Machine Learning

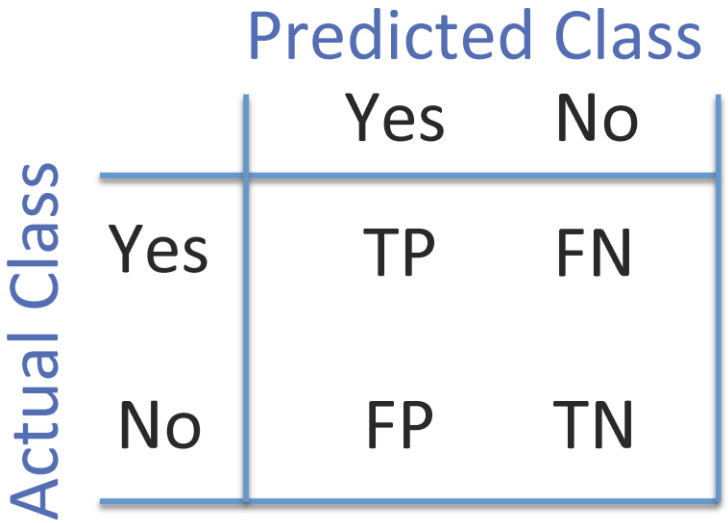
CS 539

Worcester Polytechnic Institute
Department of Computer Science
Instructor: Prof. Kyumin Lee

Advanced Evaluation Metrics

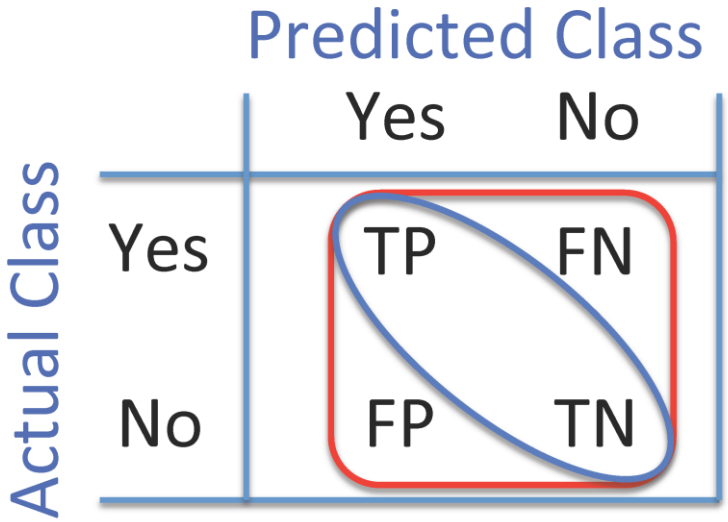
Confusion Matrix

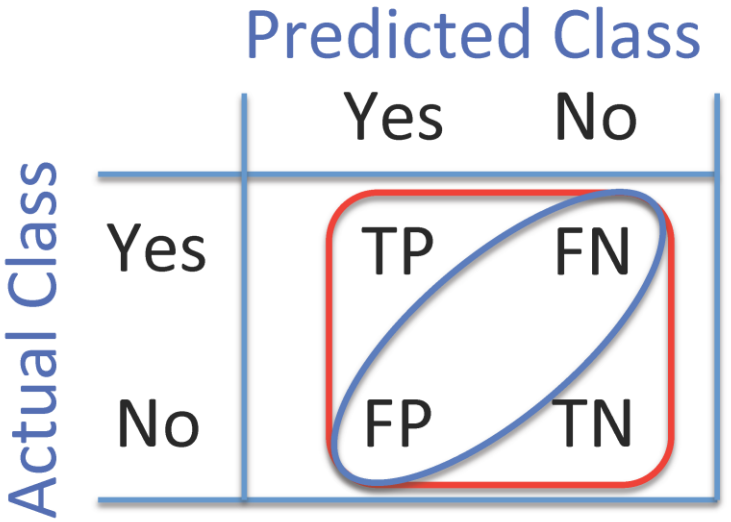
Given a dataset of P positive instances and N negative instances:



Accuracy & Error

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





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

$$\text{error} = 1 - \frac{TP + TN}{P + N}$$

$$= \frac{FP + FN}{P + N}$$

Precision & Recall

Precision

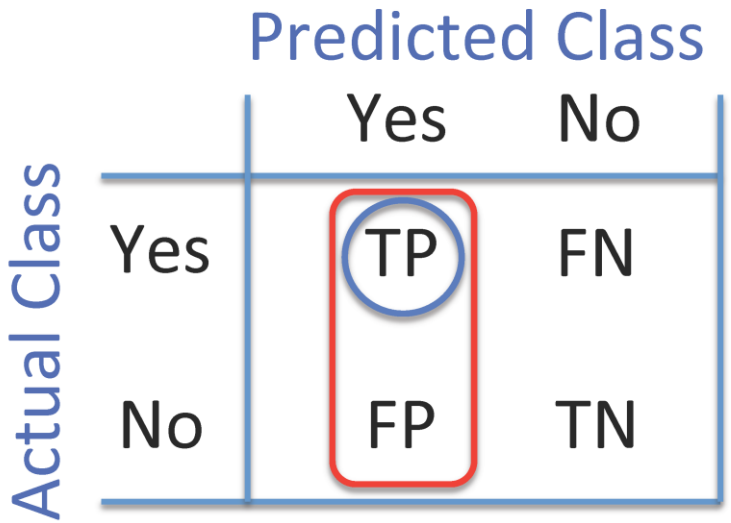
- the fraction of positive predictions that are correct
- P(is pos|predicted pos)

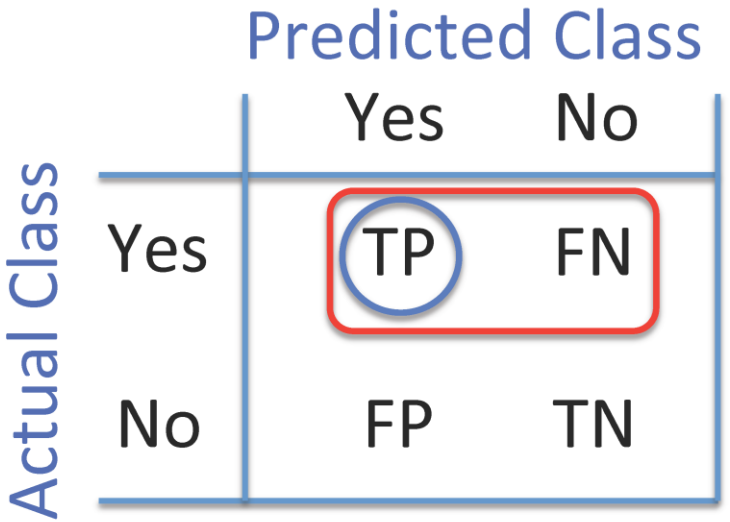
$$\text{precision} = \frac{TP}{TP + FP}$$

Recall

- fraction of positive instances that are identified
- P(predicted pos|is pos)

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





Precision & Recall

Precision

- the fraction of positive predictions that are correct
- P(is pos|predicted pos)

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

Recall

- fraction of positive instances that are identified
- P(predicted pos|is pos)

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

- You can get high recall (but low precision) by only predicting positive
- Recall is a non-decreasing function of the # positive predictions
- Typically, precision decreases as either the number of positive predictions or recall increases
- Precision & recall are widely used in information retrieval

F-Measure

- Combined measure of precision/recall tradeoff

$$F_1 = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}}$$

  - This is the harmonic mean of precision and recall
  - In the F_1 measure, precision and recall are weighted evenly
  - Can also have biased weightings that emphasize either precision or recall more (F_2 = 2 × recall; $F_{0.5}$ = 2 × precision)

A Word of Caution

- Consider binary classifiers A, B, C:

	A	.	B	.	C	.
	1	0	1	0	1	0
Predictions 1	0.9	0.1	0.8	0	0.78	0
Predictions 0	0	0	0.1	0.1	0.12	0.1

- Clearly A is useless, since it always predicts 1
- B is slightly better than C
- But, here are the performance metrics:

Metric	A	B	C
Accuracy	0.9	0.9	0.88
Precision	0.9	1.0	1.0
Recall	1.0	0.888	0.8667
F-score	0.947	0.941	0.9286

Receiver Operating Characteristic (ROC)

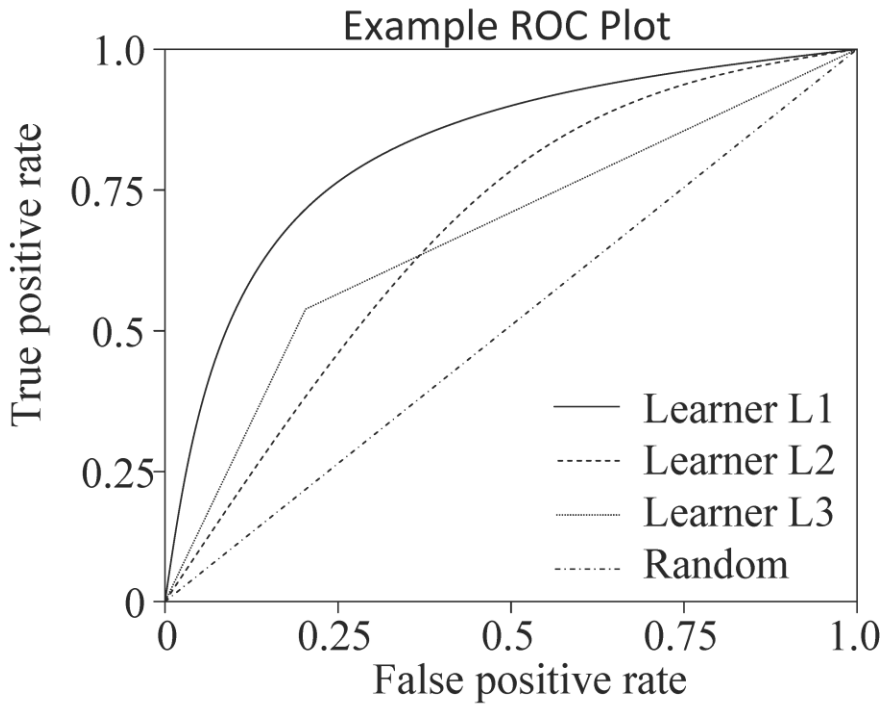
ROC curves assess predictive behavior independent of error costs or class distributions

- Originated from signal detection theory
- Common in medical diagnosis, now used for ML

Plots TP rate vs FP Rate

$$\text{TP rate} = \text{TP}/\text{P}$$
$$\text{FP rate} = \text{FP}/\text{N}$$

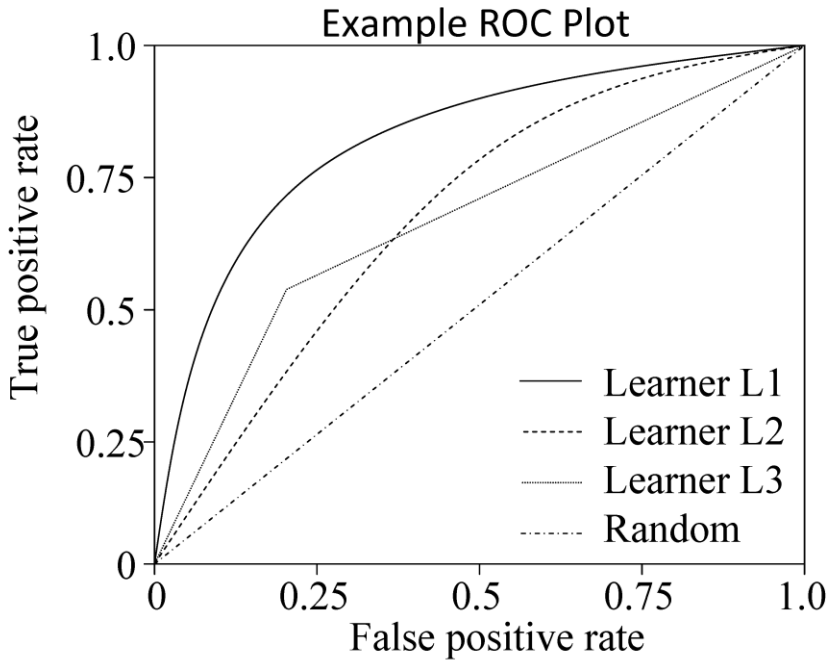
Actual Class \ Predicted Class	Yes	No
Yes	TP	FN
No	FP	TN



Performance Depends on Threshold

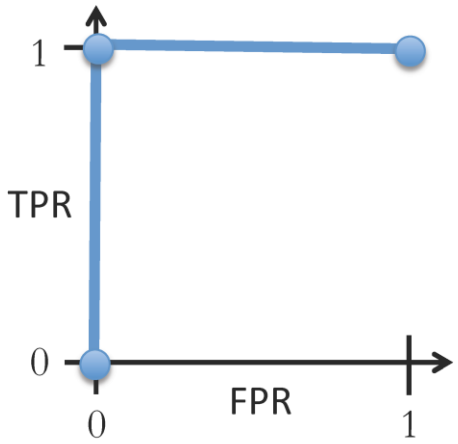
Predict positive if $\mathrm{P}(y = 1 \mid \mathbf{x}) > \theta$, otherwise negative

- Number of TPs and FPs depend on threshold θ
- As we vary θ, we get different (TPR, FPR) points



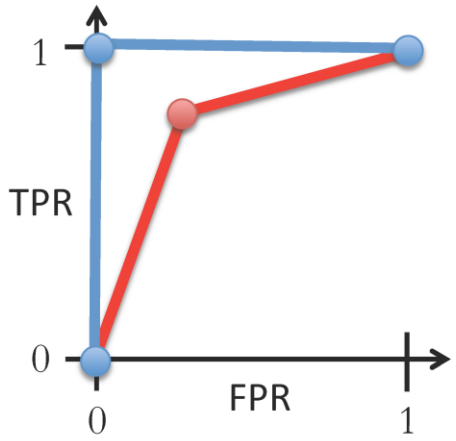
ROC Example

i	y_i	$p(y_i = 1 \mid \mathbf{x}_i)$	$h(\mathbf{x_i} \mid \theta = 0)$	$h(\mathbf{x_i} \mid \theta = 0.5)$	$h(\mathbf{x_i} \mid \theta = 1)$
1	1	0.9	1	1	0
2	1	0.8	1	1	0
3	1	0.7	1	1	0
4	1	0.6	1	1	0
5	1	0.5	1	1	0
6	0	0.4	1	0	0
7	0	0.3	1	0	0
8	0	0.2	1	0	0
9	0	0.1	1	0	0
			$TPR = 5/5 = 1$	$TPR = 5/5 = 1$	$TPR = 0/5 = 0$
			$FPR = 4/4 = 1$	$FPR = 0/4 = 0$	$FPR = 0/4 = 0$

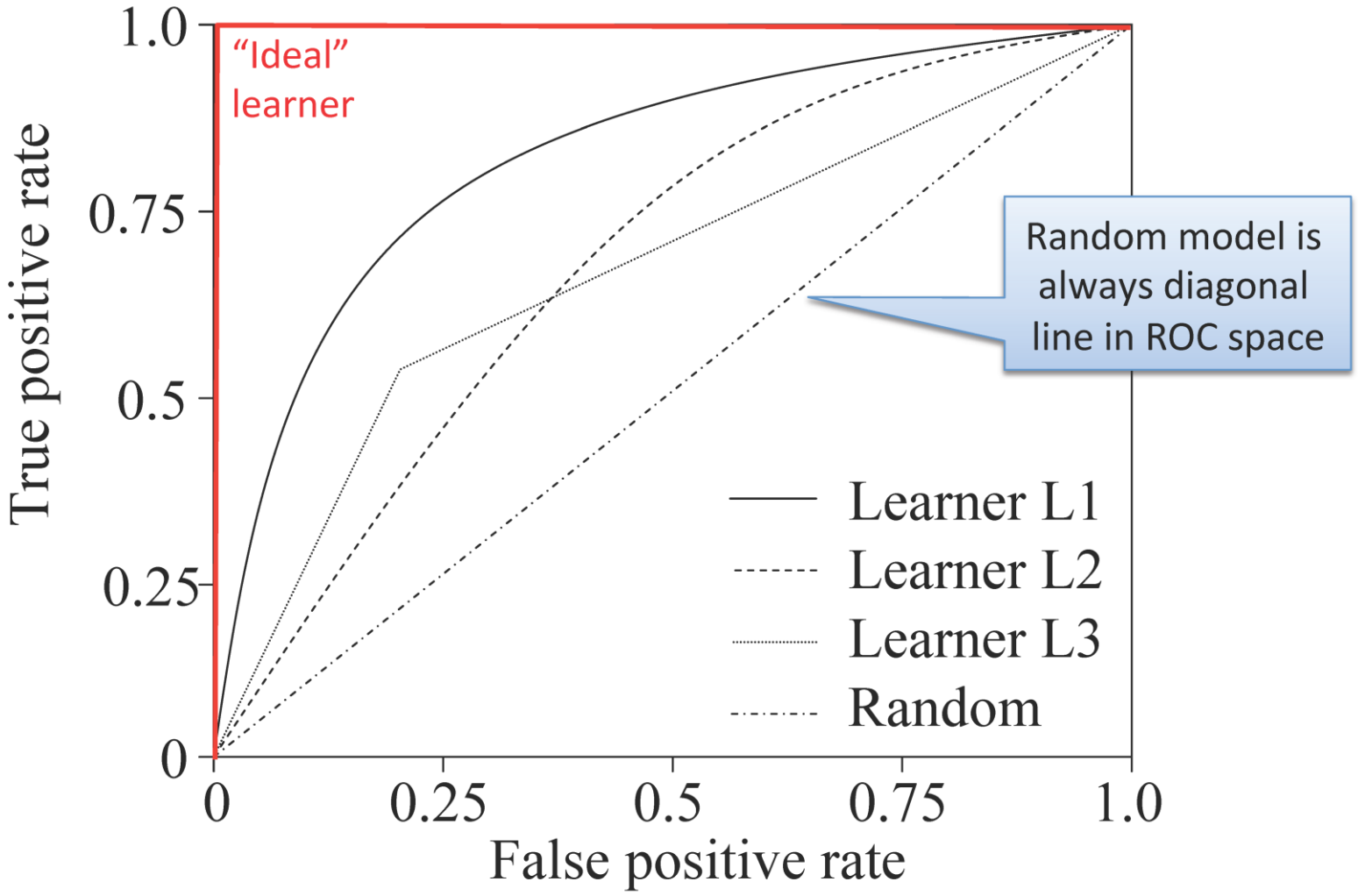


ROC Example

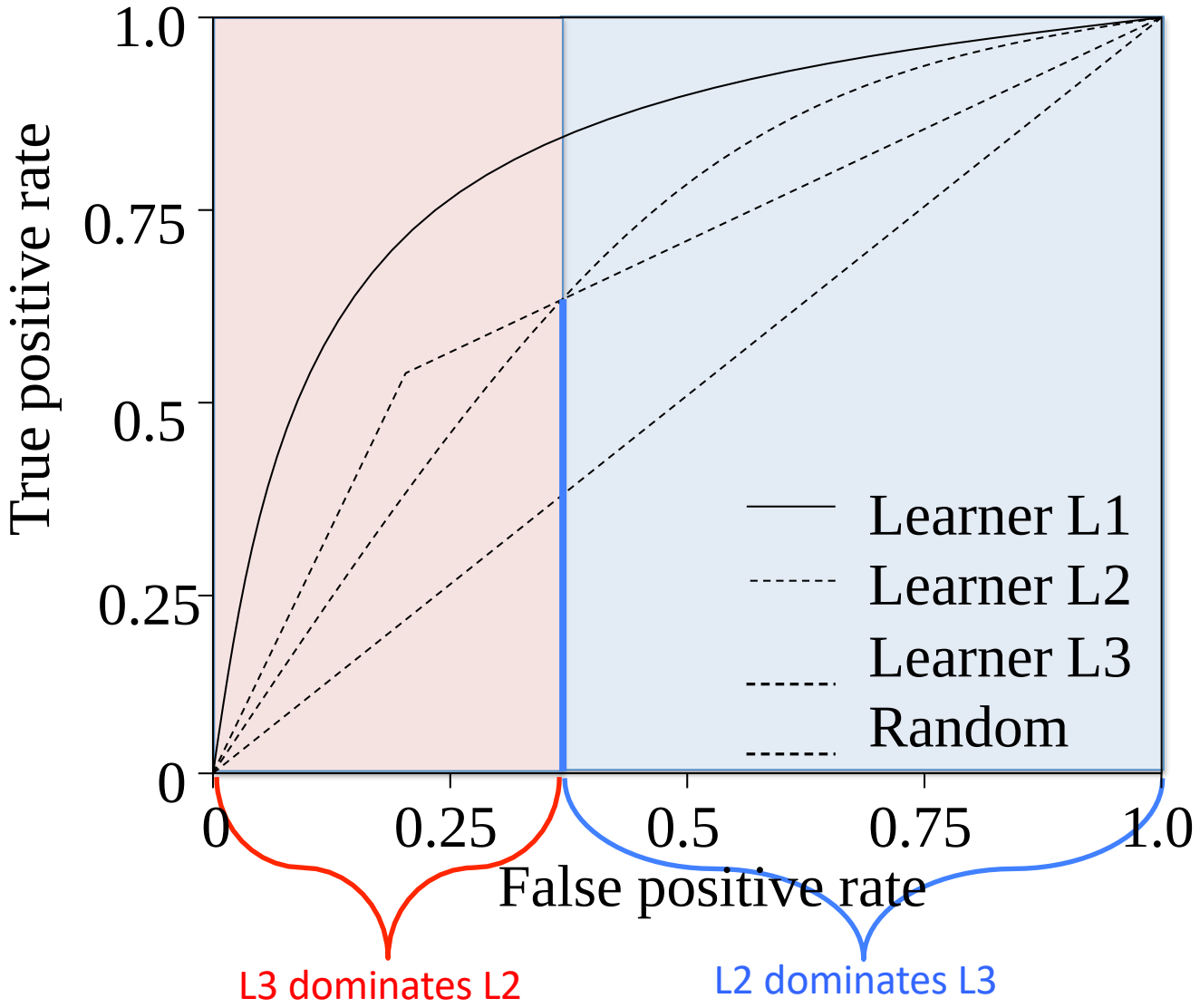
i	y_i	$p(y_i = 1 \mid \mathbf{x}_i)$	$h(\mathbf{x_i} \mid \theta = 0)$	$h(\mathbf{x_i} \mid \theta = 0.5)$	$h(\mathbf{x_i} \mid \theta = 1)$
1	1	0.9	1	1	0
2	1	0.8	1	1	0
3	1	0.7	1	1	0
4	1	0.6	1	1	0
5	1	**0.2**	1	**0**	0
6	0	**0.6**	1	**1**	0
7	0	0.3	1	0	0
8	0	0.2	1	0	0
9	0	0.1	1	0	0
			$TPR = 5/5 = 1$	$TPR = 4/5 = 0.8$	$TPR = 0/5 = 0$
			$FPR = 4/4 = 1$	$FPR = 1/4 = 0.25$	$FPR = 0/4 = 0$



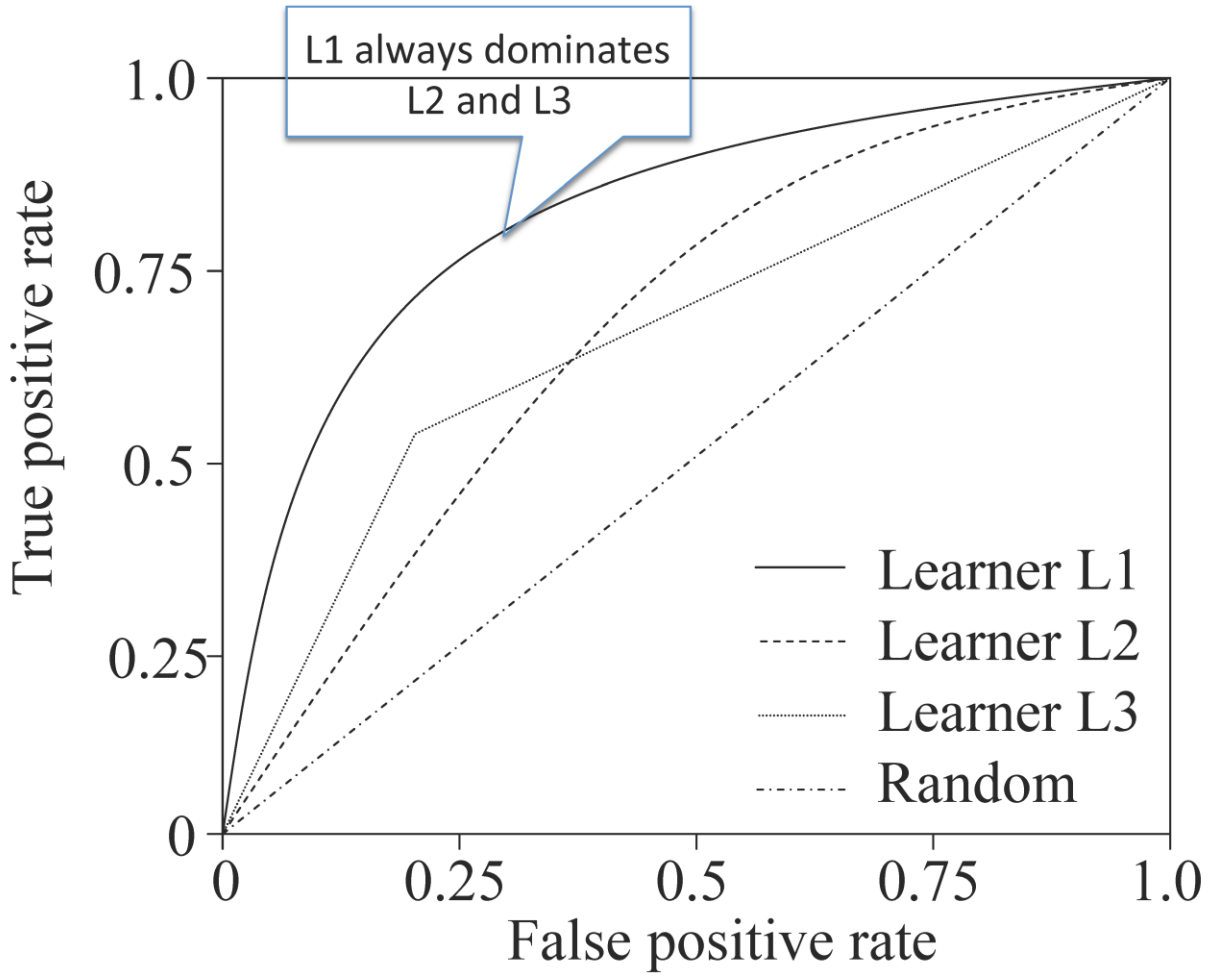
Receiver Operating Characteristic (ROC)



Receiver Operating Characteristic (ROC)

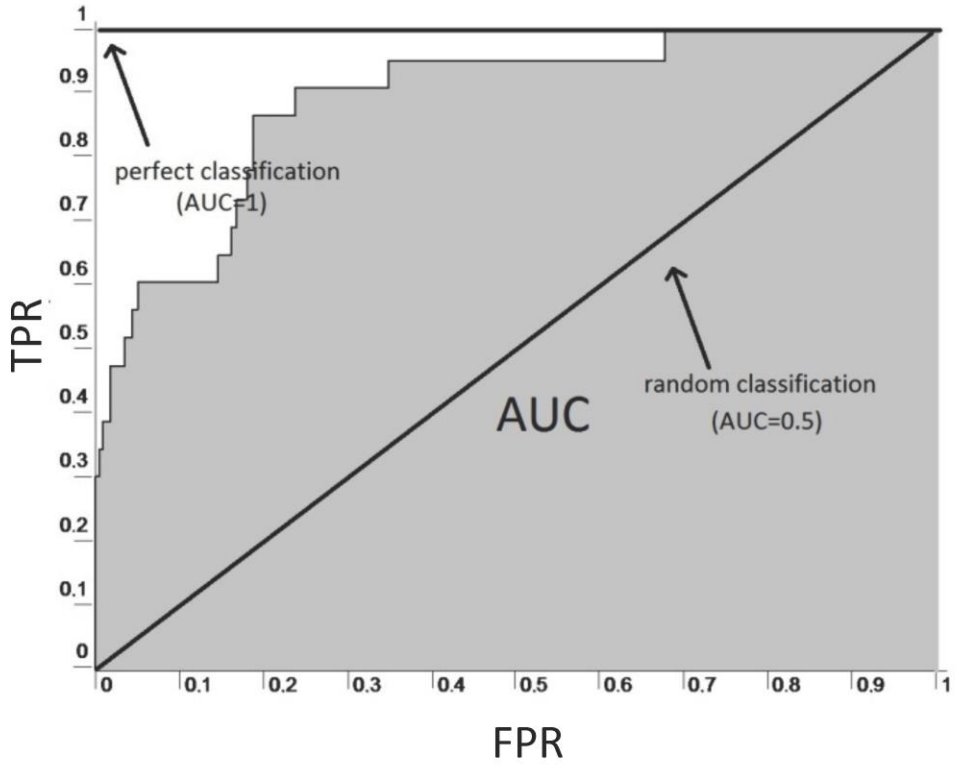


Receiver Operating Characteristic (ROC)



Area Under the ROC Curve (AUC)

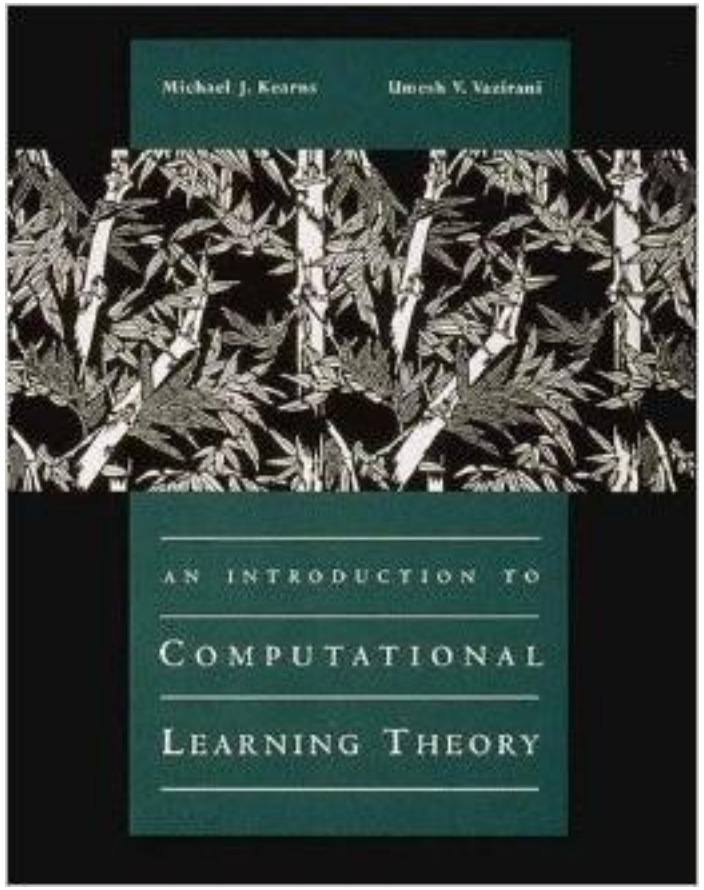
- Can take area under the ROC curve to summarize performance as a single number



Computational Learning Theory: Why ML Work

Computational Learning Theory

Entire subfield devoted to the mathematical analysis of machine learning algorithms



Annual conference: Conference on Learning Theory (COLT)

Computational Learning Theory

Fundamental Question: What general laws constrain inductive learning (i.e., learning from examples)?

Seeks theory to relate:

- Probability of successful learning
- Number of training examples
- Complexity of hypothesis space
- Accuracy to which target function is approximated
- Manner in which training examples should be presented

Sample Complexity = # of examples

Assume that $f : \mathcal{X} \mapsto \{0, 1\}$ is the target concept

How many training examples are sufficient to learn the target concept f ?

1. If learner proposed instances as queries to teacher
   - Learner proposes instance x, teacher provides $f(x)$ active learning
2. If teacher (who knows f) provides training examples passive learning
   - Teacher provides labeled examples in form $<x, f(x)>$
3. If some random process (e.g., nature) proposes instances
   - Instance x generated randomly, teacher provides $f(x)$ online learning

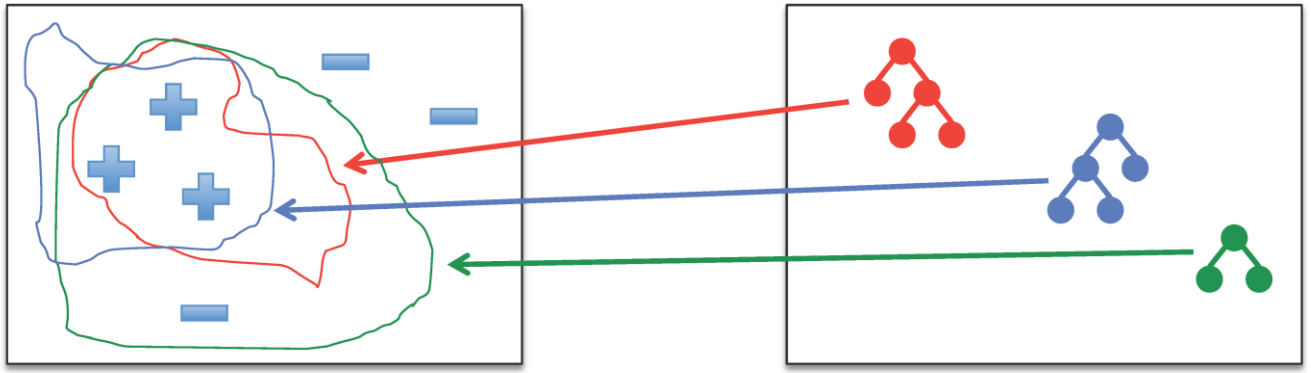
Function Approximation: The Big Picture

Instance Space $\mathcal{X} = \{0,1\}^d$

$\boldsymbol{x} = \langle x_1, x_2, \ldots, x_d \rangle \in \mathcal{X}$

Hypothesis Space

$H = \{h \mid h : \mathcal{X} \mapsto \{0,1\}\}$

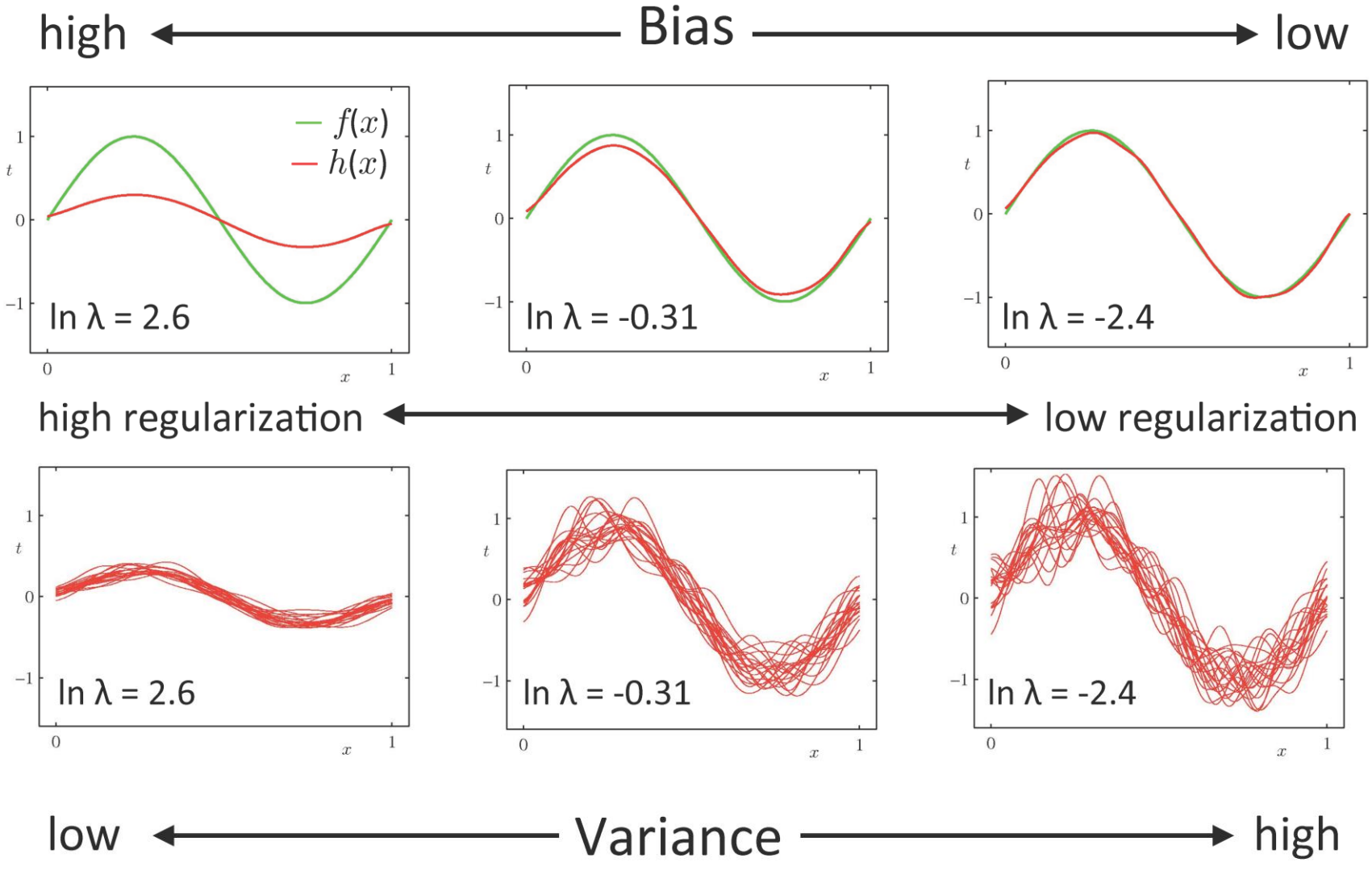


if $d = 20$, $|\mathcal{X}| = 2^{20}$

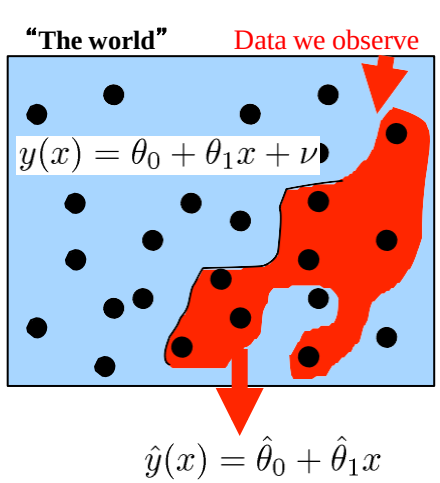
$|h| = 2^{|\mathcal{X}|} = 2^{2^{20}}$

- How many labeled instances are needed to determine which of the $2^{2^{20}}$ hypotheses are correct?
  - All 2^{20} instances in $\mathcal{X}$ must be labeled!
- Generalizing beyond the training data (inductive inference) is impossible unless we add more assumptions (e.g., priors over H)
- There is no free lunch!

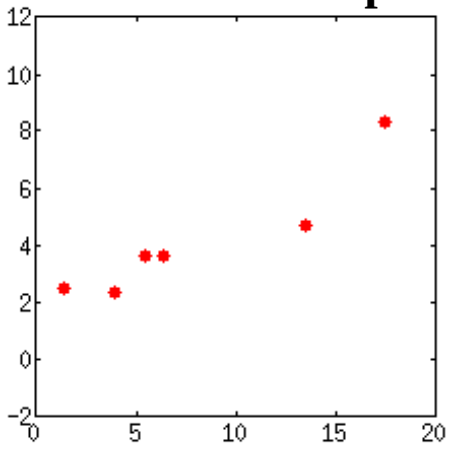
Illustration of Bias-Variance

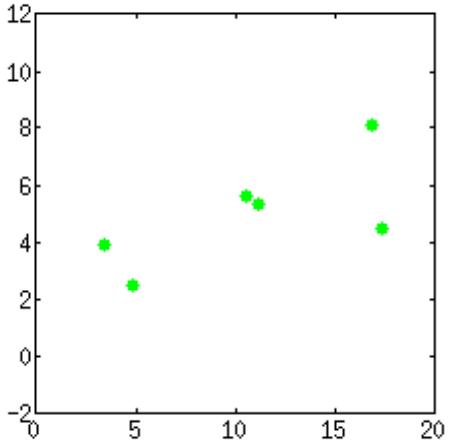


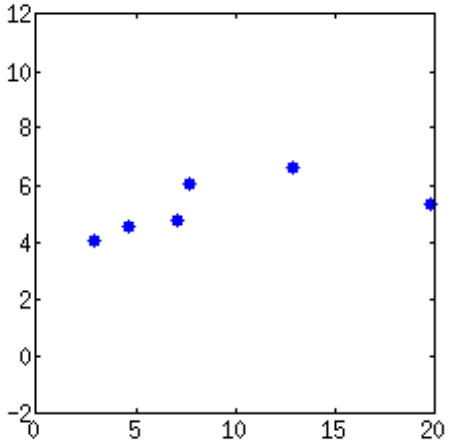
Bias & variance



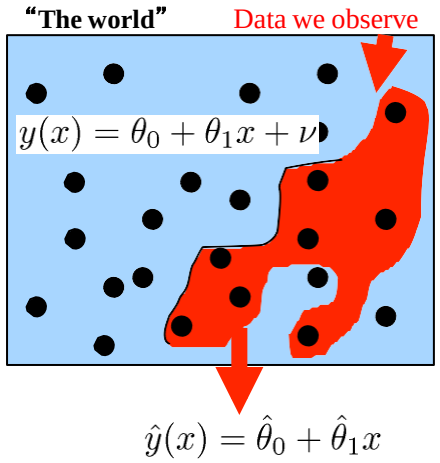
Three different possible data sets:



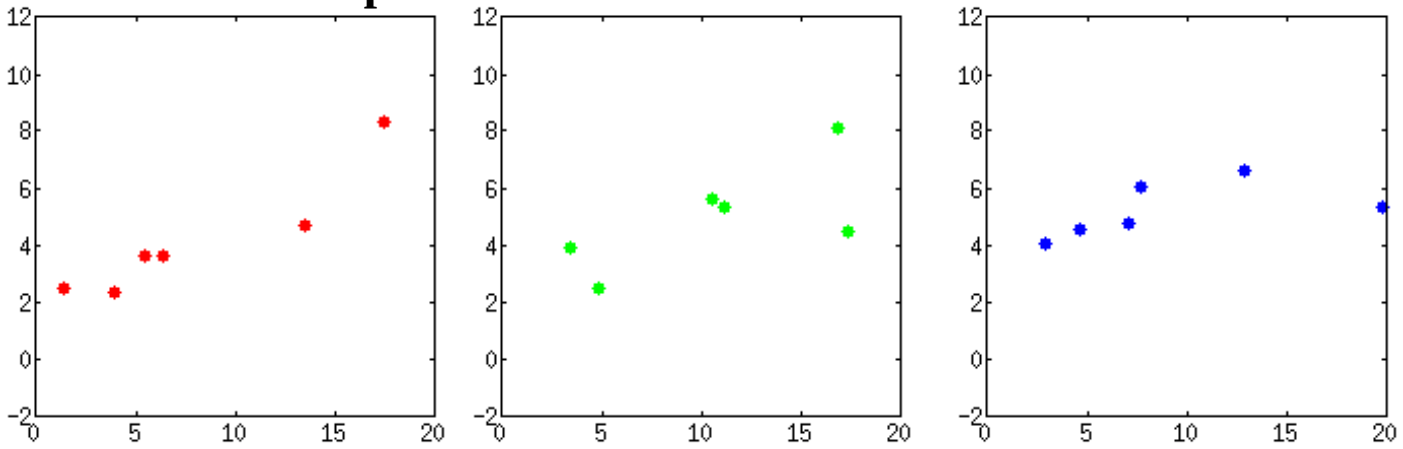




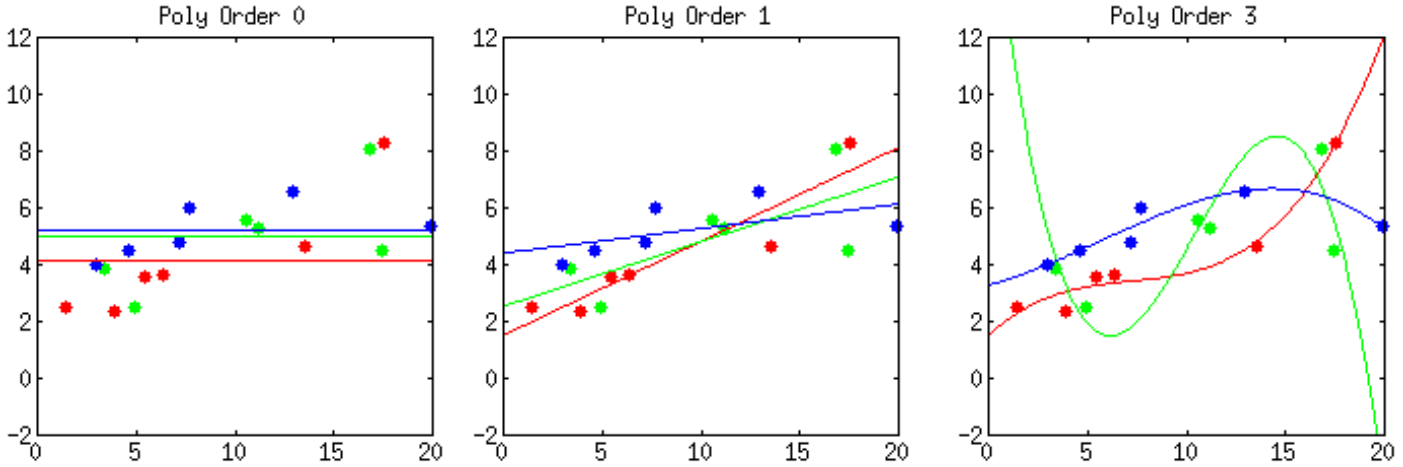
Bias & variance



Three different possible data sets:



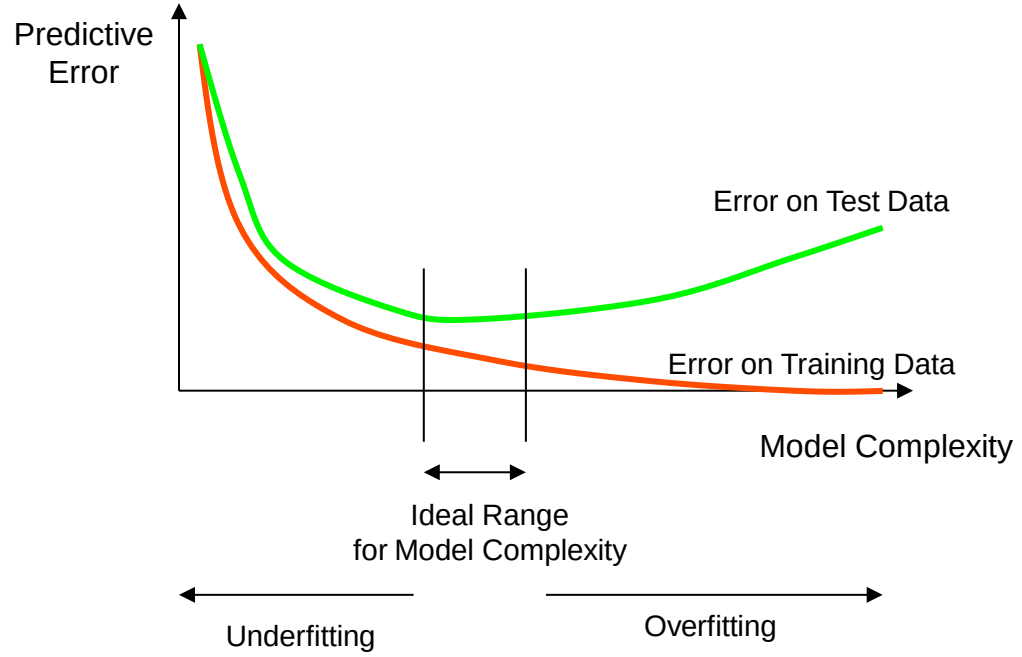
Each would give different predictors for any polynomial degree:



What to do about under/overfitting?

- Ways to increase complexity?
  - Add features, parameters

- Ways to decrease complexity?
  - Remove features (“feature selection”)
  - Regularization



A Way to Choose the Best Model

- It would be <u>really</u> helpful if we could get a guarantee of the following form:

$$\text{testingError} \leq \text{trainingError} + f(n, h, p)$$

n = size of training set

h = measure of the model complexity

p = the probability that this bound fails

We need p to allow for really unlucky test sets

- Then, we could choose the model complexity that minimizes the bound on the test error

A Measure of Model Complexity

- Suppose that we pick n data points and assign labels of + or – to them at random
- If our model class (e.g., a decision tree, polynomial regression of a particular degree, etc.) can learn any association of labels with data, it is too powerful!

> More power: can model more complex functions, but may overfit
>
> Less power: won't overfit, but limited in what it can represent

- **Idea**: characterize the power of a model class by asking how many data points it can learn perfectly for all possible assignments of labels
  - This number of data points is called the Vapnik-Chervonenkis (VC) dimension

VC Dimension

- A measure of the power of a particular class of models
  - It does not depend on the choice of training set
- The VC dimension of a model class is the maximum number of points that can be arranged so that the class of models can shatter

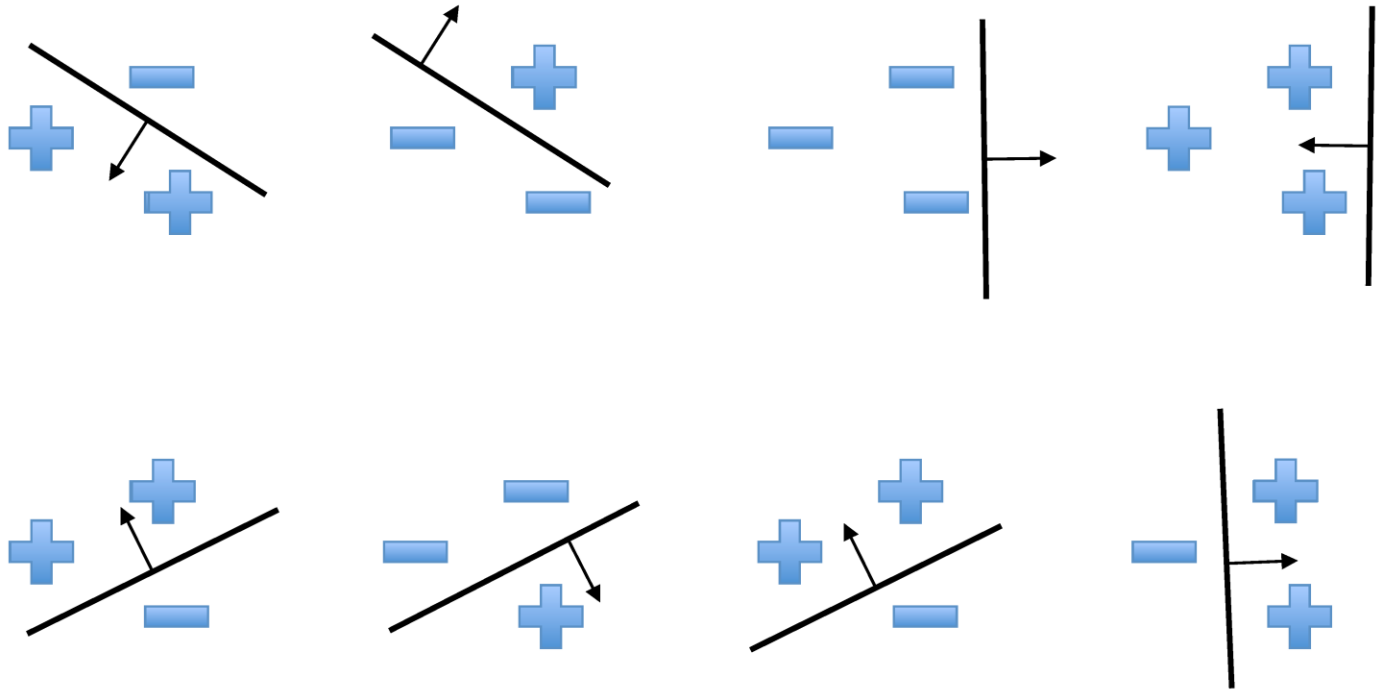
Definition: a model class can shatter a set of points

$$\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \ldots, \boldsymbol{x}^{(r)}$$

if for every possible labeling over those points, there exists a model in that class that obtains zero training error

An Example of VC Dimension

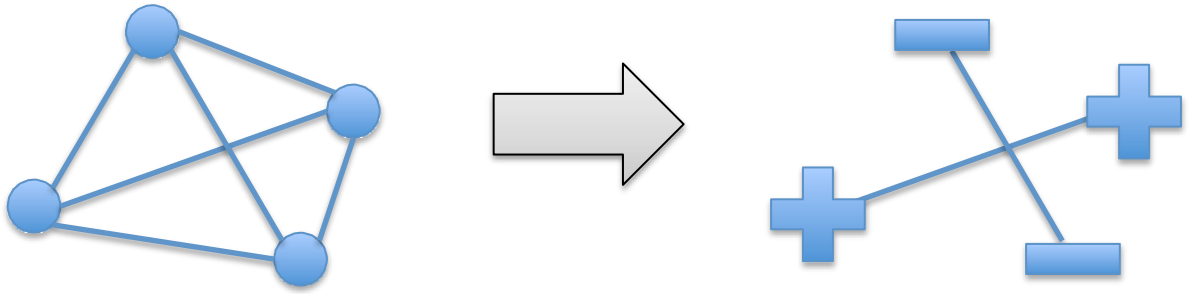
- Suppose our model class is a hyperplane (e.g., perceptron)
- Consider all labelings over three points in $\mathbb{R}^2$



- In $\mathbb{R}^2$, we can find a plane (i.e., a line) to capture any labeling of 3 points. A 2D hyperplane shatters 3 points

An Example of VC Dimension

- But, a 2D hyperplane cannot deal with some labelings of four points:



Connect all pairs of points; two lines will always cross

Can't separate points if the pairs that cross are the same class

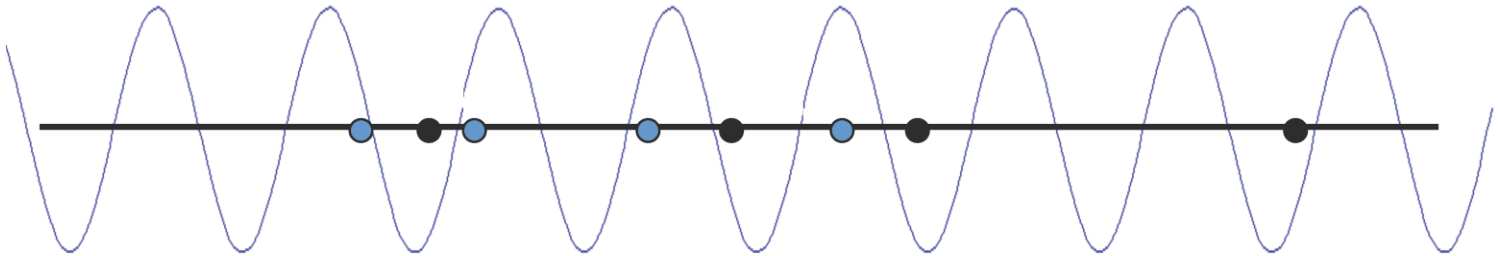
- Therefore, a 2D hyperplane cannot shatter 4 points

Some Examples of VC Dimension

- The VC dimension of a hyperplane in 2D is 3.
  - In d dimensions it is d+1
    - It's just a coincidence that the VC dimension of a hyperplane is almost identical to the # parameters needed to define a hyperplane

- A sine wave has infinite VC dimension and only 2 parameters!
  - By choosing the phase & period carefully we can shatter any random set of 1D data points (except for nasty special cases)

$$h(x) = a\sin(bx)$$



Assumptions

- Given some model class (which defines the hypothesis space H)
- Assume all training points were drawn i.i.d from distribution $\mathcal{D}$
- Assume all future test points will be drawn from $\mathcal{D}$

Definitions:

$$R(\boldsymbol{\theta}) = \text{testError}(\boldsymbol{\theta}) = E\left[\frac{1}{2}|y - h_{\boldsymbol{\theta}}(\boldsymbol{x})|\right]$$

"official" notation

notation we'll use

probability of misclassification

$$R^{\text{emp}}(\boldsymbol{\theta}) = \text{trainError}(\boldsymbol{\theta}) = \frac{1}{n}\sum_{i=1}^{n}\frac{1}{2}\left|y^{(i)} - h_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)})\right|$$

A Probabilistic Guarantee of Generalization Performance

Vapnik showed that with probability $(1 - \eta)$:

$$\text{testError}(\boldsymbol{\theta}) \leq \text{trainError}(\boldsymbol{\theta}) + \sqrt{\frac{h(\log(2n/h) + 1) - \log(\eta/4)}{n}}$$

n = size of training set

h = VC dimension of model class

η = the probability that this bound fails

- So, we should pick the model with the complexity that minimizes this bound
  - The theory provides insight, but in practice we still need some magic

Take Away Lesson

Suppose we find a model with a low training error...

- If the following holds:
  - H is sufficiently constrained in size
  - and/or the size of the training data set n is large,

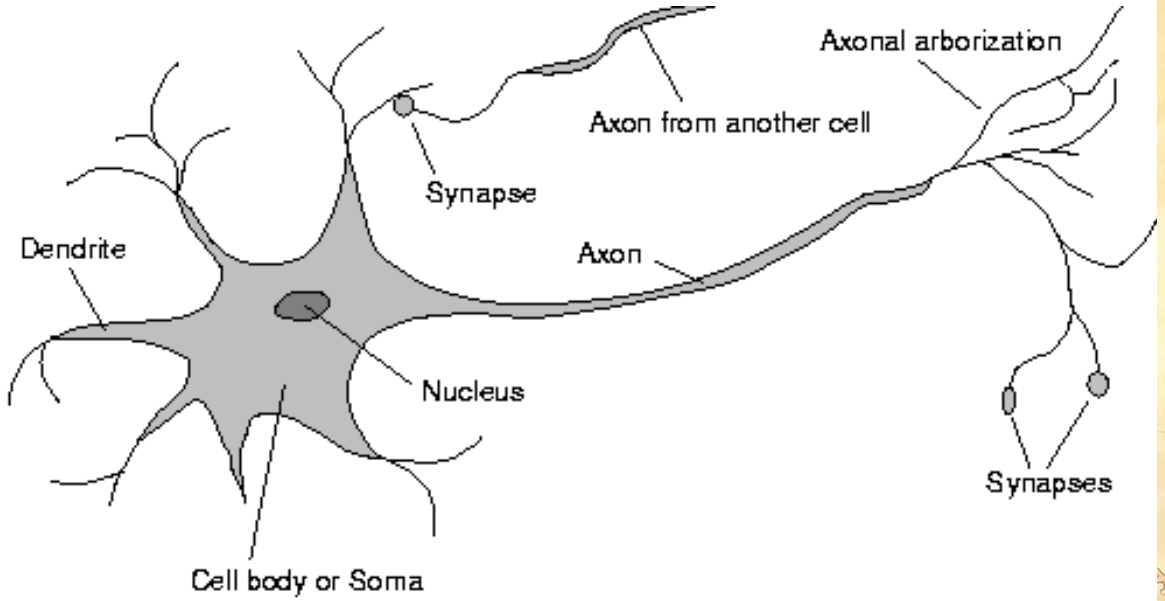
  then low training error is likely to be evidence of low generalization error

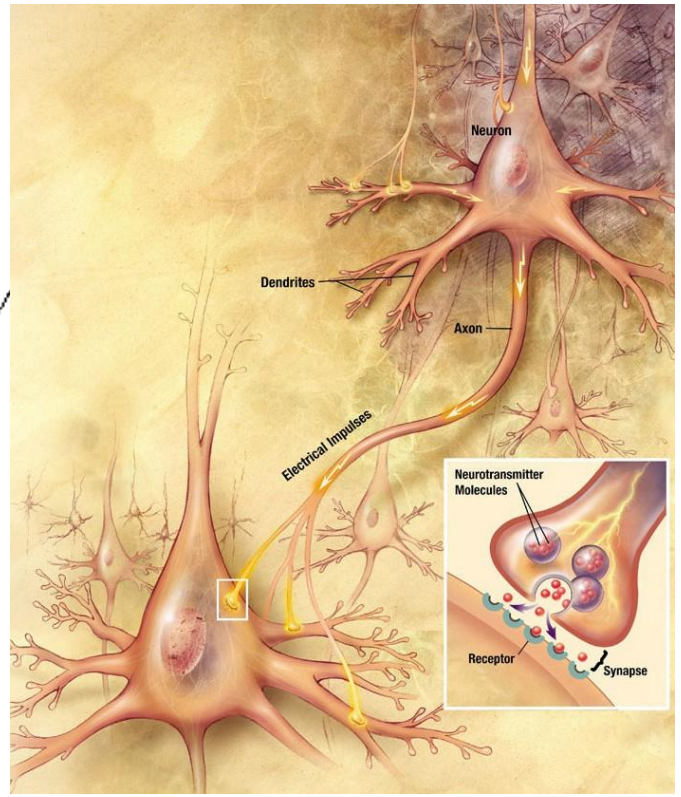
Neural Networks

Neural Function

- Brain function (thought) occurs as the result of the firing of **neurons**
- Neurons connect to each other through **synapses**, which propagate **action potential** (electrical impulses) by releasing **neurotransmitters**
  - Synapses can be **excitatory** (potential-increasing) or **inhibitory** (potential-decreasing), and have varying **activation thresholds**
  - Learning occurs as a result of the synapses' **plasticicity**: They exhibit long-term changes in connection strength
- There are about 10^{11} neurons and about 10^{14} synapses in the human brain!

Biology of a Neuron

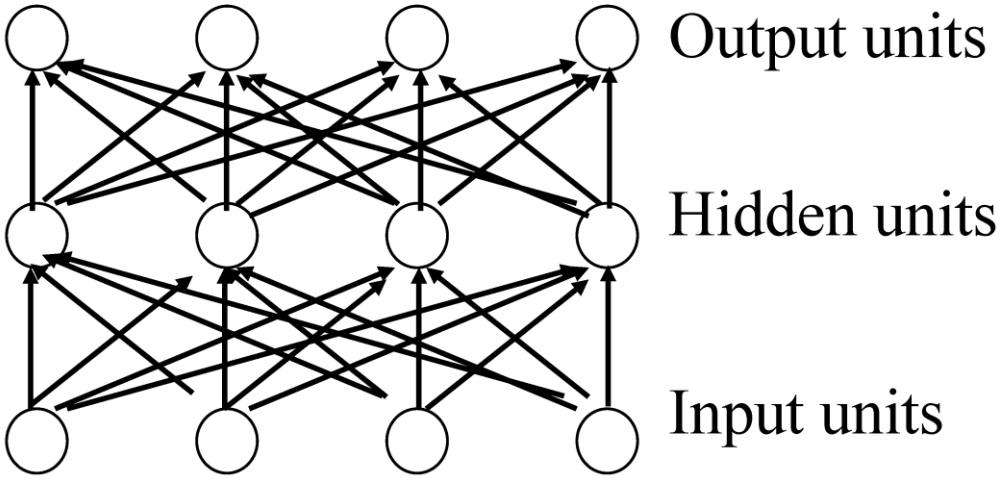




Neural Networks

- Origins: Algorithms that try to mimic the brain.
- Very widely used in 80s and early 90s; popularity diminished in late 90s.
- Recent resurgence: State-of-the-art technique for many applications
- Artificial neural networks are not nearly as complex or intricate as the actual brain structure

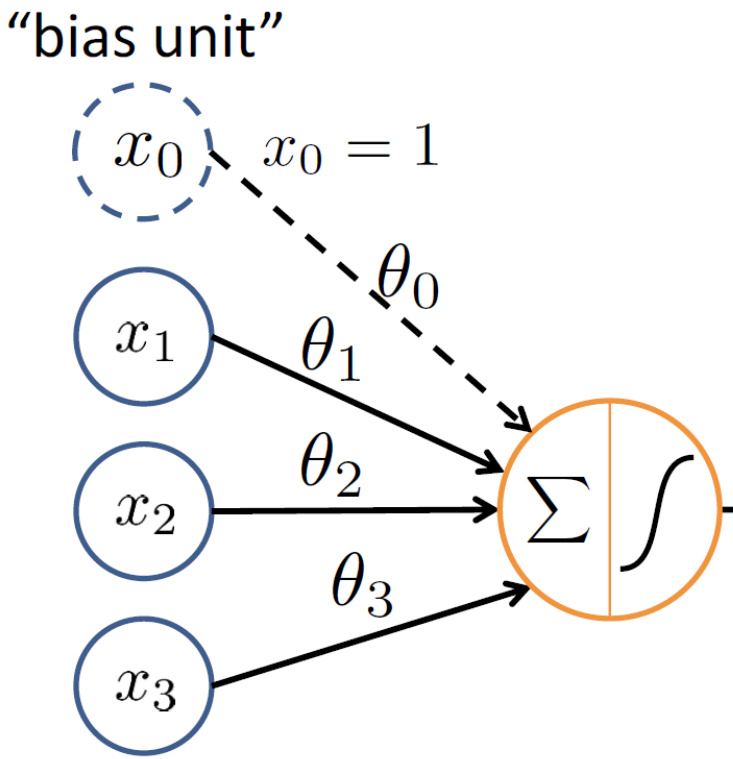
Neural Networks



Layered feed-forward network

- Neural networks are made up of **nodes** or **units**, connected by **links**
- Each link has an associated **weight** and **activation level**
- Each node has an **input function** (typically summing over weighted inputs), an **activation function**, and an **output**

Neuron Model: Logistic Unit

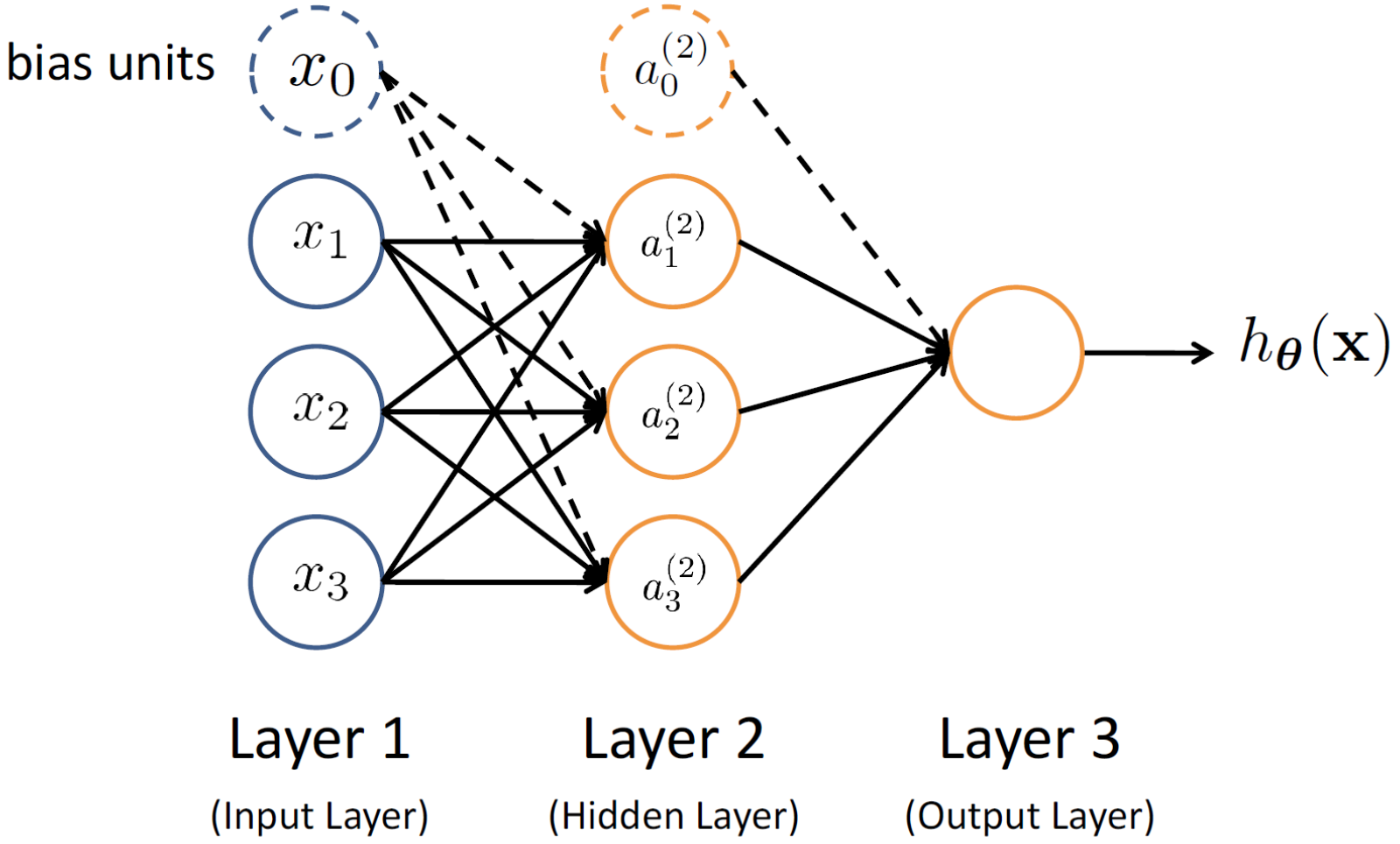


$$\mathbf{x} = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \qquad \boldsymbol{\theta} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix}$$

$$h_{\boldsymbol{\theta}}(\mathbf{x}) = g\left(\boldsymbol{\theta}^{\mathsf{T}}\mathbf{x}\right)$$

$$= \frac{1}{1 + e^{-\boldsymbol{\theta}^{\mathsf{T}}\mathbf{x}}}$$

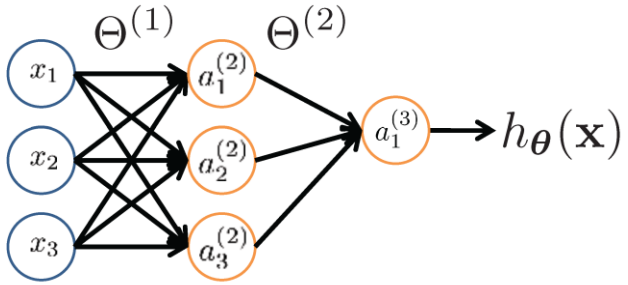
Neural Network



Feed-Forward Process

- Input layer units are set by some exterior function (think of these as **sensors**), which causes their output links to be **activated** at the specified level
- Working forward through the network, the **input function** of each unit is applied to compute the input value
  - Usually this is just the weighted sum of the activation on the links feeding into this node
- The **activation function** transforms this input function into a final value
  - Typically this is a **nonlinear** function, often a **sigmoid** function corresponding to the "threshold" of that node

Neural Network



$a_i^{(j)} =$ "activation" of unit i in layer j

$\Theta^{(j)} =$ weight matrix controlling function mapping from layer j to layer j + 1

$$a_1^{(2)} = g(\Theta_{10}^{(1)} x_0 + \Theta_{11}^{(1)} x_1 + \Theta_{12}^{(1)} x_2 + \Theta_{13}^{(1)} x_3)$$

$$a_2^{(2)} = g(\Theta_{20}^{(1)} x_0 + \Theta_{21}^{(1)} x_1 + \Theta_{22}^{(1)} x_2 + \Theta_{23}^{(1)} x_3)$$

$$a_3^{(2)} = g(\Theta_{30}^{(1)} x_0 + \Theta_{31}^{(1)} x_1 + \Theta_{32}^{(1)} x_2 + \Theta_{33}^{(1)} x_3)$$

$$h_\Theta(x) = a_1^{(3)} = g(\Theta_{10}^{(2)} a_0^{(2)} + \Theta_{11}^{(2)} a_1^{(2)} + \Theta_{12}^{(2)} a_2^{(2)} + \Theta_{13}^{(2)} a_3^{(2)})$$

If network has s_j units in layer j *and* s_{j+1} units in layer $j+1$, then $\Theta^{(j)}$ has dimension $s_{j+1} \times (s_j+1)$

$$\Theta^{(1)} \in \mathbb{R}^{3\times 4} \qquad \Theta^{(2)} \in \mathbb{R}^{1\times 4}$$

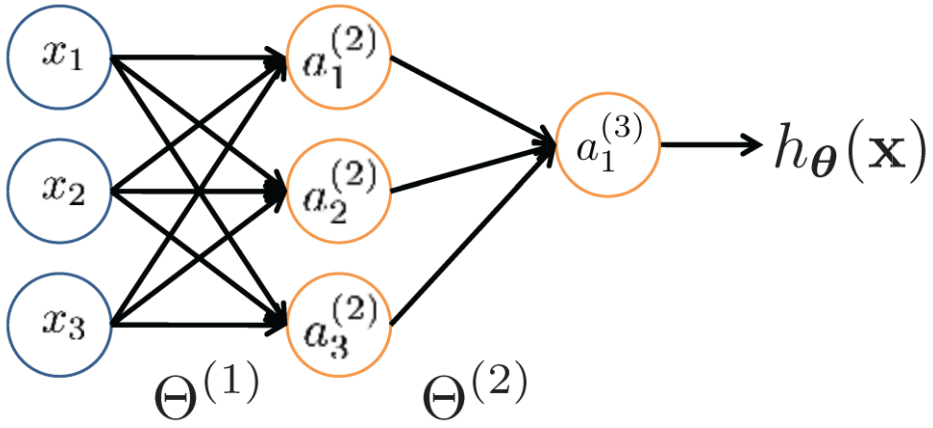
Vectorization

$$a_1^{(2)} = g\left(\Theta_{10}^{(1)}x_0 + \Theta_{11}^{(1)}x_1 + \Theta_{12}^{(1)}x_2 + \Theta_{13}^{(1)}x_3\right) = g\left(z_1^{(2)}\right)$$

$$a_2^{(2)} = g\left(\Theta_{20}^{(1)}x_0 + \Theta_{21}^{(1)}x_1 + \Theta_{22}^{(1)}x_2 + \Theta_{23}^{(1)}x_3\right) = g\left(z_2^{(2)}\right)$$

$$a_3^{(2)} = g\left(\Theta_{30}^{(1)}x_0 + \Theta_{31}^{(1)}x_1 + \Theta_{32}^{(1)}x_2 + \Theta_{33}^{(1)}x_3\right) = g\left(z_3^{(2)}\right)$$

$$h_\Theta(\mathbf{x}) = g\left(\Theta_{10}^{(2)}a_0^{(2)} + \Theta_{11}^{(2)}a_1^{(2)} + \Theta_{12}^{(2)}a_2^{(2)} + \Theta_{13}^{(2)}a_3^{(2)}\right) = g\left(z_1^{(3)}\right)$$



Feed-Forward Steps:

$$\mathbf{z}^{(2)} = \Theta^{(1)}\mathbf{x}$$

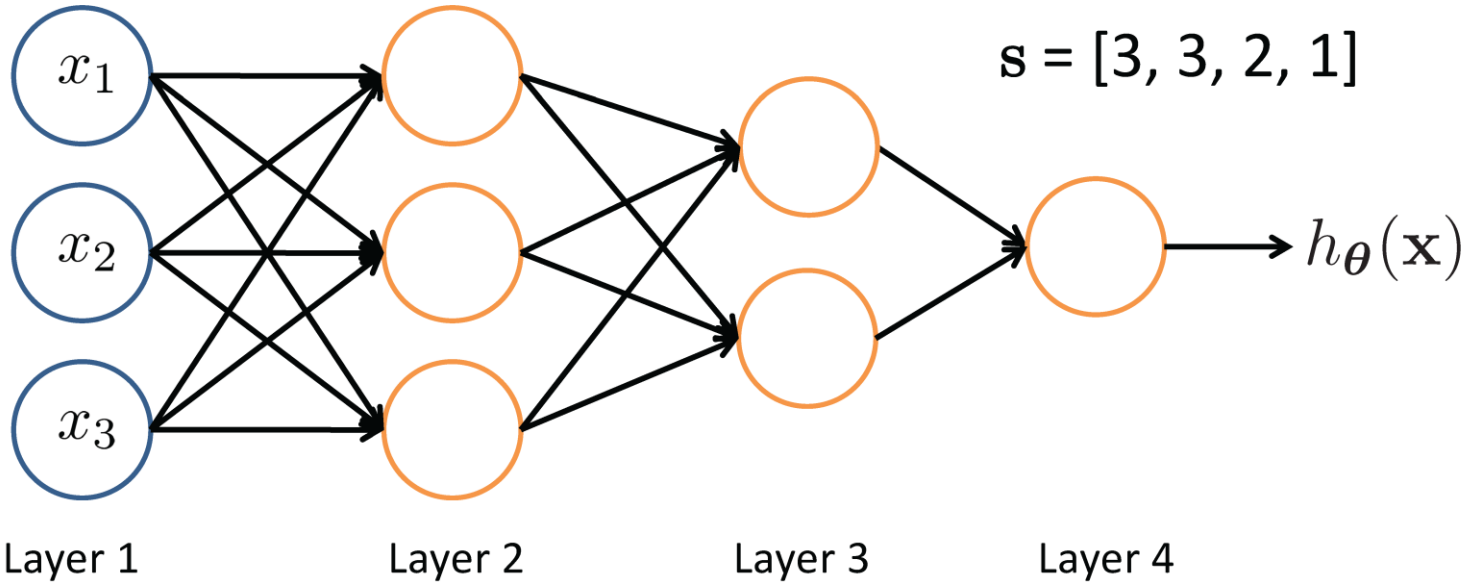
$$\mathbf{a}^{(2)} = g(\mathbf{z}^{(2)})$$

$$\text{Add } a_0^{(2)} = 1$$

$$\mathbf{z}^{(3)} = \Theta^{(2)}\mathbf{a}^{(2)}$$

$$h_\Theta(\mathbf{x}) = \mathbf{a}^{(3)} = g(\mathbf{z}^{(3)})$$

Other Network Architectures



L denotes the number of layers

$\mathbf{s} \in \mathbb{N}^{+L}$ contains the numbers of nodes at each layer

- Not counting bias units
- Typically, $s_0 = d$ (# input features) and $s_{L-1} = K$ (# classes)

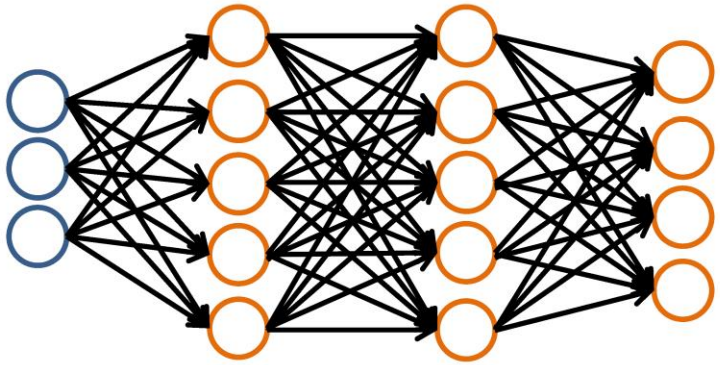
Multiple Output Units: One-vs-Rest

Pedestrian

Car

Motorcycle

Truck



$$h_\Theta(\mathbf{x}) \in \mathbb{R}^K$$

We want:

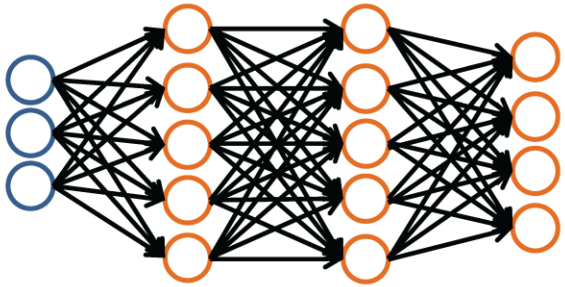
$$h_\Theta(\mathbf{x}) \approx \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$ when pedestrian

$$h_\Theta(\mathbf{x}) \approx \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$ when car

$$h_\Theta(\mathbf{x}) \approx \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$$ when motorcycle

$$h_\Theta(\mathbf{x}) \approx \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$ when truck

Multiple Output Units: One-vs-Rest



$h_\Theta(\mathbf{x}) \in \mathbb{R}^K$

We want:

$h_\Theta(\mathbf{x}) \approx \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$ when pedestrian

$h_\Theta(\mathbf{x}) \approx \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$ when car

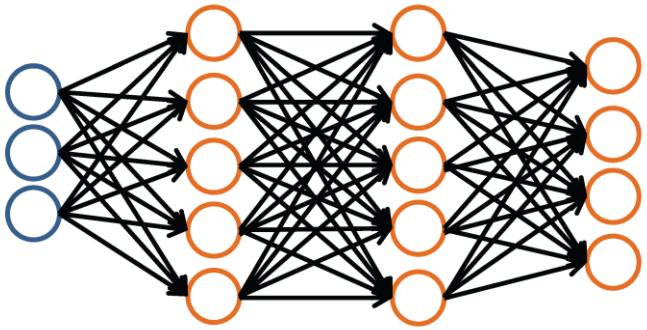
$h_\Theta(\mathbf{x}) \approx \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$ when motorcycle

$h_\Theta(\mathbf{x}) \approx \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$ when truck

---

- Given $\{(\mathbf{x}_1, y_1),\ (\mathbf{x}_2, y_2),\ \ldots,\ (\mathbf{x}_n, y_n)\}$
- Must convert labels to 1-of-K representation
  - e.g., $\mathbf{y}_i = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$ when motorcycle, $\mathbf{y}_i = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$ when car, etc.

Neural Network Classification



Given:

$\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_n, y_n)\}$

$\mathbf{s} \in \mathbb{N}^{+L}$ contains # nodes at each layer

- $s_0 = d$ (# features)

Binary classification

y = 0 or 1

1 output unit $(s_{L-1} = 1)$

Multi-class classification (K classes)

$\mathbf{y} \in \mathbb{R}^K$ e.g. $\begin{bmatrix}1\\0\\0\\0\end{bmatrix}$, $\begin{bmatrix}0\\1\\0\\0\end{bmatrix}$, $\begin{bmatrix}0\\0\\1\\0\end{bmatrix}$, $\begin{bmatrix}0\\0\\0\\1\end{bmatrix}$

pedestrian car motorcycle truck

K output units $(s_{L-1} = K)$

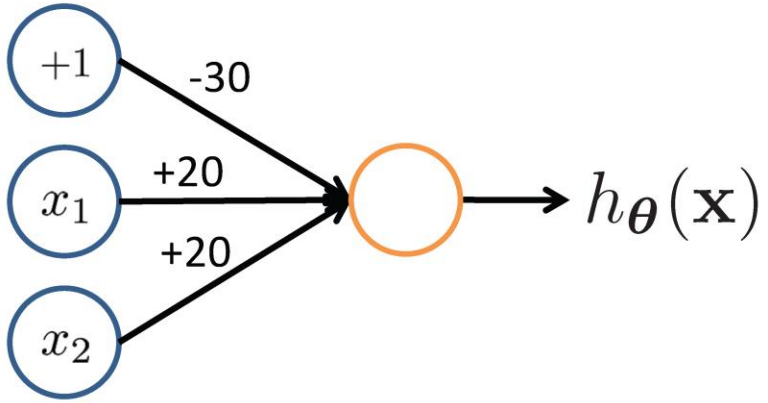
Understanding Representations

Representing Boolean Functions

Simple example: AND

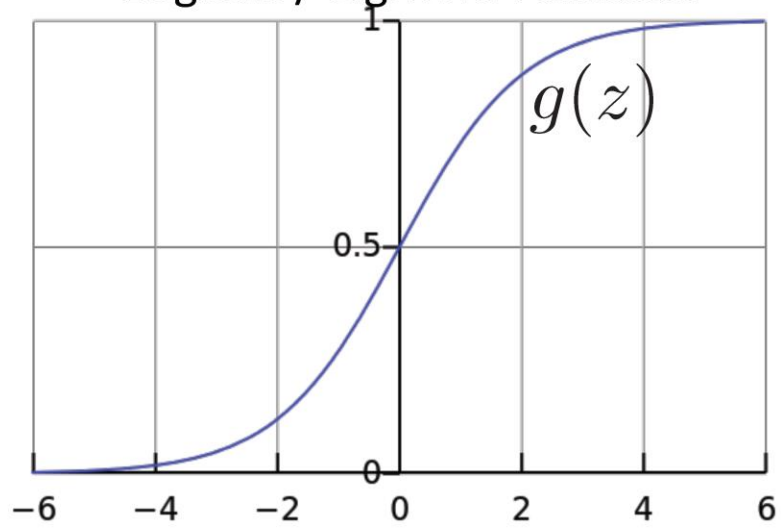
$x_1, x_2 \in \{0, 1\}$

$y = x_1 \text{ AND } x_2$



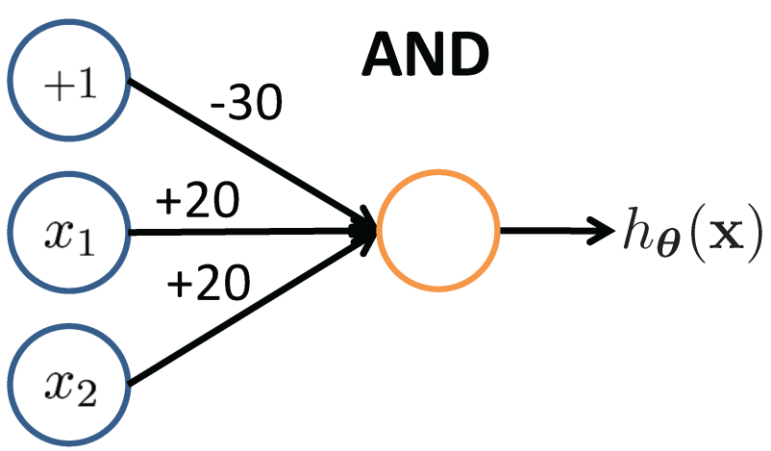
$$\mathrm{h}_\Theta(\mathbf{x}) = g(\text{-}30 + 20x_1 + 20x_2)$$

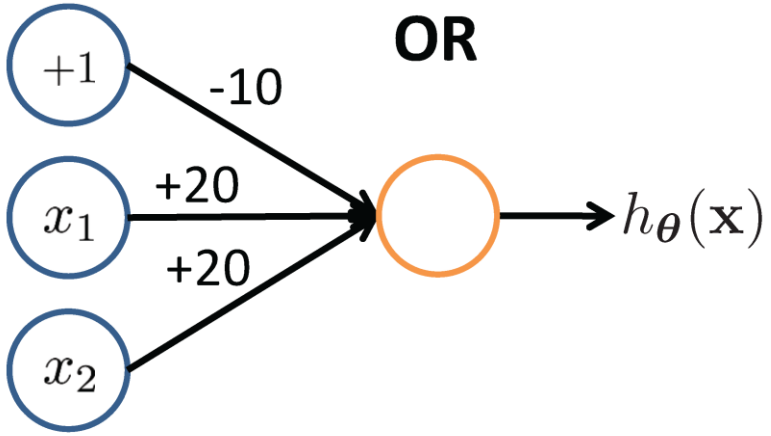
Logistic / Sigmoid Function

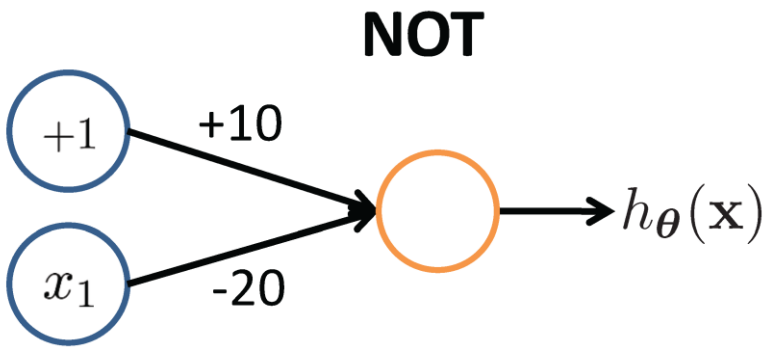


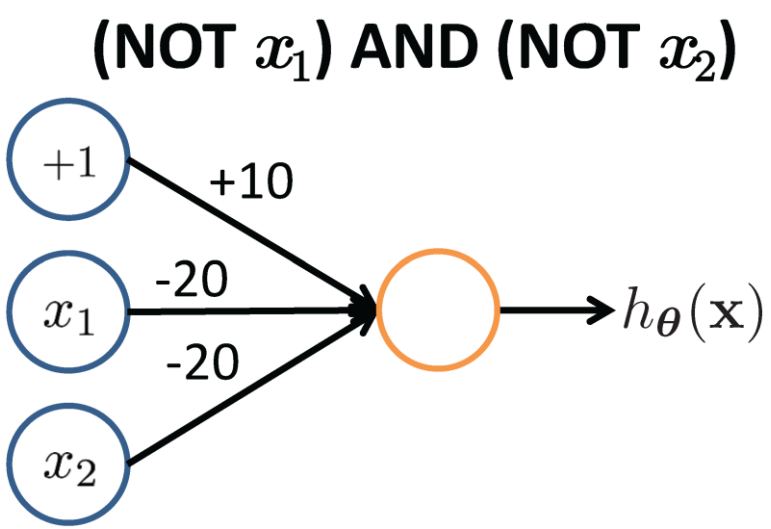
x_1	x_2	$\mathrm{h}_\Theta(\mathbf{x})$
0	0	$g(\text{-}30) \approx 0$
0	1	$g(\text{-}10) \approx 0$
1	0	$g(\text{-}10) \approx 0$
1	1	$g(10) \approx 1$

Representing Boolean Functions

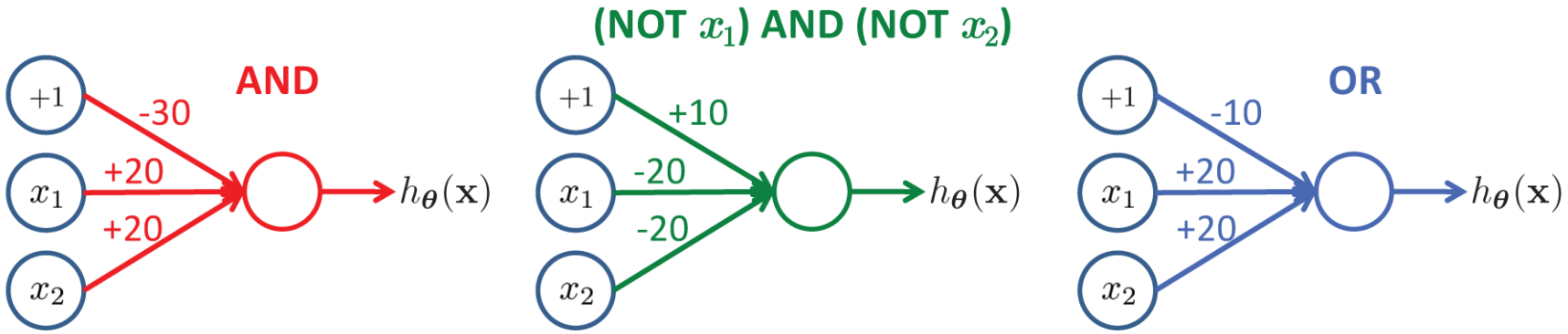


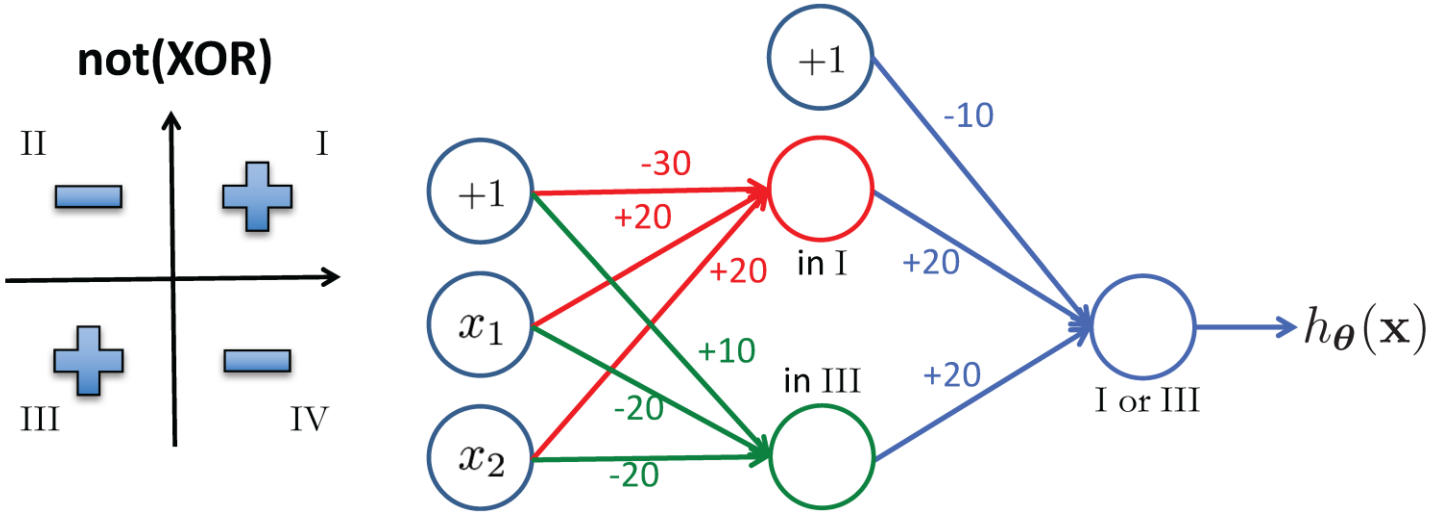




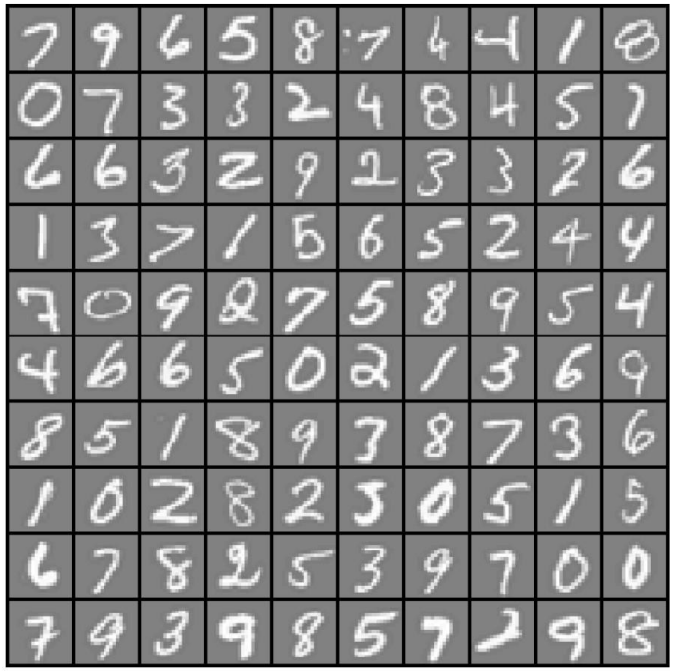


Combining Representations to Create Non-Linear Functions

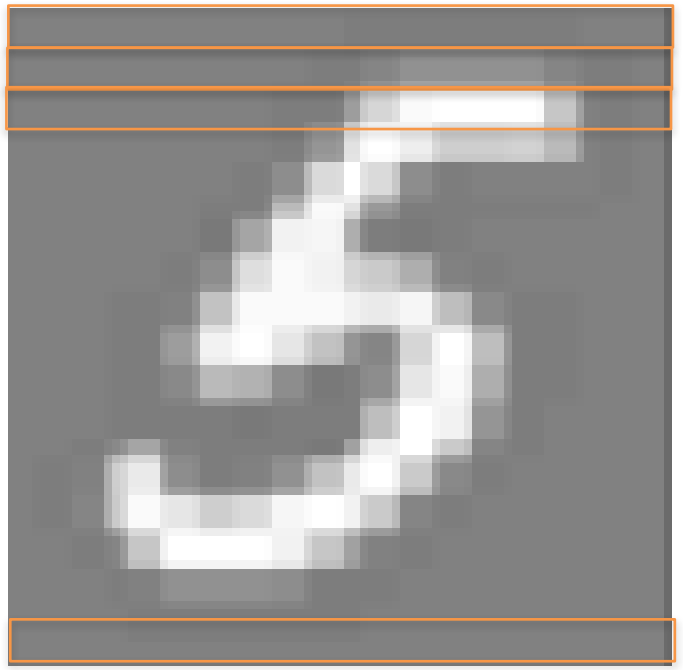




Layering Representations

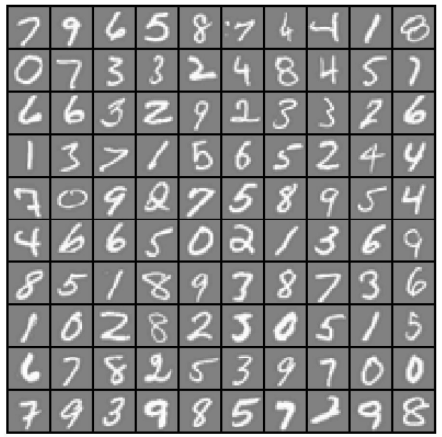


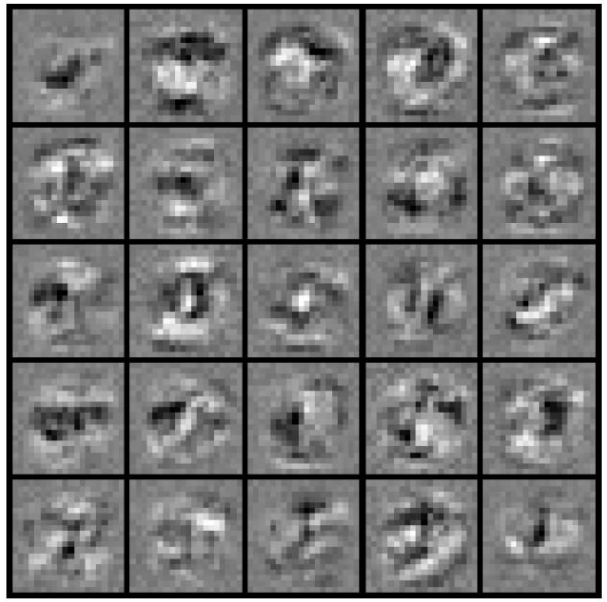
20 × 20 pixel images
$d = 400$ 10 classes

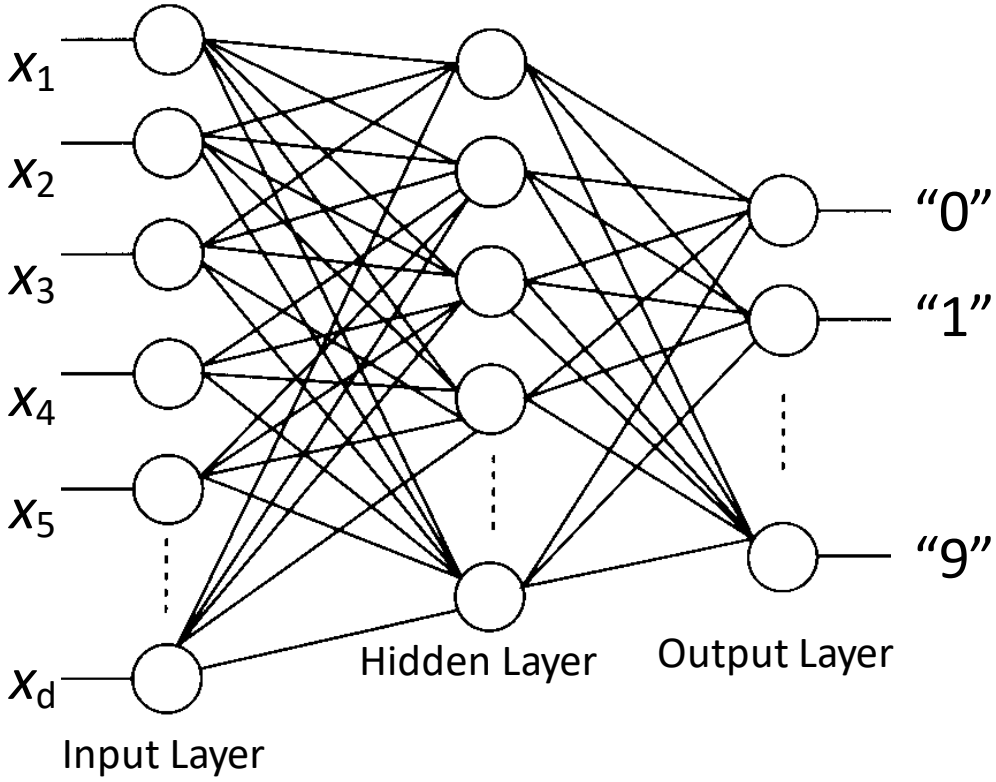


Each image is “unrolled” into a vector x of pixel intensities

Layering Representations







Visualization of
Hidden Layer

LeNet 5 Demonstration:
https://www.youtube.com/watch?v=FwFduRA_L6Q&ab_channel=YannLeCun

LeNet layers:
https://en.wikipedia.org/wiki/LeNet#:~:text=In%20general%2C%20LeNet%20refers%20to,in%20large%2Dscale%20image%20processing.