

M.S. Ramaiah Institute of Technology
(Autonomous Institute, Affiliated to VTU)
Department of Computer Science and Engineering

Artificial Intelligence and Machine Learning

(CS52)

UNIT - 4

OUTLINE

2

- ▣ **Artificial Neural Networks** - Introduction, Neural Network Representation, Appropriate problems for Neural Network Learning, Perceptrons, Multilayer Networks and the Backpropagation algorithm. Chapter 5 of Textbook2
- ▣ **Bayesian Learning** - Introduction, Bayes theorem, Naive Bayes Classifier, The EM Algorithm.
Chapter 4 and 6(6.1,6.2,6.9,6.12) of TextBook2

Outline

3

Artificial Neural Networks

- Introduction
- Neural Network representation
- Appropriate problems
- Perceptrons
- Multilayer Networks and the Backpropagation algorithm

Introduction

What is Artificial Neural Network? Explain appropriate problems for Neural Network Learning with its characteristics.

4

- ANN stands for Artificial Neural Networks
- The inventor of the first neurocomputer is Dr. Robert Hecht-Nielsen
- ANN is a computing system made up of a number of simple, highly interconnected processing elements, which process information by their dynamic state response to external inputs.

Introduction

5

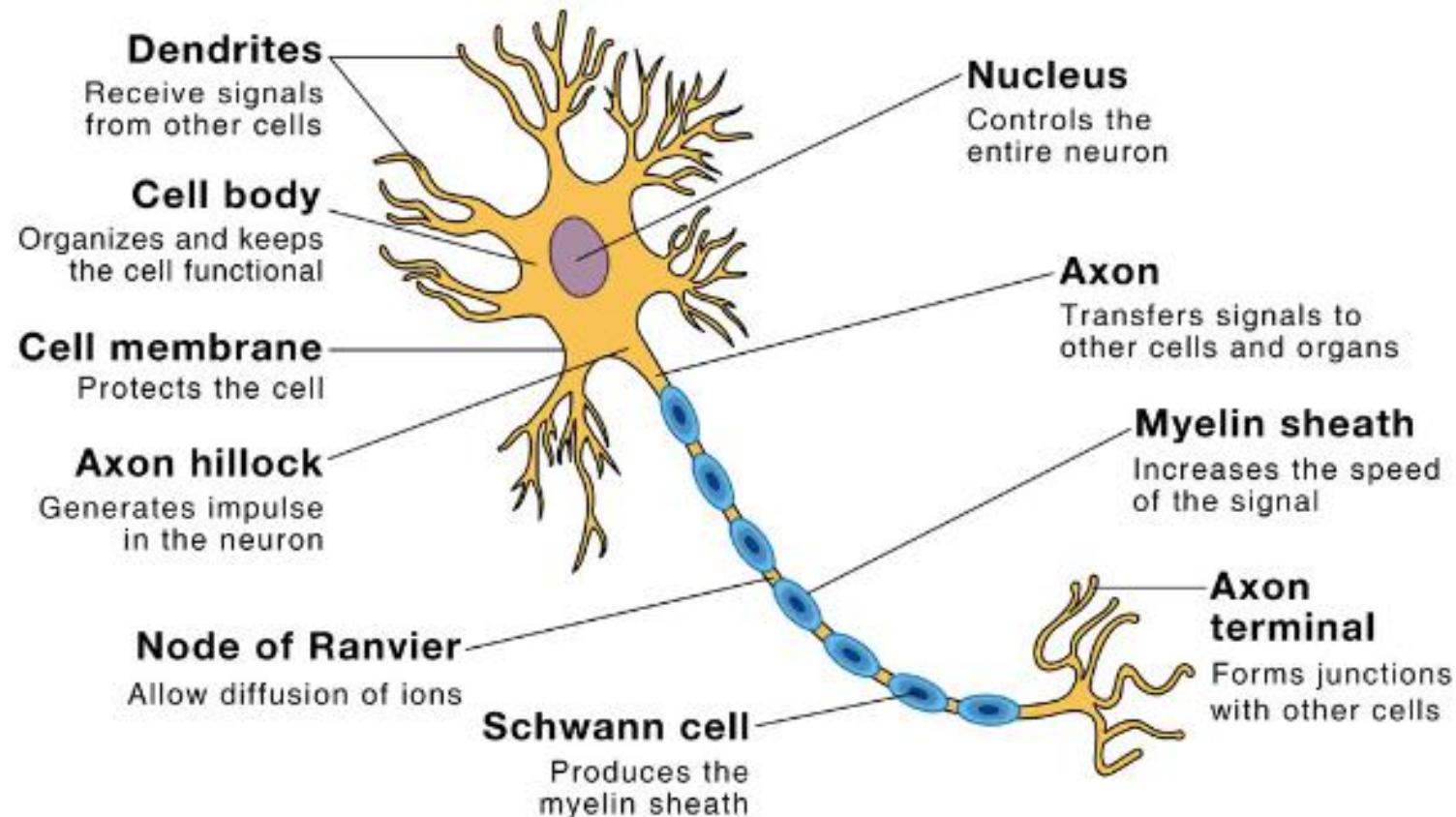
Biological Motivation

- Human information processing system consists of brain neuron: basic building block cell that communicates information to and from various parts of body
- The study of artificial neural networks (ANNs) has been **inspired by the observation that biological learning systems** are built of very complex webs of interconnected Neurons

Introduction

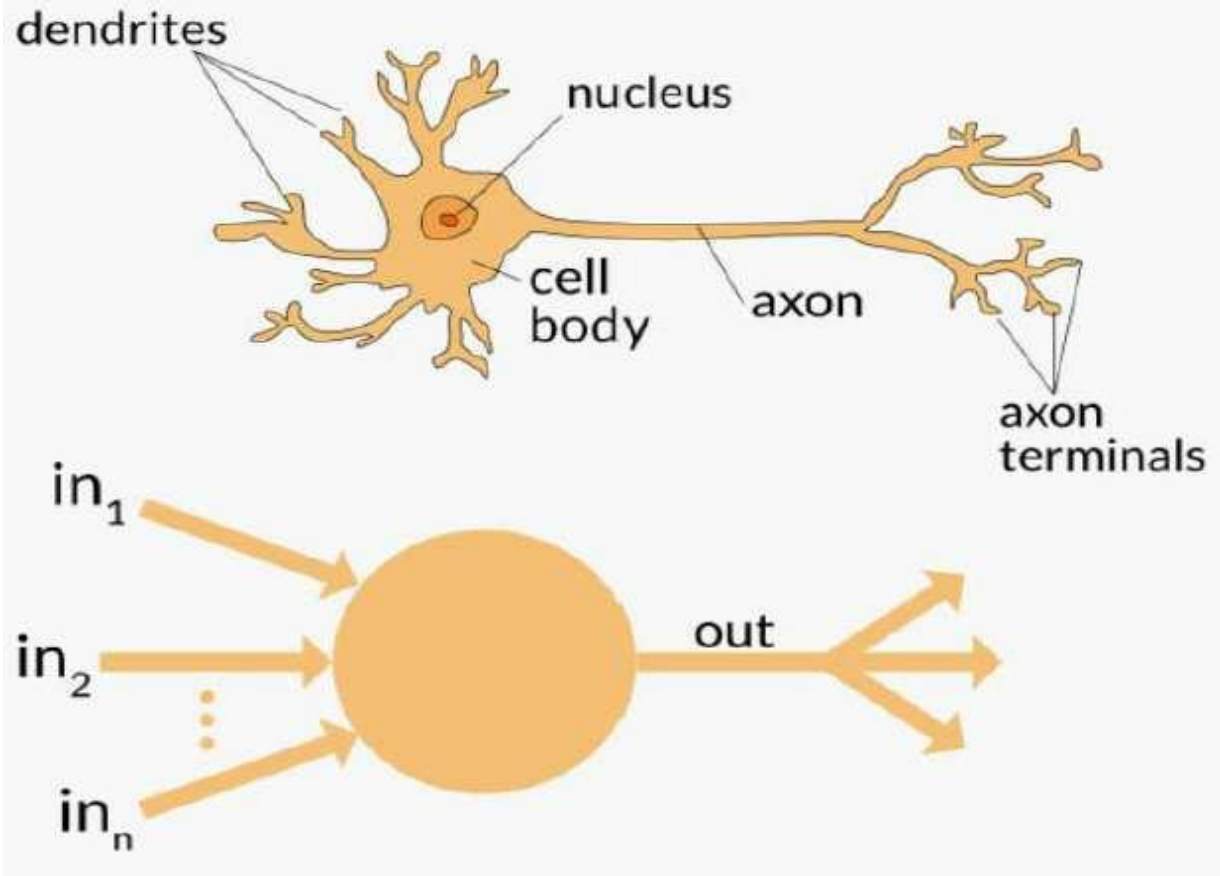
6

Parts of a Neuron with Functions



Introduction

7



Dendrites: Input
Cell body: Processor
Axon: Output

Introduction

8

Biological Motivation

□ Facts of Human Neurobiology

- ? Number of neurons $\sim 10^{11}$
- ? Connection per neuron $\sim 10^4 - 10^5$
- ? Neuron switching time ~ 0.001 second or 10^{-3}
- ? Scene recognition time ~ 0.1 second
- ? 100 inference steps doesn't seem like enough
- ? Highly parallel computation based on distributed representation

Introduction

9

Biological Motivation

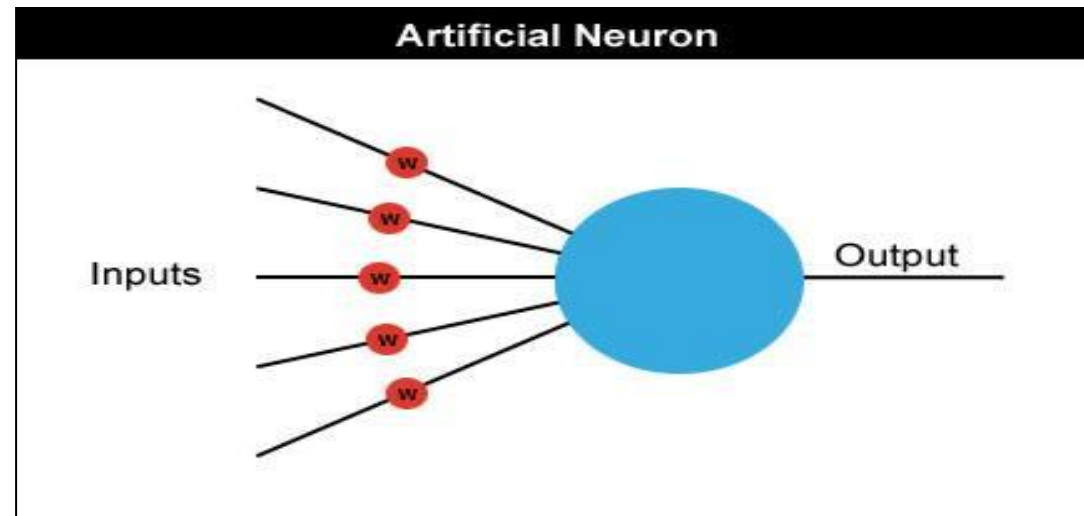
□ Properties of Neural Networks

- ? Many neuron-like threshold switching units
- ? Many weighted interconnections among units
- ? Highly parallel, distributed process
- ? Emphasis on tuning weights automatically
- ? Input is a high-dimensional discrete or real-valued (e.g, sensor input)

Neural Networks

10

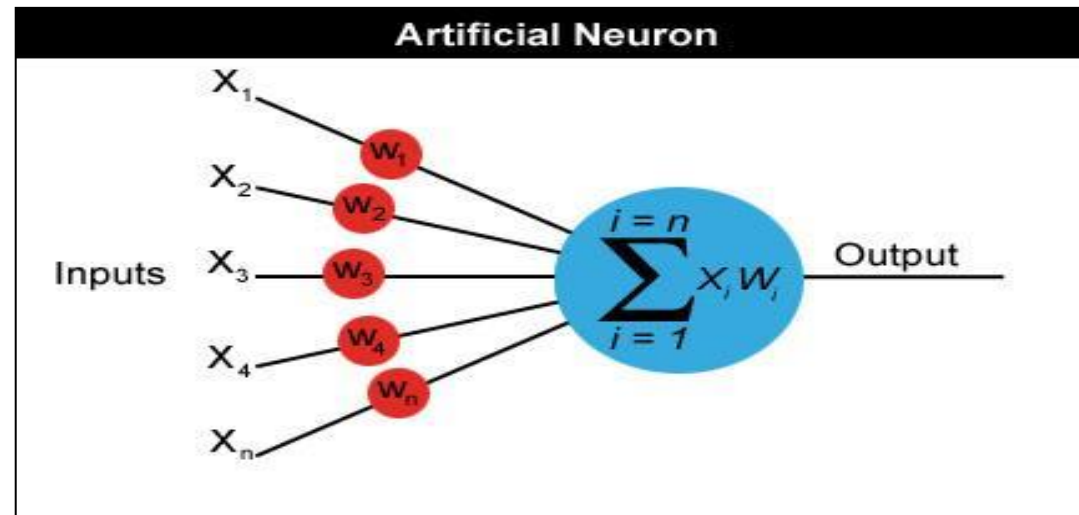
- Neural networks are made up of many artificial neurons.
- Each input into the neuron has its own weight associated with it illustrated by the red circle.
- A weight is simply a floating point number and it's these we adjust when we eventually come to train the network.



Neural networks

11

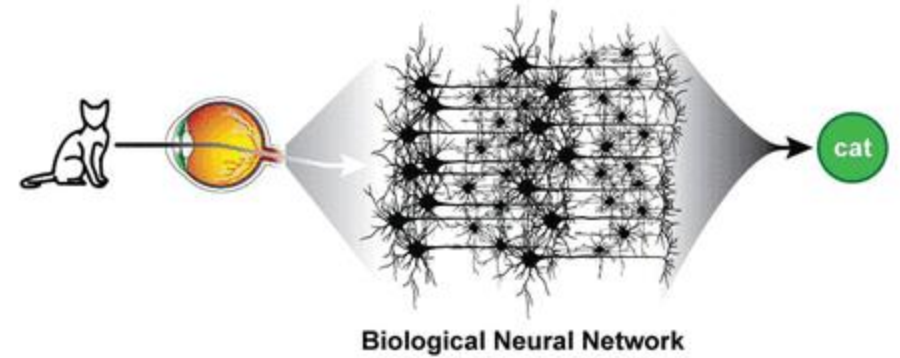
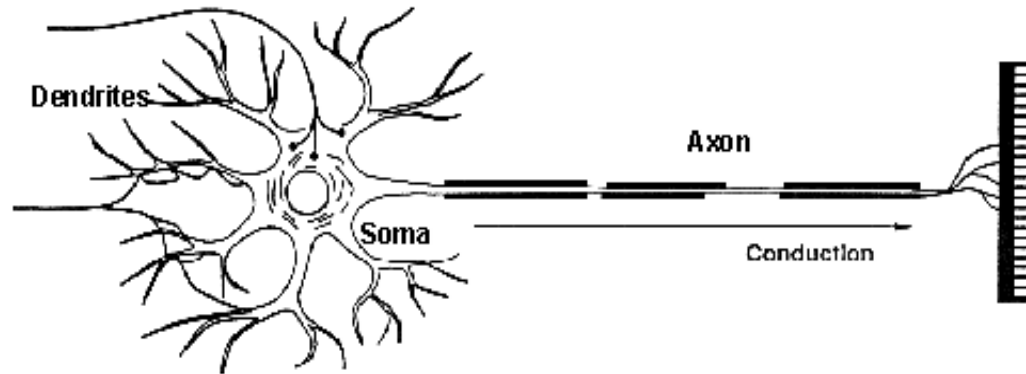
- A neuron can have any number of inputs from one to n , where n is the total number of inputs.
- The inputs may be represented therefore as $x_1, x_2, x_3 \dots x_n$.
- And the corresponding weights for the inputs as $w_1, w_2, w_3 \dots w_n$.
- Output $a = x_1w_1 + x_2w_2 + x_3w_3 \dots + x_nw_n$



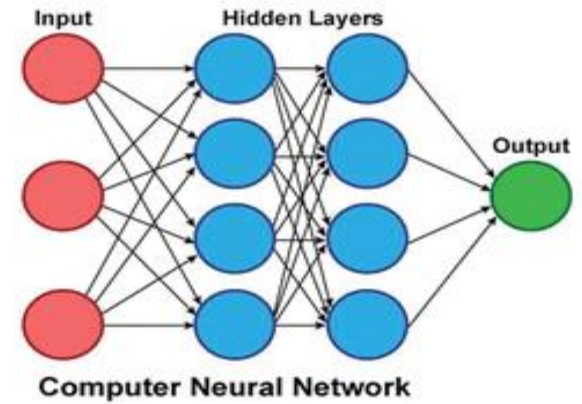
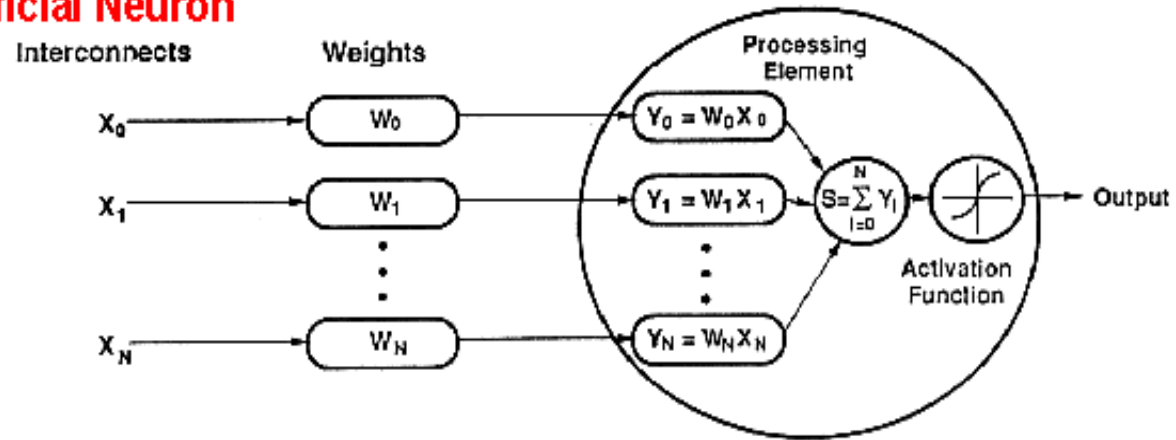
Introduction

12

Biological Neuron

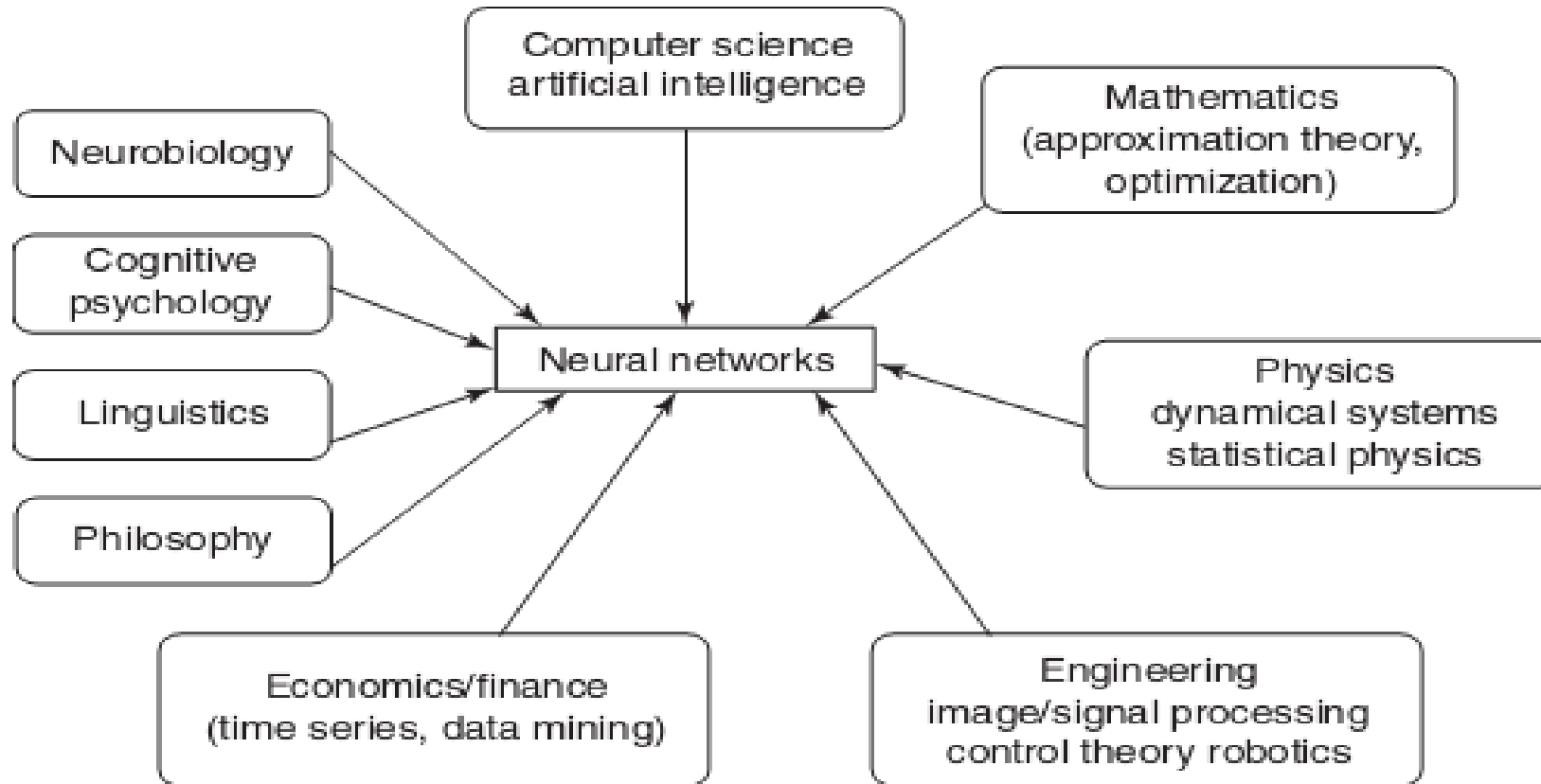


Artificial Neuron



MULTIDISCIPLINARY VIEW OF NEURAL NETWORKS

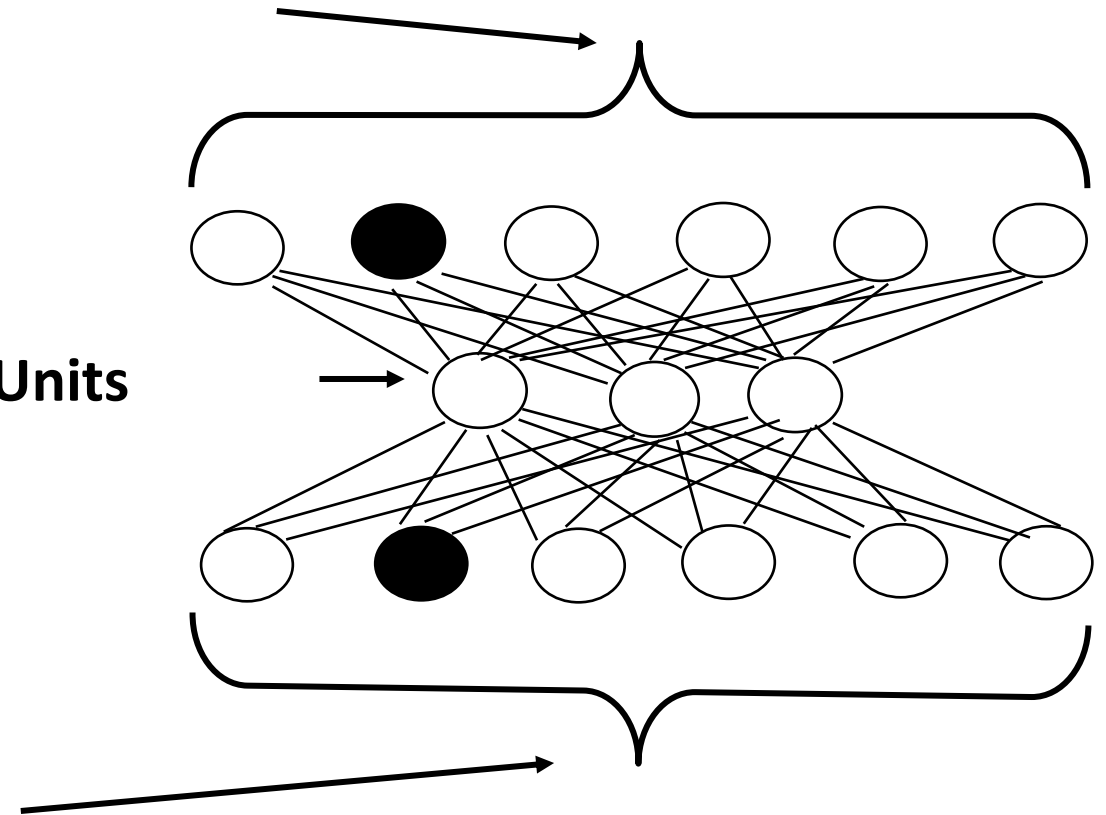
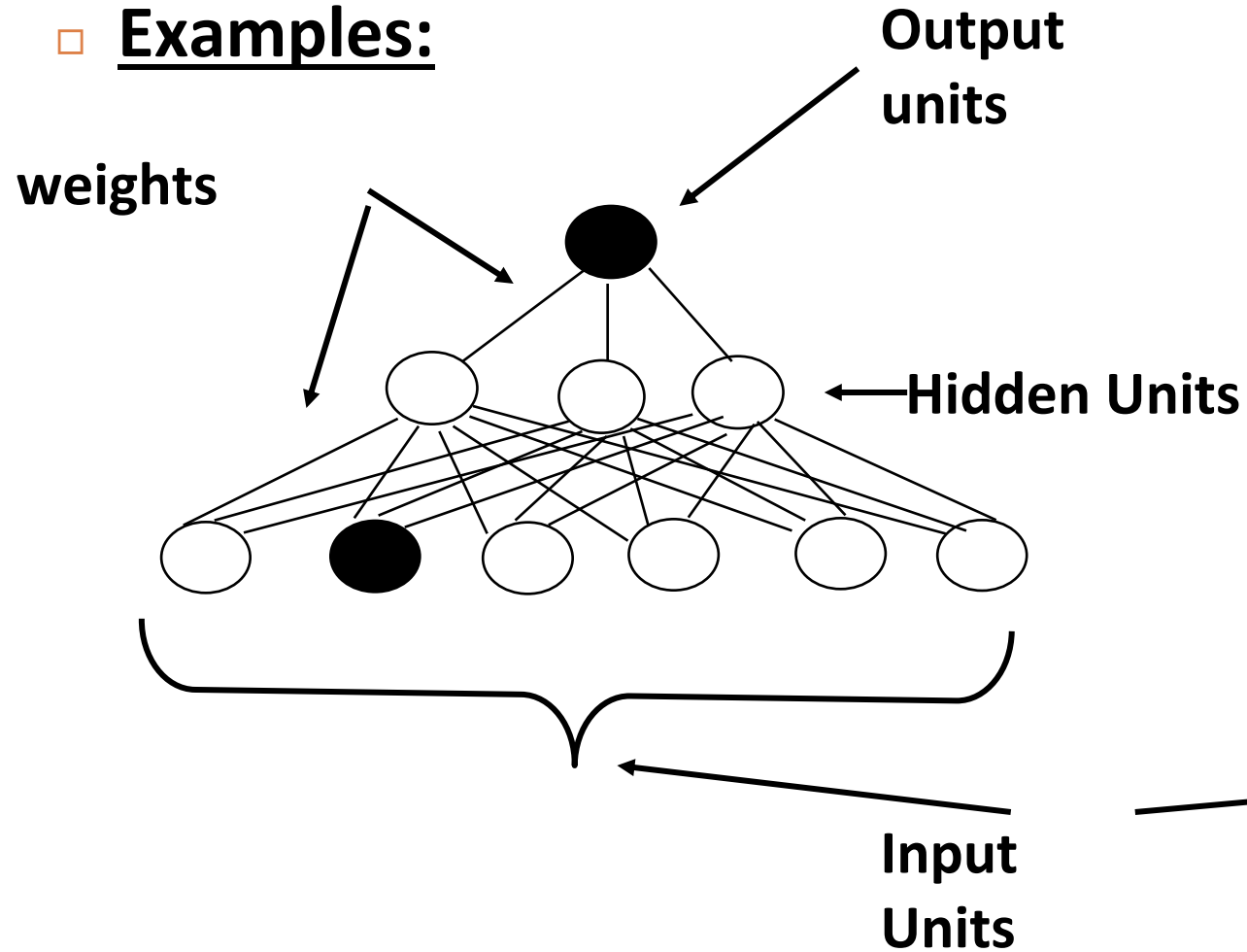
13



Neural Network Representation

14

□ Examples:



Neural Network Representation

15

Learning in Multilayer Neural Networks

- Learning consists of searching through the space of all possible matrices of weight values for a combination of weights that satisfies a database of positive and negative examples (multi-class as well as regression problems are possible).

Neural Network Representation

16

Learning in Multilayer Neural Networks

- Neural Network model with a set of adjustable weights defines a restricted hypothesis space corresponding to a family of functions.
- The size of this hypothesis space can be increased or decreased by increasing or decreasing the number of hidden units present in the network.

NEURAL NETWORK REPRESENTATIONS

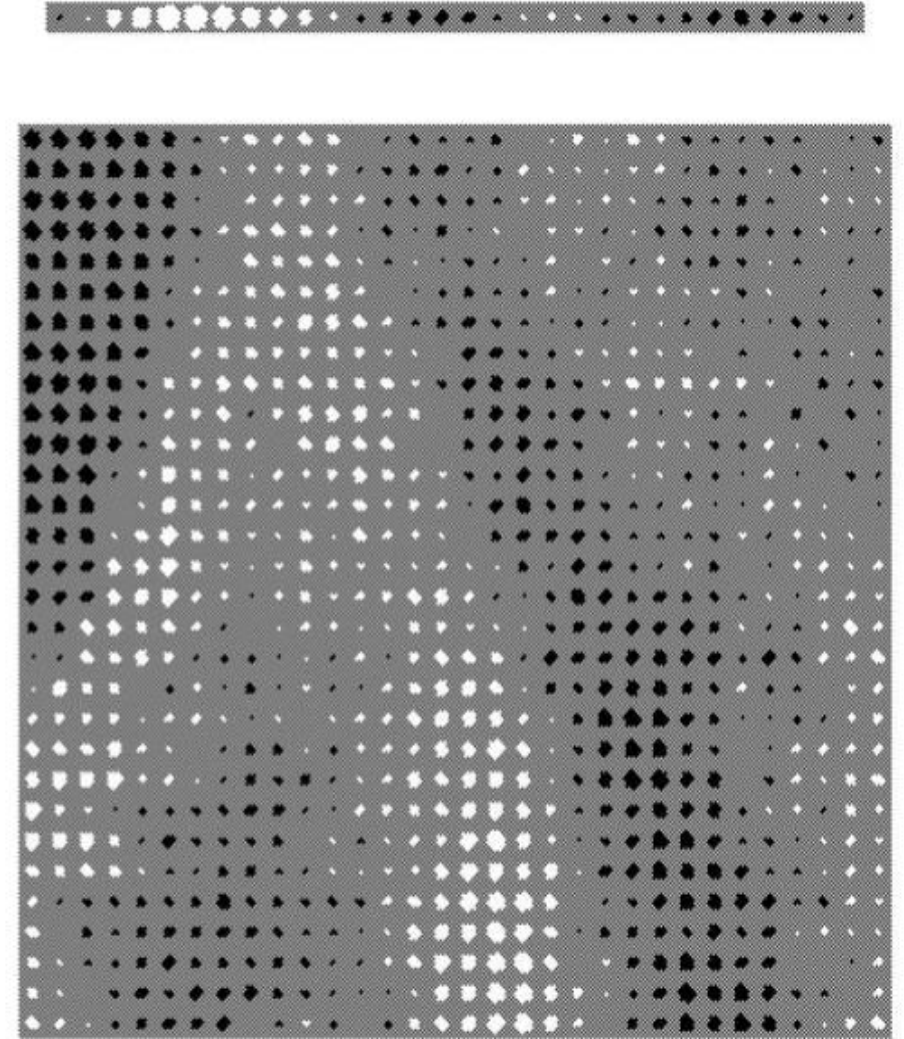
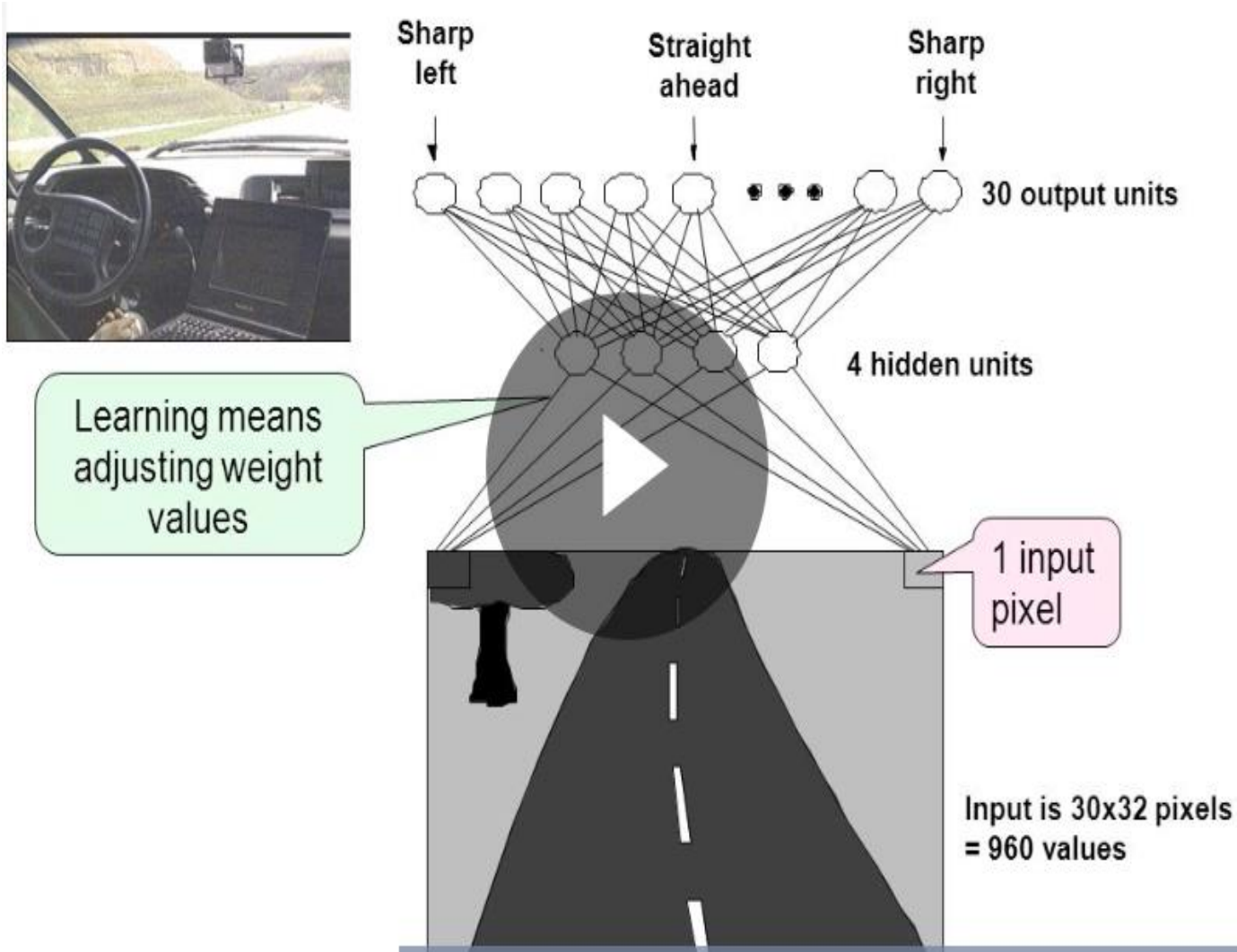
17

- A prototypical example of ANN learning is provided by Pomerleau's system ALVINN.
- ANN is trained to mimic the observed steering commands of human driving the vehicle for approx 5 minutes
- ALVINN uses a learned ANN to **steer an autonomous vehicle driving** at normal speeds on public highways.



NEURAL NETWORK REPRESENTATIONS

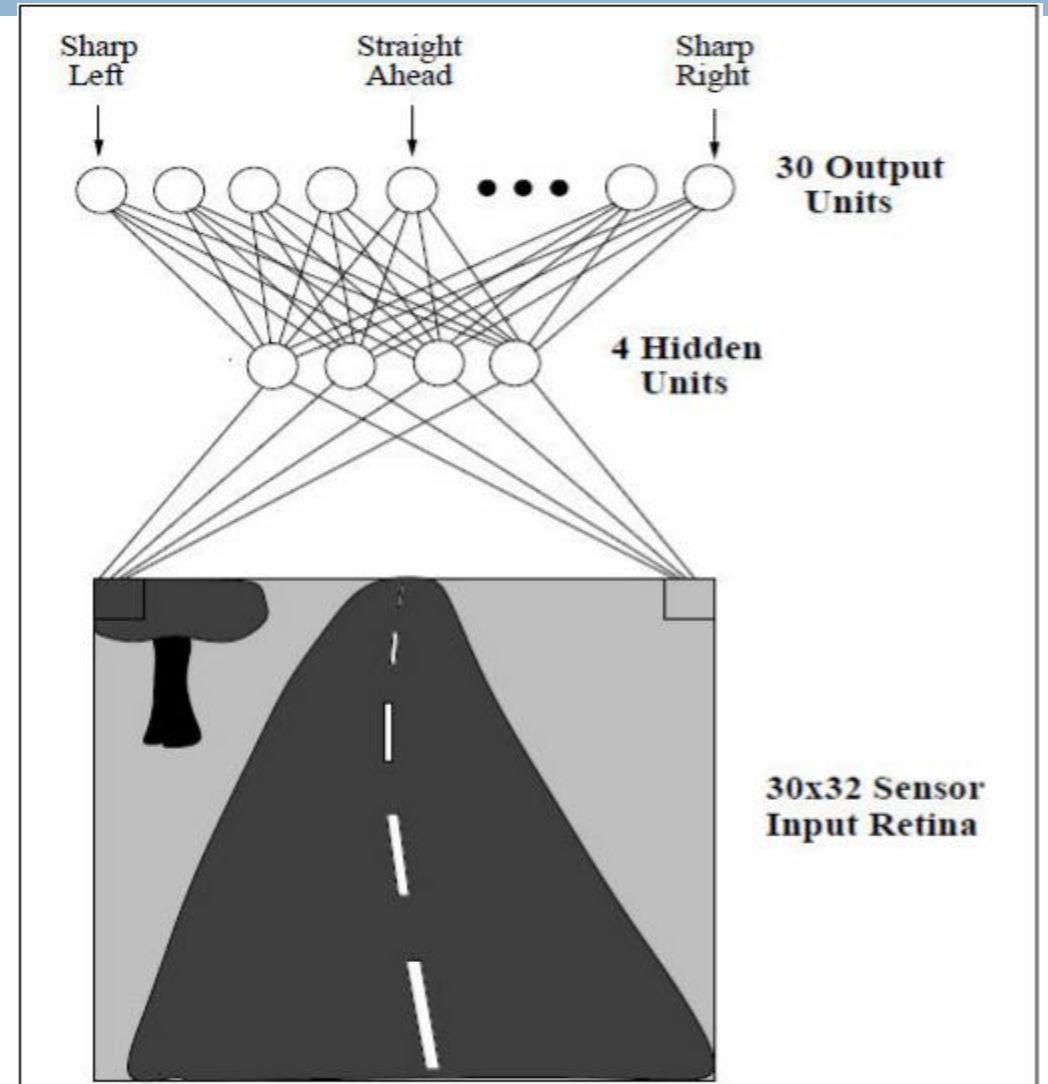
18



NEURAL NETWORK REPRESENTATIONS

19

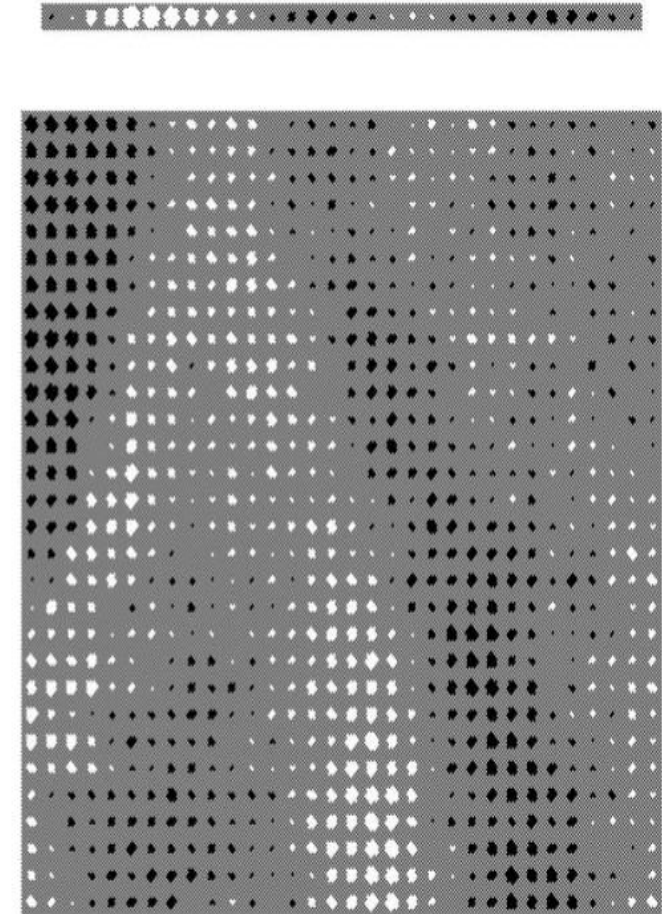
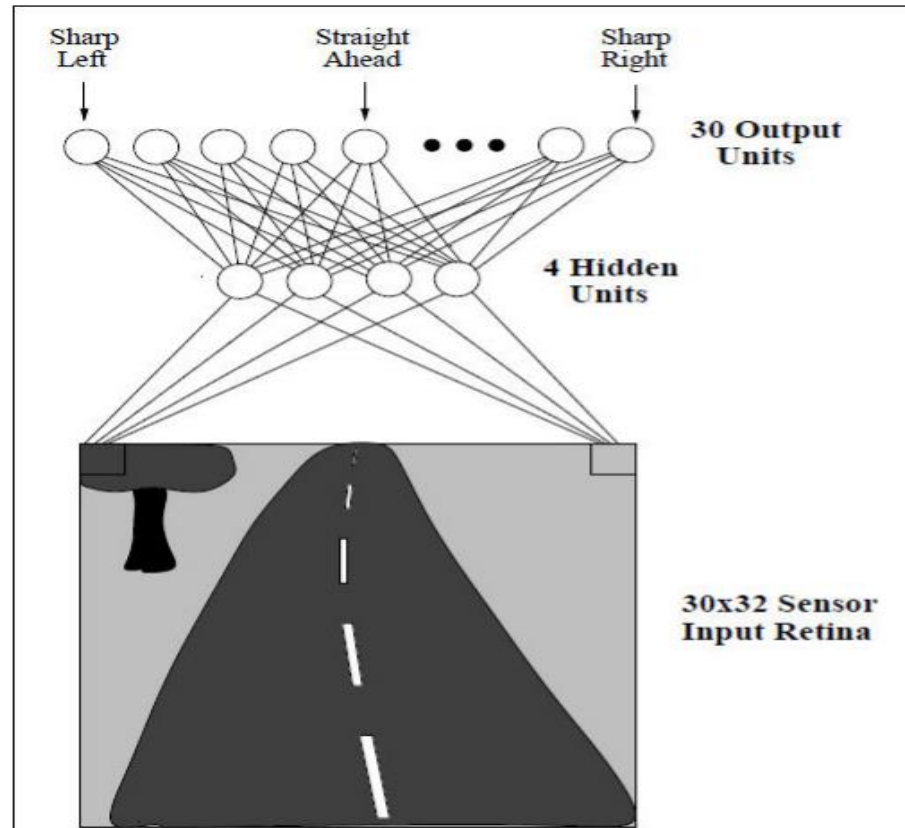
- The **input** to the neural network is a 30x32 grid of pixel intensities obtained from a forward-pointed camera mounted on the vehicle.
- The network **output** is the direction in which the vehicle is steered.
- **Hidden units** computes a single real-valued output based on weighted combination of its 960 inputs



NEURAL NETWORK REPRESENTATIONS

20

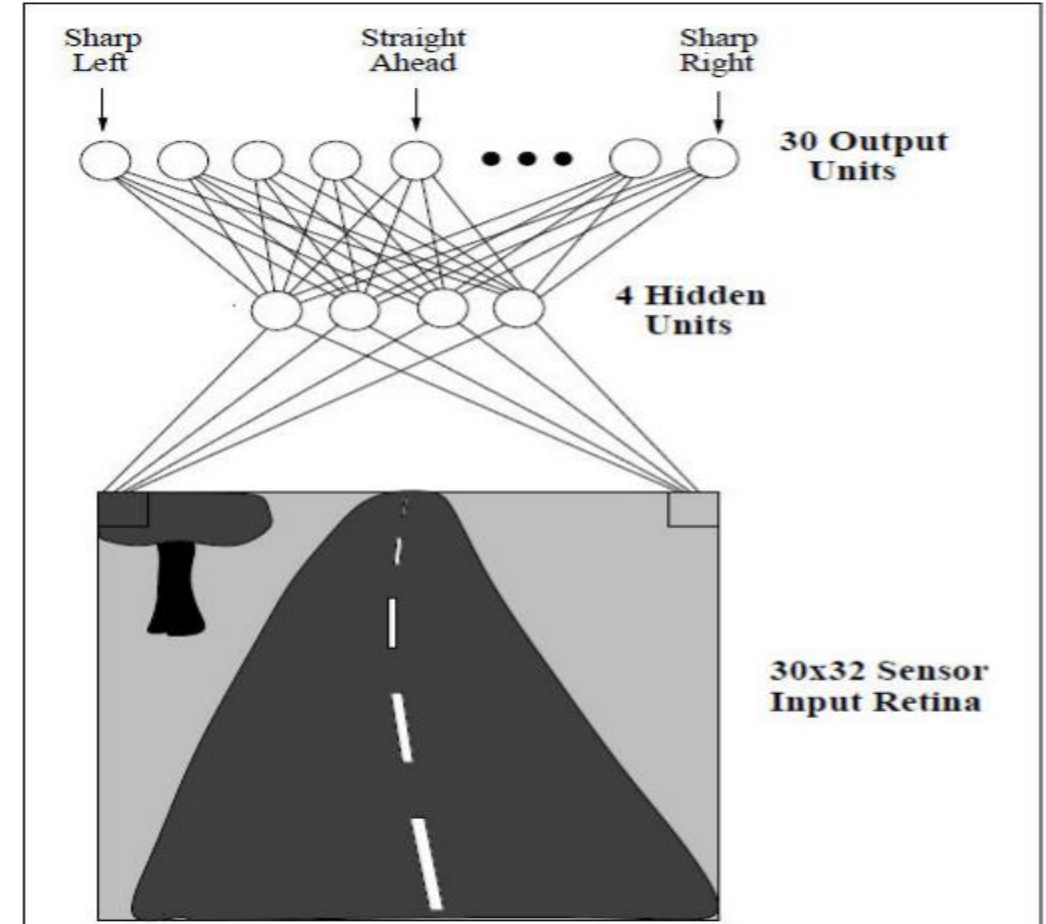
- Matrix of White and black box
 - ? White box indicates +ve weight
 - ? Black box indicate -ve weight
 - ? Rectangle Size indicate the weight magnitude.
- The smaller rectangular diagram shows the weights from this hidden unit to each of the 30 output units.



NEURAL NETWORK REPRESENTATIONS

21

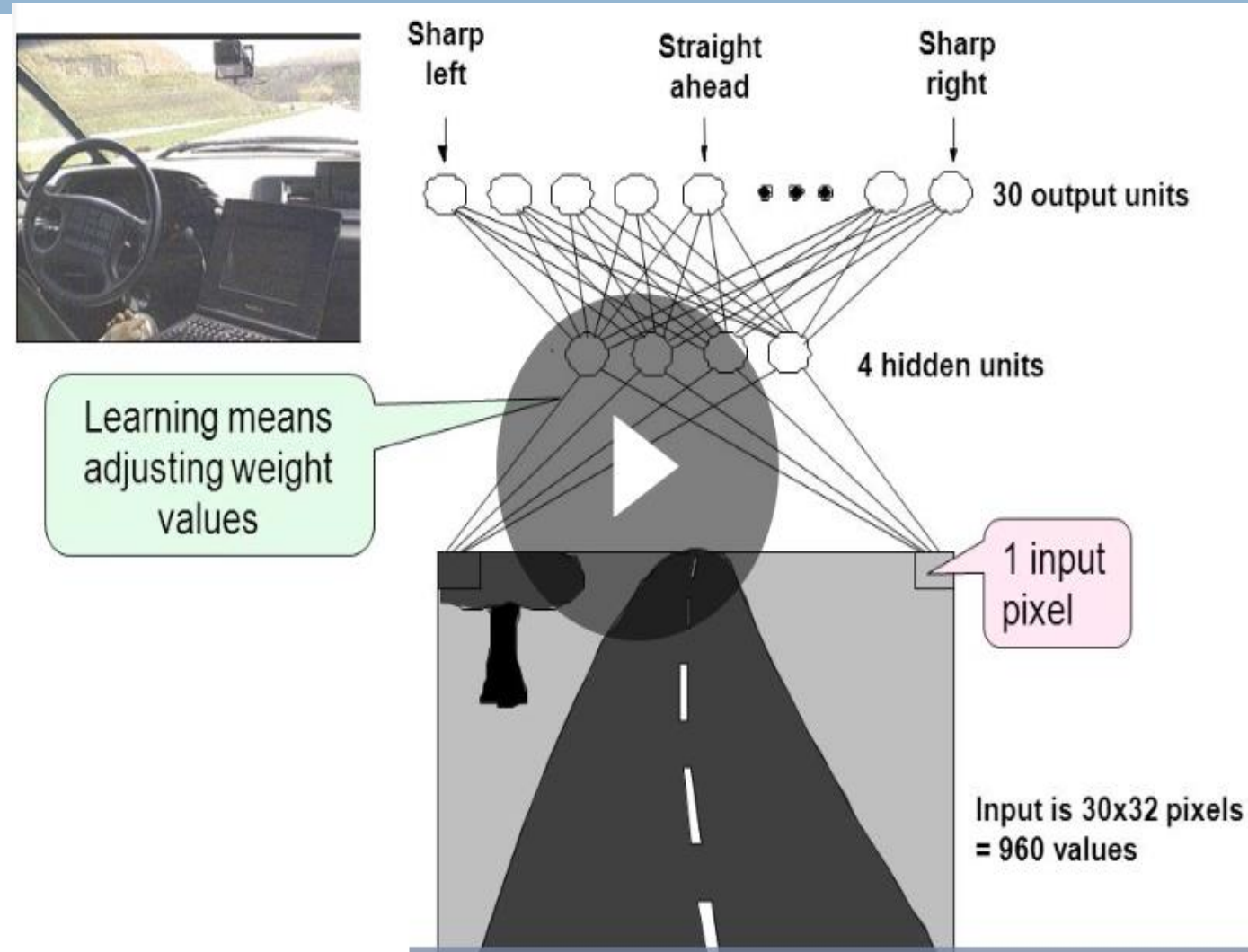
- There are **four units** that receive inputs directly from all of the 30 x 32 pixels in the image.
- These are called "**hidden**" units because their output is available only within the network
- Each of these four hidden units computes a single real-valued output based on a weighted combination of its 960 inputs



NEURAL NETWORK REPRESENTATIONS

22

- These hidden unit outputs are then used as inputs to a second layer of 30 "output" units.
- Each output unit corresponds to a particular steering direction, and the output values of these units determine which steering direction is recommended most strongly.



Appropriate Problems for Neural Network Learning

23

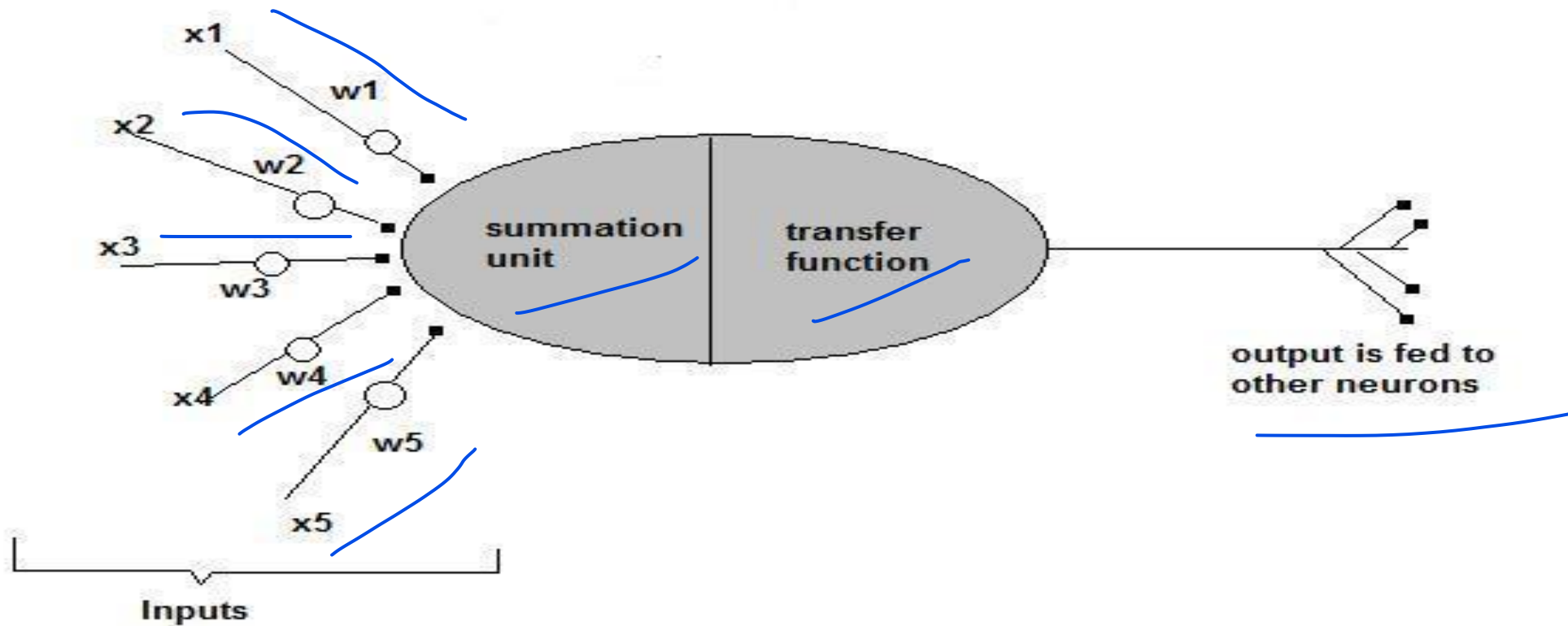
- Instances are represented by many attribute-value pairs
- The target function output may be discrete-valued, real-valued, or a vector of several real- or discrete-valued attributes.
- The training examples may contain errors.
- Long training times are acceptable.
- Fast evaluation of the learned target function may be required.
- The ability for humans to understand the learned target function is not important.

How do ANNs work?

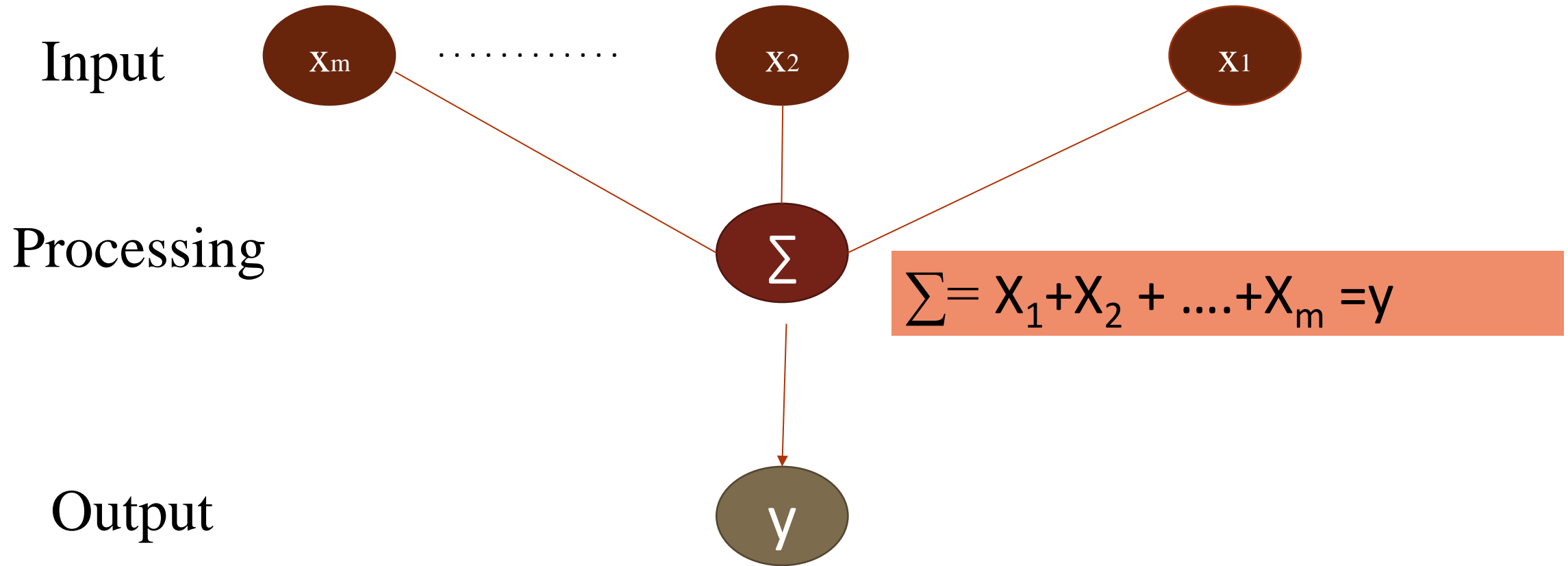
Describe working of an Artificial Neural Network (ANN).

- Now, let us have a look at the model of an artificial neuron.

A Single Neuron

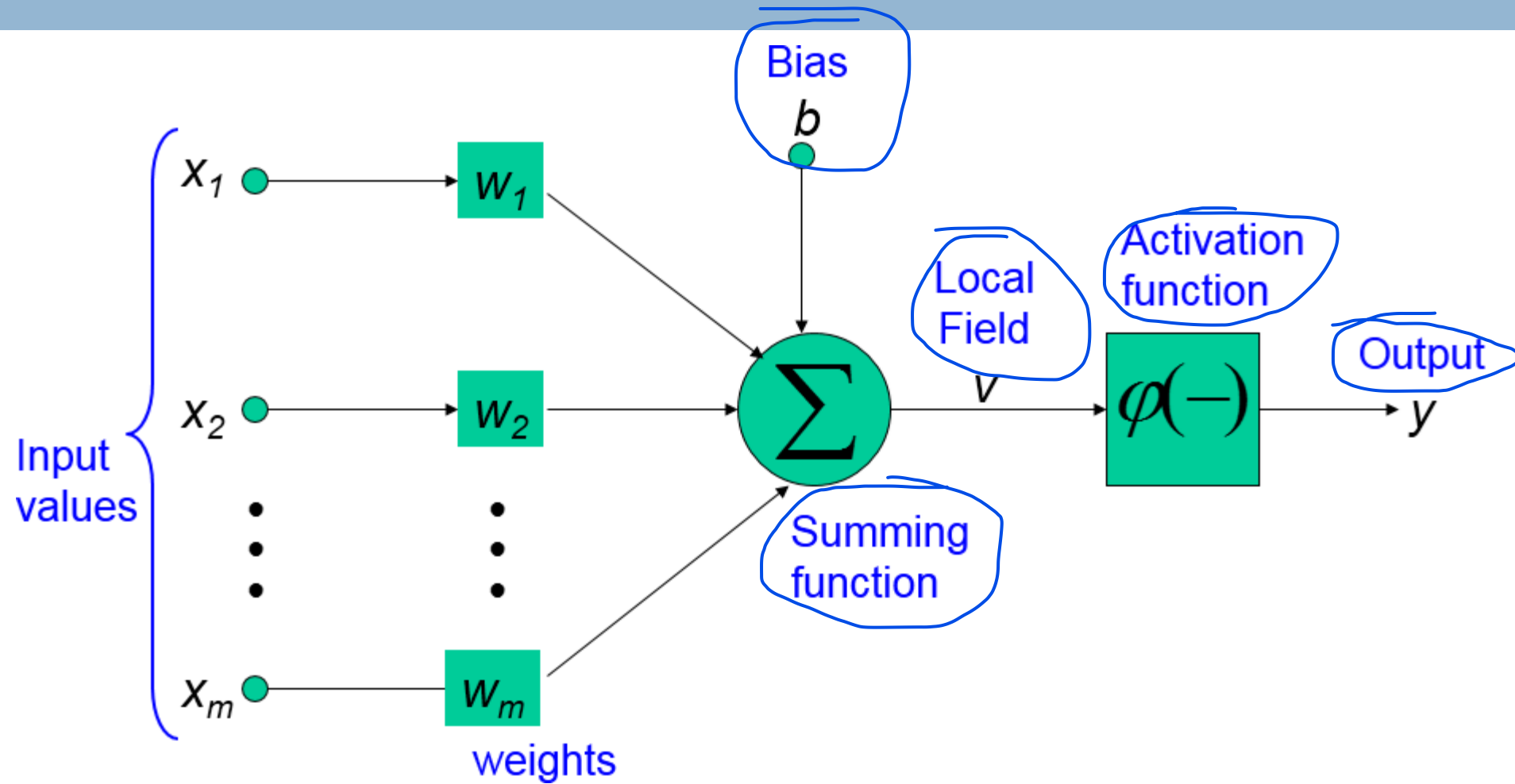


How do ANNs work?



Neuron

26

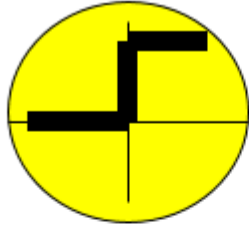


Activation Functions

- Use different functions to obtain different models.
- 3 most common choices :
 - 1) Step function
 - 2) Sign function
 - 3) Sigmoid function
- An output of **1 represents firing** of a neuron down the axon.

Neuron

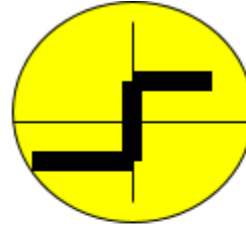
28



Step function

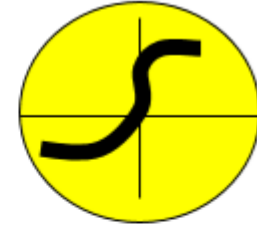
(Linear Threshold Unit)

$$\text{step}(x) = \begin{cases} 1, & \text{if } x \geq \text{threshold} \\ 0, & \text{if } x < \text{threshold} \end{cases}$$



Sign function

$$\text{sign}(x) = \begin{cases} +1, & \text{if } x \geq 0 \\ -1, & \text{if } x < 0 \end{cases}$$



Sigmoid function

$$\text{sigmoid}(x) = 1/(1+e^{-x})$$

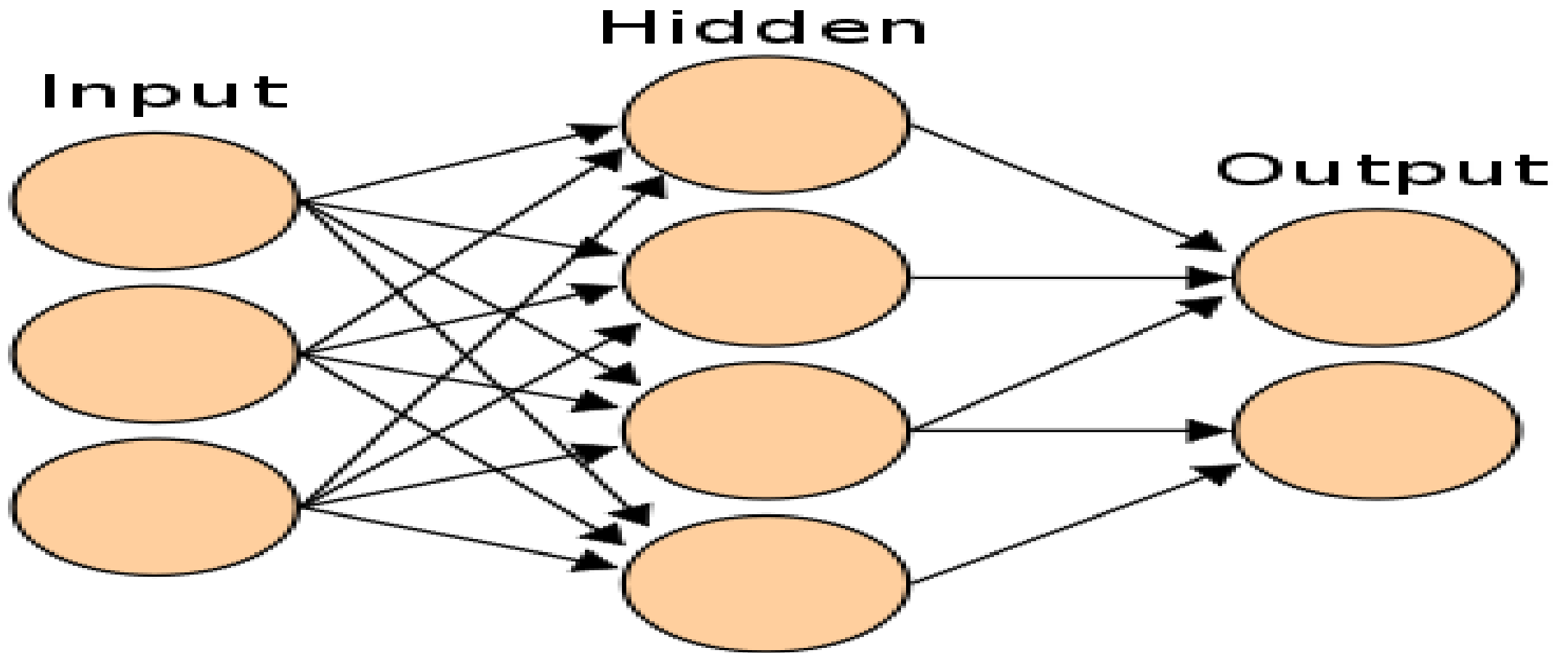
Describe the structure and working of an Artificial Neural Network (ANN).

29

Standard structure of an artificial neural network

- **Input units**
 - ? represents the input as a fixed-length vector of numbers (user defined)
- **Hidden units**
 - ? calculate thresholded weighted sums of the inputs
 - ? represent intermediate calculations that the network learns
- **Output units**
 - ? represent the output as a fixed length vector of numbers

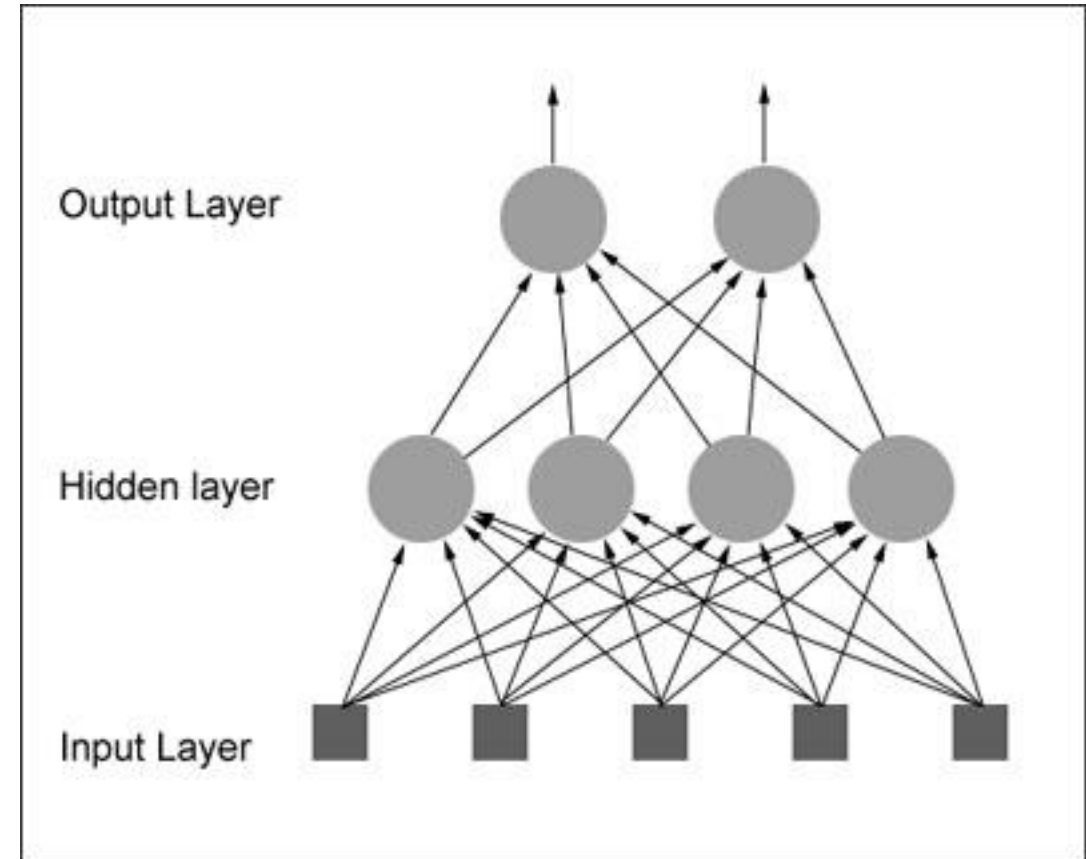
Three types of layers: Input, Hidden, and Output



How do we actually *use* an artificial neuron?

31

- ❑ Feedforward network:
- ❑ The neurons in each layer feed their output forward to the next layer until we get the final output from the neural network.
- ❑ There can be any number of hidden layers can exist within a feedforward network.
- ❑ The number of neurons can be completely arbitrary.



Outline

32

- Introduction
- Neural Network representation
- Appropriate problems
- Perceptrons
- Backpropagation algorithm

Perceptron

33

- One type of ANN system is based on a unit called a perceptron, illustrated in Figure

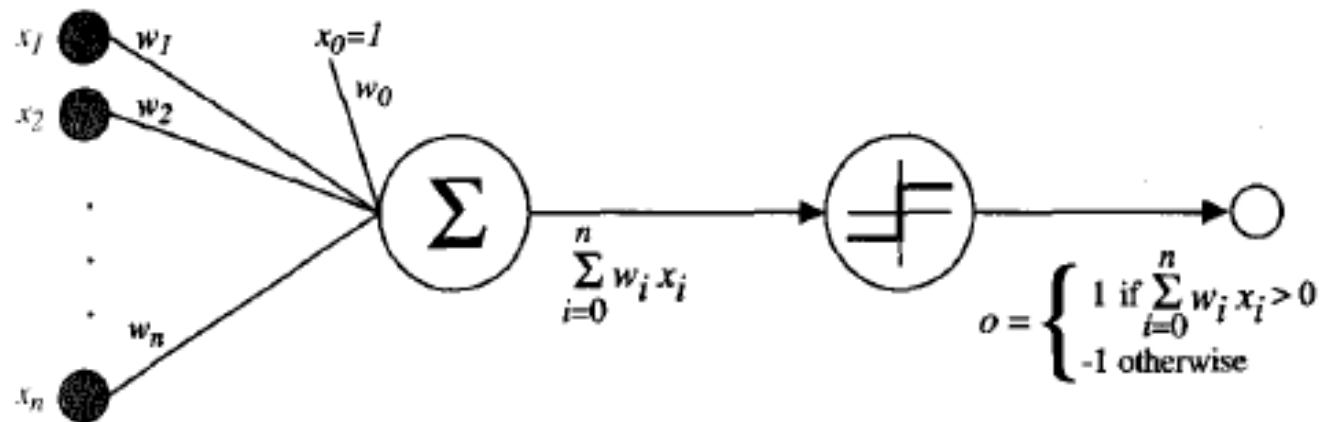


FIGURE 4.2
A perceptron.

Perceptron

34

- 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

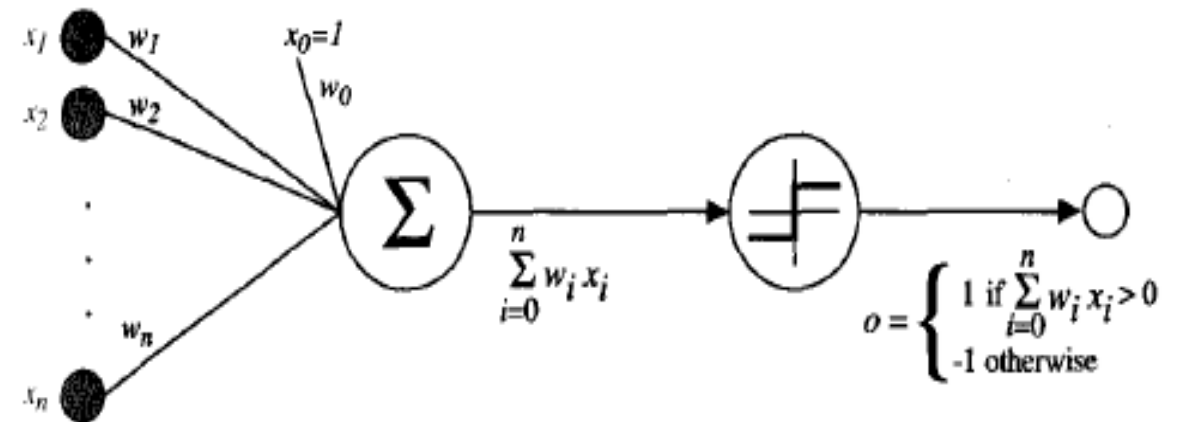


FIGURE 4.2

A perceptron.

Perceptron

35

- Given inputs x_1 through x_n the output $o(x_1, \dots, x_n)$ computed by the perceptron is

$$o(x_1, \dots, x_n) = \begin{cases} 1 & \text{if } w_0 + w_1x_1 + w_2x_2 + \dots + w_nx_n > 0 \\ -1 & \text{otherwise} \end{cases}$$

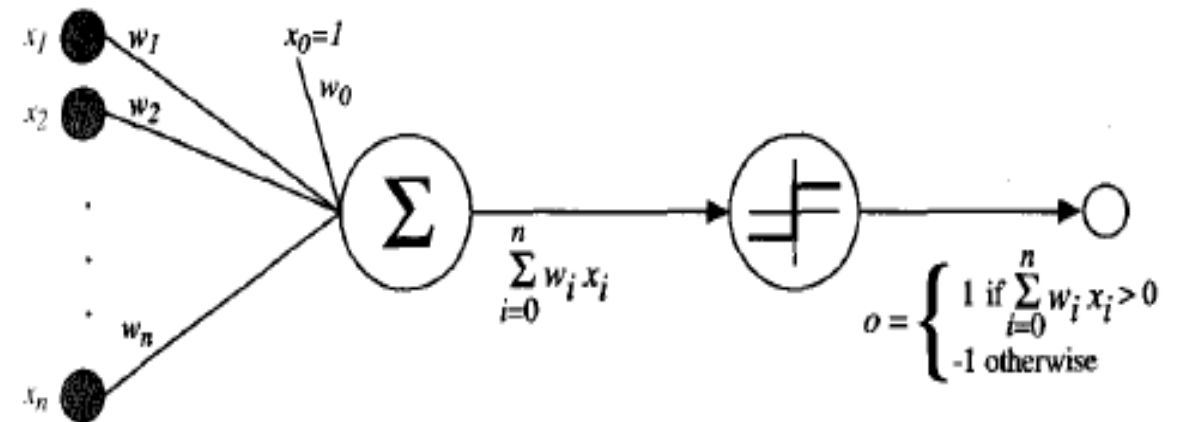
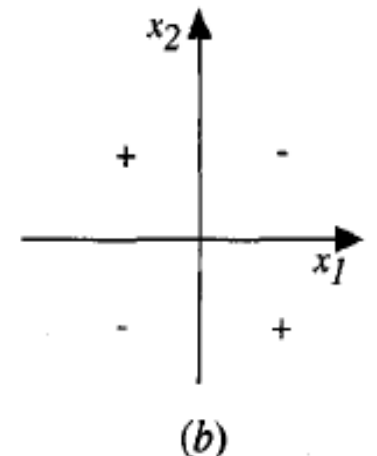
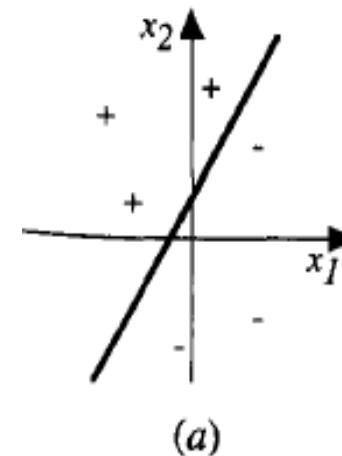


FIGURE 4.2
A perceptron.

Representational Power of Perceptrons

36

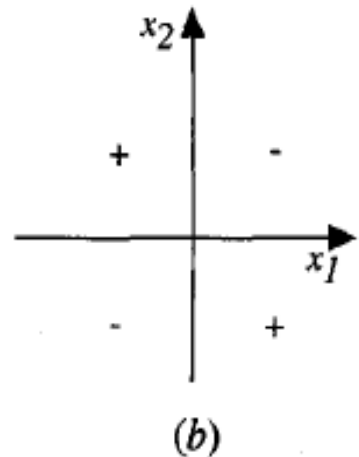
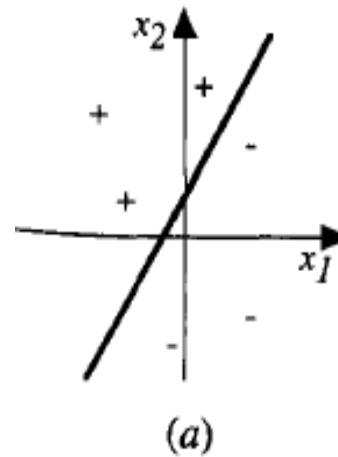
- We can view the perceptron as representing a hyperplane decision surface in the n -dimensional space of instances (i.e., points).
- The perceptron outputs a 1 for instances lying on one side of the hyperplane and outputs a -1 for instances lying on the other side, as illustrated in Figure



Representational Power of Perceptrons

37

- The equation for this decision hyperplane is $\vec{w} \cdot \vec{x} = 0$.
- Of course, some sets of positive and negative examples cannot be separated by any hyperplane. Those that can be separated are called linearly separable sets of examples.



Representational Power of Perceptrons

38

- A single perceptron can be used to represent many boolean functions.
- Perceptrons can represent all of the primitive boolean functions AND, OR, NAND (1 AND), and NOR (1 OR).

| x_1 | x_2 | Y |
|-------|-------|-----|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

Suppose that we are going to work on AND Gate problem. The gate returns if and only if both inputs are true. Let's say that $w_1 = 0.9$ and $w_2 = 0.9$. Activation threshold would be 0.5.

39

| x_1 | x_2 | $\Sigma = x_1 * w_1 + x_2 * w_2$ | Prediction output | $w = w + \alpha * \varepsilon$ | Final target output |
|-------|-------|------------------------------------------|-------------------|------------------------------------------------------------------------------------------------------|---------------------|
| 0 | 0 | $0 * 0.9 + 0 * 0.9 = 0$ $0 < 0.5$ | 0 | | 0 |
| 0 | 1 | $0 * 0.9 + 1 * 0.9 = 0.9$ $0.9 > 0.5$ | 1 0 | Error = $0 - 1 = -1$ $w_1 = w_1 + \alpha * \varepsilon$ $= 0.9 + 0.5(-1)$ $= 0.4 < 0.5$ | 0 |
| 1 | 0 | $1 * 0.4 + 0 * 0.4 = 0.4$ $0.4 < 0.5$ | 0 | | 0 |
| 1 | 1 | $1 * 0.4 + 1 * 0.4 = 0.8$ $0.8 > 0.5$ | 1 | | 1 |

The Perceptron Training Rule

40

- How to learn the weights for a single perceptron.
- Here the precise learning problem is to determine a weight vector that causes the perceptron to produce the correct ± 1 output for each of the given training examples.

The Perceptron Training Rule

41

- Several algorithms are known to solve this learning problem.
 - ? Perceptron rule
 - ? delta rule
- They are important to ANNs because they provide the basis for learning networks of many units.

The Perceptron Training Rule

42

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.
- This process is repeated, iterating through the training examples as many times as needed until the perceptron classifies all training examples correctly.

The Perceptron Training Rule

43

Perceptron Rule

- Weights are modified at each step according to the perceptron training rule, which revises the weight w_i associated with input x_i according to the rule

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

$$\text{where } \Delta w_i = \eta(t - o)x_i$$

- 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 Perceptron Training Rule

44

Perceptron Rule

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

- η is a positive constant called the **learning rate**.
- Learning rate 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

The Perceptron Training Rule

45

Perceptron Rule

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

- Why should this update rule converge toward successful weight values
 - ? Suppose the training example is correctly classified already by the perceptron. In this case, $(t - o)$ is zero, making zero, so that no weights are updated.
 - ? Suppose the perceptron outputs a -1, when target output is + 1. To make the perceptron output a + 1 instead of - 1 in this case, the weights must be altered to increase the value $\vec{w} \cdot \vec{x}$.

The Perceptron Training Rule

46

Gradient Descent and the Delta Rule

- 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.

The Perceptron Training Rule

47

Gradient Descent and the Delta Rule

- This rule is important because gradient descent provides the basis for the BACKPROPAGATION 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 through hypothesis spaces containing many different types of continuously parameterized hypotheses.

The Perceptron Training Rule

48

Delta Rule

- Considering the task of training an *unthresholded perceptron*; that is, a linear unit for which the output o is given by

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

- linear unit corresponds to the first stage of a perceptron, without the threshold.

The Perceptron Training Rule

49

Delta Rule

- Derive a weight learning rule for linear units, specifying a measure for the training error of a hypothesis (weight vector), relative to the training examples.
- Define error,
 - ? 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 .

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

DERIVATION OF THE GRADIENT DESCENT RULE

50

How can we calculate the direction of steepest descent along the error surface?

- Compute the derivative of E with respect to each component of the vector
- This vector derivative is called the gradient of E with respect to \vec{w} , written

\vec{w} .

$\nabla E(\vec{w})$.

$$\nabla E(\vec{w}) \equiv \left[\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \dots, \frac{\partial E}{\partial w_n} \right]$$

- whose components are the partial derivatives of E with respect to each of the w_i .

DERIVATION OF THE GRADIENT DESCENT RULE

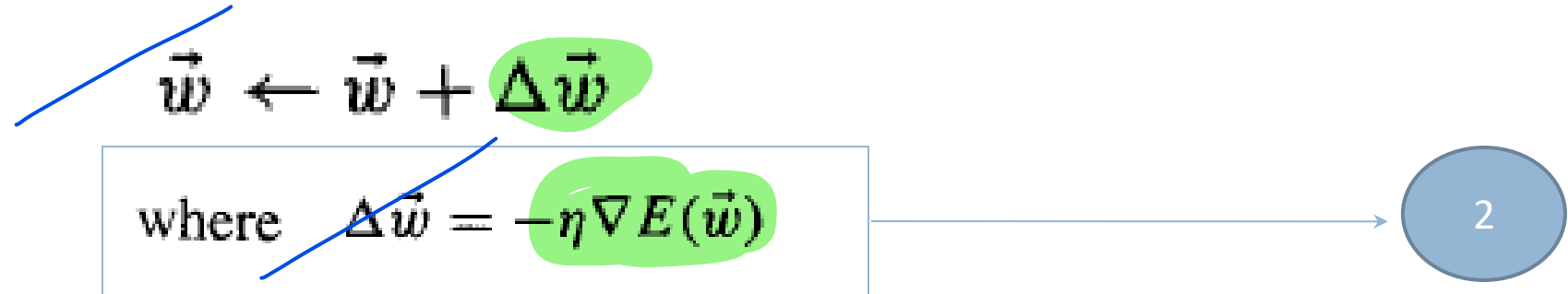
51

How can we calculate the direction of steepest descent along the error surface?

- 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 $\Delta \vec{w} = -\eta \nabla E(\vec{w})$



- 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** .

DERIVATION OF THE GRADIENT DESCENT RULE

52

How can we calculate the direction of steepest descent along the error surface?

$$\Delta \vec{w} = -\eta \nabla E(\vec{w})$$

2

$$\nabla E(\vec{w}) \equiv \left[\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \dots, \frac{\partial E}{\partial w_n} \right]$$

1

- Using Equation 1 now Equation 2 can be rewritten as

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

3

DERIVATION OF THE GRADIENT DESCENT RULE

53

How can we calculate the direction of steepest descent along the error surface?

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

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

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

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

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

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

4

DERIVATION OF THE GRADIENT DESCENT RULE

54

How can we calculate the direction of steepest descent along the error surface?

- Substitute equation 4 in equation 3 gives weight update rule for gradient descent

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

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

GRADIENT-DESCENT(*training_examples*, η)

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

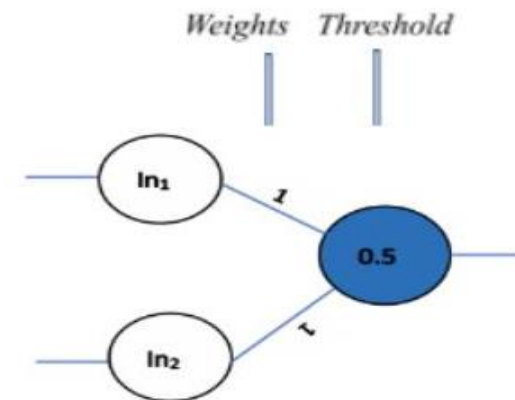
- Initialize each w_i to some small random value
- Until the termination condition is met, Do
 - Initialize each Δw_i to zero.
 - For each $\langle \vec{x}, t \rangle$ in *training_examples*, Do
 - Input the instance \vec{x} to the unit and compute the output o
 - For each linear unit weight w_i , Do

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

- For each linear unit weight w_i , Do

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

| x_1 | x_2 | Y |
|-------|-------|-----|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |



STOCHASTIC APPROXIMATION TO GRADIENT DESCENT

56

- Gradient descent is an important general paradigm for learning.
- It is a strategy for searching through a large or infinite hypothesis space that can be applied whenever
 - ? (1) the hypothesis space contains continuously parameterized hypotheses (e.g., the weights in a linear unit), and
 - ? (2) the error can be differentiated with respect to these hypothesis parameters.

STOCHASTIC APPROXIMATION TO GRADIENT DESCENT

57

The key practical difficulties in applying gradient descent are

- ? (1) converging to a local minimum can sometimes be quite slow (i.e., it can require many thousands of gradient descent steps)
- ? (2) if there are multiple local minima in the error surface, then there is no guarantee that the procedure will find the global minimum.

STOCHASTIC APPROXIMATION TO GRADIENT DESCENT

58

- One common variation on gradient descent intended to alleviate these difficulties is called incremental gradient descent, or alternatively stochastic gradient descent.
- 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*.

STOCHASTIC APPROXIMATION TO GRADIENT DESCENT

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

59

- The modified training rule to update the weight is $\vec{E}_d(\vec{w})$
- Stochastic gradient descent consider a distinct error function for each individual training example d as follows

$$E_d(\vec{w}) = \frac{1}{2}(t_d - o_d)^2$$

- where t_d and o_d are the target value and the unit output value for training example d .

The key differences between standard gradient descent and stochastic gradient descent are:

60

- In standard gradient descent, the error is summed over all examples before updating weights, whereas in stochastic gradient descent weights are updated upon examining each training example.
- Summing over multiple examples in standard gradient descent requires more computation per weight update step. On the other hand, because it uses the true gradient, standard gradient descent is often used with a larger step size per weight update than stochastic gradient descent.
- In cases where there are multiple local minima with respect to $E(\vec{w})$, stochastic gradient descent can sometimes avoid falling into these local minima because it uses the various $\nabla E_d(\vec{w})$ rather than $\nabla E(\vec{w})$ to guide its search.

Incremental (Stochastic) Gradient Descent

61

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. $\vec{w} \leftarrow \vec{w} - \eta \nabla E_d[\vec{w}]$

$$E_D[\vec{w}] \equiv \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2$$

$$E_d[\vec{w}] \equiv \frac{1}{2} (t_d - o_d)^2$$

Covergence:

The *Incremental Gradient Descent* can approximate the *Batch Gradient Descent* arbitrarily closely if η is made small enough.

- The training rule in

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

is known as the *delta rule*, or sometimes the LMS (least-mean-square) rule, Adaline rule, or Widrow-Hoff rule (after its inventors).

Outline

63

- Introduction
- Neural Network representation
- Appropriate problems
- Perceptrons
- Backpropagation algorithm

Multi-layer neural networks

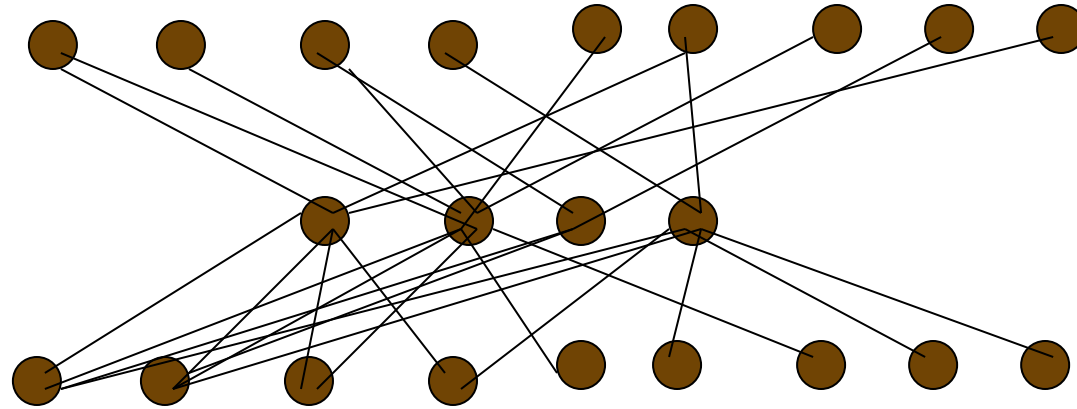
64

- In contrast to perceptrons, multilayer neural networks can not only learn multiple decision boundaries, but the boundaries may be nonlinear.

Output
nodes

Internal
nodes

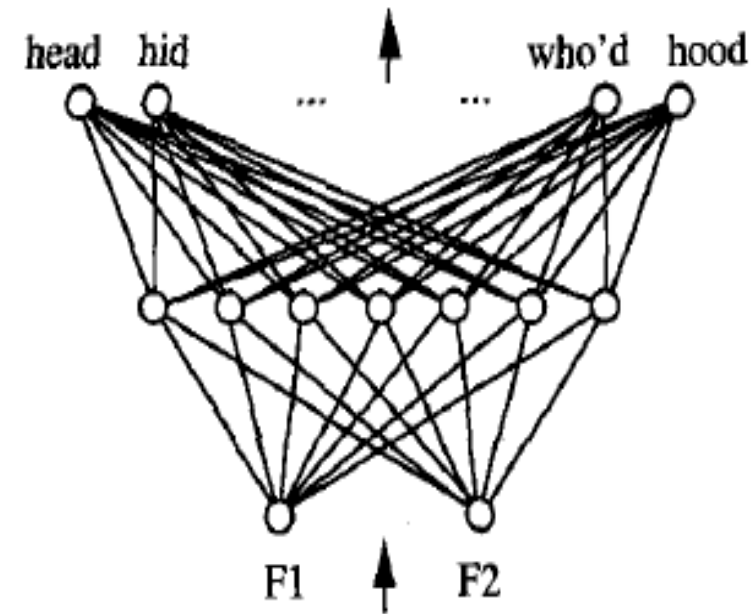
Input
nodes



MULTILAYER NETWORKS

65

- Multilayer feedforward network shown here is trained to recognize 1 of 10 vowel sounds occurring in the context "hd" (e.g., "had," "hid").
- The network input consists of two parameters, F1 and F2, obtained from a spectral analysis of the sound.
- The 10 network outputs correspond to the 10 possible vowel sounds.



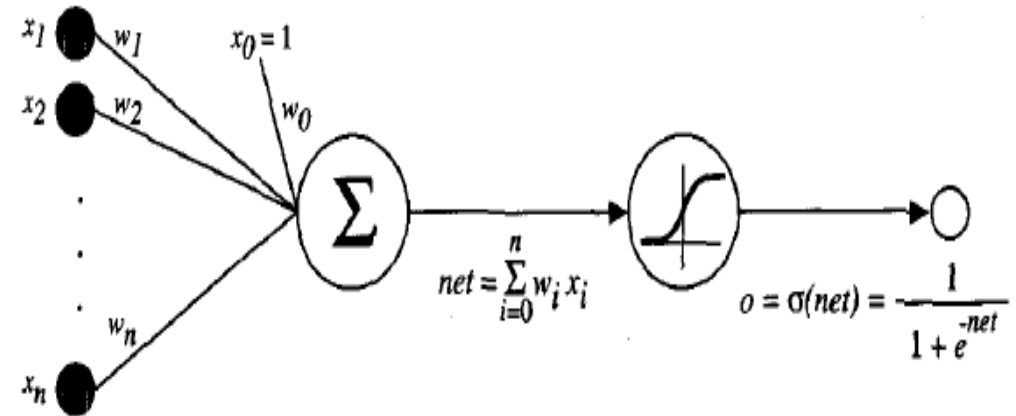
Learning non-linear boundaries

66

- To make nonlinear partitions on the space we need to define each unit as a nonlinear function (unlike the perceptron).
- One solution is to use the sigmoid (logistic) function.

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

where
$$\sigma(y) = \frac{1}{1 + e^{-y}}$$



The BACKPROPAGATION algorithm

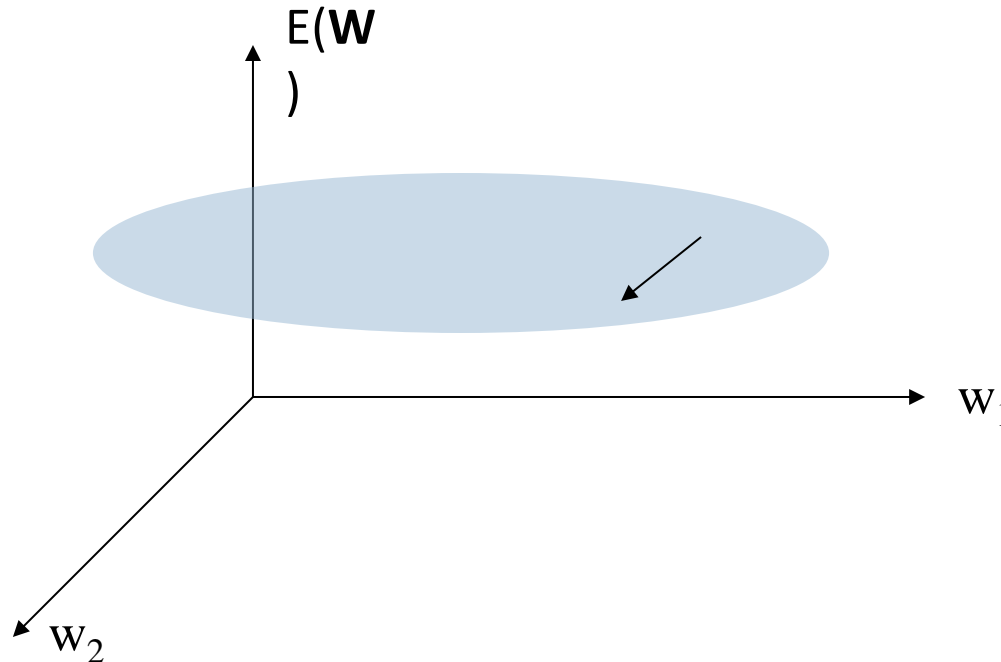
67

- 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.

What is gradient descent algorithm??

68

- Back propagation calculates the gradient of the error of the network regarding the network's modifiable weights.
- This gradient is almost always used in a simple stochastic gradient descent algorithm to find weights that minimize the error.



[3]

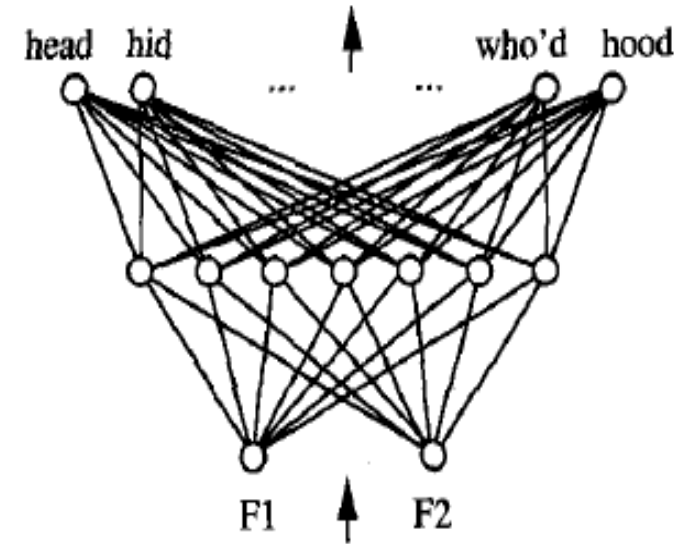
The BACKPROPAGATION algorithm

69

- Here we will consider networks with multiple output units rather than single units, so we need to redefine E to sum the errors over all of the network output units

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

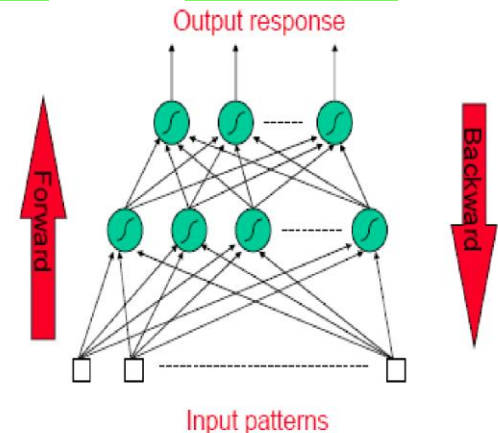
- t_{kd} and O_{kd} are the target and output values associated with the k_{th} output unit and training example d .



Back propagation Algorithm

70

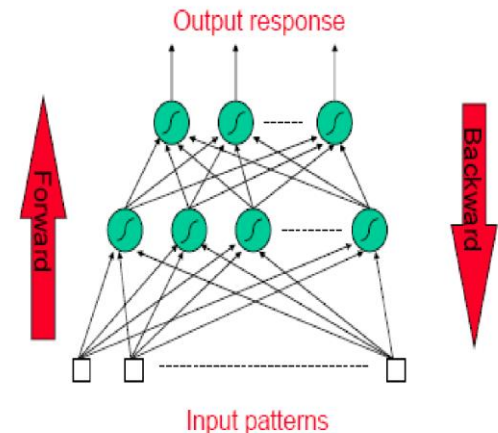
- The back propagation learning algorithm can be divided into two phases:
- Phase 1: Propagation
 - ? Forward propagation of a training pattern's input through the neural network in order to generate the propagation's output activations.
 - ? Backward propagation of the propagation's output activations through the neural network using the training pattern target in order to generate the deltas (the difference between the input and output values) of all output and hidden neurons.



Back propagation Algorithm

71

- Phase 2: Weight update
 - ? Multiply its output delta and input activation to get the gradient of the weight.
 - ? Subtract a ratio (percentage) of the gradient from the weight.



BACKPROPAGATION(*training_examples*, η , n_{in} , n_{out} , n_{hidden})

Each training example is a pair of the form $\langle \vec{x}, \vec{t} \rangle$, where \vec{x} is the vector of network input values, and \vec{t} is the vector of target network output values.

η is the learning rate (e.g., .05). n_{in} 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_{in} inputs, n_{hidden} hidden units, and n_{out} output units.
- Initialize all network weights to small random numbers (e.g., between $-.05$ and $.05$).

| x_1 | x_2 | y |
|-------|-------|-----|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

• 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 δ_k

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

3. For each hidden unit h , calculate its error term δ_h

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

4. Update each network weight w_{ji}

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

where

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

Back propagation Algorithm

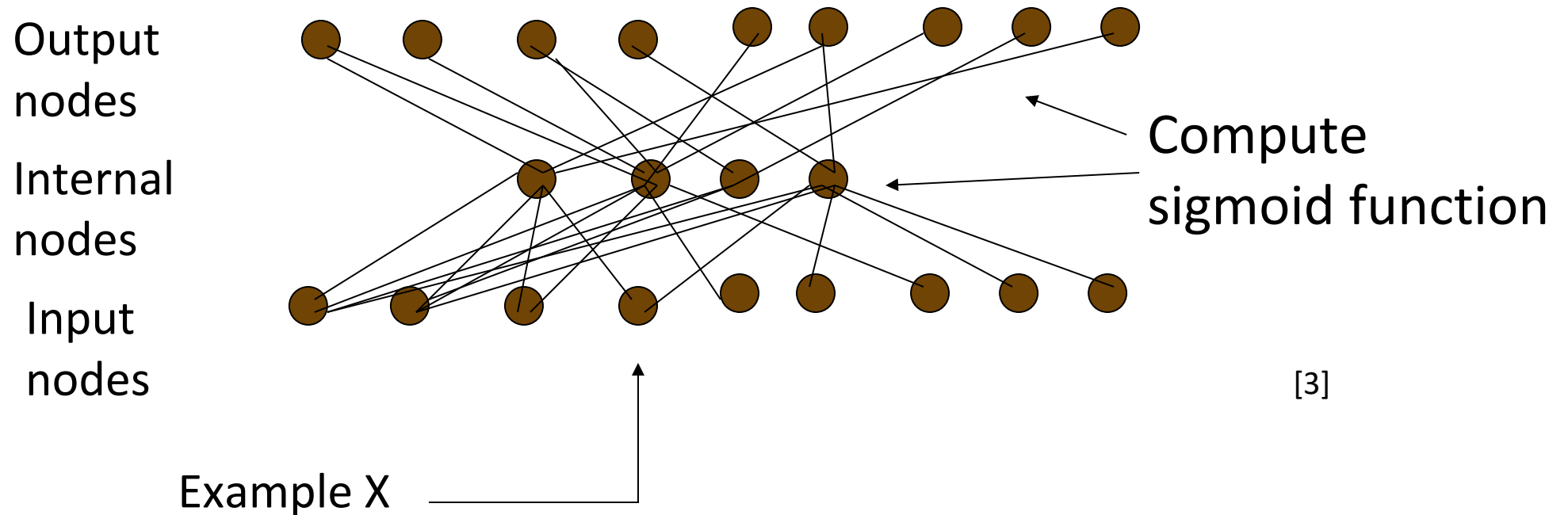
74

```
initialize network weights (often small random values)
do
  forEach training example ex
    prediction = neural-net-output(network, ex) // forward pass
    actual = teacher-output(ex)
    compute error (prediction - actual) at the output units
    compute  $\Delta w_h$  for all weights from hidden layer to output layer // backward pass
    compute  $\Delta w_i$  for all weights from input layer to hidden layer // backward pass continued
    update network weights // input layer not modified by error estimate
  until all examples classified correctly or another stopping criterion satisfied
return the network
```

Propagating forward

75

- Given example X, compute the output of every node until we reach the output nodes:



Number of Hidden Units

76

- The number of hidden units is related to the complexity of the decision boundary.
- If examples are easy to discriminate few nodes would be enough. Conversely complex problems require many internal nodes.
- A rule of thumb is to choose roughly $m / 10$ weights, where m is the number of training examples.

Learning Rates

77

- Different learning rates affect the performance of a neural network significantly.
- Optimal Learning Rate:
 - ? Leads to the error minimum in one learning step.

Limitations of the back propagation algorithm

78

- It is not guaranteed to find global minimum of the error function. It may get trapped in a local minima,
 - ? Improvements,
 - Add momentum.
 - Use stochastic gradient descent.
 - Use different networks with different initial values for the weights.
- Back propagation learning does not require normalization of input vectors; however, normalization could improve performance.
 - ? Standardize all features previous to training.

THANK YOU