

# Regression & the LMS Algorithm

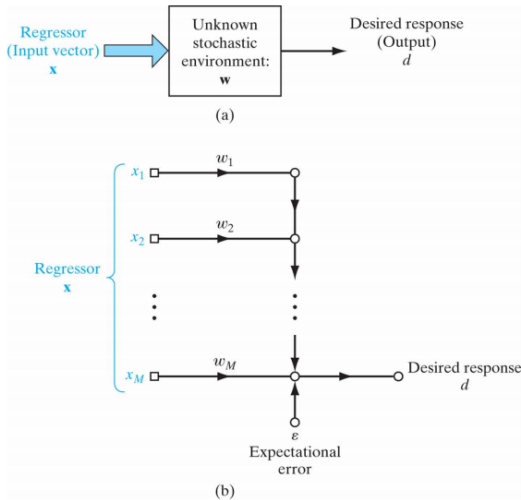
Yingming Li  
yingming@zju.edu.cn

Data Science & Engineering Research Center, ZJU

14th March 2018

Adapted from slides provided by Prof. Michael Mandel.

# 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

$$\begin{aligned} d_i &= y_i + \epsilon_i = \varphi(wx_i + b) + \epsilon \\ &= wx_i + b + \epsilon \end{aligned}$$

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

$$\epsilon_i = d_i - y_i$$

is called the (expectational) regression error.

# Linear regression

- 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

To minimize the two-variable square function, set

$$\begin{cases} \frac{\partial E}{\partial b} = 0 \\ \frac{\partial E}{\partial w} = 0 \end{cases}$$

$$\Rightarrow \begin{cases} -\sum_i (d_i - wx_i - n) = 0 \\ -\sum_i (d_i - wx_i - b)x_i = 0 \end{cases}$$

# 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  $b$

$$\begin{cases} -\sum_i (d_i - wx_i - b) &= 0 \\ -\sum_i (d_i - wx_i - b)x_i &= 0 \end{cases}$$

$\Rightarrow$

# 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  $b$

$$\begin{cases} -\sum_i (d_i - wx_i - b) = 0 \\ -\sum_i (d_i - wx_i - b)x_i = 0 \end{cases}$$

$$\Rightarrow b = \frac{\sum_i x_i^2 \sum_i d_i - \sum_i x_i \sum_i x_i d_i}{N \sum_i (x_i - \bar{x})^2}, \quad w = \frac{\sum_i (x_i - \bar{x})(d_i - \bar{d})}{\sum_i (x_i - \bar{x})^2}$$

, where an  $\bar{x}$  indicates the mean

There may exist other forms, such as  $w = \frac{\sum_i d_i (x_i - \bar{x})}{\sum_i x_i^2 - \frac{1}{m} (\sum_i x_i)^2}$ ,

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

# 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 in matrix notation

Let  $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \dots, \mathbf{x}_N]^T$ , then the model predictions are  $\mathbf{y} = \mathbf{X}\mathbf{w}$ . And the mean square error can be written as

$$E(\mathbf{w}) = \|\mathbf{d} - \mathbf{y}\|^2 = \|\mathbf{d} - \mathbf{X}\mathbf{w}\|^2$$

To find the optimal  $\mathbf{w}$ , set the gradient of the error *w.r.t.*  $\mathbf{w}$  equal to 0 and solve for  $\mathbf{w}$ .

$$\partial E(\mathbf{w}) / \partial \mathbf{w} = 0$$

## Linear regression in matrix notation

$$\begin{aligned}\frac{\partial}{\partial \mathbf{w}} E(\mathbf{w}) &= \frac{\partial}{\partial \mathbf{w}} \|\mathbf{d} - \mathbf{X}\mathbf{w}\|^2 \\&= \frac{\partial}{\partial \mathbf{w}} (\mathbf{d} - \mathbf{X}\mathbf{w})^T (\mathbf{d} - \mathbf{X}\mathbf{w}) \\&= \frac{\partial}{\partial \mathbf{w}} \mathbf{d}^T \mathbf{d} - \mathbf{d}^T \mathbf{X} \mathbf{w} - \mathbf{w}^T \mathbf{X}^T \mathbf{d} + \mathbf{w}^T \mathbf{X}^T \mathbf{X} \mathbf{w} \\&= 2\mathbf{X}^T \mathbf{X} \mathbf{w} - 2\mathbf{X}^T \mathbf{d} = 0\end{aligned}$$

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

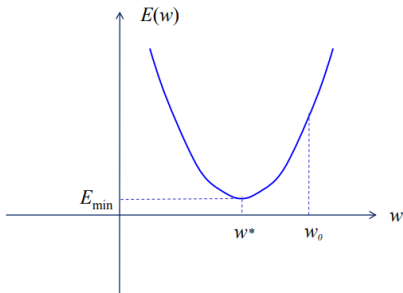
## Finding optimal parameters via search

- Often there is no closed form solution for  $\frac{\partial}{\partial \mathbf{w}} E(\mathbf{w}) = 0$
- 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  $b = 0$

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

, where  $E(w)$  is called 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

$$\begin{aligned}\nabla f &= (\partial f(x, y)/\partial x, \partial f(x, y)/\partial y)^T \Big|_{x=x_0, y=y_0} \\ &= f_x(x_0, y_0)\mathbf{u}_x + f_y(x_0, y_0)\mathbf{u}_y\end{aligned}$$

, 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

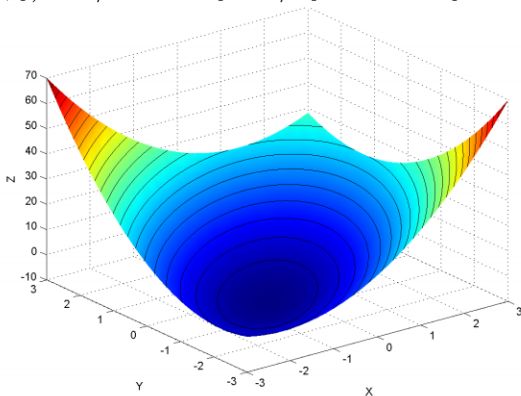
At any given direction,  $\mathbf{u} = a\mathbf{u}_x + b\mathbf{u}_y$ , with  $\sqrt{a^2 + b^2} = 1$ , the directional derivative at  $(x_0, y_0)^T$  along the unit vector  $\mathbf{u}$  is

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

Which direction has the greatest slope? The gradient! Because of the dot product.

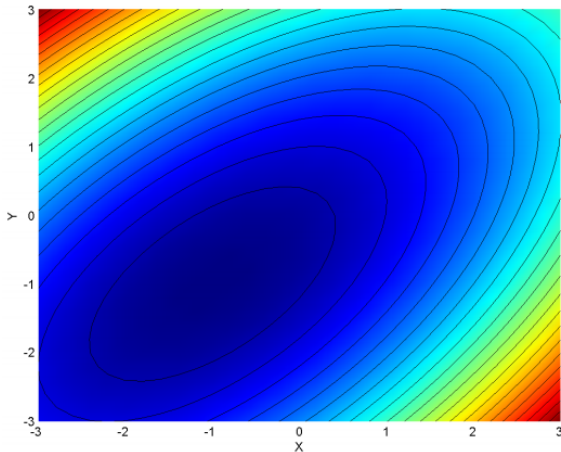
# Gradient and directional derivatives

Example:  $f(x, y) = 5/2x^2 - 3xy + 5/2y^2 + 2x + 2y$



## Gradient and directional derivatives

Example:  $f(x, y) = 5/2x^2 - 3xy + 5/2y^2 + 2x + 2y$





## 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

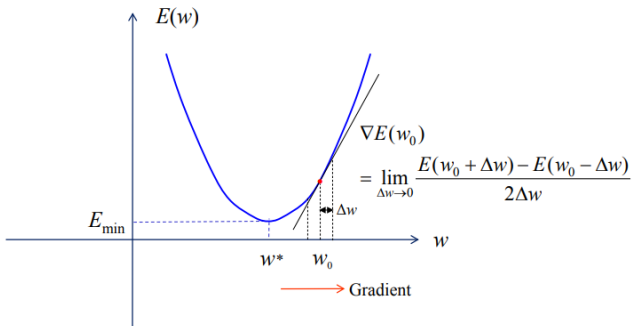
$$D_{\mathbf{u}} = \nabla f(x_0, y_0)^T \mathbf{u} = 0$$

- And the gradient vector is perpendicular to the level curve

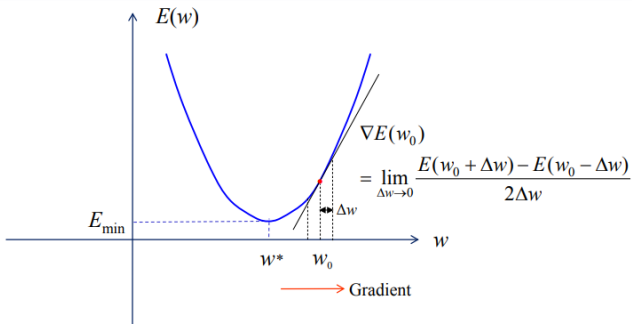
## Gradient and directional derivatives (cont.)

- 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

## Gradient descent (cont.)

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

$$\begin{aligned} E(n) &= \frac{1}{2} e^2(n) = \frac{1}{2} [d(n) - y(n)]^2 \\ &= \frac{1}{2} [d(n) - w(n)x(n)]^2 \\ \nabla E(n) &= \frac{\partial E}{\partial w(n)} = \frac{\partial e^2(n)}{2\partial w(n)} \\ &= -e(n)x(n) \end{aligned}$$

## 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

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

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

# Stochastic gradient descent

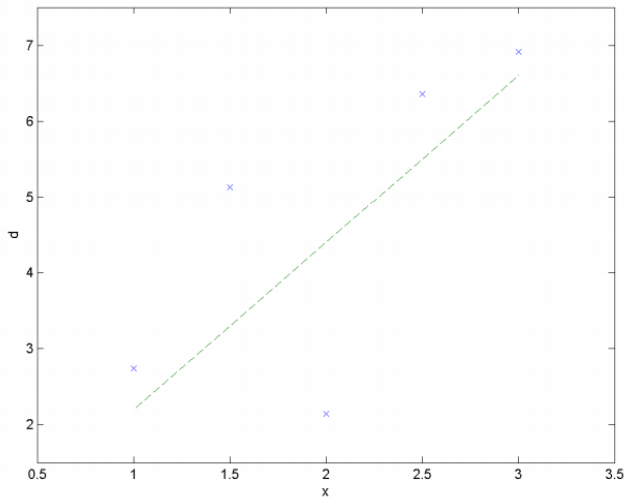
- It is generally much more computationally efficient to use

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

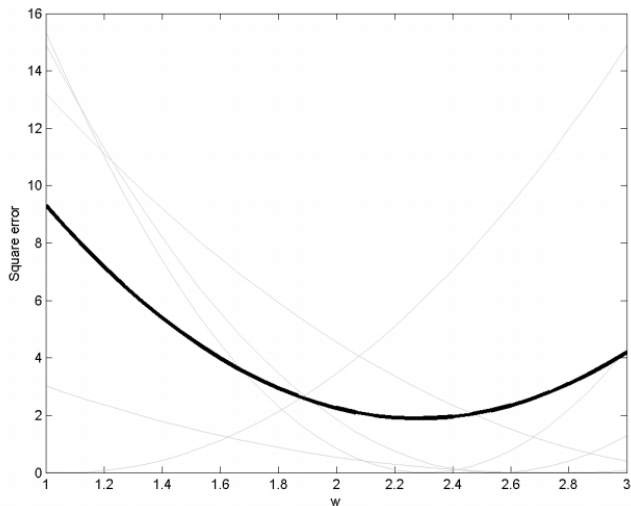
- For small values of  $n_b$
- This update rule may converge in many fewer passes through the data (epochs)



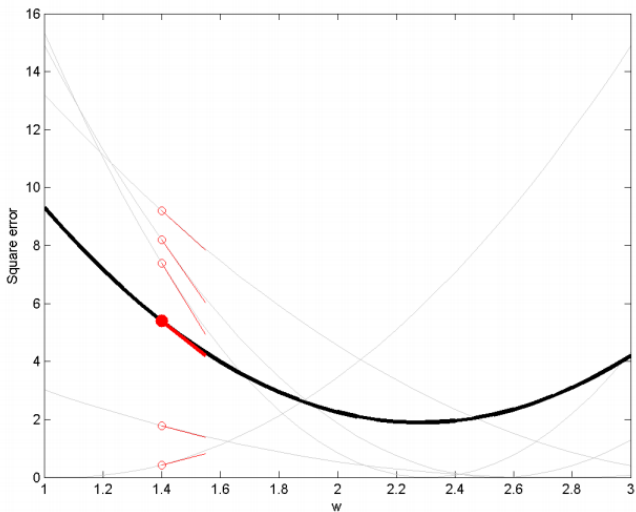
# Stochastic gradient descent example



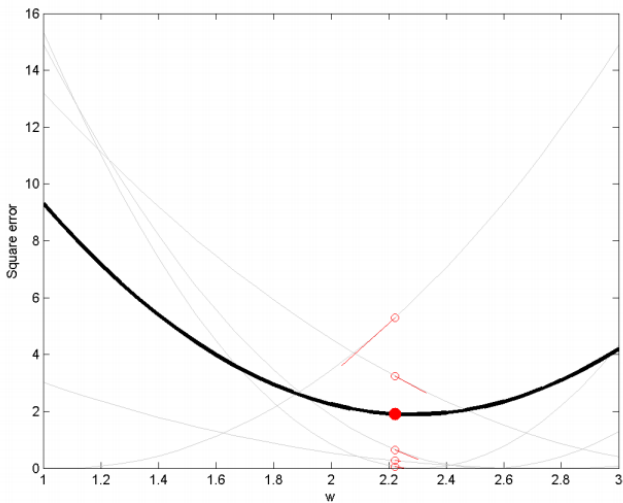
# Stochastic gradient descent error functions



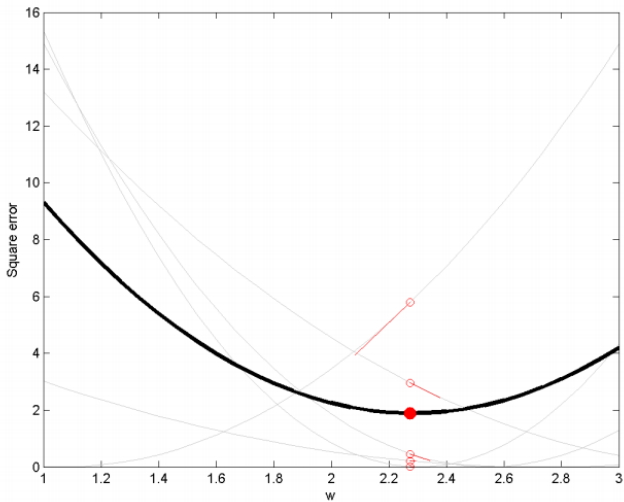
# Stochastic gradient descent gradients



# Stochastic gradient descent animation



# Gradient descent animation



## Multi-variable LMS

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

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

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

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

## Multi-variable LMS (cont.)

- The LMS algorithm

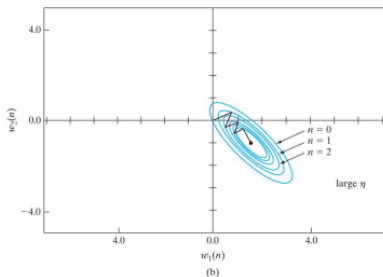
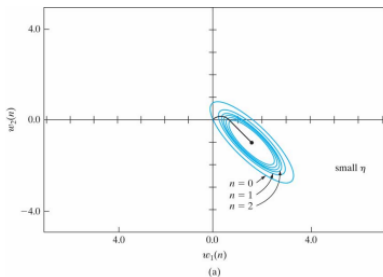
$$\begin{aligned}\mathbf{w}(n+1) &= \mathbf{w}(n) - \eta \nabla \mathbf{E}(n) \\ &= \mathbf{w}(n) + \eta e(n) \mathbf{x}(n) \\ &= \mathbf{w}(n) + \eta [d(n) - y(n)] \mathbf{x}(n)\end{aligned}$$

## 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
- Newtons method changes weights in the direction of the minimum  $E(w)$  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 does not converge

# Learning rate annealing

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

$$\eta(n) = c/n$$

$c$  is a positive parameter

## Learning rate annealing (cont.)

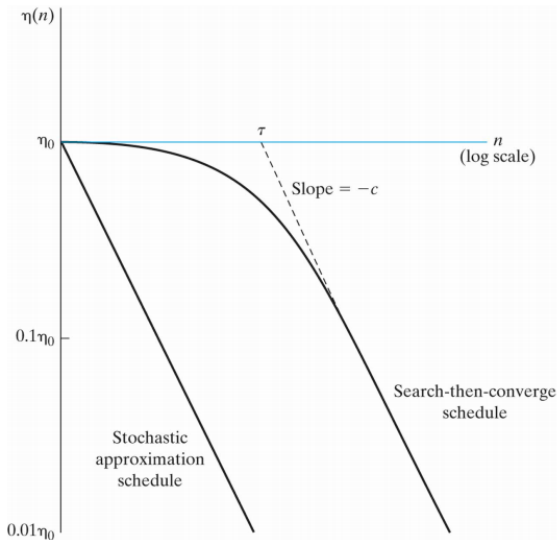
- Search-then-converge

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

$\eta_0$  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 at iteration  $n$

$$\begin{aligned} E(n) &= 1/2 [d(n) - y(n)]^2 \\ &= 1/2 \left[ d(n) - \varphi \left( \sum_j w_j x_j(n) \right) \right]^2 \end{aligned}$$

## Nonlinear neurons (cont.)

- By chain rule of differentiation

$$\begin{aligned}\frac{\partial E}{\partial w_j} &= \frac{\partial E}{\partial y} \frac{\partial y}{\partial v} \frac{\partial v}{\partial w_j} \\ &= -[d(n) - y(n)] \varphi'(v(n)) x_j(n) \\ &= -e(n) \varphi'(v(n)) x_j(n)\end{aligned}$$

## Nonlinear neurons (cont.)

- Gradient descent gives

$$\begin{aligned}w_j(n+1) &= w_j(n) + \eta e(n) \varphi'(v(n)) x_j(n) \\ &= w_j(n) + \eta \delta(n) x_j(n)\end{aligned}$$

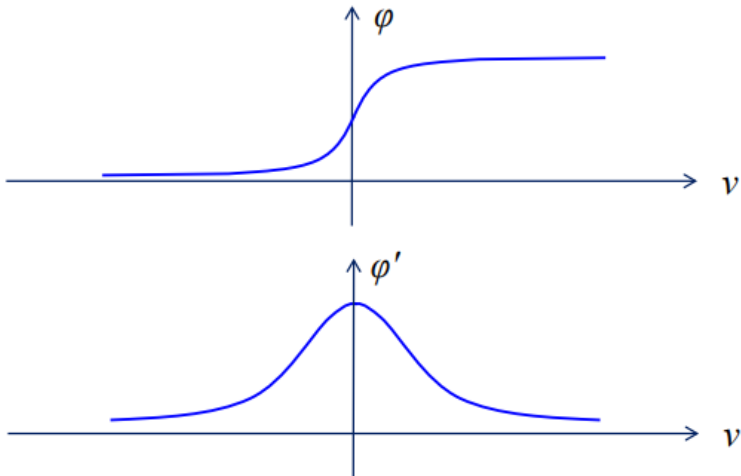
- The above is called the delta ( $\delta$ ) rule
- If we choose a logistic sigmoid for

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

then

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

## Role of activation function



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



# Thank you!