e-PGPathshala

Subject : Computer Science

Paper: Machine Learning

Module: Neural Networks – I

Module No: CS/ML/19

Quadrant I – e-text

Welcome to the e-PG Pathshala Lecture Series on Machine Learning. In this and the next modules we will be discussing another important and interesting machine learning technique – Neural networks.

Learning Objectives:

The learning objectives of this module are as follows:

- To understand the Artificial Neural Network Classification Methods
- To explain the working of ANN, types of ANN and its applications
- To discuss about back propagation and feed forward networks.

19.1 Introduction

Neural networks are considered as models of biological neural networks. The main aim of neural networks was to design artificial systems that tried to mimic the intelligent and complex behaviour of the human brain. Another objective was the possibility of enhancing our own understanding of the human brain. Most neural networks are based on some sort of training rule. In other words, neural networks learn from examples similar to the way children learn to recognize objects by observing examples of the objects. They also need to have the capability of generalization so that they can predict about unseen data beyond the training data.

19.1.1 **Real Neurons**

In order to understand the working of neural networks it is necessary to have an understanding about the components of the real neuron of the human brain. We will describe the Cell structure of the neuron (Figure 19.1). It consists of basically four components. The first component namely the cell body or SOMA is responsible for processing the inputs. The second component consists of a host of fine structures called dendrites that collects the input signals and passes it to the third component which is the the axon. The neuron sends out electrical signals through the axon which splits it into thousands of branches. Finally we

have the synaptic terminals which are responsible for the electro-chemical contacts between the neurons and inhibit or excite activity of the neuron.

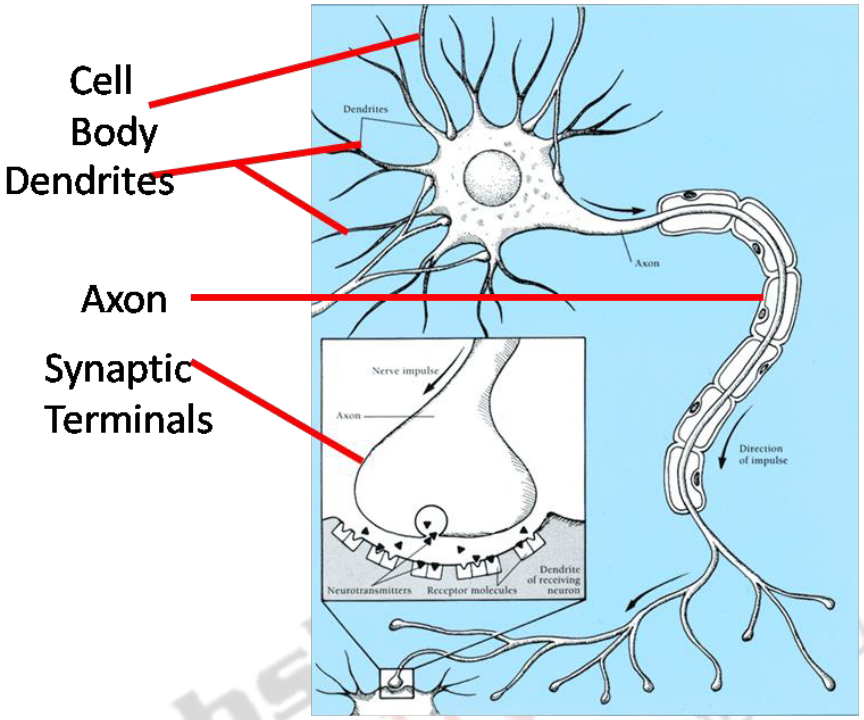

Figure 19.1 The Cell Structure

19.1.2 The Neuron Metaphor

The main purpose of neurons is to receive, analyze and transmit further information $x_1, x_2, x_3, \ldots\ldots x_N$ in the form of signals (electric pulses). The multiple inputs are multiplied by weights $w_{i1}, w_{i2}, \ldots\ldots w_{iN}$ and summed to get an input u_i .This u_i is given as input to the function to get the output y_i (Figure 19.2) . There is an additional input unit that corresponds to an nonexisting attribute $x_o = 1$. This is called the bias.

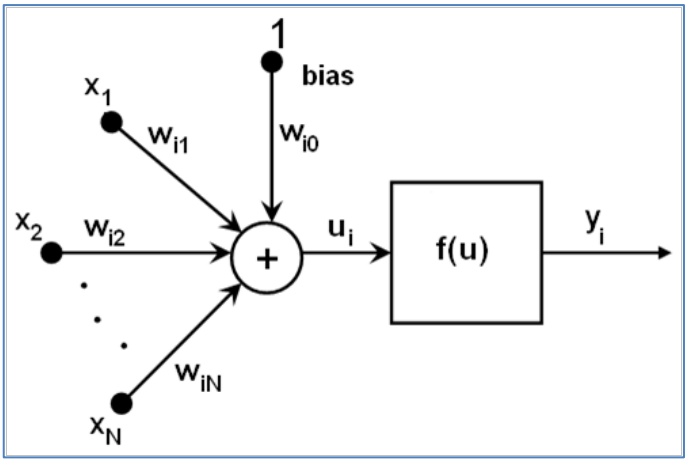

Figure 19.2 The Neuron Metaphor

19.2 Goals of Neural Computation

Before we go into the details of neural networks let us discuss some possible goals of neural computation. The first and possibly lofty goal is to understand how the brain actually works. As of now we know that it is a big, complex collection of cells. The second and practical goal is to understand a new style of computation that is inspired by neurons and their adaptive connections. Essentially this style of computation is different from sequential computation. This neural computation approach is good for cognitive processes such as computer vision. While sequential process is suitable for large scale computation, neural style of computation is not suited for such computation. Another goal is to carry out the learning for practical problems where the algorithms learnt are useful even they do not depict how the brain works.

19.3 Artificial Neural Network

Artificial neural networks or ANNs as they are called are built of densely interconnected set of simple units, each taking several real-valued inputs and producing single-valued output. ANN is characterized by its architecture, its training or learning algorithm and its activation functions. One motivation is to capture highly parallel computations on distributed processes. However most ANN software run only on sequential machines that emulate distributed processes.

19.4 Appropriate Problems for Neural Network Learning

Now let us discuss the type of problems that are suitable to be solved by ANNs. ANNs are well suited for problems that take noisy complex sensor data such as from cameras and microphones. Here the input instances are represented as attribute-value pair. The target output may be discrete-valued, real-valued, or a vector of several real or discrete-valued attributes. Training samples may not always be accurate and may contain errors. The long training time due to noisy complex data is generally acceptable. It is after the training phase, that fast evaluation of the target function is required. There is no need for humans to understand or interpret the learning of the target function.

19.5 Pros and Cons of Neural Networks

19.5.1 Pros

Basically neural networks are suited to problems that take continuous data especially in domains with little initial knowledge. Neural networks are easy to use and basically learns by example and need very little user domain-specific expertise. Neural networks have the ability to solve new kinds of problems that are difficult and sometimes impossible to explicitly define. ANNs can be used when a good functional model is not known. Neural computation provides

human characteristics to problem solving that are otherwise difficult to simulate. It is flexible and easy to maintain. Although training may take time, once trained it exhibits fast processing speed.

ANNs are generally robust and have the ability to cope with incomplete or fuzzy data. As ANNs consist of a large number of massively interconnected parallel processing units, operating on the same problem and are fast when parallel processors are used for implementation.

Cons

The main disadvantage of ANNs is that they do not produce an explicit model. ANNs do not provide explanation capabilities as the results are based on connection weights which usually do not have obvious interpretations. The main disadvantage of ANNs is that the solution is not interpretable, that is it acts as a "black box". The learning during the training phase is generally slow. For achieving good generalization many input data points may be required for training.

19.6 Linear Models

In a simple problem of linear regression we have the training data X = $\{x_1^k\}$, k=1,..,N with corresponding output Y = $\{y^k\}$, k=1,..,N and our job is to find the parameters that predict the output Y from the data X in a linear fashion that is $y^k \approx w_o + w_1 x_1^k$.

Let us first consider the general form of linear models for regression and classification as given below:

$$y(X, W) = f\{\sum_{j=1}^{M} w_j \, \emptyset_j(X)\}$$

where X is a D-dimensional vector $\phi_j(X)$ are fixed nonlinear basis functions such as Gaussian functions, or sigmoid basis functions, W are the weights we are trying to learn.

As we have already discussed for regression *f is identity function* (in the case of Linear Regression), while for classification *f is a nonlinear activation* function as specified in generalized linear regression). If the function f is sigmoid than the regression is called logistic regression and is as given below:

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

19.6.1 Extending Linear Models

The linear models that we have discussed are useful for discussing analytical and computational properties but however have limited applicability mainly due to the curse of dimensionality. For example the number of polynomial co-efficients that are needed, like finding means of Gaussians is difficult. Therefore we extend the scope by adapting basis functions ϕ_j to the parameters. This approach is suited for large scale problems.

Both SVMs and Neural Networks address this limitation. In the case of SVM a varying number of basis functions *M,* centred around training data points are used and a subset of these are selected during training. In the case of ANNs the number of basis functions *M is fixed* but ϕ_j have their own parameters $\{w_{ji}\}$ which are adapted appropriately during training.

$$y_k(X,W) = \sigma\{\sum_{j=1}^{M} w_{kj}^{(2)}\, h\left\{\sum_{i=1}^{D} w_{ji}^{(1)}\, x_i + w_{j0}^{(1)}\right\} + w_{k0}^{(2)}\}$$

19.7 SVM versus Neural Networks

Now let us compare SVM and ANNs. In the case of SVM, training involves non-linear optimization. The objective function is convex, but however the optimization is straightforward usually with a single minimum. The number of basis functions to be considered is much smaller than the number of training points. SVMs produce probabilistic outputs at the expense of non-convex optimization.

Neural Network on the other hand has only a fixed number of basis functions, and parametric forms. Multilayer perceptron uses layers of logistic regression models. ANNs also involve non-convex optimization during training (many minima). At the expense of longer periods of training, we get a more compact and faster model after the learning process is completed.

19.8 Origin of Neural Networks

The origin of neural networks was based on the desire to find information processing models of biological systems. The term covers a wide range of models and most them make exaggerated claims of biological plausibility. Moreover the biological realism imposes unnecessary constraints. However neural networks are efficient models for machine learning especially multilayer perceptrons. The neural network parameters can be obtained using the

maximum likelihood framework. It is basically a nonlinear optimization problem and requires evaluating derivative of log-likelihood function w.r.t network parameters which can be performed efficiently using error back propagation.

19.9 The Structure of the Neural Network

The neural network can be viewed as a generalization of linear models. Each ANN is composed of a collection of perceptrons grouped in layers. A typical and simple network consists of three layers namely input, intermediate (called the ***hidden layer***) and output. Several hidden layers can be placed between the input and output layers.

19.10 Types of Neural Network

The simplest and widely used neural network is the Feed-forward neural network where the information travels in one direction only (Figure 19.3). In other words there is no feedback or no cycles that is it is a Directed Acyclic graph. These types of neural networks are used for pattern recognition.

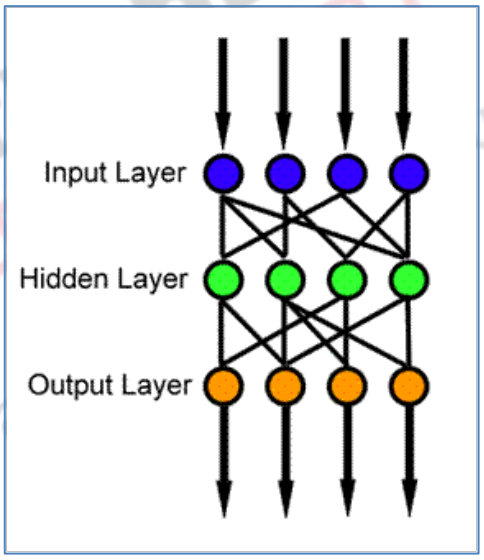

Figure 19.3 Feed Forward Network

In another type of neural networks, the signals travel in both directions and the networks become complicated and dynamic (Figure 19.4). These networks have context units that act as internal memory to store a part of the inputs and use this internal memory to process arbitrary sequences of inputs. Recurrent neural networks are usually used to learn temporal patterns

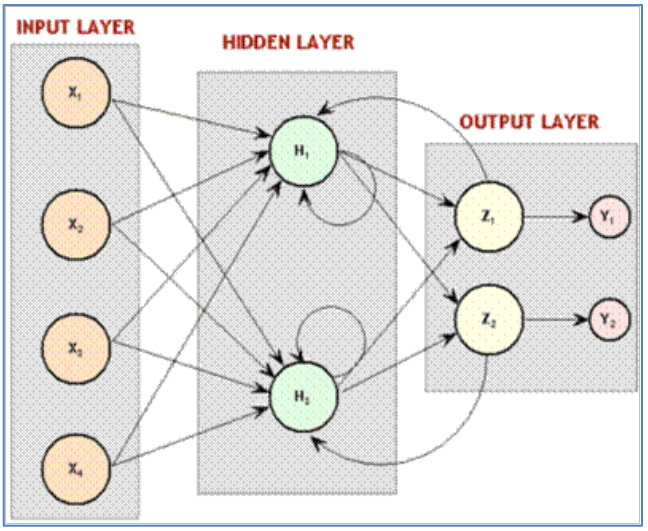

Figure 19.4 Recurrent Neural Network

19.11 Types of Neuron

Now let us see the different types of neurons that can be designed based on the function used. We show (Figure 19.5) only a few but potentially more functions can be designed. The simplest is the linear neuron where the function

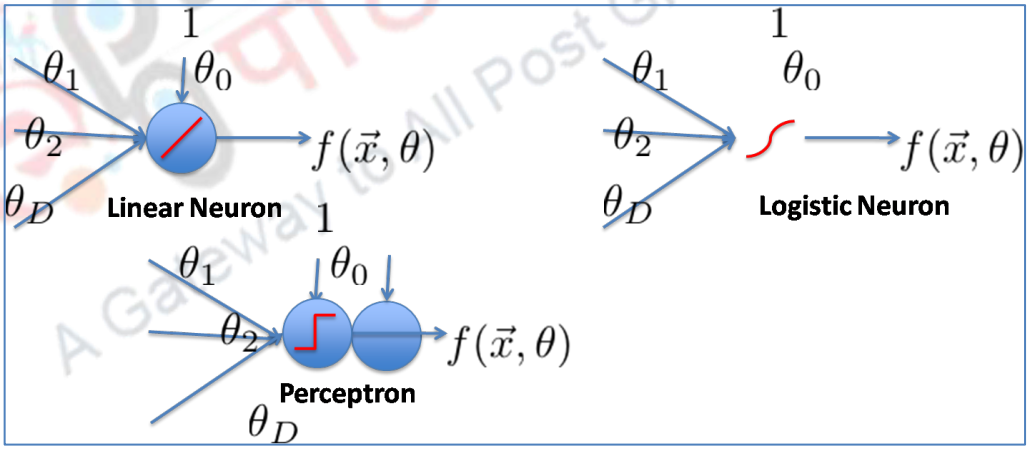

Figure 19.5 Type of Neurons

used is linear. Then we have the use of the sigmoid function resulting in the logistic neuron. We also show a simple step function resulting in the perceptron.

19.12 Feed Forward Network Functions

A neural network can also be represented like linear models but where the basis functions are generalized. The activation function used for regression is the identity function while for classification a nonlinear function like the sigmoid

function is used. The weight coefficients w_j are adjusted during training. Please note that there can be several activation functions.

The basis function $\phi_j(x)$ is a nonlinear function, such as tanh, and is a linear combination of D inputs and its parameters are also adjusted during training. Here f is the activation function and $\phi_j(x)$ is the basis function.

$$y(X,W) = f\{\sum_{j=1}^{M} w_j \, \emptyset_j(X)\}$$

19.12.1 The First Layer: Basis Functions

Let us assume that there are D input variables $x_1,..,\ x_D$ (Figure 19.6) where there are M linear combinations in the form

$$a_j = \sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)} \ where \ j = 1,\ldots..,M$$

Here the superscript (1) indicates that the parameters we are talking about are in the first layer of the network. The parameters w_{ji} are referred to as weights while w_{j0} with input 1 are referred to as biases. The quantities a_j which consider the combination of D weighted inputs and the bias are known as activations.

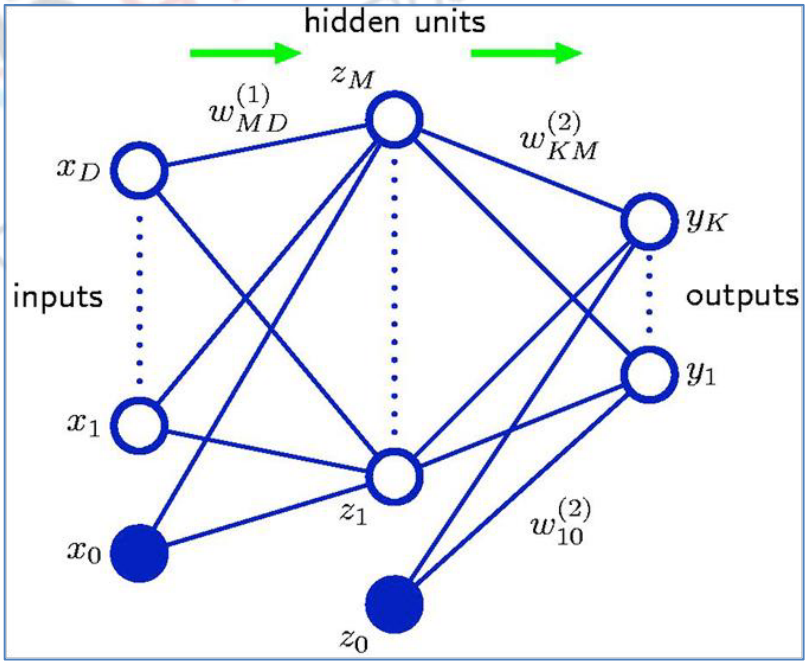

Figure 19.6 Structure of the Feed Forward Network

Each activation a_j is transformed using differentiable nonlinear activation functions $z_j=h(a_j)$. The z_j correspond to outputs of basis functions $\phi_j(x)$ or the

first layer of network or hidden units. The nonlinear functions h are chosen to be sigmoidal. Two examples of the above activation functions are

- logistic sigmoid [1/1+exp(*-a)]*
- *tanh [(e^a-e^{-a})/(e^a+e^{-a})]*

19.12.2 Activation Function

The activation function used depends on the nature of the data and the assumed distribution of target variables. A variety of activation functions can be used. Some of them are Sigmoidal (S-shaped), Logistic sigmoid [1/1+exp(*-a)*] (Used for binary classification), Hyperbolic tangent - tanh and the radial basis function given below:

$$z_j = \sum_i (x_i - w_{ji})^2$$

Softmax is another activation function often used for multi-class classification and is as given below:

$$\frac{\exp(a_k)}{\sum_j \exp(a_j)}$$

Th Identity function $y_k = a_k$ is generally useful for regression. It is possible to use different activation function in each unit.

19.12.3 Second Layer: Activation Functions

Values of z_j *are again linearly combined to give* output unit activations

$$a_k = \sum_{i=1}^{M} w_{ki}^{(2)} x_i + w_{k0}^{(2)} \; where \; k = 1, \ldots\ldots, K$$

Where *K is the total number of outputs*

Output unit activations are transformed by using appropriate activation function to give network outputs y_k (Figure 19.6)

19.12.4 Overall Network Function

Combining the stages of the overall function with sigmoidal output

$$y_k(X,W) = \sigma\{\sum_{j=1}^{M} w_{kj}^{(2)}\, h\left\{\sum_{i=1}^{D} w_{ji}^{(1)}\, x_i + w_{j0}^{(1)}\right\} + w_{k0}^{(2)}\}$$

where w is the set of all weights and bias parameters. Note presence of both σ and *h functions*

Thus a neural network is simply a set of nonlinear functions from input variables $\{x_i\}$ to output variables $\{y_k\}$ controlled by vector w of adjustable parameters.

19.12.5 Forward Propagation

The bias parameters can be absorbed into weight parameters by defining a new input variable x_0

$$y_k(X,W) = \sigma\{\sum_{j=1}^{M} w_{kj}^{(2)}\, h\left\{\sum_{i=1}^{D} w_{ji}^{(1)}\, x_i\right\}\}$$

The process of evaluation is forward propagation through network. The multilayer perceptron uses only continuous sigmoidal functions.

19.12.6 Feed Forward Networks

Therefore we connect together a number of these units to form the feed-forward network (DAG). Figure 19.6 shows a network with one layer of hidden unit and basically implements the following function

$$y_k(X,W) = \sigma\{\sum_{j=1}^{M} w_{kj}^{(2)}\, h\left\{\sum_{i=1}^{D} w_{ji}^{(1)}\, x_i + w_{j0}^{(1)}\right\} + w_{k0}^{(2)}\}$$

Therefore we have looked at generalized linear models of the form:

$$y(X,W) = f\{\sum_{j=1}^{M} w_j\, \emptyset_j(X)\}$$

For fixed non-linear basis function Ø(.), we now extend this model by allowing adaptive basis function and learning their parameters. In feed-forward networks (a.k.a. Multi-layer perceptrons) we let each basis function be another non-linear function of linear combination of the inputs.

$$\emptyset_j(x) = f(\sum_{j=1}^{M} \ldots..)$$

Starting with input x= (x_1,…,x_D), we construct linear combinations. The a_j are known as activations which pass through an activation function h(.) to get output z_j=h(a_j). The model of an individual neuron is summarized (Figure 19.7).

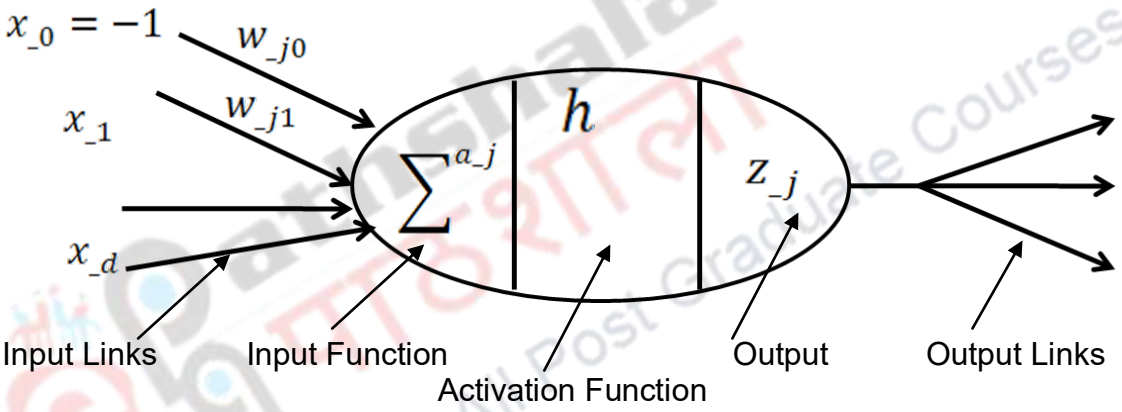

Figure 19.7 The Model of the Individual Neuron

19.13 **Representation power of Feed Forward Networks**

The representation power of the feed forward network depends on the width and depth of the networks. **Boolean functions** can be represented by network with two layers of units where the number of hidden units required grows exponentially. **Bounded continuous functions c**an be approximated with arbitrarily small error, by network with two layers of units. **Arbitrary functions** can be approximated to arbitrary accuracy by a network with three layers of units.

19.14 **Two-class Classification**

Neural Networks can also be used for two-class classification. Here there are two puts, two hidden units with *tan h activation functions.* In Figure 19.8 the red line shows the decision boundary for network., while the d ashed lines are the contours for two hidden units and the green line shows the decision boundary from distributions of the data

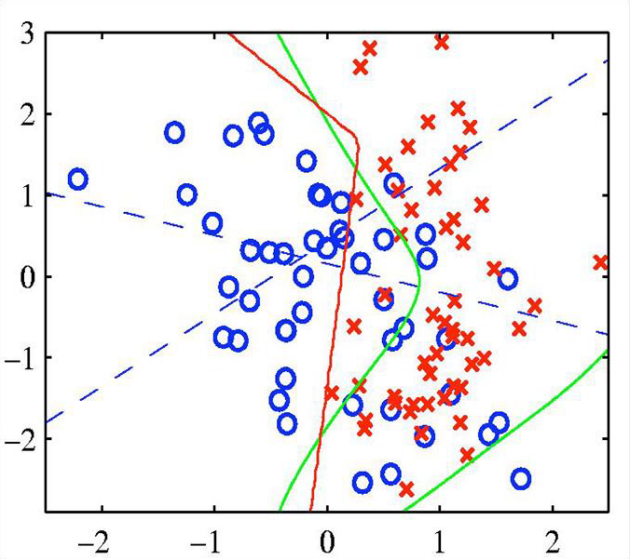

Figure 19.8 Two class Classification

19.15 **Network Training**

Now the main question to be answered is the setting of the weight parameters given a specified network structure. For this we first define a criterion to measure how well our network performs, and then optimize against it. For regression, training data are (x_n, t), t_n ε R, squared error can be defined as:

$$E(w) = \sum_{n=1}^{N} \{y(x_n, w) - t_n\}^2$$

For binary classification, this is another discriminative model, ML:

$$p(t|w) = \prod_{n=1}^{N} y_n^{t_n} \{1 - y_n\}^{1-t_n}$$

$$E(w) = -\sum_{n=1}^{N} \{t_n \ln y_n + (1 - t_n)\ln(1 - y_n)\}$$

19.15.1 Parameter Optimization: Geometrical View

For these problems, the error function E(w) is complex since it is Non-Convex with local minima. The E(w) can be viewed as a surface sitting over weight space where w_A is the local minimum and w_B is the global minimum. We need to find a minimum point w_C which is the local gradient and is given by vector

∇E(w) which essentially points in the direction of greatest rate of increase of E(w) and so correspondingly a negative gradient points to rate of greatest decrease.

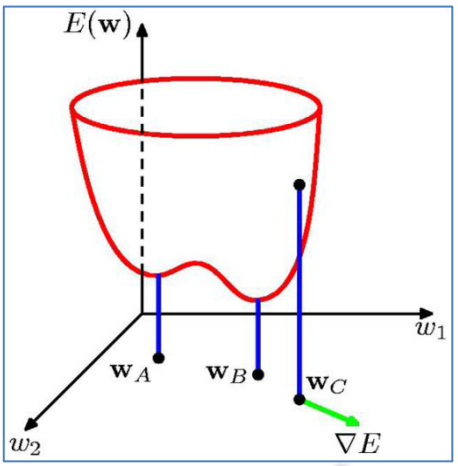

Figure 19.9 The Geometric View

19. 16 Neural Network Learning Problem

Now the goal is to learn the weights w from a labelled set of training samples. The learning procedure consists of two stages namely the evaluation of derivatives of error function ∇E(w) with respect to weights $w_1,..w_T$ and the use of these derivatives to compute adjustments to the weights.

$$w^{(t+1)} = w^{(t)} - \eta \nabla E(w^{(t)})$$

The total number of weights is T=(D+1)M+(M+1)K =M(D+K+1)+K where D is the number of inputs, M is the number of hidden units, and K is required number of outputs.

19.16.1 Descent Methods

The typical strategy for optimization problems of this sort is a descent method:

$$w^{(t+1)} = w^{(t)} + \Delta w^{(t)}$$

One such method is the gradient descent $\nabla E(w^{(t)})$ method given below

- $w^{(\tau+1)} = w^{(\tau)} - \eta \nabla E(w^{(\tau)})$ – here η is the learning rate

We can use the stochastic gradient descent $\nabla E_n(w^{(t)})$ or the second order Newton-Raphson method. The stochastic gradient descent is particularly effective. For a good optimization strategy the gradient based algorithm can run multiple times, using a different starting point every time. Starting with a range of different initial weight sets increases the chances of the chance of finding the global minimum

The function $y(x_n , w)$ implemented by a network is complex and it is necessary to compute error function derivatives with respect to weights. Numerical methods can be used for calculating error derivatives, using finite differences. However the use of gradients improves the computational speed.

Summary

- Discussed the neural network basics and Learning paradigm
- Discussed the pros and cons, types of problems handled by neural networks.
- Explained the architecture and learning method of Feed-Forward neural network