

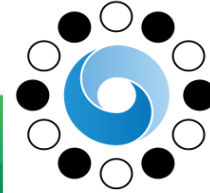
Artificial Intelligence (CS F407)



Machine Learning: Introduction

Chittaranjan Hota
Professor, Computer Science & Information Systems Department
BITS Pilani Hyderabad Campus, Hyderabad
hota@hyderabad.bits-pilani.ac.in

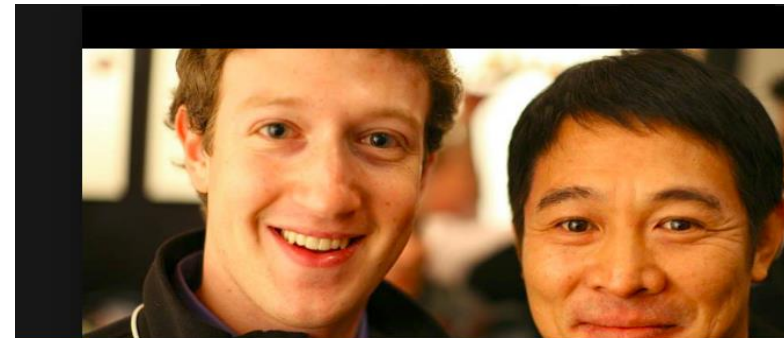
Some tasks...



AlphaGo

NETFLIX

Content Promotion, Content Price Modeling, Adaptive Streaming QoE...



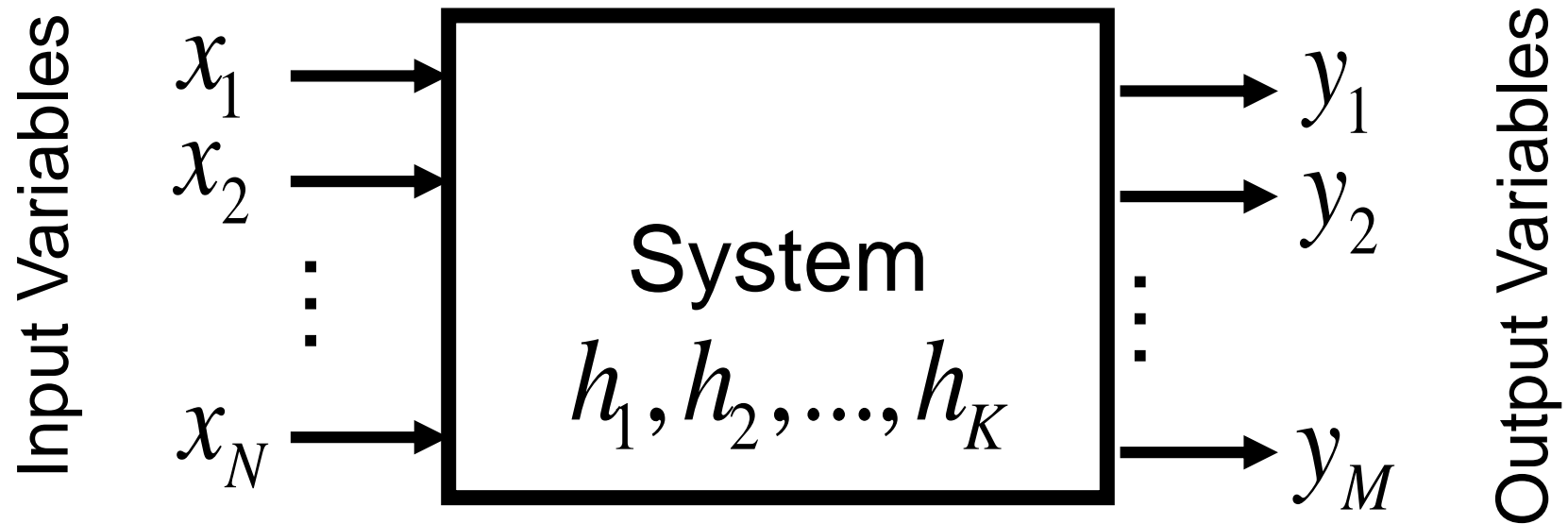
Inspector Console Debugger Style Editor Performance Memory Ne
Search HTML
<div id="fbPhotoSnowliftTagBoxes" class="fbPhotosPhotoTagBoxes tagContainer"></div>
<div id="fbPhotoSnowliftTagApproval" class="fbPhotoTagApproval hidden_elem"></div>
<div id="fbPhotoSnowliftComputerVisionInfo" class="_5bai"></div>
<div class="2-sx" style="width: 660px; height: 440px;">

‘cocktail party problem’

What is Learning?

- “Learning denotes changes in a system that enables a system to do the same task more efficiently the next time.” – Herbert Simon
(Writing your AI Midsem and then Compre...)
- Learning is the process of acquiring new or modifying existing knowledge, behaviors, skills, values, or preferences- Wiki
- Humans learn from experiences....

What is Machine Learning?



Hidden Variables:



Defining the Learning Task

Improve on task, T , with respect to performance metric, P , based on experience, E (Tom Mitchell, 1997).

T : Playing chess

P : Percentage of games won against an arbitrary opponent

E : Playing practice games against whom?

T : Recognizing hand-written words

P : Percentage of words correctly classified

E : Database of human-labeled images of handwritten words

T : Categorize email messages as spam or legitimate.

P : Percentage of email messages correctly classified.

E : Database of emails, some with human-given labels

Why Machine Learning is important?

- **Machine learning** is programming computers to optimize a performance criterion using example **data**.
- There is no need to “learn” to **calculate payroll**
- **Learning is used when:**
 - Human expertise does not exist
 - **x**: Bond graph for a new molecule, $f(\mathbf{x})$: Predicted binding strength to AIDS protease molecule.
 - Navigating on Mars
 - Humans are unable to explain their expertise
 - **x**: Bitmap picture of hand-written character, $f(\mathbf{x})$: Ascii code of the character
 - Speech recognition

Continued...

- Solution changes in time
 - \mathbf{x} : Description of stock prices and trades for last 20 days, $f(\mathbf{x})$: Recommended stock transactions
 - Routing on a computer network
- Solution needs to be adapted to particular cases
 - \mathbf{x} : Incoming email message, $f(\mathbf{x})$: Importance score for presenting to user (or deleting without presenting)
 - User biometrics
- Relationship and correlations can be hidden within large amounts of data
- New knowledge is being constantly discovered by humans

Applications of Machine Learning

- natural language processing
- search engines
- medical diagnosis
- detecting credit card fraud
- stock market analysis
- classifying DNA sequences
- speech and handwriting recog.
- Spam filtering
- Financial Investments (Real estate or Postal deposits)
- game playing
- adaptive websites
- robot locomotion
- structural health monitoring
- sentiment analysis



Apache Singa



Amazon ML



(Frameworks/Tools)

Few more examples and algos...



Rs. 5 lakhs



? lakhs

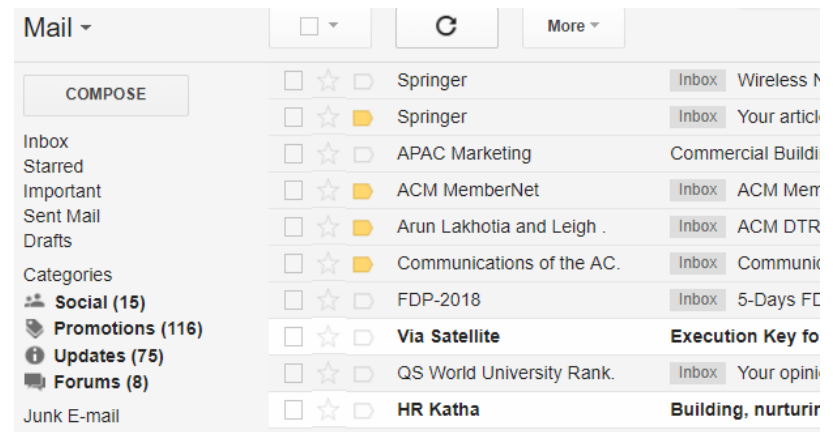
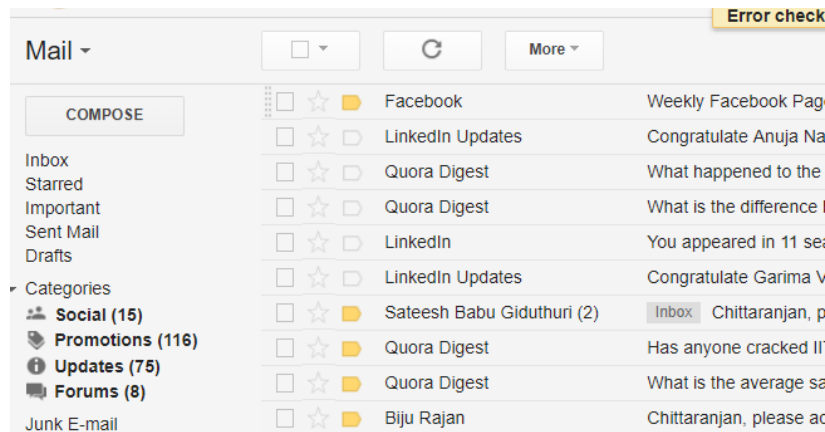


Rs. 80 lakhs

Example of Linear Regression...

Gradient descent

Few more examples and algos...



I'm getting a spam warning message

Gmail automatically identifies spam and suspicious emails and marks those emails as spam. When you open your Spam label, you'll see any emails that were marked as spam by you or Gmail. Each email will include a label at the top that explains why it was sent to Spam.

Spoofed email addresses	Identifying spam mails	▼
Phishing scams		▼

Boss
Cheap
Bulk
Dollar
Missing title etc...
Naïve Bayes

Few more examples and algos...

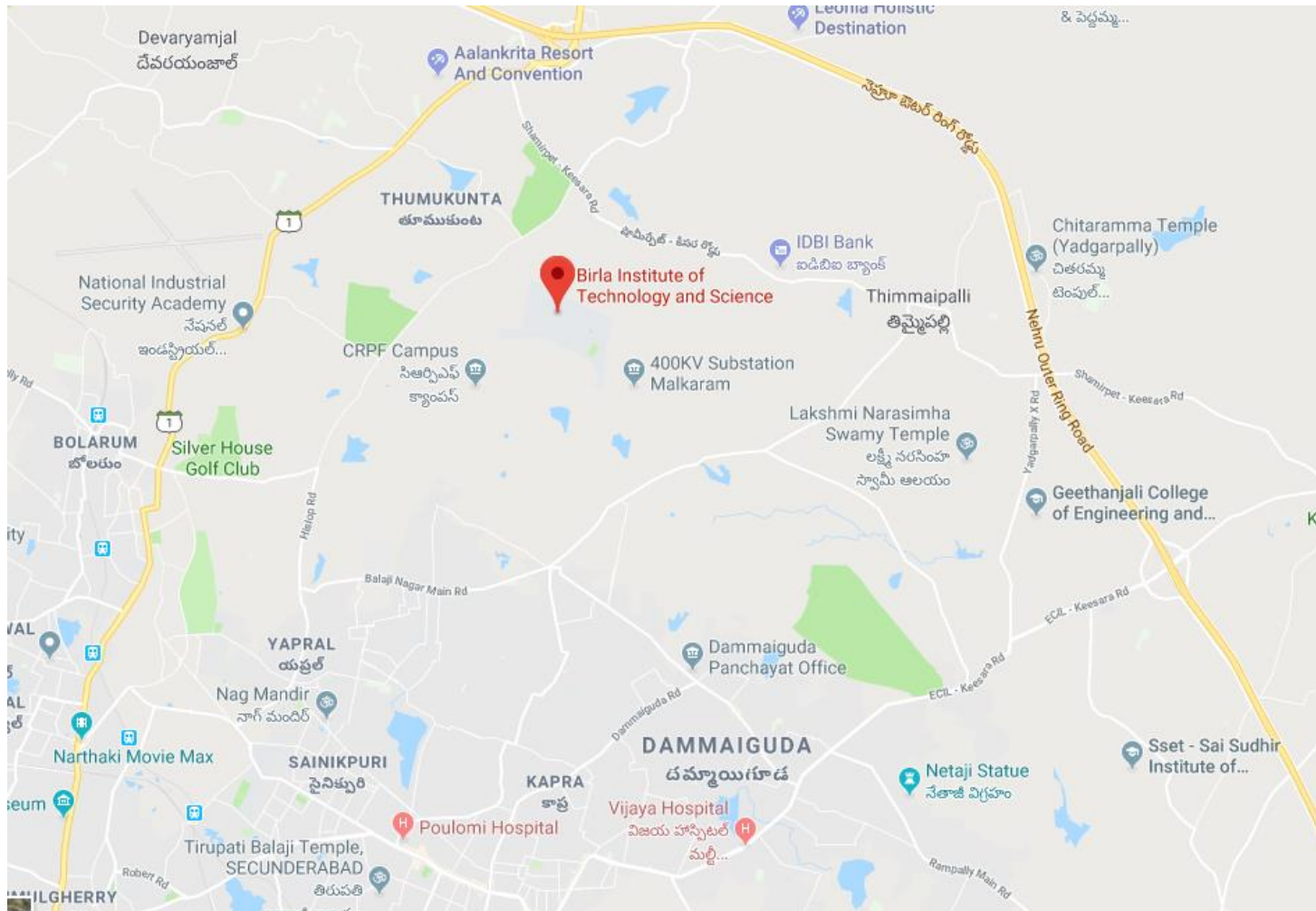


Recommending Apps to different users ...

Gender	Age	App
F	17	Instagram
F	28	WhatsApp
M	35	FB messenger
F	44	WhatsApp
M	13	Instagram
M	16	Instagram

Decision Tree.

Few more examples and algos...



Newspaper distributor.

K-means clustering

University admissions



Welcome to BITS, Pilani's Admissions website!

Birla Institute of Technology & Science (BITS), Pilani a leading Institute of Higher Ed under section 3 of the UGC act offering degree programmes in Engineering, & Management and Humanities. This web site describes the admission modalities for the offered at Pilani campus, Goa Campus and Hyderabad Campus of BITS, Pilani.

For knowing details about admissions to BITS, Pilani-Dubai campus **Click here**. For kn Work Integrated Learning Programmes of BITS, Pilani **Click here**.

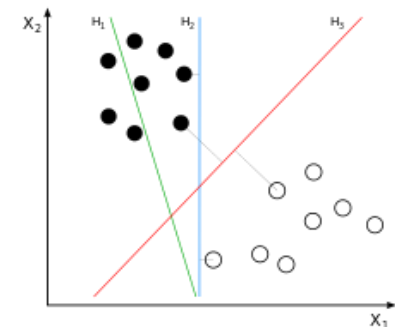
1. BITSAT-2018 online exam announcement.

BITSAT-2018 Slot booking is now enabled. Registered candidates may reser First Come First Serve basis. [CLICK HERE](#) to book your test date and test time

2. **New** BITSAT-2018 Slot booking is extended till 10th Apr

3. **New** Editing of applications under International Ac extended till 11th April 2018. Final list of Inte Programmes is also announced. **[CLICK HERE](#) for details.**

Logistic regression, SVM, ANNs etc ...



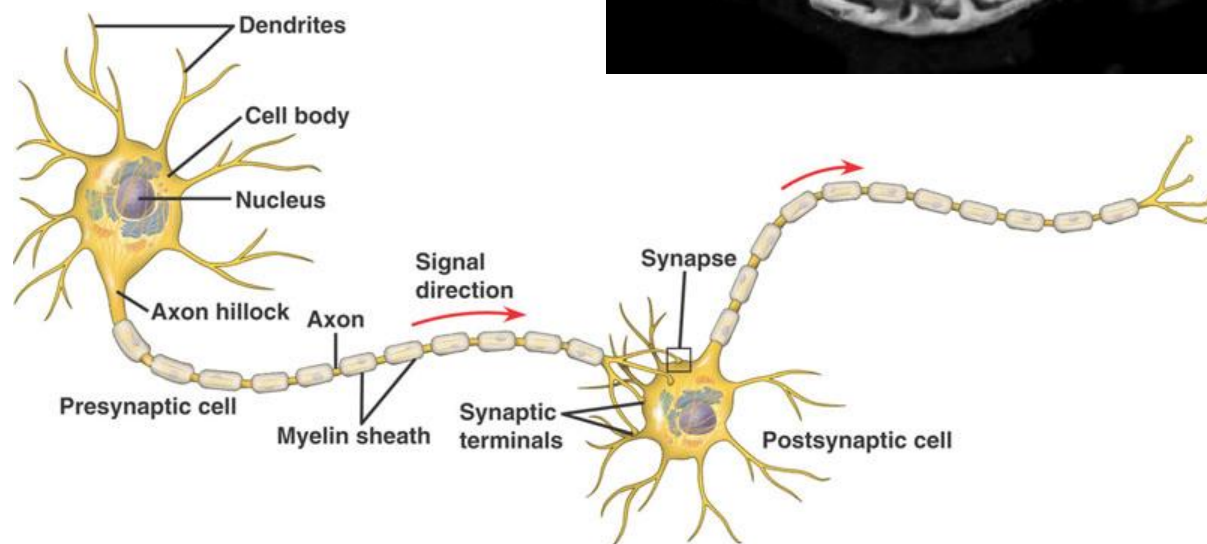
Issues in Machine Learning

- What algorithms are available for learning a concept? How well do they perform?
- How much training data is sufficient to learn a concept with high confidence?
- Are some training examples more useful than others?
- What are the best tasks for a system to learn?
- What is the best way for a system to represent it's knowledge?

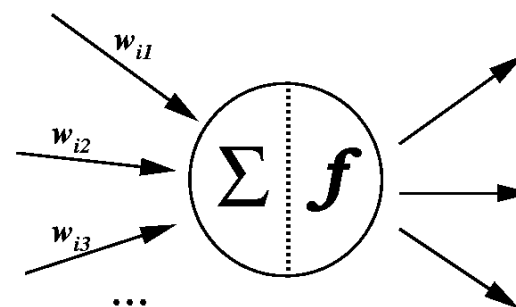
ANNs: Nerve cell or **neuron**



Nervous System	Computational Abstraction
Neuron	Node
Dendrites	Input link and propagation
Cell Body	Combination function, threshold, activation function
Axon	Output link
Synaptic strength	Connection strength/weight



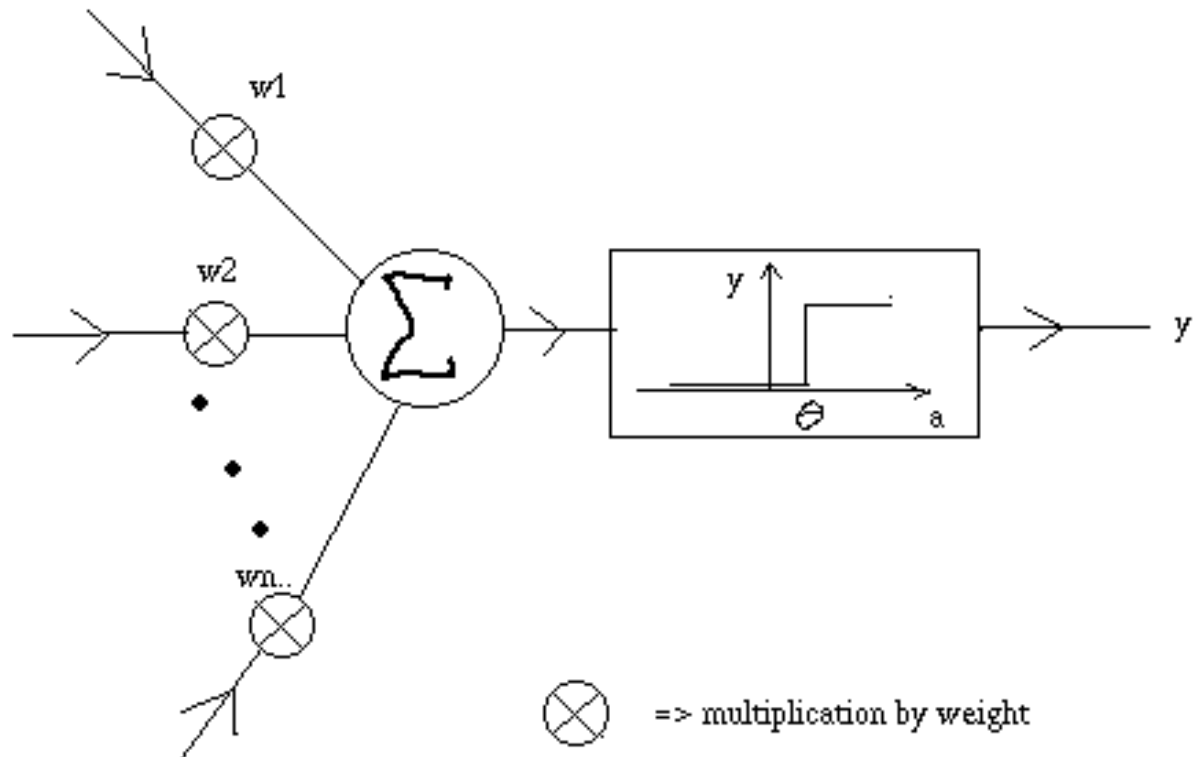
- A healthy human brain has around 100 billion neurons
- A neuron may connect to as many as 100,000 other neurons



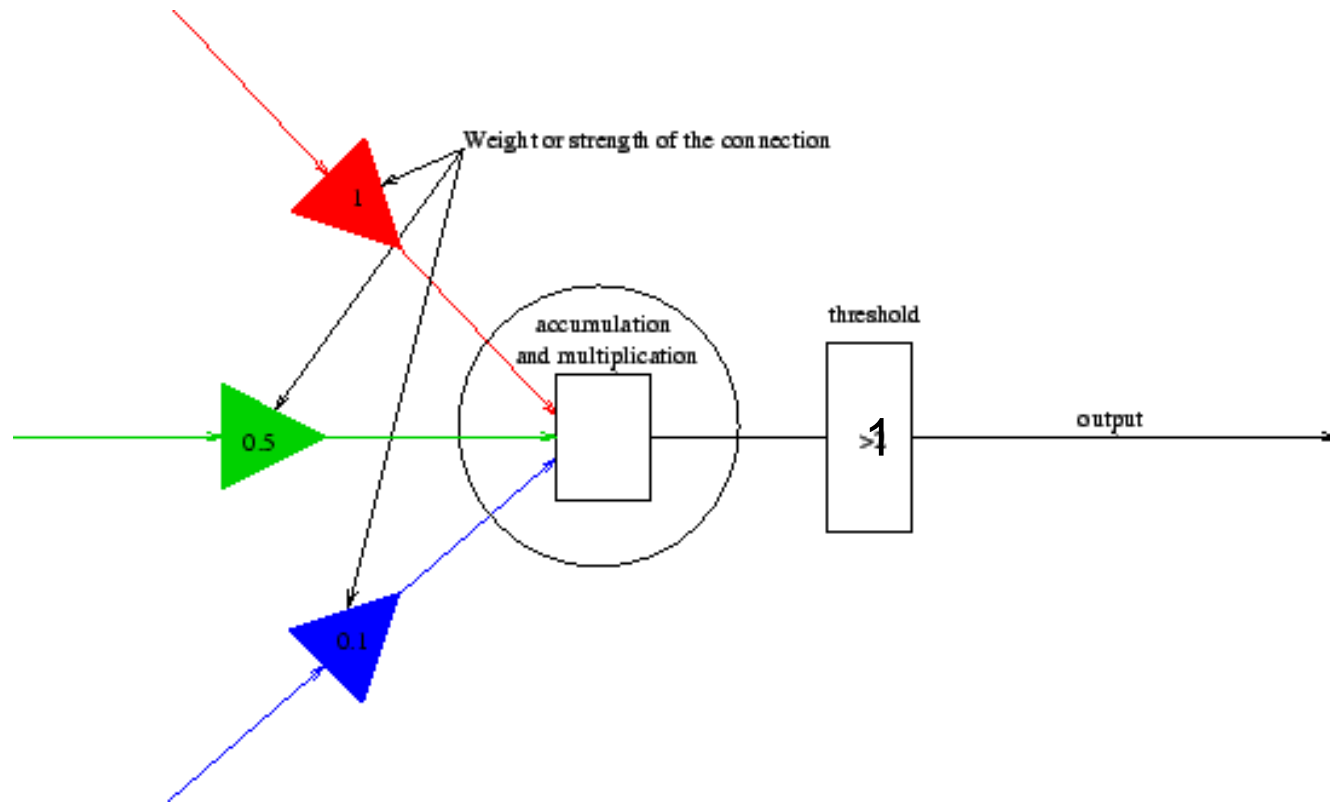
$$y_i = f(\text{net}_i)$$

Local,
parallel
computat
ion

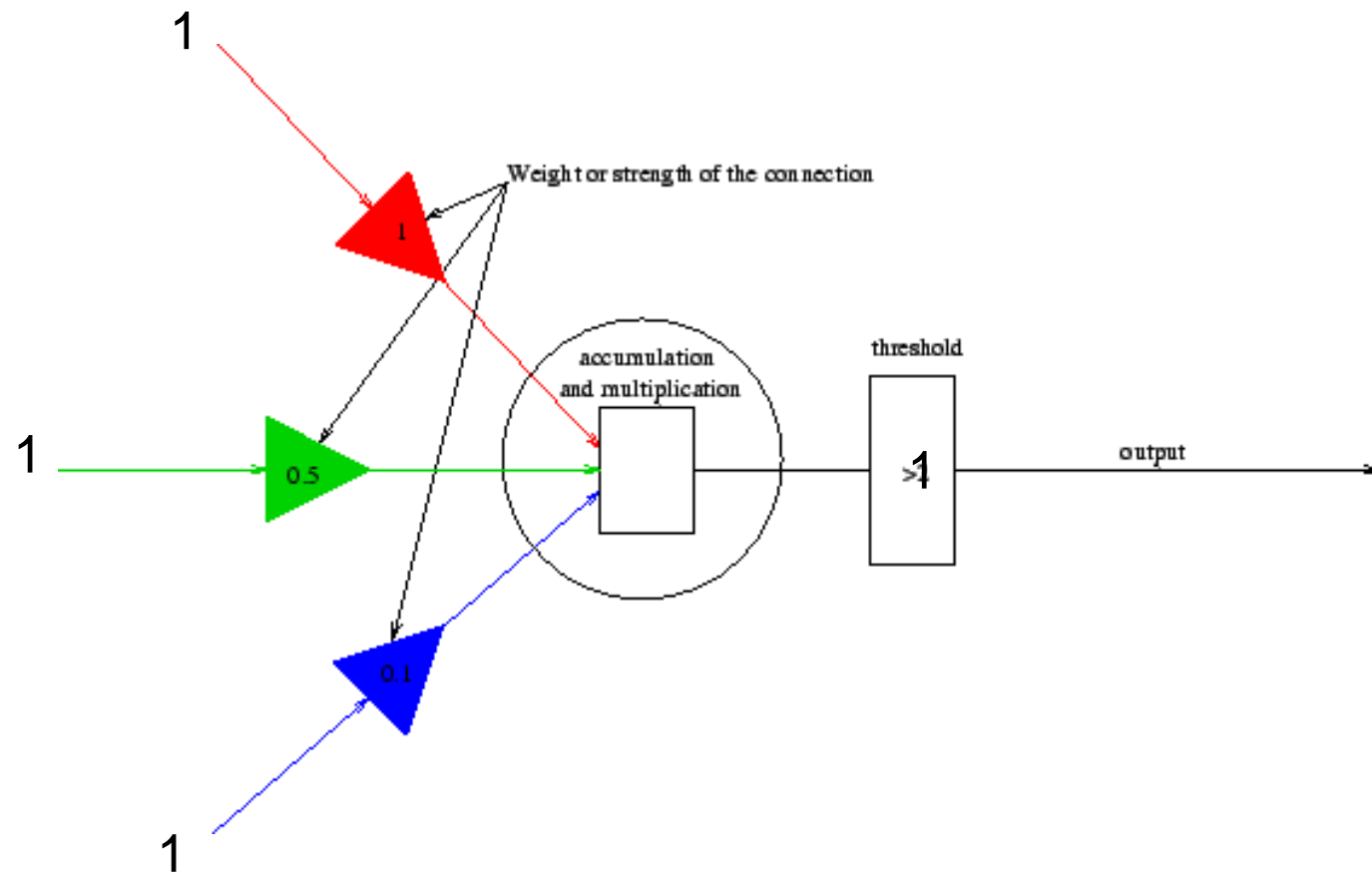
Simple Threshold Linear Unit



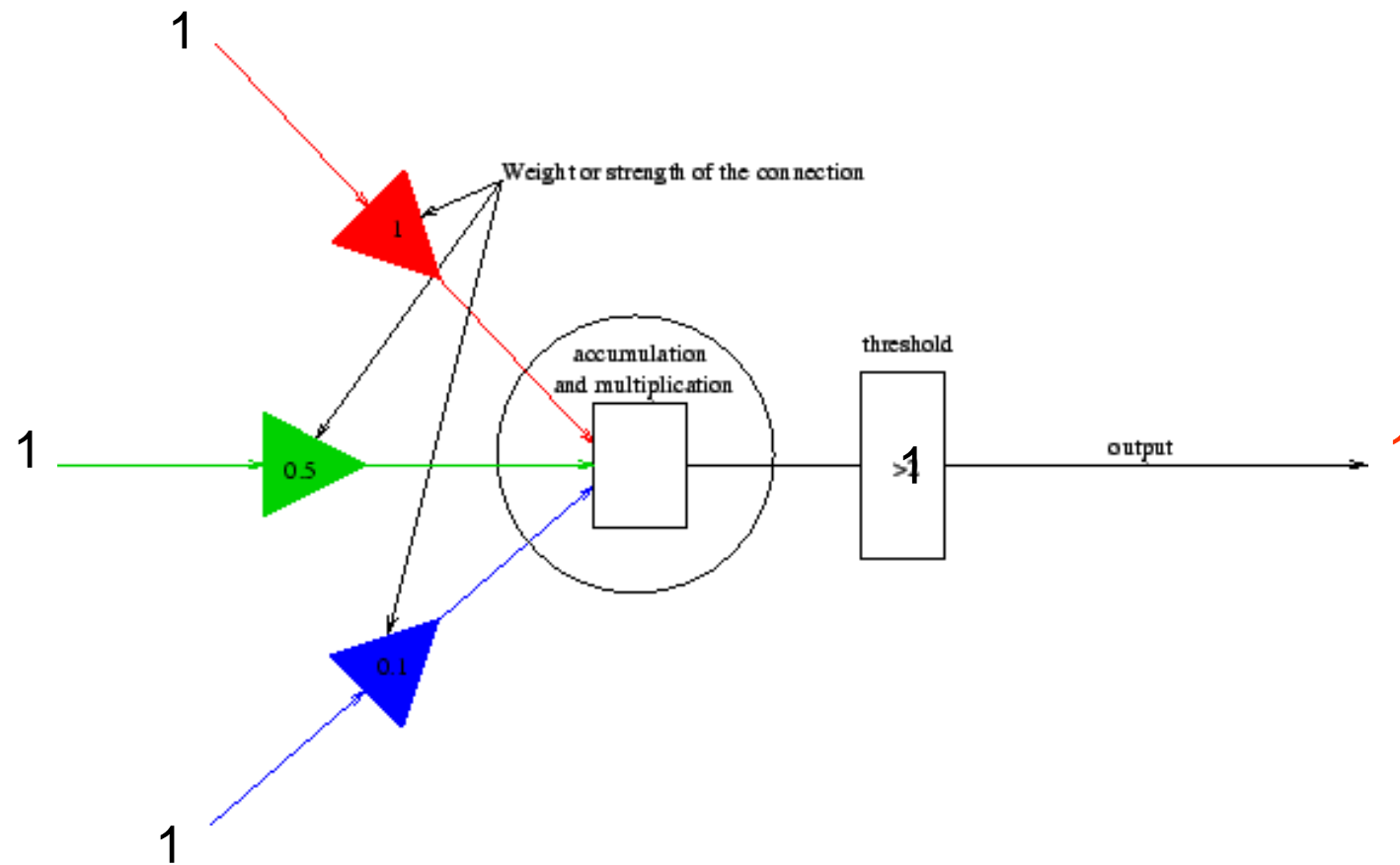
Simple Neuron Model



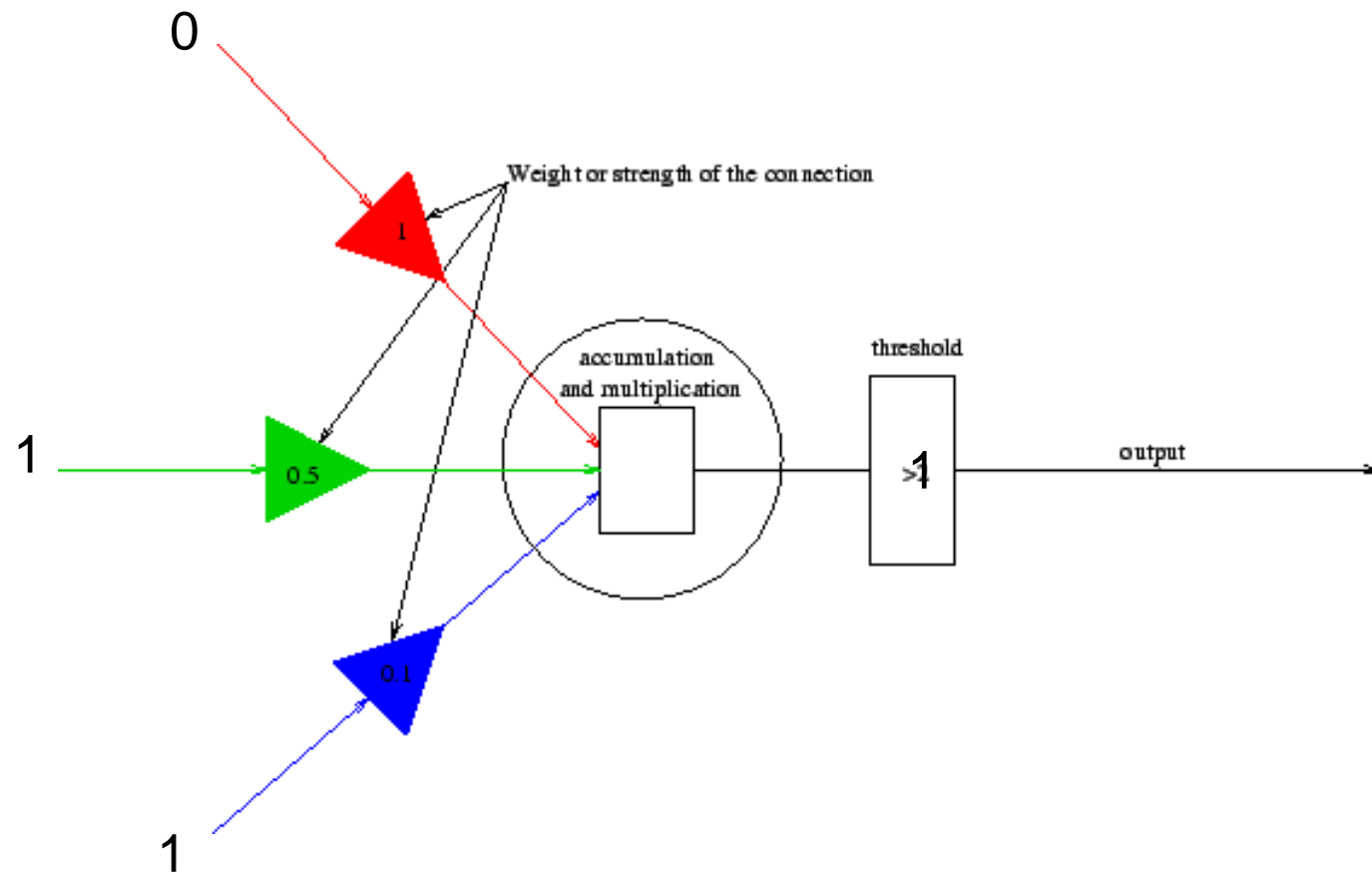
Simple Neuron Model



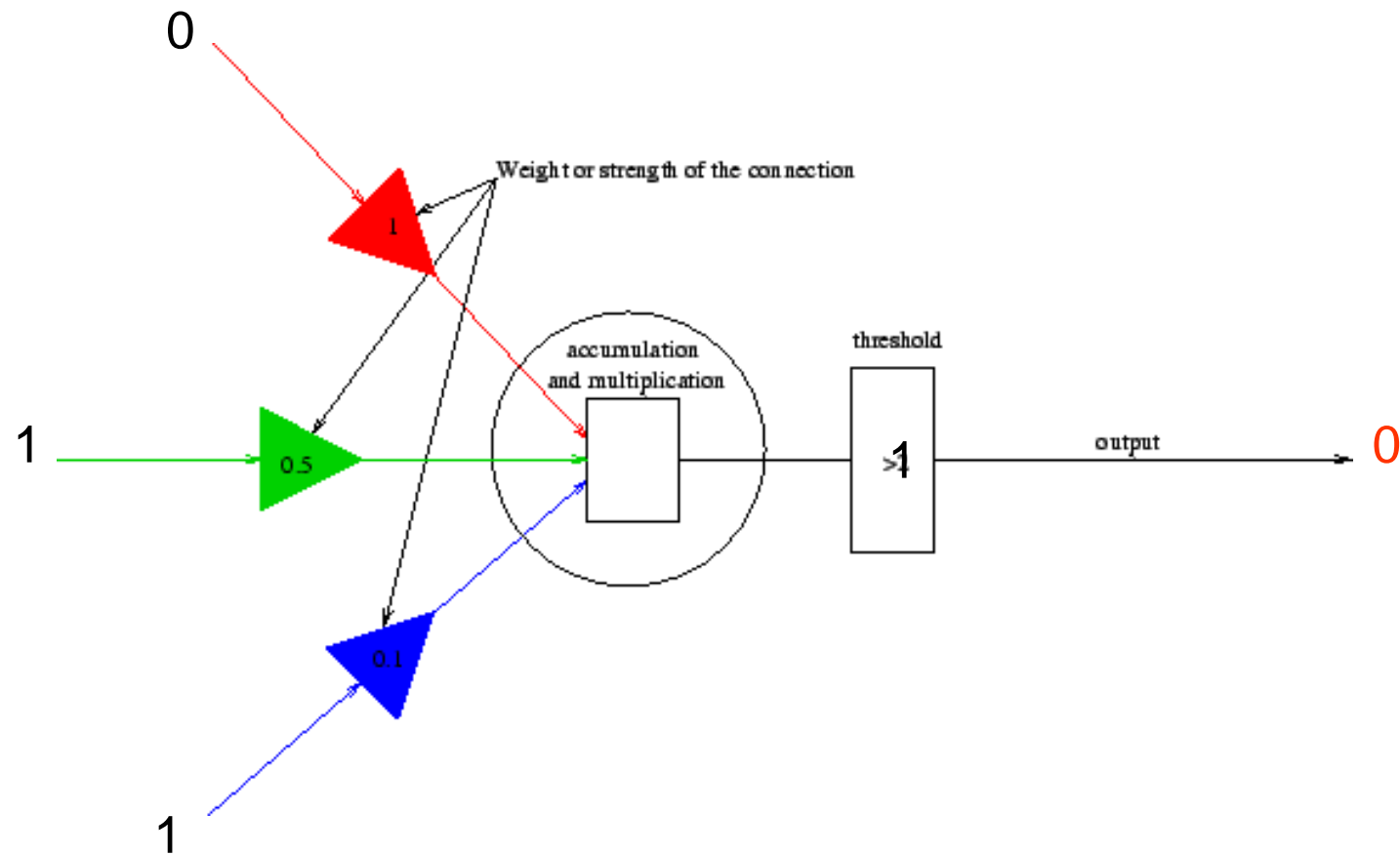
Simple Neuron Model



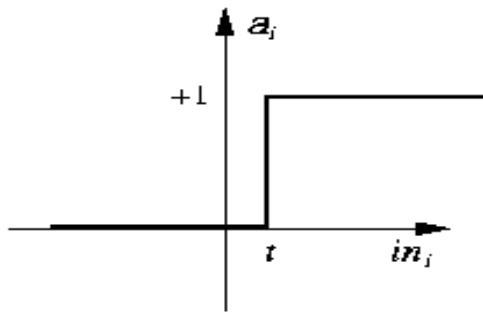
Simple Neuron Model



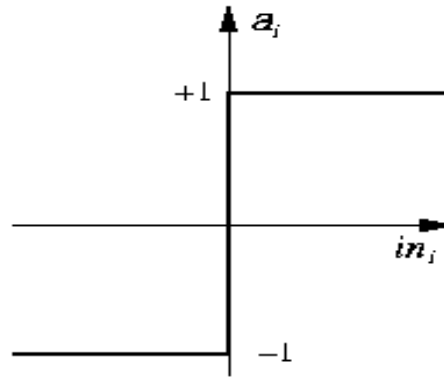
Simple Neuron Model



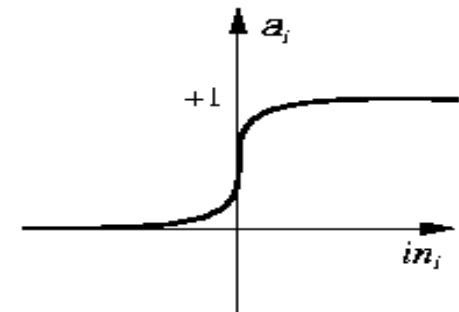
Activation Functions



(a) Step function

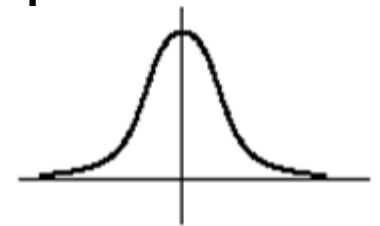


(b) Sign function



(c) Sigmoid function

- $\text{Step}_t(x) = 1$ if $x \geq t$, else 0 threshold= t
- $\text{Sign}(x) = +1$ if $x \geq 0$, else -1
- $\text{Sigmoid}(x) = 1 / (1 + e^{-x})$

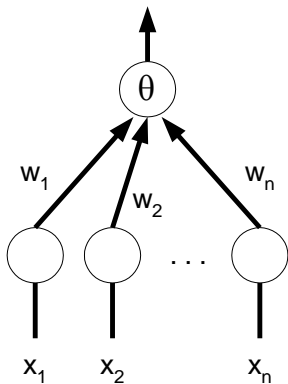


Gaussian

$$f(x) = e^{-ax^2}$$

Artificial neurons

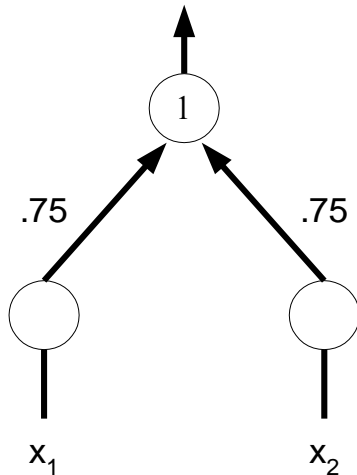
- McCulloch & Pitts (1943) described an artificial neuron
 - ▣ inputs are either electrical impulse (1) or not (0)
 - (note: original version used +1 for excitatory and -1 for inhibitory signals)
 - ▣ each input has a weight associated with it
 - ▣ the activation function multiplies each input value by its weight
 - ▣ if the sum of the weighted inputs $\geq \theta$,
 - then the neuron fires (returns 1), else doesn't fire (returns 0)



if $\sum w_i x_i \geq \theta$, output = 1

if $\sum w_i x_i < \theta$, output = 0

Computation via activation function



INPUT: $x_1 = 1, x_2 = 1$

$$.75*1 + .75*1 = 1.5 \geq 1 \quad \rightarrow \text{OUTPUT: } 1$$

INPUT: $x_1 = 1, x_2 = 0$

$$.75*1 + .75*0 = .75 < 1 \quad \rightarrow \text{OUTPUT: } 0$$

INPUT: $x_1 = 0, x_2 = 1$

$$.75*0 + .75*1 = .75 < 1 \quad \rightarrow \text{OUTPUT: } 0$$

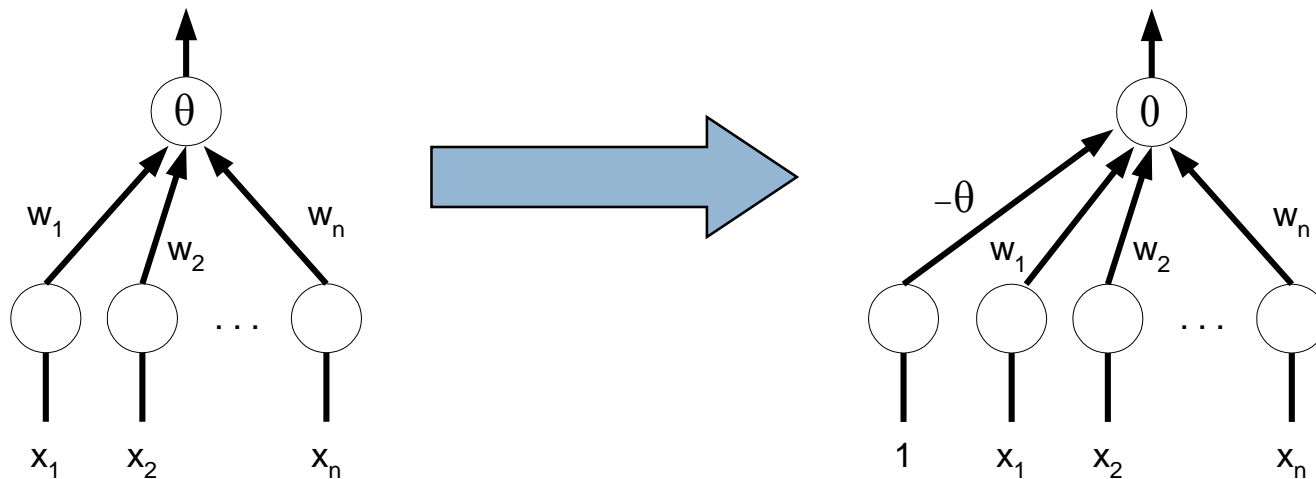
INPUT: $x_1 = 0, x_2 = 0$

$$.75*0 + .75*0 = 0 < 1 \quad \rightarrow \text{OUTPUT: } 0$$

this neuron computes the AND function

Normalizing thresholds

- to make life more uniform, can normalize the threshold to 0
 - ▣ simply add an additional input $x_0 = 1$, $w_0 = -\theta$



advantage: threshold = 0 for all neurons

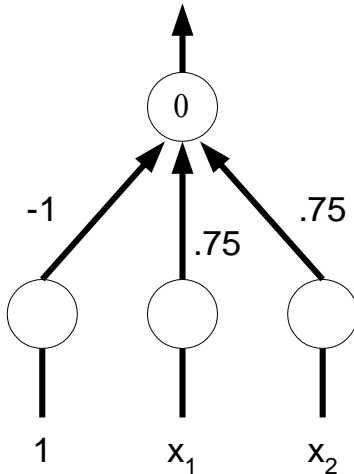
$$\sum w_i x_i \geq \theta$$

\equiv

$$-\theta * 1 + \sum w_i x_i \geq 0$$

Normalized examples

AND



INPUT: $x_1 = 1, x_2 = 1$

$$1 \cdot -1 + .75 \cdot 1 + .75 \cdot 1 = .5 \geq 0$$

→ OUTPUT: 1

INPUT: $x_1 = 1, x_2 = 0$

$$1 \cdot -1 + .75 \cdot 1 + .75 \cdot 0 = -.25 < 1$$

→ OUTPUT: 0

INPUT: $x_1 = 0, x_2 = 1$

$$1 \cdot -1 + .75 \cdot 0 + .75 \cdot 1 = -.25 < 1$$

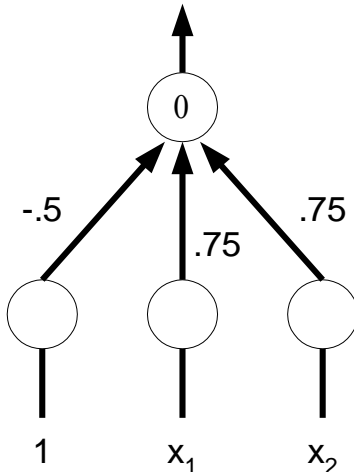
→ OUTPUT: 0

INPUT: $x_1 = 0, x_2 = 0$

$$1 \cdot -1 + .75 \cdot 0 + .75 \cdot 0 = -1 < 1$$

→ OUTPUT: 0

OR



INPUT: $x_1 = 1, x_2 = 1$

$$1 \cdot -.5 + .75 \cdot 1 + .75 \cdot 1 = 1 \geq 0$$

→ OUTPUT: 1

INPUT: $x_1 = 1, x_2 = 0$

$$1 \cdot -.5 + .75 \cdot 1 + .75 \cdot 0 = .25 > 1$$

→ OUTPUT: 1

INPUT: $x_1 = 0, x_2 = 1$

$$1 \cdot -.5 + .75 \cdot 0 + .75 \cdot 1 = .25 < 1$$

→ OUTPUT: 1

INPUT: $x_1 = 0, x_2 = 0$

$$1 \cdot -.5 + .75 \cdot 0 + .75 \cdot 0 = -.5 < 1$$

→ OUTPUT: 0

Perceptron

- Rosenblatt, 1962
- 2-Layer network.
- Threshold activation function at output
 - +1 if weighted input is above threshold.
 - -1 if below threshold.

Perceptron learning algorithm:

1. Set the weights on the connections with random values.
2. Iterate through the training set, comparing the output of the network with the desired output for each example.
3. If all the examples were handled correctly, then DONE.
4. Otherwise, update the weights for each incorrect example:
 - if should have fired on x_1, \dots, x_n but didn't, $w_i += x_i$ ($0 \leq i \leq n$)
 - if shouldn't have fired on x_1, \dots, x_n but did, $w_i -= x_i$ ($0 \leq i \leq n$)
5. GO TO 2

Learning Process for Perceptron

- Initially assign random weights to inputs between -0.5 and +0.5
- Training data is presented to perceptron and its output is observed.
- If output is incorrect, the weights are adjusted accordingly using following formula.

$$w_i \leftarrow w_i + (a \cdot x_i \cdot e), \text{ where 'e' is error produced and 'a' } (-1 < a < 1) \text{ is learning rate}$$

- 'a' is defined as 0 if output is correct, it is +ve, if output is too low and –ve, if output is too high.

Example: Perceptron to learn OR function

- Initially consider $w_1 = -0.2$ and $w_2 = 0.4$
- Training data say, $x_1 = 0$ and $x_2 = 0$, output is 0.
- Compute $y = \text{Step}(w_1 * x_1 + w_2 * x_2) = 0$. Output is correct so weights are not changed.
- For training data $x_1 = 0$ and $x_2 = 1$, output is 1
- Compute $y = \text{Step}(w_1 * x_1 + w_2 * x_2) = 0.4 = 1$. Output is correct so weights are not changed.
- Next training data $x_1 = 1$ and $x_2 = 0$ and output is 1
- Compute $y = \text{Step}(w_1 * x_1 + w_2 * x_2) = -0.2 = 0$. Output is **incorrect**, hence weights **are to be changed**.
- Assume $\alpha = 0.2$ and error $e = 1$
 $w_i = w_i + (\alpha * x_i * e)$ gives $w_1 = 0$ and $w_2 = 0.4$
- With these weights, test the remaining test data.
- Repeat the process till we get stable result.

Example: perceptron learning

- Suppose we want to train a perceptron to compute AND

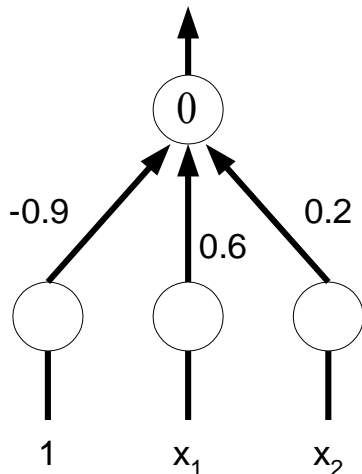
training set: $x_1 = 1, x_2 = 1 \rightarrow 1$

$x_1 = 1, x_2 = 0 \rightarrow 0$

$x_1 = 0, x_2 = 1 \rightarrow 0$

$x_1 = 0, x_2 = 0 \rightarrow 0$

randomly, let: $w_0 = -0.9, w_1 = 0.6, w_2 = 0.2$



using these weights:

$$x_1 = 1, x_2 = 1: -0.9*1 + 0.6*1 + 0.2*1 = -0.1 \rightarrow 0$$

WRONG

$$x_1 = 1, x_2 = 0: -0.9*1 + 0.6*1 + 0.2*0 = -0.3 \rightarrow 0$$

OK

$$x_1 = 0, x_2 = 1: -0.9*1 + 0.6*0 + 0.2*1 = -0.7 \rightarrow 0$$

OK

$$x_1 = 0, x_2 = 0: -0.9*1 + 0.6*0 + 0.2*0 = -0.9 \rightarrow 0$$

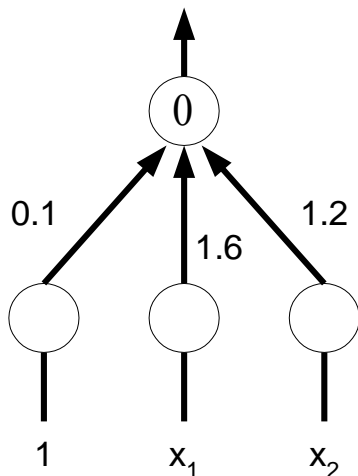
OK

new weights: $w_0 = -0.9 + 1 = 0.1$

$$w_1 = 0.6 + 1 = 1.6$$

$$w_2 = 0.2 + 1 = 1.2$$

Example: perceptron learning (cont.)



using these updated weights:

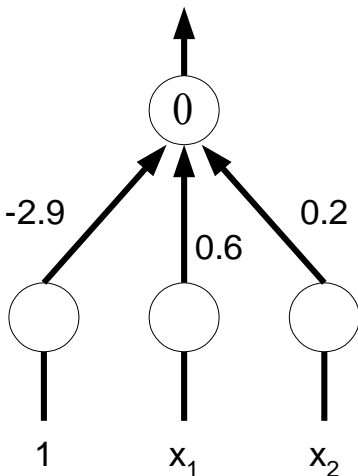
$x_1 = 1, x_2 = 1:$	$0.1*1 + 1.6*1 + 1.2*1$	$= 2.9 \rightarrow 1$	OK
$x_1 = 1, x_2 = 0:$	$0.1*1 + 1.6*1 + 1.2*0$	$= 1.7 \rightarrow 1$	WRONG
$x_1 = 0, x_2 = 1:$	$0.1*1 + 1.6*0 + 1.2*1$	$= 1.3 \rightarrow 1$	WRONG
$x_1 = 0, x_2 = 0:$	$0.1*1 + 1.6*0 + 1.2*0$	$= 0.1 \rightarrow 1$	WRONG

new weights:

$$w_0 = 0.1 - 1 - 1 - 1 = -2.9$$

$$w_1 = 1.6 - 1 - 0 - 0 = 0.6$$

$$w_2 = 1.2 - 0 - 1 - 0 = 0.2$$



using these updated weights:

$x_1 = 1, x_2 = 1:$	$-2.9*1 + 0.6*1 + 0.2*1$	$= -2.1 \rightarrow 0$	WRONG
$x_1 = 1, x_2 = 0:$	$-2.9*1 + 0.6*1 + 0.2*0$	$= -2.3 \rightarrow 0$	OK
$x_1 = 0, x_2 = 1:$	$-2.9*1 + 0.6*0 + 0.2*1$	$= -2.7 \rightarrow 0$	OK
$x_1 = 0, x_2 = 0:$	$-2.9*1 + 0.6*0 + 0.2*0$	$= -2.9 \rightarrow 0$	OK

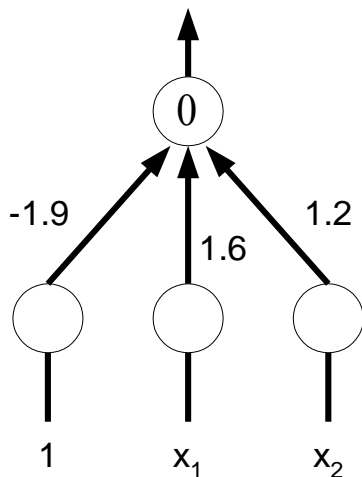
new weights:

$$w_0 = -2.9 + 1 = -1.9$$

$$w_1 = 0.6 + 1 = 1.6$$

$$w_2 = 0.2 + 1 = 1.2$$

Example: perceptron learning (cont.)



using these updated weights:

$x_1 = 1, x_2 = 1:$	$-1.9*1 + 1.6*1 + 1.2*1$	$= 0.9 \rightarrow 1$	OK
$x_1 = 1, x_2 = 0:$	$-1.9*1 + 1.6*1 + 1.2*0$	$= -0.3 \rightarrow 0$	OK
$x_1 = 0, x_2 = 1:$	$-1.9*0 + 1.6*0 + 1.2*1$	$= -0.7 \rightarrow 0$	OK
$x_1 = 0, x_2 = 0:$	$-1.9*0 + 1.6*0 + 1.2*0$	$= -1.9 \rightarrow 0$	OK

DONE!

Convergence in Perceptrons

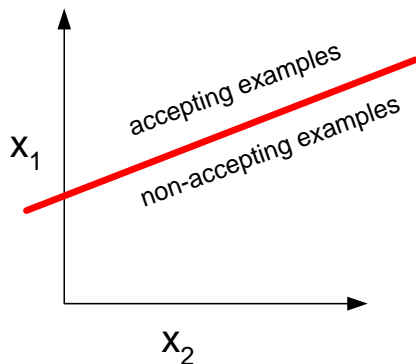
- key reason for interest in perceptrons:

Perceptron Convergence Theorem

- The perceptron learning algorithm will always find weights to classify the inputs *if such a set of weights exist*.

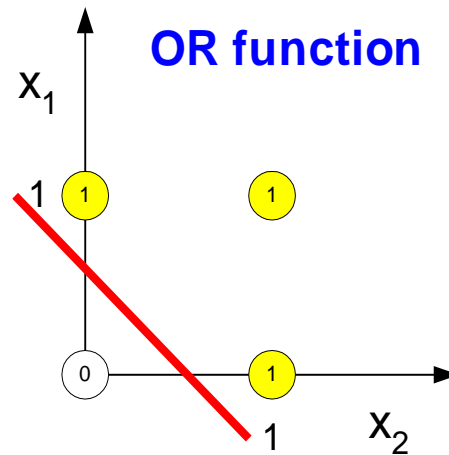
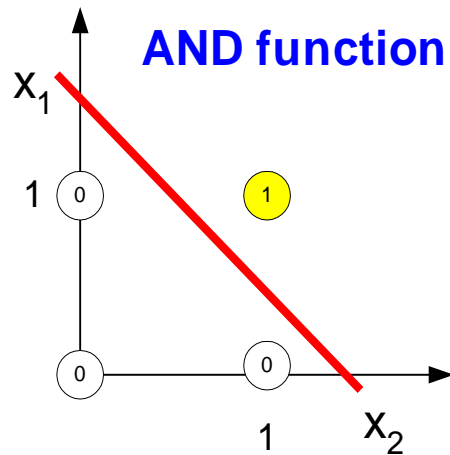
Minsky & Papert showed weights exist if and only if the problem is *linearly separable*

intuition: consider the case with 2 inputs, x_1 and x_2



if you can draw a line and separate the accepting & non-accepting examples, then *linearly separable*

Linearly separable



why does this make sense?

firing depends on

$$w_0 + w_1x_1 + w_2x_2 \geq 0$$

border case is when

$$w_0 + w_1x_1 + w_2x_2 = 0$$

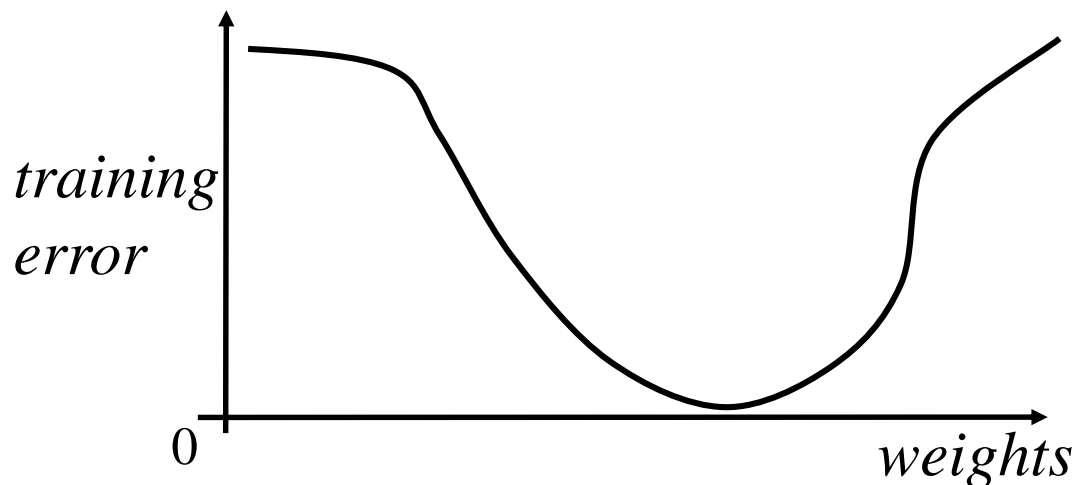
i.e.,

$$x_2 = (-w_1/w_2) x_1 + (-w_0/w_2) \quad \text{the equation of a line}$$

the training algorithm simply shifts the line around (by changing the weight) until the classes are separated

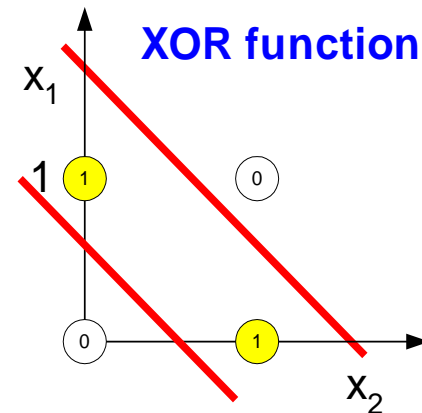
Perceptron as Hill Climbing

- The hypothesis space being searched is a set of **weights** and a **threshold**.
- Objective is to minimize classification error on the training set.
- Perceptron effectively does hill-climbing (gradient descent) in this space, changing the weights a small amount at each point to decrease training set error.

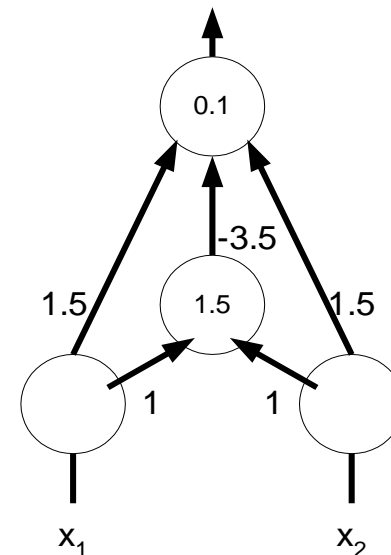


Inadequacy of perceptron

- inadequacy of perceptrons is due to the fact that many simple problems are not linearly separable



however, can compute XOR
by introducing a new, hidden
unit



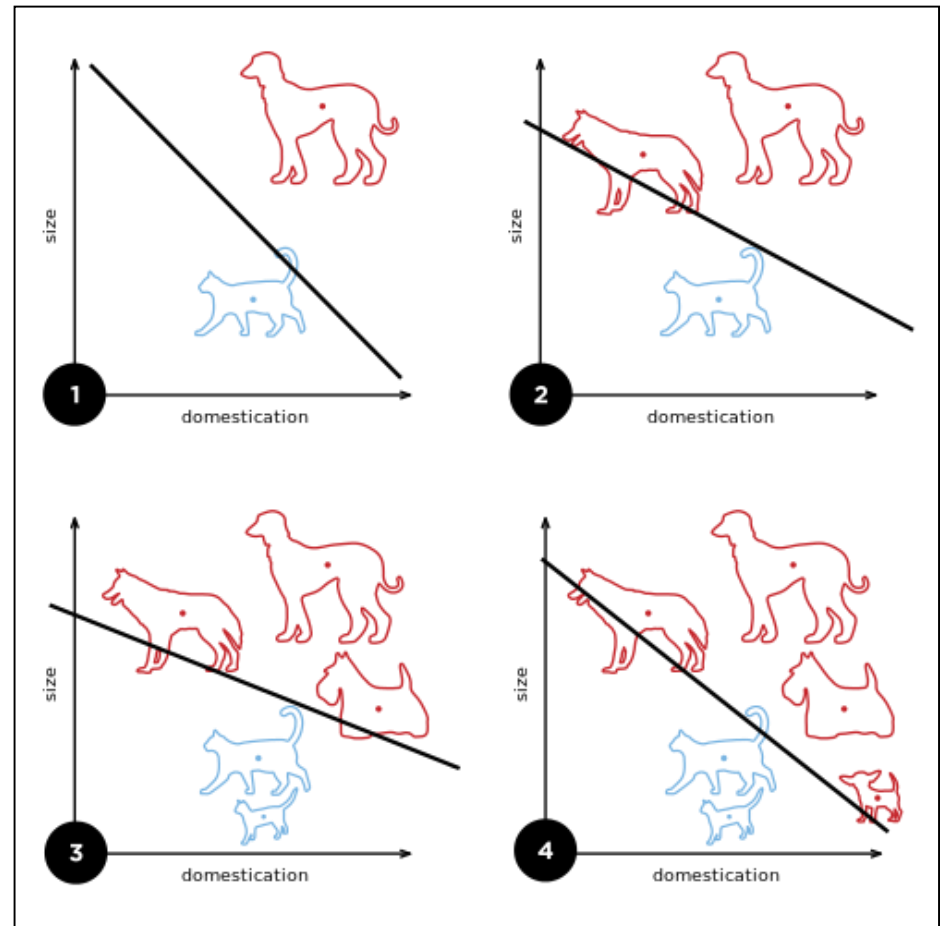
Recap: Perceptron learning

- Perceptron is an algorithm for learning a binary classifier.

$$f(x) = \begin{cases} 1 & \text{if } w \cdot x + b > 0 \\ 0 & \text{otherwise} \end{cases}$$

$$\sum_{i=1}^m w_i x_i$$

An example: A perceptron updating its linear boundary as more training examples are added. (Source: Wiki)



Recap: Building multi-layer nets using Perceptrons

- smaller example: can combine perceptrons to perform more complex computations (or classifications)

3-layer neural net

2 input nodes

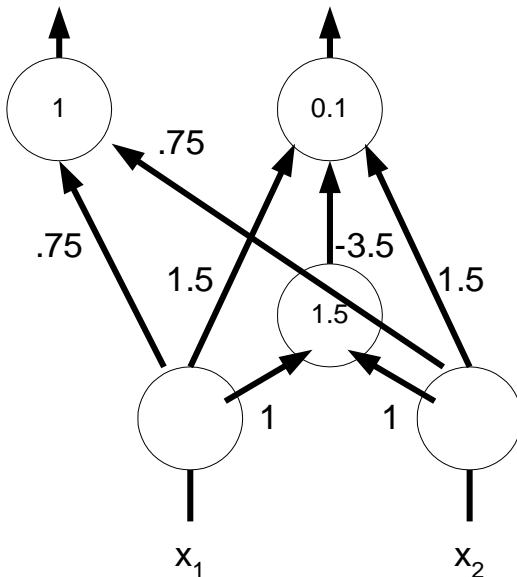
1 hidden node

2 output nodes

RESULT?

HINT: left output node is AND
right output node is XOR

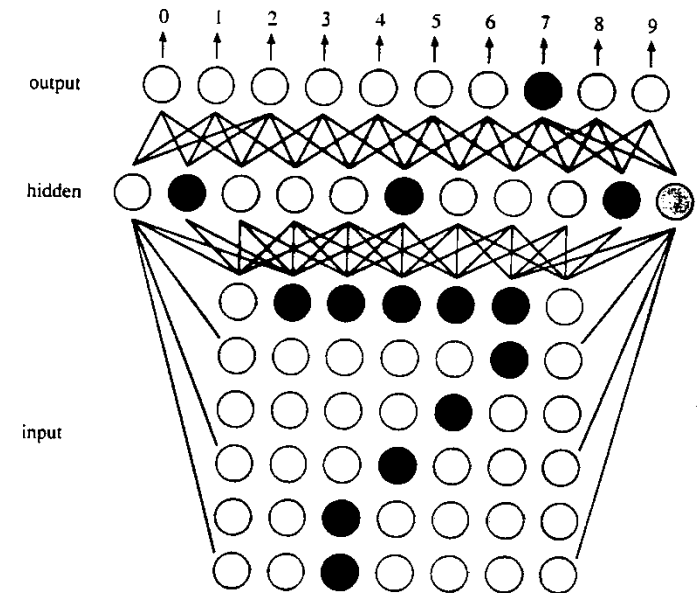
HALF ADDER



The introduction of hidden layer(s) makes it possible for the network to exhibit **non-linear behavior**.

Hidden units

- the addition of hidden units allows the network to develop complex feature detectors (i.e., internal representations)
- e.g., Optical Character Recognition (OCR)
 - ▣ perhaps one hidden unit
 - "looks for" a horizontal bar
 - ▣ another hidden unit
 - "looks for" a diagonal
 - ▣ another looks for the vertical base
 - ▣ the combination of specific
 - hidden units indicates a 7



Hidden units & learning

- every classification problem has a perceptron solution if enough hidden layers are used.
 - ▣ i.e., multi-layer networks can compute anything
 - (recall: can simulate AND, OR, NOT gates)

expressiveness is not the problem – learning is!

- it is not known how to systematically find solutions
- the Perceptron Learning Algorithm can't adjust weights between levels

Minsky & Papert's results about the "inadequacy" of perceptrons pretty much killed neural net research in the 1970's

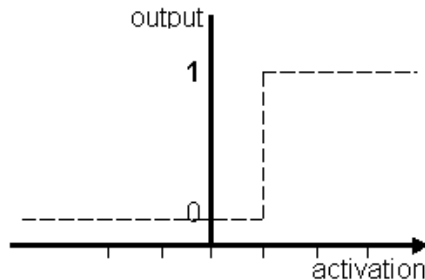
rebirth in the 1980's due to several developments

- faster, more parallel computers
- new learning algorithms e.g., backpropagation
- new architectures e.g., Hopfield nets

Backpropagation nets

backpropagation nets are multi-layer networks

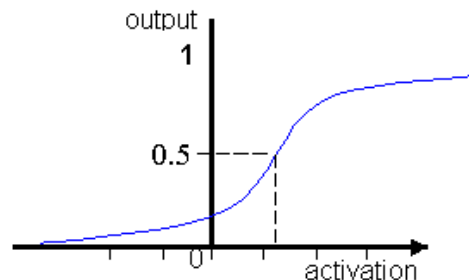
- normalize inputs between 0 (inhibit) and 1 (excite)
- utilize a continuous activation function



- perceptrons utilize a stepwise activation function

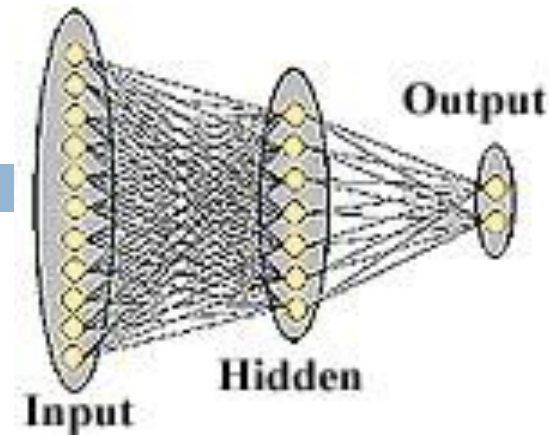
output = 1 if $\text{sum} \geq 0$

0 if $\text{sum} < 0$

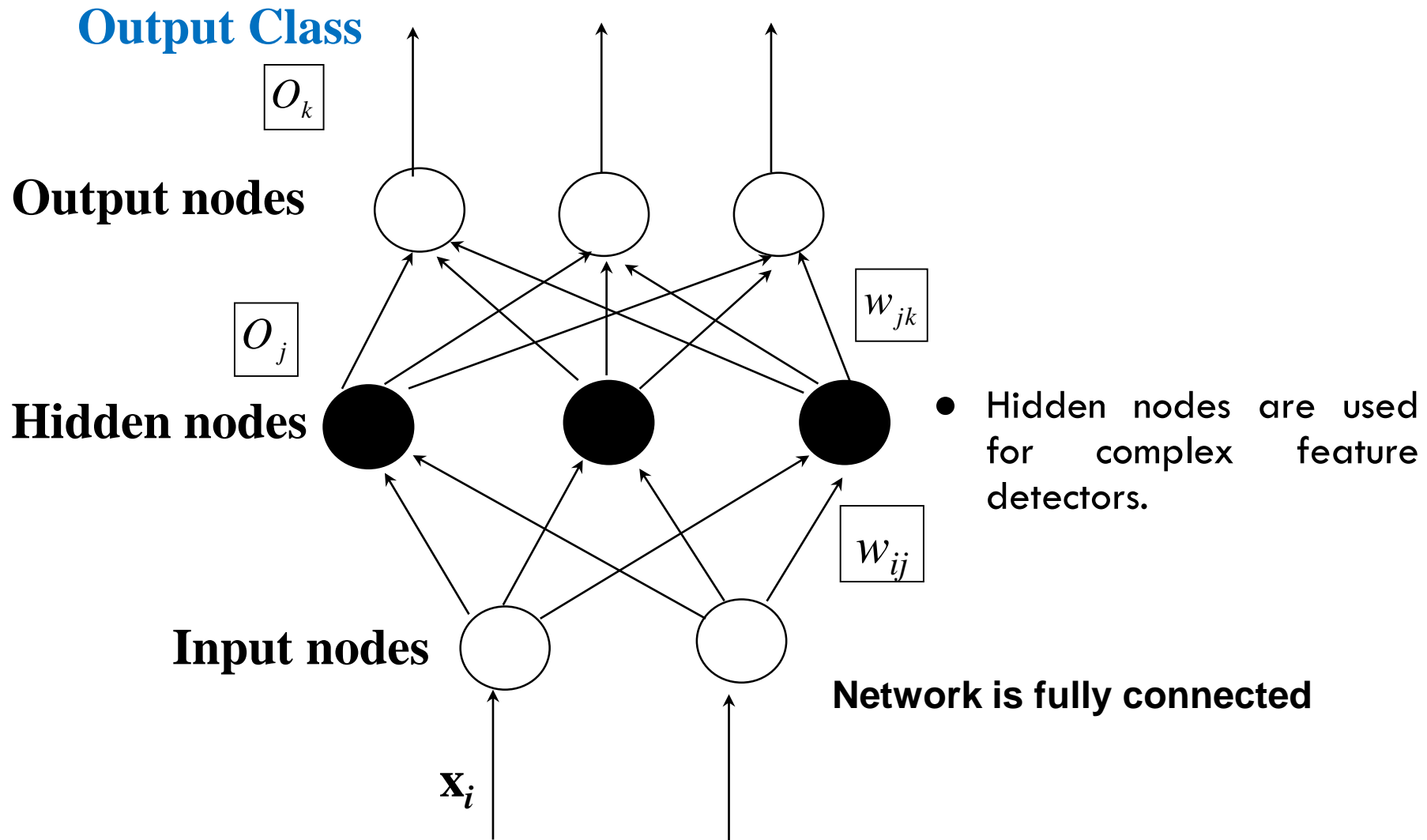


- backpropagation nets utilize a continuous activation function

output = $1 / (1 + e^{-\text{sum}})$



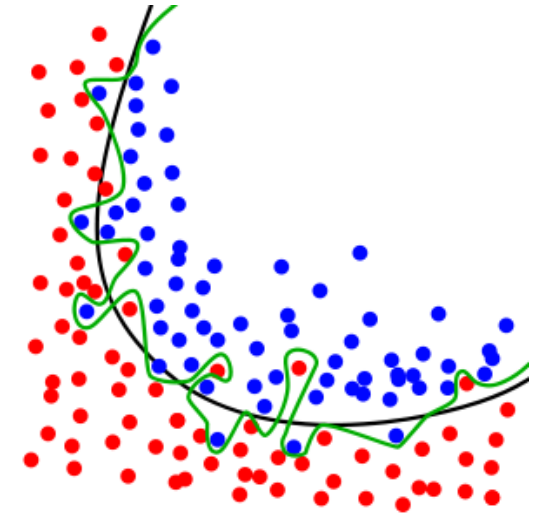
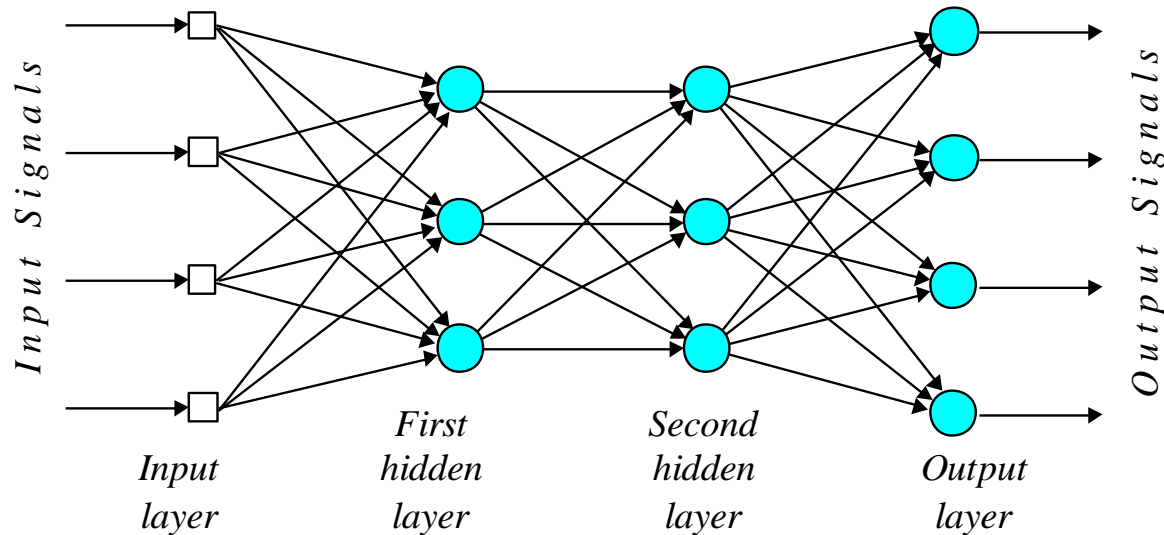
Multi layer feed-forward NN (FFNN)



What do middle layers hide?

- A hidden layer “hides” its desired output. Neurons in the hidden layer cannot be observed through the input/output behaviour of the network. There is no obvious way to know what the desired output of the hidden layer should be.

Multiple hidden layers & their neurons



- How many **hidden units** in the layer?
Too few ==> can't learn
Too many ==> poor generalization

Underfitting occurs when a model is **too simple** – informed by too few features or regularized too much – which makes it inflexible in learning from the dataset.

Overfitting is unable to generalize.

- How many hidden layers?
Usually just **one** (i.e., a 2-layer net)

Solu: Cross-validation **Remove features**

Early stopping

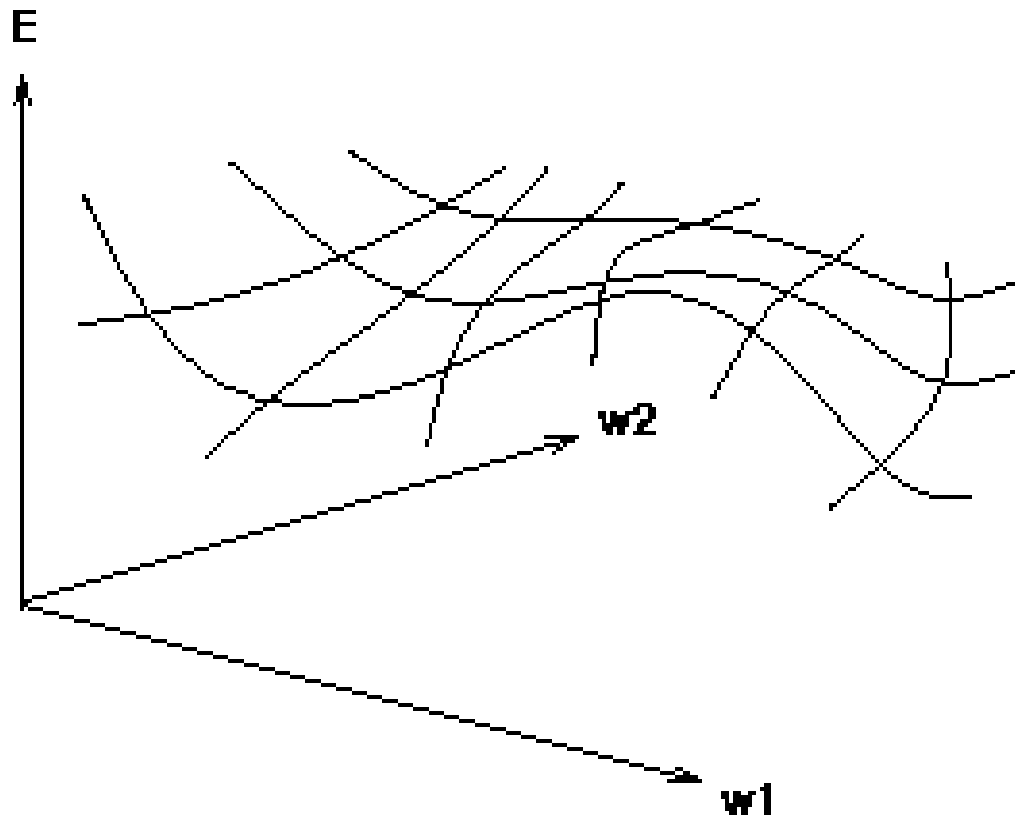
Ensembling

Regularization (pruning a DT, dropout on a neural network)

BPN Algorithm

- Initialize weights (typically random!)
- Keep doing epochs
 - **For each** example 'e' in the training set do
 - **forward pass** to compute
 - $O = \text{neural-net-output}(\text{network}, e)$
 - $\text{miss} = (T - O)$ at each output unit
 - **backward pass** to calculate **deltas** to weights
 - update all weights
 - end
- until **tuning set error stops improving**

Gradient Descent



Try to minimize your position on the “error surface.”

Error Backpropagation

- First calculate error of output units and use this to change the top layer of weights.

Current output: $o_j=0.2$

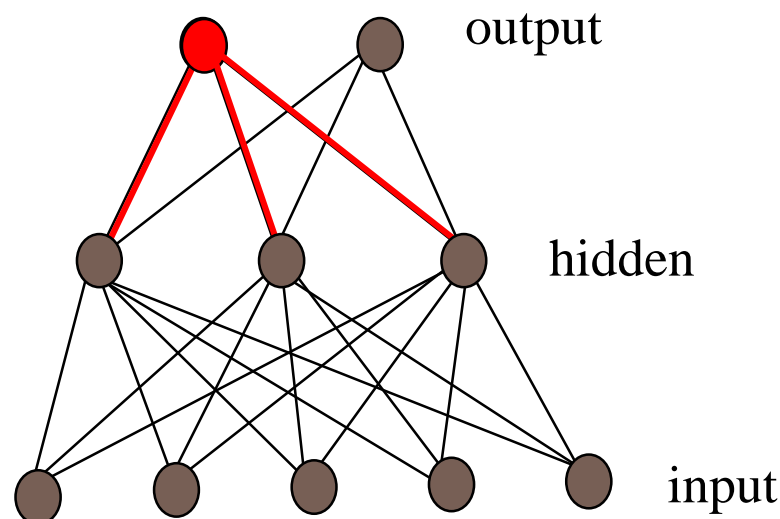
Correct output: $t_j=1.0$

Error $\delta_j = o_j(1-o_j)(t_j-o_j)$

$0.2(1-0.2)(1-0.2)=0.128$

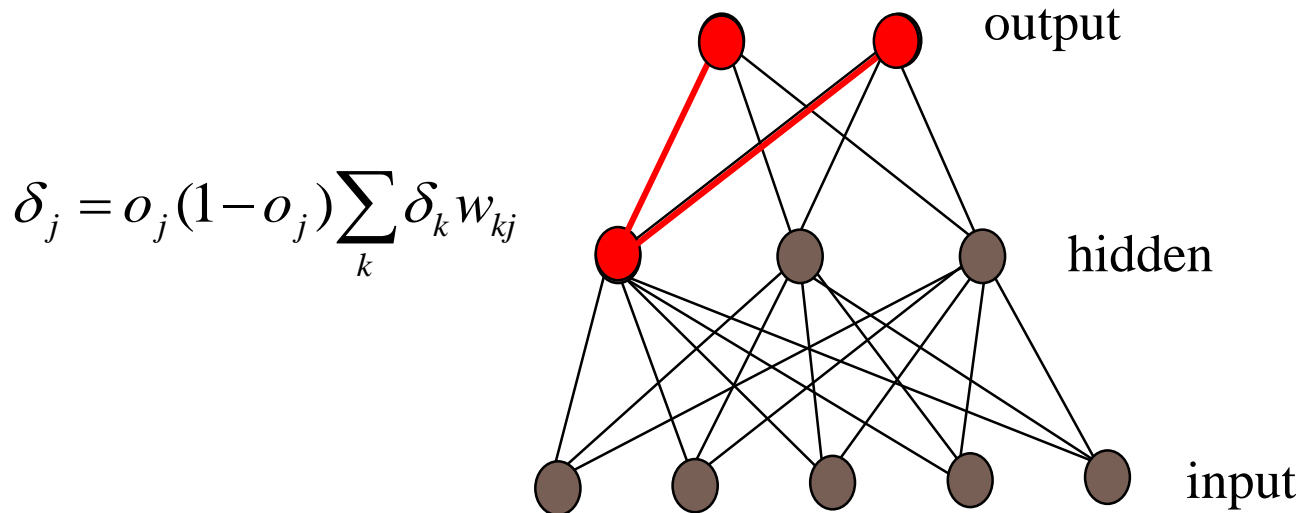
Update weights into j

$$\Delta w_{ji} = \eta \delta_j o_i$$



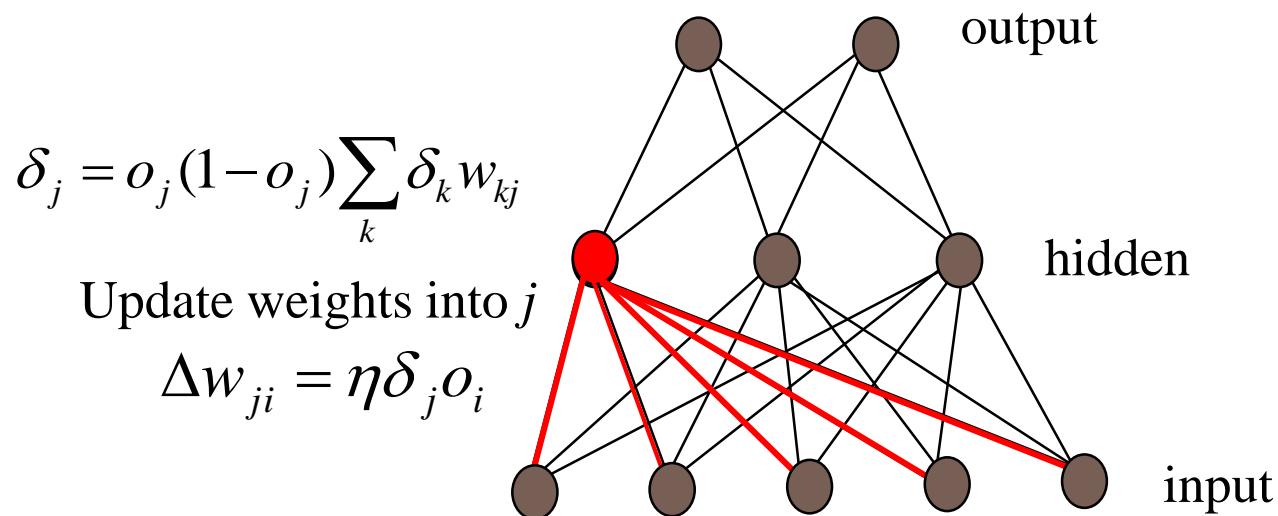
Error Backpropagation

- Next calculate error for hidden units based on errors on the output units it feeds into.

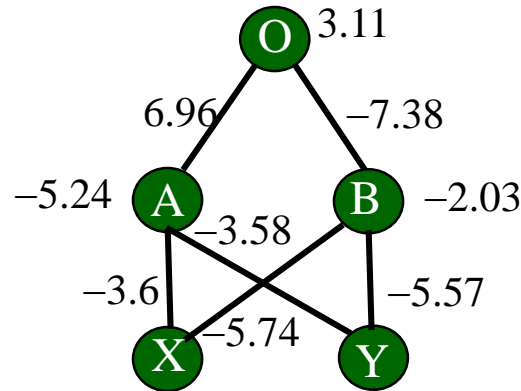


Error Backpropagation

- Finally update bottom layer of weights based on errors calculated for hidden units.



Sample Learned XOR Network

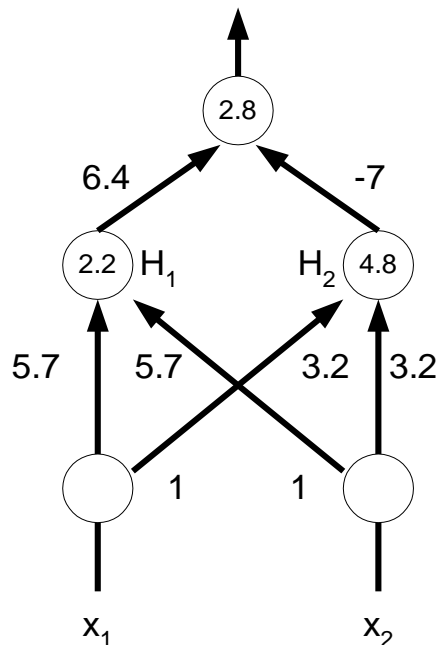


Hidden Unit A represents: $\neg(X \wedge Y)$

Hidden Unit B represents: $\neg(X \vee Y)$

Output O represents: $A \wedge \neg B = \neg(X \wedge Y) \wedge (X \vee Y)$
 $= X \oplus Y$

Backpropagation example (XOR)



$$x_1 = 1, x_2 = 1$$

$$\text{sum}(H_1) = -2.2 + 5.7 + 5.7 = 9.2, \text{ output}(H_1) = 0.99$$

$$\text{sum}(H_2) = -4.8 + 3.2 + 3.2 = 1.6, \text{ output}(H_2) = 0.83$$

$$\text{sum} = -2.8 + (0.99 \cdot 6.4) + (0.83 \cdot -7) = -2.28, \text{ output} = 0.09$$

$$x_1 = 1, x_2 = 0$$

$$\text{sum}(H_1) = -2.2 + 5.7 + 0 = 3.5, \text{ output}(H_1) = 0.97$$

$$\text{sum}(H_2) = -4.8 + 3.2 + 0 = -1.6, \text{ output}(H_2) = 0.17$$

$$\text{sum} = -2.8 + (0.97 \cdot 6.4) + (0.17 \cdot -7) = 2.22, \text{ output} = 0.90$$

$$x_1 = 0, x_2 = 1$$

$$\text{sum}(H_1) = -2.2 + 0 + 5.7 = 3.5, \text{ output}(H_1) = 0.97$$

$$\text{sum}(H_2) = -4.8 + 0 + 3.2 = -1.6, \text{ output}(H_2) = 0.17$$

$$\text{sum} = -2.8 + (0.97 \cdot 6.4) + (0.17 \cdot -7) = 2.22, \text{ output} = 0.90$$

$$x_1 = 0, x_2 = 0$$

$$\text{sum}(H_1) = -2.2 + 0 + 0 = -2.2, \text{ output}(H_1) = 0.10$$

$$\text{sum}(H_2) = -4.8 + 0 + 0 = -4.8, \text{ output}(H_2) = 0.01$$

$$\text{sum} = -2.8 + (0.10 \cdot 6.4) + (0.01 \cdot -7) = -2.23, \text{ output} = 0.10$$

Backpropagation example

- consider the following political poll, taken by six potential voters
 - ▣ each ranked various topics as to their importance, scale of 0 to 10
 - ▣ voters 1-3 identified themselves as Democrats, voters 4-6 as Republicans

	Economy	Defense	Crime	Environment
voter 1	9	3	4	7
voter 2	7	4	6	7
voter 3	8	5	8	4
voter 4	5	9	8	4
voter 5	6	7	6	2
voter 6	7	8	7	4

Backpropagation example (cont.)

- using the neural net, try to classify the following new respondents.

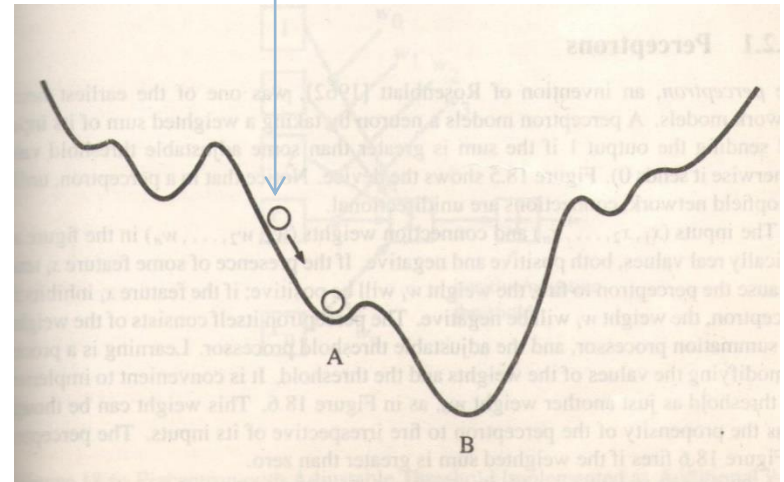
	Economy	Defense	Crime	Environment
voter 1	9	3	4	7
voter 2	7	4	6	7
voter 3	8	5	8	4
voter 4	5	9	8	4
voter 5	6	7	6	2
voter 6	7	8	7	4
voter 7	10	10	10	1
voter 8	5	2	2	7
voter 9	8	3	3	3

Recurrent networks

- FFNN is acyclic where data passes from input to the output nodes and not vice versa.
 - Once the FFNN is trained, its state is fixed and does not alter as new data is presented to it. It **does not have memory**.
- Recurrent networks can have atleast one connection that go backward from output to input nodes and model **dynamic** systems (to do temporal processing).
 - In this way, a recurrent network's internal state can be altered as sets of input data are presented. **It can be said to have memory**.
 - It is useful in solving problems where the solution depends not just on the current inputs but on all previous inputs.
- Applications
 - predict stock market price
 - weather forecast

Hopfield Network: Example Recurrent Networks

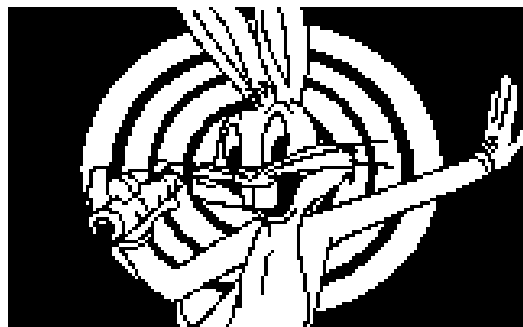
- A Hopfield network is a kind of recurrent network as output values are fed back to input in an undirected way. It has the following features:
 - Distributed representation.
 - Distributed, asynchronous control.
 - Content addressable memory.
 - Fault tolerance.



Example Hopfield net: Pattern completion and association

- The purpose of a Hopfield net is to store one or more patterns and to recall the full patterns based on partial input.

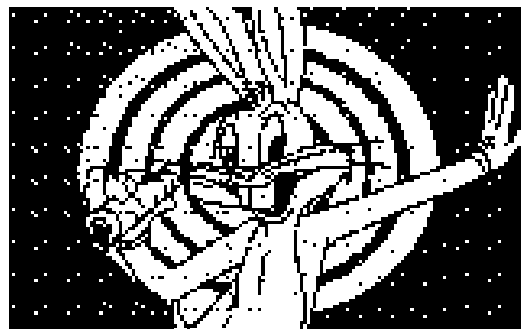
Original



Degraded



Reconstruction



Hopfield network reconstructing degraded images
from noisy (top) or partial (bottom) cues.

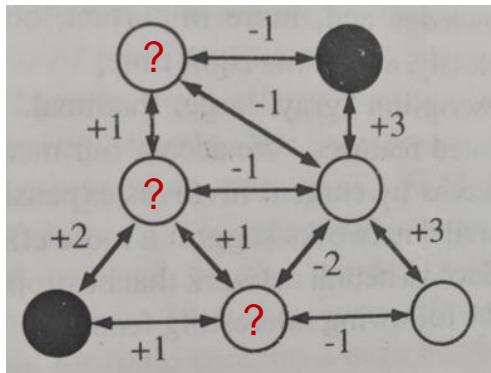
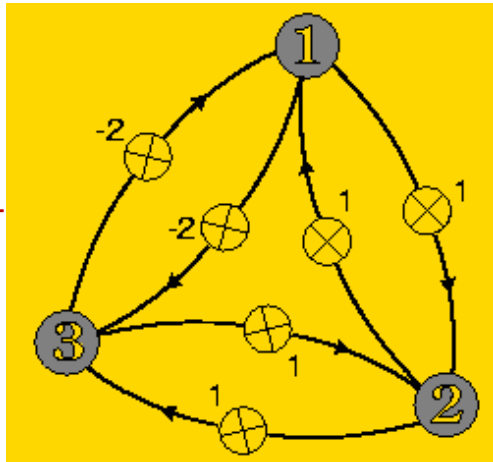
Activation Algorithm: **Parallel relaxation**

Active unit represented by 1 and inactive by 0.

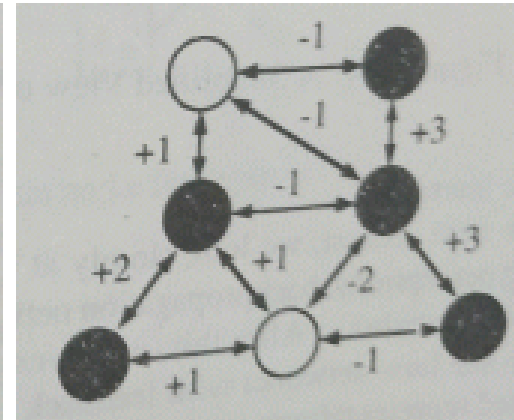
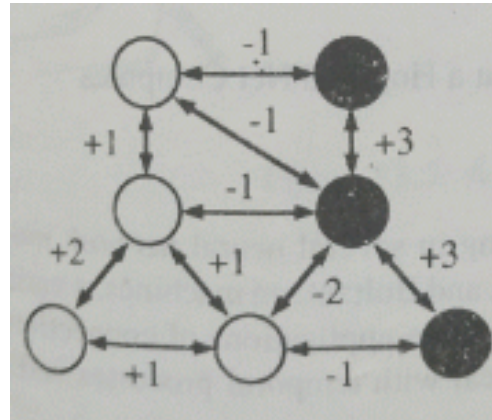
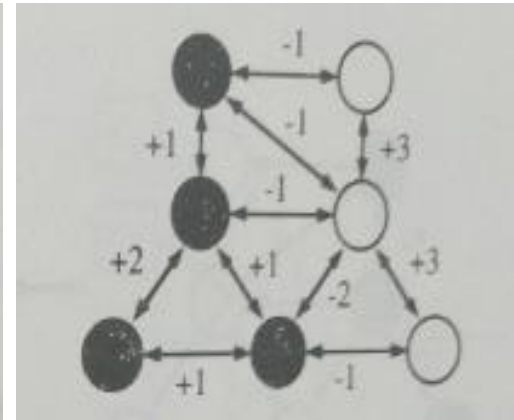
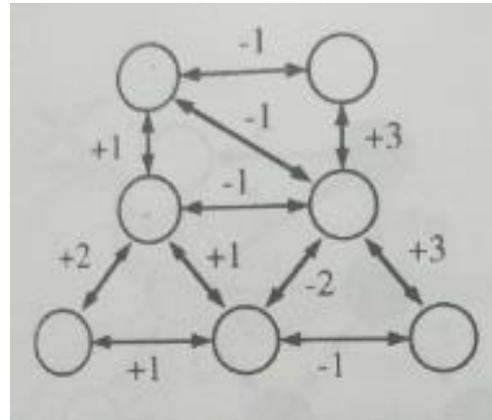
- **Repeat**
 - Choose any unit **randomly**. The chosen unit may be active or inactive.
 - For the chosen unit, compute the sum of the weights on the connection to the active neighbours only, if any.
 - If $\text{sum} > 0$ (threshold is assumed to be 0), then the chosen unit becomes active, otherwise it becomes inactive.
 - If chosen unit has no active neighbours then ignore it, and status remains same.
- **Until** the network reaches to a stable state

Example Hopfield nets

A Three node Hopfield Network



A Seven node Hopfield Network



Four stable states of the Hopfield Network (given any initial state, the network will settle into one of these four states, hence stores these patterns)

In a Hopfield network, all the nodes are inputs to each other, and they're also outputs.

Information being sent back and forth does not change after a few iterations (stability)

Weight Computation Method

- Weights are determined using training examples.

$$W = \sum X_i \cdot (X_i)^T - M.I, \text{ for } 1 \leq i \leq M$$

- Here
 - W is weight matrix
 - X_i is an input example represented by a vector of N values from the set $\{-1, 1\}$.
 - Here, N is the number of units in the network; 1 and -1 represent active and inactive units respectively.
 - $(X_i)^T$ is the transpose of the input X_i ,
 - M denotes the number of training input vectors,
 - I is an $N \times N$ identity matrix.

Example

- Let us now consider a **Hopfield network** with four units and three training input vectors that are to be learnt by the network.
- Consider three input examples, namely, X_1 , X_2 , and X_3 defined as follows:

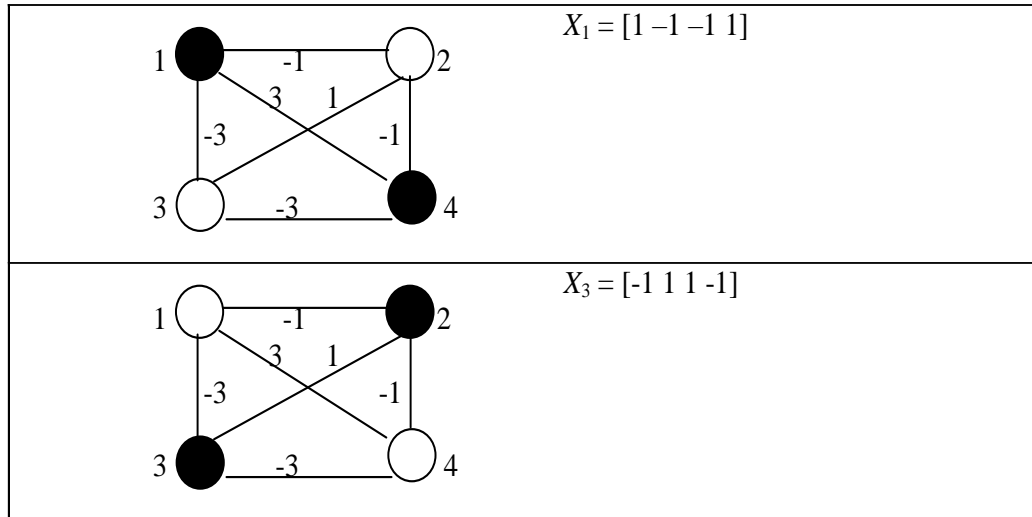
$$X_1 = \begin{pmatrix} 1 \\ -1 \\ -1 \\ 1 \end{pmatrix}$$

$$X_2 = \begin{pmatrix} 1 \\ 1 \\ -1 \\ 1 \end{pmatrix}$$

$$X_3 = \begin{pmatrix} -1 \\ 1 \\ 1 \\ -1 \end{pmatrix}$$

$$W = X_1 \cdot (X_1)^T + X_2 \cdot (X_2)^T + X_3 \cdot (X_3)^T - 3.I$$

Continued...

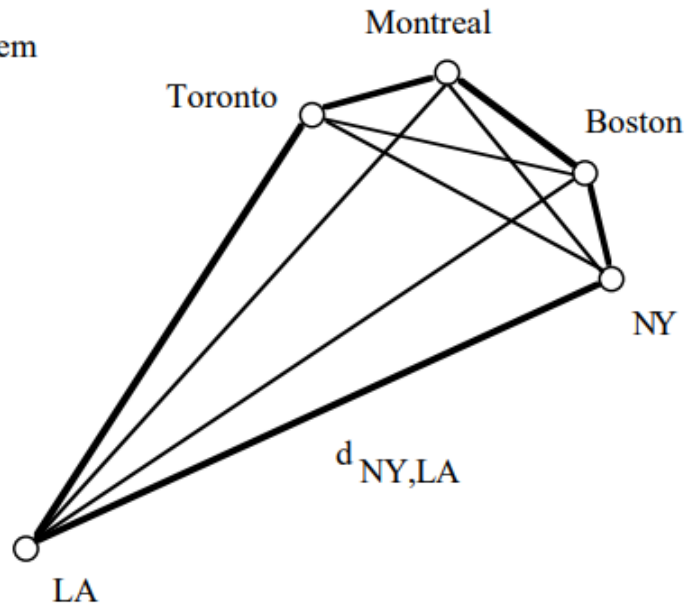


Stable positions of the network

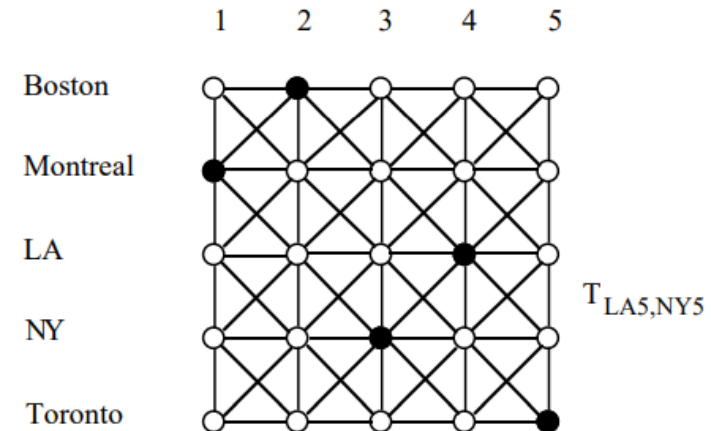
$$W = \begin{pmatrix} 3 & -1 & -3 & 3 \\ -1 & 3 & 1 & -1 \\ -3 & 1 & 3 & -3 \\ 3 & -1 & -3 & 3 \end{pmatrix} - \begin{pmatrix} 3 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 \\ 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 3 \end{pmatrix} = \begin{pmatrix} 0 & -1 & -3 & 3 \\ -1 & 0 & 1 & -1 \\ -3 & 1 & 0 & -3 \\ 3 & -1 & -3 & 0 \end{pmatrix}$$

Solving TSP using Hopfield net

(a) TSP problem



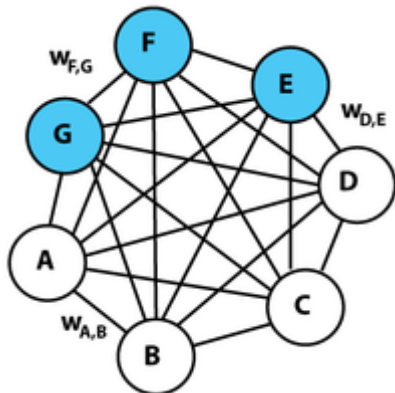
(b) Neural network representation



Source: The Traveling Salesman Problem: A Neural Network Perspective Jean-Yves Potvin Centre de Recherche sur les Transports Université de Montréal C.P. 6128, Succ. A, Montréal (Québec) Canada H3C 3J7

Boltzmann Machines

- A Boltzmann Machine is a variant of the Hopfield Network composed of N units with activations $\{x_i\}$. The state of each unit i is updated **asynchronously according** to the rule:



<https://en.wikipedia.org>

where

$$x_i = \begin{cases} +1 & \text{with probability } p_i \\ -1 & \text{with probability } 1 - p_i \end{cases}$$

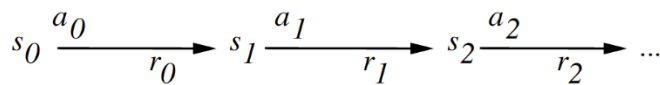
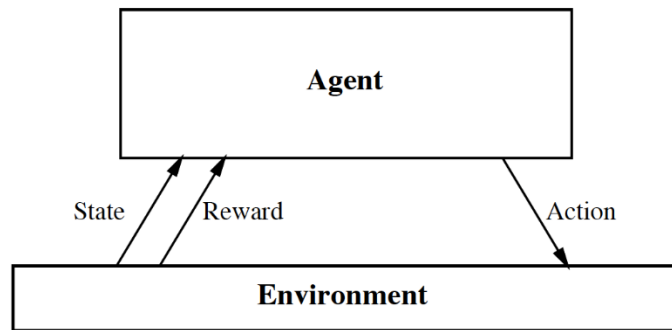
$$p_i = \frac{1}{1 + \exp\left(-\left(\sum_{j=1}^N w_{ij}x_j - \theta_j\right) / T\right)}$$

with positive temperature constant T , network weights w_{ij} and thresholds θ_i .

The fundamental difference between the Boltzmann Machine and a standard Hopfield Network is the **stochastic activation** of the units. If T is very small, the dynamics of the Boltzmann Machine approximates the dynamics of the discrete Hopfield Network, but when T is large, the network visits the whole state space.

Used in Deep learning

Reinforcement Learning



reward-motivated behaviour

- It allows machines and software agents to automatically determine the ideal behavior within a **specific context**, in order to maximize its performance. Simple reward feedback is required for the agent to learn its behavior; this is known as the reinforcement signal.

- A child to read more for the exam and score high
- A robot cleaning the room and recharging its battery
- How to invest in shares
- so on ..

Passive vs. Active learning

□ Passive learning

- ▣ The agent has a fixed policy and tries to learn the utilities of states by observing the world go by

□ Active learning

- ▣ The agent attempts to find an optimal (or at least good) policy by acting in the world

The algorithm, therefore, has a function that calculates the quality of a state-action combination:

$$Q : S \times A \rightarrow \mathbb{R} .$$

Before learning begins, Q is initialized to a possibly arbitrary fixed value (chosen by the programmer). Then, at each time t the agent selects an action a_t , observes a reward r_t , enters a new state s_{t+1} (that may depend on both the previous state s_t and the selected action), and Q is updated. The core of the algorithm is a simple **value iteration update**, using the weighted average of the old value and the new information:

$$Q(s_t, a_t) \leftarrow (1 - \alpha) \cdot \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \left(\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}} \right)$$

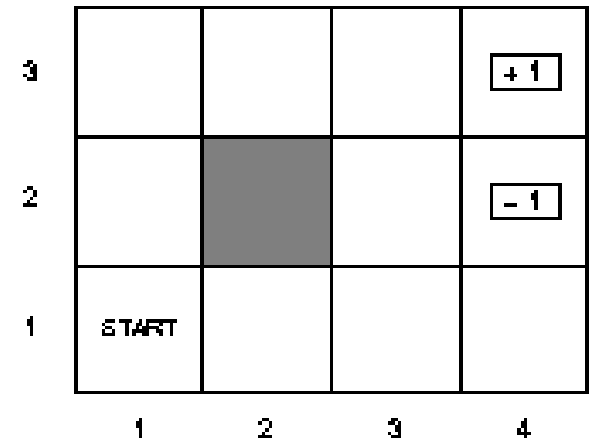
learned value

where r_t is the reward observed for the current state s_t , and α is the learning rate ($0 < \alpha \leq 1$).

<https://deepmind.com/blog/deep-reinforcement-learning/>

Passive RL

- Estimate $U^\pi(s)$
- Follow the policy for many epochs giving training sequences.

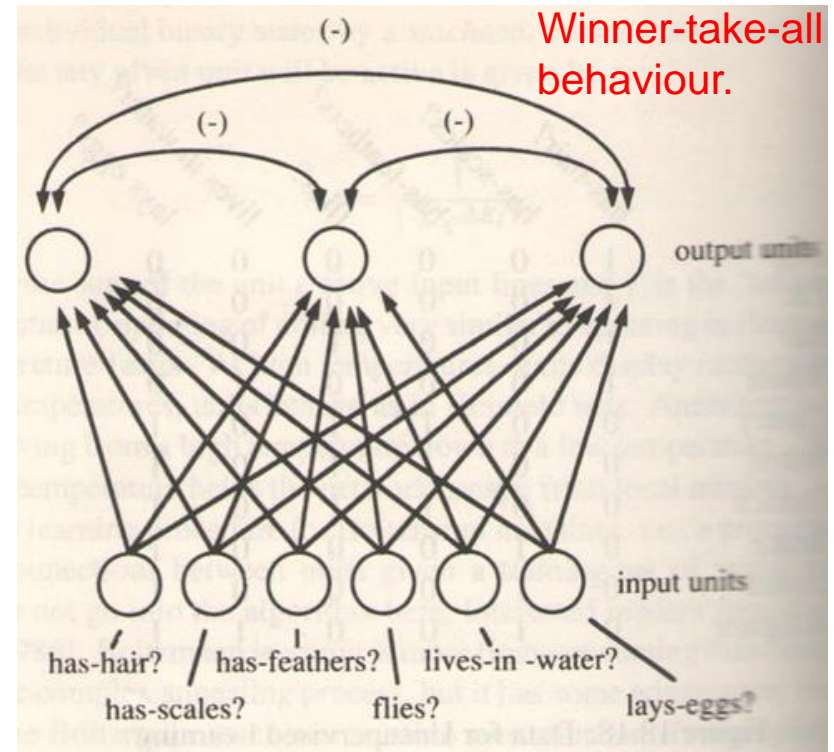


$(1,1) \rightarrow (1,2) \rightarrow (1,3) \rightarrow (1,2) \rightarrow (1,3) \rightarrow (2,3) \rightarrow (3,3) \rightarrow (3,4)$ +1
 $(1,1) \rightarrow (1,2) \rightarrow (1,3) \rightarrow (2,3) \rightarrow (3,3) \rightarrow (3,2) \rightarrow (3,3) \rightarrow (3,4)$ +1
 $(1,1) \rightarrow (2,1) \rightarrow (3,1) \rightarrow (3,2) \rightarrow (2,4)$ -1

- Assume that after entering +1 or -1 state the agent enters zero reward terminal state

Competitive Learning: Unsupervised learning using ANNs

	has-hair?	has-scales?	has-feathers?	flies?	lives in water?	lays eggs?
Dog	1	0	0	0	0	0
Cat	1	0	0	0	0	0
Bat	1	0	0	1	0	0
Whale	1	0	0	0	1	0
Canary	0	0	1	1	0	1
Robin	0	0	1	1	0	1
Ostrich	0	0	1	1	0	1
Snake	0	1	0	0	0	1
Lizard	0	1	0	0	0	1
Alligator	0	1	0	0	1	1



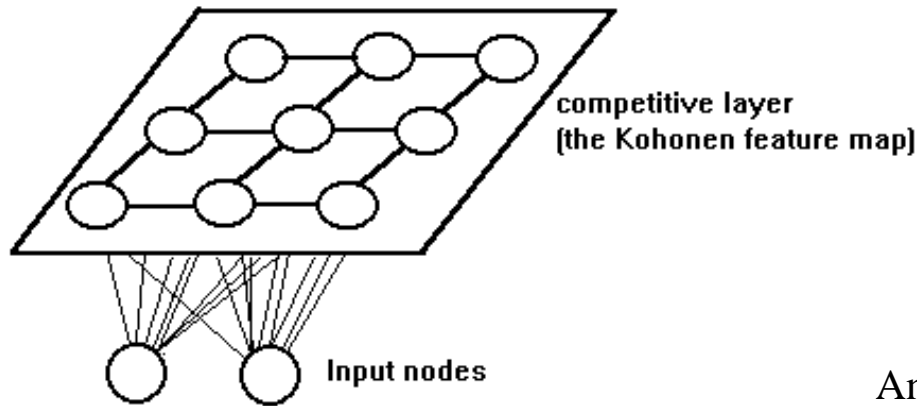
- Three groups: Mammals, reptiles and birds.
- With a teacher, we can of course use BPNs.
- How will you do it without a teacher?

$$\Delta w_j = \eta \frac{x_j}{m} - \eta w_j \quad \text{for all } j = 1, \dots, n$$

1. Present the input vector
2. Calculate the initial activation for each output unit
3. Let the output units fight till one is active
4. Increase the weights on connections between the active output and input units so that next time it will be more active
5. Repeat for many epochs.

$$d_j = \sum_i^{N-1} (x_i(t) - w_{ij}(t))^2$$

Self Organizing Maps (**Kohonen nets**) Example



Sum squared error for pattern p for all output layer neurons (deviations of predicted from actual):

$$E_p = \frac{1}{2} \sum_j (w_{ij} - x_j^p)^2$$

Any change Δw_{ij} in the weight is expected to cause a reduction in error E_p .

Rate of change of E_p with respect to any one weight value w_{ij} has to be measured by calculating its **partial derivative**

$$\Delta_p w_{ij} = -\eta \frac{\partial E_p}{\partial w_{ij}}$$

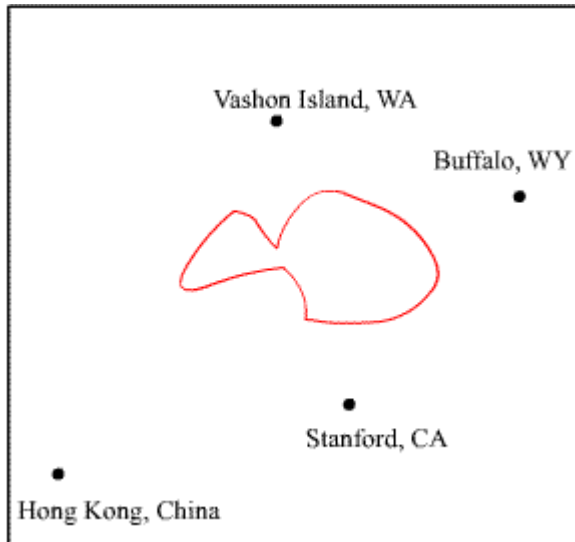
partial derivative of E_p .

$$\frac{\partial E_p}{\partial w_{ij}} = w_{ij} - x_j^p$$

$$\Delta_p w_{ij} = -\eta \frac{\partial E_p}{\partial w_{ij}} = -\eta (w_{ij} - x_j^p) = \eta (x_j^p - w_{ij})$$

TSP using Kohonen Self Organizing Nets

Img. Source: <https://cs.stanford.edu>



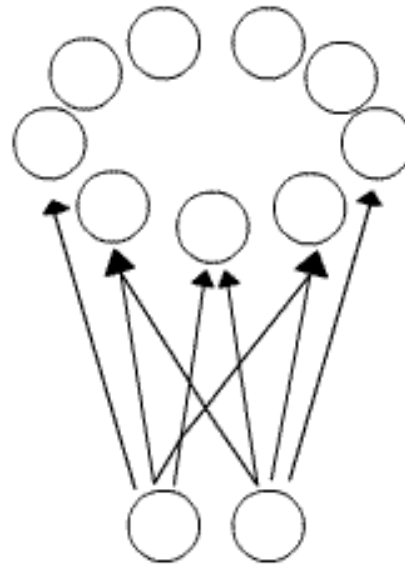
R
a
n
d
o
m

r
i
n
g

P
o
i
n
t

p
u
l
l
e
d

After many pulls for different cities, the net eventually converges into a ring around the cities. Given the elastic property of the ring, the length of the ring tends to be minimized. (TSP approximation)

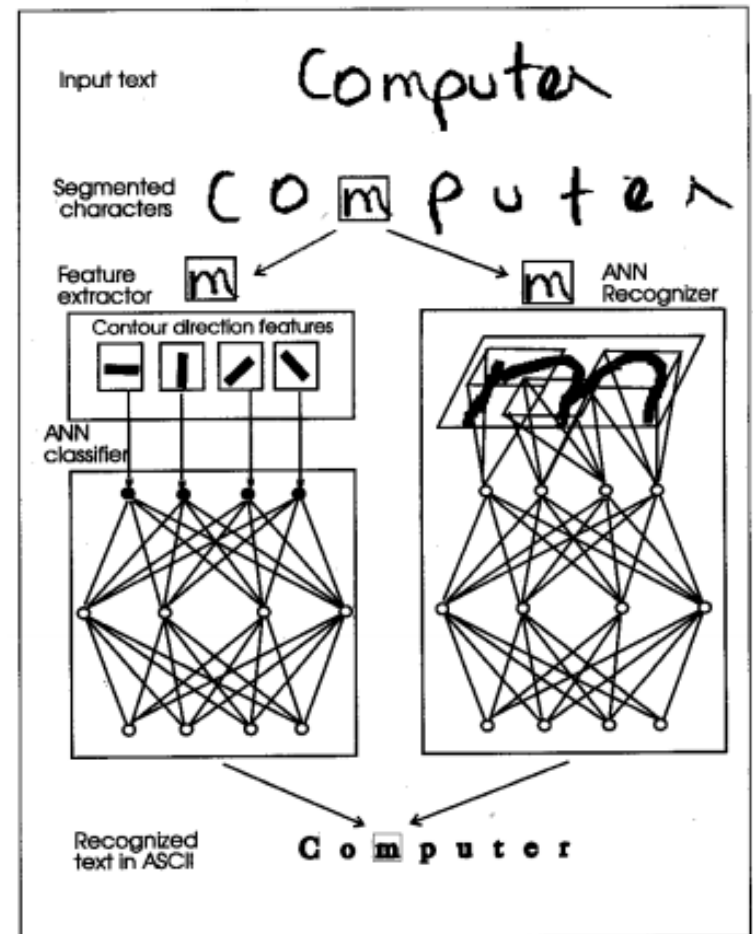
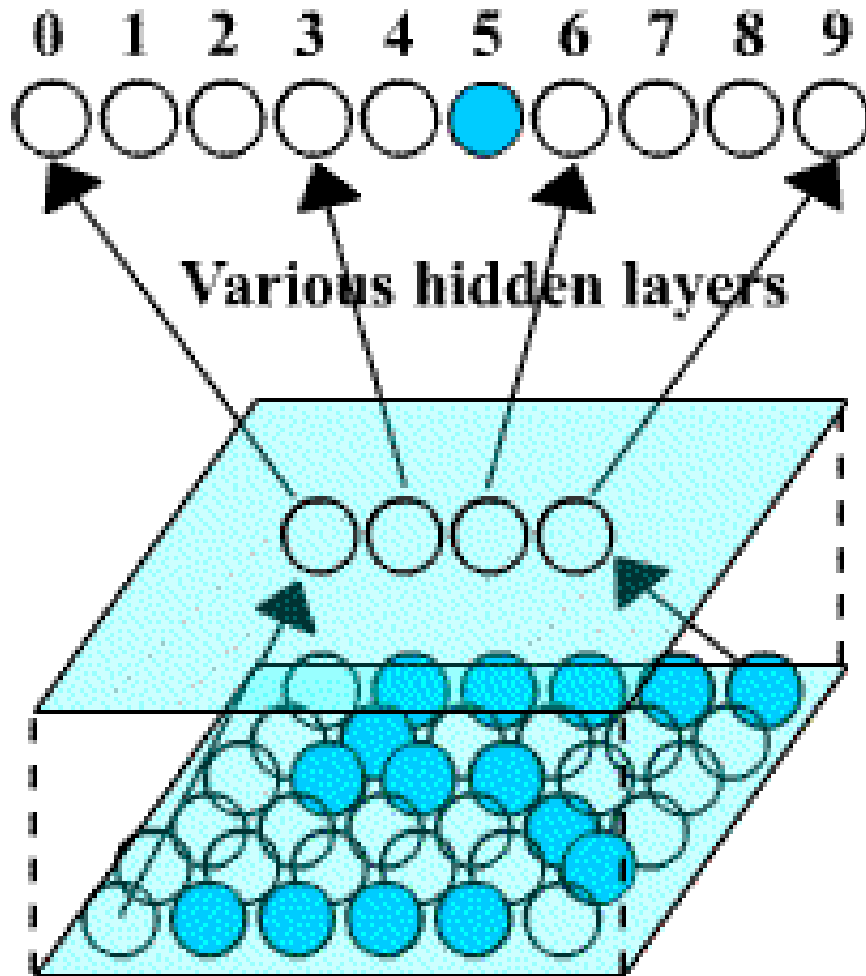


Consider the weight vectors as points on a plane.

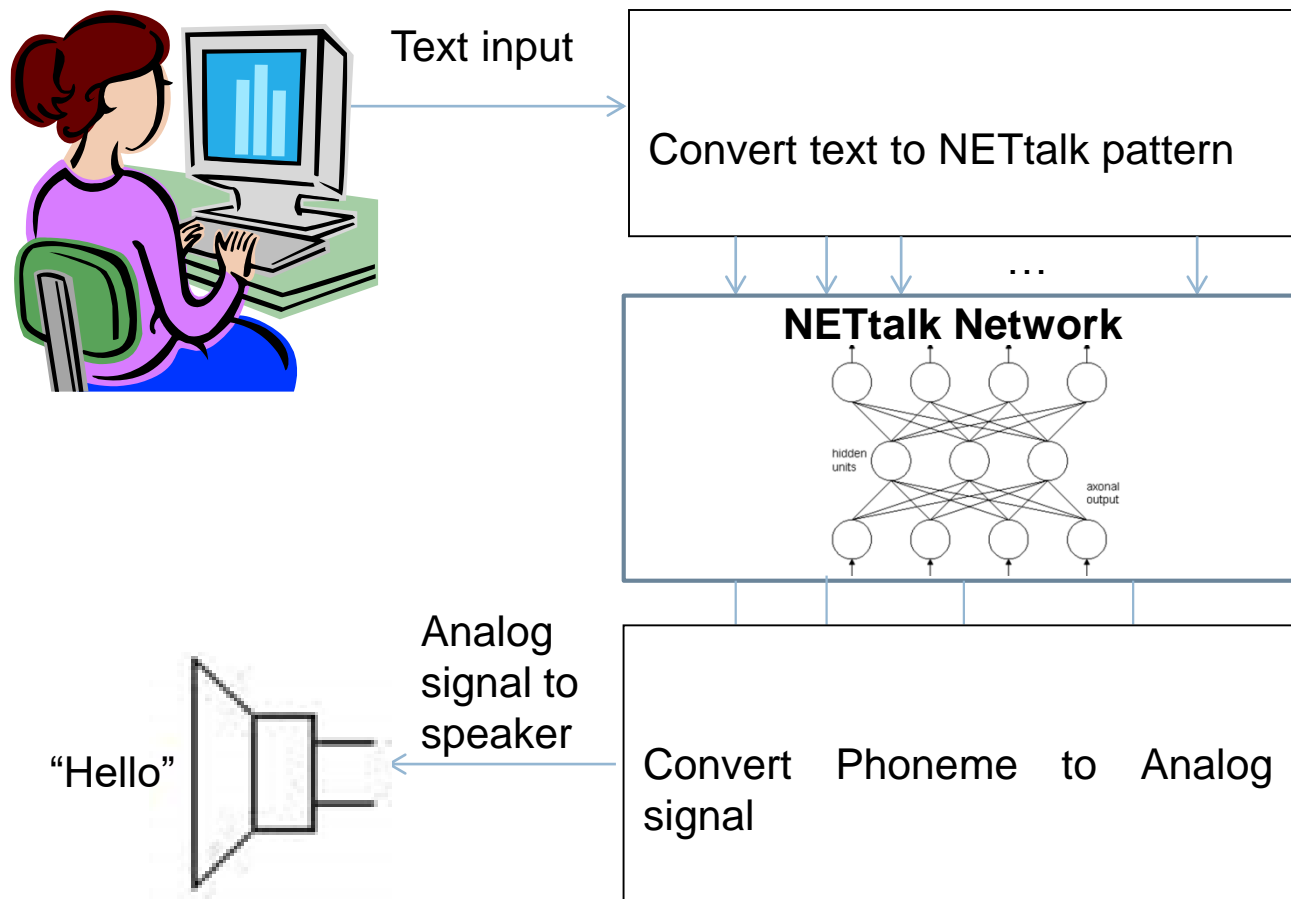
We can join these points together according to the position of their respective perceptron in the ring of the top layer of the network

Coordinates of a city (x, y) is presented as the input vector of the network, the network will identify the weight vector closest to the city and move it and its neighbors closer to the city.

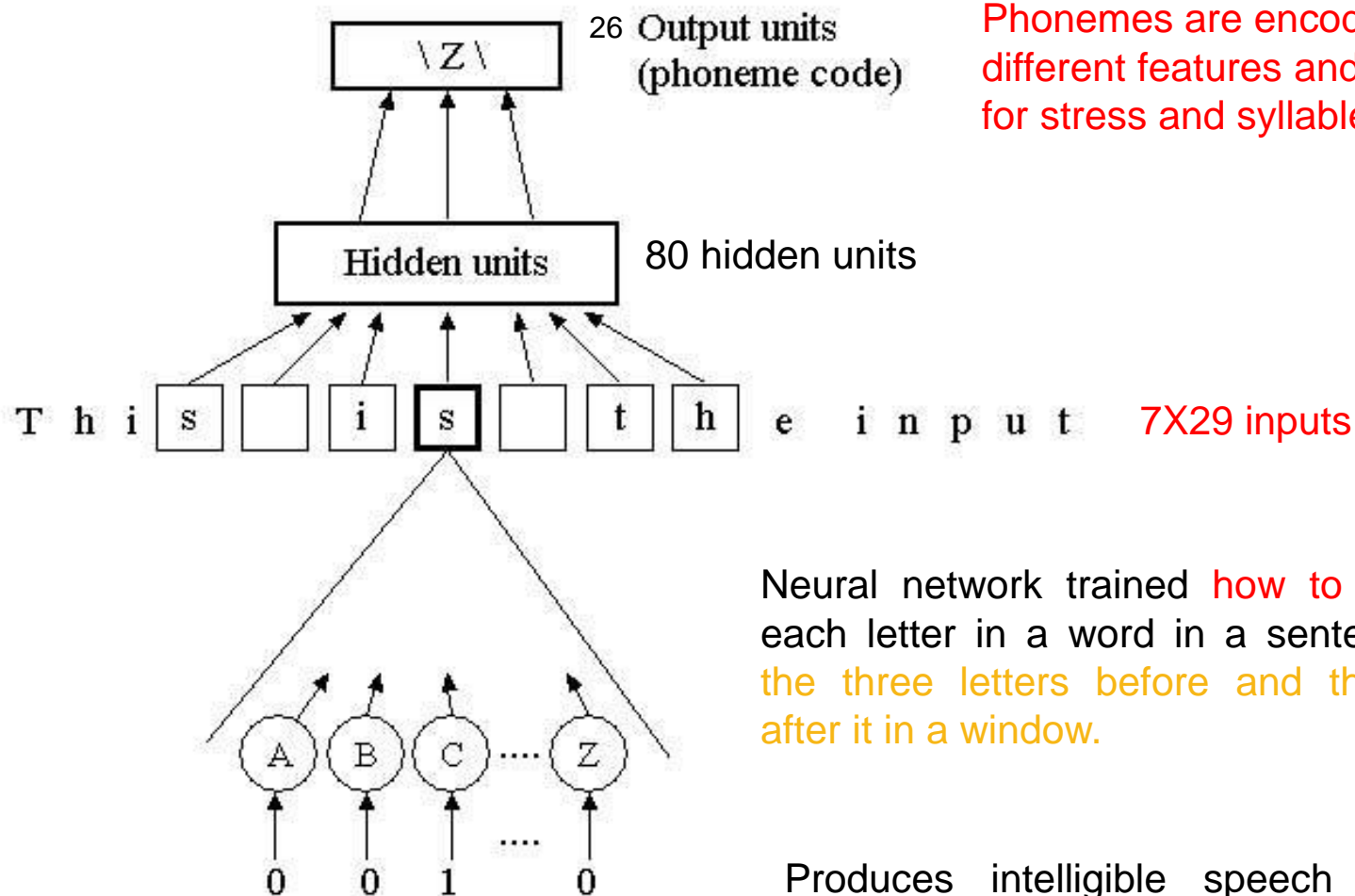
Applications of ANNs: OCR



Learned to Pronounce English text: NETtalk (Sejnowski and Rosenberg, 1987)



NETtalk continued...

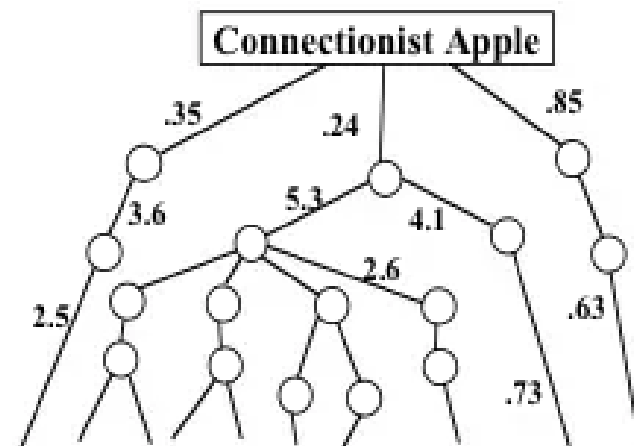
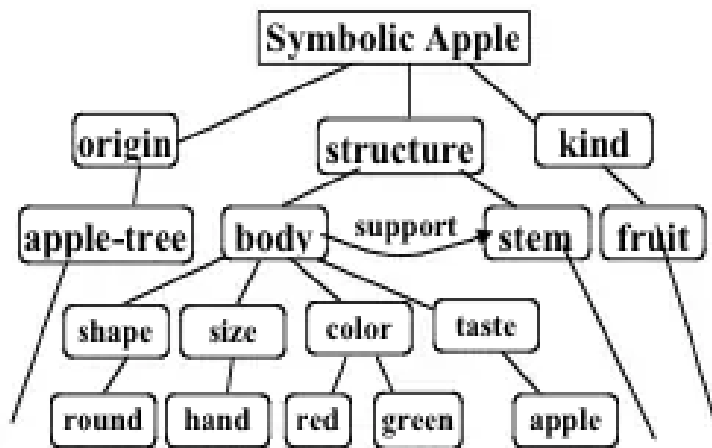


Phonemes are encoded using 21 different features and remaining 5 for stress and syllable boundaries

Neural network trained how to pronounce each letter in a word in a sentence, given the three letters before and three letters after it in a window.

Produces intelligible speech after 10 training epochs.

- symbol-based learning: the primary influence on learning is domain knowledge
 - version space search, decision trees, explanation-based learning.

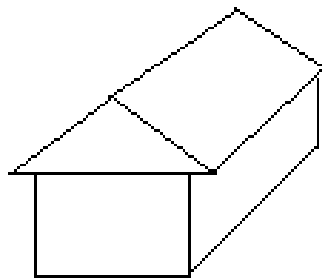


done

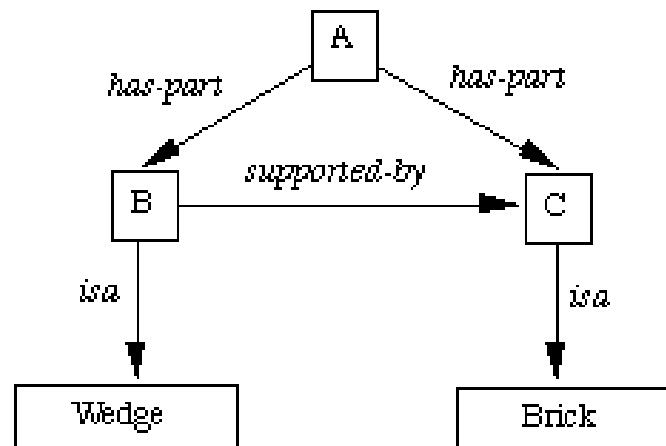
Learning from Examples: Inductive

□ Winston's Learning Program

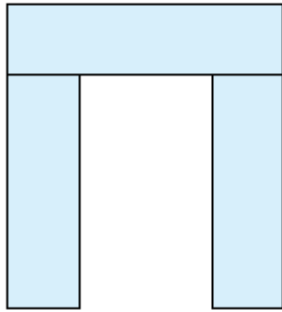
1. Select **one known instance** of the concept. Call this the concept definition.
2. Examine definitions of other known instances of the concept. **Generalize** the definition to include them.
3. Examine descriptions of near misses. **Restrict** the definition to exclude these.



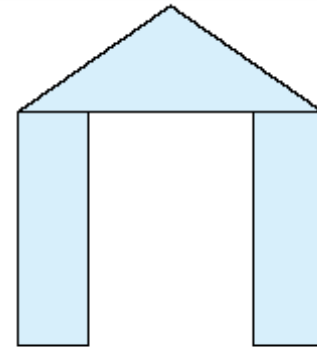
House



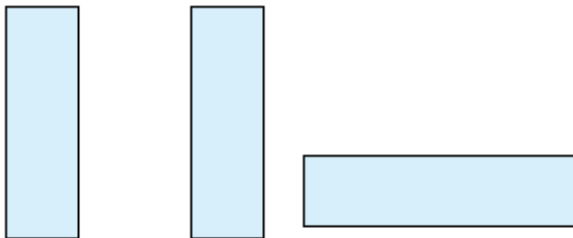
An Example: Learning concepts from positive and negative Examples



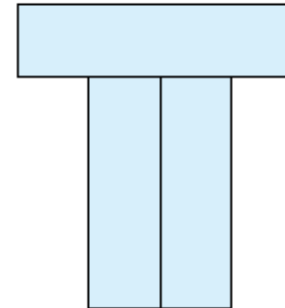
Arch



Arch



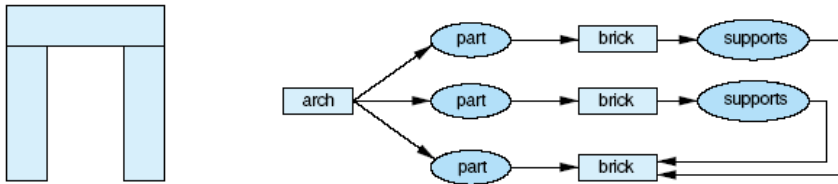
Near miss



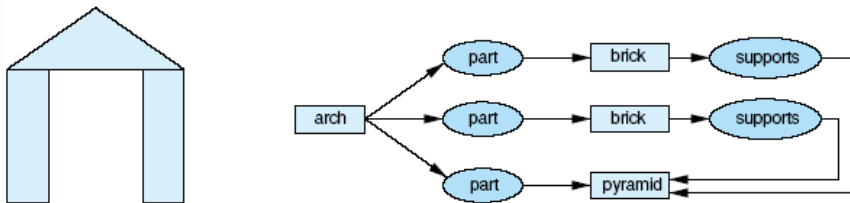
Near miss

Generalization of descriptions

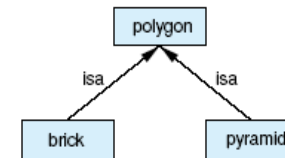
a. An example of an arch and its network description



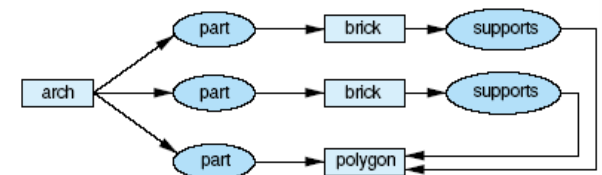
b. An example of another arch and its network description



c. Given background knowledge that bricks and pyramids are both types of polygons

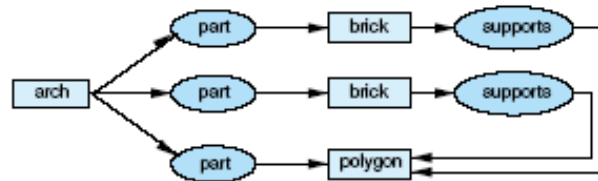


d. Generalization that includes both examples

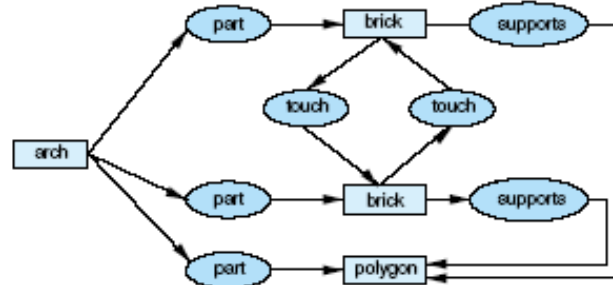
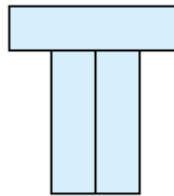


Specialization of a description

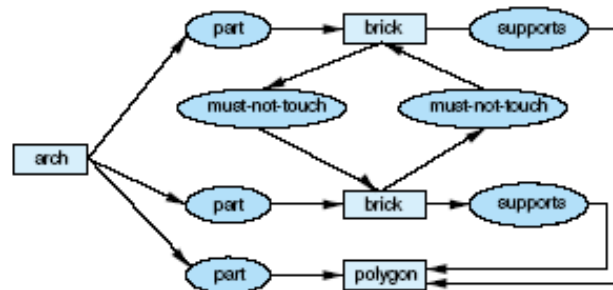
a. Candidate description of an arch



b. A near miss and its description



c. Arch description specialized to exclude the near miss



Issues with Structural concept learning

- The **teacher** must guide the system through **carefully chosen sequences of examples**.
- In Winston's program the **order** of the process is **important** since new links are added as and when new knowledge is gathered.
- The concept of **version spaces** is insensitive to order of the examples presented.

Example of Version space

Consider the task to obtain a description of the concept:
Japanese Economy car.

The attributes under consideration are:

Origin, Manufacturer, Color, Decade, Type

training data:

Positive ex: (Japan, Honda, Blue, 1980, Economy)

Positive ex: (Japan, Honda, White, 1980, Economy)

Negative ex: (Japan, Toyota, Green, 1970, Sports)

Continued...

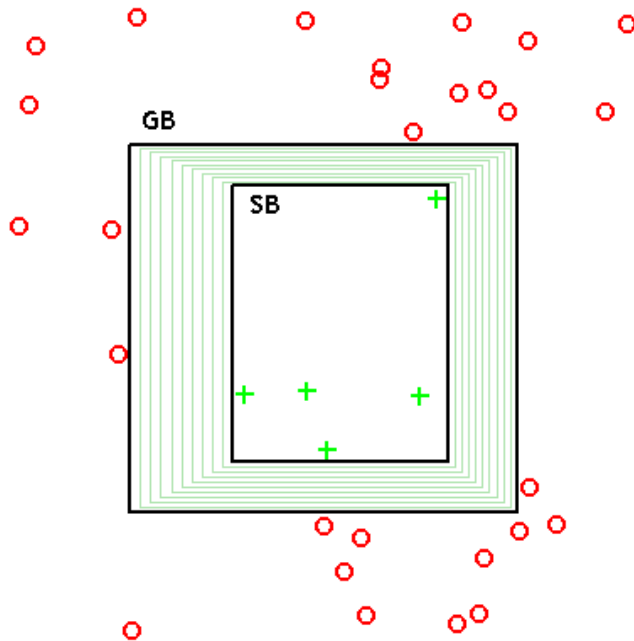
The most **general hypothesis** that matches the positive data and does not match the negative data, is:

$(x, \text{Honda}, x, x, \text{Economy})$

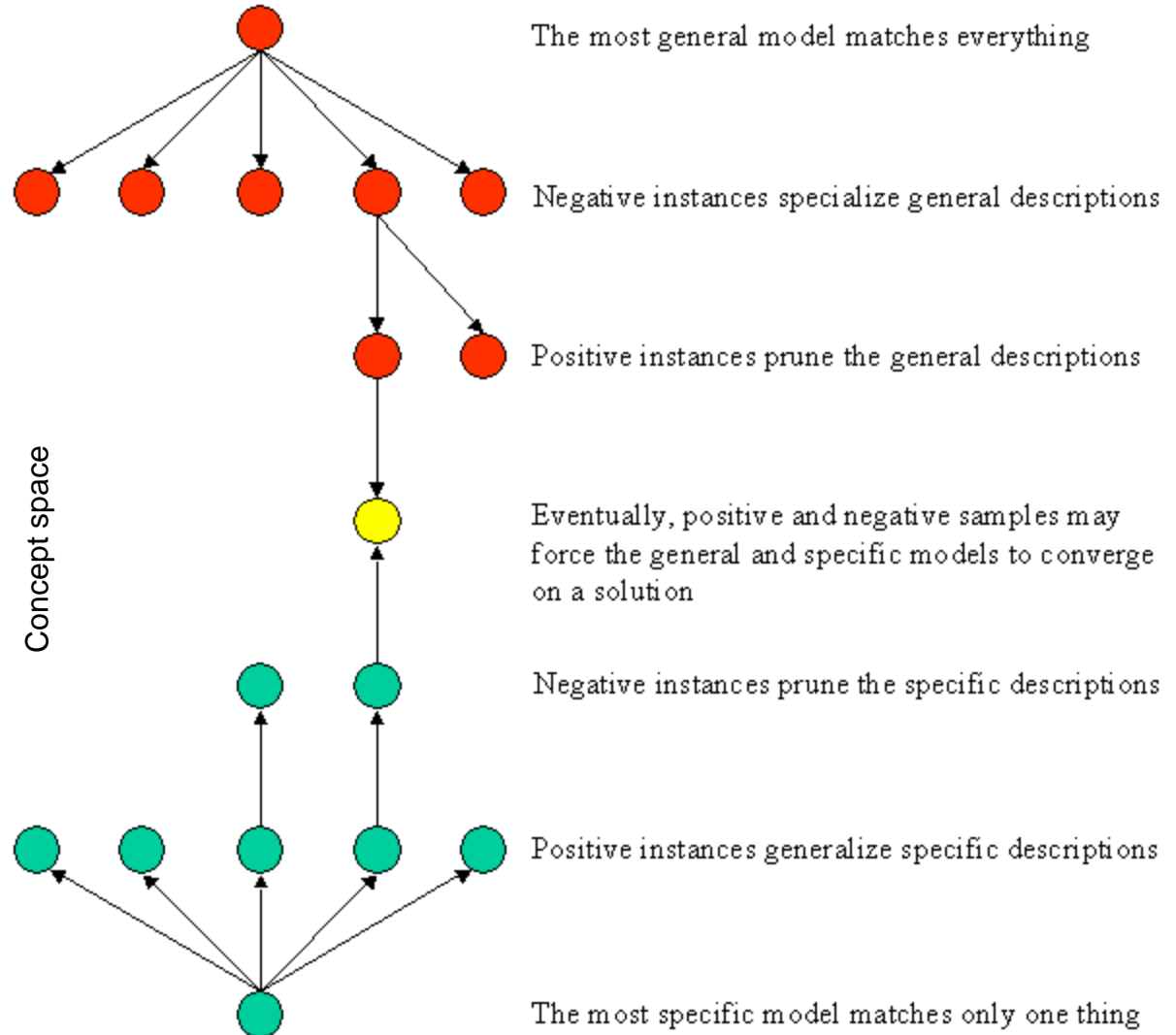
The most **specific hypothesis** that matches the positive examples is:

$(\text{Japan}, \text{Honda}, x, 1980, \text{Economy})$

Converging boundaries



Img. Source: Wiki



Candidate Elimination (Mitchell)

- Initialize G to contain one element: the null description i.e., the most general description (all features are variables).
- Initialize S to contain one element: the first positive example.
- Accept a new training example.

Process Positive Examples:

- Remove from G any descriptions that do not cover the example.
- Generalize S as little as possible so that the new training example is covered.
- Remove from S all elements that cover negative examples.

Algorithm continued...

Process Negative Examples:

- Remove from S any descriptions that cover the negative example.
- Specialize G as little as possible so that the negative example is not covered.
- Remove from G all elements that do not cover the positive examples.

Continue processing new training examples, until one of the following occurs:

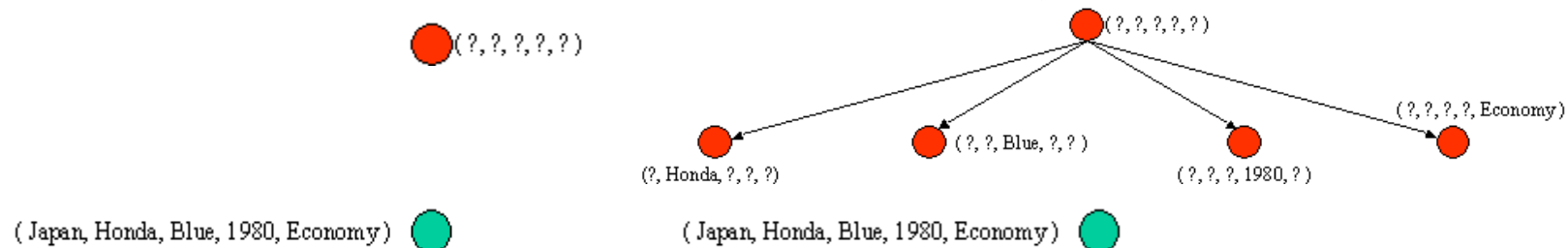
- Either S or G become empty, there are no consistent hypotheses over the training space. Stop.
- S and G are both singleton sets.
 - if they are identical, output their value and stop.
 - if they are different, the training cases were inconsistent. Output this result and stop.
- No more training examples. G has several hypotheses.

Example

Learning the concept of "Japanese Economy Car"

Origin	Manufacturer	Color	Decade	Type	Example Type
Japan	Honda	Blue	1980	Economy	Positive
Japan	Toyota	Green	1970	Sports	Negative
Japan	Toyota	Blue	1990	Economy	Positive
USA	Chrysler	Red	1980	Economy	Negative
Japan	Honda	White	1980	Economy	Positive

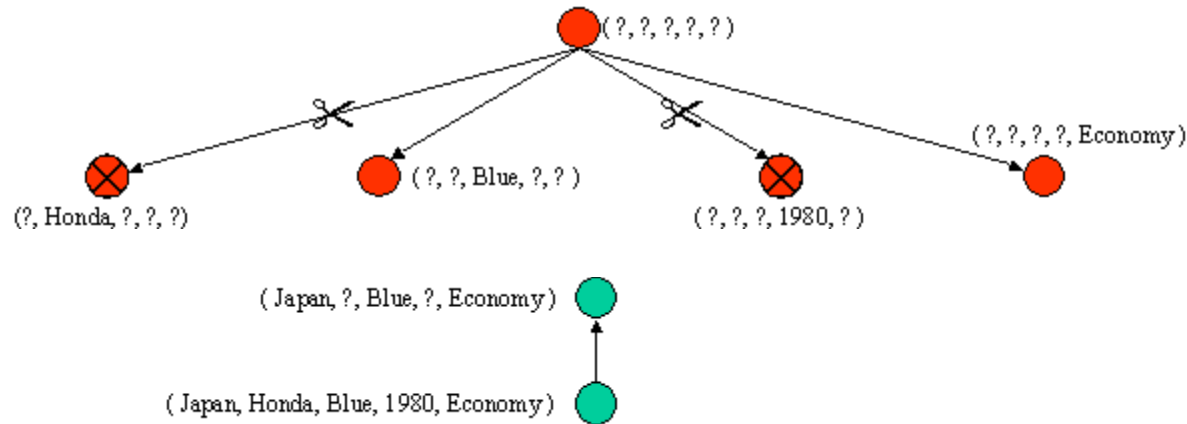
1. **Positive Example:** (Japan, Honda, Blue, 1980, Economy)



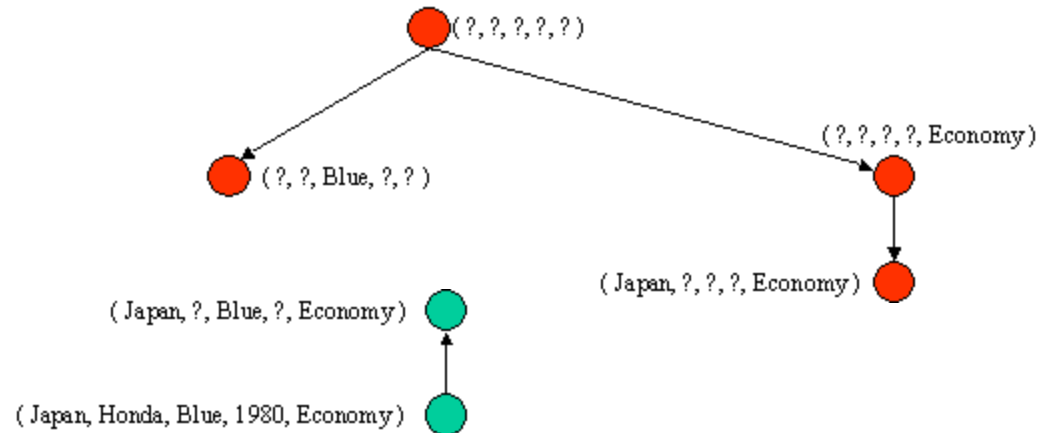
2. **Negative Example:** (Japan, Toyota, Green, 1970, Sports)

Example continued...

3. Positive Example: (Japan, Toyota, Blue, 1990, Economy)

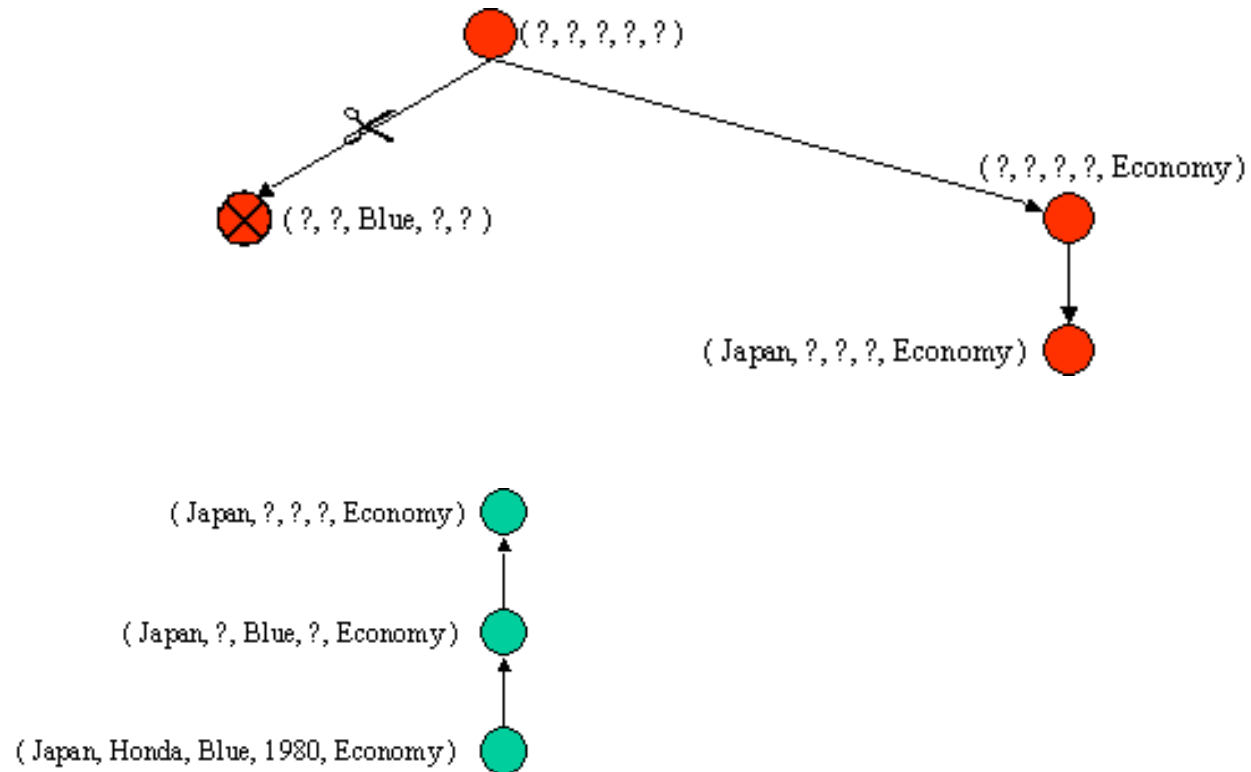


4. Negative Example: (USA, Chrysler, Red, 1980, Economy)



Example continued...

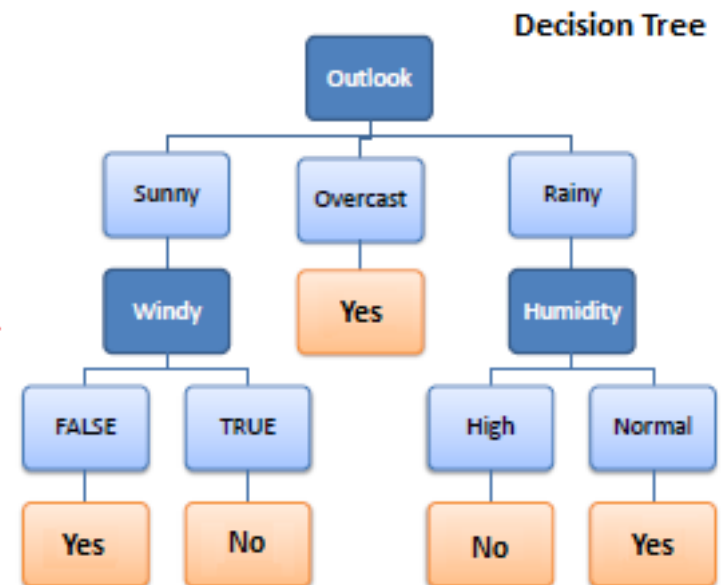
5. Positive Example: (Japan, Honda, White, 1980, Economy)



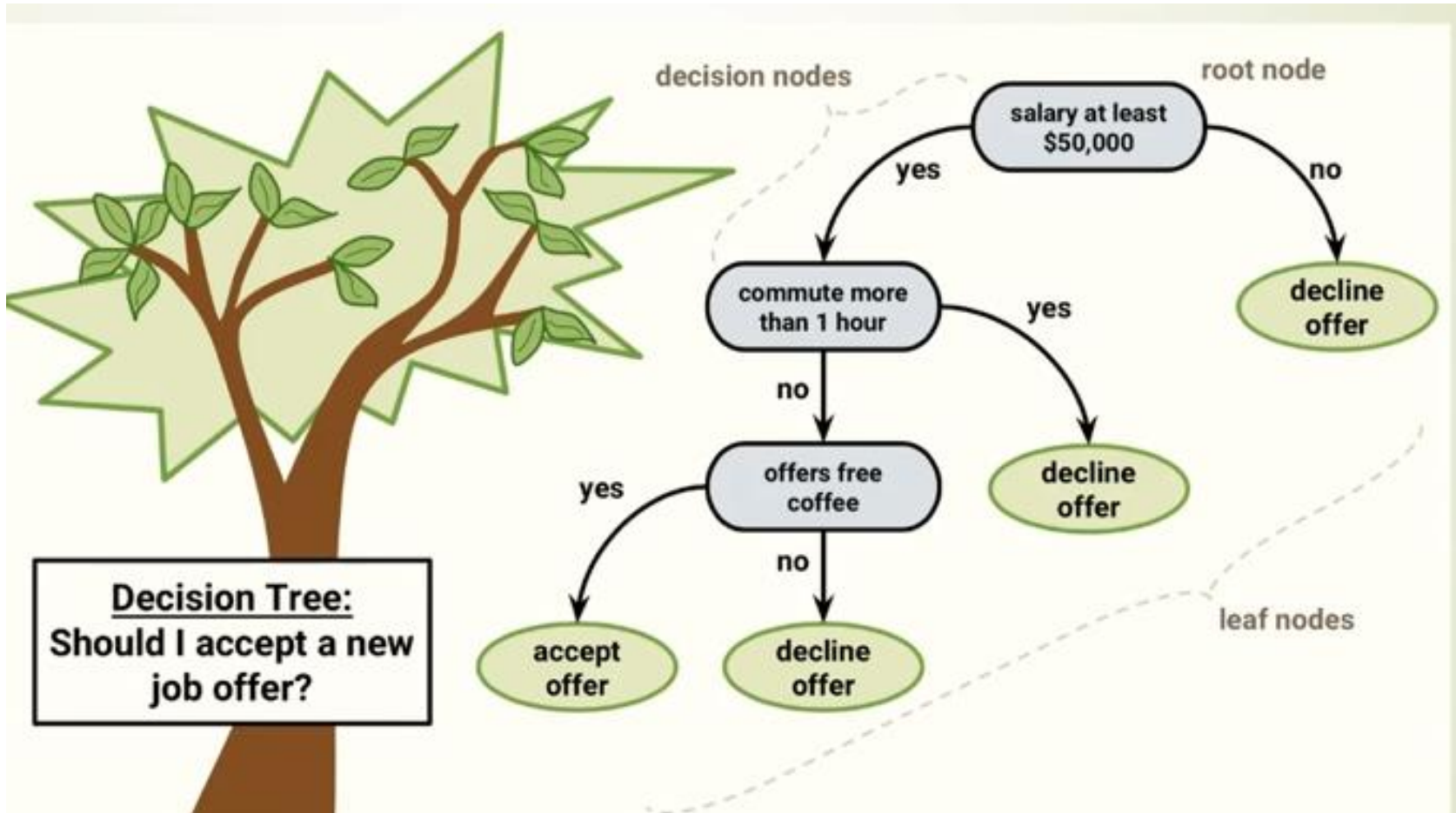
Decision Trees

Decision tree builds classification or regression models in the form of a tree structure. It breaks down a dataset into smaller and smaller subsets while at the same time an associated decision tree is incrementally developed.

Predictors				Target
Outlook	Temp.	Humidity	Windy	Play Golf
Rainy	Hot	High	False	No
Rainy	Hot	High	True	No
Overcast	Hot	High	False	Yes
Sunny	Mild	High	False	Yes
Sunny	Cool	Normal	False	Yes
Sunny	Cool	Normal	True	No
Overcast	Cool	Normal	True	Yes
Rainy	Mild	High	False	No
Rainy	Cool	Normal	False	Yes
Sunny	Mild	Normal	False	Yes
Rainy	Mild	Normal	True	Yes
Overcast	Mild	High	True	Yes
Overcast	Hot	Normal	False	Yes
Sunny	Mild	High	True	No



Another example



Quinlan's ID3 (1986)

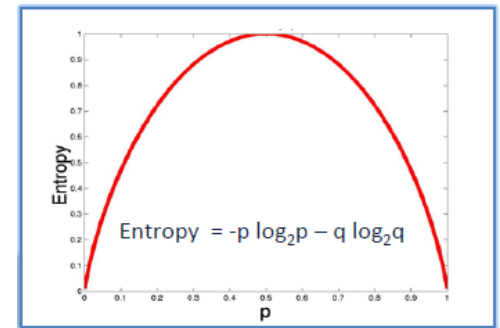
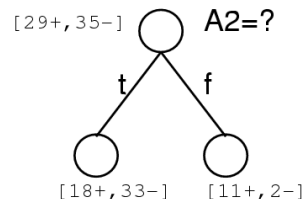
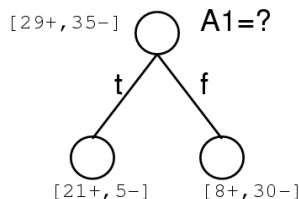
- ID3 algorithm uses entropy to calculate the **homogeneity** of a sample. If the sample is completely homogeneous the entropy is **zero** and if the sample is an equally divided it has entropy of one.

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

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

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

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













$$Entropy = -0.5 \log_2 0.5 - 0.5 \log_2 0.5 = 1$$

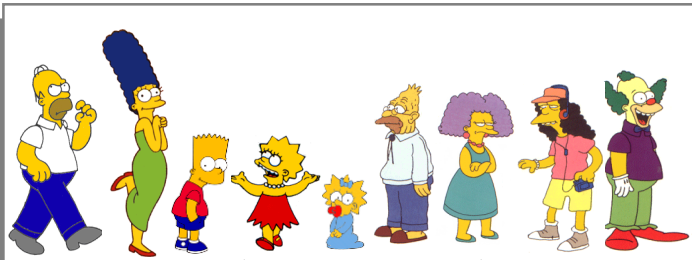
Information entropy is defined as the average amount of information produced/conveyed by a stochastic source of data.



Another Example

Person	Hair Length	Weight	Age	Class
 Homer	0"	250	36	M
 Marge	10"	150	34	F
 Bart	2"	90	10	M
 Lisa	6"	78	8	F
 Maggie	4"	20	1	F
 Abe	1"	170	70	M
 Selma	8"	160	41	F
 Otto	10"	180	38	M
 Krusty	6"	200	45	M
 Comic	8"	290	38	?

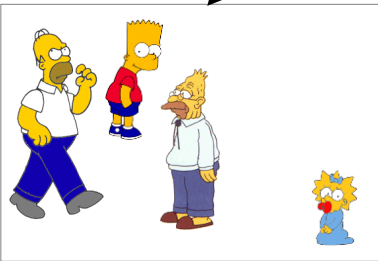
Source: Allan Neymark



$$Entropy(S) = -\frac{p}{p+n} \log_2 \left(\frac{p}{p+n} \right) - \frac{n}{p+n} \log_2 \left(\frac{n}{p+n} \right)$$

$$Entropy(4\mathbf{F}, 5\mathbf{M}) = -(4/9) \log_2(4/9) - (5/9) \log_2(5/9) = \mathbf{0.9911}$$

yes no
Hair Length ≤ 5?



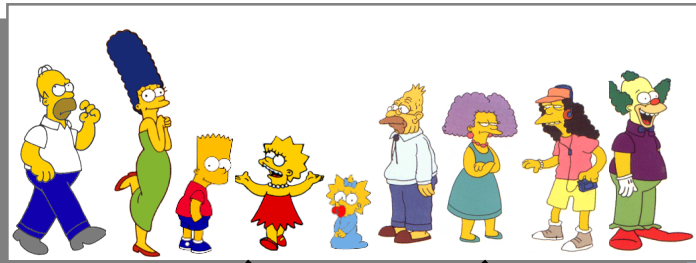
Let us try splitting on *Hair length*

$$Entropy(1\mathbf{F}, 3\mathbf{M}) = -(1/4) \log_2(1/4) - (3/4) \log_2(3/4) = \mathbf{0.8113}$$

$$Entropy(3\mathbf{F}, 2\mathbf{M}) = -(3/5) \log_2(3/5) - (2/5) \log_2(2/5) = \mathbf{0.9710}$$

$$Gain(A) = E(\text{Current set}) - \sum E(\text{all child sets})$$

$$Gain(\text{Hair Length} \leq 5) = \mathbf{0.9911} - (4/9 * \mathbf{0.8113} + 5/9 * \mathbf{0.9710}) = \mathbf{0.0911}$$



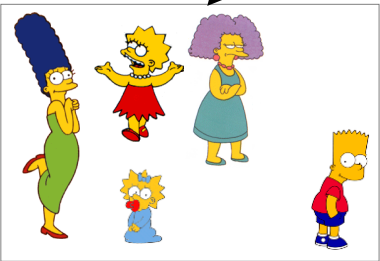
$$Entropy(S) = -\frac{p}{p+n} \log_2 \left(\frac{p}{p+n} \right) - \frac{n}{p+n} \log_2 \left(\frac{n}{p+n} \right)$$

$$Entropy(4\mathbf{F}, 5\mathbf{M}) = -(4/9) \log_2(4/9) - (5/9) \log_2(5/9) = \mathbf{0.9911}$$

yes

no

Weight ≤ 160?



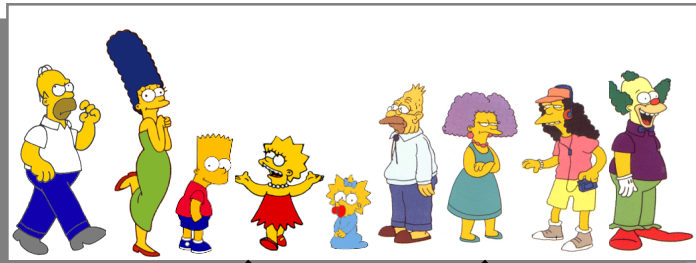
Let us try splitting on
Weight

$$Entropy(4\mathbf{F}, 1\mathbf{M}) = -(4/5) \log_2(4/5) - (1/5) \log_2(1/5) = \mathbf{0.7219}$$

$$Entropy(0\mathbf{F}, 4\mathbf{M}) = -(0/4) \log_2(0/4) - (4/4) \log_2(4/4) = \mathbf{0}$$

$$Gain(A) = E(\text{Current set}) - \sum E(\text{all child sets})$$

$$Gain(\text{Weight} \leq 160) = \mathbf{0.9911} - (5/9 * \mathbf{0.7219} + 4/9 * \mathbf{0}) = \mathbf{0.5900}$$



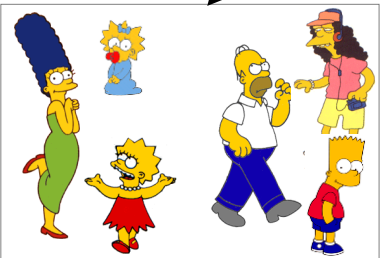
$$Entropy(S) = -\frac{p}{p+n} \log_2 \left(\frac{p}{p+n} \right) - \frac{n}{p+n} \log_2 \left(\frac{n}{p+n} \right)$$

$$Entropy(4\mathbf{F}, 5\mathbf{M}) = -(4/9) \log_2(4/9) - (5/9) \log_2(5/9) = \mathbf{0.9911}$$

yes

no

age <= 40?



Let us try splitting on Age

$$Entropy(3\mathbf{F}, 3\mathbf{M}) = -(3/6) \log_2(3/6) - (3/6) \log_2(3/6) = \mathbf{1}$$

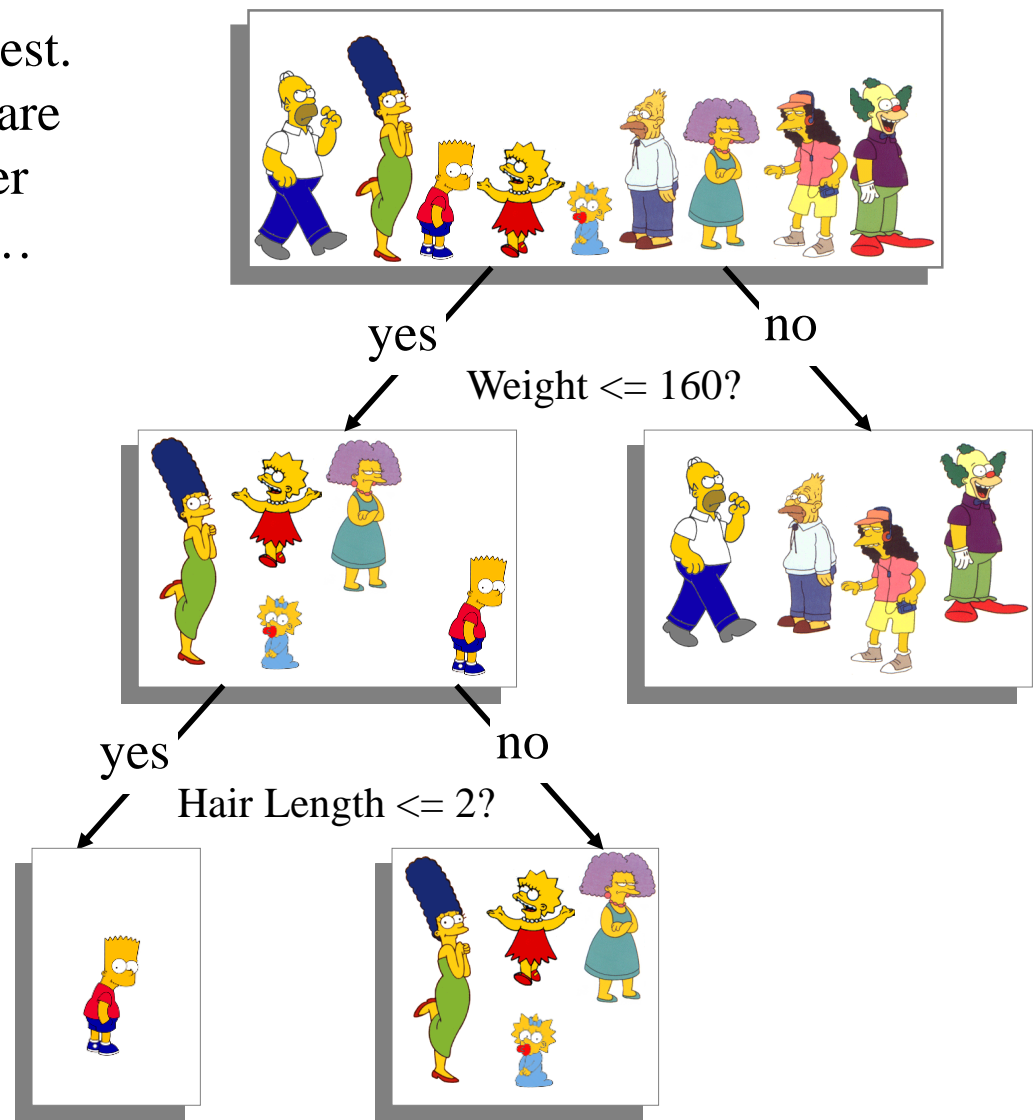
$$Entropy(1\mathbf{F}, 2\mathbf{M}) = -(1/3) \log_2(1/3) - (2/3) \log_2(2/3) = \mathbf{0.9183}$$

$$Gain(A) = E(\text{Current set}) - \sum E(\text{all child sets})$$

$$Gain(\text{Age} \leq 40) = \mathbf{0.9911} - (6/9 * \mathbf{1} + 3/9 * \mathbf{0.9183}) = \mathbf{0.0183}$$

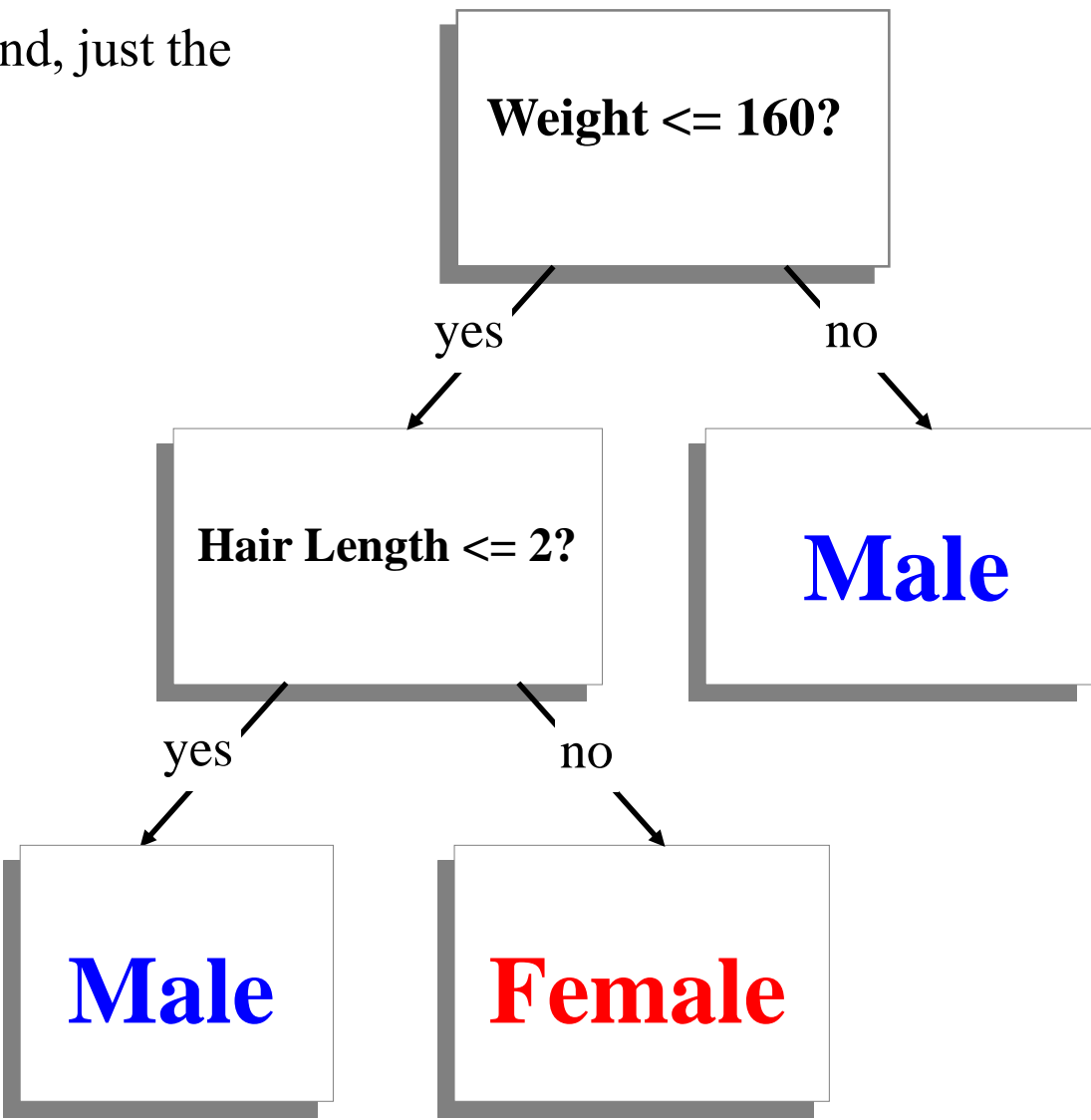
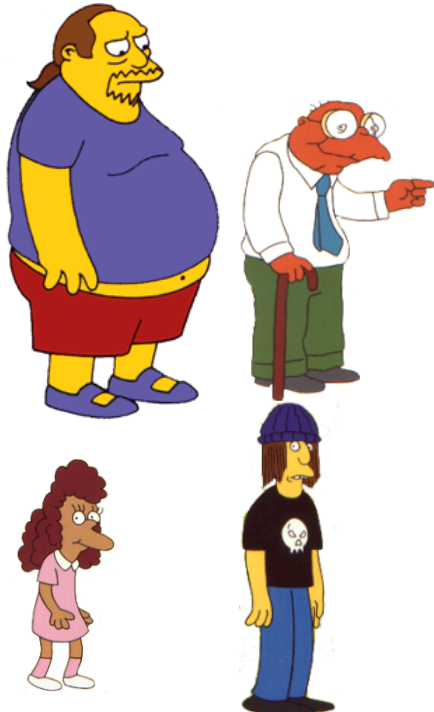
Of the 3 features we had, *Weight* was best.
But while people who weigh over 160 are
perfectly classified (as males), the under
160 people are not perfectly classified...
So we simply recurse!

This time we find that we can split
on *Hair length*, and we are done!

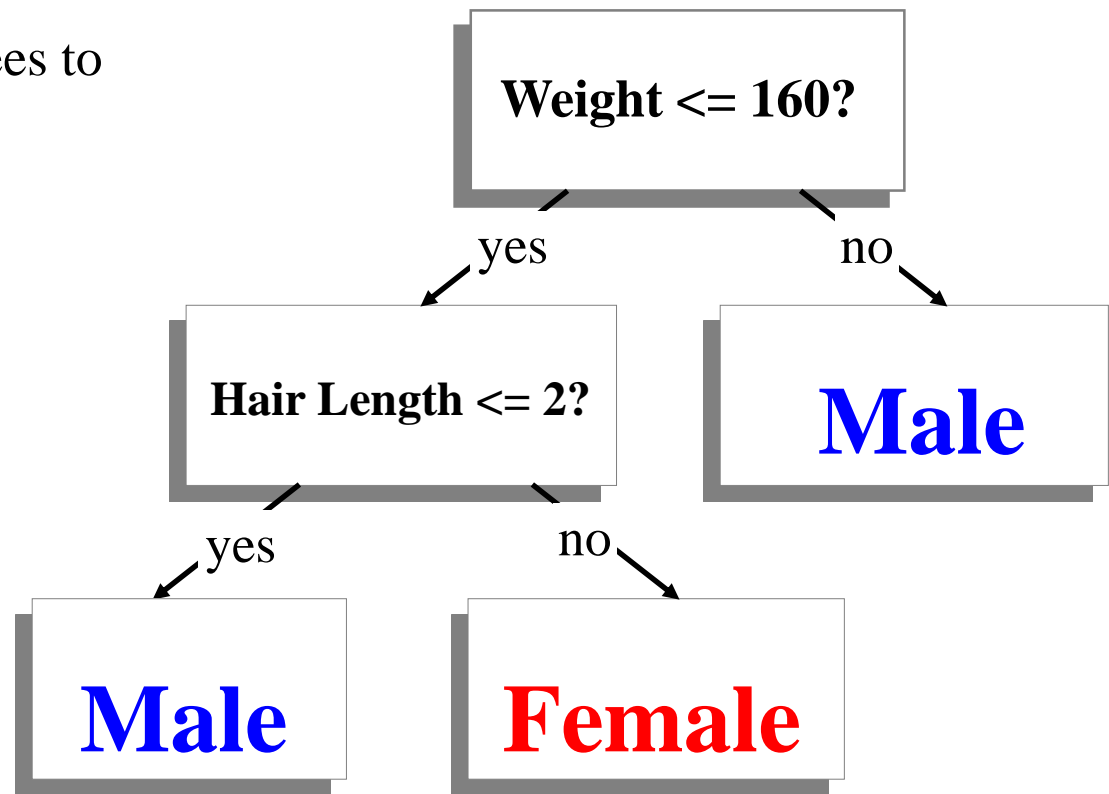


We don't need to keep the data around, just the test conditions.

How would these people be classified?



It is trivial to convert Decision Trees to rules...



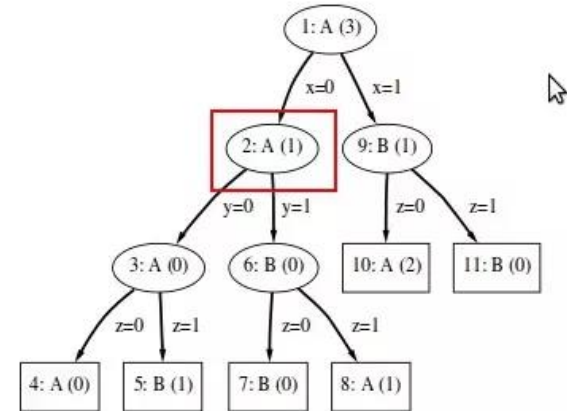
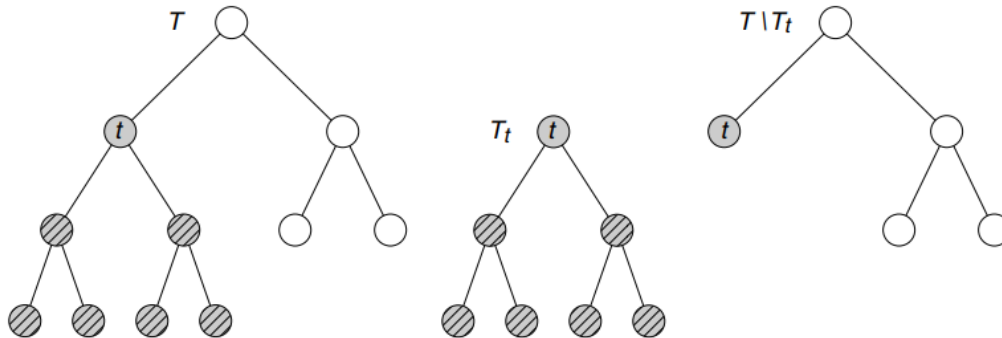
Rules to Classify Males/Females

If *Weight* **greater than** 160, classify as **Male**

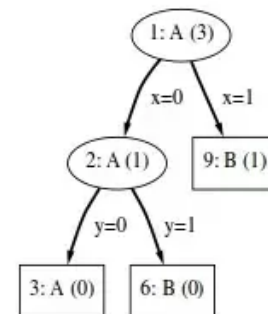
Elseif *Hair Length* **less than or equal** to 2, classify as **Male**

Else classify as **Female**

Pruning DTs



x	y	z	class
0	0	1	A
0	1	1	B
1	1	0	B
1	0	0	B
1	1	1	A



1. Keep aside a part of the dataset for post-pruning of the tree. This is different from the test set and is known as the *pruning set*.

2. For a subtree S of the tree, if replacing S by a leaf does not make more prediction errors on the pruning set than the original tree, replace S by a leaf.

3. Perform step 2 only when no subtree of S possesses the property mentioned in 2.

Explanation Based Learning



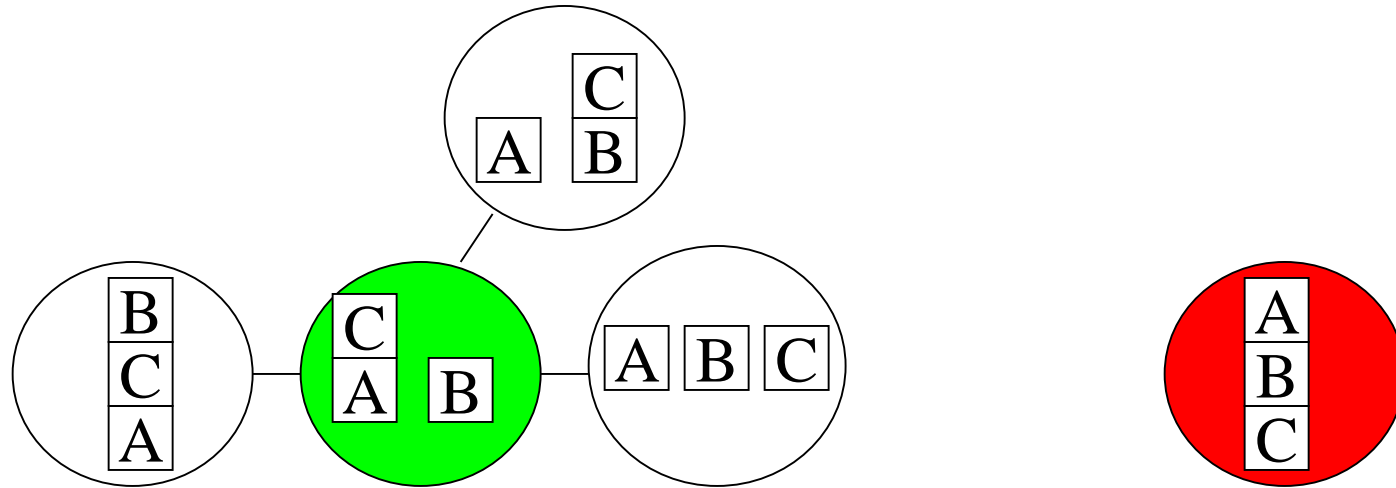
“Hey! Look what Zog do!”

(drawn by Gary Larson)



BITS Pilani
Pilani | Dubai | Goa | Hyderabad
Practice School

Explanation-Based Learning



Don't stack anything above a block, if the block has to be free in the final state

Explanation Based Learning

A target concept

The learning system finds an “**operational**” definition of this concept, expressed in terms of some primitives. The target concept is represented as a predicate.

A training example

This is an **instance** of the target concept. It takes the form of a set of simple facts, not all of them necessarily relevant to the theory.

A domain theory

This is a **set of rules**, usually in predicate logic, that can explain how the training example fits the target concept.

Operationality criteria

These are the predicates (**features**) that should appear in an effective definition of the target concept.

Continued...

A classic example: a theory and an instance of a **cup**. A cup is a container for liquids that can be easily lifted. It has some typical parts, such as a handle and a bowl, the actual containers, must be concave. Because a cup can be lifted, it should be light. And so on...

The target concept is **cup(X)**.

The domain theory has five rules.

$\text{liftable}(X) \wedge \text{holds_liquid}(X) \Rightarrow \text{cup}(X)$

$\text{part}(Z, W) \wedge \text{concave}(W) \wedge \text{points_up}(W) \Rightarrow$
 $\text{holds_liquid}(Z)$

$\text{light}(X) \wedge \text{part}(X, \text{handle}) \Rightarrow \text{liftable}(X)$

$\text{small}(A) \Rightarrow \text{light}(A)$

$\text{made_of}(A, \text{feathers}) \Rightarrow \text{light}(A)$

Continued...

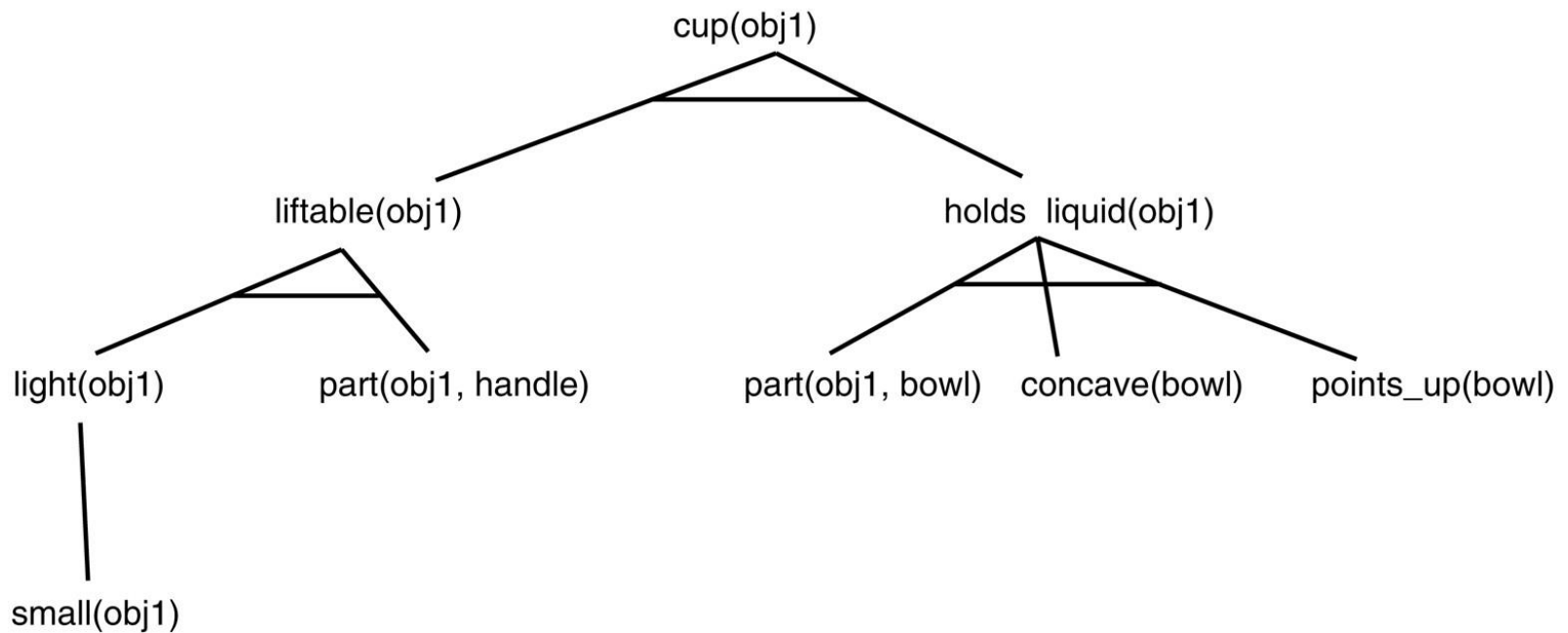
The training example lists nine facts (some of them are not relevant).

cup(obj1)	small(obj1)
part(obj1, handle)	owns(bob, obj1)
part(obj1, bottom)	part(obj1, bowl)
points_up(bowl)	concave(bowl)
color(obj1, red)	

Operationality criteria require a definition in terms of structural properties of objects (**part, points_up, small, concave**).

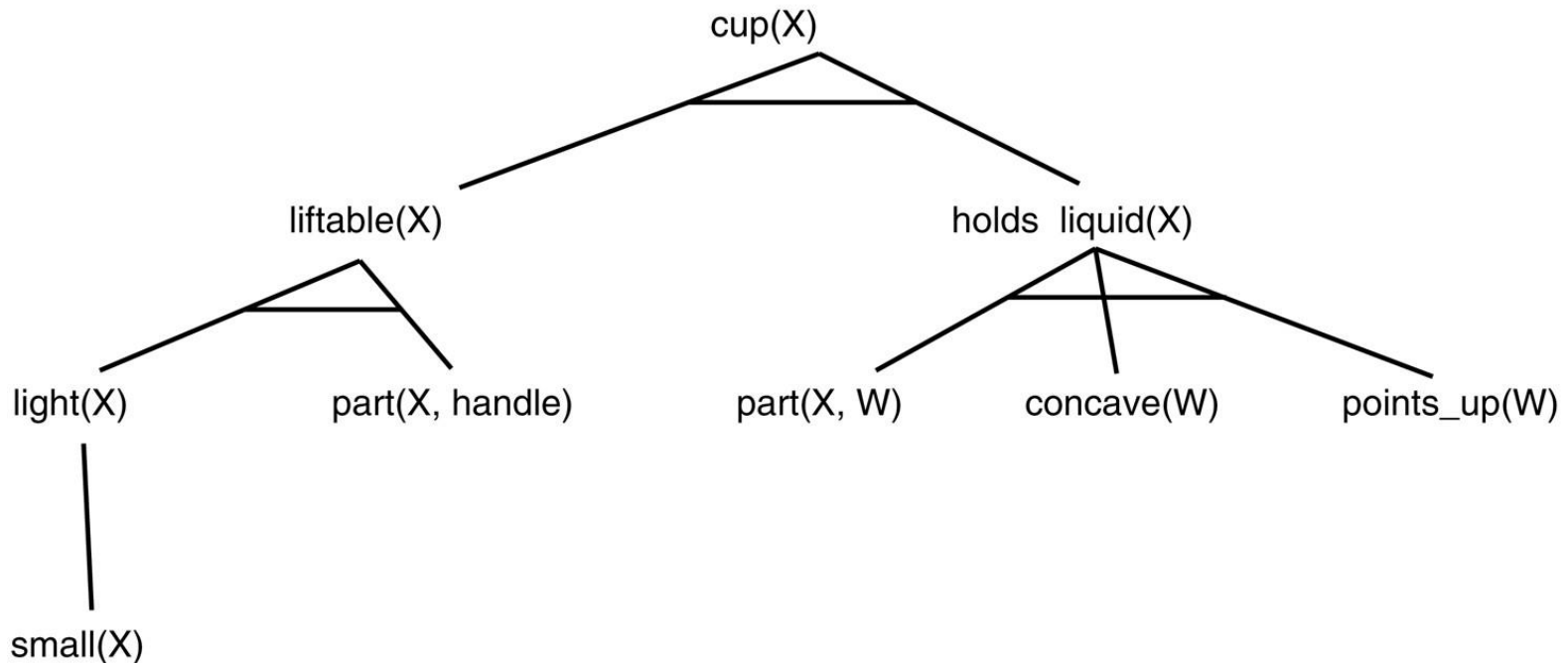
Continued...

Step 1: prove the target concept using the training example



Continued...

Step 2: **generalize the proof**. Constants from the domain theory, for example **handle**, are not generalized.



Continued...

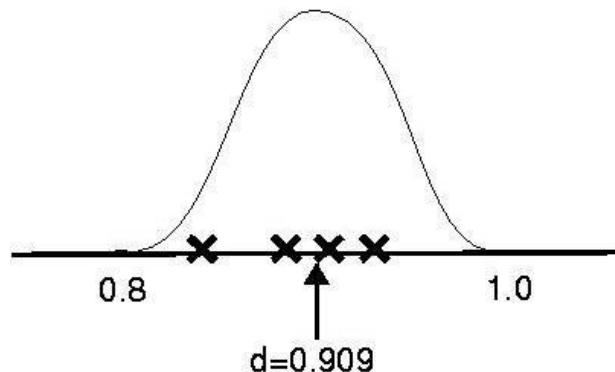
Step 3: Take the definition “off the tree”, only the root and the leaves.

In our example, we get this rule:


















































$$\begin{aligned} &\text{small}(X) \wedge \\ &\text{part}(X, \text{handle}) \wedge \\ &\text{part}(X, W) \wedge \\ &\text{concave}(W) \wedge \\ &\text{points_up}(W) \Rightarrow \text{cup}(X) \end{aligned}$$

Ensemble learning: Motivation

- When designing a learning agent, we generally make some choices:
 - *parameters, training data, representation, etc...*
- This implies some sort of variance in performance
- Why not keep all learners and average?
- ...



Example: Weather Forecast

Reality							
1							
2							
3							
4							
5							
Combine							

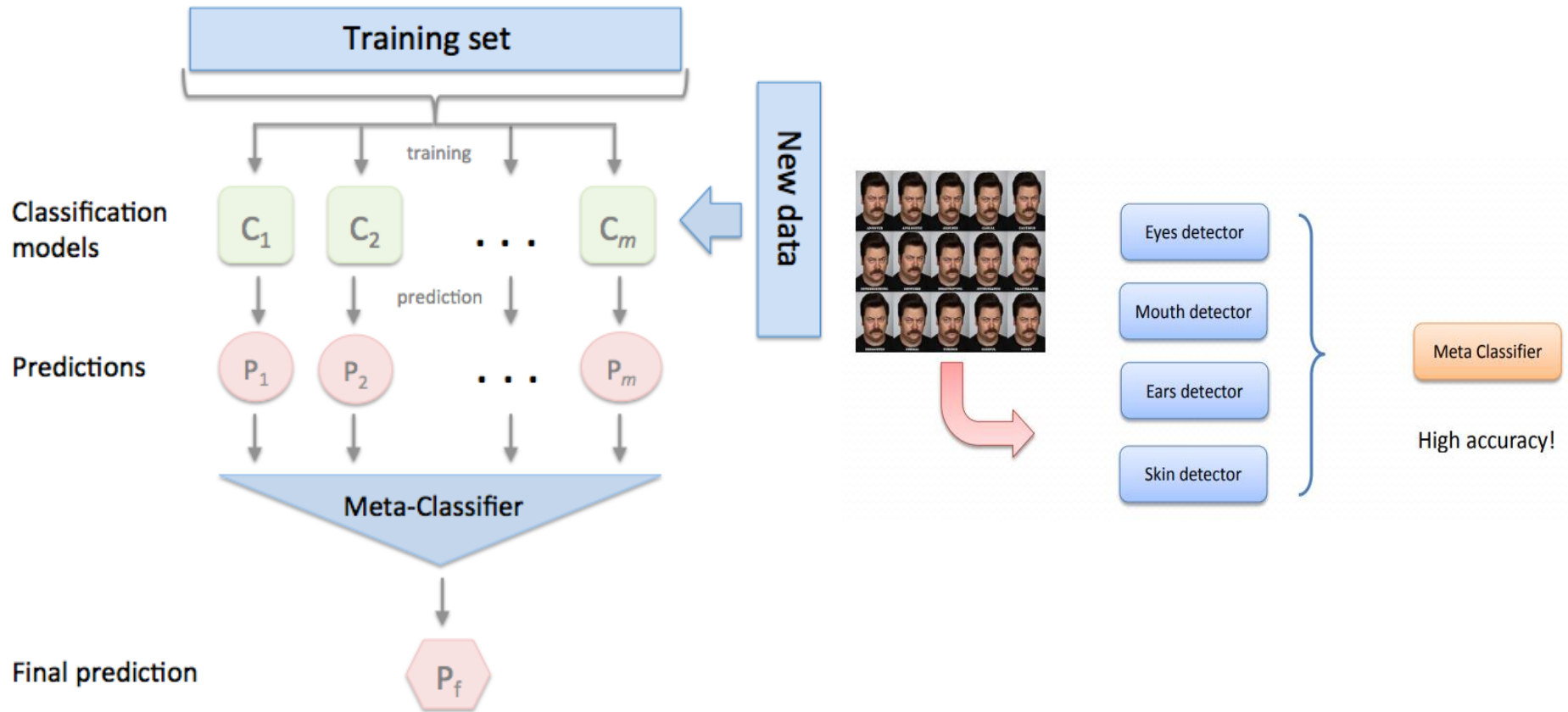
Source: Carla P. Gomes

Different types of ensemble learning

- Same algorithm, different versions of data set, e.g.
 - Bagging: resample training data
 - Boosting: Reweight training data
 - Decorate: Add additional artificial training data
 - Random Subspace (random forest): random subsets of features

In **WEKA**, these are called **meta-learners**, they take a learning algorithm as an argument (**base learner**) and create a new learning algorithm.

Ensemble learning: Stacking



Img source: <https://rasbt.github.io>

Bagging

□ Bootstrap

- ▣ Sampling with replacement
- ▣ Contains around 63.2% original records in each sample

□ Bootstrap Aggregation

- ▣ Train a classifier on each bootstrap sample
- ▣ Use **majority voting** to determine the class label of ensemble classifier

Continued...

Original Data:

x	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
y	1	1	1	-1	-1	-1	-1	1	1	1

Bootstrap samples and classifiers:

x	0.1	0.2	0.2	0.3	0.4	0.4	0.5	0.6	0.9	0.9
y	1	1	1	1	-1	-1	-1	-1	1	1

x	0.1	0.2	0.3	0.4	0.5	0.5	0.9	1	1	1
y	1	1	1	-1	-1	-1	1	1	1	1

x	0.1	0.2	0.3	0.4	0.4	0.5	0.7	0.7	0.8	0.9
y	1	1	1	-1	-1	-1	-1	-1	1	1

x	0.1	0.2	0.5	0.5	0.5	0.7	0.7	0.8	0.9	1
y	1	1	-1	-1	-1	-1	-1	1	1	1

Combine predictions by majority voting

Boosting

- **Principles**

- Boost a set of weak learners to a strong learner
- Make records currently misclassified more important

- **Example**

- Record 4 is hard to classify
- It's weight is increased, therefore it is more likely to be chosen again in subsequent rounds

Original Data	1	2	3	4	5	6	7	8	9	10
Boosting (Round 1)	7	3	2	8	7	9	4	10	6	3
Boosting (Round 2)	5	4	9	4	2	5	1	7	4	2
Boosting (Round 3)	4	4	8	10	4	5	4	6	3	4

Inductive Logic Programming (ILP)

- In ML, attribute-value representations are more usual (decision trees, rules, ...)
 - Why predicate logic?
 - More **expressive** than attribute-value representations
 - Enables flexible use of **background knowledge** (knowledge known to learner prior to learning)
1. The chair in the living room is red. The chair in the dining room is red. The chair in the bedroom is red.
2. Amit leaves for school at 7:00 a.m. Amit is always on time. Amit assumes, then, that

Deductive Vs Inductive Reasoning

T

\cup

B

\rightarrow

E (deduce)

parent(X,Y) :- mother(X,Y).
parent(X,Y) :- father(X,Y).

mother(mary,vinni).
mother(mary,andre).
father(carrey,vinni).
father(carrey,andre).

parent(mary,vinni).
parent(mary,andre).
parent(carrey,vinni).
parent(carrey,andre).

E

\cup

B

\rightarrow

T (induce)

parent(mary,vinni).
parent(mary,andre).
parent(carrey,vinni).
parent(carrey,andre).

mother(mary,vinni).
mother(mary,andre).
father(carrey,vinni).
father(carrey,andre).

parent(X,Y) :- mother(X,Y).
parent(X,Y) :- father(X,Y).

Learning definition of daughter

% Background knowledge

parent(pam, bob).

parent(tom, liz).

...

female(liz).

male(tom).

...

% Positive examples

ex(has_a_daughter(tom)). % Tom has a daughter

ex(has_a_daughter(bob)).

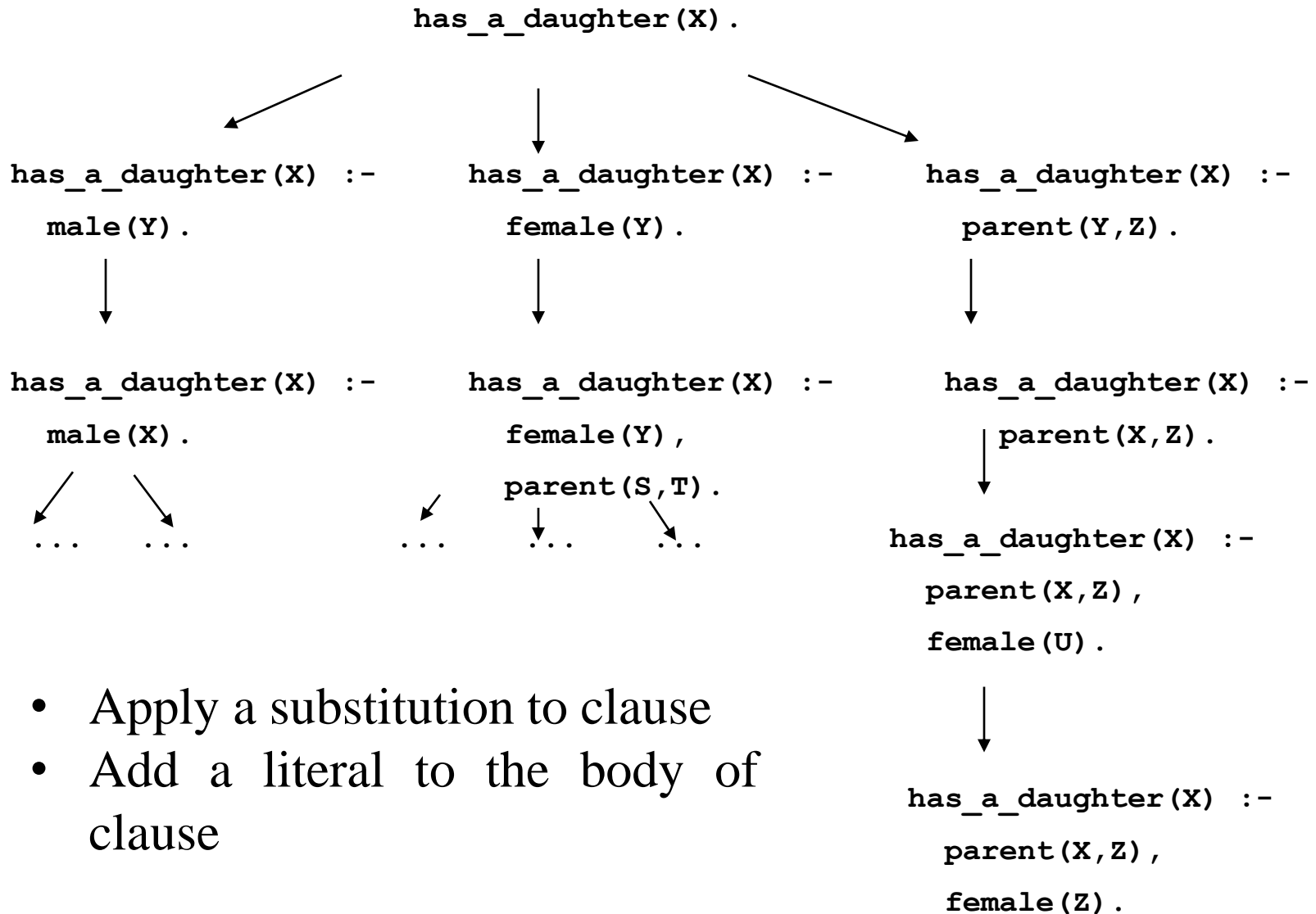
...

% Negative examples

nex(has_a_daughter(pam)). % Pam doesn't have a daughter

...

Top-down induction



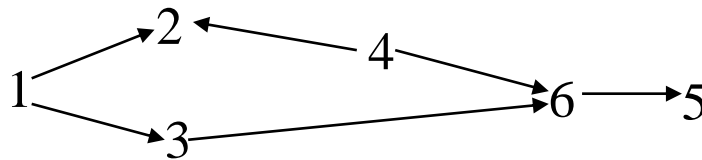
- Apply a substitution to clause
- Add a literal to the body of clause

First-Order Inductive Logic

- Example: Consider learning a definition for the target predicate **path** for finding a path in a directed acyclic graph.

$\text{path}(X,Y) \text{ :- edge}(X,Y).$

$\text{path}(X,Y) \text{ :- edge}(X,Z), \text{path}(Z,Y).$



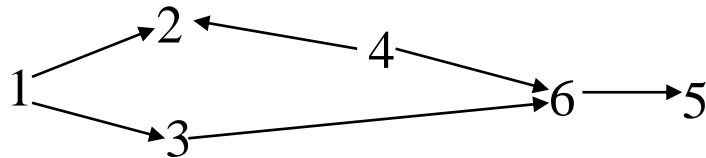
Training data:

$\text{edge: } \{ \langle 1, 2 \rangle, \langle 1, 3 \rangle, \langle 3, 6 \rangle, \langle 4, 2 \rangle, \langle 4, 6 \rangle, \langle 6, 5 \rangle \}$

$\text{path: } \{ \langle 1, 2 \rangle, \langle 1, 3 \rangle, \langle 1, 6 \rangle, \langle 1, 5 \rangle, \langle 3, 6 \rangle, \langle 3, 5 \rangle, \langle 4, 2 \rangle, \langle 4, 6 \rangle, \langle 4, 5 \rangle, \langle 6, 5 \rangle \}$

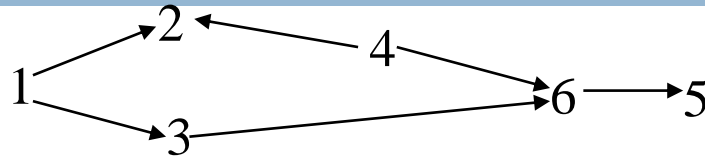
Negative Training Data

- Negative examples of target predicate can be provided directly, or generated indirectly by making a *closed world assumption*.



Negative path tuples:

Sample FOIL Induction



Pos: {<1, 2>, <1, 3>, <1, 6>, <1, 5>, <3, 6>, <3, 5>, <4, 2>, <4, 6>, <4, 5>, <6, 5>}

Neg: {<1, 1>, <1, 4>, <2, 1>, <2, 2>, <2, 3>, <2, 4>, <2, 5>, <2, 6>, <3, 1>, <3, 2>, <3, 3>, <3, 4>, <4, 1>, <4, 3>, <4, 4>, <5, 1>, <5, 2>, <5, 3>, <5, 4>, <5, 5>, <5, 6>, <6, 1>, <6, 2>, <6, 3>, <6, 4>, <6, 6>}

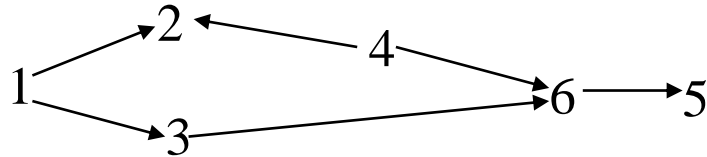
Start with clause:

path(X, Y) :- .

Possible literals to add:

edge(X, X) , edge(Y, Y) , edge(X, Y) , edge(Y, X) , edge(X, Z) ,
edge(Y, Z) , edge(Z, X) , edge(Z, Y) , path(X, X) , path(Y, Y) ,
path(X, Y) , path(Y, X) , path(X, Z) , path(Y, Z) , path(Z, X) ,
path(Z, Y) , plus negations of all of these.

Sample FOIL Induction



Pos: {<1, 2>, <1, 3>, <1, 6>, <1, 5>, <3, 6>, <3, 5>, <4, 2>, <4, 6>, <4, 5>, <6, 5>}

Neg: {<1, 1>, <1, 4>, <2, 1>, <2, 2>, <2, 3>, <2, 4>, <2, 5>, <2, 6>, <3, 1>, <3, 2>, <3, 3>, <3, 4>, <4, 1>, <4, 3>, <4, 4>, <5, 1>, <5, 2>, <5, 3>, <5, 4>, <5, 5>, <5, 6>, <6, 1>, <6, 2>, <6, 3>, <6, 4>, <6, 6>}

Test:

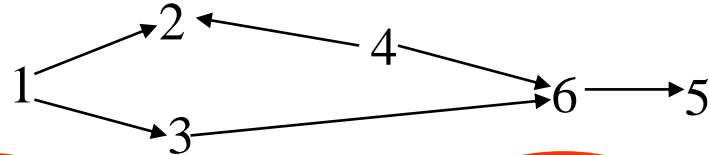
`path(X, Y) :- edge(X, X) .`

Covers 0 positive examples

Covers 6 negative examples

Not a good literal.

Sample FOIL Induction



Pos: { $\langle 1, 2 \rangle$, $\langle 1, 3 \rangle$, $\langle 1, 6 \rangle$, $\langle 1, 5 \rangle$, $\langle 3, 6 \rangle$, $\langle 3, 5 \rangle$,
 $\langle 4, 2 \rangle$, $\langle 4, 6 \rangle$, $\langle 4, 5 \rangle$, $\langle 6, 5 \rangle$ }

Neg: { $\langle 1, 1 \rangle$, $\langle 1, 4 \rangle$, $\langle 2, 1 \rangle$, $\langle 2, 2 \rangle$, $\langle 2, 3 \rangle$, $\langle 2, 4 \rangle$, $\langle 2, 5 \rangle$, $\langle 2, 6 \rangle$,
 $\langle 3, 1 \rangle$, $\langle 3, 2 \rangle$, $\langle 3, 3 \rangle$, $\langle 3, 4 \rangle$, $\langle 4, 1 \rangle$, $\langle 4, 3 \rangle$, $\langle 4, 4 \rangle$, $\langle 5, 1 \rangle$,
 $\langle 5, 2 \rangle$, $\langle 5, 3 \rangle$, $\langle 5, 4 \rangle$, $\langle 5, 5 \rangle$, $\langle 5, 6 \rangle$, $\langle 6, 1 \rangle$, $\langle 6, 2 \rangle$, $\langle 6, 3 \rangle$,
 $\langle 6, 4 \rangle$, $\langle 6, 6 \rangle$ }

Test:

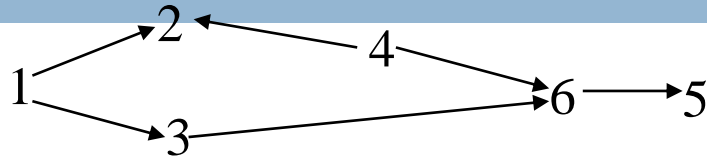
`path(X,Y) :- edge(X,Y) .`

Covers 6 positive examples

Covers 0 negative examples

Chosen as best literal. Result is base clause.

Sample FOIL Induction



Pos: {<1, 6>, <1, 5>, <3, 5>, <4, 5>}

Neg: {<1, 1>, <1, 4>, <2, 1>, <2, 2>, <2, 3>, <2, 4>, <2, 5>, <2, 6>, <3, 1>, <3, 2>, <3, 3>, <3, 4>, <4, 1>, <4, 3>, <4, 4>, <5, 1>, <5, 2>, <5, 3>, <5, 4>, <5, 5>, <5, 6>, <6, 1>, <6, 2>, <6, 3>, <6, 4>, <6, 6>}

Test:

`path(X, Y) :- edge(X, Y) .`

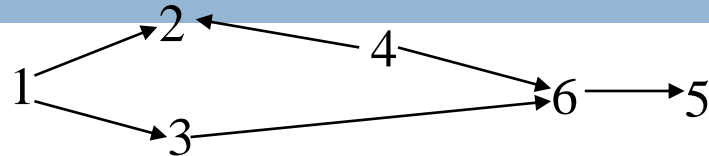
Covers 6 positive examples

Covers 0 negative examples

Chosen as best literal. Result is base clause.

Remove covered positive tuples.

Sample FOIL Induction



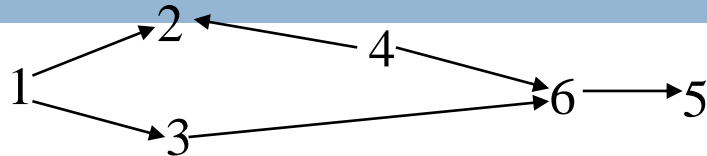
Pos: {<1, 6>, <1, 5>, <3, 5>, <4, 5>}

Neg: {<1, 1>, <1, 4>, <2, 1>, <2, 2>, <2, 3>, <2, 4>, <2, 5>, <2, 6>, <3, 1>, <3, 2>, <3, 3>, <3, 4>, <4, 1>, <4, 3>, <4, 4>, <5, 1>, <5, 2>, <5, 3>, <5, 4>, <5, 5>, <5, 6>, <6, 1>, <6, 2>, <6, 3>, <6, 4>, <6, 6>}

Start new clause

path(X, Y) :- .

Sample FOIL Induction



Pos: {<1, 6>, <1, 5>, <3, 5>, <4, 5>}

Neg: {<1, 1>, <1, 4>, <2, 1>, <2, 2>, <2, 3>, <2, 4>, <2, 5>, <2, 6>, <3, 1>, <3, 2>, <3, 3>, <3, 4>, <4, 1>, <4, 3>, <4, 4>, <5, 1>, <5, 2>, <5, 3>, <5, 4>, <5, 5>, <5, 6>, <6, 1>, <6, 2>, <6, 3>, <6, 4>, <6, 6>}

Test:

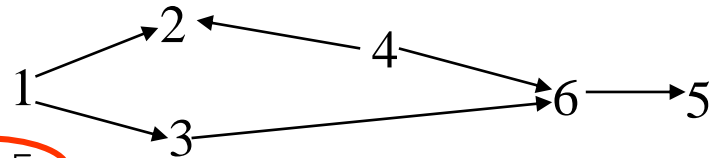
`path(X, Y) :- edge(X, Y) .`

Covers 0 positive examples

Covers 0 negative examples

Not a good literal.

Sample FOIL Induction



Pos: { $\langle 1, 6 \rangle$, $\langle 1, 5 \rangle$, $\langle 3, 5 \rangle$,
 $\langle 4, 5 \rangle$ }

Neg: { $\langle 1, 1 \rangle$, $\langle 1, 4 \rangle$, $\langle 2, 1 \rangle$, $\langle 2, 2 \rangle$, $\langle 2, 3 \rangle$, $\langle 2, 4 \rangle$, $\langle 2, 5 \rangle$, $\langle 2, 6 \rangle$,
 $\langle 3, 1 \rangle$, $\langle 3, 2 \rangle$, $\langle 3, 3 \rangle$, $\langle 3, 4 \rangle$, $\langle 4, 1 \rangle$, $\langle 4, 3 \rangle$, $\langle 4, 4 \rangle$, $\langle 5, 1 \rangle$,
 $\langle 5, 2 \rangle$, $\langle 5, 3 \rangle$, $\langle 5, 4 \rangle$, $\langle 5, 5 \rangle$, $\langle 5, 6 \rangle$, $\langle 6, 1 \rangle$, $\langle 6, 2 \rangle$, $\langle 6, 3 \rangle$,
 $\langle 6, 4 \rangle$, $\langle 6, 6 \rangle$ }

Test:

`path(X, Y) :- edge(X, Z) .`

Covers all 4 positive examples

Covers 14 of 26 negative examples

Eventually chosen as best possible literal



Next: Intro to NLP