Artificial Intelligence

CPSC 481

Machine Learning

Part B

Overview

- Non-symbolic machine learning
  - Artificial neural networks
- Unsupervised Learning
  - Clustering algorithms
- Other machine learning approaches
  - Genetic learning
  - Reinforcement learning
  - Deep learning
  - Online learning
- Issues related to machine learning

Another Categorization of Machine Learning Approaches

- Symbolic learning vs. non-symbolic learning
  - ***Symbolic learning*** builds symbolic concepts or models from experience by making improvement using symbolic computation of data.
  - ***Non-symbolic learning*** uses optimization (using numerical computation) or adaptation (evolutionary approach)

- **Some symbolic learning approaches**
  - Concepts learning through generalization and specialization
- **Some non-symbolic learning approaches**
  - Genetic learning is based on adaptation to environment using fitness function.
  - Artificial neural network is based on weights optimization or gradient descent search.

Genetic Learning using GP

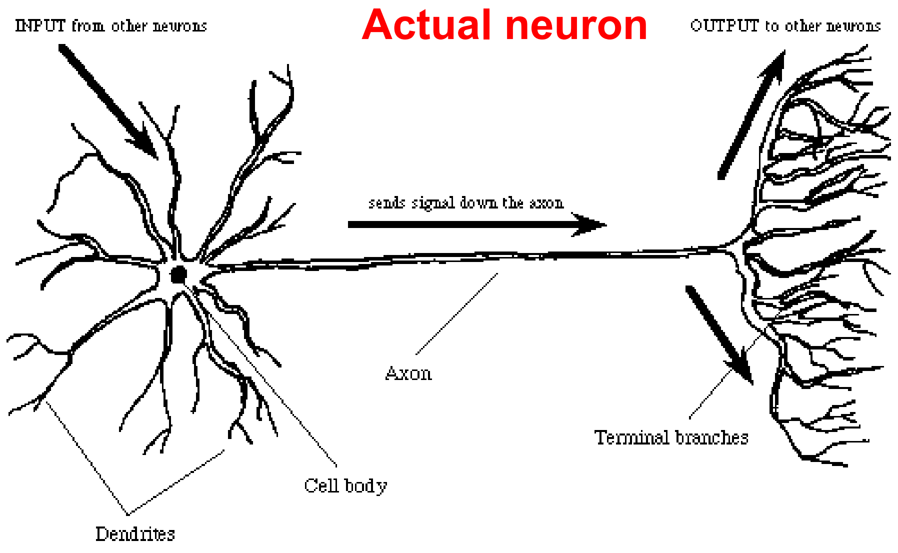
- **Learning a function using Genetic Programming** (for regression analysis)
  - Learning of the function that map a data item to a real valued prediction variable.
- **Parameters for genetic supervised learning**
  - **Objective**: evolve functions fitting the values of the cases in training data set

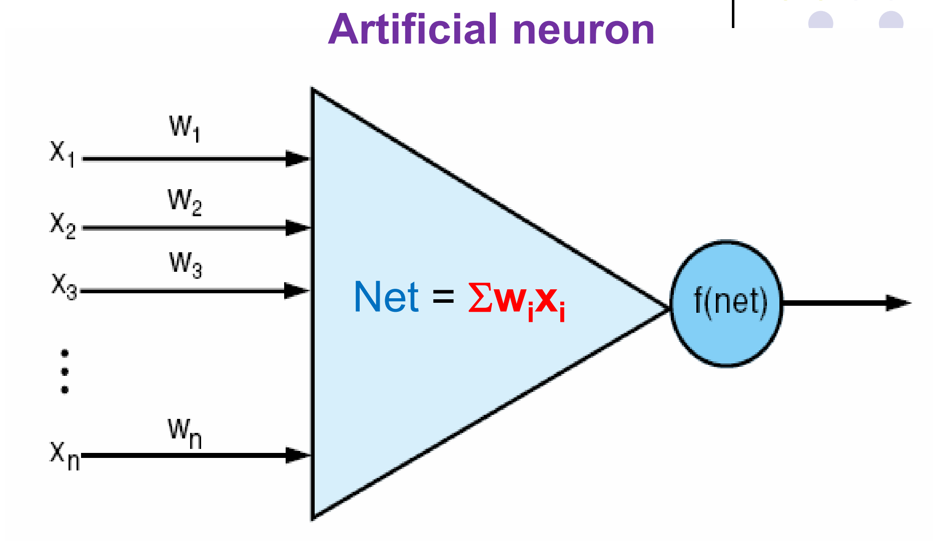
Parameters:	**Values**
Terminal set	variable x (input) and integer constants between –5 and +5
Function set	{+, –, *, %}
Fitness function	Error function
Population size	600
Crossover probability	90%
Mutation probability	5%
Selection	tournament selection, size 4
Tolerant error	0.001
Termination criterion	none
Maximum number of generations	100
Maximum depth of tree after crossover	200
Maximum mutant depth	4

Training data set

recordID	Input	Output
1	0.000	0.000
2	0.100	0.005
3	0.200	0.020
4	0.300	0.045
5	0.400	0.080
6	0.500	0.125
7	0.600	0.180
8	0.700	0.245
9	0.800	0.320
10	0.900	0.405

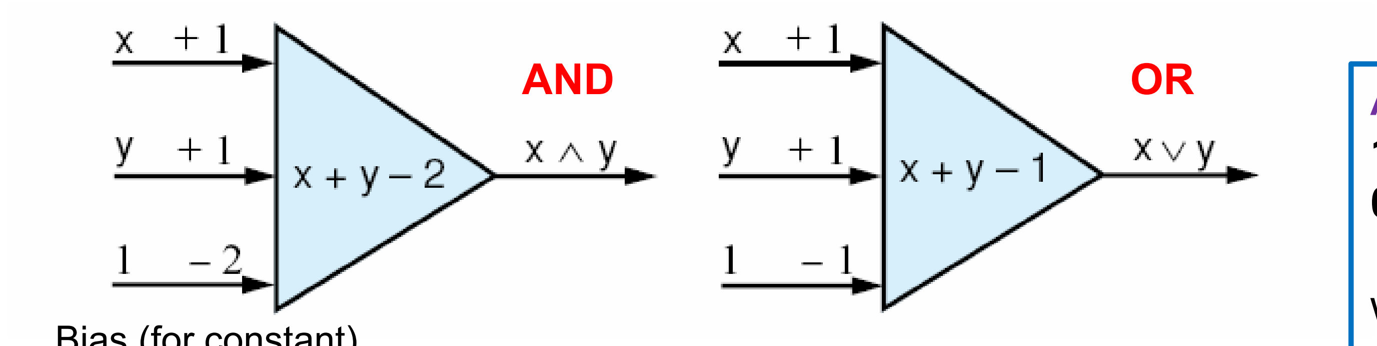
Artificial Neuron





- **Artificial neuron** consists of
  - **Input signals** in a vector, **X**: data from environment or other neurons
  - **Weights** in a vector, **W**: connection strengths
  - **Activation** function, **f** that determines the neuron's output value based **net** calculated by $\Sigma w_i x_i$ and a thresh hold value, θ.
    - Why weighted sum?
- **Artificial neuron** is the **basic unit** of **Artificial Neural Network** (**ANN**).

McCulloch-Pitts Neuron Model to Calculate the Logic Functions, AND and OR, 1943



Bias (for constant)

Activation function f(net):
1 if **net** >= θ (=0)
0 otherwise

where net= $\Sigma w_i x_i$

Input		net	AND
x	y	$x + y - 2$	Output
1	1	0	1
1	0	−1	0
0	1	−1	0
0	0	−2	0

net	OR
X+Y-1	Output
1	1
0	1
0	1
-1	0

What is the key idea used to solve these problems? Changing weights or activation function?

Can we use McCulloch-Pitts Neuron Model to Solve Learning Problems?

Class

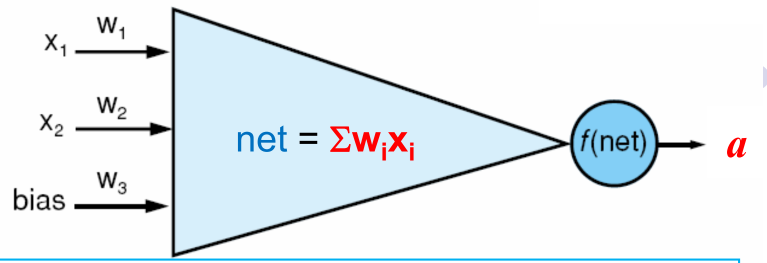
x_1	x_2	Output
1.0	1.0	1
9.4	6.4	−1
2.5	2.1	1
8.0	7.7	−1
0.5	2.2	1
7.9	8.4	−1
7.0	7.0	−1
2.8	0.8	1
1.2	3.0	1
7.8	6.1	−1

Training data set

- Can we use McCulloch-Pitts Neuron Model to solve other problems?
- If Yes, which part of the model do we need to change?
- How can we solve a learning (or classification) problem based on McCulloch-Pitts Neuron Model?

Perceptron Learning

(Rosenblatt, 1962)



Assign random values as initial weights, W

For each example **vector**, X^m from a training data set, **feed** it to the neuron as input

1. **Produce output**, a using the activation function, f(net)

   f(net) returns 1 if net >= θ where $net = \Sigma w_i x_i$

   else -1 (if net < θ)

2. **Adjust weights** for **next example**, X^{m+1} *as follows:*

   If $t = a$ use the current weights, W^m (or do nothing)

   where t: the desired output in example X^m in vector

   Else (error) adjust weights

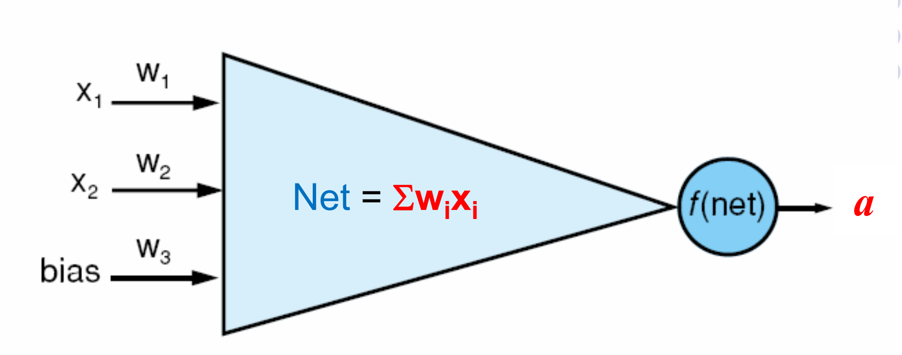
   - Compute $\Delta W^m = l(t - a)X^m$, where l is a learning rate (0.0~1.0) to control the speed of convergence
   - Weights for next example X^{m+1}, $W^{m+1} = W^m + \Delta W^m$

Repeat this process for all examples in the training set *until converged* (possibly reusing the training data).

Example: **Perceptron Learning**

x_1	x_2	**Class** Output
1.0	1.0	1
9.4	6.4	−1
2.5	2.1	1
8.0	7.7	−1
0.5	2.2	1
7.9	8.4	−1
7.0	7.0	−1
2.8	0.8	1
1.2	3.0	1
7.8	6.1	−1

Training data set



$f(net) = f(w_1{*}x_1+w_2{*}x_2+w_3{*}\mathbf{1})$, bias=1, $\theta = -0.6$

$\mathbf{W} = [0.75, 0.5, -0.6]$ (randomly generated),

$l = 0.2$

$\mathbf{f(net)^1}=f(0.75{*}1+0.5{*}1-0.6{*}1)=f(0.65) = 1$,

$(t - a) = (1-1) = 0$, so no Δw needed

$\mathbf{f(net)^2}=f(0.75{*}9.4+0.5{*}6.4-0.6{*}1)=f(9.65) = 1$,

$(t - a)=(-1-1)=-2$,we need Δw

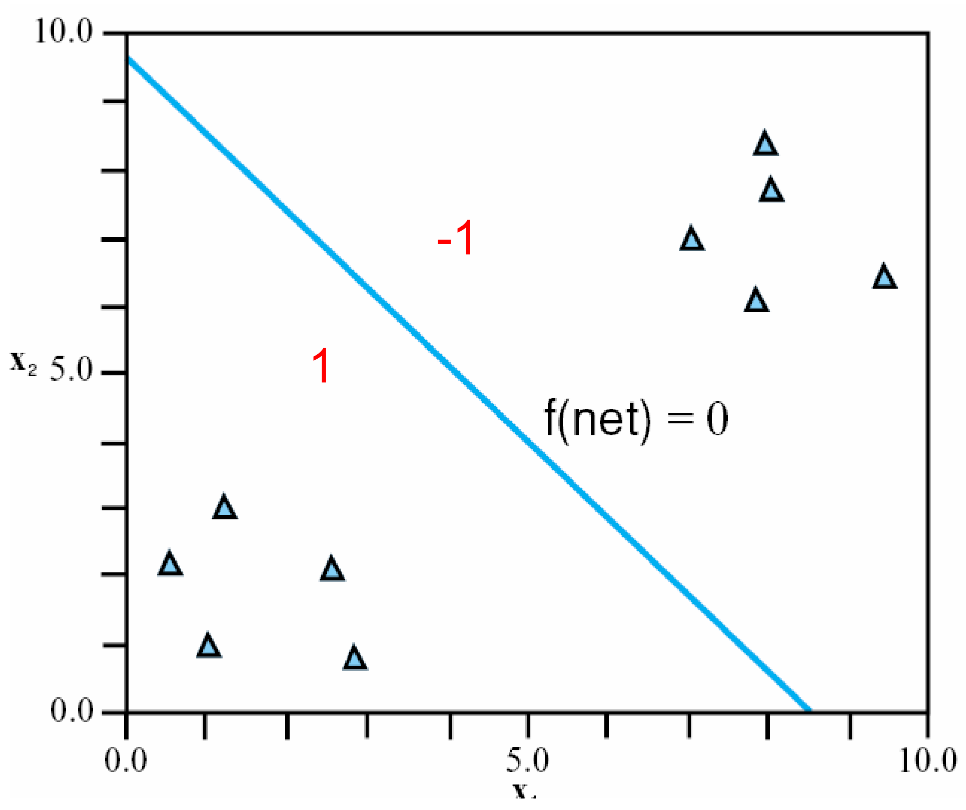
$$\mathbf{W^3} = W^2 + \mathbf{0.2}(-1-1)X^2 = \begin{bmatrix}0.75\\0.50\\-0.60\end{bmatrix} -0.4\begin{bmatrix}9.4\\6.4\\1.0\end{bmatrix} = \begin{bmatrix}-3.01\\-2.06\\-1.00\end{bmatrix}$$

$$\mathbf{f(net)^3} = f(-3.01{*}2.5 - 2.06{*}2.1 - 1.0{*}1)$$
$$= f(-12.84) = -1, \text{ we need } \Delta w$$

$$\mathbf{W^4} = W^3 + 0.2(1-(-1))X^3 = \begin{bmatrix}-3.01\\-2.06\\-1.00\end{bmatrix} + 0.4\begin{bmatrix}2.5\\2.1\\1.0\end{bmatrix} = \begin{bmatrix}-2.01\\-1.22\\-0.60\end{bmatrix}$$

*Continue this process until converged.
(This is an example of gradient descent search.)

The Perceptron that Linearly Separates the Training Data Set



A **linearly separable function**

*The weight vector converged after repeated training on the data set, about 500 iterations

W^{500} = [-1.3, -1.1, 10.9]

f(net) = $w_1x_1 + w_2x_2 + w_3$, where two discriminant functions $\mathbf{g_i}$ (1) and $\mathbf{g_j}$ (-1) are the same (border), $g_i = g_j$

f(net) = $\mathbf{-1.3x_1 + -1.1x_2 + 10.9} = 0$

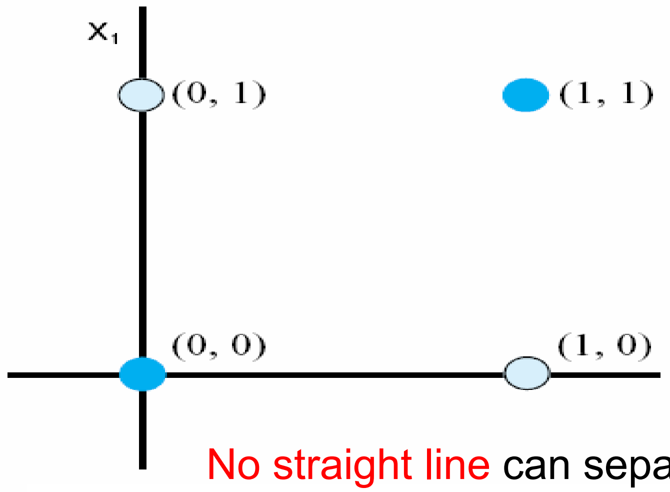
f(net) is the line separating the two classes that can be used for classification.

+How is the perceptron learning different from symbolic learning approaches such as Winston's learning program and decision tree induction?

Linearly Not Separable Case

Truth table for **exclusive-OR**

X_1	X_2	Output
1	1	0
1	0	1
0	1	1
0	0	0



No straight line can separate these data points.

From the data set, **following equations** must be solved:

w1*1 + w2*1 < θ , where θ is a threshold.

w1*1 + 0 > θ

0 + w2*1 > θ

0 + 0 < θ or θ must be positive

(This series of equations with w1, w2, and θ has no solution)

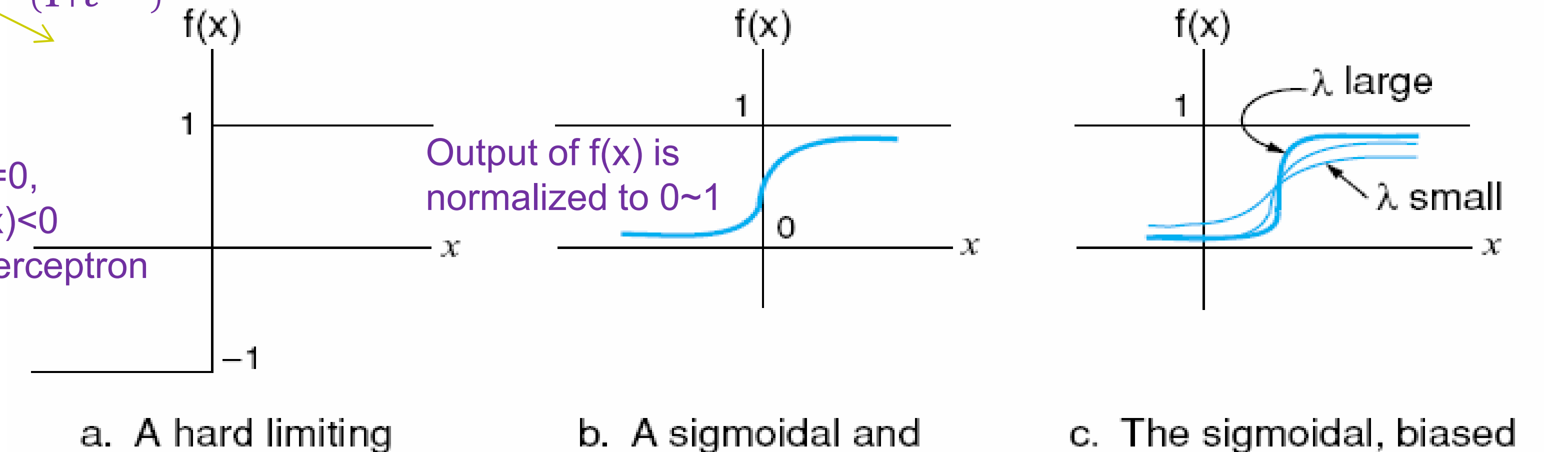
*Minsky and Papert (1969) proved that a perceptron that solves **exclusive-or** is impossible.

Sigmoidal Function for Activation Function

Sigmoidal function:

f(x) = $\frac{1}{(1+e^{-\lambda x})}$, λ is a squashing parameter to fine-tune the curve by adjusting the slope.

1 if f(x) >=0,
-1 else f(x)<0
used in perceptron



a. A hard limiting and bipolar linear threshold.

b. A sigmoidal and unipolar threshold.

c. The sigmoidal, biased and squashed. As λ gets larger the sigmoid approximates a linear threshold.

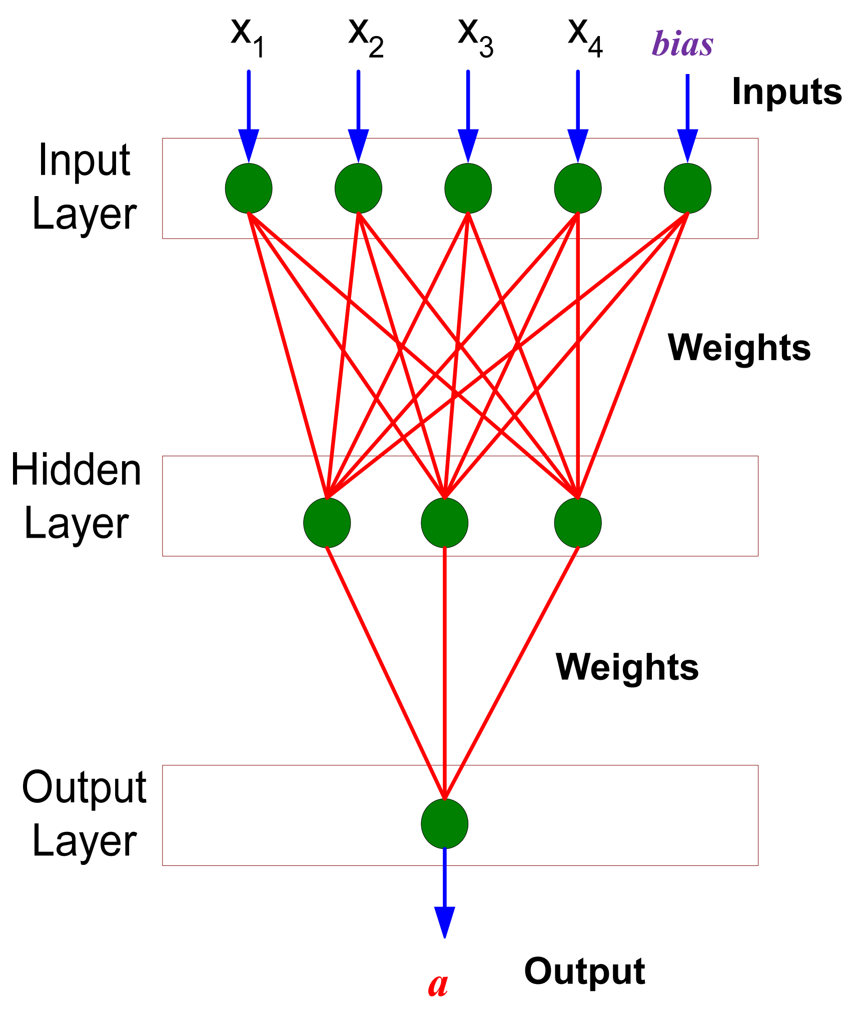
Useful features of sigmoidal function:

- S-shaped non-linear and continuous function in the range of 0 ~1
- Differentiable (has a derivative) at each point and derivative calculation is easy.
- The value of derivative is greatest where the sigmoidal function is changing most rapidly.

Sigmoidal activation function: **$f(net)=1/(1 + e^{-\lambda net})$**, where **net** = $\Sigma w_i x_i$

The value of **f(net)** is continuous 0.0~1.0, not binary.

Muti-layer Neural Networks with Backpropagation

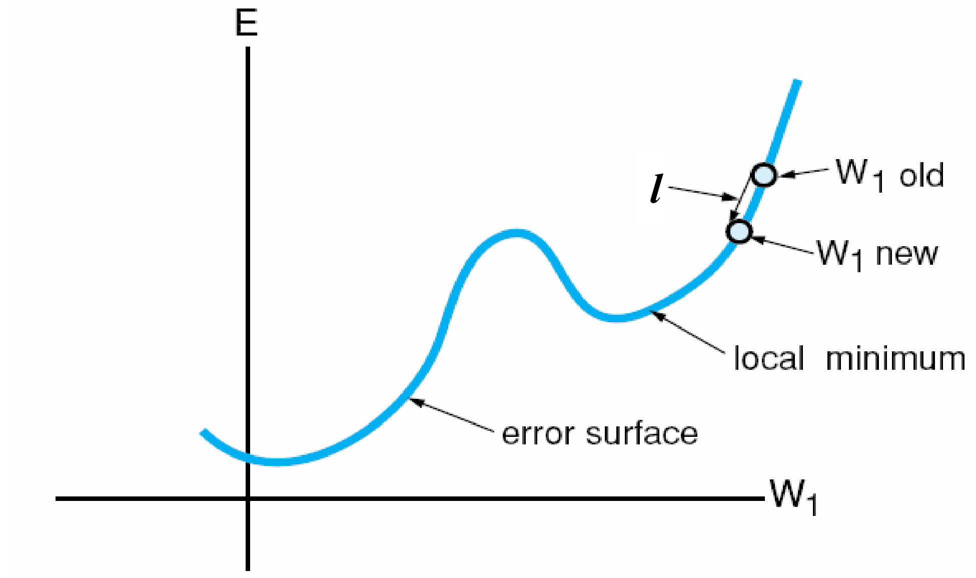


- **Goal**:
  - Given training data, **find weights** that **fit** outputs of neural net to desired output or classification.
- **Mechanism**:
  - For each training example, propagate the inputs Forward for activation.
  - Backpropagate the error to previous layer and Adjust the weights slowly in direction that reduces error.
  - Repeat this process until converged.
- **Proven to converge**

  on the nearest local minimum but
  - May not find the best weights
  - Sensitive to initial weight setting
  - Long training time
- **Many applications**:

  solved many difficult real-world problems.

Weight Adjustment with Sigmoidal Function



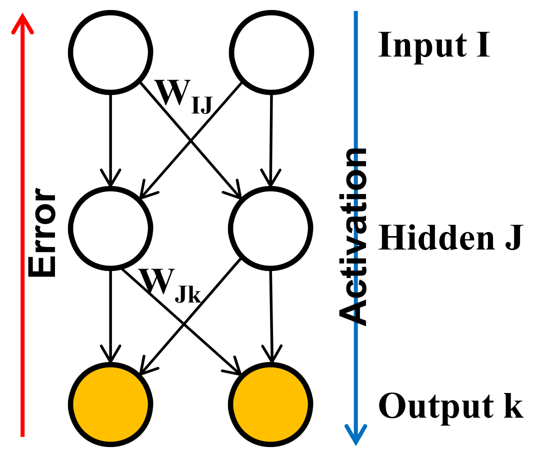
*The error surface represents **cumulative error** over a data set as a function of weights.

*l determines the size of learning step.
***Gradient** is a measure of slope, as a function of direction from a point on a surface.
*Learning algorithm uses gradient descent method to find the **direction** on the error surface which most rapidly reduces the error (ending at a local minimum).

How do we determine + or – of ΔW?

- **Weight adjustment by the Delta rule** (Rumelhart et al. 1986)
  - Weight adjustment on the jth node from the ith input: $\Delta W_j = l(t_j - a_j)f'(net_j)x_i$ where l is the learning rate; t_j is the desired output; a_j is the output of the node; $f'(net_j)$ is the derivative of f(net); x_i is an input (or output from the previous node)
  - Why f'(net)? How do we calculate f'(net)?
  - **Derive the delta rule!**

Computing Δw for Output Nodes



Symbols

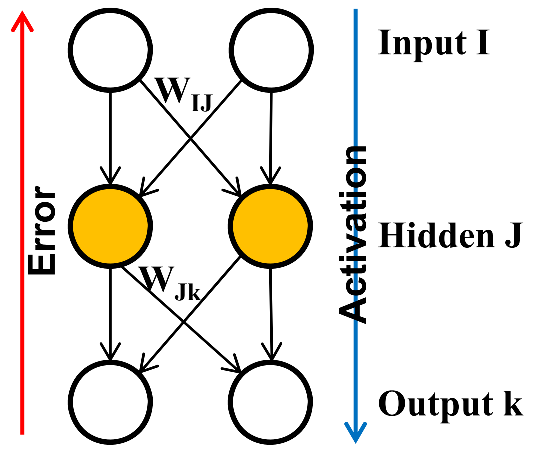
- I: input layer
- J: hidden layer
- K: output layer
- w_{ij}: weight, input → hidden
- w_{jk}: weight, hidden → output
- a: activation value, output
- t: target value
- net: the net input
- l: learning rate

Gradient descent on error

- $E = (1/2)\sum_k (t_k - a_k)^2$ the total error in a network
- $\Delta w \propto -\frac{\partial E}{\partial W}$ to adjust the weights to reduce the overall error
- $\Delta wjk \propto -\frac{\partial E}{\partial w_{jk}}$ to adjust the weight at the output layer
- $\Delta w_{jk} = -l \frac{\partial E}{\partial a_k} * \frac{\partial a_k}{\partial net_k} * \frac{\partial net_k}{\partial w_{jk}}$ since error is not directly a function of a weight

- Derivative of the error with respect to the activation
  - $\frac{\partial E}{\partial a_k} = \frac{\partial (1/2)(tk - a_k)^2}{\partial a_k} = -(t_k - a_k)$
- Derivative of the activation with respect to the net input
  - $\frac{\partial a_k}{\partial net_k} = f'(net) = \partial(1 + e^{-net_k})^{-1} / \partial net_k = a_k (1 - a_k)$
- Derivative of the net input with respect to a weight
  - $\frac{\partial net_k}{\partial w_{jk}} = \frac{\partial (w_{jk} a_j)}{\partial w_{jk}} = a_j$
- Weight change rule (delta rule) for a hidden to output weight
  - $\Delta wjk = -l \frac{\partial E}{\partial w_{jk}} = -l \frac{\partial E}{\partial a_k} * \frac{\partial a_k}{\partial net_k} * \frac{\partial net_k}{\partial w_{jk}} = l(t_k - a_k) a_k (1 - a_k) a_j$
  - If we let $\delta_k = (t_k - a_k) a_k (1 - ak)$, then $\Delta wjk = l \delta_k a_j$

Computing Δw for Hidden Nodes



Subscript notations

- I: input layer
- J: hidden layer
- K: output layer
- w_{ij}: weight, input → hidden
- w_{jk}: weight, hidden → output
- a: an activation value
- t: a target value
- net: the net input
- *l*: learning rate

Gradient descent on error

- $\Delta wij \propto -\frac{\partial E}{\partial w_{ij}}$ to adjust the weight at the output layer
- $\Delta w_{ij} = -l\frac{\partial E}{\partial a_j} * \frac{\partial a_j}{\partial net_j} * \frac{\partial net_j}{\partial w_{ij}}$

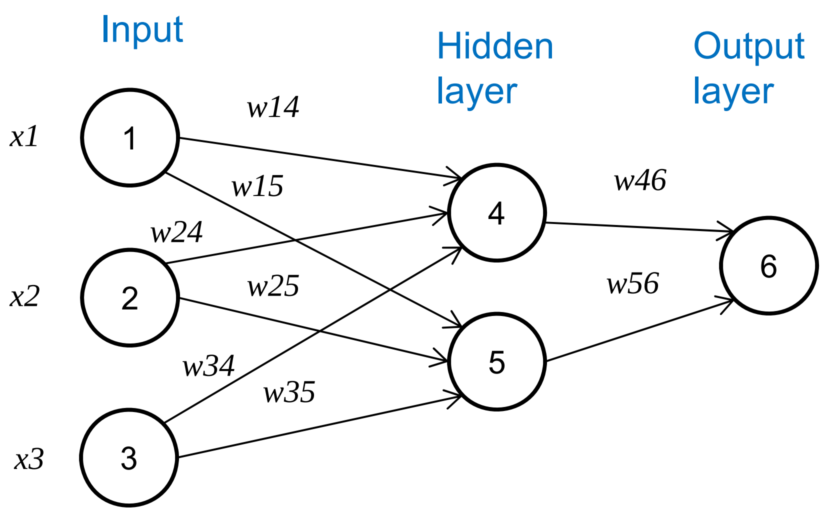
- Derivative of the output at J layer with respect to the input I
  - $\frac{\partial a_j}{\partial net_j} * \frac{\partial net_j}{\partial w_{ji}} = a_j(1-a_j) * \frac{\partial net_j}{\partial w_{ji}} = a_j(1-a_j) * \frac{\partial(wijai)}{\partial w_{ij}} = a_j(1-a_j)a_i$
- Derivative of the error at K layer with respect to the activation of J layer
  - $\frac{\partial E}{\partial a_j} = [\sum \frac{\partial E}{\partial a_k} * \frac{\partial a_k}{\partial net_k} * \frac{\partial net_k}{\partial a_j}]$ since error on output is propagated from hidden layer (all nodes at J)
  - $\frac{\partial E}{\partial a_k} = -(t_k - a_k)$
  - $\frac{\partial a_k}{\partial net_k} = a_k(1-a_k)$
  - $\frac{\partial net_k}{\partial a_j} = \frac{\partial(wjkaj)}{\partial a_j} = w_{jk}$
- Weight change rule (delta rule) for a hidden to output weight
  - Let $\delta_k = \sum_k (t_k - a_k) a_k (1-a_k) w_{jk}$
  - Let $\delta_j = \delta_k a_j (1-a_j)$
  - $\Delta wij = -l\frac{\partial E}{\partial a_j} * \frac{\partial a_j}{\partial net_j} * \frac{\partial net_j}{\partial w_{ij}} = l\delta_j a_i$

Backpropagation Algorithm

- **Input**: Training data set; learning rate, l; multi-layer network
- **Output**: A neural network trained to classify the data
- **Algorithm**:

```
1)  Initialize all weights and biases in network
2)  while terminating condition is not satisfied
3)    for each sample X in training data set
4)      for each hidden or output layer unit j
5)        $I_j = \sum_i w_{ij} a_i + \theta_j$; //compute the input of unit j, net_j with respect to the previous layer $i$.
6)        $a_j = \frac{1}{1+e^{-\lambda I_j}}$; //compute the output of each unit j; assume λ = 1.
7)      for each unit j in the output layer
8)        $E_j = a_j(1 - a_j)(t_j - a_j)$; //compute the error to backpropagate the errors
9)      for each unit j in the hidden layer
10)       $E_j = a_j(1 - a_j)\sum_k E_k w_{jk}$ ; //compute the error with respect to the next layer, k
11)     for each weight $w_{ij}$ in network
12)       $\Delta w_{ij} = l * Ej * ai$; //weight increment; $l$ is the learning rate.
13)       $w_{ij} = w_{ij} + \Delta wij$; //weight update
14)     for each bias $\theta_j$ in network
15)       $\Delta\theta_j = lEj_j$ ; //bias increment
16)       $\theta_j = \theta_j + \Delta\theta_j$; //bias update
```

Example: Backpropagation Calculation



Initial input and class

$x1$	$x2$	$x3$	class
1	0	1	1

biases

θ_4	θ_5	θ_6
-0.4	0.2	0.1

Initial weights, l=0.9

$w14$	$w15$	$w24$	$w25$	$w34$	$w35$	$w46$	$w56$
0.2	-0.3	0.4	0.1	-0.5	0.2	-0.3	-0.2

Inputs and outputs for node 1, 2, 3 or I1, I2, I3 are the same as initial inputs x1, x2, and x3, respectively. The input and output calculations for nodes 4, 5, 6 are given below:

Unit j	**Input,** $I_j = \sum_i w_{ij}a_i + \theta_j$	**Output,** $a_j = \frac{1}{1+e^{-\lambda I_j}}$, λ=1
4	I4 = 0.2 + 0 – 0.5 – 0.4 = –0.7	$1/(1+e^{0.7}) = 0.332$
5	I5 = –0.3 + 0 + 0.2 + 0.2 = 0.1	$1/(1+e^{-0.1}) = 0.525$
6	I6 = (–0.3)(0.332) – (0.2)(0.525) + 0.1 = –0.105	$1/(1+e^{0.105}) = 0.474$

Calculation of the error at each node

Unit j	$E_j = a_j(1 - a_j)(t_j - a_j)$ for output layer $E_j = a_j(1 - a_j)\sum_k E_k w_{jk}$ for hidden layer
6 (output)	(0.474)(1–0.474)(1–0.474) = 0.1311
5 (hidden)	(0.525)(1–0.525)(0.1311)(–0.2) = –0.0065
4 (hidden)	(0.332)(1–0.332)(0.1311)(–0.3) = –0.0087

Calculations for weight and bias update

Weight/bais	Updated weights by $\Delta wij = lEja_i$; $w_{ij} = wij + \Delta wij$; $\Delta\theta_j = lEj_j$; $\theta_j = \theta_j + \Delta\theta_j$; l=0.9
w46	–0.3+(0.9)(0.1311)(0.332) = –0.261
w56	–0.2+(0.9)(0.1311)(0.525) = –0.138
w14	0.2+(0.9)(–0.0087)(1) = 0.192
w15	–0.3+(0.9)(–0.0065)(1) = –0.306
w24	0.4+(0.9)(–0.0087)(0) = 0.4
w25	0.1+(0.9)(–0.0065)(0) = 0.1
w34	–0.5+(0.9)(–0.0087)(1) = –0.508
w35	0.2+(0.9)(–0.0065)(1) = 0.194
θ_6	0.1+(0.9)(0.1311) = 0.218
θ_5	0.2+(0.9)(–0.0065) = 0.194
θ_4	–0.4+(0.9)(–0.0087) = –0.408

Exercise: Show the process of solving Exclusive-OR problem.

Various Types of Neural Networks

- **Different neural networks can be built using different options:**
  - Network topology
    - Many layers instead of 2 => +100 layers (**deep learning**)
  - Learning algorithm
  - Encoding scheme

- **Neural network can be used for both**
  - Supervised learning (classification)
  - Unsupervised learning (clustering)

Unsupervised Learning

- Unsupervised learning eliminates the teacher and requires that the learners learn concepts on their own
    - **AM** was the first unsupervised discovery program to learn mathematical concepts, e.g., concepts of set theory, natural number theory, prime numbers, etc. but was not very successful, mostly due to representation (symbolic based) and inability to learn.

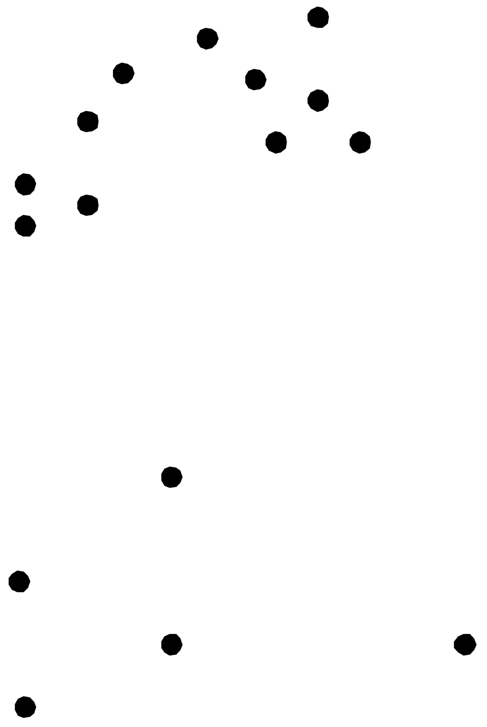
- Methods needed for unsupervised learning
    - **Discovery** such as scientific discovery process
        - hypothesis → generality → conclusion
    - **Pattern finding**
        - **Pattern recognition**
        - **Clustering** (class finding by similarity)

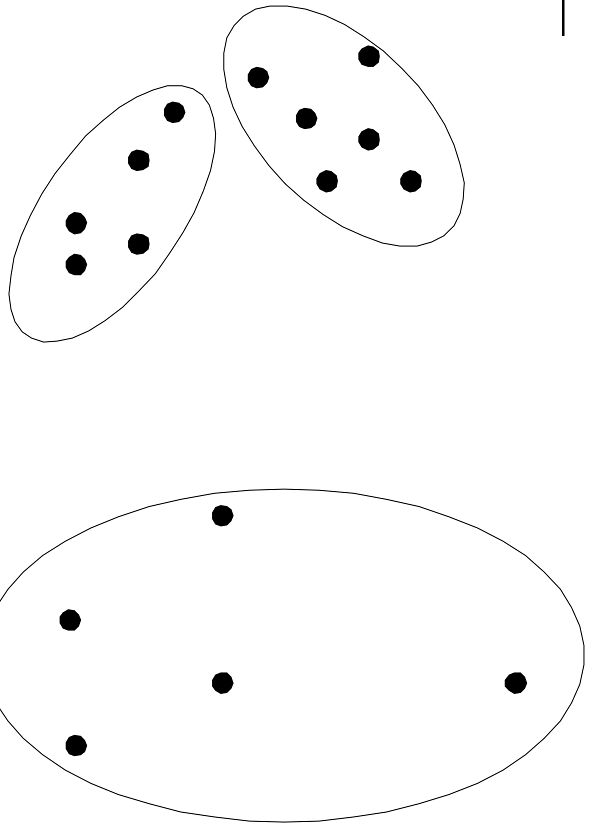
Clustering

- Clustering
  - Grouping objects into a set of clusters based on similarity

- Clustering approaches
  - Partitional clustering
  - Hierarchical clustering
  - Conceptual clustering

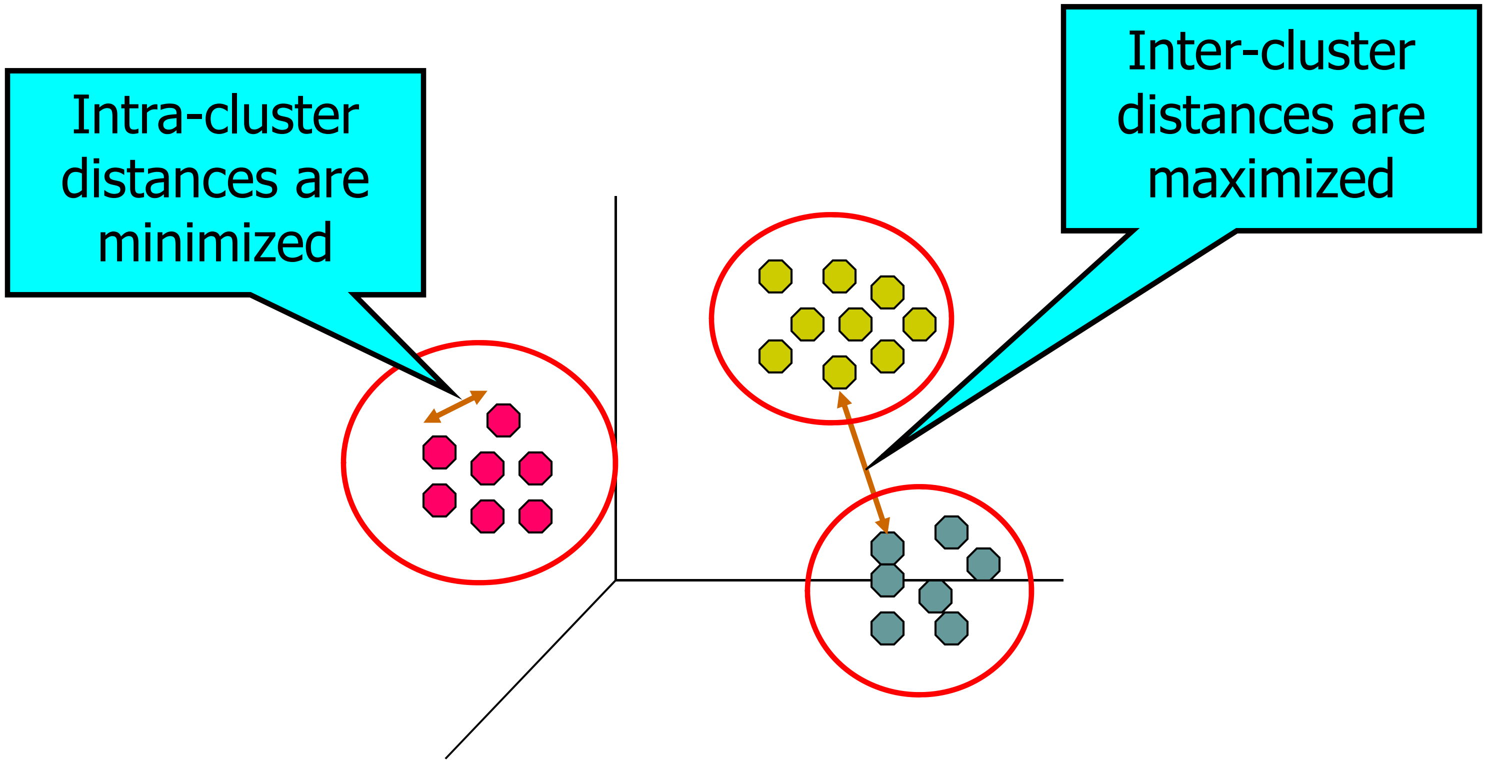
Partitional Clustering


Original Points


A Partitional Clustering

Goal of Clustering

- Finding groups of objects such that the objects in a group will be similar to one another and different from the objects in other groups. It is a computationally difficult problem (NP-hard)


Intra-cluster distances are minimized
Inter-cluster distances are maximized

Similarity and Dissimilarity

- **Similarity**
  - Numerical measure of how alike two data objects are
  - Is higher when objects are more alike
  - Often falls in the range [0,1]
- **Dissimilarity**
  - Numerical measure of how different two data objects are
  - Lower when objects are more alike
  - Minimum dissimilarity is often 0
  - Upper limit varies

$$dist = \left(\sum_{k=1}^{n} | p_k - q_k |^r \right)^{\frac{1}{r}}$$

Minkowski distance is a generalization of **Euclidean distance**, where *r* is a parameter, *n* is the number of dimensions (attributes) and p_k and q_k are, respectively, the kth attributes (components) or data objects *p* and *q*.

K-means Clustering

- **K-means algorithm:**
  - Partitional clustering approach
  - Each cluster is associated with a centroid (center point)
  - Each point is assigned to the cluster with the closest centroid
  - Number of clusters, K, must be specified as a parameter

```
1: Select $K$ points as the initial centroids.
2: repeat
3:     Form $K$ clusters by assigning all points to the closest centroid.
4:     Recompute the centroid of each cluster.
5: until The centroids don't change
```

+Several variations exist, e.g., fuzzy C-means, k-means++, etc.

Unsupervised vs. Supervised Learning

- **Applications for unsupervised learning**
  - Used in many disciplines such as Biology, Business, etc.
  - Pattern or object detection recognition: How can we group pixels into meaning regions of objects?
  - Outlier detection
  - Dimensionality reduction
  - Identification of common properties of objects, e.g., consumers
- **Some questions** to clarify
  - What's the results of unsupervised learning (clustering)?
  - What's the results of supervised learning?
  - How can we use of the results of unsupervised learning for supervised learning?
  - Is unsupervised learning more powerful than supervised learning (in terms of usefulness and accuracy of learned models)?
- How can we combine both?
  - Semi-supervised learning (like human learning)

Domain Knowledge in Learning

- Use of domain knowledge in learning
  - Human can learn patterns by only a few examples quickly.
  - Cognitive scientist argue that learners learn better and faster based on the existing domain knowledge.
  - Most machine learning algorithms are designed to learn concepts without using domain knowledge.
- Meta-DENDRAL (by Buchanan and Mitchel, 1978) used domain knowledge
  - Knowledge is represented in the form of rules (based on graphs of molecular structure for LHS and RHS)
  - Meta-DENDRAL infers the rules for the structure of organic molecules from their chemical formula and data.
  - Explanations of the results are added after the inference as positive examples for a rule.

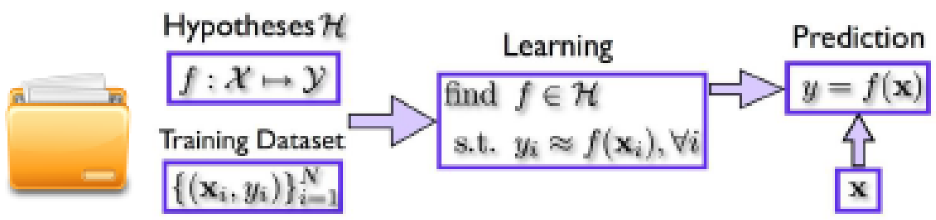
Reinforcement Learning

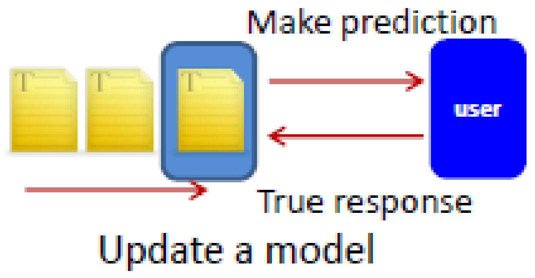
- Unlike supervised learning, learning agent is not told directly what to do, instead it goes through trial-and-error, receives feedback or reward, and learns an optimal policy to accomplish goals within its environment.
  - Inspired by old behaviorist psychology, concerned with how an agent ought to take actions in an environment so as to maximize some notion of cumulative reward.
  - Reinforcement learning is most successful when the environment is big or cannot be precisely described.
- Studied in many other disciplines
  - Control theory, operations research (called approximate dynamic programming), information theory, simulation-based optimization
  - In economics and game theory, reinforcement learning may be used to explain how equilibrium may arise under bounded rationality.

Online Learning as a Promising Method for Big Data Analysis

- Batch/Offline learning
  - Observe a **batch** of training data $\{(\mathbf{x}_i, y_i)\}_{i=1}^N$
  - Learn a model from them
  - Predict new samples accurately

- Online learning
  - Observe a **sequence** of data $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_t, y_t)$
  - Learn a model **incrementally** as instances come
  - Make the sequence of online predictions accurately


Hypotheses $\mathcal{H}$
$f : \mathcal{X} \mapsto \mathcal{Y}$
Training Dataset
$\{(\mathbf{x}_i, y_i)\}_{i=1}^N$
Learning
find $f \in \mathcal{H}$
s.t. $y_i \approx f(\mathbf{x}_i), \forall i$
Prediction
$y = f(\mathbf{x})$
$\mathbf{x}$



Categories of Machine Learning

- Symbolic learning vs. non-symbolic learning
- **Supervised learning vs. unsupervised learning**
- **Batch learning** vs. Online learning
    - Difference between online learning and reinforcement learning is very small.

- **Other types of learning**
    - Analogical learning
    - Deep learning
    - Reinforcement learning
        - Important to robot
    - Hybrid learning or universal learning
        - Important for deep learning

Applications of Machine Learning

- Data mining
  - Business intelligence, Big data analytics
- Financial modeling
  - Credit risk analysis, loan application screening
- Weather forecasting
- Speech recognition
- Games
- Many engineering problem solving
- Robotics
- Medicine and biology
- So many others

Challenges of Machine Learning

- Learning from big data
  - Online learning
- Integration of various machine learning
  - Ensemble learning
  - Deep learning
  - Universal learner
- Inconsistency in knowledge representation
- Knowledge integration and utilization
  - Use of domain knowledge
  - Use of common sense knowledge

Review Questions

- What is the principle idea used in non-symbolic learning approaches?
- What are examples of non-symbolic learning algorithms?
- How does Genetic Learning method work?
- Why may the accuracy of most non-symbolic learning algorithms not be impacted by a few bad examples in a training data set?
- What is artificial neuron?
- What are the key components of artificial neuron? How does it work?
- How are input signals processed in an artificial neuron?
- How is the output of an artificial neuron determined?
- What is Artificial Neural Network (ANN)?
- How can the McCulloch-Pitts neuron model solve a problem (e.g., AND, OR)?
- What’s the primary limitation of the McCulloch-Pitts neuron model?
- How does Perceptron Learning by Rosenblatt work?
- How can the Perceptron Learning algorithm learn the weights for a training data set? Try some examples.
- How can the learned weights be used for solving a problem? What are the applications?

- What is the primary limitation of the Perceptron Learning by Rosenblatt?
- How can we overcome the limitation of the Perceptron Learning?
- What is sigmoidal function? What are the benefits?
- How is the output of a neuron using sigmoidal function determined?
- How does a multi-layer neural network with backpropagation work?
- What is the principle idea of the Delta rule by Rumelhart? Derive the Delta rule.
- How do we determine addition (+) or subtraction (-) of ΔW in weight adjustment?
- Derive the formula of ΔW for output nodes in a multi-layer neural net.
- Derive the formula of ΔW for hidden nodes in a multi-layer neural net.
- Why is the formula for ΔW for output nodes and hidden nodes different?
- Try some example calculation of ΔW for both output nodes and hidden nodes.
- How can we implement a Deep Learning system based on the ANN approach?
- What is unsupervised learning? How can we implement an unsupervised learning?

- What’s the primary difference between supervised learning and unsupervised learning?
- How does a clustering algorithm work in general? What are the necessary components for a clustering algorithm?
- What’s the concept of “cluster”? What’s the difference between “cluster” and “class”?
- Is it always possible to find the correct number of clusters in a data set?
- How can a similarity be measured between two objects?
- How does K-means work? Try some examples using K-means algorithm.
- What are the applications of clustering or unsupervised learning?
- What’s the outcome of supervised learning?
- What’s the outcome of unsupervised learning?
- Is supervised learning more useful than unsupervised learning?
- How can we use both supervised learning and unsupervised learning methods?
- What’s the main idea of reinforcement learning? How is it different from other learning method such as supervised learning?
- Why is domain knowledge important in machine learning?

- What’s the main idea of batch/offline learning?
- What’s the main idea of online learning? Why is it nowadays important?
- What’s the main idea of deep learning?
- What’s the main idea of universal learning?
- Why is integrating various machine learning methods difficult?
- What are the applications of machine learning?
- What are the challenges of machine learning?

References

- George Fluger, Artificial Intelligence: Structures and Strategies for Complex Problem Solving, 6th edition, **Chapters 10 and 12**, Addison Wesley, 2009.