2.160 Identification, Estimation, and Learning

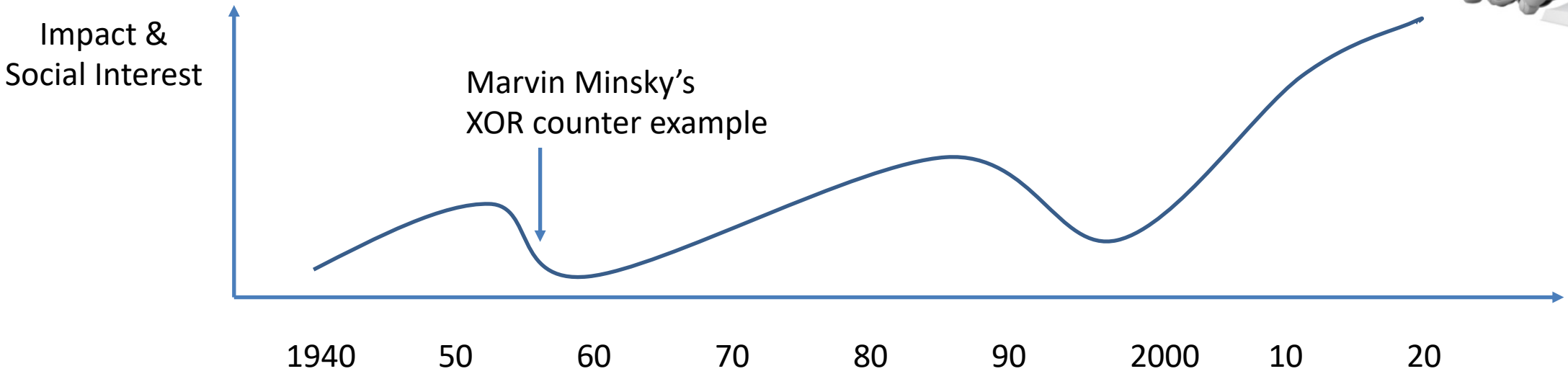
Part 4 Machine Learning and Nonlinear System Modeling

Lecture 20

Neural Networks and Error Backpropagation

H. Harry Asada
Department of Mechanical Engineering
MIT

Neural Networks



Neuron Model

- McCullock & Pitts, 1943
- Rosenblatt, 1958: Perceptron
- Widrow & Hoff, 1960: Stochastic Grad. Descent

Multi-layer Neural Nets

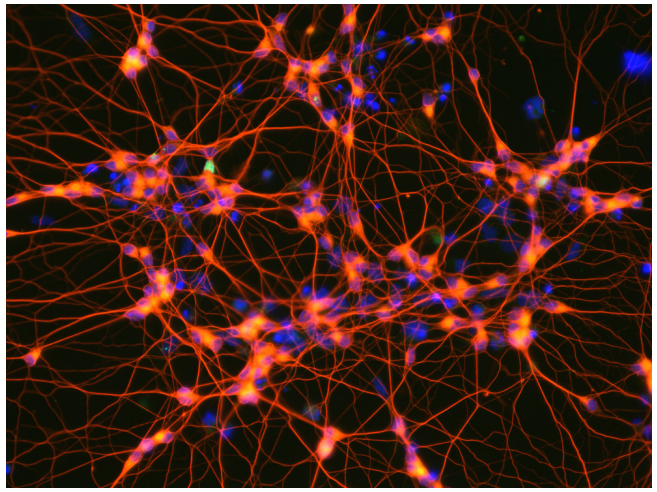
- Error Backpropagation
- Rumelhart & McClelland, 1986; Werbos, 1975
- Reinforcement Learning, Sutton 1984

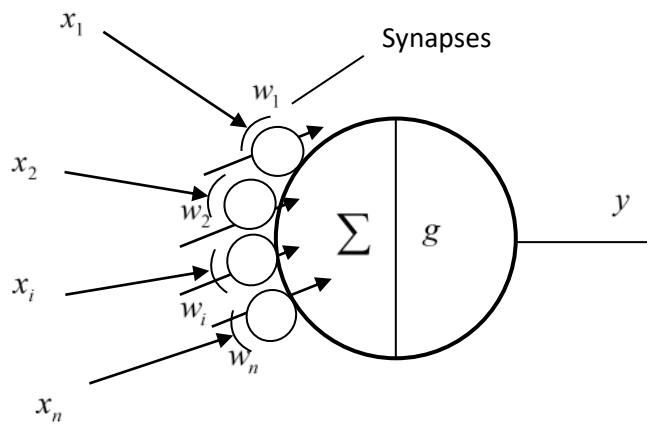
Deep Learning

- Convolutional Neural Nets (CNN), Fukushima 1980
- Recurrent Neural Nets, Elmer 1990, Jordan 1997.
- Long Short-Term Memory (LSTM) net, Hochreiter & Schmidhuber 1997
- GPU
- Successful applications in voice recognition, image processing

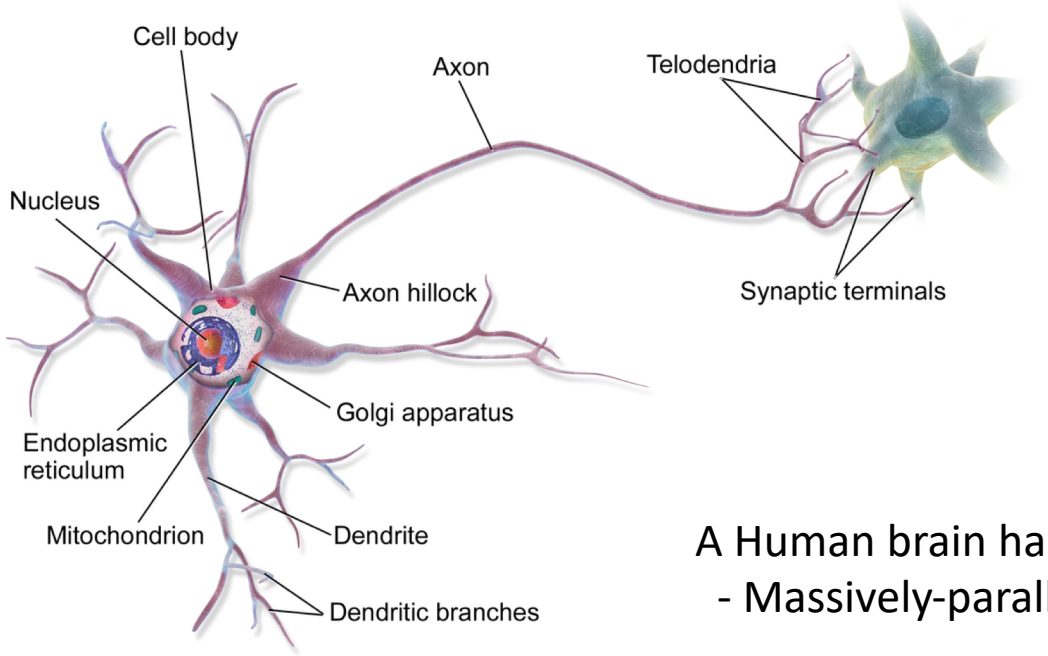
Outline

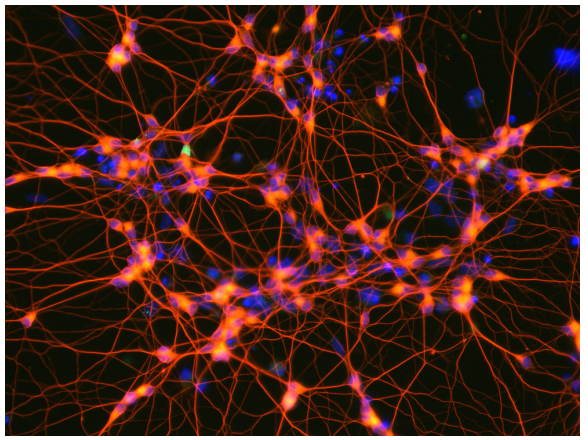
- Artificial Neural Network
  - Basic neuron model
  - Gradient descent
  - Nonlinear classification : XOR problem
  - Multi-layer neural network
  - The Error Back Propagation Algorithm
  - Properties of and tips for neural net training



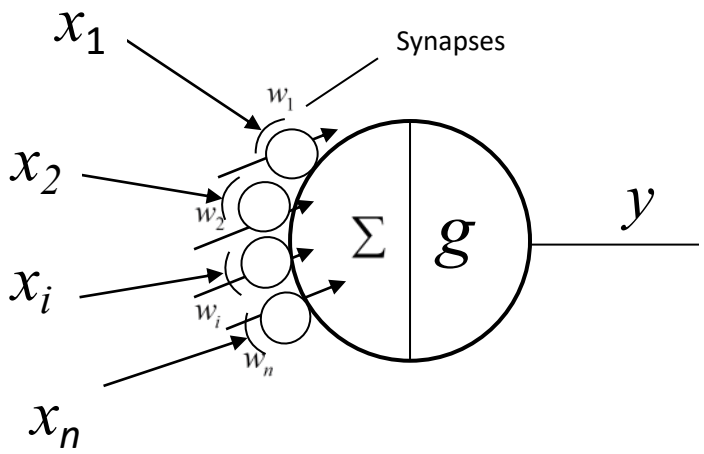


Neuron Model





A Human brain has approximately 14 billion neurons.
- Massively-parallel, distributed processing -



The Hebbian Rule

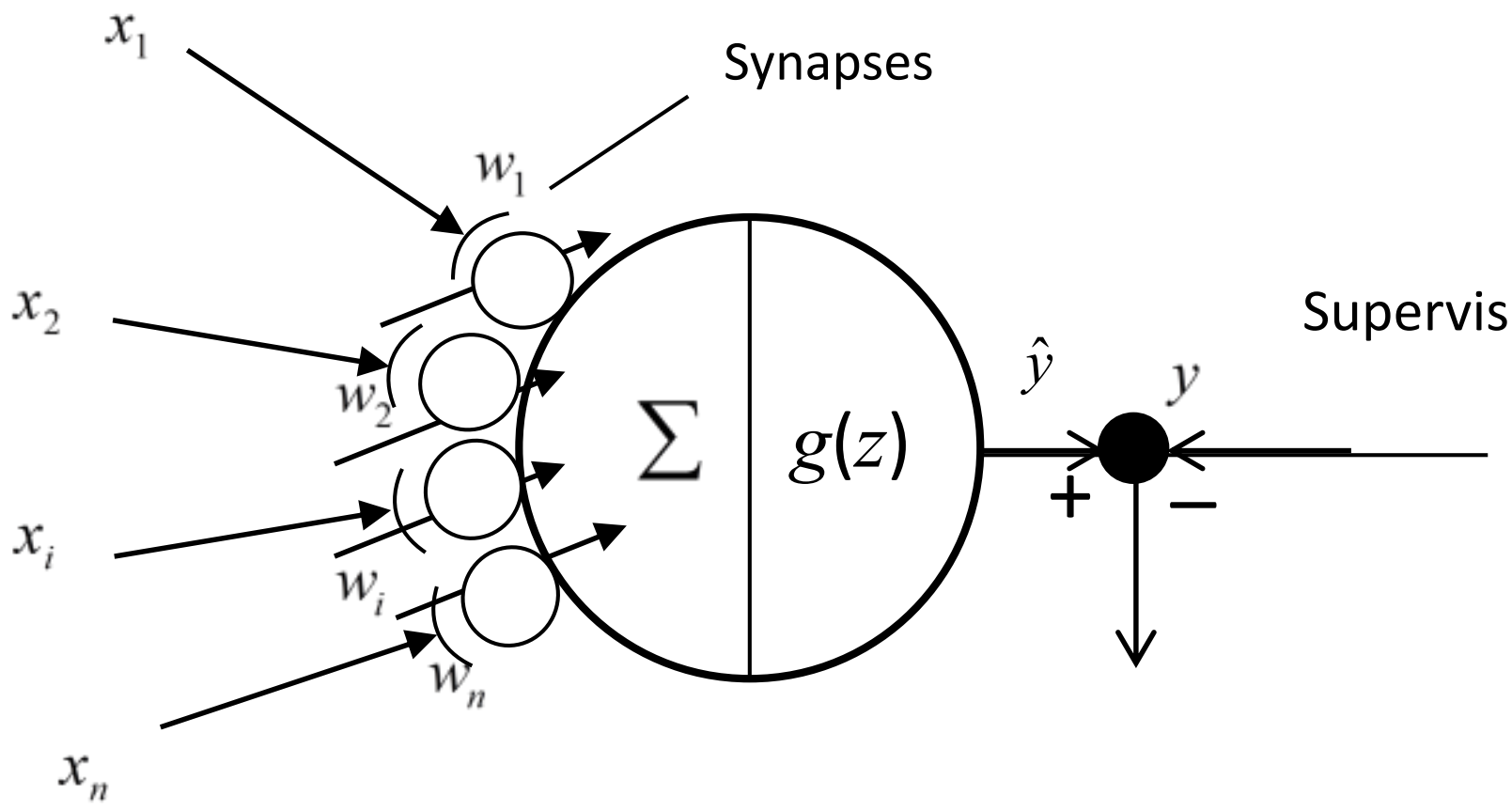
Input x_i out put y

fired fired

The i-th synapse w_i is reinforced.

Donald Hebb, 1949

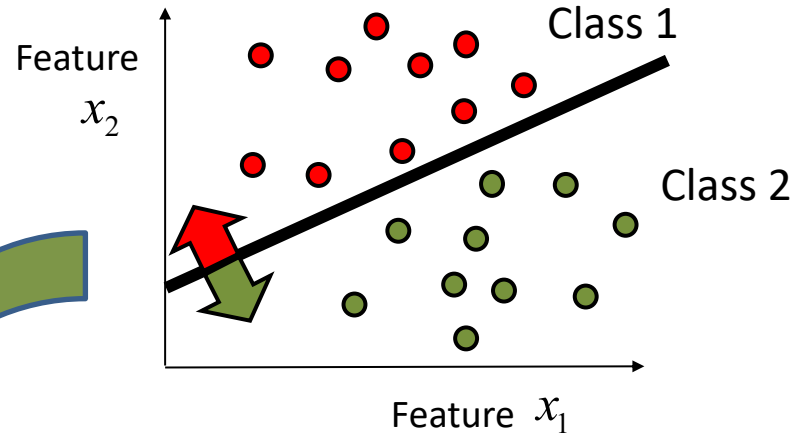
Artificial Neuron Model

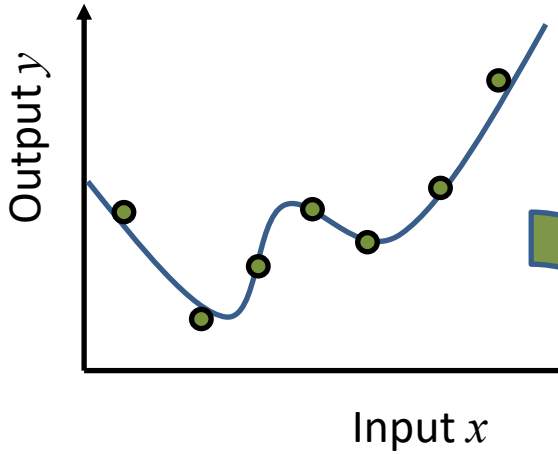


Weighted sum of inputs: $z = \sum_{i=1}^{n} w_i x_i$ Output Function: $\hat{y} = g(z)$

Supervised Learning

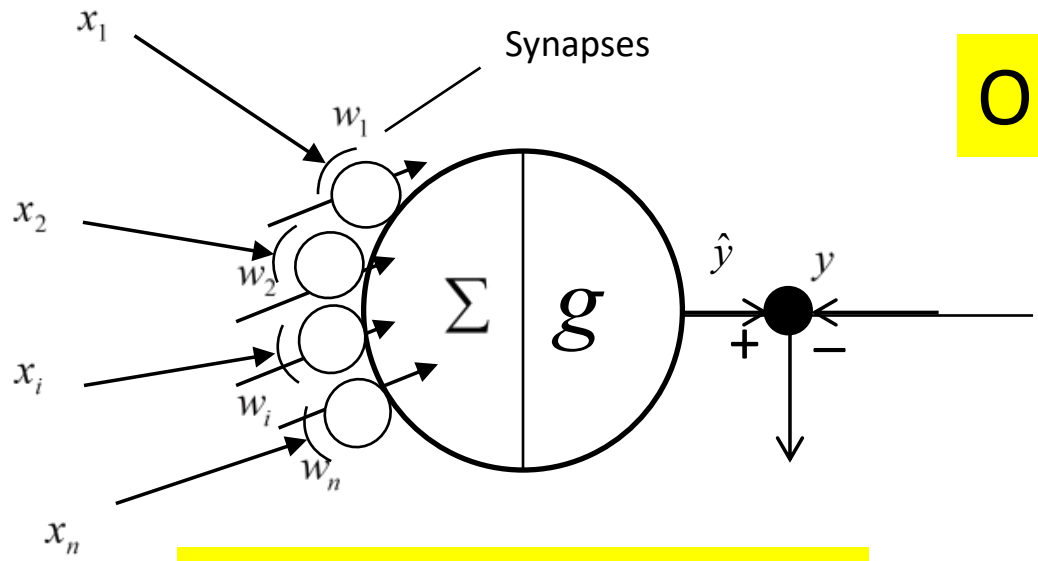
Classification:
Discrete output





Regression:
Continuous output

Input Data

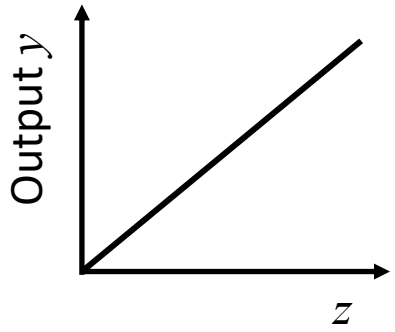


Output Data

Prediction Error Method

Neural Net Training Based on Gradient Descent

Weighted sum of inputs: $z = \sum_{i=1}^{n} w_i x_i$



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

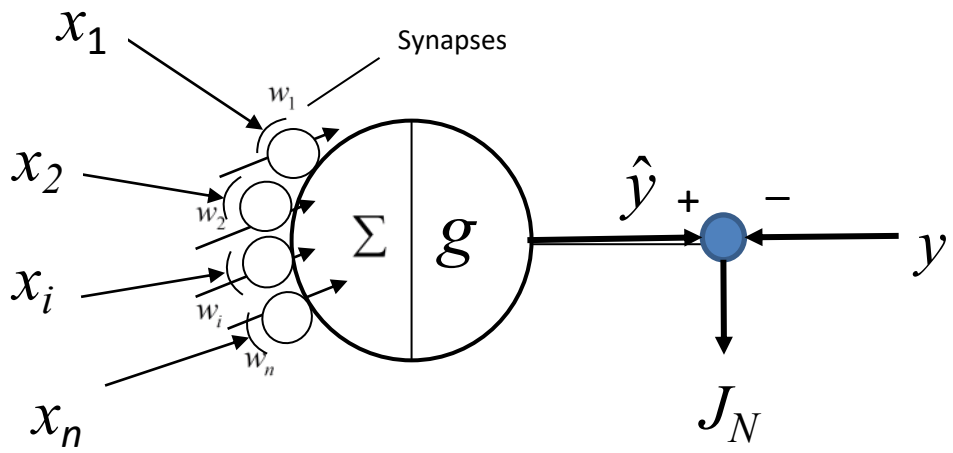
$$\hat{y} = \sum_{i=1}^{n} w_i x_i$$

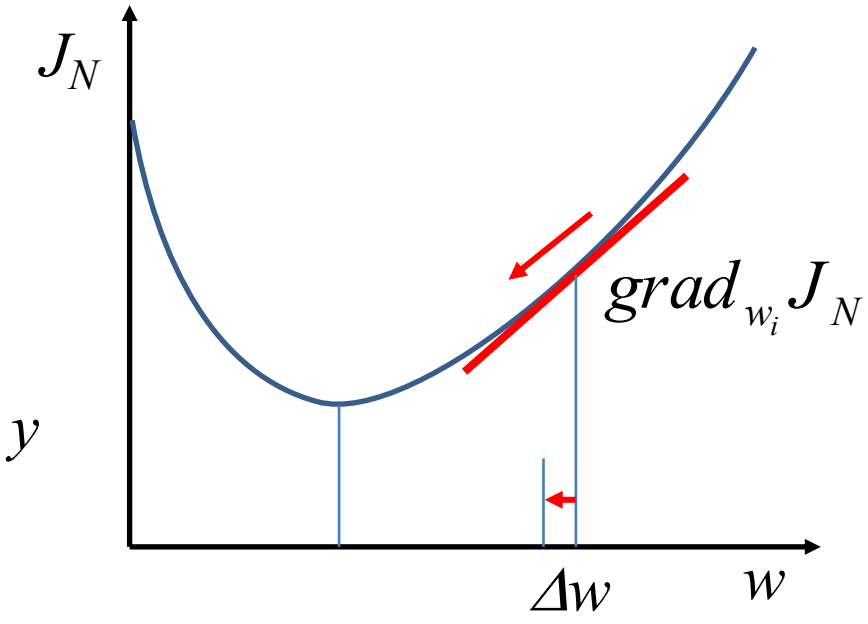
Training Data $\left\{ \left(y^j, x_1^j, \ldots x_n^j \right) \middle| \, j = 1, \ldots N \right\}$

Find weights using the training data, so that the squared error may be minimum.

Apply the Gradient Descent Method

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





Neural Net Training Based on Gradient Descent

Taking partial derivative of the squared error:

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

Learning rate

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

We assume a linear output function:

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

Therefore, the weight change is given by

$$\Delta w_i = -\rho \frac{2}{N} \sum_{j=1}^{N} \underbrace{(\hat{y}^j - y^j)}_{\delta} x_i^j$$

$$\Delta w_i \propto \text{(Prediction Error) x (Input)}$$

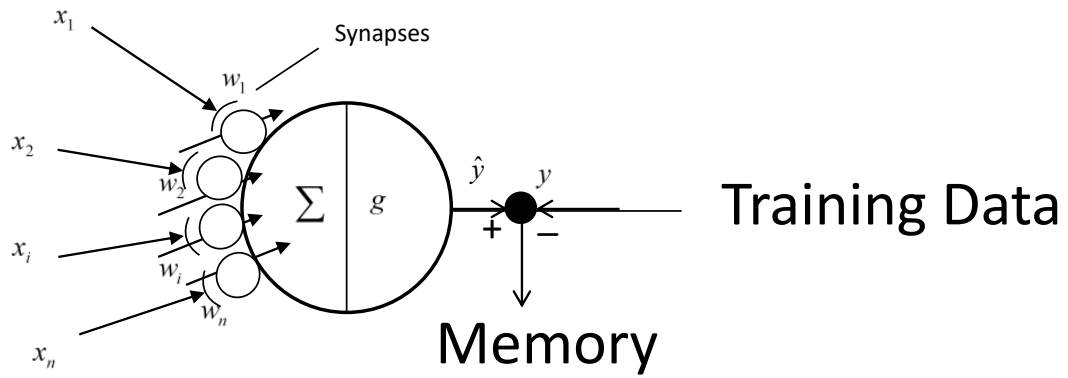
This resembles Recursive Least Squares, Kalman Filter, etc.

Neural Net Training Based on Gradient Descent

$$\Delta w_i = -\rho \frac{2}{N} \sum_{j=1}^{N} (\hat{y}^j - y^j) x_i^j \qquad \hat{y} = \sum_{i=1}^{n} w_i x_i$$

Drawbacks of this algorithm are:

- Until you present all the training data, you cannot make any correction to the weights.
- As the size of the training data increases, a large memory space is required to store the results.



The Delta Method: An alternative to the global gradient descent

The Delta Method: An alternative to the global gradient descent

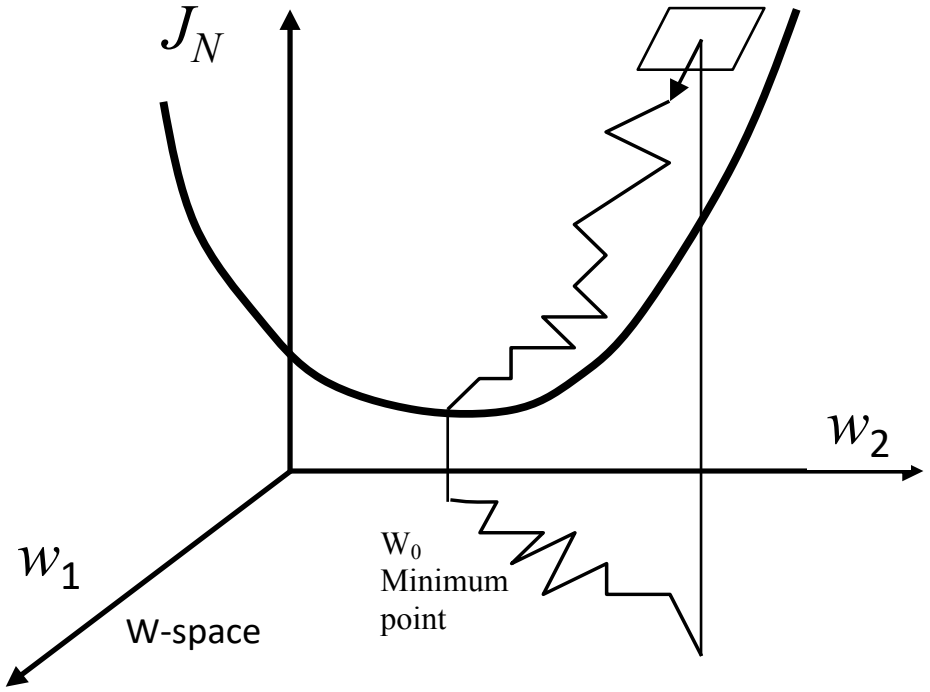
(8) $\Delta w_i[k] = \rho\delta[k]x_i[k]$

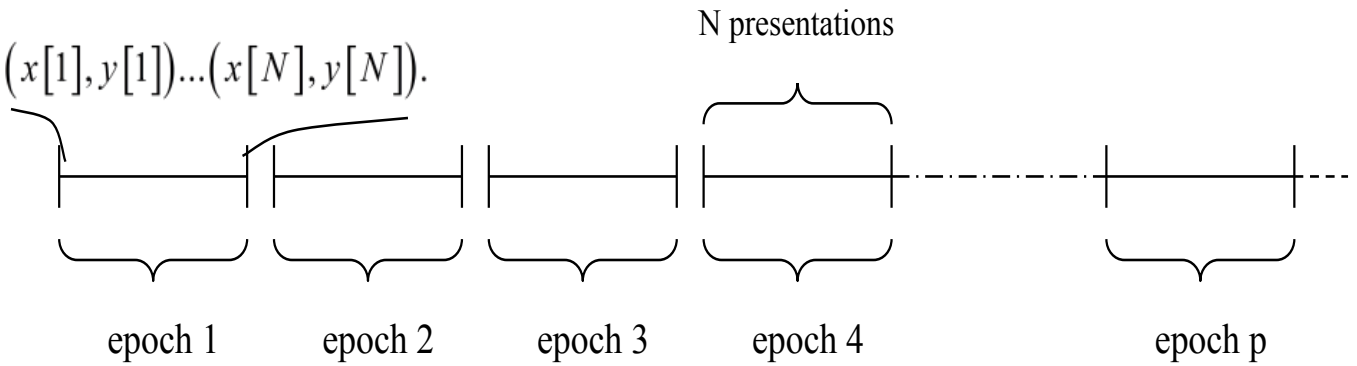
(9) where $\delta(k) = y[k] - \sum w_l[k]x_l[k]$

Make a quick correction to the weights for each presentation of the individual training data.

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





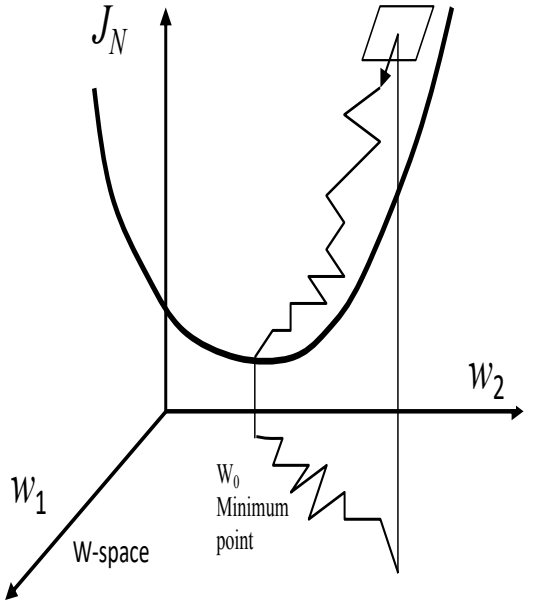
N training data are randomly presented to the neural net, and make weight changes N times. Repeat this sequence of N presentations, called an epoch, many times until it converges.

The Widrow-Hoff Algorithm: Stochastic Gradient Descent

$$\Delta w_i[k] = \rho \delta[k] x_i[k]$$

$$\text{where } \delta(k) = y[k] - \sum w_l[k] x_l[k]$$

- Compared to the full Gradient Descent method (batch processing), the Widrow-Hoff algorithm may be erratic in each step of weight correction, since it evaluates the gradient based on only one data point (one example);
- But, no need to store each presentation result; much quicker in making corrections, particularly for a large training data set. → This property has led to Massively Parallel and Distributed Processing, an important feature of Neural Nets.

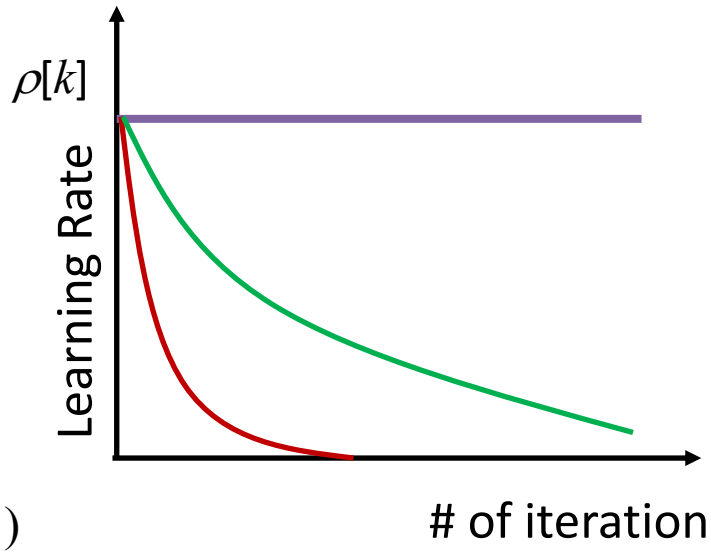


Question: Does it converge? Where to converge?

Convergence Analysis of Stochastic Gradient Descent

For linear output functions, Convergence Conditions have been obtained.

- With a constant learning rate ρ, the learning does not converge.
- The learning rate $\rho[k]$ must be varied.
- All the weights converge to their optimal values with probability 1, when the following conditions are met



1). $\lim_{k\to\infty} \rho[k] = 0$, (Robbins and Monroe, 1951)

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

Example:

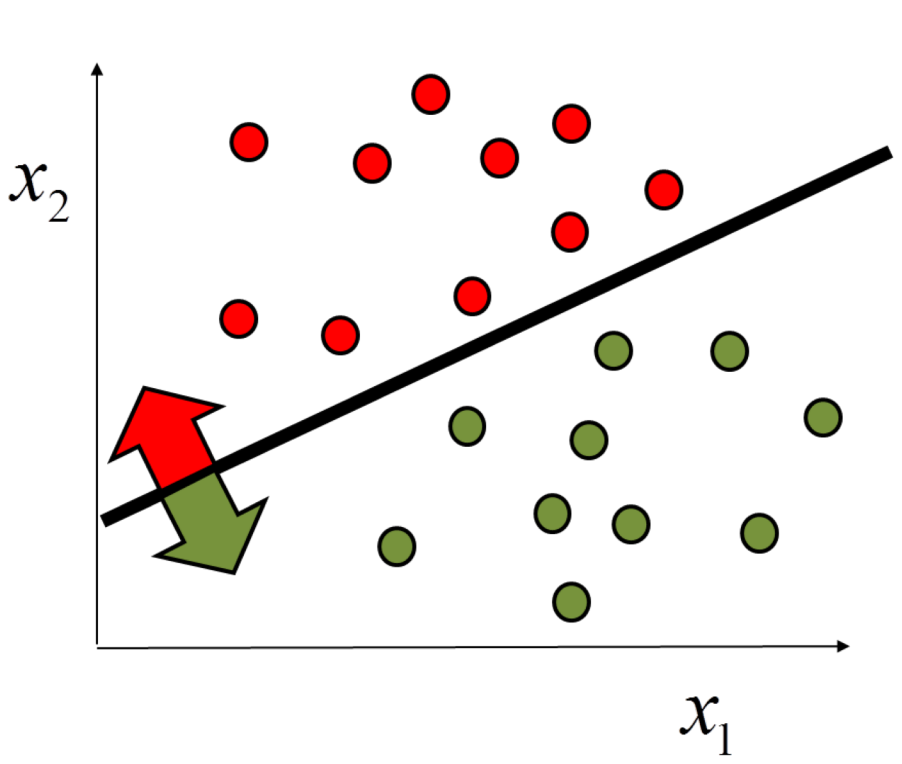
$$\rho[k] = \frac{c}{k}$$

This meets the three conditions.

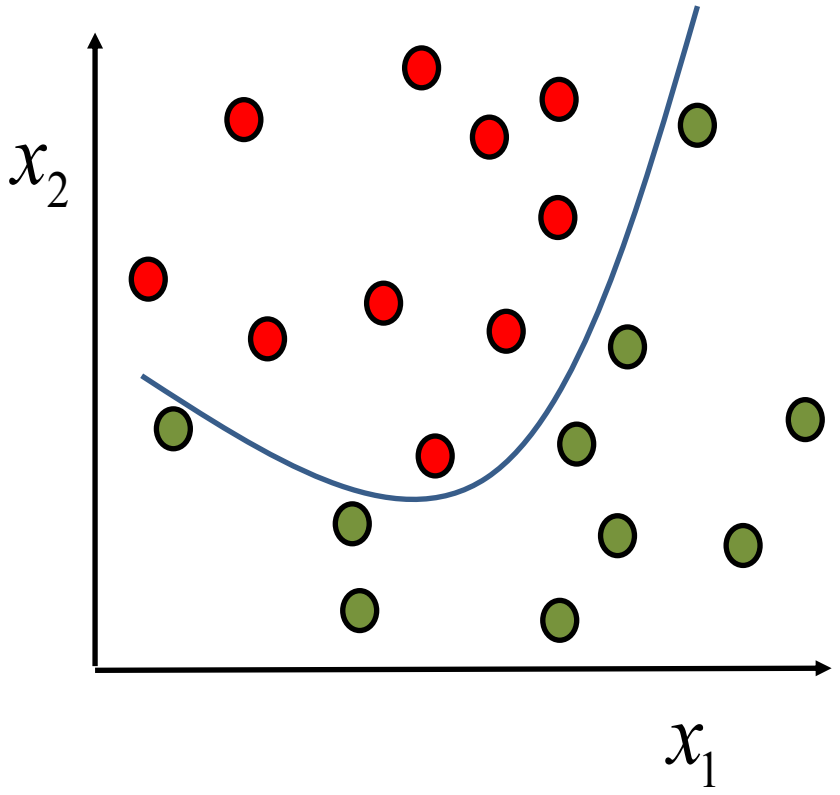
Linear Separability and Multi-Layer Neural Networks

Limitation to Rosenblatt's Perceptron and the birth of Multi-Layer Neural Network

Linear Separable Case



Linearly separable.



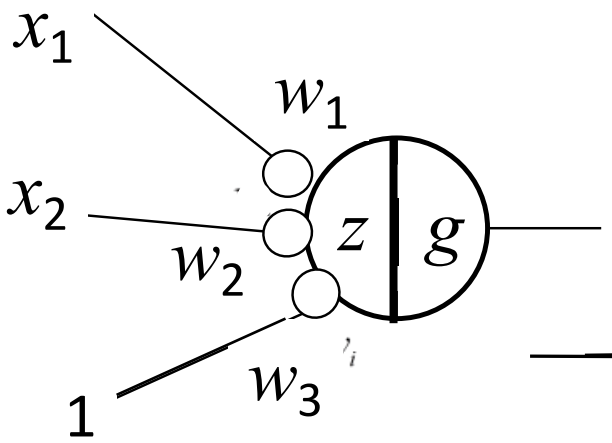
Not linearly separable.

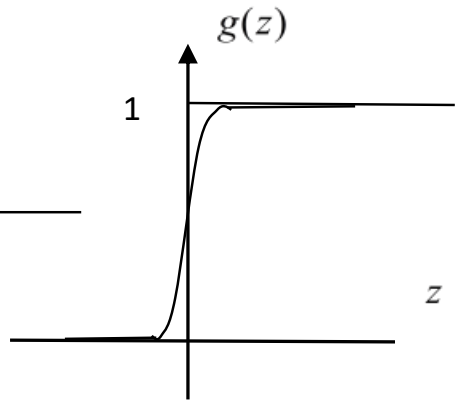
The Exclusive OR Problem

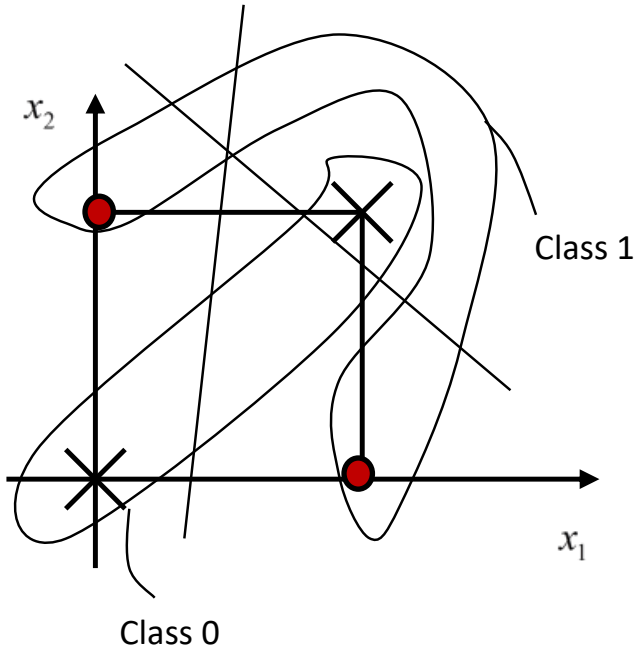
Input		Output
0	0	0
0	1	1
1	0	1
1	1	0
X_1	X_2	y

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

No, you cannot.







$$z = w_1 x_1 + w_2 x_2 + w_3$$

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.

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

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

Consider a nonlinear function

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

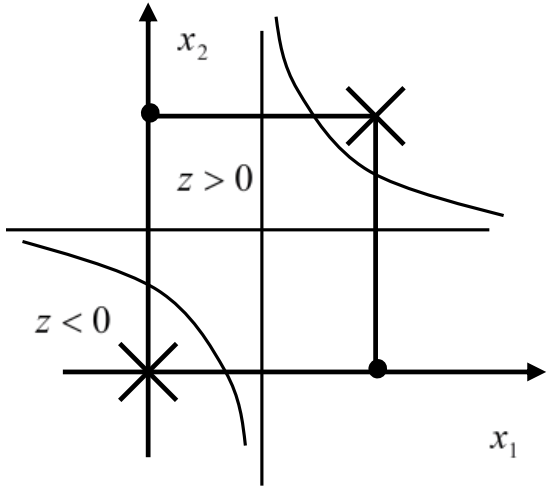
$$f(0,0) = -\frac{1}{3}$$

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

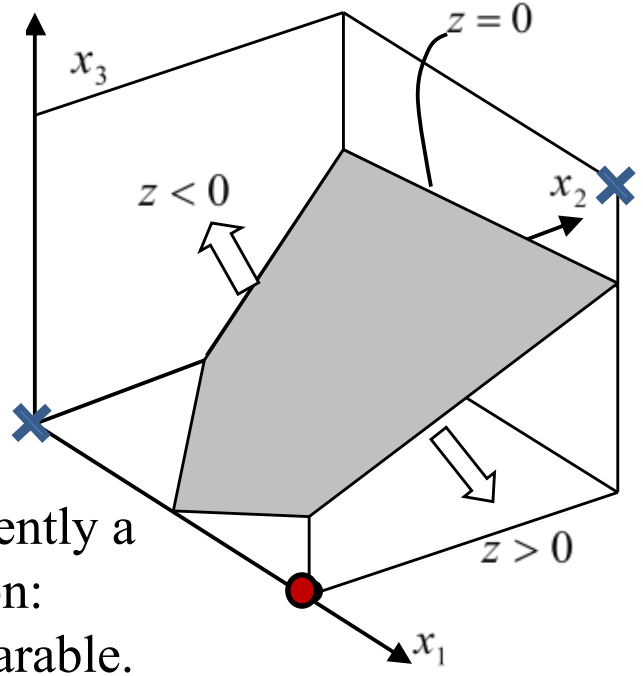
→ Class 0

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

→ Class 1

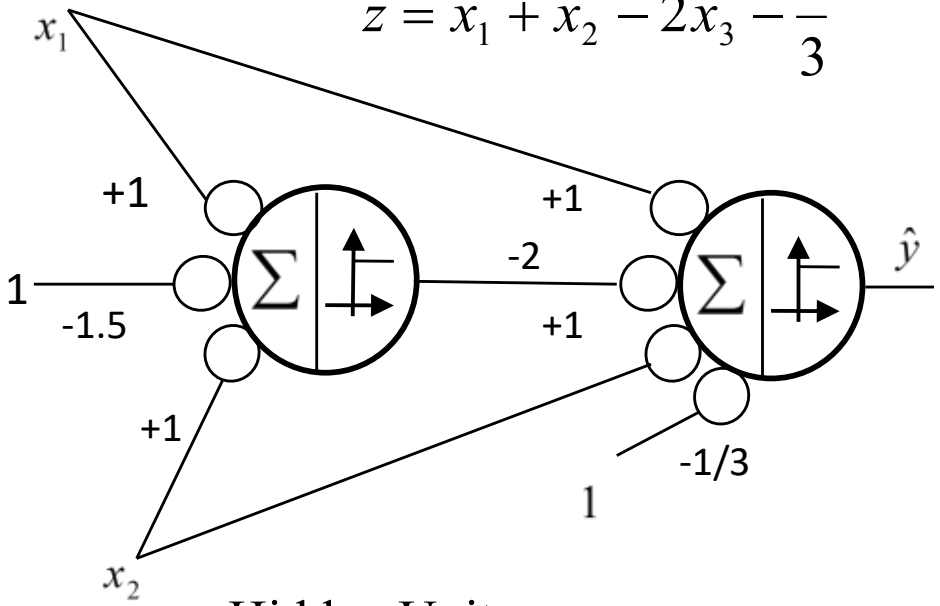


replace x_1 x_2 by a new variable x_3



This is apparently a linear function:
Linearly Separable.

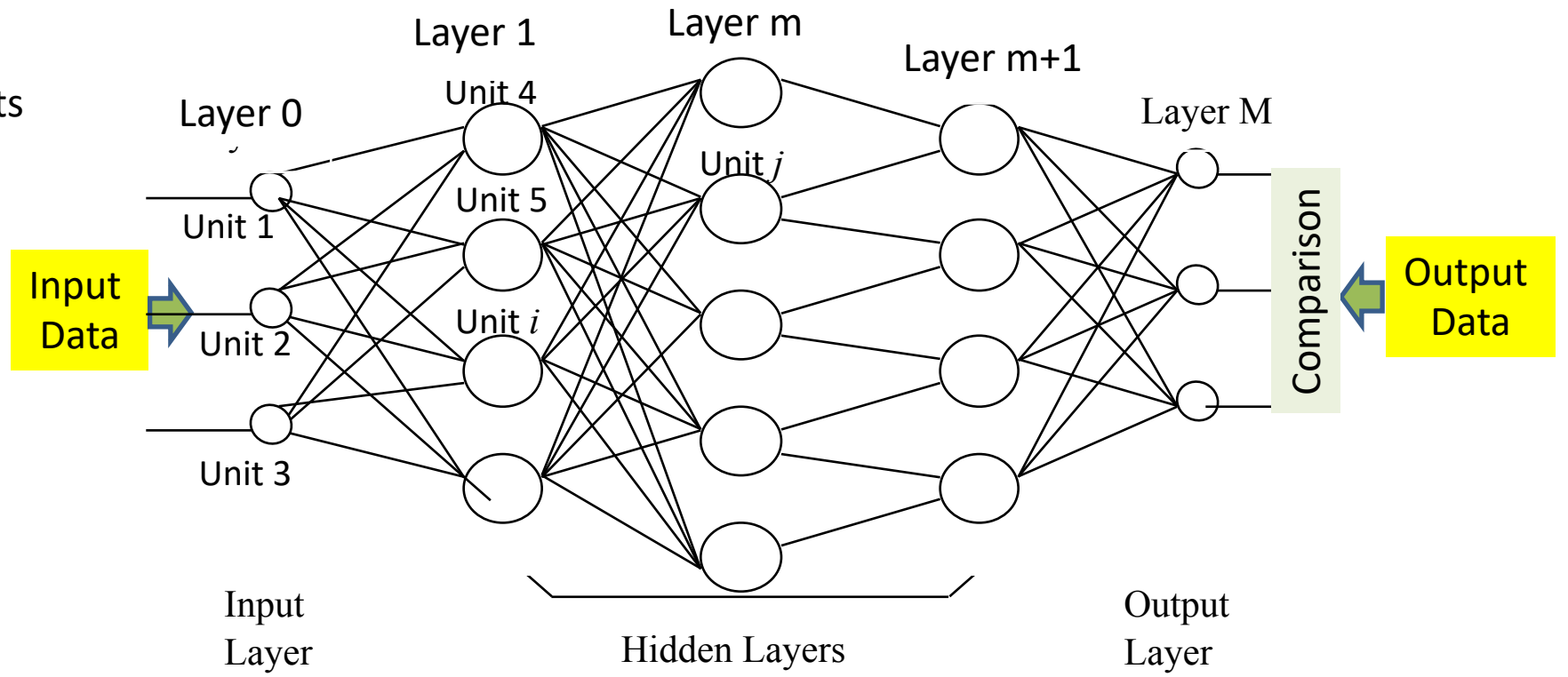
$$z = x_1 + x_2 - 2x_3 - \frac{1}{3}$$

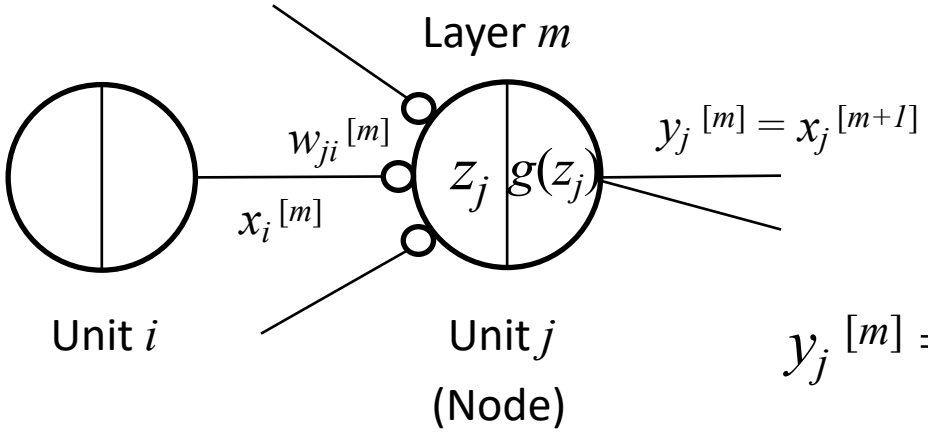


Hidden Unit
Not directly visible from output

Multi-Layer Neural Network

- The above example of XOR manifested the need for hidden units for solving a classification problem that is not linearly separable.
- The hidden unit generates an internal representation of the input pattern, providing the output unit with the critical information key to the correct classification.
- Generalizing this hidden unit's role, the architecture of Multi-Layer Neural Net was developed.





$w_{ji}^{[m]}$ Weight of the connection from unit i to unit j in layer m.

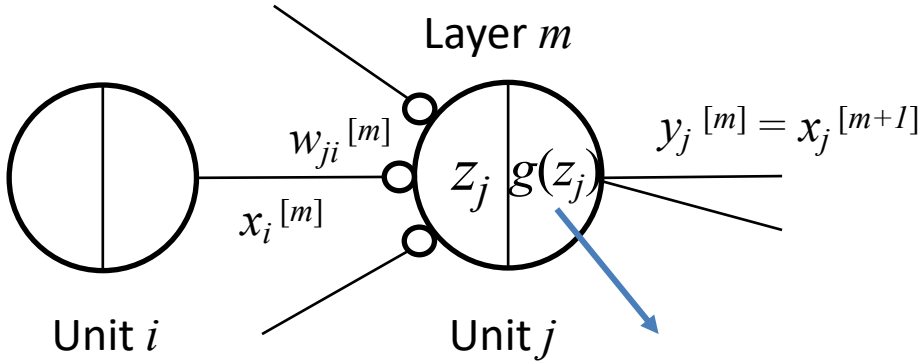
$x_i^{[m]}$ Input to a unit in layer m from unit i

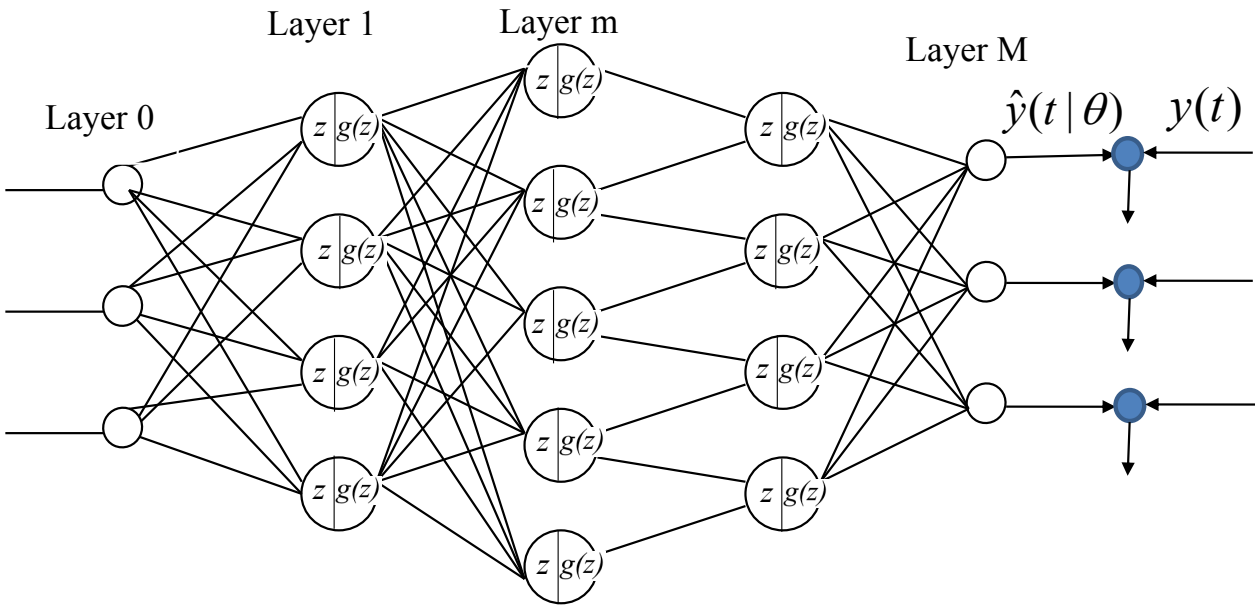
$y_j^{[m]} = x_j^{[m+1]}$ Output from unit j in layer m, which is the same as the input to a unit in layer (m+1)

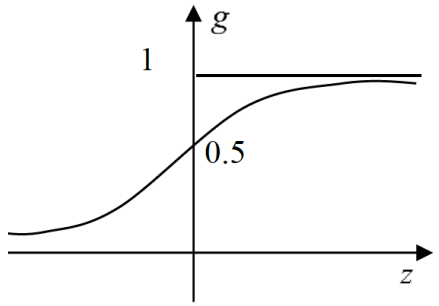
Multi-Layer Neural Network: Forward Path Computation

$$z_j^{(m)} = \sum_j w_{ji}^{(m)} x_i^{(m)}$$

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





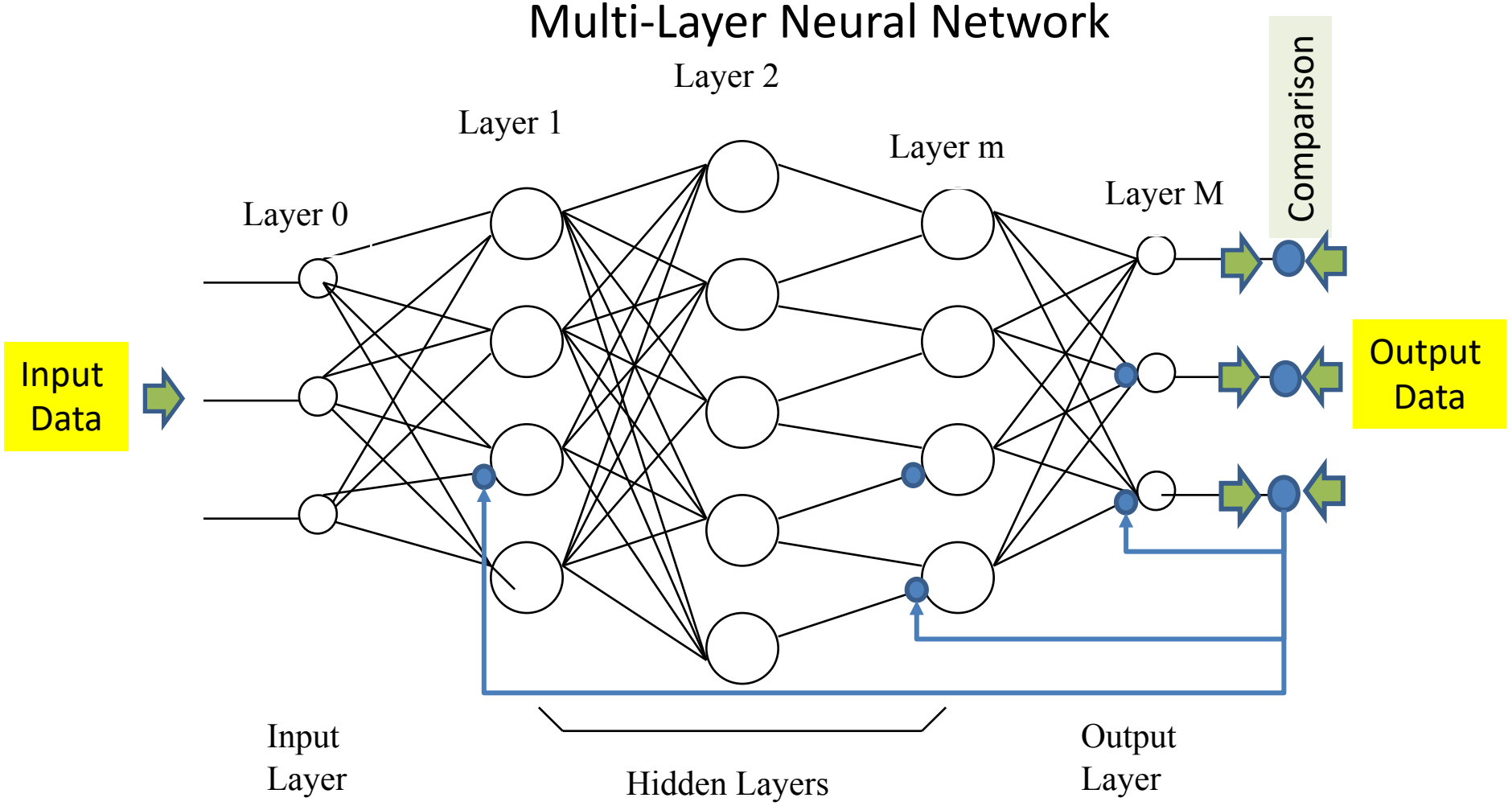


Activation Function

(Output Function)

- A 3-layer neural network with the sigmoid output function satisfies the Function Approximation Theorem, George Cybenko in 1989, Universal Approximator.
- It has been extended to deep neural nets with other output functions.

Multi-Layer Neural Network



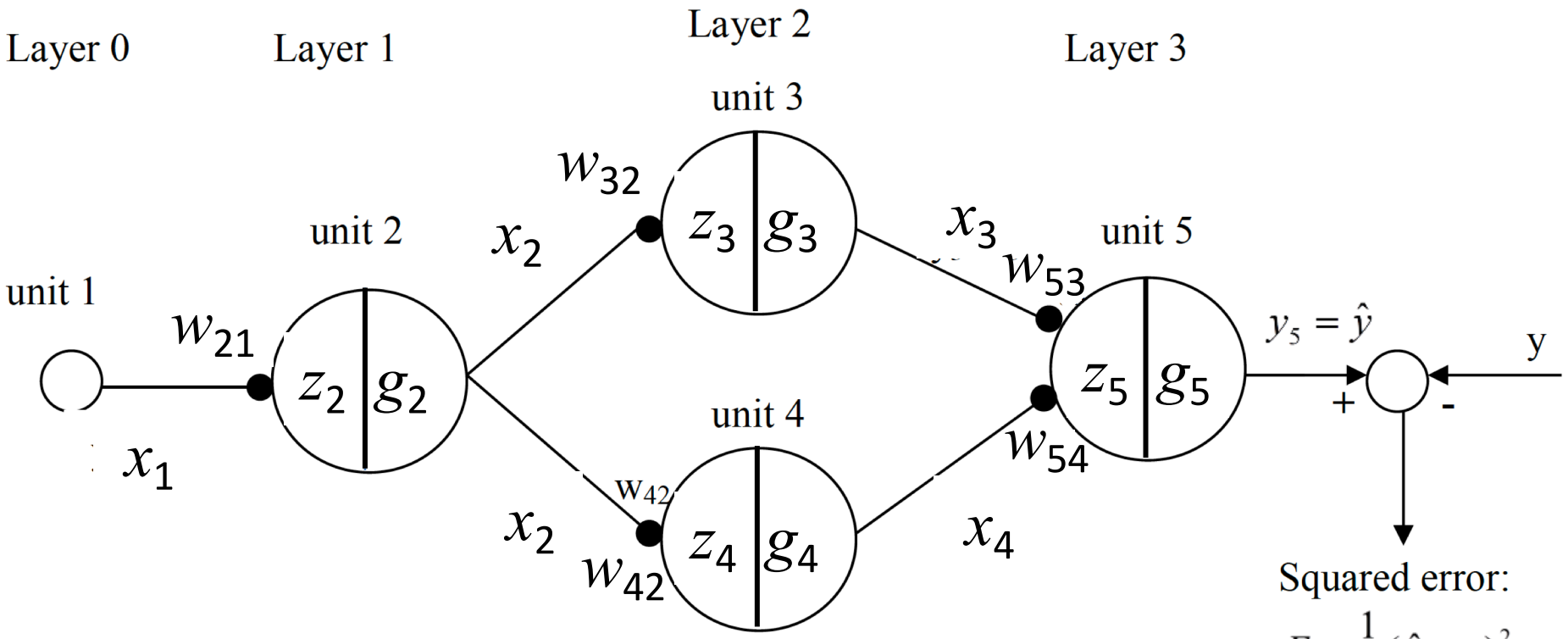
How do we train the multi-layer perceptron, given training data presented sequentially?

The Error Back Propagation Algorithm

The Error Back Propagation Algorithm

Before formulating a general algorithm, let's work out a simple example.

Example



Forward Path Computation

$$z_2 = w_{21}x_1 \qquad x_2 = g_2(z_2)$$

$$z_3 = w_{32}x_2 \qquad x_3 = g_3(z_3)$$

$$z_4 = w_{42}x_2 \qquad x_4 = g_4(z_4)$$

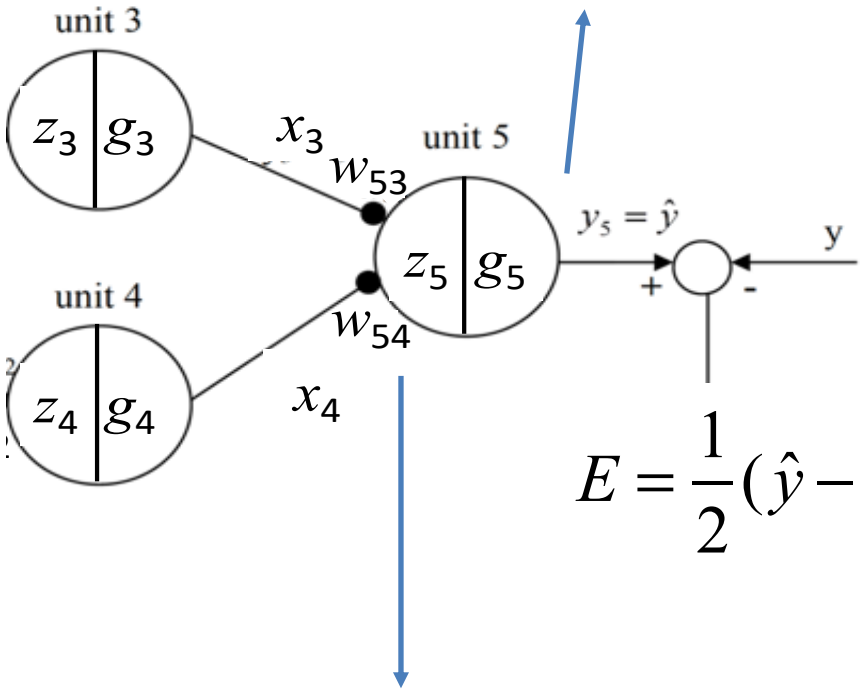
$$z_5 = w_{53}x_3 + w_{54}x_4 \qquad y_5 = g_5(z_5) = \hat{y}$$

The Error Back Propagation Algorithm

Example

Gradient Descent: $\Delta w_{ji} = -\rho \frac{\partial E}{\partial w_{ji}}$

$$y_5 = g_5(z_5) = \hat{y}$$



$$E = \frac{1}{2}(\hat{y} - y)^2$$

$$z_5 = w_{53}x_3 + w_{54}x_4$$

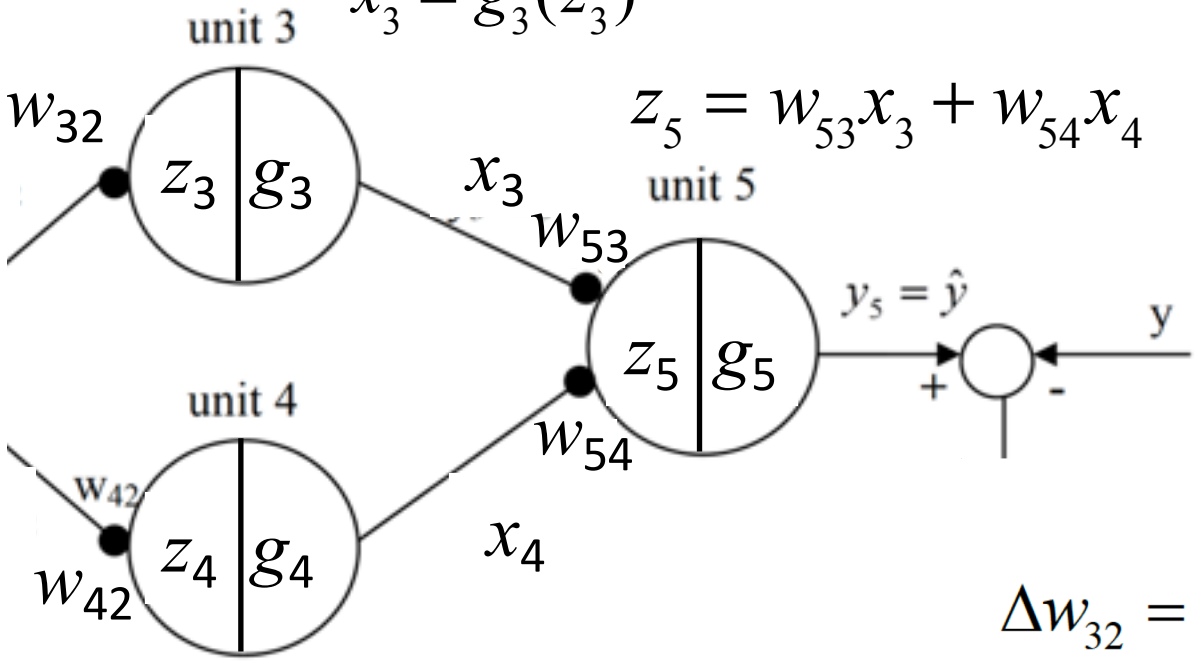
$$\Delta w_{53} = -\rho \cdot grad_{w_{53}} E$$

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

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

The Error Back Propagation Algorithm

$$z_3 = w_{32}x_2 \qquad x_3 = g_3(z_3)$$



$$\Delta w_{32} = -\rho \cdot grad_{w_{32}} E = -\rho \frac{\partial E}{\partial y_5}\frac{dy_5}{dz_5}\frac{\partial z_5}{\partial x_3}\frac{\partial x_3}{\partial z_3}\frac{\partial z_3}{\partial w_{32}}$$

(w_{53} and $\frac{dg_3}{dz_3}$ are grouped)

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

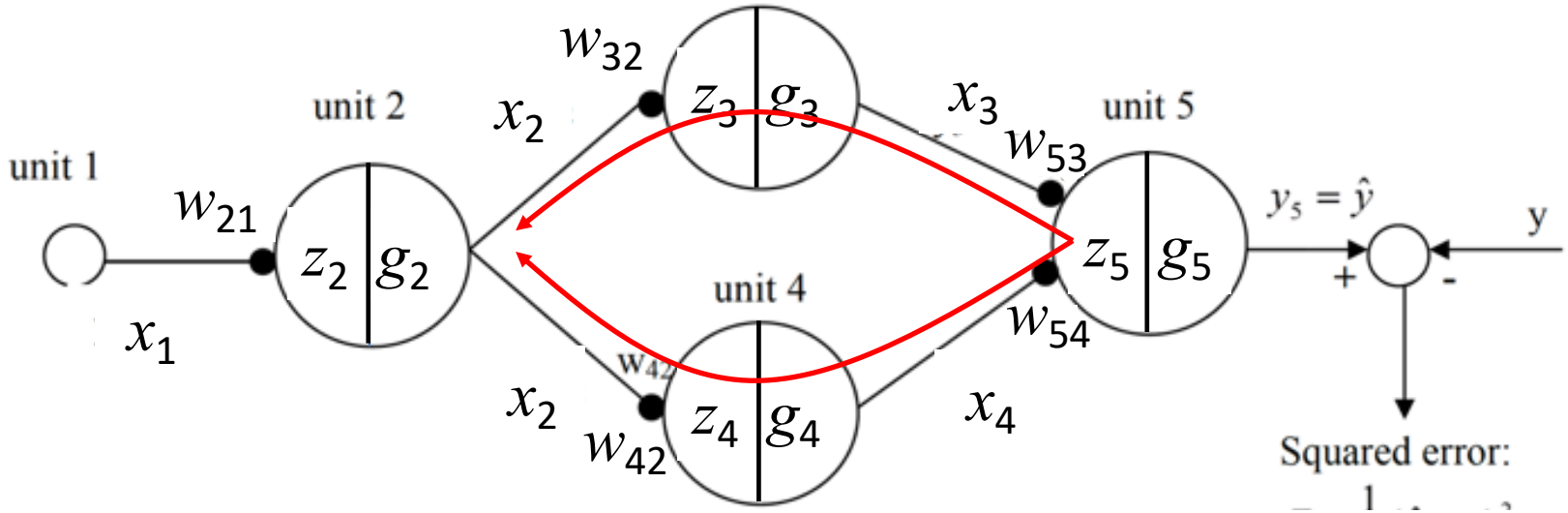
$$= \rho \underbrace{\delta_5 w_{53} g'_3(z_3)}_{\delta_3} x_2$$

Similarly,

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

Recall

$$-\delta_5 = \frac{\partial E}{\partial z_5} = \frac{\partial E}{\partial y_5}\frac{dy_5}{dz_5}$$



$$\Delta w_{21} = -\rho \cdot grad_{w_{21}} E$$

$$z_5 = w_{53}x_3 + w_{54}x_4$$

$$= -\rho \frac{\partial E}{\partial z_5}\left(\frac{\partial z_5}{\partial x_3}\frac{\partial x_3}{\partial x_2}\frac{\partial x_2}{\partial w_{21}} + \frac{\partial z_5}{\partial x_4}\frac{\partial x_4}{\partial x_2}\frac{\partial x_2}{\partial w_{21}}\right)$$

$$z_3 = w_{32}x_2$$

$$z_4 = w_{42}x_2$$

$$= \rho(\underbrace{\delta_5 w_{53} g'_3(z_3)}_{\delta_3} w_{32} + \underbrace{\delta_5 w_{54} g'_4(z_4)}_{\delta_4} w_{42}) \frac{\partial x_2}{\partial w_{21}}$$

$$z_2 = w_{21}x_1$$

$$= \rho(\delta_3 w_{32} + \delta_4 w_{42}) g'_2(z_2) x_1$$

Recursive Computation

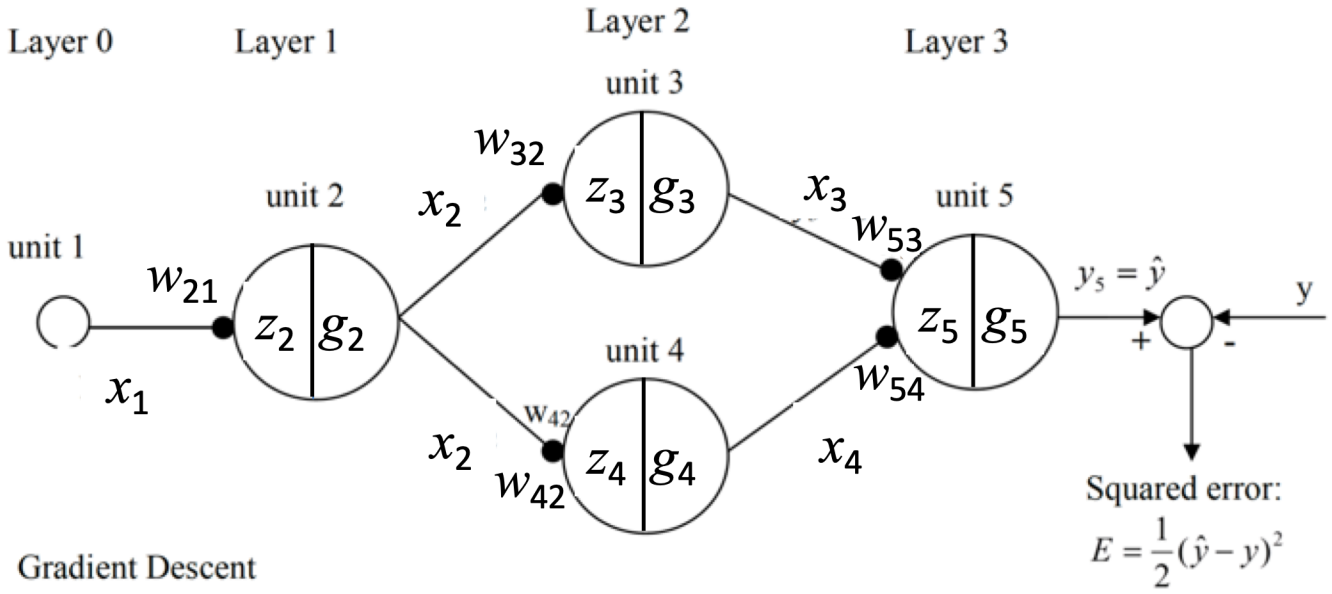
Computation of delta's from the final layer to the first layer

Error Backpropagation

$$\delta_5 = (y - \hat{y}) g'_5(z_5)$$
$$\delta_4 = \delta_5 w_{54} g'_4(z_4)$$
$$\delta_3 = \delta_5 w_{53} g'_3(z_3)$$
$$\delta_2 = (\delta_3 w_{32} + \delta_4 w_{42}) g'_2(z_2)$$

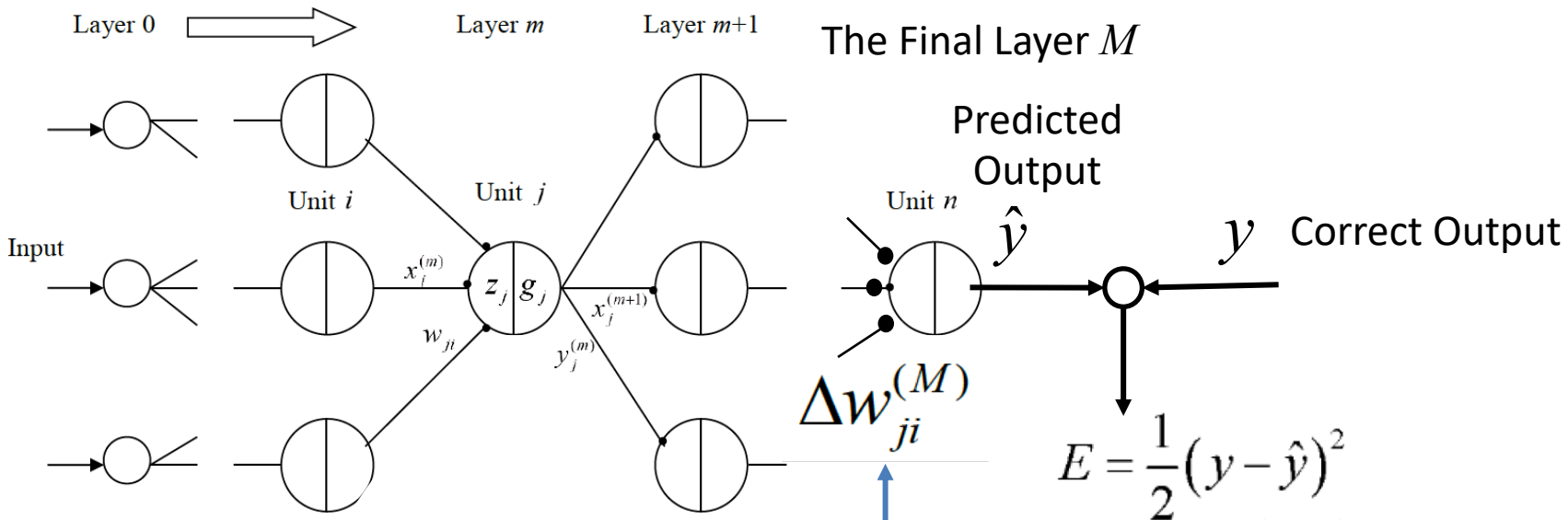
$$\Delta w_{53} = \rho \delta_5 x_3$$
$$\Delta w_{54} = \rho \delta_5 x_4$$
$$\Delta w_{32} = \rho \delta_3 x_2$$
$$\Delta w_{42} = \rho \delta_4 x_2$$

Changes to weights



Gradient Descent

The Error Back Propagation Algorithm



Gradient Descent $\Delta w = -\rho \frac{\partial E}{\partial w}$

Learning Rate

Final Layer M

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

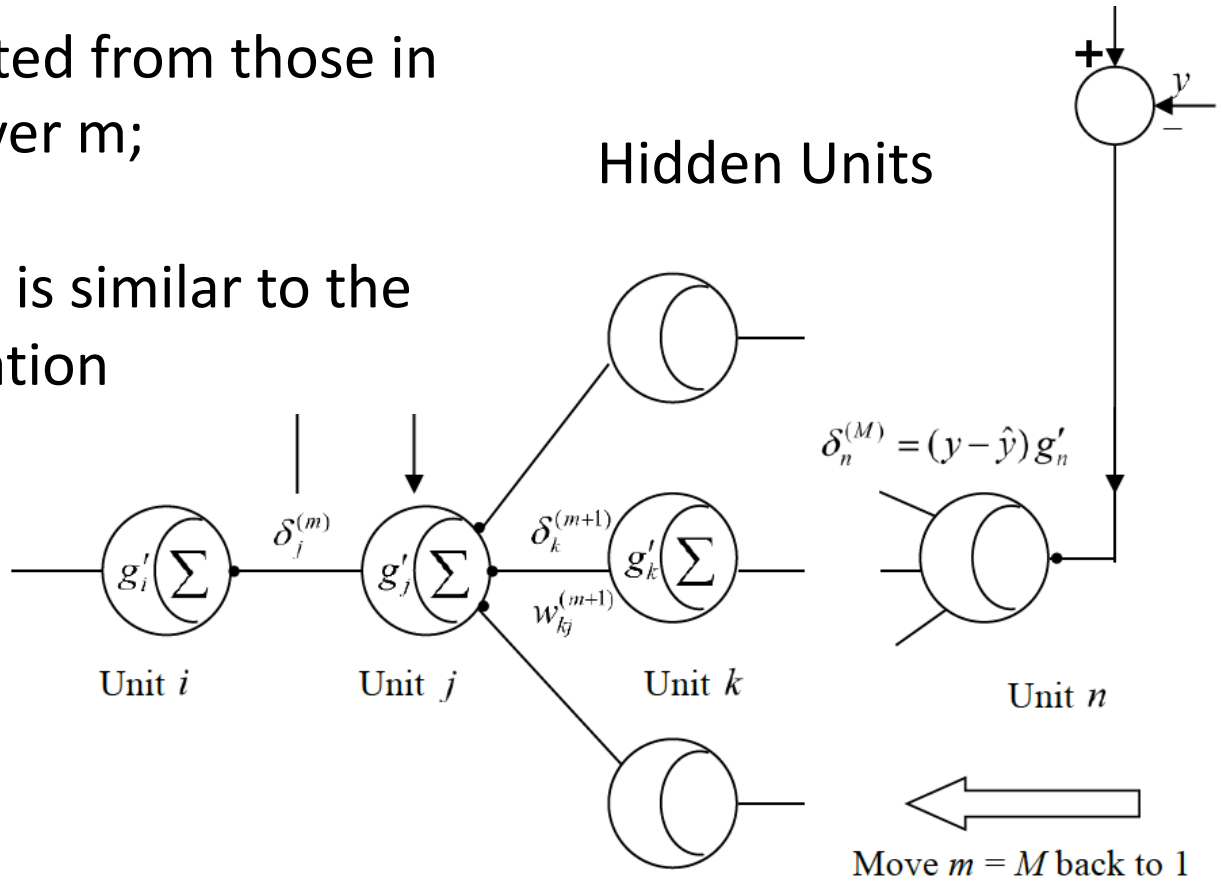
Delta can be computed directly from the correct output and the predicted output.

$$\delta_n^{(M)} = \left(y - \hat{y}^{(M)}\right) g_n'(z_n^{(M)})$$

The Error Back Propagation Algorithm

δ is backpropagated from those in layer (m+1) to layer m;

The computation is similar to the forward computation



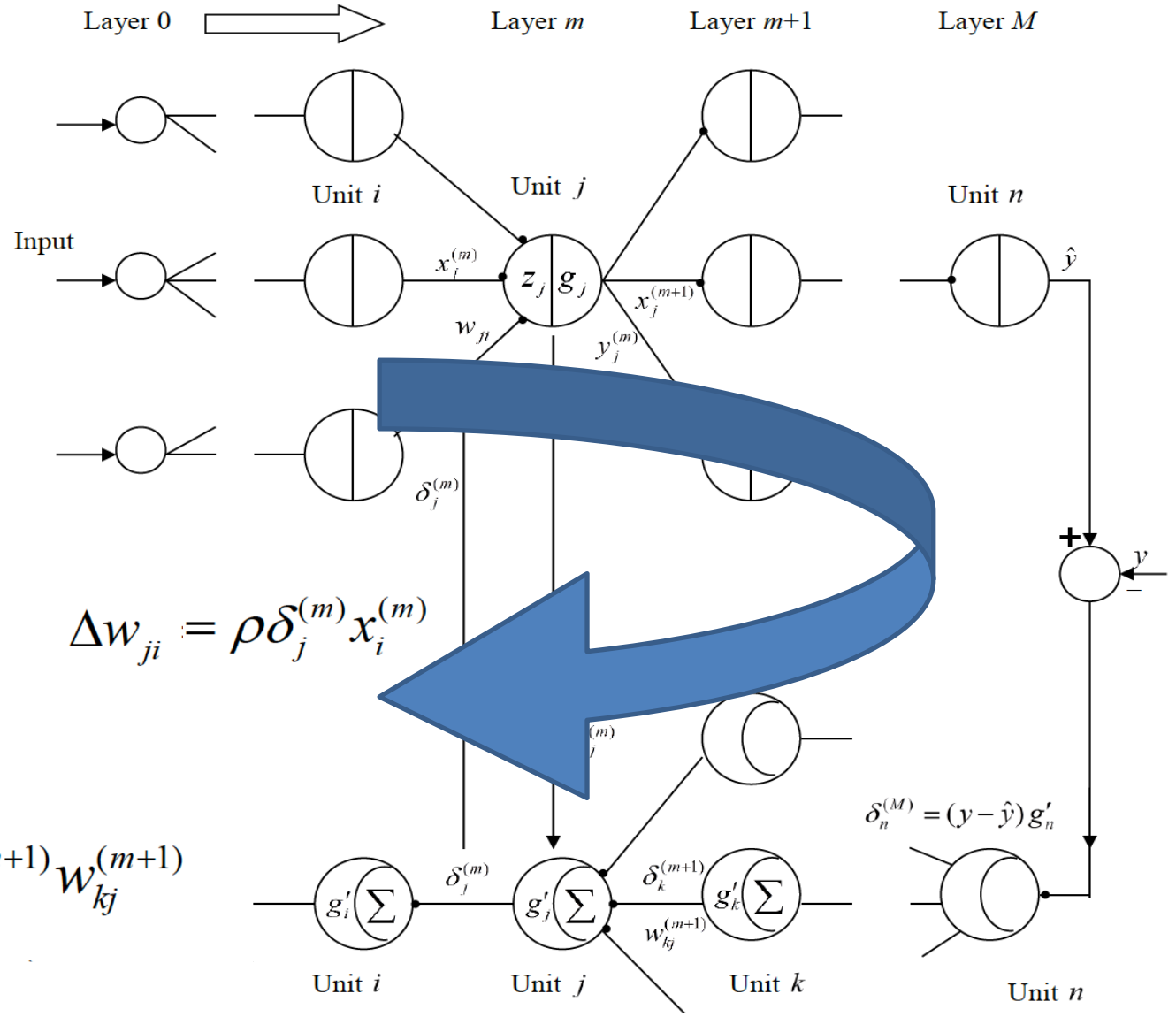
$$\delta_j^{(m)} = g_j'(z_j^{(m)}) \underbrace{\sum_k \delta_k^{(m+1)} w_{kj}^{(m+1)}}_{\text{Weighted sum}}$$

The Error Back Propagation Algorithm

Forward Computation

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

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



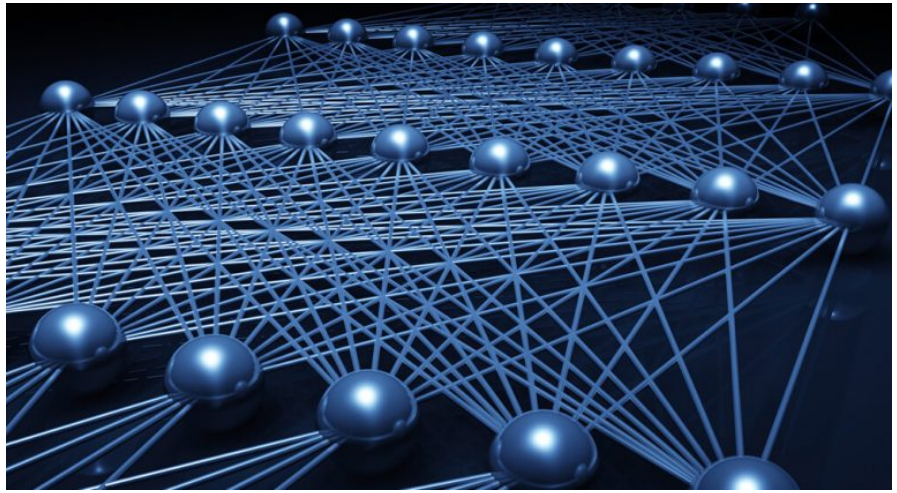
Backward Computation

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

The Error Backpropagation Algorithm
[Wobas 1974, 1994] [Rumelhart, Hinton, & Williams,1986]

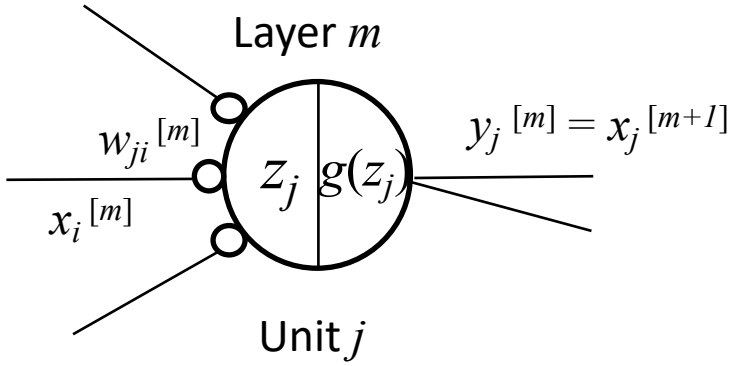
Training of Multi-Layer Neural Nets with Error Backpropagation

1. Sigmoid output function
2. Smoothing of convergence process
3. Local minima
4. Mini-batch
5. Hyperparameters

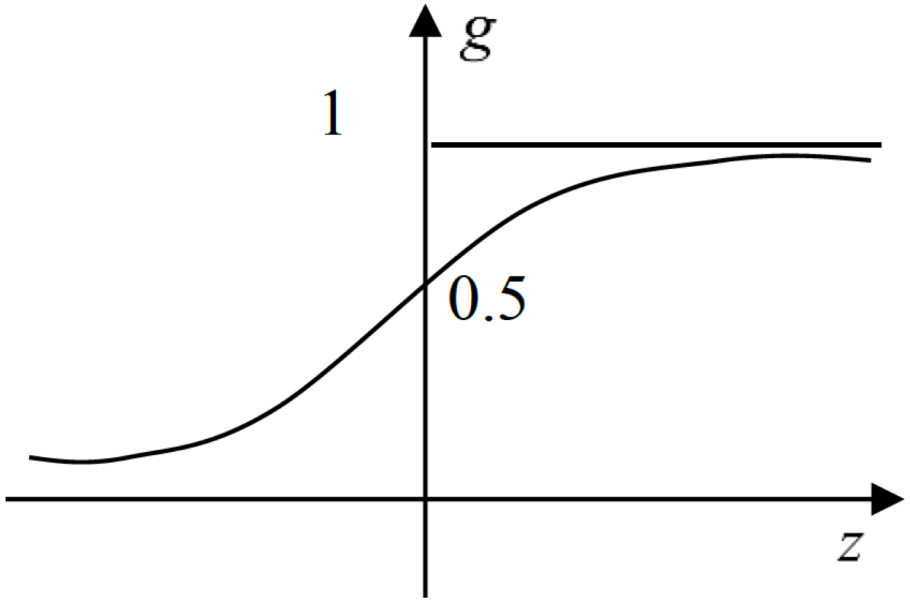


1. Sigmoid output function

Output Function: $g(z)$



Sigmoid Function



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

Nonlinear, differentiable

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

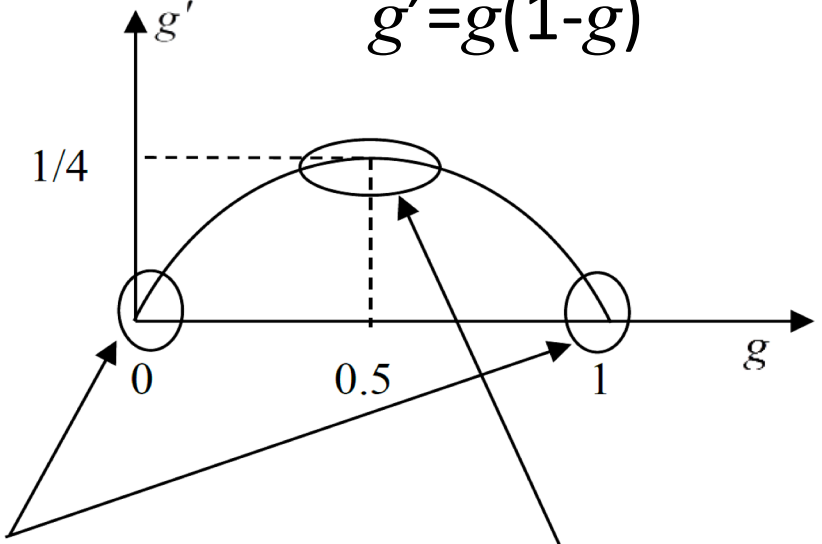
$$\Delta w_{ji} = \rho \delta_j^{(m)} x_i^{(m)}$$

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

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

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

$g'=g(1-g)$



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.

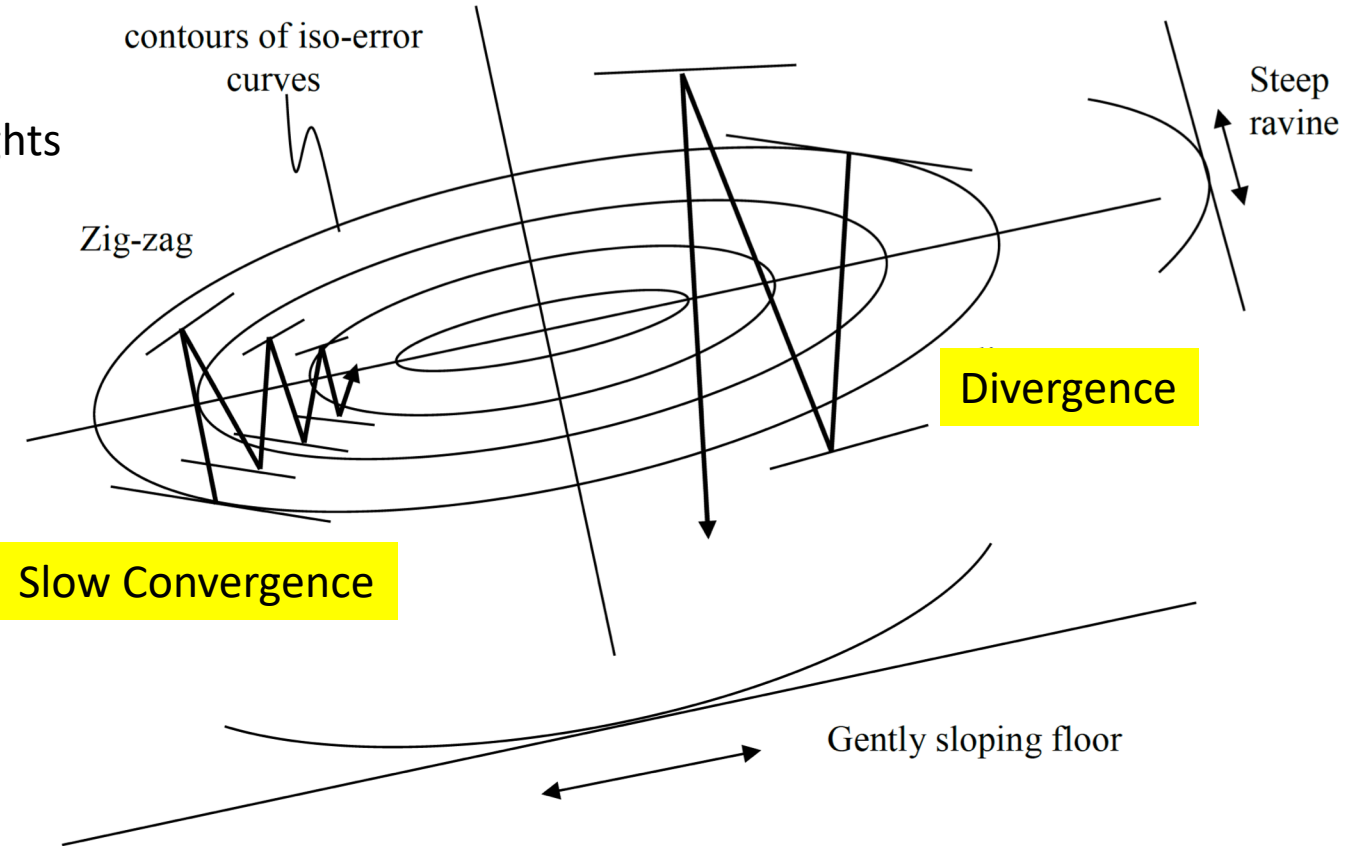
Once the unit (j) has committed to take an output value of either "0" or "1", the weight w_{ji} will no longer change very much for new inputs.

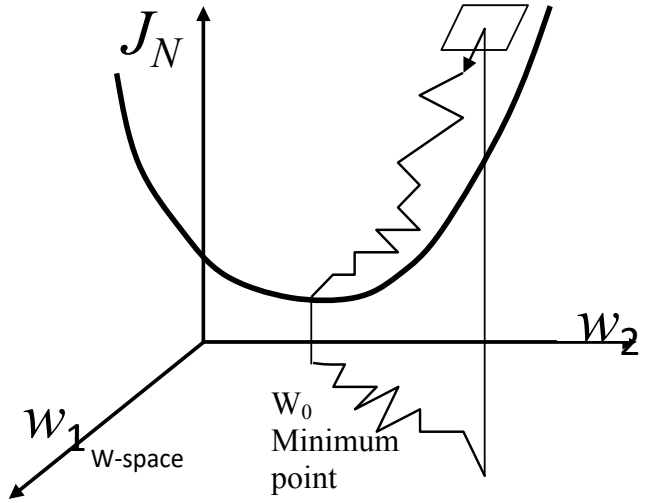
These properties contribute to stabilizing the learning process.

2. Smoothing of convergence process

A typical failure scenario of Neural Net training is “zig-zag” weight changes.
This results in a very slow convergence or even a divergence.
Suppose that the squared error function has a deep ravine.
The gradient direction bounces back and forth between the two steep walls, as shown below.

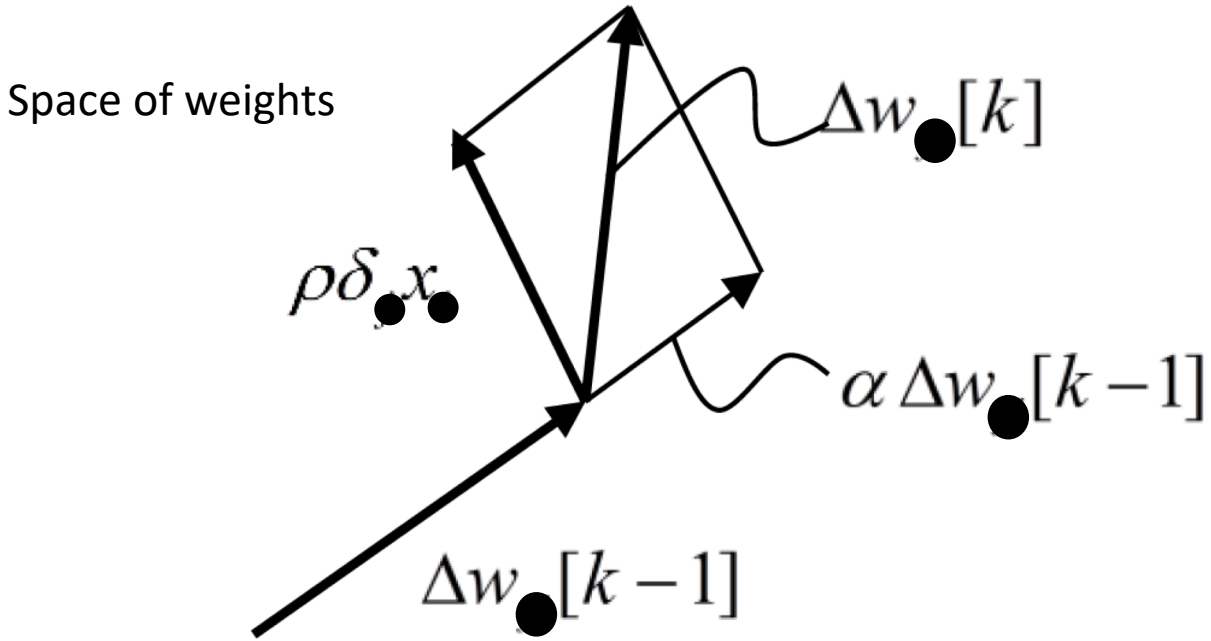
Space of weights





Adding a momentum term

Remedy: The zig-zag trajectory can be smoothed out by adding a momentum term to the weight change formula.

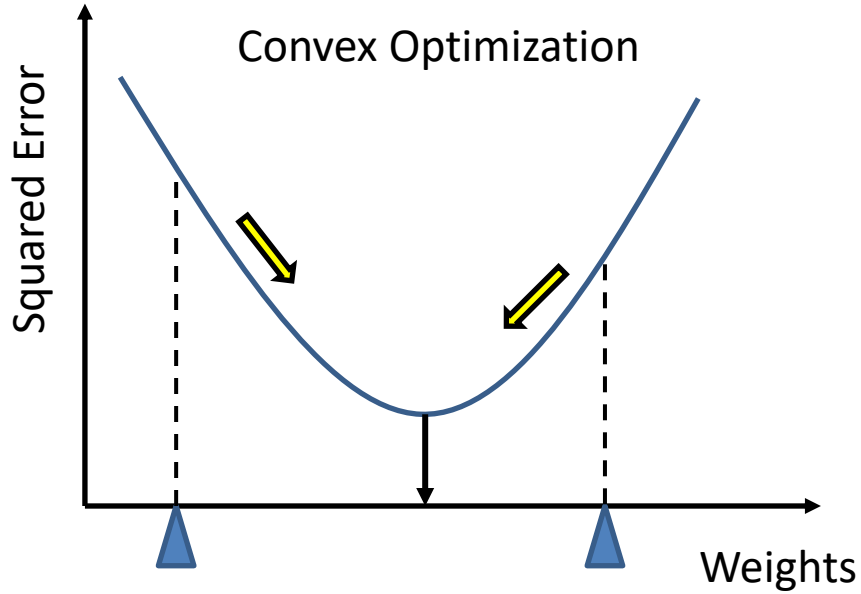


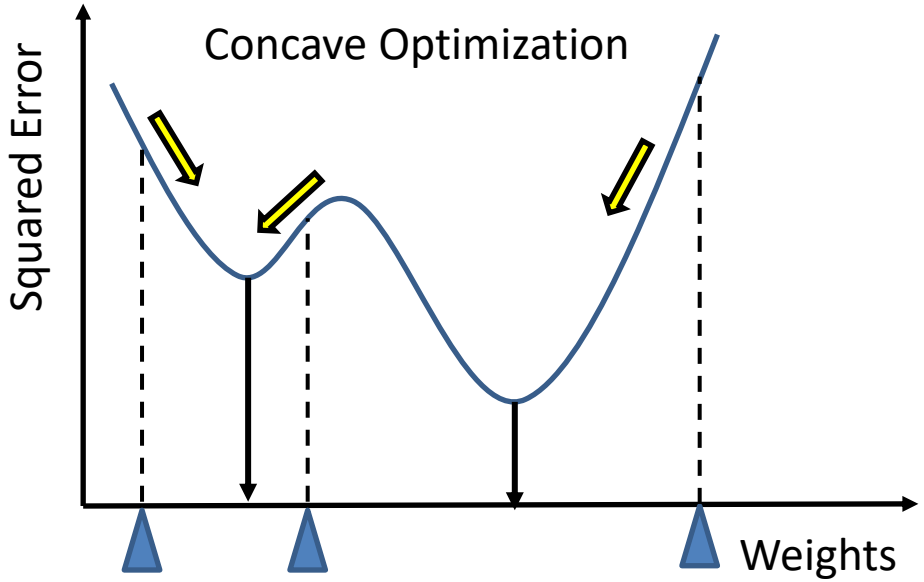
$$\Delta w_{ji}[k] = \rho\,\delta_j x_i + \alpha\,\Delta w_{ji}[k-1]$$

proportionality constant

Previous weight change

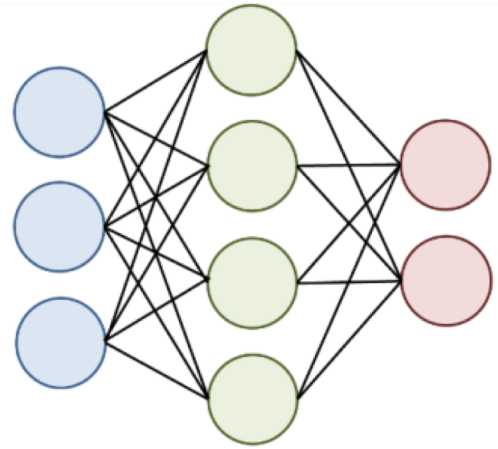
3. Local Minima





Training of a multi-layer neural net is typically a concave optimization problem.
There are multiple local minima in the weight space.

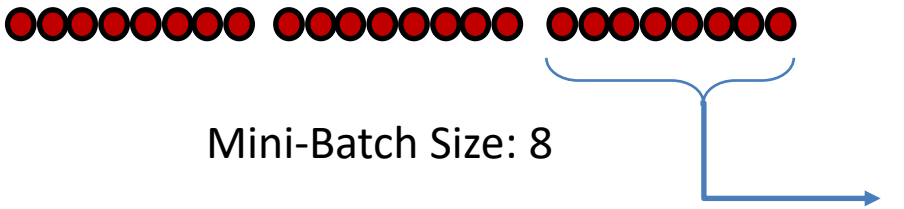
Remedy: Train a neural net multiple times starting with diverse initial conditions[1], compare the total squared errors, and pick the one that is the smallest in squared error.

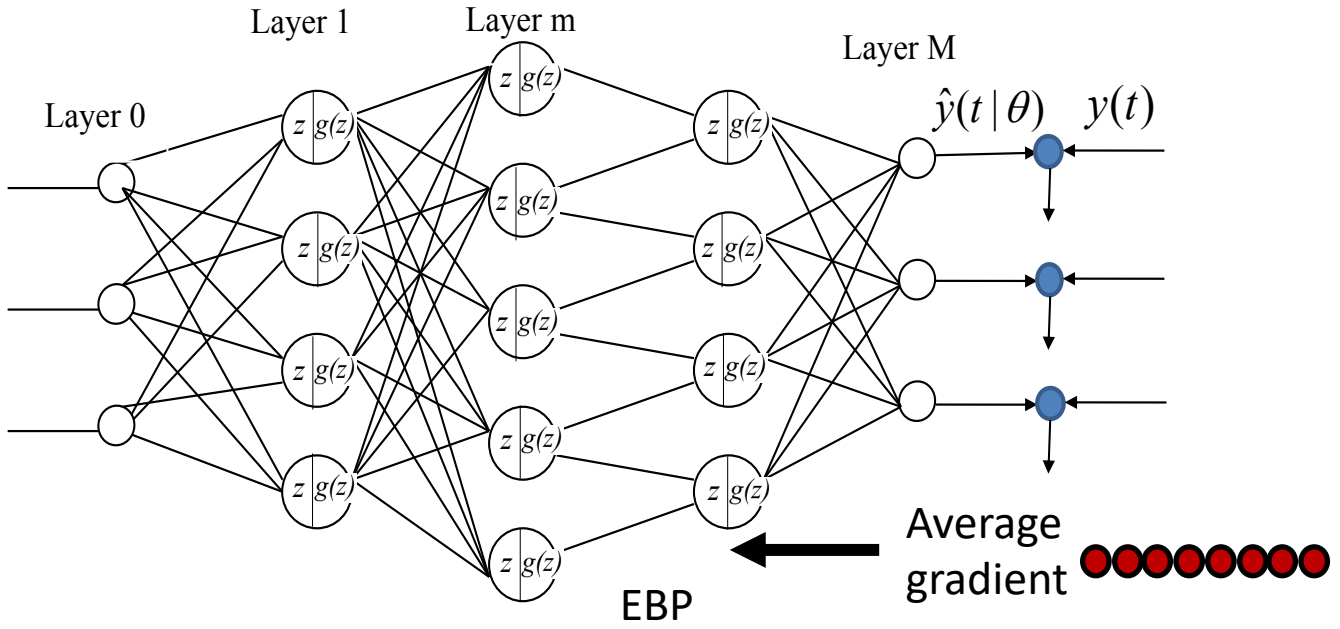


[1] Randomize initial values of the weights, and conduct the training repeatedly starting at different initial values of the weights. Each may end up with a different local minimum.

4. Mini-Batch Training

- The classical Widrow-Hoff stochastic gradient descent algorithm makes corrections to node weights for each single data point presented to the neural network.
- This has pros and cons:
  - Pros: not much memory space is required; good for getting rid of local minima
  - Cons: induces more noise in error calculations
- An alternative is **Mini-Batch** training, where a small set of data points are presented and the gradient is computed as the average of the gradients obtained from the small set of data points.



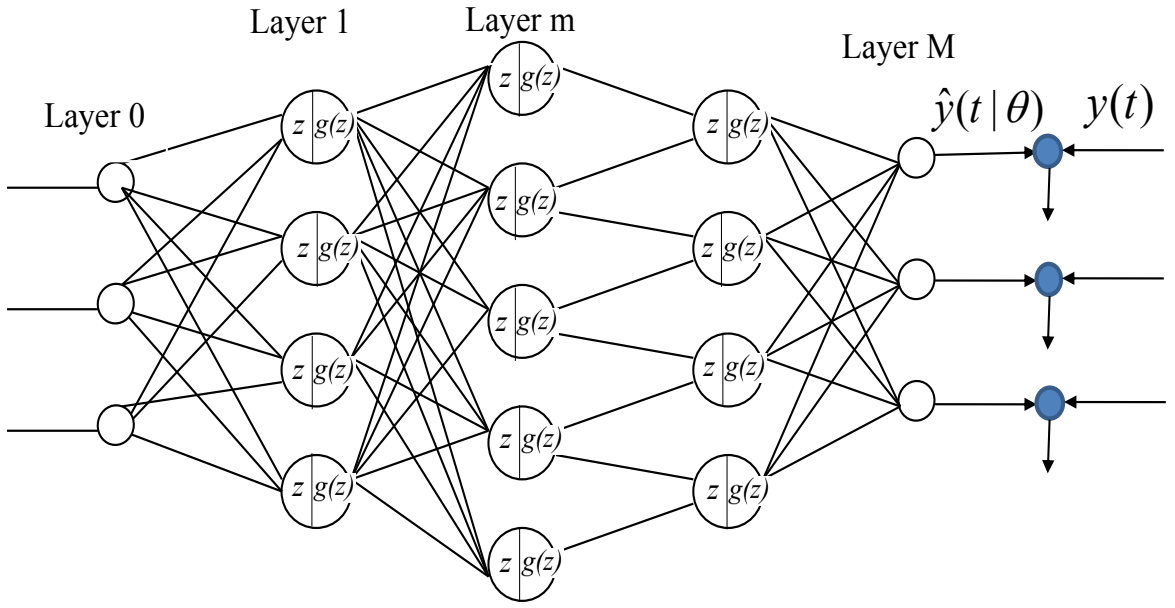


In practice, mini-batch size of 32 is commonly used.

5. Hyperparameters

- Neural net training performance depends on the structure and parameters that must be specified prior to training.
- These parameters differ from node weights, w_{ji}'s, and are called Hyperparameters.

- Learning rate ρ.
- The number of hidden layers
- The number of units in each layer
- Mini-batch size
- Epoch size
- Output function



Summary

- Artificial Neural Network
  - Basic neural network model
  - Widrow-Hoff stochastic gradient descent method
  - Nonlinear classification : XOR problem
  - Multi-layer neural net
  - The Error Back Propagation Algorithm
  - Sigmoid output function and stability
  - Momentum term for smoothing
  - Local minima
  - Mini-batch training
  - Hyperparameters