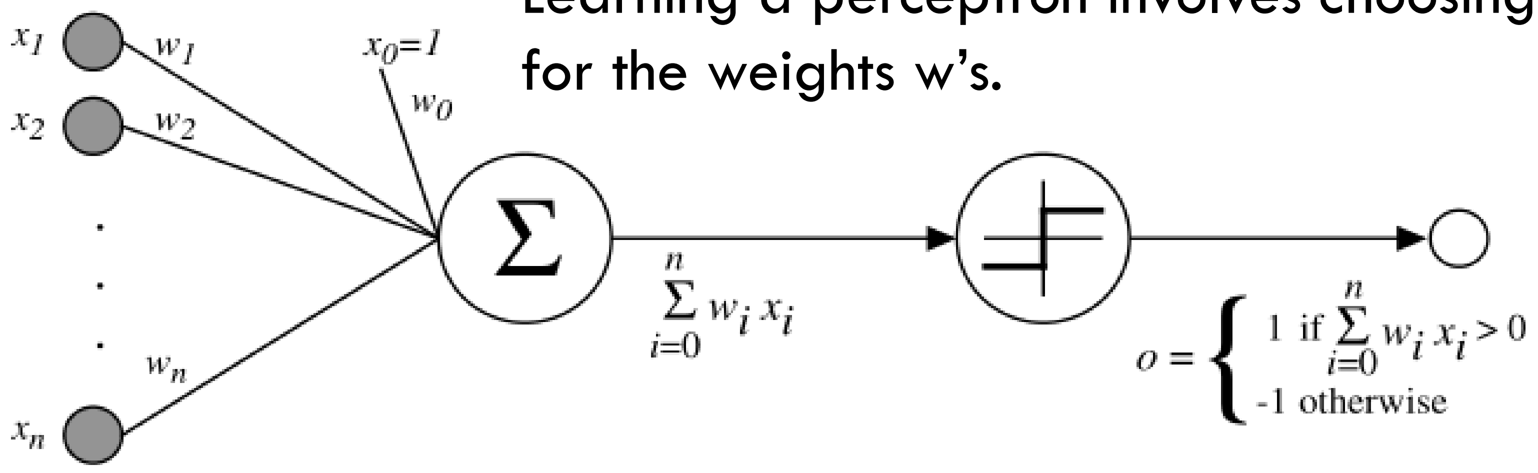
PERCEPTRON, PERCEPTRON TRAINING RULE & GRADIENT DESCENT

PERCEPTRON

Learning a perceptron involves choosing values for the weights w's.



x_1
x_2
x_n
w_1
w_2
w_n
$x_0=1$
w_0
Σ
$\sum_{i=0}^{n} w_i x_i$
$o = \begin{cases} 1 & \text{if } \sum_{i=0}^{n} w_i x_i > 0 \\ -1 & \text{otherwise} \end{cases}$

$$o(x_1, \ldots, x_n) = \begin{cases} 1 & \text{if } w_0 + w_1 x_1 + \cdots + w_n x_n > 0 \\ -1 & \text{otherwise.} \end{cases}$$

Sometimes we'll use simpler vector notation:

$$o(\vec{x}) = \begin{cases} 1 & \text{if } \vec{w} \cdot \vec{x} > 0 \\ -1 & \text{otherwise.} \end{cases}$$

A perceptron can be viewed as representing a hyperplane decision surface in the n-dimensional space of instances (i.e. points)

The equation of the decision surface is **w . x = 0**

The points that can be separated by the hyperplane are called linearly separable sets of points

A single perceptron can be used to represent many Boolean functions

- $w_0 = -0.8$, $w_1 = w_2 = 0.5$
- $w_0 = -0.3$, $w_1 = w_2 = 0.5$

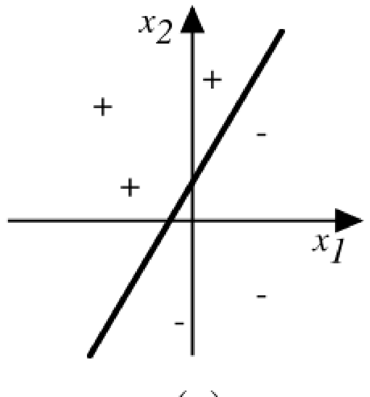
AND & OR are special cases of m-of-n functions, where atleast m of n inputs has to be true for the output to be true

Perceptrons can represent all primitive Boolean functions except XOR (which has an output only if the inputs are not the same)

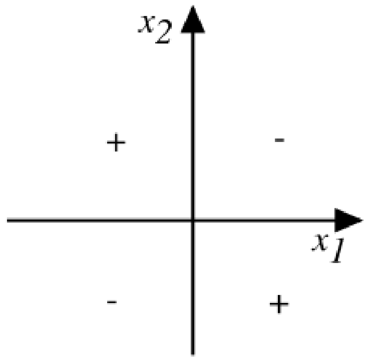
The basic perceptron can then be used to build more complicated Boolean function networks

Two levels deep – any Boolean function

Decision Surface of a Perceptron



(a)



(b)

Represents some useful functions

- What weights represent $g(x_1, x_2) = AND(x_1, x_2)$?

But some functions not representable

- e.g., not linearly separable
- Therefore, we'll want networks of these...

- Learn weights ***w***
- Several algorithms: perceptron rule and the delta rule
- Converge to somewhat different hypothesis under somewhat different conditions
- Provide the basis for learning networks of many units
- General Procedure
  - Begin with random weights
  - Iteratively apply the perceptron to each training example
  - Modify the perceptron weights whenever it misclassifies an example
  - Repeat the procedure iteratively, until the perceptron classifies all examples correctly
- The perceptron training rule is used to modify the weights at each step.
- The role of the learning rate is to moderate the degree to which weights are changed at each step
  - Sometimes made to decay as the number of weight tuning iterations increases

Perceptron training rule

$$w_i \leftarrow w_i + \Delta w_i$$

where

$$\Delta w_i = \eta(t - o)x_i$$

Where:

- $t = c(\vec{x})$ is target value
- o is perceptron output
- η is small constant (e.g., .1) called *learning rate*

- **Why should this update rule converge toward successful weight values?**
- If correctly classified, (t-o) = 0, no change in weight
- Suppose t = +1 while o = -1,
- We need to increase the value of **w . x,** by altering the value of the weights
- If $x_i > 0$, then increasing w_i will bring the perceptron closer to correctly classifying this example
- The training rule will increase w_i in this case because (t – o), η and x_i are all positive.
- For example, if $x_i = 0.8$, $\eta = 0.1$ and t = 1, and o = -1, then the weight update will be $\Delta w_i = \eta(t - o)x_i = 0.1(1-(-1))0.8 = 0.16$
- If t = -1 and o = +1, then weights associated with positive x_i will be decreased, rather than increased

Perceptron training rule

$$w_i \leftarrow w_i + \Delta w_i$$

where

$$\Delta w_i = \eta(t - o)x_i$$

Where:

- $t = c(\vec{x})$ is target value
- o is perceptron output
- η is small constant (e.g., .1) called *learning rate*

Perceptron training rule

Can prove it will converge

- If training data is linearly separable
- and η sufficiently small

- The perceptron fails to converge if the examples are not linearly separable.
- The delta rule works even when the examples are not linearly separable
  - Converges to a best fit approximation to the target concept
- The key idea behind the delta rule is to use gradient descent to search the hypothesis space of possible weight vectors to find the weights that best fit the training examples
- Gradient descent provides the basis for the ***backpropagation algorithm*** – learn any network with many interconnected units

Gradient Descent

To understand, consider simpler *linear unit*, where

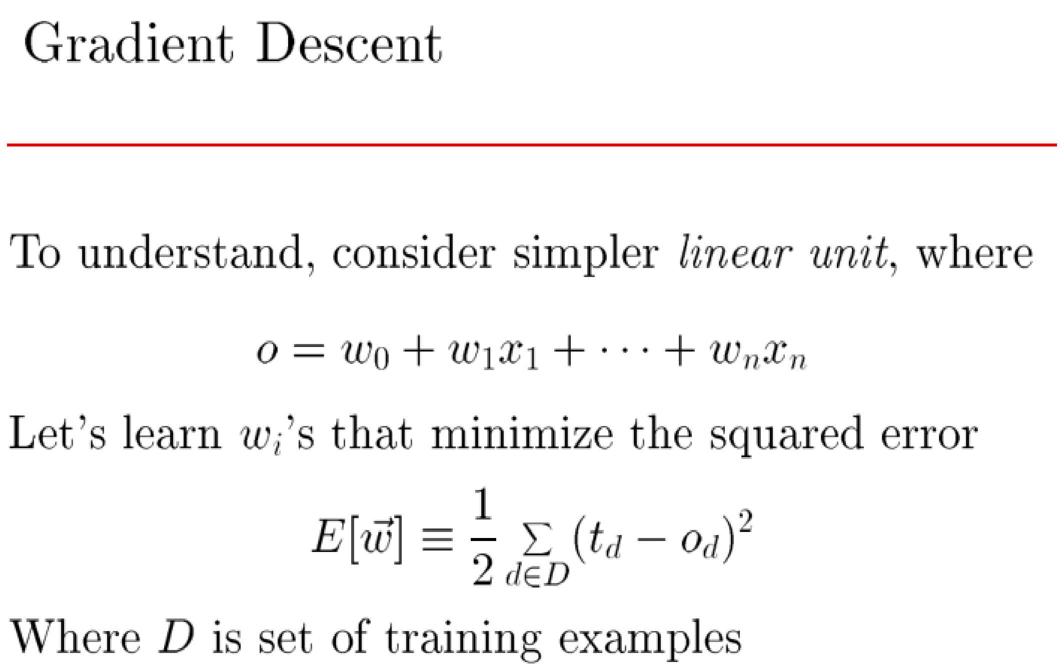
$$o = w_0 + w_1 x_1 + \cdots + w_n x_n$$

Let's learn w_i's that minimize the squared error

$$E[\vec{w}] \equiv \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2$$

Where D is set of training examples

- Linear Unit – a unthresholded perceptron where o(**x**) = **w . x**
- Specify a measure for the training error of a hypothesis (weight vector), relative to the training examples
- **E** is a function of **w,** because the linear unit output o depends on this weight vector
- **E** also depends on the particular set of training examples **D**, but training set is fixed



Gradient Descent

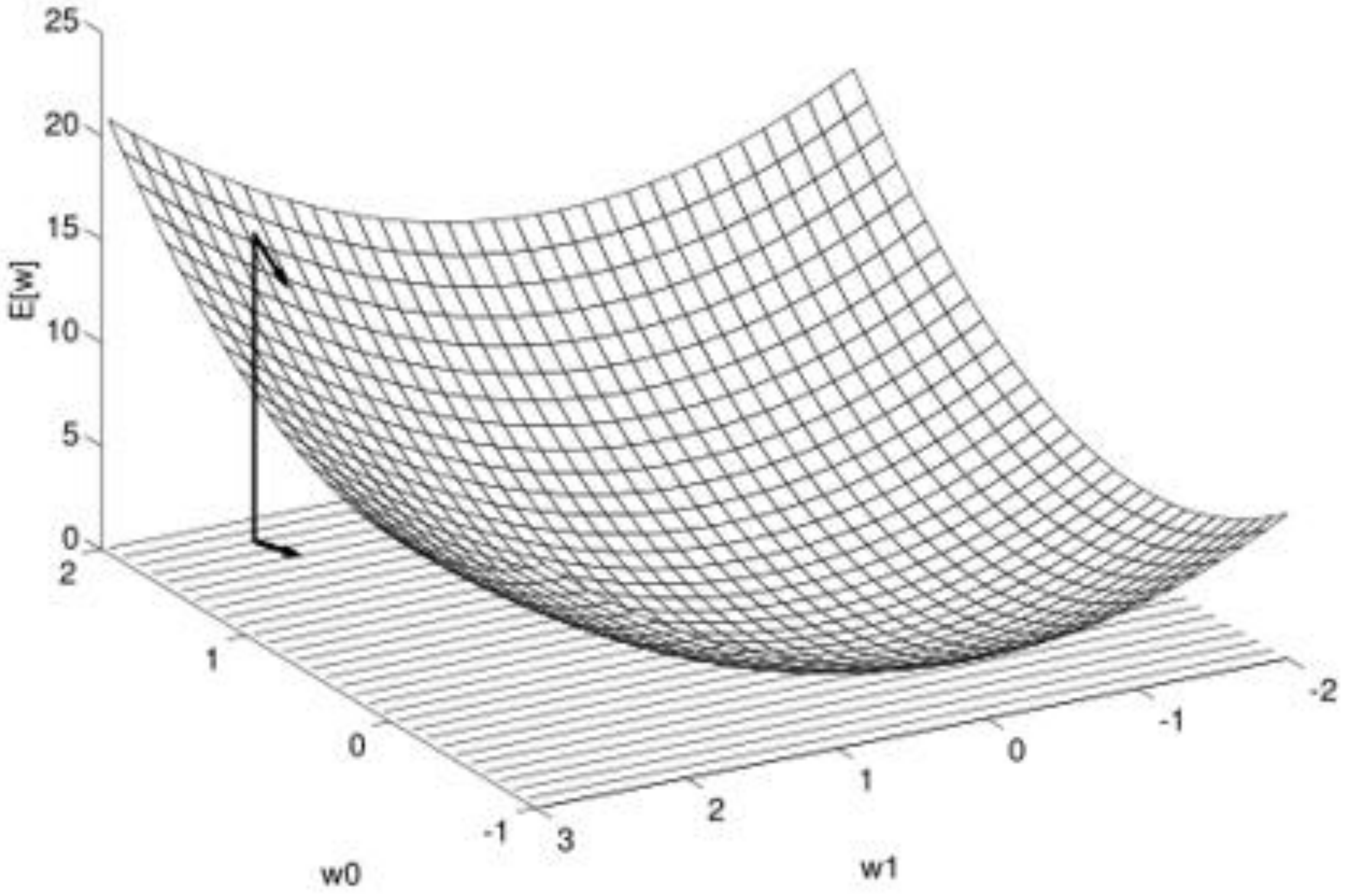
To understand, consider simpler *linear unit*, where

$$o = w_0 + w_1 x_1 + \cdots + w_n x_n$$

Let's learn w_i's that minimize the squared error

$$E[\vec{w}] \equiv \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2$$

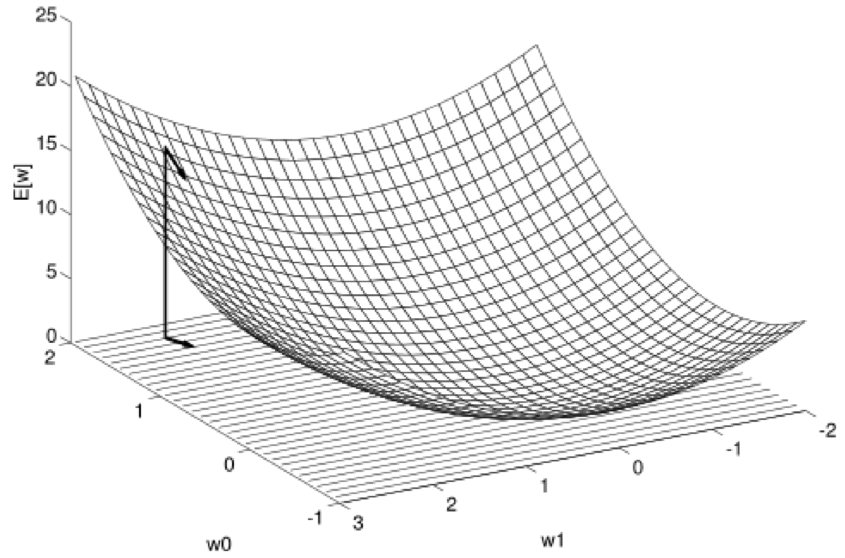
Where D is set of training examples



25
20
15
10
5
0
E[w]
2
1
0
-1
w0
3
2
1
0
-1
-2
w1

Gradient Descent

- Helpful to visualize the entire hypothesis space of **possible weight vectors** and their **associated E values.**
- The axes w_0 and w_1 represent possible values for the two weights of a simple linear unit
- The w_0, w_1 plane represents the entire hypothesis space.
- The vertical axis indicates the error E relative to some fixed set of training examples
- The error surface shown in the figure then summarizes the desirability of every weight vector in the hypothesis space
  - We desire a **hypothesis with the minimum error**



Gradient

$$\nabla E[\vec{w}] \equiv \left[\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \cdots \frac{\partial E}{\partial w_n}\right]$$

Training rule:

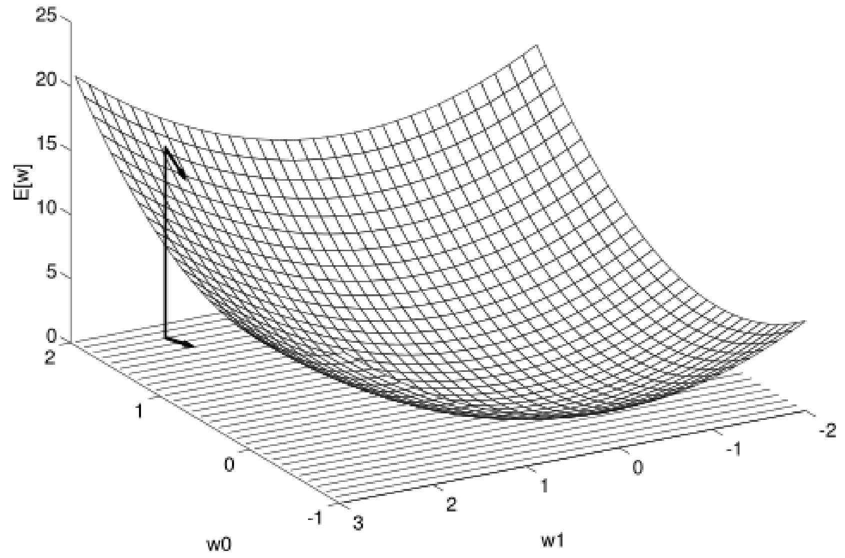
$$\Delta \vec{w} = -\eta \nabla E[\vec{w}]$$

i.e.,

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i}$$

Gradient Descent

- The shape of the error surface depends on the definition of E (the error function)
- **Here, for the definition of E, the error surface will always be parabolic with a single global minimum**
- Specific parabola is defined by the set of training examples



Gradient

$$\nabla E[\vec{w}] \equiv \left[\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \cdots \frac{\partial E}{\partial w_n}\right]$$

Training rule:

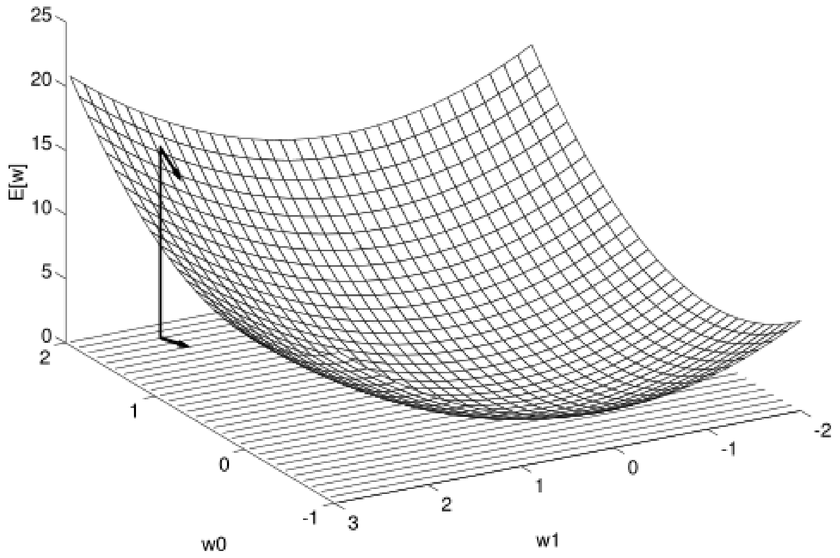
$$\Delta \vec{w} = -\eta \nabla E[\vec{w}]$$

i.e.,

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i}$$

- Gradient descent search determines a weight vector that minimizes E by starting with an arbitrary initial vector and repeatedly modifying it in small steps
- At each step, the weight vector is altered in the direction that produces the steepest descent along the error surface
- **This process is continued until the global minimum error is reached**
- The direction of the **steepest descent is found by computing the derivative of E with respect to each component of the vector w.**
- This vector derivative is called the gradient of E with respect to **w**.
- When interpreted as a vector in weight space, **the gradient specifies the direction that produces the steepest increase in E**.
- The negative of this vector gives the direction of the steepest decrease.

Gradient Descent



Gradient

$$\nabla E[\vec{w}] \equiv \left[\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \cdots \frac{\partial E}{\partial w_n}\right]$$

Training rule:

$$\Delta \vec{w} = -\eta \nabla E[\vec{w}]$$

i.e.,

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i}$$

Gradient Descent

$$\begin{aligned}
\frac{\partial E}{\partial w_i} &= \frac{\partial}{\partial w_i}\frac{1}{2}\sum_d (t_d - o_d)^2 \\
&= \frac{1}{2}\sum_d \frac{\partial}{\partial w_i}(t_d - o_d)^2 \\
&= \frac{1}{2}\sum_d 2(t_d - o_d)\frac{\partial}{\partial w_i}(t_d - o_d) \\
&= \sum_d (t_d - o_d)\frac{\partial}{\partial w_i}(t_d - \vec{w}\cdot\vec{x_d}) \\
\frac{\partial E}{\partial w_i} &= \sum_d (t_d - o_d)(-x_{i,d})
\end{aligned}$$

Gradient Descent

```
GRADIENT-DESCENT($training\_examples, \eta$)

    Each training example is a pair of the form
    $\langle \vec{x}, t \rangle$, where $\vec{x}$ is the vector of input values,
    and $t$ is the target output value. $\eta$ is the
    learning rate (e.g., .05).

  • Initialize each $w_i$ to some small random value
  • Until the termination condition is met, Do
      – Initialize each $\Delta w_i$ to zero.
      – For each $\langle \vec{x}, t \rangle$ in $training\_examples$, Do
          * Input the instance $\vec{x}$ to the unit and
            compute the output $o$
          * For each linear unit weight $w_i$, Do
                $\Delta w_i \leftarrow \Delta w_i + \eta (t - o) x_i$
      – For each linear unit weight $w_i$, Do
                $w_i \leftarrow w_i + \Delta w_i$
```

Summary

Perceptron training rule guaranteed to succeed if

- Training examples are linearly separable
- Sufficiently small learning rate η

Linear unit training rule uses gradient descent

- Guaranteed to converge to hypothesis with minimum squared error
- Given sufficiently small learning rate η
- Even when training data contains noise
- Even when training data not separable by H

LIMITATIONS OF GRADIENT DESCENT

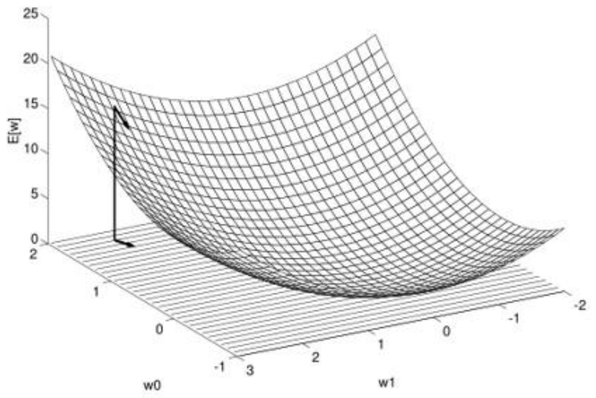
Gradient descent is an important general paradigm for learning

It is a strategy for searching through a large or infinite hypothesis space that can be applied whenever:

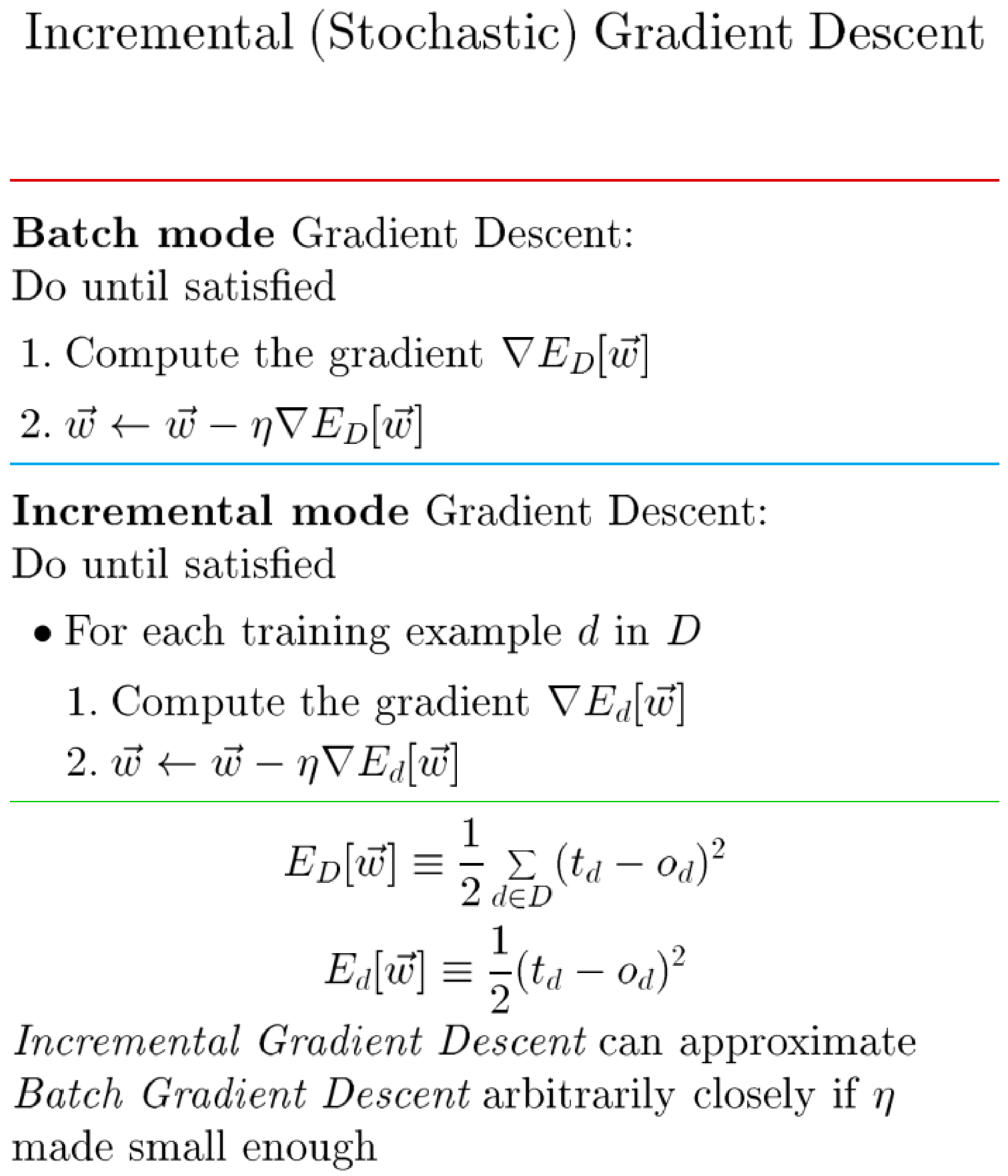
- The hypothesis space contains continuously parameterized hypothesis (e.g. weights in a linear unit)
- The error can be differentiated with respect to these hypothesis parameters

The key practical difficulties in applying gradient descent algorithm are:

- Converging to a local minimum can sometime be very slow (i.e. it can take many thousands of gradient descent steps)
- If there are multiple local minima in the error surface, then there is no guarantee that the procedure will find the global minimum



- The **gradient descent** training rule computes **weight updates** after **summing over all the training examples in D**.
- The idea behind the stochastic gradient descent is to approximate this gradient descent search **by updating weights incrementally following the calculation of error for each individual example**
- One way to view the stochastic gradient descent algorithm is to consider a distinct error function $E_d(\mathbf{w})$ defined for each individual training example d
- By making the value of the learning rate η sufficiently small, stochastic gradient descent can be made to approximate the true gradient descent arbitrarily closely



Incremental (Stochastic) Gradient Descent

Batch mode Gradient Descent:
Do until satisfied

1. Compute the gradient $\nabla E_D[\vec{w}]$
2. $\vec{w} \leftarrow \vec{w} - \eta \nabla E_D[\vec{w}]$

Incremental mode Gradient Descent:
Do until satisfied

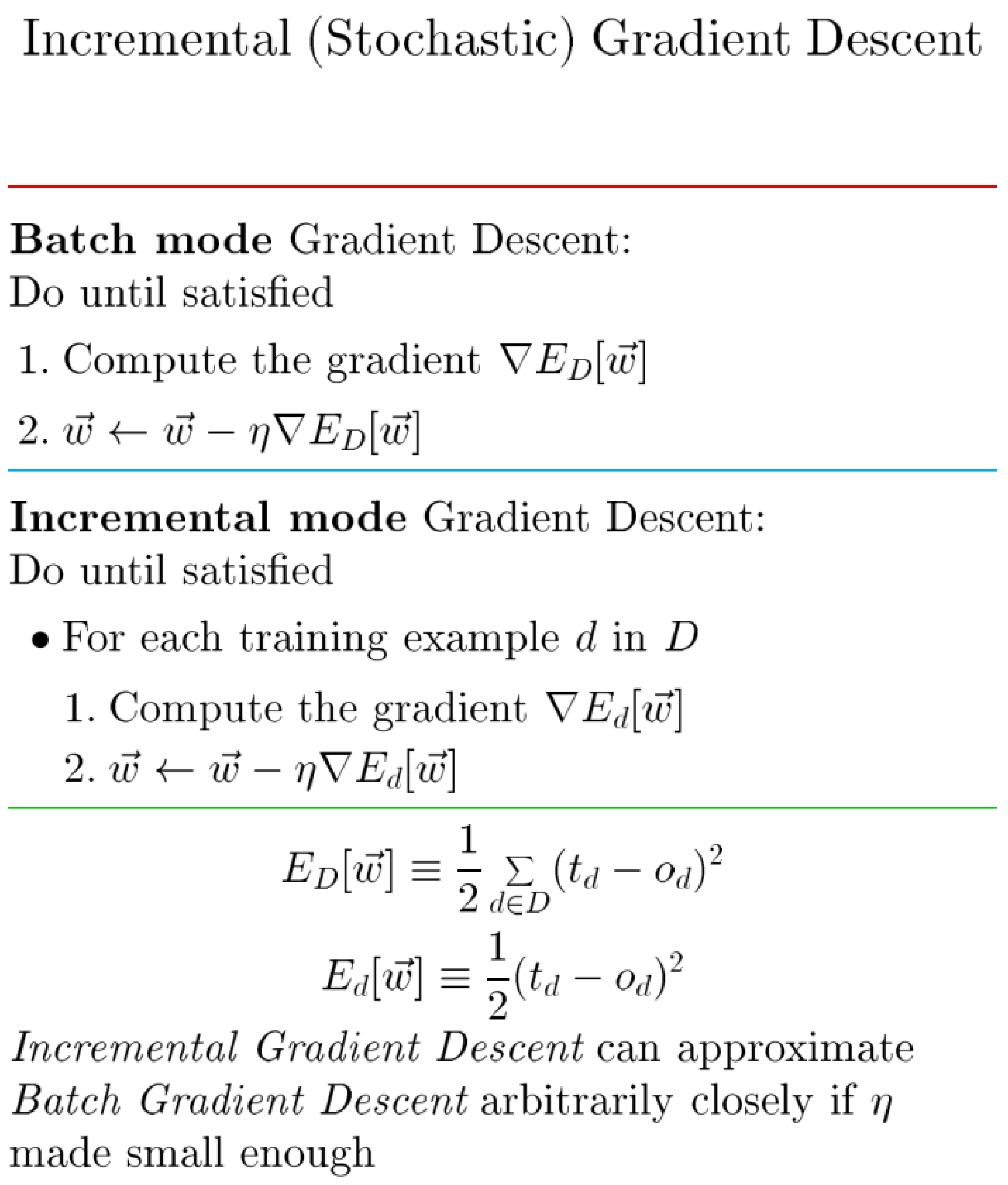
- For each training example d in D
  1. Compute the gradient $\nabla E_d[\vec{w}]$
  2. $\vec{w} \leftarrow \vec{w} - \eta \nabla E_d[\vec{w}]$

$$E_D[\vec{w}] \equiv \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2$$

$$E_d[\vec{w}] \equiv \frac{1}{2} (t_d - o_d)^2$$

Incremental Gradient Descent can approximate *Batch Gradient Descent* arbitrarily closely if η made small enough

- In standard gradient descent, the error is summed over all examples before updating weights
- In stochastic gradient descent, weights are updated upon examining each training example
- Summing over multiple examples in standard gradient descent requires more computation per weight update step.
- However, we can use a larger step size per weight update than stochastic gradient descent
- Multiple local minima – stochastic gradient descent can sometimes avoid falling into these local minima



Incremental (Stochastic) Gradient Descent

Batch mode Gradient Descent:
Do until satisfied

```
1. Compute the gradient $\nabla E_D[\vec{w}]$
2. $\vec{w} \leftarrow \vec{w} - \eta \nabla E_D[\vec{w}]$
```

Incremental mode Gradient Descent:
Do until satisfied

```
• For each training example $d$ in $D$
    1. Compute the gradient $\nabla E_d[\vec{w}]$
    2. $\vec{w} \leftarrow \vec{w} - \eta \nabla E_d[\vec{w}]$
```

$$E_D[\vec{w}] \equiv \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2$$

$$E_d[\vec{w}] \equiv \frac{1}{2} (t_d - o_d)^2$$

Incremental Gradient Descent can approximate *Batch Gradient Descent* arbitrarily closely if η made small enough

THANKS