (LSTM):

LSTMs are a special kind of RNN — capable of learning long-term dependencies by remembering information for long periods is the default behavior. All RNN are in the form of a chain of repeating modules of a neural network. In standard RNNs, this repeating module will have a very simple structure, such as a single tanh layer.

LSTMs also have a chain-like structure, but the repeating module is a bit different structure. Instead of having a single neural network layer, four interacting layers are communicating extraordinarily.

Hochreiter & Schmidhuber (1997) solved the problem of getting an RNN to remember things for a long time (like hundreds of time steps). They designed a memory cell using

logistic and linear units with multiplicative interactions. Information gets into the cell whenever its "write" gate is on. The information stays in the cell so long as its "keep" gate is on. Information can be read from the cell by turning on its "read" gate.(Refer Figure 4.6 – shown Below)

To preserve information for a long time in the activities of an RNN, we use a circuit that implements an analog memory cell.

– A linear unit that has a self-link with a weight of 1 will maintain its state.

– Information is stored in the cell by activating its write gate.

– Information is retrieved by activating the read gate.

– We can backpropagate through this circuit because logistics are had nice derivatives.

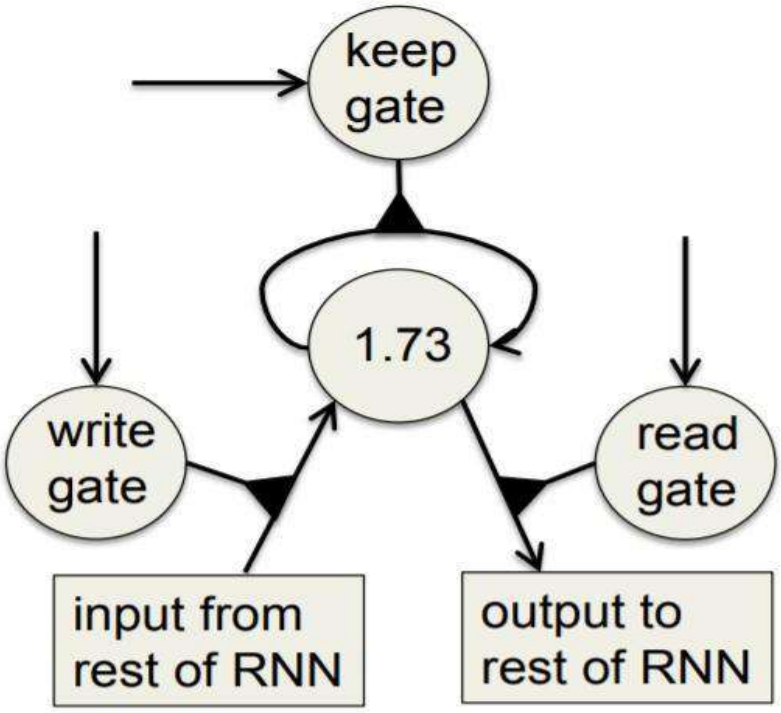


Figure 4.6 B: Read, Keep, Write gate of an LSTM

4.6.1. Steps Involved in LSTM Networks:

Step 1: Decide how much past data it should remember

The first step in the LSTM is to decide which information should be omitted from the cell in that particular time step. The sigmoid function determines this. It looks at the previous state (ht-1) along with the current input xt and computes the function.

Step 2: Decide how much this unit adds to the current state

In the second layer, there are two parts. One is the sigmoid function, and the other is the tanh function. In the sigmoid function, it decides which values to let through (0 or 1). tanh function gives weightage to the values which are passed, deciding their level of importance (-1 to 1).

Step 3: Decide what part of the current cell state makes it to the output

The third step is to decide what the output will be. First, we run a sigmoid layer, which decides what parts of the cell state make it to the output. Then, we put the cell state through tanh to push the values to be between -1 and 1 and multiply it by the output of the sigmoid gate.

4.6.2. Applications of LSTM include:

- Robot control
- Time series prediction
- Speech recognition
- Rhythm learning
- Music composition
- Grammar learning
- Handwriting recognition

4.7. Computational and Artificial Neuro-Science:

Computational neuroscience is the field of study in which mathematical tools and theories are used to investigate brain function.

The term “computational neuroscience” has two different definitions:

1. using a computer to study the brain
2. studying the brain as a computer

Computational and Artificial Neuroscience deals with the study or understanding of how signals are transmitted through and from the human brain. A better understanding of How decision is made in human brain by processing the data or signals will help us in developing Intelligent algorithms or programs to solve complex problems. Hence, we need to understand the basics of Biological Neural Networks (BNN).

4.7.1. The Biological Neurons:

The human brain consists of a large number, more than a billion of neural cells that process information. Each cell works like a simple processor. The massive interaction between all cells and their parallel processing only makes the brain’s abilities possible. Figure 1 represents a human biological nervous unit. Various parts of biological neural network(BNN) is marked in Figure 4.7.

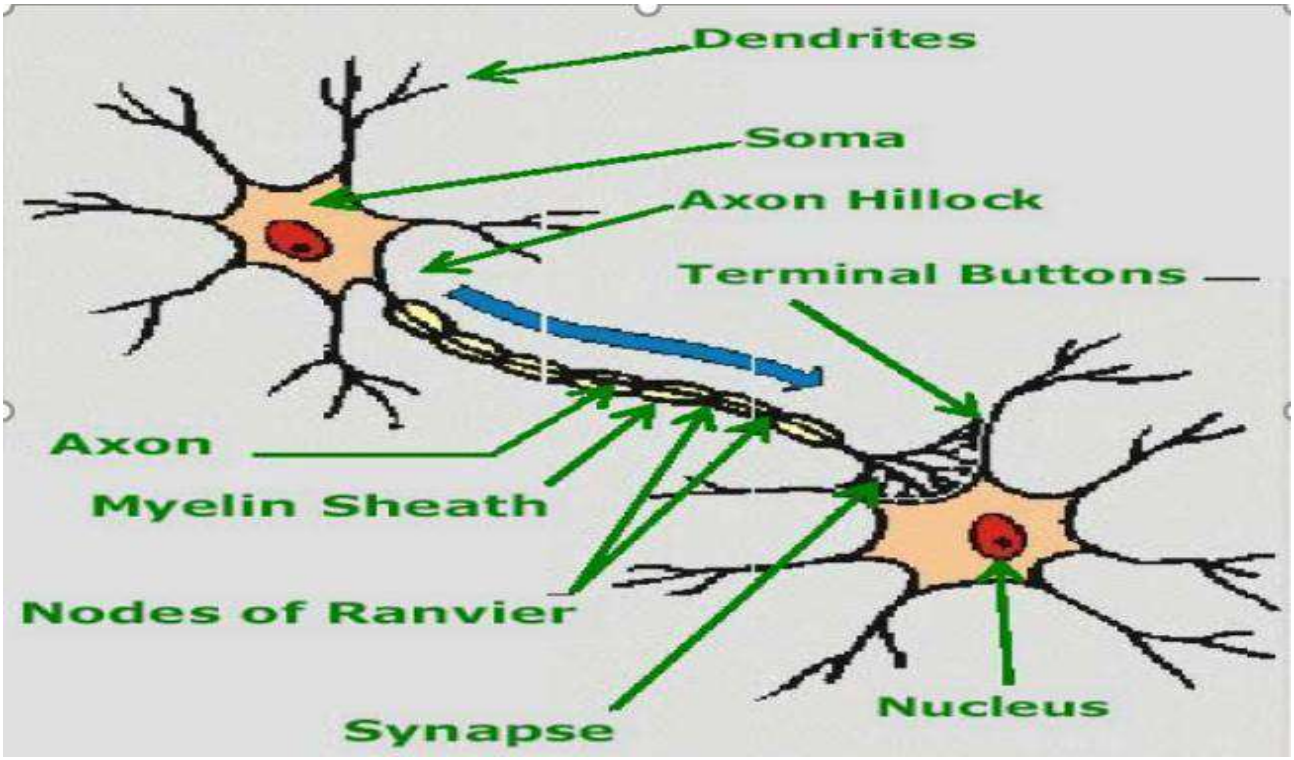


Figure 4.7: Biological Neural Network

Dendrites are branching fibres that extend from the cell body or soma.

Soma or cell body of a neuron contains the nucleus and other structures, support chemical processing and production of neurotransmitters.

Axon is a singular fiber carries information away from the soma to the synaptic sites of other neurons (dendrites ans somas), muscels, or glands.

Axon hillock is the site of summation for incoming information. At any moment, the collective influence of all neurons that conduct impulses to a given neuron will determine whether or n ot an action potential will be initiated at the axon hillock and propagated along the axon.

Myelin sheath consists of fat-containing cells that insulate the axon from electrical activity. This insulation acts to increase the rate of transmission of signals. A gap exists between each myelin sheath cell along the axon. Since fat inhibits the propagation of electricity, the signals jump from one gap to the next.

Nodes of Ranvier are the gaps (about 1 μm) between myelin sheath cells. Since fat serves as a good insulator, the myelin sheaths speed the rate of transmission of an electrical impulse along the axon.

Synapse is the point of connection between two neurons or a neuron and a muscle or a gland. Electrochemical communication between neurons take place at these junctions.

Terminal buttons of a neuron are the small knobs at the end of an axon that release chemicals called neurotransmitters.

Information flow in a neural cell

The input/output and the propagation of information are shown below.

4.7.2. Artificial neuron model

An artificial neuron is a mathematical function conceived as a simple model of a real (biological) neuron.

- The McCulloch-Pitts Neuron
  This is a simplified model of real neurons, known as a Threshold Logic Unit.
- A set of input connections brings in activations from other neuron.
- A processing unit sums the inputs, and then applies a non-linear activation function (i.e. squashing/transfer/threshold function).
- An output line transmits the result to other neurons.

4.7.3. Basic Elements of ANN:

Neuron consists of three basic components –weights, thresholds and a single activation function. An Artificial neural network(ANN) model based on the biological neural sytems is shown in figure 4.8.

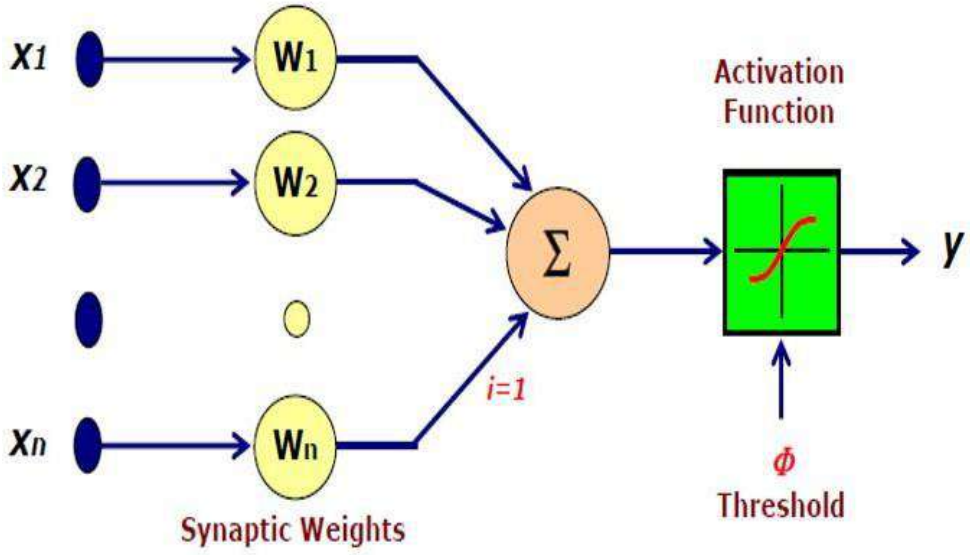


Figure 4.8: Basic Elements of Artificial Neural Network

The goal of computational neuroscience is to explain how electrical and chemical signals are used in the brain to represent and process information**.** It explains the biophysical mechanisms of computation in neurons, computer simulations of neural circuits, and models of learning.

4.7.4. Applications of Computational Neuro Science:

- Deep Learning, Artificial Intelligence and Machine Learning
- Human psychology
- Medical sciences
- Mental models
- Computational anatomy
- Information theory

Reference Books:

1. B. Yegnanarayana, "Artificial Neural Networks" Prentice Hall Publications.
2. Simon Haykin, "Artificial Neural Networks", Second Edition, Pearson Education.
3. Laurene Fausett, "Fundamentals of Neural Networks, Architectures, Algorithms and Applications", Prentice Hall publications.
4. Cosma Rohilla Shalizi, Advanced Data Analysis from an Elementary Point of View, 2015.
5. 2. Deng & Yu, Deep Learning: Methods and Applications, Now Publishers, 2013.
6. 3. Ian Goodfellow, Yoshua Bengio, Aaron Courville, Deep Learning, MIT Press, 2016.
7. 4. Michael Nielsen, Neural Networks and Deep Learning, Determination Press, 2015.

Note: For further reference, kindly refer the class notes, PPTs, Video lectures available in the Learning Management System (Moodle)

********** ALL THE BEST **********

UNIT V APPLICATIONS OF DEEP LEARNING

Imagenet- Detection-Audio WaveNet-Natural Language Processing Word2Vec - Joint Detection BioInformatics- Face Recognition- Scene Understanding- Gathering Image Captions

5.1. Imagenet:

ImageNet is an image database organized according to the WordNet hierarchy (currently only the nouns), in which each node of the hierarchy is depicted by hundreds and thousands of images. In Machine Learning and Deep Neural Networks, machines are trained on a large dataset of various images. Machines are required to learn useful features from these training images. Once learned, they can use these features to classify images and perform many other tasks associated with computer vision. ImageNet gives researchers a common set of images to benchmark their models and algorithms.

ImageNet is useful for many computer vision applications such as object recognition, image classification and object localization.Prior to ImageNet, a researcher wrote one algorithm to identify dogs, another to identify cats, and so on. After training with ImageNet, the same algorithm could be used to identify different objects. The diversity and size of ImageNet meant that a computer looked at and learned from many variations of the same object. These variations could include camera angles, lighting conditions, and so on. Models built from such extensive training were better at many computers vision tasks. ImageNet convinced researchers those large datasets were important for algorithms and models to work well.

5.1.1. Technical details of Image Net:

ImageNet consists of 14,197,122 images organized into 21,841 subcategories. These subcategories can be considered as sub-trees of 27 high-level categories. Thus, ImageNet is a well-organized hierarchy that makes it useful for supervised machine learning tasks. As many as 1,034,908 images have been annotated with **bounding boxes**. For example, if an image contains a cat as its main subject, the coordinates of a rectangle that bounds the cat are also published on ImageNet. This makes it useful for computer vision tasks such as object localization and detection. Then there's Scale-Invariant Feature Transform (SIFT) used in computer vision. SIFT helps in detecting local features in an image. ImageNet gives researchers 1000 subcategories with SIFT features covering about 1.2 million images. Images vary in resolution but it's common practice to train deep learning models on sub-sampled images of 256x256 pixels.

ImageNet did not define these subcategories on its own but derived these from WordNet. **WordNet** is a database of English words linked together by semantic relationships. Words of similar meaning are grouped together into a synonym set, simply called **synset**. Hypernyms are synsets that are more general. Thus, "organism" is a hypernym of "plant". Hyponyms are synsets that are more specific. Thus, "aquatic" is a hyponym of "plant". This hierarchy makes it useful for computer vision tasks. If the model is not sure about a subcategory,

it can simply classify the image higher up the hierarchy where the error probability is less. For example, if model is unsure that it's looking at a rabbit, it can simply classify it as a mammal.

While WordNet has 100K+ synsets, only the nouns have been considered by ImageNet.

5.1.2. How the images are labelled in ImageNet?

In the early stages of the ImageNet project, a quick calculation showed that by employing a few people, they would need 19 years to label the images collected for ImageNet. But in the summer of 2008, researchers came to know about an Amazon service called Mechanical Turk. This meant that image labelling can be crowdsourced via this service. Humans all over the world would label the images for a small fee.

Humans make mistakes and therefore we must have checks in place to overcome them. Each human is given a task of 100 images. In each task, 6 "gold standard" images are placed with known labels. At most 2 errors are allowed on these standard images, otherwise the task has to be restarted.

In addition, the same image is labelled by three different humans. When there's disagreement, such ambiguous images are resubmitted to another human with tighter quality threshold (only one allowed error on the standard images).

5.1.3. How the images of ImageNet Licensed?

Images for ImageNet were collected from various online sources. ImageNet doesn't own the copyright for any of the images. This has implication on how ImageNet shares the images to researchers.

For public access, ImageNet provides image thumbnails and URLs from where the original images were downloaded. Researchers can use these URLs to download the original images. However, those who wish to use the images for non-commercial or educational purpose, can create an account on ImageNet and request access. This will allow direct download of images from ImageNet. This is useful when the original sources of images are no longer available.

The dataset can be explored via a browser-based user interface. Alternatively, there's also an API. Researchers may want to read the API Documentation. This documentation also shares how to download image features and bounding boxes.

5.1.4. Shortcomings of ImageNet:

Images are not uniformly distributed across subcategories. One research team found that by considering 200 subcategories, they found that the top 11 had 50% of the images, followed by a long tail.

When classifying people, ImageNet uses labels that are racist, misogynist and offensive. People are treated as objects. Their photos have been used without their knowledge. About 5.8% labels are wrong. ImageNet lacks geodiversity. Most of the data represents North America and Europe. China and India are represented in only 1% and 2.1% of the images respectively. This implies that models trained on ImageNet will not work well when applied for the developing world.

Another study from 2016 found that 30% of ImageNet's image URLs are broken. This is about 4.4 million annotations lost. Copyright laws prevent caching and redistribution of these images by ImageNet itself

Content Source: https://devopedia.org/imagenet

5.2. WaveNet:

WaveNet is a deep generative model of raw audio waveforms. We show that WaveNets are able to generate speech which mimics any human voice and which sounds more natural than the best existing Text-to-Speech systems, reducing the gap with human performance by over 50%. Allowing people to converse with machines is a long-standing dream of human-computer interaction. The ability of computers to understand natural speech has been revolutionised in the last few years by the application of deep neural networks. However, generating speech with computers — a process usually referred to as speech synthesis or text-to-speech (TTS) — is still largely based on so-called concatenative TTS, where a very large database of short speech fragments are recorded from a single speaker and then recombined to form complete utterances. This makes it difficult to modify the voice (for example switching to a different speaker, or altering the emphasis or emotion of their speech) without recording a whole new database.

This has led to a great demand for parametric TTS, where all the information required to generate the data is stored in the parameters of the model, and the contents and characteristics of the speech can be controlled via the inputs to the model. So far, however, parametric TTS has tended to sound less natural than concatenative. Existing parametric models typically generate audio signals by passing their outputs through signal processing algorithms known as vocoders. WaveNet changes this paradigm by directly modelling the raw waveform of the audio signal, one sample at a time. As well as yielding more natural-sounding speech, using raw waveforms means that WaveNet can model any kind of audio, including music.

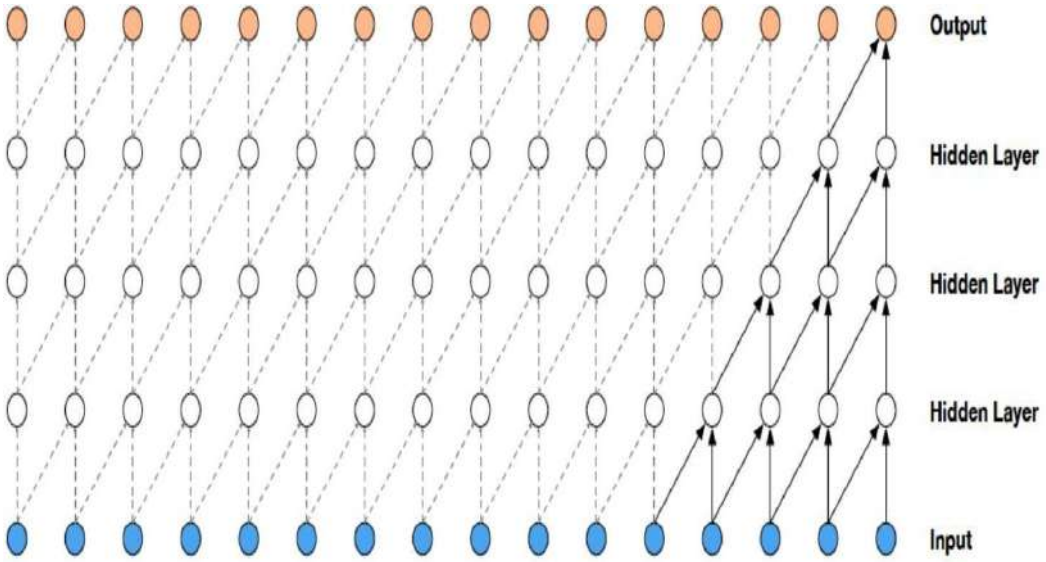


Figure 5.1: WaveNet Structure

The WaveNet proposes an autoregressive learning with the help of convolutional networks with some tricks. Basically, we have a convolution window sliding on the audio data, and at each step try to predict the next sample value that it did not see yet. In other words, it builds a network that learns the causal relationships between consecutive timesteps (as shown in figure 5.1)

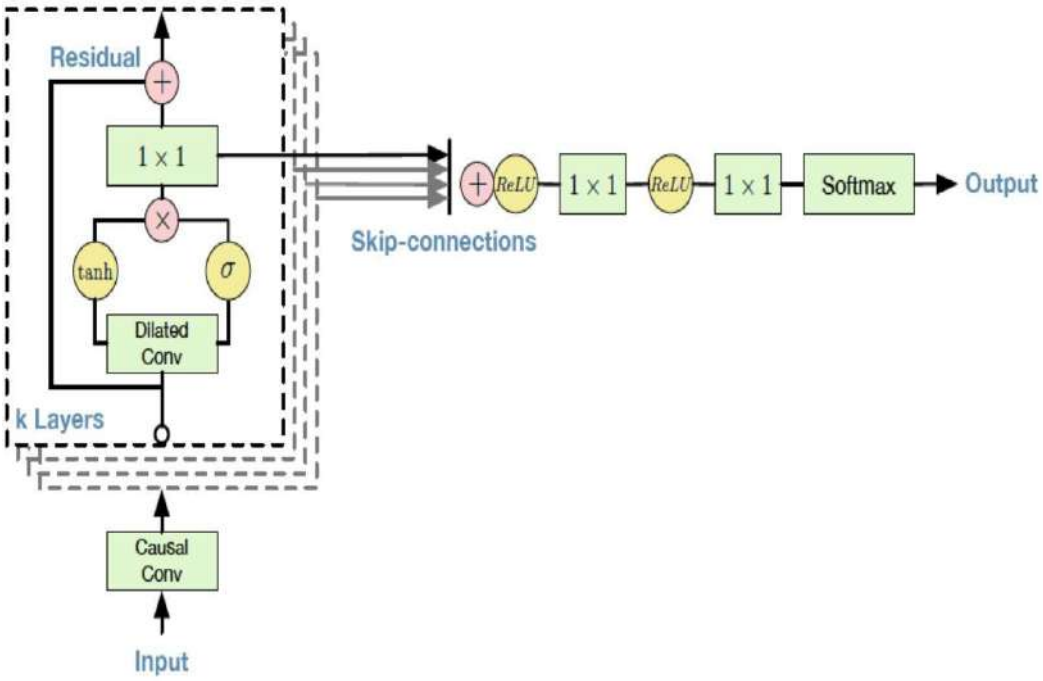


Figure 5.2: WaveNet Overall Model

Typically, the speech audio has a sampling rate of 22K or 16K. For few seconds of speech, it means there are more than 100K values for a single data and it is enormous for the network to consume. Hence, we need to restrict the size, preferably to around 8K. At the end, the values are predicted in Q channels (eg. Q=256 or 65536), which is compared to the original audio data compressed to Q distinct values. For that, the **mulaw quantization** could be used: it maps the values to the range of [0,Q]. And the loss can be computed either by cross-entropy, or discretized logistic mixture.

5.2.1. The Workflow of WaveNet:

- Input is fed into a causal 1D convolution
- The output is then fed to 2 different dilated 1D convolution layers with sigmoid and tanh activations
- The element-wise multiplication of 2 different activation values results in a skip connection

- And the element-wise addition of a skip connection and output of causal 1D results in the residual

Content Source: (1) https://www.analyticsvidhya.com/blog/2020/01/how-to-perform-automatic-music-generation/
(2) https://medium.com/@evinpinar/wavenet-implementation-and-experiments-2d2ee57105d5
(3) https://deepmind.com/blog/article/wavenet-generative-model-raw-audio

5.3. Natural Language Processing [NLP]:

Language is a method of communication with the help of which we can speak, read and write. For example, we think, we make decisions, plans and more in natural language; precisely, in words. However, the big question that confronts us in this AI era is that can we communicate in a similar manner with computers. In this sense, we can say that Natural Language Processing (NLP) is the sub-field of Computer Science especially Artificial Intelligence (AI) that is concerned about enabling computers to understand and process human language. Technically, the main task of NLP would be to program computers for analysing and processing huge amount of natural language data.

5.3.1. Natural Language Processing Phases:

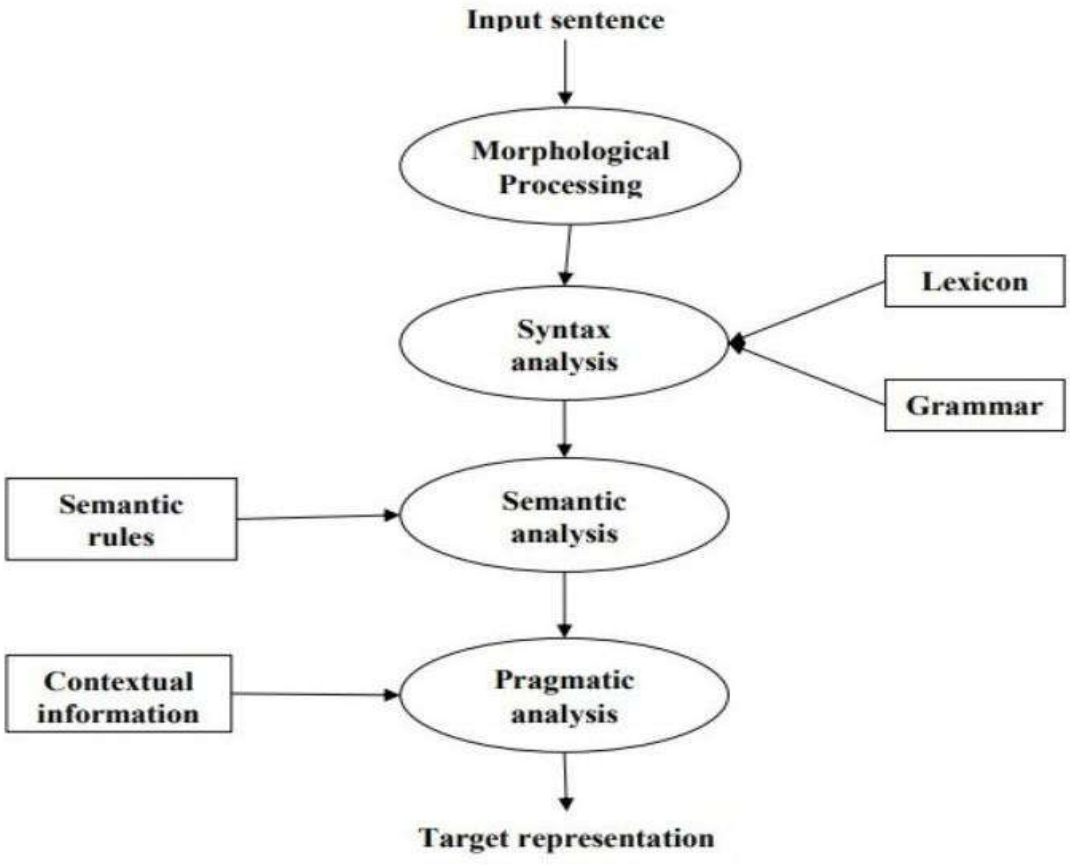


Figure 5.3: NLP Phases

The above diagram (Figure5.3) shows the phases or logical steps involved in natural language processing

5.3.1.1 Morphological Processing

It is the first phase of NLP. The purpose of this phase is to break chunks of language input into sets of tokens corresponding to paragraphs, sentences and words. For example, a word like **"uneasy"** can be broken into two sub-word tokens as **"un-easy"**.

5.3.1.2 Syntax Analysis

It is the second phase of NLP. The purpose of this phase is two folds: to check that a sentence is well formed or not and to break it up into a structure that shows the syntactic relationships between the different words. For example, the sentence like **"The school goes to the boy"** would be rejected by syntax analyser or parser.

5.3.1.3 Semantic Analysis

It is the third phase of NLP. The purpose of this phase is to draw exact meaning, or you can say dictionary meaning from the text. The text is checked for meaningfulness. For example, semantic analyser would reject a sentence like "Hot ice-cream".

5.3.1.4 Pragmatic Analysis

It is the fourth phase of NLP. Pragmatic analysis simply fits the actual objects/events, which exist in a given context with object references obtained during the last phase (semantic analysis). For example, the sentence "Put the banana in the basket on the shelf" can have two semantic interpretations and pragmatic analyser will choose between these two possibilities.

Content Source: Https:// www.tutorialspoint.com/ natural_language_processing/ natural_ language_processing_quick_guide.htm

5.3.1.5. Different types based on Working:

1. Speech Recognition — The translation of spoken language into text.
2. Natural Language Understanding (NLU) — The computer's ability to understand what we say.
3. Natural Language Generation (NLG) — The generation of natural language by a computer.

5.3.1.6. Applications of NLP:

- ✓ Spam Filters
- ✓ Algorithmic Trading
- ✓ Answering Questions
- ✓ Summarizing Information's etc

Content Source: https://www.geeksforgeeks.org/natural-language-processing-overview/

5.4. Word2Vec:

Word embedding is one of the most popular representation of document vocabulary. It is capable of capturing context of a word in a document, semantic and syntactic similarity, relation with other words, etc. What are word embeddings exactly? Loosely speaking, they are vector

representations of a particular word. Having said this, what follows is how do we generate them? More importantly, how do they capture the context? Word2Vec is one of the most popular technique to learn word embeddings using shallow neural network. It was developed by Tomas Mikolov in 2013 at Google.

The purpose and usefulness of Word2vec is to group the vectors of similar words together in vector space. That is, it detects similarities mathematically. Word2vec creates vectors that are distributed numerical representations of word features, features such as the context of individual words. It does so without human intervention.

Given enough data, usage and contexts, Word2vec can make highly accurate guesses about a word's meaning based on past appearances. Those guesses can be used to establish a word's association with other words (e.g. "man" is to "boy" what "woman" is to "girl"), or cluster documents and classify them by topic. Those clusters can form the basis of search, sentiment analysis and recommendations in such diverse fields as scientific research, legal discovery, e-commerce and customer relationship management. Measuring cosine similarity, no similarity is expressed as a 90 degree angle, while total similarity of 1 is a 0 degree angle, complete overlap.

Word2vec is a two-layer neural net that processes text by "vectorizing" words. Its input is a text corpus and its output is a set of vectors: feature vectors that represent words in that corpus. While Word2vec is not a deep neural network, it turns text into a numerical form that deep neural networks can understand.

Word2vec's applications extend beyond parsing sentences in the wild. It can be applied just as well to genes, code, likes, playlists, social media graphs and other verbal or symbolic series in which patterns may be discerned.

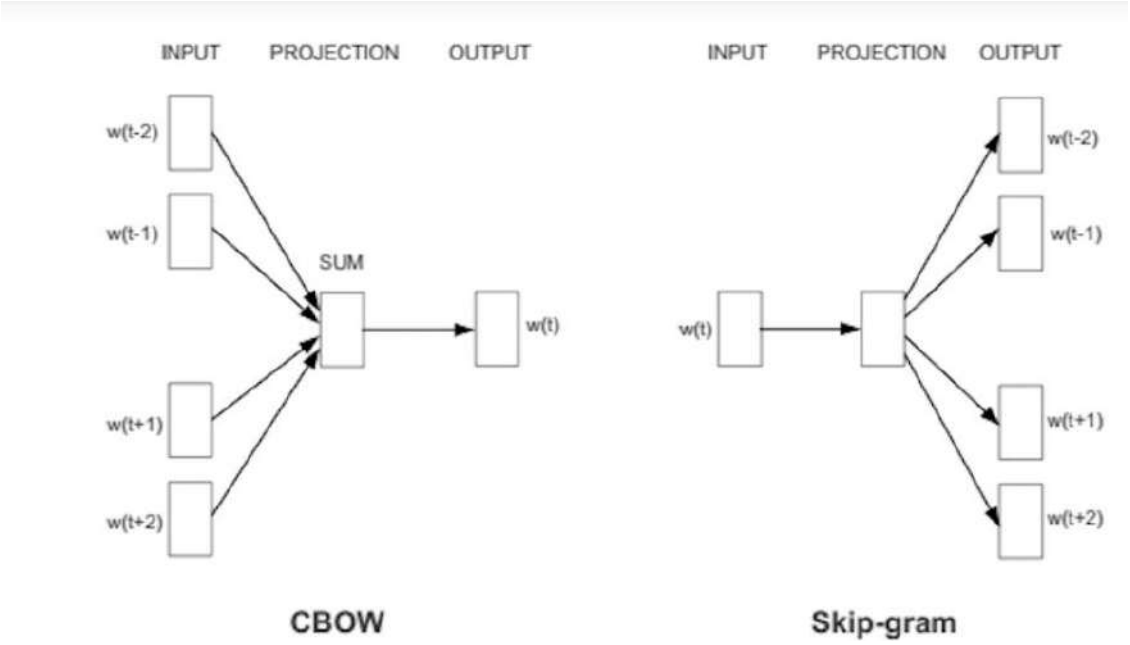


Figure 5.4: Two models of Word2Vec (A- CBOW & B- Skip-Gram model)

Word2vec is similar to an autoencoder, encoding each word in a vector, but rather than training against the input words through reconstruction, as a restricted Boltzmann machine does, word2vec trains words against other words that neighbour them in the input corpus. t does so in one of two ways, either using context to predict a target word (a method known as continuous bag of words, or CBOW), or using a word to predict a target context, which is called skip-gram.

When the feature vector assigned to a word cannot be used to accurately predict that word's context, the components of the vector are adjusted. Each word's context in the corpus is the *teacher* sending error signals back to adjust the feature vector. The vectors of words judged similar by their context are nudged closer together by adjusting the numbers in the vector.

Similar things and ideas are shown to be "close". Their relative meanings have been translated to measurable distances. Qualities become quantities, and algorithms can do their work. But similarity is just the basis of many associations that Word2vec can learn. For example, it can gauge relations between words of one language, and map them to another.

The main idea of word2Vec is to design a model whose parameters are the word vectors. Then, train the model on a certain objective. At every iteration we run our model, evaluate the errors, and follow an update rule that has some notion of penalizing the model parameters that caused the error. Thus, we learn our word vectors.

Content Source: (1) https://towardsdatascience.com/introduction-to-word-embedding-and-word2vec-652d0c2060fa

(2) https://wiki.pathmind.com/word2vec

5.5. Applications of Deep Learning Networks: Joint Detection

Deep Learning finds a lot of usefulness in the field of Biomedical and Bioinformatics. Deep Learning algorithms can be used for detecting fractures or anatomical changes in the human bones or in bone joints thereby early prediction of various diseases like arteritis can be done which helps in early curing of the so-called diseases also. Knee osteoarthritis (OA) is a very general joint disease that disturb many people especially people over 60. The severity of pain caused by knee OA is the most important portent to disable. Until now, the bad impact of osteoarthritis on health care and public health systems is still increasing.

Normal Neural Networks fails because of errors in the stages of Image Segmentation and Feature Extractions.TO avoid this we can build a Convolution based model as shown in the Figure 5.5 given below. In this example we had considered a CNN based Network. The input to this model is Knee Thermographs. Thermography is the image which senses or captures the heat intensity coming out from that particular region. Based on the patient's pressure points the color in the thermographs vary. Red regions denote more pressure locations and Yellow regions Denote less pressure locations. So, from the thermogram we can understand the effects of joint/Bone wear and tear or Damage occurred at particular spot.

The Convolution Filter is made to move over the image. The stride value here considered for this case study is 1. We have used Max Pooling and in the fully connected layer we have used Softmax aggregator.

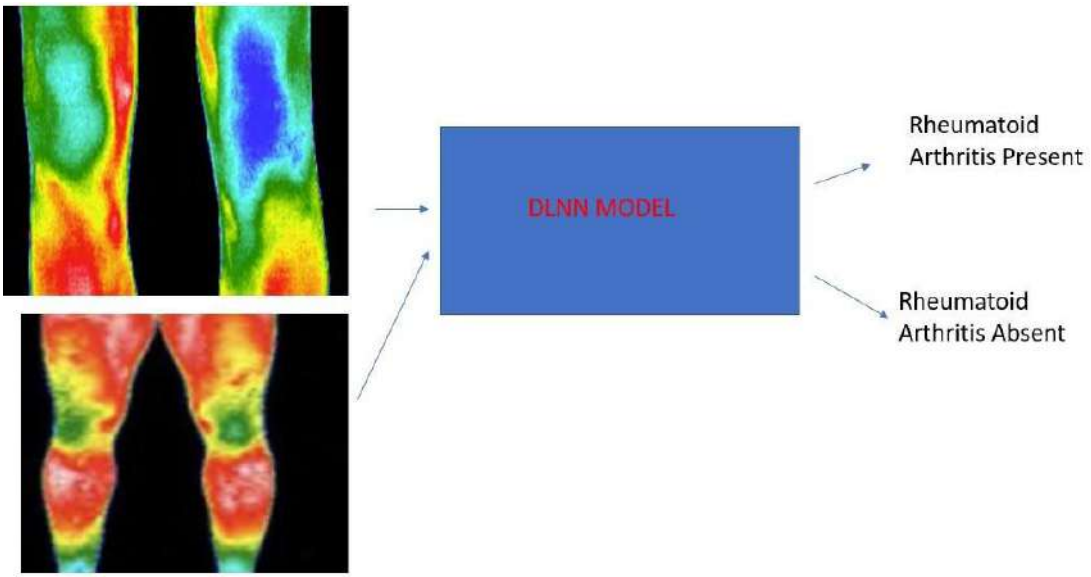


Figure 5.5: General representation of Bone Joint detection system

The above diagram is a schematic representation of a Deep Learning based network which can be used for human knee Joint deformities Identification purpose. Figure 5.6 shows the general Anatomical structure of a Human Knee Joint. Figure 5.6 shows the key structure skin texture of knee osteoarthritis. The left elevation of the image demonstrates the ordinary knee and the right elevation illustrates the contaminated joint. There have been many deep learning methods that can contribute well to the KOA diagnosis accurately as a part of early detection.

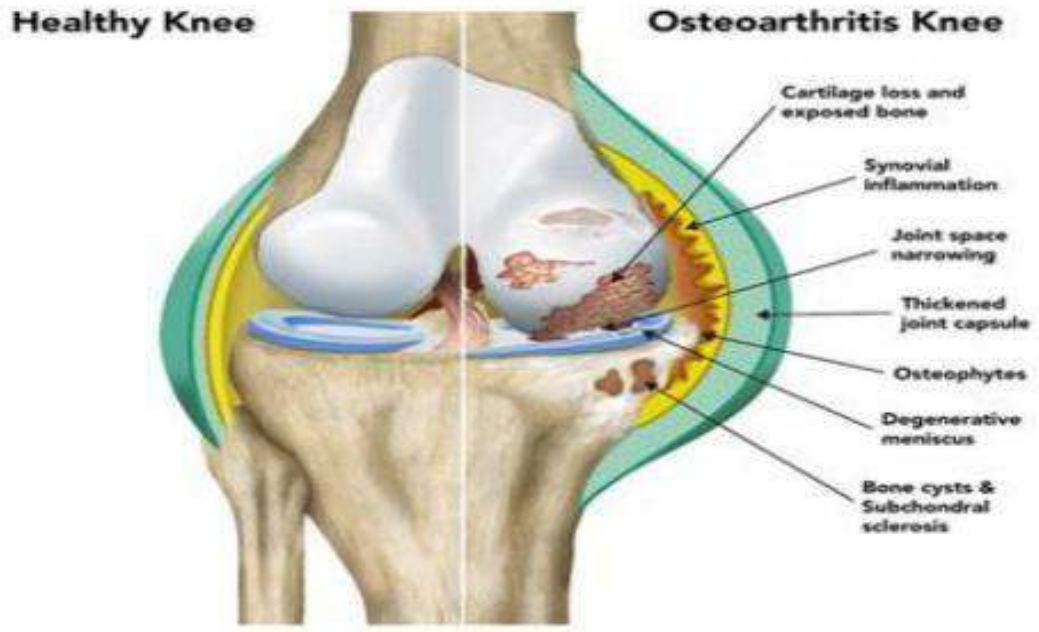


Figure 5.6: Human Knee Joint Structure

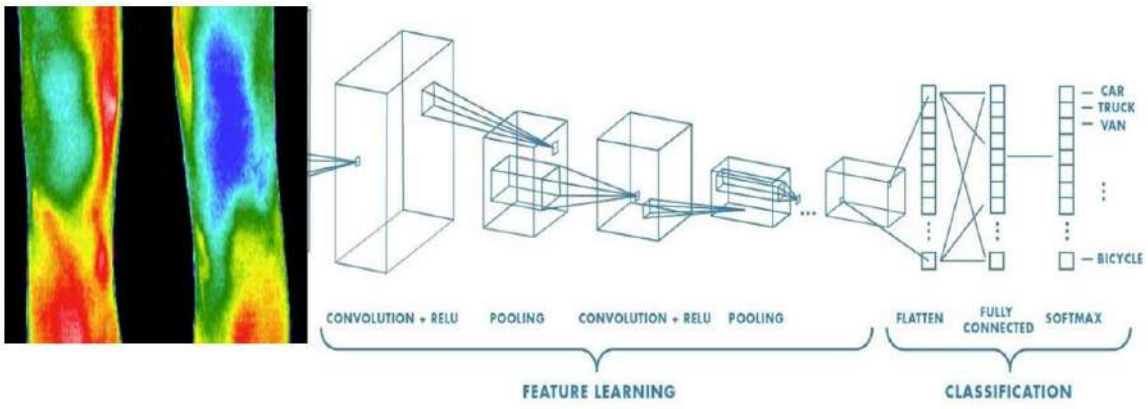


Figure 5.7: CNN based Knee Joint Detection Model

Figure 5.7 shows the full model of a joint detection procedure. The Convolution filter moves to the right with a certain Stride Value till it parses the complete width. Moving on, it hops down to the beginning (left) of the image with the same Stride Value and repeats the process until the entire image is traversed. The Kernel has the same depth as that of the input image. The objective of the Convolution Operation is to **extract the high-level features** such as edges, from the input image. Stride is the number of pixels shifts over the input matrix. When the stride is 1 then we move the filters to 1 pixel at a time. When the stride is 2 then we move the filters to 2 pixels at a time and so on

Pooling layers section would reduce the number of parameters when the images are too large. Spatial pooling also called subsampling or down sampling which reduces the dimensionality of each map but retains important information. This is to **decrease the computational power required to process the data** by reducing the dimensions

Types of Pooling:

- Max Pooling
- Average Pooling
- Sum Pooling

- The image is flattened 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. The feature map matrix will be converted as vector (x1, x2, x3, …). These features are combined together to create a model.

Finally, an activation function such as softmax or sigmoid is used to classify the outputs as Normal and Abnormal.

5.5.1 Steps Involved:

- Provide input image into convolution layer
- Choose parameters, apply filters with strides, padding if requires. Perform convolution on the image and apply ReLU activation to the matrix.
- Perform pooling to reduce dimensionality size
- Add as many convolutional layers until satisfied
- Flatten the output and feed into a fully connected layer (FC Layer)
- Output the class using an activation function (Logistic Regression with cost functions) and classifies images.

5.6. Other Applications:

Similarly for the other Applications such as Facial Recognition and Scene Matching applications appropriate Deep Learning Based Algorithms such as AlexNet, VGG, Inception, ResNet and or Deep learning-based LSTM or RNN can be used. These Networks has to be explained with necessary Diagrams and appropriate Explanations.

Reference Books:

1. B. Yegnanarayana, "Artificial Neural Networks" Prentice Hall Publications.
2. Simon Haykin, "Artificial Neural Networks", Second Edition, Pearson Education.
3. Laurene Fausett, "Fundamentals of Neural Networks, Architectures, Algorithms and Applications", Prentice Hall publications.
4. Cosma Rohilla Shalizi, Advanced Data Analysis from an Elementary Point of View, 2015.
5. 2. Deng & Yu, Deep Learning: Methods and Applications, Now Publishers, 2013.
6. 3. Ian Goodfellow, Yoshua Bengio, Aaron Courville, Deep Learning, MIT Press, 2016.
7. 4. Michael Nielsen, Neural Networks and Deep Learning, Determination Press, 2015.

Note: For further reference, kindly refer the class notes, PPTs, Video lectures available in the Learning Management System (Moodle)

********** ALL THE BEST **********