Tutorial on Machine Learning for Advanced Electronics

Maxim Raginsky

March 2017

Part I
(Some) Theory and Principles

Machine Learning:

- "estimation of dependencies from empirical data" (V. Vapnik)
- enabling a computer to perform well on a given task without explicitly programming the task
- improving performance on a task based on experience

Statistical Learning Paradigm

$$\underbrace{Y}_{\text{output}} = \underbrace{f}\left(\underbrace{X}_{\text{input}}\right)$$

- Experience comes in the form of *training data* $(X_1, Y_1), (X_2, Y_2), \ldots, (X_n, Y_n)$ sampled at random from the phenomenon of interest — unknown distribution $p(x, y)$
- Performance is measured by expected accuracy of inference (prediction on previously unseen instances):

$$\begin{aligned} R(f) &= \mathbf{E}[\ell(f(X), Y)] \\ &= \int \underbrace{\ell(f(x), y)}_{\text{prediction error}} \cdot \underbrace{p(x, y)}_{\text{data distribution}} \, \mathrm{d}x \, \mathrm{d}y \end{aligned}$$

- Improvement comes from seeing more data before settling on a hypothesis

Goals of Learning

$$\underbrace{(X_1, Y_1), \ldots, (X_n, Y_n)}_{\text{training data}} \xrightarrow{\text{algorithm}} \underbrace{\widehat{f}_n}_{\text{hypothesis}}$$

- Predictive consistency

$$R(\widehat{f}_n) \xrightarrow{n\to\infty} \min_{f\in\mathcal{F}} R(f)$$

  – eventually, learn to predict as well as the best predictor you could find *with full knowledge of* $p(x, y)$

- Generalization

$$\frac{1}{n}\sum_{i=1}^{n} \ell(\widehat{f}_n(X_i), Y_i) \approx R(\widehat{f}_n)$$

  – performance of $\widehat{f}_n$ on the training data is a good predictor of its performance on previously unseen instances

- Efficiency — minimize resource usage (data, computation time, memory, ...)

Learning and Inference Pipeline

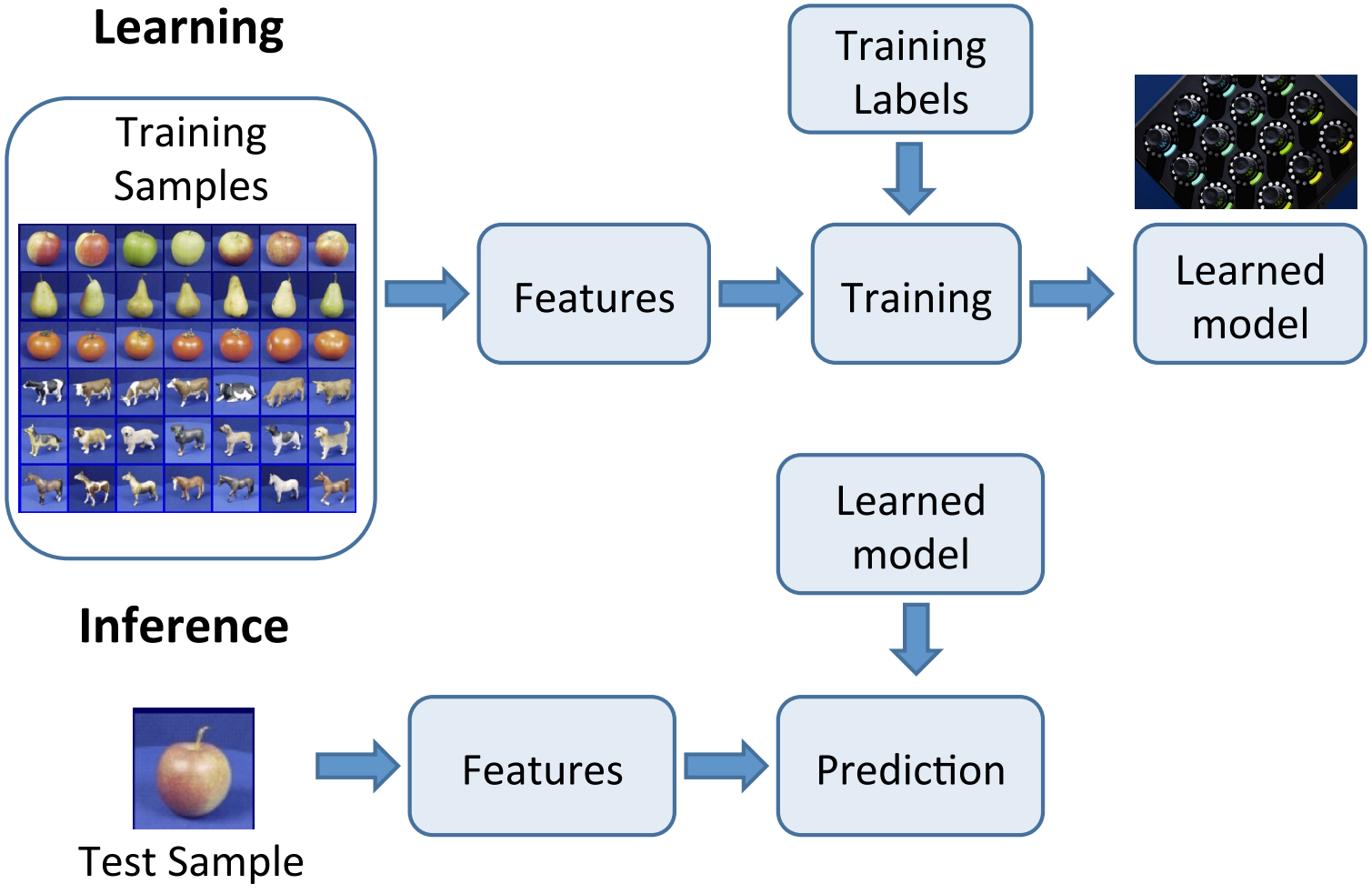


Image credit: Svetlana Lazebnik

Data Requirements

- Predictive consistency: for given $\varepsilon, \delta > 0$, what is the smallest n, such that

$$R(\widehat{f}_n) - \min_{f \in \mathcal{F}} R(f) \leq \varepsilon$$

  with probability $\geq 1 - \delta$?

- Generalization: for given $\varepsilon, \delta > 0$, what is the smallest n, such that

$$\left| \frac{1}{n} \sum_{i=1}^{n} \ell(\widehat{f}_n(X_i), Y_i) - R(\widehat{f}_n) \right| \leq \varepsilon$$

  with probability $\geq 1 - \delta$?

- Efficiency — the "golden standard" is

$$n(\varepsilon, \delta) \sim \frac{1}{\varepsilon^2} \log \frac{1}{\delta} \qquad \text{from } \mathbf{P}[\text{error diff.} \geq \varepsilon] \lesssim c_1 e^{-c_2 n \varepsilon^2}$$

Underfitting and Overfitting

- Underfitting: not enough data

$$R(\widehat{f}_n) - \min_{f \in \mathcal{F}} R(f) \text{ is large}$$

- Overfitting: low training error but large testing error

$$R(\widehat{f}_n) - \frac{1}{n}\sum_{i=1}^{n} \ell(\widehat{f}_n(X_i), Y_i) \text{ is large}$$

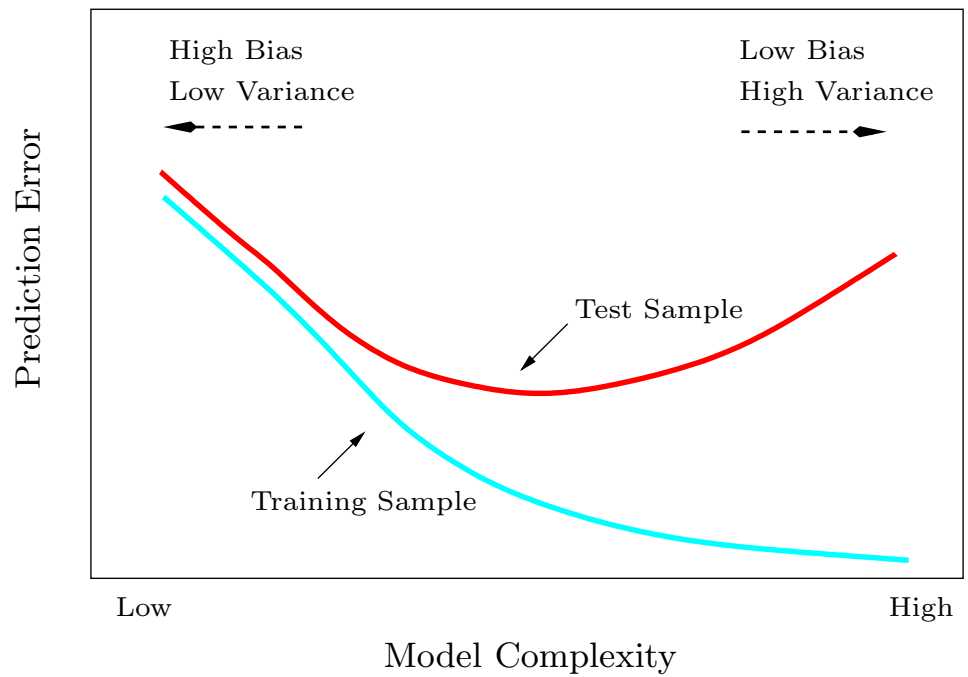


Image credit: Hastie et al., *Elements of Statistical Learning*

Generative and Discriminative Models

$$\underbrace{(X_1, Y_1), \ldots, (X_n, Y_n)}_{\text{training data}} \xrightarrow{\text{algorithm}} \underbrace{\widehat{f}_n}_{\text{hypothesis}}$$

- Recall: distribution $p(x, y)$ is unknown
- Generative modeling: use the data to *estimate* p, then find the best predictor for $\widehat{p}_n$:

$$(X_1, Y_1), \ldots, (X_n, Y_n) \longrightarrow \widehat{p}_n(x, y) \longrightarrow \widehat{f}_n$$

  - terminology: "generative model" explicitly models how input x and the output y are generated
  - advantages: $\widehat{p}_n$ can be used for other tasks besides prediction (e.g., exploring the data space)
  - disadvantages: inflated sample complexity, especially in high dimension

Generative and Discriminative Models

$$\underbrace{(X_1, Y_1), \ldots, (X_n, Y_n)}_{\text{training data}} \xrightarrow{\text{algorithm}} \underbrace{\widehat{f}_n}_{\text{hypothesis}}$$

- Recall: distribution $p(x, y)$ is unknown
- Discriminative modeling: focus on prediction only

$$(X_1, Y_1), \ldots, (X_n, Y_n) \longrightarrow \widehat{f}_n$$

  - terminology: "discriminative model" (implicitly) models the distribution of y for each fixed x, allowing to "discriminate" between different values of y
  - advantages: smaller data requirements
  - disadvantages: narrowly focused; will need retraining if the task changes

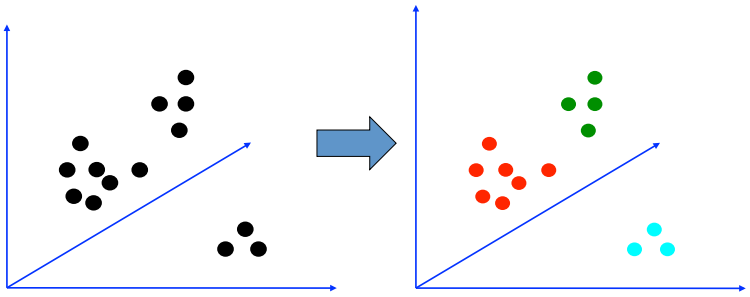
Part II
Canonical ML Tasks and Algorithms

Supervised and Unsupervised Learning

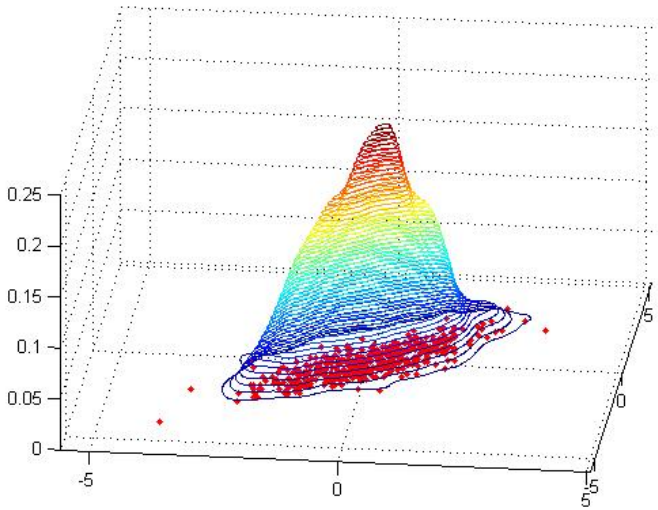
- Prediction is an example of supervised learning: training data consist of input instances matched (or labeled) with output instances to guide the learning process
- Many learning tasks are unsupervised: no labels provided, and the goal is to learn some sort of structure in the data
- Performance criteria are more vague and subjective than in supervised learning
- Unsupervised learning is often an initial step in a more complicated learning and inference pipeline

Unsupervised Learning

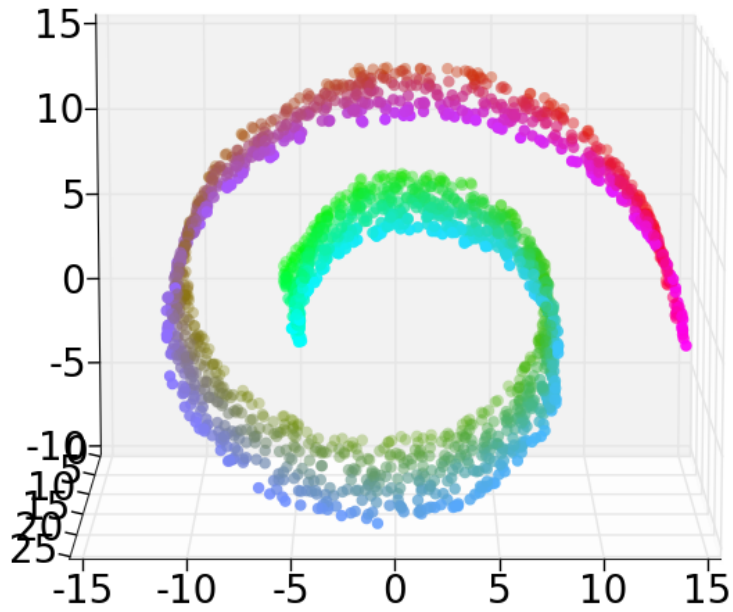
Clustering: discover groups of "similar" instances



Density estimation: approximate probability density of data



Manifold learning: discover low-dimensional structures



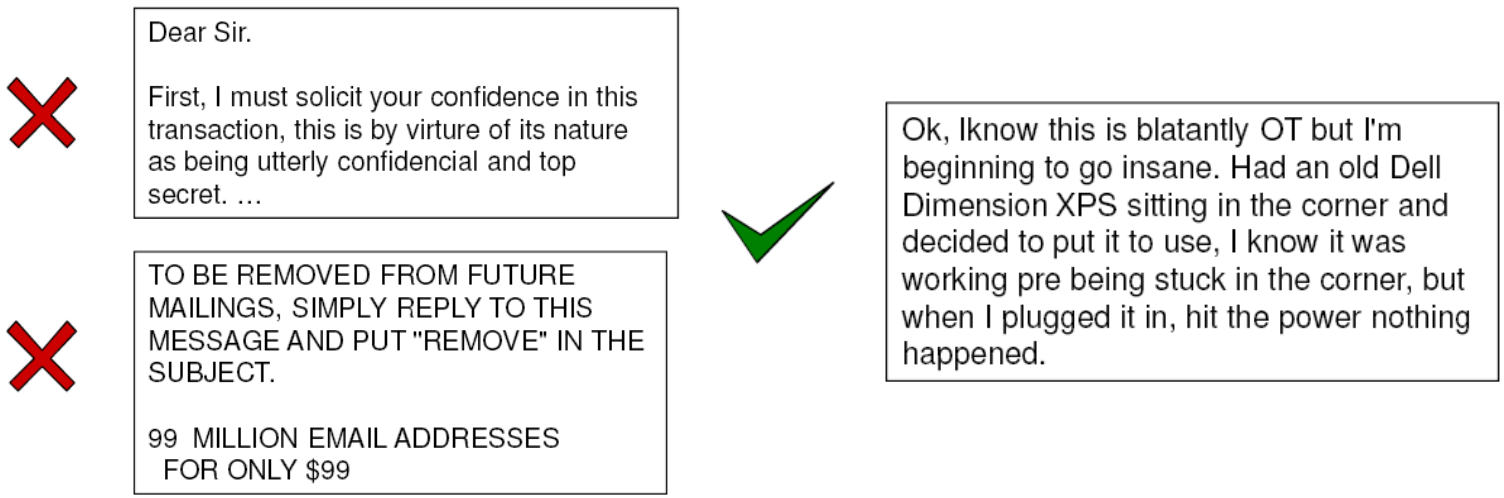
Unsupervised learning is often used at early stages of ML pipeline.

Image credits: Svetlana Lazebnik

Supervised Learning: Classification

$$(X, Y) \sim p \qquad \text{input/output}$$

- Classification: assign *attributes* X to one of M classes
- Example: email spam detection



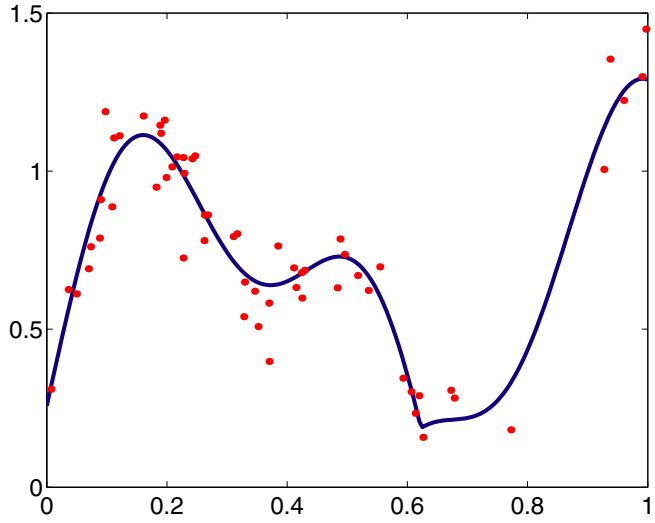
Criterion: probability of misclassification

$$R(f) = \mathbf{P}[f(X) \neq Y], \qquad \text{so that } \ell(Y, f(X)) = \begin{cases} 0, \text{if } f(X) = Y \\ 1, \text{if } f(X) \neq Y \end{cases}$$

Supervised Learning: Regression

$$(X, Y) \sim p \qquad \text{input/output}$$

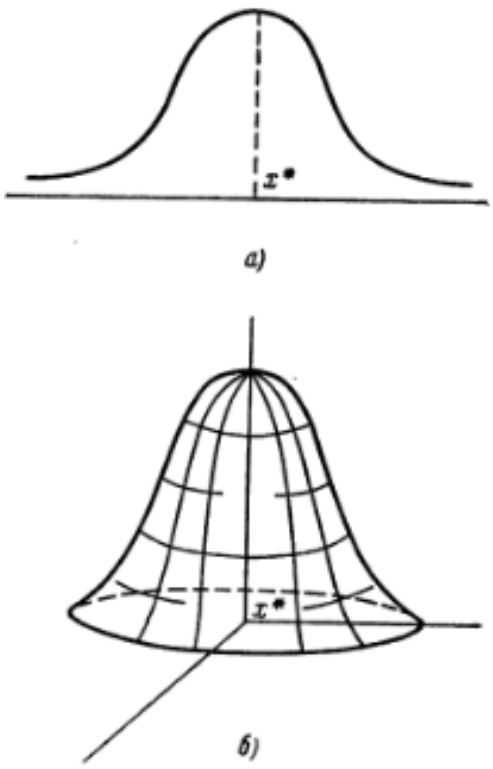
- Regression: predict a *continuous output* Y given X
- Example: curve fitting from noisy data



Criterion: mean squared error

$$R(f) = \mathbf{E}[(f(X) - Y)^2], \qquad \text{so that } \ell(Y, f(X)) = (f(X) - Y)^2$$

Example 1: Kernel Methods



$K(x - x^*)$: potential field due to "unit charge" at x^*

- Training data: $(X_1, Y_1), \ldots, (X_n, Y_n)$
- Look for $\widehat{f}_n$ of the form

$$f_{\mathbf{c}}(x) = \sum_{i=1}^{n} c_i K(x - X_i)$$

- Coefficients $c_1, \ldots, c_n$ chosen to

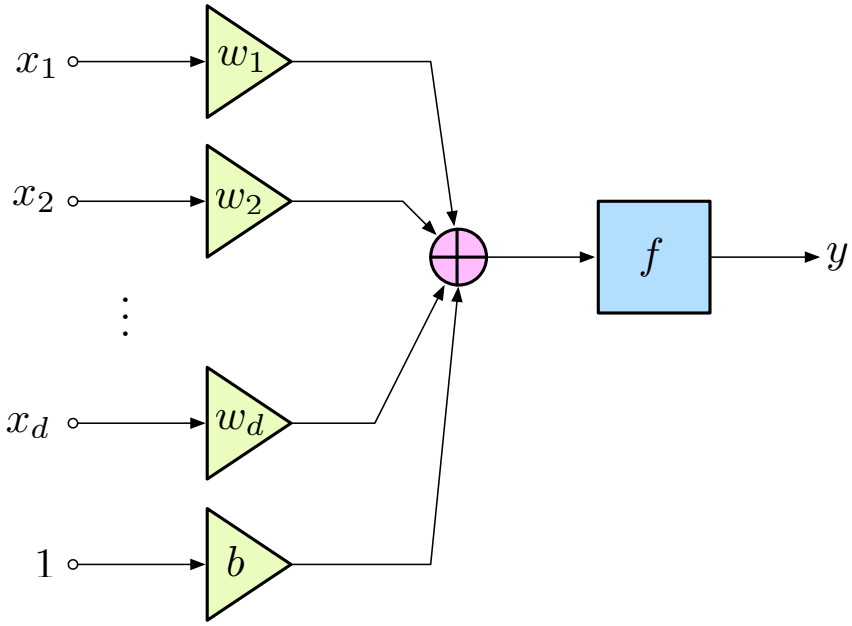
$$\text{minimize} \sum_{i=1}^{n} \left(f_{\mathbf{c}}(X_i) - Y_i\right)^2$$

(subject to additional constraints)

- Advantage: easy to optimize
- Disadvantage: need to store $\mathbf{c}$ and X_i's to do inference

Image credit: Aizerman, Braverman, Rozonoer

Example 2: Neural Networks and Deep Learning



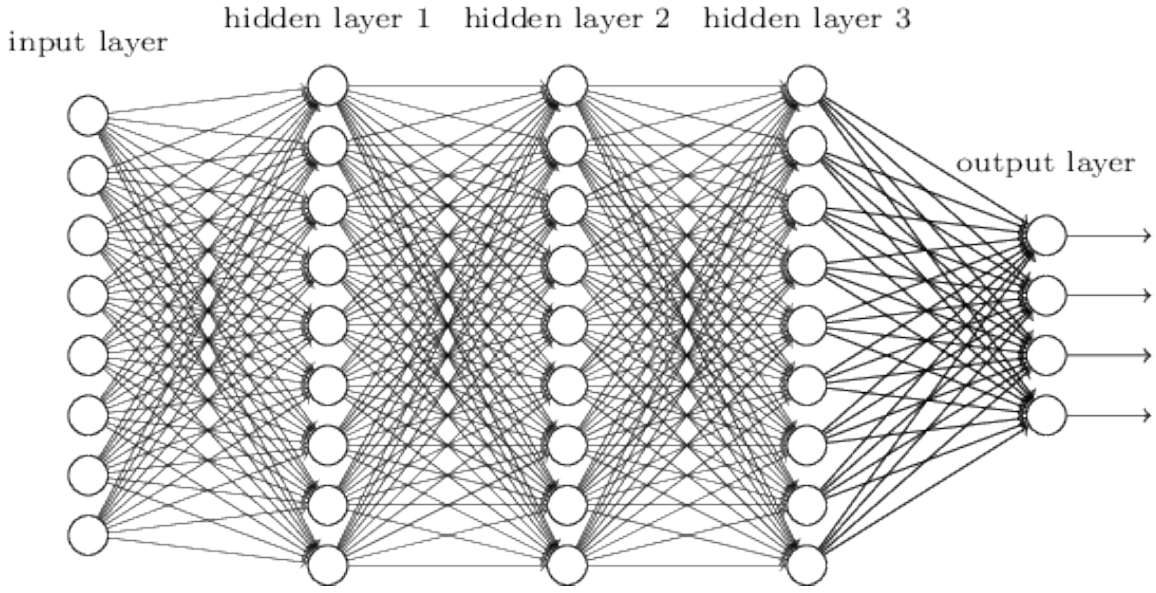
$$y = f(w_1 x_1 + \ldots + w_d x_d + b)$$

where:

- $w_1, \ldots, w_d$ – tunable gains
- b – tunable bias
- $f(\cdot)$ – activation function

- Build up complicated functions by composing simple units
- Examples of activation functions:
  - threshold, $f(z) = \begin{cases} 1, & z \geq 0 \\ 0, & z < 0 \end{cases}$
  - sigmoid, $f(z) = \frac{1}{1+e^{-z}}$
  - tanh, $f(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$
  - rectified linear unit (ReLU), $f(z) = \max(z, 0)$
- Advantage: neural networks are *universal approximators* for smooth functions $y = f(x)$

Example 2: Neural Networks and Deep Learning

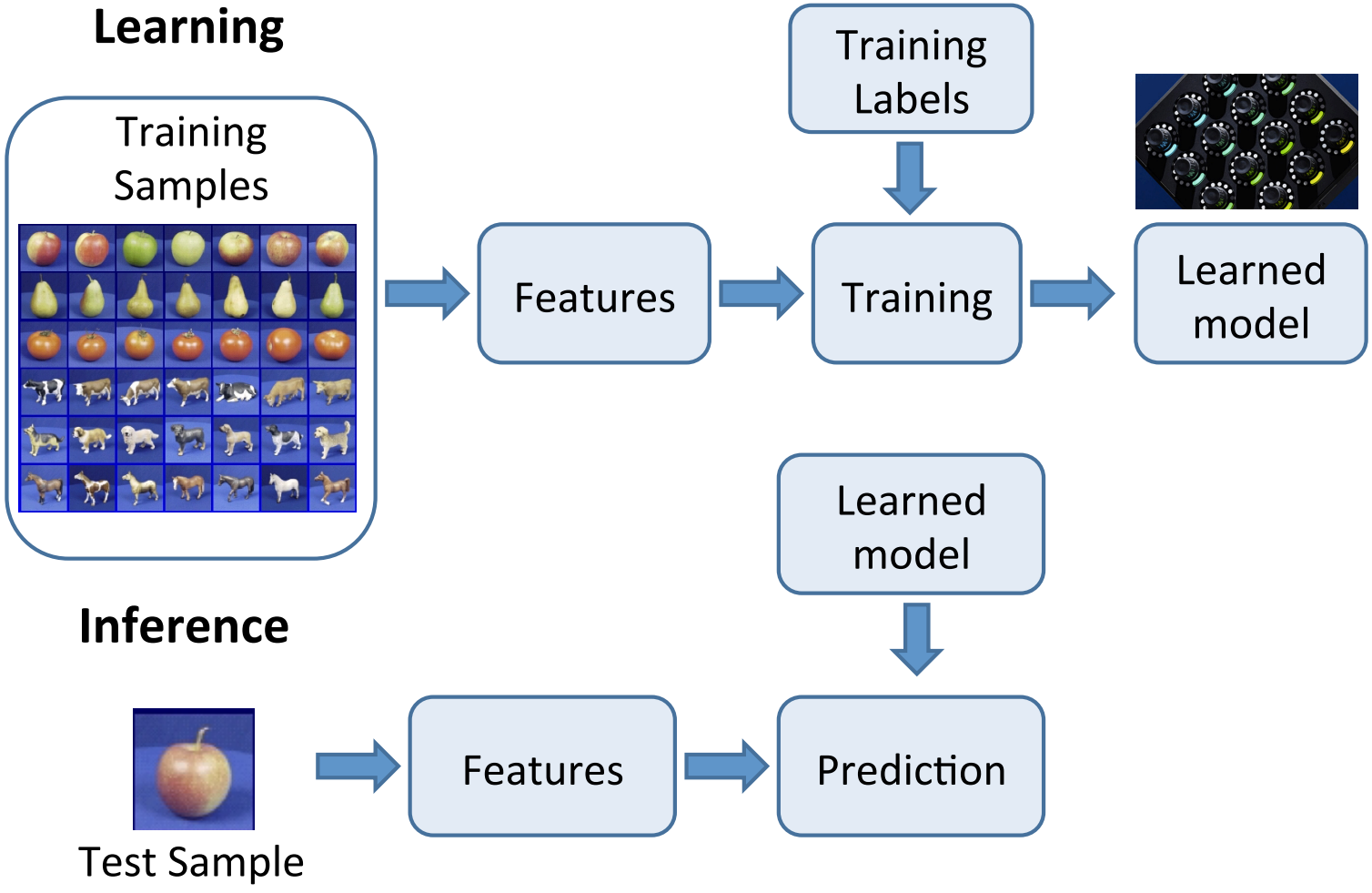


$$z_1 = \vec{f}(A_1 x + b_1),\ z_2 = \vec{f}(A_2 z_1 + b_2),\ \ldots,\ y = \vec{f}(A_k z_{k-1} + b_k)$$

- Advantages: deep networks can efficiently represent rich classes of functions; work extremely well in practice
- Disadvantages: training is computationally intensive; poorly understood theoretically

Part III
Interaction between Data and Algorithms

Recap: Learning and Inference Pipeline

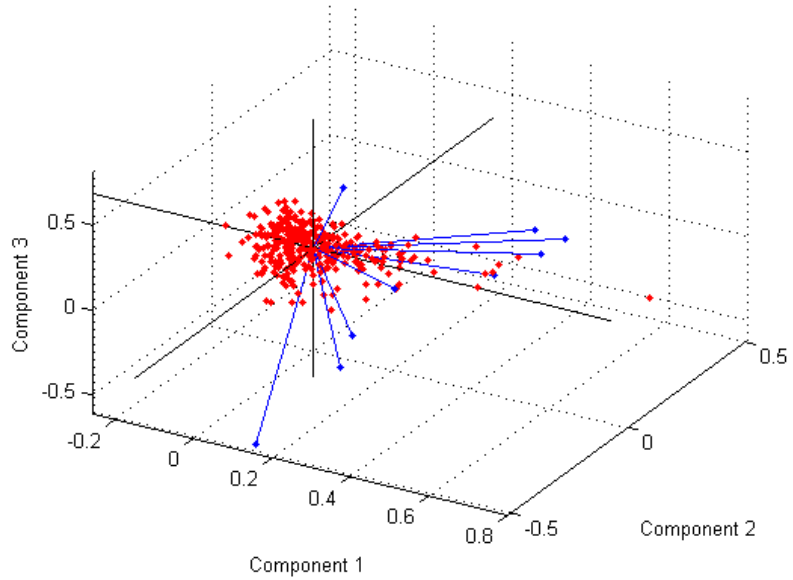


What happens before the training step?

Feature Extraction and Dimensionality Reduction

- Which features to use?
  - traditionally, this step relied on extensive expert knowledge
  - hand-engineered features tailored to the task at hand
  - deep learning aims to automate this task (think of each layer performing feature extraction)
- Dimensionality reduction
  - use unsupervised learning to discover low-dimensional structure in the data
  - for example, principal component analysis (PCA)

Principal Component Analysis (PCA)



- compute sample covariance matrix:

$$\widehat{\Sigma}_n = \frac{1}{n}\sum_{i=1}^{n}(X_i - \widehat{\mu}_n)(X_i - \widehat{\mu}_n)^T, \qquad \widehat{\mu}_n = \underbrace{\frac{1}{n}\sum_{i=1}^{n} X_i}_{\text{sample mean}}$$

- eigendecomposition: $\widehat{\Sigma}_n = UDU^T$
- retain top k eigenvectors

Image source: MathWorks

What To Do If You Don't Have Enough Data?

Bootstrap

- Training data: $(X_1, Y_1), \ldots, (X_n, Y_n)$
- If n is small, there is risk of underfitting
- Bootstrap is a *resampling method*:

```
for j = 1 to m
    draw I(j) uniformly at random from {1, ..., n}
    let Z_j = (X_{I(j)}, Y_{I(j)})
end for
return Z_1, Z_2, ..., Z_m
```

- Bootstrap can be used to get a good estimate of error bars for a chosen class of models

Gradient Descent and Its Relatives

Training is an optimization problem:

$$\text{minimize } L(w) = \frac{1}{n}\sum_{i=1}^{n}\left(Y_i - f(w, X_i)\right)^2$$

— e.g., $x \mapsto f(w, x)$ is a deep neural network, so w is a (very) high-dimensional vector

- Gradient descent

$$w_{t+1} = w_t - \eta_t \nabla L(w_t), \qquad \eta_t - \text{learning rate}$$

- Backpropagation: use *chain rule* to compute gradients efficiently
- Computing $\nabla L(w)$ using backprop takes $O(nkm)$ operations, where k is the number of layers and m is the number of connections per layer
- Still computationally expensive, requires full pass through training data at each iteration

Stochastic Gradient Descent

- Gradient descent

$$w_{t+1} = w_t - \eta_t \nabla L(w_t), \qquad \eta_t \text{ – learning rate}$$

$$\nabla L(w) = \frac{1}{n} \sum_{i=1}^{n} \nabla_w (Y_i - f(w, X_i))^2$$

- Stochastic gradient with mini-batches: essentially, bootstrap

`for` $t = 1, 2, \ldots$

`draw` $I(1,t), \ldots, I(b,t)$ `with replacement from` $\{1, \ldots, n\}$

$$w_{t+1} = w_t - \eta_t \cdot \frac{1}{b} \sum_{j=1}^{b} \nabla_w \ell(Y_{I(j,t)} - f(w_t, X_{I(j,t)}))^2$$

`end for`

- Computational-statistical trade-offs:
  - each iteration takes $O(bkm)$ operations instead of $O(nkm)$
  - gradient error is now $\propto 1/b$ instead of $\propto 1/n$

Part IV
Machine Learning for EDA

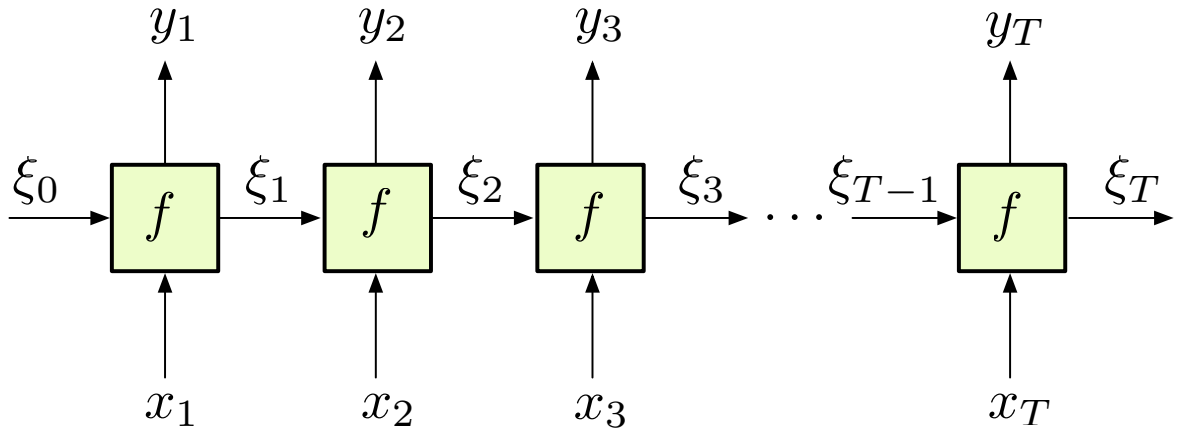
Main Challenges for EDA

- Training data are *waveforms*

$$X = (X(t))_{0 \le t \le T}, \qquad Y = (Y(t))_{0 \le t \le T}$$

  — e.g., X is voltage, Y is current
- Many different types of models/analysis:
  - steady-state
  - transient
  - small-signal
  - ...
- How do we represent waveform data?
  - sampling rate affects quality of learned models
  - different features are useful for different analyses (rise/hold time; frequency/phase; duration; ...)
- How can we optimize measurement collection process?

Recurrent Neural Networks

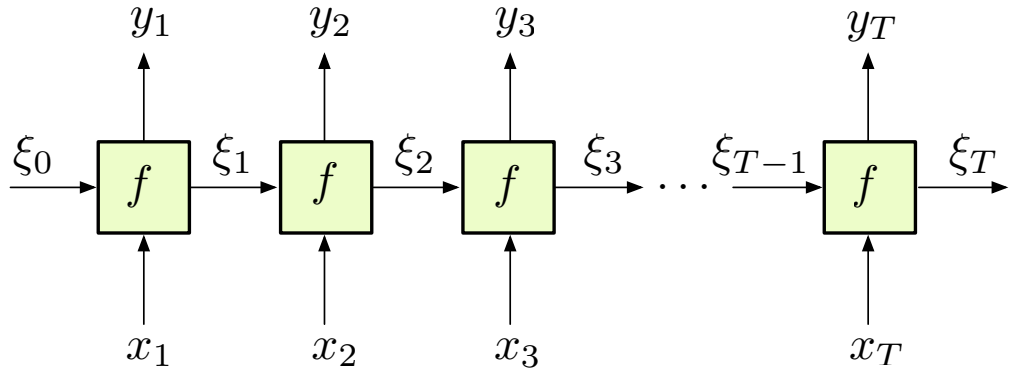


Nonlinear state-space model:

$$\xi_t = g(A\xi_{t-1} + Bx_t)$$
$$y_t = h(C\xi_t + Dx_t)$$

- Physics-based models of nonlinear electronic devices are state-space models
- RNNs are *universal approximators* for such systems
- States (internal variables) can be discovered automatically, without hand tuning

Learning RNN Models of Nonlinear Devices



- Data: sampled waveforms (τ - sampling period, T - # samples)

$$X_i = (X_i(0), X_i(\tau), \ldots, X_i(T\tau))$$
$$Y_i = (Y_i(0), Y_i(\tau), \ldots, Y_i(T\tau)) \qquad i = 1, \ldots, n$$

- RNN is a T-layer NN, with parameters shared by all neurons
- Training by SGD with Backpropagation Through Time (BPTT)
- Output: weight matrices A, B, C, D; discrete-time approximation

$$\xi_{k+1} = g(A\xi_k + Bx_k)$$
$$y_k = h(C\xi_k + Dx_k)$$

Main Challenges

- Which stimuli to use?
- The learned model may not be equally suitable for different types of analysis (transient vs. steady-state)
- The learned model should be stable, plug-and-play with Verilog-A
- Enforcing physical constraints: zero in/zero out, symmetry, polarity, etc.
- Usual issues with BPTT: vanishing/exploding gradients, etc.

Capturing Variability: Generative Modeling

$$\text{Device model:} \qquad y = g(x, \beta),$$

where x is input, y is output, β are device parameters.

- Variability: β is a random quantity which varies device to device, yet cannot be sampled directly
- Data: sample input-ouptut behavior from many devices of the same type
- Learn a *generative model* for (x, y)
  - use deep NN to learn complicated functions mapping explicit design parameters to latent variability factors
  - maximum likelihood via SGD with backprop
  - bootstrap can be used to compensate for lack of data
- Disadvantage: training takes time
- Advantage: once learned, model is easy to sample from, can be used to explore variability, 3σ typical behavior, etc.

Compositional ML

Training and inference/simulation pipeline:

$$\text{data} \xrightarrow{\text{training}} \left\{ \begin{aligned} \dot{\xi} &= f(\xi, x) \\ y &= g(\xi, x) \end{aligned} \right\} \xrightarrow{\text{Verilog-A}} \text{simulation}$$

- Theoretical and practical challenge: optimize end-to-end performance
- The learned model should be internally stable
  - enforce stability during training via appropriate penalty functions
- The learned model should be ready for Verilog-A
  - RNN outputs a discrete-time model
  - Verilog-A uses adaptive time discretization to simulate behavior
  - need deep understanding of numerical methods for ODEs to ensure end-to-end stability and robustness

For more questions:

maxim@illinois.edu