



ARTIFICIAL
NEURAL NETWORK

NEURAL NETWORK AND DEEP LEARNING

Third year computer science

Dr. Menas Ebrahim Eissa

Supervised Learning of a Neural Network

This section introduces the concepts and process of supervised learning of the neural network. It is addressed in the “Types of Machine Learning” section in Chapter 1. Of the many training methods, this book covers only supervised learning. Therefore, only supervised learning is discussed for the neural network as well. In the big picture, supervised learning of the neural network proceeds in the following steps:

- 1) Initialize the weights with adequate values.
- 2) Take the “input” from the training data, which is formatted as { input, correct output }, and enter it into the neural network. Obtain the output from the neural network and calculate the error from the correct output.
- 3) Adjust the weights to reduce the error.
- 4) Repeat Steps 2-3 for all training data

Supervised Learning of a Neural Network

These steps are basically identical to the supervised learning process of the “Types of Machine Learning” section. This makes sense because the training of supervised learning is a process that modifies the model to reduce the difference between the correct output and model’s output. The only difference is that the modification of the model becomes the changes in weights for the neural network. Figure 2-11 illustrates the concept of supervised learning that has been explained so far. This will help you clearly understand the steps described previously.

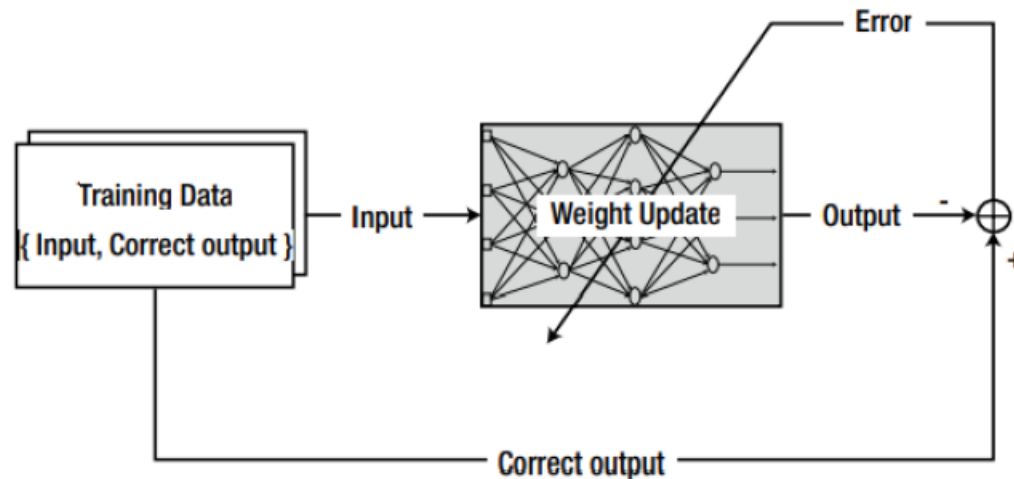
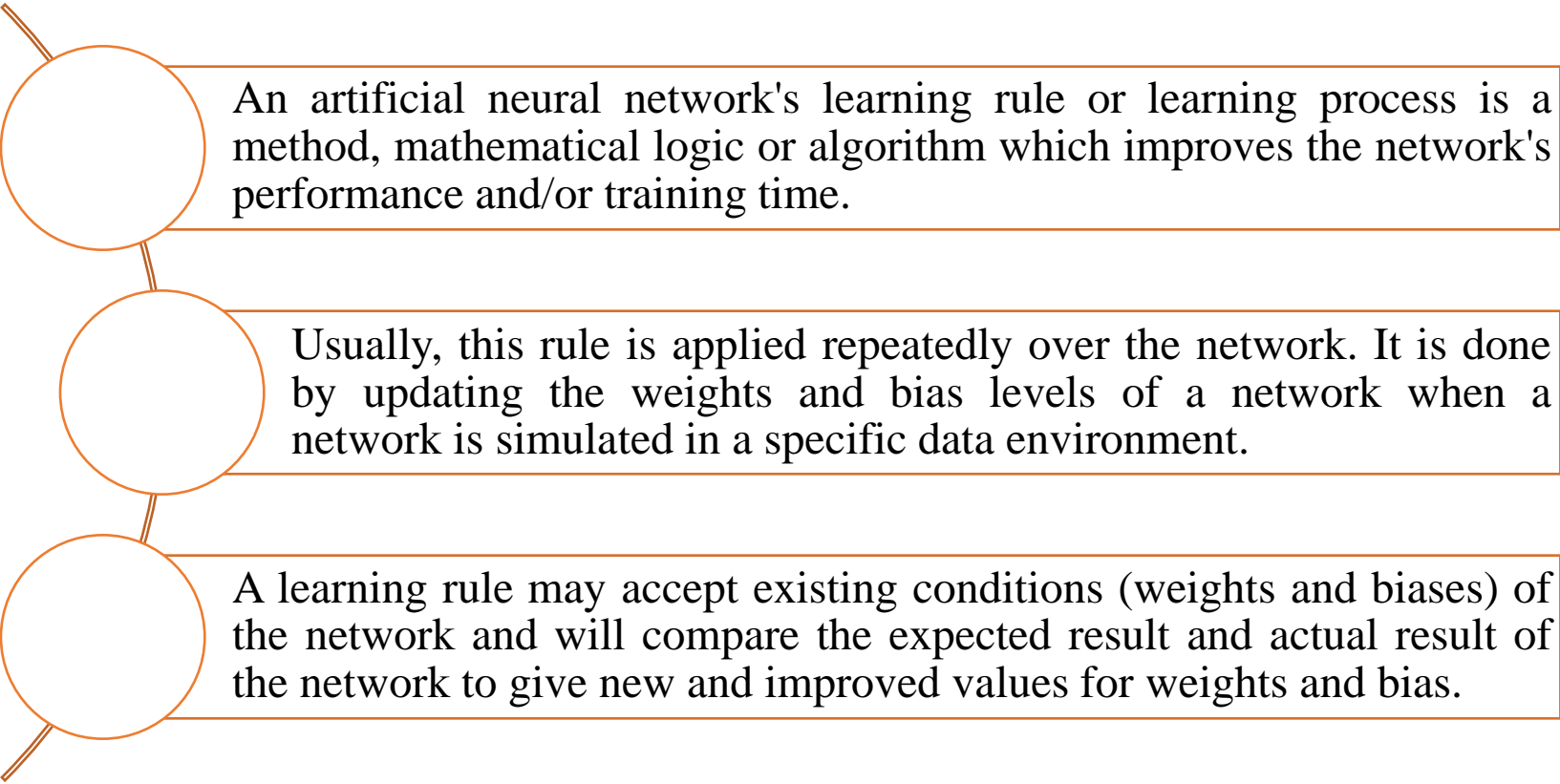


Figure 2-11 The concept of supervised learning

Learning Rules in Neural Network



An artificial neural network's learning rule or learning process is a method, mathematical logic or algorithm which improves the network's performance and/or training time.

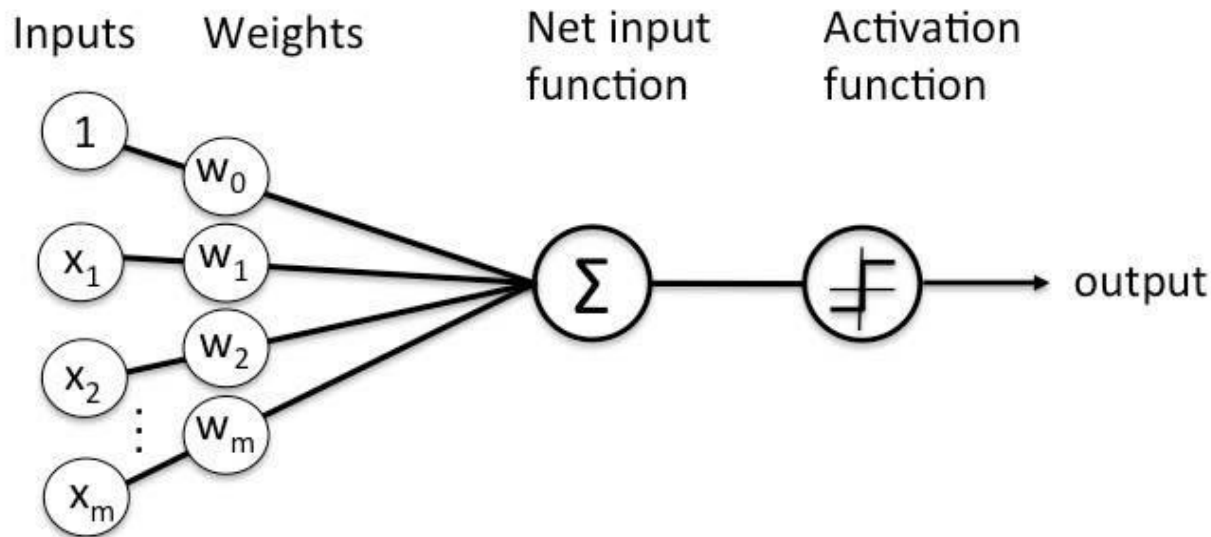
The diagram consists of three orange-outlined circles arranged vertically on the left side. Each circle is connected to a rectangular text box on the right by a thin orange line. The circles are empty, serving as visual markers for each point.

Usually, this rule is applied repeatedly over the network. It is done by updating the weights and bias levels of a network when a network is simulated in a specific data environment.

A learning rule may accept existing conditions (weights and biases) of the network and will compare the expected result and actual result of the network to give new and improved values for weights and bias.

Perceptron learning rule

- Perceptron was introduced by Frank Rosenblatt in 1957. He proposed a Perceptron learning rule based on the original MCP neuron. A Perceptron is an algorithm for supervised learning of binary classifiers. This algorithm enables neurons to learn and processes elements in the training set one at a time.



Simple Perceptron learning rule

The step-by-step algorithm is given below:

Step1:

- Initialize all weights and bias Let us take $\eta = (0:1)$ as the learning rate.

Step2:

- For each input training vector and target output pair, $\mathbf{S} : t$, do steps 2–5.

Step3:

- Set activation for input units : $x_i = S_i, i = 1, \dots, n$.

Simple Perceptron learning rule

Step 4:

Compute the output unit as $z = b + \sum_{i=1}^n x_i w_i$

- The activation function is used as

$$\hat{y}_{in} = f(z) = \begin{cases} 1, & \text{if } y_{in} > \theta \\ 0, & \text{if } -\theta \leq y_{in} \leq \theta \\ -1, & \text{if } y_{in} < -\theta \end{cases}$$

Step5:

Calculate error

$$E = y_j - \hat{y}_j$$

Step6:

The weights and bias are updated if the target is not equal to the output response.

$$\text{then } \mathbf{w}_{i,j} = \mathbf{w}_{i,j} + \eta(y_j - \hat{y}_j)\mathbf{x}_i \text{ for } j = 1, \dots, n$$

$$b = b + \eta(y_j - \hat{y}_j)$$

- The stopping conditions may be the change in weights.

The learning rate, η

- The learning rate, η , determines how much the weight is changed per time.
- If this value is too high, the output wanders around the solution and fails to converge.
- In contrast, if it is too low, the calculation reaches the solution too slowly

Example: We have a binary classification problem where the perceptron should learn the logical AND function using the perceptron learning rule (we will use the step function as an activation function).

The AND function takes two binary inputs x_1 and x_2 , and produces a single binary output y :

$$\bullet \ y = x_1 \text{AND} x_2$$

Truth Table for AND

x_1	x_2	Target y
0	0	0
0	1	0
1	0	0
1	1	1

A perceptron consists of:

- **Inputs:** x_1, x_2
- **Weights:** w_1, w_2
- **Bias:** b
- **Activation Function:** Step function

$$\hat{y} = f(w_1x_1 + w_2x_2 + b)$$

where the step activation function is:

$$f(z) = \begin{cases} 1, & \text{if } z \geq 0 \\ 0, & \text{if } z < 0 \end{cases}$$

We initialize the weights and bias **randomly**:

- $w_1 = 0.2$
- $w_2 = -0.1$
- $b = 0.1$

Epoch 1

Step 1: Input (0,0)

$$z = (0.2 \times 0) + (-0.1 \times 0) + 0.1 = 0.1$$

$$\hat{y} = f(0.1) = 1$$

Error:

$$e = y - \hat{y} = 0 - 1 = -1$$

Update weights:

$$w_1 = 0.2 + 0.1(-1 \times 0) = 0.2$$

$$w_2 = -0.1 + 0.1(-1 \times 0) = -0.1$$

$$b = 0.1 + 0.1(-1) = 0.0$$

Step 2: Input (0,1)

$$z = (0.2 \times 0) + (-0.1 \times 1) + 0.0 = -0.1$$

$$\hat{y} = f(-0.1) = 0$$

Error:

$$e = 0 - 0 = 0$$

(No weight update)

Step 3: Input (1,0)

$$z = (0.2 \times 1) + (-0.1 \times 0) + 0.0 = 0.2$$

$$\hat{y} = f(0.2) = 1$$

Error:

$$e = 0 - 1 = -1$$

Update weights:

$$w_1 = 0.2 + 0.1(-1 \times 1) = 0.1$$

$$w_2 = -0.1 + 0.1(-1 \times 0) = -0.1$$

$$b = 0.0 + 0.1(-1) = -0.1$$

Step 4: Input (1,1)

$$z = (0.1 \times 1) + (-0.1 \times 1) + (-0.1) = -0.1$$

$$\hat{y} = f(-0.1) = 0$$

Error:

$$e = 1 - 0 = 1$$

Update weights:

$$w_1 = 0.1 + 0.1(1 \times 1) = 0.2$$

$$w_2 = -0.1 + 0.1(1 \times 1) = 0.0$$

$$b = -0.1 + 0.1(1) = 0.0$$

Epoch 2

We repeat the process until all outputs match the target values.

After multiple epochs, the perceptron will learn the correct weights for the AND function:

$$w_1 = 0.2, \quad w_2 = 0.1, \quad b = -0.3$$

The final decision function is:

$$\hat{y} = f(0.2x_1 + 0.1x_2 - 0.3)$$

Testing the Final Model

x_1	x_2	$z = 0.2x_1 + 0.1x_2 - 0.3$	\hat{y}
0	0	-0.3	0
0	1	-0.2	0
1	0	-0.1	0
1	1	0.1	1

The perceptron has **successfully learned** the AND function!

Equation of Decision Boundary

The perceptron's decision function is:

$$\hat{y} = f(0.2x_1 + 0.1x_2 - 0.3)$$

The decision boundary is found by setting the **net input** to zero:

$$0.2x_1 + 0.1x_2 - 0.3 = 0$$

Solving for x_2 :

$$x_2 = -\frac{0.2}{0.1}x_1 + \frac{0.3}{0.1}$$

$$x_2 = -2x_1 + 3$$

