UNCONSTRAINED OPTIMIZATION TECHNIQUES (NON-LINEAR & CONTINUOUS)

Source: SECTIONS 3.3 AND 4.16 OF Haykin, 2009

- The task is to find an optimal solution $\mathbf{w}^*$ for a continuously differentiable cost function $\xi(\mathbf{w})$ that satisfies the following condition: $\xi(\mathbf{w}^*) \le \xi(\mathbf{w})$.
- The necessary condition for optimality is $\nabla\xi(\mathbf{w}^*) = \mathbf{0}$

  where $\nabla\xi(\mathbf{w})$ is the gradient vector of the cost function:

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

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

  Starting with an initial guess denoted by $\mathbf{w}(0)$, *generate a sequence of weight vectors* $\mathbf{w}(1)$, $\mathbf{w}(2)$, *.., such that the cost function* $\xi(\mathbf{w})$ *is reduced at each iteration of the algorithm, as shown by*

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

  where $\mathbf{w}(n)$ *is the old value of weight vector and* $\mathbf{w}(n+1)$ *is its updated value.*
- Without special precautions, there is a distinct probability of algorithm divergence.

Method of Steepest Descent (First-Oder Method):

- Here, successive adjustments applied to the weight vector $\mathbf{w}$ are in the direction of steepest descent, that is, in a direction opposite to the *gradient vector* $\nabla\xi(\mathbf{w})$:

  $\mathbf{w}(n+1) = \mathbf{w}(n) - \eta\nabla\xi(\mathbf{w})$ where η is a positive constant called the *step-size* or *learning-rate parameter*.
- One can show that the technique satisfies the condition $\xi(\mathbf{w}(n+1)) < \xi(\mathbf{w}(n))$ using a first-order Taylor series expansion:

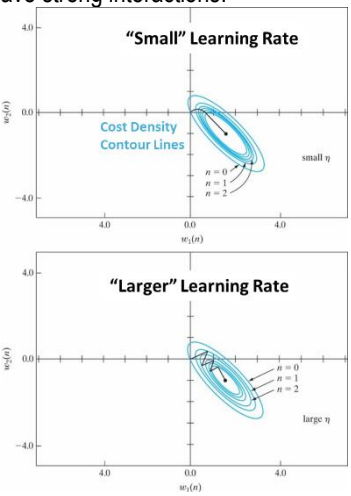
$$\xi(\mathbf{w}(n+1)) \simeq \xi(\mathbf{w}(n)) + \nabla\xi(\mathbf{w})^T\Delta\mathbf{w}(n) \quad \leftarrow \text{Justified for small } \eta$$

$$\simeq \xi(\mathbf{w}(n)) - \eta\nabla\xi(\mathbf{w})^T\nabla\xi(\mathbf{w})$$

$$\simeq \xi(\mathbf{w}(n)) - \eta\left\|\nabla\xi(\mathbf{w})\right\|^2$$

$$< \xi(\mathbf{w}(n)) \quad \leftarrow \text{Justified for a positive } \eta$$

- However, method of steepest descent converges very slowly.
- It is highly inefficient for many functions, especially if the parameters have strong interactions.



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

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

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

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

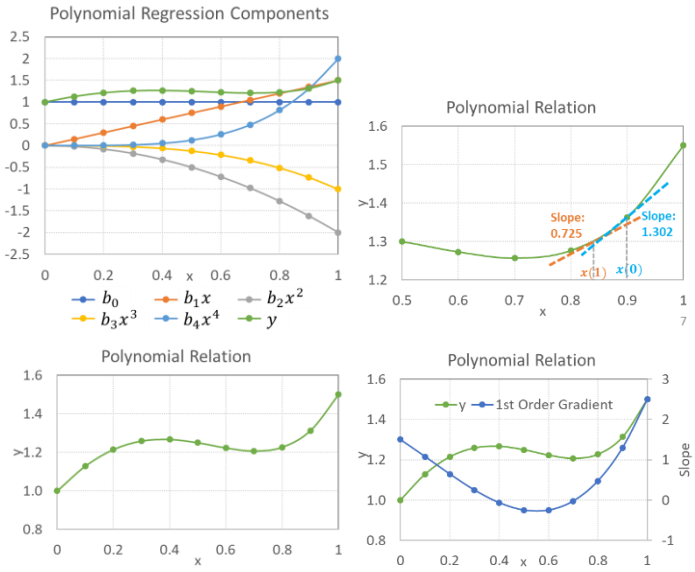
$$x(3) = 0.799 - 0.05[(1.5) + 2(-2)(0.799) + 3(-1)(0.799^2) + 4(2)(0.799^3))] = 0.799 - 0.05[0.467] = 0.775$$

...

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

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

Problem: Stuck at local optimal solution!



- What do you suppose will happen if we change $\boldsymbol{x_0}$ or $\boldsymbol{\eta}$?

Newton's Method (Second-Order Method):

- Minimizes the *quadratic approximation* of the cost function $\xi(\mathbf{w})$ around the current point $\mathbf{w}(n)$; this minimization is performed in every iteration.
- Using a second-order Taylor series expansion of $\xi(\mathbf{w})$ 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 matrix $\mathbf{H}(n)$ is the $m \times m$ *Hessian* matrix of $\xi(\mathbf{w})$ evaluated at $\mathbf{w}(n)$ and defined as follows:

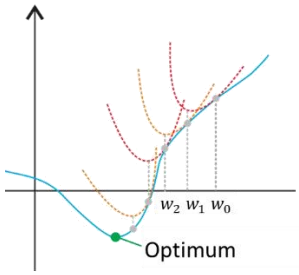
$$\mathbf{H}(n) = \nabla^2\xi(\mathbf{w}) = \begin{bmatrix} \frac{\partial^2\xi}{\partial w_1^2} & \cdots & \frac{\partial^2\xi}{\partial w_1\partial w_m} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2\xi}{\partial w_m\partial w_1} & \cdots & \frac{\partial^2\xi}{\partial w_m^2} \end{bmatrix}$$

- The quadratic approximated cost function $\xi(\mathbf{w}(n+1))$ can be optimized around the current point $\mathbf{w}(n)$ by taking the derivative

with respect to $\Delta\mathbf{w}$ and equating it to zero. This results in the following equation: $\nabla\xi(\mathbf{w}) + \mathbf{H}(n)\Delta\mathbf{w}(n) = \mathbf{0}$.

Solving the equation for $\Delta\mathbf{w}(n)$ yields $\Delta\mathbf{w}(n) = -\mathbf{H}^{-1}(n)\nabla\xi(\mathbf{w})$

The weight update rule is, $\mathbf{w}(n+1) = \mathbf{w}(n) - \mathbf{H}^{-1}(n)\nabla\xi(\mathbf{w})$



- Generally speaking, this method converges quickly and does *not* exhibit zigzagging behavior that sometimes is observed with the method of steepest descent.
- However, the practical application of Newton's method is handicapped by several factors:
  1. Method does not work unless the cost function is twice continuously differentiable. Not too bad an assumption.
  2. For $\mathbf{H}^{-1}(n)$ to be computable, $\mathbf{H}(n)$ has to be positive definite at every iteration of the algorithm.
     In the case where $\mathbf{H}(n)$ is positive definite, the error surface around the current point $\mathbf{w}(n)$ is describable by a "convex bowl." Unfortunately, there is no guarantee that the Hessian of the error surface will always fit this description. Moreover, there is the potential problem of the Hessian being rank deficient (i.e., not all the columns of $\mathbf{H}(n)$ are linearly independent), which results from the intrinsically ill-conditioned nature of supervised-learning problems, making the computational task more difficult.
  1. Requires calculation of the inverse Hessian $\mathbf{H}^{-1}(n)$, which can be computationally very expensive; Computational complexity is of order $O(W^2)$, where W is the size of weight vector $\mathbf{w}$.

Example: Newton's Method

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

$$\nabla y(x) + \mathbf{H}(n)\Delta x(n) = 0 \quad \mathbf{H}(n) = \nabla^2 y(x)$$

$$\Delta x(n) = -\frac{b_1 + 2b_2x + 3b_3x^2 + 4b_4x^3}{2b_2 + 6b_3x + 12b_4x^2}$$

$$\Delta x(0) = -\frac{1.5 + 2(-2)(0.9) + 3(-1)(0.9^2) + 4(2)(0.9^3)}{2(-2) + 6(-1)(0.9) + 12(2)(0.9^2)} = -0.13$$

$$x(1) = x(0) + \Delta x(0) = 0.9 - 0.13 = 0.77$$

$$x(2) = x(1) + \Delta x(1) = 0.77 - 0.053 = 0.718$$

$$x(3) = x(2) + \Delta x(2) = 0.718 - 0.010 = 0.708$$

$$x(4) = x(3) + \Delta x(3) = 0.708 - 0.0004 = 0.707$$

Problem: Fewer iterations but still stuck at local optimum!

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

Quasi-Newton Methods (see Section 4.16, pg. 194 of Haykin, 2009):

- To overcome some of the difficulties of the Newton's method, we may use the quasi-Newton methods, which requires only an estimate of the first-order gradient $\nabla\xi(\mathbf{w})$.
- These methods use second-order (curvature) information about the error surface without actually requiring knowledge of the Hessian.
- They do so by using two successive iterates $\mathbf{w}(n)$ and $\mathbf{w}(n+1)$, together with the respective gradient vectors $\nabla\xi(\mathbf{w}(n))$ and $\nabla\xi(\mathbf{w}(n+1))$.
- The full Hessian is built-up "numerically" using first derivatives only so that after a finite number of refinement cycles (depends on the problem structure), the method closely approximates to Newton's method in performance.
- This modification of Newton's method maintains a positive-definite estimate of the inverse matrix H- 1 directly without matrix inversion.
  - By using such an estimate, a quasi-Newton method is assured of going down-hill on the error surface.
- Employs one-dimensional line-search as a part of the method.
- Has high computational complexity of order $O(W^2)$, where W is the size of weight vector $\mathbf{w}$.
- Hence, quasi-Newton methods are good, in practice, to design of small- scale neural networks.

Conjugate-Gradient Method (see Section 4.16, pg. 188 of Haykin, 2009):

- May be regarded as being somewhat intermediate between the method of steepest descent and Newton's method.
- Use of the conjugate-gradient method is motivated by the desire to accelerate the typically slow rate of convergence experienced with the method of steepest descent, while avoiding the computational requirements associated with the evaluation, storage, and inversion of the Hessian in Newton's method.
- Employs one-dimensional line-search as a part of the method.

Comparison of Quasi-Newton Methods with Conjugate-Gradient Methods (see Section 4.16, pg. 196 of Haykin, 2009):

- Both quasi-Newton and conjugate-gradient methods avoid the need to use the Hessian.
- However, quasi-Newton methods go one step further by generating an approximation to the inverse Hessian.
- Accordingly, when the line search is accurate and we are in close proximity to a local minimum with a positive-definite Hessian, a quasi-Newton method tends to approximate Newton's method, thereby attaining faster convergence than would be possible with the conjugate- gradient method.
- Quasi-Newton methods are not as sensitive to accuracy in the line-search stage of the optimization as the conjugate-gradient method.
- Quasi-Newton methods require significant storage and has high computational complexity of order $O(W^2)$, where W is the size of weight vector $\mathbf{w}$. In contrast, the computational complexity of the conjugate-gradient method is O(W).
- Thus, when the dimension W (i. e., size of the weight vector w) is large, conjugate-gradient methods are preferable to quasi-Newton methods in computational terms.

Gauss-Newton Method (Notation is Intentionally Different from Section 3.3, pg. 98 of Haykin, 2009):

- Is a method only applicable to solve non-linear least squares problems
  - Non-linear least squares problems arise for instance in non-linear regression, where parameters in a model are sought such that the model is in good agreement with available observations
- It can be seen as a modification of Newton's method (which can find minimum of a function)
- Unlike Newton's method, the Gauss–Newton method can only be used to minimize a sum of squared function values, but it has the

advantage that second derivatives, which can be challenging to compute, are not required.

- Let us suppose that the cost function is expressed as a sum of squares (e.g., sum of error squares):

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

where all the error terms are calculated on the basis of a weight vector $\mathbf{w}$ fixed over the entire set of m observations or interval $1 \le i \le m$ (batch training).

- Given an operating point $\mathbf{w}(n)$, the dependence of $e(i)$ on $\mathbf{w}$ is linearized as follows:

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

- Equivalently, using matrix notation:
  $\mathbf{e}^{\text{lin}}(n,\mathbf{w}) = \mathbf{e}(n) + \mathbf{J}(n)(\mathbf{w} - \mathbf{w}(n))$
  and $\mathbf{J}(n)$ is the $m \times p$ *Jacobian matrix* of $\mathbf{e}(n)$:

$$\mathbf{J}(n) = \begin{bmatrix} \frac{\partial e(1)}{\partial w_1} & \cdots & \frac{\partial e(1)}{\partial w_m} \\ \vdots & \ddots & \vdots \\ \frac{\partial e(m)}{\partial w_1} & \cdots & \frac{\partial e(m)}{\partial w_m} \end{bmatrix}_{\mathbf{w}=\mathbf{w}(p)}$$

- The updated weight vector is defined by:

$$\mathbf{w}(n+1) = \arg\min_{\mathbf{w}} \left\{ \frac{1}{2} \left\| e^{\text{lin}}(n,\mathbf{w}) \right\|^2 \right\}$$

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

- Differentiating the argument with respect to $\mathbf{w}$ and equating to zero, we obtain:
  $\mathbf{J}^T(n)\mathbf{e}(n) + \mathbf{J}^T(n)\mathbf{J}(n)(\mathbf{w} - \mathbf{w}(n)) = \mathbf{0}$.
- Solving the equation for $\mathbf{w}$, we can write the "pure form" of the update rule as follows:

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

- The method is generally applied in the following "modified form":

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

  δ is a small +ve constant that ensures $\mathbf{J}^T(n)\mathbf{J}(n) + \delta \mathrm{I}$ is positive definite for all n.
- The resulting modified cost function is given below:

$$\xi(\mathbf{w}) = \frac{1}{2}\left\{ \delta \left\| \mathbf{w} - \mathbf{w}(0) \right\|^2 + \sum_{i=1}^{n} \mathrm{e}^2(i) \right\}.$$

- The effect of the added term $\delta \mathrm{I}$ is progressively reduced as the number of iterations, n, is increased (by decreasing δ with n).

Final Comments:

- Steepest descent, Newton, and quasi-Newton methods can minimize general real-valued functions
- Gauss-Newton and Levenberg-Marquardt methods only handle to 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 minimizes 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!