- Homework 3, due Friday, February 25,

Lecture 18: Kernels (continued)

$$\hat{y} = \hat{w}^T x_{new} = \sum_{i=1}^{n} \alpha_i \cdot \underbrace{x_i^T x_{new}}_{K(x_i, x_{new})} = \sum_{i=1}^{n} \alpha_i \cdot K(x_i, x_{new})$$

RBF

Recap: Kernel trick finds the optimal solution for linear models under a feature map $\phi(\cdot)$

- Once we have chosen a feature map $\phi(\cdot) \in \mathbb{R}^p$, what we want to solve is

  $\widehat{w} = \arg\min_{w \in \mathbb{R}^p} \sum_{i=1}^{n} \ell\left(y_i, w^T \phi(x_i)\right)$ for some convex loss $\ell(,)$

- Kernel trick finds the optimal solution efficiently, by searching over the model that can be represented as $\widehat{w} = \sum_{i=1}^{n} \alpha_i \phi(x_i)$, which is equivalent to $\hat{y}_{\text{new}} = \sum_{i=1}^{n} \alpha_i K(x_i, x_{\text{new}})$

- Gradient descent update (from initialization $w^{(0)} = 0$) that find the optimal solution is

  $w^{(t+1)} \leftarrow w^{(t)} - \eta \sum_{i=1}^{n} \underbrace{\ell'(y_i, w^T \phi(x_i))}_{\text{scalar}} \phi(x_i)$

- One crucial observation is that all $w^{(t)}$'s (including the optimal solution $w^{(\infty)}$) lie on the subspace spanned by $\{\phi(x_1), \ldots, \phi(x_n)\}$, which is an n-dimensional subspace in $\mathbb{R}^p$
- Hence, it is sufficient to look for a solution that is represented as

  $\widehat{w} = \sum_{i=1}^{n} \alpha_i \phi(x_i)$ to find the optimal solution

Fixed Feature V.S. Learned Feature

- Kernel method works well if we choose a good kernel such that the data is linearly separable in the corresponding (possibly infinite dimensional) feature space

- In practice, it is hard to choose a good kernel for a given problem

- Can we **learn** the feature mapping $\phi : \mathbb{R}^d \rightarrow \mathbb{R}^p$ from data also?

Bootstrap

- How to measure uncertainty in our predictions

Confidence interval

- suppose you have training data $\{(x_i, y_i)\}_{i=1}^n$ drawn i.i.d. from some true distribution $P_{x,y}$
- we train a kernel ridge regressor, with some choice of a kernel $K : \mathbb{R}^{d \times d} \to \mathbb{R}$

$$\text{minmize}_\alpha \; \|\mathbf{K}\alpha - \mathbf{y}\|_2^2 + \lambda \alpha^T \mathbf{K} \alpha$$

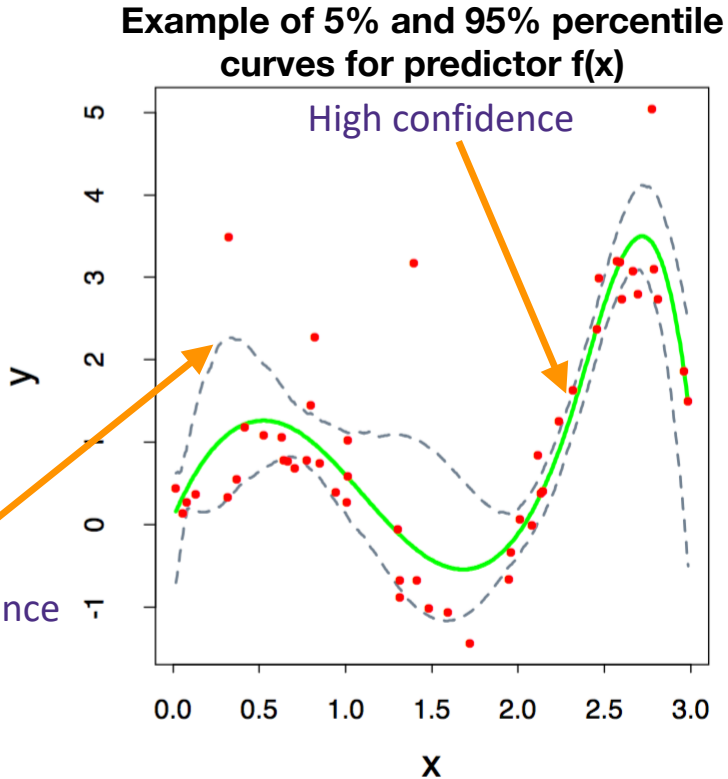
- the resulting predictor is

$$f(x) = \sum_{i=1}^n K(x_i, x)\hat{\alpha}_i,$$

where

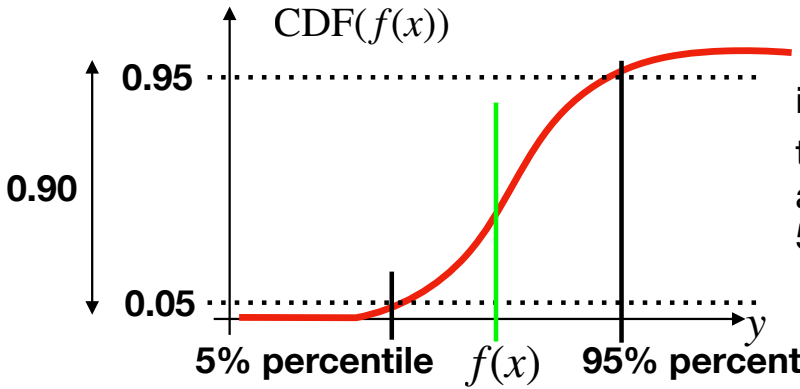
$$\hat{\alpha} = (\mathbf{K} + \lambda \mathbf{I})^{-1}\mathbf{y} \quad \in \mathbb{R}^n$$

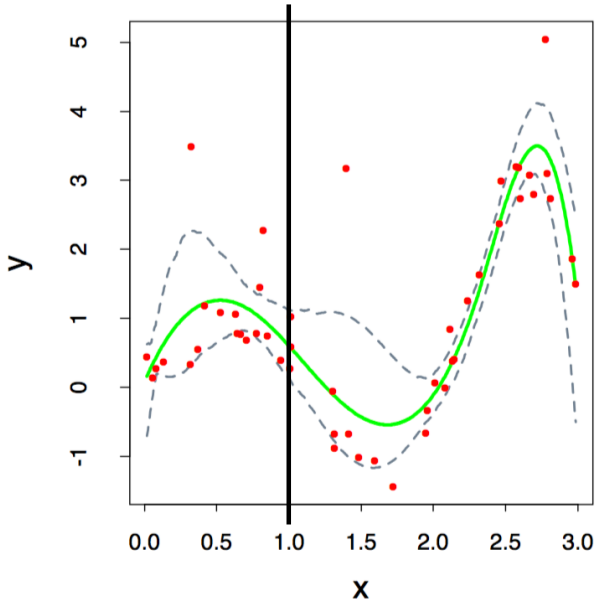
- we wish to build a confidence interval for our predictor $f(x)$, using 5% and 95% percentiles



Confidence interval

- let's focus on a single $x \in \mathbb{R}^d$
- note that our predictor $f(x)$ is a random variable, whose randomness comes from the training data $S_{\text{train}} = \{(x_i, y_i)\}_{i=1}^n$
- if we know the statistics (in particular the CDF of the random variable $f(x)$) of the predictor, then the **confidence interval** with **confidence level 90%** is defined as





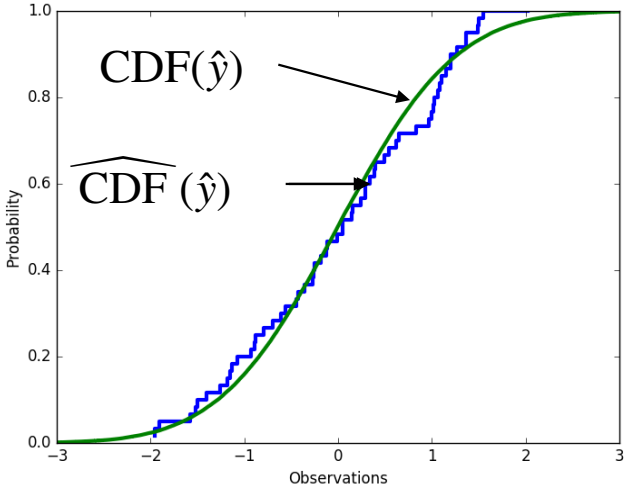
if we know the distribution of our predictor $f(x)$, the green line is the expectation $\mathbb{E}[f(x)]$ and the black dashed lines are the 5% and 95% percentiles in the figure above

- as we do not have the cumulative distribution function (CDF), we need to approximate them

Confidence interval

- hypothetically, if we can sample as many times as we want,
  then we can train $B \in \mathbb{Z}^+$ i.i.d. predictors, each trained on n fresh samples to get **empirical estimate of the CDF of** $\hat{y} = f(x)$

- for b=1,…,B
  - draw n fresh samples $\{(x_i^{(b)}, y_i^{(b)})\}_{i=1}^n$
  - train a regularized kernel regression $\alpha^{*(b)}$
  - Predict $\hat{y}^{(b)} = \sum_{i=1}^{n} K(x_i^{(b)}, x)\alpha_i^{*(b)}$



- let the empirical CDF of those B predictors $\{\hat{y}^{(b)}\}_{b=1}^B$ be $\widehat{\text{CDF}}(\hat{y})$, defined as

$$\widehat{\text{CDF}}(\hat{y}) = \frac{1}{B}\sum_{b=1}^{B} \mathbf{I}\{\hat{y}^{(b)} \le \hat{y}\} = \frac{1}{B}\sum_{b=1}^{B} \mathbf{I}\{(\alpha^{*(b)})^T h(x) \le \hat{y}\}$$

- compute the confidence interval using $\widehat{\text{CDF}}(\hat{y})$
- What is wrong?

Bootstrap

- as we cannot sample repeatedly (in typical cases), we use **bootstrap samples** instead
- bootstrap is a general tool for assessing statistical accuracy
- we learn it in the context of confidence interval for trained models

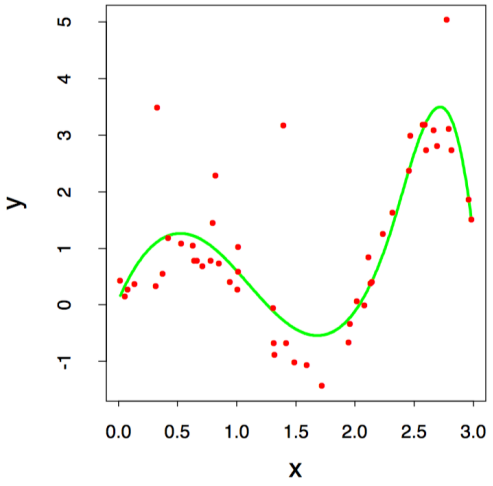
- a **bootstrap dataset** is created from the training dataset by taking n (the same size as the training data) examples uniformly at random **with replacement** from the training data $\{(x_i, y_i)\}_{i=1}^{n}$

- for b=1,…,B
  - create a bootstrap dataset $S_{\text{bootstrap}}^{(b)}$
  - train a regularized kernel regression $\alpha^{*(b)}$
  - predict $\hat{y}^{(b)} = \sum_{i=1}^{n} K(x_i^{(b)}, x)\alpha_i^{*(b)}$

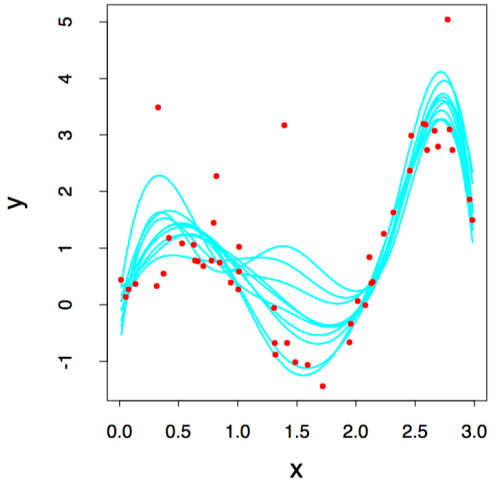
- compute the empirical CDF from the bootstrap datasets, and compute the confidence interval

bootstrap

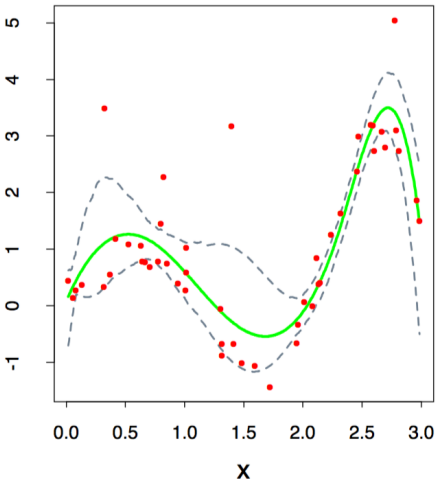
training a single predictor



multiple bootstrapped predictors



90% confidence interval



Figures from Hastie et al

Questions?

Neural Networks

Neural Networks

- Origins: Algorithms that try to mimic the brain.
- Widely used in 80s and early 90s; popularity diminished in late 90s.
- Recent resurgence from 2010s: state-of-the-art techniques for many applications:
  - Computer Vision (AlexNet 2012)
  - Natural language processing
  - Speech recognition
  - Decision-making / control problems (AlphaGo, Games, robots)
- Limited theory:
  - Why do we find good minima with SGD for Non-convex loss?
  - Why do we not overfit when # of parameters p is much larger than # of samples n?

Neural Networks

Agenda:

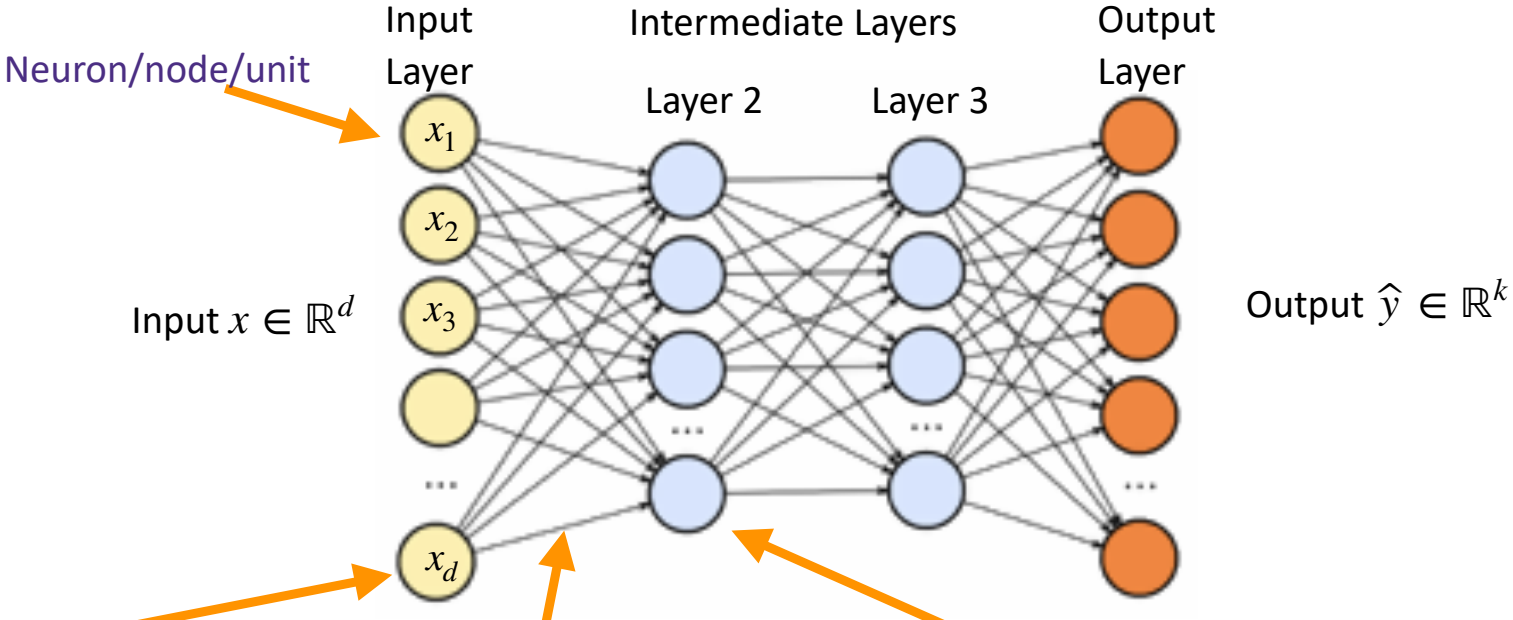
1. Definitions of neural networks

2. Training neural networks:
   1. Algorithm: back propagation
   2. Putting it to work

3. Neural network architecture design:
   1. Convolutional neural network

Neural Networks

- **Neural Network** is a parametric family of functions from $x \in \mathbb{R}^d$ to $\hat{y} = h_\theta(x) \in \mathbb{R}^k$ with parameter $\theta \in \mathbb{R}^p$
- **Computation graph** illustrates the sequence of operations to be performed by a neural network



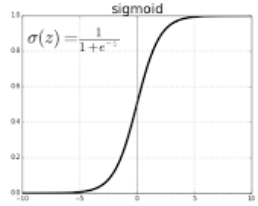
d nodes each representing a scalar value of each coordinate of x

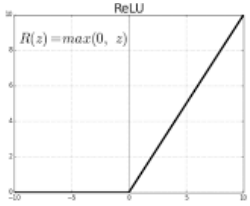
Link: maps output of a neuron to input of a neuron of the next layer, each link has a scalar weight

Neuron:
1. Input: weighted sum of previous layer
2. Apply scalar activation function
3. Output: links to the next layer

Sequence of operations performed at a single node

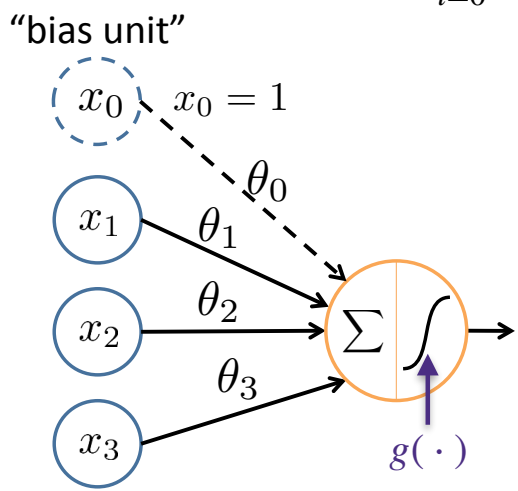
- For a single node with input $x \in \mathbb{R}^d$, the node is defined by
  - **Parameter** $\theta \in \mathbb{R}^{d+1}$ (including the intercept/bias)
  - **Activation function** $g : \mathbb{R} \to \mathbb{R}$





    - A common choice is sigmoid function: $g(z) = \dfrac{1}{1+e^{-z}}$
    - Another popular choice is Rectified Linear Unit (ReLU): $g(z) = \max\{0,z\}$

- The node performs $h_\theta(x) = g\left(\sum_{i=0}^{d} \theta_i x_i\right) = g(\theta^T x) = \dfrac{1}{1+e^{-\theta^T x}}$

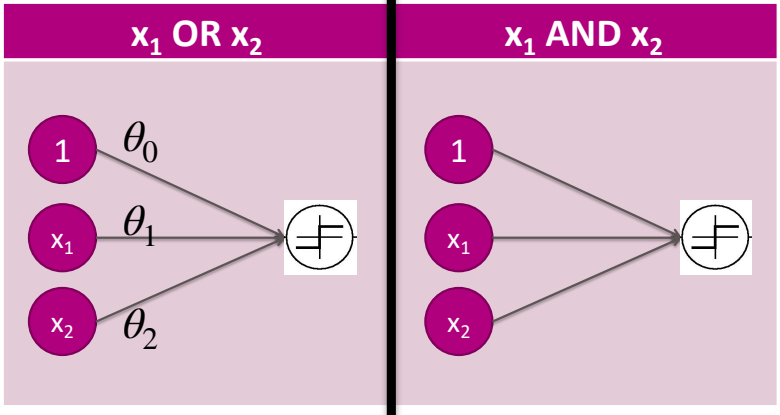


$$\mathbf{x} = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \qquad \boldsymbol{\theta} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix}$$

$$h_{\boldsymbol{\theta}}(\mathbf{x}) = g\left(\boldsymbol{\theta}^\mathsf{T}\mathbf{x}\right) = \frac{1}{1+e^{-\boldsymbol{\theta}^\mathsf{T}\mathbf{x}}}$$

Toy example: What can be represented by a single node with $g(z) = \text{sign}(z)$?

x[1]	x[2]	y
0	0	0
0	1	1
1	0	1
1	1	1

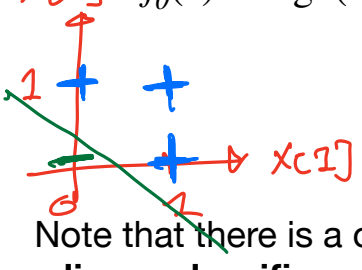


x[1]	x[2]	y
0	0	0
0	1	0
1	0	0
1	1	1

What should be the weights?

$f_\theta(x) = \text{sign}(\theta_0 + \theta_1 x[1] + \theta_2 x[2])$

$f_\theta(x) = \text{sign}(\theta_0 + \theta_1 x[1] + \theta_2 x[2])$



$\theta_0 = -1/2$
$\theta_1 = \theta_2 = 1$

Note that there is a one-to-one correspondence between
a **linear classifier** and a **neural network with a single node** of the above form

X[1] XOR X[2] ~ f

+ −

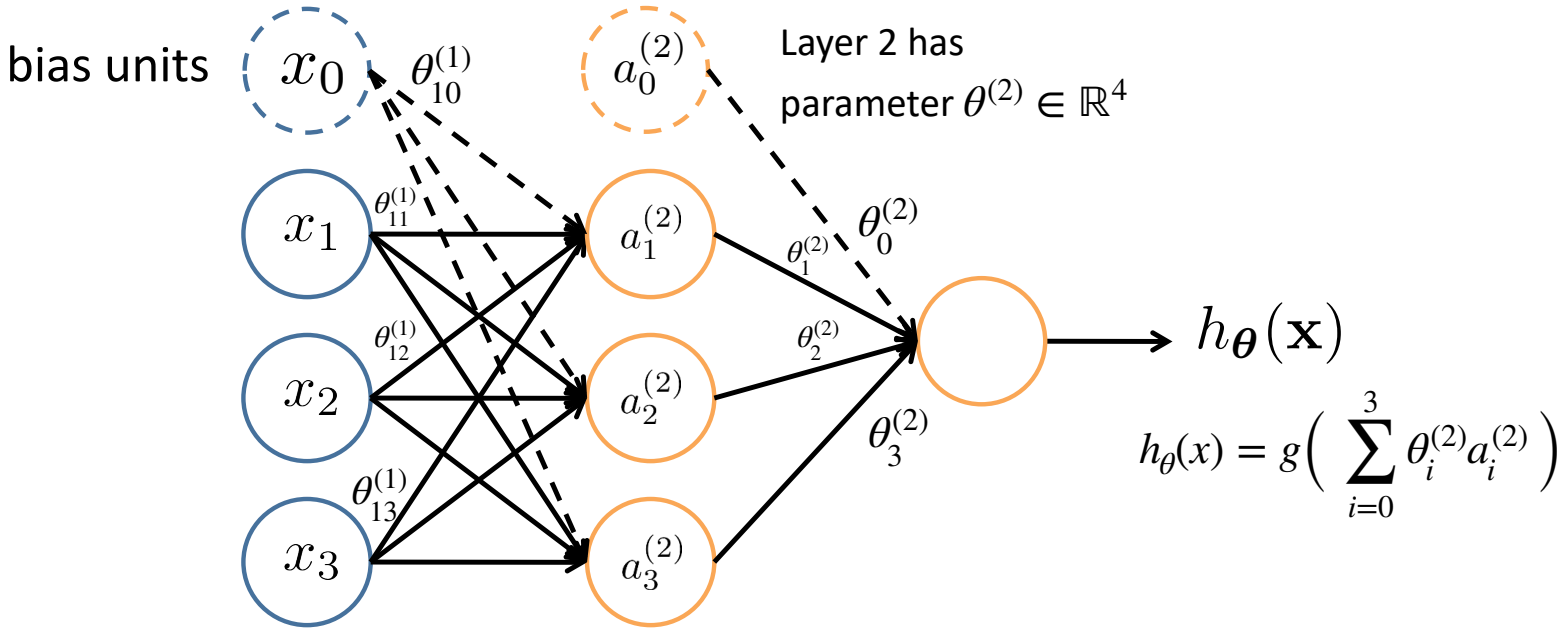
What cannot be learned?

Neural Network composes simple functions to make complex functions

- Each layer performs simple operations
- Neural Network (with parameter $\theta = (\theta^{(1)}, \theta^{(2)})$) composes multiple layers of operations

Layer 1 has parameter $\theta^{(1)} \in \mathbb{R}^{3\times 4}$

$$a_1^{(2)} = g\Big(\sum_{i=0}^{3} \theta_{1i}^{(1)} x_i\Big)$$

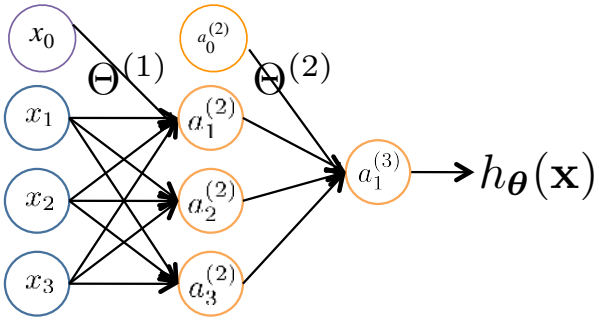


Layer 1 (Input Layer)

Layer 2 (Hidden Layer)

Layer 3 (Output Layer)

This is called a 2-layer Neural Network



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

$\Theta^{(j)}$ = weight matrix stores parameters from layer *j* to layer *j* + 1

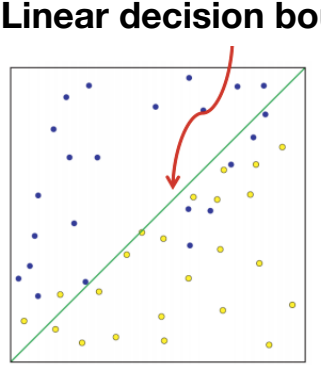
$$a_1^{(2)} = g(\Theta_{10}^{(1)}x_0 + \Theta_{11}^{(1)}x_1 + \Theta_{12}^{(1)}x_2 + \Theta_{13}^{(1)}x_3)$$
$$a_2^{(2)} = g(\Theta_{20}^{(1)}x_0 + \Theta_{21}^{(1)}x_1 + \Theta_{22}^{(1)}x_2 + \Theta_{23}^{(1)}x_3)$$
$$a_3^{(2)} = g(\Theta_{30}^{(1)}x_0 + \Theta_{31}^{(1)}x_1 + \Theta_{32}^{(1)}x_2 + \Theta_{33}^{(1)}x_3)$$
$$h_\Theta(x) = a_1^{(3)} = g(\Theta_{10}^{(2)}a_0^{(2)} + \Theta_{11}^{(2)}a_1^{(2)} + \Theta_{12}^{(2)}a_2^{(2)} + \Theta_{13}^{(2)}a_3^{(2)})$$

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

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

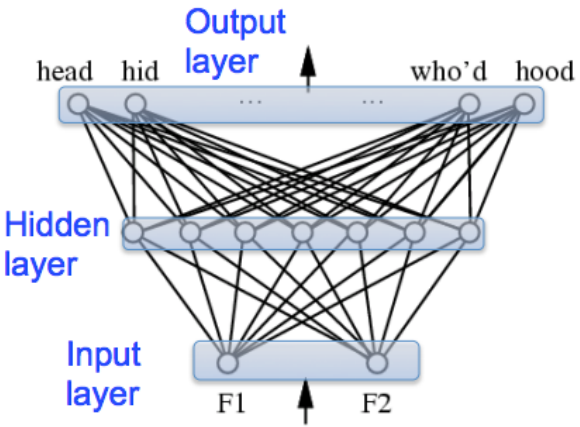
Example of 2-layer neural network in action

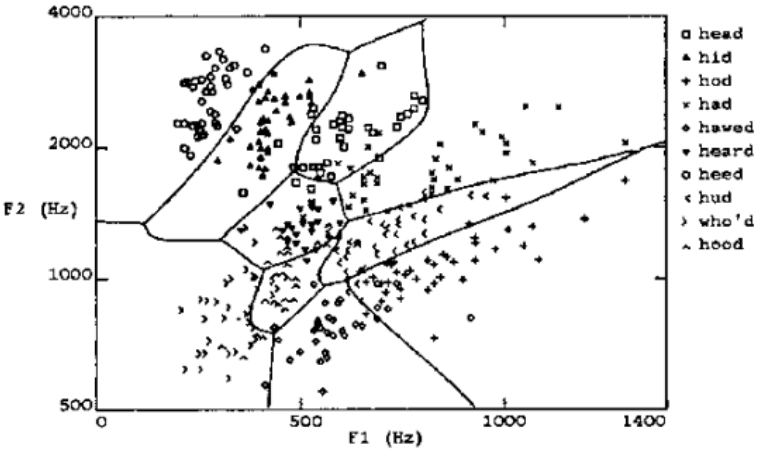
1-layer neural networks
only represents linear classifiers



Example: 2-layer neural network trained to distinguish vowel sounds using 2 formants (features)

a highly non-linear decision boundary can be learned from 2-layer neural networks





Neural Networks are arbitrary function approximators

Theorem 10 (Two-Layer Networks are Universal Function Approximators). *Let F be a continuous function on a bounded subset of D-dimensional space. Then there exists a two-layer neural network $\hat{F}$ with a finite number of hidden units that approximate F arbitrarily well. Namely, for all $\boldsymbol{x}$ in the domain of F, $\left|F(\boldsymbol{x}) - \hat{F}(\boldsymbol{x})\right| < \epsilon$.*

Cybenko, Hornik (theorem reproduced from CIML, Ch. 10)

But Deep Neural Networks have many powerful properties not yet understood theoretically.

Multi-layer Neural Network - Binary Classification in $\{0,1\}$

L-th layer plays the role of features, but trained instead of pre-determined

This is a 5-dimensional vector

$$a^{(1)} = x$$

$$a^{(2)} = g(\Theta^{(1)} a^{(1)})$$

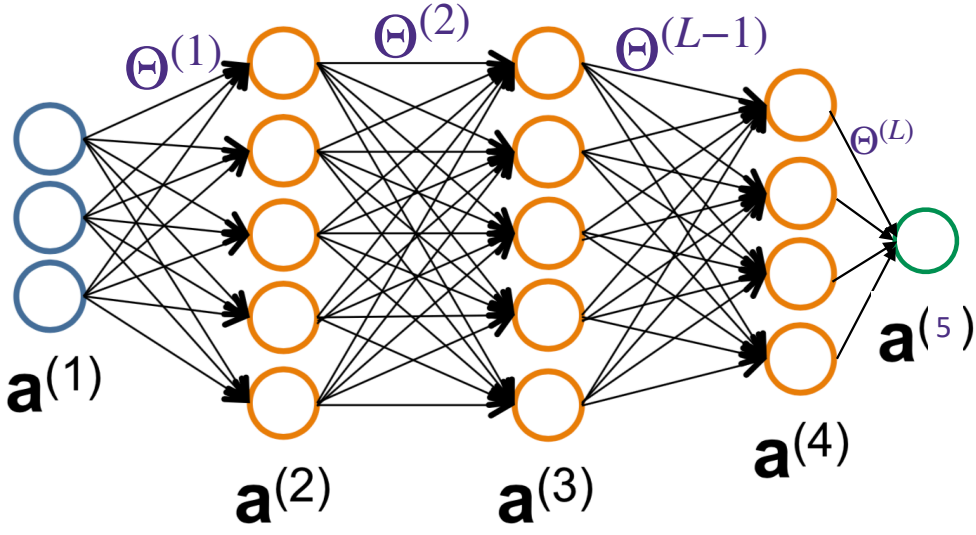
Scalar function g is applied coordinate-wise

$$a^{(l+1)} = g(\Theta^{(l)} a^{(l)})$$

$$\hat{y} = g(\Theta^{(L)} a^{(L)})$$

(Learned) feature representation

Logistic regression



$$L(y, \hat{y}) = y \log(\hat{y}) + (1 - y)\log(1 - \hat{y})$$

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

Binary Logistic Regression with learned feature $a^{(4)}$

Multi-layer Neural Network - Binary Classification

$$a^{(1)} = x$$

$$a^{(2)} = \sigma(\Theta^{(1)} a^{(1)})$$

Sigmoid

$$\vdots$$

$$a^{(l+1)} = \sigma(\Theta^{(l)} a^{(l)})$$

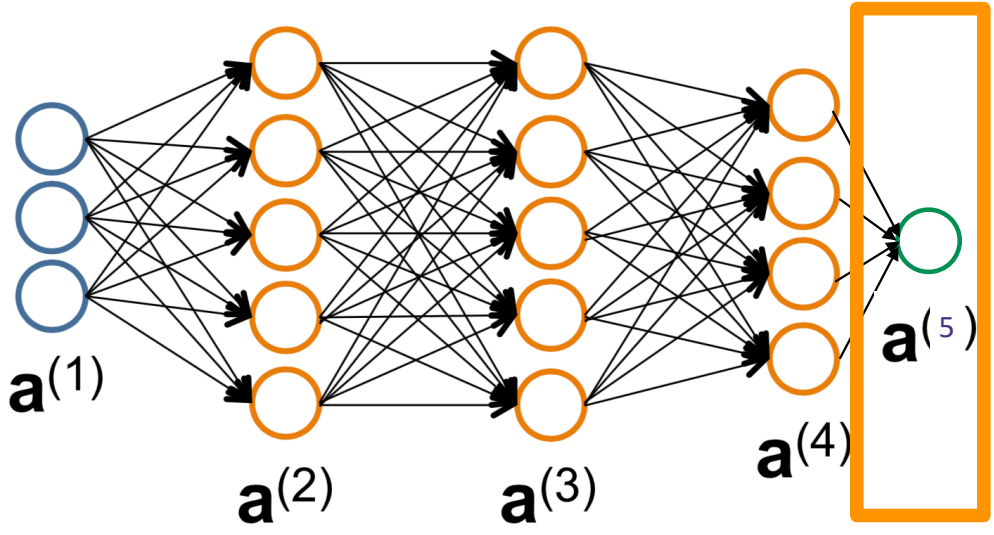
$$\vdots$$

$$\hat{y} = g(\Theta^{(L)} a^{(L)})$$

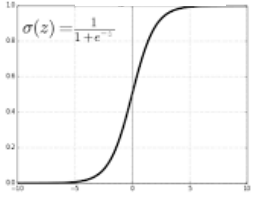
ReLU

(Learned) feature representation

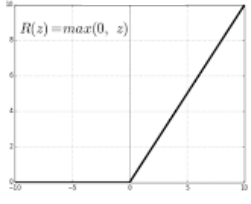
Logistic regression



Sigmoid



ReLU



- Why is ReLU better than sigmoid?

$$L(y, \hat{y}) = y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})$$

$$\sigma(z) = \max\{0, z\} \qquad g(z) = \frac{1}{1 + e^{-z}}$$

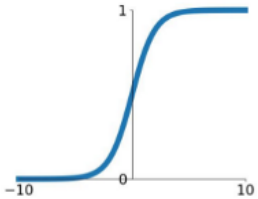
Binary Logistic Regression

Nonlinear activation function

- popular choices of activation function includes

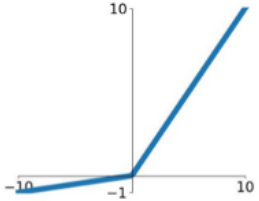
Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



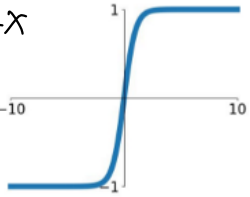
Leaky ReLU

$$\max(0.1x, x)$$



tanh

$$\tanh(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$$

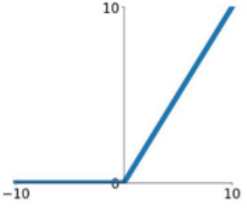


Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

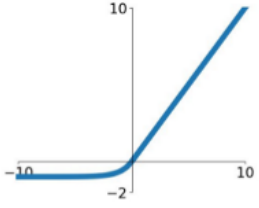
ReLU

$$\max(0, x)$$



ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



- Why is ReLU better than Sigmoid?
- Why is ELU better than ReLU?

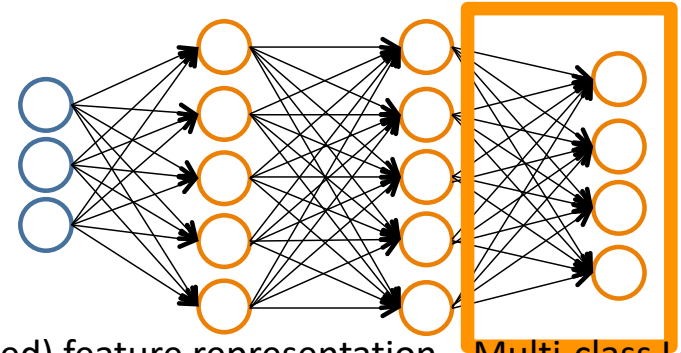
K-class Classification: multiple output units

Pedestrian

Car

Motorcycle

Truck



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

Multi-class Logistic Regression

(Learned) feature representation

Multi-class Logistic regression

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

Multi-layer Neural Network - Regression

$$a^{(1)} = x$$

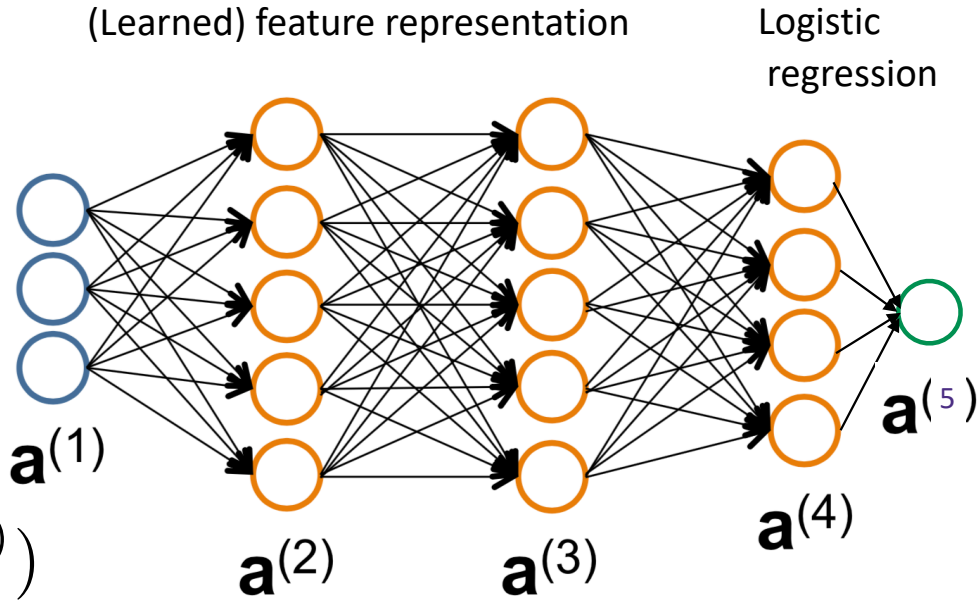
$$a^{(2)} = \sigma(\Theta^{(1)} a^{(1)})$$

$$\vdots$$

$$a^{(l+1)} = \sigma(\Theta^{(l)} a^{(l)})$$

$$\vdots$$

$$\widehat{y} = \Theta^{(L)} a^{(L)}$$



$$L(y, \widehat{y}) = (y - \widehat{y})^2$$

$$\sigma(z) = \max\{0, z\}$$

Regression

Questions?