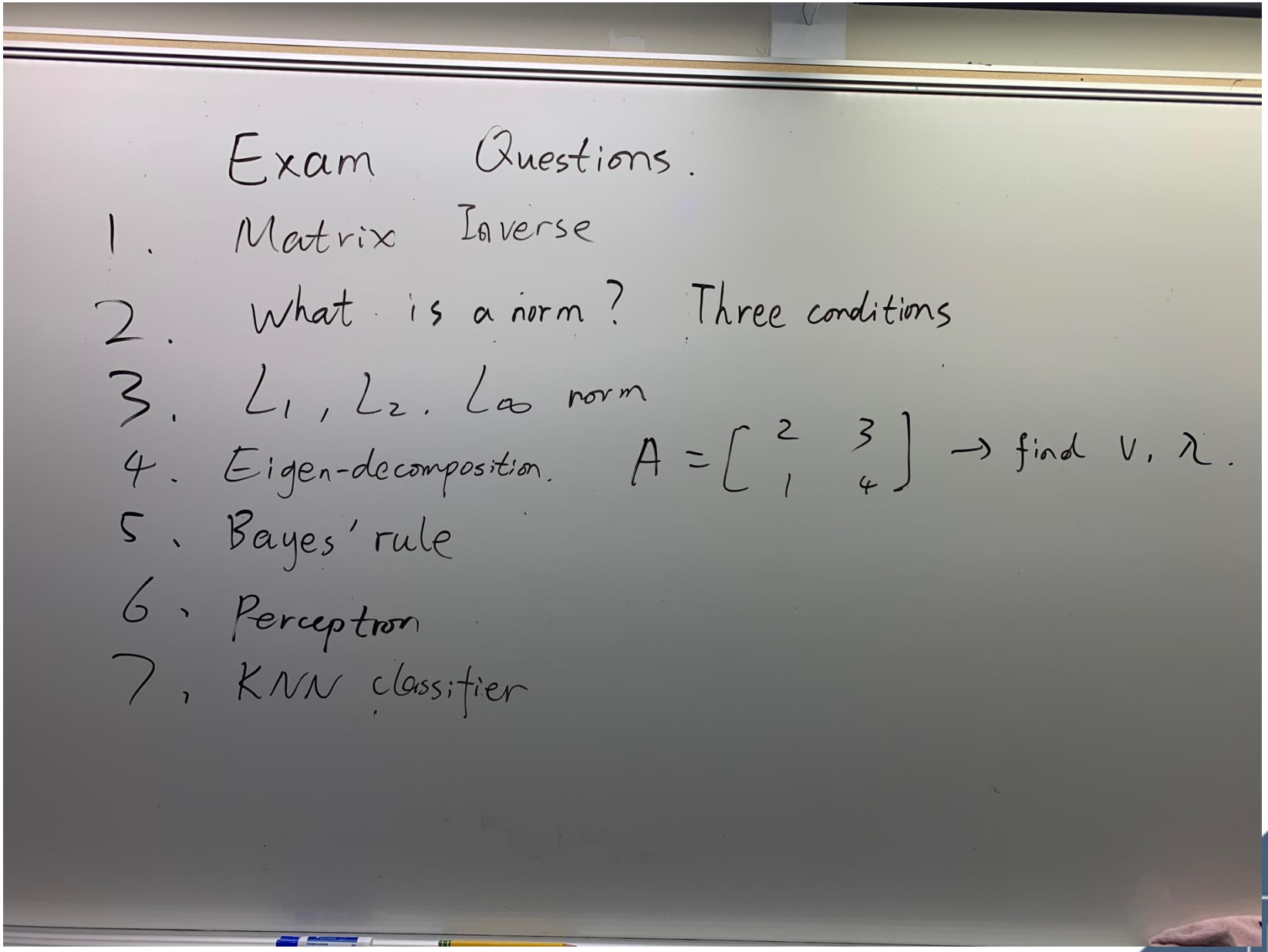
CSE 176 Introduction to Machine Learning

Midterm Review


Exam Questions.
1. Matrix Inverse
2. What is a norm? Three conditions
3. L_1, L_2. L_∞ norm
4. Eigen-decomposition. $A = \begin{bmatrix} 2 & 3 \\ 1 & 4 \end{bmatrix}$ → find V, λ.
5. Bayes' rule
6. Perceptron
7. KNN classifier

8. k-means / medians / modes objectives.

9. mean-shift update $x \leftarrow$

10. GMM. how to update soft assignment (E-step) and $\{\mu, \sigma, p\}$ (M-step) $\sum p_i = 1$.

hard assignment $\{\mu\}$

11. adjacency matrix A. Degree matix D. Laplacian Matrix. $D - A = L$

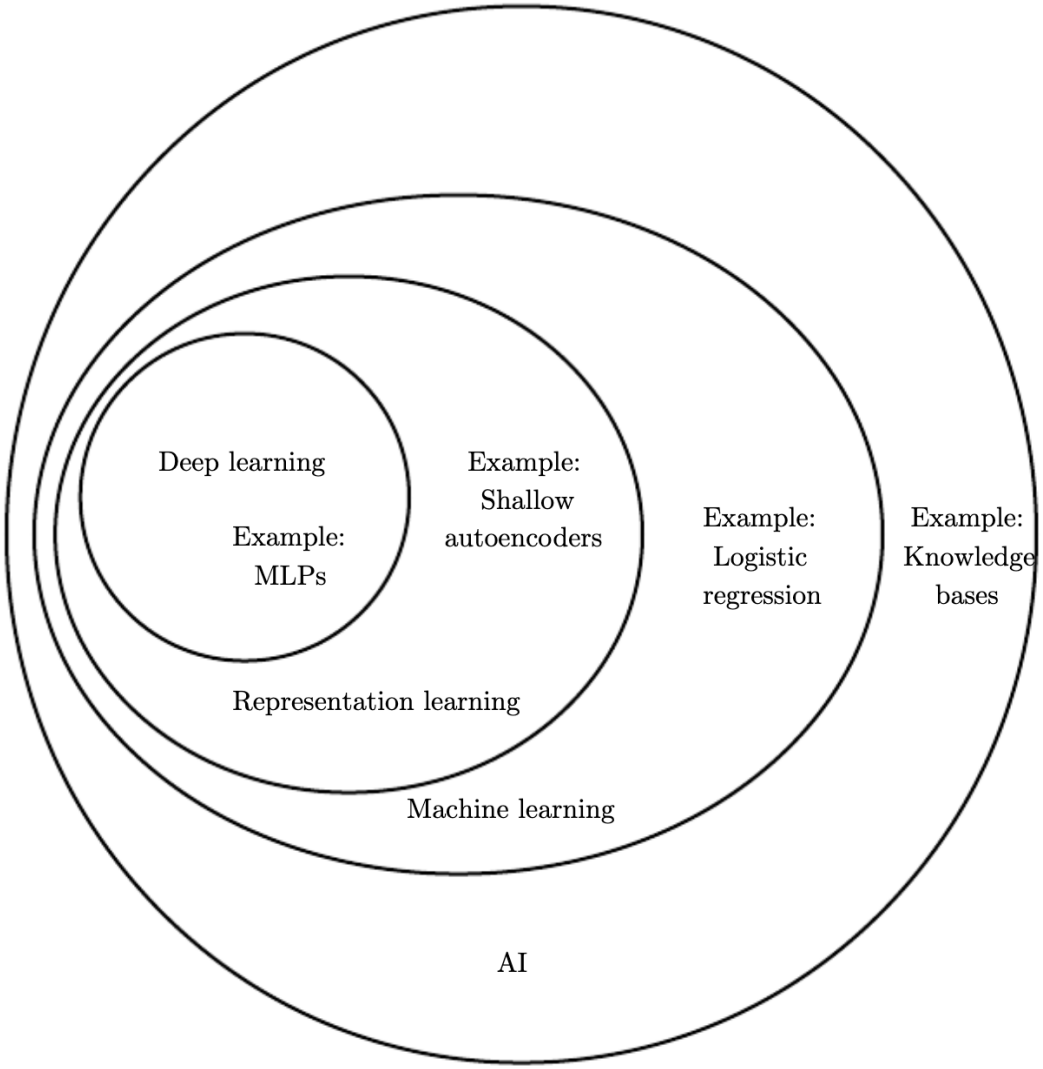
12. Normalized Cut objective

13. PCA (mean, covariance, eigendecom)

14. how to choose learning rate.

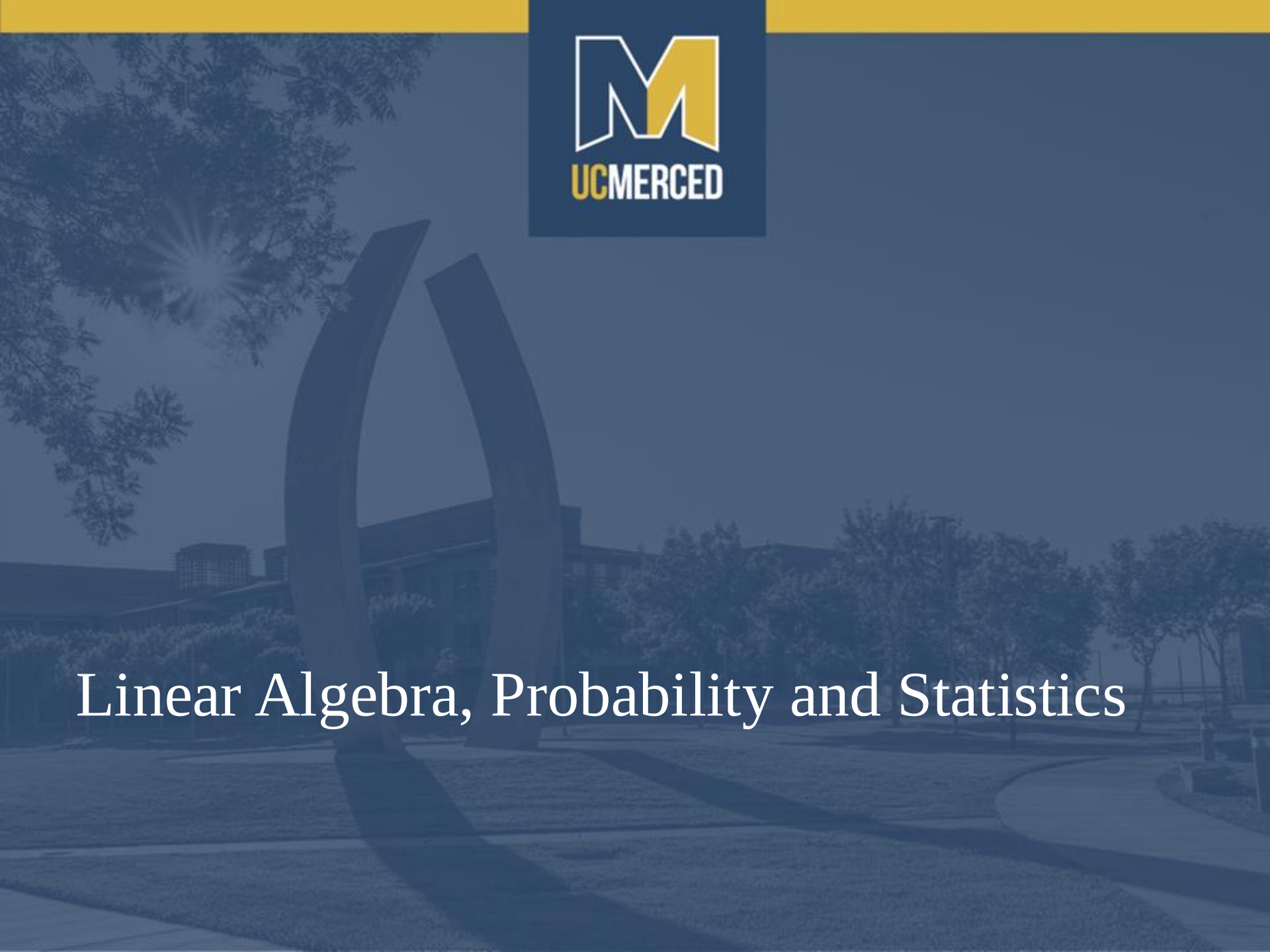
15. Back Propagation.

Different AI systems



Major Types of machine learning

- Supervised learning: Given pairs of input-output, learn to map the input to output
  - Image classification
  - Speech recognition
  - Regression (continuous output)
- Unsupervised learning: Given unlabeled data, uncover the underlying structure or distribution of the data
  - Clustering
  - Dimensionality reduction
- Reinforcement learning: training an agent to make decisions within an environment to maximize a cumulative reward
  - Game playing (e.g., AlphaGo)
  - Robot control



Linear Algebra, Probability and Statistics

Matrix inverse

- For a 2x2 matrix:

$$A = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$$

- The inverse is :

- Quiz: find the inv $A^{-1} = \frac{1}{ad - bc} \begin{pmatrix} d & -b \\ -c & a \end{pmatrix}$

- Answer:

$$A = \begin{bmatrix} 3 & -1 & 1 \\ 2 & 0 & 2 \end{bmatrix}, B = \begin{bmatrix} 2 & 2 & 1 \\ 0 & 1 & 1 \end{bmatrix}$$

$$AB^T = \begin{bmatrix} 3 & -1 & 1 \\ 2 & 0 & 2 \end{bmatrix} \cdot \begin{bmatrix} 2 & 0 \\ 2 & 1 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} 5 & 0 \\ 6 & 2 \end{bmatrix}$$

$$(AB^T)^{-1} = \frac{1}{5 \cdot 2 - 0 \cdot 6} \cdot \begin{bmatrix} 2 & 0 \\ -6 & 5 \end{bmatrix} = \frac{1}{10} \cdot \begin{bmatrix} 2 & 0 \\ -6 & 5 \end{bmatrix} = \begin{bmatrix} 0.2 & 0 \\ -0.6 & 0.5 \end{bmatrix}$$

Norms

- Used for measuring the size of a vector
- Norms map vectors to non-negative values
- Norm of vector $\boldsymbol{x}=[x_1,..,x_n]^T$ is distance from origin to $\boldsymbol{x}$
  - It is any function f that satisfies:

$$f(\boldsymbol{x})=0 \Rightarrow \boldsymbol{x}=0$$

$$f(\boldsymbol{x}+\boldsymbol{y}) \leq f(\boldsymbol{x})+f(\boldsymbol{y}) \qquad \text{Triangle Inequality}$$

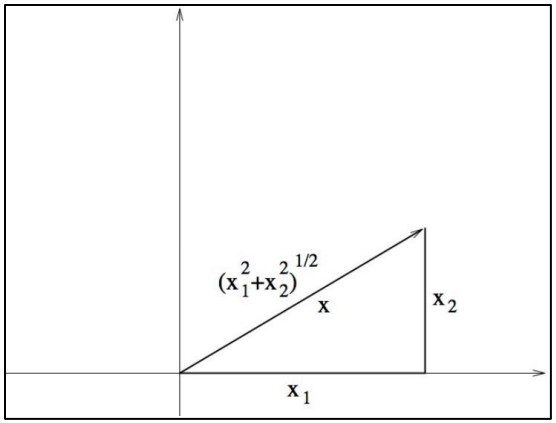
$$\forall \alpha \in R \quad f(\alpha \boldsymbol{x})=|\alpha| f(\boldsymbol{x})$$

Slide from S. Srihari

L^p Norm

- Definition:

$$\left\|\boldsymbol{x}\right\|_p = \left(\sum_i \left|x_i\right|^p\right)^{\frac{1}{p}}$$



  - L^2 Norm
    - Called Euclidean norm
      - Simply the Euclidean distance between the origin and the point $\boldsymbol{x}$
      - written simply as $||\boldsymbol{x}||$
      - Squared Euclidean norm is same as $\boldsymbol{x}^T\boldsymbol{x}$
  - L^1 Norm
    - Sum of absolute value for each x_i
  - L^∞ Norm

$$\left\|\boldsymbol{x}\right\|_\infty = \max_i \left|x_i\right|$$

- Called max norm

Eigendecomposition

- Suppose that matrix A has n linearly independent eigenvectors $\{\boldsymbol{v}^{(1)},..,\boldsymbol{v}^{(n)}\}$ with eigenvalues $\{\lambda_1,..,\lambda_n\}$
- Concatenate eigenvectors to form matrix V
- Concatenate eigenvalues to form vector

  $\lambda=[\lambda_1,..,\lambda_n]$
- Eigendecomposition of A is given by

$$A=V\text{diag}(\lambda)V^{-1}$$

Slide from S. Srihari

Chain rule of conditional probability

- Any probability distribution over many variables can be decomposed into conditional distributions over only one variable

$$P(\mathrm{x}^{(1)}, \ldots, \mathrm{x}^{(n)}) = P(\mathrm{x}^{(1)}) \Pi_{i=2}^{n} P(\mathrm{x}^{(i)} \mid \mathrm{x}^{(1)}, \ldots, \mathrm{x}^{(i-1)})$$

- An example with three variables

$$\begin{aligned} P(\mathrm{a}, \mathrm{b}, \mathrm{c}) &= P(\mathrm{a} \mid \mathrm{b}, \mathrm{c}) P(\mathrm{b}, \mathrm{c}) \\ P(\mathrm{b}, \mathrm{c}) &= P(\mathrm{b} \mid \mathrm{c}) P(\mathrm{c}) \\ P(\mathrm{a}, \mathrm{b}, \mathrm{c}) &= P(\mathrm{a} \mid \mathrm{b}, \mathrm{c}) P(\mathrm{b} \mid \mathrm{c}) P(\mathrm{c}) \end{aligned}$$

Slide from S. Srihari

Independence and conditional independence

- Independence: $\boxed{\mathrm{x} \perp \mathrm{y}}$
  - Two variables x and y are independent if their probability distribution can be expressed as a product of two factors, one involving only x and the other involving only y

$$\forall x \in \mathrm{x}, y \in \mathrm{y},\ p(\mathrm{x} = x, \mathrm{y} = y) = p(\mathrm{x} = x)p(\mathrm{y} = y)$$

- Conditional Independence: $\boxed{\mathrm{x} \perp \mathrm{y} \mid \mathrm{z}}$
  - Two variables x and y are independent given variable z, if the conditional probability distribution over x. and y factorizes in this way for every z

$$\forall x \in \mathrm{x}, y \in \mathrm{y}, z \in \mathrm{z},\ p(\mathrm{x} = x, \mathrm{y} = y \mid \mathrm{z} = z) = p(\mathrm{x} = x \mid \mathrm{z} = z)p(\mathrm{y} = y \mid \mathrm{z} = z)$$

Slide from S. Srihari

Bayes's rule

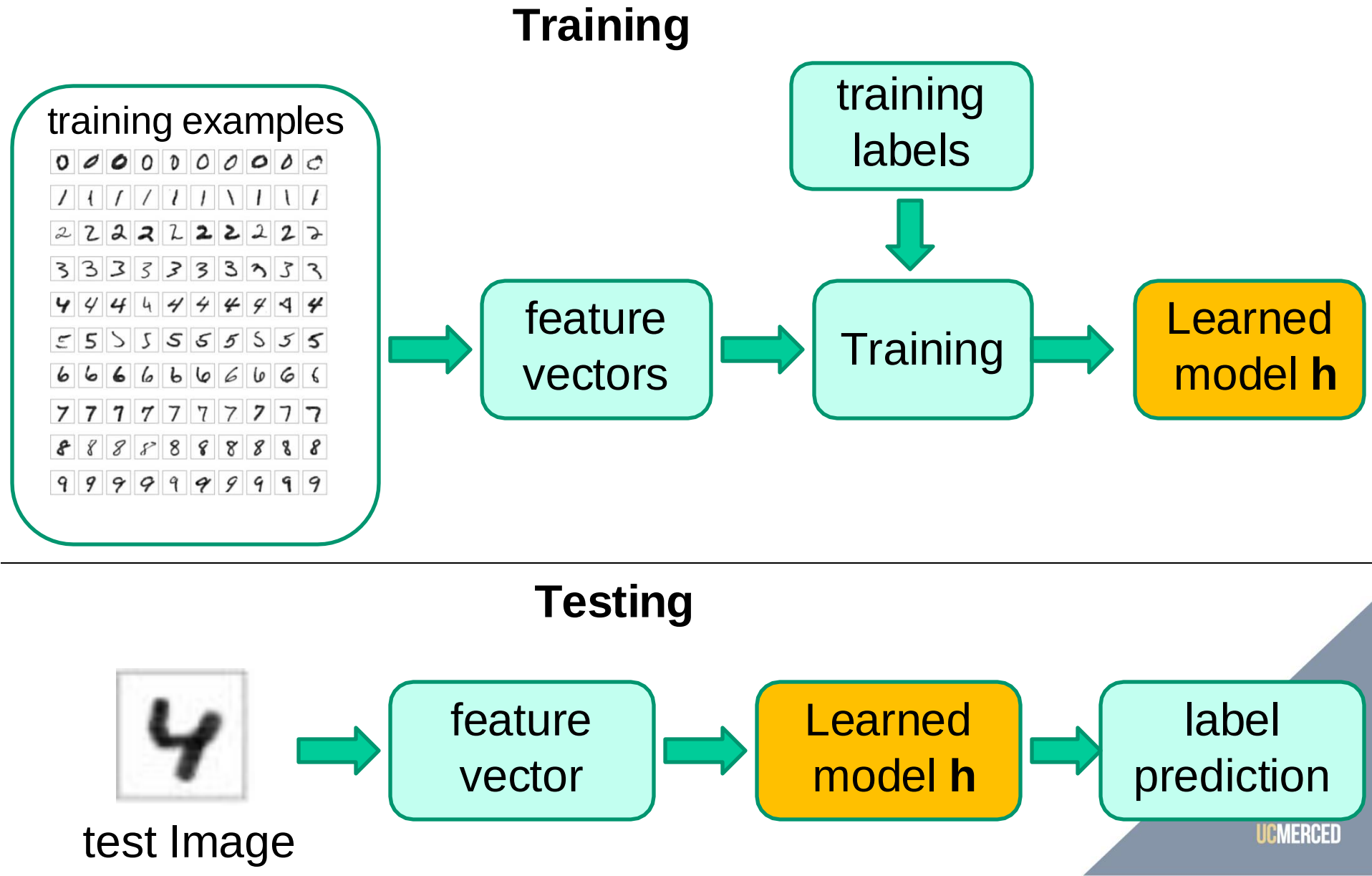
- **Bayes' theorem** (alternatively **Bayes' law** or **Bayes' rule**), named after Thomas Bayes, describes the probability of an event, based on prior knowledge of conditions that might be related to the event.
- For example, if the risk of health problems is known to increase with age, Bayes' theorem allows the risk to an individual of a known age to be assessed more accurately by conditioning it relative to their age.

$$P(A|B) = \frac{P(A \cap B)}{P(B)} = \frac{P(A) * P(B|A)}{P(B)}$$

Slide from S. Srihari

Supervised Learning: Classification and Regression

Training/Testing Phases Illustrated



Loss function

- Training dataset of I pairs of input/output examples

$$\{\mathbf{x}_n, \mathbf{y}_n\}_{n=1}^{N}$$

- Loss function or cost function measures how bad model is:

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} \Sigma_n L(\mathbf{y}_n, \mathbf{h}(\mathbf{w}, \mathbf{x}_n))$$

- $\boldsymbol{\theta}$ is also a common notation for weights

Supervised ML algorithm

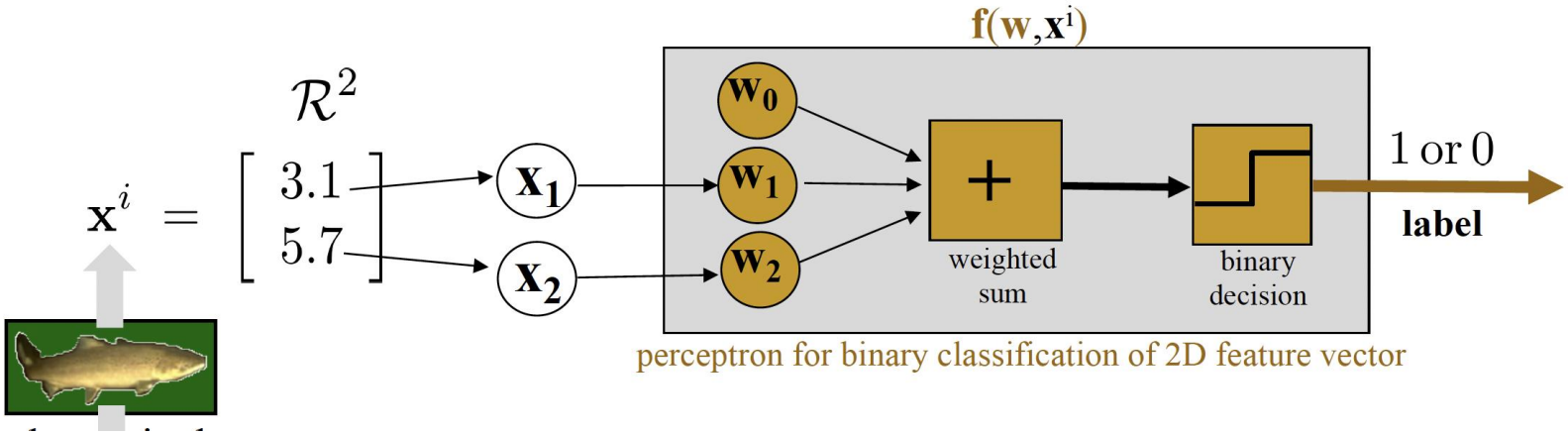
1. A model $h(\mathbf{x}; \boldsymbol{\Theta})$ (hypothesis class) with parameters $\boldsymbol{\Theta}$. A particular value of $\boldsymbol{\Theta}$ determines a particular hypothesis in the class.
   Ex: for linear models, $\boldsymbol{\Theta}$ = slope w_1 and intercept w_0.
2. A *loss function* $L(\cdot, \cdot)$ to compute the difference between the desired output (label) y_n and our prediction to it $h(\mathbf{x}_n; \boldsymbol{\Theta})$. *Approximation error (loss)*:

$$E(\boldsymbol{\Theta}; \mathcal{X}) = \sum_{n=1}^{N} L(y_n, h(\mathbf{x}_n; \boldsymbol{\Theta})) = \text{sum of errors over instances}$$

   Ex: 0/1 loss for classification, squared error for regression.
3. An optimization procedure (learning algorithm) to find parameters $\boldsymbol{\Theta}^*$ that minimize the error:

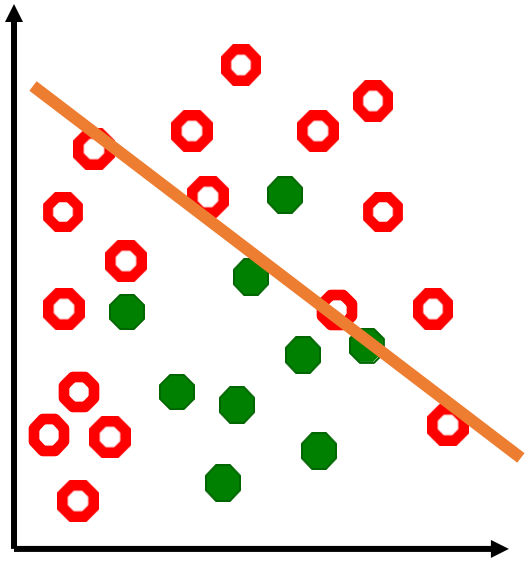
$$\boldsymbol{\Theta}^* = \arg\min_{\boldsymbol{\Theta}} E(\boldsymbol{\Theta}; \mathcal{X})$$

Linear classifier example: *perceptron*



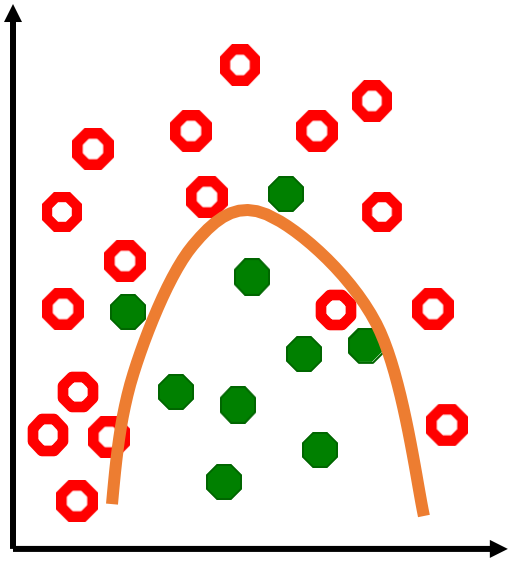
Underfitting → Overfitting

underfitting



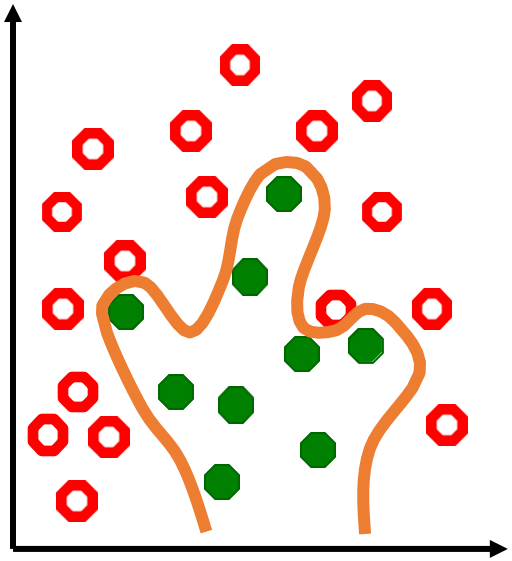
- high training error
- high test error

"just right"



- low training error
- low test error

overfitting



- low training error
- high test error

Model selection and generalization

- Machine learning problems (classification, regression and others) are typically ill-posed : the observed data is finite and does not uniquely determine the classification or regression function.
- How to choose the right inductive bias, in particular the right hypothesis class? This is the *model selection* problem.

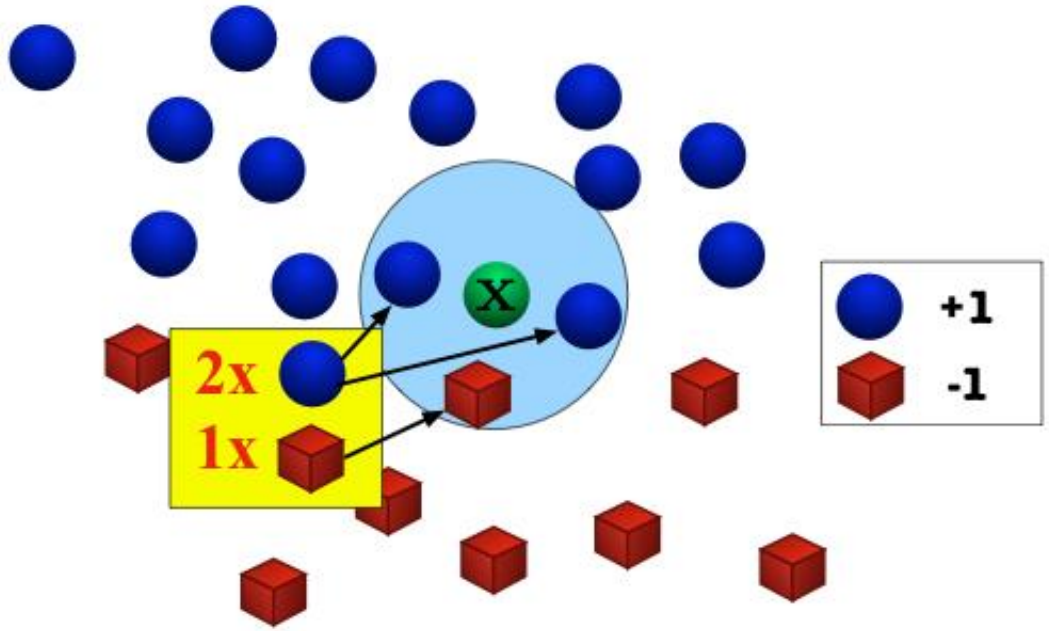
Cross Validation

- Training set:
  - Used to train, i.e., to fit a hypothesis $h \in H_i$.
  - Optimize parameters of h given the model structure and hyperparameters.
  - Usually done with an optimization algorithm (the learning algorithm).
- Validation set:
  - Used to minimize the generalization error.
  - Optimize hyperparameters or model structure.
  - Usually done with a "grid search". Ex: try all values of $H \in \{10, 50, 100\}$ and $\lambda \in \{10^{-5}, 10^{-3}, 10^{-1}\}$.
- Test set:
  - Used to report the generalization error.
  - We optimize nothing on it, we just evaluate the final model on it

KNN Classifier

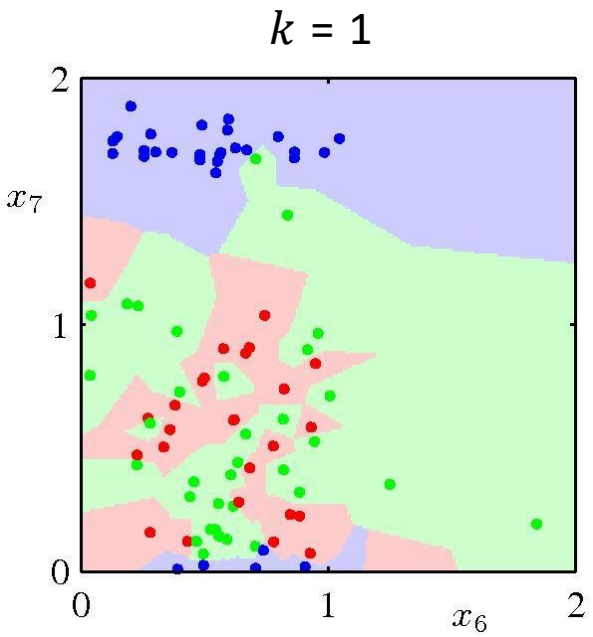
K nearest neighbor algorithm

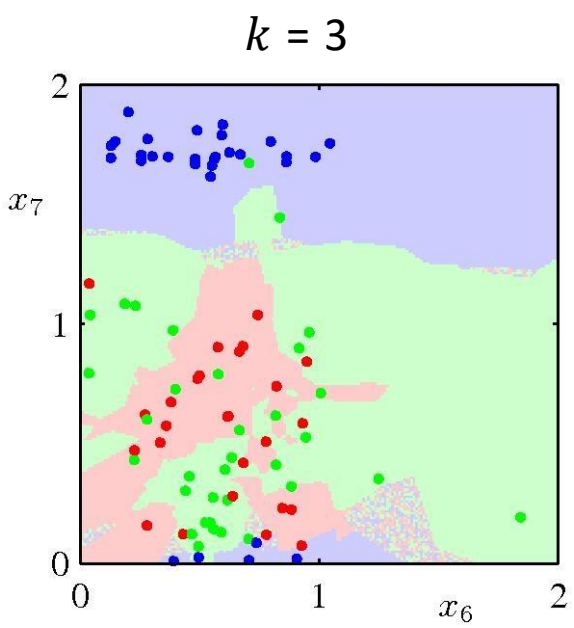
- Nearest neighbor often instable (noise)
- For a test input ***x***, assign the most common label amongst its k most similar training inputs

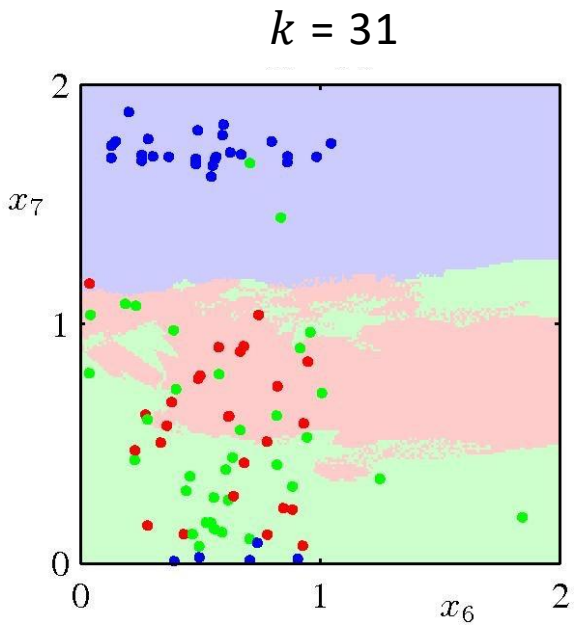


Effect of K

- Which partition do you prefer? Why?







- K controls the degree of smoothing.
- What if K=N(number of data points)?

UCMERCED

Choosing K

- How should we choose K?
  - Select K with highest test accuracy
- Can we simply split to training and testing set?
- Solution: split data into training, validation and test sets
  - Training set: compute nearest neighbour
  - Validation set: optimize hyperparameters such as K
  - Test set: measure performance

[4] The table below shows the test set for a **1-nearest-neighbor classifier** that uses Manhattan distance, i.e., the distance between two points at coordinates p and q is $|p - q|$. The only attribute, X, is real-valued, and the label, Y, has two classes, 0 and 1. Suppose a *subset* containing $n \leq 8$ examples is selected from this set to train the classifier, and the accuracy of the classifier is 100 percent when tested on this set (with *all* 8 examples). What is the *smallest* possible value for n? In case of ties in distance, use the example with smallest X value as the neighbor.

X	-5	-4	-1	0	1	3	4	8
Y	0	1	0	0	0	0	0	1

A. 2
B. 3
C. 4
D. 5
E. 6

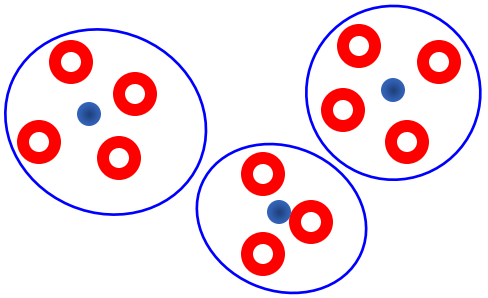
KNN for high-dimensional data

- Can we use KNN classifier for high-dimensional data?
- Assumption for KNN classifier to work:
  - K nearest neighbors are ***nearby***
- Are K nearest neighbors nearby for d>>0?

Clustering

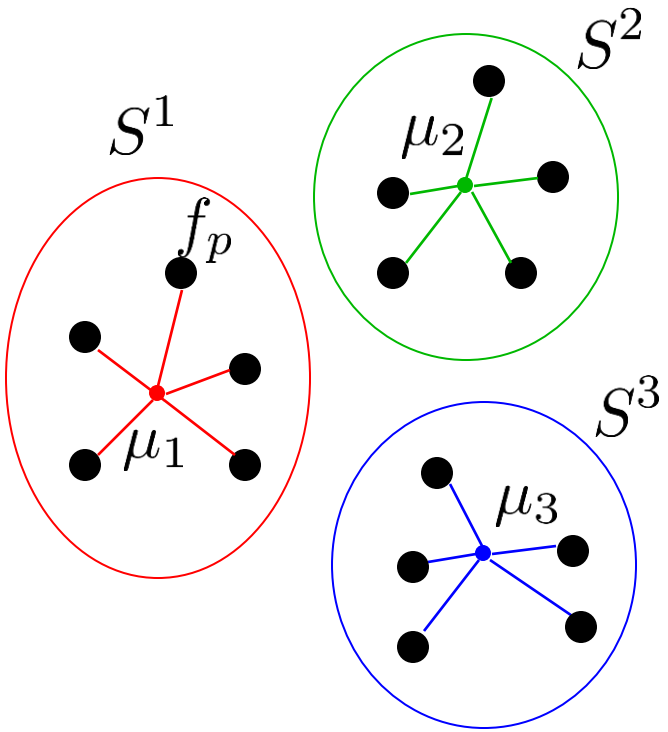
K-means Clustering: Algorithm

- Initialization step
  1. pick ***K*** cluster centers randomly
  2. assign each sample to its closest center



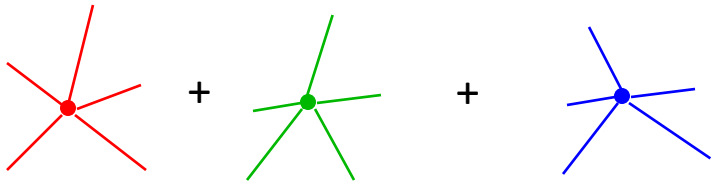
- Iteration steps
  1. compute centers as cluster means $\mu_k = \frac{1}{|S^k|} \sum_{p \in S^k} f_p$
  2. re-assign each sample to the closest mean
- Iterate until clusters stop changing

K-means objective



μ_k : extra parameters (means)

$$E(S, \mu) =$$



$$= \sum_{k=1}^{K} \sum_{p \in S^k} \|f_p - \mu_k\|^2$$

(SSD)

Squared distance as log-likelihood

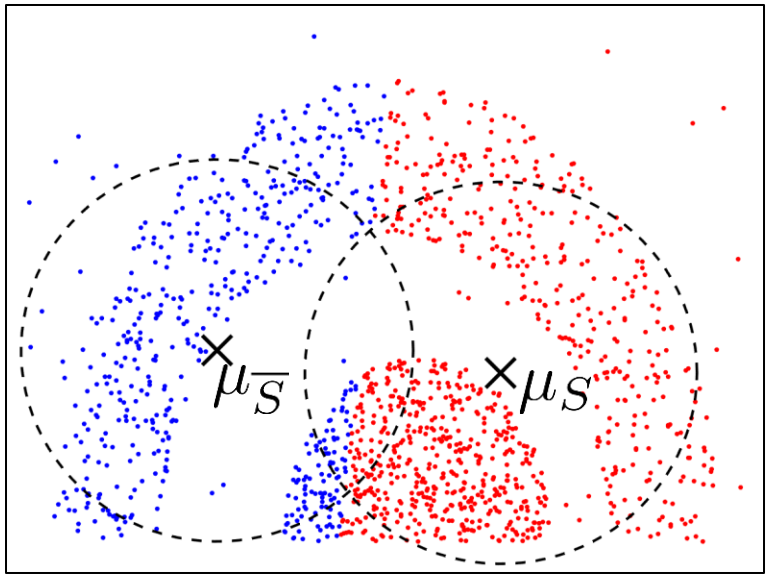
Assume K=2, $\Omega = S \cup \bar{S}$

$$\sum_{p \in S} \|f_p - \mu_S\|^2 + \sum_{p \in \overline{S}} \|f_p - \mu_{\overline{S}}\|^2$$

$$= -\sum_{p \in \mathbf{S}} \ln \mathcal{N}(f_p | \mu_{\mathbf{S}}) - \sum_{p \in \bar{\mathbf{S}}} \ln \mathcal{N}(f_p | \mu_{\bar{\mathbf{S}}})$$

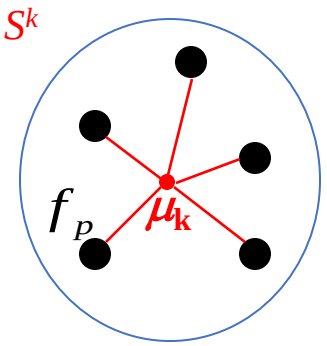
single Gaussian

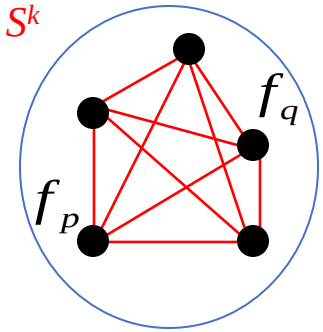
single Gaussian of ***fixed*** covariance



$$\theta_S = \{\mu_S\}$$

K-means as variance clustering criteria





both formulas can be written as

$$E(S) = \sum_{k=1}^{K} |S^k| \, \mathbf{var}(S^k)$$

sample variance: $\mathbf{var}(S^k) = \frac{1}{|S^k|} \sum_{p \in S^k} \| f_p - \mu_k \|^2 = \frac{1}{2|S^k|^2} \sum_{pq \in S^k} \| f_p - f_q \|^2$

[4] You want to cluster 7 points into 3 clusters using the ***k*-Means Clustering** algorithm. Suppose after the first iteration, clusters C_1, C_2 and C_3 contain the following two-dimensional points:

C_1 contains the 2 points: {(0,6), (6,0)}
C_2 contains the 3 points: {(2,2), (4,4), (6,6)}
C_3 contains the 2 points: {(5,5), (7,7)}

What are the **cluster centers** computed for these 3 clusters?

A. C_1: (3,3), C_2: (4,4), C_3: (6,6)
B. C_1: (3,3), C_2: (6,6), C_3: (12,12)
C. C_1: (6,6), C_2: (12,12), C_3: (12,12)
D. C_1: (0,0), C_2: (48,48), C_3: (35,35)
E. None of these

(generalization)

Distortion Clustering

can use different "distortion" measures

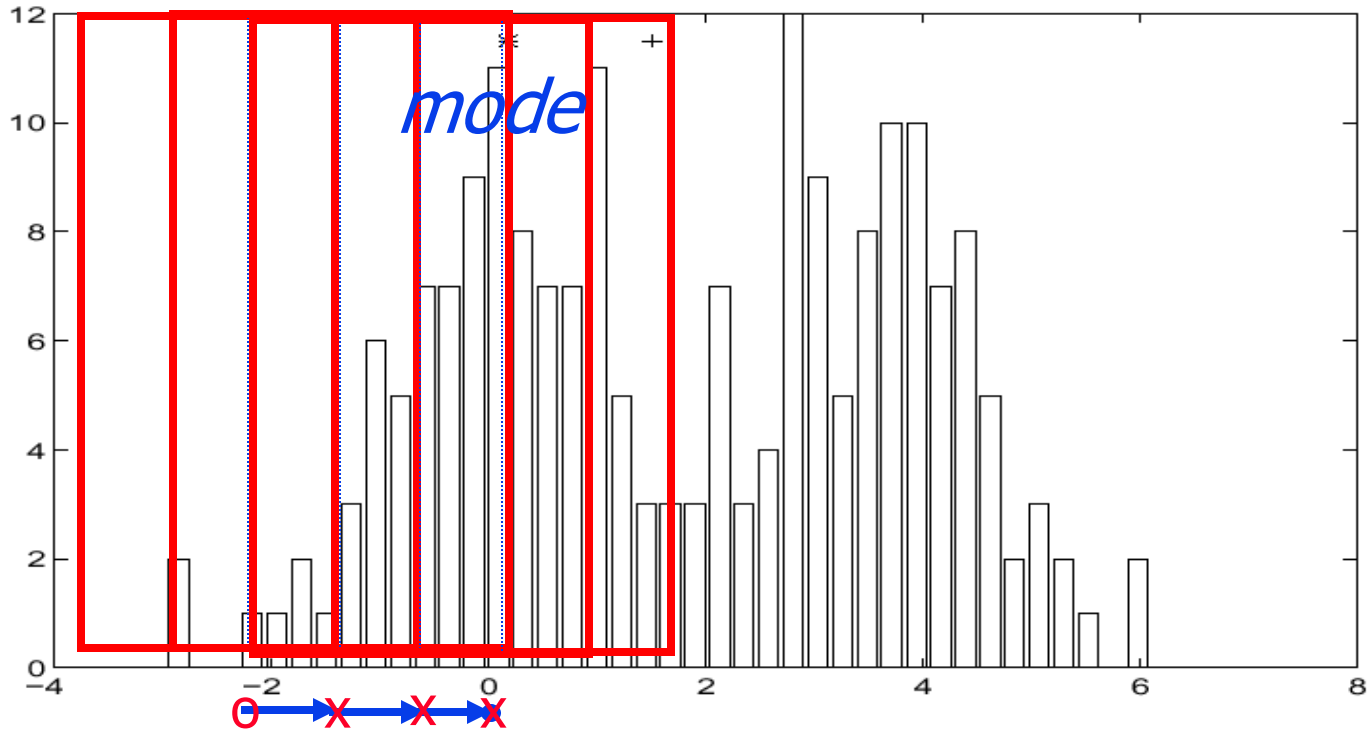
$$E(S, \mu) = \sum_{k=1}^{K} \sum_{p \in S_k} \|f_p - \mu_k\|_d$$

examples of distortion measure $\|\cdot\|_d$		interpretation of parameters μ_k
	$\|\cdot\|_d = \|\cdot\|^2$ squared L_2 norm	**K-means**
	$\|\cdot\|_d = \|\cdot\|$ absolute L_2 norm	**K-medians**
	$\|\cdot\|_d = 1 - \exp(-\|\cdot\|^2)$	**K-modes**

NOTE: besides changing the distortion measure, there are different generalizations of K-means requiring **other interpretations of SSE objective**

Mean Shift

[Fukunaga and Hostetler 1975, Cheng 1995, Comaniciu & Meer 2002]



Iterative Mode Search

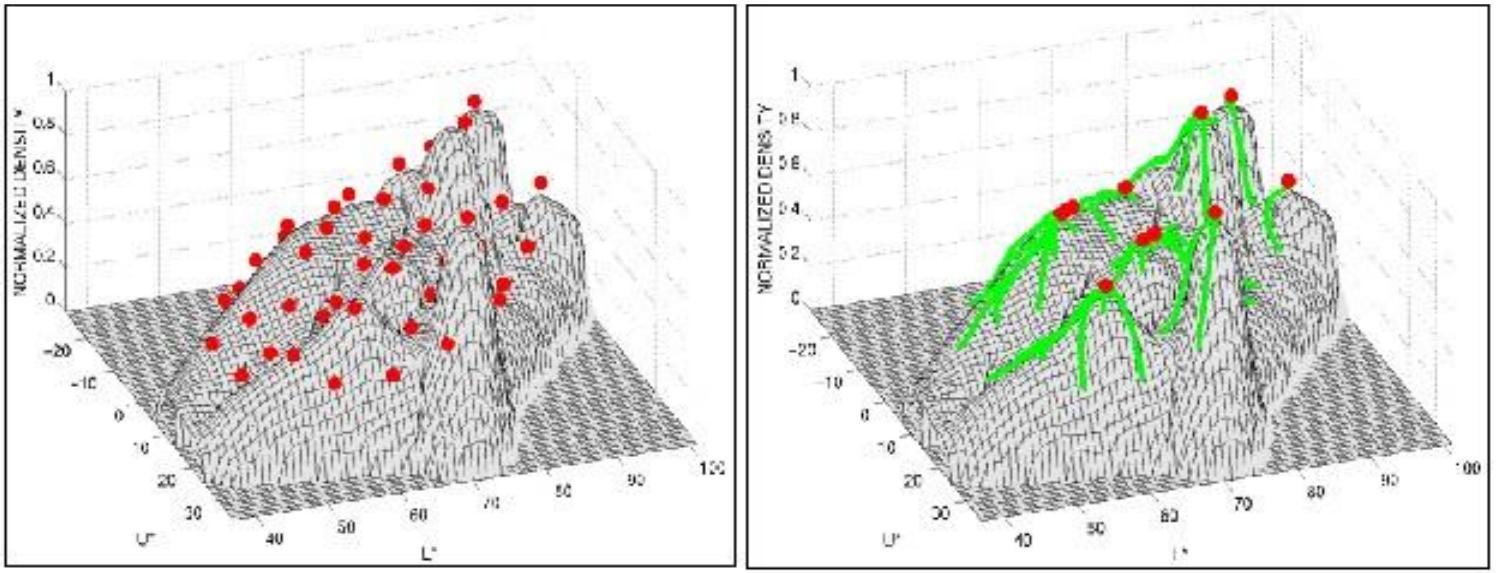
1. Initialize random seed, and fixed window
2. Calculate center of gravity ‘x’ of the window (the“mean”)
3. Translate the search window to the mean
4. Repeat Step 2 until convergence

Mean Shift as K-modes

[Salah, Mitche, Ben-Ayed 2010]

$$\sum_{k=1}^{K} \sum_{p \in S^k} \| f_p - \mu_k \|_d \qquad \| \cdot \|_d \quad : \quad \cup \; \textit{quadratic} \; (\textit{K-means}) \qquad \vee \; \textit{absolute} \; (\textit{K-medians}) \qquad \cap \; \textit{bounded} \; (\textit{K-modes})$$

Mean-shift segmentation relates to distortion clustering with a bounded loss (**K-modes**)



Mean shift trajectories

Kernel Density Estimation

Kernel density estimate with *bandwidth* σ: a mixture having one component for each data point:

$$p(\mathbf{x}) = \sum_{n=1}^{N} p(\mathbf{x}|n)p(n) = \frac{1}{N\sigma^D} \sum_{n=1}^{N} K\left(\frac{\mathbf{x} - \mathbf{x}_n}{\sigma}\right) \qquad \mathbf{x} \in \mathbb{R}^D.$$

Usually the kernel K is Gaussian: $K\left(\frac{\mathbf{x}-\mathbf{x}_n}{\sigma}\right) = (2\pi)^{-D/2} \exp\left(-\frac{1}{2}\|(\mathbf{x} - \mathbf{x}_n)/\sigma\|^2\right)$.

Mean-shift

Mean-shift algorithm: starting from an initial value of $\mathbf{x}$, it iterates the following expression:

$$\mathbf{x} \leftarrow \sum_{n=1}^{N} p(n|\mathbf{x})\mathbf{x}_n \qquad \text{where} \qquad p(n|\mathbf{x}) = \frac{p(\mathbf{x}|n)p(n)}{p(\mathbf{x})} = \frac{\exp\left(-\frac{1}{2}\|(\mathbf{x}-\mathbf{x}_n)/\sigma\|^2\right)}{\sum_{n'=1}^{N}\exp\left(-\frac{1}{2}\|(\mathbf{x}-\mathbf{x}_{n'})/\sigma\|^2\right)}.$$

$\sum_{n=1}^{N} p(n|\mathbf{x})\mathbf{x}_n$ can be understood as the weighted average of the N data points using as weights the posterior probabilities $p(n|\mathbf{x})$. The mean-shift algorithm converges to a mode of $p(\mathbf{x})$. Which one it converges to depends on the initialization. By running mean-shift starting at a data point $\mathbf{x}_n$, we effectively assign $\mathbf{x}_n$ to a mode. We repeat for all points $\mathbf{x}_1, \ldots, \mathbf{x}_N$.

K-means and MLE (maximum likelihood estimation)

"**hard**" K-means

$$E(S,\mu) = -\sum_{k=1}^{K}\sum_{p\in S^k} \log P(f_p \,|\, \mu_k)$$

multi-variate (i.e. $x, \mu \in R^N$)
Gaussian distribution
(simple special case $\Sigma = \sigma^2 \mathbf{I}$)

$$P(x|\mu) = \frac{1}{\sqrt{(2\pi\sigma^2)^N}} \exp -\frac{\|x-\mu\|^2}{2\,\sigma^2}$$

Towards soft clustering…

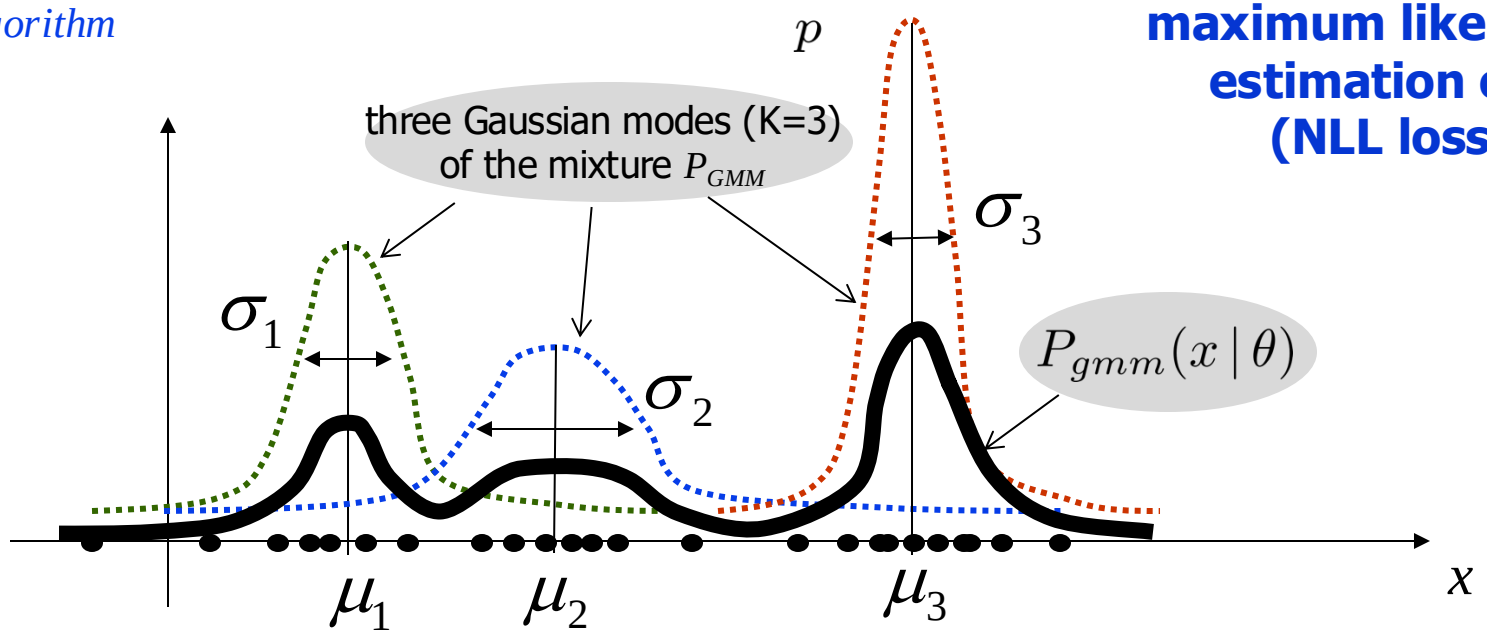
Gaussian Mixture Models (GMM)

- Soft clustering using Gaussian Mixture Model (GMM)
  - no "hard" assignments of points to K distinct (Gaussian) clusters S^k
  - all points are used to estimate parameters of one complex K-mode distribution (GMM)

approximate optimization via *EM algorithm*

$$E_{gmm}(\theta) := -\sum_p \log P_{gmm}(x_p \mid \theta)$$

maximum likelihood estimation of θ (NLL loss)

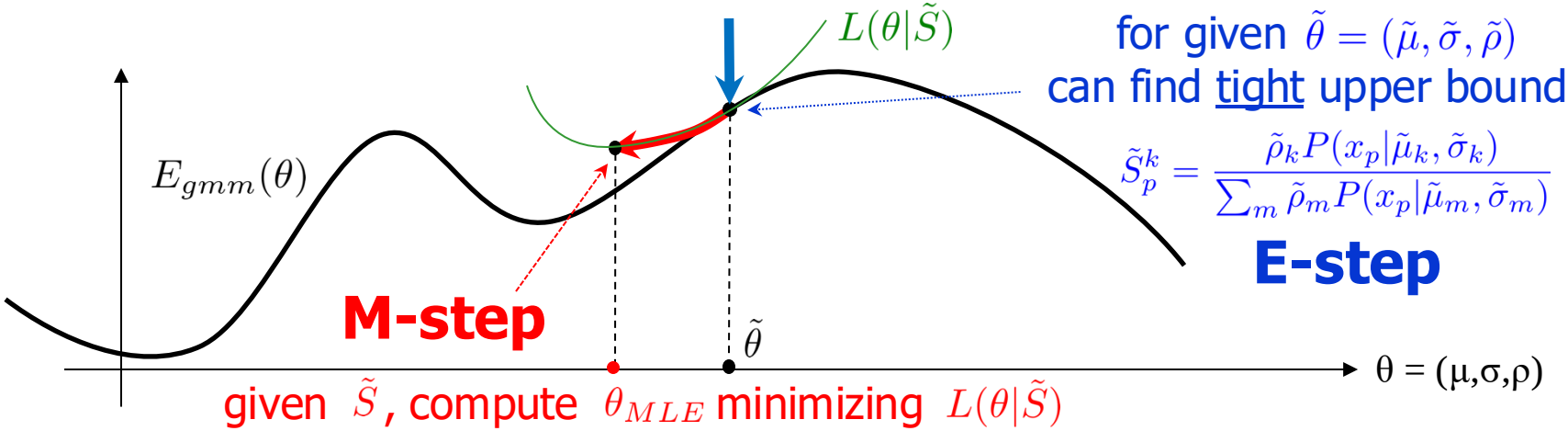


GMM distribution: $$P_{gmm}(x \mid \theta) := \sum_k \rho_k P(x \mid \mu_k, \sigma_k)$$

Expectation-Maximization (EM)

GMM estimation - optimization of ML objective (sum of Negative Log Likelihoods, a.k.a. **NLL** loss)

$$E_{gmm}(\theta) \;:=\; -\sum_p \log P_{gmm}(x_p \,|\, \theta) \;\equiv\; -\sum_p \log \left(\sum_k \rho_k \, P(x_p \,|\, \mu_k, \sigma_k) \right)$$

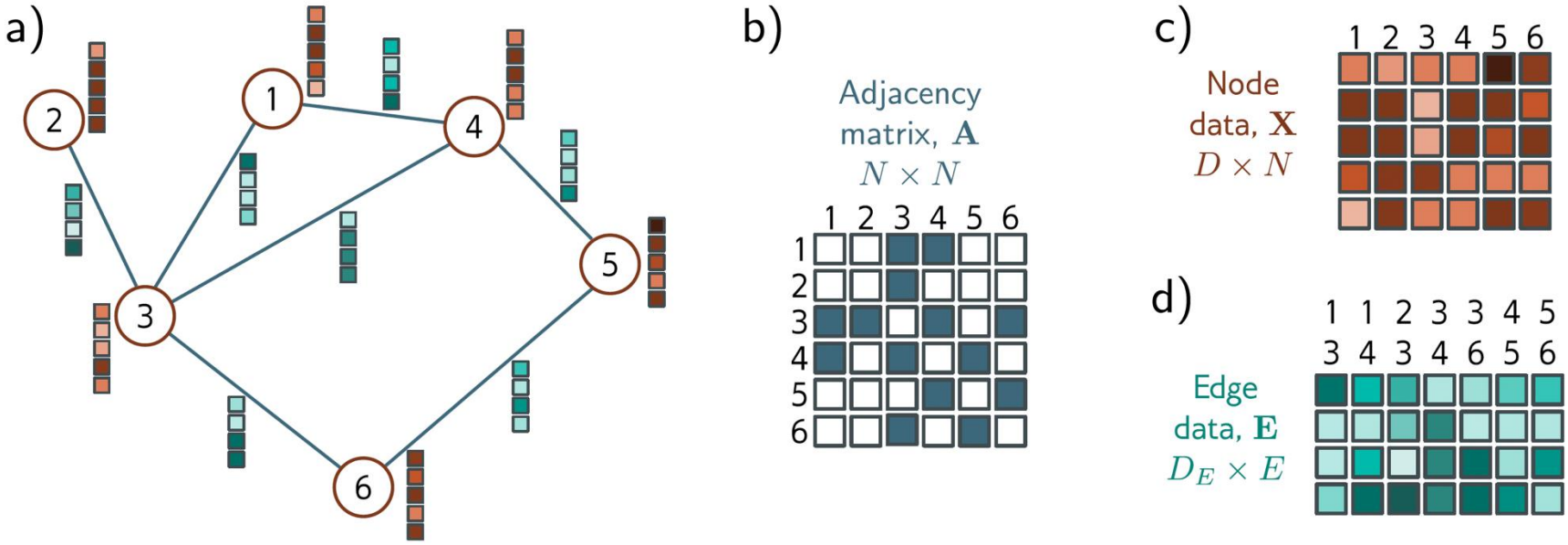


$L(\theta|S)$ - for any S defines an upper bounds for $E_{gmm}(\theta)$

$$\leq \; -\sum_k \left(\overbrace{\sum_p S_p^k}^{} \right) \log \rho_k \;-\; \sum_k \sum_p S_p^k \, \log P(x_p \,|\, \mu_k, \sigma_k) \;-\; \sum_p \mathbf{H}(S_p)$$

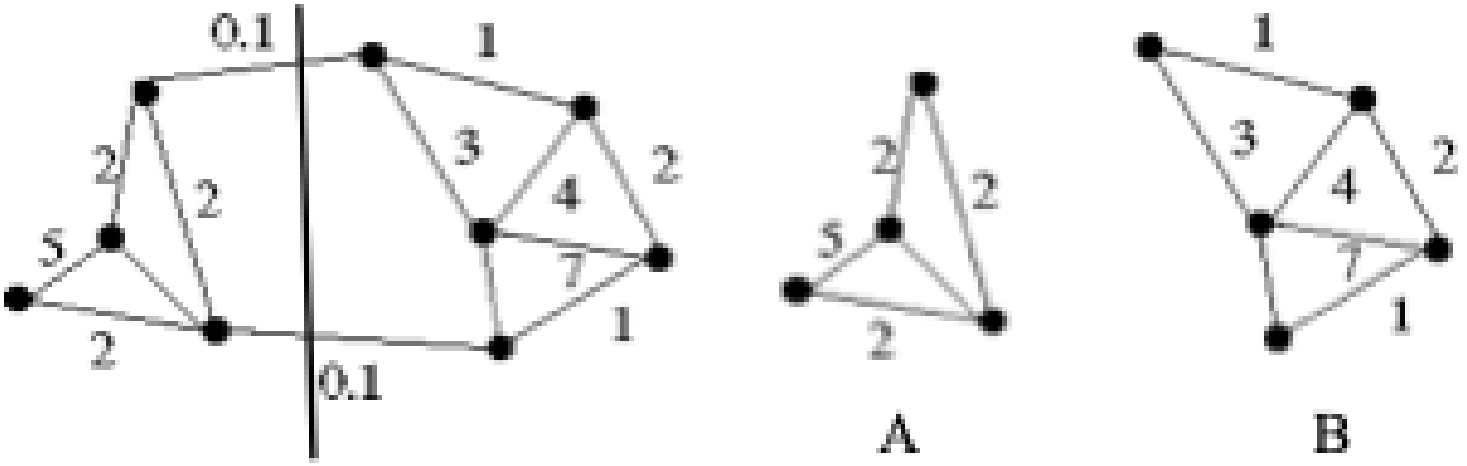
Graph representation

- A graph is defined as a tuple G = (V, E)
  - where V is a set of nodes
  - and E is a set of edges
- An example graph with 6 nodes and 7 edges



Minimum Cut

- In many applications it is desired to find the cut with minimum cost: *minimum cut*
- Well studied problem in graph theory, with many applications
- There exists efficient algorithms for finding minimum cuts

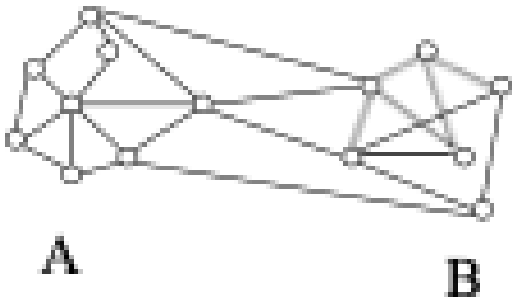


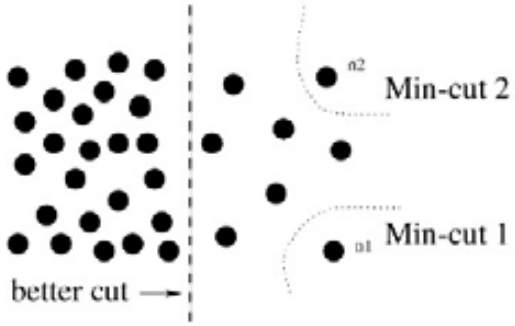
$$cut(A,B) = \sum_{u \in A, v \in B} w(u,v)$$

Normalized Cut

- Normalize *cut cost* by volume of clusters
- Compute the cut cost as a fraction of the total edge connections to all nodes in the graph

$$Ncut(A,B) = \frac{cut(A,B)}{assoc(A,V)} + \frac{cut(A,B)}{assoc(B,V)}$$

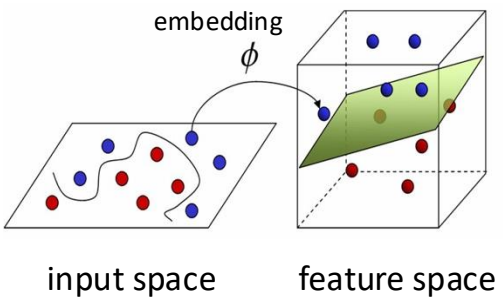




Summary of Normalized cut algorithm

- Given a set of features, construct a weighted graph by computing weight on each edge and then placing the data into W and D.
- Solve $(D-W)x=\lambda Dx$ for eigen vectors with the smallest eigenvalues.
- Use the eigen vector corresponding to the second smallest eigenvalue to bipartition the graph into two groups.
- Recursively repartition the segmented parts if necessary.

Toward Kernel K-means



$$E(S,\mu) = \sum_{k=1}^{K} \sum_{p \in S^k} \|f_p - \mu_k\|^2$$

(Basic K-means)

$$E_k(S,\hat{\mu}) = \sum_{k=1}^{K} \sum_{p \in S^k} \|\phi(f_p) - \hat{\mu}_k\|^2$$

(Kernel K-means)

Explicit Kernel ⇔ *Implicit Embedding*

$$E_k(S, \hat{\mu}) = \sum_{k=1}^{K} \sum_{p \in S^k} \|\phi(f_p) - \hat{\mu}_k\|^2$$

just plug-in

$$\hat{\mu}_k = \frac{1}{|S^k|} \sum_{q \in S^k} \phi(f_q)$$

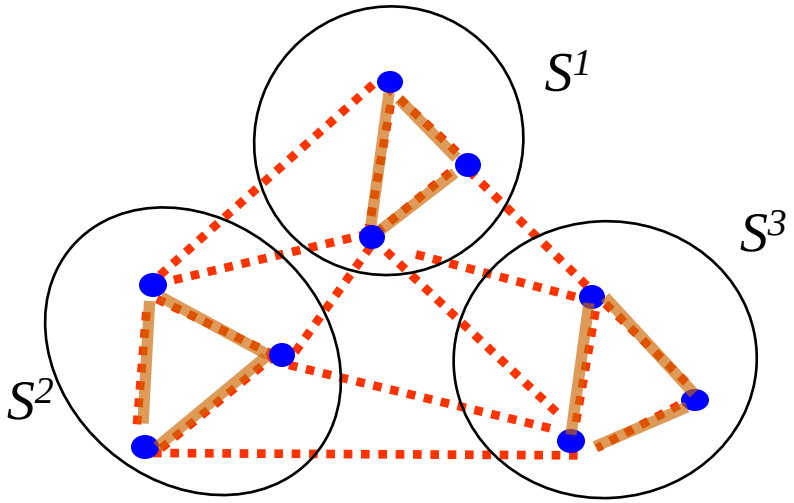
equivalent

$$E_k(S) = -\sum_{k=1}^{K} \frac{\sum_{pq \in S_k} k(f_p, f_q)}{|S^k|}$$

kernel K-means or ***average association***

$$E(S) = -\sum_{k=1}^{K} \frac{\sum_{pq \in S_k} A_{pq}}{|S^k|}$$

"self-association" of cluster S^k



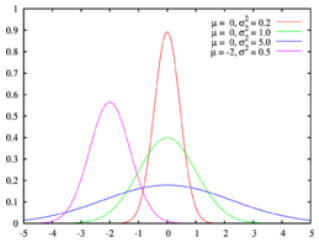
K-means

$$\sum_{p\in\mathbf{S}} \|f_p - \mu_{\mathbf{S}}\|^2 + \sum_{p\in\overline{\mathbf{S}}} \|f_p - \mu_{\overline{\mathbf{S}}}\|^2$$

probabilistic K-means

make models more complex

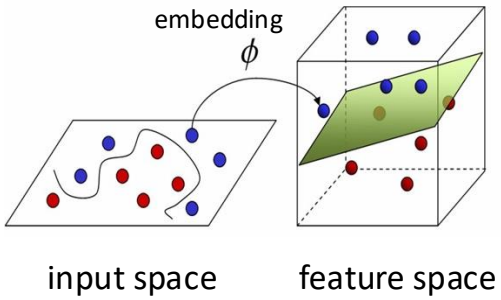
$$-\sum_{p\in\mathbf{S}} \ln \Pr(f_p|\theta_{\mathbf{S}}) - \sum_{p\in\bar{\mathbf{S}}} \ln \Pr(f_p|\theta_{\bar{\mathbf{S}}})$$



kernel K-means

make data more complex

$$\sum_{p\in\mathbf{S}} \|\phi(f_p) - \hat{\mu}_{\mathbf{S}}\|^2 + \sum_{p\in\overline{\mathbf{S}}} \|\phi(f_p) - \hat{\mu}_{\overline{\mathbf{S}}}\|^2$$

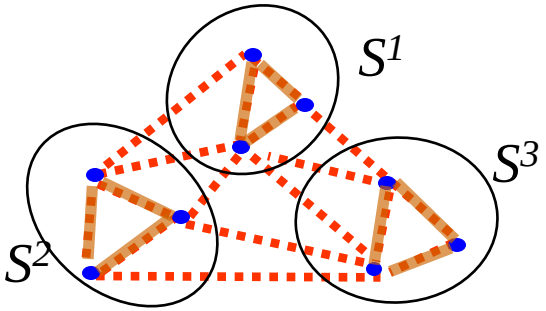


Other kernel (graph) clustering objectives

Average Association

"self-association" for S^k

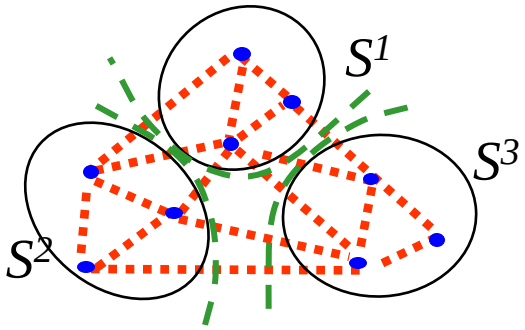
$$-\sum_{k=1}^{K} \frac{S^{k'} A S^k}{|S^k|}$$



Average Cut

"cut" for S^k

$$\sum_{k=1}^{K} \frac{S^{k'} A (1 - S^k)}{|S^k|}$$



kernel (graph) clustering objectives

- **Average Cut**

$$\sum_{k=1}^{K} \frac{S^{k'} A \left(\mathbf{1} - S^{k}\right)}{\mathbf{1}' S^{k}}$$

- **Average Association**

$$\not\Leftrightarrow \quad -\sum_{k=1}^{K} \frac{S^{k'} A\, S^{k}}{\mathbf{1}' S^{k}}$$

- **Normalized Cut**

$$\sum_{k=1}^{K} \frac{S^{k'} A \left(\mathbf{1} - S^{k}\right)}{d' S^{k}}$$

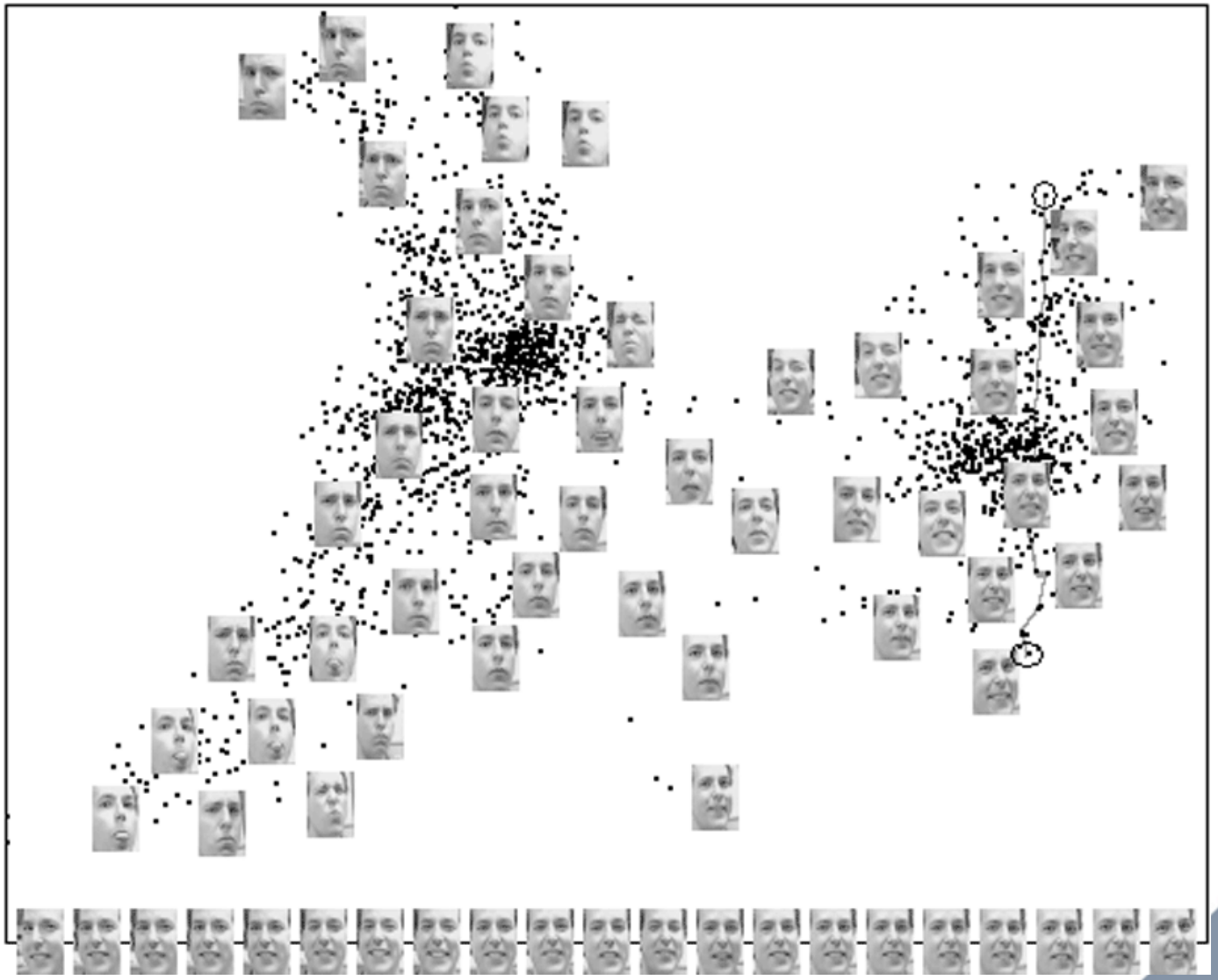
- **Normalized Association**

$$\equiv \quad K \quad -\sum_{k=1}^{K} \frac{S^{k'} A S^{k}}{d' S^{k}}$$

normalization $d := A\mathbf{1}$

Dimensionality Reduction

Motivation for Dimensionality Reduction



Principal Component Analysis

Goal: Find r-dim projection that best preserves variance

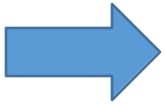
1. Compute mean vector μ and covariance matrix Σ of original points
2. Compute eigenvectors and eigenvalues of Σ
3. Select top r eigenvectors
4. Project points onto subspace spanned by them:

$$y = A(x - \mu)$$

where y is the new point, x is the old one,
and the rows of A are the eigenvectors

Covariance

- Variance and Covariance:
  - Measure of the "spread" of a set of points around their center of mass(mean)
- Variance:
  - Measure of the deviation from the mean for points in one dimension
- Covariance:
  - Measure of how much each of the dimensions vary from the mean with **respect to each other**



- **Covariance is measured between two dimensions**
- **Covariance sees if there is a relation between two dimensions**
- **Covariance between one dimension is the variance**

Principal Component Analysis

Input: $\mathbf{x} \in \mathbb{R}^D$: $\mathcal{D} = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$

Set of basis vectors: $\mathbf{u}_1, \ldots, \mathbf{u}_K$

Summarize a *D* dimensional vector *X* with *K* dimensional feature vector *h(x)*

$$h(\mathbf{x}) = \begin{bmatrix} \mathbf{u}_1 \cdot \mathbf{x} \\ \mathbf{u}_2 \cdot \mathbf{x} \\ \cdots \\ \mathbf{u}_K \cdot \mathbf{x} \end{bmatrix}$$

You are given a design matrix $X = \begin{bmatrix} 6 & -4 \\ -3 & 5 \\ -2 & 6 \\ 7 & -3 \end{bmatrix}$. Let's use PCA to reduce the dimension from 2 to 1.

(1) [6 pts] Compute the covariance matrix for the sample points. (Warning: Observe that X is not centered.) Then compute the **unit** eigenvectors, and the corresponding eigenvalues, of the covariance matrix. *Hint:* If you graph the points, you can probably guess the eigenvectors (then verify that they really are eigenvectors).

Neural Network

Neural Unit

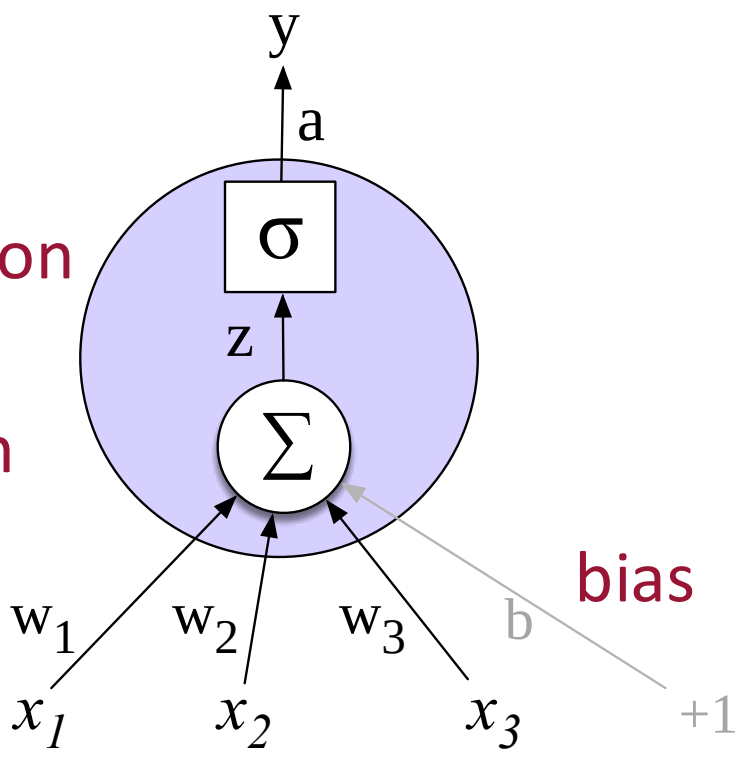
Output value

Non-linear activation function

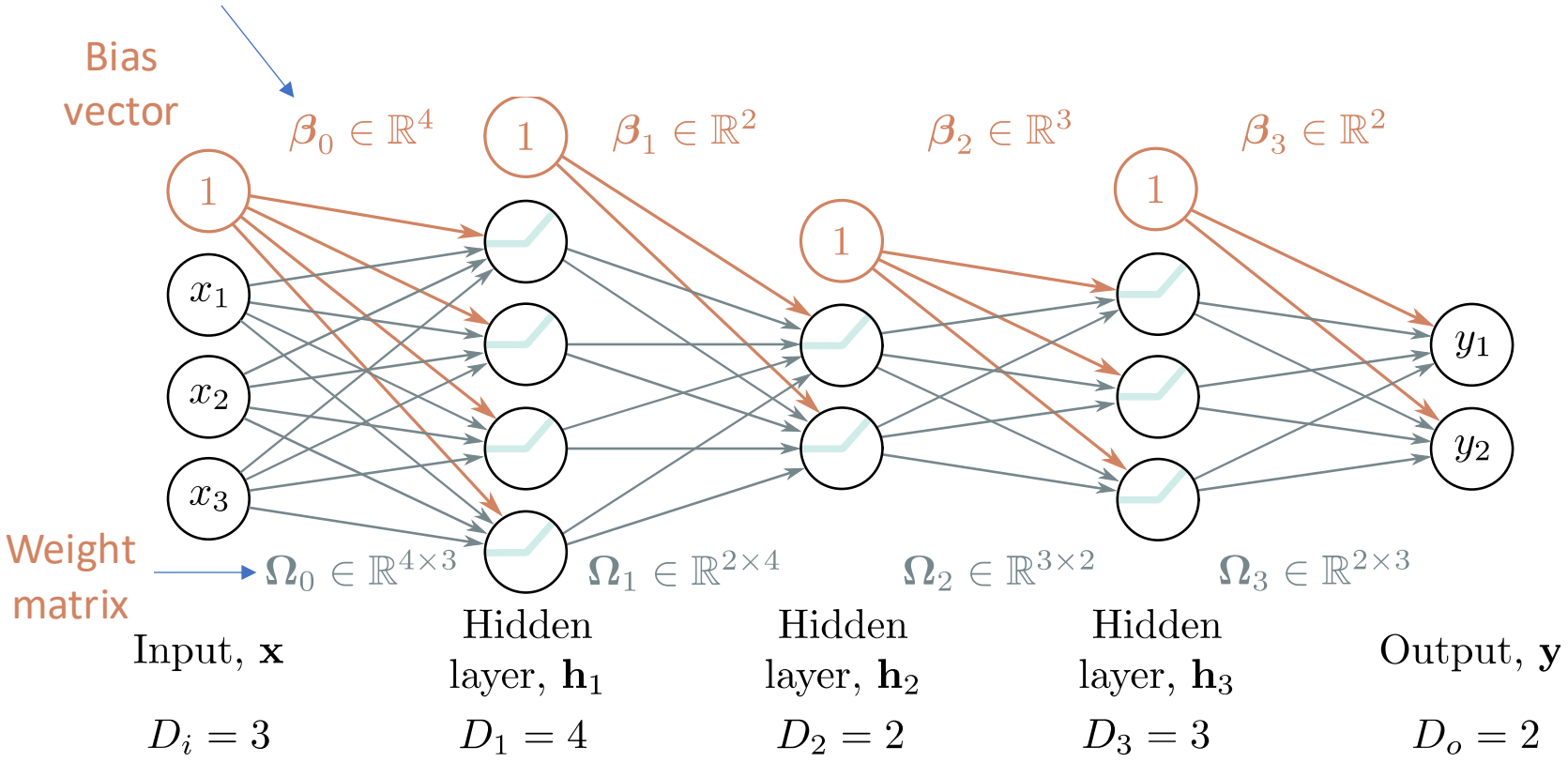
Weighted sum

Weights

Input layer

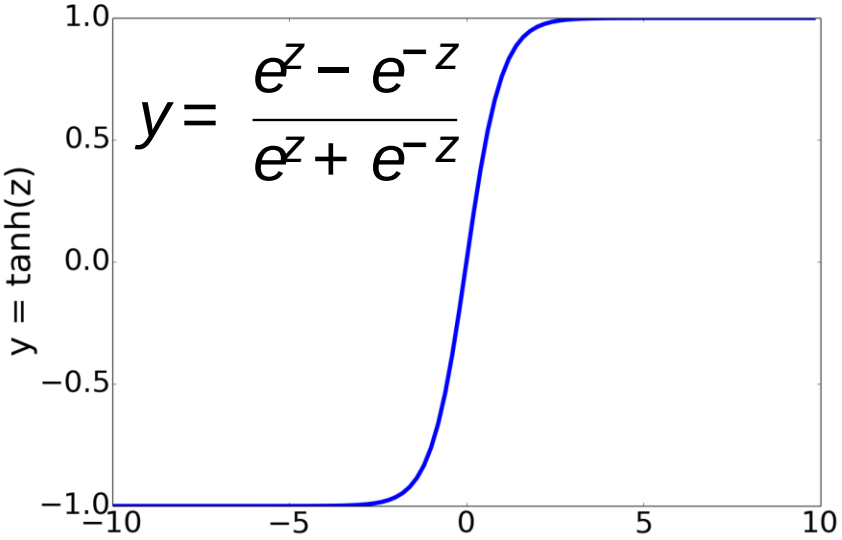


Multi-layer perceptron



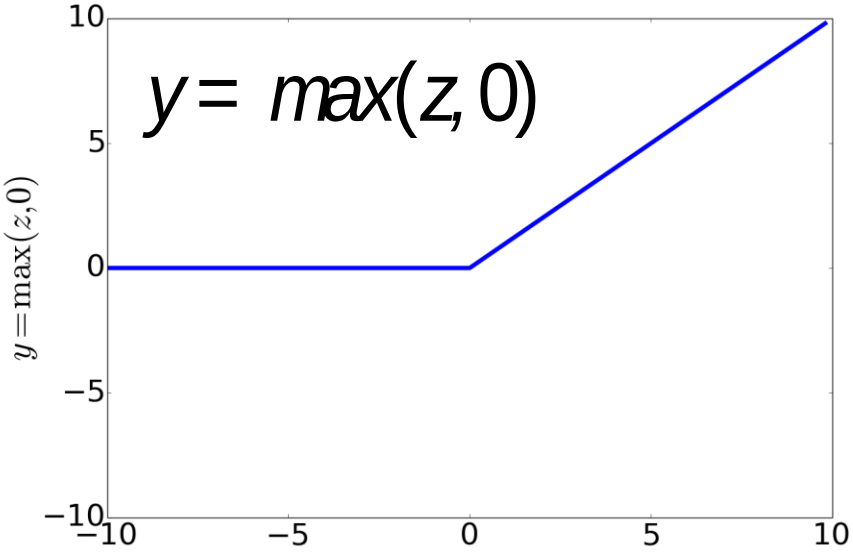
Example of Multi Layer Perceptron (MLP)

Activation function



tanh

Most Common:



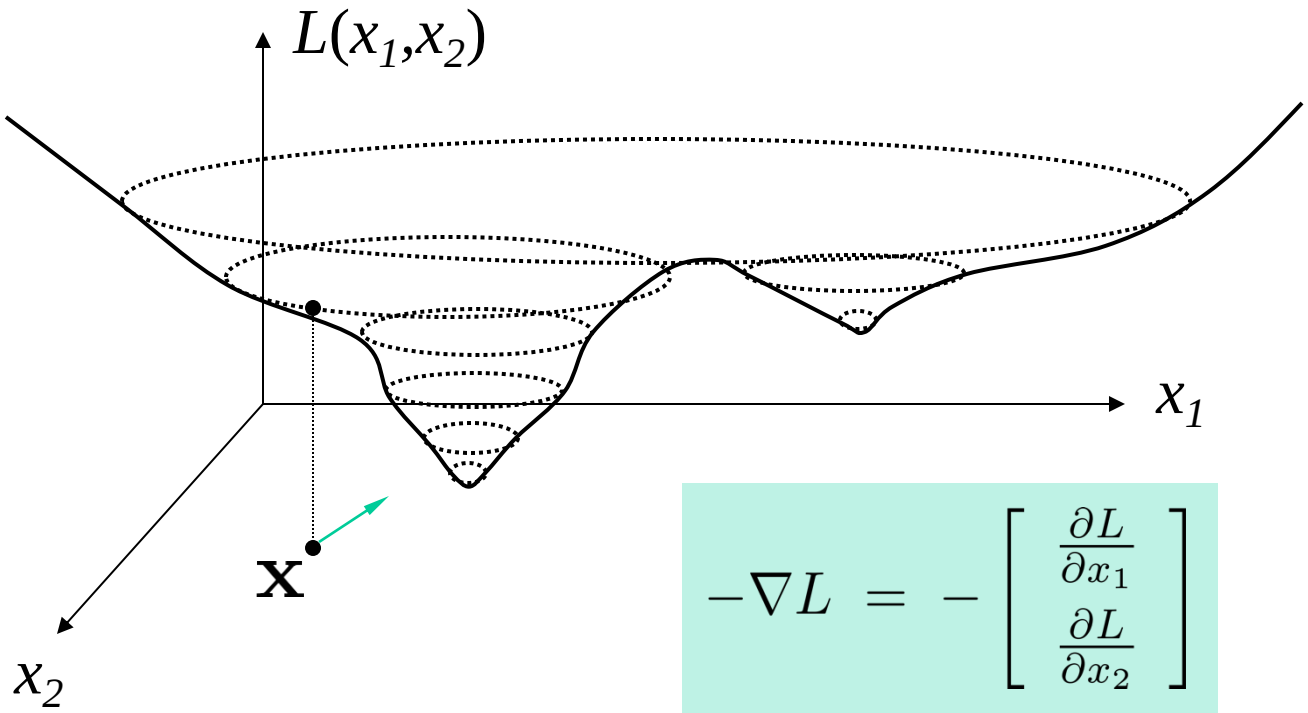
ReLU

Rectified Linear Unit

UCMERCED

Gradient Descent

Example: for a function of two variables

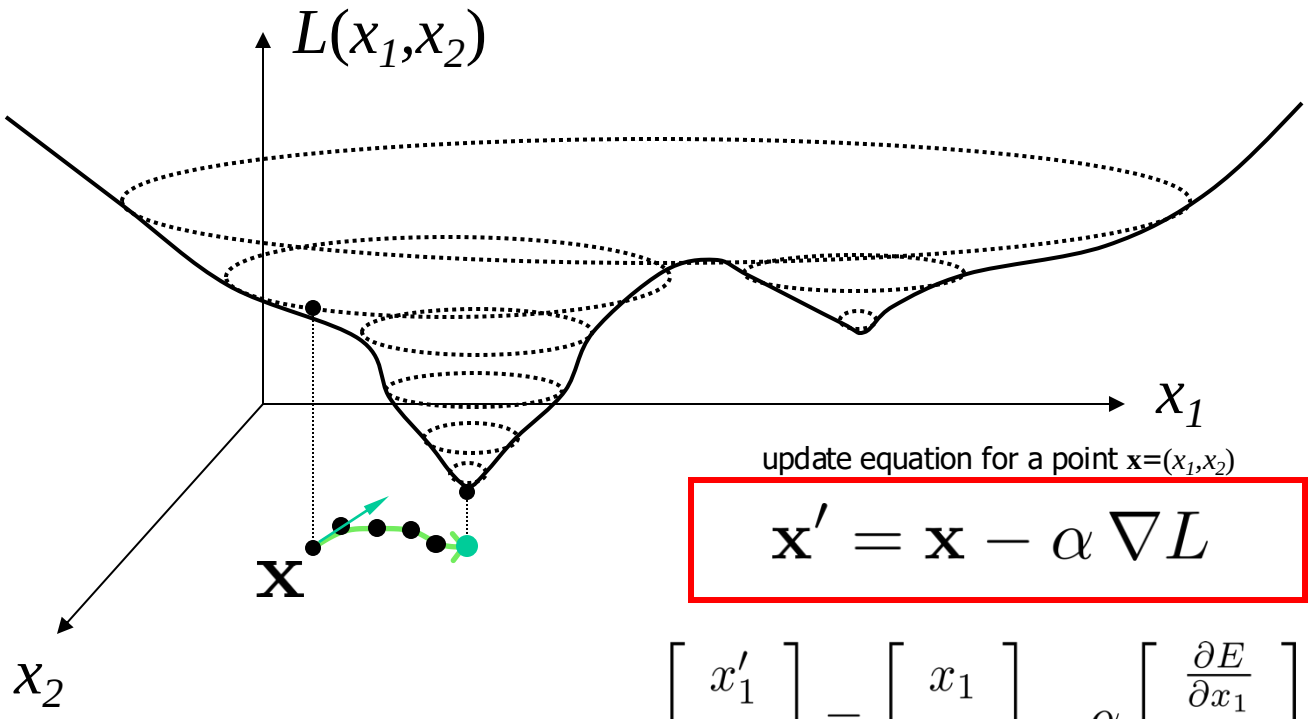


$$-\nabla L = -\begin{bmatrix} \frac{\partial L}{\partial x_1} \\ \frac{\partial L}{\partial x_2} \end{bmatrix}$$

- direction of (negative) gradient at point $\mathbf{x}=(x_1,x_2)$ is direction of the steepest descent towards lower values of function L
- magnitude of gradient at $\mathbf{x}=(x_1,x_2)$ gives the value of the slope

Gradient Descent

Example: for a function of two variables



update equation for a point $\mathbf{x}=(x_1,x_2)$

$$\mathbf{x}' = \mathbf{x} - \alpha \nabla L$$

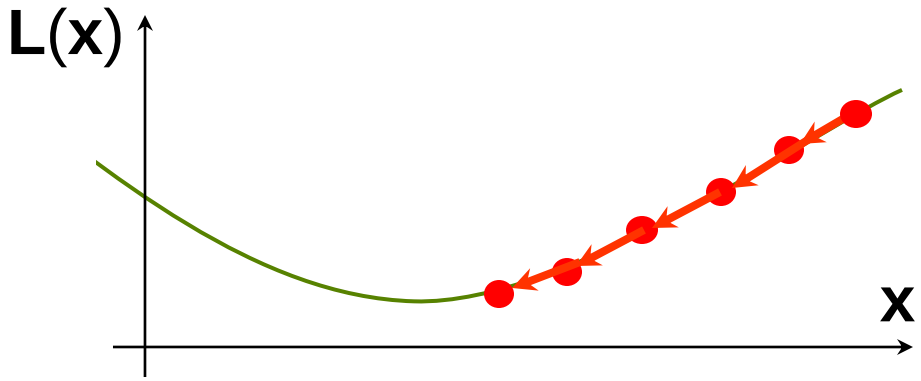
$$\begin{bmatrix} x_1' \\ x_2' \end{bmatrix} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} - \alpha \begin{bmatrix} \frac{\partial E}{\partial x_1} \\ \frac{\partial E}{\partial x_2} \end{bmatrix}$$

Stop at a local minima where $\nabla L = \vec{0}$

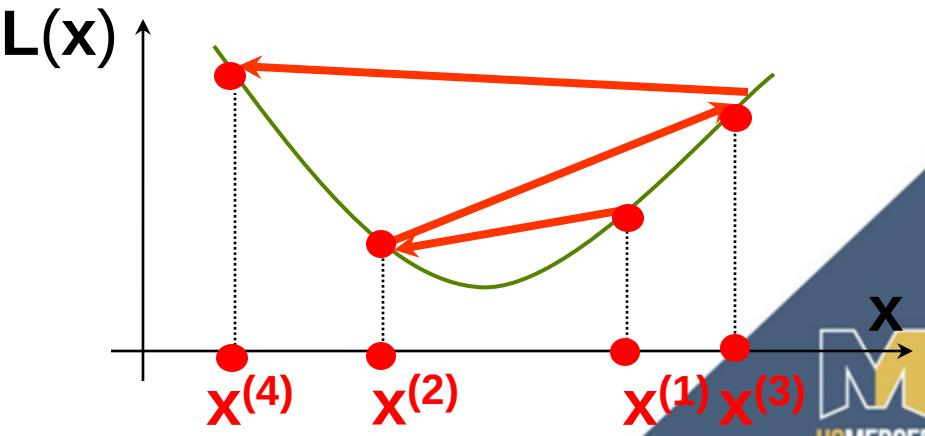
How to Set Learning Rate α?

$$\mathbf{x}' = \mathbf{x} - \alpha \nabla L$$

- If α too small, too many iterations to converge



- If α too large, may overshoot the local minimum and possibly never even converge



Variable Learning Rate

If desired, can change learning rate α at each iteration

$\mathbf{k} = 1$

$\mathbf{x}^{(1)}$ = any initial guess

choose α, ε

while $\alpha \|\nabla L(\mathbf{x}^{(k)})\| > \varepsilon$

$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - \alpha \nabla L(\mathbf{x}^{(k)})$

$\mathbf{k} = \mathbf{k} + 1$

fixed α
gradient descent

$\mathbf{k} = 1$

$\mathbf{x}^{(1)}$ = any initial guess

choose ε

while $\alpha \|\nabla L(\mathbf{x}^{(k)})\| > \varepsilon$

choose $\alpha^{(k)}$

$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - \alpha^{(k)} \nabla L(\mathbf{x}^{(k)})$

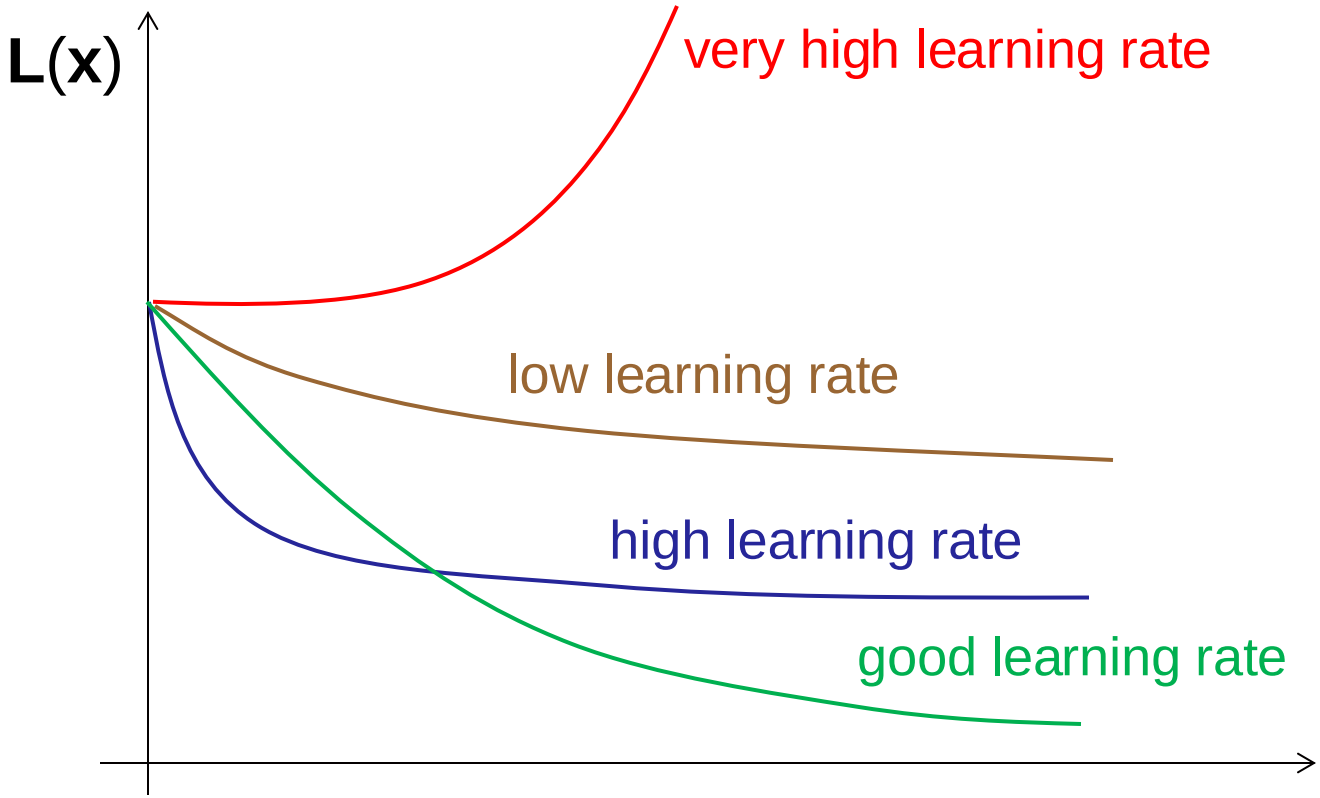
$\mathbf{k} = \mathbf{k} + 1$

variable α
gradient descent

UCMERCED

Learning Rate

- Monitor learning rate by looking at how fast the objective function decreases

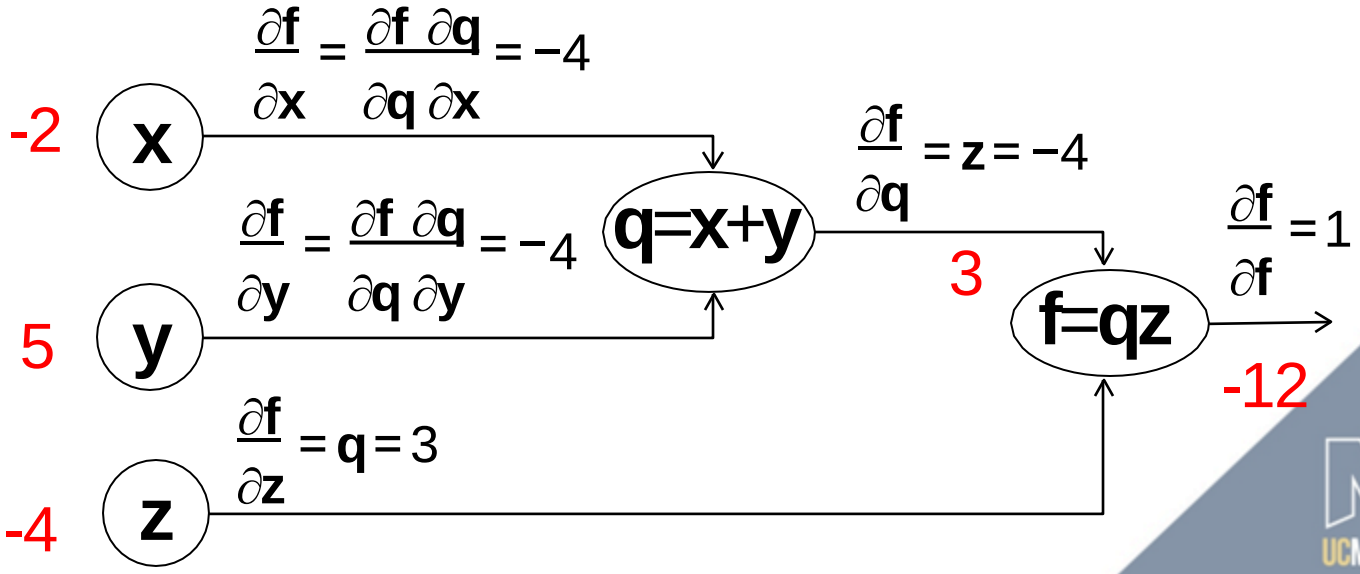


Computing Derivatives: Small Example

- Small network $\mathbf{f(x,y,z) = (x+y)z}$
- Rewrite using $\mathbf{q = x + y} \Rightarrow \mathbf{f(x,y,z) = qz}$
- Want $\frac{\partial \mathbf{f}}{\partial \mathbf{x}}, \frac{\partial \mathbf{f}}{\partial \mathbf{y}}, \frac{\partial \mathbf{f}}{\partial \mathbf{z}}$
- Compute $\frac{\partial \mathbf{f}}{\partial}$ from the end backwards
  - for each edge, with respect to the main variable at edge origin
  - using chain rule with respect to the variable at edge end, if needed

chain rule for $\mathbf{f(y(x))}$

$$\frac{\partial \mathbf{f}}{\partial \mathbf{x}} = \frac{\partial \mathbf{f}}{\partial \mathbf{y}} \frac{\partial \mathbf{y}}{\partial \mathbf{x}}$$



Computing Derivatives: Vector Notation

- Let $\mathbf{f}(\mathbf{x})$: $\mathbf{R}^n \rightarrow \mathbf{R}^m$,
  - $\mathbf{x}$ is $\mathbf{n}$-dimensional vector and output $\mathbf{f}(\mathbf{x})$ is $\mathbf{m}$-dimensional vector
- Jacobian matrix
  - has $\mathbf{m}$ rows and $\mathbf{n}$ columns
  - has $\frac{\partial \mathbf{f}_i}{\partial \mathbf{x}_j}$ in row $\mathbf{i}$, column $\mathbf{j}$

Computing Derivatives: Vector Notation

- $\mathbf{f}(\mathbf{x})$: $\mathbf{R}^n \rightarrow \mathbf{R}^m$ and $\mathbf{g}(\mathbf{x})$: $\mathbf{R}^k \rightarrow \mathbf{R}^n$
- $\mathbf{f}(\mathbf{g}(\mathbf{x}))$: $\mathbf{R}^k \rightarrow \mathbf{R}^m$
- Chain rule for vector functions

$$\frac{\partial \mathbf{f}}{\partial \mathbf{x}} = \frac{\partial \mathbf{f}}{\partial \mathbf{g}} \frac{\partial \mathbf{g}}{\partial \mathbf{x}}$$

Jacobian matrices

(10 points) Consider boolean logical operators such as AND, OR, and XOR,

AND			OR			XOR		
x1	x2	y	x1	x2	y	x1	x2	y
0	0	0	0	0	0	0	0	0
0	1	0	0	1	1	0	1	1
1	0	0	1	0	1	1	0	1
1	1	1	1	1	1	1	1	0

(a) (5 points) Can we implement OR using perception? If yes, show the corresponding perceptron network. If no, explain why.

Solution:

(b) (5 points) Can we implement XOR using perception? If yes, show the corresponding perceptron network. If no, explain why.

Solution:

(10 points) This problem explores computing derivatives on composite functions. Consider the function:

$$y = \exp[\exp[x] + \exp[x]^2] + \sin[\exp[x] + \exp[x]^2].$$

We can break this down into a series of intermediate computations so that:

$$\begin{aligned} f_1 &= \exp[x] \\ f_2 &= f_1^2 \\ f_3 &= f_1 + f_2 \\ f_4 &= \exp[f_3] \\ f_5 &= \sin[f_3] \\ y &= f_4 + f_5 \end{aligned}$$

Compute the derivatives in order using the chain rule of gradients in each case to make use of the derivatives already computed.

$$\frac{\partial y}{\partial f_5}, \frac{\partial y}{\partial f_4}, \frac{\partial y}{\partial f_3}, \frac{\partial y}{\partial f_2}, \frac{\partial y}{\partial f_1}, \text{ and } \frac{\partial y}{\partial x}.$$