Supervised Learning Algorithms

- Gradient Descent (GD) Method

. weight update rule:

$$\underline{w}_{k+1} = \underline{w}_k - \mu \nabla_k$$

where $\underline{w}_{k+1}$ represents the $k+1$th $n-$dimensional weight vector.

true gradient (batch mode): $\nabla_k = 2R\underline{w}_k - 2P$

stochastic gradient (on-line mode): $\widehat{\nabla}_k = -2\epsilon_k \underline{x}_k$

. convergence:

$$0 < \mu < 1/\lambda_{\max} \quad \text{or} \quad \mu \propto 1/k$$

In general, this method is too slow.

. computational complexity:

$O(n^2)/step$ in the batch mode

$O(n)/step$ in the on-line mode

- Newton Method

. weight update rule:

$$\underline{w}_{k+1} = \underline{w}_k - \mu R^{-1} \nabla_k$$

Since $\nabla_k = 2R\underline{w}_k - 2P$ in the batch mode,

$$\underline{w}_{k+1} = (1-2\mu)\underline{w}_k + 2\mu R^{-1} P = (1-2\mu)\underline{w}_k + 2\mu \underline{w}^*, \text{ that is,}$$

$$\underline{w}_k = \underline{w}^* + (1-2\mu)^k (\underline{w}_0 - \underline{w}^*).$$

. convergence:

If $\mu = 1/2$, $\underline{w}_1 = \underline{w}^*$, that is, one-step convergence.

. computational complexity:

The inversion of R is required, that is,

$$O(n^3)/step$$

- Quasi-Newton Method

. weight update rule in Newton method:

$$\underline{w}_{k+1} = \underline{w}_k - \alpha_k R^{-1} \nabla_k$$

$\alpha_k = 1/2$ for one-step convergence.

. Quasi-Newton method performs the recursive construction of R^{-1}. Here, we assume that R is symmetric and positive definite matrix.

. Let us consider the following equation:

$$R\underline{p}_i = \underline{q}_i, \quad i = 0, 1, \cdots, n-1.$$

Consider a matrix S_{k+1} such that

$$S_{k+1}\underline{q}_i = \underline{p}_i, \quad i = 0, 1, \cdots, k \quad \text{(Quasi-Newton condition)}$$

After n linearly independent steps, $S_n = R^{-1}$.

construction of S_{k+1}:

$$S_{k+1} = S_k + \alpha_k \underline{r}_k \underline{r}_k^T \quad \text{(rank one correlation) ... (1)}$$

$$\underline{p}_k = S_{k+1}\underline{q}_k = S_k\underline{q}_k + \alpha_k \underline{r}_k \underline{r}_k^T \underline{q}_k \quad \text{... (2)}$$

$$\underline{q}_k^T \underline{p}_k = \underline{q}_k^T S_{k+1}\underline{q}_k = \underline{q}_k^T S_k \underline{q}_k + \alpha_k \underline{q}_k^T \underline{r}_k \underline{r}_k^T \underline{q}_k \quad \text{... (3)}$$

from (2), $\alpha_k \underline{r}_k \underline{r}_k^T \underline{q}_k = \underline{p}_k - S_k \underline{q}_k$ and

from (3), $\alpha_k \underline{q}_k^T \underline{r}_k \underline{r}_k^T \underline{q}_k = \underline{q}_k^T \underline{p}_k - \underline{q}_k^T S_k \underline{q}_k$.

Here, $\alpha_k \underline{r}_k \underline{r}_k^T$ can be written as

$$\alpha_k \underline{r}_k \underline{r}_k^T = \frac{(\alpha_k \underline{r}_k \underline{r}_k^T \underline{q}_k)(\alpha_k \underline{r}_k \underline{r}_k^T \underline{q}_k)^T}{\alpha_k \underline{q}_k^T \underline{r}_k \underline{r}_k^T \underline{q}_k}$$

$$= \frac{(\underline{p}_k - S_k \underline{q}_k)(\underline{p}_k - S_k \underline{q}_k)^T}{\underline{q}_k^T(\underline{p}_k - S_k \underline{q}_k)}$$

Therefore, from (1), S_{k+1} can be written as

$$S_{k+1} = S_k + \frac{(\underline{p}_k - S_k \underline{q}_k)(\underline{p}_k - S_k \underline{q}_k)^T}{\underline{q}_k^T(\underline{p}_k - S_k \underline{q}_k)}.$$

Let

$$\underline{y}_k = \frac{\underline{p}_k - S_k \underline{q}_k}{\underline{q}_k^T(\underline{p}_k - S_k \underline{q}_k)}.$$

Then,

$$S_{k+1}\underline{q}_i = S_k \underline{q}_i + \underline{y}_k(\underline{p}_k^T \underline{q}_i - \underline{q}_k^T S_k \underline{q}_i) \quad \text{for } i < k.$$

By induction,

$$S_{k+1}\underline{q}_i = \underline{p}_i + \underline{y}_k(\underline{p}_k^T \underline{q}_i - \underline{q}_k^T \underline{p}_i).$$

Since $\underline{q}_k^T \underline{p}_i = \underline{p}_k^T R \underline{p}_i = \underline{p}_k^T \underline{q}_i$, $S_{k+1}\underline{q}_i = \underline{p}_i$.

This implies that after n steps,

$$S_n \underline{q}_i = \underline{p}_i \quad \text{for } 0 \leqq i \leqq n-1.$$

Therefore, $S_n = R^{-1}$.

. Davidon-Fletcher-Powell method

Solution of rank two correlation procedure, that is,

$$S_{k+1} = S_k + \alpha_k \underline{u}_k \underline{u}_k^T + \beta_k \underline{v}_k \underline{v}_k^T.$$

The final form of S_{k+1} is described by

$$S_{k+1} = S_k + \frac{\underline{p}_k \underline{p}_k^T}{\underline{p}_k^T \underline{q}_k} - \frac{S_k \underline{q}_k \underline{q}_k^T S_k}{\underline{q}_k^T S_k \underline{q}_k}.$$

This method is more numerically stable than
the rank one correlation method.

Quasi-Newton Algorithm

Step 1. Set $k=0$ and initialize $\underline{w}_k$ and S_k.

(S_0: any symmetric positive definite matrix)

Step 2. Set $\underline{d}_k$: $\underline{d}_k = -S_k \nabla_k$.

Step 3. Find α_k such that $\min_{\alpha_k \geqq 0} E[\underline{w}_k + \alpha_k \underline{d}_k]$.

Step 4. Update $\underline{w}_{k+1}$, $\underline{p}_k$, $\underline{q}_k$:

$\underline{w}_{k+1} = \underline{w}_k + \alpha_k \underline{d}_k$, $\underline{p}_k = \alpha_k \underline{d}_k$, and $\underline{q}_k = \nabla_{k+1} - \nabla_k$.

Step 5. Update S_{k+1}: $S_{k+1} = S_k + \dfrac{\underline{p}_k \underline{p}_k^T}{\underline{p}_k^T \underline{q}_k} - \dfrac{S_k \underline{q}_k \underline{q}_k^T S_k}{\underline{q}_k^T S_k \underline{q}_k}$

Step 6. If $E[\underline{w}_{k+1}] < \theta$, stop. Otherwise, $k \leftarrow k+1$ and go to Step 2.

. convergence:

After n linearly independent steps, weight parameter can converge to $\underline{w}^*$.

. computational complexity:

Update of S_{k+1} is required, that is,

$O(n^2)/step$

- Lebenberg-Maquardat (LM) Method

. The Levenberg-Marquardt algorithm was designed to approach second-order training speed without having to compute Hessian matrix.

. The mean square error:

$$E(\underline{w}) = \frac{1}{N}\sum_{i=1}^{N}(d_i - y_i(\underline{w}))^2$$

. The Taylor series expansion of $E(\underline{w})$ around $\underline{w}_k$:

$$E(\underline{w}) \approx E(\underline{w}_k) + \nabla_k^T(\underline{w} - \underline{w}_k) + \frac{1}{2}(\underline{w} - \underline{w}_k)^T H(\underline{w}_k)(\underline{w} - \underline{w}_k)$$

where H is the Hessian matrix defined by $H(\underline{w}_k) \equiv [\frac{\partial E}{\partial w_i \partial w_j}]|_{\underline{w} = \underline{w}_k}$.

. Here, the Hessian matrix can be approximated as

$$H(\underline{w}_k) \approx J_k^T J_k$$

where J is the Jacobian matrix defined by

$$J_k \equiv \frac{\partial E}{\partial \underline{w}}|_{\underline{w} = \underline{w}_k}.$$

. The Lebenberg-Marquardt algorithm uses this approximation to the Hessian matrix:

$$\underline{w}_{k+1} = \underline{w}_k - [J_k^T J_k + \mu I]^{-1} \nabla_k.$$

. μ is decreased after each successful step (reduction in performance function) and is increased only when
a tentative step would increase the performance function.

. convergence:
The behavior of LM algorithm is similar to the Newton method when $\mu = 0$.

. computational complexity:
The inversion of $J_k^T J_k + \mu I$ is required, that is,

$$O(n^3)/step$$

- Recursive Least Square (RLS) Method

. Let the data can be represented by

$$y_k = \underline{x}_k^T \underline{w}^* + n_k$$

where $\underline{x}_k$ represents the kth input vector, $\underline{w}^*$ represents the optimal parameter vector, and n_k represents the white Gaussian noise.

. The update rule for the parameter vector:

$$\underline{w}_{k+1} = \underline{w}_k + \underline{a}_k (y_k - \underline{x}_k^T \underline{w}_k) \quad \ldots \quad (1)$$

where $\underline{a}_k$ represents the gain vector.

. What is the optimal gain?

determine $\underline{a}_k$ in the sense of minimizing the mean-square distance from $\underline{w}_{k+1}$ to $\underline{w}^*$.

Let

$$B_k \equiv E[(\underline{w}_{k+1} - \underline{w}^*)(\underline{w}_{k+1} - \underline{w}^*)^T]. \quad \ldots \quad (2)$$

Then,

$E[\| \underline{w}_{k+1} - \underline{w}^* \|^2] = tr\left\{E[(\underline{w}_{k+1} - \underline{w}^*)(\underline{w}_{k+1} - \underline{w}^*)^T]\right\}$, that is,

$\underline{a}_k$ should be chosen to minimize $tr\{B_k\}$.

By substituting $\underline{w}_{k+1}$ of (2) for $\underline{w}_{k+1}$ of (1), we get a new matrix B_k which depends on $\underline{a}_k$.

Here, the trace of B_k is determined by

$$tr\{B_k\} = tr\{B_{k-1}\} - \frac{\underline{x}_k^T B_{k-1}^2 \underline{x}_k}{1 + \underline{x}_k^T B_{k-1} \underline{x}_k} + (1 + \underline{x}_k^T B_{k-1} \underline{x}_k) \left\| \underline{a}_k - \frac{B_{k-1} \underline{x}_k}{1 + \underline{x}_k^T B_{k-1} \underline{x}_k} \right\|^2$$

The above equation is minimized when

$$\underline{a}_k = \frac{B_{k-1} \underline{x}_k}{1 + \underline{x}_k^T B_{k-1} \underline{x}_k}.$$

In this case,

$$B_k = B_{k-1} - \frac{(B_{k-1} \underline{x}_k)(B_{k-1} \underline{x}_k)^T}{1 + \underline{x}_k^T B_{k-1} \underline{x}_k}.$$

Recursive Least Square (RLS) Algorithm

Step 1. Set $\underline{w}_0$ randomly, $B_0 = \epsilon I$ where ϵ is a small constant, and

$k = 1$.

Step 2. Update $\underline{a}_k$: $\underline{a}_k = \dfrac{B_{k-1}\underline{x}_k}{1 + \underline{x}_k^T B_{k-1}\underline{x}_k}$

Step 3. Update B_k: $B_k = B_{k-1} - \dfrac{(B_{k-1}\underline{x}_k)(B_{k-1}\underline{x}_k)^T}{1 + \underline{x}_k^T B_{k-1}\underline{x}_k}$

Step 4. Update the parameter vector: $\underline{w}_{k+1} = \underline{w}_k + \underline{a}_k(y_k - \underline{x}_k^T \underline{w}_k)$

Step 5. If $tr\{B_k\} < \theta$ (threshold value), stop.

Otherwise, $k \leftarrow k+1$ and go to step 2.

. convergence:

The behavior of RLS algorithm is similar to the Quasi-Newton method. However, RLS algorithm finds $\underline{w}^*$ asymptotically since it uses stochastic gradient $\widehat{\nabla}_k$.

. computational complexity:

The calculation of B_k is required, that is,

$O(n^2)/step$

. quick and dirty recursive linear regression:

$\underline{a}_k$ is updated by

$$\underline{a}_k = \frac{\underline{x}_k}{\sum_{j=1}^{k} \| \underline{x}_k \|^2}.$$

computational complexity: $O(n)/step$

$\underline{a}_k$ satisfies the convergence conditions of stochastic approximation.

- Conjugate Gradient (CG) Method

Let us consider the problem of solving simultaneous equations such as

$$Q\underline{x} = \underline{b}$$

This problem can be solved by minimizing the following quadratic function:

$$\min_{\underline{x}} \frac{1}{2} \underline{x}^T Q \underline{x} - \underline{b}^T \underline{x}$$

Given a symmetric matrix Q, two vectors $\underline{d}_1$ and $\underline{d}_2$ are said to be $Q-orthogonal$, or conjugate with respect to Q if $\underline{d}_1^T Q \underline{d}_2 = 0$ ($\underline{d}_1 \neq \underline{d}_2$).

Let

$$\underline{x}^* = Q^{-1}b$$

where Q is the positive definite matrix and $\underline{d}_0, \underline{d}_1, \cdots, \underline{d}_{n-1}$ be n non-zero $Q-orthogonal$ vectors. Then, $\underline{x}^*$ can be expanded as

$$\underline{x}^* = \alpha_0\underline{d}_0 + \alpha_1\underline{d}_1 + \cdots + \alpha_{n-1}\underline{d}_{n-1}$$

since

$$\underline{d}_i^T Q\underline{x}^* = \alpha_i\underline{d}_i^T Q\underline{d}_i,$$ that is,

$$\alpha_i = \frac{\underline{d}_i^T\underline{b}}{\underline{d}_i^T Q\underline{d}_i}.$$

This implies that

$$\underline{x}^* = \sum_{i=0}^{n-1}\frac{\underline{d}_i^T\underline{b}}{\underline{d}_i^T Q\underline{d}_i}\underline{d}_i.$$

Conjugate Direction Theorem

Let $\{\underline{d}_i\}_{i=0}^{n-1}$ be a set of non-zero $Q-orthogonal$ vectors. Then, for any $\underline{x}_0 \in R^n$, the sequence $\{\underline{x}_k\}$ generated according to

$\underline{x}_{k+1} = \underline{x}_k + \alpha_k \underline{d}_k, \quad k \geqq 0$ ... (1)

with

$$\alpha_k = -\frac{\underline{g}_k^T \underline{d}_k}{\underline{d}_k^T Q \underline{d}_k} \text{ and } \underline{g}_k = Q\underline{x}_k - \underline{b}$$

converges to the unique solution $\underline{x}^*$ of $Q\underline{x} = \underline{b}$ after n steps, that is, $\underline{x}_n = \underline{x}^*$.

(proof)

For some set of α_ks,

$\underline{x}^* - \underline{x}_0 = \alpha_0 \underline{d}_0 + \alpha_1 \underline{d}_2 + \cdots + \alpha_{n-1} \underline{d}_{n-1}$ and

$\underline{d}_k^T Q(\underline{x}^* - \underline{x}_0) = \alpha_k \underline{d}_k^T Q \underline{d}_k$. Then, α_k can be described by

$$\alpha_k = \frac{\underline{d}_k^T Q(\underline{x}^* - \underline{x}_0)}{\underline{d}_k^T Q \underline{d}_k}.$$

By iterating (1) from $\underline{x}_0$ upto $\underline{x}_k$, we get

$\underline{x}_k - \underline{x}_0 = \alpha_0 \underline{d}_0 + \alpha_1 \underline{d}_1 + \cdots + \alpha_{k-1} \underline{d}_{k-1}$.

From the $Q-orthogonality$,

$\underline{d}_k^T Q(\underline{x}_k - \underline{x}_0) = 0$, that is, $\underline{d}_k^T Q\underline{x}_k = \underline{d}_k^T Q\underline{x}_0$.

Therefore, α_k can be redescribed by

$$\alpha_k = \frac{\underline{d}_k^T Q(\underline{x}^* - \underline{x}_0)}{\underline{d}_k^T Q\underline{d}_k} = \frac{\underline{d}_k^T Q(\underline{x}^* - \underline{x}_k)}{\underline{d}_k^T Q\underline{d}_k} = -\frac{\underline{g}_k^T \underline{d}_k}{\underline{d}_k^T Q\underline{d}_k}$$

Reference: Luenberger, "Introduction to Linear and Nonlinear Programming," chapter 8.

<u>Conjugate Gradient Algorithm</u>

Step 1. Set any $\underline{x}_0 \in R^n$, $\underline{d}_0 = -\underline{g}_0 = \underline{b} - Q\underline{x}_0$, and $k = 0$.

Step 2. update α_k: $\alpha_k = -\dfrac{\underline{g}_k^T \underline{d}_k}{\underline{d}_k^T Q\underline{d}_k}$

Step 3. update $\underline{x}_{k+1}$: $\underline{x}_{k+1} = \underline{x}_k + \alpha_k \underline{d}_k$

Step 4. update $\underline{g}_{k+1}$: $\underline{g}_{k+1} = Q\underline{x}_{k+1} - b$

Step 5. update β_k: $\beta_k = \dfrac{\underline{g}_{k+1}^T Q\underline{d}_k}{\underline{d}_k^T Q\underline{d}_k}$

Step 6. update $\underline{d}_{k+1}$: $\underline{d}_{k+1} = -\underline{g}_{k+1} + \beta_k \underline{d}_k$

Step 7. If $\| \underline{g}_{k+1} \| < \epsilon$, stop. Otherwise, $k \leftarrow k+1$ and go to step 2.

. convergence:

The CG algorithm finds the unique solution $\underline{x}^*$ of $Q\underline{x}=\underline{b}$ after n steps, that is, $\underline{x}_n=\underline{x}^*$.

. computational complexity:

The calculation of the quadratic form of $\underline{d}_k^TQ\underline{d}_k$ is required, that is, $O(n^2)/step$

. In the learning algorithm, $\underline{x}_k$ and $\underline{d}_k$ are associated with the weight parameter and the gradient term respectively.

- Summary of Supervised Learning Algorithms

Algorithm	Gradient Descent (GD)	Conjugate Gradient (CG)	Recursive Least Square (RLS)	Lebenberg-Marquardat (LM)	Quasi-Newton	Newton
Convergence	∞	$O(n)$	$O(n)$	1	$O(n)$	1
Computational Complexity Per Step	$O(n)$	$O(n)$	$O(n^2)$	$O(n^3)$	$O(n^2)$	$O(n^3)$
Memory Requirement	$O(n)$	$O(n)$	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(n^2)$
Learning Mode	on-line	on-line	on-line	batch	batch	batch

n: the number of parameters