Machine learning for scientific applications

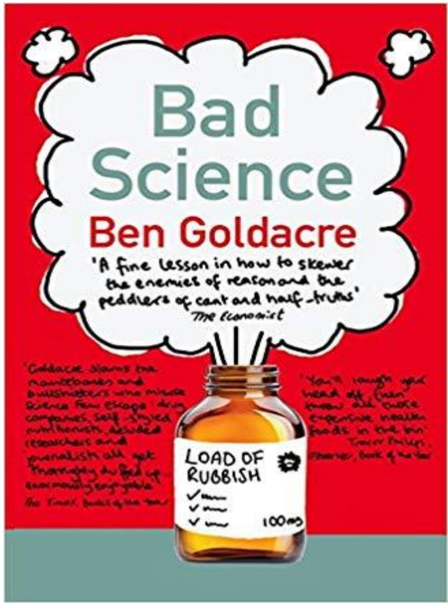
Andrea Murari and Michela Gelfusa

Overview

- Machine learning concepts
- Machine learning techniques
  - Classification
  - Regression
  - Advanced methods
- Applications in nuclear fusion
  - Disruption prediction in the JET tokamak
  - Cazusality detection

Scientific Reliability

Ioannidis's 2005 paper "*Why Most Published Research Findings Are False*" has been the most downloaded technical paper from the journal PLoS Medicine. In this paper he shows that even in the 1% of the top journals in medicine, 2/3 of the studies are contradicted by others within a few years.



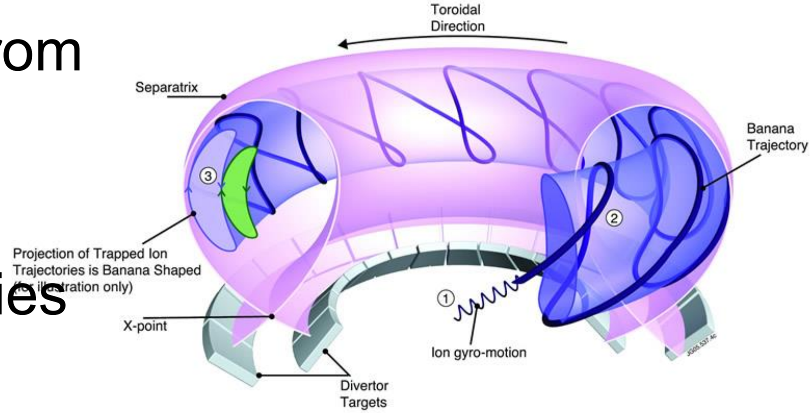
Two main fundamental reasons behind this collapse of the peer review system: <u>complexity</u> of the problems and <u>amounts</u> of data.

Complexity of Thermonuclear Plasmas

<u>Difficulty to achieve robust results is an undeniable issue in the study of complex systems</u>

The complexity of magnetised plasmas is due to:

- A different matter state far from equilibrium
- Many variables, long range interactions, high uncertainties
- Nonlinear phenomena, non separability



It is therefore very difficult to derive models from first principle. This leads to a <u>hierarchy of descriptions</u> of plasmas (particle, kinetic, fluid) and to a plethora of <u>ad ho4c models</u> of limited applicability

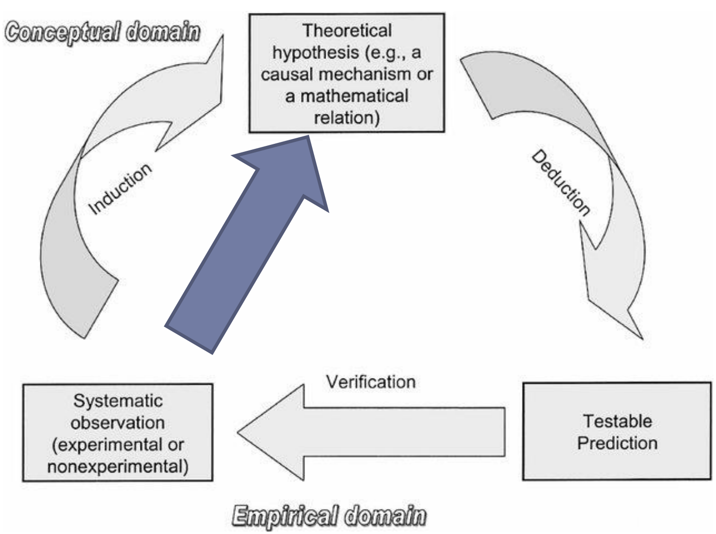
Data Deluge and Measurements

- The amount of data produced by modern societies is enormous
- JET can produce more than 55 Gbytes of data per shot (potentially about 1 Terabyte per day). Total Warehouse: almost 0.5 Petabyte
- ATLAS can produce up to about 10 Petabytes of data per year
- Hubble Space Telescope in its prime sent to earth up to 5 Gbytes of data per day
- Commercial DVD 4.7 Gbytes (Blue Ray 50 Gbytes).

These amounts of data cannot be analysed manually in a reliable way. Given the complexity of the phenomena to be studied, there is scope for the development of new data analysis tools particularly in support to theory formulation!!

Scientific cycle

- The "scientific process" relies on the formulation of testable predictions, which implies a dialectic relation between two domains: conceptual and empirical.



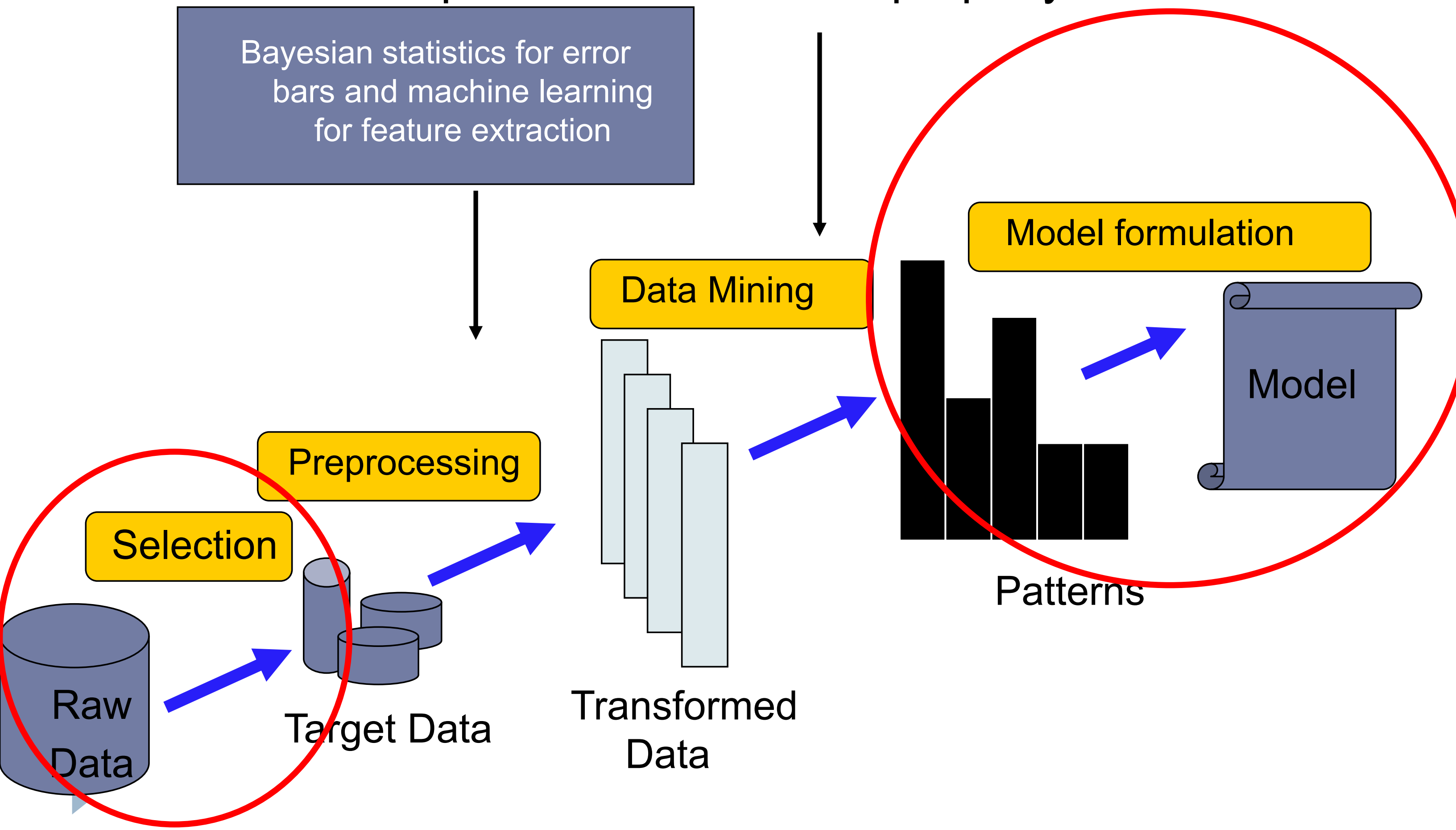
- The deduction step is very well formalised
- The induction step is more an art than a science and would benefit from: 1) more flexible tools for knowledge discovery 2) a more solid mathematization of the procedures. Data Driven Theory

1. A. Murari et al, Entropy 2017, 19, 569; doi:10.3390/e19100569
2. A. Murari et al Nuclear Fusion, Volume 57, Number 1 November 2016
3. A. Murari et al 2016 Nucl. Fusion 56 076008
4. A. Murari et al 2017 Nucl. Fusion 57 126057
5. A. Murari et at Nuclear Fusion, Volume 56, Number 2 (2015)
6. A. Murari et al Plasma Physics and Controlled Fusion (2015),57 (1),

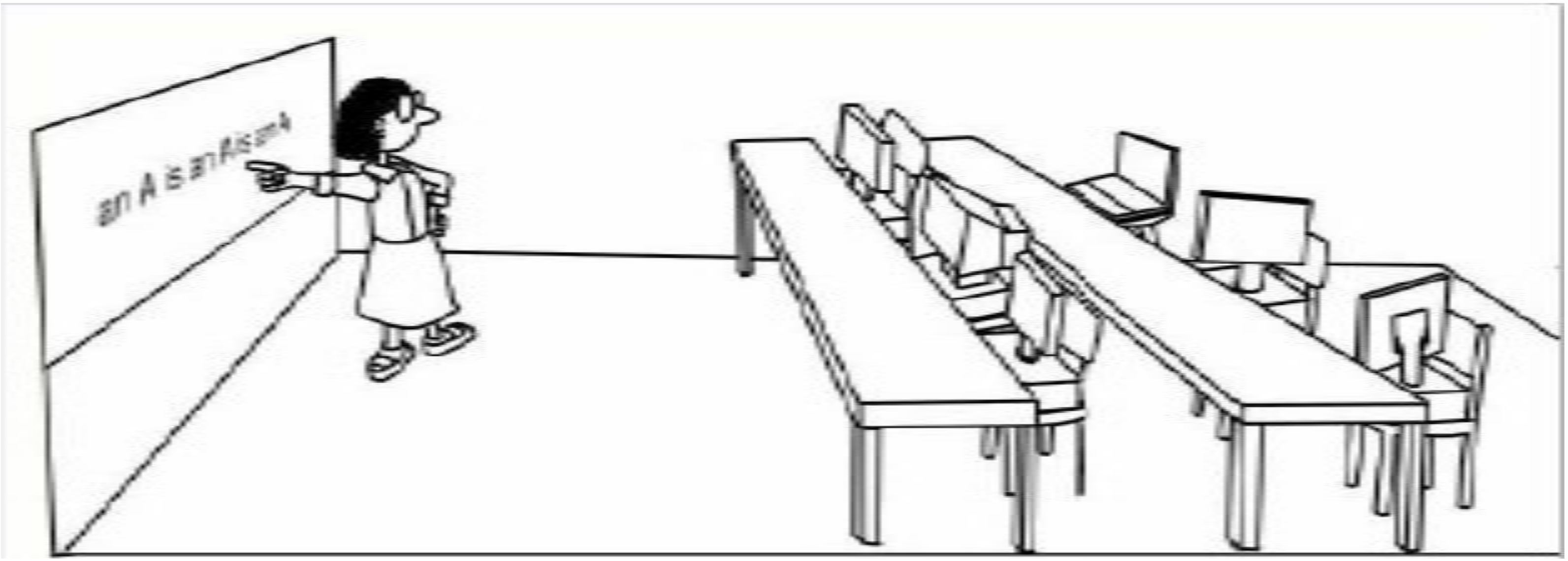
1. A. Murari et al 2016, Nuclear Fusion 56
2. A. Murari et al. Nuclear. Fusion **57** (2017) 016024 2017,
3. A. Murari et al. Nuclear Fusion , **57**, Number 12, September 2017
4. A. Murari et al Nuclear Fusion, Volume 58, Number 5, March 2018

Data Analysis: an overview

Given the complexity … amount of data, the inference proc… y structured.



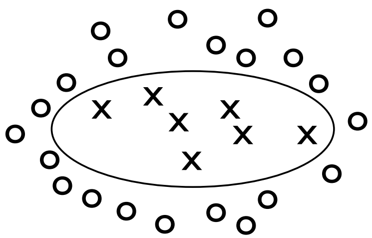
Machine learning: concepts



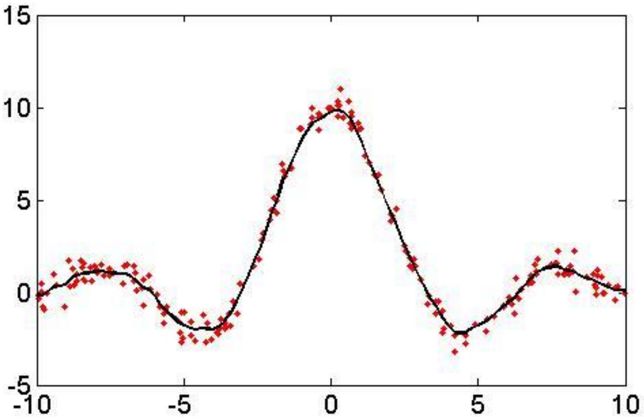
- Learning does not mean '*learning by heart*' (any computer can memorize)
- Learning means '*generalization capability*': we learn with some samples and predict for other samples

Machine learning: concepts

- The learning problem is the problem of finding a desired dependence (function) using a *limited* number of observations (training data)
  - Classification: the function represents the separation frontier between two classes
  - Regression: the function provides a fit to the data



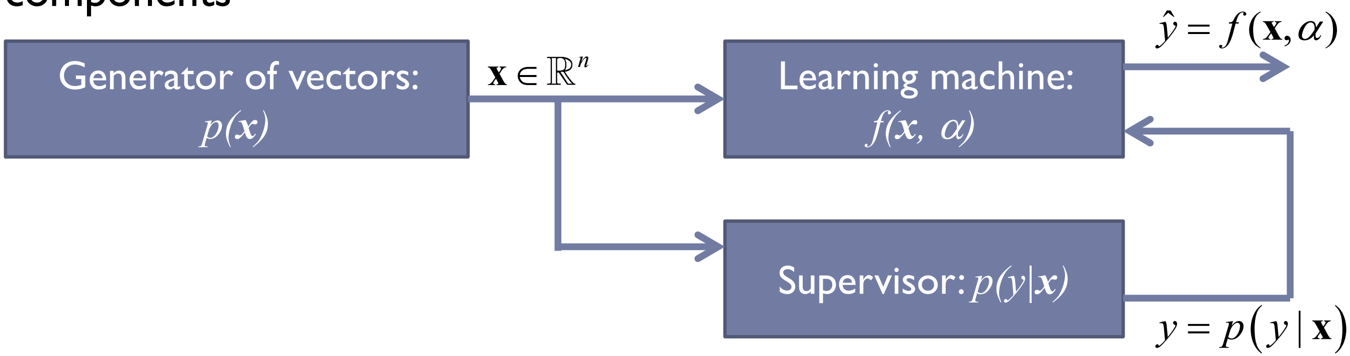
Classification



Regression

Machine learning: concepts

- The general model of *learning from examples* is described through three components

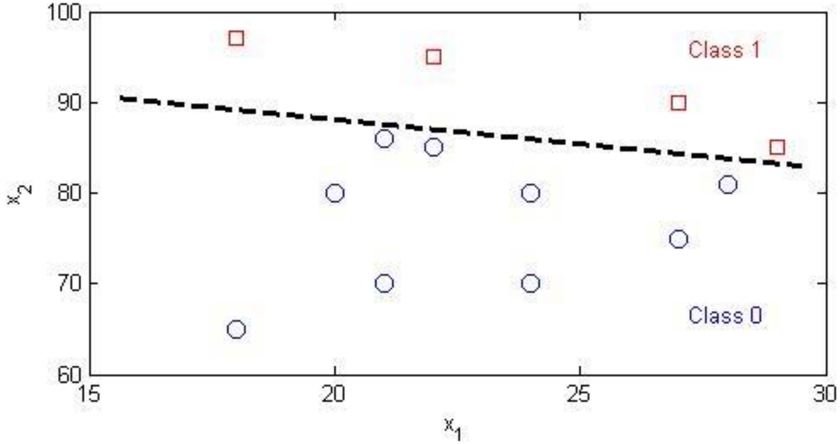


$p(\mathbf{x})$: fixed but unknown probability distribution function
$y = p(y|\mathbf{x})$ (fixed and unknown)
$f(\mathbf{x}, \alpha)$: α indicates an index in the class of functions considered
$(\mathbf{x}_i, y_i), i = 1, \ldots, N$: training samples

- The problem of learning is that of choosing from the given set of functions $f(\mathbf{x}, \alpha)$, the one that best approximates the supervisor's response

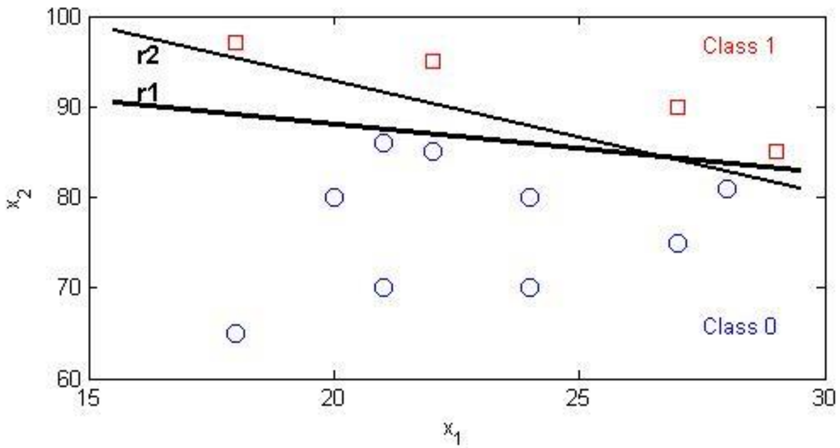
$$\{\hat{y}_1, \hat{y}_2, \ldots, \hat{y}_N\} \text{ "close" to } \{y_1, y_2, \ldots, y_N\}$$

Classification example

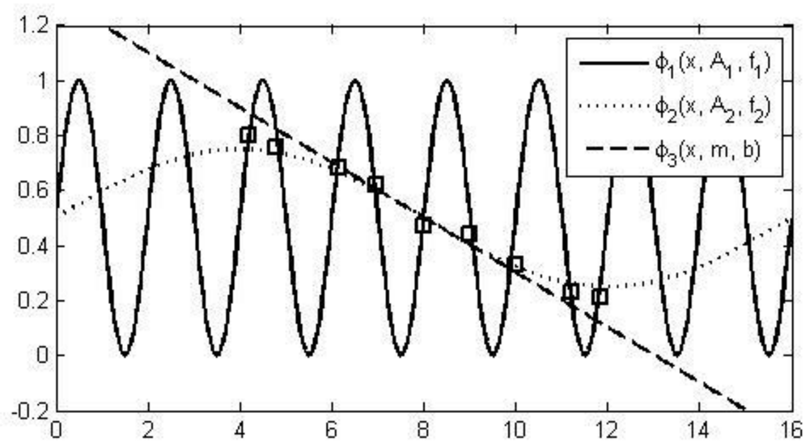


$$f(x,\alpha)=\begin{cases}0 \text{ if } x_2 < ax_1+b \\ 1 \text{ if } x_2 \geq ax_1+b\end{cases}$$

$$\alpha=(a,b)$$



Regression example



$$f(x,\alpha)=\phi_1(x,A_1,f_1), \quad \alpha=(A_1,f_1)$$

$$f(x,\beta)=\phi_2(x,A_2,f_2), \quad \beta=(A_2,f_2)$$

$$f(x,\gamma)=\phi_3(x,m,b), \quad \gamma=(m,n)$$

Classification

Description of objects

Dataset: $(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_i, y_i), \ldots, (\boldsymbol{x}_N, y_N)$

$\mathbf{x}_i \in \mathbb{R}^m$: features that are of distinctive nature (object description with attributes managed by computers)

$y_i \in \{L_1, L_2, \ldots L_K\}$: label of the sample $\boldsymbol{x}_\mathrm{i}$

- Feature types
  - Quantitative (numerical)
    - Continuous-valued (length, pressure)
    - Discrete (total basketball score, number of citizens in a town)
  - Qualitative (categorical)
    - Ordinal (education degree)
    - Nominal (profession, brand of a car)

Description of objects

- Examples of feature vectors
  - Disruptions

$$\mathbf{x}_1 = \left(\beta_p(t_s), I_p(t_s), n_e(t_s), \ldots\right), \quad y_1 \in \{D, N\}$$

$$\mathbf{x}_2 = \left(\beta_p(t_s+T), I_p(t_s+T), n_e(t_s+T), \ldots\right), \quad y_2 \in \{D, N\}$$

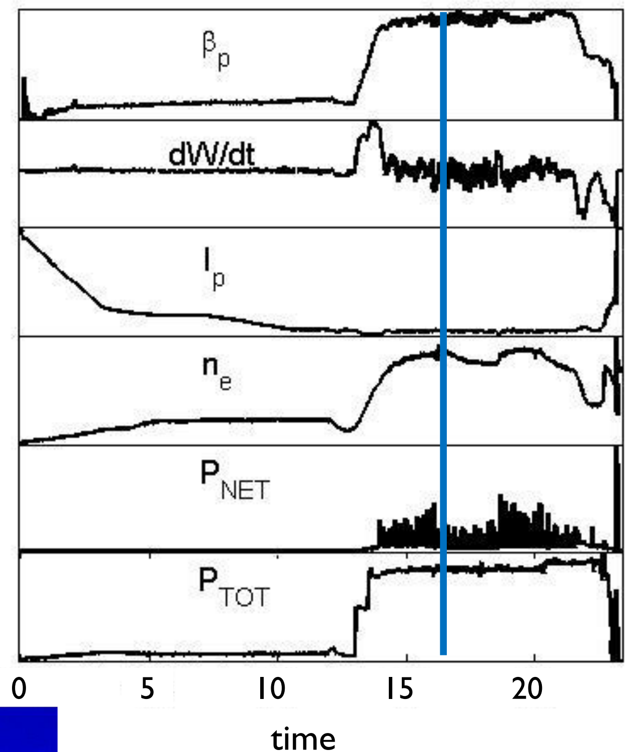
$$\mathbf{x}_3 = \left(\beta_p(t_s+2T), I_p(t_s+2T), n_e(t_s+2T), \ldots\right), \quad y_3 \in \{D, N\}$$

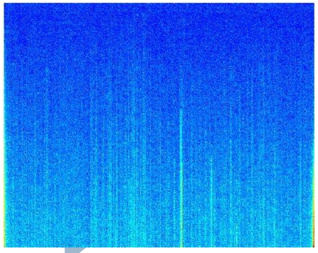
$$\cdots$$

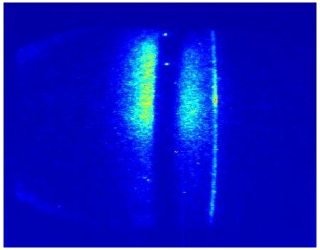
  - L/H transition

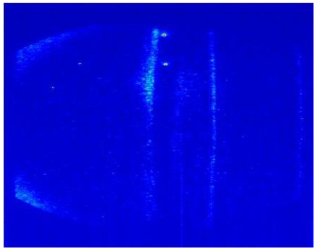
$$\left(\mathbf{x}_i, y_i\right), \quad \mathbf{x}_i \in \mathbb{R}^m, \quad y_i \in \{L, H\}$$

  - Image classification
    - $\mathbf{x}_i$: the set of pixels of an image
    - $y_i \in \{1, 2, 3\}$







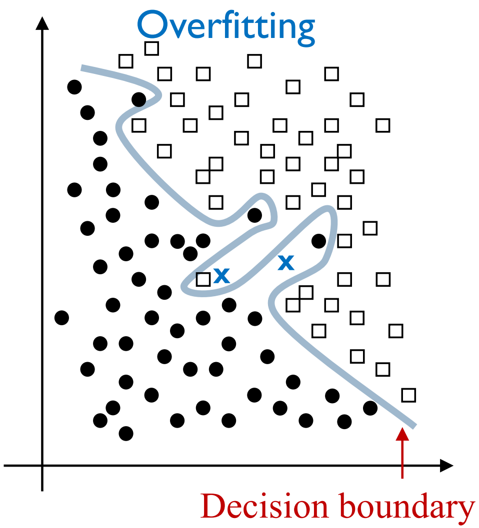


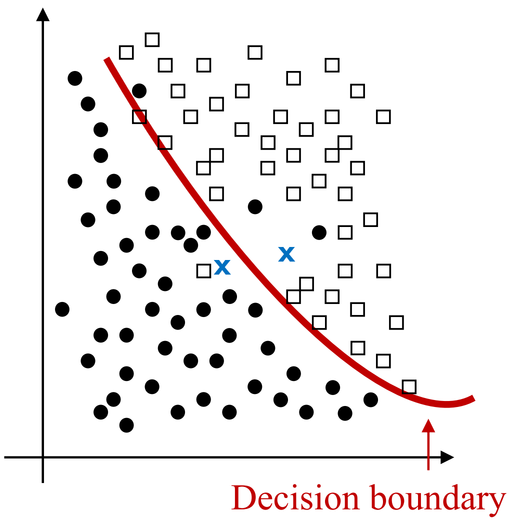
Supervised classifiers

Dataset: $(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_i, y_i), \ldots, (\boldsymbol{x}_N, y_N)$

$\boldsymbol{x}_i \in \mathbf{R}^m$: features that are of distinctive nature (object description with attributes managed by computers)
$y_i \in \{L_1, L_2, \ldots, L_K\}$: *known label* of the sample $\boldsymbol{x}_i$

Objective: to determine a separating function between classes (generalization) to predict the label of new samples with known feature vectors $((\boldsymbol{x}_{N+1}, y_{N+1}), (\boldsymbol{x}_{N+2}, y_{N+2}), \ldots)$





Supervised classifiers

- Single classifiers
  - Neural networks
  - Support Vector Machines (SVM)
  - Bayes decision theory
    - Parametric method
    - Non-parametric method
  - Classification trees
- Combining classifiers

Introduction to neural networks

- What is an (artificial) neural network
  - A large set of **nodes** (units, neurons, processing elements)
    - Each node has input and output
    - Each node performs a **simple** computation by its node function
  - **Weighted connections** between nodes
    - Connectivity gives the structure/architecture of the net
    - Connections/links have directions
    - What can be computed by a NN is primarily determined by the connections and their weights
  - A very much simplified version of networks of neurons in animal nerve systems

Introduction

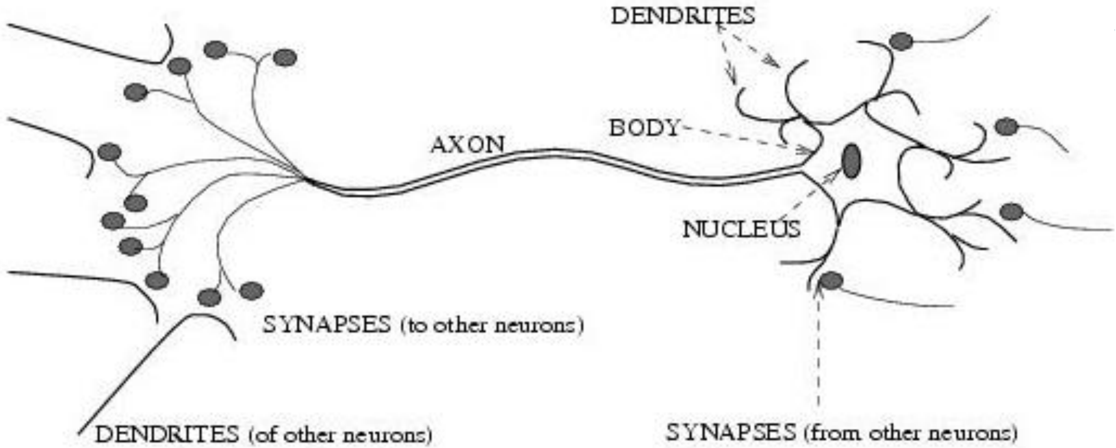
Von Neumann machine

- One or a few high speed (nm second) processors with considerable computing power
- One or a few shared high speed buses for communication
- Sequential memory access by address
- Problem-solving knowledge is separated from the computing component
- Hard to be adaptive

Human Brain

- Large # (10^{11}) of low speed processors (ms) with limited computing power
- Large # (10^{15}) of low speed connections
- Content addressable recall (CAM)
- Problem-solving knowledge resides in the connectivity of neurons
- Adaptation by changing the connectivity

- **Biological neural activity**



– Each neuron has a *body*, an *axon*, and many *dendrites*
  - Can be in one of the two states: *firing* and *rest.*
  - Neuron fires if the total incoming stimulus exceeds the threshold
– *Synapse*: thin gap between axon of one neuron and dendrite of another.
  - Signal exchange
  - Synaptic strength/efficiency

Introduction

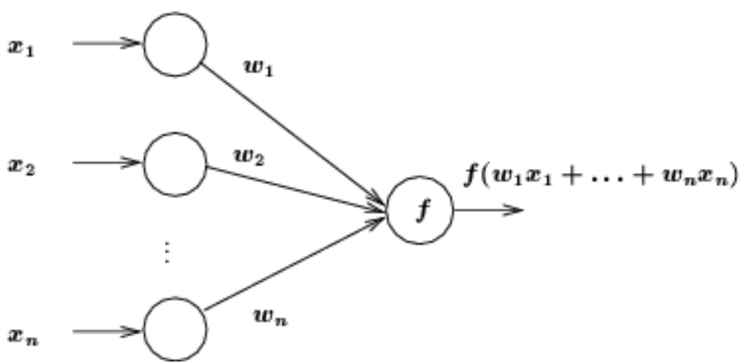
ANN	**Bio NN**
Nodes	**Cell body**
input	signals from other neurons
output	firing frequency
node function	firing mechanism
Connections	**Synapses**
connection strength	synaptic strength

- Highly parallel, simple local computation (*at neuron level*) achieves global results as emerging property of the interaction (*at network level*)
- Pattern directed (meaning of individual nodes only in the context of a pattern)
- Fault-tolerant/graceful degrading
- Learning/adaptation plays important role.

ANN Neuron Models

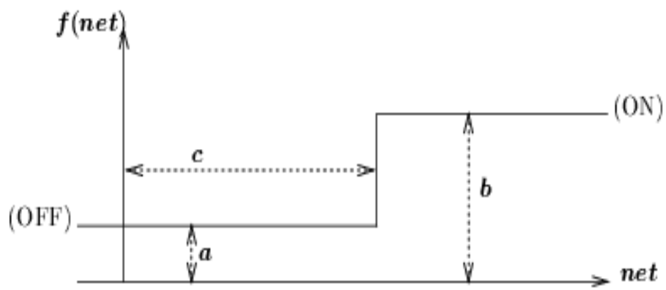
- Node Input
  - Each node has one or more inputs from other nodes, and one output to other nodes
  - Input/output values can be
    - Binary {0, 1}; Bipolar {-1, 1}; or Continuous (bounded or not)
- Weighted sum of inputs
  -

$$\text{output} = f(net) \text{ where } net = \sum_{i=1}^{n} w_i x_i$$

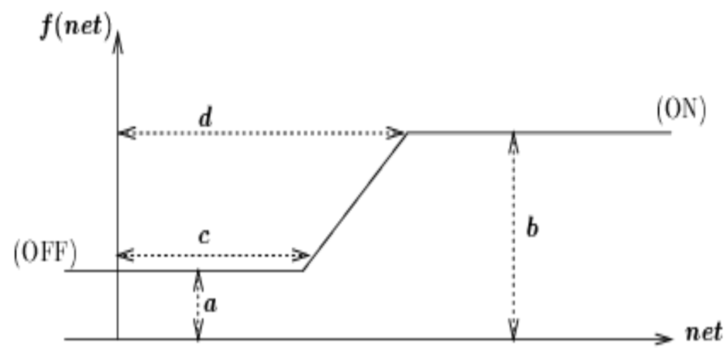


Node Functions

- Node functions (linear)
  - Identity function: $f(net) = net.$
  - Constant function: $f(net) = c.$
- Node functions (**non-linear**)
  - Step functions and ramp functions
  - Sigmoid functions (**differentiable**)



Step function



Ramp function

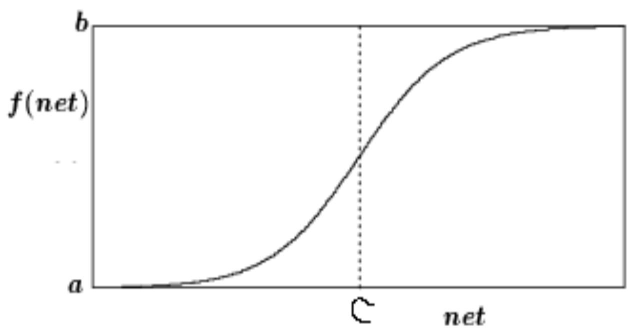
Sigmoid function

- S-shaped
- Continuous and everywhere differentiable
- Rotationally symmetric
- Asymptotically approaches saturation points
- Examples:
  - Logistic function

$$o_j = \frac{1}{1+e^{-net_j}}$$

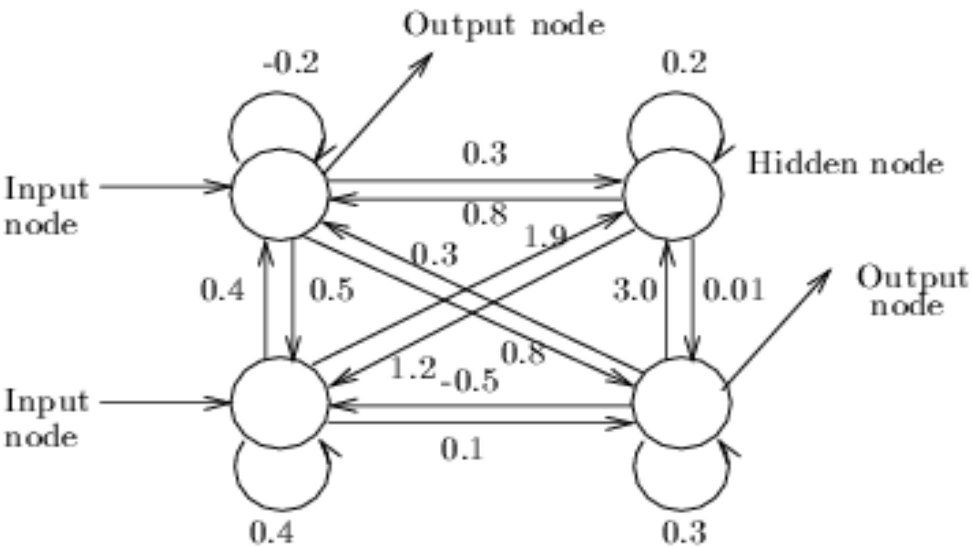
  - Hyperbolic tangent

$$o_j = \frac{e^{net_j} - e^{-net_j}}{e^{net_j} + e^{-net_j}}$$



Network Architecture

- **(Asymmetric) Fully Connected Networks**
  - Every node is connected to every other node
  - Connection may be excitatory (positive), inhibitory (negative), or irrelevant ($\approx$ 0).
  - Most general
  - Symmetric fully connected nets: weights are symmetric ($w_{ij} = w_{ji}$)



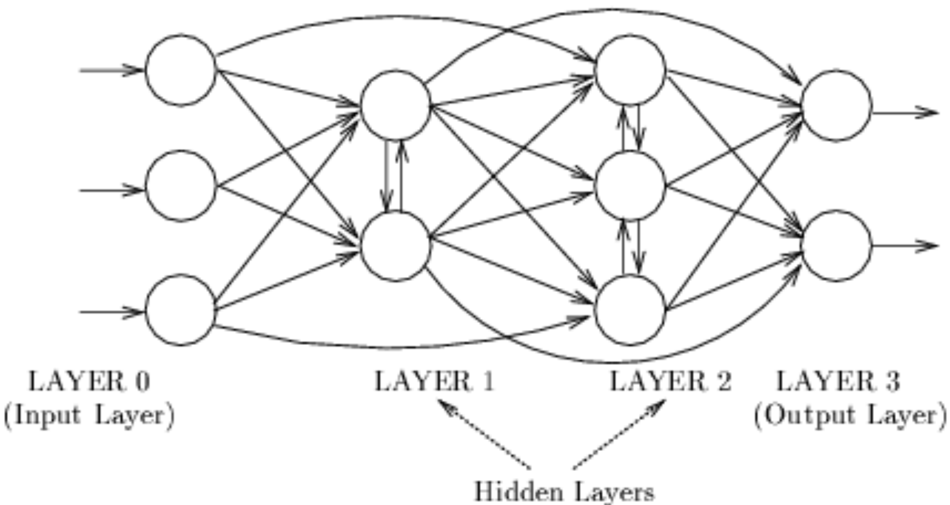
Input nodes: receive input from the environment

Output nodes: send signals to the environment

Hidden nodes: no direct interaction to the environment

Network Architecture

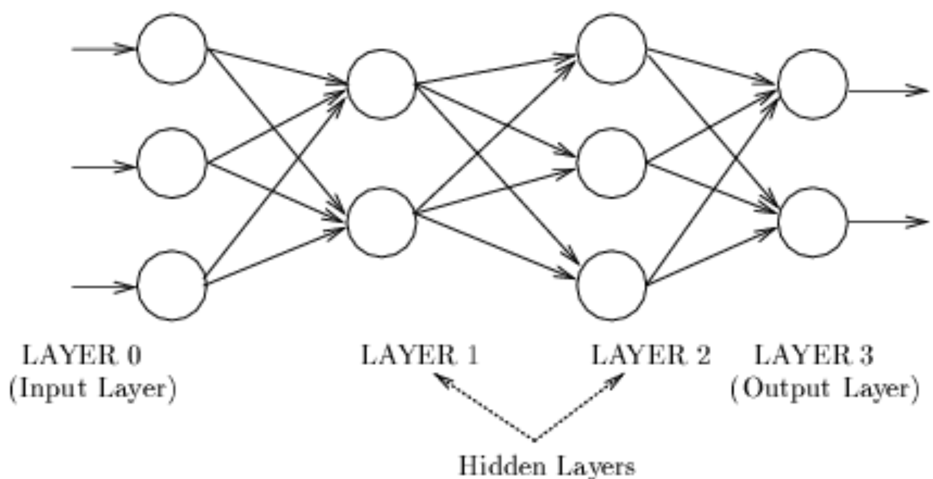
- **Layered Networks**
  - Nodes are partitioned into subsets, called layers.
  - No connections that lead from nodes in layer *j* to those in layer *k* if *j* > *k*.



- Inputs from the environment are applied to nodes in layer 0 (**input layer**).
- Nodes in input layer are place holders with no computation occurring (i.e., their node functions are identity function)

Network Architecture

- **Feedforward Networks**
  - A connection is allowed from a node in layer *i* only to nodes in layer *i* + 1.
  - Most widely used architecture.

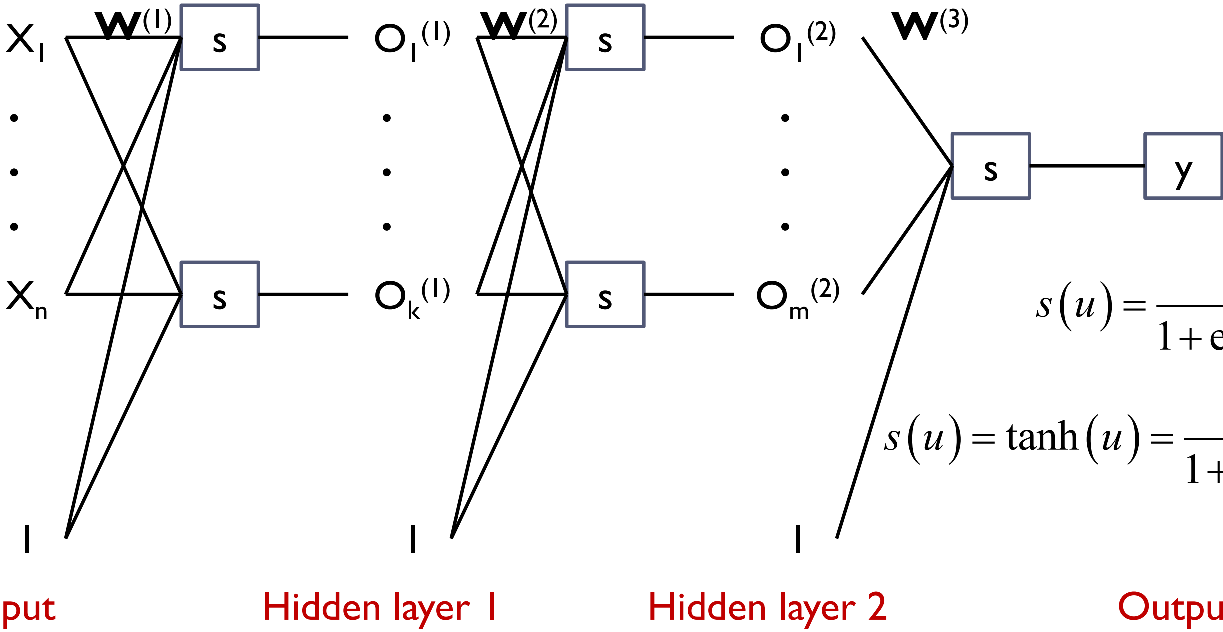


Conceptually, nodes at higher levels successively abstract features from preceding layers

Supervised classifiers: artificial neural networks

Samples: $(\mathbf{x}_j, y_j)$, $\mathbf{x}_j \in R^n$, $j = 1,\ldots,N$, $y \in \{L_1, L_2\}$

Aim: determine $\mathbf{W}$ for a minimum error



$$s(u) = \frac{1}{1+\exp(-u)}$$

$$s(u) = \tanh(u) = \frac{2}{1+\exp(-2u)} - 1$$

$\mathbf{W}^{(1)}$: (n+1 x k) matrix

$\mathbf{W}^{(2)}$: (k+1 x m) matrix

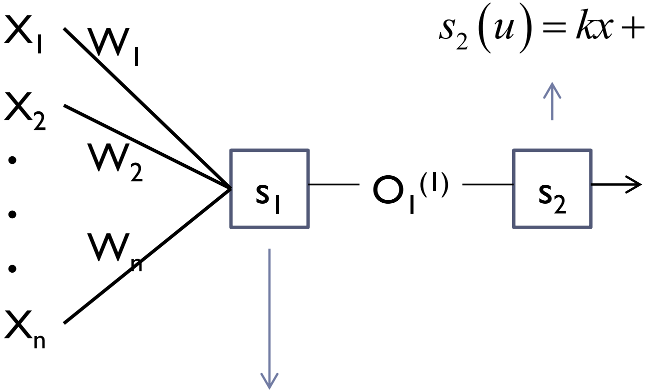
$\mathbf{W}^{(3)}$: (m+1 x 1) matrix

C. M. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press

Supervised classifiers: artificial neural networks (example)

Samples: $(\mathbf{x}_j, y_j)$, $\mathbf{x}_j \in R^n$, $j = 1,\dots,N$, $y \in \{L_1, L_2\}$

$$s_2(u) = kx + b$$



$$o_2^{(2)} = \frac{2k}{1+\exp\left[-2\left(W_1x_1 + W_2x_2 + \dots + W_nx_n\right)\right]} - k + b$$

$$s_1(u) = \tanh(u) = \frac{2}{1+\exp(-2u)} - 1$$

$$o_1^{(1)} = \frac{2}{1+\exp\left[-2\left(W_1x_1 + W_2x_2 + \dots + W_nx_n\right)\right]} - 1$$

Given $\mathbf{x}$ to classify:

if $O_2^{(2)} \geq$ threshold
$\mathbf{x} \in \{L_1\}$
otherwise
$\mathbf{x} \in \{L_2\}$

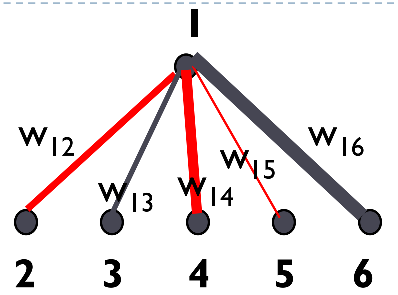
Neural Network Learning

- Real neural learning
  - Synapses change size and strength with experience.
  - **Hebbian learning**: When two connected neurons are firing at the same time, the strength of the synapse between them increases.

    "Neurons that fire together, wire together."
- ANN learning
  - Construct the NN by training using samples
  - Training is primarily to change the weights
  - We will look at two particular training methods
    - **Perceptron**
    - **backpropogation**

Perceptron

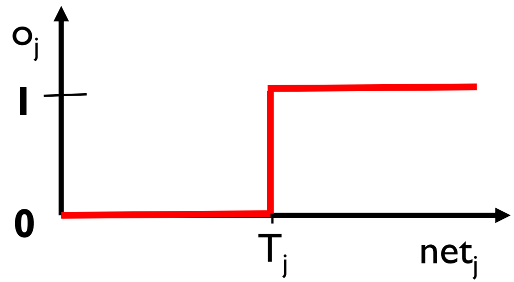
- Structure: single layer



- Non-linear output nodes

  Threshold units

$$o_j = \begin{cases} 0 \text{ if } net_j < T_j \\ 1 \text{ if } net_i \geq T_j \end{cases}$$



- Supervised learning
  - Learning weights so that output node produces correct output for each training sample

Perceptron Learning Rule

- Update weights by:

  $$w_{ji} = w_{ji} + \eta(t_j - o_j)o_i$$

  where η is the "learning rate"

  t_j is the teacher specified output for unit *j*

  $(t_j - o_j)$ is the **error** (training is error driven)

- Equivalent to rules:
  - If output is correct do nothing.
  - If output is high, lower weights on active inputs
  - If output is low, increase weights on active inputs

Perceptron Learning Algorithm

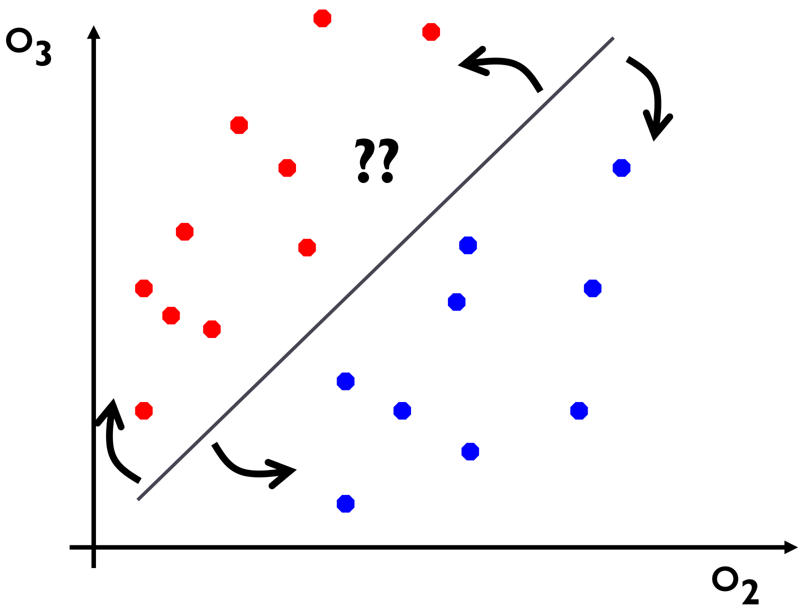
- Iteratively update weights until convergence.

Initialize weights to random values
Until outputs of all training examples are correct
    For each training pair, E, do:
        Compute current output o_j for E given its inputs
        Compare current output to target value, t_j, for E
        Update synaptic weights using learning rule

- Each execution of the outer loop is typically called an *epoch*.

Perceptron as a Linear Separator

- Since perceptron uses linear threshold function, it is searching for a linear separator that discriminates the classes.



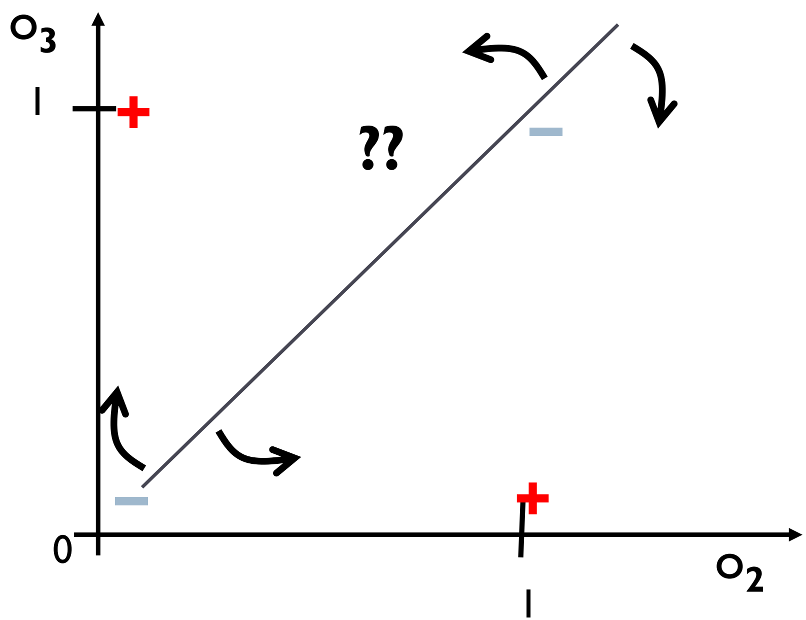
$$w_{12}o_2 + w_{13}o_3 > T_1$$

$$o_3 > -\frac{w_{12}}{w_{13}}o_2 + \frac{T_1}{w_{13}}$$

Or hyperplane in n-dimensional space

Concept Perceptron Cannot Learn

- Cannot learn exclusive-or, or parity function in general because they are not linearly separable.



Perceptron Convergence Theorem

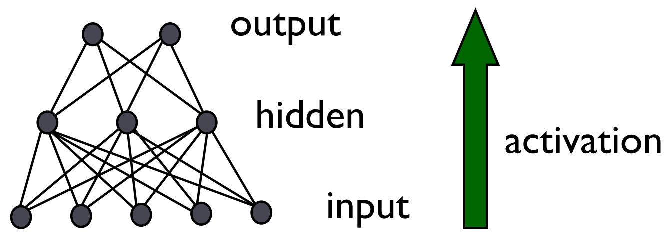
- **Perceptron convergence theorem**: If the data is linearly separable and therefore a set of weights exist that are consistent with the data, then the Perceptron algorithm will eventually converge to a consistent set of weights.
- **Perceptron cycling theorem**: If the data is not linearly separable, the Perceptron algorithm will eventually repeat a set of weights and threshold at the end of some epoch and therefore enter an infinite loop.
  - By checking for repeated weights+threshold, one can guarantee termination with either a positive or negative result.

Perceptron Limits

- System obviously cannot learn concepts it cannot represent.
- Minksy and Papert (1969) wrote a book analyzing the perceptron and demonstrating many functions it could not learn.
- These results discouraged further research on neural nets; and symbolic AI became the dominate paradigm.

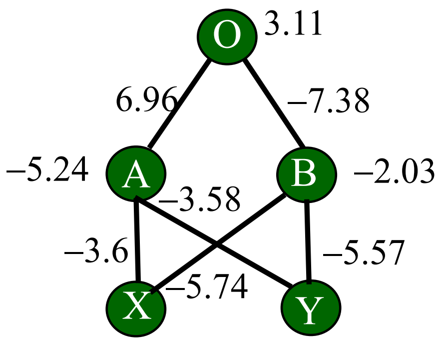
Multi-Layer Feed-Forward Networks

- A typical multi-layer network consists of an input layer, one or more hidden and output layer, each fully connected to the next, with activation feeding forward.



- **Nodes at hidden layers MUST be non-linear** (typically a sigmoid function)
- Given an arbitrary number of hidden units with a single hidden layer, there exists a set of weights which can compute any given Boolean function (or any L2 function). But an effective learning algorithm for such networks was thought to be difficult.

Sample Learned XOR Network



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

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

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

$$= X \oplus Y$$

Gradient Descent

- Define objective to minimize error:

$$E(W) = \sum_{d \in D} \sum_{k \in K} (t_{kd} - o_{kd})^2$$

where *D* is the set of training examples, *K* is the set of output units, t_{kd} and o_{kd} are, respectively, the teacher and current output for unit *k* for example *d*.

- The derivative of a sigmoid unit with respect to net input is:

$$\frac{\partial o_j}{\partial net_j} = o_j(1 - o_j)$$

- Learning rule to change weights to minimize error is:

$$\Delta w_{ji} = -\eta \frac{\partial E}{\partial w_{ji}}$$

Backpropagation Learning Rule

- Each weight changed by:

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

$$\delta_j = o_j(1-o_j)(t_j - o_j) \qquad \text{if } j \text{ is an output unit}$$

$$\delta_j = o_j(1-o_j)\sum_k \delta_k w_{kj} \qquad \text{if } j \text{ is a hidden unit}$$

where η is a constant called the learning rate

t_j is the correct teacher output for unit *j*

δ_j is the error measure for unit *j*

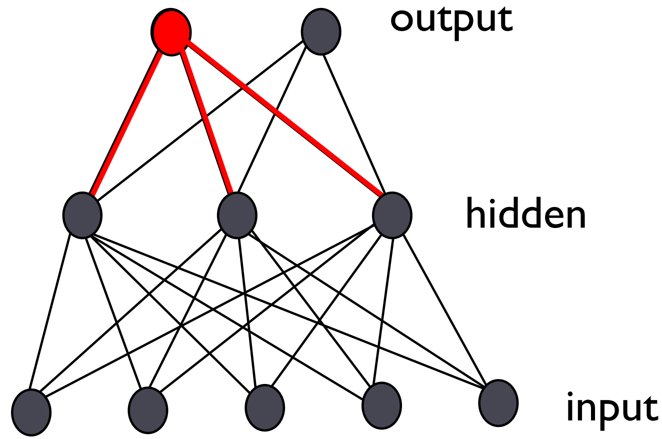
Error Backpropagation

- 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

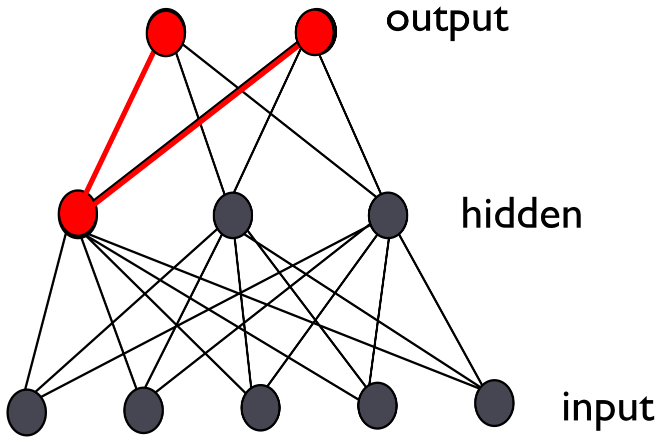
$$\Delta w_{ji} = \eta \delta_j o_i$$



Error Backpropagation

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

$$\delta_j = o_j(1-o_j)\sum_k \delta_k w_{kj}$$



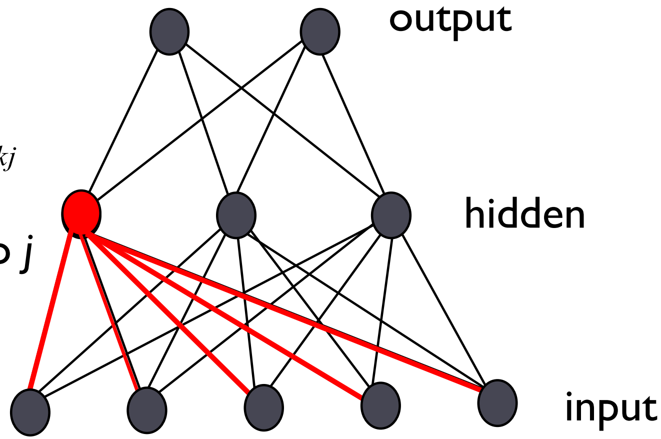
Error Backpropagation

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

$$\delta_j = o_j(1-o_j)\sum_k \delta_k w_{kj}$$

Update weights into j

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



Backpropagation Training Algorithm

Create the 3-layer network with *H* hidden units with full connectivity between layers. Set weights to small random real values.

Until all training examples produce the correct value (within ε), or mean squared error ceases to decrease, or other termination criteria:

- Begin epoch
- For each training example, *d*, do:
  - Calculate network output for *d*'s input values
  - Compute error between current output and correct output for *d*
  - Update weights by backpropagating error and using learning rule
- End epoch

Hidden Unit Representations

- Trained hidden units can be seen as newly constructed features that make the target concept linearly separable in the transformed space.
- On many real domains, hidden units can be interpreted as representing meaningful features such as vowel detectors or edge detectors, etc..
- However, the hidden layer can also become a distributed representation of the input in which each individual unit is not easily interpretable as a meaningful feature.

Representational Power

- **Boolean functions**: Any boolean function can be represented by a two-layer network with sufficient hidden units.
- **Continuous functions**: Any bounded continuous function can be approximated with arbitrarily small error by a two-layer network.
  - Sigmoid functions can act as a set of basis functions for composing more complex functions, like sine waves in Fourier analysis.
- **Arbitrary function**: Any function can be approximated to arbitrary accuracy by a three-layer network.

Successful Applications

- Text to Speech (NetTalk)
- Fraud detection
- Financial Applications
  - HNC (eventually bought by Fair Isaac)
- Chemical Plant Control
  - Pavillion Technologies
- Automated Vehicles
- Game Playing
  - Neurogammon
- Handwriting recognition

Hand-written character recognition

- MNIST: a data set of hand-written digits
  - 60,000 training samples
  - 10,000 test samples
  - Each sample consists of 28 x 28 = 784 pixels
- Various techniques have been tried

Technique	Failure rate for test samples
Linear classifier:	12.0%
2-layer BP net (300 hidden nodes)	4.7%
3-layer BP net (300+200 hidden nodes)	3.05%
Support vector machine (SVM)	1.4%
Convolutional net	0.4%
6 layer BP net (7500 hidden nodes):	**0.35%**

Strengths of BP Learning

- **Great representation power**
  - Any meaningful function can be represented by a BP net
  - Many such functions can be approximated by BP learning (gradient descent approach)
- **Easy to apply**
  - Only requires a good set of training samples
  - Does not require substantial prior knowledge or deep understanding of the domain itself (ill structured problems)
  - Tolerates noise and missing data in training samples (graceful degrading)
- **Easy to implement** the core of the learning algorithm
- **Good generalization power**
  - Often produces accurate results for inputs outside the training set

Deficiencies of BP Learning

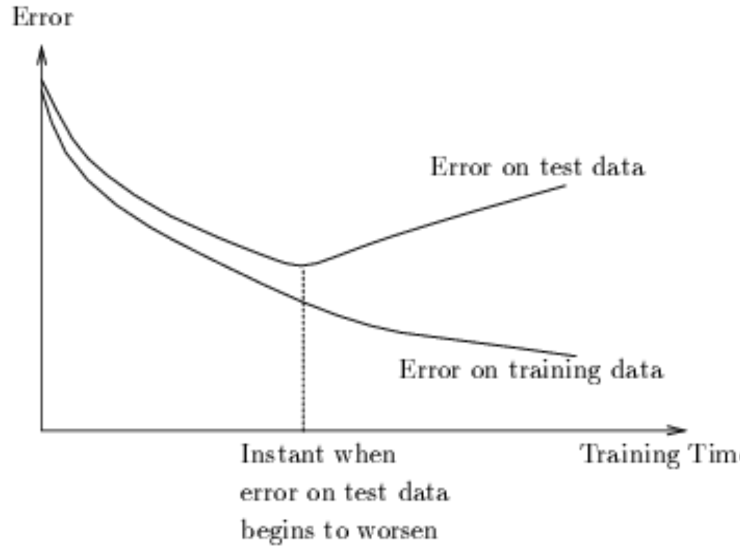
- Learning often takes a **long time** to converge
  - Complex functions often need hundreds or thousands of epochs
- The network is essentially a **black box**
  - It may provide a desired mapping between input and output vectors (***x*, *o***) but does not have the information of why a particular ***x*** is mapped to a particular ***o*.**
  - It thus cannot provide an intuitive (e.g., causal) explanation for the computed result.
  - This is because the hidden nodes and the learned weights do not have clear semantics.
    - What can be learned are operational parameters, not general, abstract knowledge of a domain
  - Unlike many statistical methods, there is no theoretically well-founded way to **assess the quality** of BP learning
    - What is the confidence level of *o* computed from input *x* using such net?
    - What is the confidence level for a trained BP net, with the final training/test error (which may or may not be close to zero)?

Deficiencies of BP Learning

- Problem with gradient descent approach
  - only guarantees to reduce the total error to a **local minimum**. (***E*** might not be reduced to zero)
    - Cannot escape from the local minimum error state
    - **Not every function that is representable can be learned**
  - How bad: depends on the shape of the error surface. Too many valleys/wells will make it easy to be trapped in local minima
  - Possible remedies:
    - Try nets with different # of hidden layers and hidden nodes (they may lead to different error surfaces, some might be better than others)
    - Try different initial weights (different starting points on the surface)
    - Forced escape from local minima by random perturbation (e.g., **simulated annealing**)

- **Generalization** is not guaranteed even if the error is reduced to 0
  - **Over-fitting**/over-training problem: trained net fits the training samples perfectly (*E* reduced to 0) but it does not give accurate outputs for inputs not in the training set

– Possible remedies:
- More and better samples
- Using smaller net if possible
- Using larger error bound (forced early termination)
- Introducing noise into samples
  - modify $(x_1,\ldots, x_n)$ to $(x_1+\alpha_1,\ldots, x_n+\alpha_n)$ where α_i are small random displacements



- **Cross-Validation**
  - leave some (~20%) samples as test data (not used for weight update)
  - periodically check error on test data
  - learning stops when error on test data starts to increase