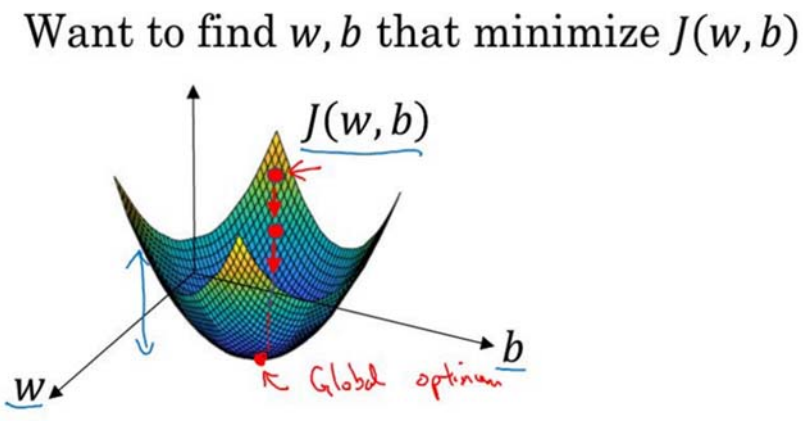
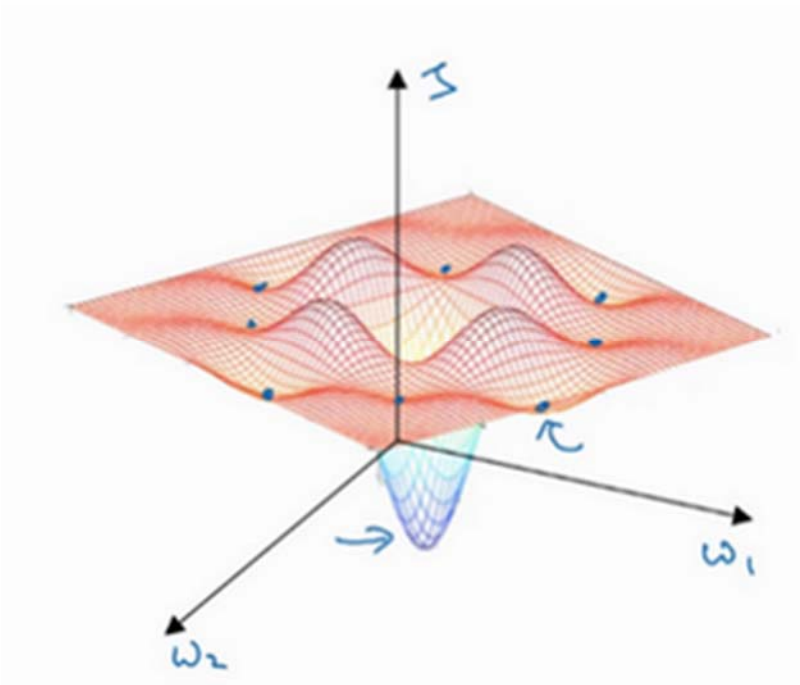
Optimization Methods

Search Methods





Comments on search algorithms:

(1) Local search algorithms: From one point to another point; May be trapped in a local minimizer; May require derivatives.

(2) Global search algorithms: Work with regions; Do not use derivatives; Global in nature.

Neural Networks

1. A neural network consists of inter-connected neurons. The connection between a pair of neurons has a connection weight. Each neuron represents a mapping between multiple inputs and a single output. The output of a neuron depends on the sum of the inputs and an activation function. See Figures 13.1 and 13.2.

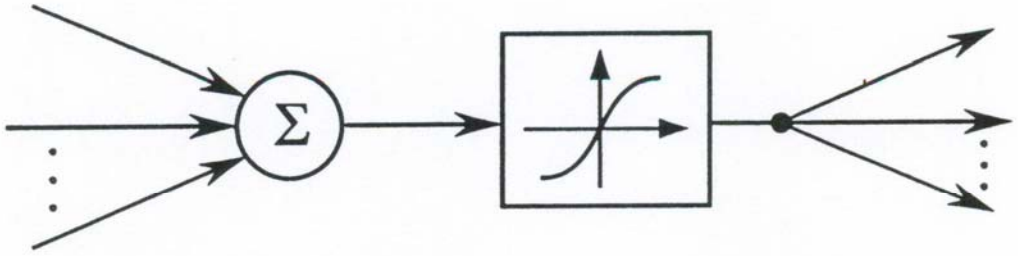


Figure 13.1 A single neuron

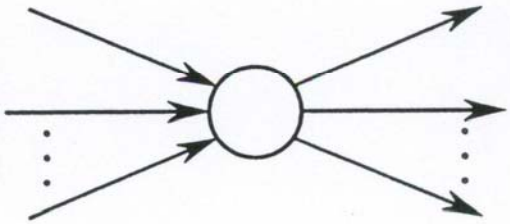


Figure 13.2 Symbol for a single neuron

Example II.21: A single neuron with 3 inputs and the ramp activation function.

2. A feedforward neural network: The neurons are divided into layers. Information flows in one way only. Each neuron receives information only from neurons in the previous layer. The inputs to each neuron are weighted outputs of neurons in the previous layer. The first layer is called the input layer, the last layer is called the output layer, and the other layers are called hidden layers. See Figure 13.3.

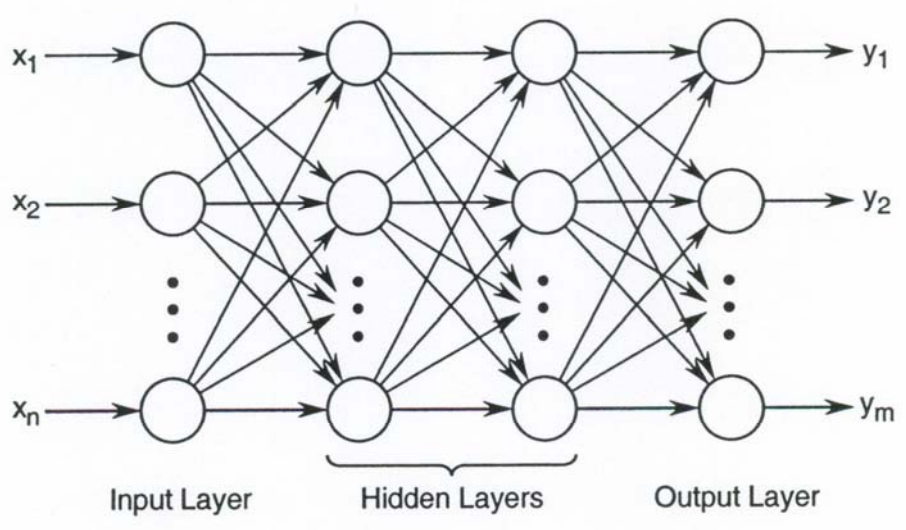


Figure 13.3 Structure of a feedforward neural network

3. A neural network can be considered to be a mapping from $\Re^n$ to $\Re^m$, where n is the number of inputs, $\mathbf{x} = (x_1, x_2, \ldots, x_n)^T$, and m is the number of outputs, $\mathbf{y} = (y_1, y_2, \ldots, y_m)^T$. For a neural network with a fixed structure, the connection weights uniquely define a specific mapping. Thus, a specific neural network can be considered to be a function, though not in a mathematical form. Given the value of $\mathbf{x}$, it gives the value of $\mathbf{y}$, $\mathbf{y} = \mathbf{F}(\mathbf{x})$.

4. We often design and train a neural network to make it behave in a certain way. For given inputs, $\mathbf{x}_{d1}, \mathbf{x}_{d2}, \ldots, \mathbf{x}_{dp}$, it should provide corresponding outputs close to $\mathbf{y}_{d1}, \mathbf{y}_{d2}, \ldots, \mathbf{y}_{dp}$ as much as possible. The data we use, $(\mathbf{x}_{d1}, \mathbf{y}_{d1}), (\mathbf{x}_{d2}, \mathbf{y}_{d2}), \ldots, (\mathbf{x}_{dp}, \mathbf{y}_{dp})$, is called the training set. This process of finding the optimal connection weights with the training set is referred to as supervised learning by the neural network. Conceptually speaking, training a neural network is similar to finding a regression equation. This is an optimization problem.

5. Training a single neuron with the identity activation function (see Figure 13.4): There are n inputs and only one output. This is exactly the same as a multiple linear regression problem. We need to find $\mathbf{w} = (w_1, w_2, \ldots w_n)^T$ such that $\mathbf{y} = w_1 x_1 + w_2 x_2 + \ldots + w_n x_n$ and the sum of squared errors using the training set is minimized. The training set is $(\mathbf{x}_{d1}, y_1)$, $(\mathbf{x}_{d2}, y_2)$, …, $(\mathbf{x}_{dp}, y_p)$. We will consider this problem under three mutually exclusive conditions: (a) p = n, (b) p < n, and (c) p > n. But, first let's look at an example.

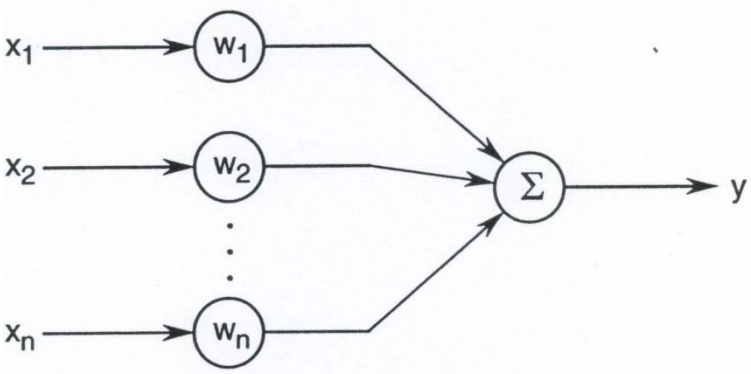


Figure 13.4 A single linear neuron

Example II.22: Consider a single neuron with the identity activation function, 3 inputs, and the following possible training data:

Data #	Input	Output
1	(1, 2, 3)	5
2	(3, 4, 1)	20
3	(3, 5, 2)	15
4	(4, 2, 6)	30

Consider the following two cases: (a) Only the first two data points are used to train the neuron; (b) All 4 data points are used to train the neuron.

Generally, because a single neuron is considered and the activation function is the identity function, the objective function can be written as

Minimize $f(\mathbf{w}) = \frac{1}{2}||X_d^T \mathbf{w} - \mathbf{y}_d||^2$

where $\mathbf{w} = (w_1, w_2, \ldots, w_n)^T$, $\mathbf{y}_d = (y_{d1}, y_{d2}, \ldots, y_{dp})^T$, and $X_d = (\mathbf{x}_{d1}, \mathbf{x}_{d2}, \ldots, \mathbf{x}_{dp})$.

(a) When p = n, the number of data points is exactly equal to the number of decision variables. If the matrix X_d is non-singular, there is an unique optimal solution:

$$X_d^T \mathbf{w} = \mathbf{y}_d, \qquad \mathbf{w}^* = (X_d^T)^{-1} \mathbf{y}_d, \qquad f(\mathbf{w}^*) = 0$$

(b) When p < n, the number of data points is less than the number of decision variables. There are infinite number of optimal solutions. One solution can be obtained by setting $w_j = 0$ for j = p+1, p+2, …, n, letting matrix A consist of the first p rows and the first p columns of the matrix X_d, and using the following equations if A is nonsingular:

$$\mathbf{w}_I^* = A^{-1} \mathbf{y}_d, \qquad \mathbf{w}_{II}^* = \mathbf{0}, \qquad f(\mathbf{w}_I^*, \mathbf{w}_{II}^*) = 0$$

where $\mathbf{w}_I = (w_1, w_2, \ldots, w_p)^T$ and $\mathbf{w}_{II} = (w_{p+1}, w_{p+2}, \ldots, w_n)^T$. Another equation has been developed for finding the **w** vector that has the smallest norm. If the rank of the matrix X_d is p, the unique solution has been found to be (refer to Section 12.3):

$$\mathbf{w}^* = X_d (X_d^T X_d)^{-1} \mathbf{y}_d, \qquad f(\mathbf{w}^*) = 0$$

(c) When p > n, the number of data points is greater than the number of decision variables. The optimal objective function value will be greater than zero. As long as the rank of the matrix X_d is n, the optimal **w** vector can be found to be (refer to Section 12.1):

$$\mathbf{w}^* = (X_d X_d^T)^{-1} X_d \mathbf{y}_d, \qquad f(\mathbf{w}^*) > 0$$

The single neuron together with the above training algorithm is illustrated in Figure 13.5 and is often called Adaline, an acronym for adaptive linear element.

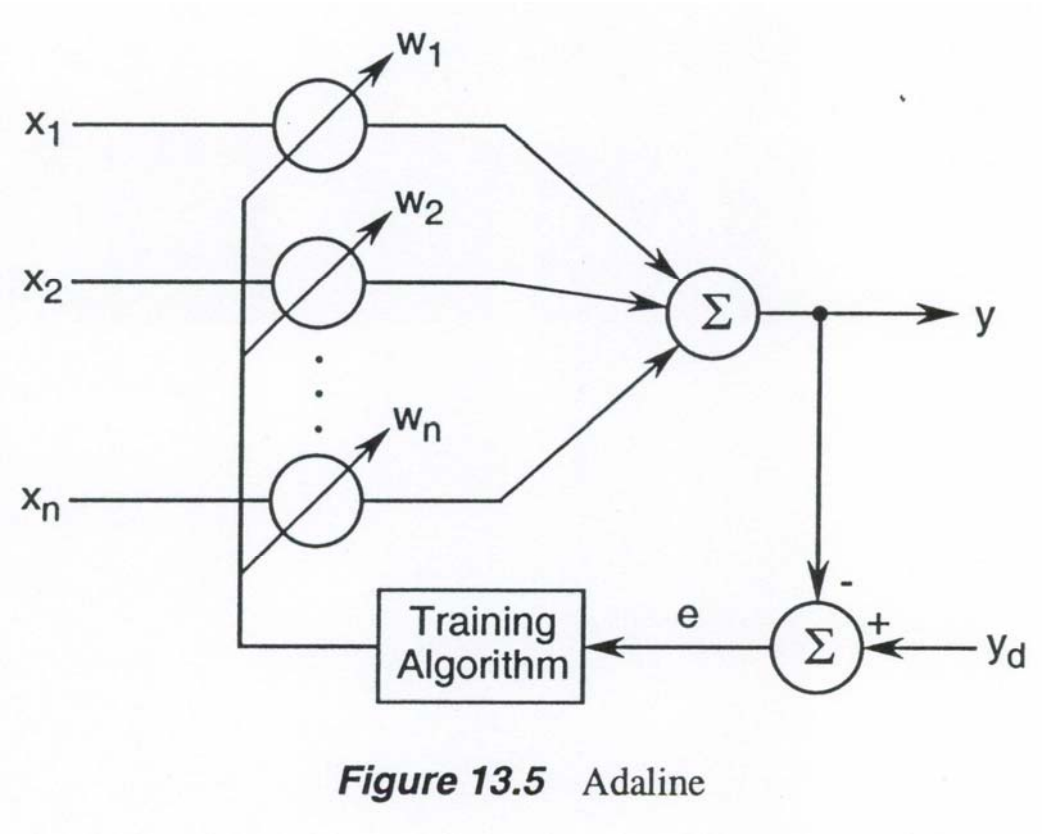


Figure 13.5 Adaline

When the activation function is not the identity function, we do not have a linear regression problem any more. Matrix algebra cannot be used to find a closed form of the optimal solution **w*** any more. Instead, we have to turn to search methods discussed earlier.

2. Backpropagation training algorithm: Consider a three layer neural network with the input layer, the hidden layer, and the output layer shown in Figure 13.6. There are n inputs, m outputs, and l neurons in the hidden layer.

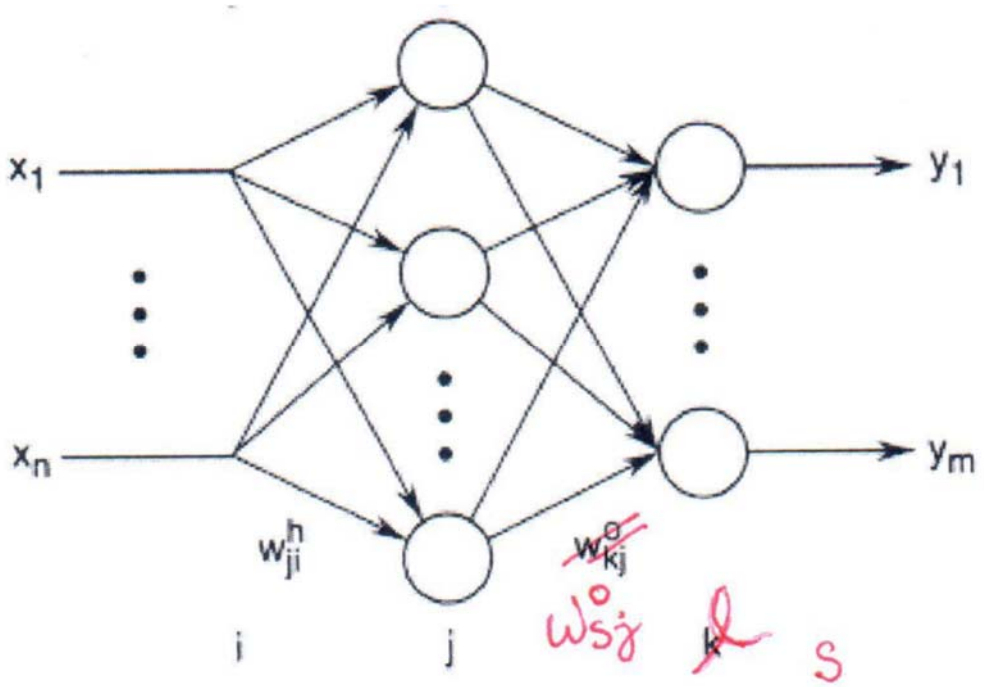


Figure 13.6 Updated: A three-layered neural network

Input: $x_1, x_2, \ldots, x_n$
Input to the hidden layer: v_j for $j = 1, 2, \ldots, l$
Output: $y_1, y_2, \ldots, y_m$
Output from the hidden layer: z_j for $j = 1, 2, \ldots, l$
Connection weights to the hidden layer: w_{ji}^h for $j = 1, 2, \ldots, l$ and $i = 1, 2, \ldots, n$
Connection weights to the output layer: w_{sj}^o for $j = 1, 2, \ldots, l$ and $s = 1, 2, \ldots, m$
Activation functions: f_j^h for $j = 1, 2, \ldots, l$ and f_s^o for $s = 1, 2, \ldots, m$

$$v_j = \sum_{i=1}^{n} w_{ji}^h x_i,$$

$$z_j = f_j^h(v_j),$$

$$y_s = f_s^o\left(\sum_{j=1}^{l} w_{sj}^o z_j\right)$$

$$y_s = f_s^o\left(\sum_{j=1}^{l} w_{sj}^o z_j\right) = f_s^o\left(\sum_{j=1}^{l} w_{sj}^o f_j^h(v_j)\right) = f_s^o\left(\sum_{j=1}^{l} w_{sj}^o f_j^h\left(\sum_{i=1}^{n} w_{ji}^h x_i\right)\right) = F_s(x_1,\ldots,x_n)$$

First consider a single training data point $(\mathbf{x}_d, \mathbf{y}_d)$, where $\mathbf{x}_d \in \Re^n$ and $\mathbf{y}_d \in \Re^m$. We need to find the weights w_{ji}^h for $j = 1, 2, \ldots, l$ and $i = 1, 2, \ldots, n$ and w_{sj}^o for $j = 1, 2, \ldots, l$ and $s = 1, 2, \ldots, m$ such that the following objective function is minimized:

$$\text{Minimize } E(\mathbf{w}) = \frac{1}{2}\sum_{s=1}^{m}(y_{ds} - y_s)^2$$

where y_s, whose equation is given earlier, is a function of input data $\mathbf{x}_d$ and the unknown weights to be optimized. To solve this unconstrained optimization problem, we may use a gradient method with a fixed step size. An iterative procedure is needed with a proper stopping criterion. We need a starting point, that is, initial guesses of the weights of the neural network.

Defining

$$\delta_s = (y_{ds} - y_s) f_s^{o'}\left(\sum_{q=1}^{l} w_{sq}^{o} z_q\right), \quad s = 1, 2, \ldots, m$$

we can express the gradient $\nabla E(w)$ (with respect to w_{ji}^{h} and w_{sj}^{o}) as follows:

$$\frac{\partial E(w)}{\partial w_{ji}^{h}} = -\left(\sum_{p=1}^{m} \delta_p w_{pj}^{o}\right) f_j^{h'}(v_j) x_{di}$$

$$\frac{\partial E(w)}{\partial w_{sj}^{o}} = -\delta_s z_j$$

The fixed step-size gradient method uses the following iterative equation:

$$\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} - \eta \nabla E(\mathbf{w}^{(k)}), \; k = 0, 1, 2, \ldots$$

where η is called the learning rate. Explicitly, we have

$$w_{ji}^{h\,(k+1)} = w_{ji}^{h\,(k)} + \eta\left(\sum_{p=1}^{m} \delta_p^{(k)} w_{pj}^{o\,(k)}\right) f_j^{h'}(v_j^{(k)}) x_{di}$$

$$w_{sj}^{o\,(k+1)} = w_{sj}^{o\,(k)} + \eta \delta_s^{(k)} z_j^{(k)}$$

The update equation for the weights w_{sj}^{o} of the output layer is illustrated in Figure 13.7. The update equation for the weights w_{ji}^{h} of the hidden layer is illustrated in Figure 13.8.

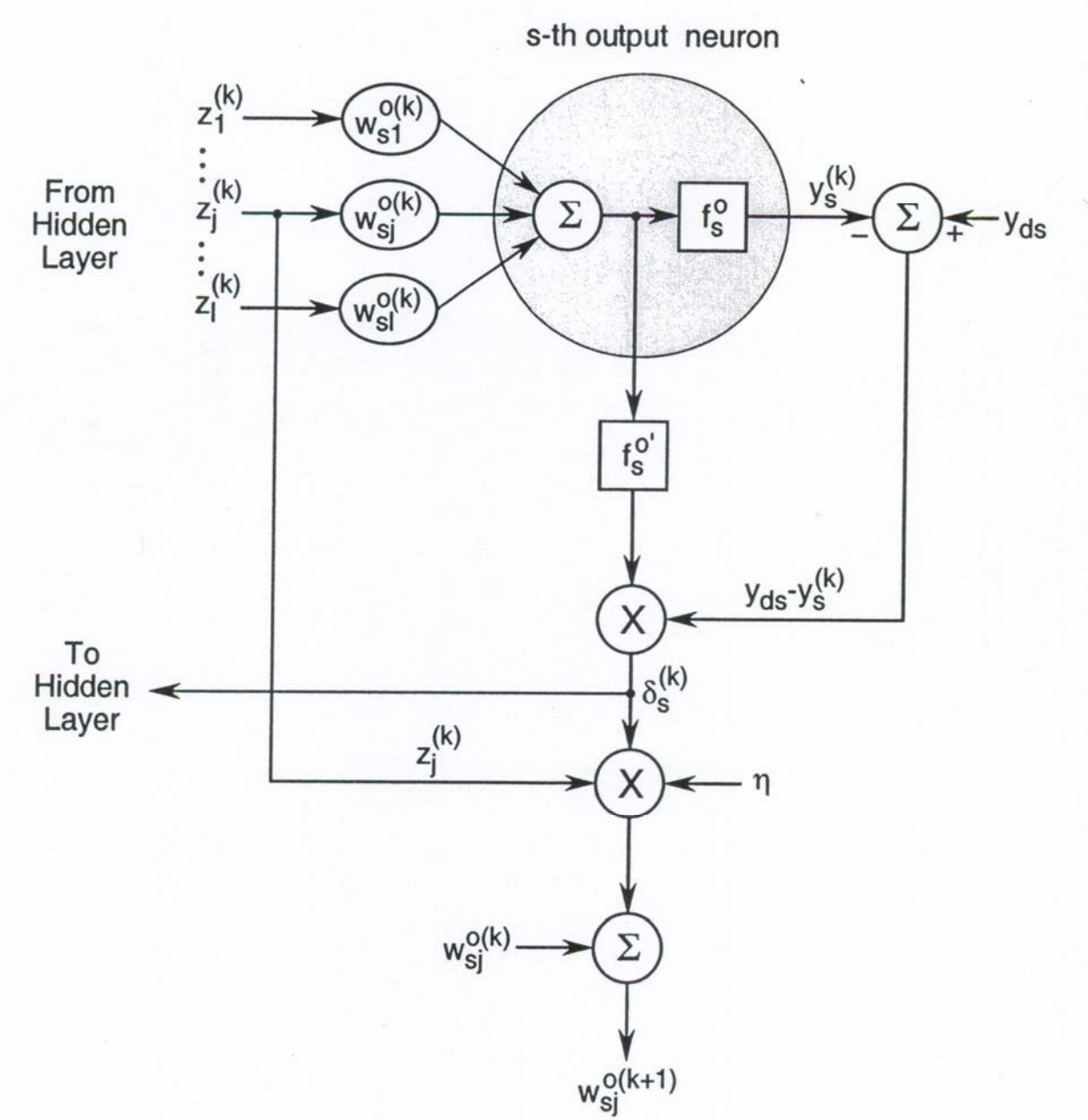


Figure 13.7 Illustration of the update equation for the output layer weights

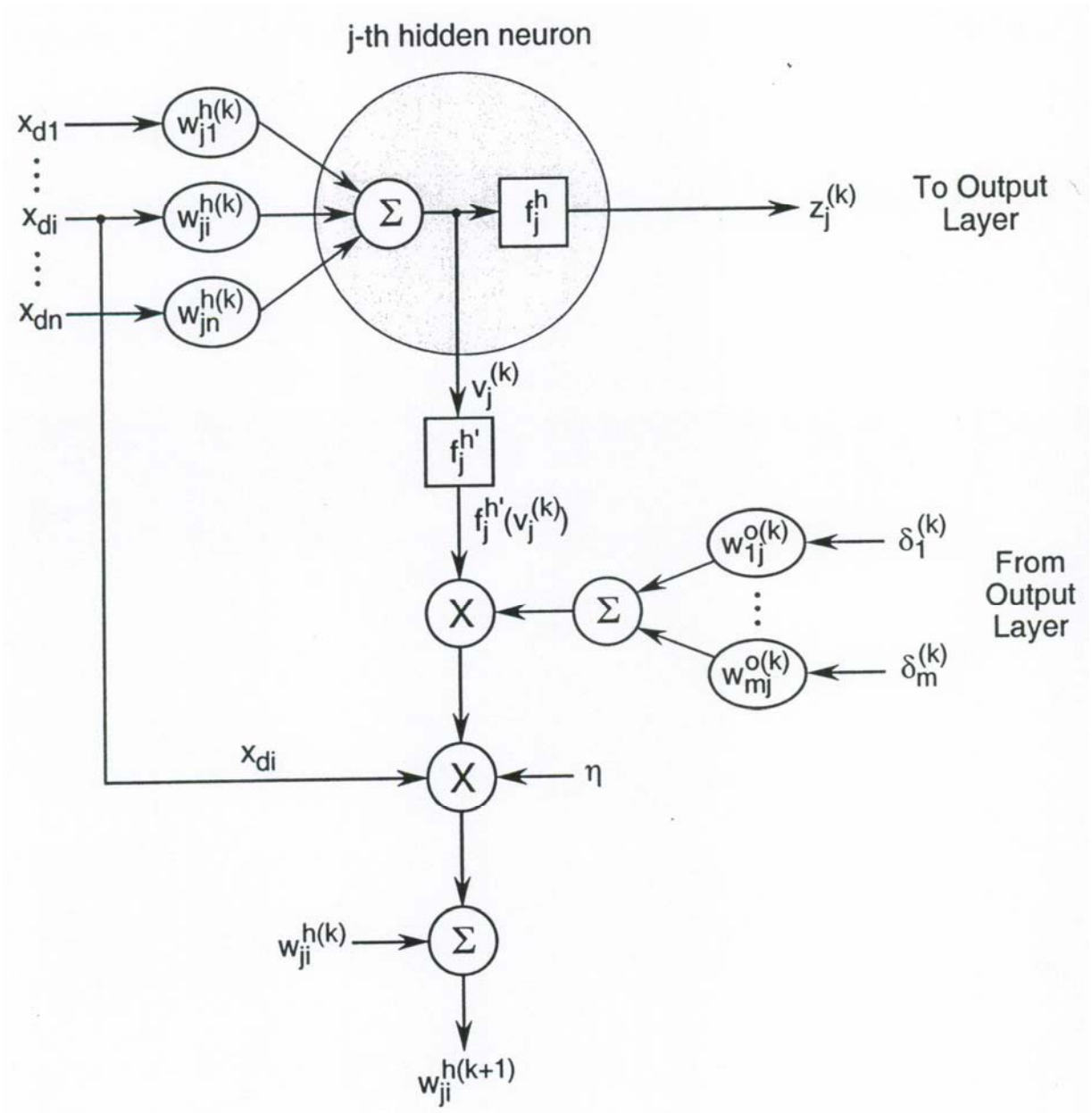


Figure 13.8 Illustration of the update equation for the hidden layer weights

This algorithm is called the backpropagation algorithm because the output errors $\delta_1, \delta_2, \ldots, \delta_m$ are propagated back from the output layer to other layers and are used to update the weights in these layers.

3. The standard sigmoid activation function has the following form:

$$f(v) = \frac{1}{1+e^{-v}}.$$

An extended sigmoid function has the following form:

$$g(v) = \frac{\beta}{1+e^{-(v-\theta)}},$$

where β is called the scale parameter and θ is called the shift parameter. If such an activation function is needed in a neural network, we would also like to be able to adjust the values of these two parameters. However, it turns out that these parameters can be incorporated into the structure of the neural network by treating them as additional weights to be adjusted. The scale parameter β can be incorporated into the weight on the output of the neuron. The shift parameter θ can be treated as the weight to a constant input (+1) to the neuron. This constant input is called the bias term. Refer to Figure 13.11 for graphical explanation. The weights on the constant inputs are often denoted by vector **b**.

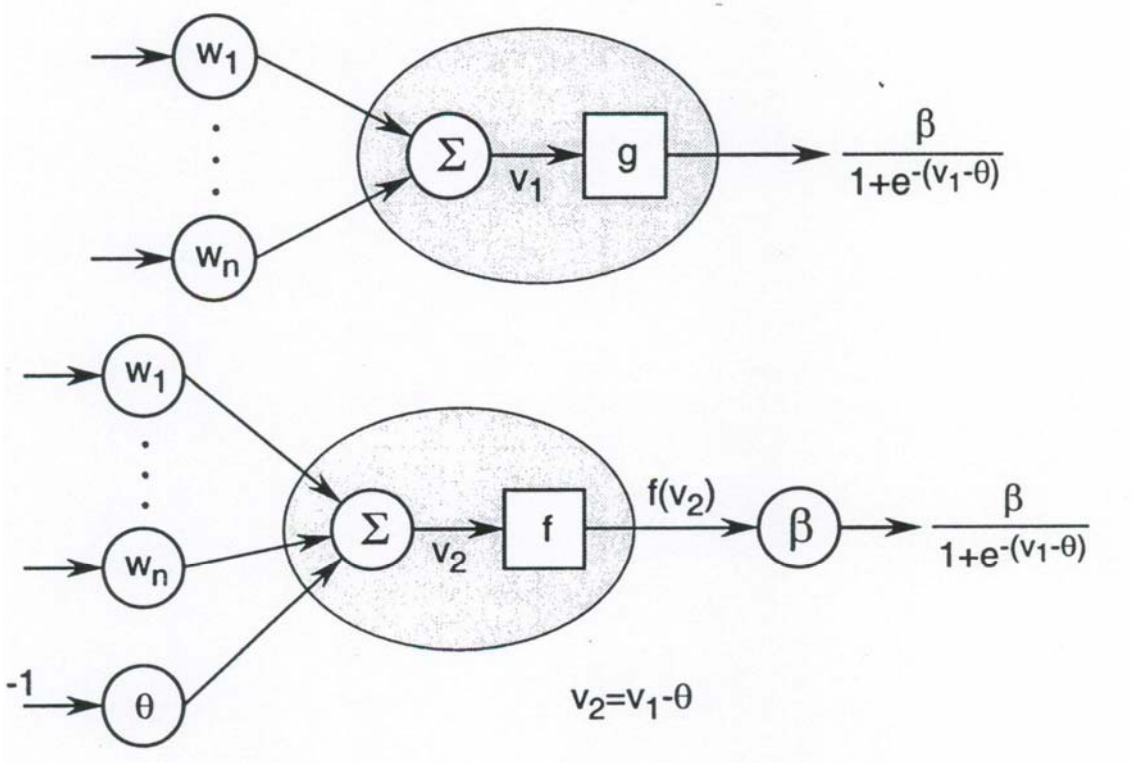


Figure 13.11 The above two configurations are equivalent

Example II.23: Consider a neural network with 2 inputs, 2 hidden neurons, and 1 output neuron. The activation function for all neurons is given by $f(v) = 1/(1+e^{-v})$. The starting points are $(w_{11}^{h(0)}, w_{12}^{h(0)}, w_{21}^{h(0)}, w_{22}^{h(0)}, w_{11}^{o(0)}, w_{12}^{o(0)}) = (0.1, 0.3, 0.3, 0.4, 0.4, 0.6)$. The learning rate $\eta = 10$. Consider a single training input-output pair with $\mathbf{x} = (0.2, 0.6)^T$ and $y = 0.7$. See Figure 13.9. The results of 21 iterations of the backpropagation algorithm are given in the attached table.

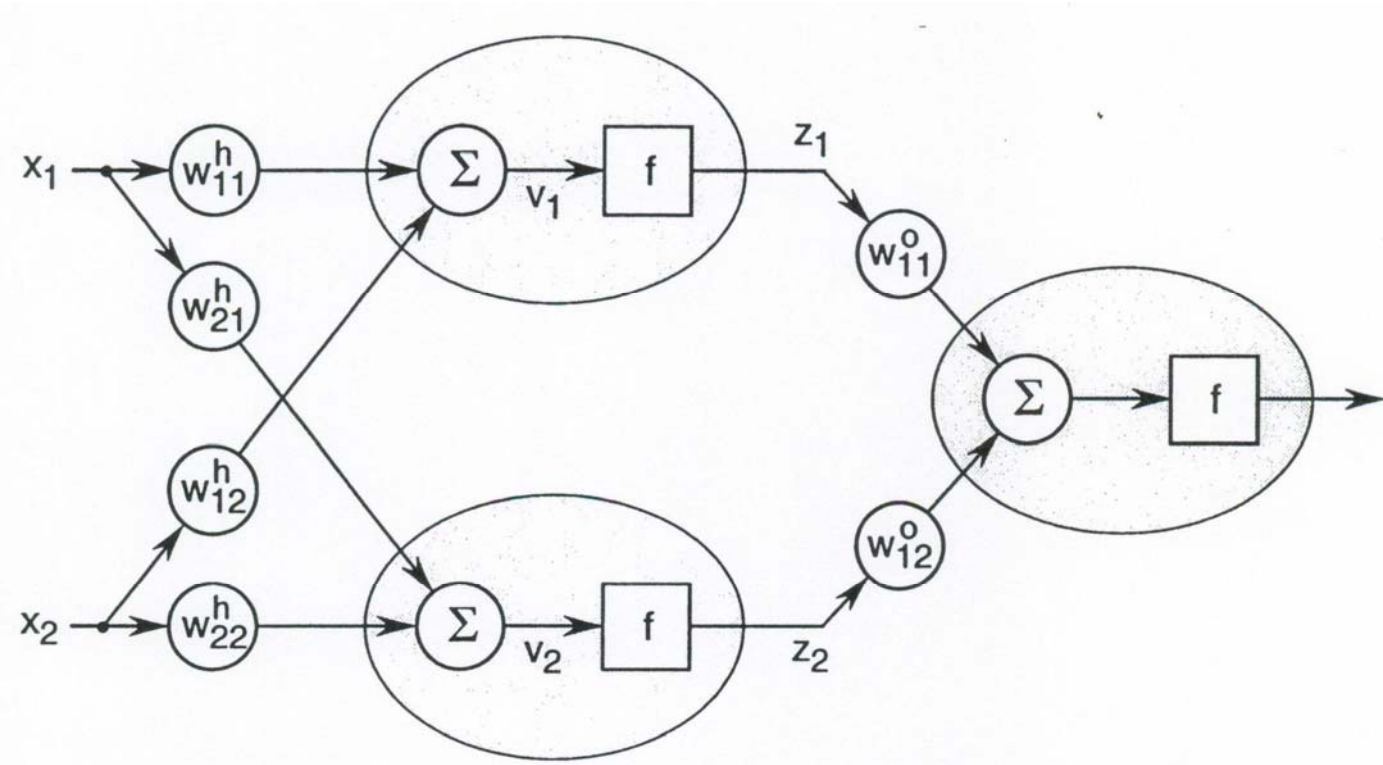


Figure 13.9 Neural network for Example 13.1

Example II.19:

A neural network with 2 inputs, 2 hidden neurons, and 1 output neuron
Notation following Chong and Zak (2001)
Activation function = sigmoid
Learning rate eta = 10
Input 1 = 0.2
Input 2 = 0.6
Output = 0.7
Backpropagation Training with a single input-output pair.

w11(h)	w12(h)	w21(h)	w22(h)	v1	v2	z1	z2	w11(o)	w12(o)	Output	Error	w11(o)-new	w12(o)-new	w11(h)-new	w12(h)-new	w21(h)-new	w22(h)-new	Iteration
0.1000	0.3000	0.3000	0.4000	0.2000	0.3000	0.5498	0.5744	0.4000	0.6000	0.6375	0.0144	0.4794	0.6829	0.1029	0.3086	0.3042	0.4127	1
0.1029	0.3086	0.3042	0.4127	0.2057	0.3085	0.5512	0.5765	0.4794	0.6829	0.6588	0.0093	0.5304	0.7363	0.1051	0.3152	0.3073	0.4220	2
0.1051	0.3152	0.3073	0.4220	0.2101	0.3146	0.5523	0.5780	0.5304	0.7363	0.6723	0.0061	0.5641	0.7716	0.1067	0.3200	0.3095	0.4285	3
0.1067	0.3200	0.3095	0.4285	0.2133	0.3190	0.5531	0.5791	0.5641	0.7716	0.6811	0.0041	0.5868	0.7954	0.1078	0.3234	0.3111	0.4332	4
0.1078	0.3234	0.3111	0.4332	0.2156	0.3221	0.5537	0.5798	0.5868	0.7954	0.6870	0.0028	0.6023	0.8116	0.1086	0.3258	0.3121	0.4364	5
0.1086	0.3258	0.3121	0.4364	0.2172	0.3243	0.5541	0.5804	0.6023	0.8116	0.6910	0.0019	0.6130	0.8227	0.1092	0.3276	0.3129	0.4387	6
0.1092	0.3276	0.3129	0.4387	0.2184	0.3258	0.5544	0.5807	0.6130	0.8227	0.6937	0.0013	0.6204	0.8305	0.1096	0.3288	0.3134	0.4403	7
0.1096	0.3288	0.3134	0.4403	0.2192	0.3269	0.5546	0.5810	0.6204	0.8305	0.6956	0.0009	0.6255	0.8359	0.1099	0.3296	0.3138	0.4414	8
0.1099	0.3296	0.3138	0.4414	0.2197	0.3276	0.5547	0.5812	0.6255	0.8359	0.6969	0.0006	0.6291	0.8396	0.1101	0.3302	0.3141	0.4422	9
0.1101	0.3302	0.3141	0.4422	0.2201	0.3281	0.5548	0.5813	0.6291	0.8396	0.6979	0.0005	0.6316	0.8422	0.1102	0.3306	0.3143	0.4428	10
0.1102	0.3306	0.3143	0.4428	0.2204	0.3285	0.5549	0.5814	0.6316	0.8422	0.6985	0.0003	0.6334	0.8441	0.1103	0.3309	0.3144	0.4432	11
0.1103	0.3309	0.3144	0.4432	0.2206	0.3288	0.5549	0.5815	0.6334	0.8441	0.6989	0.0002	0.6346	0.8454	0.1104	0.3311	0.3145	0.4434	12
0.1104	0.3311	0.3145	0.4434	0.2208	0.3290	0.5550	0.5815	0.6346	0.8454	0.6993	0.0002	0.6355	0.8463	0.1104	0.3313	0.3145	0.4436	13
0.1104	0.3313	0.3145	0.4436	0.2209	0.3291	0.5550	0.5815	0.6355	0.8463	0.6995	0.0001	0.6361	0.8469	0.1105	0.3314	0.3146	0.4438	14
0.1105	0.3314	0.3146	0.4438	0.2209	0.3292	0.5550	0.5816	0.6361	0.8469	0.6996	0.0001	0.6365	0.8474	0.1105	0.3315	0.3146	0.4439	15
0.1105	0.3315	0.3146	0.4439	0.2210	0.3292	0.5550	0.5816	0.6365	0.8474	0.6997	0.0001	0.6368	0.8477	0.1105	0.3315	0.3146	0.4439	16
0.1105	0.3315	0.3146	0.4439	0.2210	0.3293	0.5550	0.5816	0.6368	0.8477	0.6998	0.0000	0.6370	0.8479	0.1105	0.3315	0.3147	0.4440	17
0.1105	0.3315	0.3147	0.4440	0.2210	0.3293	0.5550	0.5816	0.6370	0.8479	0.6999	0.0000	0.6371	0.8481	0.1105	0.3316	0.3147	0.4440	18
0.1105	0.3316	0.3147	0.4440	0.2210	0.3293	0.5550	0.5816	0.6371	0.8481	0.6999	0.0000	0.6373	0.8482	0.1105	0.3316	0.3147	0.4440	19
0.1105	0.3316	0.3147	0.4440	0.2211	0.3294	0.5550	0.5816	0.6373	0.8482	0.6999	0.0000	0.6373	0.8482	0.1105	0.3316	0.3147	0.4441	20
0.1105	0.3316	0.3147	0.4441	0.2211	0.3294	0.5550	0.5816	0.6373	0.8482	0.7000	0.0000	0.6374	0.8483	0.1105	0.3316	0.3147	0.4441	21

The Nelder-Mead Simplex Method

1. A simplex is defined to be a geometric object determined by n+1 points, $\mathbf{p}_0$, $\mathbf{p}_1$, …, $\mathbf{p}_n$, in the n-dimensional space satisfying the following condition :

$$\begin{vmatrix} \mathbf{p}_0 & \mathbf{p}_1 & \cdots & \mathbf{p}_n \\ 1 & 1 & 1 & 1 \end{vmatrix} \neq 0.$$

   In the 2-dimensional space, the 3 points should not be on a straight line, that is, the simplex is a proper triangle. In the 3-dimensional space, the four points should not be on a plane.

2. The algorithm starts with a simplex, finds another point to replace the worst point in the current simplex, allows the simplex to change size, and eventually captures the optimizer with the small simplex meeting a certain accuracy requirement.

3. One way to find the starting simplex is to use a starting point, n step sizes, and the n i-th unit vectors, as follows:

$$\mathbf{p}_0 = \mathbf{x}^{(0)}$$
$$\mathbf{p}_1 = \mathbf{p}_0 + \lambda_1 \mathbf{e}_1$$
$$\ldots.$$
$$\mathbf{p}_n = \mathbf{p}_0 + \lambda_n \mathbf{e}_n$$

4. Illustrate the method for a two variable minimization problem (n = 2):

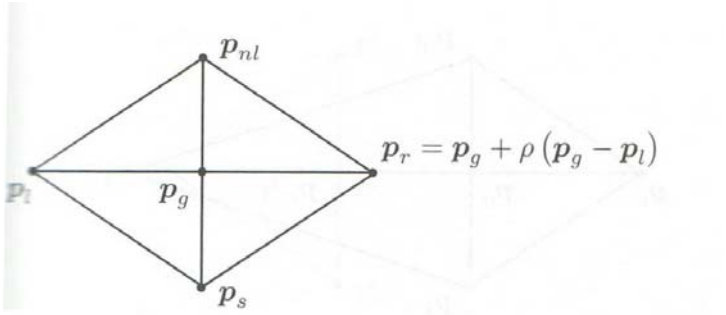
   (1) The simplex has three points. We label them the largest point $\mathbf{p}_l$ (the point with the largest function value. This is the worst point), the next largest point $\mathbf{p}_{nl}$, and the smallest point $\mathbf{p}_s$.

(2) The centroid of the best n points is denoted $\mathbf{p}_g$:

$\mathbf{p}_g = (\mathbf{p}_s + \mathbf{p}_{nl})/2$

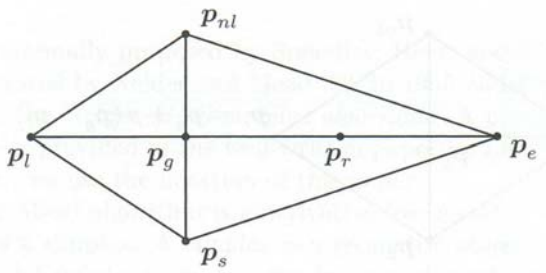
(3) The reflection move yields point $\mathbf{p}_r$. If this point is better than $\mathbf{p}_{nl}$ but worse than $\mathbf{p}_s$, it replaces $\mathbf{p}_l$ and we have a new simplex. The reflection coefficient is denoted by $\rho > 0$. We often choose $\rho = 1$.

$\mathbf{p}_r = \mathbf{p}_g + \rho\,(\mathbf{p}_g - \mathbf{p}_l)$



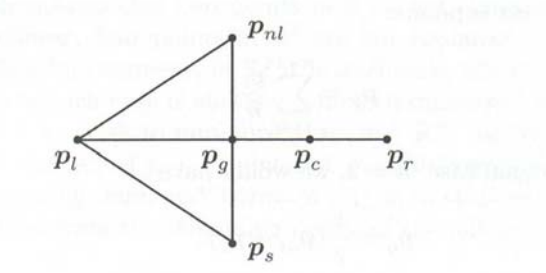
(4) The expansion move: If the point obtained from the previous reflection move, $\mathbf{p}_r$, is even better than $\mathbf{p}_s$, we conduct the so called expansion move to get $\mathbf{p}_e$. If $\mathbf{p}_e$ is better than $\mathbf{p}_r$, we use $\mathbf{p}_e$ to replace $\mathbf{p}_l$ and we have a new simplex; otherwise, we use $\mathbf{p}_r$ to replace $\mathbf{p}_l$ and we have a new simplex. The expansion coefficient $\chi > 1$, and we often choose $\chi = 2$.

$$\mathbf{p}_e = \mathbf{p}_g + \chi\,(\mathbf{p}_r - \mathbf{p}_g)$$



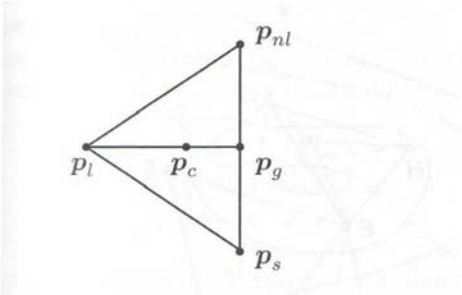
(5) Outside contraction move: If the point obtained from the previous reflection move, $\mathbf{p}_r$, is worse than $\mathbf{p}_{nl}$, but better than $\mathbf{p}_l$, we conduct the so called outside contraction move to get $\mathbf{p}_c$. If $\mathbf{p}_c$ is better than $\mathbf{p}_l$, we use $\mathbf{p}_c$ to replace $\mathbf{p}_l$ and we have a new simplex; otherwise, we go to the next step. The contraction coefficient $0 < \gamma < 1$, and we often choose $\gamma = 0.5$.

$$\mathbf{p}_c = \mathbf{p}_g + \gamma\,(\mathbf{p}_r - \mathbf{p}_g)$$

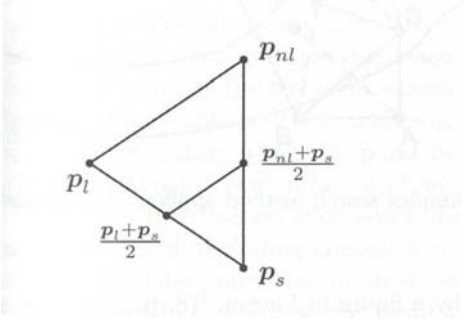


(6) Inside contraction move: If the point obtained from the previous reflection move, $\mathbf{p}_r$, is even worse than $\mathbf{p}_l$, we conduct the so called inside contraction move to get $\mathbf{p}_c$. If $\mathbf{p}_c$ is better than $\mathbf{p}_l$, we use $\mathbf{p}_c$ to replace $\mathbf{p}_l$ and we have a new simplex; otherwise, we go to the next step. The contraction coefficient $0 < \gamma < 1$, and we often choose $\gamma = 0.5$.

$\mathbf{p}_c = \mathbf{p}_g + \gamma\,(\mathbf{p}_l - \mathbf{p}_g)$



(7) Shrinkage move: If neither of the contraction moves is successful, we conduct the so called shrinkage move as illustrated below. The shrinkage coefficient σ is often chosen to be 0.5. The best point does not change.



5. The general algorithm of the simplex method for function minimization:

   (1) Generate n+1 points to form a proper simplex.

   (2) Find the function values of these points and reorder the points such that
   $f(\mathbf{p}_0) \le f(\mathbf{p}_1) \le \ldots \le f(\mathbf{p}_n)$
   Label the worst point as $\mathbf{p}_l$, the next worst point as $\mathbf{p}_{nl}$, and the best point as $\mathbf{p}_s$.

   (3) Find the centroid of the best n points:
   $\mathbf{p}_g = (\mathbf{p}_0 + \mathbf{p}_1 + \ldots + \mathbf{p}_{n-1})/n$,

   (4) Conduct moves with the following equations:
   Reflection move: $\mathbf{p}_r = \mathbf{p}_g + \rho\,(\mathbf{p}_g - \mathbf{p}_l)$
   Expansion move: $\mathbf{p}_e = \mathbf{p}_g + \chi\,(\mathbf{p}_r - \mathbf{p}_g)$
   Outside contraction move: $\mathbf{p}_c = \mathbf{p}_g + \gamma\,(\mathbf{p}_r - \mathbf{p}_g)$
   Inside contraction move: $\mathbf{p}_c = \mathbf{p}_g + \gamma\,(\mathbf{p}_l - \mathbf{p}_g)$
   Shrinkage move: $\mathbf{v}_i = \mathbf{p}_s + \sigma\,(\mathbf{p}_i - \mathbf{p}_s)$, for i = 1, 2, …, n

   (5) Stopping criterion: When the simplex is small enough?

6. An example trace of this algorithm is given below.

Example III.1: Consider the following production scheduling problem:
$f(\theta_1, \theta_2) = 100\,(\theta_1 - 15)^2 + 20\,(28 - \theta_1)^2 + 100\,(\theta_2 - \theta_1)^2 + 20\,(38 - \theta_1 - \theta_2)^2$.

The first simplex:

$$\begin{matrix} & & & \theta_1 & \theta_2 \\ \begin{pmatrix} P_1 \\ P_2 \\ P_3 \end{pmatrix} & = & \begin{pmatrix} T_2 \\ T_3 \\ T_1 \end{pmatrix} & = & \begin{pmatrix} 10 & 14 \\ 10 & 8 \\ 7 & 10 \end{pmatrix} \end{matrix}$$

and

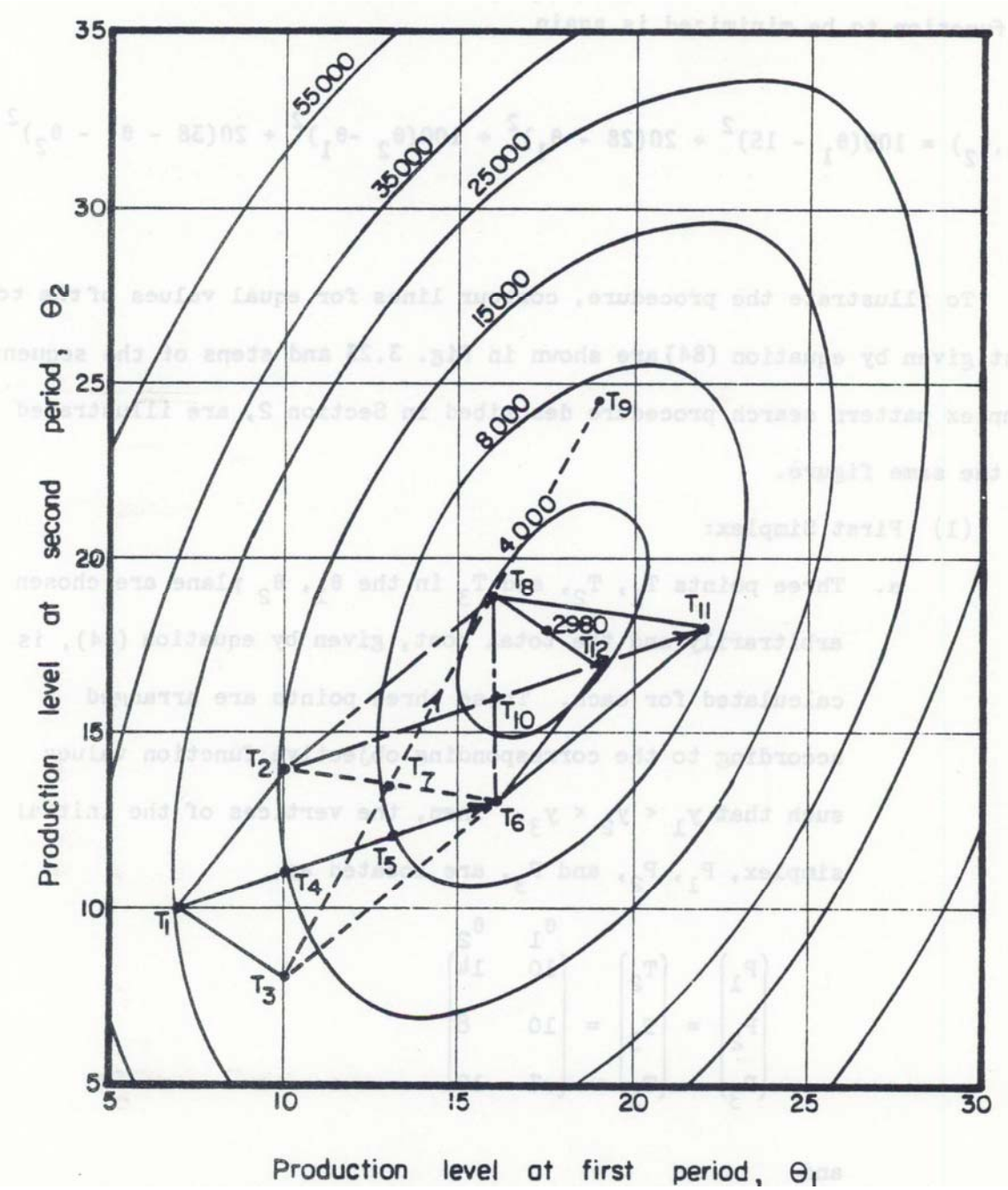
$$\begin{pmatrix} y_1(P_1) \\ y_2(P_2) \\ y_3(P_3) \end{pmatrix} = \begin{pmatrix} 14,500 \\ 17,380 \\ 24,940 \end{pmatrix}$$

The second simplex:

$$\begin{matrix} & & & \theta_1 & \theta_2 \\ \begin{pmatrix} P_1 \\ P_2 \\ P_3 \end{pmatrix} & = & \begin{pmatrix} T_6 \\ T_2 \\ T_3 \end{pmatrix} & \begin{pmatrix} 16 & 13 \\ 10 & 14 \\ 10 & 8 \end{pmatrix} \end{matrix}$$

with

$$\begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} = \begin{pmatrix} 5,500 \\ 14,500 \\ 17,380 \end{pmatrix} .$$


55000
35000
25000
15000
8000
4000
2980
T1
T2
T3
T4
T5
T6
T7
T8
T9
T10
T11
T12
Production level at second period, θ2
Production level at first period, θ1
5
10
15
20
25
30
35

The third simplex:

$$\begin{pmatrix} P_1 \\ P_2 \\ P_3 \end{pmatrix} = \begin{pmatrix} T_8 \\ T_6 \\ T_2 \end{pmatrix} = \begin{matrix} \theta_1 & \theta_2 \\ \begin{pmatrix} 16 & 19 \\ 16 & 13 \\ 10 & 14 \end{pmatrix} \end{matrix}$$

with

$$\begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} = \begin{pmatrix} 4{,}060 \\ 5{,}500 \\ 14{,}500 \end{pmatrix} .$$

After 40 function evaluations:

$$\begin{pmatrix} P_1 \\ P_2 \\ P_3 \end{pmatrix} = \begin{pmatrix} 17.8 & 18.4 \\ 18.0 & 18.3 \\ 17.7 & 18.0 \end{pmatrix}$$

and

$$\begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} = \begin{pmatrix} 2965 \\ 2965 \\ 2966 \end{pmatrix}$$

After 55 function evaluations:

$$\begin{pmatrix} P_1 \\ P_2 \\ P_3 \end{pmatrix} = \begin{pmatrix} 17.8 & 18.3 \\ 17.8 & 18.2 \\ 17.9 & 18.2 \end{pmatrix}$$

and

$$\begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} = \begin{pmatrix} 2961 \\ 2961 \\ 2961 \end{pmatrix}$$

Simulated Annealing

1. We first introduce the so called randomized search before introducing the simulated annealing method. **Randomized Search**, also called probabilistic search: Such a method starts at a certain point. The next point to be considered is randomly generated from the definition domain. If it is a better point, we move there; otherwise we stay at the current point. Another random point is generated and the process is repeated. We can stop the process when a large enough random points in the definition domain has been evaluated.

   Consider the following optimization problem:

   Minimize $f(\mathbf{x})$

   Subject to $\mathbf{x} \in \Omega$

   where Ω is called the feasible region or the definition domain.

   A randomized search method first selects a random point $\mathbf{x}^{(0)} \in \Omega$. The next random point to be selected is usually "close" to $\mathbf{x}^{(0)}$ in some way.

   Define $N(\mathbf{x}) \subset \Omega$ to be a set of points that are "close" to $\mathbf{x}$. We call $N(\mathbf{x})$ the neighborhood of x. If we are currently at point x, the next random point to be generated is usually in $N(\mathbf{x})$. When a random point is generated from $N(\mathbf{x})$, we usually follow a certain statistical distribution. Commonly used distributions are uniform distribution and the normal distribution.

The so called naïve random search algorithm:

(1) Set k = 0. Select an initial point $\mathbf{x}^{(0)} \in \Omega$.
(2) Pick a new candidate point $\mathbf{z}^{(k)}$ at random from $N(\mathbf{x}^{(k)})$.
(3) If $f(\mathbf{z}^{(k)}) < f(\mathbf{x}^{(k)})$, then set $\mathbf{x}^{(k+1)} = \mathbf{z}^{(k)}$; else set $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)}$.
(4) If the stopping criterion has been met, then stop.
(5) Set k = k + 1, go to step (2).

This naïve algorithm can also be viewed as having the form

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \mathbf{d}^{(k)} .$$

It may be trapped in a local minimum if the neighborhood is too small. Making the neighborhood size sufficiently large will solve this problem at the expense of longer search time.

Example III.2: Find the minimal point of f(x) = (x-2)(x-3)(x-4) in the interval of [1.9, 4.1]. Let N(x) = [x-1, x+1]. Use the uniform distribution. Note that we need to ensure that the neighborhood is feasible all the time.

See the Excel program.

2. Simulated Annealing is an instance of a randomized search method. It incorporates a mechanism of climbing out of a local region.

   The algorithm:

   (1) Set k = 0. Select an initial point $\mathbf{x}^{(0)} \in \Omega$.

   (2) Pick a new candidate point $\mathbf{z}^{(k)}$ at random from $N(\mathbf{x}^{(k)})$.

   (3) Toss a coin with probability of HEAD equal to $p(k, f(\mathbf{z}^{(k)}), f(\mathbf{x}^{(k)}))$. If HEAD, then set $\mathbf{x}^{(k+1)} = \mathbf{z}^{(k)}$; else set $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)}$. Keep the best so far point and its corresponding function value.

   (4) If the stopping criterion has been met, then stop with the best so far point.

   (5) Set k = k + 1, go to step (2).

   In this algorithm, the probability of selecting the new point is set to be p. If the new point is a better point, p is equal to 1. If the new point is a worse point, we still have a non-zero probability of selecting it. We can choose p in a way such that the worse the new point, the lower the probability of it being selected (but not as low as zero). The probability of selecting the new point is called the acceptance probability. A typical choice for this probability is:

$$p(k, f(\mathbf{z}^{(k)}), f(\mathbf{x}^{(k)})) = \min\left\{1, \exp(-\frac{f(\mathbf{z}^{(k)}) - f(\mathbf{x}^{(k)})}{T_k})\right\}$$

   where exp is the exponential function and T_k represents a positive sequence called the temperature schedule or cooling schedule. The temperature parameter T_k usually follows a monotonically decreasing trend towards 0 as the iteration number k increases. Thus, the term cooling and annealing are used. When T_k is large, the probability of accepting a worse point is larger than when it is small.

This form of the acceptance probability is usually credited to Boltzman and leads to a simulated annealing algorithm.

A commonly used form for the temperature constant T_k is given below:

$$T_k = \frac{\gamma}{\log(k+2)}$$

where $\gamma > 0$ is a problem dependent constant and needs to be large enough to allow the algorithm to climb out of regions around local minimizes.

Particle Swarm Optimization

Youtube Link:

http://www.youtube.com/embed/88UVJpQGi88

Particle swarm optimization is related to artificial life in general, and to bird flocking, fish schooling, and swarming theory in particular. It was developed by Kennedy and Eberhart in 1995 on the basis of social interaction principles.

They followed the following five principles of swarm intelligence in development of PSO:

(1) Proximity Principle: The population should be able to carry out simple space and time computations.
(2) Quality Principle: The population should be able to respond to quality factors in the environment.
(3) Diverse Response: The population should not commit its activities along excessively narrow channels.
(4) Principle of Stability: The population should not change its mode of behavior every time the environment changes.
(5) Principle of Adaptability: The population must be able to change behavior mode when it's worth the computational price.

This PSO algorithm aims to mimic the behavior of animals and insects, such as a swarm of bees, a flock of birds, and a herd of wildebeests.

We can consider PSO as a type of randomized search method. It is different from the randomized methods we have discussed so far in that it does not focus on updating a single point $\mathbf{x}^{(k)}$. Instead, it updates a set of candidate points simultaneously. This set of points being considered in PSO is called a population, or a swarm. Each point or candidate solution in the swarm is called a particle.

The swarm can be viewed as an apparently disorganized population of moving individuals that tend to cluster together while each individual seems to be moving in a random direction.

A swarm consists of d particles flying around in an n-dimensional space in search of better solutions. Every particle has its own position (representing a candidate solution to the problem) and its own velocity (showing its direction of movement). These particles have ability to memorize their own previous positions, exchange information with one another, and use the available information for their own decision making.

Each particle adjusts its movement towards the optimum solution according to two factors: the best position it has visited so far (this can be called personal best) and the best position visited by the whole swarm (this can be called global best). The velocity of each particle is determined by its personal best and swarm's global best. This velocity is weighted by a random term, with separate random numbers being generated for velocities toward the personal best and the global best.

Notation:

- d: population size
- n: dimension of the problem
- f(**x**): objective function to be minimized
- $\mathbf{x}_i$ position of particle i
- $\mathbf{v}_i$ velocity of particle i
- $\mathbf{p}_i$ personal best point visited by particle i
- **g** global best point visited by the swarm
- • Hadamard product or Schur product of two matrices with the same dimensions. It is an entry-by-entry multiplication. It is ".*" in Matlab.

The PSO Algorithm:

(1) Set k = 0. Generate d initial points and d initial velocities: $\mathbf{x}_i^{(0)}$ and $\mathbf{v}_i^{(0)}$ for i = 1, 2, …, d. Set $\mathbf{p}_i^{(0)} = \mathbf{x}_i^{(0)}$ and $\mathbf{g} = \arg\min_{\mathbf{x} \in \{\mathbf{x}1, \ldots, \mathbf{x}d\}} f(\mathbf{x})$.

(2) For i = 1, …, d, generate random n-vectors $\mathbf{r}_i^{(k)}$ and $\mathbf{s}_i^{(k)}$, with their components uniformly distributed in (0, 1), and set
$\mathbf{v}_i^{(k+1)} = w\, \mathbf{v}_i^{(k)} + c_1\, \mathbf{r}_i^{(k)} \bullet (\mathbf{p}_i^{(k)} - \mathbf{x}_i^{(k)}) + c_2\, \mathbf{s}_i^{(k)} \bullet (\mathbf{g}^{(k)} - \mathbf{x}_i^{(k)})$.
$\mathbf{x}_i^{(k+1)} = \mathbf{x}_i^{(k)} + \mathbf{v}_i^{(k+1)}$,

(3) For i = 1, …, d, if $f(\mathbf{x}_i^{(k+1)}) < f(\mathbf{p}^{(k)})$, then set $\mathbf{p}_i^{(k+1)} = \mathbf{x}_i^{(k+1)}$; else set $\mathbf{p}_i^{(k+1)} = \mathbf{p}_i^{(k)}$.

(4) If there exists $i \in \{1, 2, \ldots, d\}$ such that $f(\mathbf{x}_i^{(k+1)}) < f(\mathbf{g}^{(k)})$, then set $\mathbf{g}^{(k+1)} = \mathbf{x}_i^{(k+1)}$; else set $\mathbf{g}^{(k+1)} = \mathbf{g}^{(k)}$.

(5) If stopping criterion is met, stop.

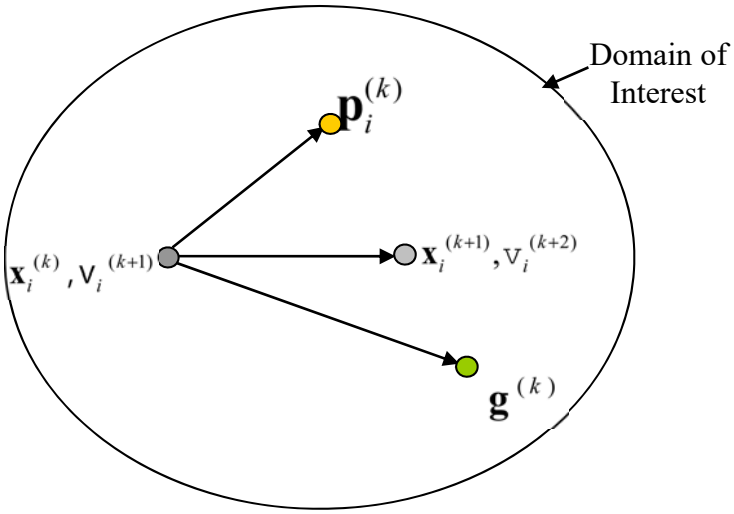
(6) Set k = k+1, go to step (2).

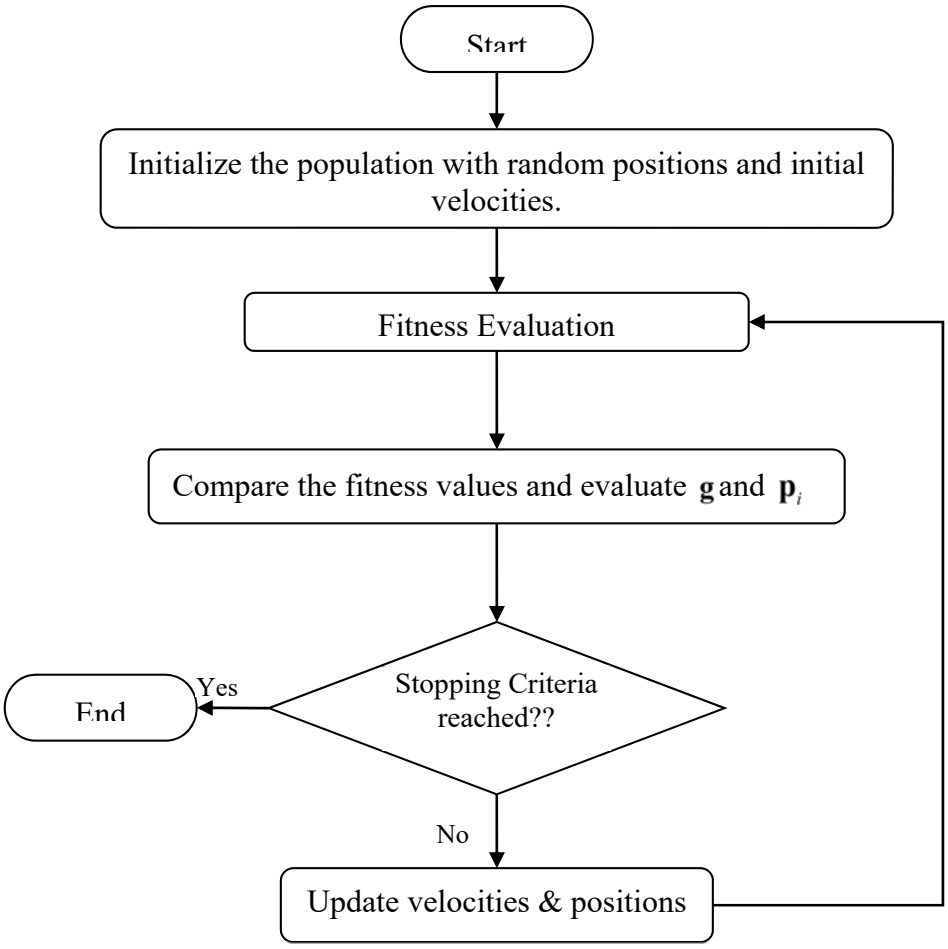
Parameter selection:

w the inertial constant, usually chosen to be slightly less than 1

c_1 a cognitive constant, representing how much the particle is drawn by its personal best, usually chosen to be around 2.

c_2 a social constant, representing how much the particle is drawn by the swarm's best point, usually chosen to be around 2.




Start
Initialize the population with random positions and initial velocities.
Fitness Evaluation
Compare the fitness values and evaluate g and p_i
Stopping Criteria reached??
Yes
End
No
Update velocities & positions

Example III.3: The objective function to be minimized is:

$$f(x_1, x_2, x_3) = x_1^2 + x_2^2 + (x_3 - 2)^2$$

Let the initial population consists of 4 particles (d=4). Each particle has three values of its coordinate (x_1, x_2 and x_3 (n=3)). Initial positions and velocities are randomly generated to start the algorithm as shown below.

Particle	$x_1(t)$	$x_2(t)$	$x_3(t)$	$f(x_1, x_2, x_3)$	
1	0.23	0.37	0.12	3.7242	
2	**0.19**	**0.42**	**0.24**	**3.3101**	**g**
3	0.21	0.34	0.15	3.5822	
4	0.24	0.48	0.23	3.4209	

We chose w = 1, $c_1 = c_2 = 2$. Randomly generated r and s are given below too. Initial velocities of the particles and their new positions are obtained as follows.

Particle	$v_1(t)$	$v_2(t)$	$v_3(t)$	r	s	$V_i(t+1)$	$v_1(t+1)$	$v_2(t+1)$	$v_3(t+1)$
1	0.08	0.14	0.31	0.8782	0.9427	⇨	0.0046	0.2343	0.5362
2	0.09	0.01	0.27	0.0140	0.4048		0.0900	0.0100	0.2700
3	0.21	0.31	0.19	0.0960	0.7922		0.1783	0.4368	0.3326
4	0.14	0.20	0.12	0.6501	0.8546		0.0545	0.0974	0.1371

$x_1(t+1)$	$x_2(t+1)$	$x_3(t+1)$	$f(x_1, x_2, x_3)$
0.2346	**0.6043**	**0.6562**	**2.2259**
0.2800	0.4300	0.5100	2.4834
0.3883	0.7768	0.4826	3.0566
0.2945	0.5774	0.3671	3.0866

Individual best values ($\mathbf{p}_i$) at the end of the 2nd population are:
For particle 1, **p1** = (0.2346, 0.6043, 0.6562) with f1 = 2.2259
For particle 2, **p2** = (0.2800, 0.4300, 0.5100) with f2 = 2.4834
For particle 3, **p3** = (0.3883, 0.7768, 0.4826) with f3 = 3.0566
For particle 4, **p4** = (0.2945, 0.5774, 0.3671) with f4 = 3.0866
The new **g** value after the 2nd population:

g = (0.2346, 0.6043, 0.6562) with f_g = 2.2259

Now generate random vector **r** = (0.1652, 0.4911, 0.7477, 0.5395), random vector **s** = (0.5166, 0.2006, 0.7393, 0.6017). With the earlier selected w=1 and c1=c2=2, we have calculated the new velocities:

For particle 1: **V1** = (0.0046, 0.2343, 0.5362)
For particle 2: **V2** = (0.0718, 0.0799, 0.3287)
For particle 3: **V3** = (-0.0490, 0.1817, 0.5893)
For particle 4: **V4** = (-0.0176, 0.1297, 0.4850)

Thus, the new points of the third population become:
X1 = (0.2392, 0.8385, 1.1925), f_1 = 1.4124
X2 = (0.3518, 0.5099, 0.8387), f_2 = 1.7325
X3 = (0.3393, 0.9585, 1.0719), f_3 = 1.8951
X4 = (0.2769, 0.7072, 0.8521), f_4 = 1.8944

With their personal bests:
P1 = (0.2392, 0.8385, 1.1925), with f_1 = 1.4124
P2 = (0.3518, 0.5099, 0.8387), with f_2 = 1.7325
P3 = (0.3393, 0.9585, 1.0719), with f_3 = 1.8951
P4 = (0.2769, 0.7072, 0.8521), with f_4 = 1.8944

The global best becomes:

g = (0.2392, 0.8385, 1.1925), with f_g = 1.4124

With new **g** and $\mathbf{p}_i$ values, new velocities and positions of the particles are calculated and fitness evaluation is done. It is repeated unless stopping criteria is met. The final **g** value obtained is the optimal/near-optimal solution.

Advantages of PSO:

- Simplicity of implementation
- Few parameters
- High convergence rate
- Parallel search

Pitfalls of Conventional PSO:

- Particles may move into the infeasible region
- Particles tend to cluster, i.e., converge too fast and get stuck at local optimum
- Conventional PSO is useful in continuous search space

To overcome some of the pitfalls, new variants and steps are proposed in PSO. If the movement carries the particles into infeasible region; some particles can be reinitialized into new positions that are feasible. To avoid premature clustering and local entrapment, time varying acceleration coefficients and time varying inertia weight factors may be used.

Modifications of PSO:

- New velocity update equation:
  $\mathbf{v}_i^{(k+1)} = u\,(\,\mathbf{v}_i^{(k)} + c_1\,\mathbf{r}_i^{(k)} \bullet (\mathbf{p}_i^{(k)} - \mathbf{x}_i^{(k)}) + c_2\,\mathbf{s}_i^{(k)} \bullet (\mathbf{g}^{(k)} - \mathbf{x}_i^{(k)}))$
  where u is called the constriction coefficient and is computed as

  $$u = \frac{2}{\left|2 - \phi - \sqrt{\phi^2 - 4\phi}\right|}$$

  $\varphi = c_1 + c_2$

  $\varphi > 4$

  For example for $\varphi = 4.1$, we have $u = 0.729$.

- The velocity may be capped at certain maximum and minimum values:
  $v = \min\{v_{max}, \max\{-v_{max}, v\}\}$.

Genetic Algorithms

1. A genetic algorithm is a randomized, population-based search technique that has its roots in the principles of genetics. Its general procedure is as follows:

   1) Starting with a group of points called a population: P(0) = {#1, #2, …, #N}, where N is called the population size. Evaluate the objective function value at each point, called the fitness value of each individual.

   2) Selection: Based on the fitness values of the individuals in the current population, a mating pool consisting of N individuals will be formed by randomly selecting members from the current population. An individual may be selected more than once. The fitter individuals have a higher probability being selected.

   3) Reproduction: The members in the mating pool are called parents. The operations like cross-over and mutation are used to generate off-springs of the current population. The size of the new population is still N.

   4) Evaluation: The fitness of the new population is evaluated. If a stopping criterion is met, stop; otherwise, goto step 2).

   To match the terminology used in the principles of genetics, we will focus on maximization problems in discussion of genetic algorithms, that is:

   Maximize f(**x**)
   Subject to $\mathbf{x} \in \Omega$

2. Base 2 and base 10 integer conversions:

   Base 2 integer number string: abcde, each symbol in the string is from {0, 1}. For example, 100101000

   Base 10 integer number string: ABCDEFG, each symbol is from {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}. For example, 2159

   abcde ➔ $a*2^4 + b*2^3 + c*2^2 + d*2^1 + e*2^0$

   ABCDEFG ➔ Divide the number and the quotient by 2 in sequence until the quotient is 0. Write the remainders in reverse order.

   Example: 101100 ➔ 44
   21 ➔ 10101

3. How to represent decimal numbers in [-3, 3] up to 4 decimal places using binary numbers?

   [-3.0000, +3.0000] ➔ [-30000, +30000] ➔ [-32767, +32767] ➔ $[-2^{15}+1, +2^{15}-1]$ ➔ $[0, 2^{16}-1]$

   We can use a 16-bit binary string to represent any real number in the range [-3,+3] accurate up to 4 decimal places. Actually it can represent a larger range, that is, [-3.2767, +3.2767].

   For example, the binary strings:
   0 000000000000000: 0 (or -0)
   1 000000000000000: 0 (or +0)
   0 100000000000000: -2^14/10,000 = -1.6384
   1 110000000000000: +(2^14+2^13)/10,000 = +2.4576
   0 111111111111111: -2^15+1 = -32,767 or 3.2767

4. Chromosomes and representation schemes: Any number that a decision variable may take up to certain accuracy in the feasible region Ω must be represented by a binary string of a fixed identical length. A different decision variable may require a different string length. The binary strings for all decision variables are concatenated to form the so called chromosome. A chromosome represents a point in the feasible region. A chromosome for a specific optimization problem has a fixed length. We use L to denote the length of the chromosome.

   Consider $f(x_1, x_2, x_3) = x_1 + x_2 + x_3$, where x_1, x_2, x_3 are the number of students registered in the three courses (300 level, 500 level, and 600 level) that I teach each year. Historically, they are around 120, 16, and 27. How do I represent the points in the feasible region?

   x_1: binary string length: 10? (Up to 1024 students?)
   x_2: binary string length: 7? (Up to 128 students?)
   x_3: binary string length: 6? (Up to 64 students?)

   (123, 25, 16) ➔ xxxxxxxxxxyyyyyyyzzzzzz
   0001111011001100 1010000

   Each chromosome is supposed to correspond to a point in the feasible region. For each point, we can evaluate its objective function value. In GA, we call this objective function value the fitness value of this chromosome. The higher the fitness value, the better the candidate point. We focus on maximization problems in GA terminology.

   x is a chromosome, f(**x**) is the fitness value of the chromosome.

   L: Length of the binary string of the chromosome

   N: Population size, the number of chromosomes in each population

P(k): The population in iteration k. Each member $\mathbf{x}^{(k)}$ has a fitness value $f(\mathbf{x}^{(k)})$.

5. Selection: The principle of survival of the fittest is used here. Or the fitter individuals have a higher chance of survival.

   The mating pool with N chromosomes is formed by selecting members from the current population. A chromosome with a larger fitness value has a larger chance of being selected multiple times to enter the mating pool. One implementation of this selection scheme is called the roulette-wheel scheme:
   - In population P(k)
   - The mating pool M(k) to be formed
   - A member in P(k) has probability $f(\mathbf{x}^{(k)})/F(k)$ to enter the mating pool, where F(k) is the sum of the fitness values of all members of P(k).

   An alternative implementation of the selection scheme is called the tournament scheme: A pair of chromosomes are selected from the current population at random. Their fitness values are compared, and the fitter one is selected to enter the mating pool. This process is repeated until we have collected N chromosomes in the mating pool.

6. Evolution: We apply the cross-over operation and the mutation operation to the chromosomes in the mating pool to produce off-springs of the current population.

   Paring of parents: From the mating pool of N chromosomes, randomly select two as a pair of parents. Select another pair from the remaining members, until all are paired up.

   Cross-over: Each pair of parents is used to produce two off-springs. Each chromosome has L binary bits.

One-point cross over: A random integer number x in the range [1, L] is generated. At position x, the chromosomes of the two parents are swapped to produce two offspring.

Example: L = 6; Parent 1: 000000 Parent 2: 111111
Random generated one-point cross-over position: 4
Offpring 1: 000011 Offspring 2: 111100
{0, 63} ➔ {3, 60}

Two-point cross over: Two random integers in the range [1, L] are generated. They denote the positions of substring swap for the parent chromosomes.

Example: L =9; Parent 1: 000000000 Parent 2: 111111111
Random generated two cross-over points: 3 and 7
Offspring 1: 000111100 Offspring 2: 111000011
{0, 511} ➔ {60, 451}

We often use the so-called cross-over probability to determine if a pair of parents will exchange bits of chromosomes. If the cross-over probability is 90%, then 10% of the parents will simply move on to the next generation (copied to the next generation).

After cross-over operations are complete, the mating pool consists of N chromosomes (children) that are produced by the N parents. We may now perform the mutation operation.

Mutation: Consider a chromosome of length L to be mutated. We can generate a random integer numbers in the range [1, L]. The bit value at this position is changed from 0 to 1 or from 1 to 0.

Example: L = 9, Starting chromosome: 110001111
Randomly generated position: 4
Resulting chromosome: 110101111
399 ➔ 431

There are many other ways of performing mutation.

We often use the so-called mutation probability to determine if mutation is to be performed on a chromosome. This probability is usually chosen to be very small, for example, 0.01. In this case, only 1% of the chromosomes will experience the mutation operation.

After all chromosomes in the mating pool after the cross-over operation are considered for mutation, we have generated a new population of chromosomes.

7. Genetic algorithm:
   1) Set k = 0. Generate an initial population P(0).
   2) Evaluate P(k).
   3) If the stopping criterion is met, stop.
   4) Select M(k) from P(k).
   5) Evolve M(k) to form P(k+1).
   6) Set k = k+1, go to step 2).

   A flow chart is given below.

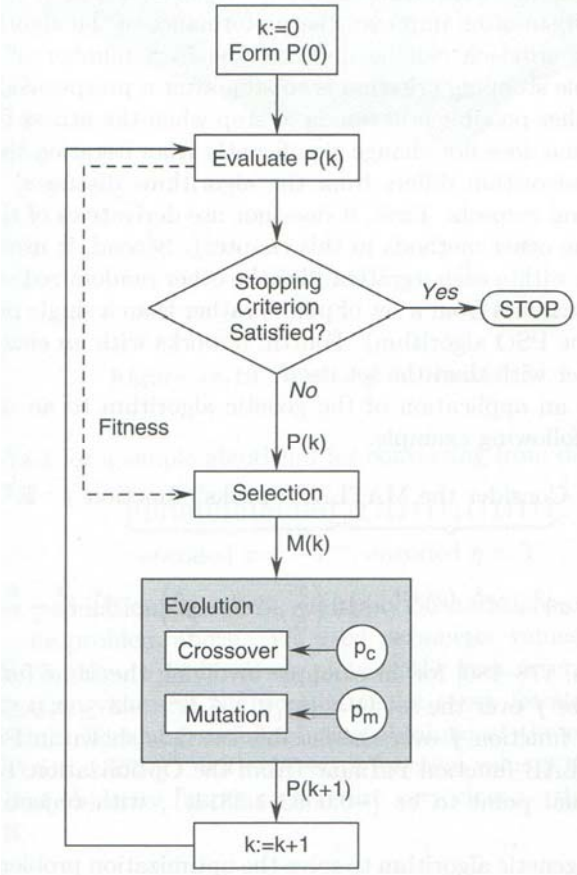


Figure 14.9 Flowchart for the genetic algorithm.

8. Notes:
   1) The best-so-far chromosome is updated from generation to generation. We may consider copying it into every new generation. This practice is referred to as “elitism”.
   2) The stopping criterion may be the maximum number of iterations, when the fitness value of the best-so-far chromosome does not change much, or others.

A figure for Example 14.3 in the textbook is given below.

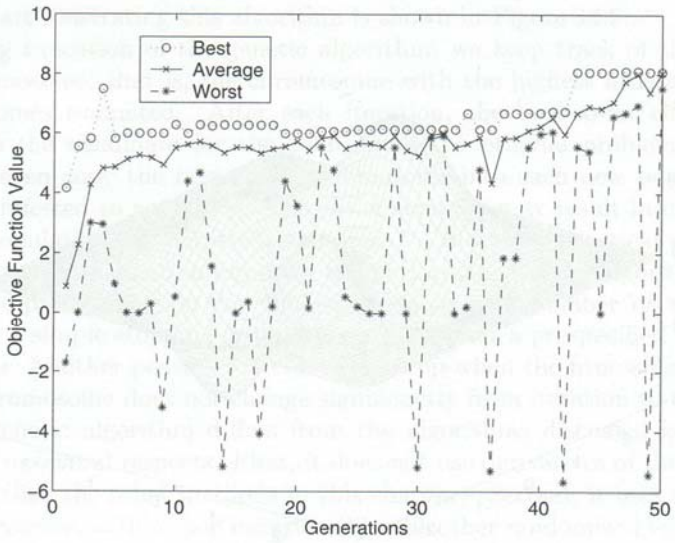


Figure 14.11 The best, average, and worst objective function values in the population for every iteration (generation) of the genetic algorithm in Example 14.3.