

CSE 417T

# Introduction to Machine Learning

Review of Exam 2

Instructor: Chien-Ju (CJ) Ho

# Logistics: Exam 2

- Exam 2 Date: April 28, 2022 (Thursday)
  - In-class exam (the same time/location as the lecture)
  - Exam duration: 75 minutes
  - Content
    - Focus on the content of 2<sup>nd</sup> half of the semester
    - However, knowledge is cumulative, and you are still expected to know the concepts earlier
- 2 sections of questions
  - ~5 long questions (written response questions with explanations required)
  - 10 multiple choice questions (no explanations needed)
- Closed-book exam. You can bring two cheat-sheets
  - Up to letter size, front and back (up to 4 pages)
  - No format limitations (it can be typed, written, or a combination)
- No calculators (you don't need them)

# Logistics: Exam Policies

- I might arrange random seat assignments
  - Will be announced on Piazza the night before the exam if I do

123456																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																				
--------	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--

# Logistics: Exam Policies

- Please arrive on time. No extensions will be given if you arrive late.
- During the exam, if you have a question or if you finish before time is up:
  - **Do not get up**
  - Raise your hand and I will come to you
  - I most likely will not answer questions to individual students
    - But I'll give clarifications to everyone if multiple students ask the same question
- When time is called:
  - **Stop writing**
  - **Do not get up**
  - We will come around and collect your exam

# Homework

- Solution Sketch of HW5 has been posted on Gradescope
  - Not intended to be comprehensive

# Plans for Today

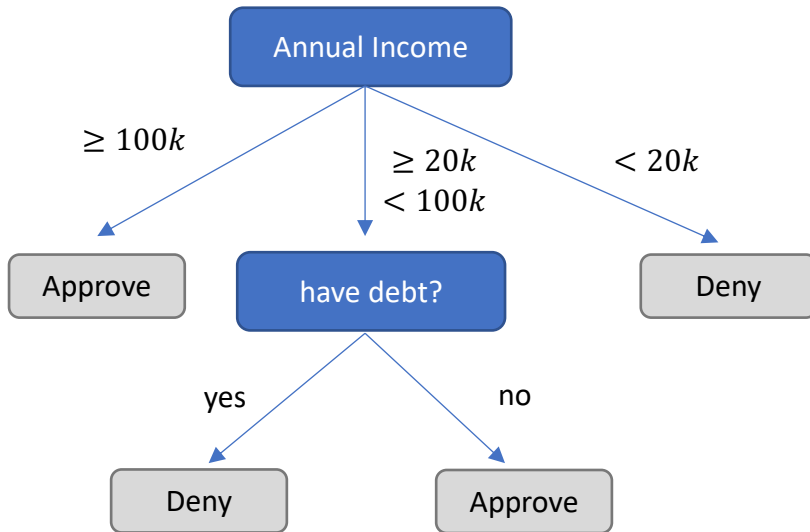
- Brief review of course content.
- Discussion of the practice questions.
- Discussion of any other questions you might have.

# Brief Review

A quick review brushed over what we talked about.

Not guaranteed (and not likely) to cover everything in the exam.

# Decision Tree Hypothesis



- Pros
  - Easy to interpret, handle multi-type data, easy to implement
- Cons
  - **bad generalization**
  - VC dimension is infinity, high variance, easily overfit
- Why we care?
  - One of the classical model
  - **Building block for other models** (e.g., random forest)

- Decision tree learning:

Goal:

minimize  $E_{in}$   
subject to  $size(tree) \leq C$

approximately

General\_DecisionTreeLearn( $D$ )

Create a root node  $r$

If **termination conditions** are met

return a single node tree with **leaf prediction** based on

Else: Greedily find a feature  $A$  to split according to **split criteria**

For each possible value  $v_i$  of  $A$

Let  $D_i$  be the dataset containing data with value  $v_i$  for feature  $A$

Create a subtree DecisionTreeLearn( $D_i$ ) that being the child of root  $r$



# ID3: Using Information Gain as Selection Criteria

- Information gain of choosing feature  $A$  to split
  - $Gain(D, A) = H(D) - \sum_i \frac{|D_i|}{|D|} H(D_i)$  [The amount of decrease in entropy]
- ID3: Choose the split that maximize  $Gain(D, A)$

Notations:

$H(D)$ : Entropy of  $D$

$|D|$  is the number of points in  $D$

General\_DecisionTreeLearn( $D$ )

Create a root node  $r$

If **termination conditions** are met

return a single node tree with **leaf prediction** based on

Else: Greedily find a feature  $A$  to split according to **split criteria**

For each possible value  $v_i$  of  $A$

Let  $D_i$  be the dataset containing data with value  $v_i$  for feature  $A$

Create a subtree DecisionTreeLearn( $D_i$ ) that being the child of root  $r$

- ID3 termination conditions
  - If all labels are the same
  - If all features are the same
  - If dataset is empty
- ID3 leaf predictions
  - Most common labels (majority voting)
- ID3 split criteria
  - Information gain

# Ensemble Learning

- Goal: Utilize a set of **weak learners** to obtain a **strong learner**.
- Format of ensemble learning
  - **Construct** many **diverse** weak learners
  - **Aggregate** the weak learners

## Bagging:

- Construct diverse weak learners
  - (**Simultaneously**) bootstrapping datasets
  - Train weak learners on them
- Aggregate the weak learners
  - **Uniform** aggregation

## Boosting

- Construct diverse weak learners
  - **Adaptively** generating datasets
  - Train weak learners on them
- Aggregate the weak learners
  - **Weighted** aggregation

Weak learner  
choice:

Fully-grown decision trees  
(low bias, high variance)

Decision stumps  
(high bias, low variance)

# Bagging and Random Forest

- Construct many random trees
  - Bootstrapping datasets (Sample with replacement from  $D$ )
  - Learn a **max-depth tree** for each of them
  - Other randomizations
    - When choosing split features, choose from a random subset (instead of all features)
    - Randomly project features (similar to non-linear transformation) for each tree
- Aggregate the random trees
  - Classification: Majority vote  $\bar{g}(\vec{x}) = \text{sign}\left(\frac{1}{M} \sum_{m=1}^M g_m(\vec{x})\right)$
  - Regression: Average  $\bar{g}(\vec{x}) = \frac{1}{M} \sum_{m=1}^M g_m(\vec{x})$

# Outline of a Boosting Algorithm

- Initialize  $D_1$
- For  $t = 1$  to  $T$ 
  - Learn  $g_t$  from  $D_t$
  - Reweight the distribution and obtain  $D_{t+1}$  based on  $g_t$  and  $D_t$
- Output  $\text{weighted-aggregate}(g_1, \dots, g_T)$ 
  - Classification:  $G(\vec{x}) = \bar{g}(\vec{x}) = \text{sign}\left(\frac{1}{T} \sum_{t=1}^T \alpha_t g_t(\vec{x})\right)$

## Questions

How to learn  $g_t$  from  $D_t$

How to reweight the distribution and obtain  $D_{t+1}$

How to perform weighted aggregation

# AdaBoost Algorithm

- Given  $D = \{(\vec{x}_1, y_1), \dots, (\vec{x}_N, y_N)\}$
- Initialize  $D_1(n) = 1/N$  for all  $n = 1, \dots, N$
- For  $t = 1, \dots, T$ 
  - Learn  $g_t$  from  $D_t$  (using decision stumps)
  - Calculate  $\epsilon_t = E_{in}^{(D_t)}(g_t)$
  - Set  $\alpha_t = \frac{1}{2} \ln \left( \frac{1 - \epsilon_t}{\epsilon_t} \right)$
  - Update  $D_{t+1}(n) = \frac{1}{Z_t} D_t(n) e^{-\alpha_t y_n g_t(\vec{x}_n)}$
- Output  $G(\vec{x}) = \text{sign}(\sum_{t=1}^T \alpha_t g_t(\vec{x}))$

# Nearest Neighbors

- Predict  $\vec{x}$  according to its nearest neighbor
  - Given  $D = \{(\vec{x}_1, y_1), (\vec{x}_2, y_2), \dots, (\vec{x}_N, y_N)\}$
  - Let  $\vec{x}_{[1]}$  be  $\vec{x}$ 's nearest neighbor, i.e., the closest point to  $\vec{x}$  in  $D$
  - Let  $y_{[i]}(\vec{x})$  or  $y_{[i]}$  be the label of  $\vec{x}_{[i]}$

- Nearest neighbor hypothesis

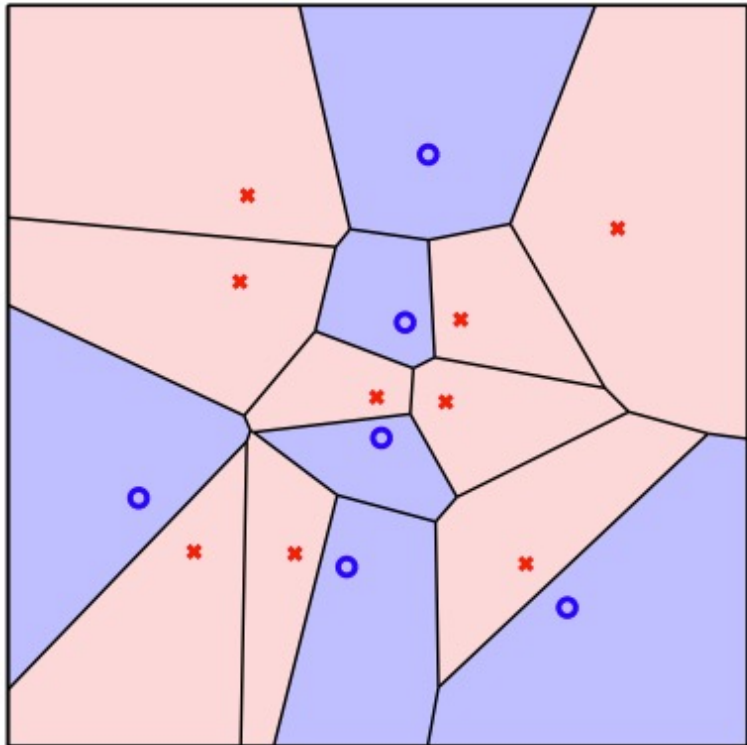
$$g(\vec{x}) = y_{[1]}(\vec{x})$$

- $k$ -nearest neighbor (K-NN)

$$g(\vec{x}) = \text{sign}\left(\sum_{i=1}^k y_{[i]}(\vec{x})\right)$$

# 1-Nearest Neighbor

$g(\vec{x})$  looks like a Voronoi diagram



- Properties of 1-Nearest Neighbor (NN)
  - No training is needed
  - Good interpretability
  - In-sample error  $E_{in} = 0$
  - VC dimension is  $\infty$
- This seems to imply bad learning models from what we talk about so far? Why we care?
- What we really care about is  $E_{out}$ 
  - VC analysis:  $E_{out} \leq E_{in} + \text{Generalization error}$ 
    - We can infer  $E_{out}$  through  $E_{in}$  and model complexity
  - NN has nice guarantees outside of VC analysis

# Properties of Nearest Neighbors

- 1-NN

- Given mild conditions, for 1-nearest neighbor, when  $N \rightarrow \infty$ , with high probability,

$$E_{out} \leq 2E_{out}^*$$

- That is, we can not infer  $E_{out}$  from  $E_{in}$ , but we know it cannot be much worse than the **best anyone can do**.

- k-NN:

- Tuning  $k$  moderates the tradeoff of **generalization vs approximation**



# Radial Basis Function

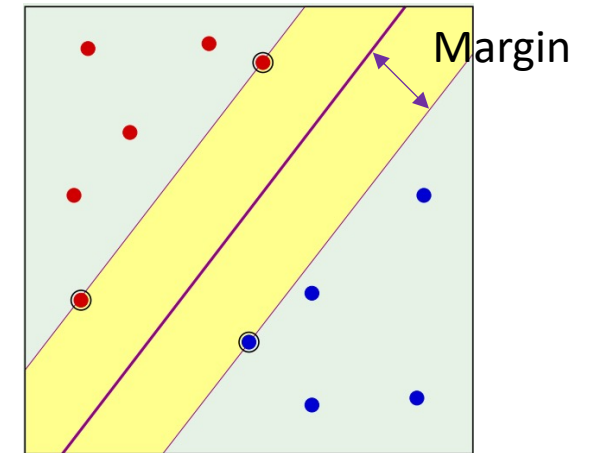
- $h(\vec{x}) = \sum_{k=1}^K w_k \phi\left(\frac{\|\vec{x} - \vec{\mu}_k\|}{r}\right)$
- Connection to linear models
  - Parametric RBF is essentially linear model with nonlinear transformation
- Connection to nearest neighbor
  - RBF is based on the similarity to a set of points
- Connection to SVM with RBF Kernel
  - Using K representative points vs. using support vectors
- Connection to Neural Networks
  - RBF can be graphically represented as a one-hidden layer network

# Support Vector Machines

- Goal: Find the **max-margin** linear separator
- If the data is **linearly separable**
  - **Hard-Margin SVM**

$$\begin{array}{ll} \text{minimize}_{\vec{w}, b} & \frac{1}{2} \vec{w}^T \vec{w} \\ \text{subject to} & y_n (\vec{w}^T \vec{x}_n + b) \geq 1, \forall n \end{array}$$

- $g(\vec{x}) = \text{sign}(\vec{w}^{*T} \vec{x} + b^*)$

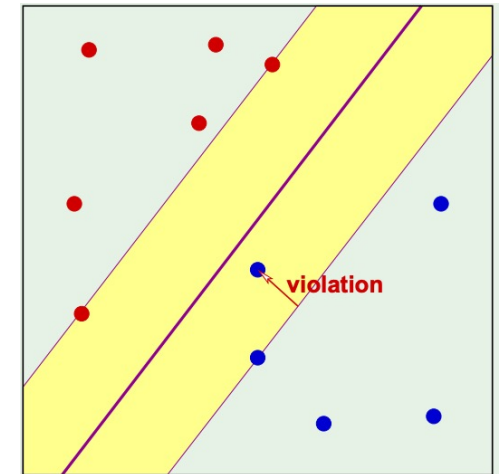


- If the data is not linearly separable
  - **Soft-margin SVM**
  - Nonlinear transformation – **Dual Formulation** and **Kernel Tricks**

# Soft-Margin SVM

- For each point  $(\vec{x}_n, y_n)$ , we allow a deviation  $\xi_n \geq 0$ 
  - The constraint becomes:  $y_n(\vec{w}^T \vec{x}_n + b) \geq 1 - \xi_n$
  - We add a penalty for each deviation: Total penalty  $C \sum_{n=1}^N \xi_n$

$$\begin{aligned} &\text{minimize}_{\vec{w}, b, \xi} \quad \frac{1}{2} \vec{w}^T \vec{w} + C \sum_{n=1}^N \xi_n \\ &\text{subject to} \quad y_n(\vec{w}^T \vec{x}_n + b) \geq 1 - \xi_n, \forall n \\ &\quad \quad \quad \xi_n \geq 0, \forall n \end{aligned}$$



Remarks:

- $C$  is a hyper-parameter we can choose, e.g., using validation
  - Larger  $C \Rightarrow$  less tolerable to noise  $\Rightarrow$  smaller margin
- Soft-margin SVM is still a Quadratic Program, with efficient solvers

# Primal-Dual Formulations of Hard-Margin SVM

- Primal

$$\begin{aligned} & \text{minimize}_{\vec{w}, b} \quad \frac{1}{2} \vec{w}^T \vec{w} \\ & \text{subject to} \quad y_n (\vec{w}^T \vec{x}_n + b) \geq 1, \forall n \end{aligned}$$

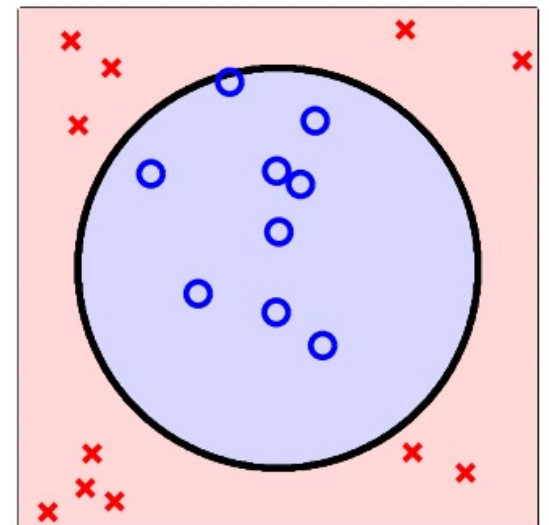
Given optimal  $\vec{\alpha}^*$ :

- $\vec{w}^* = \sum_{\alpha_n^* > 0} \alpha_n^* y_n \vec{x}_n$
- Find a  $\alpha_n^* > 0$ ,  $b^* = y_n - \vec{x}_n^T \vec{w}^*$

- Dual + Kernel Trick

$$\begin{aligned} & \text{maximize}_{\vec{\alpha}} \quad \sum_{n=1}^N \alpha_n - \frac{1}{2} \sum_{n=1}^N \sum_{m=1}^N \alpha_n \alpha_m y_n y_m K(\vec{x}_n, \vec{x}_m) \\ & \text{subject to} \quad \sum_{n=1}^N \alpha_n y_n = 0 \\ & \quad \quad \quad \alpha_n \geq 0, \forall n \end{aligned}$$

- Both can be efficiently solved using QP solver.
- We can infer the solution from one to the other



# Recover $(\vec{w}^*, b^*)$ from $\vec{\alpha}^*$ with Kernel Tricks

- Note that  $\vec{\alpha}^*$  is solved in the  $\vec{z}$  space
  - $\vec{w}^* = \sum_{\alpha_n^* > 0} \alpha_n^* y_n \Phi(\vec{x}_n)$
  - Find a  $\alpha_n^* > 0$ ,  $b^* = y_n - \vec{w}^{*T} \Phi(\vec{x}_n)$
  - We want to avoid the transformation!
- Let's look at the hypothesis
  - $g(\vec{x}) = \text{sign}(\vec{w}^{*T} \Phi(\vec{x}) + b^*)$

$$\begin{aligned}\vec{w}^{*T} \Phi(\vec{x}) &= \left( \sum_{\alpha_n^* > 0} \alpha_n^* y_n \Phi(\vec{x}_n) \right)^T \Phi(\vec{x}) \\ &= \sum_{\alpha_n^* > 0} \alpha_n^* y_n \Phi(\vec{x}_n)^T \Phi(\vec{x}) \\ &= \sum_{\alpha_n^* > 0} \alpha_n^* y_n K(\vec{x}_n, \vec{x})\end{aligned}$$

$$\begin{aligned}b^* &= y_n - \vec{w}^{*T} \Phi(\vec{x}_n) \\ &= y_n - \left( \sum_{\alpha_m^* > 0} \alpha_m^* y_m \Phi(\vec{x}_m) \right)^T \Phi(\vec{x}_n) \\ &= y_n - \sum_{\alpha_m^* > 0} \alpha_m^* y_m K(\vec{x}_m, \vec{x}_n)\end{aligned}$$

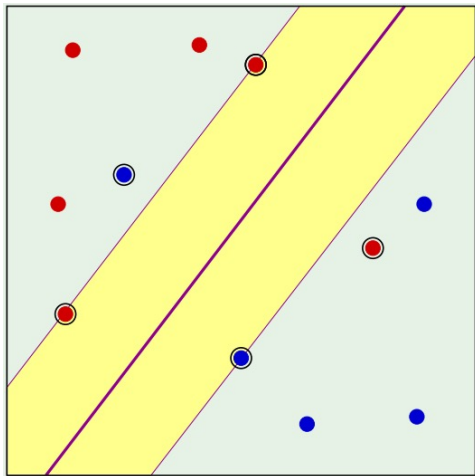
- Still can be computed in the  $\vec{x}$  space!

# Kernel Functions

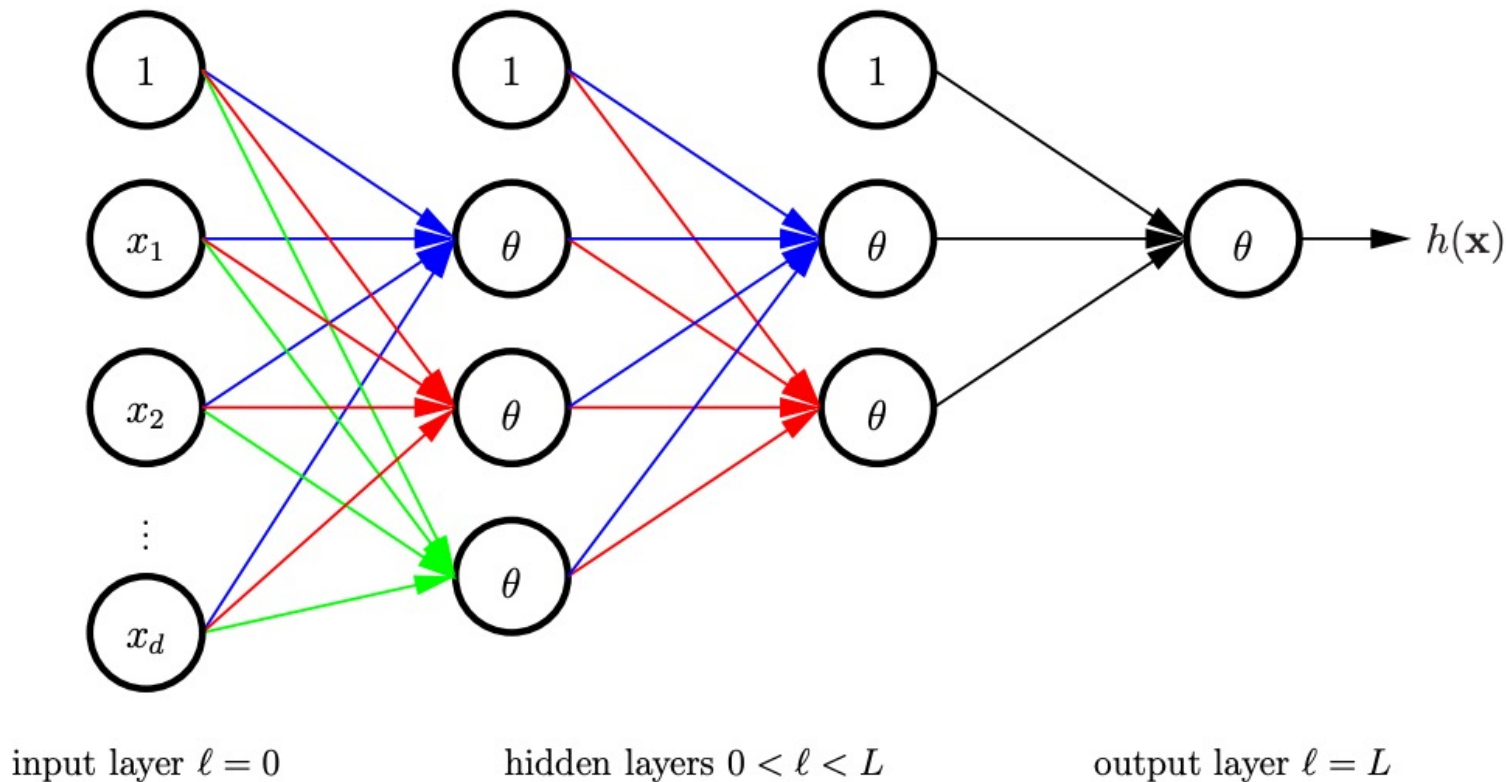
- Q-th order Polynomial kernel  $K_{\Phi_Q}(\vec{x}, \vec{x}') = (1 + \vec{x}^T \vec{x}')^Q$ 
  - The corresponding  $\Phi(x)$ : Q-th order polynomial transform
- Gaussian RBF Kernel  $K_{\Phi}(\vec{x}, \vec{x}') = e^{-\gamma \|\vec{x} - \vec{x}'\|^2}$ 
  - The corresponding  $\Phi(x) = e^{-x^2} \left( 1, \sqrt{\frac{2}{1}} x, \sqrt{\frac{2^2}{2!}} x^2, \dots \right)$
- When we plug in  $K(\vec{x}, \vec{x}')$  in dual SVM
  - We are finding the **max-margin** separator in an **infinite dimensional** space
  - Seems to introduce infinite generalization error?
    - Maximizing margin help mitigate this issue
    - The number of support vectors provides indicators on the generalization

# Support Vectors

- $\alpha_n^* > 0 \Rightarrow (\vec{x}_n, y_n)$  is a support vector
  - $y_n(\vec{w}^{*T} \vec{x}_n + b^*) = 1 - \xi_n$
- SVM classifier can be expressed using support vectors
  - $g(\vec{x}) = \text{sign}(\sum_{\alpha_n^* > 0} \alpha_n^* y_n K(\vec{x}_n, \vec{x}) + b^*)$
- Connection to generalization error through **LOOCV**



# Neural Networks



$\theta$ : **activation function**  
(Specify the “activation” of the neuron)

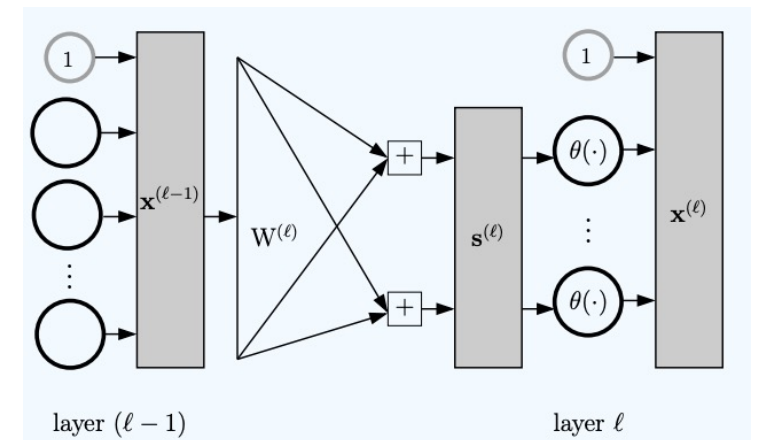
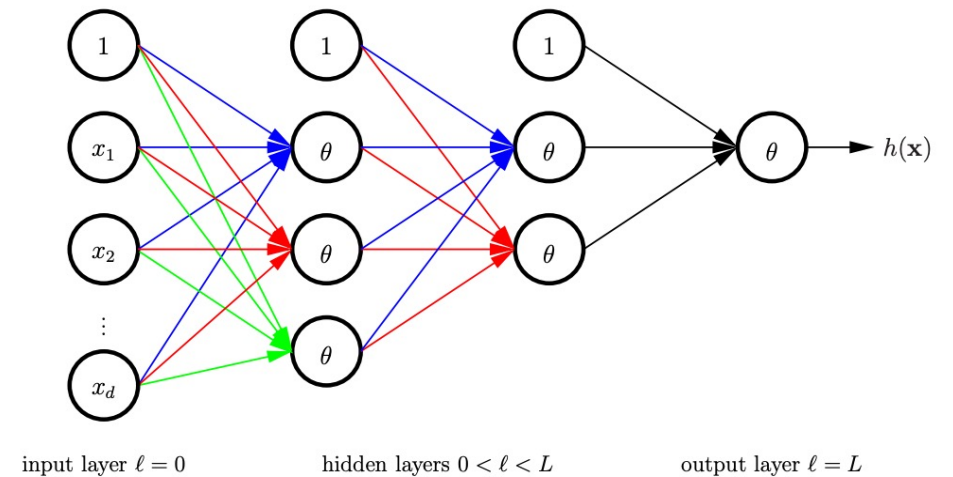


We mostly focus on **feed-forward** network structure



# Notations of Neural Networks (NN)

- Notations:
  - $\ell = 0$  to  $L$ : layer
  - $d^{(\ell)}$ : dimension of layer  $\ell$
  - $\vec{x}^{(\ell)}$ : the nodes in layer  $\ell$
  - $w_{i,j}^{(\ell)}$ : weights; characterize hypothesis in NN
  - $s_j^{(\ell)} = \sum_{i=0}^{d^{(\ell-1)}} w_{i,j}^{(\ell)} x_i^{(\ell-1)}$ : linear signals
  - $\theta$ : activation function
    - $x_j^{(\ell)} = \theta(s_j^{(\ell)})$



# Forward Propagation (evaluate $h(\vec{x})$ )

- A Neural network hypothesis  $h$  is characterized by  $\{w_{i,j}^{(\ell)}\}$
- How to evaluate  $h(\vec{x})$ ?

$$\mathbf{x} = \mathbf{x}^{(0)} \xrightarrow{w^{(1)}} \mathbf{s}^{(1)} \xrightarrow{\theta} \mathbf{x}^{(1)} \xrightarrow{w^{(2)}} \mathbf{s}^{(2)} \xrightarrow{\theta} \mathbf{x}^{(2)} \dots \xrightarrow{w^{(L)}} \mathbf{s}^{(L)} \xrightarrow{\theta} \mathbf{x}^{(L)} = h(\mathbf{x}).$$

Forward propagation to compute  $h(\mathbf{x})$ :

```
1:  $\mathbf{x}^{(0)} \leftarrow \mathbf{x}$                                 [Initialization]
2: for  $\ell = 1$  to  $L$  do                                [Forward Propagation]
3:    $\mathbf{s}^{(\ell)} \leftarrow (\mathbf{W}^{(\ell)})^T \mathbf{x}^{(\ell-1)}$ 
4:    $\mathbf{x}^{(\ell)} \leftarrow \begin{bmatrix} 1 \\ \theta(\mathbf{s}^{(\ell)}) \end{bmatrix}$ 
5: end for
6:  $h(\mathbf{x}) = \mathbf{x}^{(L)}$                                 [Output]
```

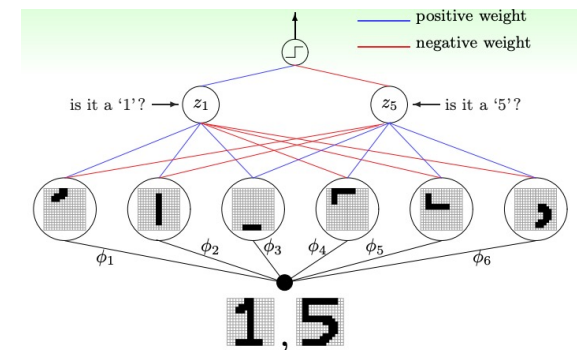
Given weights  $w_{i,j}^{(\ell)}$  and  $\vec{x}^{(0)} = \vec{x}$ , we can calculate all  $\vec{x}^{(\ell)}$  and  $\vec{s}^{(\ell)}$  through forward propagation.

# Backpropagation Algorithm

- Recall that  $\frac{\partial e_n(W)}{\partial w_{i,j}^{(\ell)}} = \delta_j^{(\ell)} x_i^{(\ell-1)}$
- Backpropagation Algorithm
  - Initialize  $w_{i,j}^{(\ell)}$  randomly
  - For  $t = 1$  to  $T$ 
    - Randomly pick a point from  $D$  (for stochastic gradient descent)
    - Forward propagation: Calculate all  $x_i^{(\ell)}$  and  $s_i^{(\ell)}$
    - Backward propagation: Calculate all  $\delta_j^{(\ell)}$
    - Update the weights  $w_{i,j}^{(\ell)} \leftarrow w_{i,j}^{(\ell)} - \eta \delta_j^{(\ell)} x_i^{(\ell-1)}$
- Return the weights

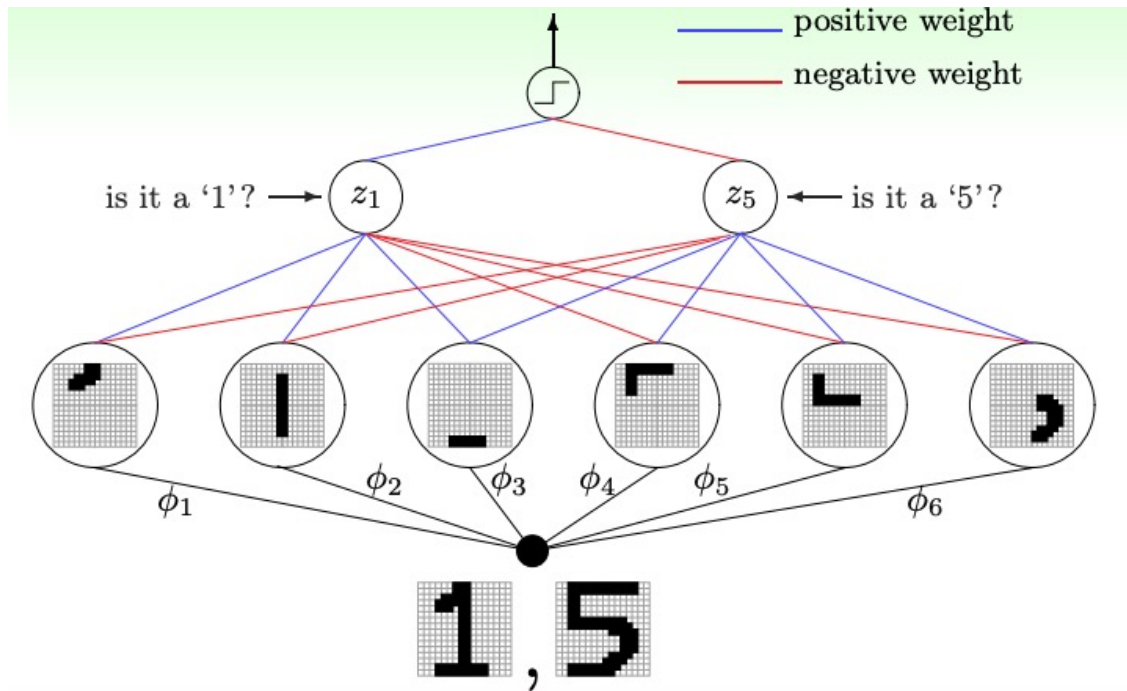
# Discussion

- Backpropagation
  - gradient descent with efficient gradient computation
  - $E_{in}$  is **not convex** in weights
  - Gradient descent doesn't guarantee to converge to global optimal
    - Run it many times, each with a different initialization (initialization matters)
- Regularization
  - Weight-based regularization, early stopping, dropout, adding noise, etc
- Deep learning
  - Neural Networks with many layers
  - Enable hierarchical representations of data



# Deep Neural Network

- “Shallow” neural network is powerful (universal approximation theorem holds with a single hidden layer). Why “deep” neural networks?



Each layer captures **features** of the previous layers.

We can use “raw data” (e.g., pixels of an image) as input. The hidden layer are extracting the **features**.

Design different **network architectures** to incorporate domain knowledge.

# Some Techniques in Improving Deep Learning

- Regularization to mitigate overfitting
  - Weight-based, early stopping, dropout, etc
- Incorporating domain knowledges
  - Network architectures (e.g., Convolutional Neural Nets)
- Improving computation with huge amount of data
  - Hardware architecture to improve parallel computation
- Improving gradient-based optimization
  - Choosing better **initialization** points

# Practice Questions