Advanced Introduction to Machine Learning

— Spring Quarter, Week 10 —

https://canvas.uw.edu/courses/1372141

Prof. Jeff Bilmes

University of Washington, Seattle
Departments of: Electrical & Computer Engineering, Computer Science & Engineering
http://melodi.ee.washington.edu/~bilmes

June 3rd/June 8th, 2020

Announcements

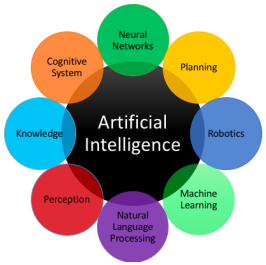
- HW5 is due Friday, June 12th, 11:59pm, see `https://canvas.uw.edu/courses/1372141/assignments/5510802`
- HW5 read through, due Friday, June 5th, 11:59pm. see `https://canvas.uw.edu/courses/1372141/assignments/5510803`
- Don't forget about the read through deadline!!
- NO LATES ACCEPTED FOR HW5!!! It must be turned in no later than Friday, June 12th, 11:59pm.
- Virtual office hours this week, Thursday night at 10:00pm via zoom (same link as class).
- Monday, June 8th, 10:30am, final lecture (lecture 20).

Class Road Map

- W1(3/30,4/1): What is ML, Probability, Coins, Gaussians and linear regression, Associative Memories, Supervised Learning
- W2(4/6,4/8): More supervised, logistic regression, complexity and bias/variance tradeoff
- W3(4/13,4/15): Bias/Variance, Regularization, Ridge, CrossVal, Multiclass
- W4(4/20,4/22): Multiclass classification, ERM, Gen/Disc, Naïve Bayes
- W5(4/27,4/29): Lasso, Regularizers, Curse of Dimensionality
- W6(5/4,5/6): Curse of Dimensionality, Dimensionality Reduction, k-NN
- W7(5/11,5/13): k-NN, LSH, DTs, Bootstrap/Bagging, Boosting & Random Forests, GBDTs
- W8(5/18,5/20): Graphs; Graphical Models (Factorization, Inference, MRFs, BNs);
- W9(5/27,6/1): Learning Paradigms; Clustering; EM Algorithm;
- W10(6/3,6/8): Spectral Clustering, Graph SSL, Deep models, (SVMs, RL); The Future.

Last lecture is 6/8 since 5/25 is holiday (or we could just have lecture on 5/25).

Class (and Machine Learning) overview

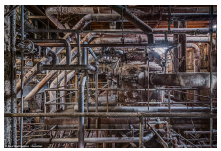


1. Introduction
- What is ML
- What is AI
- Why are we so interested in these topics right now?

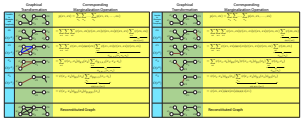
2. ML Paradigms/Concepts
- Overfitting/Underfitting, model complexity, bias/variance
- size of data, big data, sample complexity
- ERM, loss + regularization, loss functions, regularizers
- Supervised, unsupervised, and semi-supervised learning;
- reinforcement learning, RL, multi-agent, planning/control
- transfer and multi-task learning
- federated and distributed learning
- active learning, machine teaching
- self-supervised, zero/one-shot, open-set learning

3. Dealing with Features
- dimensionality reduction, PCA, LDA, MDS, T-SNE, UMAP
- Locality sensitive hashing (LSH)
- feature selection
- feature engineering
- matrix factorization & feature engineering
- representation learning

4. Evaluation
- accuracy/error, precision/recall, ROC, likelihood/posterior, cost/utility, margin
- train/eval/test data splits
- n-fold cross validation
- method of the bootstrap



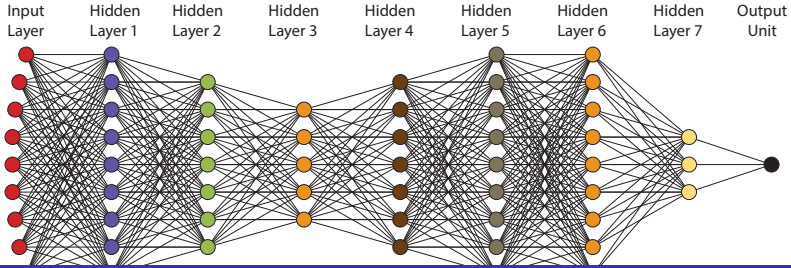
5. Optimization Methods
- Unconstrained Continuous Optimization: (stochastic) gradient descent (SGD), adaptive learning rates, conjugate gradient, 2nd order Newton
- Constrained Continuous Optimization : Frank-Wolf (conditional gradient descent), projected gradient, linear, quadratic, and convex programming
- Discrete optimization - greedy, beam search, branch-and-bound, submodular optimization.



6. Inference Methods
- probabilistic inference
- MLE, MAP
- belief propagation
- forward/backpropagation
- Monte Carlo methods

7. Models & Representation
- linear least squares, linear regression, logistic regression, sparsity, ridge, lasso
- generative vs. discriminative models
- Naive Bayes
- k-nearest neighbors
- clustering, k-means, k-mediods, EM & GMMs, single linkage
- decision trees and random forests
- support vector machines, kernel methods, max margin
- perceptron, neural networks, DNNs
- Gaussian processes
- Bayesian nonparametric methods
- ensemble methods
- the bootstrap, bagging, and boosting
- graphical models
- time-series, HMMs, DBNs, RNNs, LSTMs, Attention, Transformers
- structured prediction
- grammars (as in NLP)

12. Other Techniques
- compressed sensing
- submodularity, diversity/homogeneity modeling

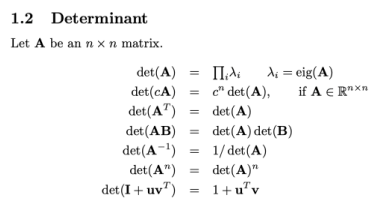


8. Philosophy, Humanity, Spirituality
- artificial intelligence (AI)
- artifical general intelligence (AGI)
- artificial intelligence vs. science fiction

9. Applications
- computational biology
- social networks
- computer vision
- speech recognition
- natural language processing
- information retrieval
- collaborative filtering/matrix factorization

10. Programming
- python
- libraries (e.g., NumPy, SciPy, matplotlib, scikit-learn (sklearn), pytorch, CNTK, Theano, tensorflow, keras, H2O, etc.
- HPC: C/C++, CUDA, vector processing

11. Background
- linear algebra
- multivariate calculus
- probability theory and statistics
- information theory
- mathematical (e.g., convex) optimization



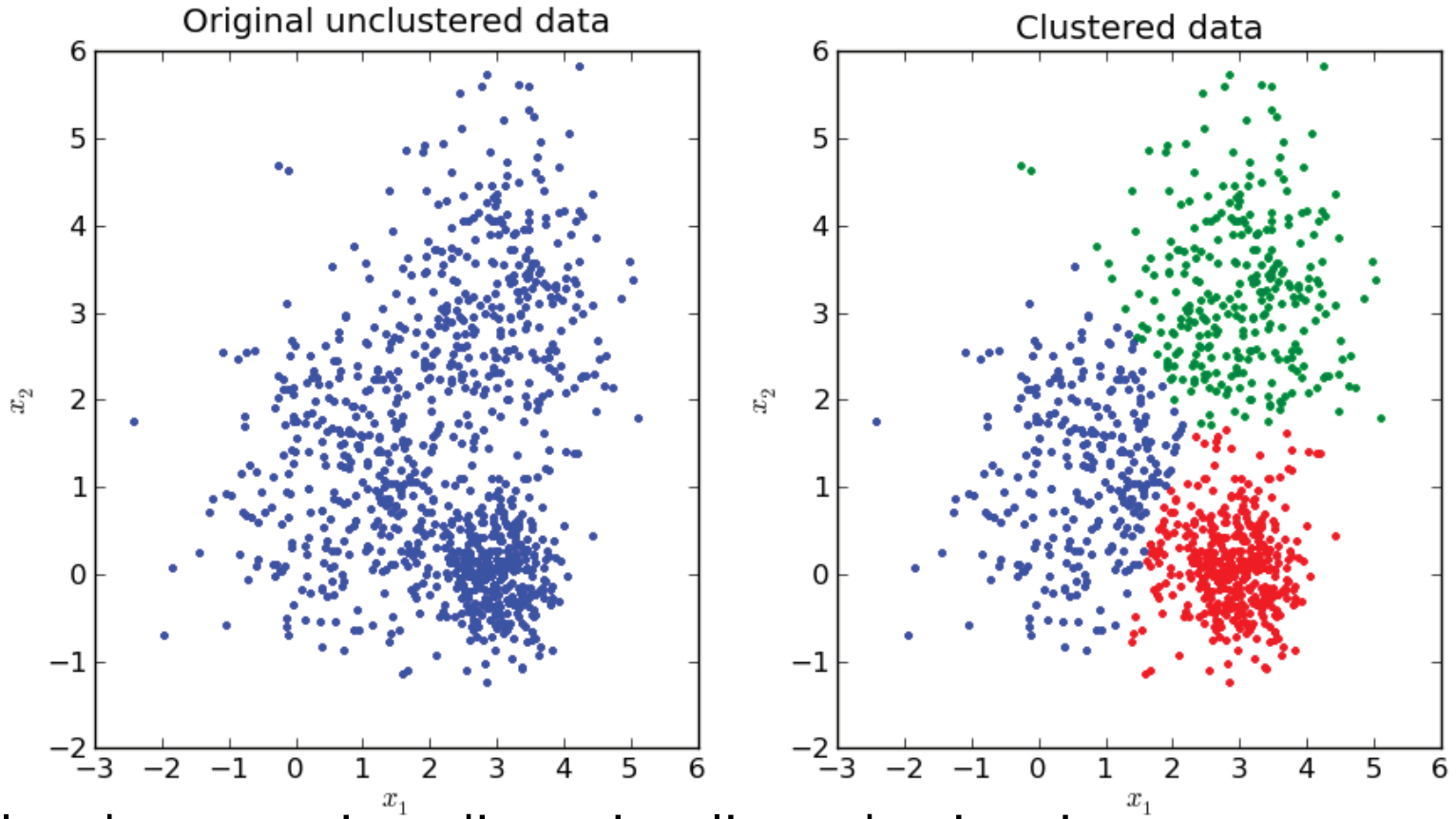
1.2 Determinant

Let $\mathbf{A}$ be an $n \times n$ matrix.

$$\begin{aligned}
\det(\mathbf{A}) &= \textstyle\prod_i \lambda_i \qquad \lambda_i = \mathrm{eig}(\mathbf{A}) \\
\det(c\mathbf{A}) &= c^n \det(\mathbf{A}), \qquad \text{if } \mathbf{A} \in \mathbb{R}^{n\times n} \\
\det(\mathbf{A}^T) &= \det(\mathbf{A}) \\
\det(\mathbf{AB}) &= \det(\mathbf{A})\det(\mathbf{B}) \\
\det(\mathbf{A}^{-1}) &= 1/\det(\mathbf{A}) \\
\det(\mathbf{A}^n) &= \det(\mathbf{A})^n \\
\det(\mathbf{I} + \mathbf{uv}^T) &= 1 + \mathbf{u}^T\mathbf{v}
\end{aligned}$$

Clustering

- Clustering lies at the core of unsupervised learning.
- Clustering is the process of finding natural groupings in data, and then assigning to each data sample the group ID of the group it lives in.



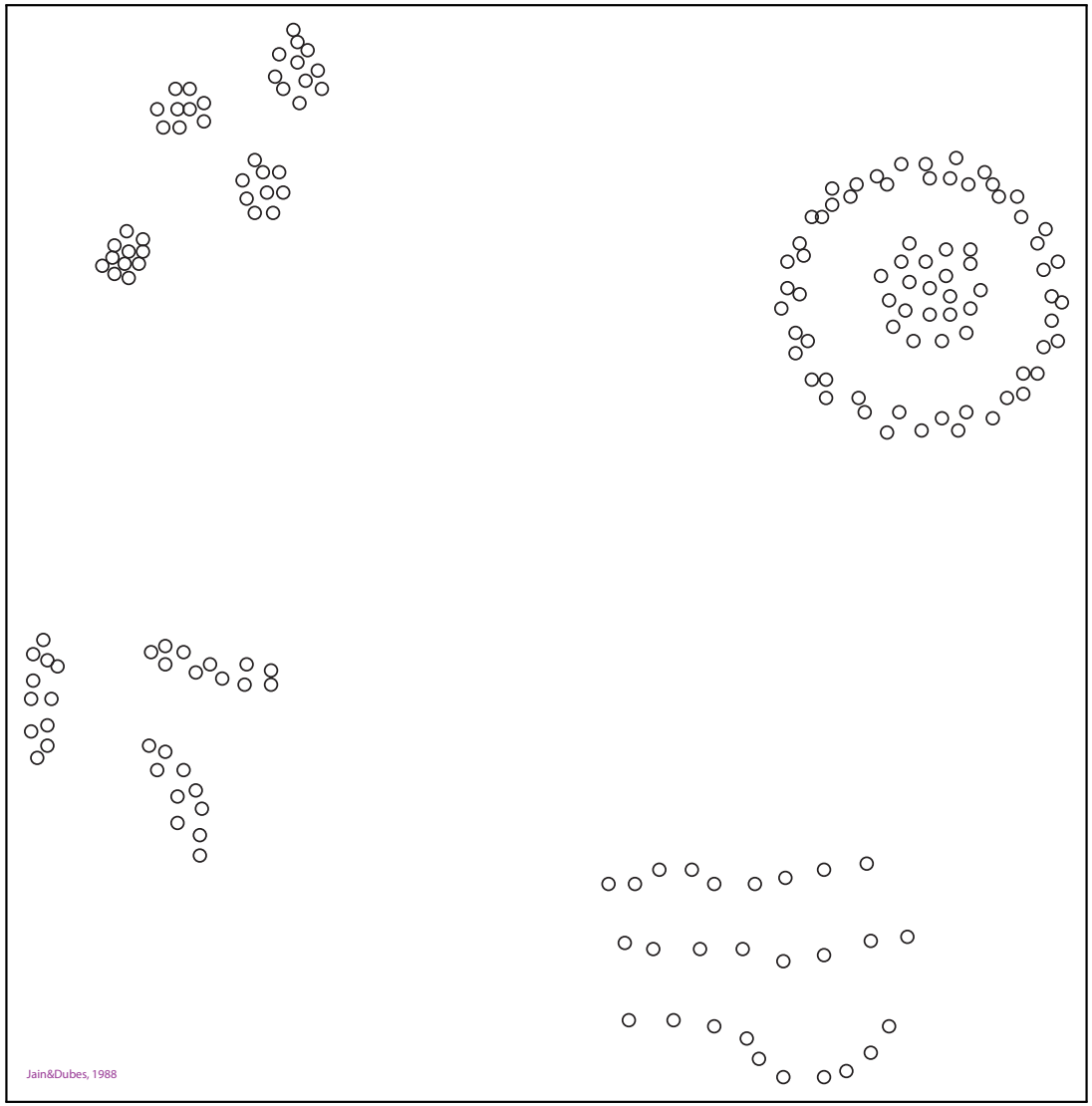
- Related to grouping, dimensionality reduction, image segmentation, compression, vector quantization, recommendation systems, density estimation, and anomaly detection.

Clustering of Clusterings

- Goal: partition (or group) data into k (typically) pre-specified clusters.
- Each cluster is assigned a representative, e.g., mean or medoid, based on the minimization of an objective.
- Clustering may be hard (each sample fully and completely in one and only one cluster) or soft (each sample has a probability distribution over clusters). Hence, hard is a special case.
- Often, clustering algorithms desire members of the same cluster to be near each other (according to a distance), members of the same cluster are alike in some way. Members of different clusters be far from each other, or are not alike.
- To cluster, need notion of "nearness", and this is typically done via a "distance", "distortion", or "dissimilarity" measure $d : \mathbb{R}^m \times \mathbb{R}^m \to \mathbb{R}_+$ that maps pairs of points to a (non-negative) real value.
- I.e., $d(x, x')$ is a non-negative real value expressing how dissimilar x and x' are from each other.

Clustering and Scale

- How many clusters? 4? 9? 12?



Clustering Aims and Goals

Clustering can have different goals (from: Hennig, "What are the true clusters?", 2015)

- delimitation of species of plants or animals in biology,
- medical classification of diseases,
- discovery and segmentation of settlements and periods in archaeology,
- image segmentation and object recognition,
- social stratification,
- market segmentation,
- efficient organization of data bases for search queries.
- exploratory data analysis, find interesting patterns without prescribing any specific interpretation, create new research questions/hypotheses,
- information reduction, compression, structuring entities from any subject area for simplification, effective communication, or effective access/action such as complexity reduction for further data analysis, or classification systems, vector quantization.
- examine correspondence of a data clustering with other (hypothesized or data-driven) groupings or characteristics, outlier or anomaly detection.

Typically Desirable Properties of a Clustering

Usually not all can be true at same time. from (Hennig, "What are the true clusters?", 2015)

- Within-cluster dissimilarities should be small.
- Between-cluster dissimilarities should be large.
- Clusters should well fit homogeneous probability models, e.g., Gaussian or a uniform distribution on a convex set, or by linear, time series or spatial process models.
- Members of a cluster should be well represented by a representative (e.g., centroid).
- The dissimilarity matrix of the data should be well represented by the resulting clustering
- Clusters should be stable, a bit of noise shouldn't change it much if at all.
- Clusters should correspond to connected areas in data space with high density.
- The areas in data space corresponding to clusters should have certain characteristics (such as being a convex set, linear, or manifold).
- Clusters characterizeable using a small number of variables (avoid curse of dimensionality issues and helps with interpretation, e.g., t-SNE, U-MAP).
- Clusters should correspond well to an externally given partition or values of one or more variables that were not used for computing the clustering (e.g., class labels).
- Features should be approximately independent within clusters.

Center based clustering: k-means objective

- k-means objective, within-cluster sum of squares (WCSS)

$$J_{\text{WCSS}}(\mathsf{C}_1, \mathsf{C}_2, \ldots, \mathsf{C}_k) = \sum_{j=1}^{k} \sum_{i \in \mathsf{C}_j} d(x^{(i)}, \mu_j)^2 \tag{10.30}$$

  often we take $\mu_j = \frac{1}{|\mathsf{C}_j|} \sum_{i \in \mathsf{C}_j} x^{(i)}$, in which case, for certain d's (e.g., Euclidean distance) the above becomes:

$$J_{\text{WCSS}}(\mathsf{C}_1, \mathsf{C}_2, \ldots, \mathsf{C}_k) = \sum_{j=1}^{k} \frac{1}{2|\mathsf{C}_j|} \sum_{i,\ell \in \mathsf{C}_j} d(x^{(i)}, x^{(\ell)})^2, \tag{10.31}$$

  like the sum of the average spread/dispersion within each cluster.
- "facility location", each point assigned square closest center:

$$J_{WCSS}(\mu_1, \ldots, \mu_k) = \sum_{i=1}^{n} \min_{j \in [k]} d(x^{(i)}, \mu_j)^2 \tag{10.32}$$

- Since squared distance, more weight on outliers than WCSD but not as much as MD.

k-means clustering: random k-partition seeds

- One of the simplest and most widely used clustering procedures.

Algorithm 2: k-means procedure

Input: Training data set $\mathcal{D} = \left\{x^{(i)}\right\}$ size n, number of clusters k

Output: Groupings $\mathsf{C}_1, \mathsf{C}_2, \ldots, \mathsf{C}_k \subseteq [n]$ and means $\{\mu_i\}_{i=1}^k$

Random k-partition: assign samples uniformly at random to k clusters;

repeat

  For $j \in [k]$, compute cluster mean $\mu_j = \frac{1}{|\mathsf{C}_k|} \sum_{i \in \mathsf{C}_k} x^{(i)}$;

  For $i \in [n]$, assign $x^{(i)}$ to cluster j if $\|x^{(i)} - \mu_j\| \leq \|x^{(i)} - \mu_{j'}\|, \forall j'$;

until *Until clusters* $\{\mathsf{C}_j\}_{j \in k}$ *don't change between iterations.*;

Output the k means and clusters $\{\mu_j, \mathsf{C}_j\}_{j \in [k]}$;

Seed Variations

- Random k-partition: Start with uniformly at random assigning each point to one of the k clusters, and then compute the means for each cluster.
- Forgy (1965) seed method: Choose k means by choosing k points from the data set uniformly at random.
- k-means++ initialization: Define $d_i(x)$ be the shortest distance from x to the closest of the i means we've already chosen. (1) choose a sample for μ_1 uniformly at random. (2) For $i = 2, \ldots, k$, choose a sample for μ_i with probability $d_{i-1}(x)^2 / \sum_{x' \in \mathcal{D}} d_{i-1}(x')^2$. Thus, each next mean has high probability of being far away from the other means.
- Above procedures are random and can be repeated multiple times to try to find best clustering.
- Many other seed methods, see "A comparative study of efficient initialization methods for the k-means clustering algorithm", Celebi, Kingravi, and Vela, 2013.

k-means analysis

- Data $\mathcal{D} = \left\{x^{(i)}\right\}_{i=1}^{n}$, under k centers $\mu_1, \ldots, \mu_k \in \mathbb{R}^m$, each sample can be given probability.
- Uniform mixture of spherical Gaussians centered at k-means centers:

$$\Pr(x) \propto \frac{1}{k} \sum_{j=1}^{k} \exp(-\|x - \mu_j\|^2), \qquad \text{so } p(j) = 1/k \tag{10.30}$$

- Let $\mu(x)$ indicate the center nearest to x. Due to exponential decay,

$$\sum_{i=j}^{k} \exp(-\|x - \mu_j\|^2) \approx \exp(-\|x - \mu(x)\|^2) \tag{10.31}$$

- Thus, probability of the data can be represented as:

$$\Pr(\mathcal{D}) \approx \text{const.} \times \exp\left(-\sum_{i=1}^{n} \|x^{(i)} - \mu(x^{(i)})\|^2\right) \tag{10.32}$$

so minimizing sum of squared distance to centers can be seen as a form of maximum likelihood estimate of the centers.

k-means convergence

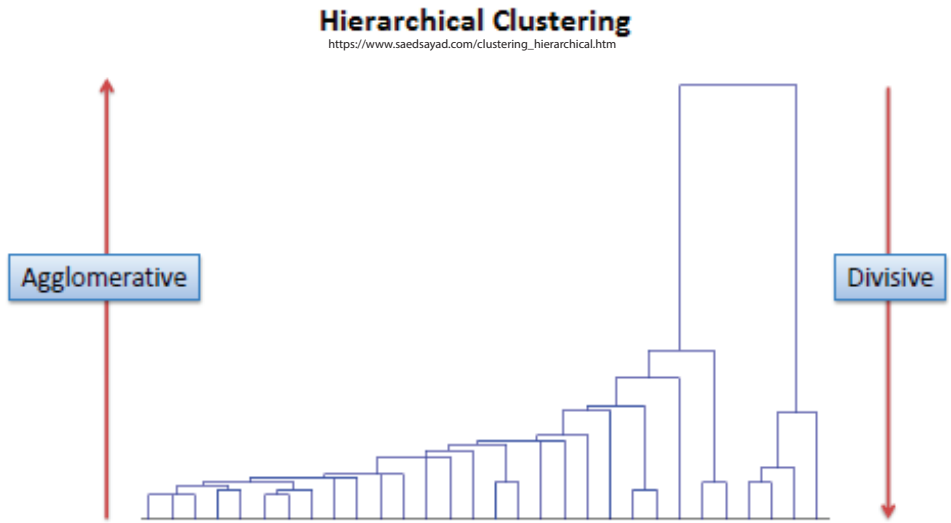
- Is k-means guaranteed to finish? Yes, k-means is guaranteed to converge in a finite number of steps to a local minimum of the k-means objective J_{WCSS}.
- Each iteration can only reduce J_{WCSS}.
  1. Assigning μ_j as the cluster means reduces the objective by the above lemma (Lemma **??**).
  2. Also, re-assigning each point to the cluster with the closest center also reduces J_{WCSS}.
  3. There are a finite number of clusters.
  4. Each iteration either reduces the objective, or stops if the clusters do not change.

K-Medoids Algorithm

- In k-medoids, we find k samples that are representative of the remaining samples, rather than finding k-means. I.e., the result of the clustering is a subset of samples.
- Like k-means but rather than compute the mean of each cluster, we find the representative samples each cluster that minimizes the objective.
- Typically used with k-median (within-cluster sum of distances) objective which is less sensitive to noise and outliers.

Agglomerative (or Hierarchical) Clustering

- Agglomerative clustering starts with each point being its own cluster, and we-repeatedly greedily merge together the two clusters $\mathsf{C}_i, \mathsf{C}_j$ that are closest together according to $d(\mathsf{C}_i, \mathsf{C}_j)$.
- This continues until we have the desired number k of clusters, or we can form an entire tree.
- Need distance between clusters. One classic version is single-linkage clustering, where $d(\mathsf{C}_i, \mathsf{C}_j) = \min_{x \in \mathsf{C}_i, x' \in \mathsf{C}_j} d(x, x')$.
- This is a bottom-up clustering. Other forms are top down, known as divisive clustering were we start with one big cluster and repeatedly make the best split. Ex: Graph cut.



Review: Multivariate Gaussian

- Recall again the Gaussian distribution from Lecture 1

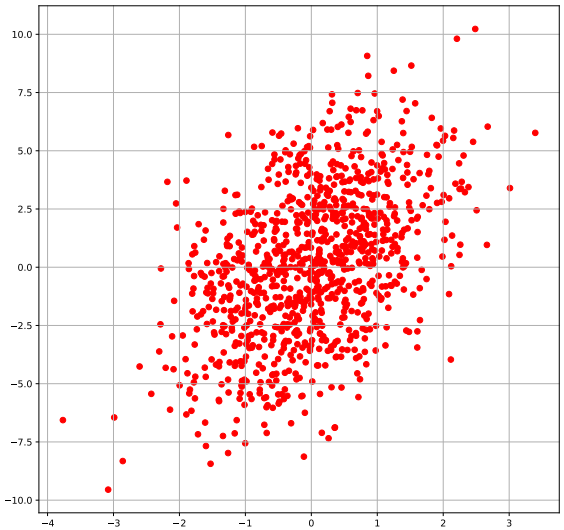
Multivariate Gaussian (Normal), and samples thereof

- $\mathcal{D} = \{x_1, x_2, \ldots, x_n\}$ is series of n independent and identically distributed (i.i.d.) m-dimensional real-valued samples $\forall i, x_i \in \mathbb{R}^m$ from a Gaussian (or normal) distribution

$$x_i \sim \Pr(x|\mu, C) = \frac{1}{|2\pi C|^{1/2}} \exp\left(-\frac{1}{2}(x-\mu)^\intercal C^{-1}(x-\mu)\right) \tag{10.3}$$

  where $\mu \in \mathbb{R}^m$ is a mean vector and C is $m \times m$ a positive definite covariance matrix. Notationally, we often say $x \sim \mathcal{N}(x; \mu, C)$.
- Symmetric matrix $C \in \mathbb{R}^{m \times m}$ is positive definite ($\in \mathbb{P}(m)$) if all of its Eigenvalues are positive (alternatively $\langle x, Cx \rangle > 0, \forall x \neq 0$).

- Example sample of a 2D Gaussian.



Multivariate Gaussian Mixtures Model (GMM)

- Recall multivariate Gaussian distribution (notation: C is a cluster and C is a covariance matrix).

Multivariate Gaussian Mixtures Model (GMM)

- Recall multivariate Gaussian distribution (notation: C is a cluster and C is a covariance matrix).
- Let $\mathcal{N}(x; \mu, C)$ be a multivariate Gaussian. We can form a Gaussian mixture model (GMM) distribution, with k means and covariance matrices, as:

$$\Pr(x) = \sum_{j=1}^{k} p(j)\mathcal{N}(x; \mu_j, C_j) = \sum_{j=1}^{k} p(x|j)p(j), \tag{10.1}$$

$p(x|j)$ is a Gaussian conditioned on being in mixture component j.

Multivariate Gaussian Mixtures Model (GMM)

- Recall multivariate Gaussian distribution (notation: C is a cluster and C is a covariance matrix).
- Let $\mathcal{N}(x; \mu, C)$ be a multivariate Gaussian. We can form a Gaussian mixture model (GMM) distribution, with k means and covariance matrices, as:

$$\Pr(x) = \sum_{j=1}^{k} p(j)\mathcal{N}(x; \mu_j, C_j) = \sum_{j=1}^{k} p(x|j)p(j), \tag{10.1}$$

  $p(x|j)$ is a Gaussian conditioned on being in mixture component j.
- Posterior probability of a point $x \in \mathbb{R}^m$ of being a member of the j^{th} component of the mixture:

$$p(j|x) = \frac{p(x|j)p(j)}{p(x)} = \frac{p(x|j)p(j)}{\sum_{j'=1}^{k} p(x|j')p(j')} \tag{10.2}$$

Multivariate Gaussian Mixtures Model (GMM)

- Recall multivariate Gaussian distribution (notation: C is a cluster and C is a covariance matrix).
- Let $\mathcal{N}(x; \mu, C)$ be a multivariate Gaussian. We can form a Gaussian mixture model (GMM) distribution, with k means and covariance matrices, as:

$$\Pr(x) = \sum_{j=1}^{k} p(j)\mathcal{N}(x; \mu_j, C_j) = \sum_{j=1}^{k} p(x|j)p(j), \tag{10.1}$$

  $p(x|j)$ is a Gaussian conditioned on being in mixture component j.
- Posterior probability of a point $x \in \mathbb{R}^m$ of being a member of the j^{th} component of the mixture:

$$p(j|x) = \frac{p(x|j)p(j)}{p(x)} = \frac{p(x|j)p(j)}{\sum_{j'=1}^{k} p(x|j')p(j')} \tag{10.2}$$

- We can think of μ_j as cluster center and C_j as a spread. Hence $p(j|x)$ is the probability of x being in cluster j.

Fitting Mixtures of Multivariate Gaussian as Clustering

- We are given data $\mathcal{D} = \left\{x^{(i)}\right\}_{i=1}^{n}$ where for all i, $x^{(i)} \in \mathbb{R}^m$ and we assume $x^{(i)} \sim \Pr_{\mathsf{GMM}(\theta_{\mathsf{true}})}(x)$, samples from some m-D multivariate k-component Gaussian mixture. We don't know the parameters θ_{true} but we want to learn them, which constitutes a form of clustering.

Fitting Mixtures of Multivariate Gaussian as Clustering

- We are given data $\mathcal{D} = \left\{x^{(i)}\right\}_{i=1}^{n}$ where for all i, $x^{(i)} \in \mathbb{R}^m$ and we assume $x^{(i)} \sim \Pr_{\mathsf{GMM}(\theta_{\mathsf{true}})}(x)$, samples from some m-D multivariate k-component Gaussian mixture. We don't know the parameters θ_{true} but we want to learn them, which constitutes a form of clustering.
- Given data $\mathcal{D}$, find the set of means and covariances to solve:

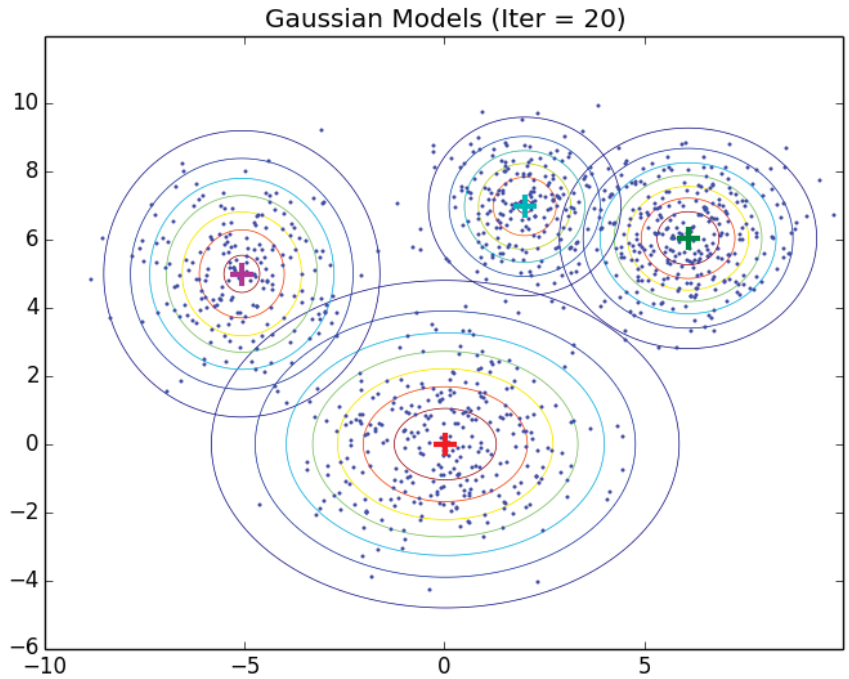
$$\max_{\{\mu_j, C_j, p(j)\}_{j=1}^{k}} \Pr(\mathcal{D}) = \prod_{i=1}^{n} \Pr\left(x^{(i)} | \{\mu_j, C_j, p(j)\}_{j=1}^{k}\right) \tag{10.3}$$

Fitting Mixtures of Multivariate Gaussian as Clustering

- We are given data $\mathcal{D} = \left\{x^{(i)}\right\}_{i=1}^{n}$ where for all i, $x^{(i)} \in \mathbb{R}^m$ and we assume $x^{(i)} \sim \Pr_{\text{GMM}(\theta_{\text{true}})}(x)$, samples from some m-D multivariate k-component Gaussian mixture. We don't know the parameters θ_{true} but we want to learn them, which constitutes a form of clustering.
- Given data $\mathcal{D}$, find the set of means and covariances to solve:

$$\max_{\{\mu_j, C_j, p(j)\}_{j=1}^{k}} \Pr(\mathcal{D}) = \prod_{i=1}^{n} \Pr\left(x^{(i)} | \{\mu_j, C_j, p(j)\}_{j=1}^{k}\right) \tag{10.3}$$

- Hence, maximum likelihood estimation of a Gaussian mixture model (GMM) can be seen as (soft, or probabilistic) clustering.



Fitting Mixtures of Multivariate Gaussian: Optimization

- As is typical, write objective to maximize as the log probability:

$$\log \Pr(\mathcal{D}) = \sum_{i=1}^{n} \log \Pr\left(x^{(i)} | \{\mu_j, C_j, p(j)\}_{j=1}^{k}\right) \tag{10.4}$$

$$= \sum_{i=1}^{n} \log \left[\sum_{j=1}^{k} p(x^{(i)} | \mu_j, C_j) p(j)\right] \tag{10.5}$$

$$= \sum_{i=1}^{n} \log \Pr_{\theta}(x^{(i)}) \tag{10.6}$$

Fitting Mixtures of Multivariate Gaussian: Optimization

- As is typical, write objective to maximize as the log probability:

$$\log \Pr(\mathcal{D}) = \sum_{i=1}^{n} \log \Pr\left(x^{(i)} | \{\mu_j, C_j, p(j)\}_{j=1}^{k}\right) \tag{10.4}$$

$$= \sum_{i=1}^{n} \log \left[\sum_{j=1}^{k} p(x^{(i)} | \mu_j, C_j) p(j)\right] \tag{10.5}$$

$$= \sum_{i=1}^{n} \log \Pr_{\theta}(x^{(i)}) \tag{10.6}$$

- Notation: Let $\alpha_j = p(j)$, $\theta_j = (\mu_j, C_j)$ and $\theta = \{\alpha_j, \theta_j\}_{j=1}^{k}$ be all the parameters.

Fitting Mixtures of Multivariate Gaussian: Optimization

- As is typical, write objective to maximize as the log probability:

$$\log \Pr(\mathcal{D}) = \sum_{i=1}^{n} \log \Pr\left(x^{(i)} | \{\mu_j, C_j, p(j)\}_{j=1}^{k}\right) \tag{10.4}$$

$$= \sum_{i=1}^{n} \log \left[\sum_{j=1}^{k} p(x^{(i)} | \mu_j, C_j) p(j)\right] \tag{10.5}$$

$$= \sum_{i=1}^{n} \log \Pr_{\theta}(x^{(i)}) \tag{10.6}$$

- Notation: Let $\alpha_j = p(j)$, $\theta_j = (\mu_j, C_j)$ and $\theta = \{\alpha_j, \theta_j\}_{j=1}^{k}$ be all the parameters.
- Taking derivatives w.r.t. the parameters and setting to zero does not yield a close-form solution.

Fitting Mixtures of Multivariate Gaussian: Optimization

- As is typical, write objective to maximize as the log probability:

$$\log \Pr(\mathcal{D}) = \sum_{i=1}^{n} \log \Pr\left(x^{(i)} | \{\mu_j, C_j, p(j)\}_{j=1}^{k}\right) \tag{10.4}$$

$$= \sum_{i=1}^{n} \log \left[\sum_{j=1}^{k} p(x^{(i)} | \mu_j, C_j) p(j)\right] \tag{10.5}$$

$$= \sum_{i=1}^{n} \log \Pr_{\theta}(x^{(i)}) \tag{10.6}$$

- Notation: Let $\alpha_j = p(j)$, $\theta_j = (\mu_j, C_j)$ and $\theta = \{\alpha_j, \theta_j\}_{j=1}^{k}$ be all the parameters.
- Taking derivatives w.r.t. the parameters and setting to zero does not yield a close-form solution.
- This is not a convex problem, problem is the log of a sum does not easily simplify. Could do SGD, but there is an easier way w/o learning rate, namely the Expectation Maximization (EM) algorithm.

What if we knew the cluster ids of each sample?

- Suppose that for every $x^{(i)}$ we happened to know the appropriate cluster, say $y^{(i)} \in [k]$, so that $\mathsf{C}_j = \left\{i : y^{(i)} = j\right\}$.

What if we knew the cluster ids of each sample?

- Suppose that for every $x^{(i)}$ we happened to know the appropriate cluster, say $y^{(i)} \in [k]$, so that $\mathsf{C}_j = \{i : y^{(i)} = j\}$.
- Then data would take the form $\mathcal{D}_{\text{complete}} = (x^{(i)}, y^{(i)})_{i=1}^n$ and the complete log-likelihood is:

$$\log \Pr(\mathcal{D}_{\text{complete}}) = \sum_{i=1}^{n} \log p(x^{(i)} | \mu_{y^{(i)}}, C_{y^{(i)}}) p(y^{(i)}) \tag{10.7}$$

$$= \sum_{j=1}^{k} \sum_{i \in \mathsf{C}_j} \log \Big(p(x^{(i)} | \mu_j, C_j) p(j) \Big) \tag{10.8}$$

What if we knew the cluster ids of each sample?

- Suppose that for every $x^{(i)}$ we happened to know the appropriate cluster, say $y^{(i)} \in [k]$, so that $\mathsf{C}_j = \left\{i : y^{(i)} = j\right\}$.
- Then data would take the form $\mathcal{D}_{\text{complete}} = (x^{(i)}, y^{(i)})_{i=1}^n$ and the complete log-likelihood is:

$$\log \Pr(\mathcal{D}_{\text{complete}}) = \sum_{i=1}^{n} \log p(x^{(i)}|\mu_{y^{(i)}}, C_{y^{(i)}})p(y^{(i)}) \tag{10.7}$$

$$= \sum_{j=1}^{k} \sum_{i \in \mathsf{C}_j} \log \Big(p(x^{(i)}|\mu_j, C_j)p(j) \Big) \tag{10.8}$$

- This breaks the problem into k independent multivariate Gaussian estimation problems with closed form solutions, e.g., μ_j (resp. C_j) is the mean (resp. covariance) of samples with index C_j and $p(j) = |\mathsf{C}_j|/n$.

What if we knew the cluster ids of each sample?

- Suppose that for every $x^{(i)}$ we happened to know the appropriate cluster, say $y^{(i)} \in [k]$, so that $\mathsf{C}_j = \left\{i : y^{(i)} = j\right\}$.
- Then data would take the form $\mathcal{D}_{\text{complete}} = (x^{(i)}, y^{(i)})_{i=1}^n$ and the complete log-likelihood is:

$$\log \Pr(\mathcal{D}_{\text{complete}}) = \sum_{i=1}^{n} \log p(x^{(i)}|\mu_{y^{(i)}}, C_{y^{(i)}})p(y^{(i)}) \tag{10.7}$$

$$= \sum_{j=1}^{k} \sum_{i \in \mathsf{C}_j} \log \Big(p(x^{(i)}|\mu_j, C_j)p(j) \Big) \tag{10.8}$$

- This breaks the problem into k independent multivariate Gaussian estimation problems with closed form solutions, e.g., μ_j (resp. C_j) is the mean (resp. covariance) of samples with index C_j and $p(j) = |\mathsf{C}_j|/n$.
- Hence, exists closed form solutions to maximum likelihood estimate.

What if we knew the cluster ids of each sample?

- Suppose that for every $x^{(i)}$ we happened to know the appropriate cluster, say $y^{(i)} \in [k]$, so that $\mathsf{C}_j = \left\{i : y^{(i)} = j\right\}$.
- Then data would take the form $\mathcal{D}_{\text{complete}} = (x^{(i)}, y^{(i)})_{i=1}^n$ and the complete log-likelihood is:

$$\log \Pr(\mathcal{D}_{\text{complete}}) = \sum_{i=1}^{n} \log p(x^{(i)}|\mu_{y^{(i)}}, C_{y^{(i)}}) p(y^{(i)}) \tag{10.7}$$

$$= \sum_{j=1}^{k} \sum_{i \in \mathsf{C}_j} \log \left(p(x^{(i)}|\mu_j, C_j) p(j) \right) \tag{10.8}$$

- This breaks the problem into k independent multivariate Gaussian estimation problems with closed form solutions, e.g., μ_j (resp. C_j) is the mean (resp. covariance) of samples with index C_j and $p(j) = |\mathsf{C}_j|/n$.
- Hence, exists closed form solutions to maximum likelihood estimate.
- Cluster ids said to be unknown, hidden, or latent. (known unknowns).

Probabilistic Clustering: EM algorithm with GMMs

- Suppose we had an initial set of parameters θ^g from which we could produce joint distribution $p(j, x; \theta^g)$ as well as posterior $p(j|x; \theta^g)$.

Probabilistic Clustering: EM algorithm with GMMs

- Suppose we had an initial set of parameters θ^g from which we could produce joint distribution $p(j, x; \theta^g)$ as well as posterior $p(j|x; \theta^g)$.
- Use this to compute the expected value of the complete log-likelihood.

$$E_{\theta^g}\left[\log \Pr_{\theta}\left(\left\{x^{(i)}, Y^{(i)}\right\}_{i=1}^{n}\right)\right] \tag{10.9}$$

$$= \sum_{i=1}^{n} \sum_{y^{(i)}=1}^{k} \left[\log\left(p_\theta(y^{(i)}) p(x^{(i)}|y^{(i)}; \theta_{y^{(i)}})\right) \prod_{i=1}^{n} p(y^{(i)}|x^{(i)}, \theta^g)\right] \tag{10.10}$$

$$= Q(\theta, \theta^g) = \text{ the auxiliary function.} \tag{10.11}$$

Probabilistic Clustering: EM algorithm with GMMs

- Suppose we had an initial set of parameters θ^g from which we could produce joint distribution $p(j, x; \theta^g)$ as well as posterior $p(j|x; \theta^g)$.
- Use this to compute the expected value of the complete log-likelihood.

$$E_{\theta^g}\left[\log \Pr_{\theta}\left(\left\{x^{(i)}, Y^{(i)}\right\}_{i=1}^{n}\right)\right] \tag{10.9}$$

$$= \sum_{i=1}^{n}\sum_{y^{(i)}=1}^{k}\left[\log\left(p_\theta(y^{(i)})p(x^{(i)}|y^{(i)};\theta_{y^{(i)}})\right)\prod_{i=1}^{n}p(y^{(i)}|x^{(i)},\theta^g)\right] \tag{10.10}$$

$$= Q(\theta, \theta^g) = \text{ the auxiliary function.} \tag{10.11}$$

- The Q function, expected complete log likelihood, strongly related to a concave lower bound on log likelihood (as we will see).

Probabilistic Clustering: EM algorithm with GMMs

- Suppose we had an initial set of parameters θ^g from which we could produce joint distribution $p(j, x; \theta^g)$ as well as posterior $p(j|x; \theta^g)$.
- Use this to compute the expected value of the complete log-likelihood.

$$E_{\theta^g}\left[\log \Pr_{\theta}\left(\left\{x^{(i)}, Y^{(i)}\right\}_{i=1}^{n}\right)\right] \tag{10.9}$$

$$= \sum_{i=1}^{n}\sum_{y^{(i)}=1}^{k}\left[\log\left(p_\theta(y^{(i)})p(x^{(i)}|y^{(i)};\theta_{y^{(i)}})\right)\prod_{i=1}^{n}p(y^{(i)}|x^{(i)},\theta^g)\right] \tag{10.10}$$

$$= Q(\theta, \theta^g) = \text{ the auxiliary function.} \tag{10.11}$$

- The Q function, expected complete log likelihood, strongly related to a concave lower bound on log likelihood (as we will see).
- We maximize $Q(\theta, \theta^g)$ w.r.t. θ (the next parameters) while holding the previous (or guessed) parameters θ^g fixed.

Probabilistic Clustering: EM algorithm with GMMs

- Suppose we had an initial set of parameters θ^g from which we could produce joint distribution $p(j, x; \theta^g)$ as well as posterior $p(j|x; \theta^g)$.
- Use this to compute the expected value of the complete log-likelihood.

$$E_{\theta^g}\left[\log \Pr_\theta\left(\left\{x^{(i)}, Y^{(i)}\right\}_{i=1}^n\right)\right] \tag{10.9}$$

$$= \sum_{i=1}^n \sum_{y^{(i)}=1}^k \left[\log\left(p_\theta(y^{(i)}) p(x^{(i)}|y^{(i)}; \theta_{y^{(i)}})\right) \prod_{i=1}^n p(y^{(i)}|x^{(i)}, \theta^g)\right] \tag{10.10}$$

$$= Q(\theta, \theta^g) = \text{ the auxiliary function.} \tag{10.11}$$

- The Q function, expected complete log likelihood, strongly related to a concave lower bound on log likelihood (as we will see).
- We maximize $Q(\theta, \theta^g)$ w.r.t. θ (the next parameters) while holding the previous (or guessed) parameters θ^g fixed.
- I.e., set $\frac{\partial}{\partial\theta} Q(\theta, \theta^g) = 0$ and solve for θ.

Probabilistic Clustering: EM algorithm with GMMs

- Solving for $\theta = \{(\alpha_j, \mu_j, C_j)\}_{j=1}^k$ in $\frac{\partial}{\partial\theta} Q(\theta, \theta^g) = 0$ does have a exact closed form solution in the GMM case. A bit of algebra shows that:

$$\alpha_j = \frac{1}{n} \sum_{i=1}^n p(j|x^{(i)}, \theta^g) \tag{10.12}$$

$$\mu_j = \frac{\sum_{i=1}^n x^{(i)} p(j|x^{(i)}, \theta^g)}{\sum_{i=1}^n p(j|x^{(i)}, \theta^g)} \tag{10.13}$$

$$C_j = \frac{\sum_{i=1}^n (x^{(i)} - \mu_j)(x^{(i)} - \mu_j)^\mathsf{T} p(j|x^{(i)}, \theta^g)}{\sum_{i=1}^n p(j|x^{(i)}, \theta^g)} \tag{10.14}$$

Probabilistic Clustering: EM algorithm with GMMs

- Solving for $\theta = \{(\alpha_j, \mu_j, C_j)\}_{j=1}^k$ in $\frac{\partial}{\partial\theta} Q(\theta, \theta^g) = 0$ does have a exact closed form solution in the GMM case. A bit of algebra shows that:

$$\alpha_j = \frac{1}{n} \sum_{i=1}^n p(j|x^{(i)}, \theta^g) \tag{10.12}$$

$$\mu_j = \frac{\sum_{i=1}^n x^{(i)} p(j|x^{(i)}, \theta^g)}{\sum_{i=1}^n p(j|x^{(i)}, \theta^g)} \tag{10.13}$$

$$C_j = \frac{\sum_{i=1}^n (x^{(i)} - \mu_j)(x^{(i)} - \mu_j)^\mathsf{T} p(j|x^{(i)}, \theta^g)}{\sum_{i=1}^n p(j|x^{(i)}, \theta^g)} \tag{10.14}$$

- This gives us an iterative procedure. We start with $\theta^0 = \theta^g$ to produce θ^1, and use θ^t to produce θ^{t+1} until convergence.

Probabilistic Clustering: EM algorithm with GMMs

- Solving for $\theta = \{(\alpha_j, \mu_j, C_j)\}_{j=1}^k$ in $\frac{\partial}{\partial\theta}Q(\theta, \theta^g) = 0$ does have a exact closed form solution in the GMM case. A bit of algebra shows that:

$$\alpha_j = \frac{1}{n}\sum_{i=1}^n p(j|x^{(i)}, \theta^g) \tag{10.12}$$

$$\mu_j = \frac{\sum_{i=1}^n x^{(i)} p(j|x^{(i)}, \theta^g)}{\sum_{i=1}^n p(j|x^{(i)}, \theta^g)} \tag{10.13}$$

$$C_j = \frac{\sum_{i=1}^n (x^{(i)} - \mu_j)(x^{(i)} - \mu_j)^\mathsf{T} p(j|x^{(i)}, \theta^g)}{\sum_{i=1}^n p(j|x^{(i)}, \theta^g)} \tag{10.14}$$

- This gives us an iterative procedure. We start with $\theta^0 = \theta^g$ to produce θ^1, and use θ^t to produce θ^{t+1} until convergence.
- Each update gives a valid construct. E.g., we get a probability distribution for α_j, a real vector for each mean, and a positive definite matrix for each C_j. Hence, we do not need to worry about constraints to ensure proper form of parameters each iteration.

EM: Simplified Notation

Goal: find $\max_\theta p(x|\theta)$, where x is the data and $p(x|\theta) = \sum_y p(x, y|\theta)$, and θ is a set of parameters of the probability distribution.

- The incomplete log-likelihood: $\ell(\theta; x) = \log p(x|\theta) = \log \sum_y p(x, y|\theta)$.

EM: Simplified Notation

Goal: find $\max_\theta p(x|\theta)$, where x is the data and $p(x|\theta) = \sum_y p(x, y|\theta)$, and θ is a set of parameters of the probability distribution.

- The incomplete log-likelihood: $\ell(\theta; x) = \log p(x|\theta) = \log \sum_y p(x, y|\theta)$. Hence, our goal is $\max_\theta \ell(\theta; x)$.

EM: Simplified Notation

Goal: find $\max_\theta p(x|\theta)$, where x is the data and $p(x|\theta) = \sum_y p(x, y|\theta)$, and θ is a set of parameters of the probability distribution.

- The incomplete log-likelihood: $\ell(\theta; x) = \log p(x|\theta) = \log \sum_y p(x, y|\theta)$. Hence, our goal is $\max_\theta \ell(\theta; x)$.
- The complete log-likelihood, where we know y $\ell(\theta; x, y) = \log p(x, y|\theta)$.

EM: Simplified Notation

Goal: find $\max_\theta p(x|\theta)$, where x is the data and $p(x|\theta) = \sum_y p(x, y|\theta)$, and θ is a set of parameters of the probability distribution.

- The incomplete log-likelihood: $\ell(\theta; x) = \log p(x|\theta) = \log \sum_y p(x, y|\theta)$. Hence, our goal is $\max_\theta \ell(\theta; x)$.
- The complete log-likelihood, where we know y $\ell(\theta; x, y) = \log p(x, y|\theta)$.
- The expected complete log likelihood: $Q(\theta, \theta^g; x) = \sum_y p(y|x, \theta^g) \log p(x, y|\theta) = E_{p(y|x,\theta^g)}[\log p(x, y|\theta)]$ $= E_{Y \sim p(y|x,\theta^g)}[\log p(x, Y|\theta)]$.

EM: Simplified Notation

Goal: find $\max_\theta p(x|\theta)$, where x is the data and $p(x|\theta) = \sum_y p(x, y|\theta)$, and θ is a set of parameters of the probability distribution.

- The incomplete log-likelihood: $\ell(\theta; x) = \log p(x|\theta) = \log \sum_y p(x, y|\theta)$. Hence, our goal is $\max_\theta \ell(\theta; x)$.
- The complete log-likelihood, where we know y $\ell(\theta; x, y) = \log p(x, y|\theta)$.
- The expected complete log likelihood: $Q(\theta, \theta^g; x) = \sum_y p(y|x, \theta^g) \log p(x, y|\theta) = E_{p(y|x,\theta^g)}[\log p(x, y|\theta)]$ $= E_{Y \sim p(y|x,\theta^g)}[\log p(x, Y|\theta)]$.
- This simplified notation corresponds to earlier notation as follows: $x = (x^{(1)}, x^{(2)}, \ldots, x^{(n)})$ and $y = (y^{(1)}, y^{(2)}, \ldots, y^{(n)})$, along with the assumption that the samples are i.i.d. from the distribution we wish to learn, i.e., $p(x|\theta) = \prod_{i=1}^n p(x^{(i)}|\theta)$, etc.

The Classic EM Algorithm

- This procedure is called the expectation-maximization (EM) algorithm and has the following steps.

The Classic EM Algorithm

- This procedure is called the expectation-maximization (EM) algorithm and has the following steps.

(0) Start with θ^0 initial parameter guess (say using k-means). Set $t = 0$

The Classic EM Algorithm

- This procedure is called the expectation-maximization (EM) algorithm and has the following steps.

(0) Start with θ^0 initial parameter guess (say using k-means). Set $t = 0$

(1) E-step: Form $Q(\theta, \theta^t) = \sum_y p(y|x, \theta^t) \log p(x, y|\theta) = E_{p(y|x,\theta^t)}[\log p(x, y|\theta)]$ where θ is the set of parameters to optimize over and θ^t is a constant.

The Classic EM Algorithm

- This procedure is called the expectation-maximization (EM) algorithm and has the following steps.

(0) Start with θ^0 initial parameter guess (say using k-means). Set $t = 0$

(1) E-step: Form $Q(\theta, \theta^t) = \sum_y p(y|x, \theta^t) \log p(x, y|\theta) = E_{p(y|x,\theta^t)}[\log p(x, y|\theta)]$ where θ is the set of parameters to optimize over and θ^t is a constant.

(2) M-step: Optimize $\theta' \in \operatorname{argmax}_\theta Q(\theta, \theta^t)$.

The Classic EM Algorithm

- This procedure is called the expectation-maximization (EM) algorithm and has the following steps.

(0) Start with θ^0 initial parameter guess (say using k-means). Set $t = 0$

(1) E-step: Form $Q(\theta, \theta^t) = \sum_y p(y|x, \theta^t) \log p(x, y|\theta) = E_{p(y|x,\theta^t)}[\log p(x, y|\theta)]$ where θ is the set of parameters to optimize over and θ^t is a constant.

(2) M-step: Optimize $\theta' \in \operatorname{argmax}_\theta Q(\theta, \theta^t)$.

(3) Set $\theta^{t+1} \leftarrow \theta'$ and if there is enough change, $t \leftarrow t + 1$ and go to (1), otherwise stop and output final θ^t.

The Classic EM Algorithm

- This procedure is called the expectation-maximization (EM) algorithm and has the following steps.

(0) Start with θ^0 initial parameter guess (say using k-means). Set $t = 0$

(1) E-step: Form $Q(\theta, \theta^t) = \sum_y p(y|x, \theta^t) \log p(x, y|\theta) = E_{p(y|x,\theta^t)}[\log p(x, y|\theta)]$ where θ is the set of parameters to optimize over and θ^t is a constant.

(2) M-step: Optimize $\theta' \in \operatorname{argmax}_\theta Q(\theta, \theta^t)$.

(3) Set $\theta^{t+1} \leftarrow \theta'$ and if there is enough change, $t \leftarrow t + 1$ and go to (1), otherwise stop and output final θ^t.

- "enough change" can mean the parameters have changed sufficiently, but more commonly, stop if $\ell(\theta^{t+1}; x) - \ell(\theta^t; x) \leq \tau$ where τ is a threshold. Relative change is often used.

Probabilistic Clustering: EM algorithm with GMMs

Theorem 10.3.1

Each iteration of the EM algorithm can only increase the incomplete data log likelihood (it will never decrease).

In practice, it increases until it convergence.

Theorem 10.3.2

The EM algorithm converges to a local maximum of the incomplete data log likelihood function.

- Note, we can also use gradient descent on the complete data log likelihood, but it requires another hyperparameter, the learning rate, and also projection (or other tricks) to make sure parameters stay in right space.

Jensen's Inequality

- A function f is convex if for any u, v and $\lambda \in [0, 1]$, we have $f(\lambda u + (1 - \lambda)v) \leq \lambda f(u) + (1 - \lambda)f(v)$.

Jensen's Inequality

- A function f is convex if for any u, v and $\lambda \in [0, 1]$, we have $f(\lambda u + (1 - \lambda)v) \leq \lambda f(u) + (1 - \lambda)f(v)$.
- A function f is concave if for any u, v and $\lambda \in [0, 1]$, we have $f(\lambda u + (1 - \lambda)v) \geq \lambda f(u) + (1 - \lambda)f(v)$.

Jensen's Inequality

- A function f is convex if for any u, v and $\lambda \in [0, 1]$, we have $f(\lambda u + (1 - \lambda)v) \leq \lambda f(u) + (1 - \lambda)f(v)$.
- A function f is concave if for any u, v and $\lambda \in [0, 1]$, we have $f(\lambda u + (1 - \lambda)v) \geq \lambda f(u) + (1 - \lambda)f(v)$.
- Jenson's is an immediate generalization. Proof uses induction on k.

Jensen's Inequality

- A function f is convex if for any u, v and $\lambda \in [0, 1]$, we have $f(\lambda u + (1 - \lambda)v) \leq \lambda f(u) + (1 - \lambda)f(v)$.
- A function f is concave if for any u, v and $\lambda \in [0, 1]$, we have $f(\lambda u + (1 - \lambda)v) \geq \lambda f(u) + (1 - \lambda)f(v)$.
- Jenson's is an immediate generalization. Proof uses induction on k.

Lemma 10.3.3 (Jenson's Inequality)

Let f be a convex function, $p = (p_1, p_2, \ldots, p_k)$ a discrete probability distribution, and $u_1, u_2, \ldots, u_k$ be k points. Then

$$f(E_p[U]) = f\left(\sum_{j=1}^{k} p_k u_k\right) \leq E_p[f(U)] = \sum_{j=1}^{k} p_k f(u_k). \tag{10.15}$$

If f is strictly convex, then equality implies $U = E_p[U]$.

Jensen's Inequality

- A function f is convex if for any u, v and $\lambda \in [0, 1]$, we have $f(\lambda u + (1 - \lambda)v) \leq \lambda f(u) + (1 - \lambda)f(v)$.
- A function f is concave if for any u, v and $\lambda \in [0, 1]$, we have $f(\lambda u + (1 - \lambda)v) \geq \lambda f(u) + (1 - \lambda)f(v)$.
- Jenson's is an immediate generalization. Proof uses induction on k.

Lemma 10.3.3 (Jenson's Inequality)

Let f be a convex function, $p = (p_1, p_2, \ldots, p_k)$ a discrete probability distribution, and $u_1, u_2, \ldots, u_k$ be k points. Then

$$f(E_p[U]) = f\left(\sum_{j=1}^{k} p_k u_k\right) \leq E_p[f(U)] = \sum_{j=1}^{k} p_k f(u_k). \tag{10.15}$$

If f is strictly convex, then equality implies $U = E_p[U]$.

- An analogous (reverse inequality) form holds when f is a concave function, such as $f(u) = \log(u)$ or $f(u) = \sqrt{u}$.

EM Algorithm: Q function as lower bound of $\ell(x;\theta)$

- The Q function is a lower bound of the true objective (incomplete log likelihood). This follows since:

$$\ell(x;\theta) = \log \sum_y p(x,y|\theta) = \log \sum_y p(y|x,\theta^g) \frac{p(x,y|\theta)}{p(y|x,\theta^g)} \tag{10.16}$$

$$\geq \sum_y p(y|x,\theta^g) \log \frac{p(x,y|\theta)}{p(y|x,\theta^g)} \tag{10.17}$$

$$= \sum_y p(y|x,\theta^g) \log p(x,y|\theta) - \sum_y p(y|x,\theta^g) \log p(y|x,\theta^g)$$

$$= Q(\theta,\theta^g) + H(\theta^g) \tag{10.18}$$

EM Algorithm: Q function as lower bound of $\ell(x;\theta)$

- The Q function is a lower bound of the true objective (incomplete log likelihood). This follows since:

$$\ell(x;\theta) = \log \sum_y p(x,y|\theta) = \log \sum_y p(y|x,\theta^g) \frac{p(x,y|\theta)}{p(y|x,\theta^g)} \tag{10.16}$$

$$\geq \sum_y p(y|x,\theta^g) \log \frac{p(x,y|\theta)}{p(y|x,\theta^g)} \tag{10.17}$$

$$= \sum_y p(y|x,\theta^g) \log p(x,y|\theta) - \sum_y p(y|x,\theta^g) \log p(y|x,\theta^g)$$

$$= Q(\theta,\theta^g) + H(\theta^g) \tag{10.18}$$

- When $\theta = \theta^g$ then $Q(\theta^g,\theta^g) + H(\theta^g) = \ell(x;\theta)$ (i.e., inequality tight if $\theta = \theta^g$).

EM Algorithm: Q function as lower bound of $\ell(x;\theta)$

- The Q function is a lower bound of the true objective (incomplete log likelihood). This follows since:

$$\ell(x;\theta) = \log \sum_y p(x,y|\theta) = \log \sum_y p(y|x,\theta^g) \frac{p(x,y|\theta)}{p(y|x,\theta^g)} \tag{10.16}$$

$$\geq \sum_y p(y|x,\theta^g) \log \frac{p(x,y|\theta)}{p(y|x,\theta^g)} \tag{10.17}$$

$$= \sum_y p(y|x,\theta^g) \log p(x,y|\theta) - \sum_y p(y|x,\theta^g) \log p(y|x,\theta^g)$$

$$= Q(\theta,\theta^g) + H(\theta^g) \tag{10.18}$$

- When $\theta = \theta^g$ then $Q(\theta^g,\theta^g) + H(\theta^g) = \ell(x;\theta)$ (i.e., inequality tight if $\theta = \theta^g$). For all other θ, we have a lower bound.

EM Algorithm: Q function as lower bound of $\ell(x;\theta)$

- The Q function is a lower bound of the true objective (incomplete log likelihood). This follows since:

$$\ell(x;\theta) = \log \sum_y p(x,y|\theta) = \log \sum_y p(y|x,\theta^g) \frac{p(x,y|\theta)}{p(y|x,\theta^g)} \tag{10.16}$$

$$\geq \sum_y p(y|x,\theta^g) \log \frac{p(x,y|\theta)}{p(y|x,\theta^g)} \tag{10.17}$$

$$= \sum_y p(y|x,\theta^g) \log p(x,y|\theta) - \sum_y p(y|x,\theta^g) \log p(y|x,\theta^g)$$

$$= Q(\theta,\theta^g) + H(\theta^g) \tag{10.18}$$

- When $\theta = \theta^g$ then $Q(\theta^g,\theta^g) + H(\theta^g) = \ell(x;\theta)$ (i.e., inequality tight if $\theta = \theta^g$). For all other θ, we have a lower bound. $H(\theta^g)$ is constant w.r.t. θ, only need to optimize Q.

EM Algorithm: Q function as lower bound of $\ell(x;\theta)$

- The Q function is a lower bound of the true objective (incomplete log likelihood). This follows since:

$$\ell(x;\theta) = \log \sum_y p(x,y|\theta) = \log \sum_y p(y|x,\theta^g) \frac{p(x,y|\theta)}{p(y|x,\theta^g)} \tag{10.16}$$

$$\geq \sum_y p(y|x,\theta^g) \log \frac{p(x,y|\theta)}{p(y|x,\theta^g)} \tag{10.17}$$

$$= \sum_y p(y|x,\theta^g) \log p(x,y|\theta) - \sum_y p(y|x,\theta^g) \log p(y|x,\theta^g)$$

$$= Q(\theta,\theta^g) + H(\theta^g) \tag{10.18}$$

- When $\theta = \theta^g$ then $Q(\theta^g,\theta^g) + H(\theta^g) = \ell(x;\theta)$ (i.e., inequality tight if $\theta = \theta^g$). For all other θ, we have a lower bound. $H(\theta^g)$ is constant w.r.t. θ, only need to optimize Q.
- Hence if we find a θ' that maximizes, so that $Q(\theta',\theta^g) > Q(\theta^g,\theta^g)$, we can only be increasing $\ell(x;\theta)$ (or at least not decrease it).

EM Algorithm: Q function as lower bound of $\ell(x;\theta)$

- The Q function is a lower bound of the true objective (incomplete log likelihood). This follows since:

$$\ell(x;\theta) = \log \sum_y p(x,y|\theta) = \log \sum_y p(y|x,\theta^g) \frac{p(x,y|\theta)}{p(y|x,\theta^g)} \tag{10.16}$$

$$\geq \sum_y p(y|x,\theta^g) \log \frac{p(x,y|\theta)}{p(y|x,\theta^g)} \tag{10.17}$$

$$= \sum_y p(y|x,\theta^g) \log p(x,y|\theta) - \sum_y p(y|x,\theta^g) \log p(y|x,\theta^g)$$

$$= Q(\theta,\theta^g) + H(\theta^g) \tag{10.18}$$

- When $\theta = \theta^g$ then $Q(\theta^g,\theta^g) + H(\theta^g) = \ell(x;\theta)$ (i.e., inequality tight if $\theta = \theta^g$). For all other θ, we have a lower bound. $H(\theta^g)$ is constant w.r.t. θ, only need to optimize Q.
- Hence if we find a θ' that maximizes, so that $Q(\theta',\theta^g) > Q(\theta^g,\theta^g)$, we can only be increasing $\ell(x;\theta)$ (or at least not decrease it).
- Often, $Q(\theta,\theta^g)$ is concave in θ, can easily find max, often in closed form.

Why tight when $\theta = \theta^g$?

$$\ell(x;\theta^g) = \log \sum_y p(x,y|\theta^g) \tag{10.19}$$

$$= [\log p(x|\theta^g)] \times 1 \tag{10.20}$$

$$= \log p(x|\theta^g) \sum_y p(y|x,\theta^g) \tag{10.21}$$

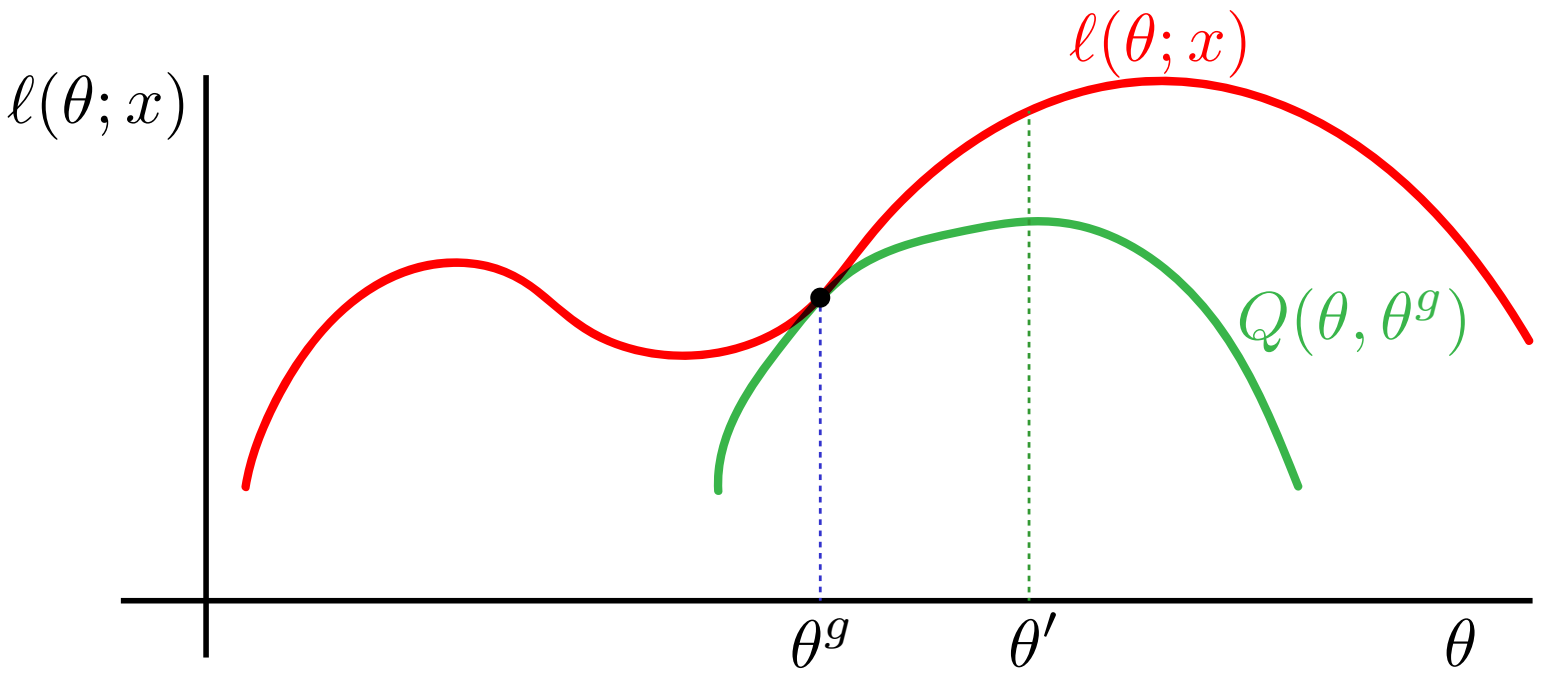
$$= \sum_y p(y|x,\theta^g) \log p(x|\theta^g) \tag{10.22}$$

$$= \sum_y p(y|x,\theta^g) \log \frac{p(y|x,\theta^g)p(x|\theta^g)}{p(y|x,\theta^g)} \tag{10.23}$$

$$= \sum_y p(y|x,\theta^g) \log \frac{p(x,y|\theta^g)}{p(y|x,\theta^g)} \tag{10.24}$$

$$= \sum_y p(y|x,\theta^g) \log p(x,y|\theta^g) - \sum_y p(y|x,\theta^g) \log p(y|x,\theta^g) \tag{10.25}$$

EM Aux. function $Q(\theta, \theta^g)$, concave lower bound of $\ell(x; \theta)$



- EM auxiliary function $Q(\theta, \theta^g)$, when seen as a function of θ for fixed θ^g is a concave lower bound to the likelihood $\ell(x; \theta)$ and so $Q(\theta, \theta^g)$ can be maximized, in one step of the EM algorithm, as a surrogate to maximizing $\ell(x; \theta)$ (the latter of which is harder). In other words, since $Q(\theta', \theta^g) > Q(\theta^g, \theta^g) = \ell(\theta^g; x)$, we have that $\ell(\theta'; x) \geq \ell(\theta^g; x) = Q(\theta^g, \theta^g)$.

Irregular Clusters

- Most of the clustering procedures we've seen are centroid based, where each cluster is based on nearness to some representative centroid (e.g., k-means, k-medians).

Irregular Clusters

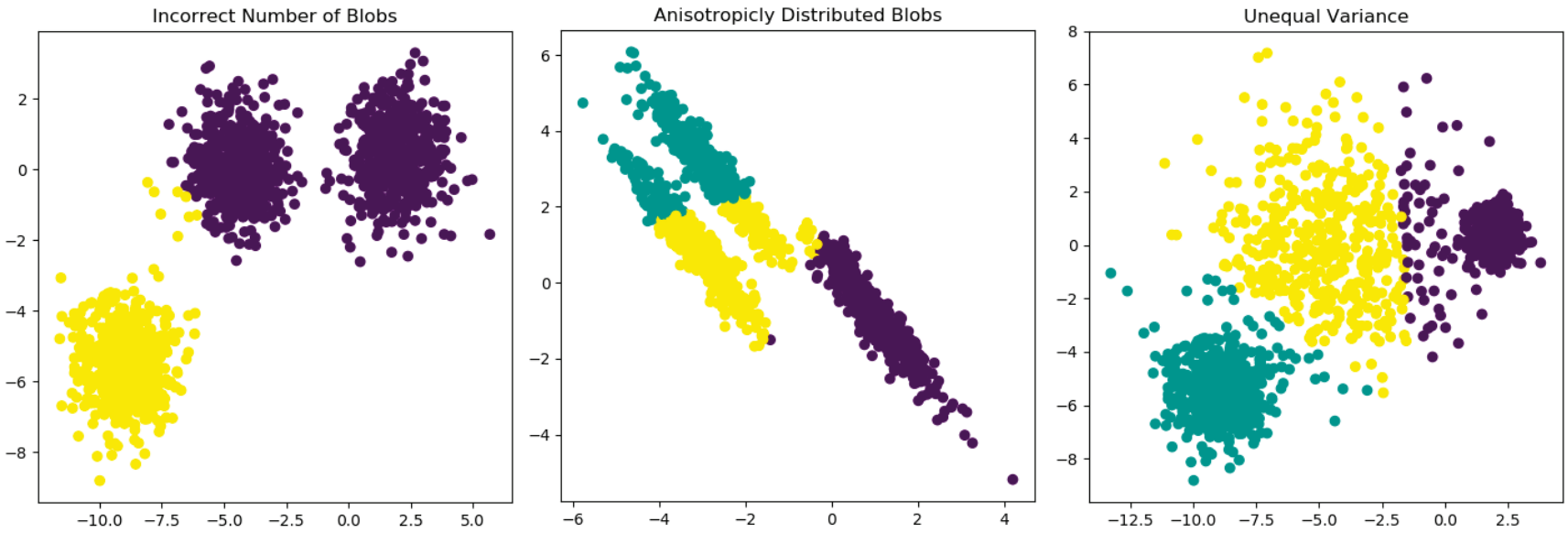
- Most of the clustering procedures we've seen are centroid based, where each cluster is based on nearness to some representative centroid (e.g., k-means, k-medians).
- Other clustering methods are high-density based, where clusters consist of potentially quite irregularly and non-convex shaped regions of high density separated by much more sparse or low-density regions.

Irregular Clusters

- Most of the clustering procedures we've seen are centroid based, where each cluster is based on nearness to some representative centroid (e.g., k-means, k-medians).
- Other clustering methods are high-density based, where clusters consist of potentially quite irregularly and non-convex shaped regions of high density separated by much more sparse or low-density regions.
- Example: separating islands from ocean, Galapagos (Ecuador)

Problems with k-means

- Sometimes k-means can fail.

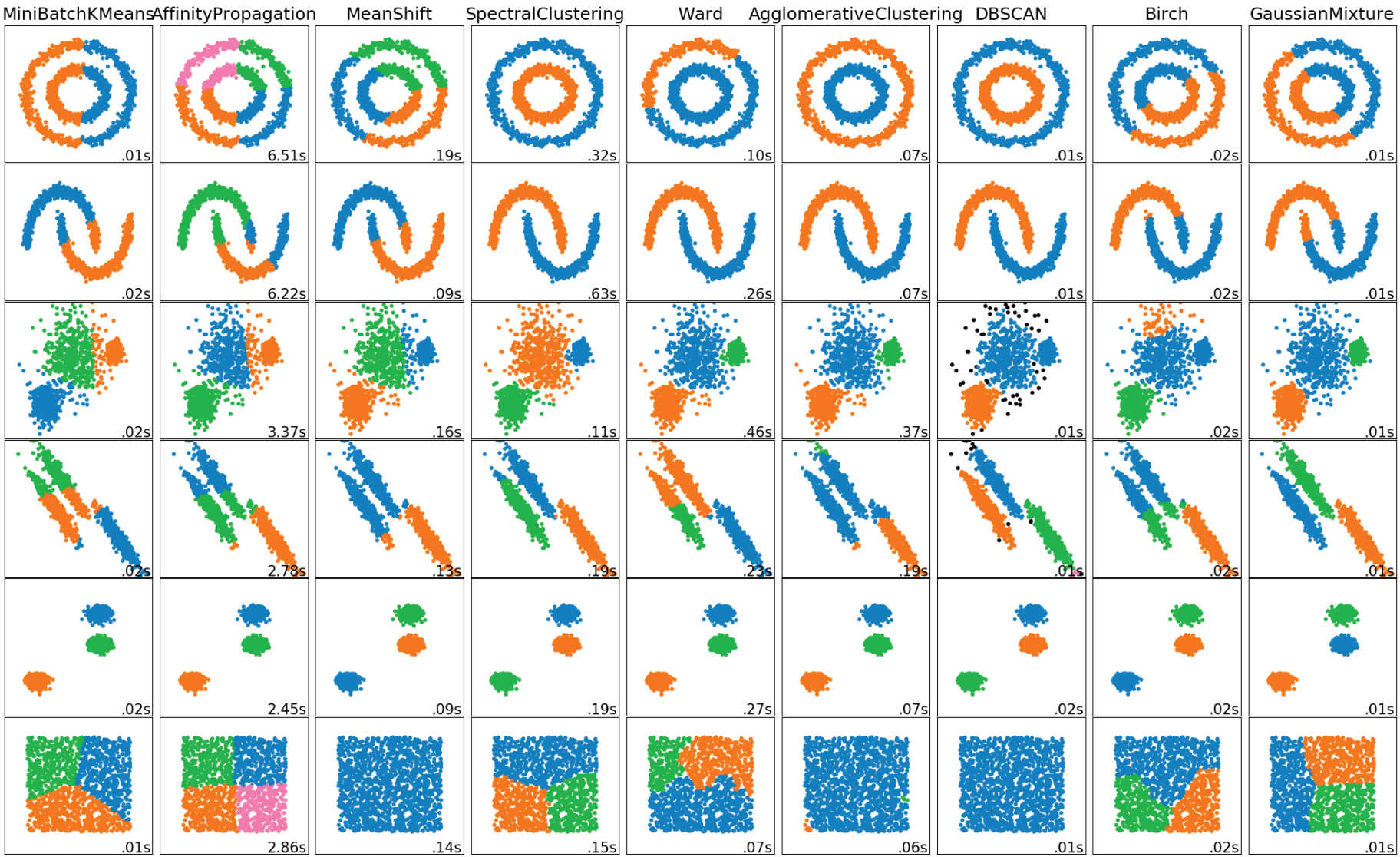


https://scikit-learn.org/stable/auto_examples/cluster/plot_kmeans_assumptions.html#sphx-glr-auto-examples-cluster-plot-kmeans-assumptions-py

- Left: can correct for this (e.g., mean shift clustering, see next slide frame, among others). Second two also also possible to (relatively) easily correct for (EM algorithm).

A Carnival of Clustering Cases

https://scikit-learn.org/stable/auto_examples/cluster/plot_cluster_comparison.html#sphx-glr-auto-examples-cluster-plot-cluster-comparison-py



Spectral Clustering

- One of the methods that works on almost all of the examples, as it allows for irregular cluster shapes.

Spectral Clustering

- One of the methods that works on almost all of the examples, as it allows for irregular cluster shapes.
- Related to graphs, graph manifolds, and forms of normalized graph cuts.

Spectral Clustering

- One of the methods that works on almost all of the examples, as it allows for irregular cluster shapes.
- Related to graphs, graph manifolds, and forms of normalized graph cuts.
- Key idea: given data $\mathcal{D} = \left\{x^{(i)}\right\}_{i=1}^{n}$ with $x^{(i)} \in \mathbb{R}^m$, change representation to new data set $\left\{c^{(i)}\right\}_{i=1}^{n}$ where $c^{(i)} \in \mathbb{R}^k$ based on properties of the graph Laplacian (TBD). This representation change typically makes cluster properties in the data much more salient and hence much easier to cluster using even, say, k-means.

Spectral Clustering

- One of the methods that works on almost all of the examples, as it allows for irregular cluster shapes.
- Related to graphs, graph manifolds, and forms of normalized graph cuts.
- Key idea: given data $\mathcal{D} = \left\{x^{(i)}\right\}_{i=1}^{n}$ with $x^{(i)} \in \mathbb{R}^m$, change representation to new data set $\left\{c^{(i)}\right\}_{i=1}^{n}$ where $c^{(i)} \in \mathbb{R}^k$ based on properties of the graph Laplacian (TBD). This representation change typically makes cluster properties in the data much more salient and hence much easier to cluster using even, say, k-means.
- Relatively simple to code using standard linear algebra techniques.

Spectral Clustering

- One of the methods that works on almost all of the examples, as it allows for irregular cluster shapes.
- Related to graphs, graph manifolds, and forms of normalized graph cuts.
- Key idea: given data $\mathcal{D} = \left\{x^{(i)}\right\}_{i=1}^{n}$ with $x^{(i)} \in \mathbb{R}^m$, change representation to new data set $\left\{c^{(i)}\right\}_{i=1}^{n}$ where $c^{(i)} \in \mathbb{R}^k$ based on properties of the graph Laplacian (TBD). This representation change typically makes cluster properties in the data much more salient and hence much easier to cluster using even, say, k-means.
- Relatively simple to code using standard linear algebra techniques.
- Is an important part of your HW5!

Spectral Clustering

- One of the methods that works on almost all of the examples, as it allows for irregular cluster shapes.
- Related to graphs, graph manifolds, and forms of normalized graph cuts.
- Key idea: given data $\mathcal{D} = \left\{x^{(i)}\right\}_{i=1}^{n}$ with $x^{(i)} \in \mathbb{R}^m$, change representation to new data set $\left\{c^{(i)}\right\}_{i=1}^{n}$ where $c^{(i)} \in \mathbb{R}^k$ based on properties of the graph Laplacian (TBD). This representation change typically makes cluster properties in the data much more salient and hence much easier to cluster using even, say, k-means.
- Relatively simple to code using standard linear algebra techniques.
- Is an important part of your HW5!
- We'll be following the Luxburg-2006 tutorial on this material. "A Tutorial on Spectral Clustering", Ulrike von Luxburg, https://arxiv.org/abs/0711.0189.

Graphs, Similarity, and Some Notation

- Data set $\mathcal{D} = \left\{x^{(i)}\right\}_{i=1}^{n}$ and a set of similarity values for every pair. I.e., for $i, j \in [n]$, $s_{i,j} \geq 0$ is how similar sample i and j are.

Graphs, Similarity, and Some Notation

- Data set $\mathcal{D} = \left\{x^{(i)}\right\}_{i=1}^{n}$ and a set of similarity values for every pair. I.e., for $i, j \in [n]$, $s_{i,j} \geq 0$ is how similar sample i and j are. E.g., $0 \leq s_{i,j} = \exp(-\|x^{(i)} - x^{(j)}\|/(2\sigma^2)) \leq 1$, exponential similarity kernel

Graphs, Similarity, and Some Notation

- Data set $\mathcal{D} = \left\{x^{(i)}\right\}_{i=1}^{n}$ and a set of similarity values for every pair. I.e., for $i, j \in [n]$, $s_{i,j} \geq 0$ is how similar sample i and j are. E.g., $0 \leq s_{i,j} = \exp(-\|x^{(i)} - x^{(j)}\|/(2\sigma^2)) \leq 1$, exponential similarity kernel
- Graph $G = (V, E, w)$ where $|V| = n$ and $V = \{v_1, \ldots, v_n\}$ represent the samples, and $w : E \to \mathbb{R}_+$ is weight function on the edges. I.e., for $e \in E$, $w(e) \in \mathbb{R}_+$. $w(u, v) = 0$ means u is not connected to v.

Graphs, Similarity, and Some Notation

- Data set $\mathcal{D} = \left\{x^{(i)}\right\}_{i=1}^{n}$ and a set of similarity values for every pair. I.e., for $i, j \in [n]$, $s_{i,j} \geq 0$ is how similar sample i and j are. E.g., $0 \leq s_{i,j} = \exp(-\|x^{(i)} - x^{(j)}\|/(2\sigma^2)) \leq 1$, exponential similarity kernel
- Graph $G = (V, E, w)$ where $|V| = n$ and $V = \{v_1, \ldots, v_n\}$ represent the samples, and $w : E \to \mathbb{R}_+$ is weight function on the edges. I.e., for $e \in E$, $w(e) \in \mathbb{R}_+$. $w(u, v) = 0$ means u is not connected to v.
- Adjacency matrix $W = (w_{ij})_{i,j \in [n]}$ is a matrix representation of the weights. Normally we set $w_{ij} = s_{ij}$, edges are weighted by similarity.

Graphs, Similarity, and Some Notation

- Data set $\mathcal{D} = \left\{x^{(i)}\right\}_{i=1}^{n}$ and a set of similarity values for every pair. I.e., for $i, j \in [n]$, $s_{i,j} \geq 0$ is how similar sample i and j are. E.g., $0 \leq s_{i,j} = \exp(-\|x^{(i)} - x^{(j)}\|/(2\sigma^2)) \leq 1$, exponential similarity kernel
- Graph $G = (V, E, w)$ where $|V| = n$ and $V = \{v_1, \ldots, v_n\}$ represent the samples, and $w : E \to \mathbb{R}_+$ is weight function on the edges. I.e., for $e \in E$, $w(e) \in \mathbb{R}_+$. $w(u, v) = 0$ means u is not connected to v.
- Adjacency matrix $W = (w_{ij})_{i,j \in [n]}$ is a matrix representation of the weights. Normally we set $w_{ij} = s_{ij}$, edges are weighted by similarity.
  - $e = (i, j) \in E(G)$ iff $w_{i,j} > 0$

Graphs, Similarity, and Some Notation

- Data set $\mathcal{D} = \left\{x^{(i)}\right\}_{i=1}^{n}$ and a set of similarity values for every pair. I.e., for $i, j \in [n]$, $s_{i,j} \geq 0$ is how similar sample i and j are. E.g., $0 \leq s_{i,j} = \exp(-\|x^{(i)} - x^{(j)}\|/(2\sigma^2)) \leq 1$, exponential similarity kernel
- Graph $G = (V, E, w)$ where $|V| = n$ and $V = \{v_1, \ldots, v_n\}$ represent the samples, and $w : E \to \mathbb{R}_+$ is weight function on the edges. I.e., for $e \in E$, $w(e) \in \mathbb{R}_+$. $w(u, v) = 0$ means u is not connected to v.
- Adjacency matrix $W = (w_{ij})_{i,j \in [n]}$ is a matrix representation of the weights. Normally we set $w_{ij} = s_{ij}$, edges are weighted by similarity.
  - $e = (i, j) \in E(G)$ iff $w_{i,j} > 0$
  - $e = (i, j) \notin E(G)$ iff $w_{i,j} = 0$

Graphs, Similarity, and Some Notation

- Data set $\mathcal{D} = \left\{x^{(i)}\right\}_{i=1}^{n}$ and a set of similarity values for every pair. I.e., for $i, j \in [n]$, $s_{i,j} \geq 0$ is how similar sample i and j are. E.g., $0 \leq s_{i,j} = \exp(-\|x^{(i)} - x^{(j)}\|/(2\sigma^2)) \leq 1$, exponential similarity kernel
- Graph $G = (V, E, w)$ where $|V| = n$ and $V = \{v_1, \ldots, v_n\}$ represent the samples, and $w : E \to \mathbb{R}_+$ is weight function on the edges. I.e., for $e \in E$, $w(e) \in \mathbb{R}_+$. $w(u, v) = 0$ means u is not connected to v.
- Adjacency matrix $W = (w_{ij})_{i,j \in [n]}$ is a matrix representation of the weights. Normally we set $w_{ij} = s_{ij}$, edges are weighted by similarity.
  - $e = (i, j) \in E(G)$ iff $w_{i,j} > 0$
  - $e = (i, j) \notin E(G)$ iff $w_{i,j} = 0$
- Degree of a vertex $d_i = \sum_{j=1}^{n} w_{ij}$, and Degree matrix $D = \text{diag}(d_1, d_2, \ldots, d_n)$.

Graphs, Similarity, and Some Notation

- Data set $\mathcal{D} = \left\{x^{(i)}\right\}_{i=1}^{n}$ and a set of similarity values for every pair. I.e., for $i, j \in [n]$, $s_{i,j} \geq 0$ is how similar sample i and j are. E.g., $0 \leq s_{i,j} = \exp(-\|x^{(i)} - x^{(j)}\|/(2\sigma^2)) \leq 1$, exponential similarity kernel
- Graph $G = (V, E, w)$ where $|V| = n$ and $V = \{v_1, \ldots, v_n\}$ represent the samples, and $w : E \to \mathbb{R}_+$ is weight function on the edges. I.e., for $e \in E$, $w(e) \in \mathbb{R}_+$. $w(u, v) = 0$ means u is not connected to v.
- Adjacency matrix $W = (w_{ij})_{i,j \in [n]}$ is a matrix representation of the weights. Normally we set $w_{ij} = s_{ij}$, edges are weighted by similarity.
  - $e = (i, j) \in E(G)$ iff $w_{i,j} > 0$
  - $e = (i, j) \notin E(G)$ iff $w_{i,j} = 0$
- Degree of a vertex $d_i = \sum_{j=1}^{n} w_{ij}$, and Degree matrix $D = \text{diag}(d_1, d_2, \ldots, d_n)$. Is weighted degree (although for 0/1-valued weights, counts the number of edges incident to v_i).

Graphs, Similarity, and Some Notation

- Data set $\mathcal{D} = \left\{x^{(i)}\right\}_{i=1}^{n}$ and a set of similarity values for every pair. I.e., for $i, j \in [n]$, $s_{i,j} \geq 0$ is how similar sample i and j are. E.g., $0 \leq s_{i,j} = \exp(-\|x^{(i)} - x^{(j)}\|/(2\sigma^2)) \leq 1$, exponential similarity kernel
- Graph $G = (V, E, w)$ where $|V| = n$ and $V = \{v_1, \ldots, v_n\}$ represent the samples, and $w : E \to \mathbb{R}_+$ is weight function on the edges. I.e., for $e \in E$, $w(e) \in \mathbb{R}_+$. $w(u, v) = 0$ means u is not connected to v.
- Adjacency matrix $W = (w_{ij})_{i,j \in [n]}$ is a matrix representation of the weights. Normally we set $w_{ij} = s_{ij}$, edges are weighted by similarity.
  - $e = (i, j) \in E(G)$ iff $w_{i,j} > 0$
  - $e = (i, j) \notin E(G)$ iff $w_{i,j} = 0$
- Degree of a vertex $d_i = \sum_{j=1}^{n} w_{ij}$, and Degree matrix $D = \text{diag}(d_1, d_2, \ldots, d_n)$. Is weighted degree (although for 0/1-valued weights, counts the number of edges incident to v_i).
- Set $A \subseteq V$ and its complement $\bar{A} = V \setminus A$.

Graphs, Similarity, and Some Notation

- Data set $\mathcal{D} = \left\{x^{(i)}\right\}_{i=1}^{n}$ and a set of similarity values for every pair. I.e., for $i, j \in [n]$, $s_{i,j} \geq 0$ is how similar sample i and j are. E.g., $0 \leq s_{i,j} = \exp(-\|x^{(i)} - x^{(j)}\|/(2\sigma^2)) \leq 1$, exponential similarity kernel
- Graph $G = (V, E, w)$ where $|V| = n$ and $V = \{v_1, \ldots, v_n\}$ represent the samples, and $w : E \to \mathbb{R}_+$ is weight function on the edges. I.e., for $e \in E$, $w(e) \in \mathbb{R}_+$. $w(u, v) = 0$ means u is not connected to v.
- Adjacency matrix $W = (w_{ij})_{i,j \in [n]}$ is a matrix representation of the weights. Normally we set $w_{ij} = s_{ij}$, edges are weighted by similarity.
  - $e = (i, j) \in E(G)$ iff $w_{i,j} > 0$
  - $e = (i, j) \notin E(G)$ iff $w_{i,j} = 0$
- Degree of a vertex $d_i = \sum_{j=1}^{n} w_{ij}$, and Degree matrix $D = \text{diag}(d_1, d_2, \ldots, d_n)$. Is weighted degree (although for 0/1-valued weights, counts the number of edges incident to v_i).
- Set $A \subseteq V$ and its complement $\bar{A} = V \setminus A$.
- Characteristic vector of a set A is $\mathbb{1}_A \in \mathbb{R}_+^n$, has for $v \in V$,

$$\mathbb{1}_A(v) = \begin{cases} 1 & \text{if } v \in A \\ 0 & \text{else} \end{cases}$$

Cuts, Size, and Volume

- The cut between $A, B \subseteq V$ is

$$W(A, B) = \sum_{i \in A, j \in B} w_{ij} \tag{10.27}$$

(note we synonymize nodes and their indices here).

Cuts, Size, and Volume

- The cut between $A, B \subseteq V$ is

$$W(A, B) = \sum_{i \in A, j \in B} w_{ij} \tag{10.27}$$

(note we synonymize nodes and their indices here).

- Cardinality $|A|$ is the number of elements in set A

Cuts, Size, and Volume

- The cut between $A, B \subseteq V$ is

$$W(A, B) = \sum_{i \in A, j \in B} w_{ij} \tag{10.27}$$

  (note we synonymize nodes and their indices here).
- Cardinality $|A|$ is the number of elements in set A
- Volume $\text{vol}(A) = \sum_{i \in A} d_i$, measures a form of overall connectivity strength of $A \subseteq V$.

Cuts, Size, and Volume

- The cut between $A, B \subseteq V$ is

$$W(A, B) = \sum_{i \in A, j \in B} w_{ij} \tag{10.27}$$

  (note we synonymize nodes and their indices here).
- Cardinality $|A|$ is the number of elements in set A
- Volume $\text{vol}(A) = \sum_{i \in A} d_i$, measures a form of overall connectivity strength of $A \subseteq V$.
- Connected component of a graph

Cuts, Size, and Volume

- The cut between $A, B \subseteq V$ is

$$W(A, B) = \sum_{i \in A, j \in B} w_{ij} \tag{10.27}$$

  (note we synonymize nodes and their indices here).
- Cardinality $|A|$ is the number of elements in set A
- Volume $\text{vol}(A) = \sum_{i \in A} d_i$, measures a form of overall connectivity strength of $A \subseteq V$.
- Connected component of a graph
- $A_1, \ldots, A_k$ is a partition of V if $\cup_i A_i = V$ and $A_i \cap A_j = \emptyset$ for $i \neq j$.

Similarity Graphs

- fully connected: E.g., $s_{ij} = \exp(-\|x^{(i)} - x^{(j)}\|/(2\sigma^2)) > 0$ so all edges are present. $G = (V, E)$ is a big clique with $n = |V|$ nodes.

Similarity Graphs

- fully connected: E.g., $s_{ij} = \exp(-\|x^{(i)} - x^{(j)}\|/(2\sigma^2)) > 0$ so all edges are present. $G = (V, E)$ is a big clique with $n = |V|$ nodes. σ is the width of the kernel, as σ gets big, edges between distant nodes have very small weight!

Similarity Graphs

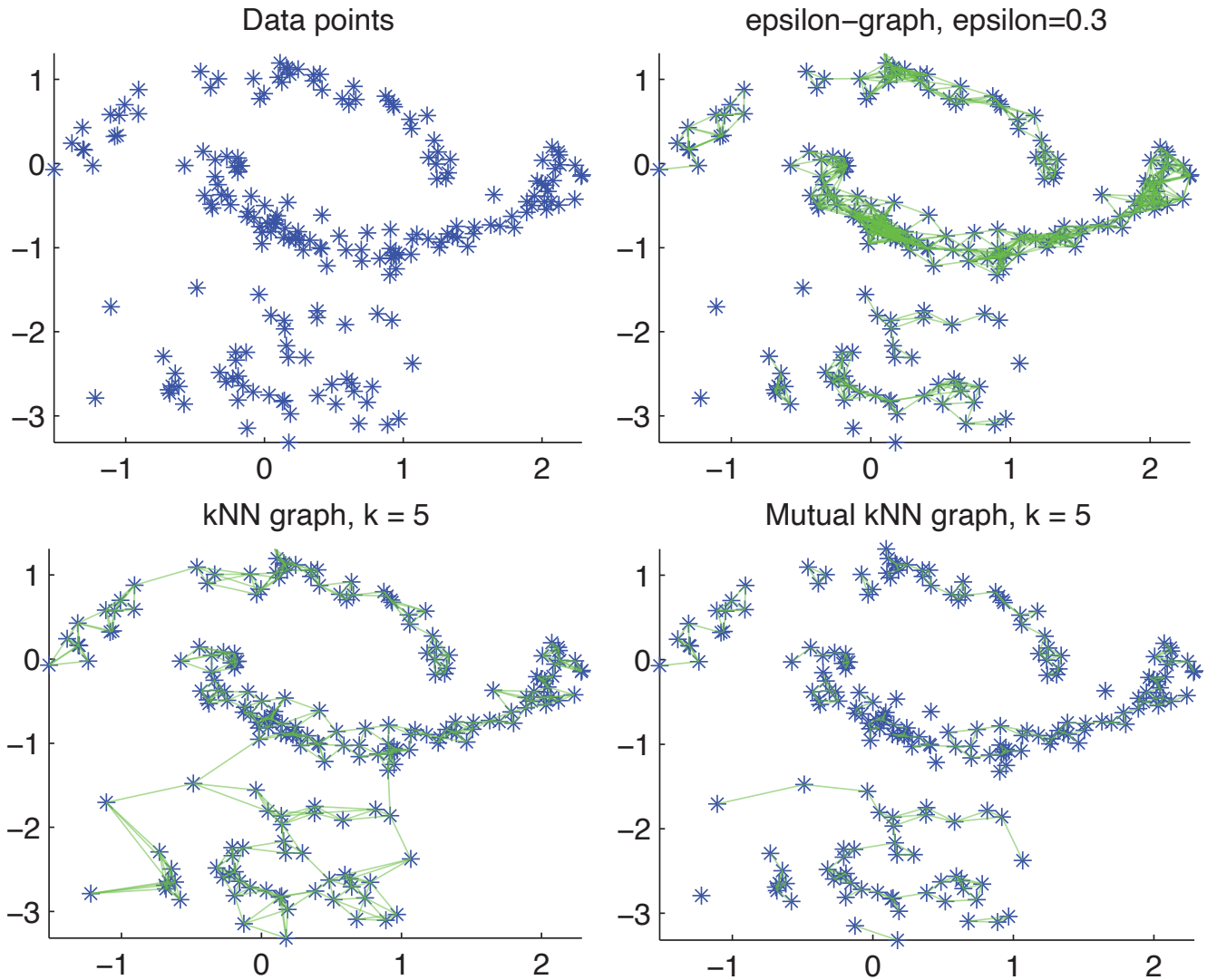
- fully connected: E.g., $s_{ij} = \exp(-\|x^{(i)} - x^{(j)}\|/(2\sigma^2)) > 0$ so all edges are present. $G = (V, E)$ is a big clique with $n = |V|$ nodes. σ is the width of the kernel, as σ gets big, edges between distant nodes have very small weight!
- k-NN graph. Each node is only connected to its k nearest neighbors. Two ways to make adjacency matrix symmetric: $W \leftarrow \max(W, W^\intercal)$ leading to k-nearest neighbor graph, or $W \leftarrow \min(W, W^\intercal)$ (the mutual k-nearest neighbor graph).

Similarity Graphs

- fully connected: E.g., $s_{ij} = \exp(-\|x^{(i)} - x^{(j)}\|/(2\sigma^2)) > 0$ so all edges are present. $G = (V, E)$ is a big clique with $n = |V|$ nodes. σ is the width of the kernel, as σ gets big, edges between distant nodes have very small weight!
- k-NN graph. Each node is only connected to its k nearest neighbors. Two ways to make adjacency matrix symmetric: $W \leftarrow \max(W, W^\intercal)$ leading to k-nearest neighbor graph, or $W \leftarrow \min(W, W^\intercal)$ (the mutual k-nearest neighbor graph).
- ϵ-neighborhood graph: connect points whose distance is $< \epsilon$. Often these are unweighted.

Similarity Graph Examples

- Which similarity graph chosen can make a big difference. Example, sample of two-moons with some additional Gaussian points below.



A few reminders

- Note that a vector $z \in R^n$ is an eigenvector of a matrix $L \in \mathbb{R}^{n \times n}$ if $Lz = \lambda z$ for some $\lambda \in R$.

A few reminders

- Note that a vector $z \in R^n$ is an eigenvector of a matrix $L \in \mathbb{R}^{n \times n}$ if $Lz = \lambda z$ for some $\lambda \in R$.
- If z is an eigenvector, than so is αz for any $\alpha \in \mathbb{R}$, $\alpha \neq 0$ with corresponding eigenvalue $\alpha\lambda$.

A few reminders

- Note that a vector $z \in R^n$ is an eigenvector of a matrix $L \in \mathbb{R}^{n \times n}$ if $Lz = \lambda z$ for some $\lambda \in R$.
- If z is an eigenvector, than so is αz for any $\alpha \in \mathbb{R}$, $\alpha \neq 0$ with corresponding eigenvalue $\alpha\lambda$.
- If z is an eigenvector for L with 0 eigenvalue, then so is αz for any α (non-trivial case if $\alpha \neq 0$).

A few reminders

- Note that a vector $z \in R^n$ is an eigenvector of a matrix $L \in \mathbb{R}^{n \times n}$ if $Lz = \lambda z$ for some $\lambda \in R$.
- If z is an eigenvector, than so is αz for any $\alpha \in \mathbb{R}$, $\alpha \neq 0$ with corresponding eigenvalue $\alpha\lambda$.
- If z is an eigenvector for L with 0 eigenvalue, then so is αz for any α (non-trivial case if $\alpha \neq 0$).
- If $z_1, z_2, \ldots, z_k \in \mathbb{R}^n$ are each eigenvectors for L and each with 0 eigenvalue (so $Lz_j = 0z_j = \vec{0}$, $j \in [k]$), then $\sum_{j \in S} z_j$ is also an eigenvector for L with 0 eigenvalue, for any set $S \subseteq [k]$.

Graph Laplacian

- An amazing amount of information about a graph can be expressed by properties of its Laplacian matrix.

Graph Laplacian

- An amazing amount of information about a graph can be expressed by properties of its Laplacian matrix.
- Given degree matrix D (i.e., a diagonal matrix with entries $d_i = \sum_{j=1}^{n} w_{ij}$) and adjacency matrix W, define the unnormalized graph Laplacian matrix as

$$L = D - W \tag{10.28}$$

Graph Laplacian

- An amazing amount of information about a graph can be expressed by properties of its Laplacian matrix.
- Given degree matrix D (i.e., a diagonal matrix with entries $d_i = \sum_{j=1}^{n} w_{ij}$) and adjacency matrix W, define the unnormalized graph Laplacian matrix as

$$L = D - W \tag{10.28}$$

- Note $L_{ii} = d_{ii} - w_{ii} = \sum_{j=1}^{n} w_{ij} - w_{ii} = \sum_{j \neq i} w_{ij}$, so diagonal of W is irrelevant to L (e.g., self-edges don't matter).

Properties of the graph Laplacian

Proposition 10.4.1 (Properties of the unnormalized graph Laplacian L)

1. *For every vector $z \in \mathbb{R}^n$, we have*

$$z^\intercal L z = \frac{1}{2} \sum_{i,j=1}^{n} w_{ij}(z_i - z_j)^2 \geq 0 \tag{10.29}$$

2. *L is symmetric and positive semi-definite.*
3. *The smallest eigenvalue of L is 0, and the corresponding eigenvector is the constant one vector $\mathbb{1} = \mathbb{1}_V \in \mathbb{R}^n$.*
4. *L has n non-negative real valued eigenvalues $0 = \lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n$.*

Properties of the graph Laplacian

Proposition 10.4.1 (Properties of the unnormalized graph Laplacian L)

1. *For every vector $z \in \mathbb{R}^n$, we have*

$$z^\intercal L z = \frac{1}{2}\sum_{i,j=1}^{n} w_{ij}(z_i - z_j)^2 \geq 0 \tag{10.29}$$

2. *L is symmetric and positive semi-definite.*
3. *The smallest eigenvalue of L is 0, and the corresponding eigenvector is the constant one vector $\mathbb{1} = \mathbb{1}_V \in \mathbb{R}^n$.*
4. *L has n non-negative real valued eigenvalues $0 = \lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n$.*

$$z^\intercal L z = z^\intercal D z - z^\intercal W z = \sum_{i=1}^{n} d_i z_i^2 - \sum_{i,j=1}^{n} z_i z_j w_{ij} \tag{10.30}$$

$$= \frac{1}{2}\left(\sum_{i=1}^{n} d_i z_i^2 - 2\sum_{i,j=1}^{n} z_i z_j w_{ij} + \sum_{j=1}^{n} d_j z_j^2\right) = \frac{1}{2}\sum_{ij=1}^{n} w_{ij}(z_i - z_j)^2$$

On Proposition

- L is symmetric since W and D are. L is positive semidefinite since r.h.s. expression is never negative.

On Proposition

- L is symmetric since W and D are. L is positive semidefinite since r.h.s. expression is never negative.
- That 0 is an eigenvalue follows since $Lx = 0x = 0$ is same as $Dx = Wx$; since $d_i = \sum_j w_{ij}$ this happens for $x = \alpha \mathbb{1}$, $\alpha \neq 0$.

On Proposition

- L is symmetric since W and D are. L is positive semidefinite since r.h.s. expression is never negative.
- That 0 is an eigenvalue follows since $Lx = 0x = 0$ is same as $Dx = Wx$; since $d_i = \sum_j w_{ij}$ this happens for $x = \alpha \mathbb{1}$, $\alpha \neq 0$.
- Positive semi-definite matrices are symmetric and have non-negative eigenvalues (defining property).

On Proposition

- L is symmetric since W and D are. L is positive semidefinite since r.h.s. expression is never negative.
- That 0 is an eigenvalue follows since $Lx = 0x = 0$ is same as $Dx = Wx$; since $d_i = \sum_j w_{ij}$ this happens for $x = \alpha \mathbb{1}$, $\alpha \neq 0$.
- Positive semi-definite matrices are symmetric and have non-negative eigenvalues (defining property).

Proposition 10.4.2 (Number of connected components of spectrum of L)

$G = (V, E, W)$ is undirected graph with non-negative weights. Multiplicity k of the 0 eigenvalues of $L = D - W$ of G equals the number of connected components $A_1, A_2, \ldots, A_k$ of G. The eigenspace of eigenvalue 0 is spanned by characteristic vectors $\mathbb{1}_{A_1}, \mathbb{1}_{A_2}, \ldots, \mathbb{1}_{A_k}$ of the connected components.

On Proposition

- L is symmetric since W and D are. L is positive semidefinite since r.h.s. expression is never negative.
- That 0 is an eigenvalue follows since $Lx = 0x = 0$ is same as $Dx = Wx$; since $d_i = \sum_j w_{ij}$ this happens for $x = \alpha \mathbb{1}$, $\alpha \neq 0$.
- Positive semi-definite matrices are symmetric and have non-negative eigenvalues (defining property).

Proposition 10.4.2 (Number of connected components of spectrum of L)

$G = (V, E, W)$ is undirected graph with non-negative weights. Multiplicity k of the 0 eigenvalues of $L = D - W$ of G equals the number of connected components $A_1, A_2, \ldots, A_k$ of G. The eigenspace of eigenvalue 0 is spanned by characteristic vectors $\mathbb{1}_{A_1}, \mathbb{1}_{A_2}, \ldots, \mathbb{1}_{A_k}$ of the connected components.

- Eigenspace of eigenvalue 0 is $\mathcal{N}(L) = \{x : Lx = 0\}$, i.e., null space.

On Proposition

- L is symmetric since W and D are. L is positive semidefinite since r.h.s. expression is never negative.
- That 0 is an eigenvalue follows since $Lx = 0x = 0$ is same as $Dx = Wx$; since $d_i = \sum_j w_{ij}$ this happens for $x = \alpha \mathbb{1}$, $\alpha \neq 0$.
- Positive semi-definite matrices are symmetric and have non-negative eigenvalues (defining property).

Proposition 10.4.2 (Number of connected components of spectrum of L)

$G = (V, E, W)$ is undirected graph with non-negative weights. Multiplicity k of the 0 eigenvalues of $L = D - W$ of G equals the number of connected components $A_1, A_2, \ldots, A_k$ of G. The eigenspace of eigenvalue 0 is spanned by characteristic vectors $\mathbb{1}_{A_1}, \mathbb{1}_{A_2}, \ldots, \mathbb{1}_{A_k}$ of the connected components.

- Eigenspace of eigenvalue 0 is $\mathcal{N}(L) = \{x : Lx = 0\}$, i.e., null space.
- Since $\{A_i\}_{i=1}^k$ partitions V, $\mathbb{1}_{A_1}, \mathbb{1}_{A_2}, \ldots, \mathbb{1}_{A_k}$ correspond to orthogonal basis vectors of the eigenspace of eigenvalue 0.

Proposition Proof and Intuition

- Let $z \in \mathbb{R}^n$ be an eigenvector of L with 0 eigenvalue (i.e., $Lz = 0$). Consider

$$0 = z^\intercal Lz = \sum_{i,j=1}^{n} w_{ij}(z_i - z_j)^2 \tag{10.31}$$

Proposition Proof and Intuition

- Let $z \in \mathbb{R}^n$ be an eigenvector of L with 0 eigenvalue (i.e., $Lz = 0$). Consider

$$0 = z^\intercal Lz = \sum_{i,j=1}^{n} w_{ij}(z_i - z_j)^2 \tag{10.31}$$

- If multiplicity is one ($k = 1$), already have $\mathbb{1}$ is an eigenvector with 0 eigenvalue.

Proposition Proof and Intuition

- Let $z \in \mathbb{R}^n$ be an eigenvector of L with 0 eigenvalue (i.e., $Lz = 0$). Consider

$$0 = z^\intercal Lz = \sum_{i,j=1}^{n} w_{ij}(z_i - z_j)^2 \tag{10.31}$$

- If multiplicity is one ($k = 1$), already have $\mathbb{1}$ is an eigenvector with 0 eigenvalue.
- For arbitrary $k > 0$, need $w_{ij}(z_i - z_j)^2 = 0$ for all i, j to achieve 0 since $w_{ij} \geq 0$ and $(z_i - z_j)^2 \geq 0$. Thus, must have either $w_{ij} = 0$ or $z_i = z_j$.

Proposition Proof and Intuition

- Let $z \in \mathbb{R}^n$ be an eigenvector of L with 0 eigenvalue (i.e., $Lz = 0$). Consider

$$0 = z^\intercal L z = \sum_{i,j=1}^{n} w_{ij}(z_i - z_j)^2 \tag{10.31}$$

- If multiplicity is one ($k = 1$), already have $\mathbb{1}$ is an eigenvector with 0 eigenvalue.
- For arbitrary $k > 0$, need $w_{ij}(z_i - z_j)^2 = 0$ for all i, j to achieve 0 since $w_{ij} \geq 0$ and $(z_i - z_j)^2 \geq 0$. Thus, must have either $w_{ij} = 0$ or $z_i = z_j$.
- Within connected component A, there is a path between any two nodes where $w_{ij} > 0$ along the path. Hence, within connected component, need $z_i = z_j$. Thus, $\alpha \mathbb{1}_A$ is an eigenvector with 0 eigenvalue.

Proposition Proof and Intuition

- Let $z \in \mathbb{R}^n$ be an eigenvector of L with 0 eigenvalue (i.e., $Lz = 0$). Consider

$$0 = z^\intercal L z = \sum_{i,j=1}^{n} w_{ij}(z_i - z_j)^2 \tag{10.31}$$

- If multiplicity is one ($k = 1$), already have $\mathbb{1}$ is an eigenvector with 0 eigenvalue.
- For arbitrary $k > 0$, need $w_{ij}(z_i - z_j)^2 = 0$ for all i, j to achieve 0 since $w_{ij} \geq 0$ and $(z_i - z_j)^2 \geq 0$. Thus, must have either $w_{ij} = 0$ or $z_i = z_j$.
- Within connected component A, there is a path between any two nodes where $w_{ij} > 0$ along the path. Hence, within connected component, need $z_i = z_j$. Thus, $\alpha \mathbb{1}_A$ is an eigenvector with 0 eigenvalue.
- Thus $\mathbb{1}_{A_j}$ for $j \in [k]$ are a set of k distinct eigenvectors.

Spectral Clustering Intuition

- The vectors $c_i = (\mathbb{1}_{A_1}(i), \mathbb{1}_{A_2}(i), \ldots, \mathbb{1}_{A_k}(i)) \in R^k$ for $i \in [n]$ are one-hot encodings for component of graph G that vector $x^{(i)}$ is in.

Spectral Clustering Intuition

- The vectors $c_i = (\mathbb{1}_{A_1}(i), \mathbb{1}_{A_2}(i), \ldots, \mathbb{1}_{A_k}(i)) \in R^k$ for $i \in [n]$ are one-hot encodings for component of graph G that vector $x^{(i)}$ is in. k-means would cluster vectors $\{c_i\}_{i=1}^n$ very easily!

Spectral Clustering Intuition

- The vectors $c_i = (\mathbb{1}_{A_1}(i), \mathbb{1}_{A_2}(i), \ldots, \mathbb{1}_{A_k}(i)) \in R^k$ for $i \in [n]$ are one-hot encodings for component of graph G that vector $x^{(i)}$ is in. k-means would cluster vectors $\{c_i\}_{i=1}^n$ very easily!
- $A_1, \ldots, A_k$ is a partition V, then eigenvectors are orthogonal since $\langle \mathbb{1}_{A_i}, \mathbb{1}_{A_j} \rangle = |A_i| \delta(i = j)$.

Spectral Clustering Intuition

- The vectors $c_i = (\mathbb{1}_{A_1}(i), \mathbb{1}_{A_2}(i), \ldots, \mathbb{1}_{A_k}(i)) \in R^k$ for $i \in [n]$ are one-hot encodings for component of graph G that vector $x^{(i)}$ is in. k-means would cluster vectors $\{c_i\}_{i=1}^n$ very easily!
- $A_1, \ldots, A_k$ is a partition V, then eigenvectors are orthogonal since $\langle \mathbb{1}_{A_i}, \mathbb{1}_{A_j} \rangle = |A_i| \delta(i = j)$. Other linearly independent combinations possible too, i.e., $\mathbb{1}_{A_1 \cup A_2} = \mathbb{1}_{A_1} + \mathbb{1}_{A_2}$ and $\mathbb{1}_{A_2}$ are independent but not orthogonal. Independent non-orthogonal is fine (see figure below).

Spectral Clustering Intuition

- The vectors $c_i = (\mathbb{1}_{A_1}(i), \mathbb{1}_{A_2}(i), \ldots, \mathbb{1}_{A_k}(i)) \in R^k$ for $i \in [n]$ are one-hot encodings for component of graph G that vector $x^{(i)}$ is in. k-means would cluster vectors $\{c_i\}_{i=1}^n$ very easily!
- $A_1, \ldots, A_k$ is a partition V, then eigenvectors are orthogonal since $\langle \mathbb{1}_{A_i}, \mathbb{1}_{A_j} \rangle = |A_i| \delta(i = j)$. Other linearly independent combinations possible too, i.e., $\mathbb{1}_{A_1 \cup A_2} = \mathbb{1}_{A_1} + \mathbb{1}_{A_2}$ and $\mathbb{1}_{A_2}$ are independent but not orthogonal. Independent non-orthogonal is fine (see figure below).
- Even if graph is connected, let $0 = \lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n$ be the set of eigenvalues of L sorted ascending. Consider first k corresponding eigenvectors $u_1, u_2, \ldots, u_k$ of L.

Spectral Clustering Intuition

- The vectors $c_i = (\mathbb{1}_{A_1}(i), \mathbb{1}_{A_2}(i), \ldots, \mathbb{1}_{A_k}(i)) \in R^k$ for $i \in [n]$ are one-hot encodings for component of graph G that vector $x^{(i)}$ is in. k-means would cluster vectors $\{c_i\}_{i=1}^n$ very easily!
- $A_1, \ldots, A_k$ is a partition V, then eigenvectors are orthogonal since $\langle \mathbb{1}_{A_i}, \mathbb{1}_{A_j} \rangle = |A_i| \delta(i = j)$. Other linearly independent combinations possible too, i.e., $\mathbb{1}_{A_1 \cup A_2} = \mathbb{1}_{A_1} + \mathbb{1}_{A_2}$ and $\mathbb{1}_{A_2}$ are independent but not orthogonal. Independent non-orthogonal is fine (see figure below).
- Even if graph is connected, let $0 = \lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n$ be the set of eigenvalues of L sorted ascending. Consider first k corresponding eigenvectors $u_1, u_2, \ldots, u_k$ of L.
- Then $c_i = (u_1(i), u_2(i), \ldots, u_k(i)) \in \mathbb{R}^k$ is a soft one-hot encoding for the high density region of the graph vector $x^{(i)}$ is in.

Spectral Clustering Intuition

- The vectors $c_i = (\mathbb{1}_{A_1}(i), \mathbb{1}_{A_2}(i), \ldots, \mathbb{1}_{A_k}(i)) \in R^k$ for $i \in [n]$ are one-hot encodings for component of graph G that vector $x^{(i)}$ is in. k-means would cluster vectors $\{c_i\}_{i=1}^n$ very easily!
- $A_1, \ldots, A_k$ is a partition V, then eigenvectors are orthogonal since $\langle \mathbb{1}_{A_i}, \mathbb{1}_{A_j} \rangle = |A_i| \delta(i = j)$. Other linearly independent combinations possible too, i.e., $\mathbb{1}_{A_1 \cup A_2} = \mathbb{1}_{A_1} + \mathbb{1}_{A_2}$ and $\mathbb{1}_{A_2}$ are independent but not orthogonal. Independent non-orthogonal is fine (see figure below).
- Even if graph is connected, let $0 = \lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n$ be the set of eigenvalues of L sorted ascending. Consider first k corresponding eigenvectors $u_1, u_2, \ldots, u_k$ of L.
- Then $c_i = (u_1(i), u_2(i), \ldots, u_k(i)) \in \mathbb{R}^k$ is a soft one-hot encoding for the high density region of the graph vector $x^{(i)}$ is in.
- We can use c_i as a representative for $x^{(i)}$ and use it to cluster. k-means (or any standard clustering algorithm for that matter) should also have an easy time doing the clustering using the representative vectors.

Unnormalized Spectral Clustering Algorithm

Algorithm 3: Unnormalized Spectral Clustering

Input: Similarity matrix $S \in \mathbb{R}^{n \times n}$, desired clusters k

Output: Cluster partition $A_1, \ldots, A_k$, $A_i \subseteq V$.

Construct a symmetric similarity graph $G = (V, E)$ (e.g., fully connected, k-NN, or ϵ-neighborhood).;

Compute unnormalized Laplacian $L = D - W$;

Compute first k eigenvectors $u_1, \ldots, u_k$ of L ;

Let $U \in \mathbb{R}^{n \times k}$ be matrix containing vectors $u_1, \ldots, u_k$ as columns ;

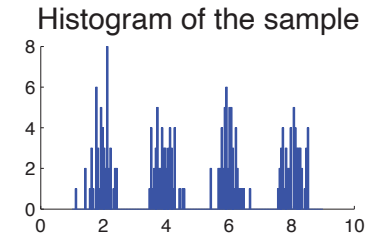
For $i \in [n]$, define $c_i \in \mathbb{R}^k$ to be the i^{th} row of U;

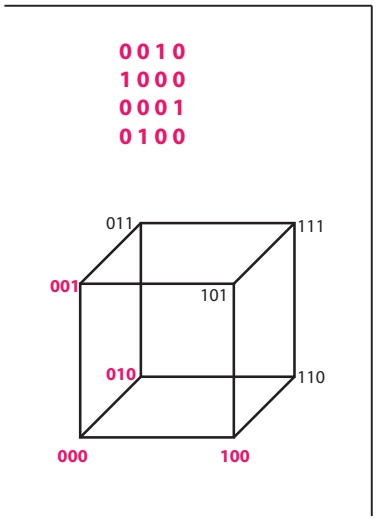
Cluster $\{c_i\}_{i=1}^n$ using k-means into clusters $\mathsf{C}_1, \mathsf{C}_2, \ldots, \mathsf{C}_k$.;

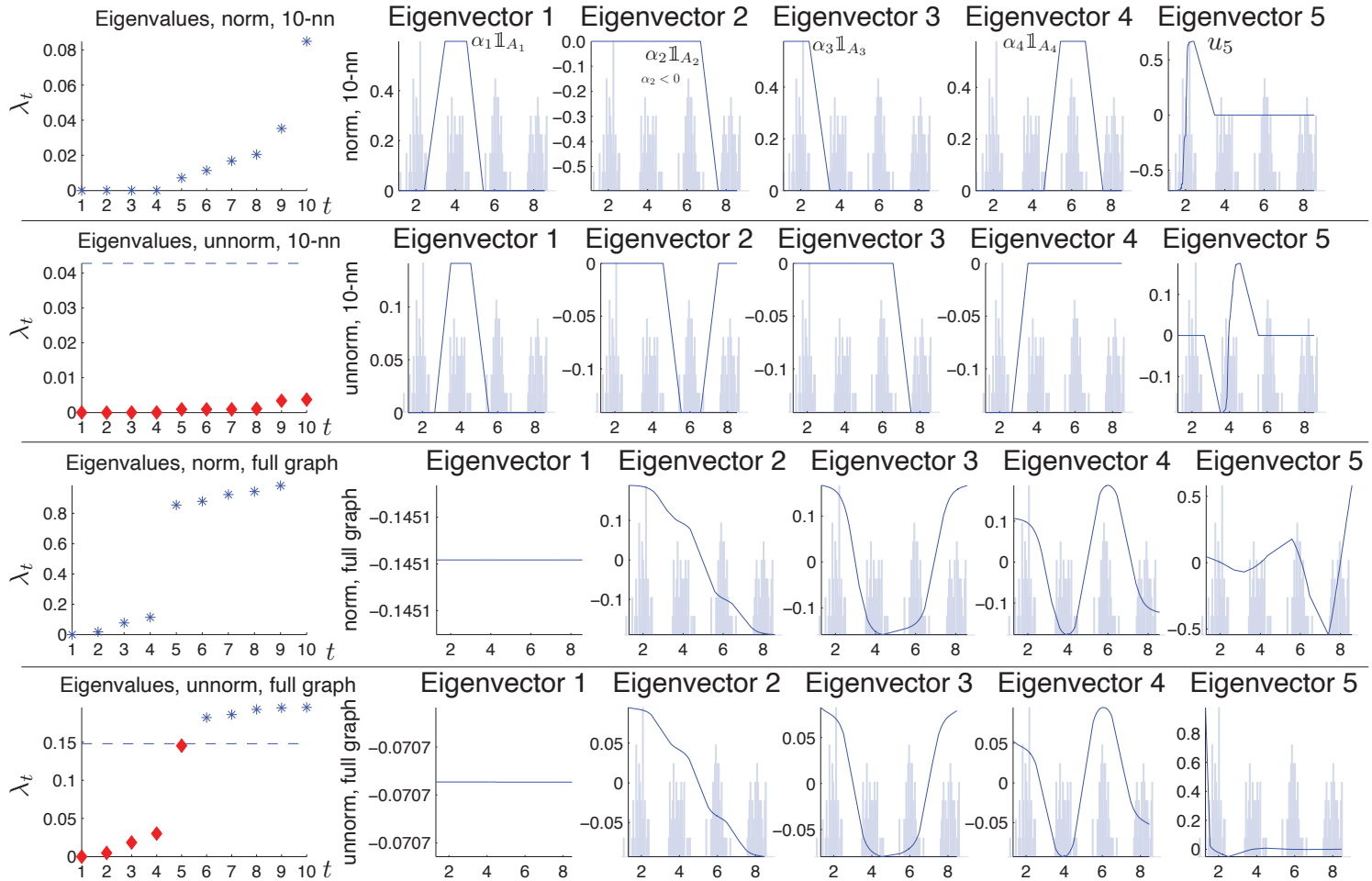
Define each set A_j to correspond to cluster C_j for $j \in [k]$;

Spectral Clustering Example

- Mixture of four 1-D Gaussians, $m = 1$, $n = 200$, $k = 4$, $\sigma = 1$, $s_{i,j} = \exp(-\|x^{(i)} - x^{(j)}\|/(2\sigma^2))$, 10-NN and full graph, normalized (L_{rw}) vs. unnormalized L, λ_t is t^{th} eigenvalue sorted ascending.







From: ``A Tutorial on Spectral Clustering'', Ulrike von Luxburg, 2007

On the previous example

- In 10-NN case, clusterings are clear, since first four eigenvalues are zero, first four eigenvectors are 0/1-valued. 10-NN graph has four connected components.

On the previous example

- In 10-NN case, clusterings are clear, since first four eigenvalues are zero, first four eigenvectors are 0/1-valued. 10-NN graph has four connected components.
- Eg: first row, is like a binary encoding of the clusters. If eigenvectors are 0100, 0001, 1000, and 0010, then binary encodings are 0010, 1000, 0001, 0100, easy to cluster. Easy to cluster into four clusters (using even first three elements, consider verticies of a 3D-cube).

On the previous example

- In 10-NN case, clusterings are clear, since first four eigenvalues are zero, first four eigenvectors are 0/1-valued. 10-NN graph has four connected components.
- Eg: first row, is like a binary encoding of the clusters. If eigenvectors are 0100, 0001, 1000, and 0010, then binary encodings are 0010, 1000, 0001, 0100, easy to cluster. Easy to cluster into four clusters (using even first three elements, consider verticies of a 3D-cube).
- In fully-connected graph case, only one connected components since Gaussian similarity is > 0, so eigenvalues (other than first) are not perfectly zero. First eigenvector is constant (the one connected component), other eigenvectors carry the clustering information.

On the previous example

- In 10-NN case, clusterings are clear, since first four eigenvalues are zero, first four eigenvectors are 0/1-valued. 10-NN graph has four connected components.
- Eg: first row, is like a binary encoding of the clusters. If eigenvectors are 0100, 0001, 1000, and 0010, then binary encodings are 0010, 1000, 0001, 0100, easy to cluster. Easy to cluster into four clusters (using even first three elements, consider verticies of a 3D-cube).
- In fully-connected graph case, only one connected components since Gaussian similarity is > 0, so eigenvalues (other than first) are not perfectly zero. First eigenvector is constant (the one connected component), other eigenvectors carry the clustering information.
- We can still easily cluster using first four eigenvectors even though we have an imperfect graph.

Normalized Laplacian

- Sometimes we normalize the Laplacian which can help with cluster balance.

Normalized Laplacian

- Sometimes we normalize the Laplacian which can help with cluster balance.
- Two types of normalized Laplacian matrix:

$$L_{\text{sym}} = D^{-1/2} L D^{-1/2} = I - D^{-1/2} W D^{-1/2} \tag{10.32}$$

and

$$L_{\text{rw}} = D^{-1} L = I - D^{-1} W \tag{10.33}$$

Normalized Laplacian

- Sometimes we normalize the Laplacian which can help with cluster balance.
- Two types of normalized Laplacian matrix:

$$L_{\text{sym}} = D^{-1/2} L D^{-1/2} = I - D^{-1/2} W D^{-1/2} \tag{10.32}$$

and

$$L_{\text{rw}} = D^{-1} L = I - D^{-1} W \tag{10.33}$$

- $L_{\text{sym}}(i,j) = L(i,j)/\sqrt{d_i d_j}$ and hence L_{sym} is symmetric.

Normalized Laplacian

- Sometimes we normalize the Laplacian which can help with cluster balance.
- Two types of normalized Laplacian matrix:

$$L_{\text{sym}} = D^{-1/2} L D^{-1/2} = I - D^{-1/2} W D^{-1/2} \tag{10.32}$$

and

$$L_{\text{rw}} = D^{-1} L = I - D^{-1} W \tag{10.33}$$

- $L_{\text{sym}}(i,j) = L(i,j)/\sqrt{d_i d_j}$ and hence L_{sym} is symmetric.
- $L_{\text{rw}}(i,j) = L(i,j)/d_i = 1 - w_{ij}/\sum_{j'=1}^{n} w_{ij'}$, and these are useful in random walk analysis (see Luxburg article) since $w_{ij}/\sum_{j'=1}^{n} w_{ij'}$ can be interpreted as a probability distribution over j.

Propositions for Normalized Laplacian

Proposition 10.4.3 (Properties of the normalized graph Laplacian L)

1. *For every vector $z \in \mathbb{R}^n$, we have*

$$z^\intercal L_{sym} z = \frac{1}{2} \sum_{i,j=1}^{n} w_{ij} (z_i/\sqrt{d_i} - z_j/\sqrt{d_j})^2 \tag{10.34}$$

2. *λ is eigenvalue of L_{rw} with eigenvalue u $\iff$ λ is eigenvalue of L_{sym} with eigenvector $w = D^{1/2}u$.*
3. *λ is eigenvalue of L_{rw} with eigenvalue u $\iff$ λ and u solve the generalized eigenvector/value problem $Lu = \lambda Du$.*
4. *0 is an eigenvalue of L_{rw} with corresponding eigenvector $\mathbb{1} = \mathbb{1}_V \in \mathbb{R}^n$. Also, 0 is an eigenvalue of L_{sym} with eigenvector $D^{1/2}\mathbb{1}$.*
5. *Both L has n non-negative real valued eigenvalues $0 = \lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n$.*
6. *L_{sym} is positive semi-definite and have n non-negative real-valued eigenvalues $0 = \lambda_1 \leq \cdots \leq \lambda_n$.*

Connected components/spectra of L_{sym} and L_{rw}

Proposition 10.4.4 (Connected components/spectra of L_{sym} and L_{rw})

Let G be an undirected graph with non-negative weights. Then the multiplicity of the eigenvalue 0 of both L_{rw} and L_{sym} equals the number of connected components $A_1, A_2, \ldots, A_k$ in the graph. For L_{rw}, the eigenspace of 0 is spanned by the indicator vectors $\mathbb{1}_{A_i}$ of those components. For L_{sym}, the eigenspace of 0 is spanned by the vectors $D^{1/2}\mathbb{1}_{A_i}$.

Normalized Spectral Clustering Algorithm

Algorithm 4: Normalized Spectral Clustering of Ng, Jordan, Weiss 2002

Input: Similarity matrix $S \in \mathbb{R}^{n \times n}$, desired clusters k

Output: Cluster partition $A_1, \ldots, A_k$, $A_i \subseteq V$.

1 Construct a similarity graph $G = (V, E)$ (e.g., fully connected, k-NN, or ϵ-neighborhood).;

2 Compute normalized Laplacian $L_{\text{sym}} = D^{-1/2}(D - W)D^{-1/2}$;

3 Compute first k eigenvectors $u_1, \ldots, u_k$ of L_{sym};

4 Let $U \in \mathbb{R}^{n \times k}$ be matrix containing vectors $u_1, \ldots, u_k$ as columns ;

5 For $i \in [n]$, define $c_i \in \mathbb{R}^k$ to be the i^{th} row of U;

6 For $i \in [n]$, define $\bar{c}_i \in \mathbb{R}^k$ to be the i^{th} normalized row of U, $\bar{c}_i = c_i / \sqrt{\sum_{j=1}^{k} c_{ij}^2}$;

7 Cluster $\{\bar{c}_i\}_{i=1}^{n}$ using k-means into clusters $\mathsf{C}_1, \mathsf{C}_2, \ldots, \mathsf{C}_k$.;

8 Define each set A_j to correspond to cluster C_j for $j \in [k]$;

Cutting Graphs as Clustering

- Recall, the cut between $A, B \subseteq V$ defined as

$$W(A, B) = \sum_{i \in A, j \in B} w_{ij} \tag{10.35}$$

and also $\text{vol}(A) = \sum_{i \in A} d_i$.

Cutting Graphs as Clustering

- Recall, the cut between $A, B \subseteq V$ defined as

$$W(A, B) = \sum_{i \in A, j \in B} w_{ij} \tag{10.35}$$

  and also $\text{vol}(A) = \sum_{i \in A} d_i$.

- Given partition $A_1, \ldots, A_k$ overall k-cut is

$$\text{cut}(A_1, \ldots, A_k) = \frac{1}{2} \sum_{j=1}^{k} W(A_j, \bar{A}_j) \tag{10.36}$$

Cutting Graphs as Clustering

- Recall, the cut between $A, B \subseteq V$ defined as

$$W(A, B) = \sum_{i \in A, j \in B} w_{ij} \tag{10.35}$$

  and also $\text{vol}(A) = \sum_{i \in A} d_i$.
- Given partition $A_1, \ldots, A_k$ overall k-cut is

$$\text{cut}(A_1, \ldots, A_k) = \frac{1}{2} \sum_{j=1}^{k} W(A_j, \bar{A}_j) \tag{10.36}$$

- Min-cut problems for $k = 2$ is well solved (max-flow/min-cut algorithms) and k-cut is polynomial time solvable for constant k (see Goldschmidt & Hochbaum, 1988).

Cutting Graphs as Clustering

- Recall, the cut between $A, B \subseteq V$ defined as

$$W(A, B) = \sum_{i \in A, j \in B} w_{ij} \tag{10.35}$$

  and also $\text{vol}(A) = \sum_{i \in A} d_i$.
- Given partition $A_1, \ldots, A_k$ overall k-cut is

$$\text{cut}(A_1, \ldots, A_k) = \frac{1}{2} \sum_{j=1}^{k} W(A_j, \bar{A}_j) \tag{10.36}$$

- Min-cut problems for $k = 2$ is well solved (max-flow/min-cut algorithms) and k-cut is polynomial time solvable for constant k (see Goldschmidt & Hochbaum, 1988).
- But using a graph cut for clustering can lead to very imbalanced clusters (e.g., consider pendant nodes in graph G).

Normalized Graph Cut

- Solution: normalized cut, two varieties:

$$\text{RatioCut}(A_1, \ldots, A_k) = \frac{1}{2} \sum_{j=1}^{k} \frac{W(A_i, \bar{A}_i)}{|A_i|} \tag{10.37}$$

and

$$\text{Ncut}(A_1, \ldots, A_k) = \frac{1}{2} \sum_{j=1}^{k} \frac{W(A_i, \bar{A}_i)}{\text{vol}(A_i)} \tag{10.38}$$

Normalized Graph Cut

- Solution: normalized cut, two varieties:

$$\text{RatioCut}(A_1, \ldots, A_k) = \frac{1}{2} \sum_{j=1}^{k} \frac{W(A_i, \bar{A}_i)}{|A_i|} \tag{10.37}$$

and

$$\text{Ncut}(A_1, \ldots, A_k) = \frac{1}{2} \sum_{j=1}^{k} \frac{W(A_i, \bar{A}_i)}{\text{vol}(A_i)} \tag{10.38}$$

- Since goal is min-cut, both forms penalize cut solution depending on how "small" A_i is, leading to more balanced partitions.

Normalized Graph Cut

- Solution: normalized cut, two varieties:

$$\text{RatioCut}(A_1, \ldots, A_k) = \frac{1}{2} \sum_{j=1}^{k} \frac{W(A_i, \bar{A}_i)}{|A_i|} \tag{10.37}$$

and

$$\text{Ncut}(A_1, \ldots, A_k) = \frac{1}{2} \sum_{j=1}^{k} \frac{W(A_i, \bar{A}_i)}{\text{vol}(A_i)} \tag{10.38}$$

- Since goal is min-cut, both forms penalize cut solution depending on how "small" A_i is, leading to more balanced partitions.
- In the next set of frames, we'll relate RatioCut to the unnormalized graph Laplacian.

RatioCut with $k = 2$

- With $k = 2$, problem becomes $\min_{A \subseteq V} \text{RatioCut}(A, \bar{A})$.

RatioCut with $k = 2$

- With $k = 2$, problem becomes $\min_{A \subseteq V} \text{RatioCut}(A, \bar{A})$.
- For any $A \subset V$, define $z \in \mathbb{R}^n$ as

$$z_i = \begin{cases} \sqrt{|\bar{A}|/|A|} & \text{if } i \in A \\ -\sqrt{|A|/|\bar{A}|} & \text{if } i \in \bar{A} \end{cases} \tag{10.39}$$

RatioCut with $k = 2$

- With $k = 2$, problem becomes $\min_{A \subseteq V} \text{RatioCut}(A, \bar{A})$.
- For any $A \subset V$, define $z \in \mathbb{R}^n$ as

$$z_i = \begin{cases} \sqrt{|\bar{A}|/|A|} & \text{if } i \in A \\ -\sqrt{|A|/|\bar{A}|} & \text{if } i \in \bar{A} \end{cases} \tag{10.39}$$

- From graph Laplacian to ratio cut, a bit of algebra shows that:

$$z^\intercal L z = \frac{1}{2} \sum_{i,j=1}^{n} w_{ij}(z_i - z_j)^2 = |V| \cdot \text{RatioCut}(A, \bar{A}) \tag{10.40}$$

and also that z is orthogonal to $\mathbb{1}$, i.e.,

$$\langle \mathbb{1}, z \rangle = \sum_{i=1}^{n} z_i = 0 \tag{10.41}$$

and also that $\|z\|^2 = n$.

Discrete Optimization and Relaxation

- This suggests a discrete optimization problem using the graph Laplacian, as follows, will solve the $k = 2$ ratio cut problem;

$$\min_{A \subset V} z^\intercal L z \text{ subject to } z \text{ defined via } A \text{ (i.e., Eq. (10.39).)} \quad (10.42)$$

Discrete Optimization and Relaxation

- This suggests a discrete optimization problem using the graph Laplacian, as follows, will solve the $k = 2$ ratio cut problem;

$$\min_{A \subset V} z^\mathsf{T} L z \text{ subject to } z \text{ defined via } A \text{ (i.e., Eq. (10.39).)} \quad (10.42)$$

- thus, for any A we have that $z \perp \mathbb{1}$ and $\|z\| = \sqrt{n}$.

Discrete Optimization and Relaxation

- This suggests a discrete optimization problem using the graph Laplacian, as follows, will solve the $k = 2$ ratio cut problem;

$$\min_{A \subset V} z^\intercal L z \text{ subject to } z \text{ defined via } A \text{ (i.e., Eq. (10.39).)} \quad (10.42)$$

- thus, for any A we have that $z \perp \mathbb{1}$ and $\|z\| = \sqrt{n}$.
- We can relax the above problem to:

$$\min_{z \in \mathbb{R}^n} z^\intercal L z \text{ subject to } z \perp \mathbb{1} \text{ and } \|z\| = \sqrt{n}. \quad (10.43)$$

Discrete Optimization and Relaxation

- This suggests a discrete optimization problem using the graph Laplacian, as follows, will solve the $k = 2$ ratio cut problem;

$$\min_{A \subset V} z^\mathsf{T} L z \text{ subject to } z \text{ defined via } A \text{ (i.e., Eq. (10.39).)} \quad (10.42)$$

- thus, for any A we have that $z \perp \mathbb{1}$ and $\|z\| = \sqrt{n}$.
- We can relax the above problem to:

$$\min_{z \in \mathbb{R}^n} z^\mathsf{T} L z \text{ subject to } z \perp \mathbb{1} \text{ and } \|z\| = \sqrt{n}. \quad (10.43)$$

- Gives eigenvector corresponding to second smallest eigenvalue, since eigenvector corresponding to smallest eigenvalue $(= 0)$ is $\mathbb{1}$.

Discrete Optimization and Relaxation

- This suggests a discrete optimization problem using the graph Laplacian, as follows, will solve the $k = 2$ ratio cut problem;

$$\min_{A \subset V} z^\intercal L z \text{ subject to } z \text{ defined via } A \text{ (i.e., Eq. (10.39).)} \quad (10.42)$$

- thus, for any A we have that $z \perp \mathbb{1}$ and $\|z\| = \sqrt{n}$.
- We can relax the above problem to:

$$\min_{z \in \mathbb{R}^n} z^\intercal L z \text{ subject to } z \perp \mathbb{1} \text{ and } \|z\| = \sqrt{n}. \quad (10.43)$$

- Gives eigenvector corresponding to second smallest eigenvalue, since eigenvector corresponding to smallest eigenvalue $(= 0)$ is $\mathbb{1}$.
- To get back a discrete solution, can either use sign (i.e., $v_i \in A$ iff $z_i \geq 0$) of solution vector z

Discrete Optimization and Relaxation

- This suggests a discrete optimization problem using the graph Laplacian, as follows, will solve the $k = 2$ ratio cut problem;

$$\min_{A \subset V} z^\mathsf{T} L z \text{ subject to } z \text{ defined via } A \text{ (i.e., Eq. (10.39).)} \quad (10.42)$$

- thus, for any A we have that $z \perp \mathbb{1}$ and $\|z\| = \sqrt{n}$.
- We can relax the above problem to:

$$\min_{z \in \mathbb{R}^n} z^\mathsf{T} L z \text{ subject to } z \perp \mathbb{1} \text{ and } \|z\| = \sqrt{n}. \quad (10.43)$$

- Gives eigenvector corresponding to second smallest eigenvalue, since eigenvector corresponding to smallest eigenvalue $(= 0)$ is $\mathbb{1}$.
- To get back a discrete solution, can either use sign (i.e., $v_i \in A$ iff $z_i \geq 0$) of solution vector z
- Alternatively, we can cluster z using k-means with $k = 2$.

Discrete Optimization and Relaxation

- This suggests a discrete optimization problem using the graph Laplacian, as follows, will solve the $k = 2$ ratio cut problem;

$$\min_{A \subset V} z^\intercal L z \text{ subject to } z \text{ defined via } A \text{ (i.e., Eq. (10.39).)} \quad (10.42)$$

- thus, for any A we have that $z \perp \mathbb{1}$ and $\|z\| = \sqrt{n}$.
- We can relax the above problem to:

$$\min_{z \in \mathbb{R}^n} z^\intercal L z \text{ subject to } z \perp \mathbb{1} \text{ and } \|z\| = \sqrt{n}. \quad (10.43)$$

- Gives eigenvector corresponding to second smallest eigenvalue, since eigenvector corresponding to smallest eigenvalue $(= 0)$ is $\mathbb{1}$.
- To get back a discrete solution, can either use sign (i.e., $v_i \in A$ iff $z_i \geq 0$) of solution vector z
- Alternatively, we can cluster z using k-means with $k = 2$.
- This last step is the normalized spectral clustering for $k = 2$. For $k > 2$ an analogous thing happens.

Discrete Optimization and Relaxation

- This suggests a discrete optimization problem using the graph Laplacian, as follows, will solve the $k = 2$ ratio cut problem;

$$\min_{A \subset V} z^\intercal L z \text{ subject to } z \text{ defined via } A \text{ (i.e., Eq. (10.39).)} \quad (10.42)$$

- thus, for any A we have that $z \perp \mathbb{1}$ and $\|z\| = \sqrt{n}$.
- We can relax the above problem to:

$$\min_{z \in \mathbb{R}^n} z^\intercal L z \text{ subject to } z \perp \mathbb{1} \text{ and } \|z\| = \sqrt{n}. \quad (10.43)$$

- Gives eigenvector corresponding to second smallest eigenvalue, since eigenvector corresponding to smallest eigenvalue $(= 0)$ is $\mathbb{1}$.
- To get back a discrete solution, can either use sign (i.e., $v_i \in A$ iff $z_i \geq 0$) of solution vector z
- Alternatively, we can cluster z using k-means with $k = 2$.
- This last step is the normalized spectral clustering for $k = 2$. For $k > 2$ an analogous thing happens.
- A similar analysis can find a correspondence between L_{sym}, RatioCut, and a similar spectral clustering algorithm (Shi and Malik, 2000).

Other issues

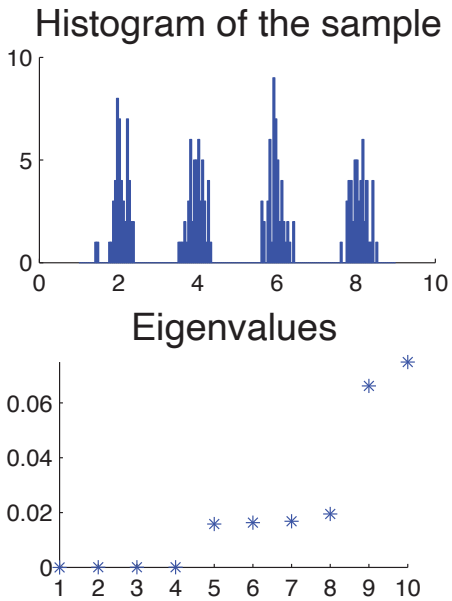
- Similarity kernel makes a big difference, as can width parameter σ^2.

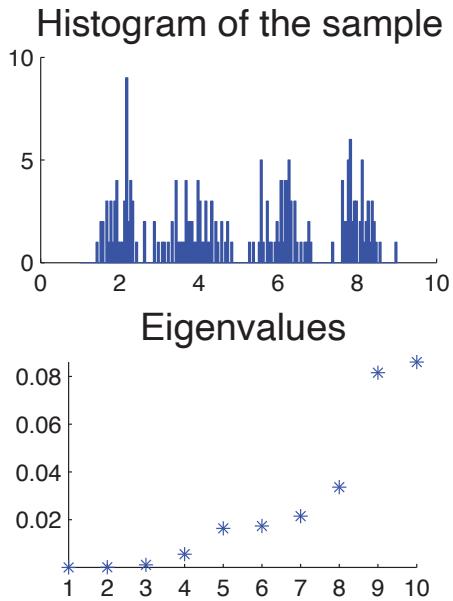
Other issues

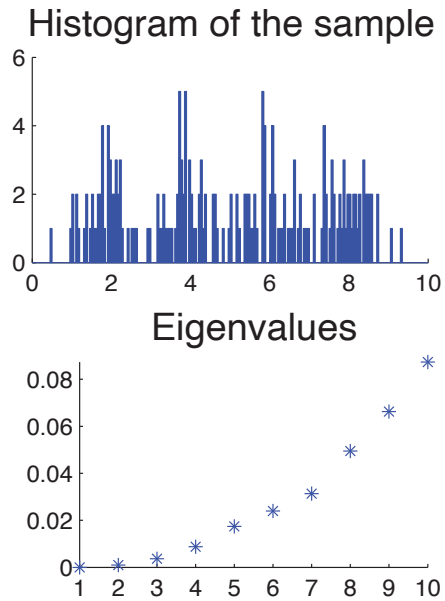
- Similarity kernel makes a big difference, as can width parameter σ^2.
- Type of similarity graph can make a difference, how many nearest neighbors, or how big ϵ.

Other issues

- Similarity kernel makes a big difference, as can width parameter σ^2.
- Type of similarity graph can make a difference, how many nearest neighbors, or how big ϵ.
- Choice of k is important. eigengap heuristic for choosing k, when $|\lambda_{t+1} - \lambda_t|$ gets large, as a function of t we stop and choose $k = t$.

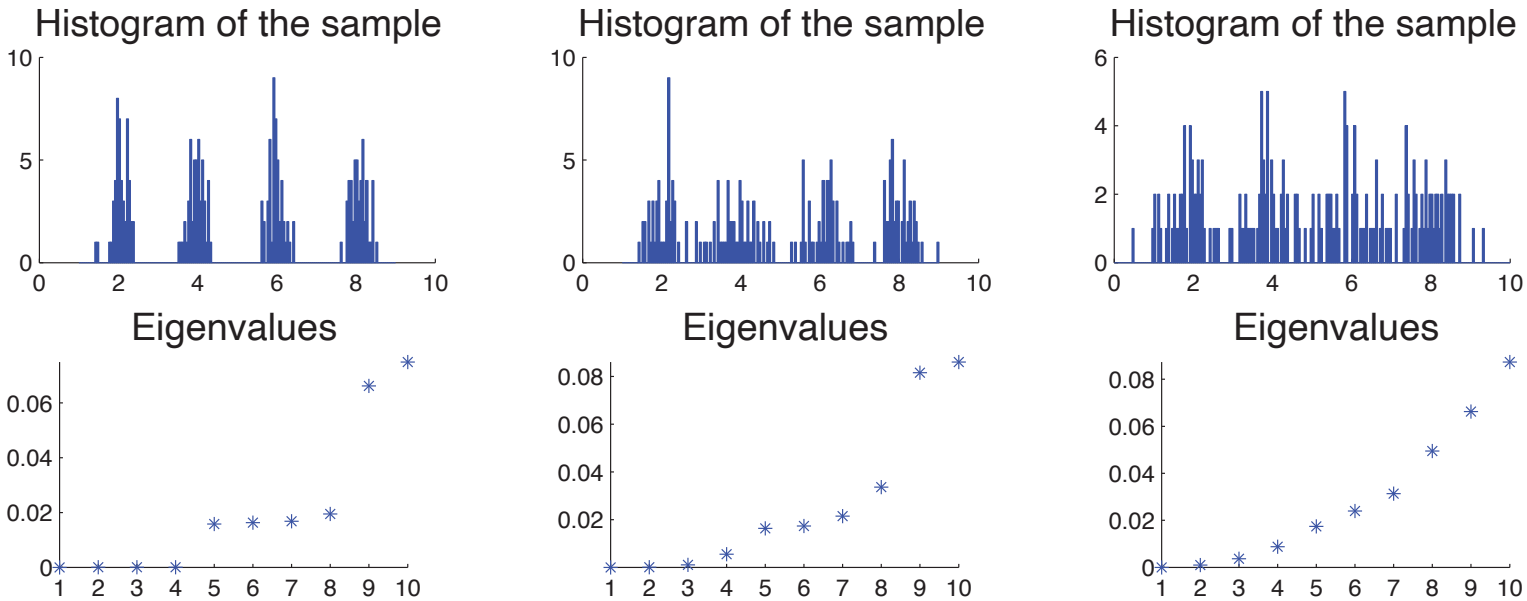






Other issues

- Similarity kernel makes a big difference, as can width parameter σ^2.
- Type of similarity graph can make a difference, how many nearest neighbors, or how big ϵ.
- Choice of k is important. eigengap heuristic for choosing k, when $|\lambda_{t+1} - \lambda_t|$ gets large, as a function of t we stop and choose $k = t$.



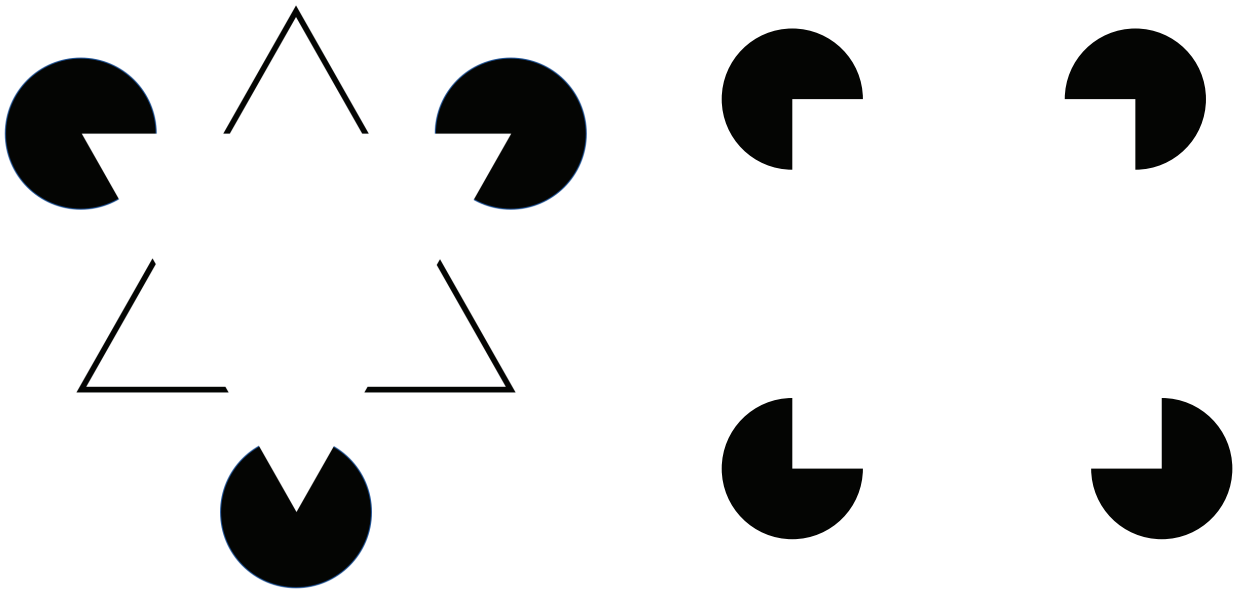
- Computing eigenvectors/eigenvalues is relatively easy using modern linear algebra packages.

Where clusterings fail

- There are situations where most all clusterings will fail. I.e., perceptual clusterings, visual occlusions, illusory contours, that violate most clustering assumptions but cluster nonetheless.

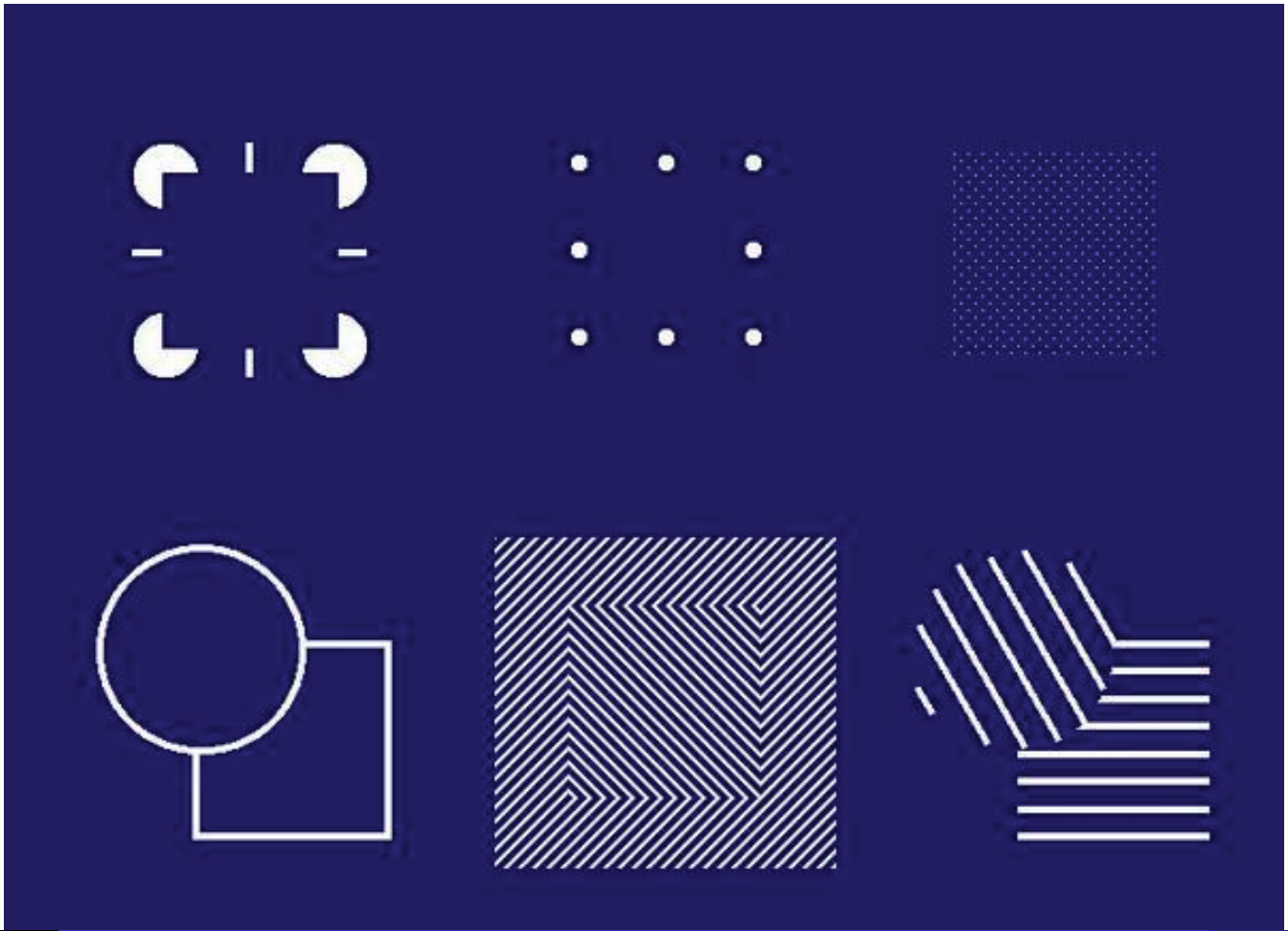
Where clusterings fail

- There are situations where most all clusterings will fail. I.e., perceptual clusterings, visual occlusions, illusory contours, that violate most clustering assumptions but cluster nonetheless.
- A few examples include:



Where clusterings fail

- There are situations where most all clusterings will fail. I.e., perceptual clusterings, visual occlusions, illusory contours, that violate most clustering assumptions but cluster nonetheless.
- A few examples include:



Graph Based Semi-Supervised Learning

- Graph $G = (V, E)$ where V corresponds to all training samples but only a subset $L \subseteq V$ are labeled.

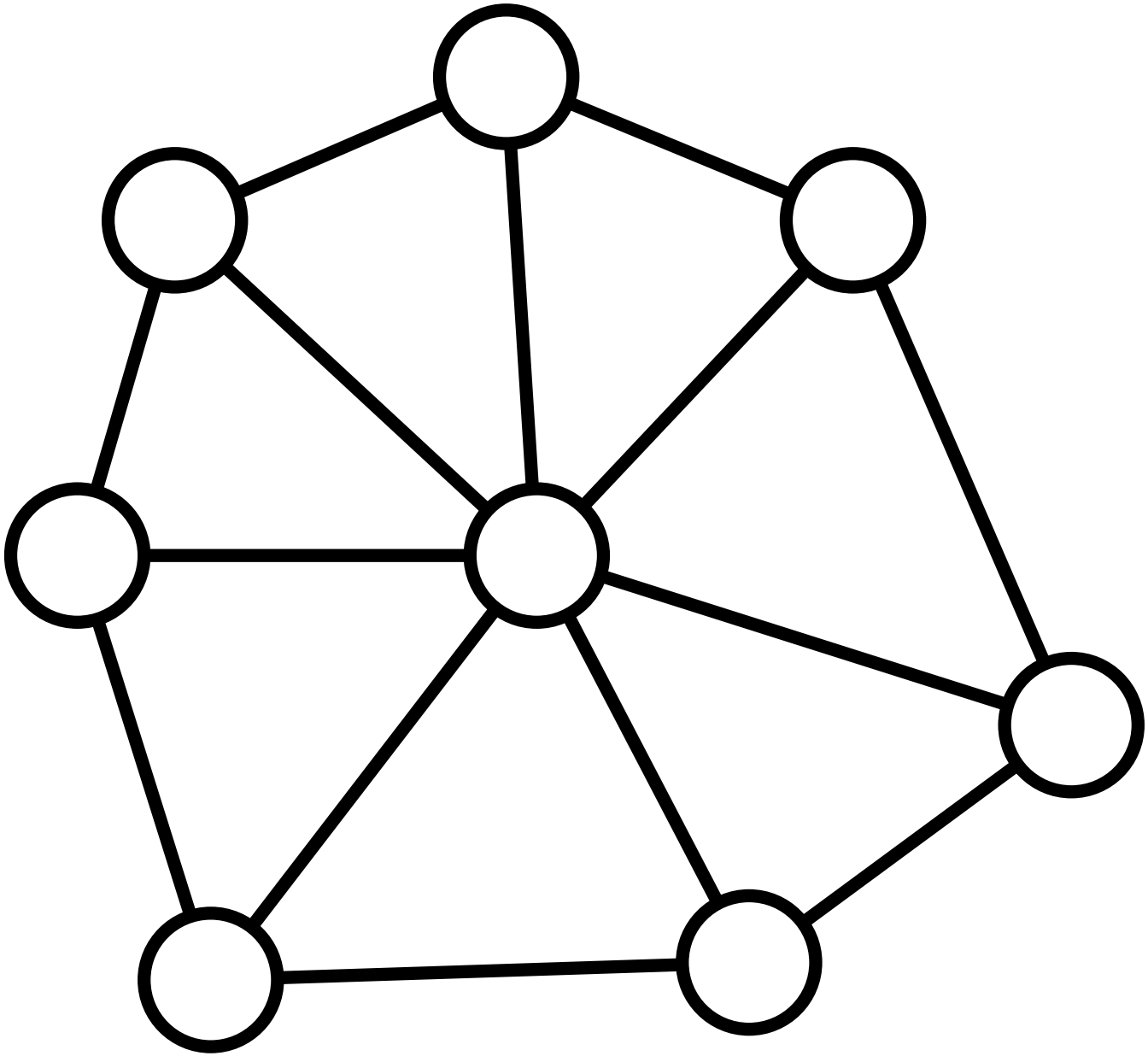
Graph Based Semi-Supervised Learning

- Graph $G = (V, E)$ where V corresponds to all training samples but only a subset $L \subseteq V$ are labeled.
- For each $v \in V$, we have a classification decision based on a probability distribution used to predict the class.

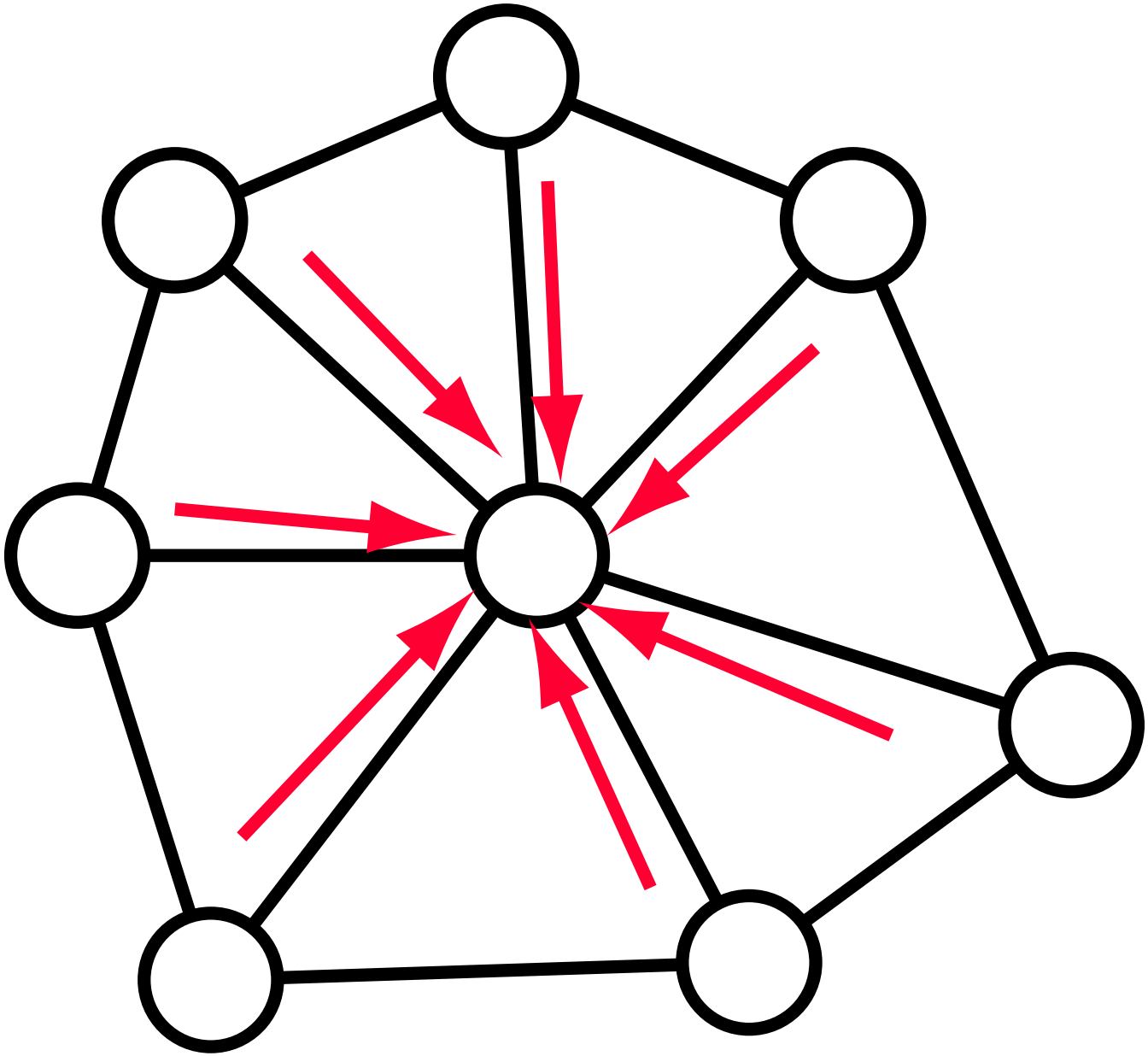
Graph Based Semi-Supervised Learning

- Graph $G = (V, E)$ where V corresponds to all training samples but only a subset $L \subseteq V$ are labeled.
- For each $v \in V$, we have a classification decision based on a probability distribution used to predict the class.
- In classic SSL, there is no input, only the partially labeled data as well as an affinity matrix W with weights w_{ij} indicating the similarity between node i and j.

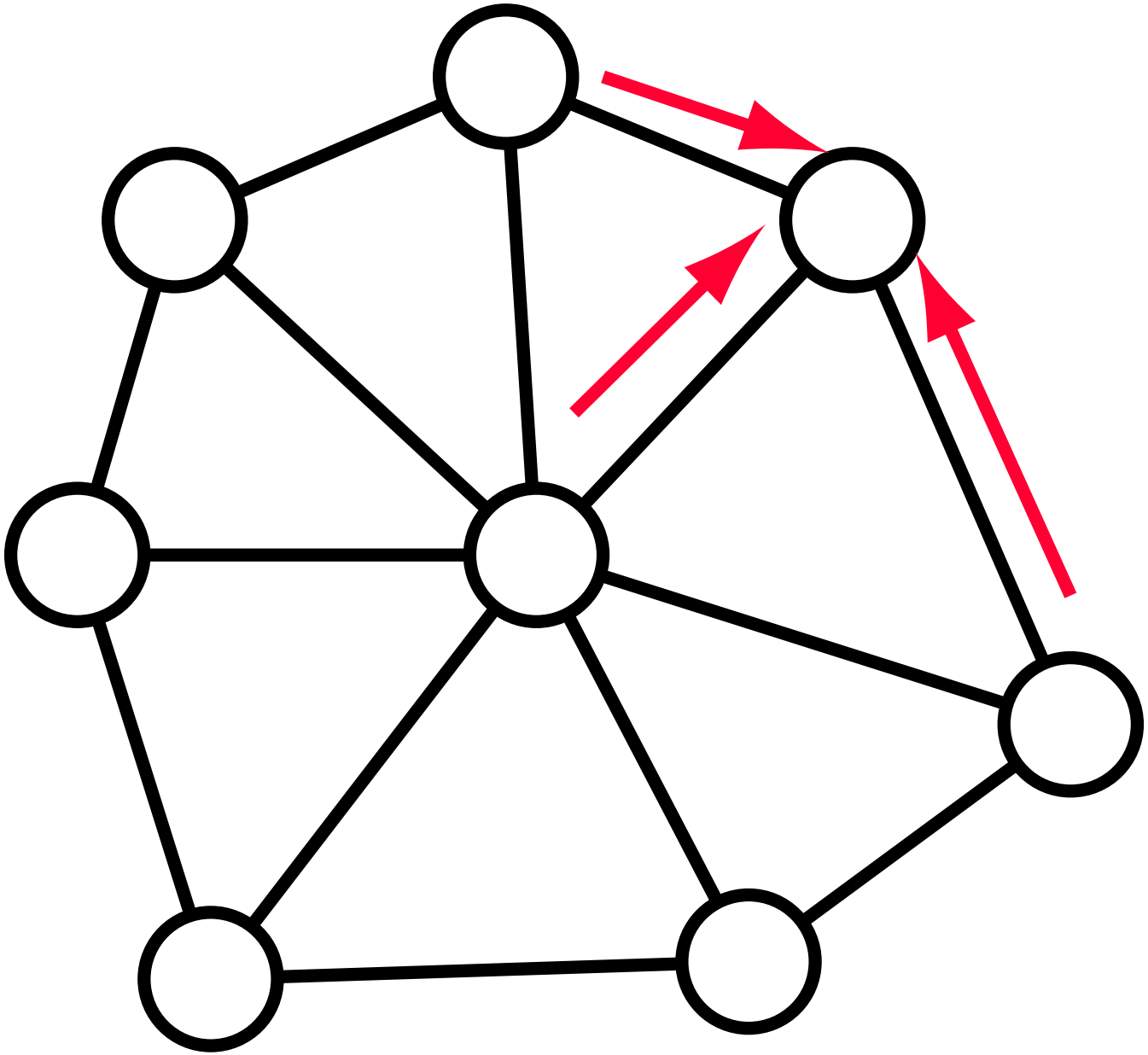
Message passing on graphs



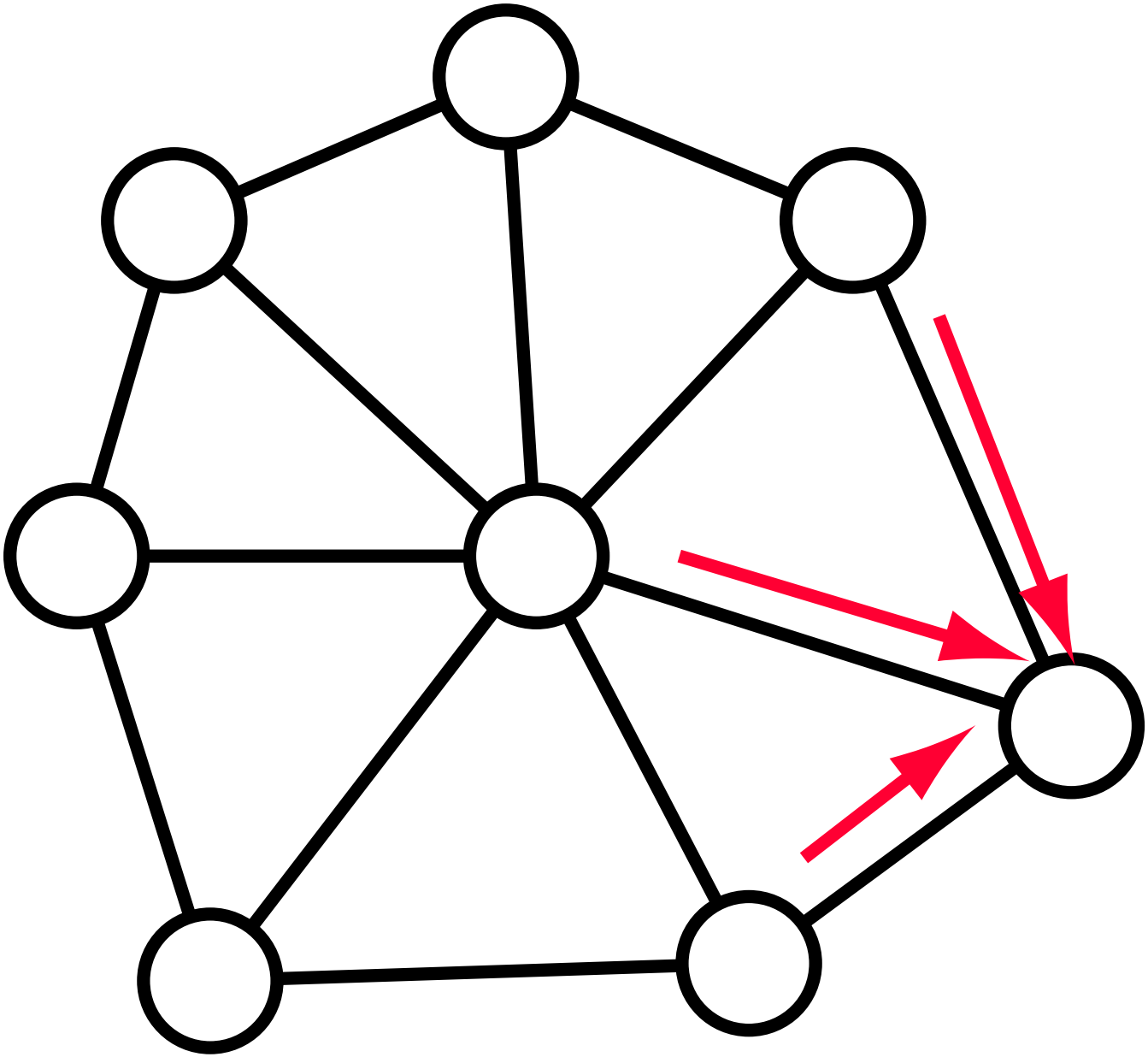
Message passing on graphs



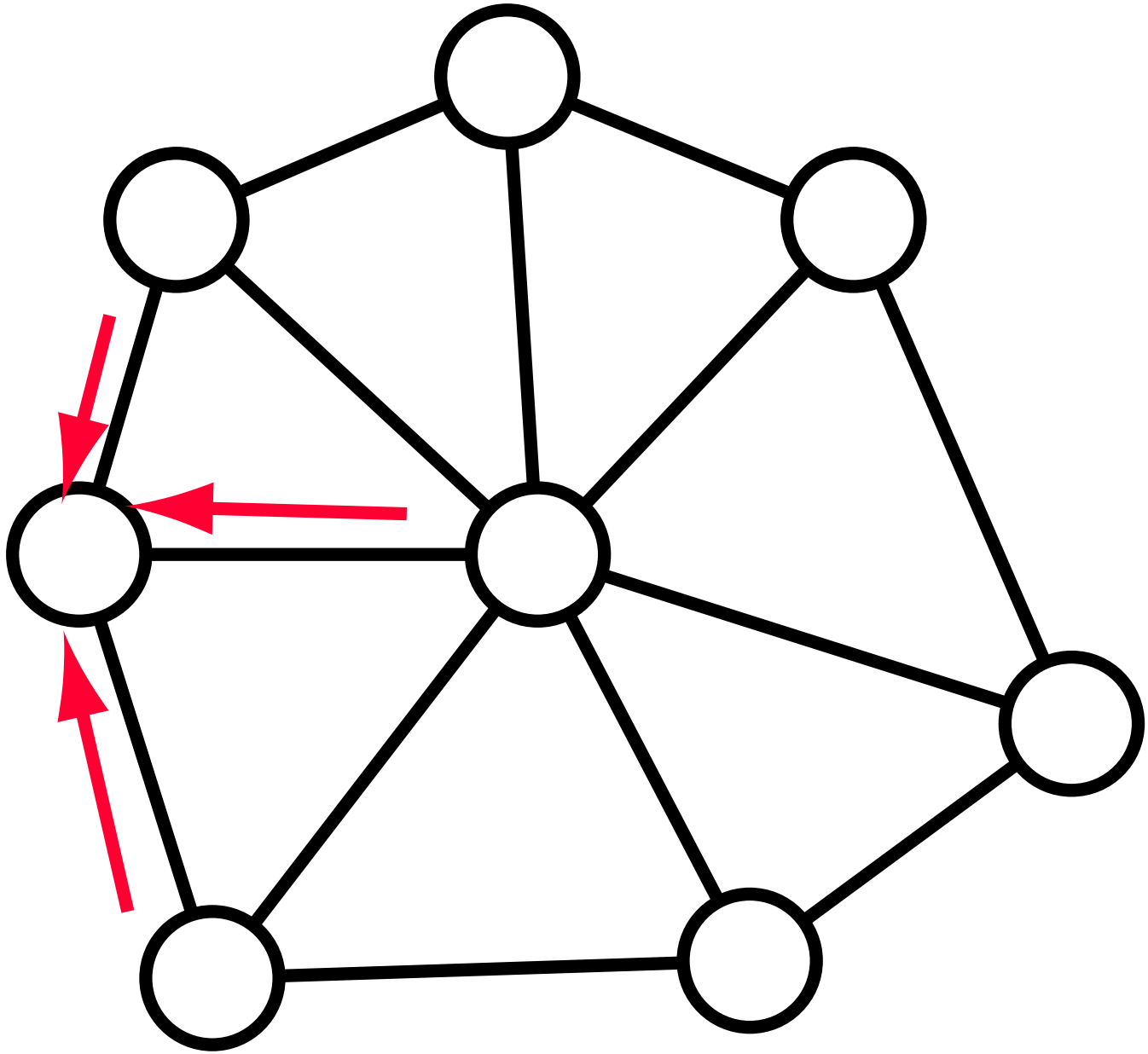
Message passing on graphs



Message passing on graphs



Message passing on graphs



Label Propagation

- Objective: Squared error

$$J(z) = \sum_{v \in V(G) \cap L} (z_v - r_v)^2 + \mu \sum_{u,v \in V} w_{u,v}(z_u - z_v)^2 + \sum_{v \in V} |z_v|^2 \tag{10.44}$$

- Update rules (label propagation)

$$p_i^{(n)}(y) = \frac{r_i(y)\delta(i \leq l) + \delta(i > l)\sum_j w_{ij} p_j^{(n-1)}(y)}{\delta(i \leq l) + \delta(i > l)\sum_j w_{ij}} \tag{10.45}$$

Measure Propagation

$$\mathcal{C}(\mathrm{p},\mathrm{q}) = \sum_{i=1}^{l} D_{KL}\big(r_i||q_i\big) + \mu \sum_{i=1}^{m} \sum_{j\in\mathcal{N}'(i)} w'_{ij} D_{KL}\big(p_i||q_j\big) - \nu \sum_{i=1}^{m} H(p_i).$$

- Entropic objective function for transductive non-parametric semi-supervised learning.
- Convex
- Optimize using the alternative minimization procedure
- Provably guaranteed to converge to the unique global minimum
- Optimization is easy, another message passing form on graphs.

Measure Propagation: Message Passing

$$p_i^{(n)}(y) = \frac{\exp\{\frac{\mu}{\gamma_i}\sum_j w'_{ij}\log q_j^{(n-1)}(y)\}}{\sum_y \exp\{\frac{\mu}{\gamma_i}\sum_j w'_{ij}\log q_j^{(n-1)}(y)\}} \quad \text{and} \quad (10.46)$$

$$q_i^{(n)}(y) = \frac{r_i(y)\delta(i \le l) + \mu\sum_j w'_{ji} p_j^{(n)}(y)}{\delta(i \le l) + \mu\sum_j w'_{ji}} \quad (10.47)$$

where $\gamma_i = \nu + \mu\sum_j w'_{ij}$.

- Easy updates, update p for each node $v \in V$, and then update q for each node $v \in V$.
- No write-contention, only read from neighbors for each phase.

Review

- The next four slides come from lecture 2.

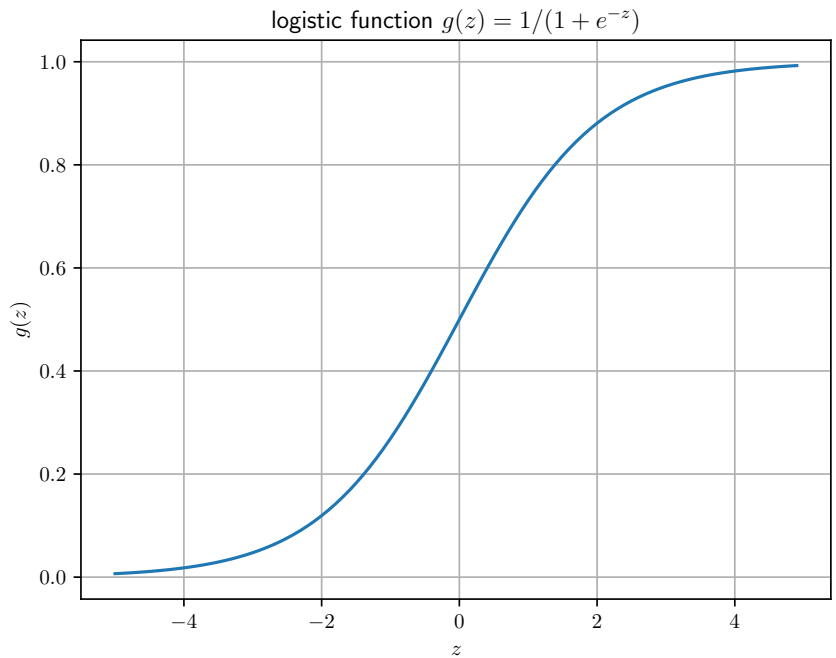
Logistic Regression

- Solution for binary classification: use a logistic function as in

$$\Pr(y = 1|x) = h_\theta(x) = g(\theta^\intercal x) = \frac{1}{1 + \exp(-\theta^\intercal x)} \tag{10.6}$$

where $\theta^\intercal x = \theta_0 + \sum_{i=1}^m \theta_i x_i$, $x_0 \equiv 1$ so θ_0 is the bias/shift.

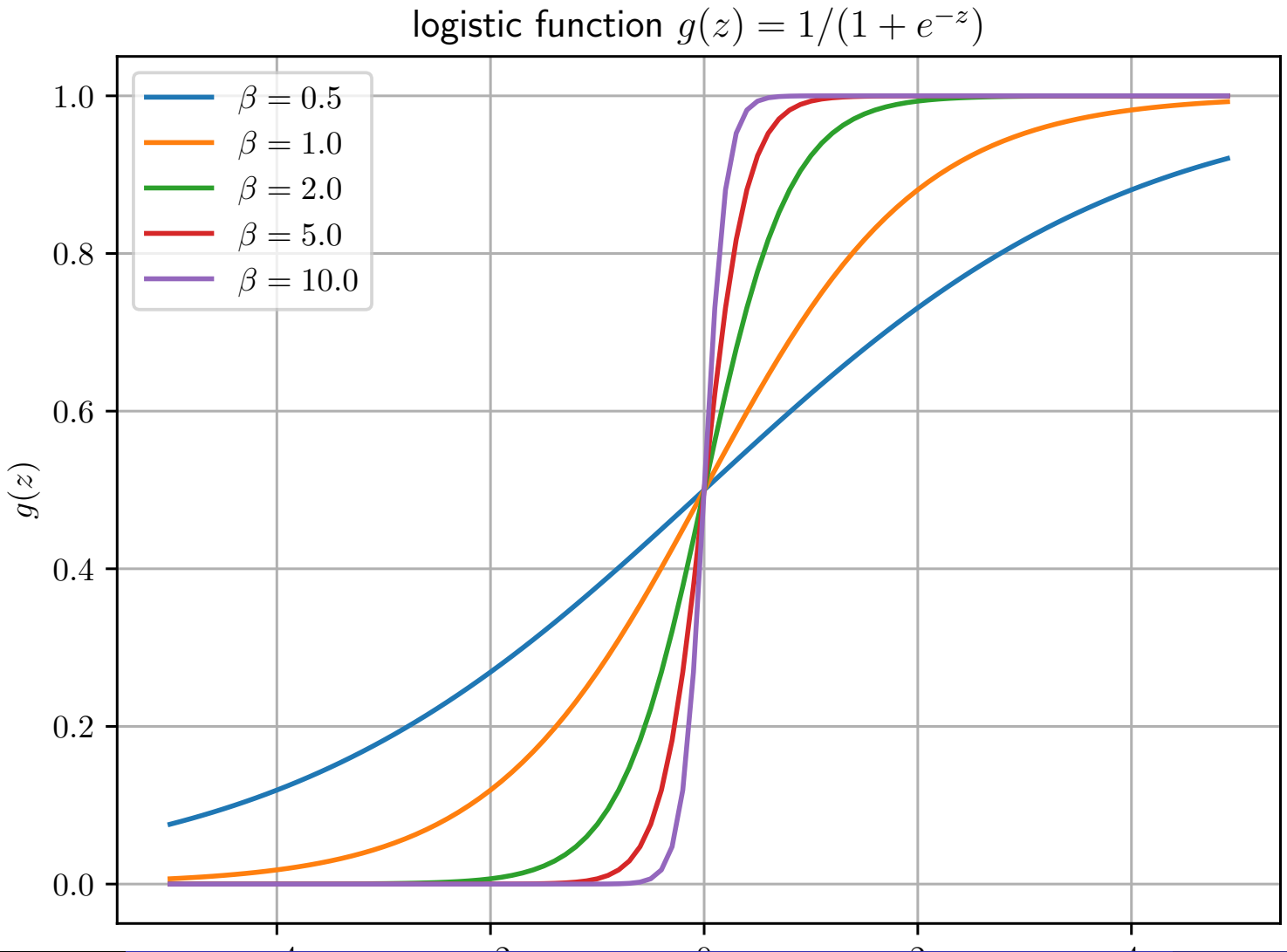
- $g(z) = 1/(1 + e^{-x})$ is known as a logistic function.



- A logistic function is one type of sigmoid function, others being hyperbolic tangent, arctan, error function, etc. (see `https://en.wikipedia.org/wiki/Sigmoid_function`).

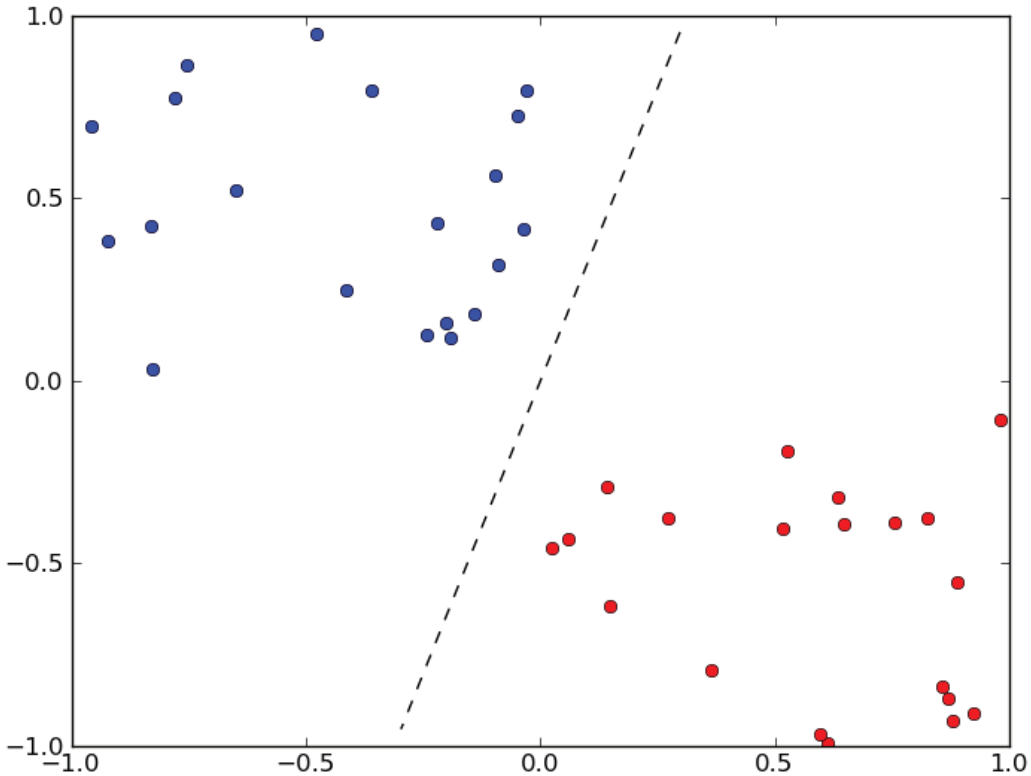
Logistic with scale parameter β

- Approximate a step function with scale parameter $\beta \in \mathbb{R}_+$, giving $g_\beta(z) = 1/(1 + e^{-\beta x})$



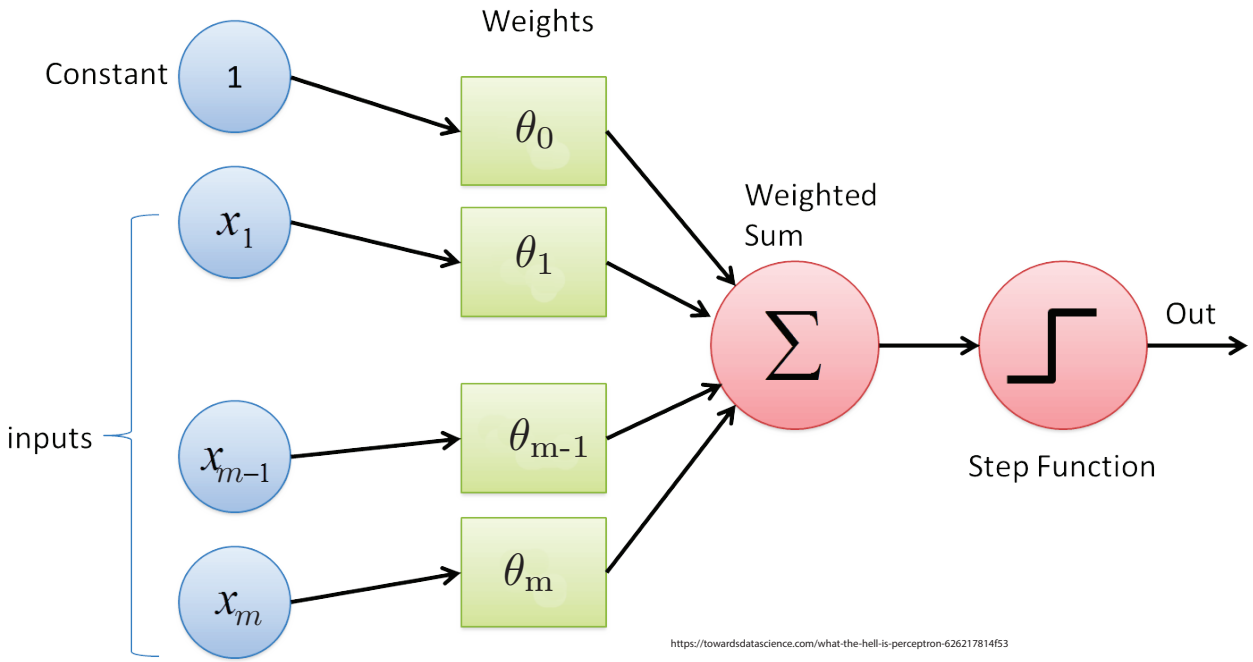
Perceptron, Linear Models, and Linearly Separable Data

- logistic regression and perceptron can do perfectly when the data is (nicely) linearly separable.



- Line designates boundary of a "ridge" or "cliff" between the categories.

Perceptron and Logistic Regression



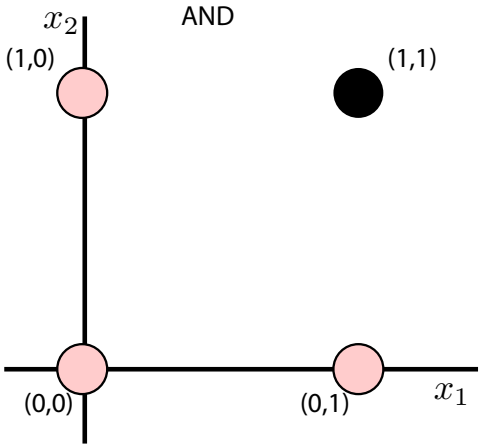
- Model $h_\theta(x) = g(\theta^\mathsf{T} x)$, perceptron uses a hard activation function

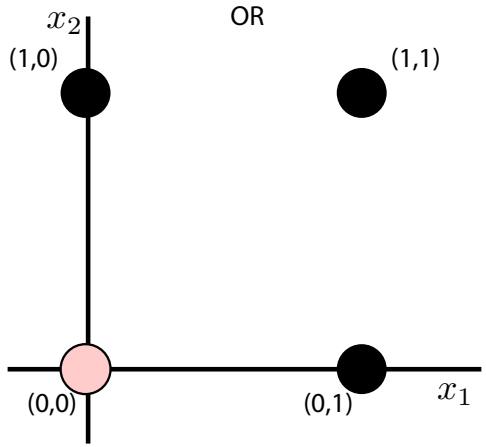
$$g(z) = \begin{cases} -1 & \text{if } z < 0 \\ +1 & \text{if } z \geq 0 \end{cases} \tag{10.8}$$

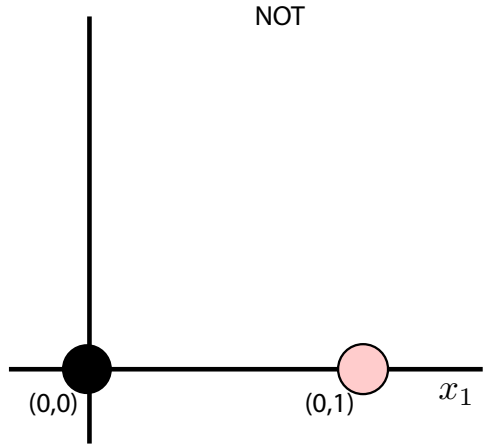
leads to same learning update rule $\theta \leftarrow \theta + \alpha(y^{(i)} - h_\theta(x^{(i)}))x^{(i)}$

Perceptron/Logistic and "and", "or", and "not"

- Given $x \in \mathbb{R}^2$, we depict the "and", "or", and "not" functions.

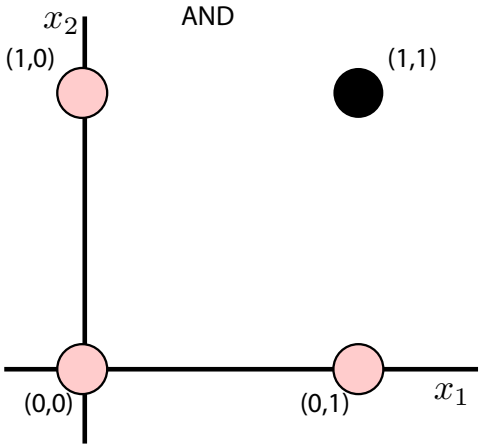


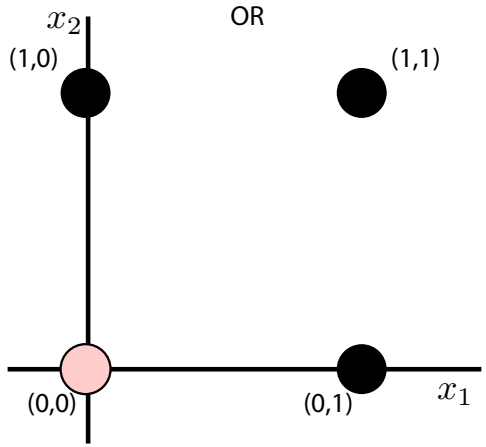


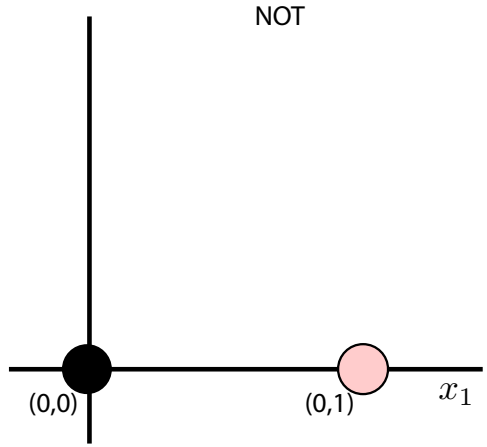


Perceptron/Logistic and "and", "or", and "not"

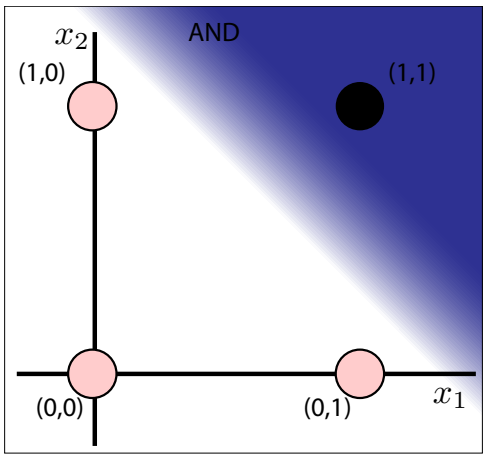
- Given $x \in \mathbb{R}^2$, we depict the "and", "or", and "not" functions.



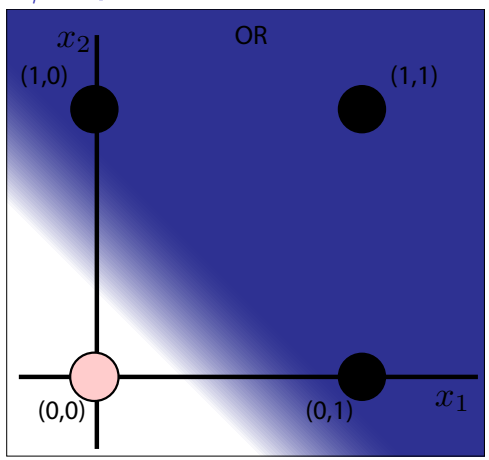




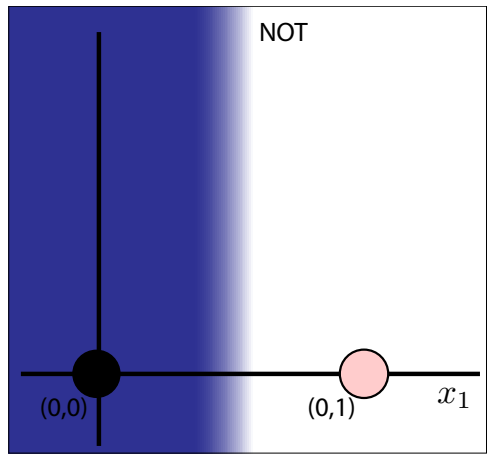
- Logistic regression with g_β (or a perceptron) can easily express these:



$g_\beta(x_1 + x_2 - 1.5)$



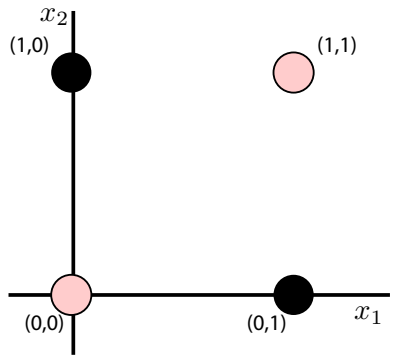
$g_\beta(x_1 + x_2 - 0.5)$



$g_\beta(-x_1 + 0.5)$

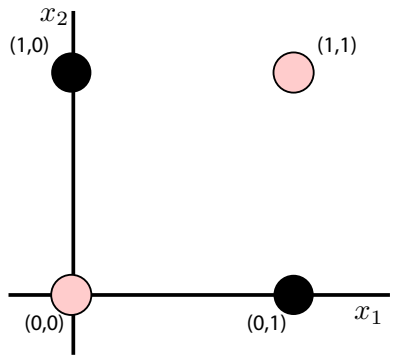
Xor problem

- Suppose we wish to classify the following four points with logistic regresssion, the xor function $y = (x_1 \text{ xor } x_2)$.

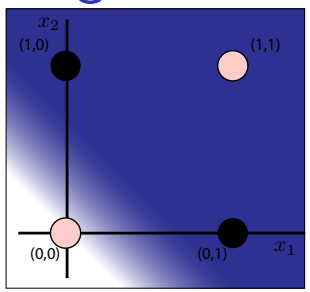


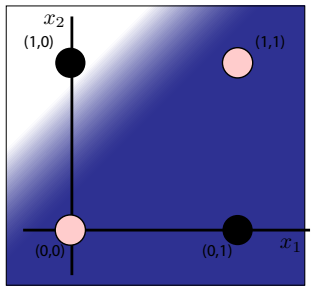
Xor problem

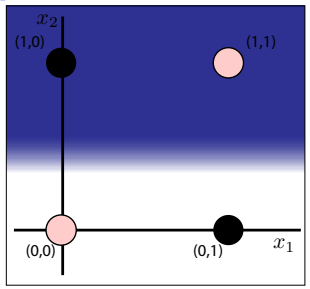
- Suppose we wish to classify the following four points with logistic regresssion, the xor function $y = (x_1 \text{ xor } x_2)$.

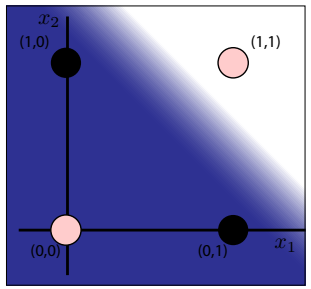


- Logistic regression is limited to linear separator, can we do xor?



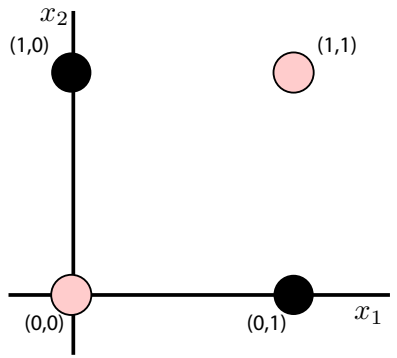




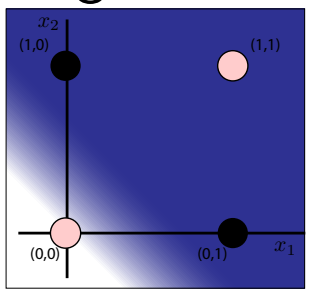


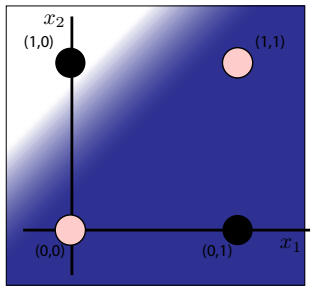
Xor problem

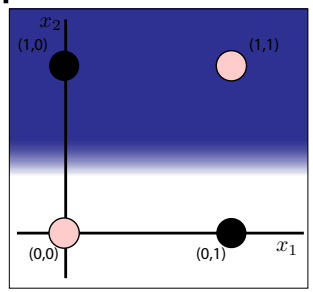
- Suppose we wish to classify the following four points with logistic regresssion, the xor function $y = (x_1 \text{ xor } x_2)$.

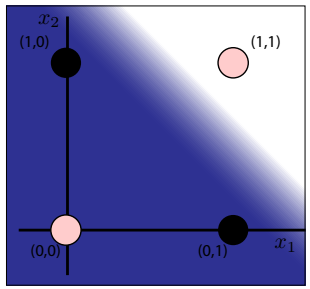


- Logistic regression is limited to linear separator, can we do xor?





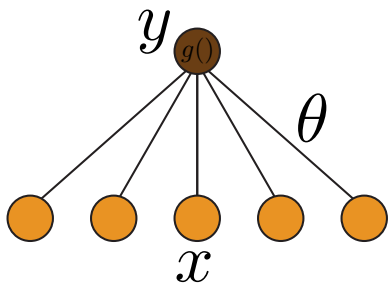




- Logistic regression $\Pr(y = 1|x) = g(\theta^\intercal x) = \frac{1}{1+\exp(-\theta^\intercal x)}$ is too biased, can't do xor for any θ.

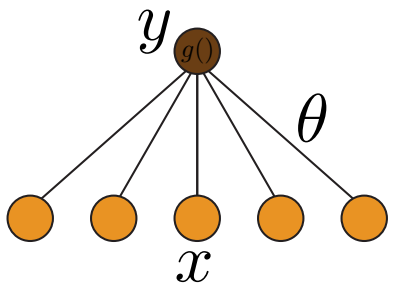
Logistic Regression as a graph

- We can use a simple graph to describe data flow of logistic regression $\Pr(y = 1|x) = g(\theta^\intercal x) = \frac{1}{1+\exp(-\theta^\intercal x)}$

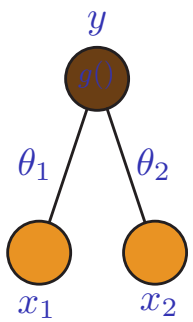


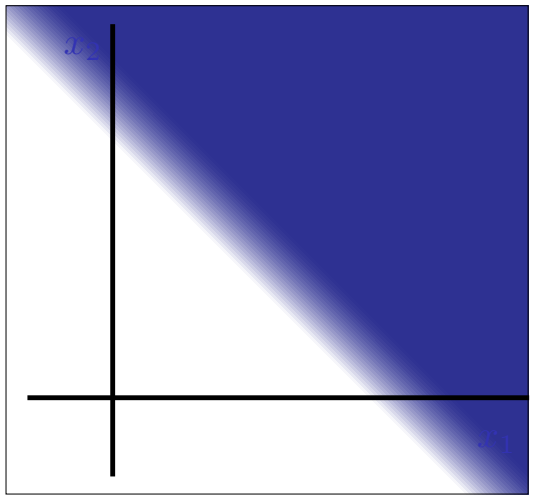
Logistic Regression as a graph

- We can use a simple graph to describe data flow of logistic regression $\Pr(y = 1|x) = g(\theta^\intercal x) = \frac{1}{1+\exp(-\theta^\intercal x)}$



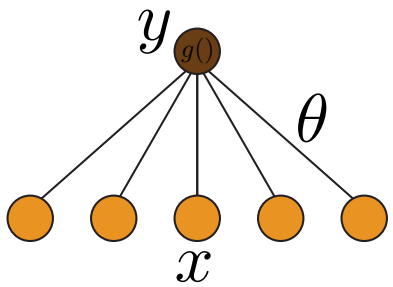
- In two-dimensions, graph becomes



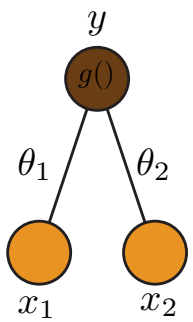


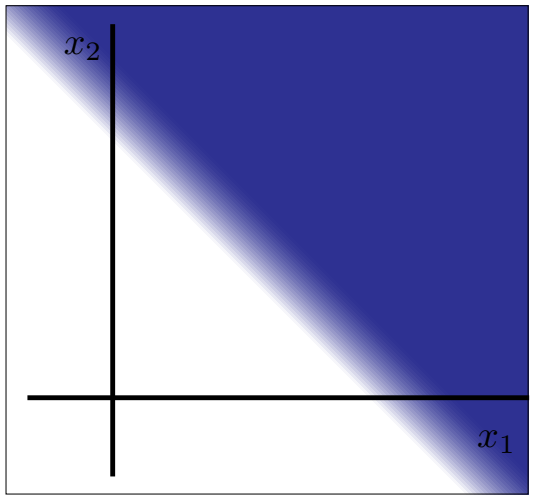
Logistic Regression as a graph

- We can use a simple graph to describe data flow of logistic regression $\Pr(y = 1|x) = g(\theta^\intercal x) = \frac{1}{1+\exp(-\theta^\intercal x)}$



- In two-dimensions, graph becomes

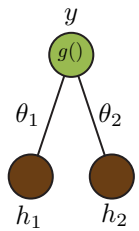


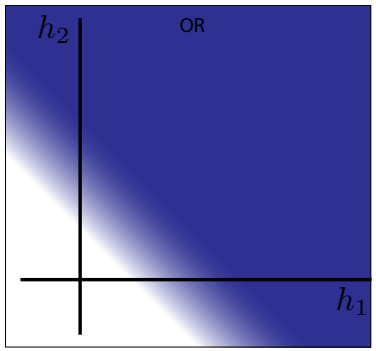


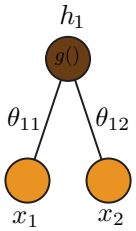
- $y = g(\theta_1 x_1 + \theta_2 x_2 + b)$ where b is the bias.

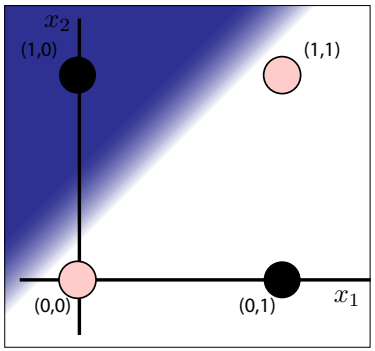
Stacked logistic regression can do xor

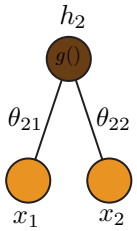
- Using two logistic regressions and stacking another on top of the two can extend the expressivity significantly.

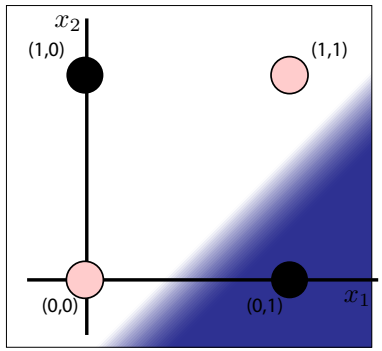


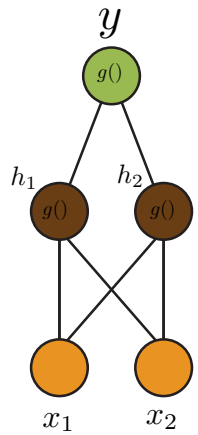


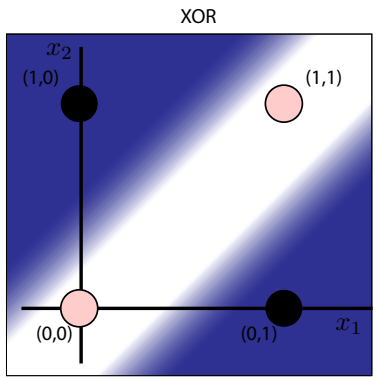






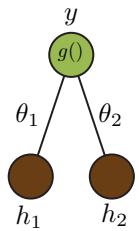


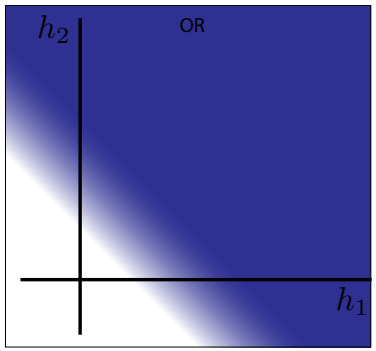


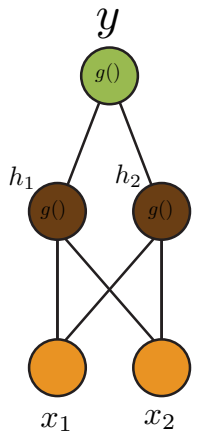


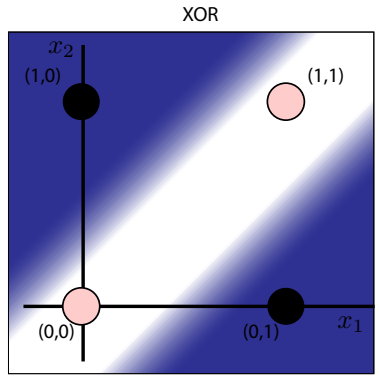
Stacked logistic regression can do xor

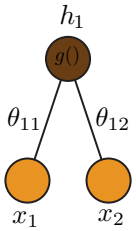
- Using two logistic regressions and stacking another on top of the two can extend the expressivity significantly.
- This is a simple multi-layered perceptron (MLP), or a "shallow" deep neural network (DNN). Can easily represent xor

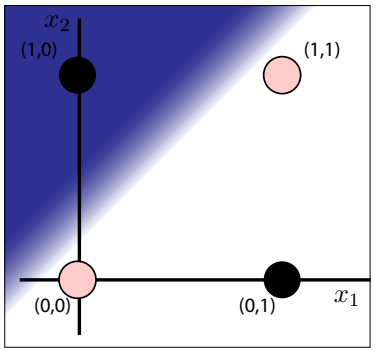


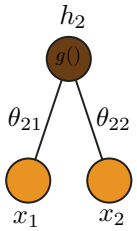


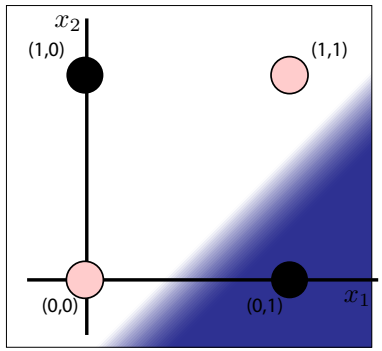






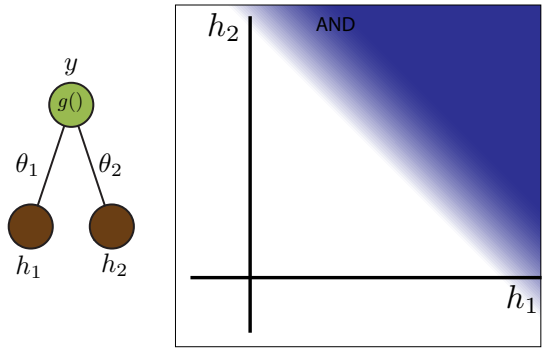


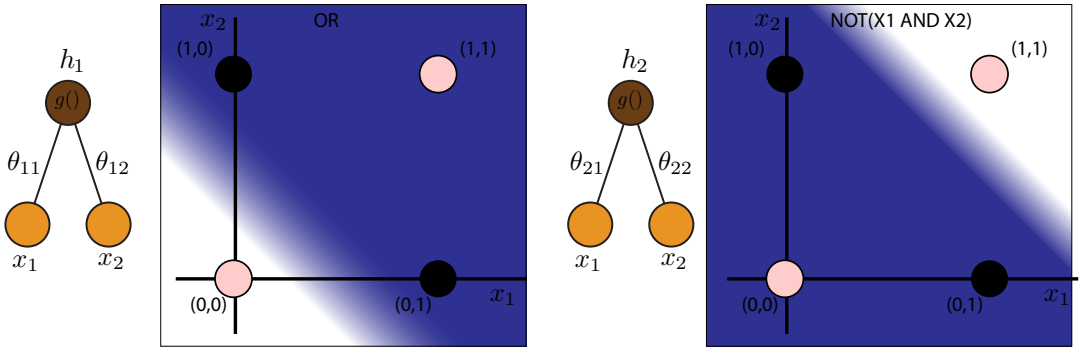


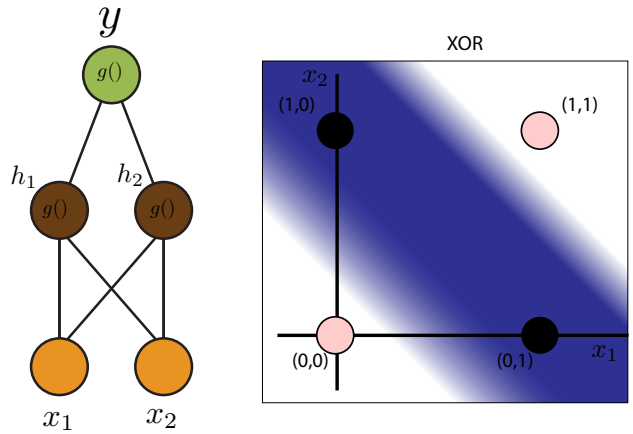


Stacked logistic regression can do xor

- Using two logistic regressions and stacking another on top of the two can extend the expressivity significantly.
- This is a simple multi-layered perceptron (MLP), or a "shallow" deep neural network (DNN). Can easily represent xor in several ways.







Logistic to MLP

- Suppose we have k logistic regression functions: $h_j(x) = g(\theta^{(j)\mathsf{T}} x) = \frac{1}{1+\exp(-\theta^{(j)\mathsf{T}} x)}$ where $x \in \mathbb{R}^m$, $\theta^{(j)} \in \mathbb{R}^m$ for $j \in \{1, 2, \ldots, k\}$ and $g : \mathbb{R} \to \mathbb{R}$ is a logistic function.

Logistic to MLP

- Suppose we have k logistic regression functions: $h_j(x) = g(\theta^{(j)\mathsf{T}} x) = \frac{1}{1+\exp(-\theta^{(j)\mathsf{T}} x)}$ where $x \in \mathbb{R}^m$, $\theta^{(j)} \in \mathbb{R}^m$ for $j \in \{1, 2, \ldots, k\}$ and $g : \mathbb{R} \to \mathbb{R}$ is a logistic function.
- Stack the vectors $\left\{\theta^{(j)\mathsf{T}}\right\}_{j=1}^{k}$ rowwise into $k \times m$ matrix W_{ih} and create vector valued logistic function $\vec{g} : \mathbb{R}^k \to \mathbb{R}^k$.

Logistic to MLP

- Suppose we have k logistic regression functions: $h_j(x) = g(\theta^{(j)\mathsf{T}} x) = \frac{1}{1+\exp(-\theta^{(j)\mathsf{T}} x)}$ where $x \in \mathbb{R}^m$, $\theta^{(j)} \in \mathbb{R}^m$ for $j \in \{1, 2, \ldots, k\}$ and $g : \mathbb{R} \to \mathbb{R}$ is a logistic function.
- Stack the vectors $\left\{\theta^{(j)\mathsf{T}}\right\}_{j=1}^{k}$ rowwise into $k \times m$ matrix W_{ih} and create vector valued logistic function $\vec{g} : \mathbb{R}^k \to \mathbb{R}^k$.
- Then we can form vector-valued function $\vec{h} : \mathbb{R}^m \to \mathbb{R}^k$ defined as $\vec{h}(x) = \vec{g}(W_{\text{ih}} \cdot x)$ which does all of the h outputs at simultaneously.

Logistic to MLP

- Suppose we have k logistic regression functions: $h_j(x) = g(\theta^{(j)\mathsf{T}} x) = \frac{1}{1+\exp(-\theta^{(j)\mathsf{T}} x)}$ where $x \in \mathbb{R}^m$, $\theta^{(j)} \in \mathbb{R}^m$ for $j \in \{1, 2, \ldots, k\}$ and $g : \mathbb{R} \to \mathbb{R}$ is a logistic function.
- Stack the vectors $\left\{\theta^{(j)\mathsf{T}}\right\}_{j=1}^{k}$ rowwise into $k \times m$ matrix W_{ih} and create vector valued logistic function $\vec{g} : \mathbb{R}^k \to \mathbb{R}^k$.
- Then we can form vector-valued function $\vec{h} : \mathbb{R}^m \to \mathbb{R}^k$ defined as $\vec{h}(x) = \vec{g}(W_{\text{ih}} \cdot x)$ which does all of the h outputs at simultaneously.
- $\vec{g}(W_{\text{ih}} \cdot x)$ represents matrix vector multiply $W_{\text{ih}} \cdot x$ followed by applying logistic function to all elements of the vector result.

Logistic to MLP

- Suppose we have k logistic regression functions: $h_j(x) = g(\theta^{(j)\mathsf{T}} x) = \frac{1}{1+\exp(-\theta^{(j)\mathsf{T}} x)}$ where $x \in \mathbb{R}^m$, $\theta^{(j)} \in \mathbb{R}^m$ for $j \in \{1, 2, \ldots, k\}$ and $g : \mathbb{R} \to \mathbb{R}$ is a logistic function.
- Stack the vectors $\left\{\theta^{(j)\mathsf{T}}\right\}_{j=1}^{k}$ rowwise into $k \times m$ matrix W_{ih} and create vector valued logistic function $\vec{g} : \mathbb{R}^k \to \mathbb{R}^k$.
- Then we can form vector-valued function $\vec{h} : \mathbb{R}^m \to \mathbb{R}^k$ defined as $\vec{h}(x) = \vec{g}(W_{\text{ih}} \cdot x)$ which does all of the h outputs at simultaneously.
- $\vec{g}(W_{\text{ih}} \cdot x)$ represents matrix vector multiply $W_{\text{ih}} \cdot x$ followed by applying logistic function to all elements of the vector result.
- One final set of parameters $\theta \in \mathbb{R}^k$ to take a weighted sum of each of the h outputs to give $y = \vec{g}(\theta^{\mathsf{T}} \vec{h}(x))$.

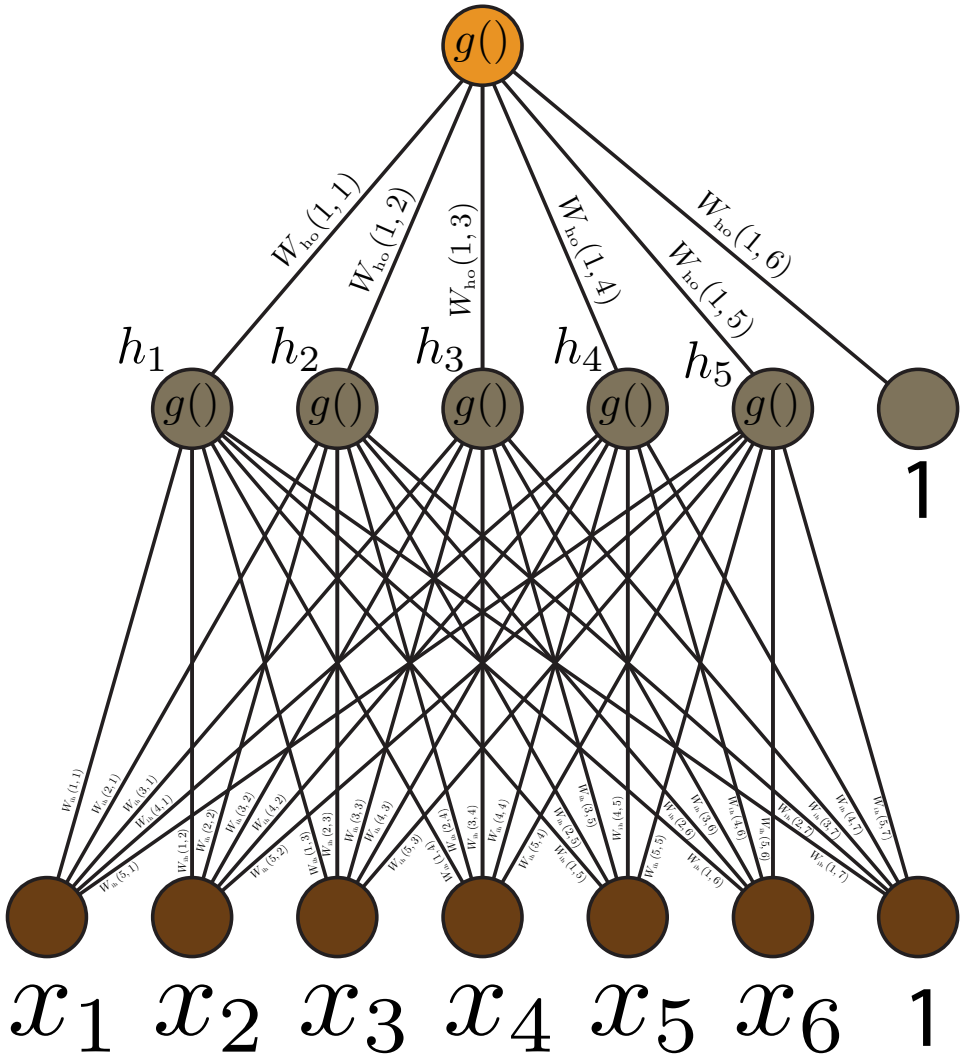
Logistic to MLP

- Suppose we have k logistic regression functions: $h_j(x) = g(\theta^{(j)\mathsf{T}} x) = \frac{1}{1+\exp(-\theta^{(j)\mathsf{T}} x)}$ where $x \in \mathbb{R}^m$, $\theta^{(j)} \in \mathbb{R}^m$ for $j \in \{1, 2, \ldots, k\}$ and $g : \mathbb{R} \to \mathbb{R}$ is a logistic function.
- Stack the vectors $\left\{\theta^{(j)\mathsf{T}}\right\}_{j=1}^{k}$ rowwise into $k \times m$ matrix W_{ih} and create vector valued logistic function $\vec{g} : \mathbb{R}^k \to \mathbb{R}^k$.
- Then we can form vector-valued function $\vec{h} : \mathbb{R}^m \to \mathbb{R}^k$ defined as $\vec{h}(x) = \vec{g}(W_{\text{ih}} \cdot x)$ which does all of the h outputs at simultaneously.
- $\vec{g}(W_{\text{ih}} \cdot x)$ represents matrix vector multiply $W_{\text{ih}} \cdot x$ followed by applying logistic function to all elements of the vector result.
- One final set of parameters $\theta \in \mathbb{R}^k$ to take a weighted sum of each of the h outputs to give $y = \vec{g}(\theta^{\mathsf{T}} \vec{h}(x))$.
- Expanding and being explicit about bias, renaming θ as W_{ho}, we get the classic expression for an MLP:

$$y = \vec{g}(W_{\text{ho}} \cdot \vec{g}(W_{\text{ih}} \cdot x + b_{\text{h}}) + b_{\text{o}}) \tag{10.48}$$

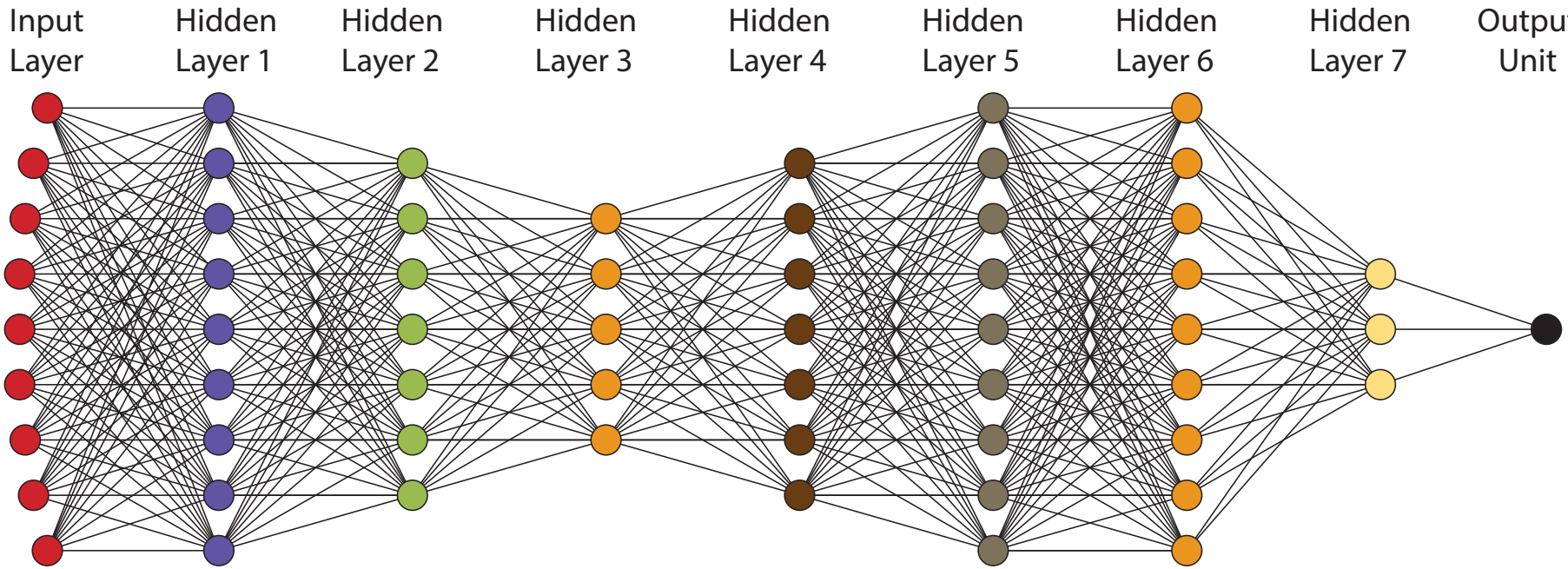
Multi-layered Perceptrons

$$y = \vec{g}(W_{\text{ho}} \cdot \vec{g}(W_{\text{ih}} \cdot x + b_{\text{h}}) + b_{\text{o}}) \tag{10.49}$$



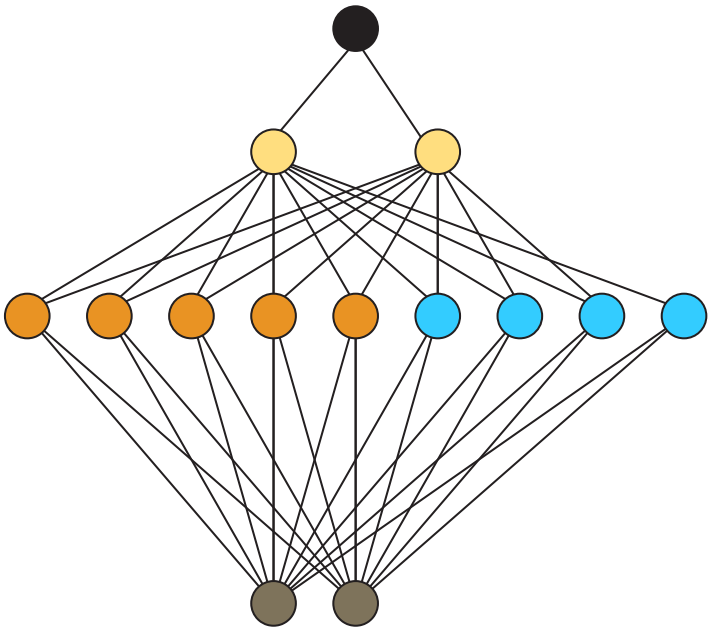
Deep Neural Network (DNN)

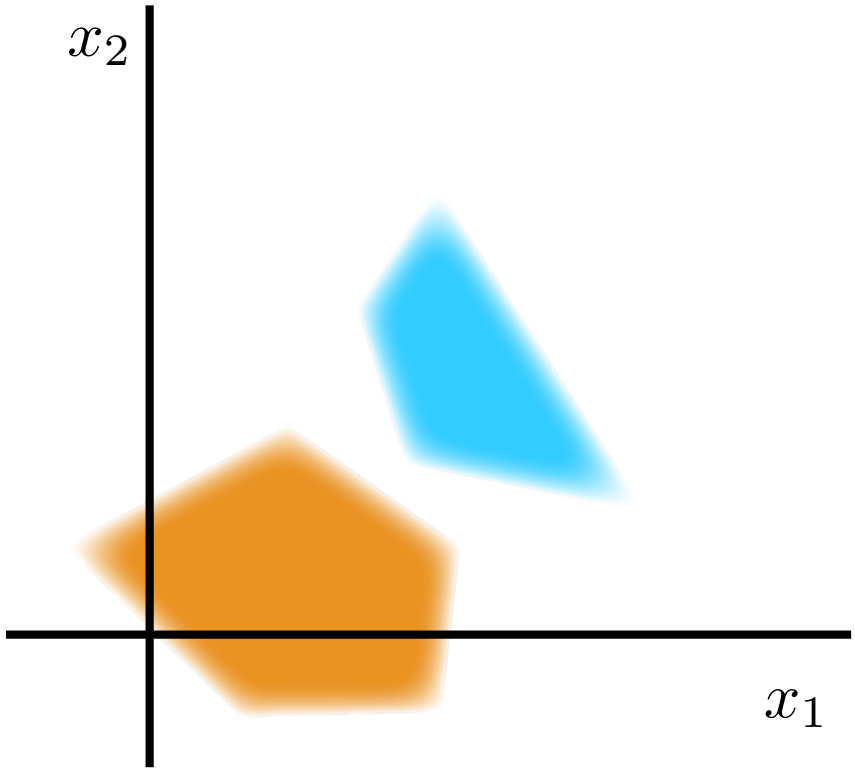
- Most simply, a DNN is a many-layered perceptron: successive layers of matrix-vector multiply followed by a non-linearity. Parameters consist of all of the weight matrices. Modern networks can have hundreds of layers and hundreds of millions of parameters, which can be very expressive and parsimonious. Given an input x, computing y is a series of matrix-vector multiplies followed by application of the non-linearity (called forwardpropagation).



Deep Neural Network (DNN)

- Key advantage: efficient parsimonious expressivity (i.e., easy to get complex decision boundaries) and distributed representations (complex features can be re-used multiple times).





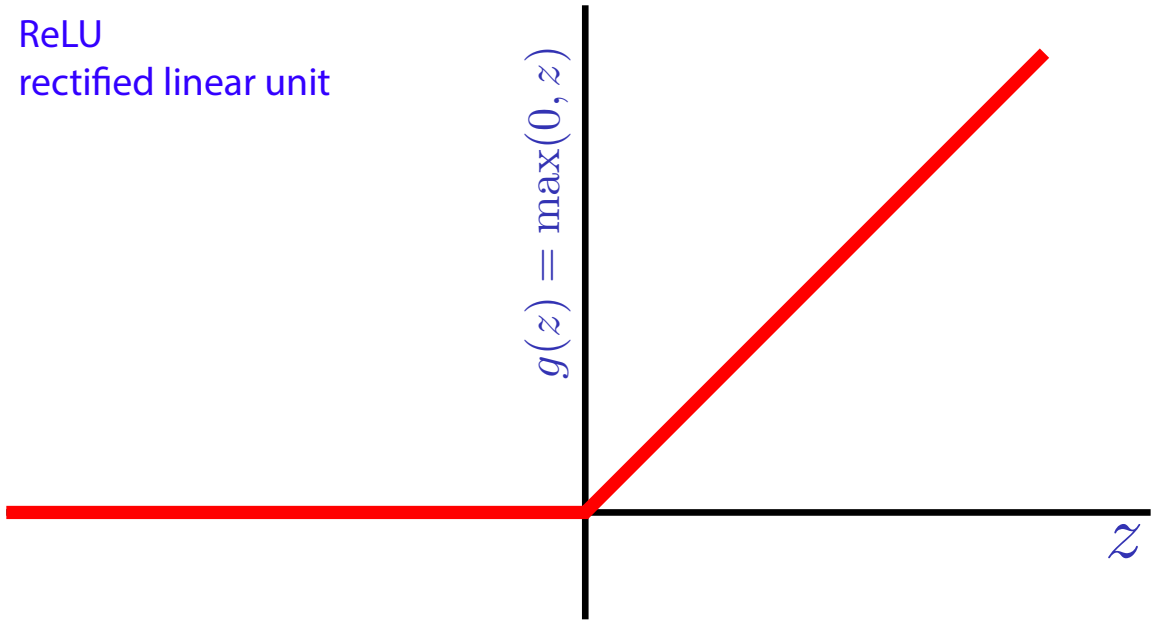
Non-linearities

- The logistic $g(z) = 1/(1 + \exp(-z)$ was traditionally the unit to use in an MLP's hidden layer.

Non-linearities

- The logistic $g(z) = 1/(1 + \exp(-z)$ was traditionally the unit to use in an MLP's hidden layer.
- The non-linearity of choice today is the ReLU (rectified linear unit), i.e., $g(z) = \max(0, z)$.

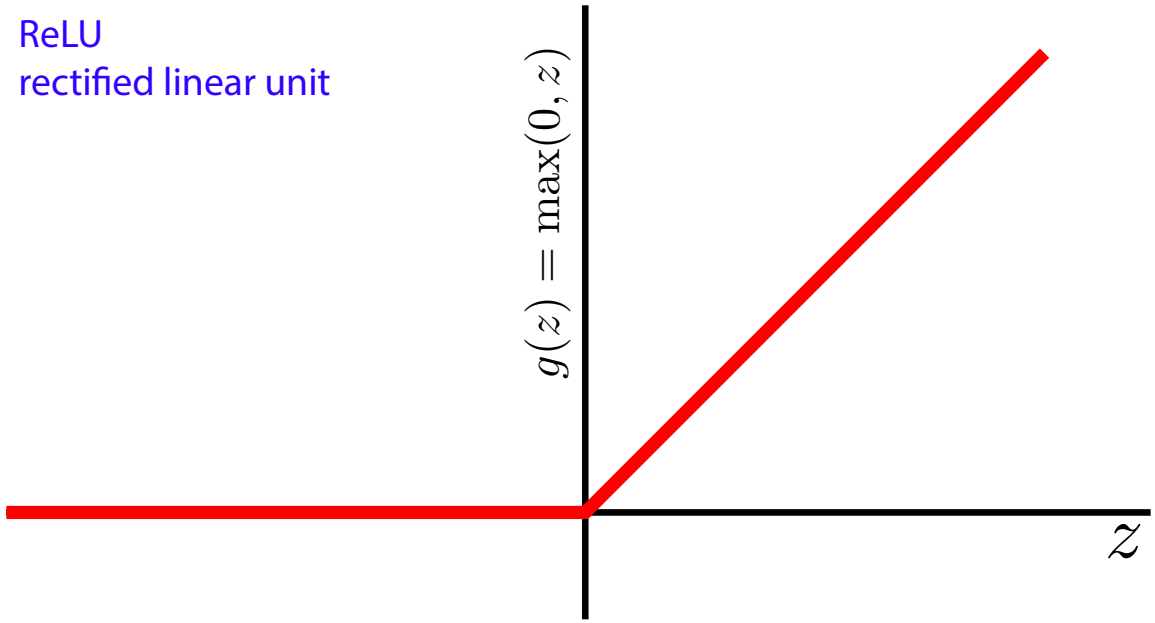
ReLU
rectified linear unit



Non-linearities

- The logistic $g(z) = 1/(1 + \exp(-z)$ was traditionally the unit to use in an MLP's hidden layer.
- The non-linearity of choice today is the ReLU (rectified linear unit), i.e., $g(z) = \max(0, z)$.

ReLU
rectified linear unit



- For the output, typically either linear (for regression problems) or softmax (when we wish the output to be a probability distribution) is used.

Convolutional Neural Networks (CNNs)

- One-dimentional discrete-time signal processing, linear time-invariant systems, FIR filter is convolution.

$$y(t) = (h * x)(t) = \sum_{\tau} h(t-\tau)x(\tau) = \sum_{\tau} h(\tau)x(t-\tau) \quad (10.50)$$

Convolutional Neural Networks (CNNs)

- One-dimentional discrete-time signal processing, linear time-invariant systems, FIR filter is convolution.

$$y(t) = (h * x)(t) = \sum_{\tau} h(t-\tau)x(\tau) = \sum_{\tau} h(\tau)x(t-\tau) \quad (10.50)$$

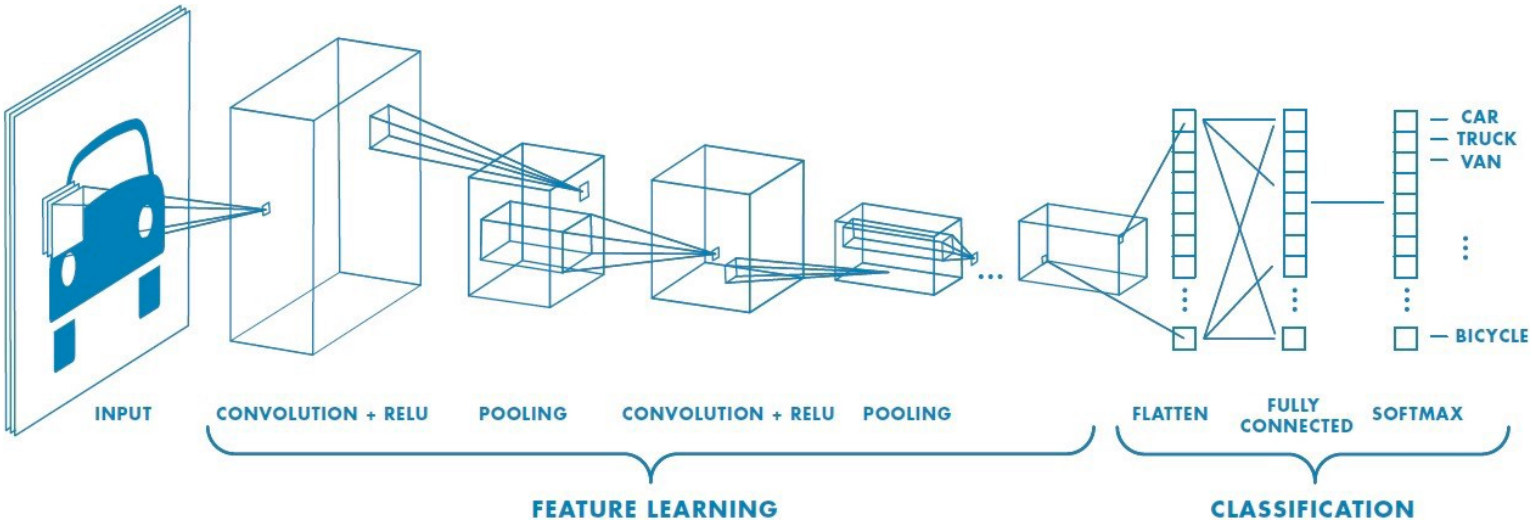
- Matched filters, set h to signal we wish to find (match) in input.

Convolutional Neural Networks (CNNs)

- One-dimentional discrete-time signal processing, linear time-invariant systems, FIR filter is convolution.

$$y(t) = (h * x)(t) = \sum_{\tau} h(t-\tau)x(\tau) = \sum_{\tau} h(\tau)x(t-\tau) \quad (10.50)$$

- Matched filters, set h to signal we wish to find (match) in input.
- CNNs, multi-dimensional, and the the matched filters are plentiful and learnt automatically.

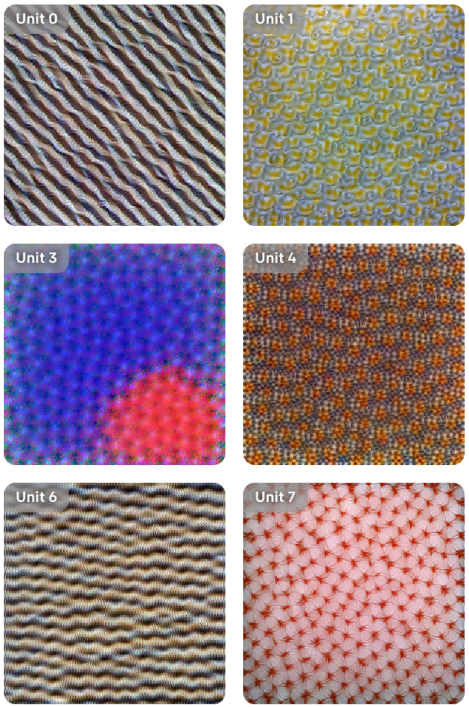


CNN visualation of weights

see https://openai.com/blog/microscope/

Visualiation of Inception v1

Early Layer



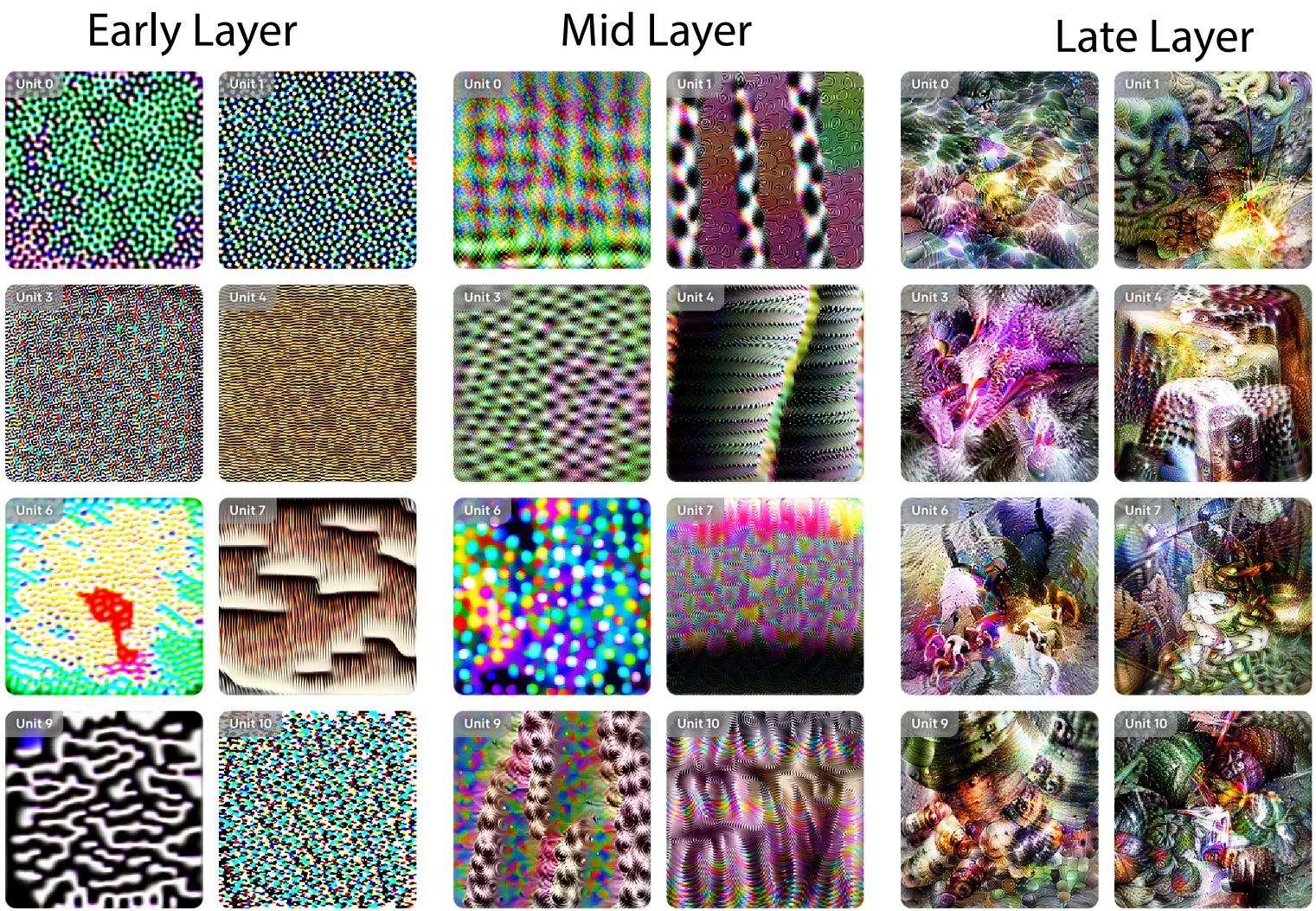
Mid Layer

Late Layer

CNN visualation of weights

see `https://openai.com/blog/microscope/`

Visualiation of ResNet50 v2



Generalized Adversarial Networks (GANs)

- A different way to train, create an adverarial "game" or "competition" between a discriminator D and a generator G.

Generalized Adversarial Networks (GANs)

- A different way to train, create an adverarial "game" or "competition" between a discriminator D and a generator G.
- Discriminator D's job is to be able to tell the difference between real and generated (fake) samples.

Generalized Adversarial Networks (GANs)

- A different way to train, create an adverarial "game" or "competition" between a discriminator D and a generator G.
- Discriminator D's job is to be able to tell the difference between real and generated (fake) samples.
- The generators job is to be able to generate real-looking samples (i.e. generate good fakes) to fool the discriminator.

Generalized Adversarial Networks (GANs)

- A different way to train, create an adverarial "game" or "competition" between a discriminator D and a generator G.
- Discriminator D's job is to be able to tell the difference between real and generated (fake) samples.
- The generators job is to be able to generate real-looking samples (i.e. generate good fakes) to fool the discriminator.
- We have a real data distribution $q(x)$ (which is usually the training set of, say, images).

Generalized Adversarial Networks (GANs)

- A different way to train, create an adverarial "game" or "competition" between a discriminator D and a generator G.
- Discriminator D's job is to be able to tell the difference between real and generated (fake) samples.
- The generators job is to be able to generate real-looking samples (i.e. generate good fakes) to fool the discriminator.
- We have a real data distribution $q(x)$ (which is usually the training set of, say, images).
- Code (or noise) distribution $z \sim p(z)$ to generate codes for the generator.

Generalized Adversarial Networks (GANs)

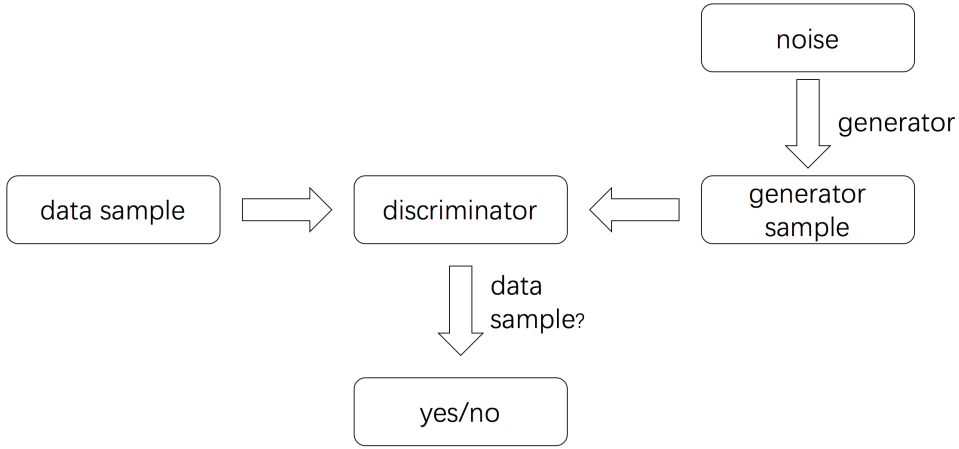
- A different way to train, create an adverarial "game" or "competition" between a discriminator D and a generator G.
- Discriminator D's job is to be able to tell the difference between real and generated (fake) samples.
- The generators job is to be able to generate real-looking samples (i.e. generate good fakes) to fool the discriminator.
- We have a real data distribution $q(x)$ (which is usually the training set of, say, images).
- Code (or noise) distribution $z \sim p(z)$ to generate codes for the generator.
- Generator conditional on code distribution $p(x|z)$.

Generalized Adversarial Networks (GANs)

- A different way to train, create an adverarial "game" or "competition" between a discriminator D and a generator G.
- Discriminator D's job is to be able to tell the difference between real and generated (fake) samples.
- The generators job is to be able to generate real-looking samples (i.e. generate good fakes) to fool the discriminator.
- We have a real data distribution $q(x)$ (which is usually the training set of, say, images).
- Code (or noise) distribution $z \sim p(z)$ to generate codes for the generator.
- Generator conditional on code distribution $p(x|z)$.
- Generated data distribution $p(x) = \int p(x|z)p(z)dz$. We sample $z \sim p(z)$ and then sample $x \sim p(x|z)$.

Generalized Adversarial Networks (GANs)

ideas from Hao Liu's tutorial on Gans



- Training is minimax which sets up the adversarial competition between discriminator and generator:

$$\min_G \max_D V(D,G) \tag{10.51}$$

$$= E_{q(x)}[\log(D(x))] + E_{p(z)}[\log(1 - D(G(z)))] \tag{10.52}$$

$$= \int q(x)\log(D(x))dx + \iint p(z)p(x|z)\log(1 - D(x))dxdz \tag{10.53}$$

Training DNNs

- Conceptually it is no different than before.

Training DNNs

- Conceptually it is no different than before.
- Given data set $\mathcal{D} = \left\{(x^{(i)}, y^{(i)}\right\}_{i=1}^{n}$ of training pairs, we compute:

$$\mathbf{W}^* \in \operatorname*{argmin}_{\mathbf{W}} \sum_{i=1}^{n} L(y^{(i)}, f_{\mathbf{W}}(x^{(i)})) + \lambda \Omega(\mathbf{W}) \tag{10.54}$$

where $\mathbf{W}$ is all of the parameters of the DNN (i.e., all of the weight matrices of every layer).

Training DNNs

- Conceptually it is no different than before.
- Given data set $\mathcal{D} = \left\{(x^{(i)}, y^{(i)}\right\}_{i=1}^{n}$ of training pairs, we compute:

$$\mathbf{W}^* \in \operatorname*{argmin}_{\mathbf{W}} \sum_{i=1}^{n} L(y^{(i)}, f_{\mathbf{W}}(x^{(i)})) + \lambda\Omega(\mathbf{W}) \tag{10.54}$$

where $\mathbf{W}$ is all of the parameters of the DNN (i.e., all of the weight matrices of every layer).

- Training uses stochastic gradient descent (SGD) or various flavors thereof.

$$\mathbf{W}^{(t+1)} \leftarrow \mathbf{W}^{(t)} - \eta_t \nabla_{\mathbf{W}} \left[L(y^{(i_t)}, f_{\mathbf{W}^{(t)}}(x^{(i_t)})) + \lambda\Omega(\mathbf{W}) \right] \tag{10.55}$$

Training DNNs

- Conceptually it is no different than before.
- Given data set $\mathcal{D} = \left\{(x^{(i)}, y^{(i)}\right\}_{i=1}^{n}$ of training pairs, we compute:

$$\mathbf{W}^* \in \operatorname*{argmin}_{\mathbf{W}} \sum_{i=1}^{n} L(y^{(i)}, f_{\mathbf{W}}(x^{(i)})) + \lambda\Omega(\mathbf{W}) \tag{10.54}$$

where $\mathbf{W}$ is all of the parameters of the DNN (i.e., all of the weight matrices of every layer).

- Training uses stochastic gradient descent (SGD) or various flavors thereof.

$$\mathbf{W}^{(t+1)} \leftarrow \mathbf{W}^{(t)} - \eta_t \nabla_{\mathbf{W}} \left[L(y^{(i_t)}, f_{\mathbf{W}^{(t)}}(x^{(i_t)})) + \lambda\Omega(\mathbf{W}) \right] \tag{10.55}$$

- Computing the gradient $\nabla_{\mathbf{W}} \left[L(y^{(i)}, f_{\mathbf{W}^{(t)}}(x^{(i)})) \right]$ is made relatively efficient via the backpropagation algorithm which uses the chain-rule of derivatives to turn the gradient into a series of matrix-vector and/or matrix-matrix calculations, which can be made to run incredibly efficiently on a modern GPU machine.

Review

- The next two slides come from week 7.

On Bayes error and Consistency

- Let h be a classifier, its error (or risk) is $R(h) = \Pr(h(X) \neq Y)$.
- Define $R^* = \min_h \Pr(h(X) \neq Y)$ to be Bayes error.
- $\mathcal{D}_n$ is a training set with n samples, randomly drawn from $p(x, y)$, hence $\mathcal{D}_n$ is a random variable.
- $\theta(\mathcal{D}_n)$ is learning process, mapping from data to learnt model.
- $h_{\theta(\mathcal{D}_n)}$ is the learnt model from training data $\mathcal{D}_n$ also a random variable with a conditional probability of error $\Pr(h_{\theta(\mathcal{D}_n)}(X) \neq Y | \mathcal{D}_n)$.
- Define $R(h_{\theta(\mathcal{D}_n)} | \mathcal{D}_n) = \Pr(h_{\theta(\mathcal{D}_n)}(X) \neq Y | \mathcal{D}_n)$.
- How does $E_{\mathcal{D}_n}\left[R(h_{\theta(\mathcal{D}_n)} | \mathcal{D}_n)\right] = \Pr(h_{\theta(\mathcal{D}_n)}(X) \neq Y)$ improve with n?

Definition 10.7.1 (Consistency)

A classification process is consistent for distribution $p(x, y)$ if $E_{\mathcal{D}_n}[R(h_{\theta(\mathcal{D}_n)} | \mathcal{D}_n)] \to R^*$ as $n \to \infty$.

Definition 10.7.2 (Universal Consistency)

A classification process is universally consistent if it is consistent for all distributions $p(x, y)$.

How far away are we from Bayes error

- Why is consistency and universal consistency a desirable property?
  - As we get more data we know we can't get worse and will approach eventually the best we can ever do, for a given distribution $p(x, y)$.
  - Universality means that we can expect this to be true regardless of the distribution.
- Ideally, we'd like to show that there is a family of classification methods that are universally consistent.
- Wasn't known until relatively recently (in the "stone age"), in 1977 Chuck Stone proved a famous theorem, that the k-NN classifier is universally consistent!
- For a point $x \in \mathbb{R}^m$, let $h_{k\text{-NN},\mathcal{D}_n}(x)$ be the decision decided by the k-NN binary classifier (majority vote over neighbors) on data set $\mathcal{D}_n$ drawn randomly from $p(x, y)$. Stone showed that:

Theorem 10.7.1 (Stone, 1977)

Let $k \to \infty$ with increasing n such that $k/n \to 0$ as $n \to \infty$. Then for all probability distributions $p(x, y)$, $E_{\mathcal{D}_n}[R(h_{k\text{-NN},\mathcal{D}_n})] \to R^*$ as $n \to \infty$.

Other models are universally consistent too, including . . .

- Nearest neighbor classifiers see Stone, 1977

Other models are universally consistent too, including . . .

- Nearest neighbor classifiers see Stone, 1977
- Histogram rule based classifiers see Devroye, Gyorfi, & Lugosi, 1996.

Other models are universally consistent too, including . . .

- Nearest neighbor classifiers see Stone, 1977
- Histogram rule based classifiers see Devroye, Gyorfi, & Lugosi, 1996.
- Support vector machines (SVMs) that use "universal" kernels see "Consistency of Support Vector Machines and Other Regularized Kernel Classifiers", Ingo Steinwart, IEEE TRANSACTIONS ON INFORMATION THEORY, VOL. 51, NO. 1, JANUARY 2005, and "Universal Consistency of Multi-Class Support Vector Classification", Glasmachers, NIPS 2010.

Other models are universally consistent too, including . . .

- Nearest neighbor classifiers see Stone, 1977
- Histogram rule based classifiers see Devroye, Gyorfi, & Lugosi, 1996.
- Support vector machines (SVMs) that use "universal" kernels see "Consistency of Support Vector Machines and Other Regularized Kernel Classifiers", Ingo Steinwart, IEEE TRANSACTIONS ON INFORMATION THEORY, VOL. 51, NO. 1, JANUARY 2005, and "Universal Consistency of Multi-Class Support Vector Classification", Glasmachers, NIPS 2010.
- Neural networks, even shallow ones with only one hidden layer. see "Strong Universal Consistency of Neural Network Classifiers", András Faragó and Gábor Lugosi, IEEE TRANSACTIONS ON INFORMATION THEORY, VOL. 39, NO. 4, JULY 1993

Other models are universally consistent too, including . . .

- Nearest neighbor classifiers see Stone, 1977
- Histogram rule based classifiers see Devroye, Gyorfi, & Lugosi, 1996.
- Support vector machines (SVMs) that use "universal" kernels see "Consistency of Support Vector Machines and Other Regularized Kernel Classifiers", Ingo Steinwart, IEEE TRANSACTIONS ON INFORMATION THEORY, VOL. 51, NO. 1, JANUARY 2005, and "Universal Consistency of Multi-Class Support Vector Classification", Glasmachers, NIPS 2010.
- Neural networks, even shallow ones with only one hidden layer. see "Strong Universal Consistency of Neural Network Classifiers", András Faragó and Gábor Lugosi, IEEE TRANSACTIONS ON INFORMATION THEORY, VOL. 39, NO. 4, JULY 1993
- In practice, we only ever have finite bounded training data sizes . . .

No Free Lunch: Informal

- The "No Free Lunch" property in machine learning, stated very informally, is that no one learning strategy will work in all circumstances.

No Free Lunch: Informal

- The "No Free Lunch" property in machine learning, stated very informally, is that no one learning strategy will work in all circumstances.
- For example, a DNN that works great for one problem and data set might fail for another; a k-NN classifier that fails in one case might be the best strategy in another.

No Free Lunch: Informal

- The "No Free Lunch" property in machine learning, stated very informally, is that no one learning strategy will work in all circumstances.
- For example, a DNN that works great for one problem and data set might fail for another; a k-NN classifier that fails in one case might be the best strategy in another.
- Name comes from "There ain't no such thing as a free lunch", in the 1930s, free (salty) lunch with a purchased drink which prompts you to buy another drink, thereby leading to more profits for the seller.

No Free Lunch: Simple Example

From "Toward a Justification of Meta-learning: Is the No Free Lunch Theorem a Show-stopper?", by Giruad-Carrier, Provost.

- Let $\mathcal{D}_{\text{tr}} = \{(x^{(i)}, y^{(i)}\}_{i=1}^{n_{\text{tr}}}$ be a training set and $\mathcal{D}_{\text{te}} = \{(x^{(i)}, y^{(i)}\}_{i=n_{\text{tr}}+1}^{n}$ be a test set where $n = n_{\text{tr}} + n_{\text{te}}$. Note that n is finite.

No Free Lunch: Simple Example

From "Toward a Justification of Meta-learning: Is the No Free Lunch Theorem a Show-stopper?", by Giruad-Carrier, Provost.

- Let $\mathcal{D}_{\text{tr}} = \{(x^{(i)}, y^{(i)}\}_{i=1}^{n_{\text{tr}}}$ be a training set and $\mathcal{D}_{\text{te}} = \{(x^{(i)}, y^{(i)}\}_{i=n_{\text{tr}}+1}^{n}$ be a test set where $n = n_{\text{tr}} + n_{\text{te}}$. Note that n is finite.
- For the simple example, let $x^{(i)} \in \{0,1\}^3$, $y^{(i)} \in \{0,1\}$, and $n_{\text{tr}} = 6$. Hence, there are six training samples and two test samples, eight samples overall, and $2^8 = 256$ ways of labeling these eight samples.

No Free Lunch: Simple Example

From "Toward a Justification of Meta-learning: Is the No Free Lunch Theorem a Show-stopper?", by Giruad-Carrier, Provost.

- Let $\mathcal{D}_{\text{tr}} = \{(x^{(i)}, y^{(i)}\}_{i=1}^{n_{\text{tr}}}$ be a training set and $\mathcal{D}_{\text{te}} = \{(x^{(i)}, y^{(i)}\}_{i=n_{\text{tr}}+1}^{n}$ be a test set where $n = n_{\text{tr}} + n_{\text{te}}$. Note that n is finite.
- For the simple example, let $x^{(i)} \in \{0,1\}^3$, $y^{(i)} \in \{0,1\}$, and $n_{\text{tr}} = 6$. Hence, there are six training samples and two test samples, eight samples overall, and $2^8 = 256$ ways of labeling these eight samples.
- Let $f_1, f_2, \ldots, f_{256}$ be the 256 set of possible labeling functions. Any given data set has $y^{(i)} = f_j(x^{(i)})$ for all i and for some j.

		Inputs			f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}	...
Training Set	$x^{(1)}$	0	0	0	0	0	0	0	0	0	0	0	0	0	...
	$x^{(2)}$	0	0	1	0	0	0	0	0	0	0	0	0	0	...
	$x^{(3)}$	0	1	0	0	0	0	0	0	0	0	0	0	0	...
	$x^{(4)}$	0	1	1	0	0	0	0	0	0	0	0	0	0	...
	$x^{(5)}$	1	0	0	0	0	0	0	0	0	0	0	1	1	...
	$x^{(6)}$	1	0	1	0	0	0	0	1	1	1	1	0	0	...
Test Set	$x^{(7)}$	1	1	0	0	0	1	1	0	0	1	1	0	0	...
	$x^{(8)}$	1	1	1	0	1	0	1	0	1	0	1	0	1	...

No Free Lunch: Simple Example

- In the example, for each possible training set there are four possible f_j's that could have generated it. No matter how well we train, if we do not train on the test set, we will choose one of the models.

No Free Lunch: Simple Example

- In the example, for each possible training set there are four possible f_j's that could have generated it. No matter how well we train, if we do not train on the test set, we will choose one of the models.
- No free lunch theorem says that if the differnet f_j's are chosen uniformly at random, then any trained model on one of the training sets will perform no better than random over all possible true models.

No Free Lunch: Simple Example

- In the example, for each possible training set there are four possible f_j's that could have generated it. No matter how well we train, if we do not train on the test set, we will choose one of the models.
- No free lunch theorem says that if the differnet f_j's are chosen uniformly at random, then any trained model on one of the training sets will perform no better than random over all possible true models.
- For example, if we learn f_1 then the performance will be the average: $(1 + 0.5 + 0.5 + 0)/4 = 0.5$, or 50%.

No Free Lunch: Simple Example

- In the example, for each possible training set there are four possible f_j's that could have generated it. No matter how well we train, if we do not train on the test set, we will choose one of the models.
- No free lunch theorem says that if the differnet f_j's are chosen uniformly at random, then any trained model on one of the training sets will perform no better than random over all possible true models.
- For example, if we learn f_1 then the performance will be the average: $(1 + 0.5 + 0.5 + 0)/4 = 0.5$, or 50%.
- In general, this means there is no universal learner. Every learner has to have some prior knowledge about a task

No Free Lunch: More formally, setup

- Given random size-n data set $\mathcal{D}_n = \left\{(X^{(i)}, Y^{(i)}\right\}_{i=1}^{n}$, let f_n be any learning algorithm that learns from this data set, and $f_n(X, \mathcal{D})$ be a prediction on a new X.

No Free Lunch: More formally, setup

- Given random size-n data set $\mathcal{D}_n = \left\{(X^{(i)}, Y^{(i)}\right\}_{i=1}^{n}$, let f_n be any learning algorithm that learns from this data set, and $f_n(X, \mathcal{D})$ be a prediction on a new X.
- Note, $f_n(X, \mathcal{D})$ is an learning algorithm (which might or might not itself be random) applied to a random data set $\mathcal{D}$ and whose resulting model is applied to a random sample X.

No Free Lunch: More formally, setup

- Given random size-n data set $\mathcal{D}_n = \left\{(X^{(i)}, Y^{(i)}\right\}_{i=1}^n$, let f_n be any learning algorithm that learns from this data set, and $f_n(X, \mathcal{D})$ be a prediction on a new X.
- Note, $f_n(X, \mathcal{D})$ is an learning algorithm (which might or might not itself be random) applied to a random data set $\mathcal{D}$ and whose resulting model is applied to a random sample X.
- Error probability is $L_n = \Pr(Y \neq f_n(X, \mathcal{D}_n)|\mathcal{D}_n)$ and average error is $EL_n = \Pr(Y \neq f_n(X, \mathcal{D}_n))$

No Free Lunch: More formally, setup

- Given random size-n data set $\mathcal{D}_n = \left\{(X^{(i)}, Y^{(i)}\right\}_{i=1}^n$, let f_n be any learning algorithm that learns from this data set, and $f_n(X, \mathcal{D})$ be a prediction on a new X.
- Note, $f_n(X, \mathcal{D})$ is an learning algorithm (which might or might not itself be random) applied to a random data set $\mathcal{D}$ and whose resulting model is applied to a random sample X.
- Error probability is $L_n = \Pr(Y \neq f_n(X, \mathcal{D}_n)|\mathcal{D}_n)$ and average error is $EL_n = \Pr(Y \neq f_n(X, \mathcal{D}_n))$
- Universal consistency would say, of a given learning procedure, that $\lim_{n\to\infty} EL_n = L^*$ where L^* is the Bayes classifer's error probability (i.e. having Bayes risk), and this would be true for all possible distributions.

No Free Lunch: More formally

From Devroye, Gyorfi, & Lugosi, 1996.

- No Free Lunch theorem essentially says the following.

No Free Lunch: More formally

From Devroye, Gyorfi, & Lugosi, 1996.

- No Free Lunch theorem essentially says the following.

Theorem 10.7.1 (Devroye 1982)

Let $\epsilon > 0$ be arbitrarily small. For any fixed integer n, and any classification rule (e.g. machine learning procedure) f_n, there exists a distribution $\Pr(x, y)$ with Bayes risk $L^* = 0$ such that

$$EL_n \geq 1/2 - \epsilon \tag{10.56}$$

No Free Lunch: More formally

From Devroye, Gyorfi, & Lugosi, 1996.

- No Free Lunch theorem essentially says the following.

Theorem 10.7.1 (Devroye 1982)

Let $\epsilon > 0$ be arbitrarily small. For any fixed integer n, and any classification rule (e.g. machine learning procedure) f_n, there exists a distribution $\Pr(x, y)$ with Bayes risk $L^* = 0$ such that

$$EL_n \geq 1/2 - \epsilon \tag{10.56}$$

- Thus, even though there are methods that are universally consistent (meaning they asymptotically give optimal performance for any distribution), their finite sample performance is always very bad for some distributions.

No Free Lunch: More formally

From Devroye, Gyorfi, & Lugosi, 1996.

- No Free Lunch theorem essentially says the following.

Theorem 10.7.1 (Devroye 1982)

Let $\epsilon > 0$ be arbitrarily small. For any fixed integer n, and any classification rule (e.g. machine learning procedure) f_n, there exists a distribution $\Pr(x, y)$ with Bayes risk $L^* = 0$ such that

$$EL_n \geq 1/2 - \epsilon \tag{10.56}$$

- Thus, even though there are methods that are universally consistent (meaning they asymptotically give optimal performance for any distribution), their finite sample performance is always very bad for some distributions.
- For the "no free lunch", we fix the training set size and then find a distribution and labeling that are bad. With a consistency guarantee, on the other hand, we fix the distribution (and labeling), and find the training set size that gets as close as we want to Bayes error.

On the Inexactitude of models

All models are wrong, but some are useful.
—George Box (Box and Draper 1987)

On the Inexactitude of models

All models are wrong, but some are useful.
—George Box (Box and Draper 1987)

Art is a lie that makes us realize truth, at least the truth that is given us to understand.
—Pablo Picasso, 1923

On the Inexactitude of models

All models are wrong, but some are useful.
—George Box (Box and Draper 1987)

Art is a lie that makes us realize truth, at least the truth that is given us to understand.
—Pablo Picasso, 1923

"...it has been said that democracy is the worst form of Government except for all those other forms that have been tried from time to time"
—Winston S Churchill, 11 November 1947

On the Inexactitude of models

All models are wrong, but some are useful.
—George Box (Box and Draper 1987)

Art is a lie that makes us realize truth, at least the truth that is given us to understand.
—Pablo Picasso, 1923

"...it has been said that democracy is the worst form of Government except for all those other forms that have been tried from time to time"
—Winston S Churchill, 11 November 1947

"the map is not the thing mapped"
— Eric Temple Bell, Numerology: The Magic of Numbers, 1933

The "Bayes Error" of Cartography

"What a useful thing a pocket-map is!" I remarked.

"That's another thing we've learned from your Nation," said Mein Herr, "map-making. But we've carried it much further than you. What do you consider the largest map that would be really useful?"

"About six inches to the mile."

"Only six inches!" exclaimed Mein Herr. "We very soon got to six yards to the mile. Then we tried a hundred yards to the mile. And then came the grandest idea of all ! We actually made a map of the country, on the scale of a mile to the mile!"

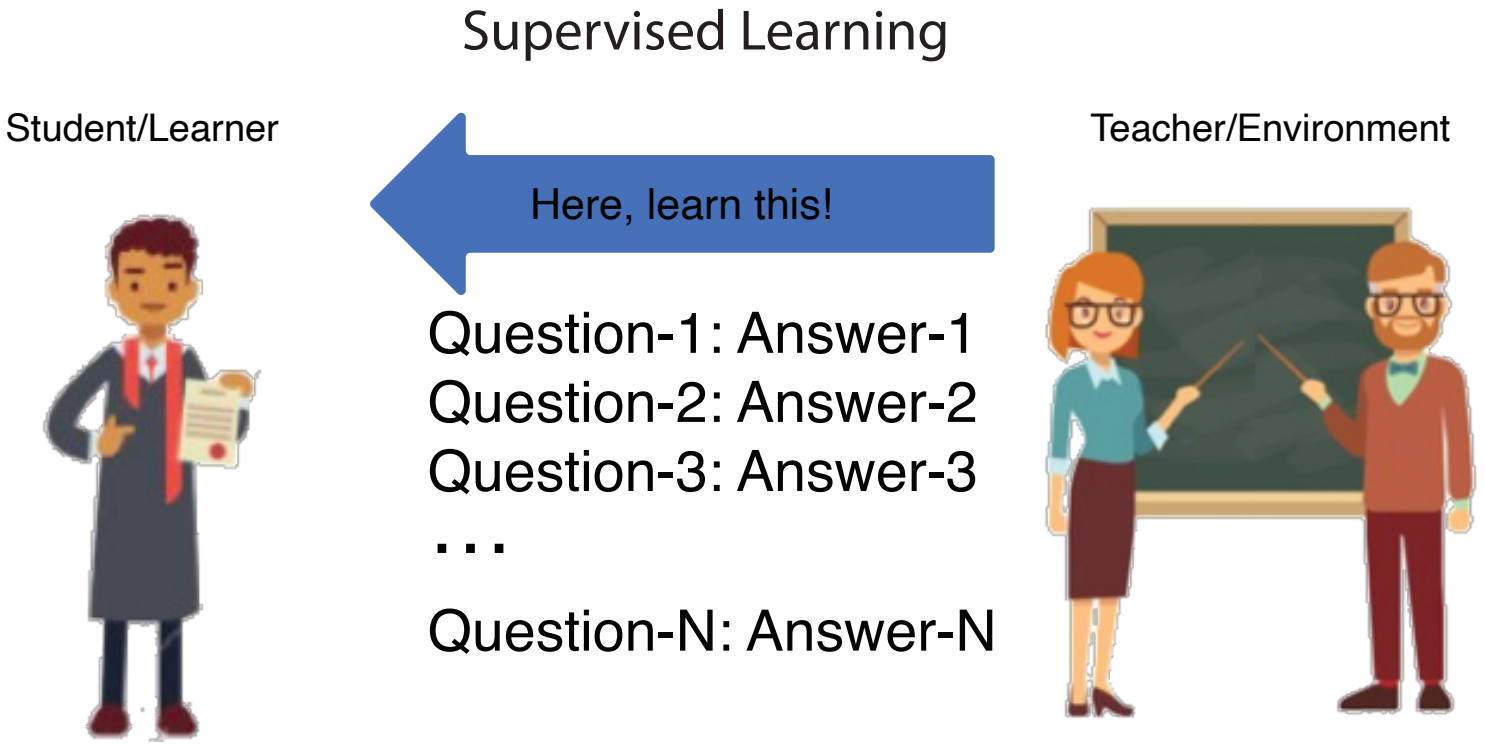
"Have you used it much?" I enquired.

"It has never been spread out, yet," said Mein Herr: "the farmers objected: they said it would cover the whole country, and shut out the sunlight! So we now use the country itself, as its own map, and I assure you it does nearly as well."

— Lewis Carroll, Sylvie and Bruno Concluded, Chapter XI, London, 1895

Supervised Learning

- We've studied mostly supervised learning, where the data set $\mathcal{D} = \left\{\left(x^{(i)}, y^{(i)}\right)\right\}_{i=1}^{n}$ represents the associations we wish to learn, and we want a mapping $h(x) = y$ that is accurate.
- For each x, the answer y is given, and the learner expected, given a large enough data set, to learn the appropriate association.
- From student/teacher perspective, looks like:



Unsupervised learning

- Is it possible to learn to associate from x to y given only the inputs?
- I.e., data set consists of $\mathcal{D} = \left\{x^{(i)}\right\}_{i=1}^{n}$. We need to figure out the y's for each x and learn to do the association and generalize. Is this possible?
- From student/teacher perspective, looks like:

Unsupervised Learning

Student/Learner

Here, learn this!

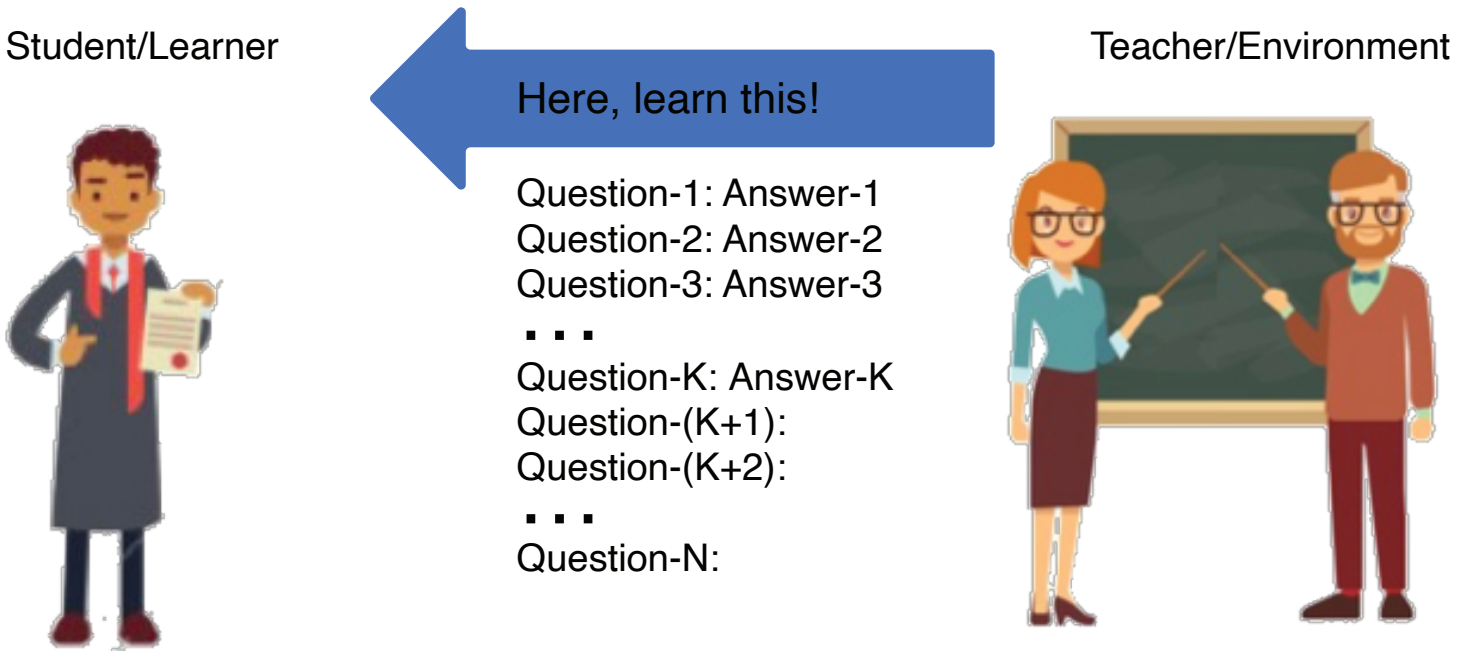
Question-1:
Question-2:
Question-3:
...
Question-N:

Teacher/Environment

Semi-supervised learning

- Is it possible to learn to associate from x to y given only some of the outputs?
- I.e., data set consists of $\mathcal{D} = \left\{(x^{(i)}, y^{(i)})\right\}_{i=1}^{\ell} \cup \left\{(x^{(i)})\right\}_{i=\ell+1}^{n}$. We need to figure out the y's for each x and learn to do the association, using both labeled and unlabeled data.
- From student/teacher perspective, looks like:

Semi-supervised learning



Self-supervised learning

- Unsupervised learning where data provides the supervision.

Self-supervised learning

- Unsupervised learning where data provides the supervision.
- Withhold some of the data, ask the model to predict it.

Self-supervised learning

- Unsupervised learning where data provides the supervision.
- Withhold some of the data, ask the model to predict it.
- Model learns semantic representation to solve the task.

Self-supervised learning

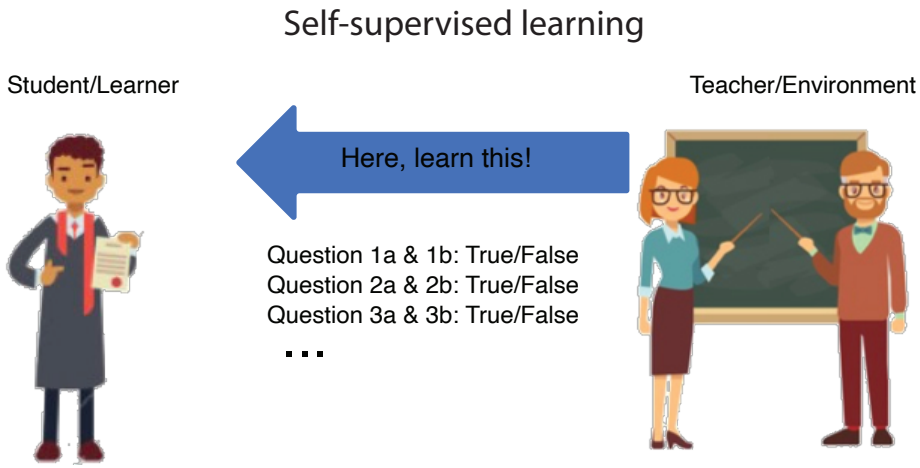
- Unsupervised learning where data provides the supervision.
- Withhold some of the data, ask the model to predict it.
- Model learns semantic representation to solve the task.
- Example word2vec, predict a word's nearby context and use resulting representation. Data consists of pairs of (word, context). With negative data, pairs are augmented with a true/false label.

Self-supervised learning

- Unsupervised learning where data provides the supervision.
- Withhold some of the data, ask the model to predict it.
- Model learns semantic representation to solve the task.
- Example word2vec, predict a word's nearby context and use resulting representation. Data consists of pairs of (word, context). With negative data, pairs are augmented with a true/false label.
- Images: given two image patches, predict relative position.

Self-supervised learning

- Unsupervised learning where data provides the supervision.
- Withhold some of the data, ask the model to predict it.
- Model learns semantic representation to solve the task.
- Example word2vec, predict a word's nearby context and use resulting representation. Data consists of pairs of (word, context). With negative data, pairs are augmented with a true/false label.
- Images: given two image patches, predict relative position.
- From student/teacher perspective:



Active Learning

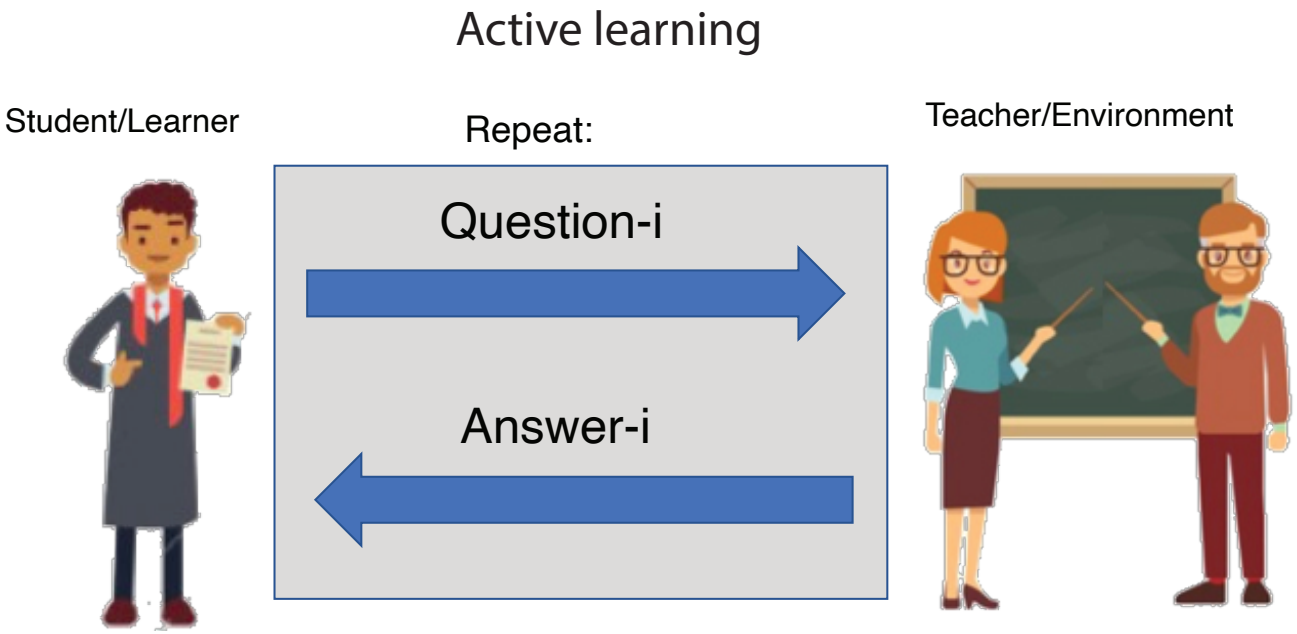
- Why should a learner not express her/his curiosity, uncertainty, in deciding what data it should next learn from, rather than passively accepting any and all data that comes its way (passive learning)?

Active Learning

- Why should a learner not express her/his curiosity, uncertainty, in deciding what data it should next learn from, rather than passively accepting any and all data that comes its way (passive learning)?
- This is a form of interactive learning, where we ask the learner what it would like next to learn. Learn by answering questions.

Active Learning

- Why should a learner not express her/his curiosity, uncertainty, in deciding what data it should next learn from, rather than passively accepting any and all data that comes its way (passive learning)?
- This is a form of interactive learning, where we ask the learner what it would like next to learn. Learn by answering questions.
- From student/teacher perspective:



Machine Teaching

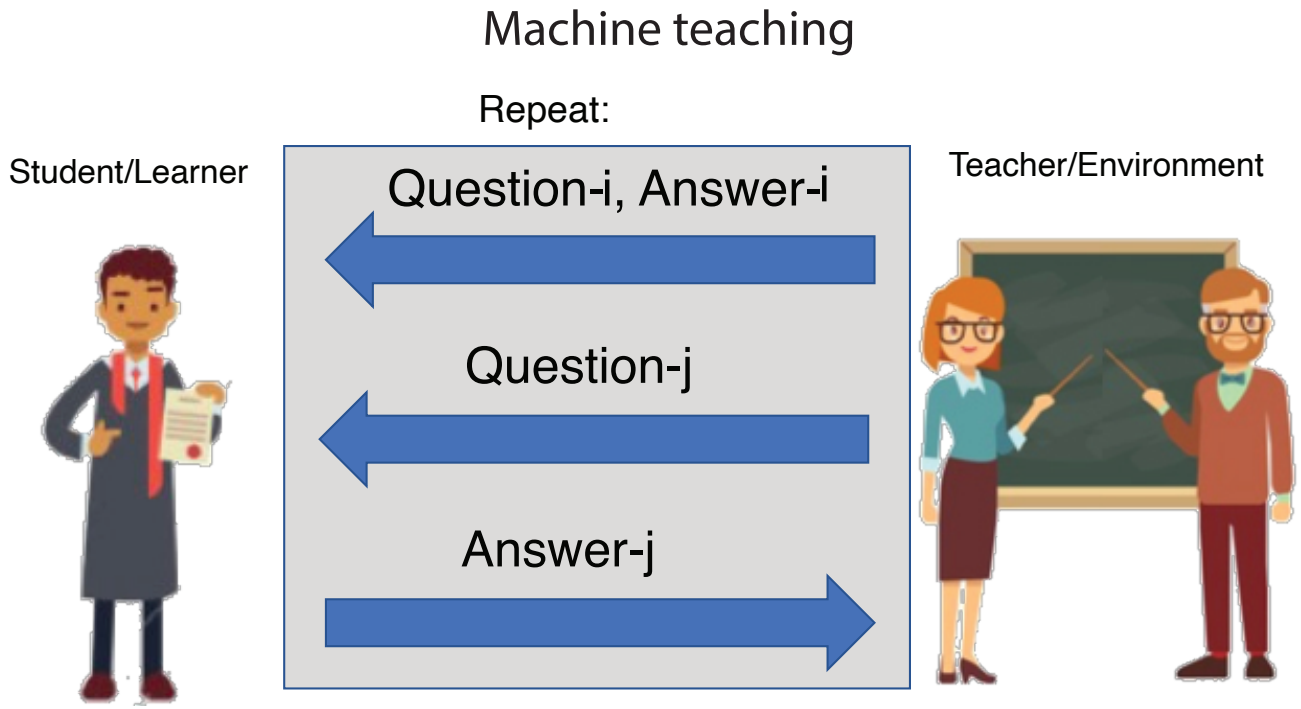
- Why should teacher not be able to express her/his ability to teach, deciding what next is best for the learner to learn.

Machine Teaching

- Why should teacher not be able to express her/his ability to teach, deciding what next is best for the learner to learn.
- This also is a form of interactive learning, where we ask a (hopefully smart) teacher what best should be learnt next. curriculum learning.

Machine Teaching

- Why should teacher not be able to express her/his ability to teach, deciding what next is best for the learner to learn.
- This also is a form of interactive learning, where we ask a (hopefully smart) teacher what best should be learnt next. curriculum learning.
- From student/teacher perspective:



Multi-task/transfer learning

- Multiple learners can learn related but not identical tasks simultaneously and share their experiences with each other.

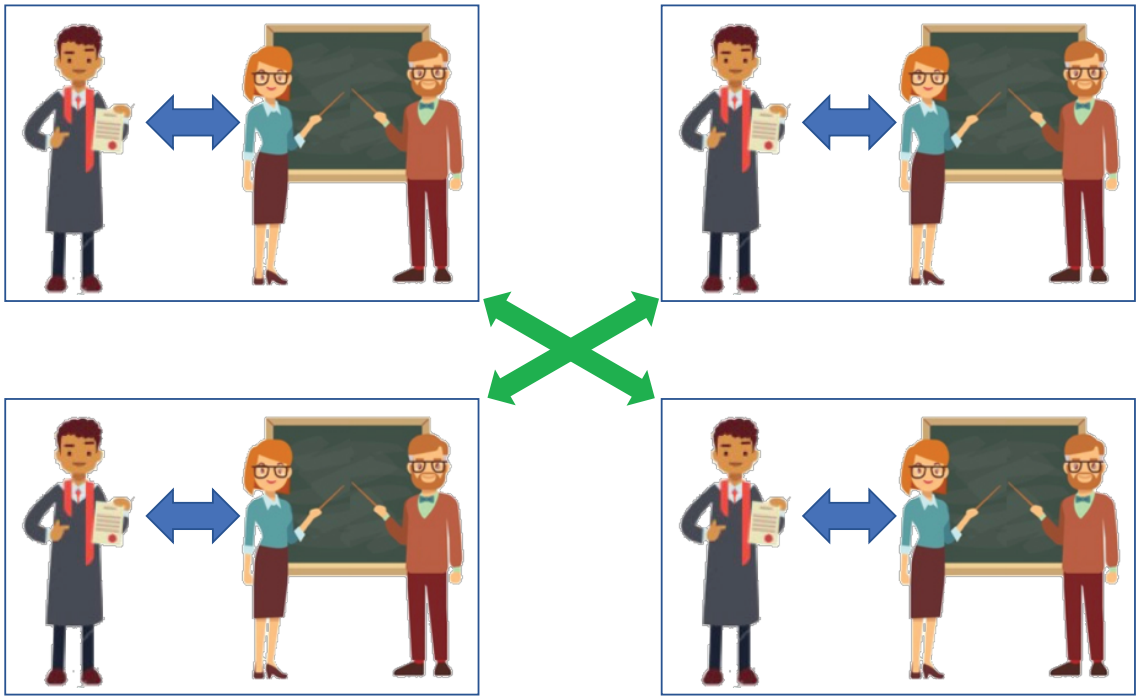
Multi-task/transfer learning

- Multiple learners can learn related but not identical tasks simultaneously and share their experiences with each other.
- Ex: speech recognition, low-level learnt speech representations seem to be trans-lingual, and work over all languages, before higher level language-specific information needs to be incorporated. Useful for languages w/o much training data.

Multi-task/transfer learning

- Multiple learners can learn related but not identical tasks simultaneously and share their experiences with each other.
- Ex: speech recognition, low-level learnt speech representations seem to be trans-lingual, and work over all languages, before higher level language-specific information needs to be incorporated. Useful for languages w/o much training data.
- From student/teacher perspective:

Multi-task, transfer, and ensemble learning



Federated learning

- Multiple distributed learners learn only part of the task.

Federated learning

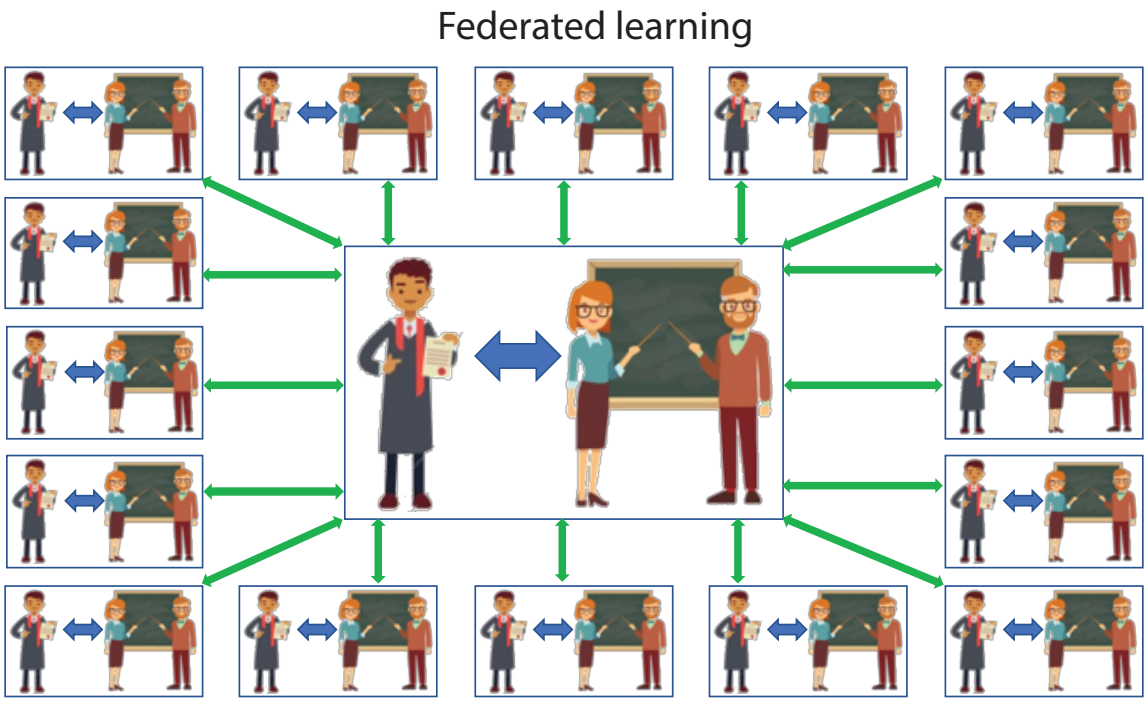
- Multiple distributed learners learn only part of the task.
- For privacy reasons, learners only transfer aggregate information about what they've learnt to centralized learner, who aggregates the aggregates, and passes learnt model back to the individual learners.

Federated learning

- Multiple distributed learners learn only part of the task.
- For privacy reasons, learners only transfer aggregate information about what they've learnt to centralized learner, who aggregates the aggregates, and passes learnt model back to the individual learners.
- Local data (e.g., on a smart phone) never in raw form never becomes non-local.

Federated learning

- Multiple distributed learners learn only part of the task.
- For privacy reasons, learners only transfer aggregate information about what they've learnt to centralized learner, who aggregates the aggregates, and passes learnt model back to the individual learners.
- Local data (e.g., on a smart phone) never in raw form never becomes non-local.
- From student/teacher perspective:



Other Paradigms

- Meta-learning: Learning how to assign a given learning algorithm to a task. I.e. given a data set, learn how to assign what learning method (e.g., k-NN classifier, logistic, DNN and its specification, GBDT, etc.) would be best for the given problem.

Other Paradigms

- Meta-learning: Learning how to assign a given learning algorithm to a task. I.e. given a data set, learn how to assign what learning method (e.g., k-NN classifier, logistic, DNN and its specification, GBDT, etc.) would be best for the given problem.
- Note there is not yet any mathematically formal universally agreed upon definition of meta learning, but this is the idea.

Other Paradigms

- Meta-learning: Learning how to assign a given learning algorithm to a task. I.e. given a data set, learn how to assign what learning method (e.g., k-NN classifier, logistic, DNN and its specification, GBDT, etc.) would be best for the given problem.
- Note there is not yet any mathematically formal universally agreed upon definition of meta learning, but this is the idea.
- Transductive learning: Data set consists of $\mathcal{D} = \left\{\left(x^{(i)}, y^{(i)}\right)\right\}_{i=1}^{\ell} \cup \left\{\left(x^{(i)}\right)\right\}_{i=\ell+1}^{n}$. We need to figure out the y's for each x only in the unlabeled data set, and then nothing more.

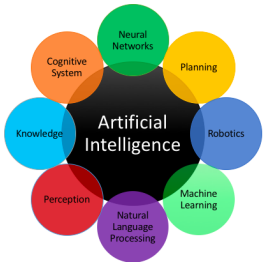
Other Paradigms

- Meta-learning: Learning how to assign a given learning algorithm to a task. I.e. given a data set, learn how to assign what learning method (e.g., k-NN classifier, logistic, DNN and its specification, GBDT, etc.) would be best for the given problem.
- Note there is not yet any mathematically formal universally agreed upon definition of meta learning, but this is the idea.
- Transductive learning: Data set consists of $\mathcal{D} = \left\{(x^{(i)}, y^{(i)})\right\}_{i=1}^{\ell} \cup \left\{(x^{(i)})\right\}_{i=\ell+1}^{n}$. We need to figure out the y's for each x only in the unlabeled data set, and then nothing more.
- Example: Statistical machine Translation of documents (e.g., camera manual given a large corpora of bi-text).

Other Paradigms

- Meta-learning: Learning how to assign a given learning algorithm to a task. I.e. given a data set, learn how to assign what learning method (e.g., k-NN classifier, logistic, DNN and its specification, GBDT, etc.) would be best for the given problem.
- Note there is not yet any mathematically formal universally agreed upon definition of meta learning, but this is the idea.
- Transductive learning: Data set consists of $\mathcal{D} = \left\{\left(x^{(i)}, y^{(i)}\right)\right\}_{i=1}^{\ell} \cup \left\{\left(x^{(i)}\right)\right\}_{i=\ell+1}^{n}$. We need to figure out the y's for each x only in the unlabeled data set, and then nothing more.
- Example: Statistical machine Translation of documents (e.g., camera manual given a large corpora of bi-text).
- Transductive learning is distinct from semi-supervised learning which is an inductive learning procedure, where we induce a model that can map from x to y not just in the (labeled and unlabeled) training set but also beyond (i.e. generalization).

Class (and Machine Learning) overview

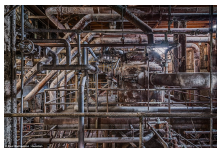


1. Introduction
- What is ML
- What is AI
- Why are we so interested in these topics right now?

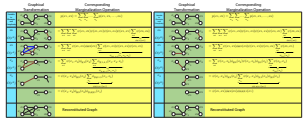
2. ML Paradigms/Concepts
- Overfitting/Underfitting, model complexity, bias/variance
- size of data, big data, sample complexity
- ERM, loss + regularization, loss functions, regularizers
- Supervised, unsupervised, and semi-supervised learning;
- reinforcement learning, RL, multi-agent, planning/control
- transfer and multi-task learning
- federated and distributed learning
- active learning, machine teaching
- self-supervised, zero/one-shot, open-set learning

3. Dealing with Features
- dimensionality reduction, PCA, LDA, MDS, T-SNE, UMAP
- Locality sensitive hashing (LSH)
- feature selection
- feature engineering
- matrix factorization & feature engineering
- representation learning

4. Evaluation
- accuracy/error, precision/recall, ROC, likelihood/posterior, cost/utility, margin
- train/eval/test data splits
- n-fold cross validation
- method of the bootstrap



5. Optimization Methods
- Unconstrained Continuous Optimization: (stochastic) gradient descent (SGD), adaptive learning rates, conjugate gradient, 2nd order Newton
- Constrained Continuous Optimization : Frank-Wolf (conditional gradient descent), projected gradient, linear, quadratic, and convex programming
- Discrete optimization - greedy, beam search, branch-and-bound, submodular optimization.



6. Inference Methods
- probabilistic inference
- MLE, MAP
- belief propagation
- forward/backpropagation
- Monte Carlo methods

7. Models & Representation
- linear least squares, linear regression, logistic regression, sparsity, ridge, lasso
- generative vs. discriminative models
- Naive Bayes
- k-nearest neighbors
- clustering, k-means, k-mediods, EM & GMMs, single linkage
- decision trees and random forests
- support vector machines, kernel methods, max margin
- perceptron, neural networks, DNNs
- Gaussian processes
- Bayesian nonparametric methods
- ensemble methods
- the bootstrap, bagging, and boosting
- graphical models
- time-series, HMMs, DBNs, RNNs, LSTMs, Attention, Transformers
- structured prediction
- grammars (as in NLP)

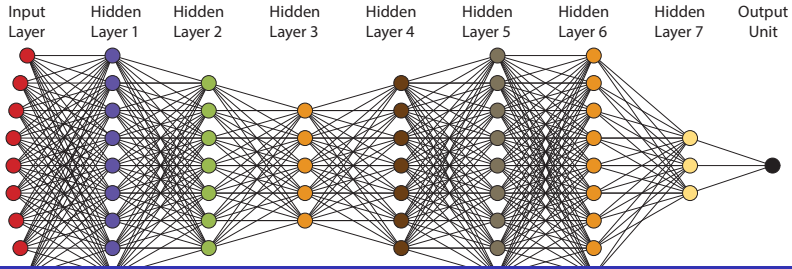
8. Philosophy, Humanity, Spirituality
- artificial intelligence (AI)
- artifical general intelligence (AGI)
- artificial intelligence vs. science fiction

9. Applications
- computational biology
- social networks
- computer vision
- speech recognition
- natural language processing
- information retrieval
- collaborative filtering/matrix factorization

10. Programming
- python
- libraries (e.g., NumPy, SciPy, matplotlib, scikit-learn (sklearn), pytorch, CNTK, Theano, tensorflow, keras, H2O, etc.
- HPC: C/C++, CUDA, vector processing

11. Background
- linear algebra
- multivariate calculus
- probability theory and statistics
- information theory
- mathematical (e.g., convex) optimization

12. Other Techniques
- compressed sensing
- submodularity, diversity/homogeneity modeling



1.2 Determinant

Let $\mathbf{A}$ be an $n \times n$ matrix.

$$\begin{aligned}
\det(\mathbf{A}) &= \textstyle\prod_i \lambda_i \quad \lambda_i = \mathrm{eig}(\mathbf{A}) \\
\det(c\mathbf{A}) &= c^n \det(\mathbf{A}), \quad \text{if } \mathbf{A} \in \mathbb{R}^{n\times n} \\
\det(\mathbf{A}^T) &= \det(\mathbf{A}) \\
\det(\mathbf{AB}) &= \det(\mathbf{A})\det(\mathbf{B}) \\
\det(\mathbf{A}^{-1}) &= 1/\det(\mathbf{A}) \\
\det(\mathbf{A}^n) &= \det(\mathbf{A})^n \\
\det(\mathbf{I} + \mathbf{uv}^T) &= 1 + \mathbf{u}^T\mathbf{v}
\end{aligned}$$

The end

The end

Stop Germs! Wash Your Hands.

When?

- After using the bathroom
- Before, during, and after preparing food
- Before eating food
- Before and after caring for someone at home who is sick with vomiting or diarrhea
- After changing diapers or cleaning up a child who has used the toilet
- After blowing your nose, coughing, or sneezing
- After touching an animal, animal feed, or animal waste
- After handling pet food or pet treats
- After touching garbage

How?

Wet your hands with clean, running water (warm or cold), turn off the tap, and apply soap.

Lather your hands by rubbing them together with the soap. Be sure to lather the backs of your hands, between your fingers, and under your nails.

Scrub your hands for at least 20 seconds. Need a timer? Hum the “Happy Birthday” song from beginning to end twice.

Rinse hands well under clean, running water.

Dry hands using a clean towel or air dry them.

Keeping hands clean is one of the most important things we can do to stop the spread of germs and stay healthy.

LIFE IS BETTER WITH CLEAN HANDS

www.cdc.gov/handwashing

This material was developed by CDC. The Life is Better with Clean Hands Campaign is made possible by a partnership between the CDC Foundation, GOJO, and Staples. HHS/CDC does not endorse commercial products, services, or companies.

CS310027-A