# Biologically Inspired Computing: Neural Computation

## Lecture 3

Patricia A. Vargas

# Lecture 3

I. Lecture 2 – Revision
II. Artificial Neural Networks (Part II)
    I. Learning Paradigms
    II. Perceptron

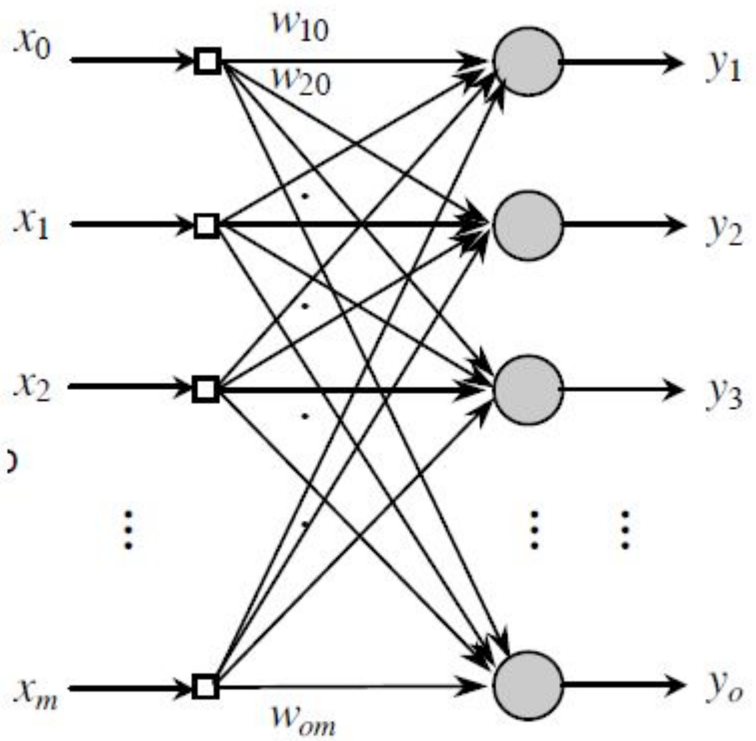
# Artificial Neural Networks (ANN)

- History

| | |
|---|---|
| 1943 | McCulloch e Pitts |
| 1948 | Wiener |
| 1949 | Hebb |
| 1957 | Rosenblatt |
| 1958 | Widrow e Hoff |
| … | … |
| 1969 | Minsky e Papert |
| … | … |
| 1960-1980 | Kohonen, Grossberg, Widrow, Anderson, Caianiello, Fukushima, Aleksander |
| … | … |
| 1974 | Werbos |
| … | … |
| 1982 | Hopfield |
| 1986 | Rumelhart e McClelland |

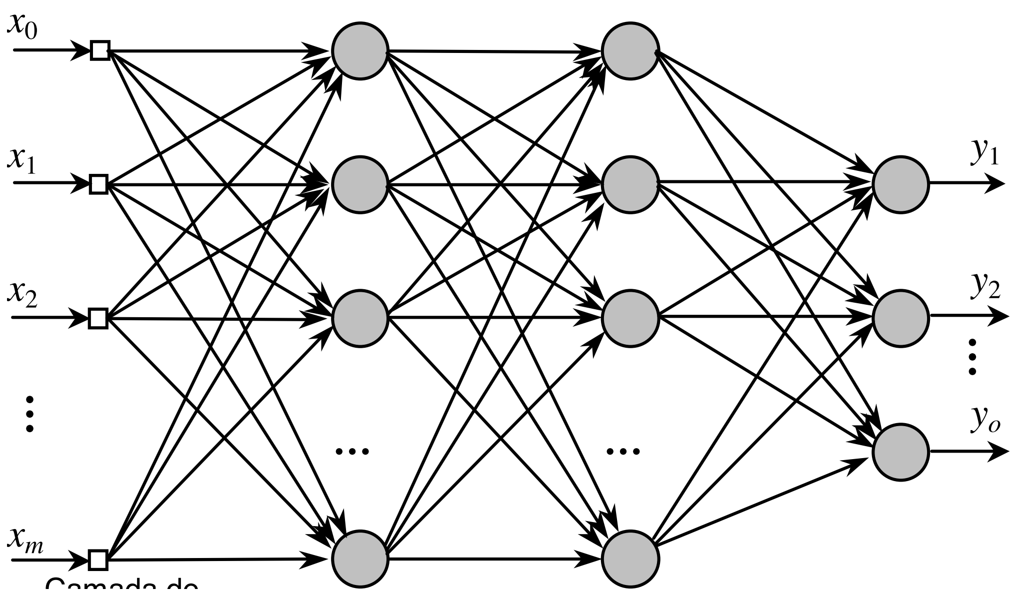
# Artificial Neural Networks

- Architectures: Single-layer Feedforward Networks



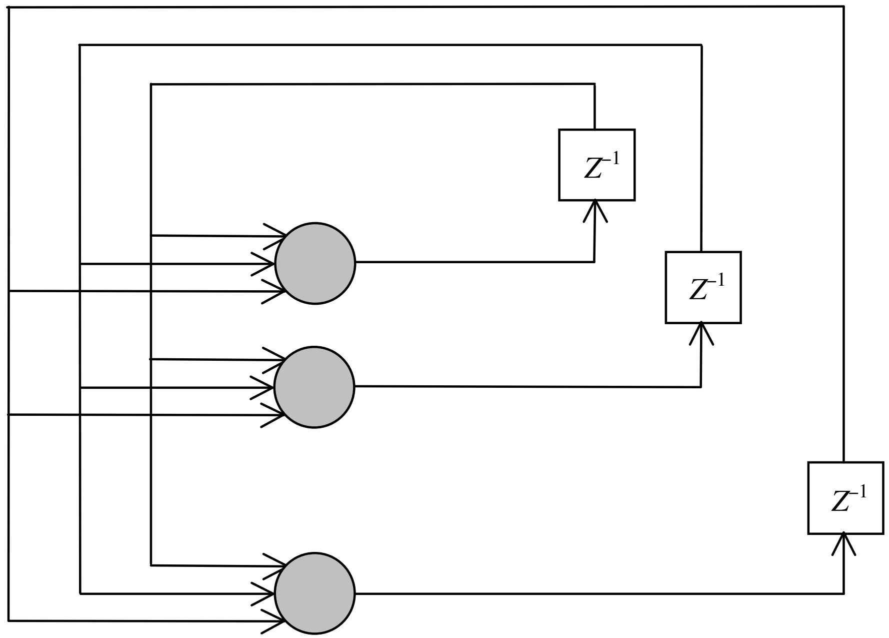
# Artificial Neural Networks

- Architectures: Multilayer Feedforward Networks



# Artificial Neural Networks

- Architectures: Recurrent Neural Networks




Ex: Hopfield Neural Network

# Artificial Neural Networks

- Learning Paradigms

$$w(t+1) = w(t) + \Delta w(t)$$

I. Supervised Learning

II. Unsupervised Learning

III. Reinforcement Learning

# Artificial Neural Networks

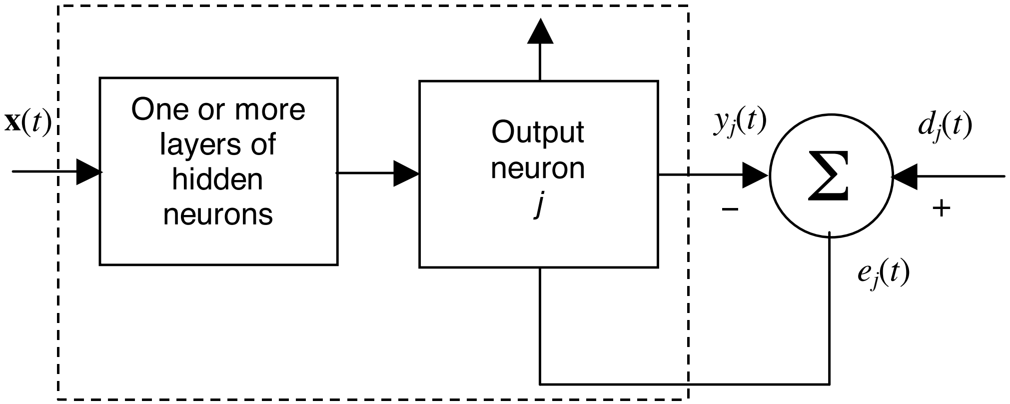
## I. Supervised Learning ?

$$w(t+1) = w(t) + \Delta w(t)$$

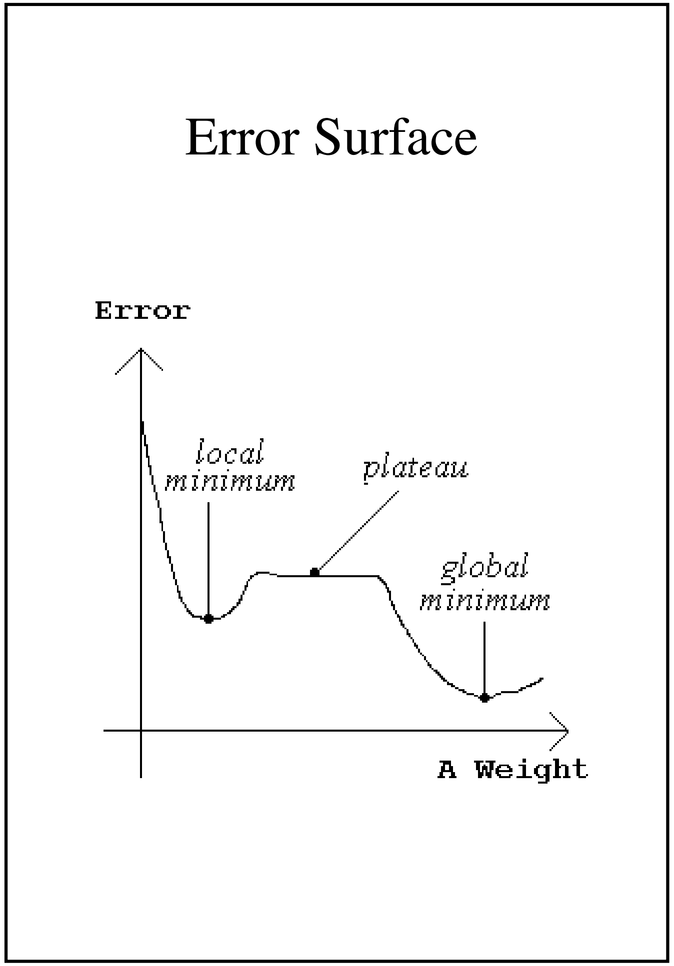
# Artificial Neural Networks

## I. Supervised Learning

$$w(t+1) = w(t) + \Delta w(t)$$



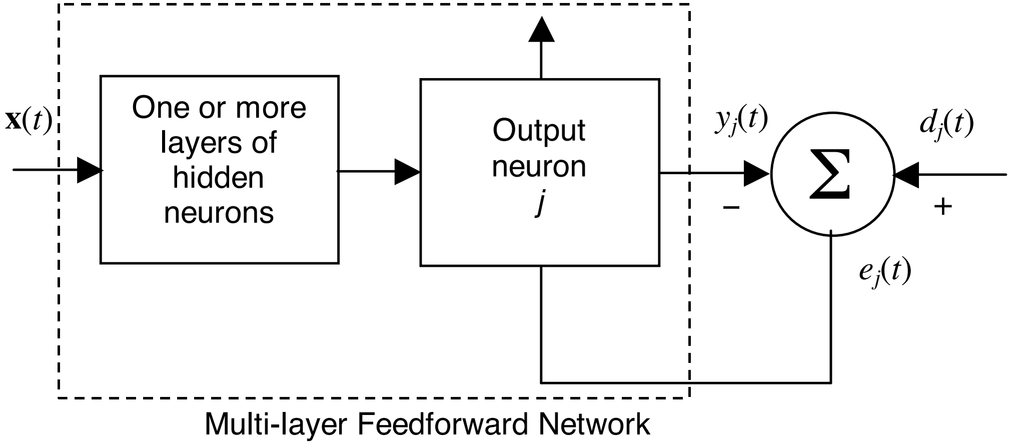
Multi-layer Feedforward Network



# Artificial Neural Networks

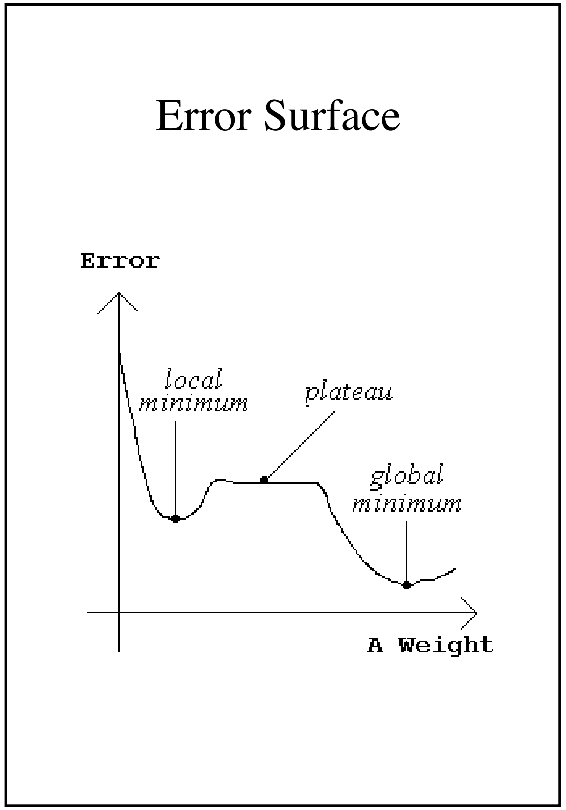
## I. Supervised Learning

- based on a set of examples of input-output mapping, i.e. input and desired output pairs
- there is a supervisor/teacher




Multi-layer Feedforward Network

$$w(t+1) = w(t) + \Delta w(t)$$



# Artificial Neural Networks

II. Unsupervised Learning ?

# Artificial Neural Networks

## II. Unsupervised/Self-organised Learning

- there is no supervisor/teacher and thus no error value
- based only on the stimuli the network receives
- no targets for the outputs
- networks which discover patterns, correlations, etc. in the input data (the ANN needs to learn how to categorise the stimuli)
- this is a self-organisation process
- usually employs a competitive learning algorithm

# Artificial Neural Networks

III. Reinforcement Learning?

# Artificial Neural Networks

## III. Reinforcement Learning

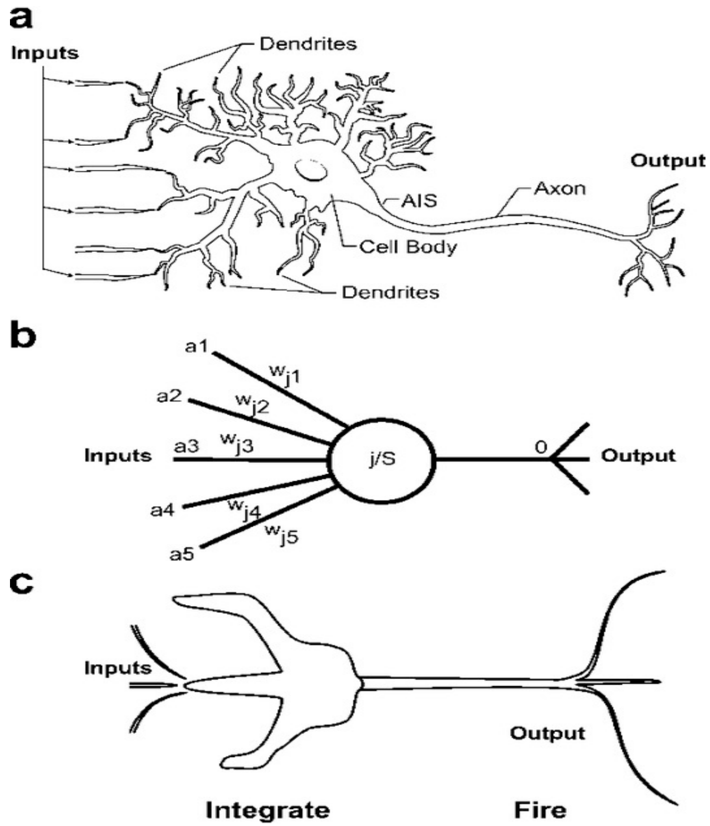
- based on goal-directed learning from interaction
- "is learning what to do--how to map situations to actions--so as to maximize a numerical reward signal"(Sutton & Barto, 1998)
- there is no supervisor and no explicit model of the environment

eBook: http://webdocs.cs.ualberta.ca/~sutton/book/ebook/index.html

# Artificial Neural Networks

| | |
|---|---|
| 1943 | McCulloch e Pitts |
| 1948 | Wiener |
| 1949 | Hebb |
| 1957 | Rosenblatt |
| 1958 | Widrow e Hoff |
| ... | ... |
| 1969 | Minsky e Papert |
| ... | ... |
| 1960-1980 | Kohonen, Grossberg, Widrow, Anderson, Caianiello, Fukushima, Aleksander |
| ... | ... |
| 1974 | Werbos |
| ... | ... |
| 1982 | Hopfield |
| 1986 | Rumelhart e McClelland |

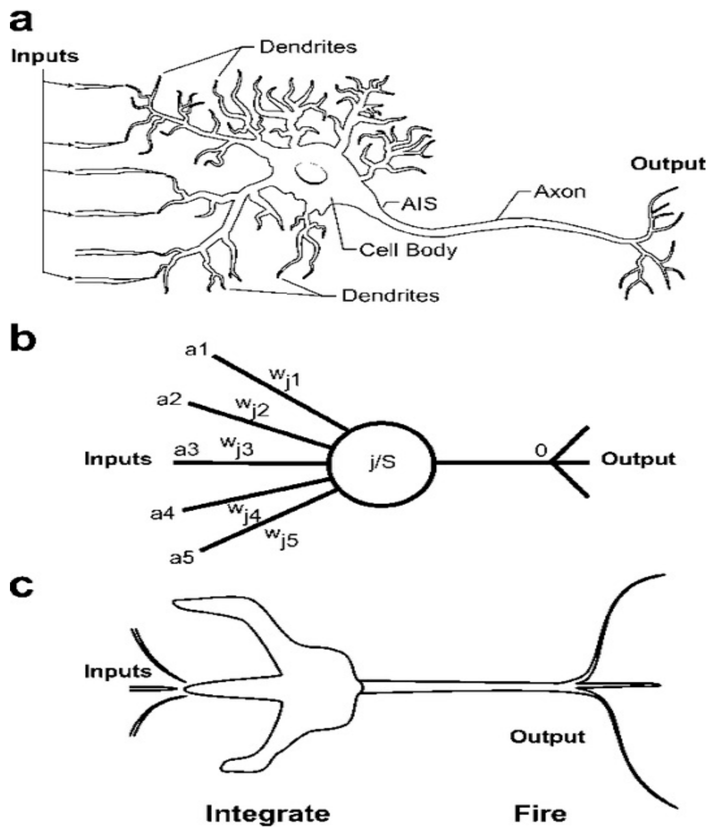
- Frank Rosenblatt (1957)
  - Perceptron

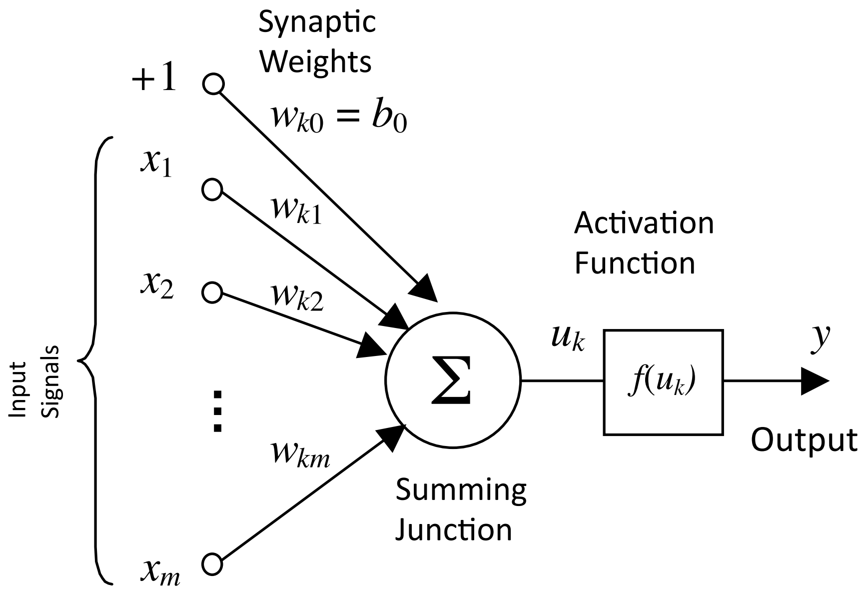



  - Rosenblatt, F. (1958), "The perceptron: A probabilistic model for information storage and organization in the brain, Psychological Review, v65, n6, pp: 386-408.

# Artificial Neural Networks

- Frank Rosenblatt (1957)
  - Perceptron

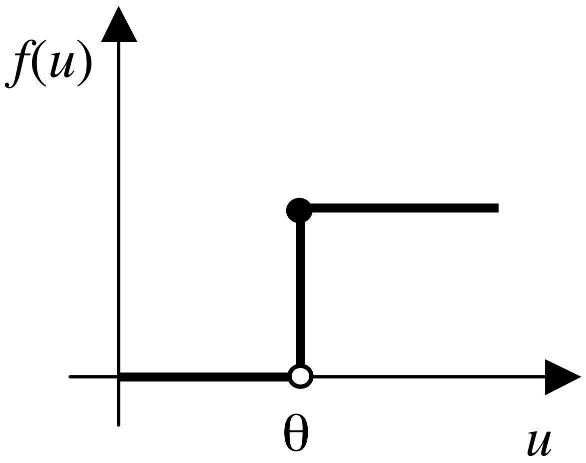


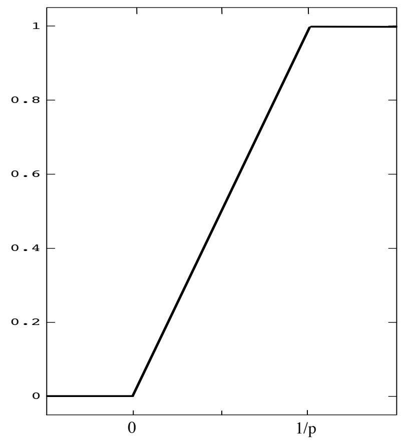



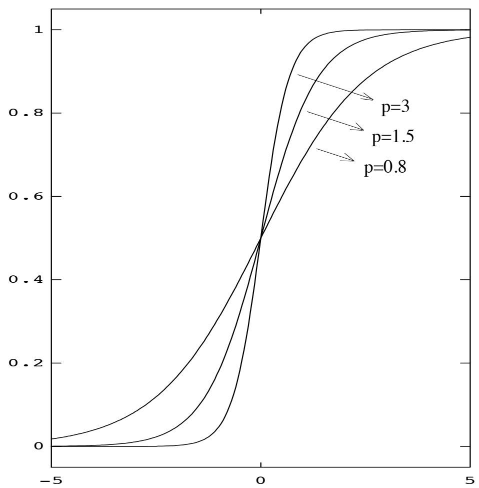

$$y_k = f(u_k) = f\left(\sum_{j=0}^{m} w_{kj} x_j\right)$$

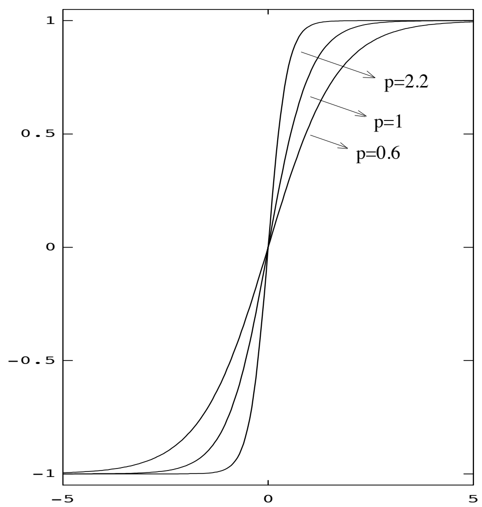
# Artificial Neural Networks

- Activation functions



# Two main forms of learning

- Supervised Learning
  - Error-correcting learning
    - Perceptron
      - delta rule
    - Multi-Layer Perceptron (MLP)
      - Backpropagation (generalized delta rule)

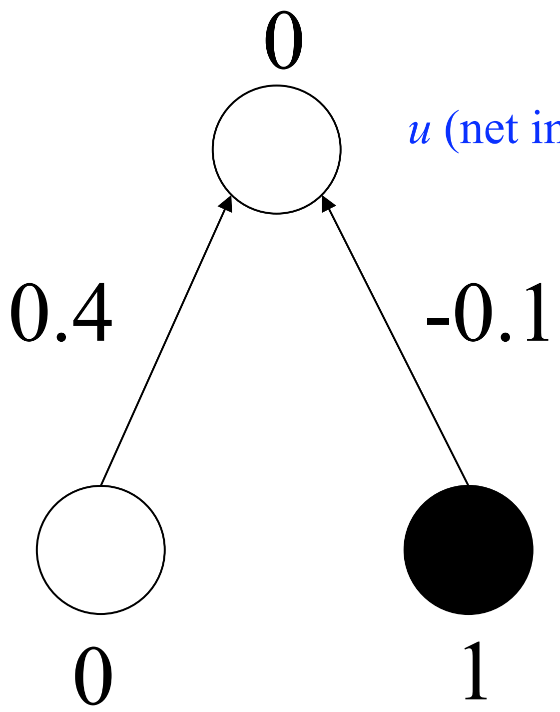
- Unsupervised Learning
  - Associative (Hebbian) learning

# The Perceptron by Frank Rosenblatt (1958, 1962)

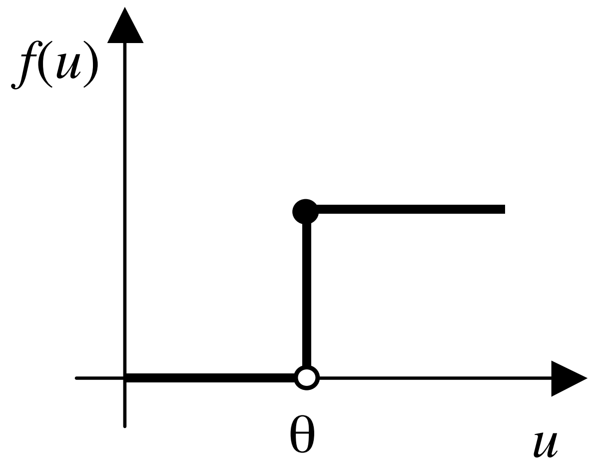
- binary nodes (McCulloch-Pitts nodes) that take values 0 or 1
- continuous weights, initially chosen randomly

# Very simple example

$$y_k = f(u_k) = f\left(\sum_{j=0}^{m} w_{kj} x_j\right)$$




$u$ (net input) = 0.4 × 0 + -0.1 × 1 = -0.1

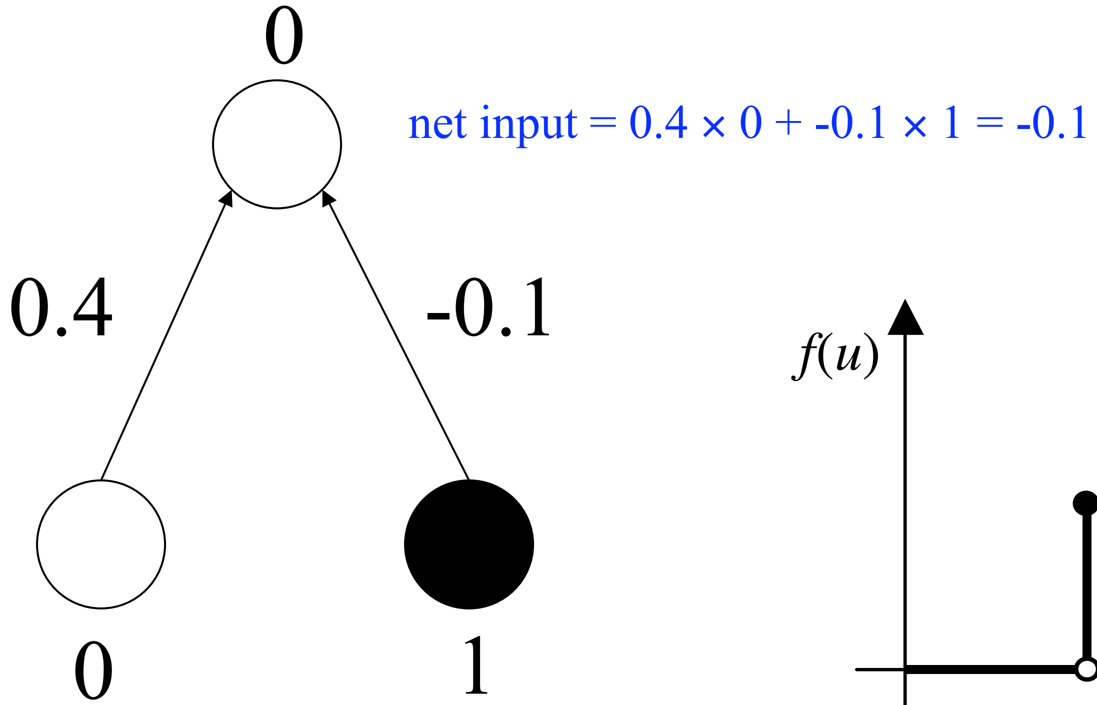


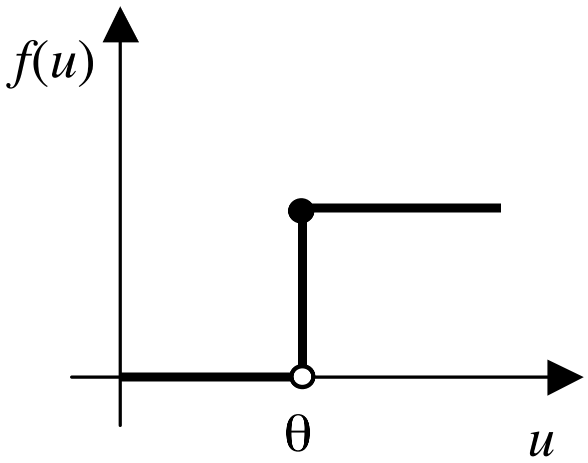
# Learning problem to be solved

- Suppose we have an input pattern (0 1)
- We have a single output pattern (1)
- We have a net input of -0.1, which gives an output pattern of (0)
- How could we adjust the weights, so that this situation is remedied and the spontaneous output matches our target output pattern of (1)?

# Learning problem to be solved

- How could we adjust the weights, so that this situation is remedied and the spontaneous output matches our target output pattern of (1)?



# Answer

- Increase the weights, so that the net input exceeds 0.0
- E.g., add 0.2 to all weights
- Observation: Weight from input node with activation 0 does not have any effect on the net input
- So we will leave it alone

# Perceptron algorithm in words

For each node in the output layer:

- Calculate the error, which can only take the values -1, 0, and 1
- If the error is 0, the goal has been achieved. Otherwise, we adjust the weights
- Do not alter weights from inactivated input nodes

# Perceptron algorithm in rules

- weight change = some small constant × (target activation - spontaneous output activation) × input activation
- if speak of *error* instead of the "target activation minus the spontaneous output activation", we have:
- weight change = some small constant × error × input activation

# Perceptron algorithm as equation

- If we call the input node *i* and the output node *j* we have:

$$\Delta w_{ji} = \mu\,(t_j - a_j)\,a_i = \mu\,\delta_j a_i$$

- $\Delta w_{ji}$ is the weight change of the connection from node *i* to node *j*
- $a_i$ is the activation of node *i*, $a_j$ of node *j*
- $t_j$ is the target value for node *j*
- $\delta_j$ is the error for node *j*
- The learning constant $\mu$ is typically chosen small (e.g., 0.1).

# Perceptron algorithm in pseudo-code

```
Start with random initial weights (e.g., uniform random in [-.3,.3])

Do
{
  For All Patterns p
  {
    For All Output Nodes j
    {
      CalculateActivation(j)

      Error_j = TargetValue_j_for_Pattern_p - Activation_j

      For All Input Nodes i To Output Node j
      {
        DeltaWeight = LearningConstant * Error_j * Activation_i
        Weight = Weight + DeltaWeight
      }
    }
  }
}
Until "Error is sufficiently small" Or "Time-out"
```

# Perceptron convergence theorem

- *If* a pattern set can be represented by a Perceptron, …
- the Perceptron learning rule will always be able to find some correct weights

- Example:

http://lcn.epfl.ch/tutorial/english/perceptron/html/index.html

# The Perceptron was a big hit

- Responsible for the first wave in ‘connectionism’
- Great interest and optimism about the future of neural networks
- First neural network hardware was built in the late fifties and early sixties

# Limitations of the Perceptron

- Can only represent linear separable problems…

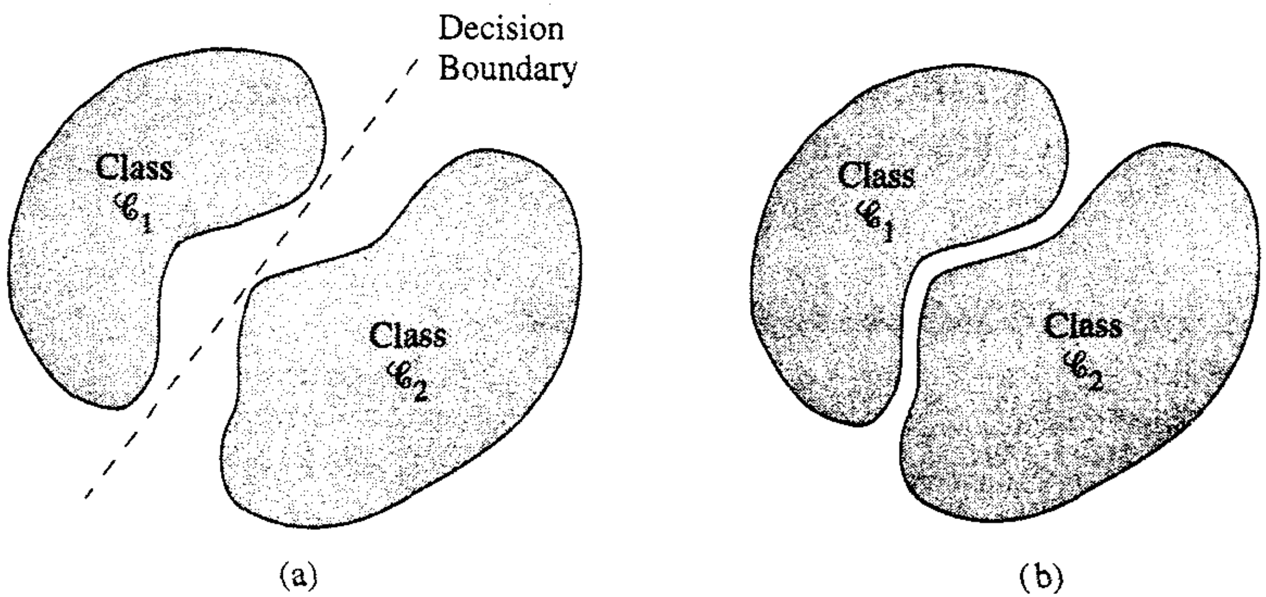


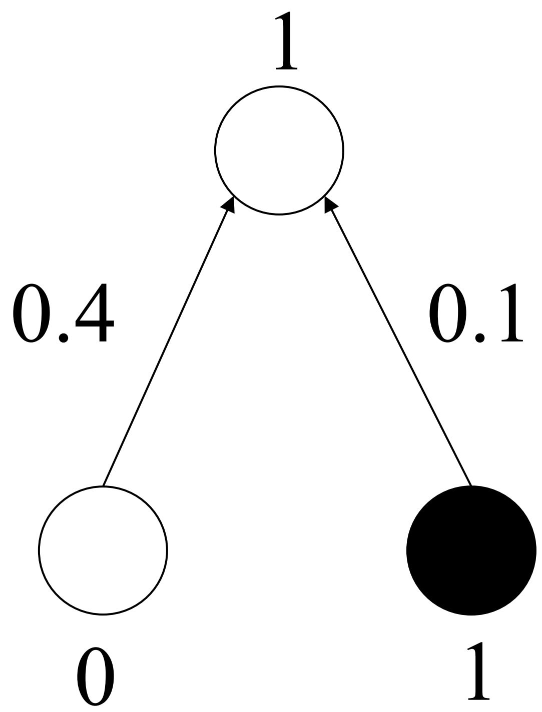

**FIGURE 3.9** (a) A pair of linearly separable patterns. (b) A pair of non-linearly separable patterns.

# Limitations of the Perceptron

- Minsky and Papert (1969) showed that a Perceptron cannot represent certain logical functions
- Some of these are very fundamental, in particular the exclusive or (XOR)

# Exclusive OR (XOR)

| In | Out |
|---|---|
| 0 1 | 1 |
| 1 0 | 1 |
| 1 1 | 0 |
| 0 0 | 0 |




Let us try it in our applet:
http://lcn.epfl.ch/tutorial/english/perceptron/html/index.html

# Minsky and Papert book caused the 'first wave' to die out

- GOOFAI was increasing in popularity
- Neural networks were very much out
- A few hardy pioneers continued
- Within five years a variant was developed by Paul Werbos that was immune to the XOR problem, but few noticed this
- Even in Rosenblatt's book many examples of more sophisticated Perceptrons are given that can learn the XOR

# An extra layer is necessary to represent the XOR

- No solid training procedure existed in 1969 to accomplish this
- Thus commenced the search for the hidden layer…

# Error-backpropagation ?

- What was needed, was an algorithm to train Perceptrons with more than two layers
- Preferably also one that used continuous activations and non-linear activation rules
- Such an algorithm was developed by
  - Paul Werbos in 1974
  - David Parker in 1982
  - LeCun in 1984
  - Rumelhart, Hinton, and Williams in 1986

# Lecture 3

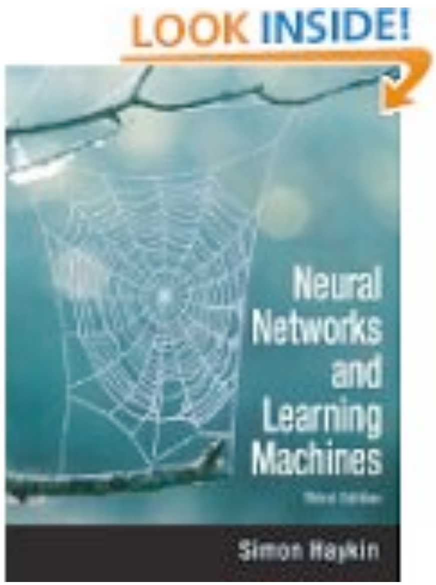
I. Lecture 2 – Revision
II. Artificial Neural Networks (Part II)
    I. Learning Paradigms
    II. Perceptron

# Lecture 3

## Reading list/Homework

- Read Introduction chapter: I.8 ("Learning Processes") and
- Chapter 1 ("Rosenblatt's Perceptron"): 1.1, 1.2 and 1.7 (inclusive) from the book:

  "Neural Networks and Learning Machines"
  Simon O. Haykin (Nov 28, 2008)




- Answer questions 7 to 16 from the Tutorial material

# Lecture 4

## What's next?

## Artificial Neural Networks

## (Part III)