# Introduction to Artificial Intelligence

Department of Computer Science

Lecture 18

- Learning

# Machine Learning

- All learning is learning *how* to do something, if only how to answer a question.
- Thus, in learning we must have some **performance element** in mind
- A **learning element** creates or modifies some data structure that the performance element uses.
- This (hopefully) results in better performance
  - speed
  - accuracy
  - coverage

# Bias and Lunch

- We want a learning algorithm to generalize, not just memorize
- But, in general, you can't generalize
  - f(1) = 1, f(2) = 2, f(4) = 16, f(3) = ??
- unless you make some assumptions, e.g. Ockham's Razor

"There ain't no such thing as a free lunch"

- Bias: a preference order among consistent hyptheses.
  - may be explicit, e.g. done by sorting hypotheses
  - may be implicit, especially in representation

# Contexts for Learning

- Supervised learning
- Reinforcement learning
- Unsupervised learning
- Active exploration

# Concept Learning

Learn a function $E \rightarrow C$

$$E = < a_1, a_2, \ldots, a_n >$$

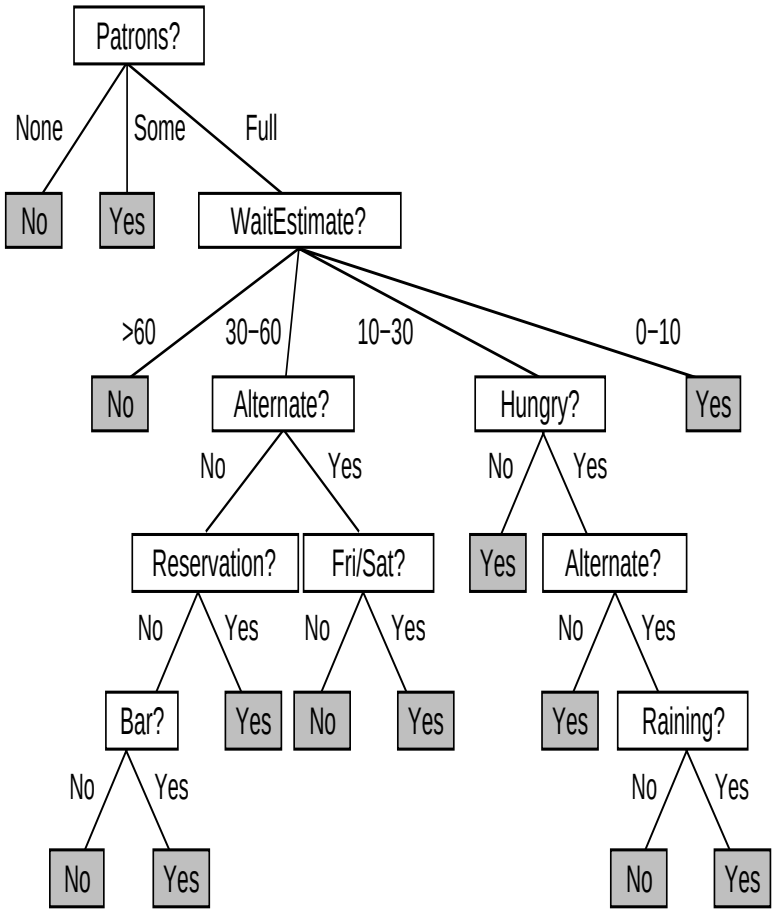
where the $a_i$ is a value for attribute $i$ and

$$C = True, False$$

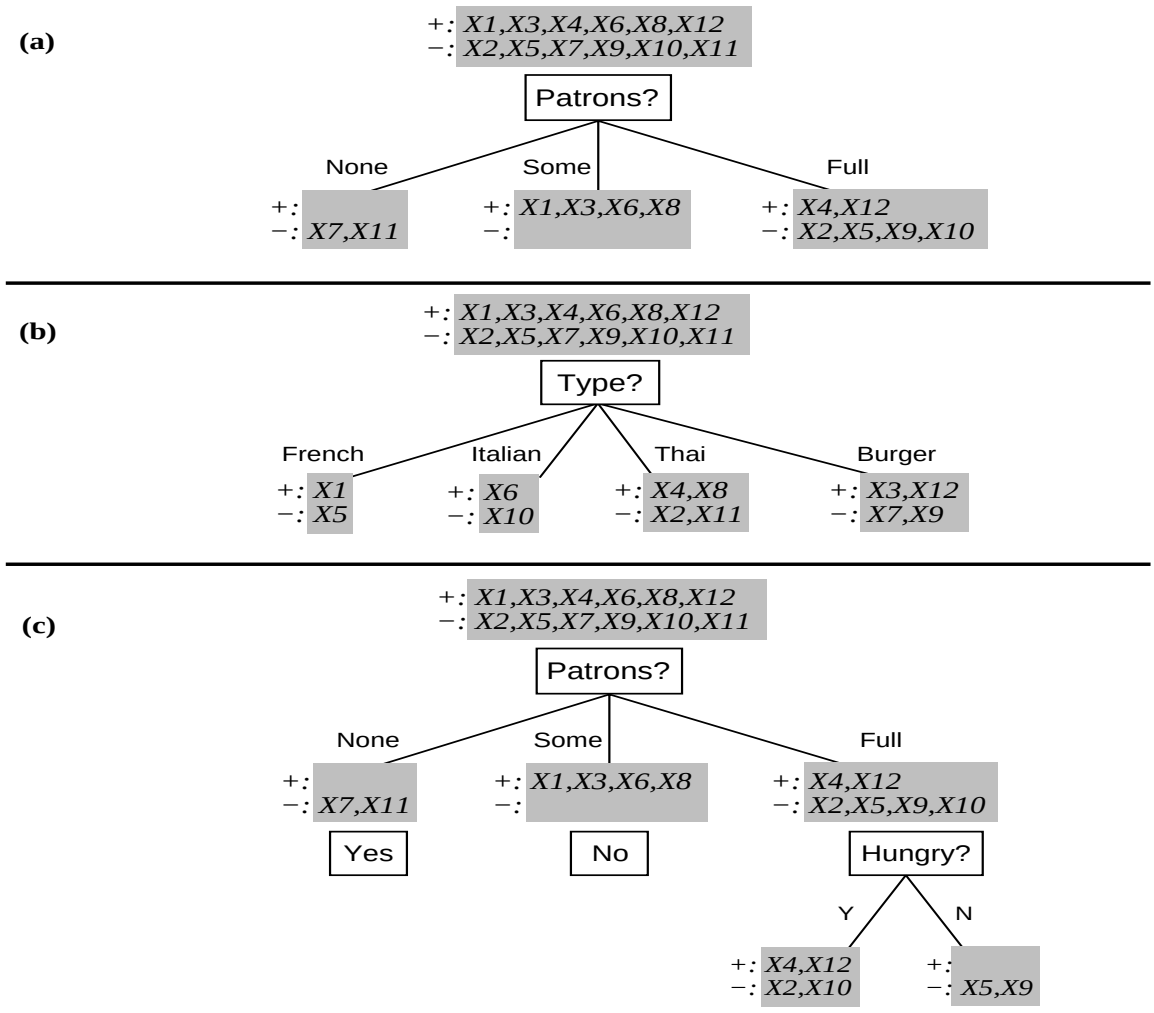
or

$$C = c_1, c_2, \ldots, c_m$$

# Decision Trees



# Building a Decision Tree from Examples



# Computational Learning Theory

What does it mean for a learning mechanism to "work"?

Assume we have

- a set of $m$ training examples
- a set of test examples

drawn from same distribution. An example is a pair $< x, f(x) >$ for some unknown function $f$.

A hypothesis $h$ is a function. If $h = f$ it is clearly a "correct" answer to the learning problem, but this is too strong.

# Approximately Correct Learning

Suppose we accept an $h$ if it is "almost always" right:

$$P(h(x) \neq f(x)) \leq \epsilon$$

Let $H_{bad}$ be set of $h$'s for which this is **not** true.

## Probably Approximately Correct (PAC) Learning

Since our training set is randomly chosen, there is some small chance it will be very misleading

- Let's accept a learning method if it "almost always" chooses an $h$ which is "almost always" correct.

$$P(h \in H_{bad}) \leq \delta$$

- Assume $h$ is consistent with the $m$ training examples.
- Require $h$ to be probably approximately consistent
- for a given $\epsilon$ and $\delta$, how big must $m$ be?

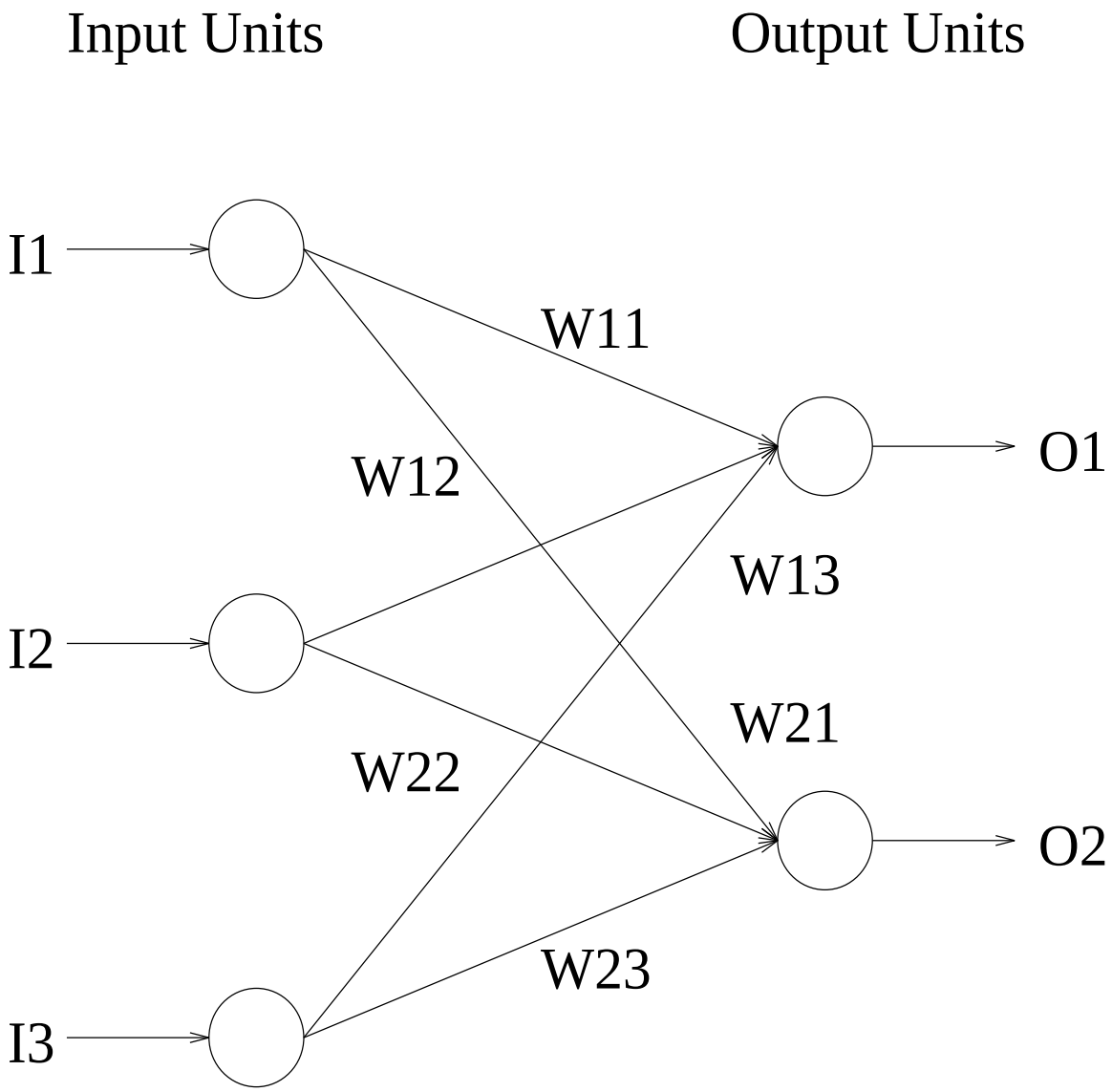
# Neural Networks

- Also Known As:
  - Connectionist Machines.
  - Parallel Distributed Processing.
- Early Work: Perceptrons.
- Current Work: Backpropagation.

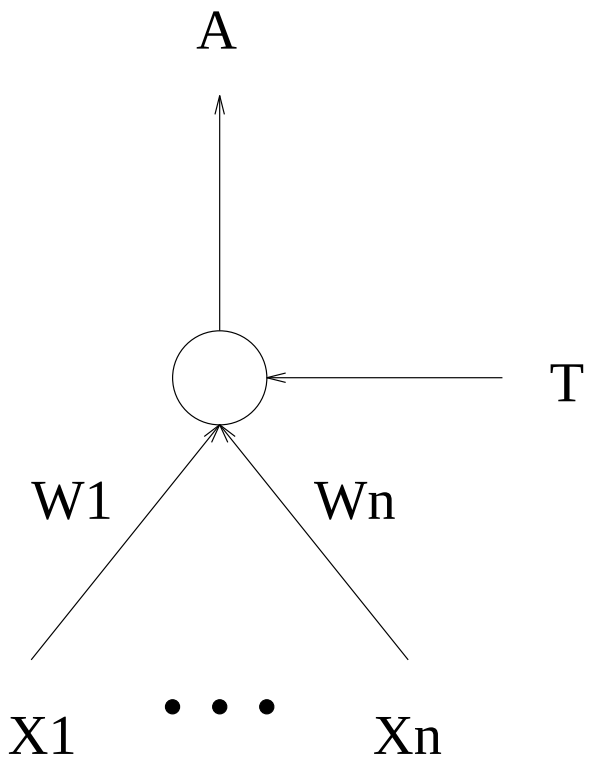
# Perceptron

- Rosenblatt, 1958.
- Very simple computing device.
- Very simple learning device.
- Inspired by rough analogy with neuron.

# Perceptron

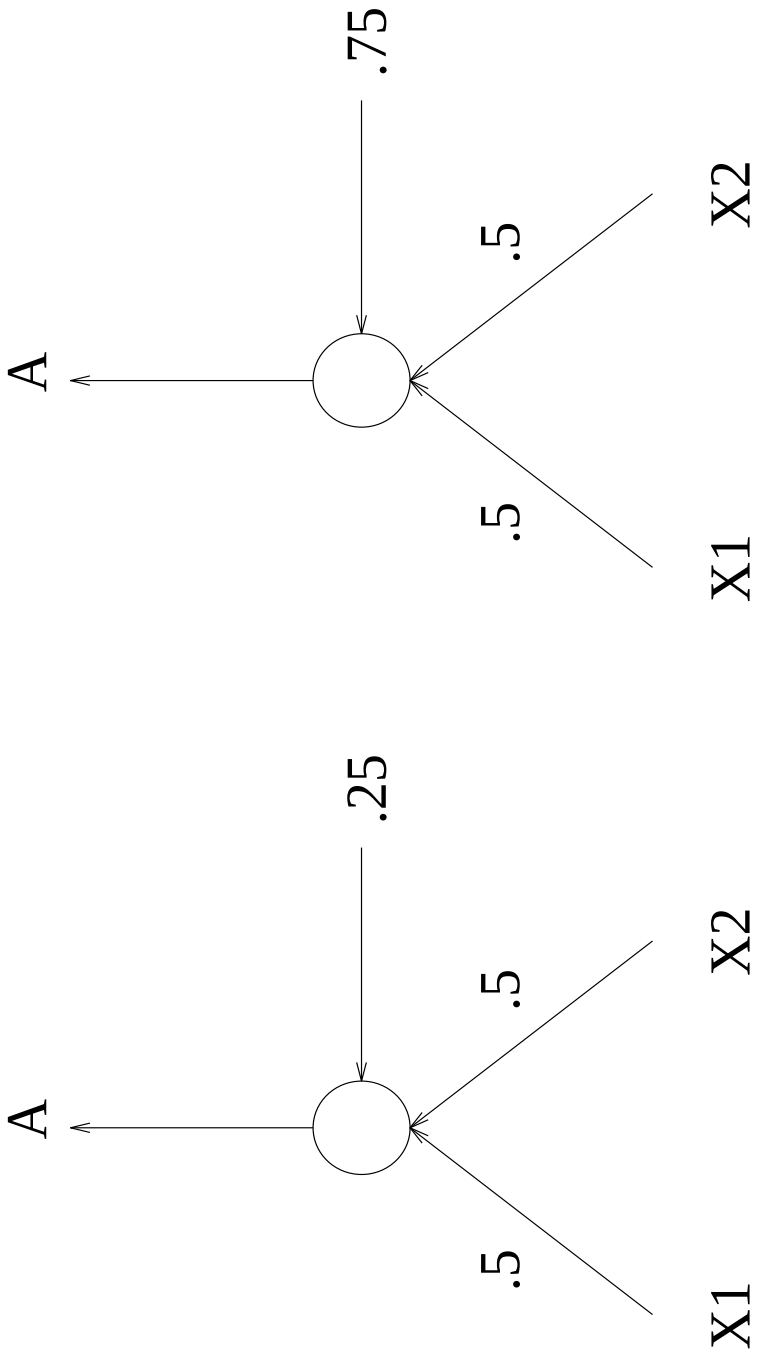


# Perceptron Output Function




$$A = \begin{cases} 1 \text{ If } W_1X_1 + \ldots + W_nX_n > T \\ 0 \text{ Otherwise.} \end{cases}$$

# Examples



# Thresholds as Weights

Notice that

$$W_1X_1 + \ldots + W_nX_n > T$$

is equivalent to

$$W_1X_1 + \ldots + W_nX_n - T > 0$$

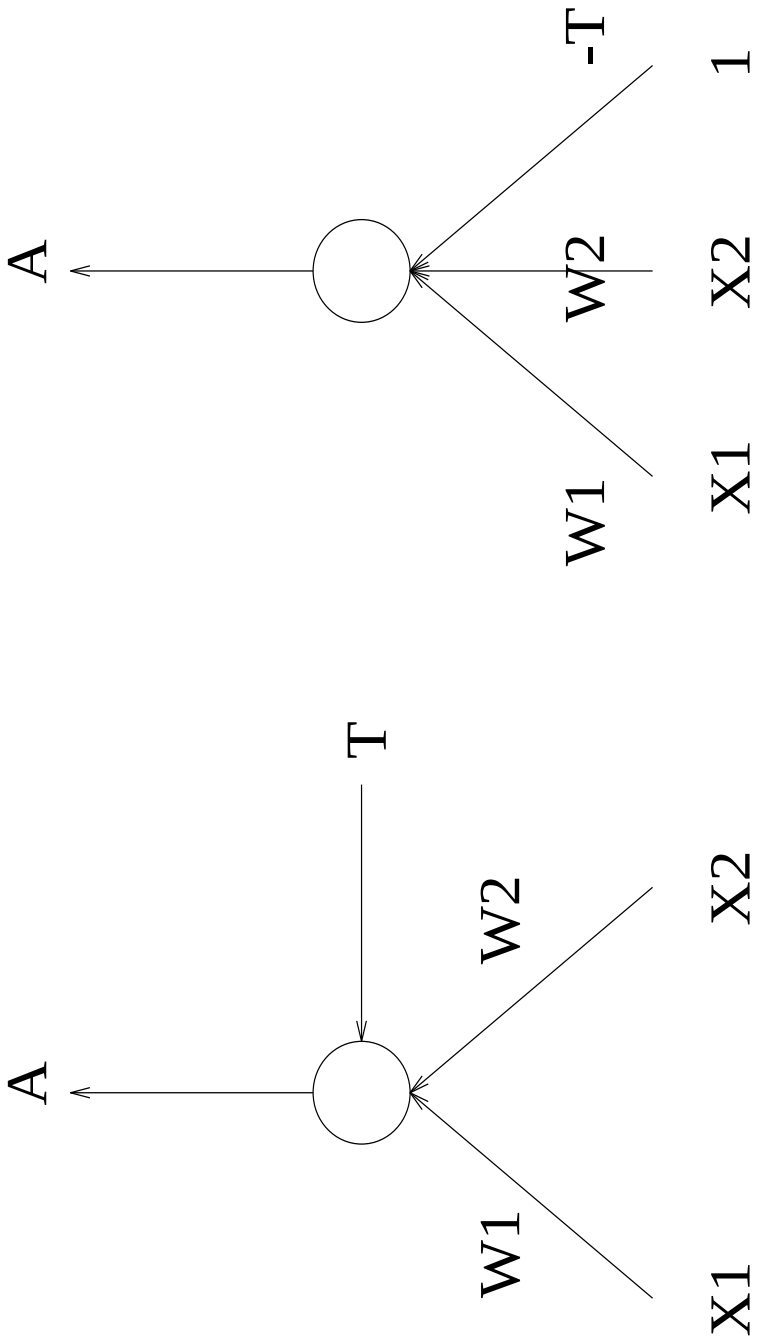
or

$$W_1X_1 + \ldots + W_nX_n - T \cdot 1 > 0$$

# Thresholds as Weights

- Pretend each node has an extra input whose value is always 1 and whose weight is $-T$, called the "bias".
- Learning updates the bias just like all the other weights.

# Thresholds as Weights



# Learning in Neural Networks

- Learn values of weights from I/O pairs.
- Start with random weights.
- Load training example's input.
- Observe computed output.
- Compare to desired output.
- Modify weights to reduce difference.
- Iterate over all training examples.
- Terminate when weights stop changing.

# Perceptron Learning Rule

$$\Delta W_i = \eta(D - A)X_i$$

$X_i$ is a node's input.

$W_i$ is the corresponding weight.

$\Delta W_i$ is the change in weight.

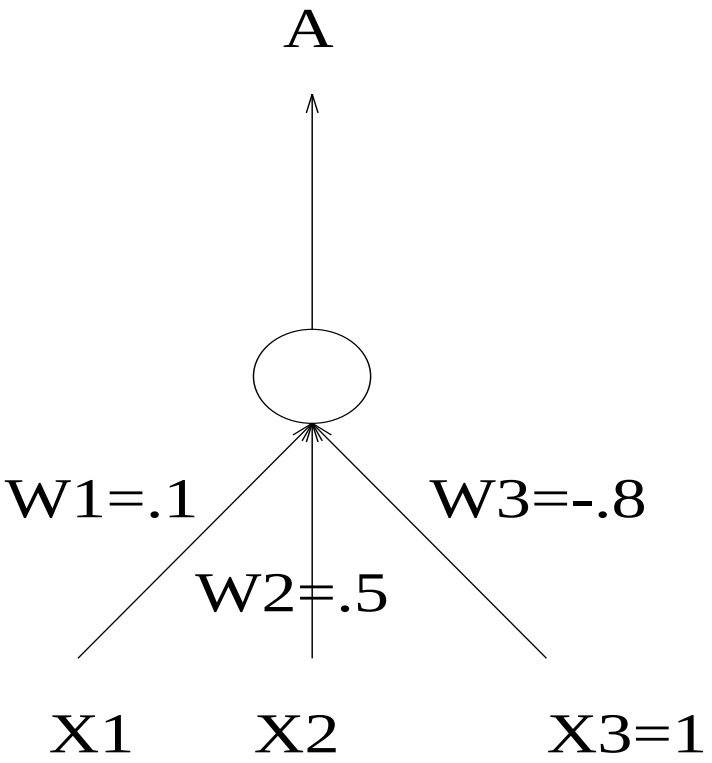
$D$ is the desired output.

$A$ is the actual observed output.

$\eta$ is the learning rate.

# Learning the *Or* Function

$$\Delta W_i = .2(D - A)X_i$$



# Learning the *Or* Function

| $X_1$ | $X_2$ | $D$ | $A$ | $\Delta W_1$ | $W_1$ | $\Delta W_2$ | $W_2$ | $\Delta W_3$ | $W_3$ |
|---|---|---|---|---|---|---|---|---|---|
| | | | | | .1 | | .5 | | -.8 |
| 0 | 0 | 0 | 0 | 0 | .1 | 0 | .5 | 0 | -.8 |
| 0 | 1 | 1 | 0 | 0 | .1 | .2 | .7 | .2 | -.6 |
| 1 | 0 | 1 | 0 | .2 | .3 | 0 | .7 | .2 | -.4 |
| 1 | 1 | 1 | 1 | 0 | .3 | 0 | .7 | 0 | -.4 |
| 0 | 0 | 0 | 0 | 0 | .3 | 0 | .7 | 0 | -.4 |
| 0 | 1 | 1 | 1 | 0 | .3 | 0 | .7 | 0 | -.4 |
| 1 | 0 | 1 | 0 | .2 | .5 | 0 | .7 | .2 | -.2 |
| 1 | 1 | 1 | 1 | 0 | .5 | 0 | .7 | 0 | -.2 |
| 0 | 0 | 0 | 0 | 0 | .5 | 0 | .7 | 0 | -.2 |
| 0 | 1 | 1 | 1 | 0 | .5 | 0 | .7 | 0 | -.2 |
| 1 | 0 | 1 | 1 | 0 | .5 | 0 | .7 | 0 | -.2 |

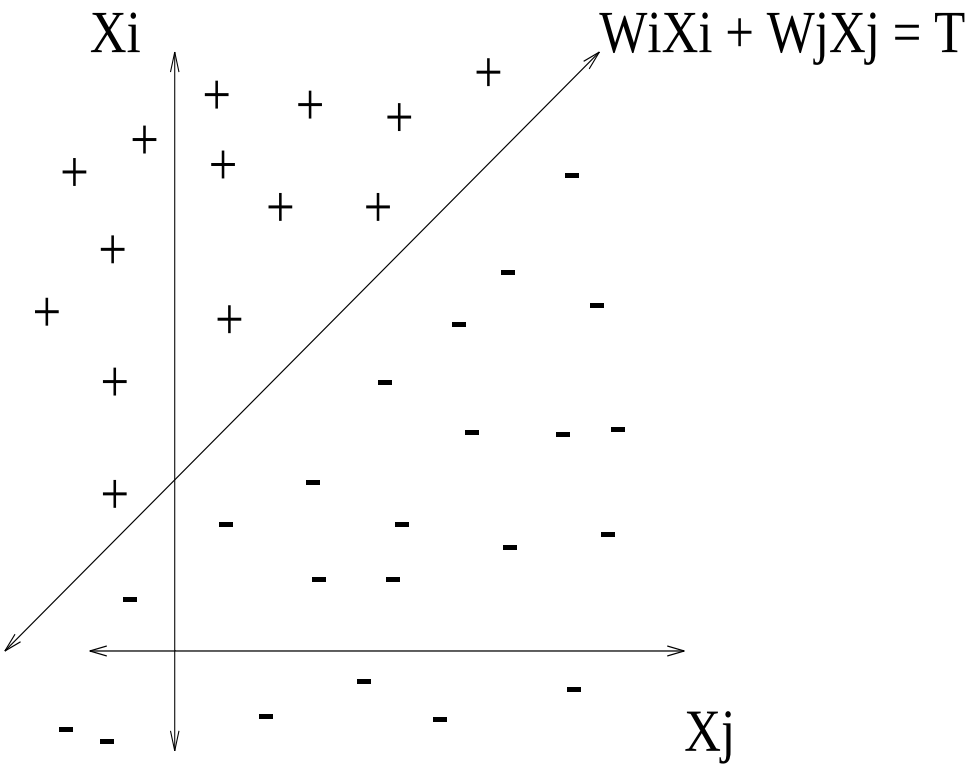
# Perceptron Convergence Theorem

"If a set of I/O pairs is learnable, then the Perceptron Learning Rule will find the necessary weights."

Minsky and Papert, 1988.

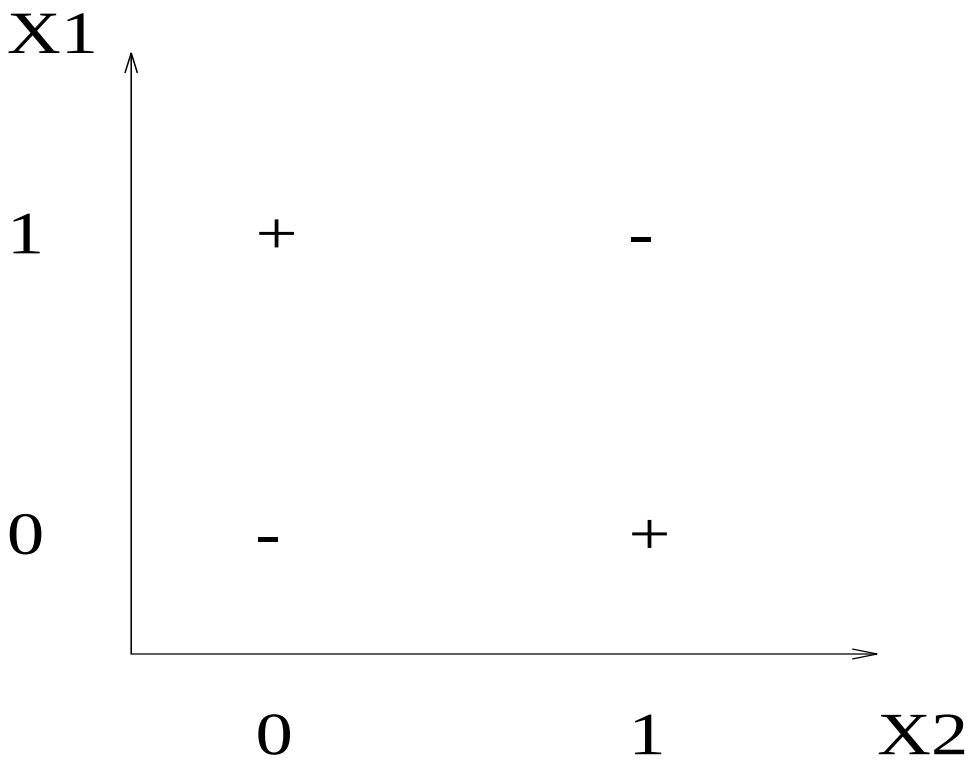
# Linear Separability

The output function $W_1X_1 + \ldots + W_nX_n > T$ defines a hyperplane that splits the input space into two half spaces.

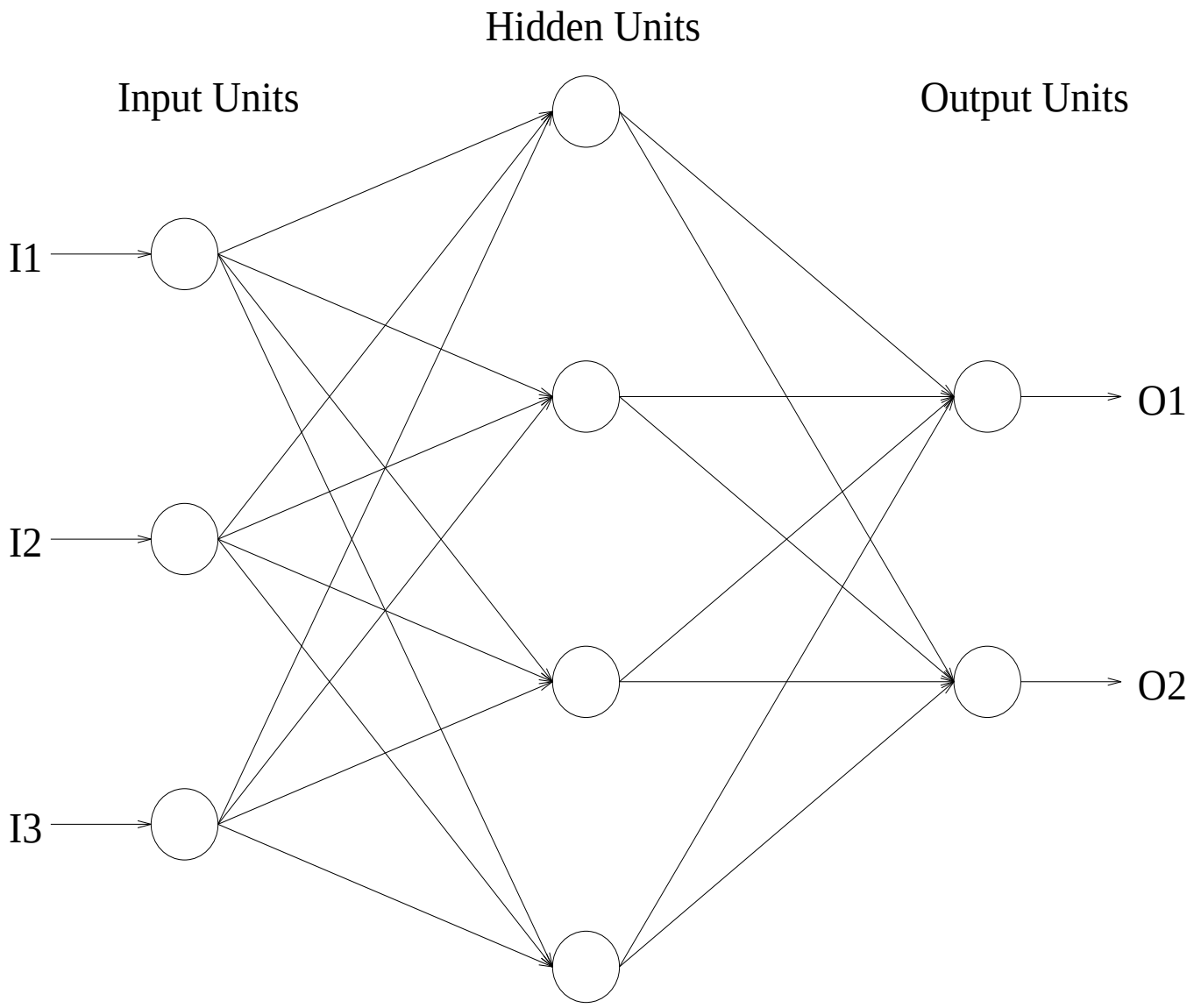


# Linear Separability: The *XOR* Function

Can a single line separate the two classes?

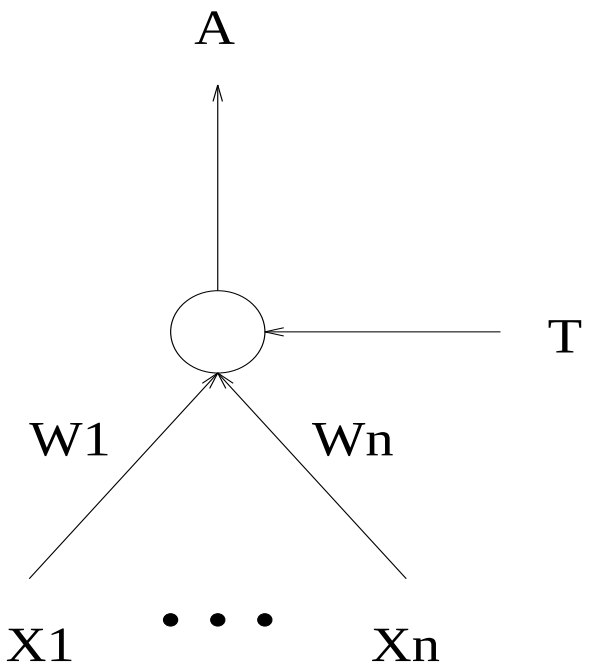


# Solution: Hidden Units



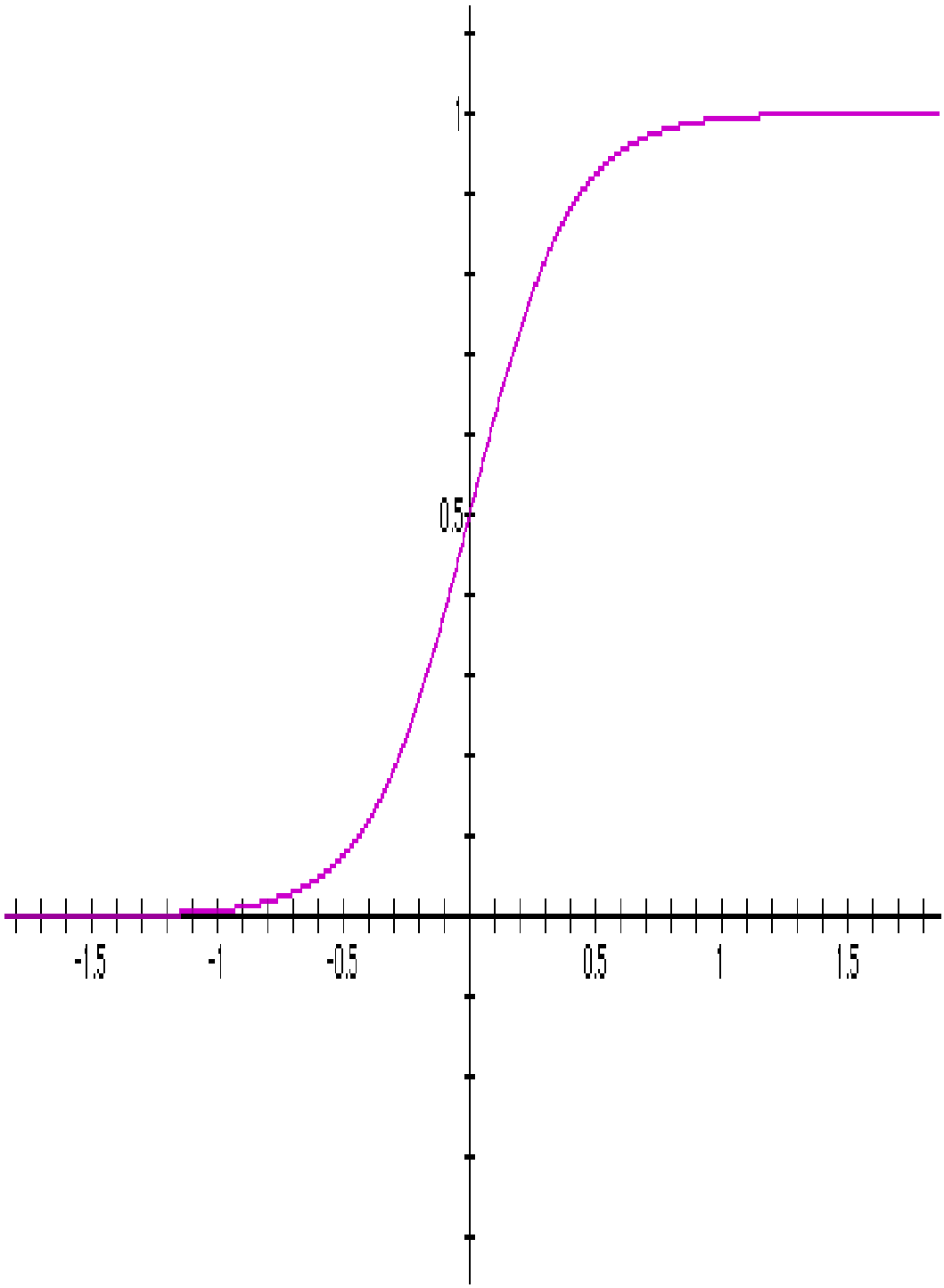
# Backpropagation (Backprop) Networks

- Network contains hidden units.
- Each unit obeys "Sigmoidal Activation Function".

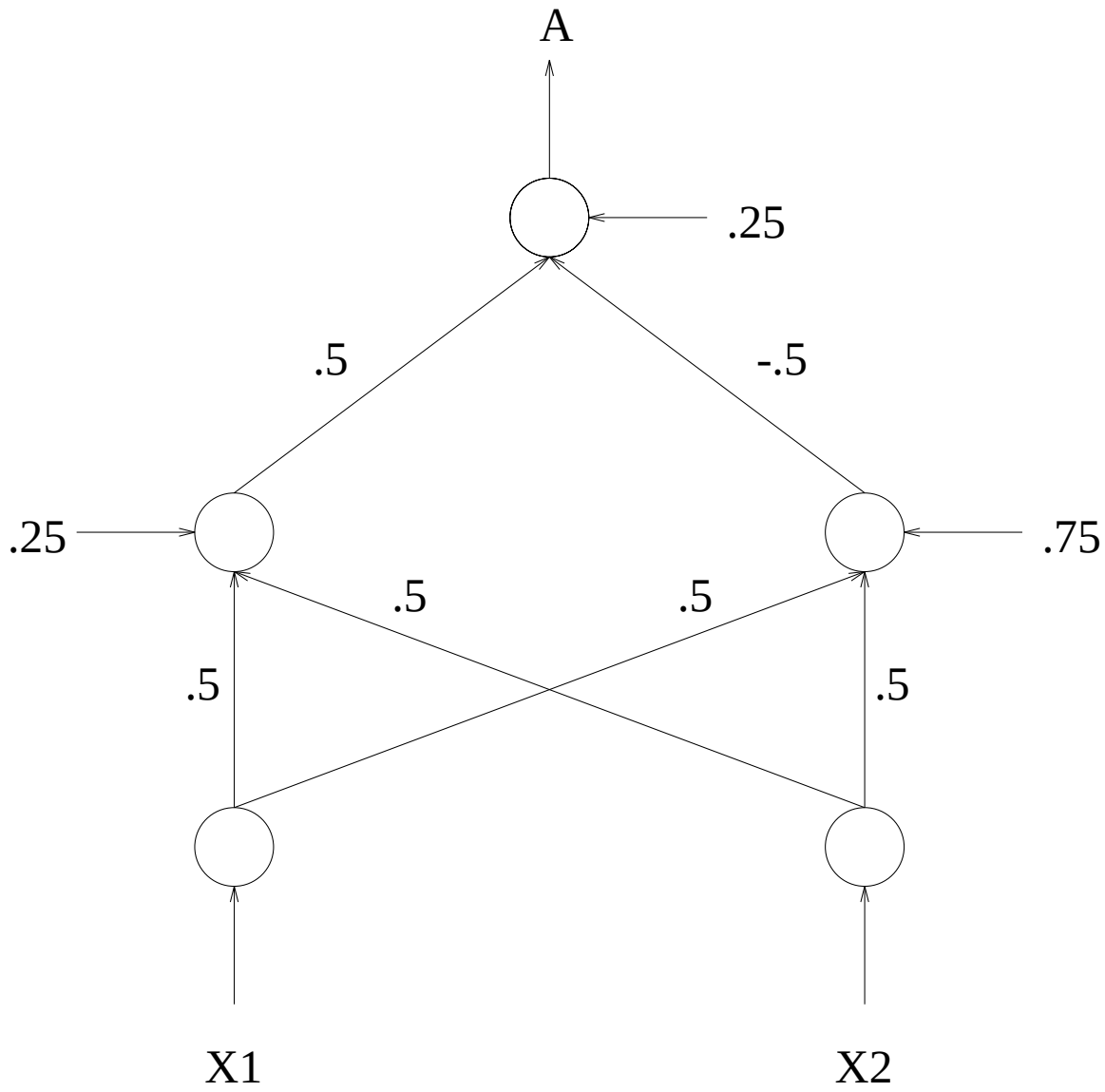



$$A = \frac{1}{1+e^{-(W_1X_1+...+W_nX_n-T)}}$$

# Behavior of Sigmoid Function



# Implementation of the $XOR$ Function



# Backpropagation Network Learning Rule for Weights of Edges from Hidden Units to Output Units

$$\Delta W_i = \eta A(1 - A)(D - A)X_i$$

$X_i$ is a node's input.

$W_i$ is the corresponding weight.

$\Delta W_i$ is the change in weight.

$D$ is the desired output.

$A$ is the actual observed output.

$\eta$ is the learning rate.

# Motivation for the Backpropagation Learning Rules

- The weights $\bar{W} = (W_1, \ldots, W_n)$ define a point in an $n$-dimensional Euclidean space.
- Each point $\bar{W}$ in the space defines a network.
- Each point $\bar{W}$ has an associated error rate:

$$E = \tfrac{1}{2}\Sigma_i (D_i - A_i)^2$$

- We compute the gradient $\nabla E$ with respect to $\bar{W}$.
- Each learning iteration changes $\bar{W}$ to $\bar{W} - \eta \nabla E$.
- A "Gradient Descent" algorithm.

# Applications of Backpropagation Networks

- NETtalk converts character strings to phonemes.
- Neurogammon won the 1989 Computer Olympiad.
- ALVINN steers a vehicle along a single lane highway.

# Advantages and disadvantages of Neural Networks

**Advantages:**

- Generalization capability.
- Low sensitivity to noise.

**Disadvantages:**

- Relative expressiveness.
- Computational efficiency.
- Transparency (black box).
- Hard to use prior knowledge.