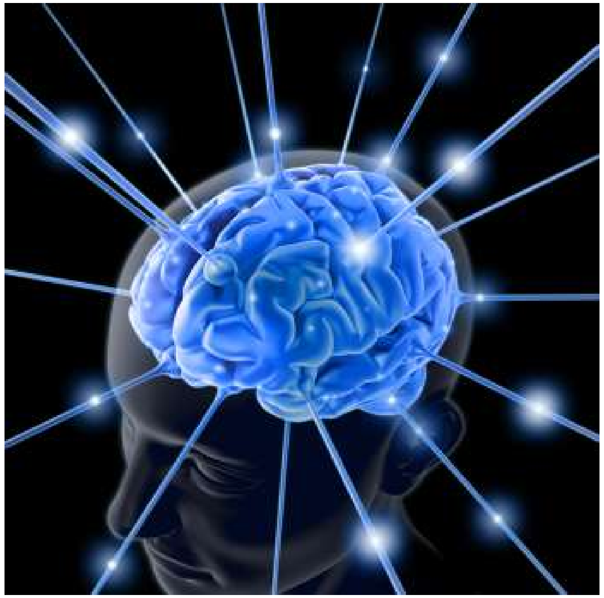
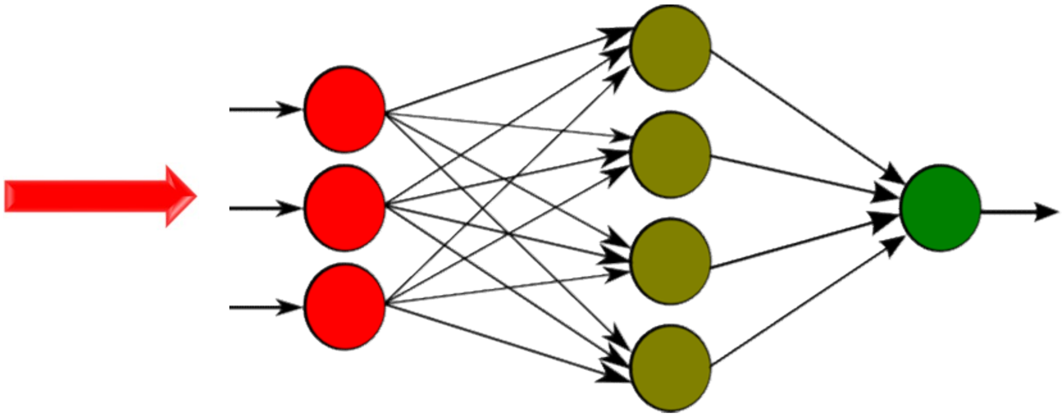
Neural Networks

Lecturer: Dr. Bo Yuan

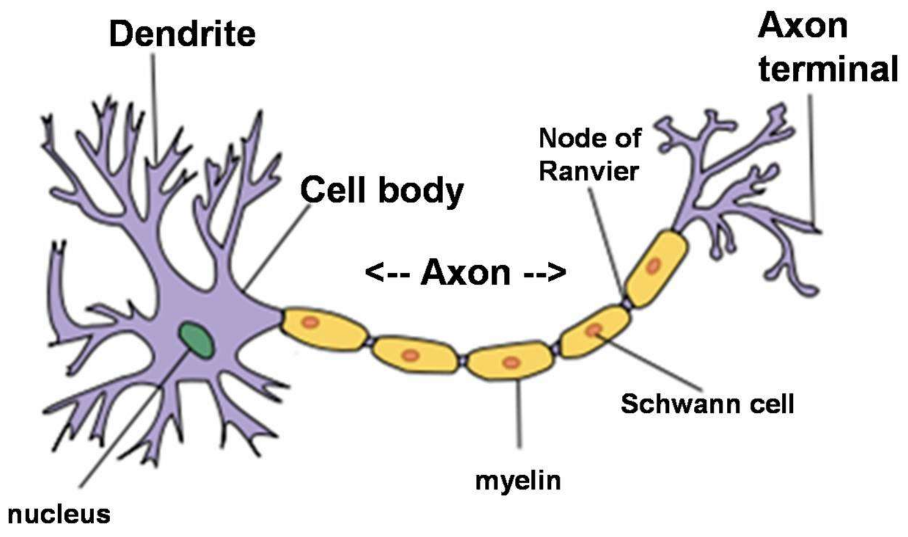
E-mail: yuanb@sz.tsinghua.edu.cn

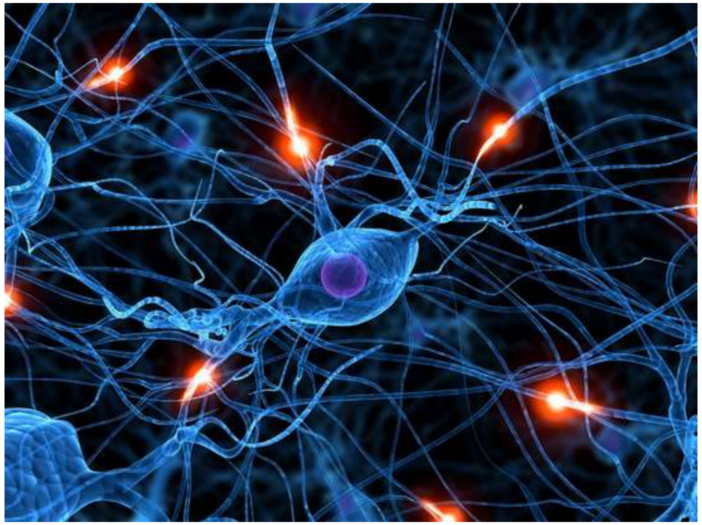
Overview





Biological Motivation





- 10^{11}: The number of neurons in the human brain
- 10^{4}: The average number of connections of each neuron
- 10^{-3}: The fastest switching time of neurons
- 10^{-10}: The switching speed of computers
- 10^{-1}: The time required to visually recognize your mother

Biological Motivation

- The power of parallelism

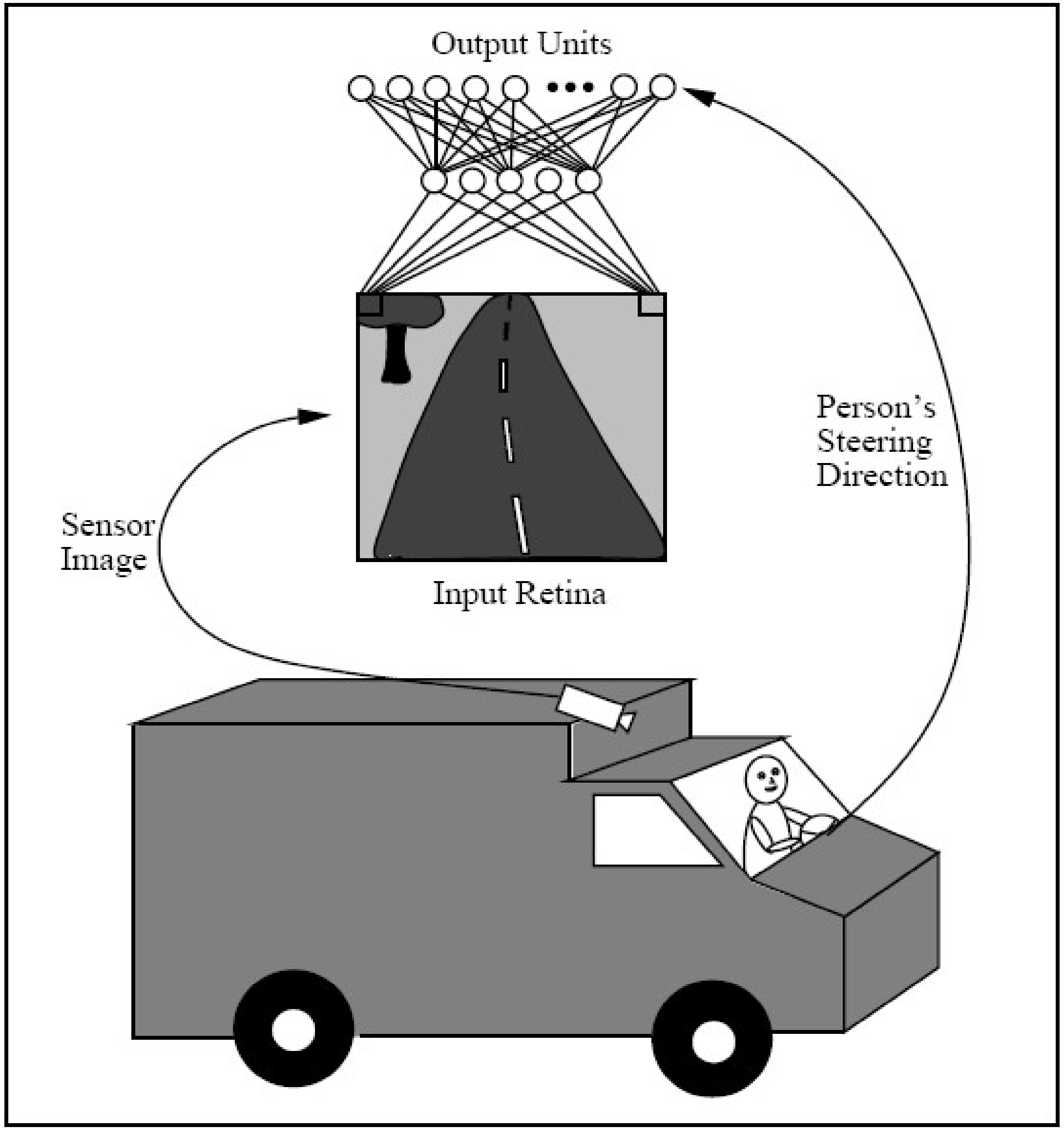
- The information processing abilities of biological neural systems follow from highly parallel processes operating on representations that are distributed over many neurons.

- The motivation of ANN is to capture this kind of highly parallel computation based on distributed representations.

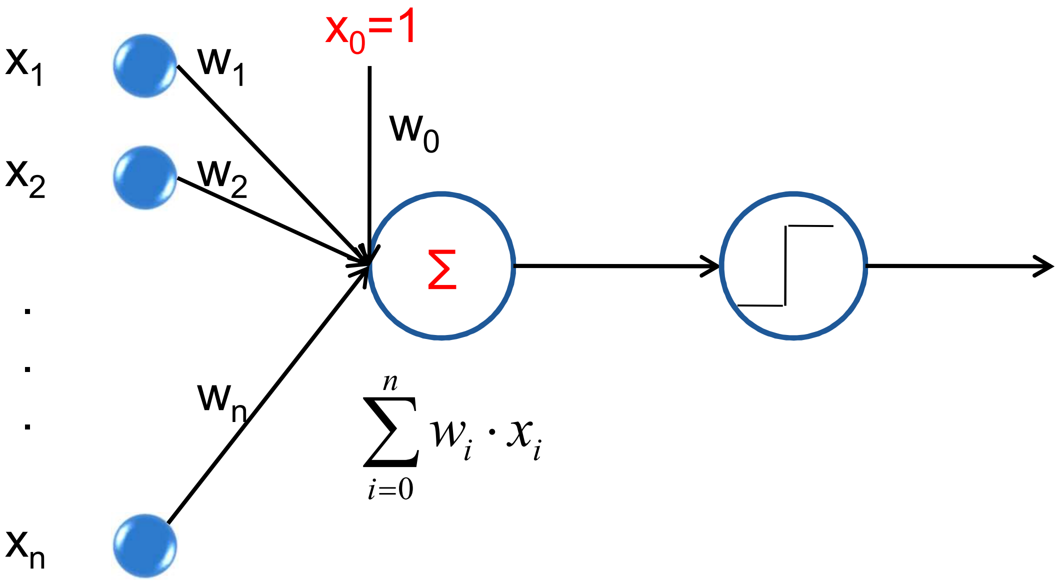
- Group A
  - Using ANN to study and model biological learning processes.

- Group B
  - Obtaining highly effective machine learning algorithms, regardless of how closely these algorithms mimic biological processes.

Case Study

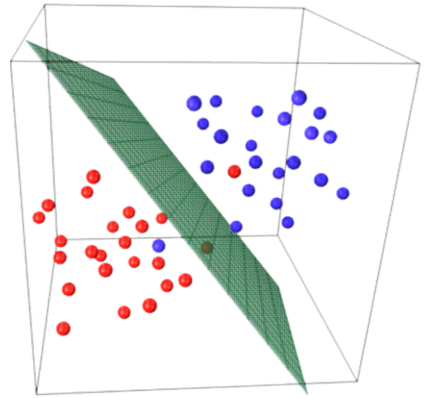

Output Units
Person's
Steering
Direction
Sensor
Image
Input Retina

Perceptrons

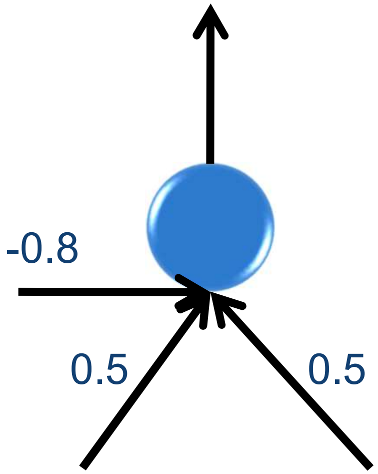


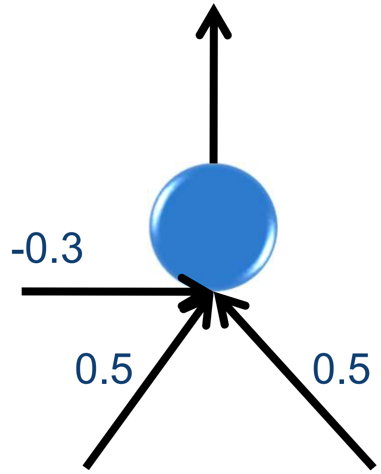
$$o = \begin{cases} 1, & if \sum_{i=0}^{n} w_i \cdot x_i > 0 \\ 0, & otherwise \end{cases}$$

$$o(x_1,\ldots,x_n) = \begin{cases} 1, & if \ w_0 + w_1 \cdot x_1 + \cdots + w_n \cdot x_n > 0 \\ 0, & otherwise \end{cases}$$



Power of Perceptrons





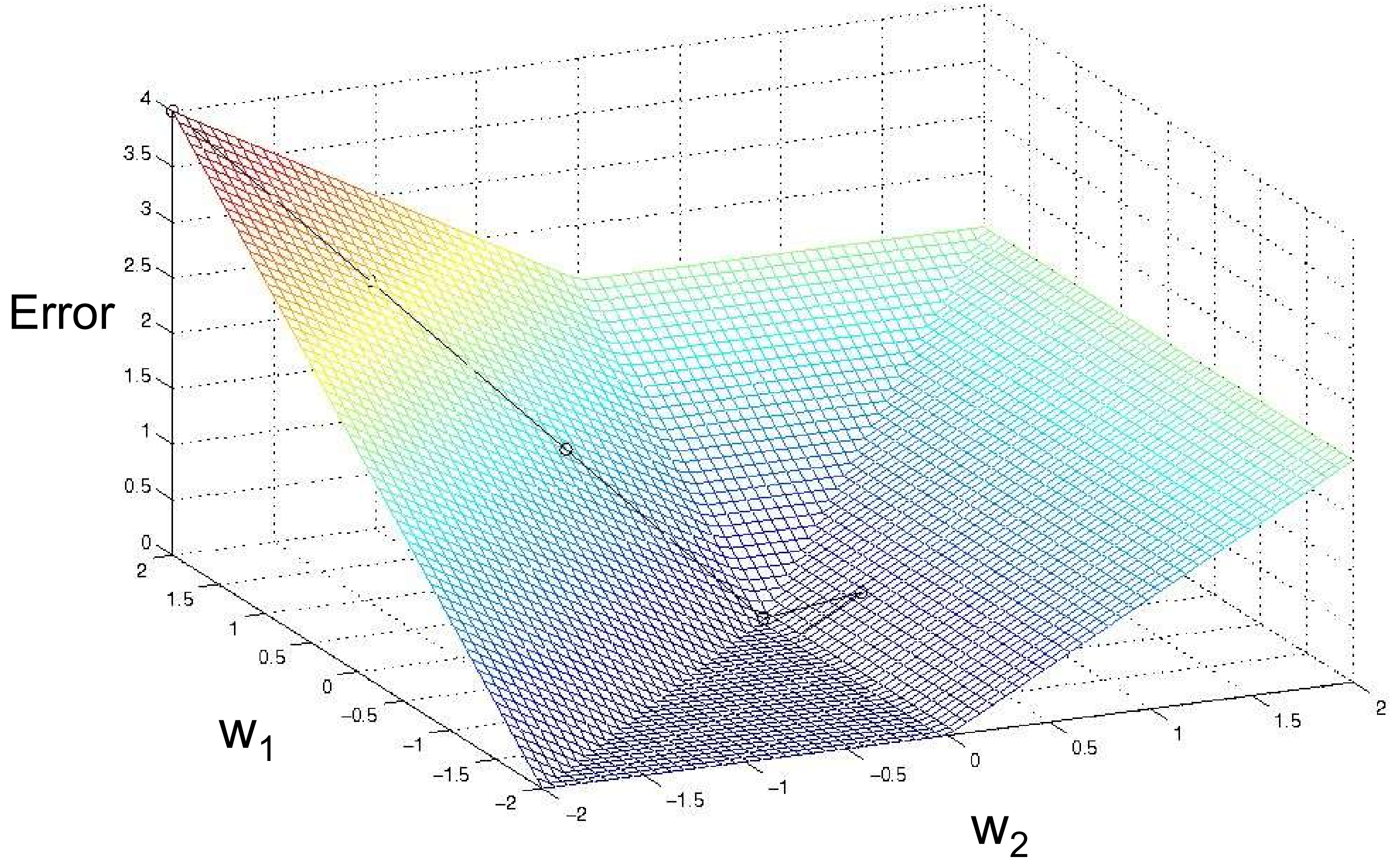
AND

Input		∑	Output
0	0	-0.8	0
0	1	-0.3	0
1	0	-0.3	0
1	1	0.3	1

OR

Input		∑	Output
0	0	-0.3	0
0	1	0.2	1
1	0	0.2	1
1	1	0.7	1

Error Surface



Gradient Descent

$$E(\vec{w}) \equiv \frac{1}{2}\sum_{d\in D}(t_d - o_d)^2$$

Batch Learning

$$\nabla E(\vec{w}) \equiv \left[\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \ldots, \frac{\partial E}{\partial w_n}\right]$$

$$w_i \leftarrow w_i + \Delta w_i \quad where \quad \Delta w_i = -\eta \frac{\partial E}{\partial w_i}$$

Learning Rate

Delta Rule

$$\frac{\partial E}{\partial w_i} = \frac{\partial}{\partial w_i}\frac{1}{2}\sum_{d\in D}(t_d - o_d)^2$$

$$= \frac{1}{2}\sum_{d\in D}\frac{\partial}{\partial w_i}(t_d - o_d)^2$$

$$= \frac{1}{2}\sum_{d\in D}2(t_d - o_d)\frac{\partial}{\partial w_i}(t_d - o_d)$$

$$= \sum_{d\in D}(t_d - o_d)\frac{\partial}{\partial w_i}(t_d - w\cdot x_d)$$

$$= \sum_{d\in D}(t_d - o_d)(-x_{id})$$

$$\boxed{\Delta w_i = \eta\sum_{d\in D}(t_d - o_d)x_{id}}$$

$$\boxed{o(x) = w\cdot x}$$

Batch Learning

GRADIENT_DESCENT (*training_examples*, η)

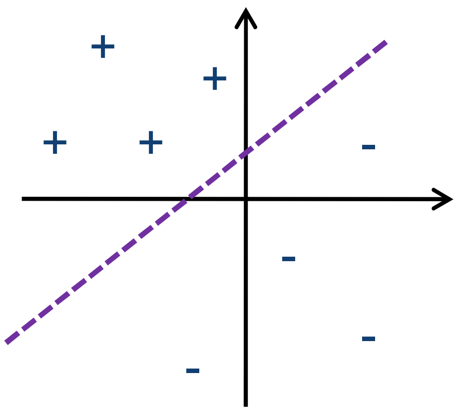
- Initialize each w_i to some small random value.
- Until the termination condition is met, Do
  - Initialize each Δw_i to zero.
  - For each <*x*, *t*> in *training_examples*, Do
    - Input the instance *x* to the unit and compute the output *o*
    - For each linear unit weight w_i, Do
      - $\Delta w_i \leftarrow \Delta w_i + \eta(t-o)x_i$
  - For each linear unit weight w_i, Do
    - $w_i \leftarrow w_i + \Delta w_i$

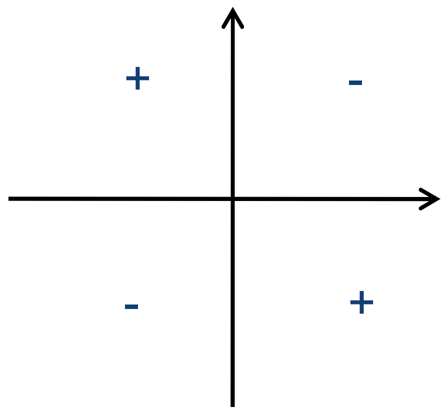
Stochastic Learning

$$w_i \leftarrow w_i + \Delta w_i \qquad where \quad \Delta w_i = \eta(t - o)x_i$$

For example, if x_i=0.8, η=0.1, t=1 and o=0

$\Delta w_i = \eta(t\text{-}o)x_i$ = 0.1×(1-0) × 0.8 = 0.08



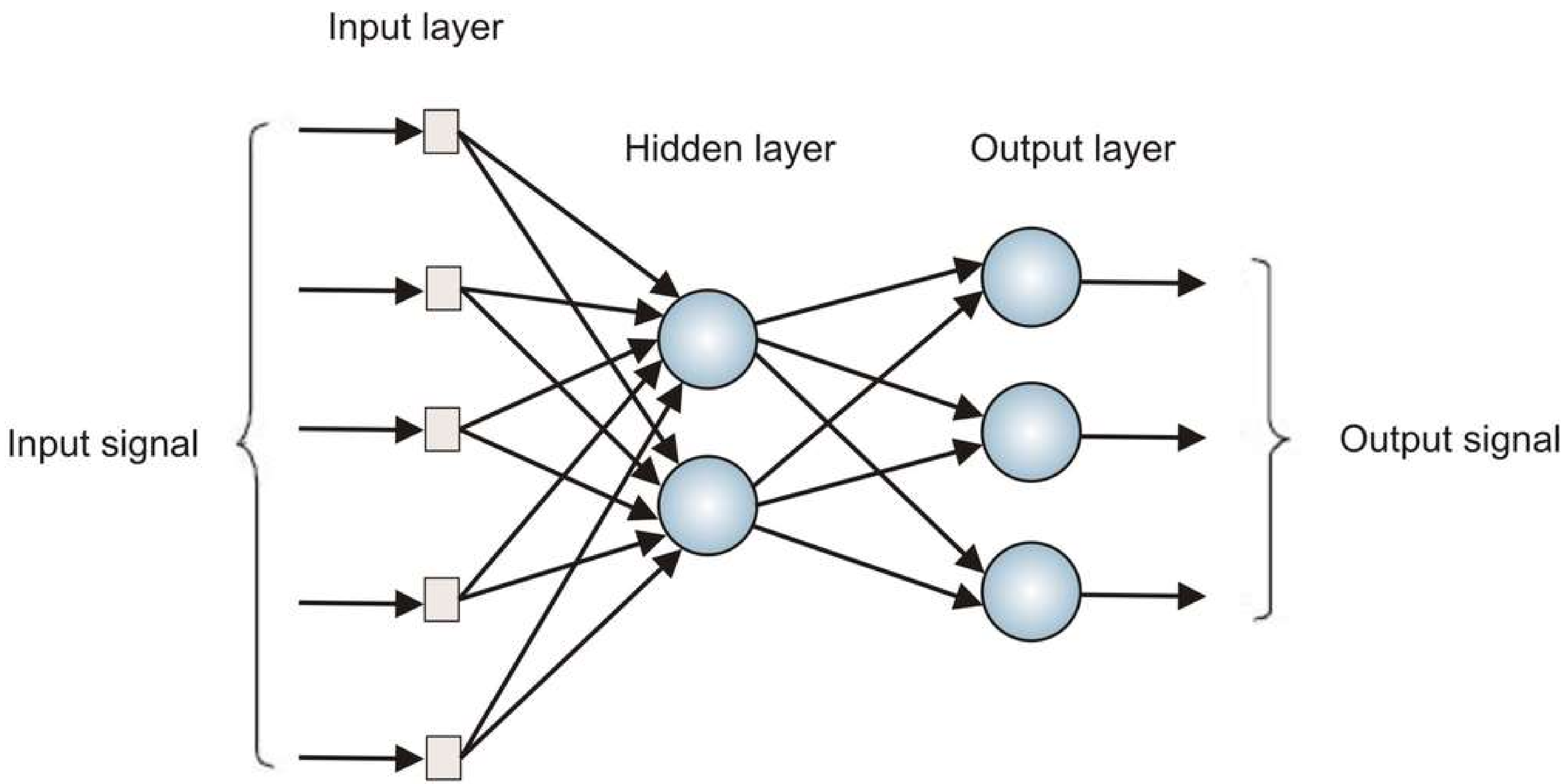


Stochastic Learning: NAND

Input			Target	Initial Weights			Output: Individual			Output: Sum	Output: Final Output	Error	Correction	Final Weights		
x_0	x_1	x_2	t	w_0	w_1	w_2	C_0	C_1	C_2	S	o	E	R	w_0	w_1	w_2
							$x_0 \cdot w_0$	$x_1 \cdot w_1$	$x_2 \cdot w_2$	$C_0+ C_1+ C_2$		t-o	LR x E			
1	0	0	1	0	0	0	0	0	0	0	0	1	+0.1	0.1	0	0
1	0	1	1	0.1	0	0	0.1	0	0	0.1	0	1	+0.1	0.2	0	0.1
1	1	0	1	0.2	0	0.1	0.2	0	0	0.2	0	1	+0.1	0.3	0.1	0.1
1	1	1	0	0.3	0.1	0.1	0.3	0.1	0.1	0.5	0	0	0	0.3	0.1	0.1
1	0	0	1	0.3	0.1	0.1	0.3	0	0	0.3	0	1	+0.1	0.4	0.1	0.1
1	0	1	1	0.4	0.1	0.1	0.4	0	0.1	0.5	0	1	+0.1	0.5	0.1	0.2
1	1	0	1	0.5	0.1	0.2	0.5	0.1	0	0.6	1	0	0	0.5	0.1	0.2
1	1	1	0	0.5	0.1	0.2	0.5	0.1	0.2	0.8	1	-1	-0.1	0.4	0	0.1
1	0	0	1	0.4	0	0.1	0.4	0	0	0.4	0	1	+0.1	0.5	0	0.1
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
1	1	0	1	0.8	-.2	-.1	0.8	-.2	0	0.6	1	0	0	0.8	-.2	-.1

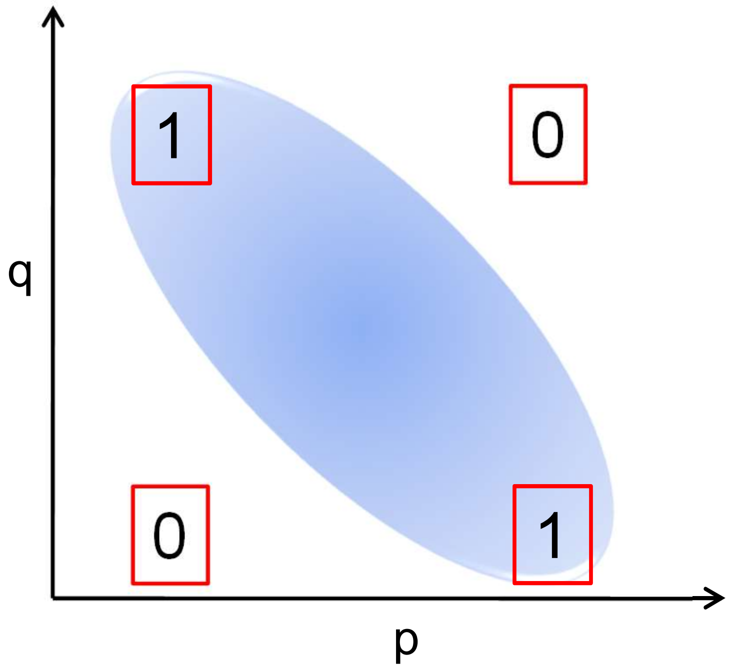
threshold=0.5 learning rate=0.1

Multilayer Perceptron



XOR

$$p \oplus q = p\overline{q} + \overline{p}q = (p+q)(\overline{pq})$$

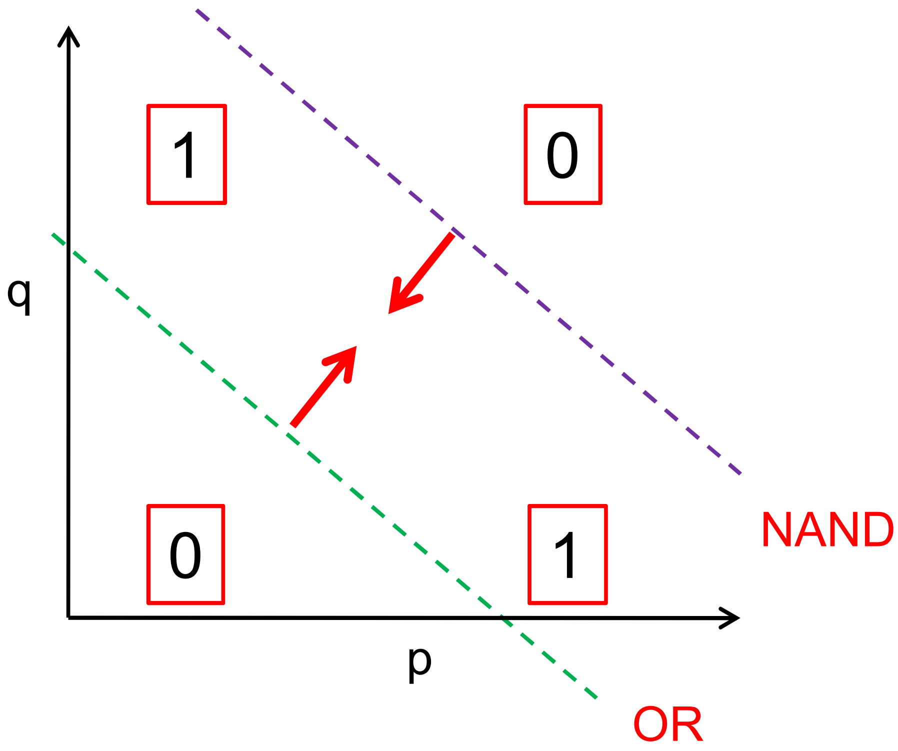


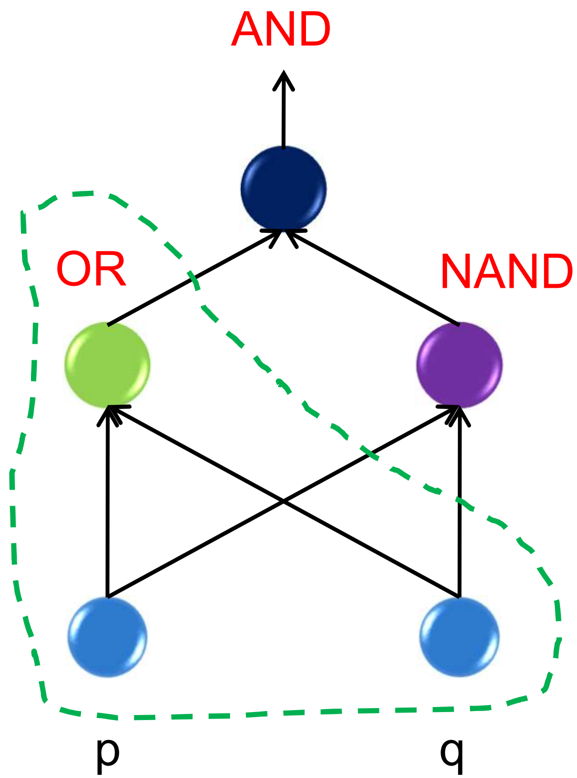
Input		Output
0	0	0
0	1	1
1	0	1
1	1	0

Cannot be separated by a single line.

XOR

$$p \oplus q = \neg(p \wedge q) \wedge (p \vee q)$$

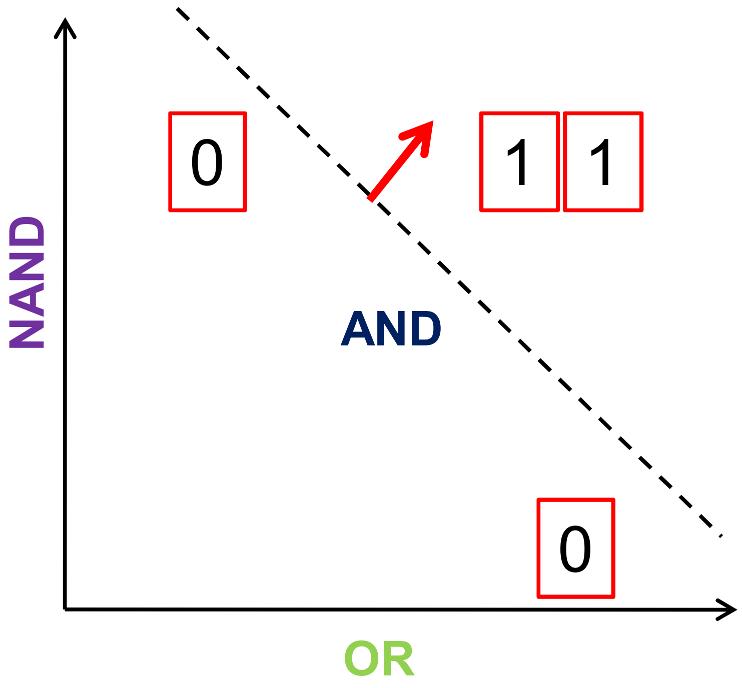


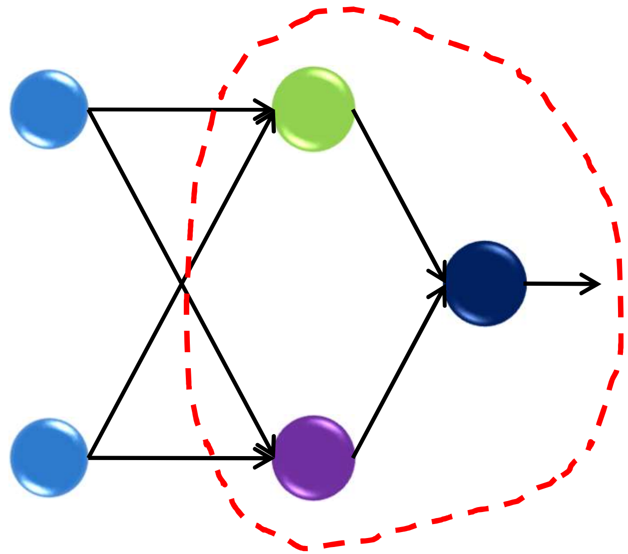


Hidden Layer Representations

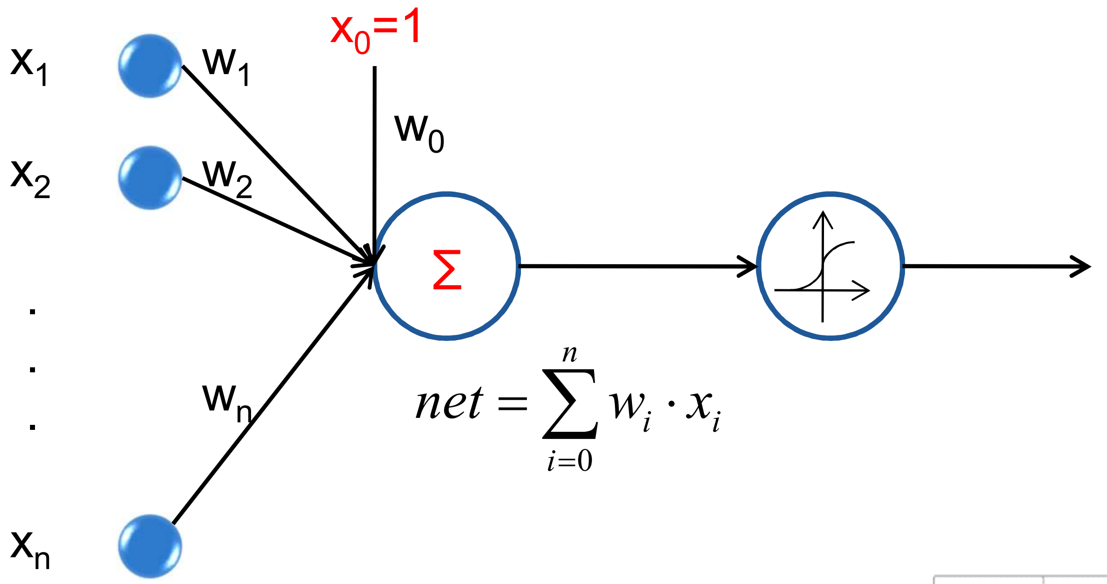
p	q	OR	NAND	AND
0	0	0	1	0
0	1	1	1	1
1	0	1	1	1
1	1	1	0	0

Input (p, q) — Hidden (OR, NAND) — Output (AND)



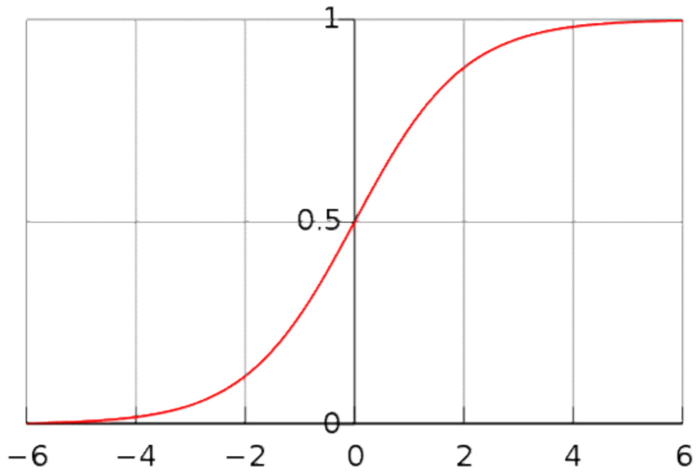


The Sigmoid Threshold Unit



$$o = \sigma(net) = \frac{1}{1+e^{-net}}$$

$$net = \sum_{i=0}^{n} w_i \cdot x_i$$



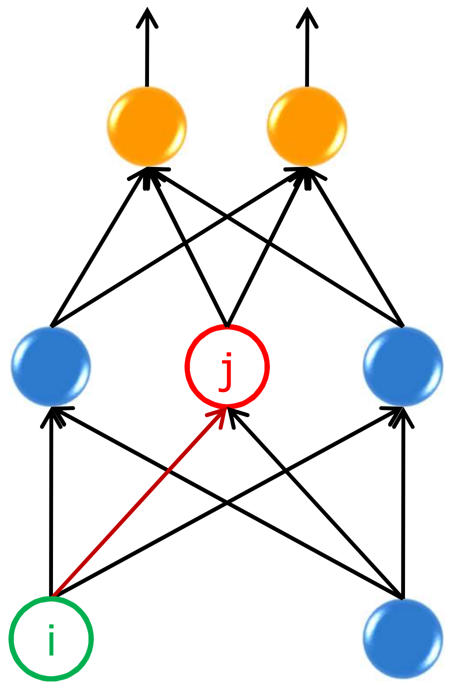
Sigmoid Function

$$\sigma(y) = \frac{1}{1+e^{-y}} \qquad \frac{d\sigma(y)}{dy} = \sigma(y)\cdot(1-\sigma(y))$$

Backpropagation Rule

$$E_d(\overline{w}) \equiv \frac{1}{2}\sum_{k \in outputs}(t_k - o_k)^2 \qquad \Delta w_{ji} = -\eta \frac{\partial E_d}{\partial w_{ji}}$$

$$\frac{\partial E_d}{\partial w_{ji}} = \frac{\partial E_d}{\partial net_j} \cdot \frac{\partial net_j}{\partial w_{ji}} = \frac{\partial E_d}{\partial net_j} x_{ji}$$



- x_{ji} = the i th input to unit j
- w_{ji} = the weight associated with the i th input to unit j
- $net_j = \sum w_{ji} x_{ji}$ (the weighted sum of inputs for unit j)
- o_j= the output of unit j
- t_j= the target output of unit j
- σ = the sigmoid function
- *outputs* = the set of units in the final layer
- *Downstream* (j) = the set of units directly taking the output of unit j as inputs

Training Rule for Output Units

$$\frac{\partial E_d}{\partial net_j} = \frac{\partial E_d}{\partial o_j} \cdot \frac{\partial o_j}{\partial net_j}$$

$$\frac{\partial E_d}{\partial o_j} = \frac{\partial}{\partial o_j} \frac{1}{2} \sum_{k \in outputs} (t_k - o_k)^2$$

$$\frac{\partial E_d}{\partial o_j} = \frac{\partial}{\partial o_j} \frac{1}{2} (t_j - o_j)^2$$

$$= \frac{1}{2} 2(t_j - o_j) \frac{\partial (t_j - o_j)}{\partial o_j}$$

$$= -(t_j - o_j)$$

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

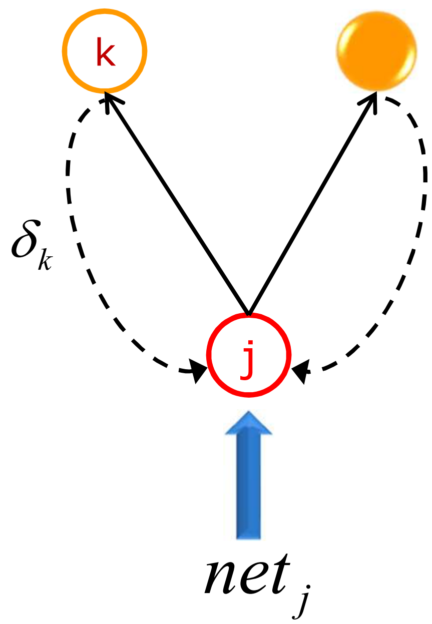
$$\frac{\partial E_d}{\partial net_j} = -(t_j - o_j) o_j (1 - o_j)$$

$$\boxed{\Delta w_{ji} = -\eta \frac{\partial E_d}{\partial w_{ji}} = \eta (t_j - o_j) o_j (1 - o_j) x_{ji}}$$

Training Rule for Hidden Units

$$\frac{\partial E_d}{\partial net_j} = \sum_{k\in Downstream(j)} \frac{\partial E_d}{\partial net_k}\frac{\partial net_k}{\partial net_j} = \sum_{k\in Downstream(j)} -\delta_k \frac{\partial net_k}{\partial o_j}\frac{\partial o_j}{\partial net_j}$$

$$= \sum_{k\in Downstream(j)} -\delta_k w_{kj}\frac{\partial o_j}{\partial net_j} = \sum_{k\in Downstream(j)} -\delta_k w_{kj} o_j(1-o_j)$$



$$\delta_k = -\frac{\partial E_d}{\partial net_k}$$

$$\delta_j = o_j(1-o_j)\sum_{k\in Downstream(j)} \delta_k w_{kj}$$

$$\boxed{\Delta w_{ji} = \eta \delta_j x_{ji}}$$

BP Framework

- BACKPROPAGATION (*training_examples*, η, n_{in}, n_{out}, n_{hidden})
- Create a network with n_{in} inputs, n_{hidden} hidden units and n_{out} output units.
- Initialize all network weights to small random numbers.
- Until the termination condition is met, Do
  - For each <*x*, *t*> in *training_examples*, Do
    - Input the instance *x* to the network and computer the output *o* of every unit.
    - For each output unit *k*, calculate its error term δ_k

$$\delta_k \leftarrow o_k(1-o_k)(t_k-o_k)$$

    - For each hidden unit *h*, calculate its error term δ_h

$$\delta_h \leftarrow o_h(1-o_h)\sum_{k \in outputs} w_{kh}\delta_k$$

    - Update each network weight w_{ji}

$$w_{ji} \leftarrow w_{ji} + \Delta w_{ji} = w_{ji} + \eta\delta_j x_{ji}$$

More about BP Networks …

- Convergence and Local Minima
  - The search space is likely to be highly multimodal.
  - May easily get stuck at a local solution.
  - Need multiple trials with different initial weights.

- Evolving Neural Networks
  - Black-box optimization techniques (e.g., Genetic Algorithms)
  - Usually better accuracy
  - Can do some advanced training (e.g., structure + parameter).
  - Xin Yao (1999) "Evolving Artificial Neural Networks", *Proceedings of the IEEE*, pp. 1423-1447.

- Representational Power

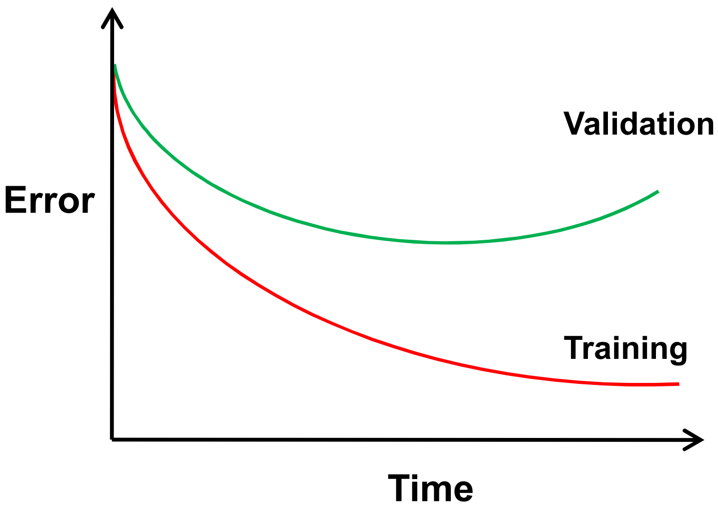
- Deep Learning

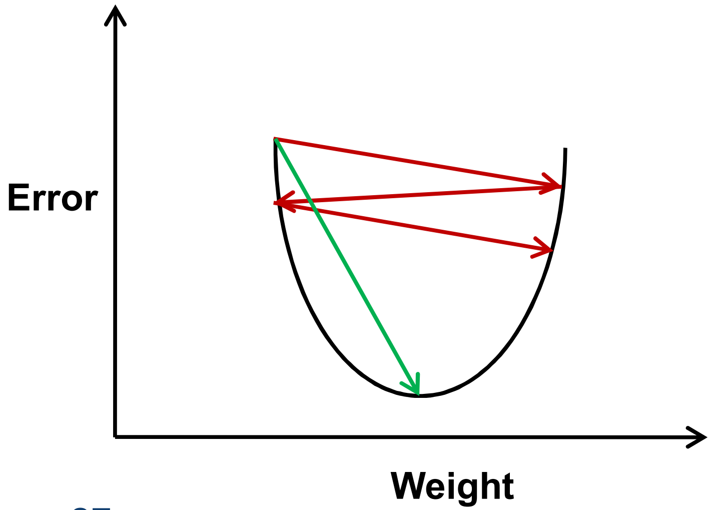
More about BP Networks ...

- Overfitting
  - Tend to occur during later iterations.
  - Use validation dataset to terminate the training when necessary.

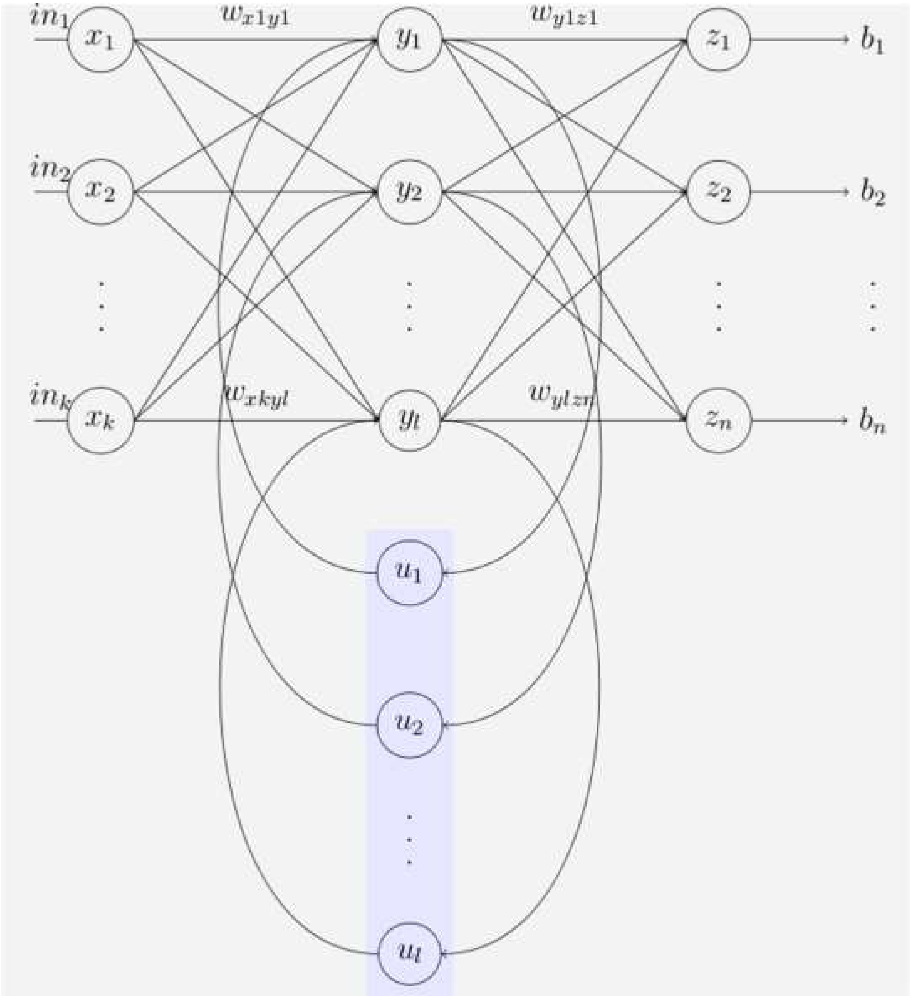
- Practical Considerations
  - Momentum
  - Adaptive learning rate
    - Small: slow convergence, easy to get stuck
    - Large: fast convergence, unstable

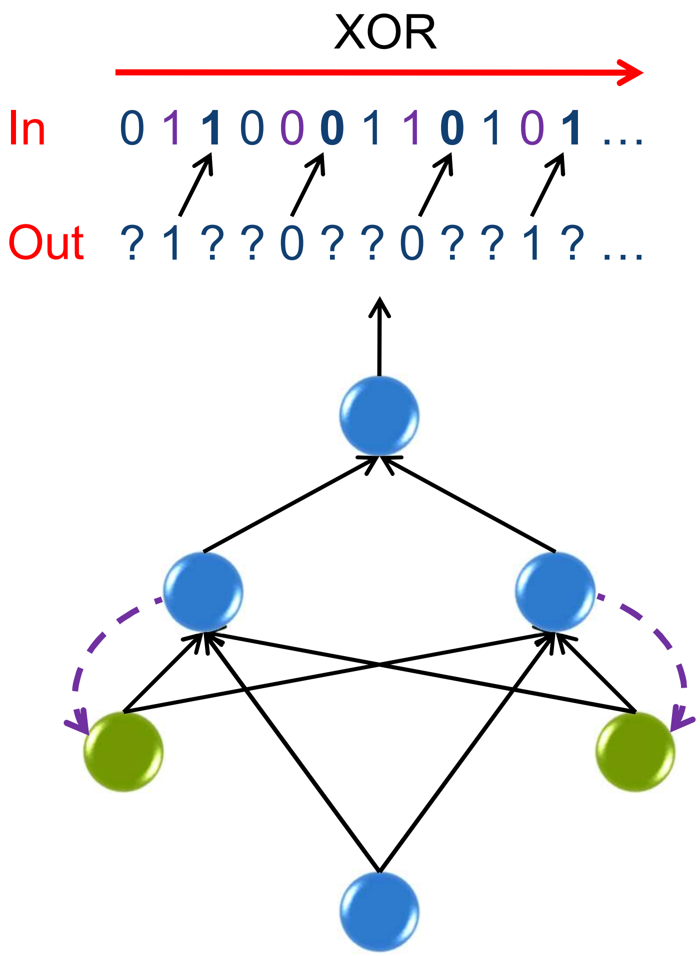
$$\boxed{\Delta w_{ji}(n) = \eta \delta_j x_{ji} + \alpha \Delta w_{ji}(n-1)}$$





Beyond BP Networks



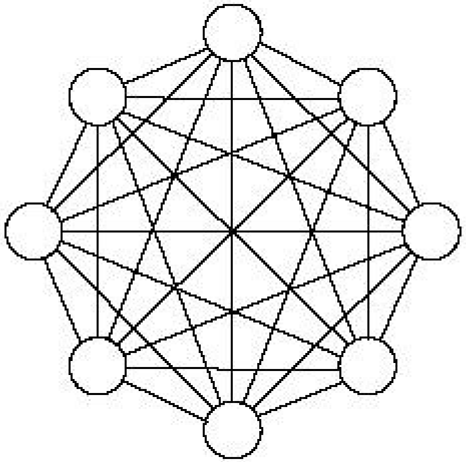


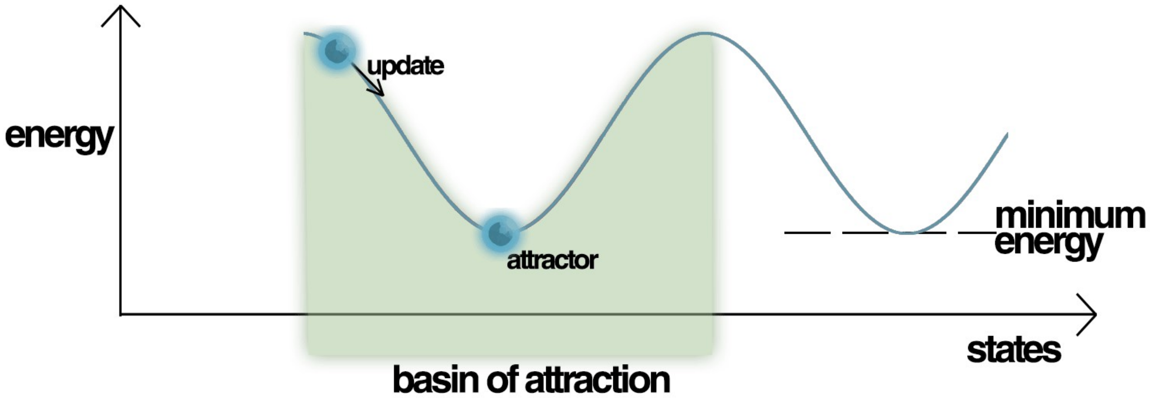
Elman Network

Beyond BP Networks

- $w_{ii} = 0, \forall i$ (no unit has a connection with itself)
- $w_{ij} = w_{ji}, \forall i, j$ (connections are symmetric)

$$E = -\frac{1}{2}\sum_{i,j} w_{ij} s_i s_j + \sum_i \theta_i \ s_i$$

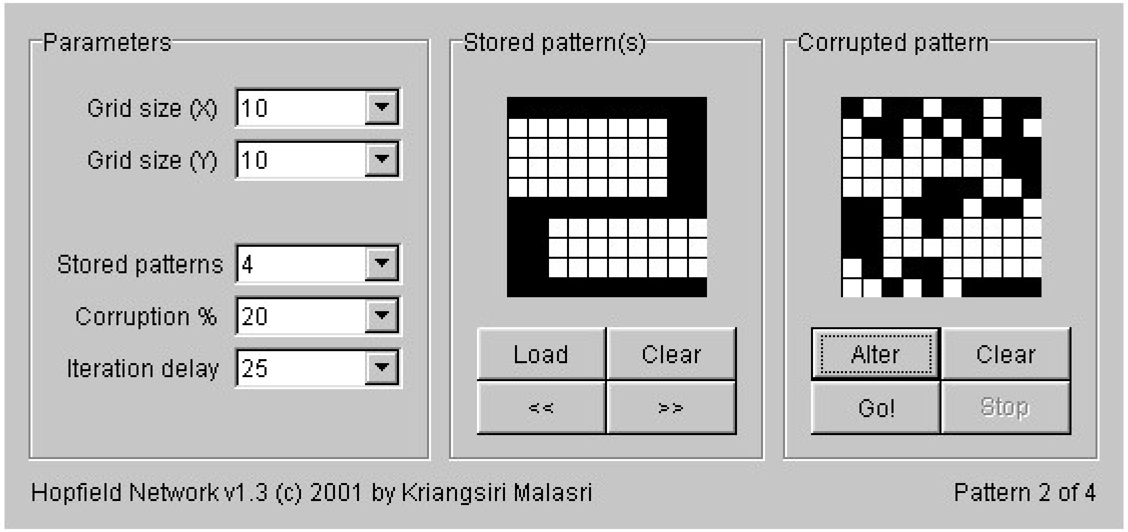


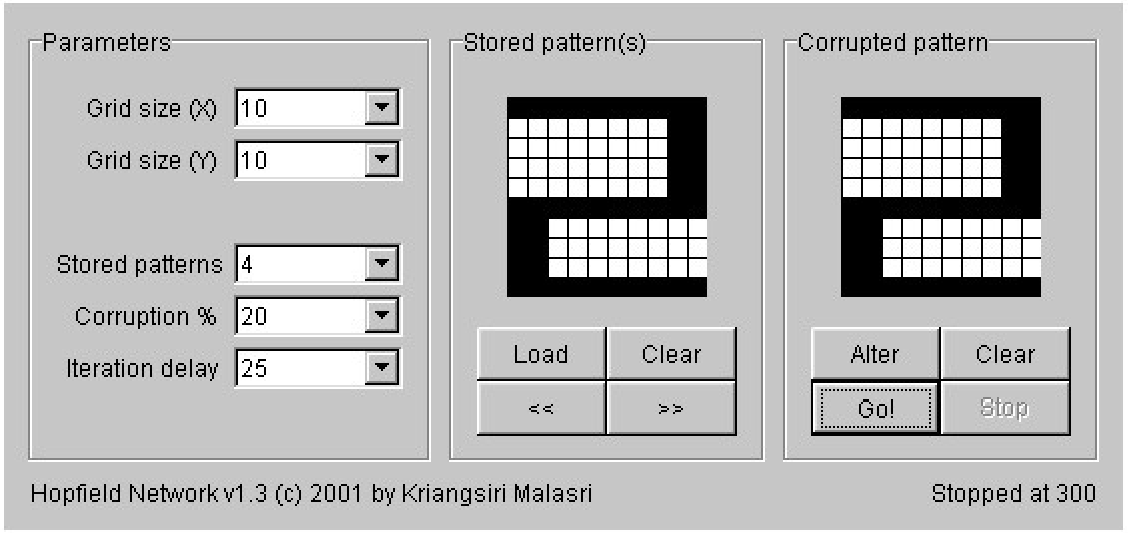


Hopfield Network

Energy Landscape of Hopfield Network

Beyond BP Networks





When does ANN work?

- Instances are represented by attribute-value pairs.
  - Input values can be any real values.
- The target output may be discrete-valued, real-valued, or a vector of several real- or discrete-valued attributes.
- The training samples may contain errors.
- Long training times are acceptable.
  - Can range from a few seconds to several hours.
- Fast evaluation of the learned target function may be required.
- The ability to understand the learned function is not important.
  - Weights are difficult for humans to interpret.

Reading Materials

- Text Book
  - R. O. Duda et al., *Pattern Classification,* Chapter 6, John Wiley & Sons Inc.
  - Tom Mitchell, *Machine Learning,* Chapter 4, McGraw-Hill.
  - http://page.mi.fu-berlin.de/rojas/neural/index.html.html
- Online Demo
  - http://neuron.eng.wayne.edu/software.html
  - http://facstaff.cbu.edu/~pong/ai/hopfield/hopfieldapplet.html
- Online Tutorial
  - http://www.autonlab.org/tutorials/neural13.pdf
  - http://www.cs.cmu.edu/afs/cs.cmu.edu/user/mitchell/ftp/faces.html