UNIT - 2

Artificial Neural Networks

* Neural Network learning method, provide a robust approach to approximations real-valued, discrete-valued and vector-valued target functions.

* ANN learning is well matched for certain types of problems, like:

- interpret complex real-world sensor data.
- Speech recognition.
- visual scenes.
- Learning robot control strategies.

* The Back propagation algorithm is one of the efficient ANN algorithm, which is successful in many problems such as

- learning to recognize handwritten characters.
- Learning to recognize faces.
- Learning to recognize spoken words.

* The Backpropagation algorithm use gradient descent to tune network parameters to best fit a training set of input-output pairs

Neurons and Biological Motivation

* The study of ANN has been inspired by the observation that biological learning systems are built of very high complex, non-linear parallel interconnection of neurons.

* The human body is made up of

billions of cells. cells of the nervous system called nerve cells or neurons. are specialized to carry "messages" through an electrochemical process.

* The information processing abilities of biological neural systems must follow from highly parallel processes operating on representations that are distributed over many neurons. one motivation for ANN is to capture this kind of highly parallel computation based on distributed representations.

* historically, two groups of reachers have worked with artificial neural networks.

* ANNs to study and model biological learning procel.

- To obtaining highly effective machine learning algorithms

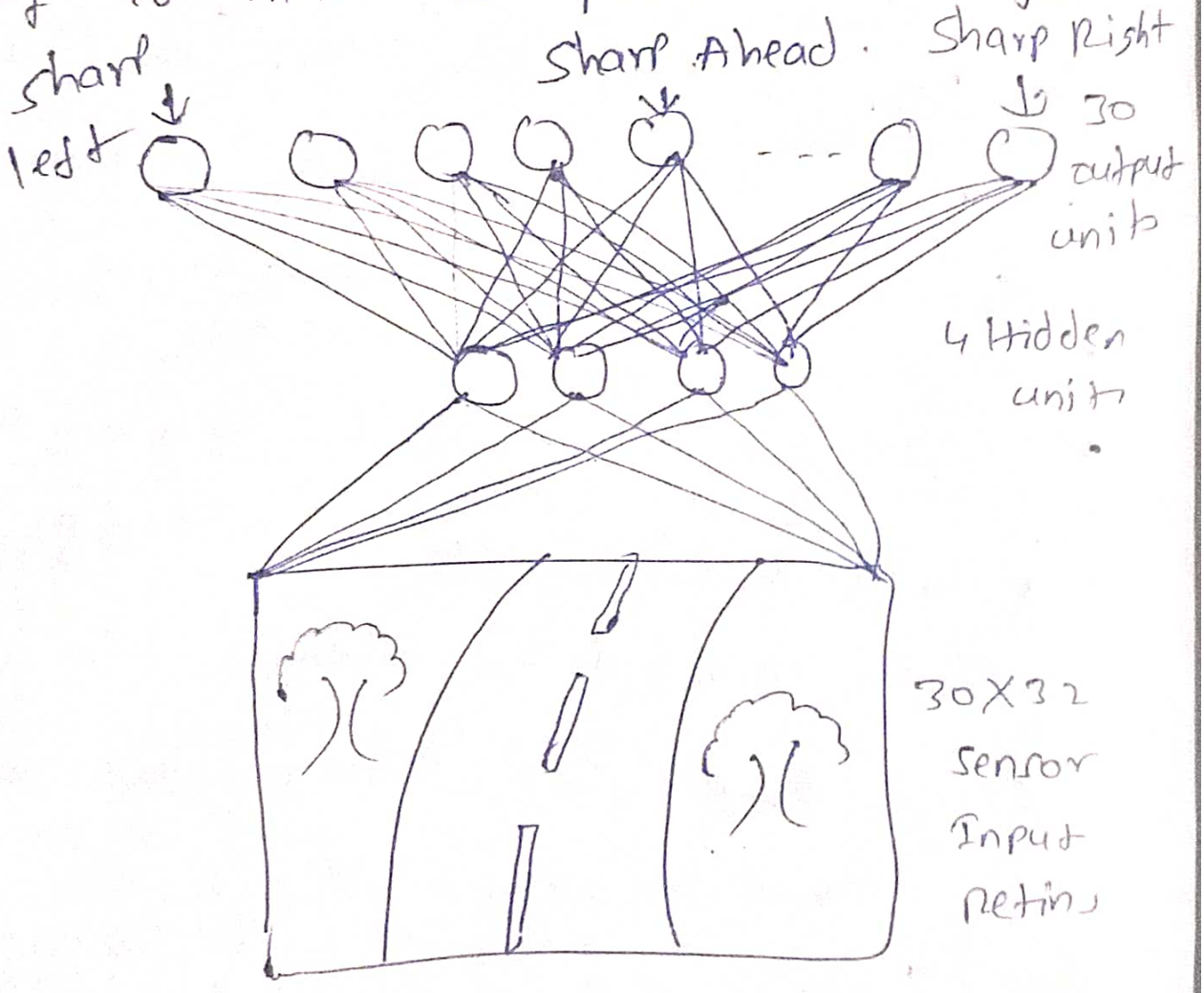
Neural Network Representations

* A prototypical example of ANN learning is ALVINN (Autonomous Land vechicle In a Neural Network) is a perception system, which uses a learned ANN to steer an autonomous vechile driving at normal speeds on public highways.

* input :- The input to the neural network is a 30 x 32 grid of pixel intensities obtained from a forward-pointed camera mounted on the vechile.

* output :- The network output is the ~~dir~~ direction in which the vechile is streered.

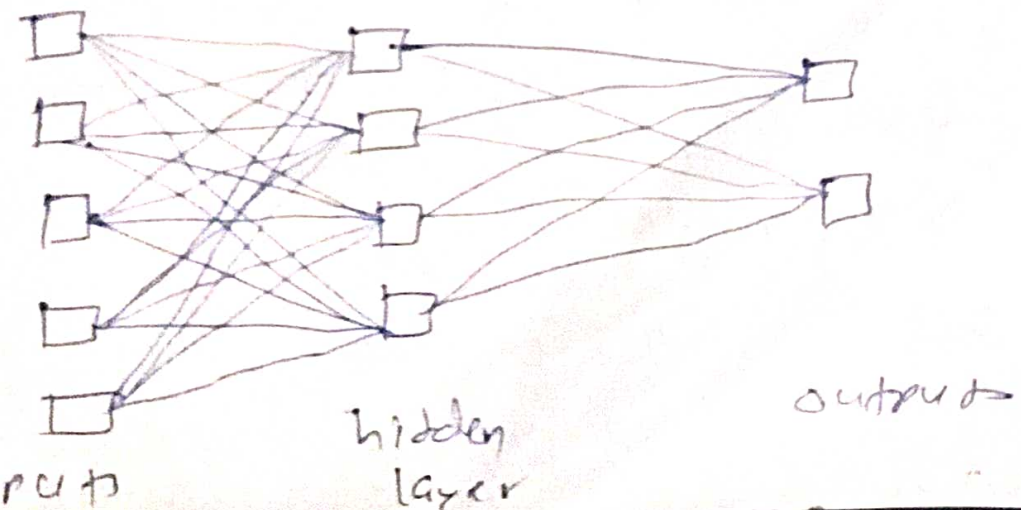
* The ANN is trained to mimic the observed steering commands of a human driving the vehicle for approximately 5 minutes. ALVINN has used its learned networks to sucessfully drive at speeds upto to two miles per hour and for distances of 90 miles on public highways.



* The ALVINN system uses Back PROPAGATION to learn to steer an autonomous vechile (photo at top) driving at speeds up to 70 miles per hour.

* The diagram shows how the image of a forward-mounted camera is mapped to 960 neural network inputs which are fed forward to 4 hidden units connected to 30 ~~opath~~ output units

* Each ANN is composed of a collection of perceptrons grouped in layers. A typical structure is shown below



Note:- The three layers: input, output and hidden layer several hidden layers can be placed between the input and output layers

Problems for Neural Network learning

* ANN learning is well suited to problems in which the training data corresponds to noisy, complex sensor data. it is also applicable to problems for which more symbolic representations are used.

* The back propagation (BP) algorithm is the most commonly used ANN learning technique. it is appropriate for problems with the characteristics:-

1) Instances are represented by many attribute-value pairs.

2) The target function output may be discrete-valued, real-valued, or a vector of several real- or discrete-valued attributes.

3) The training examples may contain errors.

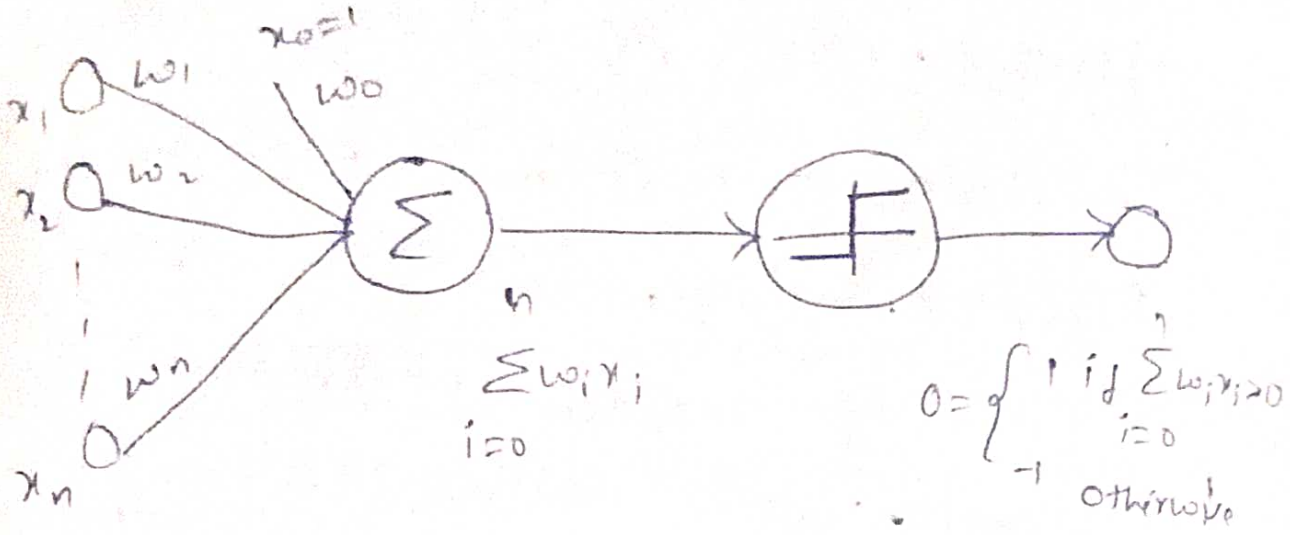
4) Long training times are acceptable.

5) Fast evaluation of the learned target function may be required.

6) The ability of humans to understand the learned target function is not important.

Perceptrons

* one type of ANN system is based on a unit called a perceptron.



A perceptron

* An ANN consists of perceptrons, Each of the perceptron receives inputs, processes input and delivers a single output

* A perceptron take a vector of real-valued inputs, then calculates a linear combination of these inputs, after that it returns 1 if the result is greater than some threshold and -1 otherwise as it

$$o(x_1, \cdots, x_n) = \begin{cases} 1 & \text{if } w_0 + w_1 x_1 + w_2 x_2 + \cdots + w_n x_n > 0 \\ -1 & \text{otherwise} \end{cases}$$

* where each * w_i is a real-valued constant, or weight, that determines the contribution of input x_i to the perceptron output.

Representational Power of perceptrons

* we can view the perceptron as representing a hyperplane decision surface in the n-dimensional space of instances. (points).

* The perceptron outputs a 1 for instances lying on one side of the hyperplane and outputs a -1 for instances lying on the other side.

* Examples can be separated by a single perceptron are called linearly separated set of examples.

* A single perceptron can be used to represent many boolean functions.

* For example, if we assume boolean values of 1 (true) and -1 (false), then one way to use a two-input perceptron to implement the AND function is to set the weights $w_0 = -0.8$, and $w_1 = w_2 = .5$.

* The perceptron can be made to represent the OR function instead by altering the threshold to $w_0 = -0.3$

* Unfortunately, however, some boolean functions cannot be represented by a single perceptron, such as the XOR function. Those are called linearly non separable training examples.

* A single perceptron can be used to represent many boolean functions.

AND function (linearly separable):-

Trainins example

x_1	x_2	output
0	0	-1
0	1	-1
1	0	-1
1	1	1

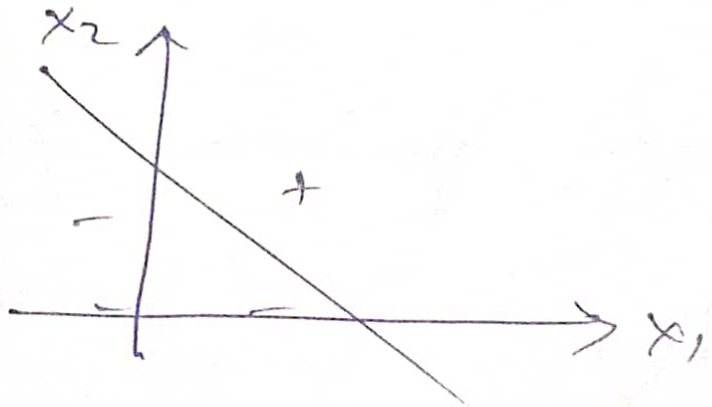
test Results

x_1	x_2	$w_i x_i$	output
0	0	-0.8	-1
0	1	-0.3	-1
1	0	-0.3	-1
1	1	0.2	1

Decision hyper plane:-

$$w_0 + w_1 x_1 + w_2 x_1 = 0.$$

$$-0.8 + 0.5 x_1 + 0.5 x_2 = 0$$



$$-0.8 + 0.5 x_1 + 0.5 x_2 = 0$$

OR function (linearly Separable)

the two-input perceptron can implement the OR function when we set the weights: $w_0 = -0.3, w_1 = w_2 = 0.5$

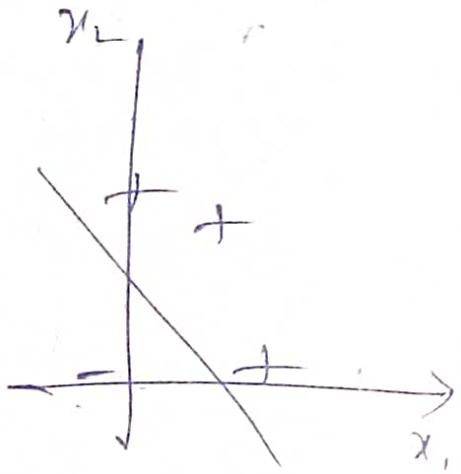
Trainins examples.

x_1	x_2	output
0	0	-1
0	1	1
1	0	1
1	1	1

Decision hyperplane:

$w_0 + w_1 x_1 + w_2 x_2 = 0$

$-0.3 + 0.5 x_1 + 0.5 x_2 = 0$



$-0.3 + 0.5 x_1 + 0.5 x_2 = 0$

Test results

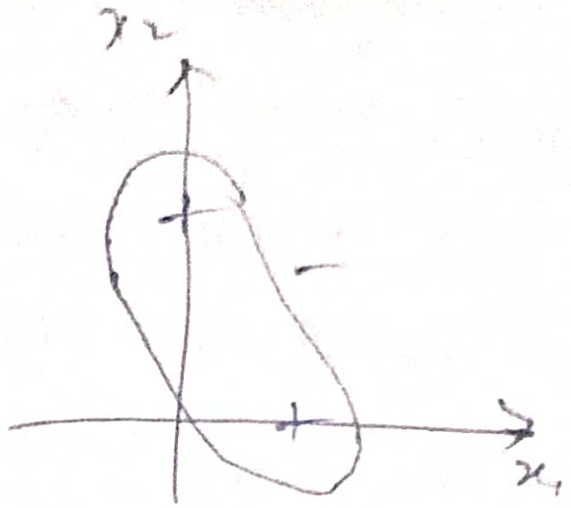
x_1	x_2	$w_i x_i$	output
0	0	-0.3	-1
0	1	0.2	-1
1	0	0.2	-1
1	1	0.7	1

XOR function (Non linearly Separable)

* It impossible to implement the XOR function by a single exception.

Training example

x_1	x_2	output
0	0	-1
0	1	1
1	0	1
1	1	-1



A two-layer network of perceptrons can represent xor function

$$x_1 \oplus x_2 = x_1 \bar{x}_2 + \bar{x}_1 x_2$$

Perceptron Trainins Rule

* They are important to ANNs be-cause they provide the basis for learning networks of many units.
* several alsorithms are known to solve this learnins problem. Here we consider two :-
  * The perceptron rule.
  * The delta rule

The Perceptron Rule :-

* One way to learn an acceptable weight vector is to begin with random weights, then iteratively apply the perceptron to each training example, modifying the perceptron weights whenever it misclassifies an example.

* The process is repeated until the perceptron classifies all training examples correctly.

*

$$w_i \leftarrow w_i + \Delta w_i$$

where

$$\Delta w_i = \eta (t - o) x_i$$

$t \rightarrow$ is the target output for the current training example

$o \rightarrow$ is the output generated by the perceptron.

$\eta \rightarrow$ is positive constant called learning rate.

Example:-

$x_i = 0.8$

$\eta = 0.1$

$t = 1$

$o = -1$

$$\Delta w_i = \eta (t - o) x_i$$

$$= 0.1 (1 - (-1)) 0.8$$

$$= 0.16$$

In this example we calculated the perceptron.

Gradient Descent and the Delta Rule:-

* The perceptron rule finds a successful weight vector when the training examples are linearly separable, it can fail to converge if the exam-ple are not linearly separable

* The second training rule, called the delta rule, is designed to overcome this difficulty.

* The key idea of rule is to use gradient descent to search the space of possible weight vector to find the weights that best fit the traning examples.

* The delta training rule is best understood by considering the task of training an unthresholded perceptron. that is, a linear unit for which the output o is given by

$$o(\vec{x}) = \vec{w} \cdot \vec{x}$$

$$E(\vec{w}) = \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2$$

$$\Delta w_i = \eta \sum_{d \in D} (t_d - o_d) x_{id}$$

update weights as

$$w_i \leftarrow w_i + \Delta w_i$$

Gradient-Descent Algorithm:-

Each training example is a pair of the form $(\vec{x}, t)$, where $\vec{x}$ is the vector of input values, and t is the target value. η is the learning rate (e.g., .05).

- Initialize each w_i to some small random value
- until the termination condition is met, DO.
  - Initialize each Δw_i to zero.
  - For each $(\vec{x}, t)$ in training-examples, DO
    - Input the instance $\vec{x}$ to the unit and compute the output o
    - for each linear unit weight w_i, DO

$$\Delta w_i \leftarrow \Delta w_i + \eta(t-o)x_i$$

- For each linear unit weight w_i, DO

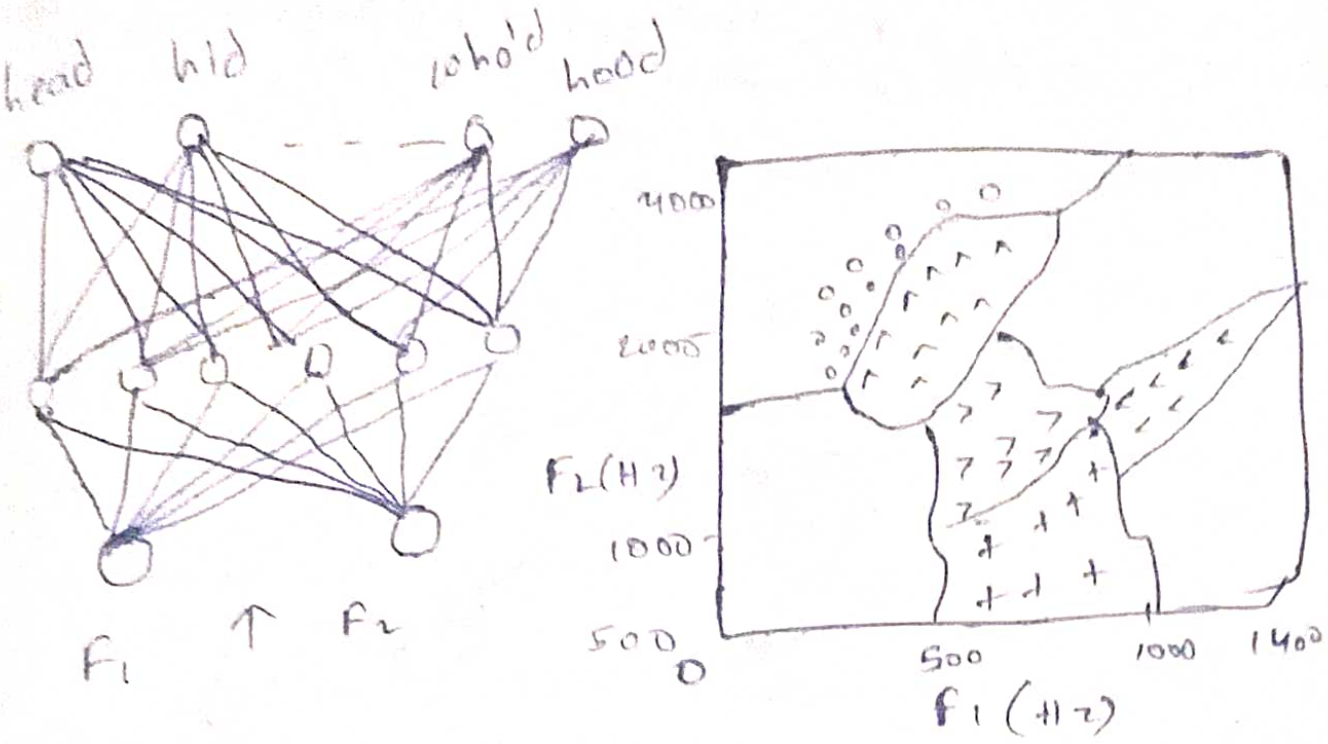
$w_i \leftarrow w_i + \Delta w_i$

Multilayer Networks

* The kind of multilayer networks learned by the BACKPROPAGATION algorithm are capable of expressing a rich variety of nonlinear decision surfaces

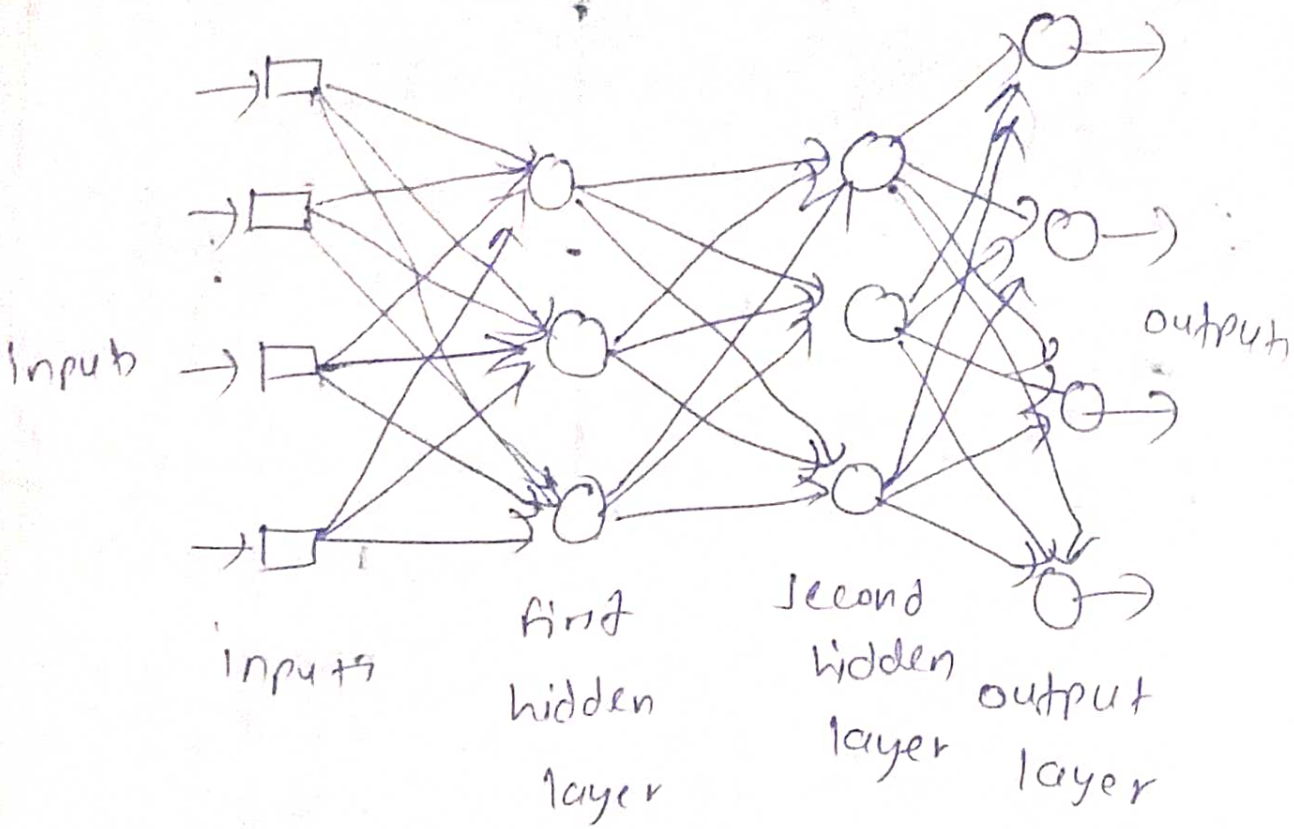
* for example,

Speech recognition task involves distinguishing among 10 possible vowels, all spoken in the context of "h-d". ("hid", "had," "head", "hood," etc).

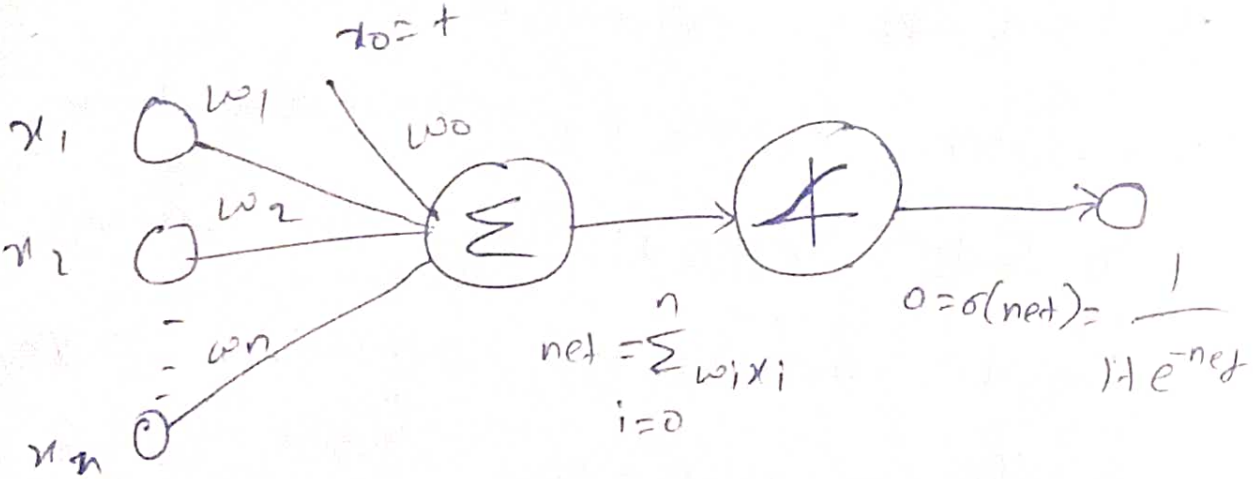


The network input consist of two parameters, f_1 and f_2 obtained from a spectral analysis of the sound. The 10 network outputs correspond to the 10 possible vowel sounds. The plot on the right illustrates the highly nonlinear decision surface

Multilayer Networks



A Differentiable Threshold unit:-



$\sigma(x)$ is the sigmoid function

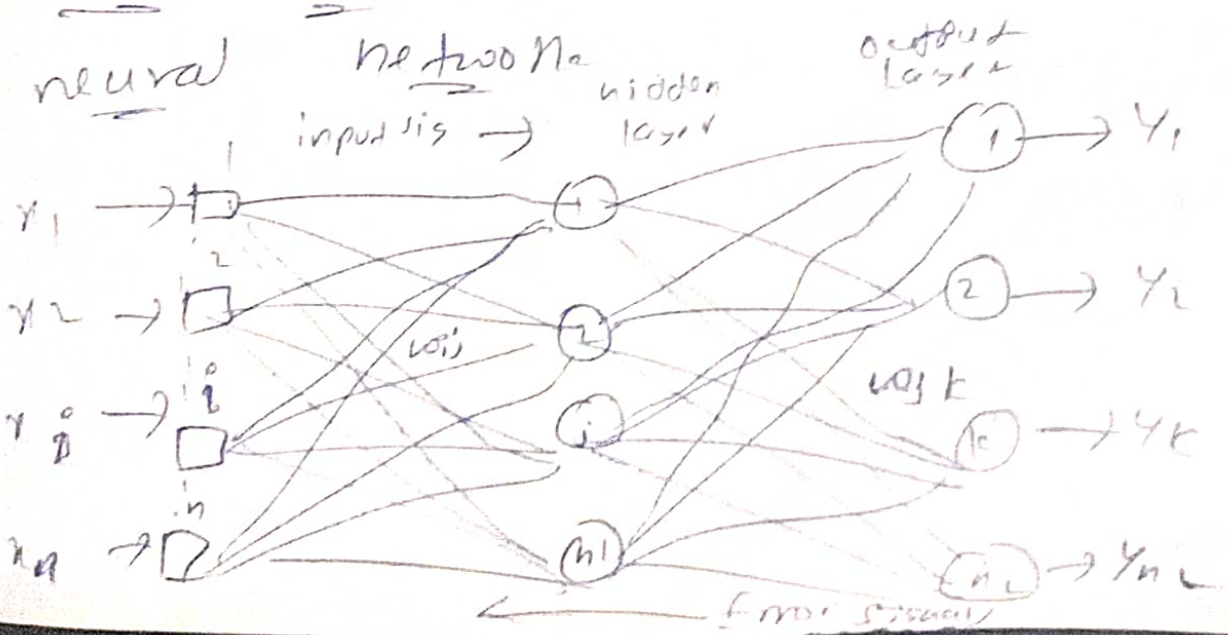
$$\frac{1}{1+e^{-x}}$$

* like the perceptron, the sigmoid unit first computes a linear combination of its inputs, then applies a threshold to the result. In the case of sigmoid unit, however, the threshold output is a continuous function of its input. The sigmoid function $\sigma(x)$ is also called the logistic function.

Interesting property:-

$$\frac{d\sigma(x)}{dx} = \sigma(x)(1-\sigma(x))$$

Two layer back-propagation neural network



The Back Propagation (BP) Algorithm:-

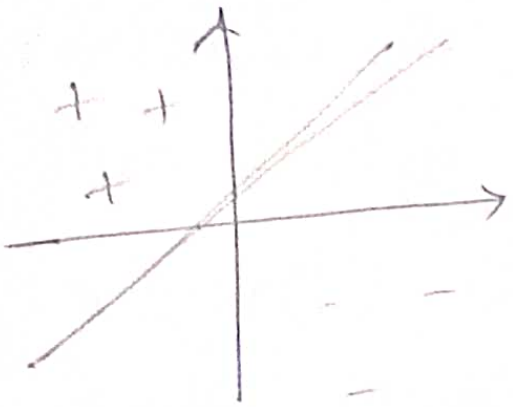
* The BP algorithm learns the weights for a multilayer network, given a network with a fixed set of units and interconnections. It employs a gradient descent to attempt to minimize the squared error between the network output values and the target values for these outputs.

* The learning problem in Back Propagation is to search a large hypothesis space defined by all possible weight values for all the units in the network

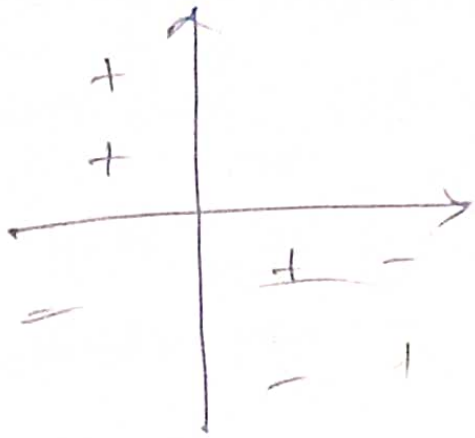
* In multilayer network the error surface can have multiple local minima.

* The gradient descent is guaranteed only to converge toward some local minimum, but not necessarily to global minimum error.

Gradient Descent and the Delta Rule



Linearly separable



Non-linearly separable

* perceptron rule finds a successful weight vector when the training examples are linearly separable, but it can fail to converse if the examples are not linearly separable.

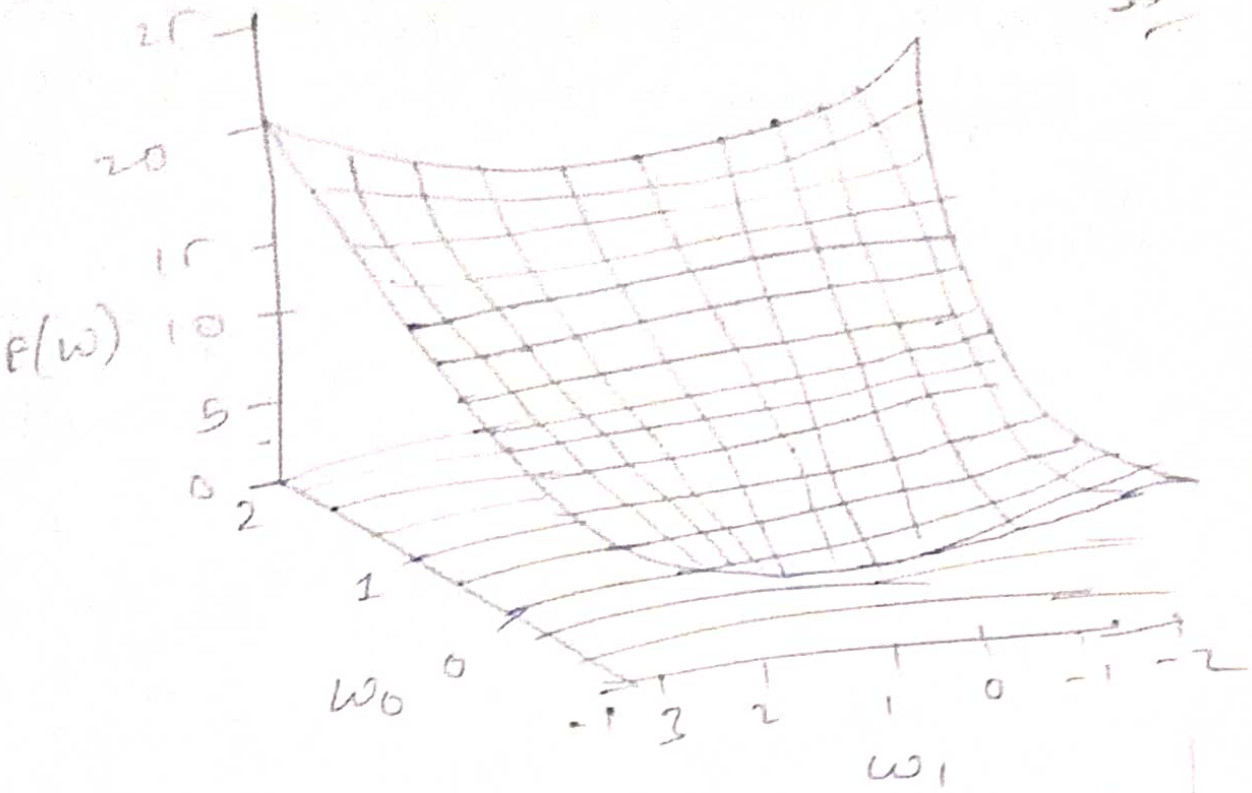
* A second trainins rule, called the delta rule, is desisned to overcom this difficulty.

* if the training examples are not linearly separable, the delta rule converges toward a best-fit approximation to the target concept.

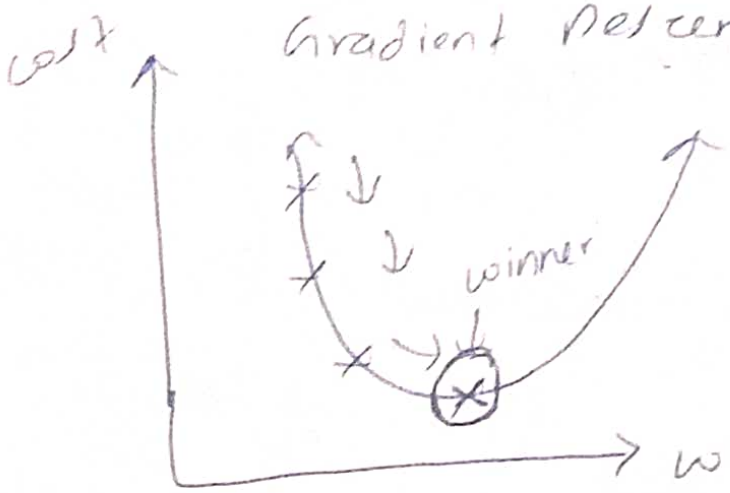
* The key idea behind the delta rule is to use gradient descent to search the hypothesis space of possible weight vectors to find the weights that best fit the training examples.

* This rule is important because gradient descent provides the basis for the Back propagation algorithm, which can learn networks with many interconnected units.

* It is also important because gradient descent can serve as the basis for learning algorithms that must search hypothesis space containing many different types of continuously parameterized hypotheses.



Gradient Descent



$$E(\vec{w}) = \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2$$

$t_{ex_3} = w(x_3, b)$

$$w_i \leftarrow w_i + \Delta w_i$$

where

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i}$$

↓

learning rate

$$\frac{\partial E}{\partial w_i} = \frac{\partial}{\partial w_i} \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2$$

$$= \frac{1}{2} \sum_{d \in D} \frac{\partial}{\partial w_i} (t_d - o_d)^2$$

$$= \frac{1}{2} \sum_{d \in D} 2 (t_d - o_d) \frac{\partial}{\partial w_i} (t_d - o_d)$$

$$= \sum_{d \in D} (t_d - o_d) \frac{\partial}{\partial w_i} (t_d - \vec{w} \cdot \vec{x}_d)$$

$$= \sum_{d \in D} (t_d - o_d)(-x_{td})$$

output calculated by

$o = w_0 + w_1 x_1 + \cdots + w_n x_n$

$$\Delta w_i = \eta \sum_{d \in D} (t_d - o_d) x_{id} \qquad \Delta w_i = -\eta \frac{\partial E}{\partial w_i}$$

$$w_i \leftarrow w_i + \Delta w_i$$

Back Propagation Algorithm

(training_example, η, n_{in}, n_{out}, n_{hidden})

- Each training example is a pair of the form (x, t), where (x) is the vector of network input values, and (t) is the vector of target network output values.
- η is the learning rate (e.g: 0.05).
- n_i is the number of network inputs.
- n_{hidden} the number of units in the hidden layer, and
- n_{out} the number of output units.
- The input from unit i into unit j is denoted x_{ji}, and the weight from unit i to unit j is denoted w_{ji}

Algorithm

- Create a feed-forward network with n_i inputs, n_{hidden} units, and n_{output} units.

- Initialize all network weights to small random numbers.
- until the termination condition is met, Do
  - for each (x, t), in training examples, Do
    - propagate the input forward through the network:
      1. Input the instances x, to the network and compute the output o_u of every unit u in the network.
    - propagate the errors backward through the network.
      2. for each network unit k, calculate its error term δ_k

$$\delta_k \leftarrow o_k(1-o_k)(t_k - o_k)$$

      3. for each network unit h, calculate its error term δ_h

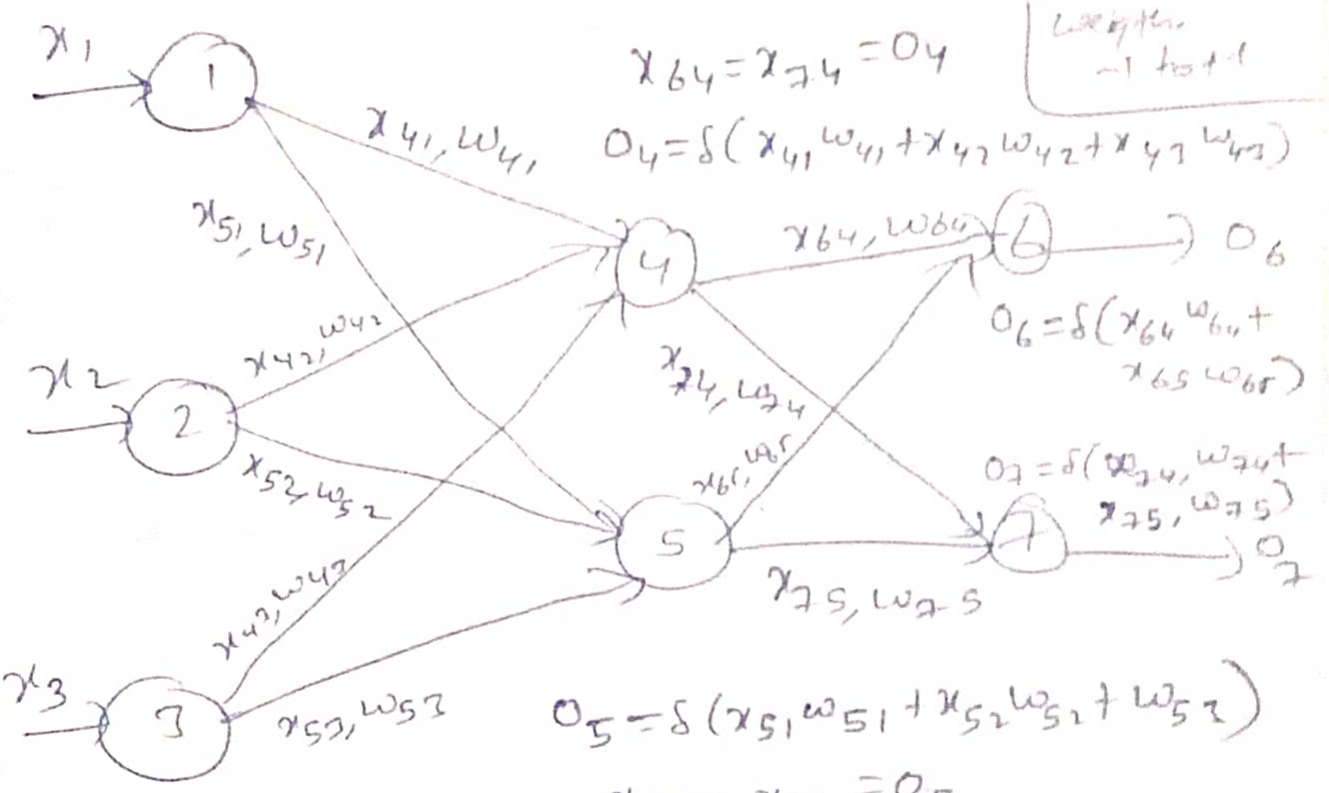
$$\delta_h \leftarrow o_h(1-o_h) \sum_{k \in outputs} w_{h,k} \delta_k$$

4. update each network weight w_{ji}

$$w_{ji} \leftarrow w_{ji} + \Delta w_{ji}$$

where

$$\Delta w_{ji} = \eta \, \delta_j \cdot x_{ji}$$



$x_{64} = x_{74} = O_4$

$O_4 = \delta(x_{41} w_{41} + x_{42} w_{42} + x_{43} w_{43})$

$O_6 = \delta(x_{64} w_{64} + x_{65} w_{65})$

$O_7 = \delta(x_{74} w_{74} + x_{75} w_{75})$

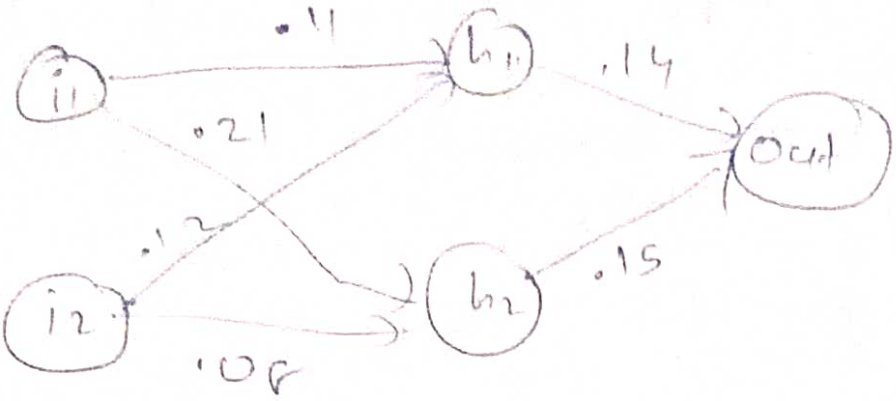
$O_5 = \delta(x_{51} w_{51} + x_{52} w_{52} + w_{53})$

$x_{75} = x_{65} = O_5$

weights -1 to $+1$

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

Example



* Neural network training is about finding weights that minimize prediction error.

* We usually start our training with a set of randomly generated weights.

* Then, backpropasation is used to update the weights in a attempt to correctly map arbitrary inputs to outputs.

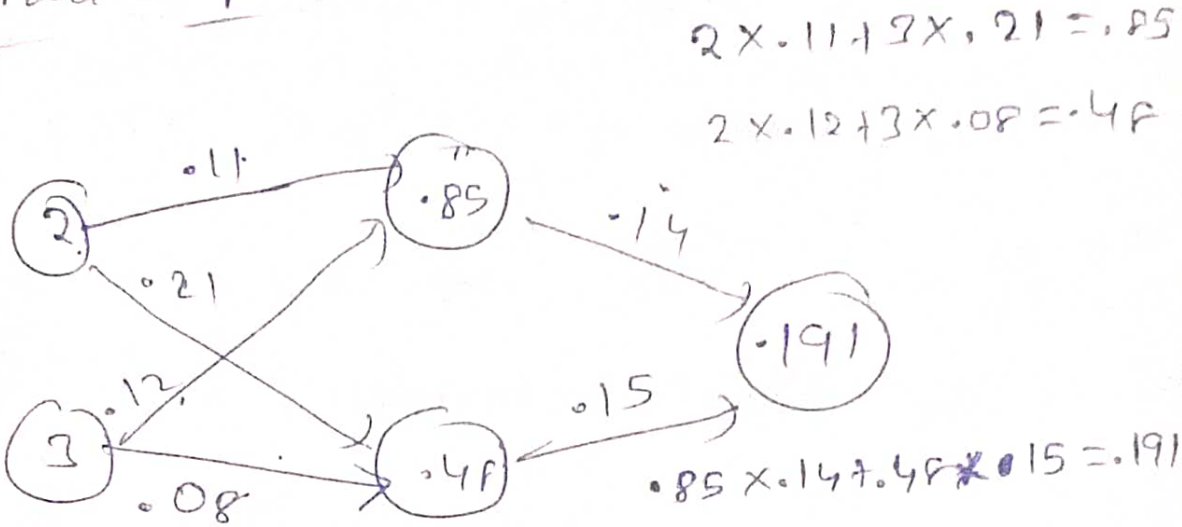
* Our initial weights will be as follows: $w_1 = 0.11$, $w_2 = 0.21$, $w_3 = 0.12$, $w_4 = 0.06$, $w_5 = 0.14$ and $w_6 = 0.15$.

Dataset

our single sample is as following

input = [2, 3] and output = 1

Forward Pass



$$[2 \quad 3] \cdot \begin{bmatrix} 0.11 & 0.12 \\ 0.21 & 0.08 \end{bmatrix} = [0.85 \quad 0.46] \cdot \begin{bmatrix} 0.14 \\ 0.15 \end{bmatrix}$$

$$= [0.191]$$

Calculating Error

$$\text{Error} = \frac{1}{2}(\text{prediction} - \text{acutal})^2$$

$$= \frac{1}{2}(0.191 - 1.0)^2 = 0.327$$

Reducing Error

- Our main goal of the training is to reduce the error.
- since actual output is constant, the only way to reduce the error is to change prediction value.
- The question now is, how to change prediction value?
- By decomposing prediction into its basic elements we can find that weights are the variable elements affecting prediction value
- In other words, in order to change Prediction value, we need to change weights values.

$$\text{Prediction} = \text{out}$$

$$\downarrow$$

$$\text{Prediction} = (h_1)\, w_{n+1} (h_2)\, w_0$$

$$\text{prediction} = (i_1 w_1 + i_2 w_2)\, w_{n+1} (i_1 w_n + i_2 w_4)\, w_0$$

to change prediction value we need to change weights.

Backpropagation

* Backpropagation short for "backward propagation of errors", is a mechanism used to update the weights using gradient descent.
* It calculates the gradient of the error function with respect to the neural network's weights.
* The calculation proceeds backwards through the network

$$w_x = w_x - a\left(\frac{\partial \text{Error}}{\partial w_x}\right)$$

(Derivate Error with respect to weight)

w_x: new weight, w_x: old weight, a: learning rate

* $w_6 = w_6 - a(h_2 \cdot \Delta)$
* $w_5 = w_5 - a(h_1 \cdot \Delta)$
* $w_4 = w_4 - a(i_2 \cdot \Delta w_6)$

* $w_3 = w_3 - a(i_1 \cdot \Delta w_6)$

* $w_2 = w_2 - a(i_1 \cdot \Delta w_5)$

* $w_1 = w_1 - a(i_1 \cdot \Delta w_5)$

$$\begin{bmatrix} w_5 \\ w_6 \end{bmatrix} = \begin{bmatrix} w_5 \\ w_6 \end{bmatrix} - a\Delta \begin{bmatrix} h_1 \\ h_2 \end{bmatrix} = \begin{bmatrix} w_5 \\ w_6 \end{bmatrix} - \begin{bmatrix} ah_1\Delta \\ ah_2\Delta \end{bmatrix}$$

$$\begin{bmatrix} w_1 & w_3 \\ w_2 & w_4 \end{bmatrix} = \begin{bmatrix} w_1 & w_3 \\ w_2 & w_4 \end{bmatrix} - a\Delta \begin{bmatrix} i_1 \\ i_2 \end{bmatrix}.$$

$$\begin{bmatrix} w_5 & w_6 \end{bmatrix} = \begin{bmatrix} w_1 & w_3 \\ w_2 & w_4 \end{bmatrix} - \begin{bmatrix} a i_1 \Delta w_5 & a i_1 \Delta w_6 \\ a i_2 \Delta w_5 & a i_2 \Delta w_6 \end{bmatrix}$$

Backward Pass

$\Delta = 0.191 - 1 = -0.809$

$a = 0.05$.

$$\begin{bmatrix} w_5 \\ w_6 \end{bmatrix} = \begin{bmatrix} 0.14 \\ 0.15 \end{bmatrix} - 0.05(-0.809)$$

$$\begin{bmatrix} 0.185 \\ 0.191 \end{bmatrix}$$

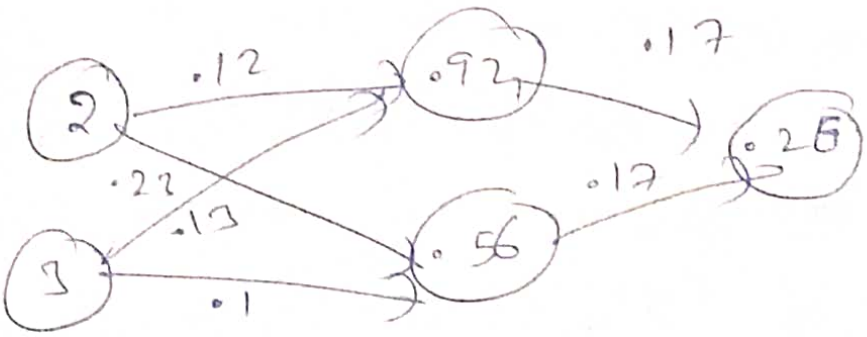
$$= \begin{bmatrix} 0.14 \\ 0.15 \end{bmatrix} - \begin{bmatrix} -0.024 \\ -0.019 \end{bmatrix} = \begin{bmatrix} 0.17 \\ 0.17 \end{bmatrix}$$

$$\begin{bmatrix} w_1 & w_3 \\ w_2 & w_4 \end{bmatrix} = \begin{bmatrix} .11 & .12 \\ .21 & .01 \end{bmatrix} - 0.05(-0.209)$$

$$\begin{bmatrix} 2 \\ 2 \end{bmatrix} \cdot \begin{bmatrix} 0.14 & 0.15 \end{bmatrix}$$

$$= \begin{bmatrix} .11 & .12 \\ .21 & .01 \end{bmatrix} - \begin{bmatrix} -0.011 & -0.012 \\ -0.017 & -0.014 \end{bmatrix}$$

$$= \begin{bmatrix} .12 & .13 \\ .23 & .10 \end{bmatrix}$$



$$\begin{bmatrix} 2 & 3 \end{bmatrix} \cdot \begin{bmatrix} 0.12 & 0.13 \\ 0.23 & 0.10 \end{bmatrix} = \begin{bmatrix} 0.92 & 0.56 \end{bmatrix} \cdot \begin{bmatrix} 0.17 \\ 0.17 \end{bmatrix}$$

$$= \begin{bmatrix} 0.26 \end{bmatrix}$$

* We notice that the Prediction 0.26 is a little bit closer to actual output that the previously Predicted one 0.191

- We can repeat the same process of backward and forward pass until error is close or equal to zero.

Gradient Descent	stochastic Gradient Descent
* Error is summed over all examples before updating weights	* weights are updated examining each trainins example
* Summing over multiple examples require more computation per weight update step	* Less computation as individual weights are updated
* Difficult when there are multiple local minimum	* uses various $E_d(w)$ rather than $E(w)$, hence handles multiple local minima in the easy.

Unit - 2 continue

Remarks on Back Propagation Algorithm:

There are 5 Remarks in it.

1. Convergence & local minima.
2. Representational power of feedforward networks.
3. Hypothesis space search and inductive bias.
4. Hidden layer Representation.
5. Generalisation, overfitting and stopping criterion.

(1) Convergence & local minima:

All the nucions are interconnected to each other you can minimize the error using local minima but not global minimum because it minimize locally (with in the n/w)

(2) Representational power of feed forward network

How detailly we will represent the network depends on the width

and depth of the n/w & Type of function used for building a n/w are

1) Boolean.
2) continous : gives us a small error.
3) Arbitary (random variables)

(3) Hypothesis Space Search and Inductive bias.

In this we move smoothly from one ~~place to~~ hypothesis to a nearby one This is a form of hypothesis space search. Inductive Bias is the set of assumptions that the learner uses to predict outputs given inputs that it has not encountered.

(4) Hidden layer Representation :-

Based on the given inputs values in some way that ~~computes~~ capture their relevant features, so that this hidden layer representation could be used by the output unit to compute the correct target values.

(5) Generalization, overfitting and stopping criterion

The termination condition for the algorithm has been left unspecified. What is an appropriate condition for terminating the weight update loop.

(1) Continue training until the Error E on the training examples fall below some predetermined threshold.

(2) Backpropagation is susceptible to overfitting the training examples at the cost of decreasing generalization accuracy over other unseen examples

An Illustrative Example:- Face Recognition:-

→ face Recognition is a category of biometric software that maps individuals facial features and stores the data as a face print.

→ Machine learning and Image processing are backbone of face Recognition.

Image Processing

Involves the process of computer vision.

Image Reading :- Reads any image between the range of 0-255.

for any image, there are 3 colors - RGB.

- OpenCV :- python library to solve computer vision.

Machine learning

takes dataset as input and learn from that data.

- identifies the pattern (height, width, color etc).

- ML does 3 major functions in face Recognition.

1. Deriving the feature vector
2. Matching Algorithm,
3. face Recognition Operations.

(i) Face-detection
(ii) Face. Analysis
(iii) Image to data conversion.
(iv) Match finding.

Face Recognition Software

Deepvision AI.
Sensetime
Amazon Rekognition.
facefirst
Truefac
cognitec

Applications

Hospital security.
Airline Industry.

Advanced topics in Neural Networks.

3 advanced topics.

1. Alternative error functions:

Adding an extra penalty to $E(w)$ can reduce the problem of overfitting.

- Each weight is multiplied with a constant in each iterations.
  ∴ we can redefine E as

$$\epsilon(\vec{w}) = \frac{1}{2} \sum_{d \in D} \sum_{k \in o/p} (t_{kd} - o_{kd})^2 + \gamma \sum w_{ji}^2$$

2. Alternative error minimisation procedures

2 procedures

1. line search : optimized way to find the distance.
2. conjugate gradient descent method.

↳ Same as line search but diff is error minimisation

3. Recurrent Neural Network (RNN) :-

ANN that can be applied to time series data and uses the output of n/w units at time t as input to other units at time $t+1$

Evaluating the Hypothesis:-

evaluating the accuracy of hypothesis is basic of ML

for evaluating the accuracy of hypothesis, we mainly focus on 3 questions.

1. Hypothesis is accurate over limited sample of data, then what about additional data.
2. we have 2 hypothesis - one is better than other when used on sample of data.

   Then will better hypothesis work good for general data also?
3. when we have have limited data, what is the best way to use this data for both learning and estimating hypothesis.

Motivation

Sometimes, very precise hypothesis is required In that cases estimating the accuracy is very important.

Example:- Medical Treatment.

Estimating hypothesis accuracy:-

Let us make some assumptions,

- There is some space of possible instances x over with vary target functions may be defined.
  - different instances will have different frequiencies
- Based on frequency, each instance in x will have some unknown probability.
- Trainer will ~~th~~ teach the machine about the training examples of target function.

Send a particular instance x_i along with target value $f(x)$

Target function: $f : x \rightarrow \{0, 1\}$

classifies each instance into diff categories based on req. Now after classification, what is the probable error in this estimation we made.

Error. 2 types

(1) Sample Error.

(2) True Error.

Sample Error :-

The error rate of hypothesis over a sample of data.

$$\text{Sample error} = \text{error}_S(h) = \frac{1}{n} \sum_{s \in S} \delta(f(x), h(x))$$

$$(f(x), h(x)) = 1 \text{ if } f(x) \neq h(x)$$

$$= 0 \quad \text{otherwise.}$$

True error :-

The error rate of hypothesis over entire distribution D.

$$\text{True error} = \text{error}_D(h) = P[f(x) \neq h(x)]$$