

LECTURE NOTES

cse352

Professor Anita Wasilewska

**NEURAL NETWORKS**

Backpropagation Algorithm

# Neural Networks Classification

## Introduction

- **INPUT**: classification data, i.e. data that contains a classification (class) attribute.
- **WE** also say that the class label is known for all data.
- **DATA** is divided, as in any classification problem, into **TRAINING** and **TEST** data sets.

# Neural Networks Classifier

- **ALL DATA must be normalized**, i.e. all values of attributes in the dataset has to be changed to contain values in the interval  $[0,1]$ , or  $[-1,1]$ .

**TWO BASIC** normalization techniques:

- **Max- Min** normalization and
- **Decimal Scaling** normalization.

# Data Normalization

- **Max-Min Normalization**

Performs a linear transformation on the original data.

- Given an attribute  $A$ , we denote by  $minA$ ,  $maxA$  the minimum and maximum values of the values of the attribute  $A$ .

- **Max-Min Normalization** maps a value  $v$  of  $A$  to  $v'$  in the range

- $[new\_minA, new\_maxA]$  as follows.

# Data Normalization

Max- Min normalization formula is as follows:

$$v' = \frac{v - \min A}{\max A - \min A} (\text{new\_max } A - \text{new\_min } A) + \text{new\_min } A$$

**Example:** we want to normalize data to range of the interval **[-1,1]**.

We put: **new\_max A= 1, new\_minA = -1.**

In general, to normalize within interval **[a,b]**, we put:  
**new\_max A= b, new\_minA = a.**

# Example of Max-Min Normalization

## Max- Min normalization formula

$$v' = \frac{v - \min A}{\max A - \min A} (\text{new\_max } A - \text{new\_min } A) + \text{new\_min } A$$

**Example:** We want to normalize data to range of the interval [0,1].

We put: **new\_max A= 1, new\_minA =0.**

Say, max A was 100 and min A was 20 ( That means maximum and minimum values for the attribute A).

Now, if  $v = 40$  ( If for this particular pattern , attribute value is 40 ),  $v'$  will be calculated as ,  $v' = (40-20) \times (1-0) / (100-20) + 0$

$$\Rightarrow v' = 20 \times 1/80 = 1/4$$

$$\Rightarrow v' = 0.25$$

# Decimal Scaling Normalization

Normalization by decimal scaling normalizes by moving the decimal point of values of attribute A.

A value  $v$  of A is normalized to  $v'$  by computing

$$v' = \frac{v}{10^j}$$

where  $j$  is the smallest integer such that  $\max|v'| < 1$ .

## Example :

A – values range from -986 to 917.     $\text{Max } |v| = 986$ .

$v = -986$  normalize to  $v' = -986/1000 = -0.986$

# Neural Network

- **Neural Network** is a set of connected **INPUT/OUTPUT UNITS**, where each connection has a **WEIGHT** associated with it.
  - **Neural Network** learning is also called **CONNECTIONIST** learning due to the connections between units.
- **Neural Network** is always fully connected.
- It is a case of **SUPERVISED, INDUCTIVE** or **CLASSIFICATION** learning.

# Neural Network Learning

- **Neural Network** learns by adjusting the weights so as to be able to correctly classify the training data and hence, after testing phase, to classify **unknown data**.
- **Neural Network** needs long time for training.
- **Neural Network** has a high tolerance to noisy and incomplete data.

# Neural Network Learning

- Learning is being performed by a **back propagation algorithm**.
- The inputs are fed simultaneously into the **input layer**.
- The weighted outputs of these units are, in turn, are fed simultaneously into a “neuron like” units, known as a **hidden layer**.
- The hidden layer’s weighted outputs can be input to another hidden layer, and so on.
- The number of hidden layers is arbitrary, but in practice, usually one or two are used.
- The weighted outputs of the last hidden layer are inputs to units making up the **output layer**.

# A Multilayer Feed-Forward (MLFF) Neural Network

Output vector;

Output nodes

(as many as classes)

$O_k$

Hidden nodes

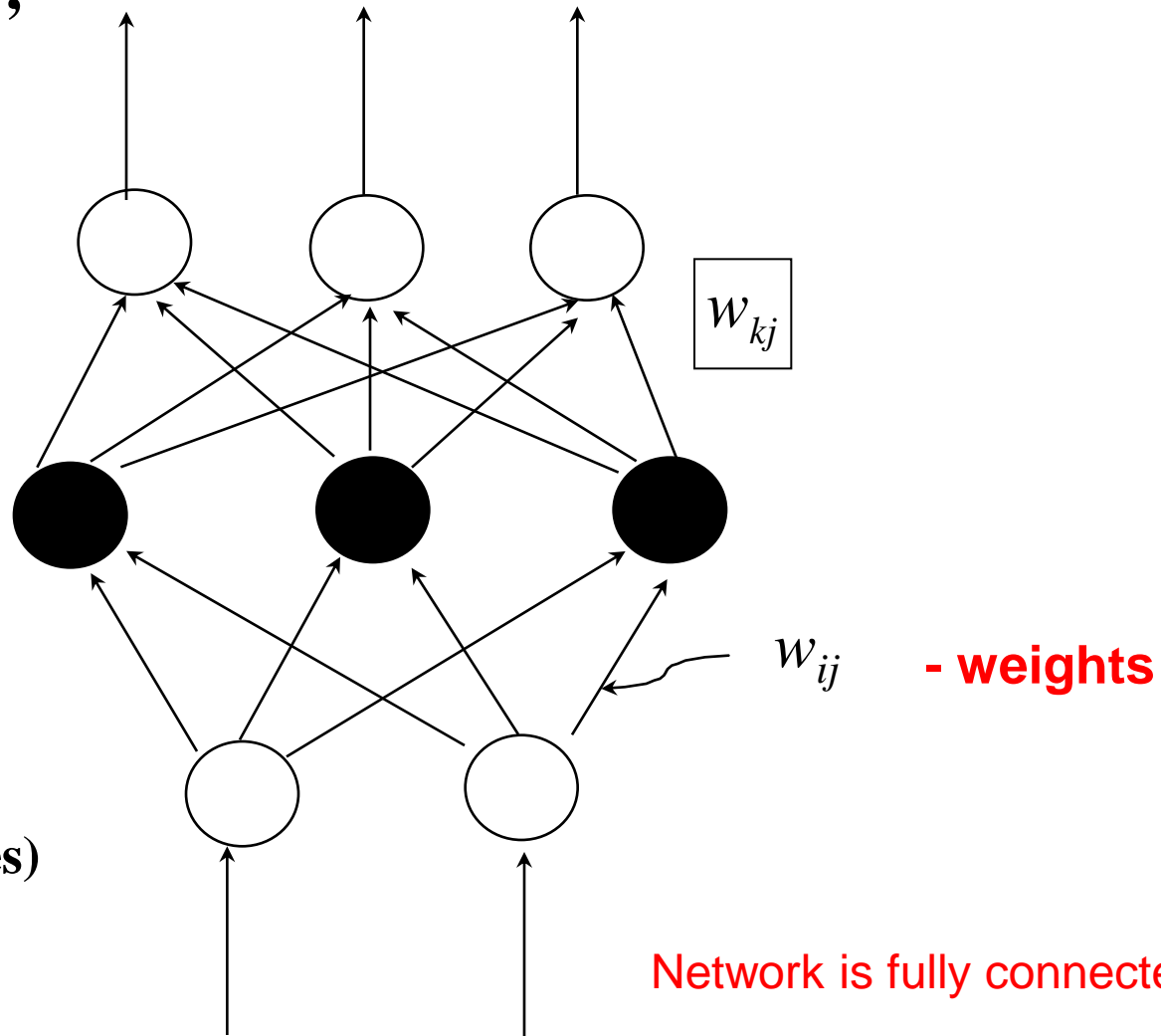
$O_j$

Input nodes

(as many as attributes)

Input vector;

Record:  $x_i$



Network is fully connected

# MLFF Neural Network

- **The units in the hidden layers and output layer** are sometimes referred to as **neurons**, due to their symbolic biological basis, or as **output units**.
- **A multilayer neural network** shown on the previous slide has **two layers** of output units.
- Therefore, we say that it is **a two-layer** neural network.

# MLFF Neural Network

- A network containing two hidden layers is called **a three-layer** neural network, and so on.
- **The network is feed-forward** in that none of the weights cycles back to an input unit or to an output unit of a previous layer.

# MLFF Neural Network

Output vector;  
classes

Output nodes

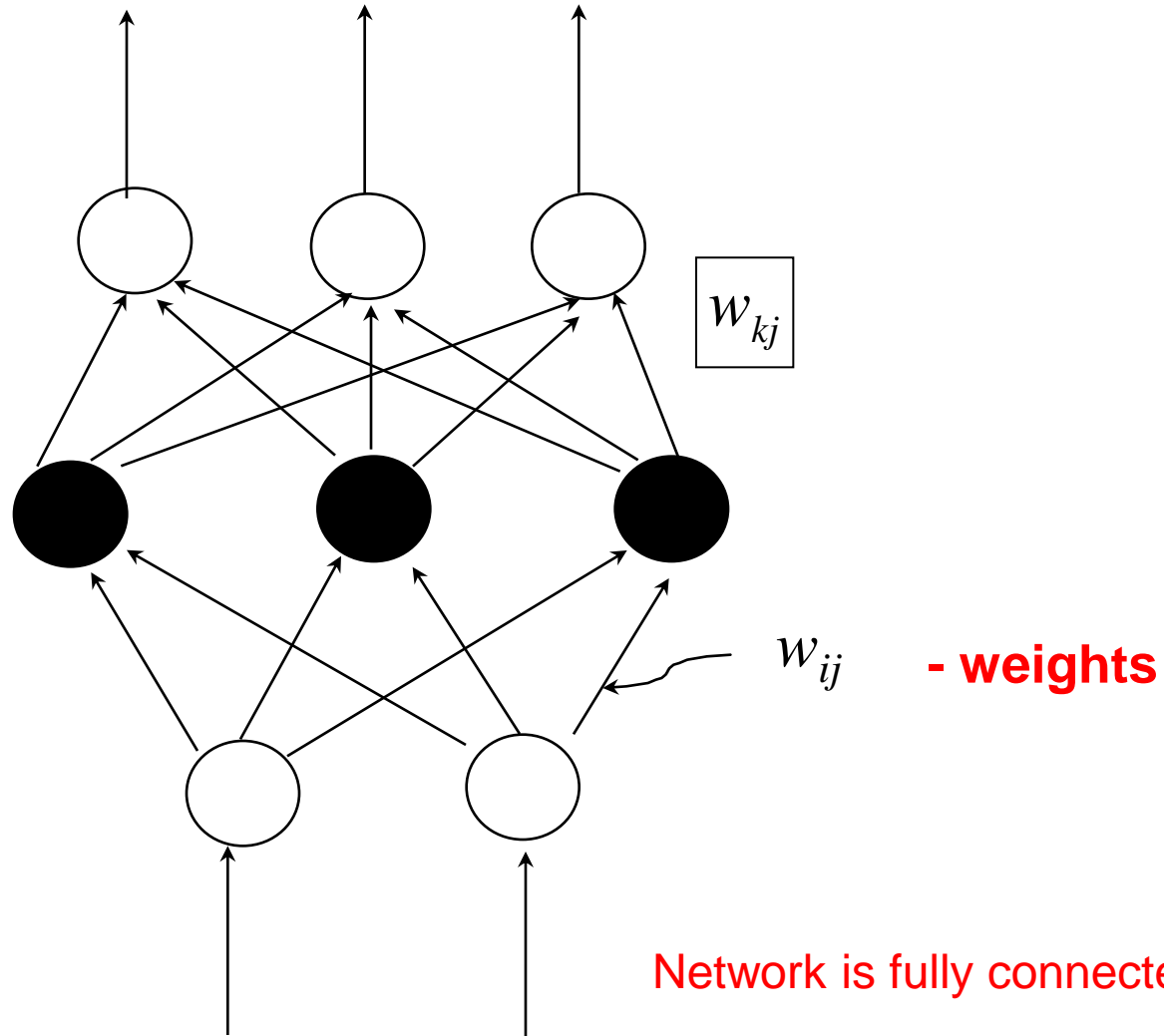
$O_k$

Hidden nodes

$O_j$

Input nodes

Input vector;  
Record:  $x_i$



Network is fully connected

# MLFF Network Input

- **INPUT:** records without class attribute with **normalized** attributes values.
- We call it an **input vector**.
- **INPUT VECTOR:**

$$X = \{ x_1, x_2, \dots, x_n \}$$

where **n** is the number of (non class) **attributes**.

# MLFF Network Topology

- **INPUT LAYER** – there are as many nodes as non-class attributes i.e. as the length of the input vector.
- **HIDDEN LAYER** – the number of nodes in the hidden layer and the number of hidden layers depends on implementation.

$$O_j$$

$j=1, 2 \dots \#$ hidden nodes

# MLFF Network Topology

- **OUTPUT LAYER** – corresponds to the **class attribute**.
- There are as many nodes as classes (values of the class attribute).

$$O_k$$

$k = 1, 2, \dots \text{\#classes}$

- Network is always **fully connected**, i.e. each unit provides input to each unit in the next forward layer.

# Classification by Backpropagation

- **Backpropagation** is a neural network learning algorithm.
- It learns by iteratively processing a set of **training data** (samples), comparing the network's classification of each record (sample) with the actual known class label (classification).

# Classification by Backpropagation

- For each training sample, the **weights are modified** as to minimize the mean squared error between the network's classification (prediction) and actual classification (value of the class attribute).
- These **weights** modifications are made in “**backwards**” direction, that is, from the **output layer**, through each hidden layer down to the **first hidden layer**.
- Hence the name **backpropagation**.

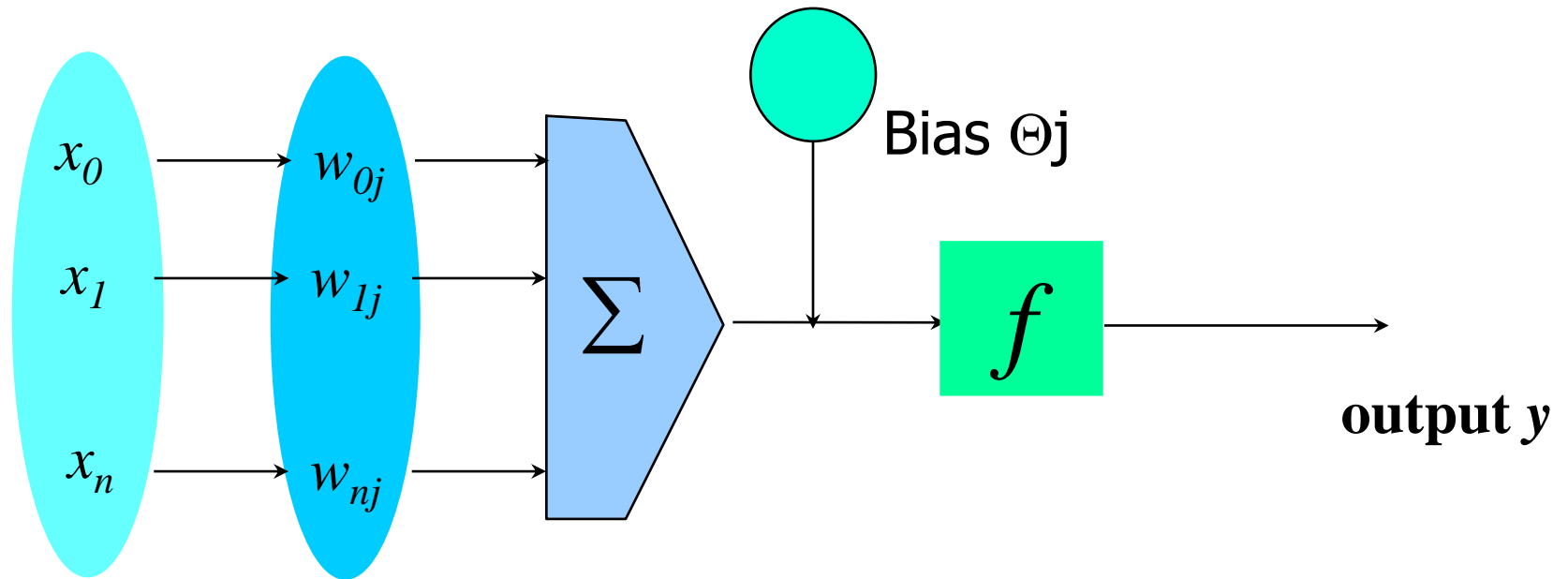
# Steps in Backpropagation Algorithm

- **STEP ONE:** initialize the weights and biases.
- The **weights** in the network are **initialized** to small random numbers ranging for example from **-1.0 to 1.0**, or **-0.5 to 0.5**.
- Each unit has a **BIAS** associated with it (see next slide).
- The **biases** are similarly **initialized** to small random numbers.
- **STEP TWO:** feed the training sample (record)

# Steps in Backpropagation Algorithm

- **STEP THREE:** propagate the inputs forward; we compute the net input and output of each unit in the hidden and output layers.
- **STEP FOUR:** backpropagate the error.
- **STEP FIVE:** update weights and biases to reflect the propagated errors.
- **STEP SIX:** repeat and apply terminating conditions.

# A Neuron; a Hidden, or Output Unit $j$



**Input**      **weight**      **weighted**      **Activation**  
**vector  $x$**    **vector  $w$**    **sum**      **function**

- The inputs to unit  $j$  are outputs from the previous layer.
- These are multiplied by their corresponding weights in order to form a **weighted sum**, which is added to **the bias** associated with unit  $j$ .
- A nonlinear **activation function  $f$**  is applied to the net input.

# Step Three: propagate the inputs forward

- For unit  $j$  in the input layer, its output is equal to its input, that is,

$$O_j = I_j$$

for input unit  $j$ .

- The net input to each unit in the hidden and output layers is computed as follows.
- Given a unit  $j$  in a hidden or output layer, the net input is

$$I_j = \sum_i w_{ij} O_i + \theta_j$$

where  $w_{ij}$  is the weight of the connection from unit  $i$  in the previous layer to unit  $j$ ;  $O_i$  is the output of unit  $i$  from the previous layer;

$$\theta_j$$

is the bias of the unit

## Step Three: propagate the inputs forward

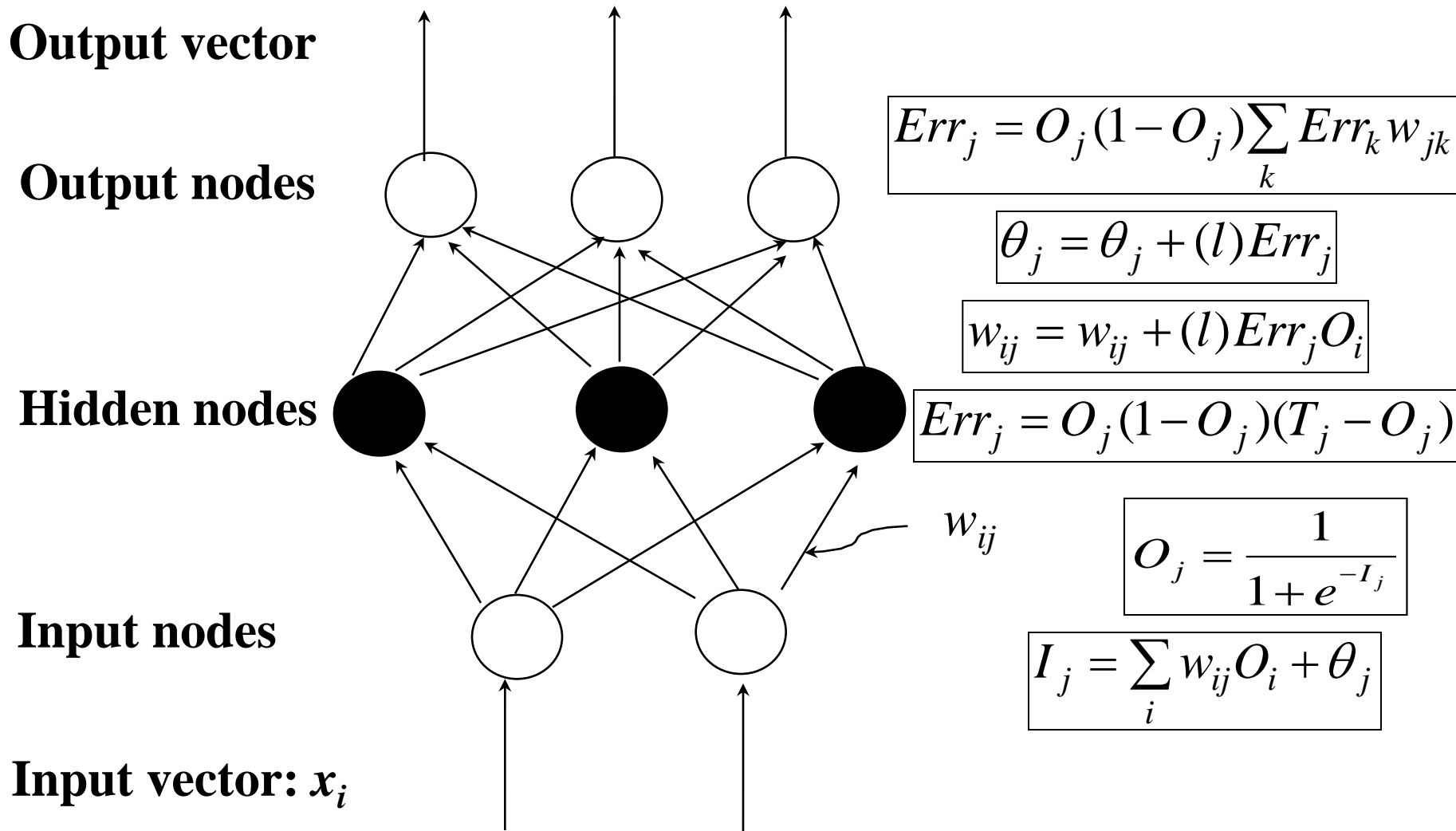
- Each unit in **the hidden** and **output layers** takes its net input and then applies **an activation function**. The function symbolizes the activation of the neuron represented by the unit. It is also called **a logistic, sigmoid, or squashing function**.
- Given a net input  $I_j$  to unit  $j$ , then

$$O_j = f(I_j),$$

the output of unit  $j$ , is computed as

$$O_j = \frac{1}{1 + e^{-I_j}}$$

# Back propagation Formulas



## Step 4: Back propagate the error

- When reaching the **output layer**, the **error** is computed and **propagated backwards**.
- For a unit  $k$  in the output layer the error is computed by a formula:

$$Err_k = O_k (1 - O_k) (T_k - O_k)$$

Where  $O_k$  is the **actual output** of unit  $k$  ( computed by activation function.

$$O_k = \frac{1}{1 + e^{-I_k}}$$

$T_k$  is the **TRUE output** based of known class label; classification of training sample

Observe:  $O_k(1-O_k)$  is a **Derivative ( rate of change )** of activation function.

# Step 4: Back propagate the error

- The error is propagated backwards by updating weights and biases to reflect the error of the network classification .
- For a unit  $j$  in the hidden layer the error is computed by a formula:

- $$Err_j = O_j(1 - O_j) \sum_k Err_k w_{jk}$$

where  $w_{jk}$  is the weight of the connection from unit  $j$  to unit  $k$  in the next higher layer, and  $Err_k$  is the error of unit  $k$ .

# Step 5: Update weights and biases

- **Weights** are updated by the following equations, where  $l$  is a constant between 0.0 and 1.0 reflecting **the learning rate**, this learning rate is **fixed for implementation**.

$$\Delta w_{ij} = (l) Err_j O_i$$

$$w_{ij} = w_{ij} + \Delta w_{ij}$$

- **Biases** are updated by the following equations

$$\Delta \theta_j = (l) Err_j$$

$$\theta_j = \theta_j + \Delta \theta_j$$

# Weights and Biases Updates

- **Case updating:** we are updating weights and biases after the presentation of each sample (record).
- **Epoch:** One iteration through the training set is called **an epoch**.
- **Epoch updating:**
  - The weight and bias increments are accumulated in variables and the weights and biases are updated after all of the samples of the training set have been presented.
- **Case updating is more accurate**

# Terminating Conditions

- Training **stops** when
  - All  $\Delta w_{ij}$  in the previous epoch are below some threshold, **or**
  - The percentage of samples misclassified in the previous epoch is below some threshold, **or**
  - a pre- specified number of epochs has expired.
  - In practice, **several hundreds of thousands of epochs** may be required before the weights will converge.

# Back Propagation Formulas

**Output vector**

$$Err_k = O_k (1 - O_k) (T_k - O_k)$$

**Output nodes**

$$O_j = \frac{1}{1 + e^{-I_j}}$$

$$Err_j = O_j (1 - O_j) \sum_k Err_k w_{jk}$$

**Hidden nodes**

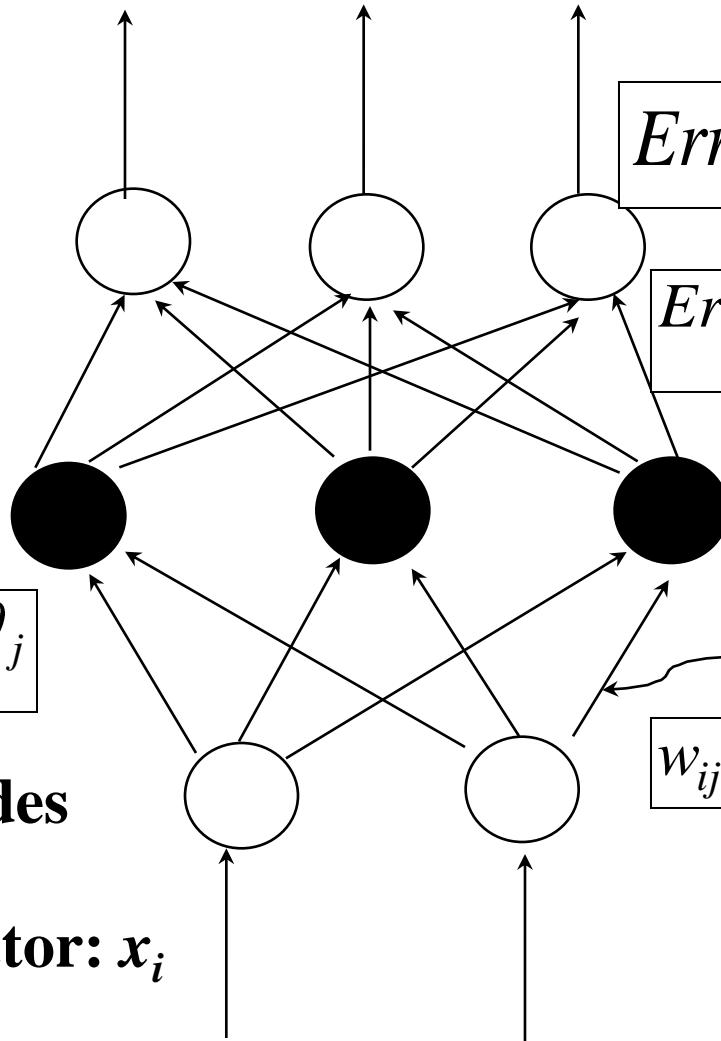
$$I_j = \sum_i w_{ij} O_i + \theta_j$$

$$w_{ij} \theta_j = \theta_j + (l) Err_j$$

**Input nodes**

$$w_{ij} = w_{ij} + (l) Err_j O_i$$

**Input vector:  $x_i$**



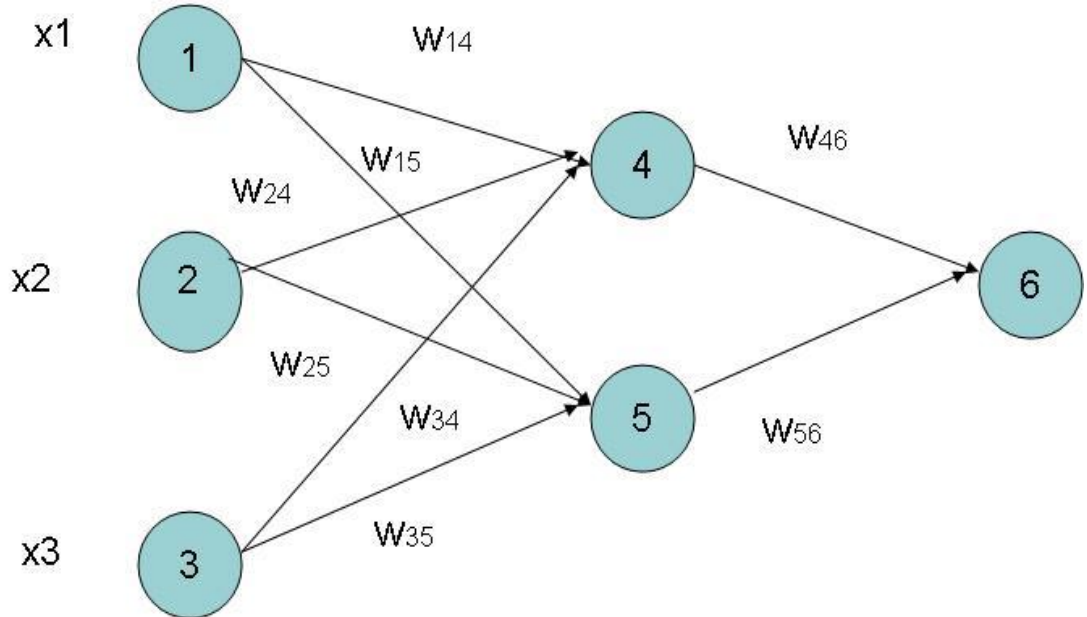
# Example of Back Propagation

Input = 3, Hidden  
Neuron = 2 Output = 1

Initialize weights :

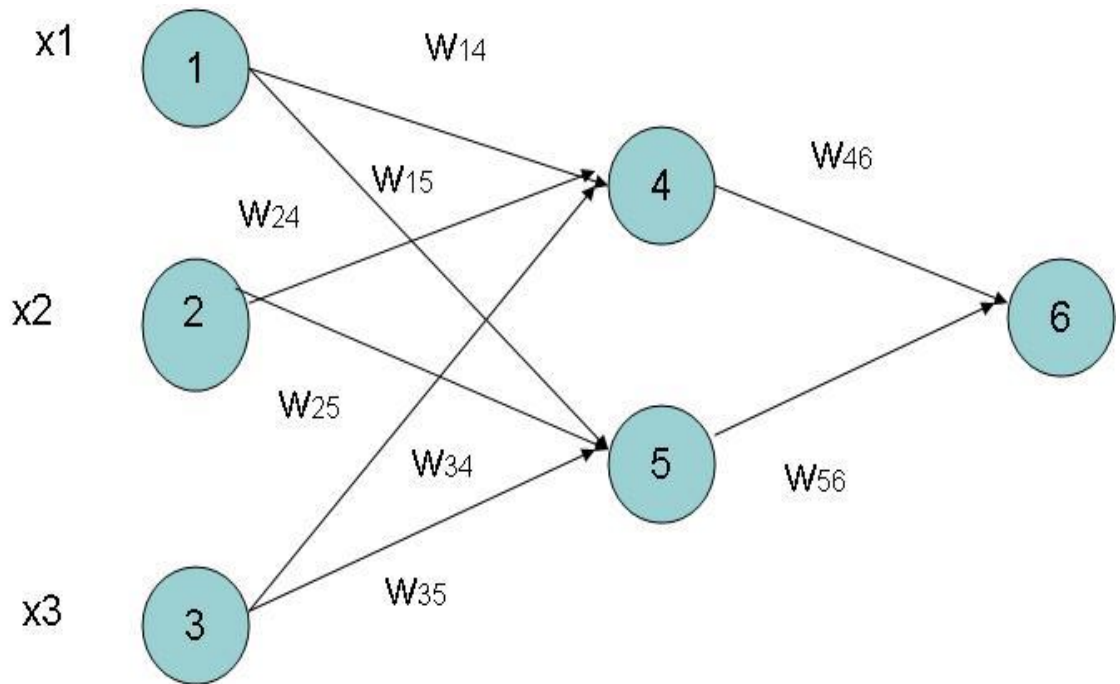
Random Numbers  
from -1.0 to 1.0

Initial Input and weight



x1	x2	x3	W14	W15	W24	W25	W34	W35	W46	W56
1	0	1	0.2	-0.3	0.4	0.1	-0.5	0.2	-0.3	-0.2

# Example of Back Propagation



- **Bias added to Hidden and output nodes**
- **Initialize Bias**
- **Bias: Random Values from -1.0 to 1.0**
- **Bias ( Random )**

$\theta_4$	$\theta_5$	$\theta_6$
-0.4	0.2	0.1

# Net Input and Output Calculation

Unit <sub>j</sub>	Net Input I <sub>j</sub>	Output O <sub>j</sub>
4	$0.2 + 0 + 0.5 - 0.4 = -0.7$	$O_j = \frac{1}{1 + e^{0.7}} = 0.332$
5	$-0.3 + 0 + 0.2 + 0.2 = 0.1$	$O_j = \frac{1}{1 + e^{-0.1}} = 0.525$
6	$(-0.3)0.332 - (0.2)(0.525) + 0.1 = -0.105$	$O_j = \frac{1}{1 + e^{0.105}} = 0.475$

# Calculation of Error at Each Node

Unit j	Error j
6	$0.475(1-0.475)(1-0.475) = 0.1311$ We assume $T_6 = 1$
5	$0.525 \times (1 - 0.525) \times 0.1311 \times$ $(-0.2) = 0.0065$
4	$0.332 \times (1-0.332) \times 0.1311 \times$ $(-0.3) = -0.0087$

# Calculation of weights and Bias Updating

Learning Rate  $\eta = 0.9$

Weight	New Values
$w_{46}$	$-0.3 + 0.9(0.1311)(0.332) = -0.261$
$w_{56}$	$-0.2 + (0.9)(0.1311)(0.525) = -0.138$
$w_{14}$	$0.2 + 0.9(-0.0087)(1) = 0.192$
$w_{15}$	$-0.3 + (0.9)(-0.0065)(1) = -0.306$
.....similarly	.....similarly
$\theta_6$	$0.1 + (0.9)(0.1311) = 0.218$
.....similarly	.....similarly

# Some Facts to be Remembered

- NNs perform well, generally better with larger number of hidden units
- More hidden units generally produce lower error
- Determining network topology is difficult
- Choosing single learning rate impossible
- Difficult to reduce training time by altering the network topology or learning parameters
- NN with Subsets (see next slides) learning often produce better results

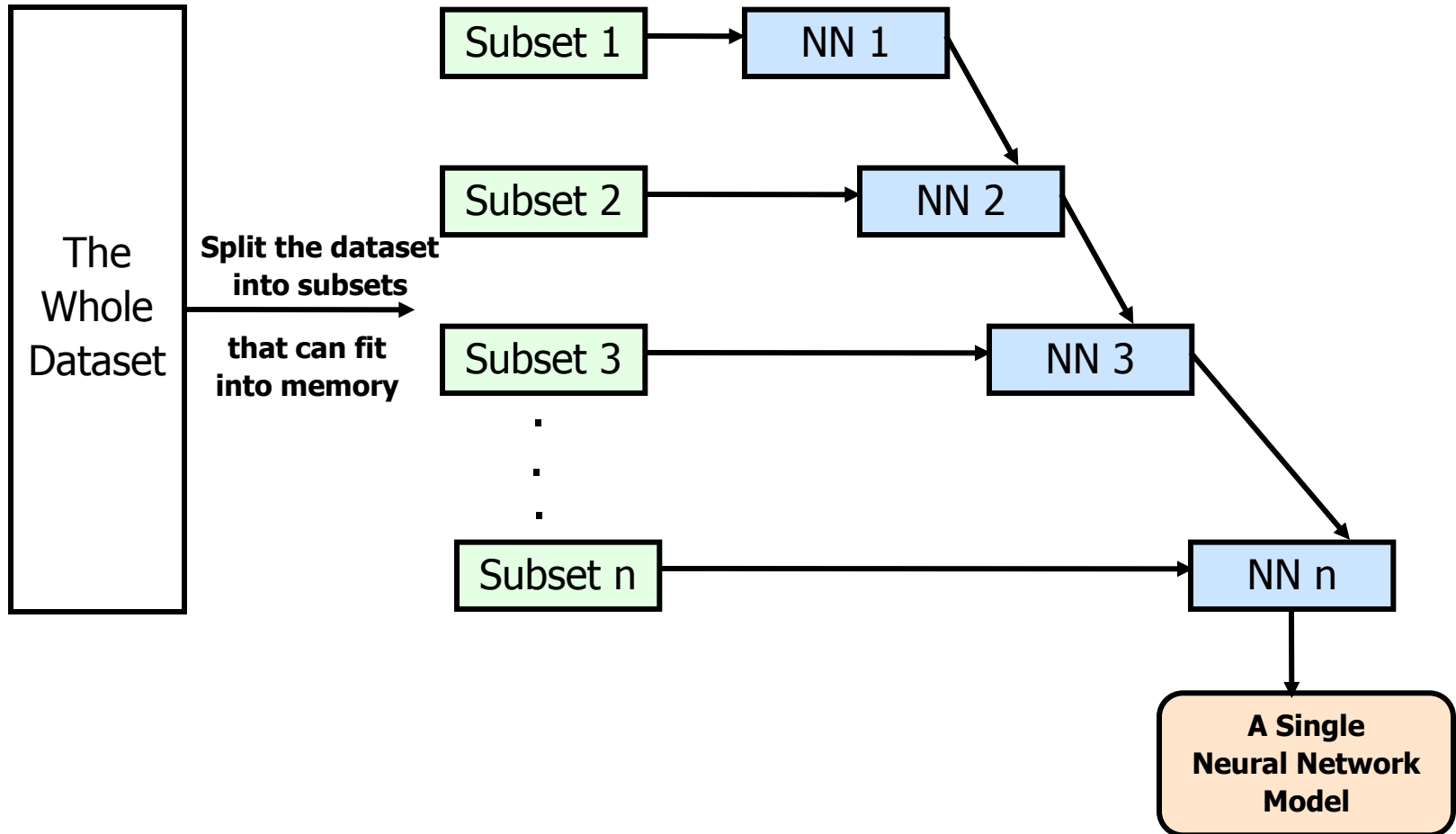
# Advanced Features of Neural Network

- Training with Subsets
- Modular Neural Network
- Evolution of Neural Network

# Training with Subsets

- Select subsets of data
- Build new classifier on subset
- Aggregate with previous classifiers
- Compare error after adding a classifier
- Repeat as long as error decreases

# Training with subsets



# Modular Neural Network

- Modular Neural Network

- Made up of a combination of several neural networks.

The idea is to reduce the load for each neural network as opposed to trying to solve the problem on a **single neural network**.

# Evolving Network Architectures

- **Small networks** without a hidden layer can't solve problems such as XOR, that are **not linearly separable**.
  - **Large networks** can easily **overfit** a problem to match the training data, **limiting** their ability to **generalize** a problem set.

# Constructive vs Destructive Algorithm

- **Constructive** algorithms take a **minimal** network and **build up** new layers nodes and connections during training.
- **Destructive** algorithms take a **maximal** network and **prunes** unnecessary layers nodes and connections during training.

# Faster Convergence

- Back propagation requires many **epochs** to converge
- (An epoch is one presentation of all the training examples in the dataset)
- Some ideas to overcome this are:
  - ***Stochastic learning***: updates weights after each example, instead of updating them after one epoch
  - ***Momentum***: This optimization is due to the fact that it speeds up the learning when the weights are moving in a single direction continuously by increasing the size of steps
  - The closer this value is to one, the more each weight change will not only include the current error, but also the weight change from previous examples (which often leads to **faster convergence**)