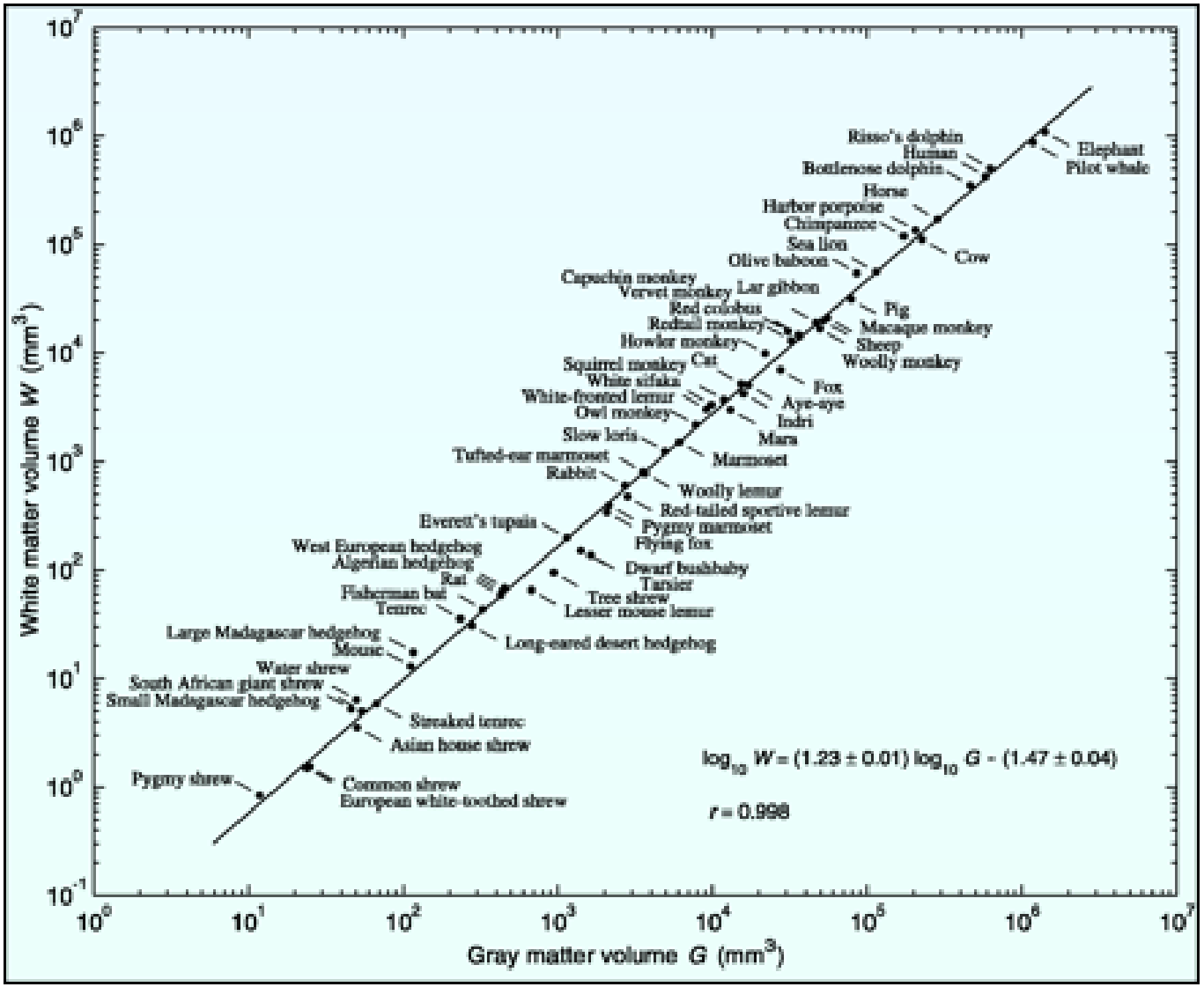
Feed-Forward mapping networks

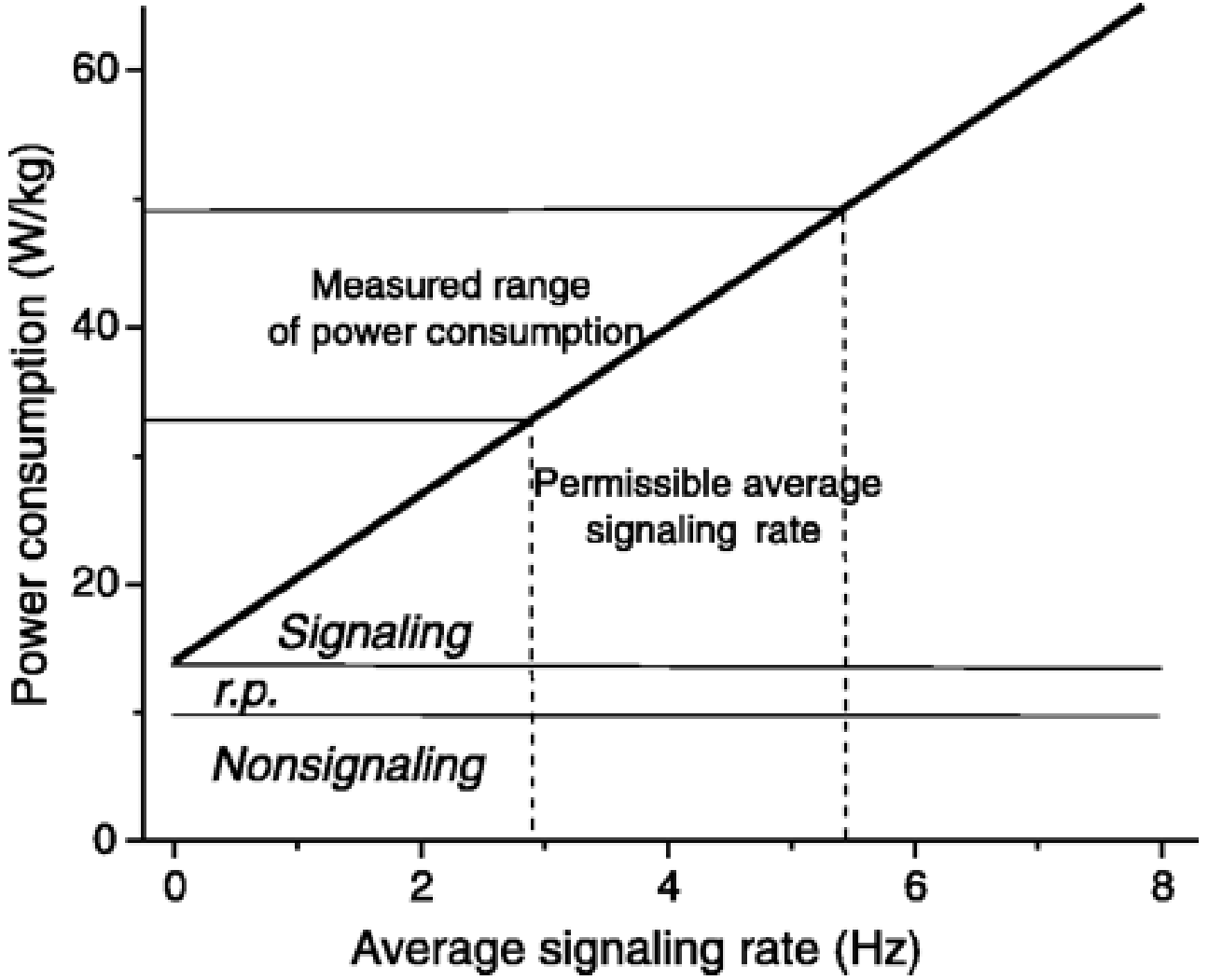
KAIST
바이오및뇌공학과
정재승

How much energy do we need for brain functions?

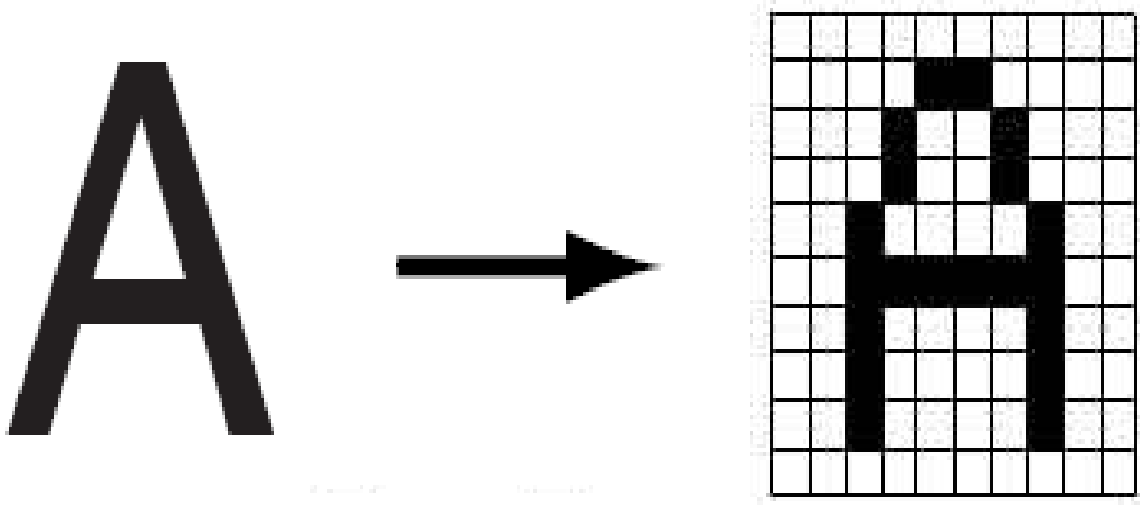
Information processing: Trade-off between energy consumption and wiring cost



Trade-off between energy consumption (wiring cost) and maximal information processing

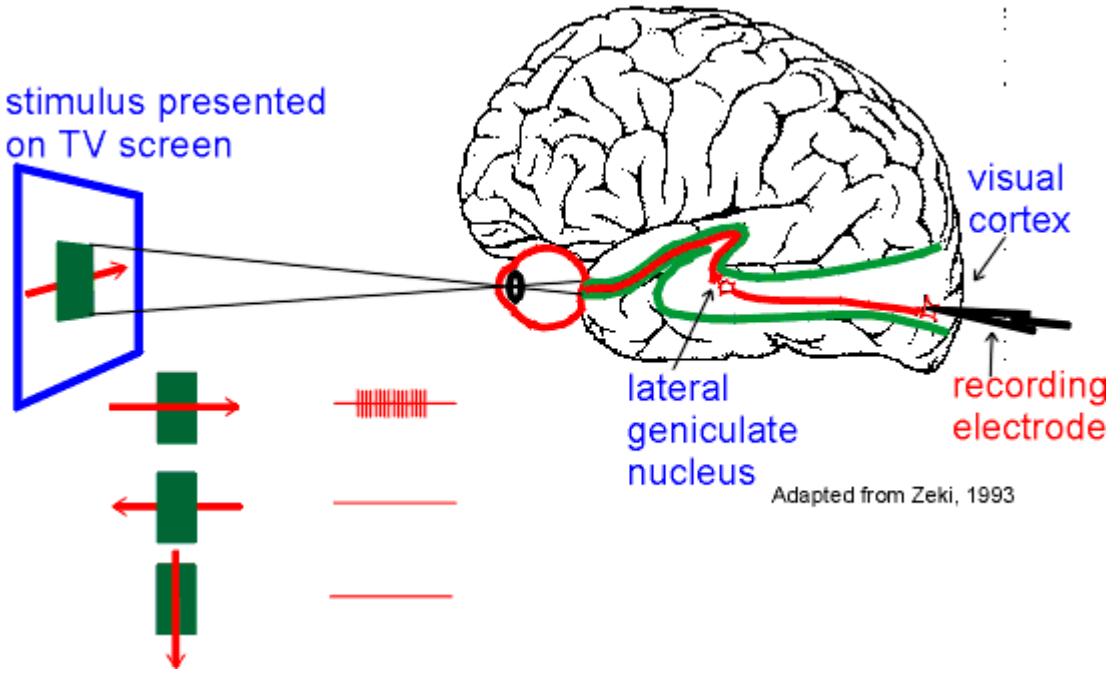


Optical character recognition (OCR)



- OCR is the process of optically scanning an image and interpreting this digital image so that the computer understands its meaning.

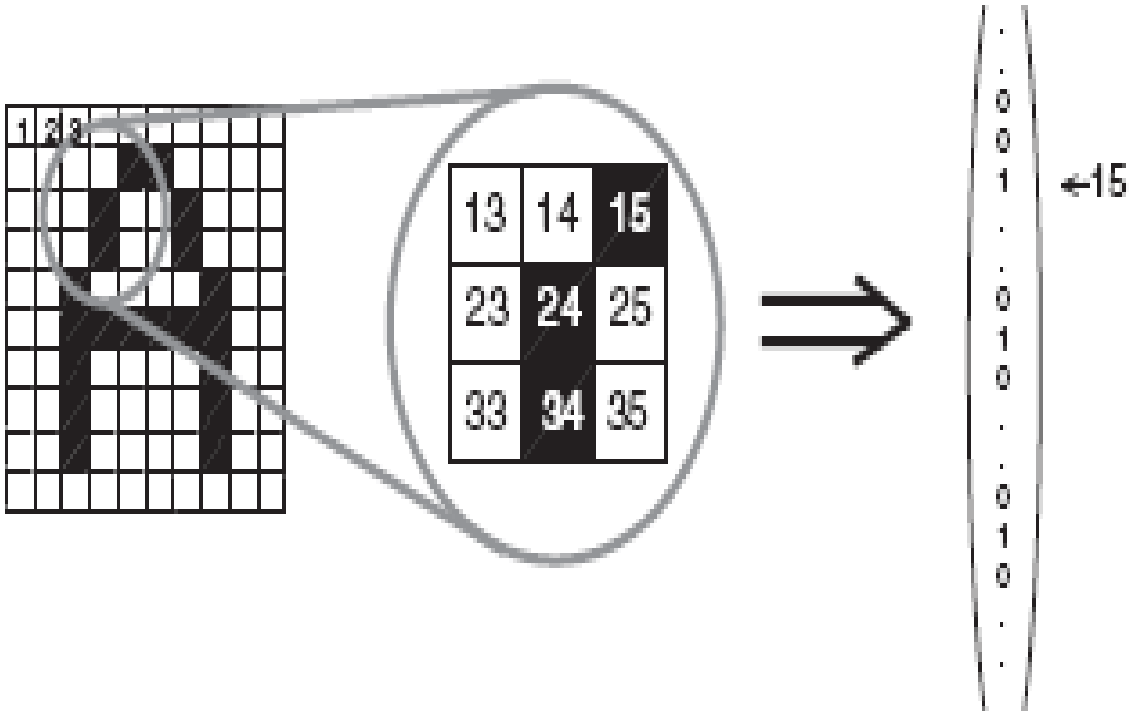
Two feed-forward process



- The perception of a letter, the physical sensing of an image of a letter.

- Process attaching meaning to such an image.

The perception of a letter



- Sensory feature vector: the number of feature values defines the dimensionality of the feature space.

Mapping functions:
the recognition process to a vector function

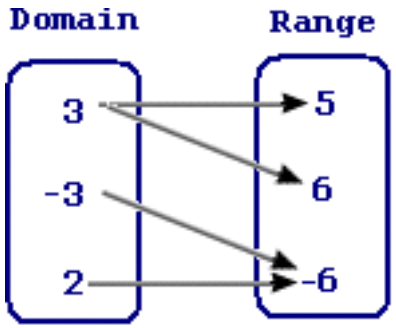


Figure 1

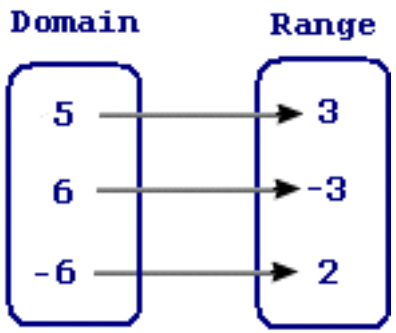


Figure 2

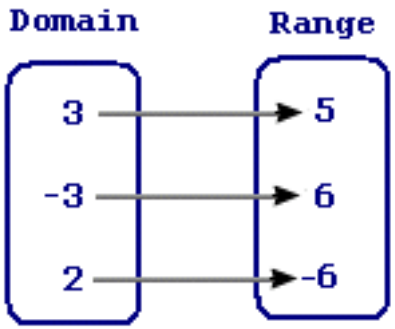


Figure 3

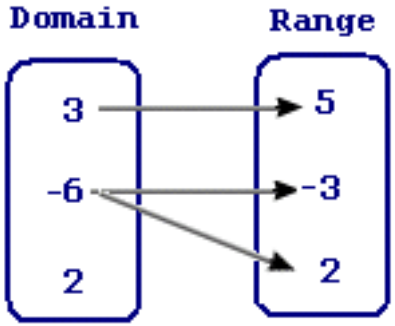
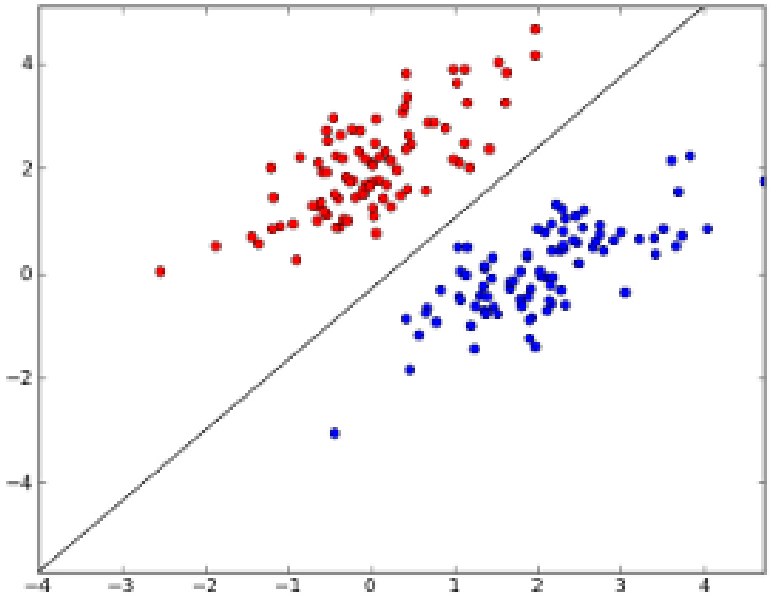


Figure 4

$$f : \mathbf{x} \in \mathbf{S}_1^n \rightarrow \mathbf{y} \in \mathbf{S}_2^m, \qquad (6.1)$$

Perceptron



- The **perceptron** is a type of artificial neural network invented in 1957 at the Cornell Aeronautical Laboratory by Frank Rosenblatt. It can be seen as the simplest kind of feed-forward neural network: a linear classifier.

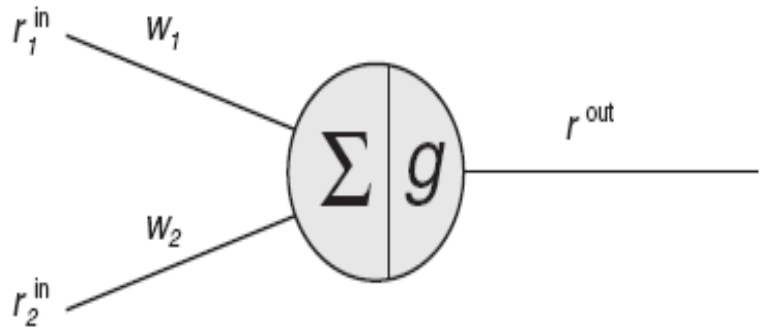
Population node as perceptron

$$r_i^{\text{in}} = x_i. \tag{6.2}$$

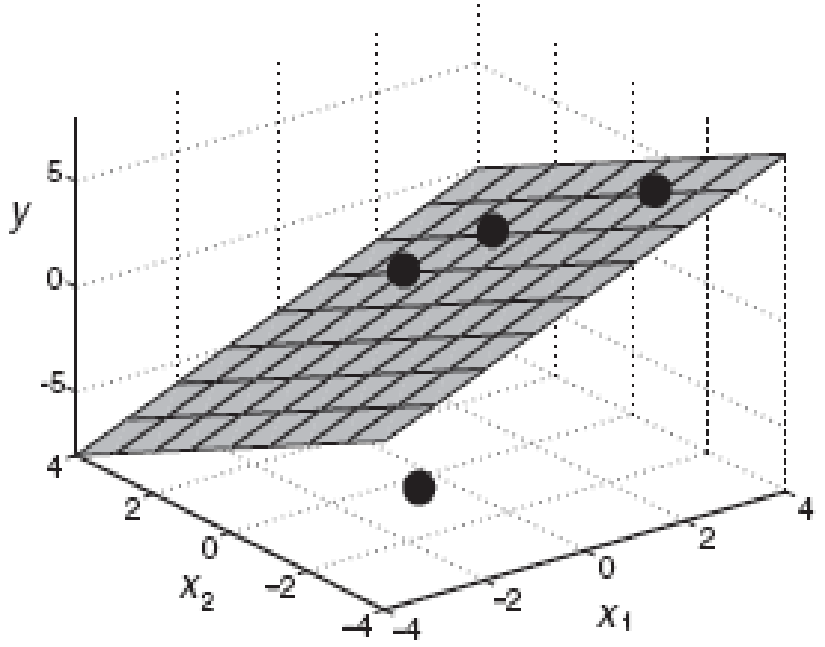
$$\tilde{y} = r^{\text{out}}. \tag{6.3}$$

$$\tilde{y} = w_1 x_1 + w_2 x_2, \tag{6.4}$$

Linear perceptron



Output manifold of population node with two input channels



$$\tilde{y}^1 = \tilde{y}(x^1) = \tilde{y}(x_1 = 1, x_2 = 2) = 1 \cdot 1 - 1 \cdot 2 = -1 = y^1. \qquad (6.5)$$

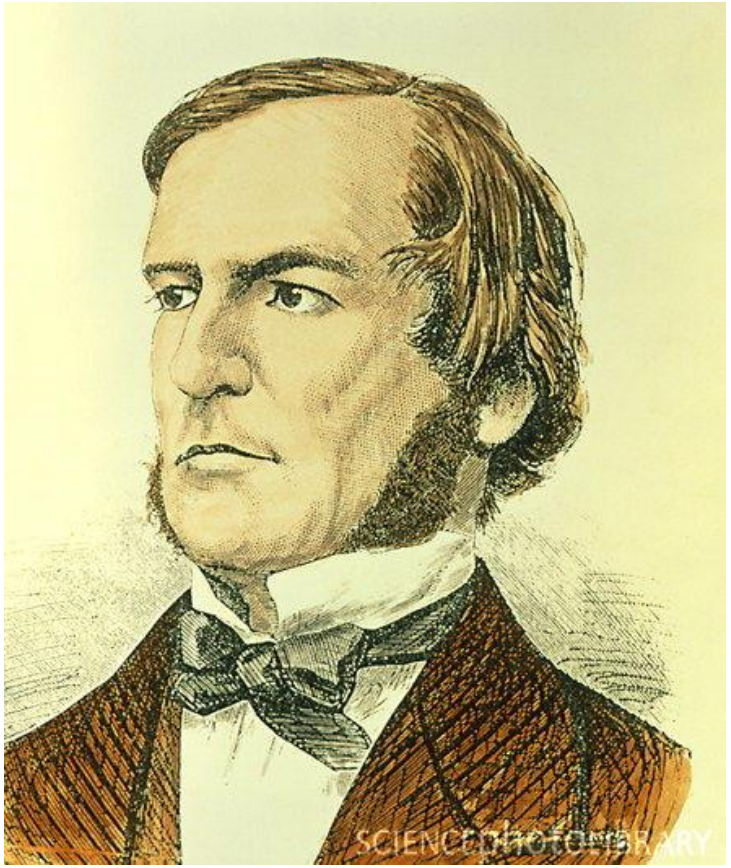
$$\tilde{y}^2 = 1 \cdot 2 - 1 \cdot 1 = 1 = y^2. \qquad (6.6)$$

$$\tilde{y}^3 = 1 \cdot 3 - 1 \cdot (-2) = 5 = y^3. \qquad (6.7)$$

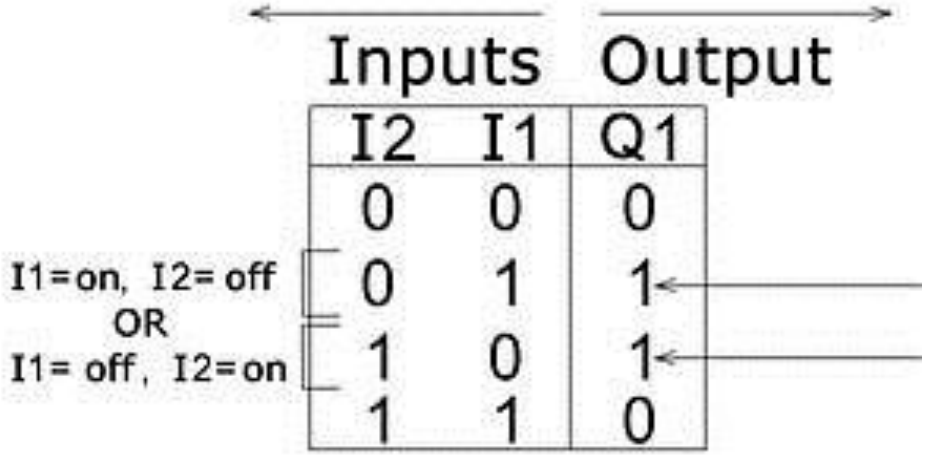
Boolean algebra

- **Boolean algebra** (or **Boolean logic**) is a logical calculus of truth values, developed by George Boole in the 1840s.

- These turn out to coincide with the set of all operations on the set {0,1} that take only finitely many arguments; there are 2^{2^n} such operations when there are n arguments.



Boolean algebra



- It resembles the algebra of real numbers, but with the numeric operations of multiplication xy, addition $x + y$, and negation $-x$ replaced by the respective logical operations of conjunction $x \wedge y$, disjunction $x \vee y$, and negation $\neg x$.

- The Boolean operations are these and all other operations that can be built from these, such as $x \wedge (y \vee z)$.

Boolean functions:
the threshold node

Table 6.1 (A) Look-up table for a specific binary mapping function (Boolean AND function) in a two-dimensional feature space. (B) Partial look-up table for a sample function in a two-dimensional feature space with discrete but not binary feature values.

A. Boolean AND function

x_1	x_2	y
0	0	0
0	1	0
1	0	0
1	1	1

B. Non-Boolean function

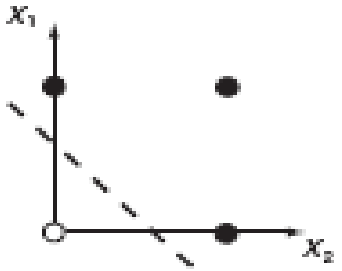
x_1	x_2	y
1	2	-1
2	1	1
3	-2	5
-1	-1	7
⋮	⋮	⋮

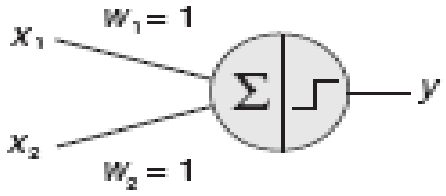
Look-up table, graphical representation, and single threshold population nodes

A.

Boolean OR function

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	1

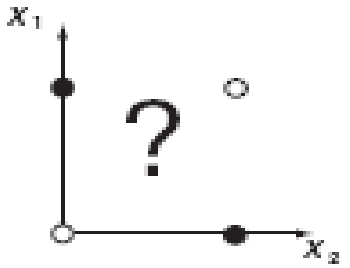


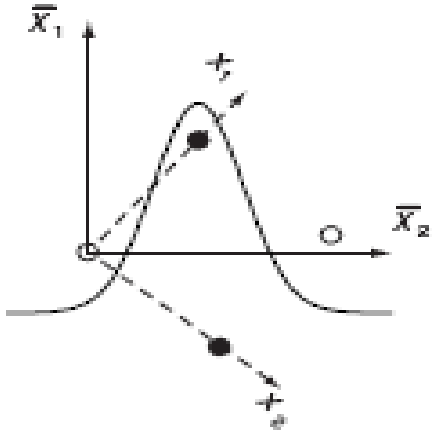


B.

Boolean XOR function

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0



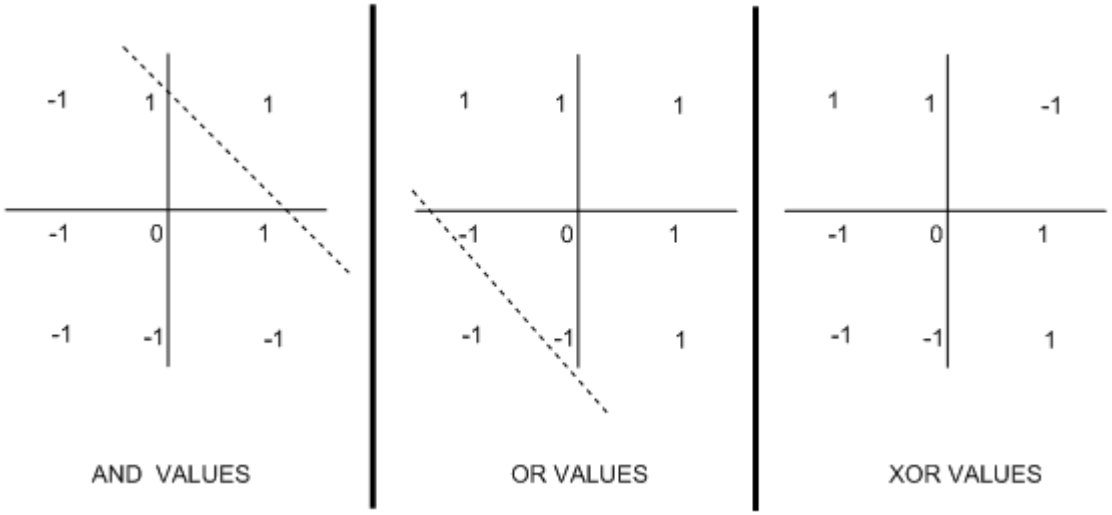


The history of perceptron

- A feed-forward neural network with two or more layers (i.e., a multilayer perceptron) had far greater processing power than perceptrons with one layer (i.e., a single layer perceptron).

- Single layer perceptrons are only capable of learning linearly separable patterns; in 1969 a famous book entitled *Perceptrons* by Marvin Minsky and Seymour Papert showed that it was not possible for these classes of network to learn an XOR function.

The history of perceptron



- Both Minsky and Papert already knew that multi-layer perceptrons were capable of producing an XOR Function.

- Three years later, Stephen Grossberg published a series of papers introducing networks capable of modelling differential, contrast-enhancing and XOR functions.

The history of perceptron

- Nevertheless the often-miscited Minsky/Papert text caused a significant decline in interest and funding of neural network research.

- It took ten more years until neural network research experienced a resurgence in the 1980s.

- This text was reprinted in 1987 as "Perceptrons - Expanded Edition" where some errors in the original text are shown and corrected.

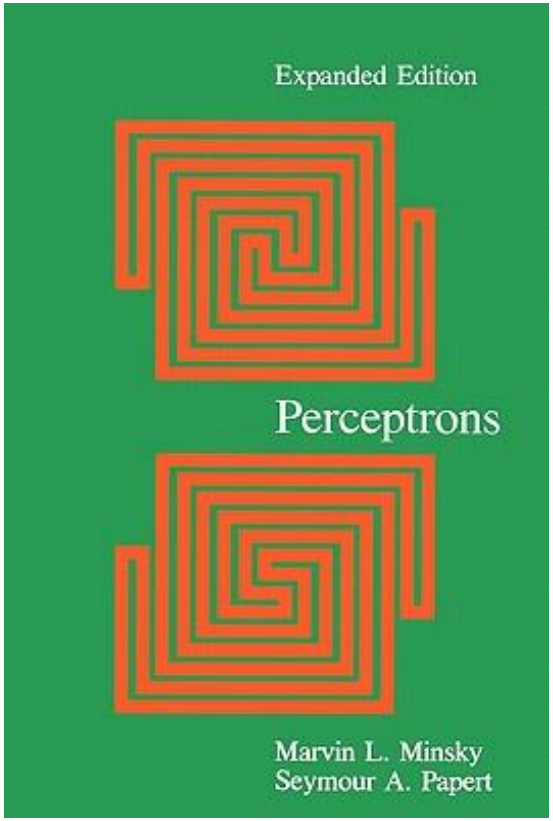
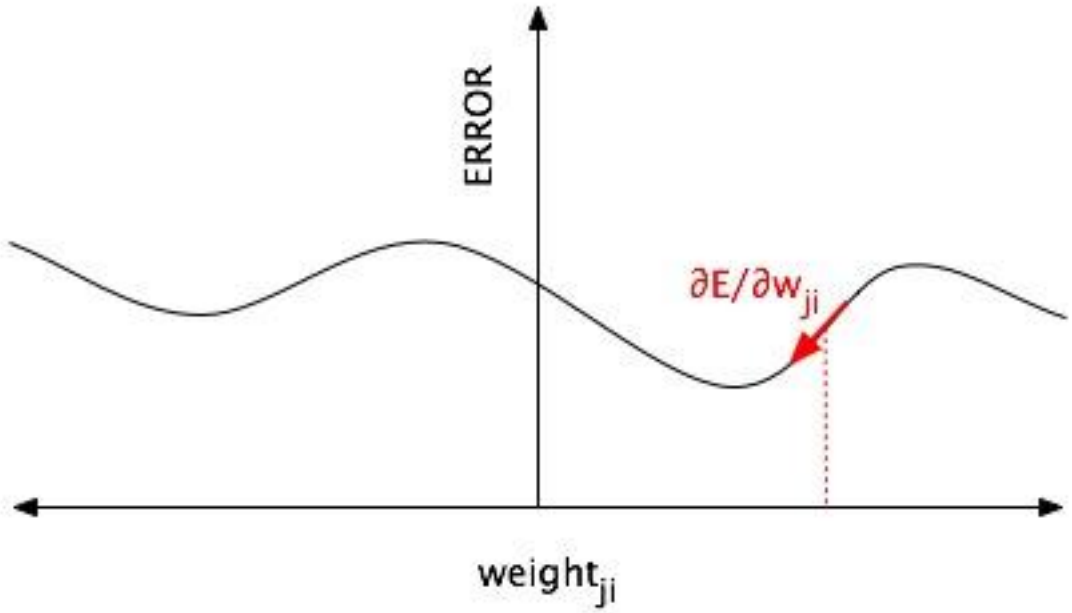


Table 6.2 Number of Boolean functions in an n-dimensional feature space and the number of linear non-separable functions.

n	Number of linear separable functions	Number of linear non-separable functions
2	14	2
3	104	151
4	1,882	63654
5	94,572	$\sim 4.310^{9}$
6	15,028,134	$\sim 1.810^{19}$

- **The number of nonlinear separable functions grows rapidly with the dimension of the feature space and soon outgrows the number of linear separable functions**

Learning delta rule



- The delta rule is a gradient descent learning rule for updating the weights of the artificial neurons in a single-layer perceptron.

- The weight matrix will be changed by small amounts in an attempt to find a better answer.

Supervised learning

- Given examples

$$R^N \rightarrow \{0,1\}$$

$$x_1 \rightarrow y_1$$

$$x_2 \rightarrow y_2$$

$$x_3 \rightarrow y_3$$

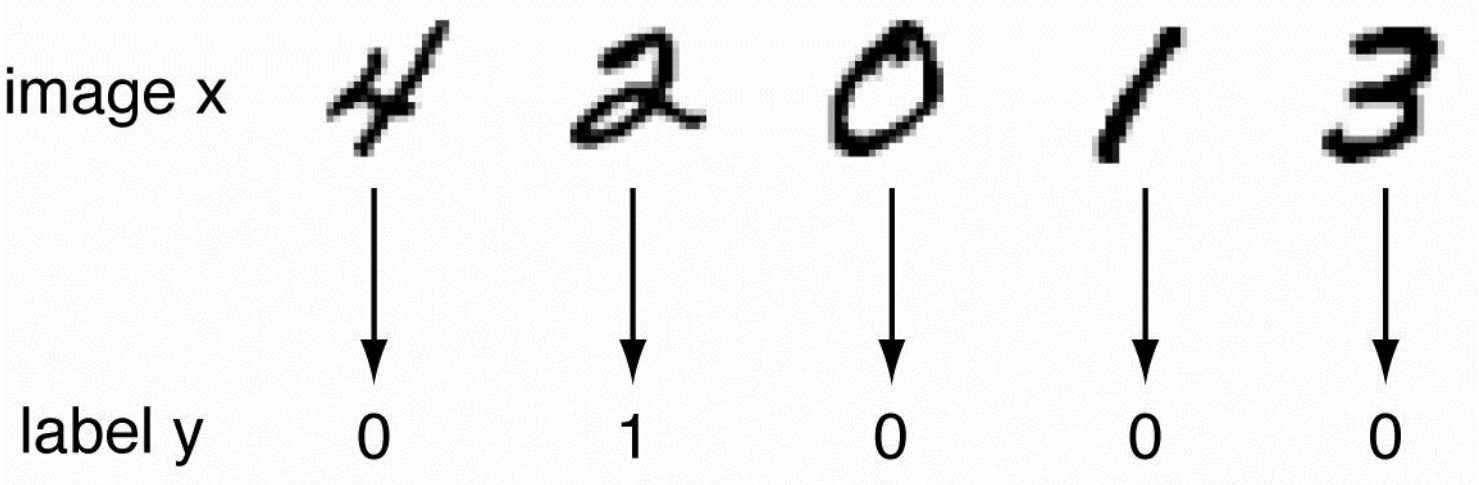
$$\vdots$$

- Find perceptron such that

$$y_a = H\left(w^T x_a\right)$$

Example: handwritten digits

- **Find a perceptron that detects "two"s.**



Delta rule

$$\Delta w = \eta\left[y - H\left(w^T x\right)\right]x$$

- **Learning from mistakes.**
- **"delta": difference between desired and actual output.**
- **Also called "perceptron learning rule"**

Two types of mistakes

- **False positive**
  - **Make *w* less like *x*.** $y = 0, \quad H\left(w^T x\right) = 1$
- **False negative** $\Delta w = -\eta x$
  - **Make *w* more like *x*.**

$$y = 1, \quad H\left(w^T x\right) = 0$$

$$\Delta w = \eta x$$

- **The update is always proportional to *x*.**

Objective function

- **Gradient update**

$$\Delta w = -\eta \frac{\partial e}{\partial w} \qquad e(w,x,y) = \left| y - H\left(w^T x\right) \right| w^T x$$

- **Stochastic gradient descent on**

$$E(w) = \left\langle e(w,x,y) \right\rangle$$

- ***E*=0 means no mistakes.**

If examples are nonseparable

- The delta rule does not converge.
- Objective function is not equal to the number of mistakes.
- No reason to believe that the delta rule minimizes the number of mistakes.

Contrast with Hebb rule

$$\Delta w = \eta y x$$

Hebb rule

$$\Delta w = \eta \left(y - \langle y \rangle \right) x$$

Perceptron learning rule

- Assume that the teacher can drive the perceptron to produce the desired output.
- What are the objective functions?

Is the delta rule biological?

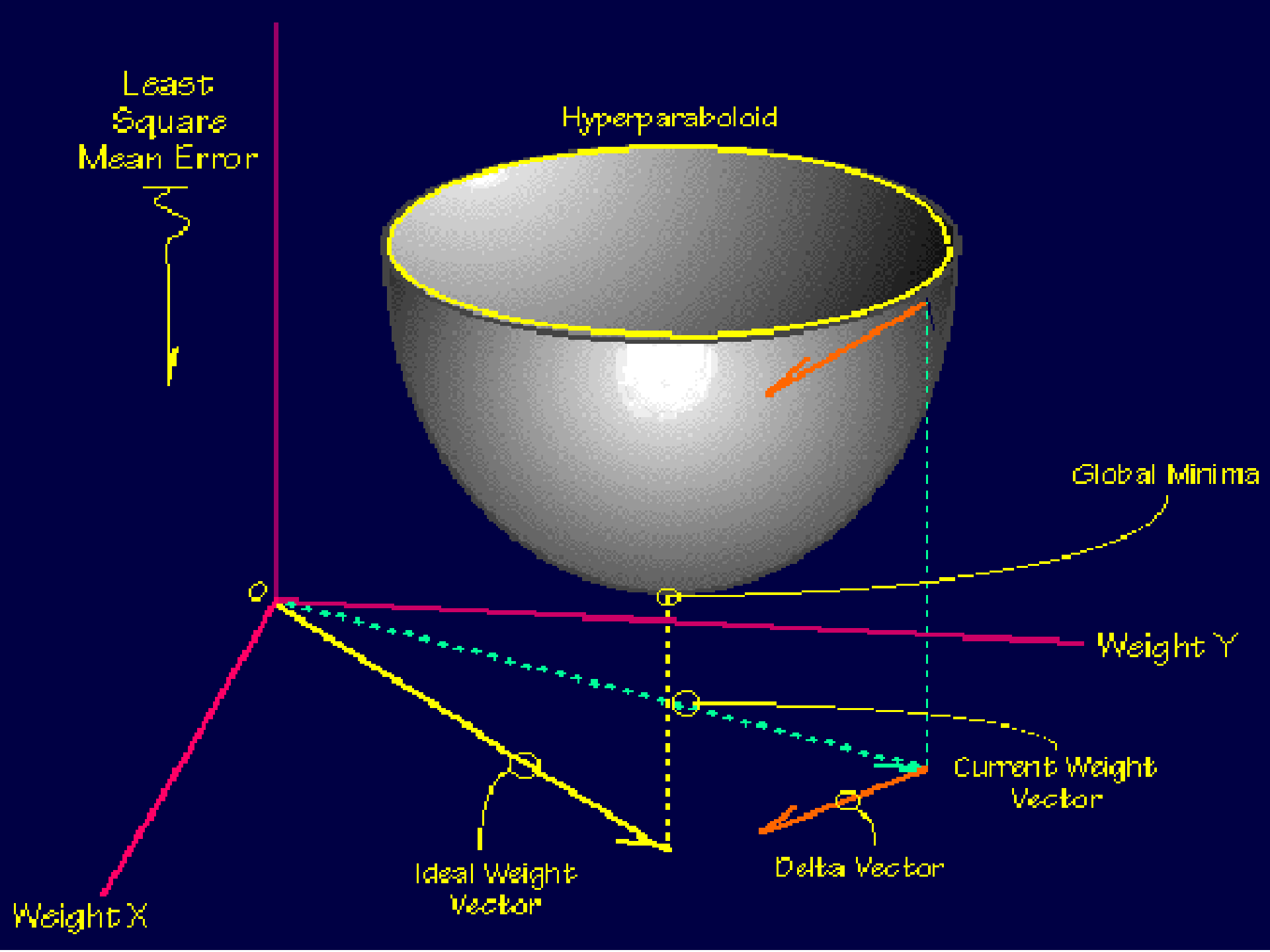
- **Actual output: anti-Hebbian**

$$\Delta w = -\eta H\left(w^T x\right)x$$

- **Desired output: Hebbian**

$$\Delta w = \eta y x$$

- **Contrastive**

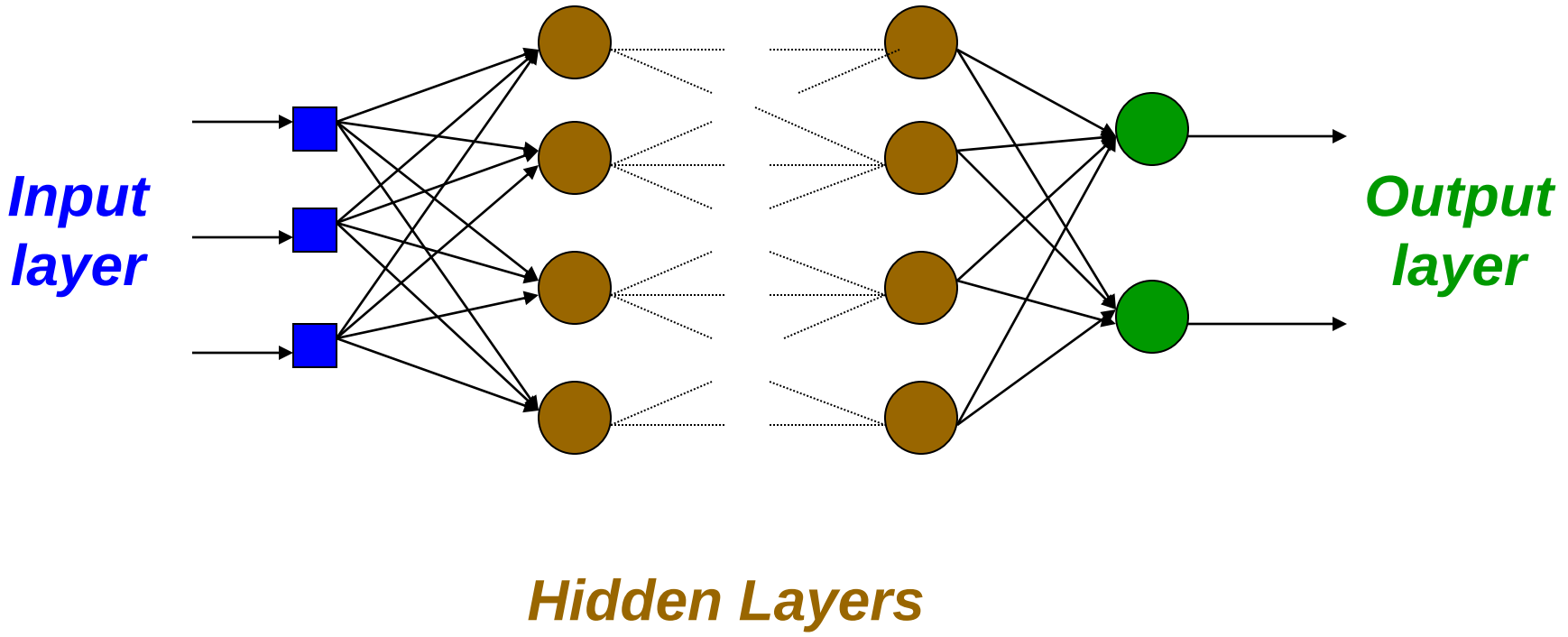

Least
Square
Mean Error
Hyperparaboloid
Global Minima
Weight Y
Weight X
Ideal Weight
Vector
Delta Vector
Current Weight
Vector

Objective function

- **Hebb rule**
  - **distance from inputs**
- **Delta rule**
  - **error in reproducing the output**

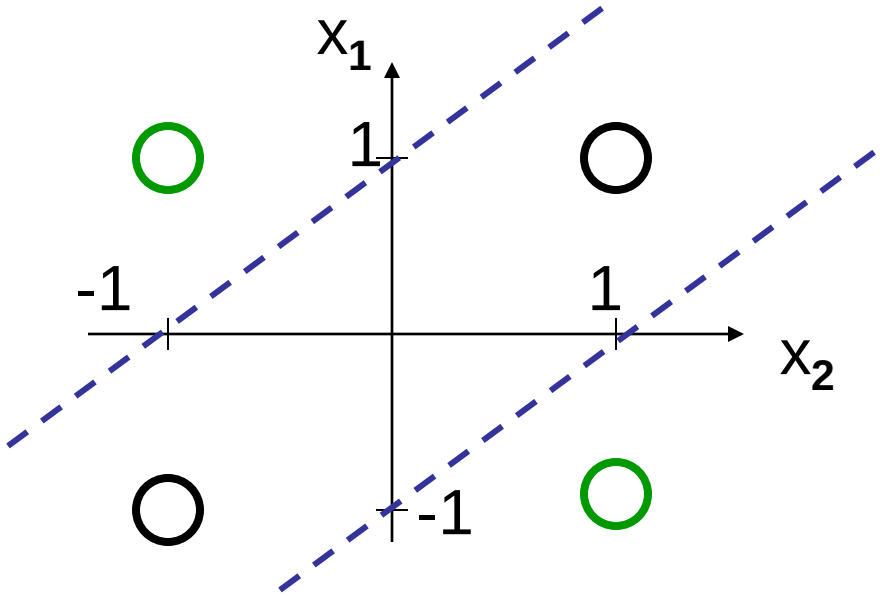
Multi-layer Perceptrons

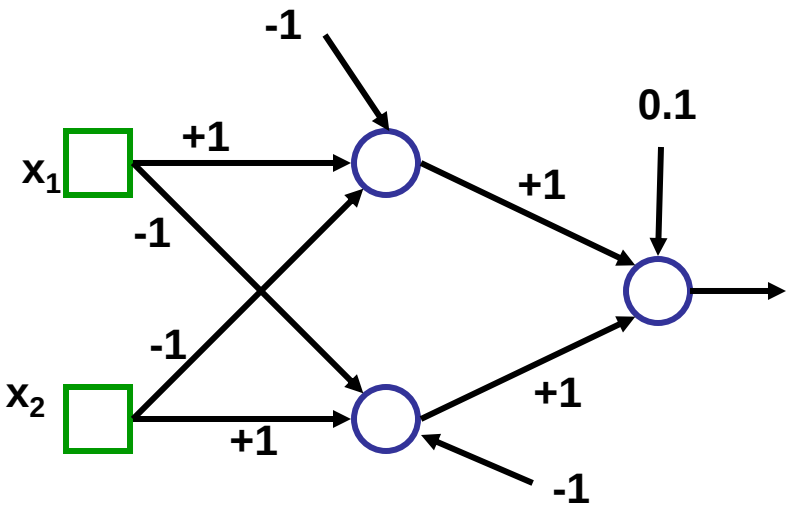
Multilayer Perceptrons: Architecture



A solution for the XOR problem

x_1	x_2	x_1 xor x_2
-1	-1	**-1**
-1	1	**1**
1	-1	**1**
1	1	**-1**



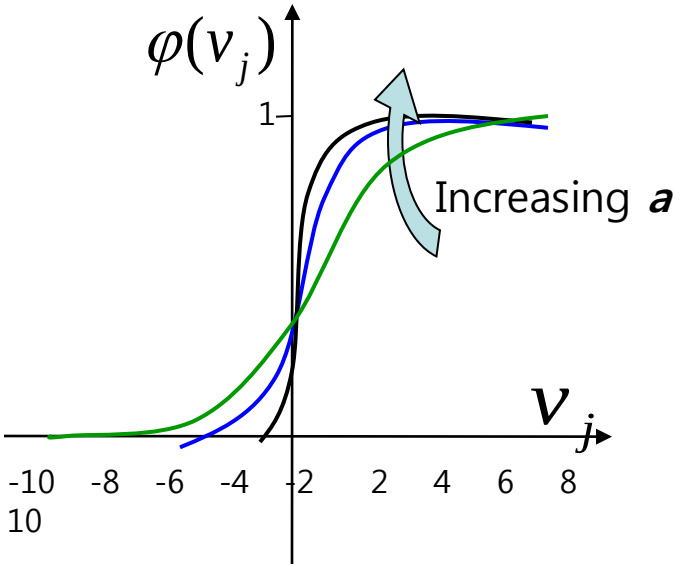


$$\varphi(v) = \begin{cases} 1 & \text{if } v > 0 \\ -1 & \text{if } v \le 0 \end{cases}$$

φ is the sign function.

NEURON MODEL

- Sigmoidal Function



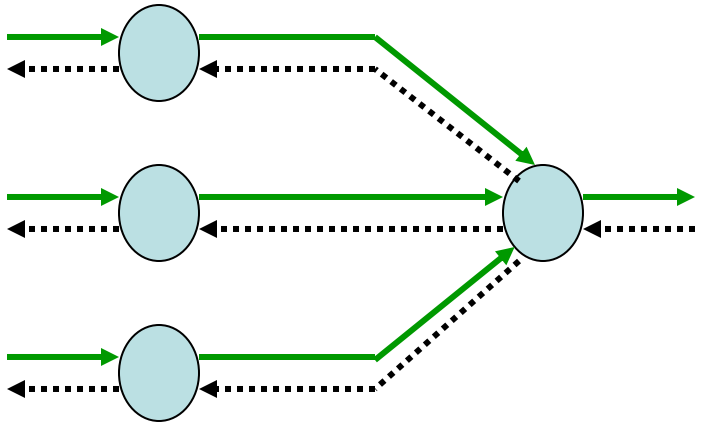
$$\varphi(\mathrm{v_j}) = \frac{1}{1+e^{-av_j}}$$

$$\mathrm{v_j} = \sum_{i=0,\ldots,m} w_{\mathrm{ji}} y_{\mathrm{i}}$$

- $\mathrm{v_j}$ induced field of neuron j
- Most common form of activation function
- a $\rightarrow\infty \Rightarrow \varphi \rightarrow$ threshold function
- Differentiable

Learning Algorithms

- Back-propagation algorithm



Function signals
Forward Step

Error signals
Backward Step

- It adjusts the weights of the NN in order to minimize the average squared error.

Average Squared Error

- Error signal of output neuron ***j*** at presentation of ***n-th*** training example:

$$e_j(n) = d_j(n) - y_j(n)$$

- Total energy at time ***n***:
- Average squared error:

$$E(n) = \frac{1}{2}\sum_{j \in C} e_j^2(n)$$

C: Set of neurons in output layer

N: size of training set

- Measure of learning performance:

$$E_{AV} = \frac{1}{N}\sum_{n=1}^{N} E(n)$$

- **Goal:** ***Adjust weights of NN to minimize E_{AV}***

Notation

e_j Error at output of neuron j

y_j Output of neuron j

$v_j = \sum_{i=0,\ldots,m} w_{ji} y_i$ Induced local field of neuron j

Weight Update Rule

Update rule is based on the gradient descent meth
take a step in the direction yielding the maximum
decrease of E

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

Step in direction opposite to the gradien

With w_{ji} weight associated to the link from neuro
to neuron j

Definition of the Local Gradient of neuron j

$$\delta_j = -\frac{\partial E}{\partial v_j}$$

Local Gradient

We obtain $$\delta_j = e_j \varphi'(v_j)$$

because

$$-\frac{\partial E}{\partial v_j} = -\frac{\partial E}{\partial e_j}\frac{\partial e_j}{\partial y_j}\frac{\partial y_j}{\partial v_j} = -e_j(-1)\varphi'(v_j)$$

Update Rule

- We obtain

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

because

$$\frac{\partial E}{\partial w_{ji}} = \frac{\partial E}{\partial v_j} \frac{\partial v_j}{\partial w_{ji}}$$

$$-\frac{\partial E}{\partial v_j} = \delta_j \qquad \frac{\partial v_j}{\partial w_{ji}} = y_i$$

Error e_j of output neuron

- Single layer Perceptron: ***j output neuron***

$$e_j = d_j - y_j$$

Then

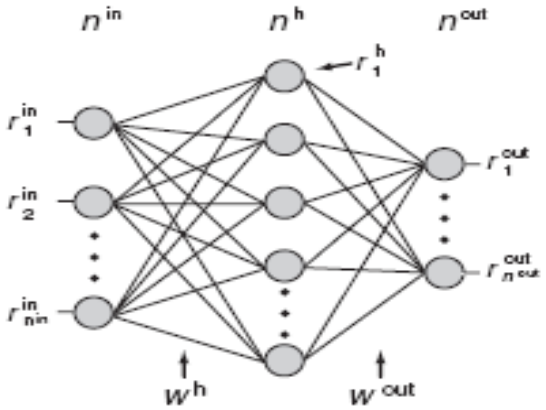
$$\delta_j = (d_j - y_j)\varphi'(v_j)$$

Multi-layer Perceptron

$$\mathbf{w} = \begin{pmatrix} w_{11} & w_{12} & w_{13} & \dots & w_{1n^{\text{out}}} \\ w_{21} & w_{22} & w_{23} & \dots & w_{2n^{\text{out}}} \\ . & . & . & \dots & . \\ . & . & . & \dots & . \\ . & . & . & \dots & . \\ w_{n^{\text{in}}1} & w_{n^{\text{in}}2} & w_{n^{\text{in}}3} & \dots & w_{n^{\text{in}}n^{\text{out}}} \end{pmatrix} \tag{6.8}$$

$$\mathbf{r}^{\text{out}} = g(\mathbf{w}\mathbf{r}^{\text{in}}), \tag{6.9}$$

$$r_i^{\text{out}} = g(\sum_j w_{ij} r_j^{\text{in}}). \tag{6.10}$$


n^{in}
n^{h}
n^{out}
r_1^h
r_1^{in}
r_2^{in}
r_{nin}^{in}
r_1^{out}
r_{nout}^{out}
w^h
w^{out}

$$n^w = n^{\text{in}} n^{\text{h}} + n^{\text{h}} n^{\text{out}}, \tag{6.11}$$

$$\mathbf{h}^{\text{h}} = \mathbf{w}^{\text{h}} \mathbf{r}^{\text{in}}, \tag{6.12}$$

$$h_i^{\text{h}} = \sum_j w_{ij}^{\text{h}} r_j^{\text{in}}. \tag{6.13}$$

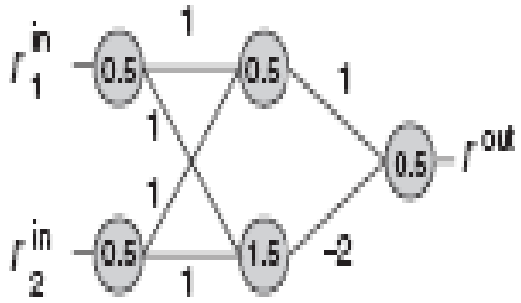
$$\mathbf{r}^{\text{h}} = g^{\text{h}}(\mathbf{h}^{\text{h}}). \tag{6.14}$$

$$\mathbf{r}^{\text{out}} = g^{\text{out}}(\mathbf{w}^{\text{out}}\mathbf{r}^{\text{h}}). \tag{6.15}$$

$$\mathbf{r}^{\text{out}} = g^{\text{out}}(\mathbf{w}^{\text{out}}g^{\text{h}}(\mathbf{w}^{\text{h}}\mathbf{r}^{\text{in}})). \tag{6.16}$$

$$\mathbf{r}^{\text{out}} = g^{\text{out}}(\mathbf{w}^{\text{out}}g^{\text{h3}}(\mathbf{w}^{\text{h3}}g^{\text{h2}}(\mathbf{w}^{\text{h2}}g^{\text{h1}}(\mathbf{w}^{\text{h1}}\mathbf{r}^{\text{in}})))). \tag{6.17}$$

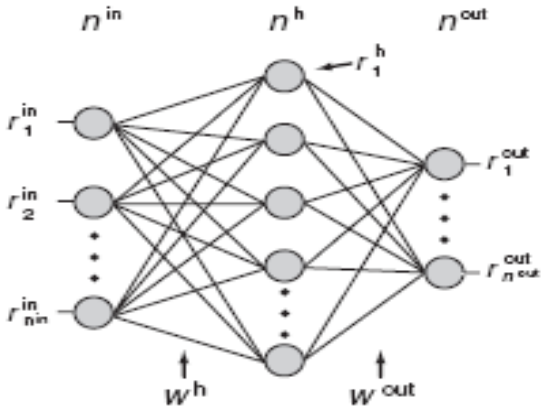
$$\begin{aligned}\mathbf{r}^{\text{out}} &= g^{\text{out}}(\mathbf{w}^{\text{out}}\mathbf{w}^{\text{h3}}\mathbf{w}^{\text{h2}}\mathbf{w}^{\text{h1}}\mathbf{r}^{\text{in}}) \\ &= g^{\text{out}}(\tilde{\mathbf{w}}\mathbf{r}^{\text{in}}).\end{aligned} \tag{6.18}$$


r_1^{in}
r_2^{in}
0.5
0.5
1.5
0.5
1
1
1
1
1
-2
r^{out}

$$\mathbf{w} = \begin{pmatrix} w_{11} & w_{12} & w_{13} & \dots & w_{1n^{\text{out}}} \\ w_{21} & w_{22} & w_{23} & \dots & w_{2n^{\text{out}}} \\ . & . & . & \dots & . \\ . & . & . & \dots & . \\ . & . & . & \dots & . \\ w_{n^{\text{in}}1} & w_{n^{\text{in}}2} & w_{n^{\text{in}}3} & \dots & w_{n^{\text{in}}n^{\text{out}}} \end{pmatrix} \tag{6.8}$$

$$\mathbf{r}^{\text{out}} = g(\mathbf{w}\mathbf{r}^{\text{in}}), \tag{6.9}$$

$$r_i^{\text{out}} = g(\sum_j w_{ij} r_j^{\text{in}}). \tag{6.10}$$


n^{in}
n^{h}
n^{out}
r_1^h
r_1^{in}
r_2^{in}
r_{nin}^{in}
r_1^{out}
r_{nout}^{out}
w^h
w^{out}

$$n^w = n^{\text{in}} n^{\text{h}} + n^{\text{h}} n^{\text{out}}, \tag{6.11}$$

$$\mathbf{h}^{\text{h}} = \mathbf{w}^{\text{h}} \mathbf{r}^{\text{in}}, \tag{6.12}$$

$$h_i^{\text{h}} = \sum_j w_{ij}^{\text{h}} r_j^{\text{in}}. \tag{6.13}$$

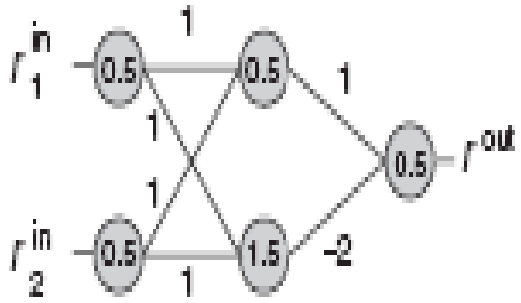
$$\mathbf{r}^{\text{h}} = g^{\text{h}}(\mathbf{h}^{\text{h}}). \tag{6.14}$$

$$\mathbf{r}^{\text{out}} = g^{\text{out}}(\mathbf{w}^{\text{out}}\mathbf{r}^{\text{h}}). \tag{6.15}$$

$$\mathbf{r}^{\text{out}} = g^{\text{out}}(\mathbf{w}^{\text{out}}g^{\text{h}}(\mathbf{w}^{\text{h}}\mathbf{r}^{\text{in}})). \tag{6.16}$$

$$\mathbf{r}^{\text{out}} = g^{\text{out}}(\mathbf{w}^{\text{out}}g^{\text{h3}}(\mathbf{w}^{\text{h3}}g^{\text{h2}}(\mathbf{w}^{\text{h2}}g^{\text{h1}}(\mathbf{w}^{\text{h1}}\mathbf{r}^{\text{in}})))). \tag{6.17}$$

$$\begin{aligned}\mathbf{r}^{\text{out}} &= g^{\text{out}}(\mathbf{w}^{\text{out}}\mathbf{w}^{\text{h3}}\mathbf{w}^{\text{h2}}\mathbf{w}^{\text{h1}}\mathbf{r}^{\text{in}}) \\ &= g^{\text{out}}(\tilde{\mathbf{w}}\mathbf{r}^{\text{in}}).\end{aligned} \tag{6.18}$$


r_1^{in}
r_2^{in}
0.5
0.5
1.5
0.5
1
1
1
1
1
-2
r^{out}