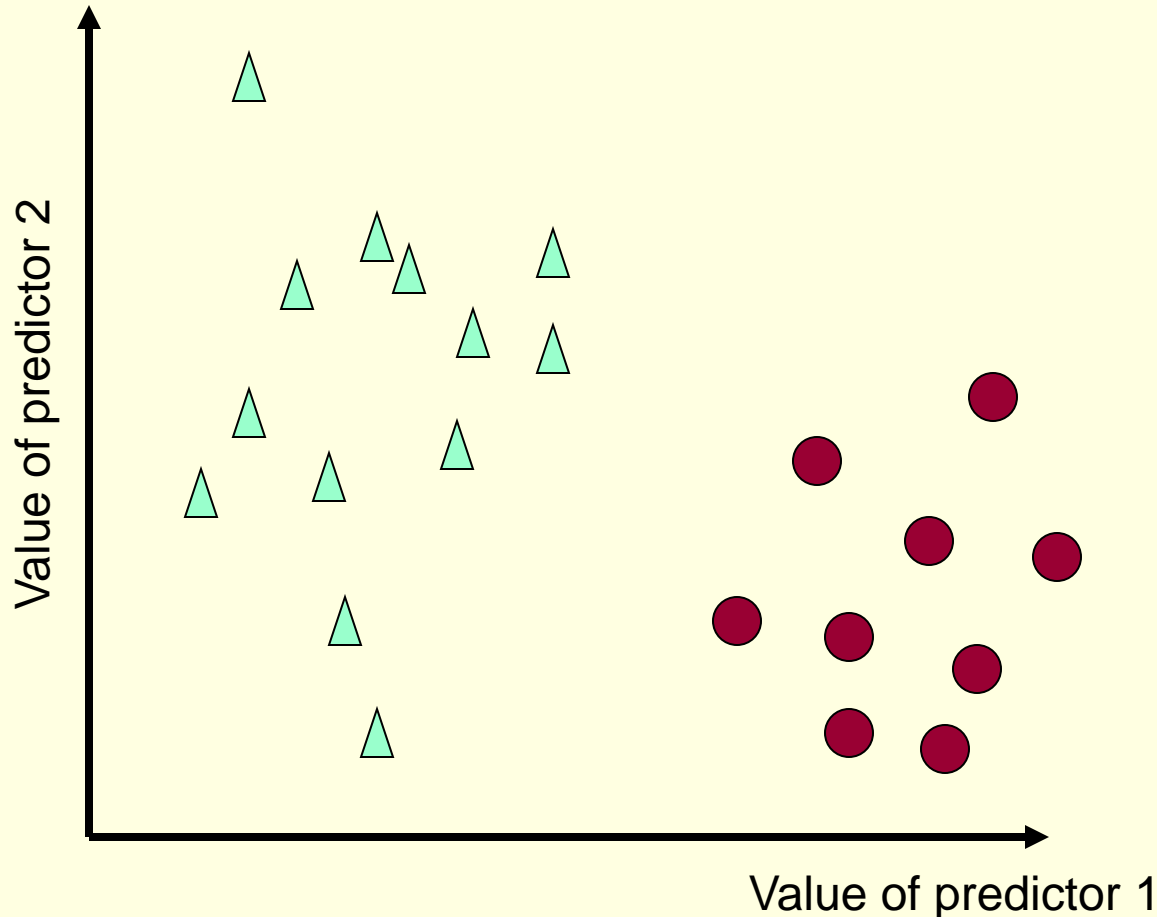




Feed-Forward Artificial Neural Networks

MEDINFO 2004,
T02: Machine Learning Methods for Decision Support and Discovery
Constantin F. Aliferis & Ioannis Tsamardinos
Discovery Systems Laboratory
Department of Biomedical Informatics
Vanderbilt University

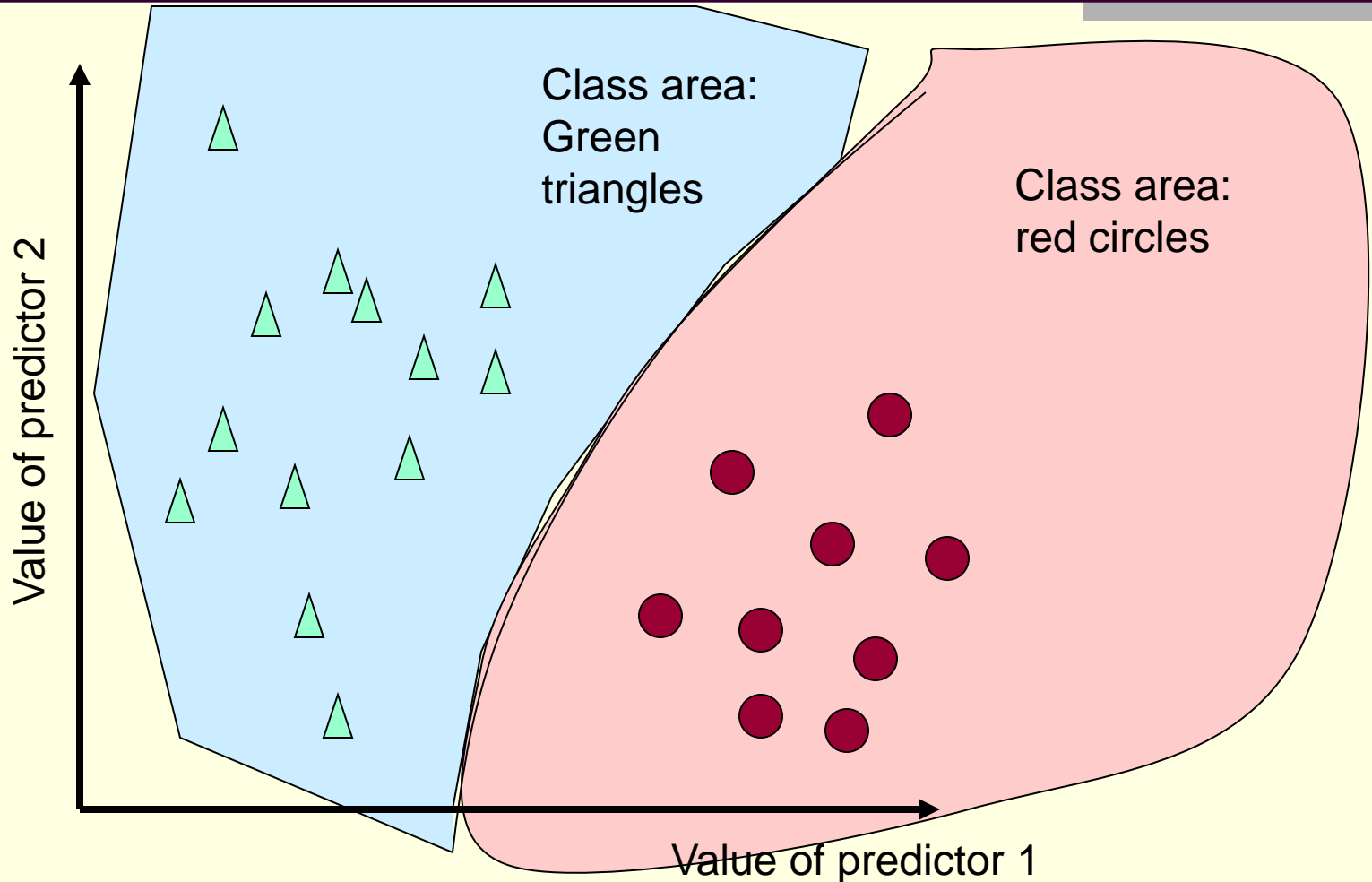
Binary Classification Example



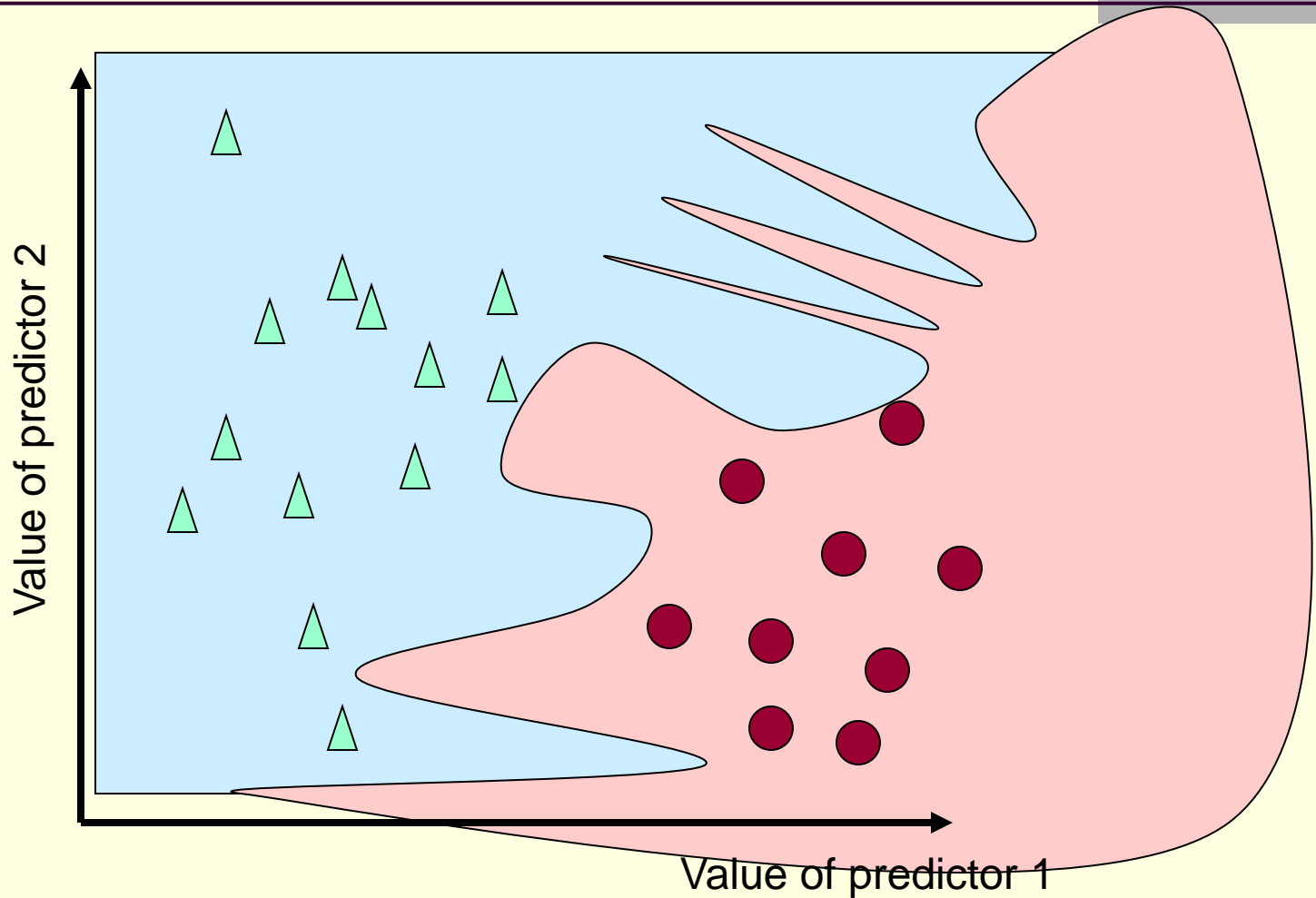
Example:

- Classification to malignant or benign breast cancer from mammograms
- Predictor 1: lump thickness
- Predictor 2: single epithelial cell size

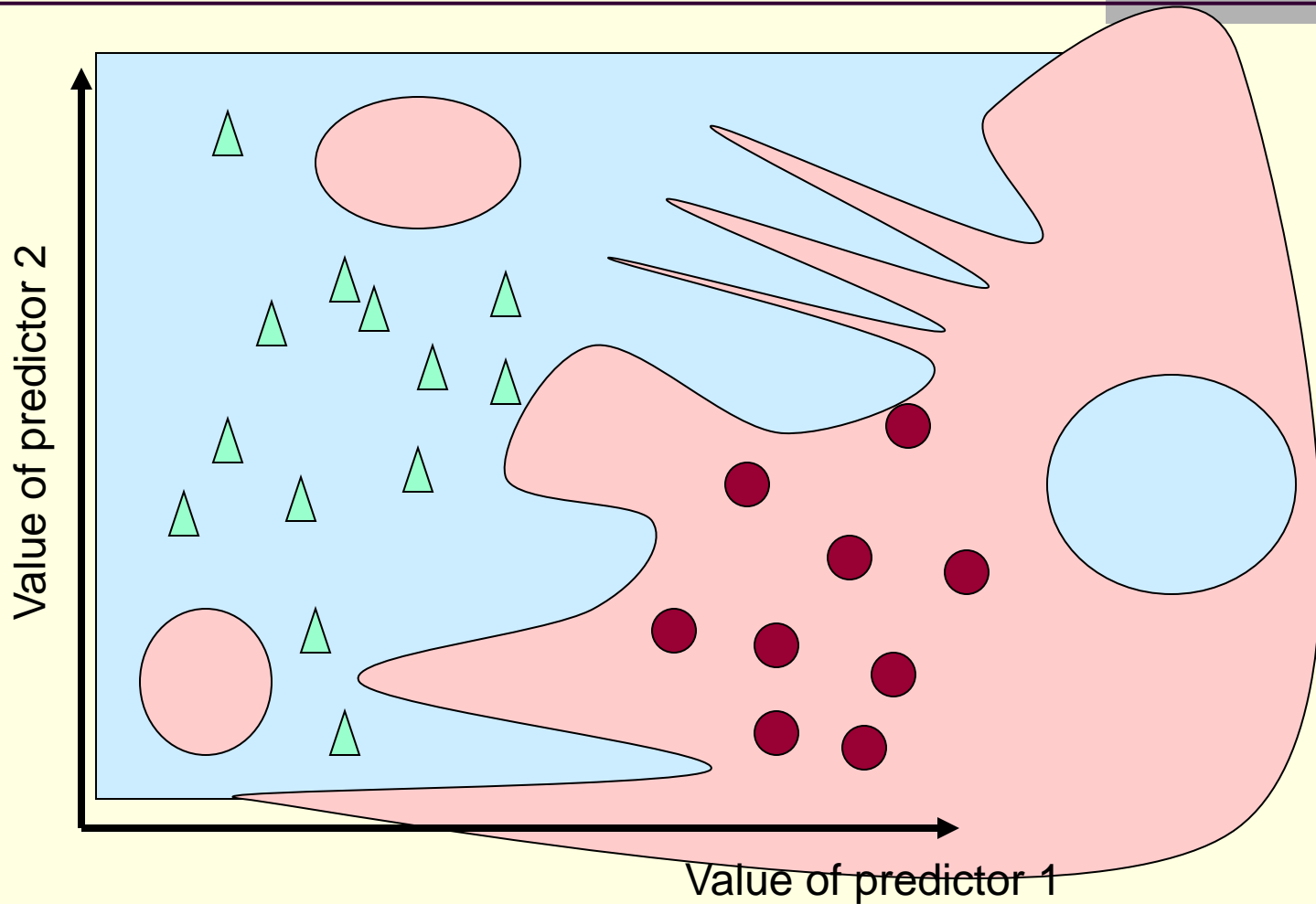
Possible Decision Area 1



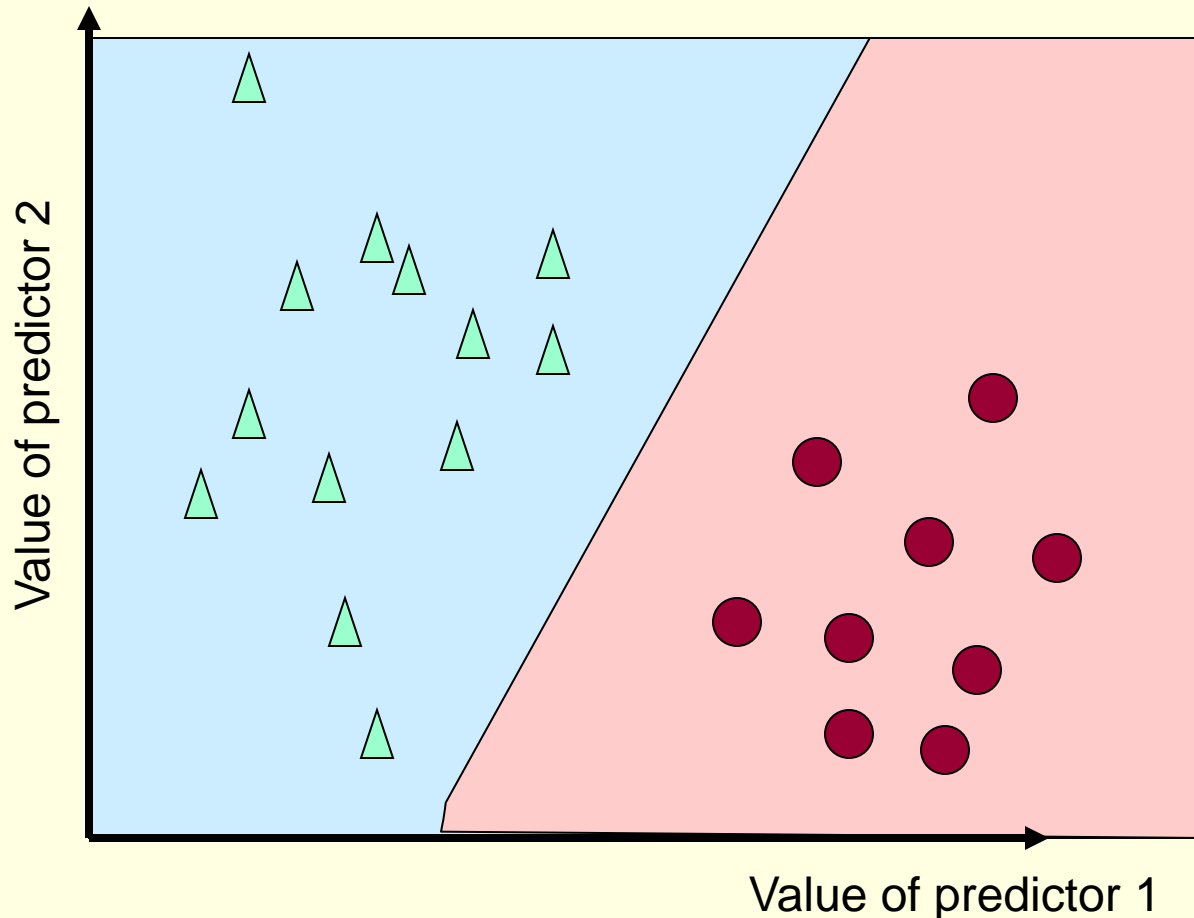
Possible Decision Area 2



Possible Decision Area 3



Binary Classification Example

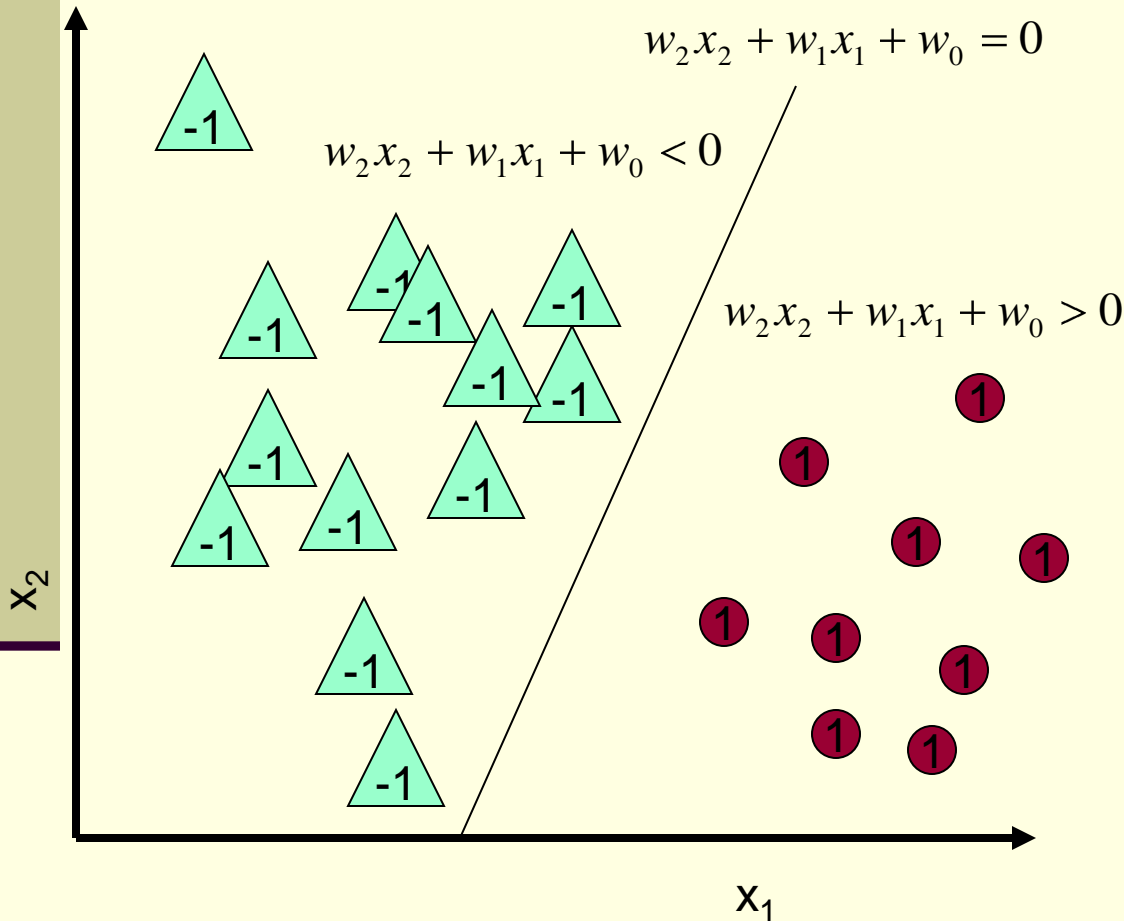


The simplest non-trivial decision function is the straight line (in general a hyperplane)

One decision surface

Decision surface partitions space into two subspaces

Specifying a Line



■ Line equation:

$$w_2x_2 + w_1x_1 + w_0 = 0$$

■ Classifier:

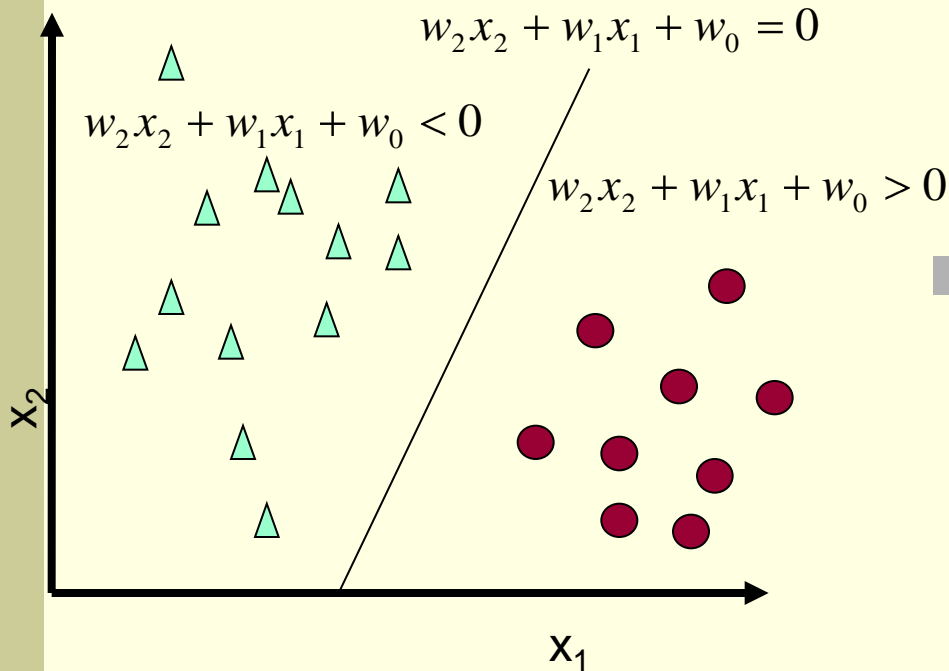
■ If $w_2x_2 + w_1x_1 + w_0 \geq 0$

■ Output 1

■ Else

■ Output -1

Classifying with Linear Surfaces



■ Classifier becomes

$\text{sgn}(w_2x_2 + w_1x_1 + w_0) =$
 $\text{sgn}(w_2x_2 + w_1x_1 + w_0x_0),$
set $x_0 = 1$ always

Let n be the number of predictors

$\text{sgn}\left(\sum_{i=0}^n w_i x_i\right),$ or

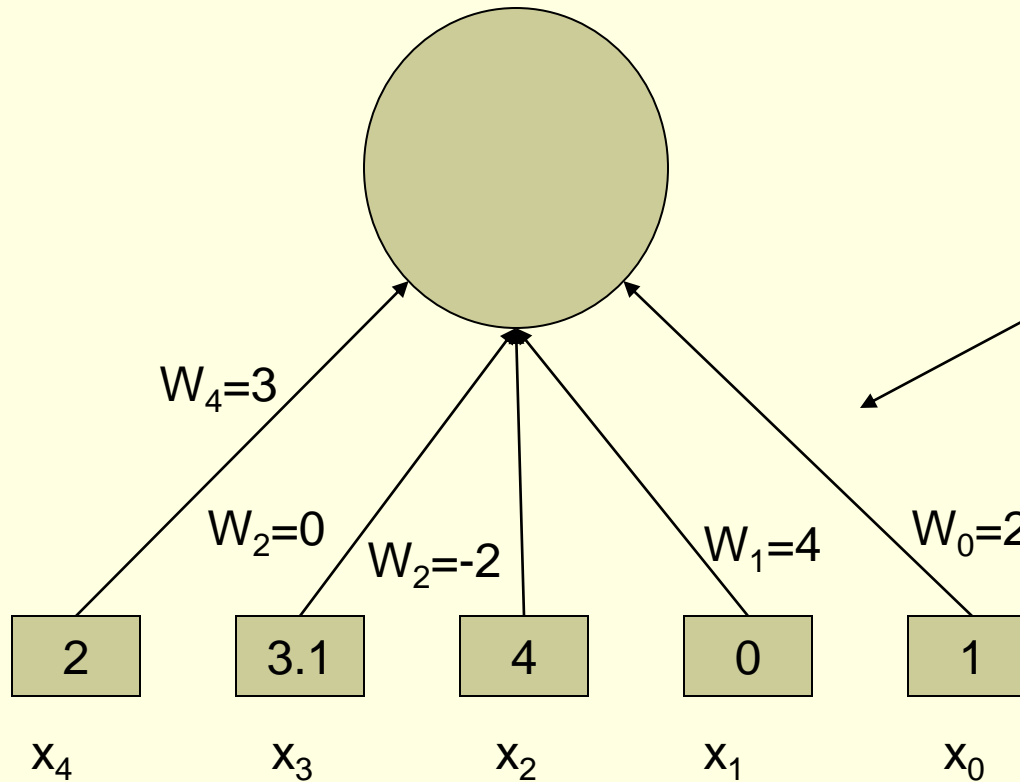
$\text{sgn}(\vec{w} \cdot \vec{x})$

The Perceptron

Output:
classification
of patient
(malignant or
benign)

Weights

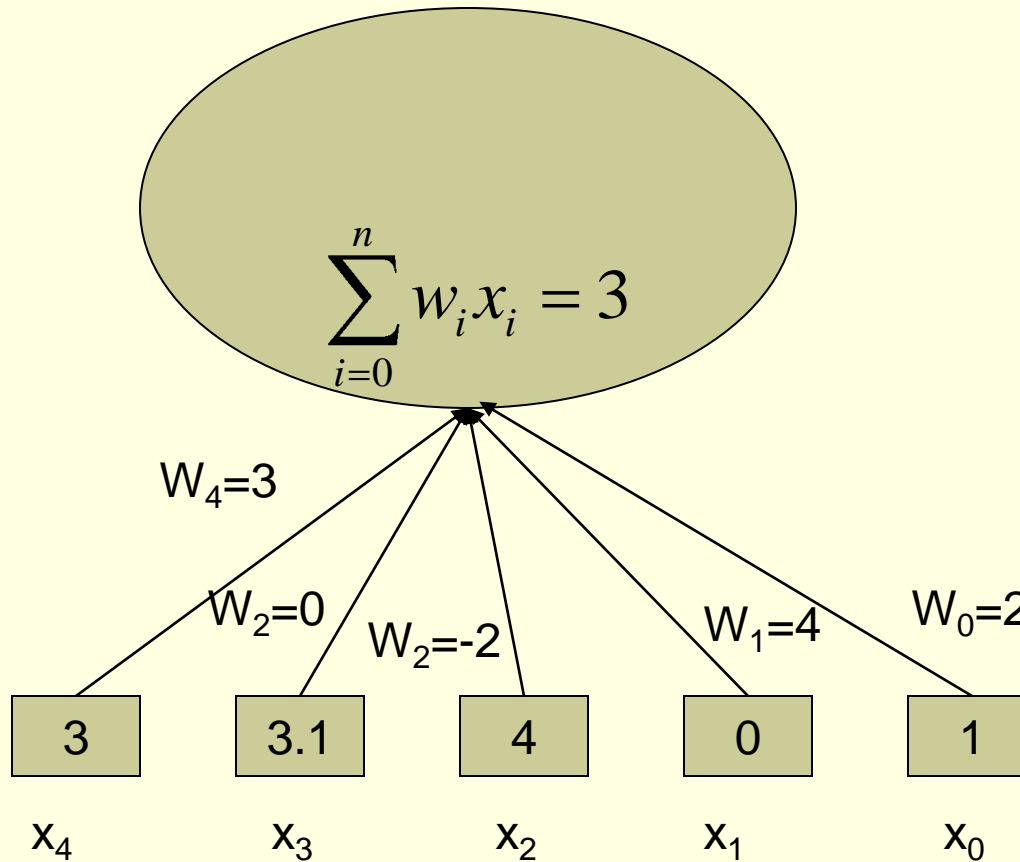
Input:
(attributes of
patient to
classify)



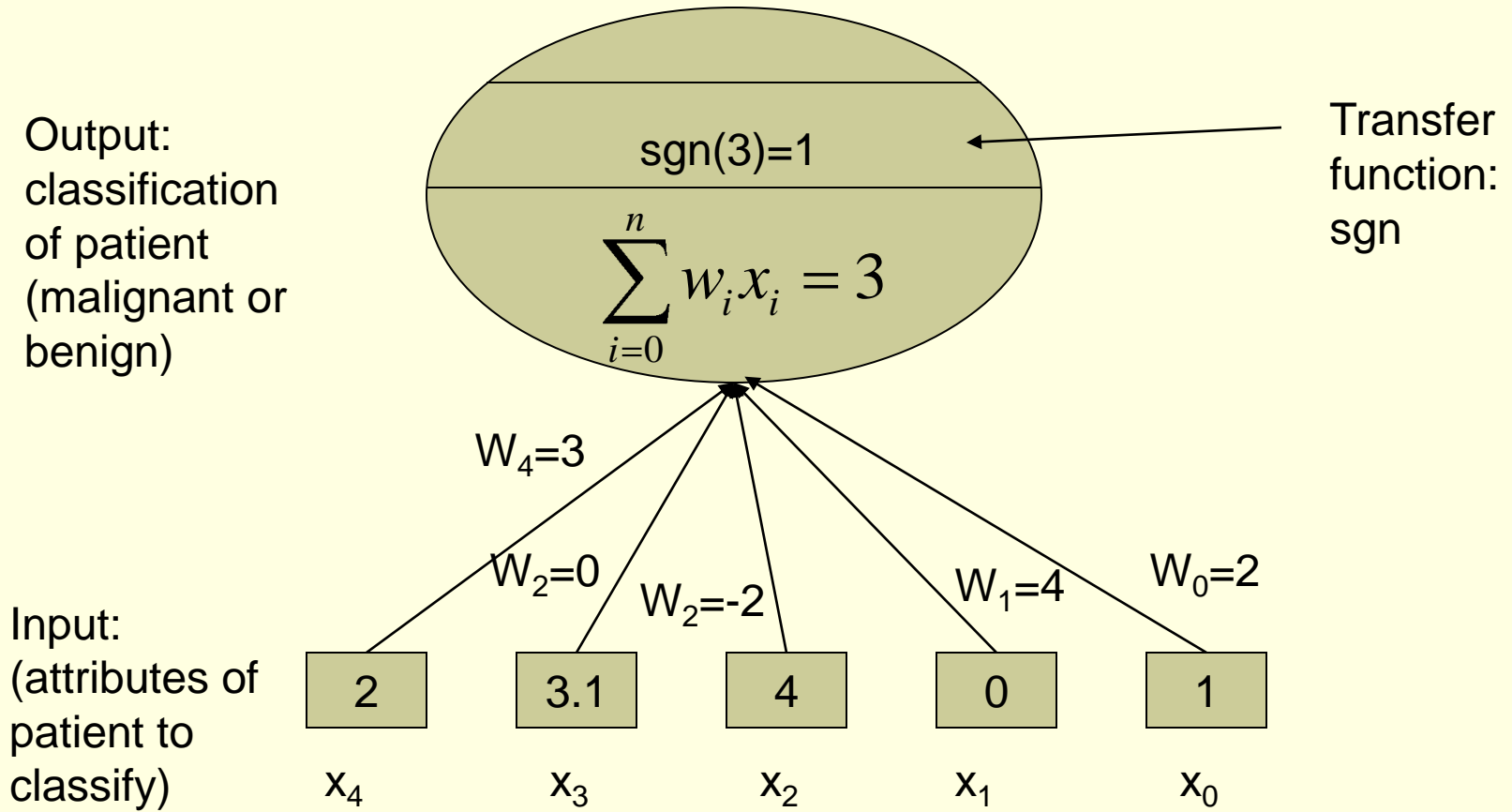
The Perceptron

Output:
classification
of patient
(malignant or
benign)

Input:
(attributes of
patient to
classify)



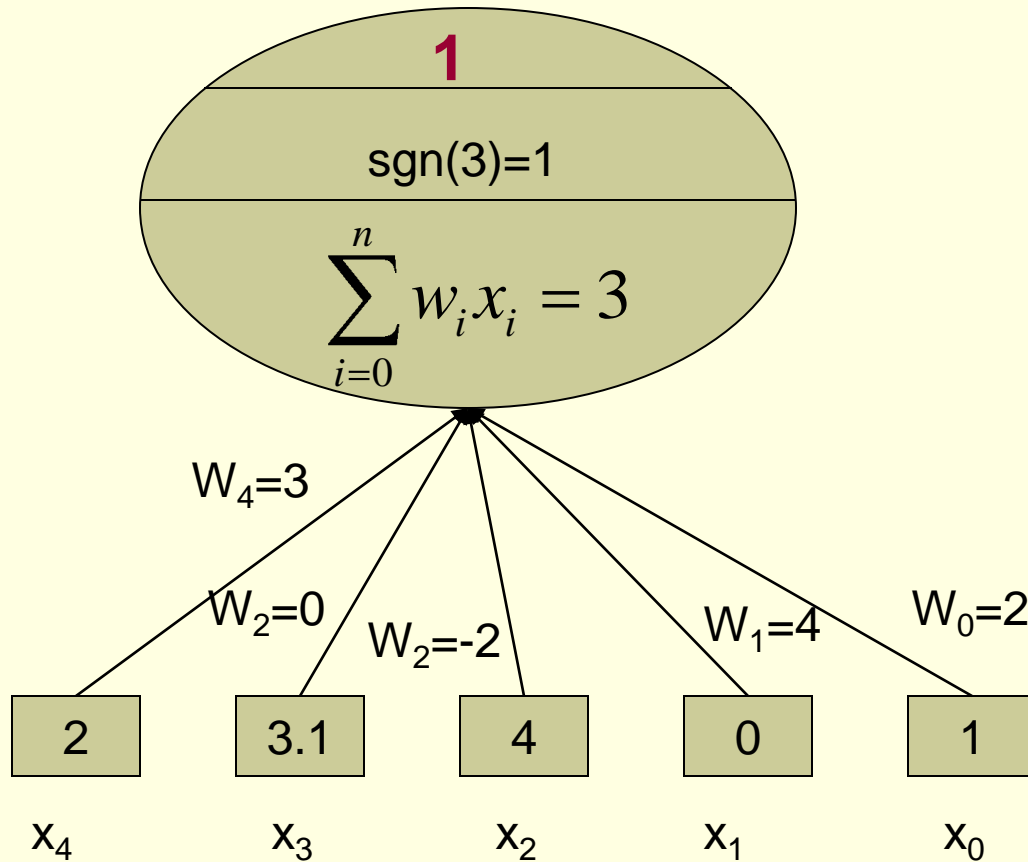
The Perceptron



The Perceptron

Output:
classification
of patient
(malignant or
benign)

Input:
(attributes of
patient to
classify)



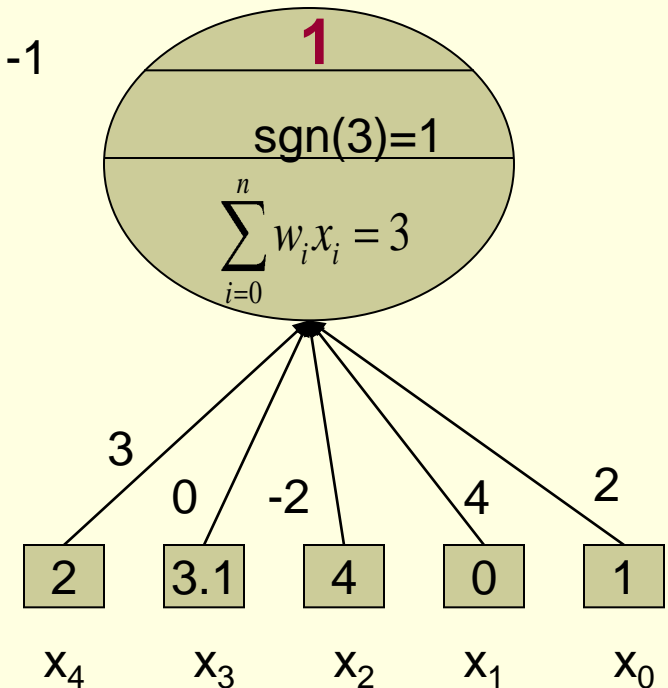
Training a Perceptron

- Use the data to learn a Perceptron that generalizes
- Hypotheses Space: $H = \{\vec{w} \mid \vec{w} \in \mathcal{R}^{n+1}\}$
- Inductive Bias: Prefer hypotheses that do not misclassify any of the training instances (or minimize an error function)
- Search method: perceptron training rule, gradient descent, etc.
- Remember: the problem is to find “good” weights

Training Perceptrons

- Start with random weights
- Update in an intelligent way to improve them using the data
- Intuitively (for the example on the right):
 - Decrease the weights that increase the sum
 - Increase the weights that decrease the sum
- Repeat for all training instances until convergence

True
Output: -1



Perceptron Training Rule

For each misclassified example \vec{x}_d update weights :

$$\Delta w_i = \eta(t_d - o_d)x_{i,d}$$

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

In vector form :

$$\vec{w}' \leftarrow \vec{w} + \eta(t_d - o_d)\vec{x}_d$$

- η : arbitrary learning rate (e.g. 0.5)
- t_d : (true) label of the d th example
- o_d : output of the perceptron on the d th example
- $x_{i,d}$: value of predictor variable i of example d
- $t_d = o_d$: No change (for correctly classified examples)

Explaining the Perceptron Training Rule

$$\text{Rule : } \vec{w}' \leftarrow \vec{w} + \eta(t_d - o_d)\vec{x}_d$$

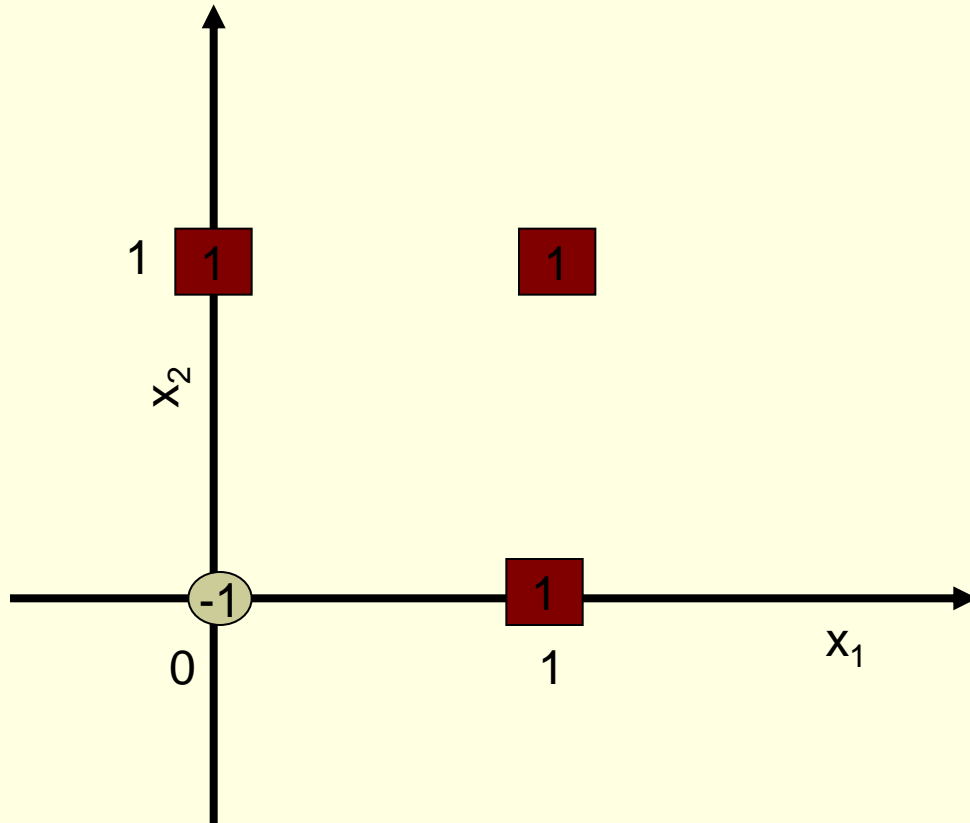
$$\text{Output : } \text{sgn}(\vec{w} \cdot \vec{x}_d)$$

Effect on the output caused by a misclassified example x_d

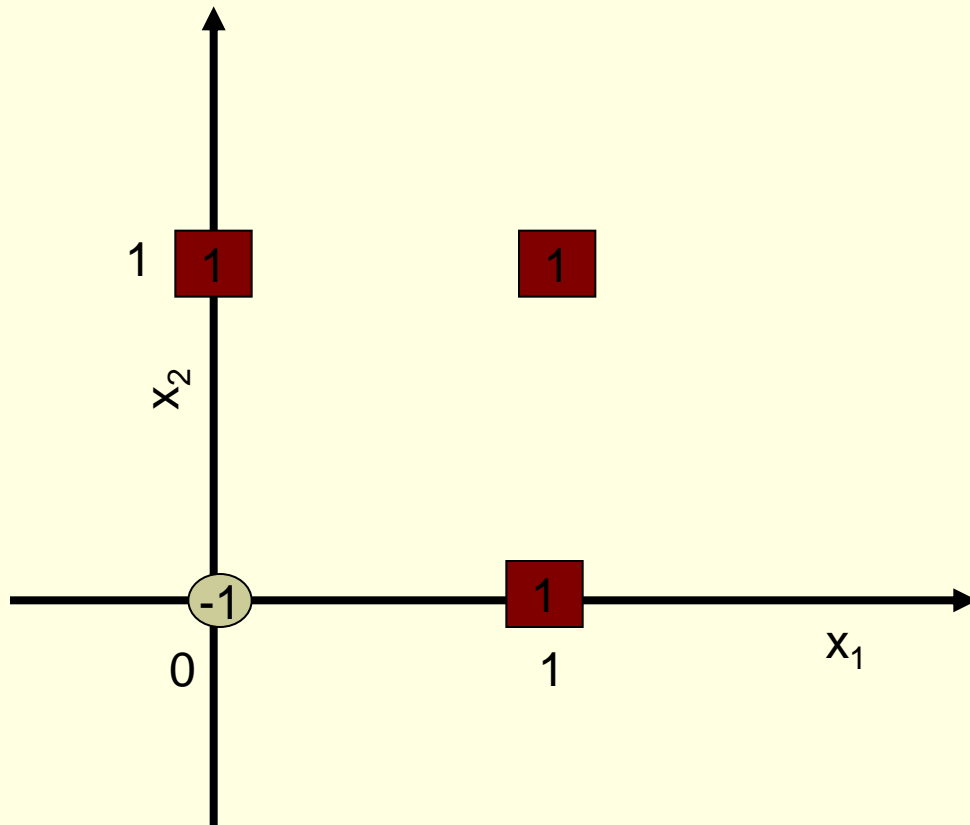
$$\vec{w}' \cdot \vec{x}_d = \vec{w} \cdot \vec{x}_d + \Delta\vec{w} \cdot \vec{x}_d = \vec{w} \cdot \vec{x}_d + \eta(t_d - o_d)\|\vec{x}_d\|^2$$

- $t_d = -1, o_d = 1$: $\vec{w} \cdot \vec{x}_d$ will decrease
- $t_d = 1, o_d = -1$: $\vec{w} \cdot \vec{x}_d$ will increase

Example of Perceptron Training: The OR function



Example of Perceptron Training: The OR function



- Initial random weights:

$$0x_2 + 1x_1 - 0.5 = 0$$

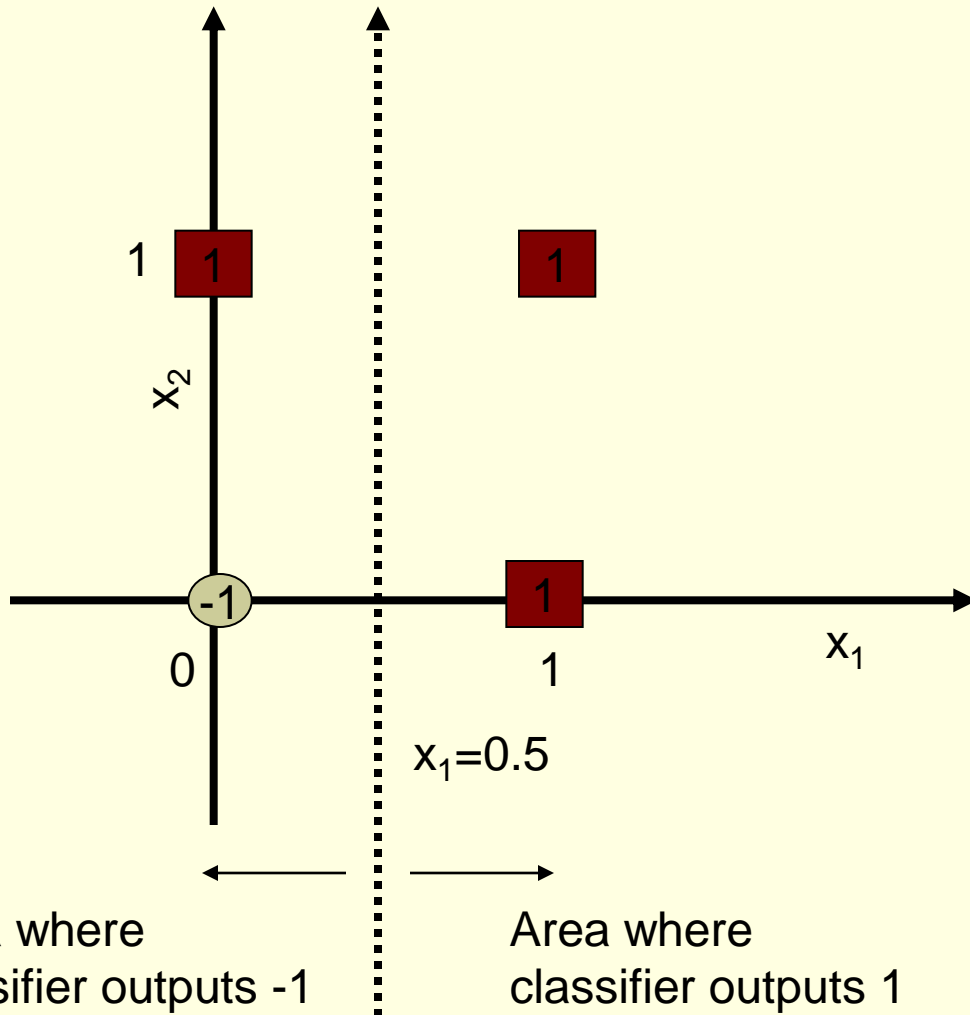
Define line:

$$x_1 = 0.5$$

Thus:

$$\begin{aligned}\vec{w} &= \langle w_2, w_1, w_0 \rangle \\ &= \langle 0, 1, -0.5 \rangle\end{aligned}$$

Example of Perceptron Training: The OR function



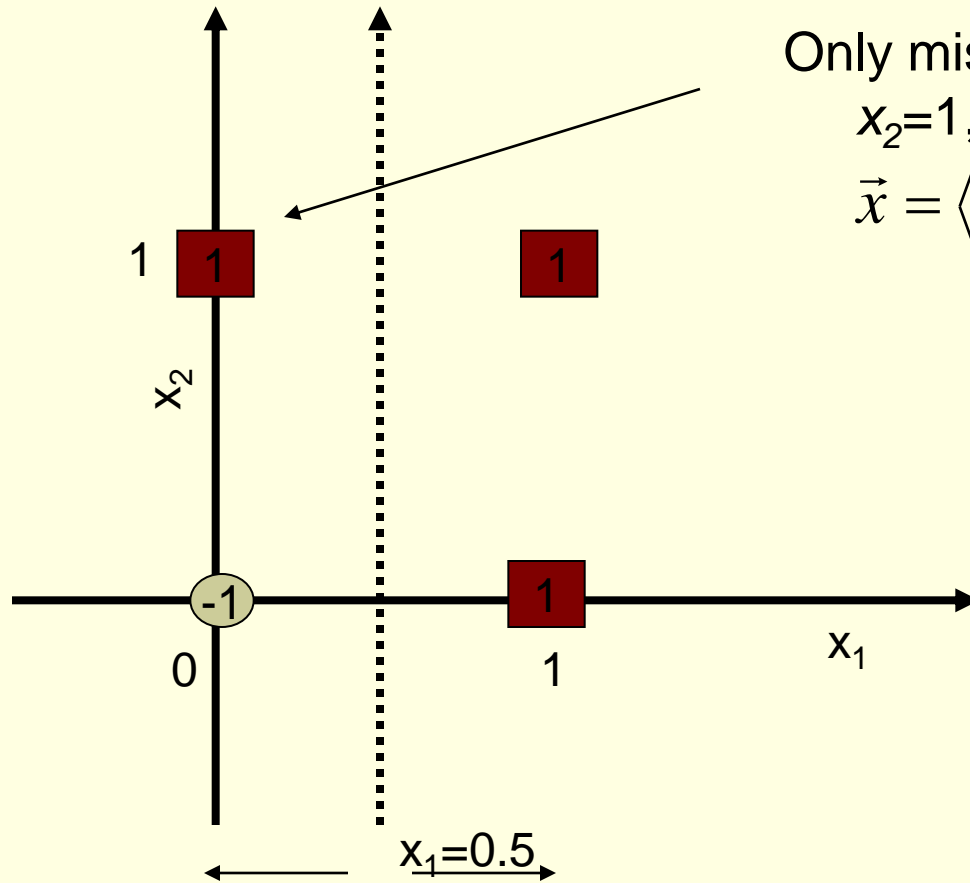
- Initial random weights:

$$0x_2 + 1x_1 - 0.5 = 0$$

Defines line:

$$x_1 = 0.5$$

Example of Perceptron Training: The OR function



Only misclassified example

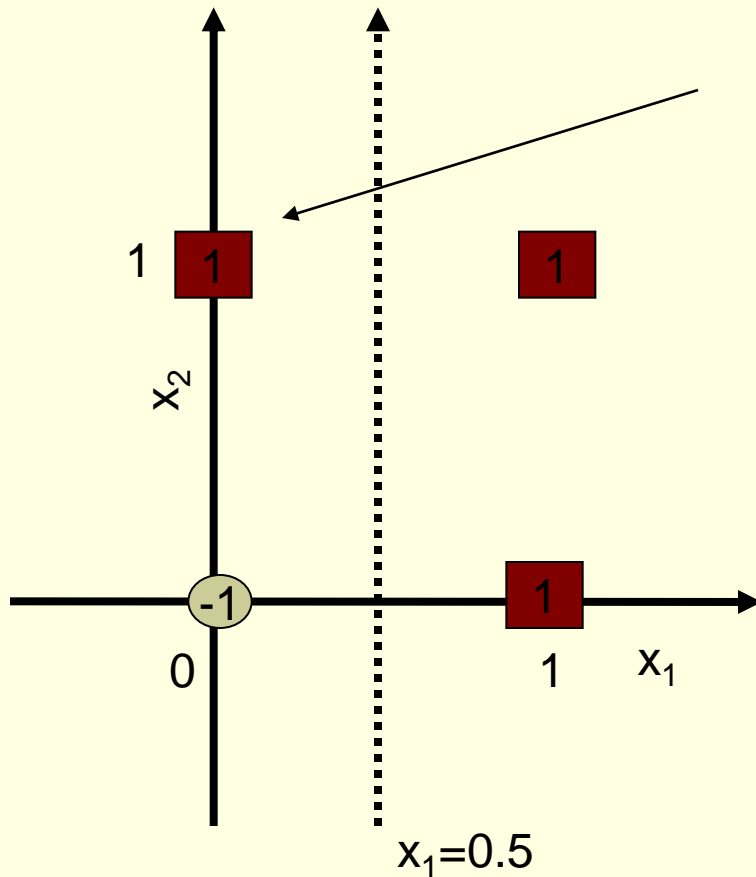
$$x_2=1, x_1=0, x_0 = 1$$

$$\vec{x} = \langle 1, 0, 1 \rangle$$

Area where
classifier outputs -1

Area where
classifier outputs 1

Example of Perceptron Training: The OR function



Only misclassified example

$$x_2=1, x_1=0, x_0 = 1$$

$$\vec{x} = \langle 1, 0, 1 \rangle$$

$$\text{Old Line : } 0x_2 + 1x_1 - 0.5 = 0$$

Update weights

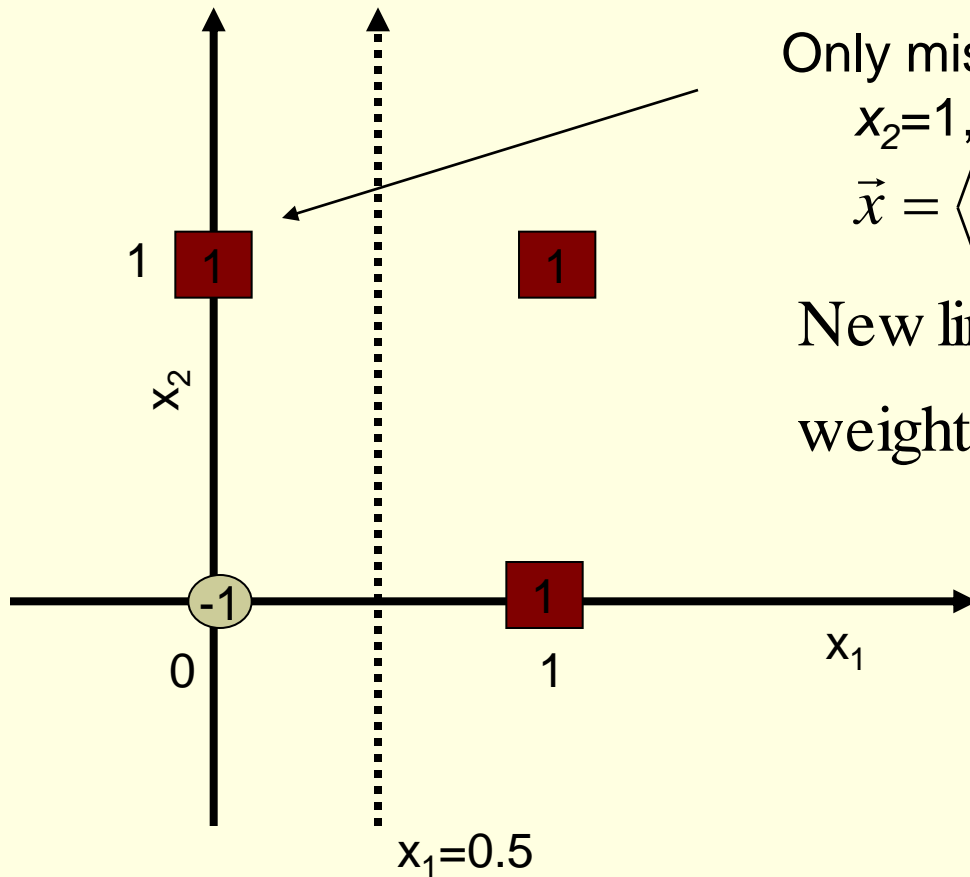
$$\vec{w}' \leftarrow \vec{w} + \eta(t - o)\vec{x}$$

$$\vec{w}' \leftarrow \langle 0, 1, -0.5 \rangle + 0.5 \cdot (1 - (-1)) \cdot \langle 1, 0, 1 \rangle$$

$$\vec{w}' \leftarrow \langle 0, 1, -0.5 \rangle + 1 \cdot \langle 1, 0, 1 \rangle$$

$$\vec{w}' \leftarrow \langle 1, 1, 0.5 \rangle$$

Example of Perceptron Training: The OR function



Only misclassified example

$$x_2=1, x_1=0, x_0=1$$

$$\vec{x} = \langle 1, 0, 1 \rangle$$

New line : $1x_2 + 1x_1 + 0.5 = 0$

weights $\vec{w} = \langle 1, 1, 0.5 \rangle$

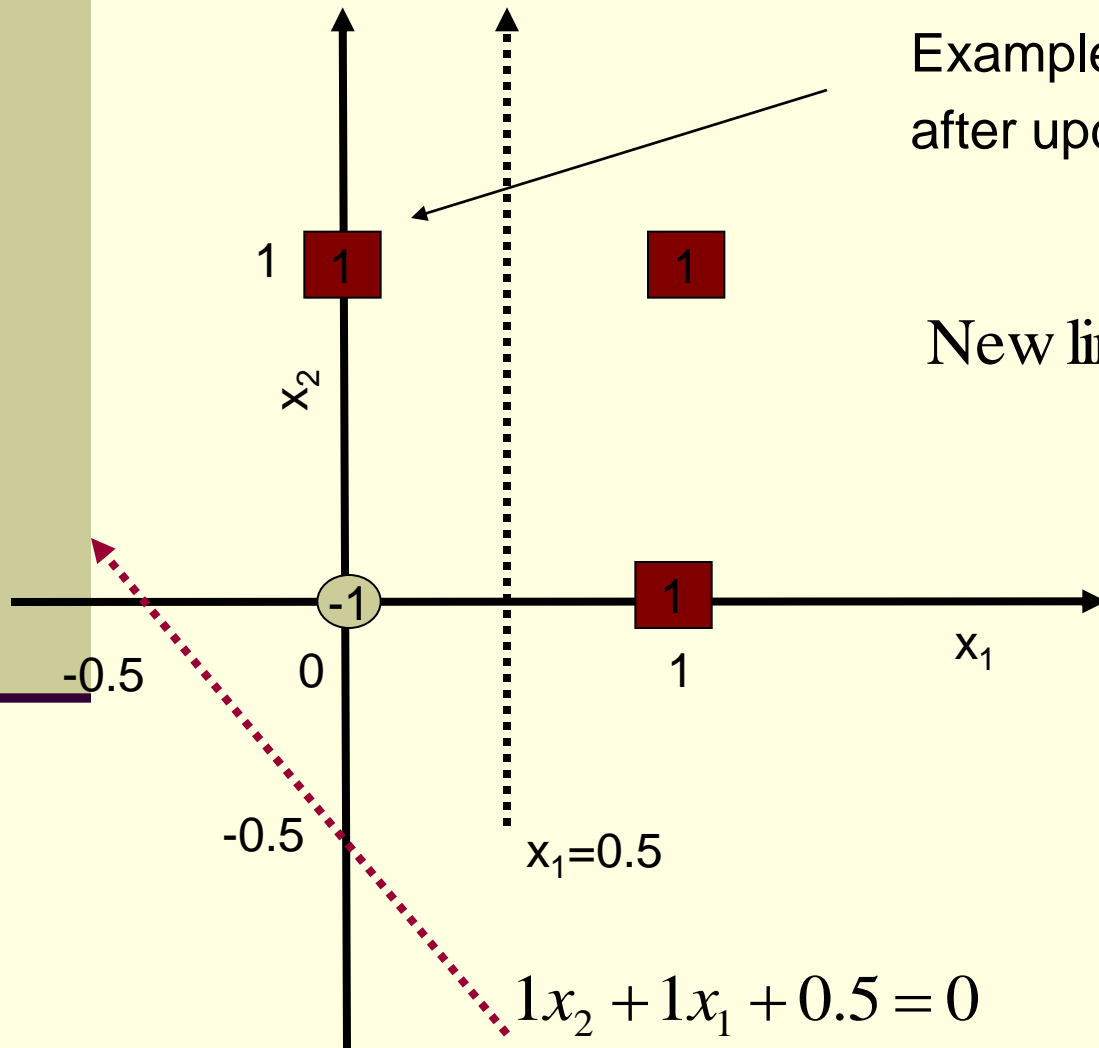
For $x_2=0, x_1=-0.5$

For $x_1=0, x_2=-0.5$

So, new line is:

(next slide)

Example of Perceptron Training: The OR function

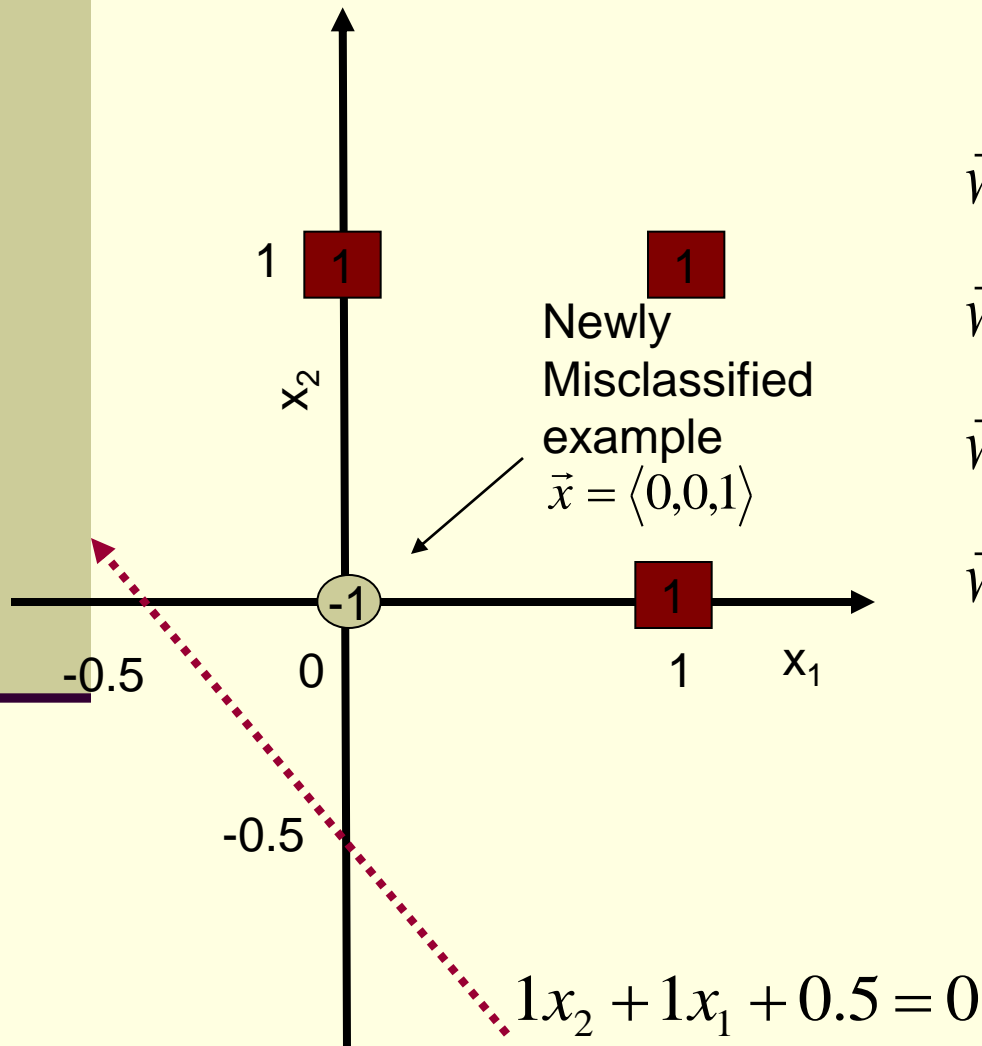


Example correctly classified
after update

$$\text{New line : } 1x_2 + 1x_1 + 0.5 = 0$$

$$1x_2 + 1x_1 + 0.5 = 0$$

Example of Perceptron Training: The OR function



Next iteration:

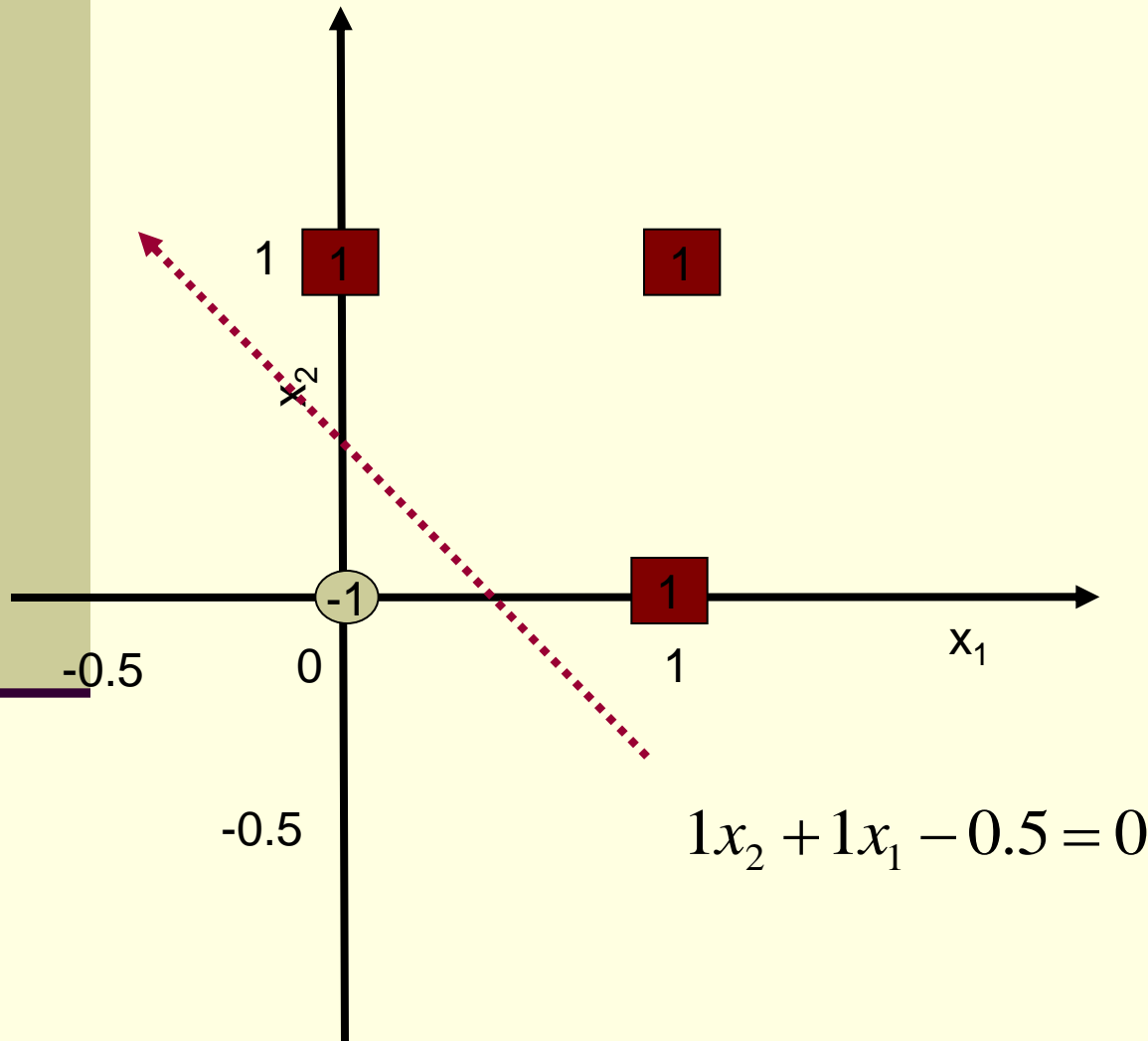
$$\vec{w}' \leftarrow \vec{w} + \eta(t - o)\vec{x}$$

$$\vec{w}' \leftarrow \langle 1, 1, 0.5 \rangle + 0.5 \cdot (-1 - 1) \cdot \langle 0, 0, 1 \rangle$$

$$\vec{w}' \leftarrow \langle 1, 1, 0.5 \rangle - 1 \langle 0, 0, 1 \rangle$$

$$\vec{w}' \leftarrow \langle 1, 1, -0.5 \rangle$$

Example of Perceptron Training: The OR function



New line:
 $1x_2 + 1x_1 - 0.5 = 0$

Perfect
classification

No change
occurs next

Analysis of the Perceptron Training Rule

- Algorithm will always converge within finite number of iterations if the data are linearly separable.
- Otherwise, it may oscillate (no convergence)

Training by Gradient Descent

- Similar but:
 - Always converges
 - Generalizes to training networks of perceptrons (neural networks) and training networks for multcategory classification or regression
- Idea:
 - Define an error function
 - Search for weights that minimize the error, i.e., find weights that zero the error gradient

Setting Up the Gradient Descent

- Squared Error: t_d label of d th example, o_d current output on d th example

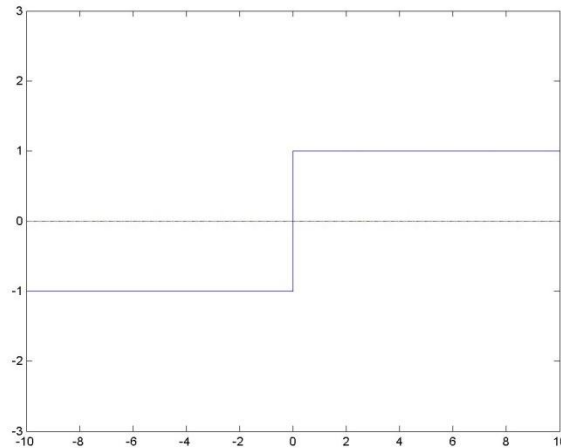
$$E(\vec{w}) = \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2$$

- Minima exist where gradient is zero:

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

The Sign Function is not Differentiable

$$\frac{\partial}{\partial w_i} (-o_d) = -\frac{\partial o_d}{\partial w_i} = 0, \text{ everywhere except } o_d = 0$$



Use Differentiable Transfer Functions

- Replace

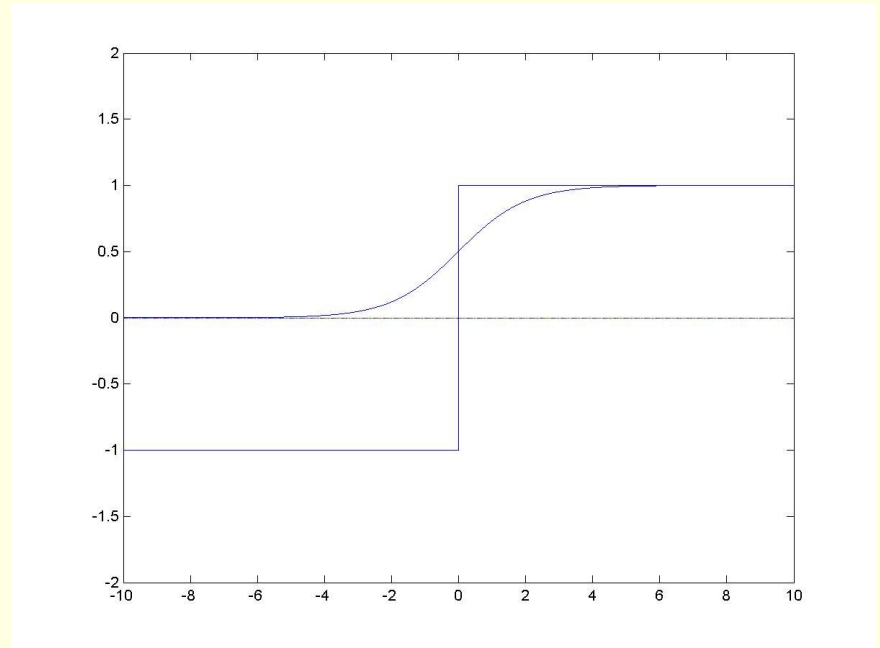
$$\text{sgn}(\vec{w} \cdot \vec{x}_d)$$

- with the sigmoid

$$\text{sig}(\vec{w} \cdot \vec{x}_d)$$

$$\text{sig}(y) = \frac{1}{1 + e^{-y}}$$

$$\frac{d\text{sig}(y)}{dy} = \text{sig}(y)(1 - \text{sig}(y))$$



Calculating the Gradient

$$\begin{aligned}\frac{\partial E}{\partial w_i} &= \sum_{d \in D} (t_d - o_d) \frac{\partial}{\partial w_i} (-o_d) \\ &= \sum_{d \in D} (t_d - o_d) \frac{\partial}{\partial w_i} (-\text{sig}(\vec{w} \cdot \vec{x}_d)) \\ &= \sum_{d \in D} (t_d - o_d) \frac{\partial}{\partial w_i} (-\text{sig}(\vec{w} \cdot \vec{x}_d)) \\ &= - \sum_{d \in D} (t_d - o_d) \frac{\partial \text{sig}(\vec{w} \cdot \vec{x}_d)}{\partial (\vec{w} \cdot \vec{x}_d)} \frac{\partial (\vec{w} \cdot \vec{x}_d)}{\partial w_i} \\ &= - \sum_{d \in D} (t_d - o_d) \text{sig}(\vec{w} \cdot \vec{x}_d) (1 - \text{sig}(\vec{w} \cdot \vec{x}_d)) \frac{\partial}{\partial w_i} (\vec{w} \cdot \vec{x}_d) \\ &= - \sum_{d \in D} (t_d - o_d) \text{sig}(\vec{w} \cdot \vec{x}_d) (1 - \text{sig}(\vec{w} \cdot \vec{x}_d)) \cdot x_{i,d} \\ \\ \nabla E(\vec{w}) &= - \sum_{d \in D} (t_d - o_d) \text{sig}(\vec{w} \cdot \vec{x}_d) (1 - \text{sig}(\vec{w} \cdot \vec{x}_d)) \cdot \vec{x}_d\end{aligned}$$

Updating the Weights with Gradient Descent

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

$$\vec{w} \leftarrow \vec{w} + \eta \sum_{d \in D} (t_d - o_d) \text{sig}(\vec{w} \cdot \vec{x}_d) (1 - \text{sig}(\vec{w} \cdot \vec{x}_d)) \cdot \vec{x}_d$$

- Each weight update goes through all training instances
- Each weight update more expensive but more accurate
- Always converges to a local minimum regardless of the data
- When using the sigmoid: output is a real number between 0 and 1
- Thus, labels (desired outputs) have to be represented with numbers from 0 to 1

Encoding Multiclass Problems

- E.g., 4 nominal classes, A, B, C, D

| X_0 | X_1 | X_2 | ... | Class |
|-------|-------|-------|-----|-------|
| 1 | 0.4 | -1 | | A |
| 1 | 9 | 0.5 | | A |
| 1 | 1 | 3 | | C |
| 1 | 8.4 | -.8 | | B |
| 1 | -3.4 | .2 | | D |

Encoding Multiclass Problems

- Use one perceptron (output unit) and encode the output as follows:
 - Use 0.125 to represent class A (middle point of [0,.25])
 - Use 0.375, to represent class B (middle point of [.25,.50])
 - Use 0.625, to represent class C (middle point of [.50,.75])
 - Use 0.875, to represent class D (middle point of [.75,1])
- The training data then becomes:

| X_0 | X_1 | X_2 | ... | Class |
|-------|-------|-------|-----|-------|
| 1 | 0.4 | -1 | | 0.125 |
| 1 | 9 | 0.5 | | 0.125 |
| 1 | 1 | 3 | | 0.625 |
| 1 | 8.4 | -0.8 | | 0.365 |
| 1 | -3.4 | .2 | | 0.875 |

Encoding Multiclass Problems

- Use one perceptron (output unit) and encode the output as follows:
 - Use 0.125 to represent class A (middle point of [0,.25])
 - Use 0.375, to represent class B (middle point of [.25,.50])
 - Use 0.625, to represent class C (middle point of [.50,.75])
 - Use 0.875, to represent class D (middle point of [.75,1])
- To classify a new input vector \vec{x} :

If $\text{sig}(\vec{w} \cdot \vec{x}) \in [0,.25]$ classify as Class A

If $\text{sig}(\vec{w} \cdot \vec{x}) \in [.25,.5]$ classify as Class B

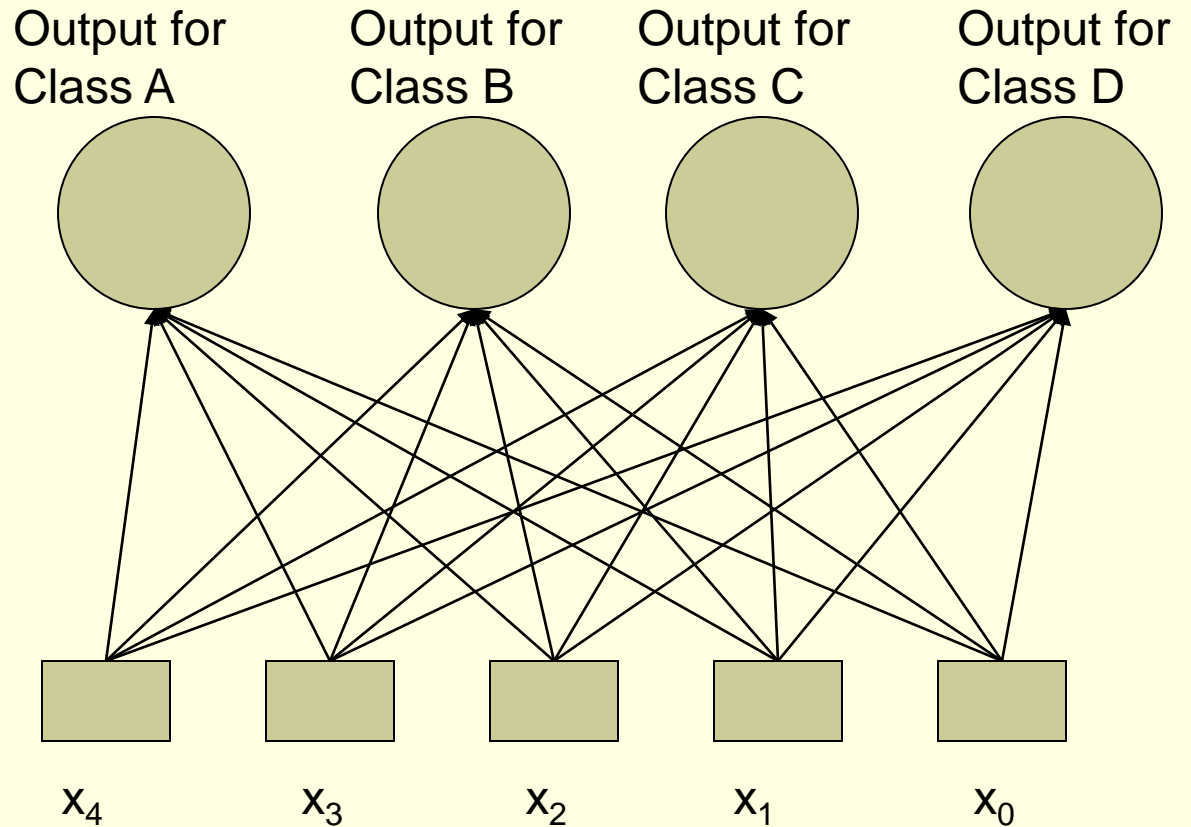
If $\text{sig}(\vec{w} \cdot \vec{x}) \in [.5,.75]$ classify as Class C

If $\text{sig}(\vec{w} \cdot \vec{x}) \in [.75,.1]$ classify as Class D

- For two classes only and a sigmoid unit suggested values 0.1 and 0.9 (or 0.25 and 0.75)

1-of-M Encoding

- Assign to class with largest output



1-of-M Encoding

- E.g., 4 nominal classes, A, B, C, D

| X_0 | X_1 | X_2 | ... | Class A | Class B | Class C | Class D |
|-------|-------|-------|-----|------------|------------|------------|------------|
| 1 | 0.4 | -1 | | 0.9 | 0.1 | 0.1 | 0.1 |
| 1 | 9 | 0.5 | | 0.9 | 0.1 | 0.1 | 0.1 |
| 1 | 1 | 3 | | 0.1 | 0.1 | 0.9 | 0.1 |
| 1 | 8.4 | -.8 | | 0.1 | 0.9 | 0.1 | 0.1 |
| 1 | -3.4 | .2 | | 0.1 | 0.1 | 0.1 | 0.9 |

Encoding the Input

- Variables taking real values (e.g. magnesium level)
 - Input directly to the Perceptron
- Variables taking discrete ordinal numerical values
 - Input directly to the Perceptron (scale linearly to $[0,1]$)
- Variables taking discrete ordinal non-numerical values (e.g. temperature low, normal, high)
 - Assign a number (from $[0,1]$) to each value in the same order:
 - Low $\leftarrow 0$
 - Normal $\leftarrow 0.5$
 - High $\leftarrow 1$
- Variables taking nominal values
 - Assign a number (from $[0,1]$) to each value (like above)
 - OR,
 - Create a new variable for each value taking. The new variable is 1 when the original variable is assigned that value, and 0 otherwise (distributed encoding)

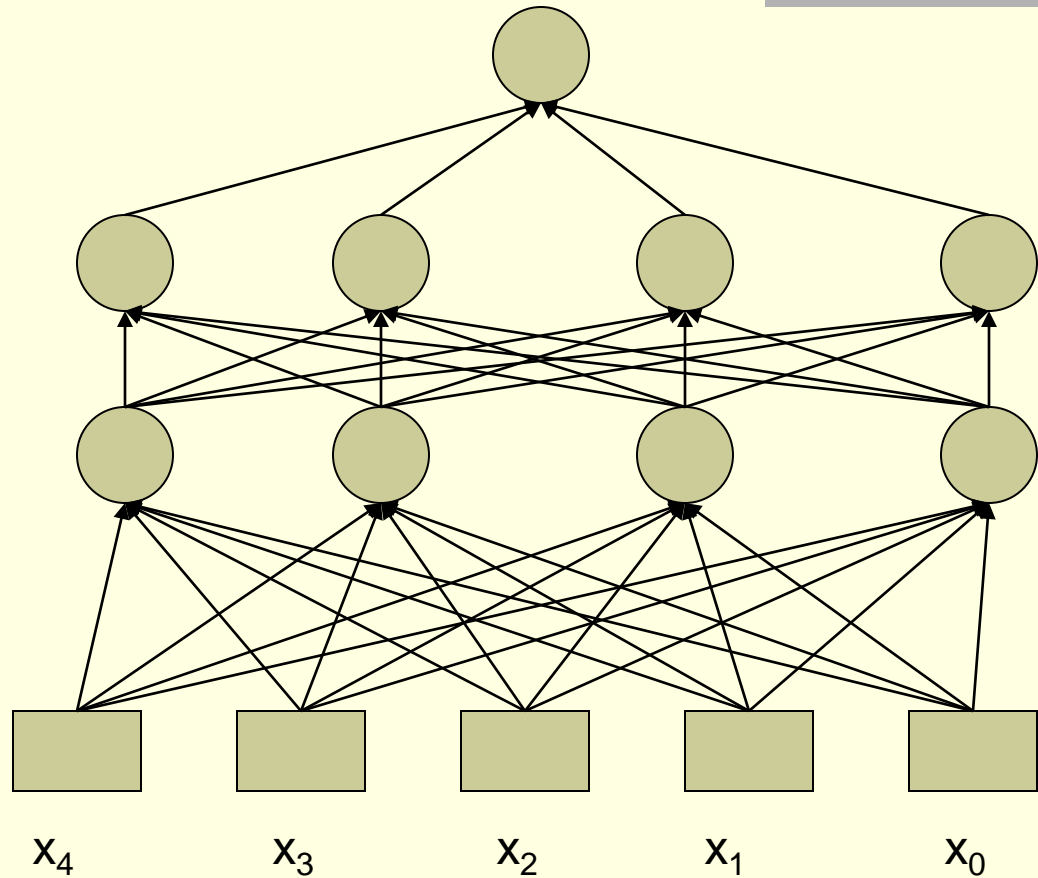
Feed-Forward Neural Networks

Output
Layer

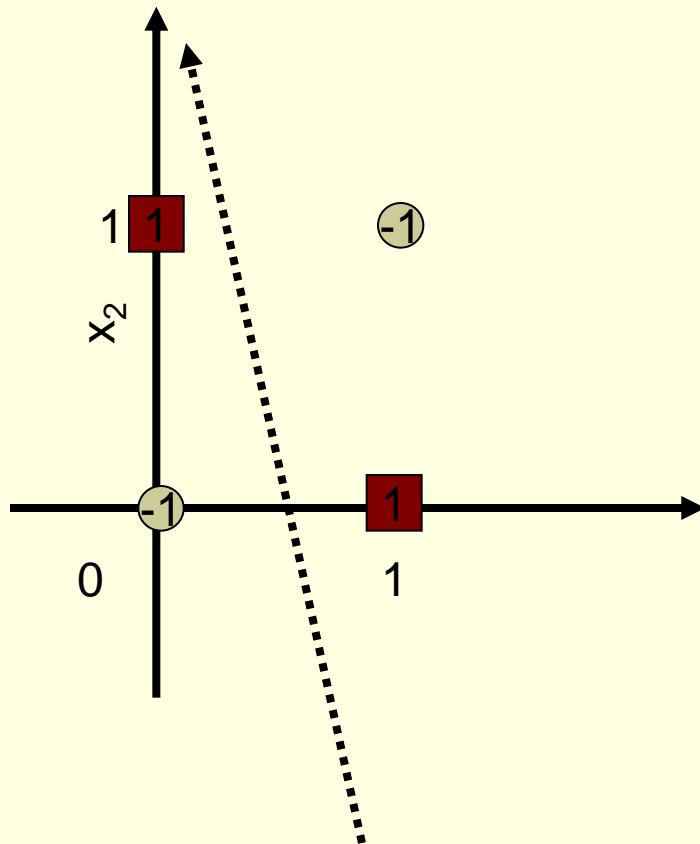
Hidden
Layer 2

Hidden
Layer 1

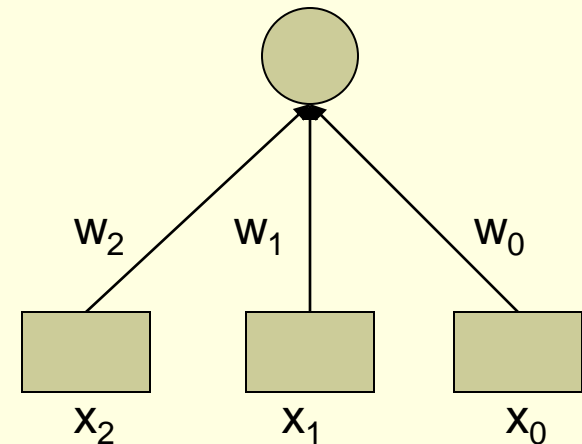
Input
Layer



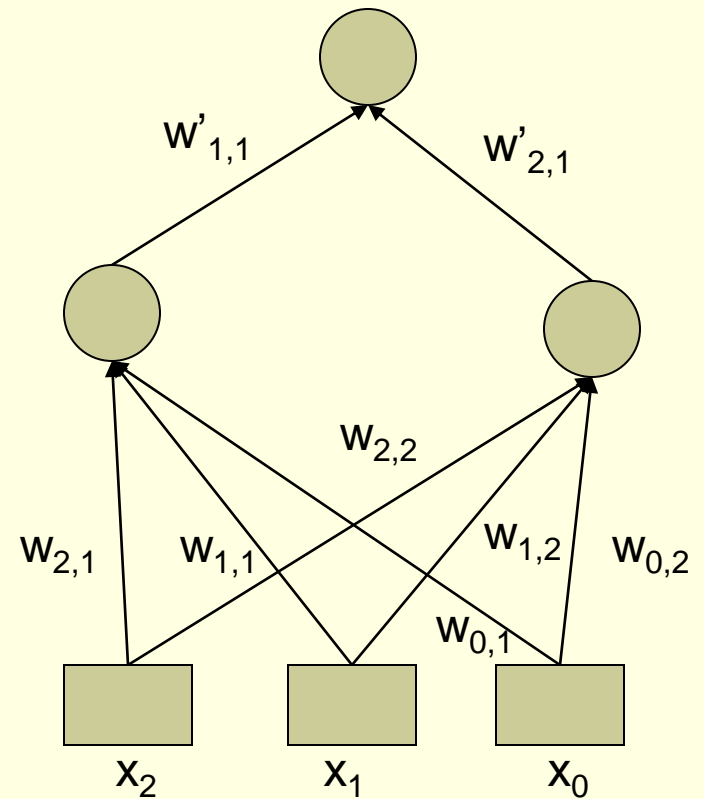
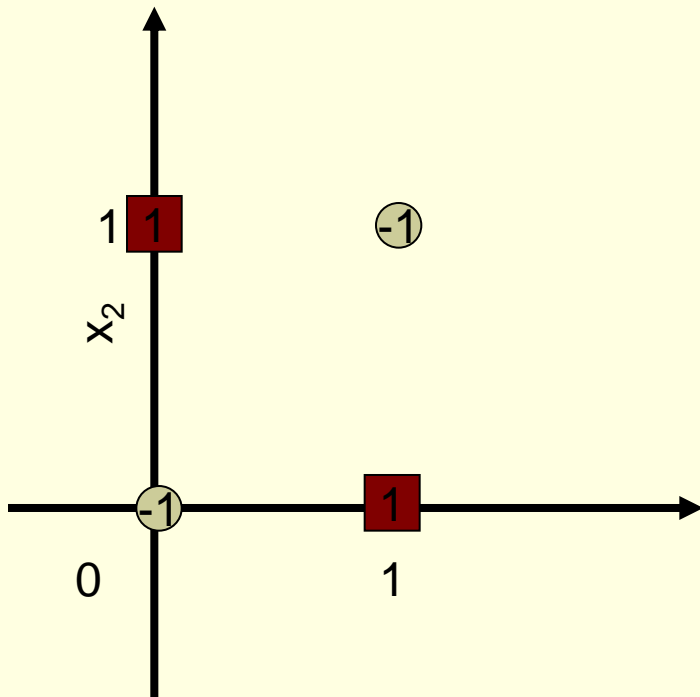
Increased Expressiveness Example: Exclusive OR



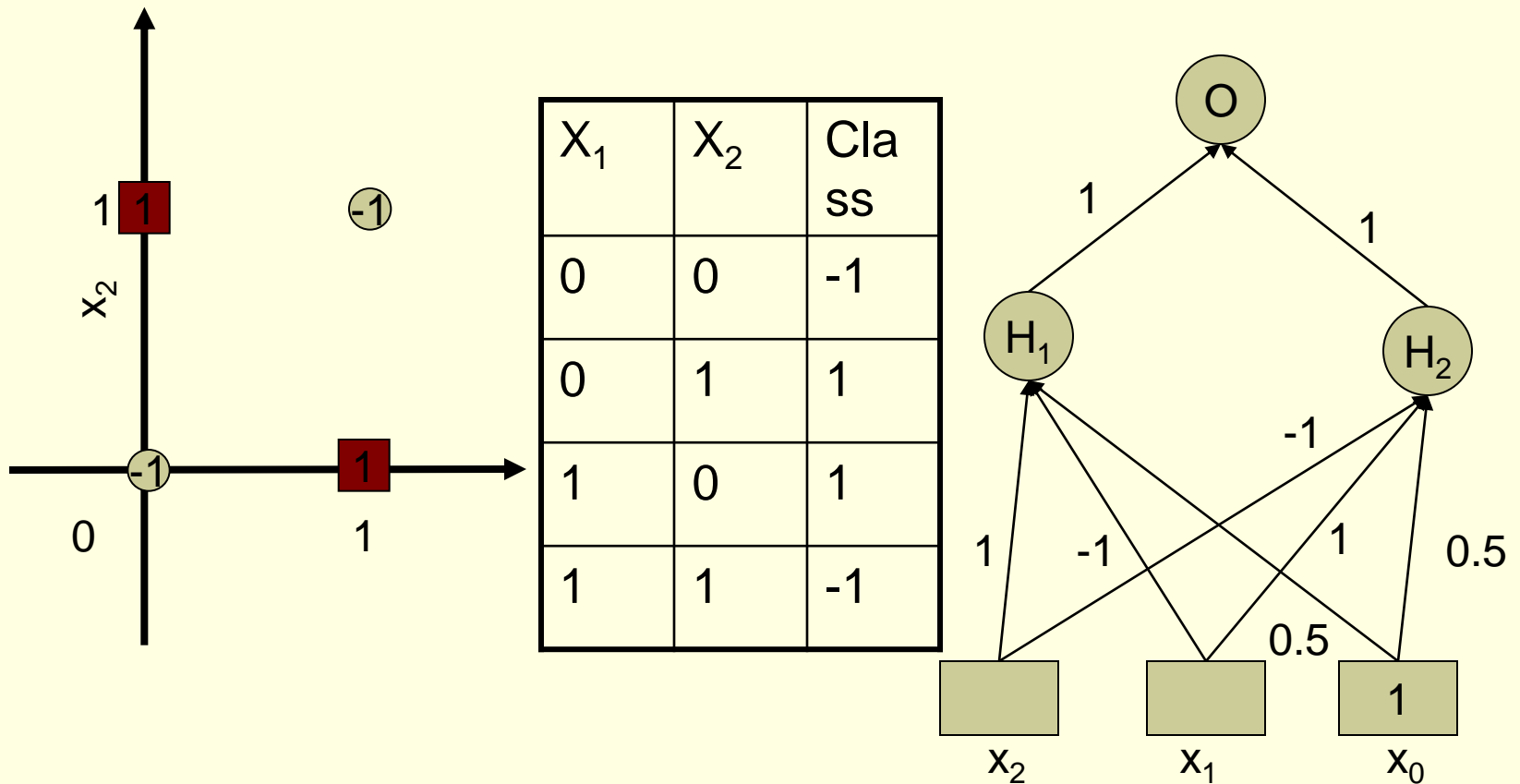
No line (no set of three weights) can separate the training examples (learn the true function).



Increased Expressiveness Example

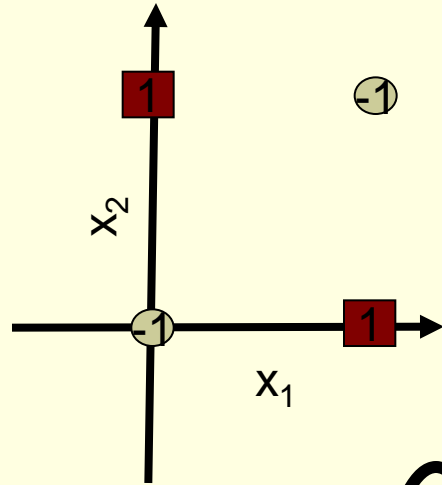


Increased Expressiveness Example

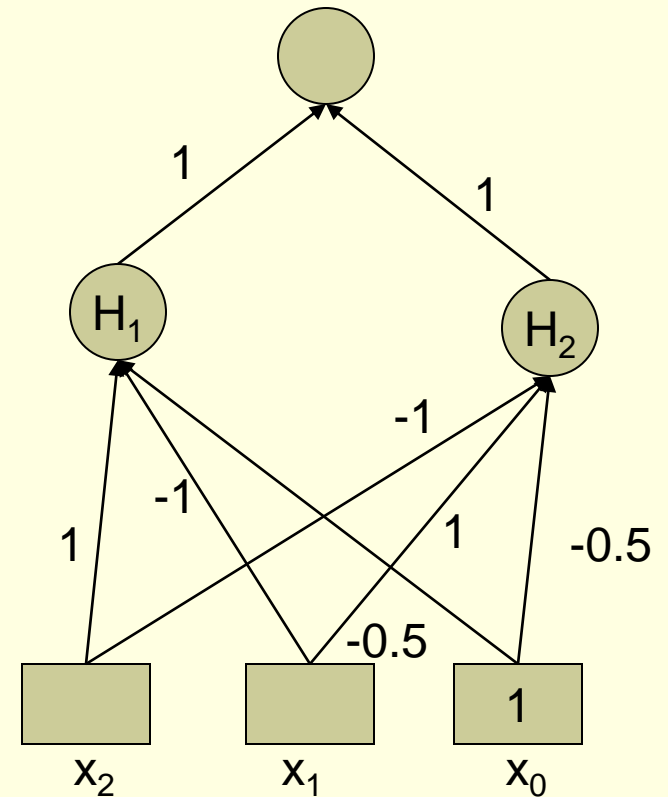


All nodes have the sign function as transfer function in this example

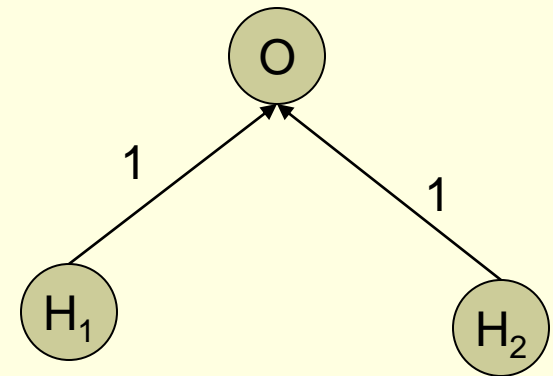
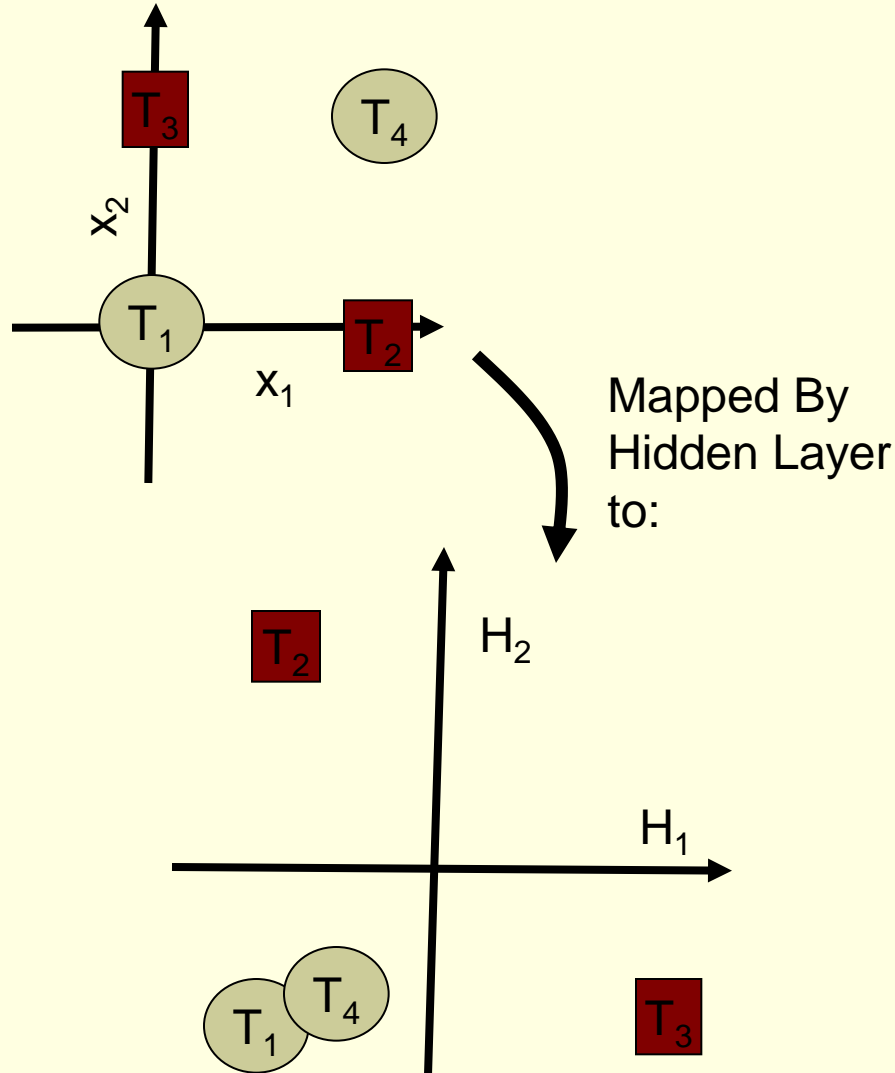
Increased Expressiveness Example



| | X_1 | X_2 | C | H_1 | H_2 | O |
|-------|-------|-------|----------|-------|-------|----------|
| T_1 | 0 | 0 | -1 | -1 | -1 | -1 |
| T_2 | 0 | 1 | 1 | -1 | 1 | 1 |
| T_3 | 1 | 0 | 1 | 1 | -1 | 1 |
| T_4 | 1 | 1 | -1 | -1 | -1 | -1 |

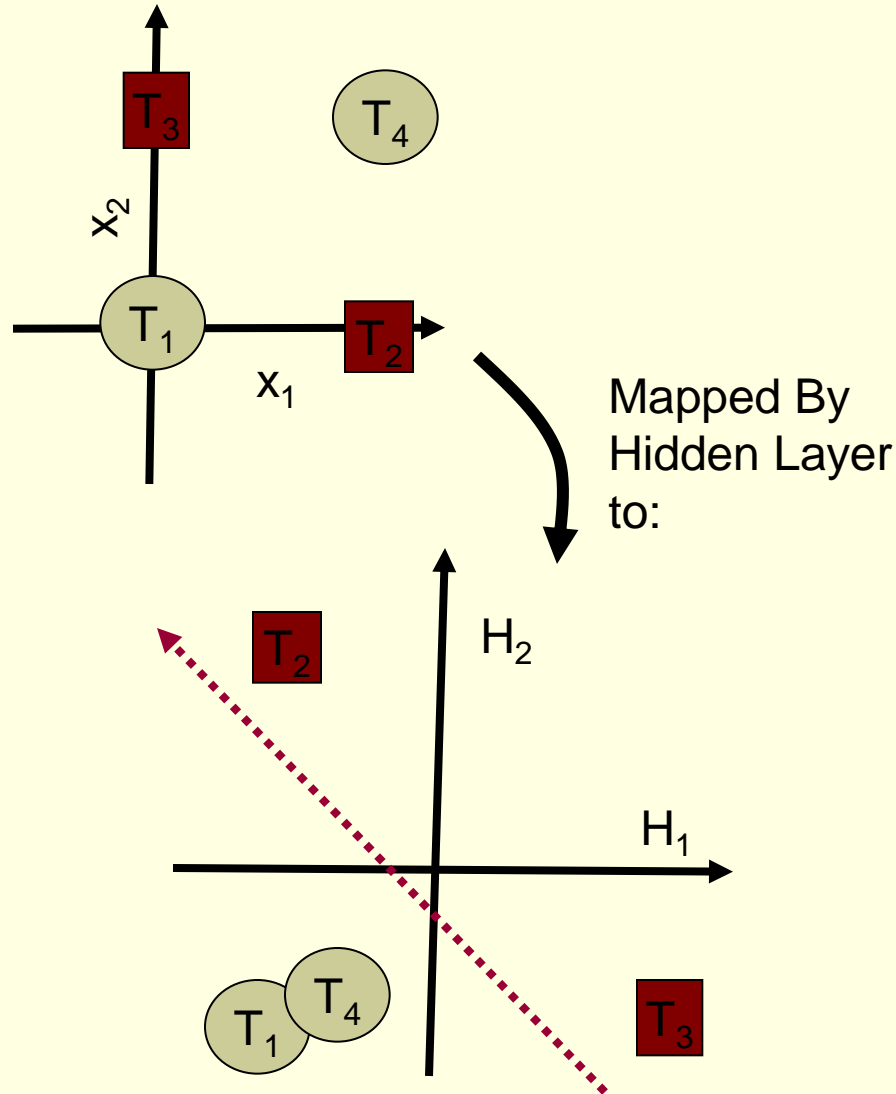


From the Viewpoint of the Output Layer



| | C | H_1 | H_2 | O |
|-------|-----------|-------|-------|-----------|
| T_1 | -1 | -1 | -1 | -1 |
| T_2 | 1 | -1 | 1 | 1 |
| T_3 | 1 | 1 | -1 | 1 |
| T_4 | -1 | -1 | -1 | -1 |

From the Viewpoint of the Output Layer



- Each hidden layer maps to a new feature space
- Each hidden node is a new constructed feature
- Original Problem may become separable (or easier)

How to Train Multi-Layered Networks

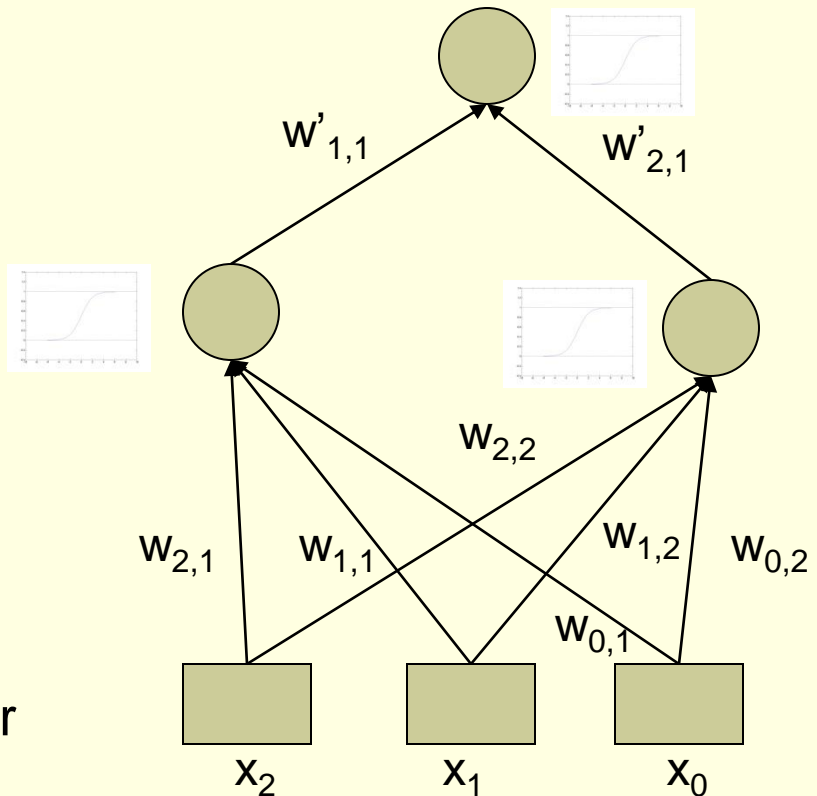
- Select a network structure (number of hidden layers, hidden nodes, and connectivity).
- Select transfer functions that are differentiable.
- Define a (differentiable) error function.
- Search for weights that minimize the error function, using gradient descent or other optimization method.
- **BACKPROPAGATION**

How to Train Multi-Layered Networks

- Select a network structure (number of hidden layers, hidden nodes, and connectivity).
- Select transfer functions that are differentiable.
- Define a (differentiable) error function.

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

- Search for weights that minimize the error function, using gradient descent or other optimization method.



BackPropagation

For a given input vector \vec{x}

Notation :

o_u output of every unit u in network

t desired output

$w_{i \rightarrow j}$ weight from unit i to unit j

$x_{i \rightarrow j}$ input from unit i going to unit j

Define :

$\delta_k = o_k(1 - o_k)(t - o_k)$, when k is the output unit

$$\delta_k = o_k(1 - o_k) \sum_{u \in \text{Outputs of unit } k} w_{k \rightarrow u} \delta_u$$

Update weights rule :

$$w'_{i \rightarrow j} = w_{i \rightarrow j} + \eta \delta_j x_{i \rightarrow j}$$

Training with BackPropagation

- Going once through all training examples and updating the weights: one epoch
- Iterate until a stopping criterion is satisfied
- The hidden layers learn new features and map to new spaces

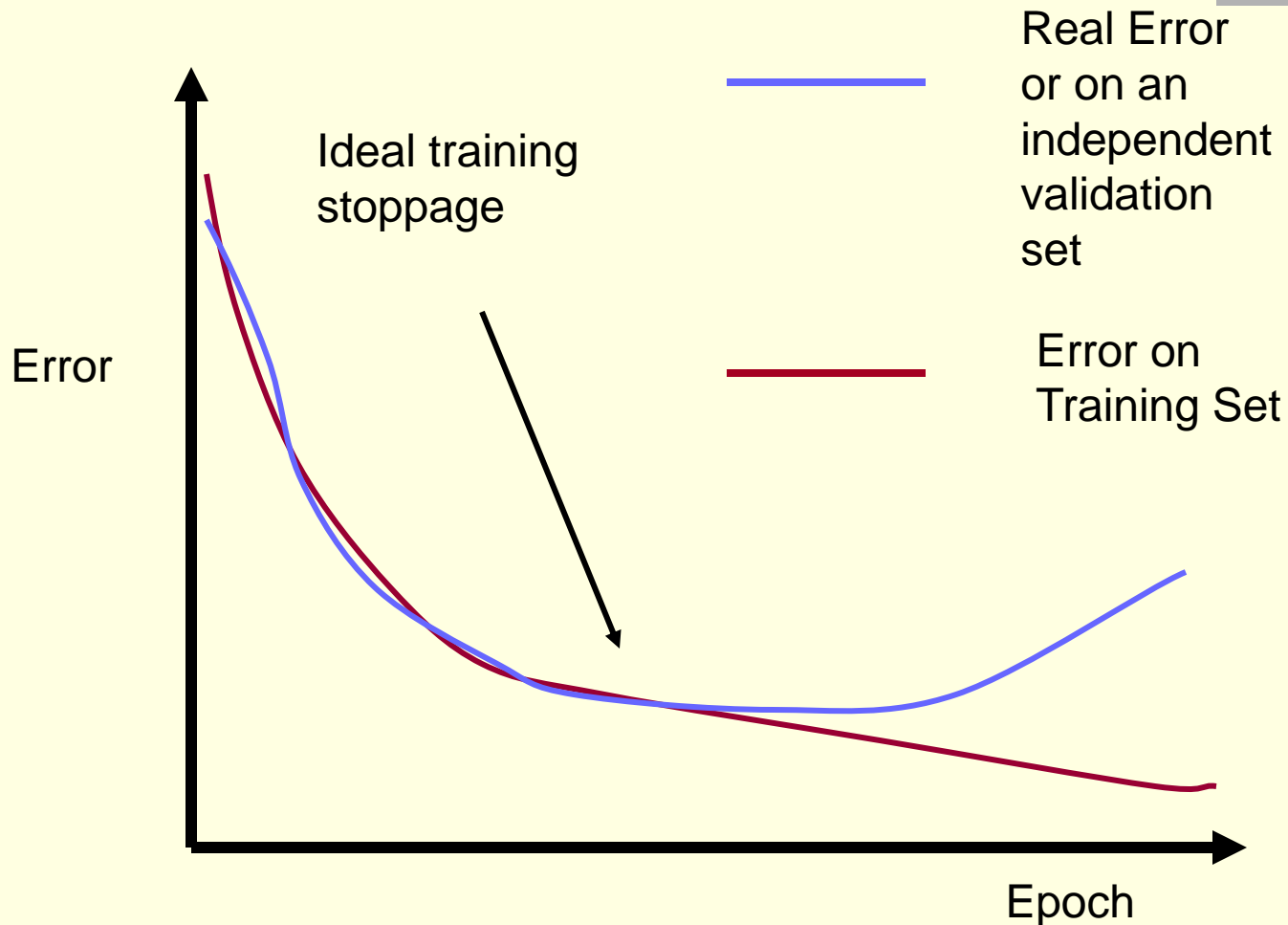
Overfitting with Neural Networks

- If number of hidden units (and weights) is large, it is easy to memorize the training set (or parts of it) and not generalize
- Typically, the optimal number of hidden units is much smaller than the input units
- Each hidden layer maps to a space of smaller dimension

Avoiding Overfitting : Method 1

- The weights that minimize the error function may create complicate decision surfaces
- Stop minimization early by using a validation data set
 - Gives a preference to smooth and simple surfaces

Typical Training Curve



Example of Training Stopping Criteria

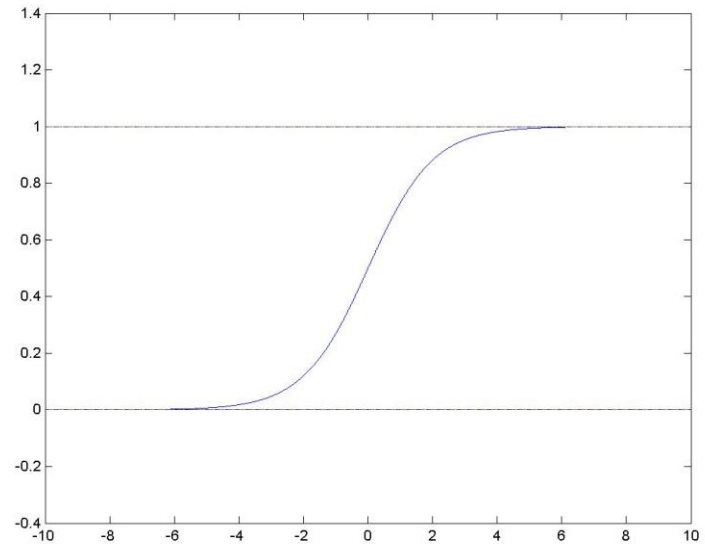
- Split data to train-validation-test sets
 - Train on train, until error in validation set is increasing (more than *epsilon* the last *m* iterations), or
 - until a maximum number of epochs is reached
- Evaluate final performance on test set

Avoiding Overfitting in Neural Networks: Method 2

- Sigmoid almost linear around zero
- Small weights imply decision surfaces that are almost linear
- Instead of trying to minimize only the error, minimize the error while penalizing for large weights

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

- Again, this imposes a preference for smooth and simple (linear) surfaces



Classification with Neural Networks

- Determine representation of input:
 - E.g., Religion {Christian, Muslim, Jewish}
 - Represent as one input taking three different values, e.g. 0.2, 0.5, 0.8
 - Represent as three inputs, taking 0/1 values
- Determine representation of output (for multiclass problems)
 - Single output unit vs Multiple binary units

Classification with Neural Networks

- Select
 - Number of hidden layers
 - Number of hidden units
 - Connectivity
 - Typically: one hidden layer, hidden units is a small fraction of the input units, full connectivity
- Select error function
 - Typically: minimize mean squared error (with penalties for large weights), maximize log likelihood of the data

Classification with Neural Networks

- Select a training method:
 - Typically gradient descent (corresponds to vanilla Backpropagation)
 - Other optimization methods can be used:
 - Backpropagation with momentum
 - Trust-Region Methods
 - Line-Search Methods
 - Congugate Gradient methods
 - Newton and Quasi-Newton Methods
- Select stopping criterion

Classifying with Neural Networks

- Select a training method:
 - May include also searching for optimal structure
 - May include extensions to avoid getting stuck in local minima
 - Simulated annealing
 - Random restarts with different weights

Classifying with Neural Networks

- Split data to:
 - Training set: used to update the weights
 - Validation set: used in the stopping criterion
 - Test set: used in evaluating generalization error (performance)

Other Error Functions in Neural Networks

- Minimizing cross entropy with respect to target values
 - network outputs interpretable as probability estimates

Representational Power

- Perceptron: Can learn only linearly separable functions
- Boolean Functions: learnable by a NN with one hidden layer
- Continuous Functions: learnable with a NN with one hidden layer and sigmoid units
- Arbitrary Functions: learnable with a NN with two hidden layers and sigmoid units
- Number of hidden units in all cases unknown

Issues with Neural Networks

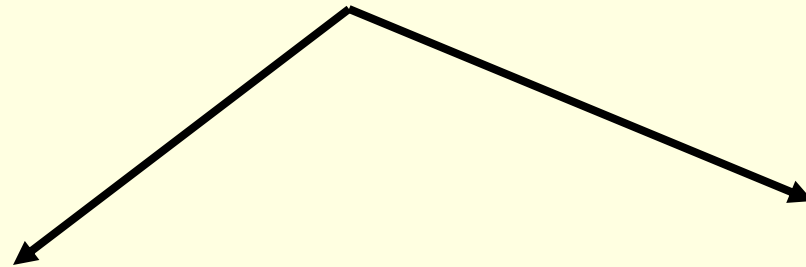
- No principled method for selecting number of layers and units
 - Tiling: start with a small network and keep adding units
 - Optimal brain damage: start with a large network and keep removing weights and units
 - Evolutionary methods: search in the space of structures for one that generalizes well
- No principled method for most other design choices

Important but not Covered in This Tutorial

- Very hard to understand the classification logic from direct examination of the weights
- Large recent body of work in extracting symbolic rules and information from Neural Networks
- Recurrent Networks, Associative Networks, Self-Organizing Maps, Committees or Networks, Adaptive Resonance Theory etc.

Why the Name Neural Networks?

Initial models that simulate real neurons to use for classification



Efforts to improve and understand classification independent of similarity to biological neural networks

Efforts to simulate and understand biological neural networks to a larger degree

Conclusions

- Can deal with both real and discrete domains
- Can also perform density or probability estimation
- Very fast classification time
- Relatively slow training time (does not easily scale to thousands of inputs)
- One of the most successful classifiers yet
- Successful design choices still a black art
- Easy to overfit or underfit if care is not applied

Suggested Further Reading

- Tom Mitchell, Introduction to Machine Learning, 1997
- Hastie, Tibshirani, Friedman, The Elements of Statistical Learning, Springer 2001
- Hundreds of papers and books on the subject