
Machine Learning

What is learning?

Common Definitions

Webster: 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

Simon: Any process by which a system improves its performance

So far we have programmed knowledge into the agent (expert system rules, probabilities, database facts and rules, search space representations), but a fully autonomous agent should acquire this knowledge on its own.

Machine Learning will make this possible

Categories of Learning

Learning by being told

Learning by examples / Supervised learning

Learning by experimentation / Unsupervised learning

A General Model of Learning Agents

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

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

Prediction Problems

Customer purchase behavior

Prediction Problems

Customer retention

Prediction Problems

Prediction Problems

Problems too difficult to program by hand

Prediction Problems

Software that customizes to user



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

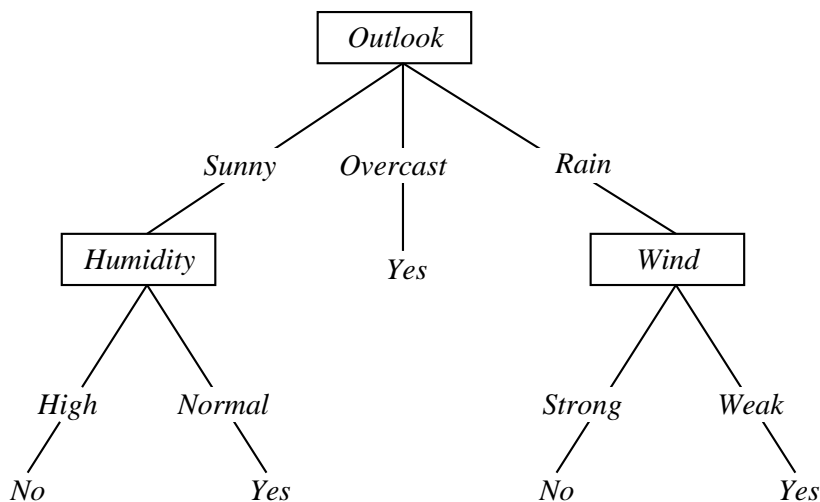
Each learning algorithm has an **inductive bias** that imposes a preference on the space of all possible hypotheses

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 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
-

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.
-

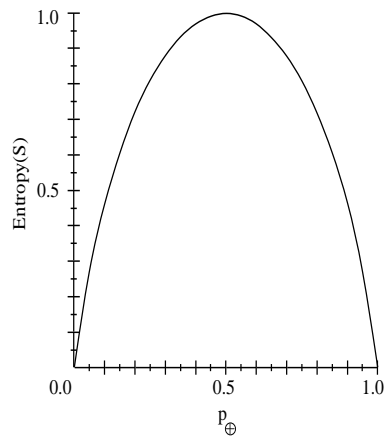
Which Attribute Is Best?

Pick one that provides the highest expected amount of **information**

Information Theory measures information content in bits (not dollars)

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_{\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
- $Entropy(S)$ = expected number of bits needed to encode class (\oplus or \ominus) of randomly drawn element of S (under the optimal, shortest-length code)
- Information theory: optimal length code assigns $-\log_2 p$ bits to message of probability p .

So, expected number of bits to encode \oplus or \ominus of random element of S :

$$p_{\oplus}(-\log_2 p_{\oplus}) + p_{\ominus}(-\log_2 p_{\ominus})$$

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

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 $P(\text{message}) = 1.0$, don't need any bits (pure node, all one class). If 10 messages, P of each is 0.1, need many bits (node with an even number of examples in each possible class). 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

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

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

$$I(0.01, 0.99) = 0.08 \text{ bits}$$

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.

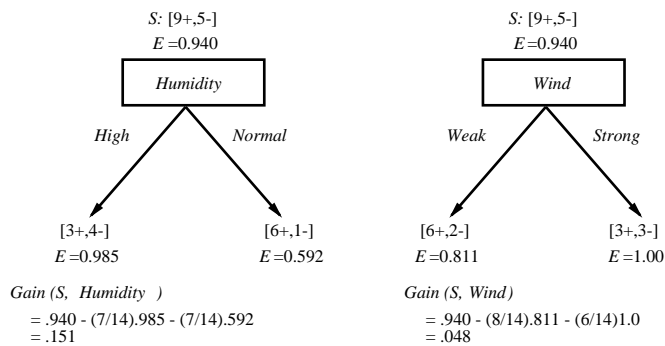
$$Gain(S, A) \equiv Entropy(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} Entropy(S_v)$$

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

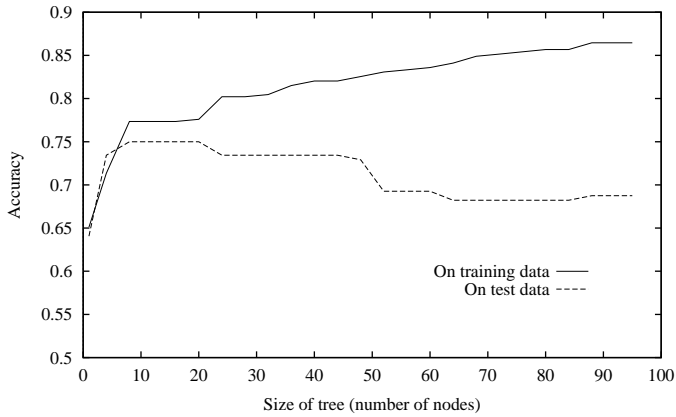
Selecting the Next Attribute

Which attribute is the best classifier?



Partially Learned Tree

Danger: Overfit



One solution: prune decision tree

Measure Performance of a Learning Algorithm

Procedure

Collect large set of examples (as large and diverse as possible)

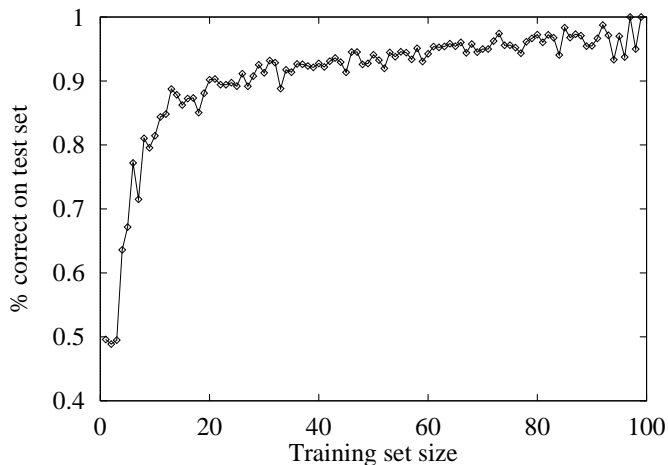
Divide into 2 disjoint sets (training set and test set)

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)

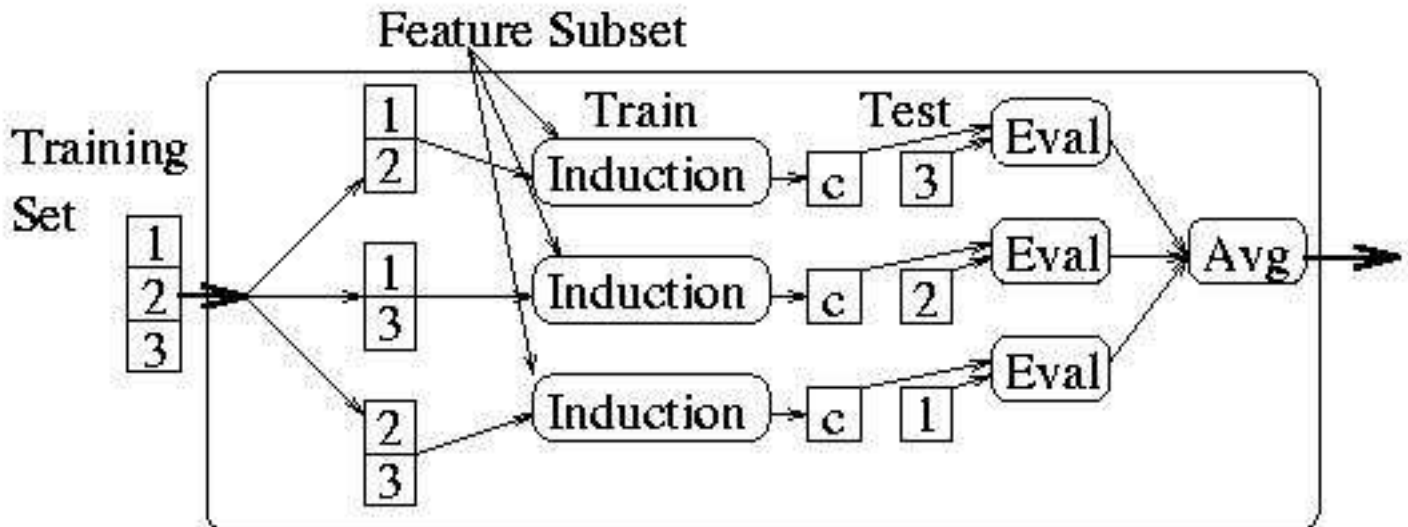
How quickly does it learn?



Measure Performance of a Learning Algorithm

Use statistics tests to determine significance of improvement

Cross-validation



Example

Restaurant concept

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

Suppose only considering attributes Patrons, Type, Hungry, Fri/Sat

$$\begin{aligned} &= \text{Original} - [\text{None} + \text{Some} + \text{Full}] \\ \text{Gain}(\text{Patrons}) &= 1 - [2/12 I(0,1) + 4/12 I(1,0) + 6/12 I(2/6,4/6)] \\ &= 0.541 \text{ bits} \end{aligned}$$

$$\begin{aligned} &= \text{Original} - [\text{French} + \text{Italian} + \text{Thai} + \text{Burger}] \\ \text{Gain}(\text{Type}) &= 1 - [2/12 I(1/2,1/2) + 2/12 I(1/2,1/2) + \\ &\quad 4/12 I(2/4,2/4) + 4/12 I(2/4,2/4)] = 0 \text{ bits} \end{aligned}$$

$$\begin{aligned} I(2/6,4/6) &= .92 \\ \text{Gain}(\text{Type}) &= .92 - [1/6I(0,1) + 1/6I(0,1) + 2/6I(1,1) + 2/6I(1,1)] \\ &= .92 - 4/6 = .253 \\ \text{Gain}(\text{Hungry}) &= .92 - [4/6I(1,1) + 2/6I(0,1)] = .92 - 4/6 = .253 \\ \text{Gain}(\text{Fri/Sat}) &= .92 - [5/6I(2/5,3/5) + 1/6I(0,1)] = .92 - ? < .253 \\ &\text{Choose attribute Hungry} \end{aligned}$$

$$\begin{aligned} I(2/4,2/4) &= 1 \\ \text{Gain}(\text{Type}) &= 1 - [0 + 1/4I(0,1) + 2/4I(1,1) + 1/4I(1,0)] = 1 - .5 = .5 \\ \text{Gain}(\text{Fri/Sat}) &= 1 - [3/4I(2/3,1/3) + 1/4I(0,1)] = 1 - .75*.92 = .31 \\ &\text{Choose attribute Type} \end{aligned}$$

$$\begin{aligned} I(1/2,1/2) &= 1 \\ \text{Gain}(\text{Fri/Sat}) &= 1 - [1/2I(1,0) + 1/2I(0,1)] = 1 - 0 = 1 \end{aligned}$$

Examples

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 $\sim .001$ 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 connection 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

$y = \text{function}(x_1, x_2, \dots, x_n)$

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 \rightarrow 1

10100 \rightarrow 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)

$f(2 \text{ legs, brown, flat, 3' high}) = \text{chair}$

Applications

Handwriting recognition

Speech recognition

Autonomous navigation

Stock market prediction

Image recognition

Alvinn drives 70 mph on highways

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 P's 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

Bias n ($w^{new} = w^{old} + n(y^d - y)x$) or learning rate

Threshold update function

Number of epochs

Perceptron Code

```
;;; PERCEPTRON is a simple system for learning from examples which uses the
;;; perceptron learning procedure to adjust a set of weights on a single
;;; linear threshold unit until all of the training examples are correctly
;;; classified. The perceptron convergence theorem assures that the system
;;; will halt if the examples are linearly separable but if not the system
;;; may not halt. The file BINARY-ENCODER contains the functions needed
;;; for converting feature vector examples to bit strings
;;; To run on multi-category problems like SOYBEAN-DATA, after loading the data
;;; you must encode instances of all categories into bit strings using
;;; ENCODE-CATEGORY-INSTANCES (e.g. (encode-category-instances soybean-cat
;;; and then partition encoded examples into training and test sets using
;;; SEPARATE-INSTANCES and then use PERCEPTRON-CATEGORIES to do the
;;; learning and TEST-CATEGORIES to test the learned perceptron.
```

```
;;; Uses functions defined in the files: BINARY-ENCODER and TESTER
```

```
;;; Copyright (c) 1988 by Raymond Joseph Mooney. This program may be freely
;;; copied, used, or modified provided that this copyright notice is included
;;; in each copy of this code and parts thereof.
```

```
(defvar *trace-perceptron* t) ; produces trace of weight updates if T.
(defvar *perceptron* nil) ; Stores the final learned perceptron.
(defvar *domains*) ; List of domains (possible value list) for
; each feature
(defvar *eta* 1) ; Learning rate

(setf testor '(((- (0 0)) ; Logical OR
(+ (0 1))
(+ (1 0))
(+ (1 1))))))

(setf test1 '((+ (0 1 0)) ; Simple testing example
(+ (1 0 1))
(+ (1 1 1))
(- (0 0 1))))

(setf test2 '(((- (0 0)) ; Infamous XOR example
(+ (0 1))
(+ (1 0))
(- (1 1))))))

(setf test3 '(((- (0 0)) ; Logical AND
(- (0 1))
```

```
(- (1 0))
(+ (1 1)))
```

```
(defmacro trace-print (test-var &rest format-form)
  ;; Print using the format string only if test-var is nonNIL
  `(if ,test-var
      (format t ,@format-form)))

(defun perceptron (examples &optional (threshold 0))
  (let* ((num-features (length (second (first examples))))
        (weights (make-array (list num-features) ; define weight vector
                              :element-type 'number
                              :initial-element 0)) ; weights initialized to 0
        (all-correct nil) (i 0) (trial-num 0))
    (when *trace-perceptron* (print-perceptron weights threshold))
    (loop (if all-correct (return nil)) ; Loop until all examples are correctly
          (setf all-correct t) ; classified.
          (dolist (example examples) ; Each trial look at all examples
            (if (compute-perceptron-output (second example) weights threshold)
                (cond ((eq (first example) '-') ; If network says + but its -
                      (trace-print *trace-perceptron*
                                    "~%%Classifies ~A wrong" example)
                      (setf all-correct nil)
                      (incf threshold *eta*) ; Then increase threshold to
                                             ; make + classification harder
                      ;; and decrement weights for features present in example
                      (setf i 0)
                      (dolist (feature-value (second example))
                        (when (eq feature-value 1)
                          (incf (aref weights i) (- *eta*))
                          (trace-print *trace-perceptron*
                                        "~%%Decrementing weight for feature ~A" (- i 1)))
                          (incf i)))
                      (t (trace-print *trace-perceptron*
                                       "~%%Classifies ~A right" example)))
                (cond ((eq (first example) '+) ; If network says - but its +
                      (trace-print *trace-perceptron*
                                    "~%%Classifies ~A wrong" example)
                      (setf all-correct nil)
                      (incf threshold (- *eta*)) ; Then decrease threshold to
                                             ; make + classification easier
                      ;; and increment weights for features present in example
                      (setf i 0)
                      (dolist (feature-value (second example))
                        (when (eq feature-value 1)
```

```

                (incf (aref weights i) *eta*)
                (trace-print *trace-perceptron*
                 "%Incrementing weight for feature ~A" (+ i 1)))
            (incf i))
        (t (trace-print *trace-perceptron*
            "%~%Classifies ~A right" example))))
    (incf trial-num) ; Keep track of the number of trials
    (when *trace-perceptron* (print-perceptron weights threshold))
    (format t "~%Trials: ~A" trial-num)
    (unless *trace-perceptron* (print-perceptron weights threshold))
    (setf *perceptron* (list weights threshold))) ; Return the final perceptron

(defun compute-perceptron-output (feature-values weights threshold)
  ;;; Determine value of perceptron for the given input. Return T or NIL
  ;;; instead of 0 or 1 to simply tests

  (let ((sum 0) (i 0))
    ;; Simply sum the weight*input for all of the features
    ;; and return T if greater than threshold.
    (dolist (feature-value feature-values)
      (when (eq feature-value 1)
        (incf sum (aref weights i)))
      (incf i))
    (> sum threshold)))

(defun print-perceptron (weights threshold)
  ;;; Printout the current weight vector and threshold

  (format t "%~%Weights:")
  (dotimes (i (length weights))
    (format t " ~A" (aref weights i)))
  (format t "%~%Threshold: ~A" threshold))

```

Example

```

(setf test1 '((+ (0 1 0)) (+ (1 0 1)) (+ (1 1 1)) (- (0 0 1))))

(perceptron test1)

Weights: 0 0 0
Threshold: 0

```

Classifies (+ (0 1 0)) wrong
Incrementing weight for feature 2

Classifies (+ (1 0 1)) right

Classifies (+ (1 1 1)) right

Classifies (- (0 0 1)) wrong
Decrementing weight for feature 3
Weights: 0 1 -1
Threshold: 0

Classifies (+ (0 1 0)) right

Classifies (+ (1 0 1)) wrong
Incrementing weight for feature 1
Incrementing weight for feature 3
Classifies (+ (1 1 1)) right

Classifies (- (0 0 1)) wrong
Decrementing weight for feature 3

Weights: 1 1 -1
Threshold: 0

Classifies (+ (0 1 0)) right

Classifies (+ (1 0 1)) wrong
Incrementing weight for feature 1
Incrementing weight for feature 3

Classifies (+ (1 1 1)) right

Classifies (- (0 0 1)) wrong
Decrementing weight for feature 3

Weights: 2 1 -1
Threshold: 0

Classifies (+ (0 1 0)) right

Classifies (+ (1 0 1)) right

Classifies (+ (1 1 1)) right

Classifies (- (0 0 1)) right

Weights: 2 1 -1

Threshold: 0

Trials: 4

(#(2 1 -1) 0)

Example

Learn AND of two inputs, x1 and x2.

Initially, let $w_1 = 0$, $w_2 = 0$, $T = 0$, $\eta = 1$

If $(x_1w_1 + x_2w_2) > T$ then $y=1$ else $y=0$

```
x1 x2 | y^d
-----|-----
0  0 | 0
0  1 | 0
1  0 | 0
1  1 | 1
```

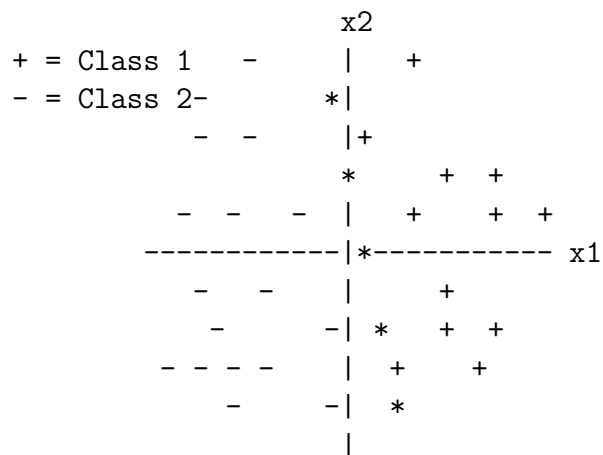
x1	x2	w1 ^{old}	w2 ^{old}	T ^{old}	y	y ^d	w1 ^{new}	w2 ^{new}	T ^{new}
0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0
1	1	0	0	0	0	1	1	1	-1
----- -----									
0	0	1	1	-1	1	0	1	1	0
0	1	1	1	0	1	0	1	0	1
1	0	1	0	1	0	0	1	0	1
1	1	1	0	1	0	1	2	1	0
----- -----									
0	0	1	1	0	0	0	2	1	0
0	1	1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

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	0	2	0	1	2	2	1
-----								-----	
0	0	2	2	1	0	0	2	2	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
-----								-----	
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

$w_1 = 2, w_2 = 1, \text{threshold} = 2$



$$2x_1 + x_2 = 2$$

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

Learn XOR of two inputs, x1 and x2.

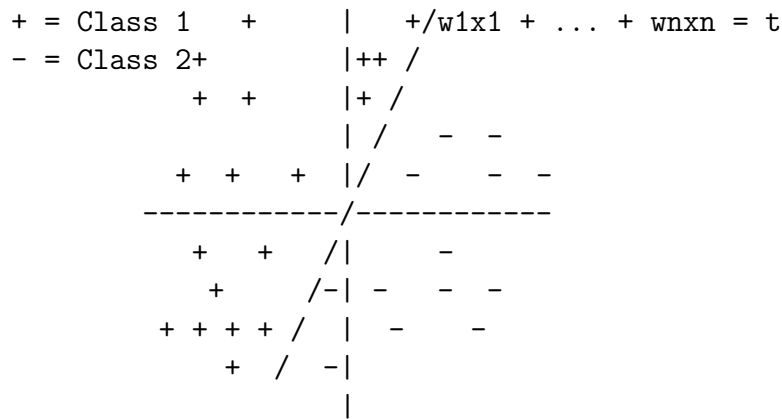
Why does perceptron run forever and never converge?

Function Learned By Perceptron

The final network can be expressed as an equation of the parameters x1 through xn

$$w_1 x_1 + w_2 x_2 + \dots + w_n x_n = \text{threshold}$$

If learned, the network can be represented as a hyperplane



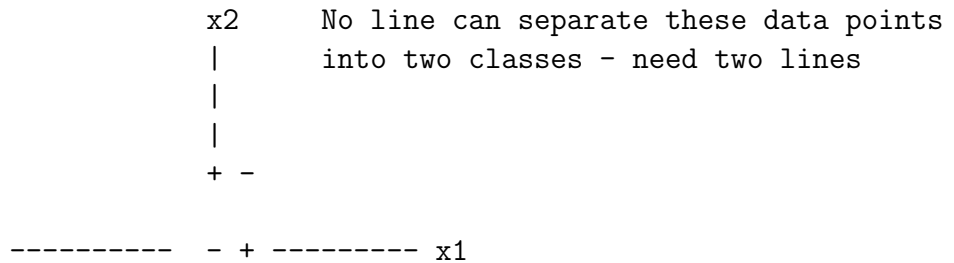
Examples

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:



|
|
|
|

How Can We Learn These Functions?

Add more layers with more neurons!

Features of perceptrons:

Only 1 neuron in output layer

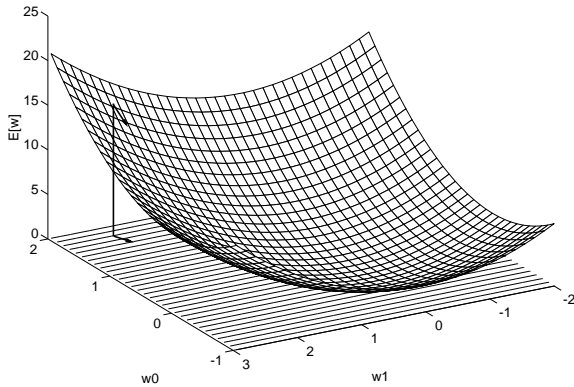
Inputs only 0 or 1

Transfer function compares weighted sum to threshold

No hidden units

Output is only 0 or 1

A Perceptron Performs Gradient Descent



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.

Delta Rule

Weight update for hidden-to-output links:

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

w_{ji} = weight of link from node j to node i

η = learning rate, eta

a_j = activation of node j

If hidden node, this is a_j . If input to input node, this is I_k .

Err_i = Error at this node (target output minus actual output, how much we need to change ALL the weights to node i)

$g'(in_i)$ = derivative of the transfer function g

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

Hidden-to-output Weights

$\Delta_i = Err_i * g'(in_i)$ or

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

t_i = true / target output for node i

o_i = actual / calculated output for node i (t - o is error for node i)

in_i = sum of inputs

g'_i = derivative of 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 contribution to output node.

First,

$$\Delta_j = g'(in_j) \sum_i w_{ji} \Delta_i$$

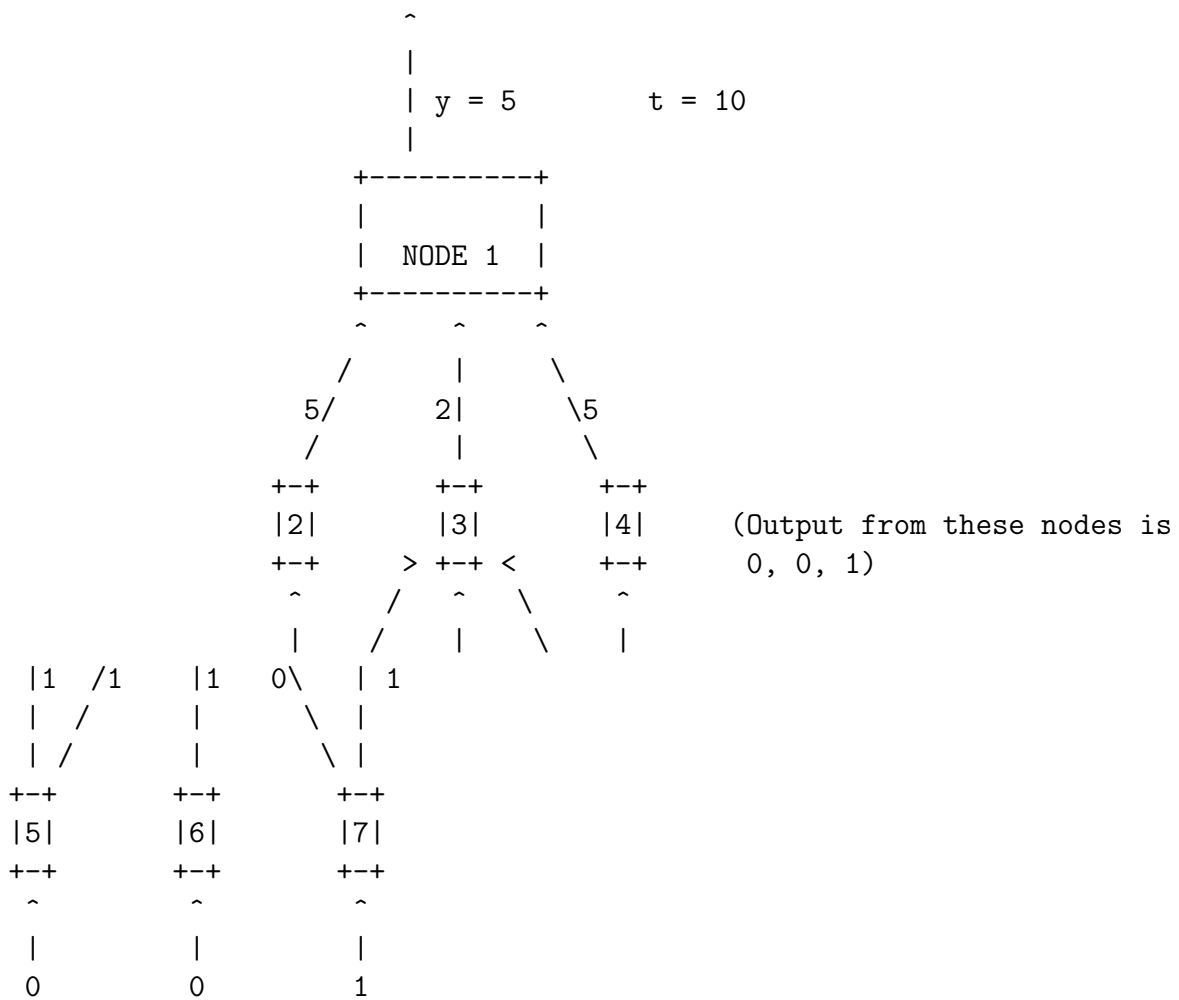
assigns a portion of the responsibility to node j.

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.

$$w_{kj} = w_{kj} + (\eta * I_k * \Delta_j), \text{ for each input node k to hidden node j.}$$

Example

If we use $g(x) = x$, $g'(x) = 1$, we let $n = 1$



$$\text{delta}_1 = (10 - 5) * 1 = 5$$

$$w_{21} = 5 + 1*0*5 = 5, \quad w_{31} = 2 + 1*0*5 = 2, \quad w_{41} = 5 + 1*1*5 = 5 + 5 = 10$$

$$\text{delta}_2 = 1 * w_{21} * \text{delta}_1 = 1 * 5 * 5 = 25$$

$$\text{delta}_3 = 1 * w_{31} * \text{delta}_1 = 1 * 2 * 5 = 10$$

$$\text{delta}_4 = 1 * w_{41} * \text{delta}_1 = 1 * 5 * 5 = 25$$

Therefore

$$w_{52} = 1 + 1*0*25 = 1, \quad w_{53} = 1 + 1*0*25 = 1$$

$$w_{63} = 1 + 1*0*10 = 1$$

$$w_{73} = 0 + 1*1*10 = 10, \quad w_{74} = 1 + 1*1*25 = 26$$

Now for same values from input nodes (0, 0, and 1), output will be $10*2 + 26*5 = 150$, which is a move in the right direction but too big a move.

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.

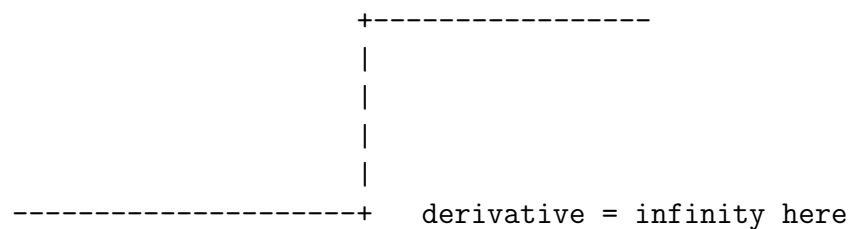
We want to move the weights in the direction of steepest descent in this space.

To do this, compute derivative of the error with respect to each weight in the equation. This results in the delta terms showed earlier.

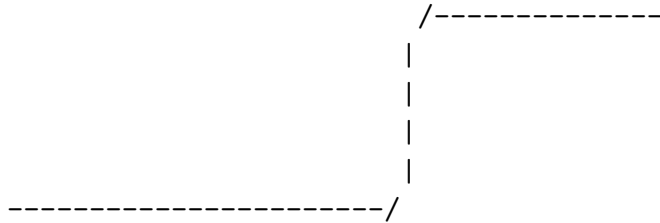
Note that the transfer function must be differentiable everywhere.

Step Function

Our perceptron transfer function will not work



Sigmoid Function



$$g = \frac{1}{1 + e^{-x}}, \quad \text{where } x \text{ is weighted sum}$$

NN application - NETtalk

Sejnowski and Rosenberg, 1985

Written English text to English speech

Based on DECtalk expert system

Look at 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

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.

$$y(t) = w_1 x_1(t) + w_2 x_2(t) + \dots + w_{n+1} x_1(t-1) + w_{n+1} x_2(t-1) + \dots$$

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

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/actuators
-

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 $U(i)$ on states
 - Agent learns action-value function
 - Can handle deterministic or probabilistic state transitions
-

Passive Learning in a Known Environment

Sample environment

Transitions between states are probabilistic, and are represented as a Markov Decision Process.

Markov Decision Processes

Assume

- finite set of states S
 - set of actions A
 - at each discrete time agent observes state $i \in S$ and chooses action $a \in A$
 - then receives immediate reward r_i
 - and state changes to j
 - Markov assumption: Resulting state j depending on M_{ij}^a and r_i
 - Reward and next state depend only on *current* state and action
 - M_{ij}^a is probability of reaching state j after executing action a in state i
 - M_{ij}^a can be estimated from observed state transition frequencies
-

Passive Learning Agent

- Learning utilities U of each state
 - Reward is accumulated over entire sequence of states
-

Two Methods For Updating Utility Values

- Naive
 - Least mean squares
 - $U[i] = \text{RunningAverage}(U[i], \text{CumReward}, N[i])$
 - $N[i]$ is number of times visit state i
 - Does not use transition probabilities constraints
 - Converges slowly
 - Adaptive dynamic programming (ADP)
 - $U(i) = R(i) + \sum_j M_{ij} * U(j)$
 - $R(i)$ = reward for being in state i
 - M_{ij} is probability of transition from state i to state j
 - Solve n equations in n unknowns, $n = |\text{states}|$
 - Is this practical?
-

Temporal Difference Learning

Instead of solving equations for all states, incrementally update visited states

TD reduces discrepancies between current and past states

If previous state has utility -100 and current state has utility +100, increase previous state utility to lessen discrepancy

- Temporal difference (TD)
 - When observed transition from state i to state j
 - $U(i) = U(i) + \alpha * (R(i) + U(j) - U(i))$
 - * α = learning rate
 - * $\alpha(N[i]) \sim 1/N[i]$
 - Slower convergence
-

Active Learning in Unknown Environment

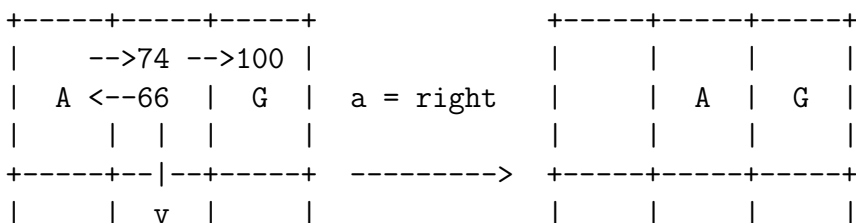
- Consider actions, their outcomes, and possible reward
 - Select action that maximizes expected reward
 - From utility theory, the expected utility of an action given evidence can be calculated as $EU(A|E) = \sum_i P(Result_i(A)|E, Do(A))U(Result_i(A))$
 - $U(i) = R(i) + \max_a \sum_j M_{ij}^a * U(j)$
-

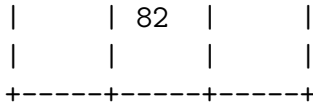
Learning an Action-Value Function

- Assigns value to action-state pairs, not just states
 - These values are called Q-values $Q(i, a)$
 - $Q(i, a)$ = value of doing action a in state i
 - Do not need transition model
 - Learned directly from explicit reward feedback
 - Provide condition action rules
 - Relation between utility values and Q values
 - $U(i) = \max_a Q(i, a)$
 - TD-based Q learning
 - When transition from state i to state j
 - $Q(i, a) = Q(i, a) + \alpha * (R(i) + \max_{a'} Q(j, a') - Q(i, a))$
-

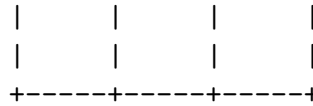
Example

Consider grid world where goal state (G) yields reward of 100 and other states yield reward of 0.
Let learning rate be 0.8.





Initial state s1

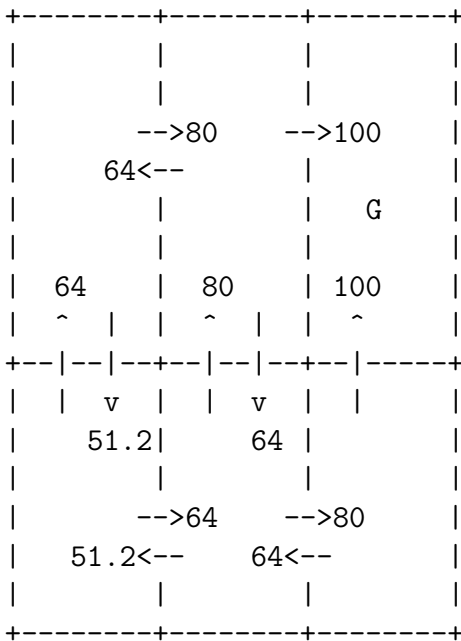


State s2

$$\begin{aligned}
 Q(s1, \text{right}) &= 74 + 0.8(0 + \max\{Q(s2, \text{left}), Q(s2, \text{right}), Q(s2, \text{down})\}) - 74 \\
 &= 74 + 0.8(0 + \max\{66, 82, 100\} - 74) = 74 + 0.8(26) = 94.8
 \end{aligned}$$

Example

Learned $Q(s,a)$ values



Initial state s1

State s2

Exploration and Exploitation

- Actions have two purposes
 - Gaining rewards
 - Gaining information leading (possibly) to better rewards
- If always use best approximate Q value, may miss better parts of space

- Space may be dynamic
 - Solution: Select actions probabilistically to balance exploration and exploitation
 - Select action a that maximizes $f(Q(j, a'), N[j, a'])$ [Russell and Norvig]
 - $P(a_i|s) = \frac{k^{Q(s, a_i)}}{\sum_j k^{Q(s, a_j)}}$ [Mitchell]
-

Examples

TD-Gammon [Tesauro, 1995]

- Learn to play Backgammon
 - Immediate reward
 - +100 if win
 - -100 if lose
 - 0 for all other states
 - Trained by playing 1.5 million games against itself
 - Now approximately equal to best human player
-

Subtleties and Ongoing Research

- Scalability
- Generalize utilities from visited states to other states (inductive learning)
- Handle case where state only partially observable
- Design optimal exploration strategies
- Extend to continuous actions, states