ECE 598NSG/498NSU
Deep Learning in Hardware
Fall 2020

Training DNNs – The LMS Algorithm

Naresh Shanbhag

Department of Electrical and Computer Engineering

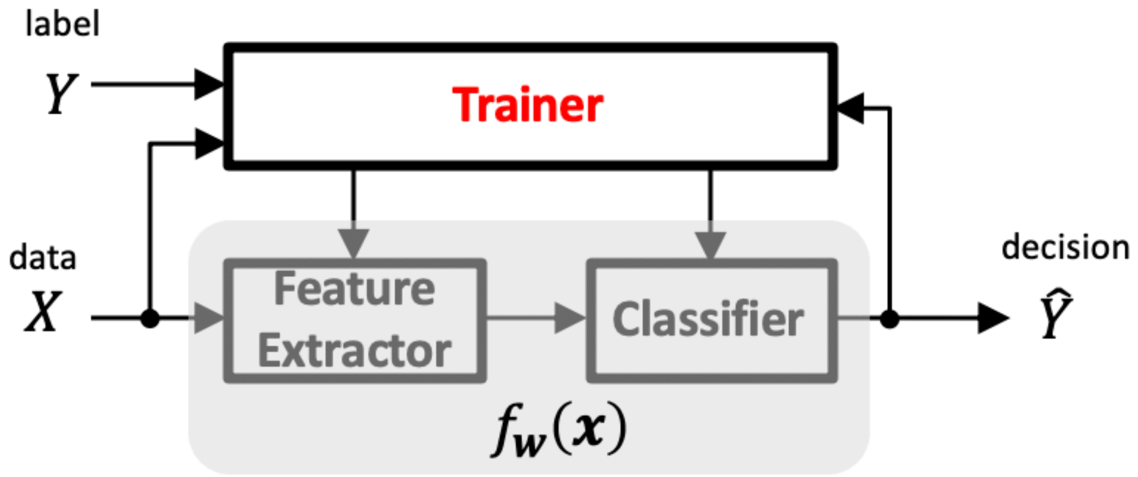
University of Illinois at Urbana-Champaign

http://shanbhag.ece.uiuc.edu

Today

- Overview of DNN training – the big picture
- The Stochastic Gradient Descent (SGD) training algorithm
- A simple learner: the least mean-squared (LMS) algorithm

Supervised Training



- Sample: $z = (\boldsymbol{x}, y) = (\text{data}, \text{label})$;
- Training sample (example): samples used in training.
- Loss function: $L(\hat{y} = f_{\boldsymbol{w}}(\boldsymbol{x}), y)$; sample-specific value
- Family of functions parametrized by $\boldsymbol{w}$: $\mathcal{F}$: $f_{\boldsymbol{w}}(\boldsymbol{x}) = \hat{y}$
- Loss function evaluated on the training set: $Q(\boldsymbol{w})$
- learning (convergence) curves; learning rate; stability

The LMS Algorithm

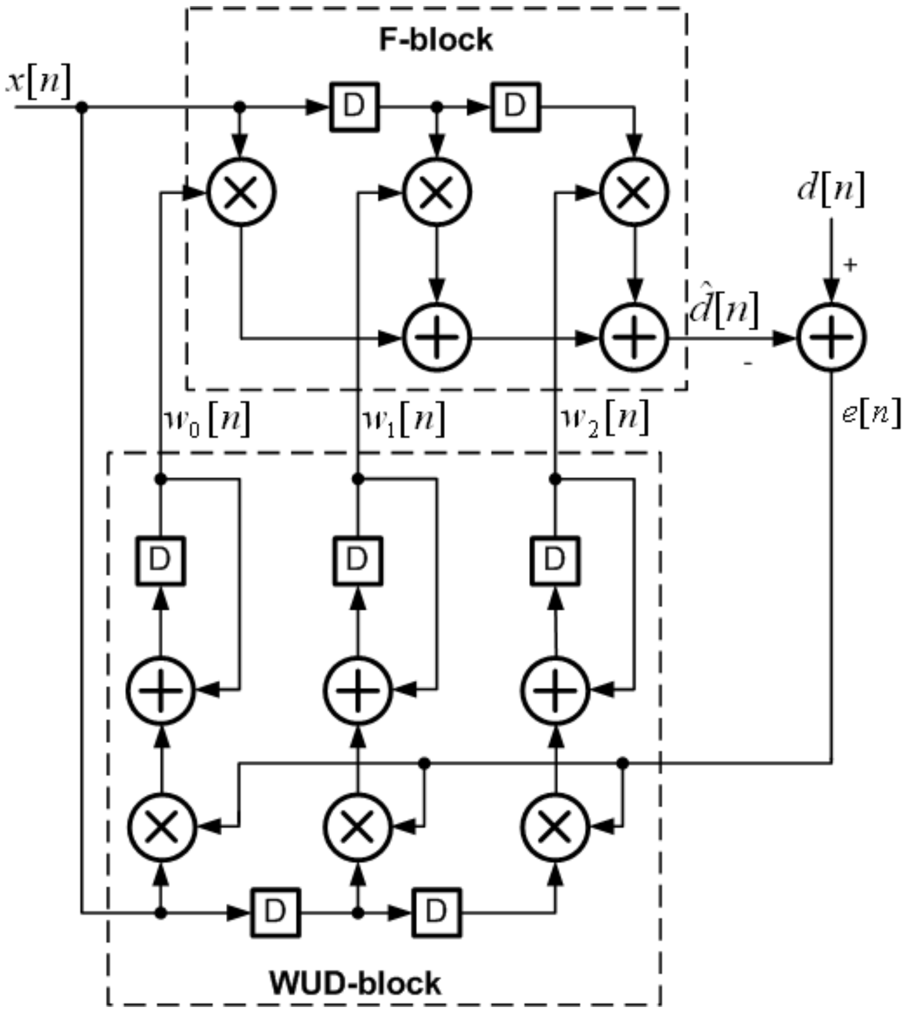
Least-Mean Square (LMS) Algorithm

$$e[n] = d[n] - \boldsymbol{W}^T[n]\boldsymbol{X}[n]$$ **(dot product)**

$$\boldsymbol{W}[n+1] = \boldsymbol{W}[n] + \mu e[n]\boldsymbol{X}[n]$$ **(weight update)**

- two steps per iteration (n: iteration index)
- dot product step**:** takes dot-product and calculates error $e[n]$
- weight update step**:** weight vector $\boldsymbol{w}[n]$ updated using $e[n]$
- minimizes mean squared error (MSE): $E\left[e^2[n]\right] = J(\boldsymbol{w})$

Example: 3-Tap LMS Adaptive Filter



D: one sample delay (register)

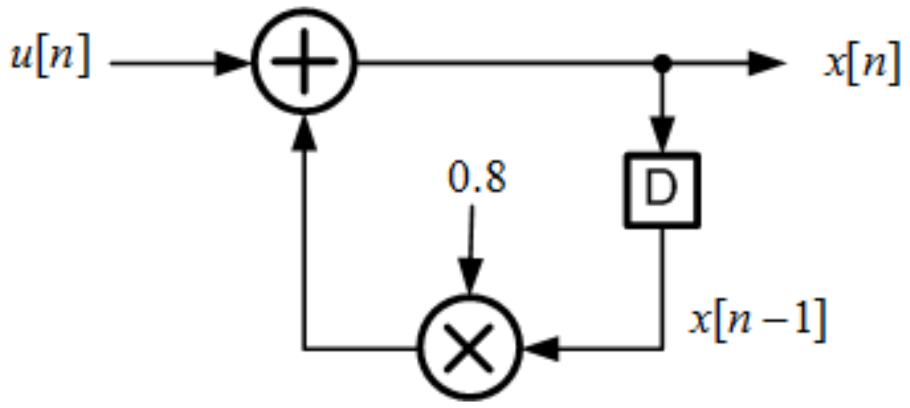
- complexity: ~ 6 MACs, 6 registers, and 1 adder
- energy ∝ complexity
- critical path delay:

$$T_{cp} = 2T_m + 4T_A$$

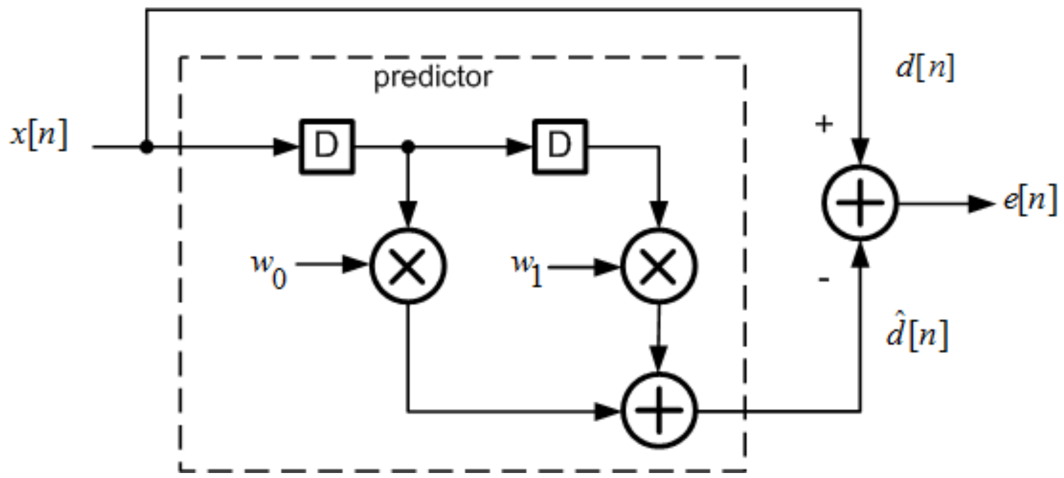
- throughput ∝ $1/T_{cp}$
- T_{cp} can be reduced via retiming, pipelining and parallelization
- latency = $2T_m + 4T_A$= T_{cp} of architecture without pipelining/parallelization

Example - Predictor

AR model

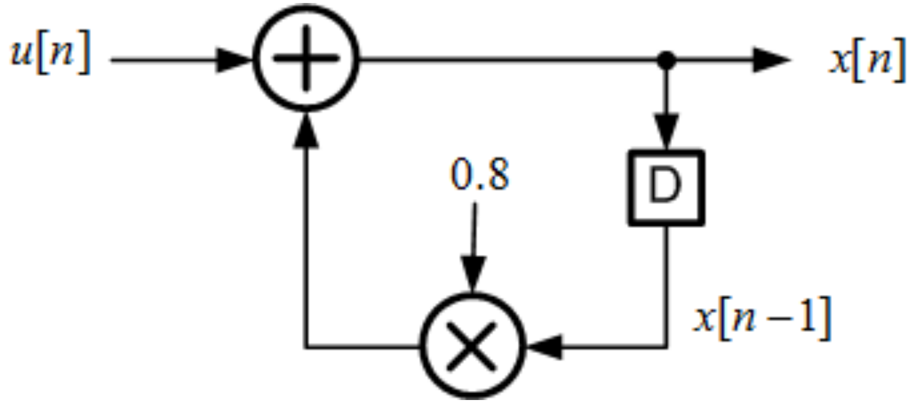


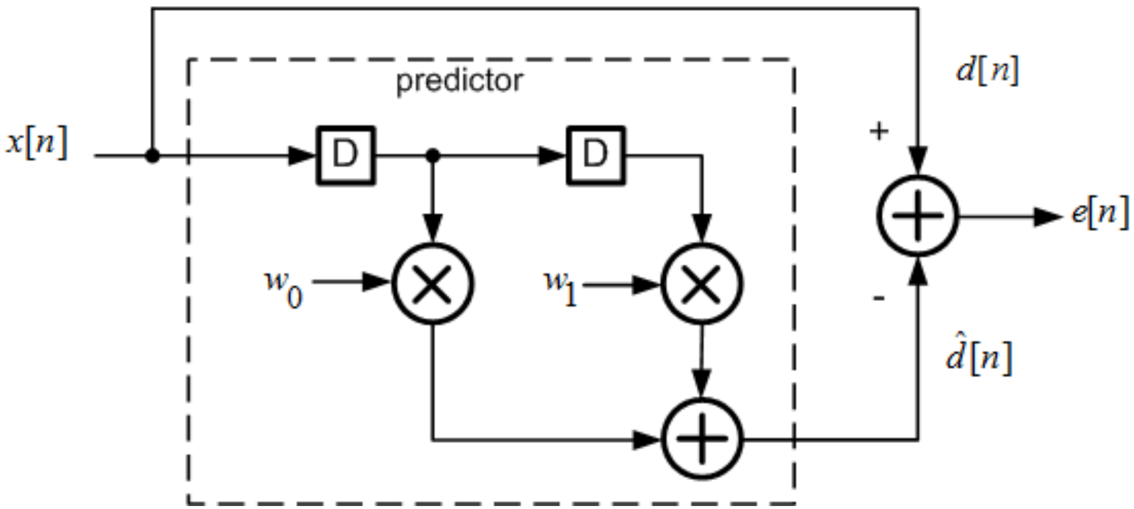
fixed predictor



- signal generation model is unknown to the predictor
- predictor 'sees' $x[n] = d[n]$ and computes a prediction $\hat{d}[n]$
- find coefficients w_0 and w_1 which will minimize the mean squared error $E[e^2[n]]$ just by observing data $x[n]$
- knowledge of the parameters of the AR model is not needed

Example





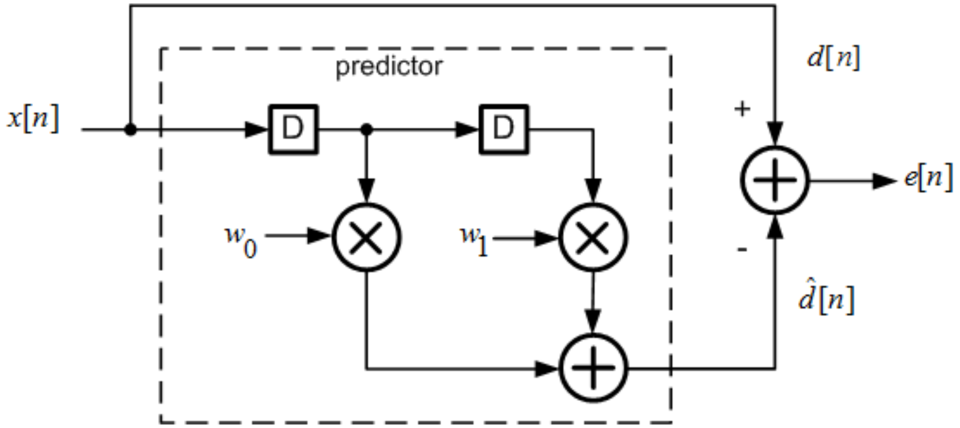
$$\mathbf{R} = \begin{bmatrix} 2.8 & 2.24 \\ 2.24 & 2.8 \end{bmatrix} \quad \mathbf{p} = \begin{bmatrix} 2.24 \\ 1.8 \end{bmatrix} \quad \boldsymbol{W}_{opt} = \begin{bmatrix} 0.8 \\ 0.008 \end{bmatrix}$$

$$J_{min} = \sigma_d^2 - \boldsymbol{p}^T \boldsymbol{W}_{opt}$$

- $J_{min} = \sigma_x^2 - (1.8 - 0.01) = 2.8 - 1.81 = 0.99$

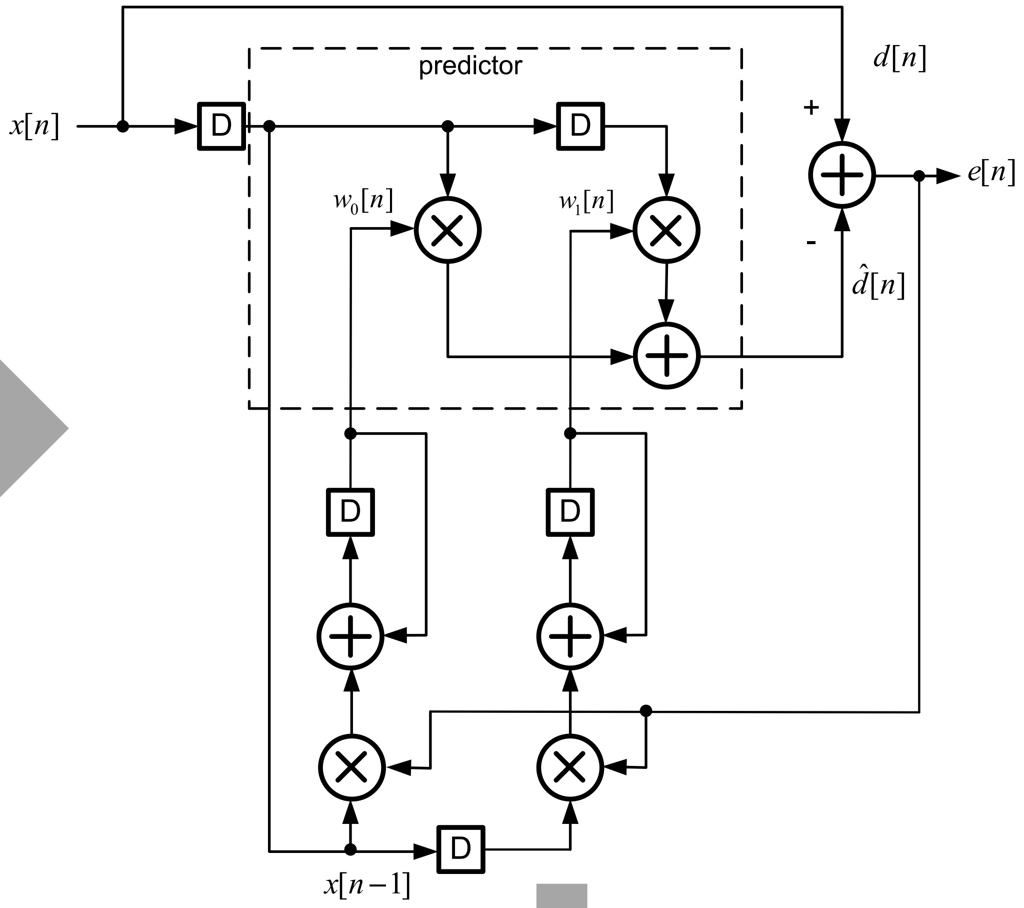
LMS Predictor

fixed predictor



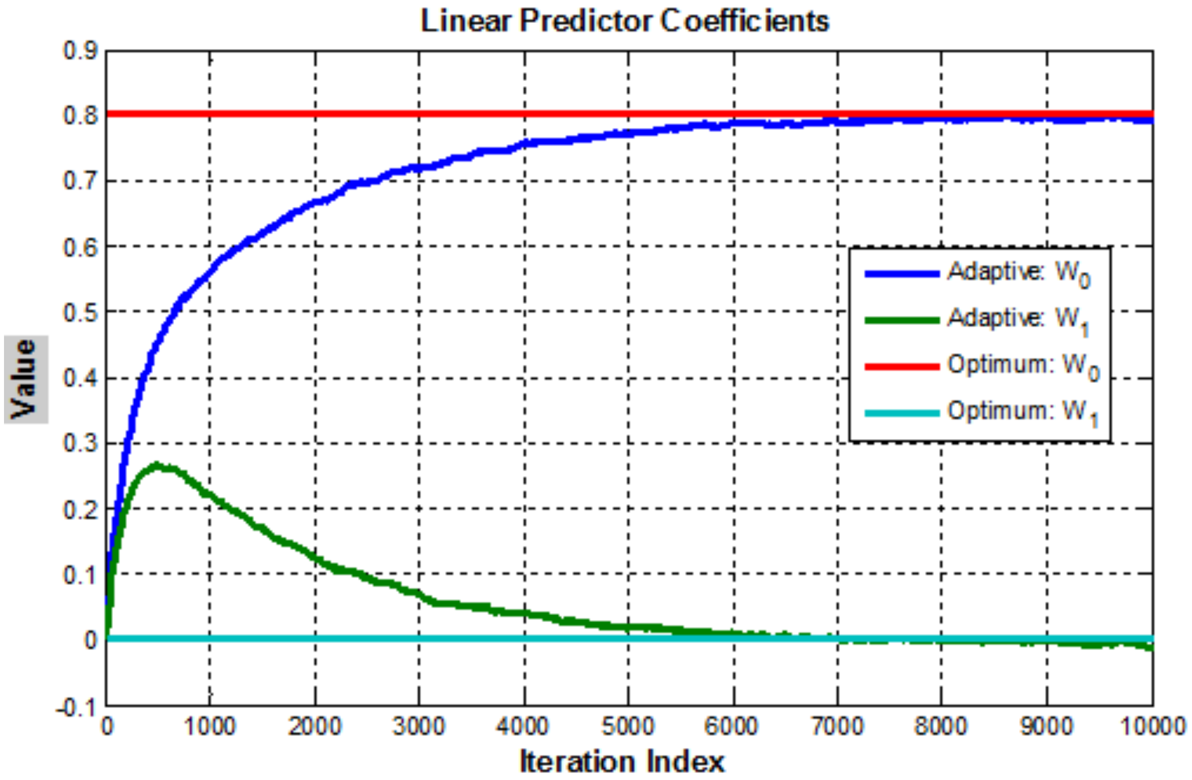
can be used only when data statistics are known in advance

LMS adaptive (learning) predictor



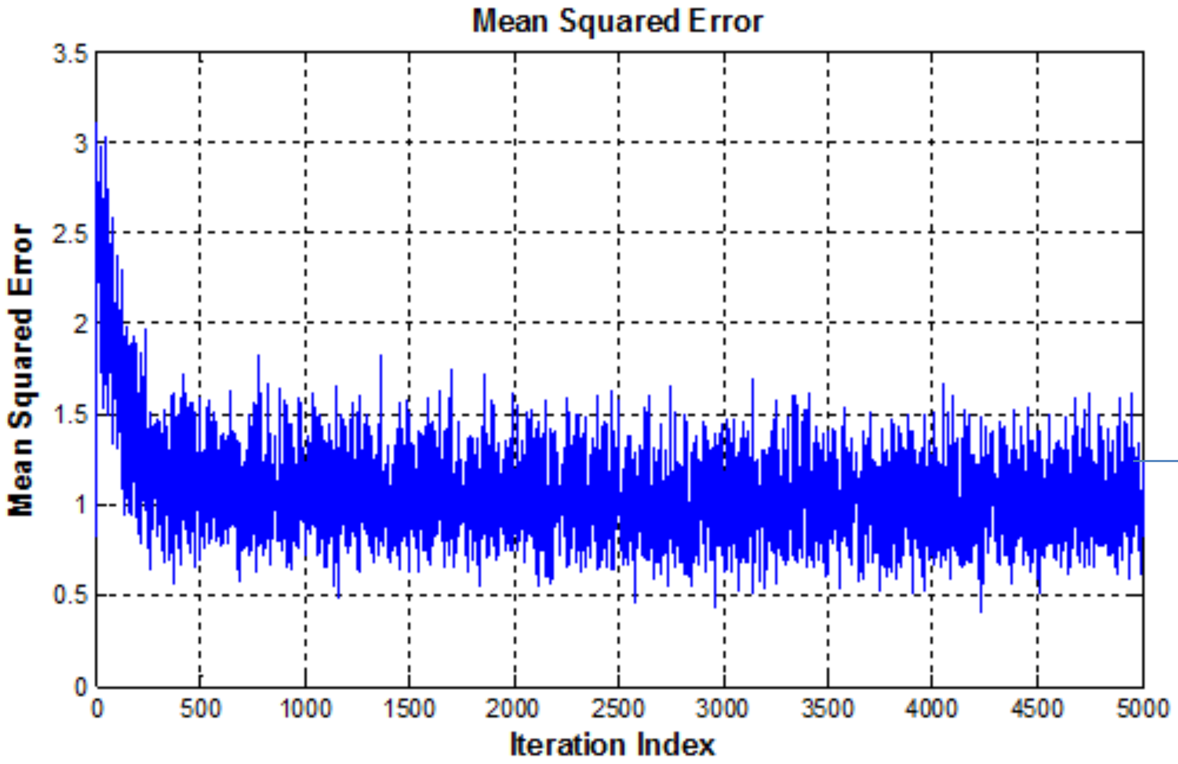
- would like to see $w_0[n] \rightarrow 0.8$ and $w_1[n] \rightarrow 0$ and $E\left[e^2[n]\right] \rightarrow 0.99$ as $n \rightarrow \infty$ and $\mu \rightarrow 0$

Convergence Curves – Predictor Weights



- convergence curves plot the evolution of the predictor coefficients
- $E\{w_0[n]\} \rightarrow 0.8$ and $E\{w_1[n]\} \rightarrow 0.008$ as $n \rightarrow \infty$
- $w_0[n] \rightarrow 0.8$ and $w_1[n] \rightarrow 0.008$ as $n \rightarrow \infty$ and $\mu \ \rightarrow 0$

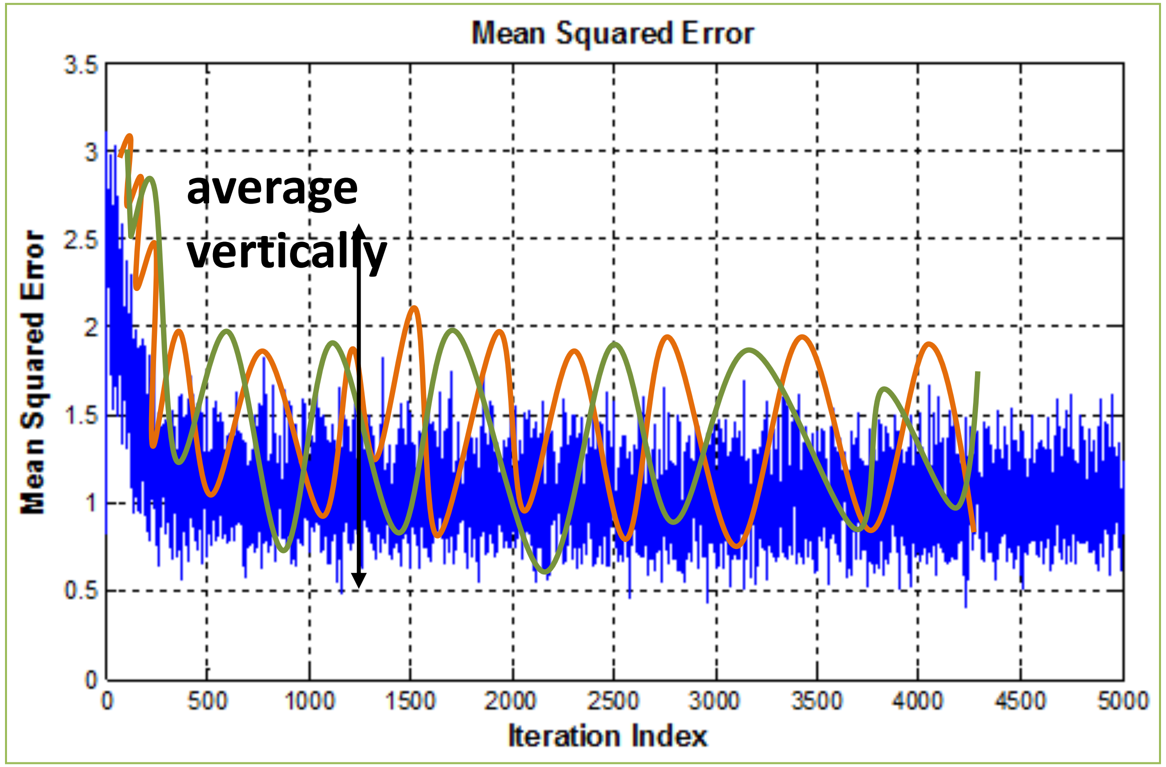
Convergence Curves – MSE



MMSE=1

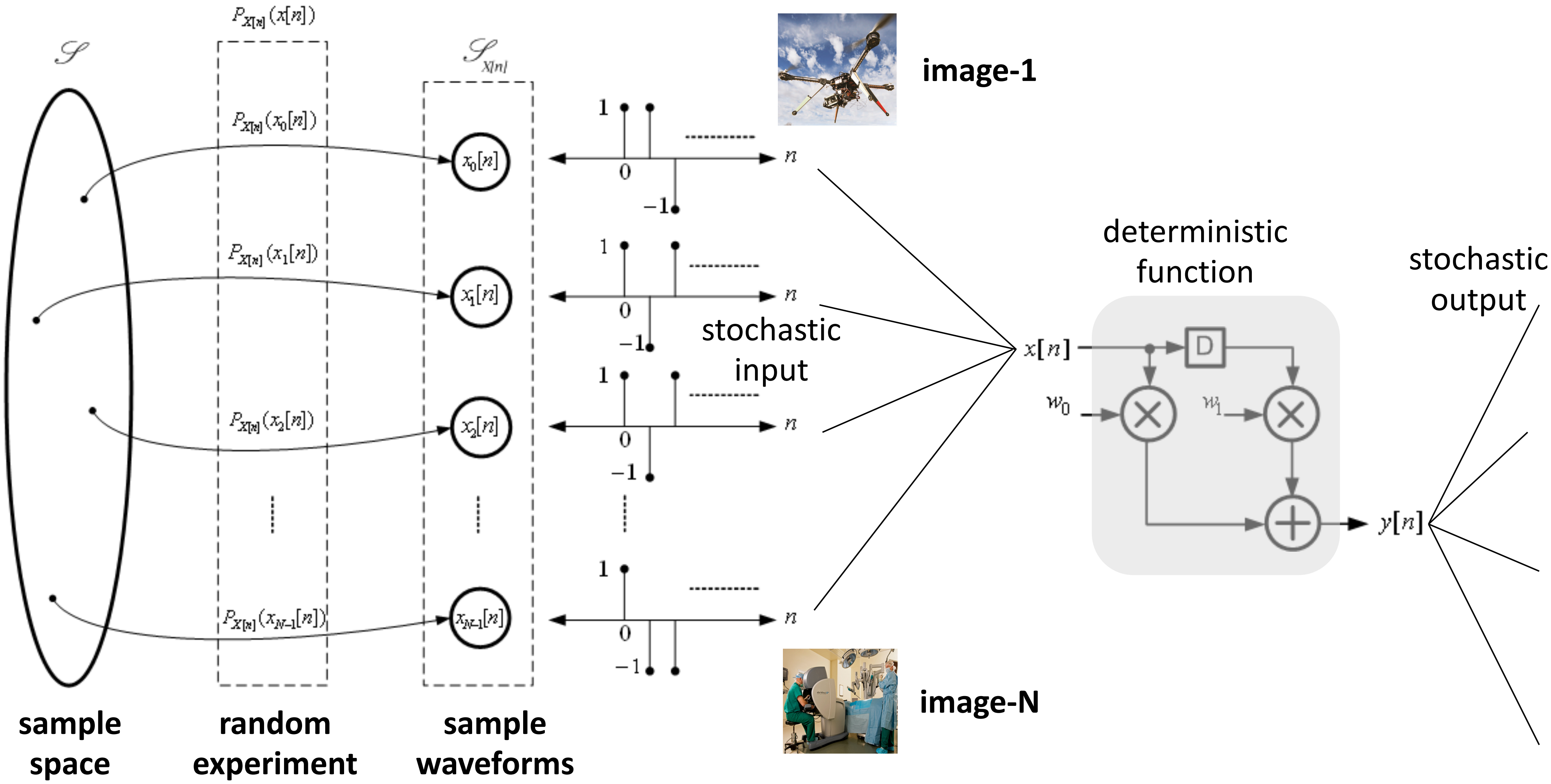
- convergence curves plot the evolution of MSE $E[e^2[n]]$
- MSE reduces over n → LMS filter is converging
- minimum MSE = 1 as expected …… why? ($e[n] = u[n]$ after convergence)

Ensemble Averaging



- these convergence curves are obtained via ensemble averaging
- input is treated as a random process – infinite sequence of RVs
- simulate independent runs and average (vertically) across each run to obtain $E[e^2[n]]$, $E[w_0[n]]$, and $E[w_1[n]]$

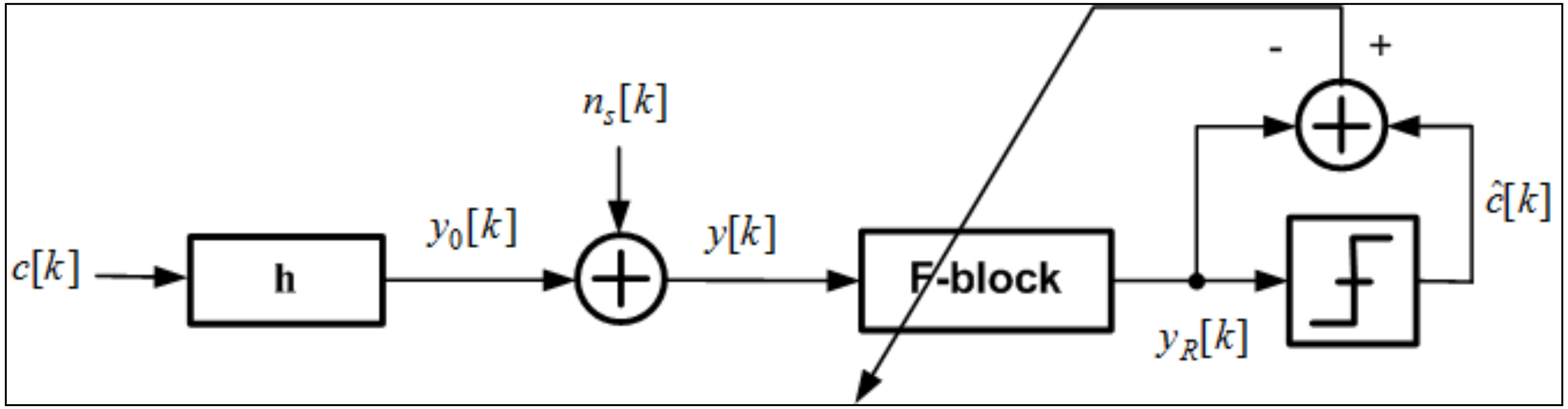
RPs and Ensemble Averaging



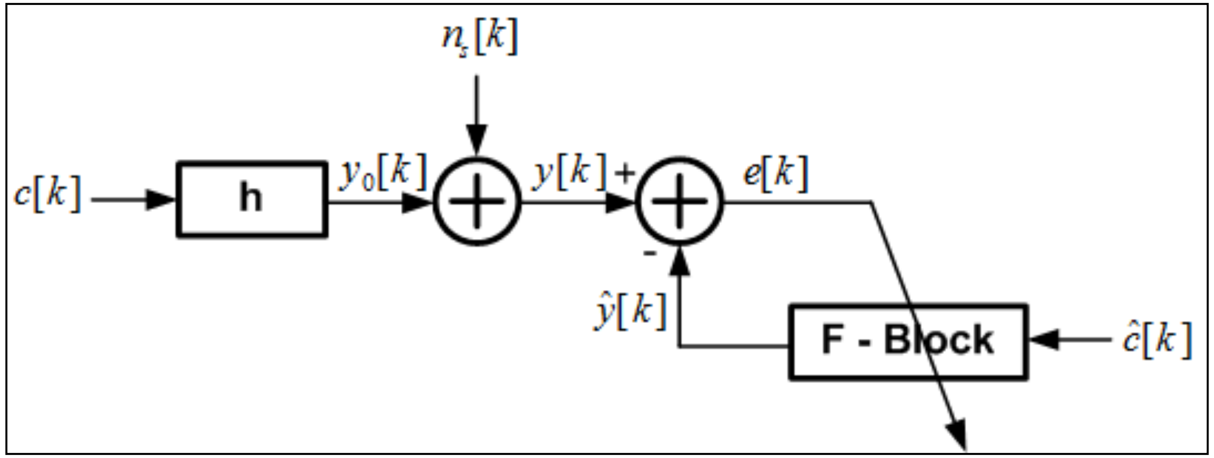
```
%clear data
clear; clc

%loop for sample runs
for run=1:50

 run;
 %Generate correlated data
 x=randn(1,100000);y=filter(1,[1 -0.8],x);

 %length of predictor
 M=2;

 %Calculate Wiener-Hopf Coefficients
 %Find correlation matrices
 %R: Toeplitz matrix with elements r(0),...r(M-1)
 %r: = [r(1); ... ;r(M)]
 [Y
corMat]=corrmtx(y,M);R=corMat(1:M,1:M);r=cor
Mat(2:M+1,1);

 %Wiener-Hopf Equation
 Wopt=inv(R)*r;
 y_est = filter([0 Wopt(2:end)],1,y);

 %LMS Adaptive predictor
 %initialize weight vector and step-size
 iter =10000;W=zeros(M,iter); u=0.001;
y=[zeros(1,M) y];

 %update taps
 for i= M+1 : iter
  y_est(i)=y(i-M:i-1)*W(:,i);
  e(i)=y(i)-y_est(i);
  W(:,i+1) = W(:,i) + u*e(i)*y(i-M:i-1)';
 end
 W_av0(run,:) = W(2,:);W_av1(run,:) = W(1,:);
 MSE(run,:) = e(M+1 : iter).^2;

end %run loop

% Take ensemble average before plotting
W_av0=mean(W_av0);W_av1=mean(W_av1);MSE=
mean(MSE);
% Plotting commands follow
```

Classroom Discussion

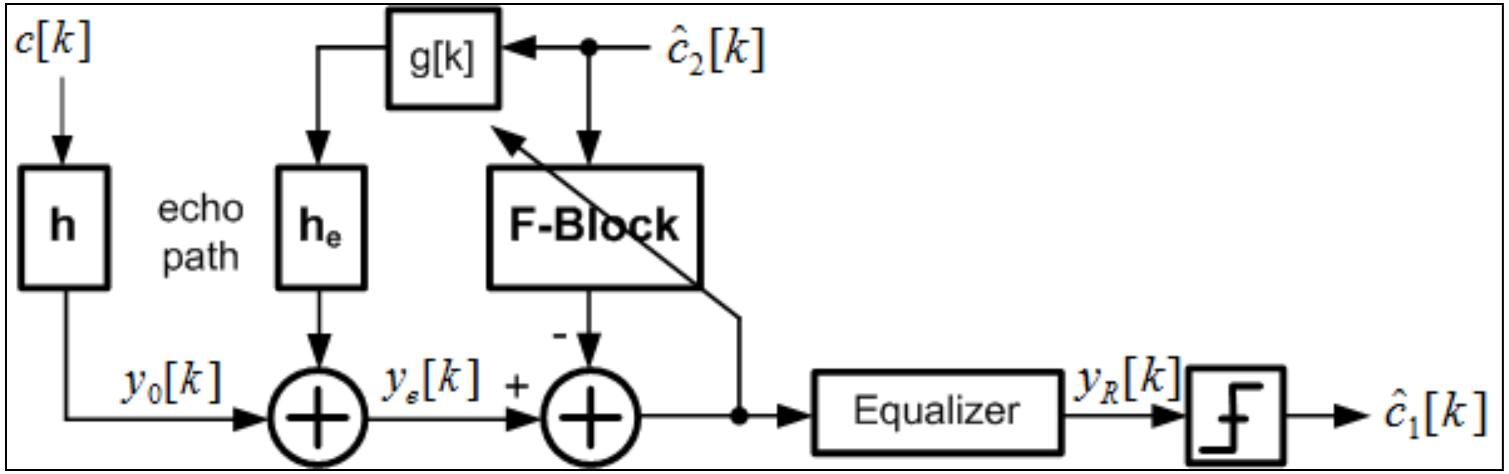
- for each of the following, determine the **input** signal, **desired** signal, **predicted** signal, and the **prediction error** the LMS learner.



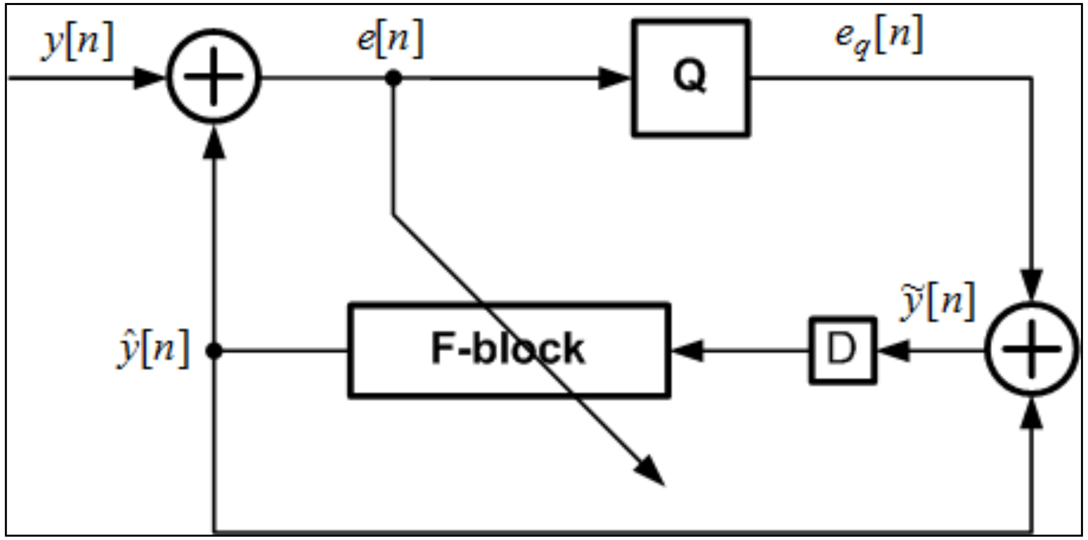
channel equalizer



channel estimator



near-end cross-talk canceller



adaptive differential
pulse-code modulation (ADPCM)
coder

Convergence Properties of the LMS Algorithm

- LMS is an iterative algorithm and hence its convergence properties are important to study

- Key convergence properties are:
  - stability (does it oscillate or diverge?)
  - rate of convergence (how fast does it settle?)
  - accuracy (how close is it to the MMSE solution?)

Stability Bounds

$$e[n] = d[n] - \boldsymbol{W}^T[n]\boldsymbol{X}[n]$$

$$\boldsymbol{W}[n+1] = \boldsymbol{W}[n] + \mu e[n]\boldsymbol{X}[n]$$

- the step-size μ should be small enough for LMS to converge
- too small $\mu \rightarrow$ slow convergence
- too large $\mu \rightarrow$ instability

- stability bounds on μ

$$0 < \mu < \frac{2}{N\sigma_x^2}$$

Example – Stability Bounds for LMS Predictor

$$0 < \mu < \frac{2}{N\sigma_x^2}$$

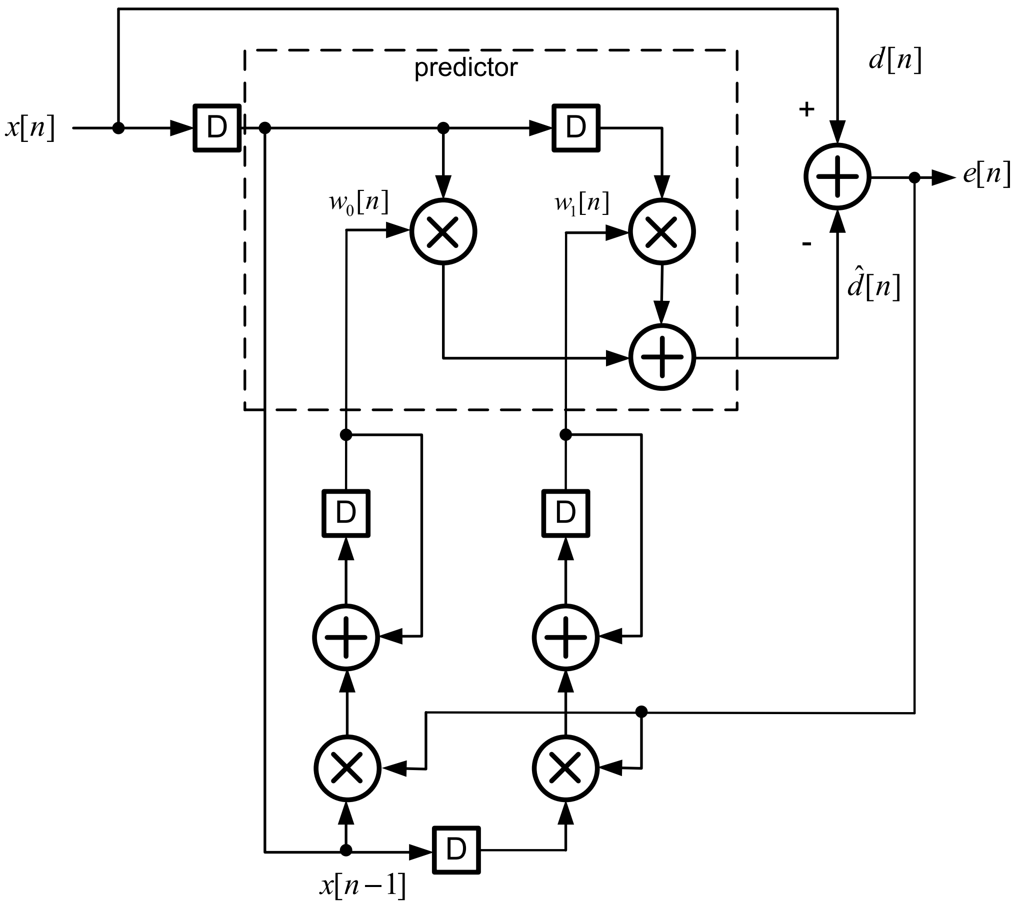
- $N = 2; \sigma_x^2 = 2.8 \Rightarrow$

$$\mu_{max} = \frac{2}{5.6} = 0.36$$

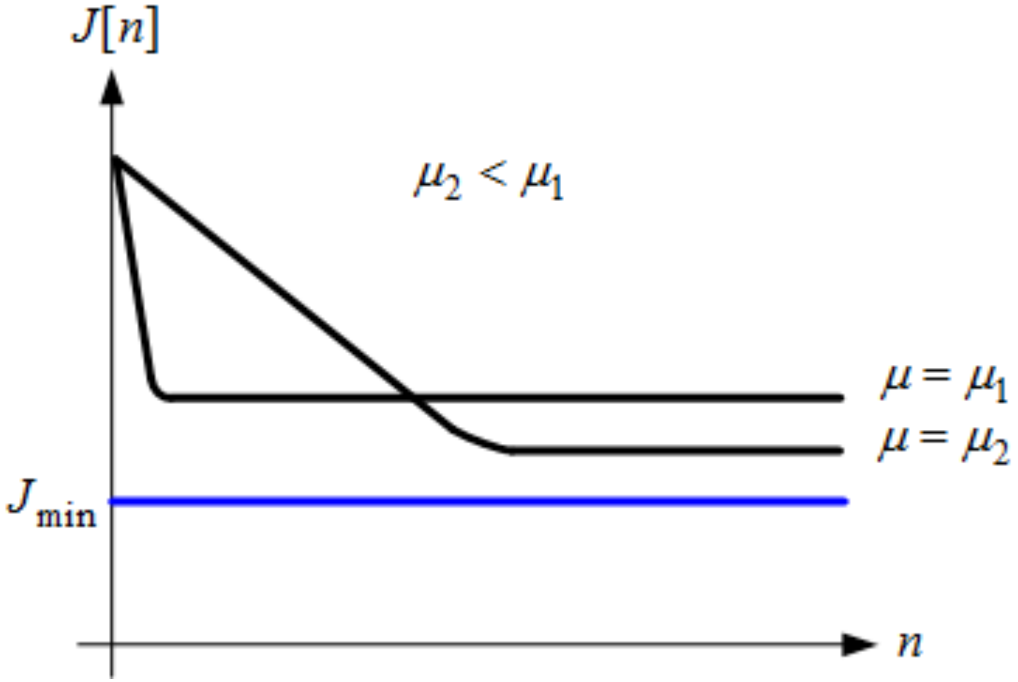
%LMS Adaptive predictor

%initialize weight vector and step-size

iter =10000;W=zeros(M,iter); u=0.001; y=[zeros(1,M) y];



Rate of Convergence and Accuracy



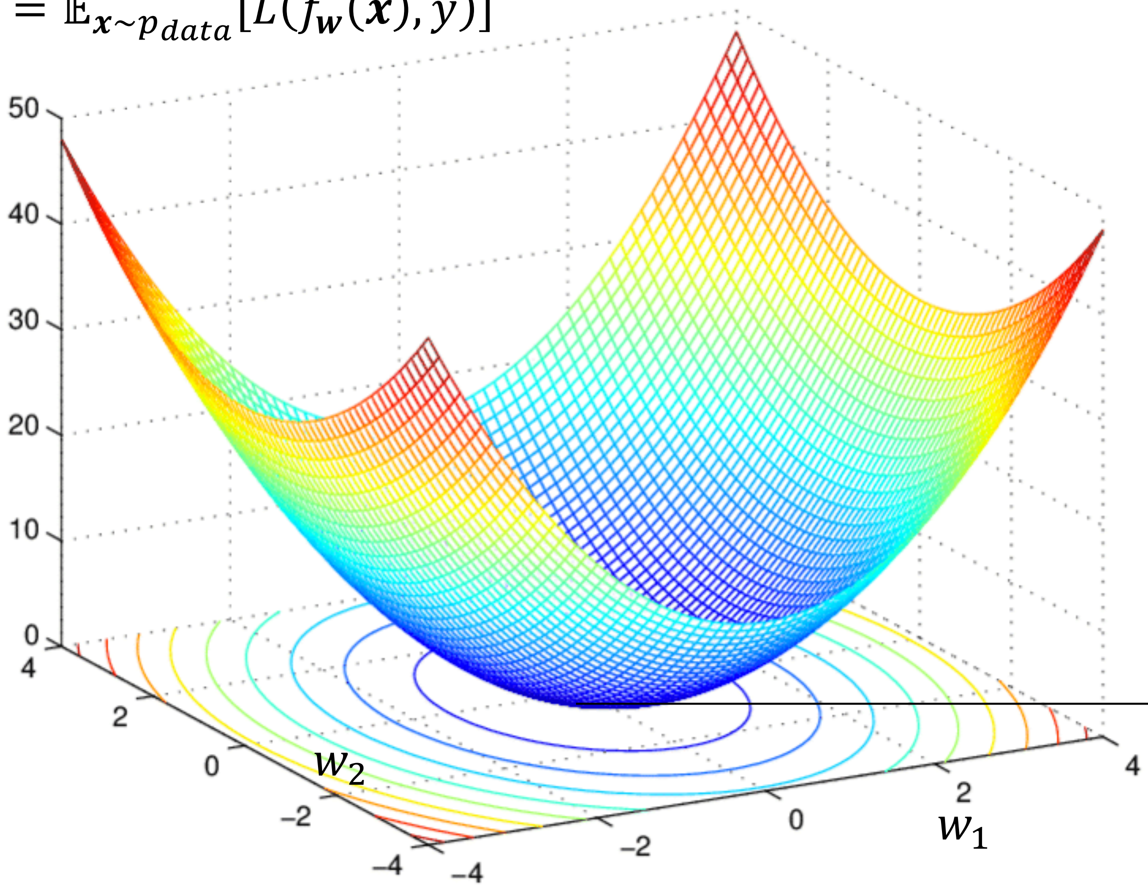
- convergence rate: number of iterations needed to settle
- (in)accuracy (misadjustment)

$$\eta = \frac{J(\infty) - J_{min}}{J_{min}} = \mu\, \mathrm{tr}(R)$$

- step-size μ tradeoffs convergence rate with accuracy
- larger $\mu \rightarrow$ faster convergence but lower accuracy (higher η)

LMS's Loss Landscape

$J(\boldsymbol{w}) = \mathbb{E}_{\boldsymbol{x} \sim p_{data}}[L(f_{\boldsymbol{w}}(\boldsymbol{x}), y)]$



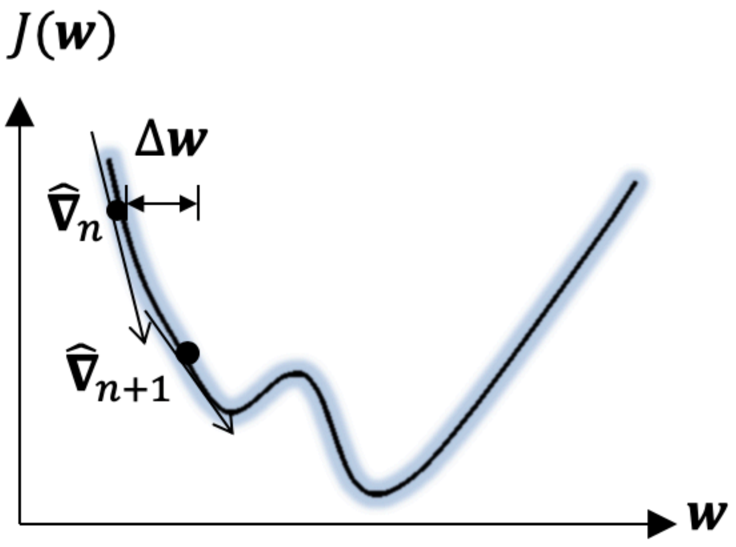
loss function is convex

unique minimum

$\boldsymbol{w}_{opt} = \boldsymbol{R}^{-1}\boldsymbol{p}$

$\boldsymbol{w} = [w_1, w_2]^T$

LMS Variants



Stochastic Gradient Descent (SGD)

- SGD is an optimization algorithm → source of many popular training algorithms, e.g., backprop for DNNs
- LMS is a special case of SGD

$$\boldsymbol{w}_{n+1} = \boldsymbol{w}_n + \mu(-\widehat{\boldsymbol{\nabla}}_n)$$

- LMS minimizes the mean squared error $J(\boldsymbol{w}) = E[e^2[n]]$
- $\widehat{\boldsymbol{\nabla}}_n = \frac{\partial \hat{J}(\boldsymbol{w}_n)}{\partial \boldsymbol{w}_n} = \frac{\partial e^2(n)}{\partial \boldsymbol{w}_n} \Rightarrow$ is the gradient of the instantaneous (stochastic) value $e^2[n]$ of $J(\boldsymbol{w})$
- many variants of LMS/SGD possible → just modify $\widehat{\boldsymbol{\nabla}}_n$

- easy to obtain a variety of LMS variants → SGD is robust to approximations
- Most/all variants are reduced complexity or robust versions of LMS. Use the simplest LMS algorithm that meets both accuracy and complexity/resource requirements
- LMS minimizes MSE. But many of its variants may minimize some other cost/loss function
- one simple variant - monitor the convergence of the MSE then power down the **WUD**-block (saves power)
- another variant (burst-mode LMS): turn on the **WUD**-block for L samples/updates in a block of M samples/updates (adjust ratio L/M to match changing data statistics).
- note: in all variants of LMS the **F**-block is always operational.

Normalized LMS

(Conventional LMS)

$$e[n] = d[n] - \boldsymbol{W}^T[n]\boldsymbol{X}[n]$$

$$\boldsymbol{W}[n+1] = \boldsymbol{W}[n] + \mu e[n]\boldsymbol{X}[n]$$

- adjusts μ in response to changes in input power level

$$\mu = \frac{\mu_0}{\alpha + N\sigma_x^2}$$

- α ensures μ doesn't blow up when $\sigma_x \to 0$
- stability bounds change to $0 < \mu < 2$
- equivalent to normalizing $x[n] \to \frac{x[n]}{\sqrt{\alpha+N\sigma_x^2}}$, i.e., standard version of an RV → related to **batch normalization** in DNN training

Gear-shifting LMS

- variable step-size LMS (gear shifting) – tries to achieve fast convergence and high accuracy simultaneously
- Learning rate schedule
- reduces μ as convergence proceeds
- large initial μ speeds up convergence
- small later μ later results in higher accuracy
- practical initialization:

$$\mu = \frac{1}{N\sigma_x^2} \quad \text{(half the stability bound)}$$

- reduce μ by factor of 2 until precision limits are reached

Sign LMS Variants

- all reduce LMS complexity:
- Sign-LMS
  - use sign of error in LMS update: $$\boldsymbol{W}[n+1] = \boldsymbol{W}[n] + \mu\, sign(e[n])\boldsymbol{X}[n]$$
  - minimizes mean absolute error $E\{|e[n]|\}$ (not MSE)
  - more stable than LMS
- Sign-sign-LMS
  - LMS update: $$\boldsymbol{W}[n+1] = \boldsymbol{W}[n] + \mu\, sign(e[n])sign(\boldsymbol{X}[n])$$
- Sign-regressor-LMS
  - LMS update: $$\boldsymbol{W}[n+1] = \boldsymbol{W}[n] + \mu\, e[n]sign(\boldsymbol{X}[n])$$
  - less stable than sign-LMS
  - guaranteed to be stable for Gaussian inputs

Momentum LMS

- these control the ‘memory’ in learning process

- update rule

$$\boldsymbol{W}[n+1] = \boldsymbol{W}[n] + \boldsymbol{U}[n]$$
$$\boldsymbol{U}[n] = \theta\boldsymbol{U}[n-1] + \mu e[n]\boldsymbol{X}[n]$$

- $\theta \approx 0.9$
- helps accelerate SGD

Leaky LMS

- implements weight decay
- minimizes:

$$\hat{J}(\boldsymbol{W}) = e^2(n) + \lambda \boldsymbol{W}^T \boldsymbol{W}$$

$\rightarrow$ second term is called a regularizer, i.e., creates a preference for low-norm $\boldsymbol{W}$

- update rule:

$$\boldsymbol{W}[n+1] = (1 - \lambda\mu)\boldsymbol{W}[n] + \mu\, e[n]\boldsymbol{X}[n]$$

Delayed LMS

- Feedback loop in LMS limits the throughput
- Delayed LMS → enables fine-grain pipelining of LMS feedback loop…(M is the delay factor)

$$\boldsymbol{W}[n+1] = \boldsymbol{W}[n-M] + \mu e[n-M]\boldsymbol{X}[n-M]$$

Block (Batch) LMS

- Block/batch LMS $\rightarrow$ update weights once in L samples

$$\boldsymbol{W}[Lk+1] = \boldsymbol{W}[Lk] + \frac{\mu}{L}\sum_{i=0}^{L-1} e[Lk-i]\boldsymbol{X}[Lk-i]$$

- reduces noise in the gradient estimate and hence the updates

Multi-Stage Network

DNN as a Multi-Stage Predictor

- Optimal coefficients for a single-stage predictor can be calculated. LMS can be used to learn those from data → how about multi-stage predictor? A DNN is a multi-stage non-linear predictor.

- Consider a 2-stage linear predictor

2-Stage Linear Predictor (DLP)

- linear stage 1: $s_n = w_1 x_{n-1} + w_2 x_{n-2}$
- linear stage 2: $\hat{x}_n = w_3 s_n + w_4 s_{n-1}$
- minimize: $L(w_1, w_2, w_3, w_4) = E\left[\left(\frac{1}{2} e_n^2\right)\right] = E\left[\frac{1}{2}(\hat{x}_n - x_n)^2\right]$
- $\hat{x}_n = w_1 w_3 x_{n-1} + (w_1 w_4 + w_2 w_3) x_{n-2} + w_2 w_4 x_{n-3} = a_1 x_{n-1} + a_2 x_{n-2} + a_3 x_{n-3}$
- unique values of a_1, a_2, a_3 can be obtained as the **Wiener-Hopf solution** $\boldsymbol{R}^{-1}\boldsymbol{p}$
- but how about w_1, w_2, w_3, w_4? → have underdetermined system of equations → 3 equations vs. 4 unknowns → infinite number of solutions!
- multi-stage networks, e.g., tend to have multiple solutions

Example

- Assume $x_n = 0.1u_n + 0.5u_{n-1} + 0.1u_{n-2} + 0.5u_{n-3}$
- $\boldsymbol{R} = \begin{bmatrix} 0.52 & 0.15 & 0.26 \\ 0.15 & 0.52 & 0.15 \\ 0.26 & 0.15 & 0.52 \end{bmatrix}, \boldsymbol{p} = \begin{bmatrix} 0.15 \\ 0.26 \\ 0.05 \end{bmatrix}, \boldsymbol{a} = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} 0.2284 \\ 0.4792 \\ -0.1562 \end{bmatrix}$
- $\boldsymbol{w_1} = \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} = \begin{bmatrix} c \\ -0.2869c \end{bmatrix}, \boldsymbol{w_2} = \begin{bmatrix} w_3 \\ w_4 \end{bmatrix} = \begin{bmatrix} \frac{0.2284}{c} \\ \frac{0.5447}{c} \end{bmatrix}$, where c is scalar
- $\boldsymbol{w_1} = \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} = \begin{bmatrix} c \\ 2.3853c \end{bmatrix}, \boldsymbol{w_2} = \begin{bmatrix} w_3 \\ w_4 \end{bmatrix} = \begin{bmatrix} \frac{0.2284}{c} \\ \frac{-0.0655}{c} \end{bmatrix}$, where c is scalar
- Sweeping c gives rise to many optimum solutions all of which have the same J_{min}

SGD-based Update Rule

- Stage 1: $s_n = w_1 x_{n-1} + w_2 x_{n-2}$;   Stage 2: $\hat{x}_n = w_3 s_n + w_4 s_{n-1}$
- Minimize: $L(w_1, w_2, w_3, w_4) = E\left[\left(\frac{1}{2} e_n^2\right)\right] = E\left[\frac{1}{2}(\hat{x}_n - x_n)^2\right]$

Weight Gradients for Stage 1

$$\frac{\partial e_n}{\partial w_1} = \frac{\partial e_n}{\partial \hat{x}_n}\left(\frac{\partial \hat{x}_n}{\partial s_n}\frac{\partial \hat{s}_n}{\partial w_1} + \frac{\partial \hat{x}_n}{\partial s_{n-1}}\frac{\partial \hat{s}_{n-1}}{\partial w_1}\right)$$

$$= (\hat{x}_n - x_n)(w_3 x_{n-1} + w_4 x_{n-2})$$

$$\frac{\partial e_n}{\partial w_2} = \frac{\partial e_n}{\partial \hat{x}_n}\left(\frac{\partial \hat{x}_n}{\partial s_n}\frac{\partial \hat{s}_n}{\partial w_2} + \frac{\partial \hat{x}_n}{\partial s_{n-1}}\frac{\partial \hat{s}_{n-1}}{\partial w_2}\right)$$

$$= (\hat{x}_n - x_n)(w_3 x_{n-2} + w_4 x_{n-3})$$

Weight Gradients for Stage 2

$$\frac{\partial e_n}{\partial w_3} = \frac{\partial e_n}{\partial \hat{x}_n}\frac{\partial \hat{x}_n}{\partial w_3} = (\hat{x}_n - x_n)s_n$$

$$= (\hat{x}_n - x_n)(w_1 x_{n-1} + w_2 x_{n-2})$$

$$\frac{\partial e_n}{\partial w_4} = \frac{\partial e_n}{\partial \hat{x}_n}\frac{\partial \hat{x}_n}{\partial w_4} = (\hat{x}_n - x_n)s_{n-1}$$

$$= (\hat{x}_n - x_n)(w_1 x_{n-2} + w_2 x_{n-3})$$

- may converge to any one of the infinite possible solutions…..also seen in DNNs

Course Web Page

https://courses.grainger.illinois.edu/ece598nsg/fa2020/
https://courses.grainger.illinois.edu/ece498nsu/fa2020/

http://shanbhag.ece.uiuc.edu