SYMBOLIC SYSTEMS 100: Introduction to Cognitive Science

Dan Jurafsky and Daniel Richardson
Stanford University
Spring 2005

May 24, 2005: Neural Networks and Machine Learning

IP Notice: Slides stolen shamelessly from all sorts of people including Jim Martin, Frank Keller, Greg Grudick, Ricardo Vilalta, Mateen Rizki, cprogramming.com, and others.

Outline

- Neural networks
  - McCulloch Pitts Neuron
  - Perceptron
  - Delta rule
  - Error Back Propagation
- Machine learning

Neural networks history

- 1943: McCulloch Pitts simplified model of the neuron as a computing element
- Described in terms of propositional logic
- Inspired by work of Turing
- In turn, inspired work by Kleene (1951) on finite automata and regular expressions.
- Not trained (no learning mechanism)

Neural networks history

- Hebbian Learning (1949)
  - Concept that information is stored in the connections
  - Learning rule for adjusting synaptic connections
- 1958: Perceptron (Rosenblatt)
  - Weight neural inputs with a learning rule
- 1960: Adaline (Widrow Hoff 1960 at stanford):
  - adaptive linear elemnt with a learning rule
- 1969: Minsky and Papert show problems with perceptrons
  - Famous "XOR" problem

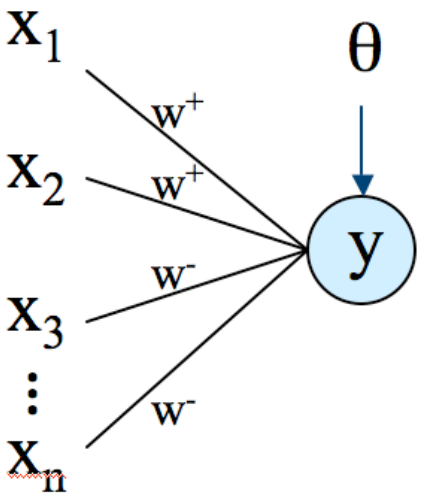
Neural networks history

- 1974-1986 Various people solve the problems with perceptrons:
  - Algorithms for training feedforward multilayered perceptrons
  - Error Back Propagation (Rumelhart et al 1986)
- 1990: Support Vector Machines
- Current: neural networks seen as just one of many tools for machine learning.

McCulloch-Pitts Neuron

- 1943
- Neuron produces a binary output (0/1)
- A specific number of inputs must be excited to fire
- Any nonzero inhibatory input prevents firing
- Fixed network structure (no learning)

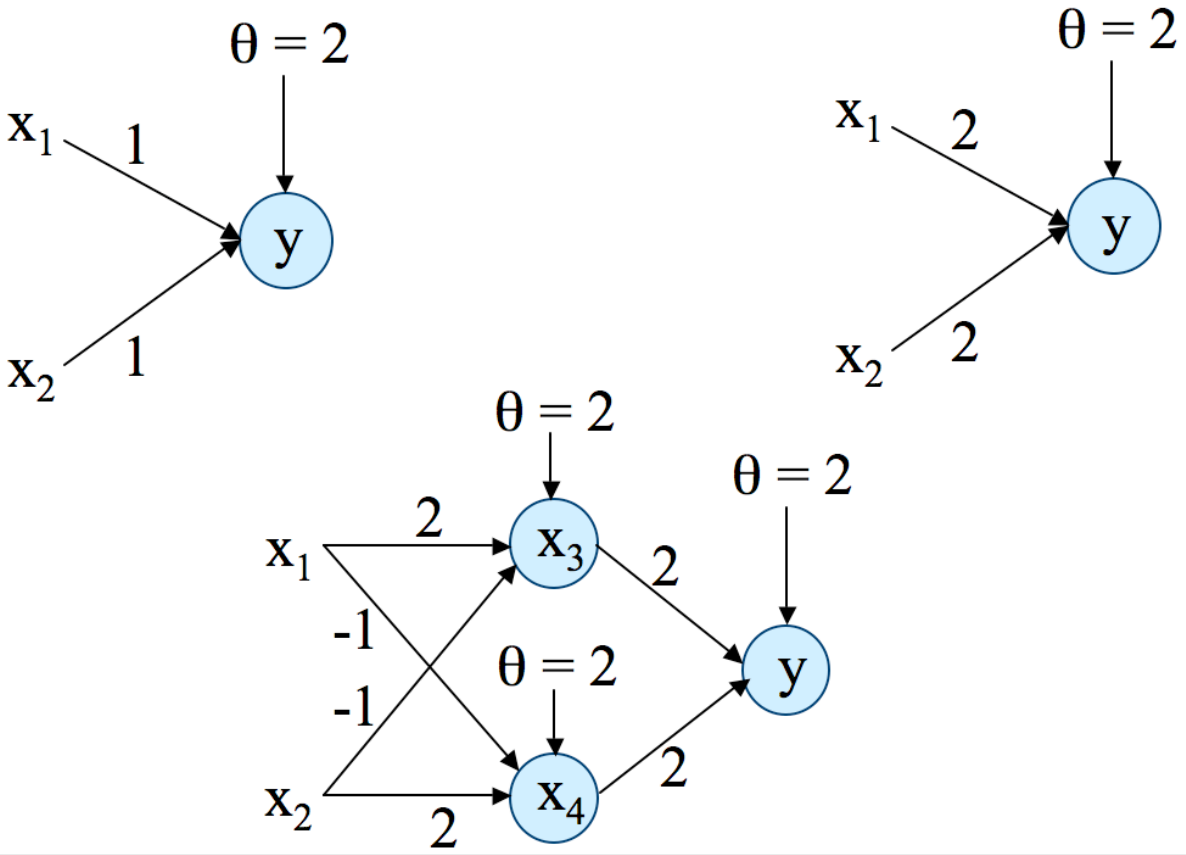
McCulloch-Pitts Neuron



x_i - input
w^+ - excitatory input ($w > 0$)
w^- - inhibitory input ($w < 0$)
θ - firing threshold
y - output

$y = 1$ if sum of excitatory inputs $>= \theta$ and no inhibitory input
$y = 0$ if sum of excitatory inputs $< \theta$ or inhibitory input

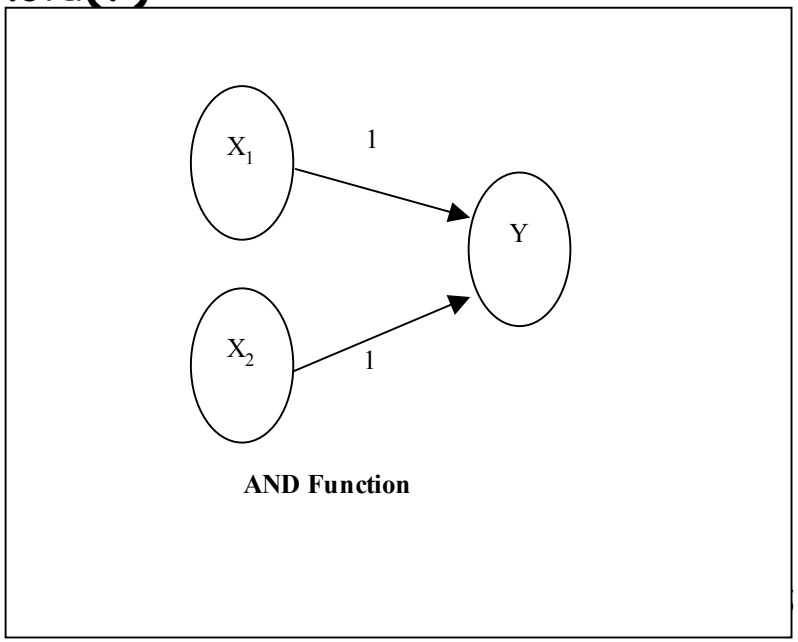
MP Neuron examples



MP Example 1

- **Logic Functions: AND**
- **True=1, False=0**
- **If both inputs true, output true**
- **Else, output false**
- **Threshold(Y)=2**

x1	x2	AND
0	0	0
0	1	0
1	0	0
1	1	1

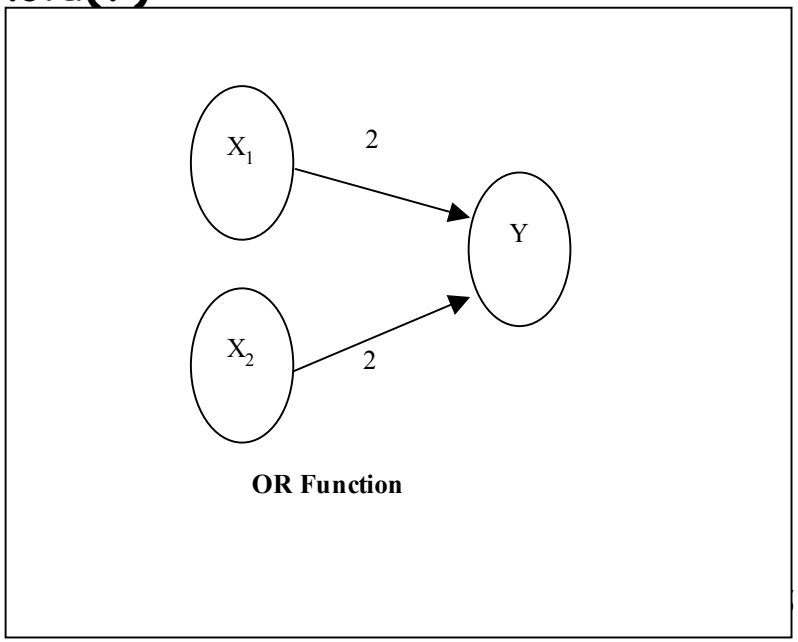


AND Function

MP Example 2

- **Logic Functions: OR**
- **True=1, False=0**
- **If either of inputs true, output true**
- **Else, output false**
- **Threshold(Y)=2**

x1	x2	OR
0	0	0
0	1	1
1	0	1
1	1	1

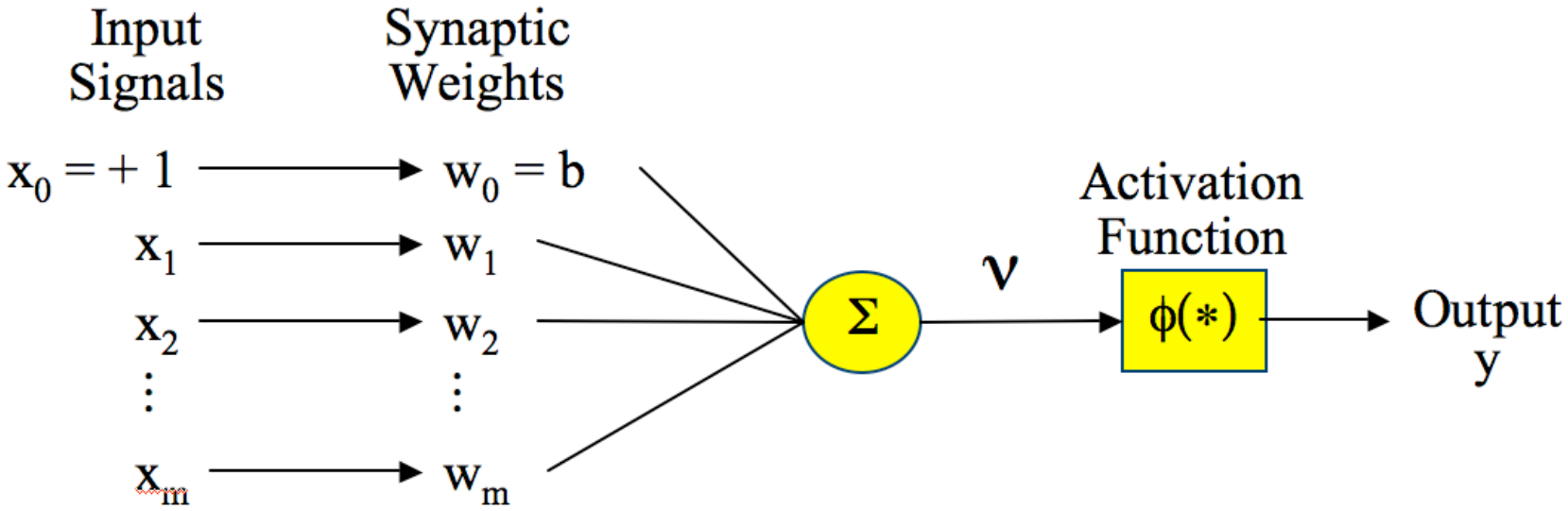


OR Function

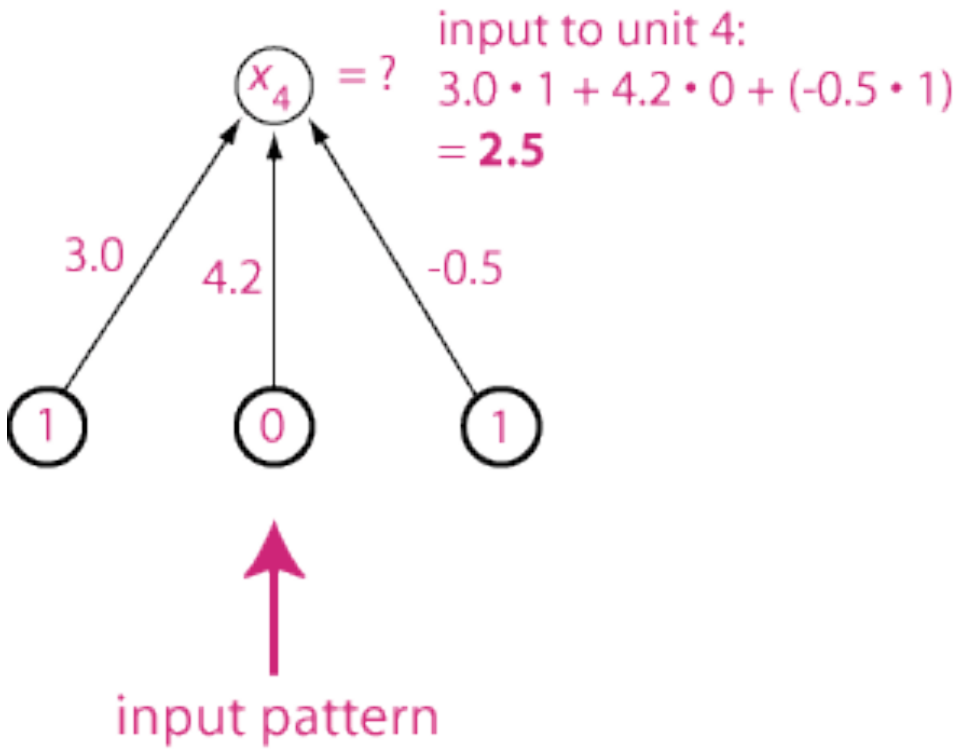
Problems with MP neuron

- Only models binary input
- Structure doesn't change
- Weights are set by hand
  - No learning!!
- But nonetheless is basis for all future work on neural nets

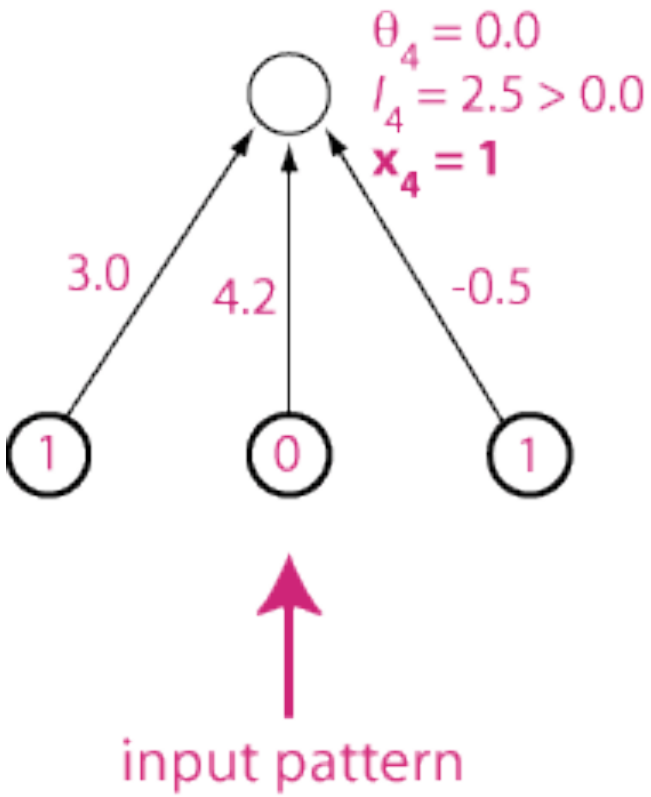
Perceptrons



$$\nu = \sum_{j=0}^{m} w_j x_j \Rightarrow \nu = \vec{x}^T \cdot \vec{w} \qquad y = \phi_{lin}(\nu)$$

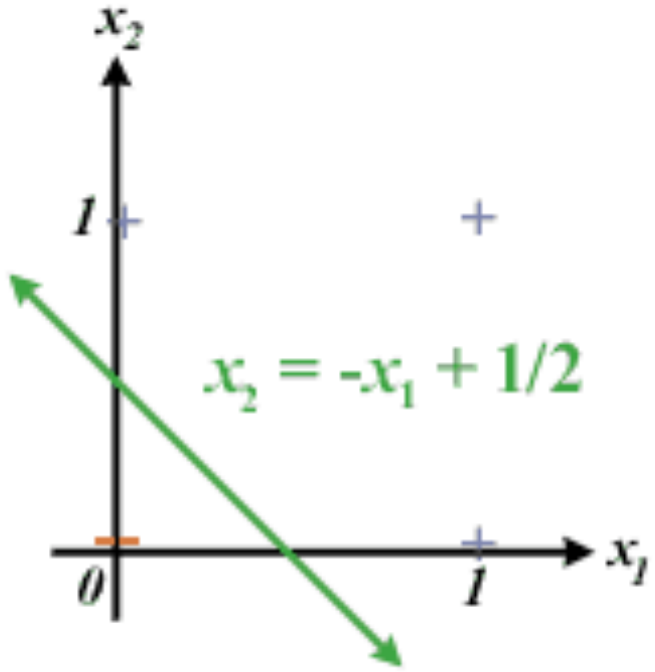

input to unit 4:
3.0 • 1 + 4.2 • 0 + (-0.5 • 1)
= 2.5
x_4
= ?
3.0
4.2
-0.5
1
0
1
input pattern

Adding a threshold ("Squashing function")



A graphical metaphor

- If you graph the possible inputs
  - on different axes
  - With pluses for firing
  - And minus for not firing
  - The weights for the perceptron make up the equation of a line that separates the pluses and the minuses



$$w_1 x_1 + w_2 x_2 + w_3 x_3 + \ldots + w_n x_n + \ldots + w_b b = \theta$$

Problems with Perceptrons

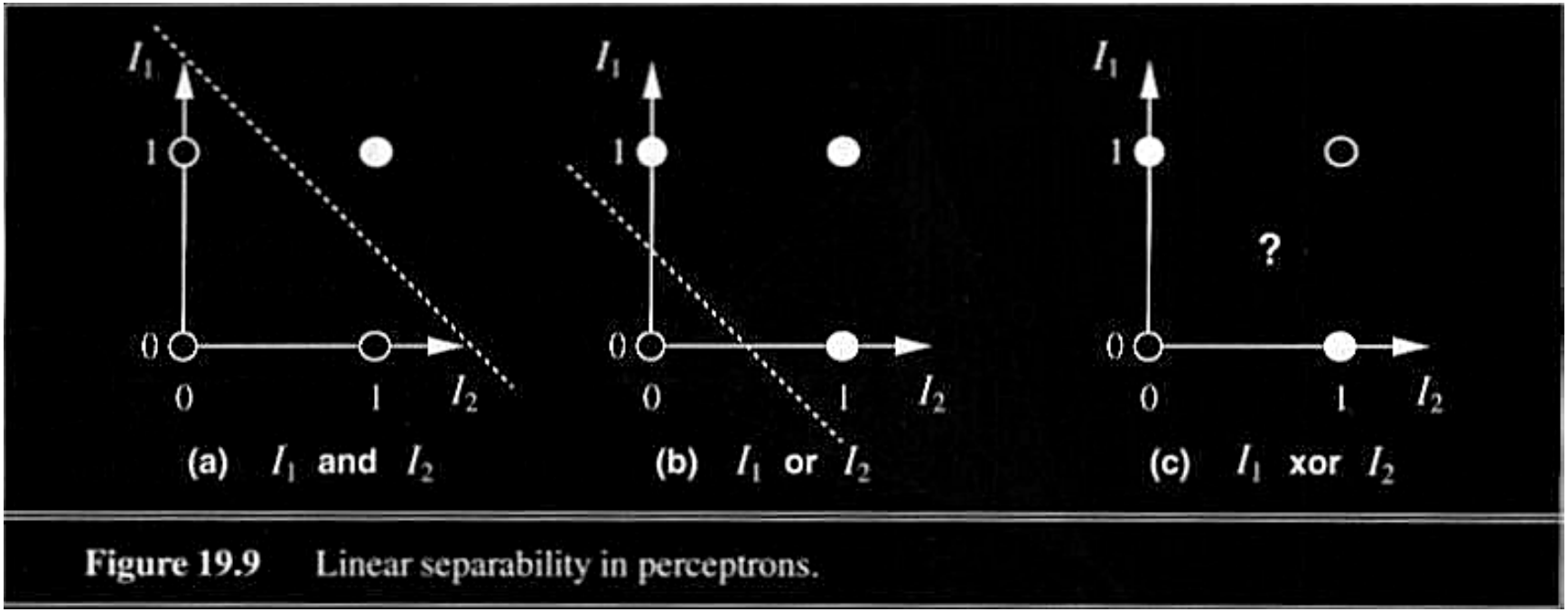
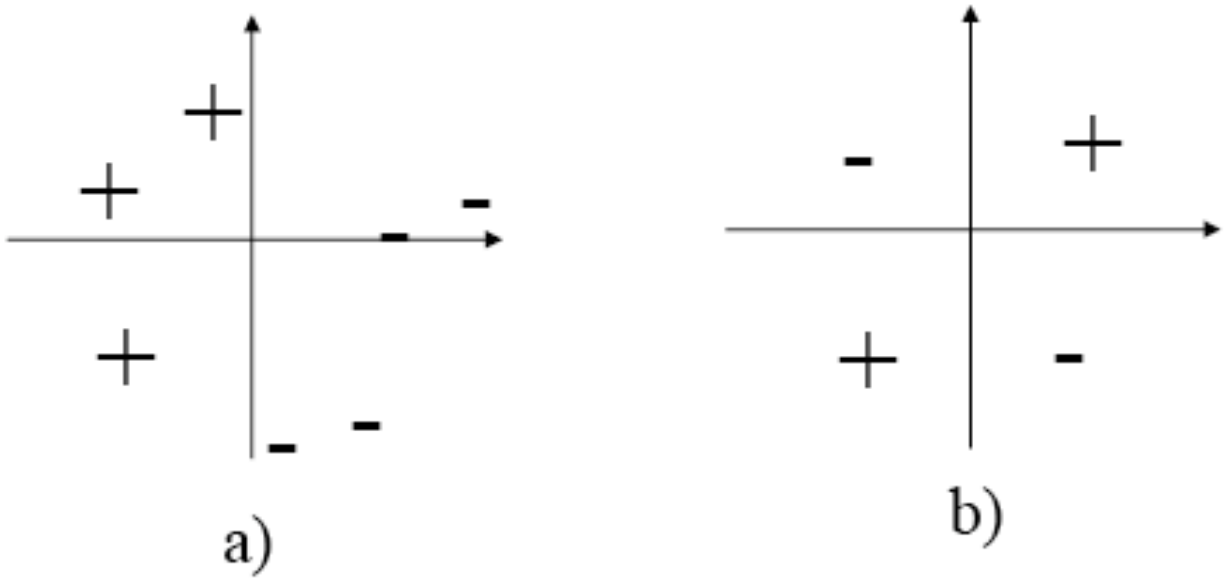


Figure 19.9 Linear separability in perceptrons.

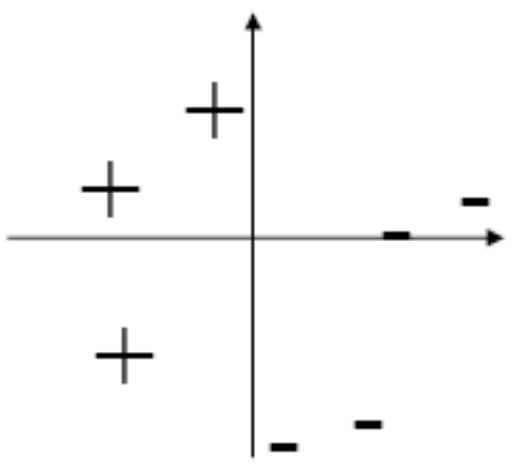
Linearly Separable Data

- Which of these datasets are separable by a linear boundary?

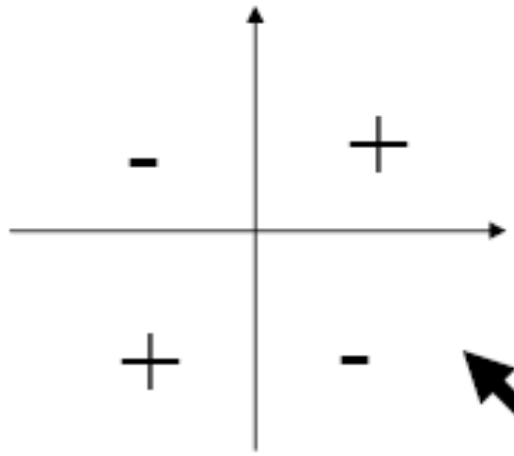


Linearly Separable Data

- Which of these datasets are separable by a linear boundary?



a)



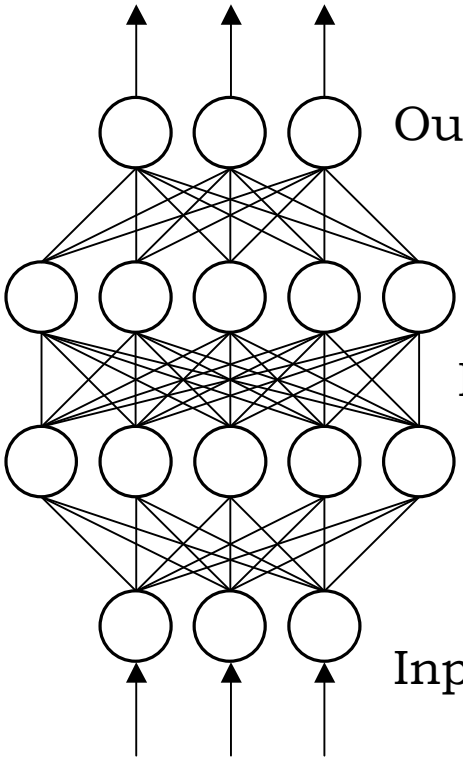
b)

Not Linearly Separable!

Solution to perceptron problem

- **Multi-layer perceptrons**
- **Hidden layer**
- **Can now represent more complex problems**

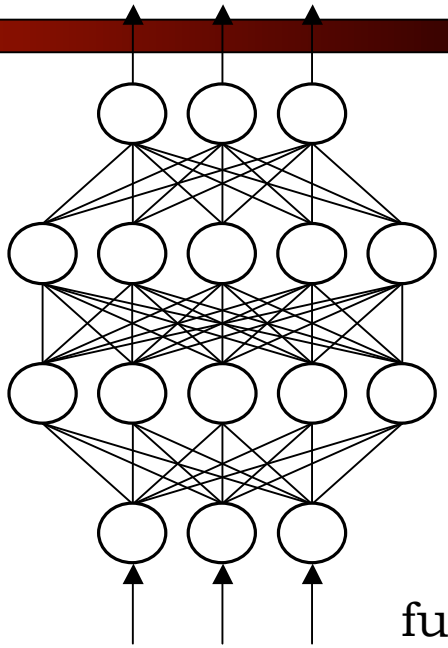
Artificial Neural Networks



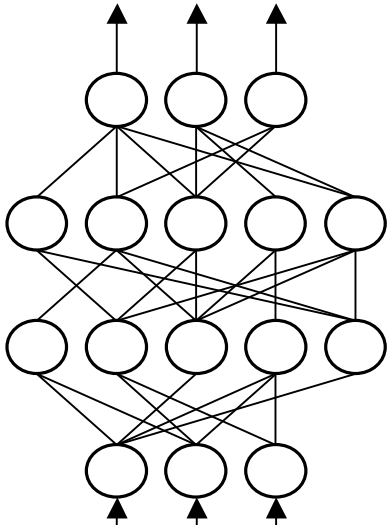
Output layer

Hidden layers

Input layer

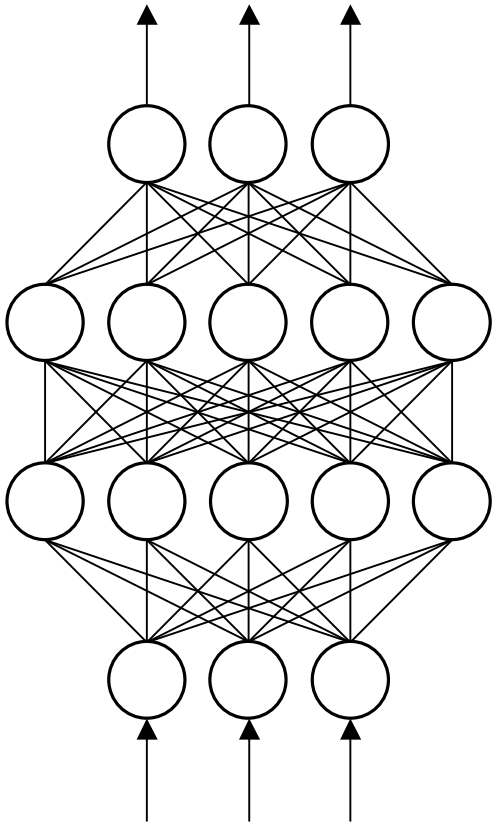


fully connected



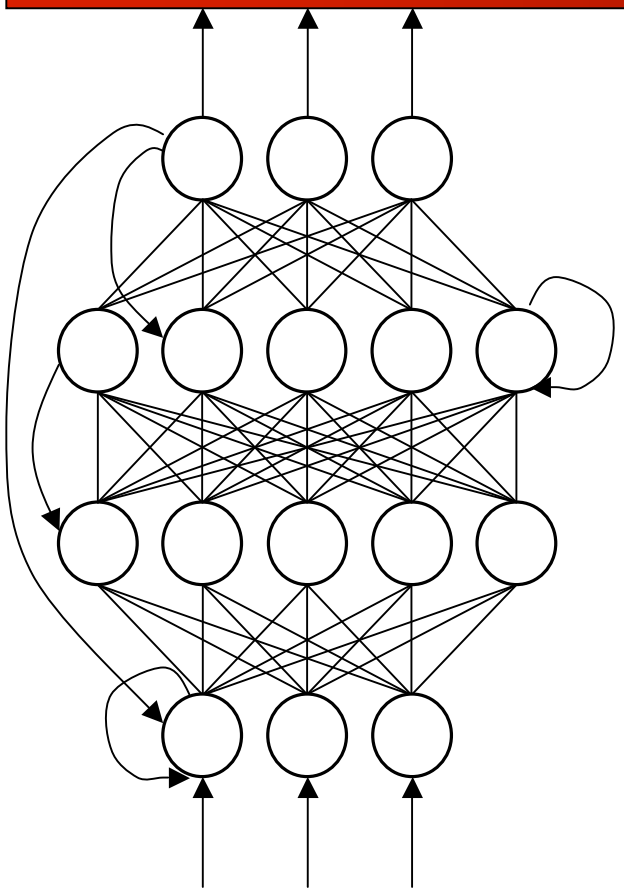
sparsely connected

Feedforward ANN Architectures



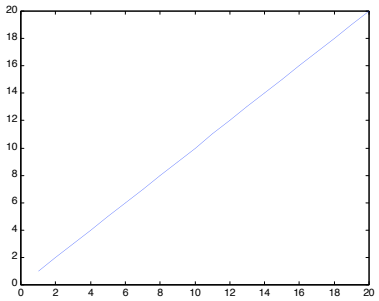
- Information flow unidirectional
- Static mapping: $y = f(x)$
- Multi-Layer Perceptron (MLP)
- Radial Basis Function (RBF)
- Kohonen Self-Organising Map (SOM)

Recurrent ANN Architectures

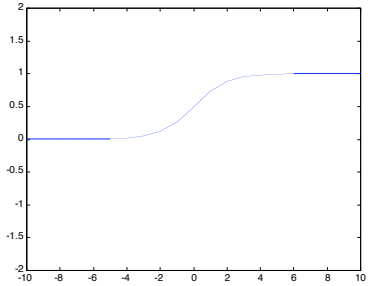


- Feedback connections
- Dynamic memory:
  $y(t+1)=f(x(\tau),y(\tau),s(\tau)) \quad \tau\in(t,t-1,\ldots)$
- Jordan/Elman ANNs
- Hopfield
- Adaptive Resonance Theory (ART)

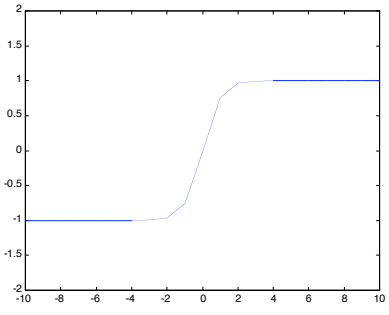
Activation functions



Linear $y = x$



Sigmoid $y = \frac{1}{1 + \exp(-x)}$



Hyperbolic tangent $y = \frac{\exp(x) - \exp(-x)}{\exp(x) + \exp(-x)}$

How does a perceptron learn?

- This is "supervised training" ("teacher signal")
- So we know the desired output
- And we know what output our network produces before learning (perhaps random weights)
- Simple intuition:
  - Change the weight by an amount proportional to the difference between the desired output and the actual output
  - Change in weight I = Current value of input I x (Desired Output - Current Output)

How does a perceptron learn?

- Change in weight I = Current value of input I x (Desired Output - Current Output)
- We'll add one more thing: a learning rate
- $\Delta w_i = \eta * (\text{Target-Output}) * \text{Input}$
- Where
  - η is learning rate
- Finally, let's call the difference between desired output (target) and current output delta (δ):
- $\Delta w_i = \eta x_i \delta$

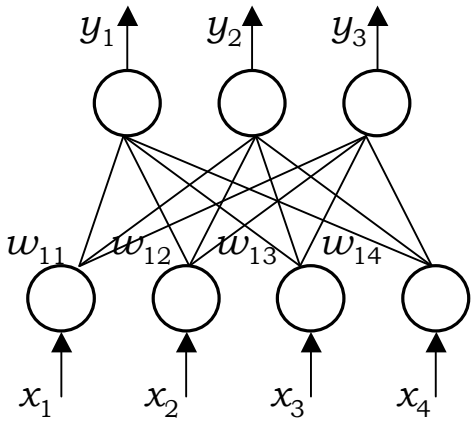
Delta Rule

$$e = d_i - y_i$$

$$\Delta w_{ij} = \lambda \cdot e \cdot x_j$$

λ=learning coefficient
w_{ij}=connection from neuron x_j to y_i
x=(x_1,x_2,…,x_n) ANN input
y=(y_1,y_2,…,y_n) ANN output
d=(d_1,d_2,…,d_n) desired output
(**x**,**d**) training example
e=ANN error

- **Least Mean Squares**
- **Widrow-Hoff iterative delta rule**
- **Gradient descent of the error surface**
- **Guaranteed to find minimum error configuration in single layer ANNs**



Perceptron Learning

- http://www.qub.ac.uk/mgt/intsys/perceptr.html
- **Error Back Propagation**
- **Just a generalization of the delta rule for multilayer networks**
- **The error (and weight changes) are propagated back through the network from the outputs back through the hidden layers.**

Machine Learning

- **Mitchell (1997)**
  - A computer program is said to learn from some experience E with respect to some class of tasks T and performance measure P if its performance at tasks in T, as measured by P, improves with experience E.
- **Witten and Frank (2000)**
  - Things learn when they change their behavior in a way that makes them perform better in the future

Motivating Example

- **Fictional data set that describes the weather conditions for playing some unspecified game**

outlook	temp.	humidity	windy	play
sunny	hot	high	false	no
sunny	hot	high	true	no
overcast	hot	high	false	yes
rainy	mild	high	false	yes
rainy	cool	normal	false	yes
rainy	cool	normal	true	no
overcast	cool	normal	true	yes

outlook	temp.	humidity	windy	play
sunny	mild	high	false	no
sunny	cool	normal	false	yes
rainy	mild	normal	false	yes
sunny	mild	normal	true	yes
overcast	mild	high	true	yes
overcast	hot	normal	false	yes
rainy	mild	high	true	no

Terminology

- **Instance**: single example in a data set. Example: each of the rows in preceding table
- **Feature**: an aspect of an instance. Example: outlook, temperature, humidity, windy. Can take categorical or numeric values
- **Value**: category that an attribute can take. Example: sunny, overcast, rainy.
- **Concept**: thing to be learned. Example: a classification of the instances into play and no play.

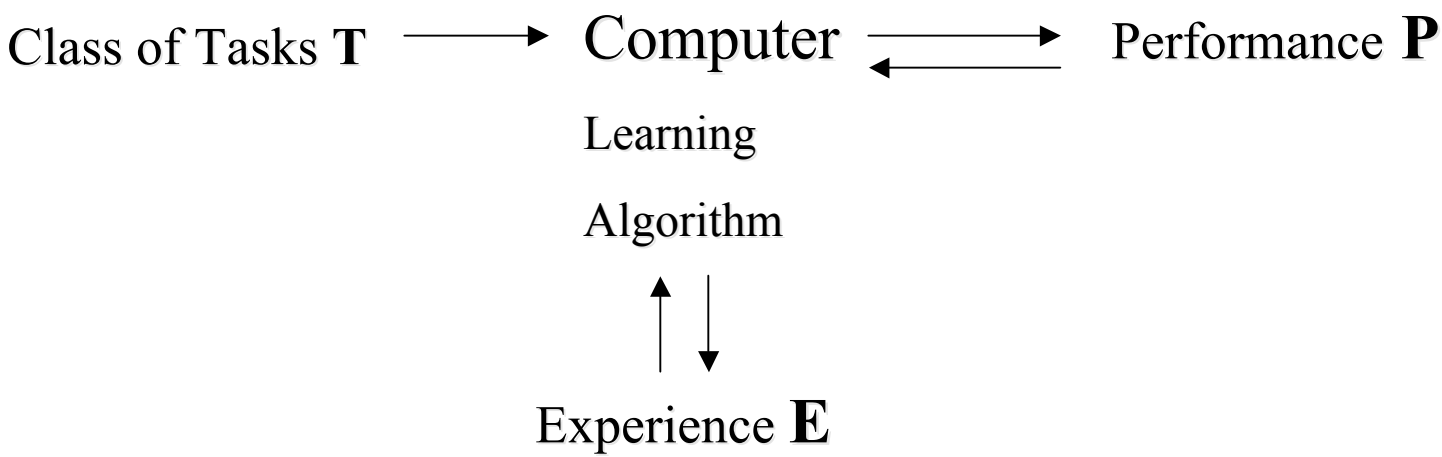
Learned Rules

- Example set of rules learned from the example data set:

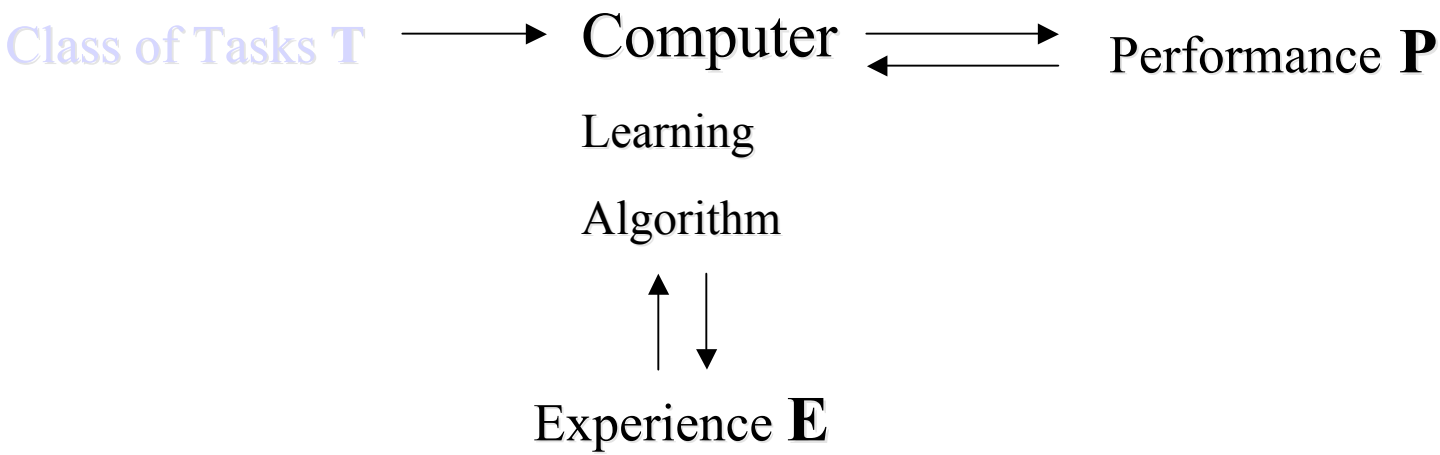
if outlook = sunny and humidity = high	then play = no
if outlook = rainy and windy = true	then play = no
if outlook = overcast	then play = yes
if humidity = normal	then play = yes
if none of the above	then play = yes

- This is a decision list:
  - Use first rule first, if doesn't apply, use 2nd rule, etc
- These are classification rules that assign an output class (play or not) to each instance

Visualization



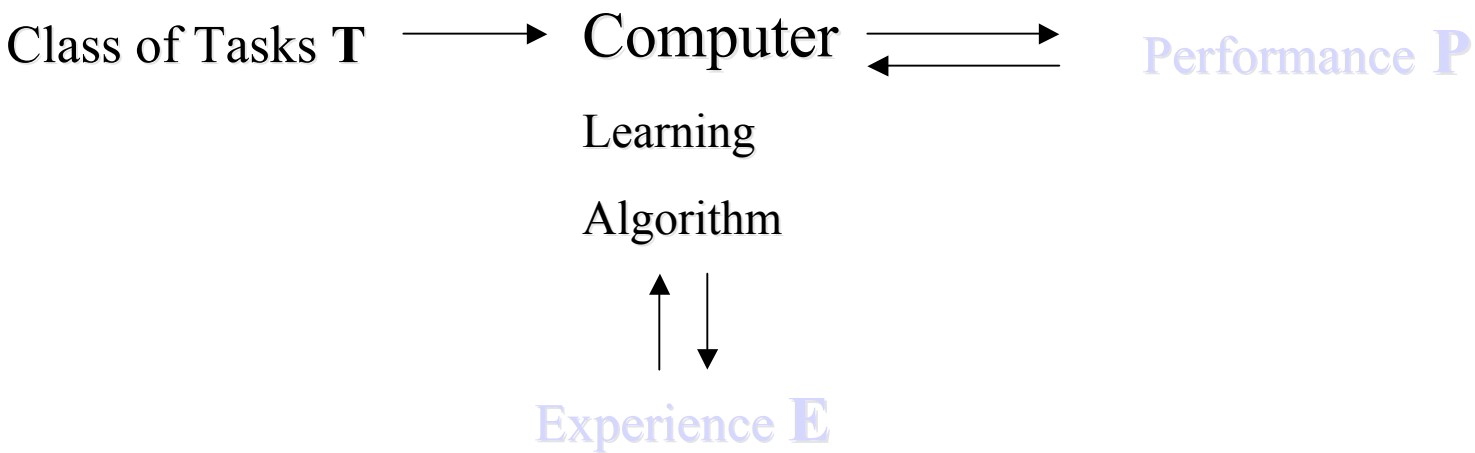
Class of Tasks



Class of Tasks

The activity on which the system will learn to improve its performance. Examples:

Experience and Performance



Experience and Performance

Experience: What has been recorded in the past

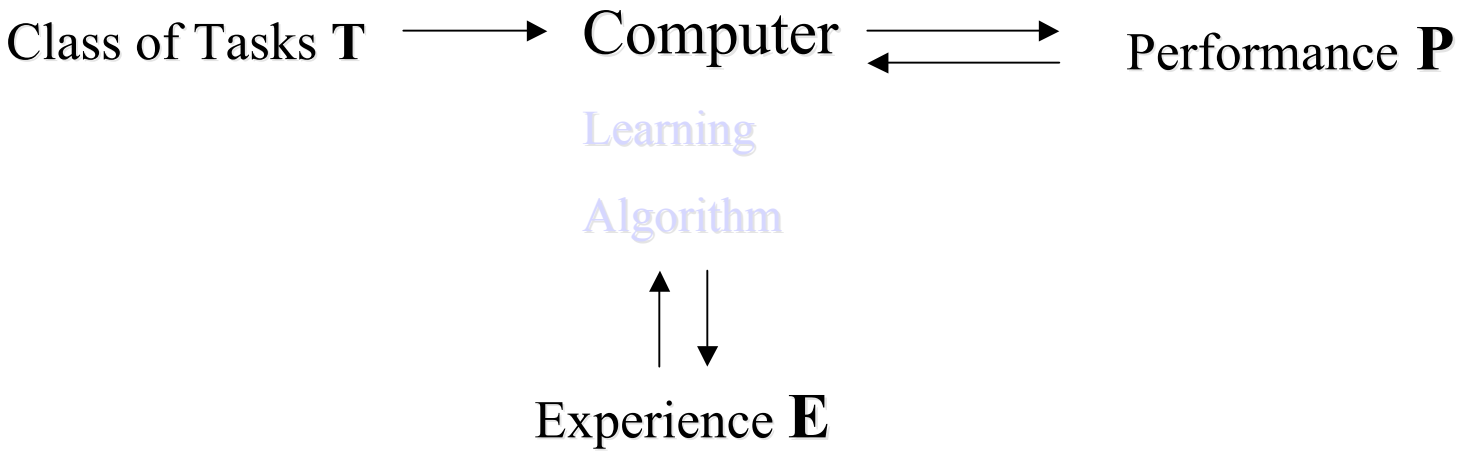
Performance: A measure of the quality of the response or action.

Example:

Handwritten recognition using Neural Networks

Experience: a database of handwritten images
with their correct classification
Performance: Accuracy in classifications

Designing a Learning System



Designing a Learning System

1. Define the knowledge to learn
2. Define the representation of the target knowledge
3. Define the learning mechanism

Example:

Handwritten recognition using Neural Networks

1. A function to classify handwritten images
2. A linear combination of handwritten features
3. A linear classifier

The Knowledge To Learn

Supervised learning: A function to predict the class of new examples

Let X be the space of possible examples
Let Y be the space of possible classes
Learn F : X ⟶ Y

Example:

In learning to play chess the following are possible interpretations:

X : the space of board configurations

Y : the space of legal moves

Representation of the Target Knowledge

Example: Diagnosing a patient coming into the hospital.

Features:

- X1: Temperature
- X2: Blood pressure
- X3: Blood type
- X4: Age
- X5: Weight
- Etc.

Given a new example $X = < x1, x2, \ldots, xn >$

$F(X) = w1x1 + w2x2 + w3x3 = \ldots + wnxn$

If $F(X) > T$ predict heart disease
otherwise predict no heart disease

The Learning Mechanism

Machine learning algorithms abound:

- ✓ Decision Trees
- ✓ Rule-based systems
- ✓ Neural networks
- ✓ Nearest-neighbor
- ✓ Support-Vector Machines
- ✓ Bayesian Methods

Kinds of Learning

- **Supervised**
  - **(And Semi-Supervised)**
- **Reinforcement**
- **Unsupervised**

- **(These are really kinds of feedback)**

Supervised Learning: Induction

- **General case:**
  - **Given a set of pairs (x, f(x)) discover the function f.**
- **Classifier case:**
  - **Given a set of pairs (x, y) where y is a label, discover a function that correctly assigns the correct labels to the x.**

Supervised Learning: Induction

- **Simpler Classifier Case:**
  - **Given a set of pairs (x, y) where x is an object and y is either a + if x is the right kind of thing or a – if it isn't. Discover a function that assigns the labels correctly.**

Error Analysis: Simple Case

Chosen \ Correct	+	-
+	Correct	False Positive
-	False Negative	Correct

Learning as Search

- **Everything is search…**
  - **A hypothesis is a guess at a function that can be used to account for the inputs.**
  - **A hypothesis space is the space of all possible candidate hypotheses.**
  - **Learning is a search through the hypothesis space for a good hypothesis.**

Hypothesis Space

- **The hypothesis space is defined by the representation used to capture the function that you are trying to learn.**
- **The size of this space is the key to the whole enterprise.**

What are the data for learning?

- **Instances**
  - **Features**
  - **values**
- **A set of such instances paired with answers, constitutes a training set.**

The Simple Approach

- **Take the training data, put it in a table along with the right answers.**
- **When you see one of them again retrieve the answer.**

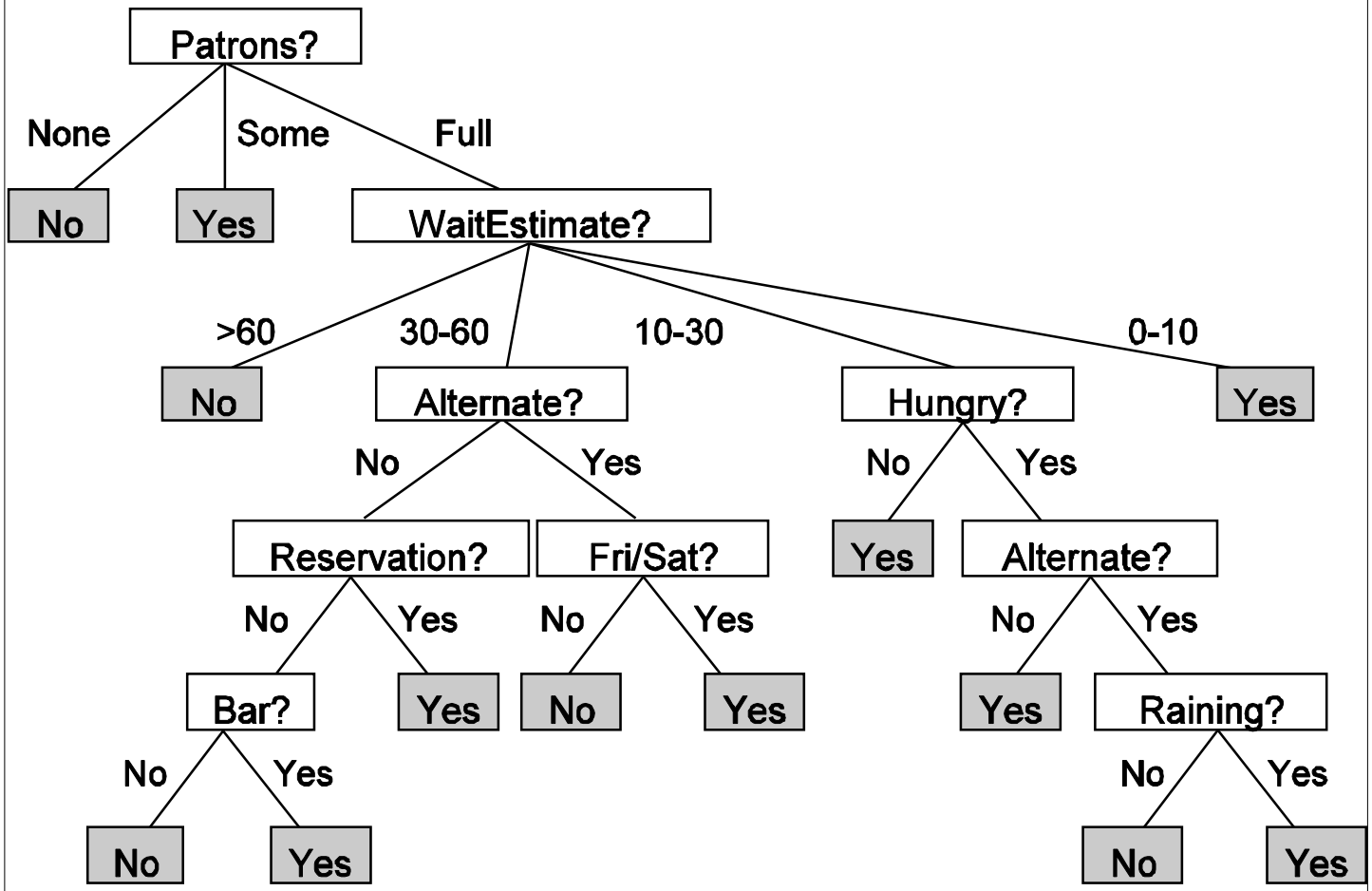
Neighbor-Based Approaches

- Build the table, as in the table-based approach.
- Provide a distance metric that allows you compute the distance between any pair of objects.
- When you encounter something not seen before, return as an answer the label on the nearest neighbor.

Decision Trees

- A decision tree is a tree where
  - Each internal node of the tree tests a single feature of an object
  - Each branch follows a possible value of each feature
  - The leaves correspond to the possible labels on the objects

Example Decision Tree



Decision Tree Learning

- Given a training set find a tree that correctly assigns labels (classifies) the elements of the training set.
- Sort of…there might be lots of such trees. In fact some of them look a lot like tables.

Training Set

Example	Attributes										Goal
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*WillWait*
X_1	*Yes*	*No*	*No*	*Yes*	*Some*	*$$$*	*No*	*Yes*	*French*	*0±10*	*Yes*
X_2	*Yes*	*No*	*No*	*Yes*	*Full*	*$*	*No*	*No*	*Thai*	*30±60*	*No*
X_3	*No*	*Yes*	*No*	*No*	*Some*	*$*	*No*	*No*	*Burger*	*0±10*	*Yes*
X_4	*Yes*	*No*	*Yes*	*Yes*	*Full*	*$*	*No*	*No*	*Thai*	*10±30*	*Yes*
X_5	*Yes*	*No*	*Yes*	*No*	*Full*	*$$$*	*No*	*Yes*	*French*	*>60*	*No*
X_6	*No*	*Yes*	*No*	*Yes*	*Some*	*$$*	*Yes*	*Yes*	*Italian*	*0±10*	*Yes*
X_7	*No*	*Yes*	*No*	*No*	*None*	*$*	*Yes*	*No*	*Burger*	*0±10*	*No*
X_8	*No*	*No*	*No*	*Yes*	*Some*	*$$*	*Yes*	*Yes*	*Thai*	*0±10*	*Yes*
X_9	*No*	*Yes*	*Yes*	*No*	*Full*	*$*	*Yes*	*No*	*Burger*	*>60*	*No*
X_{10}	*Yes*	*Yes*	*Yes*	*Yes*	*Full*	*$$$*	*No*	*Yes*	*Italian*	*10±30*	*No*
X_{11}	*No*	*No*	*No*	*No*	*None*	*$*	*No*	*No*	*Thai*	*0±10*	*No*
X_{12}	*Yes*	*Yes*	*Yes*	*Yes*	*Full*	*$*	*No*	*No*	*Burger*	*30±60*	*Yes*

Decision Tree Learning

- Start with a null tree.
- Select a feature to test and put it in tree.
- Split the training data according to that test.
- Recursively build a tree for each branch
- Stop when a test results in a uniform label or you run out of tests.

Well

- What makes a good tree?
  - Trees that cover the training data
  - Trees that are small…
- How should features be selected?
  - Choose features that lead to small trees.
  - How do you know if a feature will lead to a small tree?

Information Gain

- **Roughly…**
  - **Start with a pure guess the majority strategy. If I have a 50/50 split (y/n) in the training, how well will I do if I always guess yes?**
  - **Ok so now iterate through all the available features and try each at the top of the tree.**

Information Gain

- Then guess the majority label in each of the buckets at the leaves. How well will I do?
  - Well it's the weighted average of the majority distribution at each leaf.
- Pick the feature that results in the best predictions.

Training Set

Example	Attributes										Goal
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*WillWait*
X_1	*Yes*	*No*	*No*	*Yes*	*Some*	*$$$*	*No*	*Yes*	*French*	*0±10*	*Yes*
X_2	*Yes*	*No*	*No*	*Yes*	*Full*	*$*	*No*	*No*	*Thai*	*30±60*	*No*
X_3	*No*	*Yes*	*No*	*No*	*Some*	*$*	*No*	*No*	*Burger*	*0±10*	*Yes*
X_4	*Yes*	*No*	*Yes*	*Yes*	*Full*	*$*	*No*	*No*	*Thai*	*10±30*	*Yes*
X_5	*Yes*	*No*	*Yes*	*No*	*Full*	*$$$*	*No*	*Yes*	*French*	*>60*	*No*
X_6	*No*	*Yes*	*No*	*Yes*	*Some*	*$$*	*Yes*	*Yes*	*Italian*	*0±10*	*Yes*
X_7	*No*	*Yes*	*No*	*No*	*None*	*$*	*Yes*	*No*	*Burger*	*0±10*	*No*
X_8	*No*	*No*	*No*	*Yes*	*Some*	*$$*	*Yes*	*Yes*	*Thai*	*0±10*	*Yes*
X_9	*No*	*Yes*	*Yes*	*No*	*Full*	*$*	*Yes*	*No*	*Burger*	*>60*	*No*
X_{10}	*Yes*	*Yes*	*Yes*	*Yes*	*Full*	*$$$*	*No*	*Yes*	*Italian*	*10±30*	*No*
X_{11}	*No*	*No*	*No*	*No*	*None*	*$*	*No*	*No*	*Thai*	*0±10*	*No*
X_{12}	*Yes*	*Yes*	*Yes*	*Yes*	*Full*	*$*	*No*	*No*	*Burger*	*30±60*	*Yes*

Patrons

- Picking Patrons at the top takes the initial 50/50 split and produces three buckets
  - None: 0 Yes, 2 No
  - Some: 4 Yes, 0 No
  - Full: 2 Yes, 4 No
- How well does guessing do?
  - 2+4+4 = 10 right, 0+0+2 = 2 wrong

Iterate

- Do that for each feature, select the one that gives the best result, put that at the top of the tree.
- Recurse
  - Split the training data according to the values of the first feature
  - Build the tree recursively in the same manner

Training and Evaluation

- **Given a fixed size training set, we need a way to**
  - **Organize the training**
  - **Assess the learned system's likely performance on unseen data**

Test Sets and Training Sets

- Divide your data into three sets:
  - Training set
  - Development test set
  - Test set

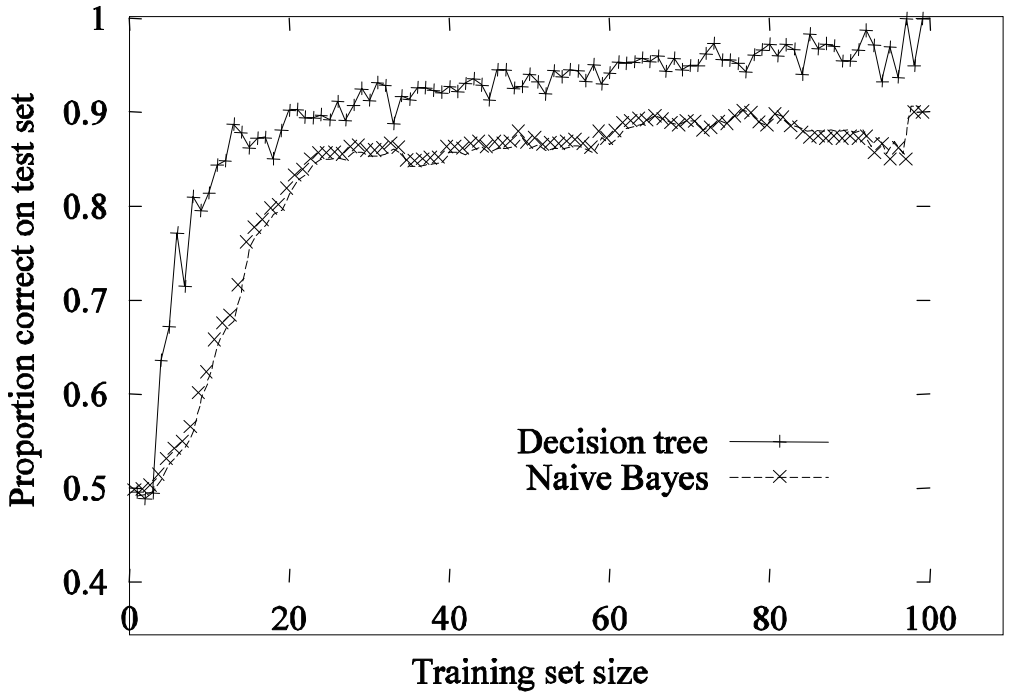
1. Train on the training set
2. Tune using the dev-test set
3. Test on withheld data

Cross-Validation

- What if you don't have enough training data for that?
  1. Divide your data into N sets and put one set aside (leaving N-1)
  2. Train on the N-1 sets
  3. Test on the set aside data
  4. Put the set aside data back in and pull out another set
  5. Go to 2
  6. Average all the results

Performance Graphs

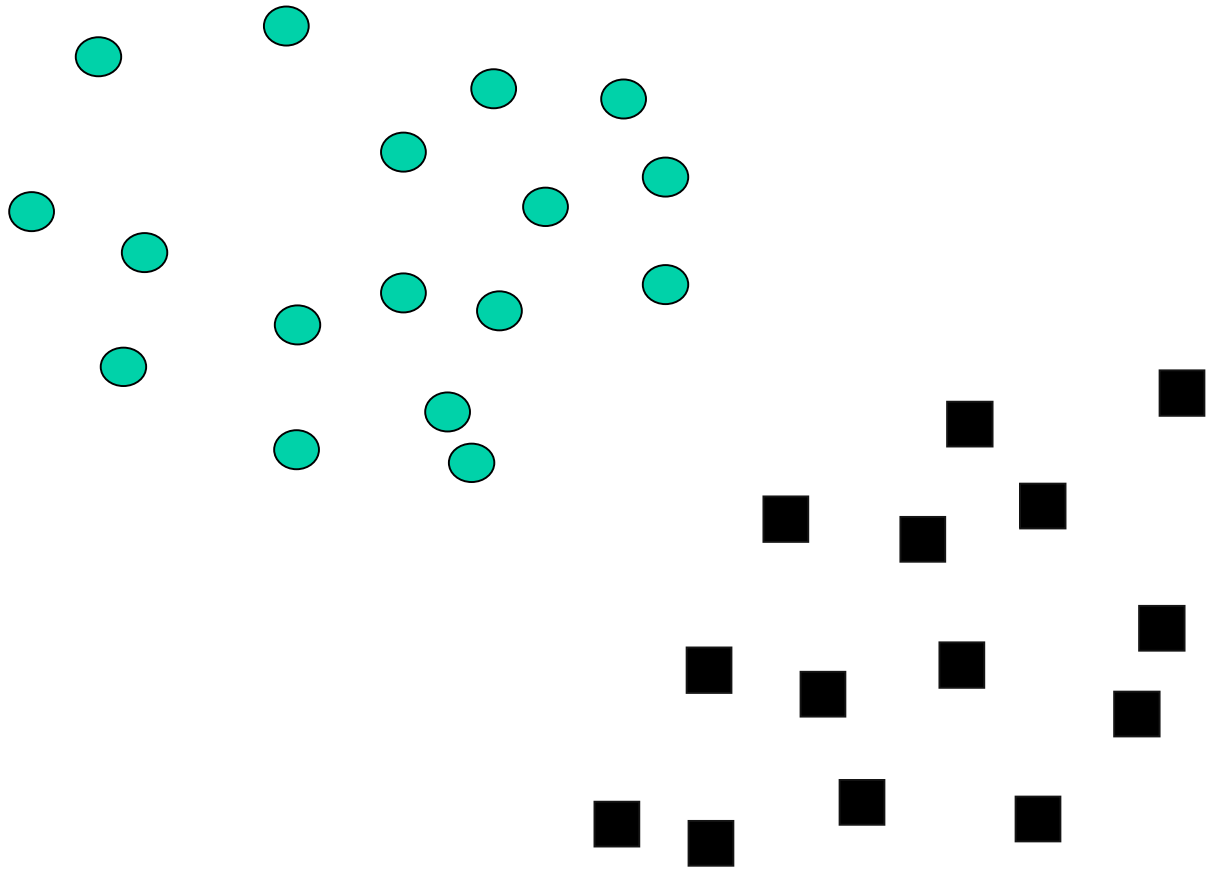
- Its useful to know the performance of the system as a function of the amount of training data.



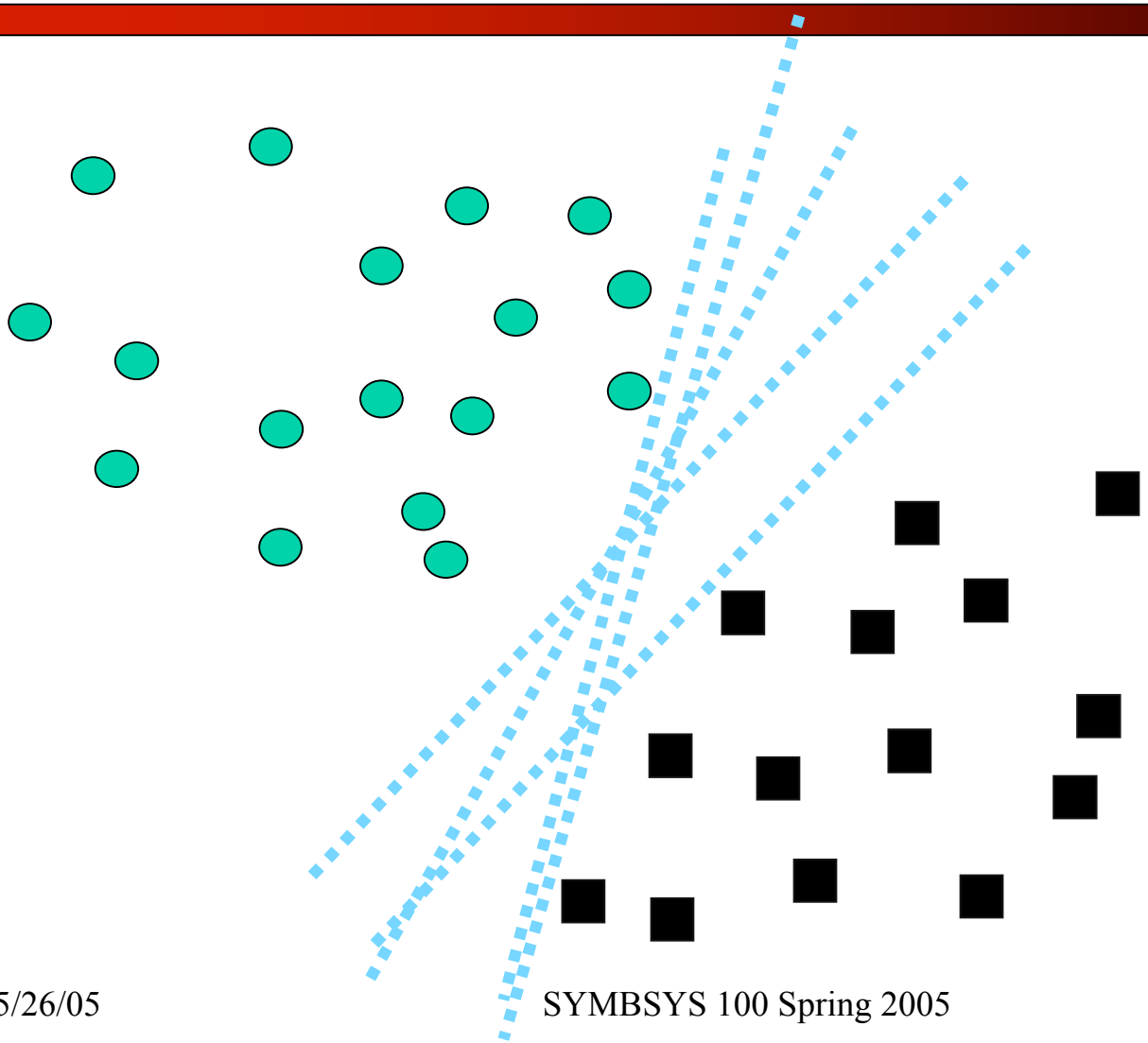
Support Vector Machines

- Can be viewed as a generalization of neural networks
- Two key ideas
  - The notion of the margin
    - Support vectors
  - Mapping to higher dimensional spaces
    - Kernel functions

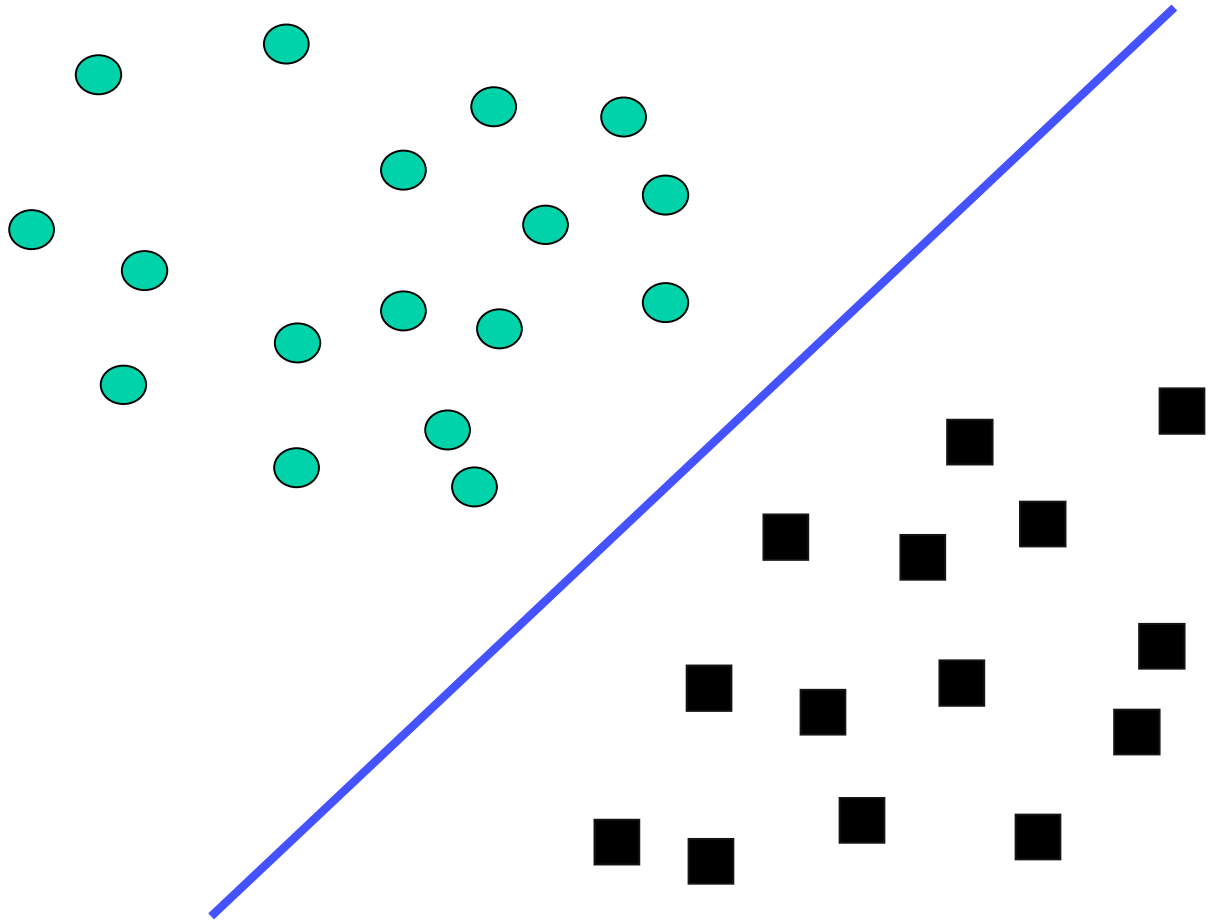
Best Linear Separator?



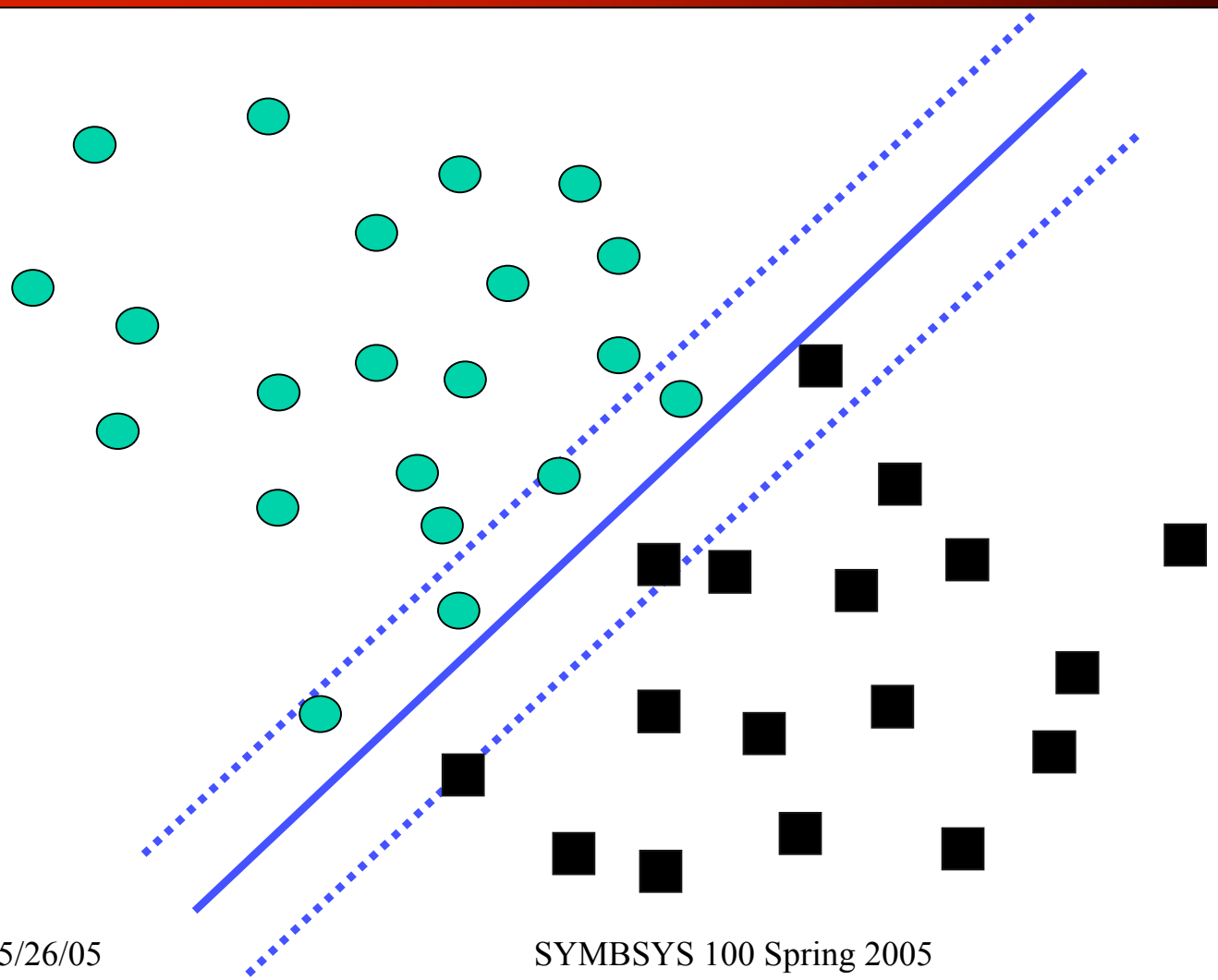
Best Linear Separator?



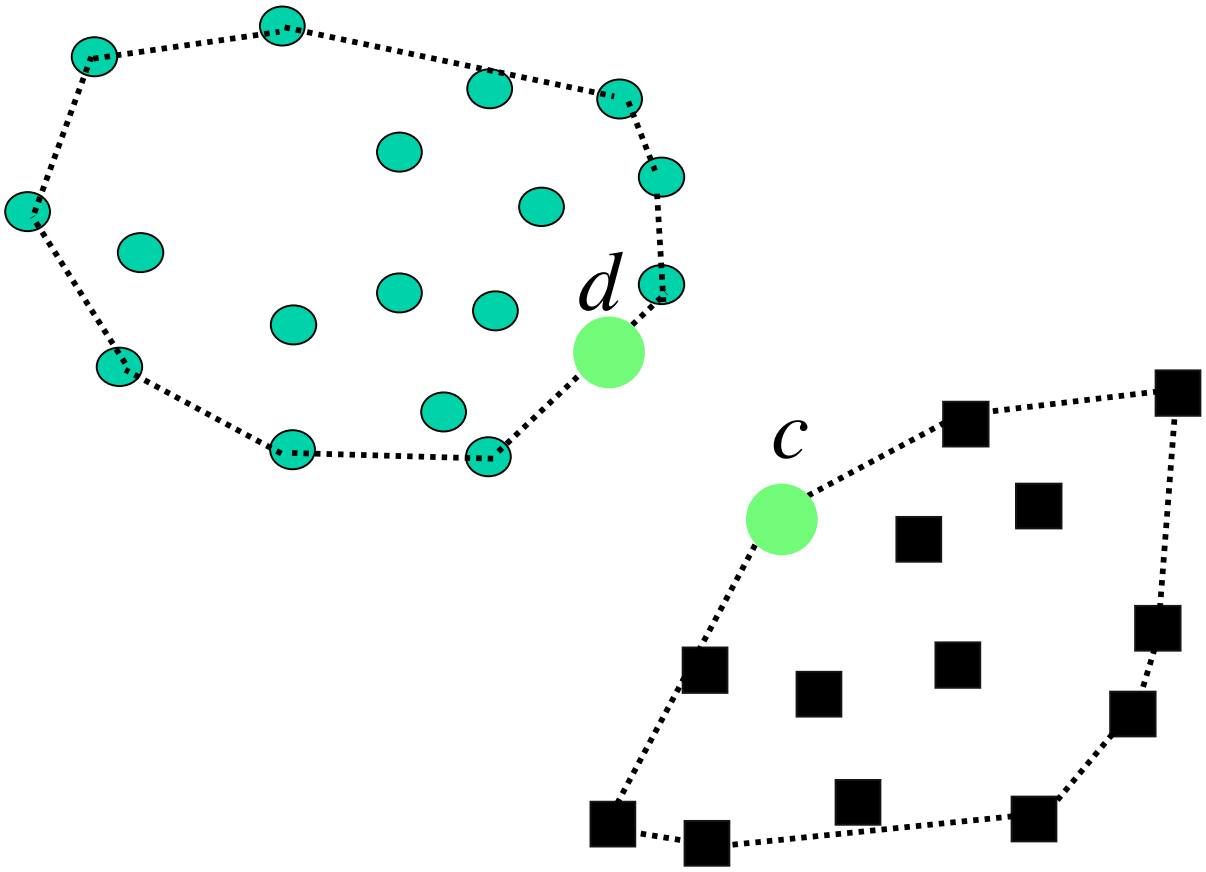
Best Linear Separator?



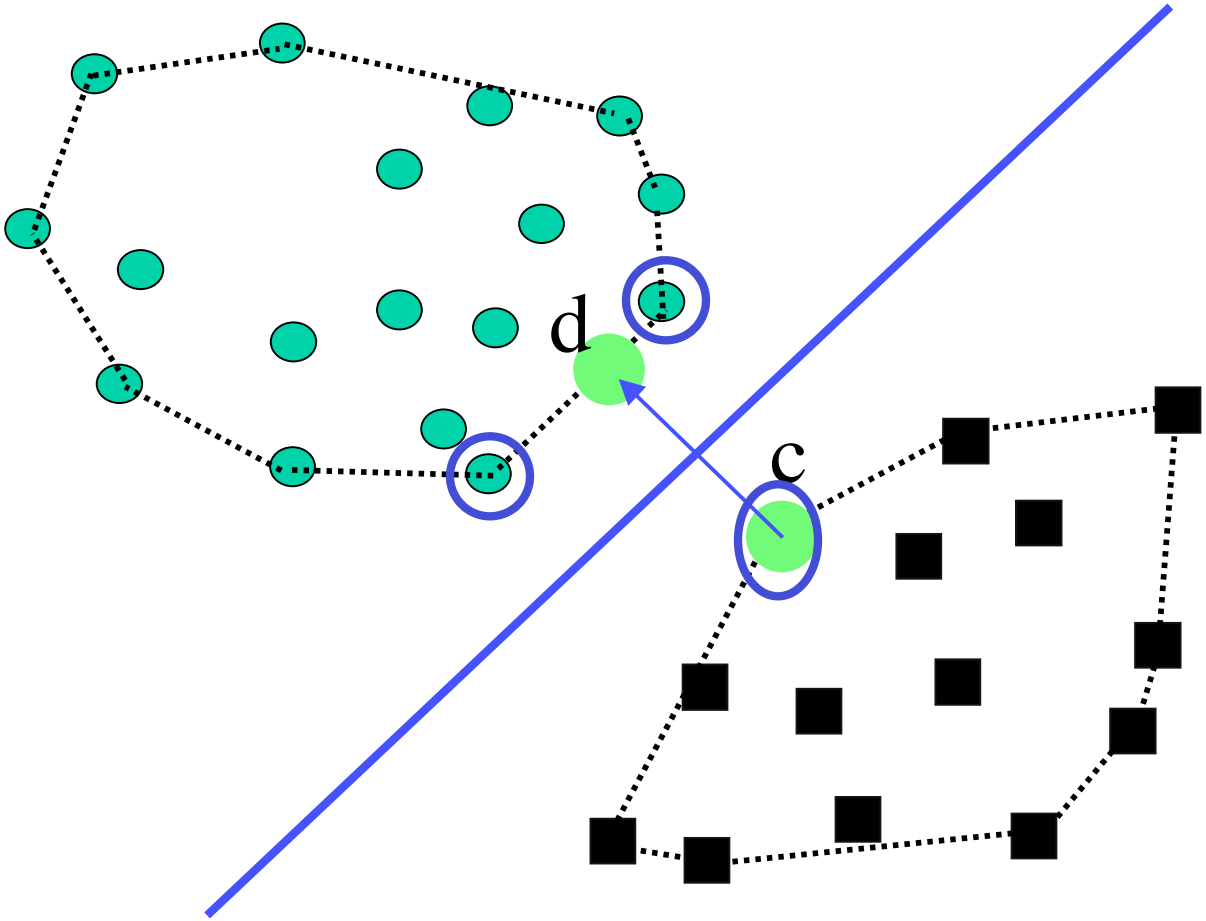
Why is this good?



Find Closest Points in Convex Hulls

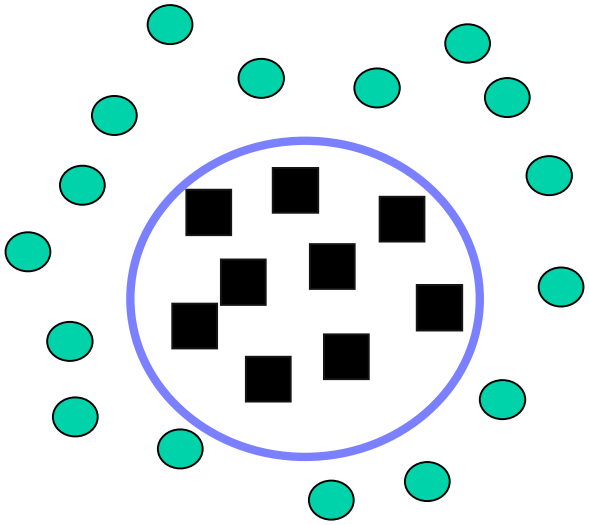


Plane Bisect Support Vectors



Higher Dimensions

- That assumes that there is a linear classifier that can separate the data.



One Solution

- Well, we could just search in the space of non-linear functions that will separate the data
- Two problems
  - Likely to overfit the data
  - The space is too large

Kernel Trick

- Map the objects to a higher dimensional space.
- Book example
  - Map an object in two dimensions (x1 and x2) into a three dimensional space
    - F1 = x1^2, F2 = x2^2, and F3 = Sqrt(2*x1*x2)
- Points not linearly separable in the original space will be separable in the new space.

But

- In the higher dimensional space, there are gazillion hyperplanes that will separate the data cleanly.
  - How to choose among them?
    - Use the support vector idea

Conclusion

- Machine learning
  - Supervised
    - Neural networks
    - Decision trees
    - Decision list
    - SVM
    - Bayesian classifiers, etc etc
  - Unsupervised
  - Reinforcement (reward) learning