

# Variations on Backpropagation

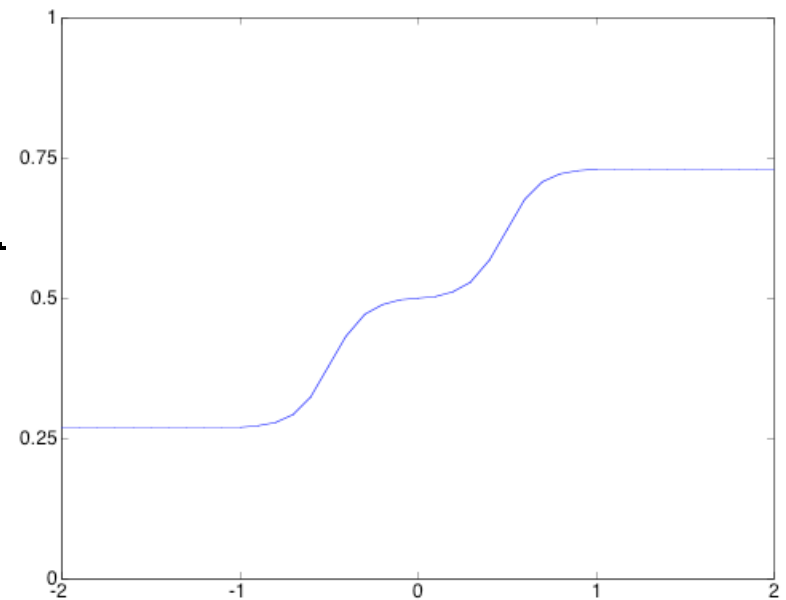
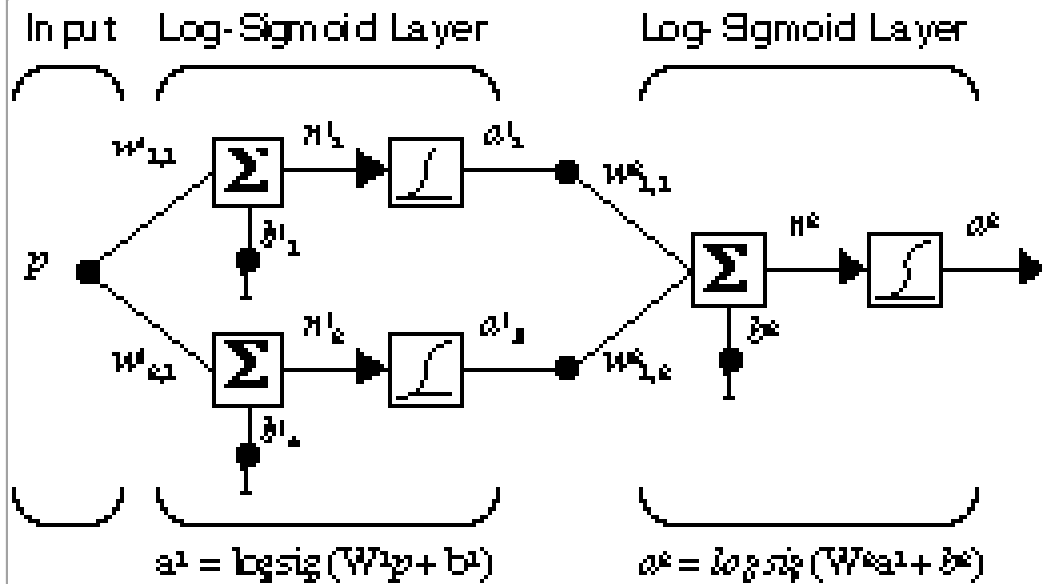
# Backprop Variations

- Heuristic Modifications
  - Momentum
  - Variable Learning Rate
  - Quickprop
- Standard Numerical Optimization
  - Conjugate Gradient
  - Newton's Method (Levenberg-Marquardt)

# Error Surface Example

Network Architecture

Nominal Function

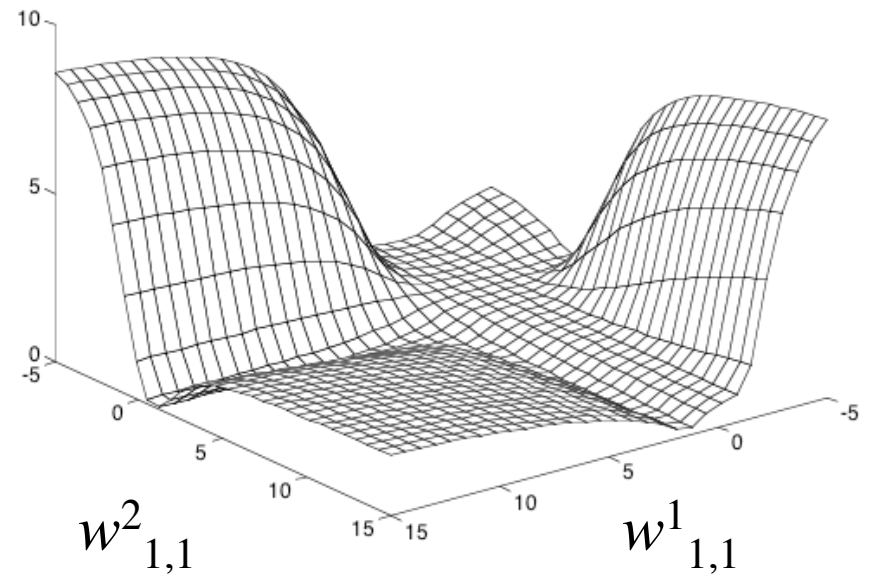
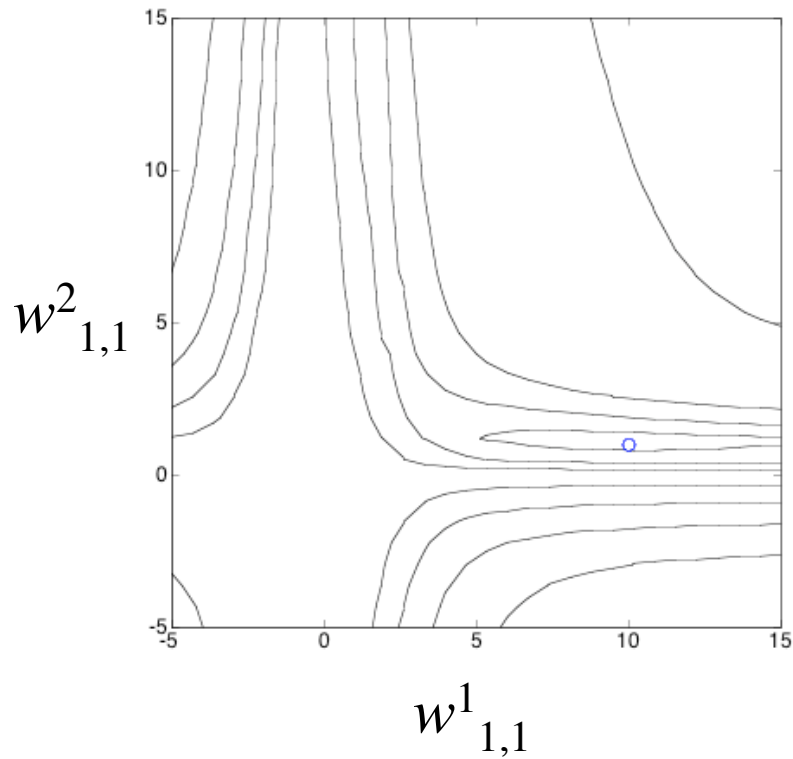


Parameter Values

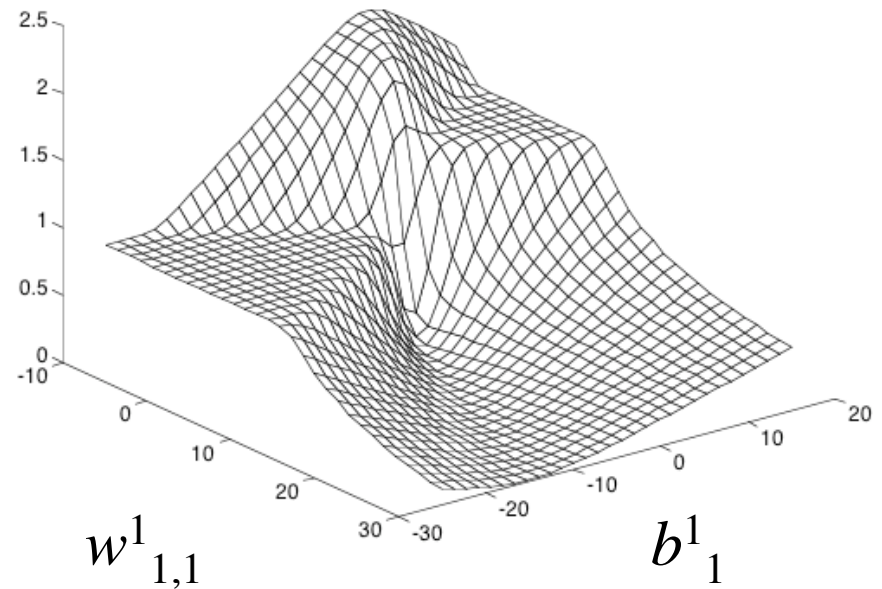
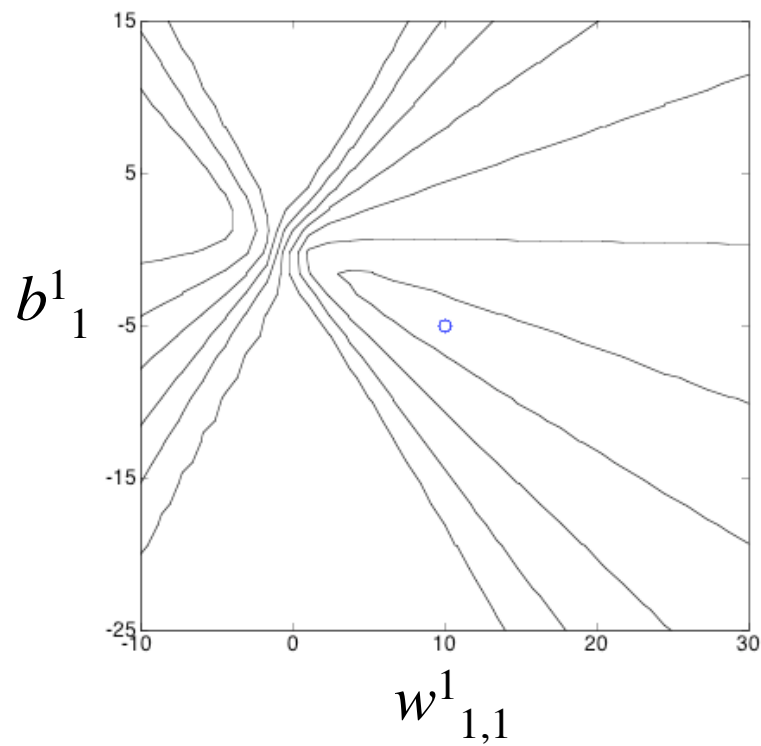
$$w_{1,1}^1 = 10 \quad w_{2,1}^1 = 10 \quad b_1^1 = -5 \quad b_2^1 = 5$$

$$w_{1,1}^2 = 1 \quad w_{1,2}^2 = 1 \quad b^2 = -1$$

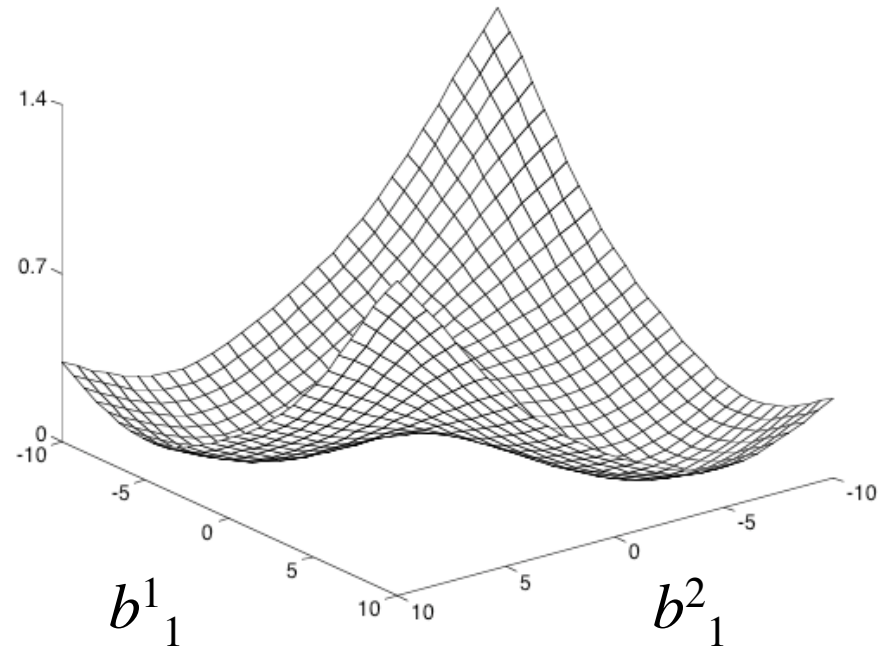
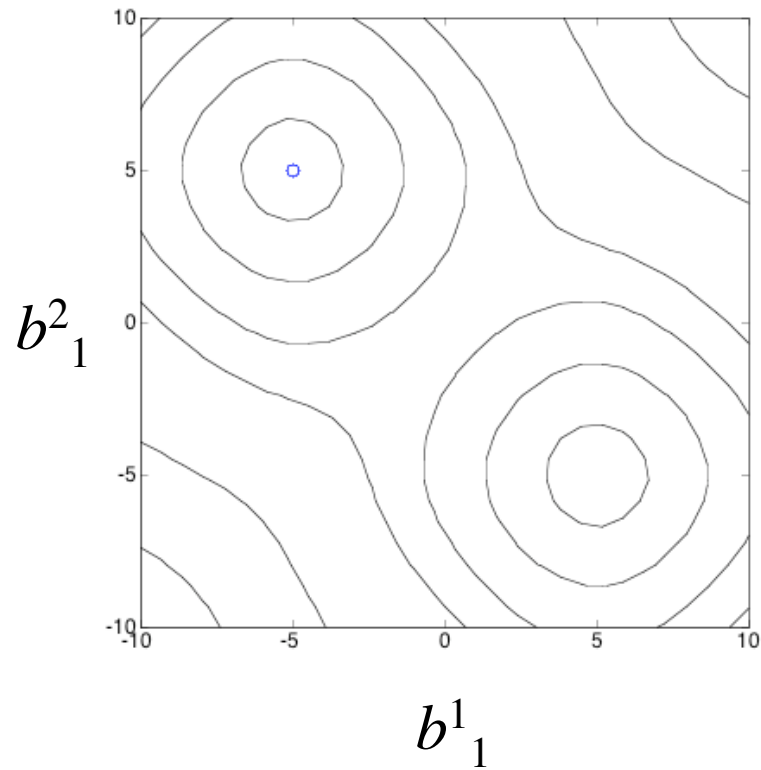
# Squared Error vs. $w^1_{1,1}$ and $w^2_{1,1}$



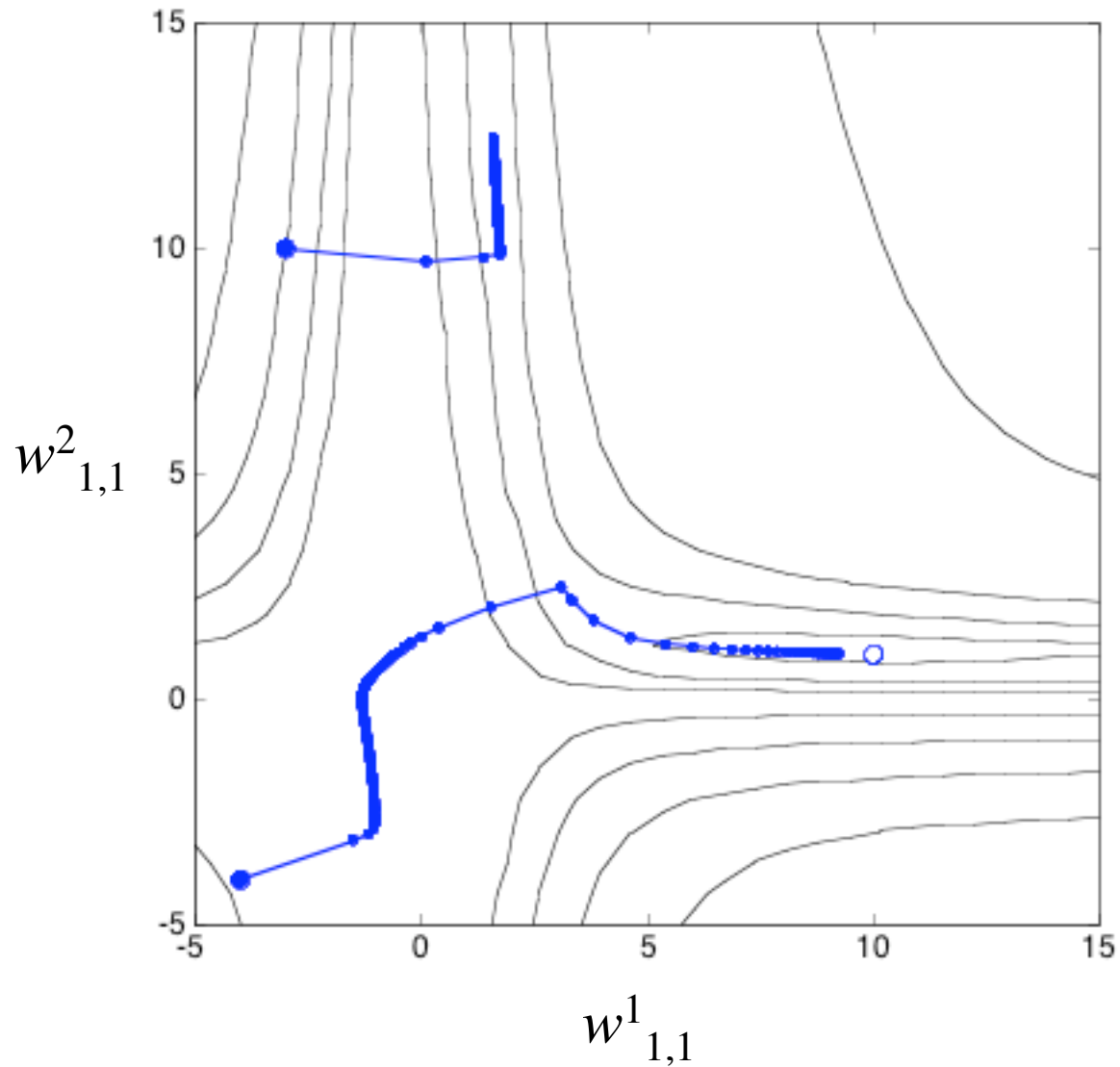
# Squared Error vs. $w_{1,1}^1$ and $b_1^1$



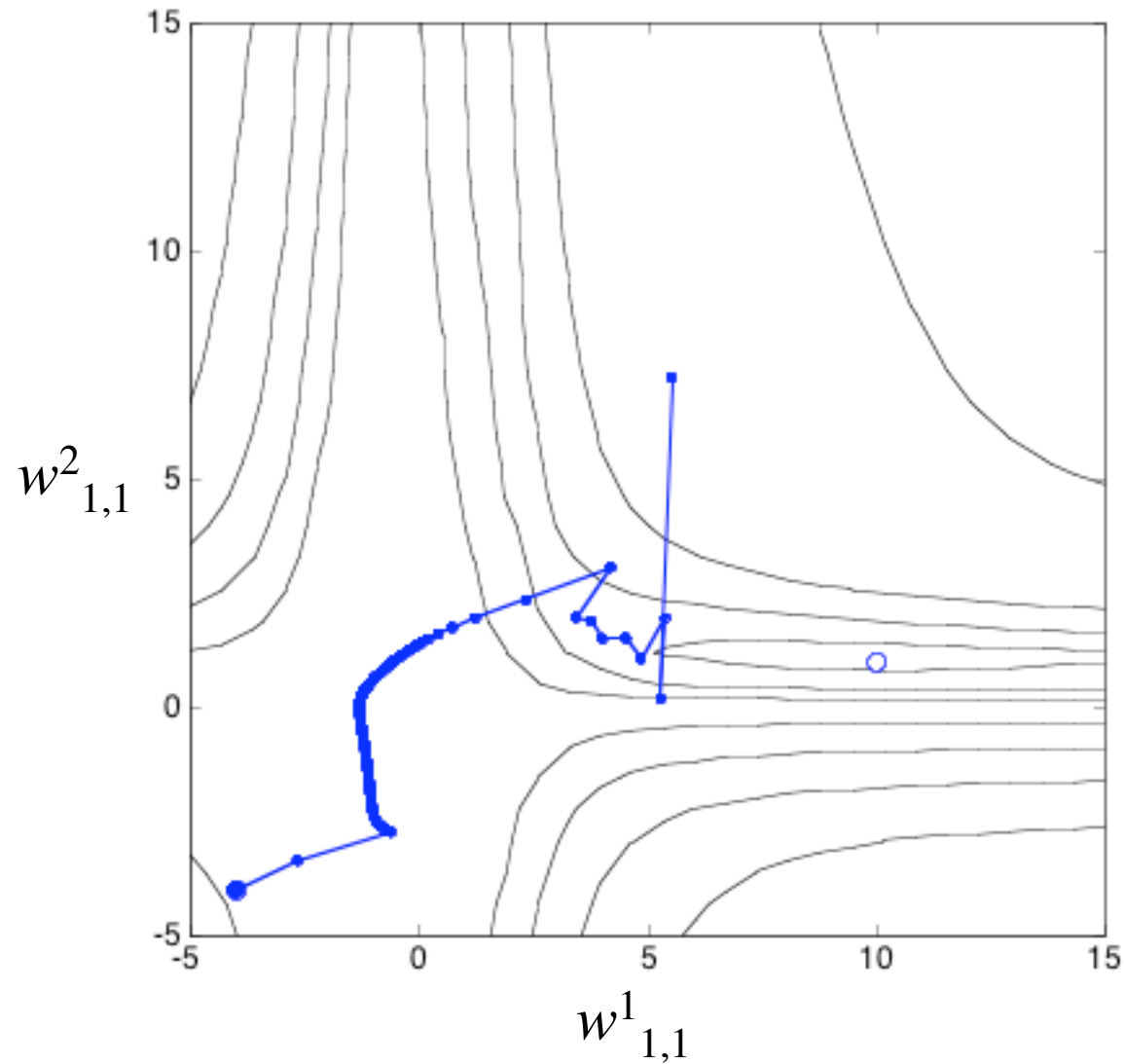
# Squared Error vs. $b^1_1$ and $b^1_2$



# Convergence Example



# Learning Rate Too Large



# Momentum Backpropagation

Steepest Descent Backpropagation  
(SDBP)

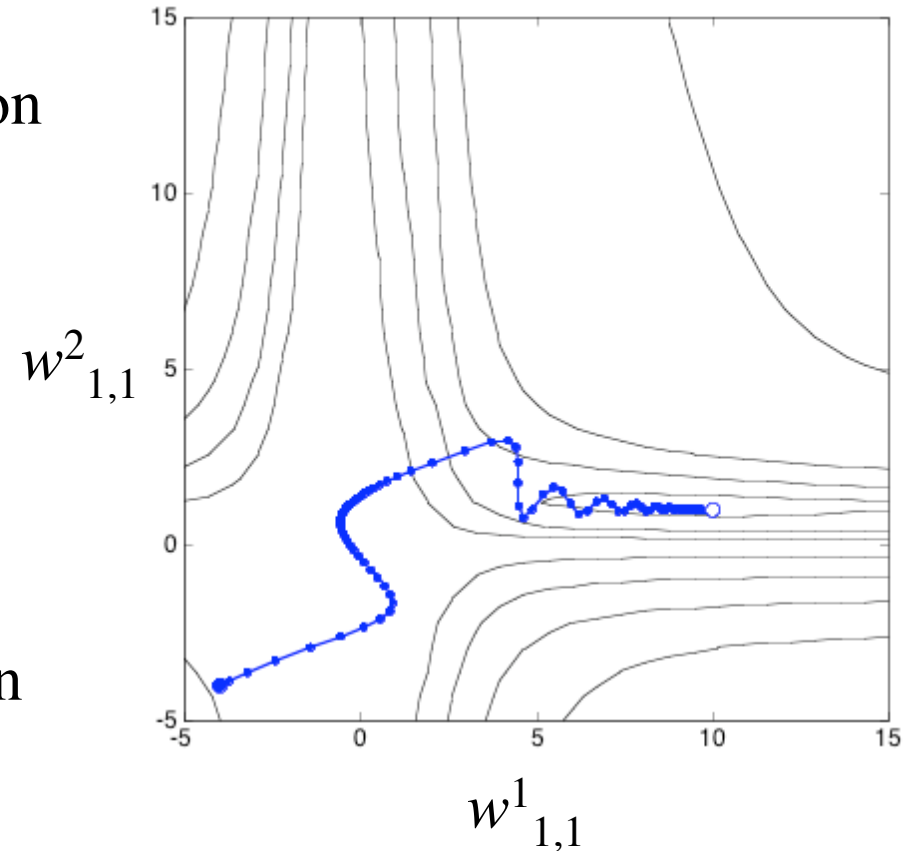
$$\Delta \mathbf{W}^m(k) = -\alpha \mathbf{s}^m (\mathbf{a}^{m-1})^T$$

$$\Delta \mathbf{b}^m(k) = -\alpha \mathbf{s}^m$$

Momentum Backpropagation  
(MOBP)

$$\Delta \mathbf{W}^m(k) = \gamma \Delta \mathbf{W}^m(k-1) - (1-\gamma) \alpha \mathbf{s}^m (\mathbf{a}^{m-1})^T$$

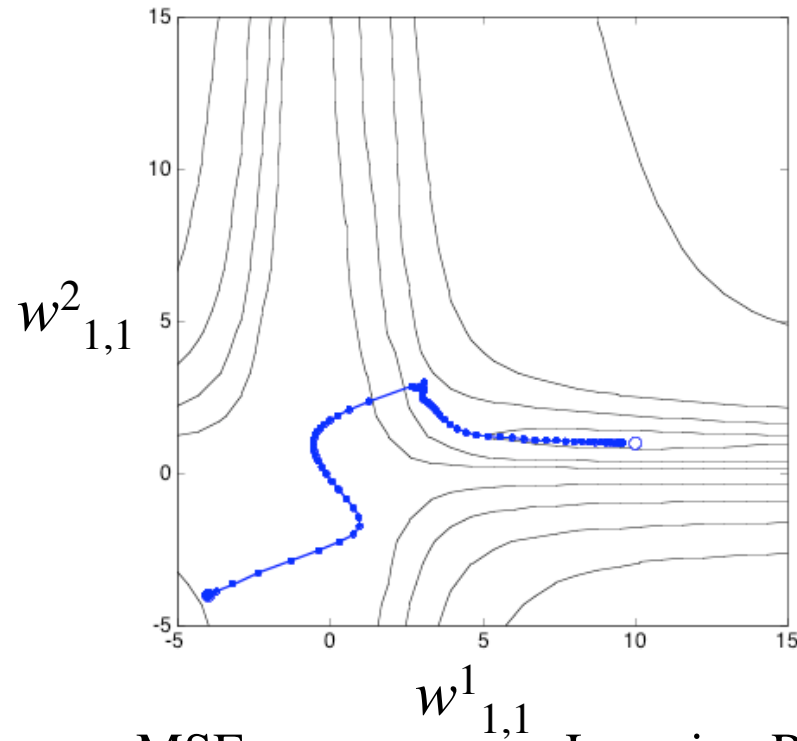
$$\Delta \mathbf{b}^m(k) = \gamma \Delta \mathbf{b}^m(k-1) - (1-\gamma) \alpha \mathbf{s}^m$$



# Variable Learning Rate (VLBP)

- If the squared error (over the entire training set) **increases by more than** some set percentage  $\zeta$  after a weight update, then:
  - the weight update is discarded,
  - the **learning rate is multiplied** by some factor ( $1 > \rho > 0$ ), and
  - the momentum coefficient  $\gamma$  is set to zero.
- If the squared error **decreases** after a weight update, then:
  - the weight update is accepted and
  - the **learning rate is multiplied** by some factor  $\eta > 1$ .
  - If  $\gamma$  has been previously set to zero, it is reset to its original value.
- **If the squared error increases by less than  $\zeta$** , then the weight update is accepted, but the learning rate and the momentum coefficient are unchanged.

# Example

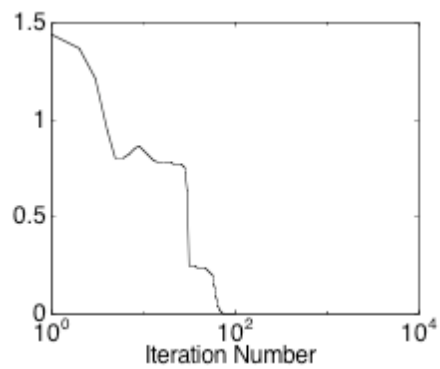


$$\eta = 1.05$$

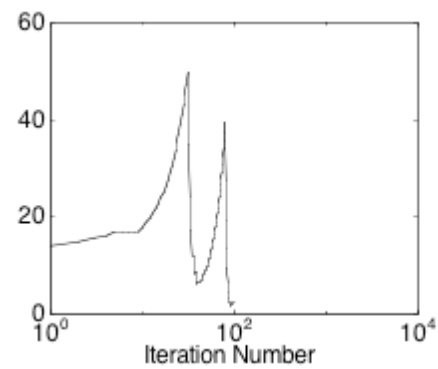
$$\rho = 0.7$$

$$\zeta = 4\%$$

MSE



Learning Rate



## Two methods for accelerating backpropagation

- Conjugate Gradient method
- Levenberg-Marquardt method

# Gradient Descent Review

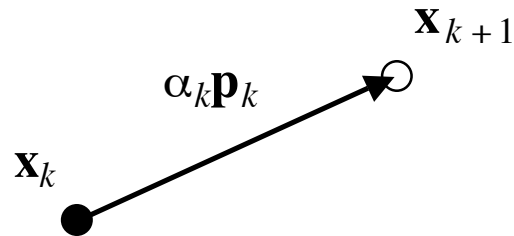
# Basic Optimization Algorithm

weight change

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k$$

or

$$\Delta \mathbf{x}_k = (\mathbf{x}_{k+1} - \mathbf{x}_k) = \alpha_k \mathbf{p}_k$$



$\mathbf{p}_k$  - Search Direction

$\alpha_k$  - Learning Rate

# Steepest Descent (Gradient Descent)

Choose the next step so that the function decreases:

$$F(\mathbf{x}_{k+1}) < F(\mathbf{x}_k)$$

For small changes in  $\mathbf{x}$  we can approximate  $F(\mathbf{x})$ :

$$F(\mathbf{x}_{k+1}) = F(\mathbf{x}_k + \Delta\mathbf{x}_k) \approx F(\mathbf{x}_k) + \mathbf{g}_k^T \Delta\mathbf{x}_k$$

where

$$\mathbf{g}_k \equiv \nabla F(\mathbf{x}) \Big|_{\mathbf{x} = \mathbf{x}_k}$$

If we want the function to decrease:

$$\mathbf{g}_k^T \Delta\mathbf{x}_k = \alpha_k \mathbf{g}_k^T \mathbf{p}_k < 0 \quad (\text{learning rate} * \text{gradient} * \text{direction})$$

We can maximize the decrease by choosing:

$$\mathbf{p}_k = -\mathbf{g}_k \quad (\text{direction} = \text{neg. gradient})$$

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k \mathbf{g}_k$$

# Example for an Analytic Function

$$F(\mathbf{x}) = x_1^2 + 2x_1x_2 + 2x_2^2 + x_1$$

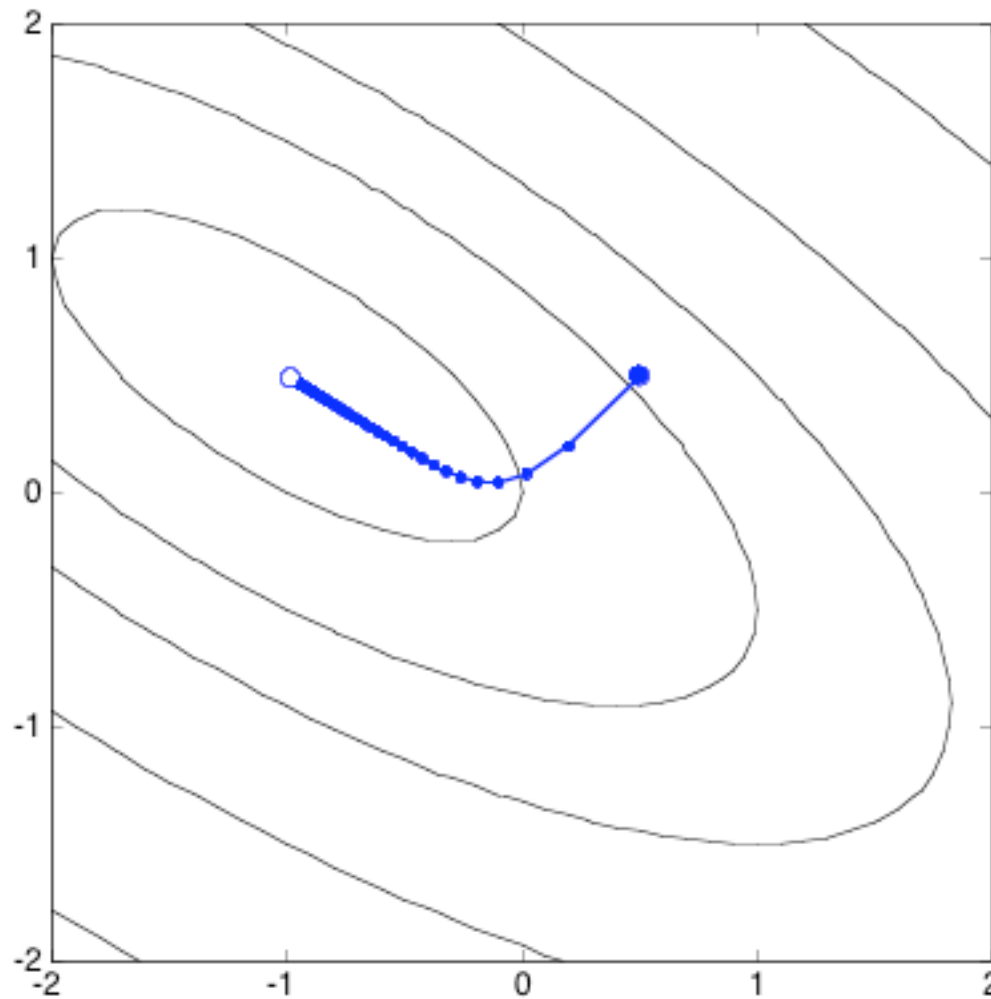
$$\mathbf{x}_0 = \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix} \quad \alpha = 0.1$$

$$\nabla F(\mathbf{x}) = \begin{bmatrix} \frac{\partial}{\partial x_1} F(\mathbf{x}) \\ \frac{\partial}{\partial x_2} F(\mathbf{x}) \end{bmatrix} = \begin{bmatrix} 2x_1 + 2x_2 + 1 \\ 2x_1 + 4x_2 \end{bmatrix} \quad \mathbf{g}_0 = \nabla F(\mathbf{x}) \Big|_{\mathbf{x} = \mathbf{x}_0} = \begin{bmatrix} 3 \\ 3 \end{bmatrix}$$

$$\mathbf{x}_1 = \mathbf{x}_0 - \alpha \mathbf{g}_0 = \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix} - 0.1 \begin{bmatrix} 3 \\ 3 \end{bmatrix} = \begin{bmatrix} 0.2 \\ 0.2 \end{bmatrix}$$

$$\mathbf{x}_2 = \mathbf{x}_1 - \alpha \mathbf{g}_1 = \begin{bmatrix} 0.2 \\ 0.2 \end{bmatrix} - 0.1 \begin{bmatrix} 1.8 \\ 1.2 \end{bmatrix} = \begin{bmatrix} 0.02 \\ 0.08 \end{bmatrix}$$

# Steepest Descent Plot



Note:  
Lots of  
*small* steps

# Conjugate Gradient Method

## Acceleration by Minimizing MSE Function *Along a Line*

Compute learning rate  $\alpha_k$  to minimize  $F(\mathbf{x}_k + \alpha_k \mathbf{p}_k)$   $\mathbf{p}_k$  is any chosen line direction, e.g.  $-\mathbf{g}_k$  (negative gradient)

By Taylor expansion:

$$\frac{d}{d\alpha_k}(F(\mathbf{x}_k + \alpha_k \mathbf{p}_k)) = \nabla F(\mathbf{x})^T \Big|_{\mathbf{x} = \mathbf{x}_k} \mathbf{p}_k + \alpha_k \mathbf{p}_k^T \nabla^2 F(\mathbf{x}) \Big|_{\mathbf{x} = \mathbf{x}_k} \mathbf{p}_k$$

Set derivative to 0 and **solve for**  $\alpha_k$  :

$$\alpha_k = - \frac{\nabla F(\mathbf{x})^T \Big|_{\mathbf{x} = \mathbf{x}_k} \mathbf{p}_k}{\mathbf{p}_k^T \nabla^2 F(\mathbf{x}) \Big|_{\mathbf{x} = \mathbf{x}_k} \mathbf{p}_k} = - \frac{\mathbf{g}_k^T \mathbf{p}_k}{\mathbf{p}_k^T \mathbf{A}_k \mathbf{p}_k} \quad \text{is value where derivative is 0}$$

where

$$\mathbf{A}_k \equiv \nabla^2 F(\mathbf{x}) \Big|_{\mathbf{x} = \mathbf{x}_k} \quad (\text{Hessian})$$

This is the **analytic** version, which assumes we know the Hessian, but we often don't. Later, we show how to minimize by search.

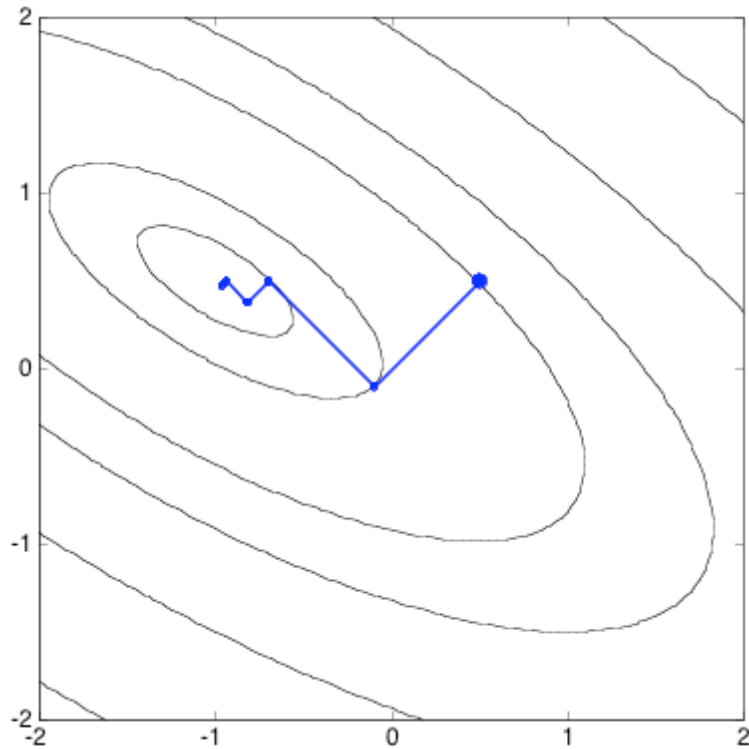
# Example

$$F(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T \begin{bmatrix} 2 & 2 \\ 2 & 4 \end{bmatrix} \mathbf{x} + [1 \ 0]\mathbf{x} \quad \mathbf{x}_0 = \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix}$$

$$\nabla F(\mathbf{x}) = \begin{bmatrix} \frac{\partial}{\partial x_1} F(\mathbf{x}) \\ \frac{\partial}{\partial x_2} F(\mathbf{x}) \end{bmatrix} = \begin{bmatrix} 2x_1 + 2x_2 + 1 \\ 2x_1 + 4x_2 \end{bmatrix} \quad \mathbf{p}_0 = -\mathbf{g}_0 = -\nabla F(\mathbf{x}) \Big|_{\mathbf{x} = \mathbf{x}_0} = \begin{bmatrix} -3 \\ -3 \end{bmatrix}$$

$$\alpha_0 = -\frac{\begin{bmatrix} 3 & 3 \end{bmatrix} \begin{bmatrix} -3 \\ -3 \end{bmatrix}}{\begin{bmatrix} -3 & -3 \end{bmatrix} \begin{bmatrix} 2 & 2 \\ 2 & 4 \end{bmatrix} \begin{bmatrix} -3 \\ -3 \end{bmatrix}} = 0.2 \quad \mathbf{x}_1 = \mathbf{x}_0 - \alpha_0 \mathbf{g}_0 = \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix} - 0.2 \begin{bmatrix} 3 \\ 3 \end{bmatrix} = \begin{bmatrix} -0.1 \\ -0.1 \end{bmatrix}$$

# Successive Line Minimizations with different directions



$$\begin{aligned}\frac{d}{d\alpha_k}F(\mathbf{x}_k + \alpha_k \mathbf{p}_k) &= \frac{d}{d\alpha_k}F(\mathbf{x}_{k+1}) = \nabla F(\mathbf{x})^T \Big|_{\mathbf{x} = \mathbf{x}_{k+1}} \frac{d}{d\alpha_k}[\mathbf{x}_k + \alpha_k \mathbf{p}_k] \\ &= \nabla F(\mathbf{x})^T \Big|_{\mathbf{x} = \mathbf{x}_{k+1}} \mathbf{p}_k = \mathbf{g}_{k+1}^T \mathbf{p}_k\end{aligned}$$

# Conjugate Vectors

$$F(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T \mathbf{A}\mathbf{x} + \mathbf{d}^T \mathbf{x} + c$$

A set of vectors  $\mathbf{p}_i$  is mutually *conjugate* with respect to a positive definite Hessian matrix  $\mathbf{A}$  provided

$$\mathbf{p}_k^T \mathbf{A} \mathbf{p}_j = 0 \quad k \neq j$$

One set of conjugate vectors consists of the eigenvectors of  $\mathbf{A}$ .

$$\mathbf{z}_k^T \mathbf{A} \mathbf{z}_j = \lambda_j \mathbf{z}_k^T \mathbf{z}_j = 0 \quad k \neq j$$

(The eigenvectors of symmetric matrices are orthogonal.)

# For *Quadratic* Functions

$$\nabla F(\mathbf{x}) = \mathbf{Ax} + \mathbf{d}$$

$$\nabla^2 F(\mathbf{x}) = \mathbf{A}$$

The change in the gradient at iteration  $k$  is

$$\Delta \mathbf{g}_k = \mathbf{g}_{k+1} - \mathbf{g}_k = (\mathbf{Ax}_{k+1} + \mathbf{d}) - (\mathbf{Ax}_k + \mathbf{d}) = \mathbf{A}\Delta \mathbf{x}_k$$

where

$$\Delta \mathbf{x}_k = (\mathbf{x}_{k+1} - \mathbf{x}_k) = \alpha_k \mathbf{p}_k$$

The conjugacy conditions can be rewritten

$$\alpha_k \mathbf{p}_k^T \mathbf{A} \mathbf{p}_j = \Delta \mathbf{x}_k^T \mathbf{A} \mathbf{p}_j = \Delta \mathbf{g}_k^T \mathbf{p}_j = 0 \quad k \neq j$$

the last term not requiring knowledge of the Hessian matrix  $\mathbf{A}$ .

# Forming *Conjugate* Directions

Choose the initial search direction as the negative of the gradient:

$$\mathbf{p}_0 = -\mathbf{g}_0$$

Choose subsequent search directions to be *conjugate*:

$$\mathbf{p}_k = -\mathbf{g}_k + \beta_k \mathbf{p}_{k-1}$$

where  $\beta_k$  is chosen according to *one of these formulae*:

$$\beta_k = \frac{\Delta \mathbf{g}_{k-1}^T \mathbf{g}_k}{\Delta \mathbf{g}_{k-1}^T \mathbf{p}_{k-1}} \quad \mathbf{or} \quad \beta_k = \frac{\mathbf{g}_k^T \mathbf{g}_k}{\mathbf{g}_{k-1}^T \mathbf{g}_{k-1}} \quad \mathbf{or} \quad \beta_k = \frac{\Delta \mathbf{g}_{k-1}^T \mathbf{g}_k}{\mathbf{g}_{k-1}^T \mathbf{g}_{k-1}}$$

Hestnes &  
Steffel

Fletcher-  
Reeves

Polak &  
Ribiere

# Conjugate Gradient algorithm

- The first search direction is the negative of the gradient.

$$\mathbf{p}_0 = -\mathbf{g}_0$$

- Select the learning rate to minimize along the line.

$$\alpha_k = -\frac{\nabla F(\mathbf{x})^T \Big|_{\mathbf{x}=\mathbf{x}_k} \mathbf{p}_k}{\mathbf{p}_k^T \nabla^2 F(\mathbf{x}) \Big|_{\mathbf{x}=\mathbf{x}_k} \mathbf{p}_k} = -\frac{\mathbf{g}_k^T \mathbf{p}_k}{\mathbf{p}_k^T \mathbf{A}_k \mathbf{p}_k} \quad (\text{e.g. for quadratic functions only.})$$

- Select the next search direction using

$$\mathbf{p}_k = -\mathbf{g}_k + \beta_k \mathbf{p}_{k-1}$$

- If the algorithm has not converged, return to second step.
- If the function were quadratic, it would be minimized in  $n$  steps, where  $n$  is the number of dimensions.

## Example Analytic Solution, Quadratic

$$F(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T \begin{bmatrix} 2 & 2 \\ 2 & 4 \end{bmatrix} \mathbf{x} + [1 \ 0]\mathbf{x} \quad \mathbf{x}_0 = \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix}$$

$$\nabla F(\mathbf{x}) = \begin{bmatrix} \frac{\partial}{\partial x_1} F(\mathbf{x}) \\ \frac{\partial}{\partial x_2} F(\mathbf{x}) \end{bmatrix} = \begin{bmatrix} 2x_1 + 2x_2 + 1 \\ 2x_1 + 4x_2 \end{bmatrix} \quad \mathbf{p}_0 = -\mathbf{g}_0 = -\nabla F(\mathbf{x}) \Big|_{\mathbf{x} = \mathbf{x}_0} = \begin{bmatrix} -3 \\ -3 \end{bmatrix}$$

$$\alpha_0 = -\frac{\begin{bmatrix} 3 & 3 \end{bmatrix} \begin{bmatrix} -3 \\ -3 \end{bmatrix}}{\begin{bmatrix} -3 & -3 \end{bmatrix} \begin{bmatrix} 2 & 2 \\ 2 & 4 \end{bmatrix} \begin{bmatrix} -3 \\ -3 \end{bmatrix}} = 0.2 \quad \mathbf{x}_1 = \mathbf{x}_0 - \alpha_0 \mathbf{g}_0 = \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix} - 0.2 \begin{bmatrix} 3 \\ 3 \end{bmatrix} = \begin{bmatrix} -0.1 \\ -0.1 \end{bmatrix}$$

# Example

$$\mathbf{g}_1 = \nabla F(\mathbf{x}) \Big|_{\mathbf{x} = \mathbf{x}_1} = \begin{bmatrix} 2 & 2 \\ 2 & 4 \end{bmatrix} \begin{bmatrix} -0.1 \\ -0.1 \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0.6 \\ -0.6 \end{bmatrix}$$

$$\beta_1 = \frac{\mathbf{g}_1^T \mathbf{g}_1}{\mathbf{g}_0^T \mathbf{g}_0} = \frac{\begin{bmatrix} 0.6 & -0.6 \end{bmatrix} \begin{bmatrix} 0.6 \\ -0.6 \end{bmatrix}}{\begin{bmatrix} 3 & 3 \end{bmatrix} \begin{bmatrix} 3 \\ 3 \end{bmatrix}} = \frac{0.72}{18} = 0.04$$

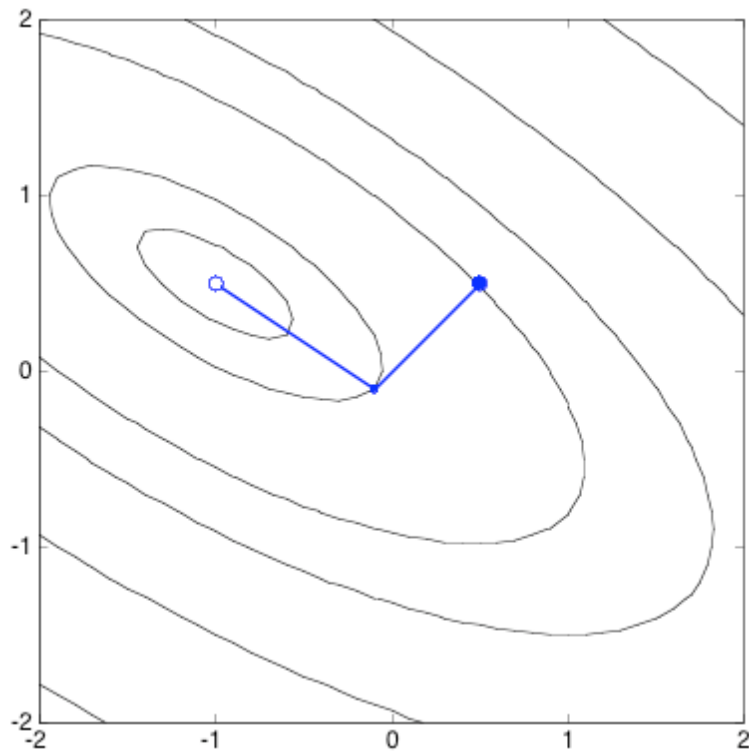
$$\mathbf{p}_1 = -\mathbf{g}_1 + \beta_1 \mathbf{p}_0 = \begin{bmatrix} -0.6 \\ 0.6 \end{bmatrix} + 0.04 \begin{bmatrix} -3 \\ -3 \end{bmatrix} = \begin{bmatrix} -0.72 \\ 0.48 \end{bmatrix}$$

$$\alpha_1 = -\frac{\begin{bmatrix} 0.6 & -0.6 \end{bmatrix} \begin{bmatrix} -0.72 \\ 0.48 \end{bmatrix}}{\begin{bmatrix} -0.72 & 0.48 \end{bmatrix} \begin{bmatrix} 2 & 2 \\ 2 & 4 \end{bmatrix} \begin{bmatrix} -0.72 \\ 0.48 \end{bmatrix}} = -\frac{-0.72}{0.576} = 1.25$$

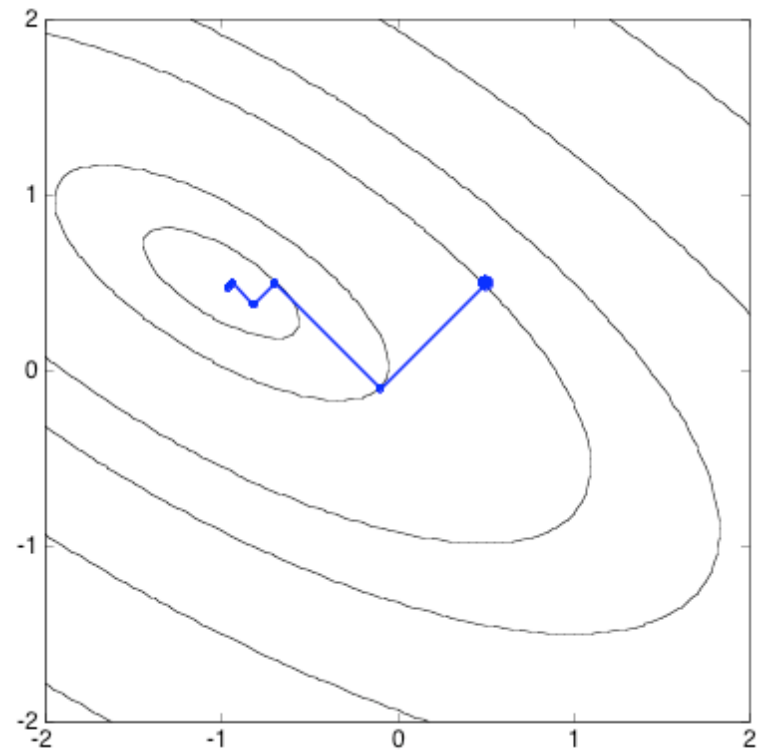
## Conjugate Gradient vs. Steepest Descent Plots

$$\mathbf{x}_2 = \mathbf{x}_1 + \alpha_1 \mathbf{p}_1 = \begin{bmatrix} -0.1 \\ -0.1 \end{bmatrix} + 1.25 \begin{bmatrix} -0.72 \\ 0.48 \end{bmatrix} = \begin{bmatrix} -1 \\ 0.5 \end{bmatrix}$$

### Conjugate Gradient



### Steepest Descent

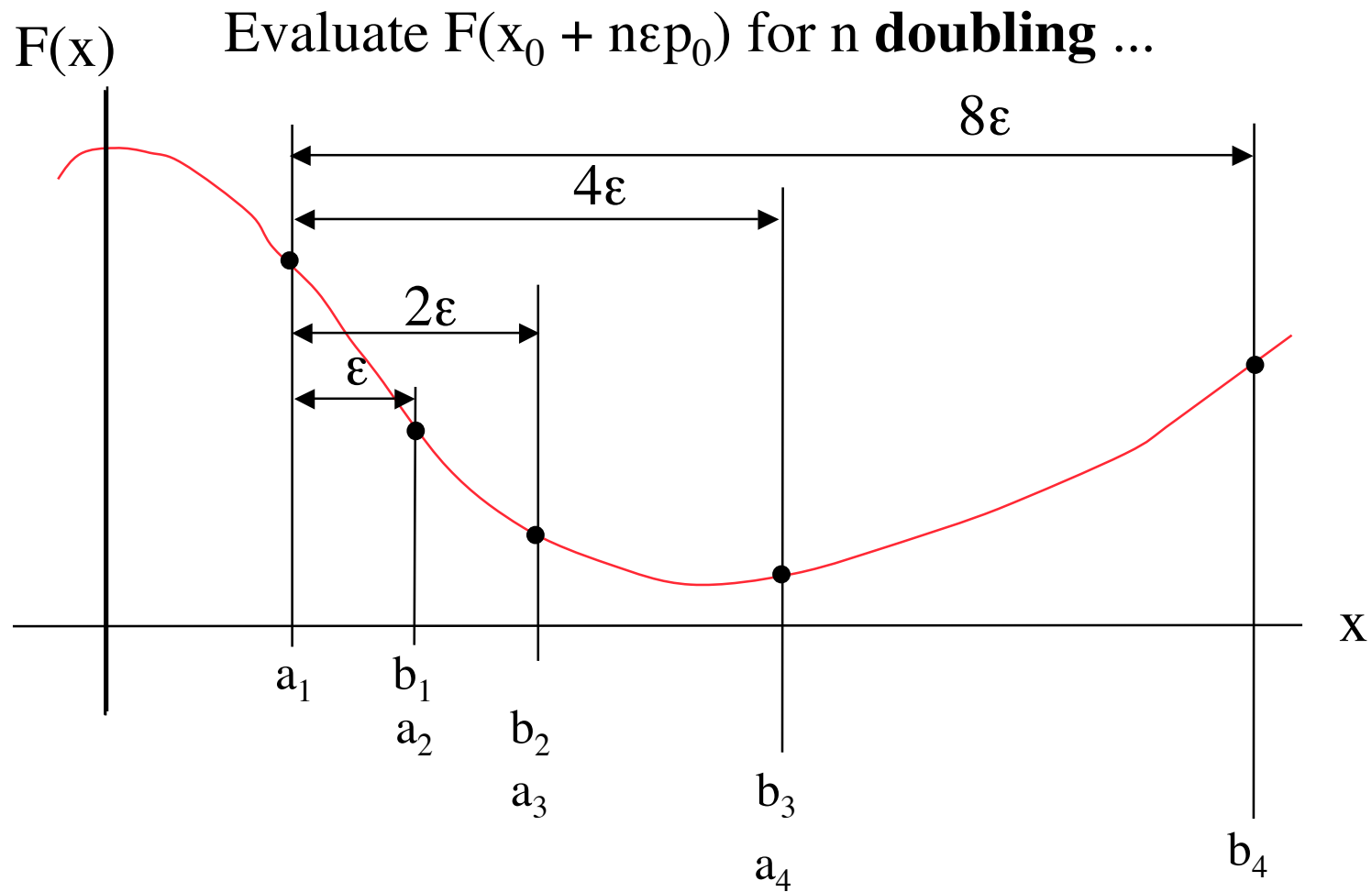


**Heuristic for Line Minimization  
for General Functions  
(not necessarily quadratic)**

# Numerical Line Minimization

- Part 1: **Locate** an **interval** containing the minimum.
- Part 2: **Reduce** the interval's width successively, until the interval is sufficiently small that we are close enough to the minimum.

## Part 1: Interval Location to Bracket Minimum

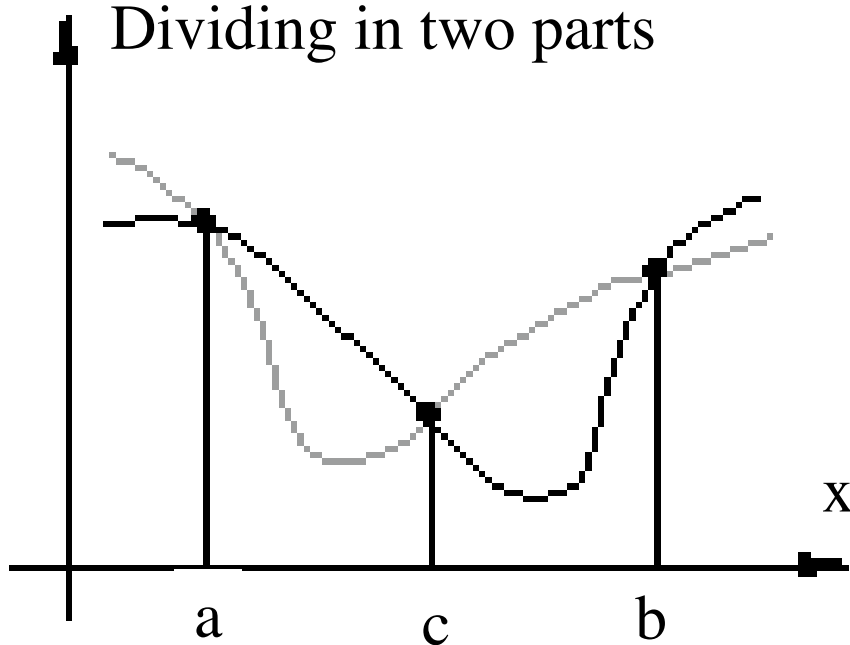


Stop when two successive increases occur.

## Part 2: Interval Reduction

What doesn't help:

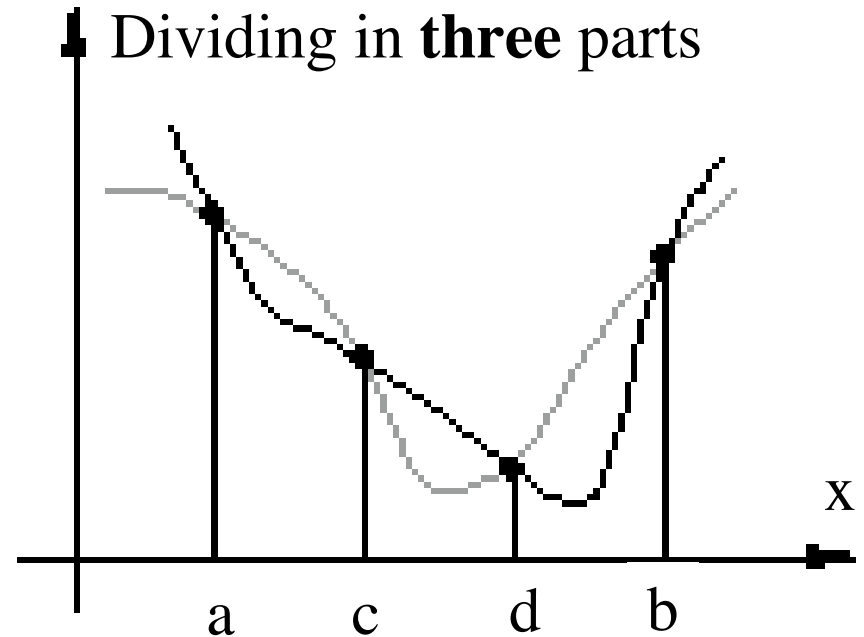
Dividing in two parts



Interval **not reduced**.

What does:

Dividing in **three** parts



$c > d \Rightarrow$  minimum is  
not between  $a$  and  $c$   
 $\Rightarrow$  **eliminate**  $a-c$ .

If  $d > c$ , eliminate  $d-b$ .

# Optimization of Function Evaluations: Fibonacci Search

$$\tau = 0.618$$

$$\text{Set } c_1 = a_1 + (1-\tau)(b_1 - a_1), F_c = F(c_1)$$

$$d_1 = b_1 - (1-\tau)(b_1 - a_1), F_d = F(d_1)$$

For  $k=1,2, \dots$  repeat

If  $F_c < F_d$  then

$$\text{Set } a_{k+1} = a_k; b_{k+1} = d_k; d_{k+1} = c_k$$

$$c_{k+1} = a_{k+1} + (1-\tau)(b_{k+1} - a_{k+1})$$

$$F_d = F_c; F_c = F(c_{k+1})$$

else

$$\text{Set } a_{k+1} = c_k; b_{k+1} = b_k; c_{k+1} = d_k$$

$$d_{k+1} = b_{k+1} - (1-\tau)(b_{k+1} - a_{k+1})$$

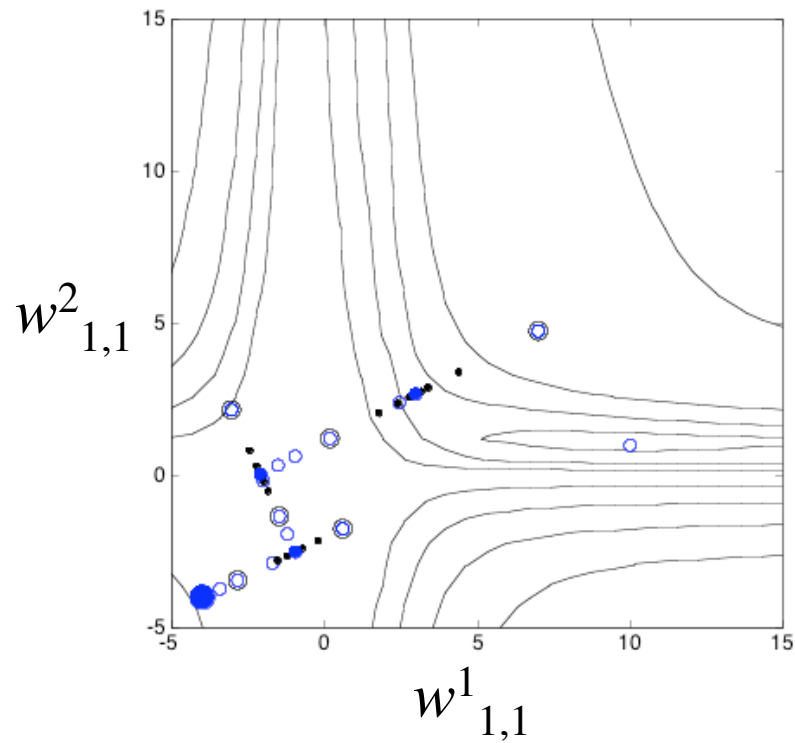
$$F_c = F_d; F_d = F(d_{k+1})$$

end

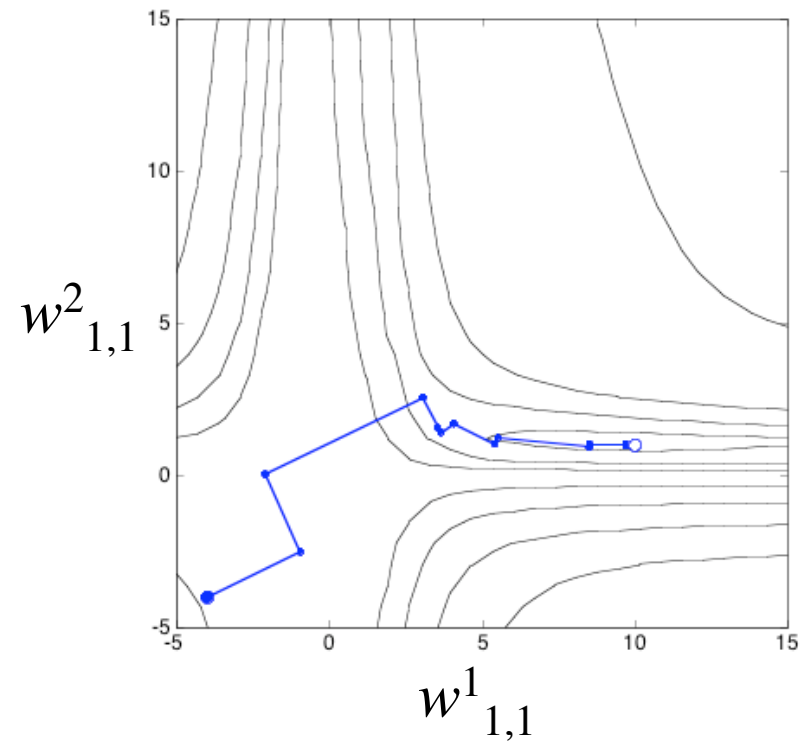
end until  $b_{k+1} - a_{k+1} < tol$

# Conjugate Gradient BP (CGBP)

## Intermediate Steps



## Complete Trajectory



# Note on Conjugate Gradient

- CG is a **batch-mode** algorithm:
  - Average the gradient over all samples.

## Newton's Method for Locating a Minimum

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \mathbf{A}_k^{-1} \mathbf{g}_k$$

$$\mathbf{A}_k \equiv \nabla^2 F(\mathbf{x}) \Big|_{\mathbf{x} = \mathbf{x}_k} \quad \mathbf{g}_k \equiv \nabla F(\mathbf{x}) \Big|_{\mathbf{x} = \mathbf{x}_k}$$

If the performance index is a sum of squares function:

$$F(\mathbf{x}) = \sum_{i=1}^N v_i^2(\mathbf{x}) = \mathbf{v}^T(\mathbf{x}) \mathbf{v}(\mathbf{x})$$

then the  $j$ th element of the gradient is

$$[\nabla F(\mathbf{x})]_j = \frac{\partial F(\mathbf{x})}{\partial x_j} = 2 \sum_{i=1}^N v_i(\mathbf{x}) \frac{\partial v_i(\mathbf{x})}{\partial x_j}$$

# Newton's Method

$$F(\mathbf{x}_{k+1}) = F(\mathbf{x}_k + \Delta\mathbf{x}_k) \approx F(\mathbf{x}_k) + \mathbf{g}_k^T \Delta\mathbf{x}_k + \frac{1}{2} \Delta\mathbf{x}_k^T \mathbf{A}_k \Delta\mathbf{x}_k$$

Take the gradient of this second-order approximation and set it equal to zero to find the stationary point:

$$\mathbf{g}_k + \mathbf{A}_k \Delta\mathbf{x}_k = \mathbf{0}$$

$$\Delta\mathbf{x}_k = -\mathbf{A}_k^{-1} \mathbf{g}_k$$

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \mathbf{A}_k^{-1} \mathbf{g}_k$$

# Example

$$F(\mathbf{x}) = x_1^2 + 2x_1x_2 + 2x_2^2 + x_1$$

$$\mathbf{x}_0 = \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix}$$

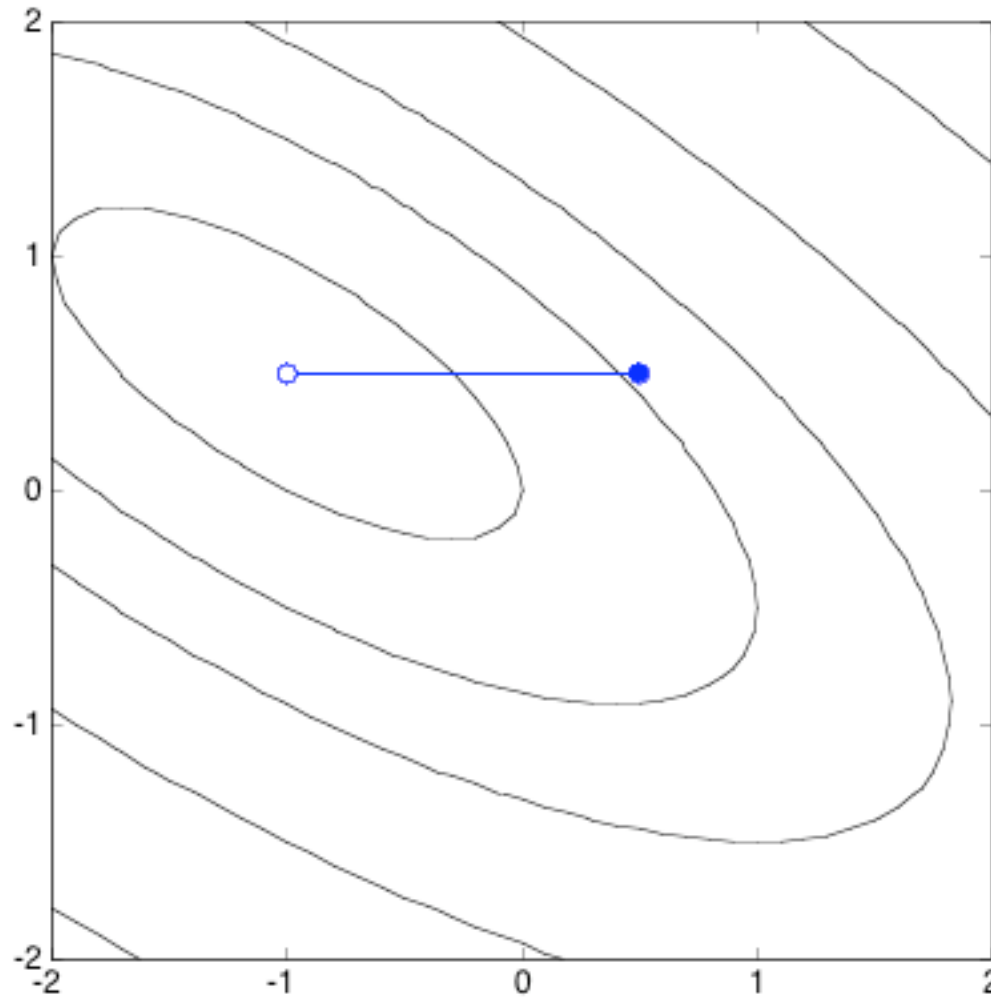
$$\nabla F(\mathbf{x}) = \begin{bmatrix} \frac{\partial}{\partial x_1} F(\mathbf{x}) \\ \frac{\partial}{\partial x_2} F(\mathbf{x}) \end{bmatrix} = \begin{bmatrix} 2x_1 + 2x_2 + 1 \\ 2x_1 + 4x_2 \end{bmatrix}$$

$$\mathbf{g}_0 = \nabla F(\mathbf{x}) \Big|_{\mathbf{x} = \mathbf{x}_0} = \begin{bmatrix} 3 \\ 3 \end{bmatrix}$$

$$\mathbf{A} = \begin{bmatrix} 2 & 2 \\ 2 & 4 \end{bmatrix}$$

$$\mathbf{x}_1 = \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix} - \begin{bmatrix} 2 & 2 \\ 2 & 4 \end{bmatrix}^{-1} \begin{bmatrix} 3 \\ 3 \end{bmatrix} = \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix} - \begin{bmatrix} 1 & -0.5 \\ -0.5 & 0.5 \end{bmatrix} \begin{bmatrix} 3 \\ 3 \end{bmatrix} = \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix} - \begin{bmatrix} 1.5 \\ 0 \end{bmatrix} = \begin{bmatrix} -1 \\ 0.5 \end{bmatrix}$$

# Plot of Newton's Method for a Quadratic

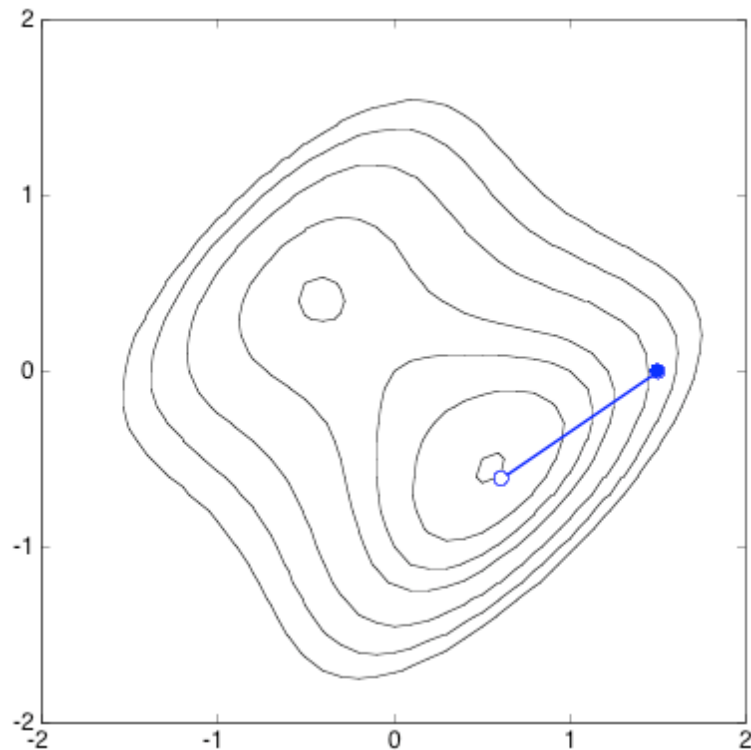


# Non-Quadratic Example

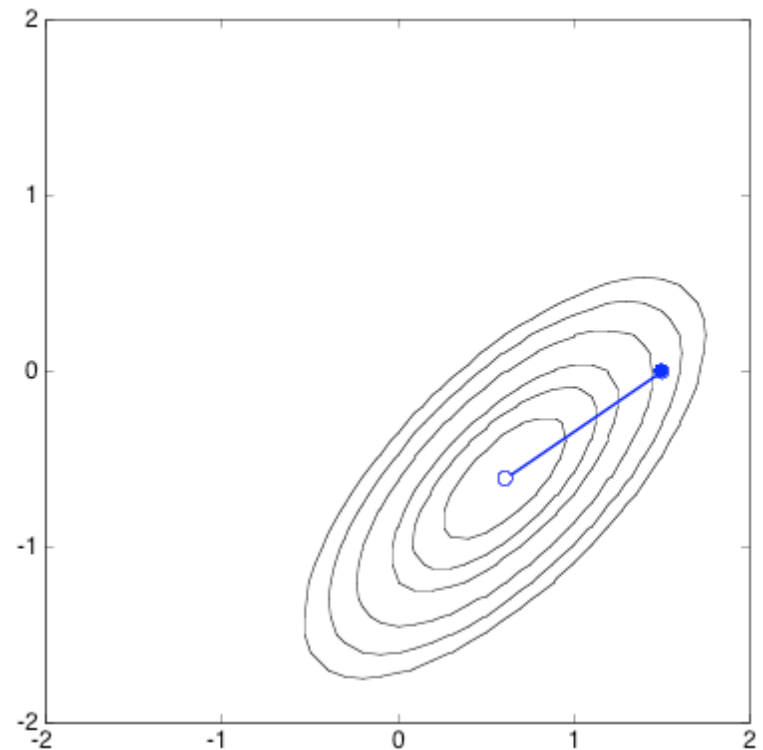
$$F(\mathbf{x}) = (x_2 - x_1)^4 + 8x_1x_2 - x_1 + x_2 + 3$$

Stationary Points:  $\mathbf{x}^1 = \begin{bmatrix} -0.42 \\ 0.42 \end{bmatrix}$        $\mathbf{x}^2 = \begin{bmatrix} -0.13 \\ 0.13 \end{bmatrix}$        $\mathbf{x}^3 = \begin{bmatrix} 0.55 \\ -0.55 \end{bmatrix}$

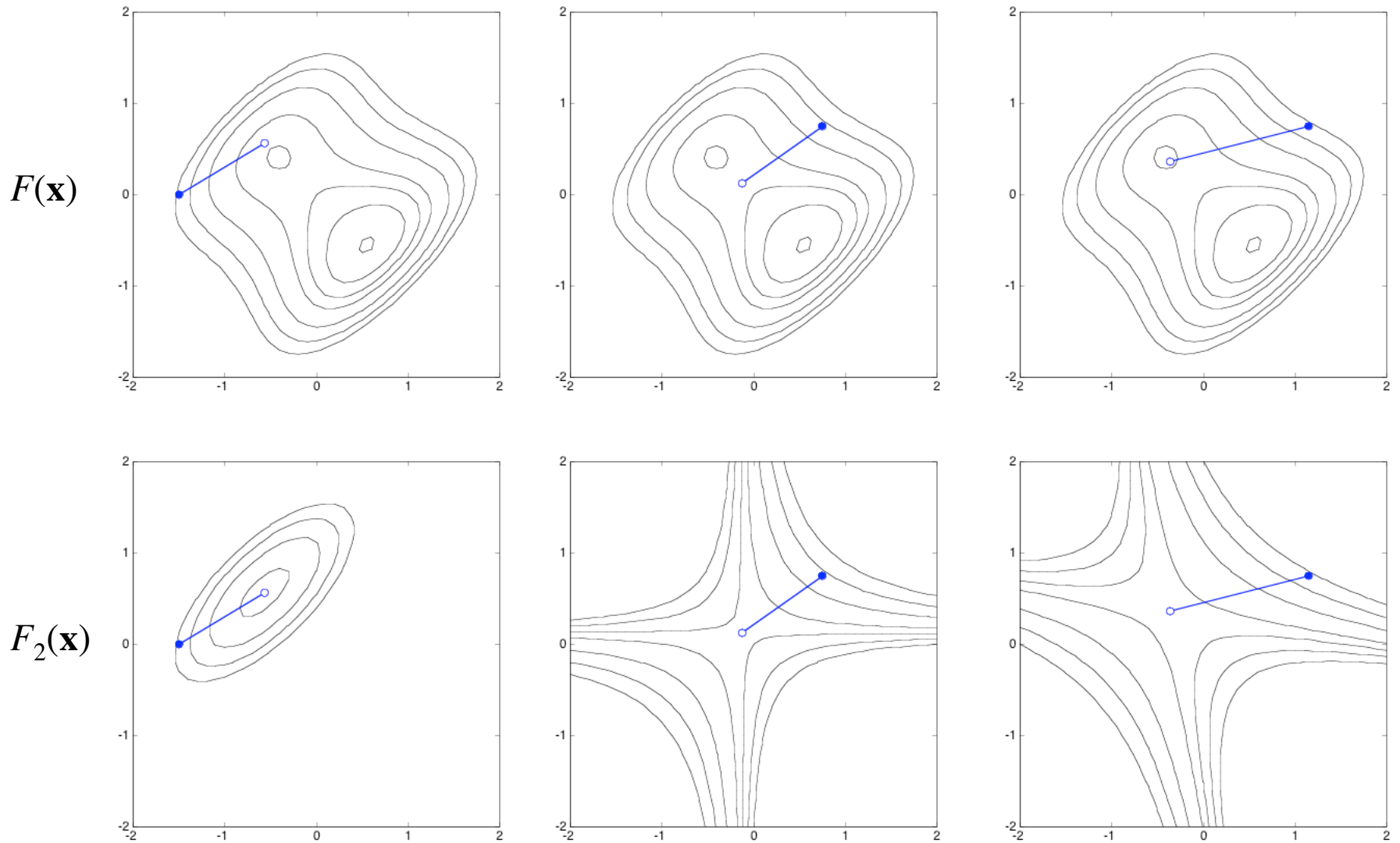
$F(\mathbf{x})$



$F_2(\mathbf{x})$ , Quadratic Approximation



# Different Initial Conditions and Quadratic Approximations to Each



# Levenberg-Marquardt Method

(blends Newton's method with  
steepest descent)

Perhaps the fastest known method for training  
(but storage intensive)

# Matrix Form

The gradient can be written in matrix form as a matrix-vector product:

$$\nabla F(\mathbf{x}) = 2\mathbf{J}^T(\mathbf{x})\mathbf{v}(\mathbf{x})$$

where  $\mathbf{J}$  is the Jacobian matrix:

$$\mathbf{J}(\mathbf{x}) = \begin{bmatrix} \frac{\partial v_1(\mathbf{x})}{\partial x_1} & \frac{\partial v_1(\mathbf{x})}{\partial x_2} & \cdots & \frac{\partial v_1(\mathbf{x})}{\partial x_n} \\ \frac{\partial v_2(\mathbf{x})}{\partial x_1} & \frac{\partial v_2(\mathbf{x})}{\partial x_2} & \cdots & \frac{\partial v_2(\mathbf{x})}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial v_N(\mathbf{x})}{\partial x_1} & \frac{\partial v_N(\mathbf{x})}{\partial x_2} & \cdots & \frac{\partial v_N(\mathbf{x})}{\partial x_n} \end{bmatrix}$$

$\mathbf{v}$  are the  
error values  
for each component  
of each sample

Express Hessian, which represents Curvature,  
in terms of Jacobian:

$$[\nabla^2 F(\mathbf{x})]_{k,j} = \frac{\partial^2 F(\mathbf{x})}{\partial x_k \partial x_j} = 2 \sum_{i=1}^N \left\{ \frac{\partial v_i(\mathbf{x})}{\partial x_k} \frac{\partial v_i(\mathbf{x})}{\partial x_j} + v_i(\mathbf{x}) \frac{\partial^2 v_i(\mathbf{x})}{\partial x_k \partial x_j} \right\}$$

$$\nabla^2 F(\mathbf{x}) = 2\mathbf{J}^T(\mathbf{x})\mathbf{J}(\mathbf{x}) + 2\mathbf{S}(\mathbf{x})$$

where  $\mathbf{S}(\mathbf{x}) = \sum_{i=1}^N v_i(\mathbf{x}) \nabla^2 v_i(\mathbf{x})$

# Gauss-Newton Method

**Approximate** the Hessian matrix as:

$$\nabla^2 F(\mathbf{x}) \cong 2\mathbf{J}^T(\mathbf{x})\mathbf{J}(\mathbf{x})$$

Newton's method becomes:

$$\begin{aligned}\mathbf{x}_{k+1} &= \mathbf{x}_k - [2\mathbf{J}^T(\mathbf{x}_k)\mathbf{J}(\mathbf{x}_k)]^{-1} 2\mathbf{J}^T(\mathbf{x}_k)\mathbf{v}(\mathbf{x}_k) \\ &= \mathbf{x}_k - [\mathbf{J}^T(\mathbf{x}_k)\mathbf{J}(\mathbf{x}_k)]^{-1} \mathbf{J}^T(\mathbf{x}_k)\mathbf{v}(\mathbf{x}_k)\end{aligned}$$

This method does not require calculating second derivatives.

# Levenberg-Marquardt

Gauss-Newton approximates the Hessian by:

$$\mathbf{H} = \mathbf{J}^T \mathbf{J}$$

This matrix may be **singular**, but can be made invertible as follows:

$$\mathbf{G} = \mathbf{H} + \mu \mathbf{I}$$

If the eigenvalues and eigenvectors of  $\mathbf{H}$  are:

$$\{\lambda_1, \lambda_2, \dots, \lambda_n\}$$

$$\{\mathbf{z}_1, \mathbf{z}_2, \dots, \mathbf{z}_n\}$$

then

Eigenvalues of  $\mathbf{G}$

$$\mathbf{G}\mathbf{z}_i = [\mathbf{H} + \mu\mathbf{I}]\mathbf{z}_i = \mathbf{H}\mathbf{z}_i + \mu\mathbf{z}_i = \lambda_i\mathbf{z}_i + \mu\mathbf{z}_i = \underbrace{(\lambda_i + \mu)}_{\text{Eigenvalues of } \mathbf{G}}\mathbf{z}_i$$

$$\mathbf{x}_{k+1} = \mathbf{x}_k - [\mathbf{J}^T(\mathbf{x}_k)\mathbf{J}(\mathbf{x}_k) + \mu_k\mathbf{I}]^{-1} \mathbf{J}^T(\mathbf{x}_k)\mathbf{v}(\mathbf{x}_k)$$

# Adjustment of $\mu_k$

As  $\mu_k \rightarrow 0$ , LM becomes Gauss-Newton.

$$\mathbf{x}_{k+1} = \mathbf{x}_k - [\mathbf{J}^T(\mathbf{x}_k)\mathbf{J}(\mathbf{x}_k)]^{-1} \mathbf{J}^T(\mathbf{x}_k)\mathbf{v}(\mathbf{x}_k)$$

As  $\mu_k \rightarrow \infty$ , LM becomes Steepest Descent with small learning rate.

$$\mathbf{x}_{k+1} \cong \mathbf{x}_k - \frac{1}{\mu_k} \mathbf{J}^T(\mathbf{x}_k)\mathbf{v}(\mathbf{x}_k) = \mathbf{x}_k - \frac{1}{2\mu_k} \nabla F(\mathbf{x})$$

# Adjustment of $\mu_k$

- **Steepest Descent is the default position:**  
Begin with a small  $\mu_k$  to use Gauss-Newton.
- If a step does not yield a smaller  $F(\mathbf{x})$ , then repeat the step with an **increased**  $\mu_k$  until  $F(\mathbf{x})$  is decreased.
- $F(\mathbf{x})$  *must* decrease eventually, since we will eventually be taking a very **small** step in the steepest descent direction.

# Application to Multilayer Network

The performance index for the multilayer network is:

$$F(\mathbf{x}) = \sum_{q=1}^Q (\mathbf{t}_q - \mathbf{a}_q)^T (\mathbf{t}_q - \mathbf{a}_q) = \sum_{q=1}^Q \mathbf{e}_q^T \mathbf{e}_q = \sum_{q=1}^Q \sum_{j=1}^{S^M} (e_{j,q})^2 = \sum_{i=1}^N (v_i)^2$$

The error vector is:

$$\mathbf{v}^T = [v_1 \ v_2 \ \dots \ v_N] = [e_{1,1} \ e_{2,1} \ \dots \ e_{S^M,1} \ e_{1,2} \ \dots \ e_{S^M,Q}]$$

$S^M =$   
number of  
outputs,  
 $Q =$  number  
of samples

The parameter vector is:

$$\mathbf{x}^T = [x_1 \ x_2 \ \dots \ x_n] = [w_{1,1}^1 \ w_{1,2}^1 \ \dots \ w_{S^1,R}^1 \ b_1^1 \ \dots \ b_{S^1}^1 \ w_{1,1}^2 \ \dots \ b_{S^M}^M]$$

The dimensions of the two vectors are:

$$N = Q \times S^M \quad n = S^1(R+1) + S^2(S^1+1) + \dots + S^M(S^{M-1}+1)$$

# Jacobian Matrix for Levenberg-Marquardt

$$\mathbf{J}(\mathbf{x}) = \begin{bmatrix}
 \frac{\partial e_{1,1}}{\partial w_{1,1}^1} & \frac{\partial e_{1,1}}{\partial w_{1,2}^1} & \dots & \frac{\partial e_{1,1}}{\partial w_{S^1,R}^1} & \frac{\partial e_{1,1}}{\partial b_1^1} & \dots \\
 \frac{\partial e_{2,1}}{\partial w_{1,1}^1} & \frac{\partial e_{2,1}}{\partial w_{1,2}^1} & \dots & \frac{\partial e_{2,1}}{\partial w_{S^1,R}^1} & \frac{\partial e_{2,1}}{\partial b_1^1} & \dots \\
 \vdots & \vdots & & \vdots & \vdots & \\
 \frac{\partial e_{S^M,1}}{\partial w_{1,1}^1} & \frac{\partial e_{S^M,1}}{\partial w_{1,2}^1} & \dots & \frac{\partial e_{S^M,1}}{\partial w_{S^1,R}^1} & \frac{\partial e_{S^M,1}}{\partial b_1^1} & \dots \\
 \frac{\partial e_{1,2}}{\partial w_{1,1}^1} & \frac{\partial e_{1,2}}{\partial w_{1,2}^1} & \dots & \frac{\partial e_{1,2}}{\partial w_{S^1,R}^1} & \frac{\partial e_{1,2}}{\partial b_1^1} & \dots \\
 \vdots & \vdots & & \vdots & \vdots & 
 \end{bmatrix}$$

M rows for every input sample

Repeated Q times

# Computing the Jacobian

Steepest descent computes terms of the form:

$$\frac{\partial \hat{F}(\mathbf{x})}{\partial x_l} = \frac{\partial \mathbf{e}_q^T \mathbf{e}_q}{\partial x_l}$$

using the chain rule:

$$\frac{\partial \hat{F}}{\partial w_{i,j}^m} = \frac{\partial \hat{F}}{\partial n_i^m} \times \frac{\partial n_i^m}{\partial w_{i,j}^m}$$

where the sensitivity

$$s_i^m \equiv \frac{\partial \hat{F}}{\partial n_i^m}$$

is computed using backpropagation.

For the Jacobian we need to compute terms of the form:

$$[\mathbf{J}]_{h,l} = \frac{\partial v_h}{\partial x_l} = \frac{\partial e_{k,q}}{\partial x_l}$$

# Marquardt Sensitivity

If we define a Marquardt sensitivity (q is the sample's index):

$$\tilde{s}_{i,h}^m \equiv \frac{\partial v_h}{\partial n_{i,q}^m} = \frac{\partial e_{k,q}}{\partial n_{i,q}^m} \quad h = (q-1)S^M + k$$

we can compute the Jacobian as follows:

weight

$$[\mathbf{J}]_{h,l} = \frac{\partial v_h}{\partial x_l} = \frac{\partial e_{k,q}}{\partial w_{i,j}^m} = \frac{\partial e_{k,q}}{\partial n_{i,q}^m} \times \frac{\partial n_{i,q}^m}{\partial w_{i,j}^m} = \tilde{s}_{i,h}^m \times \frac{\partial n_{i,q}^m}{\partial w_{i,j}^m} = \tilde{s}_{i,h}^m \times a_{j,q}^{m-1}$$

bias

$$[\mathbf{J}]_{h,l} = \frac{\partial v_h}{\partial x_l} = \frac{\partial e_{k,q}}{\partial b_i^m} = \frac{\partial e_{k,q}}{\partial n_{i,q}^m} \times \frac{\partial n_{i,q}^m}{\partial b_i^m} = \tilde{s}_{i,h}^m \times \frac{\partial n_{i,q}^m}{\partial b_i^m} = \tilde{s}_{i,h}^m$$

# Computing the Sensitivities

## Initialization

$$\tilde{s}_{i,h}^M = \frac{\partial v_h}{\partial n_{i,q}^M} = \frac{\partial e_{k,q}}{\partial n_{i,q}^M} = \frac{\partial (t_{k,q} - a_{k,q}^M)}{\partial n_{i,q}^M} = -\frac{\partial a_{k,q}^M}{\partial n_{i,q}^M}$$

$$\tilde{s}_{i,h}^M = \begin{cases} -f^{iM}(n_{i,q}^M) & \text{for } i = k \\ 0 & \text{for } i \neq k \end{cases}$$

$$\tilde{\mathbf{S}}_q^M = -\dot{\mathbf{F}}^M(\mathbf{n}_q^M)$$

## Backpropagation

$$\tilde{\mathbf{S}}_q^m = \dot{\mathbf{F}}^m(\mathbf{n}_q^m)(\mathbf{W}^{m+1})^T \tilde{\mathbf{S}}_q^{m+1}$$

$$\tilde{\mathbf{S}}^m = \left[ \tilde{\mathbf{S}}_1^m \mid \tilde{\mathbf{S}}_2^m \mid \cdots \mid \tilde{\mathbf{S}}_Q^m \right]$$

Q = number of samples

# Levenberg-Marquardt Backpropagation

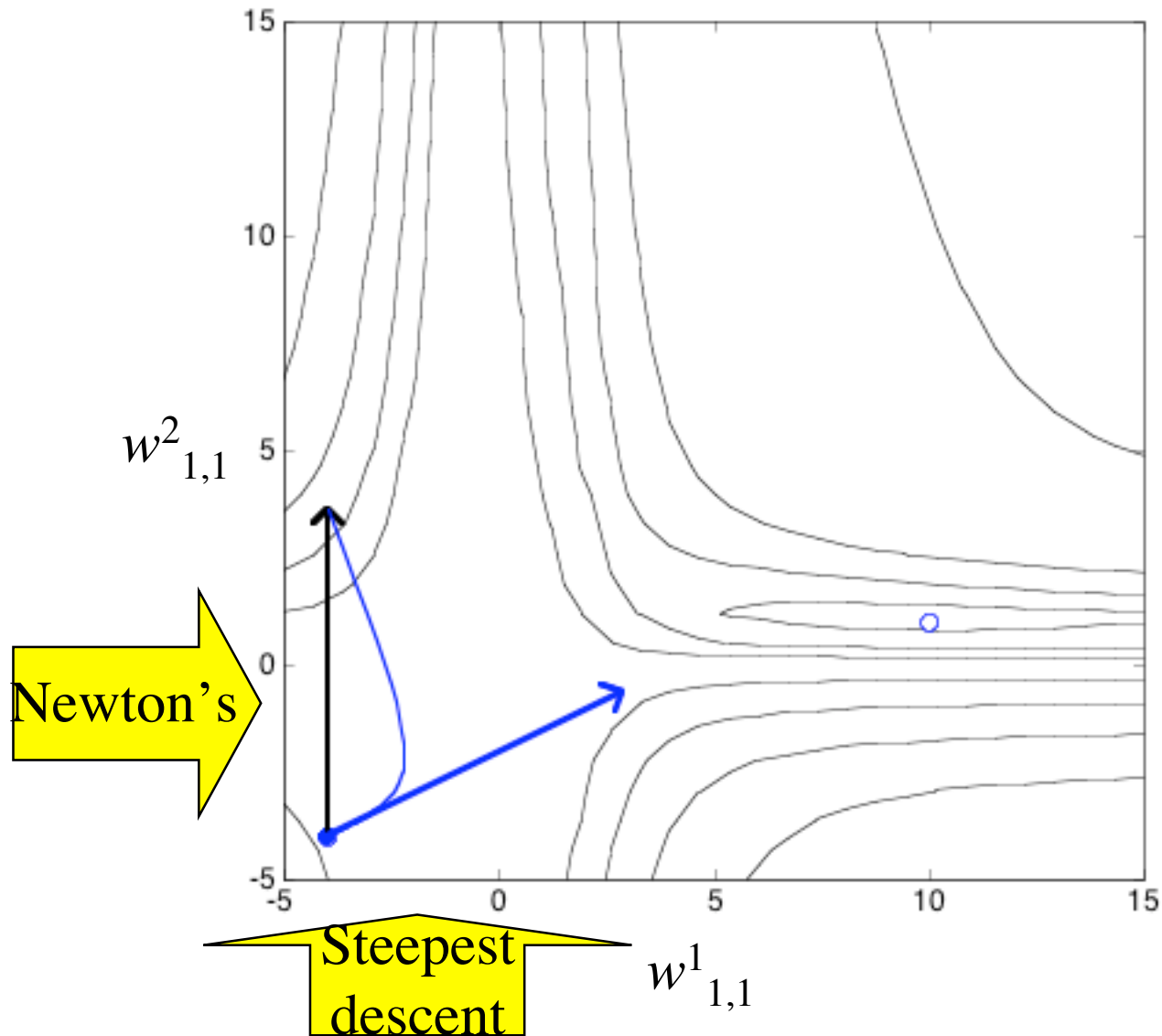
1. Present *all inputs* to the network and compute the corresponding network outputs and the errors. Compute the sum of squared errors over all inputs.
2. Compute the Jacobian matrix: Calculate the sensitivities with the backpropagation algorithm, after initializing. Augment the individual matrices into the Marquardt sensitivities. Compute the elements of the Jacobian matrix using previous equations.
3. Invert the matrix in the following equation

$$\mathbf{x}_{k+1} = \mathbf{x}_k - [\mathbf{J}^T(\mathbf{x}_k)\mathbf{J}(\mathbf{x}_k) + \mu_k\mathbf{I}]^{-1}\mathbf{J}^T(\mathbf{x}_k)\mathbf{v}(\mathbf{x}_k)$$

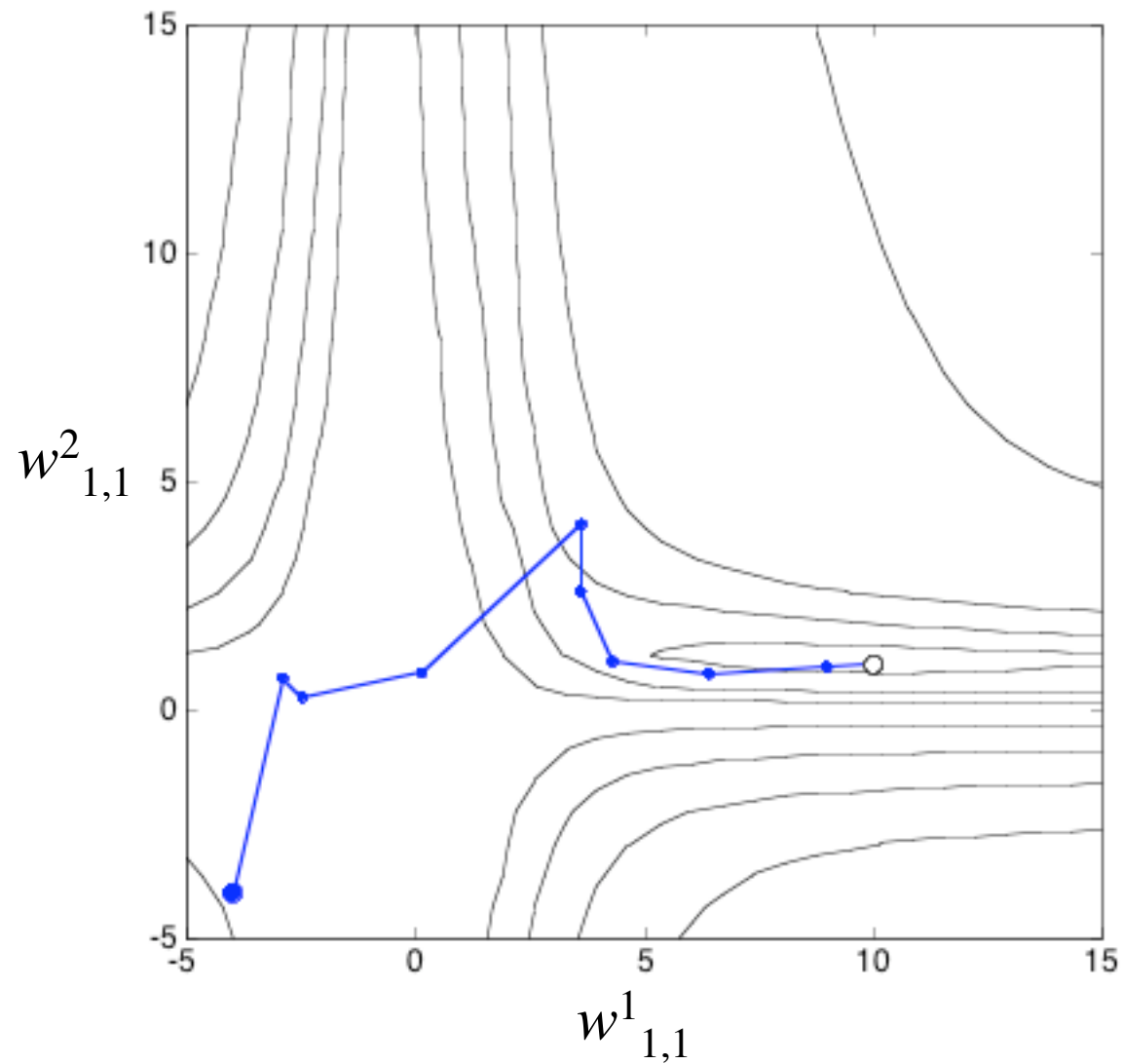
to obtain the weight updates.

4. Recompute the sum of squared errors with the new weights. If this new sum of squares is smaller than that computed in step 1, then divide  $\mu_k$  by  $\nu$ , update the weights and go back to step 1. If the sum of squares is not reduced, then multiply  $\mu_k$  by  $\nu$  and go back to step 3.

# Example LMBP Step



# LMBP Trajectory

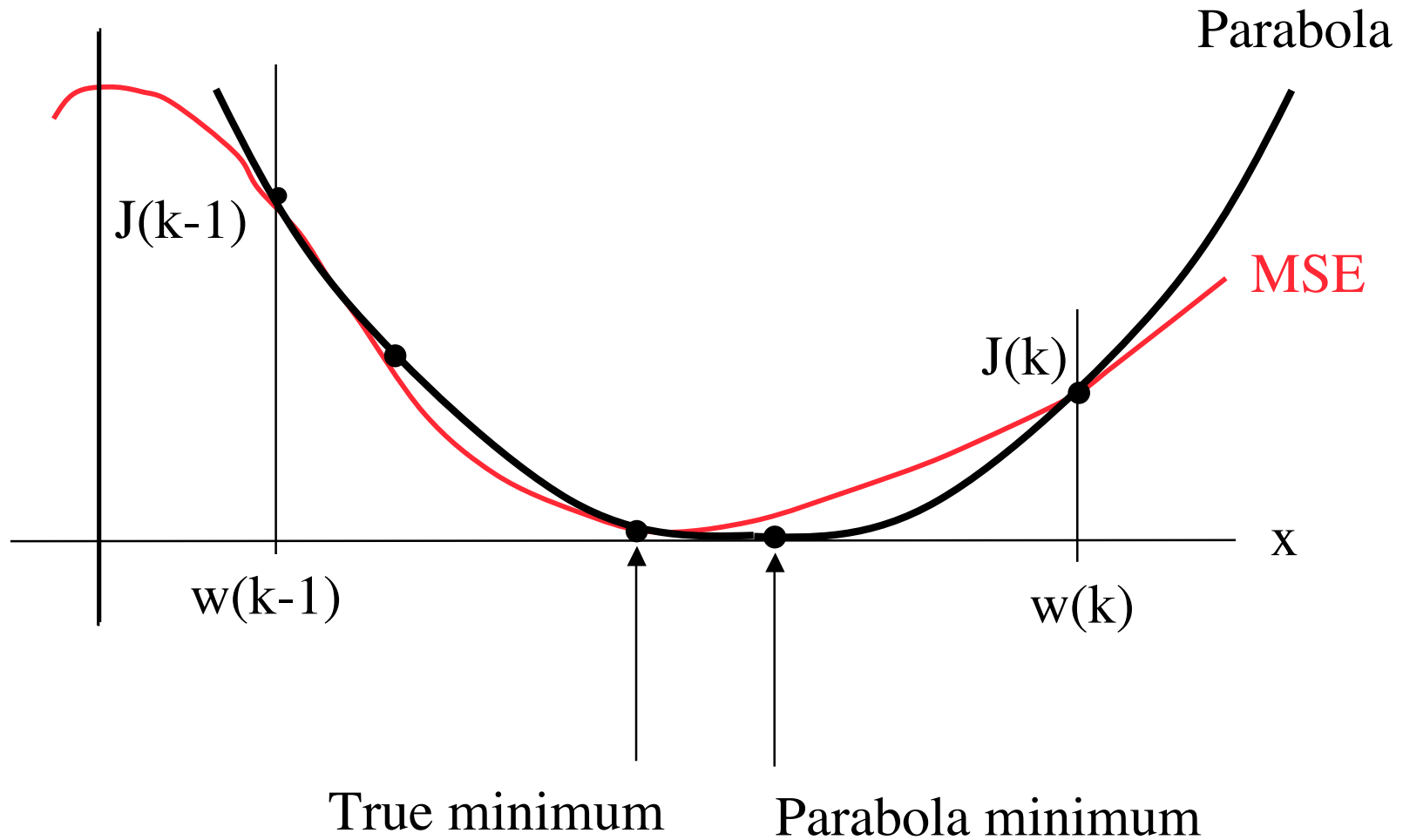


# Quickprop

## Scott Fahlman, CMU

- This is an optimization of backpropagation based on Newton's method.
- It is applicable when, between two steps, the **gradient** has decreased in magnitude and has **changed sign**.
- Then a **parabolic estimate** of the MSE is used to determine the weights for the next step.

# Quickprop, step k, in 1 dimension



# Quickprop

- Assume a parabola:

$$J(w) = aw^2 + bw + c$$

- First derivative is a line:

$$\partial J / \partial w = 2aw + b$$

abbreviate  $\partial J / \partial w$  as  $J'(w)$ .

- To find: value of  $w(k+1)$  such that  $J'(w(k+1)) = 0$ .

- We have

$$J'(w(k)) = 2aw(k) + b$$

$$J'(w(k-1)) = 2aw(k-1) + b$$

- Solving for a and b in terms of the other quantities:

$$2a = [J'(w(k)) - J'(w(k-1))] / \Delta w(k-1)$$

$$b = J'(w(k)) - [(J'(w(k)) - J'(w(k-1)))w(k) / \Delta w(k-1)]$$

where  $\Delta w(k-1) = w(k) - w(k-1)$

- (continued next page)

# Quickprop

- Set  $J'(w(k+1)) = 0$ , since we are looking for the parabolic **minimum** at the next step.
- Then  $2a w(k+1) + b = 0$ , i.e.  $w(k+1) = -b/2a$ .
- Substituting in previous equations, we get

$w(k+1) =$

$$w(k) + [J'(w(k)) \Delta w(k-1)] / [J'(w(k-1)) - J'(w(k))]$$

as the choice for  $w(k+1)$ .