Neural Networks and Fuzzy Logic (630514)

Lecture 8

Supervised Learning in Neural Networks (Part 1)

A prescribed set of well-defined rules for the solution of a learning problem is called a ***learning algorithm***. Variety of learning algorithms are existing, each of which offers advantages of its own. Basically, learning algorithms differ from each other in the way in which the adjustment Δw_{kj} to the synaptic weight w_{kj} is formulated.

Theoretically, a neural network could learn by:

1. Developing new connections.
2. Deleting existing connections.
3. Changing connecting weights, (and practically).
4. Changing the threshold values of neurons, (and **practically**).
5. Changing activation function, propagation function or output function.
6. Developing new neurons.
7. Deleting existing neurons.

Fundamentals on learning and training:

- *Learning is a process by which the free parameters (weights and biases) of a neural network are adapted through a continuing process of stimulation by the environment.*
- This definition of the learning process implies the following sequence of events:
  1. The neural network is stimulated by an environment.
  2. The neural network is changed (internal structure) as a result of this stimulation.
  3. The neural network responds in a new way to the environment.

Setting the Weights

- The method of setting the values of the weights (training) is an important characteristic of different neural nets. different types of training:
  - **Supervised**: in which the network is trained by providing it with input and matching output patterns.
  - **Unsupervised** or **Self-organization** in which an output unit is trained to respond to clusters of pattern within the input, the system must develop its own representation of the input stimuli.
  - **Reinforcement learning**, sometimes called **reward-penalty learning**, is a combination of the above two methods; it is based on presenting input vector **x** to a neural network and looking at the output vector calculated by the network. If it is considered "**good**," then a "**reward**" is given to the network in the sense that the existing

connection weights are increased; otherwise the network is "**punished**," the connection weights decreased.
  - Nets whose weights are **fixed** without an iterative training process.

- **Supervised learning network paradigms.**
- Supervised Learning in Neural Networks: **Perceptrons** and **Multilayer Perceptrons.**
- **Training set**: A training set (named **P**) is a set of training patterns, which we use to train our neural net.
- **Batch training** of a network proceeds by making weight and bias changes based on an entire set (batch) of input vectors.
- **Incremental training** changes the weights and biases of a network as needed after presentation of each individual input vector. Incremental training is sometimes referred to as "**on line**" or "**adaptive**" training.
- **Hebbian learning rule** suggested by Hebb in his classic book **Organization of Behavior**: The basic idea is that if two units **j** and **k** are active simultaneously, their interconnection must be strengthened, If **j** receives input from **k**, the simplest version of Hebbian learning prescribes to modify the weight $\mathbf{w_{jk}}$ with

$$\Delta w_{jk} = \gamma y_j y_k,$$

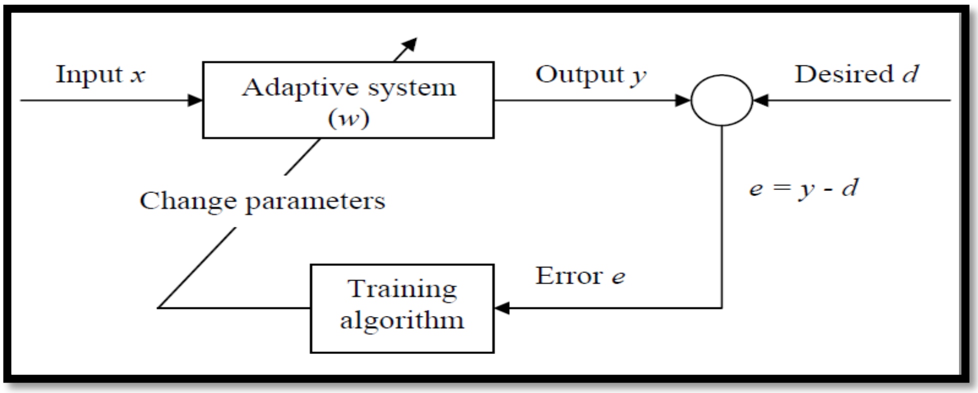
Where γ, is a positive constant of proportionality representing the learning rate.

- Another common rule uses not the actual activation of unit **k** but the deference between the actual and desired activation for adjusting the weights.

$$\Delta w_{jk} = \gamma y_j (d_k - y_k),$$

in which $\mathbf{d_k}$ is the desired activation provided by a teacher. This is often called the **Widrow-Hoff rule or the delta rule**.

Error-correction learning



Error-correction learning diagram

- Let $\mathbf{d_k(n)}$ denote some desired response or target response for neuron $\mathbf{k}$ at time $\mathbf{n}$. Let the corresponding value of the actual response (output) of this neuron be denoted by $\mathbf{y_k(n)}$.
- Typically, the actual response $\mathbf{y_k(n)}$ of neuron $\mathbf{k}$ is different from the desired response $\mathbf{d_k(n)}$. Hence, we may define an **error signal**

$$e_k(n) = y_k(n) - d_k(n)$$

- The ultimate purpose of **error-correction learning** is to minimize a **cost function** based on the error signal $\mathbf{e_k(n)}$.
- A criterion commonly used for the cost function is the **instantaneous value of the mean square-error** criterion

$$J(n) = \frac{1}{2}\sum_k e_k^2(n)$$

- The network is then optimized by **minimizing** $J(n)$ **with respect to the synaptic weights of the network**. Thus, according to the error-correction learning rule (or delta rule), the synaptic weight adjustment is given by

$$\Delta w_{kj} = \eta e_k(n) x_j(n)$$

- Let $w_{kj}(n)$ denote the value of the synaptic weight w_{kj} at time n. At time n an *adjustment* $\Delta w_{kj}(n)$ is applied to the synaptic weight $w_{kj}(n)$, yielding the updated value

$$w_{kj}(n+1) = w_{kj}(n) + \Delta w_{kj}(n)$$

The perceptron' training algorithm

The Perceptron Learning Rule

- Perceptrons are trained on ***examples of desired behavior***, which can be summarized by a set of input-output pairs

$$\{p1, t1\}, \{p2, t2\}, \dots, \{pQ, tQ\}$$

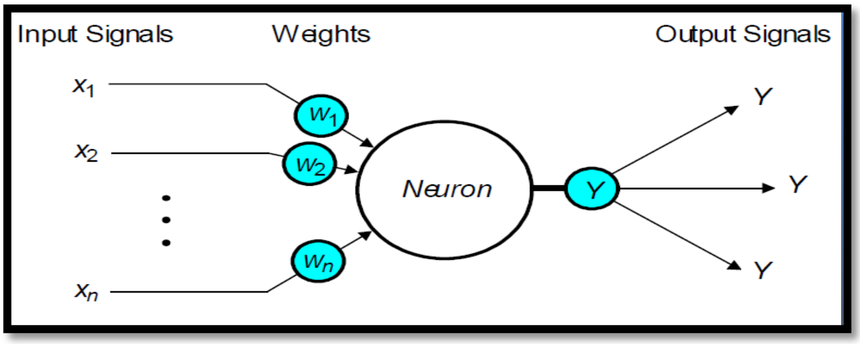
- The objective of training is to reduce the **error e**, which is the difference $t - a$ between the perceptron output a, and the target vector t.
- This is done by **adjusting** the *weights* (**W**) and *biases* (**b**) of the perceptron network according to following equations

$$W_{new} = W_{old} + \Delta W = W_{old} + eP^T$$
$$b_{new} = b_{old} + \Delta b = b_{old} + e$$

Where

$$e = t - a$$

- Diagram of a neuron:

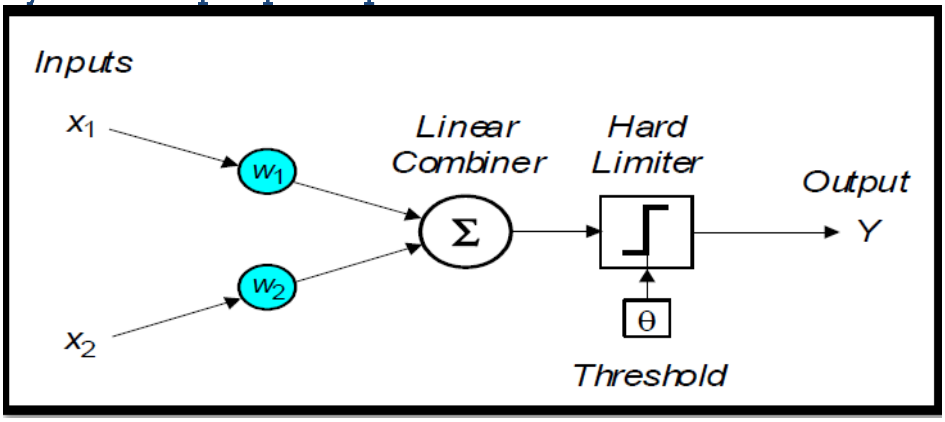


- The neuron computes the *weighted sum of the input signals and compares the result with a threshold value*, θ. If the net input is less than the threshold, the neuron output is **–1**. But if the net input is greater than or equal to the threshold, the neuron becomes activated and its output attains a value **+1**.
- The neuron uses the following *transfer or activation* function:

$$X = \sum_{i=1}^{n} x_i w_i \qquad Y = \begin{cases} +1, & \text{if } X \geq \theta \\ -1, & \text{if } X < \theta \end{cases}$$

Single neuron' training algorithm

- In 1958, Frank Rosenblatt introduced a training algorithm that provided the first procedure for training a simple ANN: a perceptron.
  Single-layer two-input perceptron

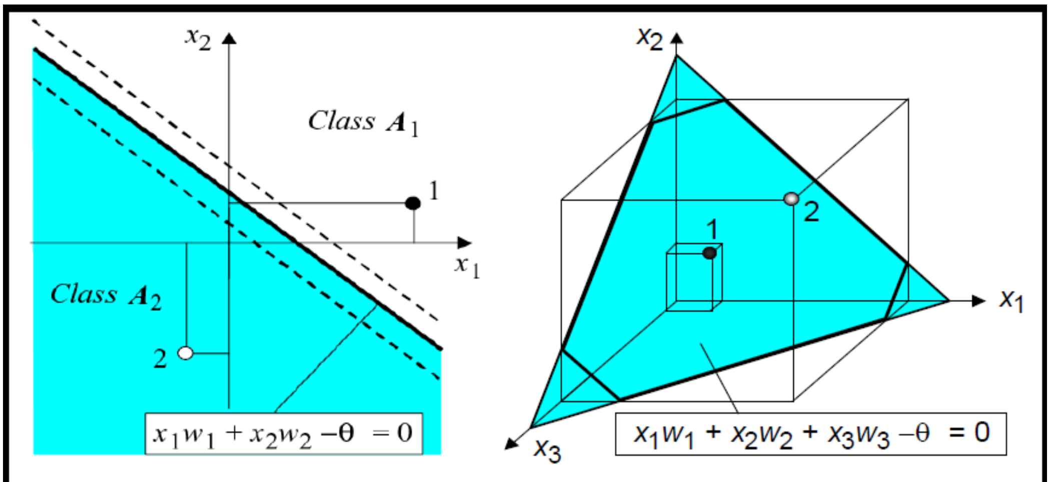


The Perceptron

- The operation of Rosenblatt's perceptron is based on the McCulloch and Pitts neuron model. The model consists of a linear combiner followed by a hard limiter.
- The weighted sum of the inputs is applied to the hard limiter, which produces an output equal to +1 if its input is positive and -1 if it is negative.
- The aim of the perceptron is to classify inputs, x1, x2, . . ., xn, into one of two classes, say **A1** and **A2**.
- In the case of an elementary perceptron, the **n- dimensional space** is divided by a ***hyperplane*** into two decision regions. The hyperplane is defined by the *linearly separable* function:

$$\sum_{i=1}^{n} x_i w_i - \theta = 0$$

Linear separability in the perceptrons



(a) Two-input perceptron. (b) Three-input perceptron.

How does the perceptron learn its classification tasks?

- This is done by making **small adjustments** in the **weights** to reduce the difference between the actual and desired outputs of the perceptron. The initial weights are randomly assigned, usually in the range **[-0.5, 0.5]**, and then updated to obtain the output consistent with the training examples.
- If at iteration **p**, the actual output is **Y(p)** and the desired output is **Yd (p),** then the error is given by:

$$e(p) = Y_d(p) - Y(p)$$

where $\boldsymbol{p}$ **= 1, 2, 3, . . .**

Iteration ***p*** here refers to the ***pth*** training example presented to the perceptron.

- If the error, ***e(p),*** is positive, we need to increase perceptron output ***Y(p),*** but if it is negative, we need to decrease ***Y(p).***

The perceptron learning rule

$$w_i(p+1) = w_i(p) + \quad x_i(p) \cdot e(p)$$

where $\boldsymbol{p}$ **=** ***1, 2, 3***, . . .

- α is the learning rate, a positive constant less than unity.
- The perceptron learning rule was first proposed by Rosenblatt in 1960. Using this rule we can derive the perceptron training algorithm for classification tasks.

Perceptron's training algorithm

Step 1: Initialization

Set initial weights $w_1, w_2, \ldots, w_n$ and threshold θ to random numbers in the range ***[-0.5, 0.5]***.

Step 2: Activation

Activate the perceptron by applying inputs $x_1(p), x_2(p), \ldots, x_n(p)$ and desired output $Y_d(p)$.

Calculate the actual output at iteration $p = 1$

$$Y(p) = step\left[\sum_{i=1}^{n} x_i(p)\, w_i(p) - \theta\right]$$

Where **n** is the number of the perceptron inputs, and **step** is a step activation function.

Step 3: Weight training

Update the weights of the perceptron

$$w_i(p+1) = w_i(p) + \Delta w_i(p)$$

where $\Delta w_i(p)$ is the weight correction at iteration p.

$$\Delta w_i(p) = \alpha \cdot x_i(p) \cdot e(p)$$

The weight correction is computed by the delta rule:

Step 4: Iteration

Increase iteration p by one, go back to ***Step 2*** and repeat the process until convergence.

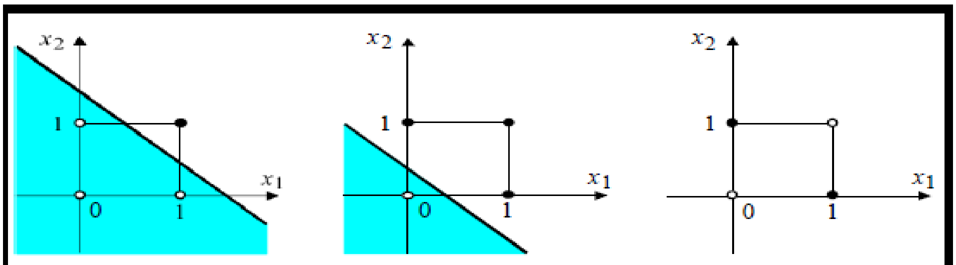
Example of perceptron learning: the logical operation *AND*

Epoch	Inputs		Desired output	Initial weights		Actual output	Error	Final weights	
	x_1	x_2	Y_d	w_1	w_2	Y	e	w_1	w_2
1	0	0	0	0.3	−0.1	0	0	0.3	−0.1
	0	1	0	0.3	−0.1	0	0	0.3	−0.1
	1	0	0	0.3	−0.1	1	−1	0.2	−0.1
	1	1	1	0.2	−0.1	0	1	0.3	0.0
2	0	0	0	0.3	0.0	0	0	0.3	0.0
	0	1	0	0.3	0.0	0	0	0.3	0.0
	1	0	0	0.3	0.0	1	−1	0.2	0.0
	1	1	1	0.2	0.0	1	0	0.2	0.0
3	0	0	0	0.2	0.0	0	0	0.2	0.0
	0	1	0	0.2	0.0	0	0	0.2	0.0
	1	0	0	0.2	0.0	1	−1	0.1	0.0
	1	1	1	0.1	0.0	0	1	0.2	0.1
4	0	0	0	0.2	0.1	0	0	0.2	0.1
	0	1	0	0.2	0.1	0	0	0.2	0.1
	1	0	0	0.2	0.1	1	−1	0.1	0.1
	1	1	1	0.1	0.1	1	0	0.1	0.1
5	0	0	0	0.1	0.1	0	0	0.1	0.1
	0	1	0	0.1	0.1	0	0	0.1	0.1
	1	0	0	0.1	0.1	0	0	0.1	0.1
	1	1	1	0.1	0.1	1	0	0.1	0.1

Threshold: $\theta = 0.2$; learning rate: $\alpha = 0.1$

Two-dimensional plots of basic logical operations

A perceptron can learn the **operations *AND* and *OR***, **but not *Exclusive-OR***.



(b) OR $(x_1 \cup x_2)$

(c) $Exclusive\text{-}OR$ $(x_1 \oplus x_2)$