Leaky LMS Algorithm

- Convergence of tap-weight error modes dependent on conditioning of input : $\chi(R)$.
- For ill-conditioned inputs tap-weight error modes are undriven and undamped.
- Stability and convergence time issues of concern for ill-conditioned inputs.

Leaky LMS Algorithm

- Cost function:

$$J(\mathbf{w}[n]) = |e[n]|^2 + \gamma ||\mathbf{w}[n]||^2.$$

- Tap-weight update:

$$\mathbf{w}[n+1] = (1 - \mu\gamma)\mathbf{w}[n] + \mu e[n]\mathbf{u}^*[n].$$

- Tap-weight converges in mean to a biased solution:

$$\lim_{n\to\infty} E\{\mathbf{w}[n]\} = (\mathbf{R}_{uu} + \gamma \mathbf{I})^{-1}\mathbf{r}_{du}$$

- Tap-weight error modes stabilized if :

$$0 < \mu < \frac{2}{\lambda_{\text{max}} + \gamma}$$

Sign--based Algorithms

- Sign-error algorithm:

$$\mathbf{w}[n+1] = \mathbf{w}[n] + \mu \ \text{sign}(e[n])\mathbf{u}[n]$$

- Sign-data algorithm:

$$\mathbf{w}[n+1] = \mathbf{w}[n] + \mu \ \text{sign}(\mathbf{u}[n])e[n]$$

- Sign-sign algorithm:

$$\mathbf{w}[n+1] = \mathbf{w}[n] + \mu \ \text{sign}(e[n])\text{sign}(\mathbf{u}[n])$$

Sign-based Algorithms

- Incorporate 2-bit quantization of estimation error e[n] and/or tap-input u[n].
- Tap-weight update not guided by gradient.
- Tap-weights do not converge to optimal Wiener solution.
- May need regularization to stabilize the tap-weight update.

Affine Projection Algorithm

- Tap-weight adjustment of NLMS: w[n+1] – w[n] in the direction of u[n].
- Movement of tap-weight adjustment a function of correlation between u[n] and u[n-1].
- Larger correlation between u[n] and u[n-1] produces a smaller tap-weight movement.
- Need to pre-whiten observations to attain a faster uniform rate of convergence.

Affine Projection Algorithm

- Modified Cost function:

$$J(n) = ||\mathbf{w}[n+1]-\mathbf{w}[n]||^2 + \sum_{k=0}^{N-1} \lambda_k (d[n-k]-\mathbf{w}^T[n+1]\mathbf{u}[n-k]).$$

- Some definitions:

$$\begin{aligned} \mathbf{A}^T[n] &= [\mathbf{u}[n], \mathbf{u}[n-1], \ldots, \mathbf{u}[n-N+1]], \\ \Lambda^T[n] &= [\lambda_o, \lambda_1, \ldots, \lambda_{N-1}], \\ \mathbf{d}^T[n] &= [d[n], d[n-1], \ldots, d[n-N+1]]. \end{aligned}$$

- Matrix form:

$$J(n) = ||\mathbf{w}[n+1]-\mathbf{w}[n]||^2 + (\mathbf{d}[n]-\mathbf{A}[n]\mathbf{w}[n+1])^T \Lambda.$$

Affine Projection Algorithm

- Derivative w.r.s.t. Λ:

$$\mathbf{d}[n] = \mathbf{A}[n]\mathbf{w}[n+1].$$

- Derivative w.r.s.t. w[n+1]:

$$\mathbf{w}[n+1] = \mathbf{w}[n] + \frac{1}{2}\mathbf{A}^T[n]\Lambda$$

- Tap-weight update:

$$\begin{aligned} \mathbf{w}[n+1] &= \mathbf{w}[n] + \tilde{\mu}\mathbf{A}_R^{\dagger}[n](\mathbf{d}[n] - \mathbf{A}[n]\mathbf{w}[n]) \\ \tilde{\mu} &= \frac{\beta}{||\mathbf{u}[n]||^2 + \delta} \end{aligned}$$

Affine Projection Algorithm

- Alternative form of tap-weight update:

$$\mathbf{w}[n+1] = \left(\mathbf{I} - \tilde{\mu}\mathbf{A}_R^{\dagger}[n]\mathbf{A}[n]\right)\mathbf{w}[n] + \tilde{\mu}\mathbf{A}_R^{\dagger}[n]\mathbf{d}[n].$$

- Regularization for pseudo-inverse:

$$\tilde{\mathbf{A}}_R^{\dagger}[n] = \mathbf{A}^T[n](\mathbf{A}[n]\mathbf{A}^T[n] + \gamma\mathbf{I})^{-1}$$

- Better convergence characteristics comes at the price of computational complexity.

Self Orthogonalizing Filters

- Tap-weight update:

$$\mathbf{w}[n+1] = \mathbf{w}[n] + \alpha \mathbf{R}_{uu}^{-1}\mathbf{u}[n]e[n].$$

- Average tap-weight error:

$$E\{\epsilon[n+1]\} = (1-\alpha)E\{\epsilon[n]\}$$

- Solution to difference equation:

$$E\{\epsilon[n]\} = (1-\alpha)^n E\{\epsilon[0]\}$$

- Convergence in mean only dependent on α

Self Orthogonalizing Filters

- Improved convergence comes at the price of computational complexity.
- Two stage principle component implementation:

$$\begin{aligned} \tilde{u}_i[n] &= \mathbf{q}_i^H \mathbf{u}[n] \\ \tilde{\mathbf{u}}^T[n] &= [u_0[n], u_1[n], \ldots, u_{L-1}[n]] \\ \mathbf{w}[n+1] &= \mathbf{w}[n] + \alpha \mathbf{\Lambda}^{-1}\tilde{\mathbf{u}}[n]e[n]. \end{aligned}$$

- Independent principle component filters:

$$w_i[n+1] = w_i[n] + \frac{\alpha}{\lambda_i}\tilde{u}_i[n]e[n].$$

Variable Step-size Algorithm (VSA)

- Constant step-size treats every tap-weight iteration identical.
- Desire the tap-weight update to proceed at faster rate in initial stages.
- Desire the tap-weight update to slow down in the final stages to avoid large misadjustment.
- Requires optimization of step-size $\mu[n]$

Variable Step-size Algorithm

- Cost-function:

$$J(n) = |e[n]|^2 = |d[n] - \mathbf{w}^T[n]\mathbf{u}[n]|^2$$

- Step-size update:

$$\mu[n+1] = \mu[n] + \rho\nabla_{\mu[n]}(J(n))$$

- Take gradient w.r.s.t $\mu[n]$:

$$\nabla_{\mu[n]}(J(n)) = -(\gamma^T[n]\mathbf{u}[n]e^*[n] + \gamma^H[n]\mathbf{u}^*[n]e[n])$$

Variable Step-size Algorithm

- Time-varying tap-weight update:

$$\mathbf{w}[n+1] = \mathbf{w}[n] + \mu[n]e[n]\mathbf{u}^*[n]$$

- Step-size update:

$$\mu[n+1] = \mu[n] + \rho(\gamma^T[n]\mathbf{u}[n]e^*[n] + \gamma^H[n]\mathbf{u}^*[n]e[n])$$

- Sensitivity update:

$$\gamma[n+1] = (\mathbf{I} - \mu[n]\mathbf{u}^*[n]\mathbf{u}^T[n])\gamma[n] + e[n]\mathbf{u}^*[n]$$

- Improved convergence with lower computational cost.

Block LMS Algorithm

- Uses type-I polyphase components of the input u[n]:

$$u_i[k] = u[kL_b + i], i = 0, 1, \ldots, L_b - 1.$$

- Block input matrix:

$$\mathbf{A}^T(k) = [\mathbf{u}[kL_b], \mathbf{u}[kL_b+1], \ldots, \mathbf{u}[kL_b+L_b-1]].$$

- Block filter output:

$$y_i[k] = \mathbf{w}^T[k]\mathbf{u}[kL_b + i] = \sum_{p=0}^{L-1} w_p[k]u_i[k-p].$$

Block LMS Algorithm

- Block estimation error:

$$e_i[k] = d_i[k] - y_i[k], \;\; i = 0, 1, \ldots, L_b - 1$$

- Tap-weight update:

$$\begin{aligned}\mathbf{w}[k+1] &= \mathbf{w}[k] + \mu \sum_{i=0}^{L_b-1} \mathbf{u}[kL_b + i]e_i[k] \\ &= \mathbf{w}[k] + \mu\mathbf{A}^T[k]\mathbf{e}[k].\end{aligned}$$

- Gradient estimate:

$$\hat{\mathbf{g}}[n] = -\frac{2}{L_b}\sum_{i=0}^{L_b-1} \mathbf{u}[kL_b + i]e_i[k].$$

Block LMS Algorithm

- More accurate gradient estimate employed.
- Identical to the standard LMS in convergence time and misadjustment.
- Reduced complexity when implementing convolution and correlation with overlap save method.
- More appropriate for block stationary inputs.