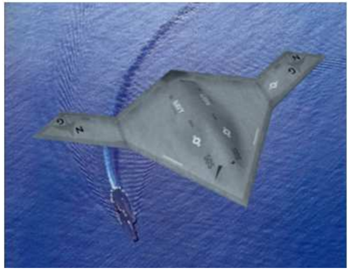
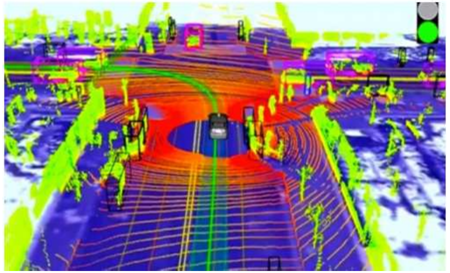
Intelligent Image and Graphics Processing

Neural Network

布树辉 bushuhui@nwpu.edu.cn
http://www.adv-ci.com



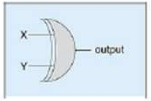


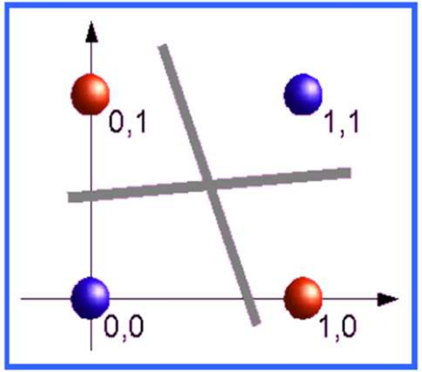
Learning highly non-linear functions

f: X → Y

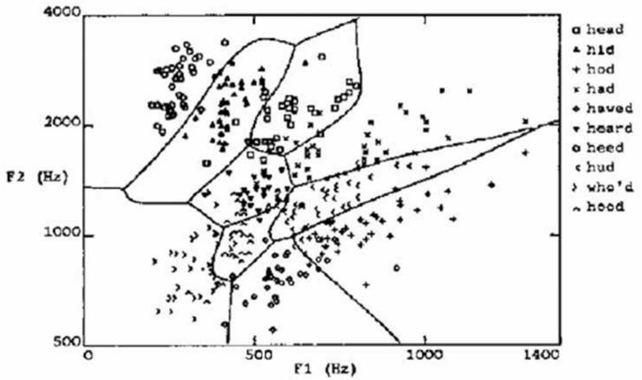
- f might be non-linear function
- X (vector of) continuous and/or discrete vars
- Y (vector of) continuous and/or discrete vars

The XOR gate



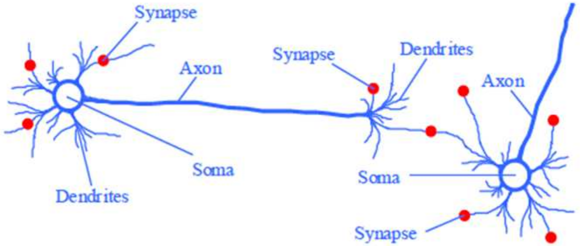


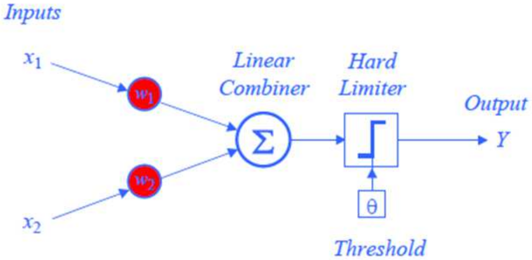
Speech recognition



Perceptron and neural nets

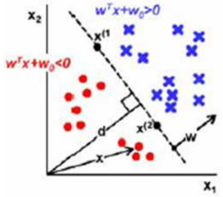
- From biological neuron to artificial neuron (perceptron)



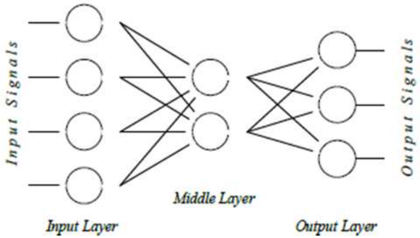


- Activation function

$$X = \sum_{i=1}^{n} x_i w_i \qquad Y = \begin{cases} +1, & \text{if } X \geq \omega_0 \\ -1, & \text{if } X < \omega_0 \end{cases}$$

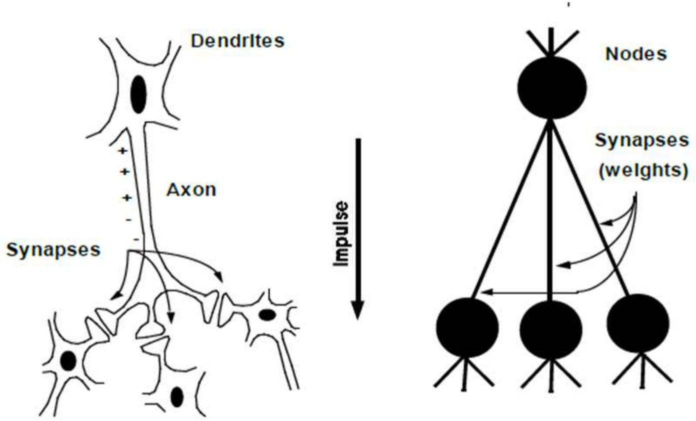


- Artificial neuron networks
  - supervised learning
  - gradient descent

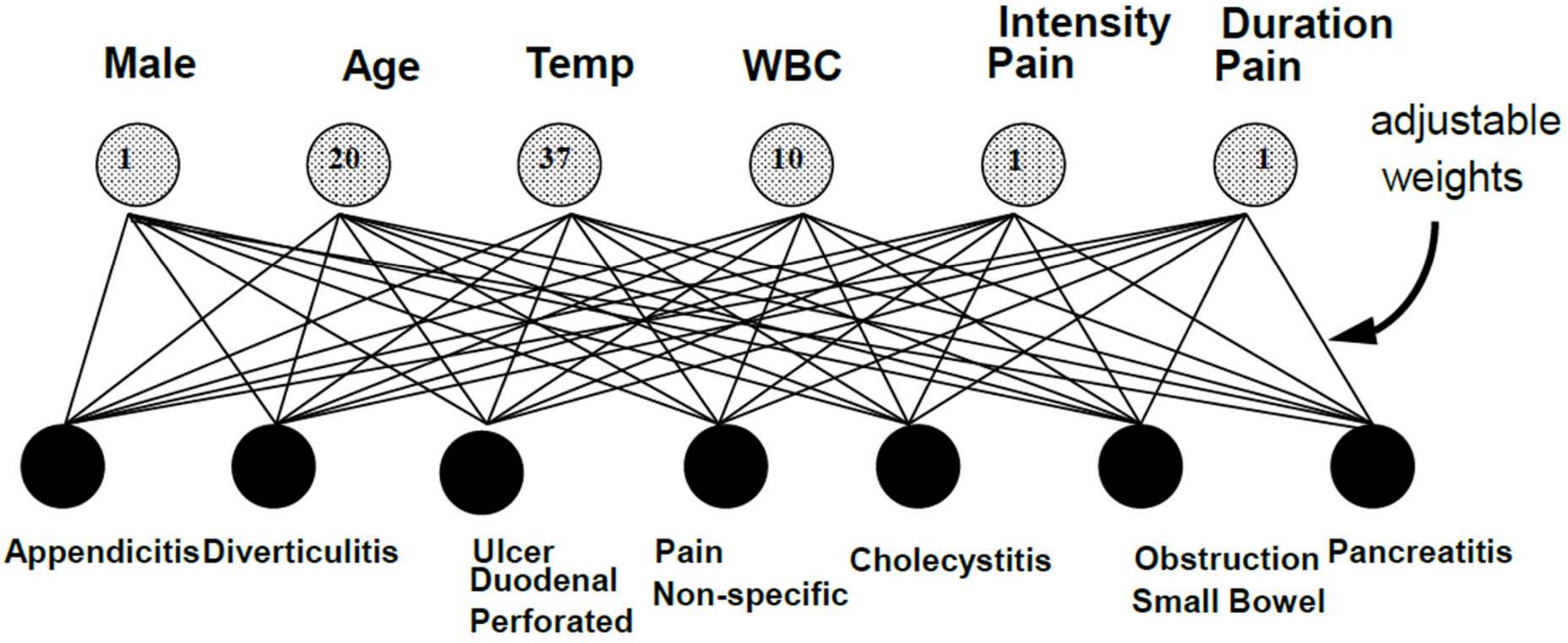


Connectionist models

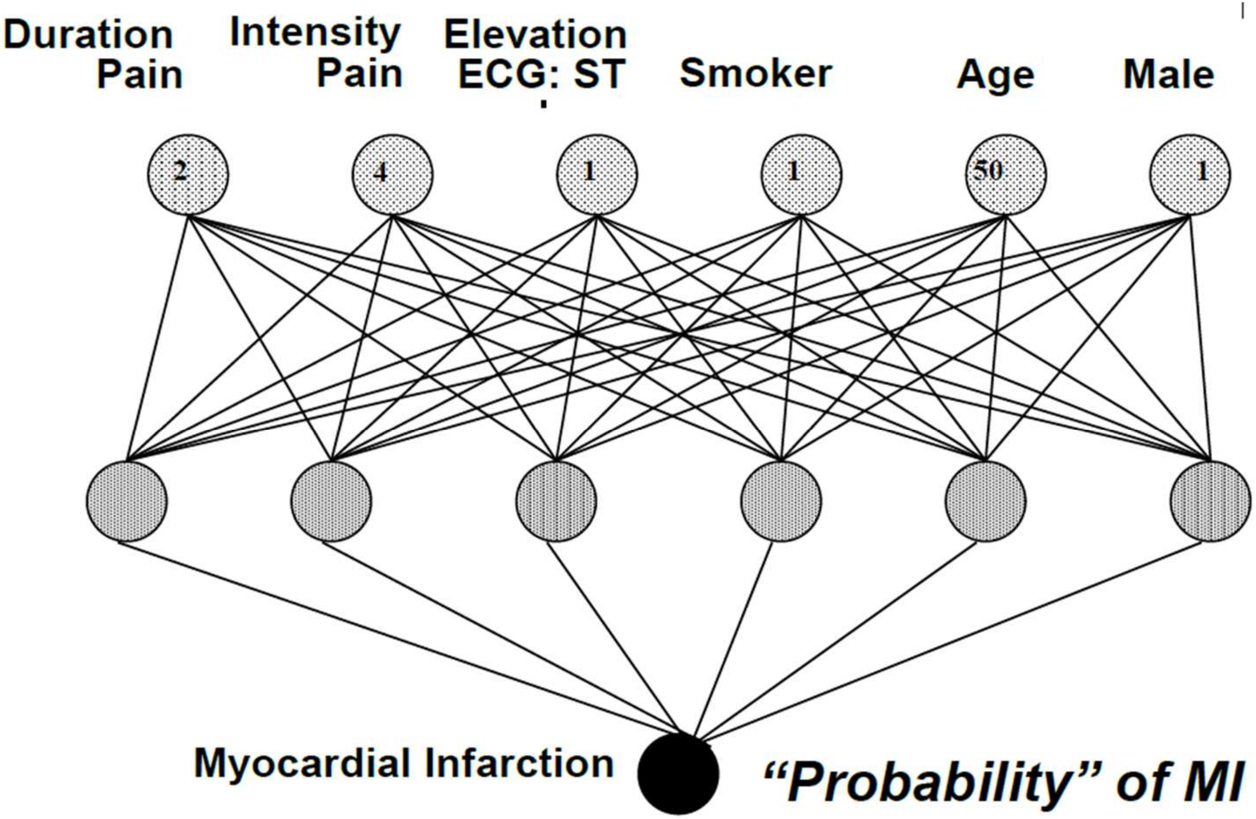
- Consider humans:
  - Neuron switching time
    ~ 0.001 second
  - Number of neurons
    ~ 10^{10}
  - Connections per neuron
    ~ $10^{4\text{-}5}$
  - Scene recognition time
    ~ 0.1 second
  - 100 inference steps doesn't seem like enough
    → much parallel computation
- Properties of artificial neural nets (ANN)
  - Many neuron-like threshold switching units
  - Many weighted interconnections among units
  - Highly parallel, distributed processes



Abdominal pain perceptron



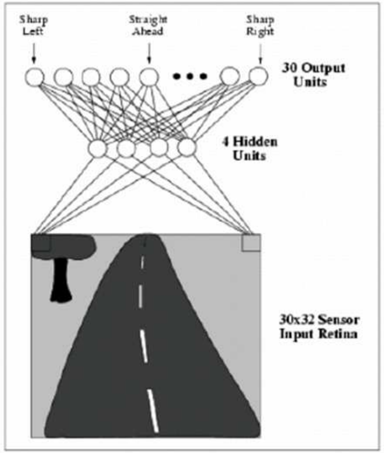
Myocardial infarction network

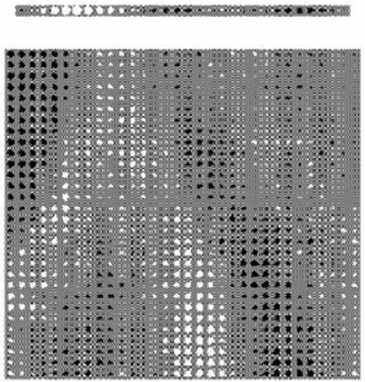


The “Driver” network

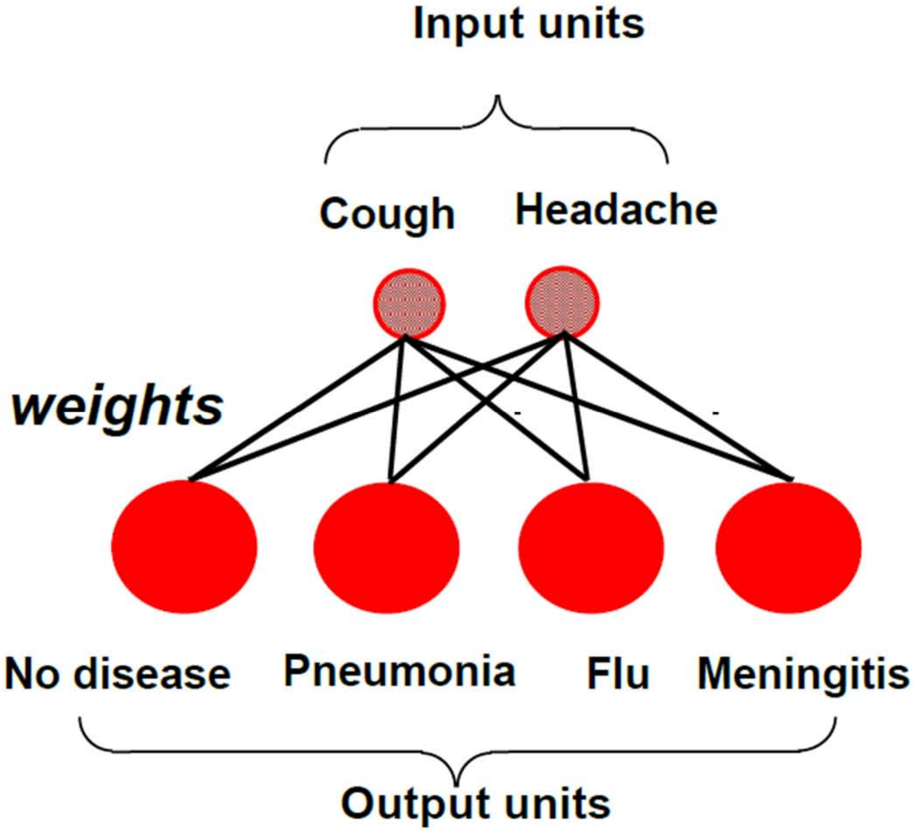
ALVINN

[Pomerleau 1993]





Perceptrons



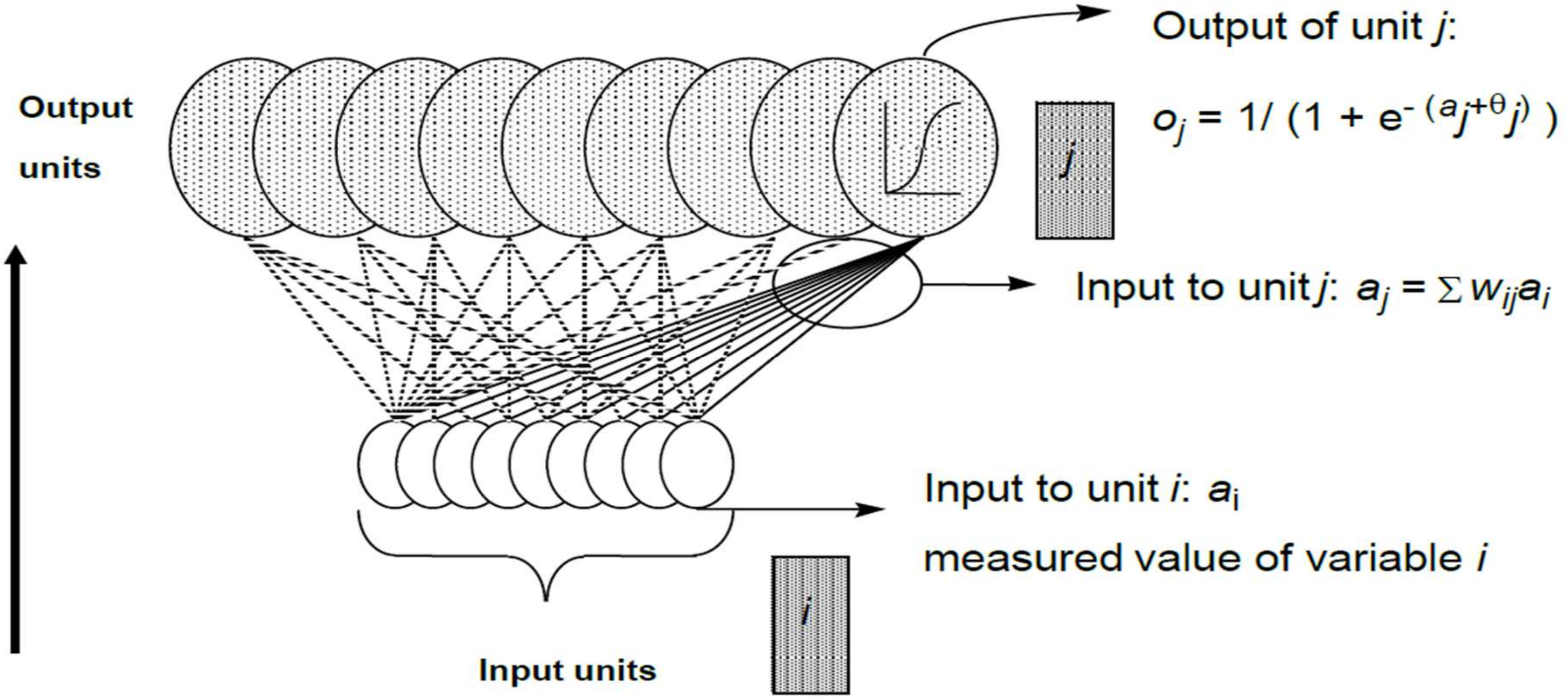
Δ rule

change weights to decrease the error

- what we got
what we wanted

error

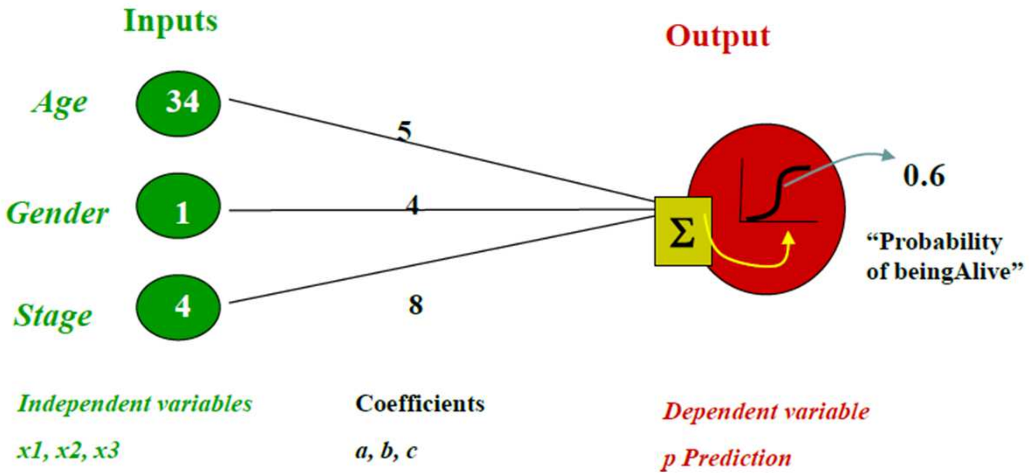
Perceptrons



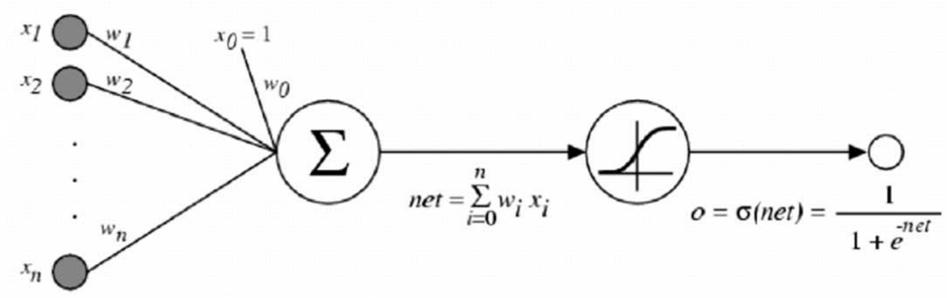
Jargon pseudo-correspondence

- Independent variable = input variable
- Dependent variable = output variable
- Coefficients = “weights”
- Estimates = “targets”

Logistic Regression Model (the sigmoid unit)



The perceptron learning algorithm

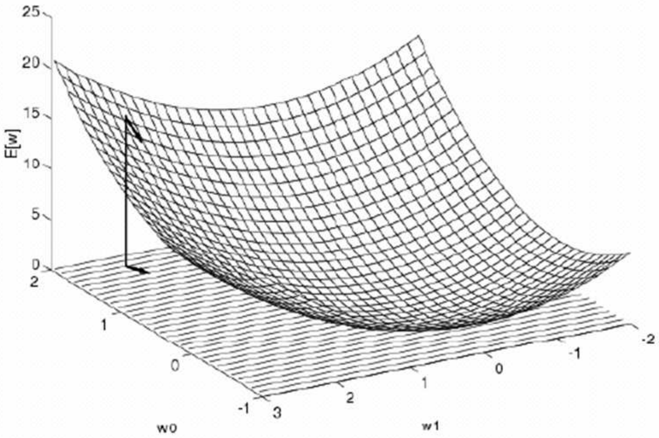


- Recall the nice property of sigmoid function $\frac{d\sigma}{dt} = \sigma(1-\sigma)$
- Consider regression problem f:X→Y , for scalar Y: $y = f(x) + \epsilon$
- Let's maximize the conditional data likelihood

$$\vec{w} = \arg\max_{\vec{w}} \ln \prod_i P(y_i | x_i; \vec{w})$$

$$\vec{w} = \arg\min_{\vec{w}} \sum_i \frac{1}{2}(y_i - \hat{f}(x_i; \vec{w}))^2$$

Gradient descent



$$\frac{\partial E[\vec{w}]}{\partial w_j} = \frac{\partial}{\partial w_i}\frac{1}{2}\sum(t_d - o_d)^2$$
$$=$$

Gradient

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

Training rule:

$$\Delta \vec{w} = -\eta \nabla E[\vec{w}]$$

i.e.,

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i}$$

The perceptron learning rules

$$
\begin{aligned}
\frac{\partial E_D[\vec{w}])}{\partial w_j} &= \frac{\partial}{\partial w_i}\frac{1}{2}\sum_d (t_d - o_d)^2 \\
&= \frac{1}{2}\sum_d 2(t_d - o_d)\frac{\partial}{\partial w_i}(t_d - o_d) \\
&= \sum_d (t_d - o_d)\Big(-\frac{\partial o_d}{\partial w_i}\Big) \\
&= -\sum_d (t_d - o_d)\frac{\partial o_d}{\partial net_i}\frac{\partial net_d}{\partial w_i} \\
&= -\sum_d (t_d - o_d)o_d(1 - o_d)x_d^i
\end{aligned}
$$

Batch mode:

Do until converge:

1. compute gradient $\nabla E_D[w]$
2. $\vec{w} = \vec{w} - \eta \nabla E_D[\vec{w}]$

Incremental mode:

Do until converge:

- **For each training example *d* in *D***

  1. compute gradient $\nabla E_d[w]$
  2. $\vec{w} = \vec{w} - \eta \nabla E_d[\vec{w}]$

  where

  $\nabla E_d[\vec{w}] = -(t_d - o_d)o_d(1 - o_d)\vec{x}_d$

MLE vs MAP

- Maximum conditional likelihood estimate

$$\vec{w} = \arg\max_{\vec{w}} \ln \prod_i P(y_i | x_i; \vec{w})$$

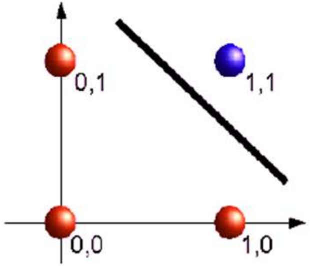
$$\boxed{\vec{w} \leftarrow \vec{w} + \eta \sum_d (t_d - o_d) o_d (1 - o_d) \vec{x}_d}$$

- Maximum a posteriori estimate

$$\vec{w} = \arg\max_{\vec{w}} \ln p(\vec{w}) \prod_i P(y_i | x_i; \vec{w})$$

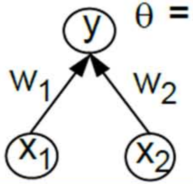
$$\boxed{\vec{w} \leftarrow \vec{w} + \eta \left(\sum_d (t_d - o_d) o_d (1 - o_d) \vec{x}_d - \lambda \vec{w} \right)}$$

What decision surface does a perceptron defines?



NAND

x	y	Z (color)
0	0	1
0	1	1
1	0	1
1	1	0



$\theta = 0.5$

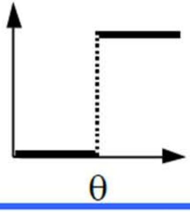
$f(x_1w_1 + x_2w_2) = y$

$f(0w_1 + 0w_2) = 1$

$f(0w_1 + 1w_2) = 1$

$f(1w_1 + 0w_2) = 1$

$f(1w_1 + 1w_2) = 0$

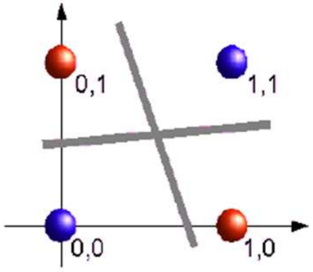


$$f(a) = \begin{cases} 1, \text{ for } a > \theta \\ 0, \text{ for } a \leq \theta \end{cases}$$

some possible values for w_1 and w_2

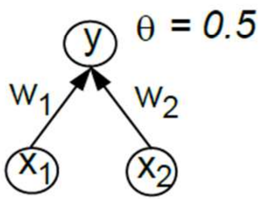
w_1	w_2
0.20	0.35
0.20	0.40
0.25	0.30
0.40	0.20

What decision surface does a perceptron defines?



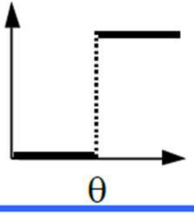
NAND

x	y	Z (color)
0	0	0
0	1	1
1	0	1
1	1	0



$$f(x_1w_1 + x_2w_2) = y$$

$$f(0w_1 + 0w_2) = 0$$
$$f(0w_1 + 1w_2) = 1$$
$$f(1w_1 + 0w_2) = 1$$
$$f(1w_1 + 1w_2) = 0$$

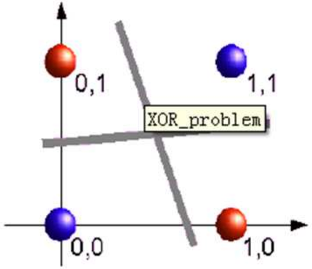


$$f(a) = \begin{cases} 1, \text{ for } a > \theta \\ 0, \text{ for } a \leq \theta \end{cases}$$

some possible values for w_1 and w_2

w_1	w_2

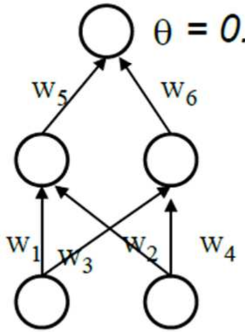
What decision surface does a perceptron defines?

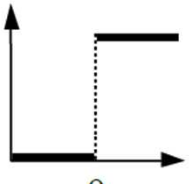


NAND

x	y	Z (color)
0	0	0
0	1	1
1	0	1
1	1	0

θ = *0.5 for all units*



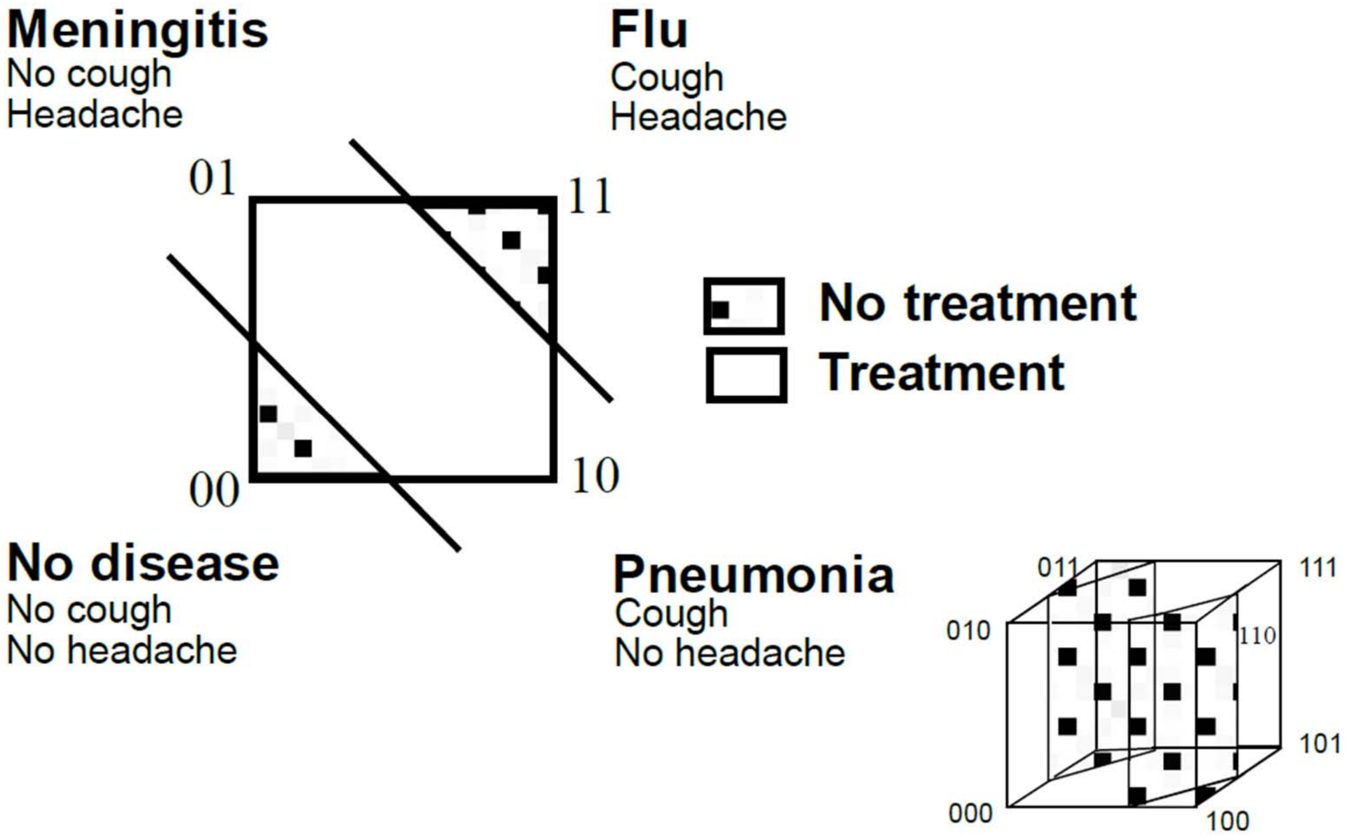


$$f(a) = \begin{cases} 1, \text{ for } a > \theta \\ 0, \text{ for } a \leq \theta \end{cases}$$

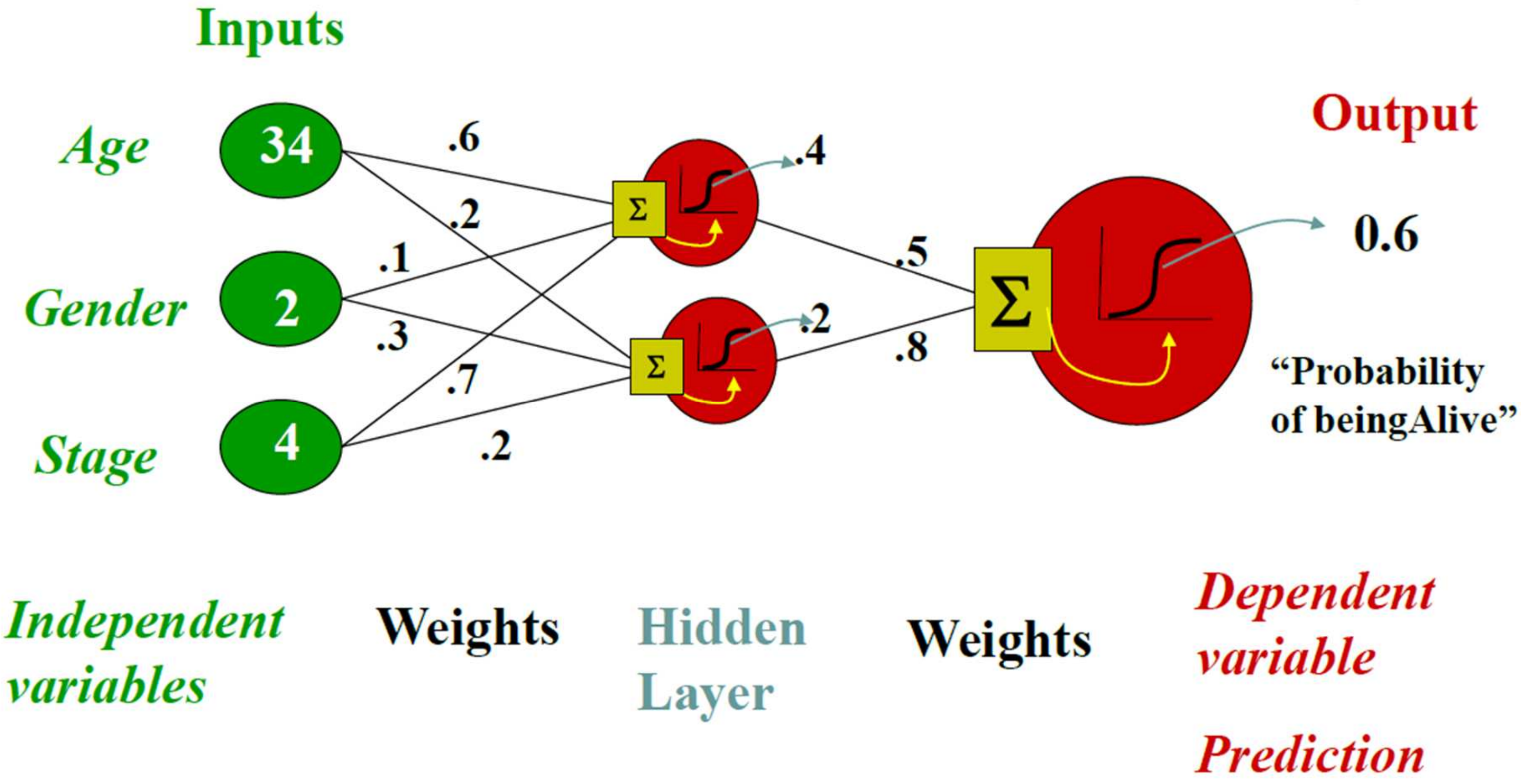
a possible set of values for $(w_1, w_2, w_3, w_4, w_5, w_6)$:

$(0.6,-0.6,-0.7,0.8,1,1)$

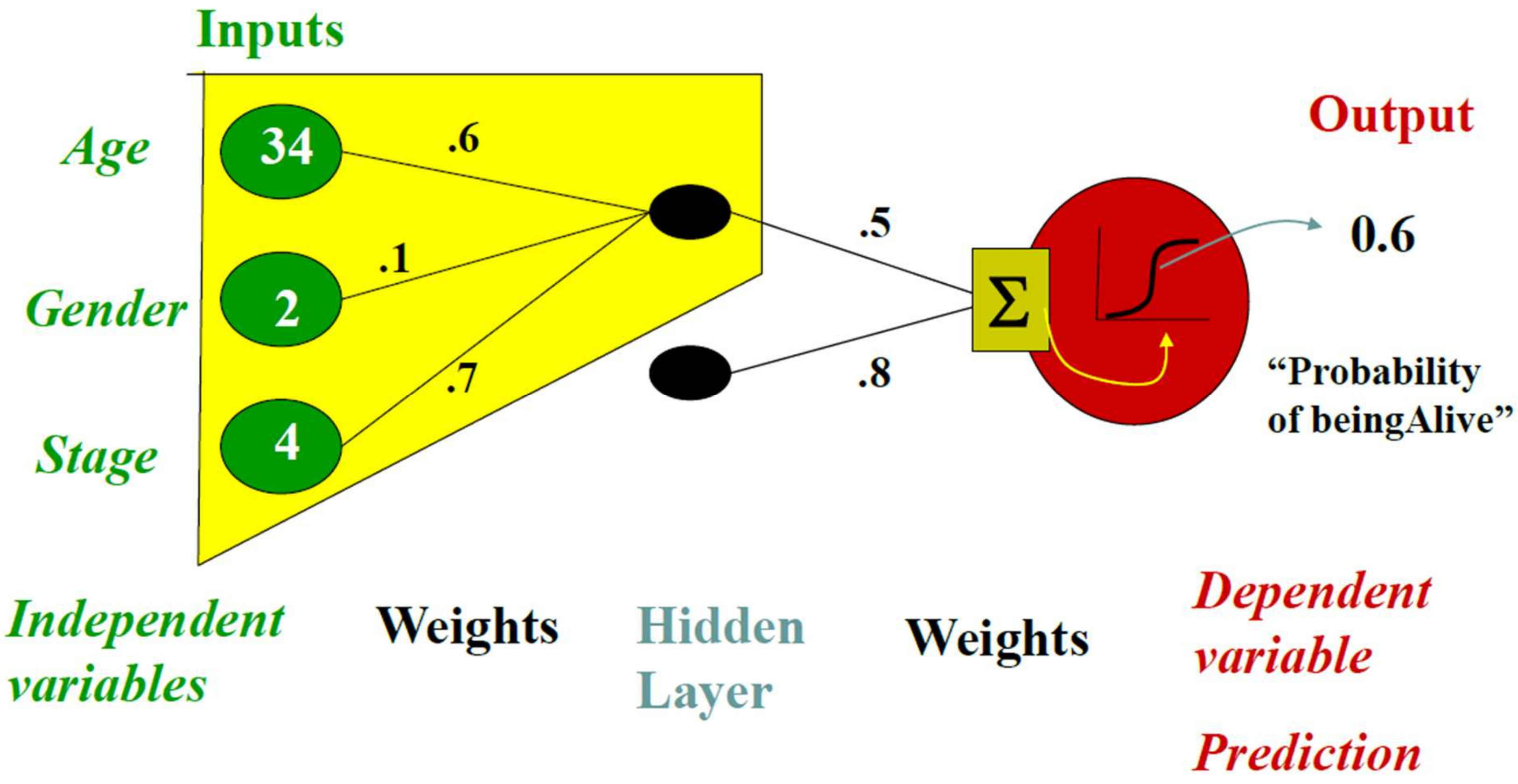
Non linear separation



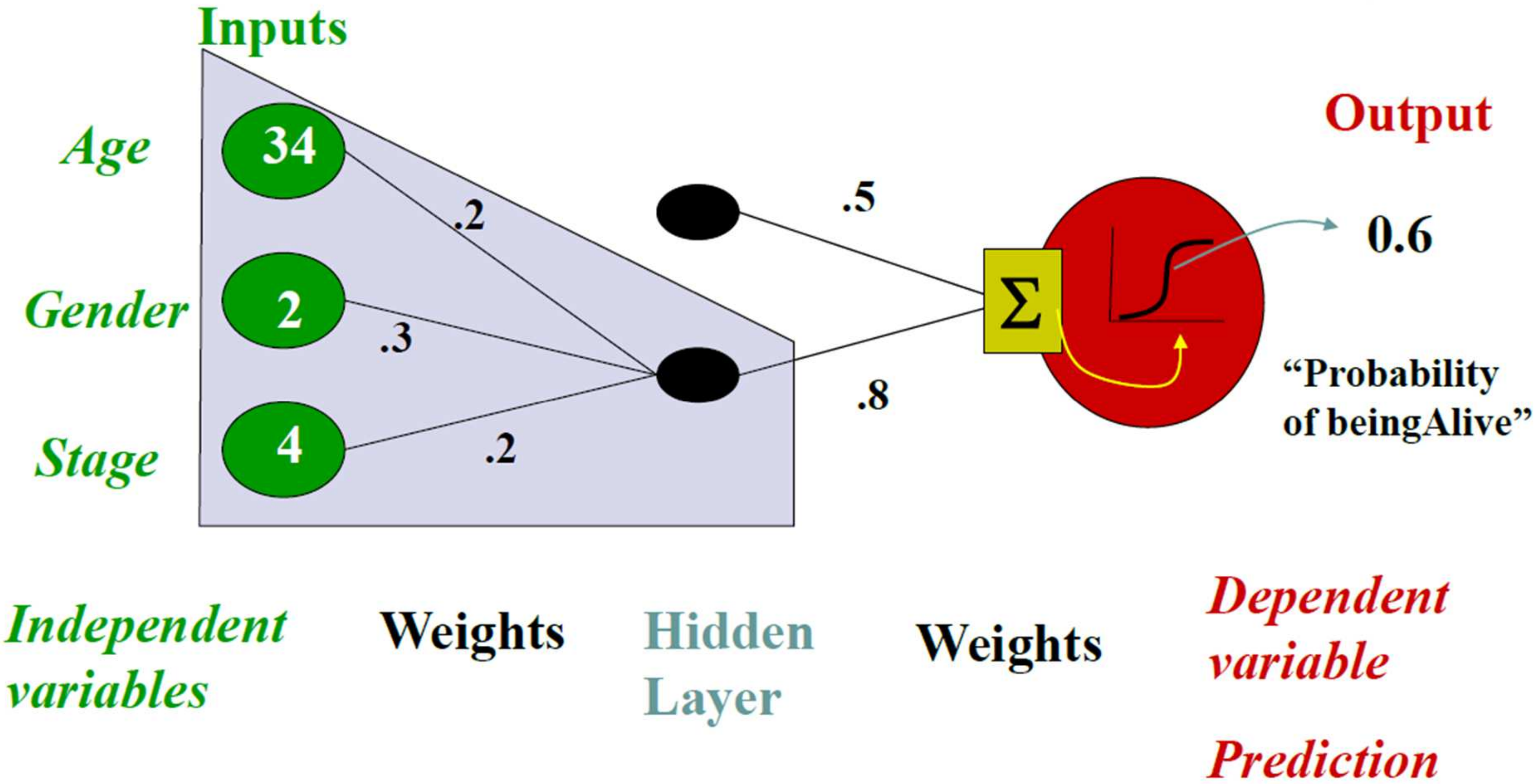
Neural network model



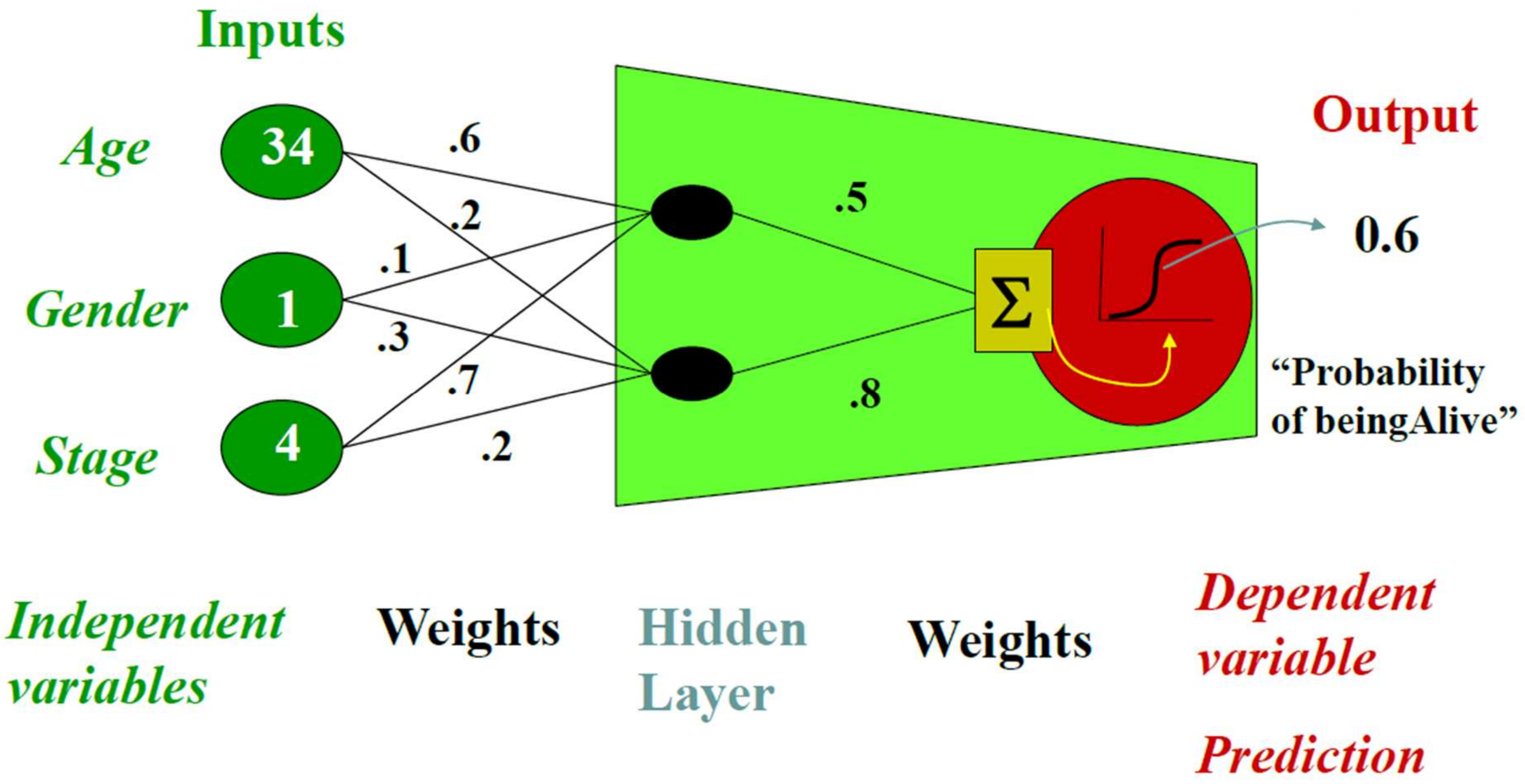
Combined logistic models



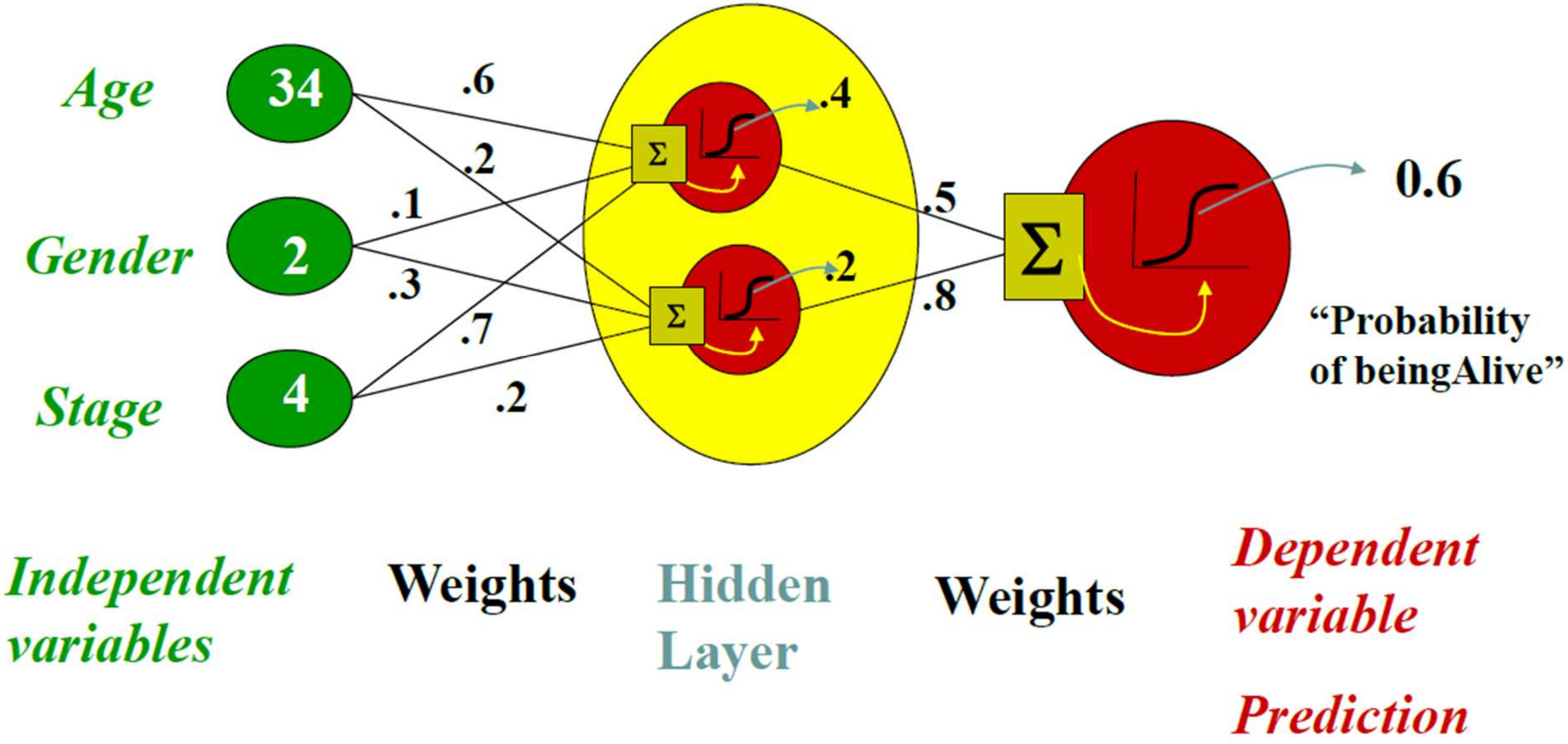
Combined logistic models



Combined logistic models

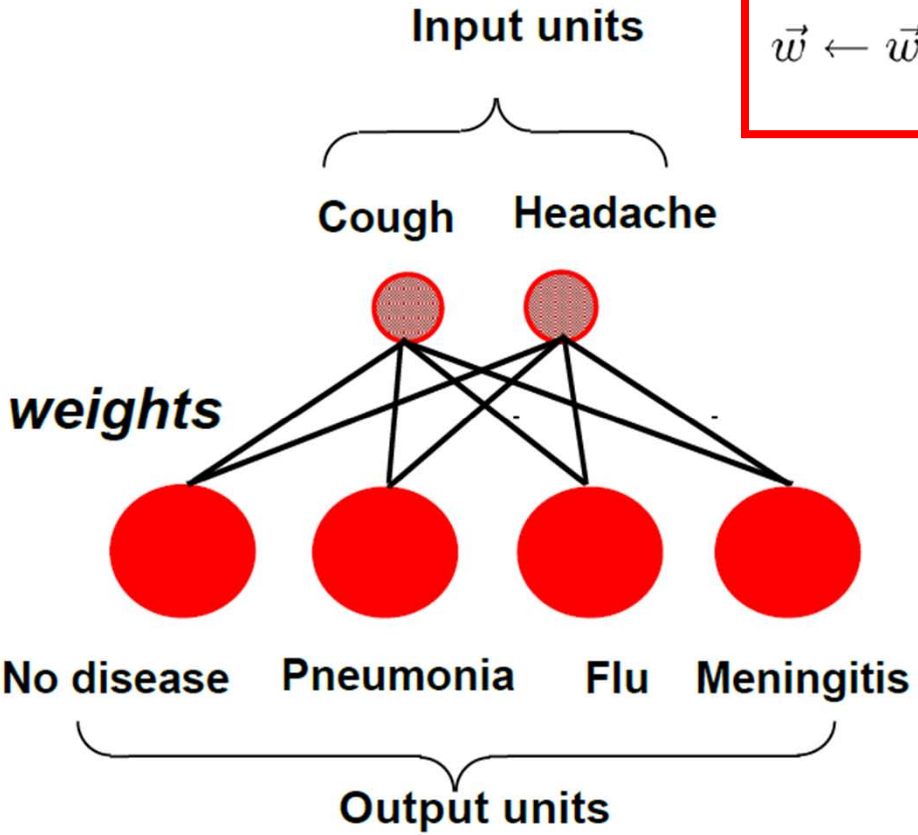


Not really, no target for hidden units



Perceptrons

$$\vec{w} \leftarrow \vec{w} + \eta \sum_{d} (t_d - o_d) o_d (1 - o_d) \vec{x}_d$$



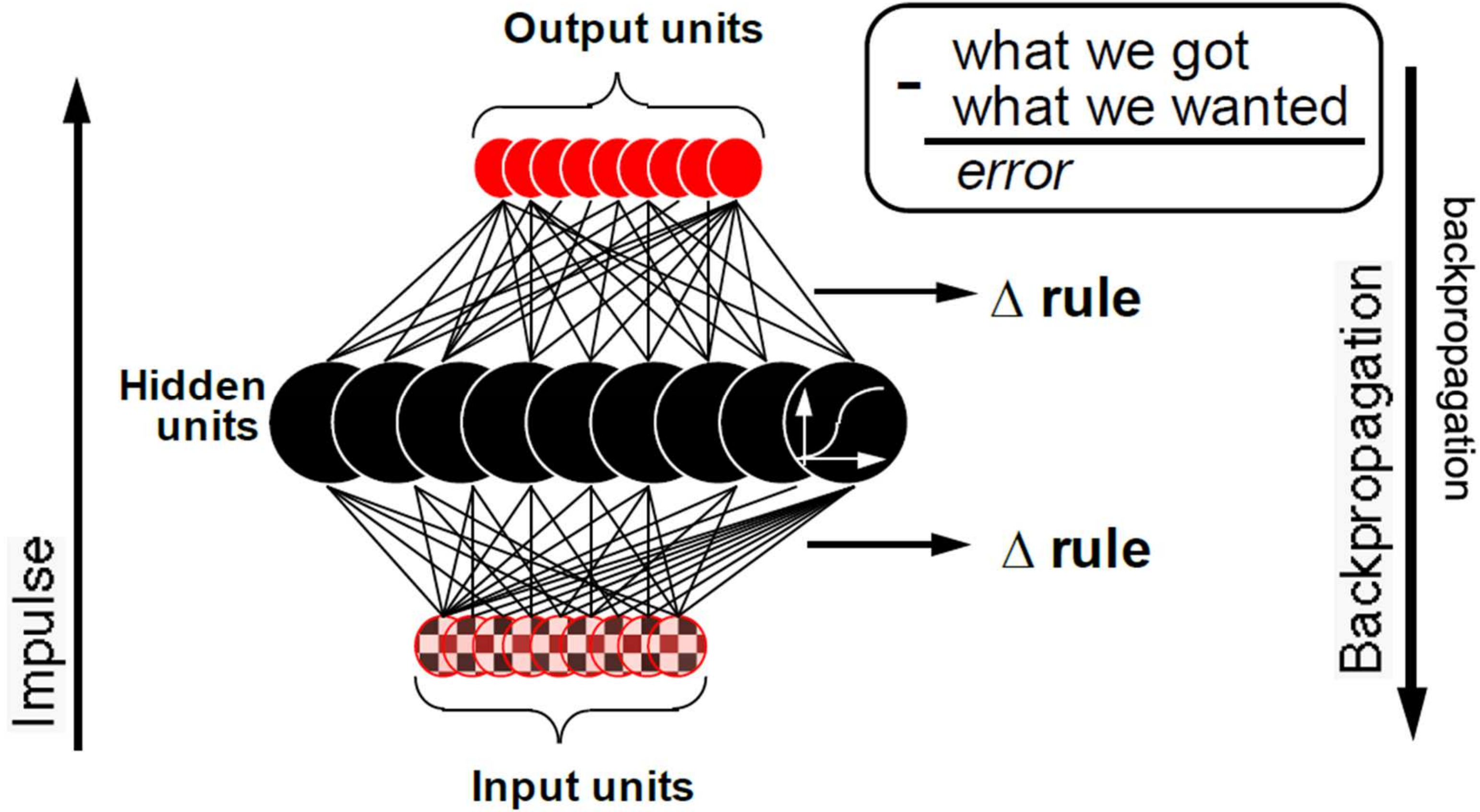
Δ rule

change weights to decrease the error

- what we got
what we wanted

error

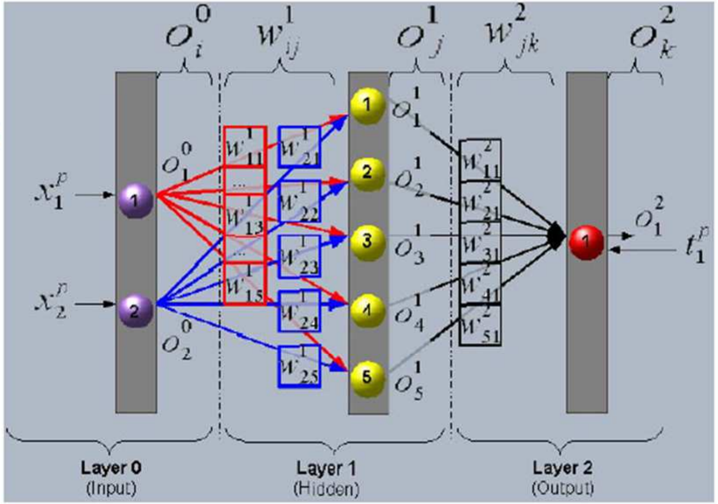
Hidden units and backpropagation



Backpropagation algorithm

- Initialize all weights to small random numbers

  Until convergence, Do



1. Input the training example to the network and compute the network outputs

1. For each output unit k

$$\delta_k \leftarrow o_k^2(1-o_k^2)(t-o_k^2)$$

2. For each hidden unit h

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

3. Undate each network weight $w_{i,j}$

$$w_{i,j} \leftarrow w_{i,j} + \Delta w_{i,j} \quad \text{where} \quad \Delta w_{i,j} = \eta\delta_j x^j$$

More on backpropagation

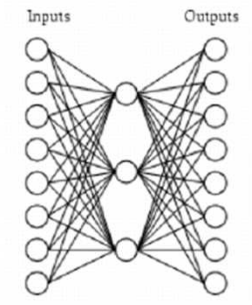
- It is doing gradient descent over entire network weight vector
- Easily generalized to arbitrary directed graphs
- Will find a local, not necessarily global error minimum
  - In practice, often works well (can run multiple times)
- Often include weight *momentum* α

$$\Delta w_{i,j}(t) = \eta \delta_j x_{i,j} + \alpha \Delta w_{i,j}(t-1)$$

- Minimizes error over *training* examples
  - Will it generalize well to subsequent testing examples?
- Training can take thousands of iterations, → very slow!
- Using network after training is very fast

Learning hidden layer representation

- A network:



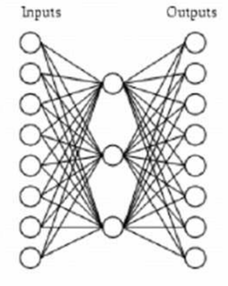
- A target function:

Input		Output
10000000	$\rightarrow$	10000000
01000000	$\rightarrow$	01000000
00100000	$\rightarrow$	00100000
00010000	$\rightarrow$	00010000
00001000	$\rightarrow$	00001000
00000100	$\rightarrow$	00000100
00000010	$\rightarrow$	00000010
00000001	$\rightarrow$	00000001

- Can this be learned?

Learning hidden layer representation

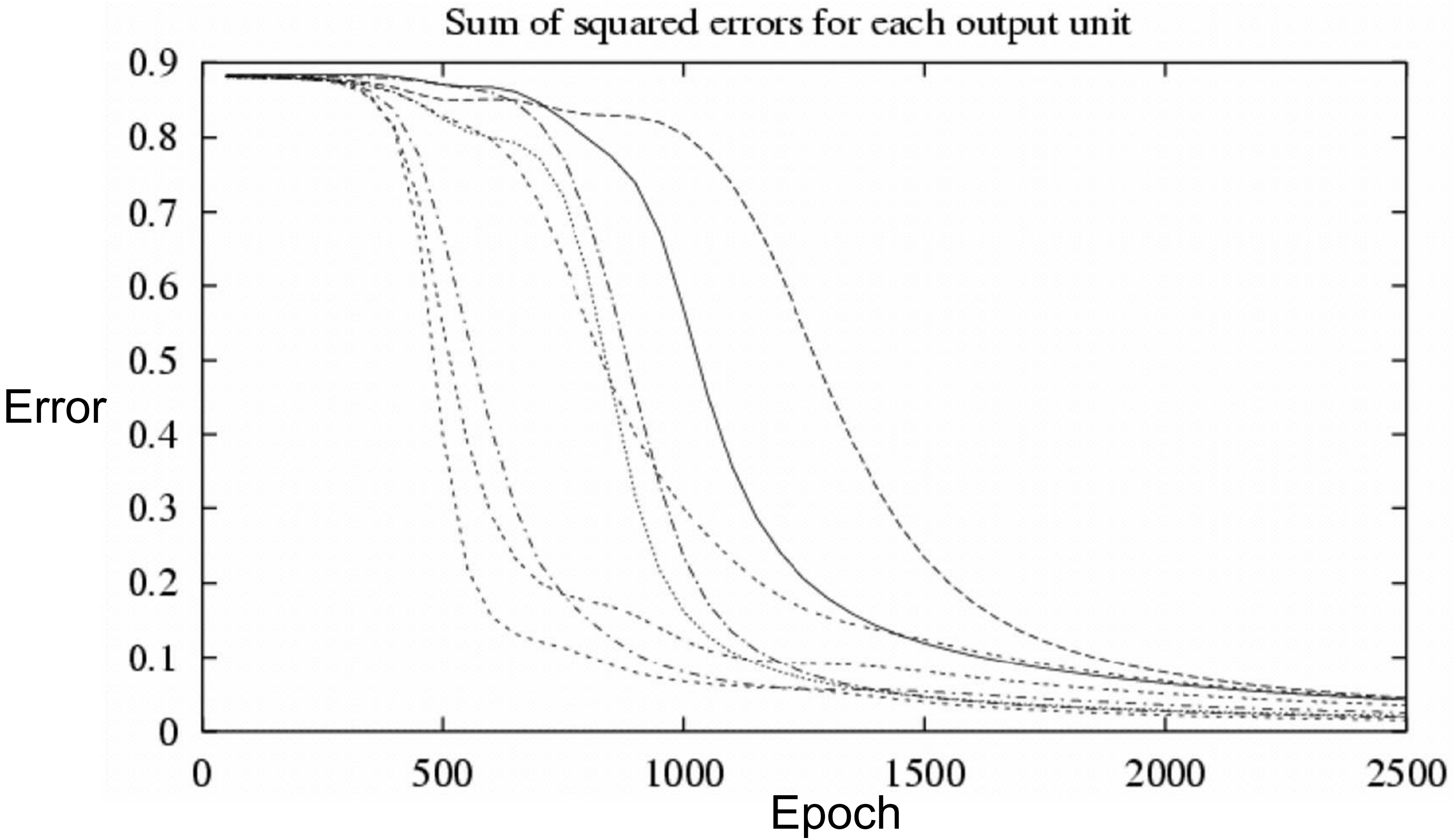
- A network:



- 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

Training



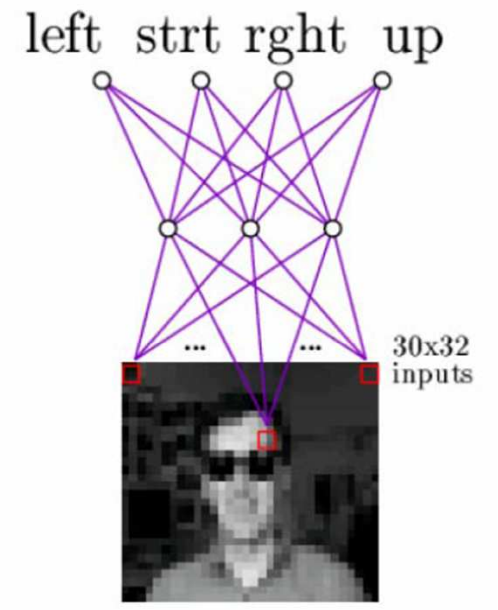
Sum of squared errors for each output unit
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
500
1000
1500
2000
2500
Error
Epoch

Expressive capabilities of ANNs

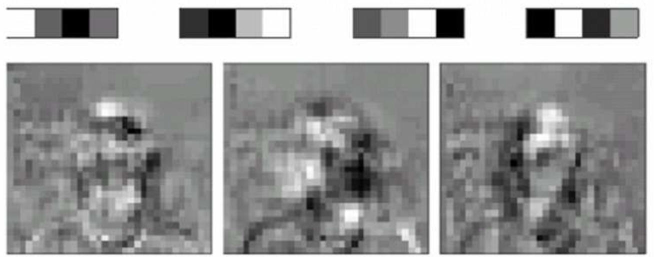
- Boolean functions:
  - Every Boolean function can be represented by network with single hidden layer
  - But might require exponential (in number of inputs) hidden units
- Continuous functions:
  - Every bounded continuous function can be approximated with arbitrary small error, by network with one hidden layer [Cybenko 1989; Hornik et al 1989]
  - Any function can be approximated to arbitrary accuracy by a network with two hidden layers [Cybenko 1988].

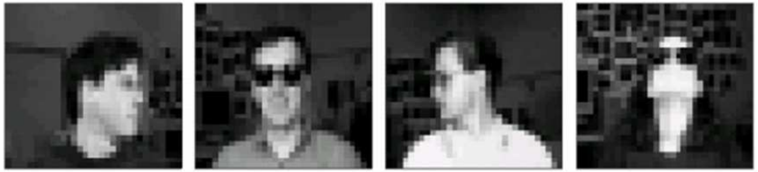
Application: ANN for face recognition

- The model



- The learned hidden unit weights

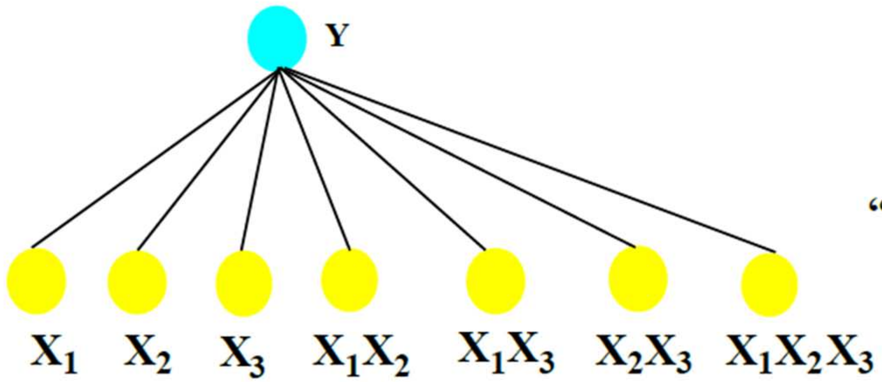


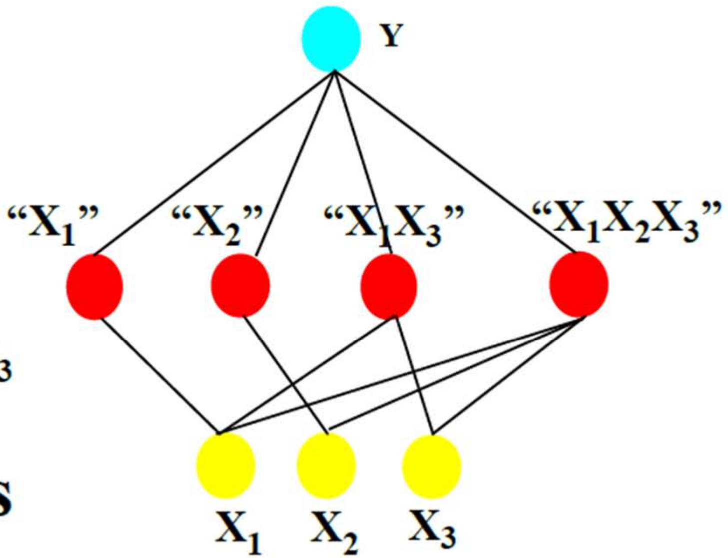


Typical input images

http://www.cs.cmu.edu/~tom/faces.html

Regression vs Neural Networks

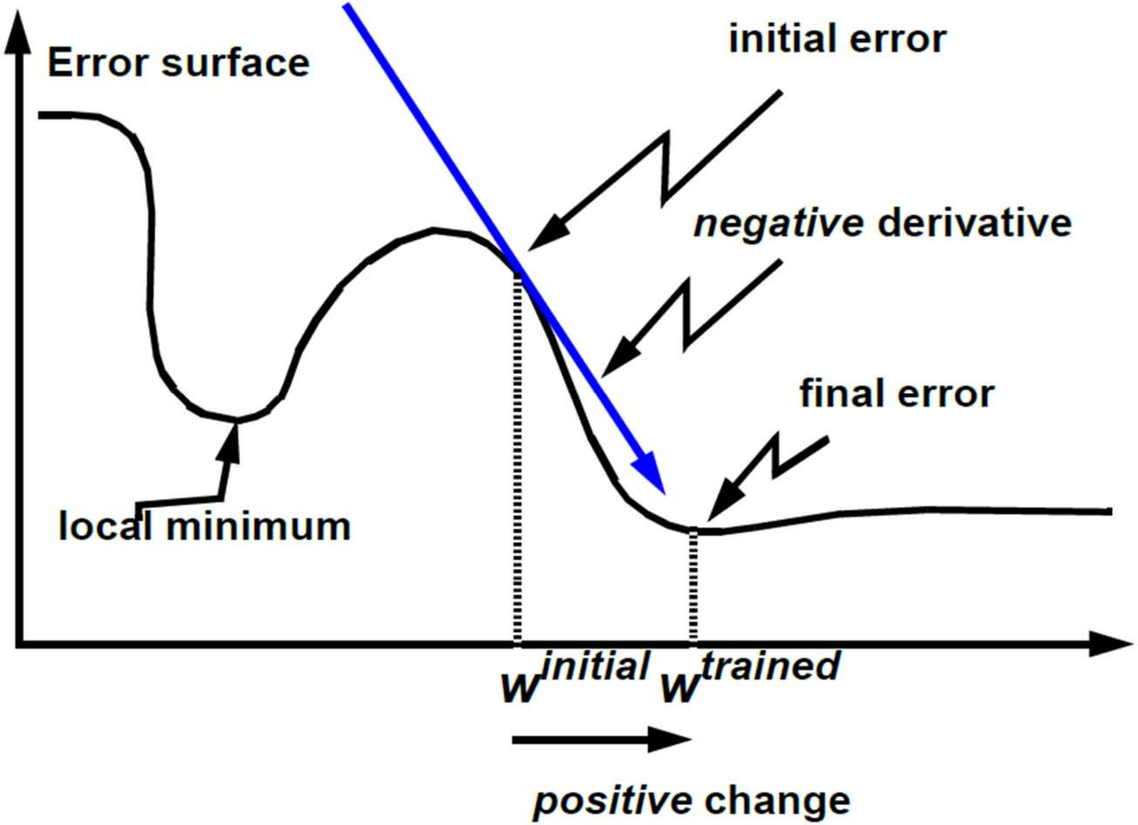




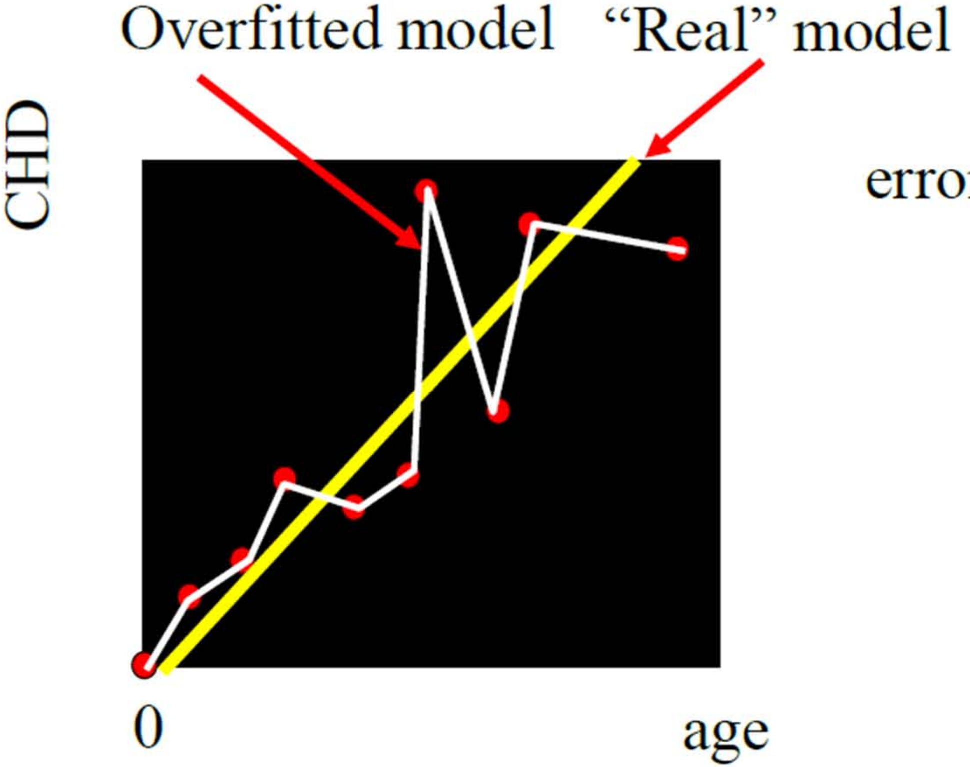
$(2^3\text{-}1)$ possible combinations

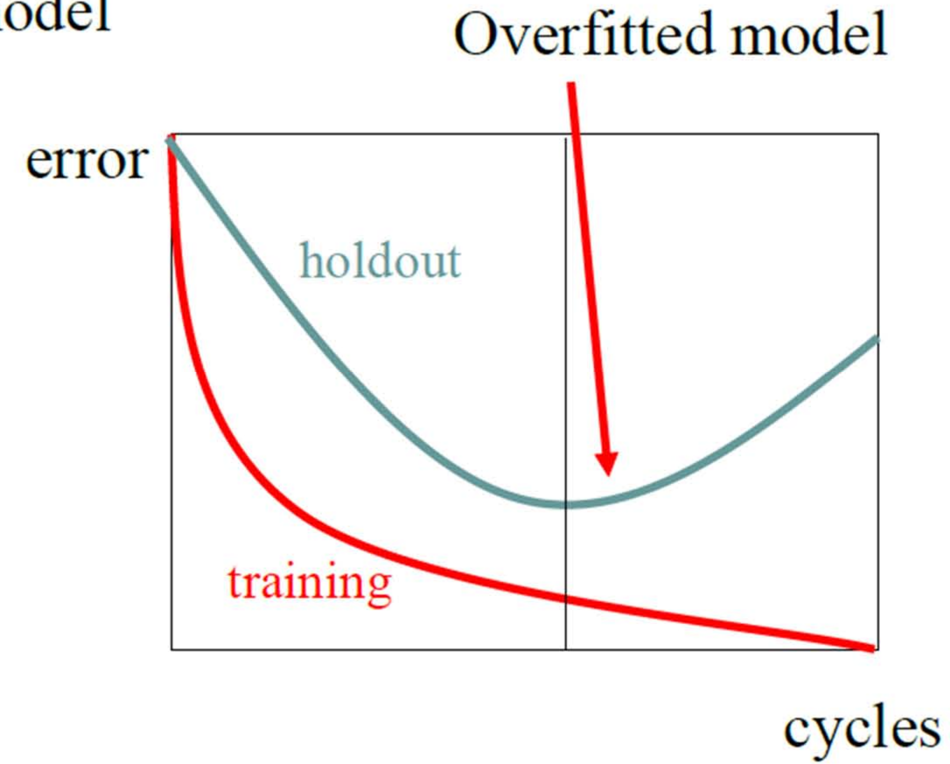
$$Y = a(X_1) + b(X_2) + c(X_3) + d(X_1X_2) + \ldots$$

Minimizing the Error



Overfitting in Neural Networks





Alternative error functions

- Penalize large weights:

$$E[\vec{w}] \equiv \frac{1}{2}\sum_{d\in D}\sum_{k\in outputs}(t_{k,d}-o_{k,d})^2+\gamma\sum_{i,j}w_{j,i}^2$$

- Training on target slopes as well as values

$$E[\vec{w}] \equiv \frac{1}{2}\sum_{d\in D}\sum_{k\in outputs}(t_{k,d}-o_{k,d})^2+\mu\sum_{,j\in inputs}\left(\frac{\partial t_{k,d}}{\partial x_d^j}-\frac{\partial o_{k,d}}{\partial x_d^j}\right)$$

- Tie together weights
  - E.g., in phoneme recognition

Artificial Neural Network – What you should know

- Highly expressive non-linear functions
- Highly parallel network of logistic function units
- Minimizing sum of squared training errors
  - Gives MLE estimates of network weights if we assume zero mean Gaussian noise on output values
- Minimizing sum of sq errors plus weight squared (regularization)
  - MAP estimates assuming weight priors are zero mean Gaussian
- Gradient descent as training procedure
  - How to derive your own gradient descent procedure
- Discover useful representations at hidden units
- Local minima is greatest problem
- Overfitting, regularization, early stopping

Artificial Neural Network – What you should know

- Highly expressive non-linear functions
- Highly parallel network of logistic function units
- Minimizing sum of squared training errors
  - Gives MLE estimates of network weights if we assume zero mean Gaussian noise on output values
- Minimizing sum of sq errors plus weight squared (regularization)
  - MAP estimates assuming weight priors are zero mean Gaussian
- Gradient descent as training procedure
  - How to derive your own gradient descent procedure
- Discover useful representations at hidden units
- Local minima is greatest problem
- Overfitting, regularization, early stopping