Introduction & Motivation

EE599 Deep Learning

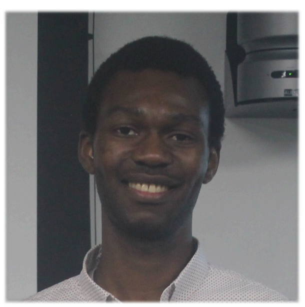
Keith M. Chugg

Spring 2020

USC University of Southern California

Teaching Team

Prof. Keith Chugg (instructor)
Hardware Accelerated Learning



Olaoluwa (Oliver) Adigun (50% TA):
PhD student (Kosko)

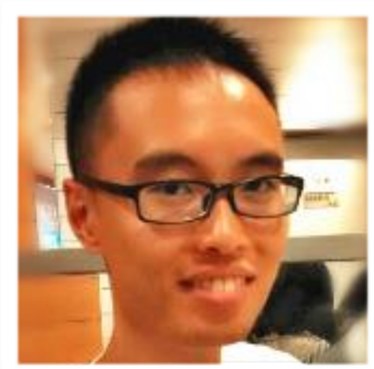
deep learning

Kuan-Wen (James) Huang
(50% TA):
PhD student (Chugg)

deep learning

Arnab Sanyal (25% TA):
PhD student (Chugg/Beerel)

hardware acceleration of Nnets



Jiali Duan (25% TA):
PhD student (Kuo)

computer vision

Syllabus Review and Tools/Websites

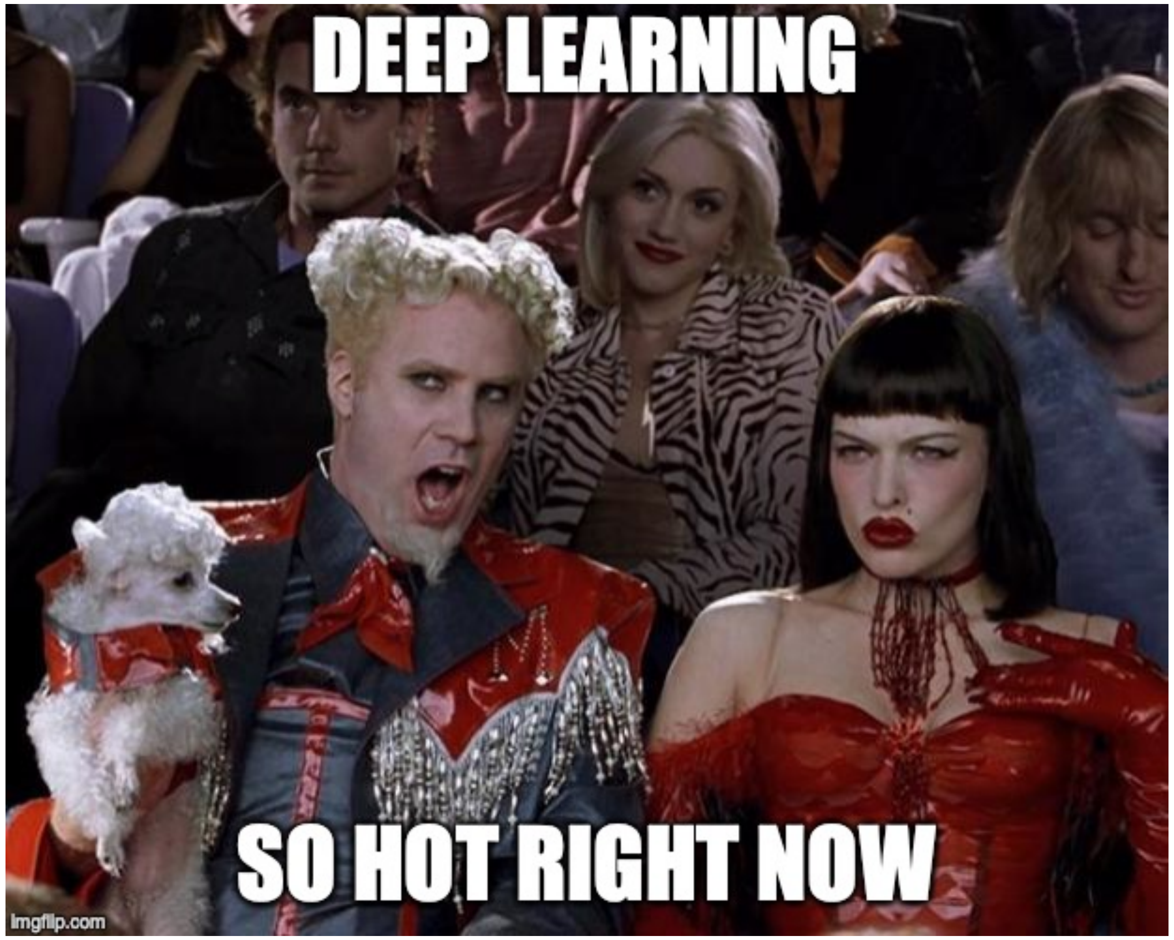
- **Piazza page** (piazza.com/usc/spring2020/ee599chugg/home)
  - Class discussion for all students, instructors, TAs
    - Use Piazza over email whenever possible
  - Handouts, lecture slides, etc.
- **Canvus Page** (URL TBD)
  - All homework assignments and grades (no Blackboard!)
- **AWS Educate** (URL TBD)
  - Used to run long training runs and other computer resources as needed.
  - You will get some AWS credit to train!
- You will have accounts set up for you for each of these

Note on Adding/Dropping

- EE510 and EE503 are pre-reqs for this class
    - students added from waitlist with D-clearances
        - All MSEE students added have pre-reqs.
        - Waitlist period is over
- With these spots and drops, we should be able to add most students who wish to add
    - *Subject to pre-reqs or instructor approval*
- Lecture room is large enough that you can monitor (try-before-buy)

Why this Class?

- Highly popular and relevant for ECE students, yet no Deep Learning class
  - ML sequence in ECE is deep and not focused on neural networks
- Different than CS point of view
  - ML is a combination of ECE topics taught for years, but with a greater emphasis on data
- Place into the context of ECE courses and culture
- Hit the right balance between theory and hands-on programming projects



This class could be titled "Neural Networks with Applications"

Comments on Class Format and Status

- Class will combine theory and practice
  - We will cover some material from:
    - EE562, EE563, EE583, EE517, EE500, EE559, EE660, EE564, EE565, EE588, EE519, EE569
    - Not a replacement for these classes, just for context, tools, and applications
- In Spring 2019, first time for:
  - Deep Learning class at USC Ming Hsieh Department
  - Teaching such a large class (esp. with programming/data projects)
- Lots of experience gained, lessons learned, and materials developed
  - Not a trial course anymore
  - Graded like any other 500-level ECE class

Course Topics (from Syllabus)

- **Course Introduction**
- Estimation and Detection with Statistical Descriptions
- Regression (data fitting)
- Optimization with Steepest Descent
- Multi-Layer Perceptrons (Feedforward Neural Networks)
- Variations on SGD
- Working with Data
- Convolutional Neural Networks
- Recurrent Neural Networks
- Additional Topics (Reinforcement Learning, GANs, etc)

Graphical Outline — Topics

statistical models

MMSE Estimation

Linear/Affine MMSE Est.

FIR Wiener filtering

Bayesian decision theory

Hard decisions

soft decisions (APP)

ML/MAP parameter estimation

Karhunen-Loeve expansion

sufficient statistics

GD, SGD, LMS

data driven

general regression

linear LS regression

stochastic gradient and batches

Classification from data

linear classifier

logistical regression (perceptron)

regularization

PCA

feature design

neural networks

for regression and classification

learning with SGD

working with data

Graphical Outline — Topics

computing skills	homework / projects
getting started with Python	**background topics:** plotting, collecting and exchanging data
numpy, scipy, scikit-learn	**Neural Network Preliminaries:** feedforward inference MLP; BP training MLP; basic model exploration; LMS; data analysis
keras and tf.keras	**Training Neural Networks** collecting and training on data, training CNNs, training RNNs
unix basics & AWS	**Training Neural Networks** GPU-based training using AWS credits

resources

Lecture
no formal programming instruction in (examples given)

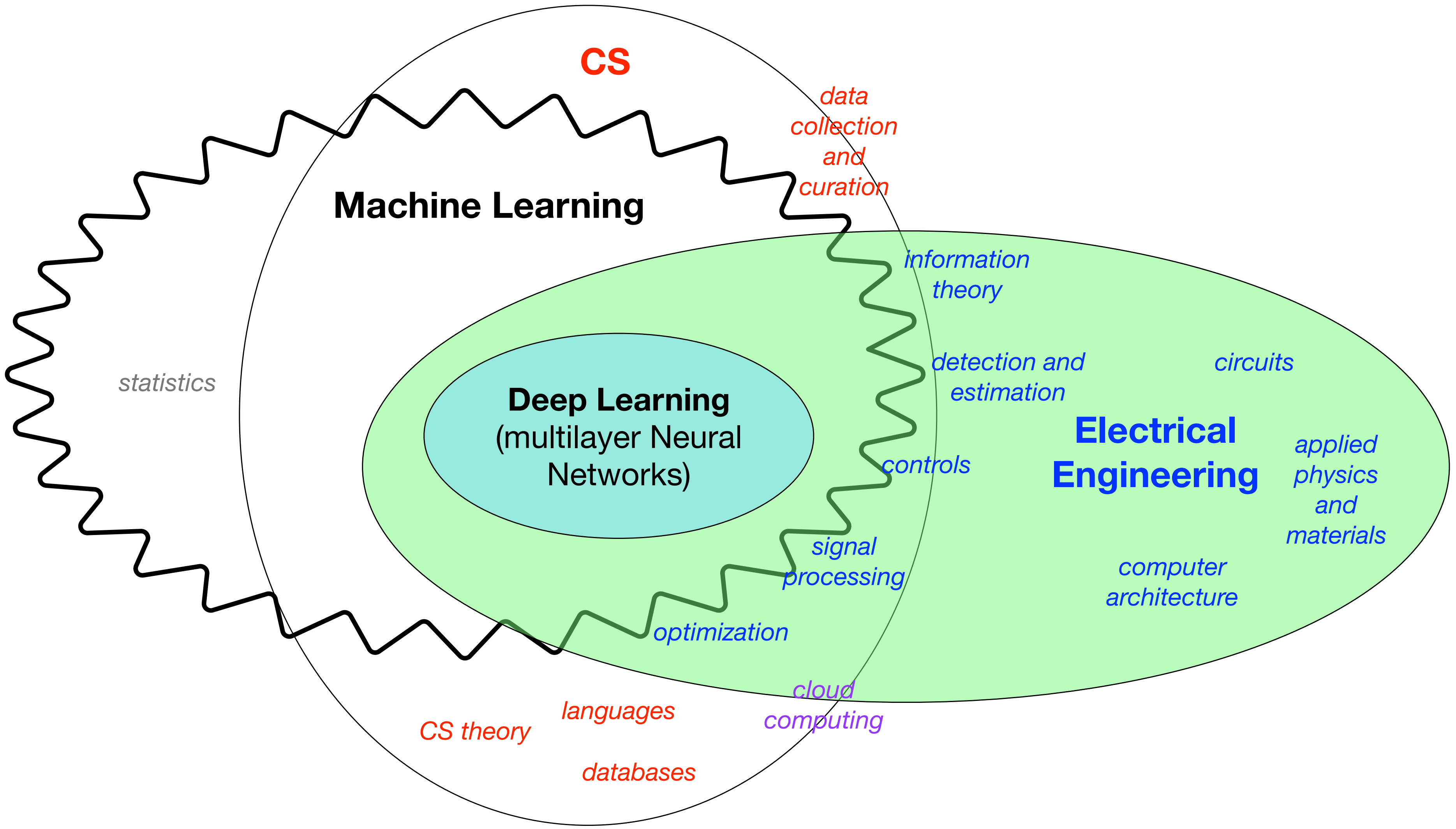
Discussion
helpful tutorials, no formal programming instruction

Supplemental
Office hours, piazza, slackedit, google, PROJECTS, etc.

Course Overview

- Machine Learning, Deep Learning, and ECE
- Classical EE view: Estimation and Decision/Detection Theory
- Regression and Classification from data
- Steepest descent and stochastic gradient descent
- Types of neural networks
- Practical tools and topics
  - Python and important packages
  - AWS
  - Working with data

Machine Learning, Deep Learning and EE



Most ML topics are part of the traditional EE curriculum

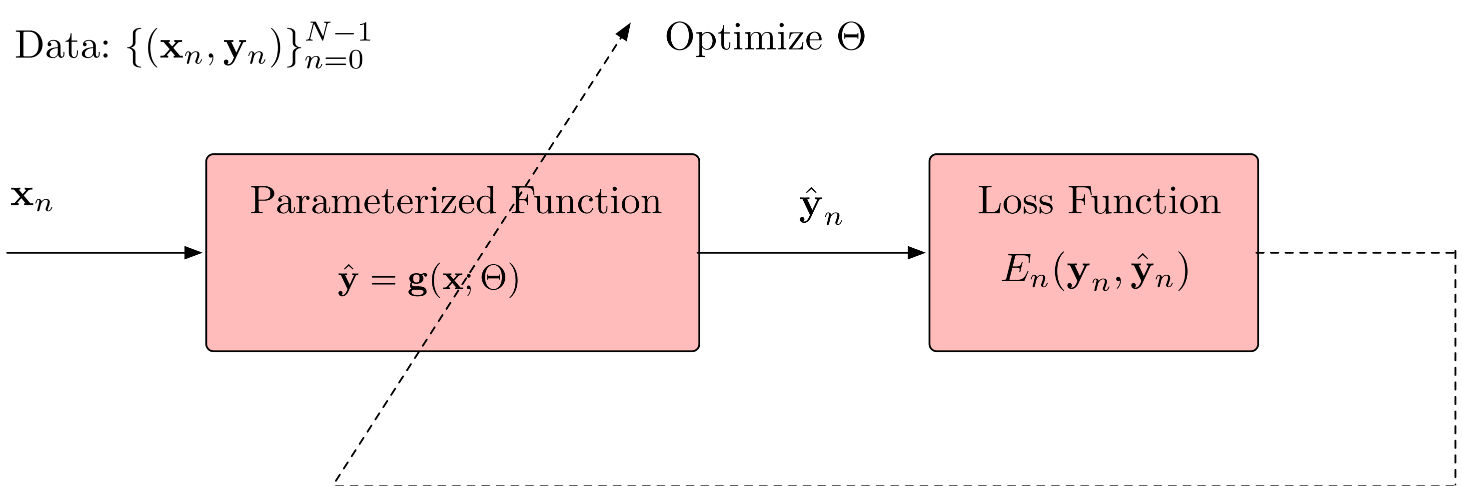
- *Inference (detection/estimation), optimization, pattern recognition*

CS has leveraged and added/emphasized

- *Applications, data, and programming methods and tools*
- *Effective branding and ownership*

Types of ML: Supervised Learning

Training (Learning)

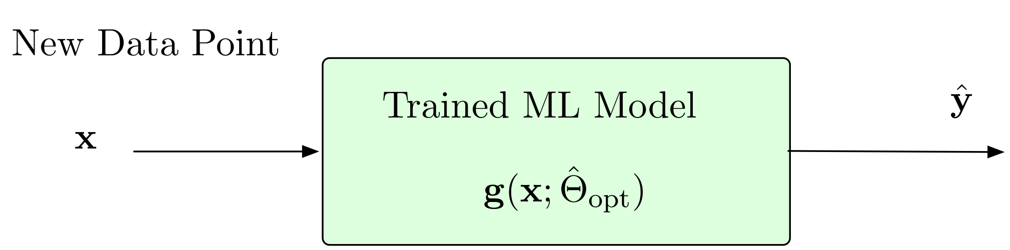


Supervised
Train with input and output (desired response) pairs

$$\mathcal{G} = \{\mathbf{g}(\cdot; \Theta) : \forall \Theta \in \mathbb{R}^D\}$$

"hypothesis set"
(class of possible inference functions)

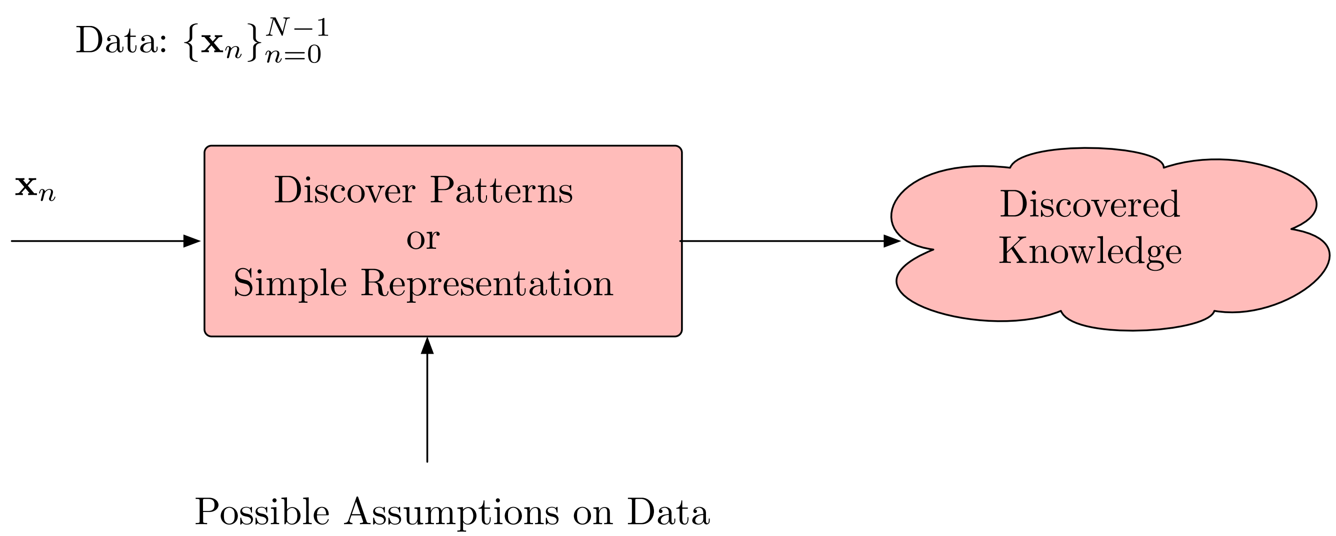
Inference Mode (after trained)



- **Examples**
  - Automatic Speech Recognition (ASR)
  - Image classification
  - Signal filtering and processing

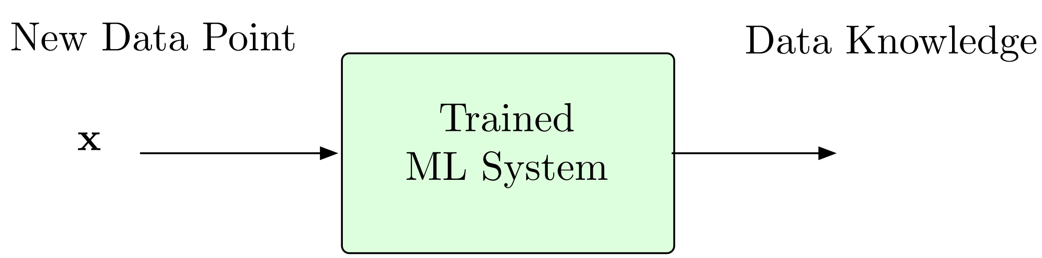
Types of ML: Unsupervised Learning

Training (Learning)



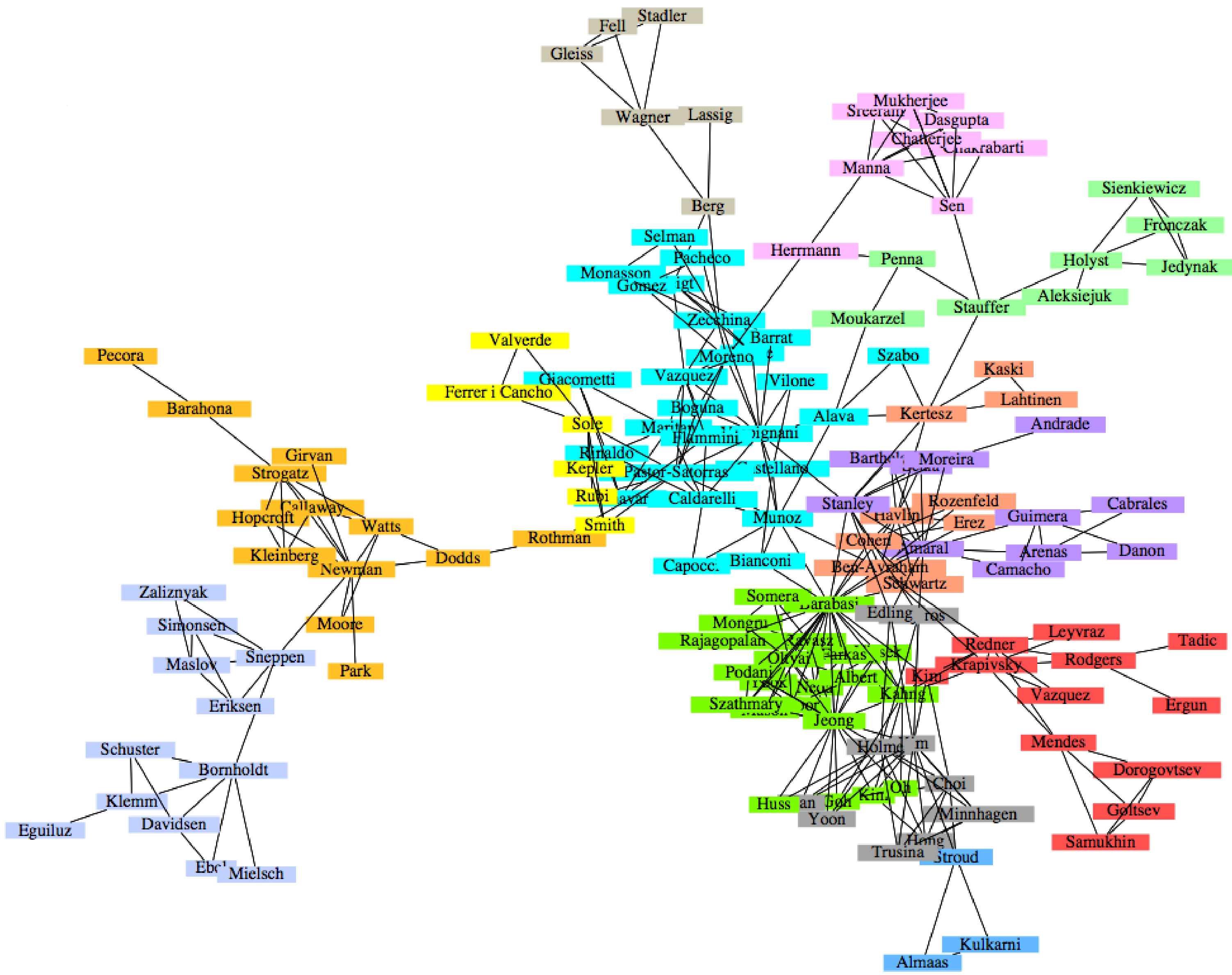
Unsupervised
Train with "input" data only and identify patterns

Inference Mode (after trained)



- **Examples**
  - Community detection in social networks
  - Political donor analysis / targeted ads
  - Sorting photos by people (unknown set)
  - Radio interference (situation awareness)

Types of ML: Unsupervised Learning

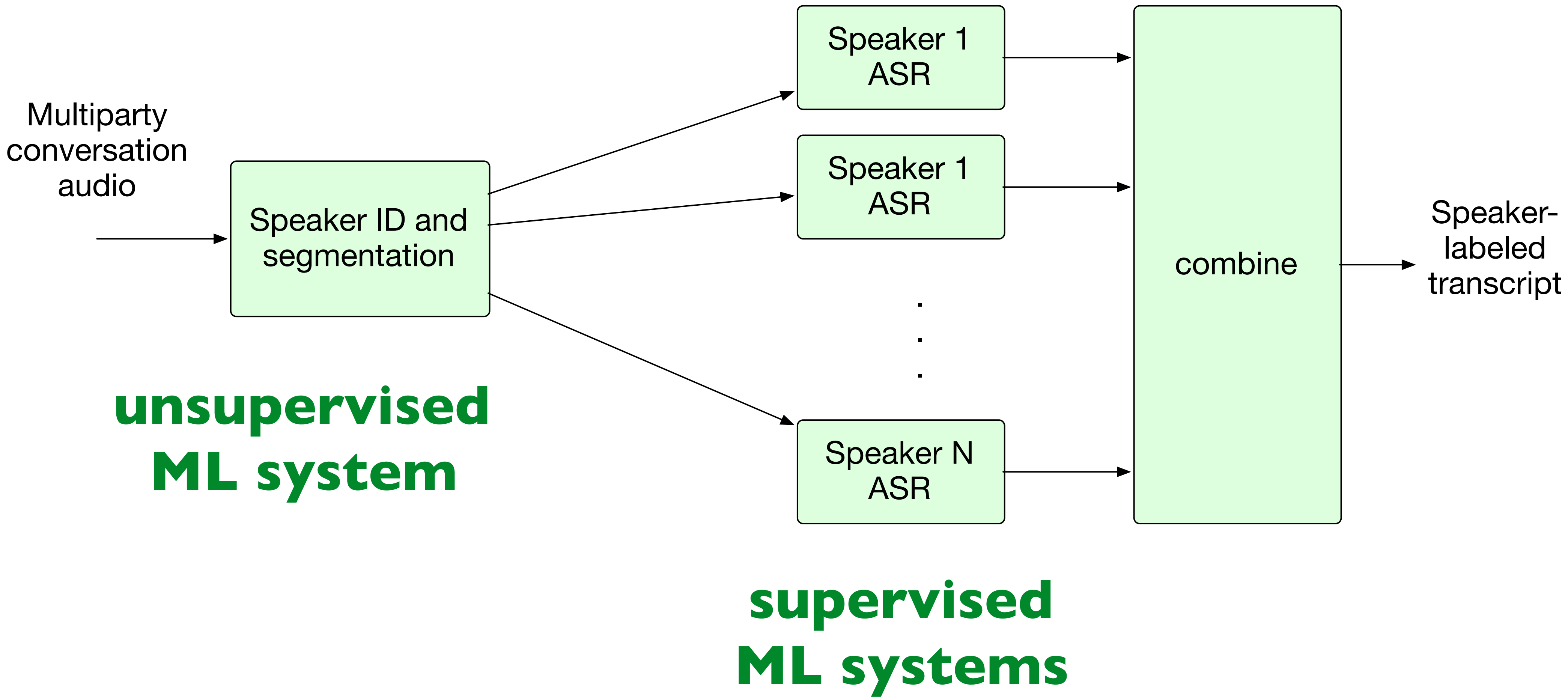


Communities are discovered, but not identified with topics

Figure 3.12: A co-authorship network of physicists and applied mathematicians working on networks [322]. Within this professional community, more tightly-knit subgroups are evident from the network structure.

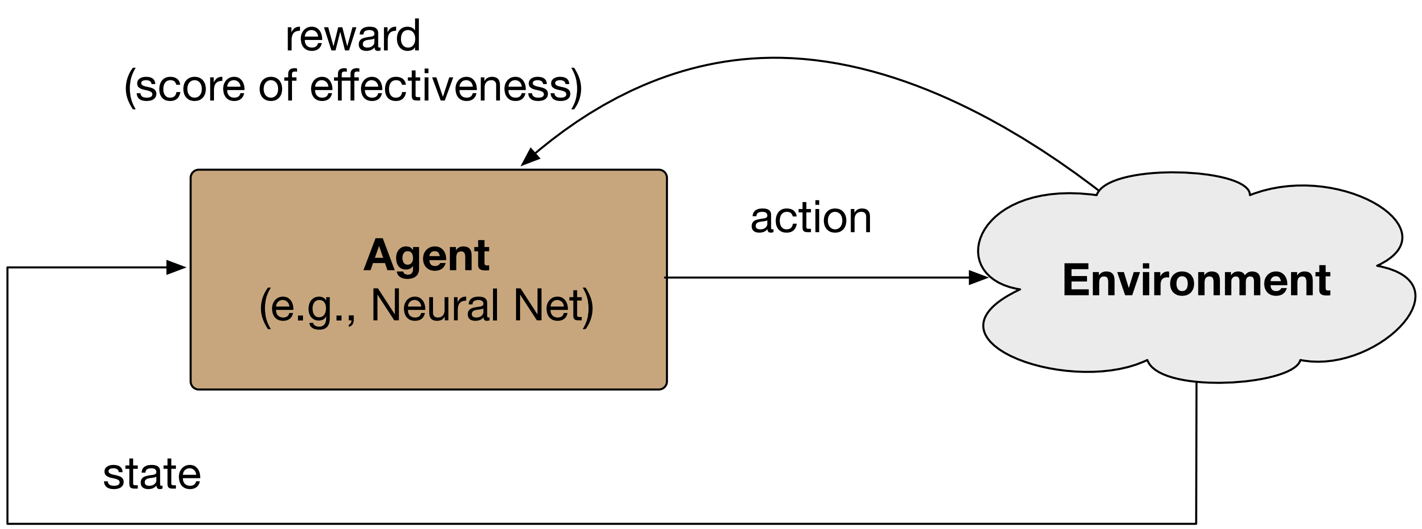
D. Easley and J. Kleinberg, Networks, Crowds, and Markets: Reasoning About a Highly Connected World, Cambridge University Press, 2010.

ML Example: Unsupervised/Supervised



Types of ML: Reinforcement Learning

Learning is typically initialized with supervised model and then refined online



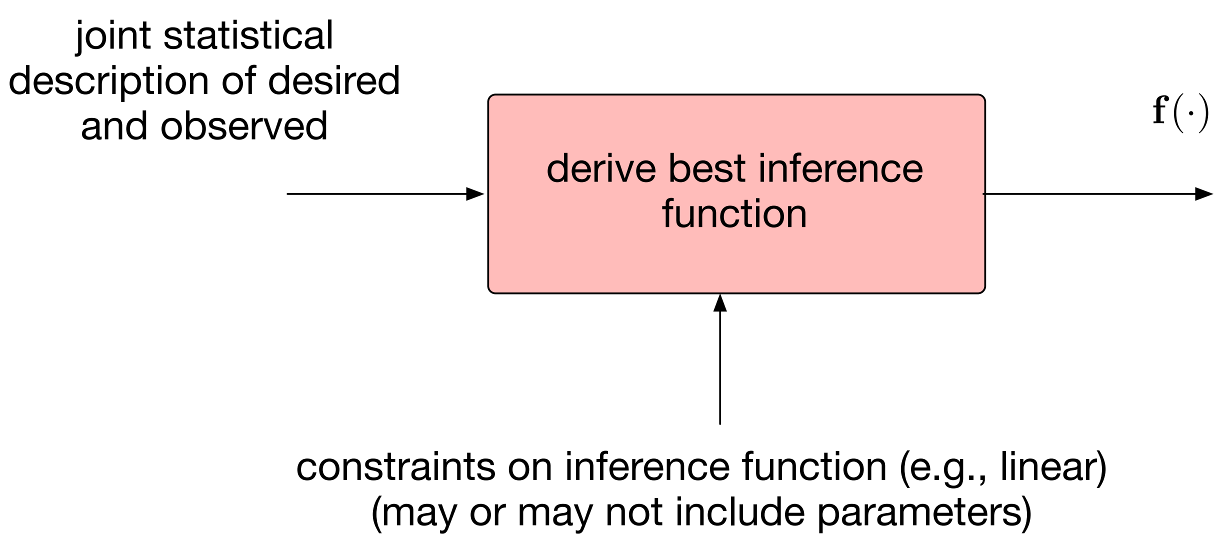
Reinforcement
Learn with feedback from environment

- **Examples**
  - Game playing
  - Autonomous systems navigation

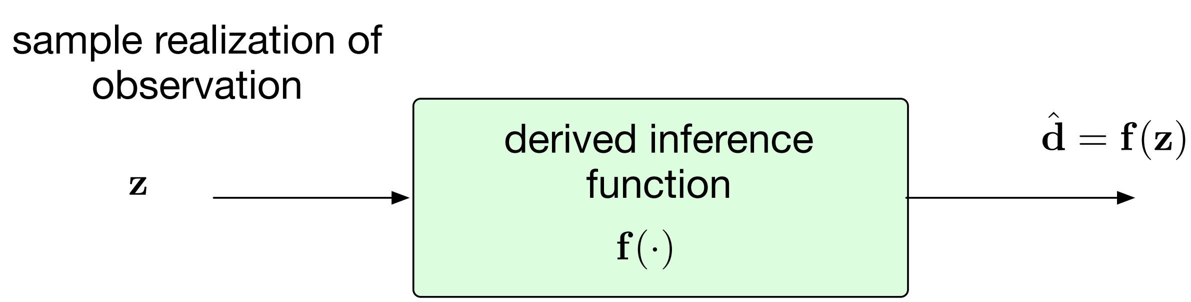
Amazon's Deep Racer

Inference from Statistical Descriptions

Design/Derivation Phase



Inference Mode

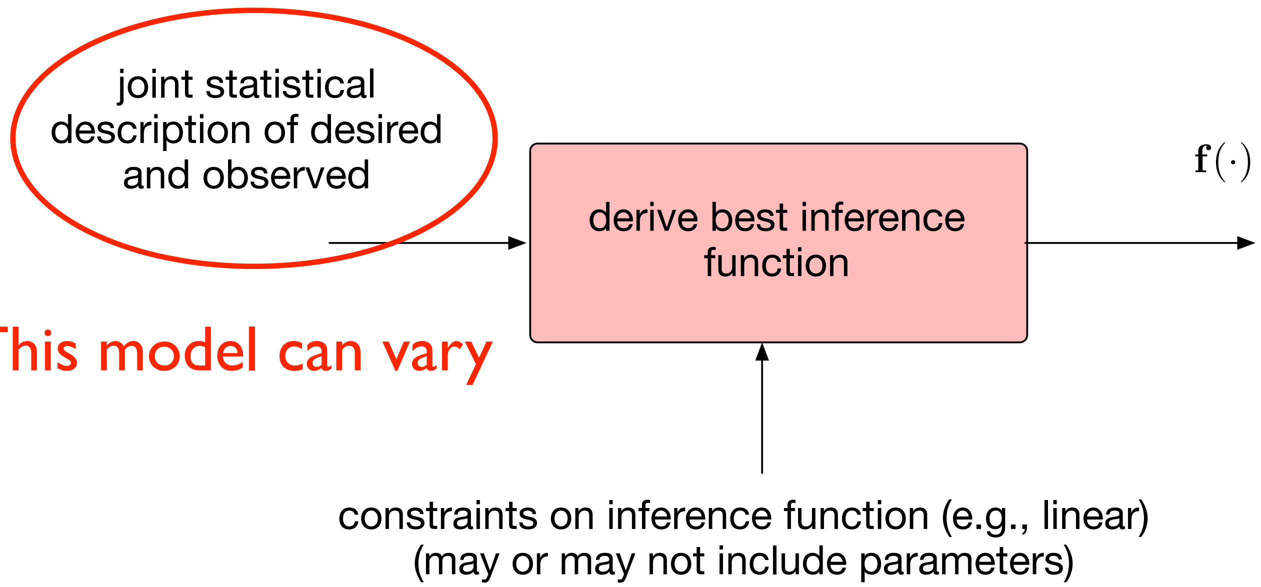


- **Examples**
  - Digital comm, Radar
  - Filtering, prediction, smoothing
  - Algorithms:
    - Kalman Filter
    - Wiener Filter
    - MMSE Estimator
    - Viterbi Algorithm

- Traditionally, EE is heavily based on this approach

Inference from Statistical Descriptions

Design/Derivation Phase



Desired: $\mathbf{d}(u)$

Observed: $\mathbf{z}(u)$

Complete Statistical Description

$p_{\mathbf{d}(u)}(\mathbf{d})$ a-priori distribution

$p_{\mathbf{z}(u)|\mathbf{d}(u)}(\mathbf{z}|\mathbf{d})$ likelihood

Second Moment Description

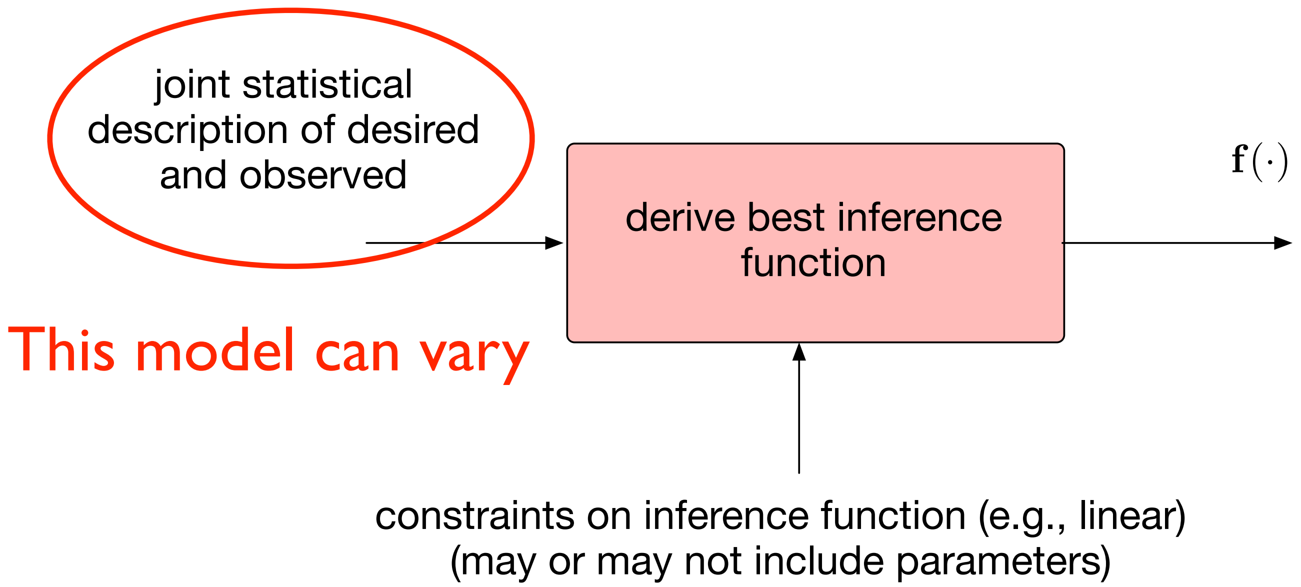
$\mathbf{K}_{\mathbf{z}} = \mathbb{E}\left\{\mathbf{z}(u)\mathbf{z}^{\mathrm{t}}(u)\right\}$ observation covariance matrix

$\mathbf{K}_{\mathbf{dz}} = \mathbb{E}\left\{\mathbf{d}(u)\mathbf{z}^{\mathrm{t}}(u)\right\}$ desired/observation covariance

$\mathbf{m}_{\mathbf{z}} = \mathbb{E}\left\{\mathbf{z}(u)\right\}, \ \mathbf{m}_{\mathbf{d}} = \mathbb{E}\left\{\mathbf{d}(u)\right\}$ means

Inference from Statistical Descriptions

Design/Derivation Phase

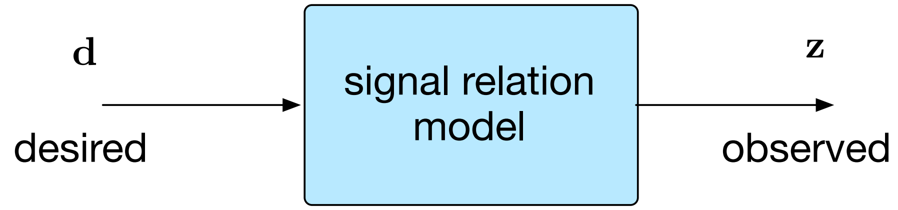


Desired: $\mathbf{d}(u)$

Observed: $\mathbf{z}(u)$

Statistical Model

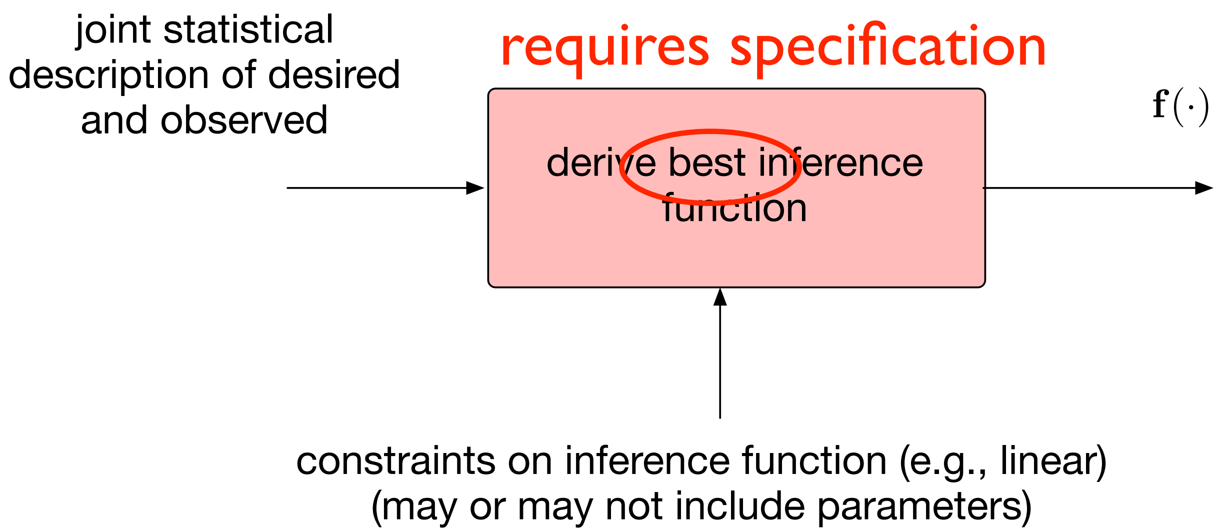
often comes from a signal model



Example: $\mathbf{z}(u) = \mathbf{H}\mathbf{d}(u) + \mathbf{w}(u)$

Inference from Statistical Descriptions

Design/Derivation Phase



Reminder
you should have seen some of this in EE503

Detection/Decision/Classification:
when desired is digital (discrete and finite set)

common performance criterion:
minimize probability of decision error

Estimation:
when desired is continuous

common performance criterion:
minimize mean squared error

Inference from Statistical Descriptions

Summary:

Design/Derive Inference Rule from statistical model for desired/observed

Use Inference Rule on data (realizations of observed)

Primary Result:

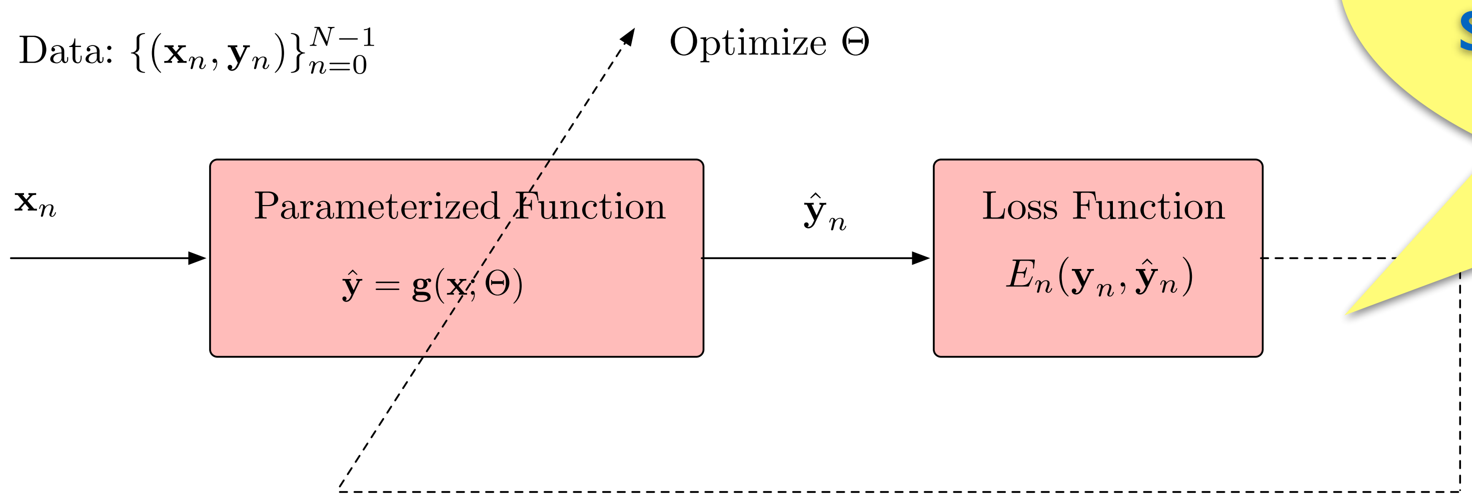
$$\mathbf{f}(\cdot)$$

inference function that is optimal in some defined sense over the ensemble of realizations

$$\hat{\mathbf{d}} = \mathbf{f}(\mathbf{z})$$

inference function used a given realization of the observation

Regression and Classification from Data



Note
same as supervised ML

$$\mathcal{G} = \{\mathbf{g}(\cdot; \Theta) : \forall \Theta \in \mathbb{R}^D\}$$

"hypothesis set"
(class of possible inference functions)

Classification:
when **y** is digital (discrete and finite set)

common performance criterion:
minimize cross entropy cost

Regression:
when **y** is continuous

common performance criterion:
minimize average squared error

Relation Between Two Views

Inference from Statistical Models:

Inference from Data (Supervised ML):

Inference from Statistical Models			Inference from Data (Supervised ML)
observation:	$\mathbf{z}$	$\mathbf{x}$	regressor or input
desired:	$\mathbf{d}$	$\mathbf{y}$	target or output
inference function (designed):	$\mathbf{f}(\cdot)$	$\mathbf{g}(\cdot;\Theta)$	parameterized inference function (Theta learned)

Connection

- **Probabilistic Viewpoint:**
  - Given statistical model: generate lots of realizations for (d,z)
  - Use (x=z, y=d) as data and perform regression/classification from data
  - If **g**(.;Theta) is rich enough, **g**(.;Theta) ~= **f**(.)

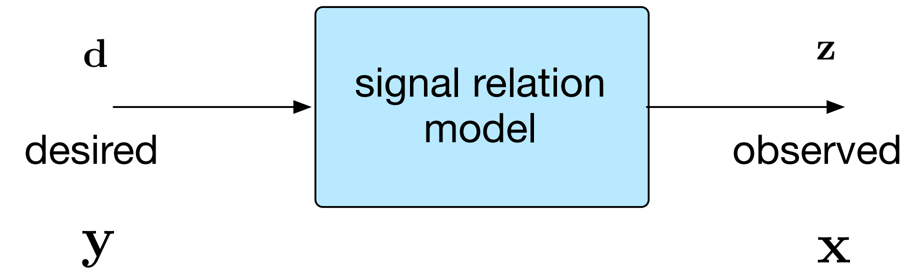
Note on Notation

Statistical Model

Desired: $\mathbf{d}(u)$ $\quad\quad$ $\mathbf{x}(u)$

Observed: $\mathbf{z}(u)$ $\quad\quad$ $\mathbf{y}(u)$

often comes from a (forward) signal model



It is awkward to think of input **y** and output **x** for the signal model

x and **y** are engrained in the data-driven models (regression aka curve fitting)

will maintain two separate notations for now (z, d) and (x,y), but will use just (x,y) eventually

Regression from Data

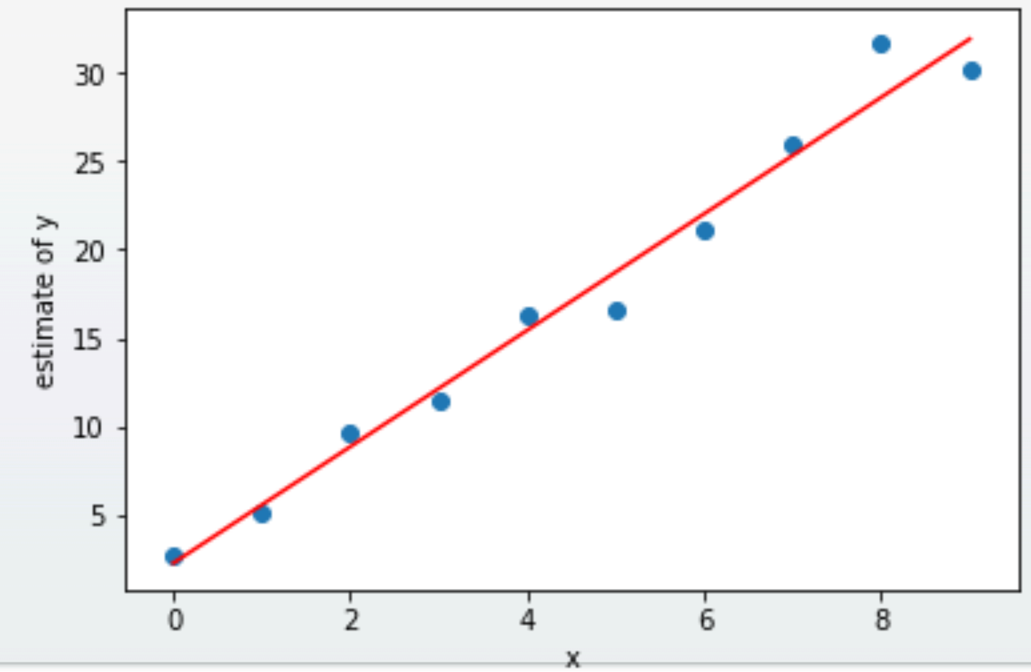
Regression is Curve-Fitting

Example: Linear Regression in Python

```
from scipy import stats
import numpy as np
import matplotlib.pyplot as plt

x = np.arange(10)
y = 3*x+4
y =  y + np.random.normal(0,2,10)
slope, intercept, r_value, p_value, std_err = stats.linregress(x,y)
y_hat = intercept + slope * x

fig = plt.figure()
plt.plot(x,y_hat, color='r')
plt.scatter(x,y)
plt.xlabel("x")
plt.ylabel("estimate of y")
#axes = plt.gca()
#axes.set_xlim([-1, 4])
```



Notes:

- data generated with a known (linear) model + noise
- in typical application, we are given the data without any model
- and need to pick a model use for the fit

Regression from Data

under-fitting

desired behavior

over-fitting

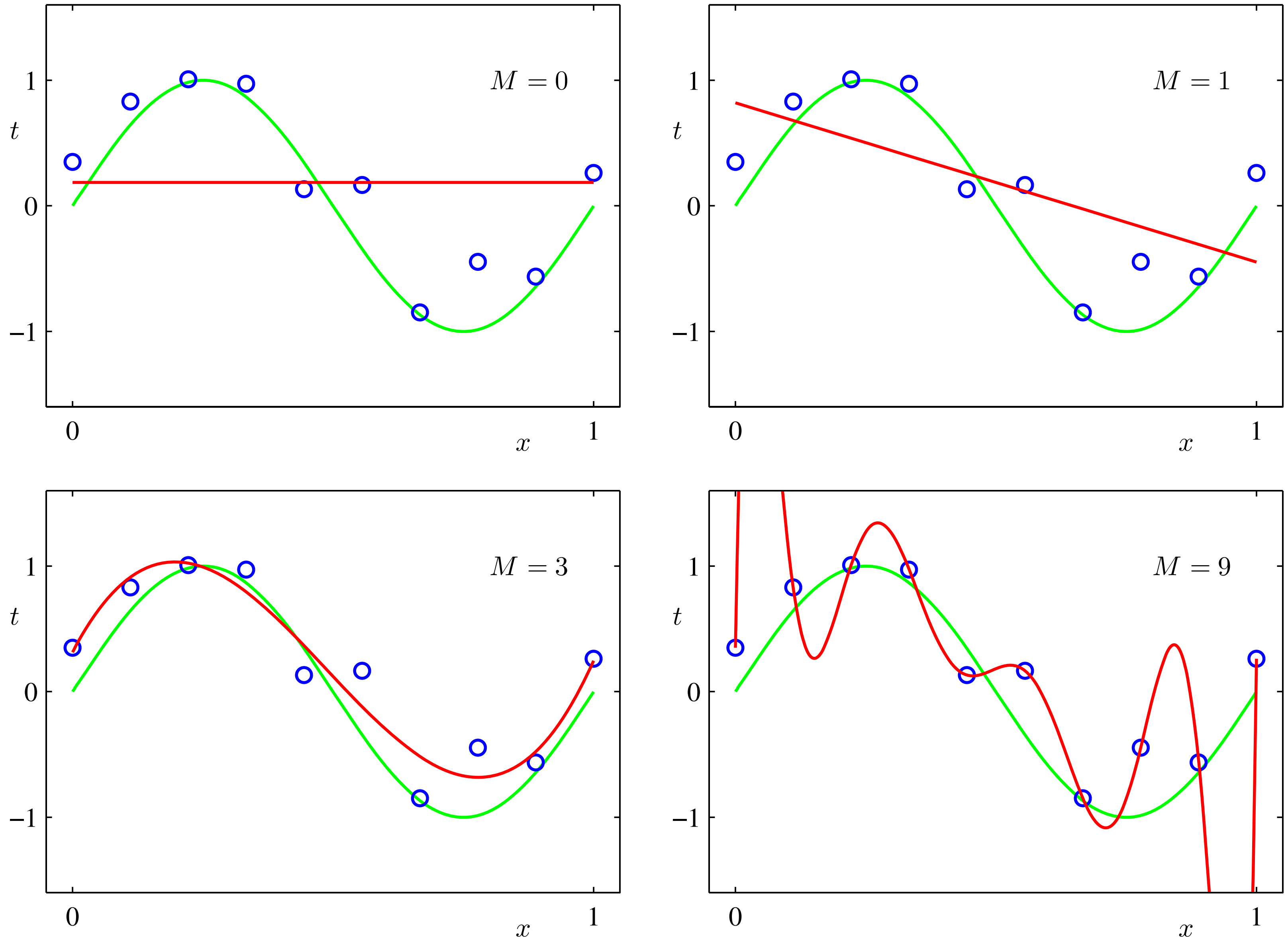
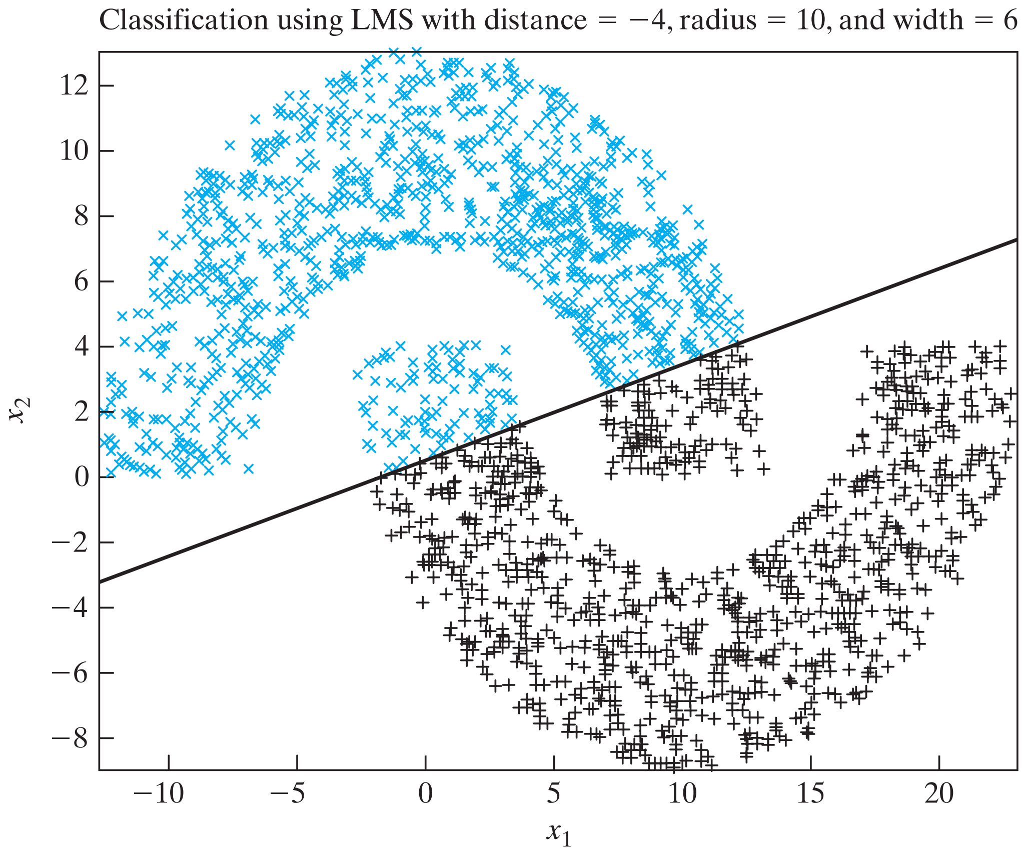


Figure 1.4 Plots of polynomials having various orders M, shown as red curves, fitted to the data set shown in Figure 1.2.

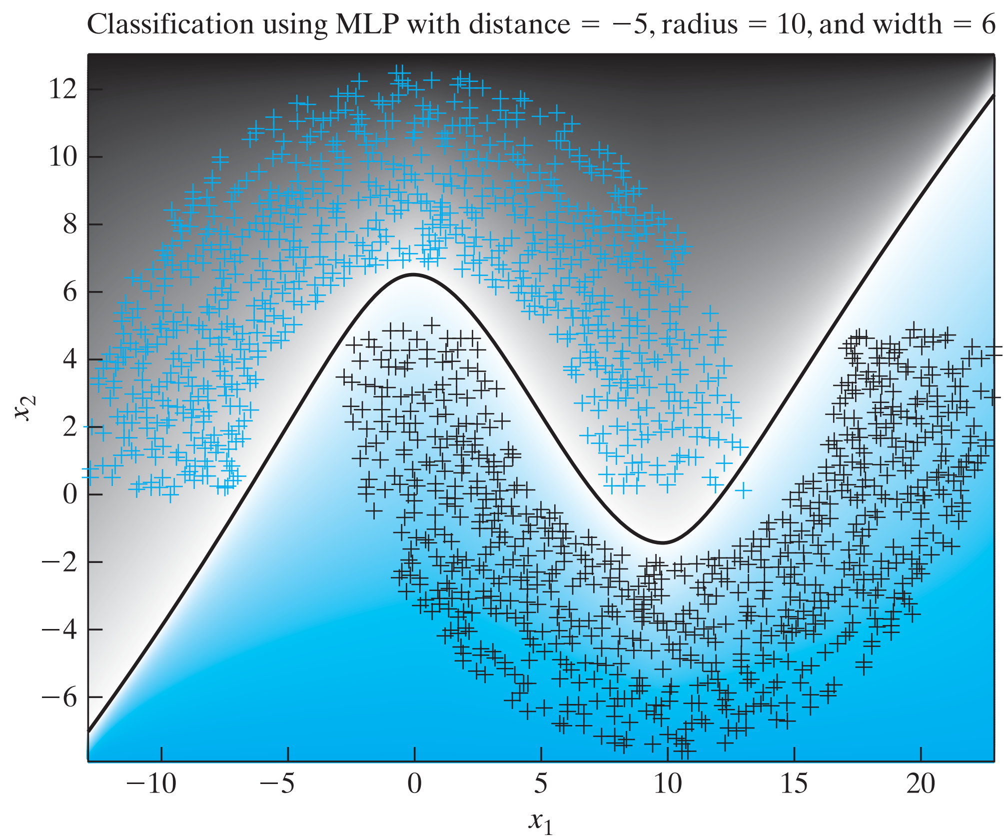
Choosing the right model (complexity) is challenging given a finite data set and no good model for what generated it!!!

[Bishop]

Classification from Data



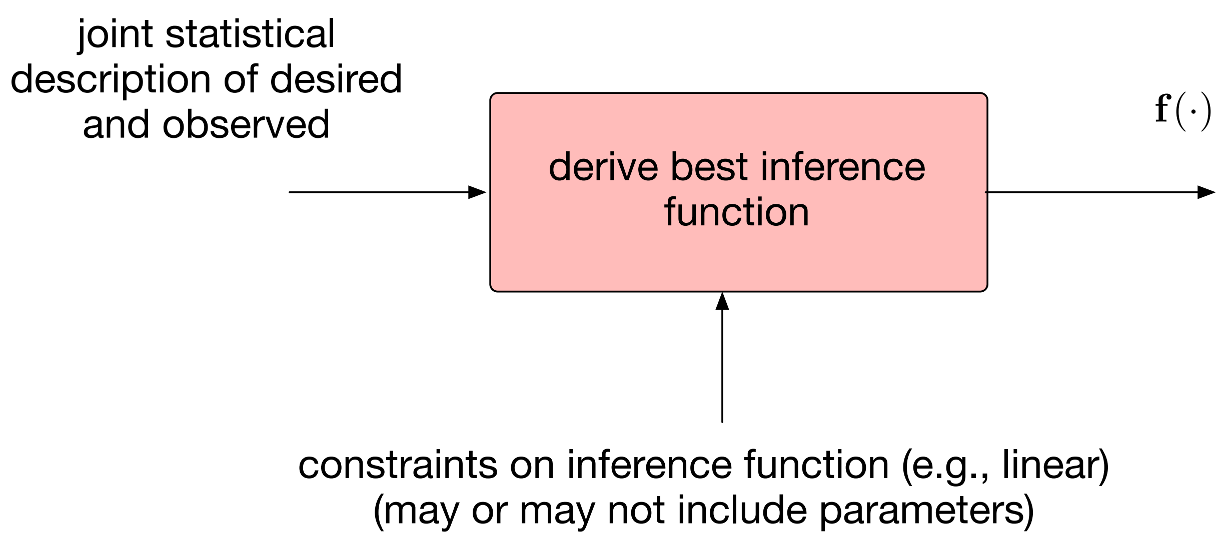
binary classification using a linear constraint on **g**(.)



binary classification using a nonlinear constraint on **g**(.)

[Haykin-NN]

Steepest (Gradient) Descent



Example: linear constraint and Mean-square Error optimality criterion

$$E(\mathbf{W}) = \mathbb{E}\left\{\|\mathbf{d}(u) - \mathbf{W}\mathbf{z}(u)\|^2\right\}$$

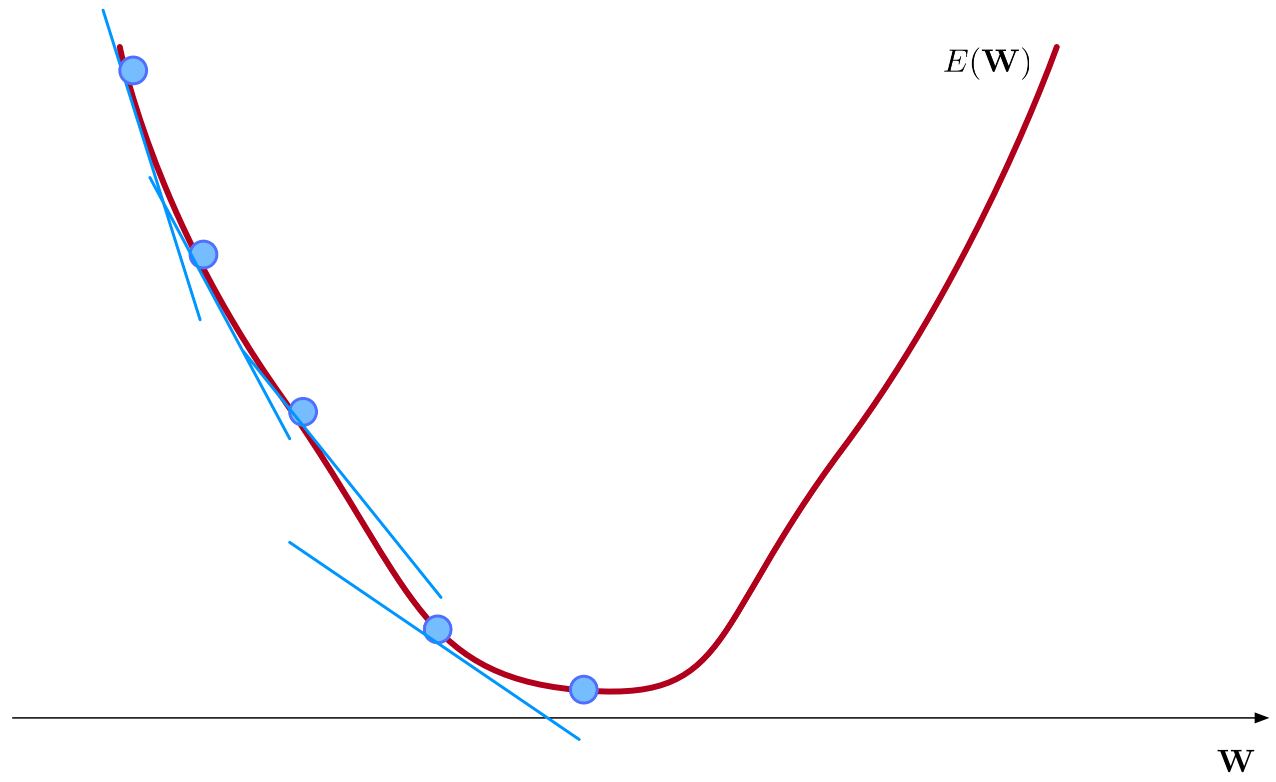
$$\widehat{\mathbf{W}}_{\text{opt}} = \arg\min_{\mathbf{W}} \quad E(\mathbf{W})$$

Steepest Descent: solve this iteratively using a first-order expansion around current bets value of **W**

$$\widehat{\mathbf{W}}_{n+1} = \widehat{\mathbf{W}}_n - \eta \nabla_{\mathbf{W}} E(\widehat{\mathbf{W}}_n)$$

Steepest (Gradient) Descent

$$\widehat{\mathbf{W}}_{n+1} = \widehat{\mathbf{W}}_n - \eta \nabla_{\mathbf{W}} E(\widehat{\mathbf{W}}_n)$$



η step size or learning rate

Steepest (Gradient) Descent

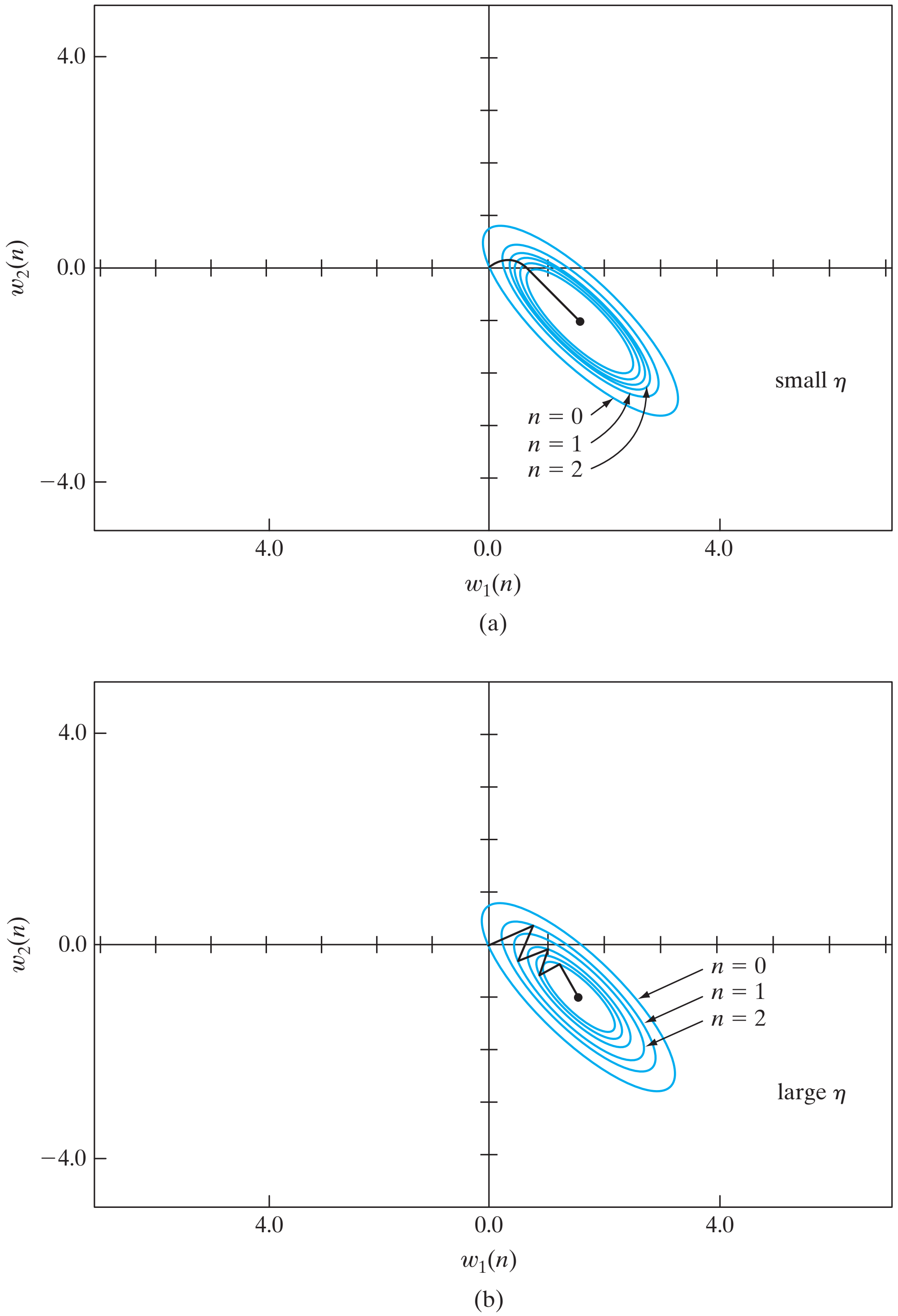
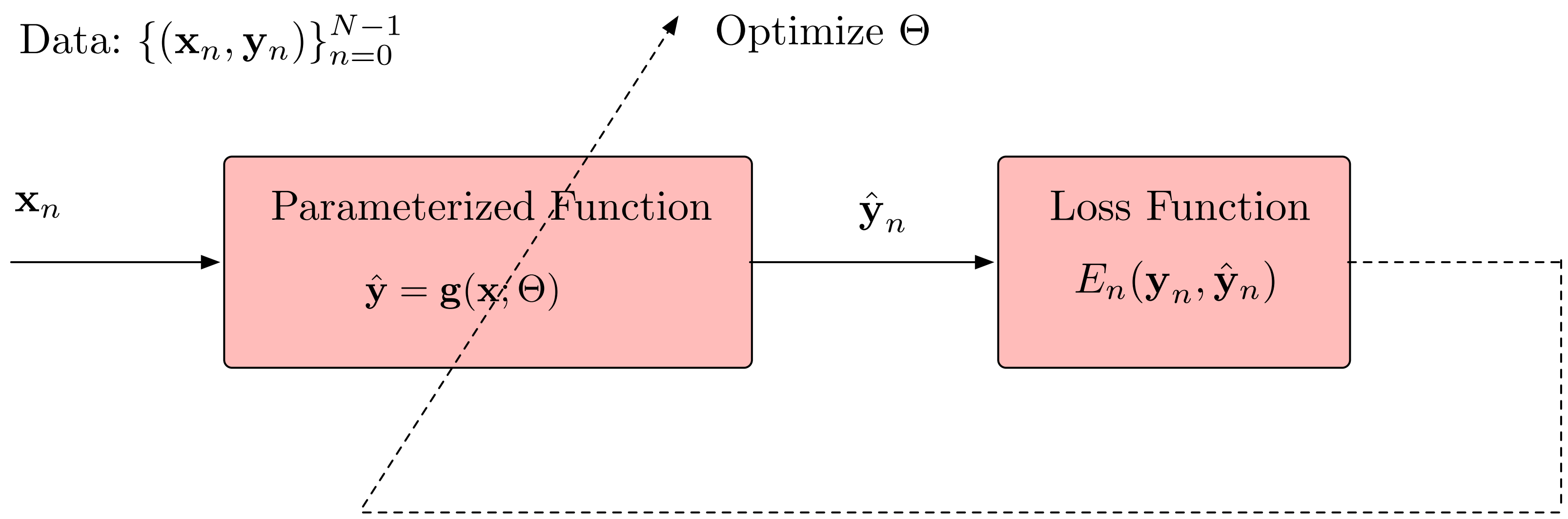


FIGURE 3.2 Trajectory of the method of steepest descent in a two-dimensional space for two different values of learning-rate parameter: (a) small η (b) large η. The coordinates w_1 and w_2 are elements of the weight vector $\mathbf{w}$; they both lie in the $\mathscr{W}$-plane.

$$\widehat{\mathbf{W}}_{n+1} = \widehat{\mathbf{W}}_n - \eta \nabla_{\mathbf{W}} E(\widehat{\mathbf{W}}_n)$$

[Haykin-NN]

Stochastic Gradient Descent (SGD)



In many cases, we do not have the exact form of the gradient or we wish to have a rough approximation of the gradient to learn from new data

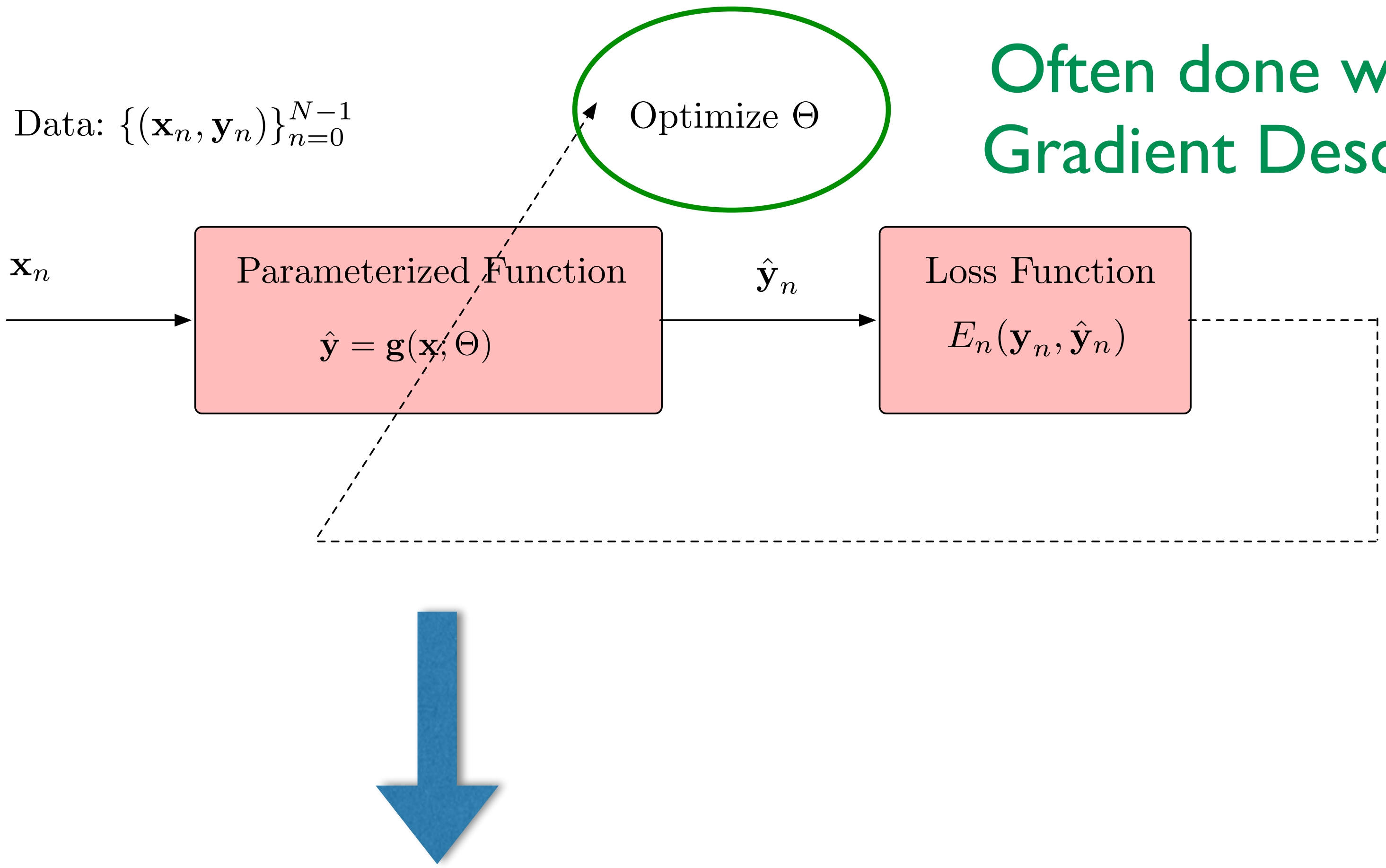
SGD: use averages over data (typically subset of data points) to approximate ensemble averaging and therefore approximate the true gradient with a noisy (stochastic) approximation

$$\widehat{\mathbf{W}}_{n+1} = \widehat{\mathbf{W}}_n - \eta \hat{\nabla}_{\mathbf{W}} E(\widehat{\mathbf{W}}_n)$$

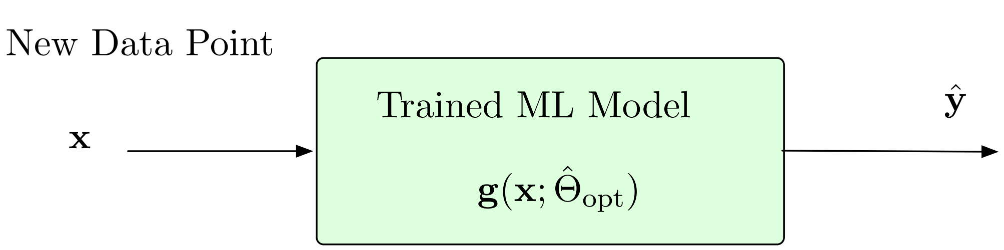
Types of ML: Supervised Learning

Training (Learning)

Often done with Stochastic Gradient Descent (learning)



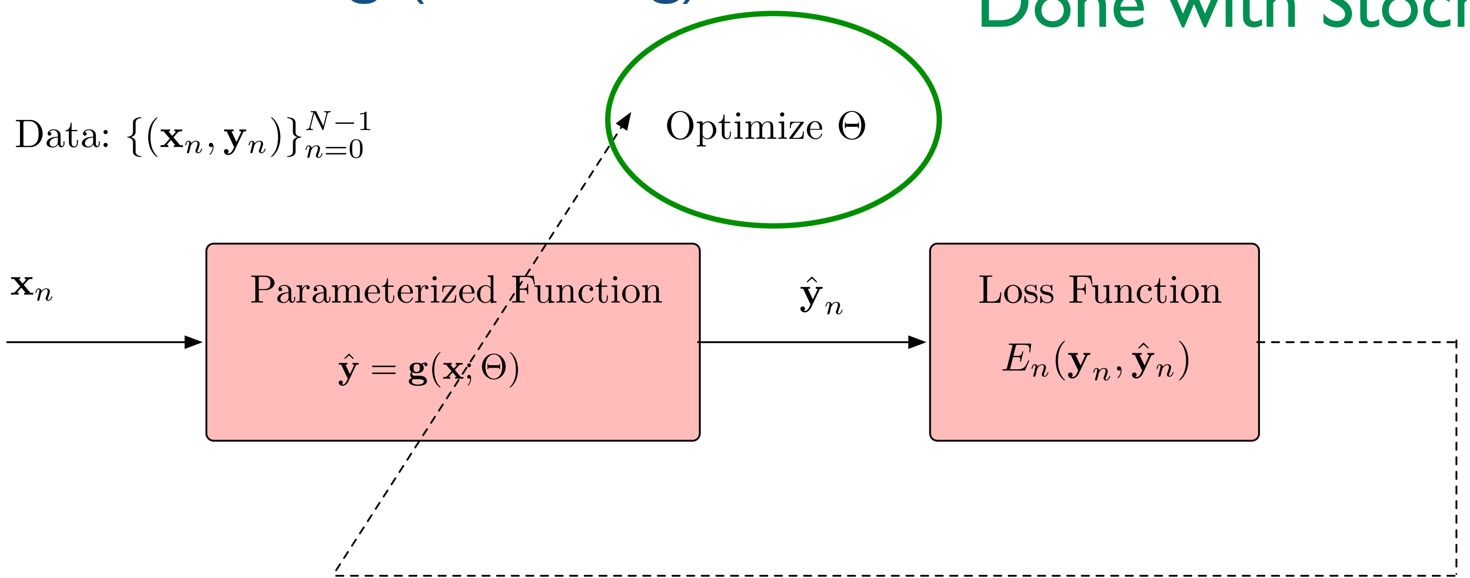
Inference Mode (after trained)



Neural Networks

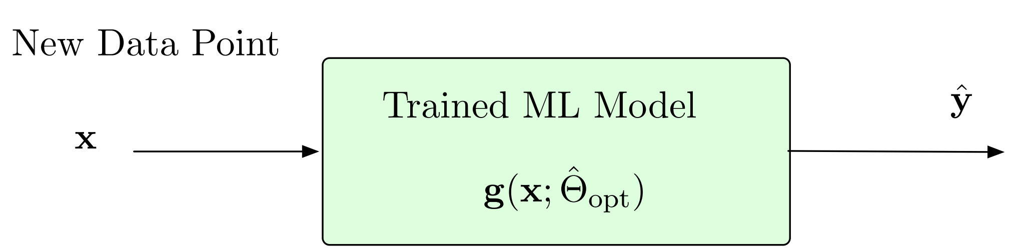
Training (Learning)

Done with Stochastic Gradient Descent (learning)
Back-propagation



Neural nets just define a very rich "hypothesis set"

Inference Mode (after trained)



$$\mathcal{G} = \{\mathbf{g}(\cdot; \Theta) : \forall \Theta \in \mathbb{R}^D\}$$

"hypothesis set"
(class of possible inference functions)

Types of Neural Networks

Feedforward Nnets (Multilayer Perceptrons)

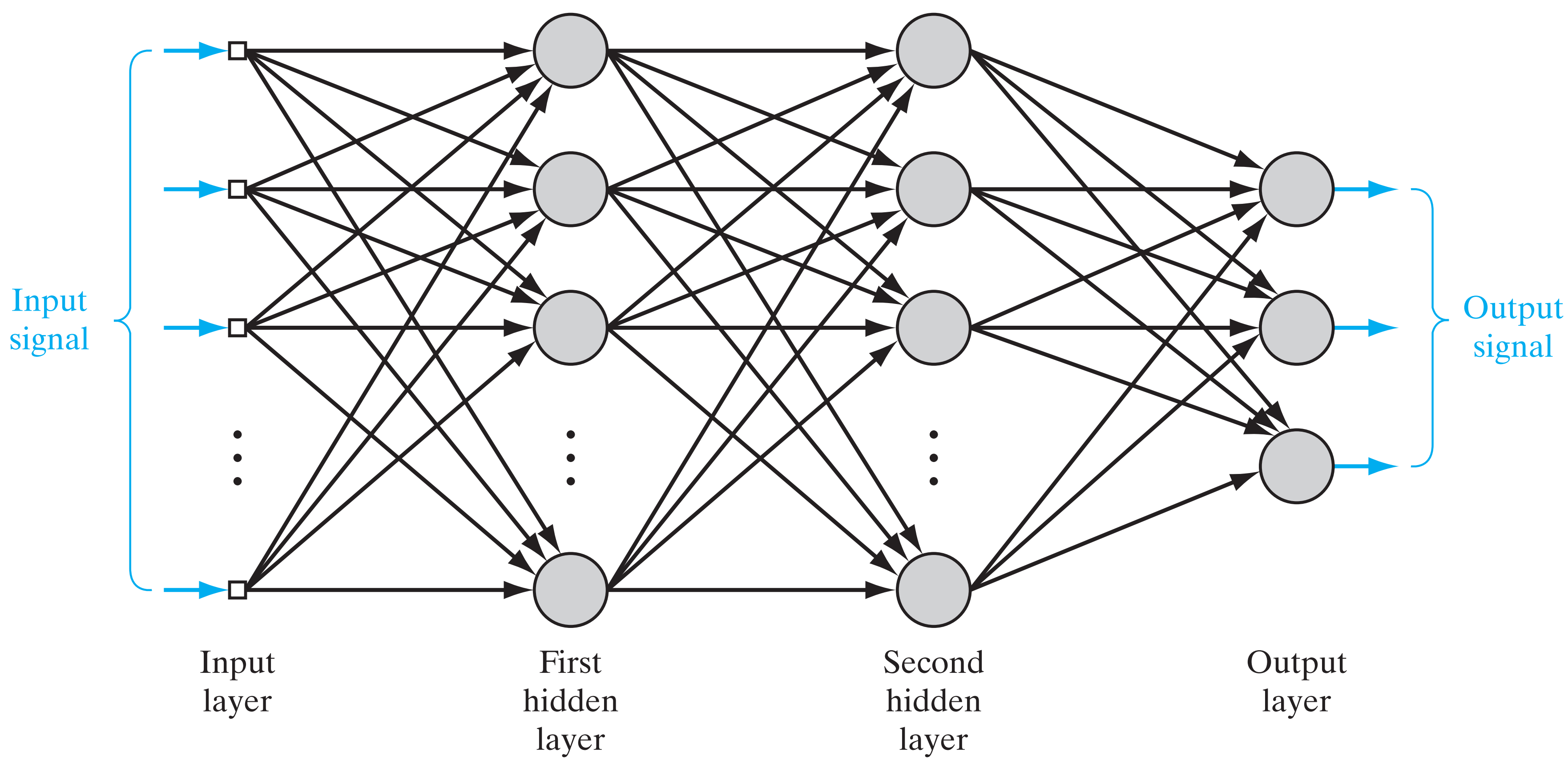


FIGURE 4.1 Architectural graph of a multilayer perceptron with two hidden layers.

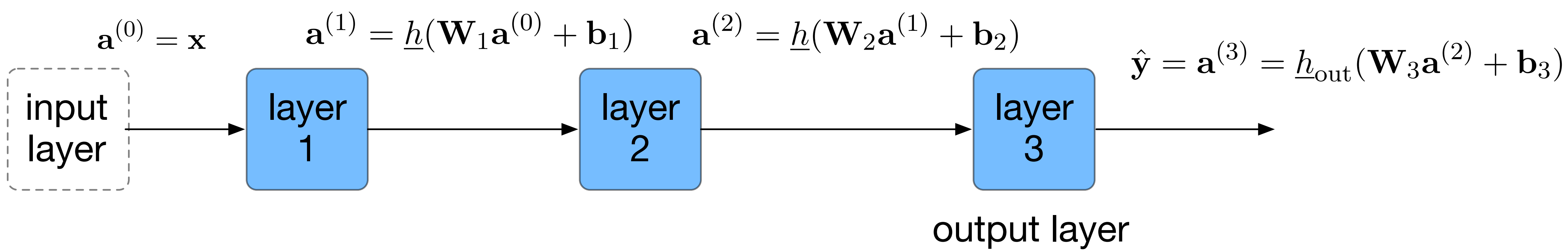
"Deep" means more than one hidden layer

think of this as a generalization of a feedforward (MA of FIR) filter

[Haykin-NN]

Types of Neural Networks

Feedforward Nnets (Multilayer Perceptrons)

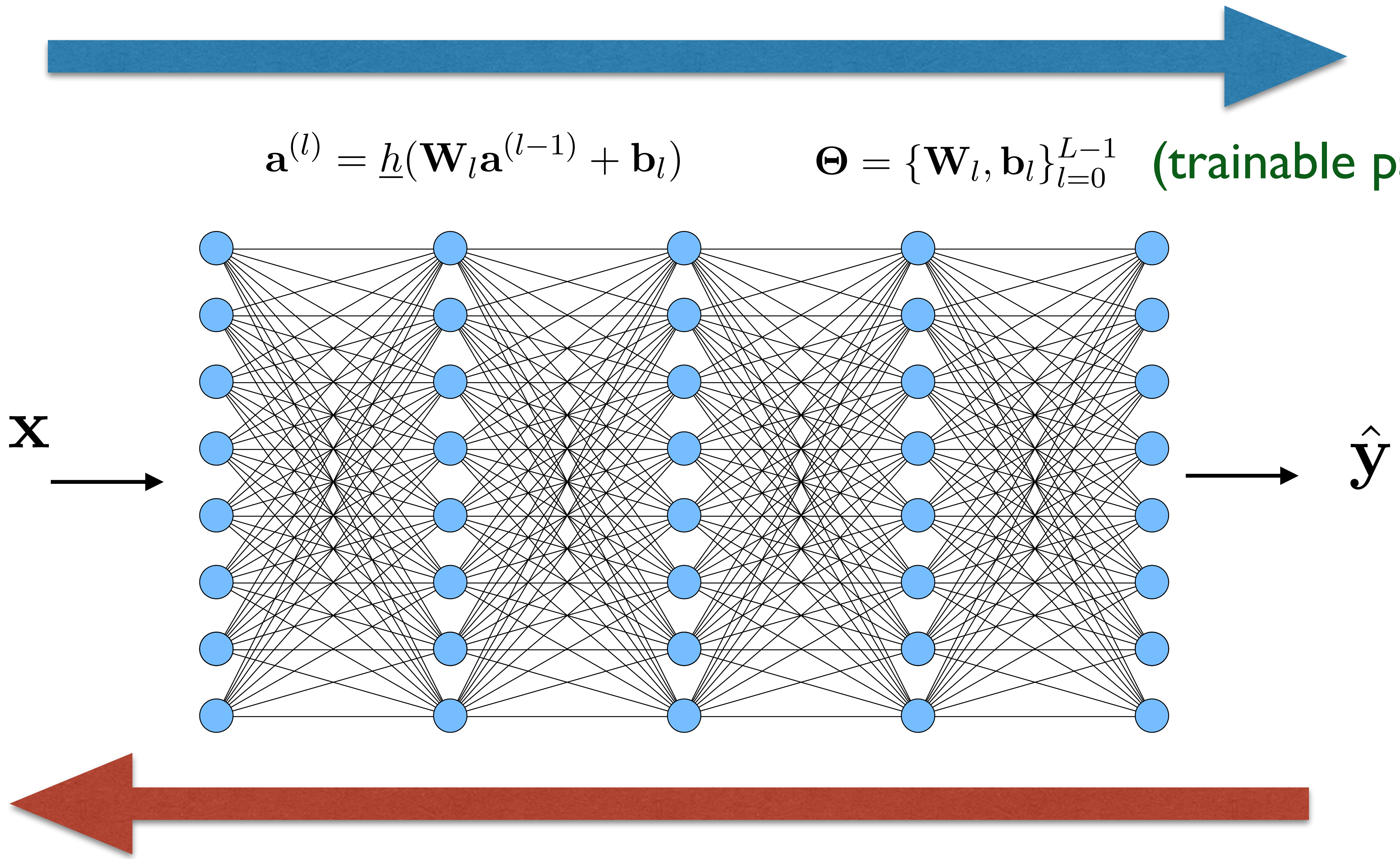


matrix-vector view of the previous diagram

MLPs

Forward propagation (inference and training)

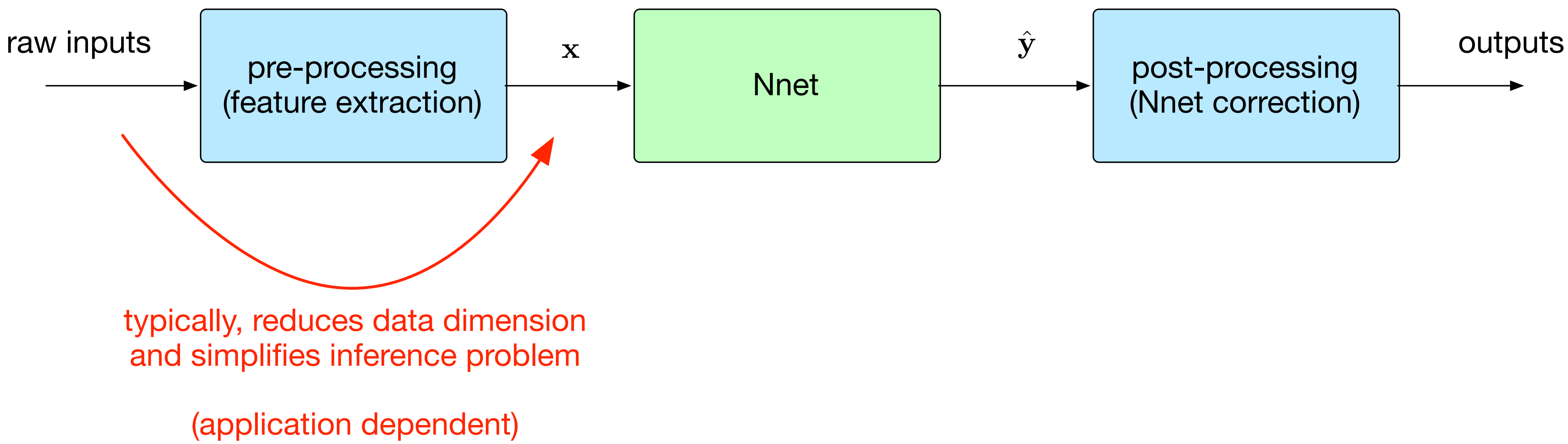
$\mathbf{a}^{(l)} = \underline{h}(\mathbf{W}_l \mathbf{a}^{(l-1)} + \mathbf{b}_l)$ $\boldsymbol{\Theta} = \{\mathbf{W}_l, \mathbf{b}_l\}_{l=0}^{L-1}$ (trainable parameters)



Backward propagation (training)

Learn the trainable parameters using SGD and the chain-rule

Typical System using Nnet

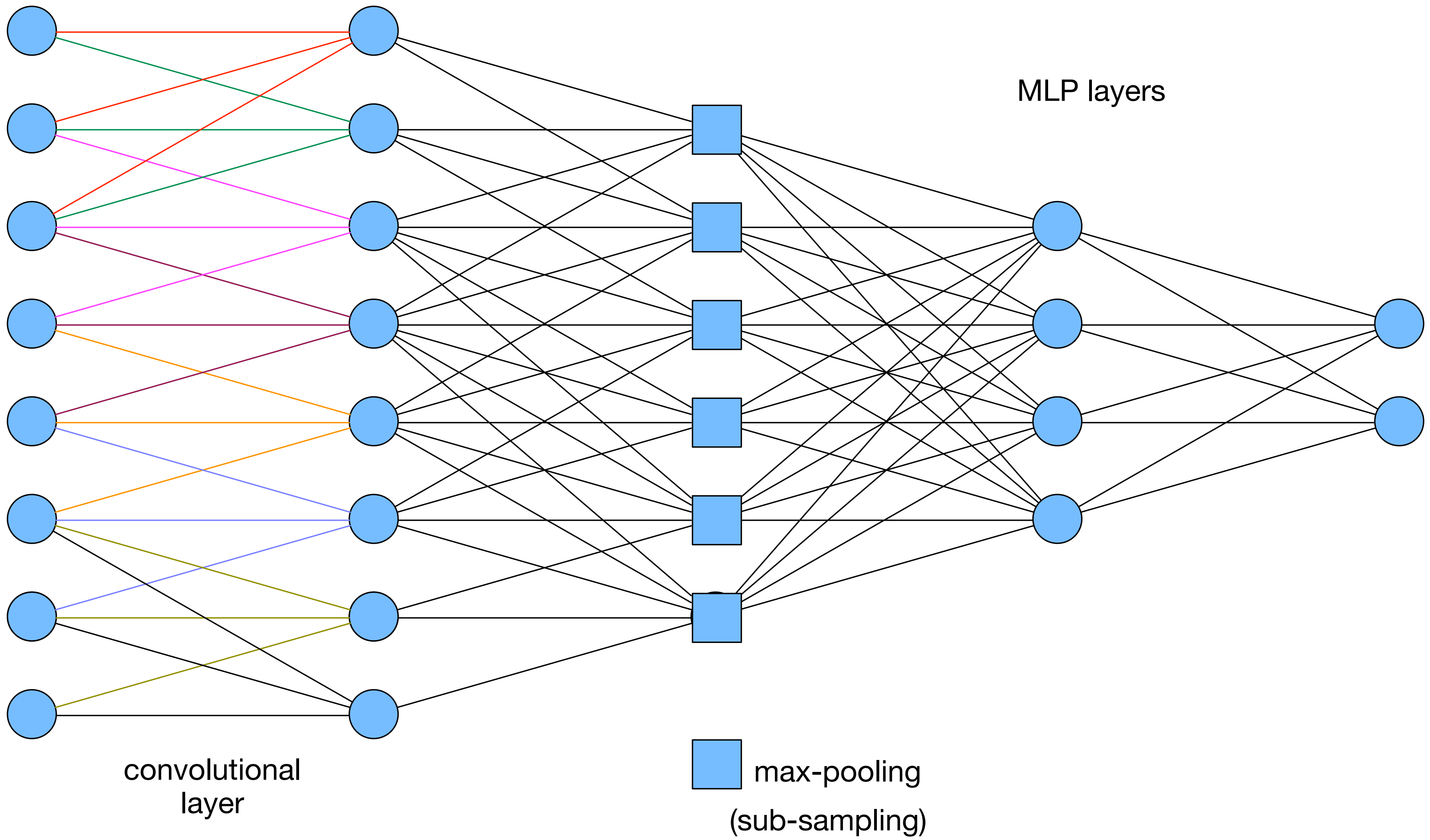


feature computation takes into account statistical properties and perceptual properties of data and overall system

in the ideal case where a statistical model is known, the ideal features are sufficient statistics for the inference problem

Types of Neural Networks

Convolutional Nnets



May be viewed as performing feature extraction before the MLP layers
(this feature extraction is learned)

Types of Neural Networks

Recurrent Neural Networks

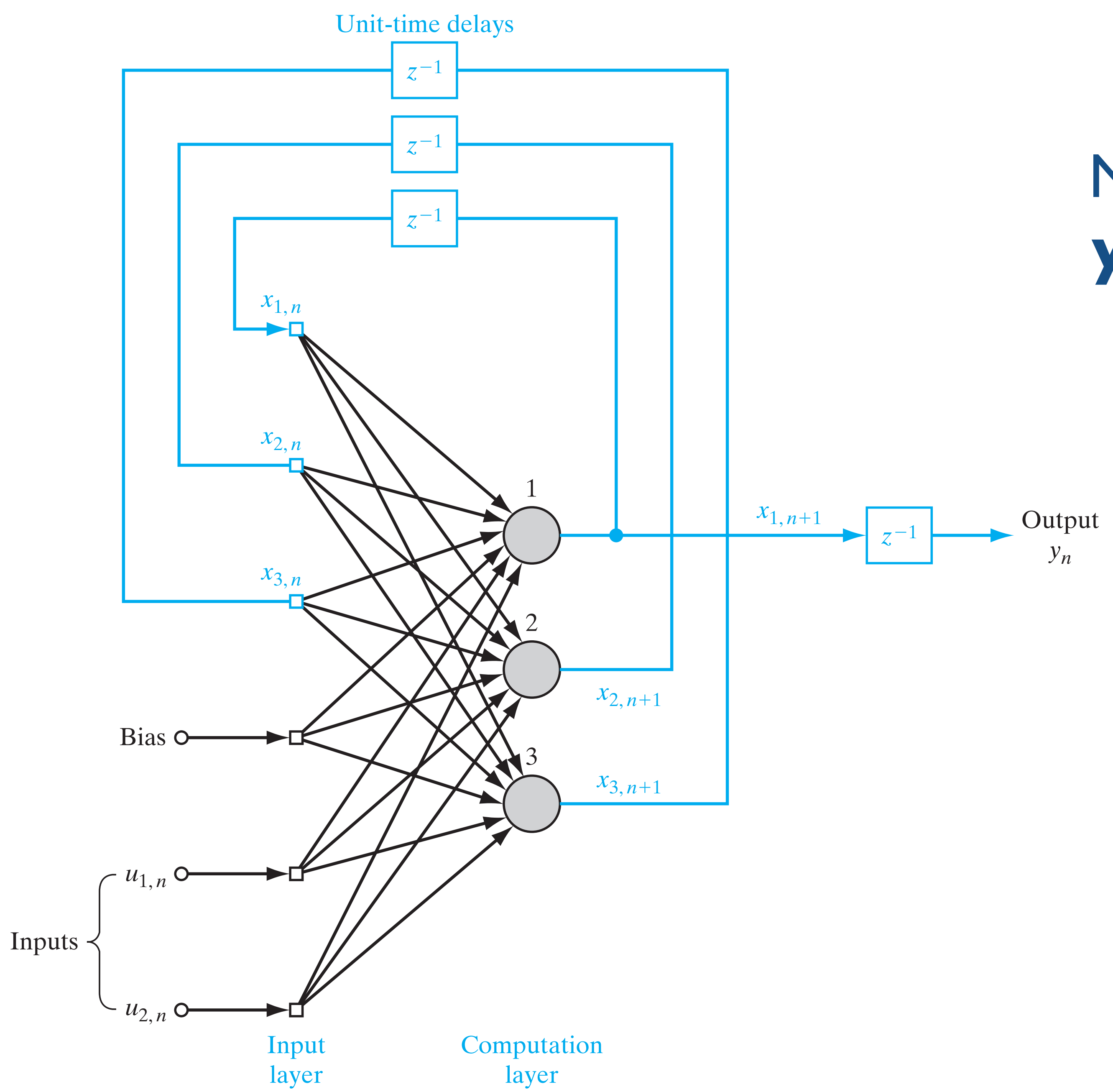


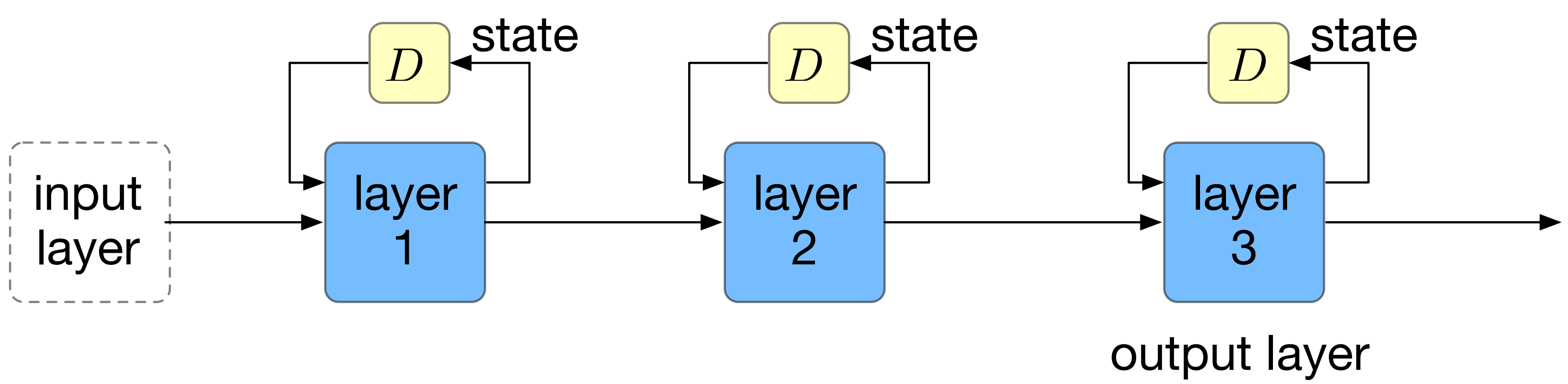
FIGURE 15.6 Fully connected recurrent network with two inputs, two hidden neurons, and one output neuron. The feedback connections are shown in red to emphasize their global role.

Network has **state** — current output **y**[n] is a function of current input **x**[n] and state (e.g., previous **x**[n-i], **y**[n-i]

think of this as a generalization of a feedback filter

Types of Neural Networks

Recurrent Neural Networks



matrix-vector view of the previous diagram
(with additional recurrent layers)

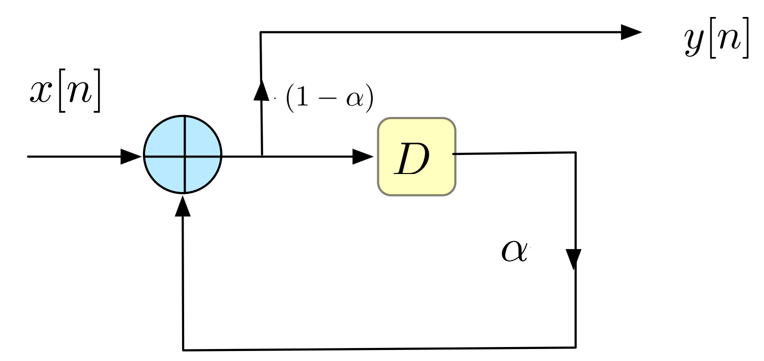
Some Connections to Simple Signal Processing

difference equation

$$y[n] = \alpha y[n-1] + \beta x[n]$$

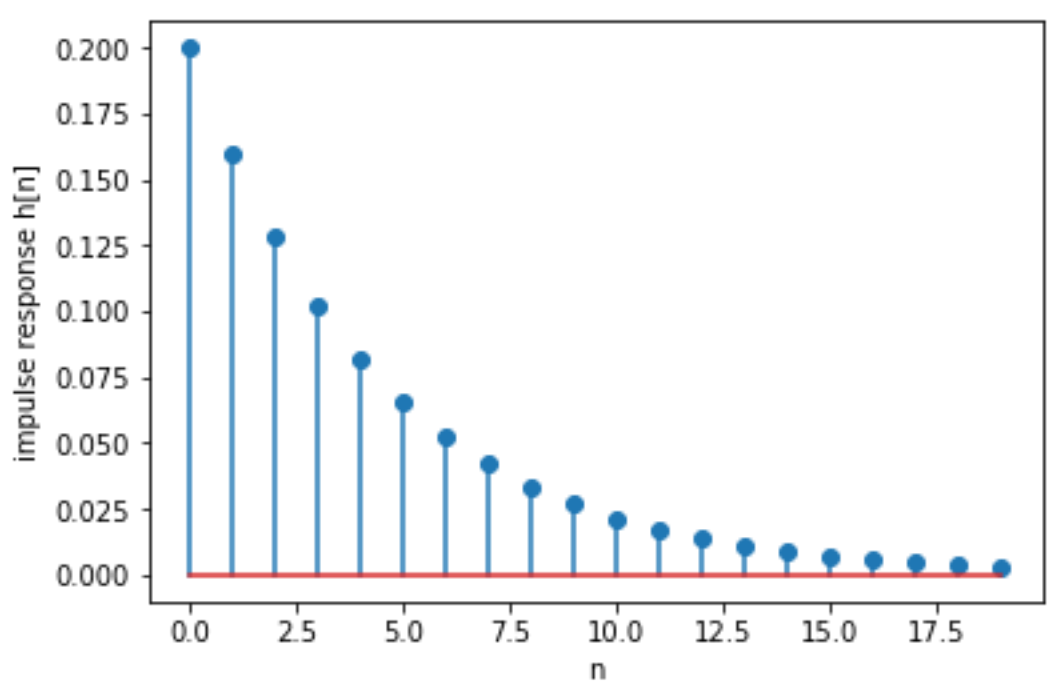
$$y[n] = \alpha y[n-1] + (1-\alpha) x[n]$$

block diagram

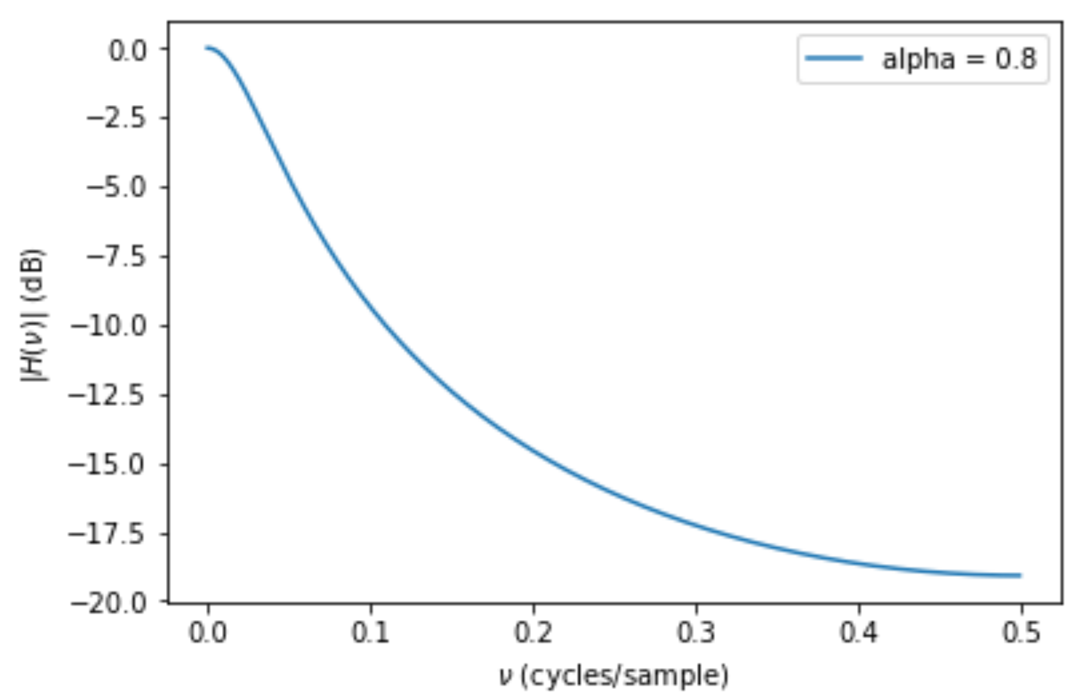


stable, low-pass filter when 0 < alpha < 1

impulse response

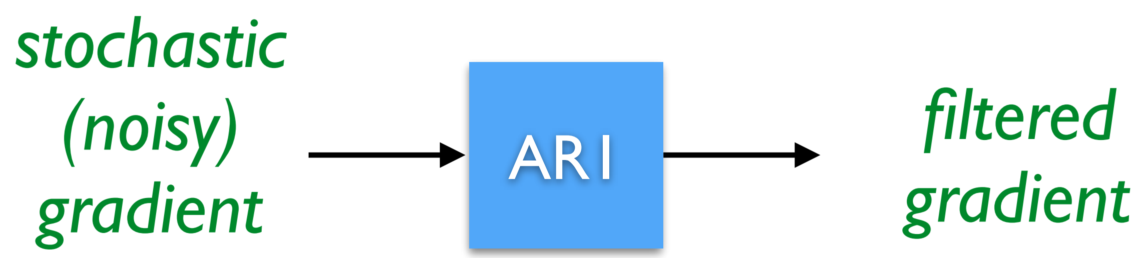


frequency response



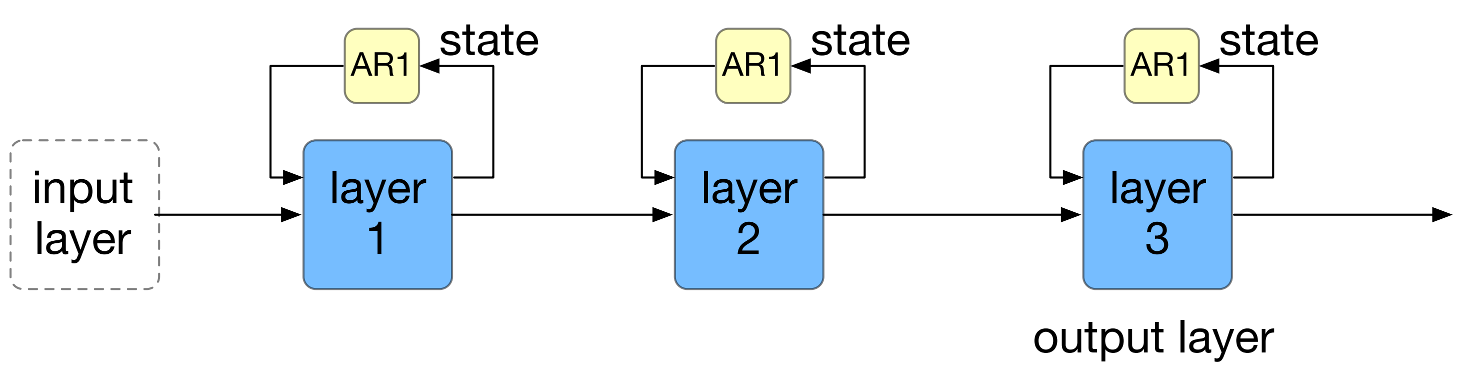
Use for AR1 Filter (preview)

variation on stochastic gradient descent



(momentum, adam optimizer, etc)

"gating" recurrent units



(GRUs, LSTMs)

When to Use Deep Learning?

- **Don't use Deep Learning when…**
  - You have a good statistical model for the inference problem and you can derive and implement the ideal inference function **f**(.)
  - A simpler form of supervised ML performs well
    - eg., Linear regression, logistical regression, Naive Bayes Classifier, etc
- **Do use Deep Learning when… there is lots of data available and…**
  - Tough to model accurately
  - You can use a classical approach, but there are many "clamps" in your algorithm (conditional statements… hacking)
  - Modeling is good, but implementing **f**(.) exactly is prohibitively complex

Model-driven: Experiments, Models, Algorithms

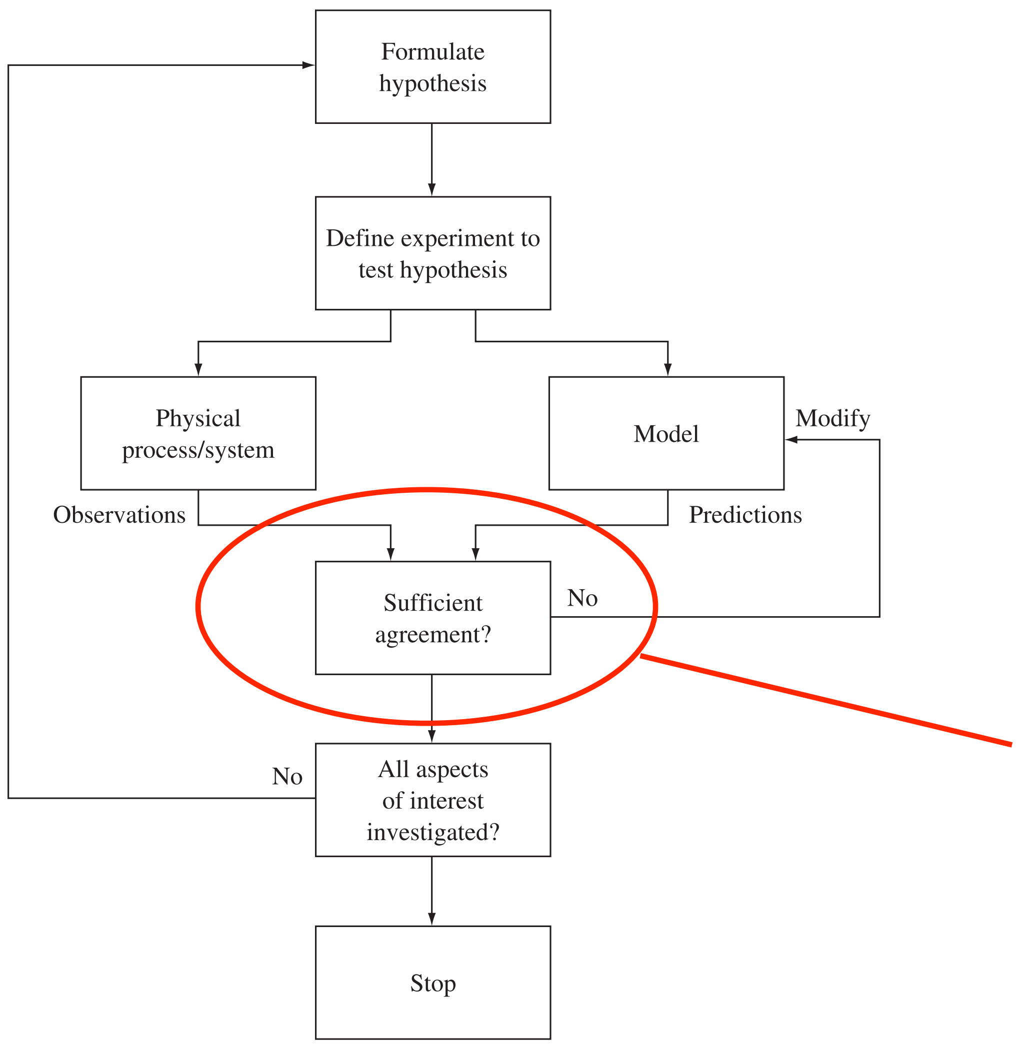


FIGURE 1.1
The modeling process.

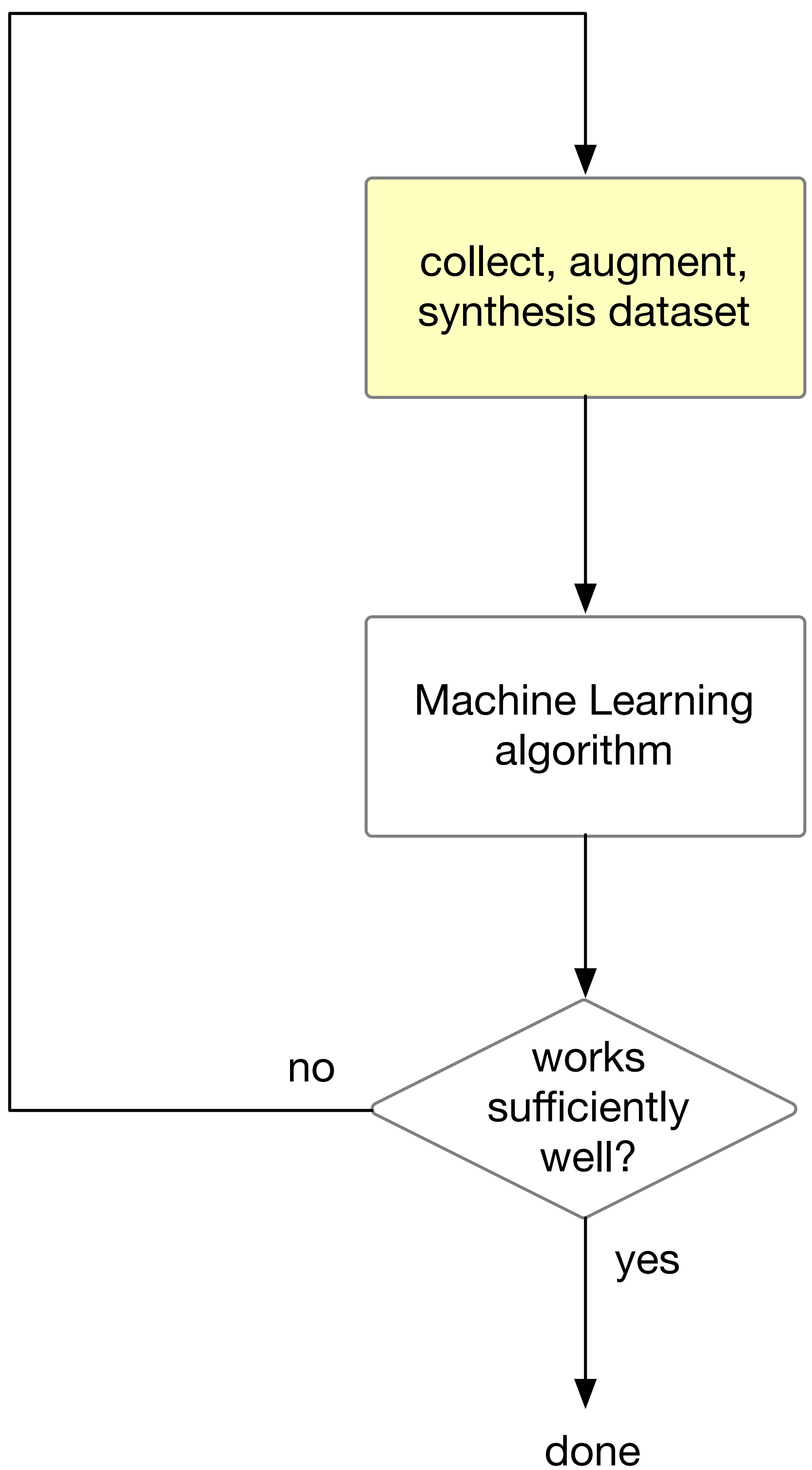
[Leon-Garcia, Probability Statistics, and Random Processes for Engineers]

there is no purely "model based" approach to any engineering problem

"All models are wrong, but some are useful"

George Box (paraphrased)

Data Driven Version of this Design Process



data (and ML) evaluated via end-to-end performance

much of the attention is here, but in practice, more iteration/time spent on data engineering

"all data are wrong, but some are useful"