Lecture 11 – ECE 275A

Generalized Gradient Descent Algorithms
**Continuous Dynamic Minimization of \( \ell(x) \)**

Let \( \ell(\cdot) : X = \mathbb{R}^n \rightarrow \mathbb{R} \) be twice differentiable with respect to \( x \in \mathbb{R}^n \) and bounded from below by 0, \( \ell(x) \geq 0 \) for all \( x \). Recalling that \( \dot{\ell}(x) = \frac{\partial \ell(x)}{\partial x} \dot{x} \), set

\[
\dot{x} = -Q(x) \nabla_x^c \ell(x) \quad \text{with} \quad \nabla_x^c \ell(x) = \left( \frac{\partial \ell(x)}{\partial x} \right)^T = \text{Cartesian gradient of } \ell(x)
\]

and \( Q(x) = Q(x)^T \) for all \( x \). This yields

\[
\dot{\ell}(x) = \| \nabla_x^c \ell(x) \|^2_Q (x) \leq 0
\]

with

\[
\dot{\ell}(x) = 0 \quad \text{if and only if} \quad \nabla_x^c \ell(x) = 0
\]

Thus continuously evolving the value of \( x \) according to \( \dot{x} = -Q(x) \nabla_x^c \ell(x) \) ensures that the cost \( \ell(x) \) dynamically decreases in value until a stationary point turns off learning.
**Generalized Gradient Descent Algorithm**

A family of algorithms for discrete-step dynamic minimization of $\ell(x)$ can be developed as follows.

Setting $x_k = x(t_k)$, approximate $\dot{x}_k = \dot{x}(t_k)$ by the first-order forward difference

$$\dot{x}_k \approx \frac{x_{k+1} - x_k}{t_{k+1} - t_k}$$

This yields

$$x_{k+1} \approx x_k + \alpha_k \dot{x}_k \quad \text{with} \quad \alpha_k = t_{k+1} - t_k$$

which suggests the **Generalized Gradient Descent Algorithm**

$$\hat{x}_{k+1} = \hat{x}_k - \alpha_k Q(\hat{x}_k) \nabla^c_x \ell(\hat{x}_k)$$

A vast body of literature in Mathematical Optimization Theory (aka Mathematical Programming) exists which gives conditions on step size $\alpha_k$ to guarantee that a generalized gradient descent algorithm will converge to a stationary value of $\ell(x)$ for various choices of $Q(x) = Q(x)^T > 0$.

Note that a Generalized Gradient Algorithm turns off once the sequence of estimates has converged to a stationary point of $\ell(x)$: $\hat{x}_k \to \hat{x}$ with $\nabla^c_x \ell(\hat{x}) = 0$. 
**Generalized Gradient Descent Algorithm – Cont.**

**IMPORTANT SPECIAL CASES:**

\[
Q(x) = I \quad \text{Gradient Descent Algorithm}
\]

\[
Q(x) = \mathcal{H}^{-1}(x) \quad \text{Newton Algorithm}
\]

\[
Q(x) = \Omega^{-1}_x \quad \text{Natural Gradient Algorithm}
\]

- The (Cartesian or naive) Gradient Descent Algorithm is simplest to implement, but slowest to converge.

- The Newton Algorithm is most difficult to implement, due to the difficulty in constructing and inverting the Hessian, but fastest to converge.

- The Natural (or true) Gradient Descent Algorithm provides improved convergence speed over (naive) gradient descent.
Derivation of the Newton Algorithm

A Taylor series expansion of $\ell(x)$ about a current estimate of a minimizing point $\hat{x}_k$ to second-order in $\Delta x = x - \hat{x}_k$ yields

$$
\ell(\hat{x}_k + \Delta x) \approx \ell(\hat{x}_k) + \frac{\partial \ell(\hat{x}_k)}{\partial x} \Delta x + \frac{1}{2} \Delta x^T \mathcal{H}(x_0) \Delta x
$$

Minimizing the above wrt $\Delta x$ results in

$$
\hat{\Delta}x_k = -\mathcal{H}^{-1}(\hat{x}_k) \nabla^c_x \ell(\hat{x}_k)
$$

Finally, updating the minimizing point estimate via

$$
\hat{x}_{k+1} = \hat{x}_k + \alpha_k \hat{\Delta}x_k = \hat{x}_{k+1} = \hat{x}_k - \mathcal{H}^{-1}(\hat{x}_k) \nabla^c_x \ell(\hat{x}_k)
$$

yields the Newton Algorithm.

As mentioned, the Newton Algorithm generally yields fast convergence. This is particularly true if it can be stabilized using the so-called Newton step-size $\alpha_k = 1$. 
**Nonlinear Least-Squares**

For the Linear Gaussian model, with known covariance matrix $C = W^{-1}$, the negative log-likelihood is

$$\ell(x) = -\log p_x(y) = \frac{1}{2} \| y - h(x) \|^2_W$$

The (Cartesian) gradient is given by $\nabla^c_x \ell(x) = \left( \frac{\partial \ell(x)}{\partial x} \right)^T$, or

$$\nabla^c_x \ell(x) = -H^T(x)W(y - h(x))$$

where $H(x)$ is the Jacobian (linearization) of $h(x)$

$$H(x) = \frac{\partial h(x)}{\partial x}$$

This yields the

**Nonlinear Least-Squares Generalized Gradient Descent Algorithm**

$$\hat{x}_{k+1} = \hat{x}_k + \alpha_k Q(\hat{x}_k)H^T(\hat{x}_k) (y - h(\hat{x}_k))$$
Nonlinear Least-Squares – Cont.

Note that the Natural (true) gradient of $\ell(x)$ is given by

$$\nabla_x \ell(x) = -\Omega_x^{-1} H^T(x) W (y - h(x)) = -H^*(x) (y - h(x))$$

where $H^*(x)$ is the adjoint of $H(x)$.

A stationary point $x$, $\nabla_x \ell(x) = 0$, must satisfy the Nonlinear Normal Equation

$$H^*(x) h(x) = H^*(x) y$$

and the prediction error $e(x) = y - h(x)$ must be in $\mathcal{N}(H^*(x)) = \mathcal{R}(H(x))^\perp$

$$H^*(x) e(x) = H^*(x) (y - h(x)) = 0$$

Provided that the step size $\alpha_k$ is chosen to stabilize the nonlinear least-squares generalized gradient descent algorithm, the sequence of estimates $\hat{x}_k$ will converge to a stationary point of $\ell(x)$, $\hat{x}_k \rightarrow \hat{x}$ with $\nabla_x \ell(\hat{x}) = 0 = \nabla_x^c \ell(\hat{x})$. 
Nonlinear Least-Squares – Cont.

IMPLEMENTING THE NEWTON ALGORITHM

One computes the Hessian from

\[ \mathcal{H}(x) = \frac{\partial}{\partial x} \left( \frac{\partial \ell(x)}{\partial x} \right)^T = \frac{\partial}{\partial x} \nabla^c_x \ell(x) \]

This yields the

**Hessian for the Weighted Least-Squares Loss Function**

\[ \mathcal{H}(x) = H^T(x) WH(x) - \sum_{i=1}^{m} \mathcal{H}^i(x) \left[ W (y - h(x)) \right]_i \]

where

\[ \mathcal{H}^i(x) \triangleq \frac{\partial}{\partial x} \left( \frac{\partial h_i(x)}{\partial x} \right)^T \]

denotes the Hessian of the the \(i\)-th scalar-valued component of the vector function \(h(x)\).

Note that all terms on the right-hand-side of of the Hessian expression are symmetric, as required if \(\mathcal{H}(x)\) is to be symmetric.
Nonlinear Least-Squares – Cont.

• Evidently, the Hessian matrix of the least-squares loss function $\ell(x)$ can be quite complex. Also note that because of the second term on the right-hand-side of the Hessian expression, $\mathcal{H}(x)$ can become singular or indefinite.

• However, as we have seen in the special case when $h(x)$ is linear, $h(x) = Hx$, we have that $H(x) = H$ and $\frac{\partial H_i(x)}{\partial x} = 0$, $i = 1, \cdots n$, yielding,

$$\mathcal{H}(x) = H^T WH,$$

which for full column-rank $A$ and positive $W$ is always symmetric and invertible.

• Also note that if we have a good model and a value $\hat{x}$ such that the prediction error $e(\hat{x}) = y - h(\hat{x}) \approx 0$, then

$$\mathcal{H}(\hat{x}) \approx H^T(\hat{x})WH(\hat{x})$$
Nonlinear Least-Squares – Gauss-Newton Algorithm

Linearizing $h(x)$ about a current estimate $\hat{x}_k$ with $\Delta x = x - \hat{x}_k$, we have

$$h(x) \approx h(\hat{x}_k) + \frac{\partial}{\partial x} h(\hat{x}_k) \Delta x = h(\hat{x}_k) + H(\hat{x}_k) \Delta x$$

This yields the loss-function approximation

$$\ell(x) = \ell(\hat{x}_k + \Delta x) \approx \frac{1}{2} \| (y - h(\hat{x}_k)) - H(\hat{x}_k) \Delta x \|^2_W$$

Assuming that $H(\hat{x}_k)$ has full column rank (which is guaranteed if $h(x)$ is one-to-one in a neighborhood of $\hat{x}_k$), then we can minimize $\ell(\hat{x}_k + \Delta x)$ wrt $\Delta x$ to obtain

$$\hat{\Delta} x_k = \left( H^T(\hat{x}_k) W H(\hat{x}_k) \right)^{-1} H^T(\hat{x}_k) W (y - h(\hat{x}_k))$$

The update rule $\hat{x}_{k+1} = \hat{x}_k + \alpha_k \hat{\Delta} x_k$ yields the a nonlinear least-squares generalized gradient descent algorithm known as the Gauss-Newton Algorithm.
Gauss-Newton Algorithm – Cont.

The Gauss-Newton algorithm corresponds to taking the weighting matrix \( Q(x) = Q^T(x) > 0 \) in the nonlinear least-squares generalized gradient descent algorithm to be (under the assumption that \( h(x) \) is locally one-to-one)

\[
Q(x) = \left( H^T(x) W H(x) \right)^{-1}
\]

Gauss-Newton Algorithm

- Note that when \( e(x) = y - h(x) \approx 0 \), the Newton and Gauss-Newton algorithms become essentially equivalent.

- Thus it is not surprising that the Gauss-Newton algorithm can result in very fast convergence, assuming that \( e(\hat{x}_k) \) becomes asymptotically small.

- This is particularly true if the Gauss-Newton algorithm can be stabilized using the Newton step-size \( \alpha_k = 1 \).