Artificial Neural Networks

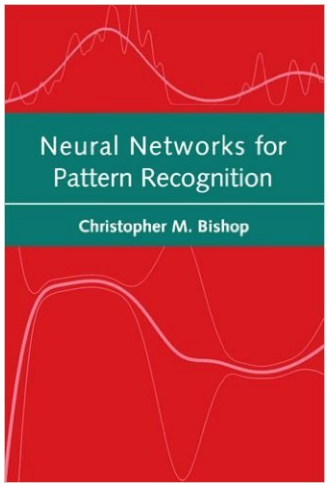
Lecture 4: Optimization

Tudor Berariu
tudor.berariu@gmail.com

Faculty of Automatic Control and Computers
University Politehnica of Bucharest

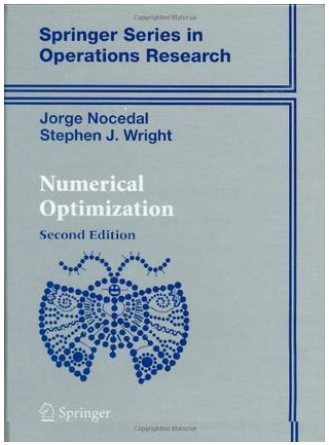
Lecture : 4^{th} of November, 2015
Last Updated: 4^{th} of November, 2015

Resources



Chapter 7 in

Christopher M. Bishop, *Neural networks for pattern recognition*, Oxford University Press, Inc., New York, NY, USA, 1995



Chapters 2-7 in

Jorge Nocedal and Stephen Wright, *Numerical optimization*, Springer Science & Business Media, 2006

Today's Outline

1. Calculus refresher
2. Optimization
3. Optimization Algorithms
4. Conjugate Gradients

Today's Outline

The gradient

Definition

The gradient of a differentiable real function $E(\mathbf{w}) : \mathbb{R}^W \rightarrow \mathbb{R}$ is a vector:

$$\nabla E = \begin{bmatrix} \frac{\partial E}{\partial w_1} \\ \frac{\partial E}{\partial w_2} \\ \vdots \\ \frac{\partial E}{\partial w_W} \end{bmatrix} \tag{1}$$

The Hessian

Definition

The second-order gradient of a differentiable real function $E(\mathbf{w}) : \mathbb{R}^W \to \mathbb{R}$ is a matrix (called *the Hessian*):

$$\mathbf{H} = \nabla^2 E = \begin{bmatrix} \frac{\partial E}{\partial^2 w_1} & \frac{\partial E}{\partial w_1 \partial w_2} & \cdots & \frac{\partial E}{\partial w_1 \partial w_W} \\ \frac{\partial E}{\partial w_1 \partial w_2} & \frac{\partial E}{\partial^2 w_2} & \cdots & \frac{\partial E}{\partial w_2 \partial w_W} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial E}{\partial w_W \partial w_1} & \frac{\partial E}{\partial w_W \partial w_2} & \cdots & \frac{\partial E}{\partial^2 w_W} \end{bmatrix} \tag{2}$$

Taylor series

- For a single-variable infinetely differentiable around a function, the Taylor expansion:

$$f(x) = \sum_{i=0}^{\infty} \frac{f^{(i)}(a)}{i!}(x-a)^i \tag{3}$$

- The quadratic approximation: $f(x) : \mathbb{R} \to \mathbb{R}$:

$$f(x) \approx f(a) + f'(a)(x-a) + \frac{1}{2!}f''(a)(x-a)^2 \tag{4}$$

- Quadratic approximation for multi-variate functions:

$$f(\mathbf{x}) \approx f(\mathbf{a}) + (\mathbf{x}-\mathbf{a})^{\mathsf{T}} \cdot \nabla f(\mathbf{a}) + (\mathbf{x}-\mathbf{a})^{\mathsf{T}} \cdot \mathbf{H}(\mathbf{a}) \cdot (\mathbf{x}-\mathbf{a}) \tag{5}$$

Today's Outline

What is Numerical Optimization?

Definition

Numerical Optimization (**mathematical programming**) is the problem of maximization or minimization of an *objective* scalar function f subject to some constraints on its variables $\mathbf{x}$.

$$\min_{\mathbf{x}\in\mathbb{R}^n} f(\mathbf{x}), \qquad \text{subject to} \begin{cases} c_e(\mathbf{x}) = 0, & \forall e \in \mathcal{E} \\ c_i(\mathbf{x}) \geq 0, & \forall i \in \mathcal{I} \end{cases} \tag{6}$$

where

- $\mathcal{E}$ are the equality constraints and
- $\mathcal{I}$ are the inequality constraints

Linear vs. Non-linear Optimization Problems

Definition

Linear programming problems are optimization problems whose objective function and constraints are all linear functions with respect to the parameters.

Definition

Nonlinear programs are optimization problems where the objective function and/or the constraints are nonlinear.

Fact

Most **ML** *models require nonlinear optimization. (exceptions: linear regression, logistic regression, single layer neural networks)*

Continuous versus Discrete Optimization

Definition

Integer programming problems are optimization problems where variables are constrained to take integer values. ($\mathbf{x} \in \mathbb{Z}^n$)

Definition

Mixed integer programming problems are optimization problems where *some* of the variables are constrained to take integer values.

Definition

Optimization problems where variables take real values ($\mathbf{x} \in \mathbb{R}^n$) are called **continuous optimization** problems.

Fact

In **ML** *we are generally interested in continuous optimization.*

Constrained versus Unconstrained Optimization

Definition

Unconstrained optimization problems have $\mathcal{E} = \mathcal{I} = \varnothing$

Definition

Constrained optimization problems have $\mathcal{E} \neq \varnothing \vee \mathcal{I} \neq \varnothing$

Fact

Constrained optimization problems can be transformed to unconstrained optimization problems by replacing constraints with penalization terms (e.g. in **ML***: regularization).*

Convexity

Definition

A set $S \in \mathbb{R}^n$ is a **convex set** if any straight line segment that connects two points in S lies entirely inside S.

Definition

A function f is a **convex function** if its domain S is a convex set and if for any two points $\mathbf{x}$ and $\mathbf{y}$ in S the following property is satisfied:

$$f\left(\alpha \mathbf{x} + (1-\alpha)\,\mathbf{y}\right) \leq \alpha f\left(\mathbf{x}\right) + (1-\alpha)\, f\left(\mathbf{y}\right) \tag{7}$$

for all $\alpha \in [0, 1]$.

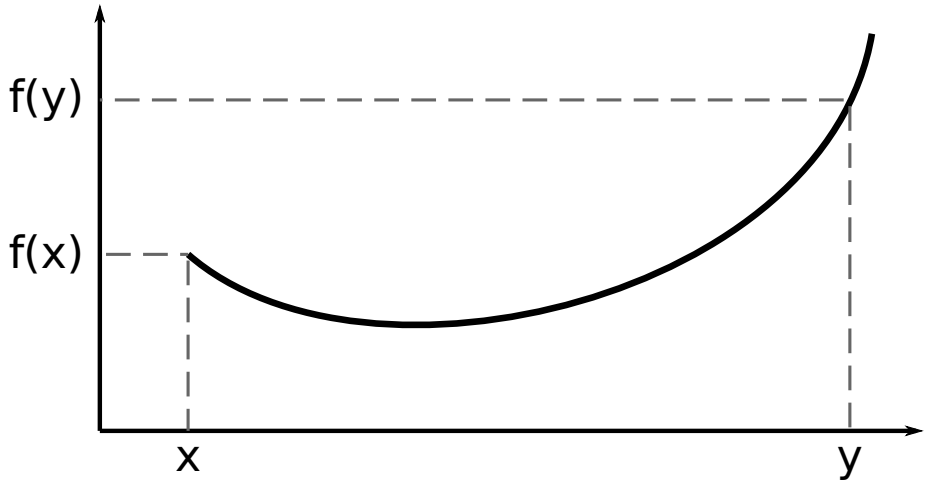
Convex function

Definition

A function f is a **convex function** if its domain S is a convex set and if for any two points $\mathbf{x}$ and $\mathbf{y}$ in S the following property is satisfied:

$$f\left(\alpha\mathbf{x} + (1-\alpha)\,\mathbf{y}\right) \leq \alpha f\left(\mathbf{x}\right) + (1-\alpha)\, f\left(\mathbf{y}\right) \tag{8}$$

for all $\alpha \in [0, 1]$.



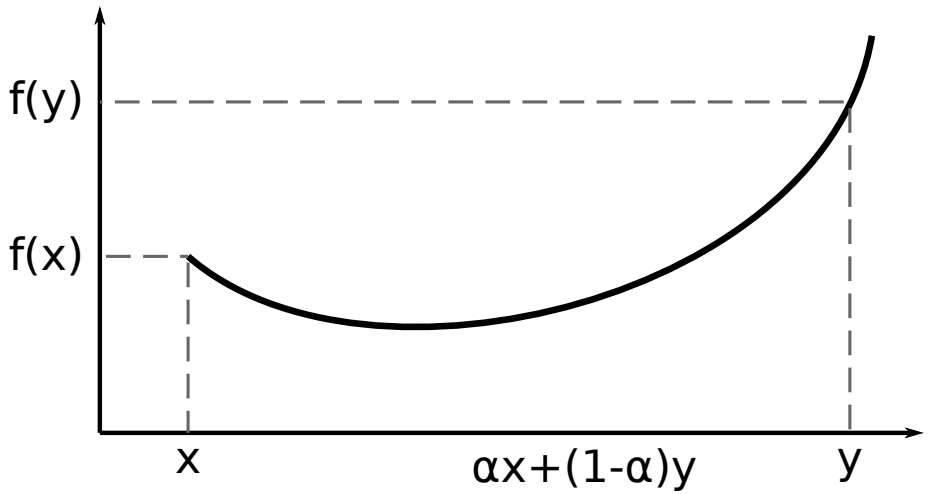
Convex function

Definition

A function f is a **convex function** if its domain S is a convex set and if for any two points $\mathbf{x}$ and $\mathbf{y}$ in S the following property is satisfied:

$$f\left(\alpha\mathbf{x} + (1-\alpha)\,\mathbf{y}\right) \leq \alpha f\left(\mathbf{x}\right) + (1-\alpha)\, f\left(\mathbf{y}\right) \tag{8}$$

for all $\alpha \in [0, 1]$.



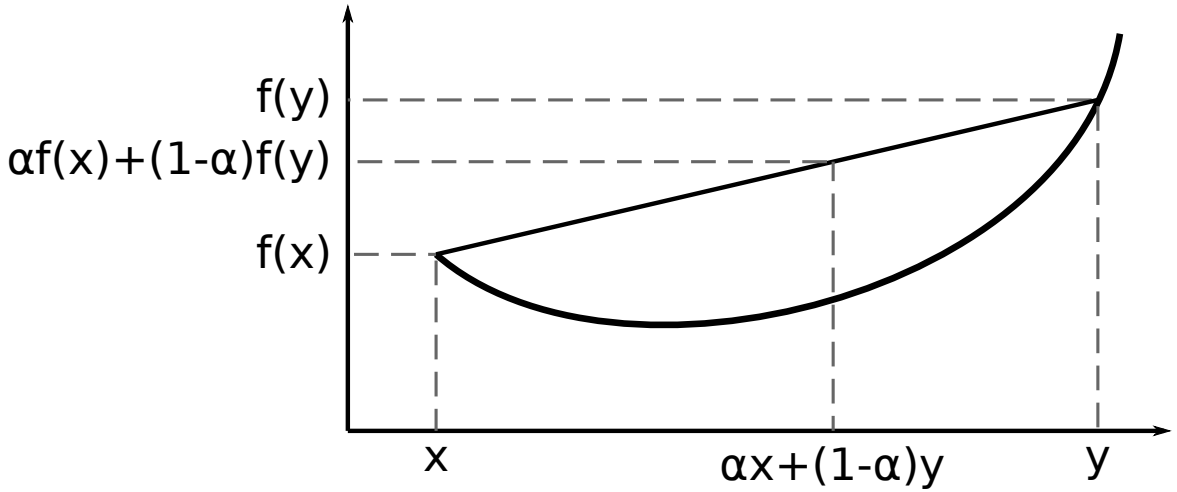
Convex function

Definition

A function f is a **convex function** if its domain S is a convex set and if for any two points $\mathbf{x}$ and $\mathbf{y}$ in S the following property is satisfied:

$$f\left(\alpha\mathbf{x} + (1-\alpha)\,\mathbf{y}\right) \leq \alpha f\left(\mathbf{x}\right) + (1-\alpha)\, f\left(\mathbf{y}\right) \tag{8}$$

for all $\alpha \in [0, 1]$.



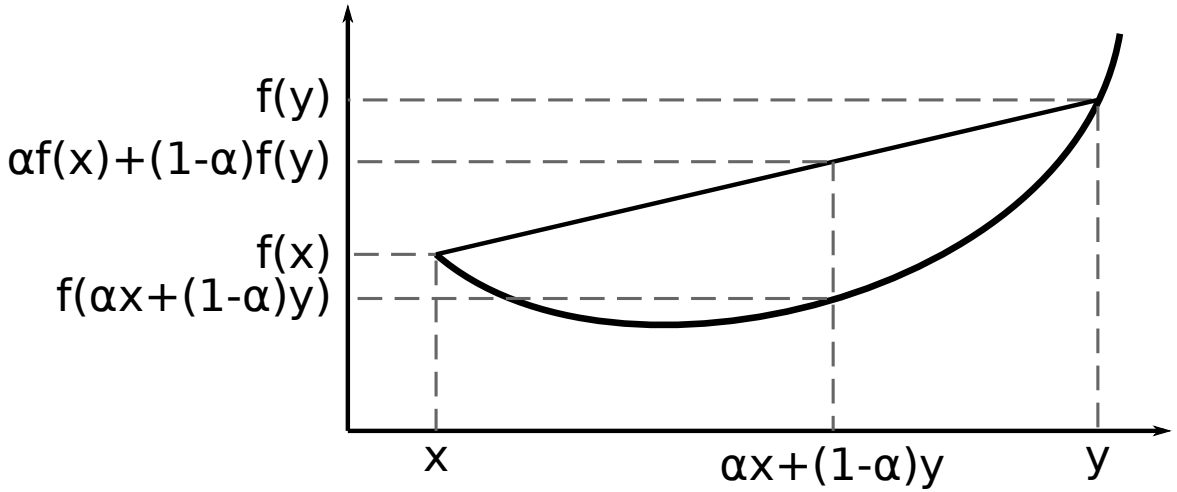
Convex function

Definition

A function f is a **convex function** if its domain S is a convex set and if for any two points $\mathbf{x}$ and $\mathbf{y}$ in S the following property is satisfied:

$$f\left(\alpha\mathbf{x} + (1-\alpha)\,\mathbf{y}\right) \leq \alpha f\left(\mathbf{x}\right) + (1-\alpha)\, f\left(\mathbf{y}\right) \tag{8}$$

for all $\alpha \in [0, 1]$.



Convex problems

Definition

Convex problems are those optimization problems in which:

- the objective function is convex,
- the equality constraint functions c_i, $i \in \epsilon$, are linear, and
- the inequality constraint functions c_i, $i \in \iota$, are concave.

Theorem

If the objective function and the feasible region are both convex, a local minimum is also a global minimum.

Fact

In general, **ML** *problems are not convex.*

Identifying solutions

Definition

A point x^* is a **global minimizer** if $f(x^*) \leq f(x), \forall x \in \mathbb{R}^n$.

Identifying solutions

Definition

A point x^* is a **global minimizer** if $f(x^*) \leq f(x), \forall x \in \mathbb{R}^n$.

Definition

A point x^* is a **local minimizer** if there is a neighborhood $\mathcal{N}$ of x^* such that $f(x^*) \leq f(x), \forall x \in \mathcal{N}$.

Identifying solutions

Definition

A point x^* is a **global minimizer** if $f(x^*) \leq f(x), \forall x \in \mathbb{R}^n$.

Definition

A point x^* is a **strict local minimizer** if there is a neighborhood $\mathcal{N}$ of x^* such that $f(x^*) < f(x), \forall x \in \mathcal{N}$ with $x \neq x^*$.

Definition

A point x^* is an **isolated local minimizer** if there is a neighborhood $\mathcal{N}$ of x^* such that x^* is the only local minimizer in $\mathcal{N}$.

Identifying solutions

Definition

A point x^* is a **global minimizer** if $f(x^*) \leq f(x), \forall x \in \mathbb{R}^n$.

Definition

A point x^* is a **strict local minimizer** if there is a neighborhood $\mathcal{N}$ of x^* such that $f(x^*) < f(x), \forall x \in \mathcal{N}$ with $x \neq x^*$.

Definition

A point x^* is an **isolated local minimizer** if there is a neighborhood $\mathcal{N}$ of x^* such that x^* is the only local minimizer in $\mathcal{N}$.

Fact

While all isolated local minimizers are strict, not all strict local minimizers are isolated.

Strict local minimizers that are not isolated

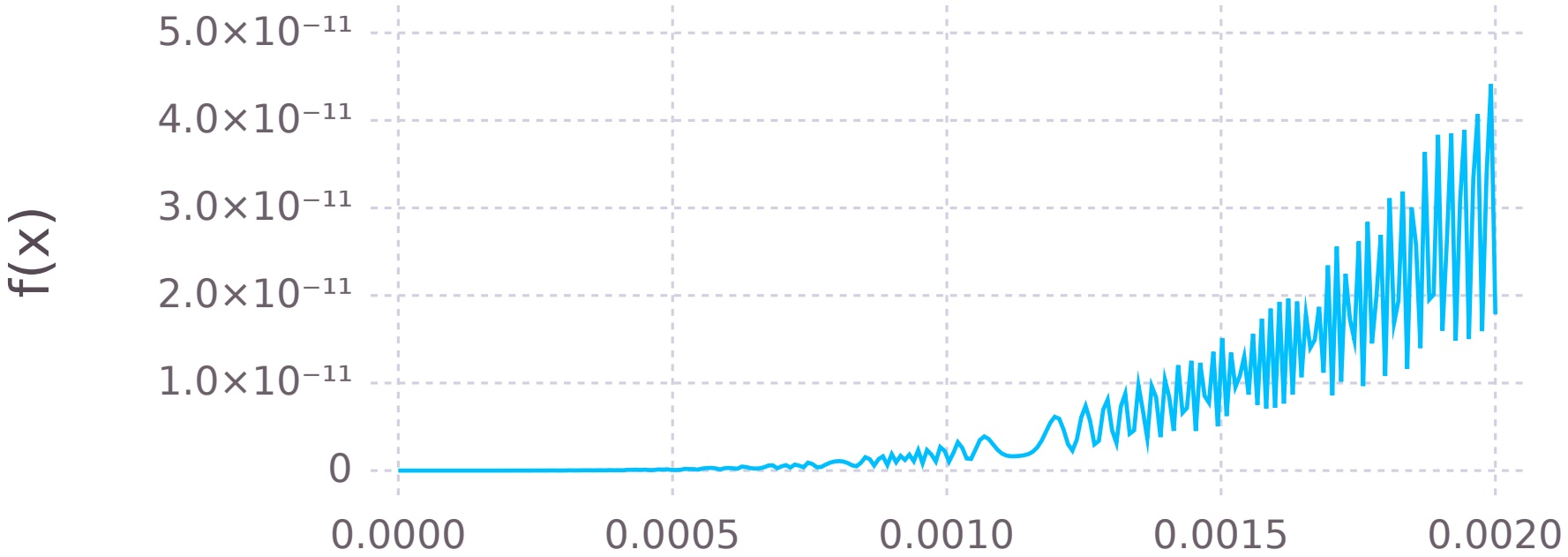
Strict local minimizers that are not isolated?

Strict local minimizers that are not isolated

Strict local minimizers that are not isolated?

Think about the following function:

$$f(x) = x^4 \cos\left(\frac{1}{x}\right) + 2x^4, f(0) = 0$$



Second-Order Sufficient Conditions

If f is twice differentiable (from Taylor):

$$f(\mathbf{x}+\mathbf{p}) = f(\mathbf{x}) + \nabla f(\mathbf{x})^{\mathsf{T}}\mathbf{p} + \frac{1}{2}\mathbf{p}^{\mathsf{T}}\nabla^2 f(\mathbf{x}+t\mathbf{p})\mathbf{p} \tag{9}$$

for some $t \in (0,1)$.

Definition

Suppose that $\nabla^2 f$ is continuous in an open neighbourhood of $\mathbf{x}^*$ and that $\nabla f(\mathbf{x}^*) = 0$ and $\nabla^2 f(\mathbf{x}^*)$ is positive definite. Then $\mathbf{x}^*$ is a strict local minimizer of f.

Motivation for ML

- Supervised learning of artificial neural networks can be interpreted as a problem in **numerical optimization**.
- The cost function E is a nonlinear function of a weight vector $\mathbf{w}$
- The task of learning is to minimize E w.r.t. $\mathbf{w}$.

Warning! Change of notation:

- objective function: $E(\mathbf{w})$ with $\mathbf{w} \in \mathbb{R}^W$

Today's Outline

Today's Outline

1. Calculus refresher
2. Optimization
3. Optimization Algorithms
   - Optimization Strategies
   - Steepest Descent
   - Geometric interpretation
   - Line Search
   - Second-order methods
   - Quasi-Newton
   - Other topics...
4. Conjugate Gradients

Iterative Algorithms

Goal: minimize $E(\mathbf{w})$

- An iterative algorithm starts with a candidate solution $\mathbf{w}_0$ and generates a sequence $\{\mathbf{w}_k\}_{k=0}^{\infty}$
- The algorithm stops when no improvement can be made or the solution has been approximated with enough accuracy.

Optimization Strategies

- two big families of optimization algorithms:
  - **line search**
  - **trust region**

Optimization Strategies

- two big families of optimization algorithms:
  - **line search**
  - **trust region**
- the **line search** strategy follows these steps:
  1. choose a direction $\mathbf{d}_k$
  2. the distance to move along $\mathbf{d}_k$ is found by minimization:

$$\min_{\alpha>0} E\left(\mathbf{w}_k + \alpha \mathbf{d}_k\right) \tag{10}$$

Optimization Strategies

- two big families of optimization algorithms:
  - **line search**
  - **trust region**
- the **line search** strategy follows these steps:
  1. choose a direction $\mathbf{d}_k$
  2. the distance to move along $\mathbf{d}_k$ is found by minimization:

$$\min_{\alpha>0} E\left(\mathbf{w}_k + \alpha \mathbf{d}_k\right) \tag{10}$$

- in the **trust region** strategy, the algorithms:
  1. construct a model function m_k
  2. find a vector $\mathbf{p}$ to move inside the **trust region**

$$\min_{\mathbf{p}} m_k\left(\mathbf{w}_k + \mathbf{p}\right), \qquad \text{such that } ||\mathbf{p}|| < \Delta \tag{11}$$

Δ represents the **trust region radius**

Today's Outline

1. Calculus refresher
2. Optimization
3. Optimization Algorithms
   - Optimization Strategies
   - Steepest Descent
   - Geometric interpretation
   - Line Search
   - Second-order methods
   - Quasi-Newton
   - Other topics...
4. Conjugate Gradients

Line search strategies: steepest descent

Definition

The **steepest descent method** is a line search algorithm that moves along $\mathbf{d}_k = -\nabla E_k = -\mathbf{g}_k$ at every step.

- does not need second order derivatives
- might be very, very slow
- you need to set how much you move along that direction

Heuristic enhancements:

- momentum
- bold driver
- delta-bar-delta
- rmsprop

Today's Outline

1. Calculus refresher
2. Optimization
3. Optimization Algorithms
   - Optimization Strategies
   - Steepest Descent
   - Geometric interpretation
   - Line Search
   - Second-order methods
   - Quasi-Newton
   - Other topics...
4. Conjugate Gradients

Local Quadratic Approximation

The Taylor expansion of $E(\mathbf{w})$ around an arbitrary point $\hat{\mathbf{w}}$:

$$E(\mathbf{w}) = E(\hat{\mathbf{w}}) + (\mathbf{w} - \hat{\mathbf{w}})^{\mathsf{T}}\mathbf{b} + \frac{1}{2}(\mathbf{w} - \hat{\mathbf{w}})^{\mathsf{T}}\mathbf{H}(\mathbf{w} - \hat{\mathbf{w}}) \tag{12}$$

where

- $\mathbf{b} = \nabla E\Big|_{\hat{\mathbf{w}}}$
- $\mathbf{H} = \left\{ \frac{\partial E}{\partial w_i \partial w_j}\Big|_{\hat{\mathbf{w}}} \right\}_{1 \le i,j \le W}$

Local approximation for the gradient:

$$\nabla E = \mathbf{b} + \mathbf{H}(\mathbf{w} - \hat{\mathbf{w}}) \tag{13}$$

Local Quadratic Approximation around a minimum

The Taylor expansion of $E(\mathbf{w})$ around a local minimum $\mathbf{w}^*$:

$$E(\mathbf{w}) = E(\mathbf{w}^*) + \frac{1}{2}(\mathbf{w} - \mathbf{w}^*)^{\mathsf{T}}\mathbf{H}(\mathbf{w} - \mathbf{w}^*) \tag{14}$$

where

- $\nabla E\Big|_{\mathbf{w}^*} = 0$
- $\mathbf{H} = \left\{ \frac{\partial E}{\partial w_i \partial w_j}\Big|_{\mathbf{w}^*} \right\}_{1 \le i,j \le W}$

Local approximation for the gradient:

$$\nabla E = \mathbf{H}(\mathbf{w} - \mathbf{w}^*) \tag{15}$$

Eigenvectors of the Hessian

Consider the eigenvector equation for the Hessian:

$$\mathbf{H}\mathbf{u}_i = \lambda_i \mathbf{u}_i \tag{16}$$

Because the eigenvectors $\mathbf{u}_i$ form a complete orthonormal set ($\mathbf{u}_i^{\mathsf{T}}\mathbf{u}_j = \delta_{ij}$):

$$\mathbf{w} - \mathbf{w}^* = \sum_i^W \alpha_i \mathbf{u}_i \tag{17}$$

Using 16 and 17 in 14, the error can be written in the following form:

$$E(\mathbf{w}) = E(\mathbf{w}^*) + \frac{1}{2}\sum_i \lambda_i \alpha_i^2$$

Eigenvectors of the Hessian

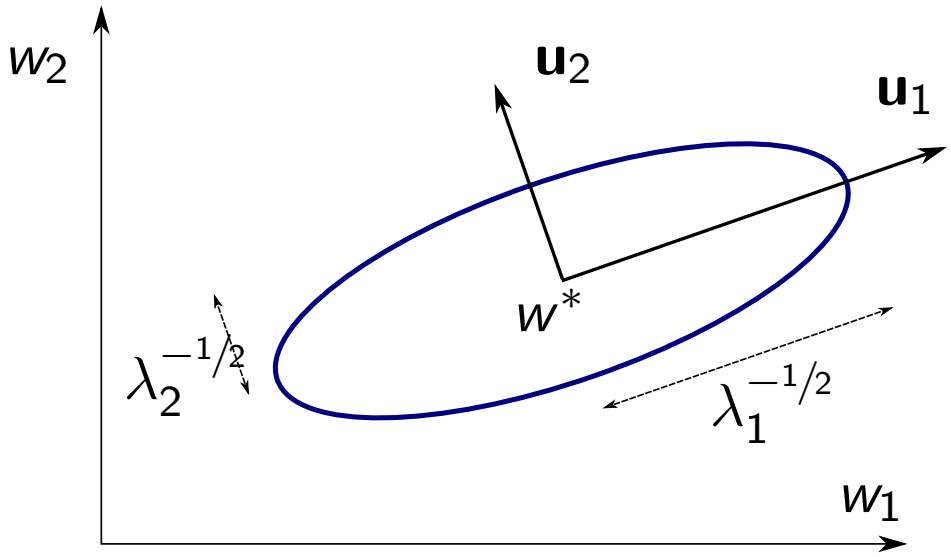


Figure: Quadratic approximation of the error function around a minimum w^*. (adapted from a similar image in [Bis95, page 259])

$$E(\mathbf{w}) = E(\mathbf{w}^*) + \frac{1}{2}\sum_i \lambda_i \alpha_i^2$$

Problems with the gradient descent

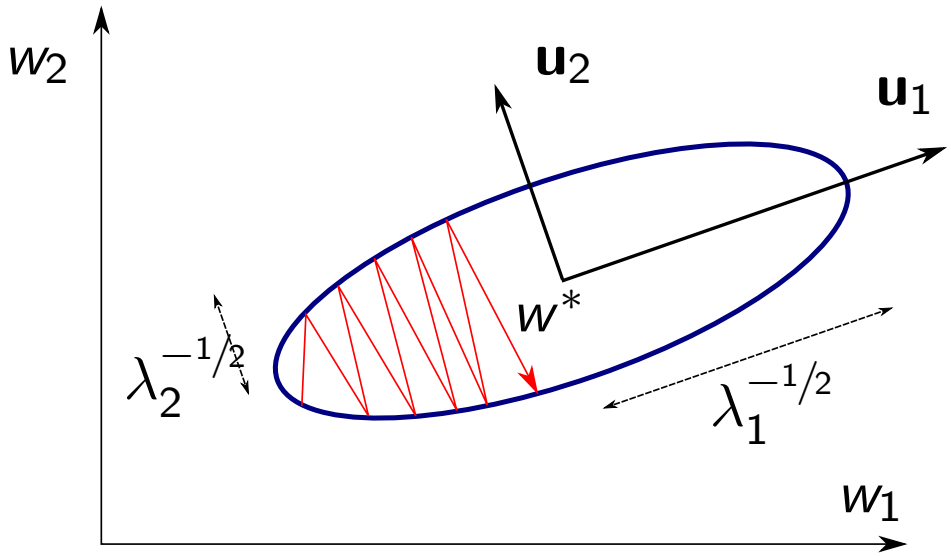


Figure: Oscillations caused by the fact that the gradient does not point towards the minimum

Problems with the gradient descent

Remember the error written as a quadratic approximation around the minimum:

$$E(\mathbf{w}) = E(\mathbf{w}^*) + \frac{1}{2}\sum_i \lambda_i \alpha_i^2$$

Problems with the gradient descent

Remember the error written as a quadratic approximation around the minimum:

$$E(\mathbf{w}) = E(\mathbf{w}^*) + \frac{1}{2}\sum_i \lambda_i \alpha_i^2$$

which leads to the expression of the gradient:

$$\nabla E = \sum_i \alpha_i \lambda_i \mathbf{u}_i \tag{16}$$

Problems with the gradient descent

Remember the error written as a quadratic approximation around the minimum:

$$E(\mathbf{w}) = E(\mathbf{w}^*) + \frac{1}{2}\sum_i \lambda_i \alpha_i^2$$

which leads to the expression of the gradient:

$$\nabla E = \sum_i \alpha_i \lambda_i \mathbf{u}_i \tag{16}$$

Remember from earlier:

$$\mathbf{w} - \mathbf{w}^* = \sum_i^W \alpha_i \mathbf{u}_i$$

Problems with the gradient descent

Remember the error written as a quadratic approximation around the minimum:

$$E(\mathbf{w}) = E(\mathbf{w}^*) + \frac{1}{2}\sum_i \lambda_i \alpha_i^2$$

which leads to the expression of the gradient:

$$\nabla E = \sum_i \alpha_i \lambda_i \mathbf{u}_i \tag{16}$$

Remember from earlier:

$$\mathbf{w} - \mathbf{w}^* = \sum_i^W \alpha_i \mathbf{u}_i$$

which gives

$$\Delta\mathbf{w} = \sum_i \Delta\alpha_i \mathbf{u}_i \tag{17}$$

Problems with the gradient descent

Replacing $\nabla E = \sum_i \alpha_i \lambda_i \mathbf{u}_i$ and $\Delta \mathbf{w} = \sum_i \Delta\alpha_i \mathbf{u}_i$ in:

$$\Delta \mathbf{w}^\tau = -\eta \nabla E\big|_{\mathbf{w}(\tau)}$$

leads to

$$\Delta\alpha_i = -\eta \lambda_i \alpha_i$$

Problems with the gradient descent

Replacing $\nabla E = \sum_i \alpha_i \lambda_i \mathbf{u}_i$ and $\Delta \mathbf{w} = \sum_i \Delta\alpha_i \mathbf{u}_i$ in:

$$\Delta \mathbf{w}^\tau = -\eta \nabla E\big|_{\mathbf{w}^{(\tau)}}$$

leads to

$$\Delta\alpha_i = -\eta\lambda_i\alpha_i$$

$$\alpha_i^{\text{new}} = (1 - \eta\lambda_i)\alpha_i^{\text{old}}$$

After T steps:

$$\alpha_i^{(T)} = (1 - \eta\lambda_i)^T \alpha_i^{(0)}$$

Problems with the gradient descent

Some words about these two results:

$$\alpha_i^{(T)} = (1 - \eta\lambda_i)^T \alpha_i^{(0)}$$

$$\mathbf{u}_i^T(\mathbf{w} - \mathbf{w}^*) = \alpha_i$$

Problems with the gradient descent

Some words about these two results:

$$\alpha_i^{(T)} = (1 - \eta\lambda_i)^T \alpha_i^{(0)}$$

$$\mathbf{u}_i^T(\mathbf{w} - \mathbf{w}^*) = \alpha_i$$

- if $|1 - \eta\lambda_i| < 1$, then $\alpha_i \to \alpha_i$ when $T \to \infty$
- increasing η will speed-up convergence
- the speed of convergence will be dominated by the smallest λ

Momentum

- Momentum
  - deals with unbalanced eigenvalues (λs)
    - moves more in the direction with consistent gradients
    - moves less on directions where gradients oscillate
  - adds inertia to the motion through weight space
- The gradient descent formula is modified as follows:

$$\Delta \mathbf{w}^{(\tau)} = -\eta \nabla E|_{\mathbf{w}^{(\tau)}} + \mu \Delta \mathbf{w}^{(\tau-1)} \tag{18}$$

The effect of adding momentum

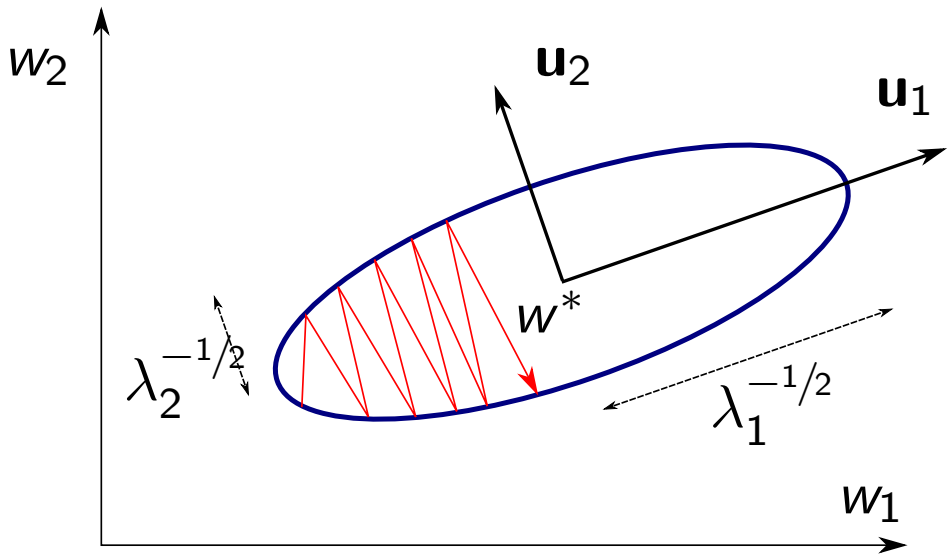


Figure: Difference between gradient descent and gradient descent with momentum

The effect of adding momentum

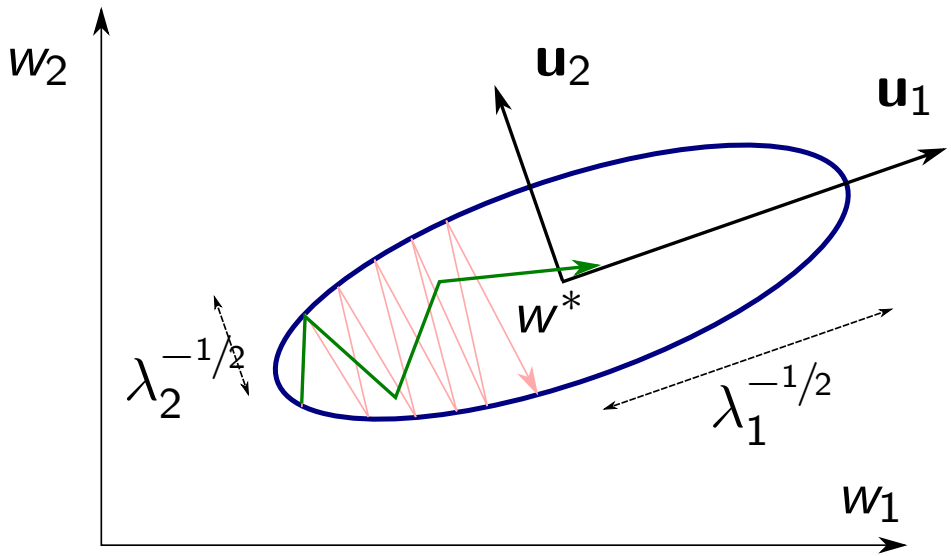


Figure: Difference between gradient descent and gradient descent with momentum

Drawbacks of using momentum

- just another parameter to adjust: μ
- good values for η and μ depend on
  - the problem solved (the data set)
  - the phase of the learning process

The *bold driver* technique

- compute new weights
- check if the error has decreased
  - if **yes**, accept new weights and increase the learning rate
  - if **not**, undo the weight change and decrease the learning rate

$$\eta^{(\text{new})} = \begin{cases} 0.5 \cdot \eta^{(\text{old})} & \text{if } \Delta E < 0 \\ 1.1 \cdot \eta^{(\text{old})} & \text{if } \Delta E > 0 \end{cases}$$

- different heuristics for increasing or decreasing the learning rate exist

delta-delta rule

- some eigenvalues of the Hessian restrict the learning rate to small values
- solution: separate learning rates for weights

$$\Delta w_i^{(\tau)} = -\eta_i^{(\tau)} \frac{\partial E}{\partial w_i^{(\tau)}}$$

- to adjust the learning rates:
  - increase when consecutive gradients had the same sign
  - decrease otherwise

$$\Delta \eta_i^{(\tau)} = \gamma g_i^{(\tau)} g_i^{(\tau-1)}$$

where $g_i^{(\tau)} = \frac{\partial E}{\partial w_i^{(\tau)}}$

- Doesn't work very well as it gets to negative learning rates.

delta-bar-delta rule

- A modified version called the *delta-bar-delta* rule works better.

$$\Delta\eta_i^{(\tau)} = \begin{cases} \kappa & \text{if} \quad \bar{g}^{(\tau-1)} g^{(\tau)} > 0 \\ -\phi\eta_i^{(\tau)} & \text{if} \quad \bar{g}^{(\tau-1)} g^{(\tau)} < 0 \end{cases}$$

where

$$\bar{g}_i^{(\tau)} = (1-\theta) g_i^{(\tau)} + \theta \bar{g}_i^{(\tau-1)}$$

- drawbacks:
  - η, ϕ, κ, μ (more and more paramters)
  - assumes that weights are independent, but the Hessian is far from being diagonal

Rprop (I)

- Rprop - first proposed by Riedmiller and Braun
- update the weights using just the sign of the partial derivatives with respect to the weights
- advantages of using just the sign of the gradient:
  - weight updates are of same order
  - escapes plateaus
- disadvantages:
  - doesn't work with stochastic or mini-batch versions of gradient descent

Rprop (II)

- the learning rate for each weight is accelerated or deccelerated, but both upper and bottom limited

$$\eta_i^{(\tau+1)} = \begin{cases} \min(\rho \cdot \eta_i^{(\tau)}, \eta_{max}) & \text{if} \quad \frac{\partial E}{\partial w_i^{(\tau)}} \cdot \frac{\partial E}{\partial w_i^{(\tau-1)}} > 0 \\ \max(\sigma \cdot \eta_i^{(\tau)}, \eta_{min}) & \text{if} \quad \frac{\partial E}{\partial w_i^{(\tau)}} \cdot \frac{\partial E}{\partial w_i^{(\tau-1)}} < 0 \end{cases} \tag{19}$$

- the weights are update using the following rule:

$$\delta w_i^{(\tau)} = -\eta_i^{(\tau)} \cdot \text{sign}\left(\frac{\partial E}{\partial w_i^{(\tau)}}\right) \tag{20}$$

Quickprop

- approximate the error surface by a quadratic function
- use successive evaluations to determine the coefficients
- move the weights to the minimum of the parabola

$$\Delta w_i^{(\tau+1)} = \frac{g_i^{(\tau)}}{g_i^{(\tau-1)} - g_i^{(\tau)}} \Delta w_i^{(\tau)}$$

QRprop

- QRprop - a combination between Quickprop and Rprop
- the QRprop algorithm works as follows:
  1. while $\text{sign}\left(\frac{\partial E}{\partial w_i^{(\tau)}}\right) = \text{sign}\left(\frac{\partial E}{\partial w_i^{(\tau-1)}}\right)$, Rprop steps are performed
  2. when $\text{sign}\left(\frac{\partial E}{\partial w_i^{(\tau)}}\right) \neq \text{sign}\left(\frac{\partial E}{\partial w_i^{(\tau-1)}}\right)$, Quick prop is used

rmsprop

- another algorithm that combines the robustness of Rprop (individual learning rates and using just the sign of the gradient) with the advantage of using mini-batch
- divide the gradient by a running average of its recent magnitude
- rmsprop (Tijmen Tieleman) computes a moving average of the squared gradient for each weight:

$$ms_i^{(\tau)} = 0.9 \cdot ms_i^{(\tau-1)} + 0.1\left(\frac{\partial E}{\partial w_i^{(\tau)}}\right)^2 \tag{21}$$

rmsprop

- another algorithm that combines the robustness of Rprop (individual learning rates and using just the sign of the gradient) with the advantage of using mini-batch
- divide the gradient by a running average of its recent magnitude
- rmsprop (Tijmen Tieleman) computes a moving average of the squared gradient for each weight:

$$ms_i^{(\tau)} = 0.9 \cdot ms_i^{(\tau-1)} + 0.1\left(\frac{\partial E}{\partial w_i^{(\tau)}}\right)^2 \tag{21}$$

$$\Delta w_j^{(\tau)} = -\eta \cdot \frac{1}{ms_j^{(\tau)}} \frac{\partial E}{\partial w_i^{(\tau)}} \tag{22}$$

Today's Outline

1. Calculus refresher
2. Optimization
3. Optimization Algorithms
   - Optimization Strategies
   - Steepest Descent
   - Geometric interpretation
   - Line Search
   - Second-order methods
   - Quasi-Newton
   - Other topics...
4. Conjugate Gradients

Line Search

- All the algorithms search through the weight space:
  1. find a direction to move along
  2. decide how far to move

Line Search

- All the algorithms search through the weight space:
  1. find a direction to move along
  2. decide how far to move
- Gradient descent
  - moves in the direction of the negative of the gradient
  - the step size is given by the learning rate

Line Search

- All the algorithms search through the weight space:
  1. find a direction to move along
  2. decide how far to move
- Gradient descent
  - moves in the direction of the negative of the gradient
  - the step size is given by the learning rate
- What about this?
  1. move along the direction of the negative gradient to the point where the error is minimized

Line Search

- All the algorithms search through the weight space:
  1. find a direction to move along
  2. decide how far to move
- Gradient descent
  - moves in the direction of the negative of the gradient
  - the step size is given by the learning rate
- What about this?
  1. move along the direction of the negative gradient to the point where the error is minimized- line search

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} + \lambda^{(\tau)}\mathbf{d}^{(\tau)} \tag{23}$$

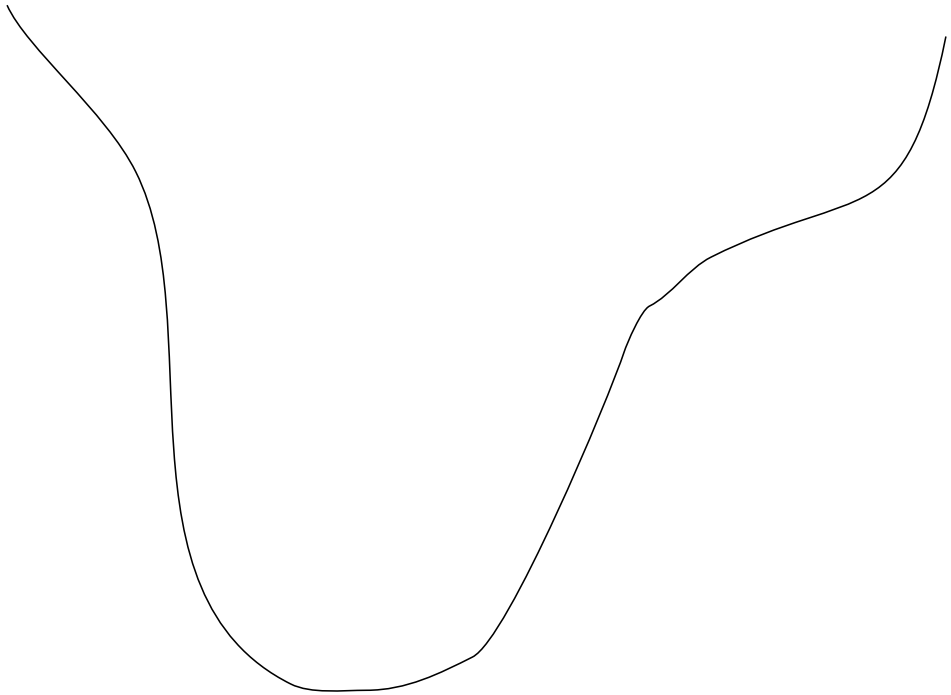
where $\lambda^{(\tau)}$ minimizes

$$E(\lambda) = E(\mathbf{w}^{(\tau)} + \lambda\mathbf{d}^{(\tau)}) \tag{24}$$

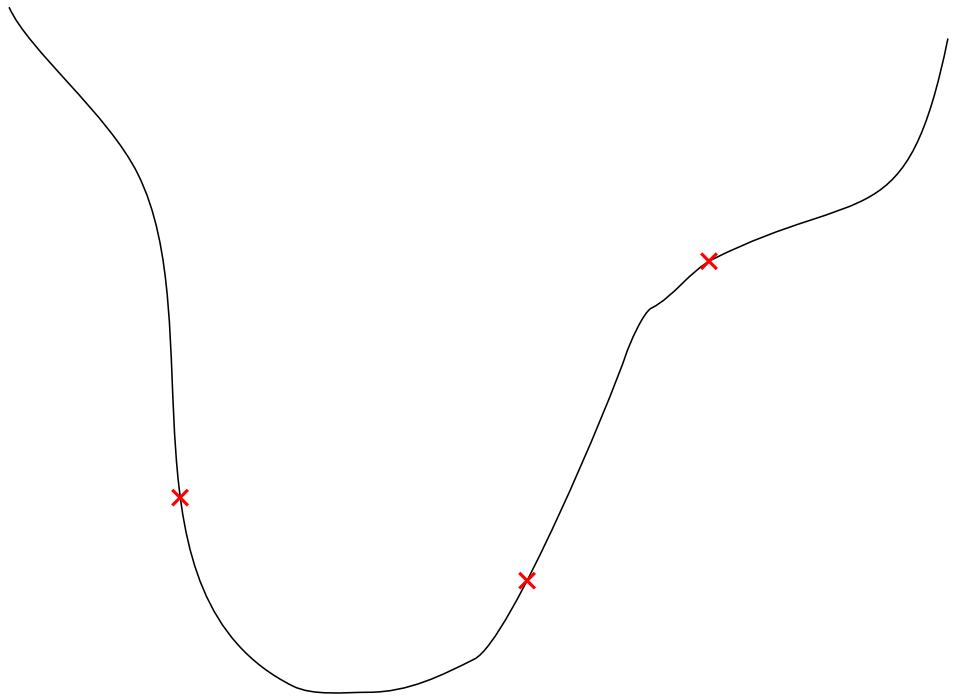
Line Search

1. *bracket* the minimum
   find $a < b < c$ such that $E(a) > E(b)$ and $E(b) < E(c)$
2. locate the minimum : parabolic interpolation
   Brent's algorithm
3. the algorithm is applied several times

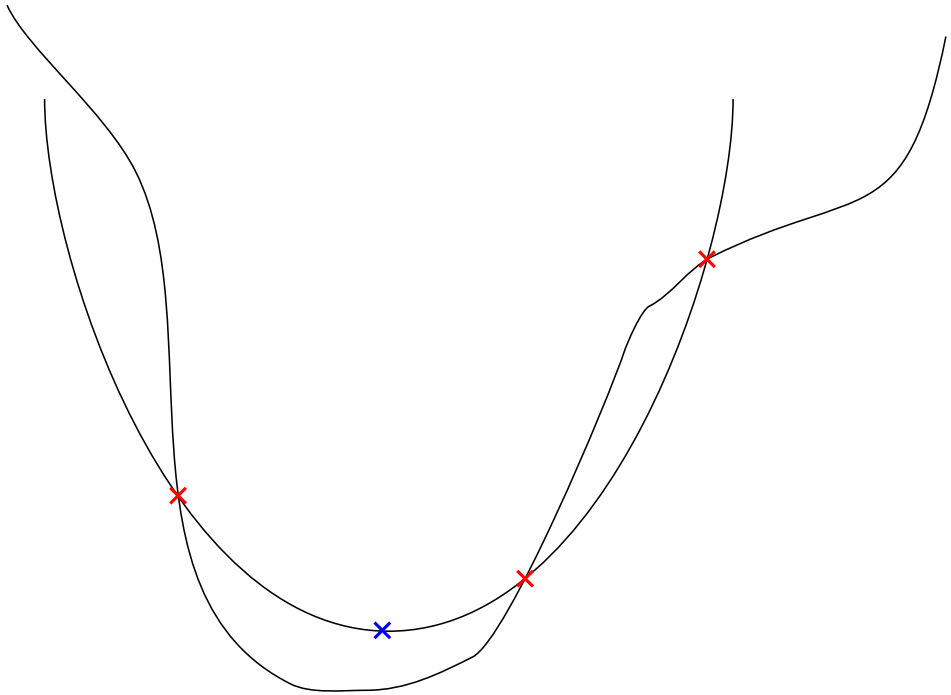
Parabolic Interpolation Example



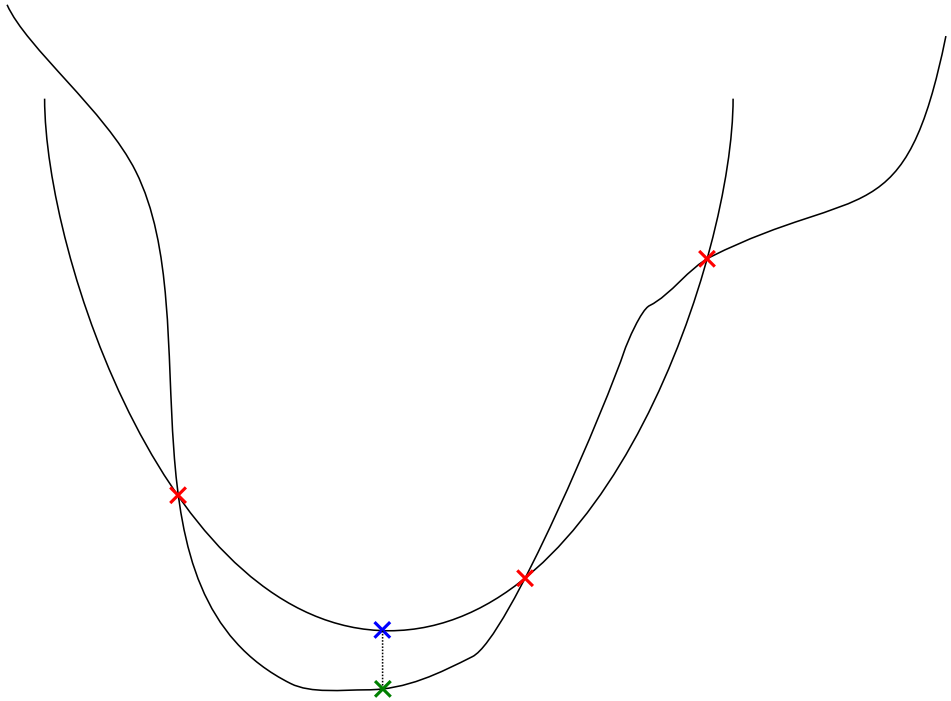
Parabolic Interpolation Example



Parabolic Interpolation Example



Parabolic Interpolation Example



Today's Outline

1. Calculus refresher
2. Optimization
3. Optimization Algorithms
   - Optimization Strategies
   - Steepest Descent
   - Geometric interpretation
   - Line Search
   - Second-order methods
   - Quasi-Newton
   - Other topics...
4. Conjugate Gradients

Line search strategies: Newton direction

- Using the second-order Taylor series approximation to $f(\mathbf{x}_k + \mathbf{d})$:

$$E(\mathbf{w}_k + \mathbf{d}) \approx E_k + d^{\mathsf{T}} \nabla E_k + \frac{1}{2} \mathbf{d}^{\mathsf{T}} \nabla^2 E_k \mathbf{d} = m_k(\mathbf{d}) \tag{25}$$

- if $\nabla^2 E_k$ is positive definite, $\mathbf{d}_k$ is found by minimizing 25:

$$\mathbf{d}_k^{\mathsf{N}} = -\left(\nabla^2 E_k\right)^{-1} \nabla E_k \tag{26}$$

- methods that use **Newton directon** have a fast rate of local convergence
- the Hessian needs to be positive definite (since $\left(\nabla^2 E_k\right)^{-1}$ may not exist)

Newton's method

- Update the weights at each step using:

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \mathbf{H}^{-1}\nabla E \tag{27}$$

- Several steps are necessary as the error function is not really quadratic
- Drawbacks:
  - directly computing the inverse of the Hessian is expensive
    - $\mathcal{O}(NW^2)$ for the Hessian
    - $\mathcal{O}(W^3)$ for inverting it
  - large memory needed (the Hessian is a $W \times W$ matrix)
  - if the Hessian is not positive definite, it moves towards a maximum or a saddlepoint
- direct application of Netwon's method is not used in practice

Illustration of the Newton direction

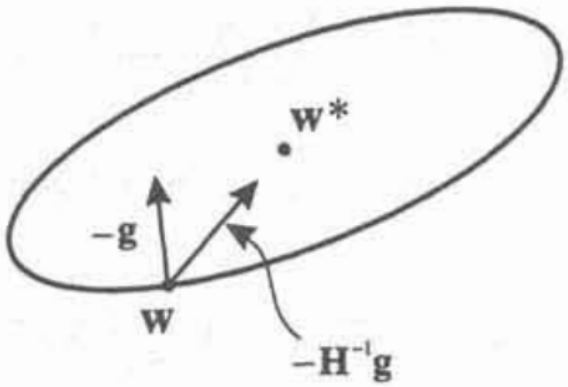


Figure 7.13. Illustration of the Newton direction for a quadratic error surface. The local negative gradient vector $-\mathbf{g}(\mathbf{w})$ does not in general point towards the minimum of the error function, whereas the Newton direction $-\mathbf{H}^{-1}\mathbf{g}(\mathbf{w})$ does.

Saving the Newton Direction

- away from the neighbourhood of the minimum, $\mathbf{H}$ may not be positive definite
- use a *trust region* approach
- use $\mathbf{H} + \lambda \mathbf{I}$ instead of $\mathbf{H}$

$$-\left(\mathbf{H} + \lambda \mathbf{I}\right)^{-1} \mathbf{g} \tag{28}$$

  - for small values of λ, the direction is close to Newton's direction
  - for large values $-(\mathbf{H} + \lambda \mathbf{I})^{-1}\mathbf{g} \approx -\frac{1}{\lambda}\mathbf{g}$

- one still needs to compute the Hessian!

Today's Outline

1. Calculus refresher
2. Optimization
3. Optimization Algorithms
   - Optimization Strategies
   - Steepest Descent
   - Geometric interpretation
   - Line Search
   - Second-order methods
   - Quasi-Newton
   - Other topics...
4. Conjugate Gradients

Quasi-Newton search directions

- **Quasi-Newton** search directions are an alternative to Newton's method
- they do not require computation of the Hessian
- they use approximations of the Hessian

$$\mathbf{p}_k = -\mathbf{B}_k^{-1} \nabla f_k \tag{29}$$

- popular formulae for updating the Hessian approximation B_k:
  - **symmetric-rank-one (SR1)**
  - **BFGS formula**

BFGS

- the *Broyden-Fletcher-Goldfrab-Shanno* expression:

$$\mathbf{G}^{(\tau+1)} = \mathbf{G}^{(\tau)} + \frac{\mathbf{p}\mathbf{p}^T}{\mathbf{p}^T\mathbf{v}} - \frac{(\mathbf{G}^{(\tau)}\mathbf{v})\mathbf{v}^T\mathbf{G}^{(\tau)}}{\mathbf{v}^T\mathbf{G}^{(\tau)}\mathbf{v}} + (\mathbf{v}^T\mathbf{G}^{(\tau)}\mathbf{v})\mathbf{u}\mathbf{u}^T \tag{30}$$

where

$$\mathbf{p} = \mathbf{w}^{(\tau+1)} - \mathbf{w}^{(\tau)} \tag{31}$$

$$\mathbf{v} = \mathbf{g}^{(\tau+1)} - \mathbf{g}^{(\tau)} \tag{32}$$

$$\mathbf{u} = \frac{\mathbf{p}}{\mathbf{p}^T\mathbf{v}} - \frac{\mathbf{G}^{(\tau)}\mathbf{v}}{\mathbf{v}^T\mathbf{G}^{(\tau)}\mathbf{v}} \tag{33}$$

Quasi-Newton methods

- Quasi-Newton methods try to approximate the inverse of the Hessian
- a sequence of matrices $\mathbf{G}^{(\tau)}$ are generated
- $\mathbf{G}^{(\tau)}$ is an increasingly better approximation of $\mathbf{H}^{-1(\tau)}$
- starting with a positive definite matrix (like $\mathbf{G}^{(0)} = I_W$) keeps $\mathbf{G}^{(\tau)}$ positive definite

Quasi-Newton methods - Updating the weights

- the update rule uses a line-search along $\mathbf{G}^{(\tau)}\nabla E(\mathbf{w}^{(\tau)})$:

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} + \alpha^{(\tau)}\mathbf{G}^{(\tau)}\nabla E(\mathbf{w}^{(\tau)}) \tag{34}$$

- the Quasi-Newton method is guaranteed to converge after W steps if line minimization is exact

Today's Outline

1. Calculus refresher
2. Optimization
3. Optimization Algorithms
   - Optimization Strategies
   - Steepest Descent
   - Geometric interpretation
   - Line Search
   - Second-order methods
   - Quasi-Newton
   - Other topics...
4. Conjugate Gradients

Conjugate gradients

$$\mathbf{d}_k = -\nabla f(\mathbf{g}_k) + \beta_k \mathbf{d}_{k-1} \tag{35}$$

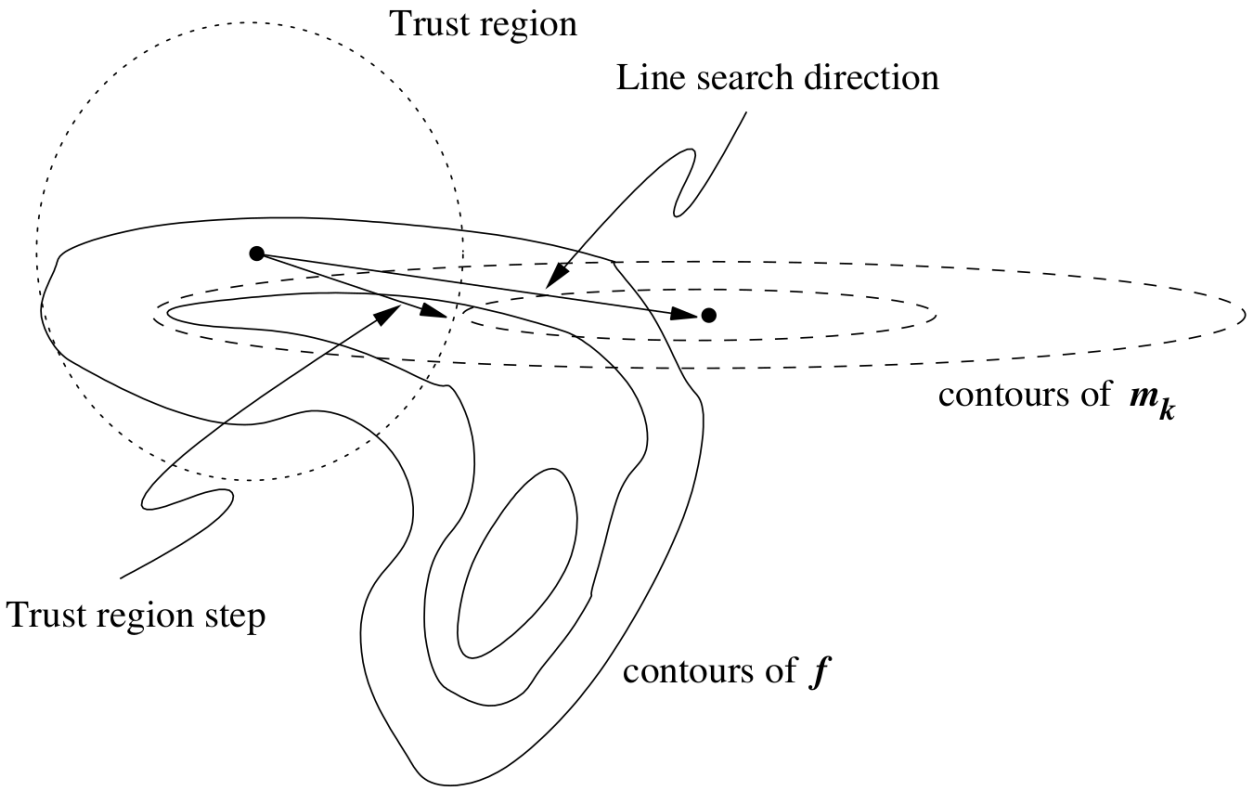
- more effective than steepest descent methods
- they don't reach the fast convergence of Newton methods

Trust region methods

$$m_k(\mathbf{x}_k + \mathbf{p}) = f_k + \mathbf{p}^T \nabla f_k + \frac{1}{2}\mathbf{p}^T \mathbf{B}_k \mathbf{p}, \quad \text{such that } ||\mathbf{p}|| \leq \Delta_k \tag{36}$$

- interesting algorithms: Levenberg-Marquardt

Trust region vs. line-search



Scaling

Definition

A problem is **poorly scaled** if changes to $\mathbf{x}$ in a certain direction produce much larger variations in the value of f than do changes to $\mathbf{x}$ in another direction.
E.g. $f(\mathbf{x}) = 10^9 x_1^2 + x_2^2$

- steepest descent method is sensitive to poor scaling
- Newton's method is not affected by scaling

Today's Outline

Why study Conjugate Gradient Methods?

- **Conjugate Gradient** methods are useful techniques for solving large *linear* systems of equations.
- **Conjugate Gradient** methods can be adapted to solve *nonlinear* optimization problems.
- the linear method was proposed by Hestens and Stiefel (1950)
- the *nonlinear* method was proposed by Fletcher and Reeves (1960)

Today's Outline

1. Calculus refresher
2. Optimization
3. Optimization Algorithms
4. Conjugate Gradients
   - The Linear Conjugate Gradient Method
   - The nonlinear case

Quadratic Error

- Suppose we have a quadratic error:

$$E(\mathbf{w}) = E_0 + \mathbf{b}^T\mathbf{w} + \frac{1}{2}\mathbf{w}^T\mathbf{H}\mathbf{w} \tag{37}$$

where $\mathbf{b}, \mathbf{H}$ are constant and $\mathbf{H}$ is positive definite

Quadratic Error

- Suppose we have a quadratic error:

$$E(\mathbf{w}) = E_0 + \mathbf{b}^T\mathbf{w} + \frac{1}{2}\mathbf{w}^T\mathbf{H}\mathbf{w} \tag{37}$$

where $\mathbf{b}, \mathbf{H}$ are constant and $\mathbf{H}$ is positive definite

- The local gradient:

$$\nabla E(\mathbf{w}) = g(\mathbf{w}) = \mathbf{b} + \mathbf{H}\mathbf{w} \tag{38}$$

is minimized for:

$$\mathbf{b} + \mathbf{H}\mathbf{w}^* = 0 \tag{39}$$

Quadratic Error

- Suppose we have a quadratic error:

$$E(\mathbf{w}) = E_0 + \mathbf{b}^T\mathbf{w} + \frac{1}{2}\mathbf{w}^T\mathbf{H}\mathbf{w} \tag{37}$$

where $\mathbf{b}, \mathbf{H}$ are constant and $\mathbf{H}$ is positive definite

- The local gradient:

$$\nabla E(\mathbf{w}) = g(\mathbf{w}) = \mathbf{b} + \mathbf{H}\mathbf{w} \tag{38}$$

is minimized for:

$$\mathbf{b} + \mathbf{H}\mathbf{w}^* = 0 \tag{39}$$

- Equation 39 is a linear system.

Conjugate vectors

Definition

A set of vectors $\{\mathbf{d}_1, \mathbf{d}_2, \ldots, \mathbf{d}_W\}$ is said to be **conjugate** with respect to a symmetric positive definite matrix $\mathbf{H}$ if

$$\mathbf{d}_i^T \mathbf{H} \mathbf{d}_j = 0, \quad \forall i \neq j \tag{40}$$

Conjugate vectors

Definition

A set of vectors $\{\mathbf{d}_1, \mathbf{d}_2, \ldots, \mathbf{d}_W\}$ is said to be **conjugate** with respect to a symmetric positive definite matrix $\mathbf{H}$ if

$$\mathbf{d}_i^T \mathbf{H} \mathbf{d}_j = 0, \quad \forall i \neq j \tag{40}$$

Corollary

Any set of **conjugate** *vectors will also be* **linearly independent** *(hence will form a basis set in weight space).*

Importance of conjugacy

Theorem

∇E can be minimized in W steps by successively minimizing it along the individual directions in a conjugate set of W vectors.

- Given
  - a starting vector $\mathbf{w}_0 \in \mathbb{R}^W$ and
  - a set of W conjugate vectors $\{\mathbf{d}_i\}_{0 \leq i < W}$

  let's generate the sequence $\{\mathbf{w}_i\}_{1 \leq i \leq W}$:

$$\mathbf{w}_{j+1} = \mathbf{w}_j + \alpha_j \mathbf{d}_j \tag{41}$$

  where α_j minimizes $\nabla E\left(\mathbf{w}_j + \alpha \mathbf{d}_j\right)$ along $\mathbf{d}_j$

Proof

- Since $\{\mathbf{d}_i\}_{0 \leq i < W}$ form a basis set:

$$\mathbf{w}^* - \mathbf{w}_0 = \sum_{i=0}^{W-1} \sigma_i \mathbf{d}_i \tag{42}$$

Proof

- Since $\{\mathbf{d}_i\}_{0 \leq i < W}$ form a basis set:

$$\mathbf{w}^* - \mathbf{w}_0 = \sum_{i=0}^{W-1} \sigma_i \mathbf{d}_i \tag{42}$$

- For any $j \in \{0, \ldots, W-1\}$, multiplying Equation 42 with $\mathbf{d}_j^T \mathbf{H}$:

$$\mathbf{d}_j^T (\mathbf{H}\mathbf{w}^* - \mathbf{H}\mathbf{w}_0) \tag{43}$$

Proof

- Since $\{\mathbf{d}_i\}_{0 \leq i < W}$ form a basis set:

$$\mathbf{w}^* - \mathbf{w}_0 = \sum_{i=0}^{W-1} \sigma_i \mathbf{d}_i \tag{42}$$

- For any $j \in \{0, \ldots, W-1\}$, multiplying Equation 42 with $\mathbf{d}_j^T \mathbf{H}$:

$$\mathbf{d}_j^T \left(\mathbf{b} + \mathbf{H}\mathbf{w}^* - \mathbf{b} - \mathbf{H}\mathbf{w}_0\right) \tag{43}$$

Proof

- Since $\{\mathbf{d}_i\}_{0 \leq i < W}$ form a basis set:

$$\mathbf{w}^* - \mathbf{w}_0 = \sum_{i=0}^{W-1} \sigma_i \mathbf{d}_i \tag{42}$$

- For any $j \in \{0, \ldots, W-1\}$, multiplying Equation 42 with $\mathbf{d}_j^T \mathbf{H}$:

$$\mathbf{d}_j^T \left(\mathbf{b} + \mathbf{H}\mathbf{w}^* - \mathbf{b} - \mathbf{H}\mathbf{w}_0 \right) \tag{43}$$

Proof

- Since $\{\mathbf{d}_i\}_{0 \leq i < W}$ form a basis set:

$$\mathbf{w}^* - \mathbf{w}_0 = \sum_{i=0}^{W-1} \sigma_i \mathbf{d}_i \tag{42}$$

- For any $j \in \{0, \ldots, W-1\}$, multiplying Equation 42 with $\mathbf{d}_j^T \mathbf{H}$:

$$-\mathbf{d}_j (\mathbf{b} + \mathbf{H}\mathbf{w}_0) = \tag{43}$$

Proof

- Since $\{\mathbf{d}_i\}_{0 \leq i < W}$ form a basis set:

$$\mathbf{w}^* - \mathbf{w}_0 = \sum_{i=0}^{W-1} \sigma_i \mathbf{d}_i \tag{42}$$

- For any $j \in \{0, \ldots, W-1\}$, multiplying Equation 42 with $\mathbf{d}_j^T \mathbf{H}$:

$$-\mathbf{d}_j (\mathbf{b} + \mathbf{H}\mathbf{w}_0) = \sum_{i=0}^{W-1} \sigma_i \mathbf{d}_j^T \mathbf{H} \mathbf{d}_i \tag{43}$$

Proof

- Since $\{\mathbf{d}_i\}_{0 \leq i < W}$ form a basis set:

$$\mathbf{w}^* - \mathbf{w}_0 = \sum_{i=0}^{W-1} \sigma_i \mathbf{d}_i \tag{42}$$

- For any $j \in \{0, \ldots, W-1\}$, multiplying Equation 42 with $\mathbf{d}_j^T \mathbf{H}$:

$$-\mathbf{d}_j (\mathbf{b} + \mathbf{H}\mathbf{w}_0) = \sigma_j \mathbf{d}_j^T \mathbf{H} \mathbf{d}_j \tag{43}$$

Proof

- Since $\{\mathbf{d}_i\}_{0 \leq i < W}$ form a basis set:

$$\mathbf{w}^* - \mathbf{w}_0 = \sum_{i=0}^{W-1} \sigma_i \mathbf{d}_i \tag{42}$$

- For any $j \in \{0, \ldots, W-1\}$, multiplying Equation 42 with $\mathbf{d}_j^T \mathbf{H}$:

$$-\mathbf{d}_j (\mathbf{b} + \mathbf{H}\mathbf{w}_0) = \sigma_j \mathbf{d}_j^T \mathbf{H} \mathbf{d}_j \tag{43}$$

we get explicit solutions for σ_j:

$$\sigma_j = -\frac{\mathbf{d}_j^T (\mathbf{b} + \mathbf{H}\mathbf{w}_0)}{\mathbf{d}_j^T \mathbf{H} \mathbf{d}_j} \tag{44}$$

Who are the coefficients?

- Remember Equation 42:

$$\mathbf{w}_j = \mathbf{w}_0 + \sum_{i=0}^{j-1} \sigma_i \mathbf{d}_i \tag{45}$$

which can be rewritten as:

$$\mathbf{d}_j^T \mathbf{H} \mathbf{w}_j = \mathbf{d}_j^T \mathbf{H} \mathbf{w}_0 \tag{46}$$

Who are the coefficients?

- Remember Equation 42:

$$\mathbf{w}_j = \mathbf{w}_0 + \sum_{i=0}^{j-1} \sigma_i \mathbf{d}_i \tag{45}$$

which can be rewritten as:

$$\mathbf{d}_j^T \mathbf{H} \mathbf{w}_j = \mathbf{d}_j^T \mathbf{H} \mathbf{w}_0 \tag{46}$$

- Using Formula 46, then Formula 44 can be written as:

$$\sigma_j = -\frac{\mathbf{d}_j^T \mathbf{g}_j}{\mathbf{d}_j^T \mathbf{H} \mathbf{d}_j} \tag{47}$$

Who are the coefficients?

- Remember Equation 42:

$$\mathbf{w}_j = \mathbf{w}_0 + \sum_{i=0}^{j-1} \sigma_i \mathbf{d}_i \tag{45}$$

which can be rewritten as:

$$\mathbf{d}_j^T \mathbf{H} \mathbf{w}_j = \mathbf{d}_j^T \mathbf{H} \mathbf{w}_0 \tag{46}$$

- Using Formula 46, then Formula 44 can be written as:

$$\sigma_j = -\frac{\mathbf{d}_j^T \mathbf{g}_j}{\mathbf{d}_j^T \mathbf{H} \mathbf{d}_j} \tag{47}$$

- σ_j represents the solution for minimizing $E(\mathbf{w}_j + \alpha \mathbf{d}_j)$

Orthogonality

- if...

$$\mathbf{g}_{j+1} - \mathbf{g}_j = \mathbf{H}\left(\mathbf{w}_{j+1} - \mathbf{w}_j\right) = \alpha_j \mathbf{H}\mathbf{d}_j \tag{48}$$

multiplying by $\mathbf{d}_j$:

$$\mathbf{d}_j^T \mathbf{g}_{j+1} = 0 \tag{49}$$

mutiplying by $\mathbf{d}_k^T$:

$$\mathbf{d}_k^T(\mathbf{g}_{j+1} - \mathbf{g}_j) = \alpha_j \mathbf{d}_k^T \mathbf{H}\mathbf{d}_j = 0, \forall k < j < W \tag{50}$$

we can demonstrate using induction:

$$\mathbf{d}_k^T \mathbf{g}_j = 0, \quad k < j \leq W \tag{51}$$

How do we choose the conjugate directions?

- **Conjugate Gradient Method** generates a set of *conjugate* directions in the following way:

$$\mathbf{d}_0 = -\mathbf{g}_0 \tag{52}$$

$$\mathbf{d}_{j+1} = -\mathbf{g}_{j+1} + \beta_j \mathbf{d}_j \tag{53}$$

How do we choose the conjugate directions?

- **Conjugate Gradient Method** generates a set of *conjugate* directions in the following way:

$$\mathbf{d}_0 = -\mathbf{g}_0 \tag{52}$$

$$\mathbf{d}_{j+1} = -\mathbf{g}_{j+1} + \beta_j \mathbf{d}_j \tag{53}$$

- Imposing the conjugacy condition:

$$\beta_j = \frac{\mathbf{g}_{j+1}^T \mathbf{H} \mathbf{d}_j}{\mathbf{d}_j^T \mathbf{H} \mathbf{d}_j} \tag{54}$$

Today's Outline

1. Calculus refresher
2. Optimization
3. Optimization Algorithms
4. Conjugate Gradients
   - The Linear Conjugate Gradient Method
   - The nonlinear case

About the CG algorithm

- real problems don't have quadratic error functions
- repeated applications of the algorithm converge to the minimum of the real error
- computing the Hessian matrix is hard
- in practice, solve for α_js using line-search
- use a formula for β_j:
  - the *Hestenes-Stiefel* expression

$$\beta_j = \frac{\mathbf{g}_{j+1}^T(\mathbf{g}_{j+1} - \mathbf{g}_j)}{\mathbf{d}_j^T(\mathbf{g}_{j+1} - \mathbf{g}_j)} \tag{55}$$

  - the *Polak-Ribiere* expression

$$\beta_j = \frac{\mathbf{g}_{j+1}^T(\mathbf{g}_{j+1} - \mathbf{g}_j)}{\mathbf{g}_j^T\mathbf{g}_j} \tag{56}$$

The Conjugate Gradient Algorithm

1. choose an initial weight $\mathbf{w}_1$
2. evaluate $g_1 = \nabla E(\mathbf{w}_1)$
3. set initial diretion $\mathbf{d}_1 = -\mathbf{g}_1$
4. Loop for j:
   1. minimize $E(\mathbf{w}_j + \alpha \mathbf{d}_j)$ with respect to α
   2. $\mathbf{w}_{j+1} = \mathbf{w}_j + \alpha_{\min} \mathbf{d}_j$
   3. evaluate g_{j+1}
   4. compute β_j using Polak-Ribiere
   5. $j \longleftarrow j + 1$

Performance issues

- the performance of the linear **Conjugate Gradient** is determined by the distribution of the eigenvalues of the coefficient matrix
- solution: preconditioning

Today's Outline

References I


Christopher M. Bishop, *Neural networks for pattern recognition*, Oxford University Press, Inc., New York, NY, USA, 1995.


Jorge Nocedal and Stephen Wright, *Numerical optimization*, Springer Science & Business Media, 2006.