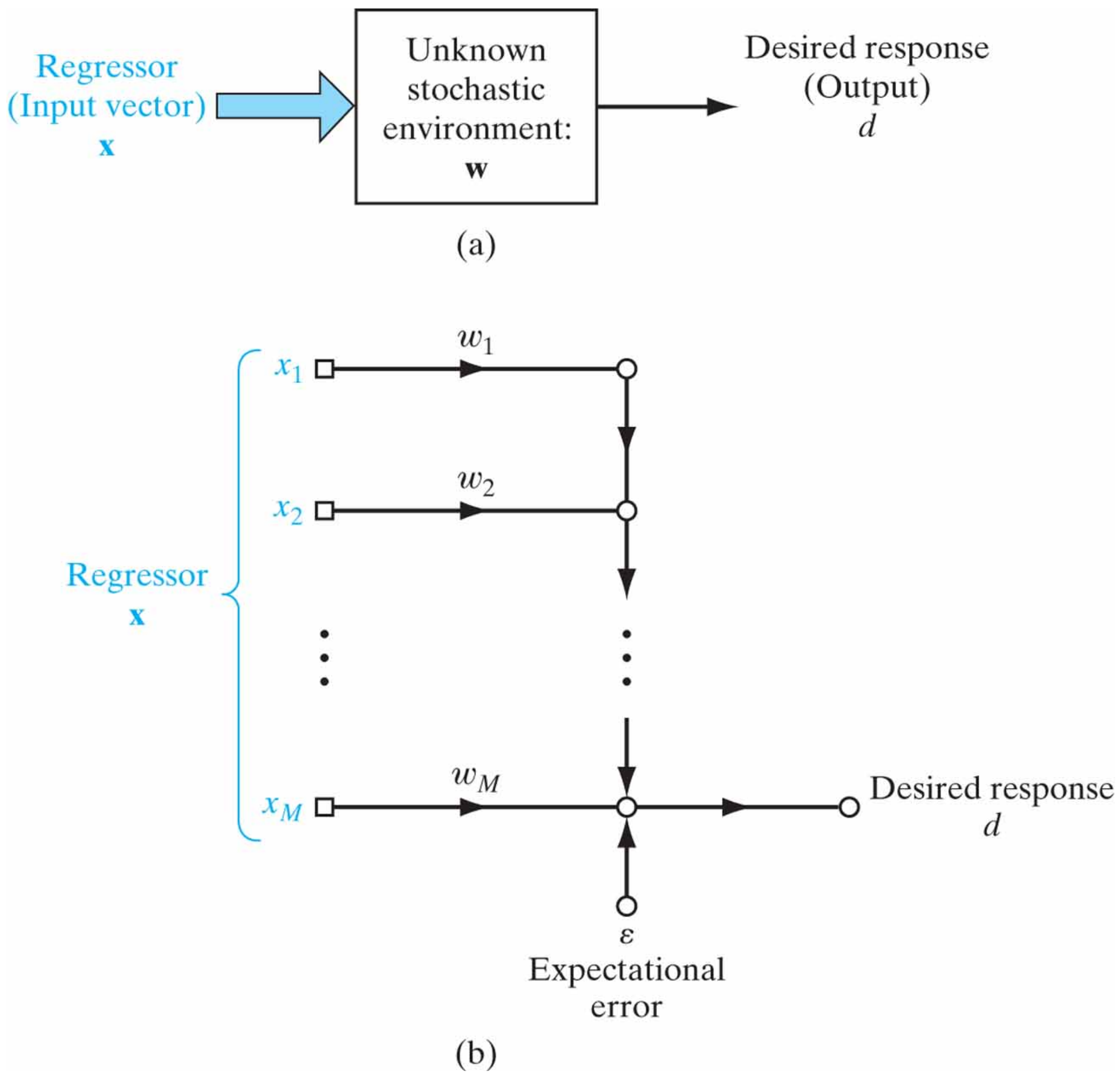
**CSE 5526: Introduction to Neural Networks**

# Regression and the LMS Algorithm

# Problem statement



# Linear regression with one variable

- Given a set of $N$ pairs of data $\{x_i, d_i\}$, approximate $d$ by a linear function of $x$ (regressor)

  i.e.

$$d \approx wx + b$$

  or

$$d_i = y_i + \varepsilon_i = \varphi(wx_i + b) + \varepsilon_i$$
$$= wx_i + b + \varepsilon_i$$

  where the activation function $\varphi(x) = x$ is a linear function, corresponding to a linear neuron. $y$ is the output of the neuron, and

$$\varepsilon_i = d_i - y_i$$

  is called the regression (expectational) error

# Linear regression (cont.)

- The problem of regression with one variable is how to choose *w* and *b* to minimize the regression error

- The least squares method aims to minimize the square error:

$$E = \frac{1}{2}\sum_{i=1}^{N}\varepsilon_i^2 = \frac{1}{2}\sum_{i=1}^{N}(d_i - y_i)^2$$

# Linear regression (cont.)

- To minimize the two-variable square function, set

$$\begin{cases} \dfrac{\partial E}{\partial b} = 0 \\[2ex] \dfrac{\partial E}{\partial w} = 0 \end{cases}$$

# Linear regression (cont.)

$$\frac{\partial E}{\partial b} = \frac{1}{2}\sum_i \frac{\partial}{\partial b}(d_i - wx_i - b)^2$$

$$= -\sum_i (d_i - wx_i - b) = 0$$

$$\frac{\partial E}{\partial w} = \frac{1}{2}\sum_i \frac{\partial}{\partial w}(d_i - wx_i - b)^2$$

$$= -\sum_i (d_i - wx_i - b)x_i = 0$$

# Analytic solution approaches

- Solve one equation for $b$ in terms of $w$
  - Substitute into other equation, solve for $w$
  - Substitute solution for $w$ back into equation for $b$
- Setup system of equations in matrix notation
  - Solve matrix equation
- Rewrite problem in matrix form
  - Compute matrix gradient
  - Solve for $w$

# Linear regression (cont.)

- Hence

$$b = \frac{\left(\sum_i x_i^2\right)\left(\sum_i d_i\right) - \left(\sum_i x_i\right)\left(\sum_i x_i d_i\right)}{N\sum_i (x_i - \bar{x})^2}$$

Derive yourself!

$$w = \frac{\sum_i (x_i - \bar{x})(d_i - \bar{d})}{\sum_i (x_i - \bar{x})^2}$$

where an overbar (i.e. $\bar{x}$) indicates the mean

# Linear regression in matrix notation

- Let $\boldsymbol{X} = [\boldsymbol{x_1}\ \boldsymbol{x_2}\ \boldsymbol{x_3} \ldots \boldsymbol{x_N}]^{\boldsymbol{T}}$
- Then the model predictions are $\boldsymbol{y} = \boldsymbol{X}\boldsymbol{w}$
- And the mean square error can be written $E(\boldsymbol{w}) = \|\boldsymbol{d} - \boldsymbol{y}\|^2 = \|\boldsymbol{d} - \boldsymbol{X}\boldsymbol{w}\|^2$
- To find the optimal w, set the gradient of the error with respect to w equal to 0 and solve for w

$$\frac{\partial}{\partial \boldsymbol{w}} E(\boldsymbol{w}) = 0 = \frac{\partial}{\partial \boldsymbol{w}} \|\boldsymbol{d} - \boldsymbol{X}\boldsymbol{w}\|^2$$

- See The Matrix Cookbook (Petersen & Pedersen)

# Linear regression in matrix notation

- $\frac{\partial}{\partial \boldsymbol{w}} E(\boldsymbol{w}) = \frac{\partial}{\partial \boldsymbol{w}} \|\boldsymbol{d} - \boldsymbol{X}\boldsymbol{w}\|^2$

$$= \frac{\partial}{\partial \boldsymbol{w}} (\boldsymbol{d} - \boldsymbol{X}\boldsymbol{w})^T (\boldsymbol{d} - \boldsymbol{X}\boldsymbol{w})$$

$$= \frac{\partial}{\partial \boldsymbol{w}} \boldsymbol{d}^T \boldsymbol{d} - \boldsymbol{2}\boldsymbol{w}^T \boldsymbol{X}^T \boldsymbol{d} + \boldsymbol{w}^T \boldsymbol{X}^T \boldsymbol{X}\boldsymbol{w}$$

$$= -2\boldsymbol{X}^T \boldsymbol{d} - 2\boldsymbol{X}^T \boldsymbol{X}\boldsymbol{w}$$

- $\frac{\partial}{\partial \boldsymbol{w}} E(\boldsymbol{w}) = 0 = -2\boldsymbol{X}^T \boldsymbol{d} - 2\boldsymbol{X}^T \boldsymbol{X}\boldsymbol{w}$

$$\Rightarrow \boldsymbol{w} = (\boldsymbol{X}^T \boldsymbol{X})^{-1} \boldsymbol{X}^T \boldsymbol{d}$$

- Much cleaner!

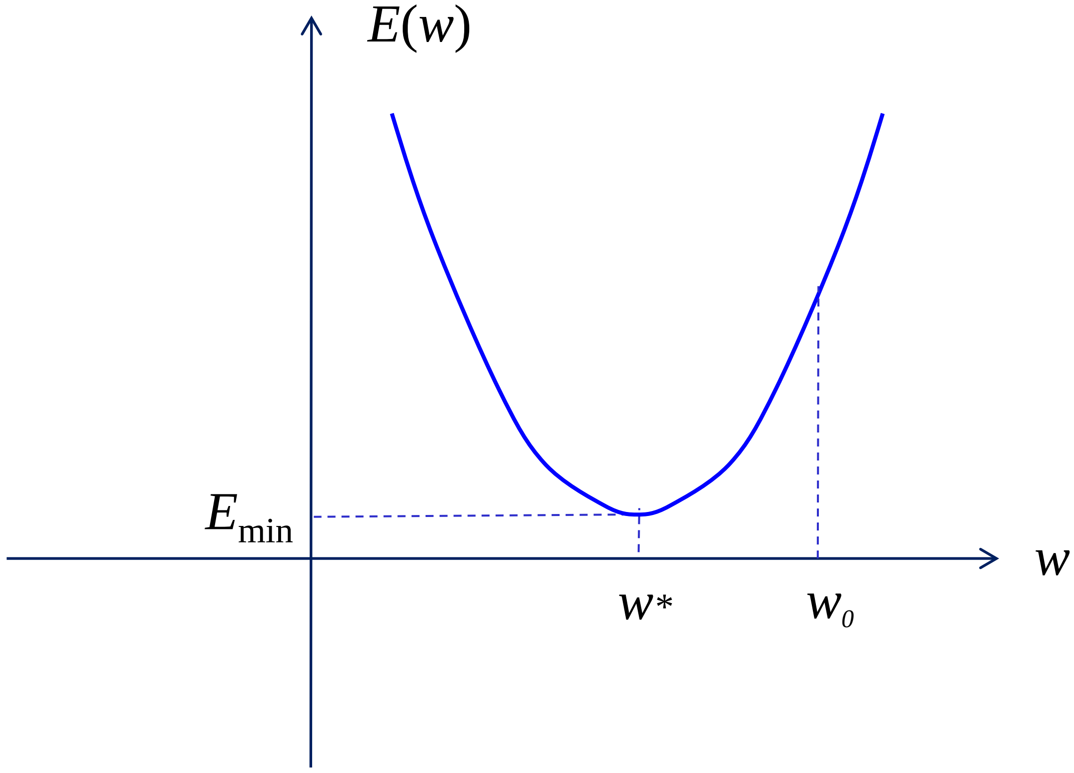
# Finding optimal parameters via search

- Often there is no closed form solution for $\frac{\partial}{\partial \boldsymbol{w}} E(\boldsymbol{w}) = 0$
- We can still use the gradient in a numerical solution
- We will still use the same example to permit comparison
- For simplicity's sake, set $b = 0$

$$E(w) = \frac{1}{2}\sum_{i=1}^{N}(d_i - wx_i)^2$$

$E(w)$ is called a cost function

# Cost function




- Question: how can we update $w$ from $w_0$ to minimize $E$?

# Gradient and directional derivatives

- Consider a two-variable function $f(x, y)$. Its gradient at the point $(x_0, y_0)^T$ is defined as

$$\nabla \mathrm{f} = \left( \frac{\partial f(x,y)}{\partial x}, \frac{\partial f(x,y)}{\partial y} \right)^T \Bigg|_{\substack{x=x_0 \\ y=y_0}}$$

$$= f_x(x_0, y_0)\mathrm{u}_x + f_y(x_0, y_0)\mathrm{u}_y$$

where $\mathbf{u}_x$ and $\mathbf{u}_y$ are unit vectors in the x and y directions, and $f_x = \partial f / \partial x$ and $f_y = \partial f / \partial y$

# Gradient and directional derivatives (cont.)

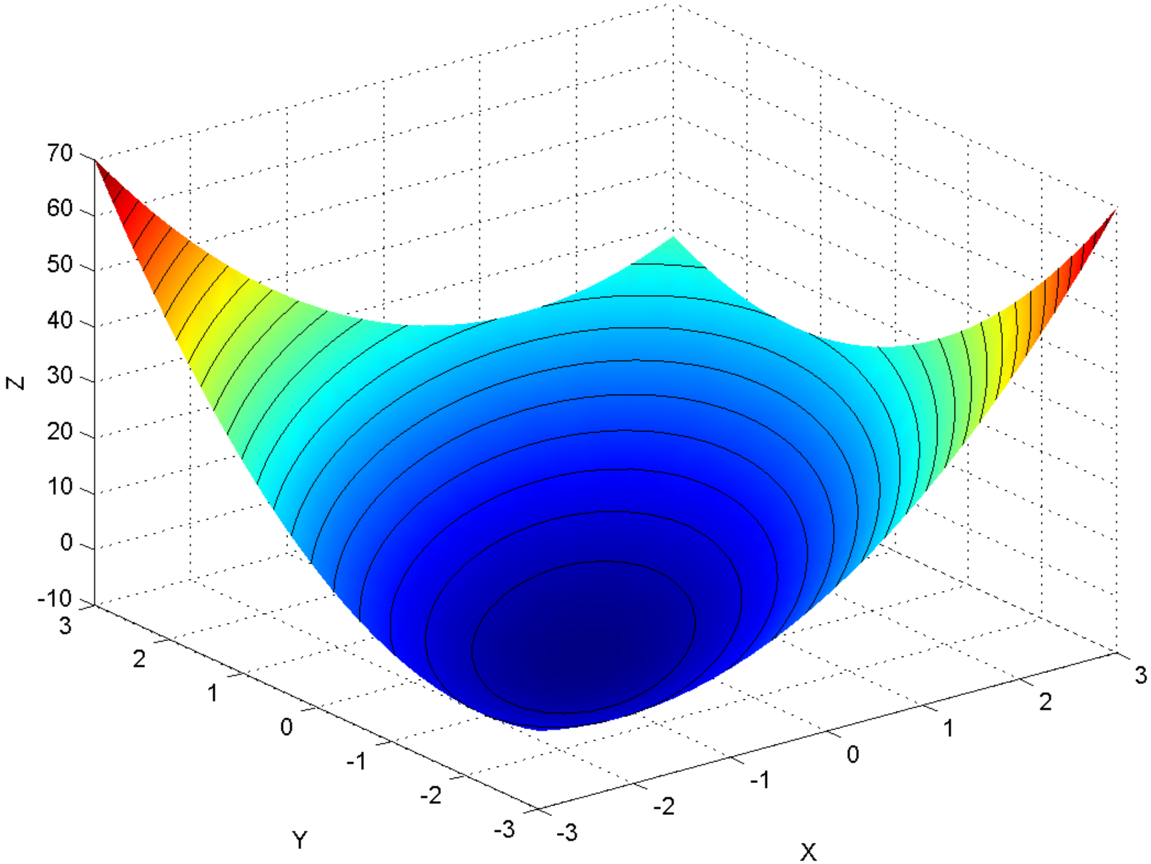
- At any given direction, u = $a$u$_x$ + $b$u$_y$, with $\sqrt{a^2+b^2}=1$, the directional derivative at $(x_0, y_0)^T$ along the unit vector u is

$$
\begin{aligned}
D_{\mathrm{u}} f_x(x_0, y_0) &= \lim_{h\to 0} \frac{f(x_0+ha, y_0+hb) - f(x_0, y_0)}{h} \\
&= \lim_{h\to 0} \frac{[f(x_0+ha, y_0+hb) - f(x_0, y_0+hb)] + [f(x_0, y_0+hb) - f(x_0, y_0)]}{h} \\
&= af_x(x_0, y_0) + bf_y(x_0, y_0) \\
&= \nabla \mathrm{f}(x_0, y_0)^T \mathrm{u}
\end{aligned}
$$

  - Which direction has the greatest slope? **The gradient** because of the dot product!

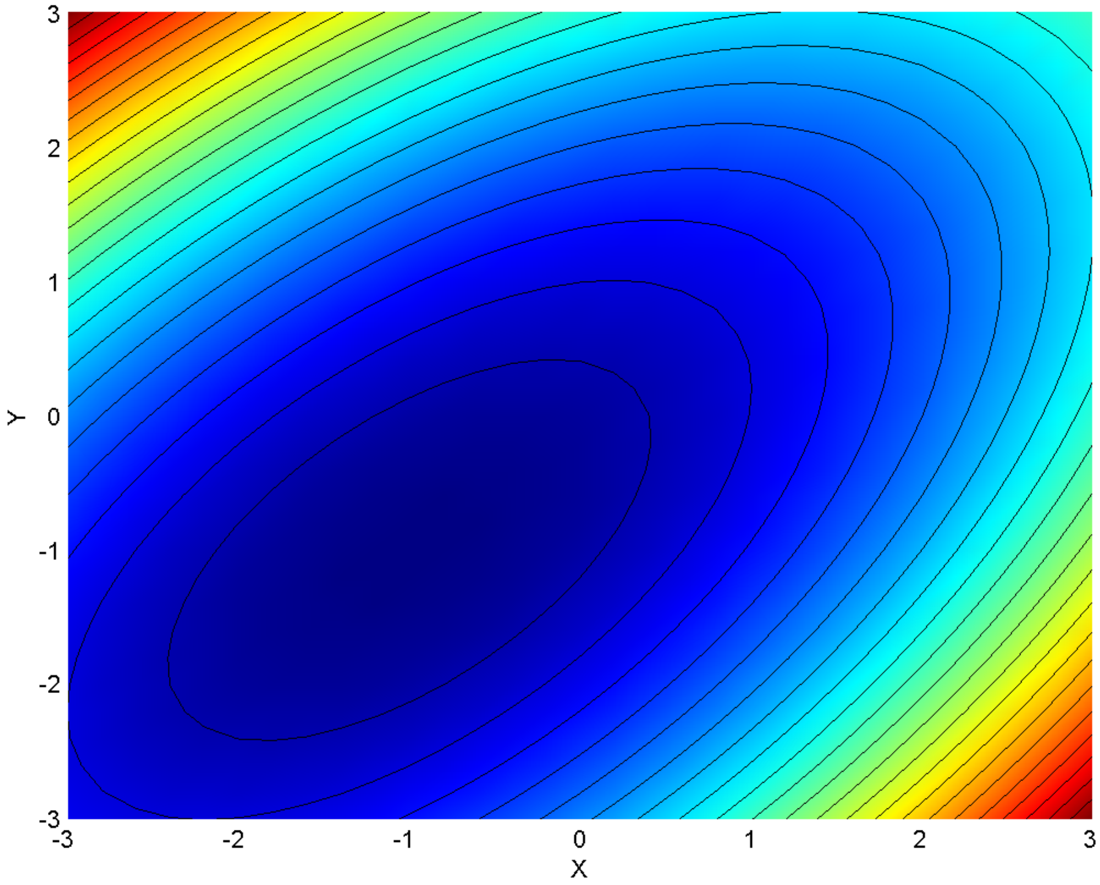
# Gradient and directional derivatives (cont.)

- Example: $\mathrm{f}(\mathrm{x}, y) = \frac{5}{2}x^2 - 3xy + \frac{5}{2}y^2 + 2x + 2y$



# Gradient and directional derivatives (cont.)

- Example: $f(x, y) = \frac{5}{2}x^2 - 3xy + \frac{5}{2}y^2 + 2x + 2y$



# Gradient and directional derivatives (cont.)

- The level curves of a function $f(x, y)$ are curves such that $f(x, y) = k$
- Thus, the directional derivative along a level curve is 0

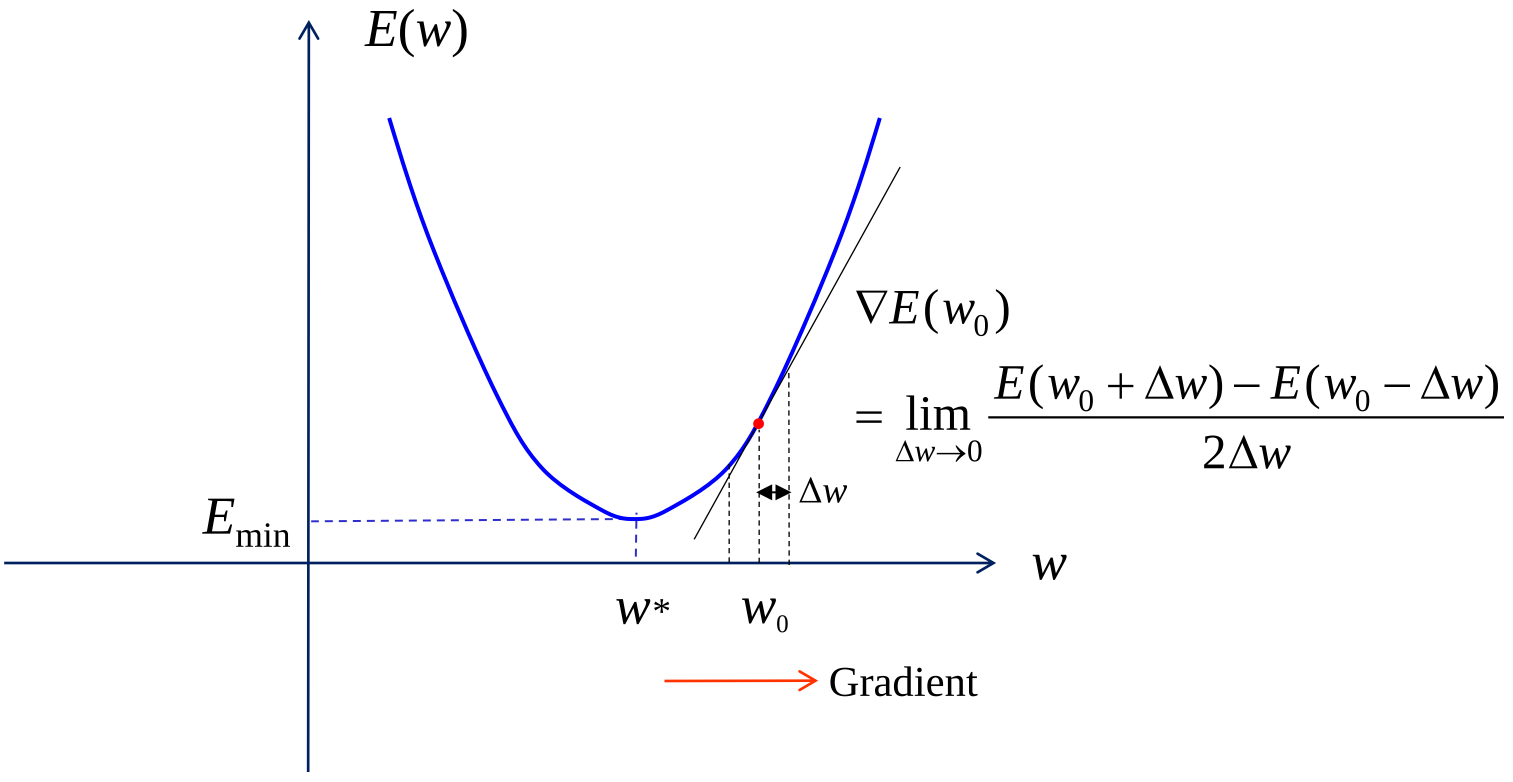
$$D_{\mathrm{u}} = \nabla \mathrm{f}(x_0, y_0)^T \mathrm{u} = 0$$

- And the gradient vector is perpendicular to the level curve

# Gradient and directional derivatives (cont.)

- The gradient of a cost function is a vector with the dimension of w that points to the direction of maximum *E* increase and with a magnitude equal to the slope of the tangent of the cost function along that direction
  - Can the slope be negative?

# Gradient illustration



# Gradient descent

- Minimize the cost function via gradient (steepest) descent – a case of hill-climbing

$$w(n+1) = w(n) - \eta \nabla E(n)$$

$n$: iteration number
$\eta$: learning rate

- See previous figure

# Gradient descent (cont.)

- For the mean-square-error cost function and linear neurons

$$E(n) = \frac{1}{2}e^2(n) = \frac{1}{2}[d(n) - y(n)]^2$$

$$= \frac{1}{2}[d(n) - w(n)x(n)]^2$$

$$\nabla E(n) = \frac{\partial E}{\partial w(n)} = \frac{1}{2}\frac{\partial e^2(n)}{\partial w(n)}$$

$$= -e(n)x(n)$$

# Gradient descent (cont.)

- Hence

$$
\begin{aligned}
w(n+1) &= w(n) + \eta e(n) x(n) \\
&= w(n) + \eta [d(n) - y(n)] x(n)
\end{aligned}
$$

  - This is the least-mean-square (LMS) algorithm, or the Widrow-Hoff rule

# Stochastic gradient descent

- If the cost function is of the form

$$E(w) = \sum_{n=1}^{N} E_n(w)$$

- Then one gradient descent step requires computing

$$\Delta \mathrm{w} = \frac{\partial}{\partial w} E(w) = \sum_{n=1}^{N} \frac{\partial}{\partial w} E_n(w)$$

- Which means computing $E(w)$ or its gradient for every data point
- Many steps may be required to reach an optimum

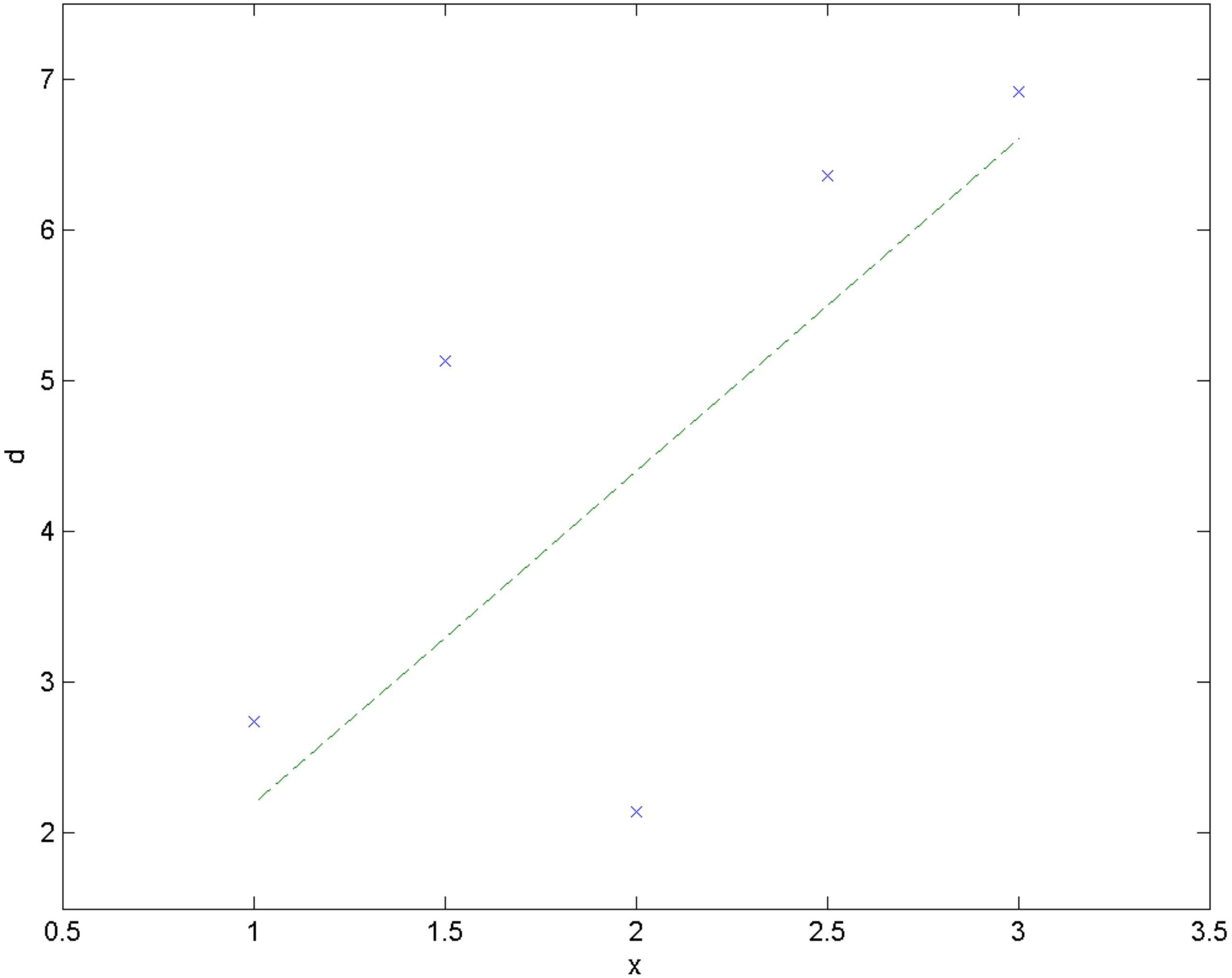
# Stochastic gradient descent

- It is generally much more computationally efficient to use

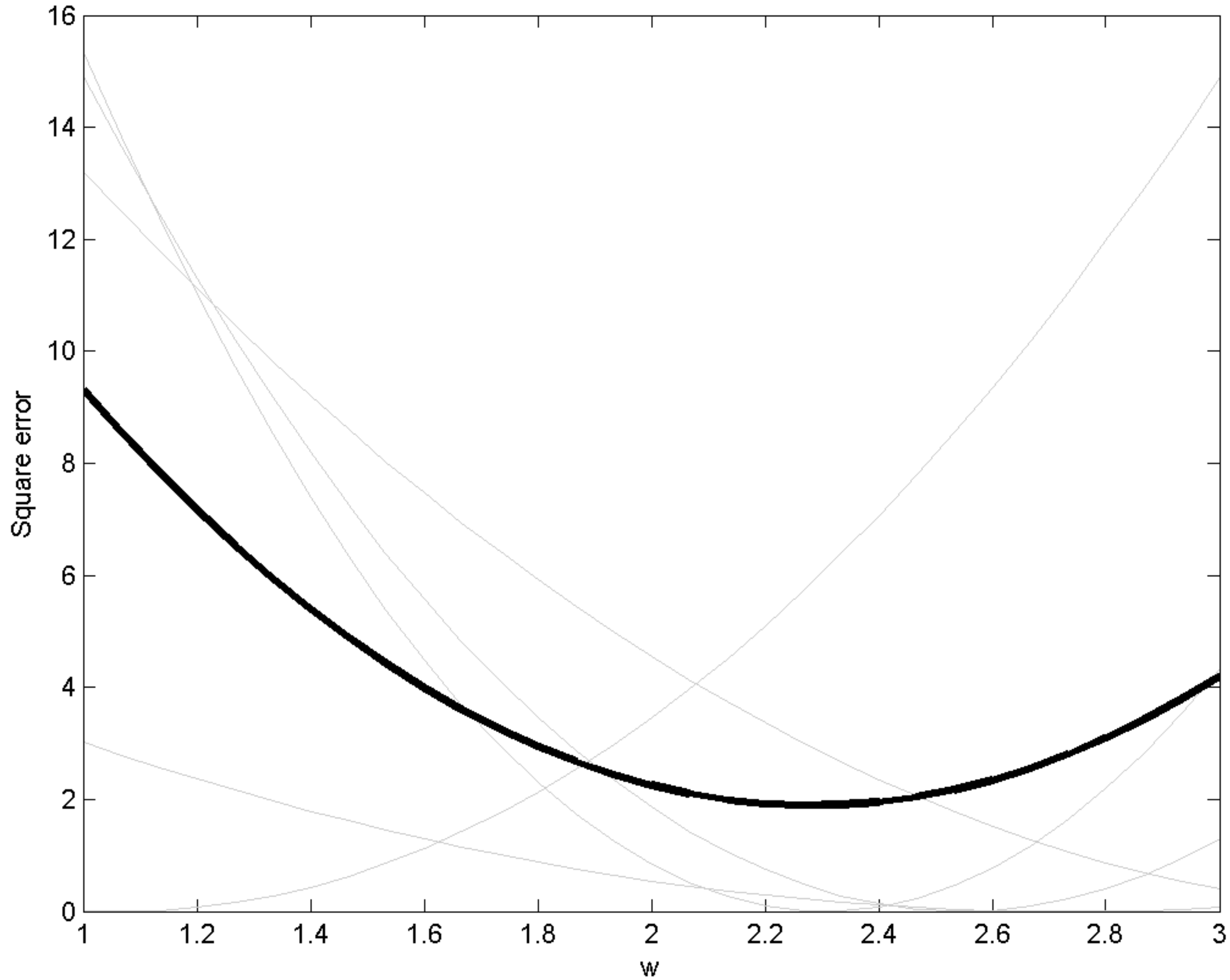
$$\Delta w = \sum_{n=n_i}^{n_i+n_b-1} \frac{\partial}{\partial w} E_n(w)$$

- For small values of $n_b$
- This update rule may converge in many fewer passes through the data (epochs)

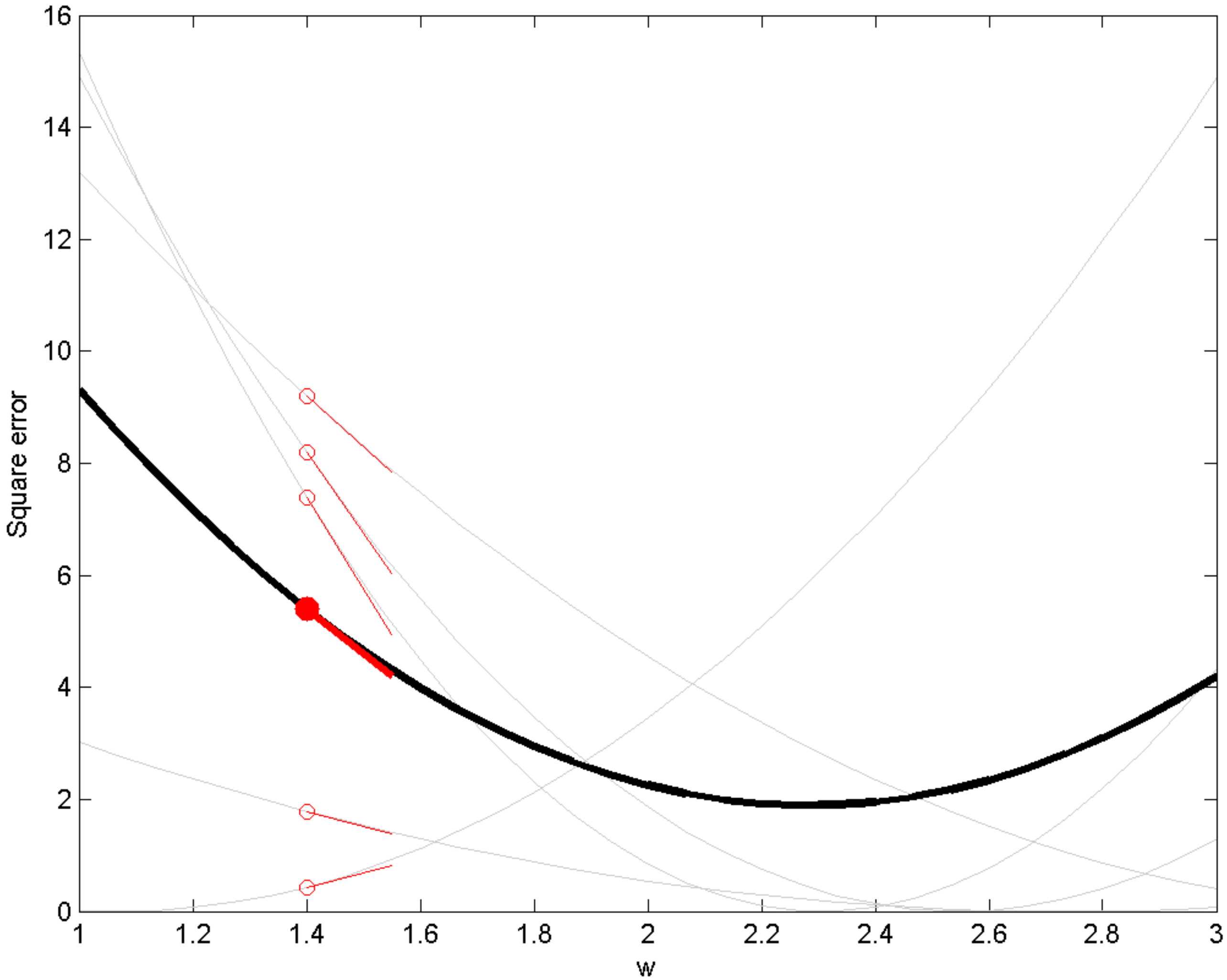
# Stochastic gradient descent example



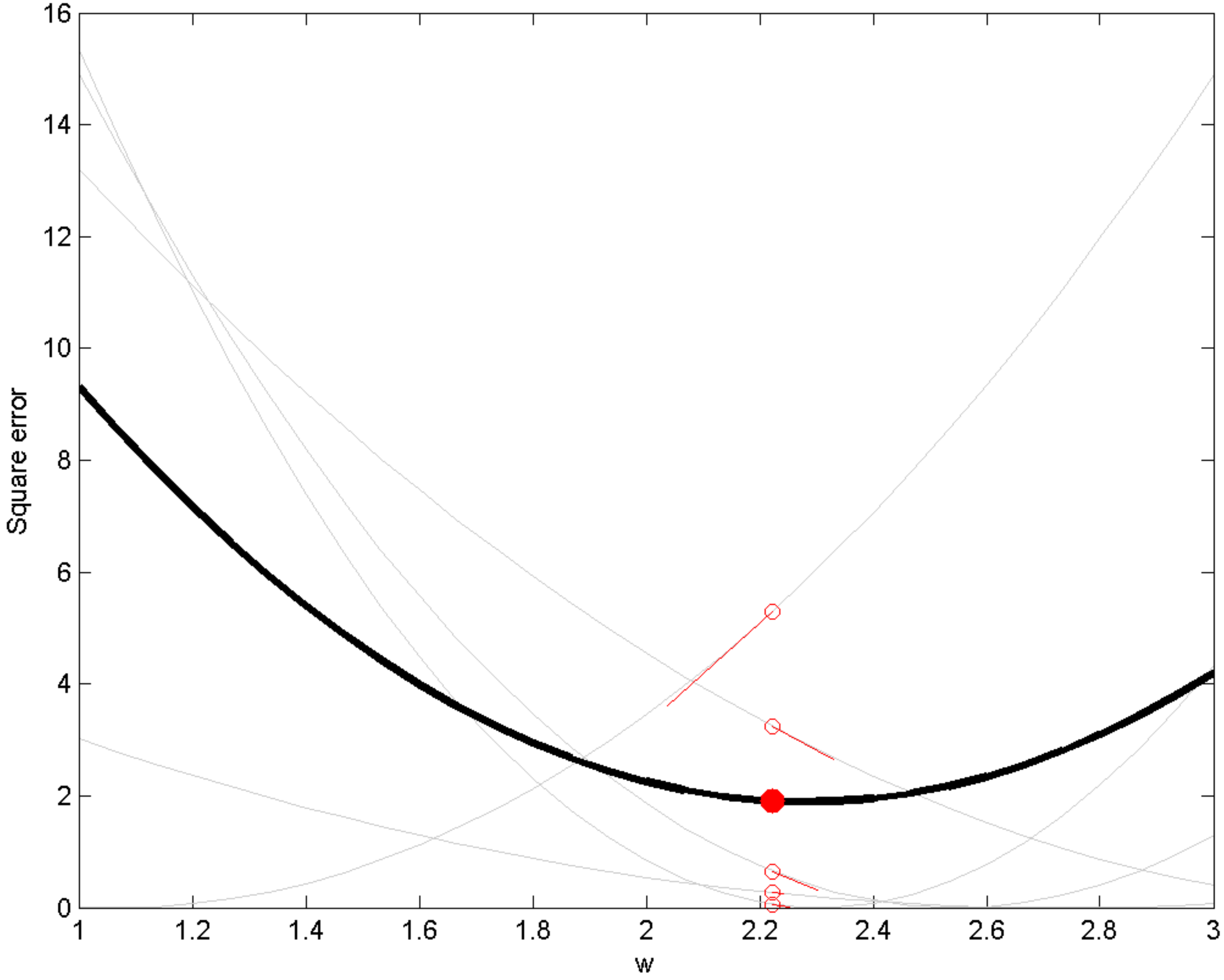
# Stochastic gradient descent error functions



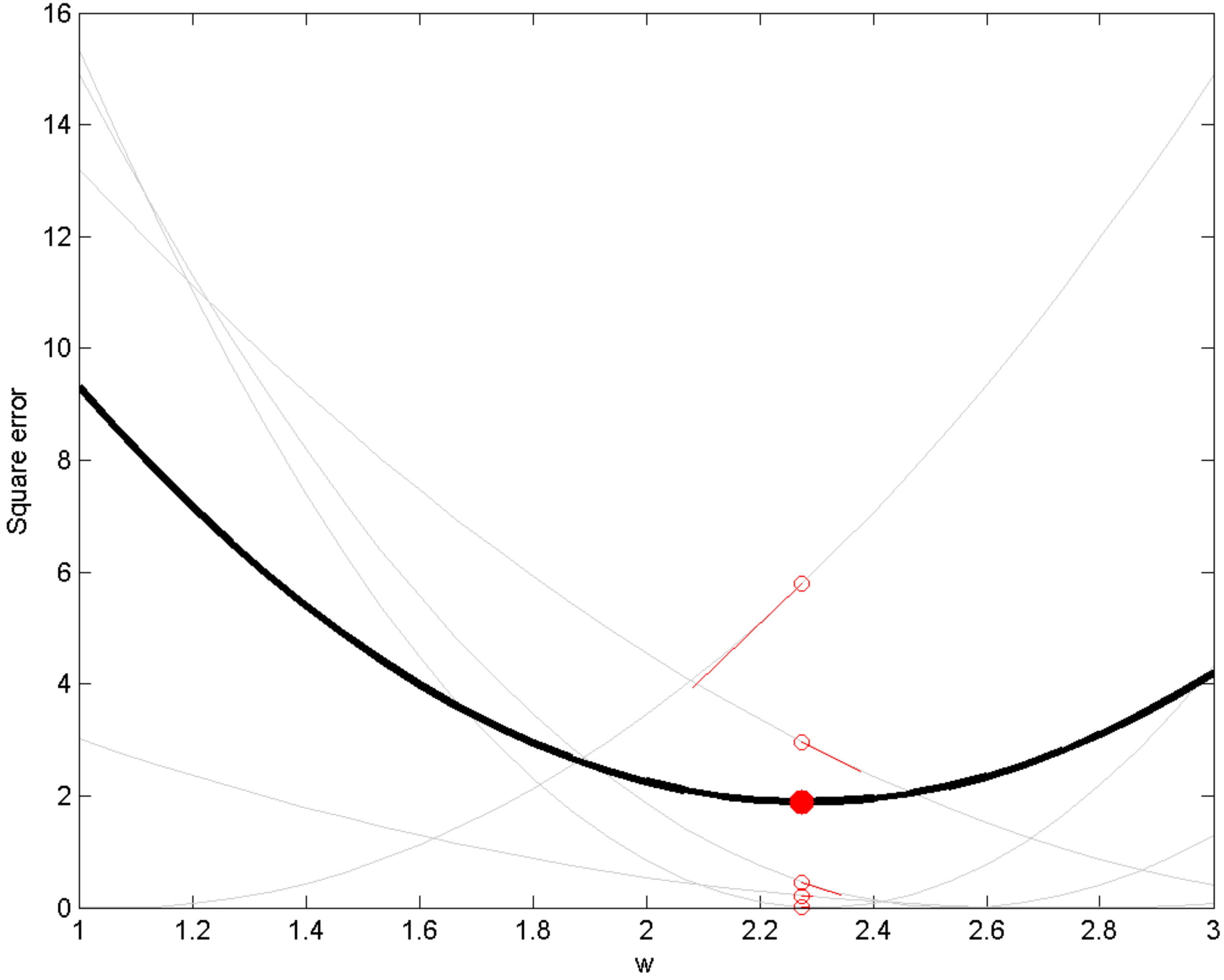
# Stochastic gradient descent gradients



# Stochastic gradient descent animation



# Gradient descent animation



# Multi-variable LMS

- The analysis for the one-variable case extends to the multi-variable case

$$E(n) = \frac{1}{2}[d(n) - \mathbf{w}^T(n)\mathbf{x}(n)]^2$$

$$\nabla \mathrm{E}(\mathrm{w}) = \left( \frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \ldots, \frac{\partial E}{\partial w_m} \right)^T$$

where $w_0 = b$ (bias) and $x_0 = 1$, as done for perceptron learning

# Multi-variable LMS (cont.)

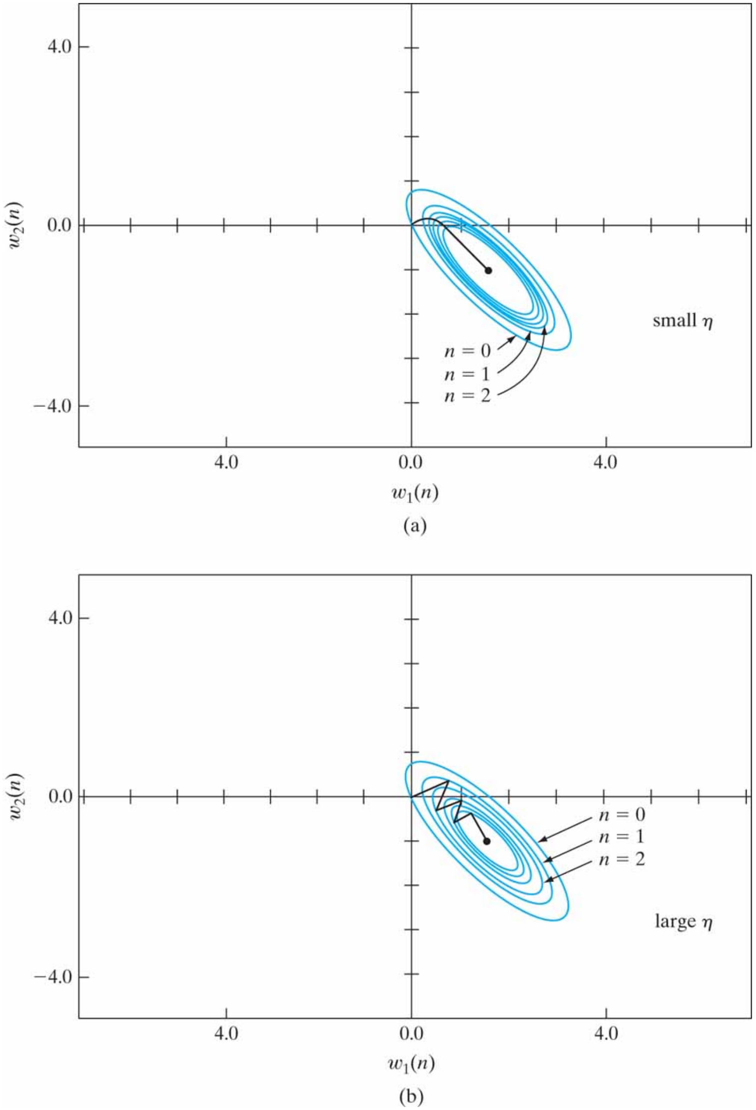
- The LMS algorithm

$$\begin{aligned}\mathbf{w}(n+1) &= \mathbf{w}(n) - \eta\nabla\mathbf{E}(n) \\ &= \mathbf{w}(n) + \eta e(n)\mathbf{x}(n) \\ &= \mathbf{w}(n) + \eta[d(n) - y(n)]\mathbf{x}(n)\end{aligned}$$

# LMS algorithm remarks

- The LMS rule is exactly the same equation as the perceptron learning rule
- Perceptron learning is for nonlinear (M-P) neurons, whereas LMS learning is for linear neurons.
  - i.e., perceptron learning is for classification and LMS is for function approximation
- LMS should be less sensitive to noise in the input data than perceptrons
  - On the other hand, LMS learning converges slowly
- Newton's method changes weights in the direction of the minimum $E(\mathrm{w})$ and leads to fast convergence.
  - But it is not online and is computationally expensive

# Stability of adaptation




- When $\eta$ is too small, learning converges slowly
- When $\eta$ is too large, learning doesn't converge

# Learning rate annealing

- Basic idea: start with a large rate but gradually decrease it
- Stochastic approximation

$$\eta(n) = \frac{c}{n}$$

$c$ is a positive parameter

# Learning rate annealing (cont.)

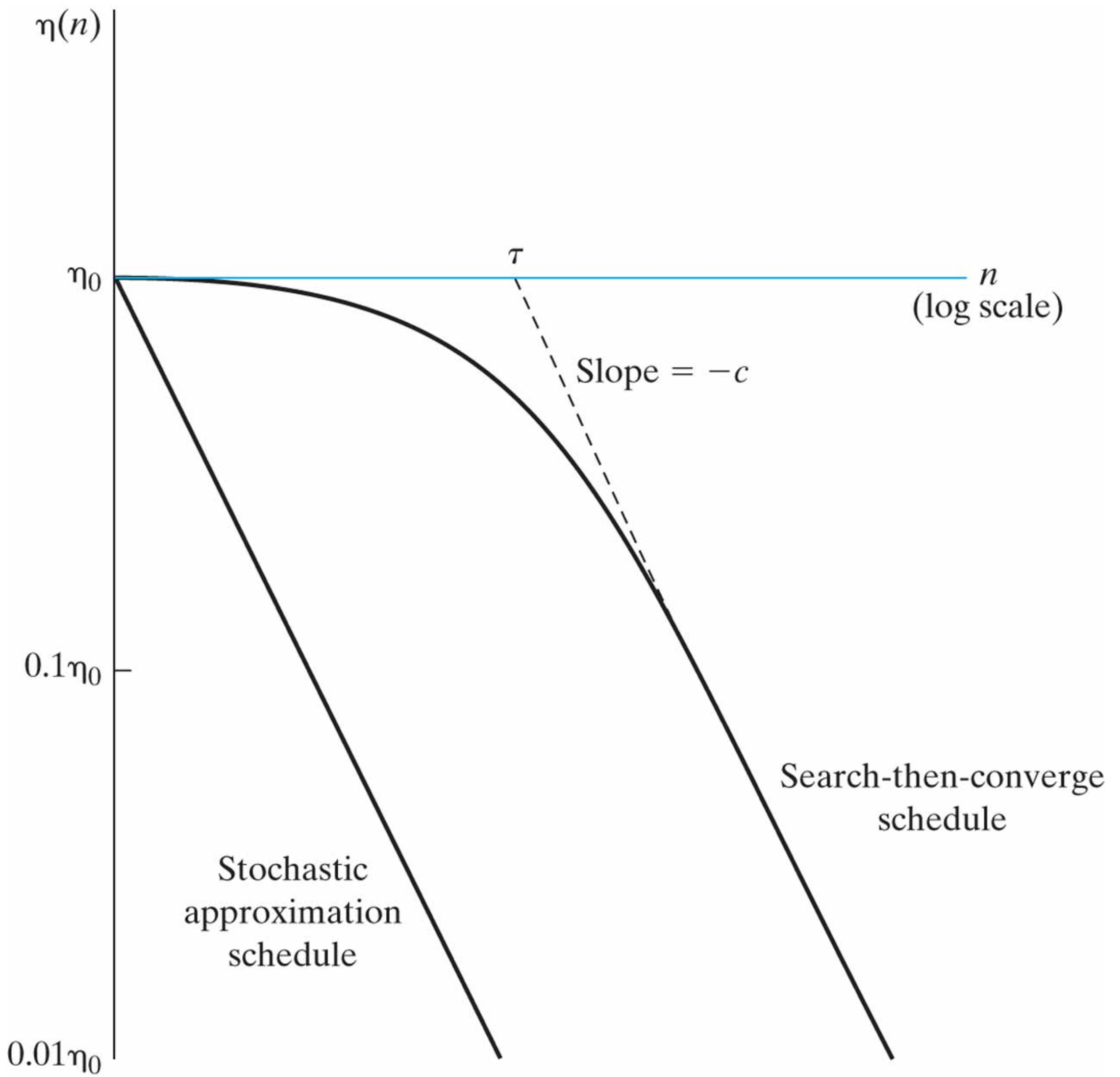
- Search-then-converge

$$\eta(n) = \frac{\eta_0}{1 + (n/\tau)}$$

$\eta_0$ and $\tau$ are positive parameters

- When $n$ is small compared to $\tau$, learning rate is approximately constant
- When $n$ is large compared to $\tau$, learning rule schedule roughly follows stochastic approximation

# Rate annealing illustration



# Nonlinear neurons

- To extend the LMS algorithm to nonlinear neurons, consider differentiable activation function $\varphi$ at iteration $n$

$$E(n) = \frac{1}{2}[d(n) - y(n)]^2$$

$$= \frac{1}{2}\left[d(n) - \varphi\left(\sum_j w_j x_j(n)\right)\right]^2$$

# Nonlinear neurons (cont.)

- By chain rule of differentiation

$$\frac{\partial E}{\partial w_j} = \frac{\partial E}{\partial y}\frac{\partial y}{\partial v}\frac{\partial v}{\partial w_j}$$

$$= -[d(n) - y(n)]\varphi'\big(v(n)\big)x_j(n)$$

$$= -e(n)\varphi'\big(v(n)\big)x_j(n)$$

# Nonlinear neurons (cont.)

- Gradient descent gives

$$w_j(n+1) = w_j(n) + \eta e(n)\varphi'(v(n))x_j(n)$$
$$= w_j(n) + \eta\delta(n)x_j(n)$$

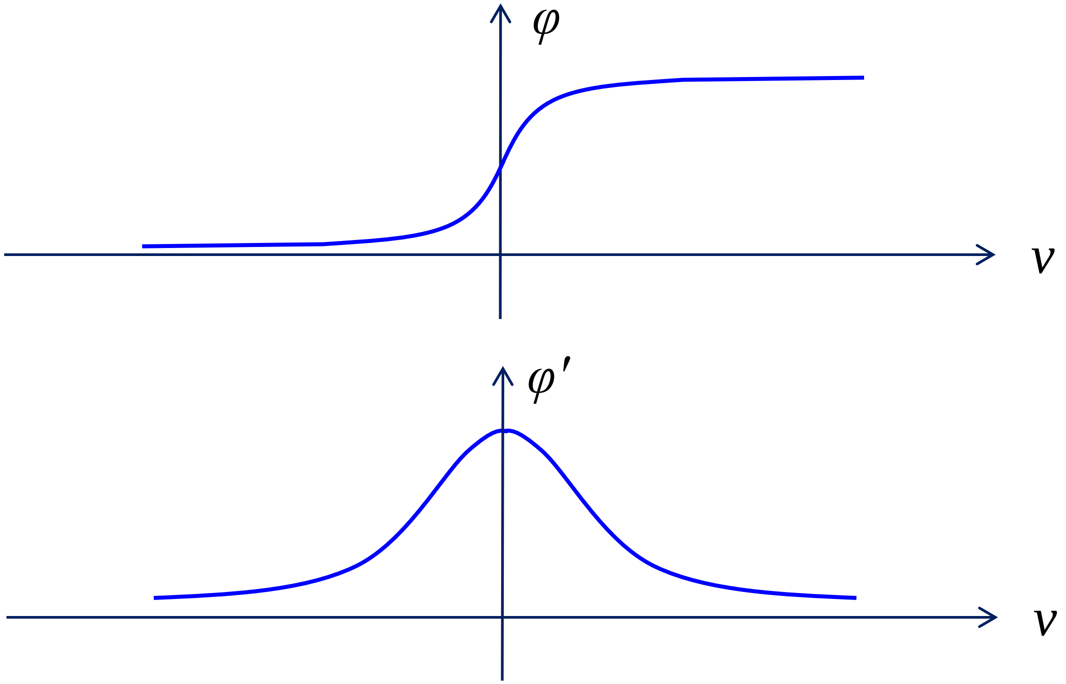
  - The above is called the delta ($\delta$) rule
- If we choose a logistic sigmoid for $\varphi$

$$\varphi(v) = \frac{1}{1+\exp(-av)}$$

then

$$\varphi'(v) = a\varphi(v)[1-\varphi(v)]$$ (see textbook)

# Role of activation function




- The role of $\varphi'$: weight update is most sensitive when $v$ is near zero