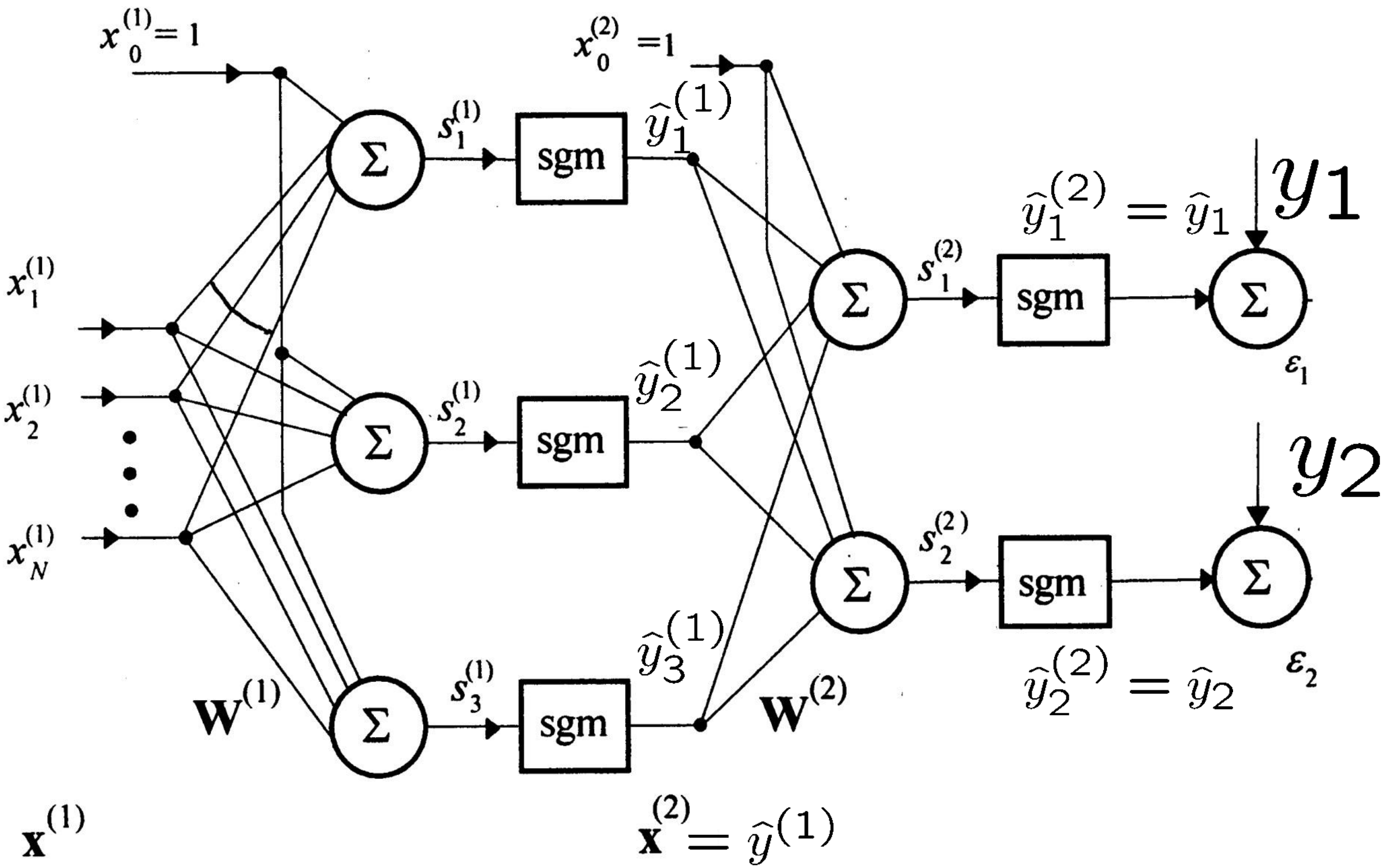
Introduction to Machine Learning

Multilayer Perceptron

Barnabás Póczos

The Multilayer Perceptron

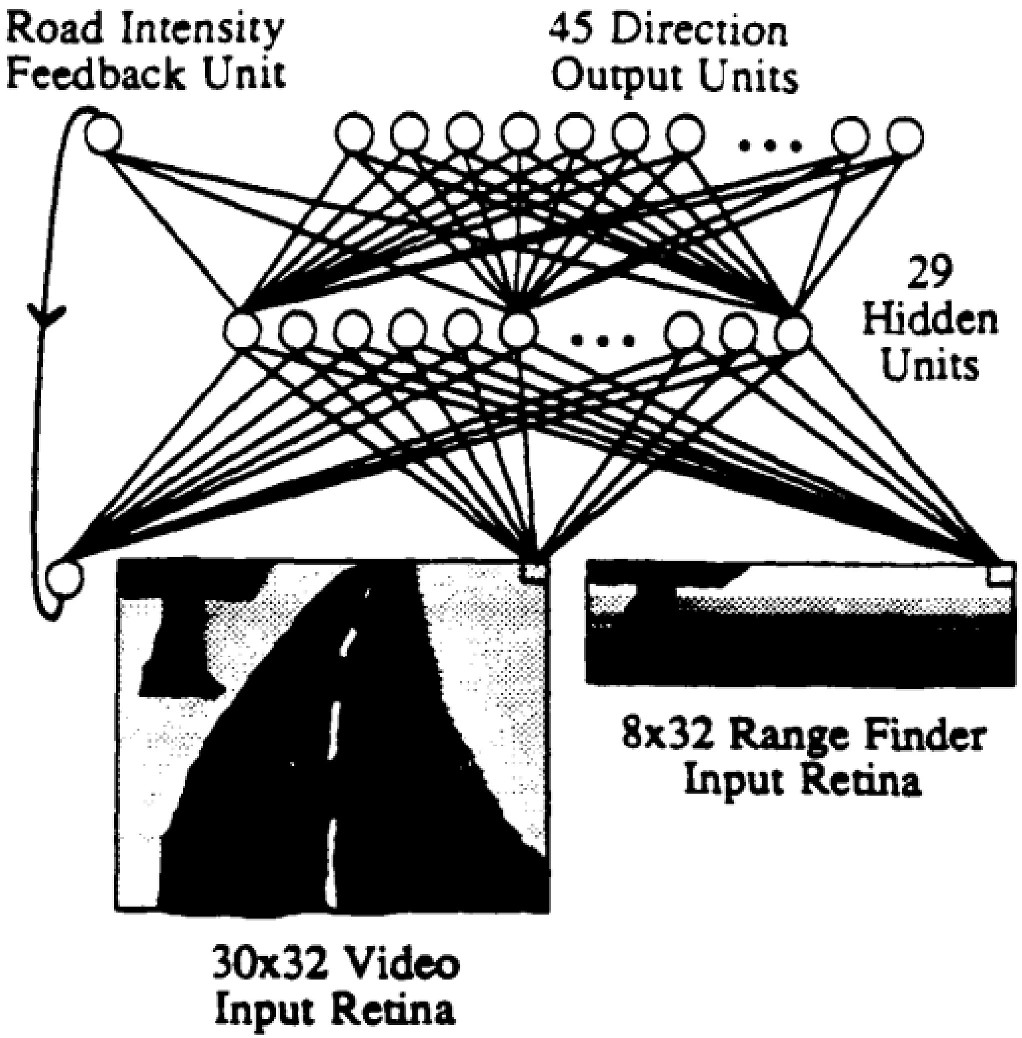
Multilayer Perceptron



ALVINN: AN AUTONOMOUS LAND VEHICLE IN A NEURAL NETWORK

Dean A. Pomerleau, Carnegie Mellon University, 1989

Training: using simulated road generator



Gradient Descent

Consider the unconstrained minimization of $f : \mathbb{R}^n \to \mathbb{R}$, differentiable function.

We want to solve:

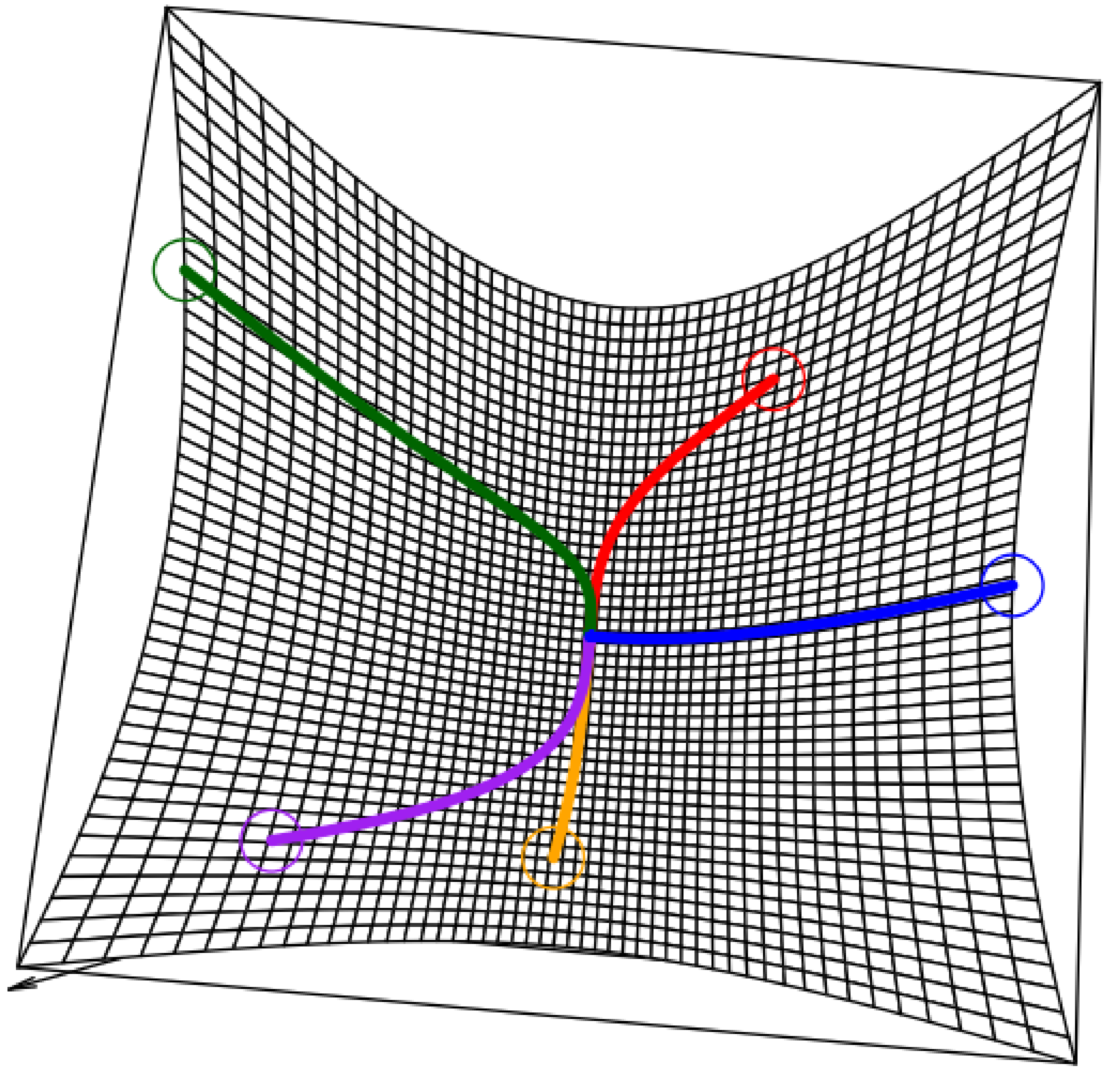
$$\min_{x \in \mathbb{R}^n} f(x),$$

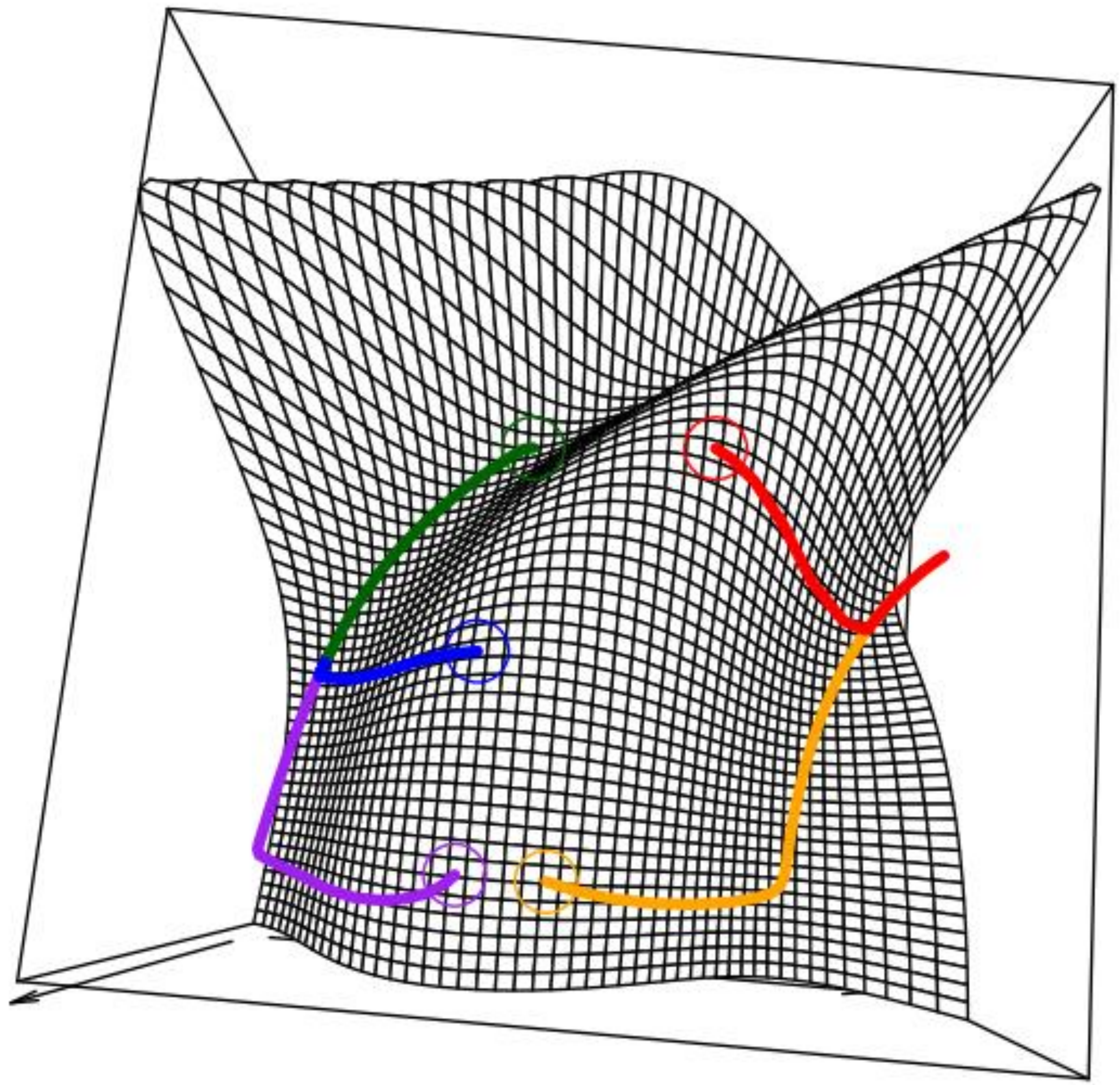
i.e., find $x^\star$ such that $f(x^\star) = \min_x f(x)$

Gradient descent: choose initial $x^{(0)} \in \mathbb{R}^n$, repeat:

$$x^{(k)} = x^{(k-1)} - t_k \cdot \nabla f(x^{(k-1)}), \quad k = 1, 2, 3, \ldots$$

Stop at some point

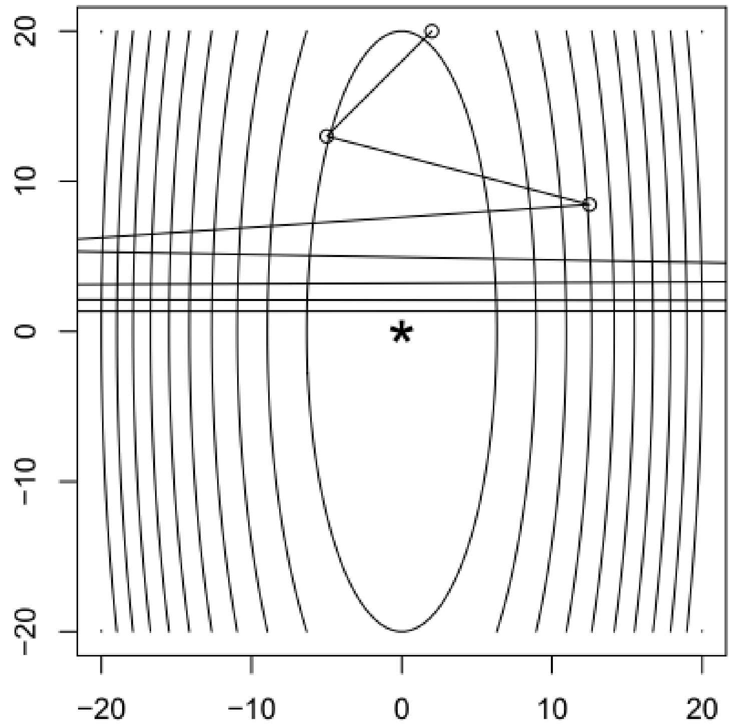




Fixed step size can be too big

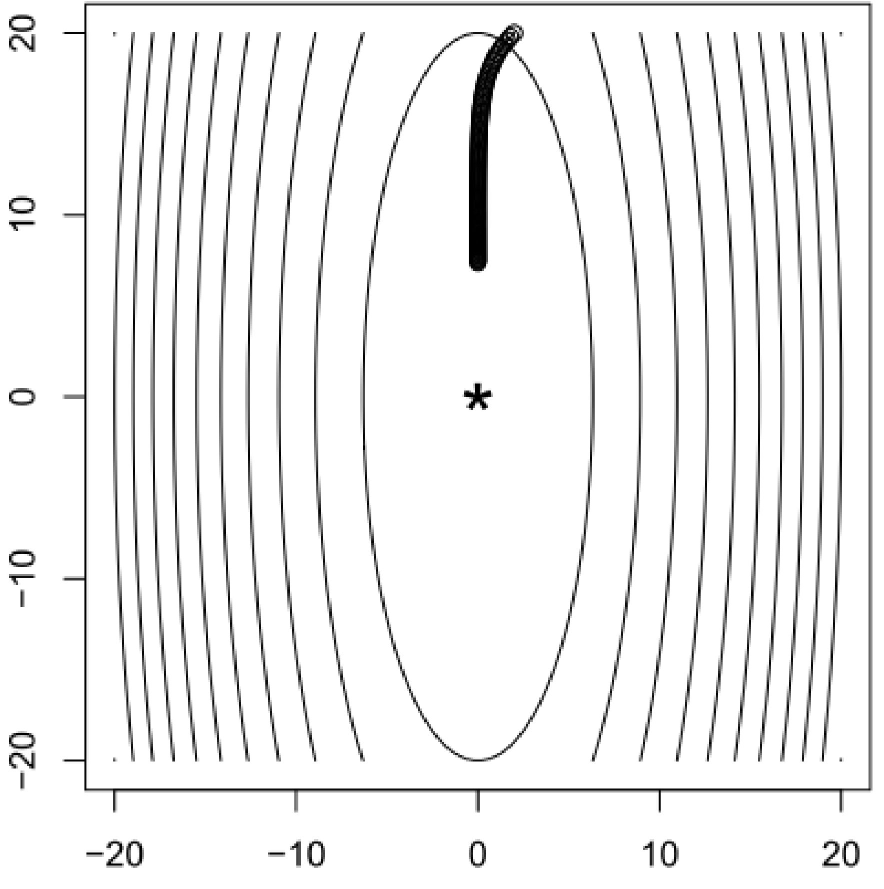
Simply take $t_k = t$ for all $k = 1, 2, 3, \ldots$
It can diverge if t is too big.

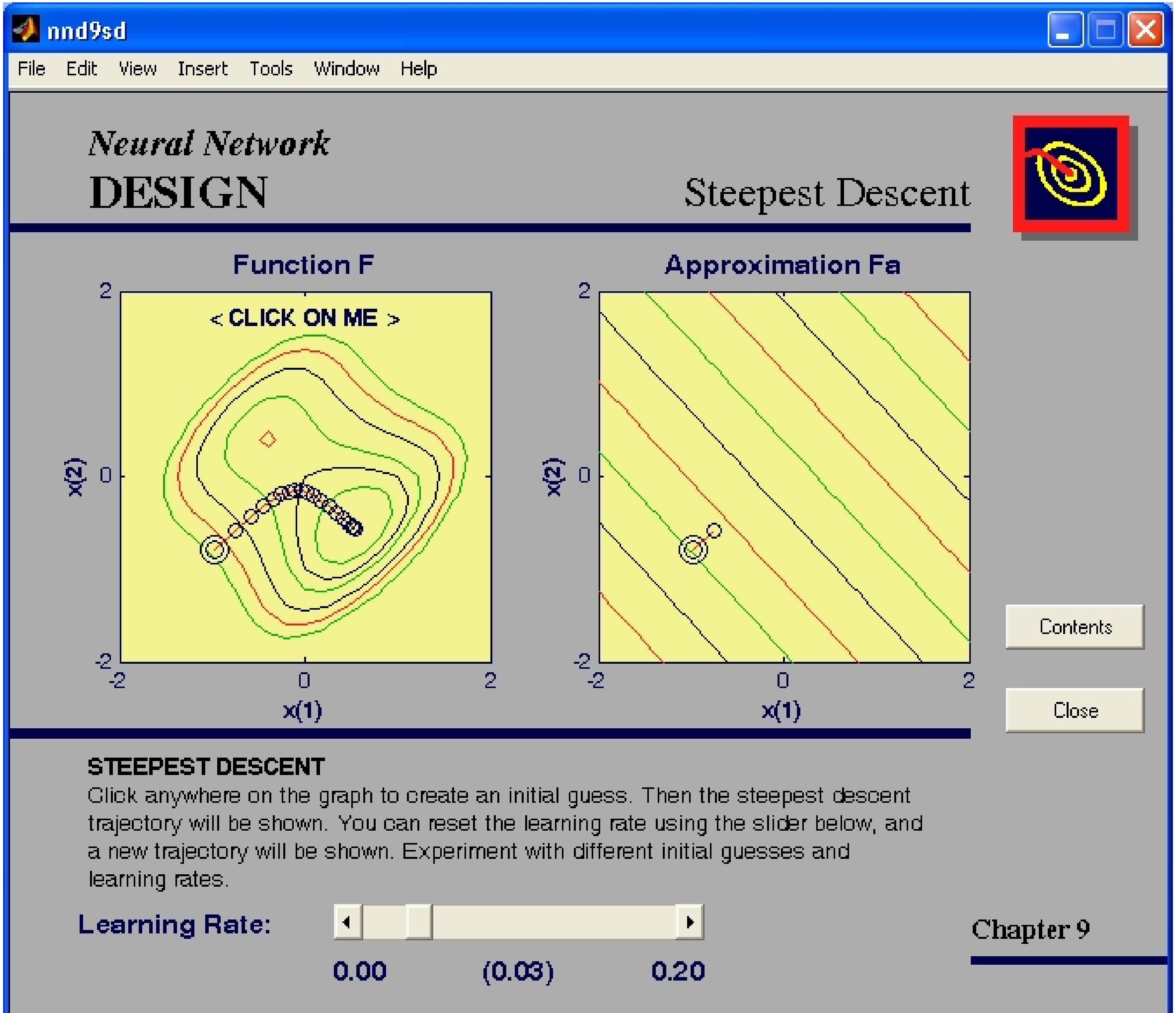
Consider $f(x) = (10x_1^2 + x_2^2)/2$, gradient descent after 8 steps:

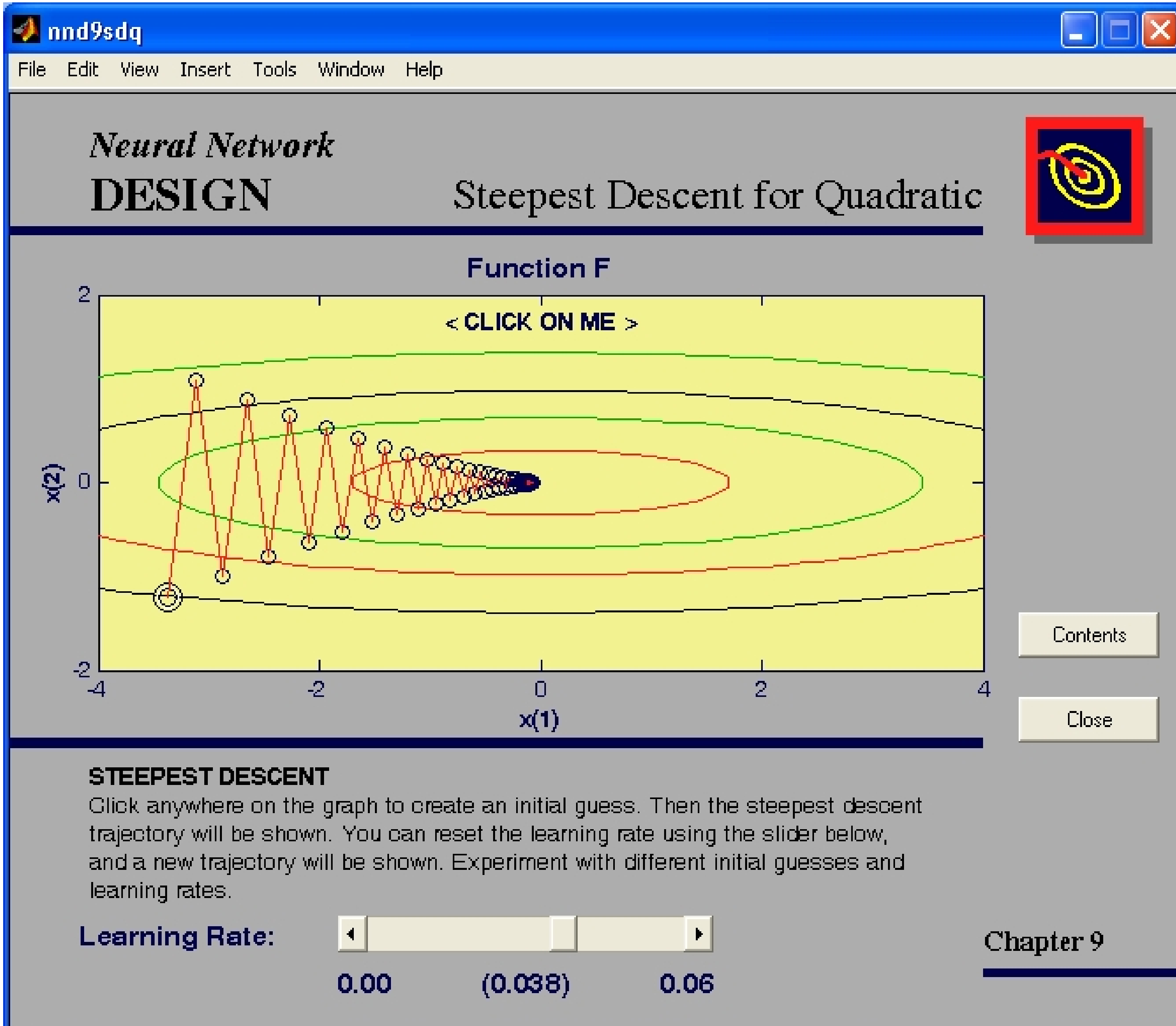


Fixed step size can be too small

Can be slow if t is too small. Same example, gradient descent after 100 steps:



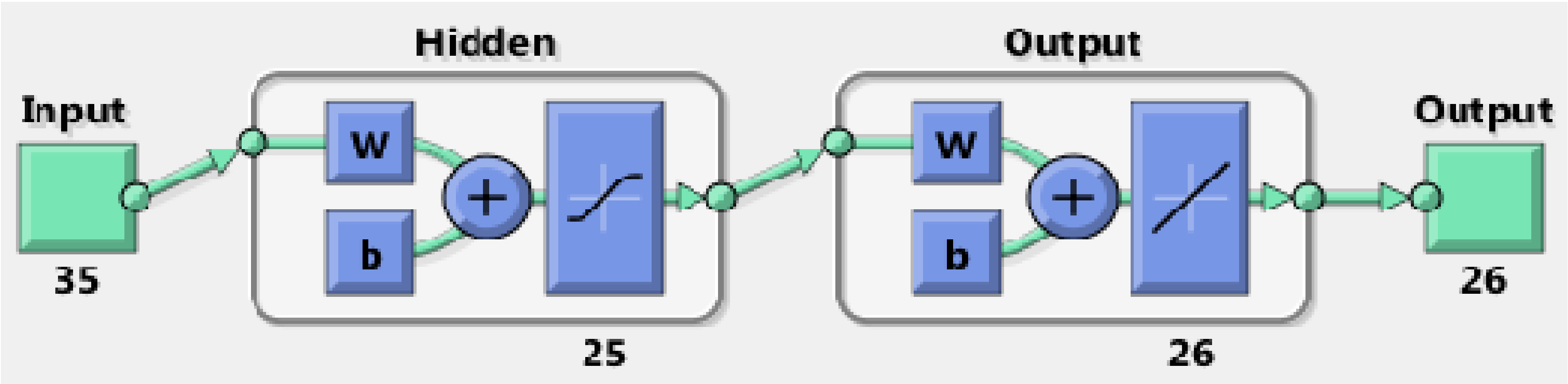

nnd9sd
File Edit View Insert Tools Window Help
Neural Network
DESIGN
Steepest Descent
Function F
< CLICK ON ME >
Approximation Fa
x(1)
x(2)
Contents
Close
STEEPEST DESCENT
Click anywhere on the graph to create an initial guess. Then the steepest descent trajectory will be shown. You can reset the learning rate using the slider below, and a new trajectory will be shown. Experiment with different initial guesses and learning rates.
Learning Rate:
0.00
(0.03)
0.20
Chapter 9


nnd9sdq
File Edit View Insert Tools Window Help
Neural Network
DESIGN
Steepest Descent for Quadratic
Function F
< CLICK ON ME >
2
0
-2
x(2)
-4
-2
0
2
4
x(1)
Contents
Close
STEEPEST DESCENT
Click anywhere on the graph to create an initial guess. Then the steepest descent trajectory will be shown. You can reset the learning rate using the slider below, and a new trajectory will be shown. Experiment with different initial guesses and learning rates.
Learning Rate:
0.00
(0.038)
0.06
Chapter 9

Character Recognition with MLP

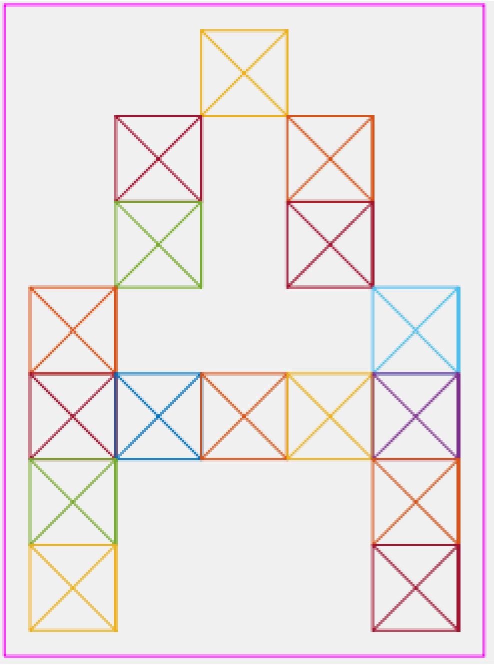
Matlab: appcr1

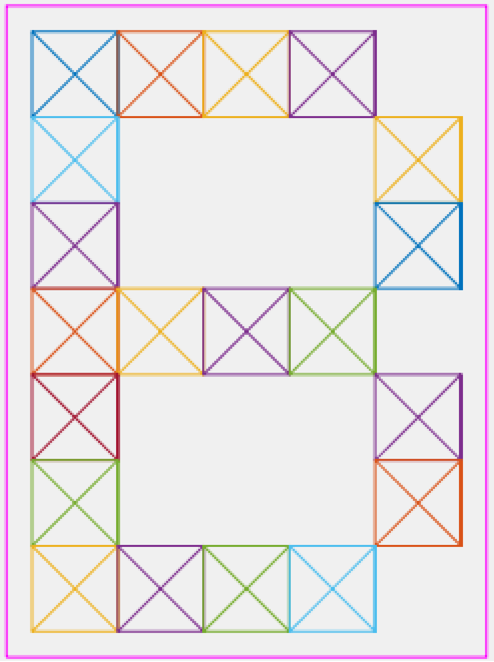
The network



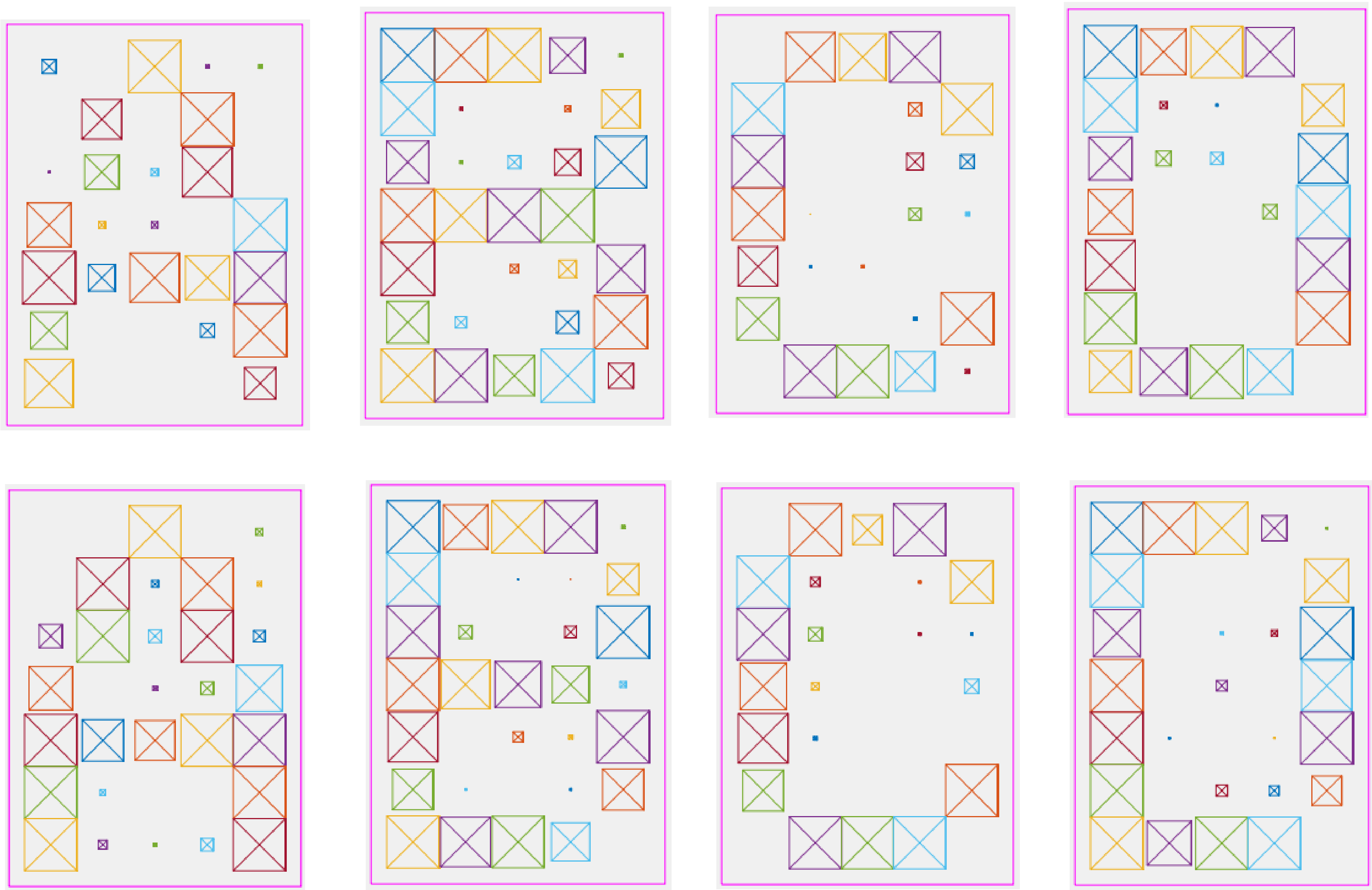
Noise-free input:

26 different letters of size 7x5





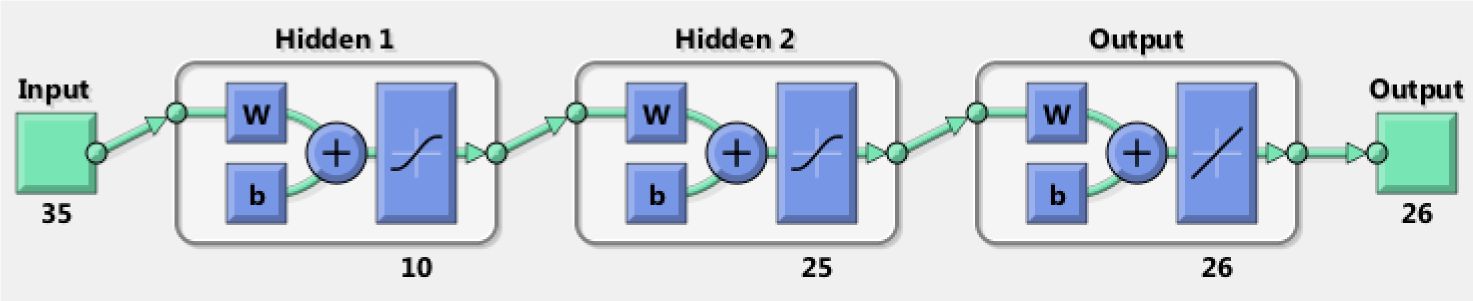
Noisy inputs



Matlab MLP Training

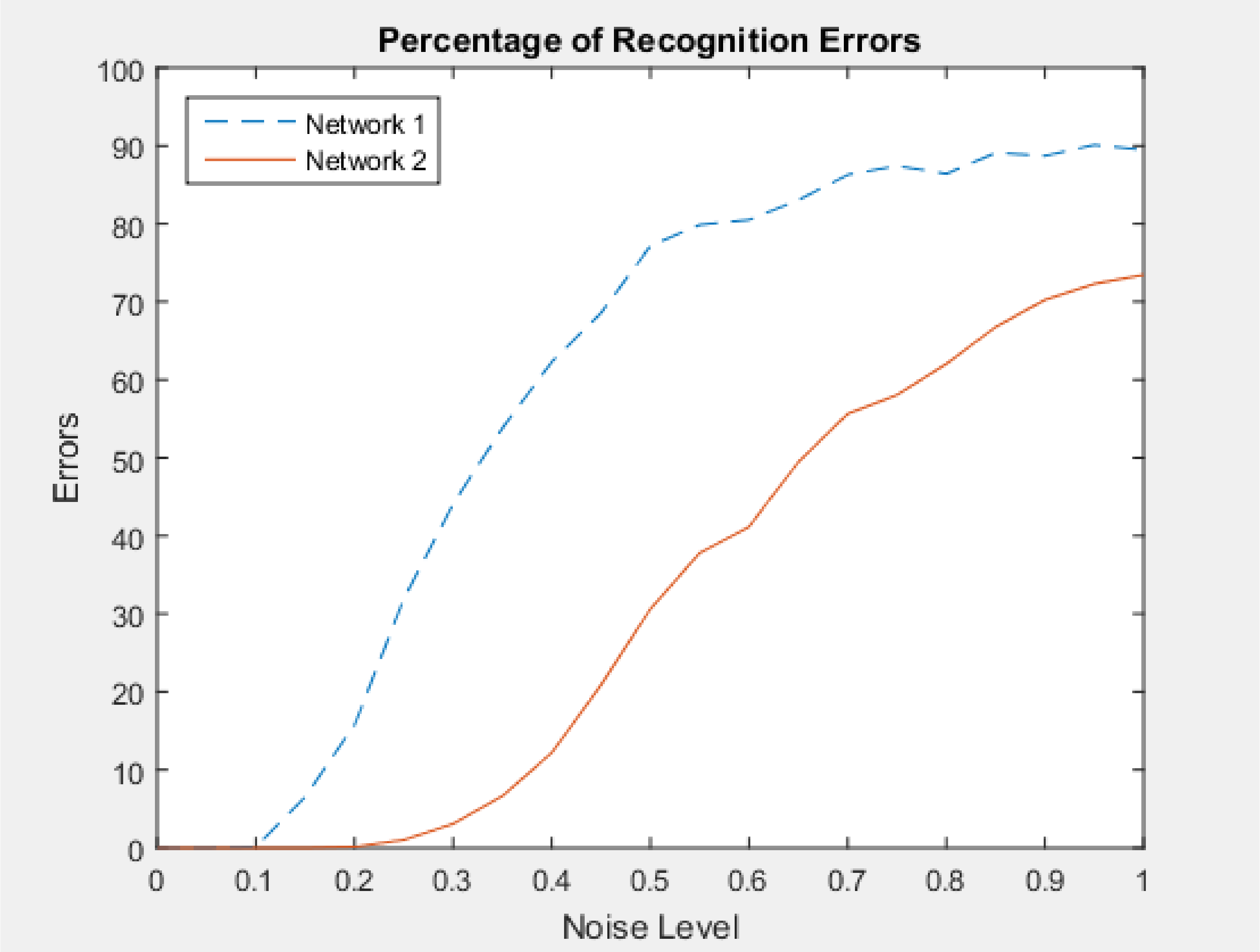
```
% Create MLP

hiddenlayers=[10, 25];

net1 = feedforwardnet(hiddenlayers);

net1 = configure(net1,X,T);
```

```
%View

view(net1);
```




```
%Train

net1 = train(net1,X,T);


%Test

Y1 = net1(Xtest);
```

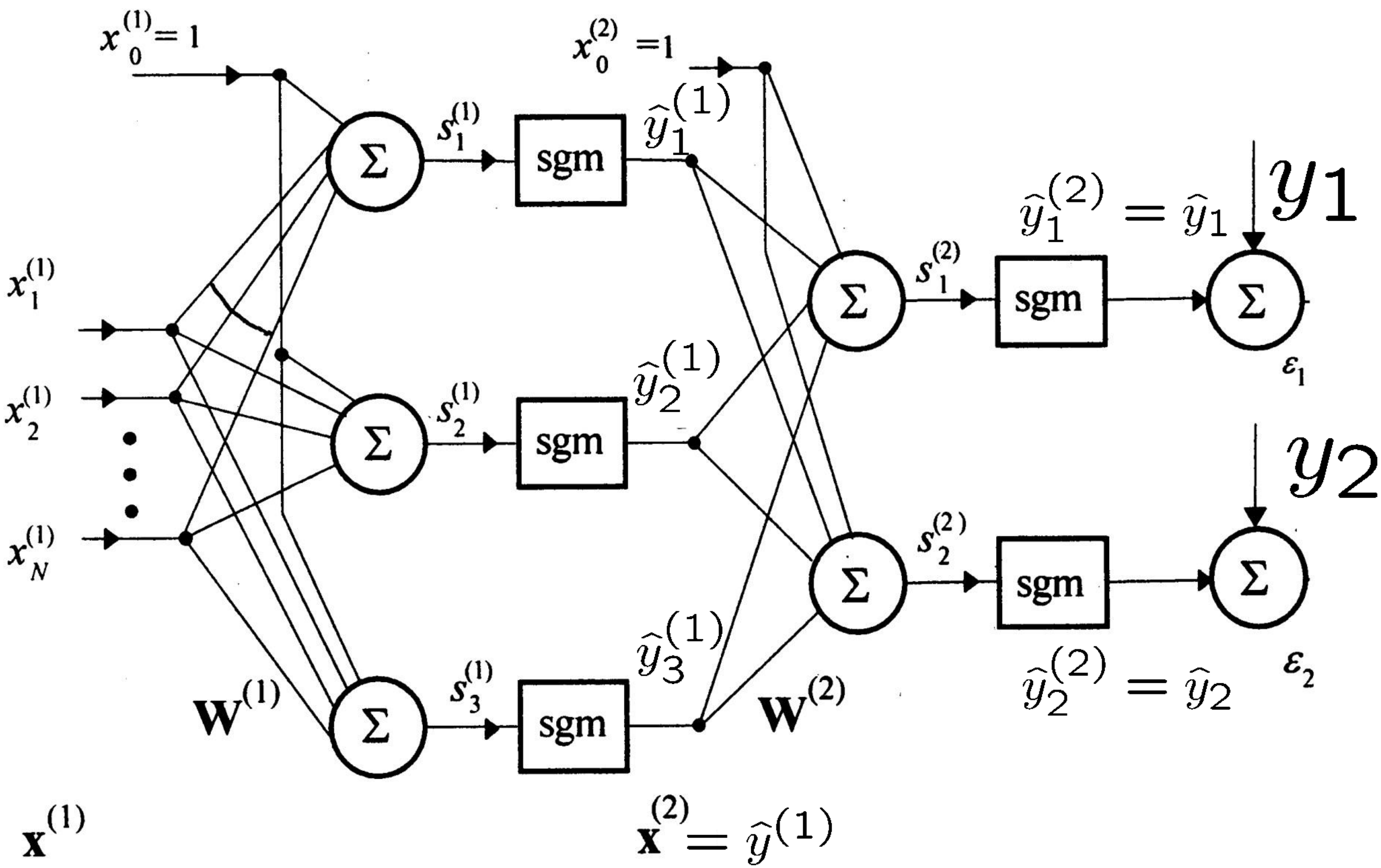
Prediction errors



- Network 1 was trained on clean images
- Network 2 was trained on noisy images. 30 noisy copies of each letter are created

The Backpropagation Algorithm

Multilayer Perceptron



The gradient of the error

The current error:

$$\varepsilon^2 = \varepsilon_1^2 + \varepsilon_2^2 = (\hat{y}_1 - y_1)^2 + (\hat{y}_2 - y_2)^2 \tag{1}$$

More generally:

$$\varepsilon^2 = \sum_{p=1}^{N_L} \varepsilon_p^2 = \sum_{p=1}^{N_L} (\hat{y}_p - y_p)^2 \tag{2}$$

We want to calculate

$$\frac{\partial \varepsilon(k)^2}{\partial W_{ij}^l(k)} = ?$$

Notation

- $W_{ij}^{l}(k)$: At time step k, the strength of connection from neuron j on layer $l-1$ to neuron i on layer l. $(i = 1 \ldots N_l,\ j = 1 \ldots N_{l-1})$

- $s_i^l(k)$: The summed input of neuron i on layer l before function f at time step k $(i = 1 \ldots N_l)$.

- $\mathbf{x}^l(k) \in \mathbb{R}^{N_{l-1}}$: The input of layer l at time step k

- $\widehat{\mathbf{y}}^l(k) \in \mathbb{R}^{N_l}$: The output of layer l at time step k

- $N_1, N_2, \ldots, N_l, \ldots N_L$: Number of neurons in layers $1, 2, \ldots, l, \ldots, L$

Some observations

$$\mathbf{x}^l = \hat{\mathbf{y}}^{l-1} \in \mathbb{R}^{N_{l-1}} \tag{1}$$

$$s_i^l = \mathbf{W}_{i.}^l \hat{\mathbf{y}}^{l-1} = \sum_{j=1}^{N_{l-1}} W_{ij}^l \mathbf{x}_j^l = \sum_{j=1}^{N_{l-1}} W_{ij}^l \underbrace{f(s_j^{l-1})}_{\hat{y}_j^{l-1}} \tag{2}$$

$$s_j^{l+1} = \sum_{i=1}^{N_l} W_{ji}^{l+1} f(s_i^l) \tag{3}$$

The backpropagated error

Introduce the notation

$$\boxed{\delta_i^l(k) = \frac{-\partial \varepsilon^2(k)}{\partial s_i^l(k)} = -\sum_{p=1}^{N_L} \frac{\partial \varepsilon_p^2(k)}{\partial s_i^l(k)}} \qquad (1)$$

where $i = 1, \ldots, N_l$

As a special case, we have that

$$\boxed{\delta_i^L(k) = -\sum_{p=1}^{N_L} \frac{\partial (y_p(k) - f(s_p^L(k)))^2}{\partial s_i^L(k)} = 2\varepsilon_i(k) f'(s_i^L(k))} \qquad (2)$$

The backpropagated error

$$\frac{\partial}{\partial x} f(g(x), h(x)) =? \quad = \frac{\partial}{\partial g} f(g(x), h(x)) \frac{\partial g(x)}{\partial x} + \frac{\partial}{\partial h} f(g(x), h(x)) \frac{\partial h(x)}{\partial x}$$

Lemma

$\delta_i^l(k)$ can be calculated from $\{\delta_1^{l+1}(k), \ldots, \delta_{N_{l+1}}^{l+1}(k)\}$ using Backward recursion.

$$\delta_i^l(k) = -\sum_{p=1}^{N_L} \frac{\partial \varepsilon_p^2}{\partial s_i^l} = \sum_{p=1}^{N_L} \sum_{j=1}^{N_{l+1}} -\frac{\partial \varepsilon_p^2}{\partial s_j^{l+1}} \underbrace{\frac{\partial s_j^{l+1}}{\partial s_i^l}}_{W_{ji}^{l+1} f'(s_i^l)} \tag{1}$$

$$= \sum_{j=1}^{N_{l+1}} \underbrace{\sum_{p=1}^{N_L} -\frac{\partial \varepsilon_p^2}{\partial s_j^{l+1}}}_{\delta_j^{l+1}} W_{ji}^{l+1} f'(s_i^l) \tag{2}$$

The backpropagated error

Therefore,

$$\boxed{\delta_i^l(k) = \left(\sum_{j=1}^{N_{l+1}} \delta_j^{l+1}(k) W_{ji}^{l+1}(k) \right) f'(s_i^l(k))}$$

where $\delta_i^l(k)$ is the backpropagated error.

Now using that

$$s_i^l(k) = \sum_{j=1}^{N_{l-1}} W_{ij}^l(k) x_j^l(k) \qquad (1)$$

The backpropagation algorithm

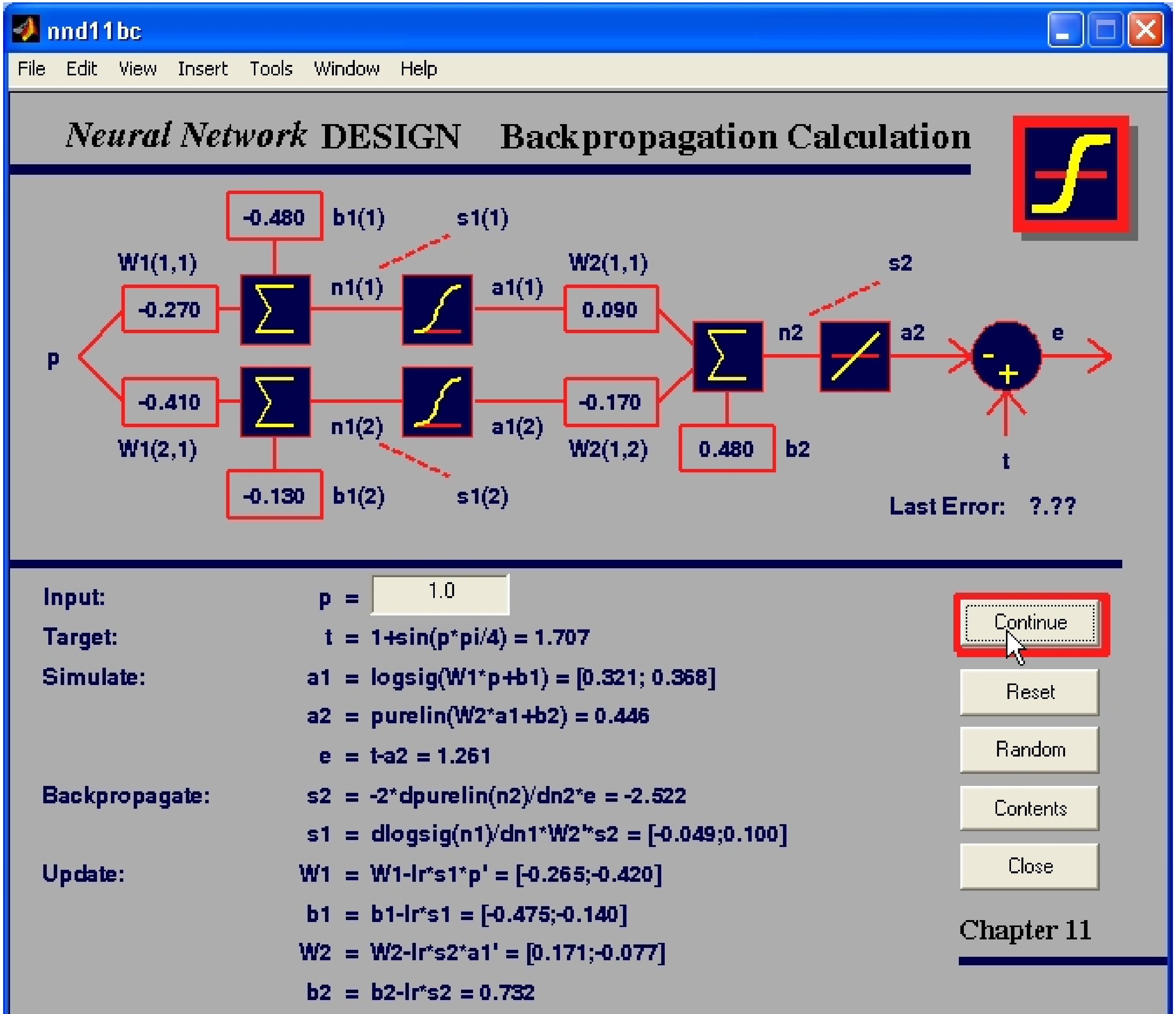
$$\frac{\partial \varepsilon(k)^2}{\partial W_{ij}^l(k)} = \underbrace{\frac{\partial \varepsilon(k)^2}{\partial s_i^l(k)}}_{-\delta_i^l(k)} \underbrace{\frac{\partial s_i^l(k)}{\partial W_{ij}^l(k)}}_{x_j^l(k)} = -\delta_i^l(k) x_j^l(k) \qquad (1)$$

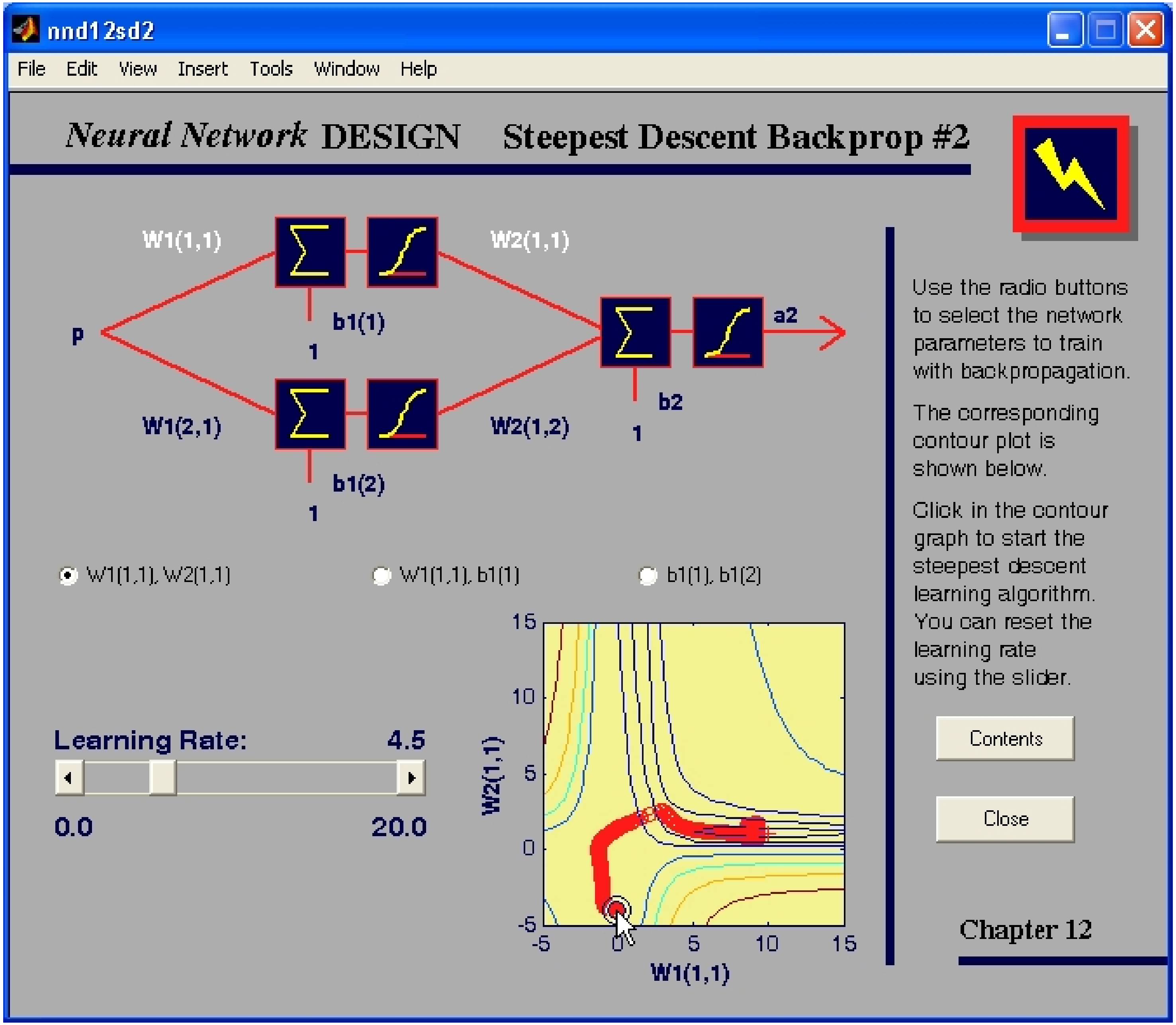
The Backpropagation algorithm:

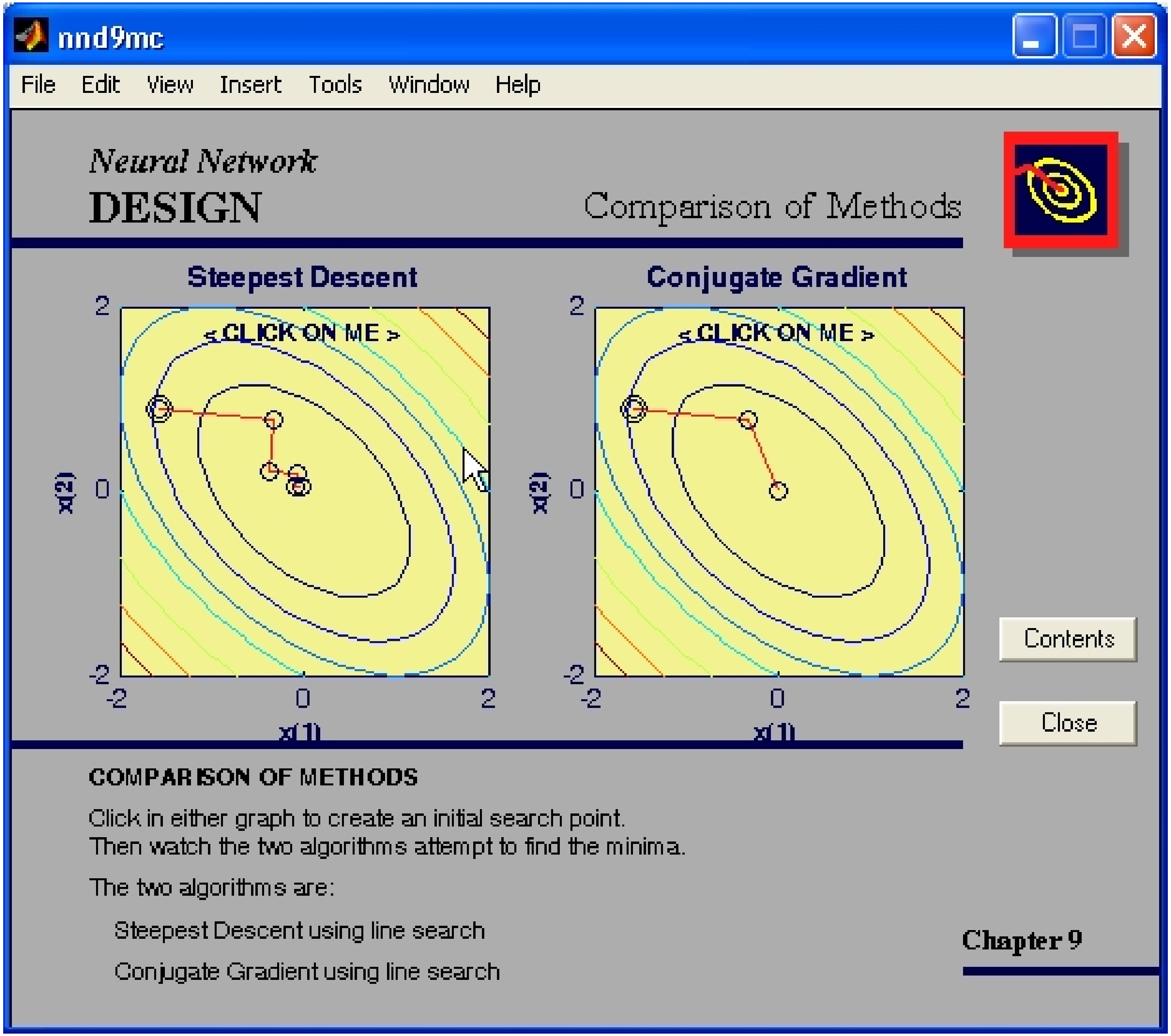
$$\boxed{W_{ij}^l(k+1) = W_{ij}^l(k) + \mu \delta_i^l(k) x_j^l(k)} \qquad (2)$$

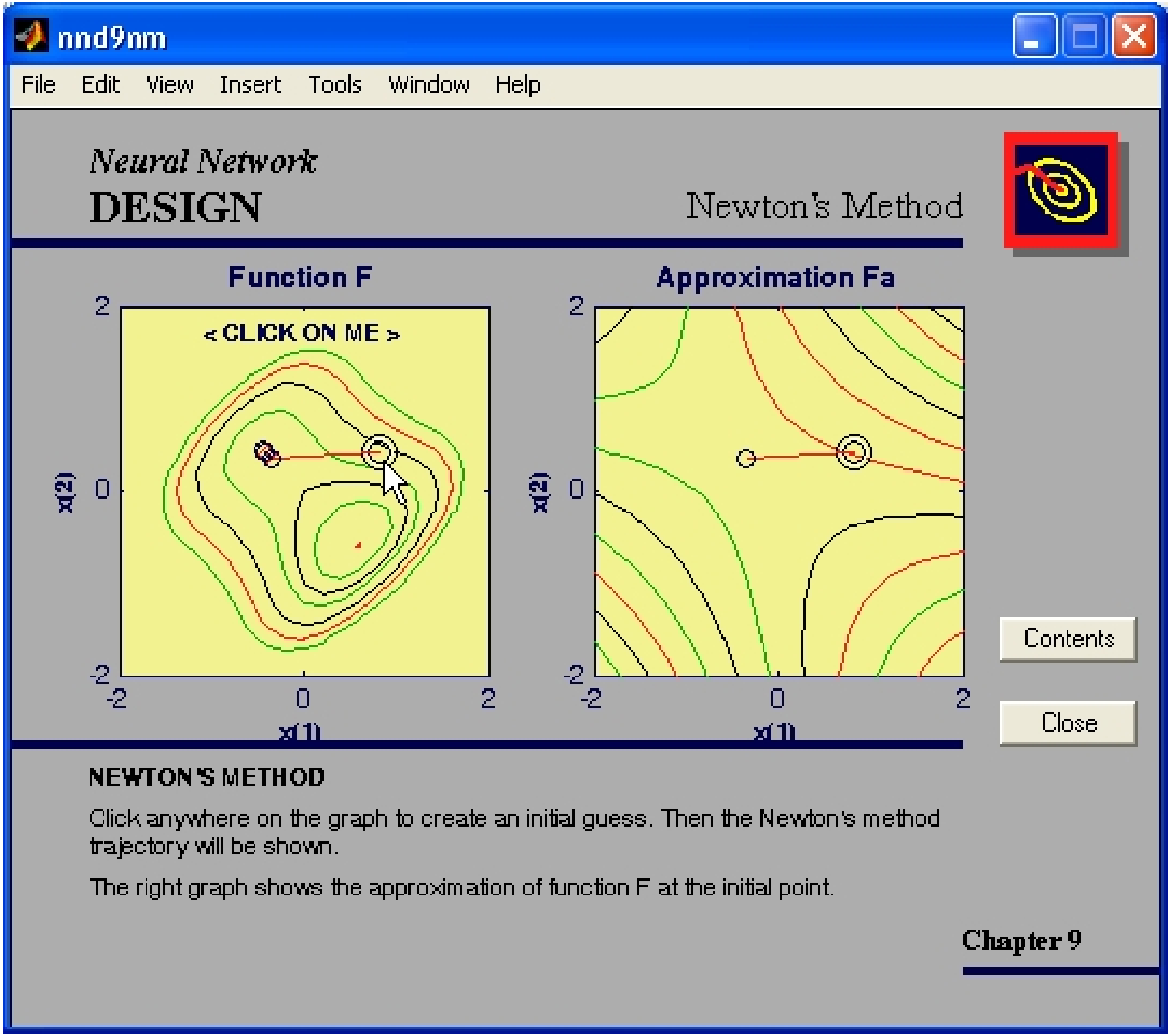
In vector form:

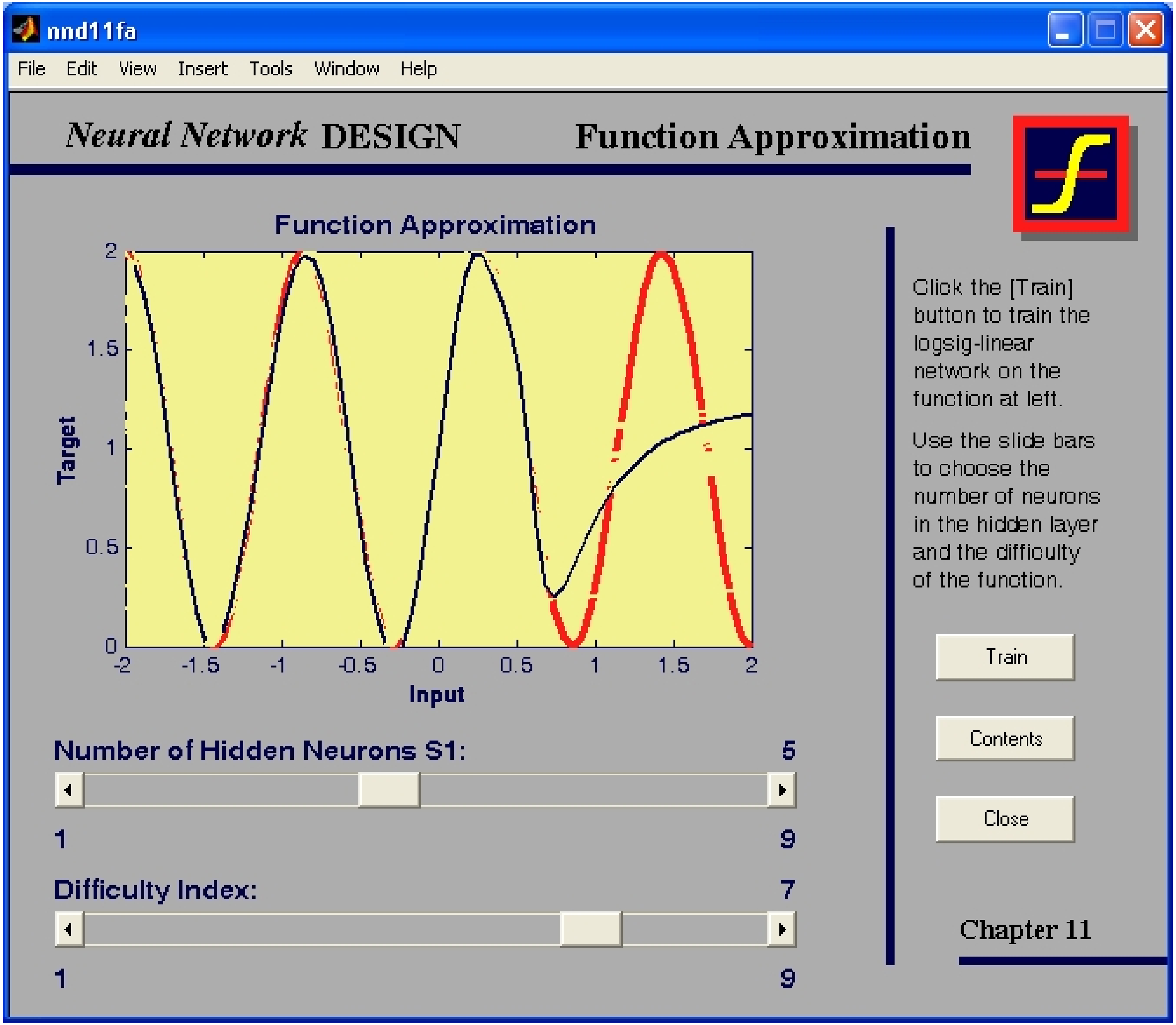
$$\boxed{\mathbf{W}_{i\cdot}^l(k+1) = \mathbf{W}_{i\cdot}^l(k) + \mu \delta_i^l(k) \mathbf{x}_{\cdot}^l(k)} \qquad (3)$$

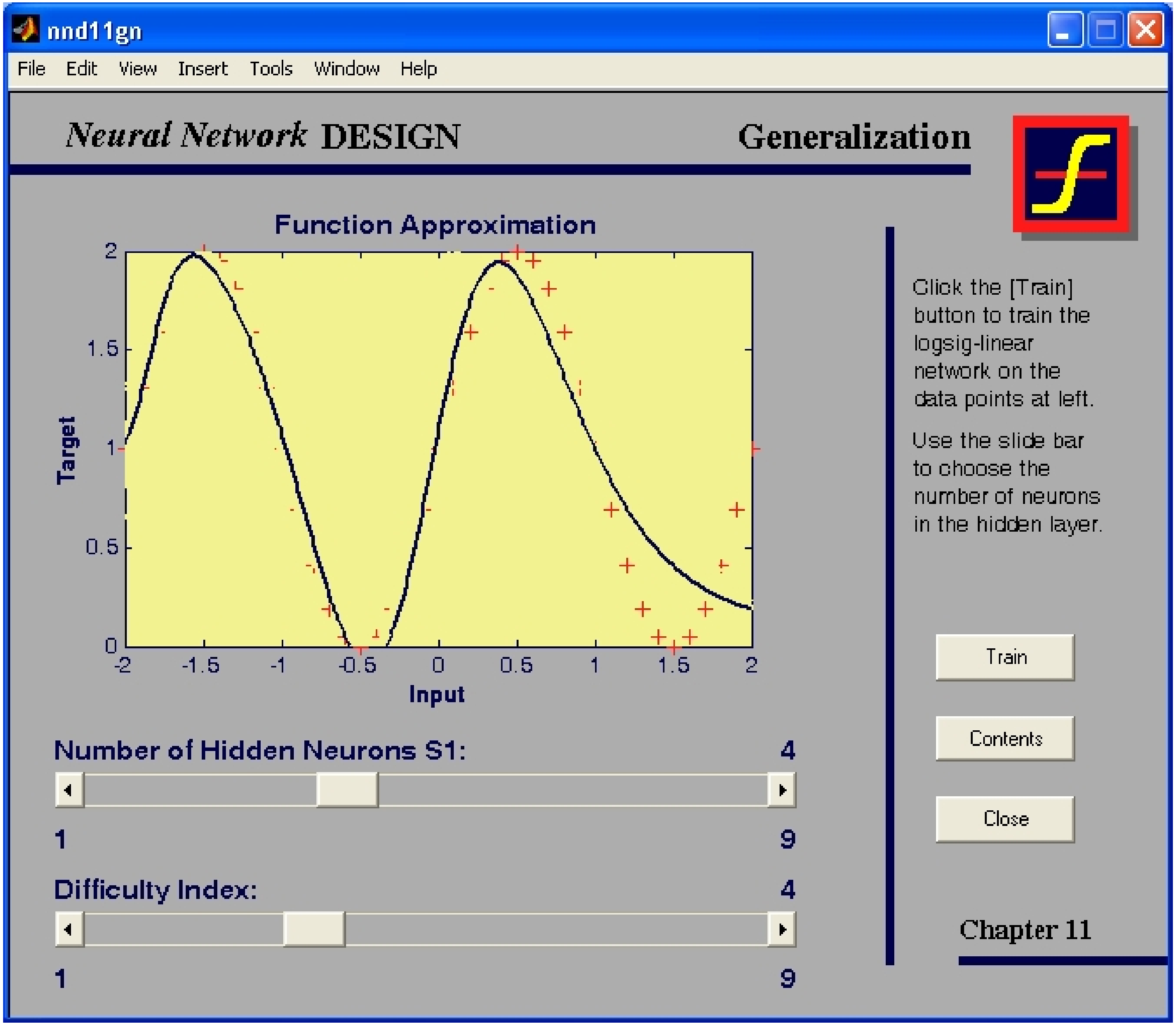

nnd11bc
File Edit View Insert Tools Window Help
Neural Network DESIGN Backpropagation Calculation
-0.480 b1(1) s1(1)
W1(1,1) W2(1,1) s2
-0.270 n1(1) a1(1) 0.090
p n2 a2 e
-0.410 -0.170
n1(2) a1(2)
W1(2,1) W2(1,2) 0.480 b2 t
-0.130 b1(2) s1(2)
Last Error: ?.??
Input: p = 1.0
Target: t = 1+sin(p*pi/4) = 1.707
Simulate: a1 = logsig(W1*p+b1) = [0.321; 0.368]
a2 = purelin(W2*a1+b2) = 0.446
e = t-a2 = 1.261
Backpropagate: s2 = -2*dpurelin(n2)/dn2*e = -2.522
s1 = dlogsig(n1)/dn1*W2'*s2 = [-0.049;0.100]
Update: W1 = W1-lr*s1*p' = [-0.265;-0.420]
b1 = b1-lr*s1 = [-0.475;-0.140]
W2 = W2-lr*s2*a1' = [0.171;-0.077]
b2 = b2-lr*s2 = 0.732
Continue
Reset
Random
Contents
Close
Chapter 11

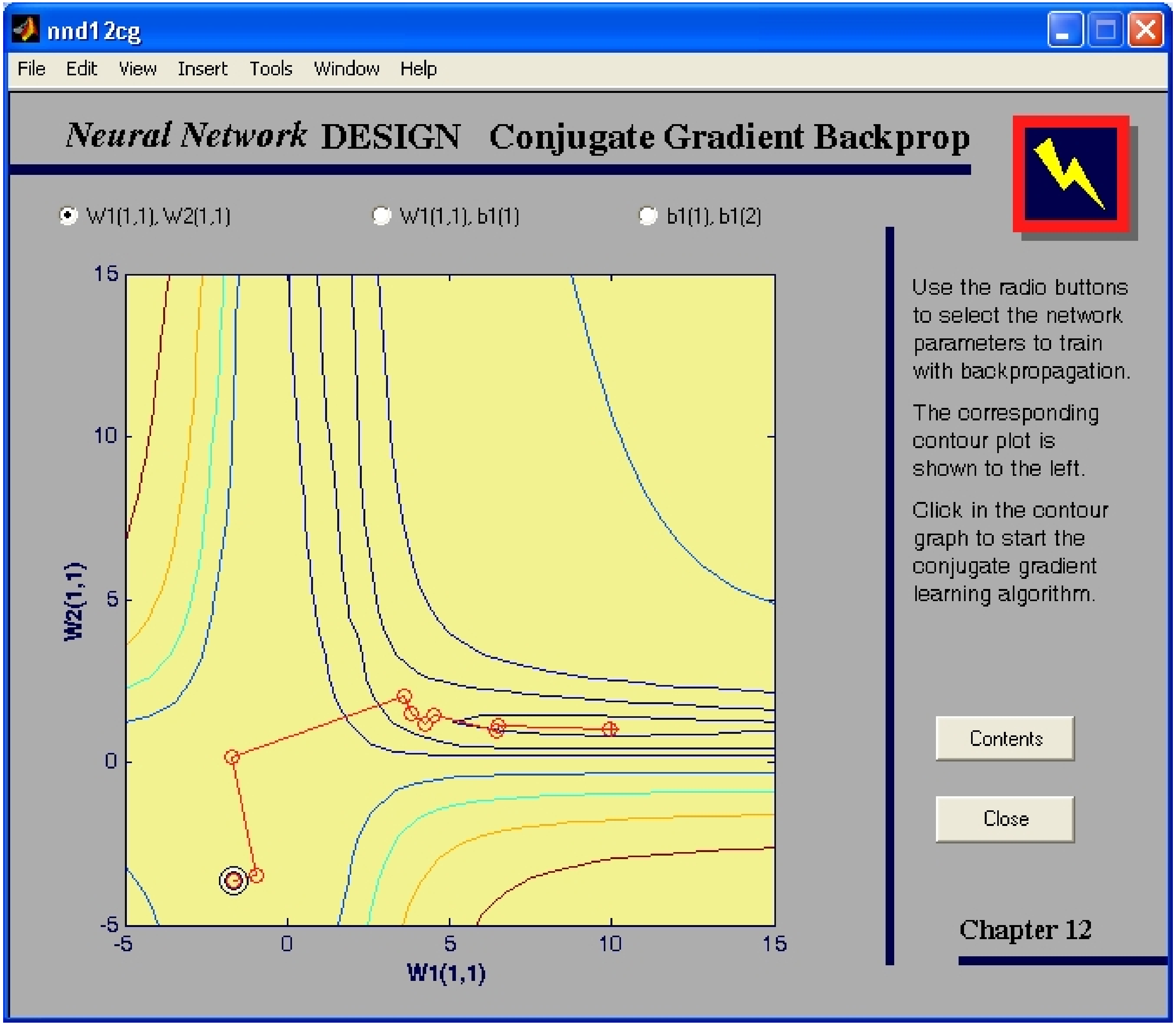

nnd12sd2
File Edit View Insert Tools Window Help
Neural Network DESIGN
Steepest Descent Backprop #2
W1(1,1)
W2(1,1)
p
b1(1)
1
a2
b2
1
W1(2,1)
W2(1,2)
b1(2)
1
W1(1,1), W2(1,1)
W1(1,1), b1(1)
b1(1), b1(2)
Learning Rate: 4.5
0.0
20.0
W2(1,1)
W1(1,1)
Use the radio buttons to select the network parameters to train with backpropagation.
The corresponding contour plot is shown below.
Click in the contour graph to start the steepest descent learning algorithm. You can reset the learning rate using the slider.
Contents
Close
Chapter 12


nnd9mc
File Edit View Insert Tools Window Help
Neural Network
DESIGN
Comparison of Methods
Steepest Descent
Conjugate Gradient
< CLICK ON ME >
< CLICK ON ME >
x(1)
x(2)
Contents
Close
COMPARISON OF METHODS
Click in either graph to create an initial search point.
Then watch the two algorithms attempt to find the minima.
The two algorithms are:
Steepest Descent using line search
Conjugate Gradient using line search
Chapter 9


nnd9nm
File Edit View Insert Tools Window Help
Neural Network
DESIGN
Newton's Method
Function F
< CLICK ON ME >
Approximation Fa
x(1)
x(2)
NEWTON'S METHOD
Click anywhere on the graph to create an initial guess. Then the Newton's method trajectory will be shown.
The right graph shows the approximation of function F at the initial point.
Contents
Close
Chapter 9


nnd11fa
File Edit View Insert Tools Window Help
Neural Network DESIGN
Function Approximation
Function Approximation
Target
Input
Number of Hidden Neurons S1: 5
1
9
Difficulty Index: 7
1
9
Click the [Train] button to train the logsig-linear network on the function at left.
Use the slide bars to choose the number of neurons in the hidden layer and the difficulty of the function.
Train
Contents
Close
Chapter 11

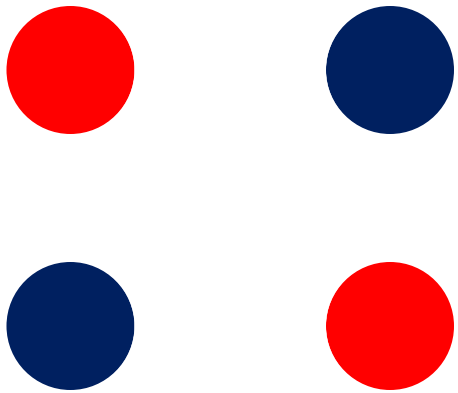

nnd11gn
File Edit View Insert Tools Window Help
Neural Network DESIGN
Generalization
Function Approximation
Target
Input
Number of Hidden Neurons S1: 4
1
9
Difficulty Index: 4
1
9
Click the [Train] button to train the logsig-linear network on the data points at left.
Use the slide bar to choose the number of neurons in the hidden layer.
Train
Contents
Close
Chapter 11


nnd12cg
File Edit View Insert Tools Window Help
Neural Network DESIGN Conjugate Gradient Backprop
W1(1,1), W2(1,1)
W1(1,1), b1(1)
b1(1), b1(2)
15
10
5
0
-5
W2(1,1)
-5
0
5
10
15
W1(1,1)
Use the radio buttons to select the network parameters to train with backpropagation.
The corresponding contour plot is shown to the left.
Click in the contour graph to start the conjugate gradient learning algorithm.
Contents
Close
Chapter 12

What functions can multilayer perceptrons represent?

Perceptrons cannot represent the XOR function

f(0,0)=1, f(1,1)=1, f(0,1)=0, f(1,0)=0



$$f(x_1, x_2) = sgn(w_1 x_1 + w_2 x_2 + w_0). \quad w_0, w_1, w_2 = ?$$

What functions can **multilayer** perceptrons represent?

Hilbert's 13th Problem

1902: 23 "most important" problems in mathematics

The 13th Problem: **"Solve 7-th degree equation using continuous functions of two parameters."**

Conjecture: It can't be solved…

Related conjecture:

Let f be a function of 3 arguments such that

$$f(a,b,c)=x, \text{ where } x^7+ax^3+bx^2+cx+1=0.$$

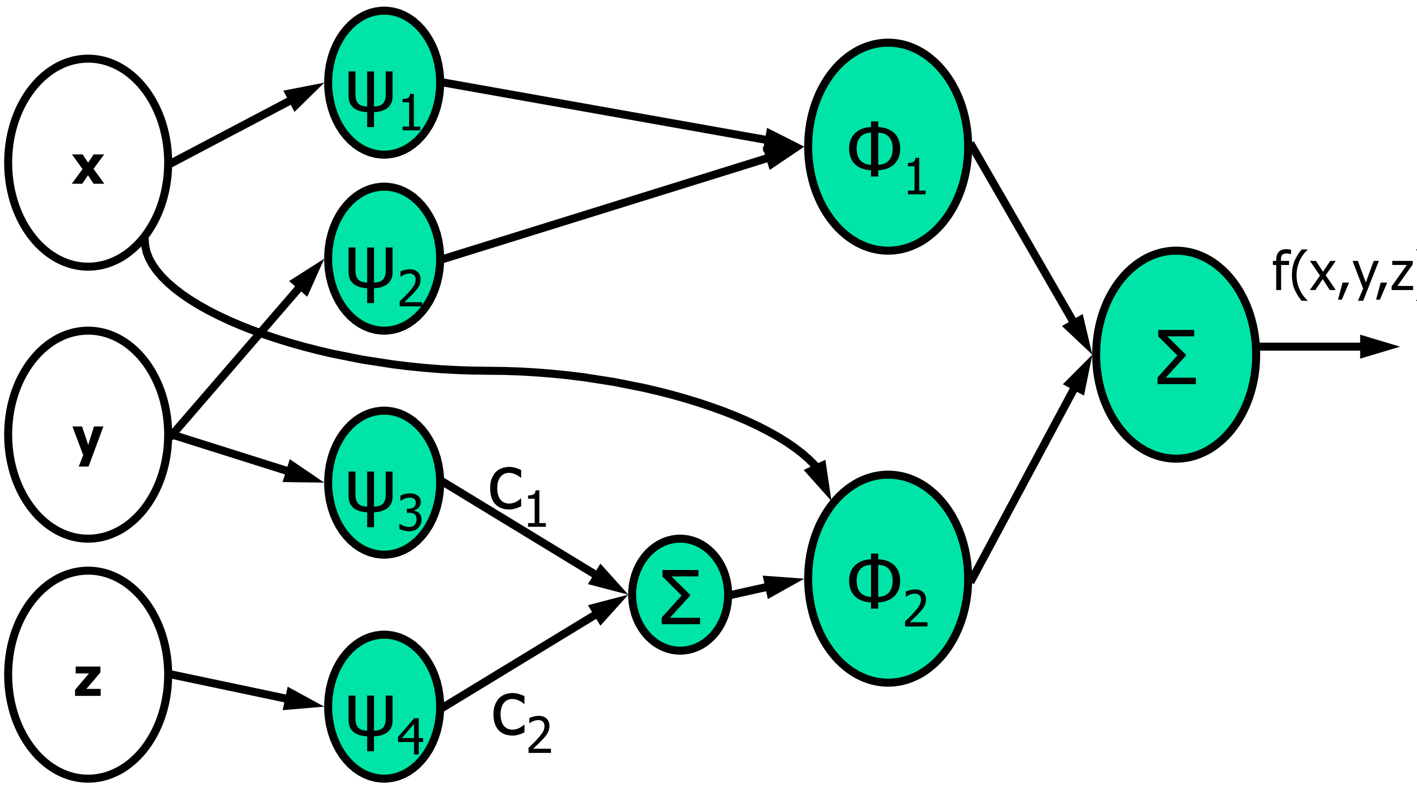
Prove that f cannot be rewritten as a composition of finitely many functions of two arguments.

Another rewritten form:

Prove that there is a nonlinear continuous system of three variables that cannot be decomposed with finitely many functions of two variables.

Function decompositions

$$f(x,y,z)=\Phi_1(\psi_1(x),\ \psi_2(y))+\Phi_2(c_1\psi_3(y)+c_2\psi_4(z),x)$$



Function decompositions

1957, Arnold disproves Hilbert's conjecture.

Let $f : [0,1]^N \to \mathbb{R}$ be an arbitrary continuous function.

Then there exisist $N(2N+1)$ functions ψ_{pq}, s.t.

$\psi_{pq} : [0,1] \to \mathbb{R}$, $p = 1, 2 \ldots N$, $q = 0, 1, \ldots 2N$,

$\star$ they are monotone increasing

$\star$ don't depend on f (only on N)

and there exisist $2N+1$ functions ϕ_q^f:

$\phi_q^f : \mathbb{R} \to \mathbb{R}$, $q = 0, 1, 2 \ldots 2N$, they can depend on f, s.t.

$$f(x_1, \ldots, x_N) = \sum_{q=0}^{2N} \phi_q^f \left(\sum_{p=1}^{N} \psi_{pq}(x_p) \right)$$

Function decompositions

Corollary:

Any $f : [0,1]^N \to \mathbb{R}$ continuous function can be represented exactly with an MLP of two hidden layers.

$$f(x_1, \ldots, x_N) = \sum_{q=0}^{2N} \phi_q^f \left(\sum_{p=1}^{N} \psi_{pq}(x_p) \right)$$

Issues: This statement is not constructive.

For a given N we dont know ψ_{pq},
and for a given N and f, we dont know ϕ_q^f.

Universal Approximators

Kur Hornik, Maxwell Stinchcombe and Halber White: "Multilayer feedforward networks are universal approximators", Neural Networks, Vol:2(3), 359-366, 1989

Definition: $\Sigma^N(g)$ neural network with 1 hidden layer:

$$\Sigma^N(g) = \left\{ f : \mathbb{R}^N \to \mathbb{R} \mid f(x_1, \ldots, x_N) = \sum_{i=1}^{M} c_i g(a_i^T x + b_i) \right\}$$

where $a_i \in \mathbb{R}^n, b_i, c_i \in \mathbb{R}, M < \infty$

Definition: g is sigmoid function if and only if

g is non-decreasing,

$lim_{x \to \infty} g(x) = 1$, $lim_{x \to -\infty} g(x) = 0$

Theorem:

If $\delta > 0$, g arbitrary sigmoid function, f is continuous on a closed and bounded set A, then there exists $\hat{f} \in \Sigma^N(g)$ such that

$$|f(x) - \hat{f}(x)| < \delta$$

for all $x \in A$

Universal Approximators

Definition:

$$\text{sgn}Net^{(2)}(\mathbf{x},\mathbf{w}) = \sum_i w_i^{(3)} \text{sgn}\left(\sum_j w_{ij}^{(2)} \text{sgn}\left(\sum_{l=0}^{d} w_{jl}^{(1)} x_l\right)\right)$$

$$x \in \mathbb{R}^{d+1},\ x_0 = 1,\ \mathbf{w} = \{\mathbf{w}^{(1)}, \mathbf{w}^{(2)}, \mathbf{w}^{(3)}\}$$

Theorem: (Blum & Li, 1991)

$\text{sgn}Net^{(2)}(\mathbf{x},\mathbf{w})$ with two hidden layers and sgn activation function is uniformly dense in L_2.

Formal statement:

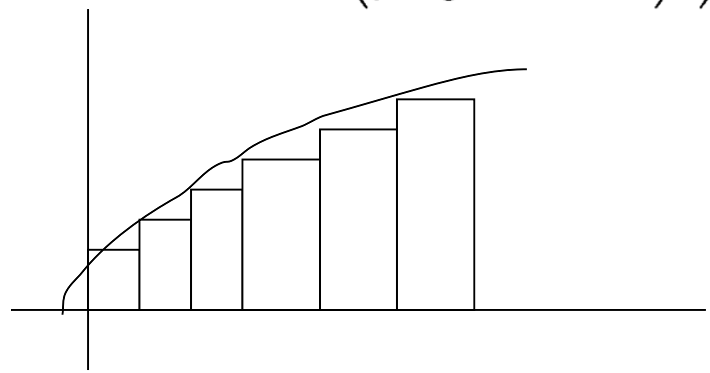
If $f \in L_2$, that is $\int f^2(x)dx < \infty$, and $\epsilon > 0$, then there exists $\mathbf{w}$ such that

$$\int \left| f(x) - \sum_i w_i^{(3)} \text{sgn}\left(\sum_{j=1}^{k_i} w_{ij}^{(2)} \text{sgn}\left(\sum_{l=0}^{d} w_{jl}^{(1)} x_l\right)\right)\right|^2 dx < \epsilon$$

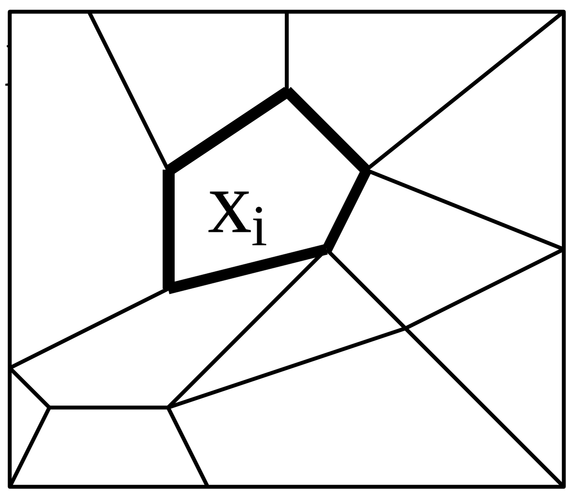
Proof

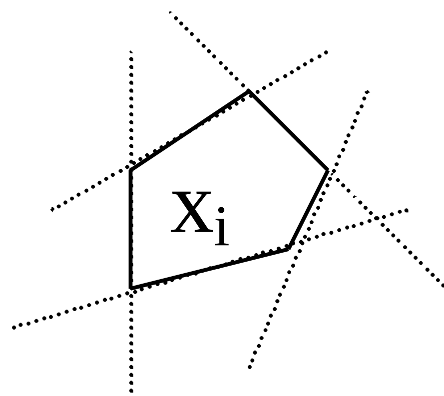
GOAL: $$\int \left| f(x) - \sum_i w_i^{(3)} \mathrm{sgn}\left(\sum_{j=1}^{k_i} w_{ij}^{(2)} \mathrm{sgn}\left(\sum_{l=0}^{d} w_{jl}^{(1)} x_l \right)\right)\right|^2 dx < \epsilon$$

Integral approximation in 1-dim:



Integral approximation in 2-dim:



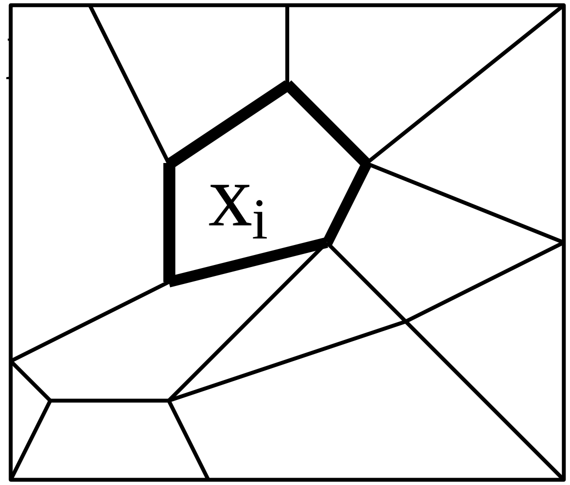


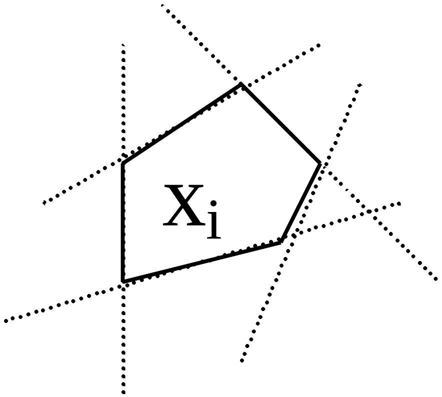
$$\bigcup_i X_i = X \quad X_i \bigcap X_j = \varnothing$$

$$\int \left| f(x) - \sum_i w_i^{(3)} I_{X_i}(x) \right|^2 dx < \epsilon$$

Proof

GOAL: $$\int \left| f(x) - \sum_i w_i^{(3)} \mathrm{sgn}\left(\sum_{j=1}^{k_i} w_{ij}^{(2)} \mathrm{sgn}\left(\sum_{l=0}^{d} w_{jl}^{(1)} x_l \right)\right)\right|^2 dx < \epsilon$$





The indicator function of X_i polygon can be learned by this neural network:

$$\mathrm{sgn}\left(\sum_{j=1}^{k_i} w_{ij}^{(2)} \mathrm{sgn}\left(\sum_{l=1}^{d} w_{jl}^{(1)} x_l \right)\right)$$

1 if x is in X_i

-1 otherwise

The weighted linear combination of these indicator functions will be a good approximation of the original function *f*

Proof

$$\begin{pmatrix} f_1 \\ \vdots \\ f_I \end{pmatrix} = \begin{pmatrix} 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1 \end{pmatrix} \begin{pmatrix} w_1^{(3)} \\ \vdots \\ w_I^{(3)} \end{pmatrix}$$

This linear equation can also be solved.

$$\begin{pmatrix} f_1 \\ \vdots \\ f_I \end{pmatrix} = \begin{pmatrix} 1 & -1 & \dots & -1 \\ -1 & 1 & \dots & -1 \\ \vdots & \vdots & \ddots & \vdots \\ -1 & -1 & \dots & 1 \end{pmatrix} \begin{pmatrix} w_1^{(3)} \\ \vdots \\ w_I^{(3)} \end{pmatrix} \Rightarrow w_i^{(3)}$$

Thanks for your attention!