2.160 IDENTIFICATION, ESTIMATION, AND LEARNING

LECTURE NOTES NO. 16

17. Neural Networks

17.1 Physiological Background

Neuro-physiology

- A Human brain has approximately 14 billion neurons, 50 different kinds of neurons. … uniform
- Massively-parallel, distributed processing
  Very different from a computer (a Turing machine)

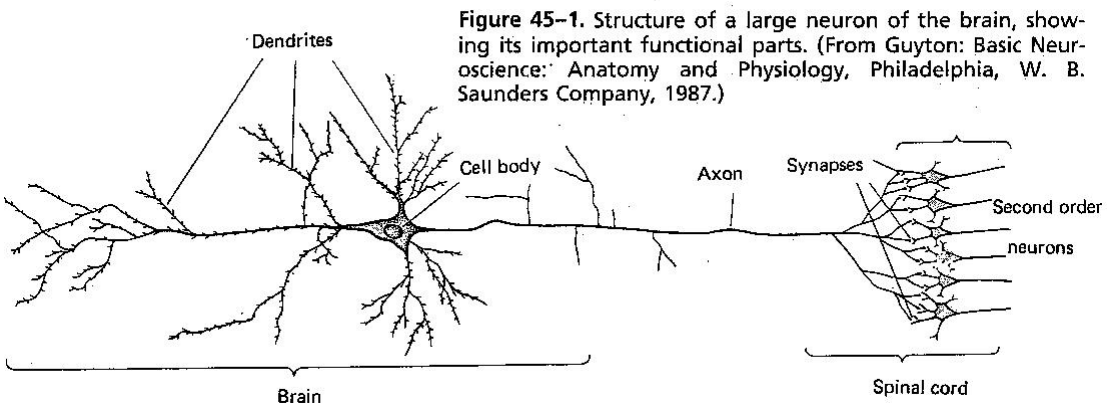
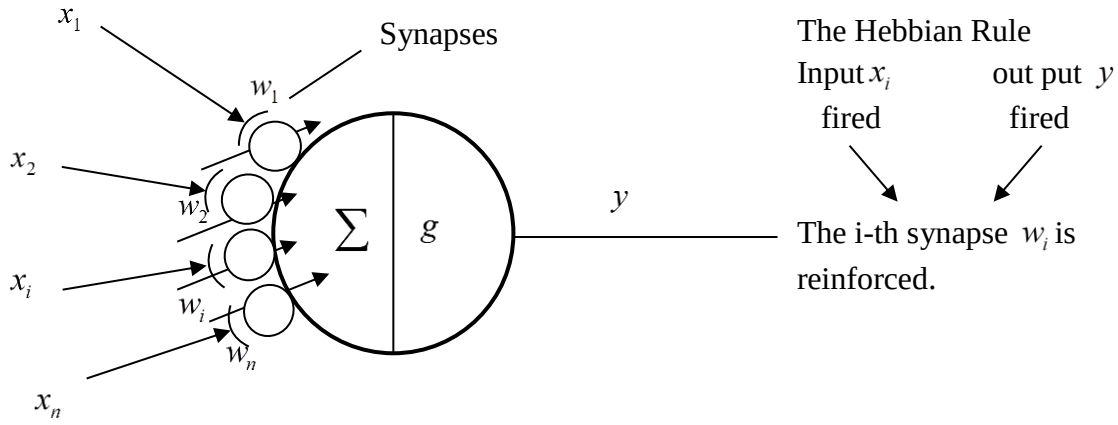


Figure 45–1. Structure of a large neuron of the brain, showing its important functional parts. (From Guyton: Basic Neuroscience: Anatomy and Physiology, Philadelphia, W. B. Saunders Company, 1987.)

McCulloch and Pitts, Neuron Model 1943
Donald Hebb, Hebbian Rule, 1949
…Synapse reinforcement learning
Rosenblatt, 1959
…The perceptron convergence theorem

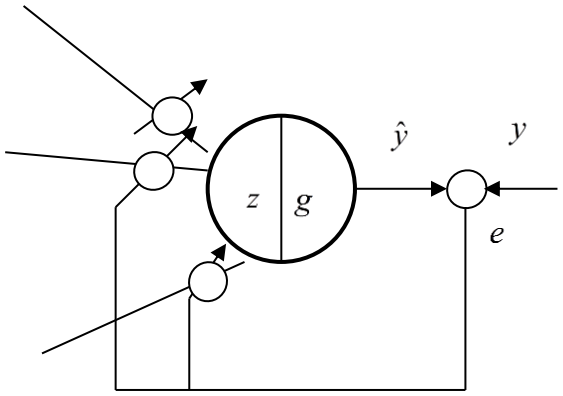


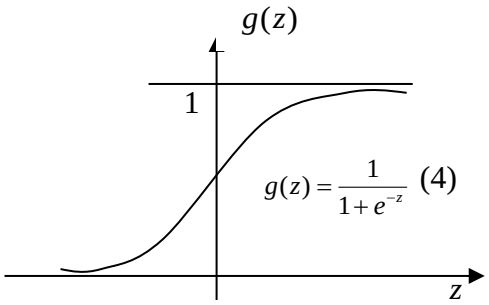
The Hebbian Rule
Input x_i fired — out put y fired

The i-th synapse w_i is reinforced.

The electric conductivity increases at w_i

Consider a single neuron model with a logistic output function $g(z)$, as shown below. The synaptic weight w_i is changed based on a gradient descent method, so that squared error between the predicted output $\hat{y}$ and its true value *y* may be reduced.





logistic function, or sigmoid function

(1) Error $e = \hat{y} - y = g(z) - y \qquad z = \sum_{i=1}^{n} w_i x_i$

Gradient Descent method

(2) $\Delta w_i = -\rho \cdot grad_{w_i} e^2 = -\rho\, 2e \frac{\partial e}{\partial w_i} = -2\rho\, e \frac{dg}{dz} \frac{\partial z}{\partial w_i}$

ρ : learning rate

(3) $\therefore \quad \Delta w_i = -2\rho\, g' e\, x_i$

Note that the above learning rule with the correct output *y* presented (called Supervised Learning) changes the weight in proportion to the product of input x_i and error *e*. Replacing *e* by $\hat{y}$ yields the Hebbian Rule, which is an Unsupervised Learning rule. Compare the following two:

$\Delta w_i \propto$ (Input x_i).(Error)	$\Delta w_i \propto$ (Input x_i).(output $\hat{y}$)
Supervised Learning	Unsupervised Learning

17.2 The Widrow-Hoff Learning Algorithm and Stochastic Approximation

Consider a linear output function for $\hat{y} = g(z)$:

(5) $\hat{y} = \sum_{i=1}^{n} w_i x_i$

Suppose that *N* sample data $\left\{ \left(y^j, x_1^j, ... x_n^j \right) \middle| \, j = 1, ... N \right\}$ are used for training the weights $w_1, ..., w_n$, so that the following mean squared error may be minimized:

(6) $J_N = \frac{1}{N} \sum_{i=1}^{n} (\hat{y}^j - y^j)^2$

Applying the gradient descent method yields

(7) $\Delta w_i = -\rho \cdot grad_{w_i} J_N = -\rho \frac{2}{N} \sum_{j=1}^{N} (\hat{y}^j - y^j) \frac{\partial \hat{y}^j}{\partial w_i}$

This method requires to store the gradient $\left(\hat{y}^j - y^j\right)\frac{\partial \hat{y}^j}{\partial w_i}$ for all the sample data $j = 1 \cdots N$ before making one correction to the weight. It is a type of batch processing.

An alternative method is to execute updating the weight Δw_i *every time* the training data is presented.

(8) $\Delta w_i[k] = \rho \delta[k] x_i[k]$ for the *k-th* presentation

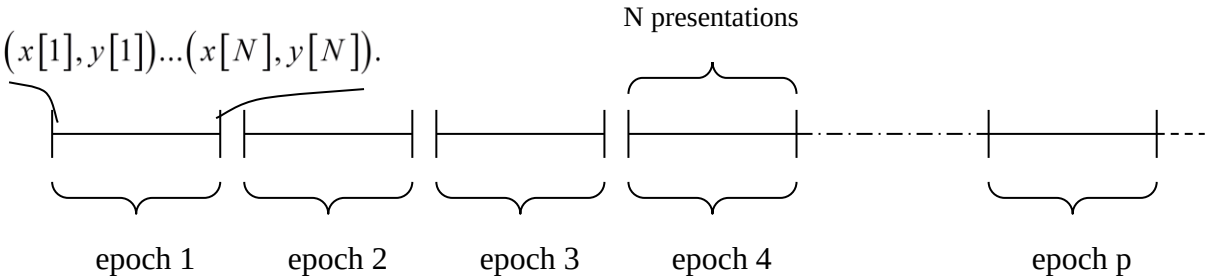
(9) where $\delta(k) = y[k] - \sum w_l[k] x_l[k]$

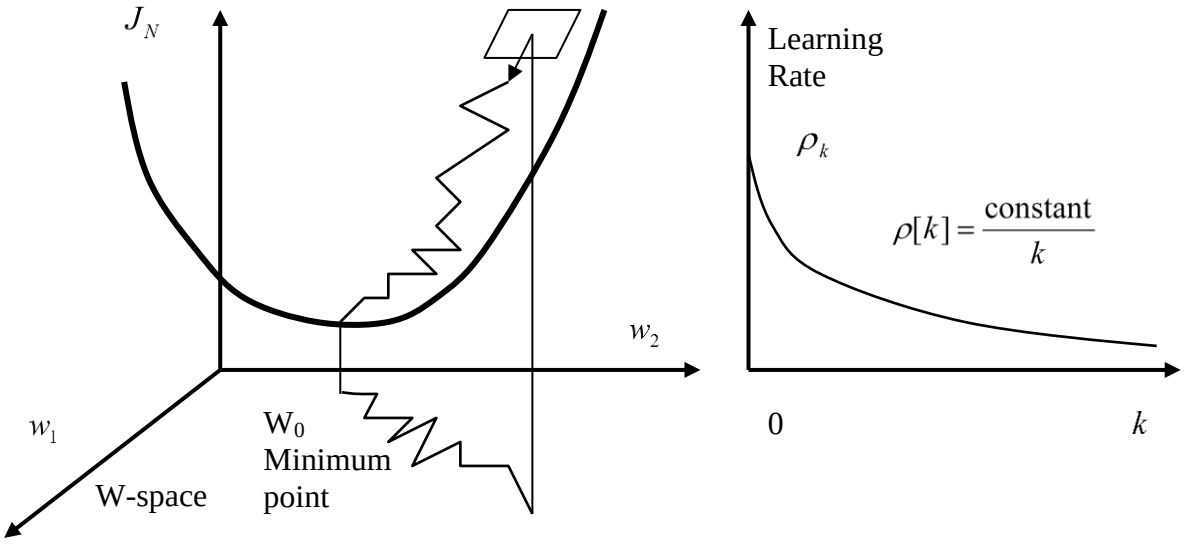
Correct output for the training data presented at the k-th time

Predicted output based on the weights $w_i[k]$ for the training data presented at the k-th time

More specifically, the learning procedure is stated as:

Present all the *N* training data in any sequence, execute (8) and (9) above for each presentation, and repeat the entire *N* presentations, called an "epoch", many times.





This learning procedure is called the Widrow-Hoff algorithm, for which convergence conditions have been obtained.

Convergence Problem: As the entire sequence of *N* presentations is repeated infinite times, do the weights $w_1,...,w_n$ converge to *the* optimal ones: $\arg\min_w J_N(w_1...w_n)$?

The following is a brief summary of convergence analysis based on Stochastic Approximation.

If a constant learning rate $\rho > 0$ is used, this does not converge, unless $\min J_N = 0$

If the learning rate is varied as a function of presentation number *k*, convergence can be guaranteed. See the above figure for $\rho[k] = (\text{constant})/k$.

It is known (Robbins and Monroe, 1951) that, if the variable learning rate $\rho[k]$ satisfies:

(10)

1). $\lim_{k\to\infty} \rho[k] = 0$,

2). $\lim_{k\to\infty} \sum_{i=1}^{k} \rho[i] = +\infty$ ⟶ This condition prevents all the weights from converging so fast that error will remain forever uncorrected.

3). $\lim_{k\to\infty} \sum_{i=1}^{k} \rho[i]^2 < \infty$ ⟶ This condition ensures that random fluctuations are eventually suppressed

then the estimated weights $w_1[k],...,w_n[k]$ converge to their optimal values $w_{10},...,w_{n0}$ with probability 1.

(11) $\lim_{k\to\infty} E[(w_i[k] - w_{i0})^2] = 0$;

This is a special case of the Method of Stochastic Approximation, where each training data *x* appears at probability $p(x)$. (In the Widrow-Hoff, it is a uniform distribution.)

The stochastic approximation minimizes the following expected loss function with respect to weight $w_1,...,w_n$:

(12) $E[L(w)] = \int L(x, y|w)p(x)dx$

(13) $L(x, y|w) = \frac{1}{2}\left(y - \hat{y}\left(x|w\right)\right)^2$

The learning rule is basically the same as (8) and (9).

(14) $w_i[k+1] = w_i[k] - \rho[k]\frac{\partial}{\partial w_i} L(x[k], y[k]|w[k])$

Convergence is guaranteed for learning rate $\rho[k]$ that satisfies conditions (10). This Stochastic Approximation method in general needs more presentations of data, i.e. the convergence process is slower than the batch processing. But the computation is very simple, requiring no large memory space.

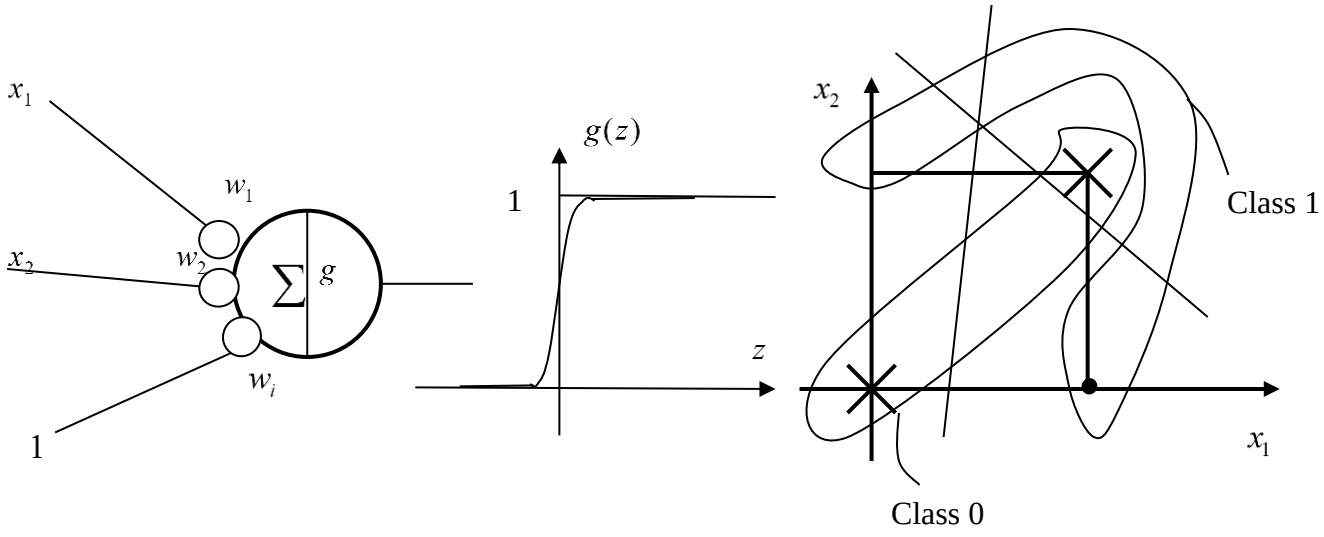
17.3 Multi-Layer Perceptrons

The Exclusive OR Problem

Input		Output
0	0	0
0	1	1
1	0	1
1	1	0
X_1	X_2	y

Can a single neural unit (perceptron) with weights $w_1, w_2, w_3,$ produce the XOR truth table?

No, it cannot



(15) $z = w_1 x_1 + w_2 x_2 + w_3$

Set z=0, then $0 = w_1 x_1 + w_2 x_2 + w_3$ represents a straight line in the $x_1 - x_2$ plane.

(16) $$g(z) = \begin{cases} 1 & z > 0 \\ 0 & z \le 0 \end{cases}$$

Class 0 and class 1 cannot be separated by a straight line. … Not linearly separable.

Consider a nonlinear function in lieu of (15)

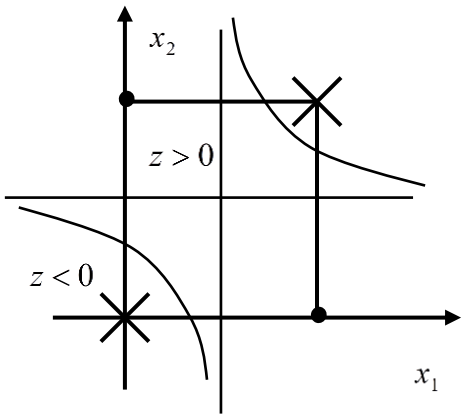
(17) $z = f(x_1, x_2) = x_1 + x_2 - 2x_1x_2 - \frac{1}{3}$

$f(0,0) = -\frac{1}{3}$

$f(1,1) = -\frac{1}{3}$

$\longrightarrow$ Class 0

$f(1,0) = f(0,1) = \frac{2}{3} > 0 \longrightarrow$ Class 1



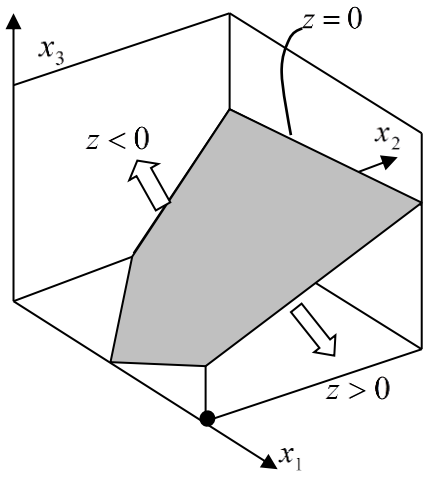
Next, replace x_1 x_2 by a new variable x_3

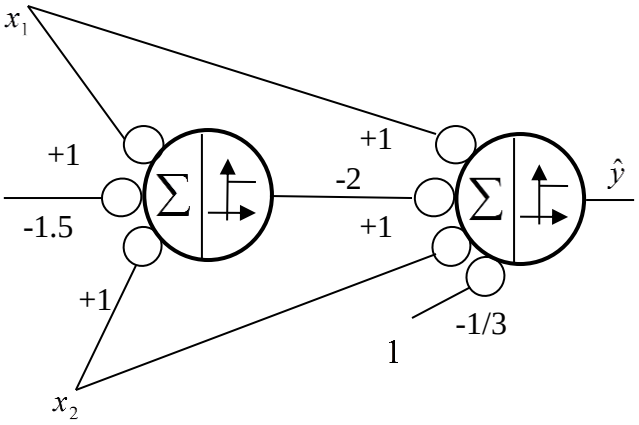
(18) $z = x_1 + x_2 - 2x_3 - \frac{1}{3}$

This is apparently a linear function: Linearly Separable.

Hidden Units

- Augment the original input patterns
- Decode the input and generate an internal representation

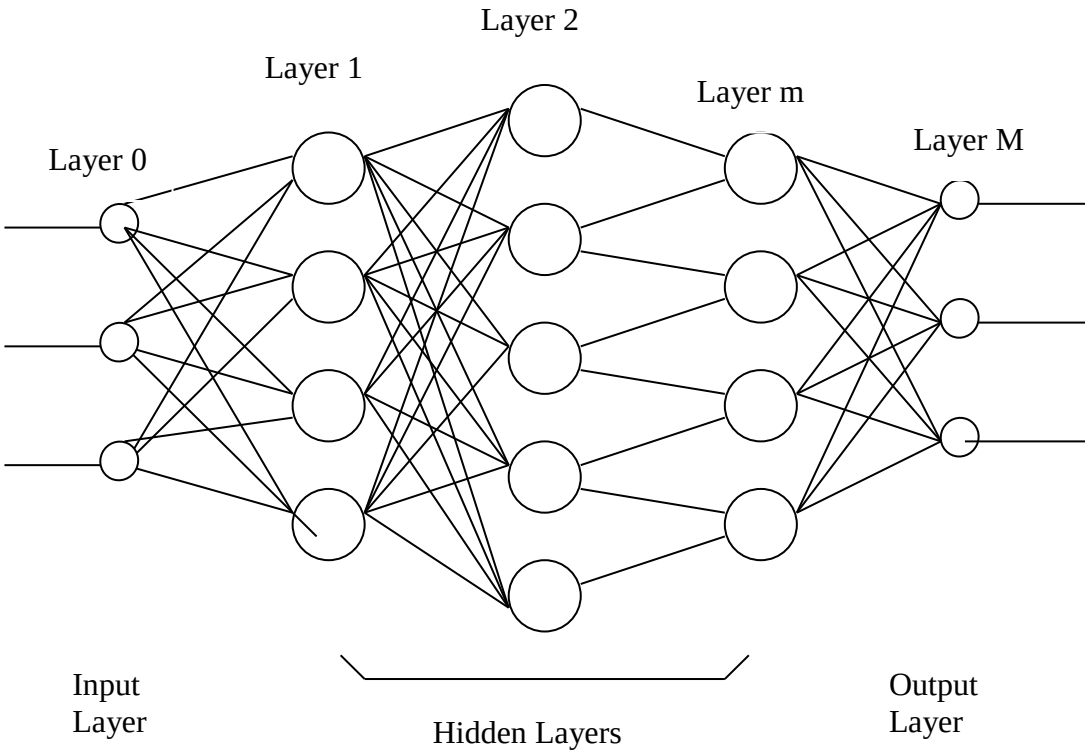




Hidden Unit
Not directly visible from output

Extending this argument, we arrive at a multi-layer network having multiple hidden layers between input and output layers.

Multi-Layer Perception



17.4 The Error Back Propagation Algorithm

The Multi-Layer Perception is a universal approximation function that can approximate an arbitrary (measurable) function to any accuracy.

Forward computation

$$z_j^{(m)} = \sum_i w_{ji}^{(m)} x_i^{(m)} \tag{19}$$

$$y_j^{(m)} = g_j(z_j^{(m)}) = x_j^{(m+1)} \tag{20}$$

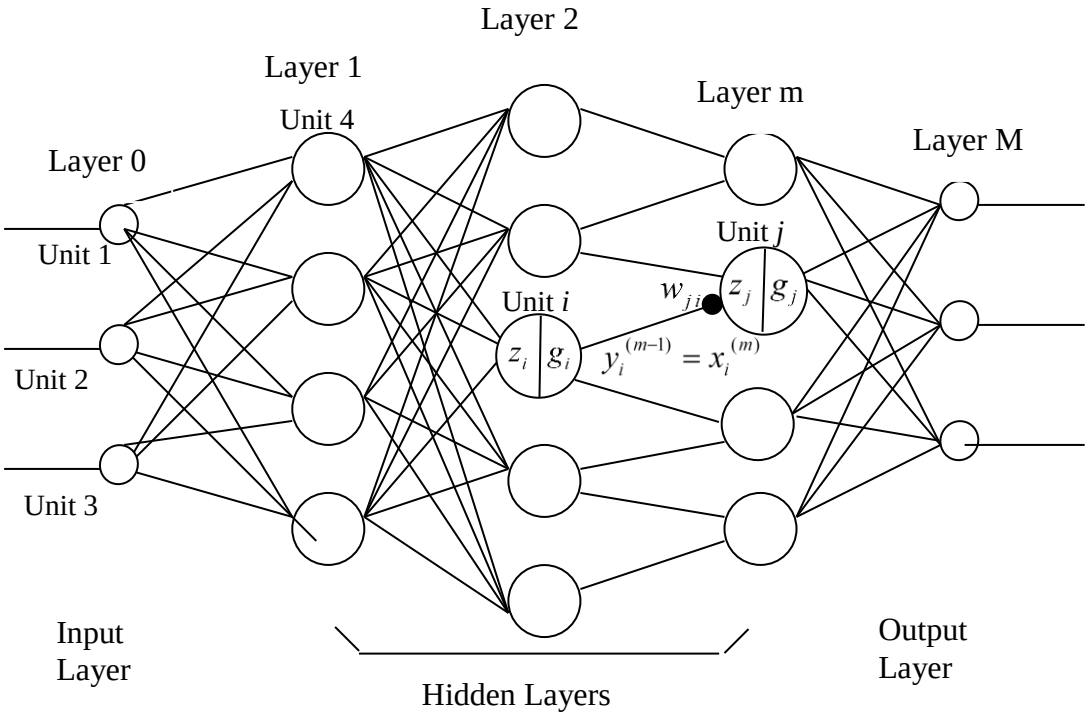
$m = 0, 1, 2, \ldots$

Starting from $m = 0$, all the units can be computed recursively towards the output layer: $m = M$.

Note: Multi-Layer Perceptrons with nonlinear activation functions, $g(z)$, are nonlinear in parameters w.

- A single-layer neural net is essentially linear in w, although $g(z)$ is nonlinear.
- If two consecutive layers have linear activation functions, they can be combined and replaced by a single layer network.

To be able to deal with nonlinear problems, such as the XOR problem, we now focus on a multi-layer perceptron with nonlinear activation functions.



$w_{ji}^{(m)}$ = weight of the connection from unit i to unit j in layer m

$y_j^{(m)}$ = output from unit j in layer m

$x_i^{(m)}$ = input to a unit in layer m from unit i

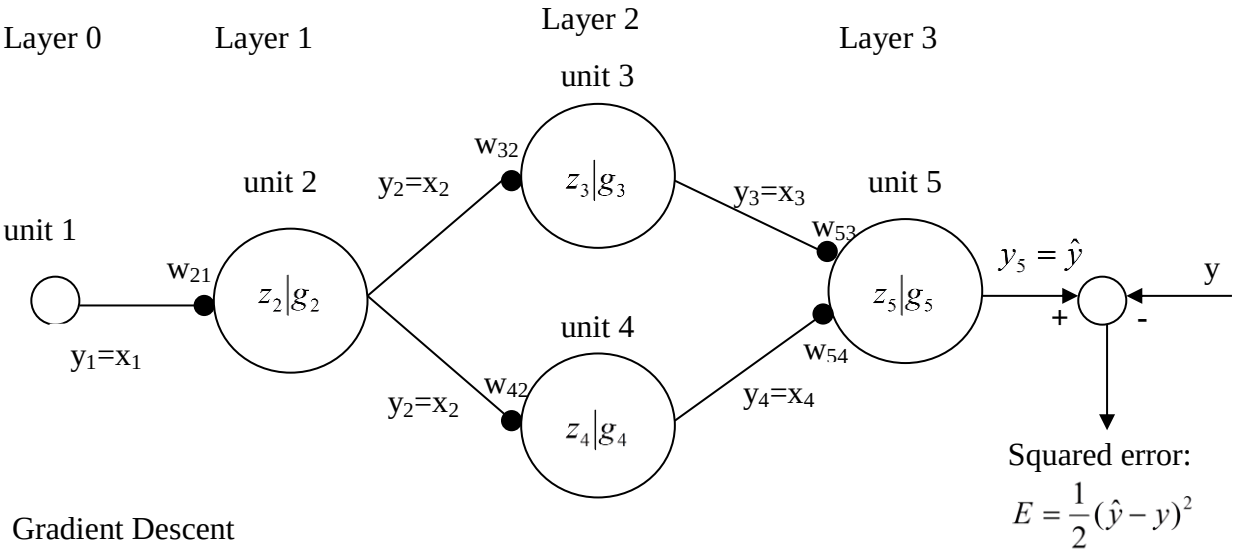
How do we train the multi-layer perceptron, given training data presented sequentially?

The theory of stochastic approximation is not applicable, since the parameters are not linearly involved in the predictor. However, the Gradient Descent Method (The Widrow-Hoff algorithm) can be extended to multi-layer perceptrons.

The algorithm is called the ***Error Backpropagation*** Algorithm.

Example

Consider a three-layer perceptron in order to derive a basic formula of error backpropagation.



Gradient Descent

$$\Delta w_{53} = -\rho \cdot grad_{w_{53}} E$$

$$= -\rho \frac{\partial E}{\partial y_5} \frac{dy_5}{dz_5} \frac{\partial z_5}{\partial w_{53}} \tag{21}$$

$$= -\rho \underbrace{(\hat{y} - y) g'_5(z_5)}_{\| \; -\delta_5 = \frac{\partial E}{\partial z_5}} \cdot x_3 = \rho \delta_5 x_3$$

$$\Delta w_{32} = -\rho \cdot grad_{w_{32}} E \overbrace{\quad w_{53} \quad \frac{dg_3}{dz_3}}$$

$$= -\rho \frac{\partial E}{\partial y_5} \frac{dy_5}{dz_5} \frac{\partial z_5}{\partial x_3} \frac{\partial x_3}{\partial z_3} \frac{\partial z_3}{\partial w_{32}} \tag{22}$$

$$= \rho \underbrace{\delta_5 w_{53} g'_3(z_3)}_{\| \; \delta_3} x_2$$

Likewise,

$$\Delta w_{42} = \rho \underbrace{\delta_5 w_{54} g'_4(z_4)}_{\| \; \delta_4} x_2 \tag{23}$$

: There are two routes between z_5 and w_{21}.

$$
\begin{aligned}
\Delta w_{21} &= -\rho \cdot grad_{w_{21}} E \\
&= -\rho \frac{\partial E}{\partial z_5}\left(\frac{\partial z_5}{\partial x_3}\frac{\partial x_3}{\partial x_2}\frac{\partial x_2}{\partial w_{21}} + \frac{\partial z_5}{\partial x_4}\frac{\partial x_4}{\partial x_2}\frac{\partial x_2}{\partial w_{21}}\right) \\
&= \rho(\underbrace{\delta_5 w_{53} g'_3(z_3)}_{\delta_3} w_{32} + \underbrace{\delta_5 w_{54} g'_4(z_4)}_{\delta_4} w_{42})\frac{\partial y_2}{\partial w_{21}} \qquad (24) \\
&= \rho(\delta_3 w_{32} + \delta_4 w_{42}) g'_2(z_2) x_1
\end{aligned}
$$

The above computation can be streamlined by computing δ_j, starting from the final layer back to the first layer.

Error $(\hat{y} - y)$ is propagated backward…
That's why it is called the Error Backpropagation Algorithm.

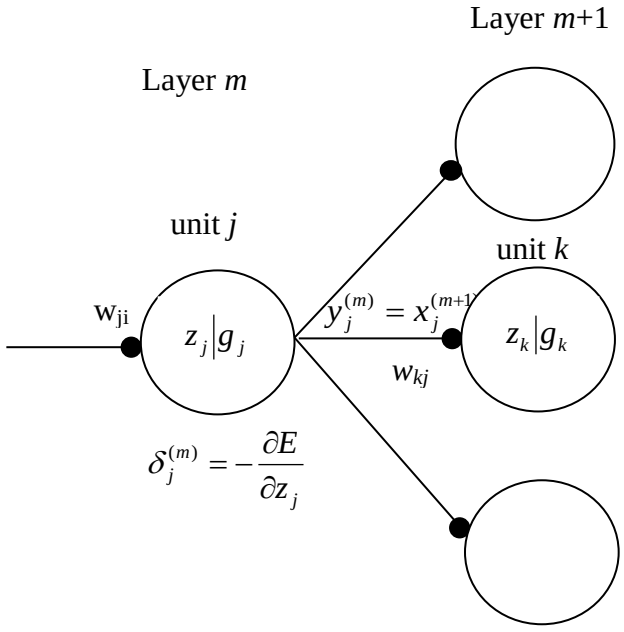
In general,

For the final layer, $m = M$,

$$\Delta w_{ji}^{(M)} = \rho \delta_j^{(M)} x_i^{(M)} \qquad (25)$$

$$\delta_j^{(M)} = (y - \hat{y}^{(M)}) g'^{(M)}_j [z_j^{(M)}] \qquad (26)$$

For hidden layers, $1 \le m \le M-1$



$$\Delta w_{ji} = -\rho\, grad_{w_{ji}} E = -\rho \frac{\partial E}{\partial z_j}\frac{\partial z_j}{\partial w_{ji}} = \rho \delta_j^{(m)} x_i^{(m)} \tag{27}$$

$$\begin{aligned}\delta_j^{(m)} &= -\frac{\partial E}{\partial z_j} \\ &= -\frac{\partial E}{\partial x_j^{(m+1)}}\frac{\partial y_j^m}{\partial z_j} = g'_j(z_j)(-\sum_k \frac{\partial E}{\partial z_k}\frac{\partial z_k}{\partial x_j^{(m+1)}}) \\ &= g'_j(z_j)\sum_k \delta_k^{(m+1)} w_{kj}^{(m+1)}\end{aligned} \tag{28}$$

Forward Input Propagation

Move from m = 1 to M

$$z_j^{(m)} = \sum_j w_{ji}^{(m)}\, x_i^{(m)}\ ,\ y_j^{(m)} = g_j(z_j^{(m)}) = x_j^{(m+1)} \tag{29}$$

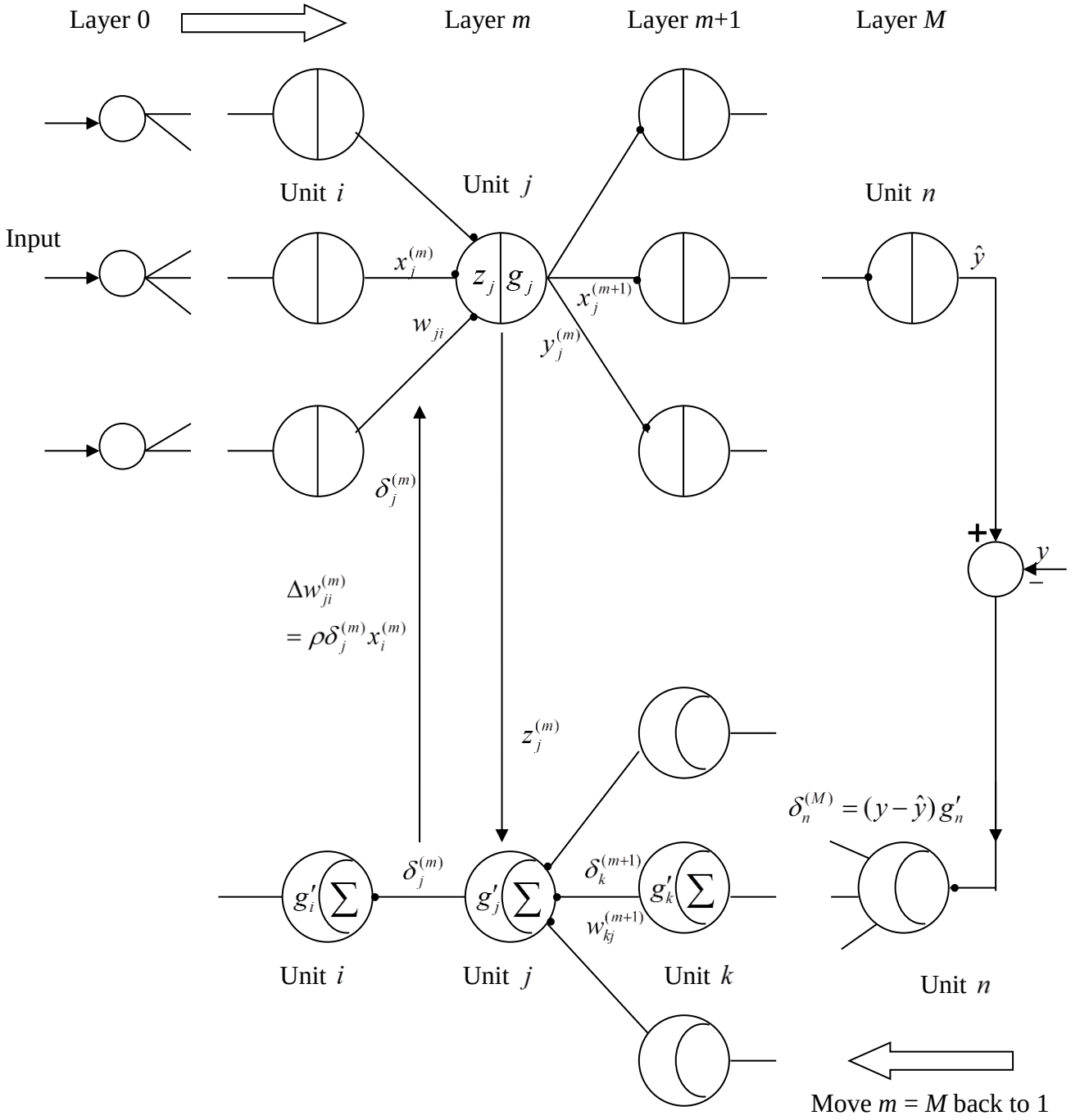
Error Back propagation

Move backward from $m = M$ to 1

$$\begin{aligned}\delta_j^{(m)} &= g'_j(z_j^{(m)})\sum_k \delta_k^{(m+1)} w_{kj}^{(m+1)} \\ \delta_n^{(M)} &= \left(y - \hat{y}^{(M)}\right) g'_n(z_n^{(M)})\end{aligned} \tag{30}$$

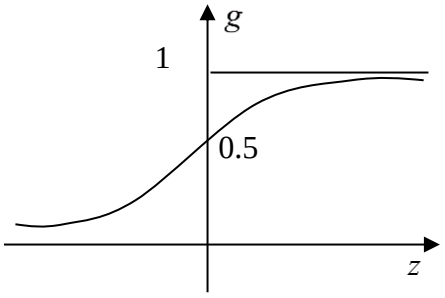
The Error Backpropagation Algorithm

[Wobas 1974, 1994] [Rumelhart, Hinton, & Williams,1986]



17.5 Stabilizing Techniques

1). Properties of the sigmoid function



$$g(z) = \frac{1}{1+e^{-z}} \tag{31}$$

Nonlinear, differentiable

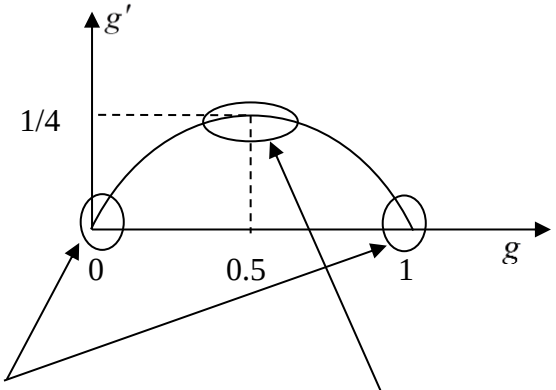
$$g'(z) = \frac{e^{-z}}{(1+e^{-z})^2} = \frac{1}{1+e^{-z}}\left(1 - \frac{1}{1+e^{-z}}\right) = g(1-g) \tag{32}$$

$$\Delta w_{ji} = \rho \delta_j^{(m)} x_i^{(m)}$$

$$\delta_j^{(m)} = g'_j \sum_k \delta_k^{(m+1)} w_{kj}^{(m+1)}$$

(33) $\Delta w_{ji} \propto g'_j(z_j)$

The incremental weight change is proportional to the derivative of $g(z)$.



For $-\infty < z < \infty$.
g varies $0 < g < 1$.
Max $g' = 1/4$
at $z = 0$ $g = 0.5$

In these ranges
weight changes are small.
$g \cong 0$ or $g \cong 1$
$|z| >> 1.$
$z_j = \sum_i w_{ji} x_i$

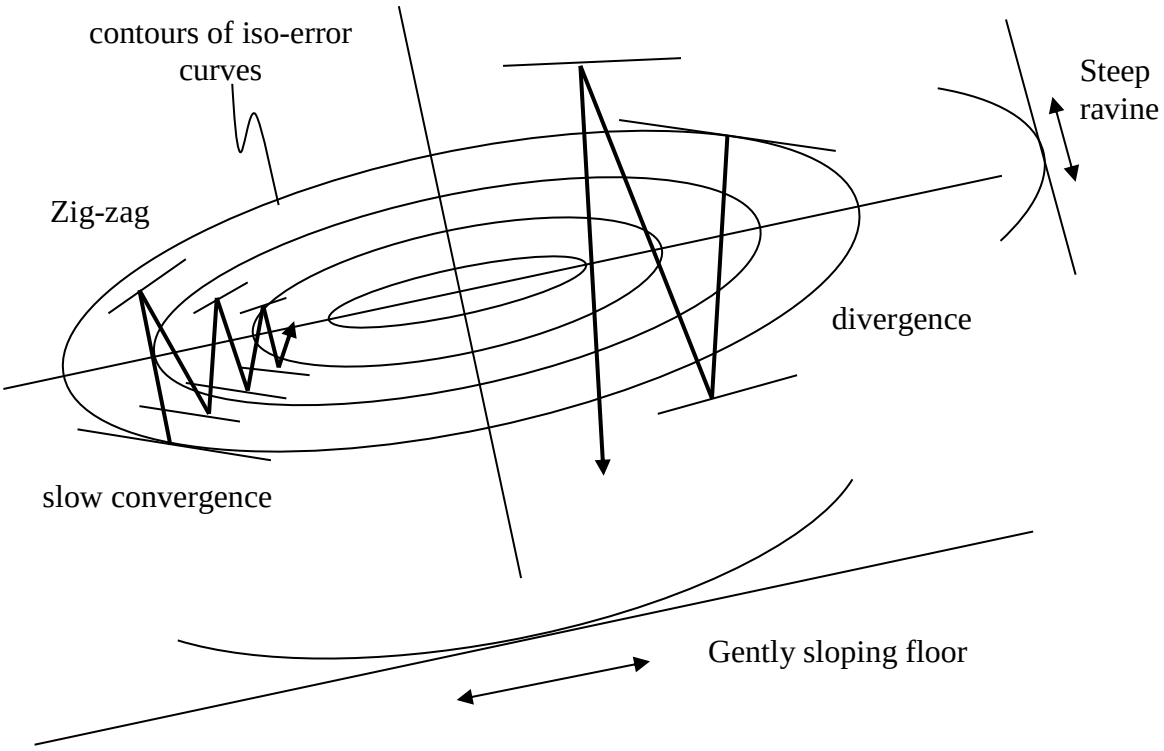
The largest weight change occurs in this range.
$g = 0.5, z = 0$
The unit has committed to neither 0 nor 1.
The error backpropagation algorithm forces the unit to react significantly to that input.

Once the unit (j) has committed to take an output value of either "0" or "1", the weight w_{ji} will no longer change very much for new inputs.

These properties contribute to stabilizing the learning process

2) Smoothing by adding a momentum term

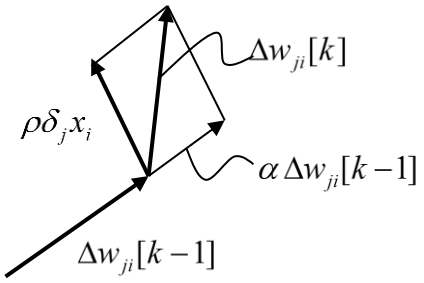
Ravine: a typical failure scenario of convergence



(34) $$\Delta w_{ji}[k] = \rho \delta_j x_i + \alpha \Delta w_{ji}[k-1]$$

proportionality constant

Previous weight change



This second term, called a momentum, filters out high frequency oscillations.

3) How to get rid of local minima

- Increase the number of hidden units
- Randomize the initial weights and repeat learning, then take the best one.