CSC242: Intro to AI

Lecture 18:
Details on Decision Trees;
Neural Networks Part I

Details on Learning Decision Trees

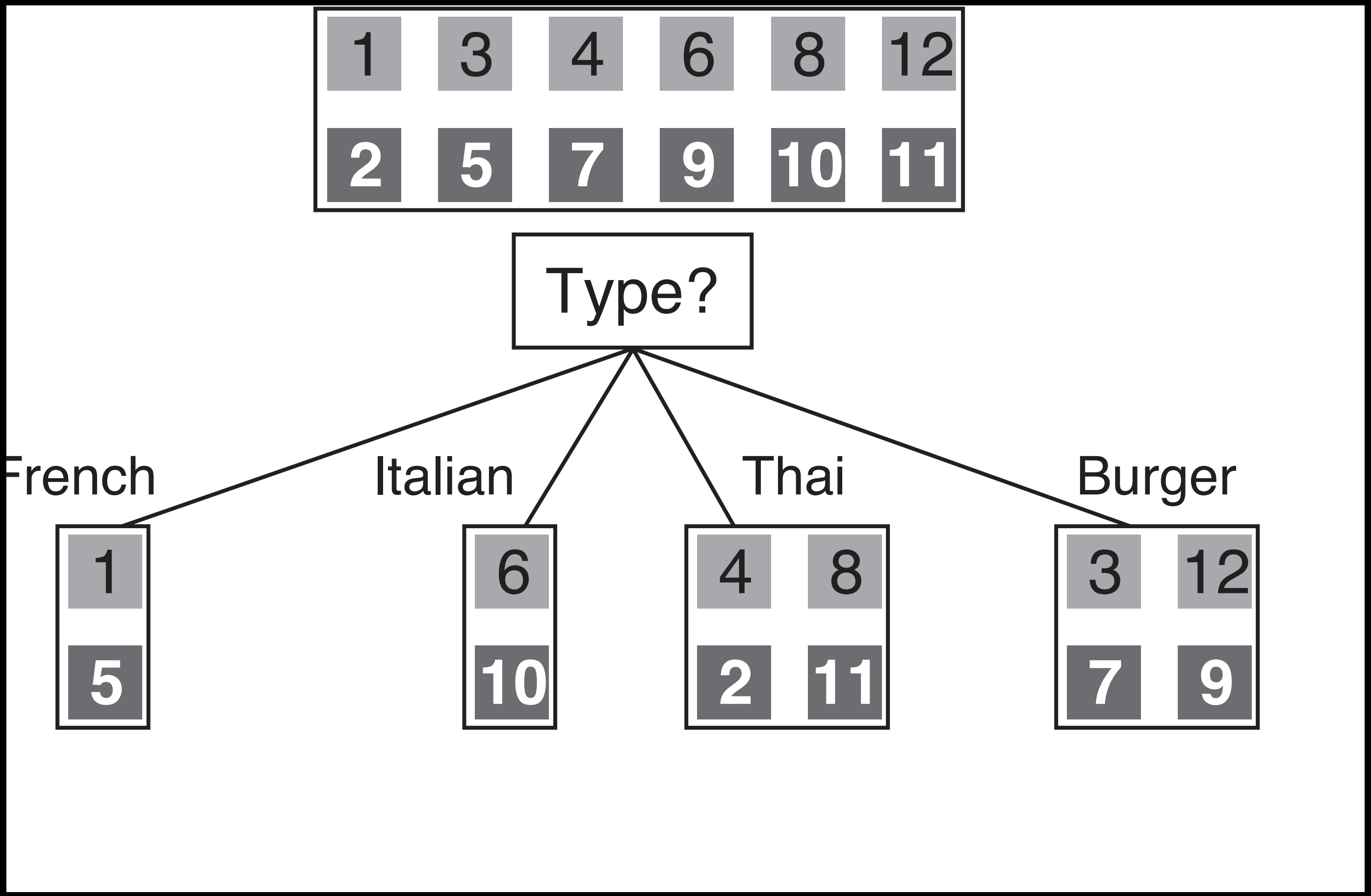
Decision Tree

- Each node in the tree represents a test on a single attribute
- Children of the node are labelled with the possible values of the feature
- Each path represents a series of tests, and the leaf node gives the value of the function when the input passes those tests

Inducing Decision Trees From Examples

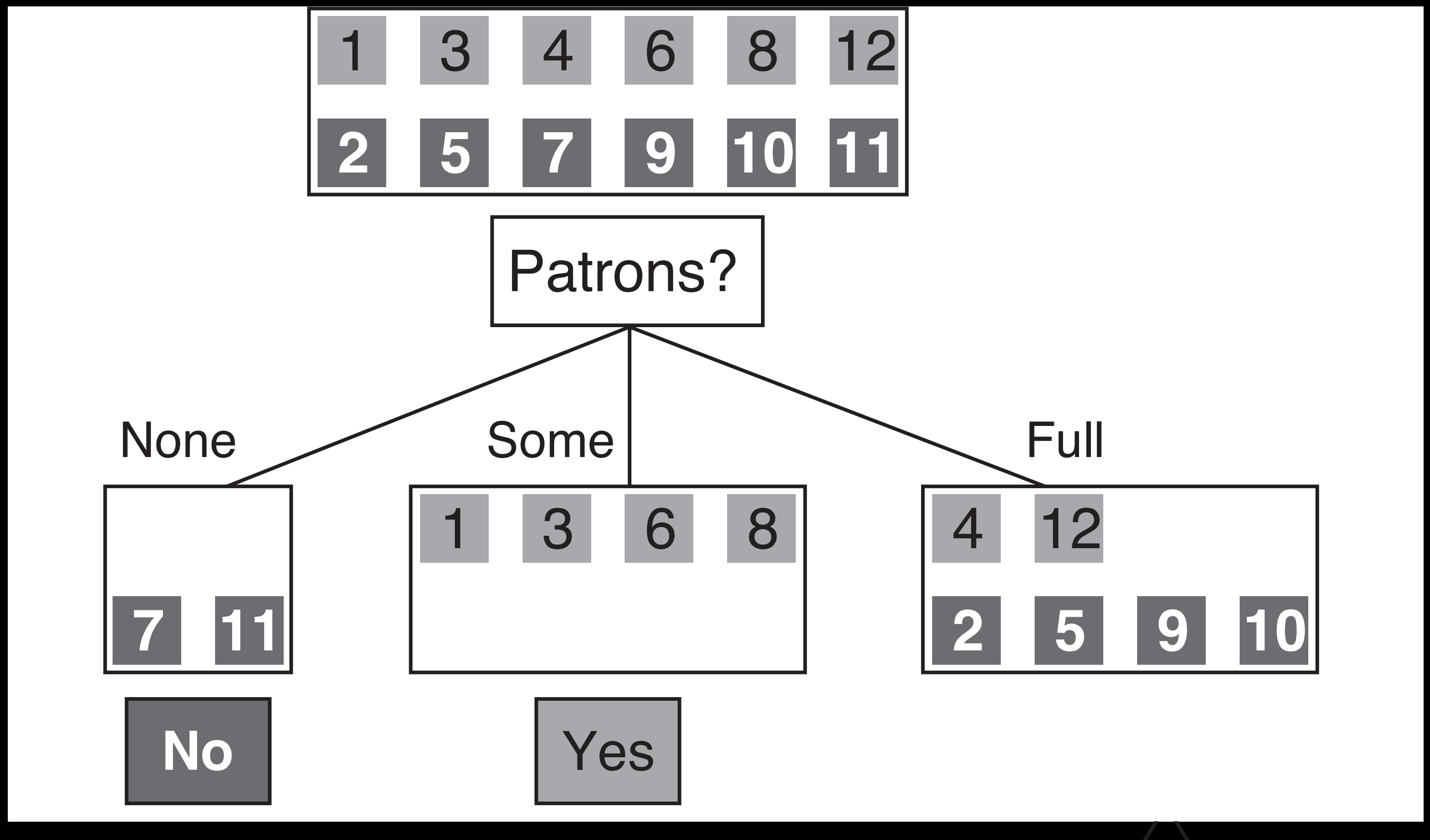
- Examples: (x, y)
- Want a shallow tree (short paths, fewer tests)
- Greedy algorithm (AIMA Fig 18.5)
  - Always test the most important attribute first
  - Makes the most difference to classification of an example

	Input Attributes										*Will Wait*
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	
x	*Yes*	*No*	*No*	*Yes*	*Some*	*$$$*	*No*	*Yes*	*French*	*0-10*	*y*
x	*Yes*	*No*	*No*	*Yes*	*Full*	*$*	*No*	*No*	*Thai*	*30-60*	*y*
x	*No*	*Yes*	*No*	*No*	*Some*	*$*	*No*	*No*	*Burger*	*0-10*	*y*
x	*Yes*	*No*	*Yes*	*Yes*	*Full*	*$*	*Yes*	*No*	*Thai*	*10-30*	*y*
x	*Yes*	*No*	*Yes*	*No*	*Full*	*$$$*	*No*	*Yes*	*French*	*>60*	*y*
x	*No*	*Yes*	*No*	*Yes*	*Some*	*$$*	*Yes*	*Yes*	*Italian*	*0-10*	*y*
x	*No*	*Yes*	*No*	*No*	*None*	*$*	*Yes*	*No*	*Burger*	*0-10*	*y*
x	*No*	*No*	*No*	*Yes*	*Some*	*$$*	*Yes*	*Yes*	*Thai*	*0-10*	*y*
x	*No*	*Yes*	*Yes*	*No*	*Full*	*$*	*Yes*	*No*	*Burger*	*>60*	*y*
x	*Yes*	*Yes*	*Yes*	*Yes*	*Full*	*$$$*	*No*	*Yes*	*Italian*	*10-30*	*y*
x	*No*	*No*	*No*	*No*	*None*	*$*	*No*	*No*	*Thai*	*0-10*	*y*
x	*Yes*	*Yes*	*Yes*	*Yes*	*Full*	*$*	*No*	*No*	*Burger*	*30-60*	*y*



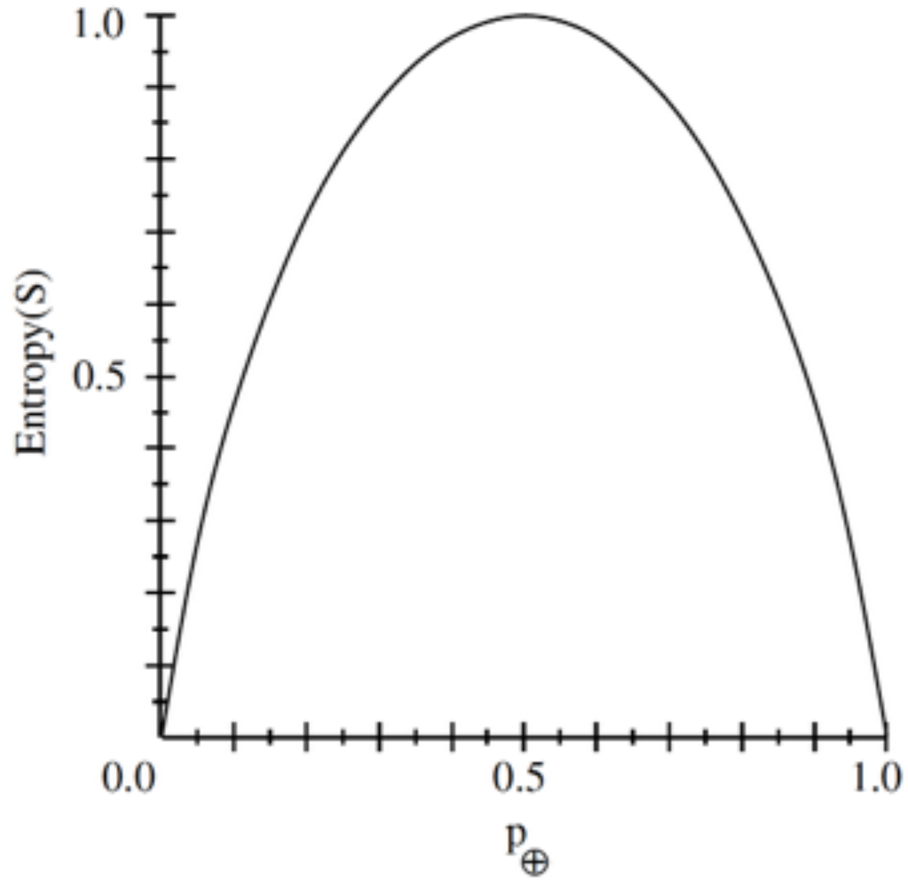
Poor split: children very mixed!

	Input Attributes										Will Wait
	Alt	Bar	Fri	Hun	Pat	Price	Rain	Res	Type	Est	
x	Yes	No	No	Yes	Some	$$$	No	Yes	French	0-10	y
x	Yes	No	No	Yes	Full	$	No	No	Thai	30-60	y
x	No	Yes	No	No	Some	$	No	No	Burger	0-10	y
x	Yes	No	Yes	Yes	Full	$	Yes	No	Thai	10-30	y
x	Yes	No	Yes	No	Full	$$$	No	Yes	French	>60	y
x	No	Yes	No	Yes	Some	$$	Yes	Yes	Italian	0-10	y
x	No	Yes	No	No	None	$	Yes	No	Burger	0-10	y
x	No	No	No	Yes	Some	$$	Yes	Yes	Thai	0-10	y
x	No	Yes	Yes	No	Full	$	Yes	No	Burger	>60	y
x	Yes	Yes	Yes	Yes	Full	$$$	No	Yes	Italian	10-30	y
x	No	No	No	No	None	$	No	No	Thai	0-10	y
x	Yes	Yes	Yes	Yes	Full	$	No	No	Burger	30-60	y



Good split: children very unbalanced!

Entropy



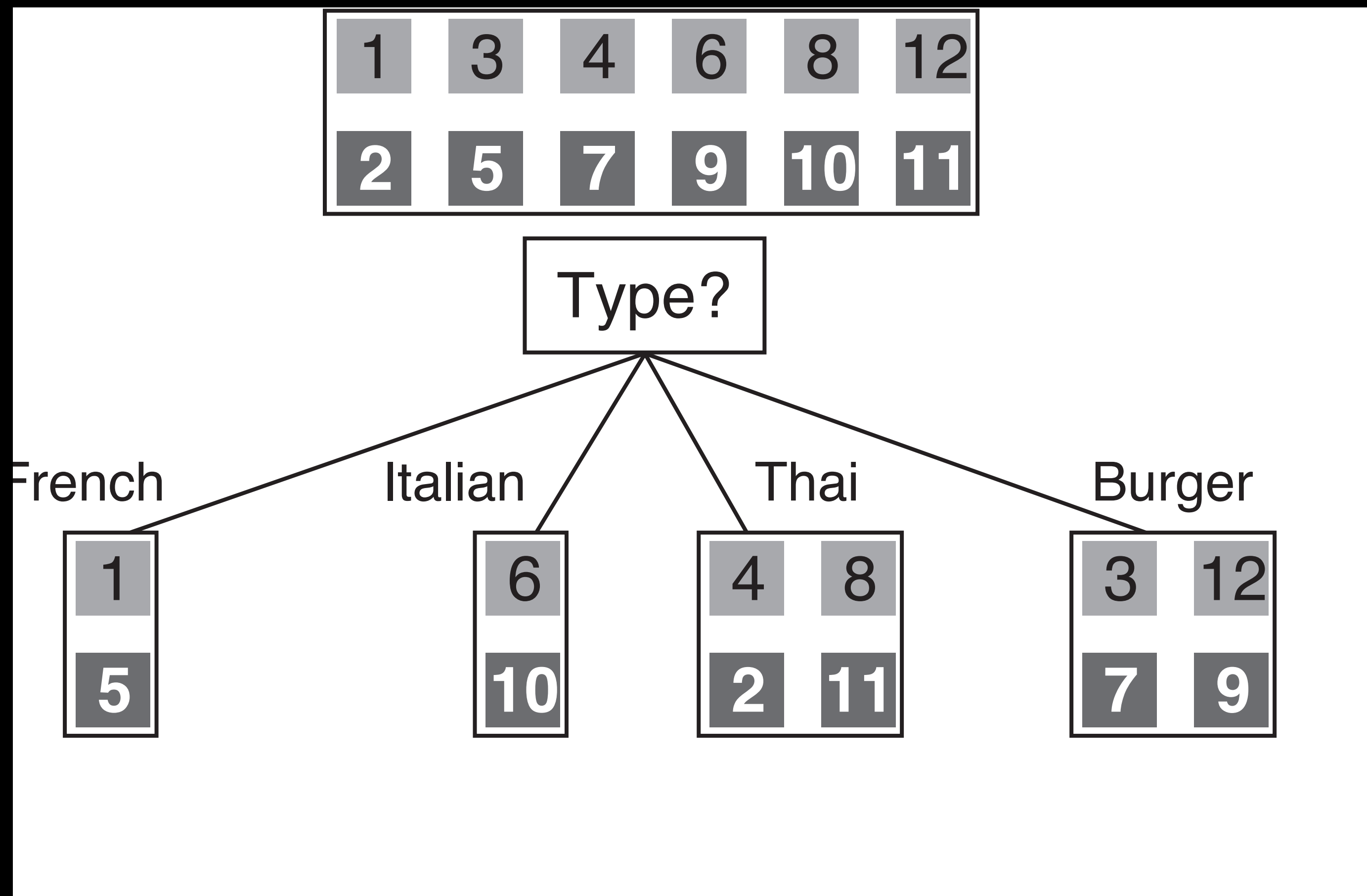
- S is a sample of training examples
- $p_{\oplus}$ is the proportion of positive examples in S
- $p_{\ominus}$ is the proportion of negative examples in S
- Entropy measures the impurity of S

$$Entropy(S) \equiv -p_{\oplus} \log_2 p_{\oplus} - p_{\ominus} \log_2 p_{\ominus}$$

Information Gain

$Gain(S, A)$ = expected reduction in entropy due to sorting on A

$$Gain(S, A) \equiv Entropy(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} Entropy(S_v)$$

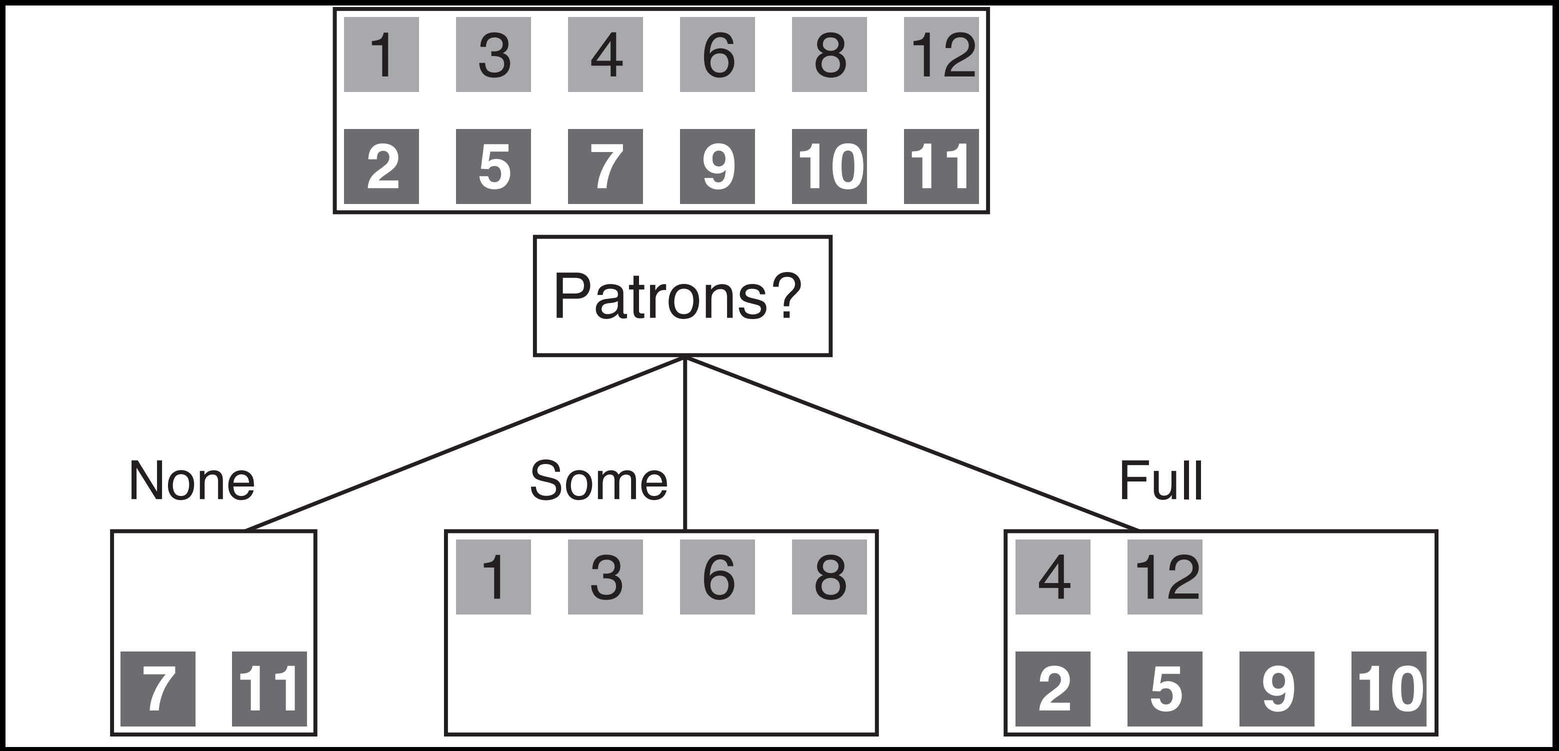


$$Entropy(S) = -0.5\log_2 0.5 - 0.5\log_2 0.5 = 1$$

$$Entropy(S_F) = Entropy(S_I) = Entropy(S_T) = Entropy(S_B) = 1$$

(a)

$$Gain(Type) = Entropy(S) - \sum_{v \in Type} \frac{|S_v|}{|S|} Entropy(S_v) = 1 - 1 = 0$$



$$Entropy(S) = -0.5\log_2 0.5 - 0.5\log_2 0.5 = 1$$

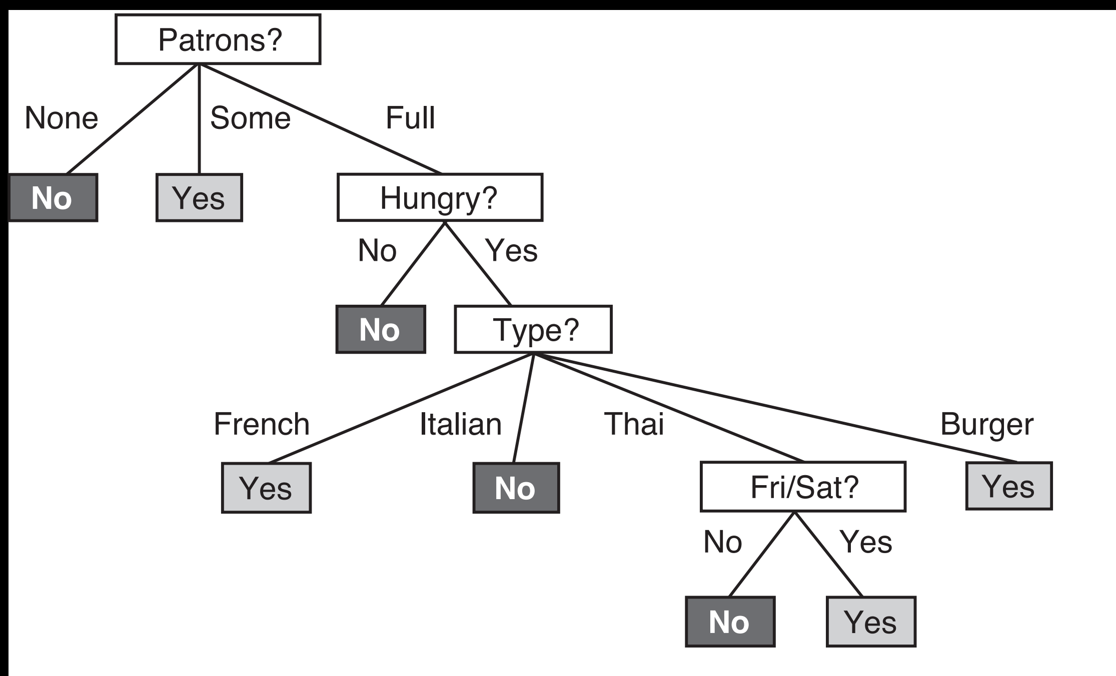
$$Entropy(S_N) = -0\log_2 0 - (1)\log_2 1 = 0$$

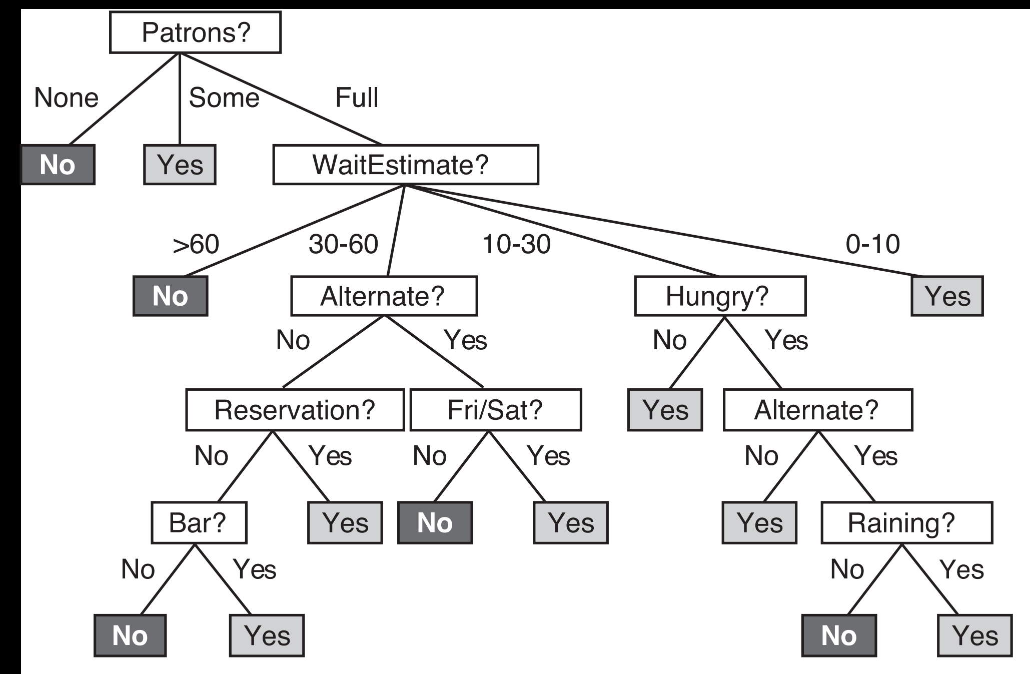
$$Entropy(S_S) = -(1)\log_2 1 - 0\log_2 0 = 0$$

$$Entropy(S_F) = -(1/3)\log_2 1/3 - 2/3\log_2 2/3 = 0.92$$

$$Gain(Patron) = 1 - \sum_{v \in Patron} \frac{|S_v|}{|S|} Entropy(S_v) = 1 - (1/2)(0.92) = 0.54$$

Avoiding Overfitting





- Problem: How to determine when to stop growing the decision tree?

Reduced-Error Pruning

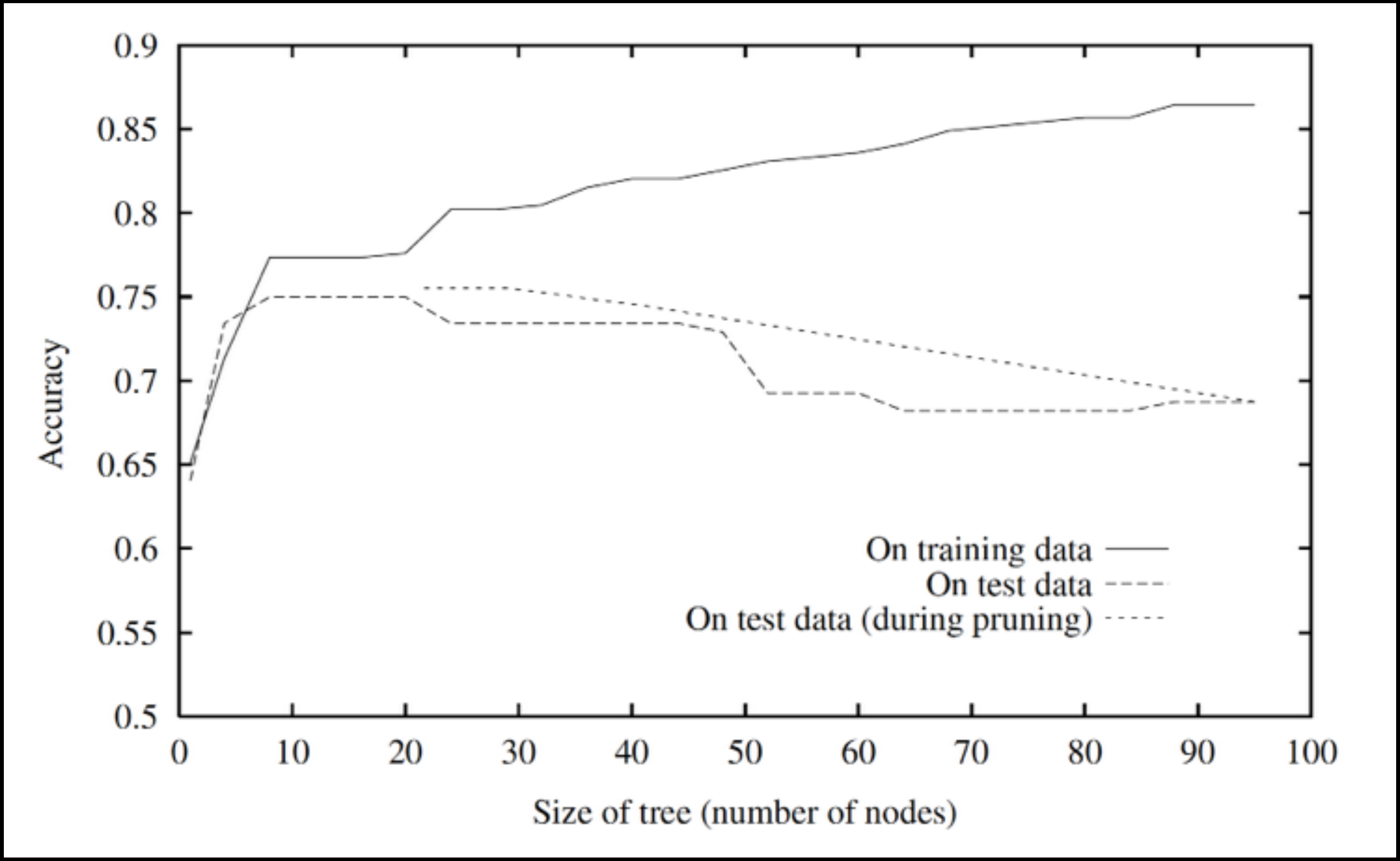
Split data into *training* and *validation* set

Do until further pruning is harmful:

1. Evaluate impact on *validation* set of pruning each possible node (plus those below it)
2. Greedily remove the one that most improves *validation* set accuracy

- produces smallest version of most accurate subtree

Effect of Reduced-Error Pruning



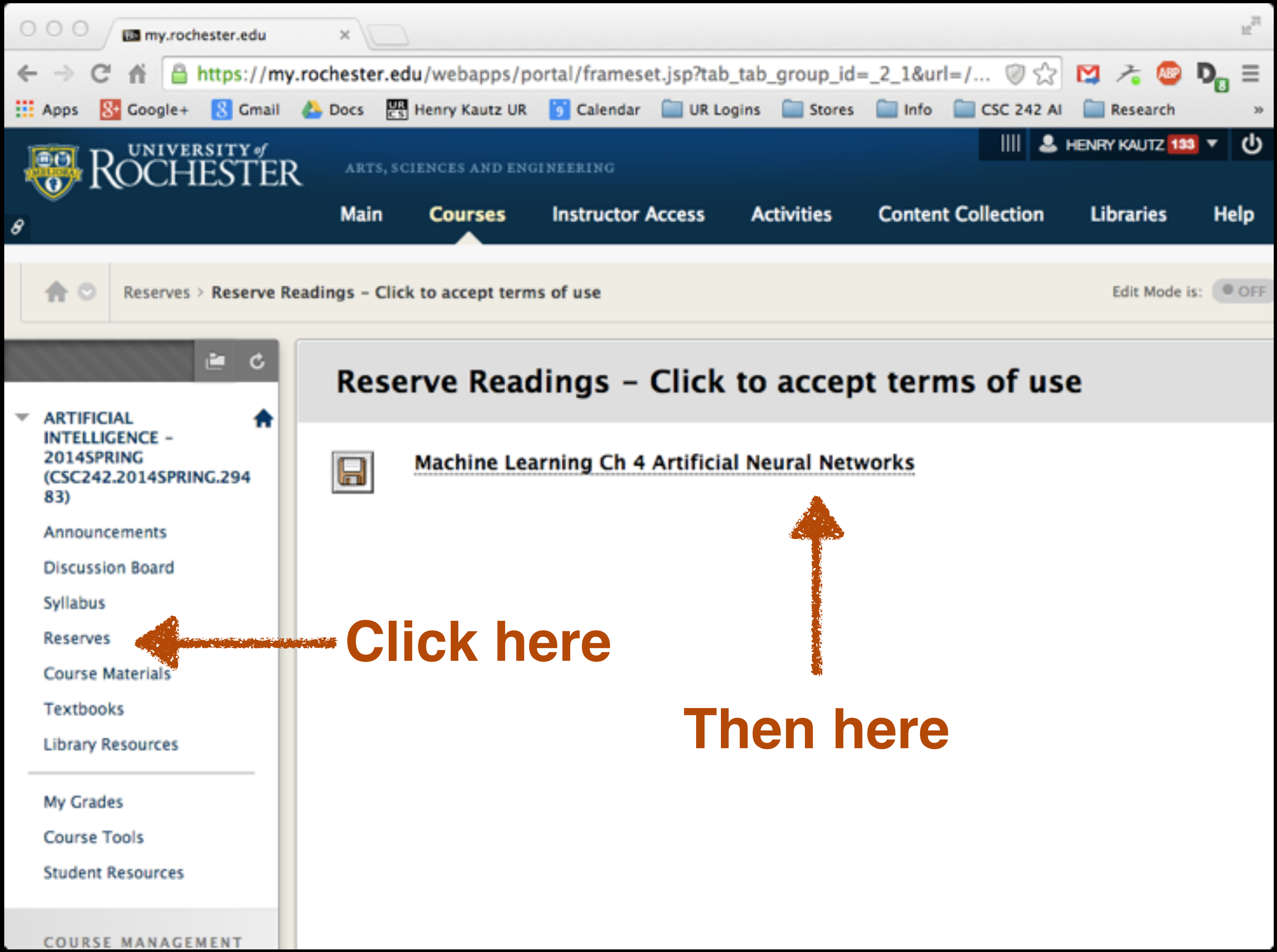
Learning
Neural Networks

Reserve Readings

Artificial Neural Networks

Chapter 4 of *Machine Learning*

by Tom Mitchell



Artificial Neural Networks

[Read Ch. 4]
[Recommended exercises 4.1, 4.2, 4.5, 4.9, 4.11]

- Threshold units
- Gradient descent
- Multilayer networks
- Backpropagation
- Hidden layer representations
- Example: Face Recognition
- Advanced topics

Connectionist Models

Consider humans:

- Neuron switching time ~ .001 second
- Number of neurons ~ 10^{10}
- Connections per neuron ~ 10^{4-5}
- Scene recognition time ~ .1 second
- 100 inference steps doesn't seem like enough

→ much parallel computation

Properties of artificial neural nets (ANN's):

- Many neuron-like threshold switching units
- Many weighted interconnections among units
- Highly parallel, distributed process
- Emphasis on tuning weights automatically

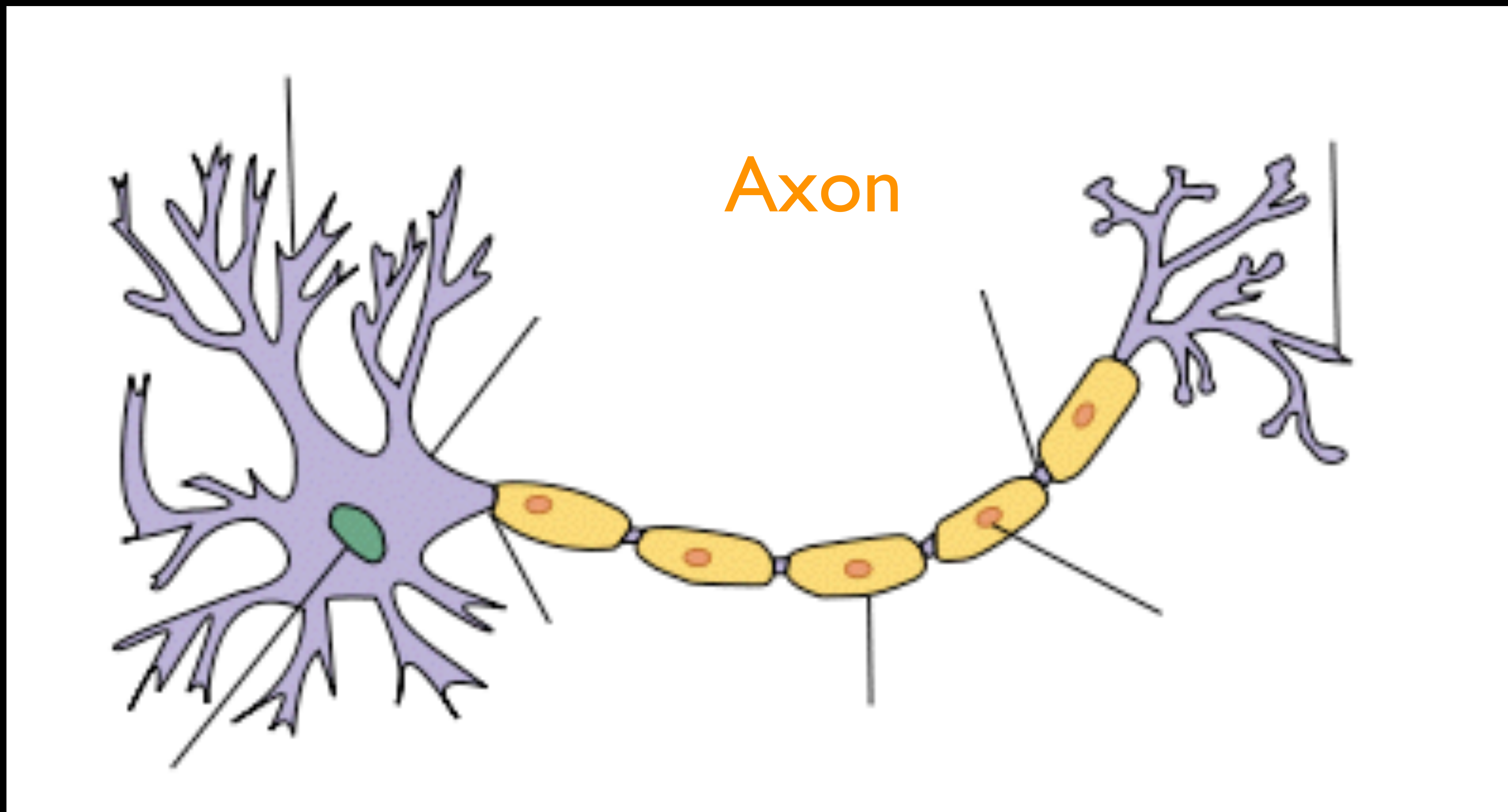
When to Consider Neural Networks

- Input is high-dimensional discrete or real-valued (e.g. raw sensor input)
- Output is discrete or real valued
- Output is a vector of values
- Possibly noisy data
- Form of target function is unknown
- Human readability of result is unimportant

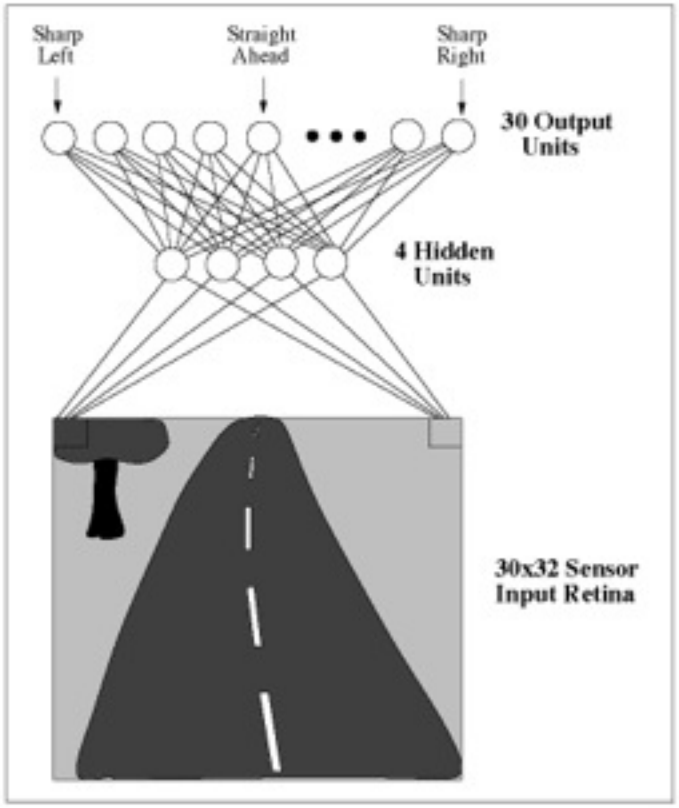
Examples:

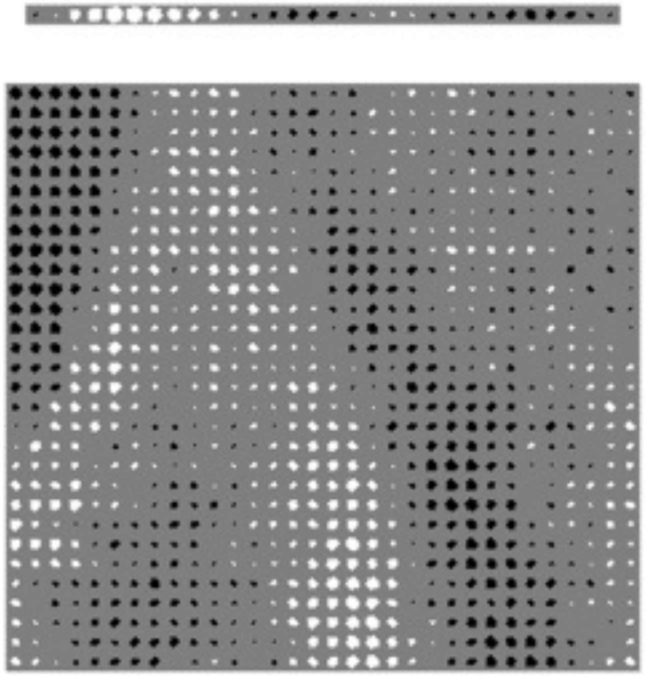
- Speech phoneme recognition [Waibel]
- Image classification [Kanade, Baluja, Rowley]
- Financial prediction

Dendrites

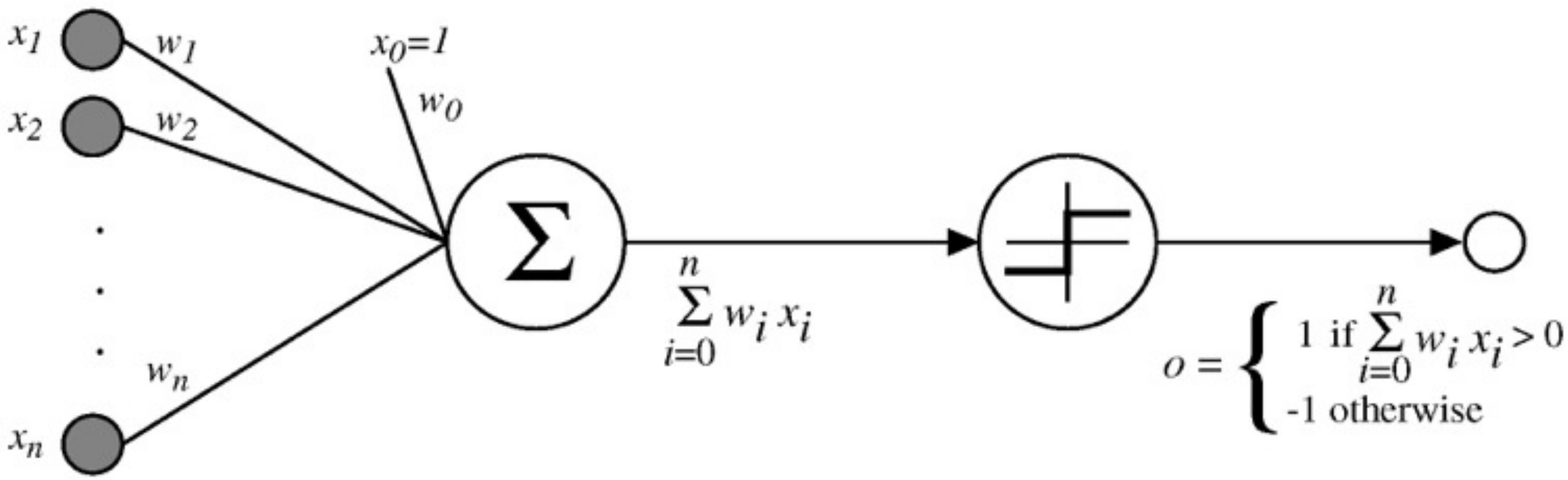


ALVINN drives 70 mph on highways





Perceptron



$$o(x_1, \ldots, x_n) = \begin{cases} 1 & \text{if } w_0 + w_1 x_1 + \cdots + w_n x_n > 0 \\ -1 & \text{otherwise.} \end{cases}$$

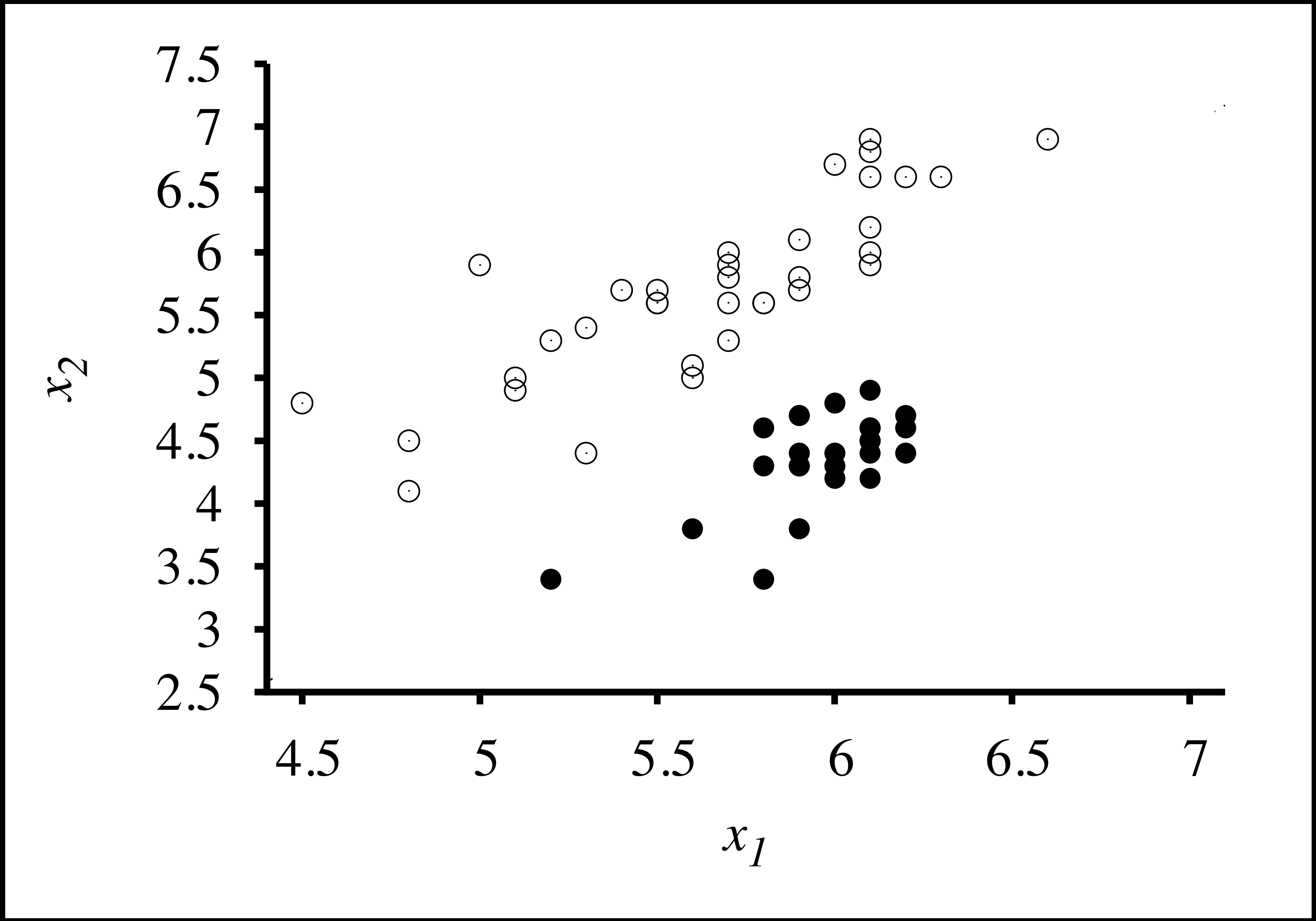
Sometimes we'll use simpler vector notation:

$$o(\vec{x}) = \begin{cases} 1 & \text{if } \vec{w} \cdot \vec{x} > 0 \\ -1 & \text{otherwise.} \end{cases}$$

What Do Perceptrons Do?

- To understand how perceptrons can be used to solve classification problems, we need to introduce the concept of a decision boundary

Earthquake or Atomic Bomb?

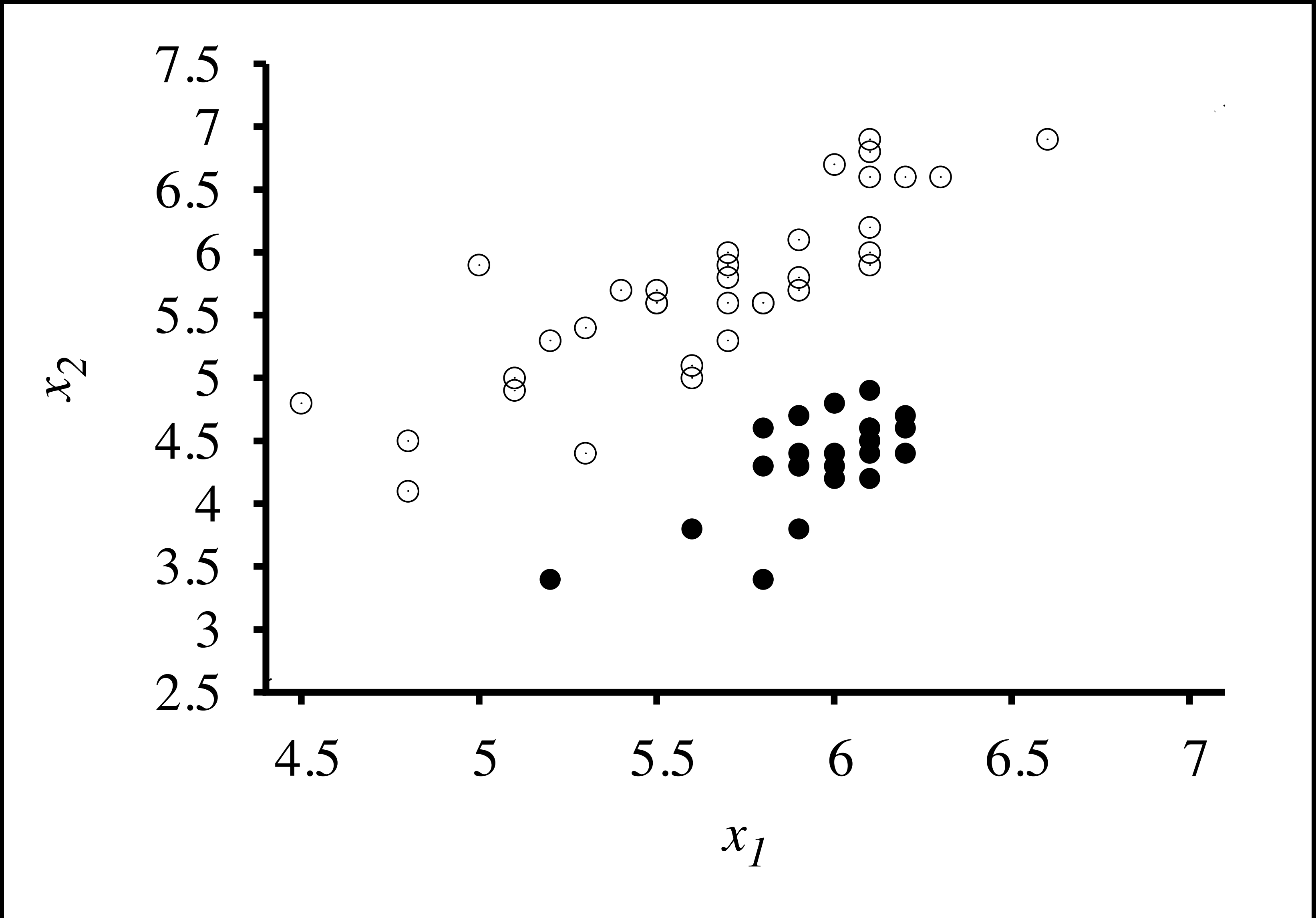


S-Wave

P-Wave


WORLD TARGETS IN MEGADEATHS
WAR ALERT ACTIONS BOOK

Earthquake or Atomic Bomb?



S-Wave

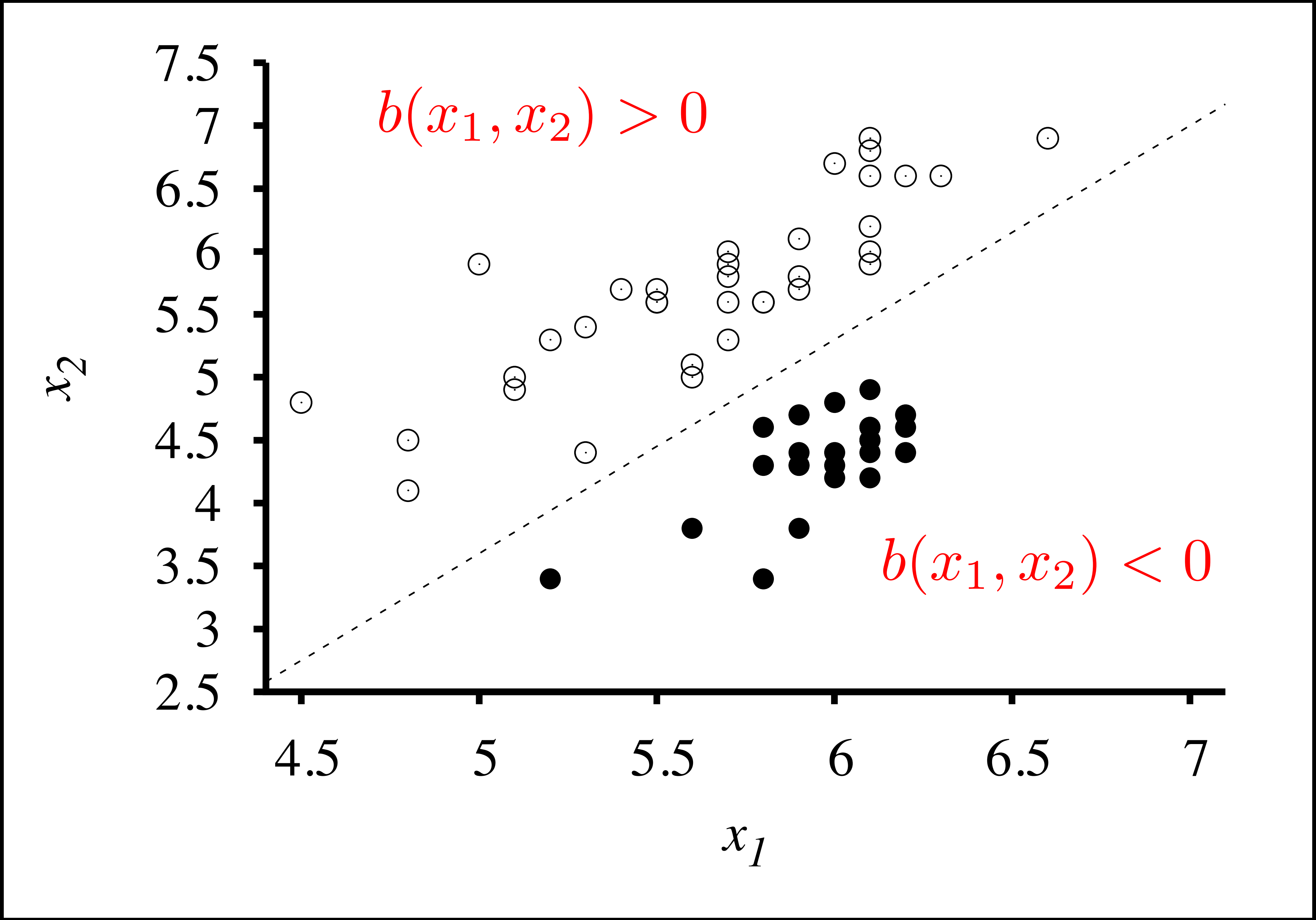
P-Wave

Decision Boundary

- Path (or surface in higher dimensions) that separates the two classes

$b(\mathbf{x_1},\mathbf{x_2})\ > 0$ if x is from an earthquake

< 0 if x is from an explosion



$$b(x_1, x_2) = x_2 - 1.7x_1 + 4.9$$

Linear Separator

- Decision boundary is a line
  - Line in 2D, plane in 3D, hyperplane in nD
- Data that admit a linear separator are said to be <u>linearly seperable</u>

Linear Classifier

$$w_0 + w_1 x_1 + w_2 x_2 = 0$$

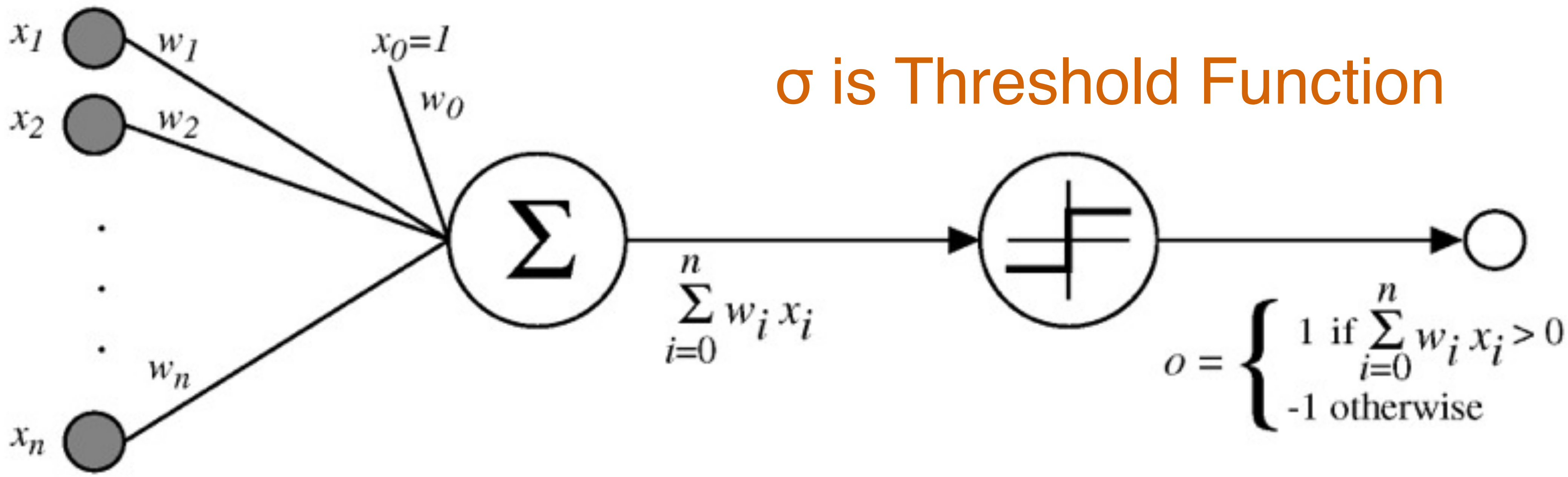
$$\mathbf{w} \cdot \mathbf{x} = 0$$

All instances of one class are above the line: $\mathbf{w} \cdot \mathbf{x} > 0$

All instances of one class are below the line: $\mathbf{w} \cdot \mathbf{x} < 0$

$$h_{\mathbf{w}}(\mathbf{x}) = Threshold(\mathbf{w} \cdot \mathbf{x})$$

Perceptron

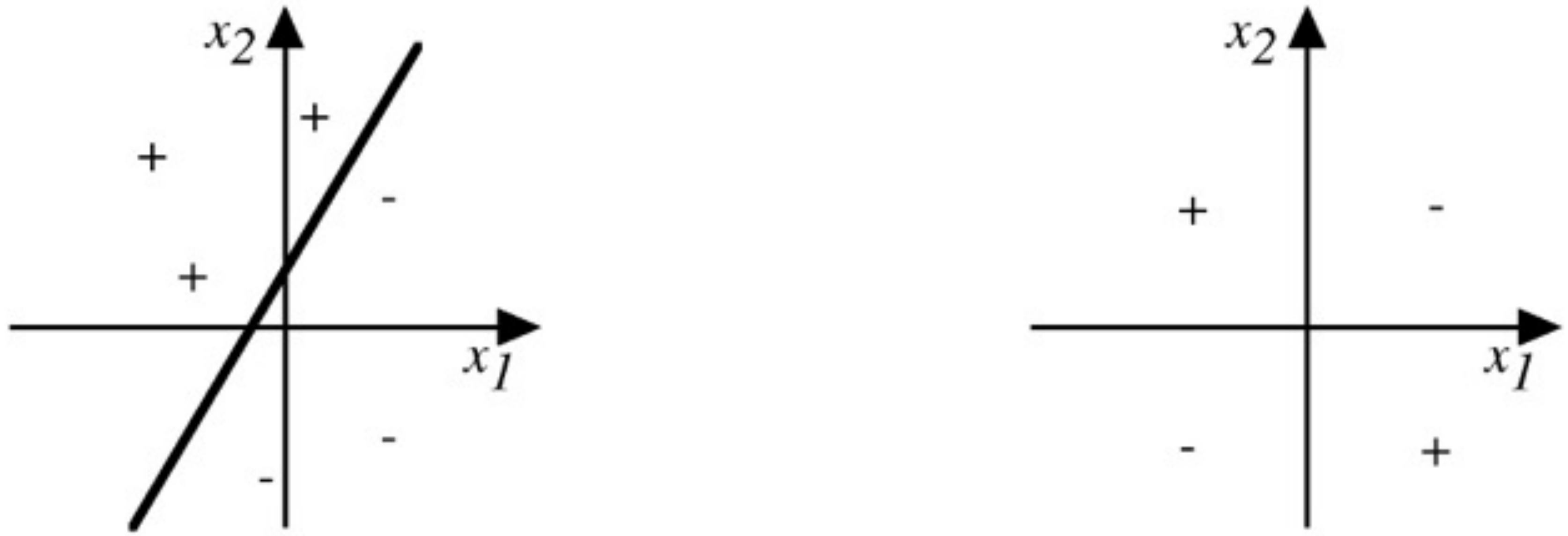


$$o(x_1, \ldots, x_n) = \begin{cases} 1 & \text{if } w_0 + w_1 x_1 + \cdots + w_n x_n > 0 \\ -1 & \text{otherwise.} \end{cases}$$

Sometimes we'll use simpler vector notation:

$$o(\vec{x}) = \begin{cases} 1 & \text{if } \vec{w} \cdot \vec{x} > 0 \\ -1 & \text{otherwise.} \end{cases}$$

Decision Surface of a Perceptron



(a) (b)

Represents some useful functions

- What weights represent
  $g(x_1, x_2) = AND(x_1, x_2)$?

But some functions not representable

- e.g., not linearly separable
- Therefore, we'll want networks of these...

Exercise

$$o(x_1, \ldots, x_n) = \begin{cases} 1 & \text{if } w_0 + w_1 x_1 + \cdots + w_n x_n > 0 \\ -1 & \text{otherwise.} \end{cases}$$

- Where true =1, false = -1, what is the perceptron for:
  - NOT(x_1)
  - AND(x_1,x_2)
  - OR(x_1,x_2)
  - XOR(x_1,x_2)

Exercise

$$o(x_1, \ldots, x_n) = \begin{cases} 1 & \text{if } w_0 + w_1 x_1 + \cdots + w_n x_n > 0 \\ -1 & \text{otherwise.} \end{cases}$$

- Where true =1, false = -1, what is the perceptron for:
  - NOT(x_1) = $\sigma((-1)x_1)$
  - AND(x_1,x_2)
  - OR(x_1,x_2)
  - XOR(x_1,x_2)

Exercise

$$o(x_1, \ldots, x_n) = \begin{cases} 1 & \text{if } w_0 + w_1 x_1 + \cdots + w_n x_n > 0 \\ -1 & \text{otherwise.} \end{cases}$$

- Where true =1, false = -1, what is the perceptron for:
  - NOT(x_1) = $\sigma((-1)x_1)$
  - AND(x_1,x_2) = $\sigma(x_1 + x_2 - 1.5)$
  - OR(x_1,x_2)
  - XOR(x_1,x_2)

Exercise

$$o(x_1, \ldots, x_n) = \begin{cases} 1 & \text{if } w_0 + w_1 x_1 + \cdots + w_n x_n > 0 \\ -1 & \text{otherwise.} \end{cases}$$

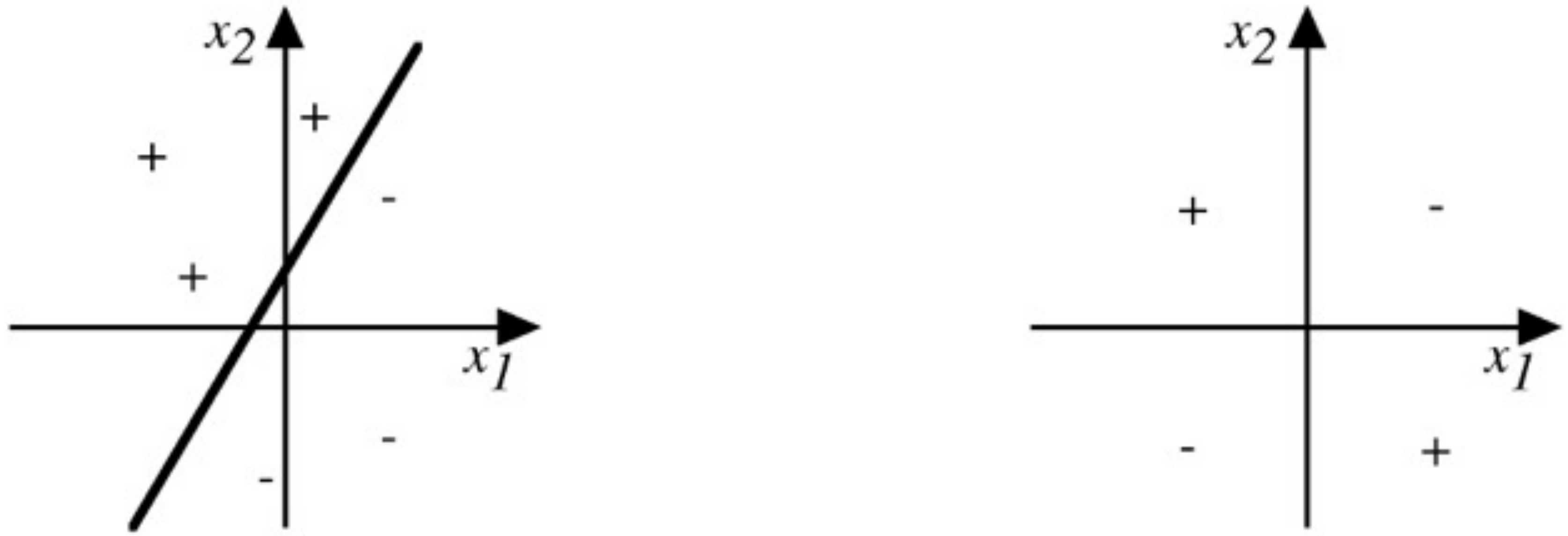
- Where true =1, false = -1, what is the perceptron for:
  - NOT(x_1) = $\sigma((-1)x_1)$
  - AND(x_1,x_2) = $\sigma(x_1 + x_2 - 1.5)$
  - OR(x_1,x_2) = $\sigma(x_1 + x_2 + 0.5)$
  - XOR(x_1,x_2)

Exercise

$$o(x_1, \ldots, x_n) = \begin{cases} 1 & \text{if } w_0 + w_1 x_1 + \cdots + w_n x_n > 0 \\ -1 & \text{otherwise.} \end{cases}$$

- Where true =1, false = -1, what is the perceptron for:
  - NOT(x_1) = o((-1)x_1)
  - AND(x_1,x_2) = o(x_1 + x_2 - 1.5)
  - OR(x_1,x_2) = o(x_1 + x_2 + 0.5)
  - XOR(x_1,x_2) NO SOLUTION!

Decision Surface of a Perceptron



(a) (b)

Represents some useful functions

- What weights represent $g(x_1, x_2) = AND(x_1, x_2)$?

But some functions not representable

- e.g., not linearly separable
- Therefore, we'll want networks of these...

Training

- Training is using data to set the weights for a perceptron (or network of perceptrons)
- Idea:
  - Start with random weights
  - For each piece of data:
    - Set inputs to the data features
    - Compare output to the label (target value)
    - If not same then adjust the weights

Perceptron training rule

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

where

$$\Delta w_i = \eta(t - o)x_i$$

Where:

- $t = c(\vec{x})$ is target value
- o is perceptron output
- η is small constant (e.g., .1) called *learning rate*

Gradient Descent

GRADIENT-DESCENT($training\_examples, \eta$)

Each training example is a pair of the form $\langle \vec{x}, t \rangle$, where $\vec{x}$ is the vector of input values, and t is the target output value. η is the learning rate (e.g., .05).

- Initialize each w_i to some small random value
- Until the termination condition is met, Do
  - Initialize each Δw_i to zero.
  - For each $\langle \vec{x}, t \rangle$ in $training\_examples$, Do
    - Input the instance $\vec{x}$ to the unit and compute the output o
    - For each linear unit weight w_i, Do

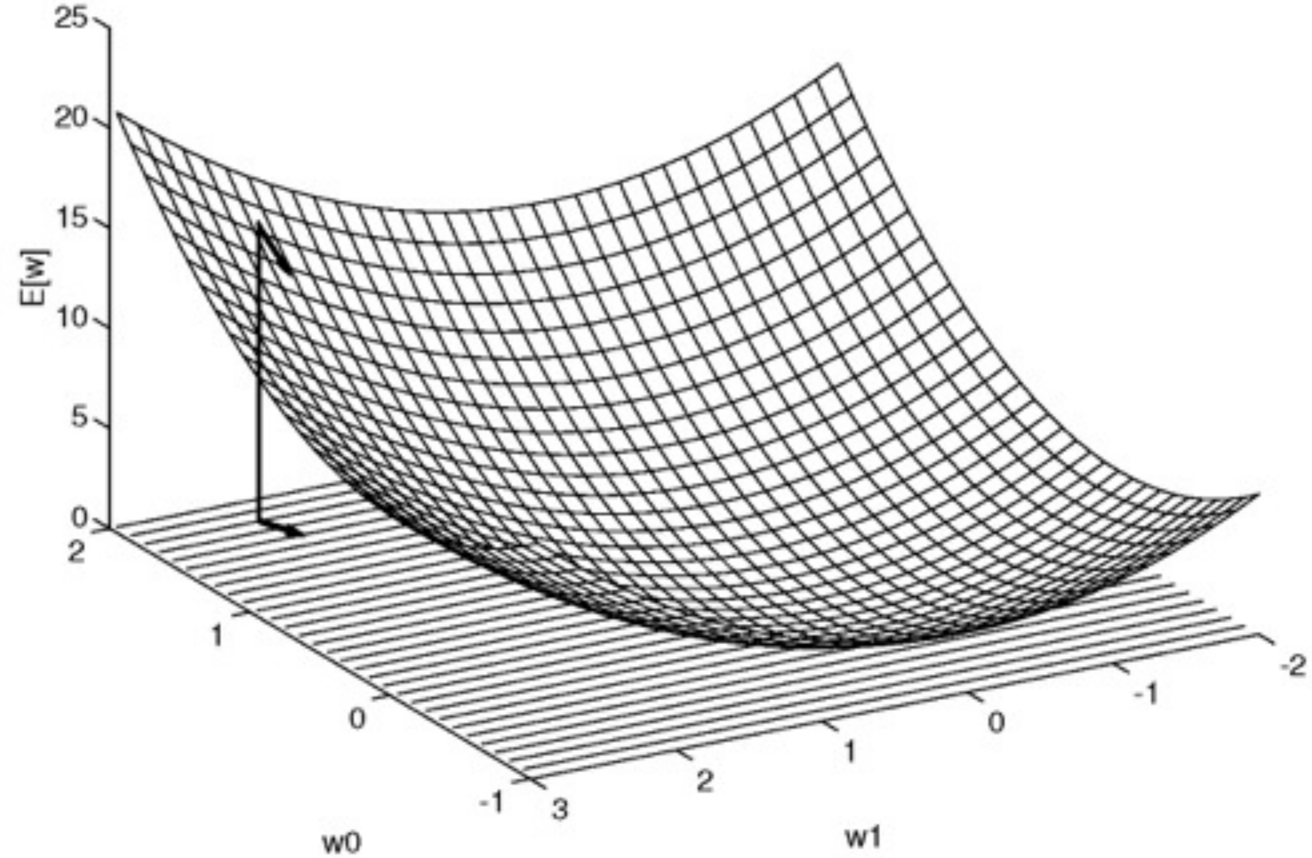
$$\Delta w_i \leftarrow \Delta w_i + \eta (t - o) x_i$$

  - For each linear unit weight w_i, Do

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

Justifying the Training Rule

- Define the error E as the sum of squared differences between the outputs and the targets across the training set
- Goal: find weights that minimize E
- Gradient descent: Repeat:
  - Compute the slope (gradient) of E with respect to each of the current weights
  - Make a small change in the weights in the "downward" direction



Gradient

$$\nabla E[\vec{w}] \equiv \left[\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \cdots \frac{\partial E}{\partial w_n}\right]$$

Training rule:

$$\Delta \vec{w} = -\eta \nabla E[\vec{w}]$$

i.e.,

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i}$$

Deriving Training Rule (Ignoring Threshold Function σ)

$$
\begin{aligned}
\frac{\partial E}{\partial w_i} &= \frac{\partial}{\partial w_i}\frac{1}{2}\sum_d (t_d - o_d)^2 \\
&= \frac{1}{2}\sum_d \frac{\partial}{\partial w_i}(t_d - o_d)^2 \\
&= \frac{1}{2}\sum_d 2(t_d - o_d)\frac{\partial}{\partial w_i}(t_d - o_d) \\
&= \sum_d (t_d - o_d)\frac{\partial}{\partial w_i}(t_d - \vec{w}\cdot\vec{x_d}) \\
\frac{\partial E}{\partial w_i} &= \sum_d (t_d - o_d)(-x_{i,d})
\end{aligned}
$$

Incremental (Stochastic) Gradient Descent

Batch mode Gradient Descent:
Do until satisfied

1. Compute the gradient $\nabla E_D[\vec{w}]$
2. $\vec{w} \leftarrow \vec{w} - \eta \nabla E_D[\vec{w}]$

Incremental mode Gradient Descent:
Do until satisfied

- For each training example d in D
  1. Compute the gradient $\nabla E_d[\vec{w}]$
  2. $\vec{w} \leftarrow \vec{w} - \eta \nabla E_d[\vec{w}]$

$$E_D[\vec{w}] \equiv \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2$$

$$E_d[\vec{w}] \equiv \frac{1}{2} (t_d - o_d)^2$$

Incremental Gradient Descent can approximate *Batch Gradient Descent* arbitrarily closely if η made small enough

Summary

Perceptron training rule guaranteed to succeed if

- Training examples are linearly separable
- Sufficiently small learning rate η

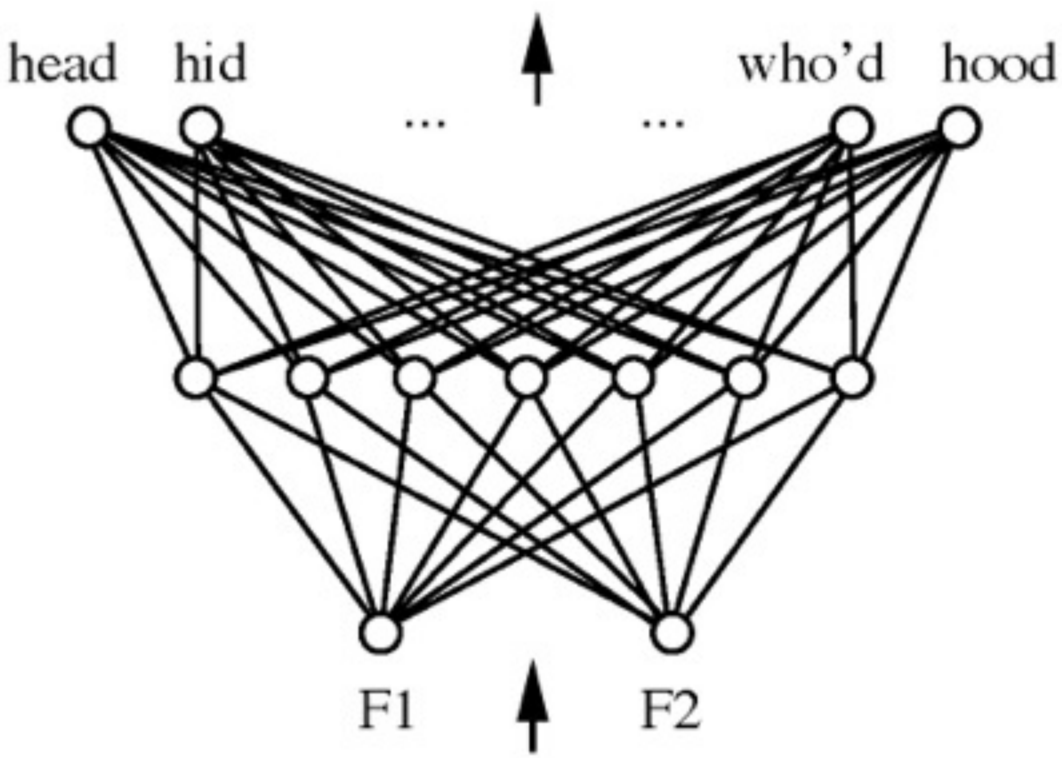
Linear unit training rule uses gradient descent

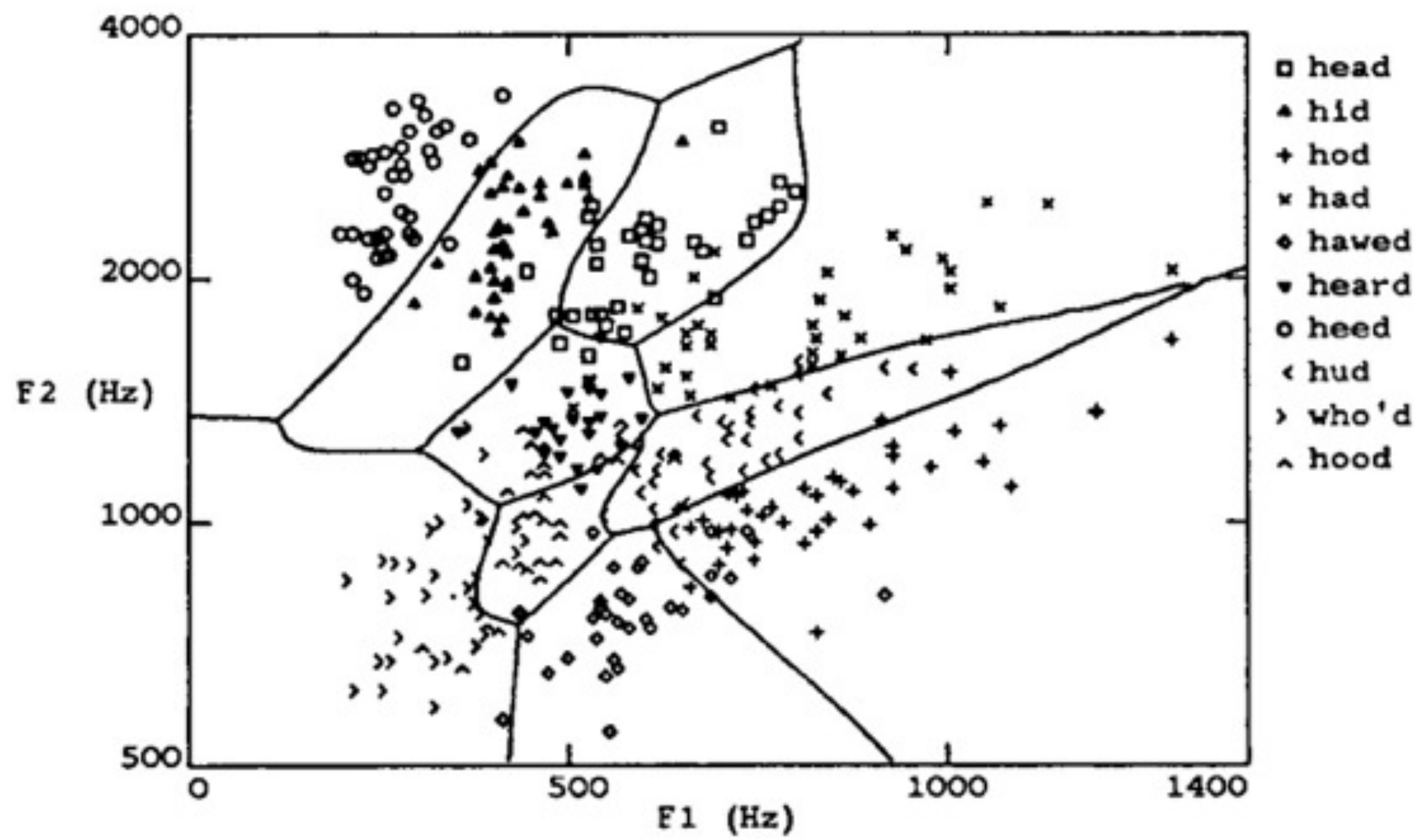
- Guaranteed to converge to hypothesis with minimum squared error
- Given sufficiently small learning rate η
- Even when training data contains noise
- Even when training data not separable by H

Epochs

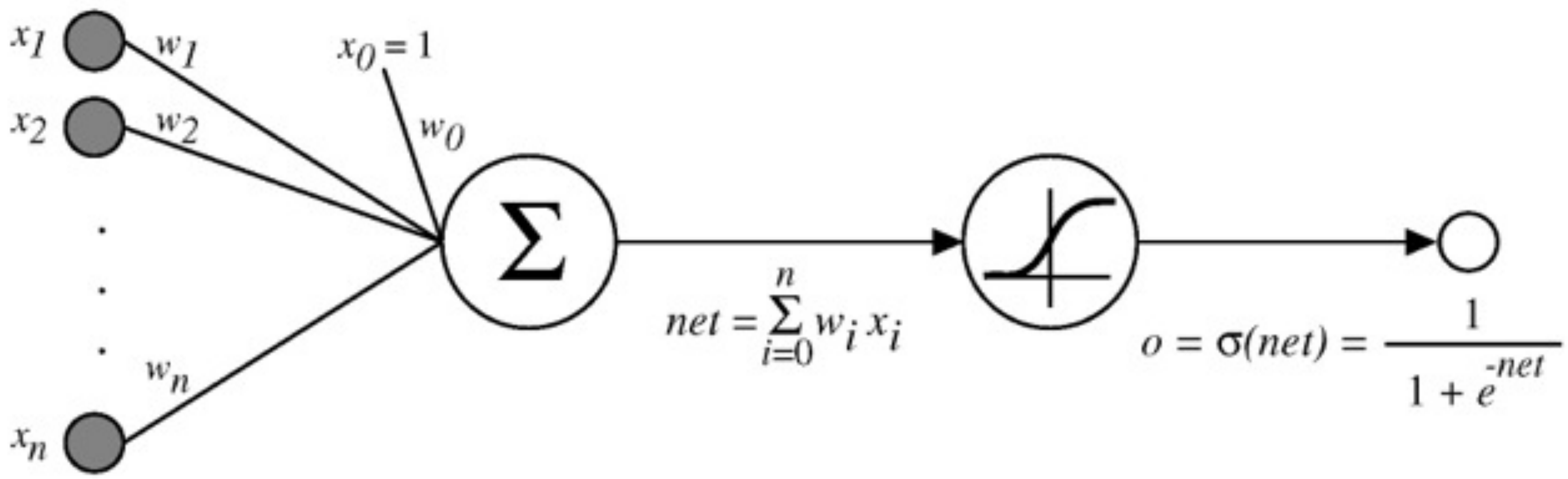
- It can take a lot of small steps to reach the optimal set of weights
- What if you run through all the training data and are not yet at the optimum?
- Run through the training data again …
- … and again …
- … and again!
- Each pass through the training data is an epoch

Multilayer Networks of Sigmoid Units





Sigmoid Unit



$\sigma(x)$ is the sigmoid function

$$\frac{1}{1 + e^{-x}}$$

Nice property: $\frac{d\sigma(x)}{dx} = \sigma(x)(1 - \sigma(x))$

We can derive gradient decent rules to train

- One sigmoid unit
- *Multilayer networks* of sigmoid units $\rightarrow$ Backpropagation

Coming Up

- April 15 - Neural Network II
  - Back by Popular Demand!
  - Even better than Neural Networks I!
- April 17 - In-Class Workshop for Project 3
  - Live highly attractive TAs will personally help you complete the project!
  - An afternoon you will not soon forget!