Lecture 6: Multilayer Perceptron

Topics of this lecture

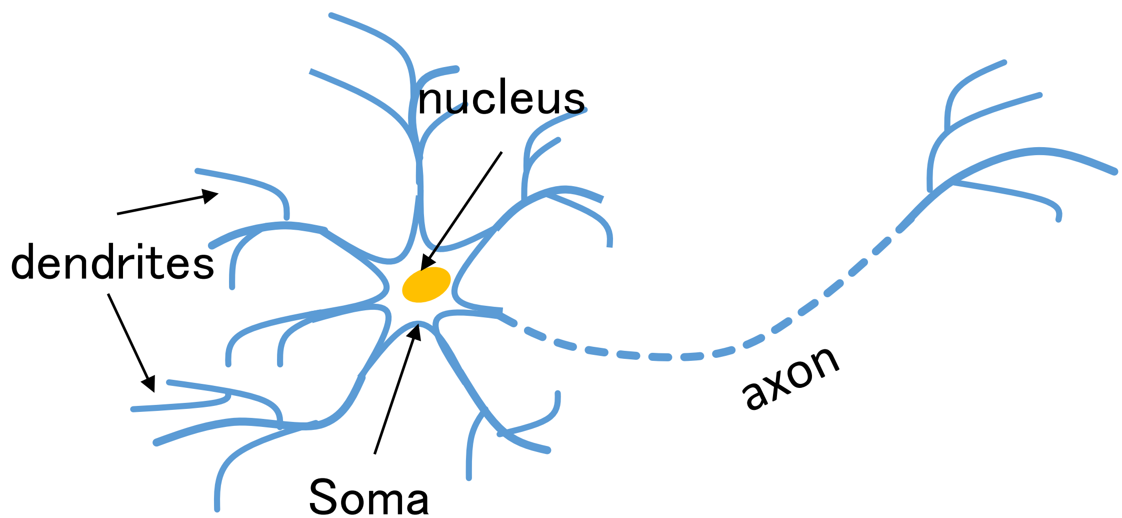
- About one neuron
  - How a bio-neuron works
  - Mathematic model of a neuron
  - The activation functions
  - Learning rule for a single neuron
- About multilayer perceptron (MLP)
  - Flow for making a decision
  - Meaning of the outputs
  - BP-based learning

What is a brain?

- Brain is the CPU for a human or animal that controls the whole body.
- Brain is a huge and complex network with approximately 10^{11} neurons and approximately 10^4 connections per neuron.
- Although the switching speed of each neuron is slow, the whole brain can make complex decisions quickly, and can even "think".

Mechanism of a bio-neuron

- A neuron (mainly) consists of a soma, many dendrites and an axon. Pulses from other neurons are input to the soma from the dendrites via synapses, and are integrated there. The neuron sends a pulse to other neurons through the axon when the potential of the soma is higher than a threshold.

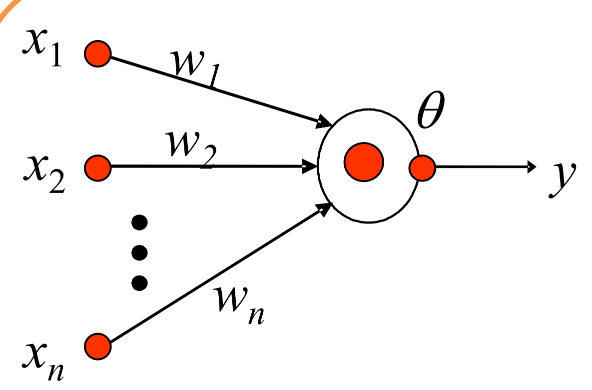


Mathematically model of a neuron

- Mathematically, a neuron is represented by

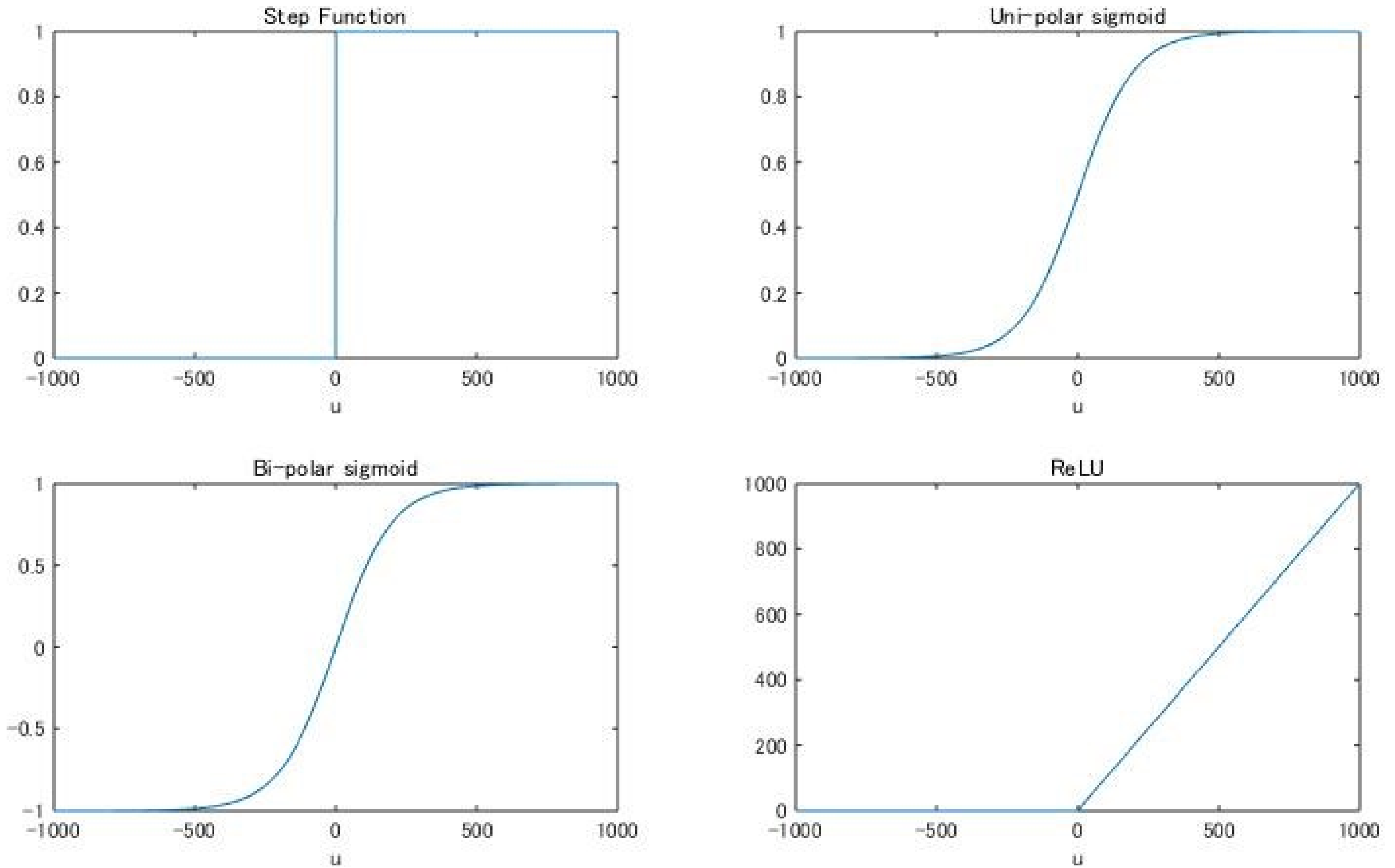
$$y = g(u) = g(\sum_{i=1}^{n} w_i x_i - \theta) = g(\sum_{i=1}^{n+1} w_i x_i) \qquad (1)$$

- where
  - $\boldsymbol{x} = (x_1, \ldots, x_n)^t$ is the input vector,
  - $\boldsymbol{w} = (w_1, \ldots, w_n)^t$ is the weight vector,
  - y is the output of the neuron,
  - θ is the bias or threshold,
  - $g()$ is an activation function.
- For convenience of discussion, the bias is usually considered a weight ($\theta = w_{n+1}, x_{n+1} \equiv -1$).

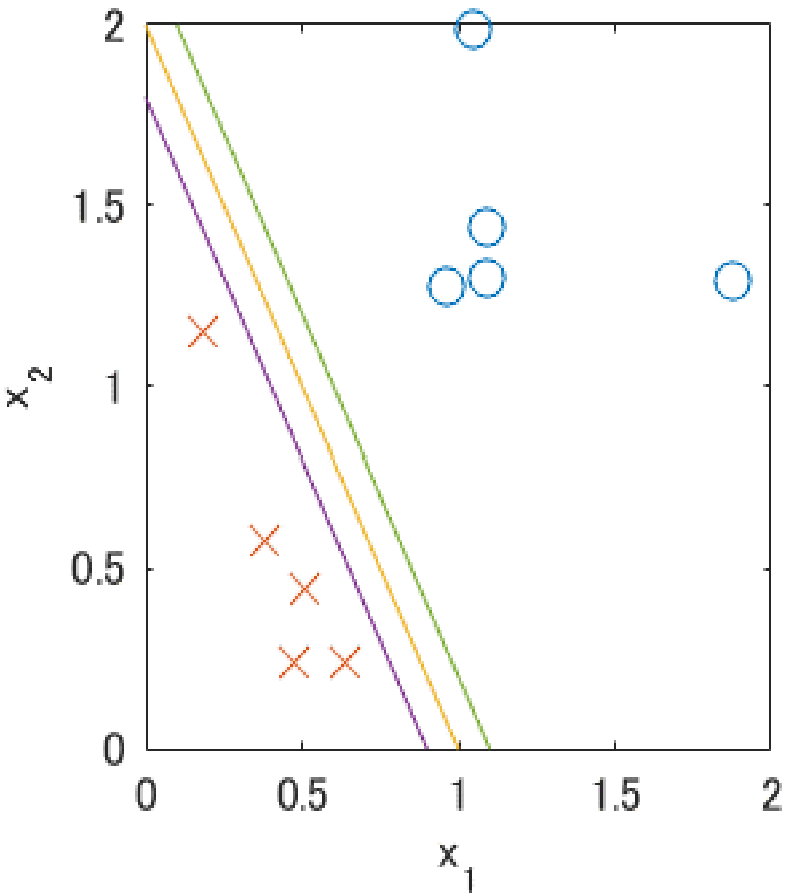


A neuron is a multi-input-single-output system.

Often used activation functions



Each neuron has a linear decision boundary



In the left example, there are two classes of patterns. The patterns can be separated by one line defined by

$$2x_1 + x_2 - 2 = 0$$

If we define w1=2, w2=1, and θ=2, this neuron can be used to classify the patterns correctly.

Learning rule for one neuron

- Given a training set, we can train a neuron using the following rule:

$$\boldsymbol{w}^{new} = \boldsymbol{w}^{old} + cr\boldsymbol{x} \qquad \textbf{(2)}$$

- where $\boldsymbol{x}$ is an example taken (randomly) from the training set, c is a learning constant or *learning rate*, and r is the *learning signal*. Note that the bias θ has been included in $\boldsymbol{w}$.
- Depends on how to choose r, we have different learning rules. For example
  - If $r = d - y$, we have the perceptron learning rule
  - If $r = (d - y) \cdot g'(u)$, we have the delta learning rule

Program for neuron learning

```
Initialization()
while Error>desired_error:
   Error=0
   for x in Training_Set:
      y=FindOutput(x)
      Error+=0.5*pow(d[x]-y, 2.0)
      r=Learning_Signal(d[x],y)
      for i in n+1:
         w[i]=w[i]+r*c*x[i]
```

- Where Learning_Signal is a method and n is the dimension of the feature space.

Remarks

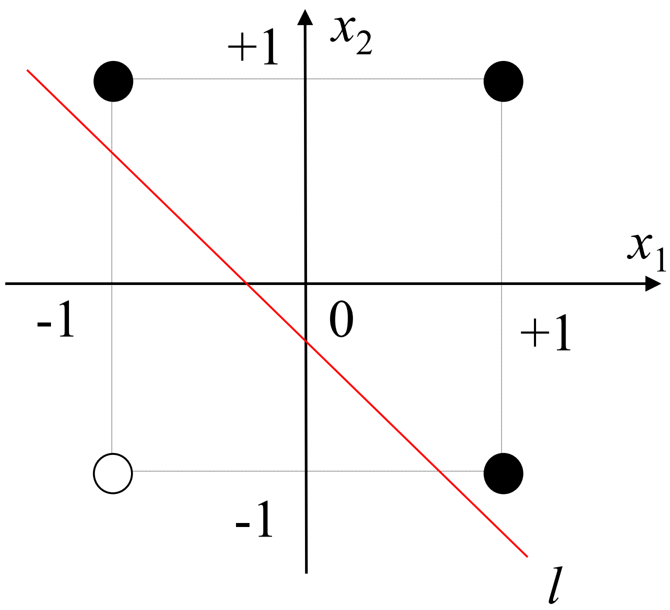
- During learning, each datum is usually selected one by one from the training set Ω.
- Training a neuron on each datum of Ω once is an epoch or a learning cycle.
- In each epoch, the error is calculated as follows:

$$E = \frac{1}{2}\sum_{x\in\Omega}(d[x] - y[x])^2 \qquad (3)$$

- We may select a "block" of data each time to increase the training efficiency. This block of data is often called a **mini-batch**.

Example - 1

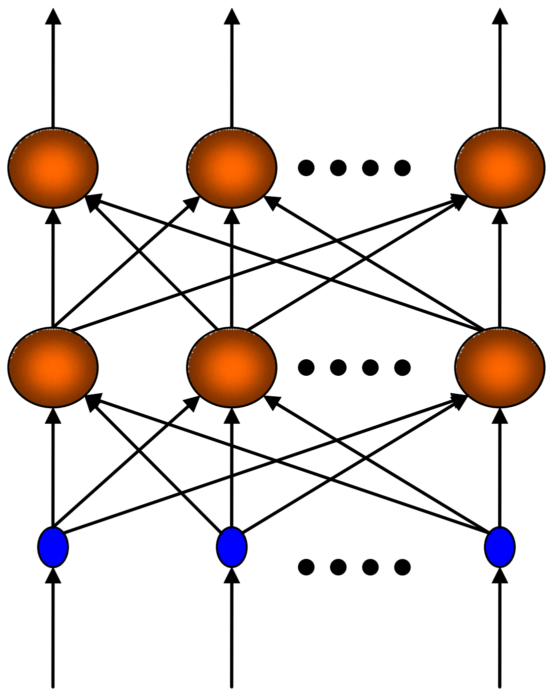
- Four patterns are given by (1,1),(1,-1),(-1,1) and (-1,-1).
- The corresponding teacher signals are 1, 1, 1, and -1.
- The problem is to find a neuron using the perceptron learning rule, to separate the four patterns into two classes.



Epochs	Connection weights	Number of errors
1	0.117988 0.106789 0.748825	1
2	1.117988 1.106789 -0.251175	0

Multilayer perceptron

- One of the most popular neural network model is the multilayer perceptron (MLP).
- In an MLP, neurons are arranged in *layers*. There is one *input layer*, one *output layer*, and *several (or many) hidden layers*.
- A three layer MLP is known as a general approximator (for solving any problem), but more layers can solve the problem more efficiently (using less neurons) and more effectively (more accurate).



This is a three layer MLP

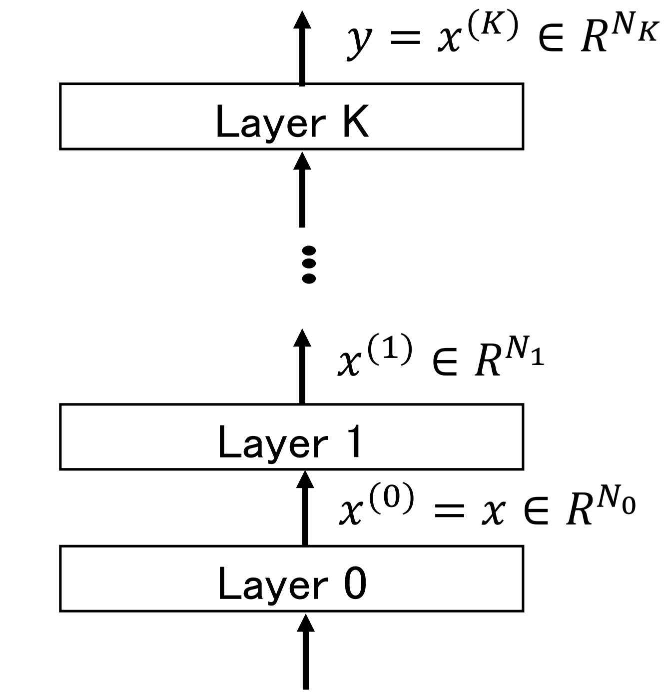
Flow of decision making using an MLP

- Suppose that $x^{(i)}$ is the output of the i-th layer, $i = 0,1,\dots,K$.
  - $x^{(0)} = x$: External input
  - $x^{(K)} = y$: The final output
- The output of the MLP can be found layer by layer (start from i=1) as follows:

$$u^{(i)} = W^{(i)}x^{(i-1)} - \theta^{(i)} \qquad (4)$$

$$x^{(i)} = g\left(u^{(i)}\right) \qquad (5)$$

- $u_j^{(i)}$ is called the *effective input* of the j-th neuron in the i-th layer.



N_i: Dimension of the i-th layer.

- N_0: Number of inputs.
- N_K: Number of classes.

Physical meaning of the outputs

- The activation functions of the hidden layers are usually the same. For the output layer, the following *softmax function* is often used for pattern classification.

$$y_n = x_n^{(K)} = \frac{\exp(u_n^{(K)})}{\sum_{m=1}^{N_K} \exp(u_m^{(K)})}, \quad n = 1,2,\dots,N_K \qquad (6)$$

- This output can be considered the *conditional probability* $p(C_n|\boldsymbol{x})$.
- That is, through training, the MLP can approximate the posterior probability. This MLP can be used as a set of discriminant functions for making decisions.

Objective function for training

- Suppose that we have a training set given by
$$\Omega = \{(\boldsymbol{x}_1, \boldsymbol{d}_1), \dots, (\boldsymbol{x}_N, \boldsymbol{d}_N)\}$$
- Neural network training is an optimization problem and the variables to optimize are the weights (including the biases).
- For regression, the following *squared error function* is often used as the objective function:

$$E(W) = \frac{1}{2}\sum_{n=1}^{N} \|\boldsymbol{d}_n - y(\boldsymbol{x}_n; W)\|^2 \qquad (7)$$

- For classification, the following *cross-entropy* is often used:

$$E(W) = -\sum_{n=1}^{N}\sum_{m=1}^{N_K} d_{nm} log\, y_m(x_n, W) \quad (8)$$

The gradient descent algorithm

- To find the weights based on the given training data, we usually use the following learning rule to update the weights iteratively (in a way similar to learning of a single neuron):

$$W^{new} = W^{old} - \epsilon \nabla E \quad (9)$$

- where ∇E is the gradient of E with respect to each element of W, and ϵ is the *learning rate*.
- To find the gradient of E is not easy because the weights of the hidden neurons are "embedded" in a complex function.
- To solve the problem, we usually use the well-known *back propagation (BP) algorithm.*

BP for squared error optimization

- For the weights in the output layer, we have

$$\frac{\partial E}{\partial w_{ji}^{(K)}} = \frac{\partial E}{\partial y_j}\frac{\partial y_j}{\partial u_j}\frac{\partial u_j}{\partial w_{ji}^{(K)}} = \left(y_j(\boldsymbol{x}) - d_j\right)g'(u_j)x_i^{(K-1)} = \delta_j^{(K)}\ x_i^{(K-1)} \quad (10)$$

- For the weights in the k-th hidden layer, we have

$$\frac{\partial E}{\partial w_{ji}^{(k)}} = \frac{\partial E}{\partial u_j^{(k)}}\frac{\partial u_j^{(k)}}{\partial w_{ji}^{(k)}} = \delta_j^{(k)} x_i^{(k-1)} \quad (11)$$

$$\delta_j^{(k)} = \frac{\partial E}{\partial u_j^{(k)}} = \sum_{n=1}^{N_{k+1}} \delta_n^{(k+1)}\left(w_{nj}^{(k+1)} g'(u_j^{(k)})\right) \quad (12)$$

- Using the gradient found above, we can update the weights layer-by-layer using Eq. (9).

BP for cross-entropy optimization

- If the outputs are calculated using the softmax function, the cross-entropy for any input x is

$$E = -\sum_{n=1}^{N_K} d_n \log y_n = -\sum_{n=1}^{N_K} d_n \log \frac{\exp(u_n^{(K)})}{\sum_{m=1}^{N_K} \exp(u_m^{(K)})} \qquad (13)$$

- For the output weights, we have

$$\delta_j^{(K)} = \frac{\partial E}{\partial u_j^{(K)}} = -\sum_{n=1}^{N_k} d_n \frac{1}{y_n} \frac{\partial y_n}{\partial u_j^{(K)}} = -d_j(1 - y_j) - \sum_{n \neq j} d_n(-y_j)$$

$$= -d_j + y_j \sum_{n=1}^{N_K} d_n = y_j - d_j \qquad (14)$$

- In the last line, we have used the property that only one output is 1 and thus the summation of all outputs is also 1.

BP for cross-entropy maximization

- Based on Eq. (14) we can find the error signal when the loss function is defined as the cross-entropy, and the error signals for other hidden layers can be obtained in the same way using equations (11) and (12).
- Therefore, the MLP for classification can be trained in the same way using the BP algorithm.
- For a classification problem, the desired outputs (or teacher signal) must be given as follows:
  - The number of outputs equals to the number of classes.
  - Only the output corresponding to the correct class is one, and all others are zeros.

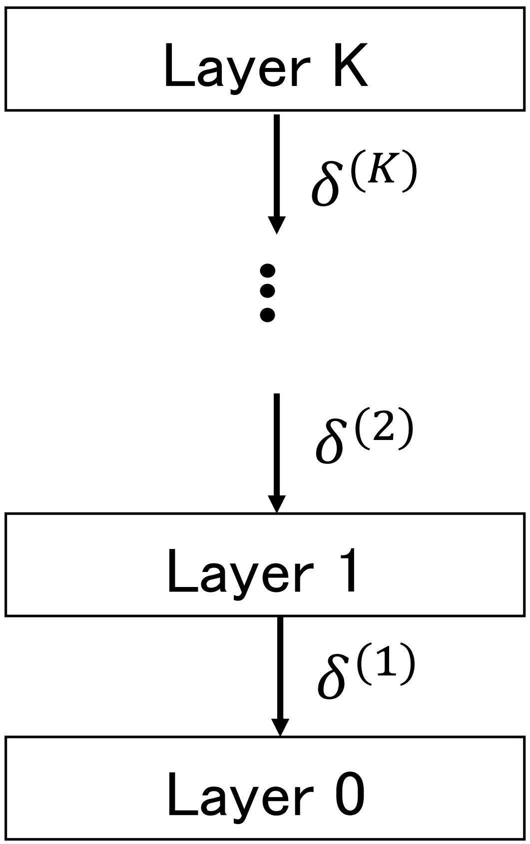
Summary of MLP learning

BP is also called the extended delta learning rule

- Step 1: Initialize the weights
- Step 2: Reset the total error
- Step 3: Forward propagation
  - Get a training example from the training set;
  - Calculate the outputs of all layers; and
  - Update the total error (for all data)
- Step 4: Back propagation
  - Calculate the "*delta*" for each layer, starting from the output layer.
  - Calculate the gradient using Eq. (10) and Eq. (11).
  - Update the weights using Eq. (9)
- Step 5: See if all data have been used. If NOT, return to Step 3.
- Step 6: See if the total error is smaller than a desired value. If NOT, reset the total error, and return to Step 2; otherwise, terminate.

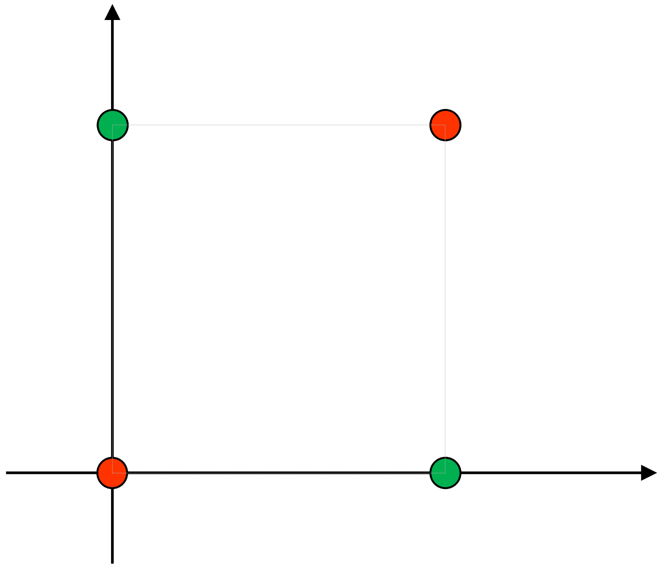
Meaning of error back propagation

- Start from the output layer, we can find the delta directly from the teacher signal and the network output.
- We can then find the delta for each hidden layer, from top to bottom.
- This delta is also called the *error signal* in the literature.
- This is why BP algorithm is also called *extended delta learning rule*.



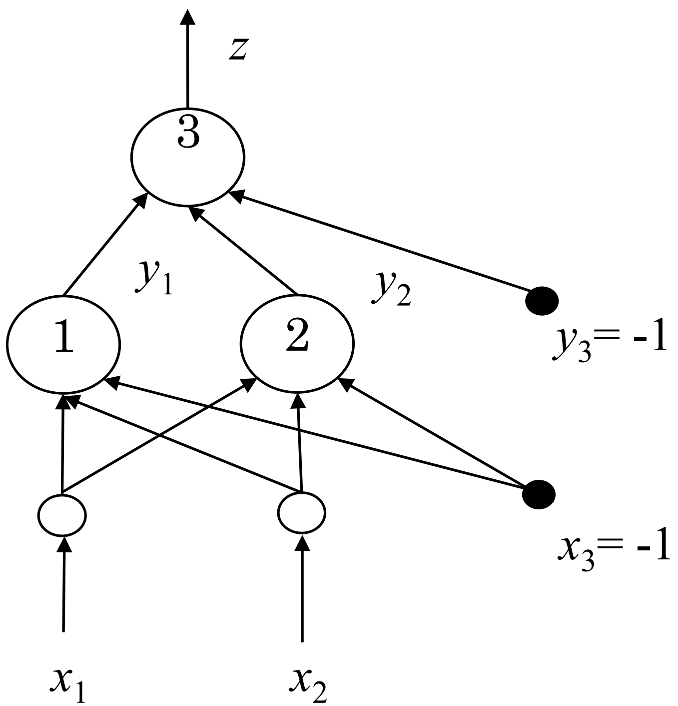
Example - 2

- Consider the XOR problem that classifies the vertices of a square into two classes.
- That is, (0,0) and (1,1) belong to the negative class; and (1,0) and (0,1) belong to the positive class.
- This is a simple problem, but cannot be solved by using a single layer perceptron.
- It was used by Dr. Minsky to show why "neural networks" are useless.



The network structure

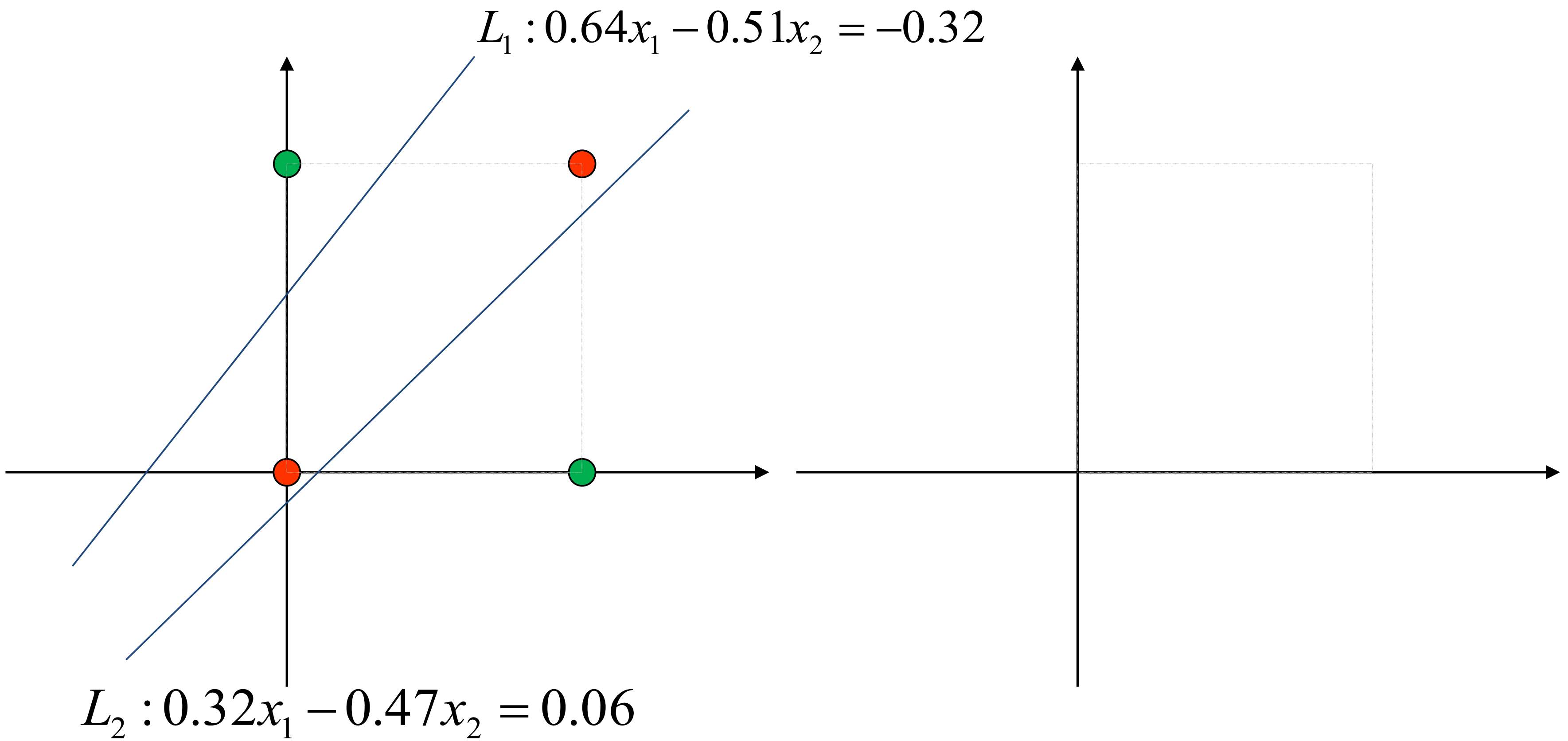
- The right figure shows the structure of the MLP.
- There are two input neurons, two hidden neurons, and one output neuron.
- The input neurons are just registers.
- The function to find is z=f(x1,x2), and a 3-layer perceptron is used to approximate f().



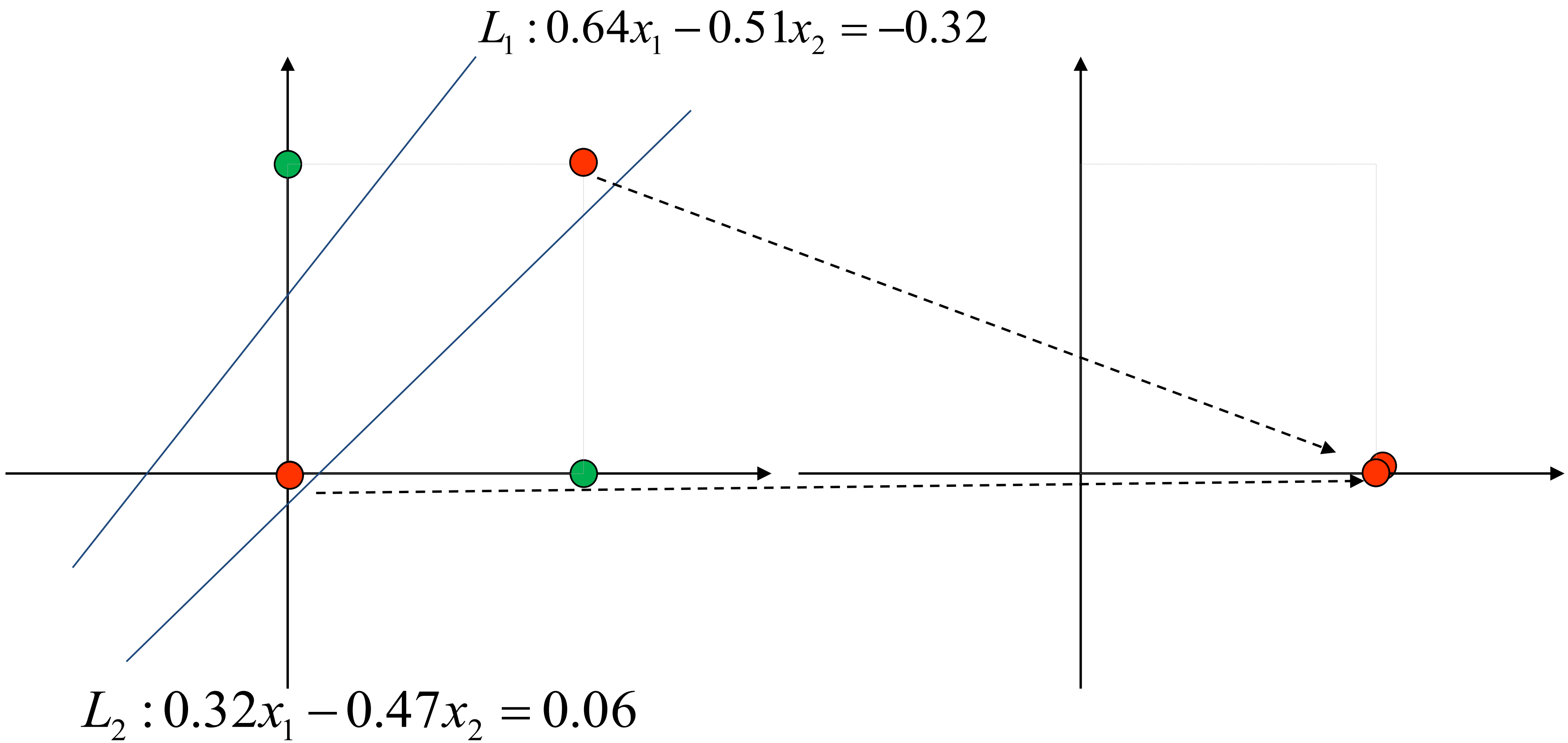
Results of BP

```
 .................
Error[1073]=0.001008
Error[1074]=0.001003
Error[1075]=0.000998

The connection weights in the output layer:
-0.839925 0.782646 -0.602153

The connection weights in the hidden layer:
0.638360 -0.509153 -0.316935
0.319389 -0.472554 0.068378
```

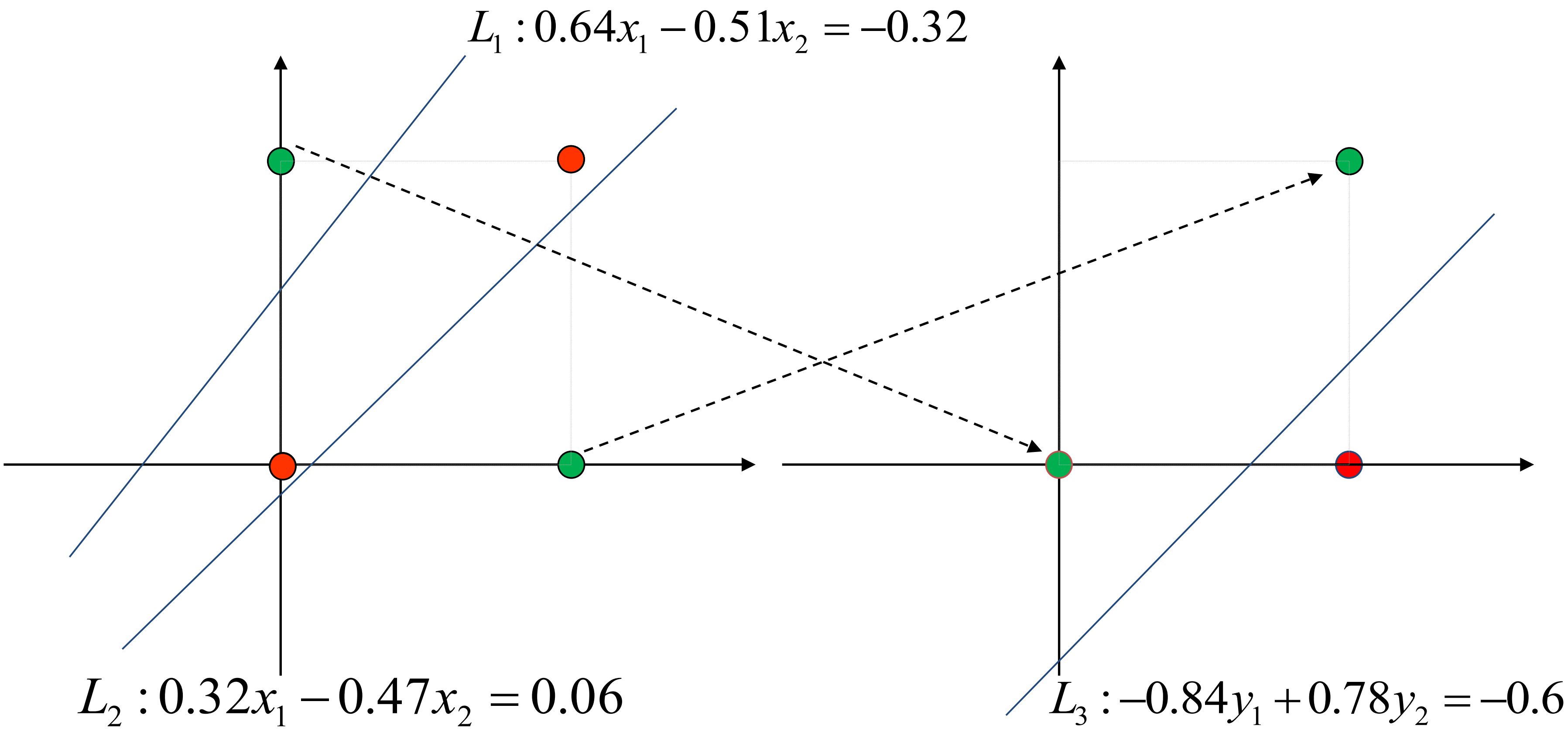
Physical meaning of the result



Physical meaning of the result



Physical meaning of the result

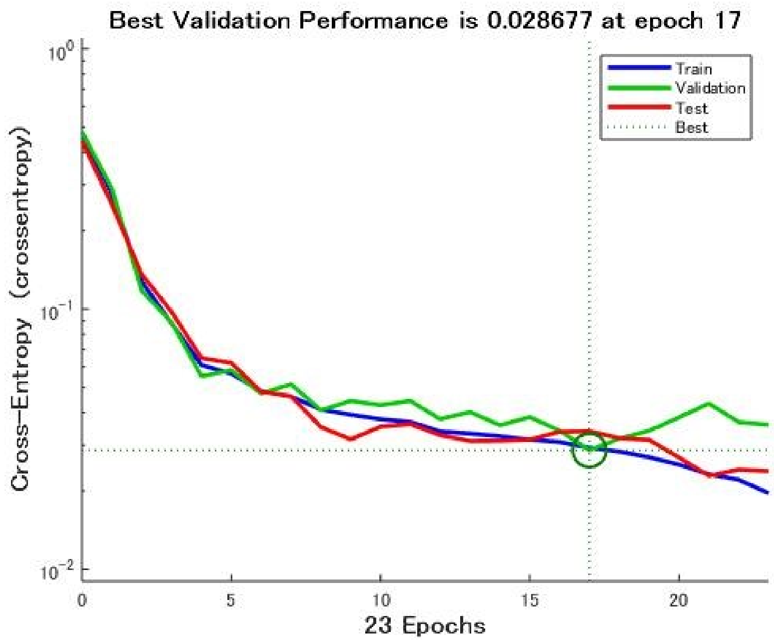


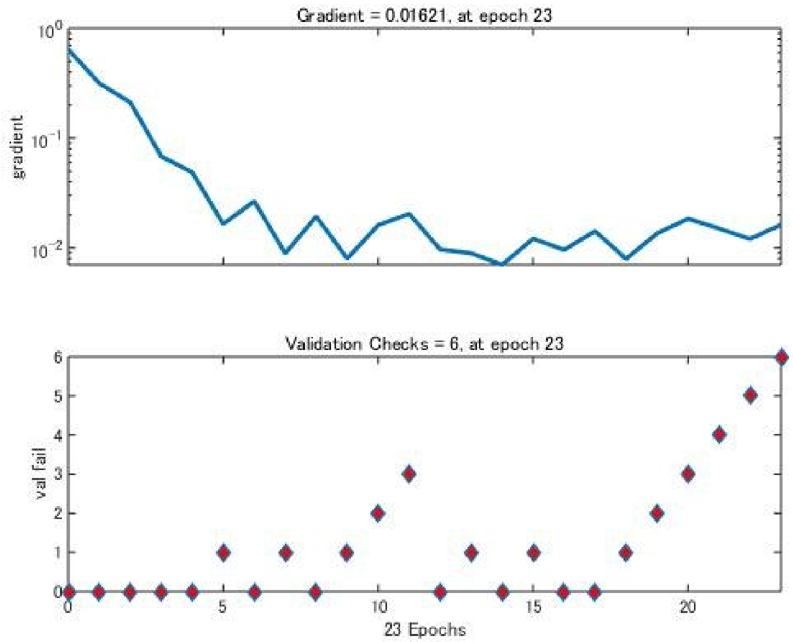
Example - 3

- In this example, we try the **Classify Patterns with a Shallow Neural Network in Matlab**.
- The database used is the cancer dataset.
  - There are 699 data, and each datum has 9 inputs and 2 outputs (1 or 0).
- We train a three layer MLP with 10 hidden neurons.
- To train the MLP, we simply use the following lines
  - load cancer_dataset
  - net = feedforwardnet(10);
  - net = train(net, cancerInputs, cancerTargets);
  - view(net) % to show the structure of the network
- The trained network can be evaluated by using it as a function
  - outputs=net(cancerInputs) ;
  - errors = gsubtract(cancerTargets,outputs);
  - performance = perform(net,cancerTargets,outputs)
  - performance = 0.0249

1) Grid search: n_hidden=1,2,…100
2) 10 fold Cross validation to be ensure the reliability of the selected value.

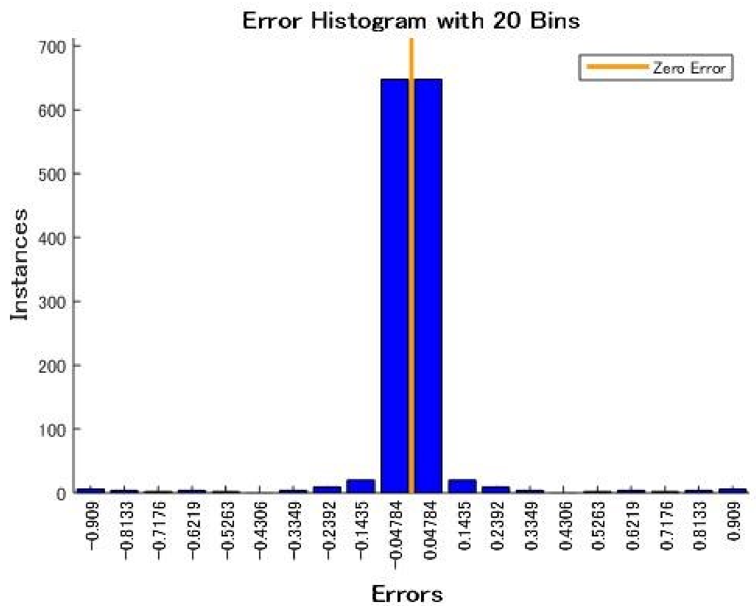
Example - 3

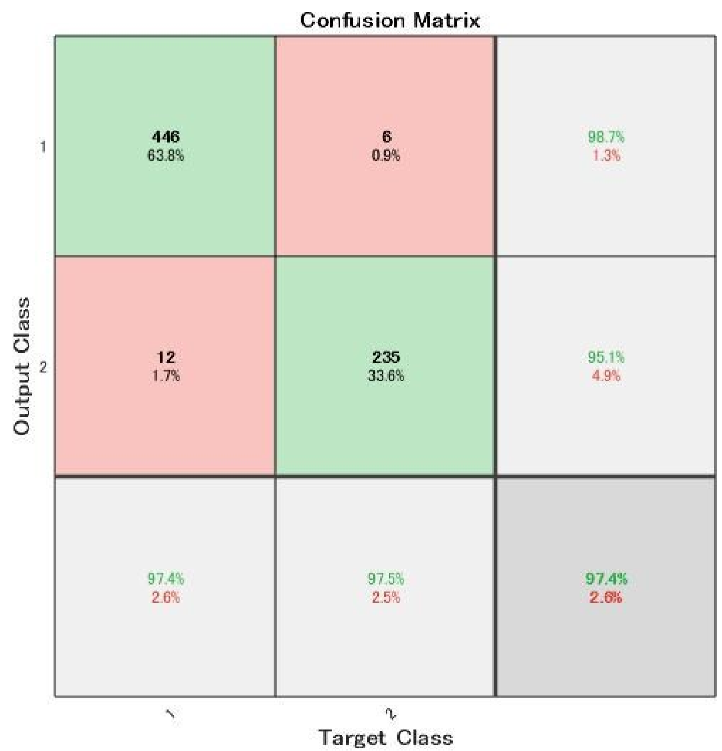




The learning curves

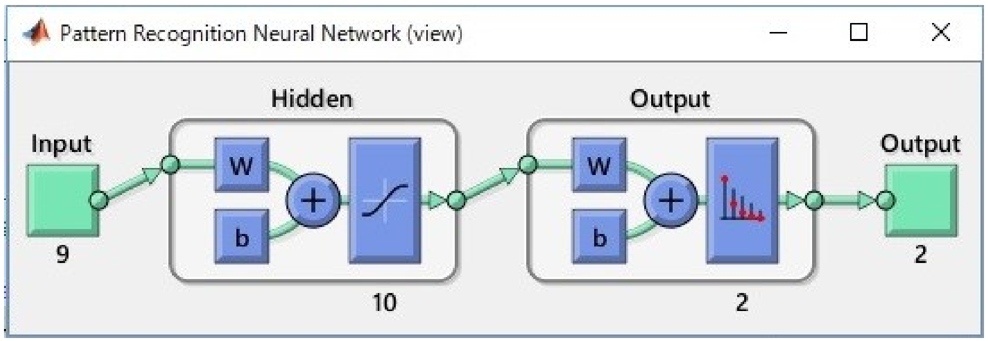
Example - 3



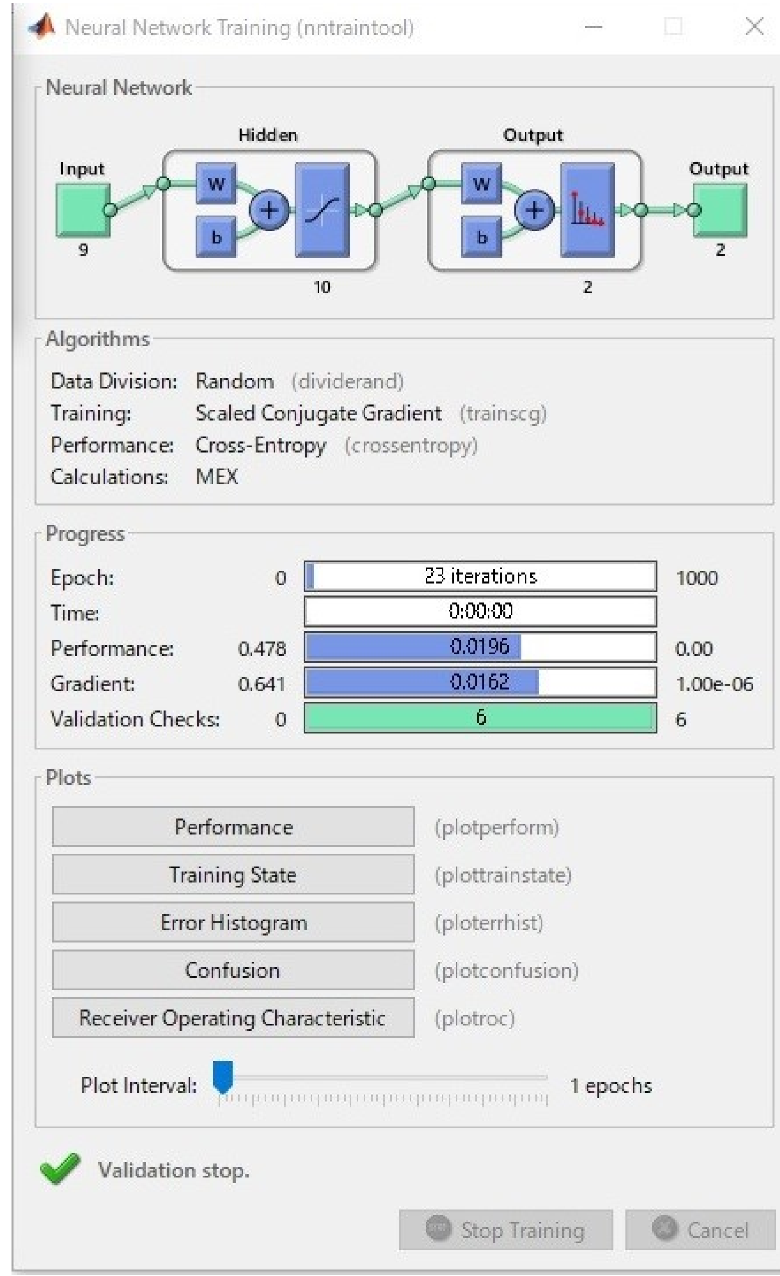


The error histogram and the confusion matrix

Example - 3



Upper: the trained network
Right: Training status



Homework of today

- In this lecture, we have studied batch learning. That is, using all data for each learning cycle (or epoch).
- When the number of data is very large, batch learning can become very time consuming.
- To reduce the computational cost, we may use mini-batch learning, or even on-line learning.
- Try to find out the meaning of mini-batch and on-line learning in the internet, and see how to conduct mini-batch learning or on-line learning by modifying the algorithm given in this lecture.