



Minia University

Faculty of Computers & information

# Artificial Neural Networks and Deep Learning

**Slides By:**



**A.T. Sarah Osama Talaat**



**E-mail:** SarahOsama.fci@gmail.com

Slides were prepared based on set of references mentioned in the last slide



**Lectures, FCI, Mina University**

# Agenda

- ❑ Fundamentals on learning and training samples
- ❑ Learning curve and error measurement
- ❑ When do we stop learning?
- ❑ Objective, loss and cost functions
- ❑ Gradient optimization procedures
- ❑ Stochastic gradient optimization



# Let's Start



# Supervised Learning Network Paradigms

## Error function

### □ Definition 5.1 (Total error $\text{Err}$ as a function of the weights):

- The total error increases or decreases depending on how we change the weights.

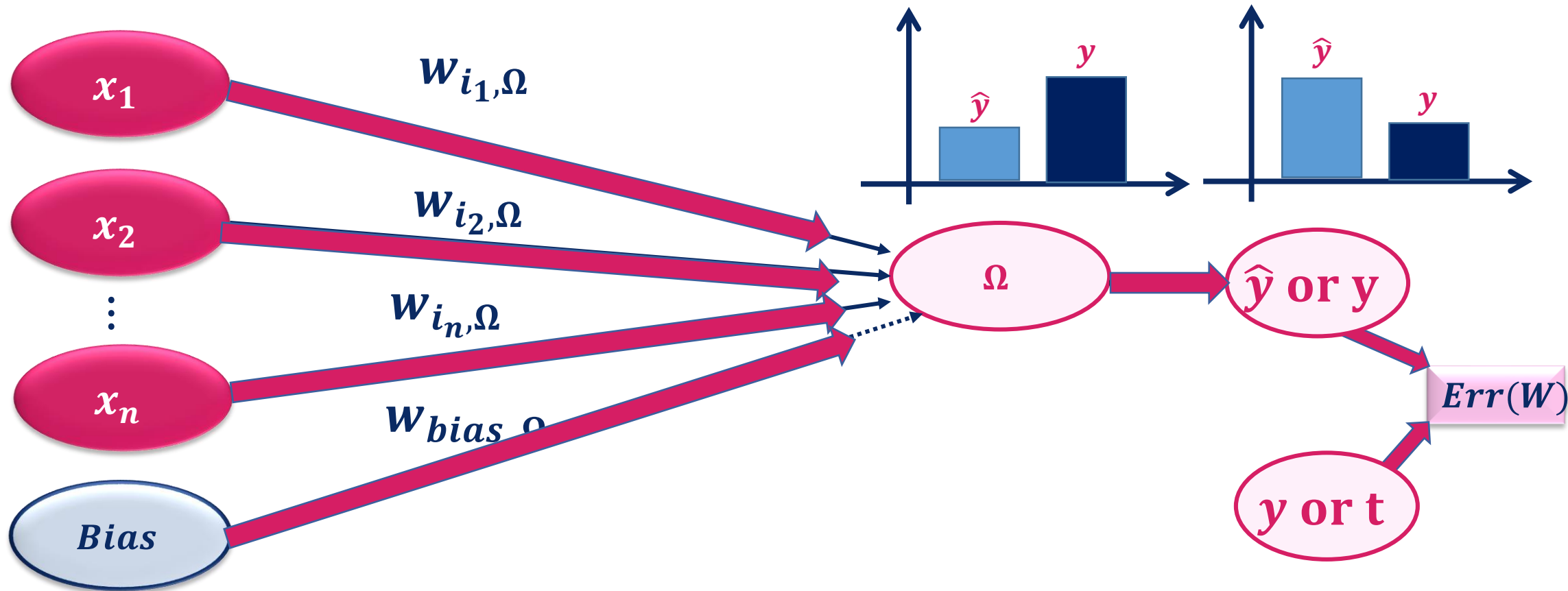
### □ Definition 5.2 (Error function):

- $\text{Err}: \mathbf{W} \rightarrow \mathbb{R}$
- Where  $\mathbf{W}$  is a set of weights and  $\mathbf{W}$  is a vector and maps the values onto the normalized output error (normalized because otherwise not all errors can be mapped onto one single  $e \in \mathbb{R}$  to perform a gradient descent). It is obvious that a **specific error function** can analogously be generated for a **single pattern  $p$**  (i.e. **single training sample**).

# Supervised Learning Network Paradigms

## Forward propagation

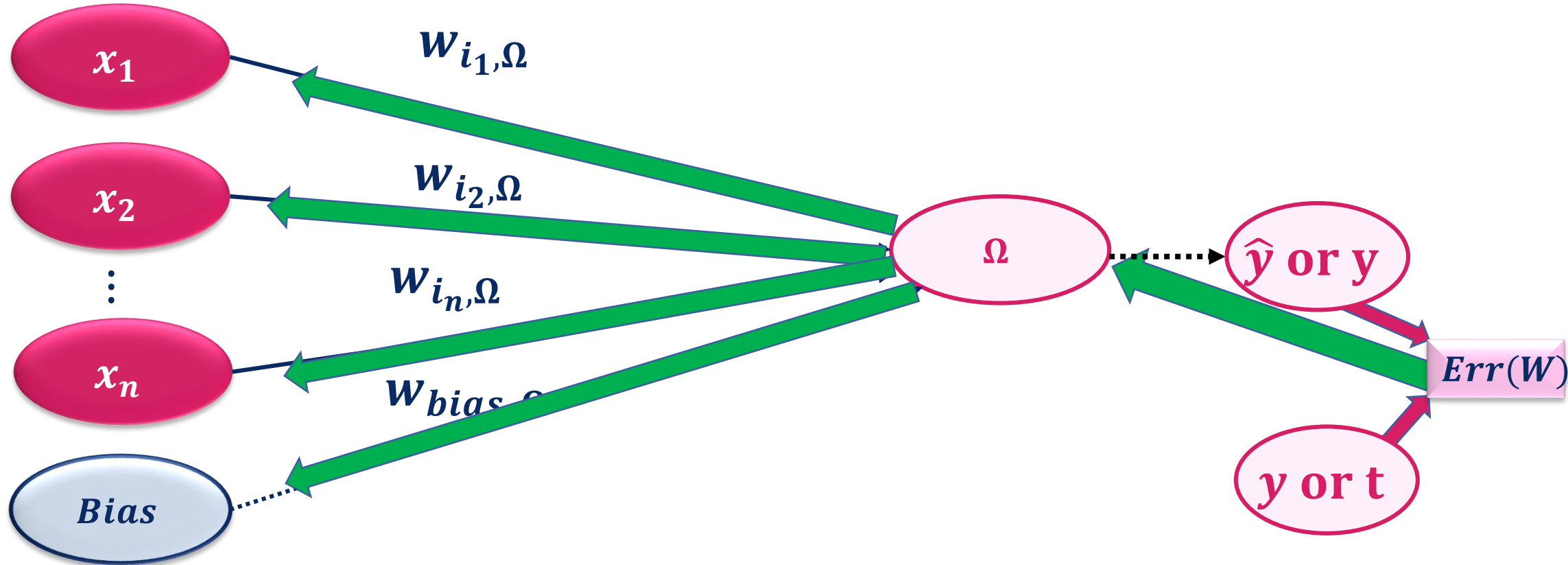
Artificial Neural Networks  
&  
Deep Learning



Figure(4.1): illustration of ANN learning when is used with dataset contains only one instance

# Supervised Learning Network Paradigms

## Backward propagation

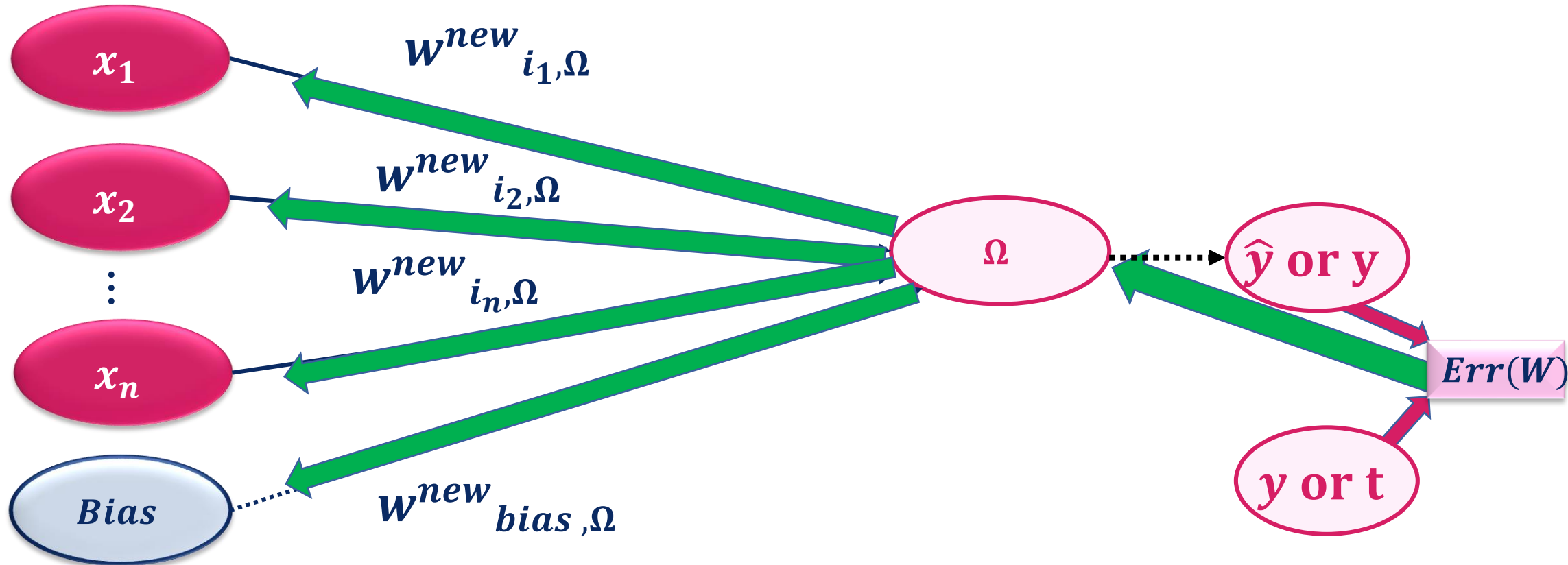


Figure(4.1): illustration of ANN learning when is used with dataset contains only one instance

# Supervised Learning Network Paradigms

## Artificial Neuronal Network learning(weight adjustment)

Artificial Neural Networks  
&  
Deep Learning



Figure(4.1): illustration of ANN learning when is used with dataset contains only one instance

# Fundamentals on learning and training samples

## Offline or online learning?

### ❑ Offline learning:

- In this learning, a **set** of training samples is presented, then the weights are changed, the total error is calculated by **means of a error function** operation or simply accumulated. Offline training procedures are also called **batch training procedures** since a batch of results is corrected all at once. Such a training section of a whole batch of training samples including the related change in weight values is called **epoch**.

### ❑ Definition 4.2 (Offline learning).

- **Several training patterns** are entered into the network at once, **the errors are accumulated** and **it learns for all patterns at the same time**.



# Fundamentals on learning and training samples

## Offline or online learning?

### □ Online learning:

- In this type of learning, after **every** sample presented the weights are changed.

### □ Definition 4.3 (Online learning):

- The network learns directly from the errors of each training sample..

# Fundamentals on learning and training samples

## □ Definition 4.4(Teaching input):

- Let  $j$  be an output neuron. The teaching input  $t_j$  is the **desired and correct** value  $j$  should output after the input of a certain training pattern. Analogously to the **vector  $p$**  the teaching inputs  $t_1, t_2, \dots, t_n$  of the neurons can also be combined into a **vector  $t$** .  $t$  always refers to a **specific training pattern  $p$**  and is, as already mentioned, contained in the set  $P$  or  $tr$  of the training patterns..

# Fundamentals on learning and training samples

## □ Definition 4.5 (Training patterns):

- A training pattern is an input vector  $p$  with the components  $p_1, p_2, \dots, p_n$  whose desired output is known. By entering the training pattern into the network we receive an output that can be compared with the teaching input, which is the desired output. The set of training patterns is called  $P$  or  $tr$ . It contains a finite number of ordered pairs  $(p, t)$  or  $(p, y)$  of training patterns with corresponding desired output.

# Fundamentals on learning and training samples

## □ Definition 4.6 (Input and output vectors):

- **Input vector  $\mathbf{x}$** , which can be entered **into** the neural network. Depending on the type of network being used the neural network will output an **output vector  $\hat{\mathbf{Y}}$** . Basically, the **training sample  $\mathbf{p}$**  is nothing more than an **input vector**. We only use it for training purposes because we know the corresponding teaching input.

## □ Definition 4.7 (Desired output vector):

- The desired output vector to the training sample is nothing more than **teaching input  $\mathbf{t}$  or  $\mathbf{y}$** .

# Fundamentals on learning and training samples

## □ Definition 4.8 (Error vector):

- Error vector  $E_p$  is the difference between the teaching input  $t$  ( $y$ ) and the actual output  $y$  ( $\hat{y}$ ).
- For several output neurons  $\Omega_1, \Omega_2, \dots, \Omega_n$  the difference between **output vector** and **teaching input** under a training input  $p$  is referred to as **error vector**, sometimes it is also called *difference vector*

$$E_p = \begin{pmatrix} t_1 - y_1 \\ \vdots \\ t_n - y_n \end{pmatrix} = \begin{pmatrix} y_1 - \hat{y}_1 \\ \vdots \\ y_n - \hat{y}_n \end{pmatrix}$$


# Fundamentals on learning and training samples

## □ Remember:

- We referred to the output values of a **neuron**  $i$  as  $o$ . Thus, the output of an output neuron  $\Omega$  is called  $o_{\Omega}$ . But the output values of a **network** are referred to as  $y_{\Omega}$  ( $\hat{y}_{\Omega}$ ).

# Fundamentals on learning and training samples

## □ Definition 4.9 (Input neuron):

- An input neuron is an identity neuron. It exactly **forwards the information received**. Thus, it represents the identity function, which should be indicated by the symbol  $/$ . Therefore the input neuron is represented by the symbol .

# Fundamentals on learning and training samples

## □ Definition 4.10 (Binary neuron):

- A **binary neuron** sums up all inputs by using the **weighted sum** as propagation function, which we want to illustrate by the sign  $\Sigma$ . Then the **activation function** of the neuron is the **binary threshold function**, which can be illustrated by  $\text{⌊}$ .



# Fundamentals on learning and training samples

## □ Definition 4.11 (Information processing neuron):

- Information processing neurons **somehow process the input information**, i.e. do not represent the identity function. It combines a **binary neuron and the activation function of binary neuron (i.e. binary threshold function)**. This leads us to the complete depiction of **information processing neurons**, namely



# Fundamentals on learning and training samples

## □ Definition 4.12 (Other neuron naming rule):

- Other neurons that use the **weighted sum** as propagation function but the **activation functions** hyperbolic tangent or Fermi function, or with a separately defined activation function  $f_{act}$ , are similarly represented by



- These neurons are also referred to as **Fermi neurons** or **Tanh neuron**.

# Fundamentals on learning and training samples

## Notations

- $\mathbf{x}$  is the input vector(matrix).
- $\mathbf{x}_i$  represents all observations of  $X_i$ , (i.e. the  $i^{th}$  vector of  $\mathbf{X}$ :  $\mathbf{x}_i = \begin{pmatrix} x_{1i} \\ \vdots \\ x_{ni} \end{pmatrix}$ ).
- $\hat{\mathbf{y}}$  or  $\mathbf{Y}$  is the output vector of a neural network which represents the predicted value of  $Y$ .
- ,  $Y = [\mathbf{y}_1, \mathbf{y}_2, \dots, \mathbf{y}_n]$  where  $\hat{\mathbf{y}} \in \mathbb{R}^{1 \times n}$
- $\mathbf{tr}$  represents training set;  $\mathbf{tr} = \{tr_i | tr_i = (x_i, y_i), i = 1, 2, 3, \dots, n\}$ , the training set is used to learn the relationship between the observations and the response.
- $(\mathbf{x}, \mathbf{y})$  or  $(\mathbf{x}, \mathbf{t})$  this pair refers to a single training example (where  $\mathbf{x} \in \mathbb{R}^{n_x \times n}$ ,  $n_x$  is an  $x$ -dimensional vector of input feature  $X$  and  $n$  is total number of training samples  $y \in Y$  )
- $(x_i, y_i)$  where  $i=1, 2, 3, \dots, |\mathbf{tr}|$  refers to the input and output for  $i$  training example.

# Fundamentals on learning and training samples

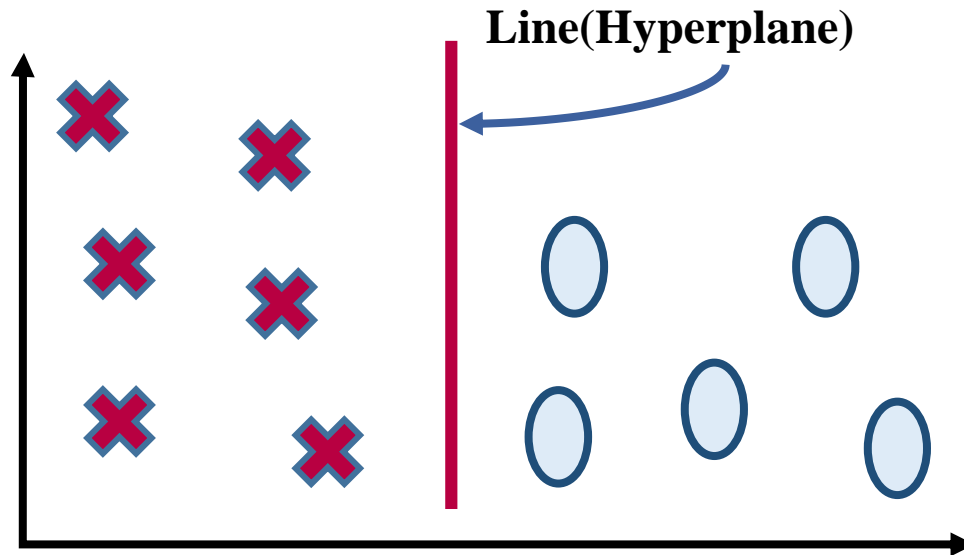
## Notations

- $\mathbf{te}$  represents testing set;  $\mathbf{te} = \{t_i | t_i = (x_i, y_i), i = 1, 2, 3, \dots, N\}$ , the testing set is used to test the result of training phase (i.e. prediction model) which is used to predict the relationship between the observations and the response.
- Output neurons are referred to as  $\Omega_1, \Omega_2, \dots, \Omega_n$
- $i$  is the input (the index of input element in  $\mathbf{x}$ ).
- $\mathbf{E_p}$  refers to error vector which represents the difference( $t-y$ ) under a certain training sample  $\mathbf{p}$ .
- $\mathbf{O}$  refers to the set of output neurons
- $\mathbf{I}$  refers to the set of input neurons.

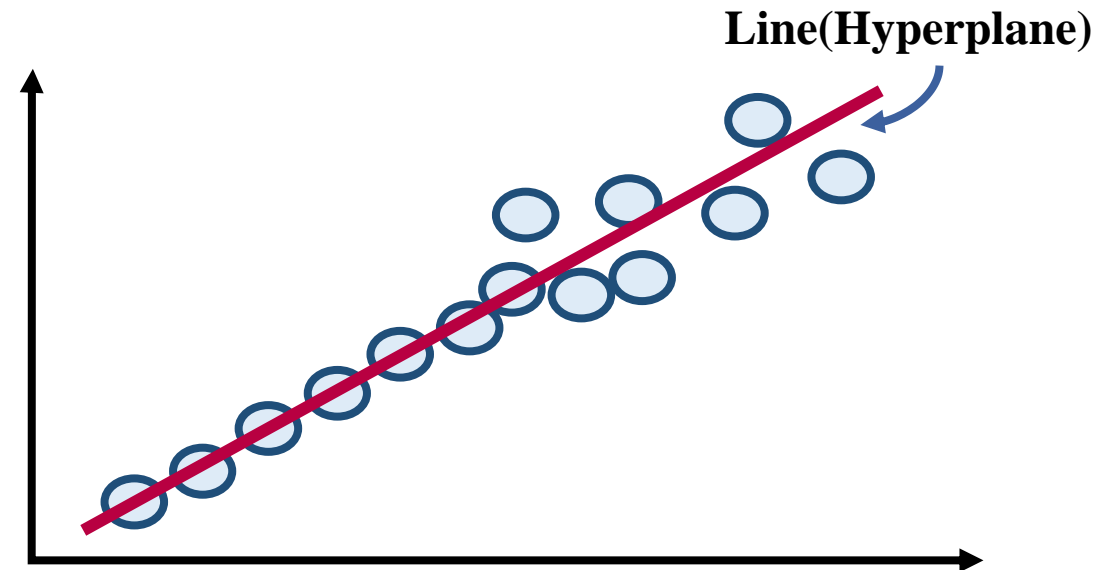
# Fundamentals on learning and training samples

## Hyperplane

□ Remember:





Linear data in 2D  
(Discrete Data)



Linear data in 2D  
(Continues Data)

# Fundamentals on learning and training samples

## Hyperplane

- ❑ **Hyperplane** is a subspace that used to perfectly separate/ fit the input data.
- ❑ It is represent the boundary between the values of first class( )  $x_1$  and the second class (  )  $x_2$  for which the machine learning algorithm gives a positive response and the value for which it gives a negative response is the **separation line**

$$b + x_1 w_1 + x_2 w_2 = 0$$

- ❑ Or (assuming that  $w_2 \neq 0$ )

$$x_2 = -\frac{w_1}{w_2} x_1 - \frac{b}{w_2}$$

# Fundamentals on learning and training samples

## Hyperplane

- The requirement for a **positive response** from the output unit is that the **net input** is receives, namely

$$b + x_1w_1 + x_2w_2 > 0$$

- During training, values of  **$w_1$** ,  **$w_2$**  and  **$b$**  are determined so that the **net** will have the correct response for the training data.

# Fundamentals on learning and training samples

## Hyperplane

- ❑ If one think in terms of a **threshold**, the requirement for a **positive response** from the output unit is that the net input is receives, namely

$$x_1 w_1 + x_2 w_2 = \theta$$

- ❑ Or (assuming that  $w_2 \neq 0$ )

$$x_2 = -\frac{w_1}{w_2} x_1 - \frac{\theta}{w_2}$$

- ❑ During training, values of  $w_1$ ,  $w_2$  are determined so that the net will have the correct response for the training data. In this case, the separating line cannot pass through the origin, but a line can be found that passes arbitrary close to the origin.



# Fundamentals on learning and training samples

## Hyperplane

- ❑ The form of the separating line found by using an adjustable bias and the form obtained by using a fixed threshold illustrate that there is no advantage to including both a bias and nonzero threshold for a neuron that uses the step function as its activation function.
- ❑ On the other hand, including neither a bias nor a threshold is equivalent to requiring the separating line(or plane or hyperplane for inputs with more components) to pass through the origin. This may or may not be appropriate for a particular problem.

# Fundamentals on learning and training samples

## Response regions

- To sum up, the **net input** to the output unit  $\Omega$  is

$$net_i = b + \sum_{i=1}^n w_i x_i$$

- It is easy to see that the boundary between the region where  $net_i > 0$  and the region where  $net_i < 0$ , which we call the **decision boundary**, is determined by the relation

$$b + \sum_{i=1}^n w_i x_i = 0$$

- Depending on the number of input units in the network, this equation represents a **line, a plane, or a hyperplane**.

# Fundamentals on learning and training samples

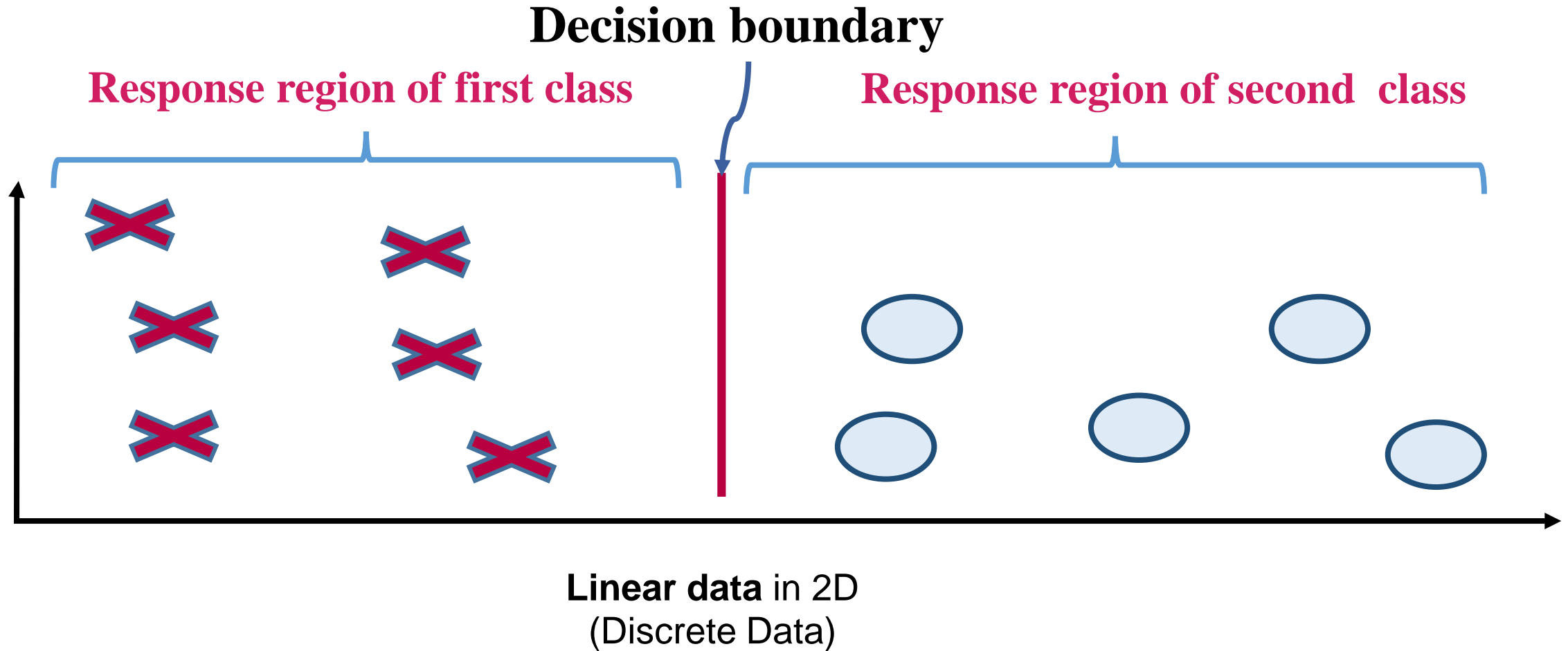
## Response regions

- ❑ As I mentioned earlier, if there are a weights and a bias so that all of the training input vectors for which the correct response is +1(first class) lie on one side of the **decision boundary** and all of the training input vectors for which the correct response is -1 (second class) lie on the other side of **decision boundary** here we say that the problem is **linearly separable**.
- ❑ The single layer networks can learn only the linear separable problems.
- ❑ The region where net is positive is separated from the region where it is negative by the line

$$x_2 = -\frac{w_1}{w_2}x_1 - \frac{b}{w_2}$$

# Fundamentals on learning and training samples

## Response regions

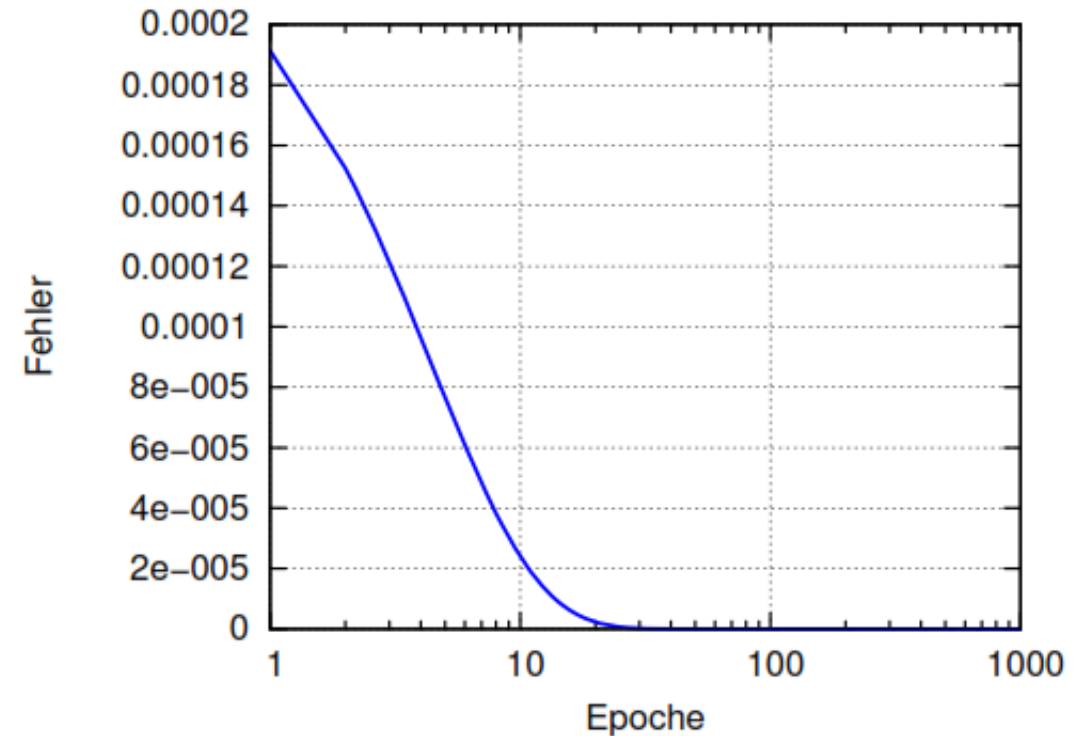
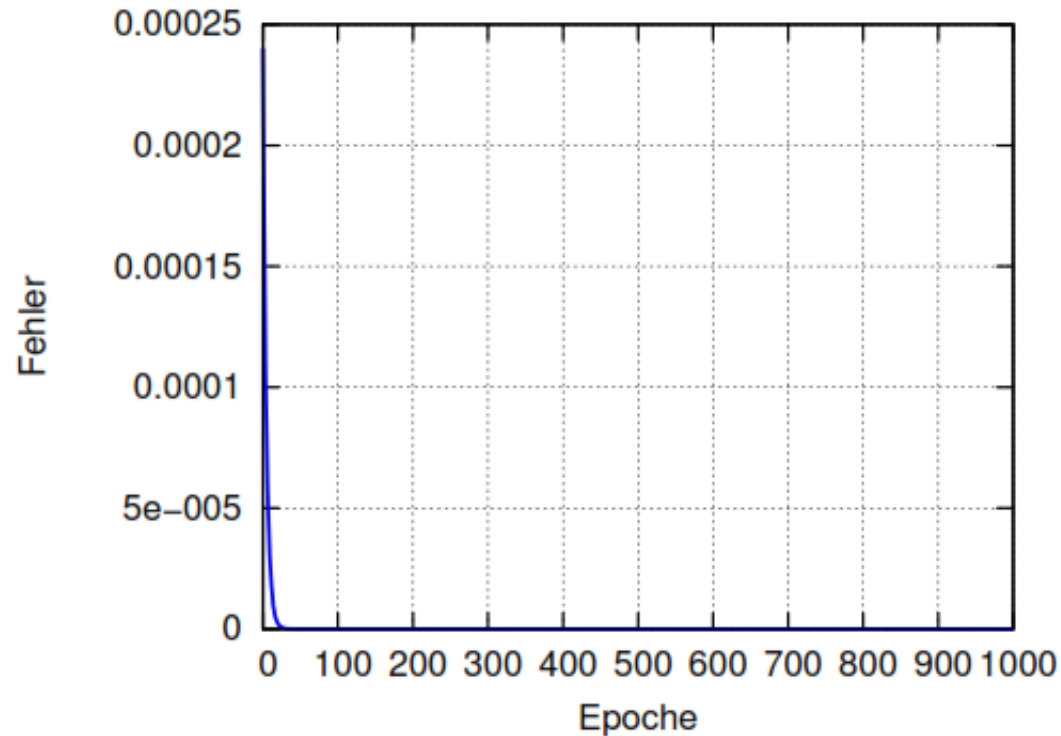


# Learning Curve and Error Measurement

## □ Definition 4.13 (Learning curve):

- The learning curve indicates the **progress of the error**, which can be determined in various ways. The motivation to create a learning curve is that such a curve can indicate whether **the network is progressing or not**. For this, the error should be **normalized**, i.e. represent a distance measure between the correct and the current output of the network.

# Learning Curve and Error Measurement



**Figure(4.1): Examples of a learning curve**

# Learning Curve and Error Measurement

## □ Definition 4.14 (Specific error):

- The specific error  $\mathbf{Err}_p$  is based on a single training sample, which means it is generated **online**.

$$Err_p = \frac{1}{2} \sum_{\Omega \in O} (t_{\Omega} - y_{\Omega})^2 ,$$

where  $\Omega$  be output neurons and  $O$  the set of output neurons

# Learning Curve and Error Measurement

## □ Definition 4.15 (Euclidean distance):

- The Euclidean distance between two vectors *t and y* is defined as

$$Err_p = \sqrt{\sum_{\Omega \in O} (t_{\Omega} - y_{\Omega})^2},$$

where  $\Omega$  be output neurons and  $O$  the set of output neurons



# Learning Curve and Error Measurement

## □ Definition 4.16 (Root Mean Square):

- The RMS of two vectors *t* and *y* is defined as

$$Err_p = \sqrt{\frac{\sum_{\Omega \in \mathcal{O}} (t_{\Omega} - y_{\Omega})^2}{|\mathcal{O}|}},$$

where  $\Omega$  be output neurons and  $\mathcal{O}$  the set of output neurons

# Learning Curve and Error Measurement

## □ Definition 4.17 (Total Error):

- The total error  $Err$  is based on all training samples, that means it is generated **offline**.

$$Err = \sum_{p \in P} Err_p$$

# Learning Curve and Error Measurement

## □ Remember:

- Analogously we can generate a **total RMS** and a **total Euclidean distance** in the course of a whole epoch. Of course, it is possible to use other types of error measurement.

# When do we stop learning?

- Generally, the training is stopped when the **user in front of the learning computer "thinks" the error was small enough.**
- That is **depends on** a more objective view on the comparison of **several learning curves.**
- On the other hand, it can be possible that a **curve descending fast** in the beginning can, after a longer time of learning, be **overtaken by another curve**: This can indicate that either the learning rate of the worse curve was too high or the worse curve itself simply got stuck in a **local minimum**, but was the first to find it.
- **Remember: Larger error values are worse than the small ones.**

# When do we stop learning?

- When the network eventually begins to **memorize the samples**, the shape of the **learning curve can provide an indication**: If the learning curve of the **testing** samples is **suddenly and rapidly rising** while the **learning** curve of the **testing data** is **continuously falling**, this could indicate **memorizing and a generalization getting poorer and poorer**. At this point it could be decided whether the network has already learned well enough at the next point of the two curves, and maybe the final point of learning is to be applied here (this procedure is called **early stopping**).

# When do we stop learning?

- To sum up, we need to generate a **learning curve** in respect of the **training data** not only that but also, we need to plot the **testing data** to get a second learning curve, which generally provides values that are **slightly worse** and with stronger oscillation. But with good generalization the curve can decrease, too.

# Objective Function

- Objective function is used to measure the performance of machine learning algorithms. This function is also called **criterion**.
- Usually, we want to minimize or maximize this function.
- When we are minimizing it, we may also call it the **cost function, loss function, or error function**.
- Sometimes, the **cost function and loss function are deferent as we are going to study**.

- **Loss (error) functions** are used to estimate how well the algorithm doing in a **single training sample** (i.e. it is defined respected to a single training example).
- **Cost functions** are used to estimate **how well the algorithm parameters are doing during all training set** (the cost over all algorithm parameters). **The cost function has its own curve and its own gradients. The slope of this curve tells us how to update our parameters to make the model more accurate.**



# Gradient Optimization Algorithm

## Introduction

- The ANN learns by **backpropagation** of the cost function.
- In order to establish the mathematical basis for some of the following learning procedures (i.e. algorithms) I want to explain briefly what is meant by **gradient descent: the backpropagation of error learning procedure**, for example, involves this mathematical basis and thus inherits the advantages and disadvantages of the gradient descent.

# Gradient Optimization Algorithm

## Introduction

- Gradient descent algorithms are generally **used where we want to maximize or minimize n-dimensional functions.**
- Gradient Descent is a general function for minimizing a **cost function** (like Mean Squared Error or Root Mean Square Error)
- Gradient Descent basically just does what we were doing by hand **change the learning algorithm parameters values, bit by bit**, until we hopefully arrived a **minimum.**

# Gradient Optimization Algorithm

## Definition

### □ Definition 4.19 (Gradient decent)

- Let  $f$  be an  $n$ -dimensional function and  $s = (s_1, s_2, \dots, s_n)$  the given starting point.

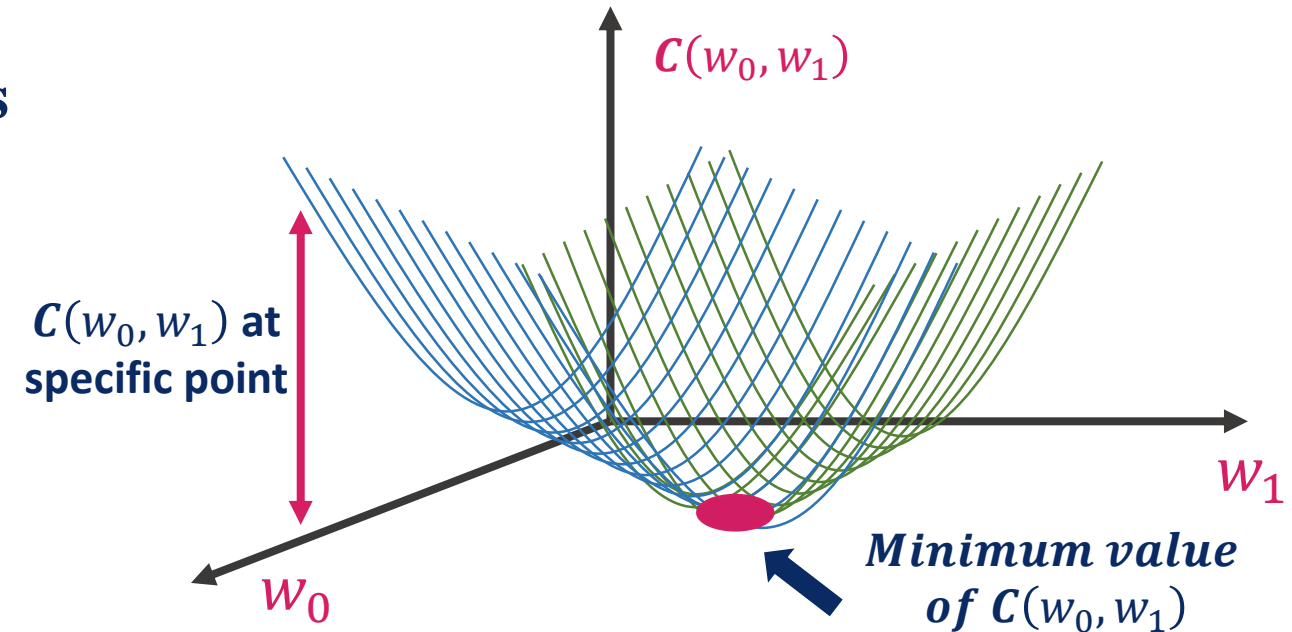
**Gradient descent** means going from  $f(s)$  **against the direction of the gradient  $g$** , i.e. towards  $-g$  with steps of the size of  $|g|$  towards smaller and smaller values of  $f$ .

□ **Gradient descent algorithm** are not an **errorless optimization algorithm** at all (as we will see in the following lectures) – however, they work still well on many problems, which makes them an optimization paradigm that is frequently used.

# Gradient Optimization Algorithm

## Graphical Description

- Assume that, we have two learning algorithm parameters  $w_0, w_1$  and the cost function  $C(w_0, w_1)$
- We want to find  $w_0, w_1$  that minimize the cost function  $c$ ,  $C(w_0, w_1)$
- $C(w_0, w_1)$  is a convex function which is like a bowl

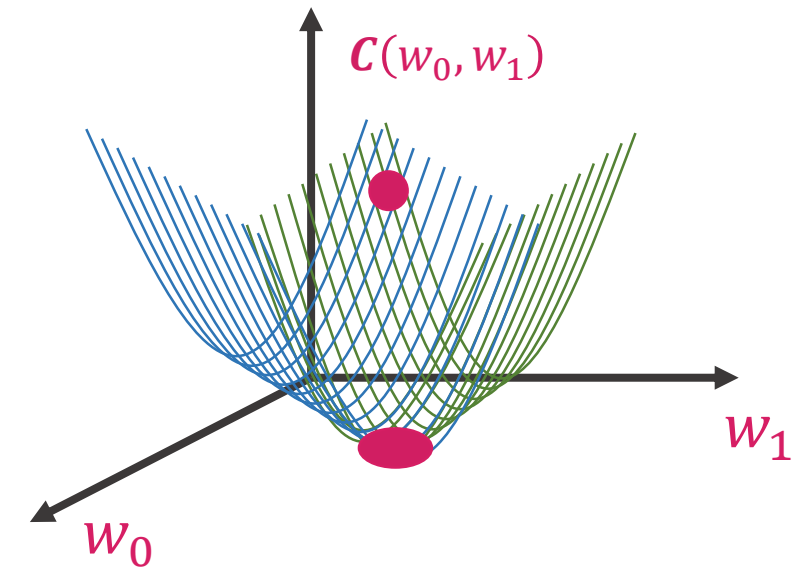


# Gradient Optimization Algorithm

## Graphical Description

### □ Gradient descent:

- What we can do to find a good values for the parameters  $w_0, w_1$ ?
- Initialize  $w_0$  and  $w_1$  to some initial values (denoted by pink dot). There many difference initialization methods; random values or assign the values to zero.
- For any **convex cost function** always assign the values to zeros because we need to start from the same points

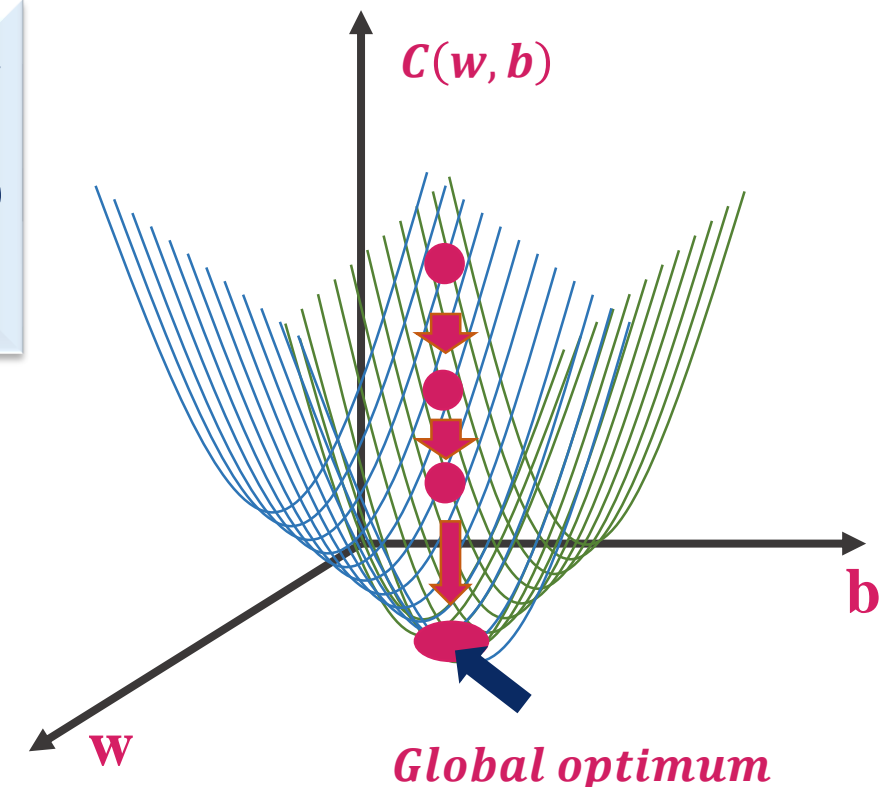


# Gradient Optimization Algorithm

## Graphical Description

- As mentioned before the gradient descent is defined by the following

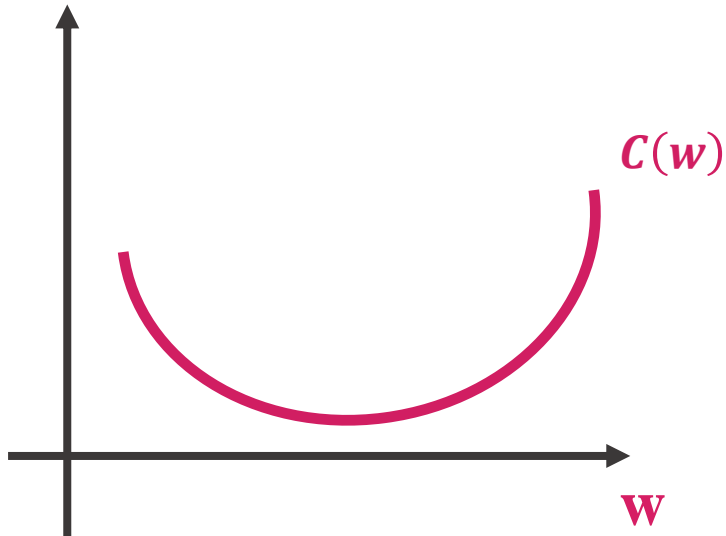
**Gradient descent** starts from staring initial point  $s = (s_1, s_2, \dots, s_n)$  and then takes a step in the (steepest) **against direction** (downhill) of  $s$



# Gradient Optimization Algorithm

## Graphical Description

- For illustration, to make one diminution plot suppose that, we have a learning algorithm that has only one parameter  $w$ . And also, assume that we have a cost function  $C(w)$  and we need to minimize this function



### Gradient descent

Repeat until convergence{

$$w := w - \eta \frac{\partial C(w)}{\partial w}, \text{ or}$$

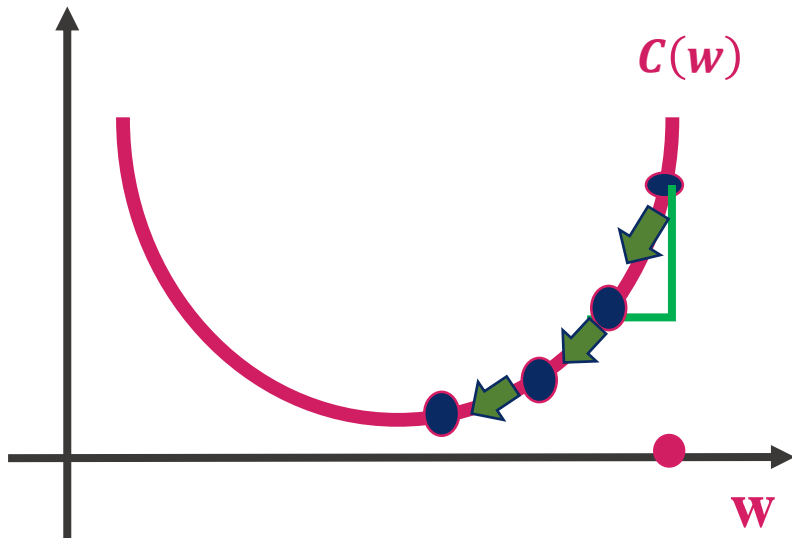
$$w := w - \eta \partial w,$$

}

# Gradient Optimization Algorithm

## Graphical Description

- We start by initializing that algorithm parameter  $w$  to any value, say 0 for both, and go from there. Formally, the gradient algorithm is as follows:



$$\frac{d C(w)}{d W} > 0$$

### Gradient descent

Repeat until convergence{

$$w := w - \eta \frac{\partial C(w)}{\partial w}, \text{ or}$$

$$w := w - \eta \partial w,$$

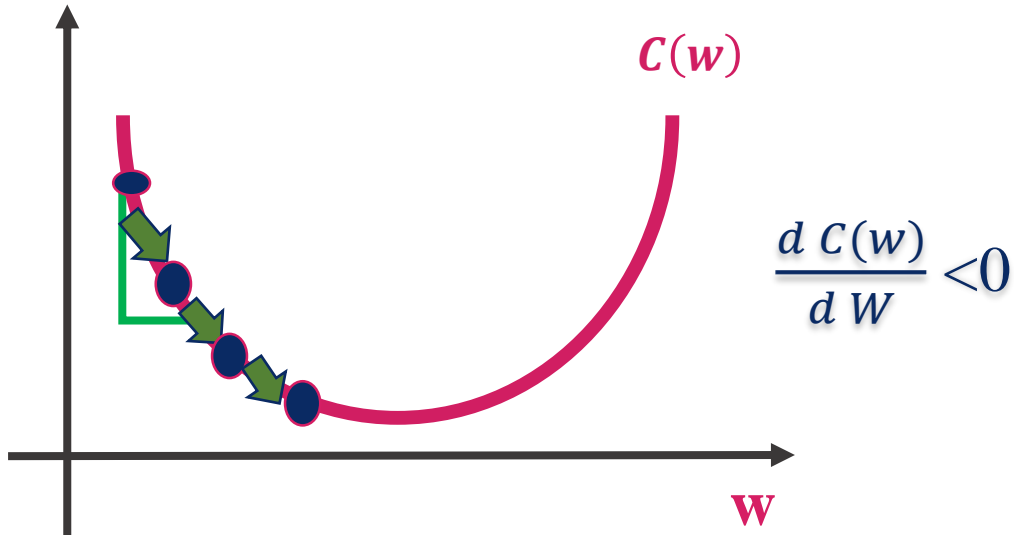
}



# Gradient Optimization Algorithm

## Graphical Description

- For illustration, to make one diminution plot suppose that, we have a learning algorithm that has only one parameter  $w$ . And also, assume that we have a cost function  $C(w)$  and we need to minimize this function



### Gradient descent

Repeat until convergence{

$$w := w - \eta \frac{\partial C(w)}{\partial W}, \text{ or}$$

$$w := w - \eta \partial w,$$

}

# Gradient Optimization Algorithm

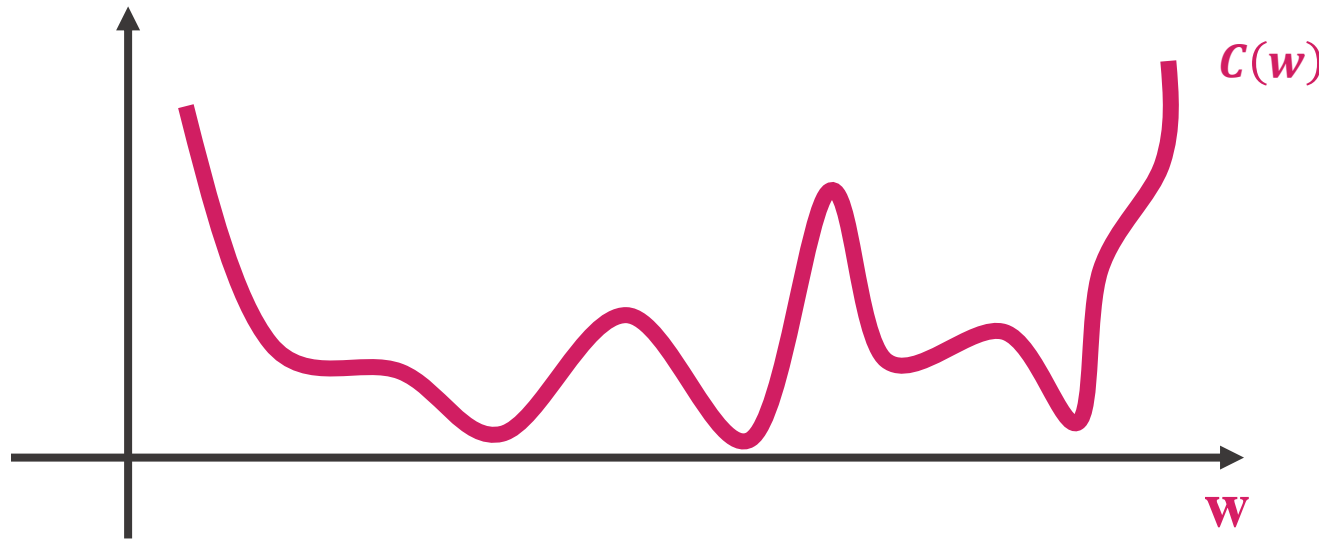
## □ Algorithm 4.1: Gradient Decent algorithm

1. **Initialization:** Initialize all parameters to any random number (usually be 0)
2. **Repeat until convergence (or gradient gets close to zero)**
3. **Activation:** compute the neuron input  $net_i = \sum_{i=0}^n w_i x_i$ , where  $i = 0$  is for the bias
4. Calculate the neuron output by applying one of activation functions to the neuron input
$$\hat{y} = o_i = a(net_i)$$
5. **Create the derivative for all parameters  $p_i$**
6. 
$$dp_i := \frac{dC}{dp_i}$$
7. **Learning : adjust the parameters values**
8. 
$$p_i := p_i - \eta \frac{dC}{dp_i}$$
, where  $\eta$  is the learning rate
9. **End for**

# Stochastic Gradient Optimization Algorithm

## Graphical Description

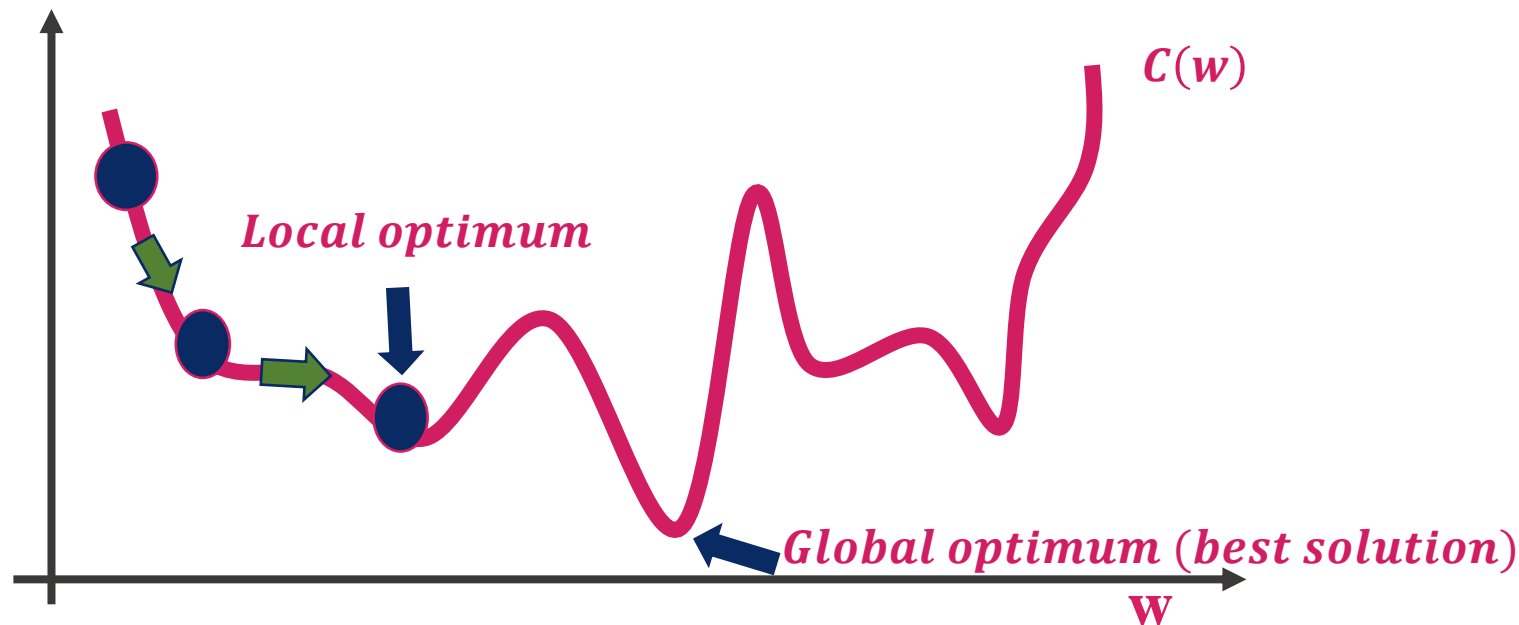
- Assume that, we have two learning algorithm parameters  $w_0, w_1$  and the cost function  $C(w_0, w_1)$ . And, we want to find  $w_0, w_1$  that minimize the cost function  $c$ ,  $C(w_0, w_1)$
- $C(w_0, w_1)$  is a non-convex function which looks smoothing i.e. curves up and down



# Stochastic Gradient Optimization Algorithm

## Graphical Description

- Assume that, we have two learning algorithm parameters  $w_0, w_1$  and the cost function  $C(w_0, w_1)$ . And, we want to find  $w_0, w_1$  that minimize the cost function  $c$ ,  $C(w_0, w_1)$
- $C(w_0, w_1)$  is a non-convex function which looks smoothing i.e. curves up and down



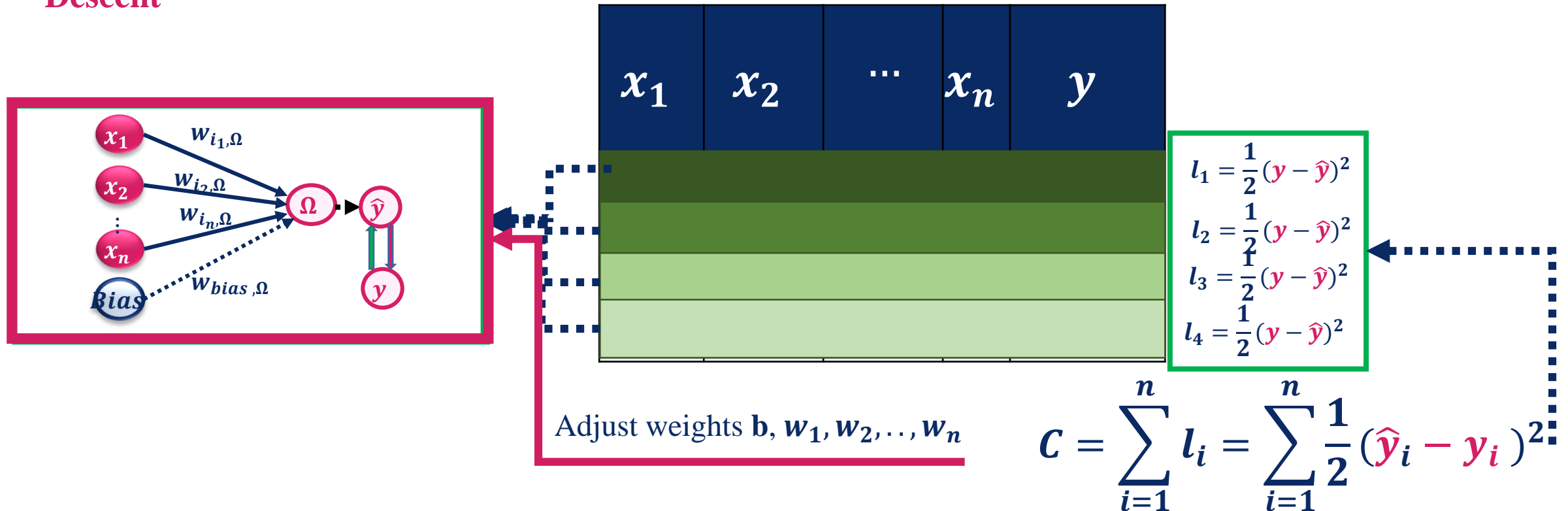
# Stochastic Gradient Optimization Algorithm

## □ Algorithm 4.1: Stochastic Gradient Decent algorithm

1. **Initialization:** Initialize all parameters to any small random number close to 0 (but not 0)
2. **Foreach instance in the input dataset do**
3.     **Forward-propagation:** From left to right the neurons are activated by the following
4.     **Activation:** compute the neuron input  $y_i = net_i = \sum_{i=0}^n w_i x_i$ , where  $i = 0$  is for the bias
5.     Calculate the neuron output by applying one of activation functions to the neuron input
$$\hat{y}_i = o_i = a(net_i)$$
6.     **Measure the error by compare the predicted result  $\hat{y}_i$  to the actual result  $y_i$  (Cost Function)**
7.     **Back-propagation:** From right to left the error is back-propagated and adjust the weights
8.     **Learning : adjust the parameters values**
9.      $p_i := p_i - \eta \frac{dC}{dp_i}$ , where  $\eta$  is the learning rate
10. **End foreach**

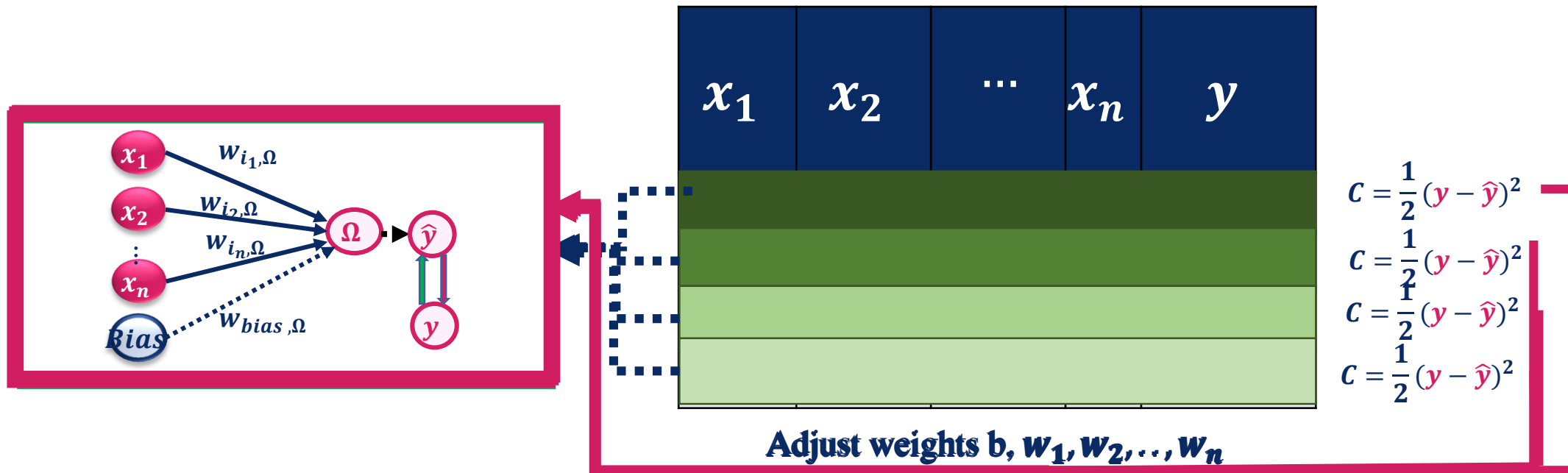
# Normal vs. Stochastic Gradient Optimization Algorithm

- When we have an ANN that learns by using an **offline learning** the input instances are being plugged into the same ANN every time. Not only that but also, the **activation function** is calculated over **all dataset**, after that we adjust weights (learning). **This process is called the Gradient Descent or Batch Gradient Descent**



# Normal vs. Stochastic Gradient Optimization Algorithm

- When we have an ANN that learns by using an **online learning** the input instances are being plugged into the ANN instance by instance. And the **activation function** is calculated **for each input**, then we adjust weights (learning) after each input. **This process is called the Stochastic Gradient Descent**



# Normal vs. Stochastic Gradient Optimization Algorithm

- **Stochastic Gradient Descent** help us to avoid the **local optimum** rather than the over all global minimum that is because the stochastic gradient descent has vary higher fluctuations the reason behind that it is **doing one iteration or use one training instance at a time and that is make the fluctuations are very higher**. So it is much more likely to find the global optimum rather than the local optimum. **Not only that but it is faster that is because it is does not have to load up all input dataset into the memory.**



# References

- Kriesel, David. "A Brief Introduction to Neural Networks. 2007." URL <http://www.dkriesel.com> (2007).
- <https://www.coursera.org/lecture/neural-networks-deep-learning>
- <https://www.udemy.com/machinelearning/learn/v4/t/>

**Any Questions!?**



*Thank you*