University of Nottingham
UK | CHINA | MALAYSIA

COMP3055
Machine Learning

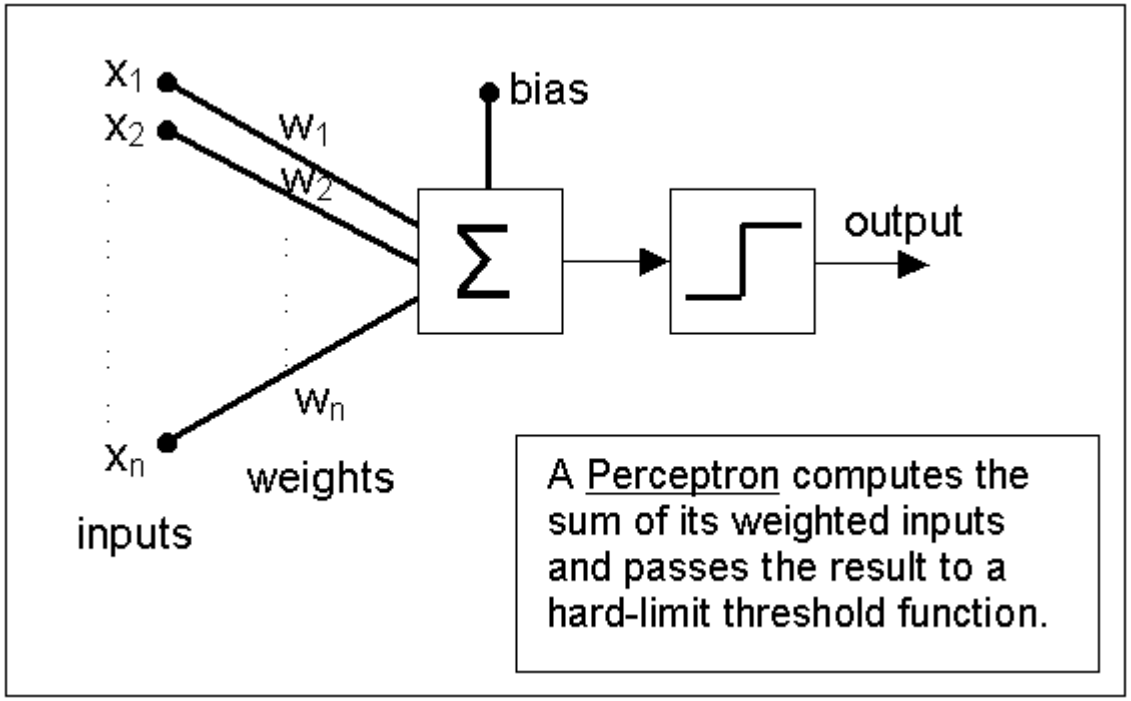
Topic 3 – Perceptron, ADLINE, and Delta Rule

Dr. Zheng LU

2018 Autumn

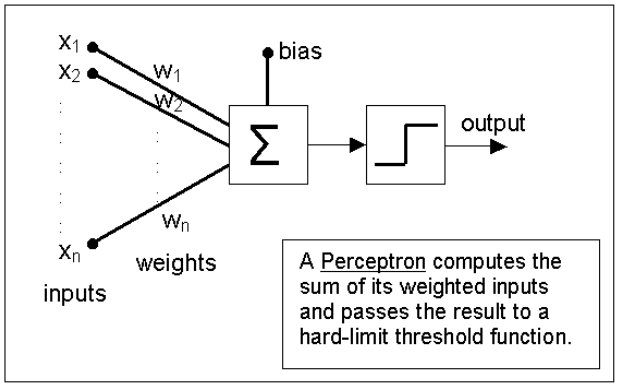
Perceptron - Basic

Perceptron is a type of Artificial Neural Network (ANN)



Perceptron - Operation

Perceptron takes a vector of real-valued inputs, calculates a *linear combination* of these inputs. Then it outputs 1 if the result is greater than some threshold and -1 otherwise.



$$R = w_0 + w_1 x_1 + w_2 x_2 + \cdots + w_n x_n = w_0 + \sum_{i=1}^{n} w_i x_i$$

$$output = o = sign(R) = \begin{cases} +1, & if\ R > 0 \\ -1, & otherwise \end{cases}$$

Perceptron – Decision Surface

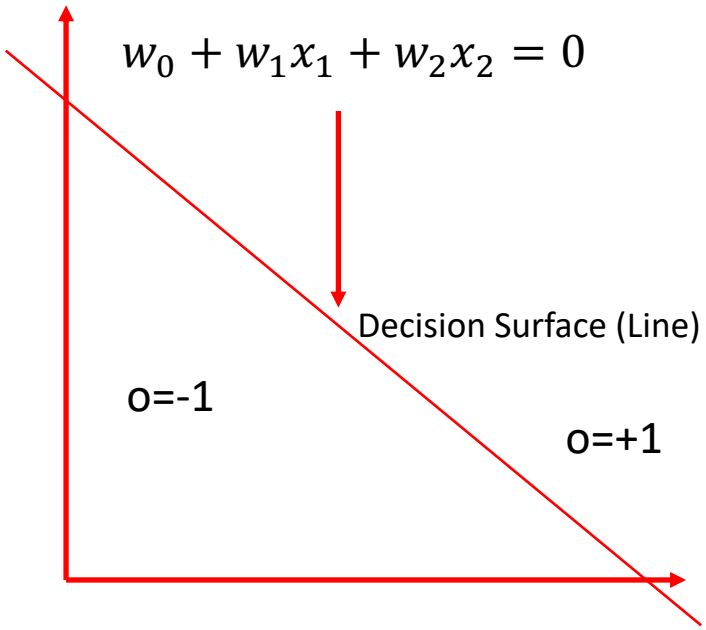
- Perceptron can be regarded as representing a hyperplane decision surface in the n-dimensional **feature space** of instances.

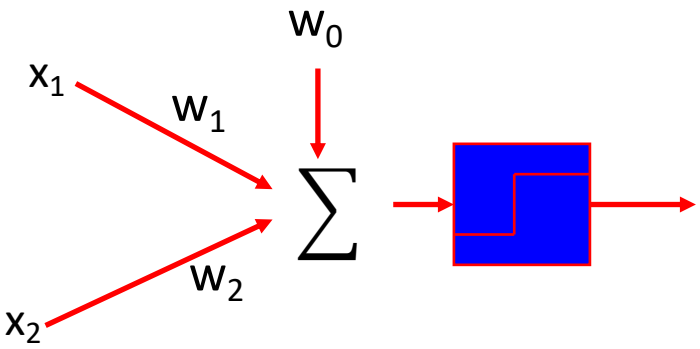
- The perceptron outputs a 1 for instances lying on one side of the hyperplane and a -1 for instances lying on the other side.

- This hyperplane is called the **Decision Surface**.

Perceptron – Decision Surface

In 2-dimensional space:

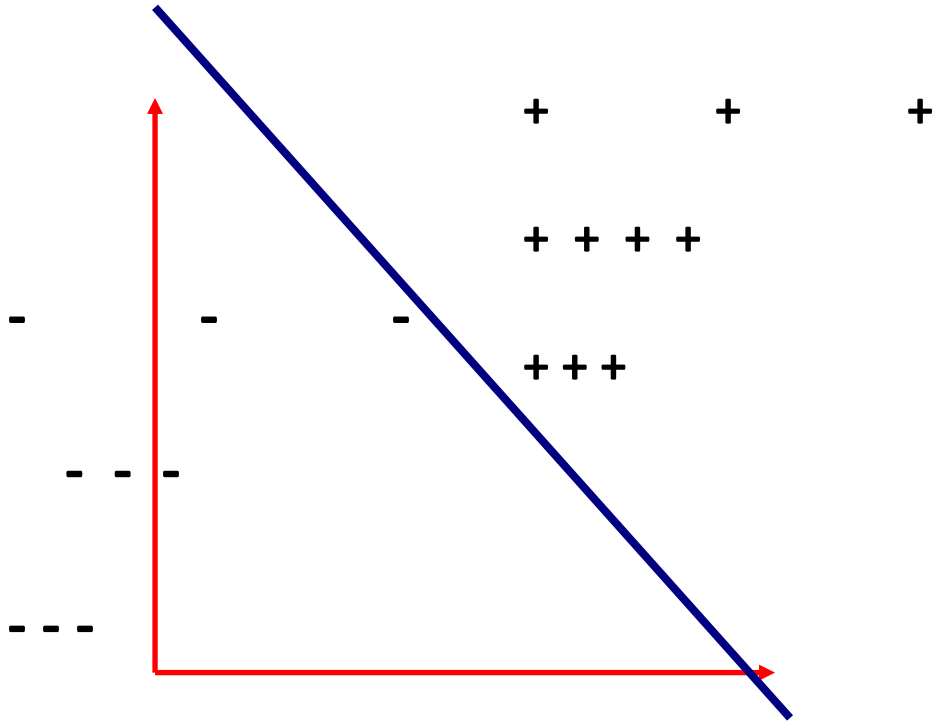




Perceptron – Representation Power

- The Decision Surface is linear.

- Perceptron can only solve **Linearly Separable Problems**.

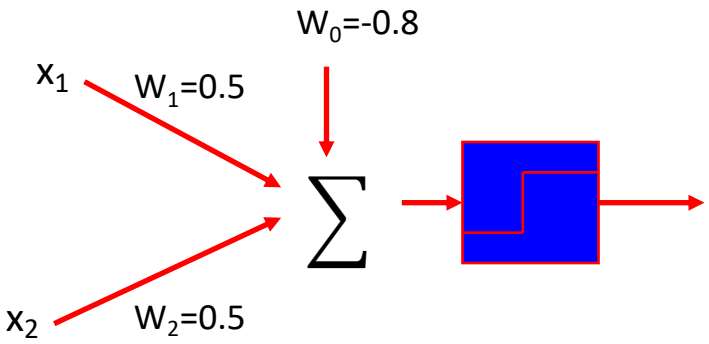


Perceptron – Representation Power

Can represent many boolean functions, assuming boolean values of 1 (true) and -1 (false).

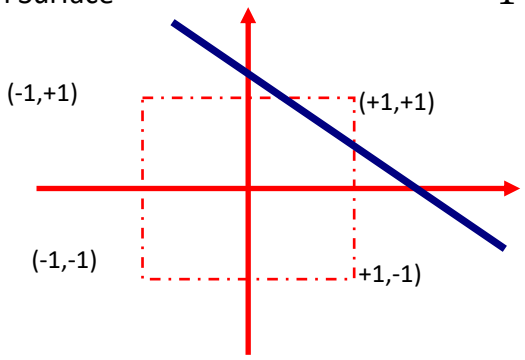
AND

x1	x2	D
-1	-1	-1
-1	+1	-1
+1	-1	-1
+1	+1	+1



Decision Surface

$$-0.8 + 0.5x_1 + 0.5x_2 = 0$$

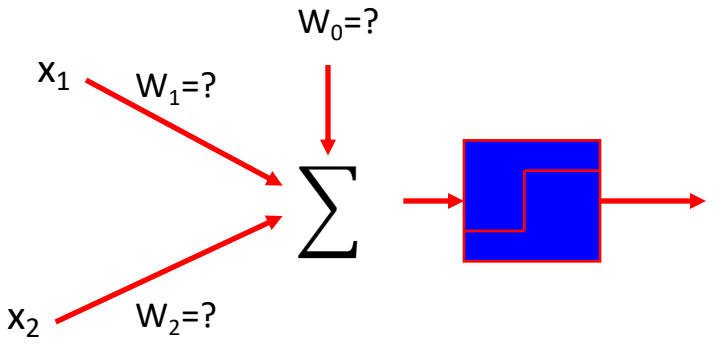


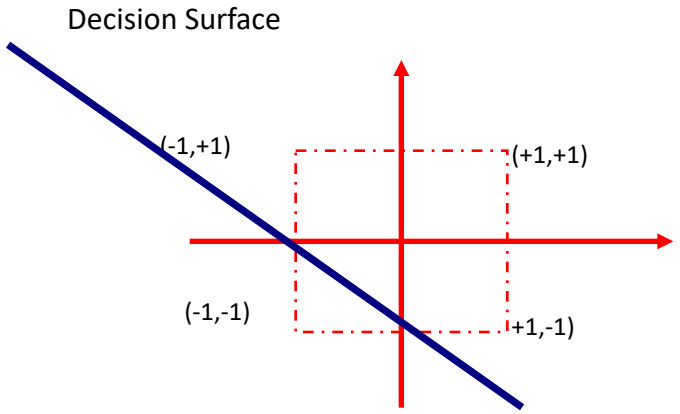
Perceptron – Representation Power

Can represent many boolean functions, assuming boolean values of 1 (true) and -1 (false).

OR

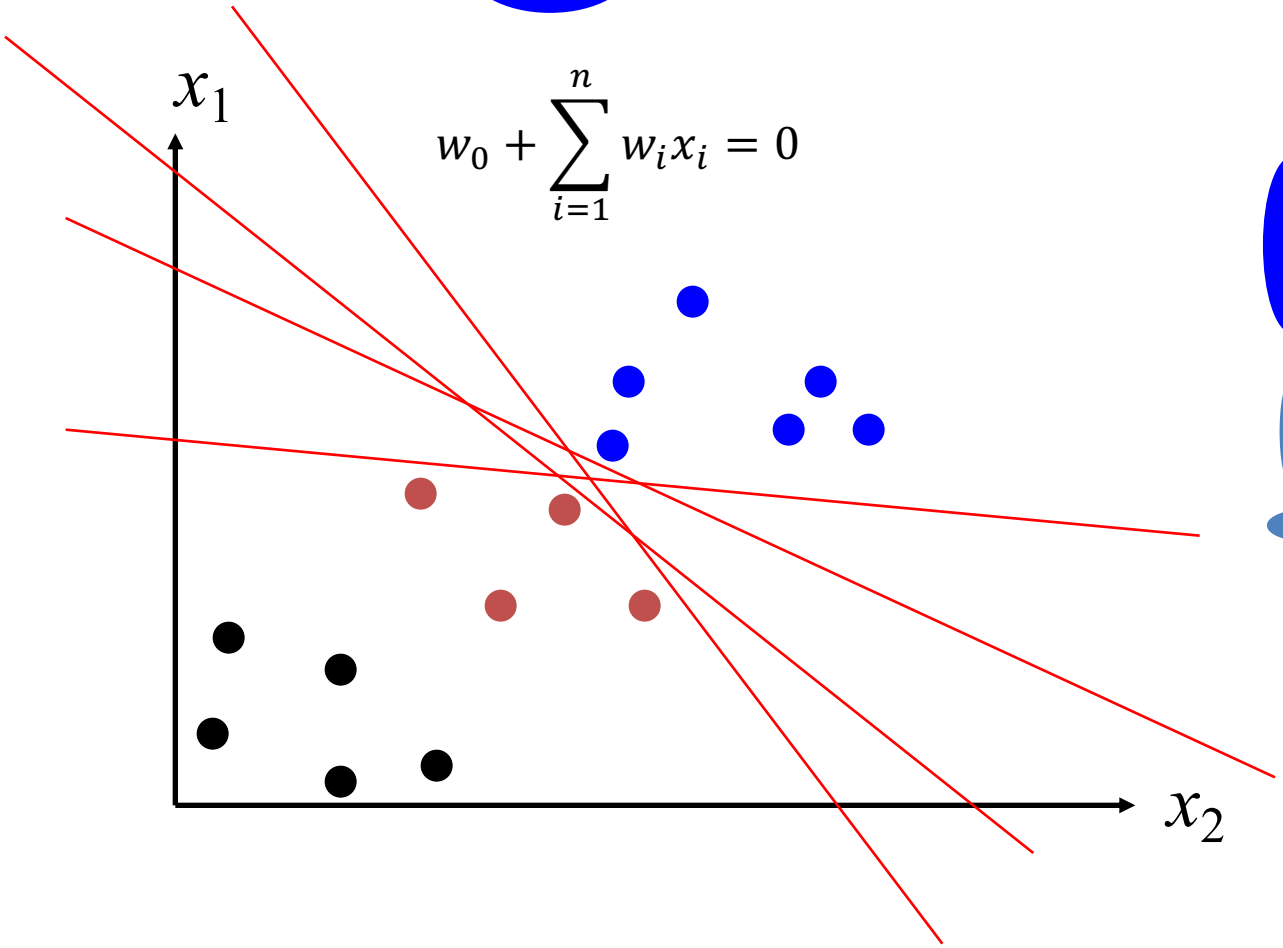
x1	x2	D
-1	-1	-1
-1	+1	+1
+1	-1	+1
+1	+1	+1

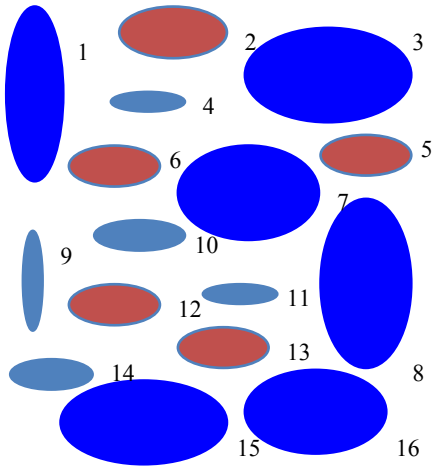




Perceptron – Representation Power

Separate the  objects from the rest.





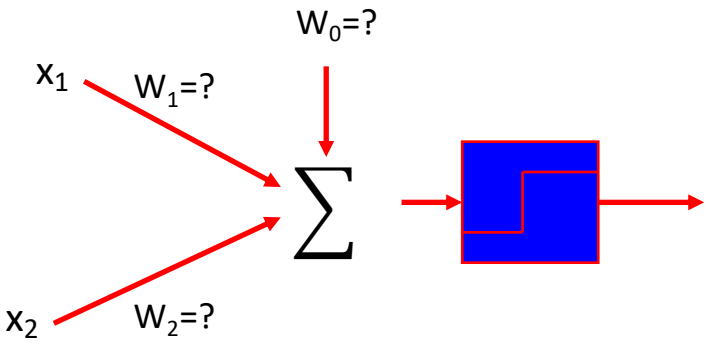
Elliptical blobs (objects)

Perceptron – Representation Power

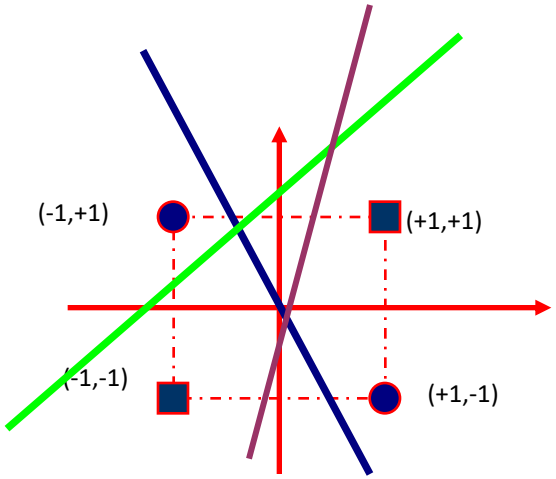
Some problems are linearly non-separable.

XOR

x1	x2	D
-1	-1	-1
-1	+1	+1
+1	-1	+1
+1	+1	-1



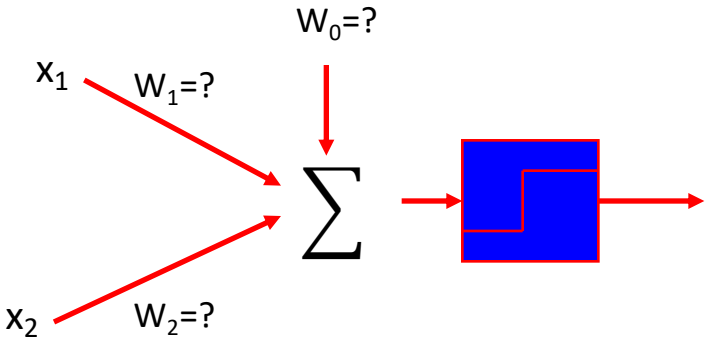
Decision Surface:
It doesn't matter where you place the line (decision surface). It is impossible to separate the space such that on one side we have o = 1 and on the other we have o = -1.

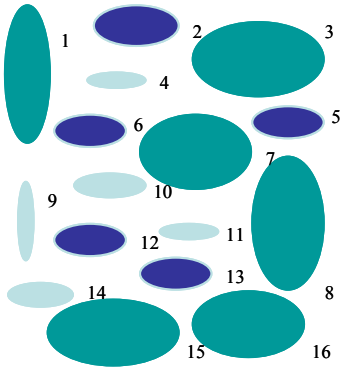


Perceptron Cannot Solve such Problem!

Perceptron – Training Algorithm

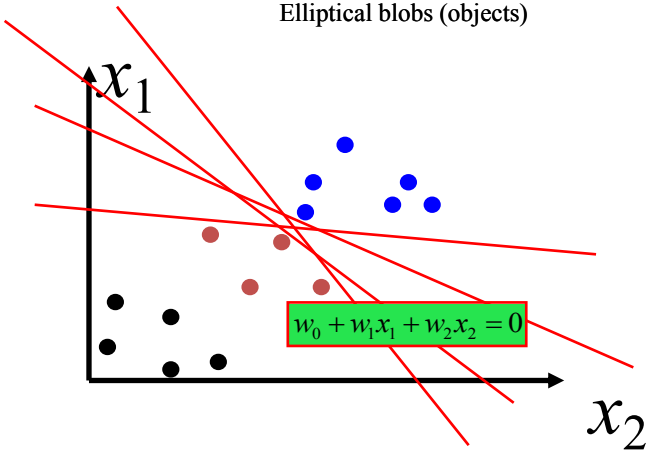
Separate the  objects from the rest.





Elliptical blobs (objects)

We are given the training sample (experience) pairs (X, D). How can we determine the weights that will produce the correct +1 and -1 outputs for the given training samples?



Perceptron – Training Algorithm

Training sample pairs (X, d), where X is the input vector, d is the input vector's classification (+1 or -1) is iteratively presented to the network for training, *one at a time*, until the process converges.

Perceptron – Training Algorithm

The Procedure is as follows:

1. Set the weights to small random values, e.g., in the range (-1, 1)
2. Present X, and calculate

$$R = w_0 + \sum_{i=1}^{n} w_i x_i \quad , \qquad o = sign(R) = \begin{cases} +1, & if\ R > 0 \\ -1, & otherwise \end{cases}$$

3. Update the weights

$$w_i \leftarrow w_i + \eta(d - o)x_i, i = 1,2, \dots, n$$

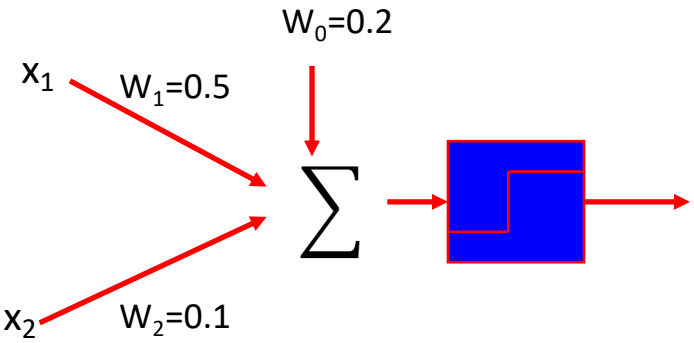
$$0 < \eta < 1, is\ the\ training\ rate, x_0 = 1\ (constant)$$

4. Repeat by going to step 2

Perceptron – Training Algorithm

Example

x1	x2	D
-1	-1	-1
-1	+1	+1
+1	-1	+1
+1	+1	+1



$$w_i \leftarrow w_i + \eta(d - o)x_i, i = 1,2, \dots, n$$

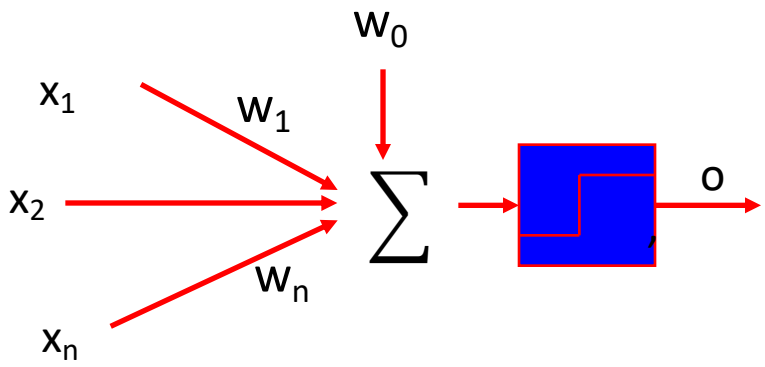
Perceptron – Training Algorithm

Convergence Theorem

The perceptron training rule will converge (finding a weight vector correctly classifies all training samples) within a finite number of iterations, **provided the training examples are linearly separable** and provided a sufficiently small η is used.

Adaptive Linear Element (ADLINE)

Perceptron

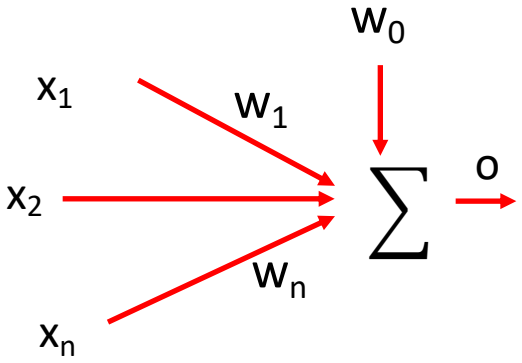


$$R = w_0 + \sum_{i=1}^{n} w_i x_i$$

$$o = sign(R) = \begin{cases} +1, & if\ R > 0 \\ -1, & otherwise \end{cases}$$

VS

ADLINE



$$o = w_0 + \sum_{i=1}^{n} w_i x_i$$

ADLINE VS Perceptron

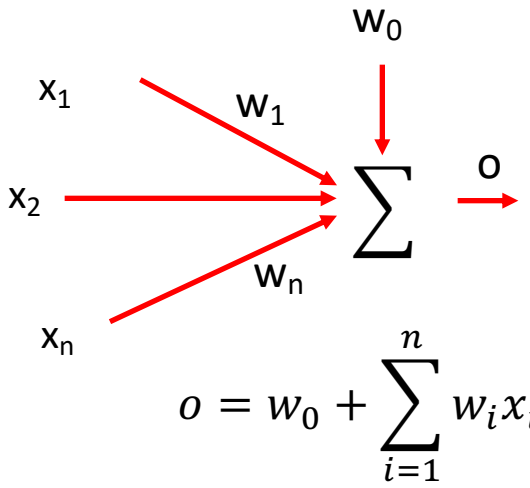
- When the problem is not linearly separable, perceptron will fail to converge.
- ADLINE can overcome this difficulty by finding a best fit approximation to the target.

The ADLINE Error Function

- We have training pairs $(X(k), d(k), k = 1, 2, \ldots, K)$, where K is the number of training samples, the **training error** specifies the difference between the output of the ALDLINE and the desired target.

- The error is defined as

$$E(W) \equiv \frac{1}{2}\sum_{k=1}^{K}(d(k) - o(k))^2$$



$$o = w_0 + \sum_{i=1}^{n} w_i x_i$$

$o(k) = W^T X(k)$ is the output of presenting the training input $X(k)$

The ADLINE Error Function

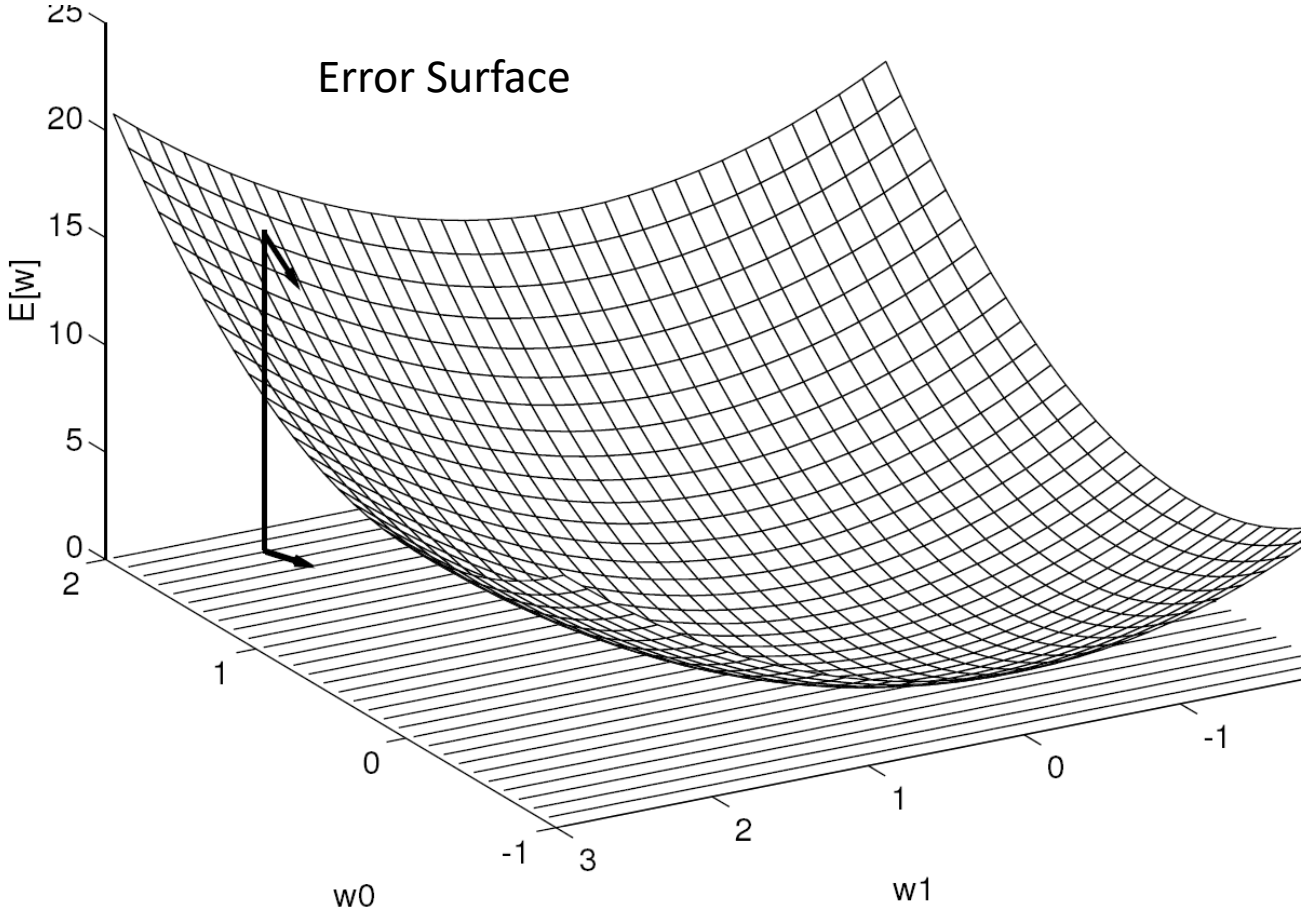
- The error is defined as

$$E(W) \equiv \frac{1}{2}\sum_{k=1}^{K}(d(k) - o(k))^2$$

- The smaller $E(W)$ is, the closer is the approximation.

- We need to find W, based on the given training set, that minimizes the error $E(W)$.

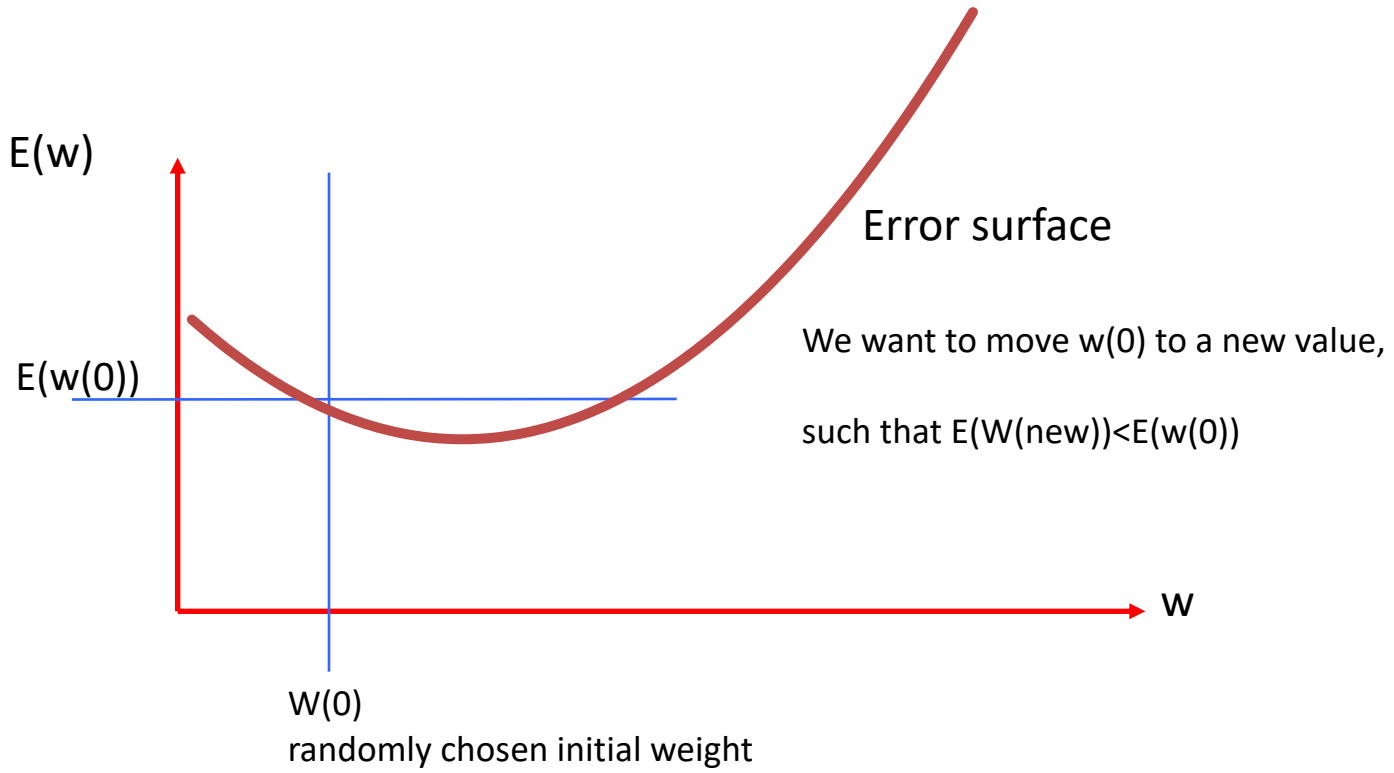
The ADLINE Error Function



The Gradient Descent Rule

An intuition

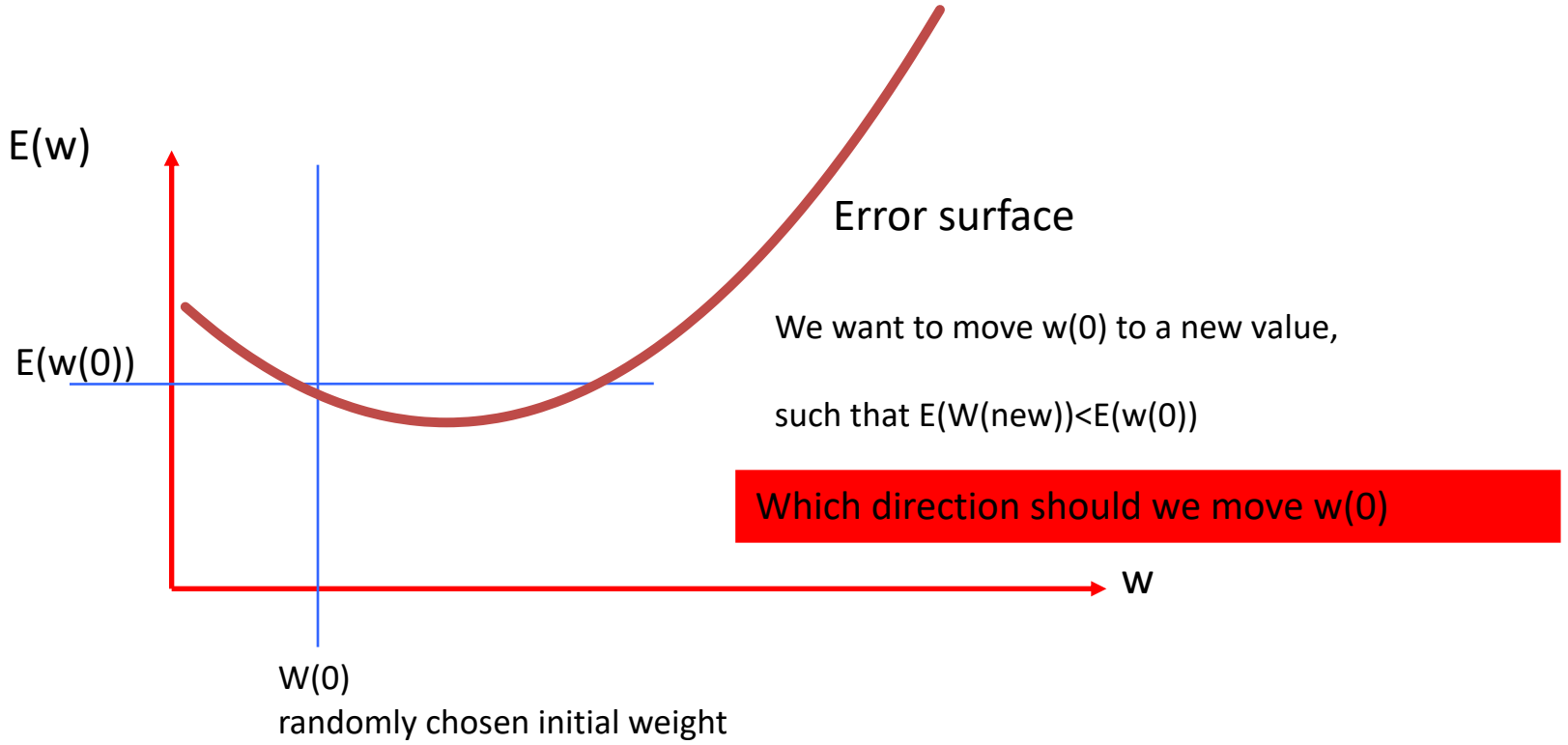
Before we formally derive the gradient decent rule, here is an intuition of what we should be doing.



The Gradient Descent Rule

An intuition

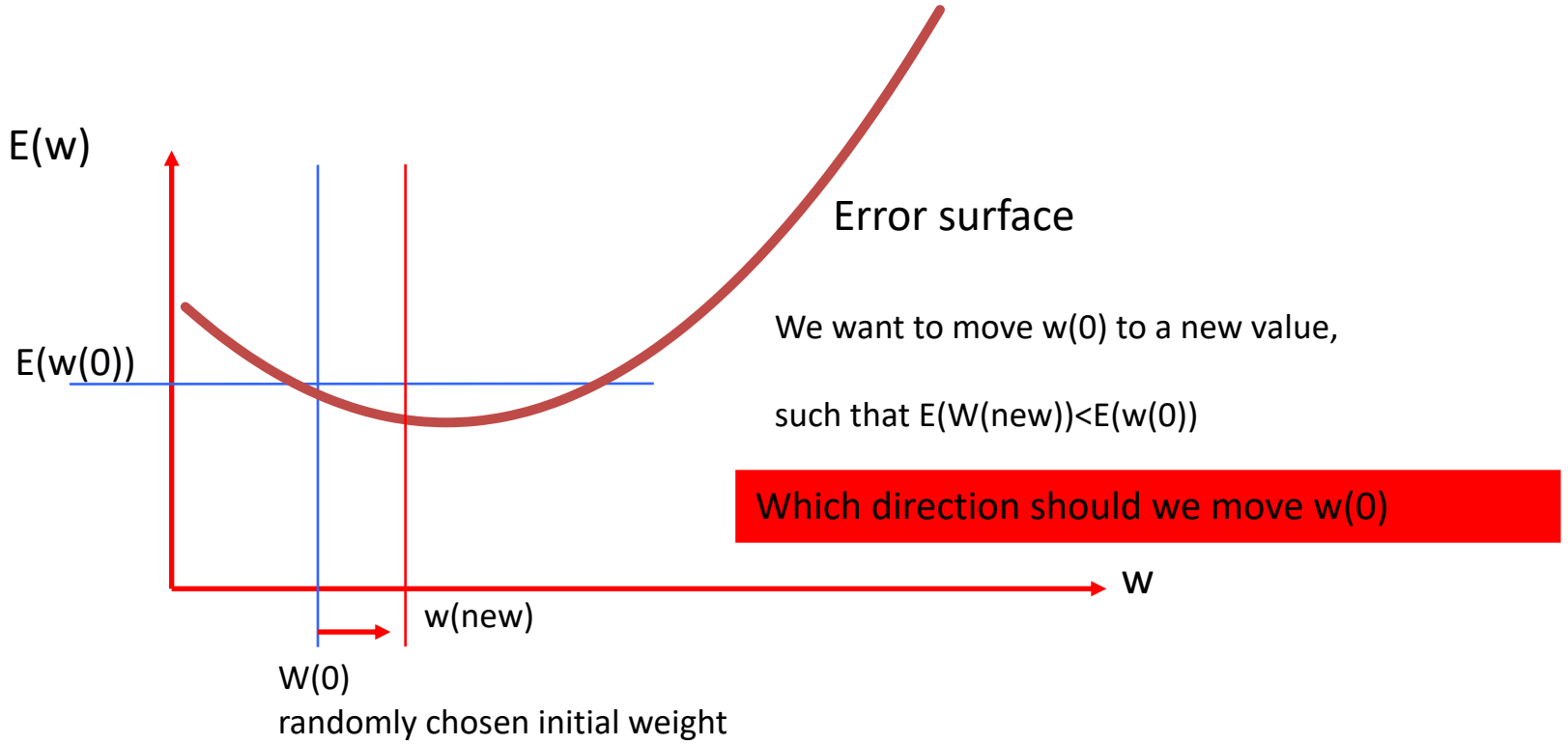
Before we formally derive the gradient decent rule, here is an intuition of what we should be doing.



The Gradient Descent Rule

An intuition

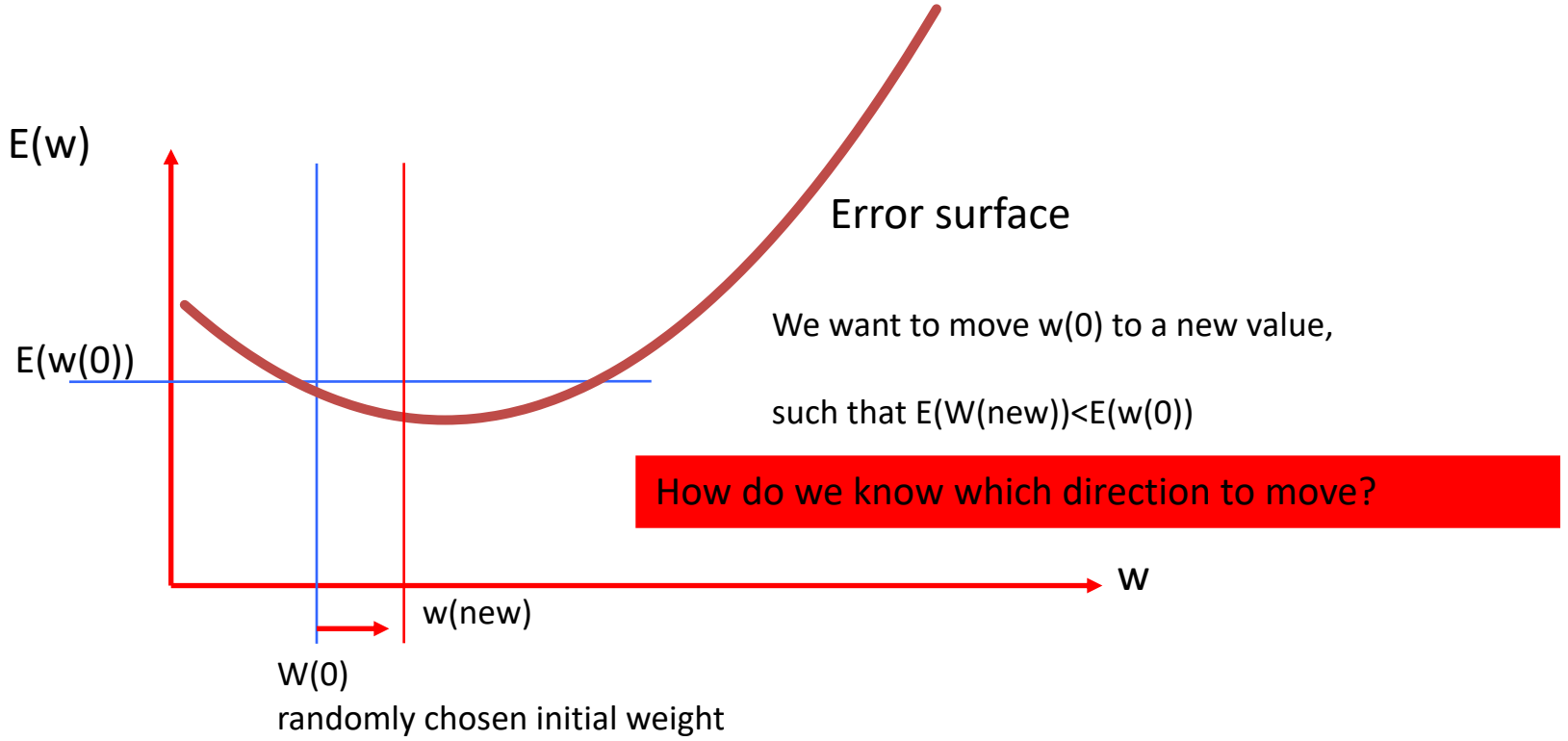
Before we formally derive the gradient decent rule, here is an intuition of what we should be doing.



The Gradient Descent Rule

An intuition

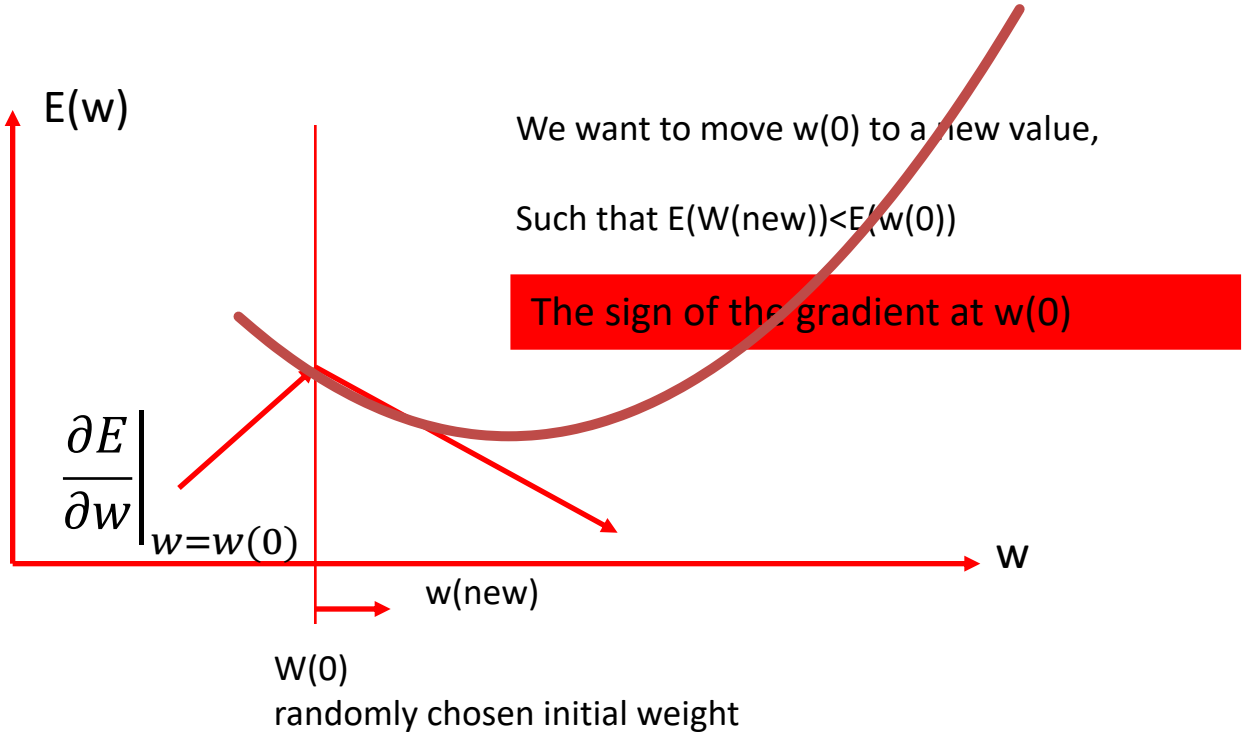
Before we formally derive the gradient decent rule, here is an intuition of what we should be doing.



The Gradient Descent Rule

An intuition

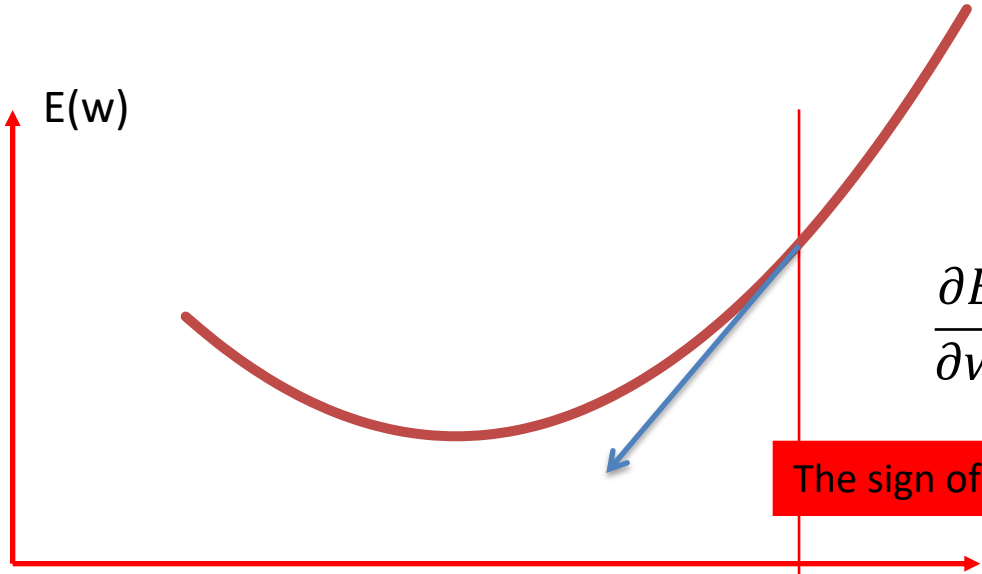
Before we formally derive the gradient decent rule, here is an intuition of what we should be doing.



The Gradient Descent Rule

An intuition

Before we formally derive the gradient decent rule, here is an intuition of what we should be doing.



We want to move w(0) to a new value,

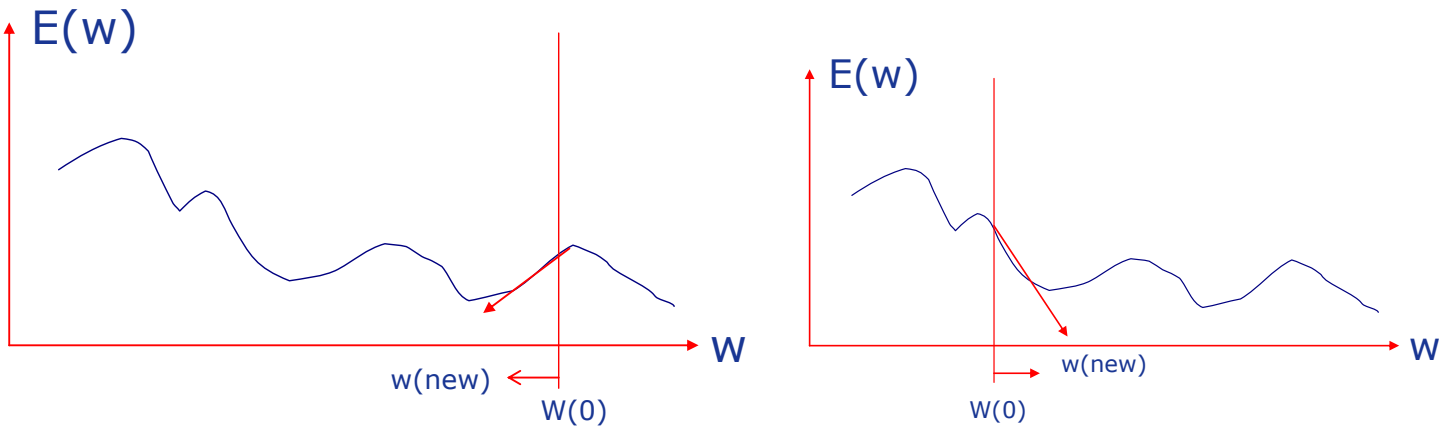
Such that E(W(new))<E(w(0))

The Gradient Descent Rule

An intuition

The intuition leads to the following rule:

$$w(new) \leftarrow w(old) - \Delta sign\left(\left.\frac{\partial E}{\partial w}\right|_{w=w(old)}\right)$$



The Gradient Descent Rule

Formal Derivation of Gradient Descent

$$\nabla E(W) = \left[\frac{\partial E}{\partial w_1}, \frac{\partial E}{\partial w_2}, \ldots, \frac{\partial E}{\partial w_n}\right]$$

- The gradient of E is a vector, whose components are the partial derivatives of E with respect to each of the w_i's
- The gradient specifies the direction that produces the steepest increase in E.
- Negative of the vector gives the direction of steepest decrease in E.

The Gradient Descent Rule

The gradient training rule is

$$W \leftarrow W - \eta \nabla E(W)$$

$$w_i \leftarrow w_i - \eta \frac{\partial E}{\partial w_i}$$

η is the training rate

The Gradient Descent Rule

Gradient of ADLINE Error Functions

$$E(W) \equiv \frac{1}{2}\sum_{k=1}^{K}(d(k)-o(k))^2$$

$$\frac{\partial E}{\partial w_i} = \frac{\partial E}{\partial w_i}\left(\frac{1}{2}\sum_{k=1}^{K}(d(k)-o(k))^2\right)$$

$$= \frac{1}{2}\sum_{k=1}^{K}\left(\frac{\partial E}{\partial w_i}(d(k)-o(k))^2\right)$$

$$= \frac{1}{2}\sum_{k=1}^{K}\left(2(d(k)-o(k))\frac{\partial E}{\partial w_i}(d(k)-o(k))\right)$$

$$= \sum_{k=1}^{K}\left((d(k)-o(k))\frac{\partial E}{\partial w_i}(d(k)-w_0-\sum_{i=1}^{n}w_i x_i(k))\right)$$

$$= \sum_{k=1}^{K}((d(k)-o(k))(-x_i(k)))$$

$$= -\sum_{k=1}^{K}\big(d(k)-o(k)\big)x_i(k)$$

The Gradient Descent Rule

ADLINE weight updating using **gradient descent rule**

$$w_i \leftarrow w_i + \eta \sum_{k=1}^{K} (d(k) - o(k)) x_i(k)$$

The Gradient Descent Rule

Gradient descent training procedure

- Initialise w_i to small vales, e.g., in the range of $(-1, 1)$, choose a learning rate, e.g., $\eta = 0.2$

- Until the termination condition is met, Do
  - For all training sample pair $(X(k), d(k))$, input the instance $X(k)$ and compute

$$\delta_i = -\sum_{k=1}^{K} (d(k) - o(k)) x_i(k)$$

  - For each weight w_i, Do

$$w_i \leftarrow w_i - \eta \delta_i$$

Batch Mode:

gradients accumulated over **ALL** samples first

Then update the weights

Stochastic (Incremental) Gradient Descent

Also called online mode, Least Mean Square (LMS), Widrow-Hoff, and Delta Rule

- Initialise w_i to small vales, e.g., in the range of *(-1, 1)*, choose a learning rate, e.g., η = 0.01 (should be smaller than batch mode)

- Until the termination condition is met, Do
  - For EACH training sample pair *(X(k), d(k))*, compute

$$\delta_i = -(d(k) - o(k))x_i(k)$$

  - For each weight w_i, Do

$$w_i \leftarrow w_i - \eta\delta_i$$

Online Mode:

Calculate gradient for **EACH** samples

Then update the weights

Training Iterations, Epochs

- Training is an iterative process; training samples will have to be used repeatedly for training
- Assuming we have K training samples *[(X(k), d(k)), k=1, 2, ..., K]*; then an epoch is the presentation of all K sample for training once
  - First epoch: Present training samples: *(X(1), d(1)), (X(2), d(2)), ... (X(K), d(K))*
  - Second epoch: Present training samples: *(X(K), d(K)), (X(K-1), d(K-1)), ... (X(1), d(1))*
  - Note the order of the training sample presentation between epochs can (and should normally) be different.
- Normally, training will take many epochs to complete

Termination of Training

To terminate training, there are normally two ways

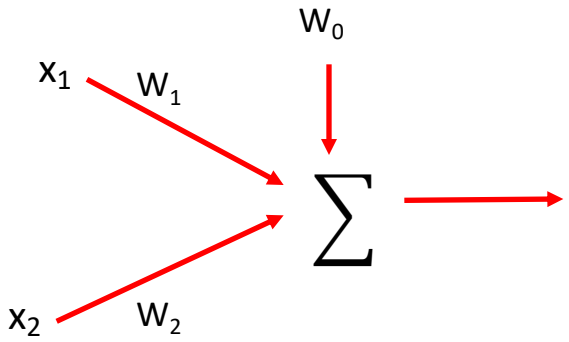
- When a pre-set number of training epochs is reached.
- When the error is smaller than a pre-set value.

$$E(W) \equiv \frac{1}{2}\sum_{k=1}^{K}(d(k) - o(k))^2$$

Gradient Descent Training

Example

x1	x2	D
-1	-1	-1
-1	+1	+1
+1	-1	+1
+1	+1	+1



Initialization

$W_0(0)=0.1$; $W_1(0)=0.2$; $W_2(0)=0.3$;

$\eta=0.5$

Further Reading

Chapter 4, T. M. Mitchell, Machine Learning, McGraw-Hill International Edition, 1997