C2P6
Artificial Intelligence in P.A.

Introduction
To
Machine Learning

Machine Learning

- ***What is learning?***

- The definition of learning is **the process or experience of gaining knowledge or skill**. An example of learning is a student understanding and remembering what they've been taught.
- The three major types of learning described by behavioral psychology are **classical conditioning, operant conditioning, and observational learning**.
- The importance of learning is that **it helps the individual to acquire the necessary skills through learning and knowledge so that he can achieve his set goals**.

Definitions

- **What is Machine Learning?**

  - To gain knowledge or understanding of or skill in by study, instruction or experience; memorize; to acquire knowledge or skill in a behavioral tendency; discovery, to obtain knowledge of for the first time.

  - Any process by which a system improves its performance.

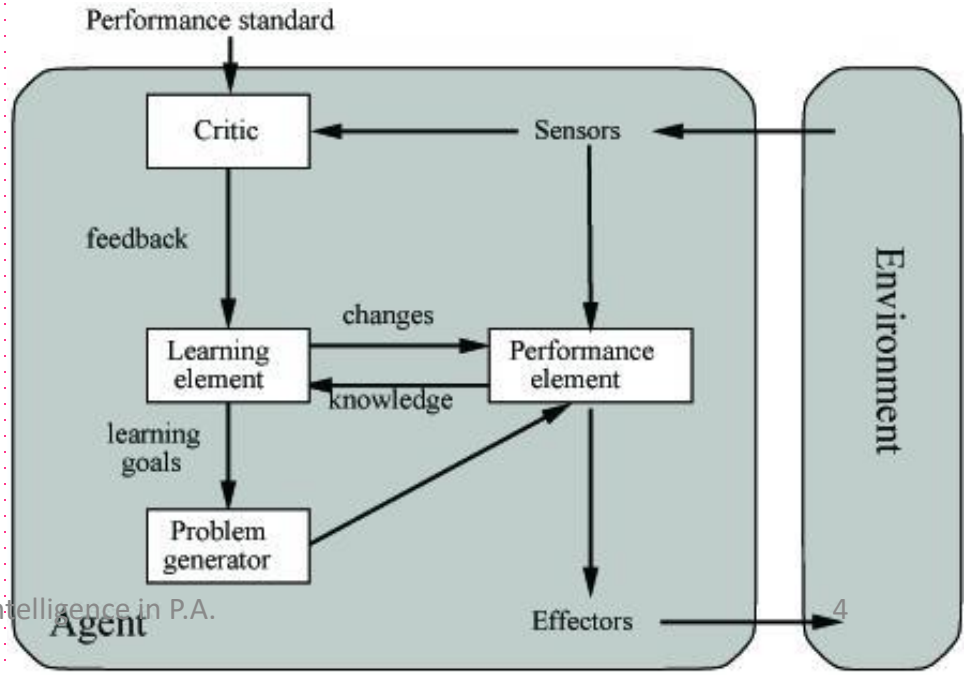
  A more ***formal definition*** is the following:

- ***Machine learning*** *is essentially a form of applied statistics with increased emphasis on the use of computers to statistically estimate complicated functions and a decreased emphasis on proving confidence intervals around these functions.*

- In this lecture therefore we will firtst present the two central approaches to statistics:
- frequentist estimators and Bayesian inference.
- Most machine learning algorithms can be divided into the categories:
- supervised learning and unsupervised learning;

A General Model of Learning Systems

- **Learning Element**
  - Adds knowledge, makes improvement to system
- **Performance Element**
  - Performs task, selects external actions
- **Critic**
  - Monitors results of performance, provides feedback to learning element
- **Problem Generator**
  - Actively suggests experiments, generates examples to test
- **Performance Standard**
  - Method / standard of measuring performance



The Learning Problem

- Learning = Improving with experience at some task
  - Improve over task T
  - With respect to performance measure P
  - Based on experience E
- Example: Learn to play checkers (Chinook)
  - T: Play checkers
  - P: % of games won in world tournament
  - E: opportunity to play against self
- Example: Learn to Diagnose Patients
  - T: Diagnose patients
  - P: Percent of patients correctly diagnosed
  - E: Pre-diagnosed medical histories of patients

Categories of Learning

- Learning by being told
- Learning by examples / Supervised learning
- Learning by discovery / Unsupervised learning
- Learning by experimentation / Reinforcement learning

- Syskill and Webert Perform Web Page Rating
- Example of supervised learning
- Example of supervised learning
- Example of unsupervised learning

Learning From Examples

- Learn general concepts or categories from examples
- Learn a task (drive a vehicle, win a game of backgammon)
- Examples of objects or tasks are gathered and stored in a database
- Each example is described by a set of attributes or features
- Each example used for training is classified with its correct label (chair, not chair, horse, not horse, 1 vs. 2 vs. 3, good move, bad move, etc.)
- The machine learning program learns general concept description from these specific examples
- The ML program should be applied to classify or perform tasks *never before seen* from learned concept

Learning Algorithms (LA)

- There is a variety of definitions of LAs. A simple definition is the following.
- A machine learning algorithm is an algorithm that is able to learn from data.
- A more precise definition is: MLs are those that can learn from data and improve by experience, without human intervention.
- Learning tasks may include learning the function that maps the input to the output, learning the hidden structure of labeled data or similar structural characteristics of unlabeled data.
- We will answer the question: what do we mean by learning? Mitchell (1997) provides a succinct, more formal, definition:
- A computer program is said to learn from 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.
- One can imagine a wide variety of experiences E, tasks T, and performance measures P.
- In the following, we will provide intuitive descriptions and examples of the different kinds of tasks, performance measures, and experiences that can be used to construct machine learning algorithms.
- Thus the clarification of what exactly is a task T is defined, the different kinds of tasks is discussed as well as the two learning paradigms: Unsupervised Learning (for unlabeled data) and Supervised Learning (for labeled data).

Learning From Examples

- First algorithm: naïve Bayes classifier
- D is training data
  - Each data point is described by attributes $a_1..a_n$
- Learn mapping from data point to a class value
  - Class values $v_1..v_j$
- We are searching through the space of possible concepts
  - Functions that map data to class value

Supervised Learning Algorithm – Naïve Bayes

1. For each hypothesis $h \in H$, calculate the posterior probability

$$P(h \mid D) = \frac{P(D \mid h)P(h)}{P(D)}$$

2. Output the hypothesis h_{MAP} (maximum a posteriori hypothesis) with the highest posterior probability

$$h_{MAP} = \arg\max_{h \in H} P(h \mid D)$$

NBC Definition

- Assume target function is f:D->V, where each instance d is described by attributes <a_1, a_2, .., a_n>. The most probable value of f(d) is

$$v_{MAP} = \arg\max_{v_j \in V} P(v_j \mid a_1, a_2, .., a_n)$$

$$v_{MAP} = \arg\max_{v_j \in V} \frac{P(a_1, a_2, .., a_n)P(v_j)}{P(a_1, a_2, .., a_n)}$$

$$= \arg\max_{v_j \in V} P(a_1, a_2, .., a_n \mid v_j)P(v_j)$$

NB Assumption

- Assume that the attributes are independent with respect to the class. The result is

$$P(a_1, a_2, .., a_n \mid v_j) = \prod_i P(a_i \mid v_j)$$

- which yields the NBC

$$v_{NB} = \arg\max_{v_j \in V} P(v_j) \prod_i P(a_i \mid v_j)$$

Using NBC

- Training
  - For each target value (class value) v_j estimate $P(v_j)$
- For each attribute value a_i of each attribute a estimate $P(a_i|v_j)$
- Classify new instance

$$v_{NB} = \arg\max_{v_j \in V} P(v_j) \prod_{a_i \in d} P(a_i \mid v_j)$$

PlayTennis Training Examples

Day	Outlook	Temperature	Humidity	Wind	PlayTennis
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

PlayTennis Example

- Target values are Yes and No
- Assume examples 1-12 comprise training data
- P(Yes) = 8/12, P(No) = 4/12
- Now we want to determine class for example 13 (Outlook=Overcast, Temperature=Hot, Humidity=Normal, Wind=Weak)
- Determine which value is larger
- P(Yes)*P(Overcast|Yes)*P(Hot|Yes)*P(Normal|Yes)*P(Weak|Yes) = 8/12 * 2/8 * 1/8 * 5/8 * 5/8 = 0.00814
- P(No)*P(Overcast|No)*P(Hot|No)*P(Normal|No)*P(Weak|No) = 4/12 * 0 * 2/4 * 1/4 * 2/4 = 0.0
- Answer is Yes

NBC Subtleties

- Conditional independence is often violated...
  - ...but it works surprisingly well anyway. We do not need the actual probability, just the class that yields the largest value.
- Some attribute values may not appear in any training examples.
  - Use small non-zero value

NBC For Text Classification

- Uses
  - Filter spam
  - Classify web pages by topic
  - Categorize email
- NBC is very effective for this application.
- What attributes shall we use?
- Do we care about word position?
- Joachims performed experiment with 20 newsgroups, 1000 articles per class, 2/3 training 1/3 testing, achieved 89% accuracy

Prediction Problems

- Customer purchase behavior

Prediction Problems

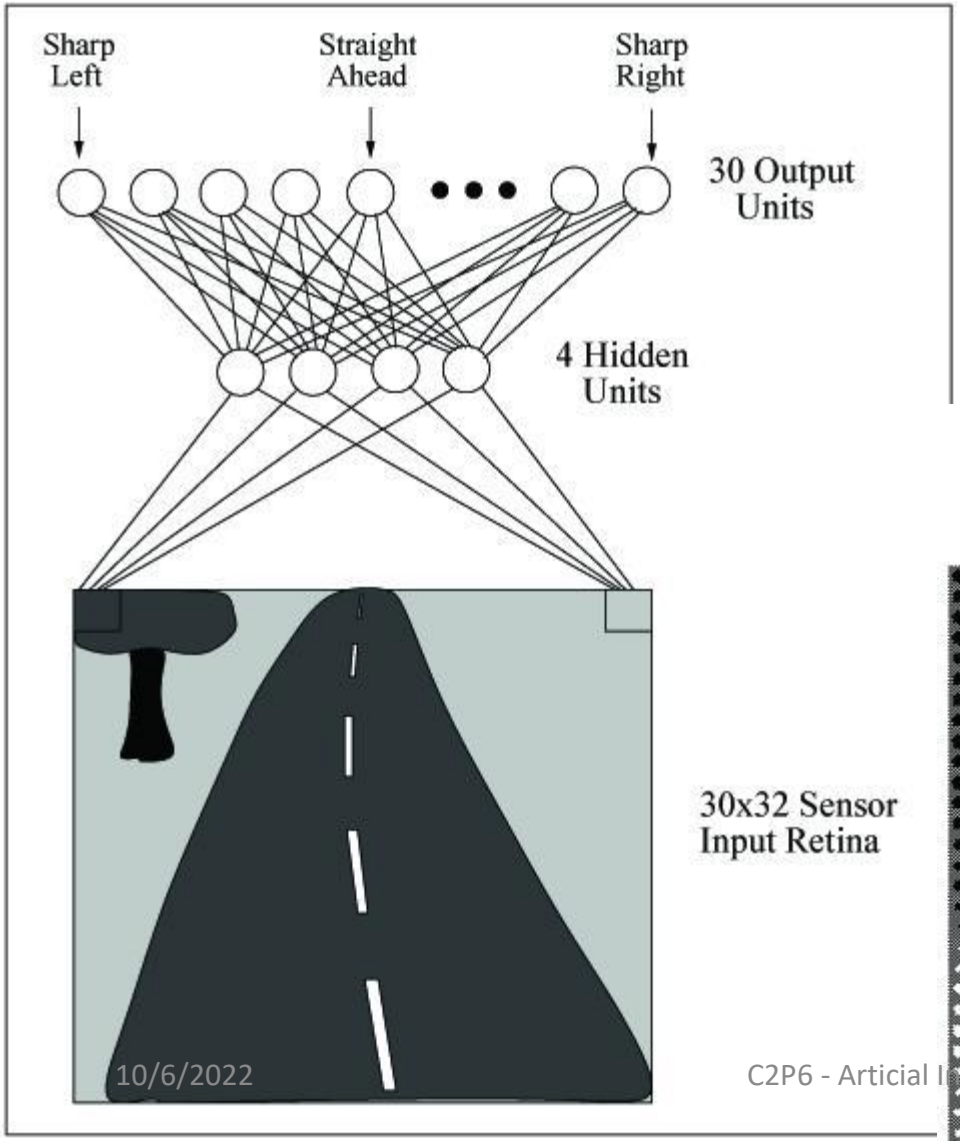
- Customer retention

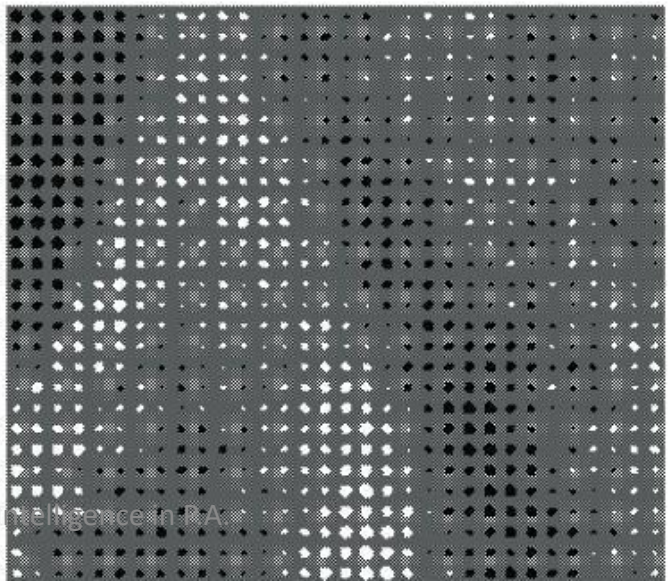
Prediction Problems

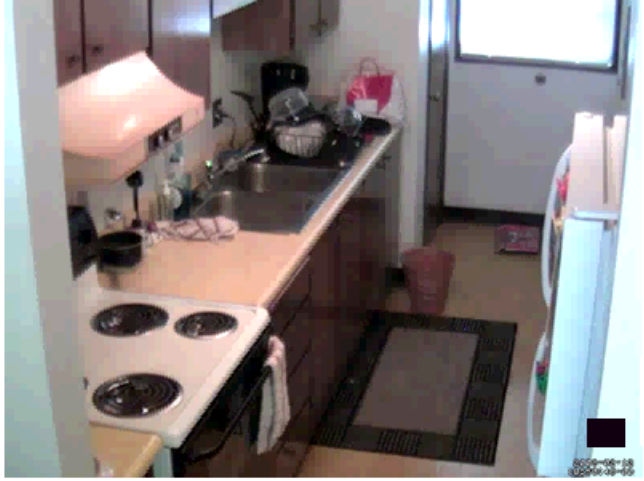
Product72: (time=t0)	Product72: (time=t1)	...	Product72: (time=tn)	
Stage: mix	Stage: cook		Stage: cool	
Mixing-speed: 60rpm	Temperature: 325		Fan-speed: medium	
Viscosity: 1.3	Viscosity: 3.2		Viscosity: 1.3	
Fat content: 15%	Fat content: 12%		Fat content: 12%	
Density: 2.8	Density: 1.1		Density: 1.2	
Spectral peak: 2800	Spectral peak: 3200		Spectral peak: 3100	
Product underweight?: ??	Product underweight?: ??		Product underweight?:	Yes
...	...		...	

Prediction Problems

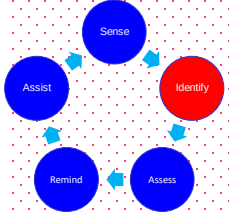
- Problems too difficult to program by hand



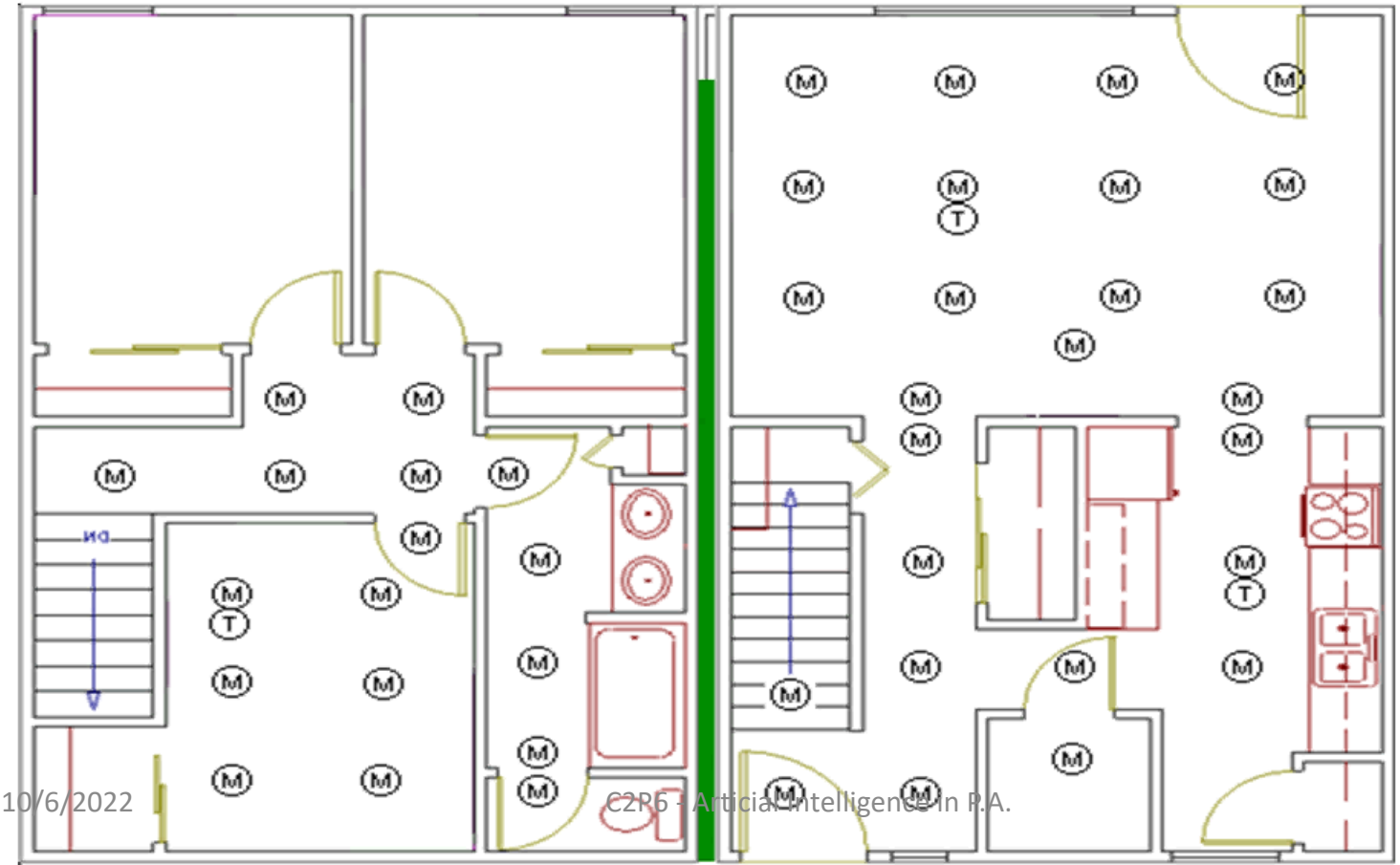




Identify




```
sensor ID          |
   date / time                    | reading
-----------------------------------------
```

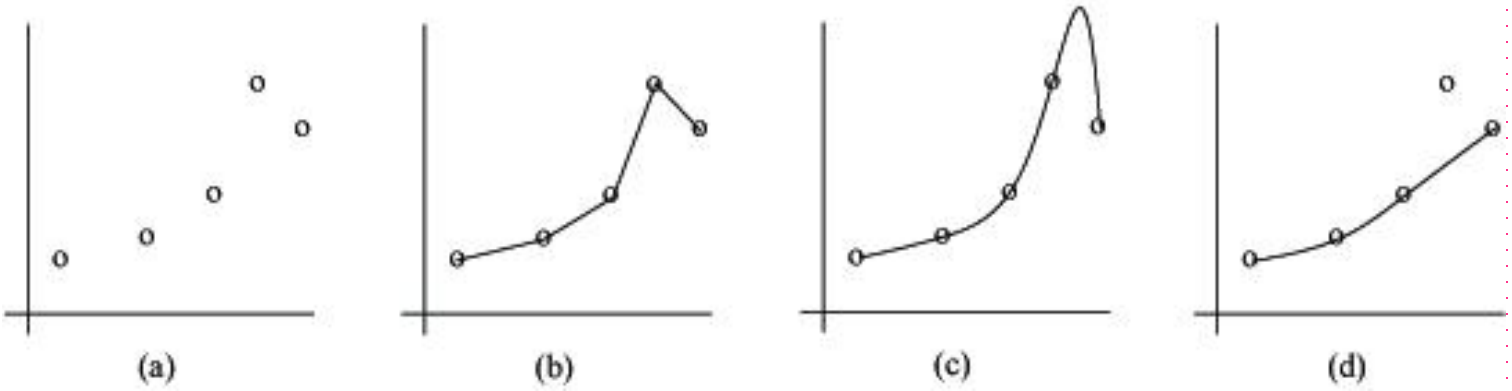


Inductive Learning Hypothesis

- Any hypothesis found to approximate the target function well over a sufficiently large set of training examples will also approximate the target function well over other unobserved examples

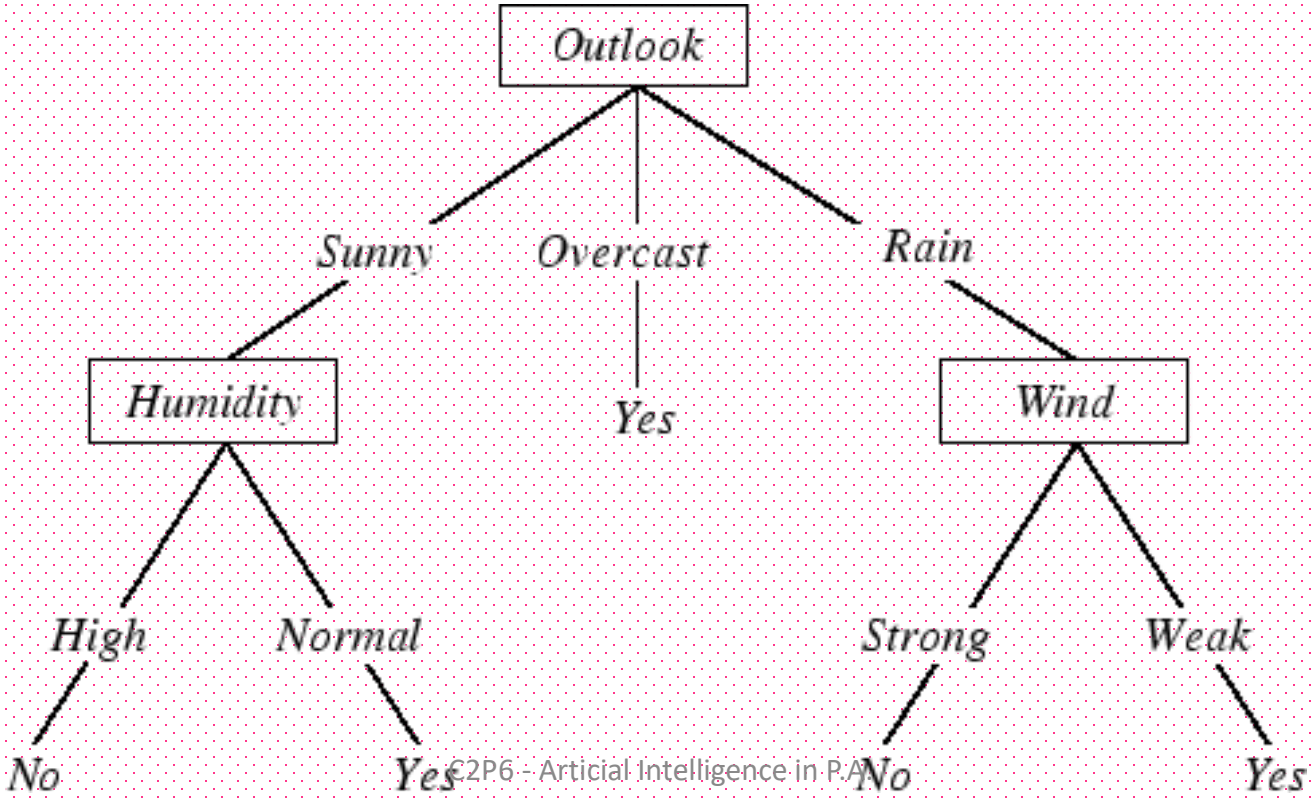
Inductive Bias

- There can be a number of hypotheses consistent with training data
- Each learning algorithm has an inductive bias that imposes a preference on the space of all possible hypotheses



Decision Trees

- A decision tree takes a description of an object or situation as input, and outputs a yes/no "decision".
- Can also be used to output greater variety of answers.
- Here is a decision tree for the concept *PlayTennis*



Decision Tree Representation

- Each internal node tests an attribute
- Each branch corresponds to attribute value
- Each leaf node assigns a classification

When to Consider Decision Trees

- Instances describable by attribute-value pairs
- Target function is discrete valued
- Disjunctive hypothesis may be required
- Possibly noisy training data

Inducing Decision Trees

- Each example is described by the values of the attributes and the value of the goal predicate (Yes/No)
- The value of the goal predicate is called the classification of the example
- If the classification is true (Yes), this is a positive example, otherwise this is a negative example
- The complete set of examples is called the training set

Decision Tree Learning

- Any concept that can be expressed as a propositional statement can be expressed using a decision tree
- No type of representation is efficient for all kinds of functions
  How represent of m of n?
- Once we know how to use a decision tree, the next question is, how do we automatically construct a decision tree?
- One possibility: search through the space of all possible decision trees
  - All possible n features at root
  - For each root, n-1 possible features at each child
  - …
  - Keep the hypotheses that are consistent with training examples
  - Among these, keep one that satisfies bias
- Too slow!
- Another possibility: construct one path for each positive example
  - Not very general
- Another possibility: find smallest decision tree consistent with all examples
  - Inductive Bias: Ockham's Razor

Definition of *Occam's razor*

General: a scientific and philosophical rule that entities should not be multiplied unnecessarily which is interpreted as requiring that the simplest of competing theories be preferred to the more complex or that explanations of unknown phenomena be sought first in terms of known quantities.

In ML: **we should prefer simpler models with fewer coefficients over complex models like ensembles**. Taken at face value, the razor is a heuristic that suggests more complex hypotheses make more assumptions that, in turn, will make them too narrow and not generalize well.

In other words,it's the idea that the simplest and most direct solution should be preferred, or that with different hypotheses, the simplest one or the one with fewest assumptions will be best applied.

Occam's razor in ML

- However, Occam's razor also has some modern applications to state-of-the-art technologies – one example is the application of the principle to machine learning. With machine learning, engineers work to train computers on sets of training data, to enable them to learn and go beyond the limits of their original codebase programming. Machine learning involves implementing algorithms, data structures and training systems to computers, to allow them to learn on their own and produce evolving results.
- With that in mind, some experts feel that Occam's razor can be useful and instructive in designing machine learning projects. Some contend that Occam's razor can help engineers to choose the best algorithm to apply to a project, and also help with deciding how to train a program with the selected algorithm. One interpretation of Occam's razor is that, given more than one suitable algorithm with comparable trade-offs, the one that is least complex to deploy and easiest to interpret should be used.
- Others point out that simplification procedures such as feature selection and dimensionality reduction are also examples of using an Occam's razor principle – of simplifying models to get better results. On the other hand, others describe model trade-offs where engineers reduce complexity at the expense of accuracy – but still argue that this Occam's razor approach can be beneficial.

Top-Down Induction of Decision Trees

1. At each point, decide which attribute to use as next test in the tree
2. Attribute splits data based on answer to question
   - Each answer forms a separate node in decision tree
   - Each node is the root of an entire sub-decision tree problem, possibly with fewer examples and one fewer attribute than its parent

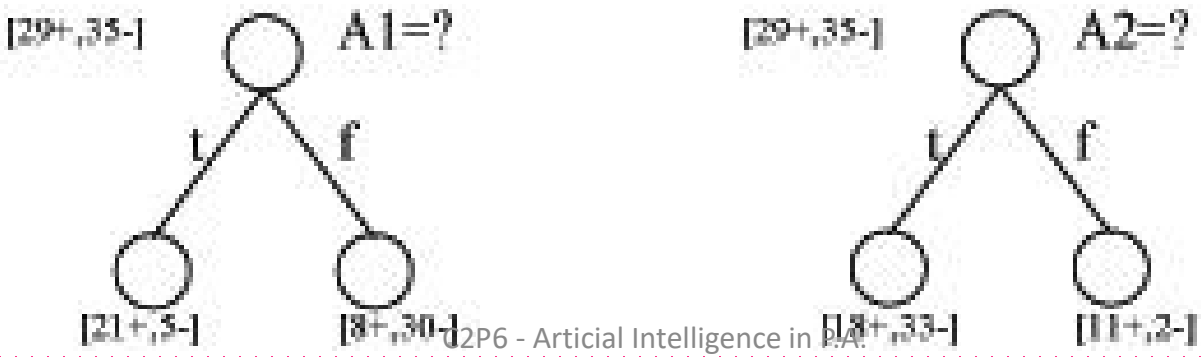
Four Cases To Consider

1. If both + and -, choose best attribute to split
2. If all + (or -), then we are done
3. If no examples, no examples fit this category, return default value (calculate using majority classification from parent)
4. If no attributes left, then there are inconsistencies, called noise

   We can use a majority vote to label the node.

Which Attribute Is Best?

- Pick one that provides the highest expected amount of information
- Information Theory measures information content in bits
- One bit of information is enough to answer a yes/no question about which one has no idea



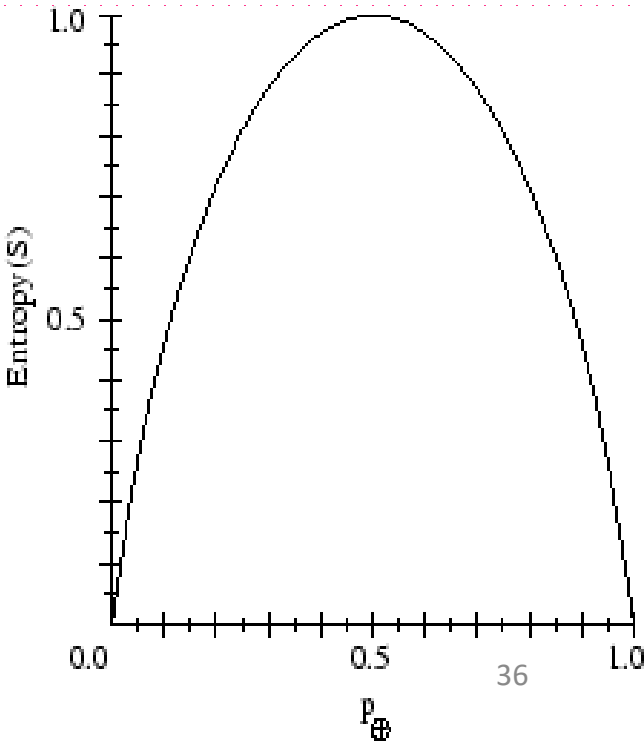
Entropy

- S is a sample of training examples
- p_+ is the proportion of positive examples in S
- p_- is the proportion of negative examples in S
- Entropy measure the impurity of S
- Entropy(S) = expected #bits needed to encode class (+ or -) of randomly drawn element of S (using optimal, shortest-length code)

Entropy

- Information theory: optimal length code assigns $-\log_2 p$ bits to message of probability p
- Expected number of bits to encode + or – of random element of S

  $p_+(-\log_2 p_+) + p_-(-\log_2 p_-)$
- Entropy(S) = $-p_+ \log_2 p_+ - p_- \log_2 p_-$



Information Content

- Entropy is also called the information content I of an actual answer
- Suppose you are sending messages to someone, could send several possible messages. How many bits are needed to distinguish which message is being sent?
- If P(message) = 1.0, don't need any bits (pure node, all one class).
- For other probability distributions, use shorter encodings for higher-probability classes. P = 0.8,0.2 requires fewer bits on average than P = 0.5, 0.5.
- Log of a fraction is always negative, so term is multiplied by -1.
- If possible answers v_i have probabilities $P(v_i)$ then the information content I of actual answer is given by

$$I((P(v_1),..,P(v_n)) = \sum_{i=1}^{n} -P(v_i)\log_2 P(v_i)$$

I(1/2, 1/2) = -(1/2 $\log_2$1/2) - (1/2 $\log_2$1/2) = 1 bit

I(0.01, 0.99) = 0.08 bits

Information Theory and Decision Trees

- What is the correct classification?
- Before splitting, estimate of probabilities of possible answers calculated as proportions of positive and negative examples
- If training set has p positive examples and n negative examples, then the information contained in a correct answer is

$$I((p/p+n), (n/p+n))$$

- Splitting on a single attribute does not usually answer entire question, but it gets us closer
- How much closer?

Information Gain

- Gain(S,A) = expected reduction in entropy due to sorting on A
- Look at decrease in information of correct answer after split

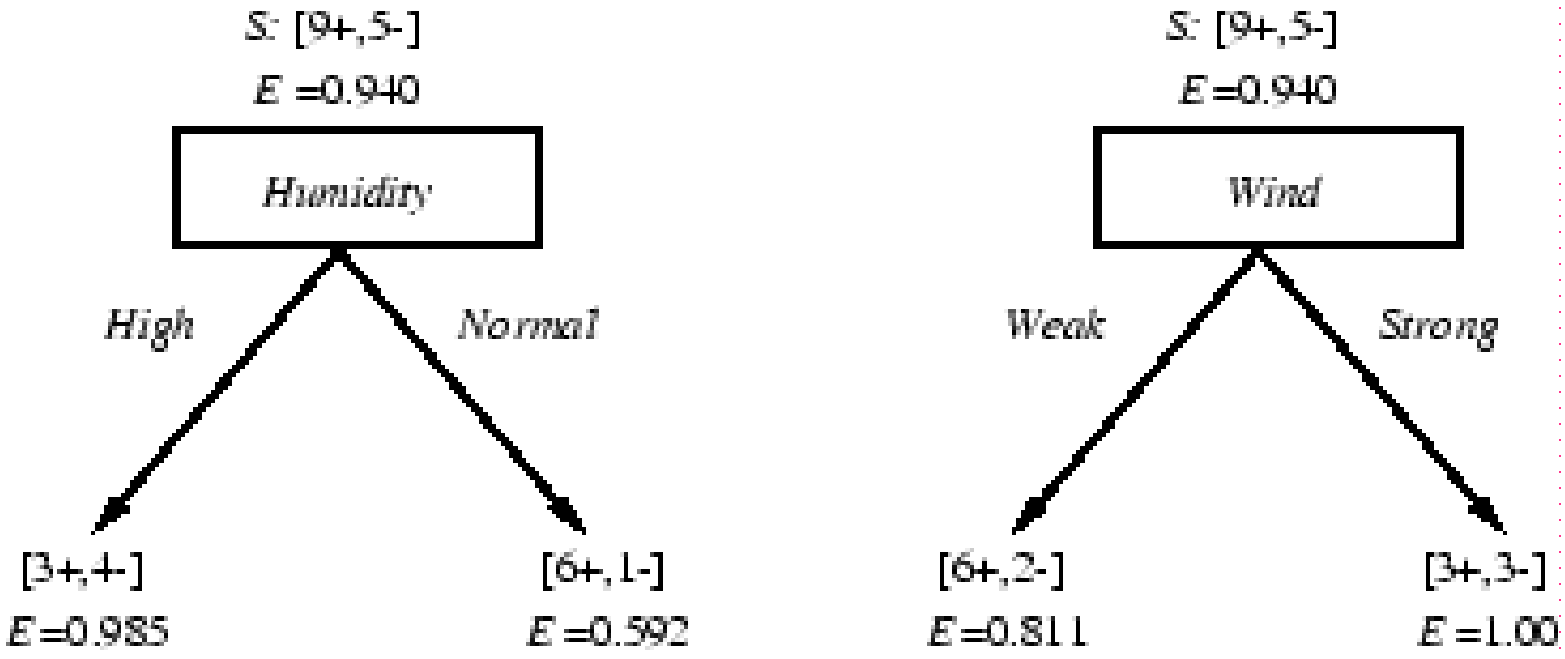
$$Gain(S,A) = Entropy(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} Entropy(S_v)$$

Training Examples

Day	Outlook	Temperature	Humidity	Wind	PlayTennis
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

Selecting the Next Attribute

Which attribute is the best classifier?



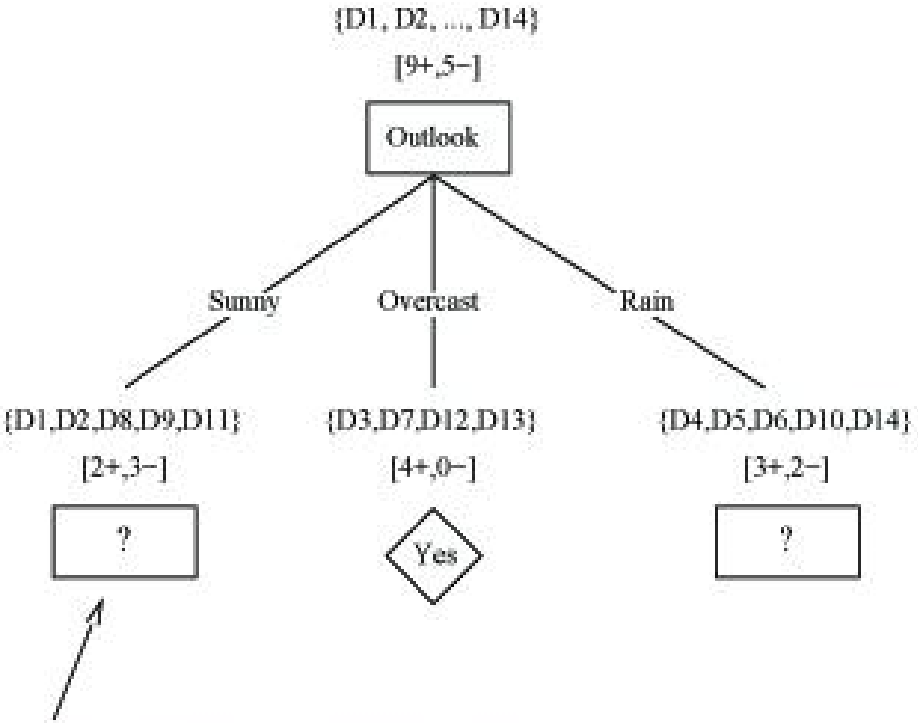
Gain (S, Humidity)

= .940 - (7/14).985 - (7/14).592
= .151

Gain (S, Wind)

= .940 - (8/14).811 - (6/14)1.0
= .048

Partially Learned Tree



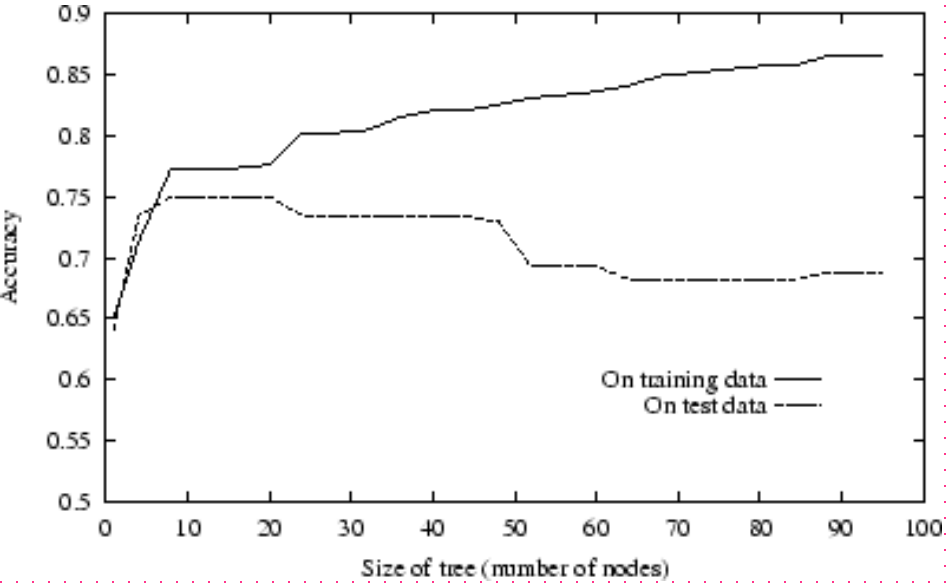
S_{sunny} = {D1,D2,D8,D9,D11}

Gain (S_{sunny} , Humidity) = .970 − (3/5) 0.0 − (2/5) 0.0 = .970

Gain (S_{sunny} , Temperature) = .970 − (2/5) 0.0 − (2/5) 1.0 − (1/5) 0.0 = .570

Gain (S_{sunny} , Wind) = .970 − (2/5) 1.0 − (3/5) .918 = .019

Danger: Overfit



One solution:
prune decision tree

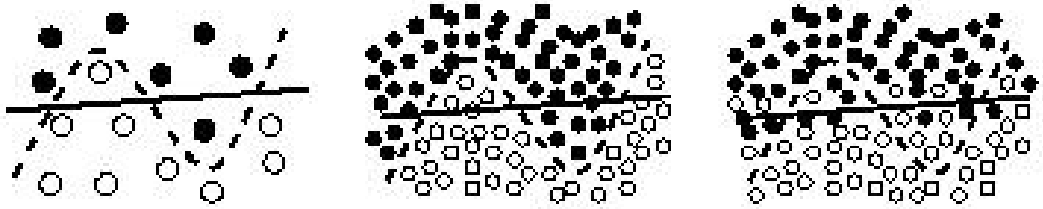


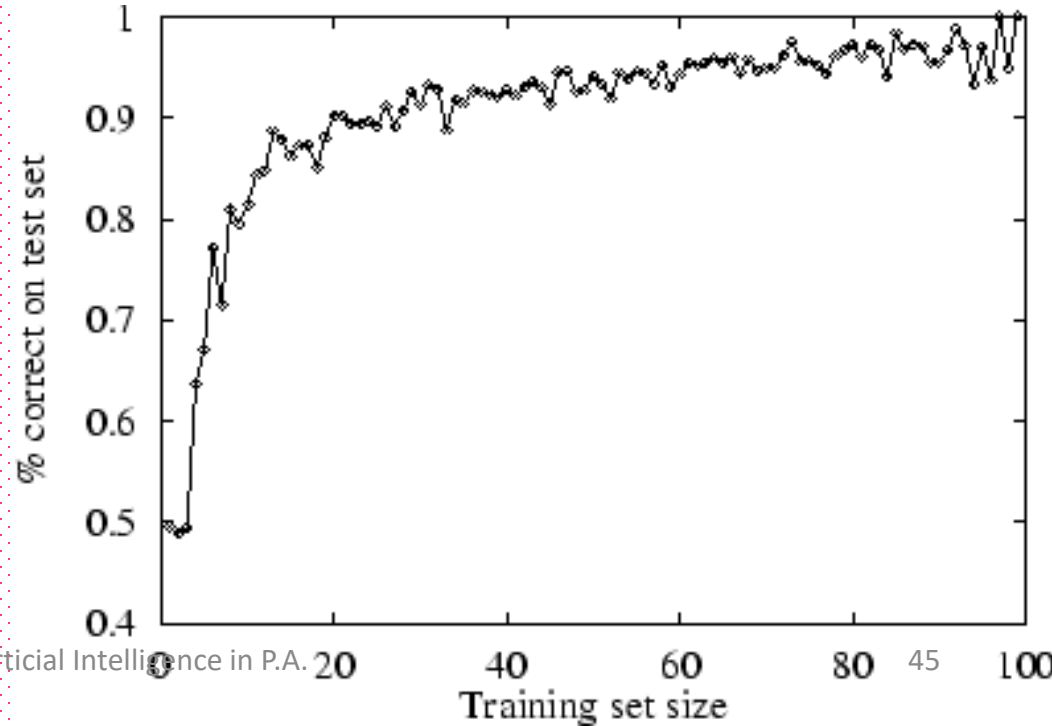
Fig. 1. Illustration of the overfitting dilemma: Given only a small sample (left) either, the solid or the dashed hypothesis might be true, the dashed one being more complex, but also having a smaller training error. Only with a large sample we are able to see which decision reflects the true distribution more closely. If the dashed hypothesis is correct the solid would underfit (middle); if the solid were correct the dashed hypothesis would overfit (right).

How Do We Prune a Decision Tree?

- Delete a decision node
- This causes entire subtree rooted at node to be removed
- Replace by leaf node, and assign it by majority vote
- Reduced error pruning: remove nodes as long as performance improves on validation set

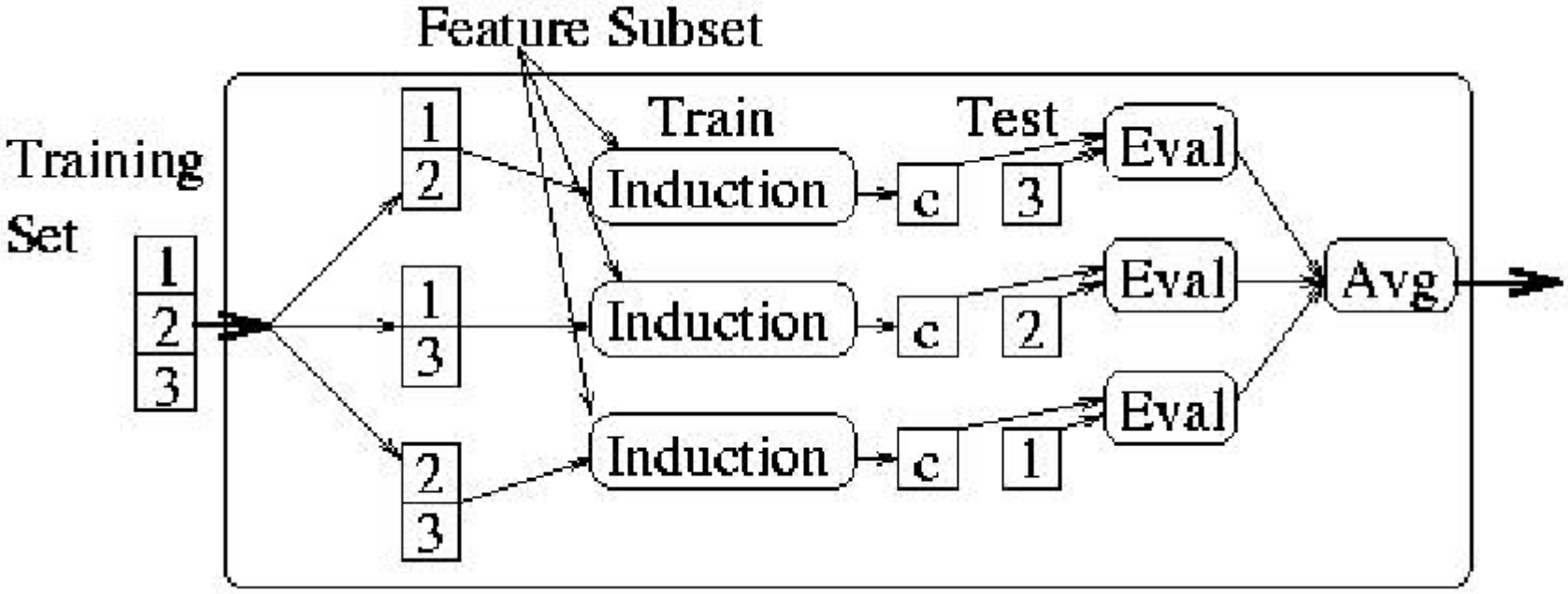
Measure Performance of a Learning Algorithm

- Collect large set of examples (as large and diverse as possible)
- Divide into 2 disjoint sets (training set and test set)
- Learn concept based on training set, generating hypothesis H
- Classify test set examples using H, measure percentage correctly classified
- Should demonstrate improved performance as training set size increases (learning curve)
- How quickly does it learn?



Measure Performance of a Learning Algorithm

- Use statistics tests to determine significance of improvement
- Cross-validation



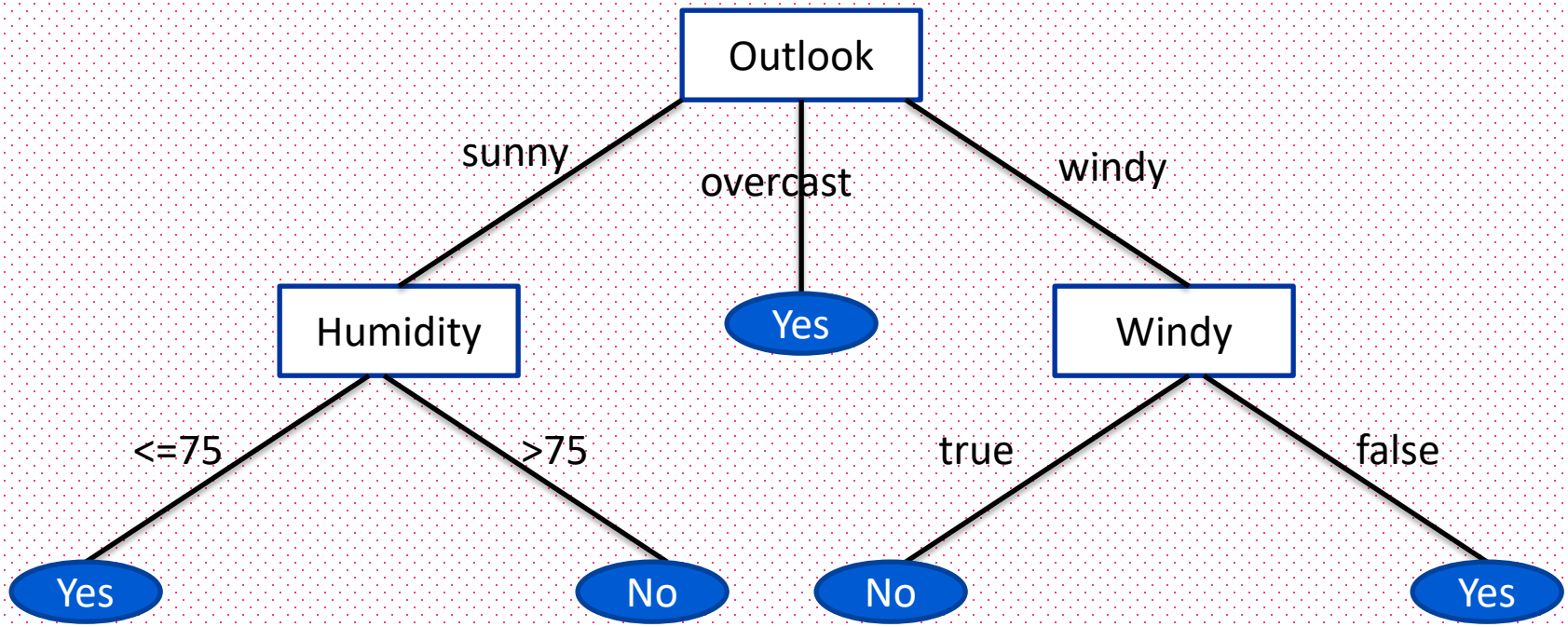
Challenges for Decision Trees

- Numeric attributes
- Missing attribute values
- Incremental updating

Training Examples

Day	Outlook	Temperature	Humidity	Wind	PlayTennis
D1	Sunny	85	85	Weak	No
D2	Sunny	80	90	Strong	No
D3	Overcast	83	86	Weak	Yes
D4	Rain	70	96	Weak	Yes
D5	Rain	68	80	Weak	Yes
D6	Rain	65	70	Strong	No
D7	Overcast	64	65	Strong	Yes
D8	Sunny	72	95	Weak	No
D9	Sunny	69	70	Weak	Yes
D10	Rain	75	80	Weak	Yes
D11	Sunny	75	70	Strong	Yes
D12	Overcast	72	90	Strong	Yes
D13	Overcast	81	75	Weak	Yes
D14	Rain	71	91	Strong	No

Decision Tree



Performance Measures

- Percentage correctly classified, averaged over folds
- Confusion matrix

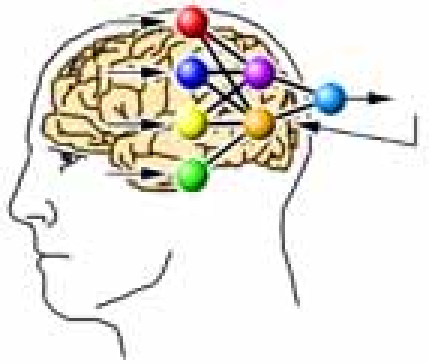
	Predicted Negative	Predicted Positive
Actual Negative	TN	FP
Actual Positive	FN	TP

- Accuracy = (TP+TN) / (TP+FP+TN+FN)
- Error = 1 - Accuracy
- Precision, Recall, ROC curves

Examples

- Bet on a basketball team
- Build decision tree
- Build decision trees

Neural Networks



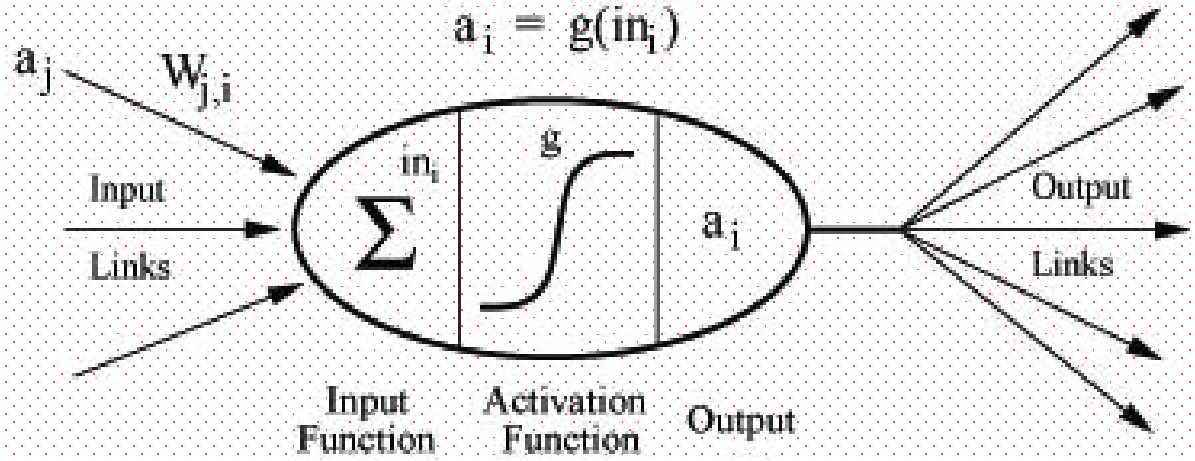
- Instead of traditional vonNeumann machines, researchers wanted to build machines based on the human brain
- An architecture as well as a learning technique
  Connection Machine
- The data structure is a network of units that act as "neurons"
- Tested as a computing device originally by researchers such as Jon Hopfield (Cal Tech), Hebb (1949), Minsky (1951), and Rosenblatt (1957)
- Also models human performance
  Voice recognition, handwriting recognition, face recognition (traditionally computers bad at these tasks, humans great)

Power In Numbers

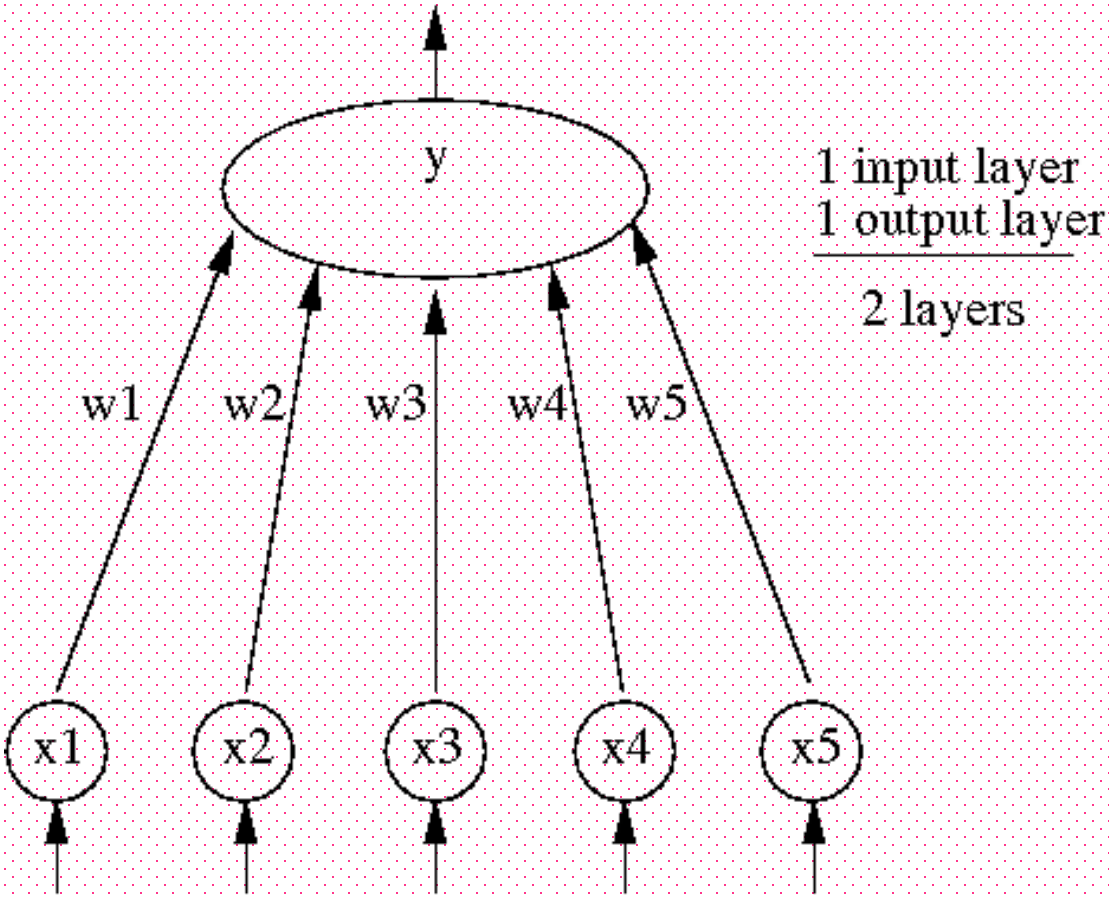
- Each neuron is not extremely powerful by itself
- Neuron switching time is ~ second
- Each message 100,000 times slower than a computer switch
- 10 billion - 1 trillion neurons
- Each neuron has 1,000 - 100,000 connections
- Computational neural networks are inspired by biology, but do not exactly imitate biology

Neuron

A neural network is made up of neurons, or processing elements



A Simple Neural Network – The Perceptron

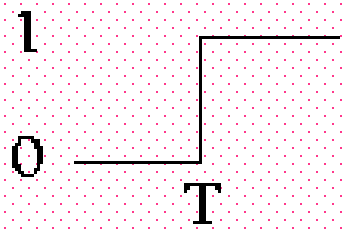


Neuron

- Each activation value x_i is weighted by w_i
- The output y is determined by the NN transfer function

$$y = \begin{cases} 1 \text{ if } \sum_{i=1}^{n} w_i x_i > \text{Threshold} \\ 0 \text{ otherwise} \end{cases}$$

- This is a step transfer function



Neural Networks Learn a Function

- y = function(x_1, x_2, ..., x_n)
- Perceptrons use one input neuron for each input parameter x_1..x_n
- y is computed using the transfer function applied to values coming in to the output node
- Suppose we are trying to learn the concept “all binary strings of length five with a 1 in the first and last positions”
- 10101 -> 1
  10100 -> 0
- 5 input units, 1 output unit
  Input units are assigned value 0 or 1
  In this case, output is 0 or 1
- Input units are always assigned a value
  If we need symbolic inputs, can map to numeric inputs (convert to binary)
- f(2 legs, brown, flat, 3' high) = chair

Applications

- Handwriting recognition
- Control problems
- Autonomous navigation
- Stock market prediction
- Image recognition
  - Alvinn drives 70 mph on highways
  - Alvinn in action

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

Two Computation Phases

1. Training phase
2. Testing / use phase

- During training, run perceptron on examples and compare network output (y) to desired output (y^d)
- Weights are adjusted after each training step using function

$$w^{new} = w^{old} + (y^d - y)x$$

- Optionally, the threshold can be adjusted as well using function

$$t^{new} = t^{old} - (y^d - y)$$

- Notice that x is the value of the input feature, thus weights are changed ONLY for nodes that are activated (used in the computation)

Parameters That Can Affect Performance

- Initial weights (can be initialized to 0, usually better if randomly set)
- Initial threshold, y = 1 if $\sum w_i * x_i > Threshold$
- Transfer function
- Learning rate, $w^{new} = w^{old} + \eta(y^d - y)x$
- Threshold update function
- Number of epochs

Learn Logical AND of x1 and x2

$$\sum w_i * x_i > Threshold$$

- Initially let w1=0, w2=0, T=0
- Epoch **1**

x1	x2	y^d
0	0	0
0	1	0
1	0	0
1	1	1

x1	x2	$w1^{old}$	$w2^{old}$	T^{old}	y	y^d	$w1^{new}$	$w2^{new}$	T^{new}
0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0
1	1	0	0	0	0	1	1	1	-1

Learn Logical AND of x1 and x2

$$\sum w_i * x_i > Threshold$$

- Epoch **2**

x1	x2	y^d
0	0	0
0	1	0
1	0	0
1	1	1

x1	x2	$w1^{old}$	$w2^{old}$	T^{old}	y	y^d	$w1^{new}$	$w2^{new}$	T^{new}
0	0	1	1	-1	1	0	1	1	0
0	1	1	1	0	1	0	1	0	1
1	0	1	0	1	0	0	1	0	1
1	1	1	0	1	0	1	2	1	0

Learn Logical AND of x1 and x2

$$\sum w_i * x_i > Threshold$$

- Epoch **3**

x1	x2	y^d
0	0	0
0	1	0
1	0	0
1	1	1

x1	x2	$w1^{old}$	$w2^{old}$	T^{old}	y	y^d	$w1^{new}$	$w2^{new}$	T^{new}
0	0	1	1	0	0	0	2	1	0
0	1	1	1	0	1	0	2	0	1
1	0	2	0	1	1	0	1	0	2
1	1	1	0	2	0	1	2	1	1

Learn Logical AND of x1 and x2

$$\sum w_i * x_i > Threshold$$

- Epoch **4**

x1	x2	y^d
0	0	0
0	1	0
1	0	0
1	1	1

x1	x2	$w1^{old}$	$w2^{old}$	T^{old}	y	y^d	$w1^{new}$	$w2^{new}$	T^{new}
0	0	2	1	1	0	0	2	1	1
0	1	2	1	1	0	0	2	1	1
1	0	2	1	1	1	0	1	1	2
1	1	1	1	2	0	1	2	2	1

Learn Logical AND of x1 and x2

$$\sum w_i * x_i > Threshold$$

- Epoch **5**

x1	x2	y^d
0	0	0
0	1	0
1	0	0
1	1	1

x1	x2	$w1^{old}$	$w2^{old}$	T^{old}	y	y^d	$w1^{new}$	$w2^{new}$	T^{new}
0	0	2	2	1	0	0	2	1	1
0	1	2	2	1	1	0	2	1	2
1	0	2	1	2	0	0	2	1	2
1	1	2	1	2	1	1	2	1	2

Learn Logical AND of x1 and x2

$$\sum w_i * x_i > Threshold$$

- Epoch **6**

x1	x2	y^d
0	0	0
0	1	0
1	0	0
1	1	1

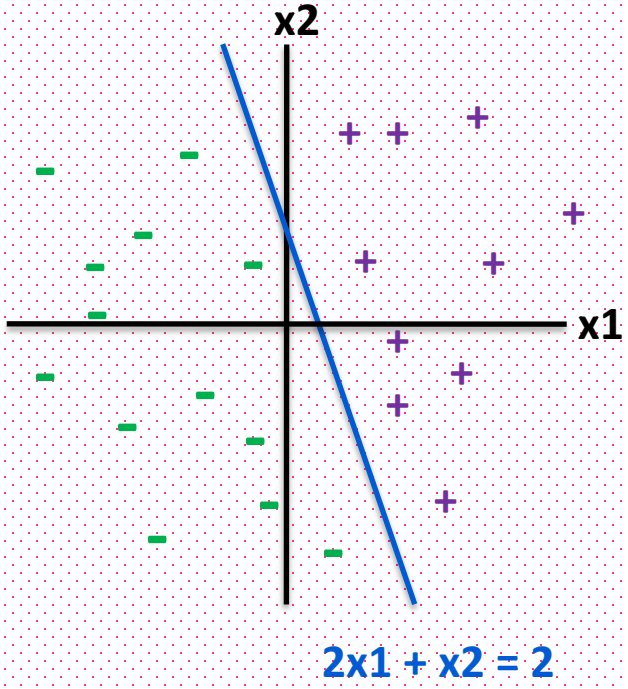
x1	x2	$w1^{old}$	$w2^{old}$	T^{old}	y	y^d	$w1^{new}$	$w2^{new}$	T^{new}
0	0	2	1	2	0	0	2	1	2
0	1	2	1	2	0	0	2	1	2
1	0	2	1	2	0	0	2	1	2
1	1	2	1	2	1	1	2	1	2

CONVERGENCE!

The AND Function

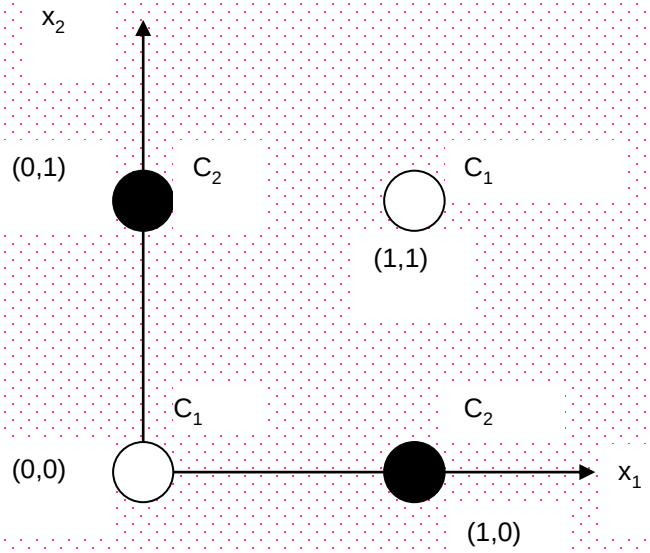
- Notice that the classes can be separated by a line (hyperplane)

+ is Class 1
- Is Class 2



- Learn XOR of two inputs, x1 and x2
  - w1=2, w2=1, Threshold=2
- Why does the perceptron run forever and never converge?

Learn XOR of two inputs, x1 and x2
w1=2, w2=1, Threshold=2
Why does the perceptron run forever and never converge?

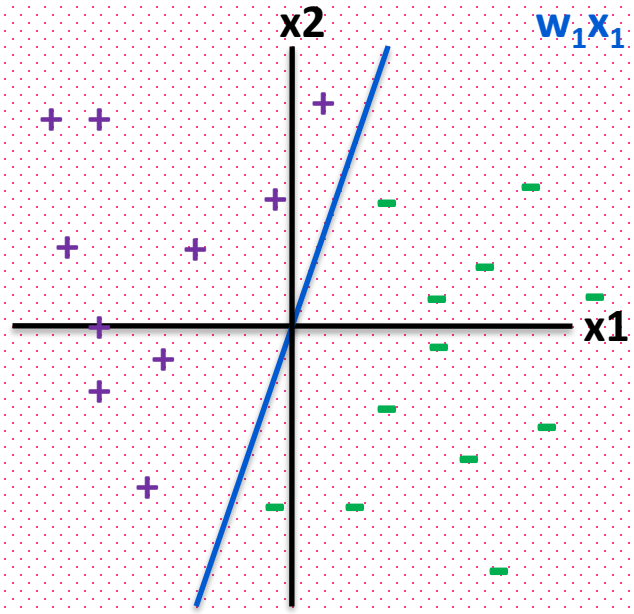


Function Learned By Perceptron

- The final network can be expressed as an equation of the parameters x_1 through x_n
- $w_1 x_1 + w_2 x_2 + \ldots + w_n x_n = \text{Threshold}$
- If learned, the network can be represented as a hyperplane

+ is Class 1

- Is Class 2



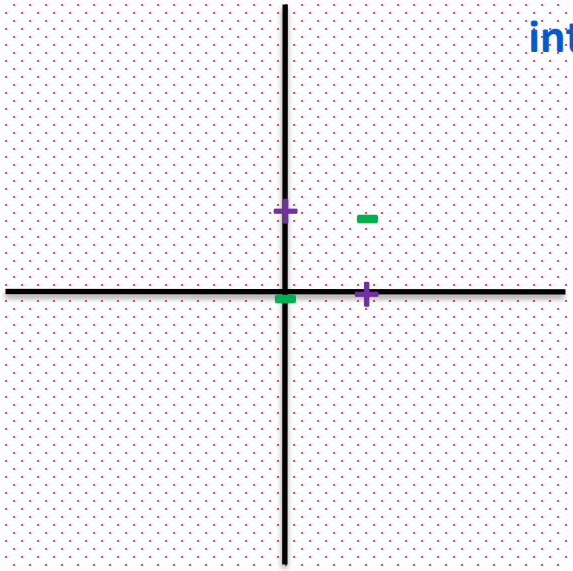
Examples

- Perceptron Example
- Perceptron Example

Linearly Separable

- If the classes can be separated by a hyperplane, then they are linearly separable.
- Linearly Separable Learnable by a Perceptron
- Here is the XOR space:

No line can separate these data points into two classes – need two lines

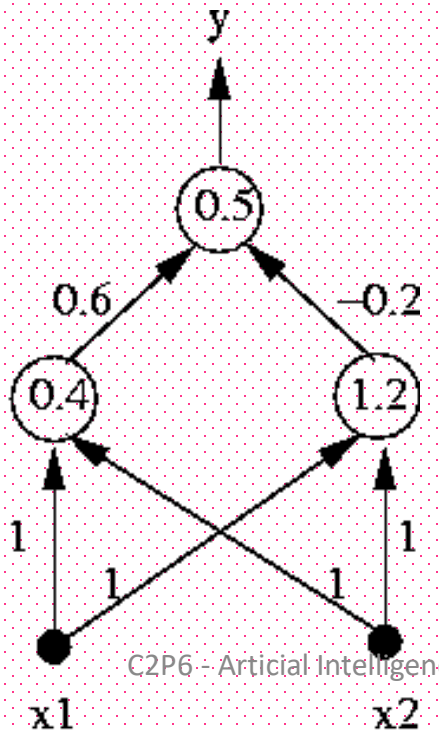


How Can We Learn These Functions?

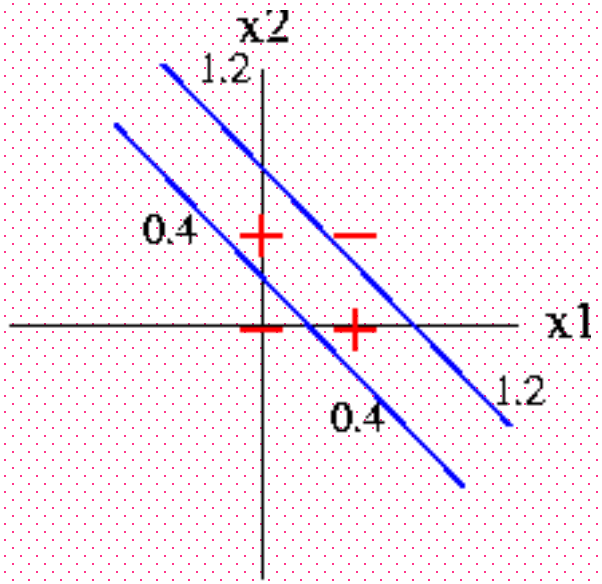
- Add more layers with more neurons!
- Features of perceptrons:
  - Only 1 neuron in output layer
  - Inputs only 0 or 1
  - Transfer function compares weighted sum to threshold
  - No hidden units
  - Output is only 0 or 1

Multilayer Neural Networks

- 1 input layer (2 units), 1 hidden layer (2 units), 1 output layer (1 unit)
- Like before, output is a function of the weighted input to the node



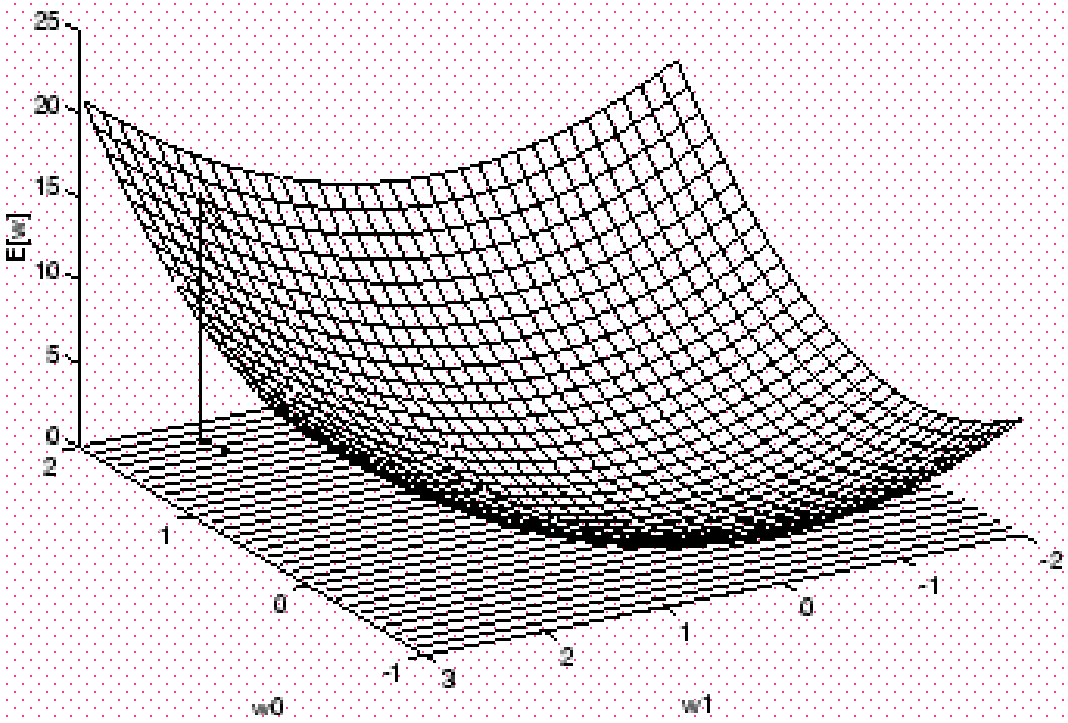
Function Learned by MNN



Learning in a Multilayer Neural Network

- How should we change (adapt) the weights in a multilayer neural network?
- A perceptron is easy - direct mapping between weights and output.
- Here, weights can contribute to intermediary functions and only indirectly affect output. These weights can indirectly affect multiple output nodes.
- If output is 12 and we want a 10, change weights to output node so that output next time would be (closer to) desired value 10.
- How do we change weights to hidden units?
- Assign portion of error to each hidden node, change weights to lessen that error next time.

Weight Learning Using Gradient Descent



Deriving an Update Function

Our goal is to reduce error (often *sum of squared errors*), which is

$$E = \frac{1}{2} Err^2 = \frac{1}{2}(t-o)^2$$

Since the gradient specifies direction of steepest increase of error, the training rule for gradient descent is to update each weight by the derivative of the error with respect to each weight, or

$$\frac{\partial E}{\partial W_j} = Err * \frac{\partial Err}{\partial W_j}$$

The derivative of a particular weight is $Err \times \frac{\partial}{\partial W_j}(t - g(\Sigma_{j=0}^{n} W_j a_j)) = -Err \times g'(in) \times a_j$

where g'(in) is the derivative of the transfer function & a_j is the activation value at source node j.

We want to eliminate the error when we adjust the weights, so we multiply the formula by -1.
We want to constrain the adjustment, so we multiply the formula again by the learning rate η.

The result is the **Delta Rule**.

Delta Rule

- Weight update for hidden-to-output links:

$$w_{ji} = w_{ji} + (\eta * a_j * Err_i * g'(in_i))$$

w_{ji} = Weight of link from node j to node i
η = Learning rate,
a_j = Activation of node j (output of hidden node j or input for input node j)
Err_i = Error at this node (target minus actual output, total weight change needed to node i)
$g'(in_i)$ = Derivative of the transfer function g

Same general idea as before. If error is positive, then network output is too small so weights are increased for positive inputs and decreased for negative inputs. The opposite happens when the error is negative.

Hidden-to-output Weights

$$w_{ji} = w_{ji} + (\eta * a_j * Err_i * g'(in_i))$$

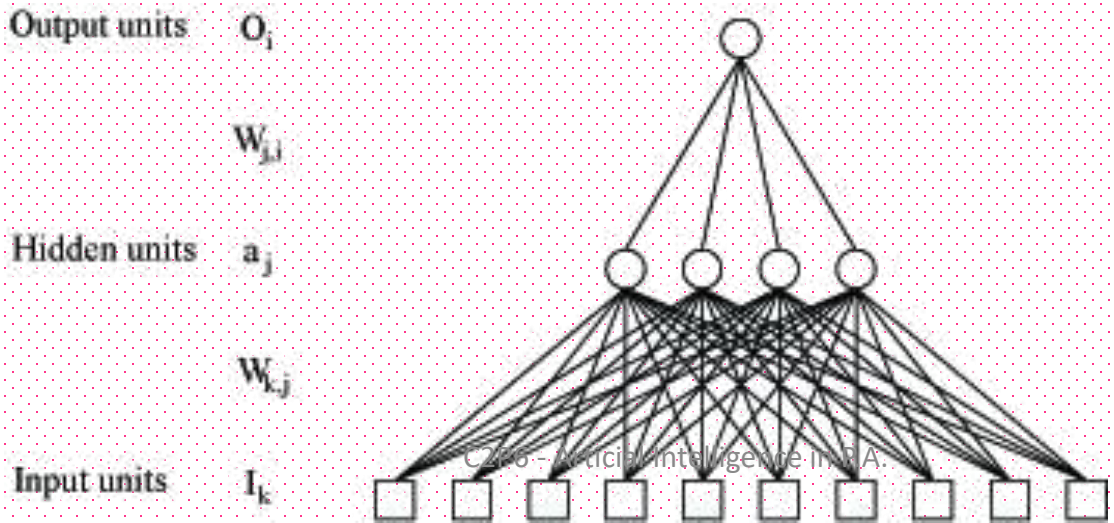
Let $\Delta_i = Err_i * g'(in_i)$ represent the error term

$$\Delta_i = (t_i - o_i) * g'(in_i)$$

t_i = true / target output for node i
o_i = actual / calculated output for node i
in_i = sum of inputs
g' = derivative of the transfer function

Next Layer

- For input-to-hidden weights, we need to define a value analogous to the error term for output nodes. Here we perform error backpropagation. Each hidden node is assigned a portion of the error corresponding to its contribution to the output node.
- The formula $\Delta_j = g'(in_j)\Sigma_i w_{ji}\Delta_i$
  - Assigns a portion of the responsibility to node j
  - The proportion is determined by the weight from j to all output nodes. Now we can give the weight update rule for links from input to hidden nodes.
- For each input node k to hidden node j use $w_{kj} = w_{kj} + (\eta * I_k * \Delta_j)$

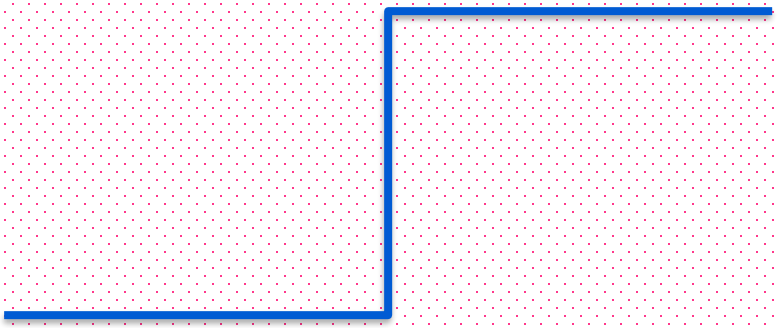


Update Function

- Why use the derivative of the transfer function in the calculation of delta?
- Note the visualization of the weight space using gradient descent.
- We want to move the weights in the direction of steepest descent in this space.
- To do this, compute derivative of the error with respect to each weight in the equation. This results in the delta terms showed earlier.
- Note that the transfer function must be differentiable everywhere.

Step Function

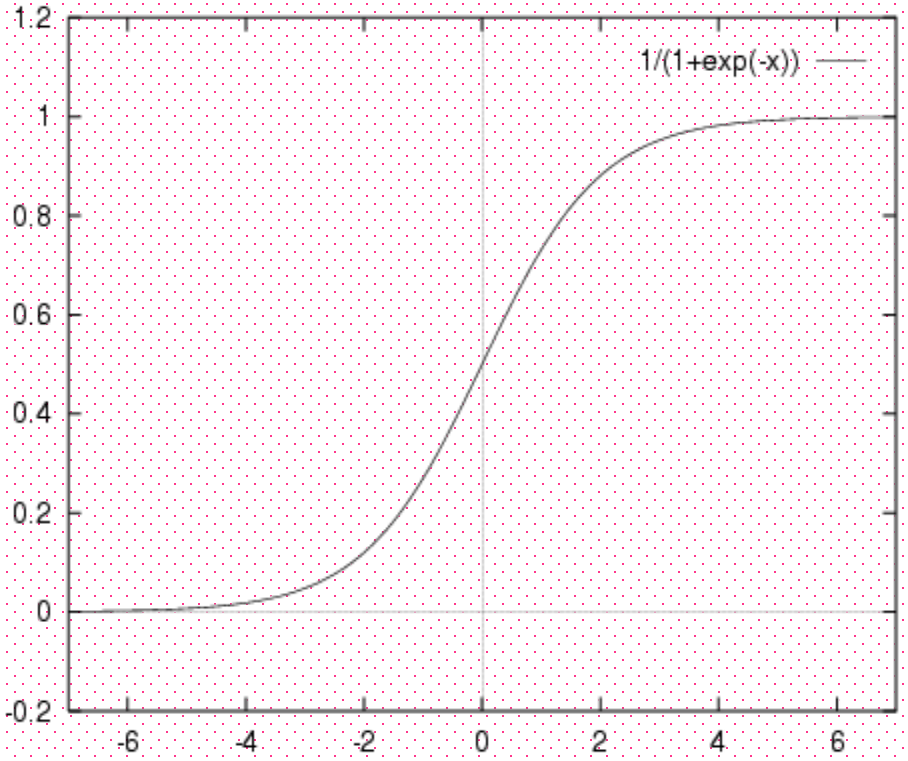
- Our perceptron transfer function will not work



derivative = infinity here

Sigmoid Function

$$g(x) = \frac{1}{1 - e^{-x}}$$ where x is weighted sum of inputs



The sigmoid function is handy to use for backpropagation because its derivative is easily computed as g(x)*(1-g(x)).

Sigmoid Update formula

- To calculate output, for each node in network
  - Calculate weighted sum of inputs, $in_i = \Sigma_j w_{ji} a_j$
  - Compute output or activation of node, $a_i = sigmoid(in_i)$
- For each node i in output layer

$$\Delta_i = sigmoid(in_i) * (1 - sigmoid(in_i)) * (t_i - a_i)$$

- For each node j in lower layers

$$\Delta_j = sigmoid(in_j) * (1 - sigmoid(in_j)) * \Sigma_i w_{ji} \Delta_i$$

- Update weight w_{ji} by $w_{ji} = w_{ji} + \eta * a_j * \Delta_i$

NN Applications - NETtalk

- Sejnowski and Rosenburg, 1985
- Written English text to English speech
- Based on DECtalk expert system
- Look a a window of 7 characters:

T H I S I S (A-Z, ",", "." " ")

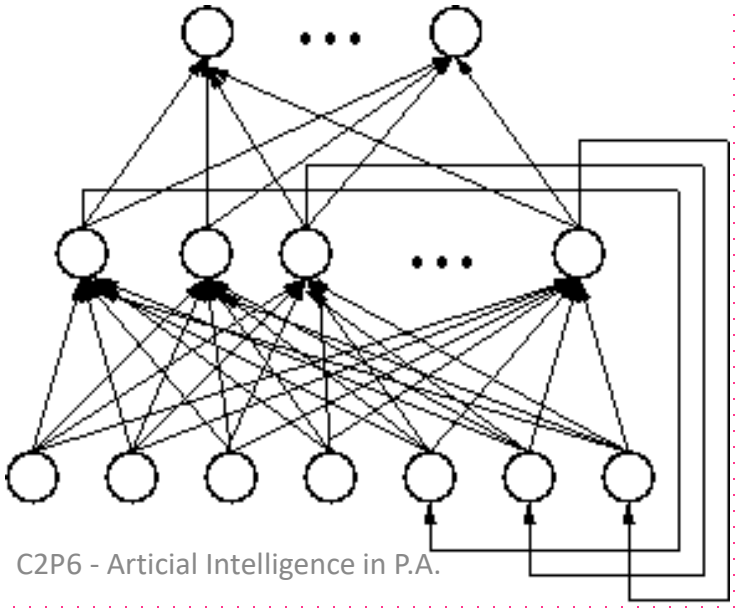
- Decide how to utter middle character
- 1 network
  - 1 hidden layer
  - 203 input units (7 character window * 29 possible characters)
  - 80 hidden units
  - Approximately 30 output units

Examples

- Handwriting Recognition
- Balancing Ball
- Learn 3D Map

Networks That Deal With Time

- In the feedforward networks we have been studying, transfer functions capture the network state (not usually spatiotemporal in nature).
- Recurrent neural networks feed signal back from output to network (output to hidden, hidden to hidden, hidden to input, others).
- In this way they can learn a function of time.
- $y(t) = w1\ x1(t) + w2\ x2(t) + \ldots + wn+1\ x1(t-1) + wn+1\ x2(t-1) + \ldots$

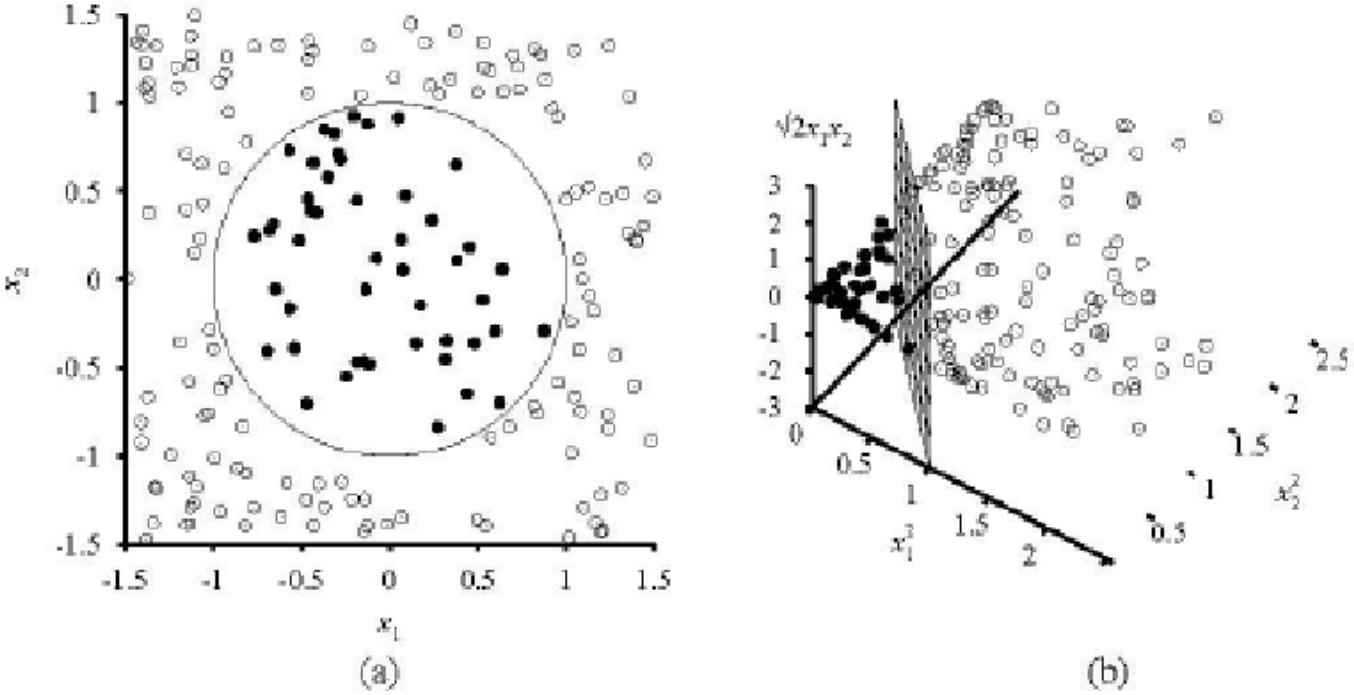


Neural Network Issues

- Usefulness
- Generalizability
- Understandability (cannot explain results)
- Self-structuring neural networks
  Add/delete nodes and links until error is minimized
- Networks and expert systems
  Can we learn rules corresponding to network?
- Input background knowledge
  Predefined structure, weights
- Computational complexity
  Blum and Rivest in 1992 proved that training even a three-node network is NP Complete
- Inductive bias is smooth interpolation between data points

Extending the Idea of Linear Separability

- No plane separates examples on the left



- We can, however, map the figures onto three *new* features

$$f_1 = x_1^2,\ f_2 = f_2^2,\ f_3 = \sqrt{2}x_1x_2$$

- and the data in the new space is separable as shown on the right.
- We can search for such mappings that leave maximal margins between classes. This is learning using support vector machines .

Reinforcement Learning

- Learn action selection for probabilistic applications
  - Robot learning to dock on battery charger
  - Learning to choose actions to optimize factory output
  - Learning to play Backgammon
- Note several problem characteristics:
  - Delayed reward
  - Opportunity for active exploration
  - Possibility that state only partially observable
  - Possible need to learn multiple tasks with same sensors/effectors

Reinforcement Learning

- Learning an optimal strategy for maximizing future reward
- Agent has little prior knowledge and no immediate feedback
- Action credit assignment difficult when only future reward
- Two basic agent designs
  - Agent learns utility function U(s) on states
    - Used to select actions maximizing expected utility
    - Requires a model of action outcomes (T(s,a,s'))
  - Agent learns action-value function
    - Gives expected utility Q(s,a) of action a in state s
    - Q-learning Q(s,a)
    - No action outcome model, but cannot look ahead
- Can handle deterministic or probabilistic state transitions

Subtleties and Ongoing Research

- Exploration vs. exploitation
- Scalability
- Generalize utilities from visited states to other states (inductive learning)
- Design optimal exploration strategies
- Extend to continuous actions, states

References

- https://eecs.wsu.edu/~cook/ai/lectures/p.html
- (Lecture 9)