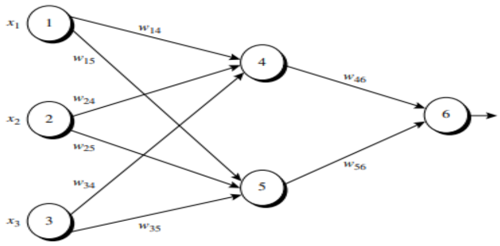
CMR
INSTITUTE OF
TECHNOLOGY

USN

Internal Assesment Test –II, November 2020

Sub:	MACHINE LEARNING							Code:	18MCA53
Date:	04-11-2020	Duration:	90 mins	Max Marks:	50	Sem:	V	Branch:	MCA

<table>
<tr><th colspan="2" rowspan="2">Answer Any 5 QUESTIONS</th><th rowspan="2">Marks</th><th colspan="2">OBE</th></tr>
<tr><th>CO</th><th>RBT</th></tr>
<tr><td>Q1</td><td>What is linearly in separable problem? Design a two-layer network of perceptron to implement X AND Y.</td><td>10</td><td>CO3</td><td>L5</td></tr>
<tr><td>Q2</td><td>Derive the Gradient Descent Rule to find the best fit in Perceptron model.</td><td>10</td><td>CO3</td><td>L4</td></tr>
<tr><td>Q3</td><td>Prove that how maximum likelihood (Bayesian Learning) can be used in any learning algorithm that are used to minimize the squared error between actual output hypothesis and predicted output hypothesis.</td><td>10</td><td>CO4</td><td>L5</td></tr>
<tr><td>Q 4</td><td>Explain Brute force Bayes Concept Learning.</td><td>10</td><td>CO4</td><td>L2</td></tr>
<tr><td>Q 5</td><td>Consider a medical diagnosis problem in which there are two alternative hypotheses: 1: that the patient has a particular form of cancer (+) and 2: That the patient does not (-). A patient takes a lab test and the result comes back positive. It is known that the test returns a correct positive result in only 98% of the cases and a correct negative result in only 97% of the cases. Furthermore, only 0.008 of the entire population has this disease. Determine whether the patient has cancer or not using MAP hypothesis.</td><td>10</td><td>CO4</td><td>L5</td></tr>
<tr><td>Q 6</td><td>Differentiate between Gradient Descent and Stochastic Gradient Descent.</td><td>10</td><td>CO3</td><td>L2</td></tr>
<tr><td>Q 7</td><td>Discuss biological neural network with neat diagram.</td><td>10</td><td>CO3</td><td>L2</td></tr>
<tr><td>Q 8</td><td>Apply back propagation algorithm for the following neural network architecture.


Initial input, weight and bias values are given as follows:
<table>
<tr><th>x_1</th><th>x_2</th><th>x_3</th><th>w_{14}</th><th>w_{15}</th><th>w_{24}</th><th>w_{25}</th><th>w_{34}</th><th>w_{35}</th><th>w_{46}</th><th>w_{56}</th><th>θ_4</th><th>θ_5</th><th>θ_6</th></tr>
<tr><td>1</td><td>0</td><td>1</td><td>0.2</td><td>−0.3</td><td>0.4</td><td>0.1</td><td>−0.5</td><td>0.2</td><td>−0.3</td><td>−0.2</td><td>−0.4</td><td>0.2</td><td>0.1</td></tr>
</table></td><td>10</td><td>CO3</td><td>L5</td></tr>
</table>

CMR
INSTITUTE OF
TECHNOLOGY

USN

Internal Assessment Test 2 – November. 2020

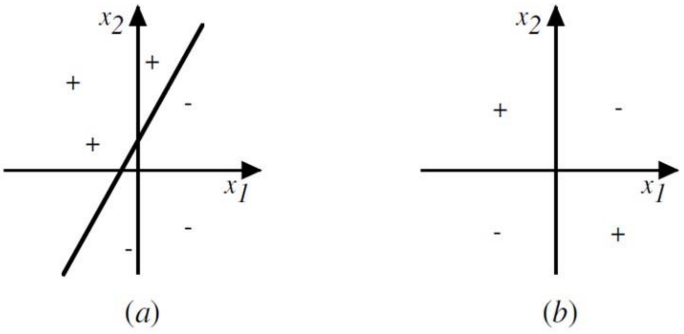
Sub:	MACHINE LEARNING							**Sub Code:**	18MCA 53
Date:	04-11-2020	Duration:	90 min’s	Max Marks:	50	**Sem**	5th	**Branch:**	**MCA**

Note : Answer FIVE FULL Questions, choosing ONE full question from each Module

PART I	MARKS	OBE	
		CO	RBT
1)	10	CO3	L5

Representational Power of Perceptrons

- The perceptron can be viewed as representing a hyperplane decision surface in the n- dimensional space of instances (i.e., 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, as illustrated in below figure



(a) (b)

Figure : The decision surface represented by a two-input perceptron.
(a) A set of training examples and the decision surface of a perceptron that classifies them correctly. *(b)* A set of training examples that is not linearly separable.
x_1 and x_2 are the Perceptron inputs. Positive examples are indicated by "+", negative by "-".

AND GATE

X1	X2	t
1	1	1
1	-1	-1
-1	1	-1
-1	-1	-1

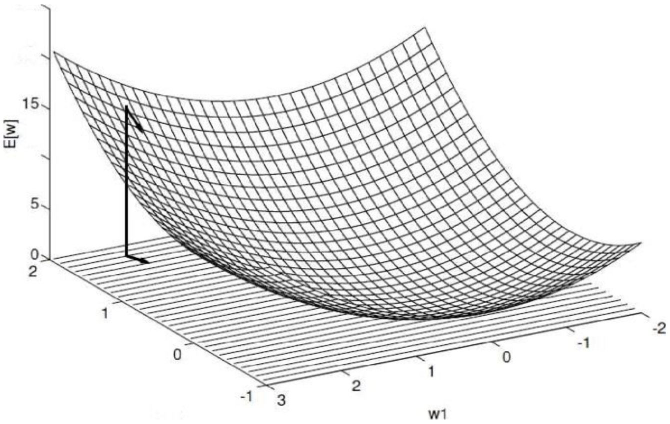
Initialization : w1= 0, w2 = 0, b =0

X1	X2	t	Yi	Y	Δw1	Δw2	Δb	w1	w2	b
EPOCH - 1										
1	1	1	0	0	1	1	1	1	1	1

1	-1	-1	1	1	-1	1	-1	0	2	0
-1	1	-1	2	1	1	-1	-1	1	1	-1
-1	-1	-1	-3	-1	-	-	-	1	1	-1
EPOCH-2										
1	1	1	1	1	-	-	-	1	1	-1
1	-1	-1	-1	-1	-	-	-	1	1	-1
-1	1	-1	-1	-1	-	-	-	1	1	-1
-1	-1	-1	-3	-1	-	-	-	1	1	-1

2)

- Perceptron learning converges to a consistent model if D (training set) is linearly separable.
- If the data is not linearly separable than this will not converge.
- 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.



- Gradient descent search determines a weight vector that minimizes E by starting with an arbitrary initial weight vector, then repeatedly modifying it in small steps.
- At each step, the weight vector is altered in the direction that produces the steepest descent along the error surface depicted in above figure. This process continues until the global minimum error is reached.

Derivation of the Gradient Descent Rule

How to calculate the direction of steepest descent along the error surface?

The direction of steepest can be found by computing the derivative of E with respect to each component of the vector $\vec{w}$. This vector derivative is called the gradient of E with respect to $\vec{w}$, written as

$$\nabla E[\vec{w}] \equiv \left[\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \cdots \frac{\partial E}{\partial w_n}\right] \qquad \text{equ. (3)}$$

10 CO3 L4

The gradient specifies the direction of steepest increase of E, the training rule for gradient descent is

$$\vec{w} \leftarrow \vec{w} + \Delta\vec{w}$$

Where,

$$\Delta\vec{w} = -\eta\nabla E(\vec{w}) \qquad \text{equ. (4)}$$

- Here η is a positive constant called the learning rate, which determines thestep size in the gradient descent search.
- The negative sign is present because we want to move the weight vector in the direction that decreases E.

This training rule can also be written in its component form

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

Where,

$$\Delta w_i = -\eta\frac{\partial E}{\partial w_i} \qquad \text{equ. (5)}$$

$$\Delta w_i = \eta\sum_{d\in D}(t_d - o_d)\ x_{id} \qquad \text{equ. (7)}$$

$$\partial w_i \qquad \partial w_i 2 \; d$$

$$\begin{aligned}
&= \frac{1}{2}\sum_d \frac{\partial}{\partial w_i}(t_d - o_d)^2 \\
&= \frac{1}{2}\sum_d 2(t_d - o_d)\frac{\partial}{\partial w_i}(t_d - o_d) \\
&= \sum_d (t_d - o_d)\frac{\partial}{\partial w_i}(t_d - \vec{w}\cdot\vec{x}_d) \\
\frac{\partial E}{\partial w_i} &= \sum_d (t_d - o_d)(-x_{i,d}) \qquad \text{equ. (6)}
\end{aligned}$$

Substituting Equation (6) into Equation (5) yields the weight update rule for gradient desc

---

3)

Consider the problem of learning a *continuous-valued target function* such as neural network learning, linear regression, and polynomial curve fitting

A straightforward Bayesian analysis will show that under certain assumptions any learning algorithm that minimizes the squared error between the output hypothesis predictions and the training data will output a *maximum likelihood (ML) hypothesis*

- Learner L considers an instance space X and a hypothesis space H consisting of some class of real-valued functions defined over X, i.e., (∀ h ∈ H)[h : X → R] and training examples of the form <xi, di>
- The problem faced by L is to learn an unknown target function f : X → R

10 CO4 L5

- A set of m training examples is provided, where the target value of each example is corrupted by random noise drawn according to a Normal probability distribution with zero mean (di = f(xi) + ei)
- Each training example is a pair of the form (xi ,di) where di = f (xi) + ei .
  - – Here f(xi) is the noise-free value of the target function and ei is a random variable representing the noise.
  - – It is assumed that the values of the ei are drawn independently and that they are distributed according to a Normal distribution with zero mean.
- The task of the learner is to *output a maximum likelihood hypothesis* or a *MAP hypothesis assuming all hypotheses are equally probable a priori*.

Using the definition of hML we have

$$h_{ML} = \underset{h \in H}{argmax}\ p(D|h)$$

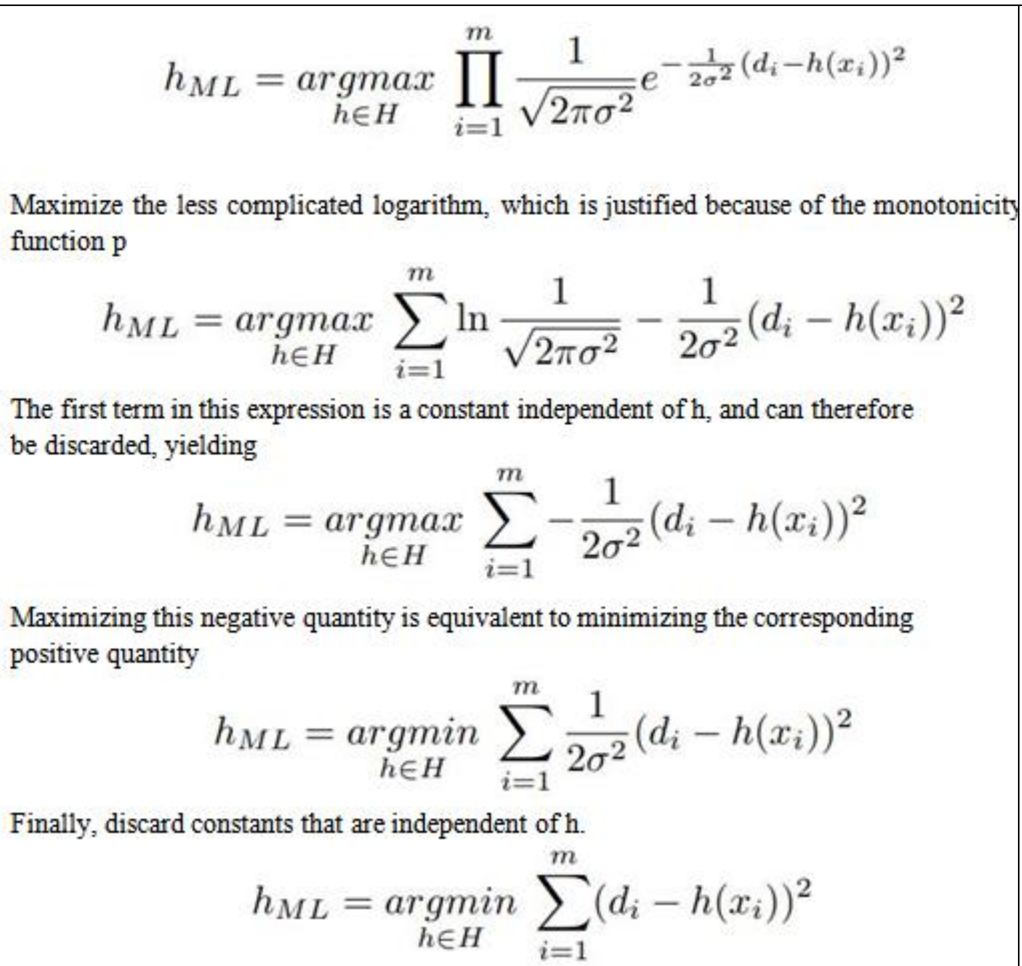
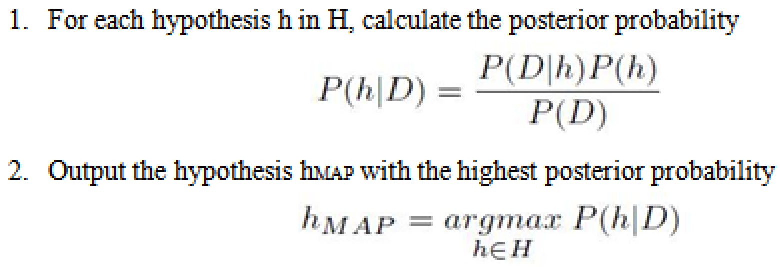
Assuming training examples are mutually independent given h, we can write P(D|h) as the product of the various (di|h)

$$h_{ML} = \underset{h \in H}{argmax} \prod_{i=1}^{m} p(d_i|h)$$

Given the noise ei obeys a Normal distribution with zero mean and unknown variance σ^2 , each di must also obey a Normal distribution around the true targetvalue f(xi). Because we are writing the expression for P(D|h), we assume h is the correct description of f.

Hence, μ = f(xi) = h(xi)

$$h_{ML} = \underset{h \in H}{\text{argmax}} \prod_{i=1}^{m} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2\sigma^2}(d_i-\mu)^2}$$

	 $$h_{ML} = \underset{h\in H}{argmax} \prod_{i=1}^{m} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2\sigma^2}(d_i-h(x_i))^2}$$ Maximize the less complicated logarithm, which is justified because of the monotonicity function p $$h_{ML} = \underset{h\in H}{argmax} \sum_{i=1}^{m} \ln \frac{1}{\sqrt{2\pi\sigma^2}} - \frac{1}{2\sigma^2}(d_i - h(x_i))^2$$ The first term in this expression is a constant independent of h, and can therefore be discarded, yielding $$h_{ML} = \underset{h\in H}{argmax} \sum_{i=1}^{m} -\frac{1}{2\sigma^2}(d_i - h(x_i))^2$$ Maximizing this negative quantity is equivalent to minimizing the corresponding positive quantity $$h_{ML} = \underset{h\in H}{argmin} \sum_{i=1}^{m} \frac{1}{2\sigma^2}(d_i - h(x_i))^2$$ Finally, discard constants that are independent of h. $$h_{ML} = \underset{h\in H}{argmin} \sum_{i=1}^{m} (d_i - h(x_i))^2$$ Thus, above equation shows that the maximum likelihood hypothesis hML is the one that minimizes the sum of the squared errors between the observed training values di and the hypothesis predictions h(xi)			
4)	**Brute-Force Bayes Concept Learning** Consider the concept learning problem - Assume the learner considers some finite hypothesis space H defined over the instance space X, in which the task is to learn some target concept c : X → {0,1}. - Learner is given some sequence of training examples ((x1, d1) . . . (xm, dm)) where xi is some instance from X and where di is the target value of xi (i.e., di = c(xi)). - The sequence of target values are written as D = (d1 . . . dm). We can design a straightforward concept learning algorithm to output the maximum a posteriori hypothesis, based on Bayes theorem, as follows: **BRUTE-FORCE MAP LEARNING algorithm:**  1. For each hypothesis h in H, calculate the posterior probability $$P(h\|D) = \frac{P(D\|h)P(h)}{P(D)}$$ 2. Output the hypothesis h_{MAP} with the highest posterior probability $$h_{MAP} = \underset{h\in H}{argmax}\ P(h\|D)$$	10	CO4	L2

In order specify a learning problem for the BRUTE-FORCE MAP LEARNING algorithm we must specify what values are to be used for P(h) and for P(D|h) ?

Let’s choose P(h) and for P(D|h) to be consistent with the following assumptions:

- The training data D is noise free (i.e., di = c(xi))
- The target concept c is contained in the hypothesis space H

Do not have a priori reason to believe that any hypothesis is more probable than any other.

- Assumptions
  - The training data D is noise-free

$$d_i = c(x_i)$$

  - The target concept c is in the hypothesis set H

$$c \in H$$

  - All hypotheses are equally likely

$$P(h) = \frac{1}{|H|}$$

- Choice: Probability of D given h

$$P(D\,|\,h) = \begin{cases} 1 & \text{if } \forall d \in D, h(d) = c(d) \\ 0 & \text{else} \end{cases}$$

$$P(h\,|\,D) = \frac{P(D\,|\,h)P(h)}{P(D)}$$

Bayes Theorem

$$= \frac{P(D\,|\,h)\cdot\frac{1}{|H|}}{P(D)}$$

Given our assumptions

If the data is not consistent

$$P(h\,|\,D) \ = 0$$

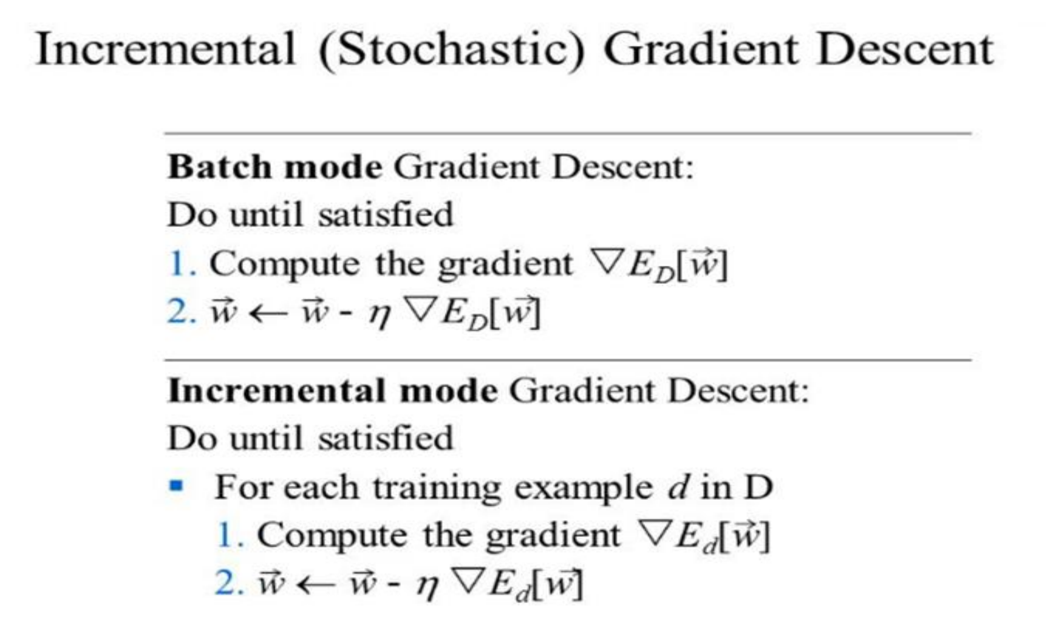
If the data is consistent

$$= \frac{1\cdot\frac{1}{|H|}}{P(D)} = \frac{1\cdot\frac{1}{|H|}}{\frac{|VS_{H,D}|}{|H|}} = \frac{1}{|VS_{H,D}|}$$

$VS_{H,D}$ is the version space

<table>
<tr><td></td><td>Given: $(\forall i \neq j)(P(h_i \wedge h_j) = 0)$ (the hypotheses are mutually exclusive):
$$P(D) = \sum_{h_i \in H} P(D \mid h_i)P(h_i)$$
$$= \sum_{h_i \in VS_{H,D}} 1 \cdot \frac{1}{|H|} + \sum_{h_i \notin VS_{H,D}} 0 \cdot \frac{1}{|H|}$$
$$= \sum_{h_i \in VS_{H,D}} 1 \cdot \frac{1}{|H|}$$
$$= \frac{|VS_{H,D}|}{|H|}$$
To summarize, Bayes theorem implies that the posterior probability P(h|D) under our assumed P(h) and P(D|h) is
$$P(D|h) = \begin{cases} \frac{1}{|V\,S_{H,D}|} & \text{if h is consistent with D} \\ 0 & \text{otherwise} \end{cases}$$</td><td></td><td></td><td></td></tr>
<tr><td>5)</td><td>P(cancer) = 0.008 P(⌐cancer) = 1 – 0.008
= 0.992
P(+|cancer) = 0.98 P(- |cancer) = 1-0.98= 0.02
P(- | ⌐cancer) = 0.97 P(+ |⌐cancer)=1-0.97= 0.03
Now a new patient, whose test result is positive, Should we diagnose the patient have cancer or not?
$$p(A \mid B) = \frac{P(A \wedge B)}{P(B)} = \frac{P(B \mid A)P(A)}{P(B)}$$
P(cancer|+) = P(+ |cancer) * P(cancer)
= 0.98 * 0.008
= 0.078
P(⌐cancer | +) = P(+ | ⌐cancer) * P(⌐cancer)
= 0.03 * 0.992
= 0.298
Since,
P(cancer|+) < P(⌐cancer | +)
So we can conclude that,
Diagnosis : Not having cancer</td><td>10</td><td>CO4</td><td>L5</td></tr>
<tr><td>6)</td><td><u>Types of Gradient Descent</u>:
1. Batch Gradient Descent:
• Parameters are updated after computing the gradient of error with respect to the entire training set.
• computationally expensive.
• very slow on very large training data.
2. Stochastic Gradient Descent (or Incremental Gradient Descent): Parameters are updated after computing the gradient of error with respect to a single training example.
3. Mini-batch Gradient Descent: Parameters are updated after computing the gradient of error with respect to a subset of the training set.</td><td>10</td><td>CO3</td><td>L2</td></tr>
</table>

Batch Gradient Descent	Stochastic Gradient Descent	Mini-Batch Gradient Descent
Since entire training data is considered before taking a step in the direction of gradient, therefore it takes a lot of time for making a single update.	Since only a single training example is considered before taking a step in the direction of gradient, thus it is faster and less computationally expensive than Batch GD	Since a subset of training examples is considered, it can make quick updates in the model parameters.
It makes smooth updates in the model parameters	It makes very noisy updates in the parameters	Depending upon the batch size, the updates can be made less noisy – greater the batch size less noisy is the update

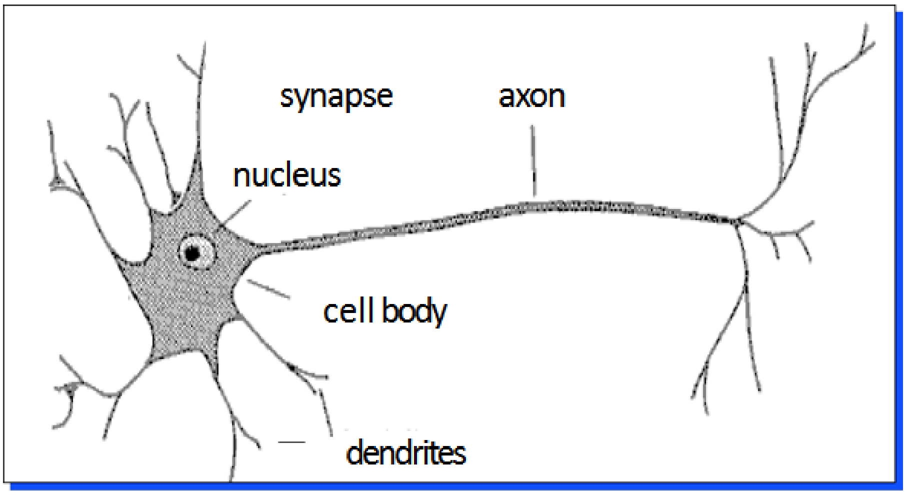


Incremental (Stochastic) Gradient Descent

```
Batch mode Gradient Descent:
Do until satisfied
1. Compute the gradient $\nabla E_D[\vec{w}]$
2. $\vec{w} \leftarrow \vec{w} - \eta \nabla E_D[\vec{w}]$

Incremental mode Gradient Descent:
Do until satisfied
▪ For each training example $d$ in D
    1. Compute the gradient $\nabla E_d[\vec{w}]$
    2. $\vec{w} \leftarrow \vec{w} - \eta \nabla E_d[\vec{w}]$
```

7)



The fundamental element of the neural network is called a neuron. A neuron mainly consists of three parts: dendrites, soma, and axon. Dentrites are the tree-like structure that receives the signal from surrounding neurons, where each line is connected to one neuron. Axon is a thin cylinder that transmits the signal from one neuron to others. At the end of axon, the contact to the dendrites is made through a

10 CO3 L2

	synapse. The inter-neuronal signal at the synapse is usually chemical diffusion but sometimes electrical impulses. A neuron fires an electrical impulse only if certain condition is met. The incoming impulse signal from each synapse to the neuron is either excitatory or inhibitory, which means helping or hindering firing. The condition of causing firing is that the excitatory signal should exceed the inhibitory signal by a certain amount in a short period of time, called the period of latent summation. As we assign a weight to each incoming impulse signal, the excitatory signal has positive weight and the inhibitory signal has negative weight. This way, we can say, ``A neuron fires only if the total weight of the synapses that receive impulses in the period of latent summation exceeds the threshold."			
8)	Step1: I1=1(=X1) 01= 1 I2=0(=X2) O2=0 I3 = 1(=X3) O3 = 1 **STEP2:** **Unit** **Net Input** **Output** **4** = 0.2* 1 + 0.4*0 – 0.5* 1 – 0.4 =1/(1+$e^{0.7}$) = -0.7 = 0.332 **5** = 0.1 = 0.525 **6** = -0.105 = 0.474 **Step 3:** a. **Calculate Output layer Error** Errj = Oj * (1-Oj)(Tj – Oj) b. **Calculate Hidden layer Error** Errj = O_j * (1-O_j) Summation(Err$_k$ * W$_{jk}$) **Unit** **Error** 6 = 0.474 * (1 - 0.474) * (1 - 0.474) = 0.1311 5 = 0.525 * (1 - 0.525)* 0.1311 * (-0.2) = -0.0065 4 = 0.332 * (1- 0.332) *0.1311 * (-0.3) = -0.0087 **Step 4:** a. **For weights** Delta wij = (l) Errj * Oi Wij = wij + delta wij b. **For Bias** Delta Theta j = (l) * Err j Thetaj = Theta j + delta Theta j	10	CO3	L5

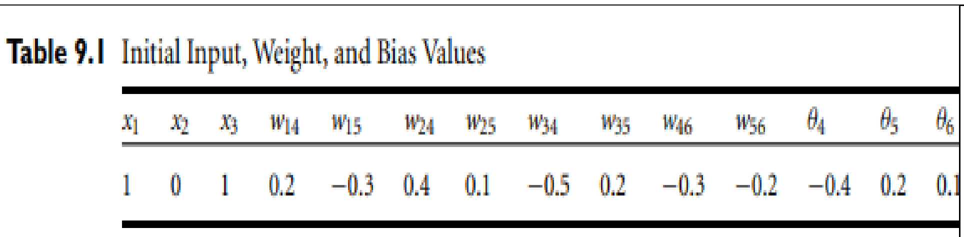


Table 9.1 Initial Input, Weight, and Bias Values

x_1	x_2	x_3	w_{14}	w_{15}	w_{24}	w_{25}	w_{34}	w_{35}	w_{46}	w_{56}	θ_4	θ_5	θ_6
1	0	1	0.2	−0.3	0.4	0.1	−0.5	0.2	−0.3	−0.2	−0.4	0.2	0.1

Let l = 0.9

Weights/Bias	**New Value**
W46	= -0.3 + 0.9*0.1311*0.332 = -0.261
W56	= -0.2 + 0.9 * 0.1311 * 0.525 = -0.138
W14	= 0.2 + 0.9 *(-0.0087) *1 = 0.192
W15	= -0.3 + 0.9 * (-0.0065) * 1 = -0.306
W24	= 0.4 + 0.9*(-0.0087) * 0 = 0.4
W25	= 0.1 + 0.9 *(-0.0065) * 0 = 0.1
W34	= -0.5 + 0.9*(-0.0087) * 1 = -0.508
W35	=0.2 + 0.9*(-0.0065) * 1 = 0.194
Theta6	= 0.1 + 0.9 *0.1311 = 0.218
Theta5	= 0.2 + 0.9 * (-0.0065) = 0.194
Theta4	= -0.4 + 0.9* (-0.0087) = -0.408