



MORE ON PERCEPTRON LEARNING

INT301 Bio-computation, Week 4, 2021





Outline

- **Gradient Descent Rule**

The Gradient Descent Rule

- Perceptron rule fails **if data is not linearly separable**
- **Idea:** uses **gradient descent** to search the **hypothesis space**
 - perceptron rule cannot be used (not differentiable)
 - hence, an **unthresholded linear unit** is an appropriate error measure:

$$E(w) = \frac{1}{2} \sum_e (y_e - o_e)^2$$

- To understand gradient descent, it is helpful to visualize the entire hypothesis space with
 - all possible weight vectors
 - associated E values

The Gradient Descent Rule

- The objective is to minimize the following error:

$$E(w) = \frac{1}{2} \sum_e (y_e - o_e)^2$$

- The training is a process of minimizing the error $E(w)$ in the steepest direction (most rapid decrease), *that is in direction opposite to the gradient:*

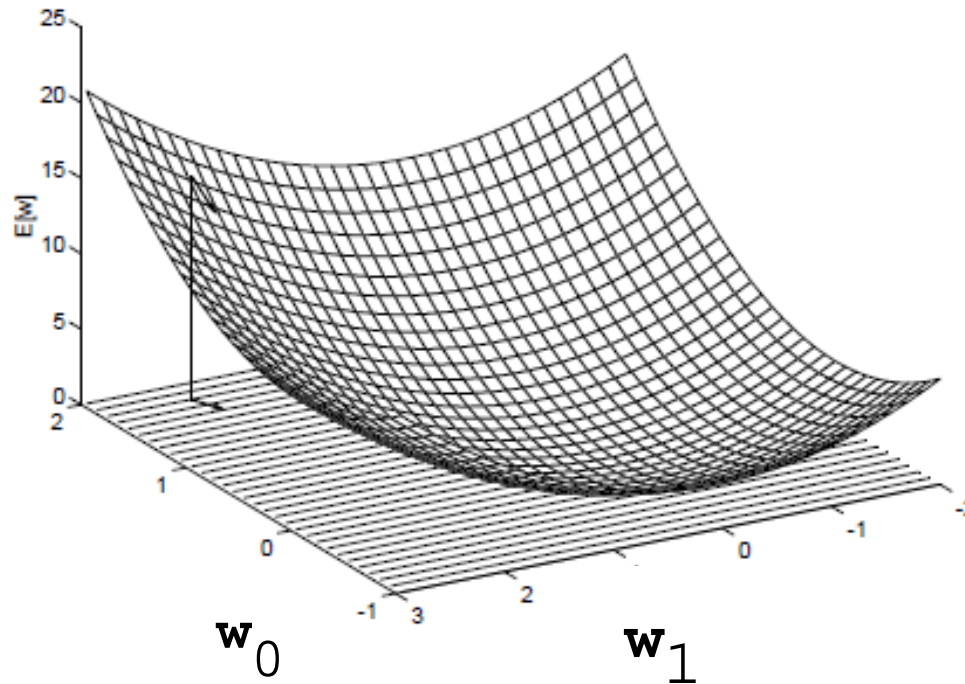
$$\nabla E(w) = [\partial E / \partial w_0, \partial E / \partial w_1, \dots, \partial E / \partial w_d]$$

which leads to the **gradient descent training rule:**

$$w_i = w_i - \eta \partial E / \partial w_i$$

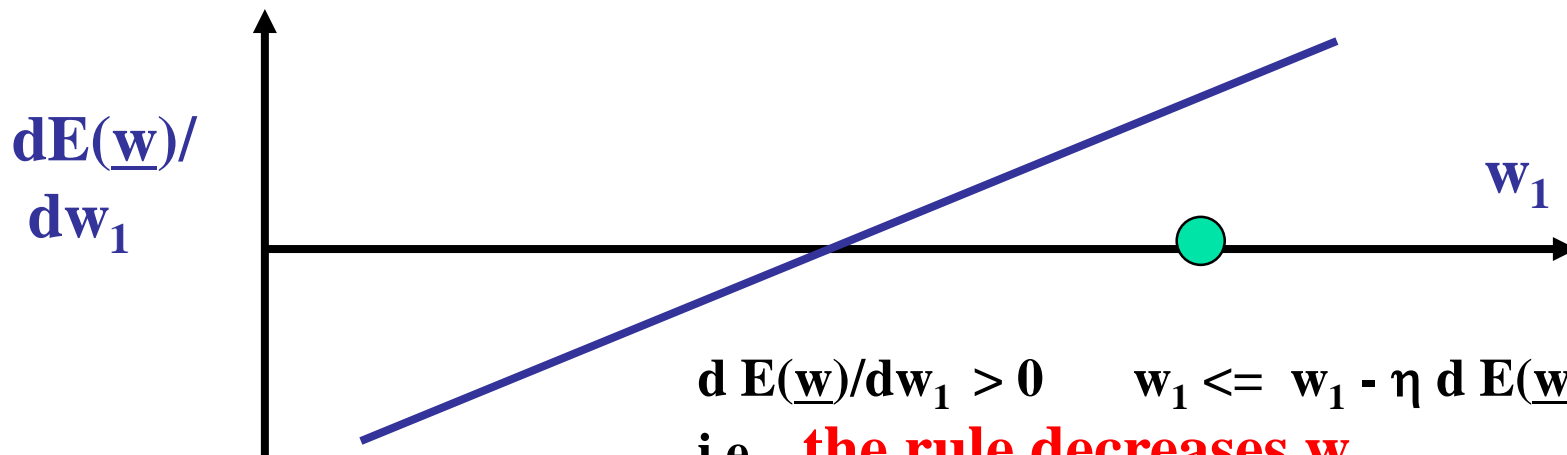
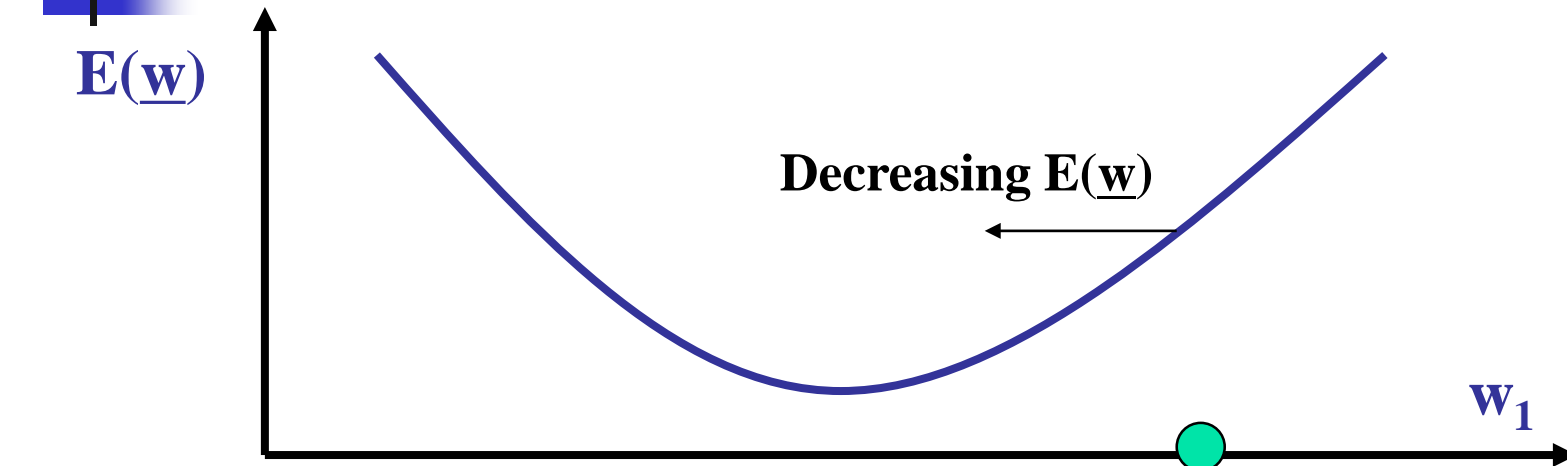
Error Surface

- the axes w_0 , w_1 represent possible values for the two weights of a simple linear unit



error surface must be **parabolic** with a **single global minimum**

Moving Downhill: Move in direction of negative derivative



$\frac{dE(\underline{w})}{dw_1} > 0 \quad w_1 \leq w_1 - \eta \frac{dE(\underline{w})}{dw_1}$
i.e., **the rule decreases w_1**

Moving Downhill: Move in direction of negative derivative

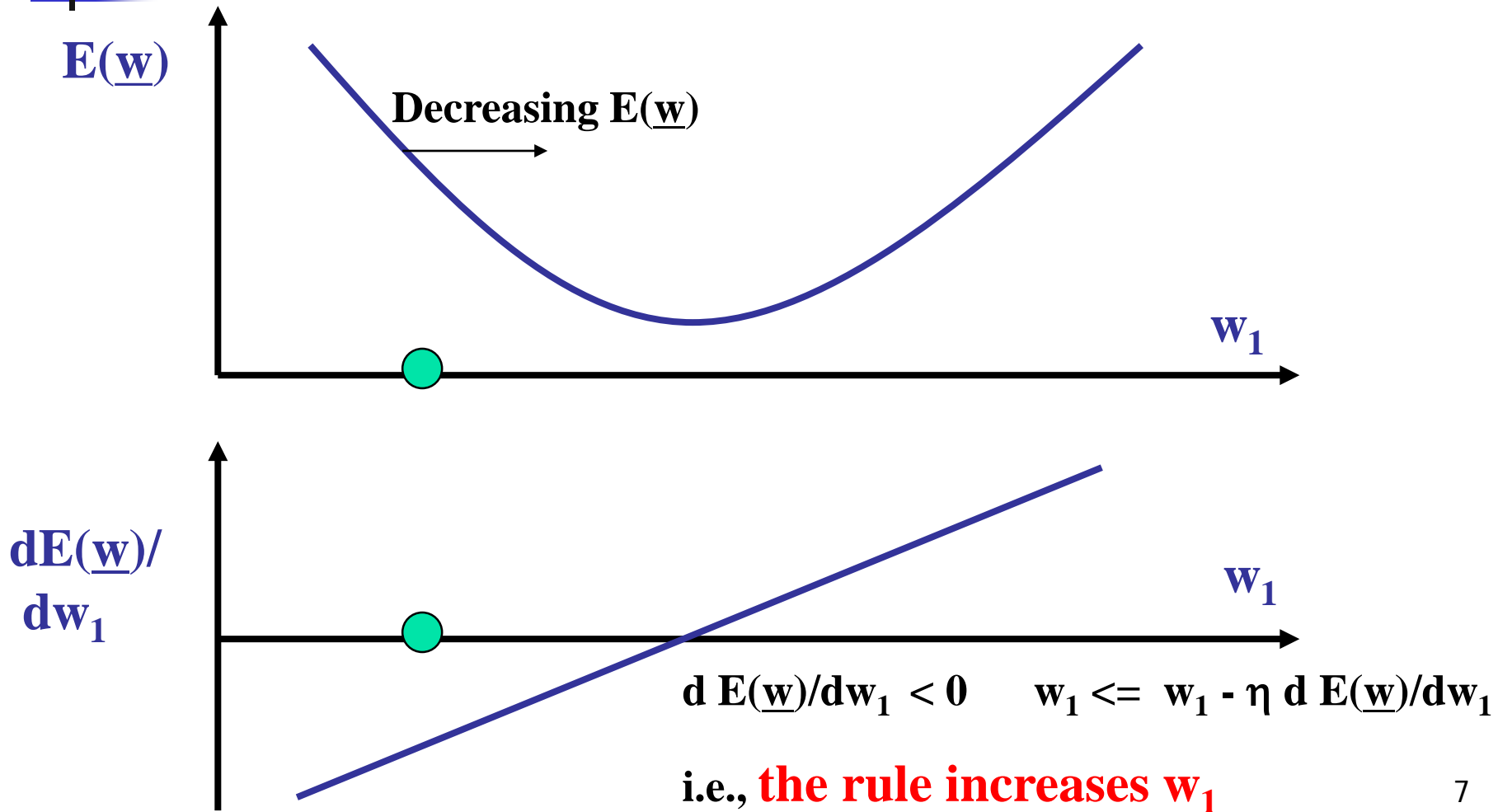


Illustration of Gradient Descent

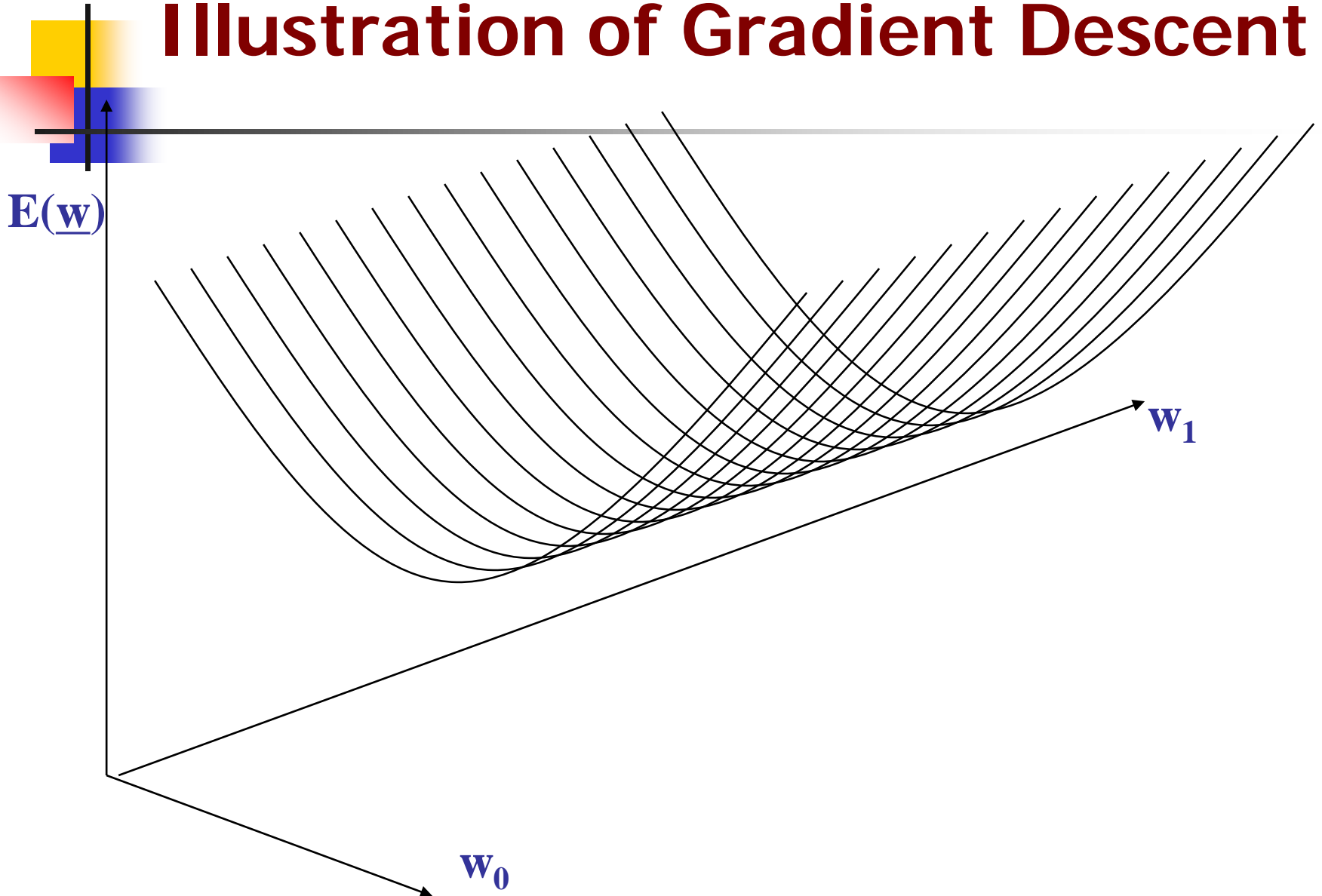


Illustration of Gradient Descent

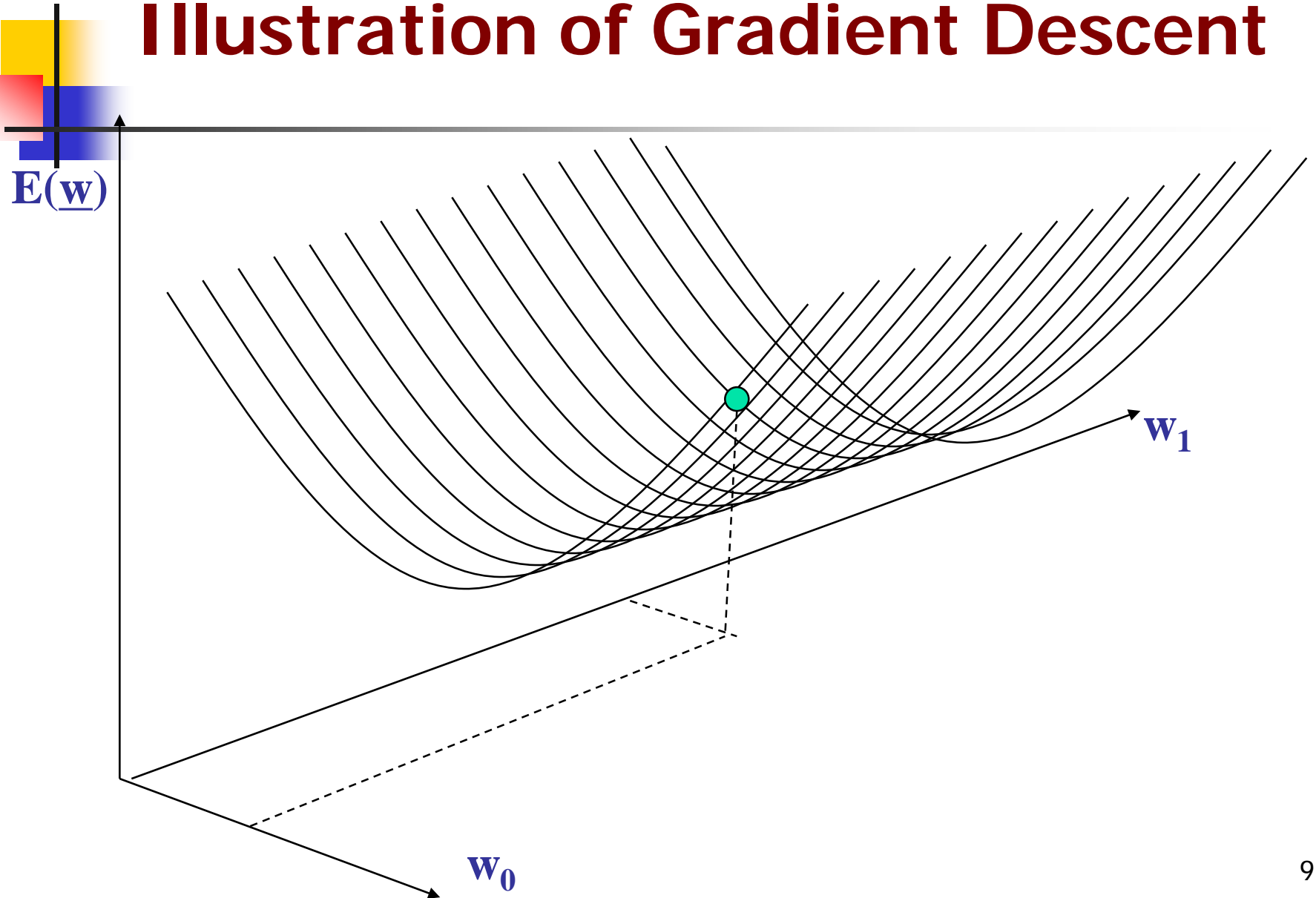


Illustration of Gradient Descent

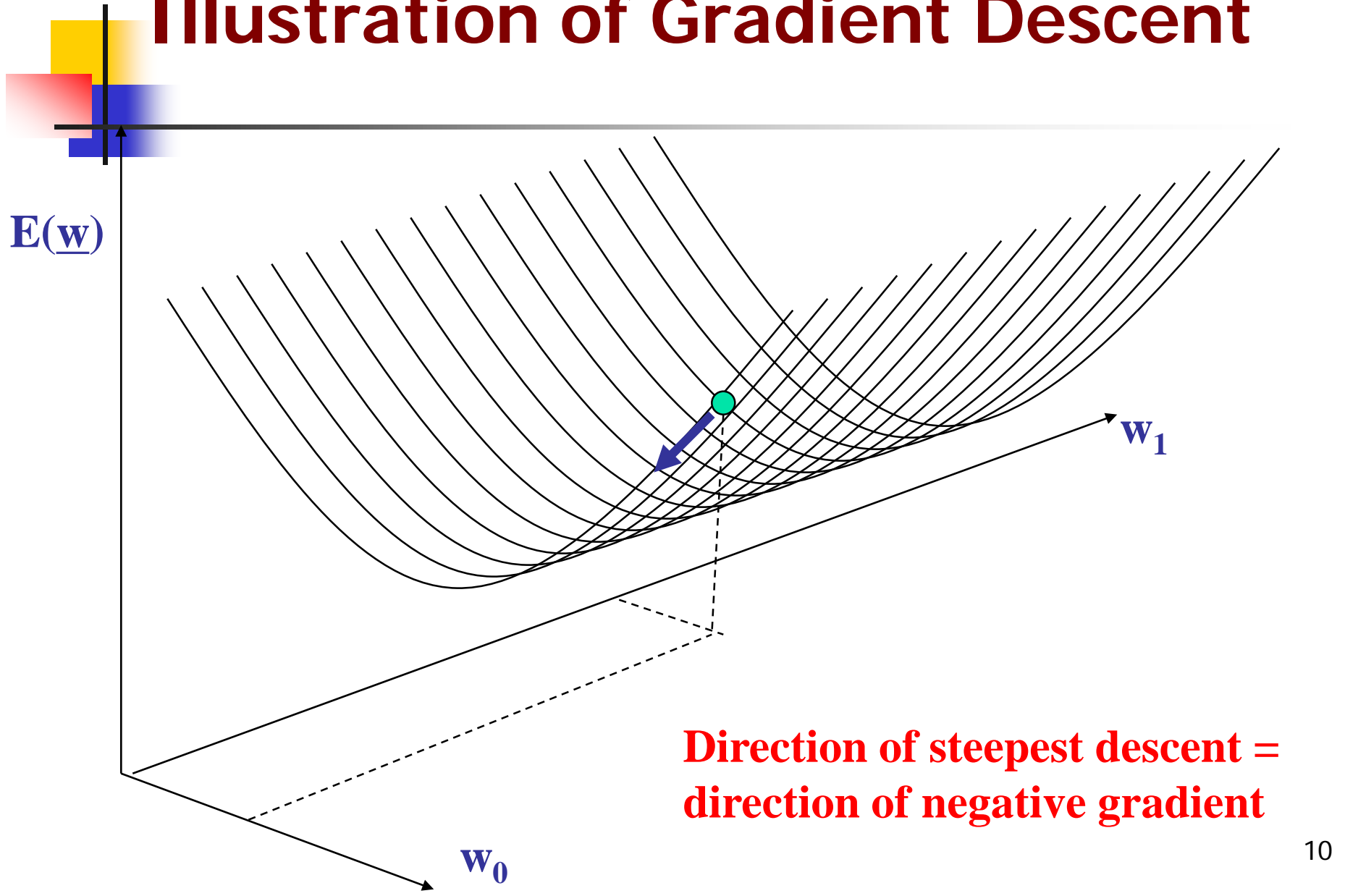
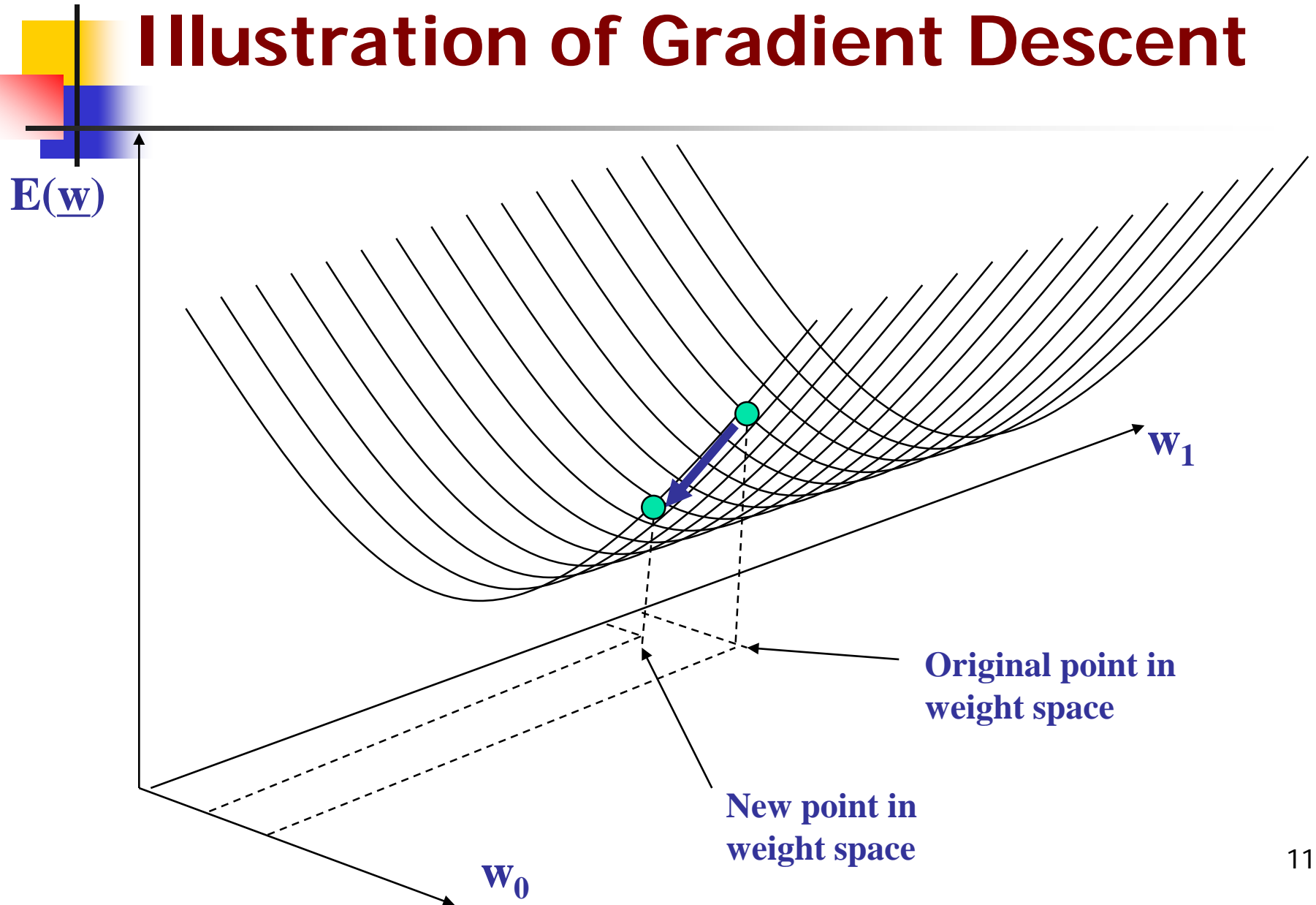


Illustration of Gradient Descent



The Gradient Descent Rule

- The weight update can be derived as follows:

$$\begin{aligned}\partial E / \partial w &= \partial \left(\frac{1}{2} \sum_e (y_e - o_e)^2 \right) / \partial w_i \\&= \frac{1}{2} \sum_e \partial (y_e - o_e)^2 / \partial w_i \\&= \frac{1}{2} \sum_e 2(y_e - o_e) \partial (y_e - o_e) / \partial w_i \\&= \sum_e (y_e - o_e) \partial (y_e - w_i x_{ie}) / \partial w_i \\&= \sum_e (y_e - o_e) (-x_{ie})\end{aligned}$$

where x_{ie} denotes the i -th component of the example e .

The gradient descent training rule becomes:

$$w_i = w_i + \eta \sum_e (y_e - o_e) x_{ie}$$

Gradient Descent Learning Algorithm

- *Initialization: Examples $\{(x_e, y_e)\}_{e=1}^N$, initial weights w_i set to small random values, learning rate parameter η*
- *Repeat*

for each training example (x_e, y_e)

- calculate the network output: $o_e = \sum_{i=0}^d w_i x_{ie}$

- if the Perceptron does not respond correctly, compute weight corrections:

$$\Delta w_i = \Delta w_i + \eta(y_e - o_e)x_{ie}$$

*update the weights with the **accumulated error** from all examples*

$$w_i = w_i + \Delta w_i$$

Gradient
Descent Rule

until termination condition is satisfied.

Example

- Suppose an example of Perceptron which accepts two inputs x_1 and x_2 , with weights $w_1 = 0.5$ and $w_2 = 0.3$ and $w_0 = -1$, learning rate = 1.
- Let the example is given: $x_1 = 2$, $x_2 = 1$, $y = 0$
The network output of the Perceptron is :

$$o = 2 * 0.5 + 1 * 0.3 - 1 = 0.3$$

- The weight updates according to the **gradient descent algorithm** will be:

$$\Delta w_1 = (0 - 0.3) * 2 = -0.6$$

$$\Delta w_2 = (0 - 0.3) * 1 = -0.3$$

$$\Delta w_0 = (0 - 0.3) * 1 = -0.3$$

Example

- Let another example is given: $x_1 = 1, x_2 = 2, y = 1$
- The network output of the Perceptron is :

$$o = 1 * 0.5 + 2 * 0.3 - 1 = 0.1$$

The weight updates according to the gradient descent algorithm will be:

$$\Delta W_1 = - 0.6 + (1 - 0.1) * 1 = 0.3$$

$$\Delta W_2 = - 0.3 + (1 - 0.1) * 2 = 1.5$$

$$\Delta W_0 = - 0.3 + (1 - 0.1) * 1 = 0.6$$

If there are no more examples, the weights will be modified as follows:

$$W_1 = 0.5 + 0.3 = 0.8$$

$$W_2 = 0.3 + 1.5 = 1.8$$

$$W_0 = - 1 + 0.6 = - 0.4$$

Incremental gradient descent

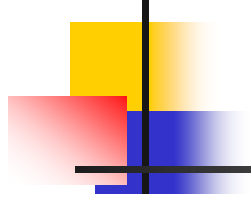


The gradient descent rule faces two difficulties in practice:

- it converges very slowly
- if there are multiple local minima in the error surface, then there is no guarantee that it will find the global minimum

- That is why, a **stochastic version** called *incremental gradient descent* rule is developed to overcome these difficulties. *Whereas the gradient descent rule updates the weights after calculating the whole error accumulated from all examples, the incremental version approximates the gradient descent error decrease by updating the weights after each training example.*

Incremental gradient descent



- Incremental gradient descent is implemented

$$w_i = w_i + \eta(y_e - o_e)x_{ie} \quad \text{where} \quad o_e = \sum_{i=0}^d w_i x_{ie}$$

Incremental Gradient Descent Learning Algorithm

Initialization: *Examples $\{(x_e, y_e)\}_{e=1}^N$, initial weights w_i set to small random values, learning rate parameter η*

Repeat

for each training example (x_e, y_e)

- calculate the network output:

$$o_e = \sum_{i=0}^d w_i x_{ie}$$

- if the Perceptron does not respond correctly update the weights:

$$w_i = w_i + \eta (y_e - o_e) x_{ie}$$

until termination condition is satisfied.

Sigmoidal Perceptrons

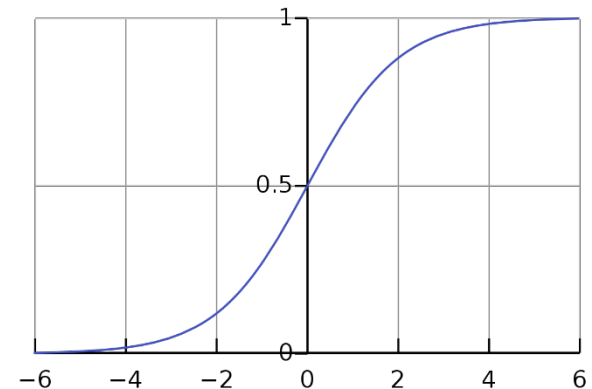
- The simple single-layer Perceptrons with threshold or linear activation functions are not generalizable to more powerful learning mechanisms like multilayer neural networks.
 - That is why, single-layer Perceptrons with sigmoidal activation functions are developed.

- The sigmoidal Perceptron produces output:

$$o = \sigma(S) = \frac{1}{1 + e^{-S}},$$

where:

$$S = \sum_{i=0}^d w_i x_x$$



Training Sigmoidal Perceptrons

- The gradient descent rule for training sigmoidal Perceptrons is again:

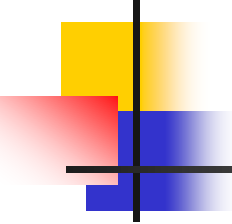
$$w_i = w_i - \eta \partial E / \partial w_i$$

- The difference is in the error derivative $\partial E / \partial w_i$ which due to the use of the sigmoidal function $\sigma(s)$ becomes:

$$\begin{aligned}\partial E / \partial w_i &= \partial \left(\left(\frac{1}{2} \right) \sum_e (y_e - o_e)^2 \right) / \partial w_i \\&= \left(\frac{1}{2} \right) \sum_e \partial (y_e - o_e)^2 / \partial w_i \\&= \left(\frac{1}{2} \right) \sum_e 2(y_e - o_e) \partial (y_e - o_e) / \partial w_i \\&= \sum_e (y_e - o_e) \partial (y_e - \sigma(s)) / \partial w_i \\&= \sum_e (y_e - o_e) \sigma'(s) (-x_{ie})\end{aligned}$$

where x_{ie} denotes the i -th component of the example

Training Sigmoidal Perceptrons

- 
- The *Gradient descent training rule* for training sigmoidal Perceptrons is:

$$w_i = w_i + \eta \sum_e (y_e - o_e) \sigma'(S) x_{ie}$$

where:

$$\sigma'(S) = \sigma(S)(1 - \sigma(S))$$

Gradient Descent Learning Algorithm for Sigmoidal Perceptrons

- *Initialization: Examples $\{(x_e, y_e)\}_{e=1}^N$, initial weights w_i set to small random values, learning rate parameter η*
- *Repeat*
 - for each training example (x_e, y_e)*
 - calculate the network output: $o = \sigma(s)$ where $s = \sum_{i=0}^d w_i x_{ie}$*
 - if the Perceptron does not respond correctly compute weight corrections:*
$$\Delta w_i = \Delta w_i + \eta(y_e - o_e)\sigma(s)(1 - \sigma(s))x_{ie}$$
 - update the weights with the accumulated error from all examples $w_i = w_i + \Delta w_i$*
- until termination condition is satisfied.*

Example

- Suppose an example of Perceptron which accepts two inputs x_1 and x_2 , with weights $w_1 = 0.5$ and $w_2 = 0.3$ and $w_0 = -1$, learning rate = 1.
- Let the following example is given: $x_1 = 2$, $x_2 = 1$, $y = 0$
The output of the Perceptron is :

$$O = \sigma(-1 + 2 * 0.5 + 1 * 0.3) = \sigma(0.3) = 0.5744$$

- The weight updates according to the gradient descent algorithm will be:

$$\Delta w_0 = (0 - 0.5744) * 0.5744 * (1 - 0.5744) * 1 = -0.1404$$

$$\Delta w_1 = (0 - 0.5744) * 0.5744 * (1 - 0.5744) * 2 = -0.2808$$

$$\Delta w_2 = (0 - 0.5744) * 0.5744 * (1 - 0.5744) * 1 = -0.1404$$

Example

Let another example is given: $x_1 = 1$, $x_2 = 2$, $y = 1$

The output of the Perceptron is :

$$O = \sigma(-1 + 1 * 0.5 + 2 * 0.3) = \sigma(0.1) = 0.525$$

The weight updates according to the gradient descent algorithm will be:

$$\Delta w_0 = -0.1404 + (1 - 0.525) * 0.525 * (1 - 0.525) * 1 = -0.0219$$

$$\Delta w_1 = -0.2808 + (1 - 0.525) * 0.525 * (1 - 0.525) * 1 = -0.1623$$

$$\Delta w_2 = -0.1404 + (1 - 0.525) * 0.525 * (1 - 0.525) * 2 = 0.0966$$

If there are no more examples in the batch, the weights will be modified as follows:

$$w_0 = -1 + (-0.0219) = -1.0219$$

$$w_1 = 0.5 + (-0.1623) = 0.3966$$

$$w_2 = 0.3 + 0.0966 = 0.3966$$

Incremental Gradient Descent Learning

Algorithm for *Sigmoidal Perceptrons*

Initialization:

Examples $\{(x_e, y_e)\}$, initial weights w_i set to small random values, learning rate parameter η

Repeat

for each training example (x_e, y_e)

- *calculate the network output: $o = \sigma(s)$*

where $s = \sum_{i=0}^d w_i x_{ie}$

- *If the Perceptron does not respond correctly update the weights: $w_i = w_i + \eta(y_e - o_e)\sigma(s)(1 - \sigma(s))x_{ie}$*

until termination condition is satisfied.

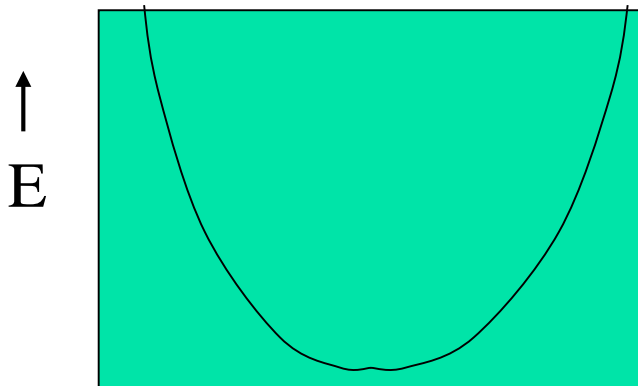
Perceptron vs. Gradient Descent

- Gradient descent finds the decision boundary which minimizes the *sum squared error* of the (target - net) value rather than the (target - output) value
 - Perceptron rule will find the decision boundary which minimizes the classification error – *if the problem is linearly separable*
 - Gradient descent decision boundary may leave more instances misclassified as compared to the perceptron rule: *could have a higher misclassification rate than with the perceptron rule*
- Perceptron rule (target - thresholded output) guaranteed to converge to a separating hyperplane if the problem is linearly separable.

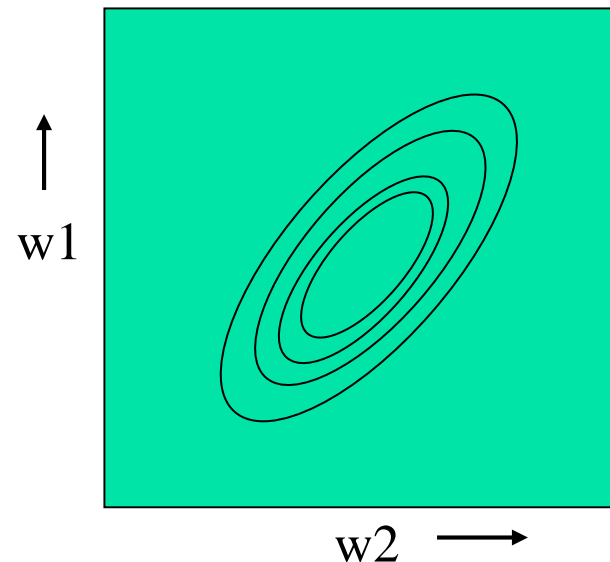
The error surface

- The error surface lies in a space with a horizontal axis for each weight and one vertical axis for the error.
 - For a linear neuron, it is a quadratic bowl.
 - Vertical cross-sections are parabolas.
 - Horizontal cross-sections are ellipses.

Vertical cross-section

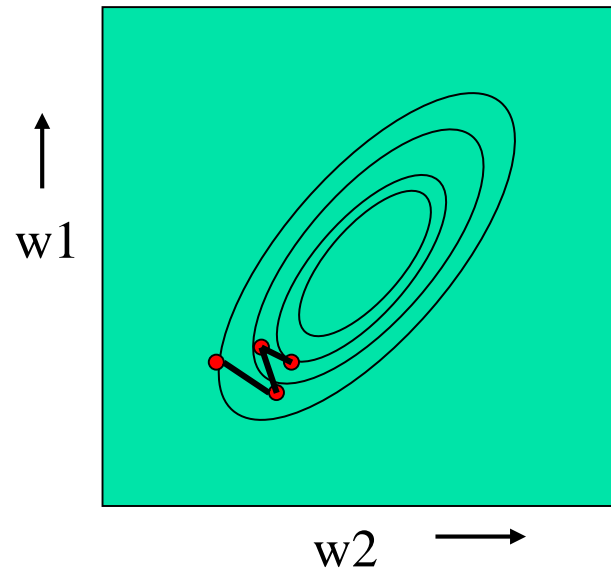
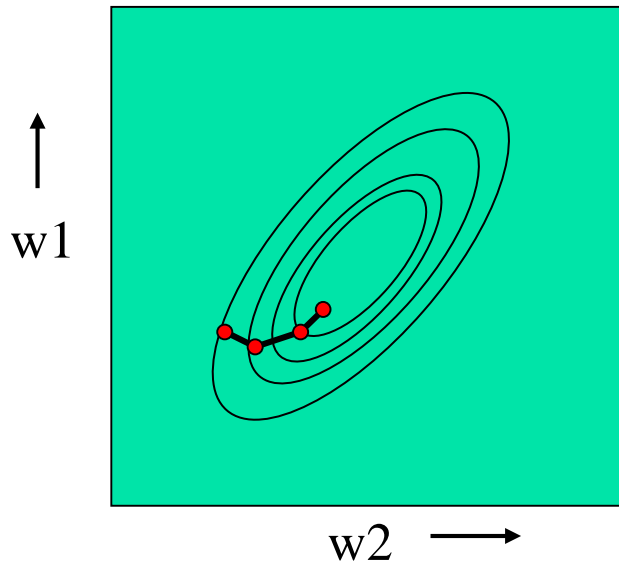


Horizontal cross-section



Batch vs incremental learning

- Batch learning does steepest descent on the error surface
- Incremental learning zig-zags around the direction of steepest descent





Summary

◆ Perceptron training:

- uses thresholded unit
- converges after a finite number of iterations
- output hypothesis classifies training data perfectly
- linearly separability necessary

◆ Gradient descent:

- uses unthresholded linear unit
- converges asymptotically toward a minimum error hypothesis
- termination is not guaranteed
- linear separability not necessary



The fall of the Perceptron

- Researchers begun to discover the Perceptron's limitations.
- Unless input categories were “linearly separable”, a perceptron could not learn to discriminate between them.
- Unfortunately, it appeared that many important categories were not linearly separable.
- E.g., those inputs to an XOR gate that give an output of 1 (namely 10 & 01) are not linearly separable from those that do not (00 & 11).



THANK YOU



VISIT US

WWW.XJTLU.EDU.CN



FOLLOW US

@XJTLU



Xi'an Jiaotong-Liverpool University

西交利物浦大學