Learning Agent: Neural Network

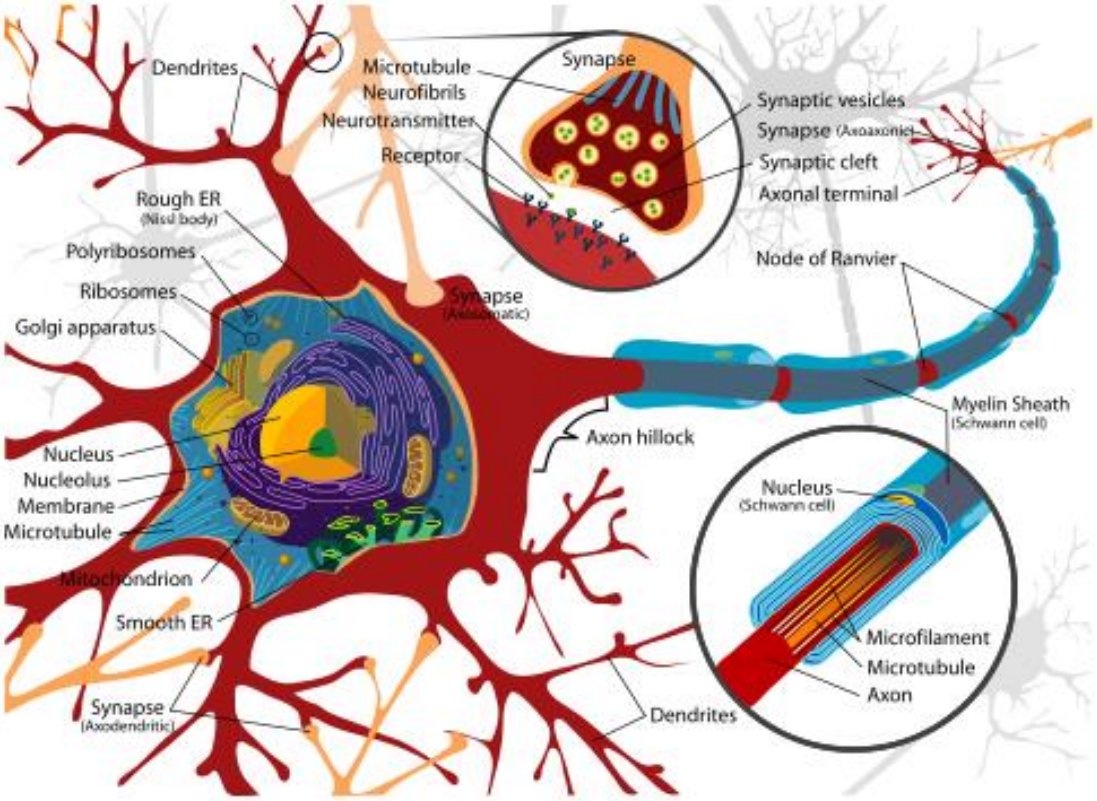
IF-3054 Inteligensi Buatan

Teknik Informatika ITB

Overview Learning Agent

- Knowledge acquisition bottleneck problem
- Learning: changes, task, experience, performance (inductive learning)
- Design learning element: goal (component of performance element), available feedback, learning algorithm (representation), prior knowledge
- Learning type: supervised, unsupervised, reinforcement
- Learning algorithm: k-NN, Naive Bayes, ID3

Biological Neuron

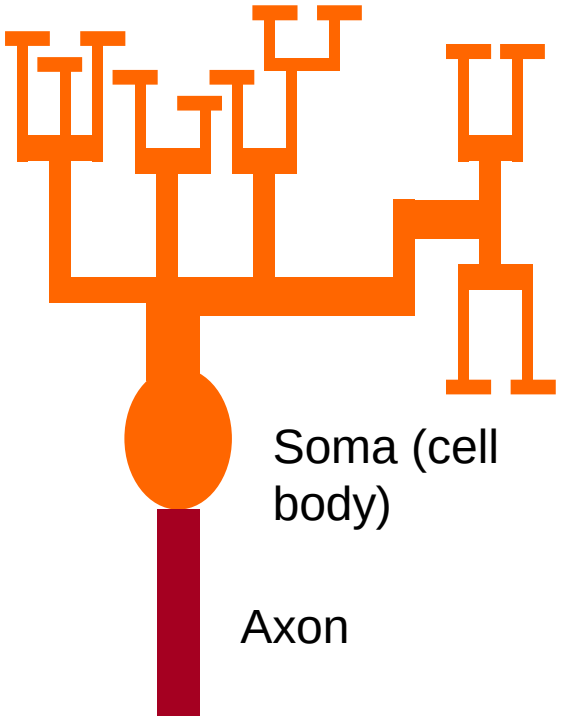


http://www.chemistry.ucsc.edu/~lokey/108A_10/images/neuron.png

- A neuron has
  - A branching input (dendrites)
  - A branching output (the axon)
- The information circulates from the dendrites to the axon via the cell body
- Axon connects to dendrites via synapses
  - Synapses vary in strength
  - Synapses may be excitatory or inhibitory

Peter Andras
peter.andras@ncl.ac.uk

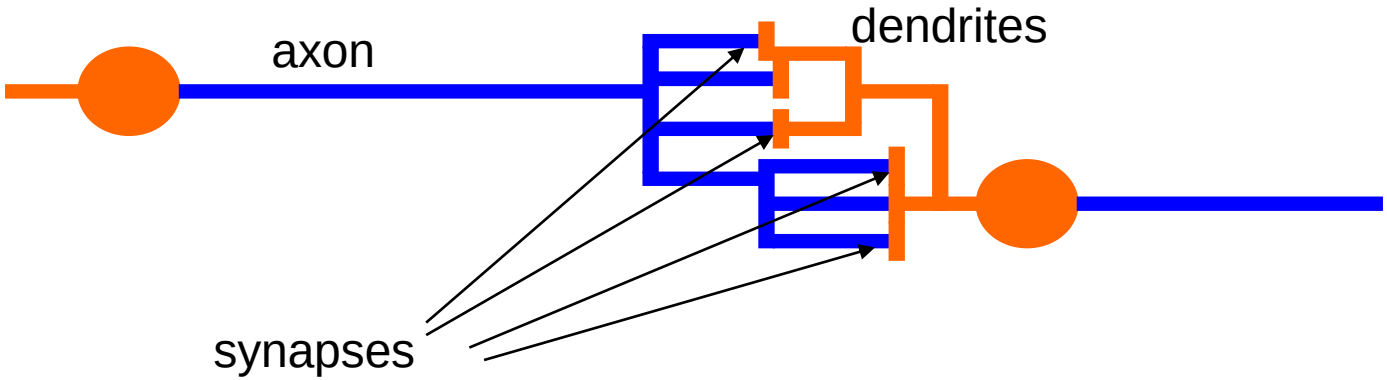
Dendrites



Soma (cell body)

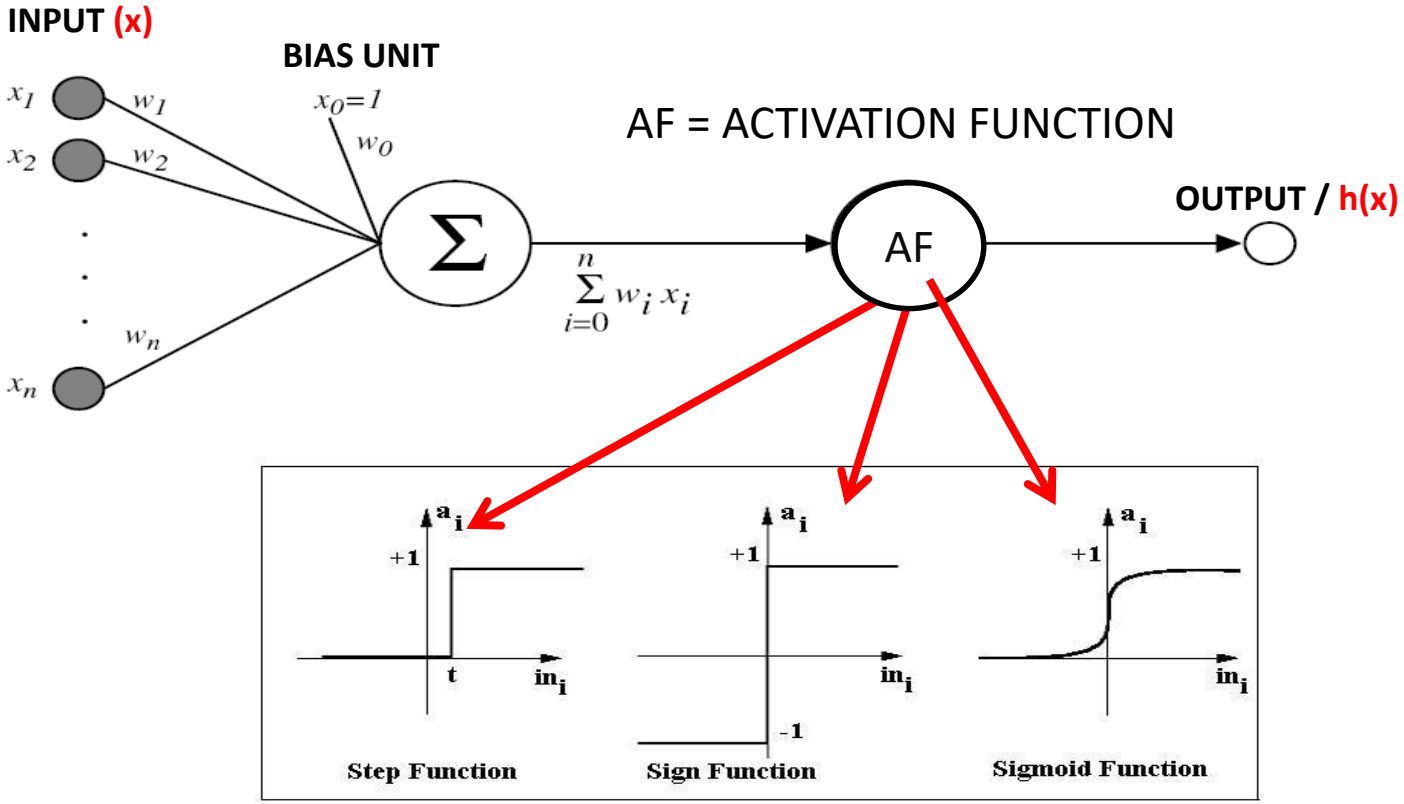
Axon

The information transmission happens at the synapses.



What is an artificial neuron ?

- Parameterized function with restricted output range
- Example: simple/single unit perceptron. $x_0=1$



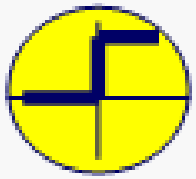
Neural Network Mathematics

Neural network: input / output transformation

$$y_{out} = F(x, W)$$

W is the matrix of all weight vectors.

Activation function (F):



Step function

(Linear Threshold Unit)

step(x) = 1, if x >= threshold
0, if x < threshold

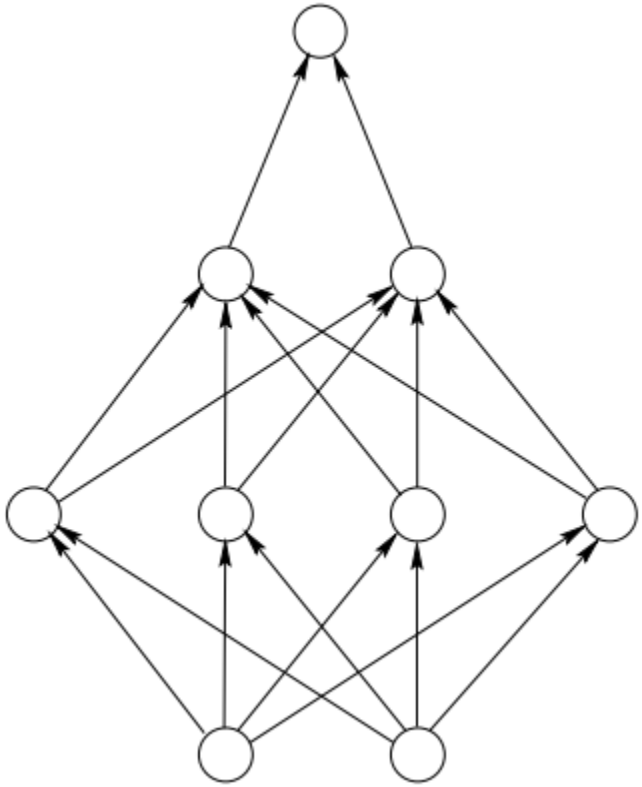
Sign function

sign(x) = +1, if x >= 0
-1, if x < 0

Sigmoid function

sigmoid(x) = $1/(1+e^{-x})$

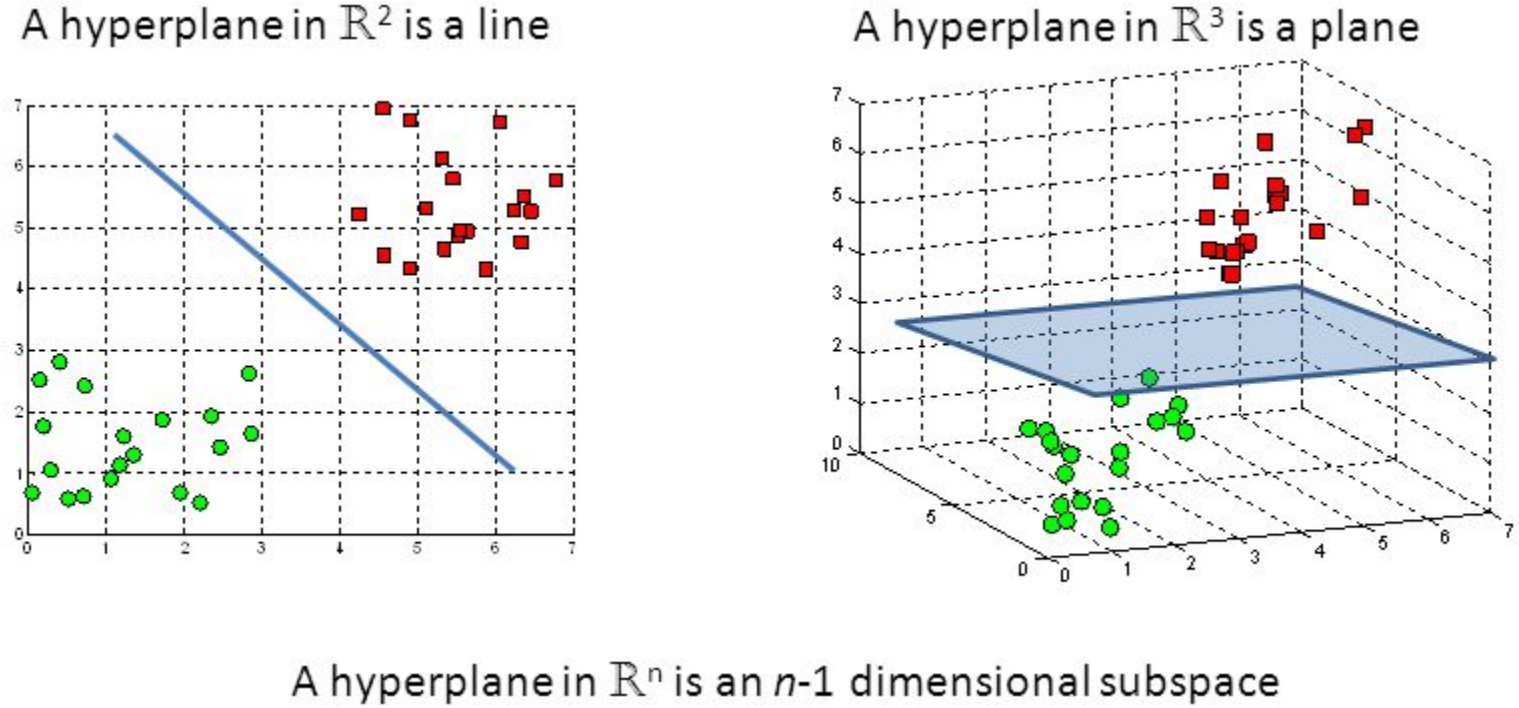
ANN Topology



- Artificial neuron, node, cell, unit, neurode (neuron & node)
- Neuron types:
  - input neuron: receives external inputs from outside the network
  - hidden neuron: has no direct interaction with the ‘outside world’
  - output neuron: produces some of the outputs of the network.

ANN

- Represent hyperplane decision surface in the n-dimensional space of instances



Sumber: http://images.slideplayer.com/5/1579281/slides/slide_32.jpg

Neural Network Learning

- Data: set of value pairs: (x^t, y_t), $y_t = g(x^t) + z_t$; z_t is random measurement noise.
- Objective: find a neural network that represents the input / output transformation (a function) F(x,W) such that F(x,W) approximates g(x) for every x

Learning Process

Error measure:

$$E = \frac{1}{N}\sum_{t=1}^{N}(F(x_t;W) - y_t)^2$$

Rule for changing the synaptic weights:

$$\Delta w_i^j = -\eta \cdot \frac{\partial E}{\partial w_i^j}(W)$$

$$w_i^{j,new} = w_i^j + \Delta w_i^j$$

η is the learning parameter
(usually a constant)

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

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

$$\Delta W_i = \eta * (D\text{-}Y).I_i$$

Learning rate; Desired output; Actual output

Learning with a perceptron

Perceptron: $y_{out} = w^T x$

Data: $(x^1, y_1), (x^2, y_2), \ldots, (x^N, y_N)$

Error: $E(t) = (y(t)_{out} - y_t)^2 = (w(t)^T x^t - y_t)^2$

Learning:

$$w_i(t+1) = w_i(t) - \eta \cdot \frac{\partial E(t)}{\partial w_i} = w_i(t) - \eta \cdot \frac{\partial (w(t)^T x^t - y_t)^2}{\partial w_i}$$

$$\boxed{w_i(t+1) = w_i(t) - \eta \cdot (w(t)^T x^t - y_t) \cdot x_i^t}$$

$$w(t)^T x = \sum_{j=1}^{m} w_j(t) \cdot x_j^t$$

Illustration

- Dataset:
  1. $<(1,0,1)^T,-1>$
  2. $<(0,-1,-1)^T,1>$
  3. $<(-1,-0.5,-1)^T,1>$
- Inisialisasi random **w** $(w_0,w_1,w_2,w_3)^T$: $(0,1,-1,0)^T$
- Fungsi aktivasi: sign
- Learning rate η: 0.1

Update Weight

- If the output is not correct, the weights are adjusted according to the formula:
- $w_{new} = w_{old} + \eta(desired - output)*input$
- Input-1 <**data**,desired>: $<(1,0,1)^T,-1>$
- **w** (w_0,w_1,w_2,w_3): $(0,1,-1,0)^T$
- Output y_{out}=sign(0*1+1*1+-1*0+0*1)=+1 (>0)
- $w_{new} = (0,1,-1,0)^T + 0.1*(-1 - 1)*(1,1,0,1)^T$
  $= (0,1,-1,0)^T + (-0.2,-0.2,0,-0.2)=(-0.2,0.8,-1,-0.2)$

Perceptron Training Rule

learning rate					0.1					
MSE threshold					0.0					
Max Iterasi					10.0					
Fungsi aktivasi					sign					

	x1	x2	x3	Target	w0	w1	w2	w3	sigma	sign
Iterasi 1	1	0	1	-1	0.00	1.00	-1.00	0.00	1.00	1
	0	-1	-1	1	-0.20	0.80	-1.00	-0.20	1.00	1
	-1	-0.5	-1	1	-0.20	0.80	-1.00	-0.20	-0.30	-1
					0.00	0.60	-1.10	-0.40		

$$w_i := w_i + \eta(y_i - o_i)x,\ i = 1, \ldots, m$$

x0	x1	x2	x3	Target	sigma	sign	(t-o)^2/2
1	1	0	1	-1	0.2	1	2
1	0	-1	-1	1	1.5	1	0
1	-1	-0.5	-1	1	0.35	1	0
Err= cum. half squared error							2

	x1	x2	x3	Target	w0	w1	w2	w3	sigma	sign
Iterasi 2	1	0	1	-1	0.00	0.60	-1.10	-0.40	0.20	1
	0	-1	-1	1	-0.20	0.40	-1.10	-0.60	1.50	1
	-1	-0.5	-1	1	-0.20	0.40	-1.10	-0.60	0.55	1
					-0.20	0.40	-1.10	-0.60		

x0	x1	x2	x3	Target	sigma	sign	(t-o)^2/2
1	1	0	1	-1	-0.4	-1	0
1	0	-1	-1	1	1.5	1	0
1	-1	-0.5	-1	1	0.55	1	0
Err= cum. half squared error							0

Contoh: AND

Fungsi aktivasi : sign

$w_1 = w_2 = 0.5$; $b=w_0 = -0.75$

Data: <(0,0),-1>; <(0,1),-1>;
<(1,0),-1>; <(1,1),+1>

- Sign($\mathbf{w}^T\mathbf{x}_1$+b)=sign(-0.75)=-1
- Sign($\mathbf{w}^T\mathbf{x}_2$+b)= sign(-0.25)=-1
- Sign($\mathbf{w}^T\mathbf{x}_3$+b)=sign(-0.25)=-1
- Sign($\mathbf{w}^T\mathbf{x}_4$+b)=sign(0.15)=+1

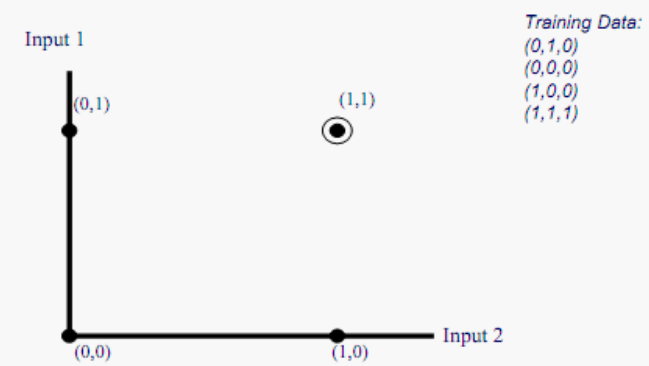
Fungsi aktivasi: step (threshold=0)

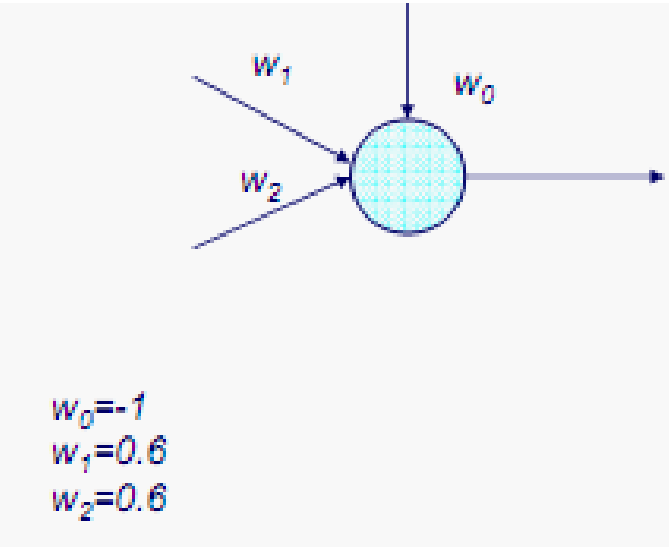
$w_1 = w_2 = 0.6$; $b=w_0 = -1$

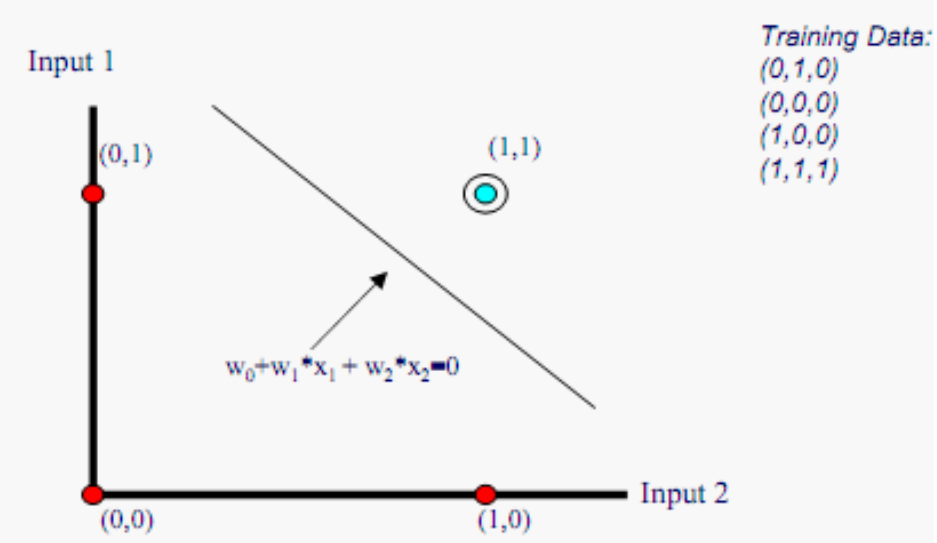
Data: <(0,0),0>; <(0,1),0>;
<(1,0),0>; <(1,1),1>

- Sign($\mathbf{w}^T\mathbf{x}_1$+b)=sign(-1)=0
- Sign($\mathbf{w}^T\mathbf{x}_2$+b)= sign(-0.4)=0
- Sign($\mathbf{w}^T\mathbf{x}_3$+b)=sign(-0.4)=0
- Sign($\mathbf{w}^T\mathbf{x}_4$+b)=sign(0.2)=1

Contoh: AND (Fungsi aktivasi step)





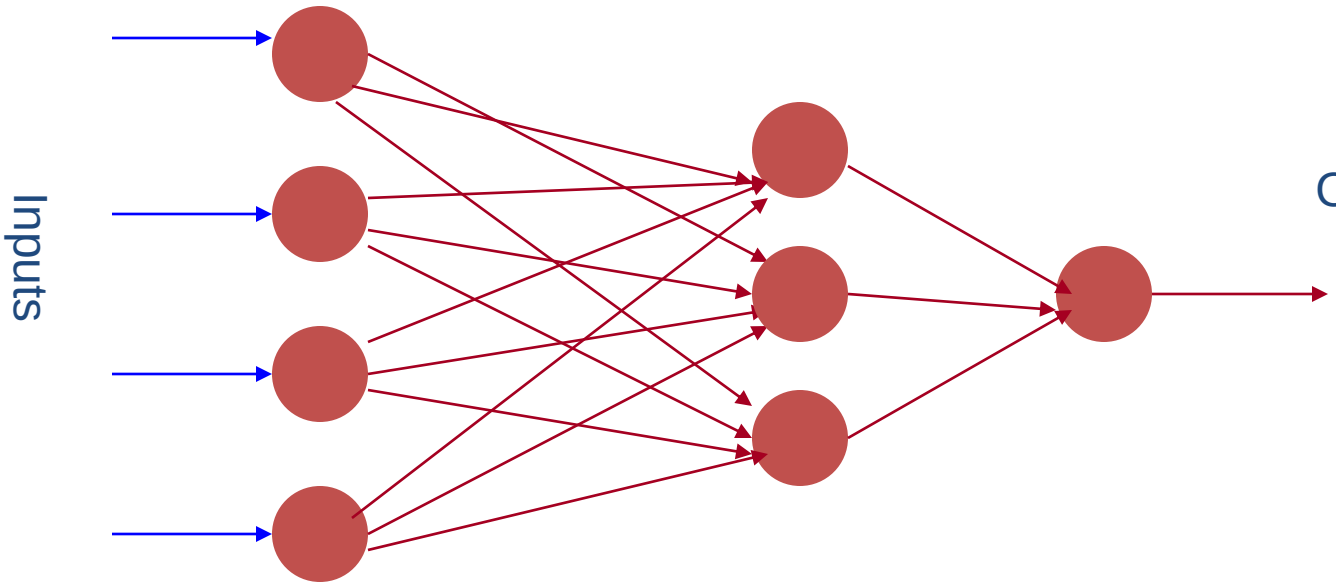


Summary of Perceptron

- Perceptron training rule guaranteed to succeed if
  - Training examples are linearly separable
  - Sufficiently small learning rate η
- Linear unit training rule uses gradient descent
  - Guaranteed to converge to hypothesis with minimum squared error
  - Given sufficiently small learning rate η
  - Even when training data contains noise
  - Even when training data not separable by H

Tom Mitchell, Machine Learning
Dept. of Computer Science, Carnegie Mellon University

MLP Neural Network



$$y_1^1 = f(x_1, w_1^1)$$
$$y_2^1 = f(x_2, w_2^1)$$
$$y_3^1 = f(x_3, w_3^1)$$
$$y_4^1 = f(x_4, w_4^1)$$

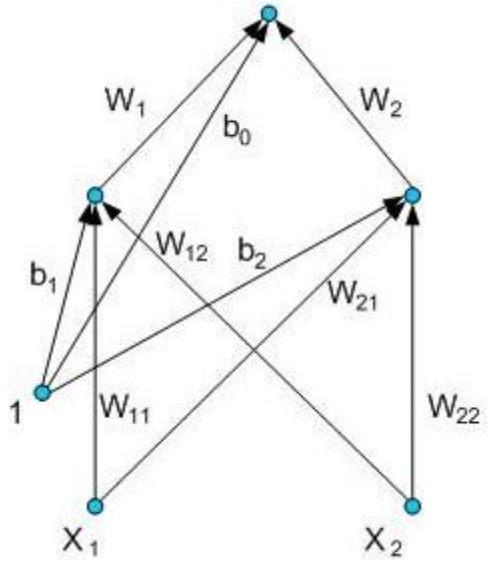
$$y^1 = \begin{pmatrix} y_1^1 \\ y_2^1 \\ y_3^1 \\ y_4^1 \end{pmatrix}$$

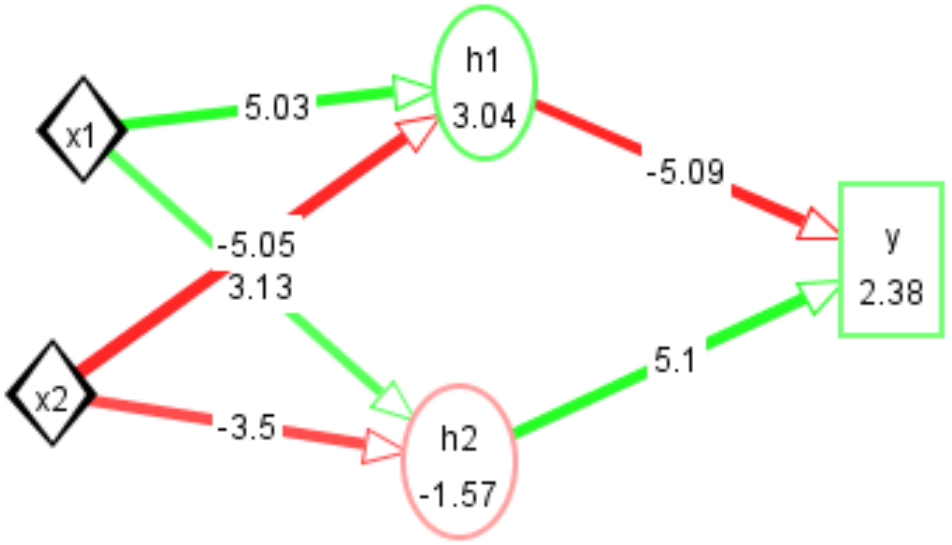
$$y_1^2 = f(y^1, w_1^2)$$
$$y_2^2 = f(y^1, w_2^2)$$
$$y_3^2 = f(y^1, w_3^2)$$

$$y^2 = \begin{pmatrix} y_1^2 \\ y_2^2 \\ y_3^2 \end{pmatrix}$$

$$y_{Out} = f(y^2, w_1^3)$$

Contoh MLP: XOR (Step)

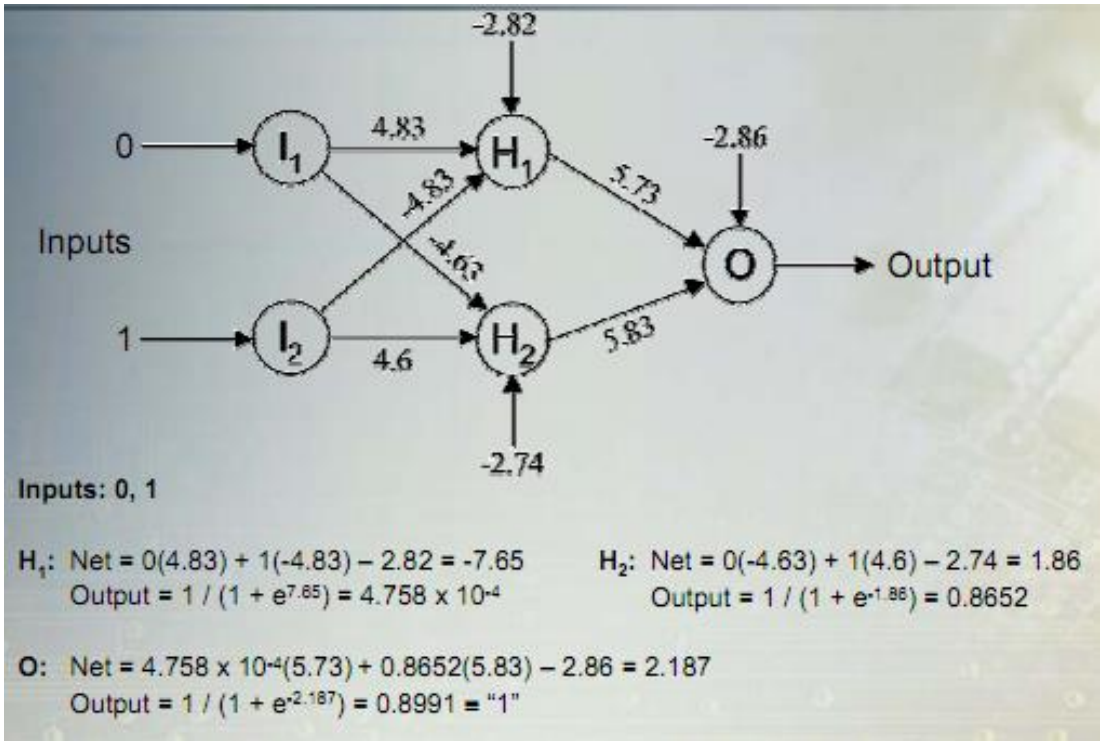




http://www.aispace.org/neural/index.shtml

w11	w12	b1	w21	w22	b2	w1	w2	b0
5.03	-5.05	3.04	3.13	-3.5	-1.57	-5.09	5.1	2.38
			step		step		step	
x1	x2	v1	y1	v2	y2	v	y	
0	1	-2.01	0	-5.07	0	2.38	1	
0	0	3.04	1	-1.57	0	-2.71	0	
1	0	8.07	1	1.56	1	2.39	1	
1	1	3.02	1	-1.94	0	-2.71	0	

Contoh MLP: XOR (Sigmoid)

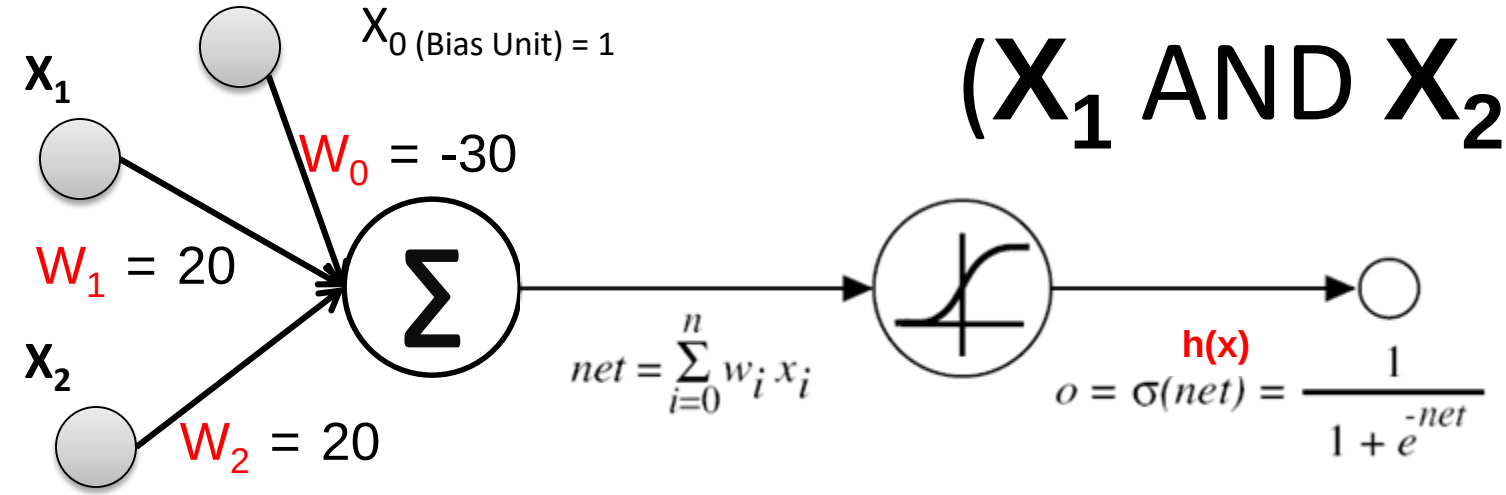


w11	w12	b1	w21	w22	b2	w1	w2	b0
4.83	-5	-2.82	-4.63	4.6	-2.74	5.73	5.83	-2.86
x1	x2	net1	y1 (sigmoid)	net2	y2 (sigmoid)	net_out	y_out (sigmoid)	
0	1	-7.65	0.0005	1.86	0.8653	2.187408	0.8991	
0	0	-2.82	0.0563	-2.74	0.0607	-2.18406	0.1012	
1	0	2.01	0.8818	-7.37	0.0006	2.19663	0.8999	
1	1	-2.82	0.0563	-2.77	0.0590	-2.19389	0.1003	

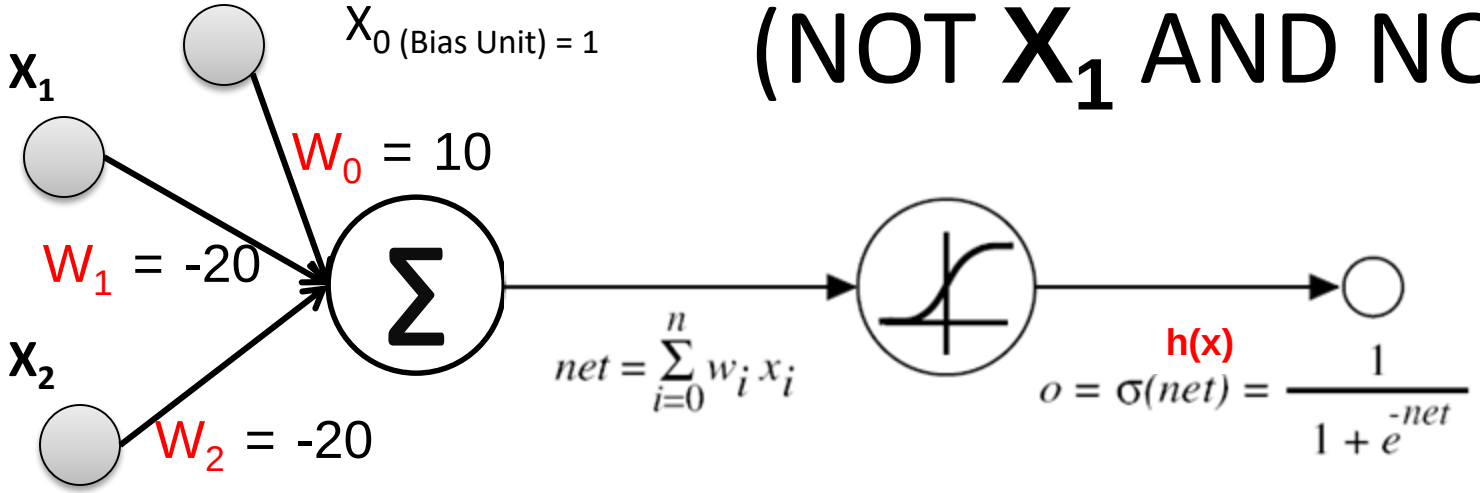
Contoh MLP: XNOR

INPUT		OUTPUT
A	B	A AND B
0	0	1
0	1	0
1	0	0
1	1	1

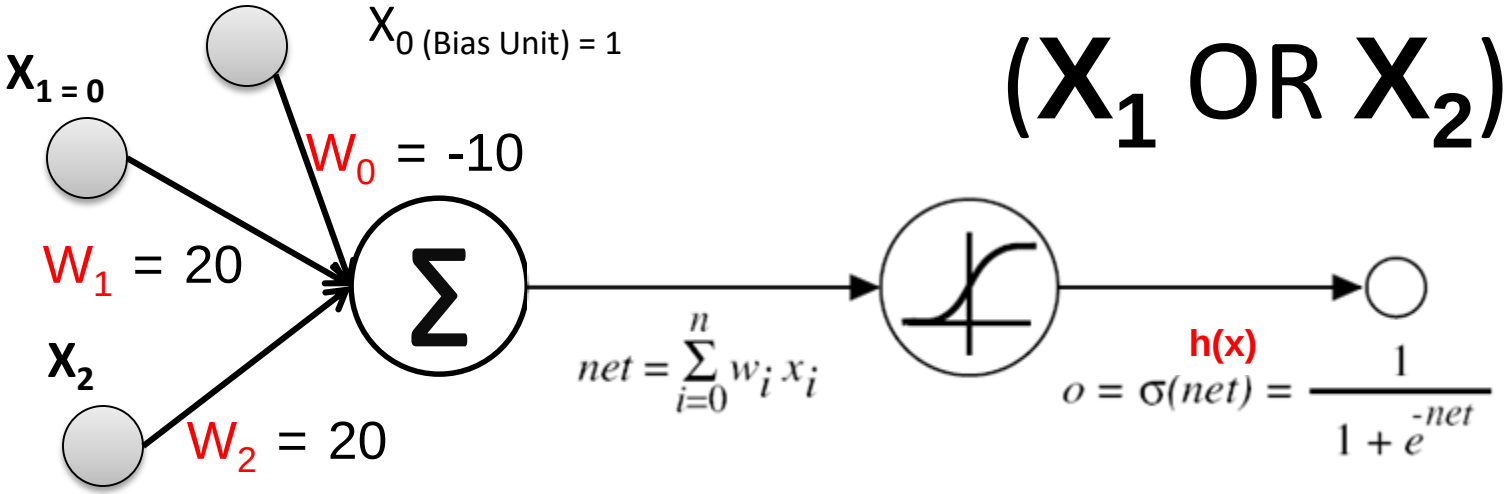
(A AND B) OR (NOT A AND NOT B)



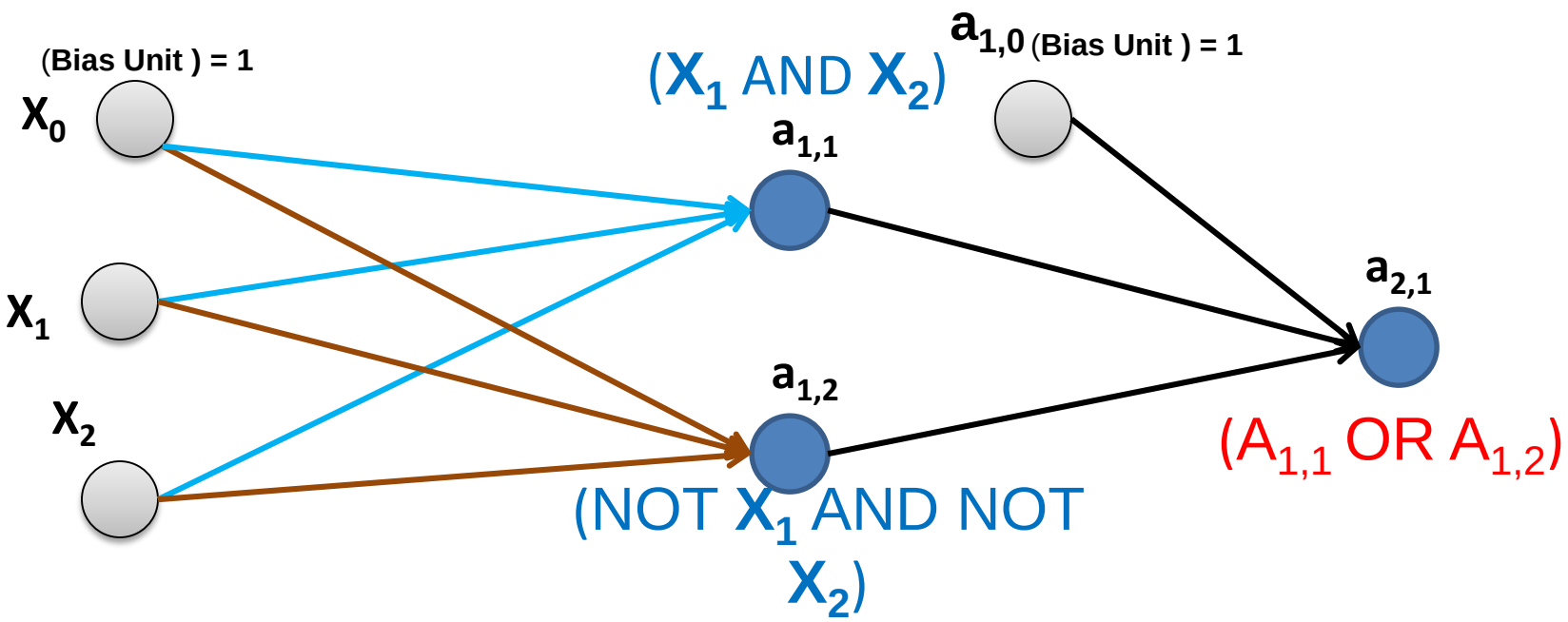
(X_1 AND X_2)

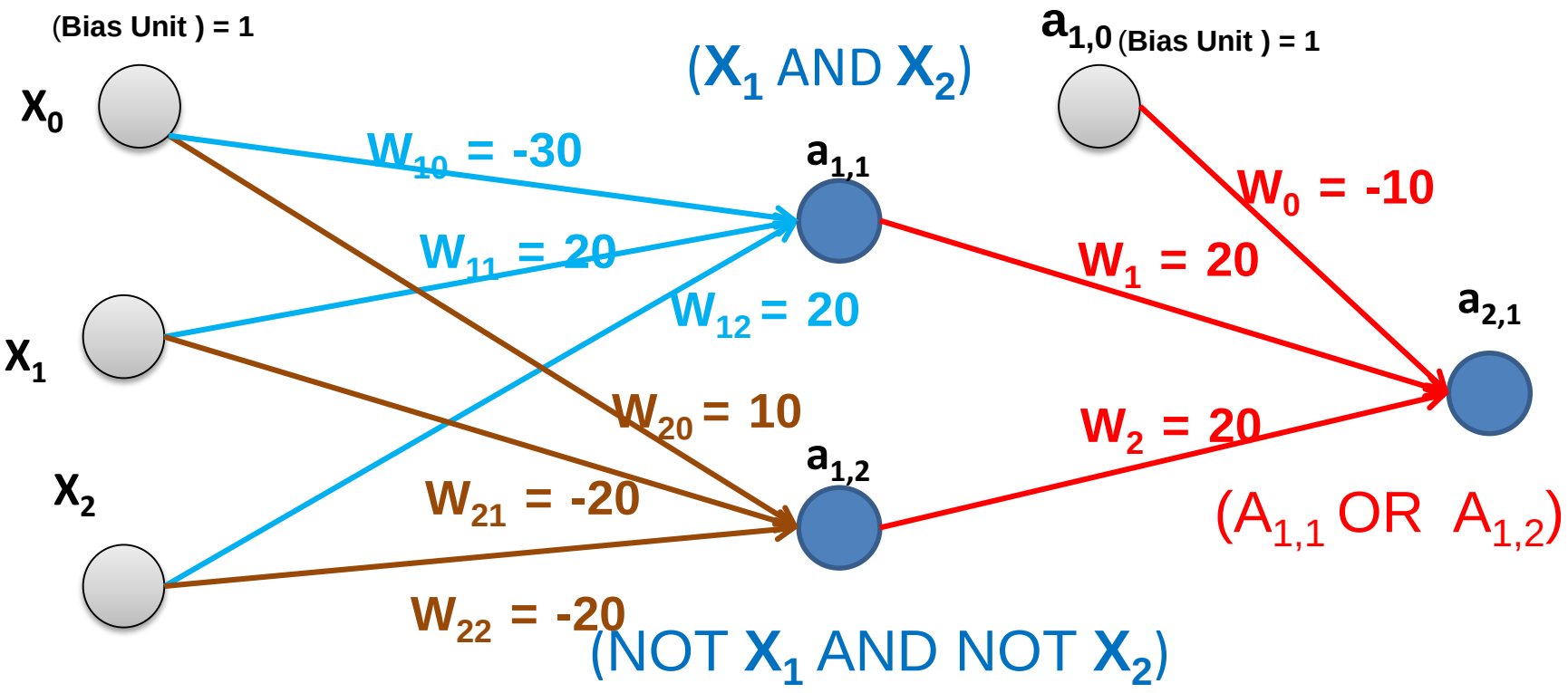


(NOT X_1 AND NOT X_2)


X0 (Bias Unit) = 1
X1 = 0
W0 = -10
W1 = 20
X2
W2 = 20
Σ
net = Σ_{i=0}^{n} w_i x_i
h(x)
o = σ(net) = 1 / (1 + e^{-net})

$(X_1$ OR $X_2)$


(Bias Unit) = 1
X0
X1
X2
(X1 AND X2)
a1,1
a1,2
(NOT X1 AND NOT X2)
a1,0 (Bias Unit) = 1
a2,1
(A1,1 OR A1,2)


(Bias Unit) = 1
X_0
X_1
X_2
W_{10} = -30
W_{11} = 20
W_{12} = 20
W_{20} = 10
W_{21} = -20
W_{22} = -20
$a_{1,1}$
$a_{1,2}$
(NOT X_1 AND NOT X_2)
(X_1 AND X_2)
$a_{1,0}$ (Bias Unit) = 1
W_0 = -10
W_1 = 20
W_2 = 20
$a_{2,1}$
($A_{1,1}$ OR $A_{1,2}$)

Menentukan Jumlah Hidden Layer

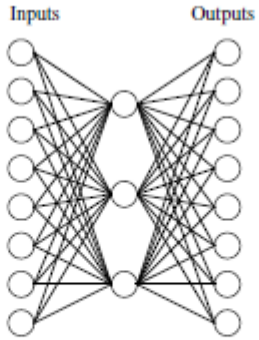
Table 5.1: Determining the Number of Hidden Layers

Number of Hidden Layers	Result
none	Only capable of representing linear separable functions or decisions.
1	Can approximate any function that contains a continuous mapping from one finite space to another.
2	Can represent an arbitrary decision boundary to arbitrary accuracy with rational activation functions and can approximate any smooth mapping to any accuracy.

Sumber: Introduction to Neural Network for Java 2nd Ed

Example: One Hidden Layer

A network:



Learned hidden layer representation:

Input		Hidden Values				Output
10000000	→	.89	.04	.08	→	10000000
01000000	→	.01	.11	.88	→	01000000
00100000	→	.01	.97	.27	→	00100000
00010000	→	.99	.97	.71	→	00010000
00001000	→	.03	.05	.02	→	00001000
00000100	→	.22	.99	.99	→	00000100
00000010	→	.80	.01	.98	→	00000010
00000001	→	.60	.94	.01	→	00000001

Example: Two Hidden Layer

Multiple output units: One-vs-all.

Pedestrian

Car

Motorcycle

Truck

$h_\Theta(x) \in \mathbb{R}^4$

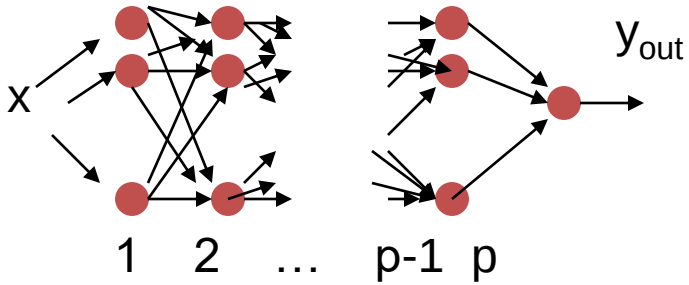
Want $h_\Theta(x) \approx \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$, $h_\Theta(x) \approx \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$, $h_\Theta(x) \approx \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$, etc.

when pedestrian when car when motorcycle

Learning with MLP neural networks

MLP neural network:

with p layers



$$y_k^1 = \frac{1}{1+e^{-w^{1kT}x - a_k^1}},\ k = 1,\ldots,M_1$$

$$y^1 = (y_1^1,\ldots,y_{M_1}^1)^T$$

$$y_k^2 = \frac{1}{1+e^{-w^{2kT}y^1 - a_k^2}},\ k = 1,\ldots,M_2$$

$$y^2 = (y_1^2,\ldots,y_{M_2}^2)^T$$

$$\ldots$$

$$y_{out} = F(x;W) = w^{pT} y^{p-1}$$

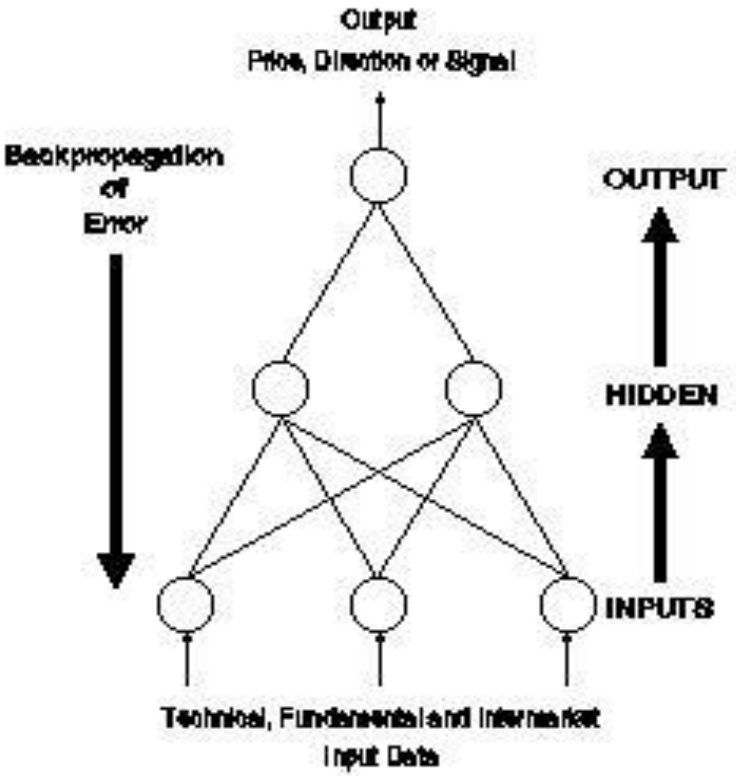
Data: $(x^1, y_1),(x^2, y_2),\ldots,(x^N, y_N)$

Error: $E(t) = (y(t)_{out} - y_t)^2 = (F(x^t;W) - y_t)^2$

Learning with backpropagation

Solution of the complicated learning:

- calculate first the changes for the synaptic weights of the output neuron;
- calculate the changes backward starting from layer p-1, and propagate backward the local error terms.



The method is still relatively complicated but it is much simpler than the original optimisation problem.

BACKPROPAGATION($training\_examples$, η, n_{in}, n_{out}, n_{hidden})

Each training example is a pair of the form $(\vec{x}, \vec{t})$, *where* $\vec{x}$ *is the vector of network input values, and* $\vec{t}$ *is the vector of target network output values.*

η *is the learning rate (e.g., .05).* n_{in} *is the number of network inputs,* n_{hidden} *the number of units in the hidden layer, and* n_{out} *the number of output units.*

The input from unit i *into unit* j *is denoted* x_{ji}, *and the weight from unit* i *to unit* j *is denoted* w_{ji}.

- Create a feed-forward network with n_{in} inputs, n_{hidden} hidden units, and n_{out} output units.
- Initialize all network weights to small random numbers (e.g., between $-.05$ and .05).
- Until the termination condition is met, Do
  - For each $(\vec{x}, \vec{t})$ in $training\_examples$, Do

Propagate the input forward through the network:

1. Input the instance $\vec{x}$ to the network and compute the output o_u of every unit u in the network.

Propagate the errors backward through the network:

2. For each network output unit k, calculate its error term δ_k

$$\delta_k \leftarrow o_k(1 - o_k)(t_k - o_k) \tag{T4.3}$$

3. For each hidden unit h, calculate its error term δ_h

$$\delta_h \leftarrow o_h(1 - o_h) \sum_{k \in outputs} w_{kh}\delta_k \tag{T4.4}$$

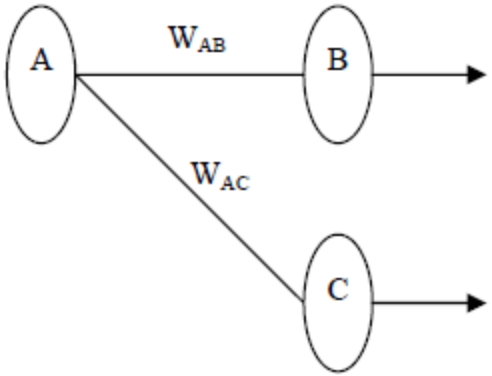
4. Update each network weight w_{ji}

$$w_{ji} \leftarrow w_{ji} + \Delta w_{ji}$$

where

$$\Delta w_{ji} = \eta \delta_j x_{ji} \tag{T4.5}$$

Backpropagation Learning



1. First apply the inputs to the network and work out the output – remember this initial output could be anything, as the initial weights were random numbers.

2. Next work out the error for neuron B. The error is *What you want – What you actually get, in other words:*

ErrorB = OutputB (1-OutputB)(TargetB – OutputB)

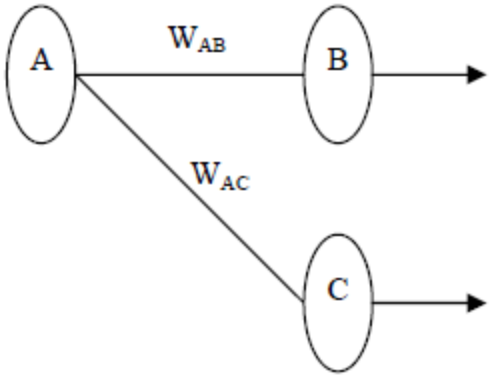
The "*Output(1-Output)" term is necessary in the equation because of the Sigmoid* Function – if we were only using a threshold neuron it would just be *(Target – Output).*

3. Change the weight. Let W^{+}_{AB} be the new (trained) weight and W_{AB} be the initial weight.

$W^{+}_{AB} = W_{AB} + (\text{ErrorB} \times \text{OutputA})$

Notice that it is the output of the connecting neuron (neuron A) we use (not B). We update all the weights in the output layer in this way.

Backpropagation Learning (2)



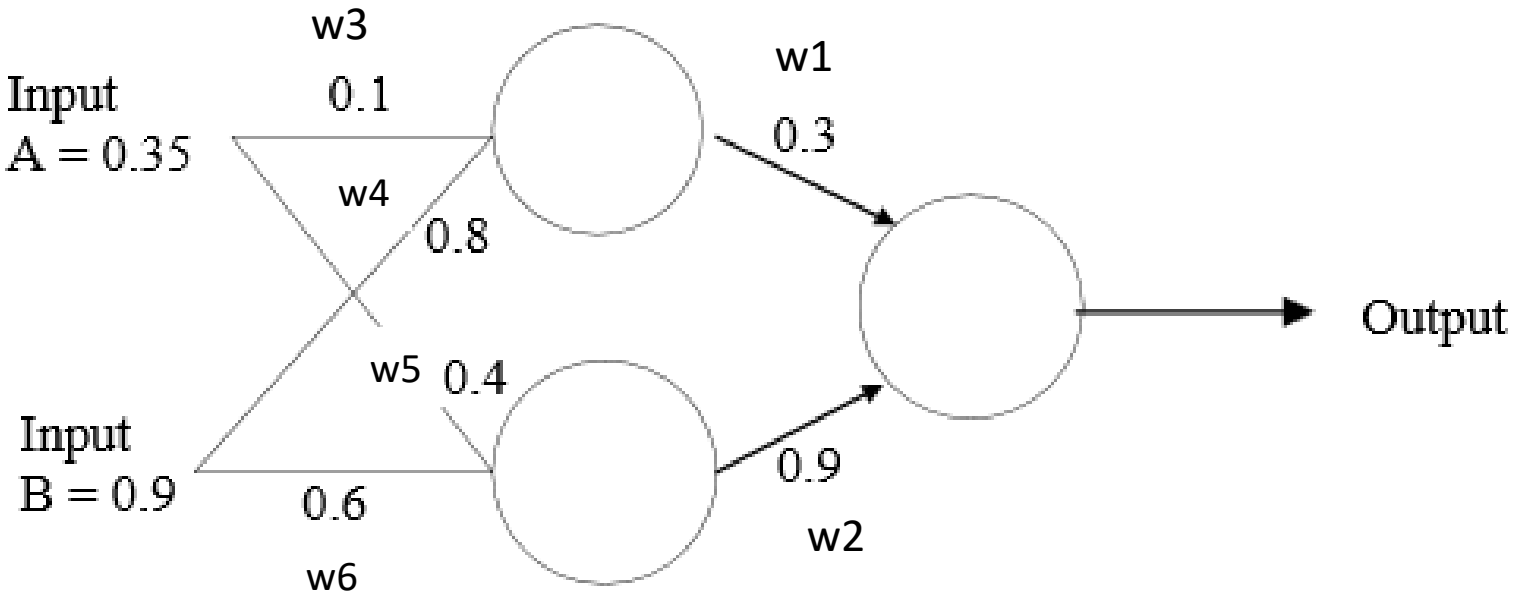
4. Calculate the Errors for the hidden layer neurons. Unlike the output layer we can't calculate these directly (because we don't have a Target), so we *Back Propagate* them from the output layer (hence the name of the algorithm). This is done by taking the Errors from the output neurons and running them back through the weights to get the hidden layer errors. For example if neuron A is connected as shown to B and C then we take the errors from B and C to generate an error for A.

$$\text{ErrorA} = \text{Output A} * (1 - \text{Output A})(\text{ErrorB} * W_{AB} + \text{ErrorC} * W_{AC})$$

Again, the factor "*Output (1 - Output)" is present because of the sigmoid squashing* function.

5. Having obtained the Error for the hidden layer neurons now proceed as in stage 3 to change the hidden layer weights. By repeating this method we can train a network of any number of layers.

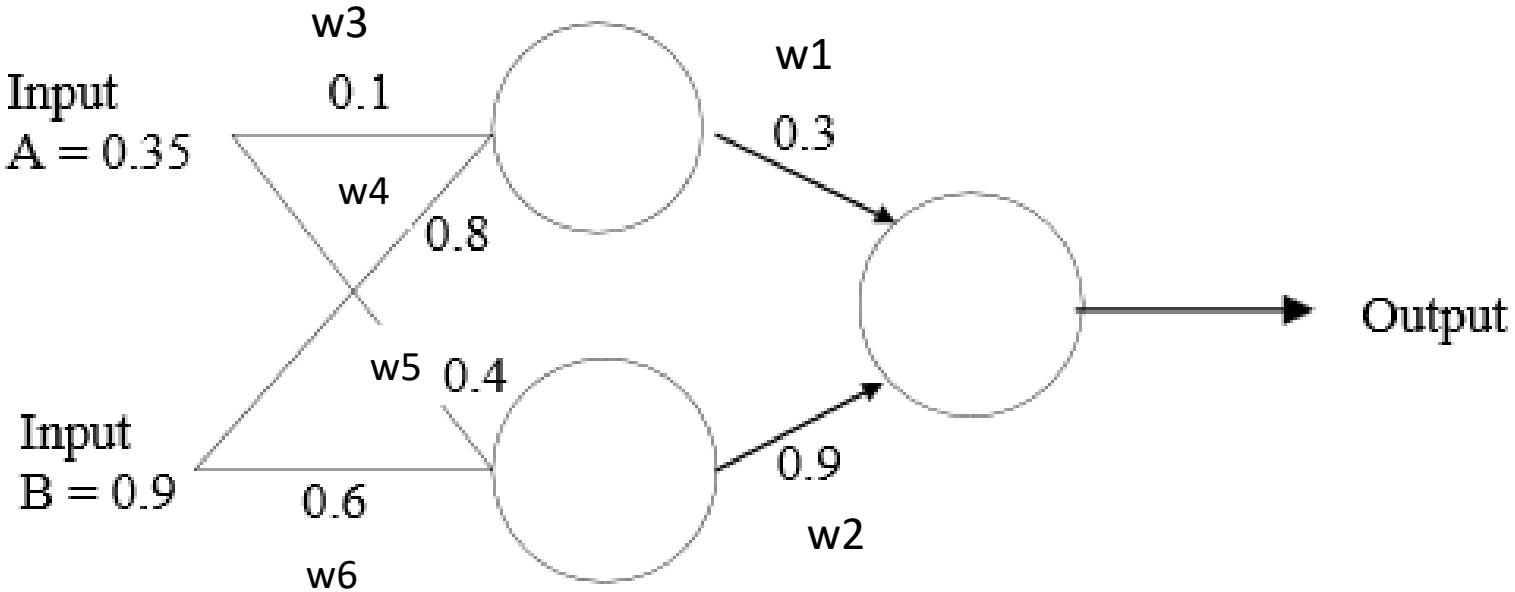
FeedForward MLP: Example



- Activation function: sigmoid (1/(1+e^(-x)))
- Output: 0.69

https://www4.rgu.ac.uk/files/chapter3%20-%20bp.pdf

FeedForward MLP : Example

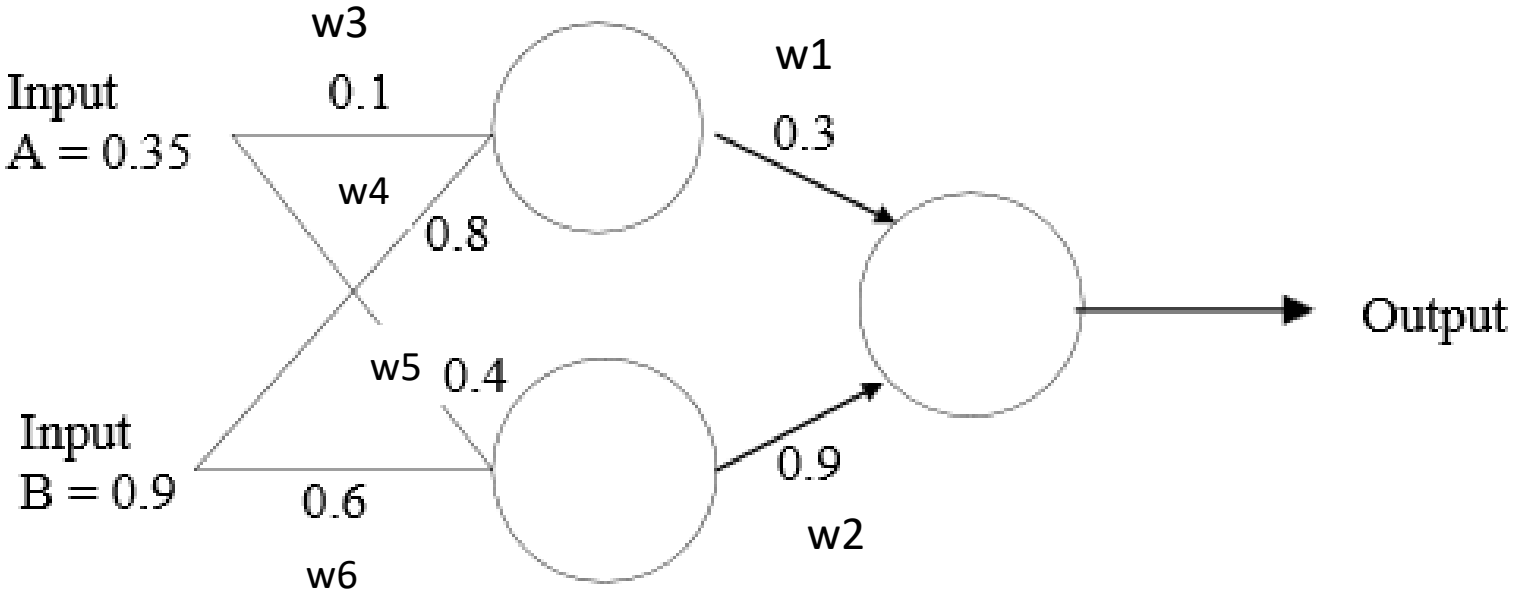


	A	B	C	D	E	F	G	H	I	J	K
1	A	B	w1	w2	w3	w4	w5	w6	Out_h11	Out_h12	Out_1
2	0.35	0.9	0.3	0.9	0.1	0.8	0.4	0.6	0.680	0.664	0.690

Out_h11=1/(1+EXP(-1*(A2*E2+B2*F2)))

Out_1=1/(1+EXP(-1*(I2*C2+J2*D2)))

Backpropagation Learning: Example (Target=0.5)



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	A	B	w1	w2	w3	w4	w5	w6	Out_h11	Out_h12	Out_1	Target	Err_1	Err_h11	Err_h12
2	0.35	0.9	0.3	0.9	0.1	0.8	0.4	0.6	0.680	0.664	0.690	0.5	-0.04068	-0.00241	-0.00793
3			0.272	0.873	0.099	0.798	0.397	0.593							

Err_1=K2*(1-K2)*(L2-K2)

$W1^{new}$=C2+M2*I2

$W2^{new}$=D2+M2*J2

Err_h11=I2*(1-I2)*(M2*C3)

Err_h12=J2*(1-J2)*(M2*D3)

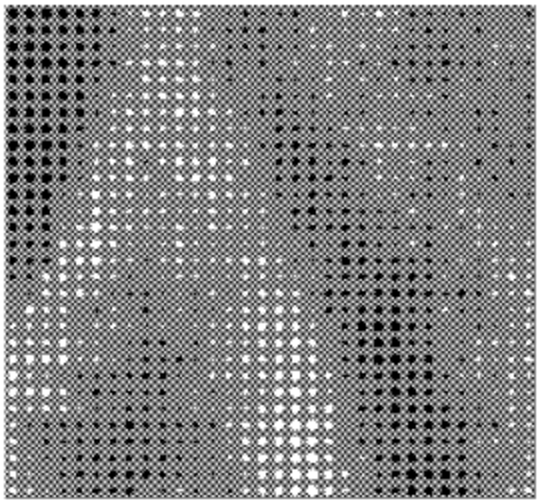
$W3^{new}$=E2+N2*A2

$W6^{new}$=H2+O2*B2

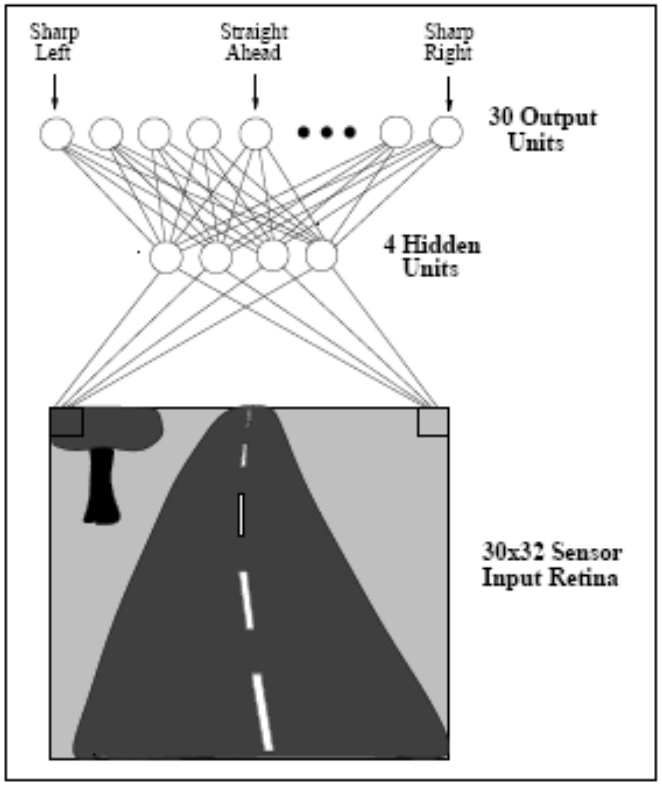
Artificial Neural Network

- ANN: most effective learning methods for complex real world sensor data
  - ALVINN, face recognition, handwritten recognition
  - Financial prediction
- Well suited to problems in which the training data corresponds to noisy, complex sensor data (cameras, microphones)
  - Input/output: discrete, real value, vector of value
  - Human readability of result is not important

ALVINN: 70 mph



camera



960 unit input → 30 output unit

Referensi

- Peter Andras, Artificial Neural Network: Introduction
- Prévotet Jean-Christophe, Tutorial on Neural Networks, University of Paris VI, FRANCE
- Burchan (bourch-khan) Bayazit, Machine Learning: Artificial Neural Networks, http://www.cse.wustl.edu/~bayazit/courses/cs527a/
- A Simple Introduction to Support Vector Machines; Martin Law; Michigan State Univ.

THANK YOU