CSE-E4810 Machine Learning and Neural Networks (5 cr)

Lecture 4: Multilayer Perceptron Networks and Backpropagation Algorithms

Prof. Juha Karhunen

https://mycourses.aalto.fi/course/view.php?id=13086

Introduction

- This lecture deals with **multilayer perceptron networks** trained by **backpropagation** type learning algorithms.
- This has been the most widely used neural network method which is still important.
- It is discussed in Chapter 4 in Du's and Swamy's book.
- However, our notations are from the earlier used textbook F. Ham and I. Kostanic, "Principles of Neurocomputing for Science and Engineering", McGraw-Hill, 2001.
- Some material comes from the book S. Haykin, "Neural Networks: A Comprehensive Foundation", Prentice-Hall, 1998.
- The discussion is pretty similar in all these 3 books, and the detailed notation is necessarily complicated.

Feedforward multilayer perceptron network

Overall description

- The architecture of a standard feedforward multilayer perceptron (MLP) network is shown in Figure 1.
- We discuss here exclusively feedforward MLPs, but there exists also recurrent MLP networks.
- In the **input layer** of the MLP network in Figure 1, the data (input) vectors $\mathbf{x}$ are fed to the network.
- In the j:th node (neuron) of the input layer, j:th component x_j of the data vector $\mathbf{x}$ is fed to the MLP network.
- No computations take place in the input layer.
- Therefore it is usually not counted as a proper layer.

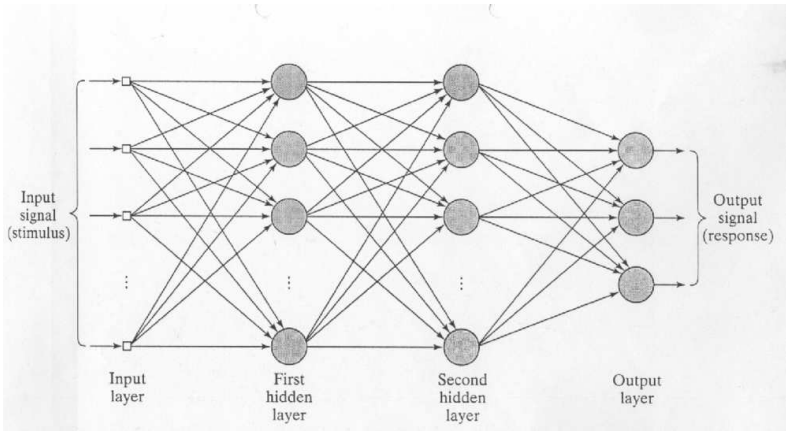

Figure 1: The architecture of a multilayer preceptron network with two hidden layers.

- All the components of the input vector are then inputted to all the neurons in the first **hidden layer.**
- Similarly, the outputs of the neurons in the first hidden layer are inputted to all the neurons in the second hidden layer.
- And furthermore all their outputs are inputted to the neurons in the **output layer.**
- The output vector $\mathbf{y}$ of the MLP network in Figure 1 consists of the output of the neurons in the output layer.
- All the computations take place in the hidden layers and the output layer.
- Note that after an input data vector is fed to the network, the signals proceed layer-by-layer through the network.
- There are no feedback connections.

- The neurons in each layer are not connected to each other.
- The network is called **fully connected,** because all the neurons of a layer receive signals from all the neurons in the previous layer.
- After learning, MLP network operates in this **feedforward manner.**
- The learning is **supervised**.
- That is, there are available some K known training pairs $\{\mathbf{x}(k), \mathbf{d}(k)\}$, $k = 1, 2, \ldots, K$.
- There $\mathbf{d}(k)$ is the **desired response** (output) corresponding to the input vector $\mathbf{x}(k)$.
- The output vector $\mathbf{y}(k)$ produced by the MLP network for the input vector $\mathbf{x}(k)$ is compared to the desired response $\mathbf{d}(k)$.
- The error signals are then propagated backwards through the MLP network layer-by-layer from the output layer to the first hidden layer.

- The name **backpropagation learning algorithm** comes from this.
- These error signals are used to adapt the weights of the neurons.
- Usually the training data must be used many times until the backpropagation algorithm converges.
- Using the training samples once is called an **epoch** in learning.

Detailed structure and notation

- Figure 2 shows detailed structure and mathematical notation of a three-layer multilayer perceptron network.
- It appears as Figure 3.4 in Ham's and Kostanic's book.
- Enlarge the figure to distinguish the small-sized notation.
- In the input "layer", there are n_0 nodes ("neurons").

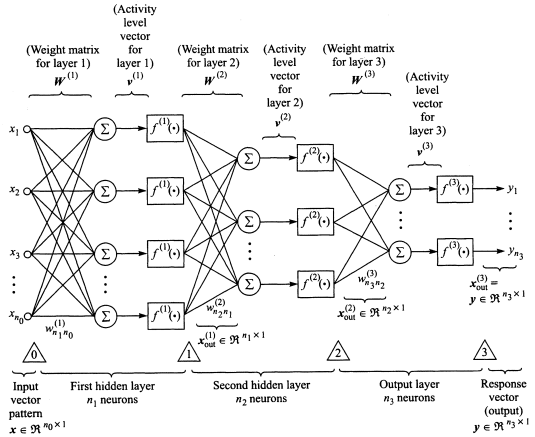

Figure 2: Notation and schematic diagram of a MLP network.

- Each of them feeds to the MLP network one component x_i of the input vector $\mathbf{x} = [x_1, x_2, \ldots, x_{n_0}]^T$.
- In the first hidden layer, there are n_1 neurons.
- The weight matrix $\mathbf{W}^{(1)}$ contains the weights of the connections between the input nodes and the neurons in the first hidden layer.
- The (i,j):th element $w_{ij}^{(1)}$ of the weight matrix $\mathbf{W}^{(1)}$ is the connection weight between neuron j in the input layer and neuron i in the first hidden layer.
- The activity level (linear response) vector of the first hidden layer is denoted by $\mathbf{v}^{(1)}$.
- Its n_1 components contain the linear responses of the neurons of the first hidden layer: $\mathbf{v}^{(1)} = \mathbf{W}^{(1)}\mathbf{x}$.
- The activation function $f^{(1)}(\cdot)$ of the first hidden layer provides its

n_1-dimensional output vector

$$\mathbf{x}_{out}^{(1)} = \mathbf{f}^{(1)}(\mathbf{v}^{(1)}) \tag{1}$$

- The notation $\mathbf{f}^{(1)}(\mathbf{v}^{(1)})$ means that the scalar activation function $f^{(1)}(\cdot)$ is applied componentwise to the vector $\mathbf{v}^{(1)}$.
- The output vector $\mathbf{x}_{out}^{(1)}$ of the first hidden layer is inputted to the n_2 neurons in the second hidden layer.
- The connection weights are collected to the $n_2 \times n_1$ weight matrix $\mathbf{W}^{(2)}$.
- Its element $w_{ij}^{(2)}$ is the connection weight between neuron j in the first hidden layer and neuron i in the second hidden layer.
- Quite similarly as for the first hidden layer, the n_2-dimensional output vector of the second hidden layer is

$$\mathbf{x}_{out}^{(2)} = \mathbf{f}^{(2)}(\mathbf{v}^{(2)}) \tag{2}$$

- There $\mathbf{f}^{(2)}(\cdot)$ is a vector of activation functions of the second hidden layer, defined similarly as for the first hidden layer.
- And $\mathbf{v}^{(2)}$ its n_2-dimensional linear response vector $\mathbf{v}^{(2)} = \mathbf{W}^{(2)}\mathbf{x}_{out}^{(1)}$.
- Finally, $\mathbf{x}_{out}^{(2)}$ is inputted to the n_3 neurons in the output layer.
- The connection weights are elements of the $n_3 \times n_2$ weight matrix $\mathbf{W}^{(3)}$.
- The final n_3-dimensional output vector of the MLP network in Figure 2 is

$$\mathbf{y} = \mathbf{x}_{out}^{(3)} = \mathbf{f}^{(3)}(\mathbf{v}^{(3)}) \tag{3}$$

where $\mathbf{f}^{(3)}(\cdot)$ is the activation vector of the output layer, and $\mathbf{v}^{(3)}$ its linear response vector, $\mathbf{v}^{(3)} = \mathbf{W}^{(3)}\mathbf{x}_{out}^{(2)}$

- Finally, the overall nonlinear input-output mapping $\mathbf{\Omega}$ of the MLP

network in Figure 2 is

$$\mathbf{y} = \mathbf{\Omega}[\mathbf{x}] = \mathbf{f}^{(3)}[\mathbf{W}^{(3)}\mathbf{f}^{(2)}[\mathbf{W}^{(2)}\mathbf{f}^{(1)}[\mathbf{W}^{(1)}\mathbf{x}]]] \tag{4}$$

- Bias terms are included in all the neurons in the computation layers.
- Usually the same activation function $\mathbf{f}(\cdot)$ is used in all the layers.

Backpropagation learning algorithms

- Feedforward multilayer perceptron network is the most widely used neural network structure today.
- When trained properly, it yields good results in many difficult nonlinear learning problems.
- Originally, MLPs were trained by the standard backpropagation algorithm which has been re-invented several times.
- It became popular after the work by Rumelhart, Hinton, and

Williams in 1986.

- The basic backpropagation algorithm is an instantaneous stochastic gradient algorithm.
- It converges slowly, requiring long learning times.
- Therefore nowodays most people use more efficient versions of it.
- We shall first consider the basic backpropagation algorithm, its properties, and some variants of it.
- And on the next lecture faster converging alternatives to it.

Basic backpropagation algorithm for MLP

- The derivation of the basic backpropagation algorithm is presented in many textbook.
- But it is fairly involved, and we skip it in our course.

- In most textbooks, it is represented for the MLP network of Figure 2 having two hidden layers.
- This is the most widely used MLP network architecture.
- However, it is quite easy to extend the resulting algorithm for a MLP network with any number of hidden layers.
- The notation follows Figure 2, which is reproduced here for convenience.
- In principle, the backpropagation algorithm tries to minimize the **mean-square error**

$$J_{MSE} = \frac{1}{2}\mathsf{E}\{\| \mathbf{d} - \mathbf{y} \|^2\} = \frac{1}{2}\mathsf{E}\{\| \mathbf{d} - \mathbf{x}_{out}^{(3)} \|^2\} \qquad (5)$$

- Here $\mathbf{d}$ is the desired response and $\mathbf{y} = \mathbf{x}_{out}^{(3)}$ the corresponding output of the MLP network in Figure 3.

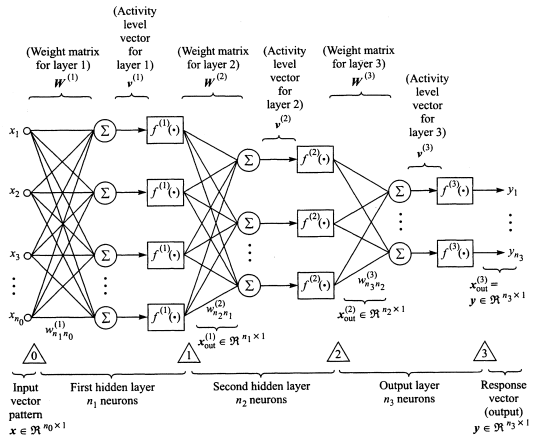

Figure 3: The three-layer MLP network used in deriving the standard backpropagation algorithm.

- Quite similarly as in the LMS algorithm, the mean-square error (5) is approximated on each iteration by the **instantaneous error**

$$E_q = \frac{1}{2}\{\| \mathbf{d}_q - \mathbf{x}_{out}^{(3)} \|^2\} = \frac{1}{2}\sum_{h=1}^{n_3}(d_{qh} - x_{out,h}^{(3)})^2 \qquad (6)$$

- There the subscript q denotes the q:th training sample, and $\mathbf{y}_q = \mathbf{x}_{out}^{(3)}$ is the actual output of the MLP network for it.
- The single q:th training pair $(\mathbf{x}_q, \mathbf{d}_q)$ approximation (6) of the MSE is very coarse.
- But when sufficient number q of training pairs has been used by the MLP network, the combined effect of the instantaneous errors (6) approaches the MSE (5).

- We use the steepest descent approach for the instantaneous error (6).
- The learning rule for a weight $w_{ji}^{(s)}$ in any layer $s = 1, 2, 3$ of the MLP network is given by

$$\Delta w_{ji}^{(s)} = -\mu^{(s)} \frac{\partial E_q}{\partial w_{ji}^{(s)}} \tag{7}$$

  where $\mu^{(s)} > 0$ is the corresponding learning parameter.
- The learning rule has somewhat simpler form for the output layer than for the hidden layers.
- For the **output layer,** one can derive the **local error** or delta term

$$\delta_j^{(3)} = -\frac{\partial E_q}{\partial v_j^{(3)}} = (d_{qj} - x_{out,j}^{(3)}) g(v_j^{(3)}) \tag{8}$$

- There $g(\cdot)$ is the first derivative of the nonlinear activation function $f(\cdot)$.
- The learning rule for the weights in the output layer of the MLP network in Figure 3 is in short

$$\Delta w_{ji}^{(3)} = \mu^{(3)} \delta_j^{(3)} x_{out,i}^{(2)} \tag{9}$$

  or written out more completely

$$w_{ji}^{(3)}(k+1) = w_{ji}^{(3)}(k) + \mu^{(3)} \delta_j^{(3)} x_{out,i}^{(2)} \tag{10}$$

- In general, for the **hidden layers** the local error is

$$\delta_j^{(s)} = (\sum_{h=1}^{n_{s+1}} \delta_h^{(s+1)} w_{hj}^{(s+1)}) g(v_j^{(s)}) \tag{11}$$

and the update rule for the weights in the hidden layer is

$$w_{ji}^{(s)}(k+1) = w_{ji}^{(s)}(k) + \mu^{(s)} \delta_j^{(s)} x_{out,i}^{(s-1)} \tag{12}$$

- The name backpropagation comes from the formula (11).
- The local errors $\delta_h^{(s+1)}$ of the next layer $s+1$ are propagated backwards to the previous layer s for computing its local errors.
- The backpropagation takes place layer-by-layer.
- In the following, we summarize the standard backpropagation algorithm for training a multilayer perceptron network.

Summary of the standard backpropagation algorithm

1. Initialize the weights of the MLP network to small random values.
2. Select an input vector from the available training set and calculate the output of the network for it.
3. Compare this output with the corresponding desired response, and calculate all the local errors using (8) for the output layer and (11) for the hidden layers.
4. Update the weights according to (12) for the hidden layers and to (10) for the output layer.
5. Repeat steps 2 through 4 until the backpropagation algorithm has converged to predetermined level of accuracy.

Some practical issues in using standard backpropagation

Initialization of weights

- The weights of the MLP are initially set to small random values.
- If not, learning may be quite slow.
- This is because for larger initial weights, some of the neurons may be in saturated area.
- Assume that the number of neurons in layer i is n_i.
- The weights of the synaptic connections coming to that layer are commonly set to uniformly distributed random numbers in the interval $[-0.5/n_i, 0.5/n_i]$.
- Other initialization methods have been proposed.

Activation function

- The configuration of the multilayer perceptron (MLP) neural network is determined by:
  - The number of hidden layers;
  - The number of neurons in each layer;
  - The type of activation function(s) used.
- It has been shown that the performance of the MLP network does not depend much on the activation function.
- As long as it is nonlinear.
- The sigmoidal activation functions $f(v) = \tanh(\alpha v)$ and $f(v) = 1/(1 + e^{-\alpha v})$ are used routinely.
- They provide good results, and keep the network always stable.
- Because their values are always limited to the intervals $(-1, +1)$ or

$(0, 1)$.

- The activation function $f(v) = \tanh(\alpha v)$ is often considered to be preferable because of its symmetricity.

Number of neurons in hidden layer(s)

- An important theoretical result by Hornik et al. states that an MLP neural network with one hidden layer only has **universal approximation property.**
- That is, it can approximate smooth nonlinear mappings with any desired degree of accuracy.
- Provided that there are enough neurons in the hidden layer.
- It practice, it is very difficult to determine a sufficient number of neurons to achieve the desired accuracy.
- Frequently, this is done by trial and error.

Number of hidden layers

- If the MLP network has only one hidden layer, its neurons seem to "interact" with each other.
- That is, when the approximation is improved for one point in the mapping it often degrades at some other point.
- For this reason, MLP neural networks are commonly designed with two hidden layers.
- The first hidden layer is responsible for extracting local features.
- While the second hidden layer extracts global features.

Generalization and overfitting

- To solve a real-world problem, a relatively large MLP network architecture must usually be trained.
- A large number of units in the hidden layers guarantees good

performance for the training set.

- However, an overdesigned architecture tends to **overfit** the training data.
- This results in a loss of the generalization property of the network.
- That is, the error for the unseen test data grows when the network has too many neurons.
- We shall consider these issues in a more general setting later on.

Independent validation

- A separate test set is needed for assessing the generalization ability of the learned network.
- In independent validation, the available data is randomly divided into a training and a test set.
- Furthermore, the training data is split into two partitions.

- The first one is used for updating the weight of the network.
- The second partition is used to assess the training performance.
- That is, to decide when to stop training.
- The test data is then used to assess how well the network has generalized.

Speed of convergence

- The backpropagation algorithm is a generalization of the LMS algorithm discussed briefly in Lecture 2.
- The convergence properties of the LMS algorithm (speed and accuracy) depend critically on choice of the learning parameter.
- The considerations presented there carry largely over to the backpropagation algorithm, too.
- The learning parameter must be set to a relatively small value.

- To guarantee convergence of the MLP network and to avoid oscillations during the training.
- But this leads to a relatively slow convergence.
- If some neurons operate in their saturation region, the network may stay long on a flat plateau of the error surface until convergence.
- In the following, we shall first consider some heuristic improvements to the standard backpropagation algorithm.
- They are very much case-specific, and it is not easy to establish their performance characteristics.
- On Lecture 5 we consider faster converging Levenberg-Marquardt algorithm which has much higher computational load per iteration.
- And Extreme Learning Machine which uses linear least-squares method with no iterations.

Backpropagation with momentum updating

- Backpropagation learning algorithm with momentum updating is one of the most popular modifications to the standard backpropagation (BP) algorithm.
- The weights are updated according to the formula

$$\Delta w_{ji}^{(s)}(k+1) = \mu^{(s)} \delta_j^{(s)}(k) x_{out,i}^{(s-1)}(k) + \alpha \Delta w_{ji}^{(s)}(k-1) \qquad (13)$$

- Hence the update (13) is a **linear combination of two terms:**
  - The current gradient $\delta_j^{(s)}(k) x_{out,i}^{(s-1)}(k)$ of the error;
  - The previous weight update $\Delta w_{ji}^{(s)}(k-1)$.
- The **forgetting factor** $\alpha \in (0,1)$ determines how much the momentum term $\alpha \Delta w_{ji}^{(s)}(k-1)$ affects learning.
- The momentum term improves the convergence speed of the standard BP algorithm by stabilizing changes in the weights.

- Intuitively, if the weight are changed in the same direction as in the previous step, their rate of change is increased.
- If these directions are different, the rate of change is decreased.
- The momentum term significantly improves convergence of the BP algorithm in some important situations which are handled poorly by the standard BP algorithm.

Batch updating

- In the standard backpropagation algorithm, weights are updated for every input-output training pair.
- In batch updating, the weight corrections are accumulated over several training patterns before performing the update.
- Possibly over an entire epoch (all the training samples).
- The update is then formed as an average of the corrections for each

individual input-output pair.

- The batch updating corresponds to using **steepest descent method.**
- While standard BP is an **instantaneous stochastic gradient method.**
- The main **advantages** of the batch-updating approach are:
  - Using several or possibly all training pairs gives a much better estimate of the error surface than standard instantaneous BP.
  - The batch approach is suitable for more sophisticated optimization methods such as the conjugate gradient or Newton's method.
- The main **drawbacks** of the batch-updating approach are:
  - Its computational load and memory requirements are higher.
  - It is more prone to converging to a local minimum.

- In general, the performance of the batch updating BP algorithm is very case dependent.
- A simple heuristic strategy to increase the convergence speed of the batch update BP algorithm is to use **variable learning rate.**
- If the previous step has decreased the total error function, the learning rate is increased typically by 5%.
- If the error function has increased by more than a few percent, the learning rate is decreased typically by a factor 0.7.
- Otherwise, no changes are made to the learning rate.

Search-then-converge method

- The search-then-converge method is a fairly simple heuristic strategy for speeding up backpropagation learning.
- According to this strategy, backpropagation learning can be divided

into two phases:

1. In the first **search phase,** the MLP network is relatively far from the global minimum.
   - The learning rate is kept sufficiently large and relatively constant to allow the BP algorithm to get fairly close to the global minimum.
2. In the second **converge phase,** the learning rate is decreased at each iteration.
   - This allows the network to perform fine-tuning of the weight in the vicinity of the global minimum.

- In practice, it is impossible to tell how far the MLP network is from the global minimum.
- Therefore, the strategy for the learning rate decrease must be decided before training.

- A common choice for the learning rate parameter is

$$\mu(k) = \frac{\mu_0}{1 + k/k_0} \tag{14}$$

- There μ_0 is the initial learning rate parameter.
- The constant $100 < k_0 < 500$ defines how fast the learning rate decreases.
- When $k \ll k_0$, the learning rate is nearly constant and close to μ_0, corresponding to the search phase.
- When $k \gg k_0$, the learning rate $\mu(k)$ in (14) decreases in proportion to $1/k$, corresponding to the converge phase.

Computer experiment: two overlapping Gaussians

The classification problem

- This computer experiment is taken from Section 4.8 of the previously used textbook S. Haykin, "Neural Networks: A Comprehensive Foundation", 2nd ed., Prentice-Hall 1998.
- The two-dimensional data vectors belong to two overlapping classes $\mathcal{C}_1$ and $\mathcal{C}_2$.
- The goal is to train an MLP network using the backpropagation algorithm to classify new data vectors as well as possible.
- The probability densities of both the classes are Gaussian:

$$f(\mathbf{x} \mid \mathcal{C}_i) = \frac{1}{2\pi\sigma_i^2} \exp(\frac{1}{2\sigma_i^2} \parallel \mathbf{x} - \mathbf{m}_i \parallel^2) \tag{15}$$

- For the first class $\mathcal{C}_1$, the mean vector is $\mathbf{m}_1 = [0, 0]^T$ and variance $\sigma_1^2 = 1$.
- For the second class $\mathcal{C}_2$, the mean vector is $\mathbf{m}_2 = [2, 0]^T$ and variance $\sigma_2^2 = 4$.
- These probability densities are depicted in Figure 4.
- The classes are equiprobable: a data vector can belong to either of the classes with probability $p = 1/2$.
- Figure 5 shows samples of data vectors belonging to the classes $\mathcal{C}_1$ and $\mathcal{C}_2$.
- Note different scales of the vertical and horizontal axes.
- Because the classes are overlapping (see the bottom subfigure in Figure 5), some classification error is inevitable.

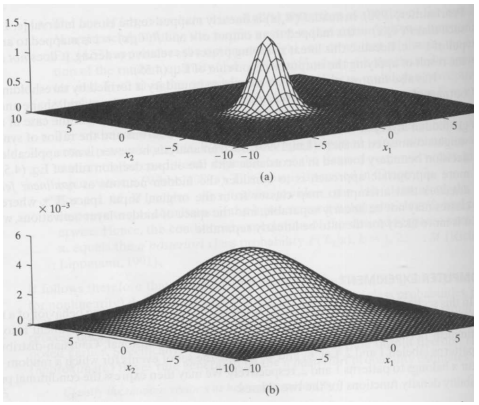

Figure 4: The Gaussian probability density functions of class $\mathcal{C}_1$ (upper figure a) and $\mathcal{C}_2$ (lower figure b).

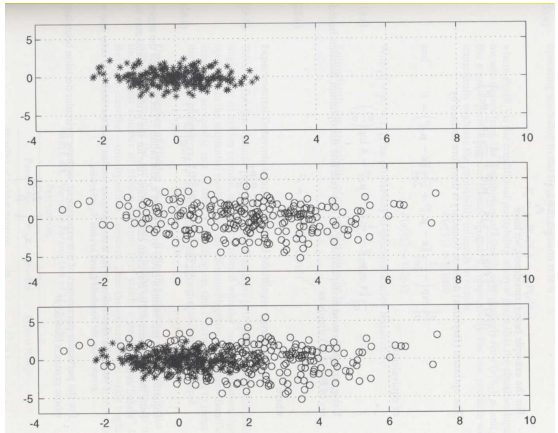

Figure 5: Data vectors of class $\mathcal{C}_1$ (top), class $\mathcal{C}_2$ (middle), and both the classes (bottom).

- For this simple example, the theoretically optimum classification rate can be computed analytically.
- This optimum Bayes classifier achieves 81.51% accuracy.
- The optimal decision boundary is a circle centered at $\mathbf{x}_c = [-2/3, 0]^T$ and having a radius $r \approx 2.34$.
- The data vectors falling inside this circle should be classified to the class $\mathcal{C}_1$, and those falling outside it to the class $\mathcal{C}_2$.

Experimental determination of optimal multilayer perceptron

- An MLP network with one hidden layer trained by the backpropagation algorithm was used for classification.
- The effect of various design parameters to the classification accuracy was tested experimentally.

- The number of hidden neurons was varied.
- In turned out that using two hidden neurons only is enough.
- The classification accuracy did not improve markedly with more hidden neurons.
- The optimum learning parameter was $\mu = 0.5$, and momentum parameter $\alpha = 0.5$.
- The amount of training data was varied between 500 and 8000 input vectors.
- It turned out that the performance did not improve after 1000 training vectors.
- These 1000 training vectors were used 700 times for learning.
- That is, the number of epochs needed for learning the MLP network sufficiently well was 700.

- 20 independent data sets (realizations) of 1000 training vectors were used for teaching the MLP network with different initializations.
- The average classification accuracy of these 20 MLP networks was 79.70%.
- The best classification rate achieved was 80.43%, and worst 71.59%.
- The decision boundaries of the three best MLP networks are shown in Figure 6.
- And for the three worst performing MLP networks in Figure 7.

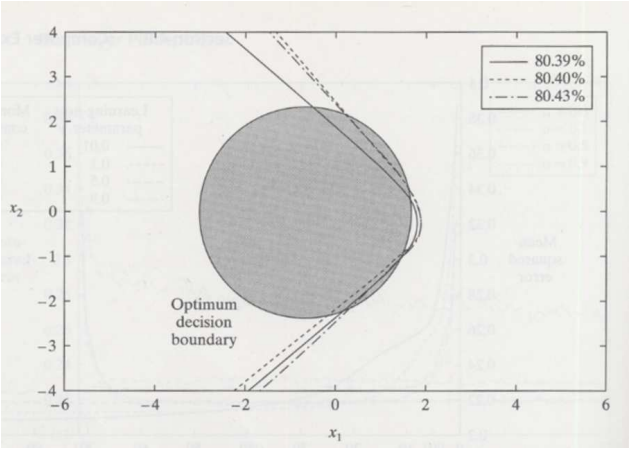

Figure 6: Three best decision boundaries given be the MLP network trained using the backpropagation algorithm.

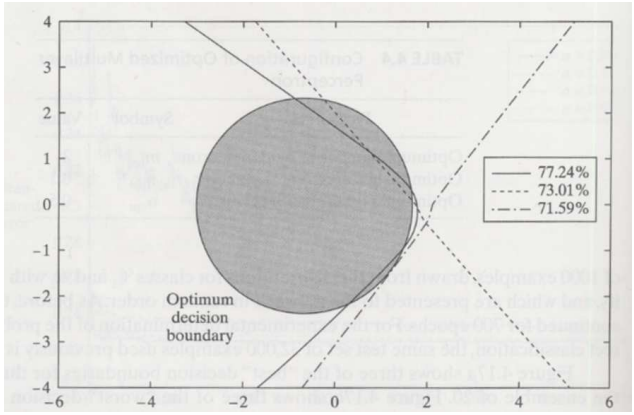

Figure 7: Three worst decision boundaries given be the MLP network trained using the backpropagation algorithm.