DS 4400

Machine Learning and Data Mining I

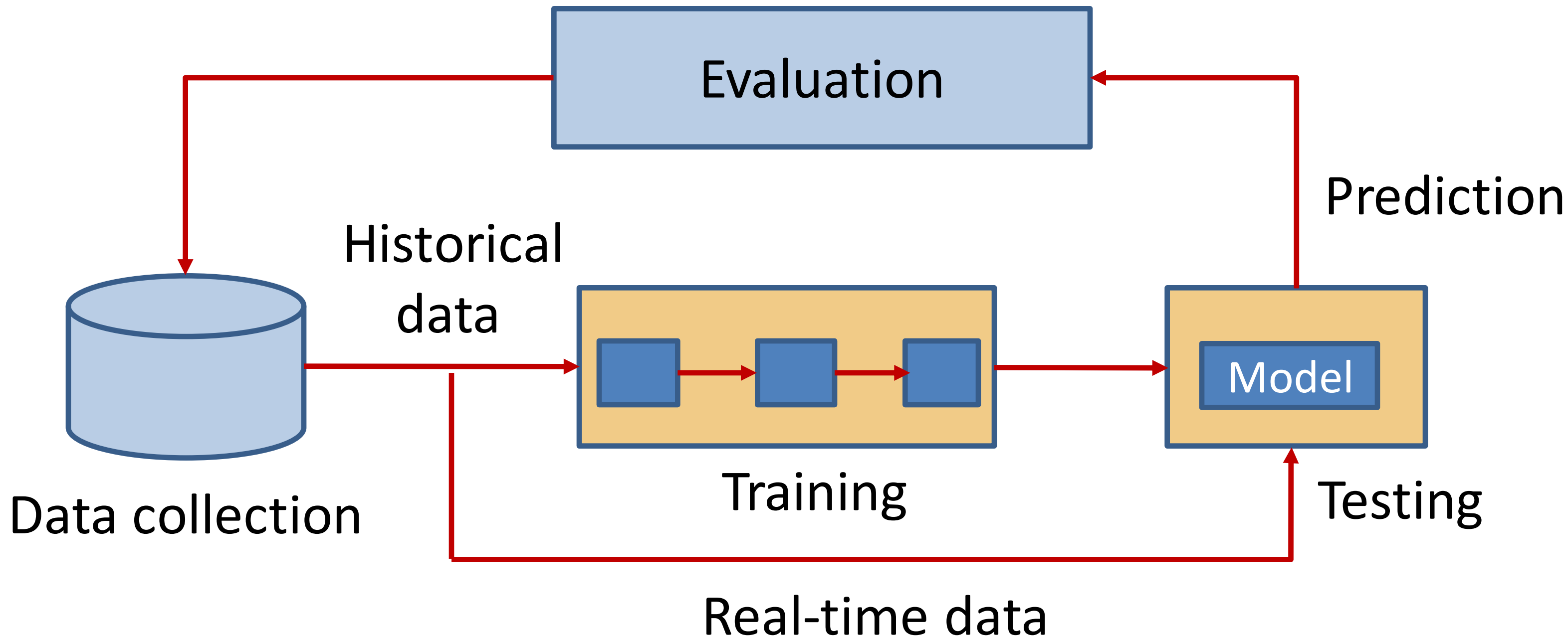
Alina Oprea
Associate Professor, CCIS
Northeastern University

November 29 2018

Logistics

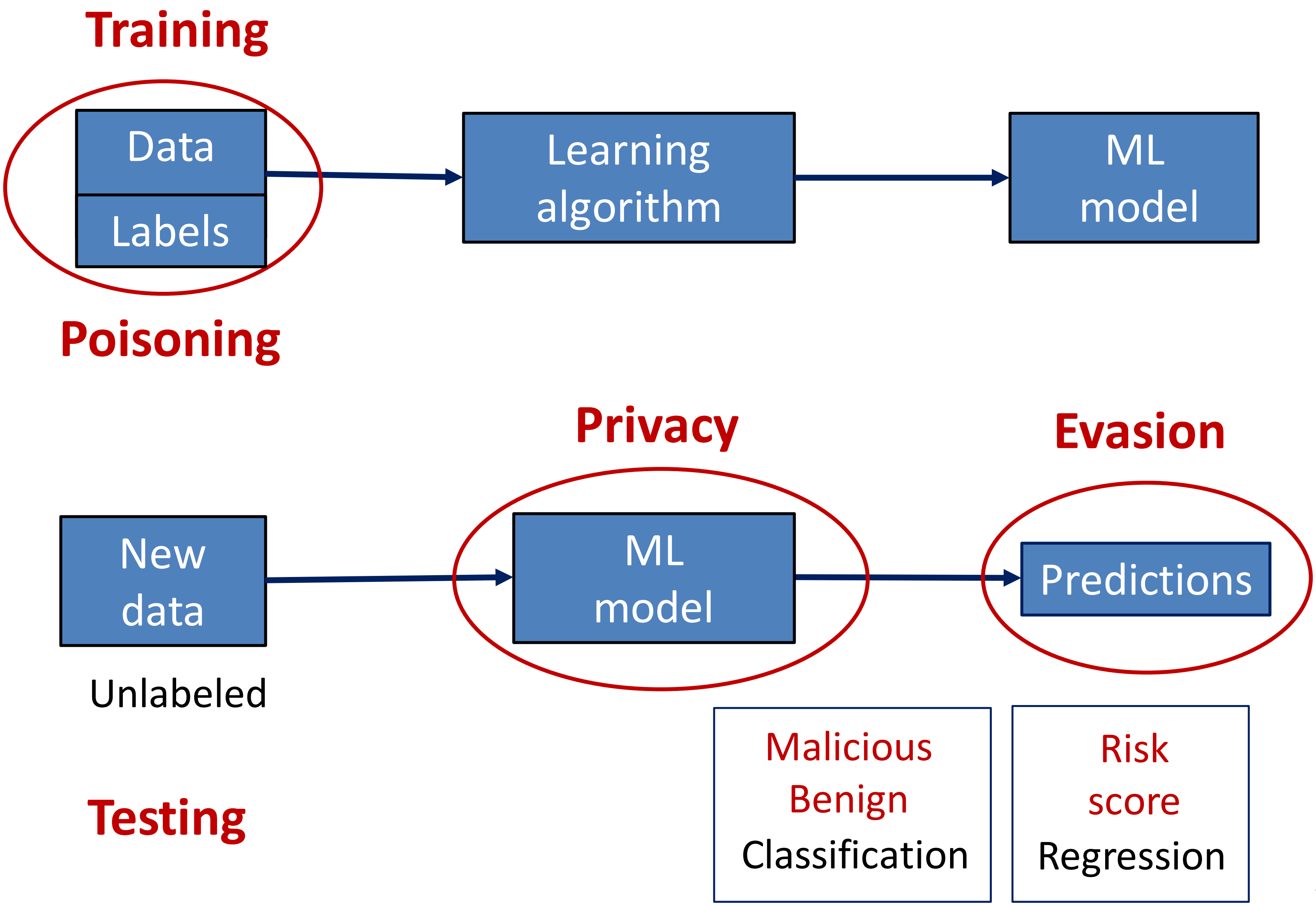
- Final projects
  - Presentations: Monday, Dec 3, 3-5:30pm in ISEC 655
  - Report: Friday, Dec 7 in Gradescope
- No class on Dec 4
- Final Exam
  - Office hours: Monday, Dec 10, 2-4pm
  - Tuesday, Dec 11, 2-5pm in ISEC 655

Adversarial Machine Learning



- Studies attacks against machine learning systems
- Designs robust machine learning algorithms that resist sophisticated attacks
- Many challenging open problems!

Attacks against supervised learning



Evasion Attacks

Given original example x, $f(x) = c$
Find adversarial example x'

$$\min ||x - x'||_2^2$$

Such that $f(x') = t$

x' is in range

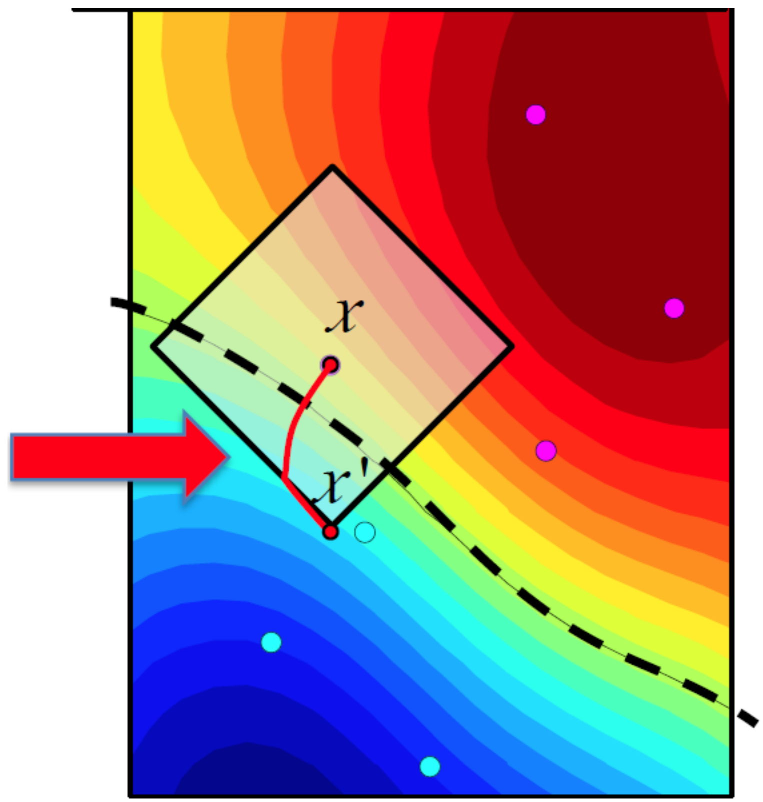
Equivalent formulation

$$\min c||\delta||_2^2 + \ell_t(x + \delta)$$

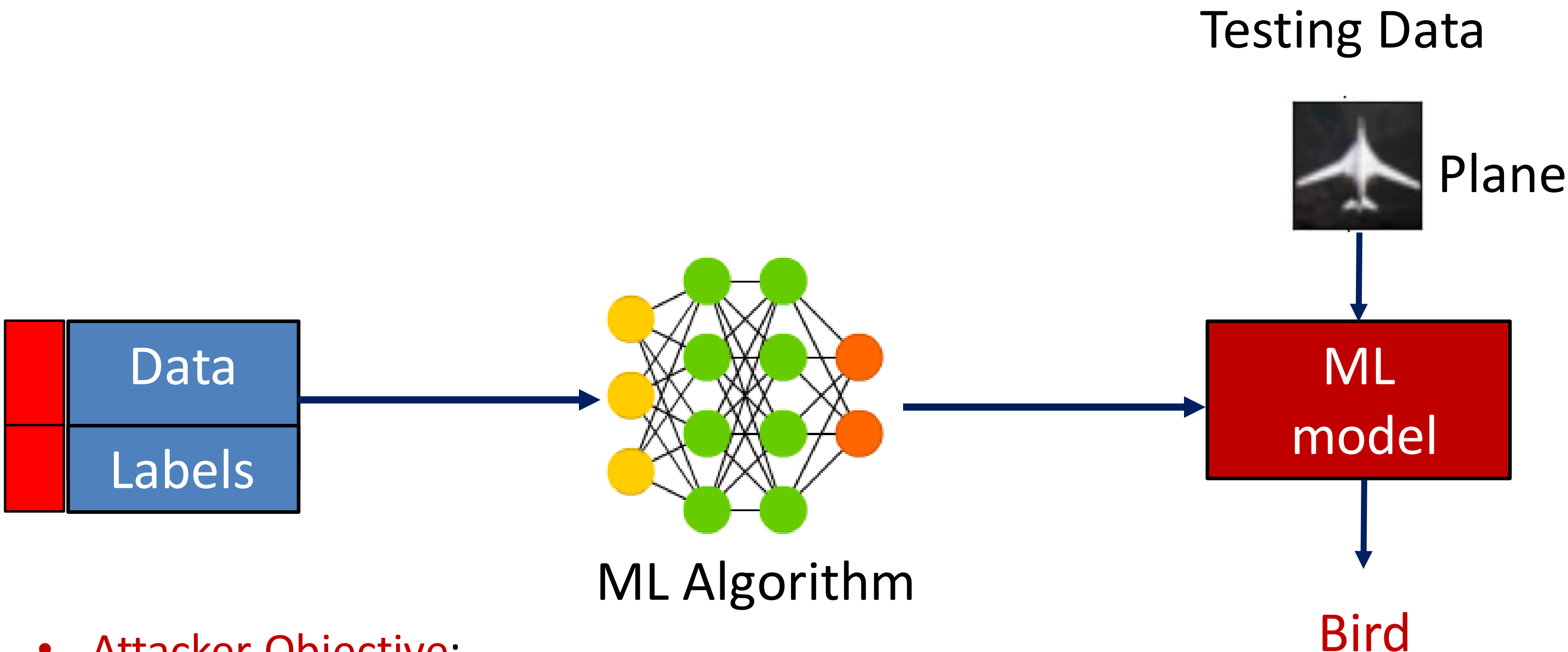
$$x' = x + \delta$$

$\ell_t(x')$ is loss function on x'

[Szegedy et al. 13] Intriguing properties of neural networks



Poisoning Availability Attacks



- Attacker Objective:
  - Corrupt the predictions by the ML model significantly
  - Predictions on *most points* are impacted in testing
- Attacker Capability:
  - Insert fraction of poisoning points in training
- [M. Jagielski, A. Oprea, B. Biggio, C. Liu, C. Nita-Rotaru, and B. Li. Manipulating Machine Learning: Poisoning Attacks and Countermeasures for Regression Learning. In IEEE S&P 2018]

Optimization Formulation

Given a training set D find a set of poisoning data points D_p

that maximizes the adversary objective A on validation set D_{val}

where corrupted model $\boldsymbol{\theta}_p$ is learned by minimizing the loss function L on $D \cup D_p$

$$\underset{D_p}{\operatorname{argmax}}\, A(D_{val}, \boldsymbol{\theta}_p)\ s.t.$$

$$\boldsymbol{\theta}_p \in \underset{\boldsymbol{\theta}}{\operatorname{argmin}}\, L(D \cup D_p, \boldsymbol{\theta})$$

Implicit dependence

Optimization formulation in white-box setting

- Attacker knows training data D
- Attacker knows ML model and loss function L

Gradient Ascent Algorithm

- **Input**: poisoned point x_0, label y_0
  - Adversarial objective A
- **Output**: poisoned point x, label y

1. Initialize poisoned point $x \leftarrow x_0; y \leftarrow y_0$
2. Repeat
   - Store previous iteration $x_{pr} \leftarrow x; y_{pr} \leftarrow y$
   - Update in direction of gradients choosing α with line search and project to feasible space

$$x \leftarrow \Pi(\mathrm{x} + \alpha \nabla_x A(x, y))$$
$$\mathrm{y} \leftarrow \Pi(\mathrm{y} + \alpha \nabla_y A(x, y))$$

3. Until $\left|A(x, y) - A\left(x_{pr}, y_{pr}\right)\right| < \epsilon$
4. Return x, y

- Most ML models are vulnerable in face of attacks!
  - Evasion (testing-time) attacks
  - Poisoning (training-time) attacks
  - Privacy attacks
- How to make ML more robust to attacks?

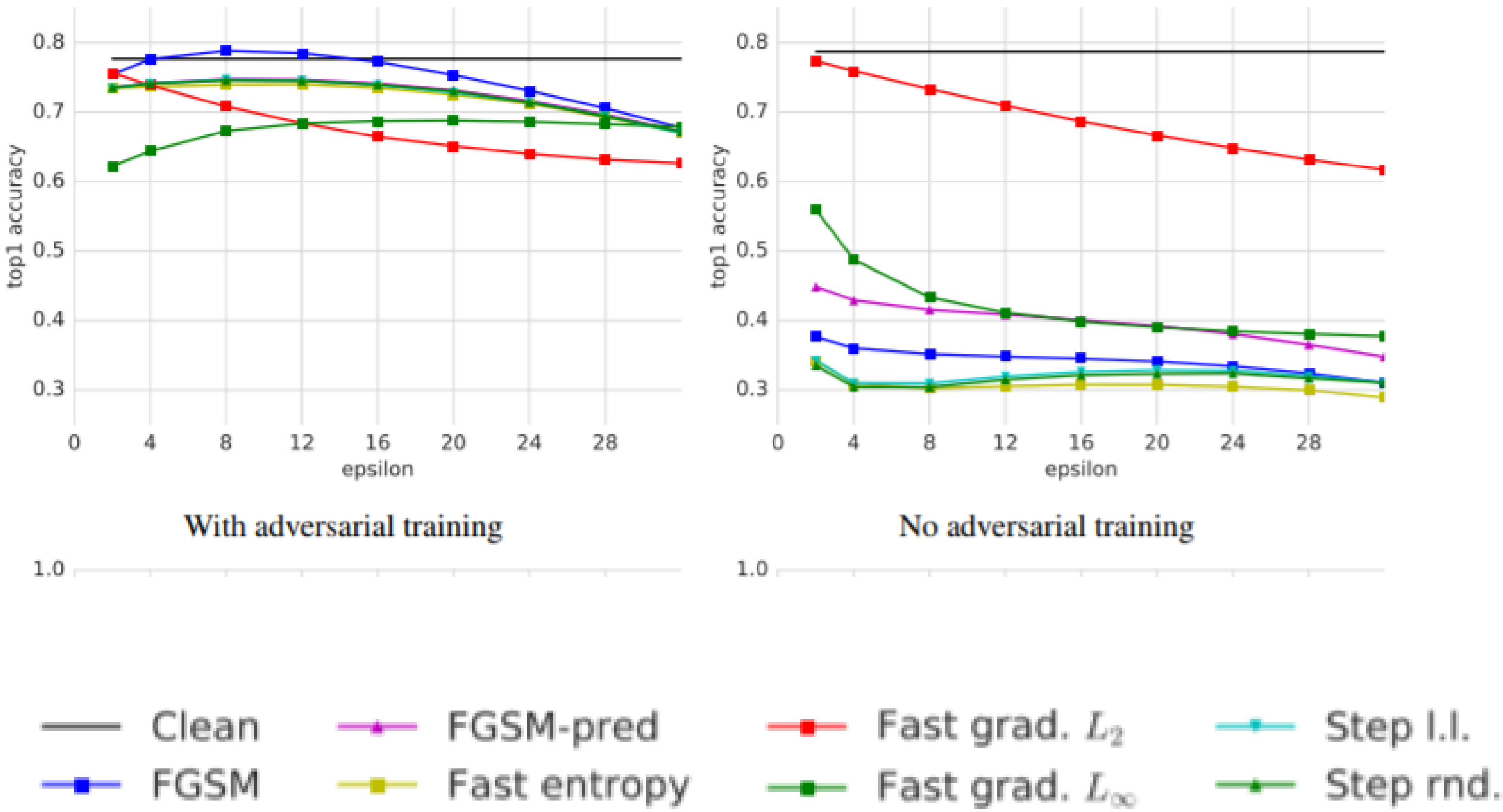
Adversarial Training

Algorithm 1 Adversarial training of network N.
Size of the training minibatch is m. Number of adversarial images in the minibatch is k.

Randomly initialize network N
repeat
    Read minibatch $B = \{X^1, \ldots, X^m\}$ from training set
    Generate k adversarial examples $\{X^1_{adv}, \ldots, X^k_{adv}\}$ from corresponding clean examples $\{X^1, \ldots, X^k\}$ using current state of the network N
    Make new minibatch $B' = \{X^1_{adv}, \ldots, X^k_{adv}, X^{k+1}, \ldots, X^m\}$
    Do one training step of network N using minibatch B'
until training converged

- I. Goodfellow et al. Explaining and harnessing adversarial examples, ICLR 2015.
- A. Kurakin et al. Adversarial Machine Learning at Scale, ICLR 2017.

Is Adv Training Effective?

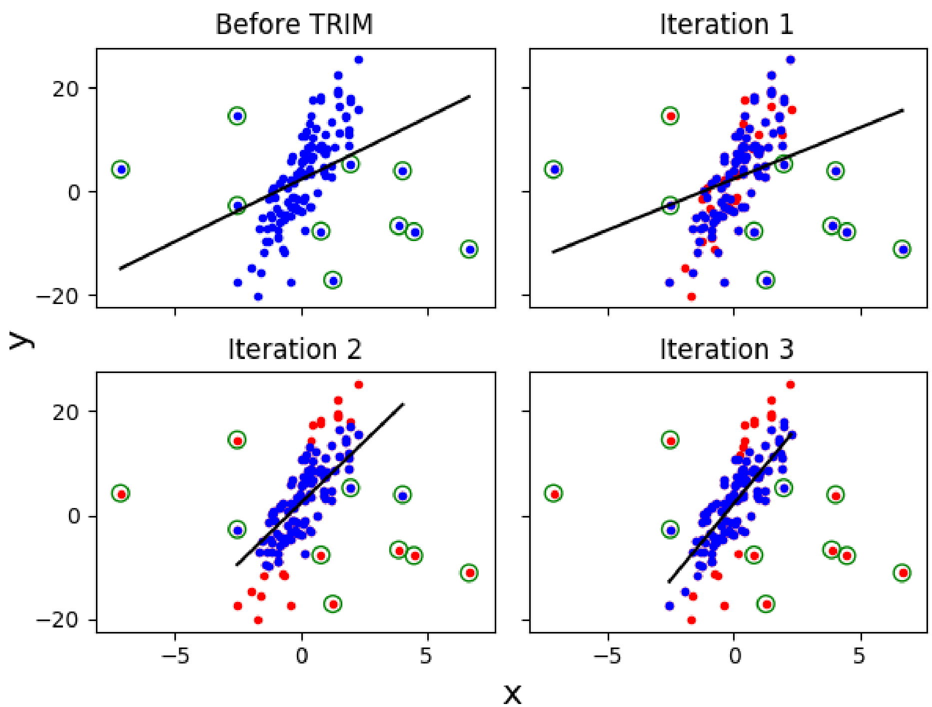


Resilient Linear Regression

- Goal
  - Train a robust linear regression model, assuming $\alpha \cdot n$ poisoned points among N points in training
  - MSE should be close to original MSE
  - No ground truth on data distribution available
- Existing techniques
  - Robust statistics
    - Huber [Huber 1964], RANSAC [Fischler and Bolles 1961]
    - Resilient against outliers and random noise
  - Adversarial resilient regression: [Chen et al. 13]
    - Make simplifying assumption on data distribution (e.g., Gaussian)

Our Defense: TRIM

- Given dataset on n points and αn attack points, find best model on n of $(1+\alpha)n$ points
- If $\boldsymbol{w}, b$ are known, find points with smallest residual
- But $\boldsymbol{w}, b$ and true data distribution are unknown!



TRIM: alternately estimate model and find low residual points

$$\underset{w,b,I}{\operatorname{argmin}} L(w,b,I) = \frac{1}{|I|}\sum_{i\in I}(f(\boldsymbol{x}_i) - y_i)^2 + \lambda\Omega(\boldsymbol{w})$$

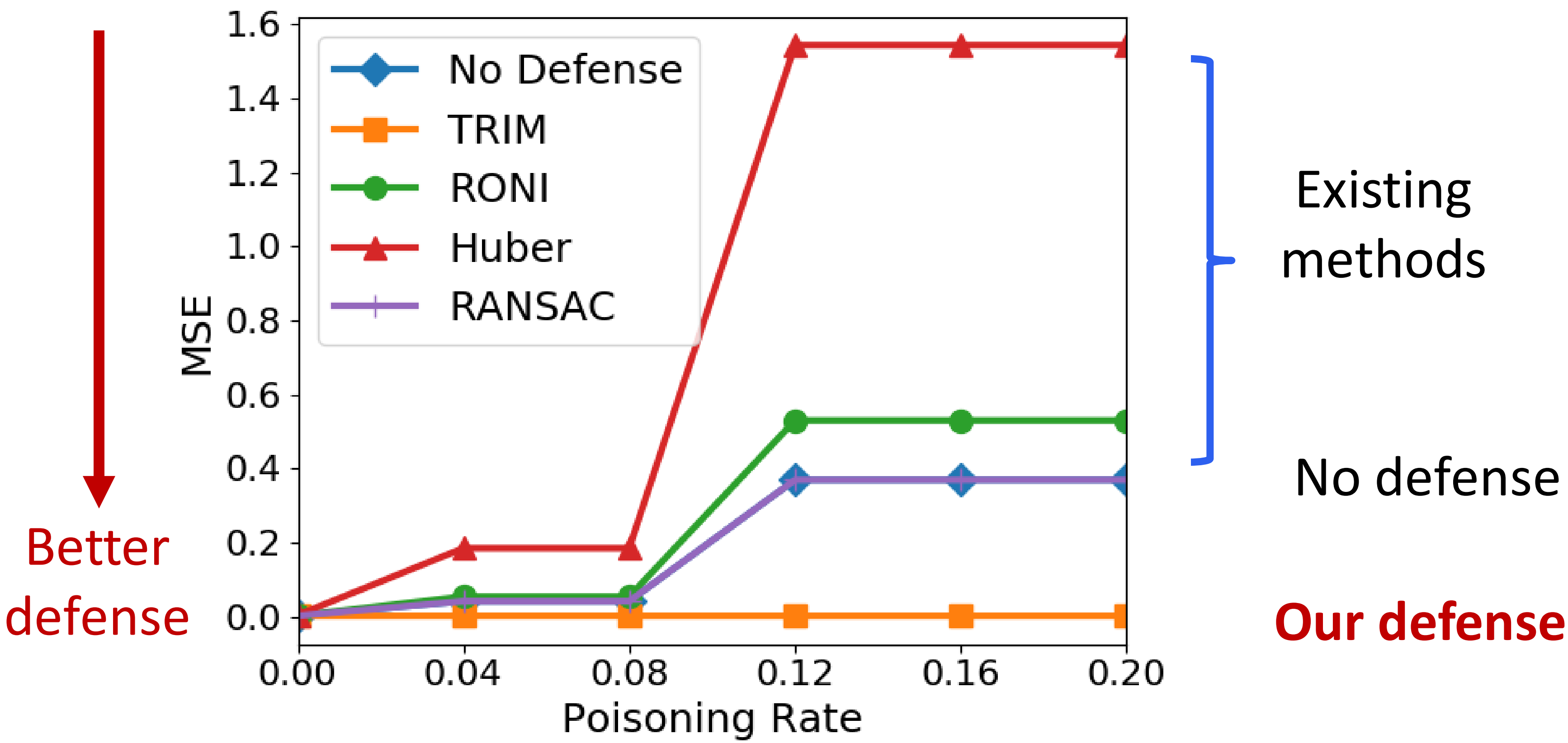
$$N = (1+\alpha)n, \qquad I \subset [1,\dots,N], \qquad |I| = n$$

Trimmed optimization

- Estimate model parameters and identify points with minimum residual alternatively
  - Alternating optimization
- Select I a random subset in $\{1, \dots, N\}, |I| = n$
  - Assume poisoning rate (or upper bound) is known
- Repeat
  - Estimate $(w, b) = \operatorname{argmin} L(w, b, I)$
  - Select new set I of points, $|I| = n$, with lowest residuals under new model
- Until convergence (loss does not decrease)

Defense results

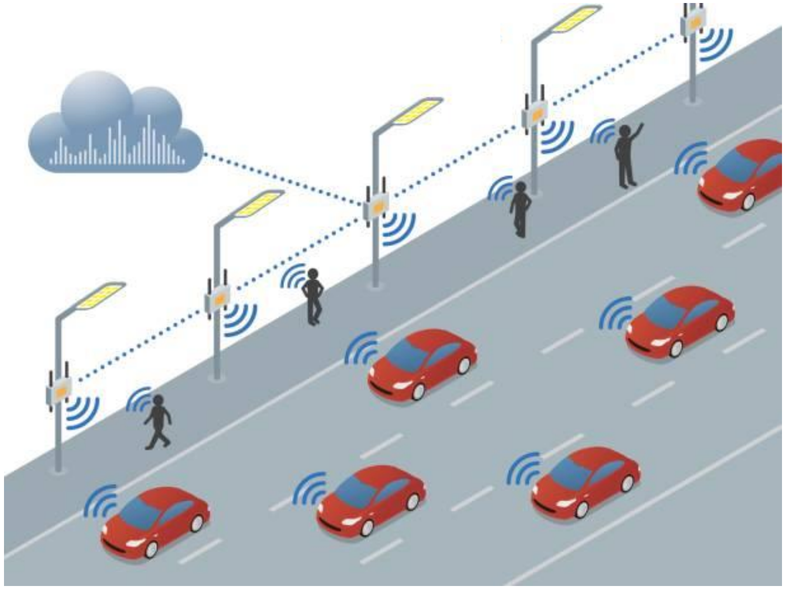
- TRIM MSE is within 1% of the original model MSE
- Significant improvement over existing methods

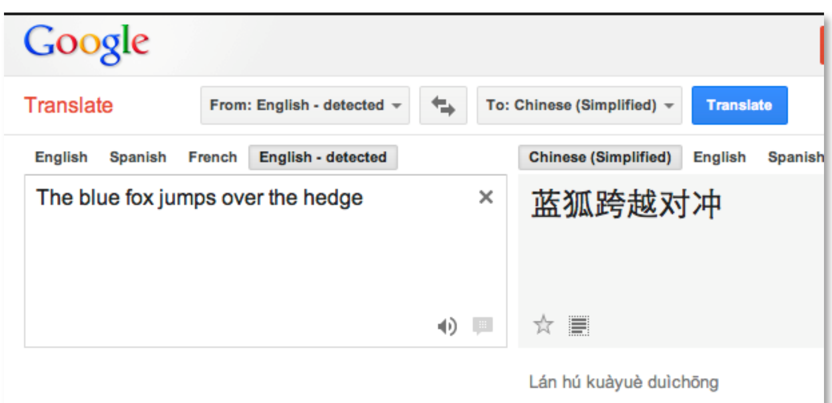


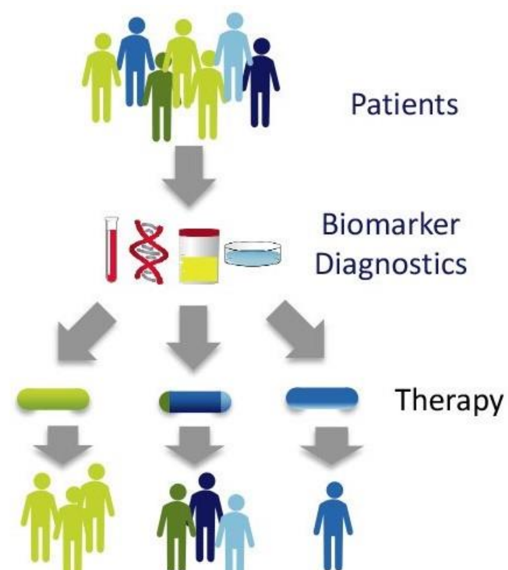
Predict house price with LASSO regression
(i.e., with L1 regularization)

Review

Machine learning is everywhere







DS-4400 Course objectives

- Become familiar with machine learning tasks
  - Supervised learning vs unsupervised learning
  - Classification vs Regression vs Clustering
- Study most well-known algorithms and understand to which problem they apply
  - Regression (linear regression)
  - Classification (SVM, decision trees, neural networks)
  - Clustering (k-means)
- Learn to apply ML algorithms to real datasets
  - Using existing packages in R and Python
- Learn about security challenges of ML
  - Introduction to adversarial ML

http://www.ccs.neu.edu/home/alina/classes/Fall2018/

What we covered

Adversarial ML

Ensembles
- Bagging
- Random forests
- Boosting
- AdaBoost

Deep learning
- Feed-forward Neural Nets
- Convolutional Neural Nets
- Recurrent Neural Nets
- Back-propagation

Unsupervised
- PCA
- Auto-encoders
- Clustering

Linear classification
- Perceptron
- Logistic regression
- LDA
- Linear SVM

Non-linear classification
- kNN
- Decision trees
- Kernel SVM
- Naïve Bayes

- Metrics
- Cross-validation
- Regularization
- Feature selection
- Gradient Descent
- Density Estimation

Linear Regression

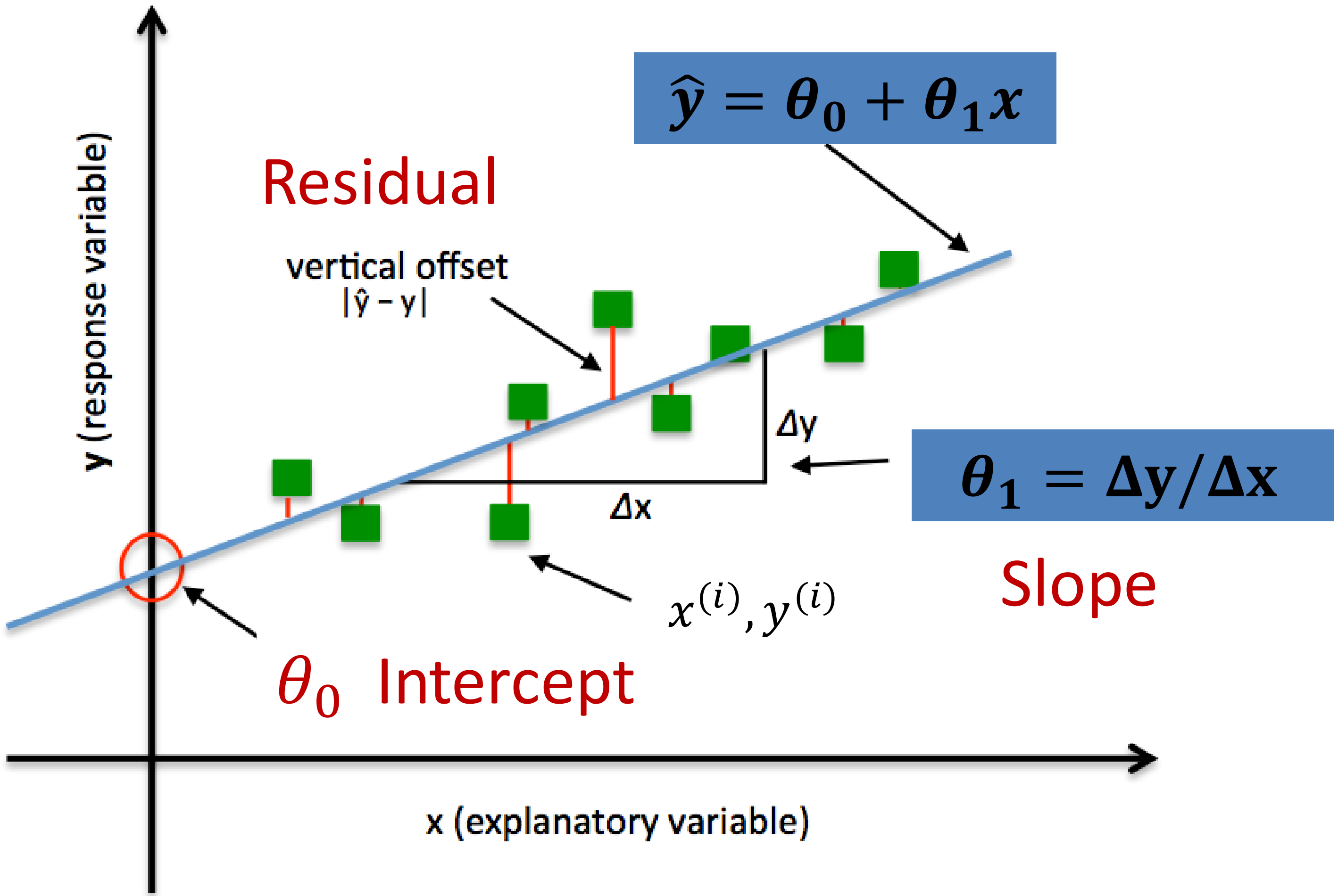
Linear algebra

Probability and statistics

Terminology

- Hypothesis space $H = \{f: X \to Y\}$
- Training data $\mathrm{D} = (x_\mathrm{i}, y_\mathrm{i}) \in X \times Y$
- Features: $x_i \in X$
- Labels $y_\mathrm{i} \in Y$
  - Classification: discrete $y_\mathrm{i} \in$ {-1,1}
  - Regression: $y_\mathrm{i} \in \mathrm{R}$
- Loss function: $L(f, D)$
  - Measures how well f fits training data
- Training algorithm: Find hypothesis $\hat{f}: X \to Y$
  - $\hat{f} = \underset{f \in H}{\operatorname{argmin}} \; L(f, D)$

Linear Regression



$$h_\theta(x) = \theta_0 + \theta_1 x$$

MSE$= \frac{1}{n}\sum_{i=1}^{n}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2$

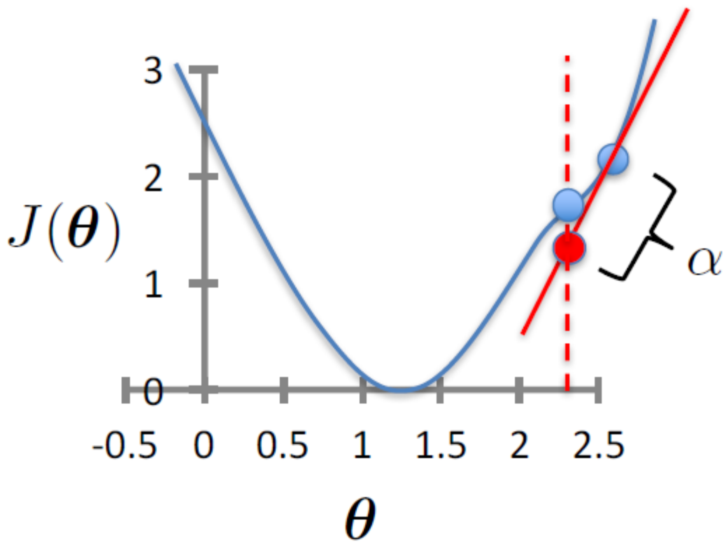
Gradient Descent

- Initialize $\boldsymbol{\theta}$
- Repeat until convergence

$$\theta_j \leftarrow \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\boldsymbol{\theta})$$

simultaneous update
for j = 0 ... d

learning rate (small)
e.g., α = 0.05

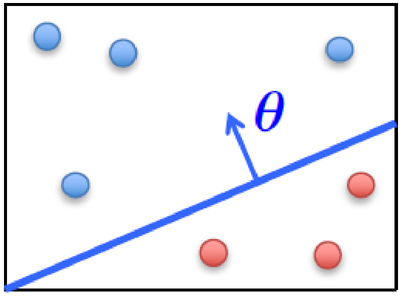


Gradient = slope of line tangent
to curve at the same point

Linear classifiers

- **Linear classifiers**: represent decision boundary by hyperplane

$$\boldsymbol{\theta} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_d \end{bmatrix} \quad \boldsymbol{x}^\intercal = \begin{bmatrix} 1 & x_1 & \dots & x_d \end{bmatrix}$$



$$h(\boldsymbol{x}) = \text{sign}(\boldsymbol{\theta}^\intercal \boldsymbol{x}) \quad \text{where} \quad \text{sign}(z) = \begin{cases} 1 & \text{if } z \geq 0 \\ -1 & \text{if } z < 0 \end{cases}$$

– Note that: $\boldsymbol{\theta}^\intercal \boldsymbol{x} > 0 \implies y = +1$

$$\boldsymbol{\theta}^\intercal \boldsymbol{x} < 0 \implies y = -1$$

All the points x on the hyperplane satisfy: $\theta^T x = 0$

Online Perceptron

Let $\boldsymbol{\theta} \leftarrow [0, 0, \ldots, 0]$
Repeat:
  Receive training example $(\boldsymbol{x}^{(i)}, y^{(i)})$
  if $y^{(i)}\theta^T x^{(i)} \leq 0$ // prediction is incorrect
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + y^{(i)}\boldsymbol{x}^{(i)}$

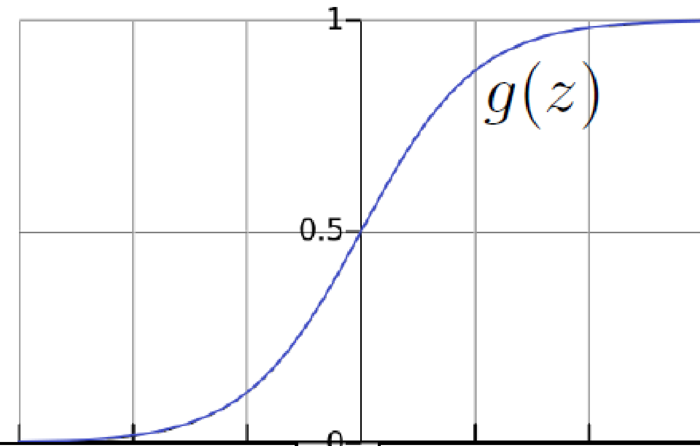
Online learning – the learning mode where the model update is performed each time a single observation is received

Batch learning – the learning mode where the model update is performed after observing the entire training set

Logistic Regression

$$h_{\boldsymbol{\theta}}(\boldsymbol{x}) = g\left(\boldsymbol{\theta}^\intercal \boldsymbol{x}\right)$$

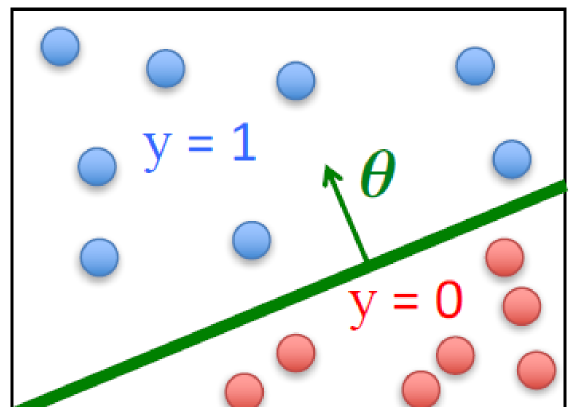
$$g(z) = \frac{1}{1 + e^{-z}}$$



$\boldsymbol{\theta}^\intercal \boldsymbol{x}$ should be large <u>negative</u> values for negative instances

$\boldsymbol{\theta}^\intercal \boldsymbol{x}$ should be large <u>positive</u> values for positive instances

- Assume a threshold and...
  - Predict y = 1 if $h_{\boldsymbol{\theta}}(\boldsymbol{x}) \geq 0.5$
  - Predict y = 0 if $h_{\boldsymbol{\theta}}(\boldsymbol{x}) < 0.5$



Logistic Regression is a linear classifier!

LDA

Given training data $(x^{(i)}, y^{(i)}), i = 1, \dots, n, y^{(i)} \in \{1, \dots, K\}$

1. Estimate mean and variance

$$\hat{\mu}_k = \frac{1}{n_k} \sum_{i:y_i=k} x^{(i)}$$

$$\hat{\sigma}^2 = \frac{1}{n-K} \sum_{k=1}^{K} \sum_{i:y_i=k} (x^{(i)} - \hat{\mu}_k)^2$$

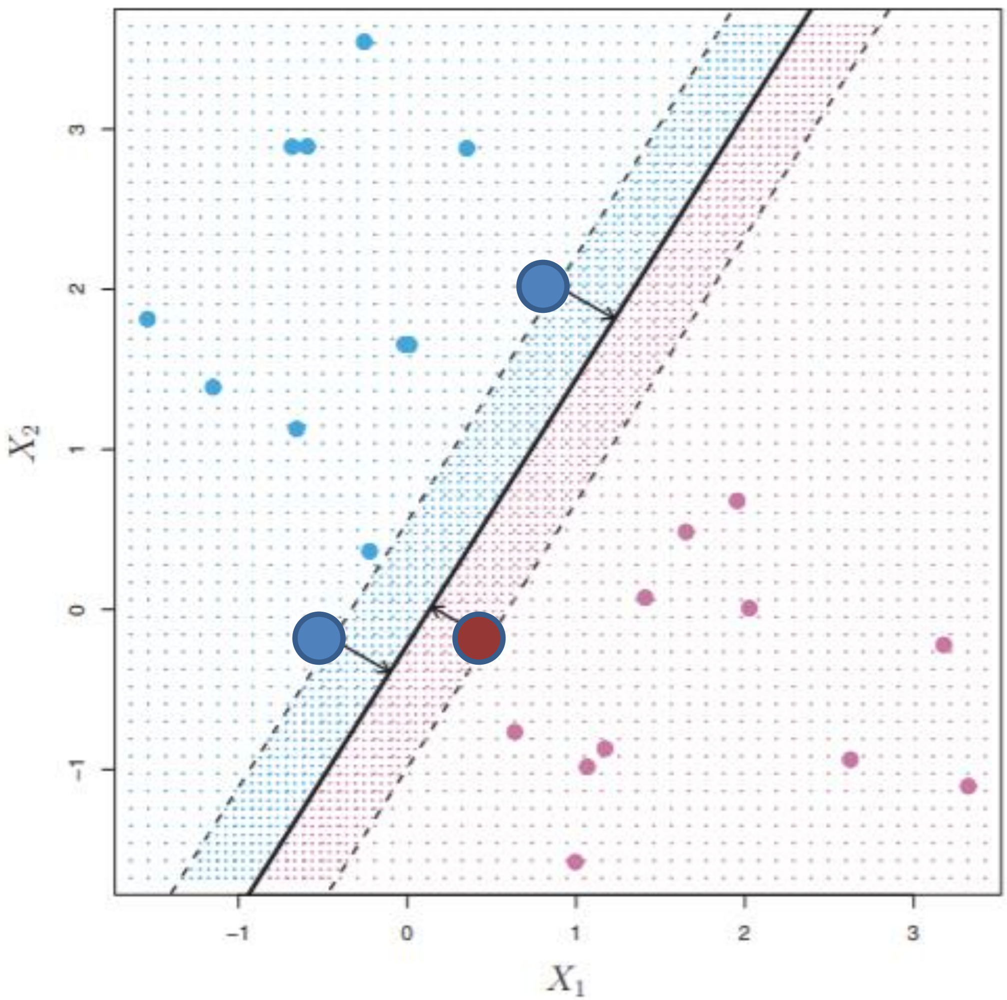
2. Estimate prior

$$\hat{\pi}_k = n_k / n.$$

Given testing point x, predict k that maximizes:

$$\hat{\delta}_k(x) = x \cdot \frac{\hat{\mu}_k}{\hat{\sigma}^2} - \frac{\hat{\mu}_k^2}{2\hat{\sigma}^2} + \log(\hat{\pi}_k)$$

SVM - Max Margin



- Support vectors are “closest” to hyperplane
- If support vectors change, classifier changes

SVM with Kernels

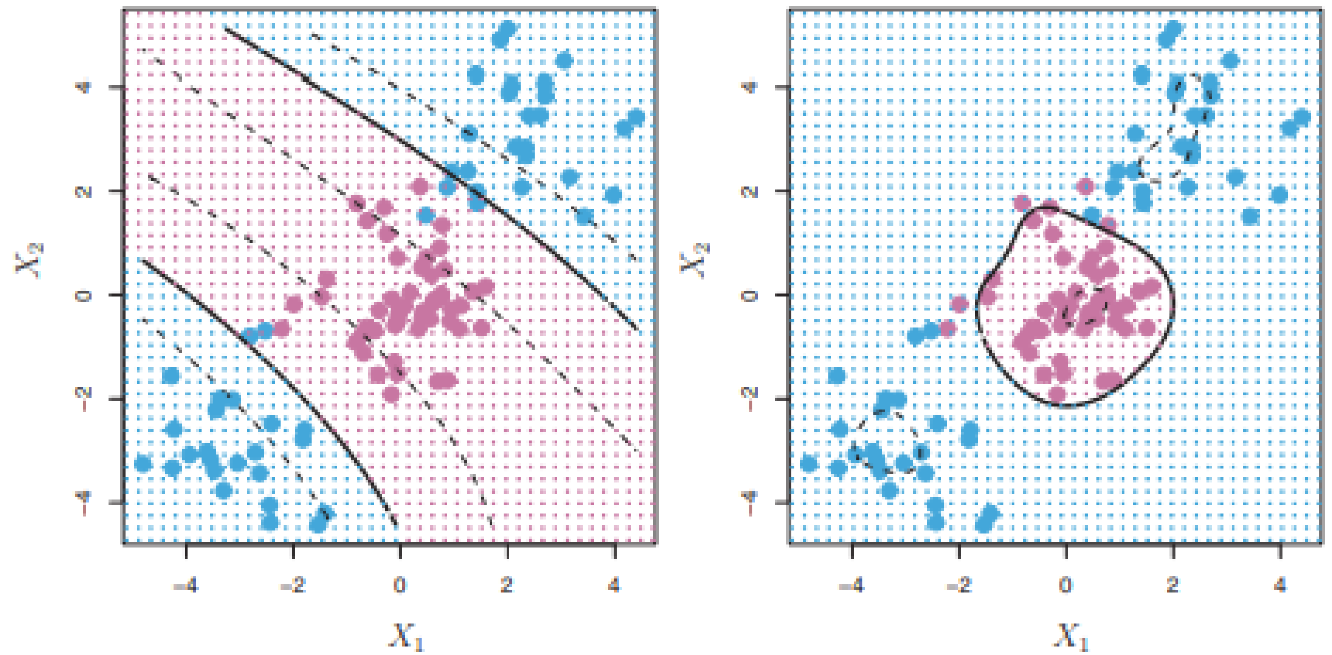


FIGURE 9.9. Left: *An SVM with a polynomial kernel of degree 3 is applied to the non-linear data from Figure 9.8, resulting in a far more appropriate decision rule.* Right: *An SVM with a radial kernel is applied. In this example, either kernel is capable of capturing the decision boundary.*

Kernels

- Linear kernels
  - $K(a,b) =< a,b > = \sum_i a_i b_i$
- Polynomial kernel of degree *m*
  - $K(a,b) = \left(1 + \sum_{i=0}^{d} a_i b_i\right)^m$
- Radial Basis Function (RBF) kernel (or Gaussian)
  - $K(a,b) = \exp\left(-\gamma \sum_{i=0}^{d} (a_i - b_i)^2\right)$
- Support vector machine classifier
  - $\mathrm{h}(z) = \theta_0 + \sum_{i \in S} \alpha_i K(z, x^{(i)})$

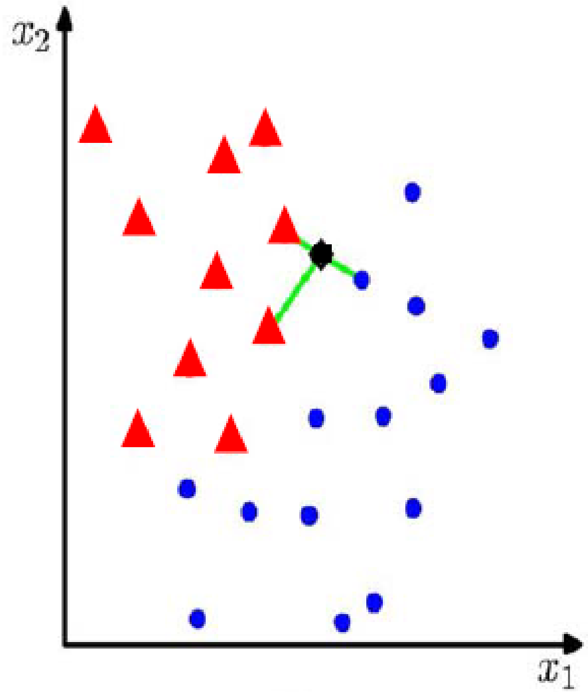
K Nearest Neighbour (K-NN) Classifier

Algorithm

- For each test point, x, to be classified, find the K nearest samples in the training data
- Classify the point, x, according to the majority vote of their class labels

e.g. K = 3

- applicable to multi-class case



Learning Decision Trees

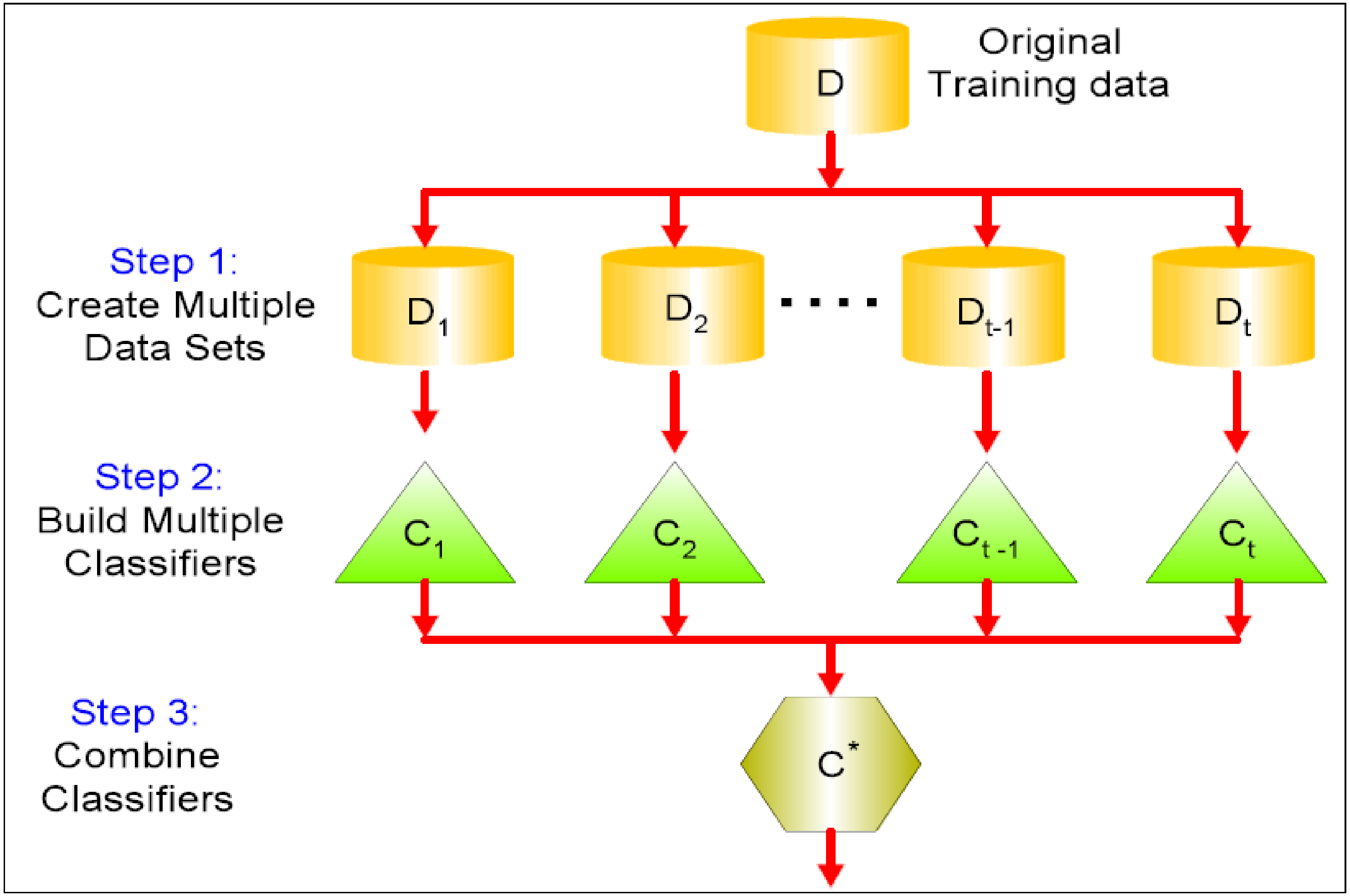
- Start from empty decision tree
- Split on **next best attribute (feature)**
  - Use, for example, information gain to select attribute:

$$\arg\max_i IG(X_i) = \arg\max_i H(Y) - H(Y \mid X_i)$$

- Recurse

ID3 algorithm uses Information Gain
Information Gain reduces uncertainty on Y

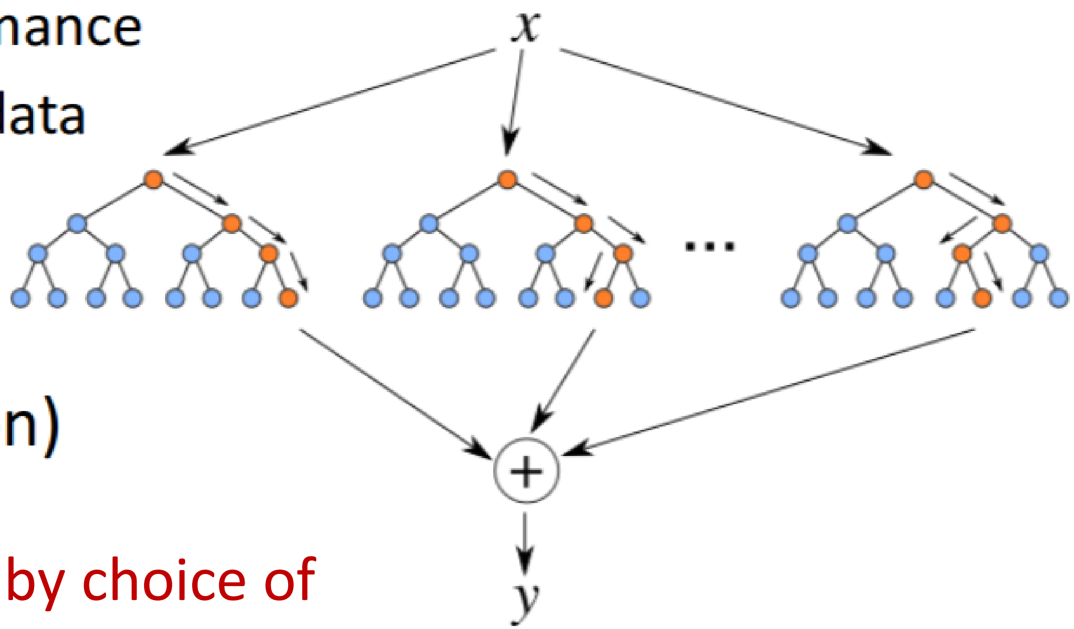
Ensembles



Majority Votes

Random Forests

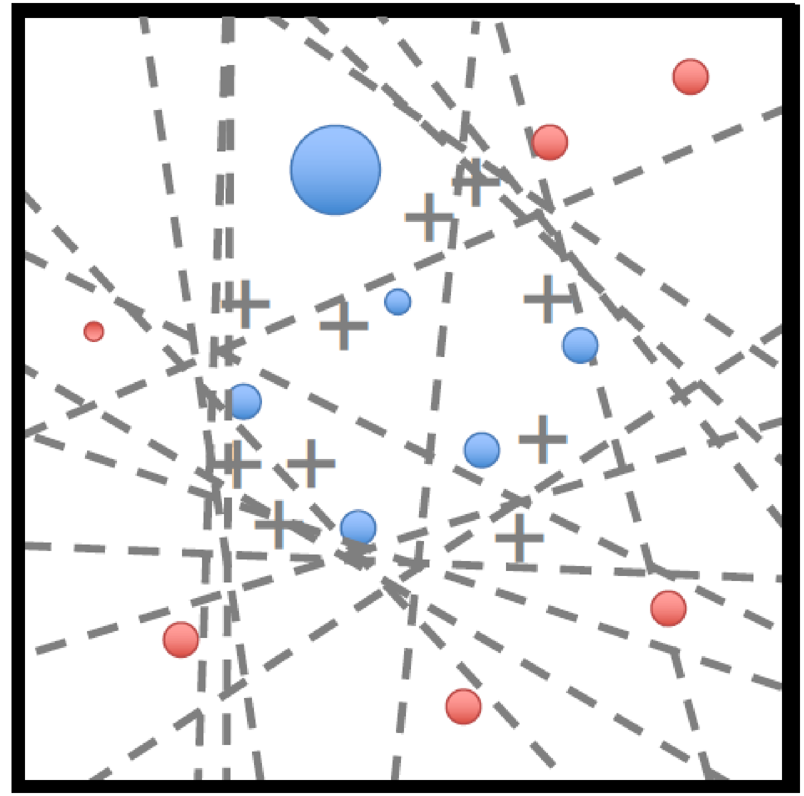
- Construct decision trees on bootstrap replicas
  - Restrict the node decisions to a small subset of features picked randomly for each node
- Do not prune the trees
  - Estimate tree performance on out-of-bootstrap data
- Average the output of all trees (or choose mode decision)



Trees are de-correlated by choice of random subset of features

AdaBoost

$t = \mathrm{T}$



Initialize a vector of n uniform weights $\mathbf{w}_1$
for $t = 1, \ldots, T$
    Train model h_t on X, y with weights $\mathbf{w}_t$
    Compute the weighted training error of h_t
    Choose $\beta_t = \frac{1}{2} \ln \left(\frac{1-\epsilon_t}{\epsilon_t} \right)$
    Update all instance weights:

$$w_{t+1,i} = w_{t,i} \exp(-\beta_t y^{(i)} h_t(x^{(i)}))$$

    Normalize $\mathbf{w}_{t+1}$ to be a distribution
end for
Return the hypothesis

$$H(\mathbf{x}) = \text{sign}\left(\sum_{t=1}^{T} \beta_t h_t(\mathbf{x}) \right)$$

- Final model is a weighted combination of members
  - Each member weighted by its importance

Naïve Bayes Classifier

Idea: Use the training data to estimate

$$P(X \mid Y) \text{ and } P(Y) .$$

Then, use Bayes rule to infer $P(Y|X_{\text{new}})$ for new data

---

Easy to estimate from data

Impractical, but necessary

$$\mathrm{P}[Y = k|X = x] = \frac{\mathrm{P}[Y = k]\,\mathrm{P}[X_1 = x_1 \wedge \cdots \wedge X_d = x_d | Y = k]}{\mathrm{P}[X_1 = x_1 \wedge \cdots \wedge X_d = x_d]}$$

Unnecessary, as it turns out

- Recall that estimating the joint probability distribution $P(X_1, X_2, \ldots, X_d \mid Y)$ is not practical

Confusion Matrix

- Given a dataset of P positive instances and N negative instances:

Predicted Class

Actual Class	Yes	No
Yes	TP	FN
No	FP	TN

$$\text{accuracy} = \frac{TP + TN}{P + N}$$

- Imagine using classifier to identify positive cases (i.e., for information retrieval)

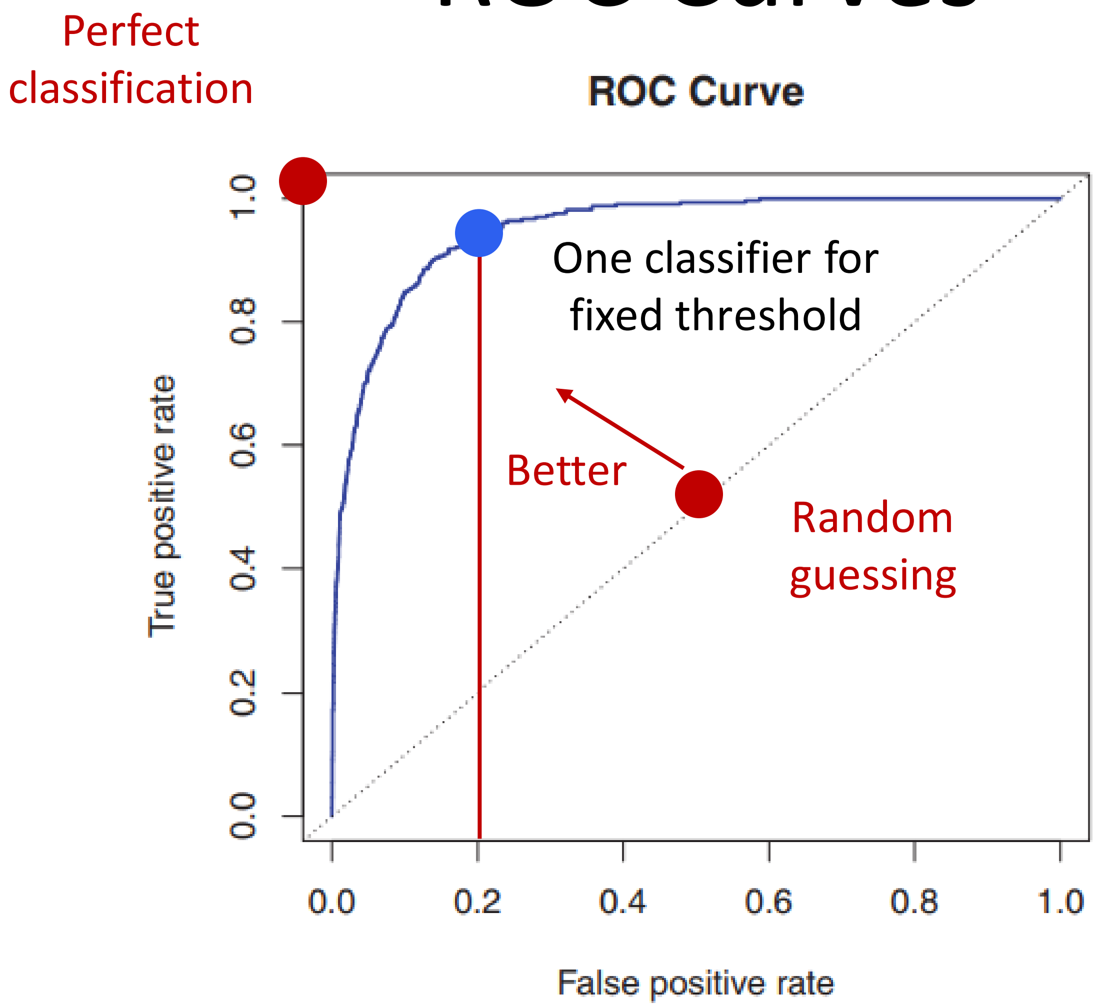
$$\text{precision} = \frac{TP}{TP + FP}$$

$$\text{recall} = \frac{TP}{TP + FN}$$

Probability that classifier predicts positive correctly

Probability that actual class is predicted correctly

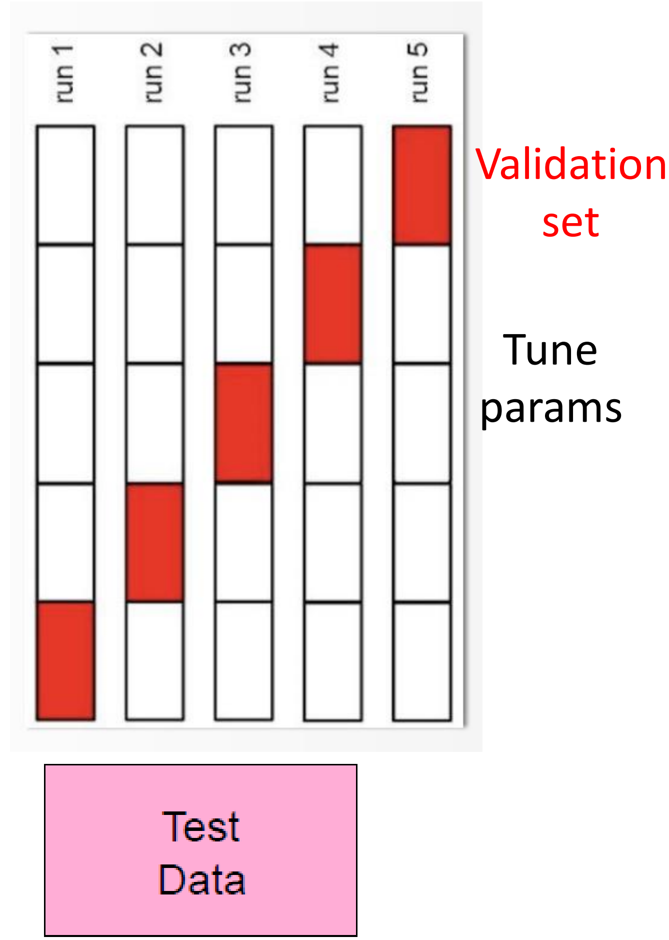
ROC Curves



- Receiver Operating Characteristic (ROC)
- Determine operating point (e.g., by fixing false positive rate)

Cross Validation

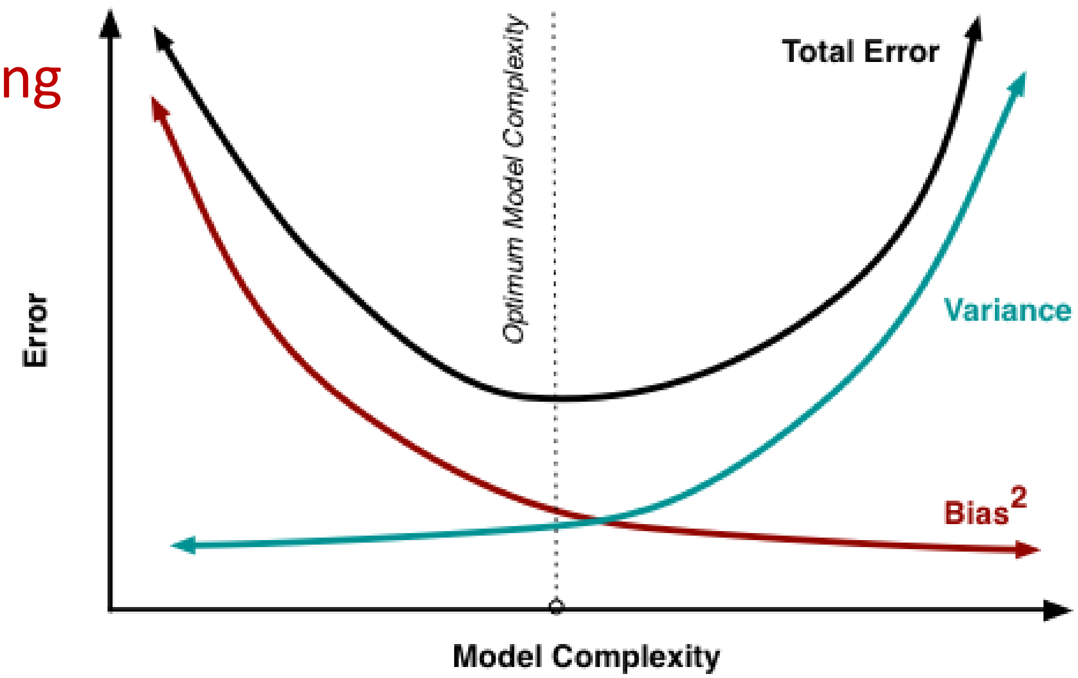
- Data: labeled instances, e.g. emails marked spam/ham
  - Training set
  - Test set
  - Randomly split training set into training and validation, e.g., 66% - 33%
- Features: attribute-value pairs which characterize each x
- Experimentation cycle
  - Select a hypothesis *f* (Tune hyperparameters on held-out or *validation* set)
  - Estimate and reduce average error during multiple runs by randomly choosing validation set
  - Compute final error on testing set
- Evaluation
  - Accuracy: fraction of instances predicted correctly
  - Use other metrics as appropriate (precision, recall)



- Improves model generalization
- Avoids overfitting

Bias-Variance Tradeoff

Under-fitting



Over-fitting

- Bias = Difference between estimated and true models
- Variance = Model difference on different training sets

Regularization

- A method for controlling the complexity of learned hypothesis

$$J(\boldsymbol{\theta}) = \underbrace{\frac{1}{2}\sum_{i=1}^{n}\left(h_{\boldsymbol{\theta}}\left(\boldsymbol{x}^{(i)}\right) - y^{(i)}\right)^2}_{\text{model fit to data}} + \underbrace{\frac{\lambda}{2}\sum_{j=1}^{d}\theta_j^2}_{\text{regularization}}$$

Ridge

$$J(\theta) = \underbrace{\sum_{i=1}^{n}(h_\theta(x^{(i)}) - y^{(i)})^2}_{\text{Squared Residuals}} + \underbrace{\lambda\sum_{j=1}^{d}|\theta_j|}_{\text{Regularization}}$$

LASSO

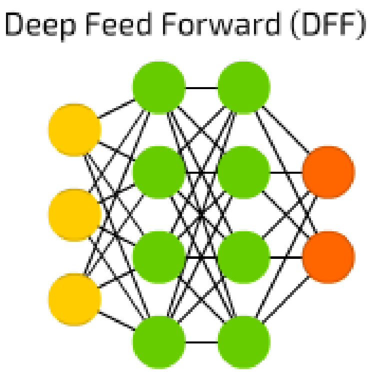
Methods for Feature Selection

- Wrappers
  - Select subset of features that gives best prediction accuracy (using cross-validation)
  - Model-specific
- Filters
  - Compute some statistical metrics (correlation coefficient, mutual information)
  - Select features with statistics higher than threshold
- Embedded methods
  - Feature selection done as part of training
  - Example: Regularization (Lasso, L1 regularization)

Neural Network Architectures

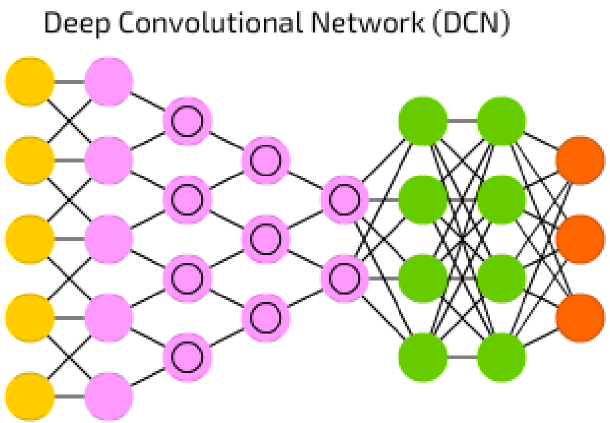
Feed-Forward Networks

- Neurons from each layer connect to neurons from next layer



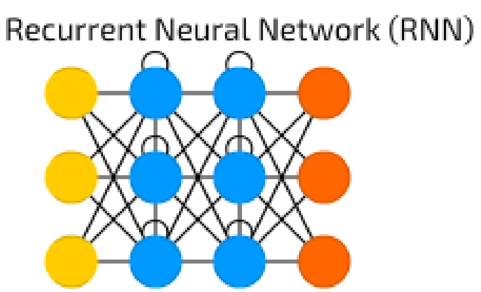
Convolutional Networks

- Includes convolution layer for feature reduction
- Learns hierarchical representations

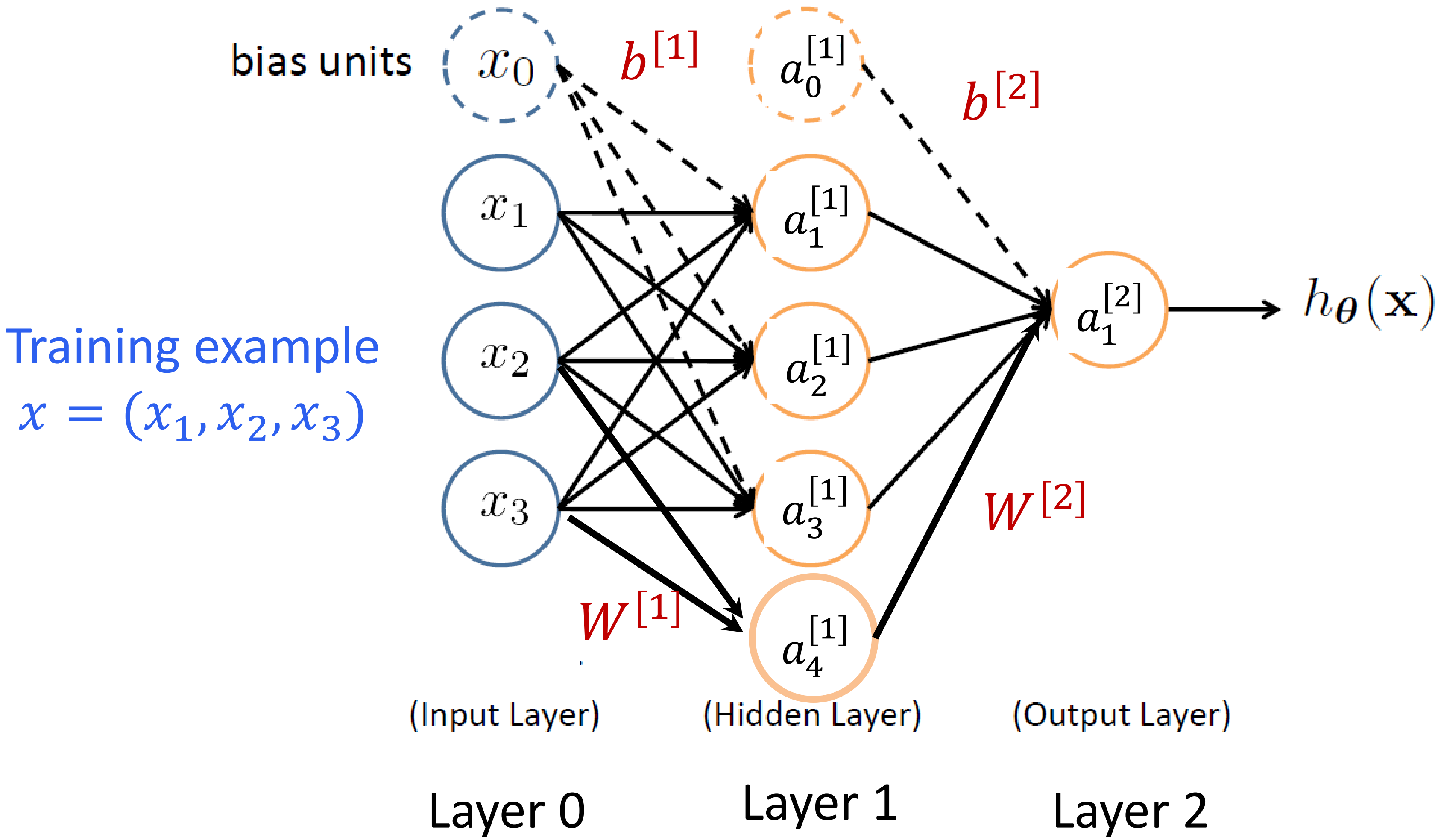


Recurrent Networks

- Keep hidden state
- Have cycles in computational graph

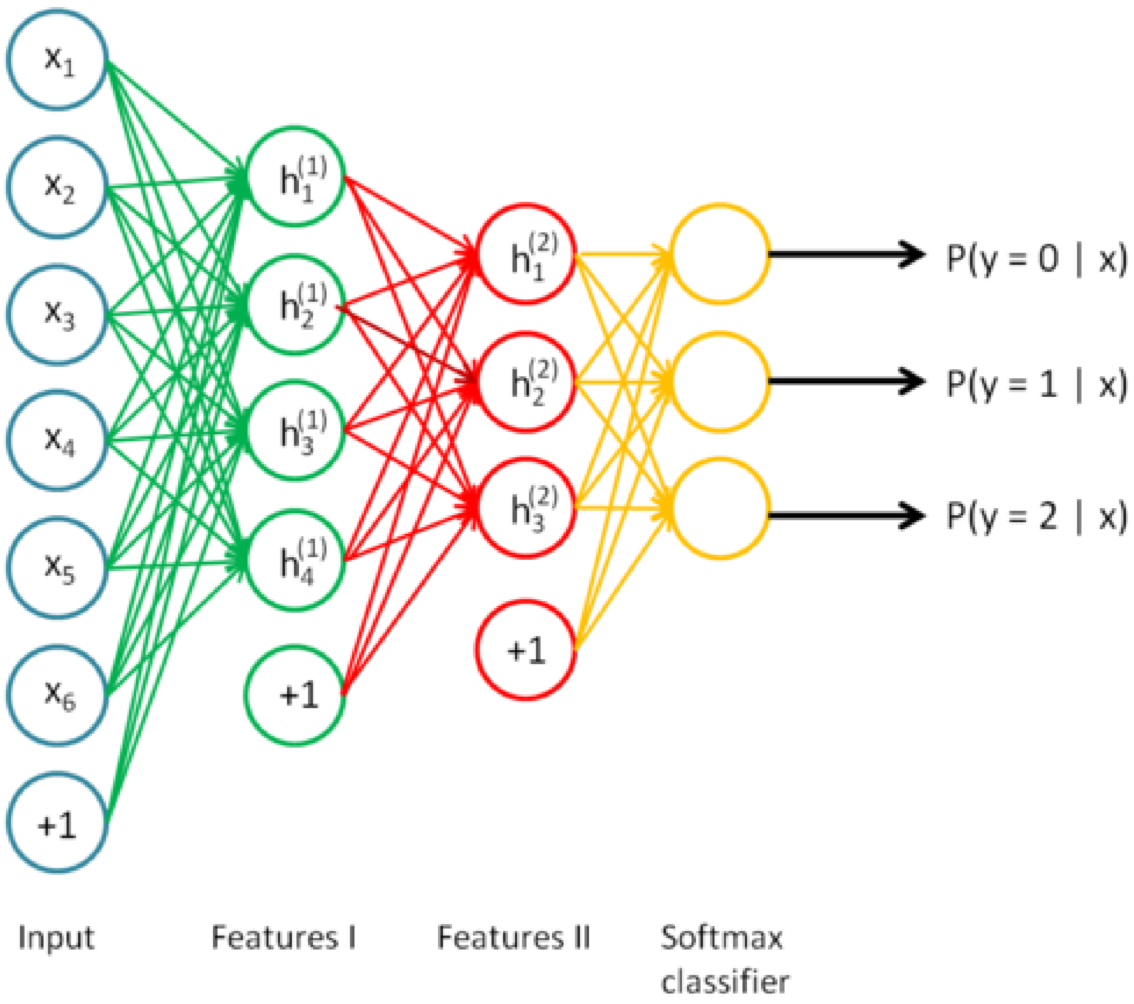


Feed-Forward Neural Network

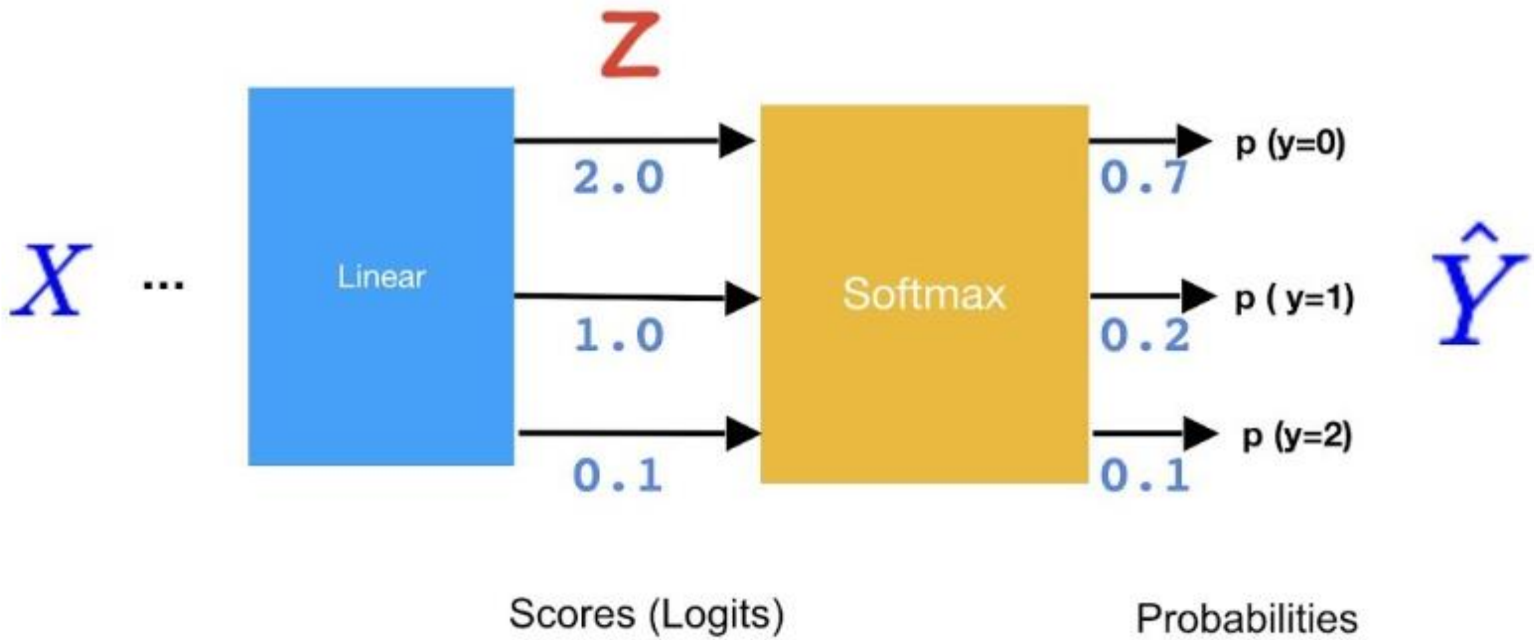


No cycles $\theta = (b^{[1]}, W^{[1]}, b^{[2]}, W^{[2]})$

Multi-class classification



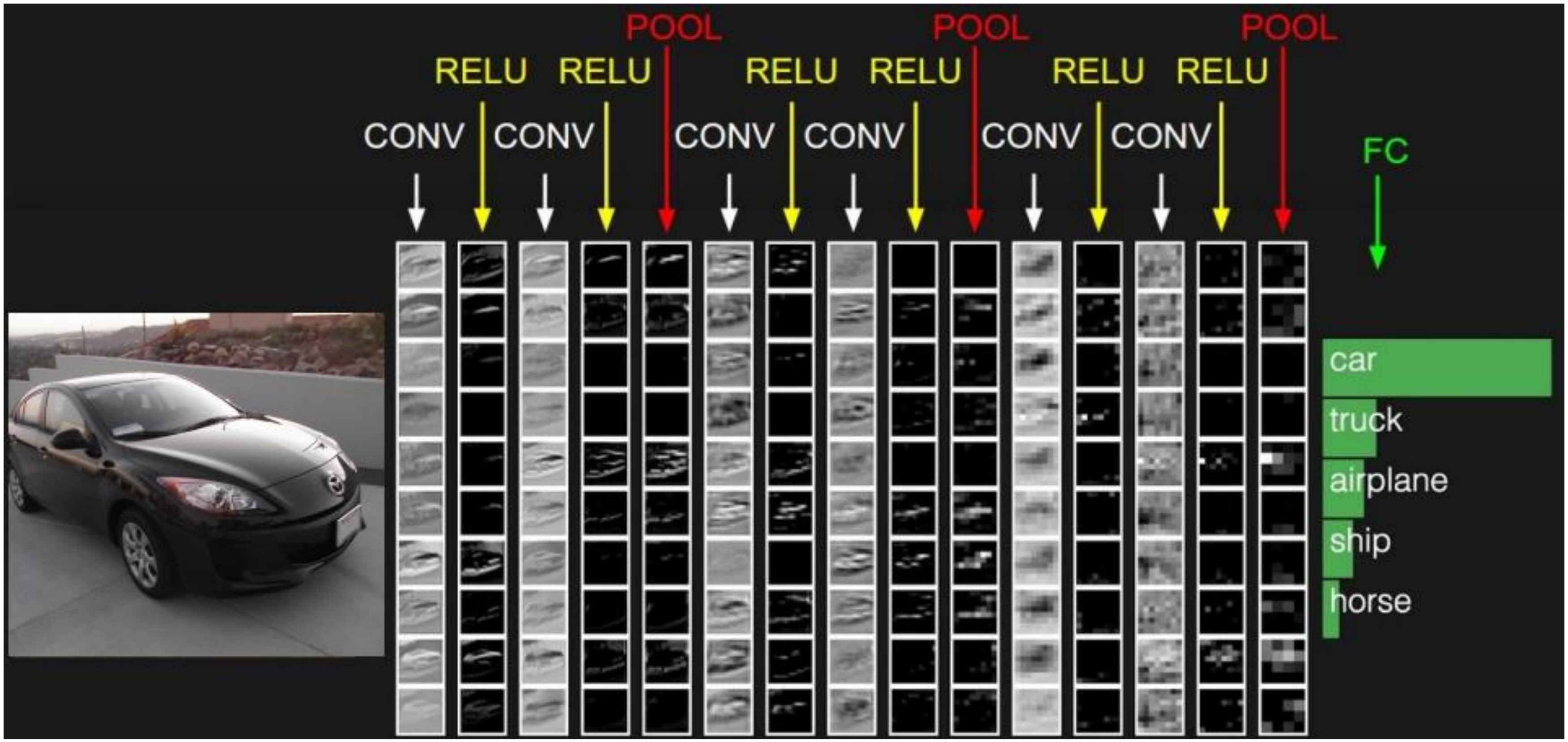
Softmax classifier



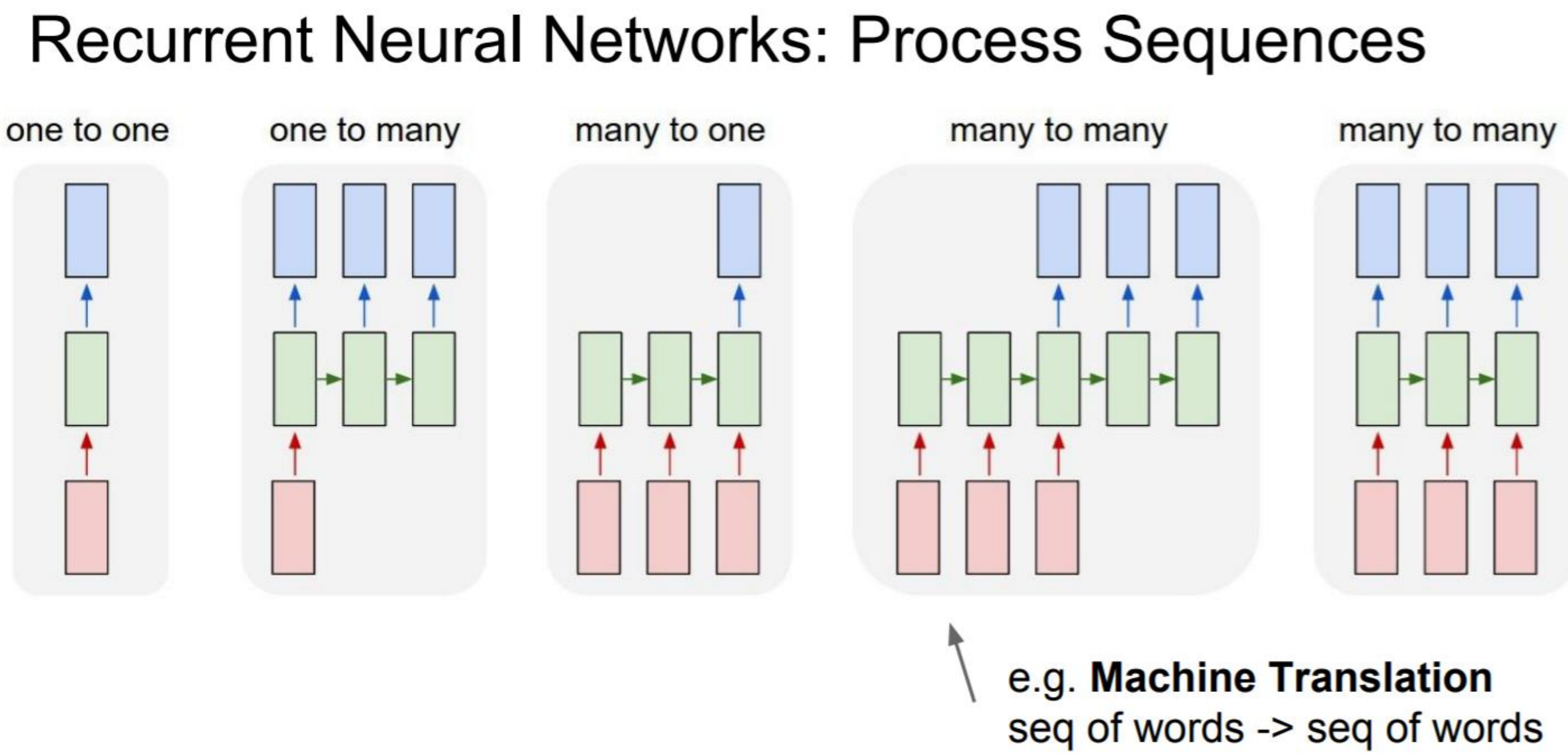
$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \quad \text{for } j = 1, \ldots, K.$$

- Predict the class with highest probability
- Generalization of sigmoid/logistic regression to multi-class

Convolutional Nets



RNN Architectures



Training NN with Backpropagation

Given training set $(x_1, y_1), \ldots, (x_N, y_N)$
Initialize all parameters $W^{[\ell]}, b^{[\ell]}$ randomly, for all layers ℓ
Loop

Set $\Delta_{ij}^{(l)} = 0 \quad \forall l, i, j$ (Used to accumulate gradient)
For each training instance $(x^{(i)}, y^{(i)})$
Set $\mathbf{a}^{(1)} = \mathbf{x}_i$
Compute $\{\mathbf{a}^{(2)}, \ldots, \mathbf{a}^{(L)}\}$ via forward propagation
Compute $\boldsymbol{\delta}^{(L)} = \mathbf{a}^{(L)} - y^{(i)}$
Compute errors $\{\boldsymbol{\delta}^{(L-1)}, \ldots, \boldsymbol{\delta}^{(2)}\}$
Compute gradients $\Delta_{ij}^{(l)} = \Delta_{ij}^{(l)} + a_j^{(l)} \delta_i^{(l+1)}$

EPOCH

Update weights via gradient step

- $W_{ij}^{[\ell]} = W_{ij}^{[\ell]} - \alpha \frac{\Delta_{ij}^{[\ell]}}{N}$
- Similar for $b_{ij}^{[\ell]}$

Until weights converge or maximum number of epochs is reached

Mini-batch Gradient Descent

- Initialization
  - For all layers ℓ
    - Set $W^{[\ell]}, b^{[\ell]}$ at random
- Backpropagation
  - Fix learning rate α
  - For all layers ℓ (starting backwards)
    - For all batches b of size B with training examples $x^{(ib)}, y^{(ib)}$

$$- W^{[\ell]} = W^{[\ell]} - \alpha \sum_{i=1}^{B} \frac{\partial L(\hat{y}^{(ib)}, y^{(ib)})}{\partial W^{[\ell]}}$$

$$- b^{[\ell]} = b^{[\ell]} - \alpha \sum_{i=1}^{B} \frac{\partial L(\hat{y}^{(ib)}, y^{(ib)})}{\partial b^{[\ell]}}$$

PCA

- We can apply these formulas to get the new representation for each instance $\mathbf{x}$

$$X = \begin{bmatrix} 0\ 1\ 0\ 1\ 1\ 0\ 0\ 1 \ldots \\ 1\ 1\ 0\ 1\ 1\ 1\ 0\ 0 \ldots \\ 0\ 0\ 1\ 1\ 1\ 0\ 0\ 0 \ldots \\ \vdots \\ 1\ 0\ 1\ 0\ 1\ 0\ 0\ 0 \ldots \end{bmatrix} \begin{matrix} \\ \\ \mathbf{x}_3 \\ \\ \\ \end{matrix} \quad \hat{Q} = \begin{bmatrix} 0.34 & 0.23 \\ 0.04 & 0.13 \\ -0.64 & 0.93 \\ \vdots & \vdots \\ -0.20 & -0.83 \end{bmatrix}$$

- The new 2D representation for $\mathbf{x}_3$ is given by:

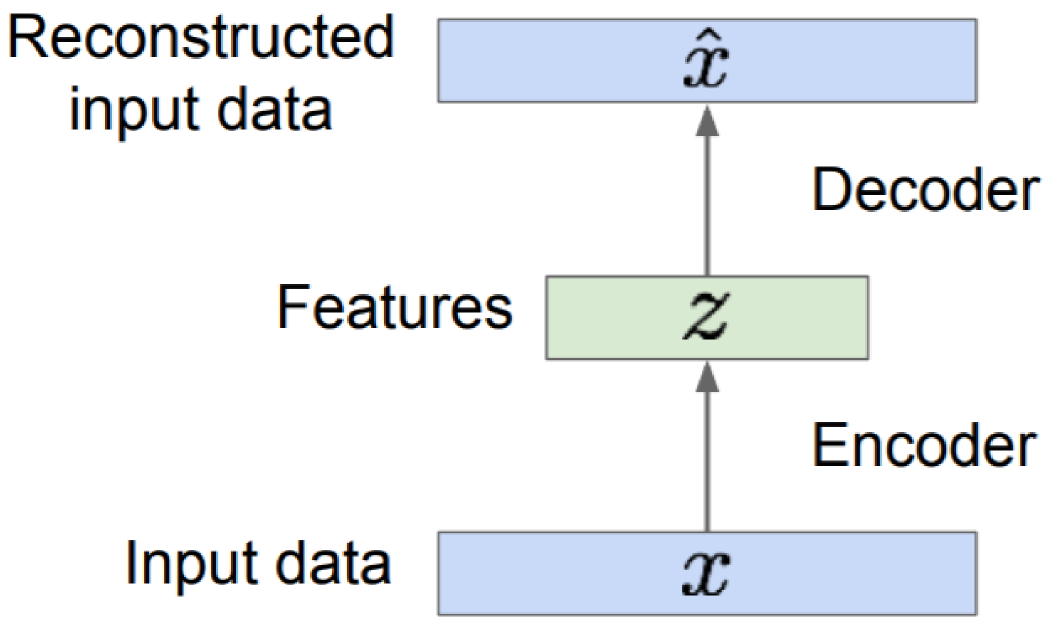
$$\hat{x}_{31} = 0.34(0) + 0.04(0) - 0.64(1) + \ldots$$

$$\hat{x}_{32} = 0.23(0) + 0.13(0) + 0.93(1) + \ldots$$

- The re-projected data matrix is given by $\hat{\mathrm{X}} = \mathrm{X}\hat{\mathrm{Q}}$

Autoencoders

How to learn this feature representation?
Train such that features can be used to reconstruct original data
"Autoencoding" - encoding itself



K means Algorithm

1. Initialization
   - Data are $\mathbf{x}_{1:N}$
   - Choose initial cluster means $\mathbf{m}_{1:k}$ (same dimension as data).
2. Repeat
   1. Assign each data point to its closest mean

$$z_n = \arg \min_{i \in \{1,\ldots,k\}} d(\mathbf{x}_n, \mathbf{m}_i)$$

   2. Compute each cluster mean to be the coordinate-wise average over data points assigned to that cluster,

$$\mathbf{m}_k = \frac{1}{N_k} \sum_{\{n : z_n = k\}} \mathbf{x}_n$$

3. Until assignments $\mathbf{z}_{1:N}$ do not change

Agglomerative clustering

- Algorithm:
  - Place each data point into its own singleton group (cluster)
  - Repeat
    - Iteratively merge *the two closest groups/clusters*
  - Until: stopping condition is satisfied
- Output
  - Set of clusters
  - Dendrogram (tree of how data was merged)
- Need to define distance or similarity between groups


THANK
YOU