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

DNN Function

Naresh Shanbhag
Department of Electrical and Computer Engineering
University of Illinois at Urbana-Champaign

http://shanbhag.ece.uiuc.edu

Today

- DNN – function
- a simple inference function - linear prediction

Deep Neural Network - Function

- Big question: how do we decide CNN parameters for a given application?
- Parameter space:
  - L: number of layers
  - n_i: dimension of output of layer i
  - $B_{x,i}, B_{w,i}, B_{o,i}$: precisions of the input, weight, and output of layer i
- Answer: no deep theory to guide the design. Use trial and error, leverage the experience from existing good designs, e.g., LeNet5, for an existing application, e.g., handwritten digit recognition, and AlexNet for image recognition.

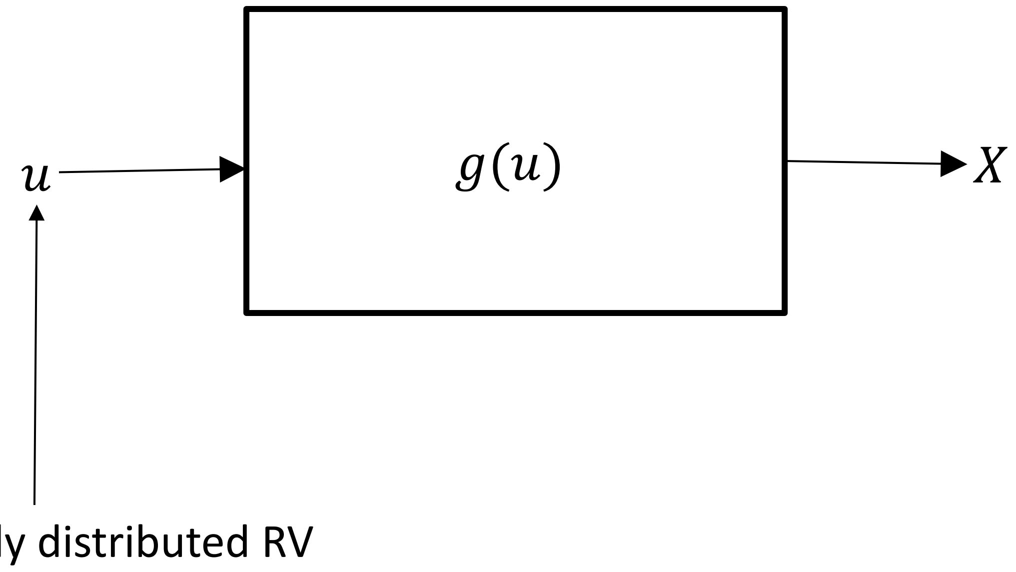
[LeCun, Proc. IEEE, Nov. 1998]

[Krizhevsky, Sutskever, Hinton, NIPS-2012]

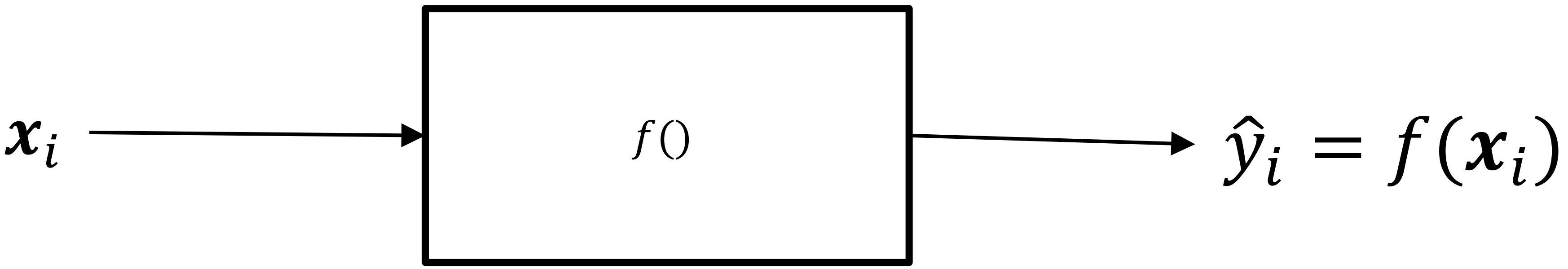
Data Generation Models

Data (*X*)

data generation model



- Data is modeled as a random variable → $X \sim P_X(x)$ (distribution)
- More generally → $(X, Y) \sim P_{XY}(x, y)$ (joint distribution)

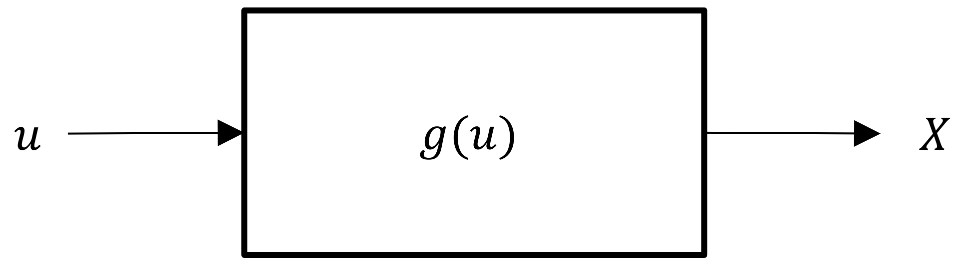


- if $P_X(x)$ is known then all information is known
- use ML or MAP rules (recall ECE 313) for inference

$$\hat{y} = argmax\ P_{Y|X}(y|x) = f(x)$$

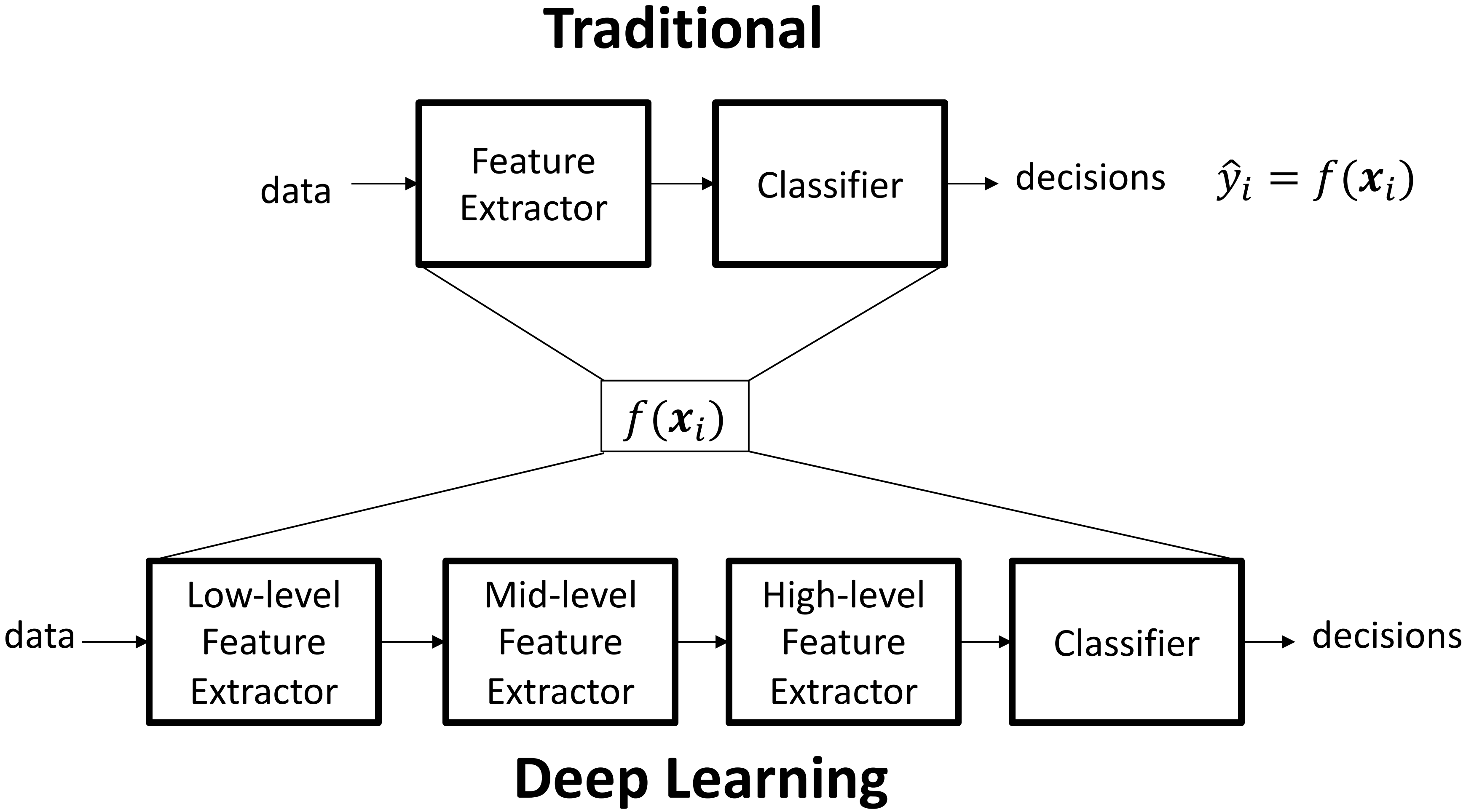
Data (X)

data generation model



- But data distribution is unknown → ML techniques are data-driven
- X is the observed data → develop a data generation model or learn it implicitly and use it for inference
- → ML techniques are data-driven

Traditional vs. Deep Learning



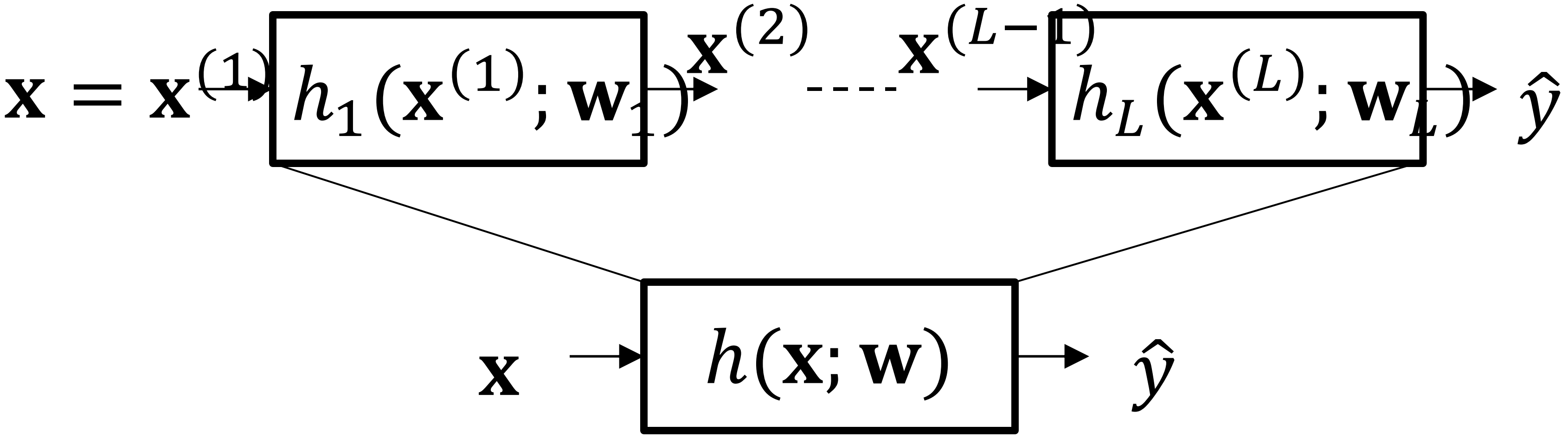
Deep Learning

- machine learning with multiple levels of feature learning starting with the simplest (local) to complex (global) features
- exploits the fact that the world around us is *compositional*
  - image recognition: Pixel → edge → …… → object
- two layer networks are universal approximators

- some functions realized with multiple simple layers may require exponentially complex 2 layers → *exponential advantage of depth*

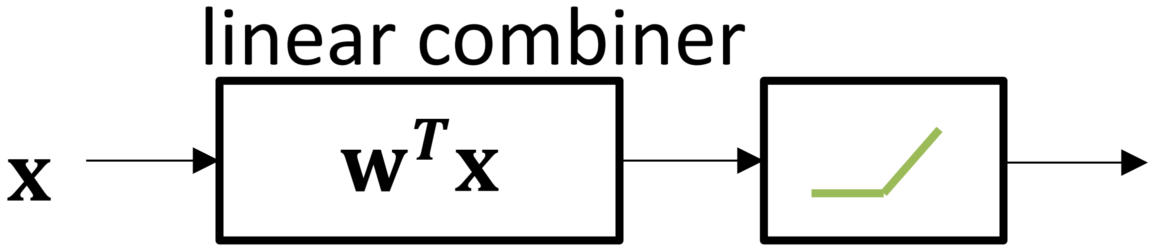
[*Deep Learning*, NIPS 2015 tutorial, Hinton, Bengio, LeCun]

Deep (distributed) Representation



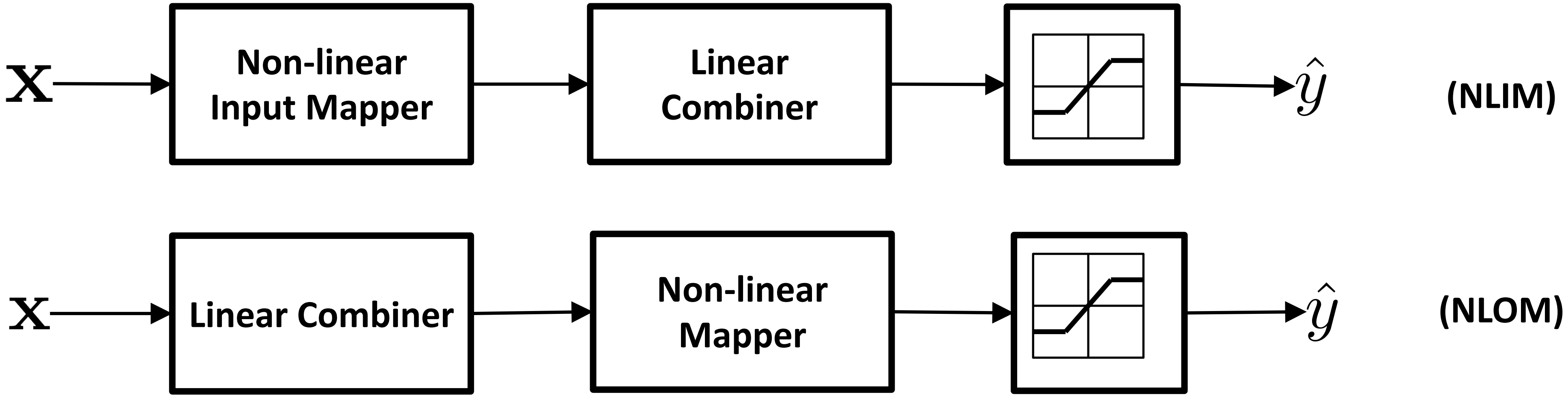
- composition of many functions: $h_i\left(\mathbf{x}^{(i)}; \mathbf{w}_i\right) = f\left(\mathbf{w}_i \mathbf{x}^{(i)}\right)$
- activation function $f(x) \rightarrow$ sigmoid $(1/(1 + e^{-x}))$; hinge function or ReLU $(\max(0, x))$
- overall prediction function $h(\mathbf{x}; \mathbf{w})$
- determine $\mathbf{w}$ that minimizes a loss function, e.g. MSE = $||y - \hat{y}||^2$

Role of Non-linearity in Deep Nets



- non-linear input mapping increases input dimensionality
- separability is enhanced in higher dimensions, e.g., use of ECC in communications
- increases linear combiner's complexity

Types of Non-linearity



- Two approaches:
  - non-linear input mapping + linear combiner (NLIM)
  - linear combiner + non-linear output mapping (NLOM)
- NLIM is more complex than NLOM
- NLOM weights are more constrained than NLIM – employed in Deep Nets

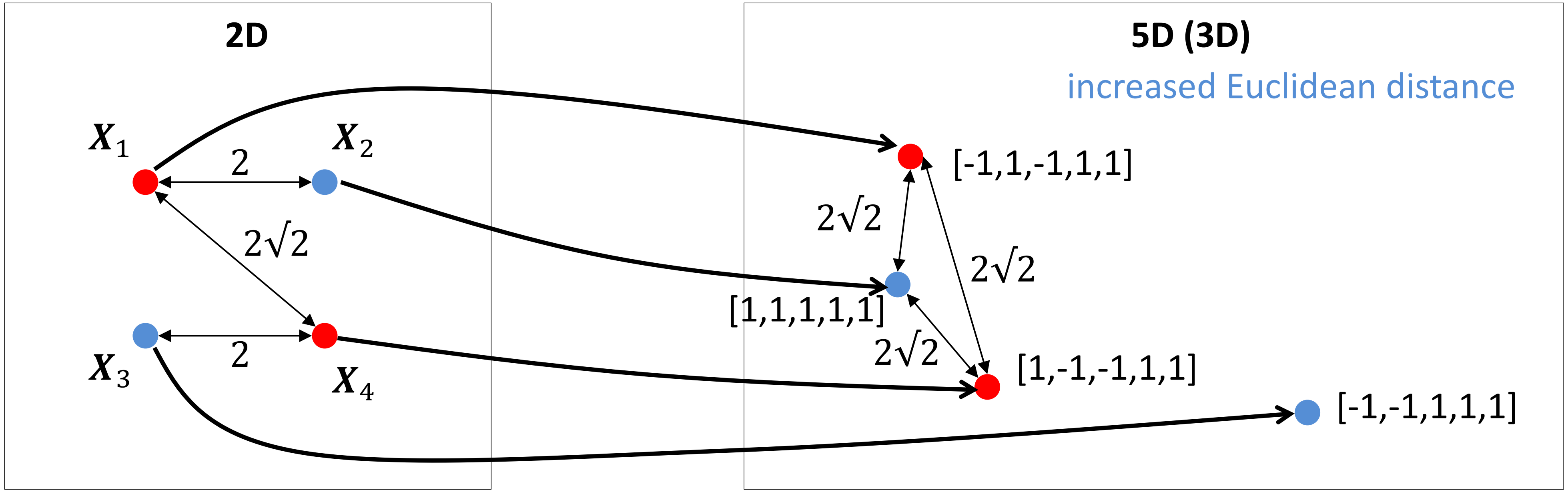
- NLIM Example:

$$\mathbf{x} = [x_1, x_2]^T$$

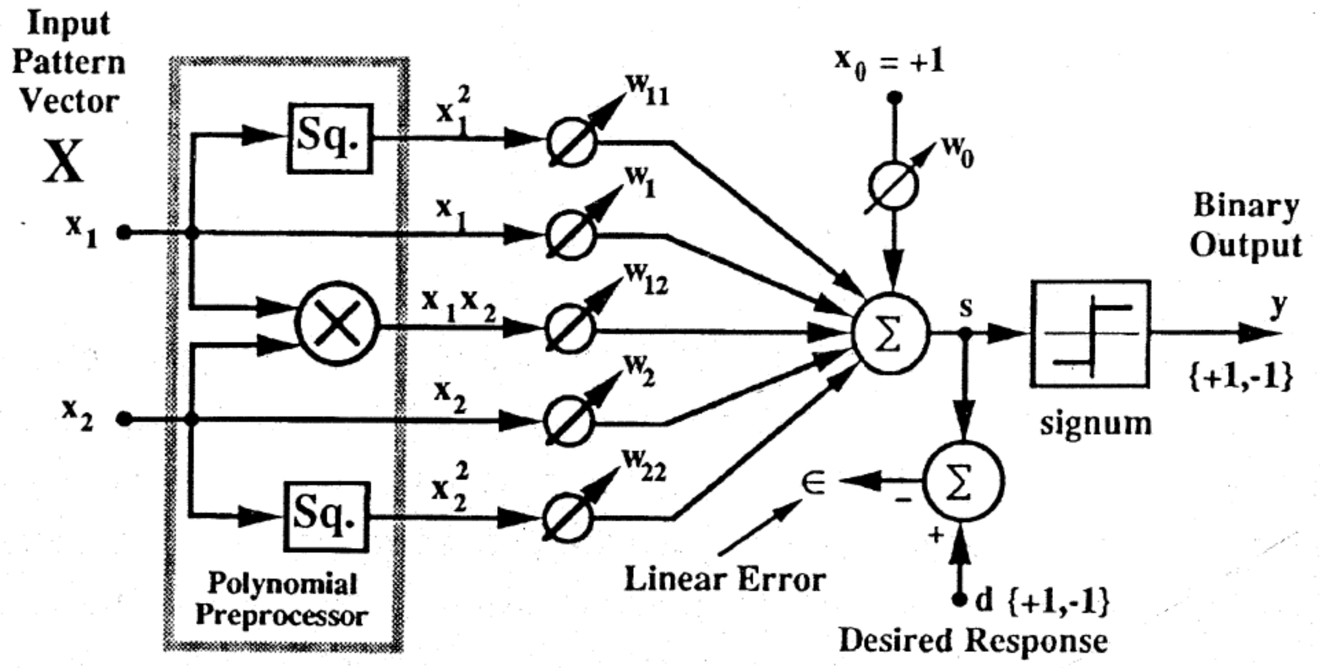
$$\mathbf{x}_{nl} = [x_1, x_2, x_1x_2, x_1^2, x_2^2]^T$$

$$y = \mathbf{w}^T\mathbf{x}_{nl}$$

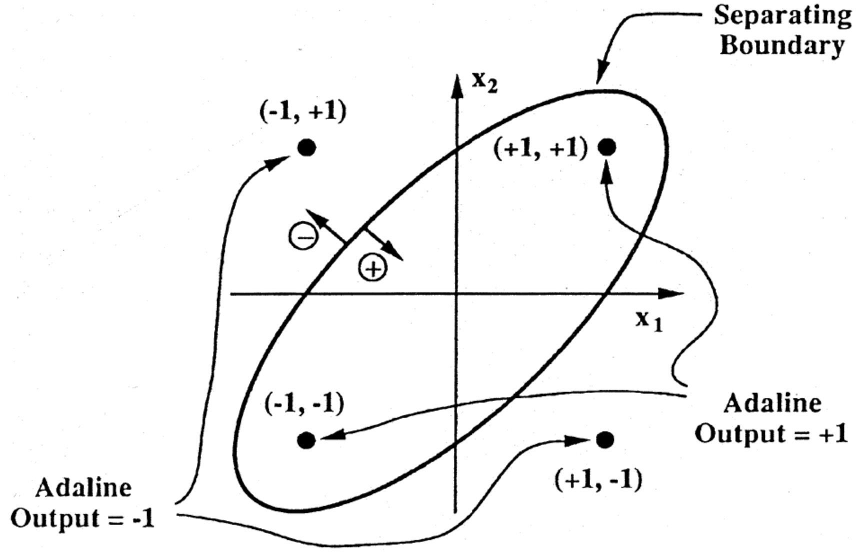
$$= [w_1, w_2, w_3, w_4, w_5]\mathbf{x}_{nl}$$



ADALINE with NLIM



linearly non-separable



- how to choose appropriate input non-linear mapping?

- NLOM Example:

$$\mathbf{x} = [x_1, x_2]^T$$

$$\begin{aligned} y &= (\mathbf{w}^T\mathbf{x})^2 \\ &= (w_1x_1 + w_2x_2)^2 = h_1x_1^2 + h_2x_2^2 + h_3x_1x_2 \end{aligned}$$

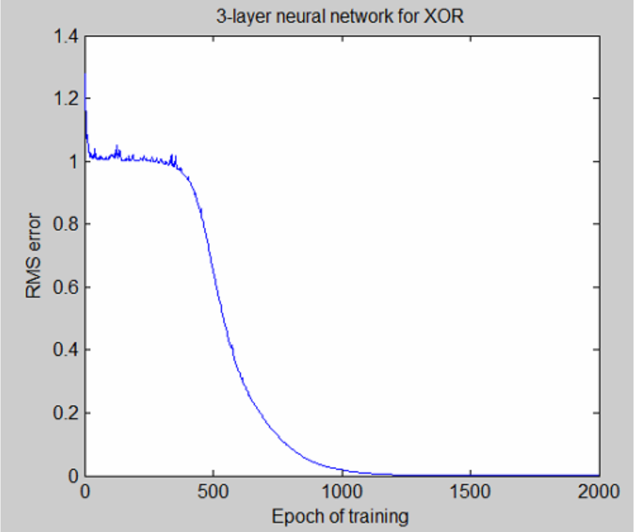
- inclusion of a bias term b inside the square introduces x_1^2 , x_2^2 and b^2 as well
- lower computational complexity than NLIM → 2 MACs+1 squaring vs. 5 MACs+1 multiplication+2 squarings
- restricted class of weight vectors in high dimensions - h's are dependent → smaller representational power than NLIM

Matlab Code of DNN for XOR

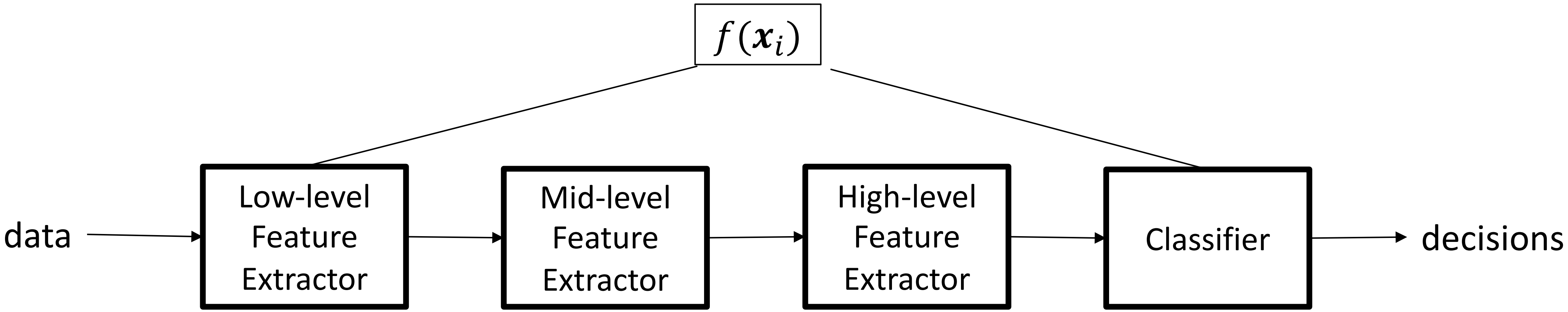
```
clear all;

num_of_layer = 3;
epochs = 2000;

% ------- load in the data -------

% XOR data
train_data = [1 1; 1 0; 0 1; 0 0];
train_label = [1; 0; 0; 1];

num_of_data = size(train_data,1);

%add a bias as an element of input vector
bias = ones(num_of_data,1);
train_data = [train_data bias];

vector_length = size(train_data,2);

% ---------- set weights -----------------
%set initial random weights
W1 = (randn(vector_length,num_of_layer) - 0.5)/10;
W2 = (randn(1,num_of_layer) - 0.5)/10;


for repeat = 1:epochs

   gamma1 = 0.1;
   gamma2 = gamma1 / 10;

   %loop through the num_of_data, selecting randomly
   for j = 1:num_of_data
```

```
%select a random pattern
      data_index = round((rand * num_of_data) + 0.5);

      %set the current pattern
      data_current = train_data(data_index,:);
      out_layer2 = train_label(data_index,1);

      out_layer1_tanh = (tanh(data_current*W1))';
      predicted_out = out_layer1_tanh'*W2';
      error = predicted_out - out_layer2;

      % adjust weight hidden -> output
      delt_layer2 = error.*gamma2 .*out_layer1_tanh;
      W2 = W2 - delt_layer2';

      % adjust the weights input -> hidden
      delt_layer1= gamma1.*error.*W2'.*(1-
(out_layer1_tanh.^2))*data_current;
      W1 = W1 - delt_layer1';

   end


   predicted_out = W2*tanh(train_data*W1)';
   error = predicted_out' - train_label;
   rms_error(repeat) =  (sum(error.^2))^0.5;

   figure(1);
   plot(rms_error)


end
```

3 layer DNN for XOR computation



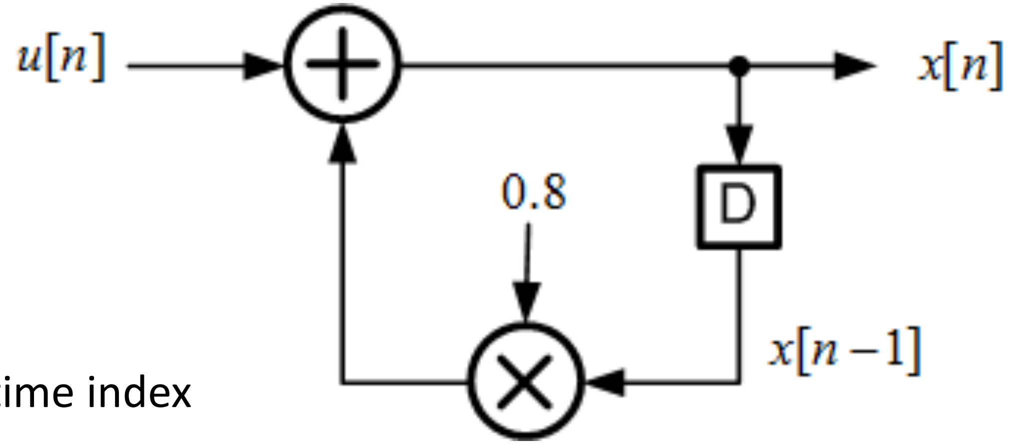
How to find function $f(x)$ from x?



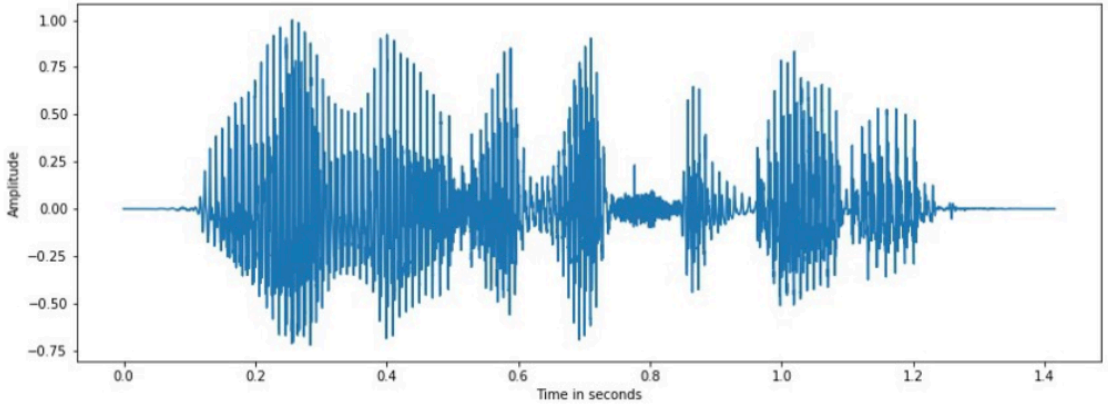
- Can this be obtained analytically? This is hard.
- need lots of data and a learning/optimization algorithm
- A special case – linear predictor has an analytical solution

Linear Predictor (combiner)

Example – Data Generation Model



n: time index

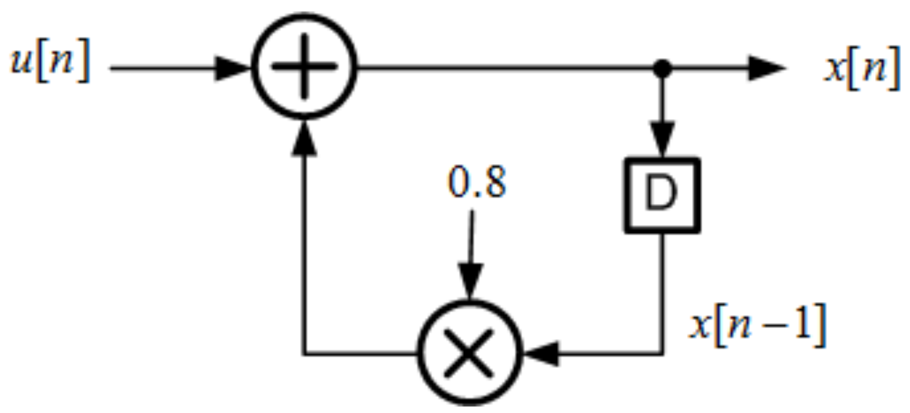


CMU US RMS ARCTIC speech database

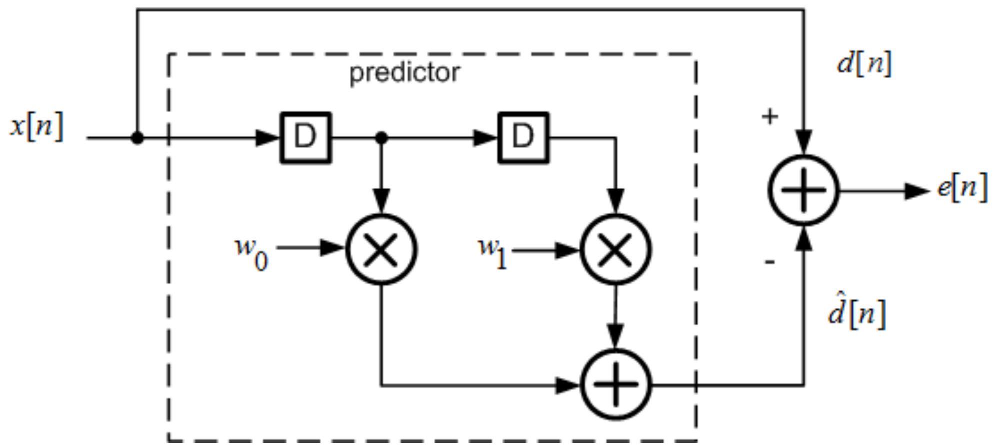
- data generation model: $x[n]$ is the observed data, e.g., pixels in an image or samples of speech
- autoregressive (AR) model: $u[n]$ is a uniformly distributed uncorrelated RV
- models correlated data such as images, video, speech samples very well
- higher order models possible
- can cascade a moving average section to form ARMA models

Example - Predictor

AR model



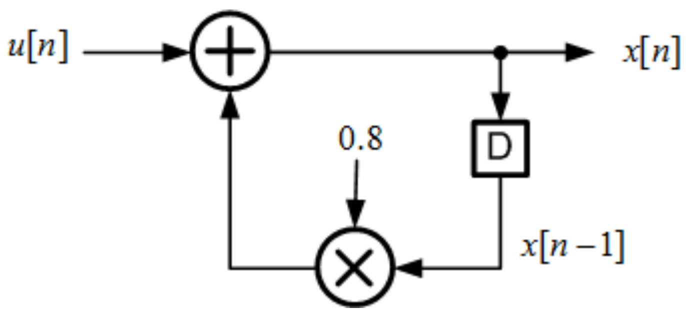
fixed predictor



- 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

- can we figure out the best coefficients by inspection?
- AR model says: $x[n] = u[n] + 0.8x[n-1]$

AR model



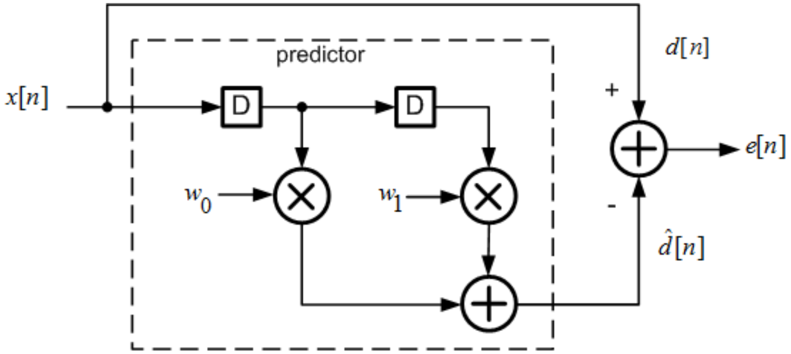
- predictor computes:

$$\hat{d}[n] = w_0 x[n-1] + w_1 x[n-2] \rightarrow x[n]$$

- what should be optimum values of w_0 and w_1?

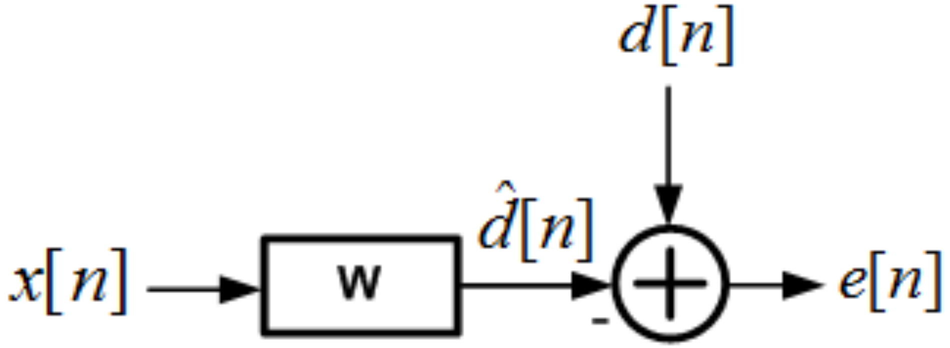
$$w_0 = 0.8; w_1 = 0$$

fixed predictor



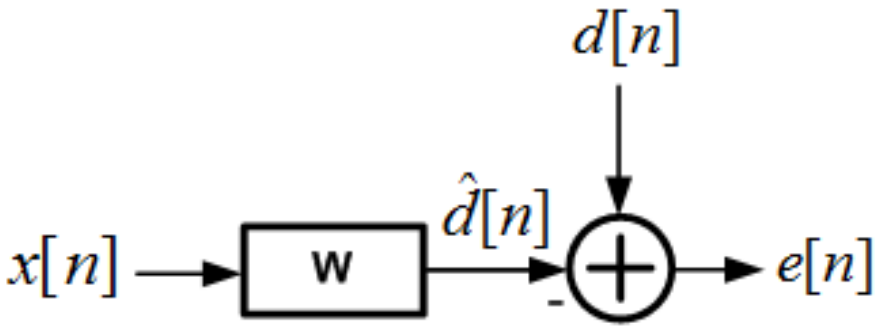
- note: $e[n] = u[n]$, when w_0 and w_1 have optimum values
- needed to know the data model parameters to solve this way

- what are the optimum (in the MMSE sense) weights $\boldsymbol{W}$ of a linear combiner? Difficult question to ask in case of non-linear regressors, e.g., DNNs. Easy for linear combiner.



- note: the time index k is missing in $\boldsymbol{W}$ → we are asking for a fixed weight vector that minimizes the MSE
- the answer should depend upon the statistics of $x[n]$ and the dependence of $d[n]$ on $x[n]$

MMSE Optimum Weights



the Weiner-Hopf (normal) Equation

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

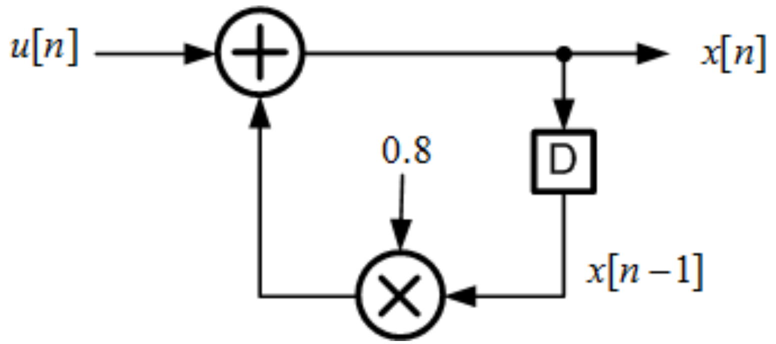
- $\boldsymbol{R} = E\left[\boldsymbol{X}[n]\boldsymbol{X}^T[n]\right]$: the autocorrelation matrix of data vector $\boldsymbol{X}[n]$
- E.g., $\boldsymbol{R} = \begin{bmatrix} \sigma_x^2 & \rho_x(1) \\ \rho_x(1) & \sigma_x^2 \end{bmatrix}$ when $\boldsymbol{X}[n] = \left[x[n] \quad x[n-1]\right]^T$ (vector of length 2) and $x[n]$ is a (wide-sense) stationary process

- $\boldsymbol{p} = E[d[n]\boldsymbol{X}[n]]$: cross-correlation vector capturing the dependence between desired signal $d[n]$ and $\boldsymbol{X}[n]$
- e.g., $\boldsymbol{p} = \begin{bmatrix} \sigma_x^2 \\ \rho_x(1) \end{bmatrix}$ when $d[n] = x[n]$

Example - Calculate *R* and *p*

- from AR model:

$$x[n] = u[n] + 0.8x[n-1]$$



- variance of $x[n]$:

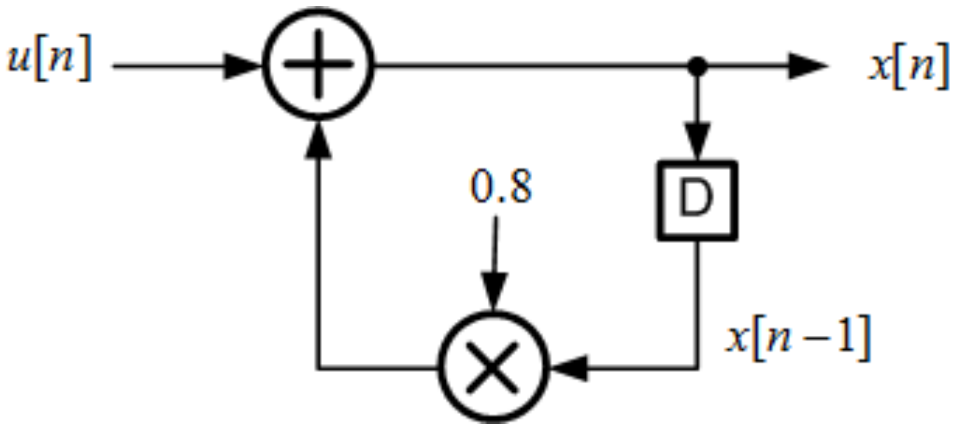
$$\sigma_x^2 = E\{x^2[n]\} = E\{u^2[n] + 1.6u[n]x[n-1] + 0.64x^2[n-1]\}$$
$$= \sigma_u^2 + 0.64\sigma_x^2$$

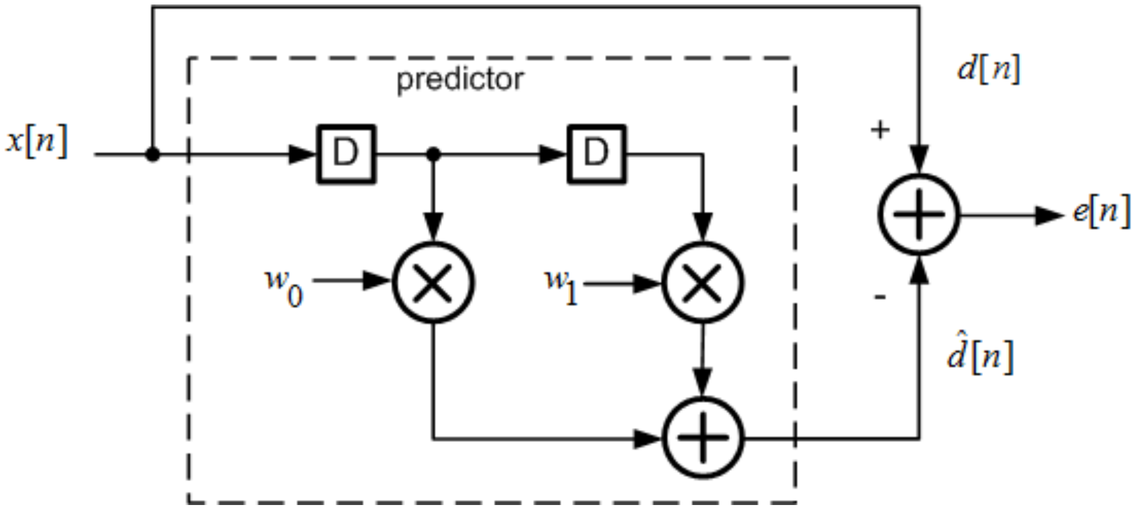
- *u* is unit variance →

$$\sigma_x^2 = \frac{1}{1-0.64} = 2.8$$

$$\rho_x[1] = E\{x[n]x[n-1]\} = E\{(u[n] + 0.8x[n-1])x[n-1]\} = 0.8\sigma_x^2 = 2.24$$
$$\rho_x[2] = E\{x[n]x[n-2]\} = E\{(u[n] + 0.8x[n-1])x[n-2]\} = 0.8\rho_x[1] = 1.8$$

MMSE Predictor

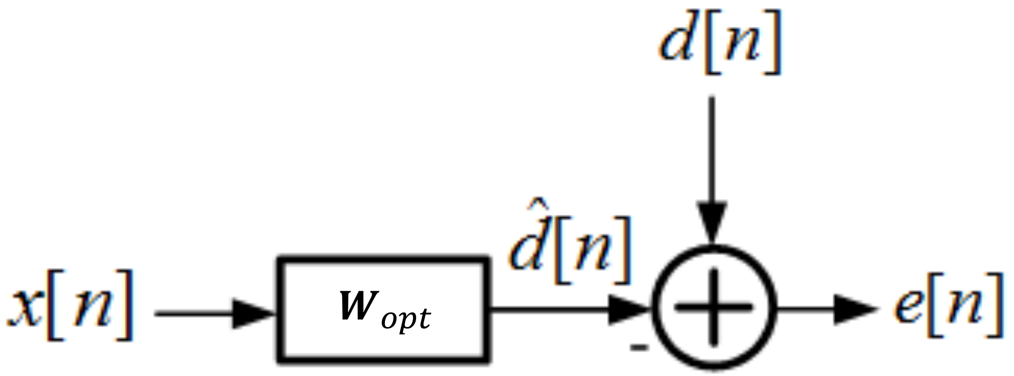




- Solving Wiener-Hopf equations:

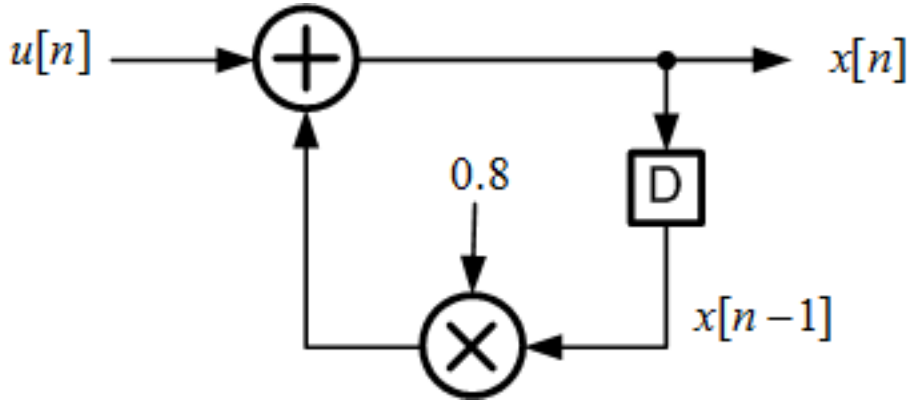
$$\mathbf{R} = \begin{bmatrix} 2.8 & 2.24 \\ 2.24 & 2.8 \end{bmatrix} \qquad \mathbf{p} = \begin{bmatrix} 2.24 \\ 1.8 \end{bmatrix} \qquad \mathbf{w} = \begin{bmatrix} 0.8 \\ 0.008 \end{bmatrix}$$

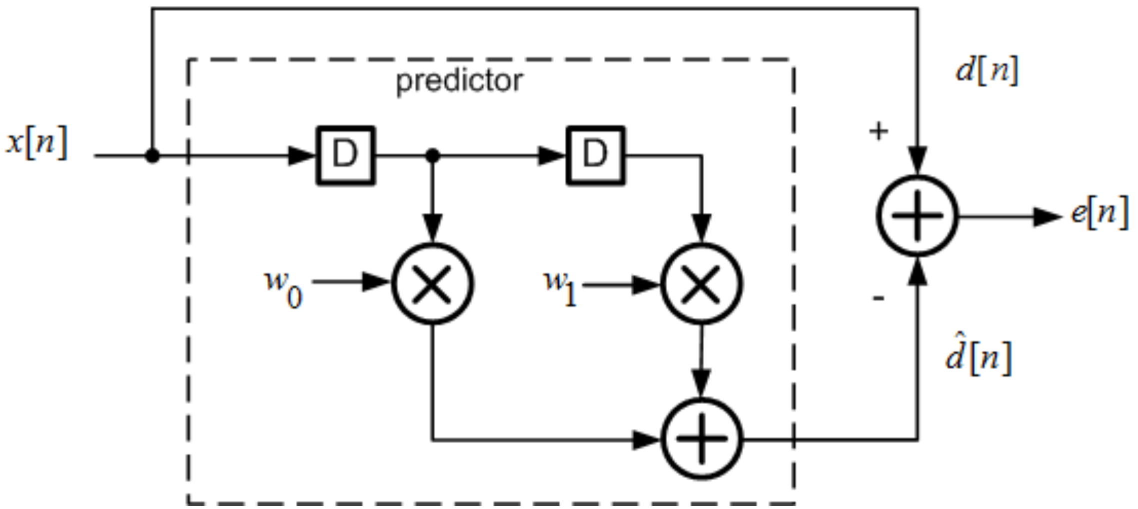
Minimum MSE (MMSE)



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

Example



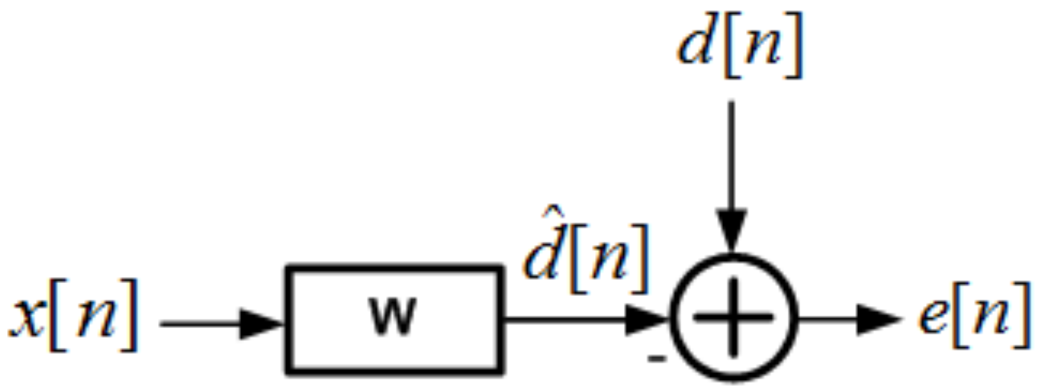


$$\mathbf{R} = \begin{bmatrix} 2.8 & 2.24 \\ 2.24 & 2.8 \end{bmatrix} \quad \mathbf{p} = \begin{bmatrix} 2.24 \\ 1.8 \end{bmatrix} \quad \boldsymbol{W}_{opt} = \begin{bmatrix} 0.8 \\ 0.008 \end{bmatrix}$$

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

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

MMSE Combiner- Summary



- MMSE weight vector: $\mathbf{w}_{opt} = \mathbf{R}^{-1}\mathbf{p}$

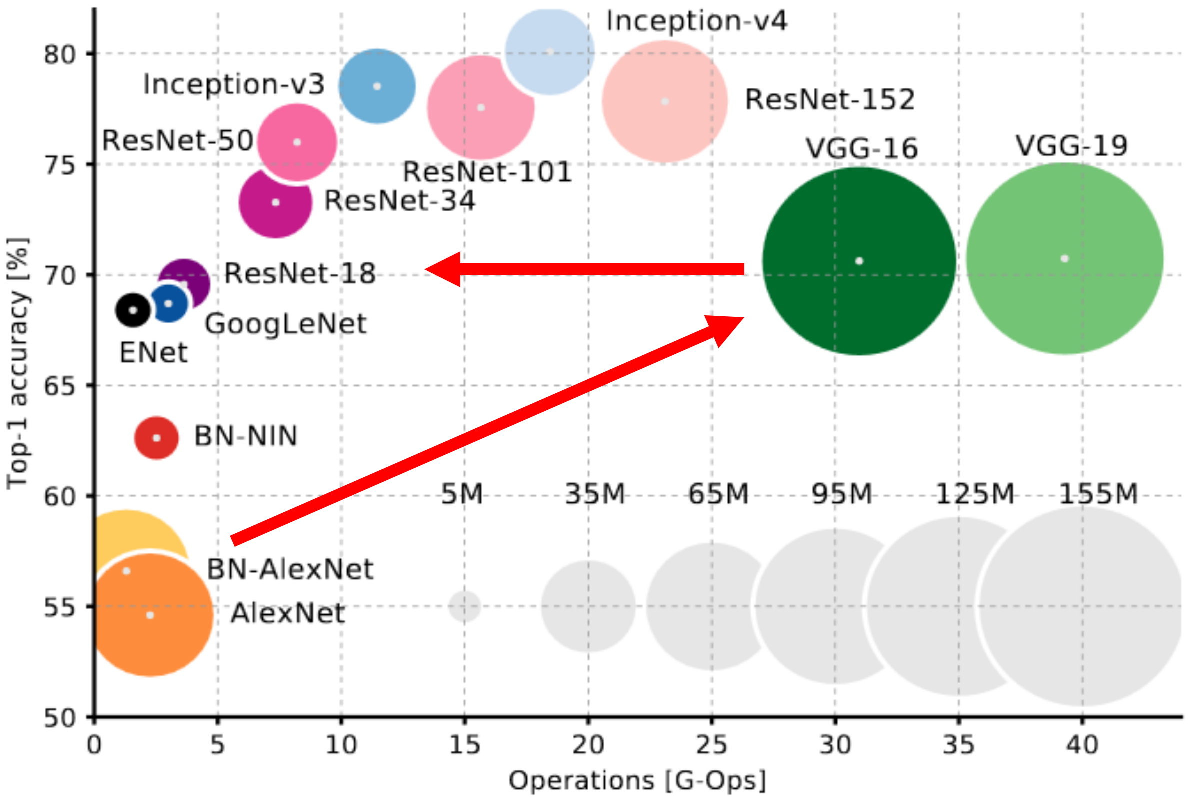
- MMSE: $J_{min} = \sigma_d^2 - \boldsymbol{p}^T \boldsymbol{W}_{opt}$

- can be used only when data statistics are known/can be estimated
- need a learning algorithm when this is not the case

Accuracy vs. Complexity Trade-off

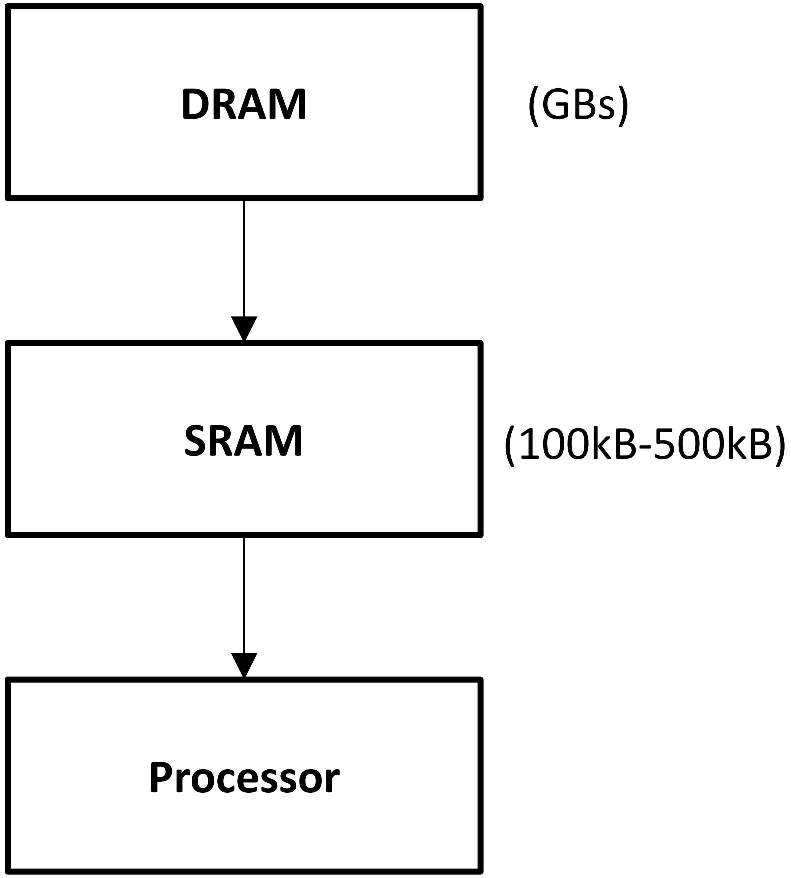
Accuracy vs. Complexity Trade-off in DNNs

[Canziani et al., arXiv2016]



- AlexNet came first – can be thought of as baseline
- VGG-Net achieved high accuracy *at the cost of complexity (storage & compute)*
- GoogleNet and ResNet achieved *high accuracy* while maintaining *moderate complexity*

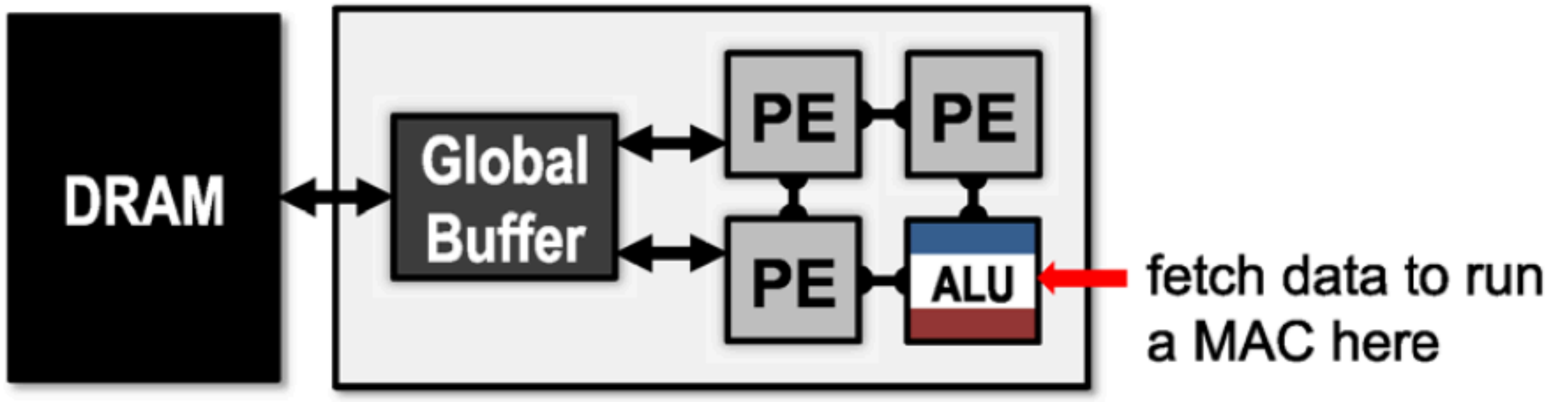
Storage Challenge

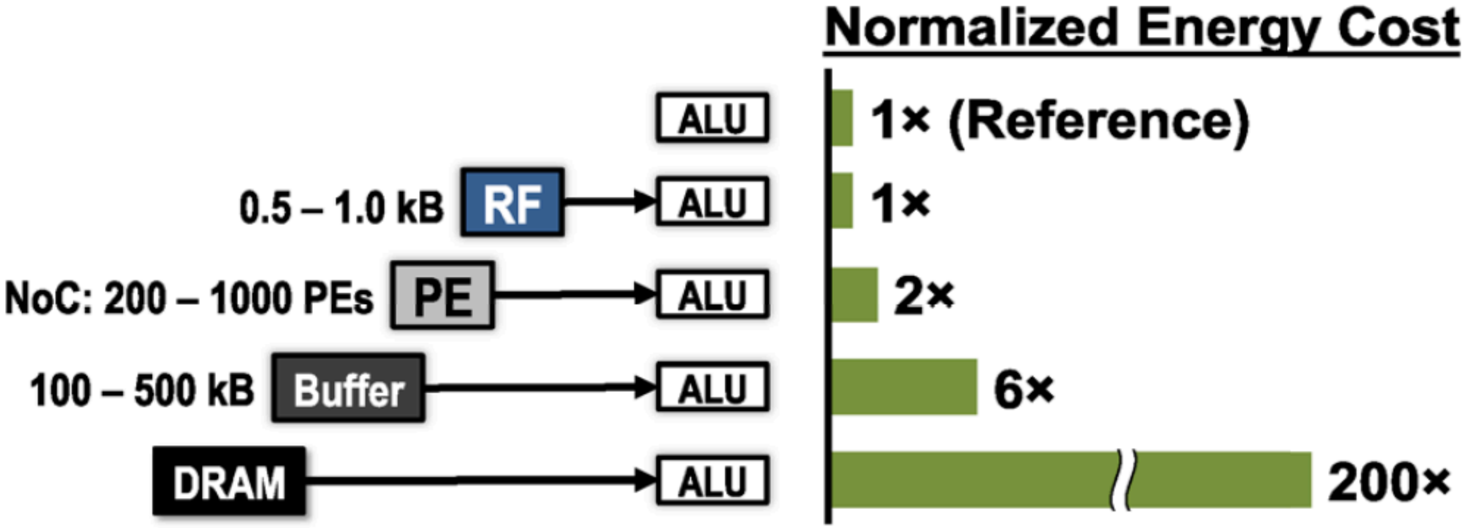


- AlexNet – 63M weights
- 8b/weight → 63MB storage requirements
- overwhelms current on-chip SRAM capacity
- this is inference only – not training

Energy Challenge

[Sze, IEEE Proceedings, December 2017]





- Large model sizes imply a data movement problem:

  DRAM→ SRAM→ PE

- energy and latency costs amplified when data resides far from compute

Computing DNNs in Finite-Precision

- precision reduction is a powerful knob for reducing storage and computational requirements

- Cannot reduce precision arbitrarily:
  - reducing precision impacts inference accuracy
  - some variables are more important than others
  - impact of quantization depends on signal distribution
- Next:
  - quantization of variables
  - number representations
  - fixed-point dot-product

Summary

- DL function – learning and inference steps
- Inference – making predictions/decisions in a data-driven manner (minimal information about data distributions)
- analytical expression for (optimum) DNN function is intractable
  - optimum linear predictor function can be obtained analytically

- energy and latency of inference dominated by data movement
- reduce data movement by implementing DNNs in reduced precision

Course Web Page

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

http://shanbhag.ece.uiuc.edu