



CE213 Artificial Intelligence – Lecture 14

Neural Networks: Part 2

Learning Rules

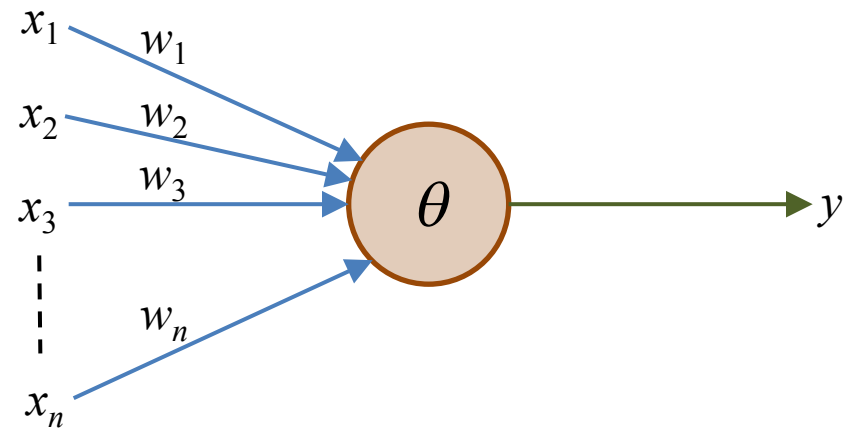
- Hebb Rule
- Perceptron Rule
- Delta Rule

Neural Networks Using Linear Units without Thresholding

Learning in Neural Networks

Let's consider learning in one MP neuron first before considering learning in a neural network.

It can be described by a diagram:



or described by equations:

If $\sum_{i=1}^n w_i x_i \geq \theta$, then $y = 1$

If $\sum_{i=1}^n w_i x_i < \theta$, then $y = 0$

The function of the MP neuron is determined by the values of its weights and threshold.

Given a labelled sample dataset, how to determine the values of w_i and θ so that the MP neuron will classify or fit the sample data correctly?

- 1) Analytical way (usually when $n \leq 3$)
- 2) Machine learning

Learning in Neural Networks (2)



The function of an MP neuron can be constructed by **appropriate choice of weight and threshold values**, e.g., making the following hold if (x_1, x_2) and (x'_1, x'_2) are from different classes:

$$w_1 x_1 + w_2 x_2 > \theta \quad \rightarrow \quad y=1 \text{ (class 1)}$$

$$w_1 x'_1 + w_2 x'_2 < \theta \quad \rightarrow \quad y=0 \text{ (class 2)}$$

This is feasible when there are 2 inputs only, but with a large number of inputs we have to **train** a neuron or neural network to achieve the desired weights by fitting a set of samples through machine learning.

What information is available from samples and the MP neuron?

The current input:	$x_1 \dots x_n$		from the samples
The desired output (label):	z		
The current weights:	$w_1 \dots w_n$		from the MP neuron
The actual output :	y		

How to make use of the above information to learn – to change weight values?

Learning in Neural Networks (3)

A general solution to machine learning in neural networks is to use the following learning rule:

$$w_i(t + 1) = w_i(t) + \Delta(x_i, w_i, y, z)$$

where

$w_i(t)$ is the weight of the i th input at time t

Δ is a function that determines the change in weight.

A learning procedure would simply be:

Initialisation of weight values;

REPEAT

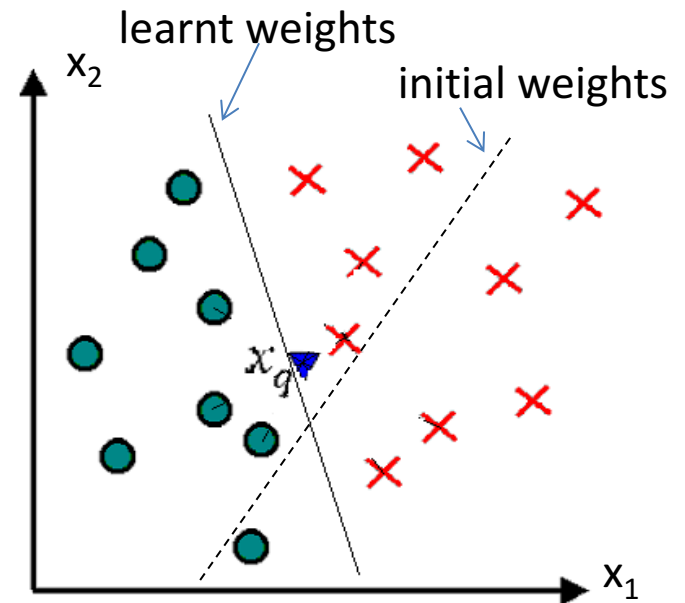
Present next sample;

Adjust all weights using Δ ;

UNTIL Training complete;

(There could be different completion criteria.)

<https://www.youtube.com/watch?v=vGwemZhPlsA>



Learning in Neural Networks (4)

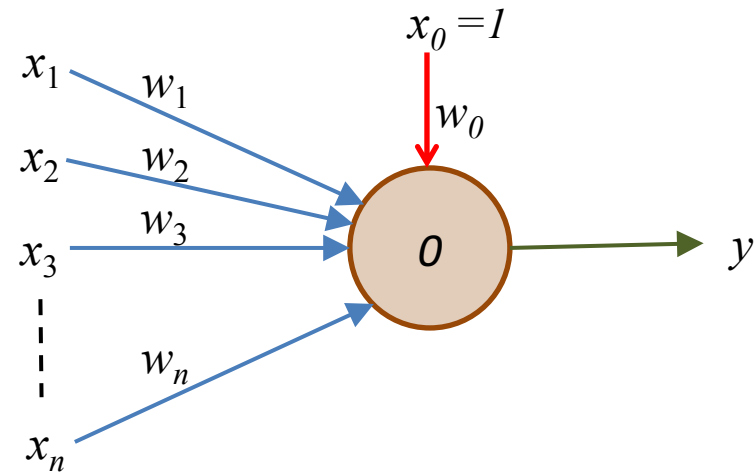
How to update the value of the threshold through machine learning?

If we set threshold to zero and add an extra input x_0 with weight w_0 , then the threshold can be regarded as a special weight.

Let x_0 be always 1, then the MP neuron can be described as follows:

If $\sum_{i=0}^n w_i x_i = \sum_{i=1}^n w_i x_i + w_0 \geq 0$ then $y = 1$

If $\sum_{i=0}^n w_i x_i = \sum_{i=1}^n w_i x_i + w_0 < 0$ then $y = 0$



Now there is no θ in the diagram or the equations of the MP neuron. This is equivalent to set $\theta = -w_0$

Learning in Neural Networks (5)

So, the weight with fixed input is essentially an adaptive threshold, because the equations describing the MP neuron are as follows:

$$\text{If } \sum_{i=1}^n w_i x_i \geq -w_0 \text{ then } y = 1$$

$$\text{If } \sum_{i=1}^n w_i x_i < -w_0 \text{ then } y = 0$$

($-w_0$ plays the role of threshold θ)

In this way, we can arrange for the threshold to be modified in the same way as the weights.

Now let's focus on **how to determine** $\Delta(x_i, w_i, y, z)$.

Hebb Rule

Hebb (1949) proposed that learning could take the form of “strengthening the connections” between any pair of neurons that were **simultaneously active**.

This implies a learning rule of the form (x_i may be the output of another neuron)

$$w_i(t + 1) = w_i(t) + \alpha x_i y$$

where α is a constant that determines the rate of learning.

This rule was modified later for training MP neurons.

During learning, the output of the neuron (y) is forced to the desired value (z). Hence the learning rule becomes

$$w_i(t + 1) = w_i(t) + \alpha x_i z$$

This is now known as the “**Hebb Rule**”.

Is this supervised learning?

Perceptron Rule

Perceptron Rule was the second learning rule in the history of machine learning, which was proposed by Prof. Marvin Minsky in late 1950s when he was a PhD student.

The basic idea is to adjust the weights only when the neuron would have given the wrong output.

The Perceptron Rule:

$$w_i(t + 1) = w_i(t) + \alpha x_i z \quad \text{only if } y \neq z$$

which is actually the Hebb rule restricted to **error correction**.

However, such a minor change to the Hebb rule made a big difference.

Delta Rule

The delta rule generalises the idea of **error correction** introduced in the perceptron rule.

Rather than having a rule that is only applied if an error occurs, delta rule includes the error in it:

$$w_i(t + 1) = w_i(t) + \alpha x_i(z - y) = w_i(t) + \alpha x_i \delta$$

This delta rule was proposed in 1960, also known as the Widrow-Hoff rule.

Can δ be $(y-z)$?

Why is it called the delta rule?

Because the difference or error $(z - y)$ is often written as δ .

Delta Rule (2)

Delta rule can be used to train neural networks for both classification and function approximation.

Let's consider the task of training a ***function approximator*** First.

If we take away the thresholding, we have a linear sum unit:

$$y = \sum_{i=0}^n w_i x_i \quad (\text{normally } x_0 = 1)$$

The input and output could be real numbers, not just 1 or 0.

Such a computing unit clearly computes a hyperplane in n -dimensional space.

Delta Rule (3)

How will this computing unit behave if it is trained?

Suppose training data can be described by $z = f(x_1, \dots, x_n)$.

Then the weights will adapt to values such that the computing unit provides a **linear** approximation to the function f .

Therefore, the computing unit can be used for function approximation as well as for classification.

Function approximation and classification are closely related:

Function approximation is a **surface** in n -dimensional space.

This surface separates two classes of points in n -dimensional space.

For function approximation, it can be proved that the delta rule is an optimal strategy for changing the weights. (Strict proof is beyond the scope of CE213)

Limitations of the Delta Rule

Two major limitations

- It can only learn linear functions or linear separation for classification.
- It is equivalent to linear regression.

However, standard techniques for calculating regressions are orders of magnitude faster; And there is no need to choose an appropriate value of learning rate α .

Of course, delta rule can start training neurons as soon as data is available.

Choosing appropriate value for learning rate α

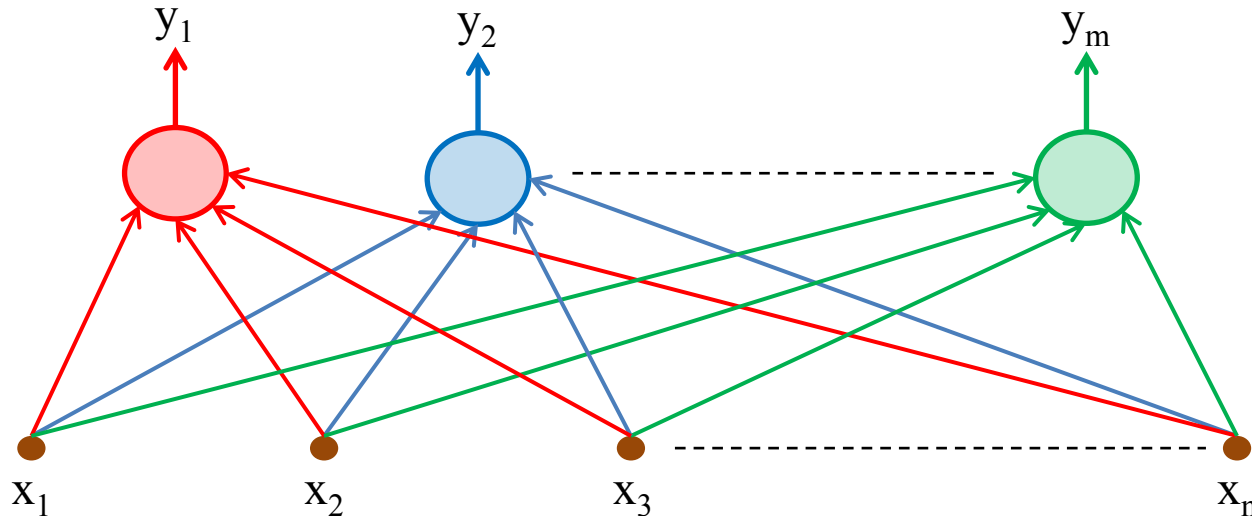
If α is too small, learning will be very slow.

If α is too large, the final weight values will cycle around the optimum values.

In many applications, 'trial and error' could be the only practical method for choosing appropriate learning rate.

More Than One Output (Network)

A Single Layer Linear Network (consisting of linear units):



The behaviour of this network can be most conveniently expressed by

$$\bar{y} = \mathbf{W}\bar{x}$$

where

\bar{x} is the input vector

\bar{y} is the output vector

\mathbf{W} is the weight matrix (w_{ij} , $i=1,2,\dots,m$; $j=1,2,\dots,n$)

More Than One Output (2)

The weight of the j^{th} input to the i^{th} neuron is element $[i,j]$ of the weight matrix \mathbf{W} .

So the delta learning rule now takes the following form

$$\mathbf{W}(t + 1) = \mathbf{W}(t) + \alpha \bar{\delta} \bar{x}^{\top} \quad (\top - \text{transpose})$$

where

$$\bar{\delta} \equiv (\bar{z} - \bar{y}) = (\bar{z} - \mathbf{W} \bar{x})$$

$$\bar{\delta}, \bar{x}, \bar{y}, \bar{z} \text{ are vectors, e.g., } \bar{x} = [x_1, x_2, \dots, x_n]^{\top}$$

For each element in the matrix \mathbf{W} , the delta learning rule is as follows:

$$w_{ij}(t + 1) = w_{ij}(t) + \alpha x_j (z_i - y_i)$$

More Than One Output (3)

How does such a single layer network with multiple outputs behave?

Clearly it can be viewed as m single neurons operating in parallel:

There are no connections that could allow the behaviour of one neuron to affect another.

A single computing unit finds an approximation to a scalar function

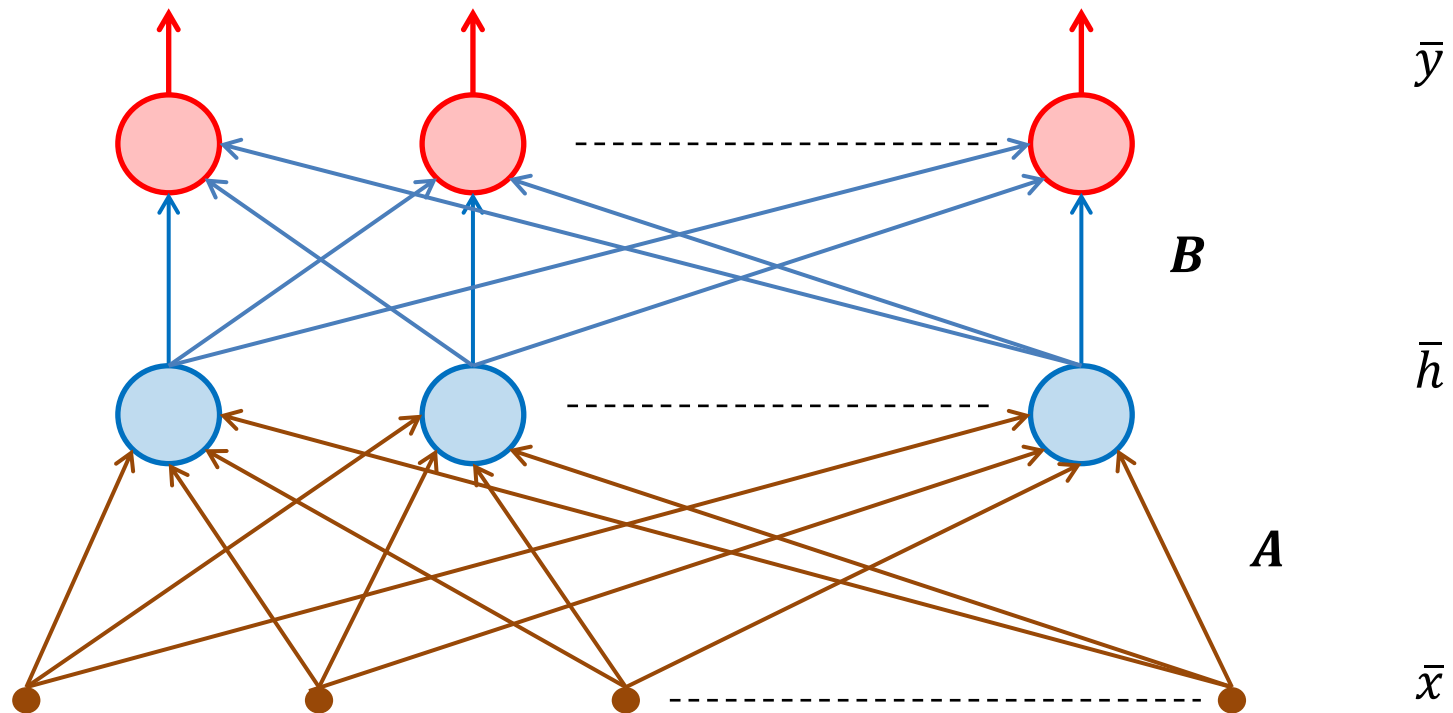
$$y = f(\bar{x})$$

The multiple output network simply extends that to the case where the function has vector rather than scalar values.

$$\bar{y} = \boldsymbol{f}(\bar{x})$$

Multiple Layer Network Using Linear Units

Let's try adding another layer of linear computing units
(not linear separation units, because there is no thresholding):



This is called a **feed-forward network** because signals flow in forward direction only.

Multiple Layer Network Using Linear Units (2)

Now let's see what this multilayer network does:

$$\bar{h} = \mathbf{A}\bar{x} \quad (\text{lower layer})$$

$$\bar{y} = \mathbf{B}\bar{h} \quad (\text{higher layer})$$

Therefore, the relationship between the output and input of the multilayer network can be described by

$$\bar{y} = \mathbf{B}(\mathbf{A}\bar{x}) = (\mathbf{BA})\bar{x}$$

But the product of two matrices is another matrix.

So let

$$\mathbf{W} = \mathbf{BA}$$

Hence

$$\bar{y} = \mathbf{W}\bar{x}$$

A multilayer linear network can be equivalently implemented by a single layer network.

Multiple Layer Network Using Linear Units (3)

Conclusion

There is no point building a multilayer feed-forward network if the computing units perform linear transformations of their inputs.

Therefore

The only way to build multilayer neural networks that are more powerful than single layer networks is to use computing units (neurons) with non-linear output functions.

But

The delta rule is designed to find the best weights for linear units only.

Need of new learning rule/algorithm!

Summary

Learning in Neural Networks Using the Delta Rule

Hebb Rule

Perceptron Rule

Delta Rule

Limitations of the Delta Rule

Multilayer Networks Using Linear Computing Units

→ Need of Nonlinear Computing Units or Neurons
in Multilayer Neural Networks and More Powerful
Learning Algorithms