Linear Units

A. Preliminaries

\[ y = s = w^T x \]
Model Assumptions and Parameters

- Training examples \((x(k), d(k))\) drawn randomly, second order zero mean sequences.
- Parameters
  - Inputs: \(x(k) \in \mathbb{R}^n\)
  - Weights: \(w(k) \in \mathbb{R}^n\)
  - Outputs: \(y(k) = w(k)^T x(k)\)
  - Desired outputs: \(d(k)\)
  - Error: \(e(k) = d(k) - y(k)\)
- Error criterion (MSE)
  \[
  \min J(w) = E \left[ .5(e(k))^2 \right]
  \]
Wiener solution

Define $P = E(x(k)d(k))$ and $R = E(x(k)x(k)^T)$.

$$J(w) = .5 E[(d(k)-y(k))^2]$$
$$= .5E(d(k)^2) - E(x(k)d(k))^T w + w^T E(x(k)x(k)^T)w$$
$$= .5E[d(k)^2] - P^T w + .5w^T R w$$

Note $J(w)$ is a quadratic function of $w$. To minimize $J(w)$ find gradient, $\nabla J(w)$ and set to 0.

$$\nabla J(w) = -P + Rw = 0$$

$Rw = P$ (Wiener solution)

If $R$ is nonsingular, then $w = R^{-1} P$.

Resulting MSE = $.5E[d(k)^2] - .5P^T R^{-1} P$
Gradient based iterative algorithms

- Steepest descent algorithm (move in direction of negative gradient)
  \[ w(k+1) = w(k) - \mu \nabla J(w(k)) = w(k) + \mu (P - Rw(k)) \]

- Least mean square algorithm (approximate gradient from training example)
  \[ \nabla J(w(k)) = -e(k)x(k) \]
  \[ w(k+1) = w(k) + \mu e(k)x(k) \]
Adaptive Filter

\[ \sum \] 

Tap Delay Line

\[ X_n \rightarrow D \rightarrow X_{n-1} \rightarrow D \rightarrow X_{n-2} \]

\[ w_0 \]

\[ w_1 \]

\[ w_2 \]

\[ \sum \] 

\[ e(n) \]

\[ y(n) \]

\[ d(n) \]
Steepest Descent Convergence

- \( w(k+1) = w(k) + \mu (P-Rw(k)) \); Let \( w^* \) be solution.
- Center weight vector \( v = w - w^* \)
- \( v(k+1) = v(k) - \mu (Rw(k)) \); Assume \( R \) is nonsingular.
- Decorrelate weight vector \( u = Q^{-1}v \) where \( R = Q\Lambda Q^{-1} \) is the transformation that diagonalizes \( R \).
- \( u(k+1) = (I - \mu \Lambda)u(k), \quad u(k) = (I - \mu \Lambda)^k u(0) \).

Conditions for convergence \( 0 < \mu < 2/\lambda_{\text{max}} \).
Step Size $\mu$

$\mu$ too large

$\mu$ too small
Rate of Convergence

- Rate of convergence depends on eigenvalues, $\lambda_i$ as convergence rate for this eigenvalue is $(1 - \mu \lambda_i)$. Key eigenvalues are $\lambda_{\text{min}}$ and $\lambda_{\text{max}}$.
- Fastest rate of convergence achieved when setting $\mu = 2 / (\lambda_{\text{min}} + \lambda_{\text{max}})$. This results in smallest and largest eigenvalue having same convergence rate.
- Convergence of SD depends on condition number of matrix $\lambda_{\text{max}} / \lambda_{\text{min}}$. 
Energy Function

- Energy Function:
  \[ J(w) = 0.5 \sigma_d^2 - P^T w + 0.5 w^T R w \]

For optimal weight \( R w^* = P \) and

\[ J_{\text{min}} = J(w^*) = 0.5 \sigma_d^2 - 0.5 P^T w^* \]

- SD energy function behavior

\[ J(w(k)) = J_{\text{min}} + 0.5 (w(k)-w^*)^T R (w(k)-w^*) \]

\[ = J_{\text{min}} + 0.5 u(k)^T \Lambda u(k) \]

\[ = J_{\text{min}} + 0.5 \sum_i (1 - \mu \lambda_i)^{2k} u_i(0)^2 \]
LMS Algorithm

- SD requires knowledge of R and P. In many applications these second order statistics are unknown.
- Least mean square algorithm
  \[ \nabla J(w(k)) = -e(k)x(k) \]
  \[ w(k+1) = w(k) + \mu e(k)x(k) \]
- LMS algorithm is an iterative noisy gradient descent algorithm that approximates SD from one training example.
- LMS algorithm attempts to find weight that minimizes mean squared error cost function, J(w).
LMS Algorithm Properties

- Steepest Descent and LMS algorithm convergence depends on step size $\mu$ and eigenvalues of $R$.
- LMS algorithm is simple to implement.
- LMS algorithm convergence is relatively slow.
- Tradeoff between convergence speed and excess MSE.
- LMS algorithm can track training data that is time varying.
LMS Convergence Behavior

- **Assumptions:** $x(n)$ iid sequence, $x(n)$ independent of $d(n-k)$, $k > 0$, $d(n)$ independent of $y(n-k)$, $k>0$, $x(n)$ and $d(n)$ are jointly Gaussian.

- **Mean convergence analysis:** Let $e^*(k) = d(k) - w^*^T x(k)$, denote error from optimal weight at time $k$.
  - $E(v(k+1)) = (I - \mu R) E(v(k)) + \mu E(x(k)e^*(k))$
  - Asymptotically assuming step size is chosen correctly, then $\lim_k E(v(k)) = 0$ and $E(w(k))$ converges to $w^*$

- **Mean squared analysis** studies cost function $J(w(k))$. Note $\text{tr}(R) > \lambda_{\text{max}}$ and more conservative bound given by $0 < \mu < 2 /\text{tr}(R)$. 
Iterative Algorithm Comments

- Algorithms based on descending energy surface by examining first and second derivatives.
- LMS (stochastic gradient descent), tradeoffs between algorithm complexity and convergence speed.
- Can use other cost functions besides quadratic cost functions: Absolute error, Minkowski error, entropy function.
- Can apply to nonlinear activation units or multi-layer networks.
- Levenberg-Marquardt algorithm: another approximation of energy function using Taylor series. Uses pseudo inverse and can approximate Newton’s method or gradient descent.
Least Squares Algorithm

- Let \((x(k), d(k)), 1 \leq k \leq m\) then LS algorithm finds weight \(w\) such that squared error is minimized. Let \(e(k) = d(k) - w^T x(k)\), then cost function for LS algorithm given by \(J(w) = .5 \sum_k e(k)^2\)
- In matrix form can represent

\[
J(w) = .5 \|d-Xw\|^2 = .5\|d\|^2 - d^TXw + .5w^TX^TXw
\]

where \(d\) is vector of desired outputs and \(X\) contains inputs arranged in rows.
Least Squares Solution

- Let $X$ be the data matrix, $d$ the desired output, and $w$ the weight vector
- Previously we showed that
  \[
  J(w) = .5 \|d - Xw\|^2 = .5\|d\|^2 - d^T Xw + .5w^T X^T Xw
  \]
  where $d$ is vector of desired outputs and $X$ contains inputs arranged in rows.
- LS solution given by $X^T X w^* = X^T d$ (normal equation) with $w^* = X^\dagger d$. If $X^T X$ is of full rank then $X^\dagger = (X^T X)^{-1} X^T$.
- Output $y = Xw^*$ and error $e = d - y$
- Desired output often of form $d = Xw^* + v$
Adaptive Filter

\[ y(n) = \sum w_0 X_u(n) + w_1 X_u(n-1) + w_2 X_u(n-2) \]

\[ e(n) = d(n) - y(n) \]