MCA 205C: Neural Networks

UNIT I

Introduction: What is Neural network, Human Brain, Models of a Neuron, Neural networks viewed as Directed Graphs, Network Architectures, Knowledge Representation, Artificial Intelligence and Neural Networks, Learning Process: Error Correction learning, Memory based learning, Hebbian learning, Competitive, Boltzmann learning, Credit Assignment Problem, Memory, Adaption, Statistical nature of the learning process,

UNIT II

Single Layer Perceptrons : Adaptive filtering problem, Unconstrained Organization Techniques, Linear least square filters, least mean square algorithm, learning curves, Learning rate annealing techniques, perception –convergence theorem, Relation between perception and Bayes classifier for a Gaussian Environment.

UNIT III

Multilayer Perceptron: Back propagation algorithm XOR problem, Heuristics, Output representation and decision rule, Computer experiment, feature detection, BACK PROPAGATION - back propagation and differentiation, Hessian matrix, Generalization, Cross validation, Network pruning Techniques, Virtues and limitations of back propagation learning, Accelerated convergence, supervised learning.

UNIT IV

Self-Organization Maps: Two basic feature mapping models, Self-organization map, SOM algorithm, properties of feature map, computer simulations, learning vector quantization, Adaptive patter classification, Hierarchal Vector quantifier, contexed Maps.

UNIT V

Neuro Dynamics: Dynamical systems, stability of equilibrium states, Attractors, Neurodynamical models, manipulation of attractors' as a recurrent network paradigm

HOPFIELD MODELS – Hopfield models.

Text Book:

1. Neural networks A comprehensive foundations, Simon Hhaykin, Pearson Education 2nd Edition 2004

Reference Books:

1. Artificial neural networks - B.Vegnanarayana Prentice Halll of India P Ltd 2005
2. Neural networks in Computer intelligence, Li Min Fu TMH 2003
3. Neural networks James A Freeman David M S kapura Pearson Education 2004

Lecture Notes

UNIT-1

Neural Network:

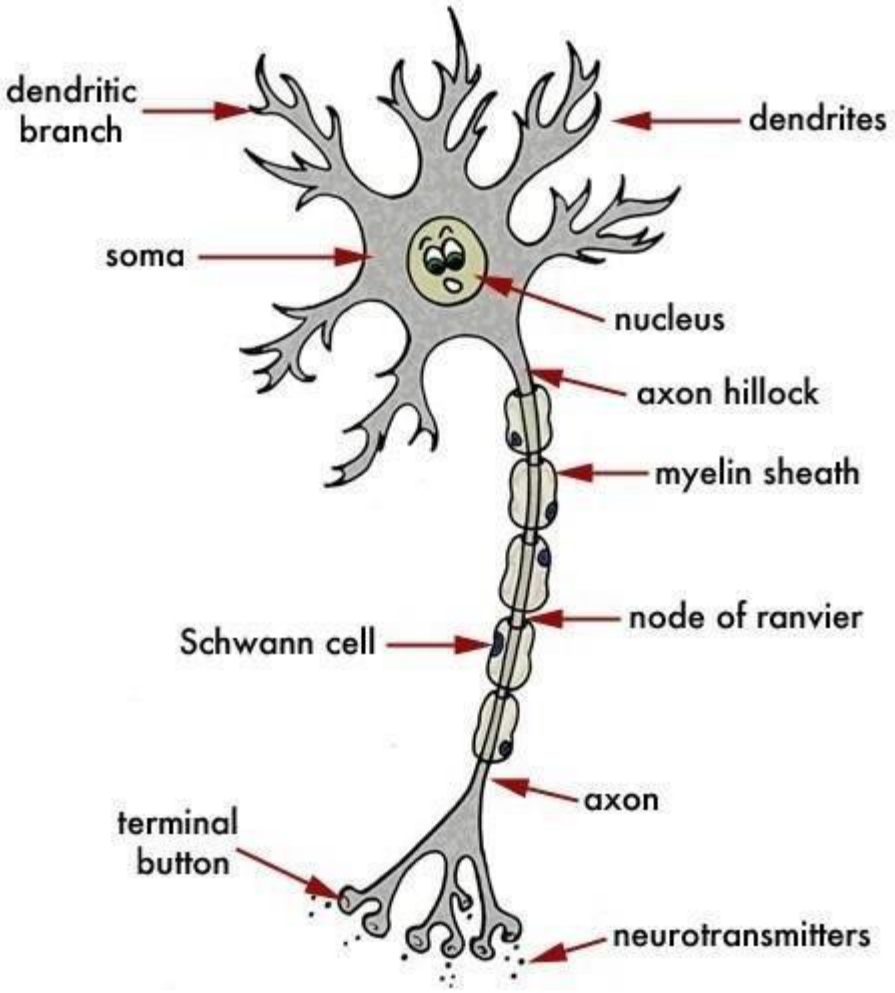
A neural network is a series of algorithms that endeavors to recognize underlying relationships in a set of data through a process that mimics the way the human brain operates. In this sense, neural networks refer to systems of neurons,either organic or artificial in nature.

Neural networks can adapt to changing input; so the network generates the best possible result without needing to redesign the output criteria. The concept of neural networks, which has its roots in artificial intelligence, is swiftly gaining popularity in the development of trading systems.

Human Brain in Neural Networks:

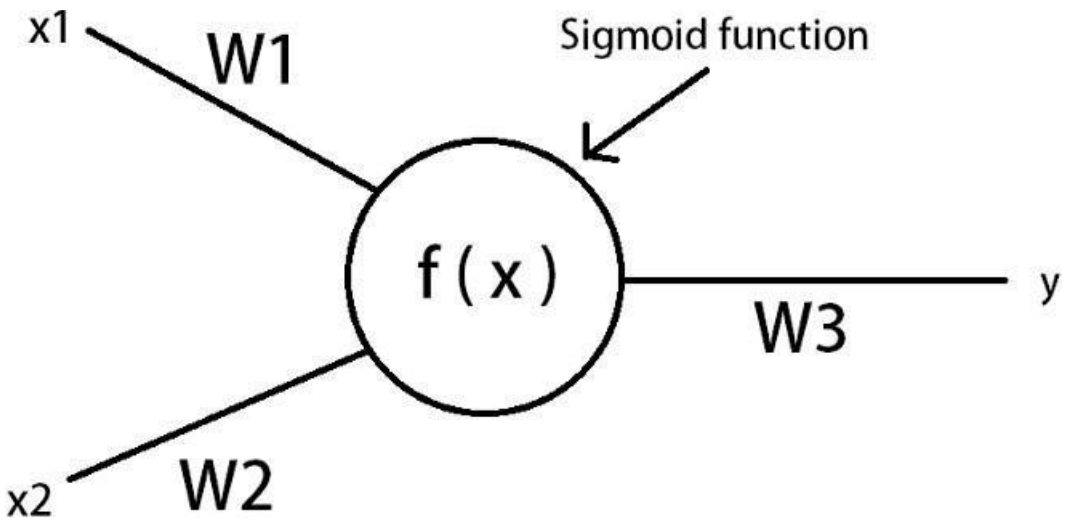
The human brain consists of neurons or nerve cells which transmit and process the information received from our senses. Many such nerve cells are arranged together in our brain to form a network of nerves. These nerves pass electrical impulses i.e the excitation from one neuron to the other.

The dendrites receive the impulse from the terminal button or synapse of an adjoining neuron. Dendrites carry the impulse to the nucleus of the nerve cell which is also called as soma. Here , the electrical impulse is processed and then passed on to the axon. The axon is longer branch among the dendrites which carries the impulse from the soma to the synapse. The synapse then , passes the impulse to dendrites of the second neuron. Thus, a complex network of neurons is created in the human brain.



The same concept of the network of neurons is used in machine learning algorithms. In this case , the neurons are created artificially on a computer . Connecting many such artificial neurons creates an artificial neural network. The working of an artificial neuron is similar to that of a neuron present in our brain.

Construction of an Artificial Neural:



The data in the network flows through each neuron by a connection. Every connection has a specific weight by which the flow of data is regulated.

In the above diagram ,

x1 and x2 are the two inputs. They could be an integer , float etc. Here , for example , we assume them as 1 and 0 respectively.

When these inputs pass through the connections (through W1 and W2) , they are adjusted depending on the connection weights.

Let us assume that W1 = 0.5 and W2 = 0.6 and W3 = 0.2 , then for adjusting the weights , we use ,

x1 * W1 + x2 * W2 = 1 * 0.5 + 0 * 0.6 = 0.5

Here we multiply the input with its weight and add them together. Hence , 0.5 is the value adjusted by the weights of the connection.The connections are assumed to be the dendrites in an artificial neuron.

Now , for processing the information we need an activation function (here , it can be assumed as the soma) . Here , I have used the simple sigmoid function .

f (x) = f (0.5) = 0.6224

The above value could be assumed as the output of the neuron (axon). This value needs to be multiplied by W3 .

0.6224 * W3 = 0.6224 * 0.2 = 0.12448

Now finally we apply the activation function to the above value ,

f (x) = f (0.12448) = 0.5310

Hence y (final prediction) = 0.5310

In this example the weights were randomly generated. Artificial Neural Network is a supervised machine learning algorithm usually used for regression problems. In the next article , I will be also showing the back propagation method to adjust the weights in a systematic way.

Models of Neuron:

Artificial neural network models are behind many of the most complex applications of machine learning. Classification, regression problems, and sentiment analysis are some of the ways artificial neural networks are being leveraged today. As

an emerging field, there are many different types of artificial neural networks. They vary for a variety of reasons, such as complexity, network architecture, density, and the flow of data. But the different types share a common goal of modelling and attempting to replicate the behaviour of neurons to improve machine learning.

Artificial neural networks have a wide range of uses in machine learning. Each type of artificial neural network model has different strengths and use cases. Overall, they are mainly used to solve more complex problems than would be possible with more traditional machine learning techniques. Examples may include complex natural language processing and machine learning-power language translation, which all rely on artificial neural networks. Recurrent neural networks are often utilised for analysis sentiment or translating text too. The depth and scale of the neural architecture means a non-linear decision making process can be achieved.

Artificial neural networks are used in the deep learning form of machine learning. It's called deep learning as models use the 'deep', multi-layered architecture of an artificial neural network. As each layer of an artificial neural network can process data, models can build an abstract understanding of the data. This architecture means models can perform increasingly complex tasks, for example understanding natural language or categorising complex file types.

Artificial neural networks are already used in machine learning to power:

- Recommendation systems for customers, users and consumers in products like streaming services or e-commerce.
- To power virtual assistance and speech recognition software.
- Complex image, audio and document classification models, for example in facial recognition software.
- In automatic feature extraction from raw, unlabelled data.

There are different types of artificial neural networks which vary in complexity. This guide explores the different types of artificial neural networks, including what they are and how they're used.

Artificial Neural Networks:

Artificial neural networks are designed to replicate the behaviour of neural networks found in human or animal brains. By mirroring and modelling the behaviour of neurons, machine learning gains the model architecture to process increasingly complex data. There are a variety of different types of artificial neural networks, with many early iterations seeming simple in comparison to emerging techniques. For example, artificial neural networks are used as the architecture for complex deep learning models.

Artificial neurons or nodes are modelled as a simplified version of neurons found in the brain. Each artificial neuron is connected to other nodes, though the density and amount of connections differ with each type of artificial neural network. The network is usually grouped into layers of nodes, which exist between the input and output layer. This multi-layered network architecture is also known as a deep neural network because of the depth of these layers. These different layers in the artificial neural network models can learn different features of data. Hidden hierarchical layers allow the understanding of complex concepts or patterns from processed data.

The structure of artificial neural networks represent a simplified reflection of the complexity of the human or animal brain. A web of interconnected artificial nodes mimic the behaviour of neurons within a nervous system. These artificial neural networks are much less complex than a human brain, but are still incredibly powerful at performing tasks such as classification. Data starts in the input layer and leaves from the output layer. But with the more complex artificial neural networks, data will move between many different layers in a non-linear way.

Complex artificial neural networks are developed so that models can mirror the nonlinear decision-making process of the human brain. This means models can be trained to make complex decisions or understand abstract concepts and objects. The model will build from low-level features to complex features, understanding complex concepts. Each node within the network is weighted depending on its influence on other artificial neural network nodes.

Like other machine learning models, optimisation of artificial neural networks is based on a loss function. This is the difference between a predicted and actual output. The weighting of each node and layer is adjusted by the model to achieve a minimum

loss. Artificial neural network models can understand multiple levels of data features, and any hierarchical relationship between features. So when used for a classification problem, an artificial neural network model can understand complex concepts by processing multiple layers of features.

5 types of neural network models explained:

There are many different types of artificial neural networks, varying in complexity. They share the intended goal of mirroring the function of the human brain to solve complex problems or tasks. The structure of each type of artificial neural network in some way mirrors neurons and synapses. However, they differ in terms of complexity, use cases, and structure. Differences also include how artificial neurons are modelled within each type of artificial neural network, and the connections between each node. Other differences include how the data may flow through the artificial neural network, and the density of the nodes.

5 examples of the different types of artificial neural network include:

- Feedforward artificial neural networks
- Perceptron and Multilayer Perceptron neural networks
- Radial basis function artificial neural networks
- Recurrent neural networks
- Modular neural networks

Feedforward artificial neural networks

As the name suggests, a Feedforward artificial neural network is when data moves in one direction between the input and output nodes. Data moves forward through layers of nodes, and won't cycle backwards through the same layers. Although there may be many different layers with many different nodes, the one-way movement of data makes Feedforward neural networks relatively simple. Feedforward artificial neural network models are mainly used for simplistic classification problems. Models will perform beyond the scope of a traditional machine learning model, but don't meet the level of abstraction found in a deep learning model.

Perceptron and Multilayer Perceptron neural networks

A perceptron is one of the earliest and simplest models of a neuron. A Perceptron model is a binary classifier, separating data into two different classifications. As a linear model it is one of the simplest examples of a type of artificial neural network.

Multilayer Perceptron artificial neural networks adds complexity and density, with the capacity for many hidden layers between the input and output layer. Each individual node on a specific layer is connected to every node on the next layer. This means Multilayer Perceptron models are fully connected networks, and can be leveraged for deep learning.

They're used for more complex problems and tasks such as complex classification or voice recognition. Because of the model's depth and complexity, processing and model maintenance can be resource and time-consuming.

Radial basis function artificial neural networks

Radial basis function neural networks usually have an input layer, a layer with radial basis function nodes with different parameters, and an output layer. Models can be used to perform classification, regression for time series, and to control systems. Radial basis functions calculate the absolute value between a centre point and a given point. In the case of classification, a radial basis function calculates the distance between an input and a learned classification. If the input is closest to a specific tag, it is classified as such.

A common use for radial basis function neural networks is in system control, such as systems that control power restoration after a power cut. The artificial neural network can understand the priority order to restoring power, prioritising repairs to the greatest number of people or core services.

Recurrent neural networks

Recurrent neural networks are powerful tools when a model is designed to process sequential data. The model will move data forward and loop it backwards to previous steps in the artificial neural network to best achieve a task and improve predictions. The layers between the input and output layers are recurrent, in that relevant information is looped back and retained. Memory of outputs from a layer is looped back to the input where it is held to improve the process for the next input.

The flow of data is similar to Feedforward artificial neural networks, but each node will retain information needed to improve each step. Because of this, models can better understand the context of an input and refine the prediction of an output. For example, a predictive text system may use memory of a previous word in a string of words to better predict the outcome of the next word. A recurrent artificial neural network would be better suited to understand the sentiment behind a whole sentence compared to more traditional machine learning models.

Recurrent neural networks are also used within sequence to sequence models, which are used for natural language processing. Two recurrent neural networks are used within these models, which consists of a simultaneous encoder and decoder. These models are used for reactive chatbots, translating language, or to summarise documents.

Modular neural networks

A Modular artificial neural network consists of a series of networks or components that work together (though independently) to achieve a task. A complex task can therefore be broken down into smaller components. If applied to data processing or the computing process, the speed of the processing will be increased as smaller components can work in tandem.

Each component network is performing a different subtask which when combined completes the overall tasks and output. This type of artificial neural network is beneficial as it can make complex processes more efficient, and can be applied to a range of environments.

4Explain network architecture in neural networks?

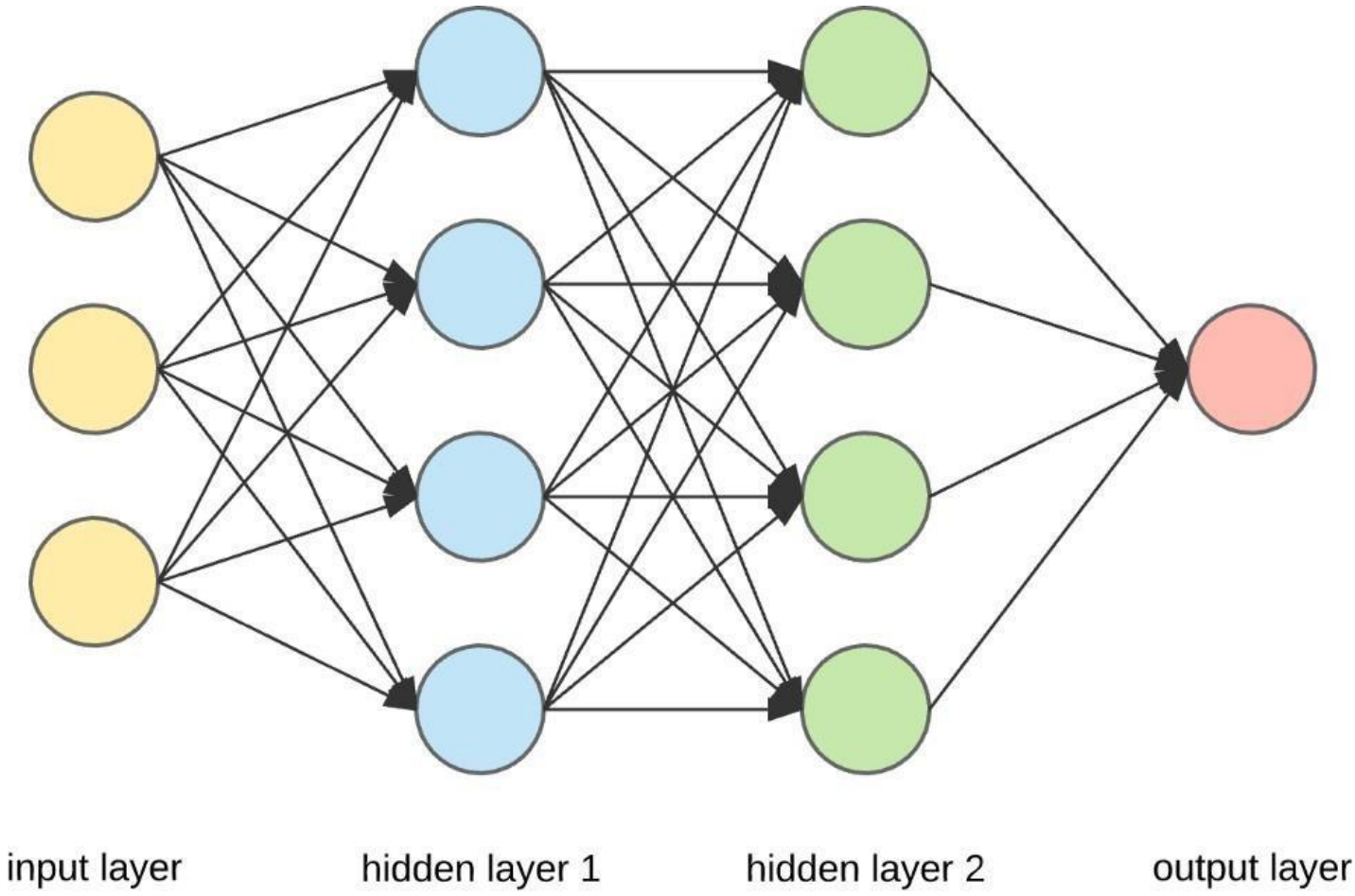
Ans: Neural Networks are complex structures made of artificial neurons that can take in multiple inputs to produce a single output. This is the primary job of a Neural Network – to transform input into a meaningful output. Usually, a Neural Network consists of an input and output layer with one or multiple hidden layers within. It is also known as Artificial Neural Network or ANN. ANN architecture in Neural Network functions just like a human brain and is very important.

In a Neural Network, all the neurons influence each other, and hence, they are all connected. The network can acknowledge and observe every aspect of the dataset at hand and how the different parts of data may or may not relate to each other. This is

how Neural Networks are capable of finding extremely complex patterns in vast volumes of data.

In a Neural Network, the flow of information occurs in two ways –

- **Feedforward Networks:** In this model, the signals only travel in one direction, towards the output layer. Feedforward Networks have an input layer and a single output layer with zero or multiple hidden layers. They are widely used in pattern recognition.
- **Feedback Networks:** In this model, the recurrent or interactive networks use their internal state (memory) to process the sequence of inputs. In them, signals can travel in both directions through the loops (hidden layer/s) in the network. They are typically used in time-series and sequential tasks.



Neural Network: Components

Artificial intelligence and neural networks:

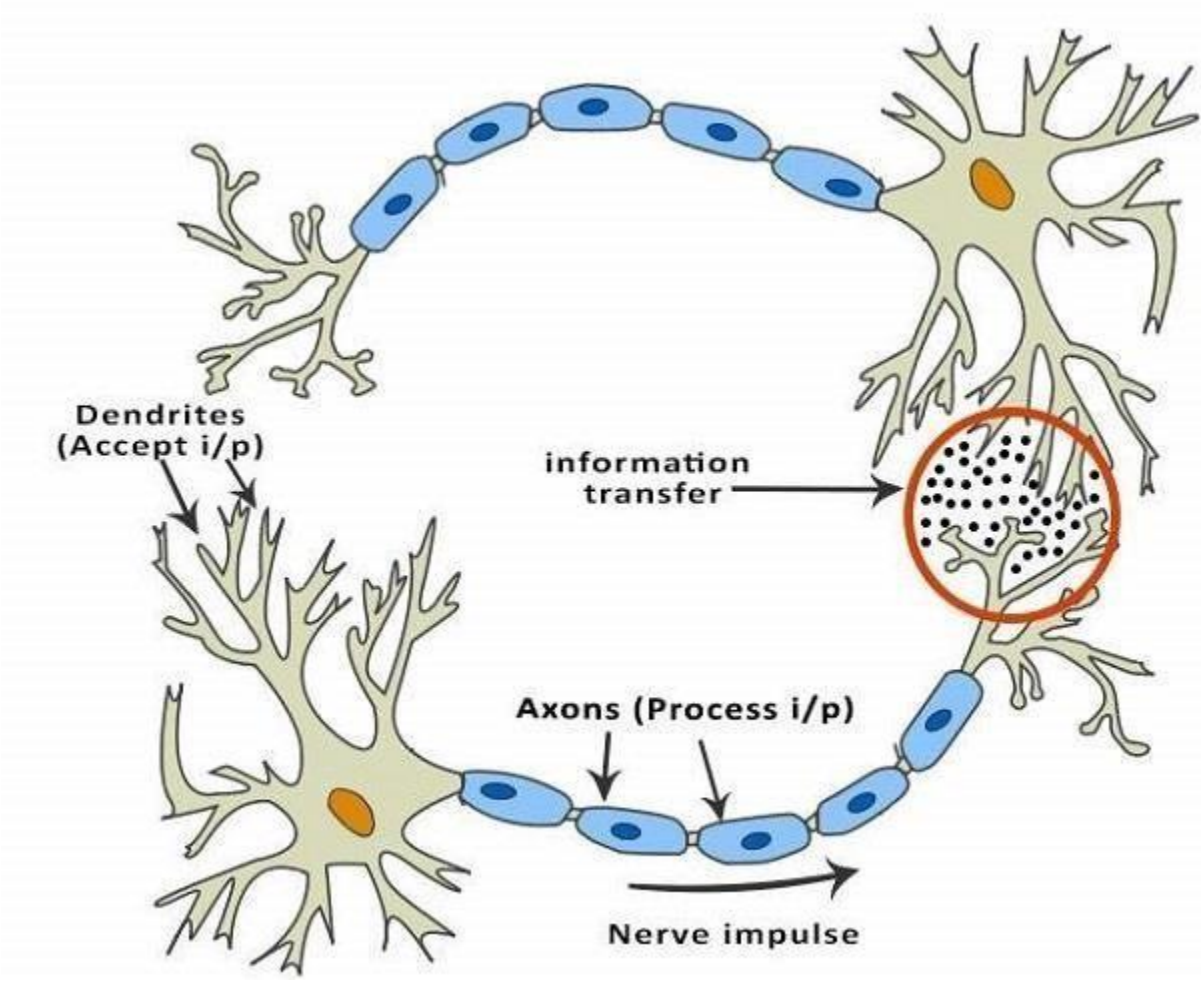
He inventor of the first neurocomputer, Dr. Robert Hecht-Nielsen, defines aneural network as −

"...a computing system made up of a number of simple, highly interconnected processing elements, which process information by their dynamic state response to external inputs."

Basic Structure of ANNs

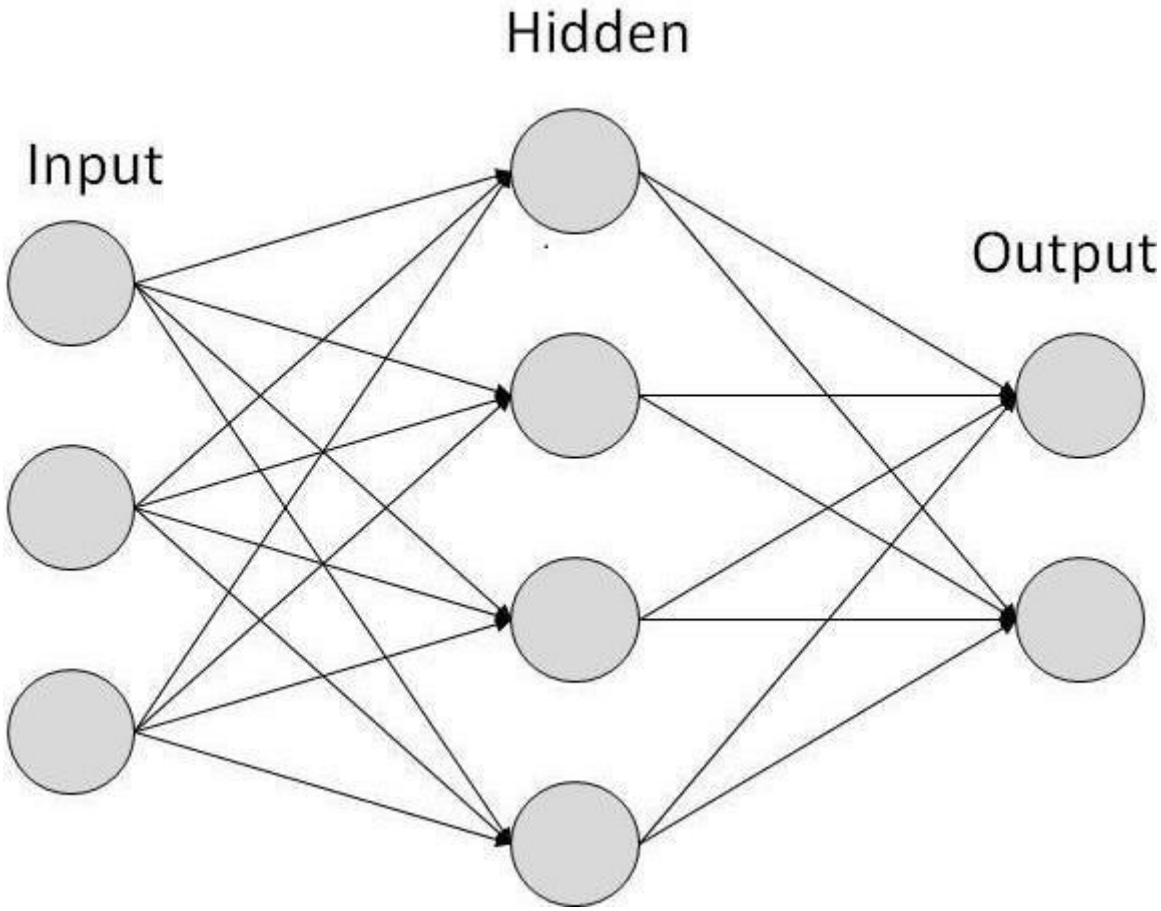
The idea of ANNs is based on the belief that working of human brain by making the right connections, can be imitated using silicon and wires as living **neurons** and **dendrites**.

The human brain is composed of 86 billion nerve cells called **neurons.** They are connected to other thousand cells by **Axons.** Stimuli from external environment or inputs from sensory organs are accepted by dendrites. These inputs create electric impulses, which quickly travel through the neural network. A neuron can then send the message to other neuron to handle the issue or does not send it forward.



ANNs are composed of multiple **nodes**, which imitate biological **neurons** of human brain. The neurons are connected by links and they interact with each other. The nodes can take input data and perform simple operations on the data. The result of these operations is passed to other neurons. The output at each node is called its **activation** or **node value.**

Each link is associated with **weight.** ANNs are capable of learning, which takes place by altering weight values. The following illustration shows a simple ANN −

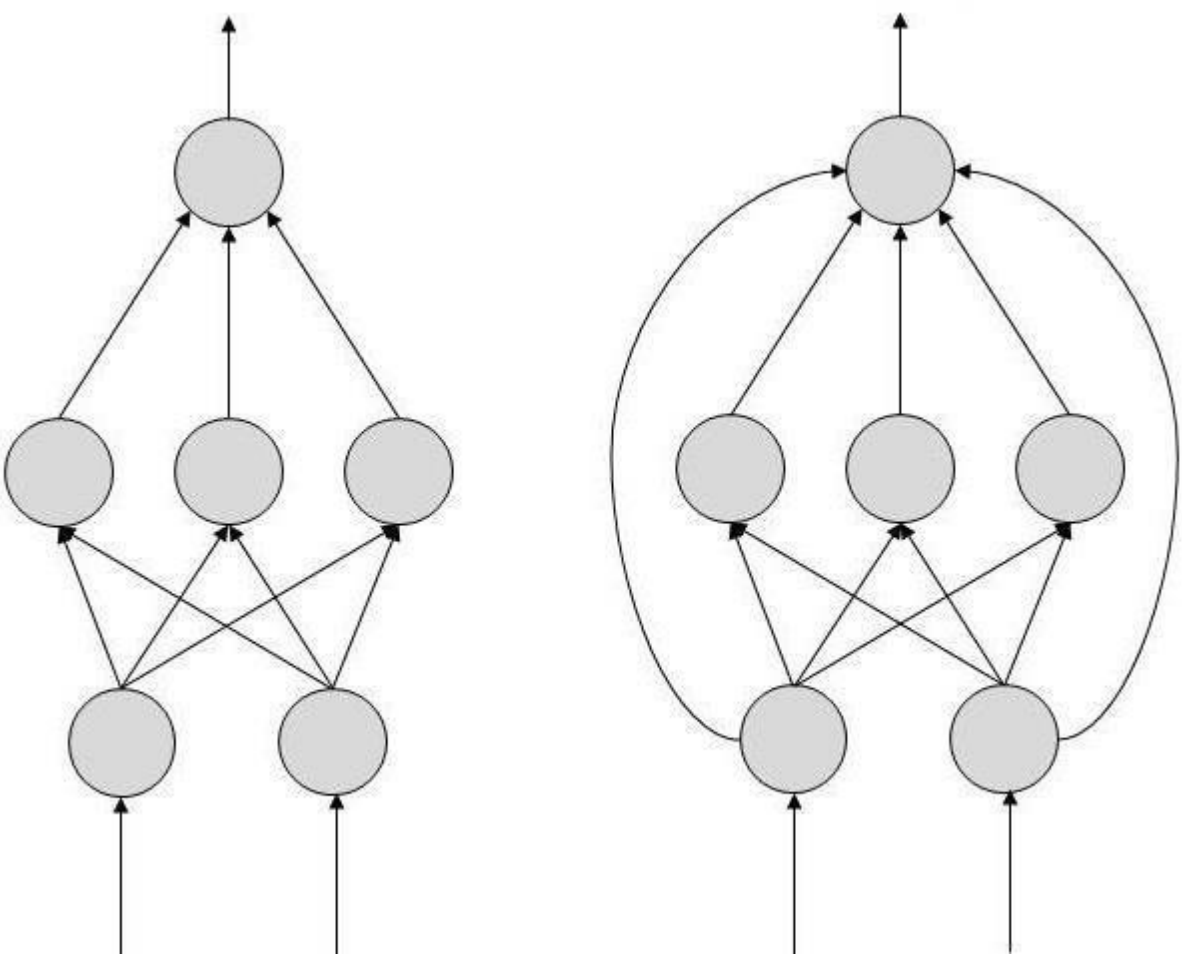


Types of Artificial Neural Networks

There are two Artificial Neural Network topologies − **FeedForward** and **Feedback.**

FeedForward ANN

In this ANN, the information flow is unidirectional. A unit sends information to other unit from which it does not receive any information. There are no feedback loops. They are used in pattern generation/recognition/classification. They have fixed inputs and outputs.

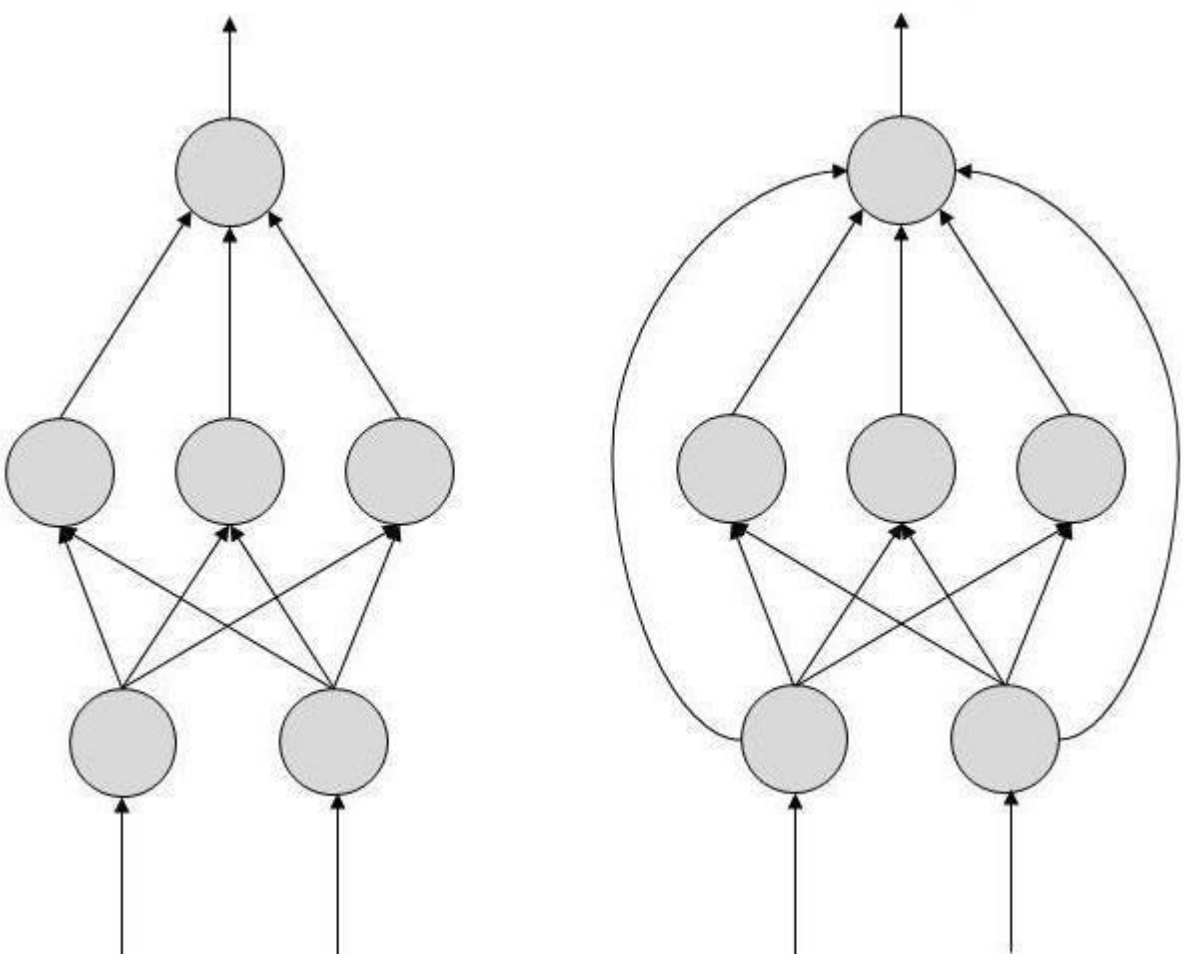


Types of Artificial Neural Networks

There are two Artificial Neural Network topologies − **FeedForward** and **Feedback.**

FeedForward ANN

In this ANN, the information flow is unidirectional. A unit sends information to other unit from which it does not receive any information. There are no feedback loops. They are used in pattern generation/recognition/classification. They have fixed inputs and outputs.



Error correction learning in neural networks:

Ans: Error-Correction Learning, used with supervised learning, is the technique of comparing the system output to the desired output value, and using that error to direct the training. In the most direct route, the error values can be used to directly adjust the tap weights, using an algorithm such as the backpropagation algorithm. If the system output is *y*, and the desired system output is known to be *d*, the error signal can be defined as:

Error correction learning algorithms attempt to minimize this error signal at each training iteration. The most popular learning algorithm for use with error-correction learning is the backpropagation algorithm, discussed below.

Gradient Descent[edit | edit source]

The gradient descent algorithm is not specifically an ANN learning algorithm. It has a large variety of uses in various fields of science, engineering, and mathematics. However, we need to discuss the gradient descent algorithm in order to fully understand the backpropagation algorithm. The gradient descent algorithm is used to minimize an error function *g(y)*, through the manipulation of a weight vector *w*. The cost function should be a linear combination of the weight vector and an input vector *x*. The algorithm is:

Here, η is known as the step-size parameter, and affects the rate of convergence of the algorithm. If the step size is too small, the algorithm will take a long time to converge. If the step size is too large the algorithm might oscillate or diverge.

The gradient descent algorithm works by taking the gradient of the weight space to find the path of steepest descent. By following the path of steepest descent at each iteration, we will either find a minimum, or the algorithm could diverge if the weight space is infinitely decreasing. When a minimum is found, there is no guarantee that it is a global minimum, however.

Backpropagation[edit | edit source]

The **backpropagation** algorithm, in combination with a supervised error-correction learning rule, is one of the most popular and robust tools in the training of artificial neural networks. Back propagation passes error signals backwards through the network during training to update the weights of the network. Because of this dependence on bidirectional data flow during training, backpropagation is not a plausible reproduction of biological learning mechanisms. When talking about backpropagation, it is useful to define the term **interlayer** to be a layer of neurons, and the corresponding input tap weights to that layer. We use a superscript to denote a specific interlayer, and a subscript to denote the specific neuron from within that layer. For instance:

(1)

(2)

Where x_i^{l-1} are the outputs from the previous interlayer (the inputs to the current interlayer), w_{ij}^{l} is the tap weight from the *i* input from the previous interlayer to the *j* element of the current interlayer. N^{l-1} is the total number of neurons in the previous interlayer.

The backpropagation algorithm specifies that the tap weights of the network are updated iteratively during training to approach the minimum of the error function. This is done through the following equation:

(3)

(3)

The relationship between this algorithm and the gradient descent algorithm should be immediately apparent. Here, η is known as the learning rate, not the step-size, because it affects the speed at which the system learns (converges). The parameter μ is known as the momentum parameter. The momentum parameter forces the search to take into account its movement from the previous iteration. By doing so, the system will tend to avoid local minima or saddle points, and approach the global minimum. We will discuss these terms in greater detail in the next section.

The parameter δ is what makes this algorithm a "back propagation" algorithm. We calculate it as follows:

(4)

The δ function for each layer depends on the δ from the previous layer. For the special case of the output layer (the highest layer), we use this equation instead:

(5)

In this way, the signals propagate backwards through the system from the output layer to the input layer. This is why the algorithm is called the backpropagation algorithm.

Log-Sigmoid Backpropagation**[edit | edit source]**

If we use log-sigmoid activation functions for our neurons, the derivatives simplify, and our backpropagation algorithm becomes:

For the output layer, and

Memory based learning in neural networks:

The basic idea behind memory-based learning is that concepts can be classified by their similarity with previously seen concepts. In a memory-based system, learning amounts to storing the training data items. The strength of such a system lies in its capability to compute the similarity between a new data item and the training data items. The most simple similarity metric is the overlap metric [Daelemans et al.(2000)]. It compares corresponding features of the data items and adds 1 to a similarity rate when they are different. The similarity between two data items is represented by a number between zero and the number of features, n, in which value zero corresponds with an exact match and n corresponds with two items which share no feature value. Here is an example:

TRAIN1	man	saw	the	V
TRAIN2	the	saw	.	N
TEST1	boy	saw	the	?

It contains two training items of a part-of-speech (POS) tagger and one test item for which we want to obtain a POS tag. Each item contains three features: the word that needs to be tagged (*saw*) and the preceding and the next word. In order to find the best POS tag for the test item, we compare its features with the features of the training data items. The test item shares two features with the first training data item and one with the second. The similarity value for the first training data item (1) is smaller than that of the second (2) and therefore the overlap metric will prefer the first.

A weakness of the overlap metric is that it regards all features as equally valuable for computing similarity values. Generally some features are more important than others. For example, when we add a line ``TRAIN3 boy and the C'' to our training data, the overlap metric will regard this new item as equally important as the first training item. Both the first and the third training item share two feature values with the test item but we would like the third to receive a lower similarity value because it does not contain the word for which we want find a POS tag (*saw*). In order to accomplish this, we assign weights to the features in such a way that the second feature receives a higher weight than the other two.

The method which we use to assign weights to the features is called Gain Ratio, a normalized variant of information gain [Daelemans et al.(2000)]. It estimates feature weights by examining the training data and determines for each feature how much information it contributes to the knowledge of the classes of the training data items. The weights are normalized in order to account for features with many different values. The Gain Ratio computation of the weights is summarized in the following formulas:

$$w_i = \frac{H(C) - \sum_{v \in V_i} P(v) \times H(C \mid v)}{H(V_i)} \quad (1)$$

$$H(X) = -\sum_{x \in X} P(x) \log_2 P(x) \quad (2)$$

Here w_i is the weight of feature i, C the set of class values and V_i the set of values that feature i can take. $H(C)$ and $H(V_i)$ are the entropy of the

sets C and V_i respectively and $H(C \mid v)$ is the entropy of the subset of elements of C that are associated with value v of feature i. $P(v)$ is the probability that feature i has value v. The normalization factor $H(V_i)$ was introduced to prevent that features with low generalization capacities, like identification codes, would obtain large weights.

The memory-based learning software which we have used in our experiments, TiMBL [Daelemans et al.(2000)], contains several algorithms with different parameters. In this paper we have restricted ourselves to using a single algorithm (k nearest neighbor classification) with a constant parameter setting. It would be interesting to evaluate every algorithm with all of its parameters but this would require a lot of extra work. We have changed only one parameter of the nearest neighbor algorithm from its default value: the size of the nearest neighborhood region. The learning algorithm computes the distance between the test item and the training items. The test item will receive the most frequent classification of the nearest training items (nearest neighborhood size is 1). [Daelemans et al.(1999)] show that using a larger neighborhood is harmful for classification accuracy for three language tasks but not for noun phrase chunking, a task which is central to this paper. In our experiments we have found that using the three nearest sets of data items leads to a better performance than using only the nearest data items. This increase of the neighborhood size used leads to a form of smoothing which can get rid of the influence of some data inconsistencies and exceptions.

Hebbian learning in neural networks:

Definition

Hebbian learning is a form of activity-dependent synaptic plasticity where correlated activation of pre- and postsynaptic neurons leads to the strengthening of the connection between the two neurons. The learning principle was first proposed by Hebb (1949), who postulated that a presynaptic neuron A, if successful in repeatedly activating a postsynaptic neuron B when itself (neuron A) is active, will gradually

become more effective in activating neuron B. The concept is widely employed and investigated in both experimental and computational neuroscience.

Detailed Description

In this article, we will review computational formulations and theoretical bases of Hebbian learning and its variants. We will also briefly review neurobiological underpinnings of Hebbian learning. See Frégnac (2003), Gerstner and Kistler (2002), Shouval (2007), and Dayan and Abbott (2001) for a more in-depth review of this topic.

Short Questions

1. What is neural network?
2. Explain error Correction learning?
3. What is credit assignment problem?
4. What are models of a neuron?

Long Questions

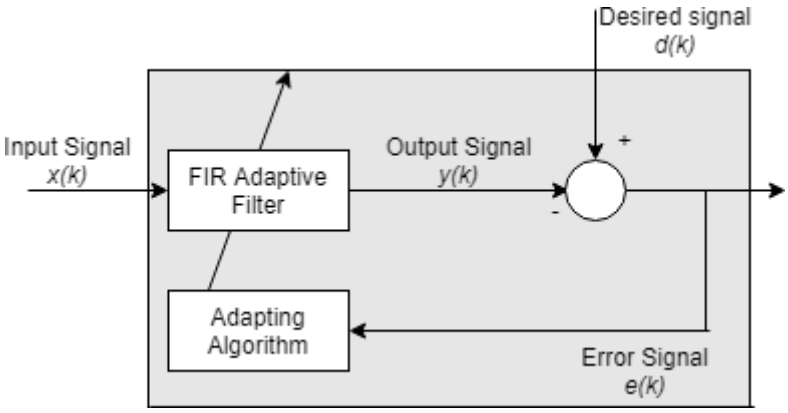
1. Explain about network architectures?
2. Explain about artificial intelligence and neural networks?
3. Explain the process of Hebbian learning?
4. Explain the process of Boltzmann learning?

UNIT-2

Adaptive filtering problem:

Adaptive filters are digital filters whose coefficients change with an objective to make the filter converge to an optimal state. The optimization criterion is a cost function, which is most commonly the mean square of the error signal between the output of the adaptive filter and the desired signal. As the filter adapts its coefficients, the mean square error (MSE) converges to its minimal value. At this state, the filter is adapted and the coefficients have converged to a solution. The filter output, *y(k)*, is then said to match very closely to the desired signal, *d(k)*. When you change the input data characteristics, sometimes called *filter environment*, the filter adapts to the new environment by generating a new set of coefficients for the new data.

General Adaptive Filter Algorithm



Adaptive Filters in DSP System Toolbox

Least Mean Squares (LMS) Based FIR Adaptive Filters

Adaptive Filter Object	**Adapting Algorithm**
dsp.BlockLMSFilter	Block LMS FIR adaptive filter algorithm
dsp.FilteredXLMSFilter	Filtered-x LMS FIR adaptive filter algorithm
dsp.LMSFilter	LMS FIR adaptive filter algorithm Normalized LMS FIR adaptive filter algorithm Sign-data LMS FIR adaptive filter algorithm Sign-error LMS FIR adaptive filter algorithm Sign-sign LMS FIR adaptive filter algorithm
Adaptive Filter Block	**Adapting Algorithm**
Block LMS Filter	Block LMS FIR adaptive filter algorithm
Fast Block LMS Filter	Block LMS FIR adaptive filter algorithm in freque
LMS Filter	LMS FIR adaptive filter algorithm Normalized LMS FIR adaptive filter algorithm Sign-data LMS FIR adaptive filter algorithm Sign-error LMS FIR adaptive filter algorithm Sign-sign LMS FIR adaptive filter algorithm
LMS Update	LMS FIR weight update algorithm

Adaptive Filter Object	Adapting Algorithm
	Normalized LMS FIR weight update algorithm
	Sign-data LMS FIR weight update algorithm
	Sign-error LMS FIR weight update algorithm
	Sign-sign LMS FIR weight update algorithm

Recursive Least Squares (RLS) Based FIR Adaptive Filters

Adaptive Filter Object	Adapting Algorithm
dsp.FastTransversalFilter	Fast transversal least-squares adaptation algorithm
	Sliding window FTF adaptation algorithm
dsp.RLSFilter	QR-decomposition RLS adaptation algorithm
	Householder RLS adaptation algorithm
	Householder SWRLS adaptation algorithm
	Recursive-least squares (RLS) adaptation algorith
	Sliding window (SW) RLS adaptation algorithm
Adaptive Filter Block	**Adapting Algorithm**
RLS Filter	Exponentially weighted recursive least-squares (R

Affine Projection (AP) FIR Adaptive Filters

Adaptive Filter Object	Adapting Algorithm
dsp.AffineProjectionFilter	Affine projection algorithm that uses direct matr
	Affine projection algorithm that uses recursive m
	Block affine projection adaptation algorithm

FIR Adaptive Filters in the Frequency Domain (FD)

Adaptive Filter Object	Adapting Algorithm
dsp.FrequencyDomainAdaptiveFilter	Constrained frequency domain adaptation
	Unconstrained frequency domain adaptat

Adaptive Filter Object	Adapting Algorithm
	Partitioned and constrained frequency do
	Partitioned and unconstrained frequency
Adaptive Filter Block	**Adapting Algorithm**
Frequency-Domain Adaptive Filter	Constrained frequency domain adaptation
	Unconstrained frequency domain adapta
	Partitioned and constrained frequency do
	Partitioned and unconstrained frequency

Lattice-Based (L) FIR Adaptive Filters

Adaptive Filter Object	Adapting Algorithm
dsp.AdaptiveLatticeFilter	Gradient adaptive lattice filter adaptation al
	Least squares lattice adaptation algorithm
	QR decomposition RLS adaptation algorith

For more information on these algorithms, refer to the algorithm section of the respective reference pages. Full descriptions of the theory appear in the adaptive filter references [1] and [2].

Choosing an Adaptive Filter

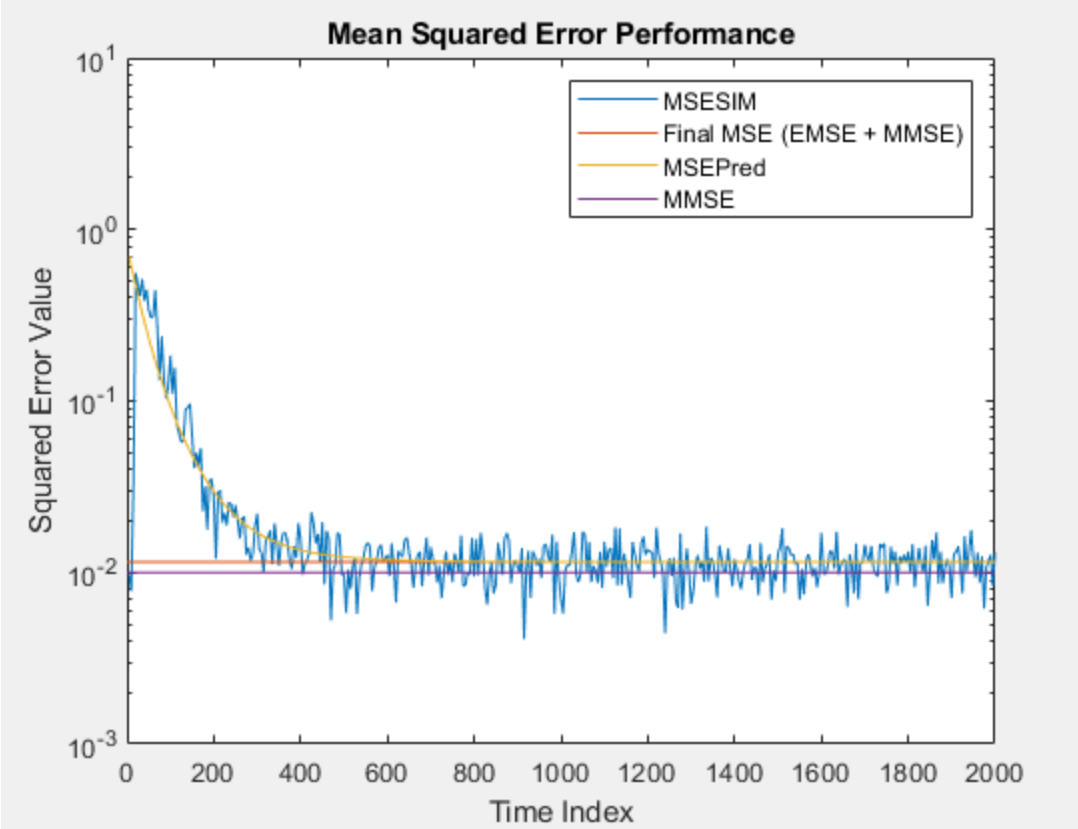
At steady state when the filter has adapted, the error between the filter output and the desired signal is minimal, not zero. This error is known as the steady state error. The speed with which the filter converges to the optimal state, known as the convergence speed, depends on multiple factors such nature of the input signal, choice of the adaptive filter algorithm, and step size of the algorithm. The choice of the filter algorithm usually depends factors such as convergence performance required for the application, computational complexity of the algorithm, filter stability in the environment, and any other constraints.

LMS algorithm is simple to implement, but has stability issues. The normalized version of the LMS algorithm comes with improved convergence speed, more stability, but has increased computational complexity. For an example that compares

the two, see Compare Convergence Performance Between LMS Algorithm and Normalized LMS Algorithm. RLS algorithms are highly stable, do very well in time-varying environments, but are computationally more complex than the LMS algorithms. For a comparison, see Compare RLS and LMS Adaptive Filter Algorithms. Affine projection filters do well when the input is colored and have a very good convergence performance. Adaptive lattice filters provide good convergence but come with increased computational cost. The choice of the algorithm depends on the environment and the specifics of the application.

Mean Squared Error Performance

Minimizing the mean square of the error signal between the output of the adaptive filter and the desired signal is the most common optimization criterion for adaptive filters. The actual MSE (MSESIM) of the adaptive filter you are implementing can be determined using the msesim function. The trajectory of this MSE is expected to follow that of the predicted MSE (MSEPred), which is computed using the msepred function. The minimum mean square error (MMSE) is estimated by the msepred function using a Wiener filter. The Weiner filter minimizes the mean squared error between the desired signal and the input signal filtered by the Wiener filter. A large value of the mean squared error indicates that the adaptive filter cannot accurately track the desired signal. The minimal value of the mean squared error ensures that the adaptive filter is optimal. The excess mean square error (EMSE), determined by the msepred function, is the difference between the MSE introduced by the adaptive filters and the MMSE produced by the corresponding Wiener filter. The final MSE shown below is the sum of EMSE and MMSE, and equals the predicted MSE after convergence.



Unconstrained organization techniques:

Optimization is a very old subject of a great interest; we can search deep into a human history to find important examples of applying optimization in the usual life ofa human being, for example, the need of finding the best way to produce food yielded finding the best piece of land for producing, as well as (later on, how the time was going) the best ways of treatment of the chosen land and the chosen seedlings to get the best results.

From the very beginning of manufacturing, the manufacturers were trying to find the ways to get maximum income with minimum expenses.

There are plenty of examples of optimization processes in pharmacology (for determination of the geometry of a molecule), in meteorology, in optimization of a trajectory of a deep-water vehicle, in optimization of power management (optimization of the production of electrical power plants), etc.

Optimization presents an important tool in decision theory and analysis of physical systems.

Optimization theory is a very developed area with its wide application in science, engineering, business management, military, and space technology.

Optimization can be defined as the process of finding the best solution to a problem in a certain sense and under certain conditions.

Along with the passage of time, optimization was evolving. Optimization became an independent area of mathematics in 1940, when Dantzig presented the so-called simplex algorithm for linear programming.

The development of nonlinear programming became great after presentation of conjugate gradient methods and quasi-Newton methods in the 1950s.

Today, there exist many modern optimization methods which are made to solve a variety of optimization problems. Now, they present the necessary tool for solving problems in diverse fields.

At the beginning, it is necessary to define an objective function, which, for example, could be a technical expense, profit or purity of materials, time, potential energy, etc.

The object function depends on certain characteristics of the system, which are known as variables. The goal is to find the values of those variables, for which the object function reaches its best value, which we call an extremum or an optimum.

It can happen that those variables are chosen in such a way that they satisfy certain conditions, i.e., restrictions.

The process of identifying the object function, variables, and restrictions for the given problem is called *modeling*.

The first and the most important step in an optimization process is the construction of the appropriate model, and this step can be the problem by itself. Namely, in the case that the model is too much simplified, it cannot be a faithful reflection of the practical

problem. By the other side, if the constructed model is too complicated, then solving the problem is also too complicated.

After the construction of the appropriate model, it is necessary to apply the appropriate algorithm to solve the problem. It is no need to emphasize that there does not exist a universal algorithm for solving the set problem.

Sometimes, in the applications, the set of input parameters is bounded, i.e., the input parameters have values within the allowed space of input parameters Dx ; we can write

$$x \in Dx. \quad (1) \quad E1$$

Except (1), the next conditions can also be imposed:

$$\varphi l(x1,\ldots,xn)=\varphi 0l, l=1,\ldots,m1<n, \quad (2) \quad E2$$
$$\psi j(x1,\ldots,xn)\leq\psi 0j, j=1,\ldots,m2. \quad (3) \quad E3$$

Optimization task is to find the minimum (maximum) of the objective function $f(x)=f(x1,\ldots,xn)$, under the conditions (1), (2), and (3).

If the object function is linear, and the functions $\varphi l(x1,\ldots,xn)l=1,\ldots,m1$ and $\psi j(x1,\ldots,xn)j=1,\ldots,m2$ are linear, then it is about the linear programming problem, but if at least one of the mentioned functions is nonlinear, it is about the nonlinear programming problem.

Unconstrained optimization problem can be presented as

$$(4) \quad E4$$

where f ∈ R n is a smooth function.

Problem (4) is, in fact, the unconstrained minimization problem. But, it is well known that the unconstrained minimization problem is equivalent to an unconstrained maximization problem, i.e.

$$\min f\,x = -\max -f\,x\,, \quad (5) \quad E5$$

as well as

$$\max f\, x = - \min - f\, x \,. \quad (6)\ E6$$

Definition 1.1.1 $x *$ *is called a global minimizer of* f *if* $f\, x * \leq f\, x$ *for all* $x \in R\, n$.

The ideal situation is finding a global minimizer of f . Because of the fact that our knowledge of the function f is usually only local, the global minimizer can be very difficult to find. We usually do not have the total knowledge about f . In fact, most algorithms are able to find only a local minimizer, i.e., a point that achieves the smallest value of f in its neighborhood.

So, we could be satisfied by finding the local minimizer of the function f . We distinguish weak and strict (or strong) local minimizer.

Formal definitions of local weak and strict minimizer of the function f are the next two definitions, respectively.

Definition 1.1.2 $x *$ *is called a weak local minimizer of* f *if there exists a neighborhood* N *of* $x *$ *, such that* $f\, x * \leq f\, x$ *for all* $x \in N$.

Definition 1.1.3 $x *$ *is called a strict (strong) local minimizer of* f *if there exists a neighborhood* N *of* $x *$ *, such that* $f\, x * < f\, x$ *for all* $x \in N$.

Considering backward definitions 1.1.2 and 1.1.3, the procedure of finding local minimizer (weak or strict) does not seem such easy; it seems that we should examine all points from the neighborhood of $x *$, and it looks like a very difficult task.

Fortunately, if the object function f satisfies some special conditions, we can solve this task in a much easier way.

For example, we can assume that the object function f is smooth or, furthermore, twice continuously differentiable. Then, we concentrate to the gradient $\nabla f\, x *$ as well as to the Hessian $\nabla 2\, f\, x *$.

All algorithms for unconstrained minimization require the user to start from a certain point, so-called the starting point, which we usually denote by $x\, 0$. It is good to

choose x 0 such that it is a reasonable estimation of the solution. But, to find such estimation, a little more knowledge about the considered set of data is needed, and the systematic investigation is needed also. So, it seems much simpler to use one of the algorithms to find x 0 or to take it arbitrarily.

There exist two important classes of iterative methods—*line search methods* and *trust-region methods*—made in the aim to solve the unconstrained optimization problem (4).

In this chapter, at first, we discuss different kinds of line search. Then, we consider some line search optimization methods in details, i.e., we study steepest descent method, Barzilai-Borwein gradient method, Newton method, and quasi-Newton method.

Also, we try to give some of the most recent results in these areas.

2. Line search

Now, let us consider the problem

$$\min x \in R n f x , (7) \quad E7$$

where f : R n → R is a continuously differentiable function, bounded from below.

There exists a great number of methods made in the aim to solve the problem (7).

The optimization methods based on line search utilize the next iterative scheme:

$$x k + 1 = x k + t k d k , (8) \quad E8$$

where x k is the current iterative point, x k + 1 is the next iterative point, d k is the search direction, and t k is the step size in the direction d k .

At first, we consider the monotone line search.

Now, we give the iterative scheme of this kind of search.

Algorithm 1.2.1. *(Monotone line search).*

Assumptions: $\epsilon > 0$, $x\ 0$, $k := 0$.

Step 1. If $\| g\ k \| \leq \epsilon$, then STOP.

Step 2. Find the descent direction $d\ k$.

Step 3. Find the step size $t\ k$, such that $f\ x\ k + t\ k\ d\ k < f\ x\ k$.

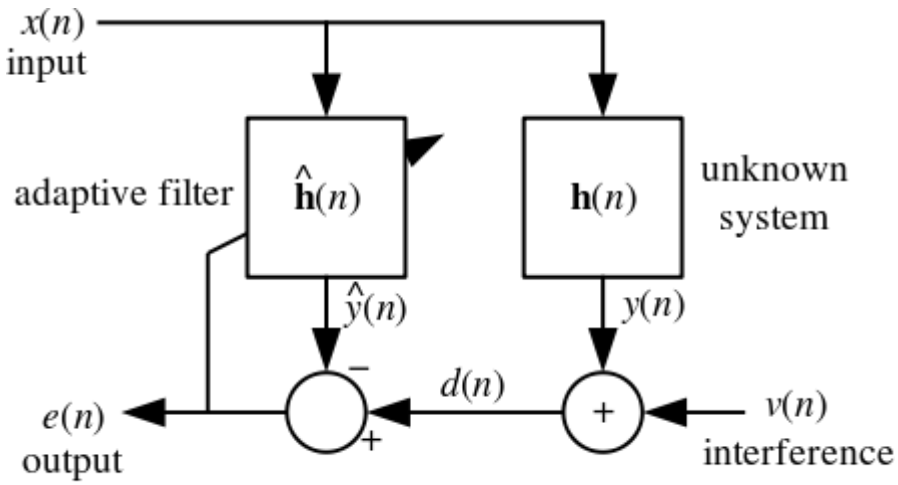
Step 4. Set $x\ k + 1 = x\ k + t\ k\ d\ k$.

Step 5. Take $k := k + 1$ and go to Step 1.

Denote

$$\Phi\ t = f\ x\ k + td\ k\ .$$

1) Explain linear least square filters?



Ans:

Relationship to the Wiener filter**[edit]**

The realization of the causal Wiener filter looks a lot like the solution to the least squares estimate, except in the signal processing domain. The least squares solution,

for input matrix and output vector is

The FIR least mean squares filter is related to the Wiener filter, but minimizing the error criterion of the former does not rely on cross-correlations or auto-correlations. Its solution converges to the Wiener filter solution. Most linear adaptive filtering problems can be formulated using the block diagram above. That is, an unknown system is to be identified and the adaptive filter attempts to adapt the filter to make it as close as possible to , while using only observable signals , and ; but , and are not directly observable. Its solution is closely related to the Wiener filter.

Convergence theorem:

The perceptron convergence theorem basically states that the perceptron learning algorithm converges in finite number of steps, given a linearly separable dataset. More precisely, if for each data point x, $\|\mathbf{x}\| < R$ where R is certain constant number, $\gamma = (\theta^{\ast})^{T} \mathbf{x_c}$ where $\mathbf{x_c}$ is the data point that is the closest to the linear separate hyperplane. It should be noted that mathematically $\frac{\gamma}{\|\theta^{\ast}\|^2}$ is the distance d of the closest datapoint to the linear separate hyperplane (it could be negative). The number of steps is bounded by $\frac{R^{2}\|\theta^{\ast}\|^{2}}{\gamma^2}$ or $\frac{R^{2}}{d^2}$.

In some materials, for simplicity, someone added assumption without generality that the weight of separate hyperplane is a unit vector ($\|\theta^{\ast}\|^{2} = 1$) and one could claim that the physical meaning of γ is the distance of the closest data point to the linear separate hyperplane. However, sometimes people ignored this assumption and claim γ is the distance of the closest data point to the linear separate hyperplane. That was wrong.

Learning Curves to Diagnose Machine Learning

Model Performance

by **Jason Brownlee** on February 27, 2019 in **Deep Learning Performance**
Tweet Tweet **Share**
Last Updated on August 6, 2019

A learning curve is a plot of model learning performance over experience or time.

Learning curves are a widely used diagnostic tool in machine learning for algorithms that learn from a training dataset incrementally. The model can be evaluated on the training dataset and on a hold out validation dataset after each update during training and plots of the measured performance can created to show learning curves.

Reviewing learning curves of models during training can be used to diagnose problems with learning, such as an underfit or overfit model, as well as whether the training and validation datasets are suitably representative.

In this post, you will discover learning curves and how they can be used to diagnose the learning and generalization behavior of machine learning models, with example plots showing common learning problems.

After reading this post, you will know:

- Learning curves are plots that show changes in learning performance over time in terms of experience.
- Learning curves of model performance on the train and validation datasets can be used to diagnose an underfit, overfit, or well-fit model.
- Learning curves of model performance can be used to diagnose whether the train or validation datasets are not relatively representative of the problem domain.

Kick-start your project with my new book Better Deep Learning, including *step-by-step tutorials* and the *Python source code* files for all examples.
Let's get started.

Overview

This tutorial is divided into three parts; they are:

1. Learning Curves
2. Diagnosing Model Behavior
3. Diagnosing Unrepresentative Datasets

Learning Curves in Machine Learning

Generally, a learning curve is a plot that shows time or experience on the x-axis and learning or improvement on the y-axis.

Learning curves (LCs) are deemed effective tools for monitoring the performance of workers exposed to a new task. LCs provide a mathematical representation of the learning process that takes place as task repetition occurs.

— Learning curve models and applications: Literature review and research directions, 2011.

For example, if you were learning a musical instrument, your skill on the instrument could be evaluated and assigned a numerical score each week for one year. A plot of the scores over the 52 weeks is a learning curve and would show how your learning of the instrument has changed over time.

- **Learning Curve**: Line plot of learning (y-axis) over experience (x-axis).

Learning curves are widely used in machine learning for algorithms that learn (optimize their internal parameters) incrementally over time, such as deep learning neural networks.

The metric used to evaluate learning could be maximizing, meaning that better scores (larger numbers) indicate more learning. An example would be classification accuracy.

It is more common to use a score that is minimizing, such as loss or error whereby better scores (smaller numbers) indicate more learning and a value of 0.0 indicates that the training dataset was learned perfectly and no mistakes were made.

During the training of a machine learning model, the current state of the model at each step of the training algorithm can be evaluated. It can be evaluated on the training dataset to give an idea of how well the model is "*learning*." It can also be evaluated on a hold-out validation dataset that is not part of the training dataset. Evaluation on the validation dataset gives an idea of how well the model is "*generalizing*."

- **Train Learning Curve**: Learning curve calculated from the training dataset that gives an idea of how well the model is learning.
- **Validation Learning Curve**: Learning curve calculated from a hold-out validation dataset that gives an idea of how well the model is generalizing.

It is common to create dual learning curves for a machine learning model during training on both the training and validation datasets.

In some cases, it is also common to create learning curves for multiple metrics, such as in the case of classification predictive modeling problems, where the model may be optimized according to cross-entropy loss and model performance.

Short Questions

1.What is adaptive filtering problem?
2.Explain least mean square algorithm?
3. Explain learning curves?
4.Expalin linear least square filters?

Long Questions

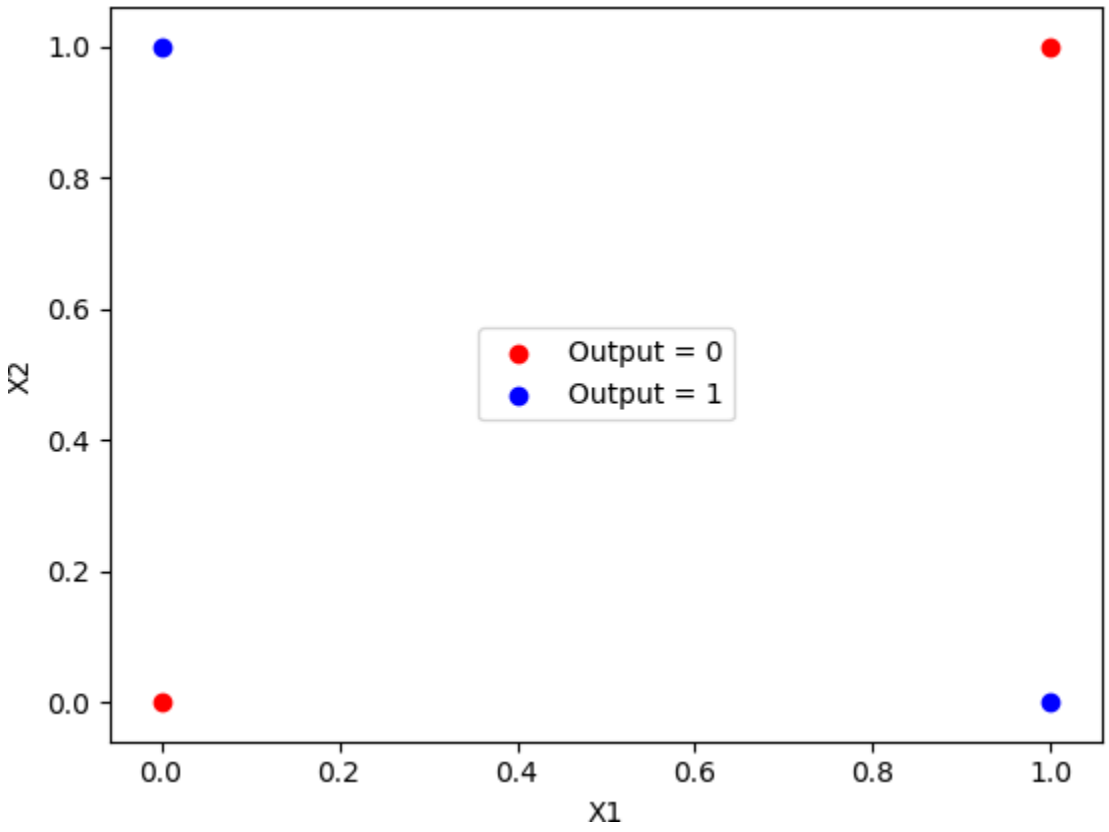
1.Explain learning rate annealing techniques?
2.Explain convergence theorem?
3.Explain Linear least square filters?
4. Explain between perception and Bayes classifier for a Guassian Environment?

UNIT-3

Back Propagation algorithm XOR problem:

Ans: Implementing logic gates using neural networks help understand the mathematical computation by which a neural network processes its inputs to arrive at a certain output. This neural network will deal with the XOR logic problem. An XOR (exclusive OR gate) is a digital logic gate that gives a true output only when both its inputs differ from each other. The truth table for an XOR gate is shown below:

The goal of the neural network is to classify the input patterns according to the above truth table. If the input patterns are plotted according to their outputs, it is seen that these points are not linearly separable. Hence the neural network has to be modeled to separate these input patterns using *decision planes.*



As mentioned before, the neural network needs to produce two different decision planes to linearly separate the input data based on the output patterns. This is achieved by using the concept of *hidden layers*. The neural network will consist of one input layer with two nodes (X1,X2); one hidden layer with two nodes (since two decision planes are needed); and one output layer with one node (Y). Hence, the neural network looks like this:

THE SIGMOID NEURON

To implement an XOR gate, I will be using a Sigmoid Neuron as nodes in the neural network. The characteristics of a Sigmoid Neuron are:

1. Can accept real values as input.

2. The value of the activation is equal to the weighted sum of its inputs
i.e. $\sum wi\ xi$

3. The output of the sigmoid neuron is a function of the sigmoid function, which is also known as a logistic regression function. The sigmoid function is a continuous function which outputs values between 0 and 1:

1) Explain output representation and decision rule?
Ans: A decision rule is a simple IF-THEN statement consisting of a condition (also called antecedent) and a prediction. For example: IF it rains today AND if it is April (condition), THEN it will rain tomorrow (prediction). A single decision rule or a combination of several rules can be used to make predictions.

Decision rules follow a general structure: IF the conditions are met THEN make a certain prediction. Decision rules are probably the most interpretable prediction models. Their IF-THEN structure semantically resembles natural language and the way we think, provided that the condition is built from intelligible features, the length of the condition is short (small number of feature=value pairs combined with an AND) and there are not too many rules. In programming, it is very natural to write IF-THEN rules. New in machine learning is that the decision rules are learned through an algorithm.
Imagine using an algorithm to learn decision rules for predicting the value of a house (low, medium or high). One decision rule learned by this model could be: If a house is bigger than 100 square meters and has a garden, then its value is high. More formally: IF size>100 AND garden=1 THEN value=high.
Let us break down the decision rule:

- size>100 is the first condition in the IF-part.
- garden=1 is the second condition in the IF-part.

- The two conditions are connected with an ‘AND’ to create a new condition. Both must be true for the rule to apply.
- The predicted outcome (THEN-part) is value=high.

A decision rule uses at least one feature=value statement in the condition, with no upper limit on how many more can be added with an ‘AND’. An exception is the default rule that has no explicit IF-part and that applies when no other rule applies, but more about this later.
The usefulness of a decision rule is usually summarized in two numbers: Support and accuracy.

Support or coverage of a rule: The percentage of instances to which the condition of a rule applies is called the support. Take for example the rule size=big AND location=good THEN value=high for predicting house values. Suppose 100 of 1000 houses are big and in a good location, then the support of the rule is 10%. The prediction (THEN-part) is not important for the calculation of support.
Accuracy or confidence of a rule: The accuracy of a rule is a measure of how accurate the rule is in predicting the correct class for the instances to which the condition of the rule applies. For example: Let us say of the 100 houses, where the rule size=big AND location=good THEN value=high applies, 85 have value=high, 14 have value=medium and 1 has value=low, then the accuracy of the rule is 85%. Usually there is a trade-off between accuracy and support: By adding more features to the condition, we can achieve higher accuracy, but lose support.

To create a good classifier for predicting the value of a house you might need to learn not only one rule, but maybe 10 or 20. Then things can get more complicated and you can run into one of the following problems:

- Rules can overlap: What if I want to predict the value of a house and two or more rules apply and they give me contradictory predictions?
- No rule applies: What if I want to predict the value of a house and none of the rules apply?

There are two main strategies for combining multiple rules: Decision lists (ordered) and decision sets (unordered). Both strategies imply different solutions to the problem of overlapping rules.

A **decision list** introduces an order to the decision rules. If the condition of the first rule is true for an instance, we use the prediction of the first rule. If not, we go to the next rule and check if it applies and so on. Decision lists solve the problem of overlapping rules by only returning the prediction of the first rule in the list that applies.

A **decision set** resembles a democracy of the rules, except that some rules might have a higher voting power. In a set, the rules are either mutually exclusive, or there is a strategy for resolving conflicts, such as majority voting, which may be weighted by the individual rule accuracies or other quality measures. Interpretability suffers potentially when several rules apply.

Both decision lists and sets can suffer from the problem that no rule applies to an instance. This can be resolved by introducing a default rule. The default rule is the rule that applies when no other rule applies. The prediction of the default rule is often the most frequent class of the data points which are not covered by other rules. If a set or list of rules covers the entire feature space, we call it exhaustive. By adding a default rule, a set or list automatically becomes exhaustive.

There are many ways to learn rules from data and this book is far from covering them all. This chapter shows you three of them. The algorithms are chosen to cover a wide range of general ideas for learning rules, so all three of them represent very different approaches.

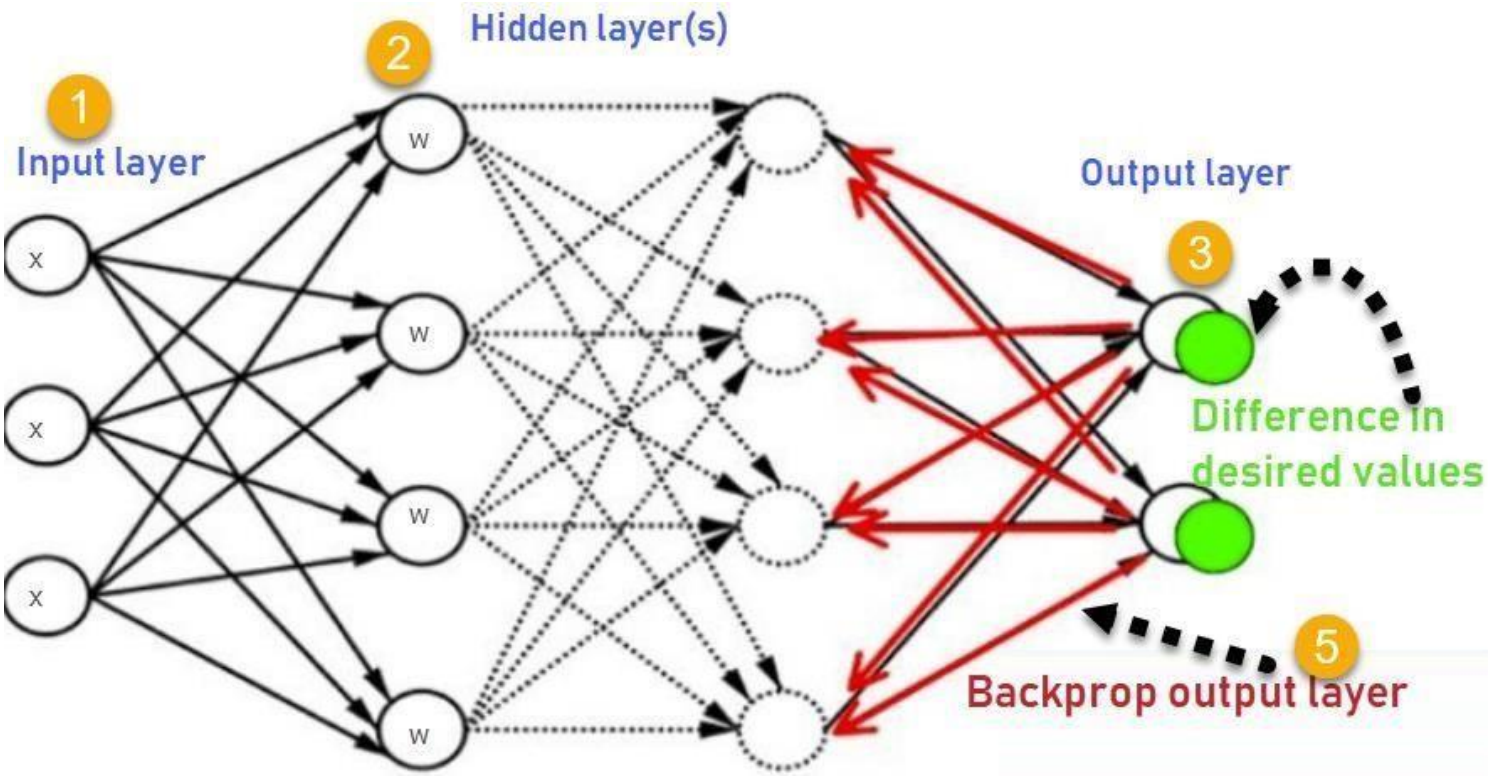
1. **OneR** learns rules from a single feature. OneR is characterized by its simplicity, interpretability and its use as a benchmark.
2. **Sequential covering** is a general procedure that iteratively learns rules and removes the data points that are covered by the new rule. This procedure is used by many rule learning algorithms.
3. **Bayesian Rule Lists** combine pre-mined frequent patterns into a decision list using Bayesian statistics. Using pre-mined patterns is a common approach used by many rule learning algorithms.

Computer Experiment:

Backpropagation is the essence of neural network training. It is the method of fine-tuning the weights of a neural network based on the error rate obtained in the previous epoch (i.e., iteration). Proper tuning of the weights allows you to reduce error rates and make the model reliable by increasing its generalization.
Backpropagation in neural network is a short form for “backward propagation of errors.” It is a standard method of training artificial neural networks. This method helps calculate the gradient of a loss function with respect to all the weights in the network.

The Back propagation algorithm in neural network computes the gradient of the loss function for a single weight by the chain rule. It efficiently computes one layer at a time, unlike a native direct computation. It computes the gradient, but it does not define how the gradient is used. It generalizes the computation in the delta rule.

Consider the following Back propagation neural network example diagram to understand:



1. Inputs X, arrive through the preconnected path

2. Input is modeled using real weights W. The weights are usually randomly selected.
3. Calculate the output for every neuron from the input layer, to the hidden layers, to the output layer.
4. Calculate the error in the outputs

$Error_B$= Actual Output – Desired Output

5. Travel back from the output layer to the hidden layer to adjust the weights such that the error is decreased.

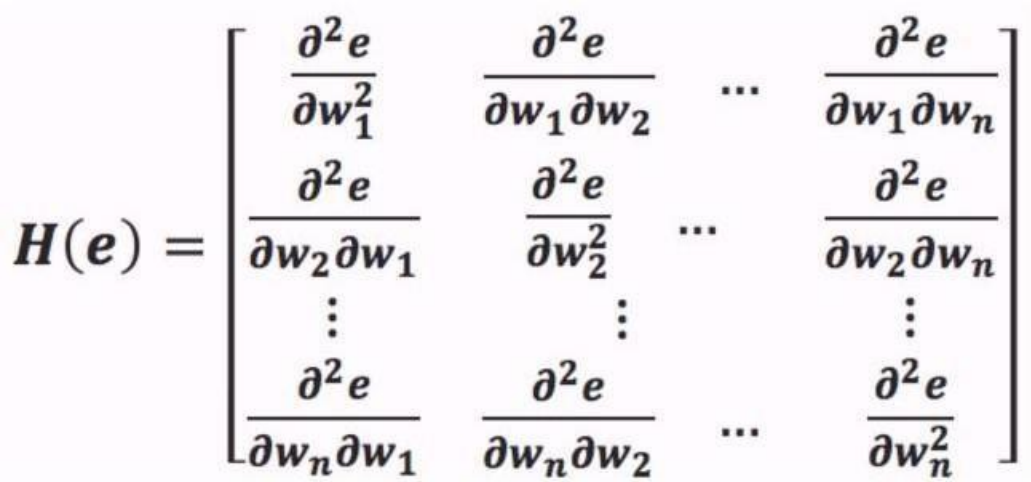
Keep repeating the process until the desired output is achieved

Hessian matrix:

A Hessian Matrix is square matrix of second-order partial derivatives of a scalar,which describes the local curvature of a multi-variable function.

Specifically in case of a Neural Network, the Hessian is a square matrix with the number of rows and columns equal to the total number of parameters in the Neural Network.

The Hessian for Neural Network looks as follows:



$$H(e) = \begin{bmatrix} \frac{\partial^2 e}{\partial w_1^2} & \frac{\partial^2 e}{\partial w_1 \partial w_2} & \cdots & \frac{\partial^2 e}{\partial w_1 \partial w_n} \\ \frac{\partial^2 e}{\partial w_2 \partial w_1} & \frac{\partial^2 e}{\partial w_2^2} & \cdots & \frac{\partial^2 e}{\partial w_2 \partial w_n} \\ \vdots & \vdots & & \vdots \\ \frac{\partial^2 e}{\partial w_n \partial w_1} & \frac{\partial^2 e}{\partial w_n \partial w_2} & \cdots & \frac{\partial^2 e}{\partial w_n^2} \end{bmatrix}$$

Hessian Matrix of a Neural Network

Why is Hessian based approach theoretically better than SGD?

Now, the second-order optimization using the Newton's method of iteratively finding the optimal 'x' is a clever hack for optimizing the error surface because, unlike SGD where you fit a plane at the point x_0 and then determine the step-wise jump, in second-order optimization, we find a tightly fitting quadratic curve at x_0 and directly find the minima of the curvature. This is supremely efficient and fast.

But !!! Empirically though, can you now imagine computing a Hessian for a network with millions of parameter? Of course it gets very in-efficient as the amount of storage and computation required to calculate the Hessian is of quadratic order as well. So, though in theory, this is awesome, in practice it sucks.

We need a hack for the hack ! And the answer seems to lie in Conjugate Gradients.

Conjugate Gradients, clever trick.

Actually, there are several quadratic approximation methods for a convex function. But Conjugate Gradient Method works quite well for a symmetric matrix, which are positive-definite. In fact, Conjugate Gradients are meant to work with very-large, sparse systems.

Note that a Hessian is symmetric around the diagonal, the parameters of a Neural Network are typically sparse, and the Hessian of a Neural Network is positive-definite (Meaning, it only has positive Eigen Values). Boy, are we in luck?

If you need a thorough introduction of Conjugate Gradient Methods, go through the paper titled "An Introduction to the Conjugate Gradient Method Without the Agonizing Pain" by Jonathan Richard Shewchuk. I find this quite thorough and useful. I would suggest that you study the paper in free-time to get a in-depth understanding of Conjugate Gradients.

The easiest way to explain the Conjugate Gradient (CG) is as follows:

- The CG Descent is applicable on any quadratic form.
- CG uses a step-size 'alpha' value similar to SGD but instead of a fixed alpha, we find the alpha through a line search algorithm.
- CG also needs a 'beta' a scalar value that helps find the next direction which is "conjugate" to the first direction.

You can check most of the hairy-math around arriving at a CG equation by the paper cited above. I shall directly jump to the section of the algorithm of the conjugate gradient:

For solving a equation ***Ax=b***, we can use the following algorithm:

$$\mathbf{r}_0 := \mathbf{b} - \mathbf{A}\mathbf{x}_0$$
$$\mathbf{p}_0 := \mathbf{r}_0$$
$$k := 0$$
repeat
$$\alpha_k := \frac{\mathbf{r}_k^\mathsf{T}\mathbf{r}_k}{\mathbf{p}_k^\mathsf{T}\mathbf{A}\mathbf{p}_k}$$
$$\mathbf{x}_{k+1} := \mathbf{x}_k + \alpha_k\mathbf{p}_k$$
$$\mathbf{r}_{k+1} := \mathbf{r}_k - \alpha_k\mathbf{A}\mathbf{p}_k$$
if r_{k+1} is sufficiently small then exit loop
$$\beta_k := \frac{\mathbf{r}_{k+1}^\mathsf{T}\mathbf{r}_{k+1}}{\mathbf{r}_k^\mathsf{T}\mathbf{r}_k}$$
$$\mathbf{p}_{k+1} := \mathbf{r}_{k+1} + \beta_k\mathbf{p}_k$$
$$k := k + 1$$
end repeat
The result is $\mathbf{x}_{k+1}$

image credit

- Here r_k is the residual value,
- p_k is the conjugate vector and,
- x_k+1 is iteratively updated with previous value x_k and the dot product of the step-size alpha_k and conjugate vector p_k.

Short Questions

1. Explain back propogation algorithm XOR problem?
2. Explain feature detection?
3. Explain cross validation?
4. Explain accelerated convergence?

Long Questions:

1. Explain back propagation and differentiation?
2. Explain Generalization?
3. Explain Network pruning techniques?
4. Explain supervised learning?

UNIT-4

Basic feature mapping models:

Kohonen Self-Organizing feature map (SOM) refers to a neural network, which is trained using competitive learning. Basic competitive learning implies that the competition process takes place before the cycle of learning. The competition process suggests that some criteria select a winning processing element. After the winning processing element is selected, its weight vector is adjusted according to the used learning law (Hecht Nielsen 1990).

The self-organizing map makes topologically ordered mappings between input data and processing elements of the map. Topological ordered implies that if two inputs are of similar characteristics, the most active processing elements answering to inputs that are located closed to each other on the map. The weight vectors of the processing elements are organized in ascending to descending order. $W_i < W_{i+1}$ for all values of i or W_{i+1} for all values of i (this definition is valid for one-dimensional self-organizing map only).

The self-organizing map is typically represented as a two-dimensional sheet of processing elements described in the figure given below. Each processing element has its own weight vector, and learning of SOM (self-organizing map) depends on the adaptation of these vectors. The processing elements of the network are made competitive in a self-organizing process, and specific criteria pick the winning processing element whose weights are updated. Generally, these criteria are used to limit the Euclidean distance between the input vector and the weight vector. SOM (self-organizing map) varies from basic competitive learning so that instead of adjusting only the weight vector of the winning processing element also weight vectors of neighboring processing elements are adjusted. First, the size of the neighborhood is largely making the rough ordering of SOM and size is diminished as time goes on. At last, only a winning processing element is adjusted, making the fine-

tuning of SOM possible. The use of neighborhood makes topologically ordering procedure possible, and together with competitive learning makes process non-linear.

It is discovered by Finnish professor and researcher Dr. Teuvo Kohonen in 1982. The self-organizing map refers to an unsupervised learning model proposed for applications in which maintaining a topology between input and output spaces. The notable attribute of this algorithm is that the input vectors that are close and similar in high dimensional space are also mapped to close by nodes in the 2D space. It is fundamentally a method for dimensionality reduction, as it maps high-dimension inputs to a low dimensional discretized representation and preserves the basic structure of its input space.

All the entire learning process occurs without supervision because the nodes are self-organizing. They are also known as feature maps, as they are basically retraining the features of the input data, and simply grouping themselves as indicated by the similarity between each other. It has practical value for visualizing complex or huge quantities of high dimensional data and showing the relationship between them into a low, usually two-dimensional field to check whether the given unlabeled data have any structure to it.

A self-Organizing Map (SOM) varies from typical artificial neural networks (ANNs) both in its architecture and algorithmic properties. Its structure consists of a single layer linear 2D grid of neurons, rather than a series of layers. All the nodes on this lattice are associated directly to the input vector, but not to each other. It means the nodes don't know the values of their neighbors, and only update the weight of their associations as a function of the given input. The grid itself is the map that coordinates itself at each iteration as a function of the input data. As such, after clustering, each node has its own coordinate (i.j), which enables one to calculate Euclidean distance between two nodes by means of the Pythagoras theorem.

A Self-Organizing Map utilizes competitive learning instead of error-correction learning, to modify its weights. It implies that only an individual node is activated at each cycle in which the features of an occurrence of the input vector are introduced to the neural network, as all nodes compete for the privilege to respond to the input.

The selected node- the Best Matching Unit (BMU) is selected according to the similarity between the current input values and all the other nodes in the network. The node with the fractional Euclidean difference between the input vector, all nodes, and its neighboring nodes is selected and within a specific radius, to have their position slightly adjusted to coordinate the input vector. By experiencing all the nodes present on the grid, the whole grid eventually matches the entire input dataset with connected nodes gathered towards.

Self-organization algorithm:

Algorithm:

Step:1

Each node weight w_ij initialize to a random value.

Step:2

Choose a random input vector x_k.

Step:3

Repeat steps 4 and 5 for all nodes on the map.

Step:4

Calculate the Euclidean distance between weight vector w_{ij} and the input vector x(t) connected with the first node, where t, i, j =0.

Step:5

Track the node that generates the smallest distance t.

Step:6

Calculate the overall Best Matching Unit (BMU). It means the node with the smallest distance from all calculated ones.

Step:7

Discover topological neighborhood *βij(t) its radius σ(t)* of BMU in Kohonen Map.

Step:8

Repeat for all nodes in the BMU neighborhood: Update the weight vector w_ij of the first node in the neighborhood of the BMU by including a fraction of the difference between the input vector x(t) and the weight w(t) of the neuron.

Step:9

Repeat the complete iteration until reaching the selected iteration limit t=n.

Here, step 1 represents initialization phase, while step 2 to 9 represents the training phase.

Where;

t = current iteration.

i = row coordinate of the nodes grid.

J = column coordinate of the nodes grid.

W= weight vector

w_ij = association weight between the nodes i,j in the grid.

X = input vector

X(t)= the input vector instance at iteration t

β_ij = the neighborhood function, decreasing and representing node i,j distance from the BMU.

$\sigma(t)$ = The radius of the neighborhood function, which calculates how far neighbor nodes are examined in the 2D grid when updating vectors. It gradually decreases over time.

Computer Simulations:

The use of a computer to represent the dynamic responsesof one system by the behaviour of another system modeled after it. A simulation usesa mathematical description, or model, of a real system in the form of a computer program. This model is composed of equations that duplicate the functional relationships within the real system. When the program is run, the resulting mathematical dynamics form an analog of the behaviour of the real system, with the results presented in the form of data. A simulation can also take the form of a computer-graphics image that represents dynamic processes in an animated sequence.

Computer simulations are used to study the dynamic behaviour of objects or systems in response to conditions that cannot be easily or safely applied in real life. For example, a nuclear blast can be described by a mathematical model that incorporates such variables as heat, velocity, and radioactive emissions. Additional mathematical equations can then be used to adjust the model to changes in certain variables, such as the amount of fissionable material that produced the blast. Simulations are especially useful in enabling observers to measure and predict how the functioning of an entire system may be affected by altering individual components within that system.

The simpler simulations performed by personal computers consist mainly of business models and geometric models. The former includes spreadsheet, financial, and statistical software programs that are used in business analysis and planning. Geometric models are used for numerous applications that require simple mathematical modeling of objects, such as buildings, industrial parts, and the molecular structures of chemicals. More advanced simulations, such as those that emulate weather patterns or the behaviour of macroeconomic systems, are usually performed on powerful workstations or supercomputers. In engineering, computer models of newly designed structures undergo simulated tests to determine their responses to stress and other physical variables. Simulations of river systems can be manipulated to determine the potential effects of dams and irrigation networks

before any actual construction has taken place. Other examples of computer simulations include estimating the competitive responses of companies in a particular market and reproducing the movement and flight of space vehicles.

Contexed vector

Context Vectors are fixed-length vector representations useful for document retrieval and word sense disambiguation. Context vectors were motivated by four goals:

1. Capture “similarity of use” among words (“car” is similar to “auto”, but not similar to “hippopotamus”).
2. Quickly find constituent objects (eg., documents that contain specified words).
3. Generate context vectors automatically from an unlabeled corpus.
4. Use context vectors as input to standard learning algorithms.

Context Vectors lack, however, a natural way to represent syntax, discourse, or logic. Accommodating all these capabilities into a “Grand Unified Representation” is, we maintain, a prerequisite for solving the most difficult problems in Artificial Intelligence, including natural language understanding.

Short Questions:

1.Explain two basic feature mapping models?
2.Explain properties of a feature map?
3.Explain computer simulations?
4.Expalin contexed Maps?

Long Questions:

1.Explain Self-organization map?
2.Explain SOM algorithm?
3. Explain about learning vector quantization?
4.Explain about adaptive patter classification?

UNIT-5

Dynamic systems:

Dynamical systems see widespread use in natural sciences like physics, biology, chemistry, as well as engineering disciplines such as circuit analysis, computational fluid dynamics, and control. For simple systems, the differential equations governing the dynamics can be derived by applying fundamental physical laws. However, for more complex systems, this approach becomes exceedingly difficult. Data-driven modeling is an alternative paradigm that seeks to learn an approximation of the dynamics of a system using observations of the true system. In recent years, there has been an increased interest in data-driven modeling techniques, in particular neural networks have proven to provide an effective framework for solving a wide range of tasks. This paper provides a survey of the different ways to construct models of dynamical systems using neural networks. In addition to the basic overview, we review the related literature and outline the most significant challenges from numerical simulations that this modeling paradigm must overcome. Based on the reviewed literature and identified challenges, we provide a discussion on promising research areas.

Equilibrium states:

Head-direction cells have been found in several areas in the mammalian brains. The firing rate of an ideal head-direction cell reaches its peak value only when the animal's head points in a specific direction, and this preferred direction stays the same regardless of spatial location. In this paper we combine mathematical analytical techniques and numerical simulations to fully analyze the equilibrium states of a generic ring attractor network, which is a widely used modeling framework for the head-direction system. Under specific conditions, all solutions of the ring network are bounded, and there exists a Lyapunov function that guarantees the stability of the network for any given inputs, which may come from multiple sources in the biologicalsystem, including self-motion information for inertially based updating and landmark information for calibration. We focus on the first few terms of the Fourier series of thering network to explicitly solve for all possible equilibrium states, followed by a stability analysis based on small perturbations. In particular, these equilibrium states include the standard single-peaked activity pattern as well as double-peaked activity pattern, whose

existence is unknown but has testable experimental implications. To our surprise, we have also found an asymmetric equilibrium activity profile even whenthe network connectivity is strictly symmetric. Finally we examine how these differentequilibrium solutions depend on the network parameters and obtain the phase diagrams in the parameter space of the ring network.

Introduction

Head-direction cells were first reported in several brain areas related to the limbic system in the rodents (Taube, 2007) and later in other mammalian species such as monkeys (Robertson et al., 1999) and bats (Finkelstein et al., 2015). A stereotypical head-direction cell increases its firing rate when the animal's head is facing in a specific direction in a world-centered coordinate system regardless of the animal's spatial location, and the firing rate decreases to its baseline level as the animal's head turns away from the preferred direction (Taube et al., 1990). It has been proposed that the head-direction cells may form a ring network that allows an activity bump to be self-sustained by attractor dynamics, and the peak position of the activity bump is updated by self-motion information and calibrated by learned landmarks (Skaggs et al., 1995; Redish et al., 1996; Zhang, 1996). Multiple versions of the ring network have been studied for the head-direction cells (Goodridge and Touretzky, 2000; Arleo and Gerstner, 2001; Sharp et al., 2001; Stringer et al., 2002; Xie et al., 2002; Song andWang, 2005) as well as for a variety of applications beyond the original head-directionsystem (Ben-Yishai et al., 1995; Pouget et al., 1998; Hahnloser et al., 2000; Kakaria and de Bivort, 2017; Zhang et al., 2019).

head-direction cells, attractor networks have been used as a general theoretical framework for modeling other types of spatial cells in the hippocampus and related systems (Knierim and Zhang, 2012).

The equilibrium state of the head-direction ring network is often visualized as a single bump of activity whose peak position corresponds to the animal's current heading direction (Figure 1, top and middle rows). While this picture is compelling and highly intuitive, it is not the only theoretical possibility for explaining the experimental data. For instance, imagine that the ring network can sustain two activity bumps instead of one (Figure 1, bottom row), then if one records from an individual cell in the ring, one would still find a head-direction cell with a perfectly normal, single-peaked tuning curve, assuming that the activity bumps now rotates at half of the speed as the single activity bump. Indeed, if we focus on a single cell corresponding to north, we see that

in both situations, the cell fires at maximal rate only when the animal is facing north (*N*).

Figure 1

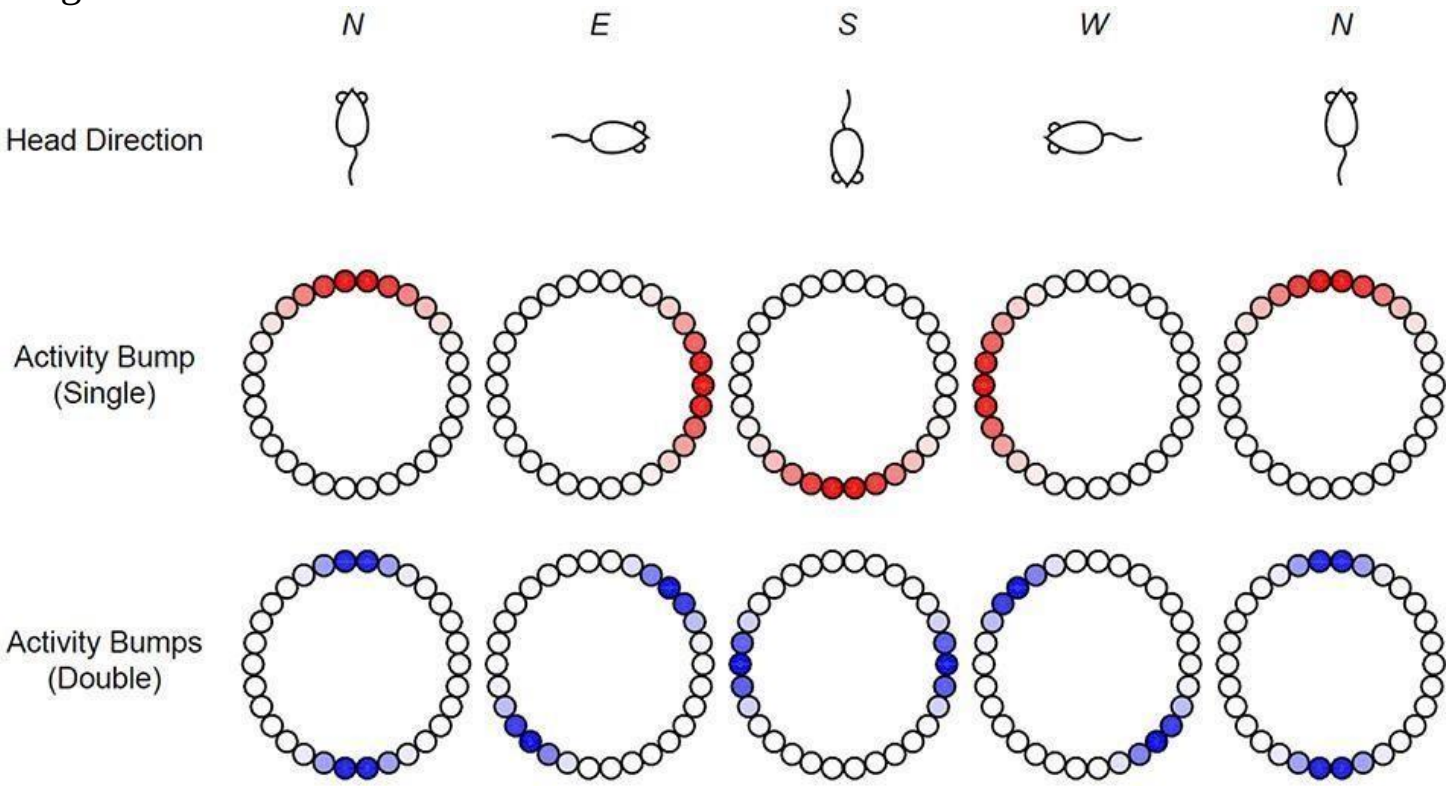


FIGURE 1. Head-direction ring network. In the classic view, the head direction of a rat (TOP) is represented by the peak location of the activity bump (MIDDLE: red shades) in a ring of head-direction cells. An alternative possibility is a ring network that allows two stable activity bumps that rotate at half of the speed (BOTTOM).

significance has motivated us to examine the equilibrium states in the ring network model in greater depth.

This paper is aimed at a thorough analysis of the equilibrium states in the ring network, with a focus on the exact conditions for the existence of activity patterns with multiple peaks. We will use the simple continuous ring network to simplify the mathematical analysis. The rotational symmetry of the system allows Fourier analysis to be used effectively. We strive to derive the exact analytical conditions whenever possible, and the analytical treatments are complemented by systematic numerical simulations. Once the exact expressions of all different kinds of equilibrium states are obtained, we will employ small perturbations and eigen-analysis of the linearized system to determine the stabilities of these equilibrium states. We will examine the dependence of various equilibrium states on the network parameters and summarize the results by the phase diagrams. Our analysis may provide a necessary step for extending the application and analysis of the ring network beyond the classic single-peaked condition.

1. Materials and Methods

We consider a continuous formulation of the head-direction system which has a continuous ring structure (Zhang, 1996). Such continuous formulation has a long history in neural modeling (Wilson and Cowan, 1972; Amari, 1977; Bressloff, 2012). The standard simplified time evolution continuous dynamics is governed by the equation

$$\tau\partial u(\theta,t)\partial t=-u(\theta,t)+w(\theta,t)*g(u(\theta,t))+I(\theta,t),\quad \theta\in[0,2\pi),\quad (1)$$

$\tau\partial u(\theta,t)\partial t=-u(\theta,t)+w(\theta,t)*g(u(\theta,t))+I(\theta,t),\quad \theta\in[0,2\pi),\quad (1)$

where the convolution is defined by

$$w(\theta,t)*g(u(\theta,t))=12\pi\int 2\pi 0 w(\theta-\varphi,t)g(u(\varphi,t))d\varphi.\quad (2)$$

$w(\theta,t)*g(u(\theta,t))=12\pi\int 02\pi w(\theta-\varphi,t)g(u(\varphi,t))d\varphi.\quad (2)$

In this system, $u(\theta, t)$ represents the state of voltage of a unit with θ as its preferred direction, $w(\theta - \varphi, t)$ represents the synaptic weight between units with $\theta - \varphi$ being the

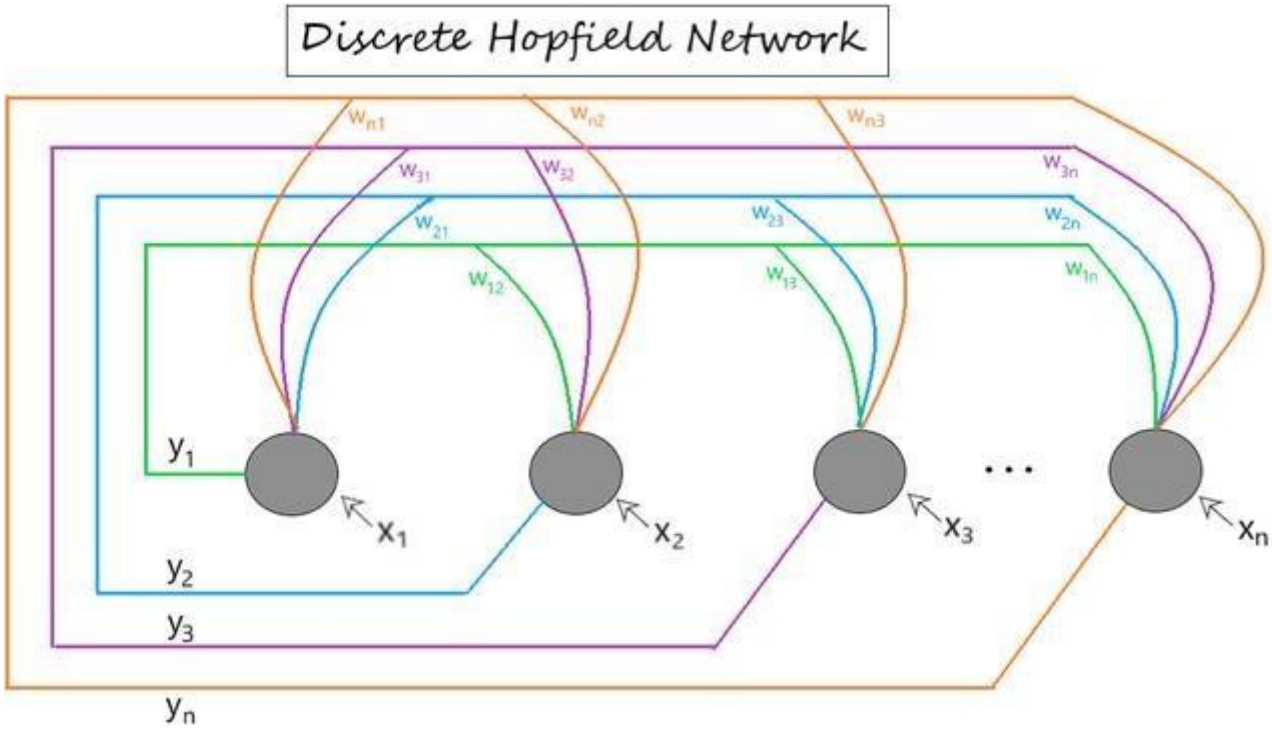
difference of their preferred directions, $g(u)$ is a monotone increasing and sigmoid

FIGURE 1. Head-direction ring network. In the classic view, the head direction of a rat (TOP) is represented by the peak location of the activity bump (MIDDLE: red shades) in a ring of head-direction cells. An alternative possibility is a ring network that allows two stable activity bumps that rotate at half of the speed (BOTTOM).

significance has motivated us to examine the equilibrium states in the ring network model in greater depth.

This paper is aimed at a thorough analysis of the equilibrium states in the ring network, with a focus on the exact conditions for the existence of activity patterns with multiple peaks. We will use the simple continuous ring network to simplify the mathematical analysis. The rotational symmetry of the system allows Fourier analysis to be used effectively. We strive to derive the exact analytical conditions whenever possible, and the analytical treatments are complemented by systematic numerical simulations. Once the exact expressions of all different kinds of equilibrium states are obtained, we will employ small perturbations and eigen-analysis of the linearized system to determine the stabilities of these equilibrium states. We will examine the dependence of various equilibrium states on the network parameters and summarize the results by the phase diagrams. Our analysis may provide a necessary step for extending the application and analysis of the ring network beyond the classic single-peaked condition.

Despite the functional equivalence, the structures of the two ring networks are different. More specifically, unlike the standard single-peaked network, the double- peaked network has strong connections between cells in opposing directions althoughthey are not as strong as the connections between neighboring cells. In the consideration above, we assume that the rotation speed is halved in the double-peakednetwork. If the rotation speed is kept the same, then the network should generate tuning curves with two peaks that are 180° apart. In fact, double-peaked head- direction tuning curves have been reported in the retrosplenial cortex (Jacob et al., 2017), although the phenomenon could be attributed to a single preferred direction switching back and forth in time rather than implying a truly double-peaked activity pattern (Page and Jeffery, 2018). The consideration above can be generalized readily to activity patterns with three or more peaks. The possible existence of multi-peaked activities in the head-direction system together with their potential functional



Training Algorithm

For storing a set of input patterns S(p) [p = 1 to P], where **$S(p) = S_1(p) \ldots S_i(p) \ldots S_n(p)$**, the weight matrix is given by:

- # For binary patterns

  - **For bipolar patterns**

(i.e. weights here have no self connection)

Steps Involved

Step 1 - Initialize weights **(w_{ij})** to store patterns **(using training algorithm)**.
Step 2 - For each input vector y_i, perform **steps 3-7**.
Step 3 - Make initial activators of the network equal to the external input vector x.

Step 4 - For each vector y_i, perform **steps 5-7**.
Step 5 - Calculate the total input of the network **y_{in}** using the equation given below.

Step 6 - Apply activation over the total input to calculate the output as per the equation given below:

(where θ_i (threshold) and is normally taken as 0)

Step 7 - Now feedback the obtained output y_i to all other units. Thus, the activation vectors are updated.

Step 8 - Test the network for convergence.

Example Problem

Consider the following problem. We are required to create Discrete Hopfield Network with bipolar representation of input vector as [1 1 1 -1] or [1 1 1 0] (in case of binary representation) is stored in the network. Test the hopfield network with missing entries in the first and second component of the stored vector (i.e. [0 0 1 0]).

Step by Step Solution

Step 1 - given input vector, x = [1 1 1 -1] (bipolar) and we initialize the weight matrix **(w_{ij})** asand weight matrix with no self connection

Step 3 - As per the question, input vector x with missing entries, x = [0 0 1 0] (**[x1 x2 x3 x4]**) (binary)

- Make y_i = x = [0 0 1 0] **([y1 y2 y3 y4])**

Step 4 - Choosing unit y_i **(order doesn't matter)** for updating its activation.

- Take the i^{th} column of the weight matrix for calculation.

(we will do the next steps for all values of yi and check if there is convergence or not)

Short Questions:

1. Explain about dynamical systems?
2. Explain about stability of equilibrium states?
3. Explain about neurodynamical models?

Long Questions:

1.Explain different models of neurodynamical?
2.Explain manipulation of Hopfield models?
3.Explain dynamical states?

Question Papers for Sample:

S.V.U. College of Commerce Management and Computer Science: Tirupati
Department of Computer Science

Time:2hours Internal Examination-I MaxMarks:30

Section-A (5*2=10 Marks)

Answer any five questions

1. What is Neural Network?
2. Write about Network Architecture? 3.Write about learning process and it's types?
4. What is statistical nature of the learning process? 5.What are unconstrained organization techniques? 6.Explain about least mean square algorithm?
7. What are Learning rate annealing techniques?
8. Explain about convergence theorem?

Section-B (2*10=20 Marks)

Answer any one from each unit

Unit-1

9. Explain briefly about Learning Process?

Or

10. What are models of Neuron & Artificial intelligence and Neural Networks?

UNIT-2

11. Write about Single layer perceptons?

Or

12. Explain about relation between perception and Bayes classifier for a Gaussian Environment?

S.V.U. College of Commerce Management and Computer Science: Tirupati
Department of Computer Science

Time:2hours Internal Examination-II MaxMarks:30

Section-A (5*2=10 Marks)

Answer any five questions

1.What is Output representation and decision rules?2.Write about feature detection ?
3.What are Network pruning Techniques? 4.Write about two basic feature mapping models?5.Explain about SOM algorithm ?
6.What is Hierarchical vector quantifier ?7.What is Neuro Dynamics ?
8. What are HOPFIELD models?

Section-B (2*10=20)

Answer any one from each unit

UNIT-3

9. Explain briefly about multilayer perceptron ?

Or

10. What is Back propagation and write about hessian matrix?

UNIT-4

11. Explain about self organization models?

Or

UNIT-5

12. Explain briefly about HOPFIELD models?

MASTER OF COMPUTER APPLICATIONS DEGREE EXAMINATION
SECOND SEMESTER
Paper MCA 205C: Neural Networks
(Under C.B.S.C Revised Regulations w.e.f.2021-2023)
(Common papers to University and all affiliated Colleges)

Time:3hours Marks:70

PART-A

Answer any five from the following (5*4=20 Marks)
Each question Carries four marks

1. What is Neural Networks?
2. Write about learning process and it's types? 3.Explain about least mean square algorithm? 4.Explain about convergence theorem?
5. Write about feature detection?
6. Write about two basic feature mapping models?7.Explain about feature detection?
8.Explain about SOM algorithm?9.What is Neuro Dynamics?
10. What are HOPFIELD models?

PART-B (5*12=60)

ANSWER ANY ONE QUESTION FROM EACH UNIT

Each question Carries 12 marks

UNIT-1

11. What are the models of Neurons and explain about artificial intelligence & NeuralNetworks?

Or

12. Explain briefly about Learning process and it's types?

UNIT-2

13. Explain the relation between perception and Bayes classifier for a Gaussian Environment?

Or

14. Explain briefly about single layer perceptions?

UNIT-3

15. What is Back propagation and write about Hessian Matrix?

Or

16. Explain briefly about multilayer perceptron?

UNIT-4

17. Write about Self-Organization Maps?

Or

18. Write about Hierarchical vector quantifier and contexed maps?

UNIT-5

19. Explain briefly about HOPFIELD models?

Or

20. Explain briefly about Neuro Dynamics and it's various methods?

`