POLYTECHNIC UNIVERSITY
Department of Computer and Information Science

ADALINE for Pattern Classification

K. Ming Leung

Abstract: A supervised learning algorithm known as the Widrow-Hoff rule, or the Delta rule, or the LMS rule, is introduced to train neuron networks to classify patterns into two or more categories.

Directory

Last Revision Date: February 26, 2008

Table of Contents

1. Simple ADELINE for Pattern Classification

Although the Perceptron learning rule always converges, in fact in a finite number of steps, to a set of weights and biases, provided that such a set exists, the set obtained is often not the best in terms of robustness. We will discuss here the ADALINE, which stands for **Ada**ptive **Li**near **Ne**uron, and a learning rule which is capable, at least in principle, of finding such a robust set of weights and biases.

The architecture for the NN for the ADALINE is basically the same as the Perceptron, and similarly the ADALINE is capable of performing pattern classifications into two or more categories. Bipolar neurons are also used. The ADALINE differs from the Perceptron in the way the NNs are trained, and in the form of the transfer function used for the output neurons during training. For the ADALINE, the transfer function is taken to be the identity function *during training*. However, after training, the transfer function is taken to be the bipolar Heaviside step function when the NN is used to classify any input patterns. Thus the transfer function is

$$f(y_{\text{in}}) = y_{\text{in}}, \qquad \text{during training,}$$

$$f(y_{\text{in}}) = \begin{cases} +1, & \text{if } y_{\text{in}} \geq 0 \\ -1, & \text{if } y_{\text{in}} < 0 \end{cases} \qquad \text{after training.}$$

We will first consider the case of classification into 2 categories only, and thus the NN has only a single output neuron. Extension to the case of multiple categories is treated in the next section.

The total input received by the output neuron is given by

$$y_{\text{in}} = b + \sum_{i=1}^{N} x_i w_i.$$

Just like Hebb's rule and the Perceptron learning rule, the Delta rule is also a supervised learning rule. Thus we assume that we are given a training set:

$$\{\mathbf{s}^{(q)}, t^{(q)}\}, \qquad q = 1, 2, \ldots, Q.$$

where $\mathbf{s}^{(q)}$ is a training vector, and $t^{(q)}$ is its corresponding targeted output value.

Also like Hebb's rule and the Perceptron rule, one cycles through the training set, presenting the training vectors one at a time to the

NN. For the Delta rule, the weights and bias are updated so as to minimize the square of the difference between the net output and the target value for the particular training vector presented at that step.

Notice that this procedure is not exactly the same as minimizing the overall error between the NN outputs and their corresponding target values for all the training vectors. Doing so would require the solution to a large scale optimization problem involving N weight components and 1 bias.

1.1. Multi-Parameter Minimization

To better understand the updating procedure for the weights and bias in the Delta rule, we need to digress and consider the topic of multi-parameter minimization. We assume that $E(\mathbf{w})$ is a scalar function of a vector argument, $\mathbf{w}$. We want to find the point $\mathbf{w} \in \mathcal{R}^n$ at which E takes on its minimum value.

Suppose we want to find the minimum value iteratively starting with $\mathbf{w}(0)$. The iteration amounts to

$$\mathbf{w}(k+1) = \mathbf{w}(k) + \Delta\mathbf{w}(k), \qquad k = 0, 1, \ldots.$$

The question is how should the changes in the weight vector be chosen in order that we end up with a lower value for E:

$$E(\mathbf{w}(k+1)) < E(\mathbf{w}(k)).$$

For sufficiently small $\Delta\mathbf{w}(k)$, we obtain from Taylor's theorem

$$E(\mathbf{w}(k+1)) = E(\mathbf{w}(k) + \Delta\mathbf{w}(k)) \approx E(\mathbf{w}(k)) + \mathbf{g}(k) \cdot \Delta\mathbf{w}(k),$$

where $\mathbf{g}(k) = \nabla E(\mathbf{w})|_{\mathbf{w}=\mathbf{w}(k)}$ is the gradient of $E(\mathbf{w})$ at $\mathbf{w}(k)$. It is clear that $E(\mathbf{w}(k+1)) < E(\mathbf{w}(k))$ if $\mathbf{g}(k) \cdot \Delta\mathbf{w}(k) < 0$. The largest decrease in the value of $E(\mathbf{w})$ occurs in the direction $\Delta\mathbf{w}(k) = -\alpha\mathbf{g}(k)$, if α is sufficiently small and positive. This direction is called the **steepest descent direction**, and α controls the size of the step and is called the learning rate. Thus starting from $\mathbf{w}(0))$, the idea is to find a minimum of the function $E(\mathbf{w})$ iteratively by making successive steps along the local gradient direction, according to

$$\mathbf{w}(k+1) = \mathbf{w}(k) - \alpha\mathbf{g}(k), \qquad k = 0, 1, \ldots.$$

This method of finding the minimum is known as the steepest descent method.

This is a greedy method which may lead to convergence to a local but not a global minimum of E.

2. Delta Rule

Suppose at the k-th step in the training process, the current weight vector and bias are given by $\mathbf{w}(k)$ and $b(k)$, respectively, and the q-th training vectors, $\mathbf{s}(k) = \mathbf{s}^{(q)}$, is presented to the NN. The total input received by the output neuron is

$$y_{\text{in}} = b(k) + \sum_{i=1}^{N} s_i(k) w_i(k).$$

Since the transfer function is given by the identity function during training, the output of the NN is

$$y(k) = y_{\text{in}} = b(k) + \sum_{i=1}^{N} s_i(k) w_i(k).$$

However the target output is $t(k) = t^{(q)}$, and so if $y(k) \neq t(k)$ then there is an error given by $y(k) - t(k)$. This error can be positive or

negative. The Delta rule aims at finding the weights and bias so as to minimize the square of this error

$$E(\mathbf{w}(k)) = (y(k) - t(k))^2 = \left(b(k) + \sum_{i=1}^{N} s_i(k) w_i(k) - t(k) \right)^2 .$$

We can absorb the bias term by introducing an extra input neuron, X_0, so that its activation (signal) is always fixed at 1 and its weight is the bias. Then the square of the error in the k-th step is

$$E(\mathbf{w}(k)) = \left(\sum_{i=0}^{N} s_i(k) w_i(k) - t(k) \right)^2 .$$

The gradient of this function, $\mathbf{g}(k)$, in a space of dimension $N+1$ (N weights and 1 bias) is

$$g_j(k) = \partial_{w_j(k)} E(\mathbf{w}(k)) = 2 \left(\sum_{i=0}^{N} s_i(k) w_i(k) - t(k) \right) s_j(k).$$

Using the steepest descent method, we have

$$\mathbf{w}(k+1) = \mathbf{w}(k) - 2\alpha \left(\sum_{i=0}^{N} s_i(k) w_i(k) - t(k) \right) \mathbf{s}(k).$$

The $i = 1, 2, \ldots, N$ components of this equation gives the updating rule for the weights. The zeroth component of this equation gives the updating rule for the bias

$$b(k+1) = b(k) - 2\alpha \left(\sum_{i=0}^{N} s_i(k) w_i(k) - t(k) \right).$$

Notice that in the textbook by Fausett, the factors of 2 are missing from these two updating formulas. We can also say that the learning rate there is twice the value here.

We will now summarize the Delta rule. To save space, we use vector notation, where vectors are denoted by boldface quantities.

The Delta rule is:

1. Set learning rate α and initialize weights and bias.
2. Repeat the following steps, while cycling through the training set $q = 1, 2, \ldots, Q$, until changes in the weights and bias are insignificant.
   (a) Set activations for input vector $\mathbf{x} = \mathbf{s}^{(q)}$.
   (b) Compute total input for the output neuron:

$$y_{\text{in}} = \mathbf{x} \cdot \mathbf{w} + b$$

   (c) Set $y = y_{\text{in}}$.
   (d) Update the weights and bias

$$\mathbf{w}^{\text{new}} = \mathbf{w}^{\text{old}} - 2\alpha(y - t^{(q)})\mathbf{x},$$

$$b^{\text{new}} = b^{\text{old}} - 2\alpha(y - t^{(q)}).$$

Notice that for the Delta rule, unlike the Perceptron rule, training does not stop even after all the training vectors have been correctly classified. The algorithm continuously attempts to produce more ro-

bust sets of weights and bias. Iteration is stopped only when changes in the weights and bias are smaller than a preset tolerance level.

In general, there is no proof that the Delta rule will always lead to convergence, or to a set of weights and bias that enable the NN to correctly classify all the training vectors. One also needs to experiment with the size of the learning rate. Too small a value may require too many iterations. Too large a value may lead to non-convergence.

Also because the identity function is used as the transfer function during training, the error at each step of the training process may never become small, even though an acceptable set of weights and bias may have already been found. In that case the weights will continually change from one iteration to the next. The amount of changes are of course proportional to α.

Therefore in some cases, one may want to gradually decrease α towards zero during iteration, especially when one is close to obtaining the best set of weights and bias. Of course there are many ways in which α can be made to approach zero.

3. Exact Optimal Choice of Weights and Bias

Actually one can find, at least in principle, a set of weights and bias that will perform best for a given training set. To see this, it is better to absorb the bias to simplify the expressions. What this problem intends to accomplish mathematically is to find a vector $\mathbf{w}$ that minimizes the overall squares of the errors (the least mean squares, or LMS)

$$F(\mathbf{w}) = \frac{1}{Q}\sum_{q=1}^{Q}(y - t^{(q)})^2 = \frac{1}{Q}\sum_{q=1}^{Q}\left(\sum_{i=0}^{N} s_i^{(q)} w_i - t^{(q)}\right)^2 .$$

Since $F(\mathbf{w})$ is quadratic in the weight components, the solution can be readily obtained, at least formally. To obtain the solution, we take the partial derivatives of $F(\mathbf{w})$, set them to zero, and solve the resulting set of equations. Since $F(\mathbf{w})$ is quadratic in the weight components, its partial derivatives are linear, and the resulting equation for the weight components are linear and can therefore be solved.

Taking the partial derivative of $F(\mathbf{w})$ with respect to the j-th

component of the weight vector gives

$$\begin{aligned}\partial_{w_j} F(\mathbf{w}) &= \frac{2}{Q}\sum_{q=1}^{Q}(y-t^{(q)})\partial_{w_j}\sum_{i=0}^{N}s_i^{(q)}w_i = \frac{2}{Q}\sum_{q=1}^{Q}(y-t^{(q)})s_j^{(q)} \\ &= \frac{2}{Q}\sum_{q=1}^{Q}\left(\sum_{i=0}^{N}s_i^{(q)}w_i - t^{(q)}\right)s_j^{(q)} = 2\left(\sum_{i=0}^{N}w_i C_{ij} - v_j\right),\end{aligned}$$

where we have defined the correlation matrix $\mathbf{C}$ such that

$$C_{ij} = \frac{1}{Q}\sum_{q=1}^{Q}s_i^{(q)}s_j^{(q)}$$

and a vector $\mathbf{v}$ having components

$$v_j = \frac{1}{Q}\sum_{q=1}^{Q}t^{(q)}s_j^{(q)}.$$

Setting the partial derivatives to zero gives the set of linear equations (written in matrix notation):

$$\mathbf{wC} = \mathbf{v}.$$

Notice that the correlation matrix $\mathbf{C}$ and the vector $\mathbf{v}$ can be easily computed from the given training set.

Assuming that the correlation matrix is nonsingular, the solution is therefore given by

$$\mathbf{w} = \mathbf{v}\mathbf{C}^{-1},$$

where $\mathbf{C}^{-1}$ is the inverse matrix for $\mathbf{C}$. Notice that the correlation matrix is symmetric and has dimension $(N+1) \times (N+1)$.

Although the exact solution is formally available, computing it this way requires the computation of the inverse of matrix $\mathbf{C}$ or solving a system of linear equations. The computational complexity involved is of $\mathcal{O}(N+1)^3$. For most practical problems, N is so large that computing the solution this way is really not feasible.

4. Application: Bipolar Logic Function: AND

We use the Delta rule here to train the same NN (the bipolar logic function: AND) that we have treated before using different training rules. The training set is given by the following table:

q	$\mathbf{s}^{(q)}$	$t^{(q)}$
1	[1 1]	1
2	[1 -1]	-1
3	[-1 1]	-1
4	[-1 -1]	-1

We assume that the weights and bias are initially zero, and apply the Delta rule to train the NN. We find that for a learning rate α larger than about 0.3, there is no convergence as the weight components increase without bound. For α less than 0.3 but larger than 0.16, the weights converge but to values that fail to correctly classify all the training vectors. The weights converge to values that correctly classify all the training vectors if α is less than about 0.16. They become closer and closer to the most robust set of weights and bias when α is below 0.05.

We also consider here the exact formal solution given in the last section. We will absorb the bias by appending a 1 in the leading position of each of the training vectors so that the training set is

q	$\mathbf{s}^{(q)}$	$t^{(q)}$
1	[1 1 1]	1
2	[1 1 -1]	-1
3	[1 -1 1]	-1
4	[1 -1 -1]	-1

We first compute the correlation matrix

$$\begin{aligned}
\mathbf{C} &= \frac{1}{4}\sum_{q=1}^{4}\mathbf{s}^{(q)T}\mathbf{s}^{(q)} \\
&= \frac{1}{4}\left(\begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}\begin{bmatrix} 1 & 1 & 1 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix}\begin{bmatrix} 1 & 1 & -1 \end{bmatrix}\right. \\
&\quad \left. + \begin{bmatrix} 1 \\ -1 \\ 1 \end{bmatrix}\begin{bmatrix} 1 & -1 & 1 \end{bmatrix} + \begin{bmatrix} 1 \\ -1 \\ -1 \end{bmatrix}\begin{bmatrix} 1 & -1 & -1 \end{bmatrix}\right) \\
&= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}
\end{aligned}$$

Since $\mathbf{C}$ is an identity matrix (the training vectors are as independent of each other as they can be), its inverse is just itself. Then we

compute the vector $\mathbf{v}$

$$\begin{aligned}\mathbf{v} &= \frac{1}{4}\sum_{q=1}^{4} t^{(q)}\mathbf{s}^{(q)} \\ &= \frac{1}{4}\Big(\begin{bmatrix} 1 & 1 & 1 \end{bmatrix} - \begin{bmatrix} 1 & 1 & -1 \end{bmatrix} \\ &\quad - \begin{bmatrix} 1 & -1 & 1 \end{bmatrix} - \begin{bmatrix} 1 & -1 & -1 \end{bmatrix} \Big) \\ &= \begin{bmatrix} -\frac{1}{2} & \frac{1}{2} & \frac{1}{2} \end{bmatrix}.\end{aligned}$$

Therefore we have

$$\mathbf{W} = \mathbf{v}\mathbf{C}^{-1} = \begin{bmatrix} -\frac{1}{2} & \frac{1}{2} & \frac{1}{2} \end{bmatrix}.$$

This means that

$$b = -\frac{1}{2}, \qquad \mathbf{W} \begin{bmatrix} \frac{1}{2} & \frac{1}{2} \end{bmatrix},$$

and so the best decision is boundary is given by the line

$$x_2 = 1 - x_1,$$

which we know before is the correct result.

5. NN with multiple Output Neurons

We now extend our discussions here to NN with multiple output neurons, and thus are capable of clustering input vectors into more than 2 classes. As before, we need to have M neurons in the output layer.

We will absorb the biases as we did before with the Perceptron. Suppose at the k-th step in the training process, the current weight matrix and bias vector are given by $\mathbf{W}(k)$ and $\mathbf{b}(k)$, respectively, and one of the training vectors $\mathbf{s}(k) = \mathbf{s}^{(q)}$, for some integer q between 1 and Q, is presented to the NN. The output of neuron $\mathbf{Y}_j$ is

$$y_j(k) = y_{\text{in,j}} = \sum_{i=0}^{N} s_i(k) w_{ij}(k).$$

However the target is $t_j(k) = t_j^{(q)}$, and so the error is $y_j(k) - t_j(k)$. Thus we want to find a set of w_{mn} that minimizes the quantity

$$E(\mathbf{W}(k)) = \sum_{j=1}^{M} \left(y_j(k) - t_j(k)\right)^2 = \left(\sum_{i=0}^{N} s_i(k) w_{ij}(k) - t_j(k)\right)^2.$$

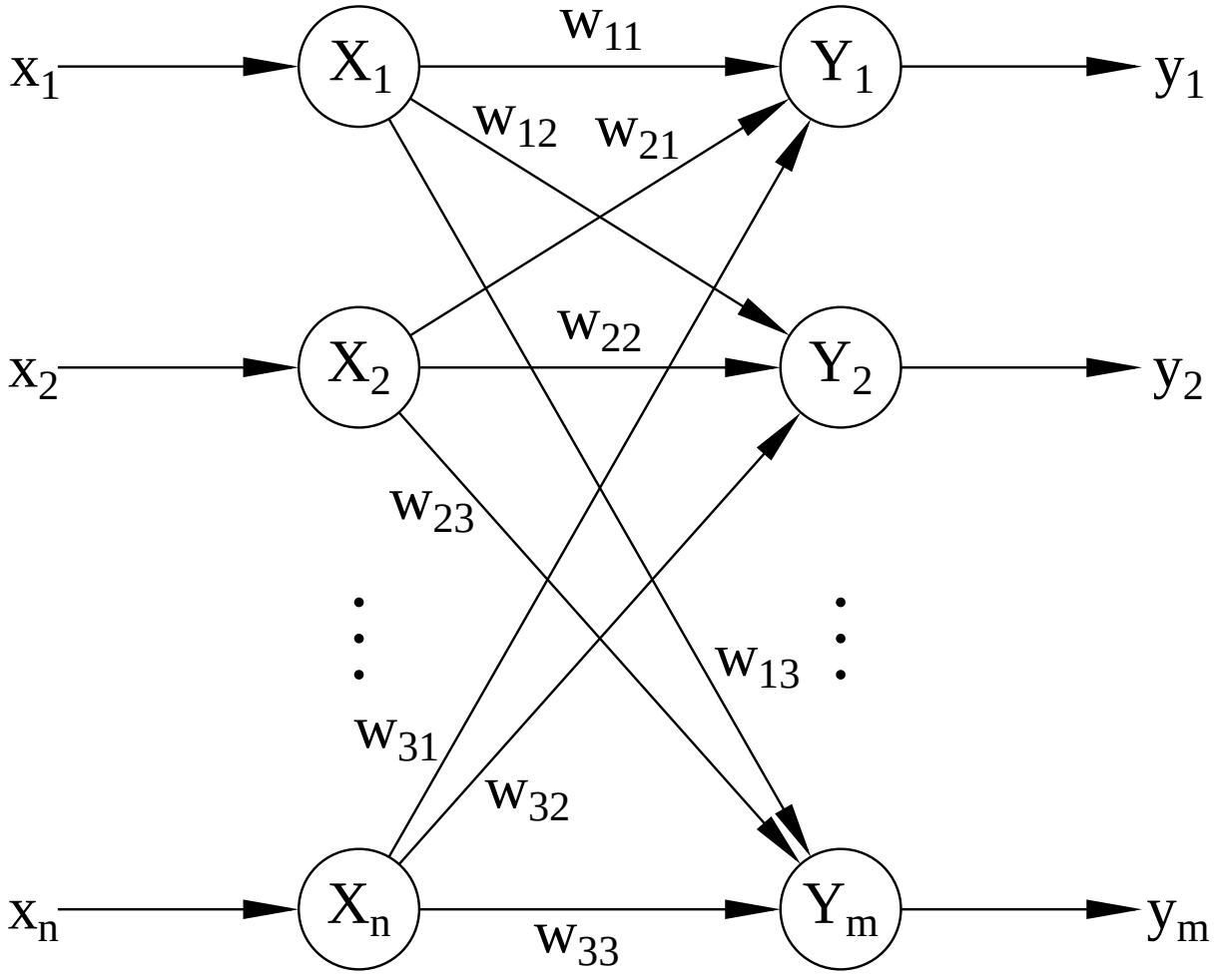

Figure 1: *A neural network for multi-category classification.*

We take the gradient of this function with respect to w_{mn}

$$\begin{aligned}\partial_{w_{mn}} E(\mathbf{W}(k)) &= \partial_{w_{mn}} \sum_{j=1}^{M} \left(y_j(k) - t_j(k)\right)^2 \\ &= 2 \sum_{j=1}^{M} \left(y_j(k) - t_j(k)\right) \partial_{w_{mn}} y_j.\end{aligned}$$

$$\partial_{w_{mn}} y_j = \partial_{w_{mn}} \sum_{i=0}^{N} s_i(k) w_{ij}(k) = \sum_{i=0}^{N} s_i(k) \partial_{w_{mn}} w_{ij}(k)$$

Since

$$\partial_{w_{mn}} w_{ij}(k) = \delta_{i,m}\delta_{j,n},$$

thus

$$\partial_{w_{mn}} y_j = \partial_{w_{mn}} \sum_{i=0}^{N} s_i(k) w_{ij}(k) = \sum_{i=0}^{N} s_i(k) \delta_{i,m}\delta_{j,n} = \delta_{j,n} s_m(k),$$

and so we have

$$\begin{aligned}\partial_{w_{mn}} E(\mathbf{W}(k)) &= 2\sum_{j=1}^{M} (y_j(k) - t_j(k))\, \delta_{j,n} s_m(k) \\ &= 2 s_m(k)\, (y_n(k) - t_n(k))\,.\end{aligned}$$

Using the steepest descent method, we have

$$\mathbf{w}_{ij}(k+1) = \mathbf{w}_{ij}(k) - 2\alpha s_i(k)\, (y_j(k) - t_j(k))\,.$$

The $i = 1, 2, \ldots, N$ components of this equation gives the updating rule for the weights. The $i = 0$ component of this equation gives the updating rule for the bias

$$b_j(k+1) = b_j(k) - 2\alpha\, (y_j(k) - t_j(k))\,.$$

The general multiple output neuron Delta rule is:

1. Set learning rate α and initialize weights and bias.
2. Repeat the following steps, while cycling through the training set $q = 1, 2, \ldots, Q$, until changes in the weights and biases are within tolerance.
   (a) Set activations for input vector $\mathbf{x} = \mathbf{s}^{(q)}$.
   (b) Compute total input for the output neuron:

$$\mathbf{y}_{\text{in}} = \mathbf{x} \cdot \mathbf{W} + \mathbf{b}$$

   (c) Set $\mathbf{y} = \mathbf{y}_{\text{in}}$.
   (d) Update the weights and biases

$$\mathbf{W}^{\text{new}} = \mathbf{W}^{\text{old}} - 2\alpha \mathbf{x}^T(\mathbf{y} - \mathbf{t}^{(q)}),$$

$$\mathbf{b}^{\text{new}} = \mathbf{b}^{\text{old}} - 2\alpha(\mathbf{y} - \mathbf{t}^{(q)}).$$

6. An Example

We will now treat the same example that we have considered before for the Perceptron with multiple output neurons. We use bipolar output neurons and the training set:

(class 1)

$$\mathbf{s}^{(1)} = \begin{bmatrix} 1 & 1 \end{bmatrix}, \mathbf{s}^{(2)} = \begin{bmatrix} 1 & 2 \end{bmatrix} \quad \text{with} \quad \mathbf{t}^{(1)} = \mathbf{t}^{(2)} = \begin{bmatrix} -1 & -1 \end{bmatrix}$$

(class 2)

$$\mathbf{s}^{(3)} = \begin{bmatrix} 2 & -1 \end{bmatrix}, \mathbf{s}^{(4)} = \begin{bmatrix} 2 & 0 \end{bmatrix} \quad \text{with} \quad \mathbf{t}^{(3)} = \mathbf{t}^{(4)} = \begin{bmatrix} -1 & 1 \end{bmatrix}$$

(class 3)

$$\mathbf{s}^{(5)} = \begin{bmatrix} -1 & 2 \end{bmatrix}, \mathbf{s}^{(6)} = \begin{bmatrix} -2 & 1 \end{bmatrix} \quad \text{with} \quad \mathbf{t}^{(5)} = \mathbf{t}^{(6)} = \begin{bmatrix} 1 & -1 \end{bmatrix}$$

(class 4)

$$\mathbf{s}^{(7)} = \begin{bmatrix} -1 & -1 \end{bmatrix}, \mathbf{s}^{(8)} = \begin{bmatrix} -2 & -2 \end{bmatrix} \quad \text{with} \quad \mathbf{t}^{(7)} = \mathbf{t}^{(8)} = \begin{bmatrix} 1 & 1 \end{bmatrix}$$

It is clear that $N = 2$, $Q = 8$, and the number of classes is 4. The number of output neuron is chosen to be $M = 2$ so that $2^M = 4$ classes can be represented.

Our exact calculation of the weights and bias for the case of a single output neuron can be extended to the case of multiple output neurons. One can then obtain the following exact results for the weights and biases:

$$\mathbf{W} = \begin{bmatrix} \frac{-91}{153} & \frac{1}{6} \\ \frac{-8}{153} & \frac{-2}{3} \end{bmatrix}$$

$$\mathbf{b} = \begin{bmatrix} \frac{2}{153} & \frac{1}{6} \end{bmatrix}$$

Using these exact results, we can easily see how good or bad our iterative solutions are.

It should be remarked that the most robust set of weights and biases is determined only by a few training vectors that lie very close to the decision boundaries. However in the Delta rule, all training vectors contribute in some way. Therefore the set of weights and biases obtained by the Delta rule is not necessarily always the most robust.

The Delta rule usually gives convergent results if the learning rate is not too large. The resulting set of weights and biases typically leads

to correct classification of all the training vectors, provided such a set exist. How close this set is to the best choice depends on the starting weights and biases, the learning rate and the number of iterations. We find that for this example much better convergence can be obtained if the learning rate at step k is set to be $\alpha = 1/k$.

References

[1] See Chapter 2 in Laurene Fausett, "Fundamentals of Neural Networks - Architectures, Algorithms, and Applications", Prentice Hall, 1994.