“Unconstrained” Optimization for Machine Learning

Dr. Ratna Babu Chinnam

Industrial & Systems Engineering

Wayne State University

Primary Source: Simon Haykin (2009)

Introduction

- **Task**: Find optimal solution $\mathbf{w}^*$ for a differentiable cost function $\xi(\mathbf{w})$

$$\xi(\mathbf{w}^*) \leq \xi(\mathbf{w})$$

- Necessary condition for optimality: $\nabla\xi(\mathbf{w}^*) = \mathbf{0}$

  where $\nabla\xi(\mathbf{w})$ is gradient vector: $\nabla\xi(\mathbf{w}) = \left[\frac{\partial\xi}{\partial w_1}, \frac{\partial\xi}{\partial w_2}, \ldots, \frac{\partial\xi}{\partial w_m}\right]^T$

- A class of "unconstrained" optimization algorithms suited for "error-correction learning" is based on the idea of ***local iterative descent****:*

Starting with initial guess $\boldsymbol{w}(0)$, generate a sequence of vectors $\boldsymbol{w}(1)$, $\boldsymbol{w}(2)$,.., such that cost function $\xi(\boldsymbol{w})$ is reduced at each iteration:

$$\xi(\boldsymbol{w}(n+1)) < \xi(\boldsymbol{w}(n))$$

- Without precautions, there is a possibility of algorithm divergence

Method of “Steepest” Descent (First-Oder Method)

- Adjustments applied to weight vector $\mathbf{w}$ are in the direction of steepest descent (i.e., direction opposite to *gradient vector* $\nabla\xi(\mathbf{w})$):

$$\boldsymbol{w}(n+1) = \boldsymbol{w}(n) - \eta\nabla\xi(\boldsymbol{w})$$

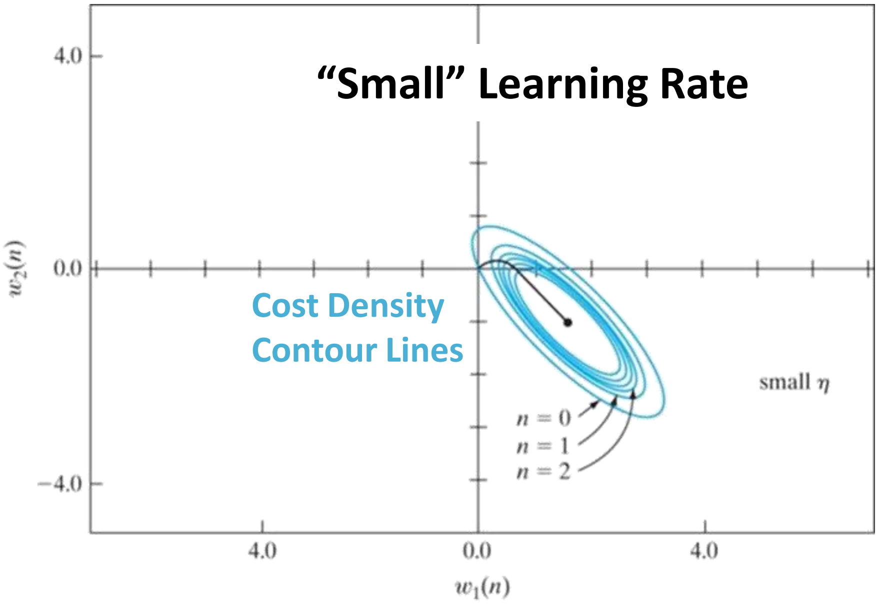
  where η is a +ve constant called *step-size* or ***learning-rate*** parameter.

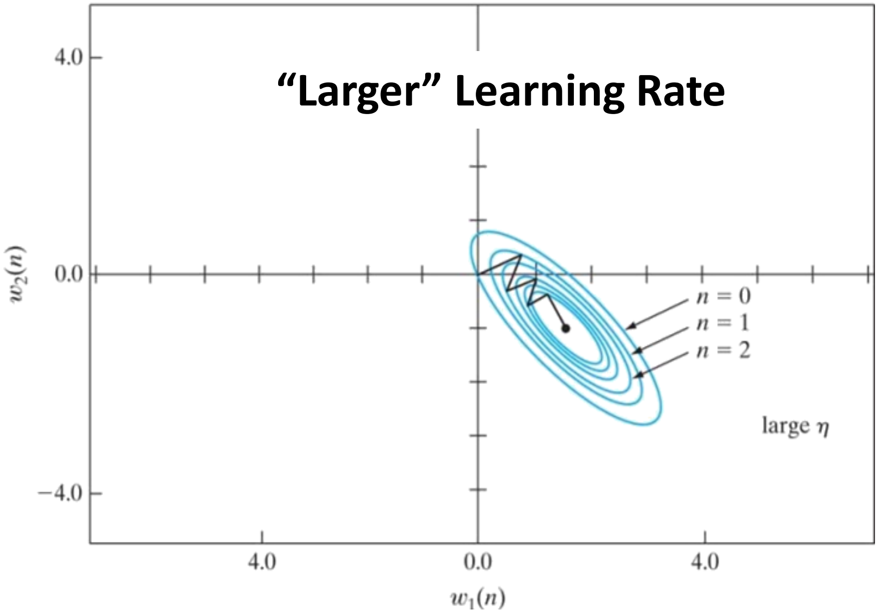
- One can show that technique satisfies condition $\xi(\boldsymbol{w}(n+1)) < \xi(\boldsymbol{w}(n))$ using a “first-order” Taylor series expansion:

$$\begin{aligned} \xi(\mathbf{w}(n+1)) &\simeq \xi(\mathbf{w}(n)) + \nabla\xi(\mathbf{w})^T \Delta\mathbf{w}(n) && \leftarrow \text{Justified for small } \eta \\ &\simeq \xi(\mathbf{w}(n)) - \eta\nabla\xi(\mathbf{w})^T\nabla\xi(\mathbf{w}) \\ &\simeq \xi(\mathbf{w}(n)) - \eta\|\nabla\xi(\mathbf{w})\|^2 \\ &< \xi(\mathbf{w}(n)) && \leftarrow \text{Justified for a +ve } \eta \end{aligned}$$

Method of Steepest Descent (First-Oder Method) …

- **Stochastic Gradient Descent**: Descent based on individual observations rather than overall training dataset
  - Path becomes stochastic for observations are not always in agreement
  - Can help accelerate convergence and even help escape local optima!
- **Steepest descent methods converge slowly:**





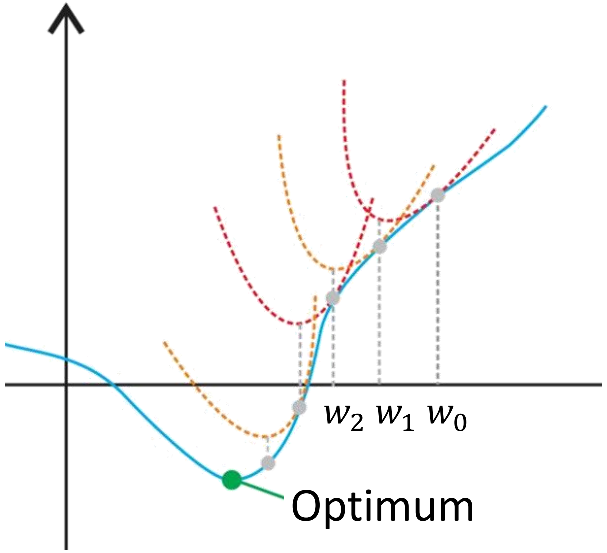
Newton's Method (Second-Order Method)

- Minimizes *quadratic approximation* of cost $\boldsymbol{\xi}(\boldsymbol{w})$ at $\boldsymbol{w}(n)$
- Using a "2nd-order" Taylor series expansion of $\boldsymbol{\xi}(\boldsymbol{w})$around $\mathbf{w}(n)$:

$$\xi(\mathbf{w}(n+1)) - \xi(\mathbf{w}(n)) \simeq \nabla\xi(\mathbf{w})^T \Delta\mathbf{w}(n) + \frac{1}{2}\Delta\mathbf{w}(n)^T \mathbf{H}(n)\Delta\mathbf{w}(n)$$

where $\mathbf{H}(n)$ is $m \times m$ *Hessian* matrix of $\xi(\boldsymbol{w})$ evaluated at $\mathbf{w}(n)$:

$$\mathbf{H}(n) = \nabla^2\xi(\mathbf{w}) = \begin{bmatrix} \frac{\partial^2\xi}{\partial w_1^2} & \cdots & \frac{\partial^2\xi}{\partial w_1 \partial w_m} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2\xi}{\partial w_m \partial w_1} & \cdots & \frac{\partial^2\xi}{\partial w_m^2} \end{bmatrix}$$



- $\boldsymbol{\xi}(\mathbf{w}(n+1))$ can be optimized around current point $\mathbf{w}(n)$ by taking derivative with respect to $\boldsymbol{\Delta}\mathbf{w}$ and equating it to zero:

$$\nabla\xi(\mathbf{w}) + \mathbf{H}(n)\Delta\mathbf{w}(n) = 0$$

- Solving equation for $\Delta\mathbf{w}(n)$ yields: $\Delta\mathbf{w}(n) = -\mathbf{H}^{-1}(n)\nabla\xi(\mathbf{w})$
- Weight update rule: $\boldsymbol{w}(n+1) = \boldsymbol{w}(n) - \boldsymbol{H}^{-1}(n)\nabla\xi(\boldsymbol{w})$

Newton's Method (Second-Order Method) …

- Generally, this method converges quickly and does not exhibit zigzagging behavior observed with method of steepest descent
- However, practical application of Newton’s method is handicapped:
  - Method does not work unless cost function is twice differentiable
  - For $\mathbf{H}^{-\mathbf{1}}(n)$ to be computable, $\mathbf{H}(n)$ has to be positive definite
    - Hessian may also be rank deficient (i.e., columns of $\mathbf{H}$ are not independent); Results from ill- conditioned nature of supervised-learning problems
  - Requires calculation of inverse Hessian $\mathbf{H}^{-\mathbf{1}}(n)$, which can be computationally very expensive

Example: Steepest Descent

Relation: $y(x) = b_0 + b_1x + b_2x^2 + b_3x^3 + b_4x^4$
$$y(x) = 1 + 1.5x - 2x^2 - 1x^3 + 2x^4$$

Gradients: $\nabla y(x) = b_1 + 2b_2x + 3b_3x^2 + 4b_4x^3$
$$\nabla^2 y(x) = 2b_2 + 6b_3x + 12b_4x^2$$

Algorithm: $x(n+1) = x(n) - \eta\, \nabla y(x(n))$

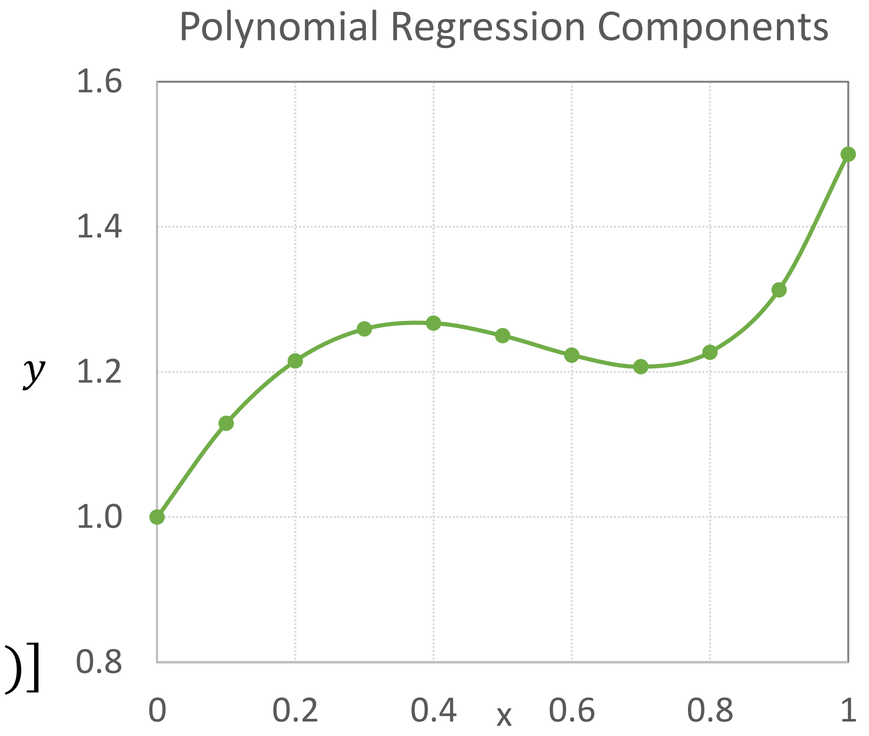
$$x(0) = 0.9; \eta = 0.05$$

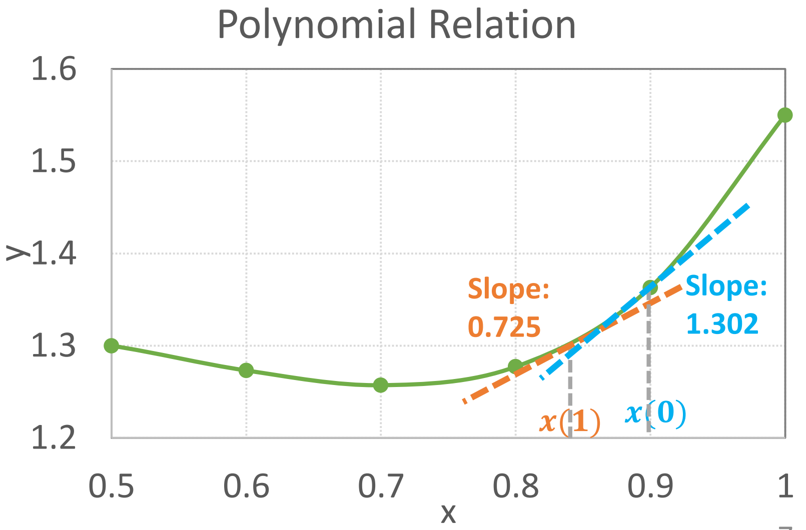
$$x(1) = 0.9 - 0.05[(1.5) + 2(-2)(0.9) + 3(-1)(0.9^2) + 4(2)(0.9^3))]$$
$$= 0.9 - 0.05[1.302] = 0.835$$
$$x(2) = 0.835 - 0.05[(1.5) + 2(-2)(0.835) + 3(-1)(0.835^2) + 4(2)(0.835^3))]$$
$$= 0.835 - 0.05[0.725] = 0.799$$
$$x(3) = 0.799 - 0.05[(1.5) + 2(-2)(0.799) + 3(-1)(0.799^2) + 4(2)(0.799^3))]$$
$$= 0.799 - 0.05[0.467] = 0.775$$

...

$$x(14) = 0.714 - 0.05[(1.5) + 2(-2)(0.714) + 3(-1)(0.714^2) + 4(2)(0.714^3))]$$
$$= 0.714 - 0.05[0.025] = 0.712$$
$$x(15) = 0.712 - 0.05[(1.5) + 2(-2)(0.712) + 3(-1)(0.712^2) + 4(2)(0.712^3))]$$
$$= 0.712 - 0.05[0.020] = 0.711$$

Problem: Stuck at local optimal solution!





Example: Newton's Method

Relation:

$$y(x) = b_0 + b_1 x + b_2 x^2 + b_3 x^3 + b_4 x^4$$
$$y(x) = 1 + 1.5x - 2x^2 - 1x^3 + 2x^4$$

Gradients:

$$\nabla y(x) = b_1 + 2b_2 x + 3b_3 x^2 + 4b_4 x^3$$
$$\nabla^2 y(x) = 2b_2 + 6b_3 x + 12 b_4 x^2$$

Algorithm:

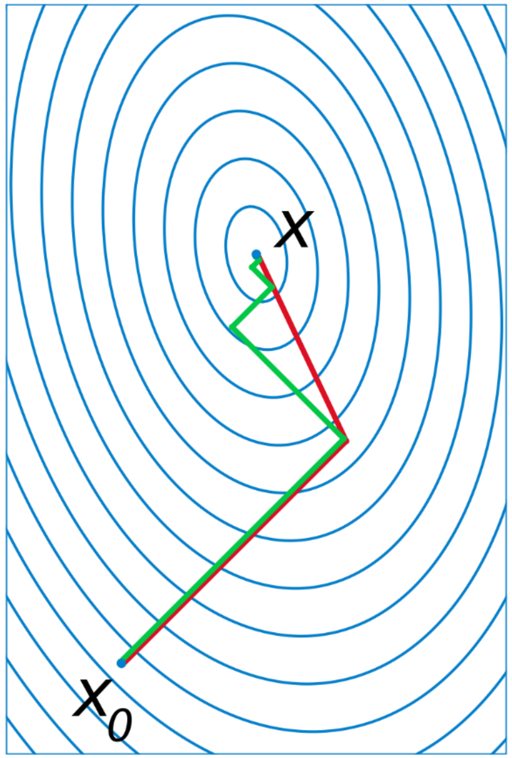
$$x(0) = 0.9; \eta = 0.05$$
$$\nabla y(x) + \mathbf{H}(n) \Delta x(n) = 0 \quad \mathbf{H}(n) = \nabla^2 y(x)$$
$$\Delta x(n) = -\frac{b_1 + 2b_2 x + 3b_3 x^2 + 4b_4 x^3}{2b_2 + 6b_3 x + 12 b_4 x^2}$$
$$\Delta x(0) = -\frac{1.5 + 2(-2)(0.9) + 3(-1)(0.9^2) + 4(2)(0.9^3)}{2(-2) + 6(-1)(0.9) + 12(2)(0.9^2)} = -0.13$$
$$x(1) = x(0) + \Delta x(0) = 0.9 - 0.13 = 0.77$$
$$x(2) = x(1) + \Delta x(1) = 0.77 - 0.053 = 0.718$$
$$x(3) = x(2) + \Delta x(2) = 0.718 - 0.010 = 0.708$$
$$x(4) = x(3) + \Delta x(3) = 0.708 - 0.0004 = 0.707$$

Problem: Fewer iterations but still stuck at local optimum!

Why didn't we use $\mathbf{H}^{-1}(n)$?
X is 1-dimensional

Conjugate-Gradient Method

- An intermediate approach between method of steepest descent and Newton's method
- **Goal**: Accelerate slow rate of convergence experienced with steepest descent while avoiding computational burden of Newton's method
- Employs one-dimensional line-search as a part of the method



Comparison of convergence of gradient descent with optimal step size (green) and conjugate vector (red)

Gauss-Newton Method (Least Squares Problems)

- Method can <u>only</u> be used to minimize sum of squared function values
  - Advantage: Second derivatives (challenging to compute) are not required
- Suppose cost function is a sum of squares (e.g., "batch" training):

$$\xi(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{m} e^2(i)$$

- Given a point $w(n)$, dependence of $e(i)$ on w is linearized as follows:

$$e^{lin}(i, \boldsymbol{w}) = e(i) + \left[\frac{\partial e(i)}{\partial \boldsymbol{w}}\right]^T_{\boldsymbol{w}=\boldsymbol{w}(n)} \big(\boldsymbol{w} - \boldsymbol{w}(n)\big) \quad i = 1,2,\ldots,m$$

- Using matrix notation: $e^{lin}(n, \boldsymbol{w}) = e(n) + \boldsymbol{J}(n)(\boldsymbol{w} - \boldsymbol{w}(n))$
- $\boldsymbol{J}(n)$ is $m \times p$ *Jacobian matrix* of $e(n)$: $\boldsymbol{J}(n) = \begin{bmatrix} \frac{\partial e(1)}{\partial w_1} & \cdots & \frac{\partial e(1)}{\partial w_p} \\ \vdots & \ddots & \vdots \\ \frac{\partial e(m)}{\partial w_1} & \cdots & \frac{\partial e(m)}{\partial w_p} \end{bmatrix}_{w=w(n)}$

Gauss-Newton Method (Least Squares Problems) …

- Updated weight vector is defined by: $\mathbf{w}(n+1) = \underset{\mathbf{w}}{\operatorname{argmin}} \left\{\frac{1}{2}\left\|e^{\mathrm{lin}}(n,\mathbf{w})\right\|^2\right\}$

$$= \underset{\mathbf{w}}{\operatorname{argmin}} \left\{\frac{1}{2}\|\mathbf{e}(n)\|^2 + \mathbf{e}^T(n)\mathbf{J}(n)(\mathbf{w}-\mathbf{w}(n)) + \frac{1}{2}(\mathbf{w}-\mathbf{w}(n))^T\mathbf{J}(n)(\mathbf{w}-\mathbf{w}(n))\right\}$$

- Differentiating argument with respect $\mathbf{w}$ and equating to zero:

$$\mathbf{J}^T(n)\mathbf{e}(n) + \mathbf{J}^T(n)\mathbf{J}(n)(\mathbf{w}-\mathbf{w}(n)) = \mathbf{0}.$$

- Solving for $\mathbf{w}$, we can write “pure form” of update rule as follows:

$$\mathbf{w}(n+1) = \mathbf{w}(n) - (\mathbf{J}^T(n)\mathbf{J}(n))^{-1}\mathbf{J}^T(n)\mathbf{e}(n)$$

- Generally applied in the following “modified form” to avoid singularity:

$$\mathbf{w}(n+1) = \mathbf{w}(n) - (\mathbf{J}^T(n)\mathbf{J}(n) + \delta\boldsymbol{I})^{-1}\mathbf{J}^T(n)\mathbf{e}(n)$$

  $\boldsymbol{\delta}$ is a small +ve constant that ensures $\mathbf{J}^{\boldsymbol{T}}(\boldsymbol{n})\mathbf{J}(\boldsymbol{n}) + \boldsymbol{\delta}\boldsymbol{I}$ is positive definite

- Effect of added term $\boldsymbol{\delta}\boldsymbol{I}$ is progressively reduced as n is increased (by decreasing $\boldsymbol{\delta}$ with *n*)

Final Comments

- Steepest descent, Newton, and quasi-Newton methods can minimize general real-valued functions
- Gauss-Newton and Levenberg-Marquardt methods <u>only</u> handle nonlinear least-squares problems
- Levenberg-Marquardt interpolates between the Gauss–Newton algorithm and the method of gradient descent
  - More robust than Gauss–Newton
  - For well-behaved functions, tends to be a bit slower than Gauss–Newton
- Bayesian Regularization technique minimize a linear combination of squared errors and weights
  - Employs Levenberg-Marquardt algorithm
- **In practice, Conjugate-Gradient, Levenberg-Marquardt, and Bayesian Regularlization methods are quite effective!**