## Module-3

5 a. Define perceptron. Explain the concept of single perceptron with neat diagram. **(06 Marks)**

b. Explain the back propagation algorithm. Why is it not likely to be trapped in local minima? **(10 Marks)**

**OR**

6 a. List the appropriate problems for neural network learning. **(04 Marks)**

b. Discuss the perceptron training rule and delta rule that solves the learning problem of perceptron. **(08 Marks)**

c. Write a remark on representation of feed forward networks. **(04 Marks)**

## PERCEPTRON

- One type of ANN system is based on a unit called a perceptron. Perceptron is a single layer neural network.

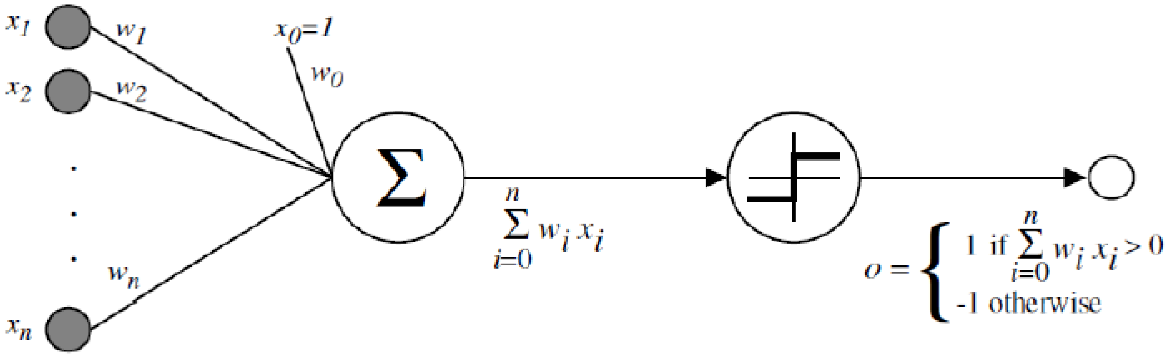



**Figure:** A perceptron

- A perceptron takes a vector of real-valued inputs, calculates a linear combination of these inputs, then outputs a 1 if the result is greater than some threshold and -1 otherwise.
- Given inputs $x$ through $x$, the output $O(x_1, \ldots, x_n)$ computed by the perceptron is

$$o(x_1, \ldots, x_n) = \begin{cases} 1 \text{ if } w_0 + w_1x_1 + \cdots + w_nx_n > 0 \\ -1 \text{ otherwise.} \end{cases}$$

---

- Where, each $w_i$ is a real-valued constant, or weight, that determines the contribution of input $x_i$ to the perceptron output.
- $-w_0$ is a threshold that the weighted combination of inputs $w_1x_1 + \ldots + w_nx_n$ must surpass in order for the perceptron to output a 1.

Sometimes, the perceptron function is written as,

$$O(\vec{x}) = \text{sgn}(\vec{w} \cdot \vec{x})$$

Where,

$$\text{sgn}(y) = \begin{cases} 1 \text{ if } y > 0 \\ -1 \text{ otherwise.} \end{cases}$$

Learning a perceptron involves choosing values for the weights $w_0, \ldots, w_n$. Therefore, the space H of candidate hypotheses considered in perceptron learning is the set of all possible real-valued weight vectors

$$H = \{\vec{w} \mid \vec{w} \in \Re^{(n+1)}\}$$

## 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

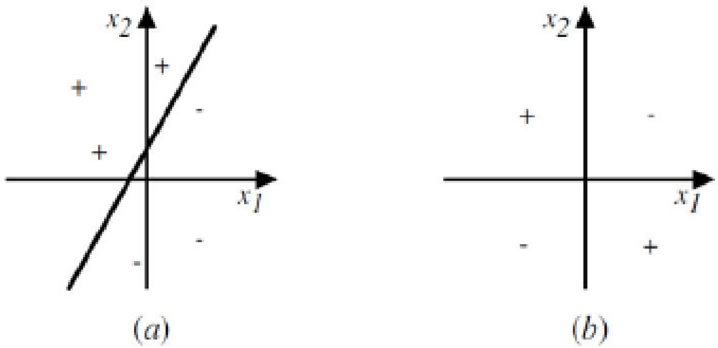



*Figure :* The decision surface represented by a two-input perceptron.
*(a)* A set of training examples and the decision surface of a perceptron that classifies them correctly. *(b)* A set of training examples that is not linearly separable.
$x_1$ and $x_2$ are the Perceptron inputs. Positive examples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

Perceptrons can represent all of the primitive Boolean functions AND, OR, NAND (~ AND), and NOR (~OR)

Some Boolean functions cannot be represented by a single perceptron, such as the XOR function whose value is 1 if and only if $x_1 \neq x_2$

---

**5b)**

## The BACKPROPAGATION Algorithm

- The BACKPROPAGATION Algorithm learns the weights for a multilayer network, given a network with a fixed set of units and interconnections. It employs gradient descent to attempt to minimize the squared error between the network output values and the target values for these outputs.
- In BACKPROPAGATION algorithm, we consider networks with multiple output units rather than single units as before, so we redefine E to sum the errors over all of the network output units.

$$E(\vec{w}) \equiv \frac{1}{2} \sum_{d \in D} \sum_{k \in outputs} (t_{kd} - o_{kd})^2 \quad \text{........equ. (1)}$$

where,

- *outputs* - is the set of output units in the network
- $t_{kd}$ and $O_{kd}$ - the target and output values associated with the $k_{th}$ output unit
- $d$ - training example

BACKPROPAGATION (*training_example*, $\eta$, $n_{in}$, $n_{out}$, $n_{hidden}$ )

*Each training example is a pair of the form* ($\vec{x}$, $\vec{t}$ ), *where* ($\vec{x}$ ) *is the vector of network input values,* ($\vec{t}$ ) *and is the vector of target network output values.*
$\eta$ *is the learning rate (e.g., .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* $w_{ji}$

- Create a feed-forward network with $n_i$ inputs, $n_{hidden}$ hidden units, and $n_{out}$ output units.
- Initialize all network weights to small random numbers
- Until the termination condition is met, Do
  - For each ($\vec{x}$, $\vec{t}$ ), in training examples, Do

    *Propagate the input forward through the network:*

  1. Input the instance $\vec{x}$, 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 output unit k, calculate its error term $\delta_k$

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

  3. For each hidden unit *h*, calculate its error term $\delta_h$

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

  4. Update each network weight $w_{ji}$

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

Where

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

## REMARKS ON THE BACKPROPAGATION ALGORITHM

### 1. Convergence and Local Minima

- The BACKPROPAGATION multilayer networks is only guaranteed to converge toward some local minimum in E and not necessarily to the global minimum error.
- Despite the lack of assured convergence to the global minimum error, BACKPROPAGATION is a highly effective function approximation method in practice.
- Local minima can be gained by considering the manner in which network weights evolve as the number of training iterations increases.

Common heuristics to attempt to alleviate the problem of local minima include:

1. Add a momentum term to the weight-update rule. Momentum can sometimes carry the gradient descent procedure through narrow local minima
2. Use stochastic gradient descent rather than true gradient descent
3. Train multiple networks using the same data, but initializing each network with different random weights

**6a)**

## APPROPRIATE PROBLEMS FOR NEURAL NETWORK LEARNING

ANN learning is well-suited to problems in which the training data corresponds to noisy, complex sensor data, such as inputs from cameras and microphones.

ANN is appropriate for problems with the following 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

---

**6b)**

### The Perceptron Training Rule

The learning problem is to determine a weight vector that causes the perceptron to produce the correct + 1 or - 1 output for each of the given training examples.

To learn an acceptable weight vector

- Begin with random weights, then iteratively apply the perceptron to each training example, modifying the perceptron weights whenever it misclassifies an example.
- This process is repeated, iterating through the training examples as many times as needed until the perceptron classifies all training examples correctly.
- Weights are modified at each step according to the perceptron training rule, which revises the weight $w_i$ associated with input xi according to the rule.

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

Where,

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

Here,
***t*** is the target output for the current training example
***o*** is the output generated by the perceptron
η is a positive constant called the ***learning rate***

- The role of the learning rate is to moderate the degree to which weights are changed at each step. It is usually set to some small value (e.g., 0.1) and is sometimes made to decay as the number of weight-tuning iterations increases

**Gradient Descent and the Delta Rule**

- 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.

To understand the delta training rule, consider the task of training an unthresholded perceptron. That is, a linear unit for which the output ***O*** is given by

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

$$O(\vec{x}) = (\vec{w} \cdot \vec{x}) \qquad \text{equ. ( 1 )}$$

To derive a weight learning rule for linear units, specify a measure for the ***training error*** of a hypothesis (weight vector), relative to the training examples.

$$E[\vec{w}] \equiv \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2 \qquad \text{equ. ( 2 )}$$

Where,

- D is the set of training examples,
- $t_d$ is the target output for training example d,
- $o_d$ is the output of the linear unit for training example d
- E ( $\vec{w}$ ) is simply half the squared difference between the target output $t_d$ and the linear unit output $o_d$, summed over all training examples.

---

**6c)**

## 2. Representational Power of Feedforward Networks

*What set of functions can be represented by feed-forward networks?*

The answer depends on the width and depth of the networks. There are three quite general results are known about which function classes can be described by which types of Networks

1. Boolean functions – Every boolean function can be represented exactly by some network with two layers of units, although the number of hidden units required grows exponentially in the worst case with the number of network inputs

2. Continuous functions – Every bounded continuous function can be approximated with arbitrarily small error by a network with two layers of units

3. Arbitrary functions – Any function can be approximated to arbitrary accuracy by a network with three layers of units.

## Module-3

5 a. Explain artificial neural network based on perception concept with diagram. (06 Marks)
b. What is gradient descent and delta rule? Why stochastic approximation to gradient descent is needed? (04 Marks)
c. Describe the multilayer neural network. Explain why back propagation algorithm is required. (06 Marks)



**5b)**

### Stochastic Approximation to Gradient Descent

- The gradient descent training rule presented in Equation (7) computes weight updates after summing over all the training examples in D
- The idea behind stochastic gradient descent is to approximate this gradient descent search by updating weights incrementally, following the calculation of the error for each individual example

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

- where t, o, and $x_i$ are the target value, unit output, and $i^{th}$ input for the training example in question

**5c)**

**Not available ----search or refer anything**

OR

6 a. Derive the back propagation rule considering the output layer and training rule for output unit weights. (08 Marks)
b. What is squashing function 3 why is it needed? (04 Marks)
c. List out and explain in briefly representation power of feed forward networks. (04 Marks)

**6a)**

### Derivation of the BACKPROPAGATION Rule

- Deriving the stochastic gradient descent rule: Stochastic gradient descent involves iterating through the training examples one at a time, for each training example d descending the gradient of the error $E_d$ with respect to this single example
- For each training example d every weight $w_{ji}$ is updated by adding to it $\Delta w_{ji}$

$$\Delta w_{ji} = -\eta \frac{\partial E_d}{\partial w_{ji}} \qquad \text{.........equ. (1)}$$

where, $E_d$ is the error on training example d, summed over all output units in the network

$$E_d(\vec{w}) \equiv \frac{1}{2} \sum_{k \in output} (t_k - o_k)^2$$

Here outputs is the set of output units in the network, $t_k$ is the target value of unit $k$ for training example $d$, and $o_k$ is the output of unit $k$ given training example $d$.

The derivation of the stochastic gradient descent rule is conceptually straightforward, but requires keeping track of a number of subscripts and variables

- $x_{ji}$ = the $i^{th}$ input to unit j
- $w_{ji}$ = the weight associated with the $i^{th}$ input to unit j
- $net_j = \Sigma_i w_{ji}x_{ji}$ (the weighted sum of inputs for unit j )
- $o_j$ = the output computed by unit j
- $t_j$ = the target output for unit j
- $\sigma$ = the sigmoid function
- outputs = the set of units in the final layer of the network
- Downstream(j) = the set of units whose immediate inputs include the output of unit j

derive an expression for $\frac{\partial E_d}{\partial w_{ji}}$ in order to implement the stochastic gradient descent rule

seen in Equation $\Delta w_{ji} = -\eta \frac{\partial E_d}{\partial w_{ji}}$

notice that weight $w_{ji}$ can influence the rest of the network only through $net_j$.

Use chain rule to write

$$\frac{\partial E_d}{\partial w_{ji}} = \frac{\partial E_d}{\partial net_j} \frac{\partial net_j}{\partial w_{ji}}$$

$$= \frac{\partial E_d}{\partial net_j} x_{ji} \qquad \text{......equ(2)}$$

Derive a convenient expression for $\frac{\partial E_d}{\partial net_j}$

**Consider two cases:** The case where unit j is an output unit for the network, and the case where j is an internal unit (hidden unit).

**Case 1: Training Rule for Output Unit Weights.**

$w_{ji}$ can influence the rest of the network only through $net_j$ , $net_j$ can influence the network only through $o_j$. Therefore, we can invoke the chain rule again to write

$$\frac{\partial E_d}{\partial net_j} = \frac{\partial E_d}{\partial o_j}\frac{\partial o_j}{\partial net_j} \qquad \text{.....equ( 3)}$$

To begin, consider just the first term in Equation (3)

$$\frac{\partial E_d}{\partial o_j} = \frac{\partial}{\partial o_j}\frac{1}{2}\sum_{k \in outputs}(t_k - o_k)^2$$

The derivatives $\frac{\partial}{\partial o_j}(t_k - o_k)^2$ will be zero for all output units $k$ except when $k = j$. We therefore drop the summation over output units and simply set $k = j$.

$$\begin{aligned}\frac{\partial E_d}{\partial o_j} &= \frac{\partial}{\partial o_j}\frac{1}{2}(t_j - o_j)^2 \\ &= \frac{1}{2}2(t_j - o_j)\frac{\partial(t_j - o_j)}{\partial o_j} \\ &= -(t_j - o_j) \qquad \text{......equ (4)}\end{aligned}$$

Next consider the second term in Equation ( 3). Since $o_j = \sigma(net_j)$, the derivative $\frac{\partial o_j}{\partial net_j}$ is just the derivative of the sigmoid function, which we have already noted is equal to $\sigma(net_j)(1 - \sigma(net_j))$. Therefore,

$$\begin{aligned}\frac{\partial o_j}{\partial net_j} &= \frac{\partial \sigma(net_j)}{\partial net_j} \\ &= o_j(1 - o_j) \qquad \text{.......equ(5)}\end{aligned}$$

Substituting expressions (4) and ( 5) into (3), we obtain

$$\frac{\partial E_d}{\partial net_j} = -(t_j - o_j)\ o_j(1 - o_j) \qquad \text{.......equ (6)}$$

and combining this with Equations (1) and ( 2), we have the stochastic gradient descent rule for output units

$$\Delta w_{ji} = -\eta\frac{\partial E_d}{\partial w_{ji}} = \eta\ (t_j - o_j)\ o_j(1 - o_j)x_{ji} \qquad \text{.......equ (7)}$$

**6c)**

## 2. Representational Power of Feedforward Networks

*What set of functions can be represented by feed-forward networks?*

The answer depends on the width and depth of the networks. There are three quite general results are known about which function classes can be described by which types of Networks

1. Boolean functions – Every boolean function can be represented exactly by some network with two layers of units, although the number of hidden units required grows exponentially in the worst case with the number of network inputs

2. Continuous functions – Every bounded continuous function can be approximated with arbitrarily small error by a network with two layers of units

3. Arbitrary functions – Any function can be approximated to arbitrary accuracy by a network with three layers of units.

---

**Module-3**

| | | | |
|---|---|---|---|
| 5 | a. | Draw the perceptron network with the notation. Derive an equation of gradient descent rule to minimize the error. | (08 Marks) |
| | b. | Explain the importance of the terms : (i) Hidden layer (ii) Generalization (iii) Overfitting (iv) Stopping criterion | (08 Marks) |

**5a)**

## 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

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

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 the step 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,

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

Calculate the gradient at each step. The vector of $\frac{\partial E}{\partial wi}$ derivatives that form the gradient can be obtained by differentiating E from Equation (2), as

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

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

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

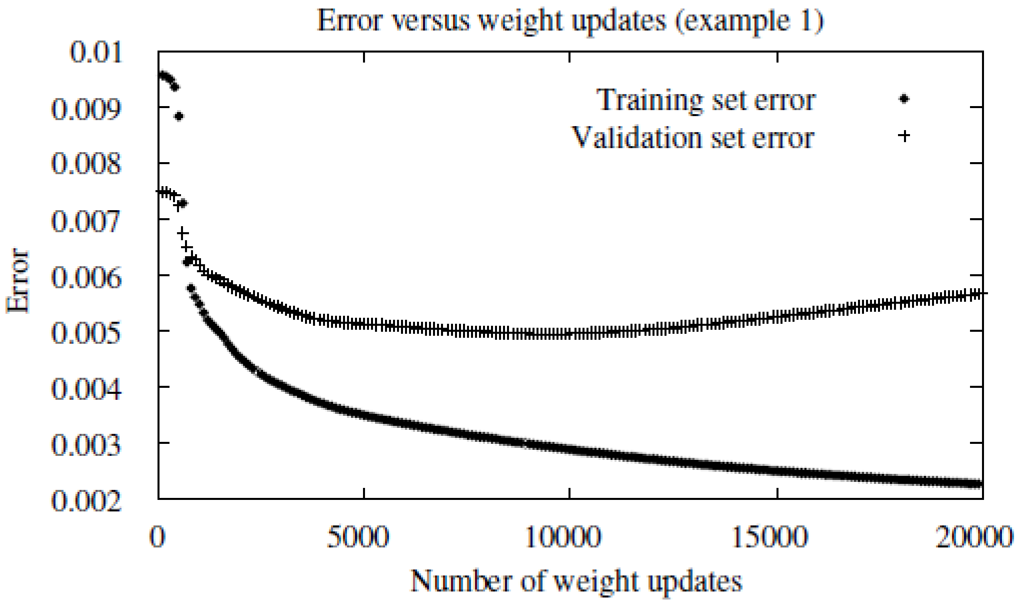
---

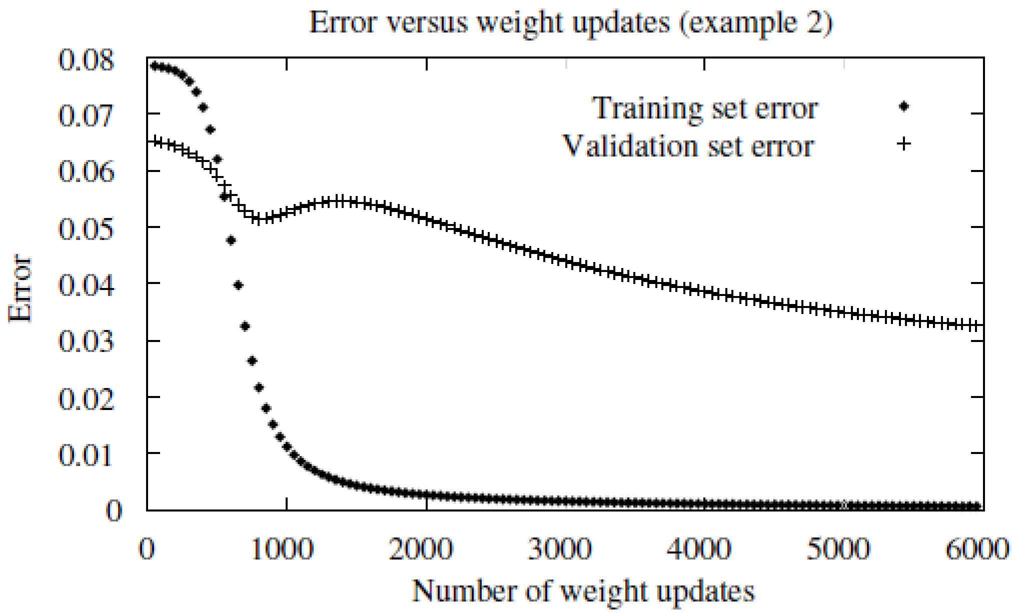
**5b)**

**4. Hidden Layer Representations**

BACKPROPAGATION can define new hidden layer features that are not explicit in the input representation, but which capture properties of the input instances that are most relevant to learning the target function.

**5. Generalization, Overfitting, and Stopping Criterion**




• Consider first the top plot in this figure. The lower of the two lines shows the monotonically decreasing error E over the training set, as the number of gradient descent iterations grows. The upper line shows the error E measured over a different validation set of examples, distinct from the training examples. This line measures the generalization accuracy of the network-the accuracy with which it fits examples beyond the training data.

□ The generalization accuracy measured over the validation examples first decreases, then increases, even as the error over the training examples continues to decrease. How can this occur? This occurs because the weights are being tuned to fit idiosyncrasies of the training examples that are not representative of the general distribution of examples. The large number of weight parameters in ANNs provides many degrees of freedom for fitting such idiosyncrasies

□ Why does overfitting tend to occur during later iterations, but not during earlier iterations?
By giving enough weight-tuning iterations, BACKPROPAGATION will often be able to create overly complex decision surfaces that fit noise in the training data or unrepresentative characteristics of the particular training sample.

OR

6 a. Discuss the application of Neural network which is used for learning to steer an autonomous vehicle. (06 Marks)

b. Write an algorithm for back propagation algorithm which uses stochastic gradient descent method. Comment on the effect of adding momentum to the network. (10 Marks)

• A prototypical example of ANN learning is provided by Pomerleau's system ALVINN, which uses a learned ANN to steer an autonomous vehicle driving at normal speeds on public highways

 The input to the neural network is a 30x32 grid of pixel intensities obtained from a forward-pointed camera mounted on the vehicle.

 The network output is the direction in which the vehicle is steered

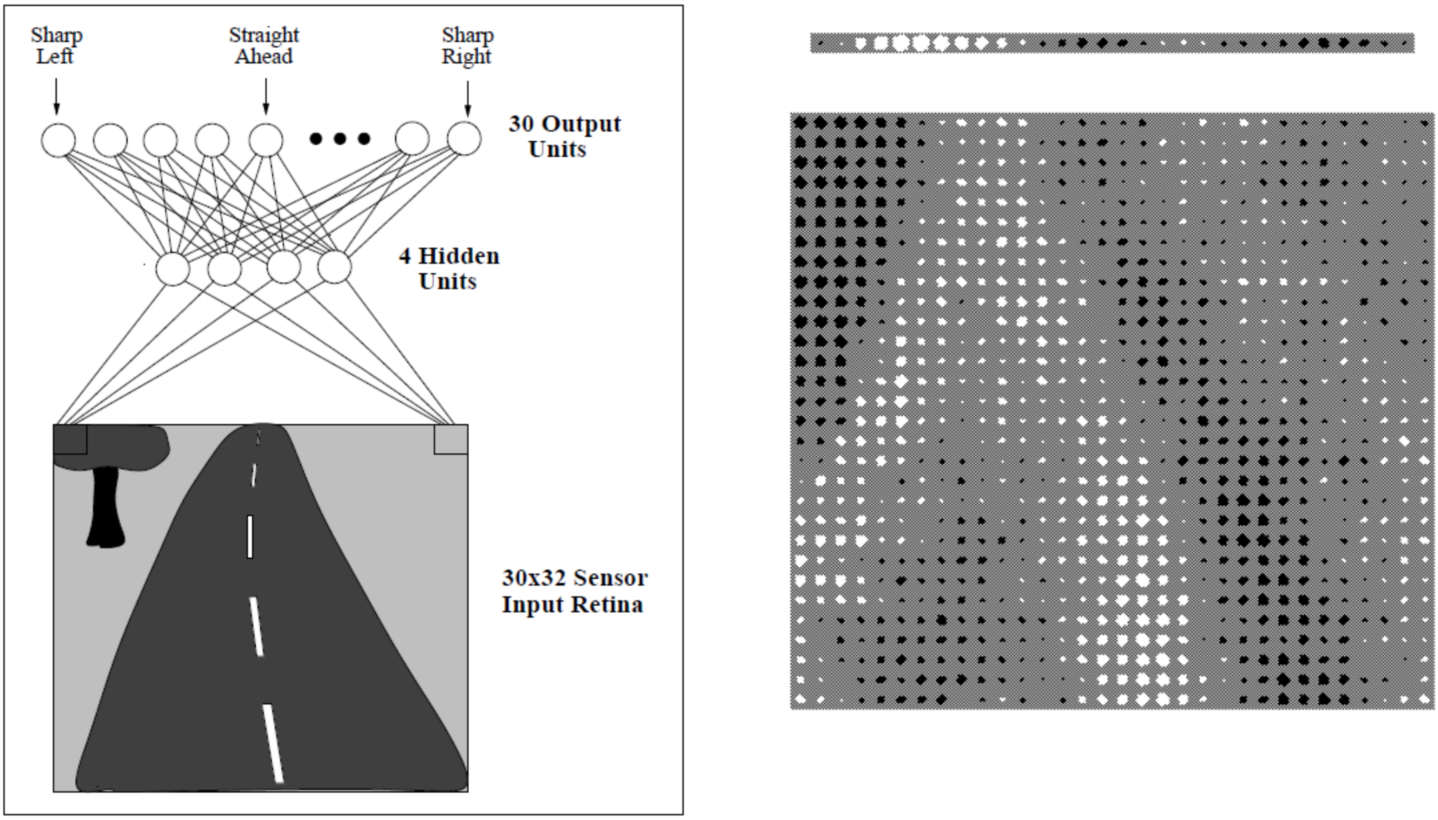



**Figure:** Neural network learning to steer an autonomous vehicle.

---

6b) repeated

## Adding Momentum

Because BACKPROPAGATION is such a widely used algorithm, many variations have been developed. The most common is to alter the weight-update rule the equation below

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

by making the weight update on the nth iteration depend partially on the update that occurred during the $(n - 1)^{th}$ iteration, as follows:

$$\Delta w_{ji}(n) = \eta \, \delta_j \, x_{ji} + \alpha \Delta w_{ji}(n-1)$$