Data Mining

Part 5. Prediction

5.5. Prediction by Neural Networks

Spring 2010

Instructor: Dr. Masoud Yaghini

Outline

- How the Brain Works
- Artificial Neural Networks
- Simple Computing Elements
- Feed-Forward Networks
- Perceptrons (Single-layer, Feed-Forward Neural Network)
- Perceptron Learning Method
- Multilayer Feed-Forward Neural Network
- Defining a Network Topology
- Backpropagation Algorithm
- Backpropagation and Interpretability
- Discussion
- References

How the Brain Works

How the Brain Works

- **Neuron** (**nerve cell**)
  - the fundamental functional unit of all nervous system tissue, including the brain.
  - There 10^{11} neurons in the human brain

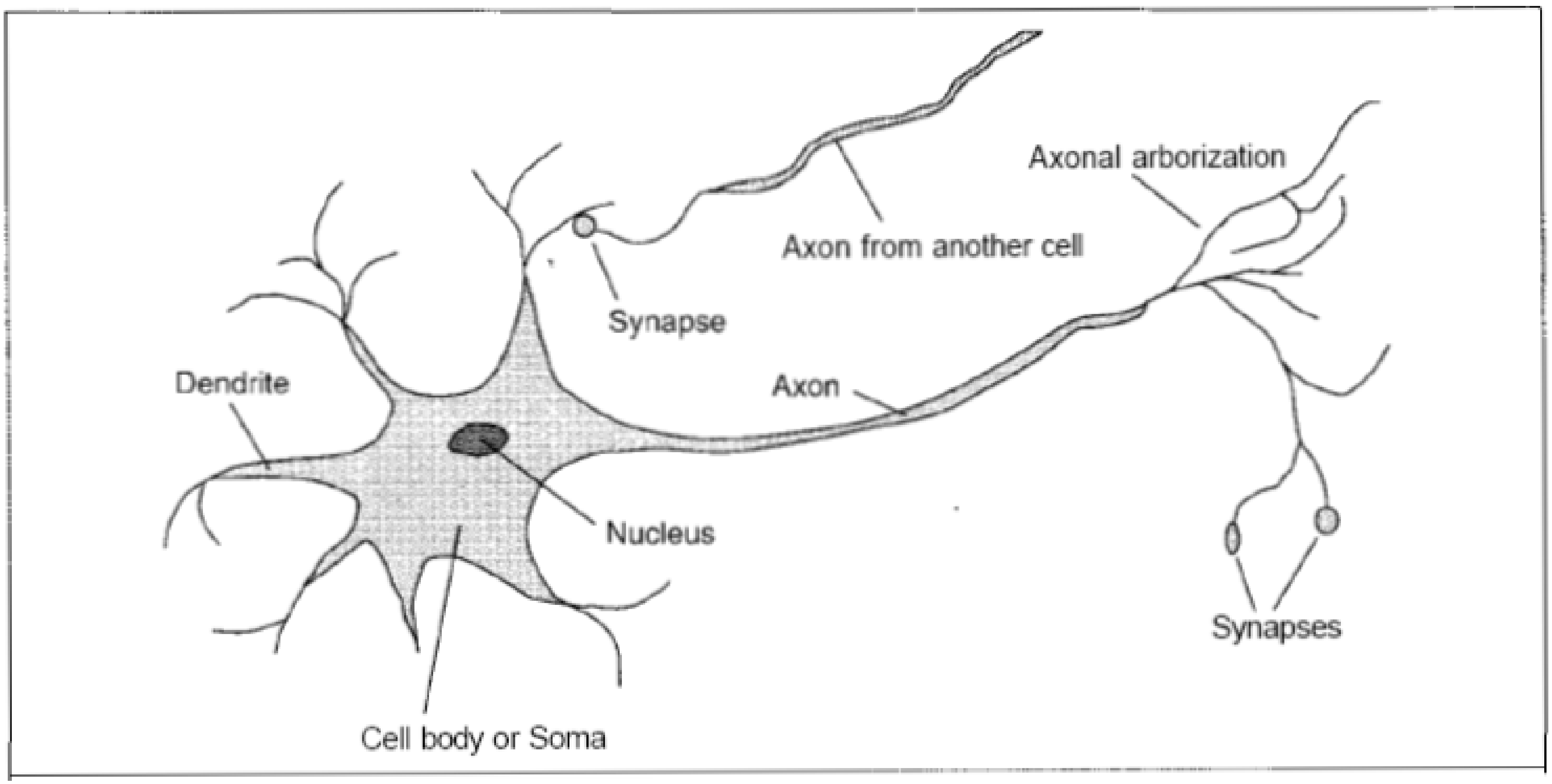
- **Neuron components**
  - **Soma** (**cell body**):
    - provides the support functions and structure of the cell, that contains a **cell nucleus**.
  - **Dendrites**:
    - consist of more branching fibers which **receive signal** from other nerve cells

How the Brain Works

- **Neuron components (cont.)**
  - **Axon**:
    - a branching fiber which carries **signals away from** the neuron that connect to the dendrites and cell bodies of other neurons.
    - In reality, the length of the axon should be about 100 times the diameter of the cell body.
  - **Synapse**:
    - The connecting junction between axon and dendrites.

How the Brain Works

- **The parts of a nerve cell or neuron.**



Neuron Firing Process

- **Neuron Firing Process**
  1. Synapse receives incoming signals, change electrical potential of cell body
  2. When a potential of cell body reaches some limit, neuron “fires”, electrical signal (action potential) sent down axon
  3. Axon propagates signal to other neurons, downstream

How the Brain Works

- **How synapse works:**
  - **Excitatory synapse**: increasing potential
  - **Synaptic connection**: plasticity
  - **Inhibitory synapse**: decreasing potential
- **Migration of neurons**
  - Neurons also form new connections with other neurons
  - Sometimes entire collections of neurons can migrate from one place to another.
  - These mechanisms are thought to form the basis for learning in the brain.
- A collection of simple cells can lead to **thoughts**, **action**, and **consciousness**.

Comparing brains with digital computers

- Advantages of a human brain vs. a computer
  - **Parallelism**: all the neurons and synapses are active simultaneously, whereas most current computers have only one or at most a few CPUs.
  - **More fault-tolerant**: A hardware error that flips a single bit can doom an entire computation, but brain cells die all the time with no ill effect to the overall functioning of the brain.
  - **Inductive algorithm:** To be trained using an inductive learning algorithm

Artificial Neural Networks

Artificial Neural Networks

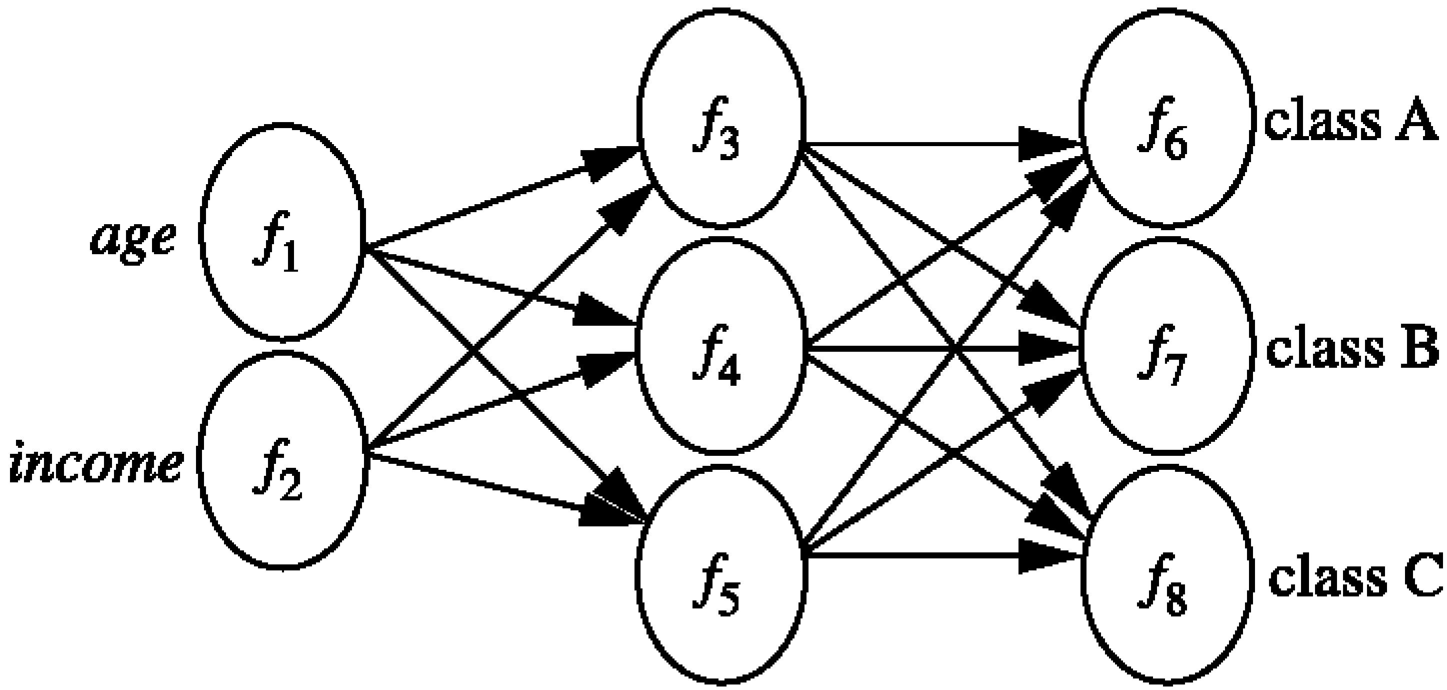
- **Artificial Neural Networks (ANN)** Started by psychologists and neurobiologists to develop and test computational analogues of neurons
- Other names:
  - connectionist learning,
  - parallel distributed processing,
  - neural computation,
  - adaptive networks, and
  - collective computation

Artificial Neural Networks

- **Artificial neural networks components:**
  - **Units**
    - A neural network is composed of a number of nodes, or units
    - Metaphor for nerve cell body
  - **Links**
    - Units connected by links.
    - Links represent synaptic connections from one unit to another
  - **Weight**
    - Each link has a numeric weight

Artificial Neural Networks

- An example of ANN



Artificial Neural Networks

- **Long-term memory**
  - Weights are the primary means of **long-term storage** in neural networks
- **Learning method**
  - Learning usually takes place by adjusting the weights.
- **Input and Output Units**
  - Some of the units are connected to the external environment, and can be designated as **input units** or **output units**

Artificial Neural Networks

- **Components of a Unit**
  - a set of **input links** from other units,
  - a set of **output links** to other units,
  - a current **activation level**, and
  - a means of computing the activation level at the next step in time, given its inputs and weights.
- The idea is that each unit does a local computation based on inputs from its neighbors, but without the need for any global control over the set of units as a whole.

Artificial Neural Networks

- Real (Biological) Neural Network vs. Artificial Neural Network

Real Neural Network	Artificial Neural Network
Soma / Cell body ⟺	Neuron / Node / Unit
Dendrite ⟺	Input links
Axon ⟺	Output links
Synapse ⟺	Weight

Artificial Neural Networks

- Neural networks can be used for both
  - **supervised learning**, and
  - **unsupervised learning**
- For supervised learning neural networks can be used for both
  - classification (to predict the class label of a given example) and
  - prediction (to predict a continuous-valued output).
- In this chapter we want to discuss about application of neural networks for **supervised learning**

Artificial Neural Networks

- To build a neural network must decide:
  - how many units are to be used
  - what kind of units are appropriate
  - how the units are to be connected to form a network.
- Then
  - initializes the weights of the network, and
  - trains the weights using a **learning algorithm** applied to a set of training examples for the task.
- The use of examples also implies that one must decide how to encode the examples in terms of inputs and outputs of the network.

Simple Computing Elements

Simple computing elements

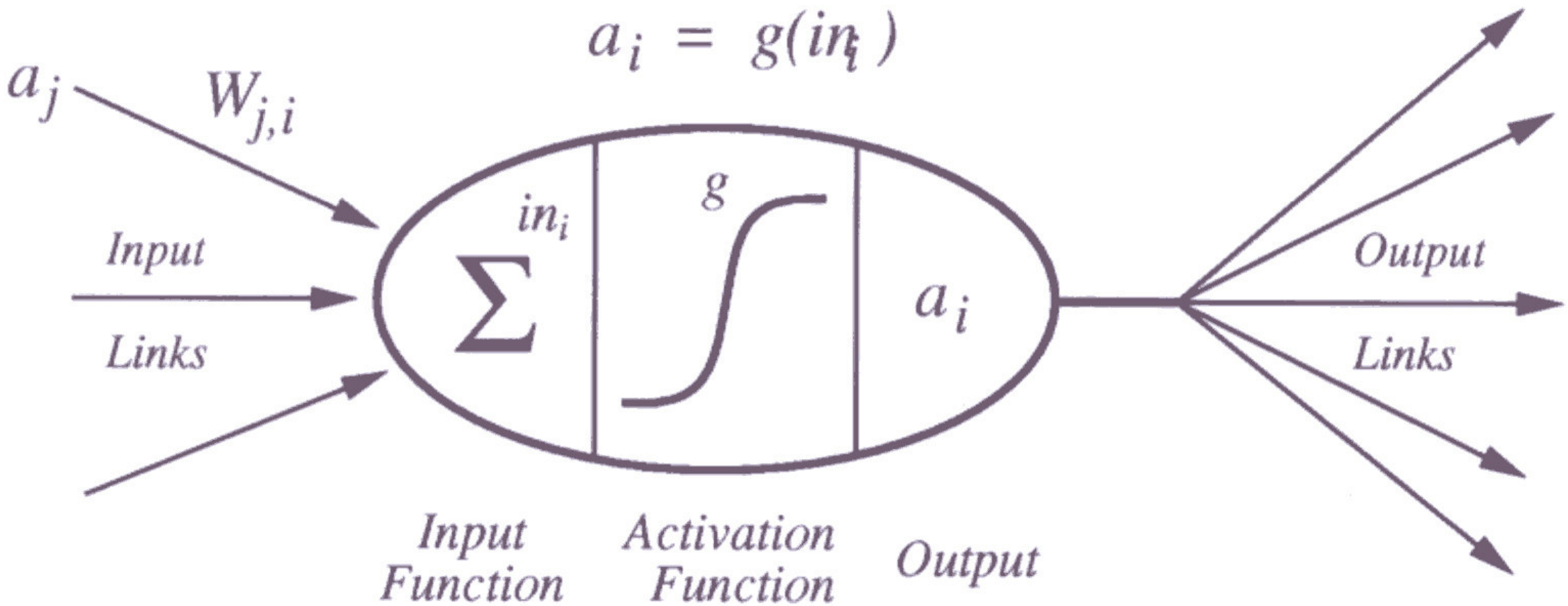
- Each unit performs a simple process:
  - Receives n-inputs
  - Multiplies each input by its weight
  - Applies activation function to the sum of results
  - Outputs result

Simple computing elements

- Two computational components
  - **Linear component**:
    - **input function**, that in_i, that computes the weighted sum of the unit's input values.
  - **Nonlinear component**:
    - **activation function**, g, that transforms the weighted sum into the final value that serves as the **unit's activation value ,** a_i
    - Usually, all units in a network use the same activation function.

Simple computing elements

- A typical unit



Simple computing elements

- Total weighted input

$$in_i = \sum_j W_{j,i} a_j$$

  - the weights on links from node *j* into node *i* are denoted by $W_{j,i}$
  - The input values is called a_j

Example: Total weighted input

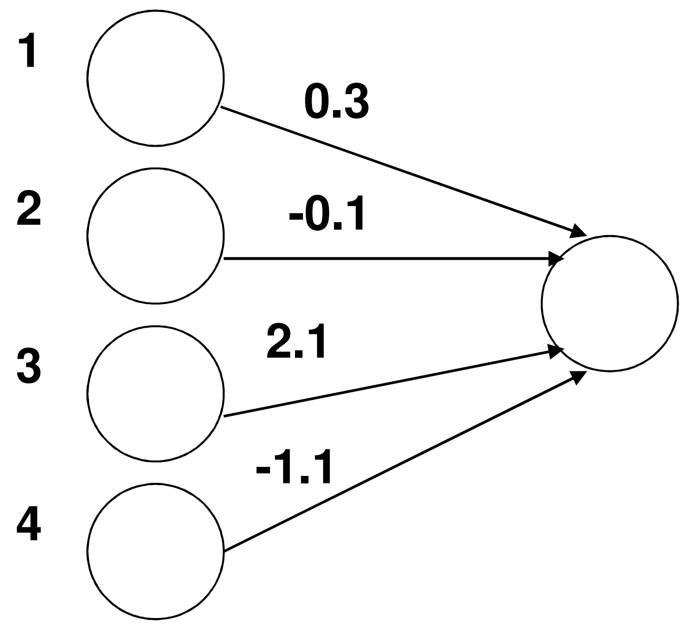
Input: (3, 1, 0, -2)

Processing:

3(0.3) + 1(-0.1) + 0(2.1) + -1.1(-2)

= 0.9 + (-0.1) + 2.2

= 3



Simple computing elements

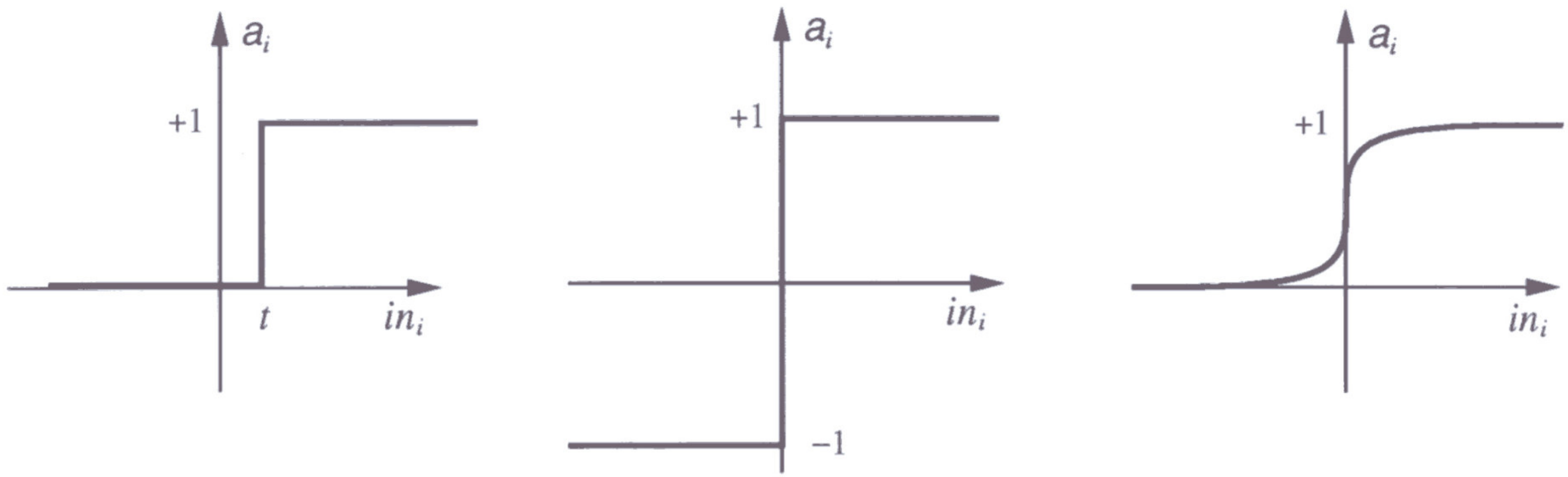
- The **activation function** g

$$a_i = g(in_i) = g(\sum_j W_{j,i} a_j)$$

- Three common mathematical functions for g are
  - **Step function**
  - **Sign function**
  - **Sigmoid function**

Simple computing elements

- Three common mathematical functions for *g*



(a) Step function (b) Sign function (c) Sigmoid function

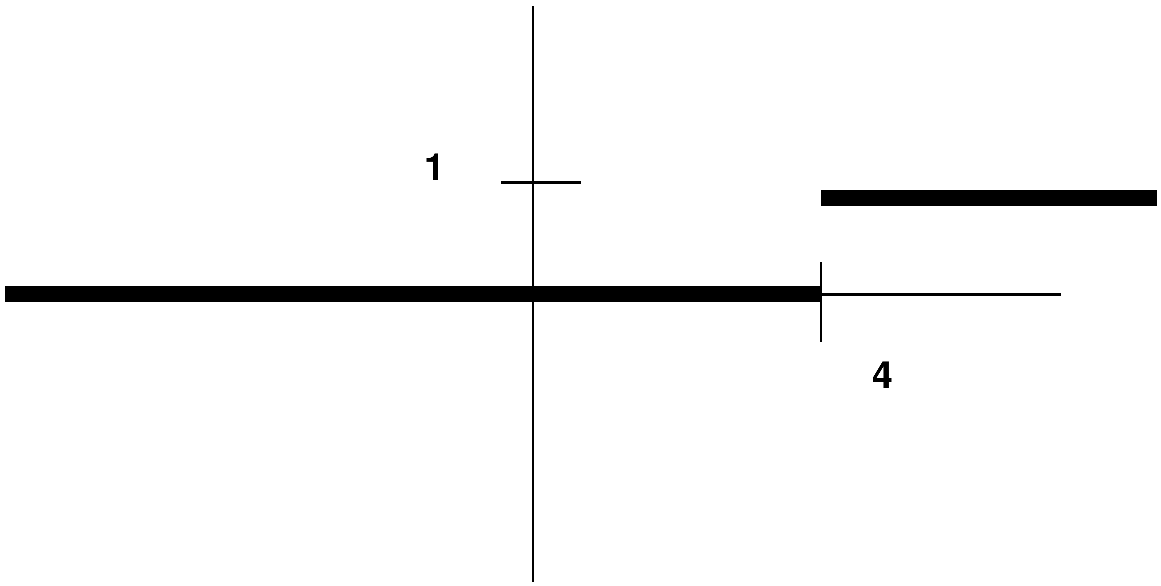
$$\text{step}_t(x) = \begin{cases} 1, & \text{if } x \geq t \\ 0, & \text{if } x < t \end{cases} \quad \text{sign}(x) = \begin{cases} +1, & \text{if } x \geq 0 \\ -1, & \text{if } x < 0 \end{cases} \quad \text{sigmoid}(x) = \frac{1}{1 + e^{-x}}$$

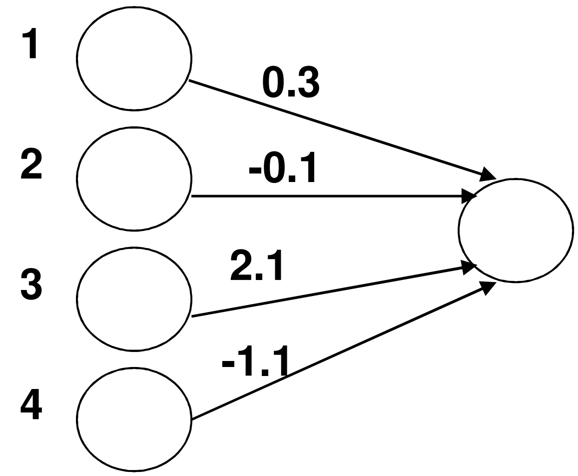
Step Function

- The **step function** has a threshold ***t*** such that it outputs a 1 when the input is greater than its threshold, and outputs a 0 otherwise.
- The biological motivation is that a 1 represents the firing of a pulse down the axon, and a 0 represents no firing.
- The threshold represents the minimum total weighted input necessary to cause the neuron to fire.

Step Function Example

- Let t = 4





$$Step_4(3) = 0$$

Step Function

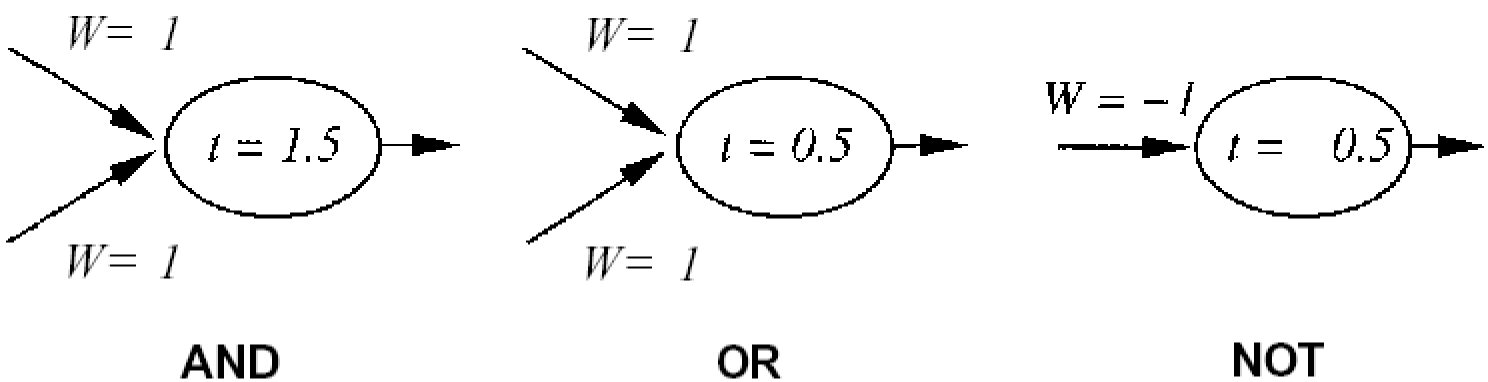
- It mathematically convenient to replace the threshold with an extra input weight.
- Because it need only worry about adjusting weights, rather than adjusting both **weights** and **thresholds**.
- Thus, instead of having a threshold *t* for each unit, we add an extra input whose activation a_0

$$a_i = step_t\left(\sum_{j=1}^{n} W_{j,i} a_j\right) = step_0\left(\sum_{j=0}^{n} W_{j,i} a_j\right)$$

Where $W_{0,\,i} = t$ and $a_0 = -1$ ← fixed

Step Function

- The Figure shows how the Boolean functions *AND, OR*, and *NOT* can be represented by units with a step function and suitable weights and thresholds.
- This is important because it means we can use these units to build a network to compute any Boolean function of the inputs.



Sigmoid Function

- A sigmoid function often used to approximate the step function

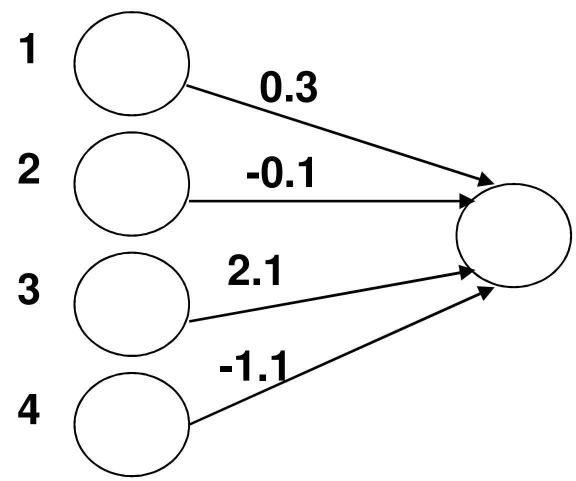
$$f(x) = \frac{1}{1 + e^{-\sigma x}}$$

σ : the steepness parameter

Sigmoid Function

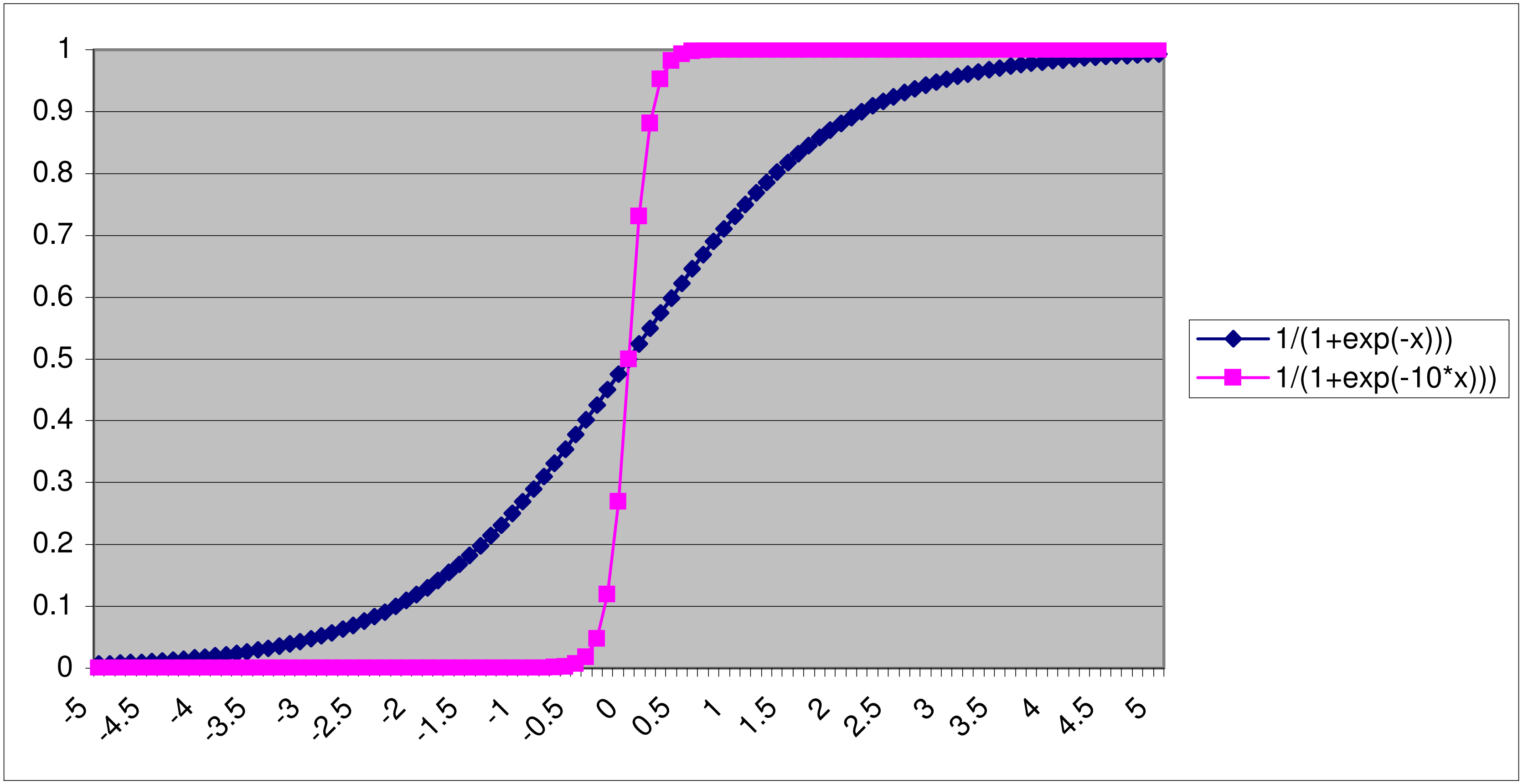
- *Input*: (3, 1, 0, -2), σ=1

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



$$f(3) = \frac{1}{1+e^{-x}} \approx 0.95$$

Sigmoid Function

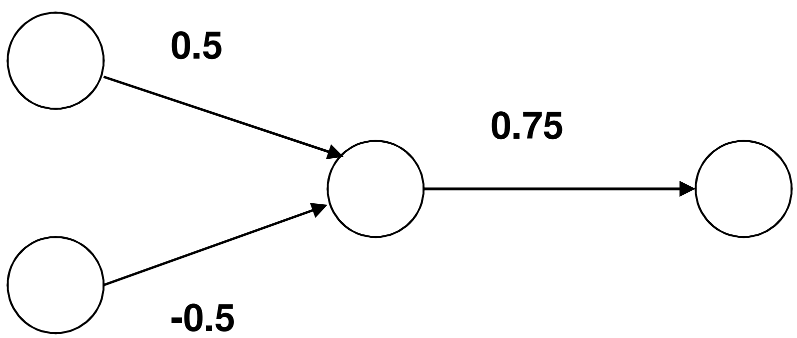


sigmoidal(0) = 0.5

Another Example

- A two weight layer, feedforward network
- Two inputs, one output, one ‘hidden’ unit
- *Input*: (3, 1)

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



- What is the output?

Computing in Multilayer Networks

- Computing:
  - Start at leftmost layer
  - Compute activations based on inputs
  - Then work from left to right, using computed activations as inputs to next layer
- Example solution

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

  - Activation of hidden unit
    - f(0.5(3) + -0.5(1)) = f(1.5 – 0.5) = f(1) = 0.731
  - Output activation
    - f(0.731(0.75)) = f(0.548) = 0.634

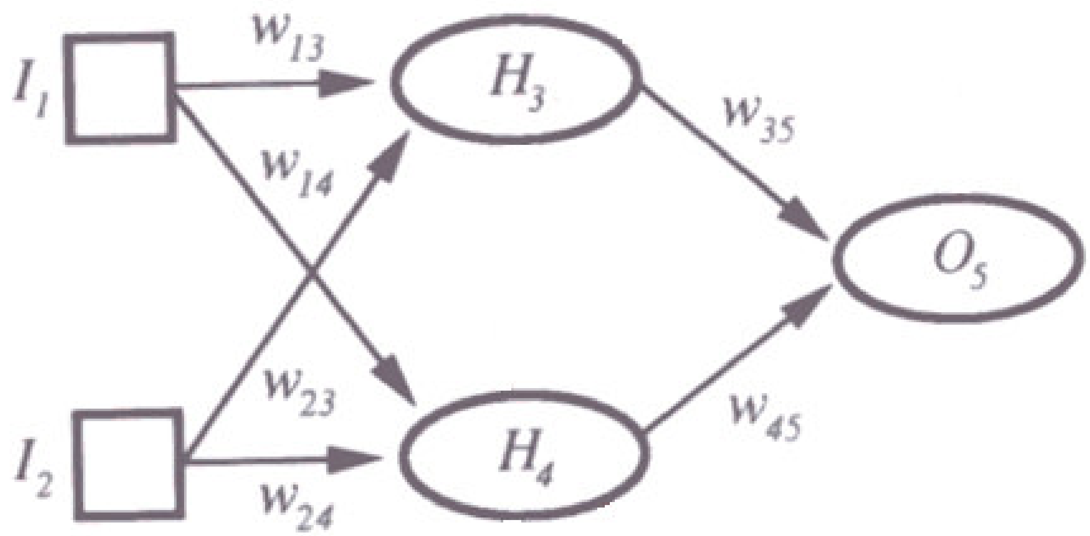
Feed-Forward Networks

Feed-forward Networks

- **Feed-forward networks**
  - Unidirectional links
  - Directed acyclic (no cycles) graph (DAG)
  - No links between units in the same layer
  - No links backward to a previous layer
  - No links that skip a layer.
  - Uniformly processing from input units to output units

Feed-forward Networks

- An example: A **two-layer**, **feed-forward network** with **two inputs**, **two hidden nodes**, and **one output node**.



Feed-forward Networks

- **Units**
  - **Input units**: the activation value of each of these units is determined by the environment.
  - **Output units**: at the right-hand end of the network units
  - **Hidden units**: they have no direct connection to the outside world.

- Because the input units (square nodes) simply serve to pass activation to the next layer, they are not counted

Feed-forward Networks

- Types of feed-forward networks:
  - **Perceptrons**
    - No hidden units
    - This makes the learning problem much simpler, but it means that perceptrons are very limited in what they can represent.
  - **Multilayer networks**
    - one or more hidden units

Feed-forward Networks

- Feed-forward networks have a fixed structure and fixed activation functions g
- The functions have a specific parameterized structure
- The weights chosen for the network determine which of these functions is actually represented.
- For example, the network calculates the following function:

$$
\begin{aligned}
a_5 &= g(W_{3,5}a_3 + W_{4,5}a_4) \\
&= g(W_{3,5}g(W_{1,3}a_1 + W_{2,3}a_2) + W_{4,5}g(W_{1,4}a_1 + W_{2,4}a_2))
\end{aligned}
$$

  – where g is the activation function, a_i and , is the output of node i.

What neural networks do

- Because the activation functions g are nonlinear, the whole network represents a complex nonlinear function.
- If you think of the weights as parameters or coefficients of this function, then learning just becomes:
  - a process of tuning the parameters to fit the data in the training set—a process that statisticians call **nonlinear regression**.

Optimal Network Structure

- Too small network
  - incapable of representation
- Too big network
  - not generalized well
  - Overfitting when there are too many parameters.

Perceptrons

(Single-layer, Feed-forward Neural Networks)

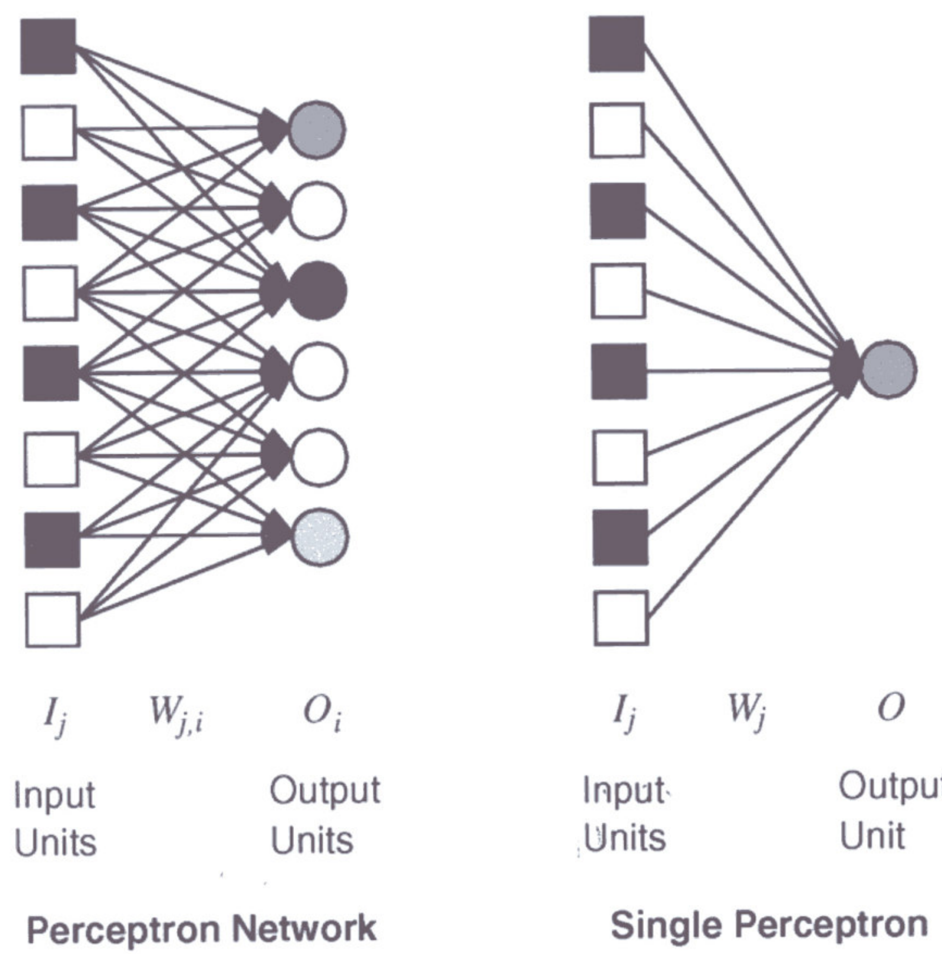
Perceptrons

- **Perceptrons**
  - **Single-layer feed-forward network**
  - were first studied in the late 1950s

- **Types of Perceptrons:**
  - **Single-output Perceptron**
    - perceptrons with a single output unit
  - **Multi-output perceptron**
    - perceptrons with several output units

Perceptrons

- Each output unit is independent of the others
- Each weight only affects one of the outputs.



Perceptrons

- Activation of output unit:

$$O = Step_0\left(\sum_j W_j I_j\right) = Step_0(\mathbf{W}.\mathbf{I})$$

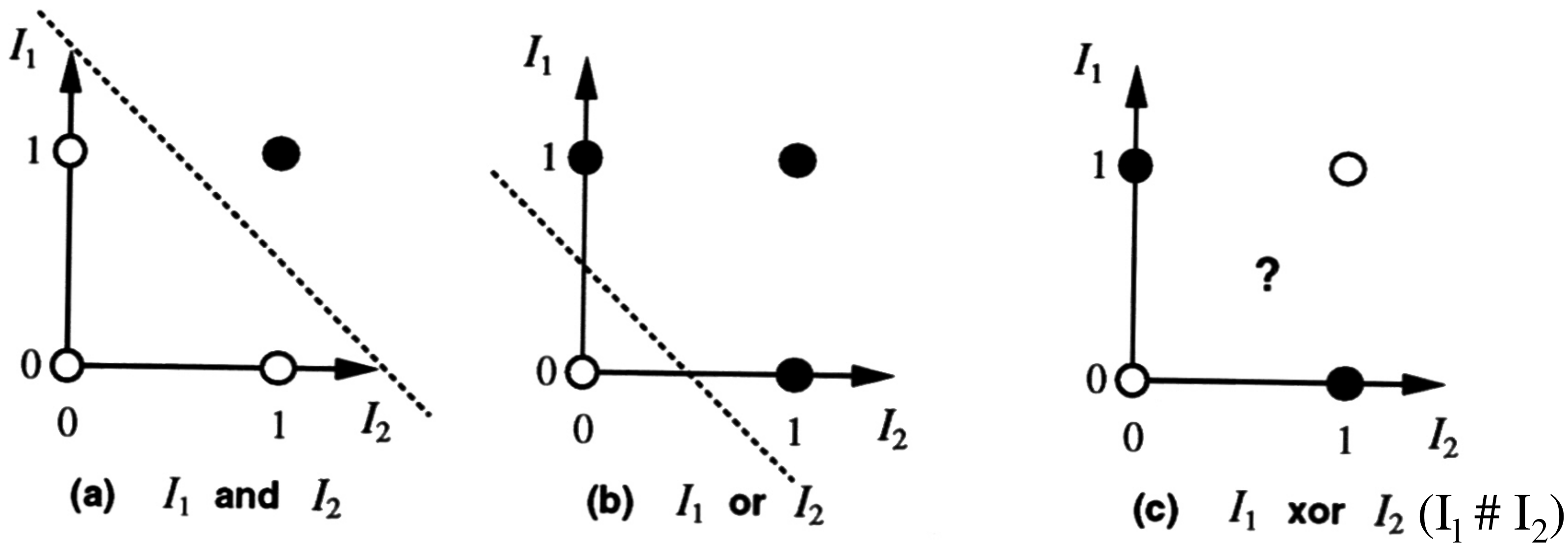
  - W_j : The weight from input unit j
  - I_j : The activation of input unit j
  - we have assumed an additional weight W_0 to provide a threshold for the step function, with $I_0 = -1$.

Perceptrons

- Perceptrons are severely limited in the Boolean functions they can represent.
- The problem is that any input I_j can only influence the final output in one direction, no matter what the other input values are.
- Consider some input vector a.
  - Suppose that this vector has $a_j = 0$ and that the vector produces a 0 as output. Furthermore, suppose that when a_j is replaced with 1, the output changes to 1. This implies that W_j must be positive.
  - It also implies that there can be no input vector b for which the output is 1 when $b_j = 0$, but the output is 0 when b_j is replaced with 1.

Perceptrons

- The Figure shows three different Boolean functions of two inputs, the AND, OR, and XOR functions.



- Black dots indicate a point in the input space where the value of the function is 1, and white dots indicate a point where the value is 0.

Perceptrons

- As we will explain, a perceptron can represent a function only if there is some line that separates all the white dots from the black dots.
- Such functions are called **linearly separable.**
- Thus, a perceptron can represent AND and OR, but not XOR (if I_1 # I_2)**.**

Perceptrons

- The fact that a perceptron can only represent linearly separable functions follows directly from Equation:

$$O = Step_0\left(\sum_j W_j I_j\right) = Step_0(\mathbf{W}.\mathbf{I})$$

- A perceptron outputs a 1 only if W . I > 0.
  - This means that the entire input space is divided in two along a boundary defined by W . I = 0,
  - that is, a plane in the input space with coefficients given by the weights.

Perceptrons

- It is easiest to understand for the case where *n = 2. In Figure (a),* one possible separating "plane" is the dotted line defined by the equation

$$I_1 = -I_2 + 1.5 \qquad \text{or} \qquad I_1 + I_2 = 1.5$$

- The region above the line, where the output is 1 , is therefore given by

$$-1.5 + I_1 + I_2 > 0$$

Perceptron Learning Method

Perceptron Learning Method

- The initial network has randomly assigned weights, usually from the range [-0.5,0.5].
- The network is then updated to try to make it consistent with the training examples (instances).
- This is done by making small adjustments in the weights to reduce the difference between the observed and predicted values.
- The algorithm is the need to repeat the update phase several times for each example in order to achieve convergence.

Perceptron Learning Method

- **Epochs**
  - The updating process is divided into **epochs.**
  - Each epoch involves updating all the weights for all the examples.

Perceptron Learning Method

- The generic neural network learning method

```
function NEURAL-NETWORK-LEARNING(examples) returns network

   network ← a network with randomly assigned weights
   repeat
      for each e in examples do
         O ← NEURAL-NETWORK-OUTPUT(network, e)
         T ← the observed output values from e
         update the weights in network based on e, O, and T
      end
   until all examples correctly predicted or stopping criterion is reached
   return network
```

an epoch

Err = T-O

The generic neural network learning method: adjust the weights until predicted output values **O** and true values **T** agree.

Perceptron Learning Method

- The weight update rule
  - If the predicted output for the single output unit is O, and the correct output should be T, then the error is given by

Err = T – O

  - If the ***Err*** is positive, we need to increase O
  - If the ***Err*** is negative, we need to decrease O
  - Each input unit contributes $W_j I_j$ to the total input, so
  - If I_j is positive, an increase in W_j will tend to increase O
  - If I_j is negative, an increase in W_j will tend to decrease O.

Perceptron Learning Method

- We can achieve the effect we want with the following rule:

$$W_j \leftarrow W_j + \alpha * I_j * Err$$

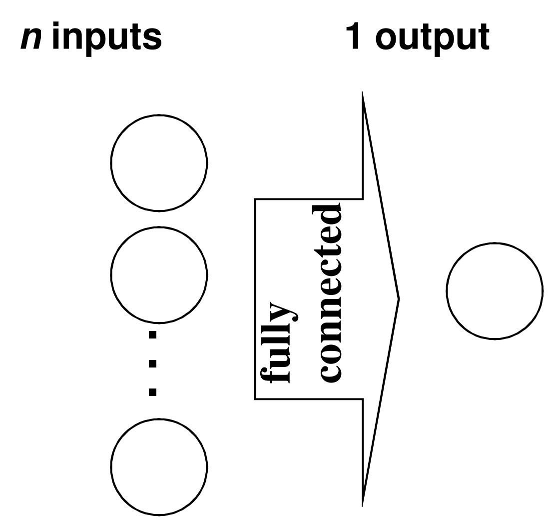
  - **α** : is the **learning rate**

- This rule is a variant of the **perceptron learning rule** proposed by **Frank Rosenblatt**.
  - **Rosenblatt** proved that a learning system using the perceptron learning rule will converge to a set of weights that correctly represents the examples, as long as the examples represent a linearly separable function.

Delta Rule for a Single Output Unit

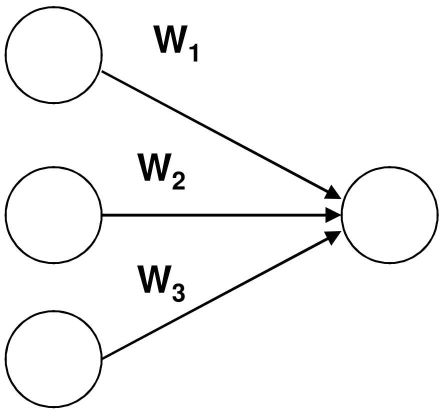
$$\Delta W_j = \alpha (T - O) I_j$$

ΔW_j	Change in *j* th weight of weight vector
α	Learning rate
T	Target or correct output
O	Net (summed, weighted) input to output unit
I_j	j th input value



Example

- W = (W1, W2, W3)
  - Initially: W= (.5 .2 .4)
- Let $\alpha = 0.5$
- Apply delta rule



Sample	Input	Output
1	0 0 0	0
2	1 1 1	1
3	1 0 0	1
4	0 0 1	1

One Epoch of Training

Step	Input	Desired output (T)	Actual output (O)	Starting Weights	Weight updates
1	(0 0 0)	0	0	(.5 .2 .4)	
2	(1 1 1)	1			
3	(1 0 0)	1			
4	(0 0 1)	1			

Delta rule: $\Delta W_j = \alpha (T - O) I_j$

One Epoch of Training

Step	Input	Desired output (T)	Actual output (O)	Starting Weights	Weight updates
1	(0 0 0)	0	0	(.5 .2 .4)	W1: 0.1(0 – 0)0 W2: 0.1(0 – 0)0 W3: 0.1(0 – 0)0

Delta rule: $\Delta W_j = \alpha(T - O)I_j$

delta-rule1.xls

One Epoch of Training

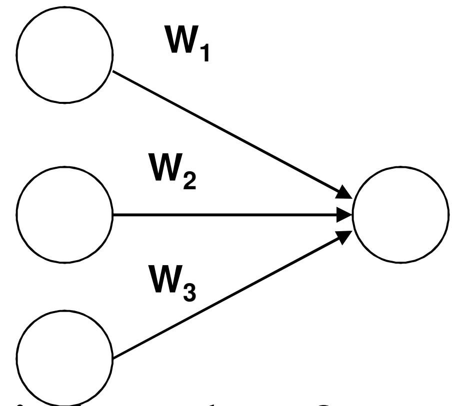
Step	Input	Desired output (T)	Actual output (O)	Starting Weights	Weight updates
1	(0 0 0)	0	0	(.5 .2 .4)	(0 0 0)
2	(1 1 1)	1		(.5 .2 .4)	
3	(1 0 0)	1			
4	(0 0 1)	1			

Remaining Steps in First Epoch of Training

Step	Input	Desired output (T)	Actual output (O)	Starting Weights	Weight updates
1	(0 0 0)	0	0	(.5 .2 .4)	(0 0 0)
2	(1 1 1)	1	1.1	(.5 .2 .4)	(-.05 -.05 -.05)
3	(1 0 0)	1	.45	(.45 .15 .35)	(.275 0 0)
4	(0 0 1)	1	.35	(.725 .15 .35)	(0 0 .325)

Completing the Example

- After 18 epochs
  - Weights
    - W1= 0.990735
    - W2= -0.970018005
    - W3= 0.98147



- Does this adequately approximate the training data?

Sample	Input	Output
1	0 0 0	0
2	1 1 1	1
3	1 0 0	1
4	0 0 1	1

Example

- Actual Outputs

Sample	Input	Desired Output	Actual Output
1	0 0 0	0	0
2	1 1 1	1	1.002187
3	1 0 0	1	0.990735
4	0 0 1	1	0.98147

examples in ANN

- There is a slight difference between the example descriptions used for neural networks and those used for other attribute-based methods such as decision trees.
- In a neural network, all inputs are **real numbe**rs in some fixed range, whereas decision trees allow for multivalued attributes with a discrete set of values.
- For example, an attribute may has values *None*, *Some*, and *Full*.

Perceptron Learning Method

- **There are two ways to handle this.**
  - **Local encoding**
    - we use a single input unit and pick an appropriate number of distinct values to correspond to the discrete attribute values.
    - For example, we can use *None = 0.0, Some = 0.5, and Full = 1.0.*
  - **Distributed encoding**
    - we use one input unit for each value of the attribute, turning on the unit that corresponds to the correct value.

Multilayer Feed-Forward Neural Network

Multilayer Feed-Forward Neural Network

- A multilayer feed-forward neural network consists of several layers includes:
  - an **input layer**,
  - one or more **hidden layers**, and
  - an **output layer**.

Multilayer Feed-Forward Neural Network

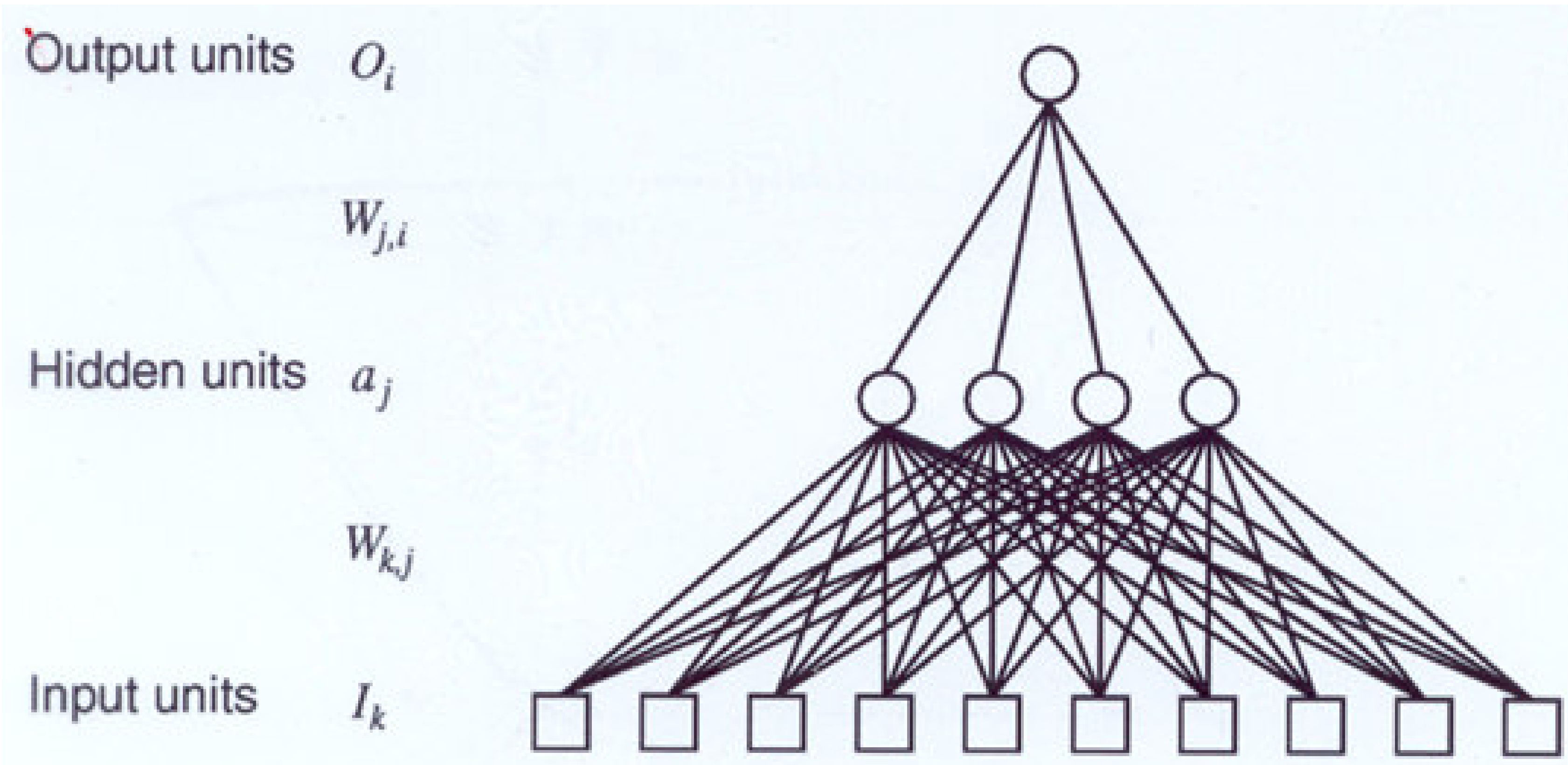
- Each layer is made up of units.
- A two-layer neural network has a hidden layer and an output layer.
- The input layer is not counted because it serves only to pass the input values to the next layer.
- A network containing two hidden layers is called a **three-layer neural network**, and so on.

Multilayer Feed-Forward Neural Network

- Suppose we want to construct a network for a problem.
- We have **ten attributes** describing each example, so we will need ten input units.
- How many hidden units are needed?
  - The problem of choosing the right number of hidden units in advance is still not well-understood.
- We use a network with four hidden units.

Multilayer Feed-Forward Neural Network

- A two-layer feed-forward network



Multilayer Feed-Forward Neural Network

- The **inputs** to the network correspond to the attributes measured for each training example.
- Inputs are fed simultaneously into the units making up the **input layer**
- They are then weighted and fed simultaneously to a **hidden layer**
- The number of hidden layers is arbitrary, although in practice, usually only one is used.
- The weighted outputs of the last hidden layer are input to units making up the **output layer**, which sends out the network's prediction.

Multilayer Feed-Forward Neural Network

- The network is **feed-forward** in that none of the weights cycles back to an input unit or to an output unit of a previous layer
- From a statistical point of view, networks perform **nonlinear regression**
- Given enough hidden units and enough training samples, they can closely approximate any function

Multilayer Feed-Forward Neural Network

- **Learning method**
  - example inputs are presented to the network and the network computes an output vector that matches the target.
  - If there is an error (a difference between the output and target), then the weights are adjusted to reduce this error.
  - The trick is to assess the blame for an error and divide it among the contributing weights.
  - In perceptrons, this is easy, because there is only one weight between each input and the output.
  - But in multilayer networks, there are many weights connecting each input to an output, and each of these weights contributes to more than one output.

Defining a Network Topology

Defining a Network Topology

- First decide the **network topology:**
  - the number of units in the **input layer**
  - the number of **hidden layers** (if > 1),
  - the number of units in each hidden layer
  - the number of units in the **output layer**

- Normalizing the input values for each attribute measured in the training examples to [0.0—1.0] will help speed up the learning phase.

Defining a Network Topology

- **Input units**
  - Normalizing the input values for each attribute measured in the training examples to [0.0—1.0] will help speed up the learning phase.
  - Discrete-valued attributes may be encoded such that there is one input unit per domain value.
    - Example, if an attribute A has three possible or known values, namely $\{a_0, a_1, a_2\}$, then we may assign three input units to represent A. That is, we may have, say, I_0, I_1, I_2 as input units.
    - Each unit is initialized to 0.
    - Then
      - I_0 *is set* to 1, If $A = a_1$
      - I_1 *is set to 1*, If $A = a_2$
      - I_2 *is set to 1*, If $A = a_3$

Defining a Network Topology

- **Output unit**
  - For classification, one output unit may be used to represent two classes (where the value 1 represents one class, and the value 0 represents the other).
  - If there are more than two classes, then one output unit per class is used.

Defining a Network Topology

- **Hidden layer units**
  - There are no clear rules as to the "best" number of hidden layer units
  - Network design is a trial-and-error process and may affect the accuracy of the resulting trained network.

- Once a network has been trained and its accuracy is **unacceptable**, repeat the training process with a **different network topology** or a **different set of initial weights**

Optimal Network Structure

- Using **genetic algorithm**: for finding a good network structure
- Hill-climbing search (modifying an existing network structure)
  - Start with a big network: **optimal brain damage algorithm**
- Removing weights from fully connected model
  - Start with a small network: **tiling algorithm**
- Start with single unit and add subsequent units
- **Cross-validation techniques**: are useful for deciding when we have found a network of the right size.

Backpropagation Algorithm

Backpropagation

- The **backpropagation algorithm** performs learning on a **multilayer feed-forward neural network**.
- It is the most popular method for learning in multilayer networks
- **Backpropagation** iteratively process a set of training examples & compare the network's prediction with the actual known target value
- The target value may be the known class label of the training example (for classification problems) or a continuous value (for prediction problems).

Backpropagation

- For each training example, the weights are modified to **minimize the mean squared error** between the network's prediction and the actual target value

- Modifications are made in the "**backwards**" direction
  - from the output layer, through each hidden layer down to the first hidden layer, hence "**backpropagation**"
  - Although it is not guaranteed, in general the weights will eventually converge, and the learning process stops.

Backpropagation

- **Backpropagation algorithm** steps:
  - Initialize the weights
    - Initialize weights to small random and biases in the network
  - Propagate the inputs forward
    - by applying activation function
  - Backpropagate the error
    - by updating weights and biases
  - Terminating condition
    - when error is very small, etc.

Backpropagation Algorithm

- Input:
  - ***D***, a data set consisting of the training examples and their associated target values
  - ***l***, the learning rate
  - ***network***, a multilayer feed-forward network
- Output:
  - A trained neural network.

Initialize the weights

- **1) Initialize the weights**
  - The weights in the network are initialized to small random numbers
  - e.g., ranging from -1.0 to 1.0 or -0.5 to 0.5
  - Each unit has a bias associated with it
  - The biases are similarly initialized to small random numbers.

- **Each training example is processed by the steps 2 to 8.**

Propagate the inputs forward

- **2) determining the output of input layer units**
  - the training example is fed to the input layer of the network.
  - The inputs pass through the input units, unchanged.
  - For an input unit, *j*,
    - its input value, I_j
    - its output, O_j, is equal to its input value, I_j.

Propagate the inputs forward

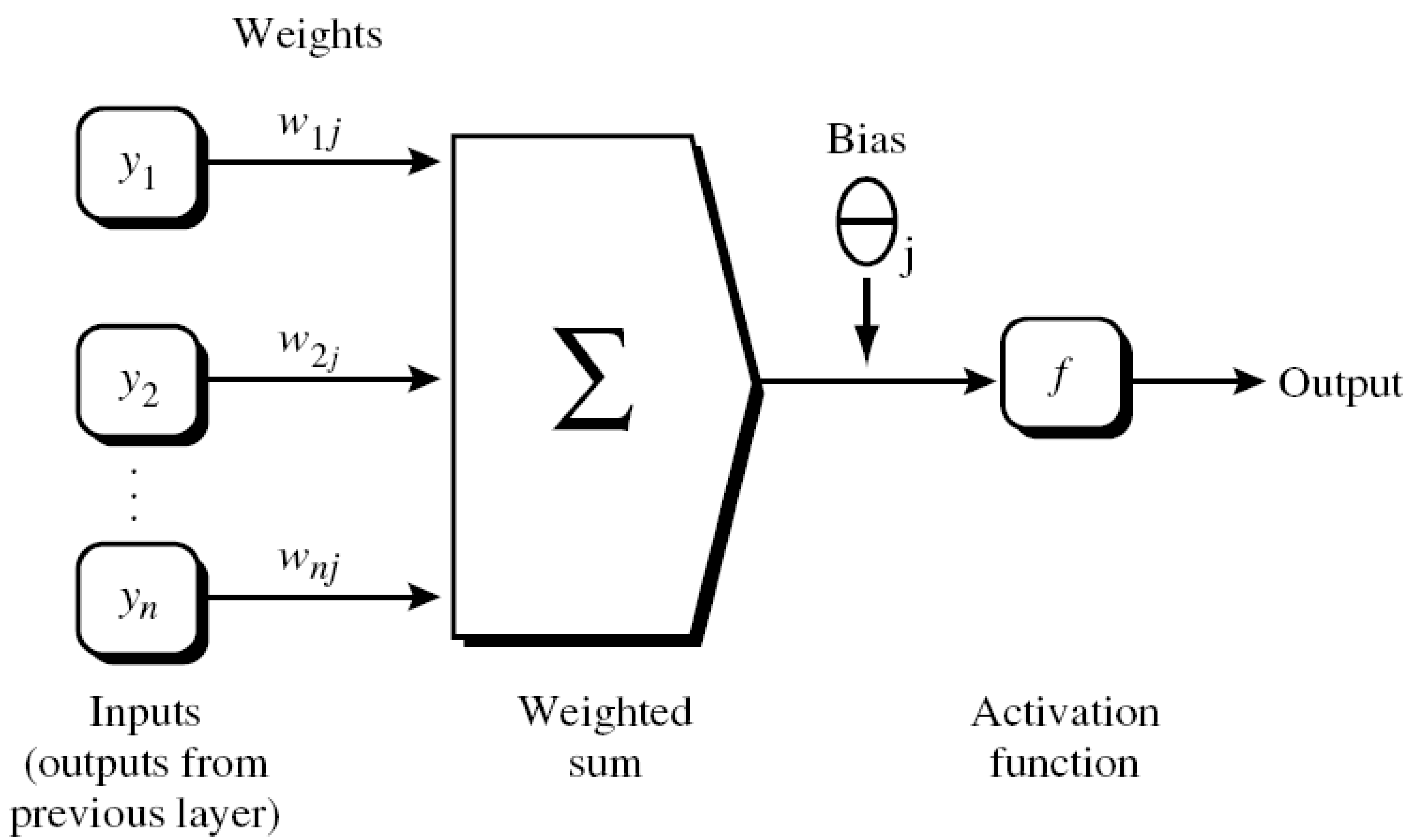
- **3) compute the net input of each unit in the hidden and output layers**
  - The net input to a unit in the hidden or output layers is computed as a linear combination of its inputs.
  - Given a unit *j* in a hidden or output layer, the net input, I_j, to unit *j* is

$$I_j = \sum_i w_{ij} O_i + \theta_j$$

    - where w_{ij} is the weight of the connection from unit i in the previous layer to unit j
    - O_i is the output of unit i from the previous layer
    - Θ_j is the bias of the unit

Propagate the inputs forward

- A hidden or output layer unit j



Propagate the inputs forward

- **4) compute the output of each unit j in the hidden and output layers**
  - The output of each unit is calculating by applying an **activation function** to its net input
  - The **logistic**, or **sigmoid**, function is used.
  - Given the net input I_j to unit j, then O_j, the output of unit j, is computed as:

$$O_j = \frac{1}{1 + e^{-I_j}}$$

Backpropagate the Error

- **5) compute the error for each unit j in the output layer**
  - For a unit *j* in the output layer, the error $\boldsymbol{Err_j}$ is computed by

$$Err_j = O_j(1 - O_j)(T_j - O_j)$$

  - O_j is the actual output of unit j,
  - T_j is the known target value of the given training example
  - Note that O_j (1 - O_j) is the derivative of the logistic function.

Backpropagate the Error

- **6) compute the error for each unit j in the hidden layers, from the last to the first hidden layer**
  - The error of a hidden layer unit j is

$$Err_j = O_j(1-O_j)\sum_k Err_k w_{jk}$$

  - w_{jk} is the weight of the connection from unit j to a unit k in the next higher layer, and
  - Err_k is the error of unit k.

Backpropagate the Error

- **7) update the weights for each weight w_{ij} in network**
  - Weights are updated by the following equations

$$w_{ij} = w_{ij} + \Delta w_{ij}$$

$$\Delta w_{ij} = (l) Err_j O_i$$

  - Δw_{ij} is the change in weight w_{ij}
  - The variable l is the **learning rate**, a constant typically having a value between 0.0 and 1.0

Backpropagate the Error

- **Learning rate**
  - Backpropagation learns using a method of gradient descent
  - The learning rate helps avoid getting stuck at a local minimum in decision space (i.e., where the weights appear to converge, but are not the optimum solution) and encourages finding the global minimum.
  - If the learning rate is **too small**, then learning will occur at a very slow pace.
  - If the learning rate is **too large**, then oscillation between inadequate solutions may occur.
  - A rule to set the learning rate to **1 / t**, where t is the number of iterations through the training set so far.

Backpropagate the Error

- **8) update the for each bias $\boldsymbol{\Theta_j}$ in network**
  - Biases are updated by the following equations below:

$$\theta_j = \theta_j + \Delta\theta_j$$

$$\Delta\theta_j = (l)Err_j$$

  - $\Delta\Theta_j$ is the change in bias Θ_j
  - There are two strategies for updating the weights and biases

Backpropagate the Error

- **Updating strategies:**
  - **Case updating**
    - updating the weights and biases after the presentation of each example.
    - case updating is more common because it tends to yield more accurate result
  - **Epoch updating**
    - The weight and bias increments could be accumulated in variables, so that the weights and biases are updated after all of the examples in the training set have been presented.
    - One iteration through the training set is an epoch.

Terminating Condition

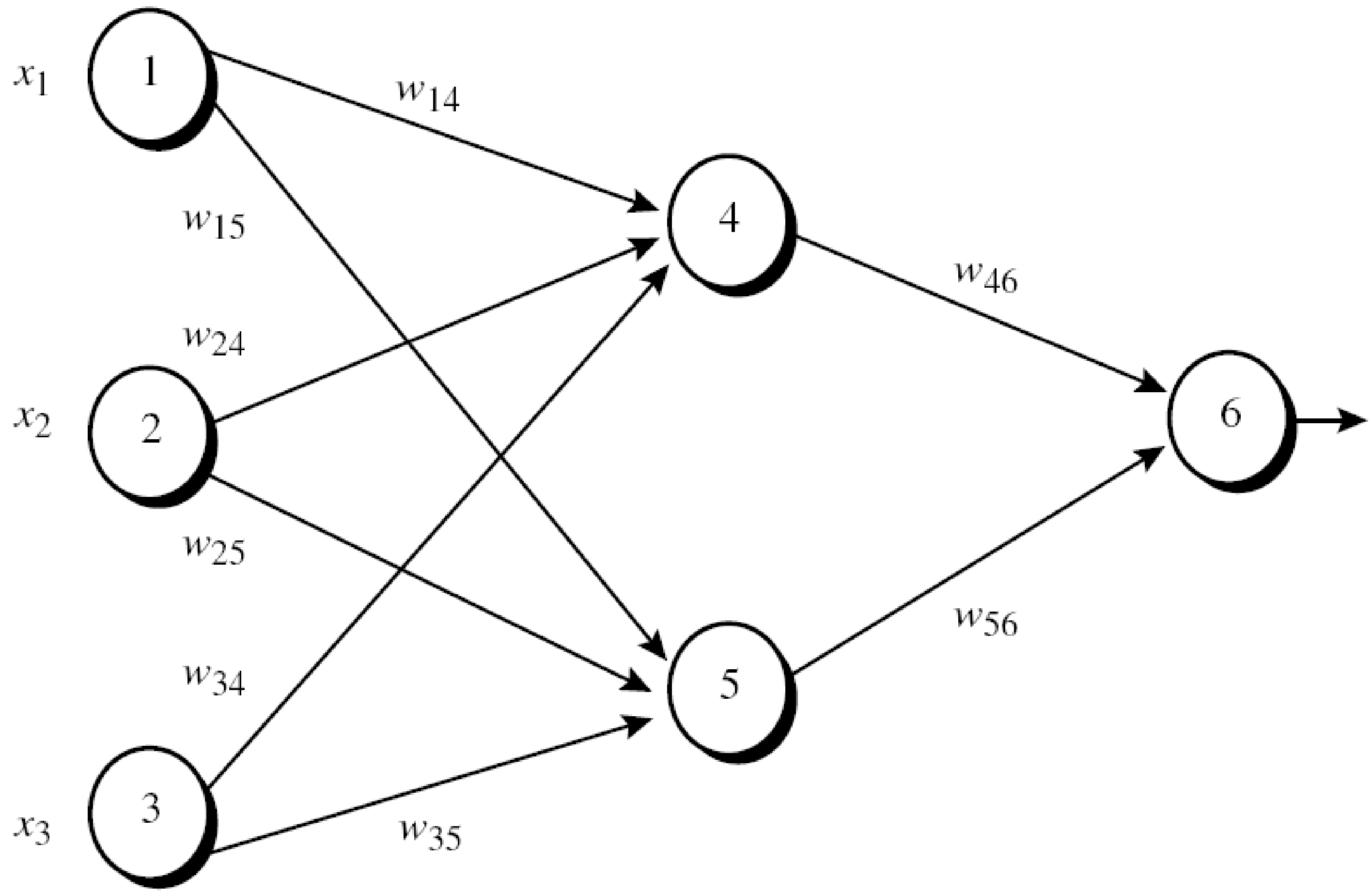
- **9) Checking the stopping condition**
  - After finishing the processed for all training examples, we must evaluate the stopping condition
  - Stopping condition: Training stops when
    - All Δw_{ij} in the previous epoch were so small as to be below some specified threshold, or
    - The percentage of examples misclassified in the previous epoch is below some threshold, or
    - A prespecified number of epochs has expired.
  - In practice, several hundreds of thousands of epochs may be required before the weights will converge.
  - If stopping condition was not true steps 2 to 8 should repeat for all training examples

Efficiency of Backpropagation

- The computational efficiency depends on the time spent training the network.
- However, in the worst-case scenario, the number of epochs can be exponential in n, the number of inputs.
- In practice, the time required for the networks to converge is highly variable.
- A number of techniques exist that help speed up the training time.
  - Metaheuristic algorithms such as **simulated annealing algorithm** can be used, which also ensures convergence to a global optimum.

Example

- The Figure shows a multilayer feed-forward neural network



Example

- This example shows the calculations for backpropagation, given the first training example, X.
- Let the **learning rate** be 0.9.
- The initial weight and bias values of the network are given in the Table, along with the first training example, X = (1, 0, 1), whose class label is 1.

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

Example

- The net input and output calculations:

$$I_j = \sum_i w_{ij} O_i + \theta_j$$

$$O_j = \frac{1}{1 + e^{-I_j}}$$

Unit j	Net input, I_j	Output, O_j
4	$0.2 + 0 - 0.5 - 0.4 = -0.7$	$1/(1 + e^{0.7}) = 0.332$
5	$-0.3 + 0 + 0.2 + 0.2 = 0.1$	$1/(1 + e^{-0.1}) = 0.525$
6	$(-0.3)(0.332) - (0.2)(0.525) + 0.1 = -0.105$	$1/(1 + e^{0.105}) = 0.474$

Example

- Calculation of the error at each node:
  - **The output layer**

    $$Err_j = O_j(1-O_j)(T_j - O_j)$$

  - **The hidden layer**

    $$Err_j = O_j(1-O_j)\sum_k Err_k w_{jk}$$

Unit j	Err_j
6	$(0.474)(1-0.474)(1-0.474) = 0.1311$
5	$(0.525)(1-0.525)(0.1311)(-0.2) = -0.0065$
4	$(0.332)(1-0.332)(0.1311)(-0.3) = -0.0087$

Example

- Calculations for weight and bias updating:

Weight or bias	*New value*
w_{46}	$-0.3+(0.9)(0.1311)(0.332)=-0.261$
w_{56}	$-0.2+(0.9)(0.1311)(0.525)=-0.138$
w_{14}	$0.2+(0.9)(-0.0087)(1)=0.192$
w_{15}	$-0.3+(0.9)(-0.0065)(1)=-0.306$
w_{24}	$0.4+(0.9)(-0.0087)(0)=0.4$
w_{25}	$0.1+(0.9)(-0.0065)(0)=0.1$
w_{34}	$-0.5+(0.9)(-0.0087)(1)=-0.508$
w_{35}	$0.2+(0.9)(-0.0065)(1)=0.194$
θ_6	$0.1+(0.9)(0.1311)=0.218$
θ_5	$0.2+(0.9)(-0.0065)=0.194$
θ_4	$-0.4+(0.9)(-0.0087)=-0.408$

$$w_{ij} = w_{ij} + \Delta w_{ij}$$

$$\Delta w_{ij} = (l)Err_j O_i$$

$$\Delta \theta_j = (l)Err_j$$

$$\theta_j = \theta_j + \Delta \theta_j$$

Example

- Several variations and alternatives to the backpropagation algorithm have been proposed for classification in neural networks.
- These may involve:
  - the dynamic adjustment of the network topology and of the learning rate
  - New parameters
  - The use of different error functions

Backpropagation and Interpretability

Backpropagation and Interpretability

- Neural networks are like a black box.
- A major disadvantage of neural networks lies in their knowledge representation.
- Acquired knowledge in the form of a network of units connected by weighted links is difficult for humans to interpret.
- This factor has motivated research in extracting the knowledge embedded in trained neural networks and in representing that knowledge symbolically.
- Methods include:
  - **extracting rules from networks**
  - **sensitivity analysis**

Backpropagation and Interpretability

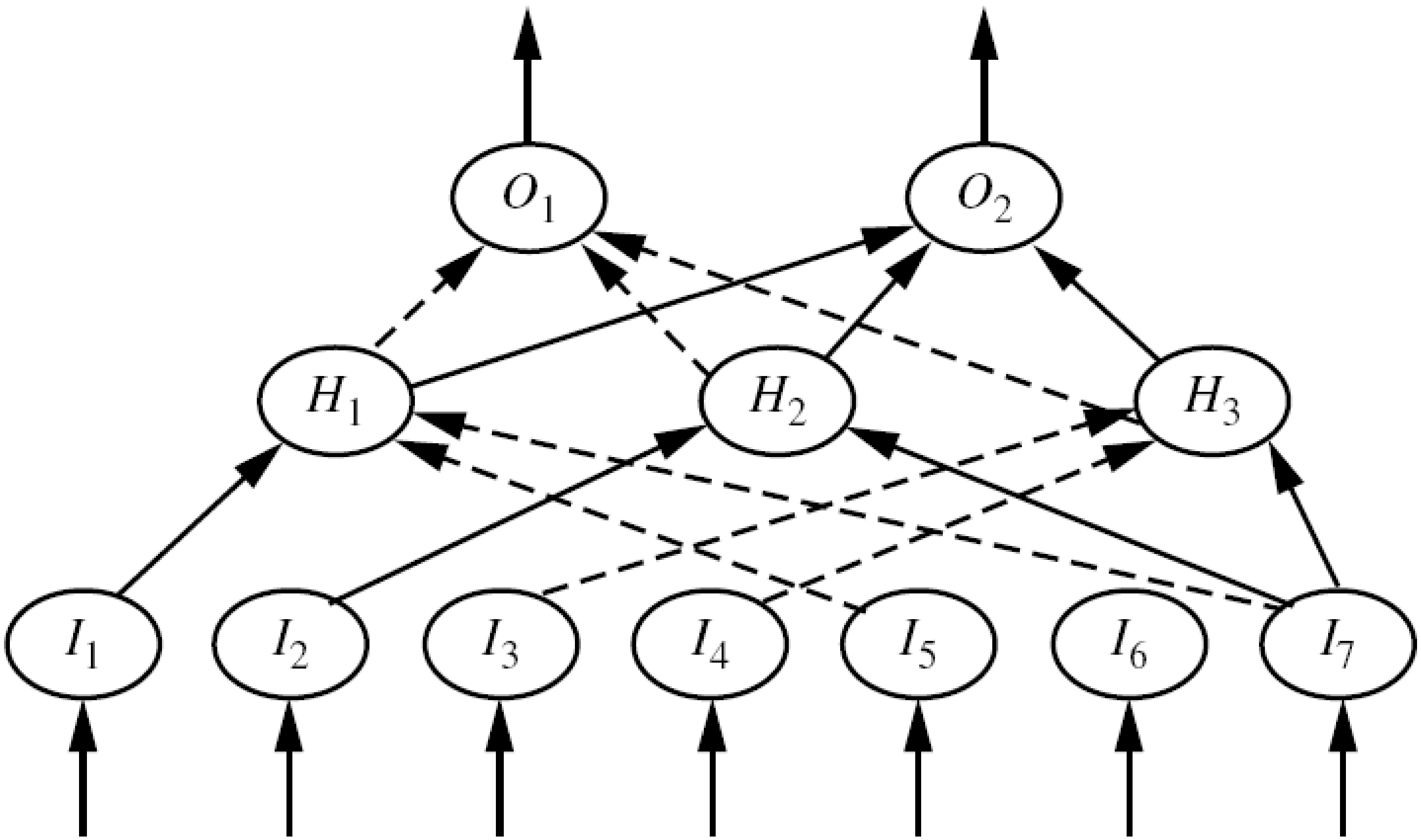
- Often the first step toward extracting rules from neural networks is **network pruning**
  - This consists of simplifying the network structure by removing weighted links that have the least effect on the trained network.

Backpropagation and Interpretability

- Rule extraction from networks
  - Often, the first step toward extracting rules from neural networks is **network pruning**.
    - This consists of simplifying the network structure by removing weighted links that have the least effect on the trained network
  - Then perform link, unit, or activation value clustering
    - In one method, for example, clustering is used to find the set of common activation values for each hidden unit in a given trained two-layer neural network.
  - The set of input and activation values are studied to derive rules describing the relationship between the input and hidden unit layers

Backpropagation and Interpretability

- Rules can be extracted from training neural networks



Backpropagation and Interpretability

Identify sets of common activation values for each hidden node, H_i: for H_1: (–1,0,1) for H_2: (0.1) for H_3: (–1,0.24,1)
Derive rules relating common activation values with output nodes, O_j: IF ($H_2 = 0$ AND $H_3 = -1$) OR ($H_1 = -1$ AND $H_2 = 1$ AND $H_3 = -1$) OR ($H_1 = -1$ AND $H_2 = 0$ AND $H_3 = 0.24$) THEN $O_1 = 1$, $O_2 = 0$ ELSE $O_1 = 0$, $O_2 = 1$

Derive rules relating input nodes, I_j, to output nodes, O_j:
IF ($I_2 = 0$ AND $I_7 = 0$) THEN $H_2 = 0$
IF ($I_4 = 1$ AND $I_6 = 1$) THEN $H_3 = -1$
IF ($I_5 = 0$) THEN $H_3 = -1$

Obtain rules relating inputs and output classes:
IF ($I_2 = 0$ AND $I_7 = 0$ AND $I_4 = 1$ AND $I_6 = 1$) THEN class = 1
IF ($I_2 = 0$ AND $I_7 = 0$ AND $I_5 = 0$) THEN class = 1

Backpropagation and Interpretability

- **Sensitivity analysis**
  - assess the impact that a given input variable has on a network output.
  - The knowledge gained from this analysis can be represented in rules
  - Such as "IF X decreases 5% THEN Y increases 8%."

Discussion

Discussion

- Weakness of neural networks
  - Long training time
  - Require a number of parameters typically best determined empirically
    - e.g., the network topology or structure.
  - Poor interpretability
    - Difficult to interpret the symbolic meaning behind the learned weights and of "hidden units" in the network

Discussion

- Strength of neural networks
  - High tolerance to noisy data
  - It can be used when you may have little knowledge of the relationships between attributes and classes
  - Well-suited for continuous-valued inputs and outputs
  - Successful on a wide array of real-world data
  - Algorithms are inherently parallel
  - Techniques have recently been developed for the extraction of rules from trained neural networks

Research Areas

- **Finding optimal network structure**
  - e.g. by genetic algorithms
- **Increasing learning speed (efficiency)**
  - e.g. by simulated annealing
- **Increasing accuracy (effectiveness)**
- **Extracting rules from networks**

References

References

- J. Han, M. Kamber, **Data Mining: Concepts and Techniques**, Elsevier Inc. (2006). (Chapter 6)
- S. J. Russell and P. Norvig, **Artificial Intelligence, A Modern Approach**, Prentice Hall,1995. (Chapter 19)

The end