Chapter 4

4.1 The Basic idea about steepest-descent Algorithm

Consider a cost function $J(\vec{w})$ that is continuously differentiable function of some unknown weight vector $\vec{w}$. The function $J(\vec{w})$ maps the elements of $\vec{w}$ into real numbers.

For a mathematical statemat of unconstrained optimization, the optimal solution $\vec{w}_o$ should satisfy

$$J(\vec{w}_o) \leq J(\vec{w}) \quad \text{for all } \vec{w}.$$

We can apply the idea of local iterative descent to search for the optimal tap-weight vector $\vec{w}_o$, such that

$$J(\vec{w}(n+1)) < J(\vec{w}(n)),$$

where $\vec{w}(n)$ is the old value of the weight vector at the iteration n and $\vec{w}(n+1)$ is its update value

The method of steepest descent can be described as

$$\vec{w}(n+1) = \vec{w}(n) - \frac{1}{2}\mu \vec{g}(n),$$

where $\vec{g}(n) = \nabla J(\vec{w}(n)) = \frac{\partial J(\vec{w})}{\partial \vec{w}}$ at the iteration n

μ is so-called the step-size, $\mu > 0$.

The weight adjustment is defined as

$$\delta\vec{w}(n) \equiv \vec{w}(n+1) - \vec{w}(n) = -\frac{1}{2}\mu \vec{g}(n)$$

Therefore, by a first-order Taylor series expansion around $\vec{w}(n)$,

$$J(\vec{w}(n+1)) \approx J(\vec{w}(n)) + \vec{g}^H(n)\,\delta\vec{w}(n)$$
$$= J(\vec{w}(n)) - \frac{1}{2}\mu \|\vec{g}(n)\|^2.$$

It shows that $J(\vec{w}(n+1)) < J(\vec{w}(n))$ and hence the cost function $J(\vec{w}(n))$ decrease. as n increases, approaching to the minimum value J_{min} at $n = \infty$.

4.2 The steepest-descent (gradient-descent) for a Wiener Filter

The estimation error for an adaptive filter can be written as

$$e(n) = d(n) - \hat{d}(n|\vec{u}_n)$$
$$= d(n) - \vec{w}^H(n)\,\vec{u}(n),$$

where $\vec{w}(n) = [w_0(n), w_1(n), \ldots, w_{M-1}(n)]^T$,

and $\vec{u}(n) = [u(n), u(n-1), \ldots, u(n-M+1)]^T$

The mean-square error or cost function $J(\vec{w}(n))$ at time (iteration) n is a quadratic function of the tap-weight vector, such that

$$J(\vec{w}(n)) = \sigma_d^2 - \vec{w}^H(n)\vec{p} - \vec{p}^H\vec{w}(n)$$
$$+ \vec{w}^H(n)\,\tilde{R}\,\vec{w}(n),$$

where σ_d^2 = variance of the desired response $d(n)$

$\vec{p}$ = cross-correlation vector between $\vec{u}(n)$ and $d(n)$

$\tilde{R}$ = correlation matrix of $\vec{u}(n)$

The gradient vector is given by

$$\nabla J(\vec{w}(n)) \equiv \begin{bmatrix} \frac{\partial J}{\partial a_0(n)} + j\frac{\partial J}{\partial b_0(n)} \\ \frac{\partial J}{\partial a_1(n)} + j\frac{\partial J}{\partial b_1(n)} \\ \\ \frac{\partial J}{\partial a_{M-1}(n)} + j\frac{\partial J}{\partial b_{M-1}(n)} \end{bmatrix}$$

$$= -2\vec{p} + 2\tilde{R}\vec{w}(n).$$

Therefore, from Eq. (44), we obtain the simple recursive steepest-descent algorithm:

$$\vec{w}(n+1) = \vec{w}(n) + \mu[\vec{p} - \tilde{R}\vec{w}(n)],$$

$n = 0, 1, 2, \ldots$

4.3 Stability Analysis of the Steepest-descent Algorithm

The weight-error vector at time n can be defined as

$$\vec{c}(n) \equiv \vec{w}_o - \vec{w}(n).$$

According to Eq (2.34) and Eq. (4.10), we obtain

$$\vec{c}(n+1) = (\tilde{I} - \mu \tilde{R})\, \vec{c}(n),$$

where $\tilde{I}$ is the identity matrix.

Since $\tilde{R} = \tilde{Q} \tilde{\Lambda} \tilde{Q}^H$ (eigen-decomposition),

$$\vec{c}(n+1) = (\tilde{I} - \mu \tilde{Q} \tilde{\Lambda} \tilde{Q}^H)\, \vec{c}(n)$$

$$\tilde{Q}^H \vec{c}(n+1) = (\tilde{I} - \mu \tilde{\Lambda})\, \tilde{Q}^H \vec{c}(n).$$

We define a new set of coordinates as follows:

$$\vec{v}(n) \equiv \tilde{Q}^H \vec{c}(n) = \tilde{Q}^H [\vec{w}_o - \vec{w}(n)]$$

Therefore, $\vec{v}(n+1) = (\tilde{I} - \mu\tilde{\Lambda})\,\vec{v}(n)$.

The initial value of $\vec{v}(n)$ can be denoted as

$$\vec{v}(0) = \tilde{Q}^H [\vec{w}_o - \vec{w}(0)]$$
$$= \tilde{Q}^H \vec{w}_o \quad \text{if } \vec{w}(0) = \vec{0}$$

The k^{th} element in $\vec{v}(n+1)$, or the k^{th} natural mode of the steepest-descent algorithm, can be formulated as

$$v_k(n+1) = (1-\mu\lambda_k)\,v_k(n), \quad k = 1, 2, \cdots, M$$
$$= (1-\mu\lambda_k)\,v_k(0), \quad k = 1, 2, \cdots, M$$

The elements in the weight-error vector will die out iff $-1 < 1-\mu\lambda_k < 1$, $\forall k$

$\Rightarrow \quad 0 < \mu < \frac{2}{\lambda_{max}}$ will be the necessary and sufficient condition for the convergence or the stability of the steepest-descent algorithm.

The necessary and sufficient condition for the convergence or stability of the steepest descent algorithm is that

$$0 < \mu < \frac{2}{\lambda_{max}} ,$$

where λ_{max} is the largest eigenvalue of $\hat{R}$.

The time constant τ_k is assumed to be the duration of approximate convergence such that

$$1 - \mu\lambda_k = \exp\left(-\frac{1}{\tau_k}\right) ,$$

or
$$\tau_k = \frac{-1}{\ln(1-\mu\lambda_k)}$$

$$\approx \frac{1}{\mu\lambda_k} , \quad \mu \ll 1$$

X. Transient Behaviour of the Mean-square Error

$$J(\vec{w}(n)) = J_{min} + \sum_{k=1}^{M} \lambda_k |v_k(n)|^2$$

$$= J_{min} + \sum_{k=1}^{M} \lambda_k (1-\mu\lambda_k)^{2n} |v_k(0)|^2,$$

where $v_k(0)$ is the initial value of $v_k(n)$.

$$\lim_{n\to\infty} J(n) = J_{min} \quad \text{if} \quad 0 < \mu < \frac{2}{\lambda_{max}}$$

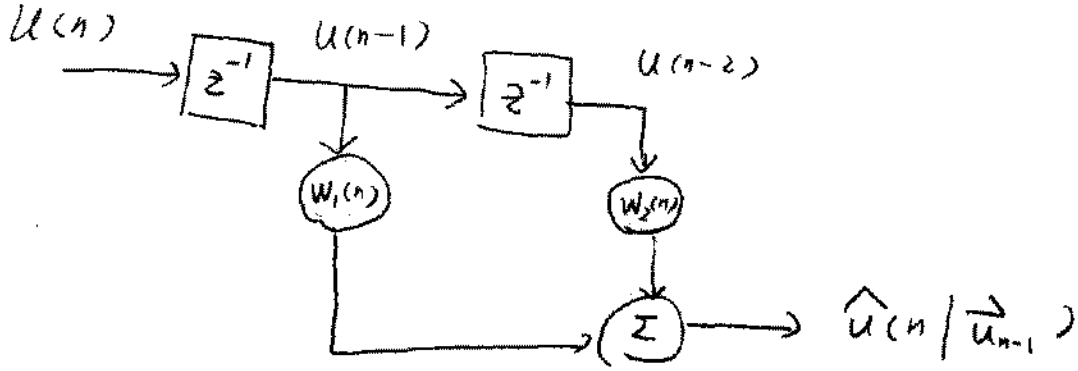
Therefore the mean-square error will converge to J_{min} approximately at

$$\tau_{k,mse} \approx \frac{-1}{2\ln(1-\mu\lambda_k)}$$

$$\approx \frac{1}{2\mu\lambda_k}$$

4.4 Example

For a forward predictor, the system is depicted as below:



The input $u(n)$ is generated from a real AR process such that

$$u(n) + a_1 u(n-1) + a_2 u(n-2) = v(n),$$

where $v(n)$ is a white process.

$$\tilde{R} = \begin{bmatrix} r(0) & r(1) \\ r(1) & r(0) \end{bmatrix},$$

$$r(0) = \sigma_u^2$$

$$r(1) = -\frac{a_1}{1+a_2}\sigma_u^2 \quad \text{since} \quad \underbrace{E[u(n)u(n-1)]}_{r(1)} + a_1 \underbrace{E[u^2(n-1)]}_{\sigma_u^2} + a_2 \underbrace{E[u(n-1)u(n-2)]}_{r(1)} = \underbrace{E[v(n)u(n-1)]}_{0}$$

From Yule-Walker Equations,

$$\begin{bmatrix} r(0) & r(1) \\ r(1) & r(0) \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} = \begin{bmatrix} r(1) \\ r(2) \end{bmatrix}$$

$$w_1 = -a_1 = \frac{r(1)[r(0) - r(2)]}{r^2(0) - r^2(1)}$$

$$w_2 = -a_2 = \frac{r(0)r(2) - r^2(1)}{r^2(0) - r^2(1)}$$

$$\therefore \quad r(2) = \left(-a_2 + \frac{a_1^2}{1+a_2}\right)\sigma_u^2$$

In addition, $\sigma_v^2 = r(0) + a_1 r(1) + a_2 r(2)$

$$\Rightarrow \sigma_v^2 = \sigma_u^2 - \frac{a_1^2}{1+a_2}\sigma_u^2 + \left(-a_2^2 + \frac{a_1^2 a_2}{1+a_2}\right)\sigma_u^2$$

$$= \frac{1+a_2 - a_1^2 - a_2^2(1+a_2) + a_1^2 a_2}{1+a_2}\sigma_u^2$$

$$= \frac{(1+a_2)(1-a_2^2) - a_1^2(1-a_2)}{1+a_2}\sigma_u^2$$

$$= \frac{(1-a_2)\left[(1+a_2)^2 - a_1^2\right]}{1+a_2}\sigma_u^2$$

$$\sigma_u^2 = \left(\frac{1+a_2}{1-a_2}\right)\frac{\sigma_v^2}{(1+a_2)^2 - a_1^2}$$

Two eigen values of $\tilde{R}$ are

$$\lambda_1 = \left(1 - \frac{a_1}{1+a_2}\right)\sigma_u^2$$

$$\lambda_2 = \left(1 + \frac{a_1}{1+a_2}\right)\sigma_u^2$$

The eigenvalue spread is

$$\chi(\tilde{R}) = \frac{\lambda_1}{\lambda_2} = \frac{1 - a_1 + a_2}{1 + a_1 + a_2}$$

The eigenvectors associated with the eigenvalues λ_1 and λ_2, are, respectively,

$$\vec{q}_1 = \begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix} \quad \text{and} \quad \vec{q}_2 = \begin{bmatrix} \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} \end{bmatrix}$$

※ Read Experiment 1 and 2 !

4.5 The steepest-descent algorithm as a deterministic search method

① Specify the arbitrary initial tap-weight vector $\vec{w}(0)$

② Estimate

$$\hat{\tilde{R}} = \frac{1}{N} \sum_{n=0}^{N-1} \vec{u}(n)\, \vec{u}^{H}(n)$$

$$\hat{\vec{p}} = \frac{1}{N} \sum_{n=0}^{N-1} \vec{u}(n)\, d^{*}(n)$$

③ Do eigen-decomposition such that

$\hat{\tilde{R}} = \tilde{Q} \tilde{\Lambda} \tilde{Q}^{H}$, where $\tilde{\Lambda} = \text{diag}(\lambda_1, \lambda_2, \cdots, \lambda_M)$

and $\lambda_{max} = \max\{\lambda_1, \lambda_2, \cdots, \lambda_M\}$

④ Choose μ such that $0 < \mu < \frac{2}{\lambda_{max}}$

⑤ Iteratively search $\vec{w}(n+1)$ according to Eq. (4.10)

4.6 Newton's Method

We can invoke the second-order Taylor series expansion of the cost function $J(\vec{w}(n+1))$ instead to search $\vec{w}_o$ iteratively.

$$J(\vec{w}(n+1)) \approx J(\vec{w}(n)) + (\vec{w}(n+1) - \vec{w}(n))^H \vec{g}(n) + \frac{1}{2} (\vec{w}(n+1) - \vec{w}(n))^H \tilde{H}(n) (\vec{w}(n+1) - \vec{w}(n)),$$

where

$$\vec{g}(n) \equiv \left. \frac{\partial J(\vec{w}(n+1))}{\partial \vec{w}(n+1)} \right|_{\vec{w}(n+1) = \vec{w}(n)}$$

and

$$\tilde{H}(n) \equiv \left. \frac{\partial^2 J(\vec{w}(n+1))}{\partial \vec{w}^2(n+1)} \right|_{\vec{w}(n+1) = \vec{w}(n)}$$

is the so-called Hessian matrix

Therefore the up-date rule can be formulated

According to Eq. (4.8) and (4.9),

$$\tilde{H}(n) = 2\tilde{R}$$

Hence

$$\vec{w}(n+1) = \vec{w}(n) - \frac{1}{2}\tilde{R}^{-1}(-2\vec{p} + 2\tilde{R}\vec{w}(n))$$

$$= \tilde{R}^{-1}\vec{p} = \vec{w}_0$$

It shows that the Newton's method attains the optimum solution $\vec{w}_0$ from an arbitrary point $\vec{w}(n)$ on the error-surface in a single iteration