第二讲（上）：
多层神经网络和反向传播算法

吕宝粮
计算机科学与工程系
电信楼群3号楼431
Tel: 021-3420-5422

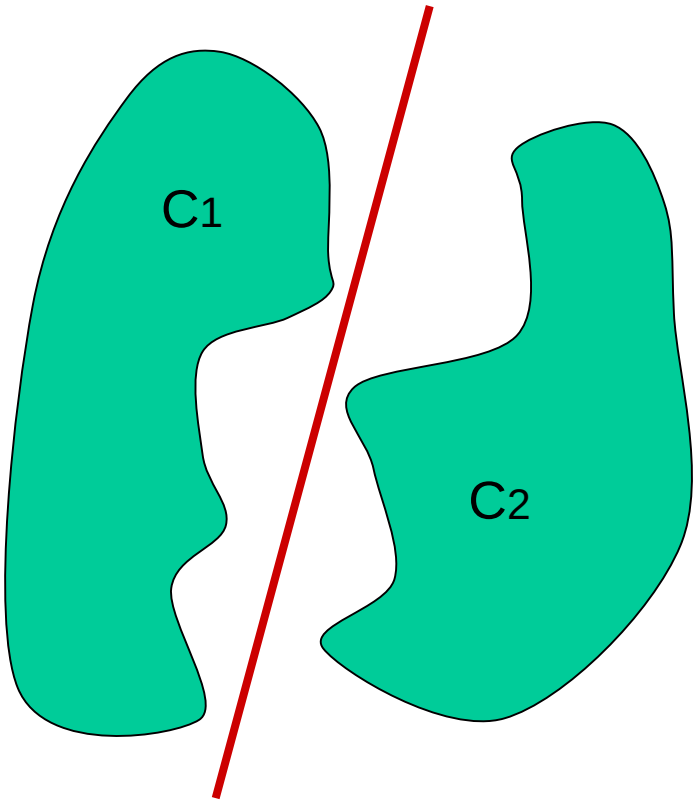
bllu@sjtu.edu.cn
http://bcmi.sjtu.edu.cn/~blu

第二讲内容回顾

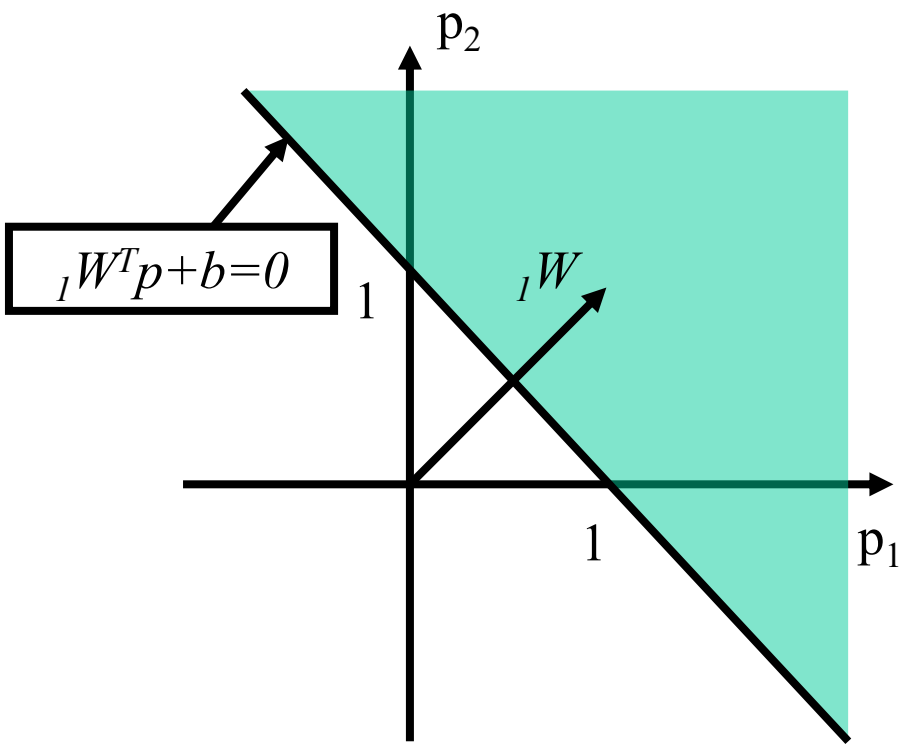
- 感知器
- 感知器学习规则
- 梯度下降法则
- Delta学习规则

线性可分与超平面

线性可分(Linear Separable)



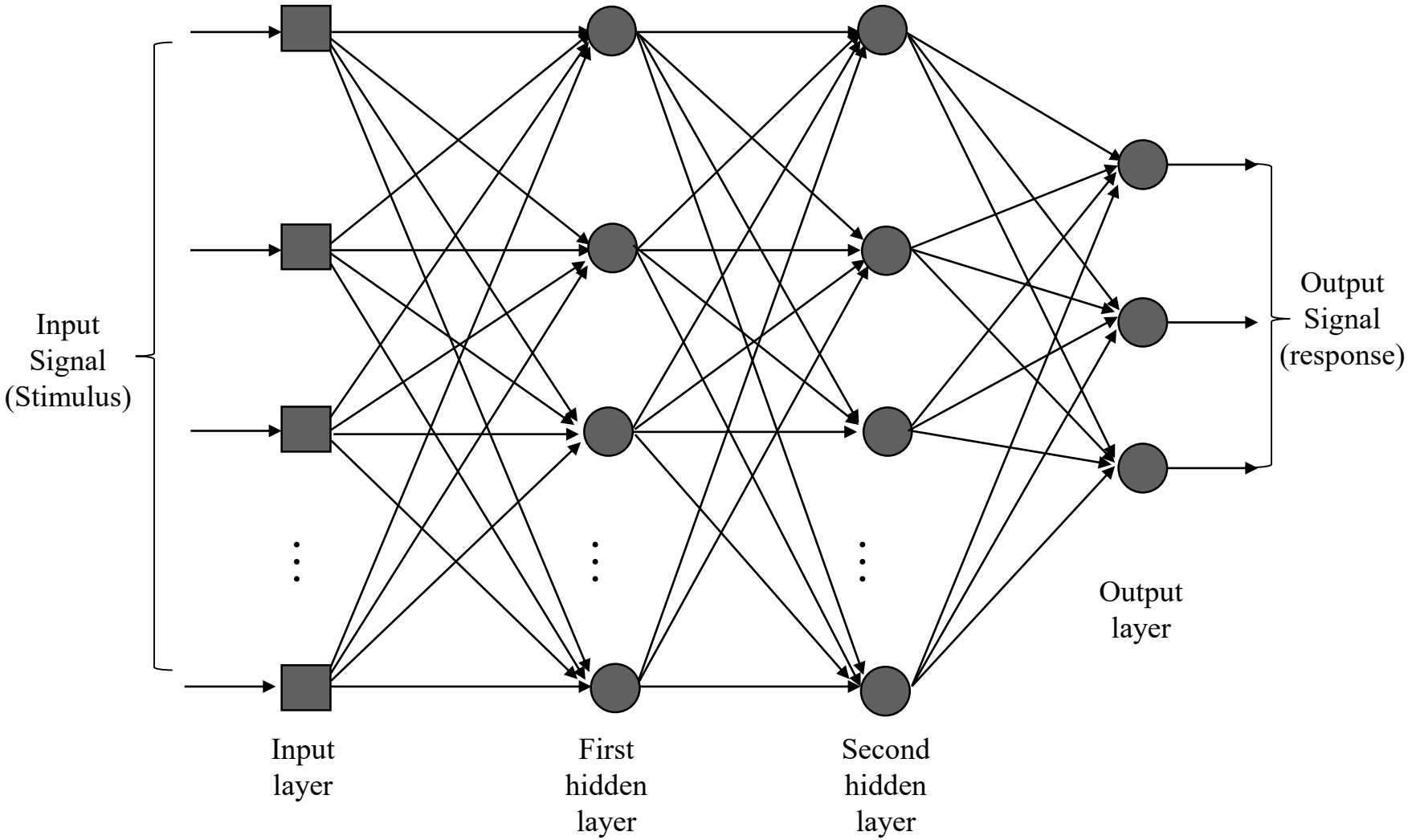
$w_{1,1} = 1 \qquad w_{1,2} = 1 \qquad b = -1$



第二讲（上）主要内容

- 多层感知机
- 可微的阈值函数：Sigmoid单元
- 反向传播(Backpropagation)算法
- BP算法的说明

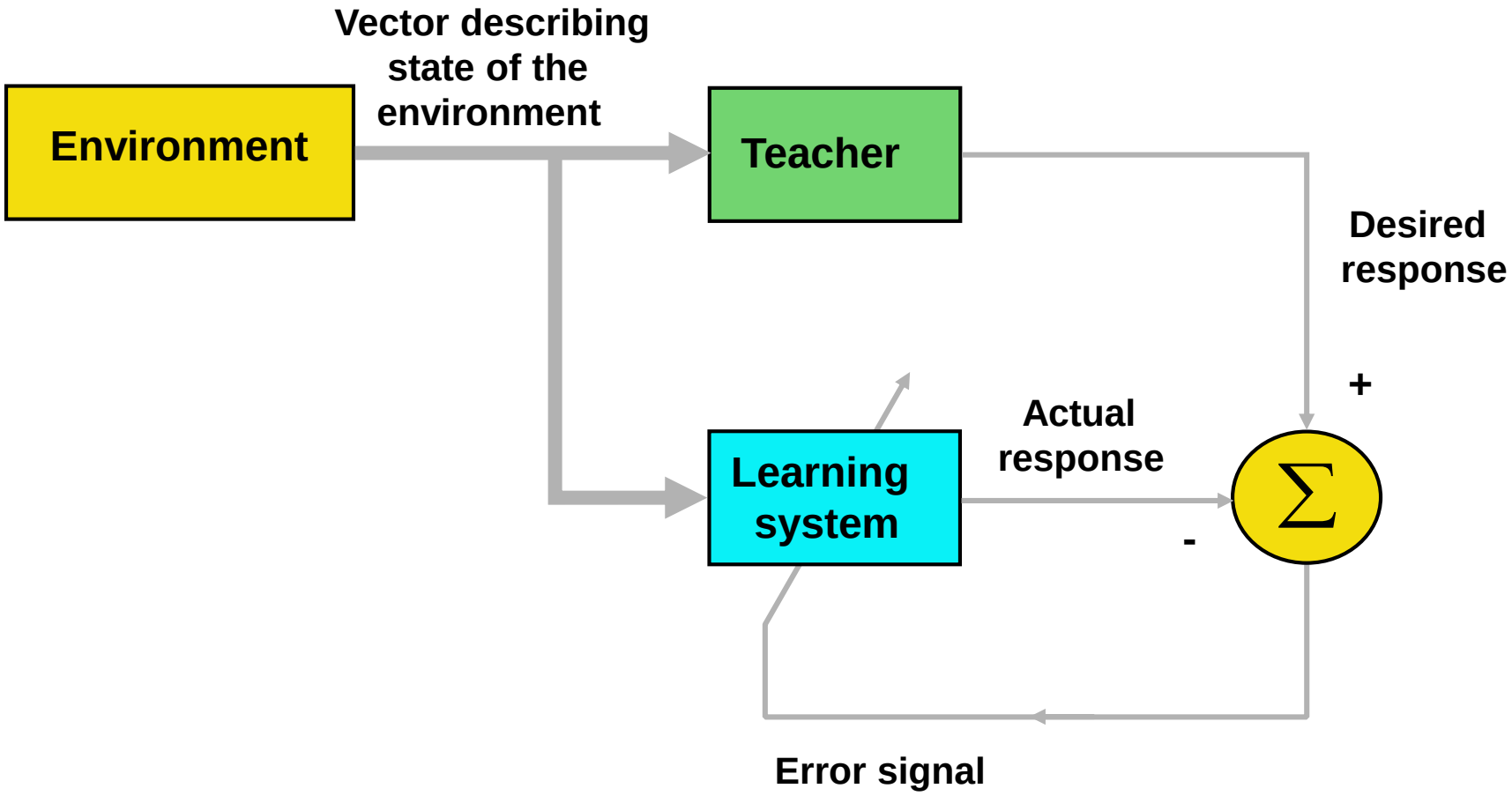
Multilayer Perceptron with Two Hidden Layers



Multilayer Perceptrons

- A multilayer feedforward network consists of an input layer, one or more hidden layers, and an output layer.
- Computations take place in the hidden and output layers only.
- The input signal propagates through the network in a forward direction, layer-by-layer.
- Such neural networks are called *multilayer perceptrons* (MLPs).
- They have been successfully applied to many difficult and diverse problems.
- Multilayer perceptrons are typically trained using so-called error *back-propagation algorithm.*
- This is a supervised error-correction learning algorithm.

Error-Corrective Learning



Multilayer Perceptrons -2

- It can be viewed as a generalization of the LMS algorithm.
- Back-propagation learning consists of two passes through the different layers of a MLP network.
- In the *forward pass*, the output (response) of the network to an input vector is computed.
- Here all the synaptic weights are kept fixed.
- During the *backward pass*, the weights are adjusted using an error-correction rule.
- The error signal is propagated backward through the network.
- After adjustment, the output of the network should have moved closer to the desired response in a statistical sense.

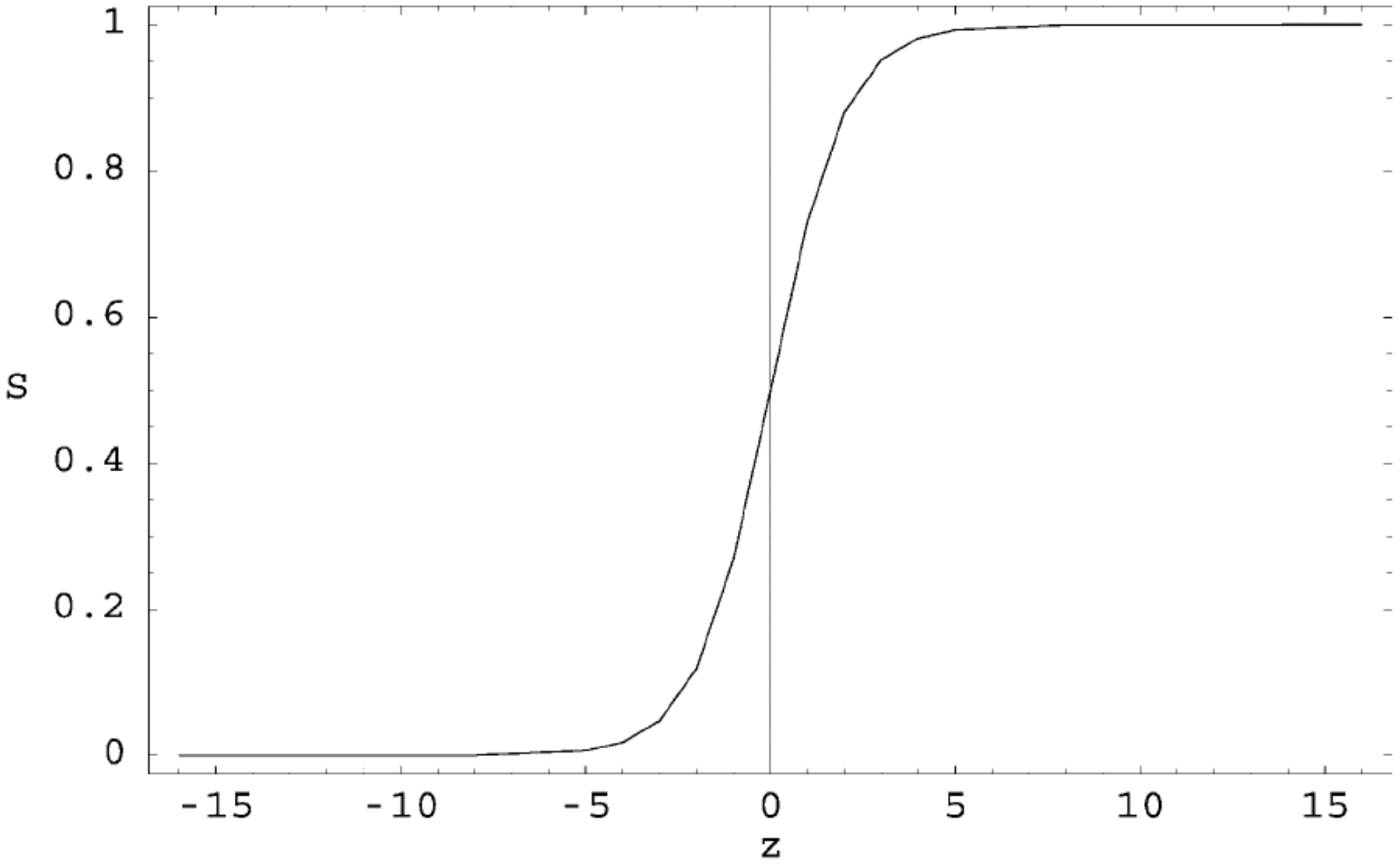
Properties of MLP

- Each neuron has a *smooth* (differentiable everywhere) *nonlinear activation function*.
- This is usually a sigmoidal nonlinearity defined by the *logistic function*

$$y_j = \frac{1}{1 + \exp(-v_j)}$$

  where v_j is the local field (weighted sum of inputs plus bias).
- Nonlinearities are important: otherwise the network could be reduced to a linear single-layer perceptron.

Sigmoidal Activation Function



Properties of MLP -2

- The network contains hidden layer(s), enabling learning complicated tasks and mappings.
- The network has a high connectivity.
- These properties give the multilayer perceptron its computational power.
- On the other hand, distributed nonlinearities make the theoretical analysis of a MLP network difficult.
- Back-propagation learning is more difficult and in its basic form slow because of the hidden layer(s).

Some Preliminaries

- Figure 4.1 shows an architectural graph of a multilayer perceptron with two hidden layers and an output layer.
- Recall that in the input layer, no computations take place; the input vector is only fed in componentwise.
- The network is *fully connected*.
- Two kinds of signals appear in the MLP network

1. *Function Signals.* Input signals propagating forward through the network, producing in the last phase output signals.
2. *Error signals.* Originate at output neurons, and propagate layer by layer backward through the network.

Two Basic Signal Flows

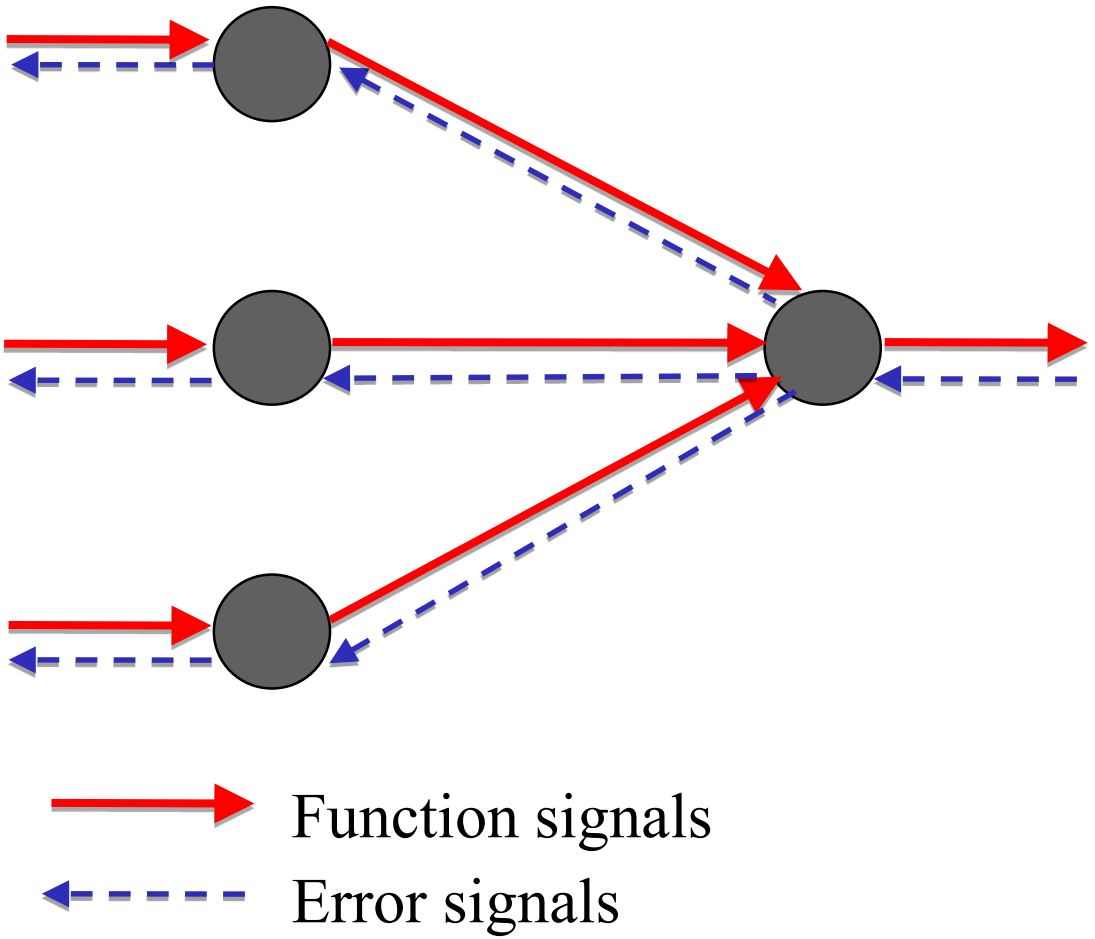


Figure 4.2

Some Preliminaries -2

- Each hidden or output neuron performs two computations:
  1. The computation of the function signal appearing at its output. This is a nonlinear function of the input signal and synaptic weights of that neuron.
  2. The computation of an estimate of the gradient vector, needed in the backward pass.
- The derivation of the back-propagation algorithm is rather involved.
- Before doing that, let us introduce the notation used.

Notation

- The indices i, j and k refer to neurons in different layers. Neuron j lies in the layer right to neuron i, and neuron k right to neuron j.
- In iteration n, the nth training vector is presented to the network.
- The symbol $\mathcal{E}(n)$ refers to the instantaneous sum of error squares or error energy at iteration n.
  - $\mathcal{E}_{av}$ is the average of $\mathcal{E}(n)$ over all n.
- $e_j(n)$ is the error signal at the output of neuron j for iteration n.
- $d_j(n)$ is the desired response for neuron j.
- $y_j(n)$ is the function signal at the output of neuron j for iteration n.

Notation -2

- $w_{ji}(n)$ is the weight connecting the output of neuron i to the input of neuron j at iteration n.
  - The correction applied to this weight is denoted by $\Delta w_{ji}(n)$.
- $v_j(n)$ denotes the local field of neuron j at iteration n.
  - It is the weighted sum of inputs plus bias of that neuron.
- The activation function (nonlinearity) associated with neuron j is denoted by $\varphi_j(.)$.
- b_j denotes the bias applied to neuron j, corresponding to the weight $w_{j0} = b_j$ and a fixed input $+1$.
- $x_i(n)$ denotes the ith element of the input vector.

Notation -3

- $o_k(n)$ denotes the kth element of the overall output vector.
- η denotes the learning-rate parameter.
- m_l denotes the number of neurons in layer l.
  - The network has L layers.
  - For output layer, the notation $m_L = M$ is also used.

Signal-flow graph of output neuron *k*

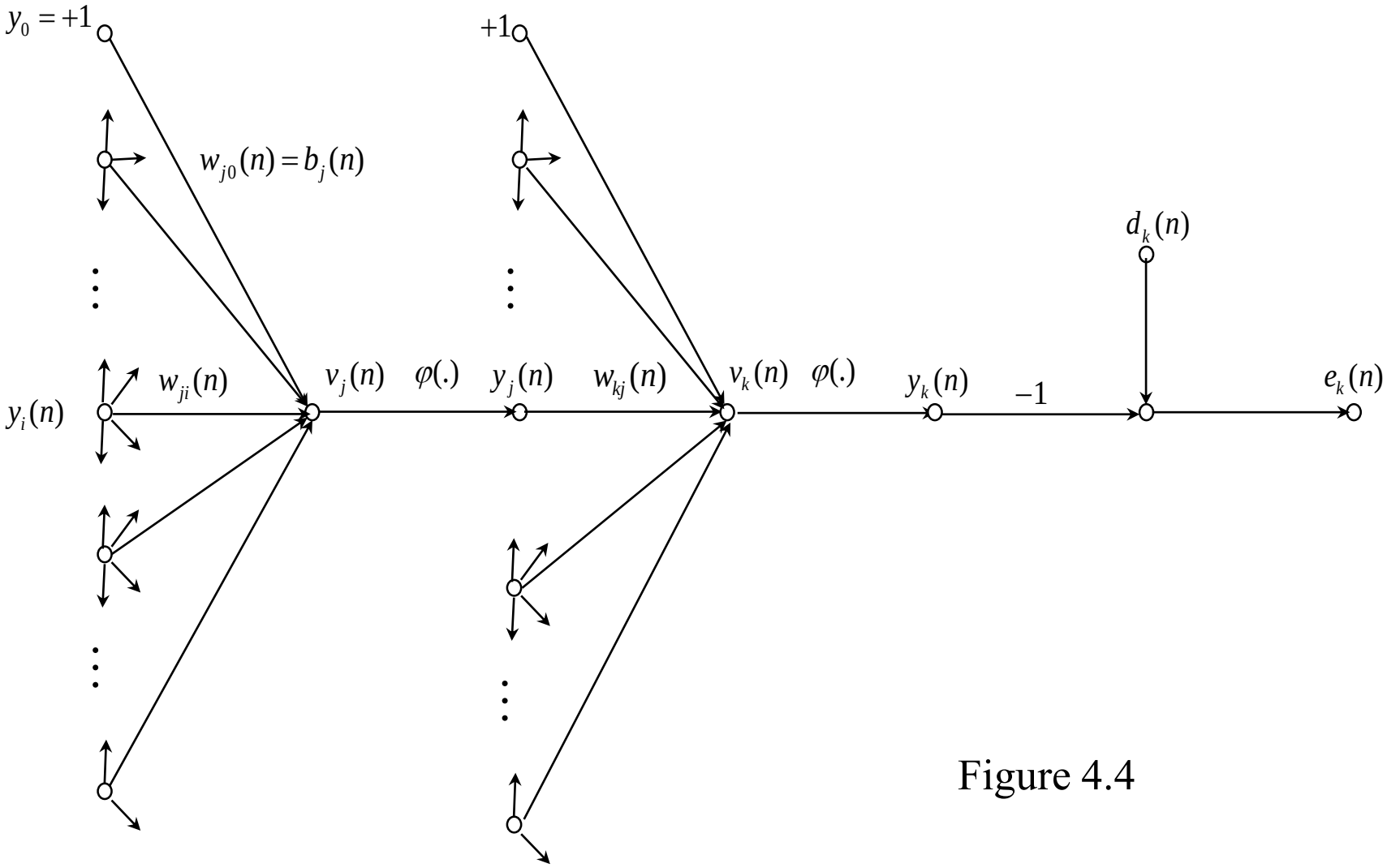


Figure 4.4

Back-Propagation Algorithm

- The error signal at the output of neuron j at iteration n is defined by

$$e_j(n) = d_j(n) - y_j(n), \text{ neuron j is an output node} \qquad (1)$$

- The instantaneous value of the error energy for neuron j is defined by $e_j^2(n)/2$.
- The total instantaneous error energy $\mathcal{E}(n)$ for all the neurons in the output layer is therefore

$$\mathcal{E}(n) = \frac{1}{2}\sum_{j \in C} e_j^2(n) \qquad (2)$$

where the set C contains all the neurons in the output layer.

Back-Propagation Algorithm -2

- Let N be the total number of training vectors (examples, patterns).
- Then the *average squared error energy* is

$$\mathcal{E}_{av} = \frac{1}{N}\sum_{n=1}^{N}\mathcal{E}(n) \tag{3}$$

- For a given training set, $\mathcal{E}_{av}$ is the *cost function* which measures the learning performance.
- It depends on all the free parameters (weights and biases) of the network.
- The objective is to derive a learning algorithm for minimizing $\mathcal{E}_{av}$ with respect to the free parameters.

Back-Propagation Algorithm -3

- In the basic back-propagation, a similar training method as in the LMS algorithm is used.
- Weights are updated on a pattern-by-pattern basis during each epoch.
- *Epoch* is one complete presentation of the entire training set.
- In other words, instantaneous stochastic gradient based on a single sample only is used for getting simple adaptive update formulas.
- The average of these updates over one epoch estimates the gradient of $\mathcal{E}_{av}$.

Signal-flow graph of output neuron *j*

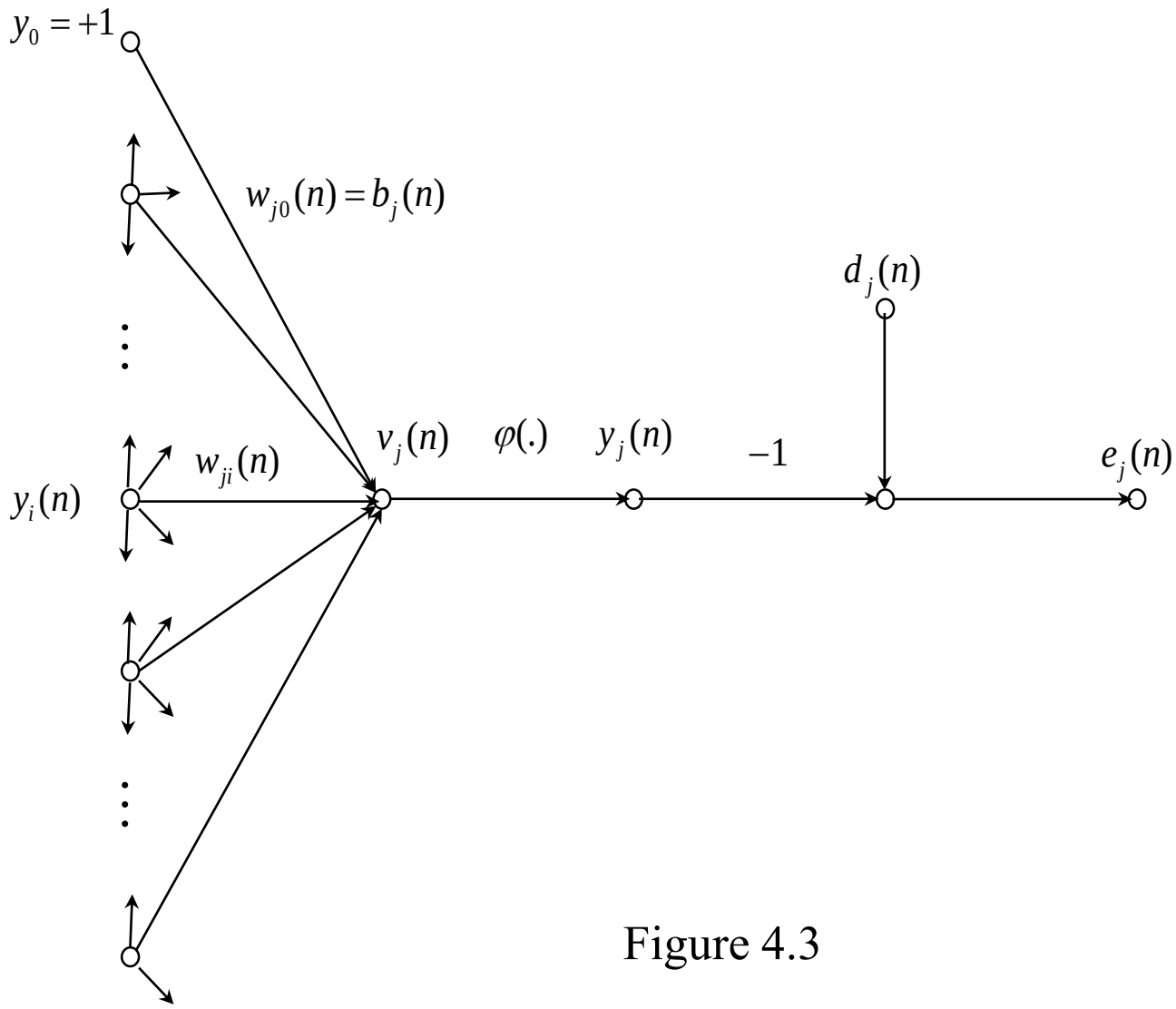


Figure 4.3

Back-Propagation Algorithm -4

- Consider now Figure 4.3 showing neuron j.
- It is fed by a set of function signals produced by a layer of neurons to its left.
- The local field $v_j(n)$ of neuron j is clearly

$$v_j(n) = \sum_{i=0}^{m} w_{ji}(n) y_i(n) \tag{4}$$

- The function signal $y_j(n)$ appearing at the output of neuron j at iteration n is then

$$y_j(n) = \varphi_j(v_j(n)). \tag{5}$$

Back-Propagation Algorithm -5

- The correction $\Delta w_{ji}(n)$ made to the synaptic weight $w_{ji}(n)$ is proportional to the partial derivative $\partial \mathcal{E}(n)/\partial w_{ji}(n)$ of the instantaneous error.
- Using the *chain rule* of calculus, this gradient can be expressed as follows:

$$\frac{\partial \mathcal{E}(n)}{\partial w_{ji}(n)} = \frac{\partial \mathcal{E}(n)}{\partial e_j(n)} \frac{\partial e_j(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)} \frac{\partial v_j(n)}{\partial w_{ji}(n)} \tag{6}$$

- The partial derivative $\partial \mathcal{E}(n)/\partial w_{ji}(n)$ represents a *sensitivity factor*.
- It determines the direction of search for the weight $w_{ji}(n)$.

Back-Propagation Algorithm -6

- Differentiating both sides of Eq. (2) with respect to $e_j(n)$, we get

$$\frac{\partial \mathcal{E}(n)}{\partial e_j(n)} = e_j(n) \tag{7}$$

- Differentiating Eq. (1) with respect to $y_j(n)$ yields

$$\frac{\partial e_j(n)}{\partial y_j(n)} = -1 \tag{8}$$

- Differentiating Eq. (5) with respect to $v_j(n)$, we get

$$\frac{\partial y_j(n)}{\partial v_j(n)} = \varphi_j'(v_j(n)) \tag{9}$$

where φ_j' denotes the derivative of φ_j.

Back-Propagation Algorithm -7

- Finally, differentiating (4) with respect to $w_{ji}(n)$ yields

$$\frac{\partial v_j(n)}{\partial w_{ji}(n)} = y_i(n). \tag{10}$$

- Inserting these partial derivatives into (6) yields

$$\frac{\partial \mathcal{E}(n)}{\partial w_{ji}(n)} = -e_j(n)\varphi_j'(v_j(n))y_i(n) \tag{11}$$

- The correction $\Delta w_{ji}(n)$ applied to the weight $w_{ji}(n)$ is defined by the *delta rule*:

$$\Delta w_{ji}(n) = -\eta \frac{\partial \mathcal{E}(n)}{\partial w_{ji}(n)} \tag{12}$$

where η is the learning-rate parameter of the back-propagation algorithm.

Back-Propagation Algorithm -8

- The minus sign comes from using *gradient descent* in learning for minimizing the error $\mathcal{E}(n)$.
- Inserting (11) into (12) yields

$$\Delta w_{ji}(n) = \eta \delta_j(n) y_i(n) \tag{13}$$

where the *local gradient* is defined by

$$\delta_j(n) = -\frac{\partial \mathcal{E}(n)}{\partial v_j(n)} = e_j(n)\varphi_j'(v_j(n)) \tag{14}$$

- We note that a key factor in the calculation of the weight adjustment $\Delta w_{ji}(n)$ is the error signal $e_j(n)$ at the output of neuron j.
- This error signal depends on the location of the neuron in the MLP network.

Back-Propagation Algorithm -9

Case 1: Neuron j is an Output Node

- Computation of the error $e_j(n)$ is straightforward in this case.
- The desired response $d_j(n)$ for the neuron j is directly available.
- One can use the previous formulas (13) and (14).

Case 2: Neuron j is a Hidden Node

- Now there is no desired response available for neuron j.
- Question: how to compute the responsibility of this neuron for the error made at the output?
- This is the *credit-assignment problem*

Back-Propagation Algorithm -10

- The error signal for a hidden neuron must be determined recursively in terms of the error signals of all neurons connected to it.
- Here the development of the back-propagation algorithm gets complicated.
- Figure 4.4 illustrates the situation where neuron j is a hidden node.
- Using Eq. (14), we may redefine the local gradient $\delta_j(n)$ for hidden neuron j as follows:

$$\delta_j(n) = -\frac{\partial \mathcal{E}(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)} = -\frac{\partial \mathcal{E}(n)}{\partial y_j(n)} \varphi_j'(v_j(n)) \qquad (15)$$

Signal-flow graph of output neuron *k*

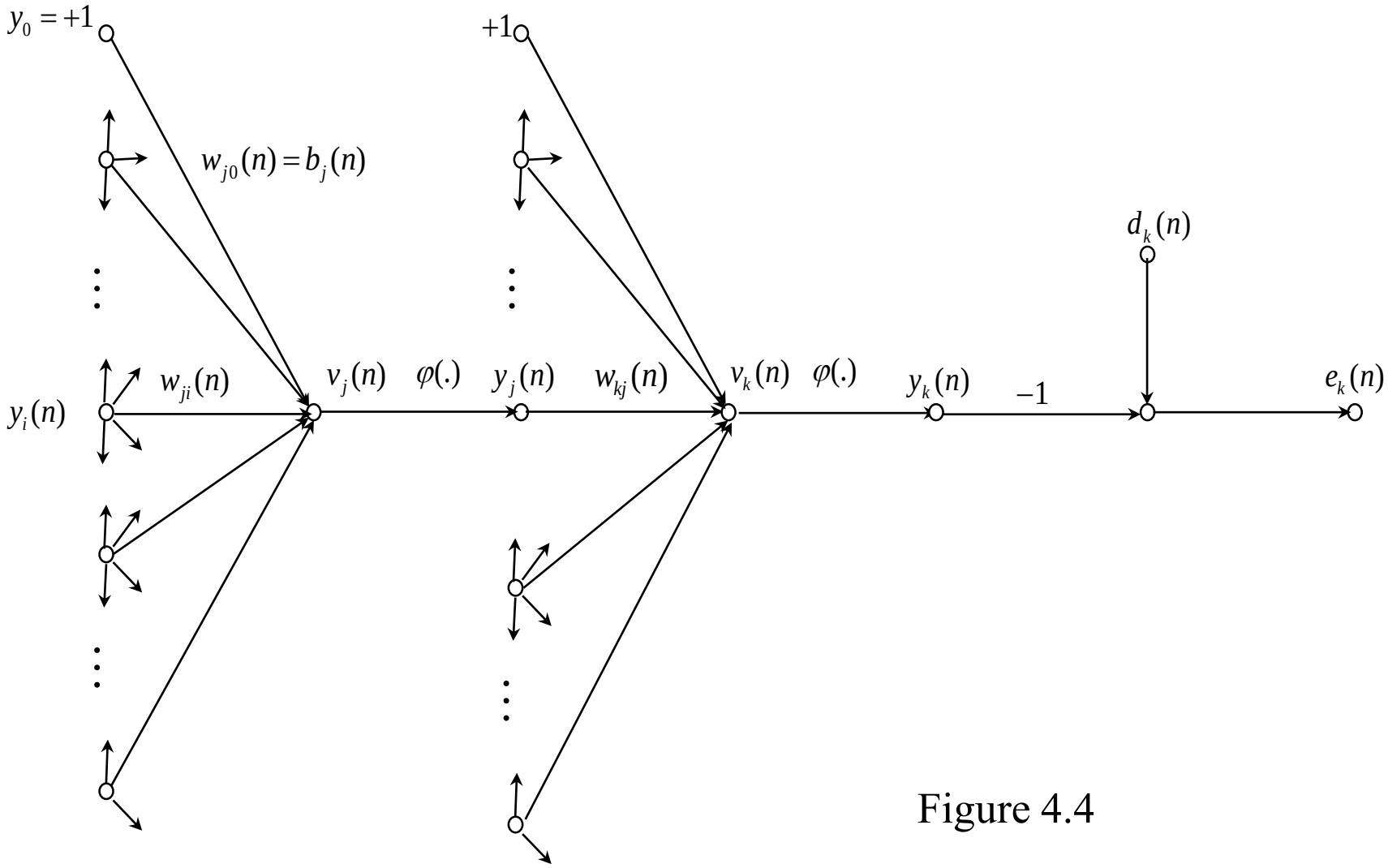


Figure 4.4

Back-Propagation Algorithm -11

- The partial derivative $\partial \mathcal{E}(n)/\partial y_j(n)$ may be calculated as follows.

- From Figure 4.4 we see that

$$\mathcal{E}(n) = \frac{1}{2}\sum_{k\in C} e_k^2(n), \text{ neuron k is an output node} \qquad (16)$$

- Differentiating this with respect to the function signal $y_j(n)$ and using the chain rule we get

$$\frac{\partial \mathcal{E}(n)}{\partial y_j(n)} = \sum_k e_k \frac{\partial e_k(n)}{\partial y_j(n)} = \sum_k e_k \frac{\partial e_k(n)}{\partial v_k(n)} \frac{\partial v_k(n)}{\partial y_j(n)} \qquad (17)$$

Back-Propagation Algorithm -12

- From Figure 4.4 we note that when the neuron k is an output node

$$e_k(n) = d_k(n) - y_k(n) = d_k(n) - \varphi_k(v_k(n)) \qquad (19)$$

so that

$$\frac{\partial e_k(n)}{\partial v_k(n)} = -\varphi_k'(v_k(n)) \qquad (20)$$

- Figure 4.4 shows also that the local field of neuron k is

$$v_k(n) = \sum_{j=0}^{m} w_{kj}(n) y_j(n) \qquad (21)$$

where the bias term is again included as the weight $w_{k0}(n)$.

- Differentiating this with respect to $y_j(n)$ yields

$$\frac{\partial v_k(n)}{\partial y_j(n)} = w_{kj}(n) \qquad (22)$$

Back-Propagation Algorithm -13

- Inserting these expressions into (3) we get the desired partial derivative

$$\frac{\partial \mathcal{E}(n)}{\partial y_j(n)} = -\sum_k e_k(n)\varphi'_k(v_k(n))w_{kj}(n) = -\sum_k \delta_k(n)w_{kj}(n) \quad (23)$$

- Here again $\delta_k(n)$ denotes the local gradient for neuron k.
- Finally, inserting (8) into (1) yields the *back-propagation formula* for the local gradient $\delta_j(n)$:

$$\delta_j(n) = \varphi'_j(v_j(n)) \sum_k \delta_k(n)w_{kj}(n) \quad (24)$$

- This holds when neuron j is hidden.

Back-propagation of local gradient

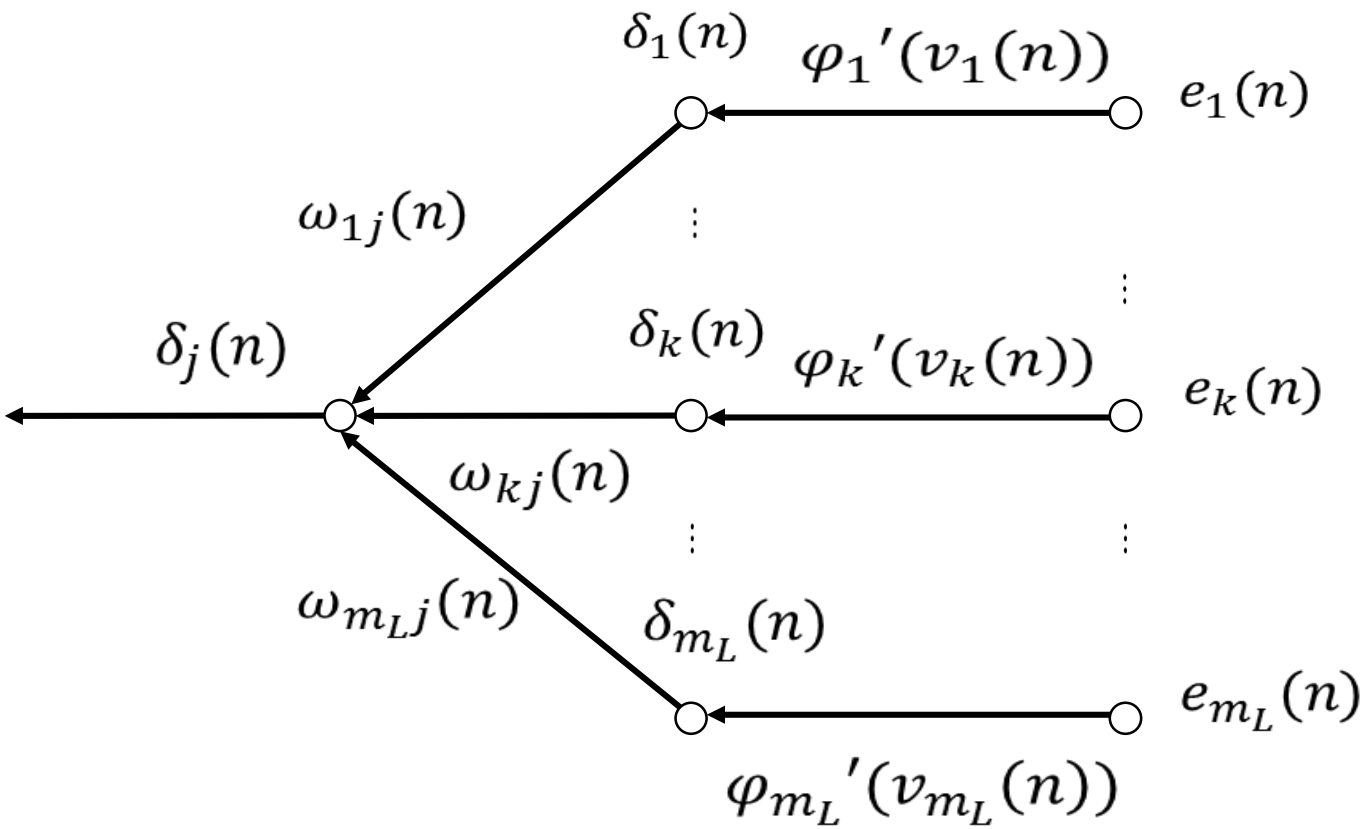


Figure 4.5

Back-Propagation Algorithm -14

- Let us briefly study the factors in this formula:
  - $\varphi_j'(v_j(n))$ depends solely on the activation function $\varphi_j(.)$ of the hidden neuron j.
  - The local gradients $\delta_k(n)$ require knowledge of the error signals $e_k(n)$ of the neurons in the next (right-hand side) layer.
  - The synaptic weights $w_{kj}(n)$ describe the connections of neuron j to the neurons in the next layer to the right.

Back-Propagation Algorithm -15

- We may summarize the results derived thus far in this section as follows:
- The correction $\Delta w_{ji}(n)$ of the weight connecting neuron i to neuron j is described by Eq. (25) (above Figure 4.5).
- The local gradient $\delta_j(n)$ is computed from Eq. (14) if neuron j lies in the output layer.
- If neuron j lies in the hidden layer, the local gradient is computed from Eq. (24)

$$\begin{pmatrix} Weight \\ correction \\ \Delta w_{ji}(n) \end{pmatrix} = \begin{pmatrix} Learning \\ parameter \\ \eta \end{pmatrix} \begin{pmatrix} Local \\ gradient \\ \delta_j(n) \end{pmatrix} \begin{pmatrix} Input\ signal \\ of\ neuron\ j \\ y_i(n) \end{pmatrix} \quad (25)$$

Back-Propagation Algorithm -16

- The local gradient is given by

$$\delta_j(n) = e_j(n)\varphi_j'(v_j(n)) \tag{14}$$

when the neuron j is in the output layer.

- In the hidden layer, the local gradient is

$$\delta_j(n) = \varphi_j'(v_j(n)) \sum_k \delta_k(n) w_{kj}(n) \tag{24}$$

computed recursively from the local gradients of the following layer, back-propagating error

The Two-Passes of Computation

- In applying the back-propagation algorithm, two distinct passes of computation are distinguished.
- **Forward pass**
  - The weights are not changed in this phase.
  - The function signal appearing at the output of neuron j is computed as

$$y_j(n) = \varphi(v_j(n)) \tag{26}$$

  - Here the local field $v_j(n)$ of neuron j is

$$v_j(n) = \sum_{i=0}^{m} w_{ji}(n) y_i(n) \tag{27}$$

The Two-Passes of Computation -2

- In the first hidden layer, $m = m_0$ is the number of input signals $x_i(n)$, $i = 1, \ldots, m_0$, and in (27)

$$y_i(n) = x_i(n)$$

- In the output layer, $m = m_L$ is the number of outputs (26).
- The outputs (components of the output vector) are denoted by

$$y_j(n) = o_j(n)$$

- These outputs are then compared with the respective desired responses $d_j(n)$, yielding the error signals $e_j(n)$.
- In the forward pass, computation starts from the first hidden layer and terminates at the output layer.

The Two-Passes of Computation -3

- **Backward pass**
  - In the backward pass, computation starts at the output layer, and ends at the first hidden layer.
  - The local gradient δ is computed for each neuron by passing the error signal through the network layer by layer.
  - The delta rule of Eq. (25) is used for updating the synaptic weights.
  - The weight updates are computed recursively layer by layer.
- The input vector is fixed through each round-trip (forward pass followed by a backward pass).
- After this, the next training (input) vector is presented to the network.

Learning Rate

- Back-propagation approximates steepest descent method.
- A small learning-rate parameter η leads to a slow learning rate.
- Generally, basic back-propagation suffers from very slow learning if the network is large (several layers, a lot of nodes).
- On the other hand, choosing too large a learning parameter may lead to oscillatory behavior.
- A simple method of improving the learning speed without oscillatory behavior:
- Use a *generalized delta rule* including a momentum term:

$$\Delta w_{ji}(n) = \alpha \Delta w_{ji}(n-1) + \eta \delta_j(n) y_i(n) \tag{39}$$

- Here α is a positive *momentum constant*.
- If $\alpha = 0$, the corresponding momentum term vanishes.

Learning Rate-2

- Then (39) reduces to the standard delta rule derived earlier.
- The effect of the momentum term is analyzed somewhat in Haykin's book on p. 170.
- The conclusions are:
  1. The momentum constant should be in the interval $0 \leq \alpha < 1$.
  2. The momentum term tends to accelerate descent in steady downhill direction.
  3. In directions where the partial derivative $\partial \mathcal{E}(t)/\partial w_{ji}(t)$ oscillates in sign, the momentum term has a stabilizing effect.

Learning Rate-3

- In deriving the back-propagation algorithm, it was assumed that the learning parameter η is a constant.
- In practice, it is better to use a *connection-dependent* learning parameter η_{ij}.
- This will be discussed later on in this chapter.

Sequential and Batch Modes of Training

- Recall that one complete presentation of the entire training set is called an epoch.
- The learning process is continued over several/many epochs.
- Learning is stopped when the weight values and biases stabilize, and the average squared error converges to some minimum value.
- It is useful to present the training samples in a randomized order during each epoch.
- In back-propagation, one may use either sequential (on-line, stochastic) or batch learning mode.

Sequential and Batch Modes of Training -2

1. Sequential Mode

- The weights are updated after presenting each training example (input vector).
- The derivation before was given for this mode.

2. Batch Mode

- Here the weights are updated after each epoch only.
- All the training examples are presented once before updating the weights and biases.
- In batch mode, the cost function is the average squared error

$$\mathcal{E}_{av} = \frac{1}{2N} \sum_{n=1}^{N} \sum_{j \in C} e_j^2(n) \qquad (13)$$

Sequential and Batch Modes of Training -3

- The synaptic weight is updated using the batch delta rule

$$\Delta w_{ji} = -\eta \frac{\partial \mathcal{E}_{av}}{\partial w_{ji}} = -\frac{\eta}{N} \sum_{n=1}^{N} e_j(n) \frac{\partial e_j(n)}{\partial w_{ji}} \qquad (43)$$

- The partial derivative $\partial e_j(n)/\partial w_{ji}$ may be computed as in the sequential mode.
- *Advantages of sequential mode:*

  - requires less storage
  - less likely to get trapped in a local minimum.

- *Advantages of the batch mode:*

  - Provides an accurate estimate of the gradient vector.
  - Convergence to a local minimum at least is guaranteed.

Sequential and Batch Modes of Training -4

- The sequential mode of back-propagation has several disadvantages.
- In spite of that, it is highly popular for two important practical reasons:
  - The algorithm is simple to implement.
  - It provides effective solutions to large and difficult problems.

Stopping Criteria

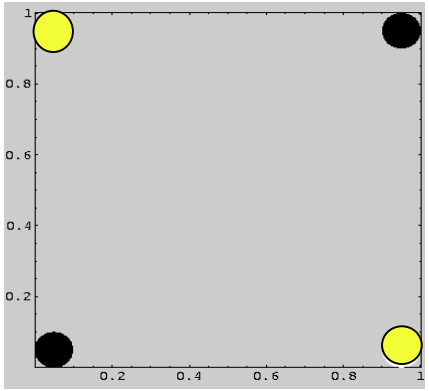
- In general, the back-propagation algorithm cannot be shown to converge.
- There are no well-defined criteria for stopping its operation.
- However, there are reasonable practical criteria for terminating learning.
- Consider first the unique properties of a *local* or *global minimum* of the error surface.
- Denote by $\mathbf{w}^*$ a weight vector at a local or global minimum.
- Necessary condition: the gradient vector $\mathbf{g}(\mathbf{w})$ vanishes at the minimum point $\mathbf{w}^*$.

Stopping Criteria -2

- This yield the following criterion for the convergence of back-propagation learning:
- *Stop learning when the Euclidean norm* $\| \mathbf{g}(\mathbf{w}) \|$ *of the gradient vector is below a certain threshold.*
- Drawbacks of this stopping criterion:
  - Learning times may be long.
  - Requires computation of the gradient vector $\mathbf{g}(\mathbf{w})$.
- Another criterion is based on the stationarity of the average squared error measure $\mathcal{E}_{av}$ at the point $\mathbf{w} = \mathbf{w}^*$:
- *Stop learning when the absolute rate of change in the average squared error per epoch is sufficiently small.*
- A small rate of change is usually taken to be 0.1% - 1% per epoch.

XOR Problem

- The patterns in the first class are (1,1) and (0,0).
- The patterns in the second class are (0,1) and (1,0).
- A single-layer perceptron is not sufficient for solving this problem.
- Reason: the classes are not linearly separable.
- However, the problem may be solved by adding a hidden layer.
- McCulloch-Pitts neuron model (a hard-limiting nonlinearity) is used here.



Heuristics for the BP Algorithm

- Design of a MLP network using back-propagation learning is partly art, not science.
- Numerous heuristic methods have been proposed for improving the learning speed and performance of back-propagation.

1. Sequential versus batch update

- Sequential learning mode is computationally faster than the batch mode.
- This is especially true when the training data set is large and highly redundant.

Heuristics for the BP Algorithm -2

2. Maximizing information content

- Every training example should be chosen so that it contains as much as possible useful information for the learning task.
- Two ways of achieving this aim are:
  - Using an example that results in the largest training error.
  - Using an example that is radically different from the previously used ones.
- The training examples should be presented in randomized order in different epochs.
- A more refined technique is to emphasize diffucult patterns in learning.
- However, this has problems also:
  - Distribution of the training data is distorted.
  - Outliers may have a catastrophic effect on performance.

Heuristics for the BP Algorithm -3

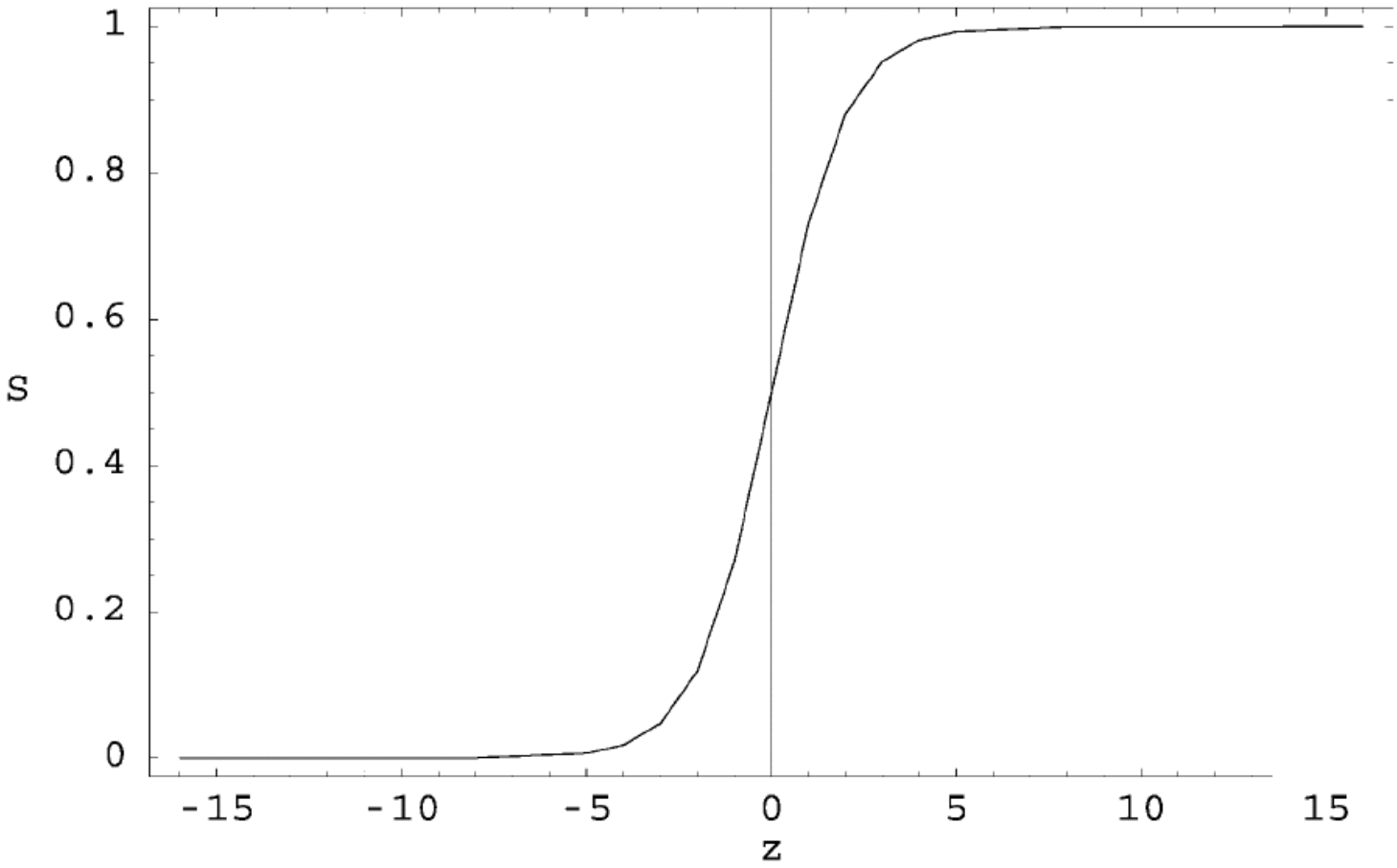
3. Activation function

- A MLP network trained with backpropagation typically learns faster if an antisymmetric sigmoid function is used.
- An activation function $\varphi(v)$ is *antisymmetric (odd)* if $\varphi(-v) = -\varphi(v)$.
- The standard logistic function $a/[1+\exp(-bv)]$ is nonsymmetric, but $\tanh(bv)$ is antisymmetric.
- A good choice for an activation function:

$$\varphi(v) = a\tanh(bv)$$

  where $a = 1.7159$ and $b = 2/3$.
- Then $\varphi(1) = 1$, $\varphi(-1) = -1$, and the first and second derivatives of $\varphi(v)$ have suitable values.

Sigmoidal Activation Function



Heuristics for the BP Algorithm -4

4. Target values

- The target values (desired responses) should be chosen within the range of the sigmoid activation function.
- The desired responses should be somewhat smaller than the extremal (saturation) values of the activation function.
- Otherwise, the back-propagation algorithm tends to drive the free parameters of the networks to infinity.
- This slows down the learning process by driving the hidden neurons into saturation.
- For example, for the activation function $\varphi(v) = 1.716 \tanh(0.667v)$ discussed before, convenient target values are $d_j = \pm 1$ (see Fig. 4.10a).

Heuristics for the BP Algorithm -5

5. Normalizing the inputs

- For speeding up back-propagation learning, the input vectors (variables) should be *preprocessed*.
- Recommended preprocessing steps for the training patterns:
  1. The mean value of the training vectors should be made zero (or small enough).
     - prevents slow, zigzagging type learning.
  2. The input variables (different components of training vectors) should be uncorrelated.
     - Can be realized using Principal Components Analysis (Chapter 8).
     - Removes second-order statistical redundancies.
  3. The decorrelated input variables should be scaled to have approximately the same variances.
     - Ensures that different synaptic weights learn with roughly the same speed.

Normalization of Inputs and Outputs

$$\bar{x}_{ik} = X_{\min} + \frac{x_{ik} - x_{k,\min}}{x_{k,\max} - x_{k,\min}}(X_{\max} - X_{\min}) = \alpha_k + \beta_k x_{ik}, \qquad (4)$$

where $x_{ik} \in \mathbf{R}$ is the kth element of the ith training input, $x_{k,\min} = \min\{x_{ik} | i = 1, 2, \ldots, M\}$, $x_{k,\max} = \max\{x_{ik} | i = 1, 2, \ldots, M\}$, $\bar{x}_{ik}$ is the kth element of the ith normalized training input, $\alpha_k = X^k_{\min} - x_{k,\min}(X_{\max} - X_{\min})/(x_{k,\max} - x_{k,\min})$, and $\beta_k = (X_{\max} - X_{\min})/(x_{k,\max} - x_{k,\min})$.

Heuristics for the BP Algorithm -6

6. Initialization

- Good initial values for the synaptic weights and thresholds (biases) of the network can help tremendously in designing a good network.
- Assume first that the synaptic weights have large initial values.
- Then it is likely that the neurons will be driven into saturation.
- Results in slow learning.
- Assume now that synaptic weights are assigned small initial values.

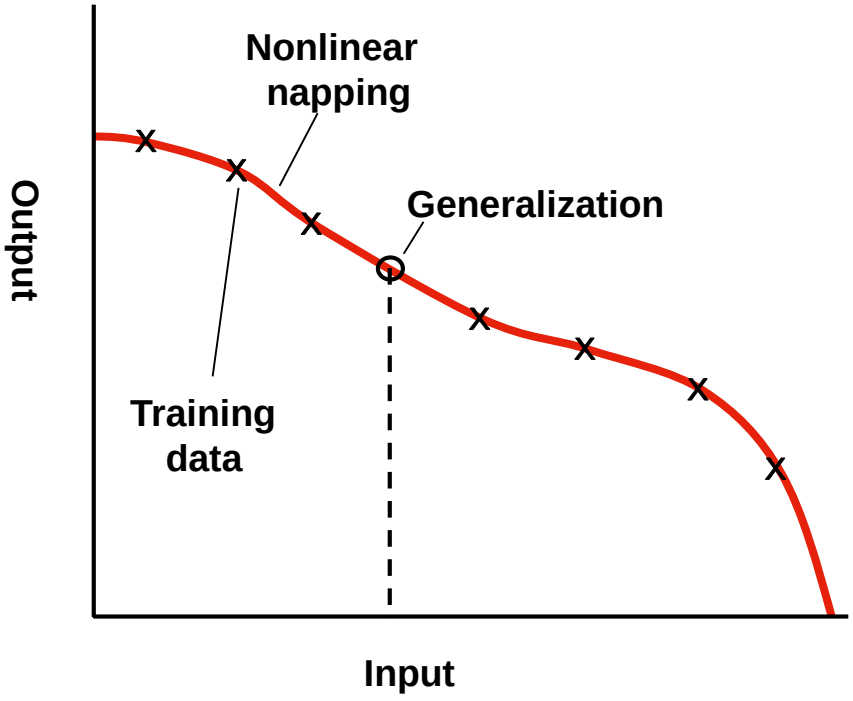
Heuristics for the BP Algorithm -7

- Assume now that:
  - The input variables have zero mean and unit variance.
  - They are mutually uncorrelated.
  - The tanh nonlinearity is used.
  - The thresholds (biases) are set to zero for all neurons.
  - The initial values of the synaptic weights are drawn from a uniform distribution with zero mean and the same variance σ_w^2
- It is then fairly easy to show (see Haykin, pp. 183-184) that:
- For the activation function $\varphi(v) = 1.716 \tanh(0.667v)$ discussed earlier, we should choose $\sigma_w^2 = m^{-1}$.
- Here m is the number of synaptic connections of a neuron.

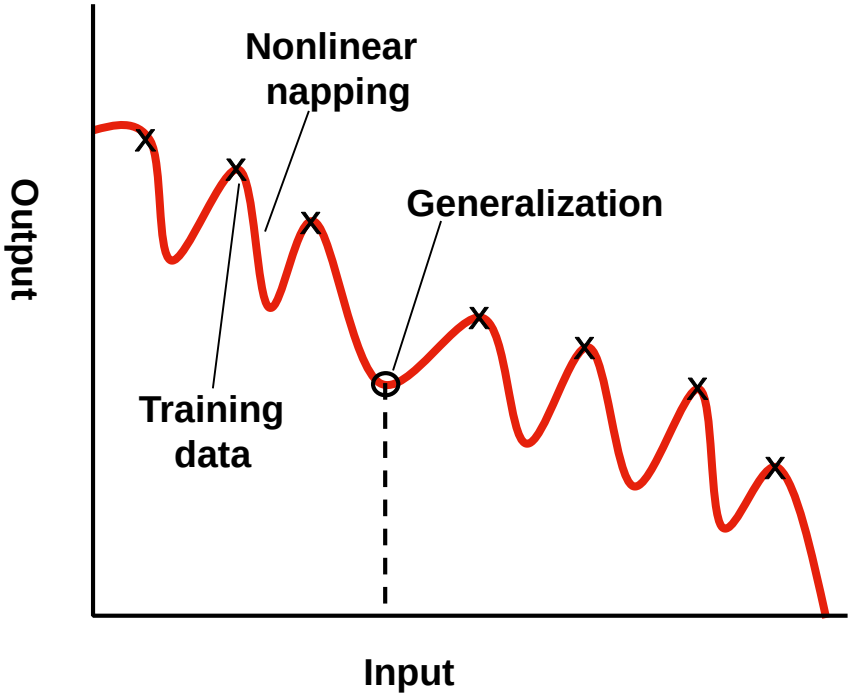
Generalization

- In back-propagation learning, as many training examples as possible are typically used.
- It is hoped that the network so designed generalizes well.
- A network *generalizes* well when its input-output mapping is (almost) correct also for test data.
- The test data is not used in creating or training the network.
- Assumption: test data comes from the same population (distribution) as the training data.
- Training of a neural network may be viewed as a curve fitting (nonlinear mapping) problem.
- The network can simply be considered as a nonlinear input-output mapping.

Good and Poor Generalization



Properly fitted data
(good generalization)



Overfitted data
(poor generalization)

Generalization -2

- Generalization can be studied in terms of the nonlinear interpolation ability of the network.
- MLP networks with continuous activation functions perform useful interpolation because they have continuous outputs.
- Figure 4.19a shows an example of good generalization for a data vector not used in training.
- Using too many training examples may lead to a poor generalization ability.
- This is called *overfitting* or *overtraining*.
- The network then learns even unwanted noise present in the training data.
- More generally, it learns "features" which are present in the training set but actually not in the underlying function to be modeled.

Generalization -3

- Basic reason for overfitting: there are more hidden neurons than necessary in the network.
- Similar phenomena appear in other modeling problems if the chosen model is too complicated, containing too many free parameters.
- For example least-squares fitting, autoregressive modeling, etc.
- *Occam's razor* principle in model selection: select the simplest model which describes the data adequately.
- In the neural network area, this implies choosing the smoothest function that approximates the input-output mapping for a given error criterion.
- Such a choice generally demands the fewest computational resources.

Sufficient Training Data

- Three factors affect generalization:
  1. The size and representativeness of the training set.
  2. The architecture of the neural network.
  3. The physical complexity of the problem at hand.
- Only the first two factors can be controlled.
- The issue of generalization may be viewed from two different perpectives:
  - *The architecture of the network is fixed.* Determine the size of the training set for a good generalization.
  - *The size of the training set is fixed.* Determine the best architecture of network for achieving a good generalization.

Sufficient Training Data -2

- Here we focus on the first viewpoint, hoping that the fixed architecture matches the complexity of the problem.
- Distribution-free, worst-case formulas are available for estimating the size of sufficient training set for a good generalization performance.
- *A practical condition for a good generalization:*
- The size N of the training set must satisfy the condition

$$N = O\left(\frac{W}{\varepsilon}\right)$$

- Here W is the total number of free parameters (weights and biases) in the network.
- ε denotes the fraction of classification errors permitted on test data (as in pattern classification).

Sufficient Training Data -3

- $O(.)$ is the order of quantity enclosed within it.
- Example: If an error of 10% is permitted, the number of training examples should be about 10 times the number of free parameters.

Approximation of Functions

- A MLP network trained with back-propagation is a practical tool for performing a *general nonlinear input-output mapping.*
- Let m_0 be the number of input nodes (neurons), and $M = m_L$ the number of output nodes.
- The input-output mapping of the MLP network is from m_0-dimensional input space to M-dimensional output space.
- If the activation function is infinitely continuously differentiable, the mapping is also.
- A fundamental question: *What is the minimum number of hidden layers in a MLP network providing an approximate realization of any continuous mapping?*

Universal Approximation Theorem

- The *universal approximation theorem* for a nonlinear input-output mapping provides the answer.
- The theorem is presented in Haykin's book, pp. 208-209.
- Its essential contents are as follows:
- *Let* $\varphi(.)$ *be a nonconstant, bounded, and monotonically increasing continuous function.*
- *Let* I_{m_0} *denote the* m_0*-dimensional unit hypercube* $[0,1]^{m_0}$*, and* $C(I_{m_0})$ *the space of continuous functions on* I_{m_0}*.*
- *For any given function* $f \in C(I_{m_0})$ *and* $\varepsilon > 0$*, there exist an integer* M *and sets of real constants* α_i, b_i*, and* w_{ij}*, where* $i = 1, \ldots, m_1$ *and* $j = 1, \ldots, m_0$ *so that:*

Universal Approximation Theorem -2

- *The function*

$$F(x_1, \ldots, x_{m_0}) = \sum_{i=1}^{m_1} \alpha_i \varphi \left(\sum_{j=1}^{m_0} w_{ij} x_j + b_i \right)$$

is an approximate realization of the function $f(.)$.

- *That is,*

$$| F(x_1, \ldots, x_{m_0}) - f(x_1, \ldots, x_{m_0}) | < \varepsilon$$

for all $x_1, x_2, \ldots, x_{m_0}$ *that lie in the input space.*

- The universal approximation theorem is directly applicable to multilayer preceptrons.

Universal Approximation Theorem -3

- The logistic function $\varphi(v) = 1/[1 + \exp(-v)]$ is a nonconstant, bounded, and monotonically increasing function.
- Furthermore, Eq. (4.86) represent the output of a MLP network described as follows:
  1. The network has m_0 input nodes with inputs $x_1, x_2, \ldots, x_{m_0}$, and a single hidden layer consisting of m_1 neurons.
  2. Hidden neuron i has synaptic weights $w_{i1}, \ldots, w_{im_0}$, and bias b_i.
  3. The network output is a linear combination of the outputs of the hidden neurons, with $\alpha_1, \ldots, \alpha_{m_1}$ defining the weights of the output layer.

Universal Approximation Theorem -4

- The universal approximation theorem is an *existence* theorem.
- In effect, the theorem states that *a MLP network with a single hidden layer is sufficient for uniform approximation with accuracy* ε.
- However, the theorem does not say that a single hidden layer is optimal with respect to:
  - learning time
  - ease of implementation
  - generalization ability (most important property).

Bounds on Approximation Errors

- This theoretical treatment is not essential in this course.
- A result worth mentioning: the size of the hidden layer m_1 must be large for getting a good approximation.

Practical Consideration

- The universal approximation theorem is important from a theoretical viewpoint.
- It gives a rigorous mathematical foundation for using multilayer perceptrons in approximating nonlinear mappings.
- However, the theorem is not constructive.
- It does not actually tell how to specify a MLP network with the stated approximation properties.
- Some assumptions made in the theorem are unrealistic in most practical applications:
  - The continuous function to be approximated is given.
  - A hidden layer of unlimited size is available.
- A problem with MLP's using a single hidden layer: the hidden neurons tend to interact globally.

Practical Consideration -2

- In complex situations, improving the approximation at one point typically worsens it at some other point.
- With two hidden layers, the approximation (nonlinear mapping) process becomes more manageable.
- One can proceed as follows:
  1. *Local features* are extracted in the first hidden layer.
     - The input space is divided into regions by some neurons.
     - Other neurons in the first hidden layer learn the local features characterizing these regions.
  2. *Global features* corresponding to each region are extracted in the second hidden layer.
     - Neurons in this layer combine the outputs of the neurons describing certain region.
- This procedure somehow corresponds to piecewise polynomial (spline) approximation in curve fitting.

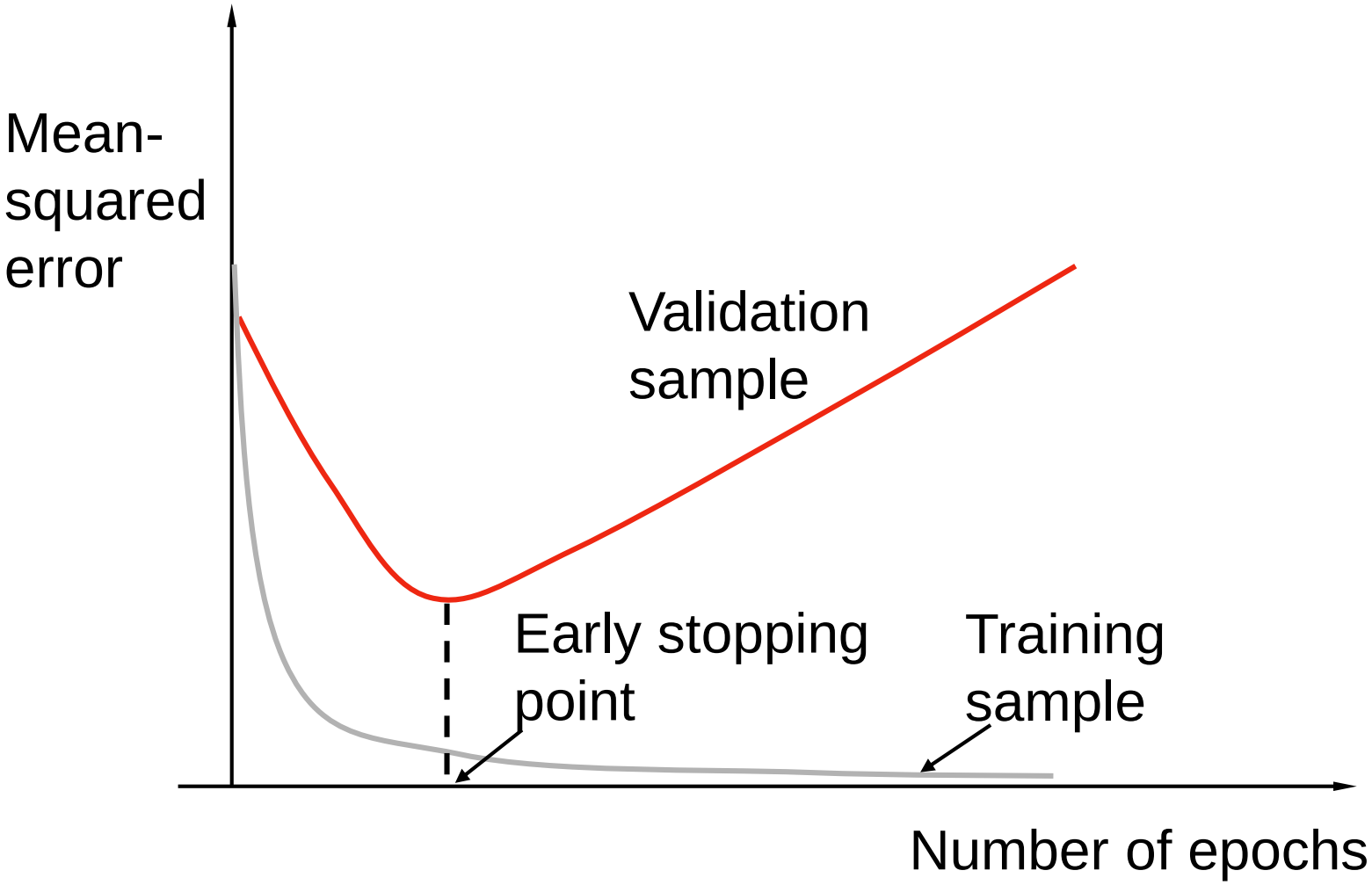
Cross-Validation

- It is hoped that a MLP network trained with back-propagation learns enough from the past to generalize to the future.
- How the network parameterization should be chosen for a specific data set?
- This is a model selection problem: choose the best one of a set of candidate model structures (parameterizations).
- A useful statistical technique for model selection: *cross-validation*.
- The available data set is first randomly partitioned into a training set and a test set.
- The training set is further partitioned into two disjoint subsets:
  - *Estimation subset*, used to select the model.
  - *Validation subset*, used to test or validate the model.

Cross-Validation -2

- The motivation is to validate the model on a data set different from the one used for parameter estimation.
- In this way, the training set can be used to assess the performance of various model.
- The "best" of the candidate models is then chosen.
- This procedure ensures that a model which might in the worst case end up with overfitting the validation subset is not chosen.
- The use of cross-validation is appealing when one should design a large network with good generalization ability.
- For MLP networks, cross-validation can be used to determine:
  - the optimal number of hidden neurons.
  - when it is best to stop training.

Early-Stopping Rule



Network Pruning Techniques

- To minimize the size of the network while maintaining good performance
- To generalize better to new data
- Two pruning techniques:
  - Network growing
  - Network pruning
- Reference: R. Reed, IEEE Trans. Neural Networks, vol. 4, pp. 740-774, 1993.

小结

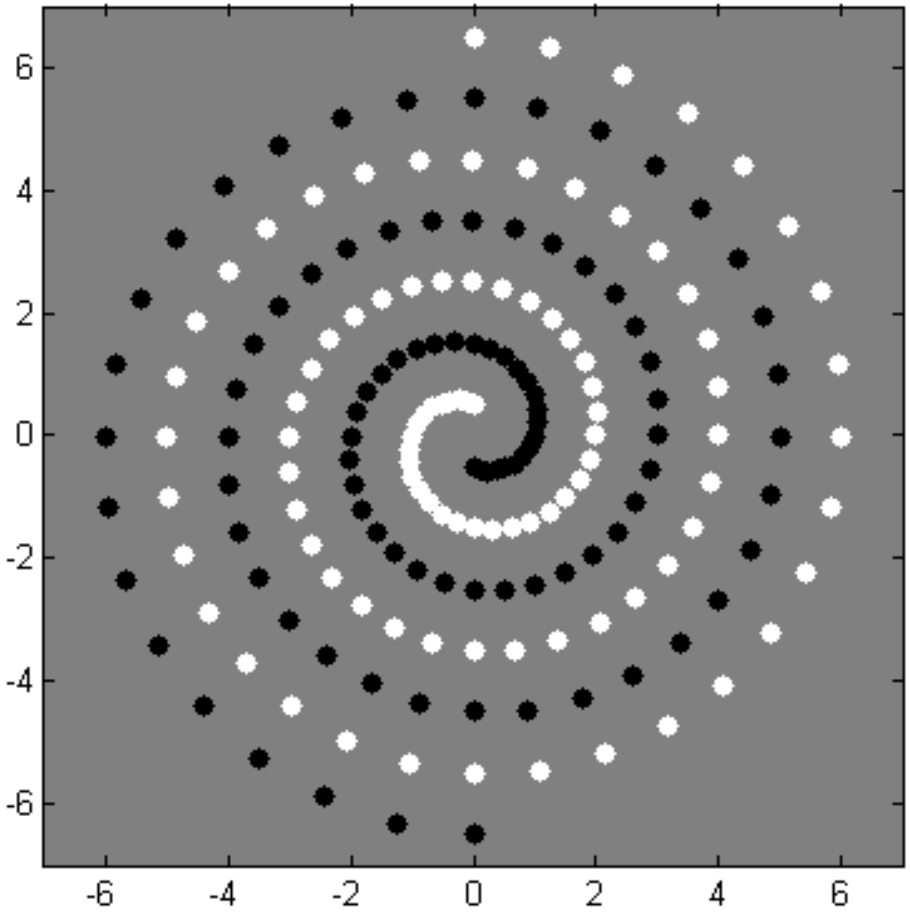
- 多层感知器
- BP算法
- BP算法的推导
- 多层感知器的表征能力

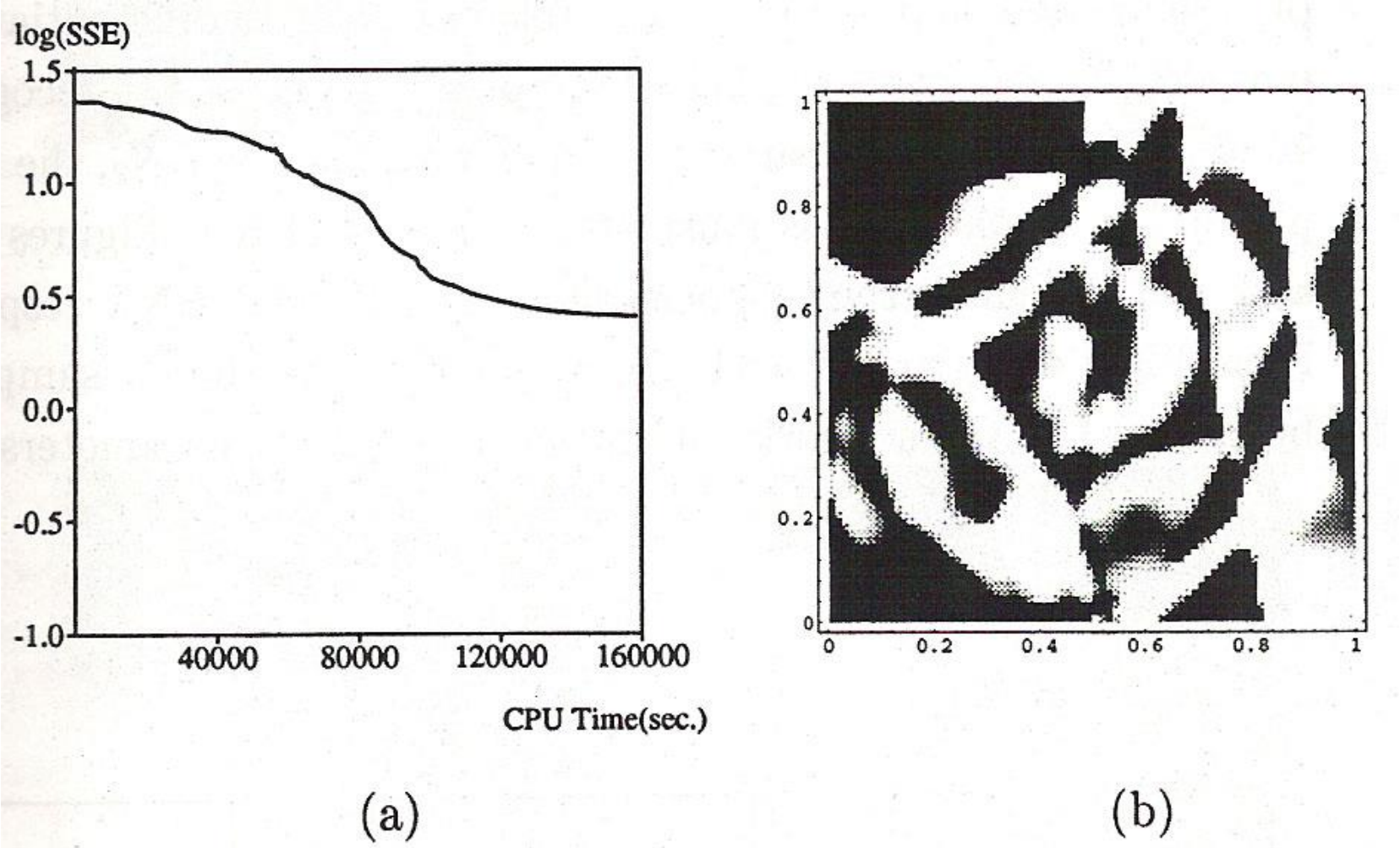
第二讲（下）：
多级筛选网络最小最大模块化网络

吕宝粮
计算机科学与工程系
电信楼群3号楼431
Tel: 021-3420-5422

bllu@sjtu.edu.cn
http://bcmi.sjtu.edu.cn/~blu

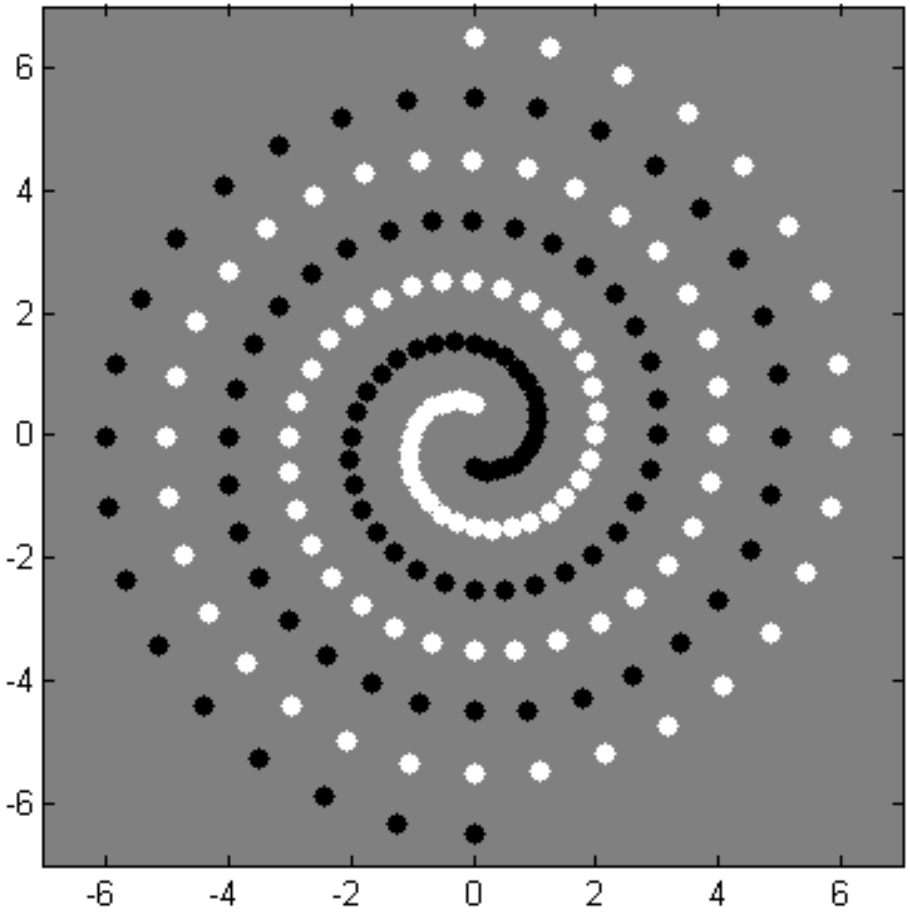
Two Spirals Problem

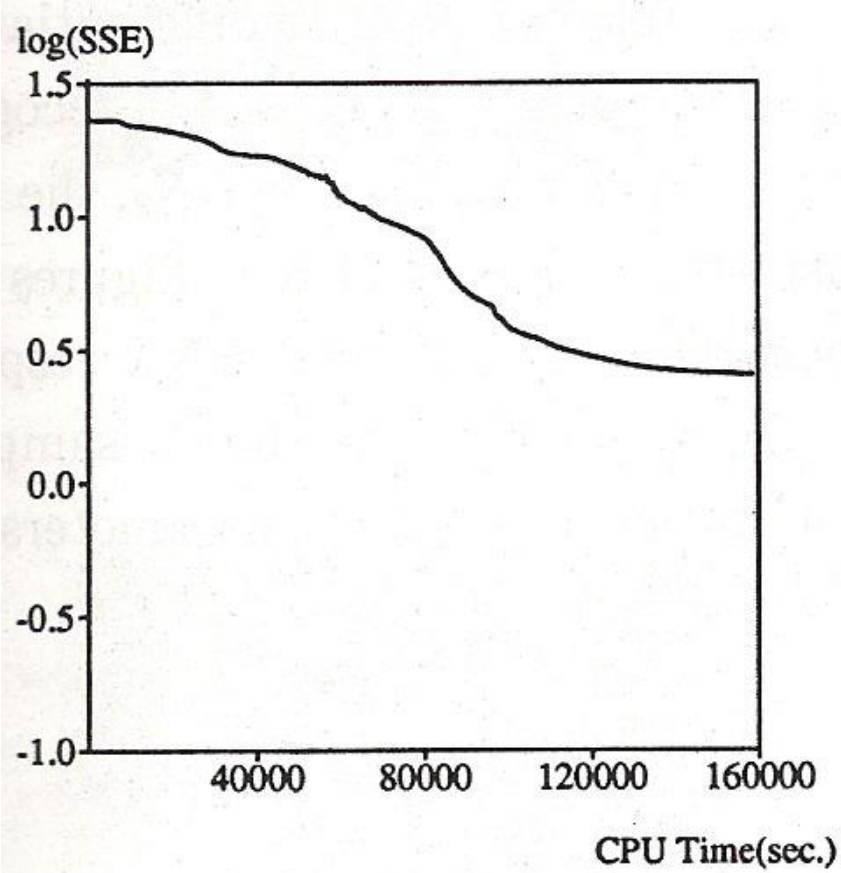



log(SSE)
1.5
1.0
0.5
0.0
-0.5
-1.0
40000
80000
120000
160000
CPU Time(sec.)
(a)
1
0.8
0.6
0.4
0.2
0
0
0.2
0.4
0.6
0.8
1
(b)

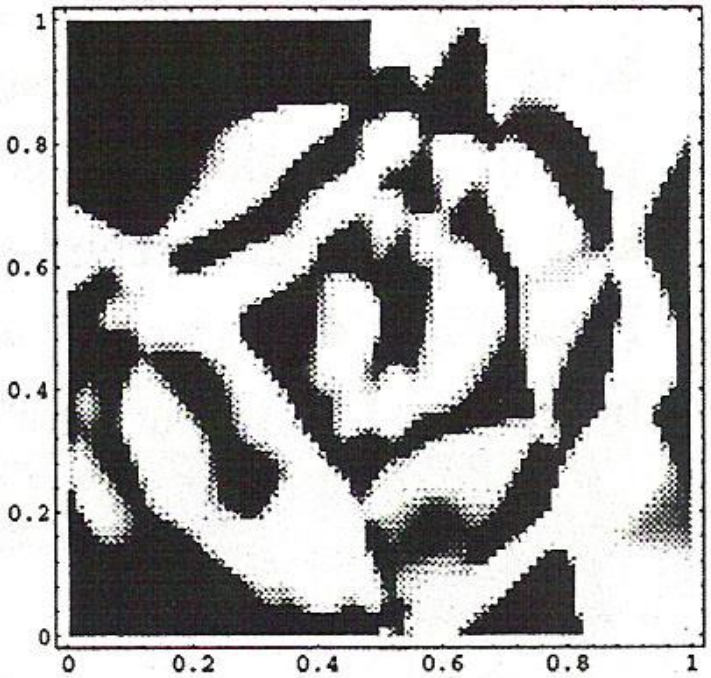
Multi-Sieving Neural Network (Lu, et al., 1994)

Two Spirals Problem



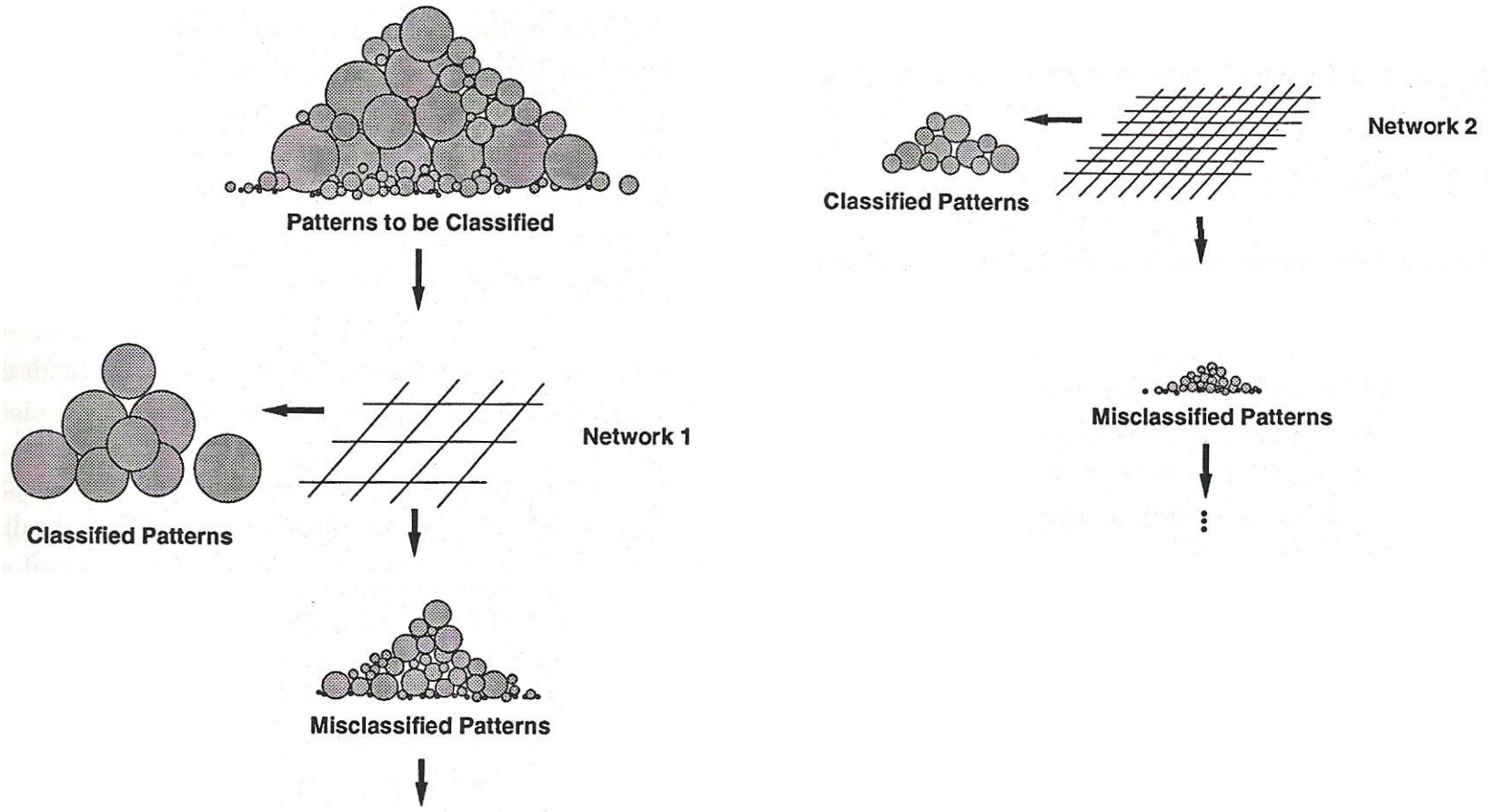

log(SSE)
1.5
1.0
0.5
0.0
-0.5
-1.0
40000
80000
120000
160000
CPU Time(sec.)

(a)


1
0.8
0.6
0.4
0.2
0
0
0.2
0.4
0.6
0.8
1

(b)

Multi-Sieving Network Model



Structure of the Multi-Sieving Network

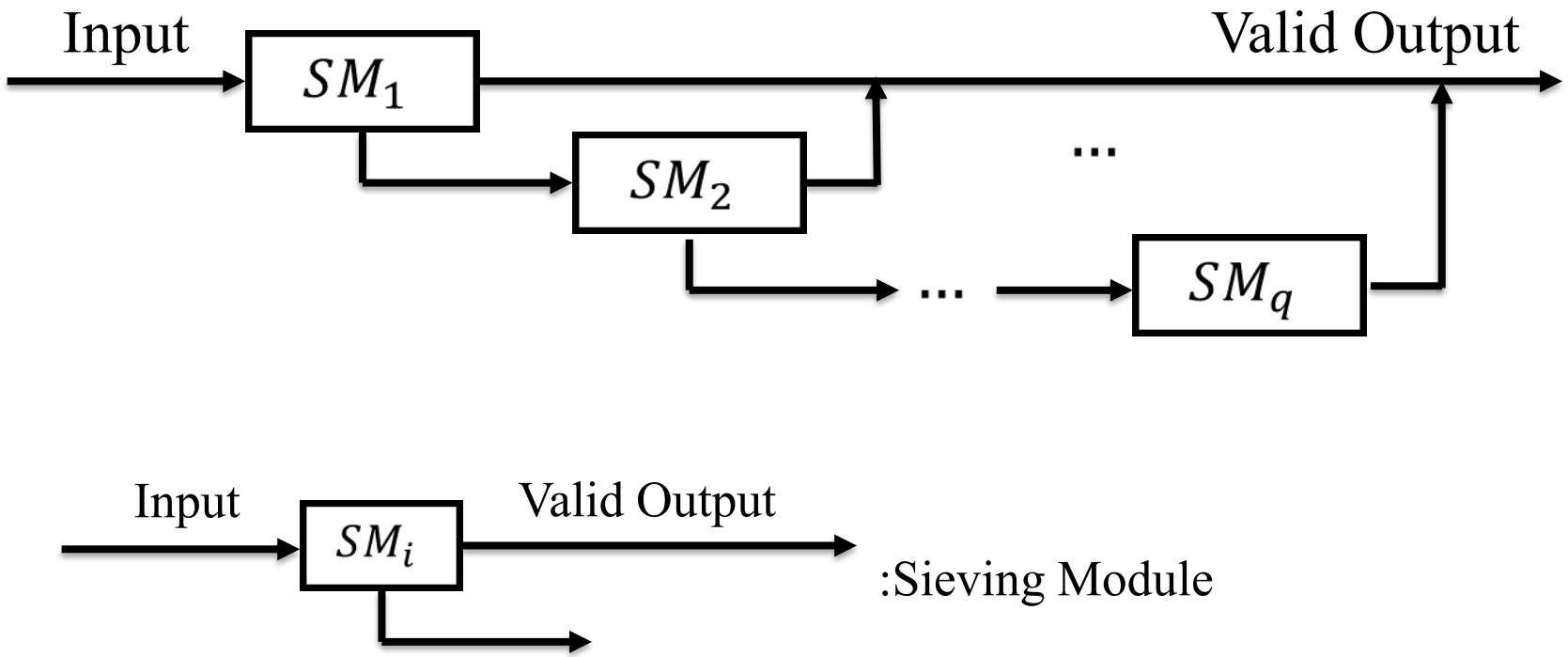


Figure 4.2: The structure of the multi-sieving neural network

Structure of the Multi-Sieving Network (2)

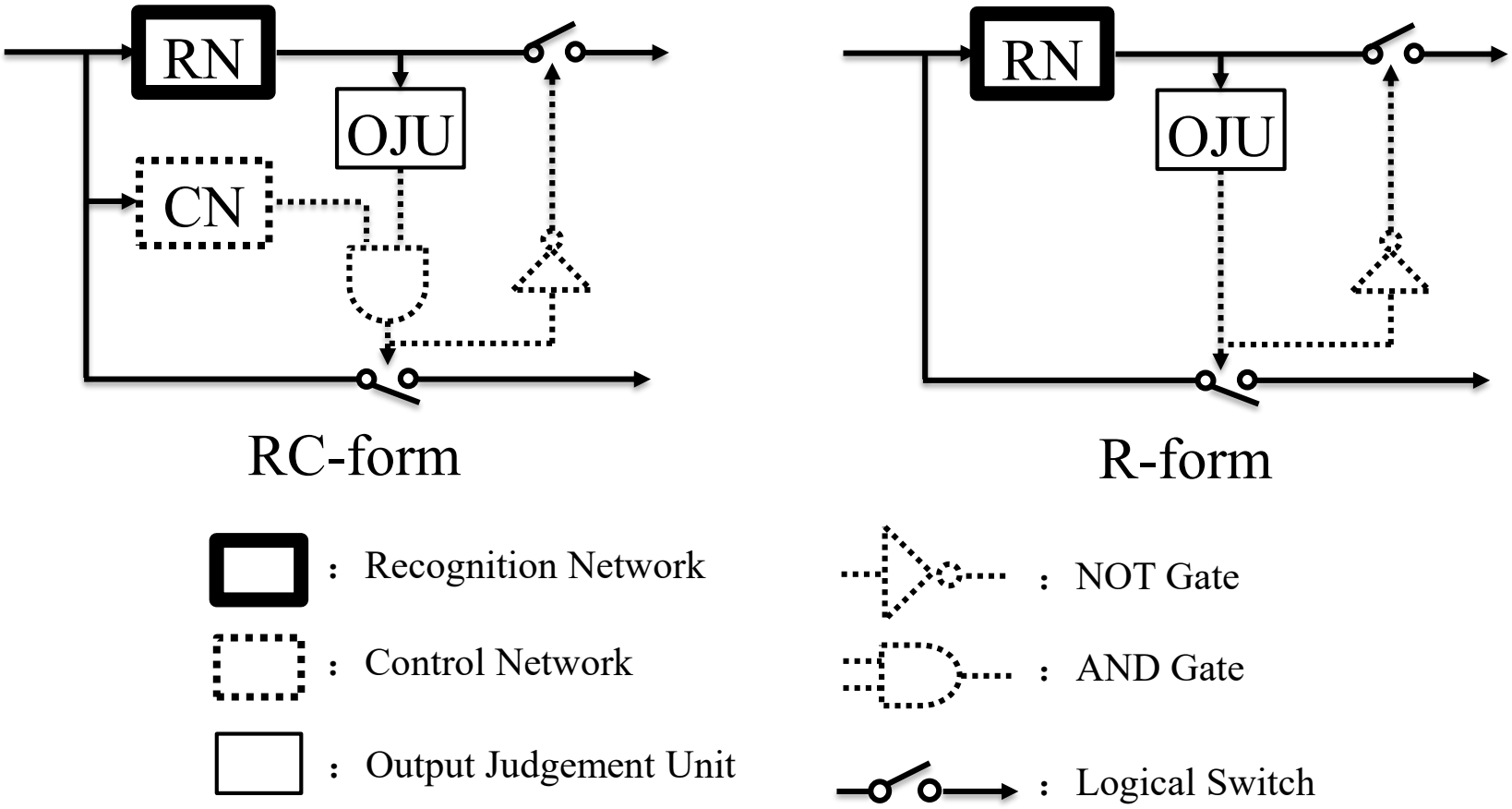


Figure 4.3: Two forms of sieving modules

Valid, Pseudo Valid, and Invalid Outputs

For classification task of $p+1$ sets, we use p or $p+1$ output units. For the kth recognition network RN_k, a desired output pattern $\bar{\boldsymbol{x}}_i^O = \{\bar{x}_{i1}^O, \bar{x}_{i2}^O, \ldots, \bar{x}_{iN_k}^O\}$ must satisfy one of the following rules:

(a) $\forall j(\bar{x}_{ij}^O < x_{low}^O)$ for $j \in B_k$ or

(b) $\exists j\left(\bar{x}_{ij}^O > x_{high}^O\right) \wedge \forall l_{l\neq j}(\bar{x}_{il}^O < x_{low}^O)$ for j and $l \in B_k$

where $B_k = \{1, 2, \ldots, N_k\}$, N_k is the number of output units in RN_k, and x_{low}^O and x_{high}^O represent the low and high bounds for the outputs, respectively. For example, three output units can only represent four valid outputs as follows: (0,0,0), (0,0,1), (0,1,0) and (1,0,0). Other four binary codes, (0,1,1), (1,0,1), (1, 1,0) and (1, 1, 1), are considered to be invalid.

Valid, Pseudo Valid, and Invalid Outputs (2)

For a given input pattern $\bar{\boldsymbol{x}}_i^I$, RN_k may generate three kinds of actual outputs:

(a). *Valid output*: The valid output is the correct output generated by the recognition network. That is,

$$\forall j | \bar{x}_{ij}^O - x_{ij}^{RN_k} | < \delta \quad \text{for} \quad j \in B_k \qquad (4.1)$$

where $\bar{x}_{ij}^O$ is the desired output of the jth unit, $x_{ij}^{RN_k}$ is the actual output of the jth output unit of the kth recognition network RN_k, and δ denotes a tolerance. If the desired values of the output units are set to 0 and 1, then, $x_{low}^O = \delta$ and $x_{high}^O = 1 - \delta$.

Valid, Pseudo Valid, and Invalid Outputs (3)

(b). *Pseudo valid output*: The recognition network may generate an output which follows the coding rule mentioned above, but is not correct. We called such an output a *pseudo valid output*:

$$\exists j_{j\neq h}\left(x_{ij}^{RN_k} > x_{high}^{O}\right) \wedge \forall l_{l\neq j}\left(x_{il}^{RN_k} < x_{low}^{O}\right) \vee \forall l\left(x_{il}^{RN_k} < x_{low}^{O}\right) \quad \text{for } j \text{ and } l \in B_k \qquad (4.2)$$

where the desired output of the hth unit satisfies $\bar{x}_{ih}^{O} > x_{high}^{O}$.

(c). *Invalid output*: Otherwise.

For example, if the desired output pattern is (0, 0, 1), $\delta = 0.2$, $x_{high}^{O} = 0.8$, and $x_{low}^{O} = 0.2$, then, (0.1, 0.1, 0.9) is a valid output, (0.9, 0.1, 0.1) and (0.1, 0.1, 0.1) are two pseudo valid outputs, and (0.9, 0.1, 0.9) is an invalid output.

(a) The output judgement unit is used to differentiate the invalid output from other two kinds of outputs. OJU generates 1 or 0 according to

$$O_{OJU,k} = \begin{cases} 1, \text{if } x_{ij}^{RN_k} \text{ is a valid or pseudo valid output;} \\ 0, \text{otherwise} \end{cases}$$

where $O_{OJU,k}$ is the output of OJU in the kth sieving module, $A = \{1, 2, \ldots, t_k\}$, t_k is the number of examples used for training the kth recognition network.

(b) The control network is used to differentiate the valid output from the pseudo valid output. Its output is also 1 or 0, which is determined by

$$O_{CN,k} = \begin{cases} 1, \text{if } x_{ij}^{RN_k} \text{ is a valid output;} \\ 0, \text{otherwise} \end{cases}$$

where $O_{CN,k}$ is the output of the control network in the kth sieving module.

Multi-Sieving Learning Algorithm

Let T_1 be a set of t_1 training samples to be learned:

$$T_1 = \{(\bar{\boldsymbol{x}}_i^I, \bar{\boldsymbol{x}}_i^O) | for\ i = 1, 2, \ldots, t_1\}$$

where $\bar{\boldsymbol{x}}_i^I \in \boldsymbol{R}^{N_I}$ and $\bar{\boldsymbol{x}}_i^O \in \boldsymbol{R}^{No}$ are the input and the desired output of the ith sample, respectively.

The multi-sieving learning algorithm works as follows:

Step 1 : Initially, one recognition network, namely RN_1, is trained on the original set T_1 up to a certain number of epochs. Let $m = 1$,and proceed to the following steps.

Step 2 : Compute the number of valid outputs, $N_{VO,m}$, and the number of pseudo valid outputs, $N_{PVO,m}$, according to Eqs. 4.1 and 4.2, respectively.

Multi-Sieving Learning Algorithm (2)

Step 3 : If $\sum_{i=1}^{m} N_{VO,i} = t_1$, ie., if all t_1 samples are learned by the multi-sieving network, then the training is completed.

Step 4 : If $N_{PVO,m} = 0$, i.e., if there are no pseudo valid outputs, then the control network is unnecessary. Go to Step 6.

Step 5 : If $N_{PVO,m} > 0$, i.e., if there exist $N_{PVO,m}$ pseudo valid outputs, then, a control network, namely CN_m, is trained on the set of $N_{VO,m} + N_{PVO,m}$ samples until all of the samples are classified correctly. These samples correspond to valid or pseudo valid outputs generated by RN_m.

Multi-Sieving Learning Algorithm (3)

Step 6 : Freeze the parameters of RN_m and CN_m (if it exists), and construct the mth sieving module as shown in Fig. 4.3.

Step 7 : Remove $N_{VO,m}$ samples which have been successfully classified by RN_m from T_m and create a new set of $t_{m+1}(t_{m+1} = t_m - N_{VO,m})$ samples $T_{m+1}(T_{M+1} \subset T_m)$, which are not classified by RN_m.

Step 8 : Train RN_{m+1} on T_{m+1} up to a certain number of epochs. Let $m = m + 1$ and go back to Step 2.

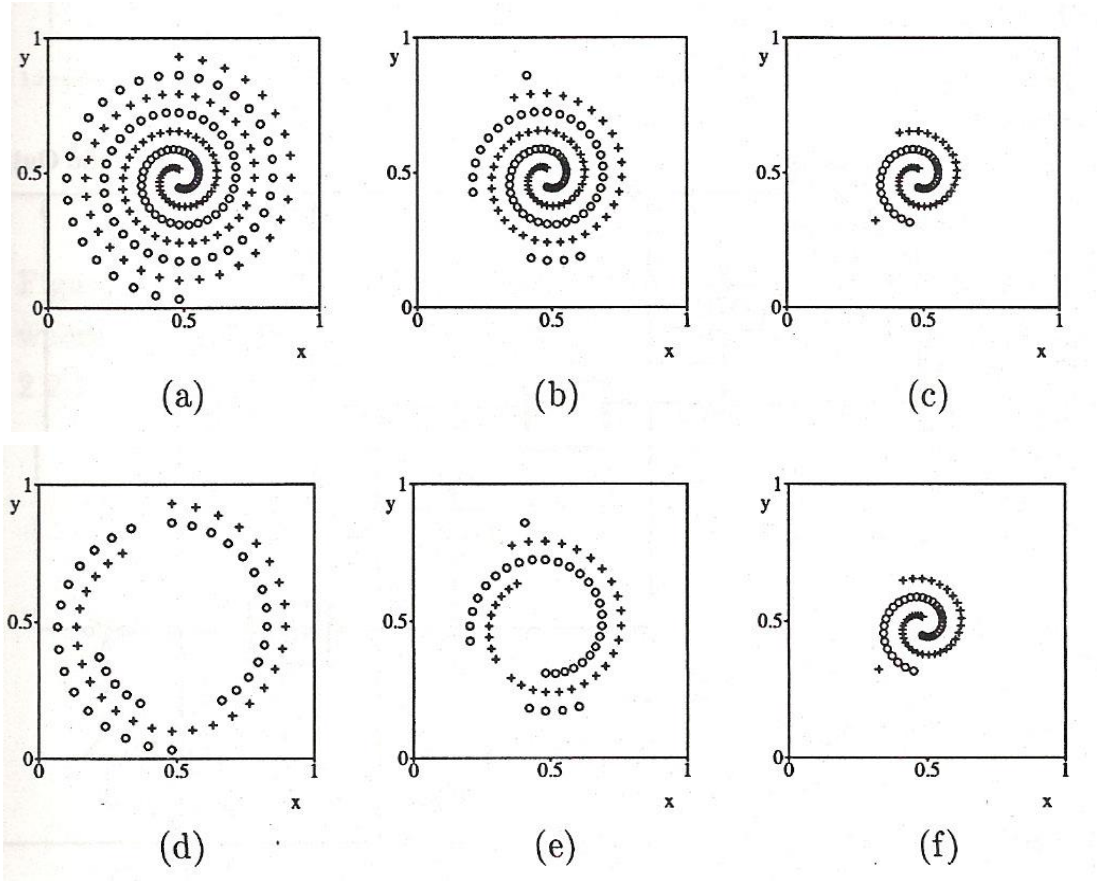


Figure 4.5: Patterns used for training RN_1 (a), RN_2 (b), and RN_3 (c). Patterns learned by the first sieving module (d), the second sieving module (e), and the third sieving module (f). For symbols "o" and "+", the network is required to generate output 0 and 1, respectively.

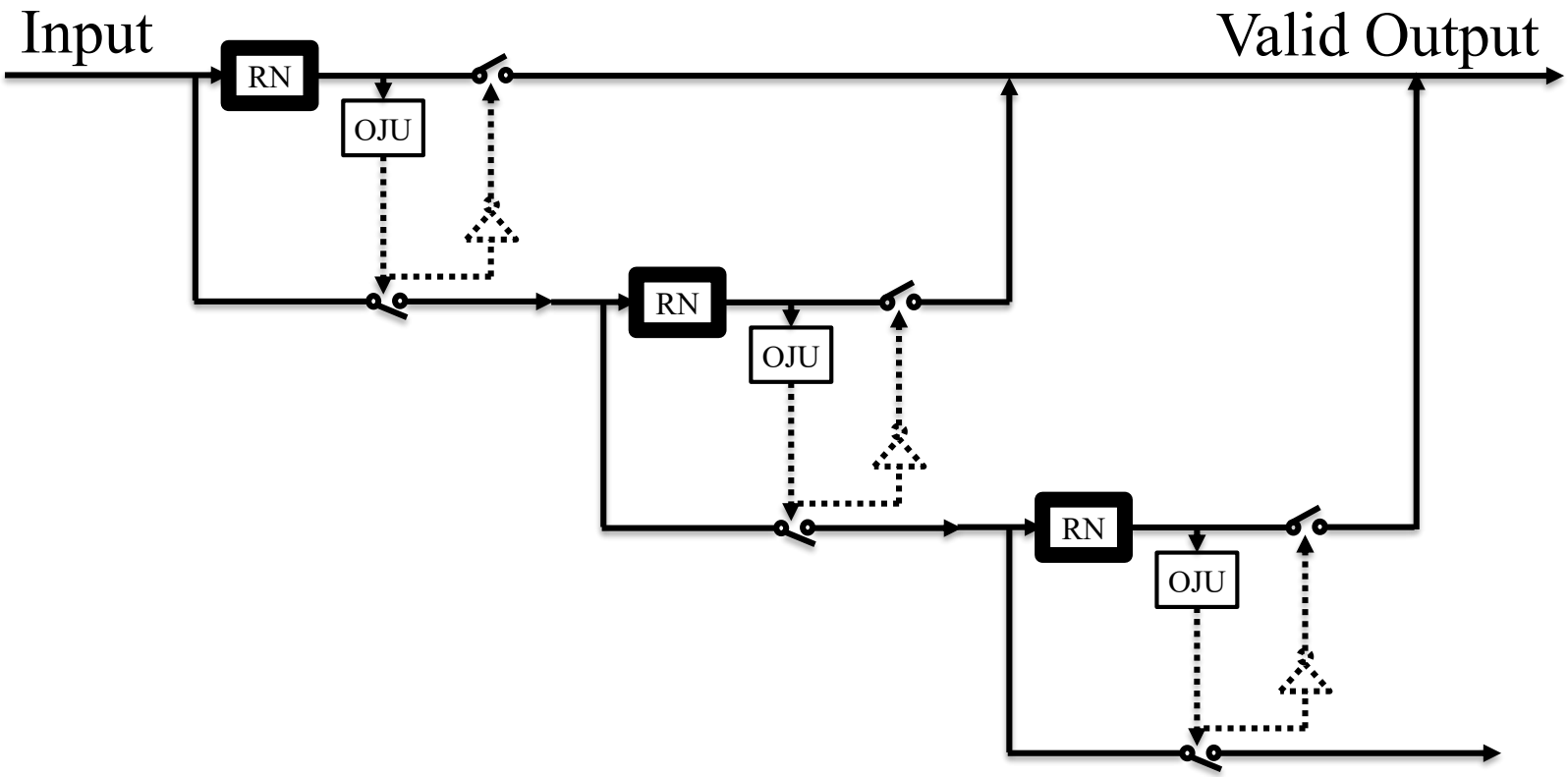


Figure 4.6: The multi-sieving network with three sieving modules for the "two-spirals" problem, namely MSN_{ts}^{3}.

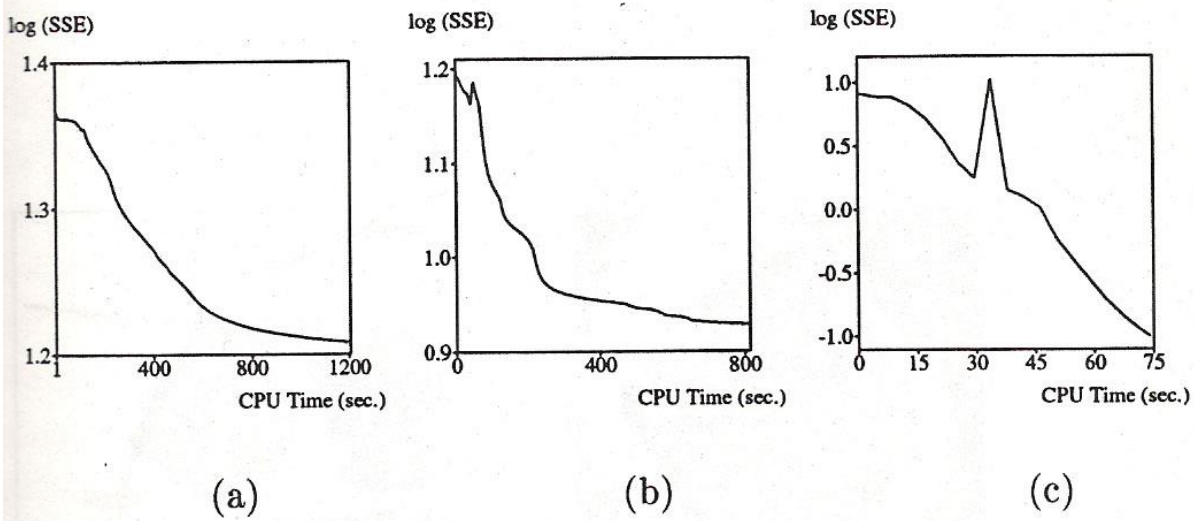


Figure 4.7: The learning curves of RN_1 (a), RN_2 (b), and RN_3 (c), where SSE means the sum of squared error as defined in Subsection 2.2.1

Table 4.1: The learning rates for the recognition networks

	RN_1	RN_2	RN_3
η_1	0.084	0.11	0.21
η_2	0.41	0.81	0.51

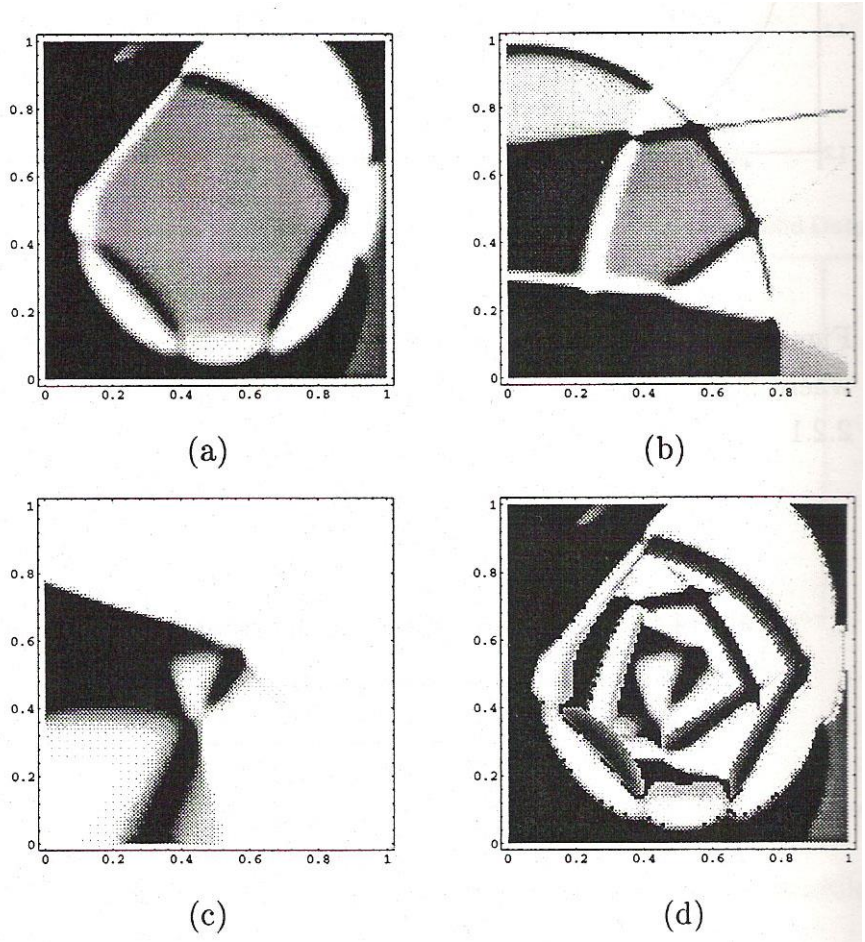


Figure 4.8: Response plots of the first sieving module (a), the second sieving module (b), the third sieving module (c), and the whole multi-sieving network (d). Black and white represent output of "0" and "1", respectively, and grey represents intermediate value.

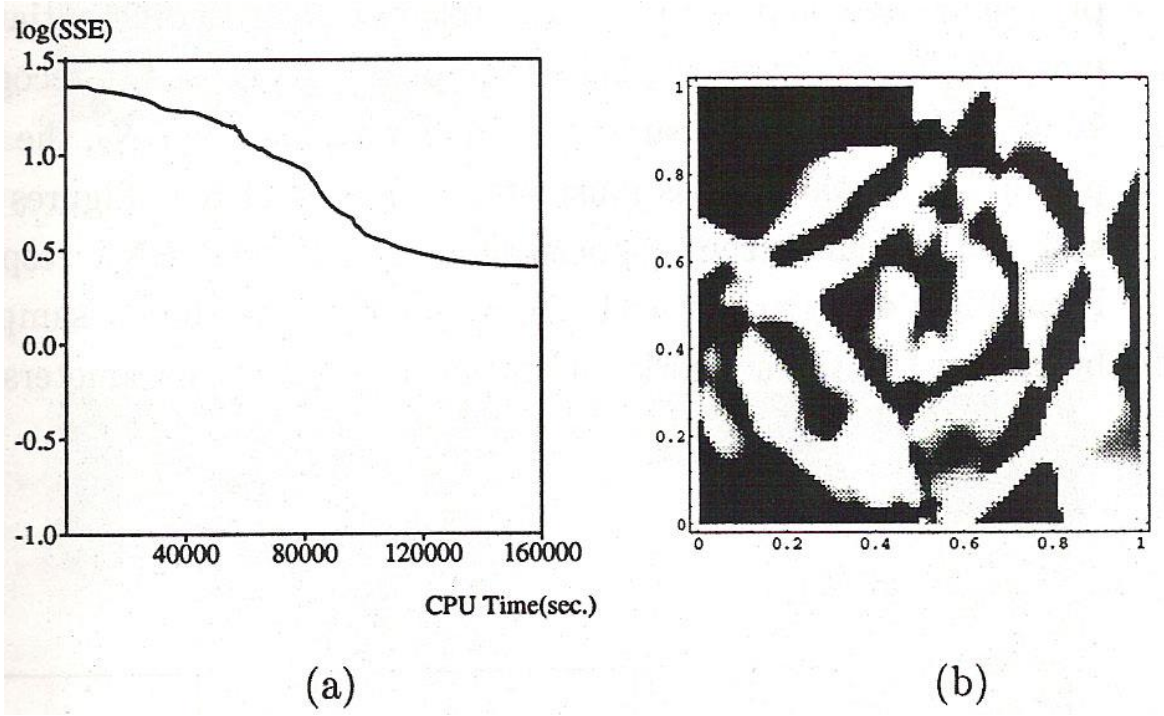


Figure 4.9: The learning curve (a) and the response plot of a plain MLQP for the "two-spirals" problem (b).

Incremental Learning based on MSL

Suppose a set of t_1 samples T_1 has been successfully learned by an MSN with q sieving modules, namely MSN_{old}^{q}. Now, the problem is how to add a new set of u_1 samples U_1

$$U_1 = \{(\overline{x}_j^I, \overline{x}_j^O) | \text{for } j = 1, 2, \ldots, u_1\}$$

to MSN_{old}^{q}.

Incremental Learning based on MSL

We suppose that all q sieving modules are RC-form sieving modules. Let the numbers of the valid outputs and the pseudo valid outputs produced by the kth recognition network RN_k^q in MSN_{old}^q be $N_{VO,k}^q$, and $N_{PVO,k}^q$, respectively, and let $N_{VO,k}^{new}$ and $N_{PVO,k}^{new}$ be the numbers of valid outputs and pseudo valid outputs generalized by the kth recognition network RN_k^q, respectively, when the new training inputs are presented to MSN_{old}^q.

Incremental Learning Algorithm (1)

Step 1 : Initially, present all u_1 new training inputs to the first recognition network RN_1^q, let $m = 1$, and proceed the following steps.

Step 2 : Compute $N_{VO,m}^{new}$ and $N_{PVO,m}^{new}$ according to Eqs. 4.1 and 4.2.

Step 3 : If $((m \leq q) \wedge (\sum_{i=1}^{m} N_{VO,i}^{new} = u_1))$, i.e., if all u_1 new training inputs are generalized correctly by RN_q^q through RN_m^q then the additional learning is completed.

Incremental Learning Algorithm (2)

Step 4 : If $((m > q) \wedge \left(\sum_{i=1}^{q} N_{VO,i}^{new} < u_1\right))$, then go to Step 8.

Step 5 : If $((m \leq q) \wedge \left(\sum_{i=1}^{m} N_{VO,i}^{new} < u_1\right) \wedge (N_{VO,m}^{new} = 0) \wedge (N_{PVO,m}^{new} = 0))$, then CN_m^q remains unchanged. Go to Step 7.

Step 6 : If $((m \leq q) \wedge \left(\sum_{i=1}^{m} N_{VO,i}^{new} < u_1\right))$, then replace CN_m^q with a new control network CN_m^{new} which is trained on a set of $N_{vo,m}^{new} + N_{VO,m}^{q} + N_{PVO,m}^{new} + N_{PVO,m}^{q}$ training data corresponding to valid or pseudo valid outputs which are produced and generalized by RN_m^q, respectively.

Step 7 : Let $m = m + 1$, and go back to Step 2.

Incremental Learning Algorithm (3)

Step 8 : Remove $\sum_{i=1}^{q} N_{VO,i}^{new}$ samples, which are generalized correctly by RN_1^q through RN_q^q, from U_1 and train a new multi-sieving network, namely MSN_{new}^p, on a set of $u_1 - \sum_{i=1}^{q} N_{VO,i}^{new}$ samples according to the MSL algorithm.

Step 9 : Suppose the set of $u_1 - \sum_{i=1}^{q} N_{VO,i}^{new}$ samples has been successfully learned by MSN_{new}^p. Connect MSN_{old}^q to MSN_{new}^p in series as shown in Fig. 4.4.

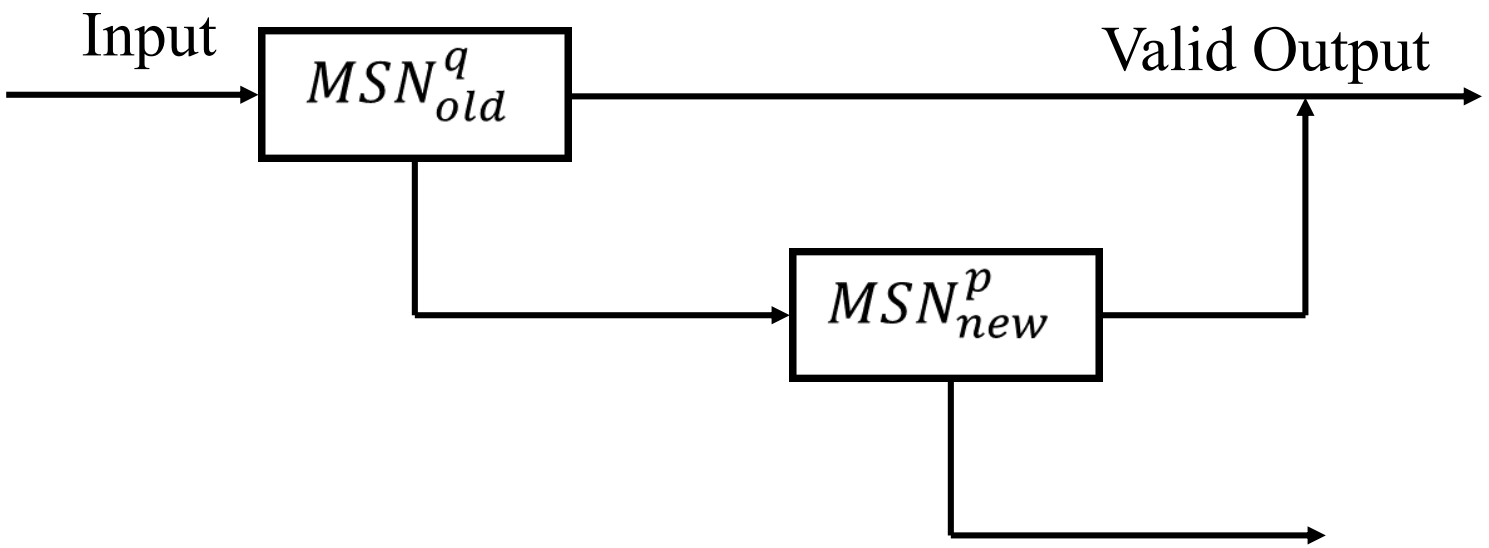


Figure 4.4: The connection of MSN_{old}^{q} to MSN_{new}^{p}

Modification of MSN

Suppose a set of t_1 samples T_1 has been successfully learned by an MSN with q sieving modules, namely MSN_{old}^{q}. Now, there is a set of v_1 samples $V_1 (V_1 \subset T_1)$, whose desired outputs are changed due to the changed environment.

Let the numbers of the valid outputs and the pseudo valid outputs produced by the kth recognition network RN_k^q in MSN_{old}^{q} be $N_{VO,k}^{q}$ and $N_{PVO,k}^{q}$, respectively, and let $N_{VO,k}^{new}$ and $N_{PVO,k}^{new}$ be the numbers of valid outputs and pseudo valid outputs generalized by the kth recognition network RN_k^q respectively, when the training inputs from V_1 are presented to MSN_{old}^{q}.

Step 1 : Initially, present all v_1 updated sample inputs to the first recognition network RN_1^q, let $m = 1$, and proceed the following steps.

Step 2 : Compute $N_{VO,m}^{new}$ and $N_{PVO,k}^{new}$ according to Eqs. 4.1 and 4.2.

Step 3 : If $((m \leq q) \wedge (\sum_{i=1}^{m} N_{VO,i}^{new} = v_1))$, i.e., if all v_1 updated training samples are generalized correctly by RN_1^q through RN_m^q, then the modifications to MSN_{old}^q is completed.

Step 4 : If $((m > q) \wedge (\sum_{i=1}^{q} N_{VO,i}^{new} < v_1))$, then go to Step 8.

Step 5 : If $((m \leq q) \wedge \left(\sum_{i=1}^{m} N_{VO,i}^{new} < v_1\right) \wedge (N_{VO,m}^{new} = 0) \wedge (N_{PVO,m}^{new} = 0))$, then CN_m^q remains unchanged. Go to Step 7.

Step 6 : If $((m \leq q) \wedge \left(\sum_{i=1}^{m} N_{VO,i}^{new} < v_1\right) \wedge (N_{VO,m}^{new} > 0) \wedge (N_{PVO,m}^{new} > 0))$, then replace CN_m^q with a new control network CN_m^{new} which is trained on a set of $N_{VO,m}^{new} + N_{VO,m}^{q} + N_{PVO,m}^{new} + N_{PVO,m}^{q}$ training samples.

Step 7 : Remove $N_{VO,m}^{new}$ samples which have been generalized properly by RN_m^q from V_m and create a new set of $v_{m+1}(v_{m+1} = v_m - N_{VO,m}^{new})$ updated samples $V_{m+1}(V_{m+1} \subset V_m)$. Let $m = m + 1$, and go back to Step 2.

Step 8 : Train a new multi-sieving network, namely MSN_{new}^{p}, on a set of $v_1 - \sum_{i=1}^{q} N_{VO,i}^{new}$ updated samples according t the MSL algorithm.

Step 9 : Suppose the set of $v_1 - \sum_{i=1}^{q} N_{VO,i}^{new}$ updated samples has been successfully learned by MSN_{new}^{p}. Connect MSN_{old}^{q} to MSN_{new}^{p} in series as shown in Fig. 4.4.

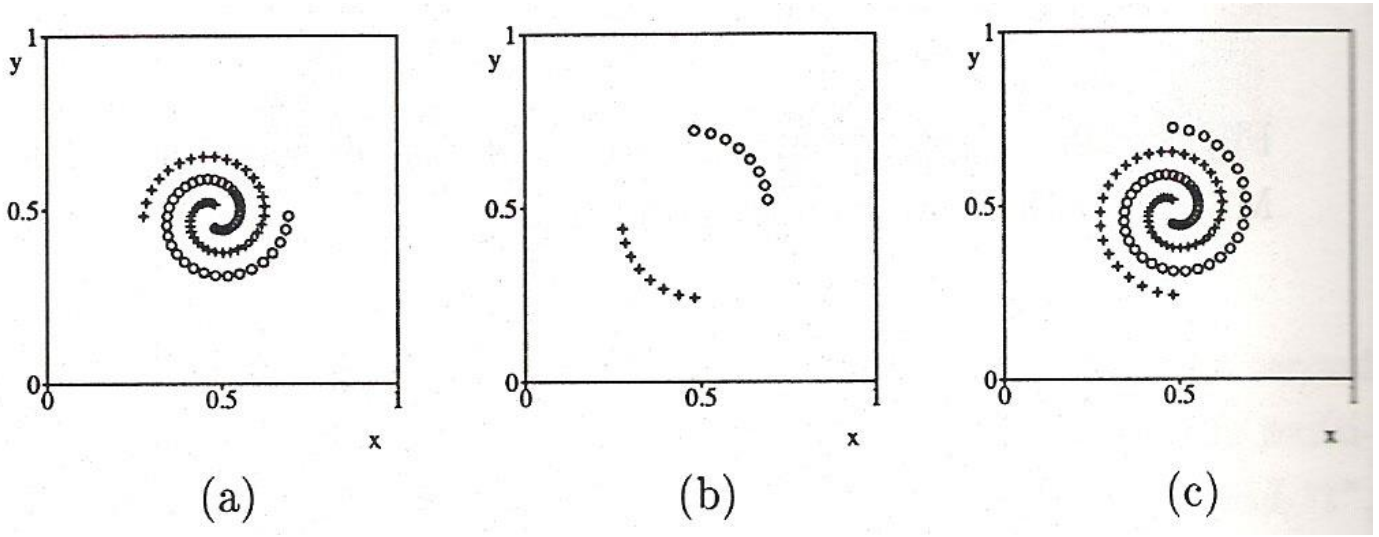


Figure 4.10: The original 82 samples (a), the 16 new samples to be added (b), and the augmented problem (c).

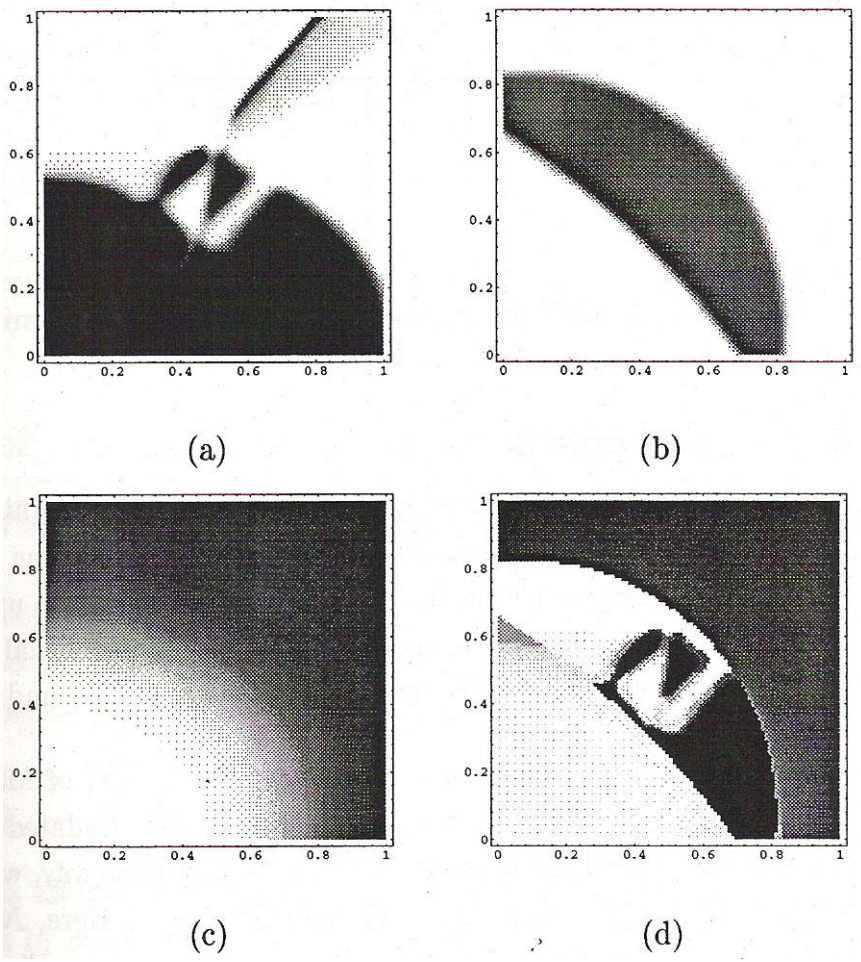


Figure 4.11: Response plots of RN_1 (a), CN_1 (b), RN_2 (c), and the whole multi−sieving network (d).

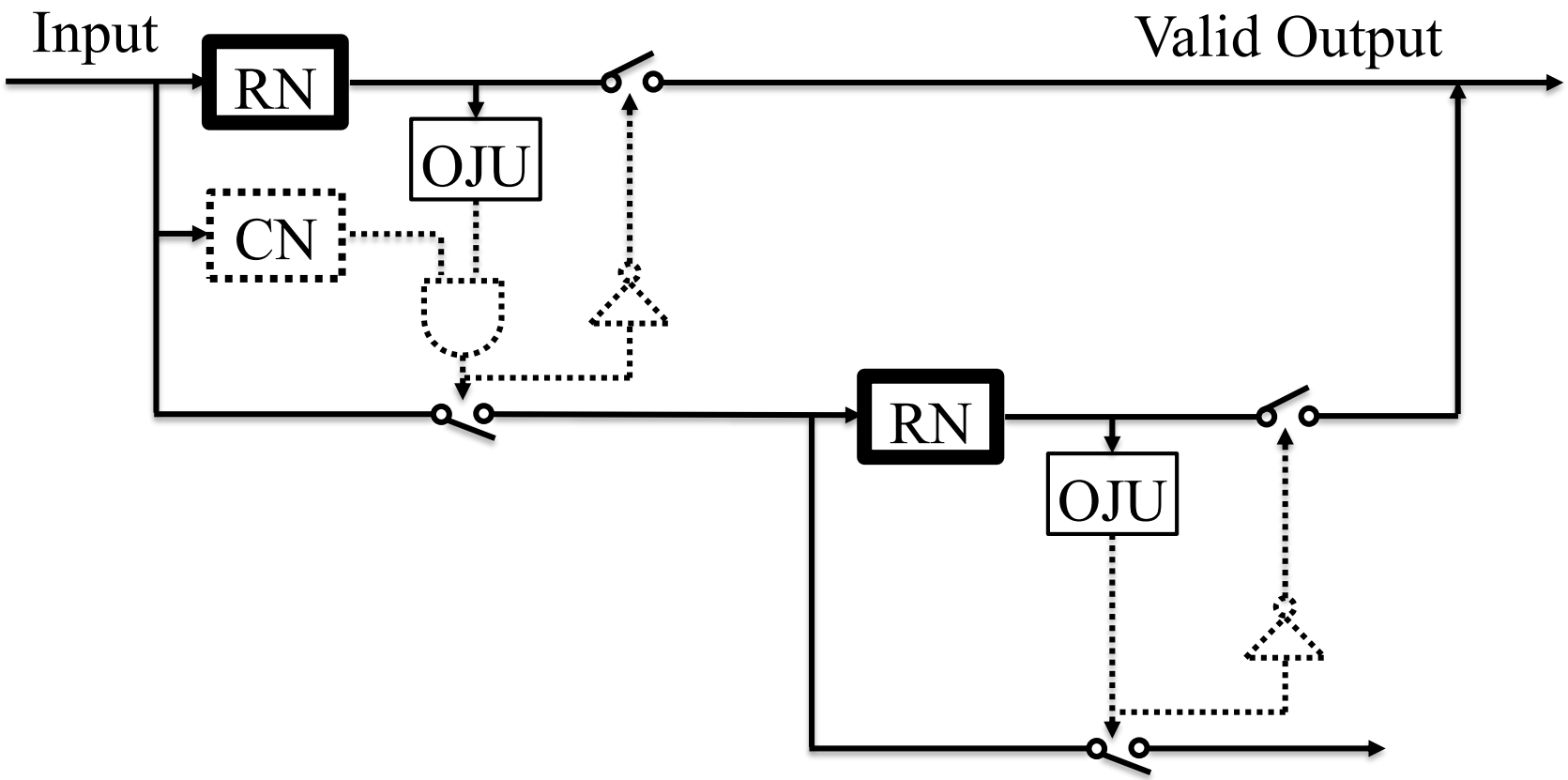


Figure 4.12: MSN for implementing the additional learning.

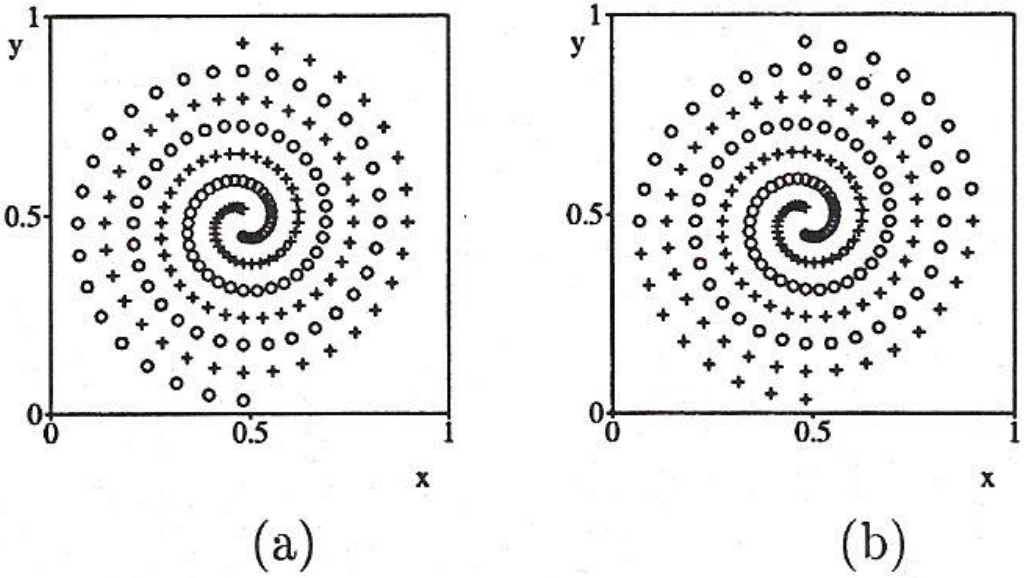


Figure 4.13: The original (a) and the updated (b) two-spirals problems.

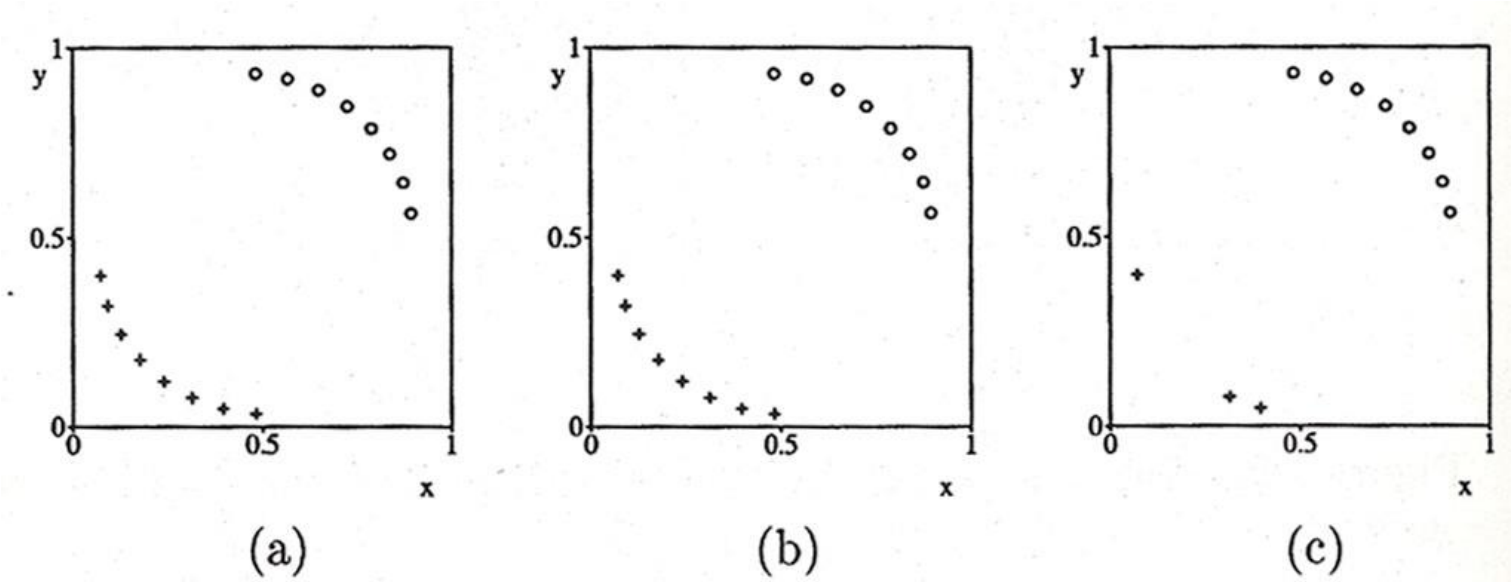


Figure 4.14: The input patterns which are mapped into pseudo valid outputs by RN_1^3 (a), RN_2^3 (b), and RN_3^3 (c), respectively.

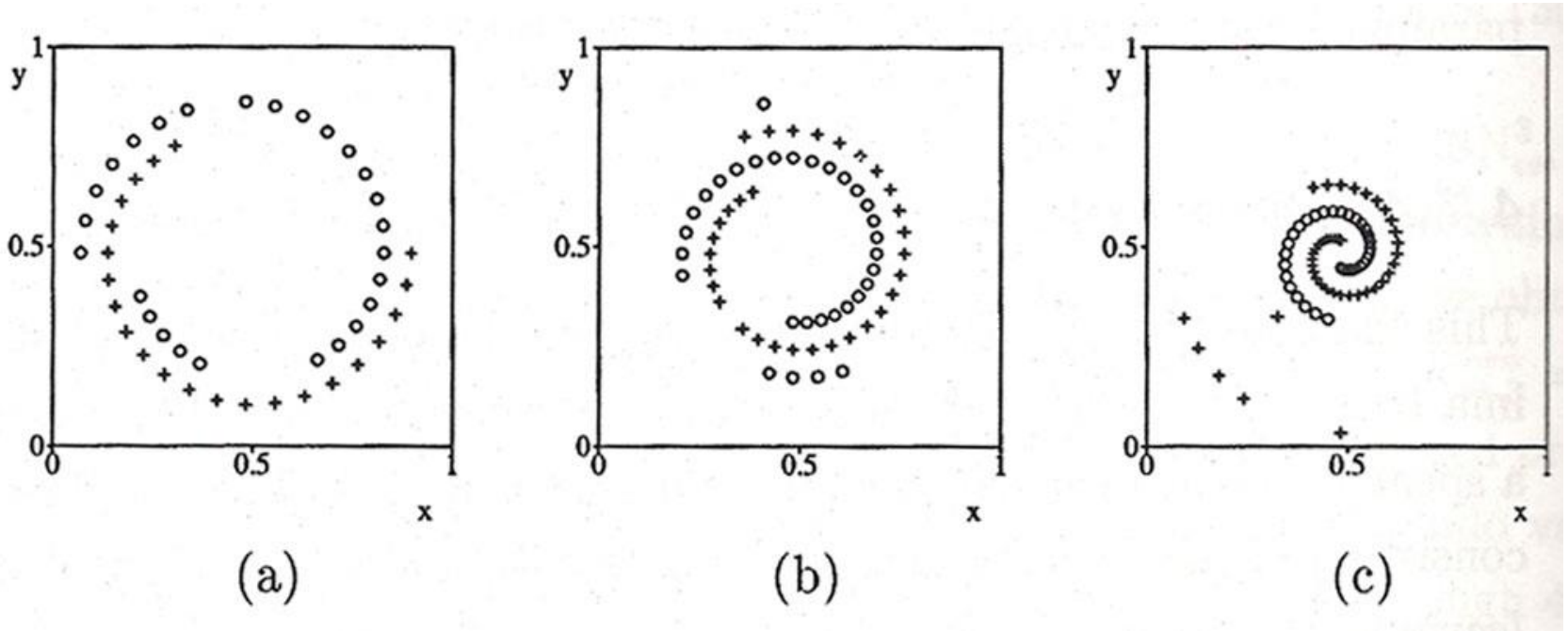


Figure 4.15: The input patterns which are generalized correctly by RN_1^3 (a), RN_2^3 (b), and RN_3^3 (c), respectively.

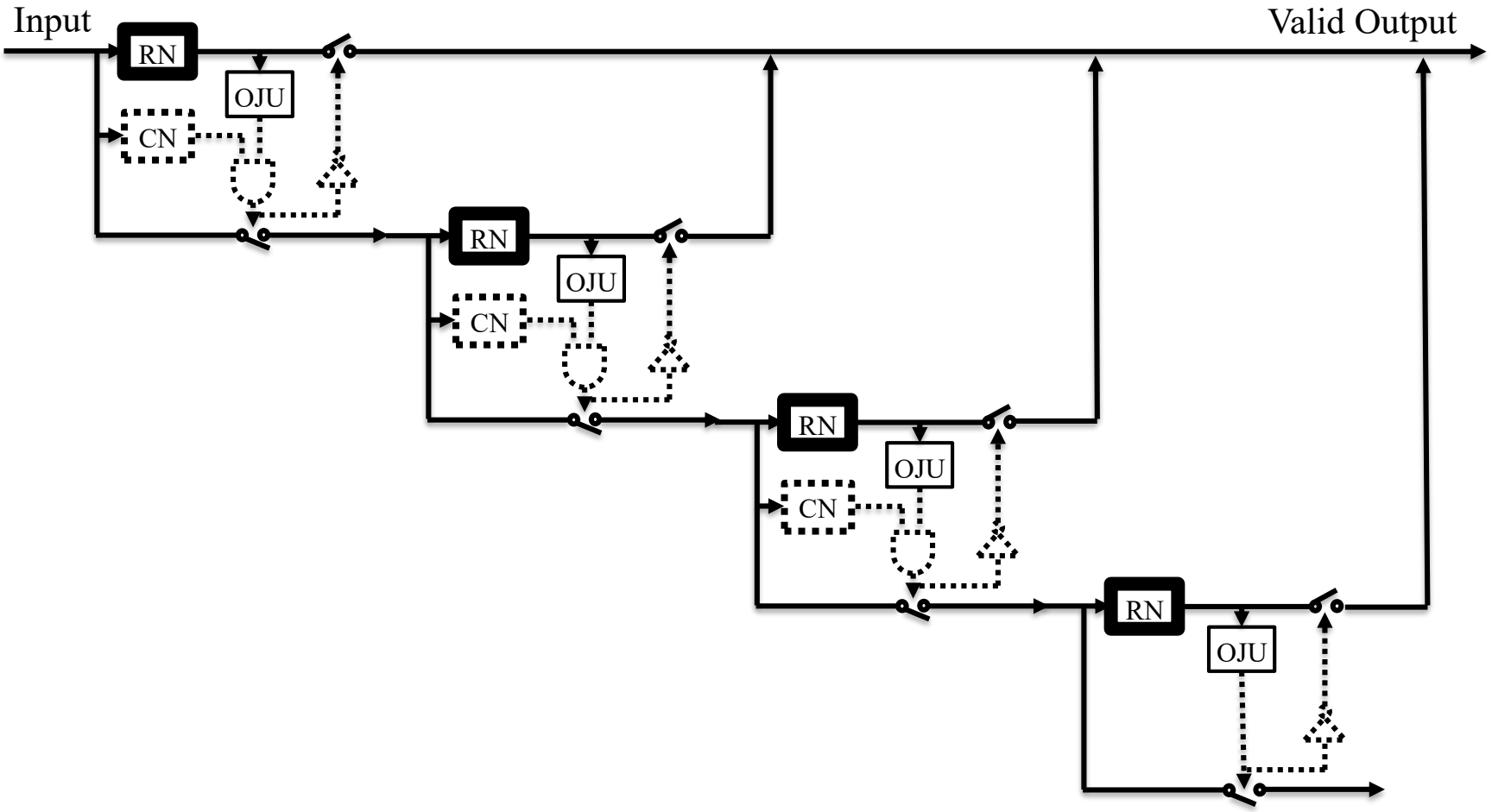


Figure 4.16: The multi-sieving network with four sieving modules for modifying MSN_{ts}^{3}, namely MSN_{uts}^{4}.

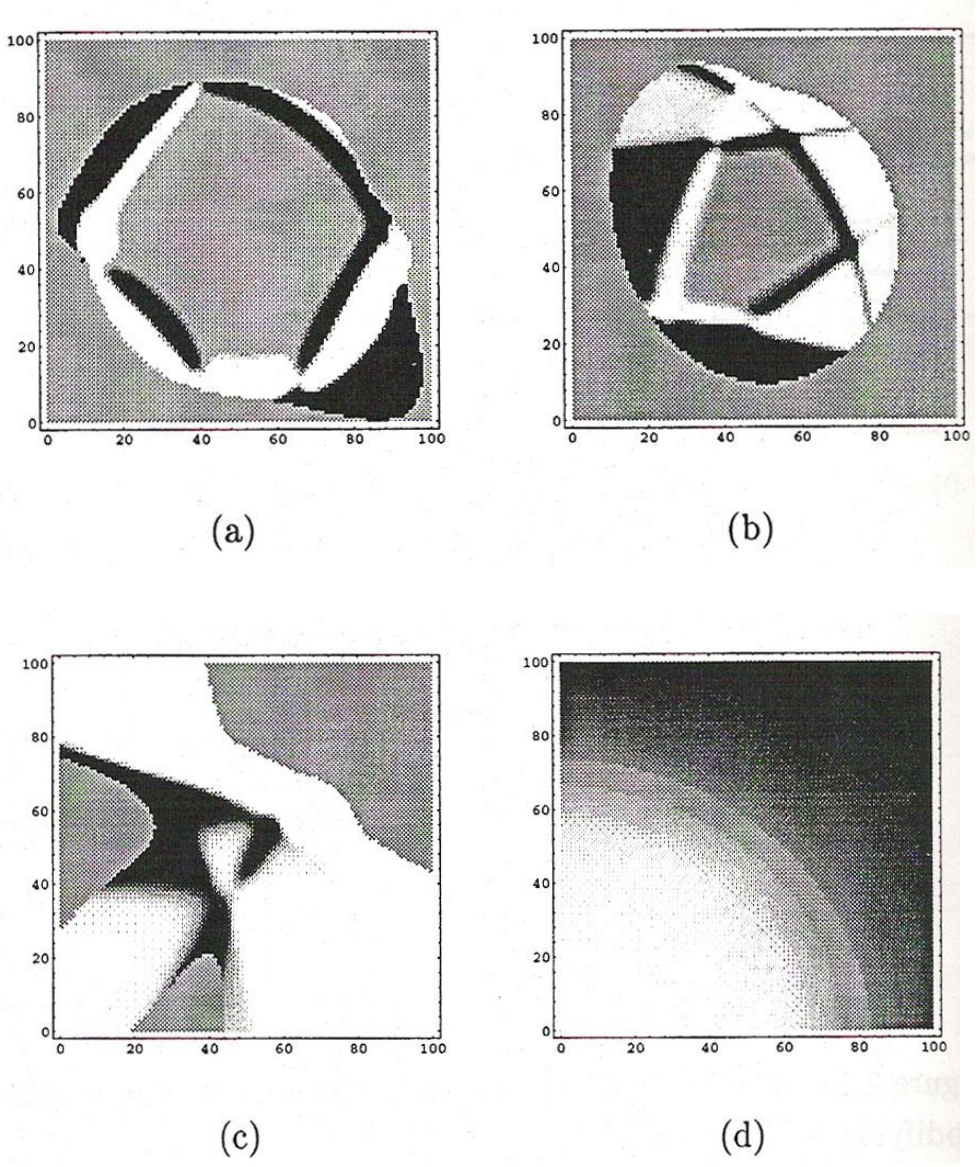


Figure 4.17: Response plots of the first sieving module (a), the second sieving module (b), the third sieving module (c), and the fourth sieving module ((1). In (a), (b) and (c), the grey areas indicate that the inputs located in these areas will pass to the next sieving module.

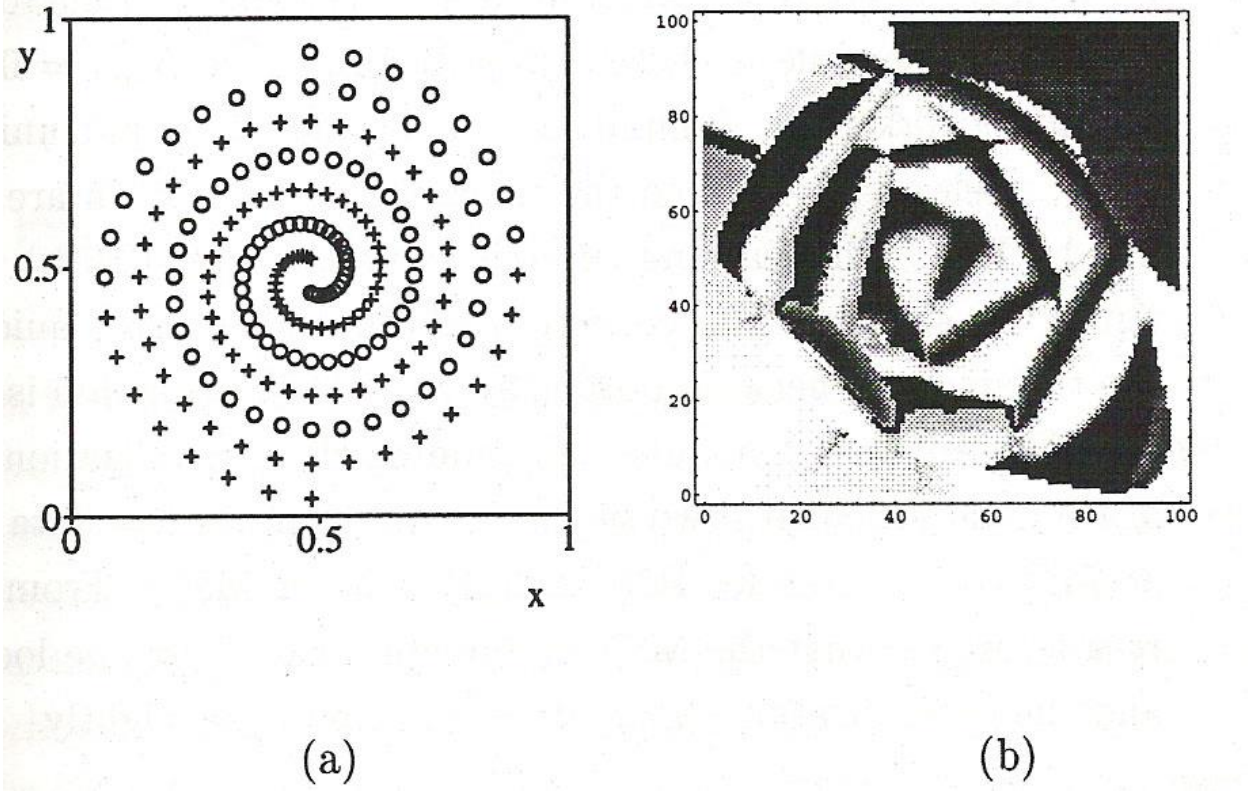
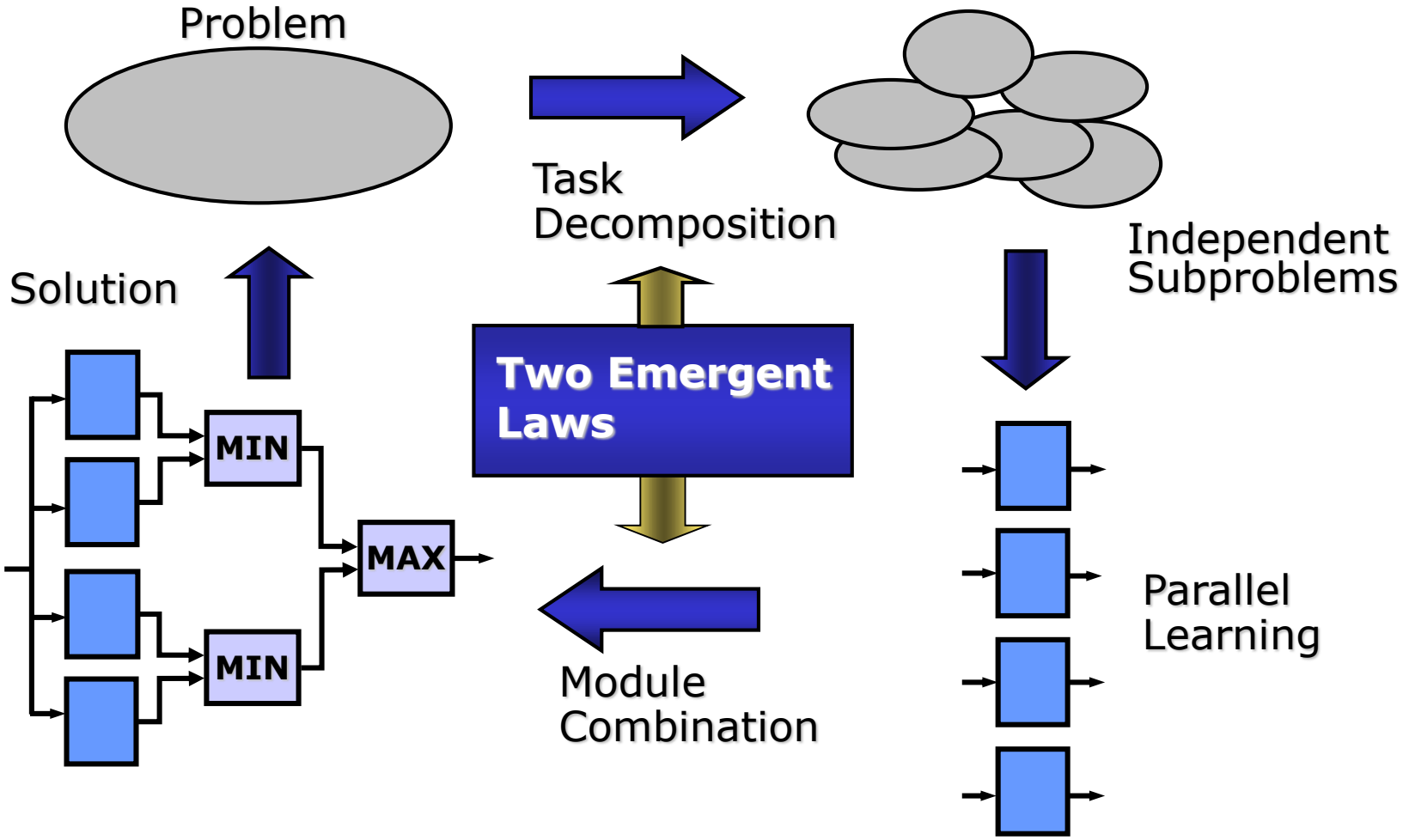


Figure 4.18: The updated two spirals problem (a) and the response plot of MSN_{uts}^{4} (b).

Min-Max Modular Neural Network

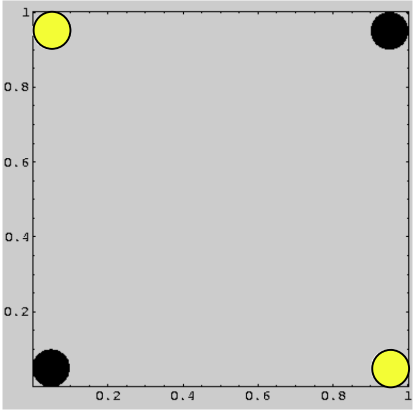
(B. L. Lu and M. Ito, 1997, 1999)

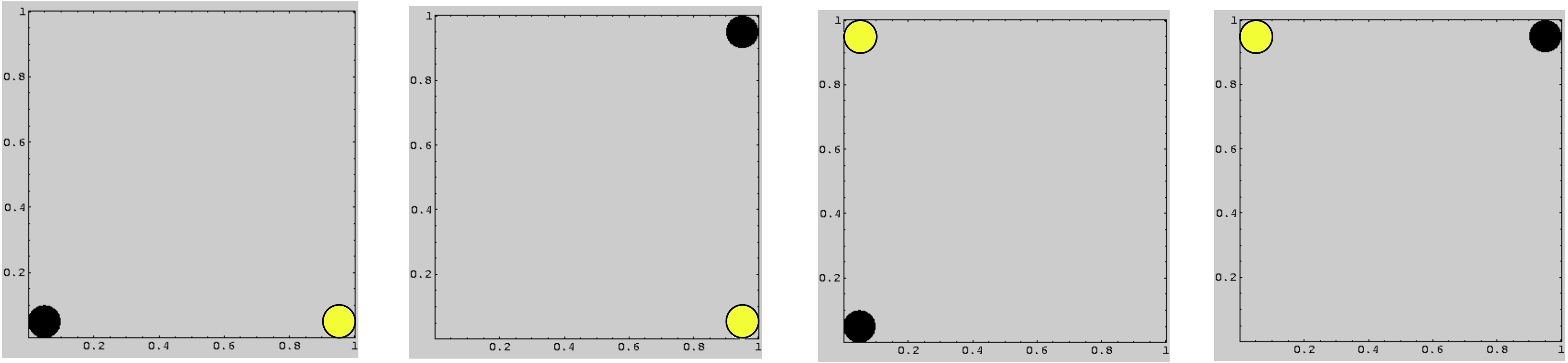
Emergent Learning in Batch Mode



Decomposition of the XOR Problem

XOR Problem:

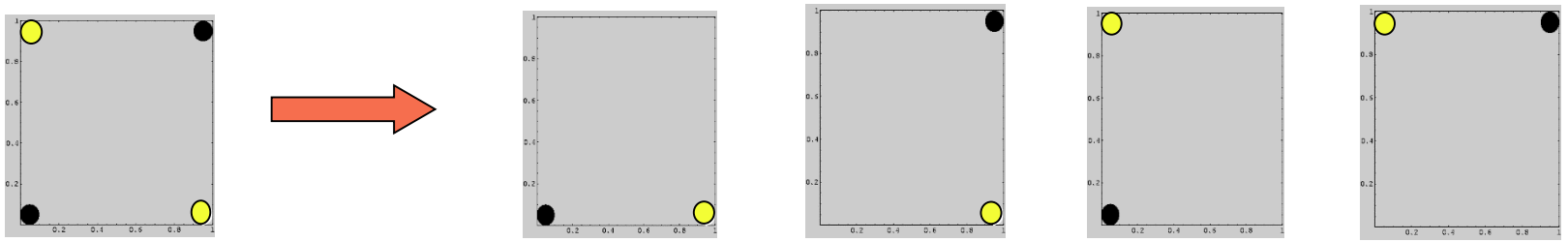


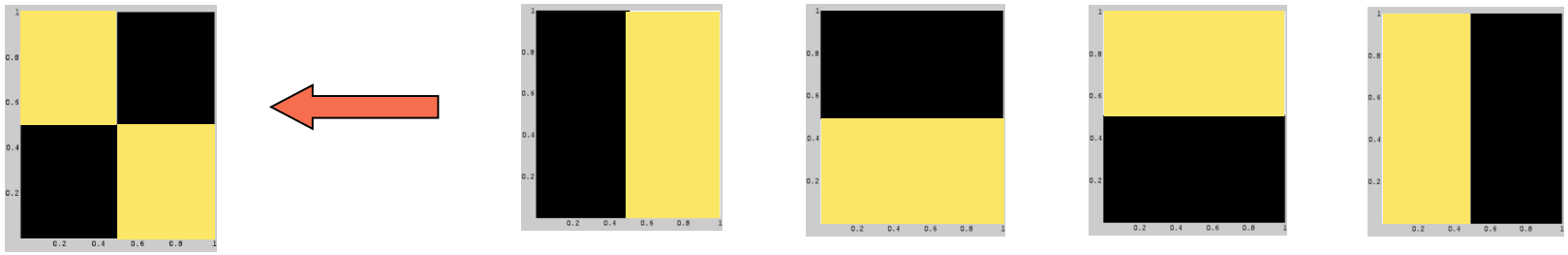


Four linearly separable problems

Decomposition *vs* Combination

Task Decomposition: Top-Down

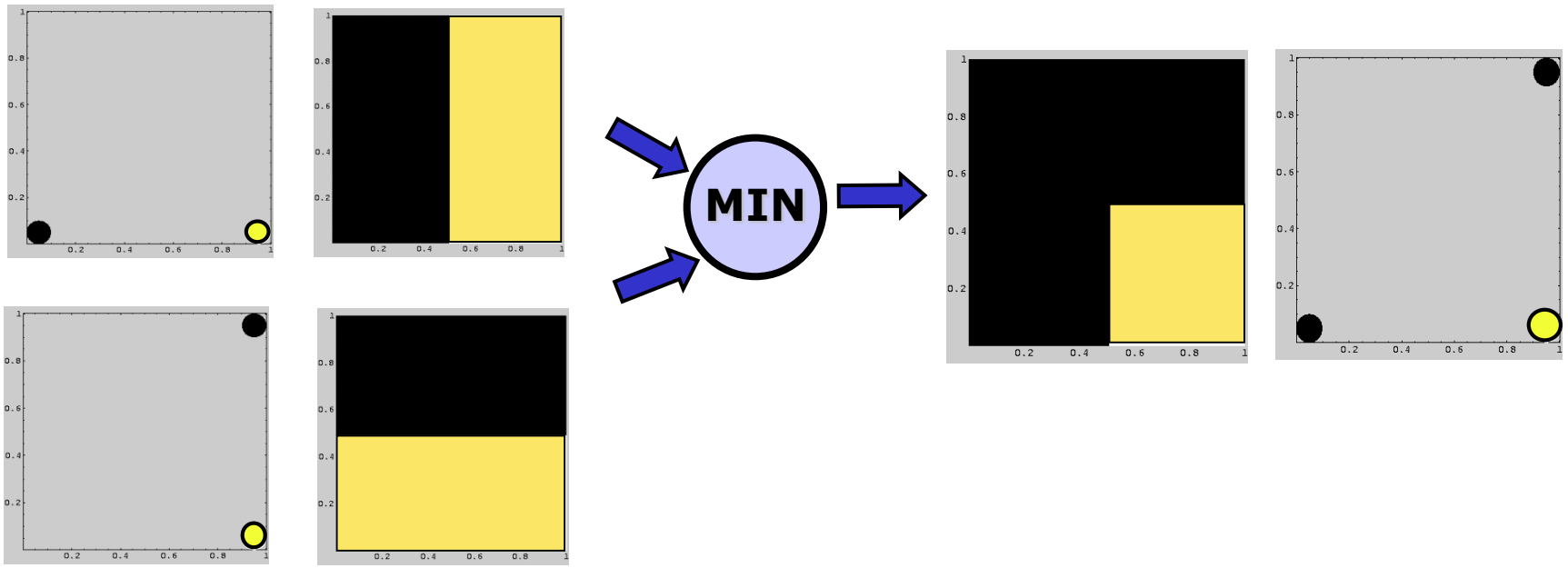




Module Combination: Bottom-Up

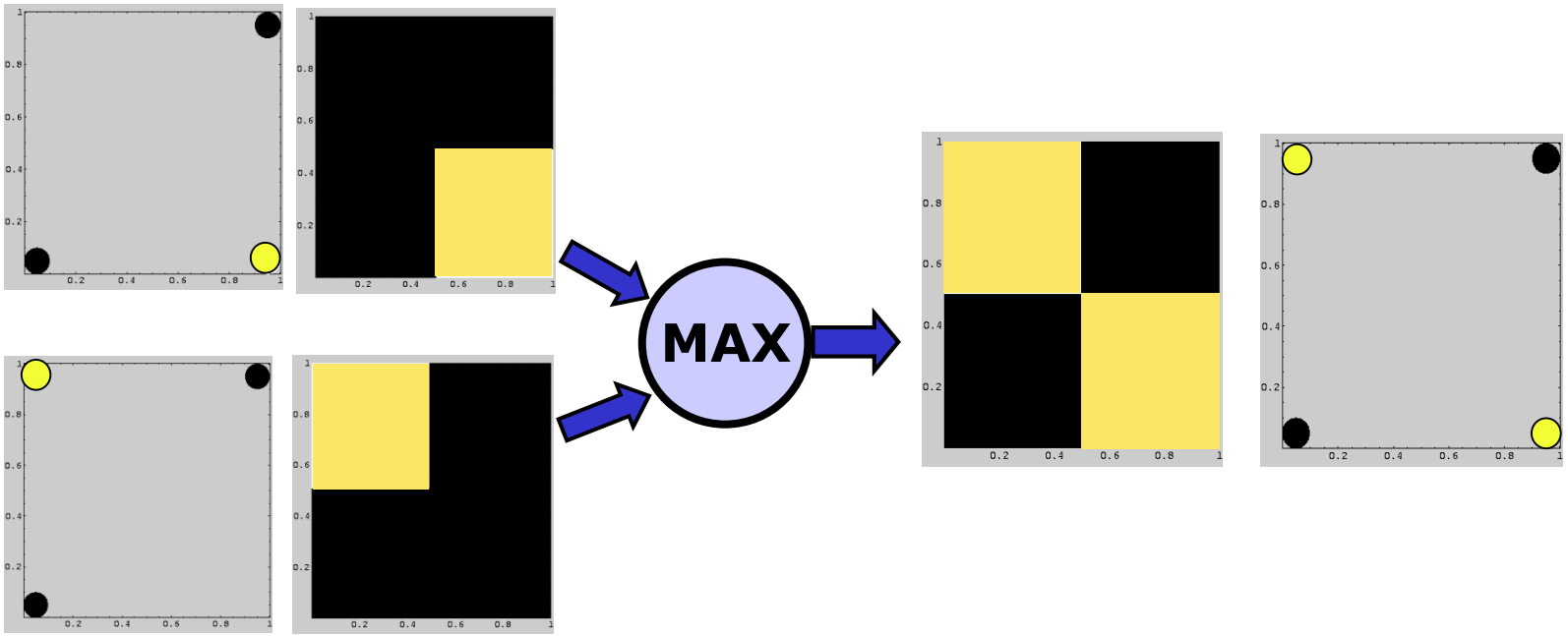
Emergent Law : Minimization

The modules, which were trained on the data sets which have the same training inputs corresponding to desired output “1”, should be integrated by the MIN unit.

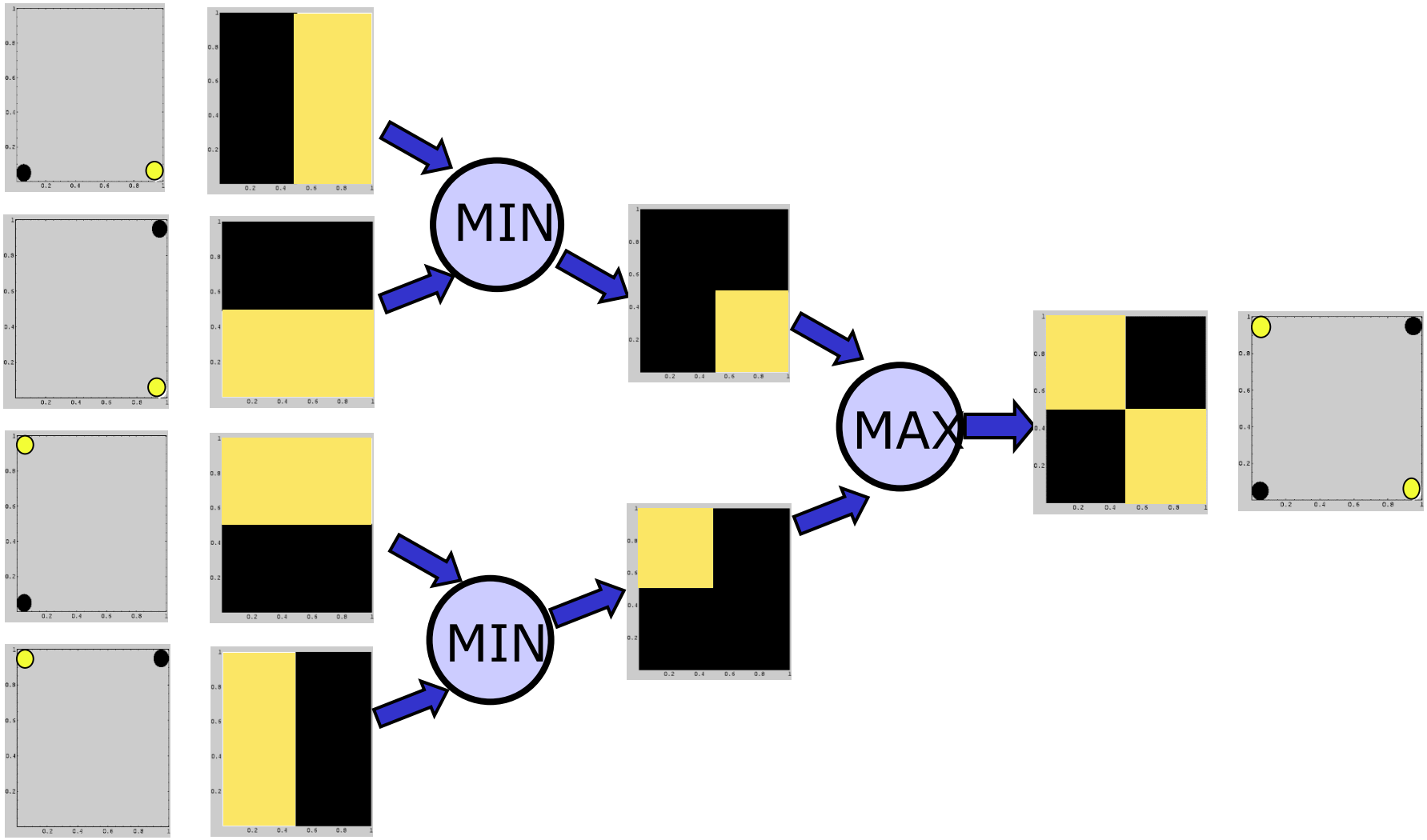


Emergent Law: Maximization

The modules, which were trained on the data sets which have the same training inputs corresponding to desired output "0", should be integrated by the MAX unit.

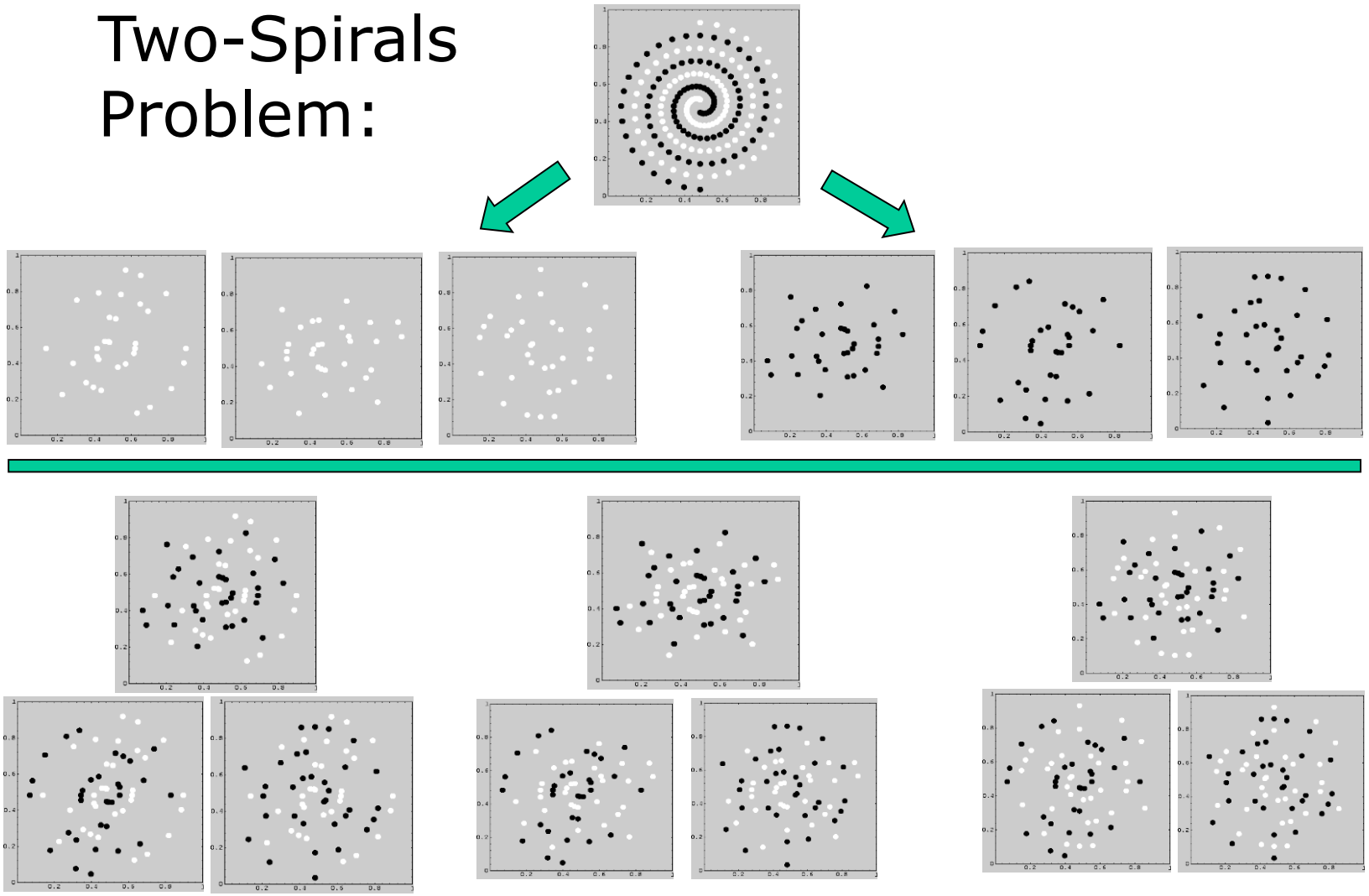


Emergent Learning of XOR problem

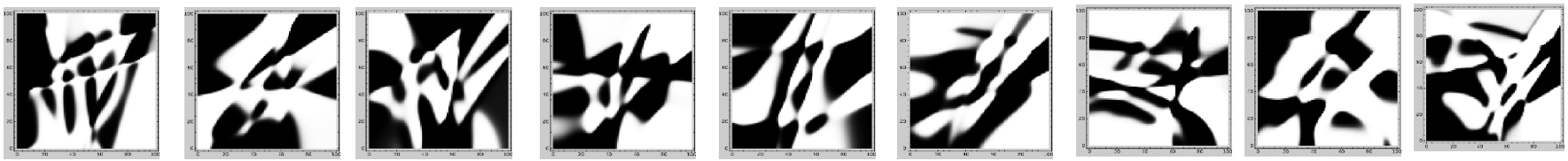


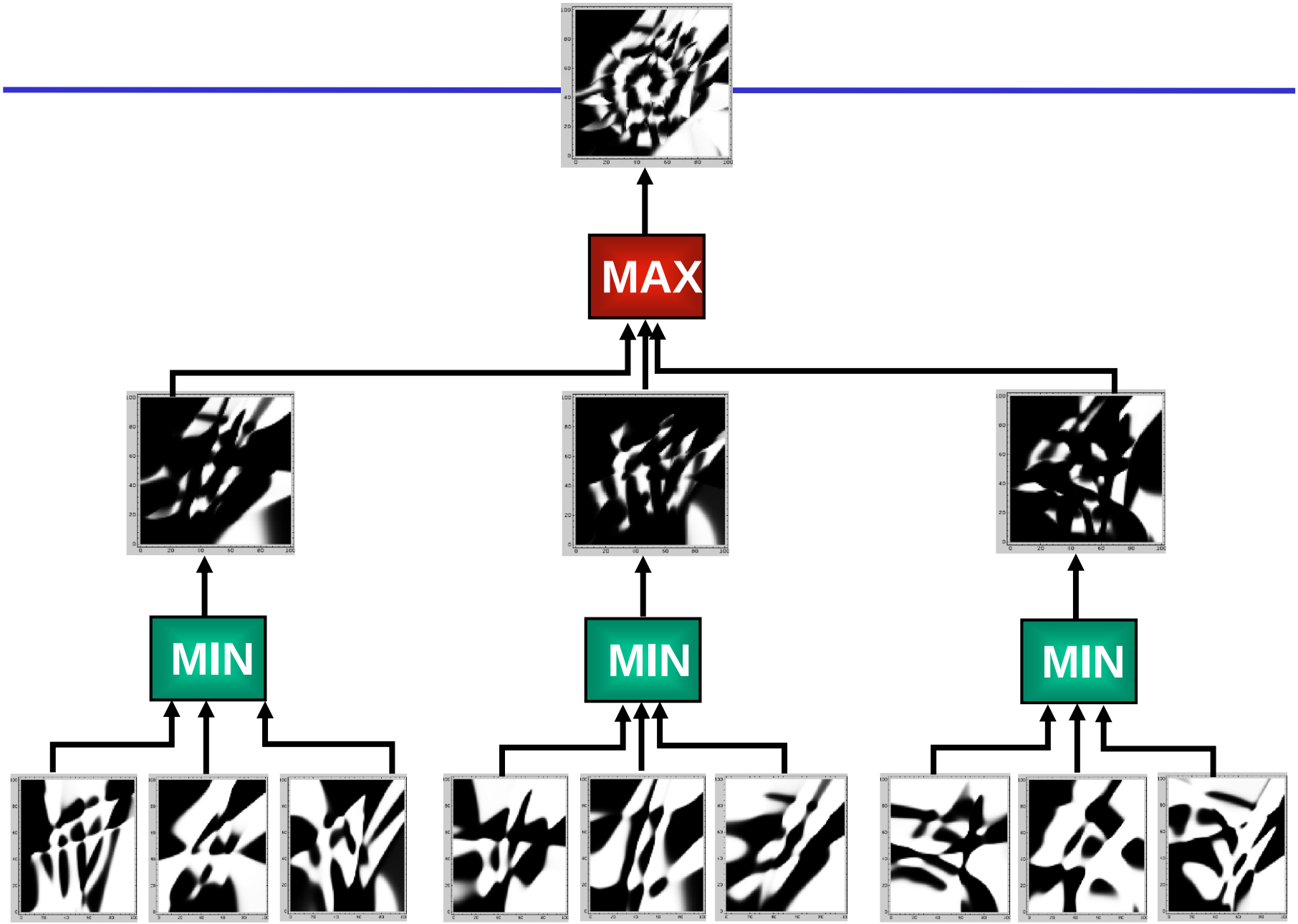
Decomposition of the Two-Spirals Problem

Two-Spirals Problem:

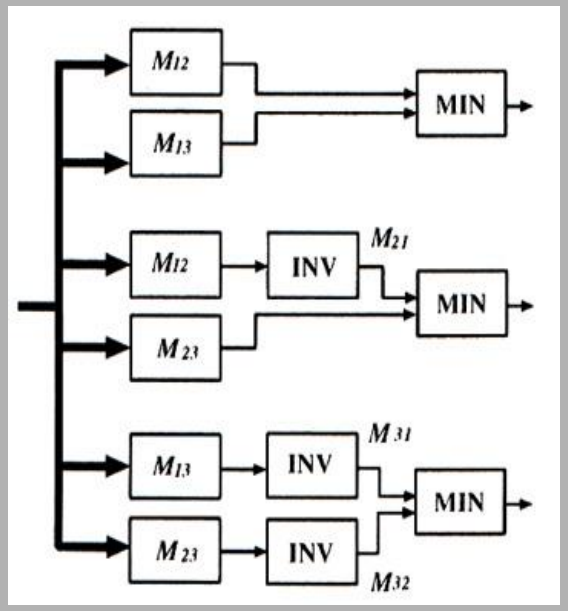


Massively Parallel Learning

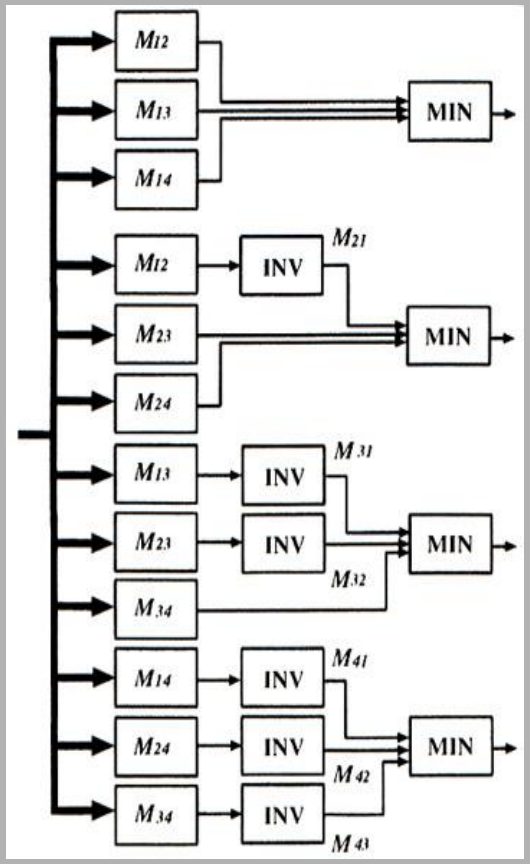



MAX
MIN
MIN
MIN

Min-Max Modular Neural Networks



Three-class



Four-class

Task Decomposition

- **Training data for a K-class problem**

$$T = \{ (X_l, Y_l) \}_{l=1}^{L}$$

- **Decompose a K-class problem into K(K-1)/2 two-class problems**

$$X_i = \{ X_l^{(i)} \}_{l=1}^{L_i} \quad \text{for } i = 1,2,\ldots,\mathrm{K}$$

$$T_{ij} = \{ (X_l^{(i)}, 1-\varepsilon) \}_{l=1}^{L_i} \cup \{ (X_l^{(j)}, \varepsilon) \}_{l=1}^{L_j} \text{ for } i = 1,\ldots K \text{ and } \quad j = i+1$$

- **Decompose a two-class problem into a number of relatively balanced two-class problems as smaller as needed**

$$\text{Partition} \quad \text{of} \quad X_i \quad \text{into} \quad N_i \quad \text{subsets} \quad X_{ij} = \{ X_l^{(ij)} \}_{l=1}^{L_i^{(j)}} \quad \text{for} \quad j = 1,\ldots,N_i$$

$$T_{ij}^{(u,v)} = \{ (X_l^{(iu)}, 1-\varepsilon) \}_{l=1}^{L_i^{(u)}} \cup \{ (X_l^{(jv)}, \varepsilon) \}_{l=1}^{L_j^{(v)}}$$

$$\text{for } u = 1,\ldots,N_i, v = 1,\ldots,N_j, \text{ and } \quad j \neq i$$

Number of Two-class Problems

- **Number of smaller two-class problems**

$$\sum_{i=1}^{K-1} \sum_{j=i+1}^{K} N_i \times N_j$$

N_i is the number of subsets for class C_i

- **Number of training data for each of the two-class s**

$$\lceil L_i / N_i \rceil + \lceil L_j / N_j \rceil$$

L_i is the number of training data for class C_i

Time Complexity Analysis

- **Empirical observation(T. Joachims, 2002):**

$$O((l^+ + l^-)^c) \qquad c \text{ is domain-specific.}$$

- **Time complexity of M³-SVM in a parallel way:**

$$O\left(\left(\left\lfloor \frac{l^+}{N^+} \right\rfloor + \left\lfloor \frac{l^-}{N^-} \right\rfloor\right)^c\right)$$

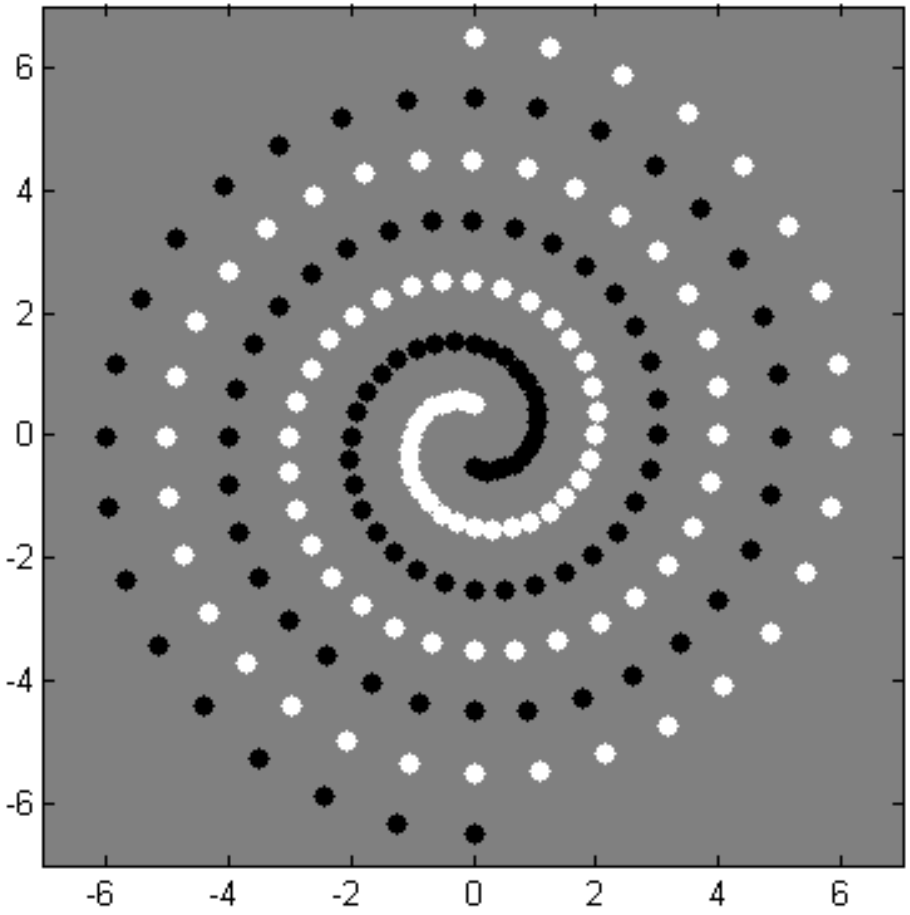
- **Time complexity of M³-SVM in a serial way:**

$$O\left(\frac{N^2}{N^c}\left(l^+ + l^-\right)^c\right) \qquad \text{suppose } N^+ = N^- = N$$

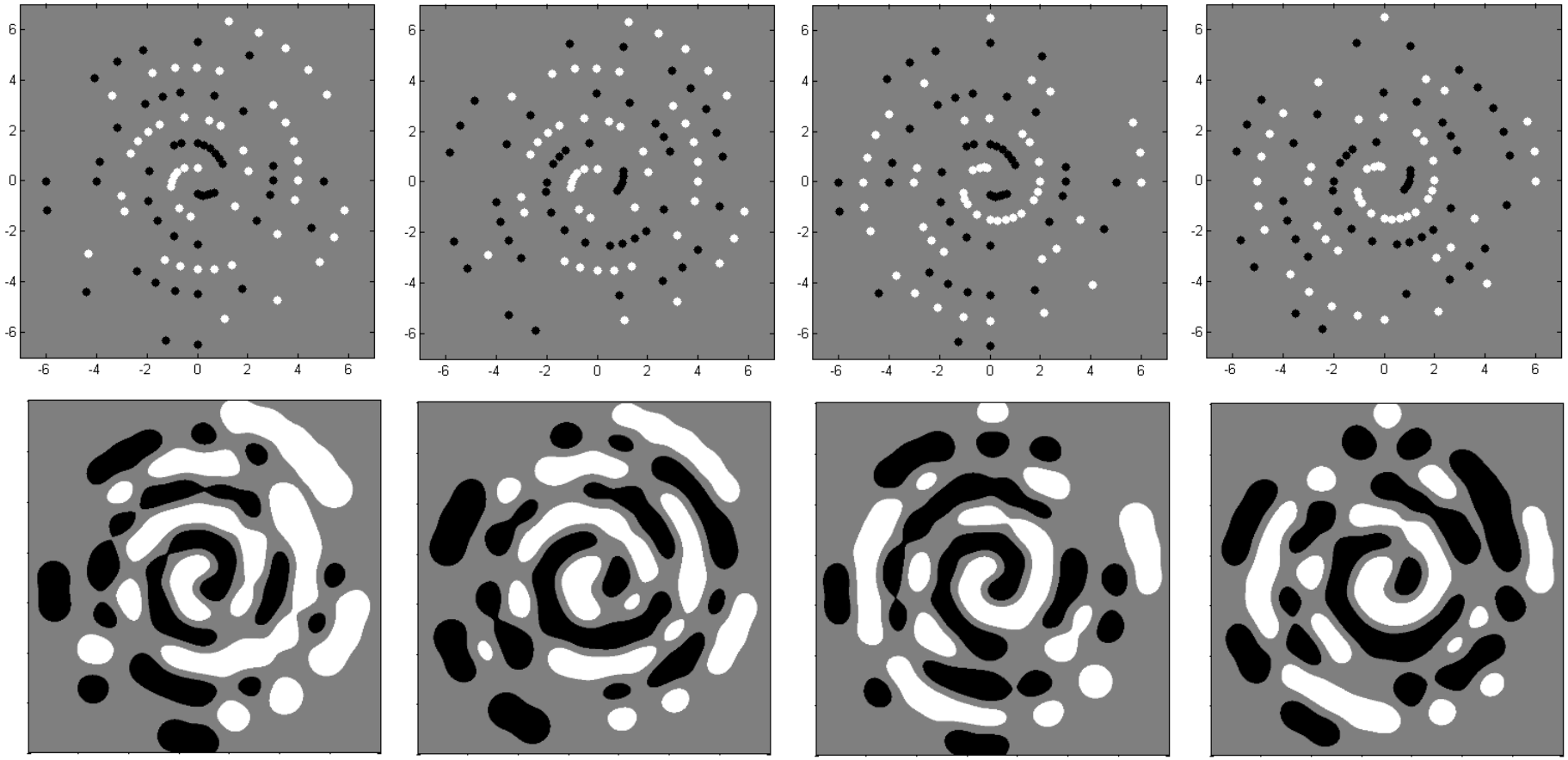
Task Decomposition Method

- How to divide training data set effectively is an important issue in M^3-SVM.
- Divide training data set randomly is a simple and straightforward approach.
- The geometric relation among the original data set may be damaged.

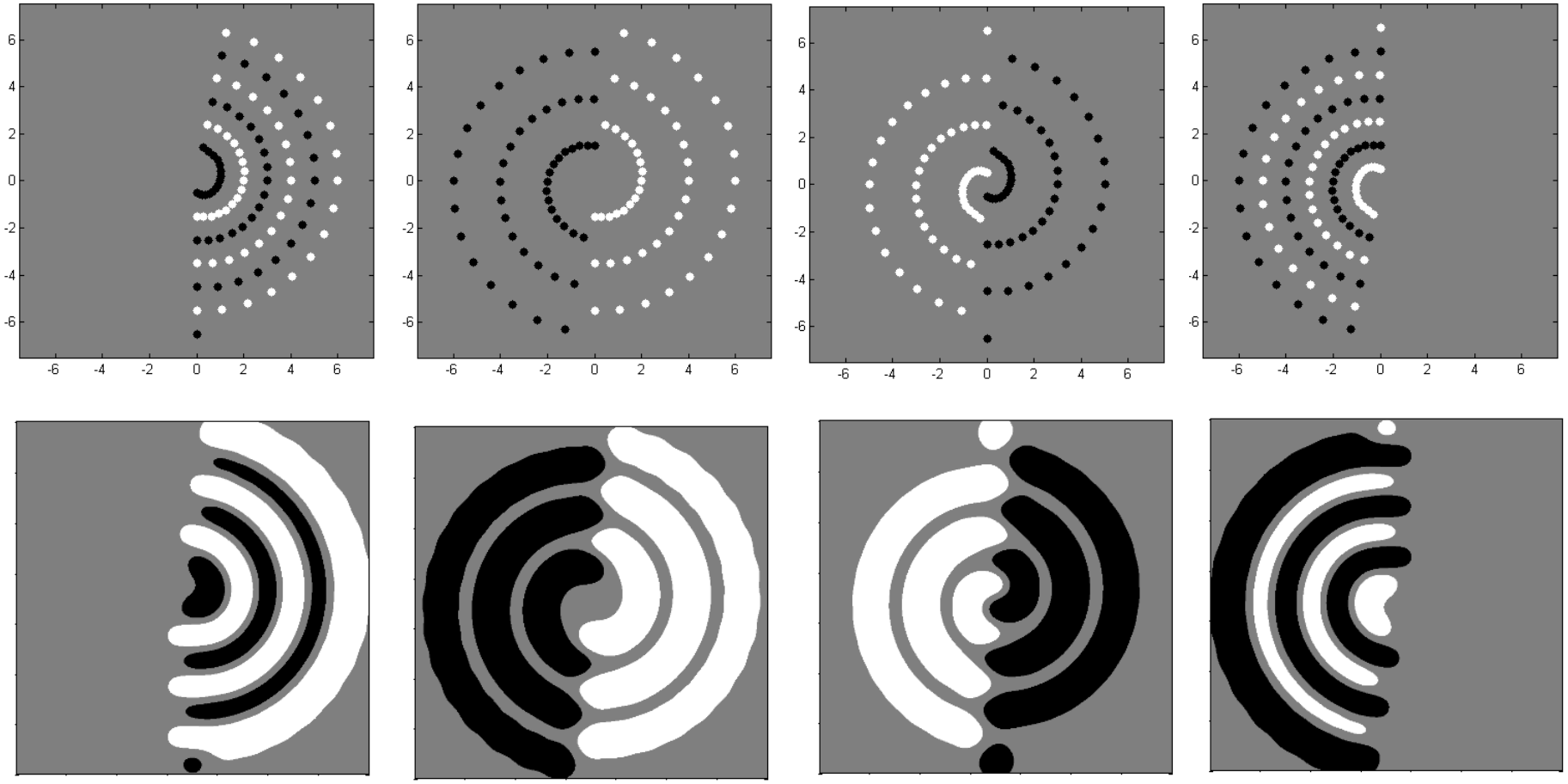
Two Spirals Problem



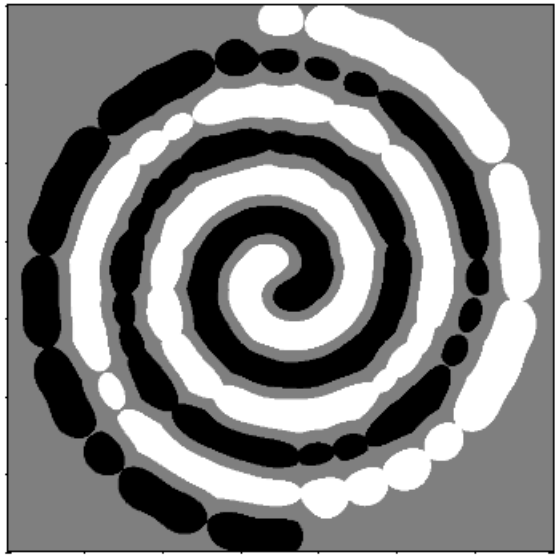
Random Partition



Y-axis partition



Boundaries of M3 Using Three Methods



random

Y-axis

Y-axis & overlap*

*overlap means two subsets share the training data along the y-axis

Hyperplane method

For more complicated high-dimensional problems, we divide the training data set using a hyperplane such as

$$a_1 z_1 + a_2 z_2, \ldots, a_n z_n = b$$

However, we never need to construct hyperplanes explicitly. Suppose we divide the training data set of class C_i to N_i subsets.

Hyperplane Method (cont.)

- Compute the distance between each training sample x of class Ci and basic hyperplane as follows,

$$H : a_1 z_1 + a_2 z_2, \ldots, a_n z_n = 0$$

$$dist(x, H) = \frac{a_1 x_1 + a_2 x_2, \ldots, a_n x_n}{\sqrt{a_1^2 + a_2^2, \ldots, a_n^2}}$$

Here, we take the direction of distance into account, the training data with positive value is located in the different side compared with the training data with negative value.

Hyperplane Method (cont.)

- Sort the training data according to the values of *dist*(x,H), namely, rearrange the training data according to their space position.
- Divide the reordered sequence of training data to N_i parts almost equally.

An Example of Hyperplane Method

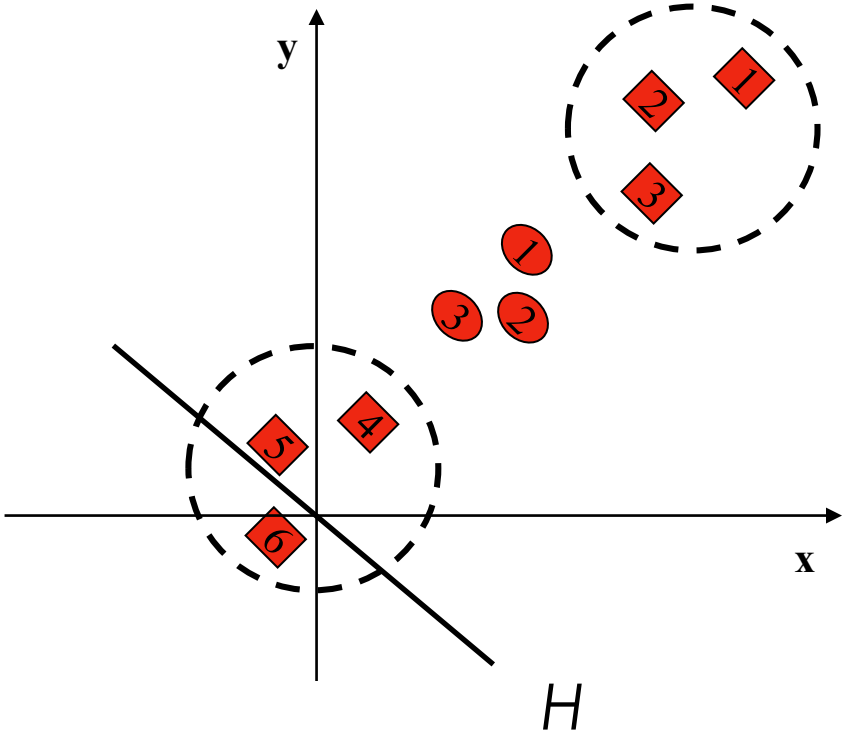
Here, we use hyperplane

$$H : z_1 + z_2, \ldots, z_n = 0$$

as basic hyperplane

We compute the distance and reordere the training data as the figure shows.

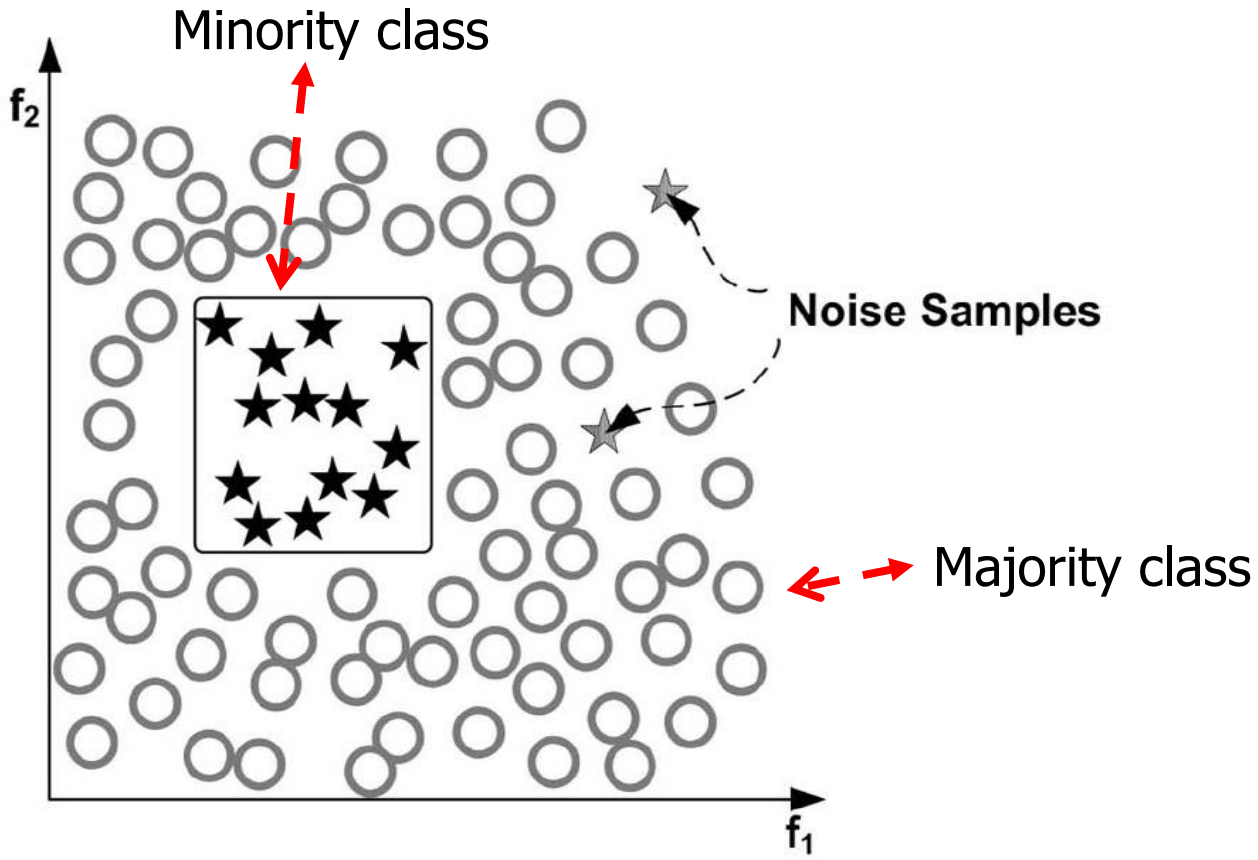
Then we divide the training data sets into two sets including 1,2,3 and 4,5,6 respectively.



Imbalanced data Set

- In a multi-class classification problem, any data set that exhibits an unequal distribution between its classes can be considered imbalanced
- In a two-class classification problem, a data set is said to present a class imbalance if it contains more training examples to one class (majority class) than the other (minority class).
- The data sets from real-world applications are usually imbalanced
- There are two kinds of imbalance forms:
  - Between-class imbalance
  - Within-class imbalance

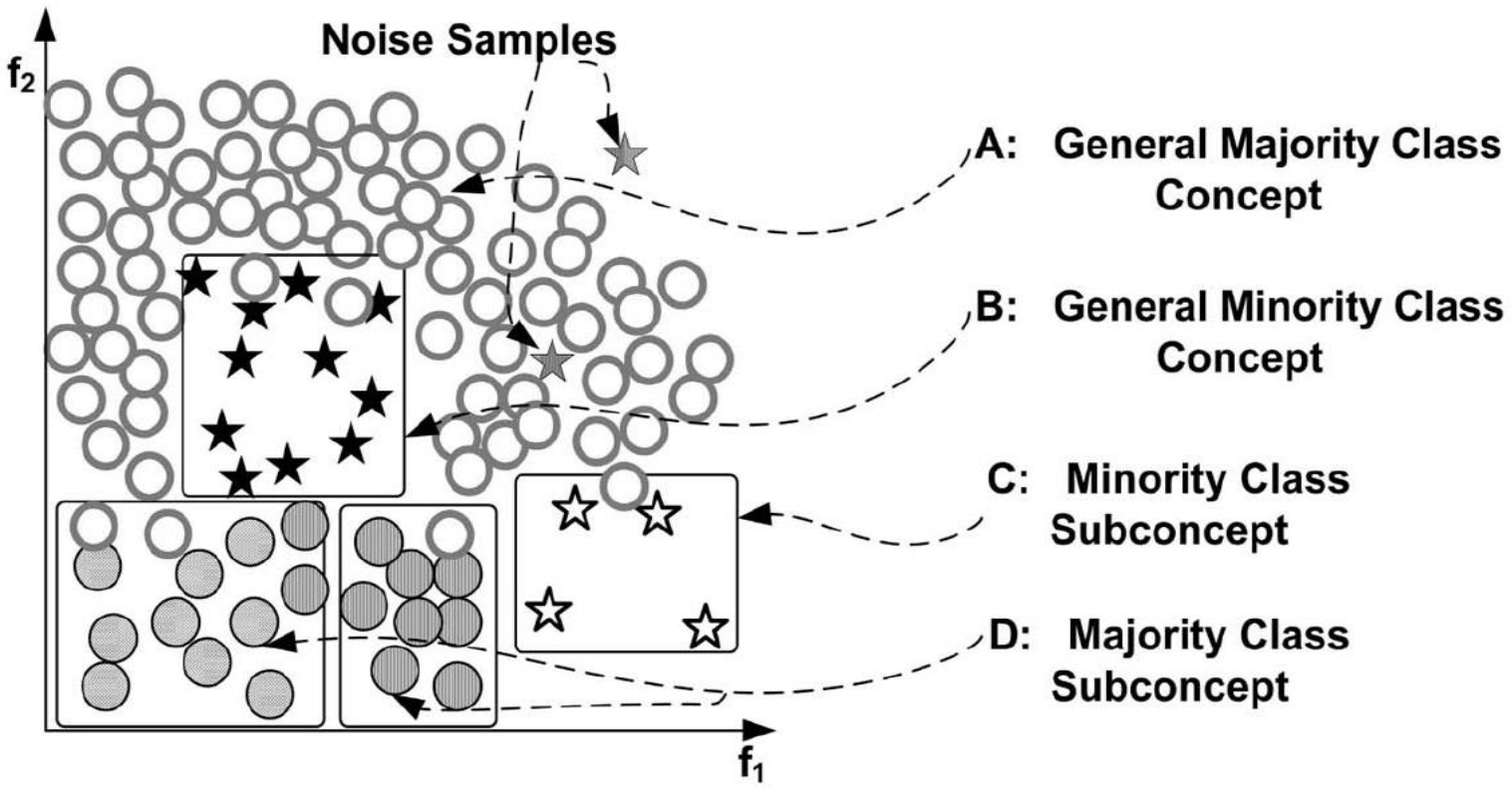
Between-class imbalance



The stars and circles represent the minority and majority classes, respectively

(He and Garcia, 2009)

Within-class imbalance



A high complexity data set with both between-class and within-class imbalances, multiple concepts, overlapping, noise, and lack of representative data.

(He and Garcia, 2009)

Reasons of leading to imbalances

- Rare instance: data for minority class are very limited, i.e., where the target concept is rare!
- Imbalances of this form are referred to as intrinsic
- Man-made imbalance: imbalanced data set is generated artificially
- Imbalances of this form are referred to as extrinsic
- Imbalances are caused by combination of intrinsic and extrinsic factors

Yeast Protein Sequences

Location	Sequence No.	Location	Sequence No.
Actin	29	Lipid particle	19
Bud	23	Microtubule	20
Bud neck	60	Mitochondrion	494
Cell periphery	106	Nuclear periphery	59
Cytoplasm	1576	Nucleolus	157
Early Golgi	51	Nucleus	1333
Endosome	43	Peroxisome	20
ER	272	Punctate composite	123
ER to Golgi	6	Spindle pole	58
Golgi	40	Vacuolar membrane	54
Late Golgi	37	vacuole	129

Reasons of leading to imbalances

- Rare instance: data for minority class are very limited, i.e., where the target concept is rare!
- Imbalances of this form are referred to as intrinsic
- Man-made imbalance: imbalanced data set is generated artificially
- Imbalances of this form are referred to as extrinsic
- Imbalances are caused by combination of intrinsic and extrinsic factors

Man-made imbalanced data sets

- One-versus-all (OVA)
  - K-class problem → K two-class sub-problems

$$X_i = \left\{ X_l^{(i)} \right\}_{l=1}^{L_i} \quad \text{for } i = 1,2,\ldots,\mathrm{K}$$

$$T_i = \left\{ (X_l^{(i)}, 1-\varepsilon) \right\}_{l=1}^{L_i} \bigcup \left(\bigcup_{j=1, j\neq i}^{K} \left\{ (X_l^{(j)}, \varepsilon) \right\}_{l=1}^{L_j} \right) \text{ for } i = 1,\ldots K$$

- After this task decomposition, the original balanced problem becomes an imbalanced one and imbalanced problems becomes serious imbalanced ones
- For example, the order of between-class imbalance is 1:588 (6:3530) for Yeast Protein Sequences problem

Problems of imbalanced learning

- The problem with class imbalance is that conventional classifiers are often biased towards the majority class
- Data from majority class are well-classified, whereas data from the minority class tend to be misclassified
- The reason is that standard classifiers attempt to reduce global quantities such as the error rate, not taking the data distribution into learning
- However, the most important knowledge usually resides in the rare cases

An example: Mommography Data Set

- A collection of images acquired from mammography exams
- Positive and negative samples represent cancerous patient and healthy patient, respectively
- The data set contains 10923 negative samples and 260 positive samples
- Classifiers have accuracy near to 100 percent on the majority class and accuracies of 0-10 percent on minority class
- Suppose a classifier achieves 10 percent accuracy on minority class
- As a result, 234 cancerous patients are classified into noncancerous patients

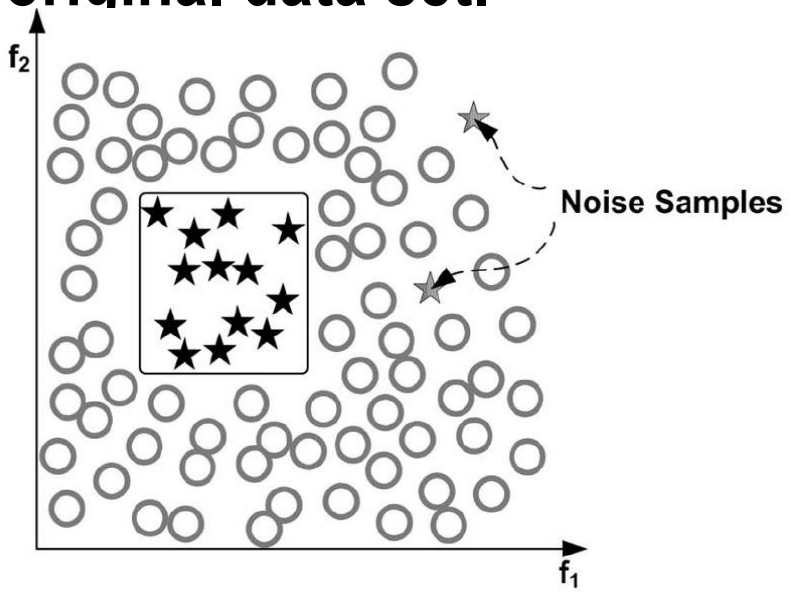
Existing Approaches

- Re-sampling Approach
  - Down-sampling
  - Over-sampling
- Cost-sensitive learning
- Active learning

Re-sampling Approach

Convert an imbalanced data set into a balanced one by manipulating the samples

- Under-sampling : to remove majority samples from the original data set
- Over-sampling : to add minority samples to the original data set.



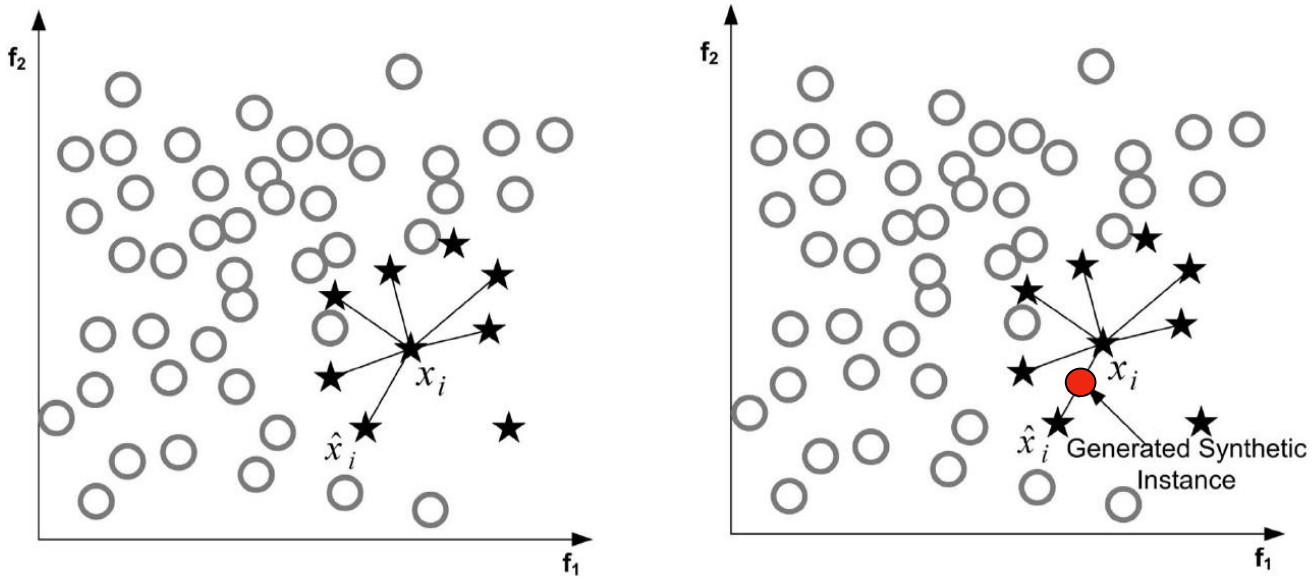
Re-sampling Approach: Down-Sampling

Make representative subsets of the majority class via data cleaning techniques

- the one-sided selection (OSS)
- the neighborhood cleaning rule (NCL)
- edited nearest neighbor (ENN)

Re-sampling Approach: Over-sampling

- **Synthetic Minority Over-sampling Technique (SMOTE): create artificial minority samples**



$$x_{new} = x_i + (\hat{x}_i - x_i) \times \delta$$

Cost-sensitive Learning

Associate high misclassifying cost to the minority classes

- Adacost: cost-sensitive weight updating strategy of Adaboost
- Cost-sensitive decision tree
- Cost-sensitive Bayesian classifiers

Active learning and one-class learning

- **Active Learning:**
  - Prefer the minority classes when interactively collecting new samples
- **One-class Learning:**
  - learn a class mainly through positive samples only
  - suitable for extremely imbalanced data sets

Remarks (I)

- Re-sampling Approach: manipulate the data
  - advantage: simplicity
  - disadvantage : under-sampling does not use the whole data set
- Cost-Sensitive Learning: adapt the classifiers
  - advantage: better performance
  - disadvantage : depend on the classifiers

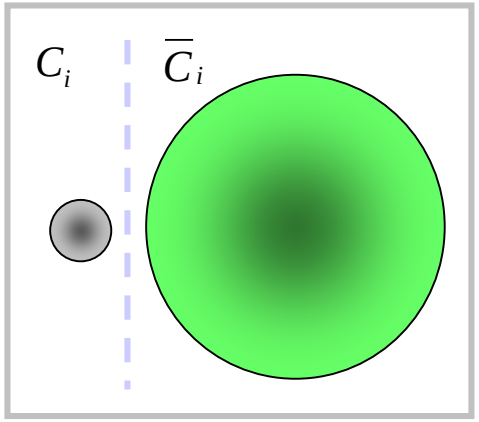
Remarks (II)

Active learning and one-class learning

- Almost unaffected by the sample ratio among classes
- Particularly suitable for extremely imbalanced problem where both re-sampling approaches and cost-sensitive learning fail

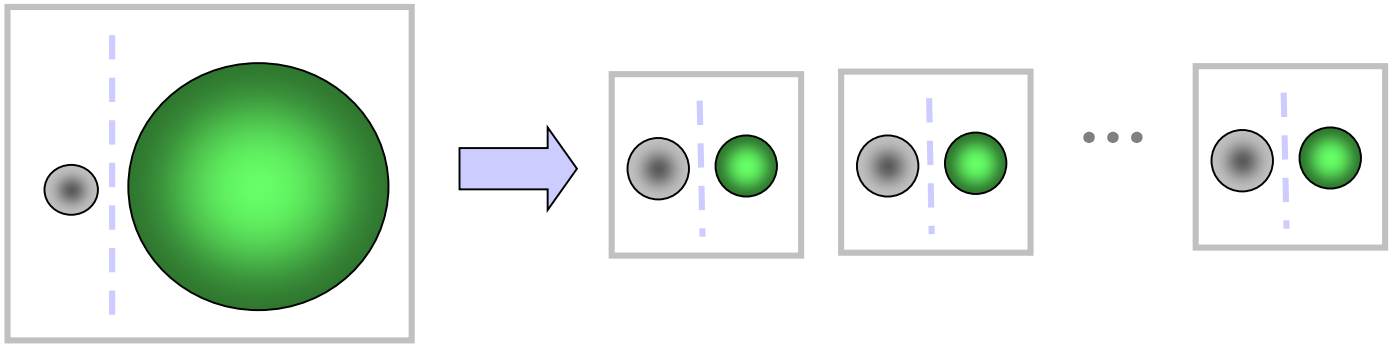
Main deficiencies of existing methods

- They are evaluated on small data sets
- Some of the two-class problems are very large and training classifiers on large-scale data sets are very time-consuming
- It is hard to incorporate prior knowledge into learning
- It is difficult to implement massively parallel learning

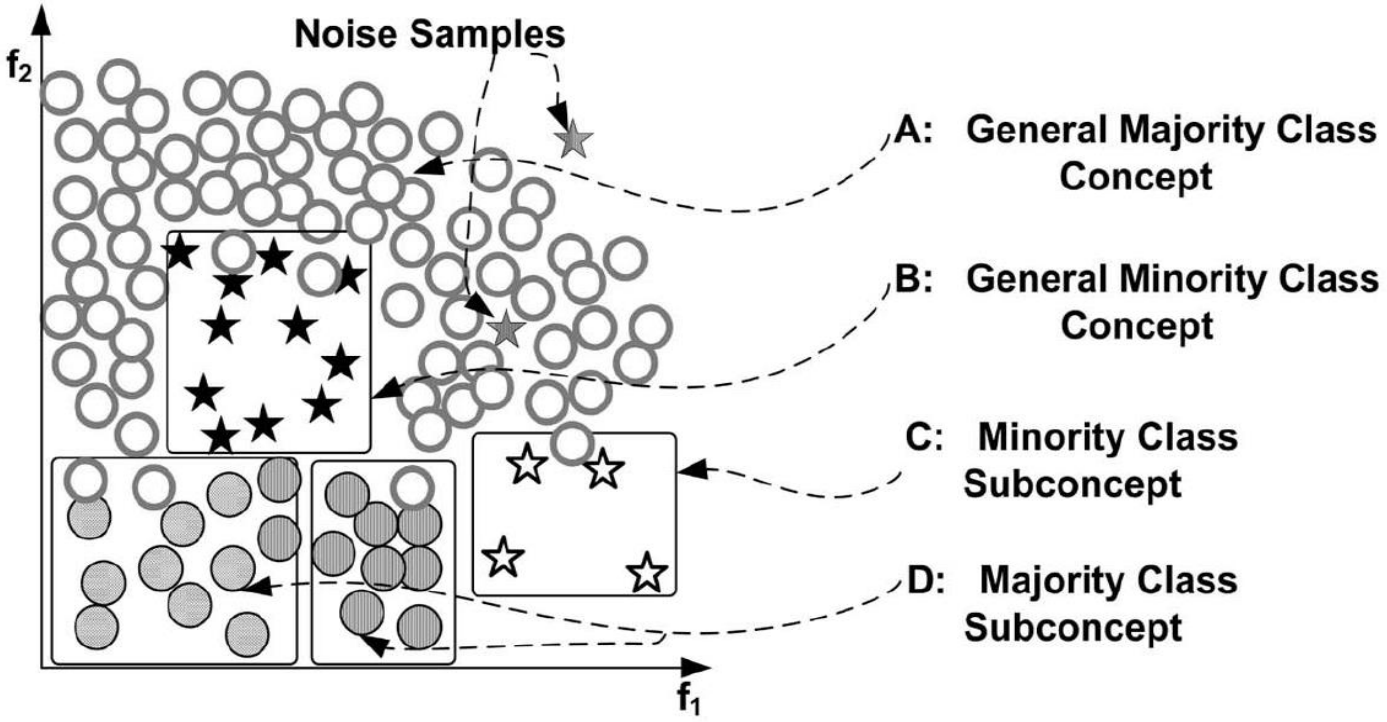


Basic idea of our approach

- Dispel imbalance by task decomposition instead of re-sampling
- Divide each of the imbalanced two-class problems into a number of relatively more balanced two-class subproblems with a part-versus-part task decomposition strategy
- Incorporate prior knowledge into learning by dividing training data



Why is task decomposition useful?



- A large-scale data set is organized following some rules
- These rules describes the data distribution in some degree
- Divide imbalanced data set using these rules as prior knowledge

第2次作业, 提交时间3月15日12:00

Suppose the output of each neuron in a multilayer quadratic perceptron (MLQP) network is

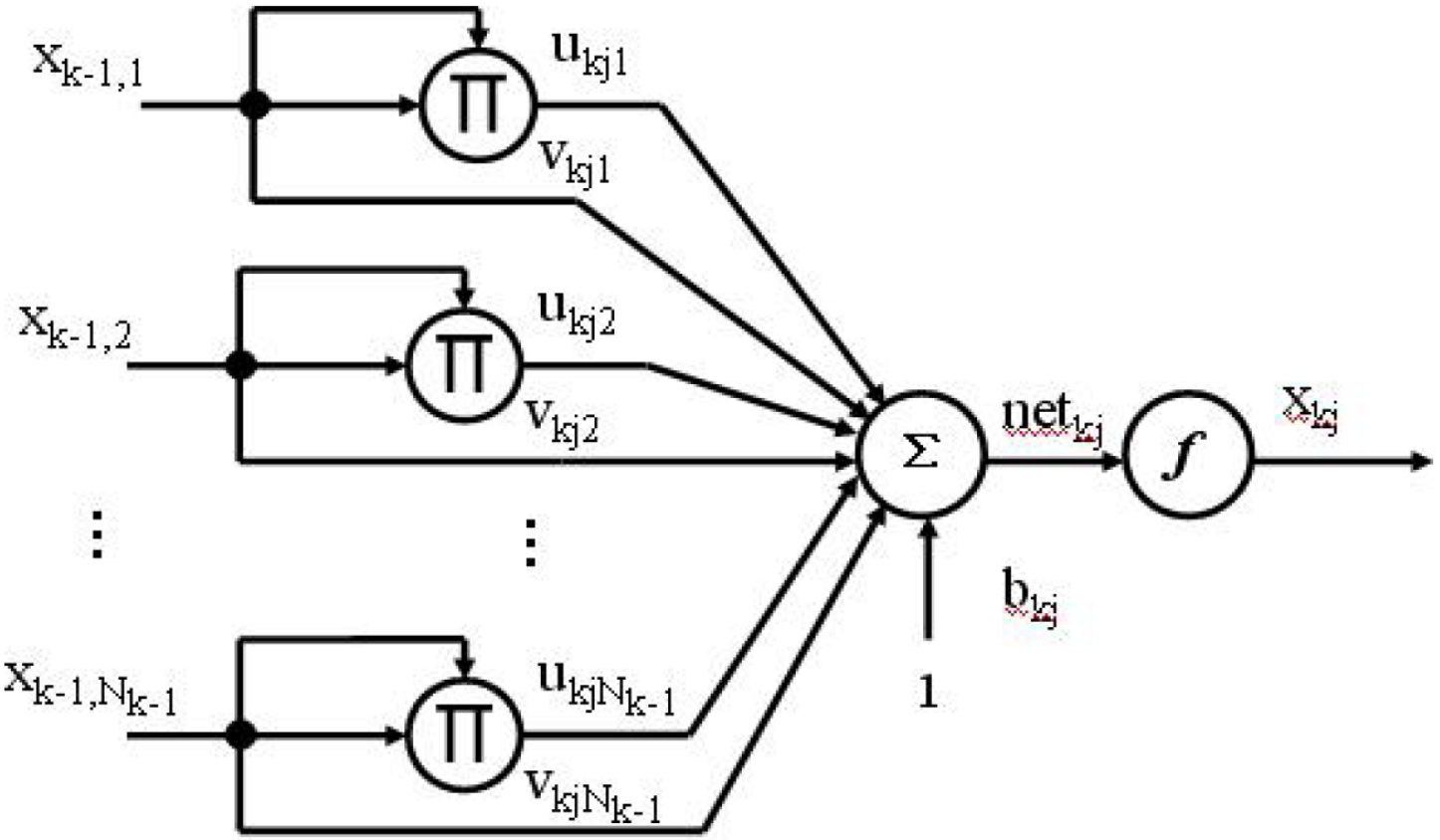
$$x_{kj} = f\left(\sum_{i=1}^{N_{k-1}} (u_{kji} x_{k-1,i}^2 + v_{kji} x_{k-1,i}) + b_{kj}\right) \tag{1}$$

$$\text{for } k = 2, 3, \cdots, M \text{ and } j = 1, 2, \cdots, N_k \tag{2}$$

where both u_{kji} and v_{kji} are the weights connecting the ith unit in the layer $k-1$ to the jth unit in the layer k, b_{kj} is the bias of the jth unit in the layer k, N_k is the number of units in the k $(1 \leq k \leq M)$, and $f(.)$ is the sigmoidal activation function.

This network is called multi-layer quadratic perceptron (MLQP).

第2次作业（2）:MLQP结构



第2次作业（3）：要求

1） 试推导训练上述MLQP的下列两种BP算法：

a)批量学习；b）在线学习。

2)编程实现问题1）中的训练MLQP的在线BP算法（编程语言不限，可采用常用的程序设计语言、Matlab或机器学习工具（Tensorflow等）

3）用含有一层中间层（中间层的神经元数目可设成10）的两层MLQP学习双螺旋问题，比较在三种不同学习率下网络的训练时间和决策面。

关于作业的要求：

1）严谨抄袭！

2）一旦发现，抄者和被抄者作业纪零分！

3）报告要规范书写、图清晰，不接收拍照的报告！

答疑QQ群

- 搜索QQ 群号加群： 812640820
- 或扫描二维码进群

工科创4J 答疑群

扫一扫二维码，加入该群。