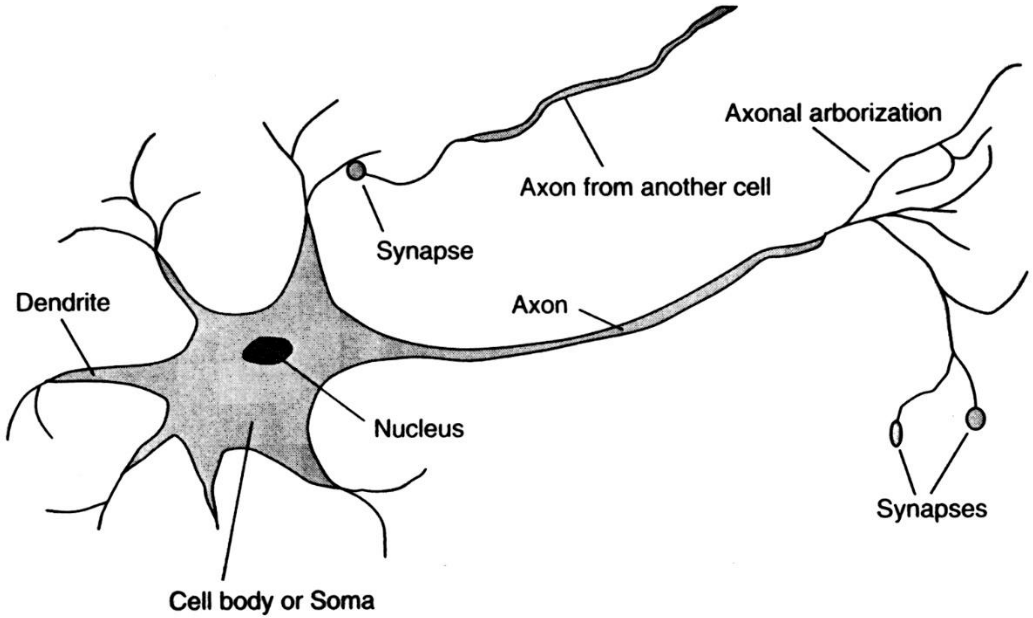
# Artificial Intelligence

Intro to Neural Networks

# How the Brain Works (sort of)




- Neuron is fundamental functional unit
  - Soma: cell body
  - Axon: long single fiber that connects to other neurons
  - Dendrites: connected to axons from other neurons
  - Synapse: connecting junction

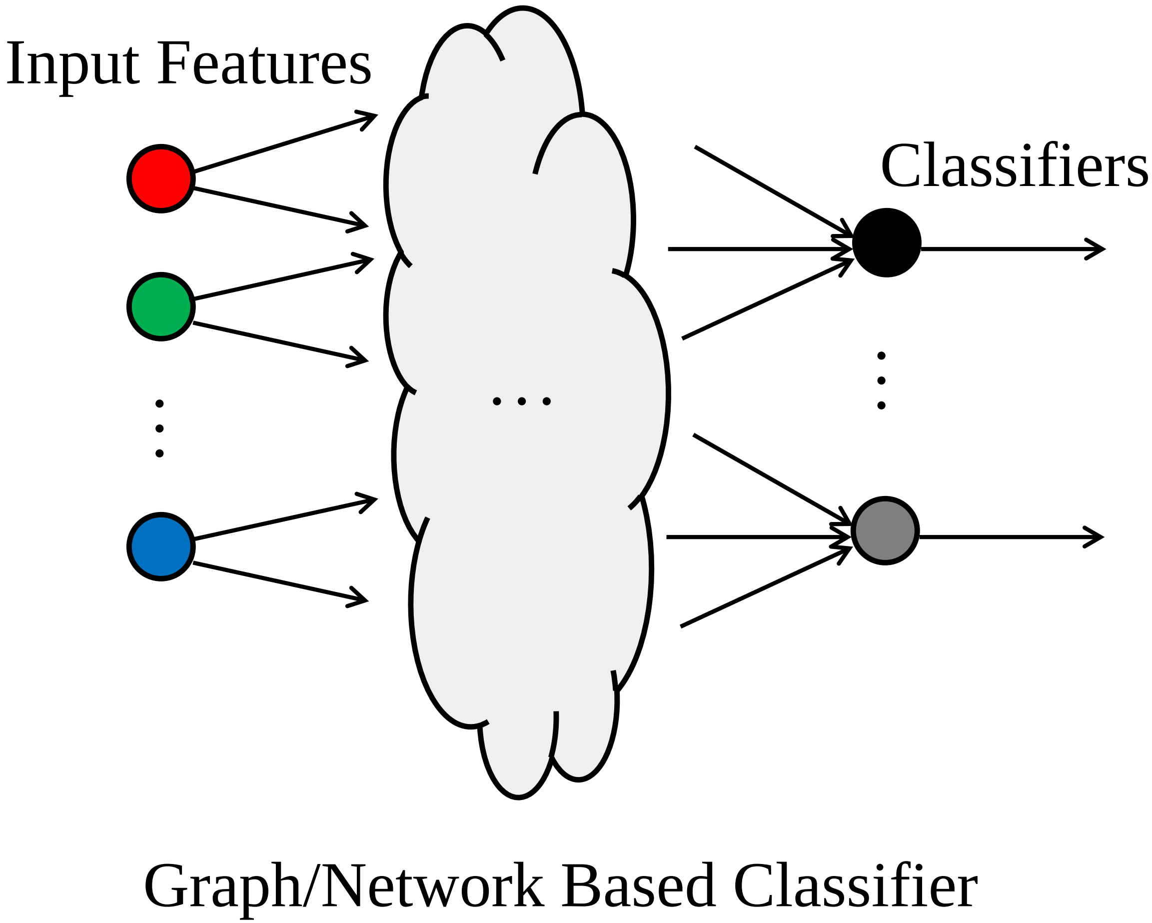
# How the Brain Works

- Signals propagated between neurons by electrochemical reaction
  - Chemical substances released from synapses and enter dendrite, raising or lowering electric potential of cell body
    - Synapses that increase potential are <u>excitatory</u>
    - Synapses that decrease potential are <u>inhibitory</u>
  - <u>Action potential</u> (electrical pulse) sent down axon when electric potential of cell body reaches a threshold

# How the Brain Works

- A collection of simple cells can lead to thought, action, and consciousness
  - Bottom-up statement
  - Long way from a theory of consciousness
  - "Brains cause minds" (Searle 1992)

# Neural Networks



# Neural Networks

- Neural net is composed of nodes (units)
  - Some connected to outside world as input or output units
- Nodes are connected by links
  - Input and output links
- Each link has numeric weight associated with it
  - Primary means of **long-term storage/memory**
  - Weights are modified to bring network's input/output behavior to goal response
- Nodes have activation level
  - Given its inputs and weights
  - Local computation based on inputs from neighbors (no global control)

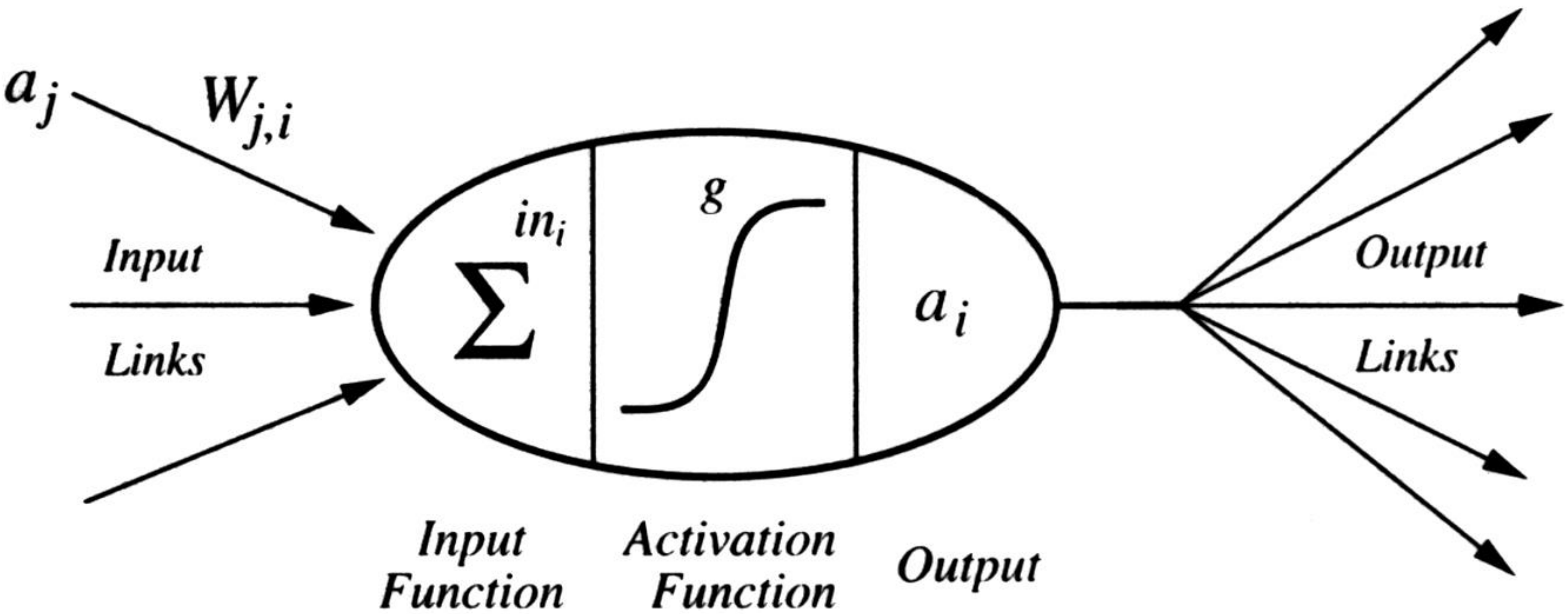
# Using Neural Networks

- One must first decide
  - How many nodes to use
  - What kind of nodes are appropriate
  - How nodes are to be connected into a network
- Weights are randomly initialized, then training learns correct weight values given a particular set of training examples
  - Input examples are labeled with correct outputs
  - One must decide how to encode examples in terms of network inputs and outputs

# Simple Computing Elements

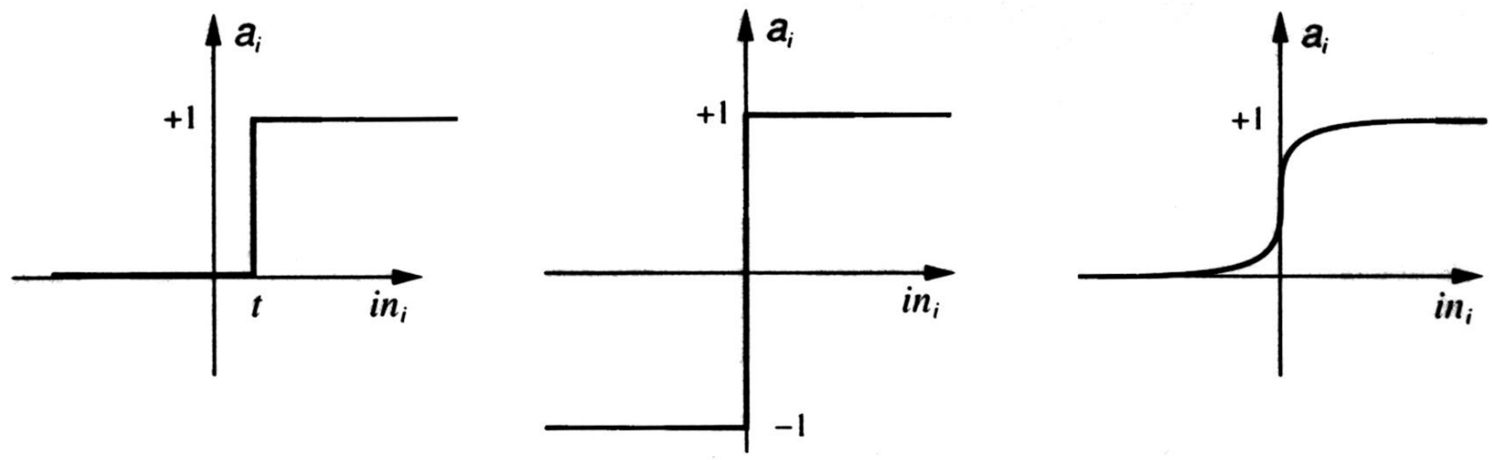
- Each unit performs simple computation
  - Receives signals from input links
  - Computes new activation level
  - Sends activation level along each output link
- Computation split into two components
  - Linear input function
    - Computes weighted sum of inputs
  - Nonlinear activation function
    - Transforms weighted sum into final activation value

# Simple Computing Elements




$$\sum_j W_{j,i} a_j = \mathbf{W} \cdot \mathbf{a} \qquad\qquad g\left(\sum_j W_{j,i} a_j\right)$$

# Types of Activation Functions



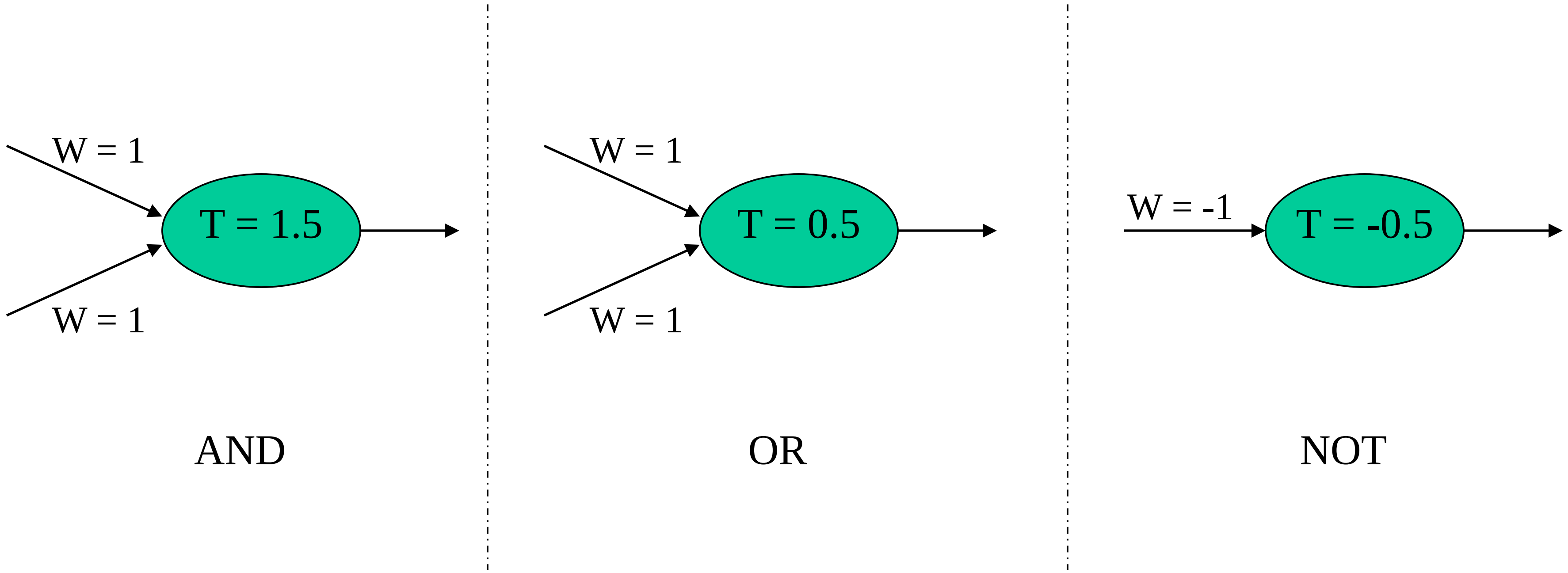

(a) Step function (b) Sign function (c) Sigmoid function

Three different activation functions for units.

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

Non-linear functions!

# Boolean Functions Using Step Activation Function

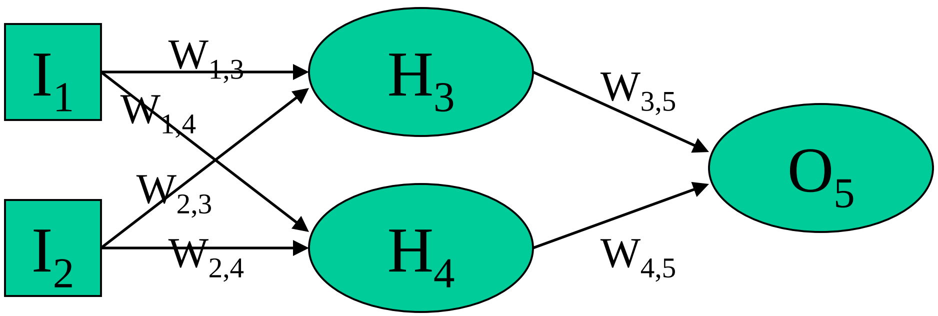


# Network Structures

- Two main varieties
  - Feed-forward
    - Unidirectional links with no cycles
    - Directed acyclic graph (DAG)
  - Recurrent
    - Links can form arbitrary topologies
    - Contains cycles

# Network Structures

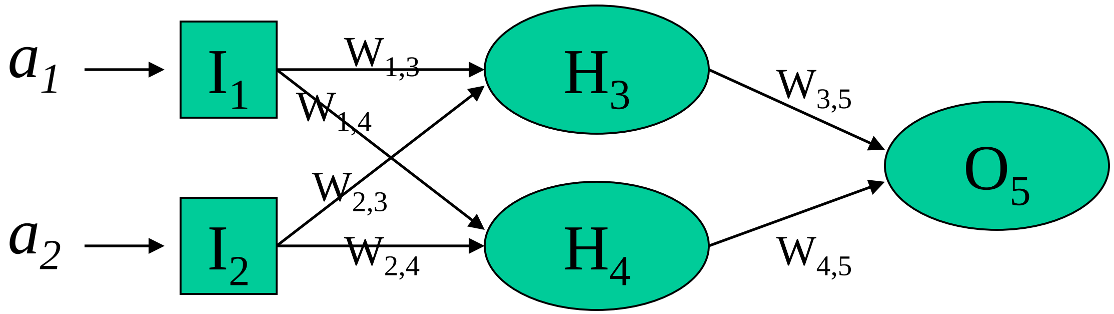
- Feed-forward networks have no internal state (other than weight values)
  - Simply computes function of input values using weights




Two-layer, feed-forward network

- Not like the brain!
  - We have memory
  - Many back connections

# Network Structures




Two-layer, feed-forward network

Network calculates function ($g$ is nonlinear activation):

$$O_5 = g\left(W_{3,5} \cdot \underbrace{g\left(W_{1,3}\ a_1 + W_{2,3}\ a_2\right)}_{\text{output of } H_3} + W_{4,5} \cdot \underbrace{g\left(W_{1,4}\ a_1 + W_{2,4}\ a_2\right)}_{\text{output of } H_4}\right)$$

**Learning just becomes a process of tuning parameters to fit data in training set!!!**

# Linear Activation Functions?

- Consider what happens to the previous example if we use a linear activation function
  - Let's pick an easy one: $g(x) = x$
- On our two hidden nodes, the output is then just the input

$$O_5 = g\left(W_{3,5} \cdot g\left(W_{1,3}\ a_1 + W_{2,3}\ a_2\right) + W_{4,5} \cdot g\left(W_{1,4}\ a_1 + W_{2,4}\ a_2\right)\right)$$

$$= g\left(W_{3,5} \cdot \left(W_{1,3}\ a_1 + W_{2,3}\ a_2\right) + W_{4,5} \cdot \left(W_{1,4}\ a_1 + W_{2,4}\ a_2\right)\right)$$

$$= g\left(\left(W_{3,5}\ W_{1,3} + W_{4,5}\ W_{1,4}\right)a_1 + \left(W_{3,5}\ W_{2,3} + W_{4,5}\ W_{2,4}\right)a_2\right)$$

But these boil down to a single coefficient, i.e. parameter

$$= g(W_1\ a_1 + W_2\ a_2)$$

- So equivalent to a single node
  - This is generally true: Any linear activation function will cause a network to always have a single-layer equivalent
  - In other words, non-linear activation functions are required to make linked sequences of nodes useful

# Network Structures

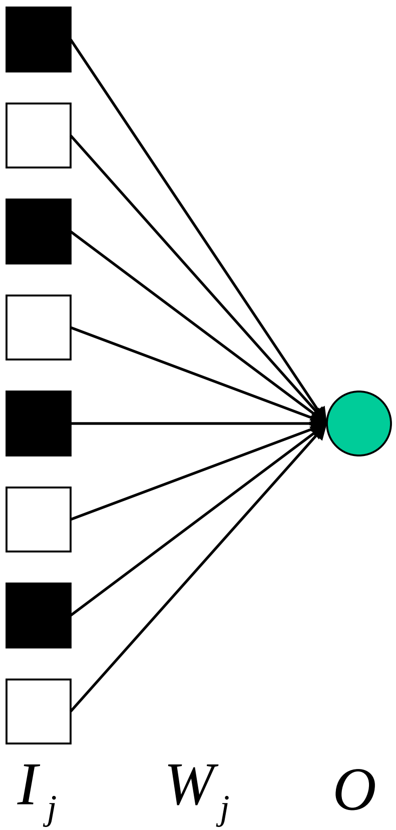
- Input units
  - Value of each unit determined by environment
- Output units
- Hidden units
  - Internal units that are neither input or output units
  - (Perceptrons have no hidden units)
- Multilayer networks
  - Networks with one or more layers of hidden units
  - One hidden layer
    - Theoretically can represent <u>any continuous function</u> of the inputs
  - Two hidden layers
    - Theoretically can represent even <u>discontinuous functions</u>

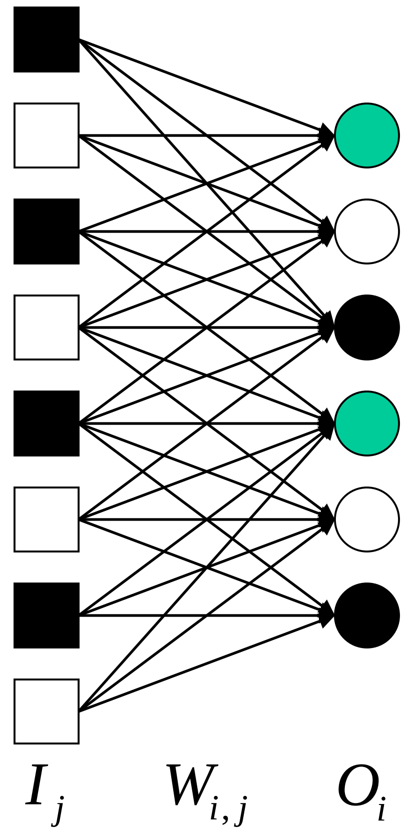
# Optimal Network Structure

- Neural networks are subject to overfitting
  - When use too many parameters (weights) in model
  - Cross validation techniques are useful for determining right size of network

# Perceptrons

- First studied in late 1950's
- Single-layer, feed-forward network



# Perceptrons

- **Step activation** of output unit for (single) perceptron ($I_0$ = -1, $W_0$ = threshold)

$$O = \text{Step}_0\left(\sum_j W_j\, I_j\right) = \text{Step}_0(\boldsymbol{W} \cdot \boldsymbol{I})$$

- Perceptrons represent functions that are <u>linearly separable</u> ***

# Dividing the Space (+,-)

- Consider, 2 inputs: $x$, $y$

$$
\begin{aligned}
O &= \mathrm{Step}_0(\boldsymbol{W}\cdot\boldsymbol{I}) = \mathrm{Step}_0\left(\begin{bmatrix}1 & -m & b\end{bmatrix}\cdot\begin{bmatrix}y\\ x\\ -1\end{bmatrix}\right)\\
&= \mathrm{Step}_0(y - mx - b)
\end{aligned}
$$

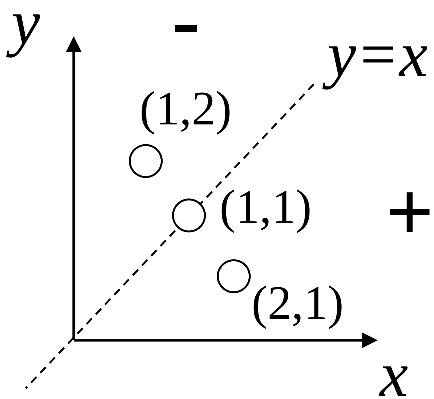
- The threshold of Step() is 0, so the important aspect: Is the input to step above or below 0?

?

$$
\begin{matrix} y - mx - b \geq 0 \\ y - mx - b < 0 \end{matrix} \longrightarrow y = mx + b
$$

Decision boundary

# Dividing the Space (+,-)




$$\boldsymbol{0 = x - y}$$

$$\boldsymbol{y = x}$$

(1,2):

**-1** = 1 - 2

(2,1):

**+1** = 2 - 1

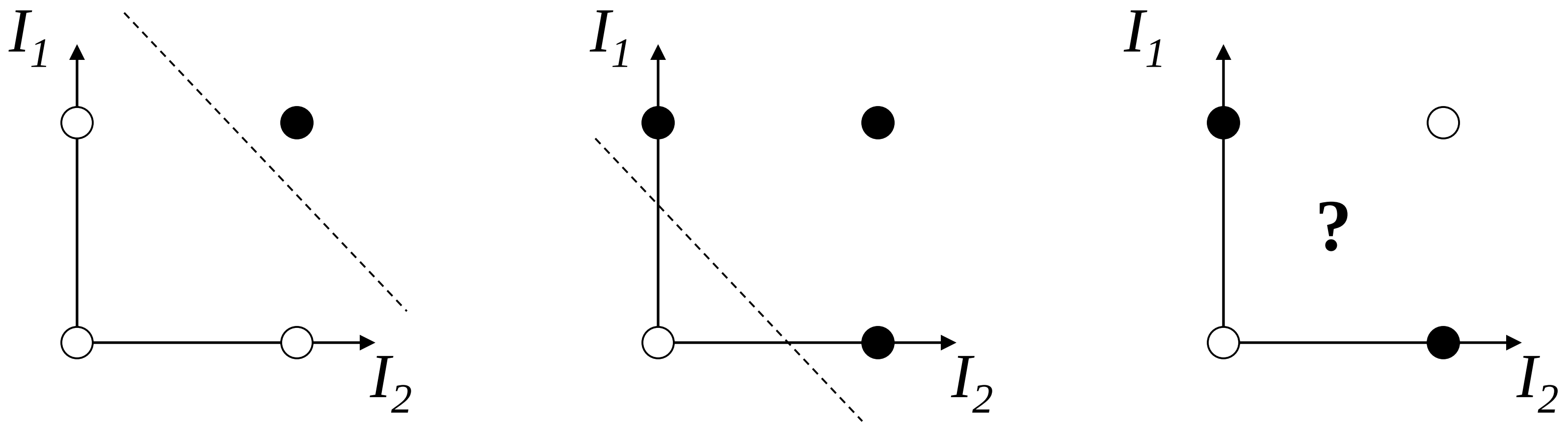
(1,1):

**0** = 2 - 2

# Linear Separability in Perceptrons

Limited in Boolean functions they can represent

**AND, OR, but not XOR**

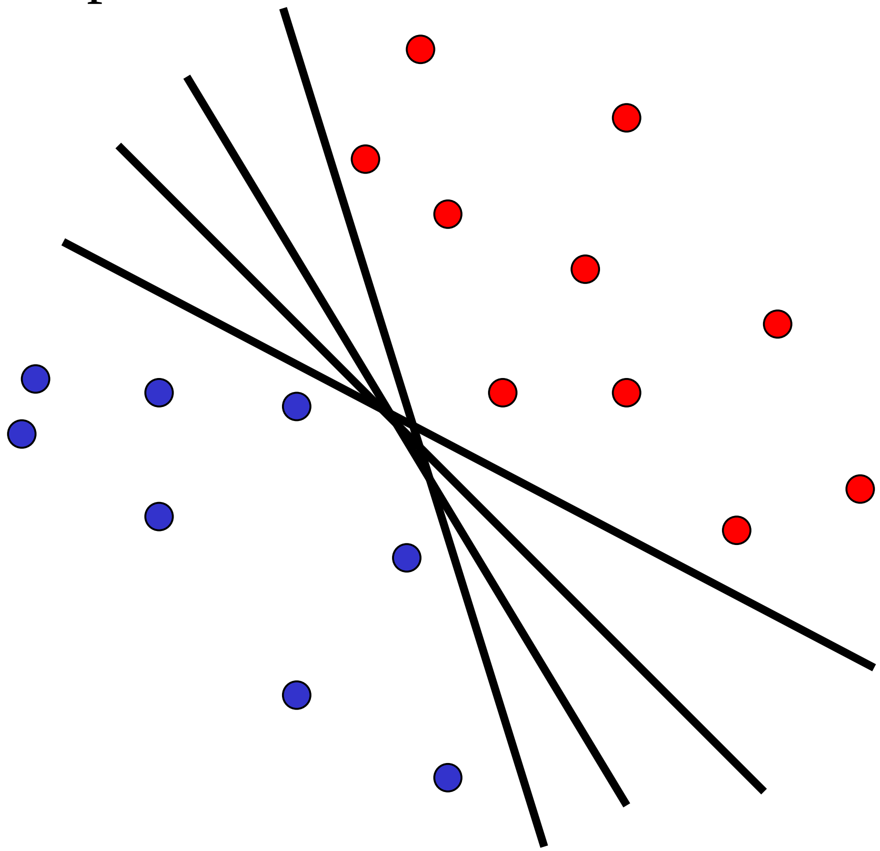



$I_1$ AND $I_2$ $I_1$ OR $I_2$ $I_1$ XOR $I_2$

"A perceptron can represent a function only if some <u>line</u> can separate all white dots from black dots"

# Linear Classifiers

- Multiple Perceptron solutions to separate positive and negative examples



# Perceptron Learning Algorithm

- Initially assign random weights [-0.5…0.5]
- Update network to try to make consistent with examples
  - Make small adjustments in weights to reduce difference between observed and predicted values
  - Updating process divided into "epochs"
    - Epoch involves updating all weights for all examples

# Weight Updating via Gradient Descent

Error function:

$$E = \frac{1}{2} Err^2 = \frac{1}{2}\left[ O - g\left( \sum_j W_j a_j \right) \right]^2$$

Desired output value

Perceptron output value

$$\begin{aligned}
\frac{\partial E}{\partial W_j} &= \frac{1}{2}\frac{\partial Err^2}{\partial W_j} = Err \cdot \frac{\partial Err}{\partial W_j} \\
&= Err \cdot \frac{\partial}{\partial W_j}\left[ O - g\left( \sum_j W_j a_j \right) \right] \\
&= Err \cdot g'() \cdot (-a_j)
\end{aligned}$$

$g' = sigmoid * (1 - sigmoid)$

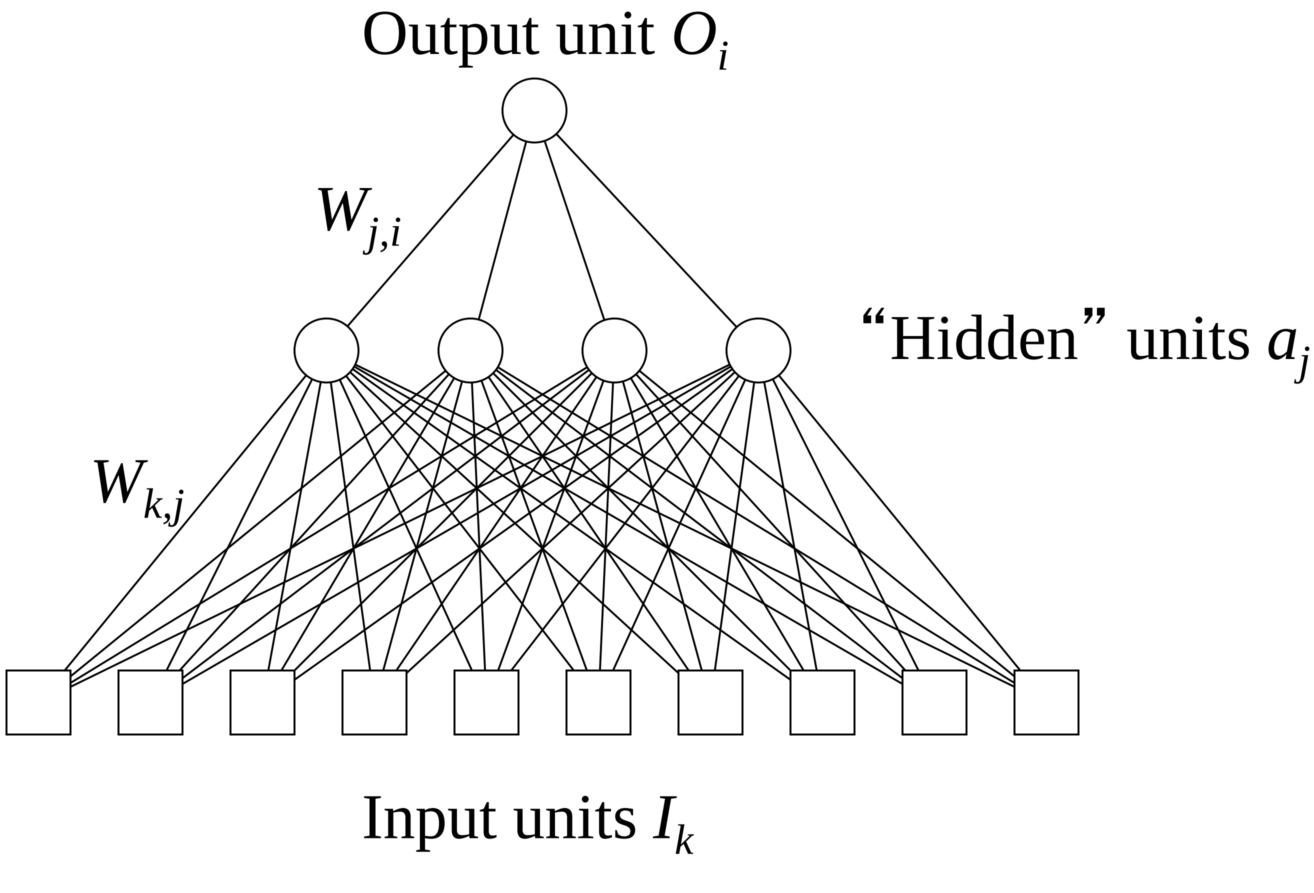
$$\begin{aligned}
W_j &= W_j - \alpha \cdot \frac{\partial E}{\partial W_j} \\
&= W_j + \alpha \cdot Err \cdot g'() \cdot a_j
\end{aligned}$$

Note that *g'()* is omitted from "threshold" perceptrons

# Perceptron Learning

- Perceptron convergence theorem is doing <u>gradient descent</u> through the weight space
- *Perceptrons*, by Minsky and Papert 1969
  - Clearly demonstrated the limits of linearly separable functions

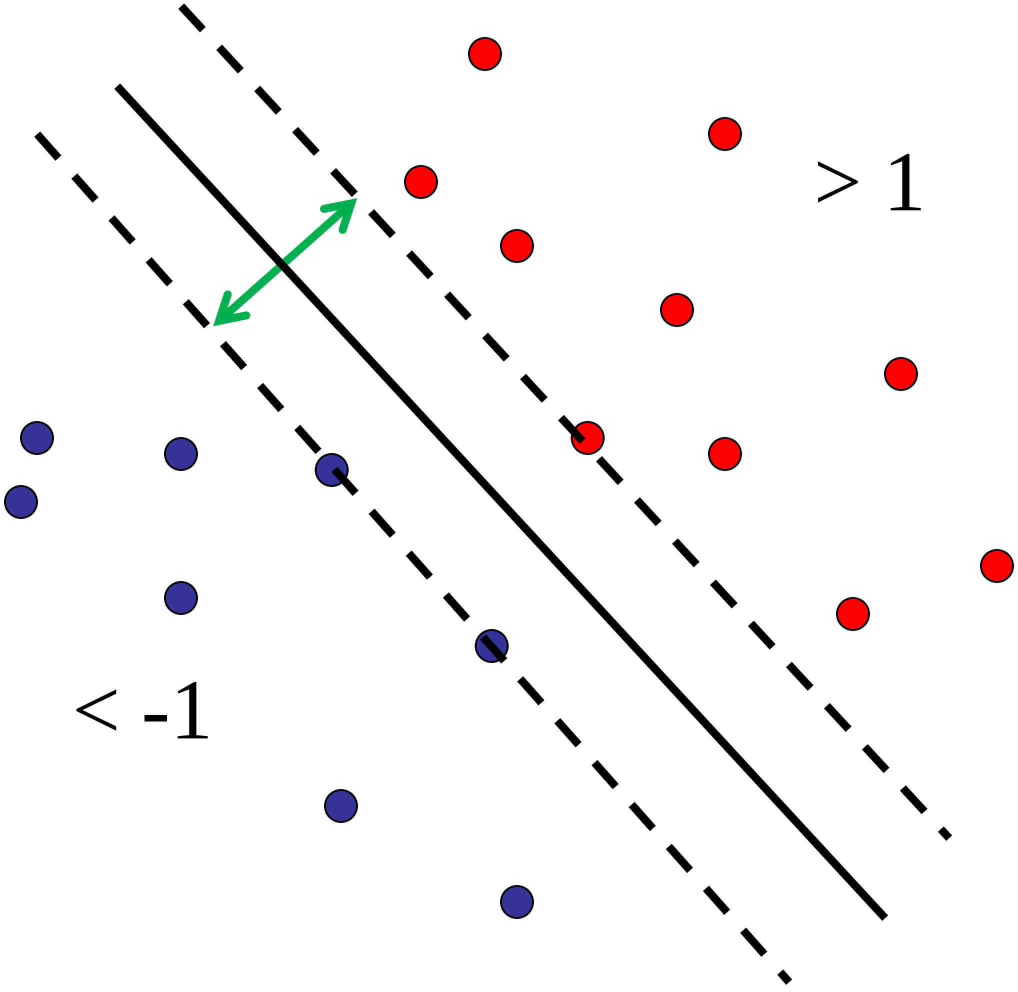
# Multilayer Feed-Forward Networks



# Learning in Multilayer Feed-Forward Networks

- Back-propagation learning algorithm
  - Assess blame for an error and divide it "locally" among contributing weights (divide contribution of each weight) and update layer by layer backwards

# Support Vector Machines (SVMs)

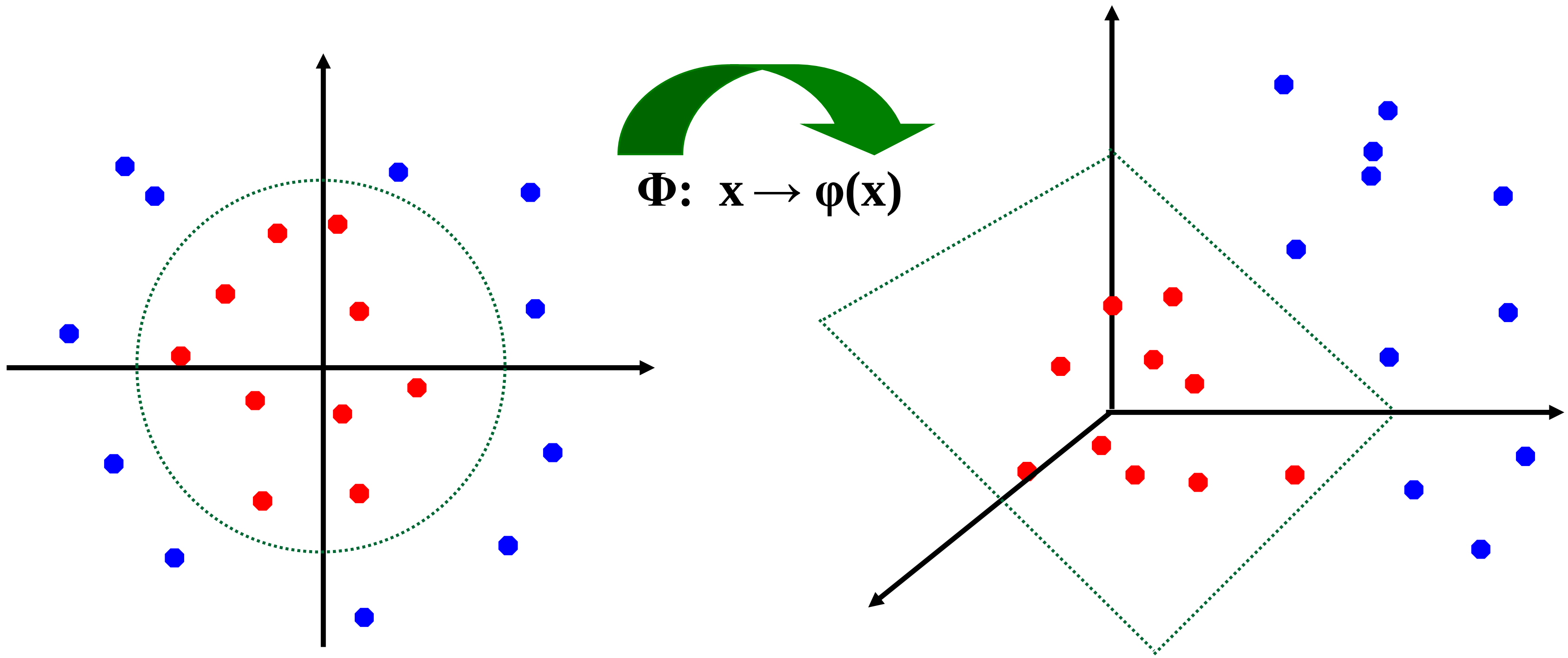



- Discriminative classifier based on *optimal separating hyperplane (i.e., line for 2D case)*

- Maximize the ***margin*** between the positive and negative training examples

# Non-Linear SVMs: Feature Spaces

- General idea: The original *input space* is mapped to some higher-dimensional *feature space* where the training set is more likely to be linearly separable:



# Summary

- Neural net
  - Nodes, links, weights, activation level
- Each unit performs simple computation
  - Receives signals from input links
  - Computes new activation level
  - Sends activation level along each output link
- Feed-forward network
  - Unidirectional links with no cycles
- Perceptrons
  - Single-layer, feed-forward network
  - Represent functions that are <u>linearly separable</u>
- Back-propagation for multilayer networks