



COSC 3337
Data Science I
Section 14623

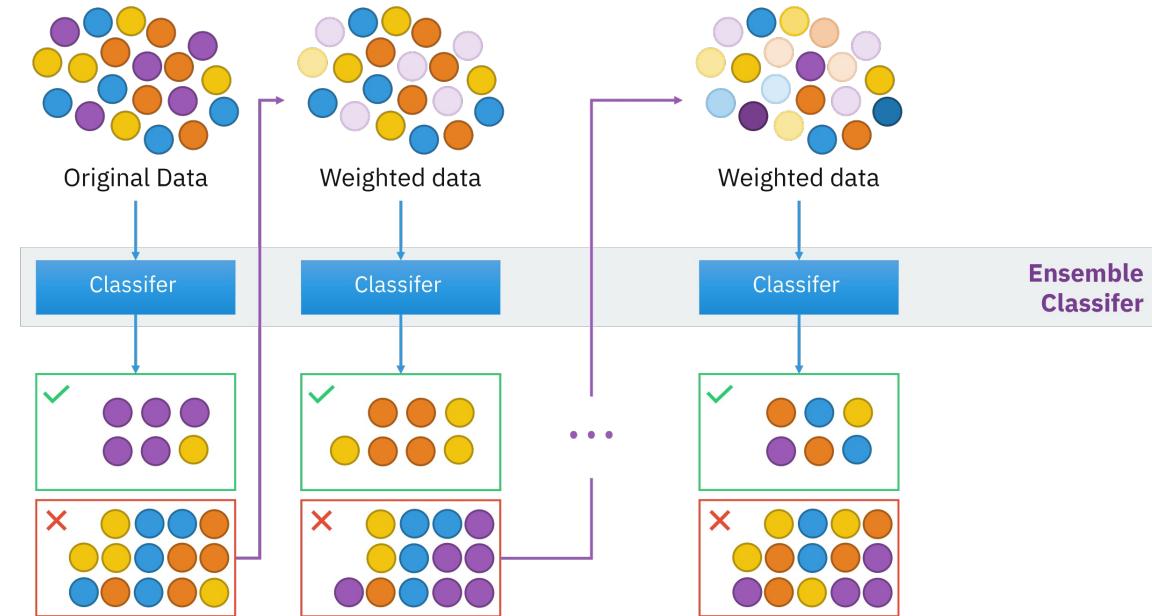
Classification (contd.), Clustering

Instructor: Jingchao Ni
Fall 2024

Boosting

- Models are trained in sequence

1. Create a base model and let it predict some results
2. Identify the data mislabeled as a wrong class
3. The data weights are readjusted, known as "re-weighting"
 - Misclassified input data gain a higher weight and data that are classified correctly lose weight
4. Create a next model that focuses more on the data misclassified by the previous models
5. Repeat this process till the whole training data set gets predicted accurately or if we reach the maximum number of models



Boosting

- Inference/Classifying new data
 1. Each model makes its prediction
 2. The final classifier is formed as the linear combination of the votes of the m weak classifiers (coefficient larger if training error is small)
- The main variation between many boosting algorithms is their method of weighting training data points and hypotheses
 - E.g., AdaBoost, XGBoost, etc.
- Boosting tends to achieve better accuracy than Bagging, but it also risks overfitting the model to misclassified data

Summary of Bagging and Boosting

- **Bias:** the difference between a classifier's expected prediction of a data point and the true class of the data point (from wrong assumption, simplified classifier, not from fluctuations)
- **Variance:** the amount by which the performance of a classifier changes when it is trained on different subsets of the training data

Bagging	Boosting
Different training data subsets are selected using sampling with replacement and random sampling methods from the entire training dataset	Iteratively train models, with each new model focusing on correcting the errors (misclassifications or high residuals) of the previous models
Each model is built independently	New models are influenced by the performance of previously built models
Each model receives equal weight	Models are weighted according to their performance
Bagging tries to solve the over-fitting problem, decrease variance, not bias	Boosting tries to reduce bias, not variance
If the classifier is unstable (high variance), then apply bagging	If the classifier is stable and simple (high bias), then apply boosting
Classifiers are trained in parallel	Classifiers are trained sequentially
Example: Random forest	Example: AdaBoost, XGBoost, etc.

Ensemble Method in Sklearn

Bagging

```
class sklearn.ensemble.BaggingClassifier(estimator=None, n_estimators=10, *, max_samples=1.0, max_features=1.0, bootstrap=True, bootstrap_features=False, oob_score=False, warm_start=False, n_jobs=None, random_state=None, verbose=0)

>>> from sklearn.ensemble import BaggingClassifier
>>> from sklearn.svm import SVC
>>> from sklearn.model_selection import train_test_split
>>> clf = BaggingClassifier(estimator=SVC(), n_estimators=10)
>>> X_tr, X_te, y_tr, y_te = train_test_split(X, y,
train_size=0.9)
>>> clf.fit(X_tr, y_tr)
>>> y_te = clf.predict(X_te)
```

Random Forest

```
class sklearn.ensemble.RandomForestClassifier(n_estimators=100, *, criterion='gini', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='sqrt', max_leaf_nodes=None, min_impurity_decrease=0.0, bootstrap=True, oob_score=False, n_jobs=None, random_state=None, verbose=0, warm_start=False, class_weight=None, ccp_alpha=0.0, max_samples=None, monotonic_cst=None)
```

Boosting

```
class sklearn.ensemble.GradientBoostingClassifier(*, loss='log_loss', learning_rate=0.1, n_estimators=100, subsample=1.0, criterion='friedman_mse', min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_depth=3, min_impurity_decrease=0.0, init=None, random_state=None, max_features=None, verbose=0, max_leaf_nodes=None, warm_start=False, validation_fraction=0.1, n_iter_no_change=None, tol=0.001, ccp_alpha=0.0)
```

```
>>> from sklearn.ensemble import GradientBoostingClassifier
>>> from sklearn.model_selection import train_test_split
>>> clf = GradientBoostingClassifier(n_estimators=100,
learning_rate=1.0, max_depth=1)
>>> X_tr, X_te, y_tr, y_te = train_test_split(X, y,
train_size=0.9)
>>> clf.fit(X_tr, y_tr)
>>> y_te = clf.predict(X_te)
```

```
class sklearn.ensemble.AdaBoostClassifier(estimator=None, *, n_estimators=50, learning_rate=1.0, algorithm='SAMME.R', random_state=None)
```

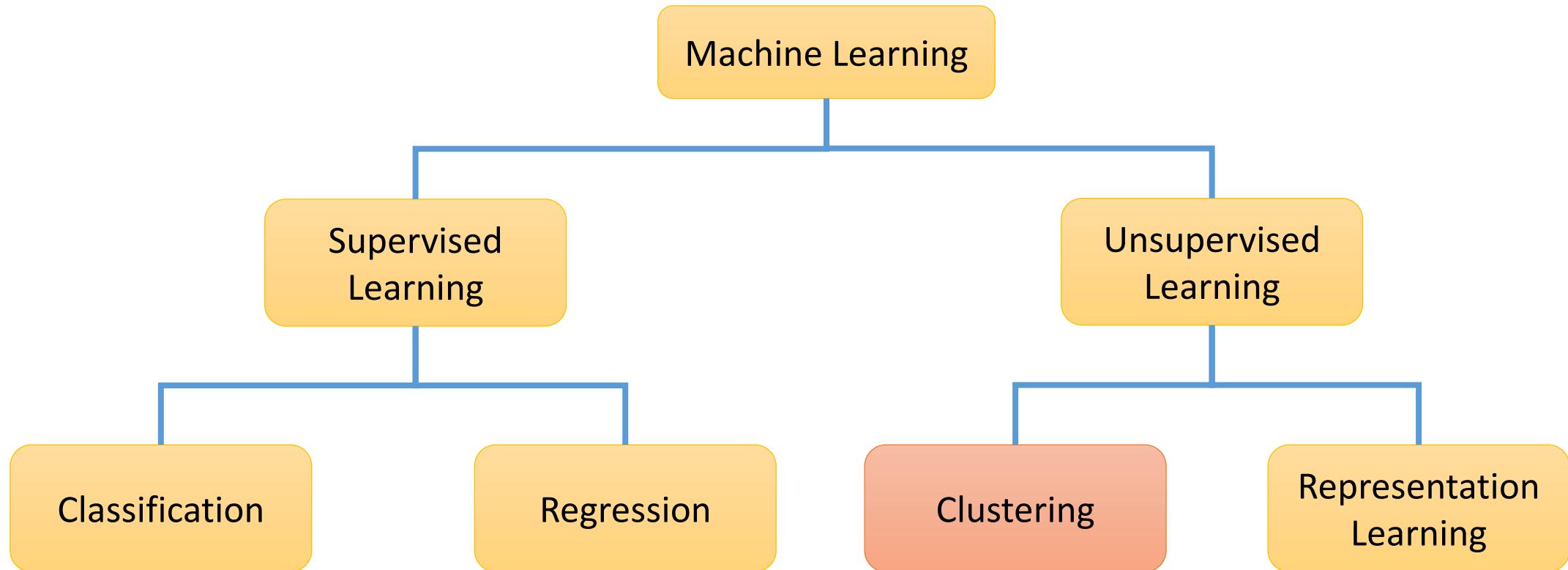
Summary of Classification

- **Classification** is a form of data analysis that extracts **models** describing important data classes
- Effective and scalable methods have been developed for **decision tree induction**, **Naive Bayesian classification**, **support vector machine**, and many other classification methods
- **Evaluation metrics** include accuracy, classification error, precision, recall, F1 score, F_β score, AUPRC, AUROC, Macro-F1 score, Micro-F1 score, etc.
- **K-fold cross-validation** is recommended for accuracy estimation. **Bagging** and **boosting** can be used to increase the overall accuracy by learning and combining a series of individual models

Summary of Classification

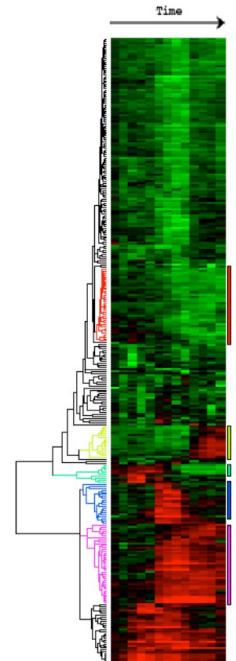
- There have been numerous **comparisons of different classification methods**; the matter remains a research topic
- No single method has been found to be superior to all other methods for all datasets
- Issues such as accuracy, training time, robustness, scalability, and interpretability must be considered and can involve trade-offs, further complicating the quest for an overall superior method

Machine Learning: Topics Taxonomy



Machine Learning: Topics Taxonomy

- Unsupervised learning
 - Requires data attributes, but no labels
- Identification of a finite set of categories, classes or groups (clusters) in the dataset
 - Group documents
 - Image segmentation
 - Clustering gene expression data
- Principle
 - Data points in the same cluster are **more similar** to each other
 - Minimize **intracluster** distance
 - Data points in different clusters are **less similar** to each other
 - Maximize **intercluster** distance

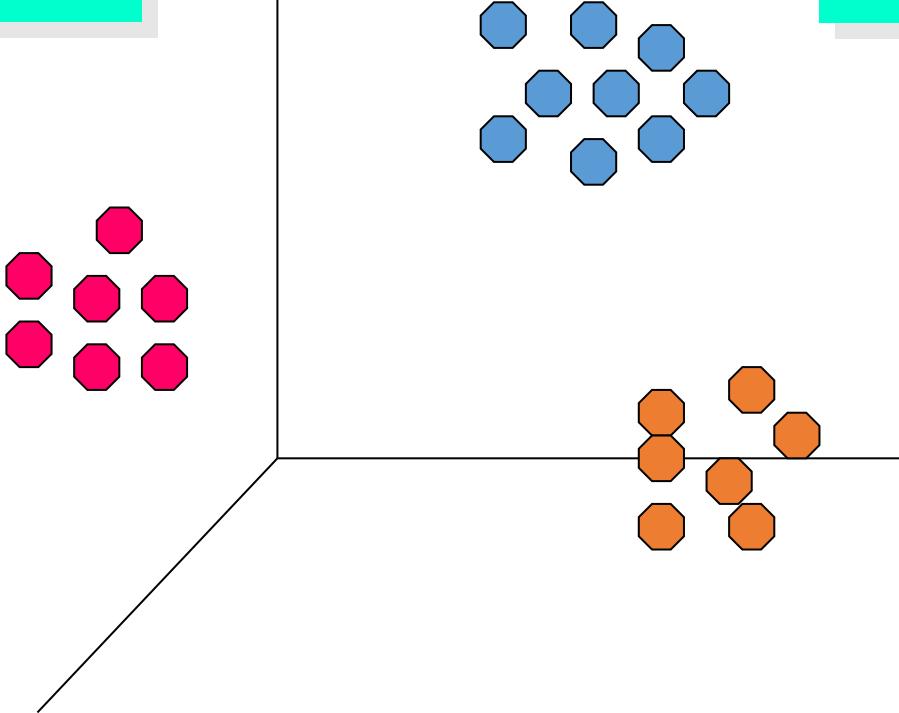


Eisen et al. PNAS 1998

Illustrating Clustering

Intracluster distances
are minimized

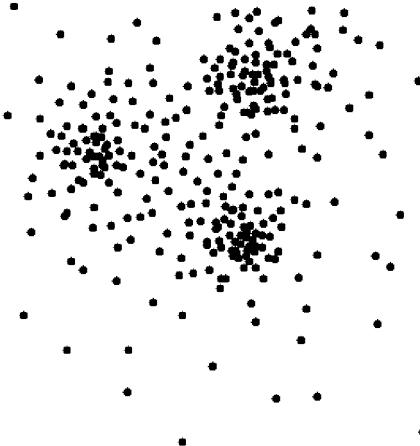
Intercluster distances
are maximized



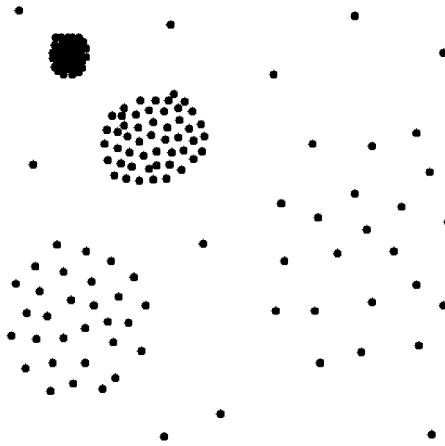
Euclidean Distance Based Clustering in 3-D space

Different Clusters

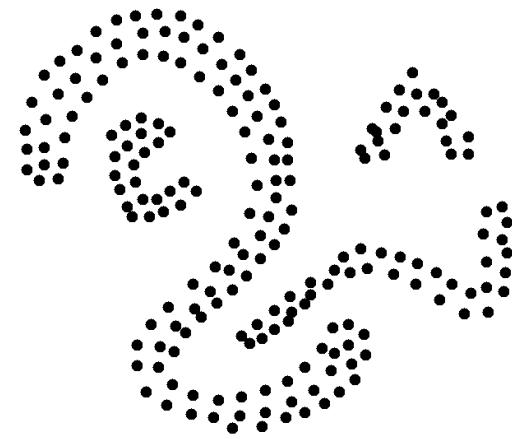
- Clusters of different sizes, shapes, densities
- Hierarchical clusters
- Disjoint / overlapping clusters



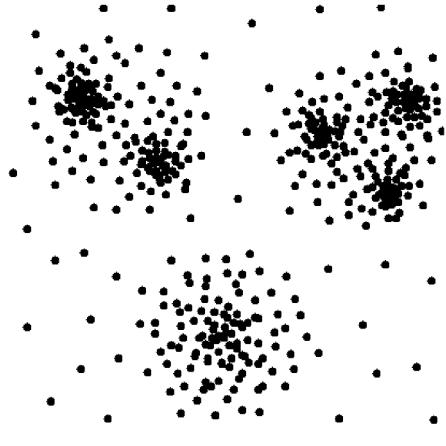
(1)



(2)



(3)



(4)

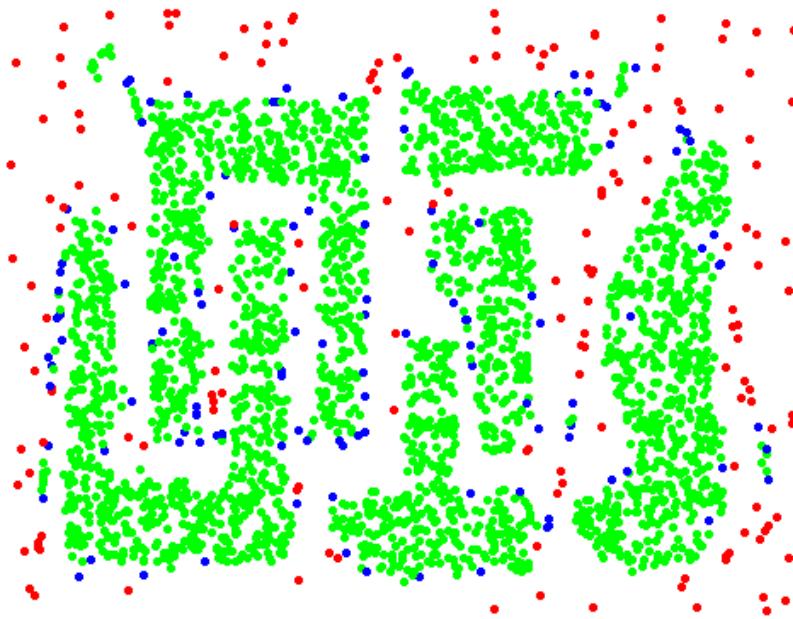
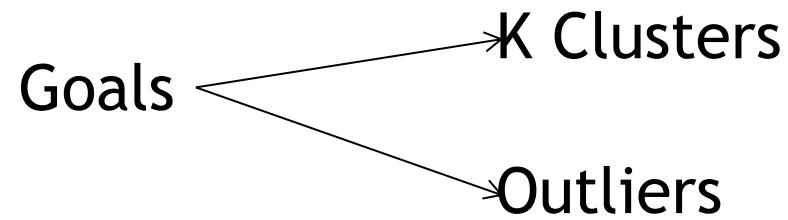
Motivation: Why Clustering?

- **Problem:** Identify (a small number of) groups of similar objects in a given (large) set of objects
- **Uses:**
 - Find representatives for homogeneous groups → **Data Compression**
 - Find “natural” clusters and describe their properties → **“natural” Data Types**
 - Find suitable and useful grouping → **“useful” Data Classes**
 - Find unusual data object → **Outlier Detection**

Goal of Clustering



Original Points



Point types: **core**, **border** and **noise**

DBSCAN Result, Eps = 10, MinPts = 4

Requirements of Clustering in Data Mining

Scalability

Ability to deal with different types of attributes

Discovery of clusters with arbitrary shape

Minimal requirements for domain knowledge to determine input parameters

Able to deal with noise and outliers

Insensitive to the order of input records

High dimensionality

Incorporation of user-specified constraints

Interpretability and usability

Data Structures for Clustering

- Data Matrix
 - (n objects, p attributes)

$$\begin{bmatrix} x_{11} & \cdots & x_{1f} & \cdots & x_{1p} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ x_{i1} & \cdots & x_{if} & \cdots & x_{ip} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ x_{n1} & \cdots & x_{nf} & \cdots & x_{np} \end{bmatrix}$$

- (Dis)Similarity Matrix
 - (n objects x n objects)

$$\begin{bmatrix} 0 & & & & \\ d(2,1) & 0 & & & \\ d(3,1) & d(3,2) & 0 & & \\ \vdots & \vdots & \vdots & & \\ d(n,1) & d(n,2) & \dots & \dots & 0 \end{bmatrix}$$

Major Clustering Approaches



Partitioning algorithms/Representative-based/Prototype-based Clustering Algorithm:
Construct various partitions and then evaluate them by some criterion or fitness function



Hierarchical algorithms: Create a hierarchical decomposition of the set of data (or objects) using some criterion



Density-based: based on connectivity and density functions



Grid-based: based on a multiple-level granularity structure



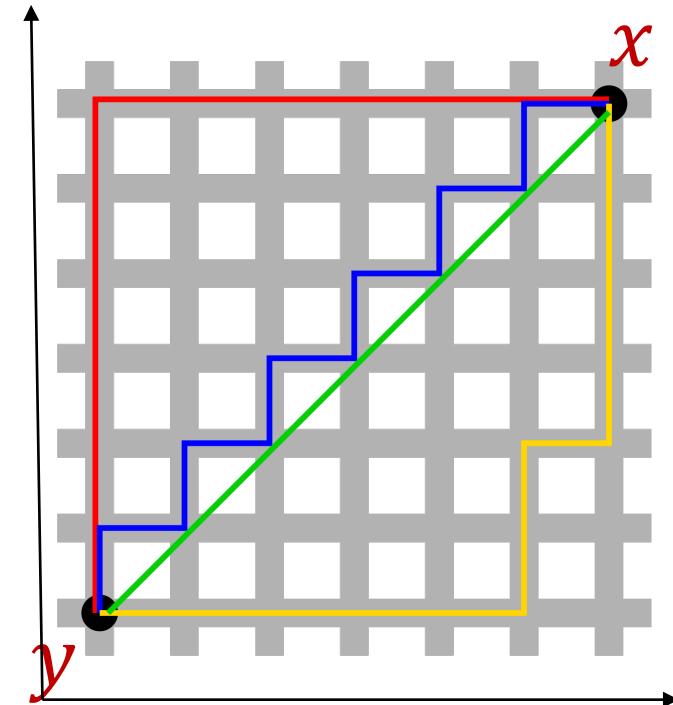
Model-based: A model is hypothesized for each of the clusters and the idea is to find the best fit of that model to the data distribution

Representative-Based Clustering

- Aims at finding a set of objects among all objects (called **representatives**) in the data set that **best represent the objects in the data set**. Each representative corresponds to a cluster
- The remaining objects in the data set are then clustered around these representatives by assigning objects to the cluster of the **closest** representative
- **Remarks:**
 - The popular k-medoid algorithm, **Partition Around Medoids (PAM)**, is a representative-based clustering algorithm; **K-means** also shares the characteristics of representative-based clustering, except that the representatives used by k-means do not have to belong to the dataset
 - If the representative do not need to belong to the dataset we call the algorithms **prototype-based clustering**. K-means is a prototype-based clustering algorithm

Distance Function

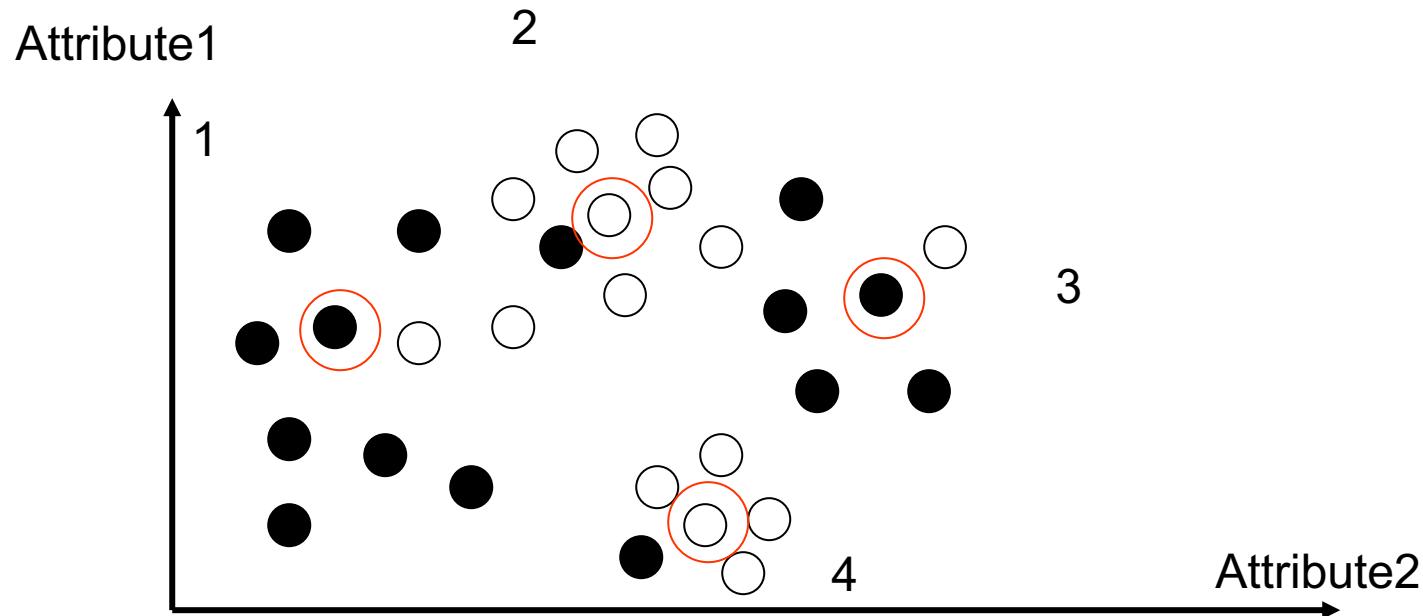
- Manhattan Distance $d(x, y) = \sum_{i=1}^d |x_i - y_i|$
- Cosine Distance $d(x, y) = 1 - \frac{\sum_{i=1}^d x_i y_i}{\sqrt{\sum_{i=1}^d x_i^2} \sqrt{\sum_{i=1}^d y_i^2}}$
- Euclidean Distance $d(x, y) = \sqrt{\sum_{i=1}^d (x_i - y_i)^2}$
- Minkowski Distance $d(x, y) = \left(\sum_{i=1}^d |x_i - y_i|^p \right)^{\frac{1}{p}}$



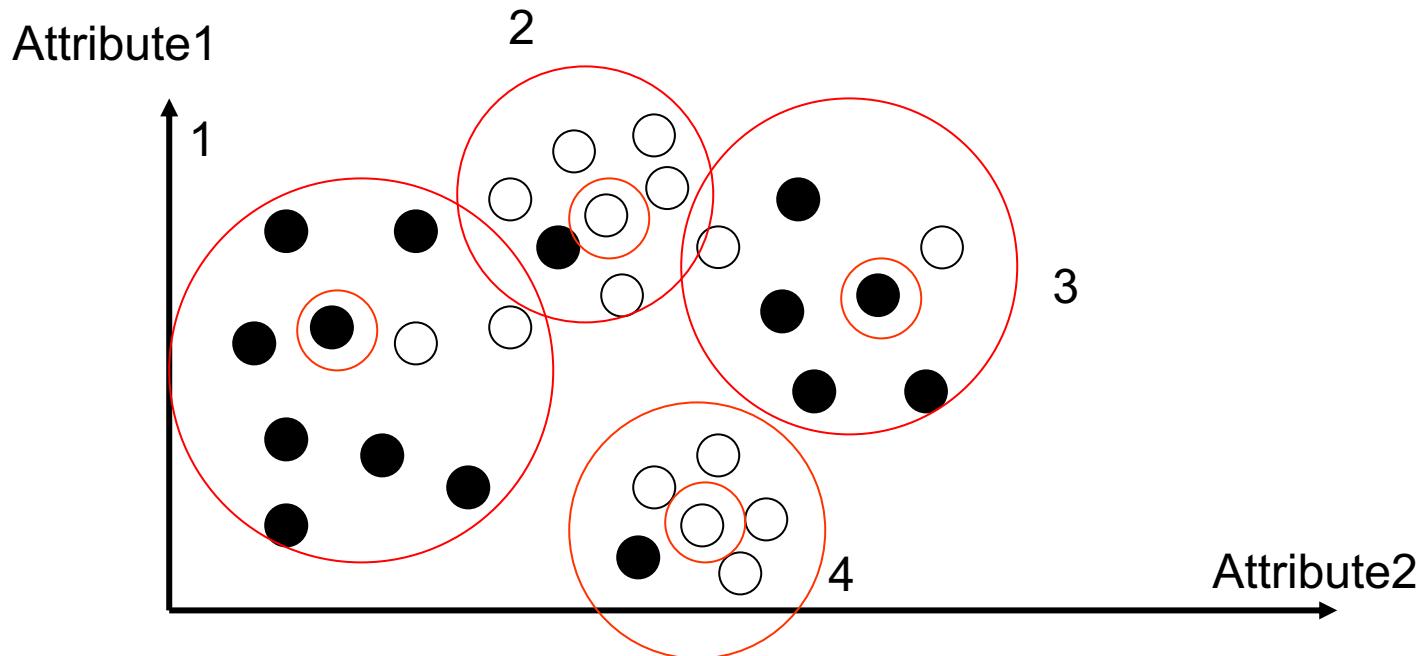
Properties of Distance Measure

- Symmetric
 - $D(A, B) = D(B, A)$
 - Otherwise, we can say A looks like B but B does not look A
- Positivity, and self-similarity
 - $D(A, B) \geq 0$, and $D(A, B) = 0$ iff $A = B$
 - Otherwise, there will be different objects that we cannot tell apart
- Triangle inequality
 - $D(A, B) + D(B, C) \geq D(A, C)$
 - Otherwise, one can say “A is like B, B is like C, but A is not like C at all”

Representative-Based Clustering



Representative-Based Clustering



Objective of RBC: Find a subset O_R of O such that the clustering X obtained by using the objects in O_R as representatives minimizes $q(X)$; q is an objective/fitness function

Partitioning Algorithms: Basic Concept

Partitioning method: Construct a partition of a database D of n objects into a set of k clusters

Given a k , find a partition of k clusters that optimizes the chosen partitioning criterion or fitness function.

Global optimal: exhaustively enumerate all partitions

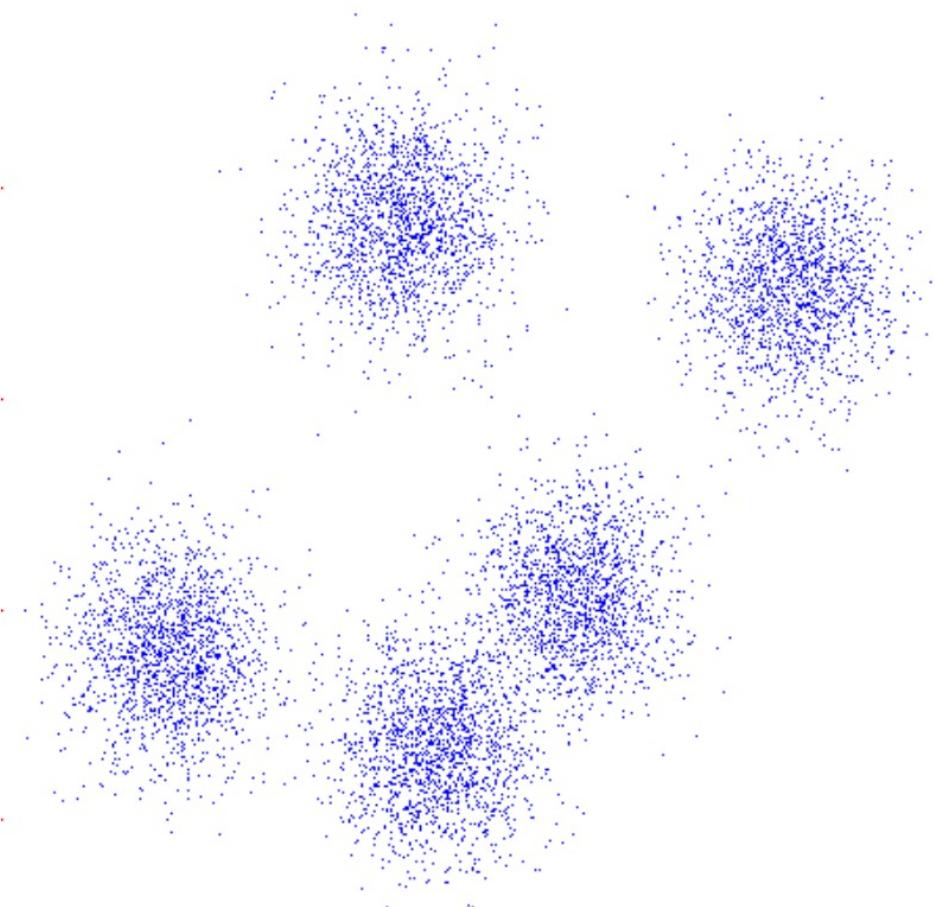
Heuristic methods: $k\text{-means}$ and $k\text{-medoids}$ algorithms

$k\text{-means}$ (MacQueen'67): Each cluster is represented by the center of the cluster (prototype)

$k\text{-medoids}$ or PAM (Partition around medoids) (Kaufman & Rousseeuw'87): Each cluster is represented by one of the objects in the cluster; truly representative-based.

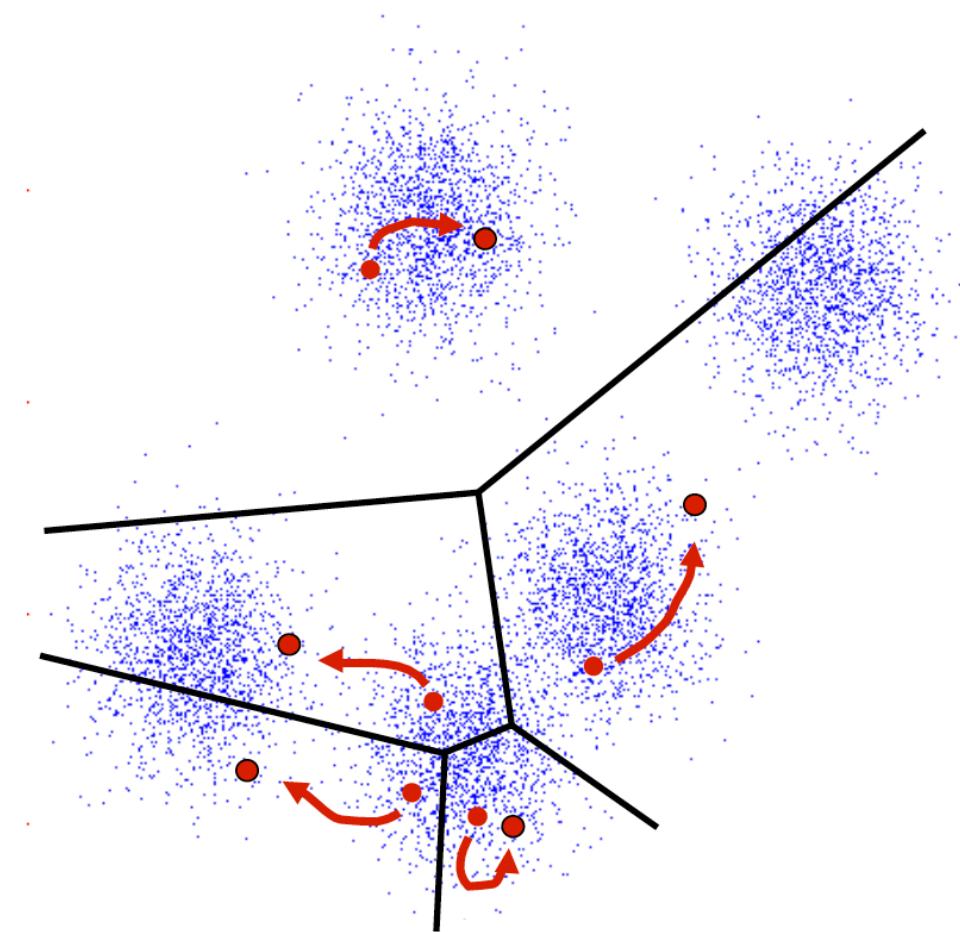
K-Means Clustering

- An iterative clustering algorithm
- Initialize: Pick K random points as cluster centers
- Alternate:
 - Assign each data point to its closest cluster center
 - Change the cluster center to the average of its assigned points
- Stop when no points' assignments change

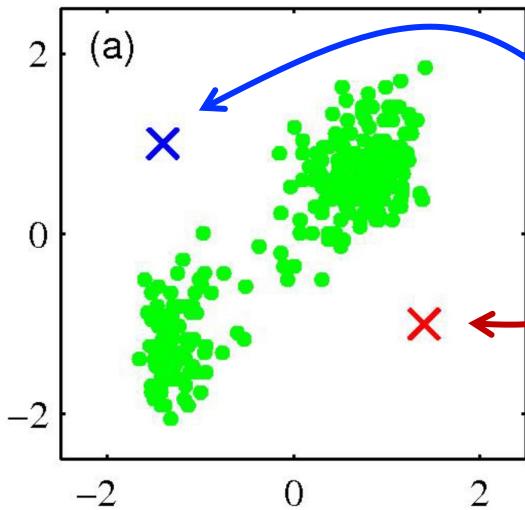


K-Means Clustering

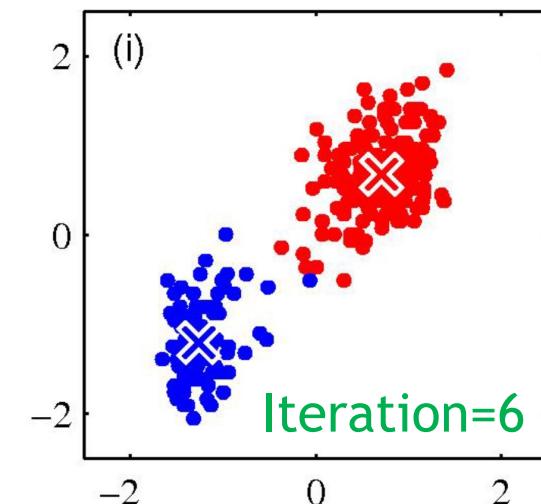
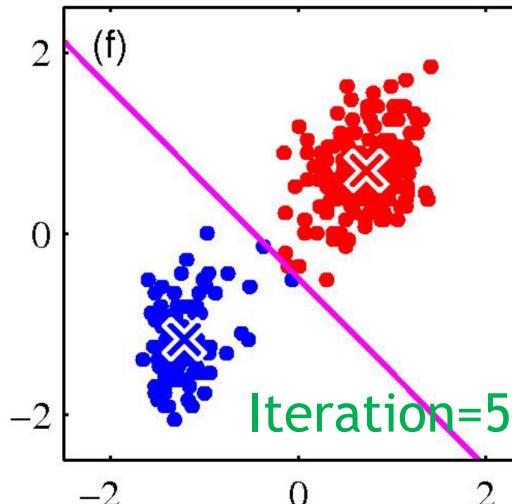
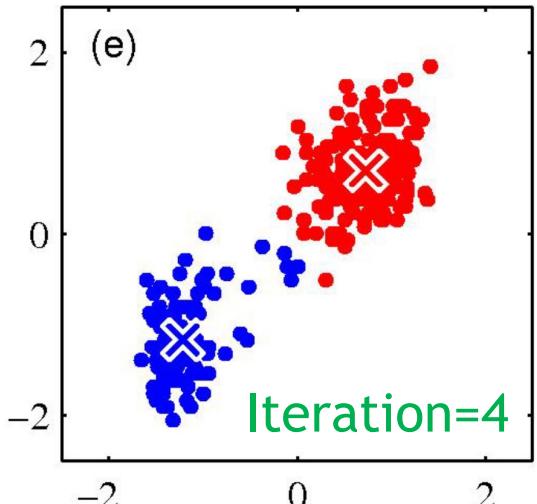
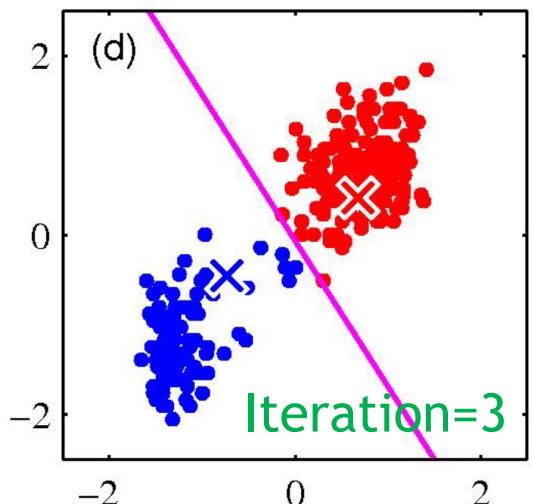
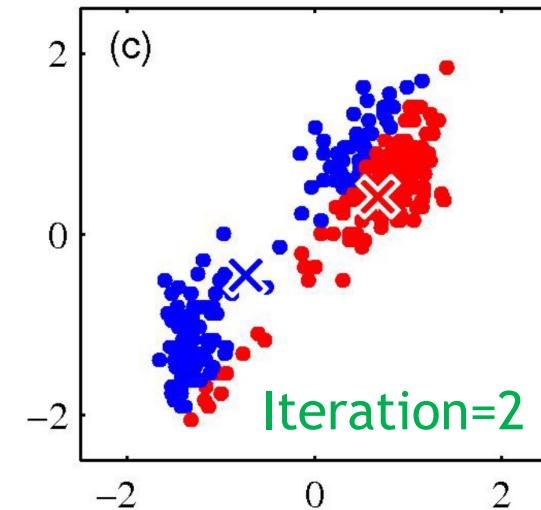
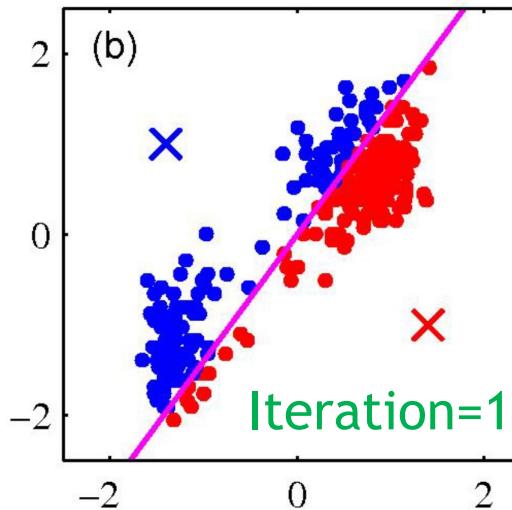
- An iterative clustering algorithm
- Initialize: Pick K random points as cluster centers
- Alternate:
 - Assign each data point to its closest cluster center
 - Change the cluster center to the average of its assigned points
- Stop when no points' assignments change



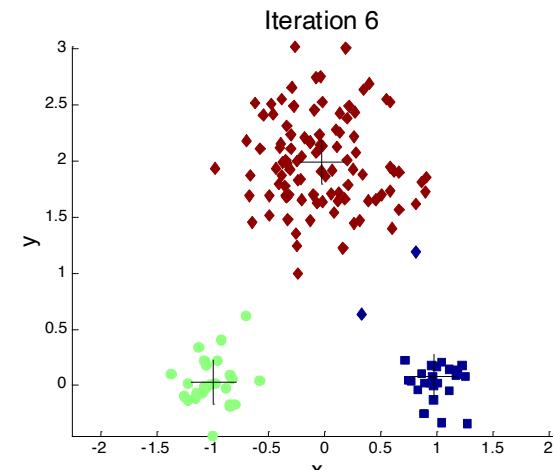
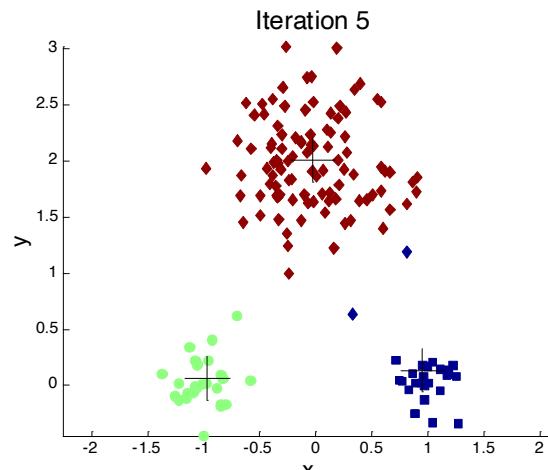
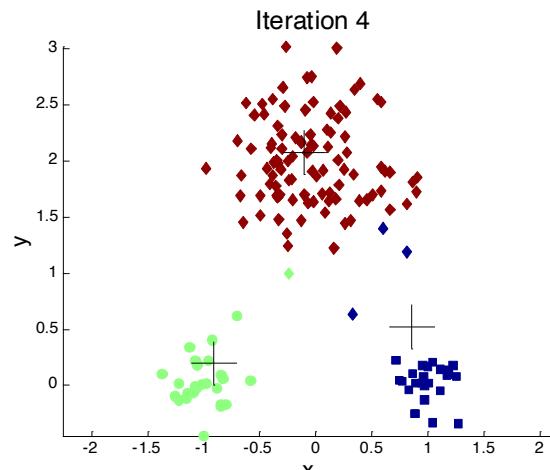
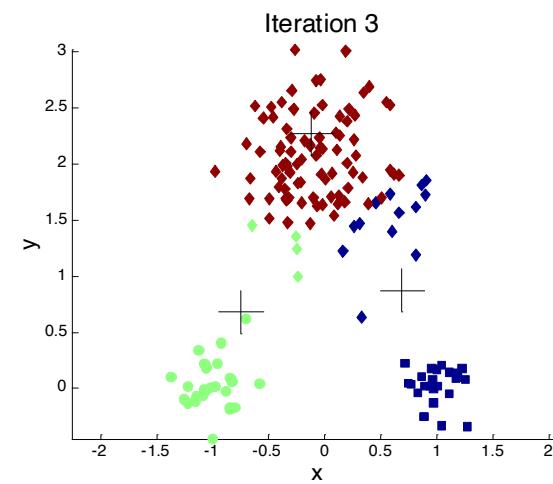
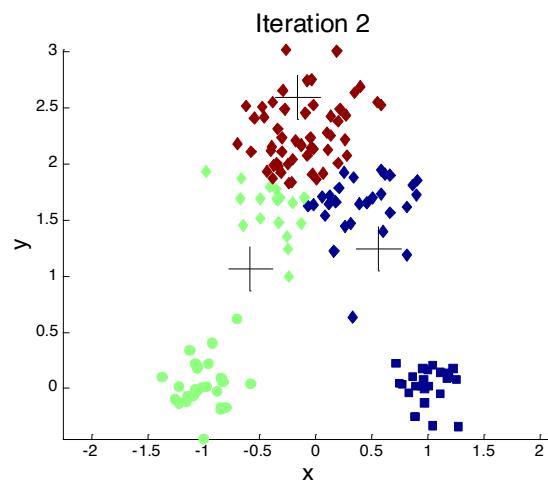
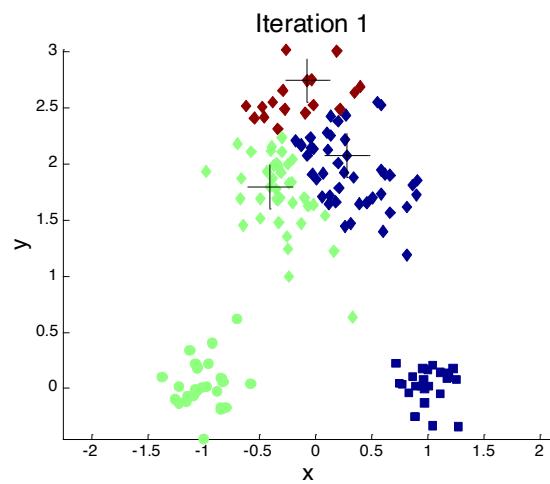
Example of K-Means Clustering



Pick K random points as cluster centers
▪ Here K=2



Another Example of K-Means Clustering



Properties of K-Means Clustering

- Guaranteed to converge in a finite number of iterations
- Running time per iteration:
 - Assign data points to the closest cluster center: $O(KND)$
 - K is the number of clusters, N is the number of data points, D is the number of dimensions
 - Change the cluster center to the average of its assigned points: $O(ND)$
- Partition Data into Disjoint Clusters
 - Let D be a dataset; then $C=\{C_1, \dots, C_k\}$ is a clustering of D with $C_i \subseteq D$ (for $i=1, \dots, k$), $C_1 \cup \dots \cup C_k \subseteq D$ and $C_i \cap C_j = \emptyset$ (for $i \neq j$)

Properties of K-Means Clustering

- Objective

$$\min_{\mu} \min_C \sum_{i=1}^k \sum_{x \in C_i} \text{dist}(x, \mu_i)$$

Step 1 of K-means

- Fix μ , optimize C :

$$\min_C \sum_{i=1}^k \sum_{x \in C_i} \text{dist}(x, \mu_i) = \min_C \sum_{i=1}^n \text{dist}(x_i, \mu_{x_i})$$

- Fix C , optimize μ :

$$\min_{\mu} \sum_{i=1}^k \sum_{x \in C_i} \text{dist}(x, \mu_i)$$

Take partial derivative of
 μ_i and set to zero, we have

$$\mu_i = \frac{1}{|C_i|} \sum_{x \in C_i} x$$

Step 2 of K-means

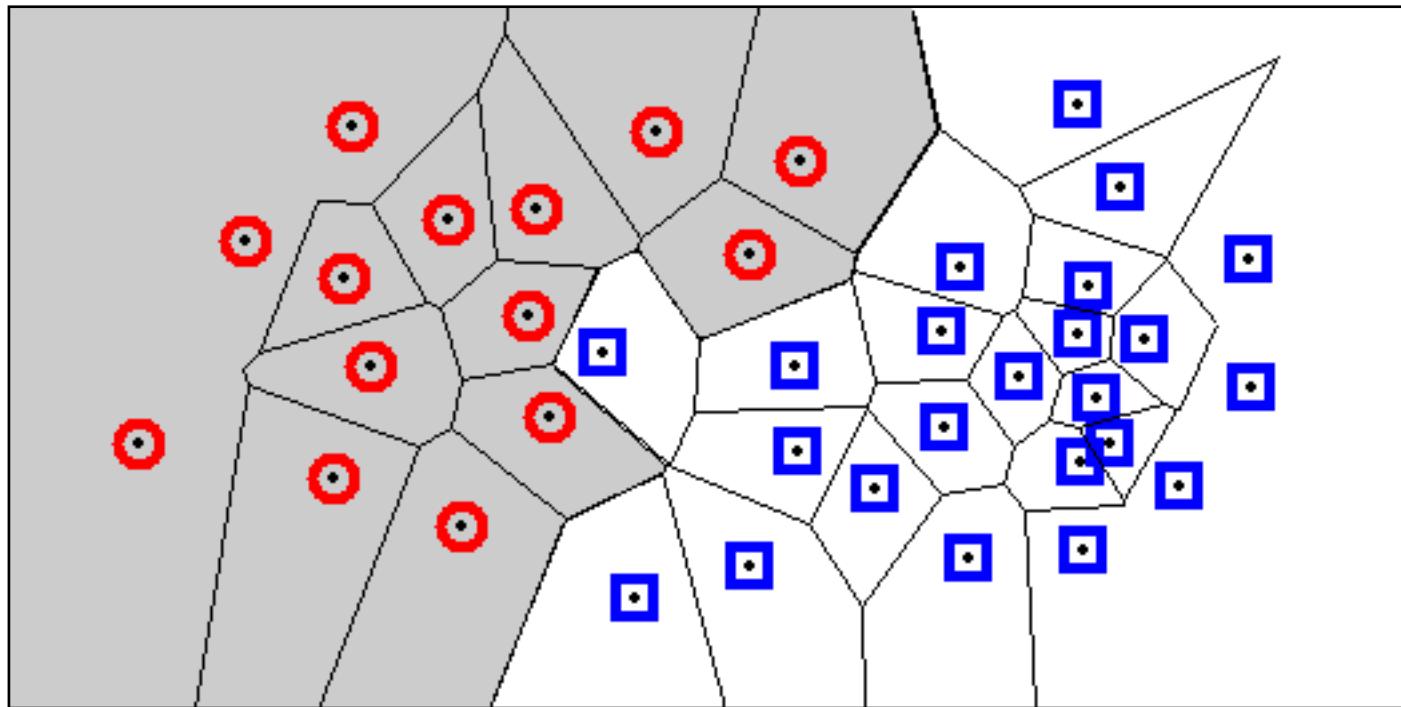
K-means takes an alternating optimization approach, each step is guaranteed to decrease the objective, thus guaranteed to converge

Convex Shape Cluster

- Convex Shape: if we take two points belonging to a cluster then all the points on a direct line connecting these two points must also be in the cluster
- Shape of K-means/K-medoids clusters are convex polygons
 \subseteq Convex Shape
- Shapes of clusters of a representative-based clustering algorithm can be computed as a Voronoi diagram for the set of cluster representatives
- Voronoi cells are always convex

Voronoi Diagram

- Voronoi Diagram for A Representative-Based Clustering



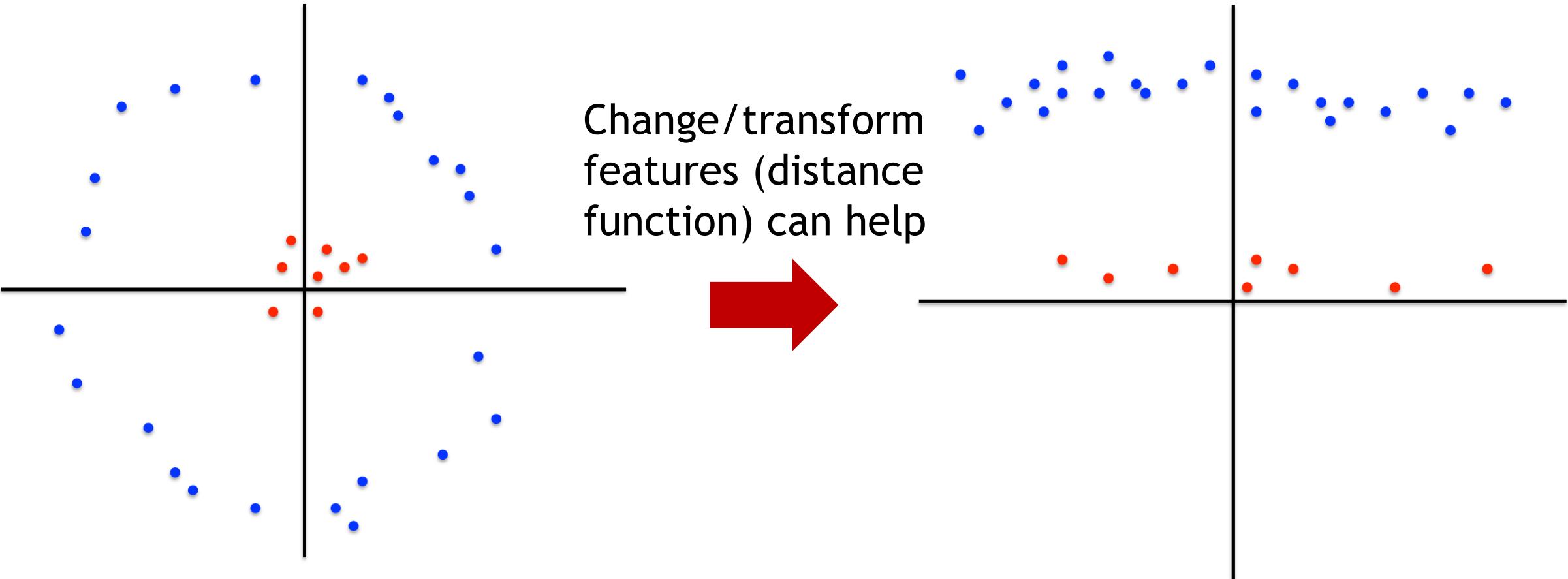
Cluster Representative (e.g. medoid/centroid)

Each cell contains one representatives, and every location within the cell is closer to that sample than to any other sample.

A **Voronoi diagram** divides the space into such cells.

Voronoi cells define cluster boundary!

Example of Non-Convex Shape Cluster





COSC 3337
Data Science I
Section 14623

Clustering (contd.)

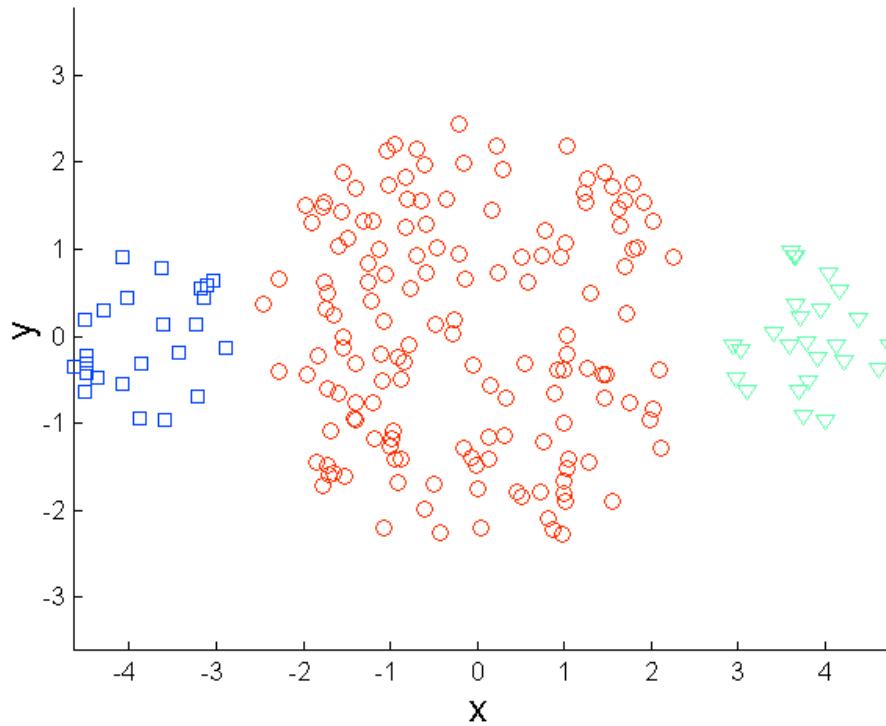
Instructor: Jingchao Ni
Fall 2024

Limitations of K-Means Clustering

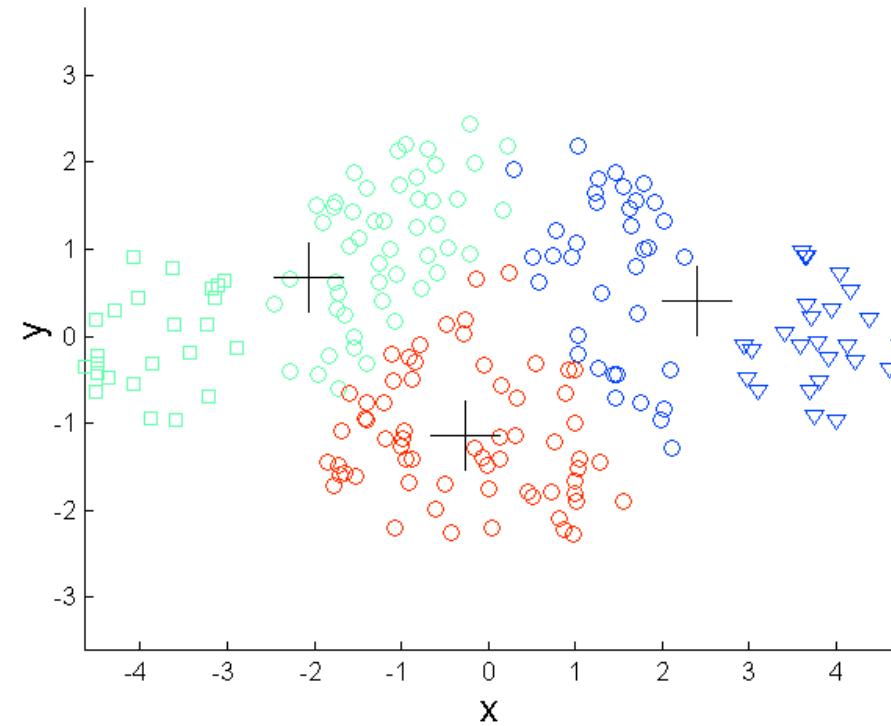
- K-means has problems when clusters are of different
 - Sizes
 - Densities
 - Non-convex shapes
- K-means has problems when the data contains outliers
 - One possible solution is to remove outliers before clustering

Limitations of K-Means Clustering

- Different Sizes



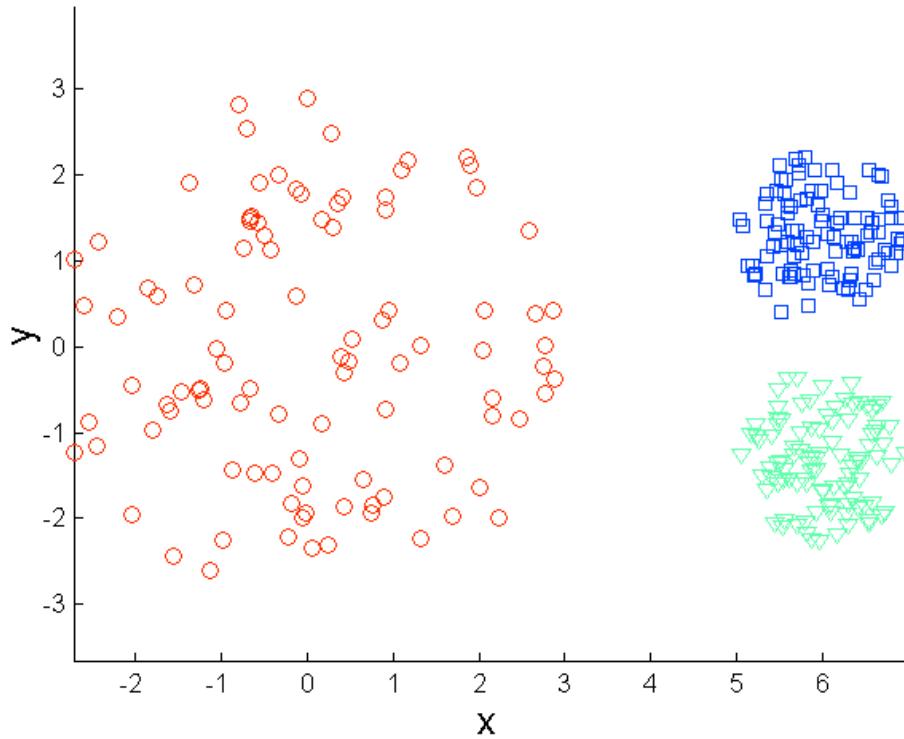
Original Points



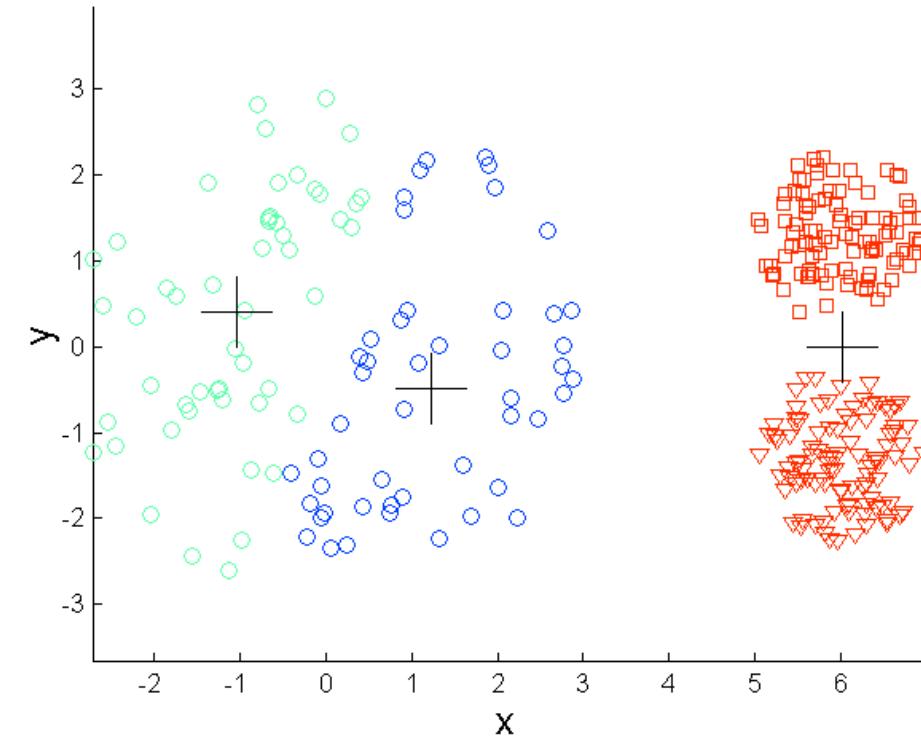
K-means (3 Clusters)

Limitations of K-Means Clustering

- Different Density



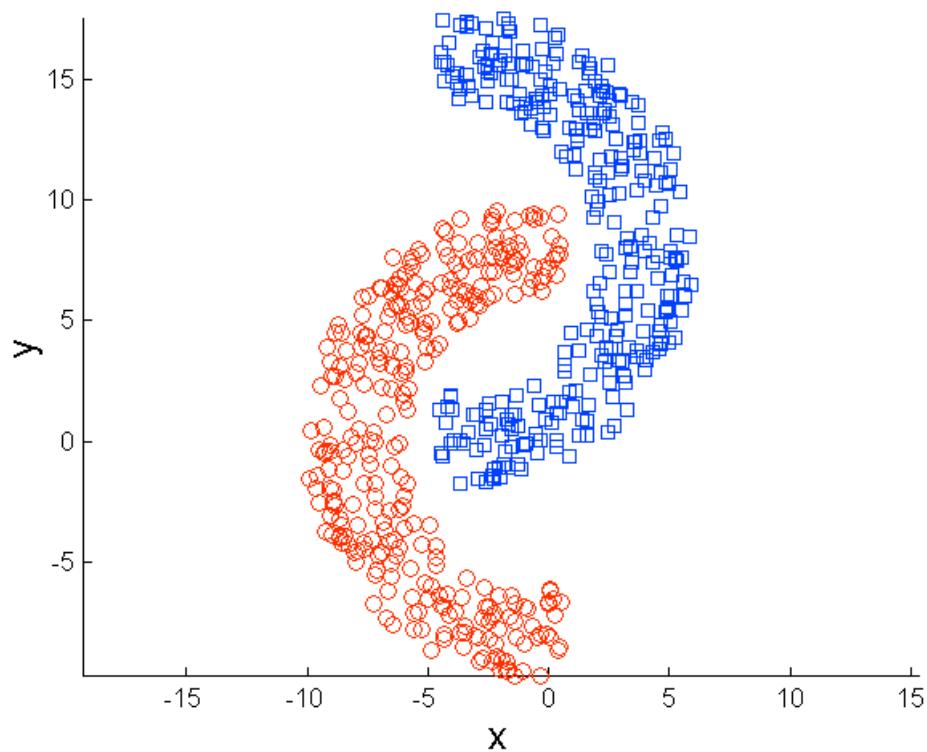
Original Points



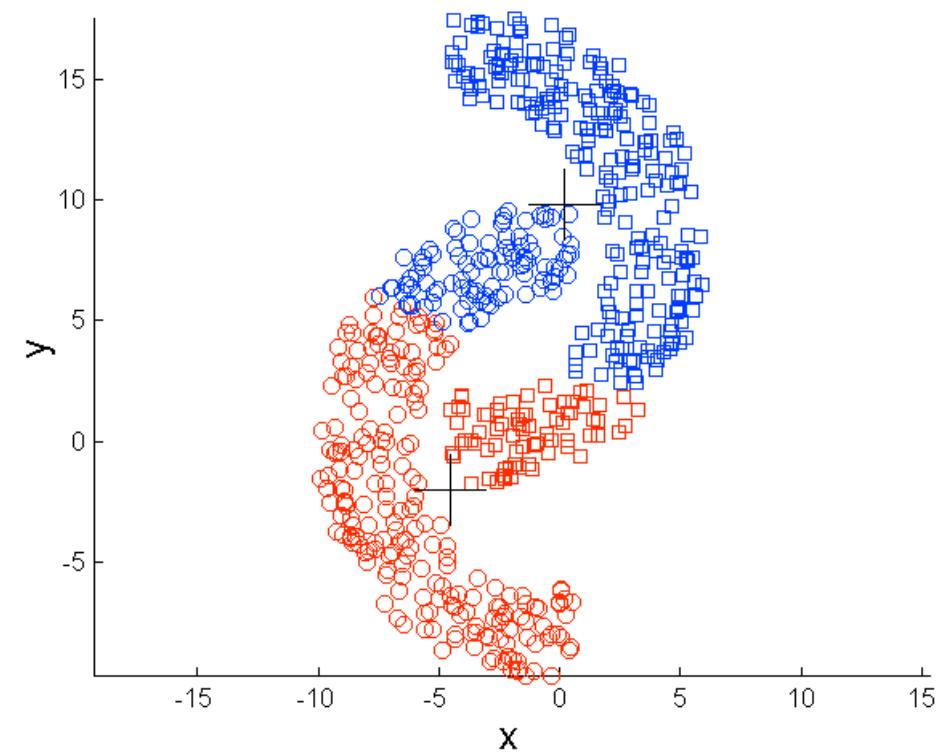
K-means (3 Clusters)

Limitations of K-Means Clustering

- Non-Convex Shapes



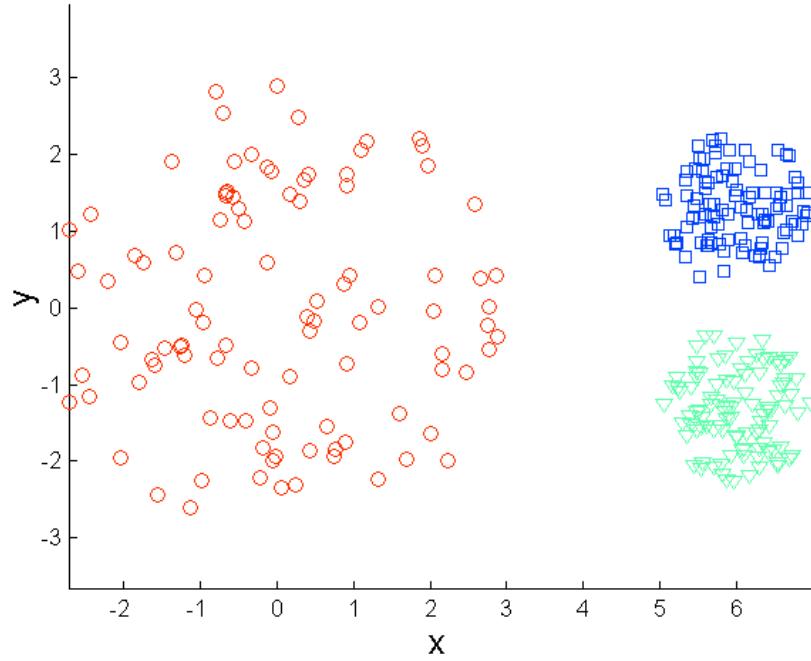
Original Points



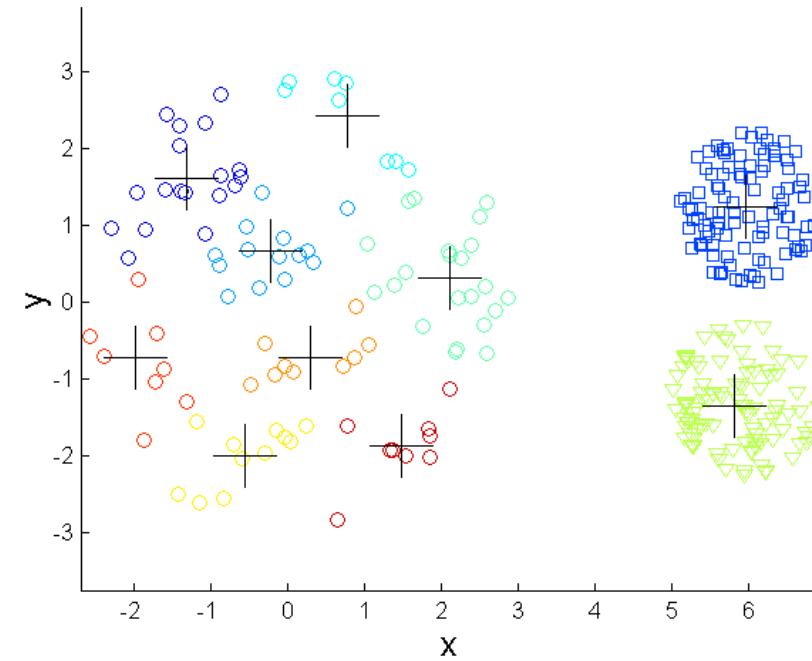
K-means (2 Clusters)

Addressing K-Means Limitations

- Different density



Ground Truth

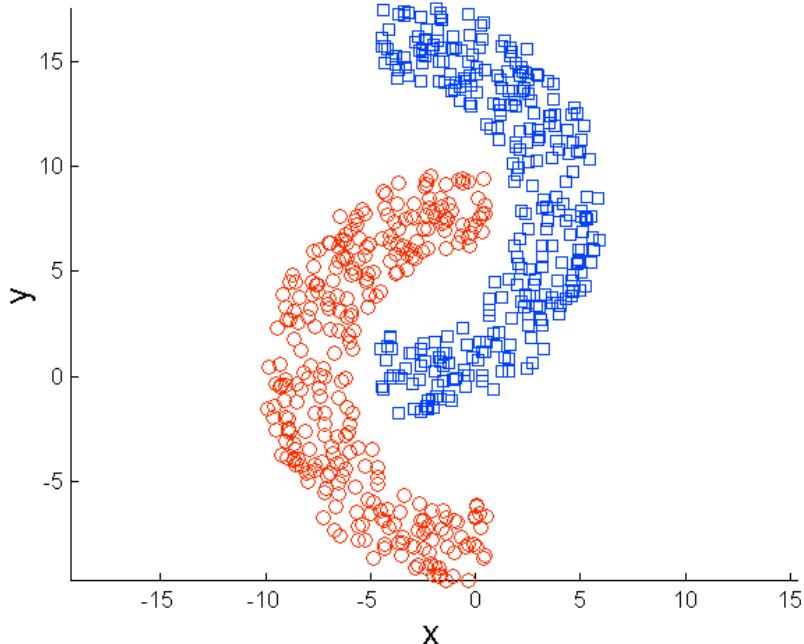


K-means Clusters)

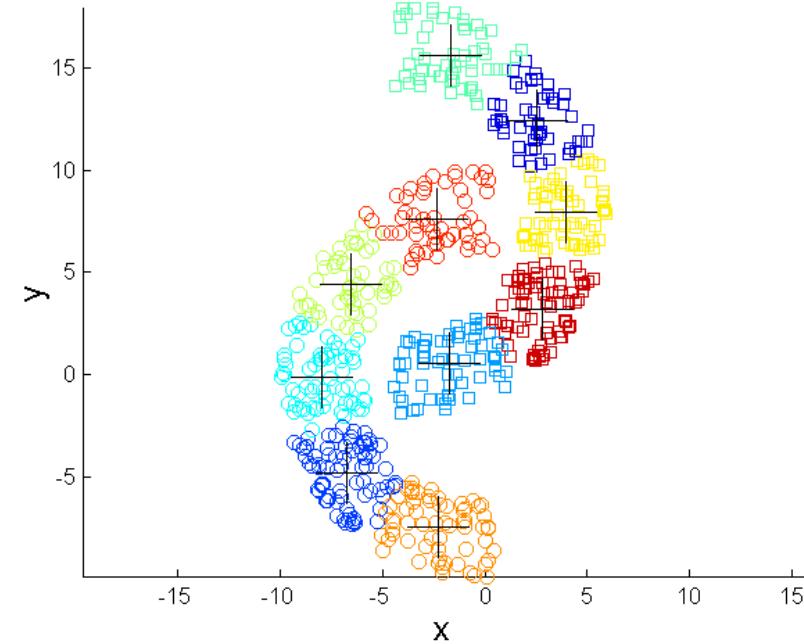
One solution is to find a large number of clusters such that each of them represents a part of a natural cluster. But these small clusters need to be put together in a post-processing step.

Addressing K-Means Limitations

- Non-Convex Shapes



Ground Truth



K-means Clusters)

One solution is to find a large number of clusters such that each of them represents a part of a natural cluster. But these small clusters need to be put together in a post-processing step.

Summary of K-Means Clustering

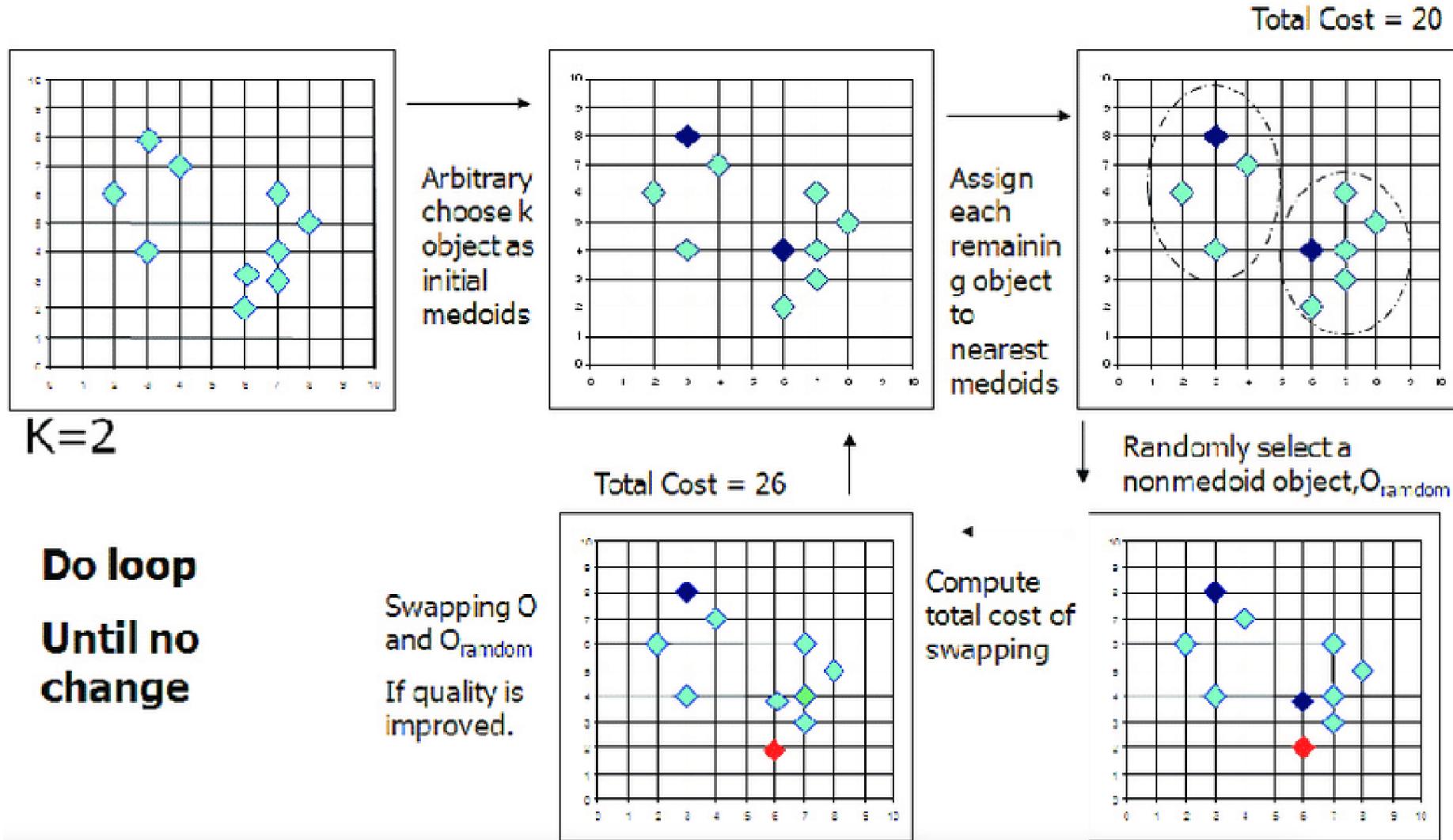
- Strength
 - Relatively efficient: $O(t*k*n*d)$, where n is # objects, k is # clusters, and t is # iterations, d is the # dimensions. Usually, $d, k, t \ll n$; in this case, K-Mean's runtime is $O(n)$
 - Storage only $O(n)$: in contrast to other representative-based algorithms, only computes distances between centroids and objects in the dataset, and not between objects in the dataset; therefore, the distance matrix does not need to be stored
 - Easy to use; well studied; we know what to expect
 - Finds local minimum of its objective function. The global optimum may be found using techniques such as: deterministic annealing and genetic algorithms
 - Implicitly uses an objective function (finds a local minimum), does not waste time computing fitness values
- Weakness
 - Applicable only when mean is defined (what about categorical data?)
 - Need to specify k , the number of clusters, in advance
 - Sensitive to outliers; does not identify outliers
 - Sensitive to initialization; bad initialization might lead to bad results
 - Problems with different cluster sizes, varying densities and non-convex shapes

Partitioning Around Medoids (PAM)

■ Algorithm

- PAM uses a greedy search which may not find the optimum solution, but it is faster than exhaustive search. It works as follows:
 1. **(BUILD)** Initialize: greedily select k of the n data points as the medoids to minimize the cost
 2. Associate each data point to the closest medoid
 3. **(SWAP)** While the cost of the configuration decreases:
 1. For each medoid m , and for each non-medoid data point x :
 2. Consider the swap of m and x , and compute the cost change
 3. If the cost change is the current best, remember (m, x) as (m_{best}, x_{best})
 4. Perform the best swap of m_{best} and x_{best} , if it decreases the cost function. Otherwise, the algorithm terminates

Partitioning Around Medoids (PAM)



Partitioning Around Medoids (PAM)

- Fitness/Cost Function: most common measure to evaluate a clustering of X is the Sum of Squared Error (SSE)
 - For each point, the error is the distance to the nearest cluster representative
 - To get SSE, we square these errors and sum them

$$\text{SSE}(X) = \sum_{i=1}^k \sum_{x \in C_i} \text{dist}^2(x, m_i)$$

- x is a data point in cluster C_i , m_i is the representative point for cluster C_i
- The MSE-error computes the average of the squared error value instead
- Algorithm complexity per iteration: $O(k(n-k)^2)$
 - For each pair (m_i, x) , consider swap, then compute cost change, which is between the new medoid after swap and other data points, takes $O(n-k)$

Comparison of K-Means and PAM

- Form the clusters the same way by assigning objects to the closest representative
- Find convex shape clusters; they discover clusters of similar shape
- K-means uses centroids and PAM uses objects in the dataset as cluster representatives
- K-means is much faster than PAM!
- You can run PAM with other fitness functions $q(X)$, in addition to SSE → can be applied to a wider range of clustering problems

K-Means in Sklearn

K-Means

```
class sklearn.cluster.KMeans(n_clusters=8, *, init='k-means++', n_init='auto', max_iter=300, tol=0.0001, verbose=0, random_state=None, copy_x=True, algorithm='lloyd'
```

Init: {‘k-means++’, ‘random’}, callable or array-like of shape (n_clusters, n_features), default=‘k-means++’

Method for initialization:

- ‘k-means++’: selects initial cluster centroids using sampling based on an empirical probability distribution of the points’ contribution to the overall inertia. This technique speeds up convergence. The algorithm implemented is “greedy k-means++”. It differs from the vanilla k-means++ by making several trials at each sampling step and choosing the best centroid among them.
- ‘random’: choose n_clusters observations (rows) at random from data for the initial centroids.

```
from sklearn.cluster import Kmeans
>>> import numpy as np
>>> X = np.array([[1, 2], [1, 4], [1, 0], ... [10, 2],
[10, 4], [10, 0]])
>>> kmeans = KMeans(n_clusters=2, n_init="auto").fit(X)
>>> kmeans.labels_
array([1, 1, 1, 0, 0, 0], dtype=int32)
>>> kmeans.predict([[0, 0], [12, 3]])
array([1, 0], dtype=int32)
>>> kmeans.cluster_centers_
array([[10., 2.], [1., 2.]])
```

K-Medoids in Sklearn_extra

K-Medoids

```
class sklearn_extra.cluster.KMedoids(n_clusters=8,  
metric='euclidean', method='alternate', init='heuristic',  
max_iter=300, random_state=None)
```

Metric: string, or callable, optional, default: 'euclidean'
What distance metric to use.

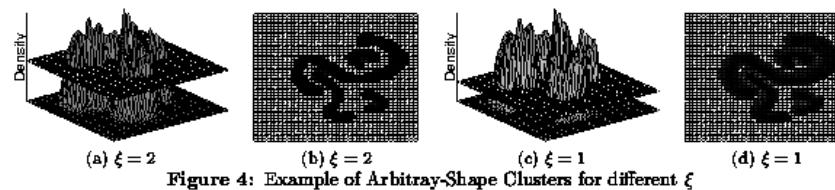
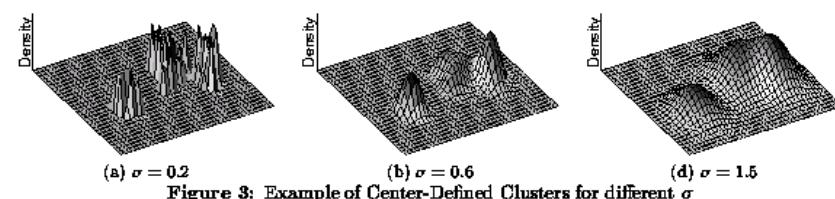
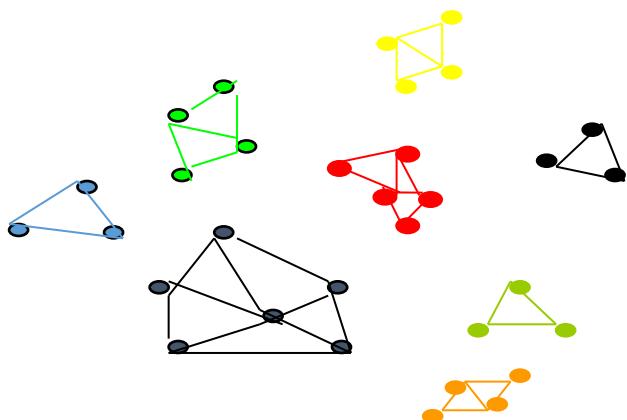
Method: {'alternate', 'pam'}, default: 'alternate'
Which algorithm to use. 'alternate' is faster while 'pam'
is more accurate.

Init: {'random', 'heuristic', 'k-medoids++', 'build'}, or
array-like of shape
(n_clusters, n_features), optional, default: 'heuristic'
Specify medoid initialization method. 'heuristic' picks the
n_clusters points with the smallest sum distance to every
other point. 'k-medoids++' follows an approach based on k-
means++, and in general, gives initial medoids which are
more separated than those generated by the other methods.
'build' is a greedy initialization used in the original PAM
algorithm. Often 'build' is more efficient but slower than
other initializations on big datasets and it is also very
non-robust, if there are outliers in the dataset, use
another initialization.

```
>>> from sklearn_extra.cluster import Kmedoids  
>>> import numpy as np  
>>> X = np.asarray([[1, 2], [1, 4], [1, 0], ... [4, 2],  
[4, 4], [4, 0]])  
>>> kmmedoids = KMedoids(n_clusters=2).fit(X)  
>>> kmmedoids.labels_  
array([0, 0, 0, 1, 1, 1])  
>>> kmmedoids.predict([[0,0], [4,4]]) array([0, 1])  
>>> kmmedoids.cluster_centers_  
array([[1., 2.], [4., 2.]])
```

Density-Based Clustering

- Density-based Clustering algorithms use density-estimation techniques:
 - to obtain density functions over the space of the attributes; then clusters are identified as areas whose density is above a certain threshold θ (DENCLUE's Approach)
 - to create a proximity graph which connects objects whose density is above a certain threshold θ in the neighborhood of an object; then clustering algorithms identify contiguous, connected subsets in the graph which are dense (DBSCAN's Approach). DBSCAN employs a naïve density estimation approach to estimate the density of dataset points



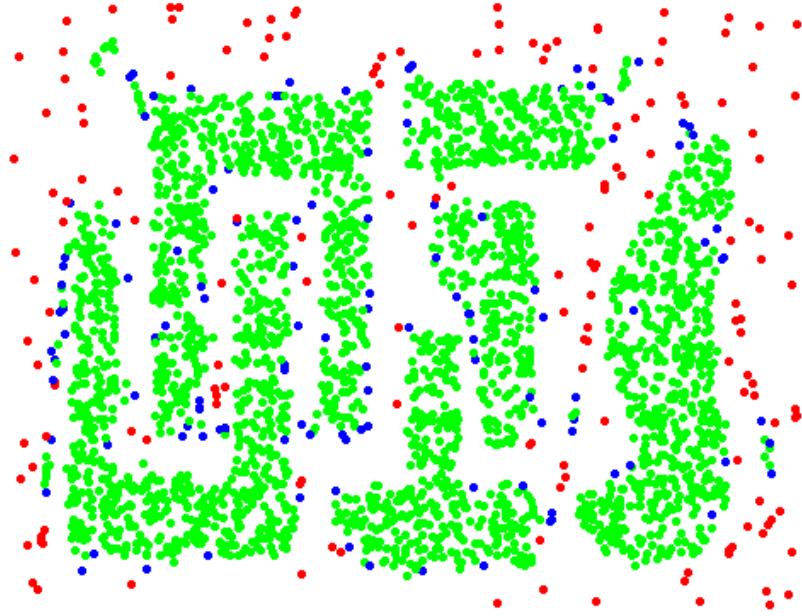
Density Estimation

- In probability and statistics, **density estimation** is the construction of an estimate, based on observed data, of **an unobservable underlying probability density function**. The unobservable density function is thought of as the density according to which a large population is distributed; **the data are usually thought of as a random sample from that population** (Wikipedia)
- Different Density Estimation Approaches:
 - Naïve Approaches which estimate density by counting the number of objects in grids or other shapes.
 - Parametric approaches which employ Gaussian or other models as density functions.
 - Non-parametric approach which use the points in the dataset in influence functions to estimate the density in a query point. Most popular approach: Kernel density estimation

DBSCAN



Original Points

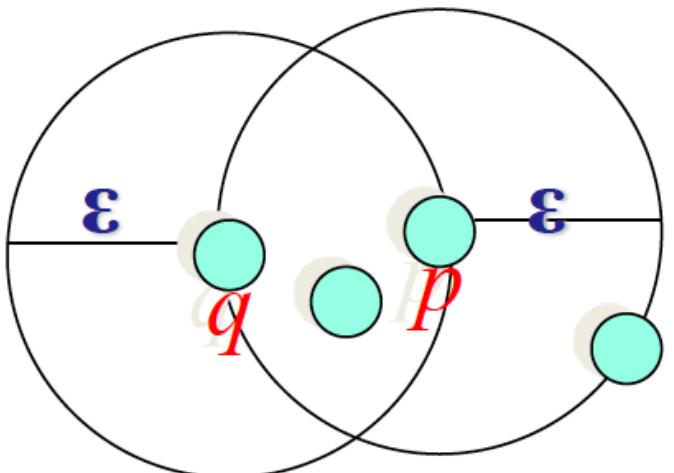


Point types: **core**, **border** and **noise**

DBSCAN Result, Eps = 10, MinPts = 4

DBSCAN

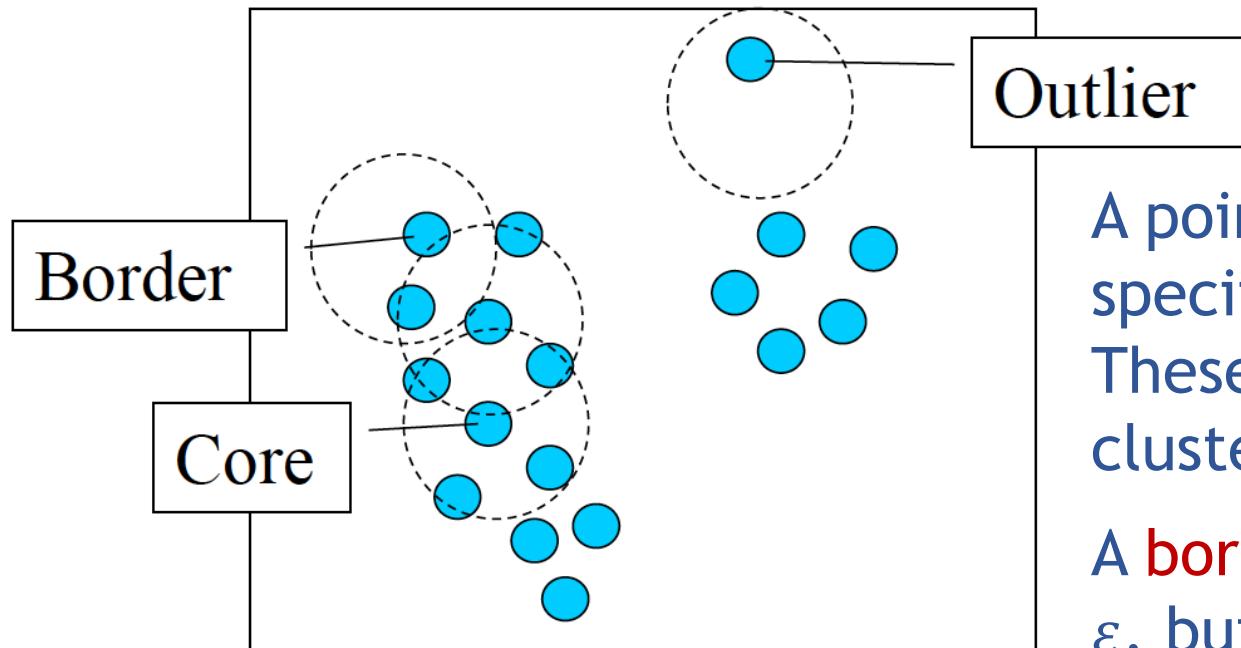
- Density-based spatial clustering of applications with noise (DBSCAN)
 - Density Definition
 - ε -Neighborhood: Objects within a radius of ε from an object
 - “High density”: ε -Neighborhood of an object contains at least MinPts of objects



ε -Neighborhood of p
 ε -Neighborhood of q
Density of p is “high” (MinPts=4)
Density of p is “low” (MinPts=4)

DBSCAN

- Core, Border and Noise Points



$$\epsilon = 1 \text{ unit}, \text{MinPts} = 5$$

Input: ϵ and MinPts

DBSCAN categorizes the data objects into three exclusive groups

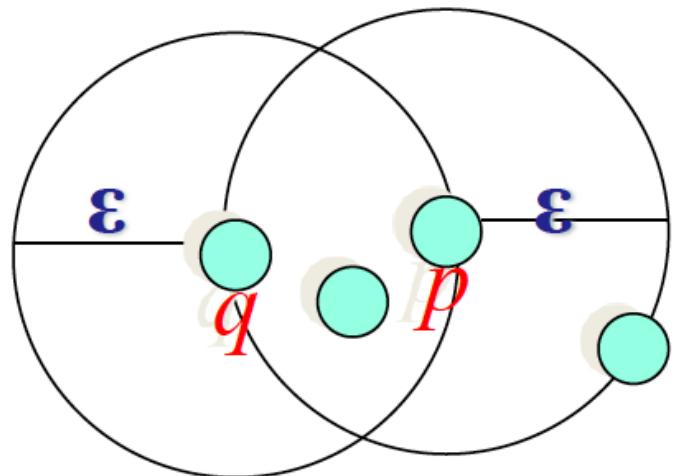
A point is a **core point** if it has more than a specified number of points (MinPts) within ϵ : These are points that are at the interior of a cluster.

A **border point** has fewer than MinPts within ϵ , but is in the neighborhood of a core point.

A **noise point** is any point that is not a core point nor a border point.

Density-Reachability

- Directly Density-Reachable
 - An object q is directly density-reachable from object p if p is a **core object** and q is in p 's ε -neighborhood.

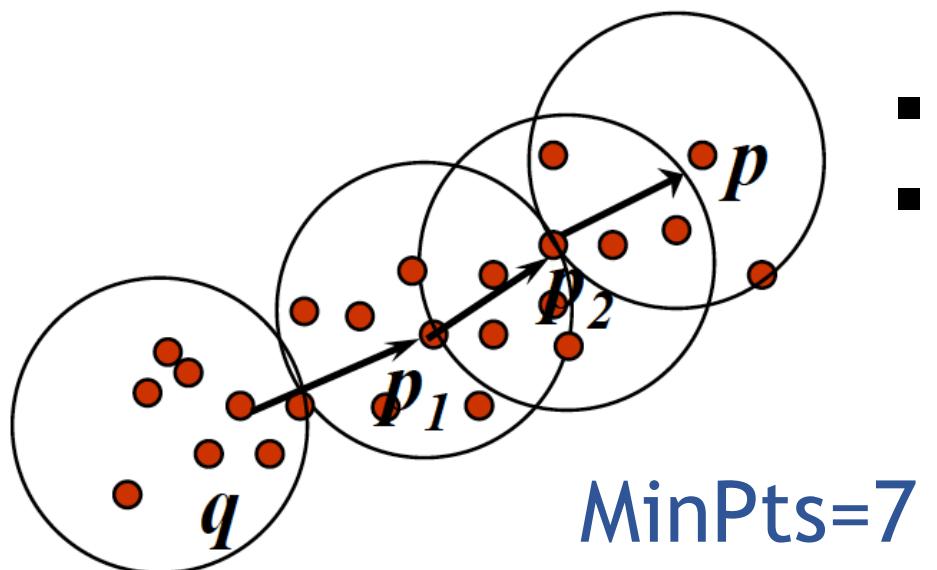


MinPts=4

- q is directly density-reachable from p
- p is not directly density-reachable from q
- Density-reachability is **asymmetric**

Density-Reachability

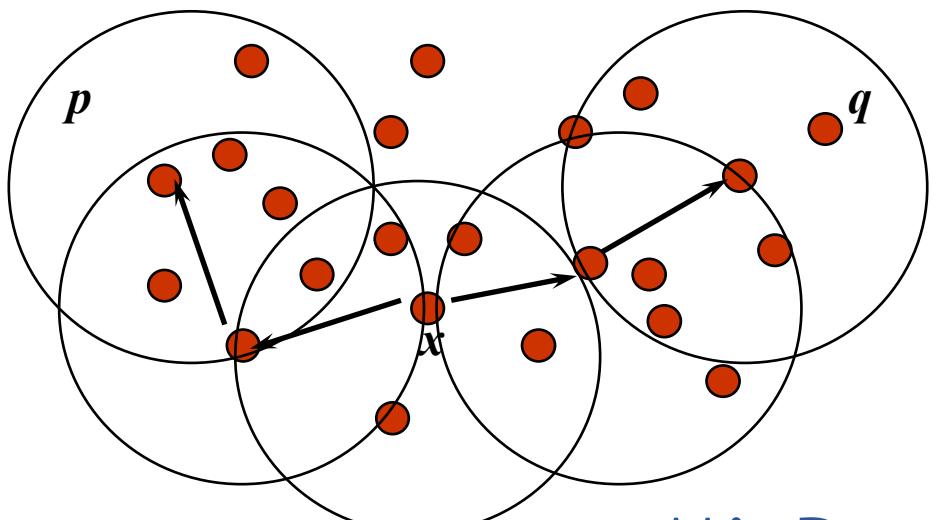
- Density-Reachable (directly and indirectly):
 - A point p is directly density-reachable from p_2
 - p_2 is directly density-reachable from p_1
 - p_1 is directly density-reachable from q
 - $q \rightarrow p_1 \rightarrow p_2 \rightarrow p$ form a chain



- p is (indirectly) density-reachable from q
- q is not density-reachable from p

Density-Reachability

- Density-Connected:
 - A point p is density-connected to a point q wrt. ε , MinPts if there is a point x such that both p and q are density-reachable from x wrt. x and MinPts.



- p is (indirectly) density-reachable from x
- q is (indirectly) density-reachable from x

DBSCAN Algorithm

- Relies on a density-based notion of cluster: A cluster is defined as a maximal set of **density-connected points**
- Capable to discover clusters of arbitrary shape in spatial datasets with noise

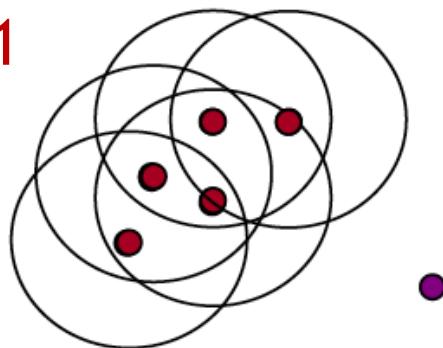
1. Arbitrarily select a point p
2. Retrieve all points density-reachable from p wrt Eps and MinPts
3. If p is a core point, a cluster is formed
4. If p is not a core point, no points are density-reachable from p and DBSCAN visits the next point of the database
5. Continue the process until all of the points have been processed

Remark: Some bookkeeping is needed to make sure that only points that have not been assigned to a cluster yet, will be used in step 2.

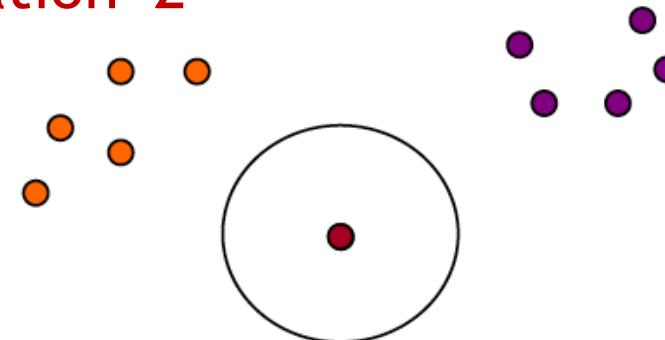
DBSCAN Algorithm

- Example
 - $\varepsilon = 2$, MinPts=3

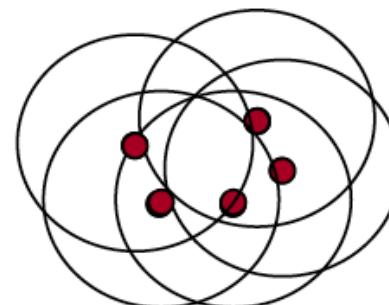
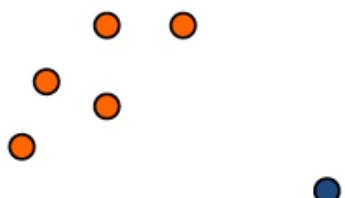
Iteration=1



Iteration=2

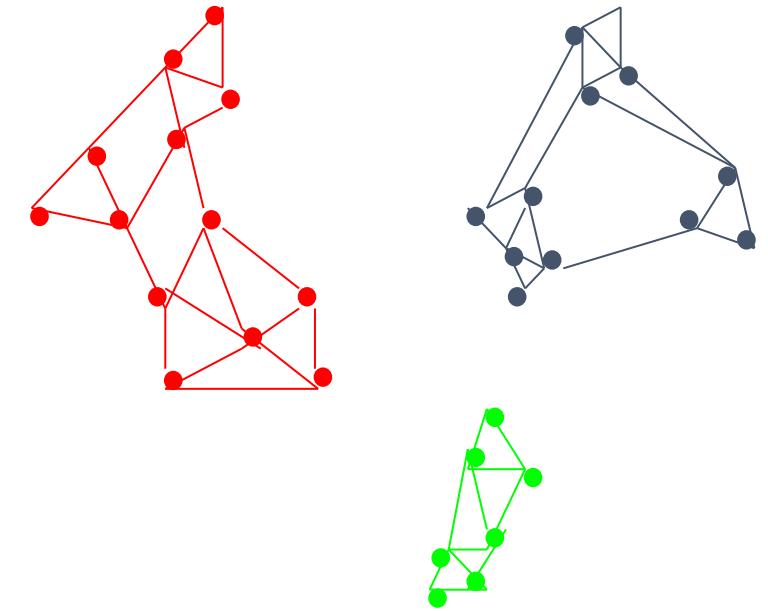


Iteration=3



DBSCAN Algorithm (Graph View)

1. Create a graph whose nodes are the points to be clustered
2. For each core-point c create an edge from c to every point p in the ε -neighborhood of c
3. Set N to the nodes of the graph
4. If N does not contain any core points terminate
5. Pick a core point c in N
6. Let X be the set of nodes that can be reached from c by going forward (i.e., **density-reachable**)
 1. create a cluster containing $X \cup \{c\}$
 2. $N=N/(X \cup \{c\})$
7. Continue with step 4



Remark: points that are not assigned to any cluster are outliers.

DBSCAN Exercise

A dataset consisting of object A, B, C, D, E, F with the following distance matrix is given:

Distance	A	B	C	D	E	F
A	0	1	2	4	6	7
B		0	3	8	9	10
C			0	11	12	13
D				0	14	15
E					0	16
F						0

Assume DBSCAN is run for this dataset with **MinPts=3** and $\varepsilon=5$. How many clusters will DBSCAN return and how do they look like? Which objects are outliers and border points in the clustering result obtained earlier?

DBSCAN Exercise

Distance	A	B	C	D	E	F
A	0	1	2	4	6	7
B		0	3	8	9	10
C			0	11	12	13
D				0	14	15
E					0	16
F						0

MinPts=3 and $\varepsilon=5$

ε -neighbor of A: A, B, C, D \rightarrow Core point \rightarrow DBSCAN Cluster: {A, B, C, D}

ε -neighbor of B: A, B, C \rightarrow Core point

ε -neighbor of C: A, B, C \rightarrow Core point

ε -neighbor of D: A, D \rightarrow Border point

ε -neighbor of E: E \rightarrow Noise Point

ε -neighbor of F: F \rightarrow Noise Point

DBSCAN: Sensitive to Parameters

Figure 8. DBScan results for DS1 with MinPts at 4 and Eps at (a) 0.5 and (b) 0.4.

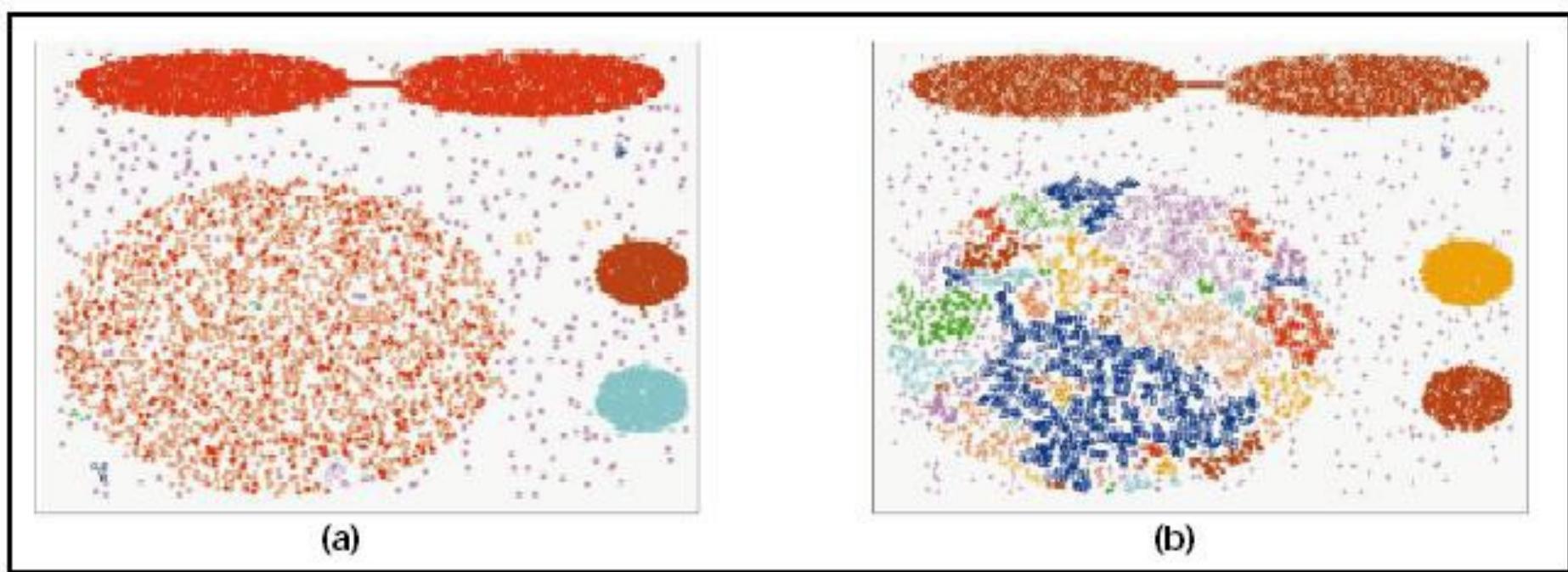
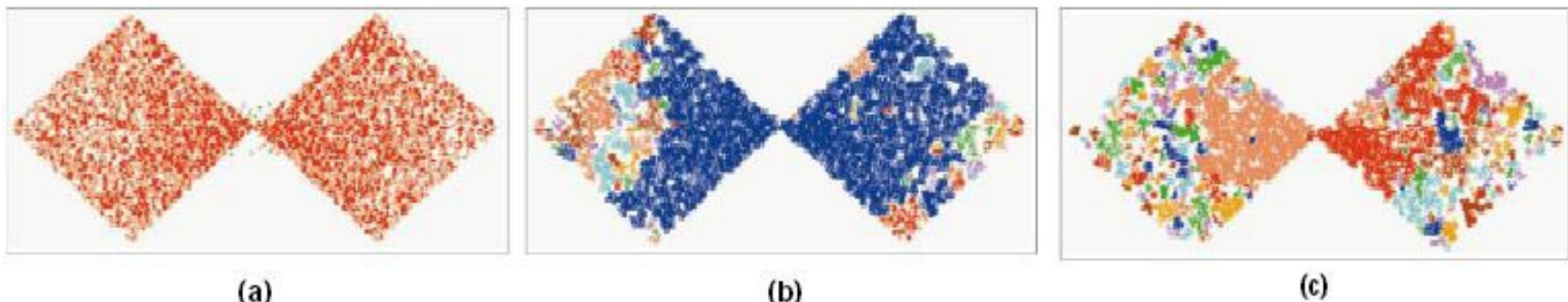
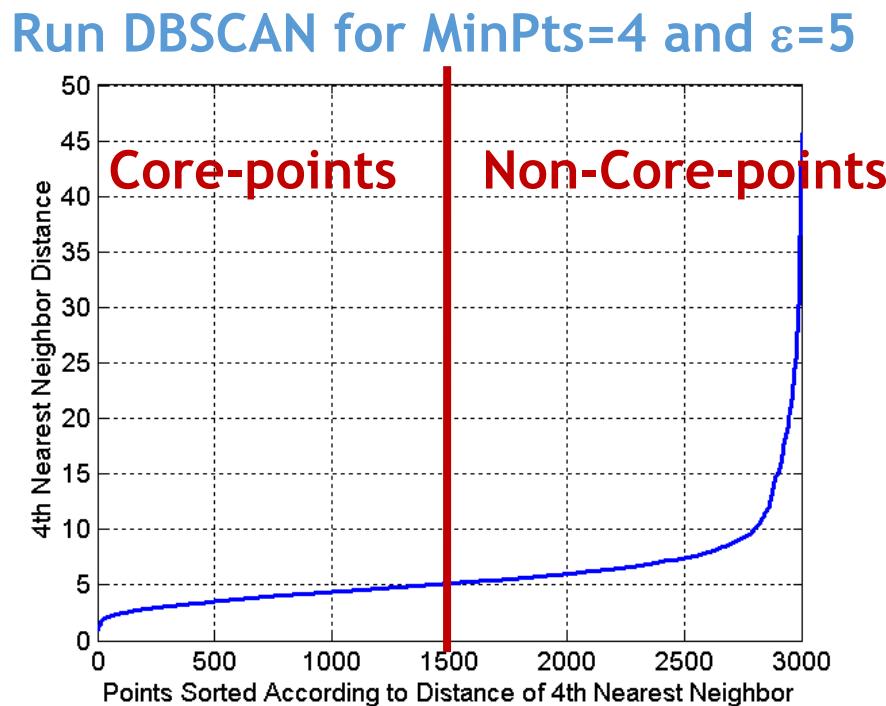


Figure 9. DBScan results for DS2 with MinPts at 4 and Eps at (a) 5.0, (b) 3.5, and (c) 3.0.



DBSCAN: Determining ε and MinPts

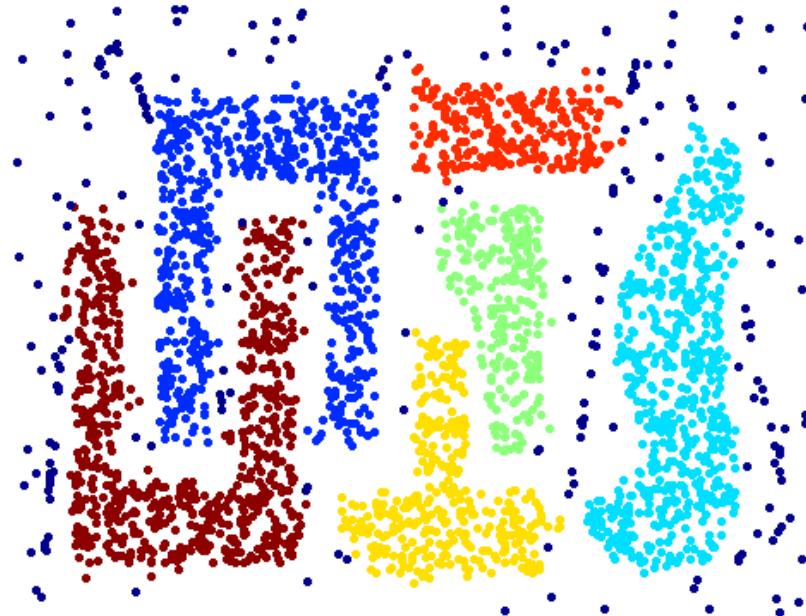
- Idea is that for points in a cluster, their k^{th} nearest neighbors are at roughly the same distance
- Noise points have the k^{th} nearest neighbor at farther distance
- So, plot sorted distance of every point to its k^{th} nearest neighbor



When DBSCAN Works Well



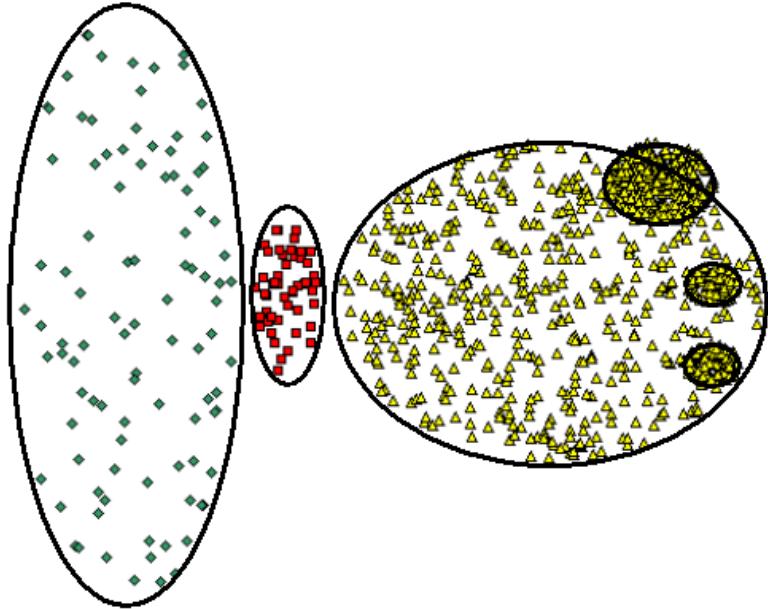
Original Points



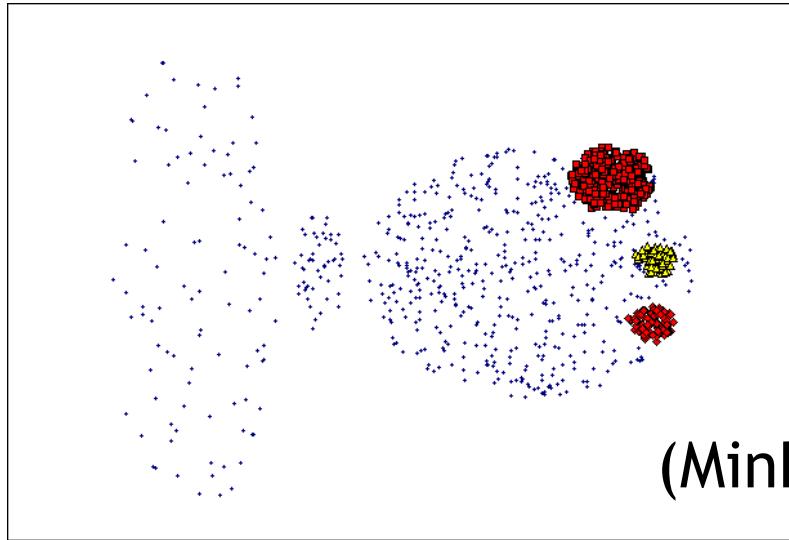
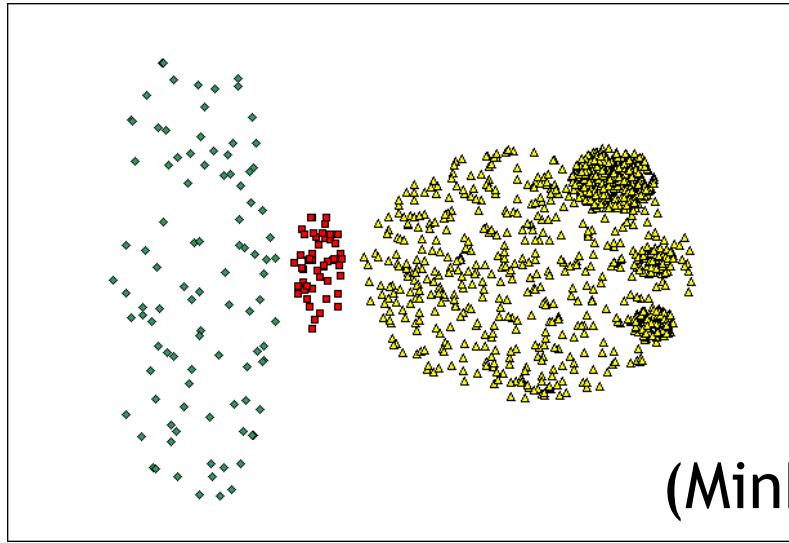
Clusters

- Resistant to Noise
- Supports Outliers
- Can handle clusters of different shapes and sizes

When DBSCAN Does Not Work Well



- Cannot handle varying densities
- Sensitive to parameters—hard to determine the correct set of parameters



DBSCAN in Sklearn

```
class sklearn.cluster.DBSCAN(eps=0.5, *, min_samples=5, metric='euclidean', metric_params=None, algorithm='auto', leaf_size=30, p=None, n_jobs=None)
```

eps: float, default=0.5

The maximum distance between two samples for one to be considered as in the neighborhood of the other.

min_samples: int, default=5

The number of samples (or total weight) in a neighborhood for a point to be considered as a core point. This includes the point itself. If `min_samples` is set to a higher value, DBSCAN will find denser clusters, whereas if it is set to a lower value, the found clusters will be more sparse.

metric: str, or callable, default='euclidean'

The metric to use when calculating distance between instances in a feature array.

```
>>> from sklearn.cluster import DBSCAN
>>> import numpy as np
>>> X = np.array([[1, 2], [2, 2], [2, 3], ... [8, 7], [8, 8], [25, 80]])
>>> clustering = DBSCAN(eps=3, min_samples=2).fit(X)
>>> clustering.labels_
array([ 0,  0,  0,  1,  1, -1])
>>> clustering
DBSCAN(eps=3, min_samples=2)
```

metric	Function
'cityblock'	metrics.pairwise.manhattan_distances
'cosine'	metrics.pairwise.cosine_distances
'euclidean'	metrics.pairwise.euclidean_distances
'haversine'	metrics.pairwise.haversine_distances
'l1'	metrics.pairwise.manhattan_distances
'l2'	metrics.pairwise.euclidean_distances
'manhattan'	metrics.pairwise.manhattan_distances
'nan_euclidean'	metrics.pairwise.nan_euclidean_distances

Summary of Density-Based Clustering

- +: can (potentially) discover clusters of arbitrary shape
- +: not sensitive to outliers and supports outlier detection
- +: can handle noise
- +-: medium algorithm complexities $O(n^2)$, $O(n * \log(n))$
- -: finding good density estimation parameters is frequently difficult; more difficult than using K-means
- -: usually, does not do well in clustering high-dimensional datasets
- -: cluster models are not well understood (yet)



COSC 3337
Data Science I
Section 14623

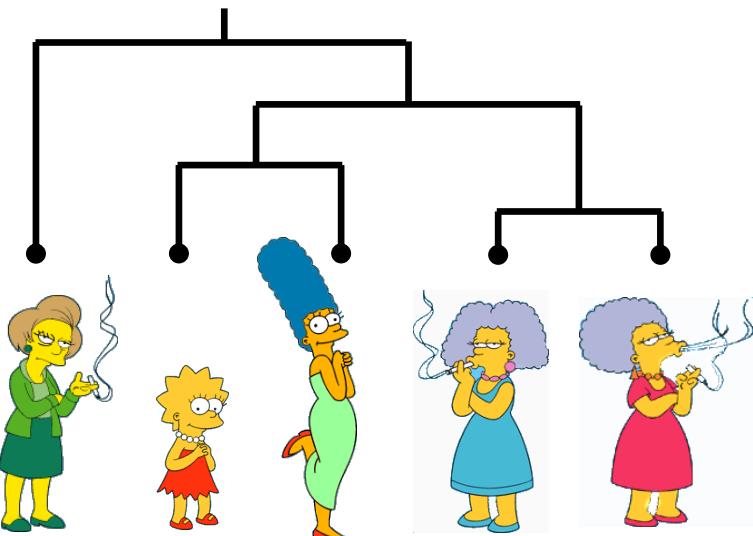
Clustering (contd.)

Instructor: Jingchao Ni
Fall 2024

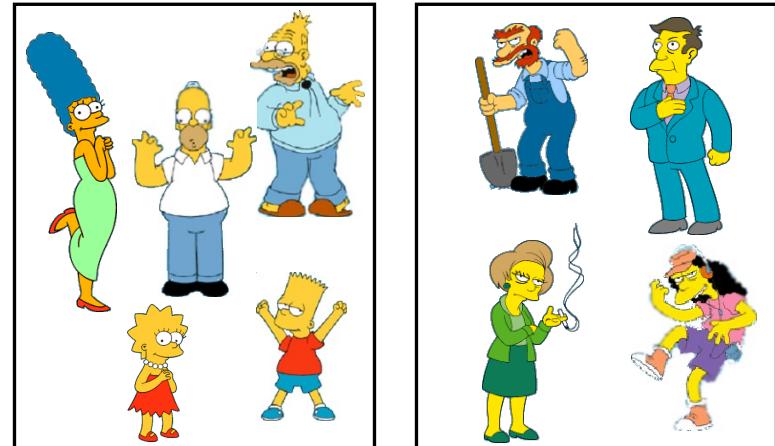
Two Types of Clustering

- **Partitional algorithms:** Construct various partitions and then evaluate them by some criterion
 - **Hierarchical algorithms:** Create a hierarchical decomposition of the set of objects using some criterion

Hierarchical

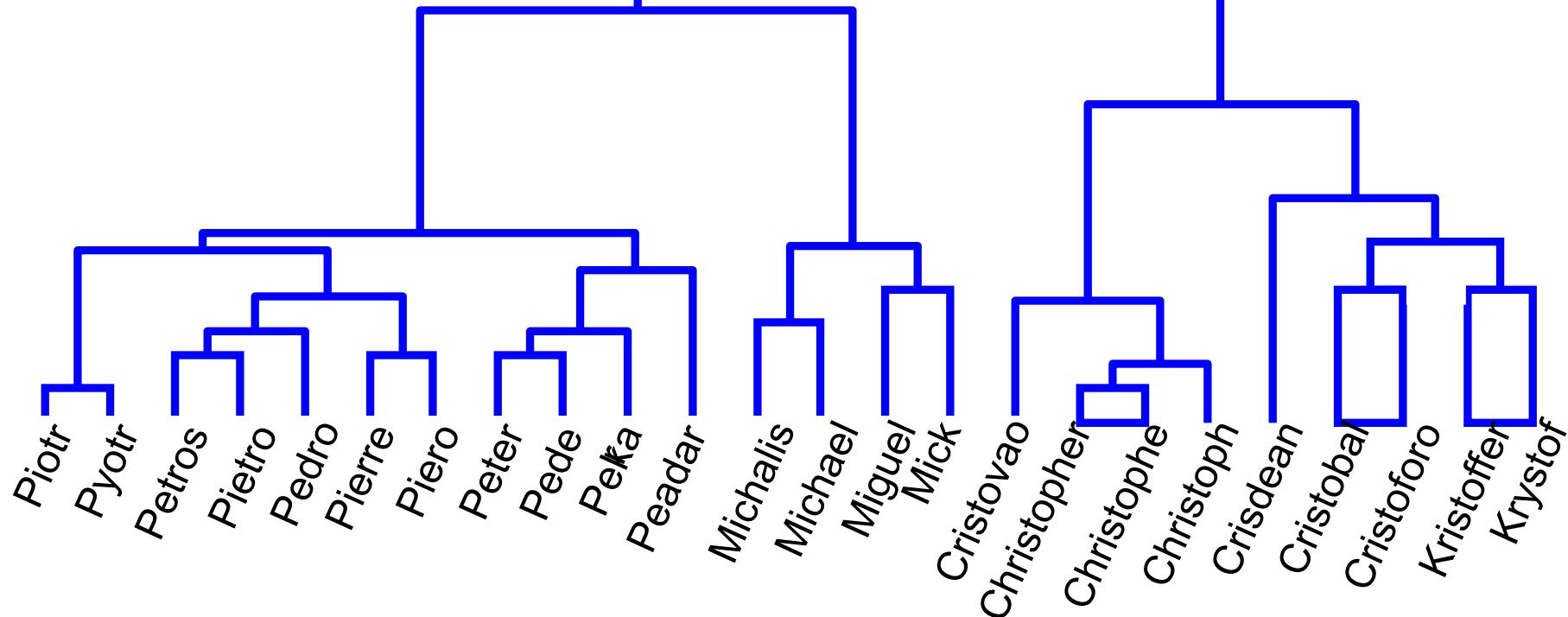


Partitional



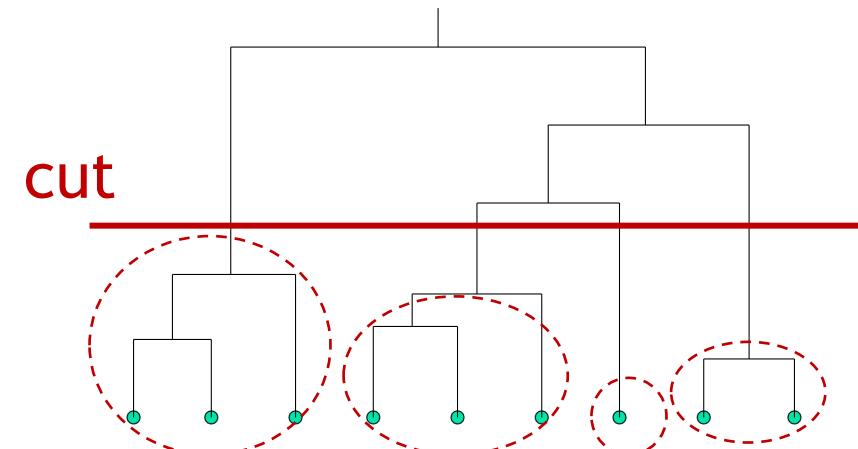
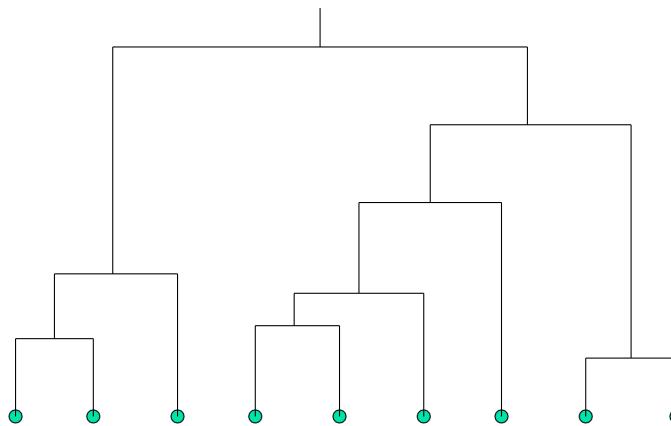
Hierarchical Clustering

- A demonstration of hierarchical clustering using string edit distance



HC Produces Dendrogram

- A diagram representing a tree. It illustrates the arrangement of the clusters produced by the corresponding analyses
 - Decompose data objects into a several levels of nested partition (tree of clusters), called a **dendrogram**
 - A clustering of the data objects is obtained by cutting the dendrogram at the desired level, then each connected component forms a cluster
 - A dendrogram indicates how clusters are merged

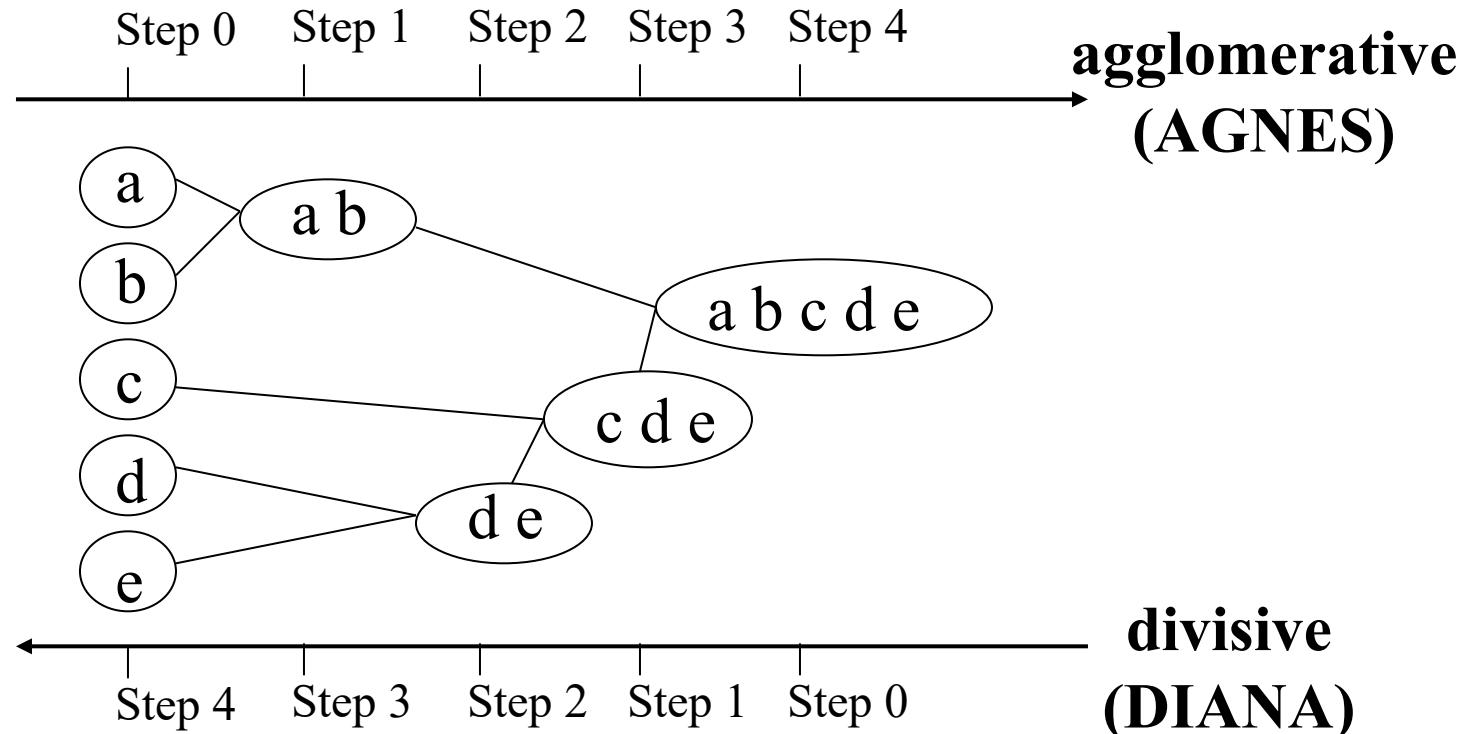


Hierarchical Clustering

- Two main types of hierarchical clustering algorithms:
 - Agglomerative (bottom-up):
 - Start with the points as individual clusters
 - At each step, merge the closest pair of clusters until only one cluster (or k clusters) left
 - Divisive (top-down):
 - Start with one, all-inclusive cluster
 - At each step, split a cluster until each cluster contains an individual point (or there are k clusters)
- Traditional hierarchical algorithms use a similarity or distance matrix
 - Merge or split one cluster at a time

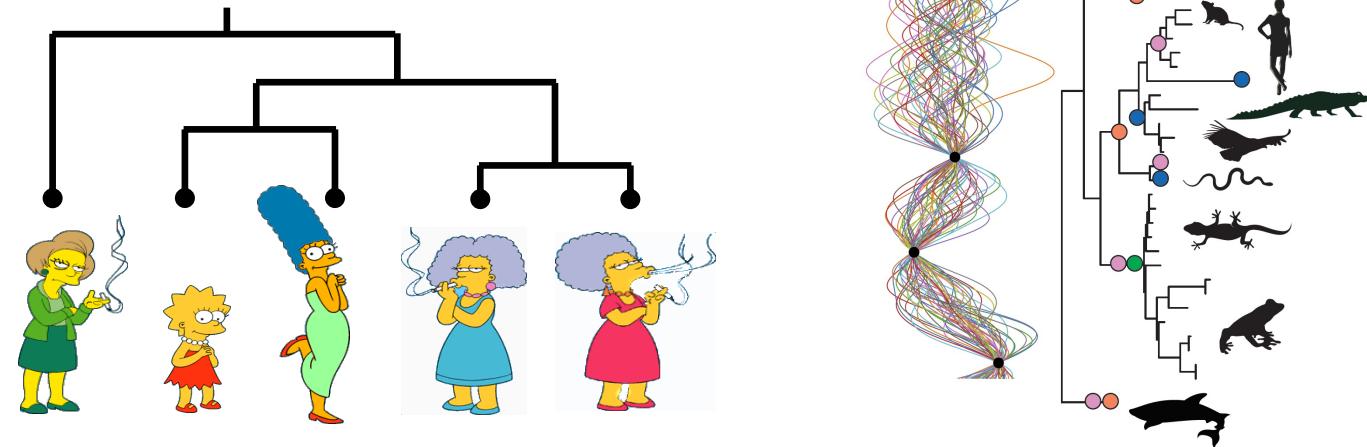
Hierarchical Clustering

- Use distance matrix as clustering criteria. This method does not require the number of clusters k as an input, but needs a termination condition



Advantages of Hierarchical Clustering

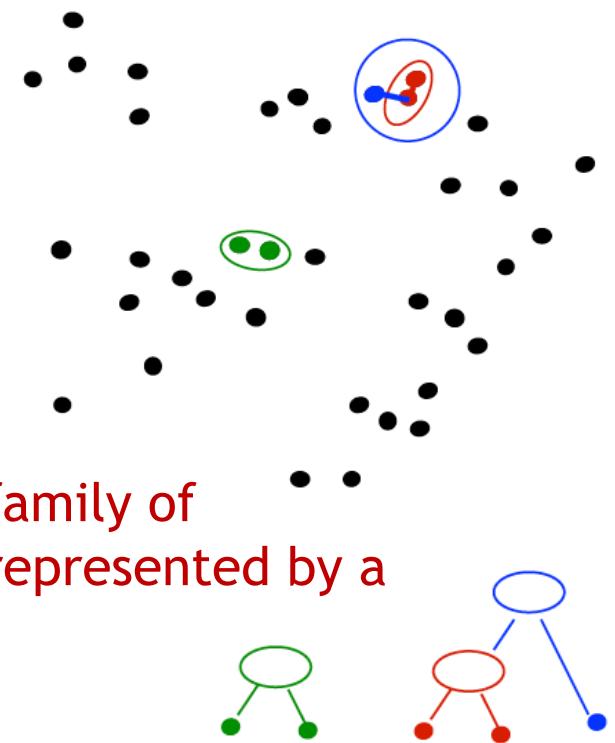
- It constructs sets of clusters, and not a single cluster
- It does not have to assume any particular number of clusters: Any desired number of clusters can be obtained by ‘cutting’ the dendrogram at the proper level



- They may correspond to meaningful taxonomies
- Example in biological sciences (e.g., animal kingdom, phylogeny reconstruction)

Agglomerative Clustering Algorithm

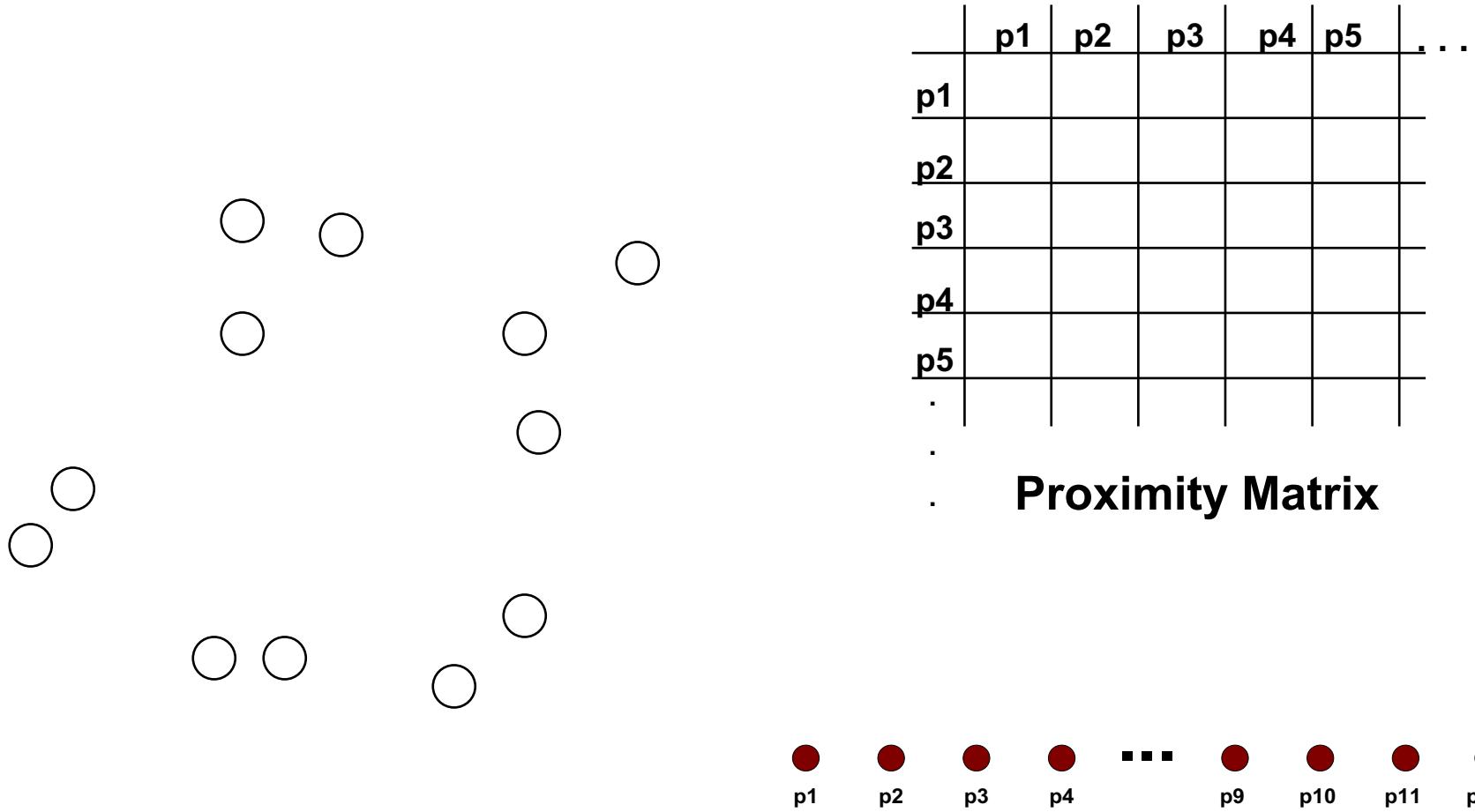
- Most popular hierarchical clustering technique
- Basic algorithm is straightforward
 1. Compute the proximity matrix
 2. Let each data point be a cluster
 3. **Repeat**
 4. Merge the two closest clusters
 5. Update the proximity matrix
 6. **Until only a single cluster remains**
- Key operation is the computation of the proximity of two clusters
- Different approaches to define the distance between clusters distinguish the different algorithms



Produces a family of clusterings represented by a dendrogram

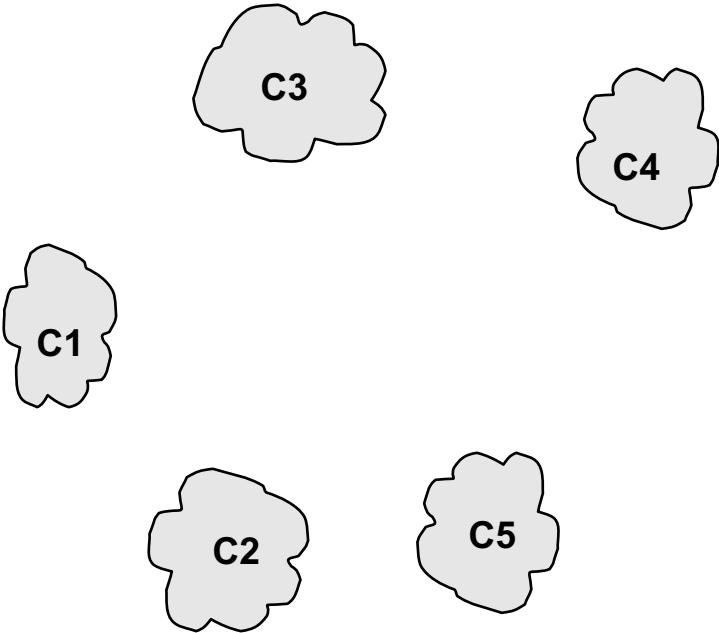
Example: Starting Situation

- Start with clusters of individual points and a proximity matrix



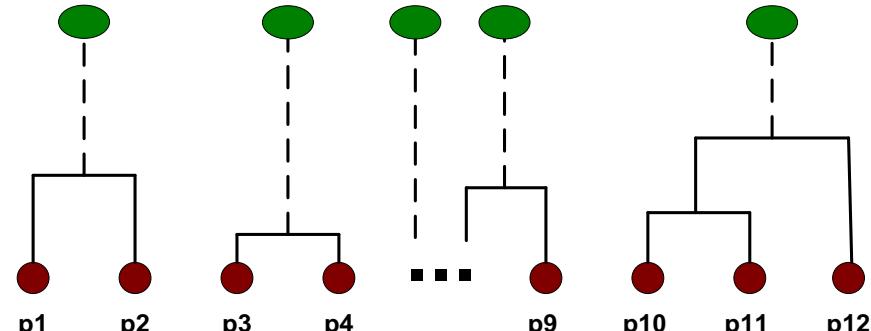
Example: Intermediate Situation

- After some merging steps, we have some clusters



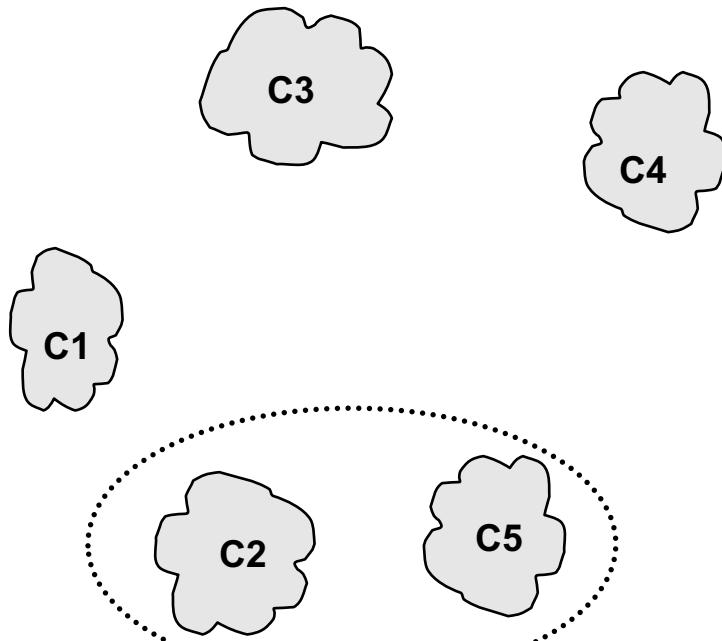
	C1	C2	C3	C4	C5
C1					
C2					
C3					
C4					
C5					

Proximity Matrix



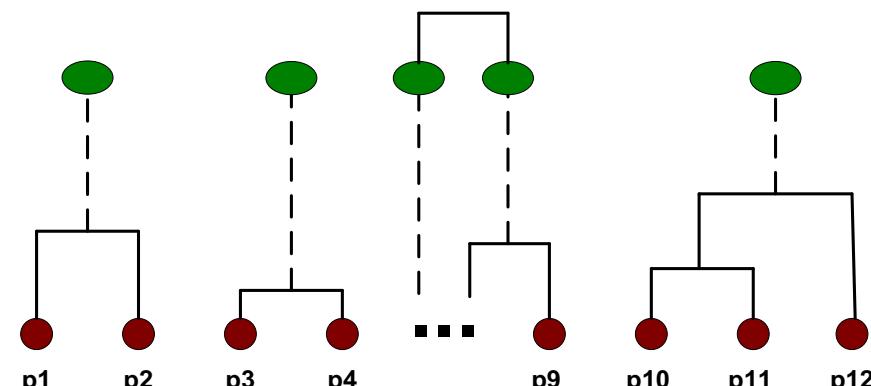
Example: Intermediate Situation

- We want to merge the two closest clusters (C_2 and C_5) and update the proximity matrix



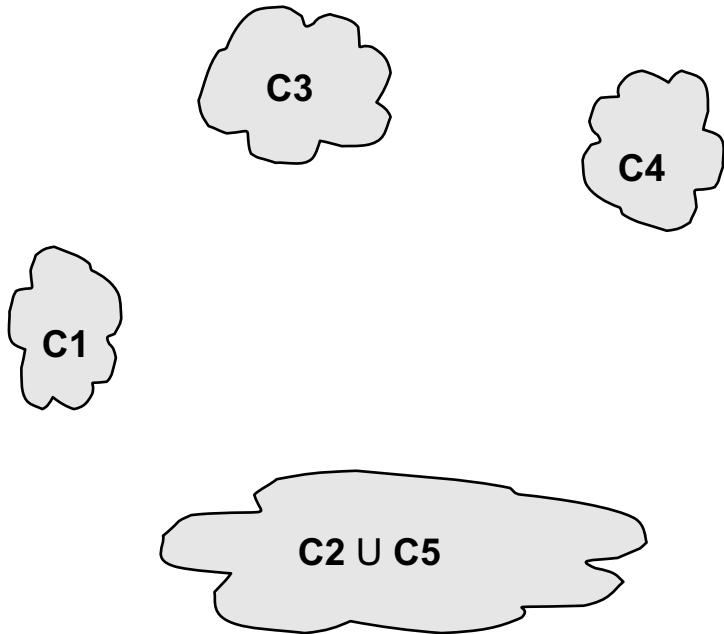
	C1	C2	C3	C4	C5
C1					
C2					
C3					
C4					
C5					

Proximity Matrix



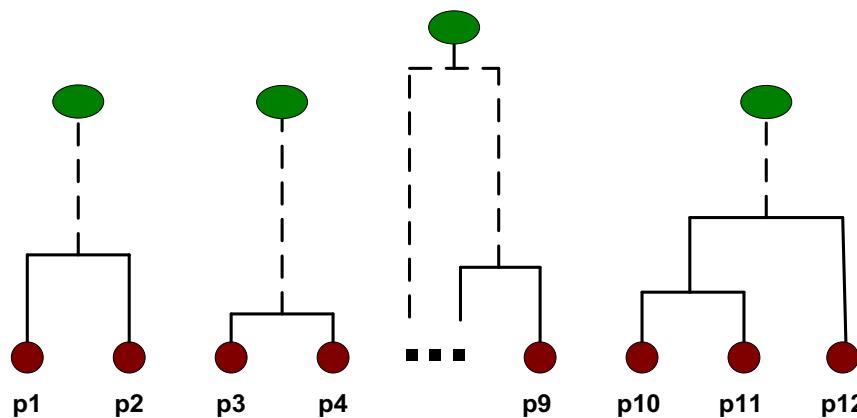
Example: After Merging

- The question is “How do we update the proximity matrix?”

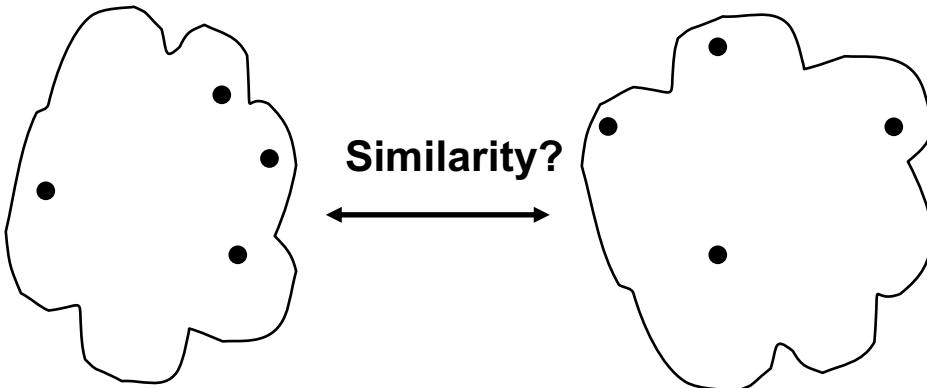


		C2 U			
		C1	C5	C3	C4
C1	C1	?			
	C2 U C5	?	?	?	?
		C3	?		
		C4	?		

Proximity Matrix



How to Define Inter-Cluster Distance

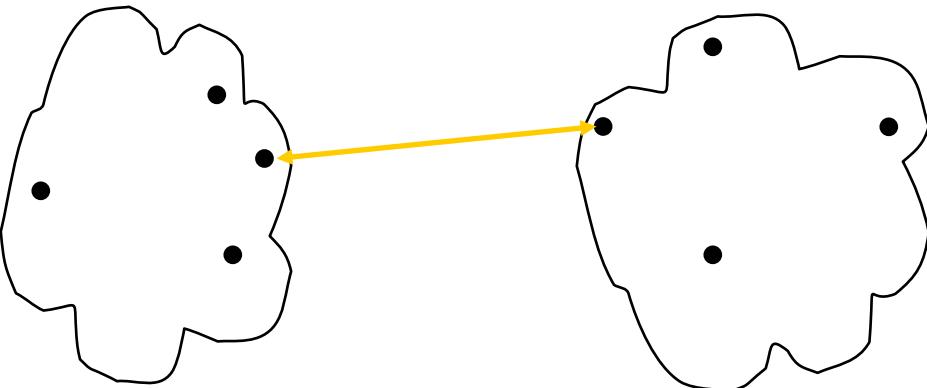


- MIN
- MAX
- Group Average
- Distance Between Centroids
- Other methods driven by an objective function
 - Ward's Method uses squared error

	p1	p2	p3	p4	p5	...
p1						
p2						
p3						
p4						
p5						
.						
.						
.						

Proximity Matrix

How to Define Inter-Cluster Distance

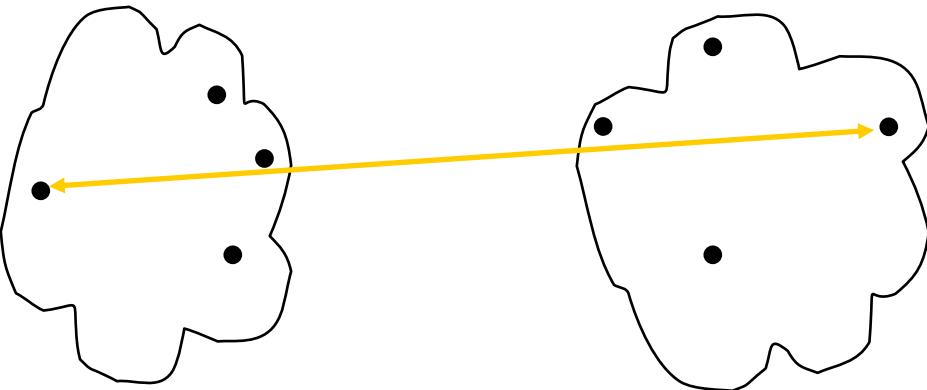


- **MIN**
- **MAX**
- **Group Average**
- **Distance Between Centroids**
- Other methods driven by an objective function
 - Ward's Method uses squared error

	p1	p2	p3	p4	p5	...
p1						
p2						
p3						
p4						
p5						
.						.
.						.

Proximity Matrix

How to Define Inter-Cluster Distance

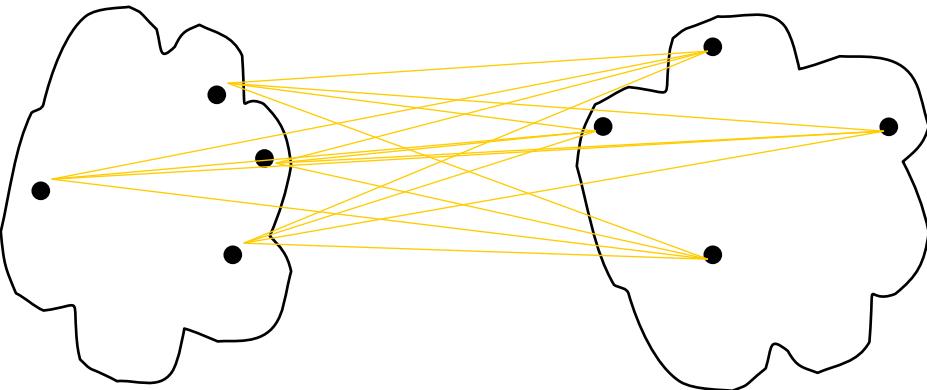


- MIN
- MAX
- Group Average
- Distance Between Centroids
- Other methods driven by an objective function
 - Ward's Method uses squared error

	p1	p2	p3	p4	p5	...
p1						
p2						
p3						
p4						
p5						
.						.
.						.
.						.

Proximity Matrix

How to Define Inter-Cluster Distance

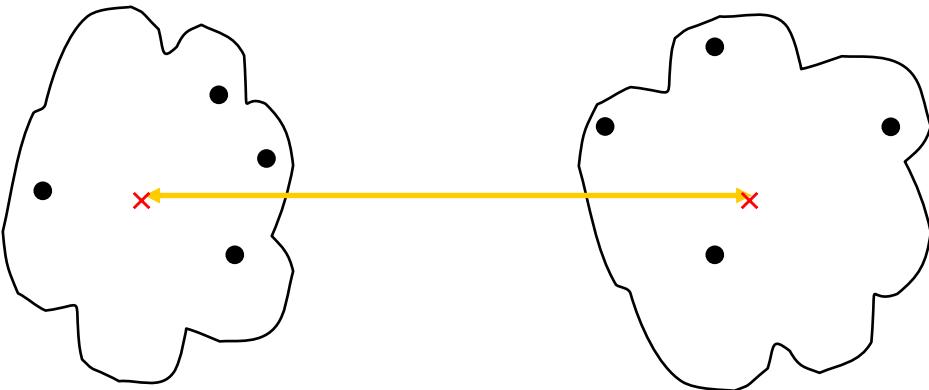


- MIN
- MAX
- **Group Average**
- Distance Between Centroids
- Other methods driven by an objective function
 - Ward's Method uses squared error

	p1	p2	p3	p4	p5	...
p1						
p2						
p3						
p4						
p5						
.						
.						
.						

Proximity Matrix

How to Define Inter-Cluster Distance



- MIN
- MAX
- Group Average
- **Distance Between Centroids**
- Other methods driven by an objective function
 - Ward's Method uses squared error

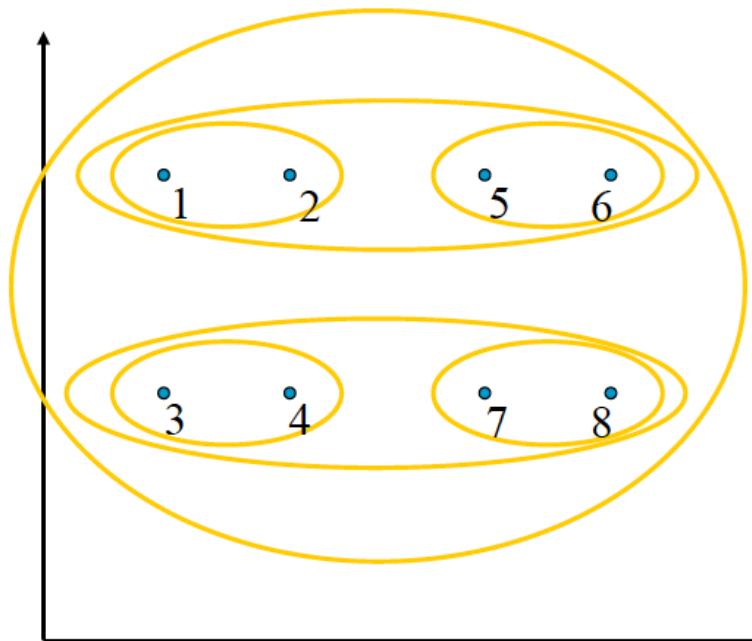
	p1	p2	p3	p4	p5	...
p1						
p2						
p3						
p4						
p5						
.						
.						
.						

Proximity Matrix

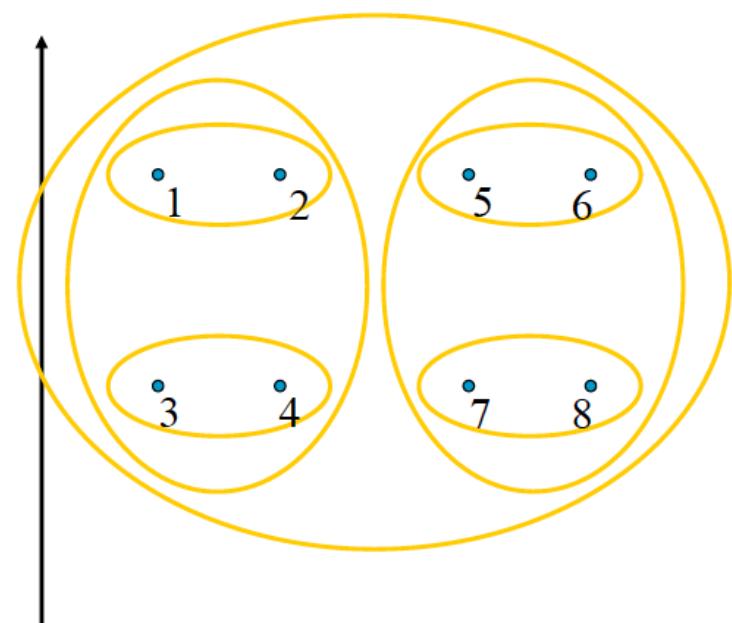
Agglomerative Clustering

- Different choices create different clustering behaviors

Closest pair (MIN Distance)
(single-link clustering)



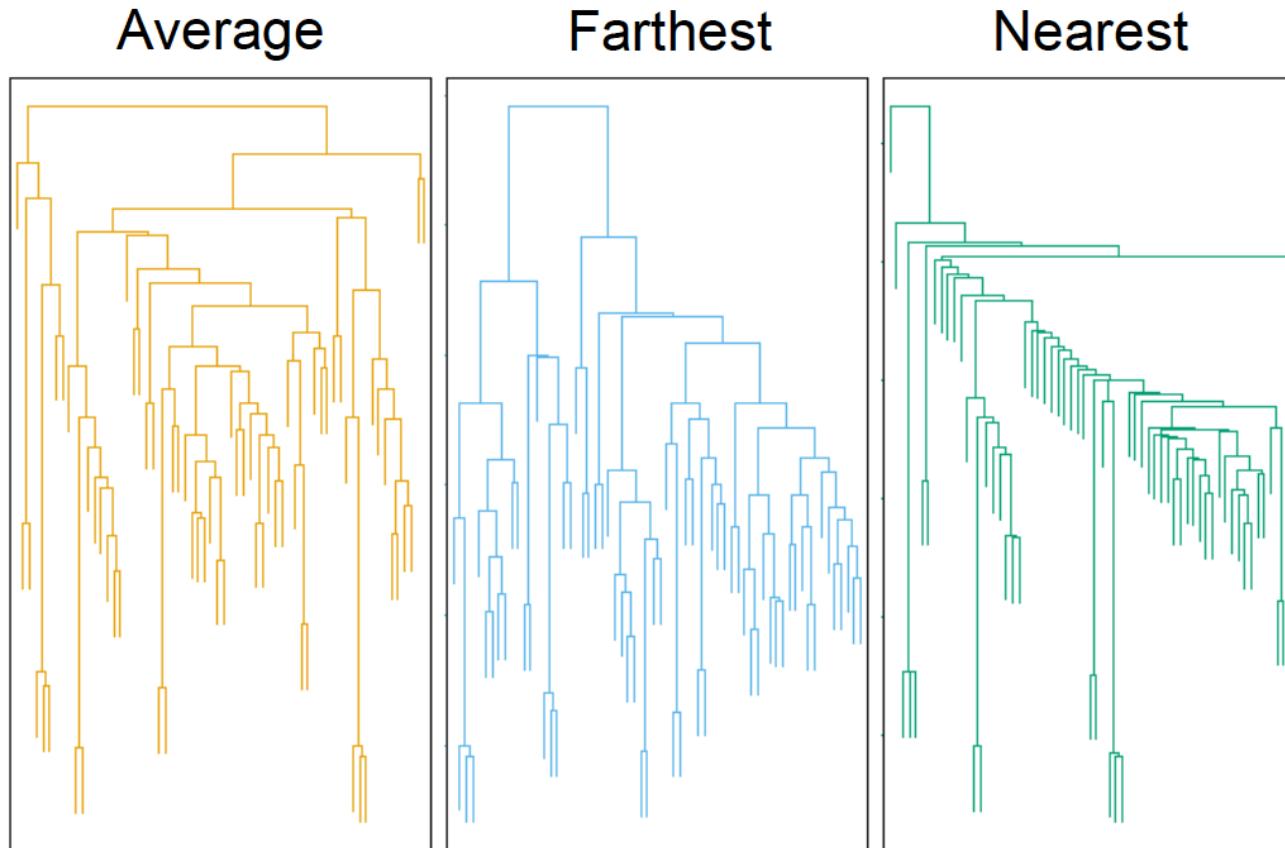
Farthest pair (MAX Distance)
(complete-link clustering)



[Pictures from Thorsten Joachims]

Agglomerative Clustering

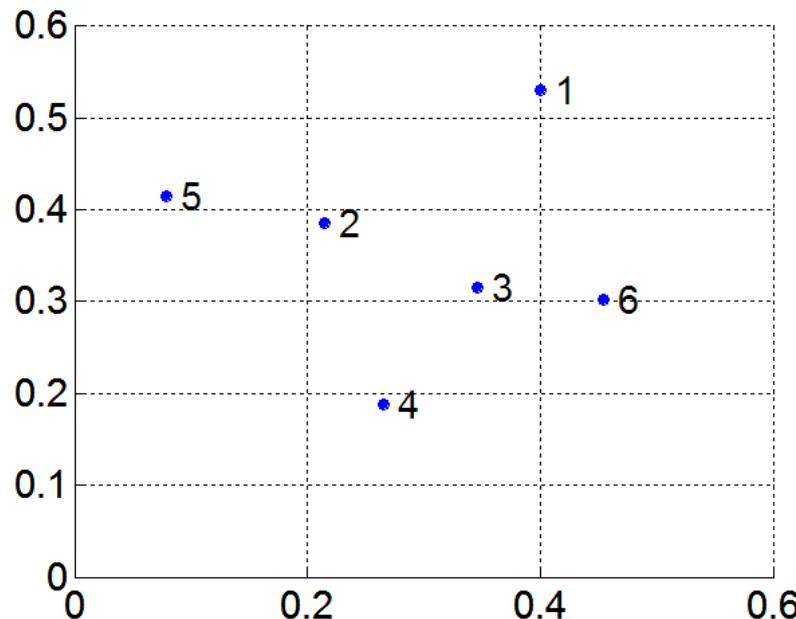
- Different choices create different clustering behaviors



[Pictures from Hastie et al.]

MIN Distance (Single Link) Clustering

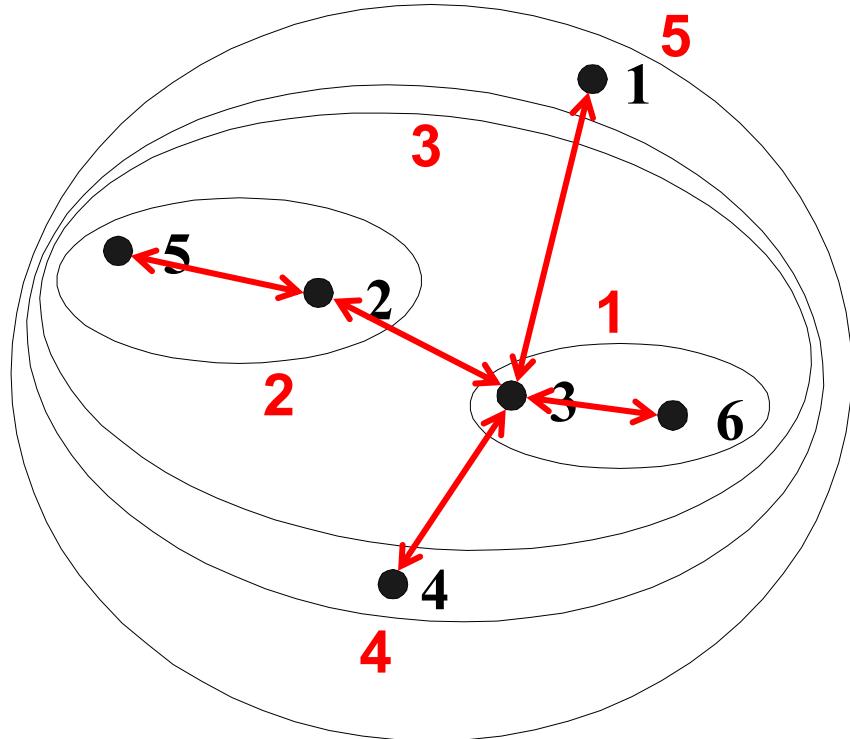
- Proximity of two clusters is based on the two closest points in different clusters
 - Determined by one pair of points, i.e., by one link in the proximity graph
- Example:



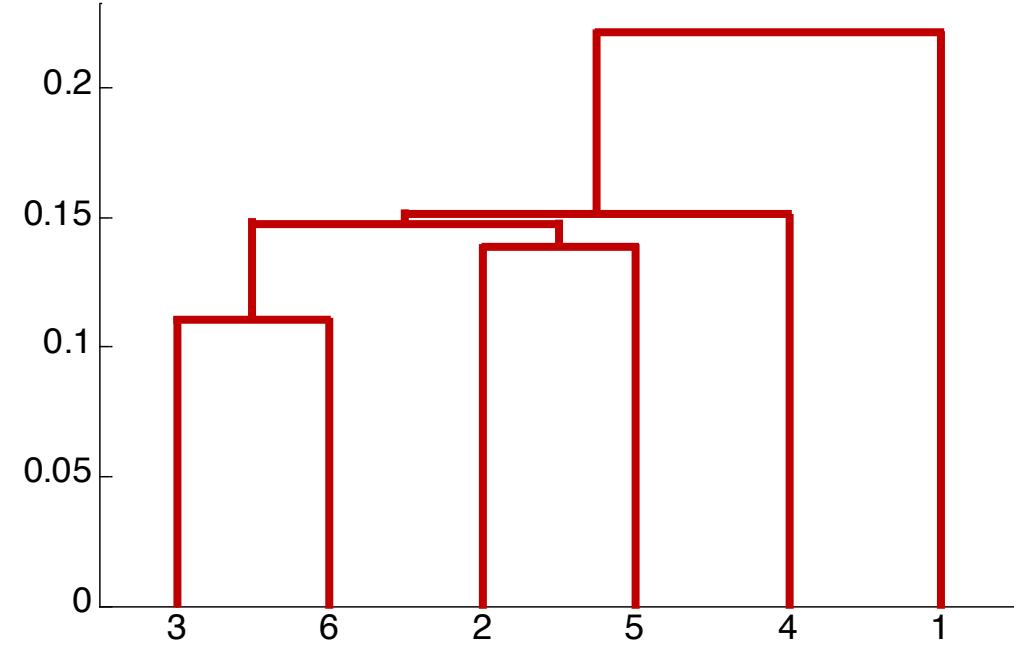
Distance Matrix

	p1	p2	p3	p4	p5	p6
p1	0.00	0.24	0.22	0.37	0.34	0.23
p2	0.24	0.00	0.15	0.20	0.14	0.25
p3	0.22	0.15	0.00	0.15	0.28	0.11
p4	0.37	0.20	0.15	0.00	0.29	0.22
p5	0.34	0.14	0.28	0.29	0.00	0.39
p6	0.23	0.25	0.11	0.22	0.39	0.00

MIN Distance (Single Link) Clustering

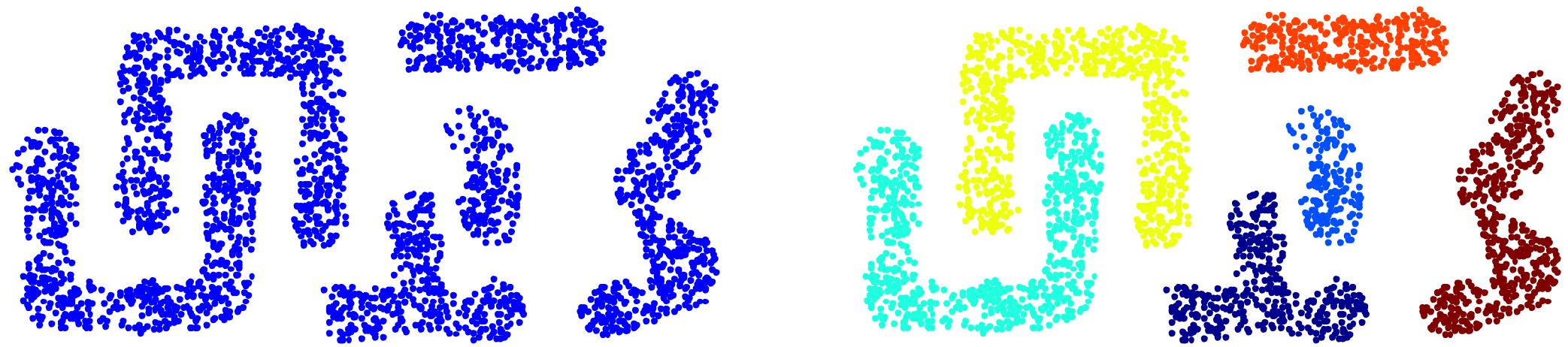


Nested Clusters



Dendrogram

MIN Distance (Single Link) Clustering

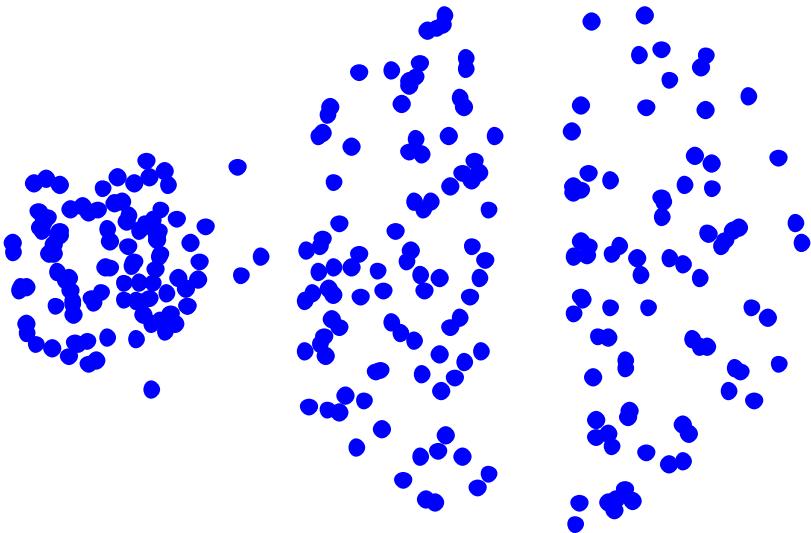


Original Data

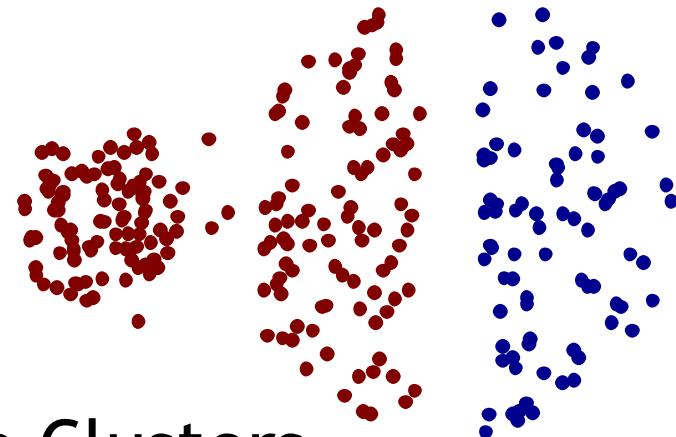
Six Clusters

- Can handle non-elliptical shapes

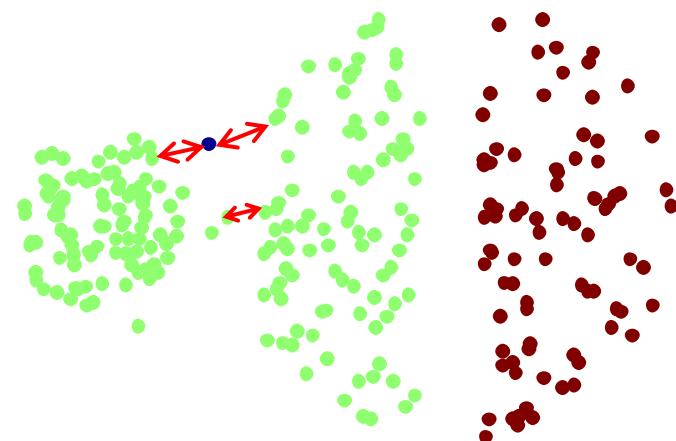
Limitations of MIN Distance Clustering



Original Data



Two Clusters

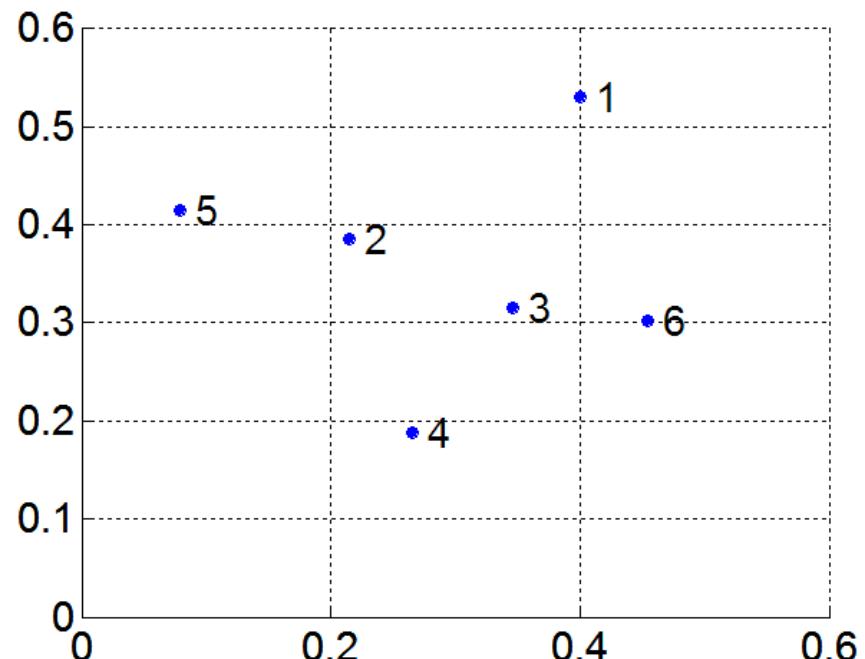


Three Clusters

- Sensitive to noise and outliers

MAX Distance (Complete Link) Clustering

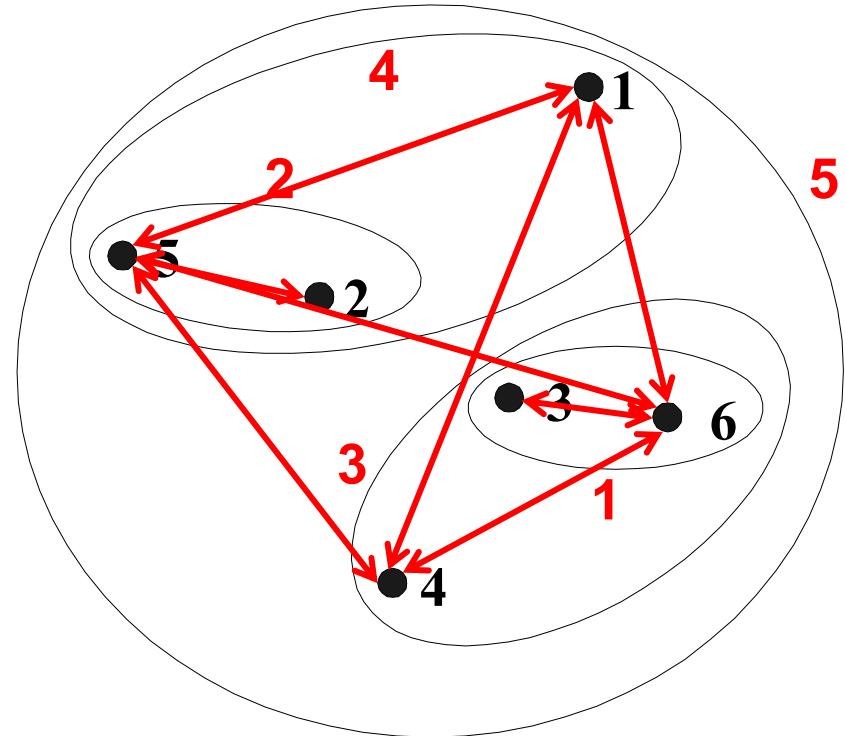
- Proximity of two clusters is based on the two most distant points in different clusters
 - Two clusters are merged if the distance between their “farthest nodes” is the “shortest” among all clusters



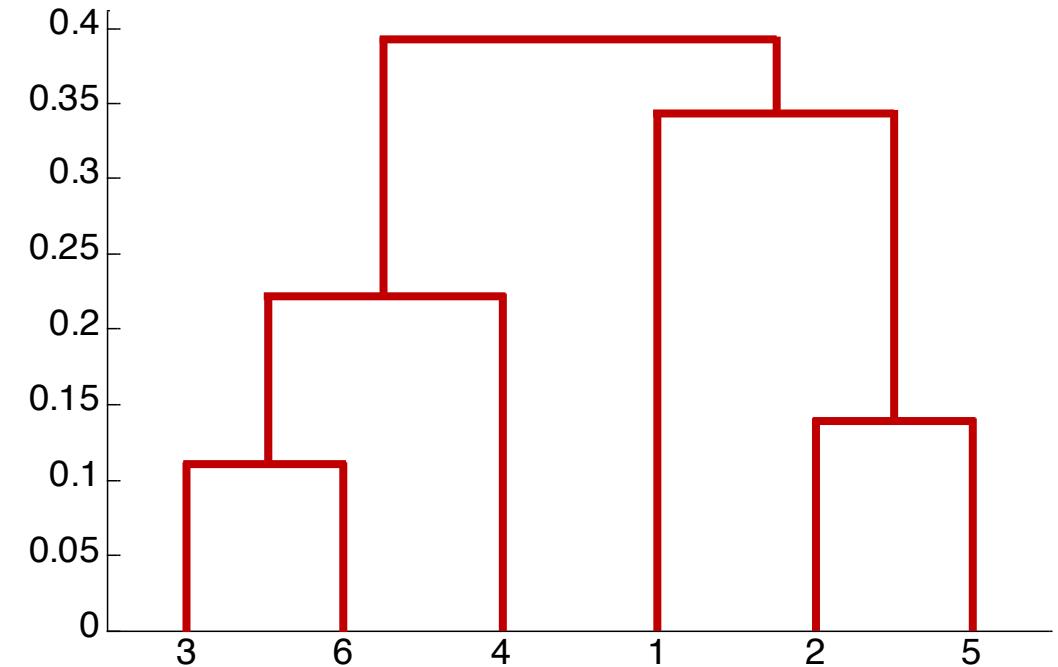
Distance Matrix

	p1	p2	p3	p4	p5	p6
p1	0.00	0.24	0.22	0.37	0.34	0.23
p2	0.24	0.00	0.15	0.20	0.14	0.25
p3	0.22	0.15	0.00	0.15	0.28	0.11
p4	0.37	0.20	0.15	0.00	0.29	0.22
p5	0.34	0.14	0.28	0.29	0.00	0.39
p6	0.23	0.25	0.11	0.22	0.39	0.00

MAX Distance (Complete Link) Clustering

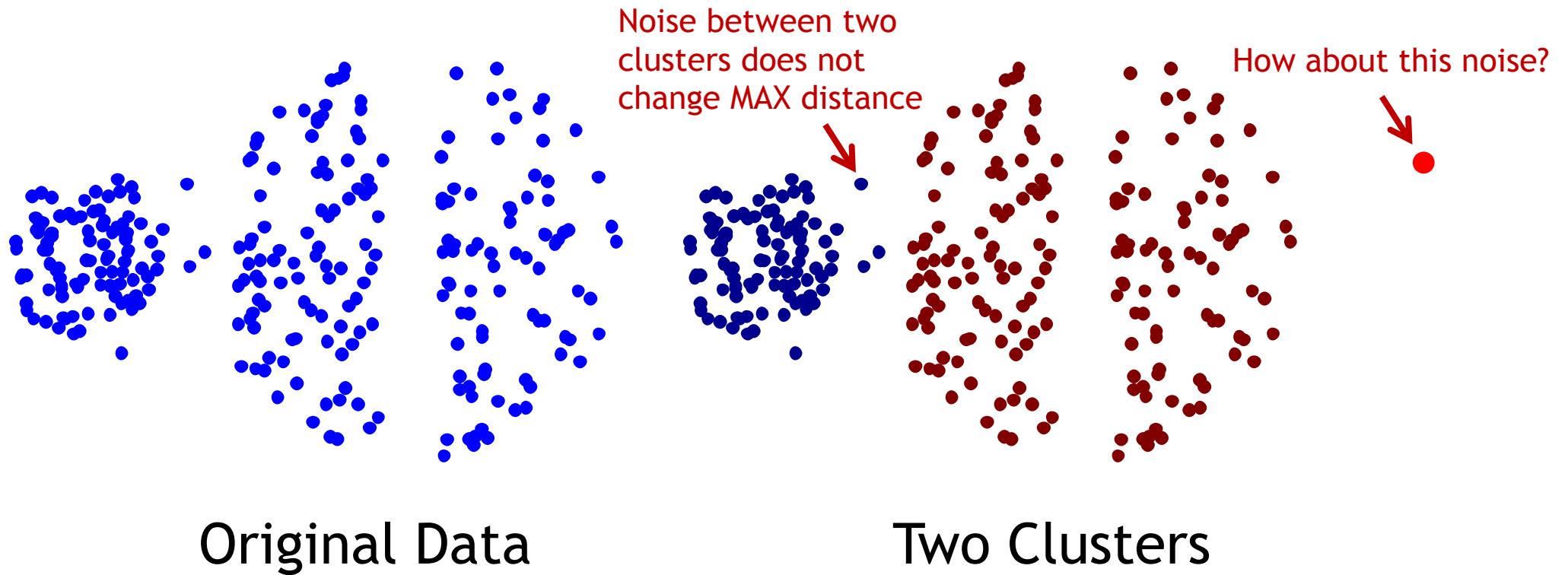


Nested Clusters



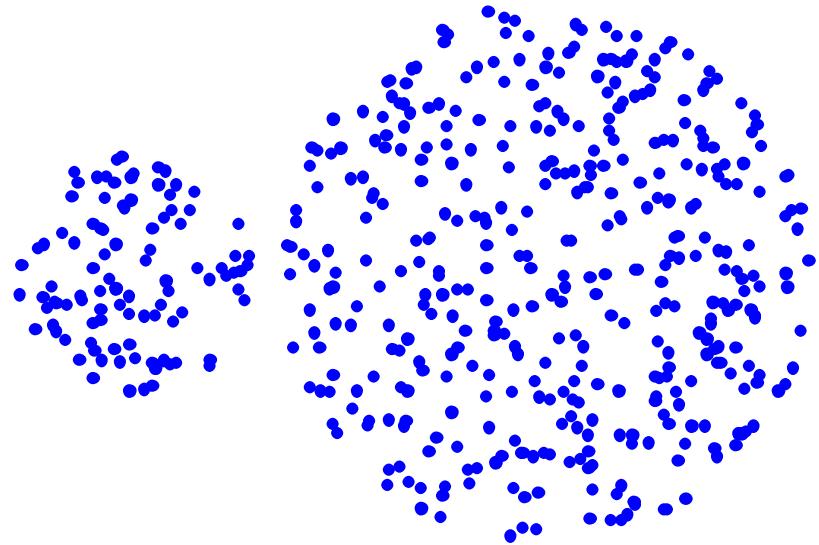
Dendrogram

Advantage of MAX Distance Clustering

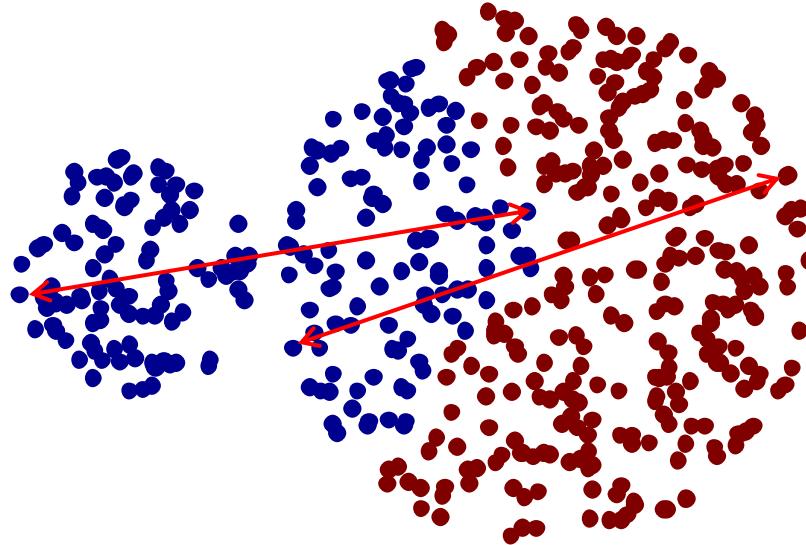


- Less sensitive to noise and outliers

Limitation of MAX Distance Clustering



Original Data



Two Clusters

- Tends to break large clusters

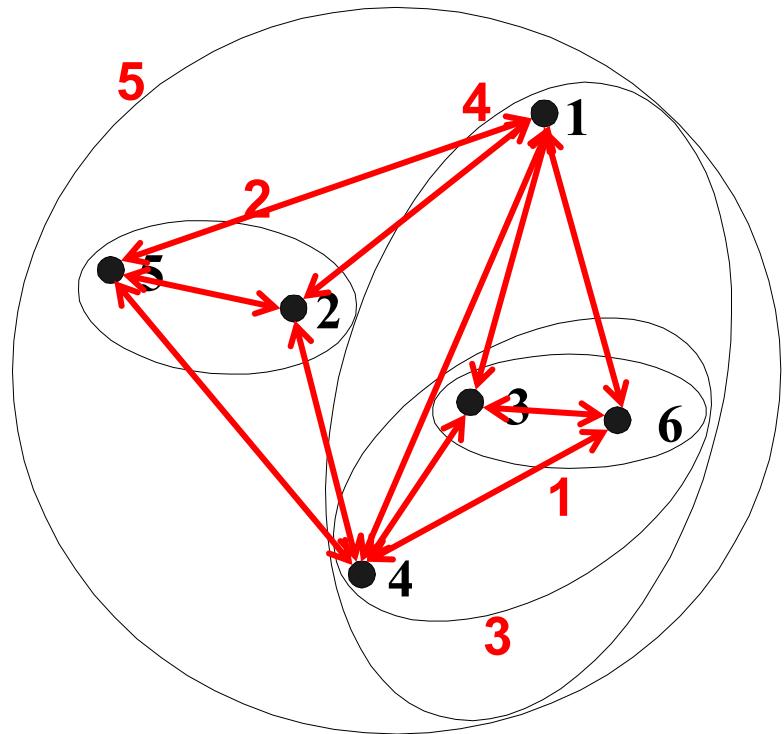
Group Average Clustering

- Proximity of two clusters is the average of pairwise proximity between points in the two clusters

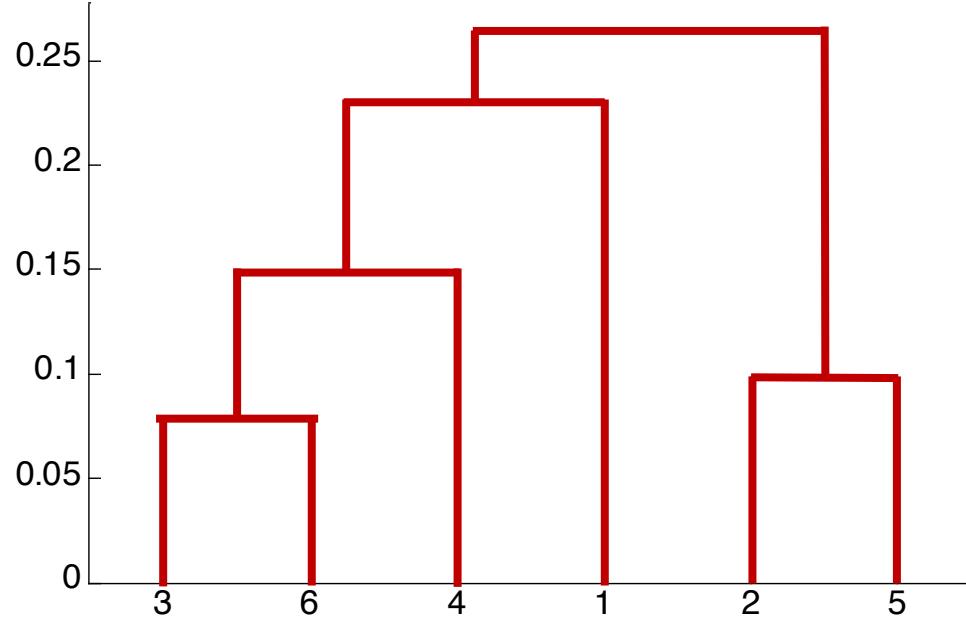
$$\text{proximity}(\text{Cluster}_i, \text{Cluster}_j) = \frac{\sum_{\substack{p_i \in \text{Cluster}_i \\ p_j \in \text{Cluster}_j}} \text{proximity}(p_i, p_j)}{|\text{Cluster}_i| \times |\text{Cluster}_j|}$$

- Need to use average connectivity for scalability since total proximity favors large clusters

Group Average Clustering



Nested Clusters



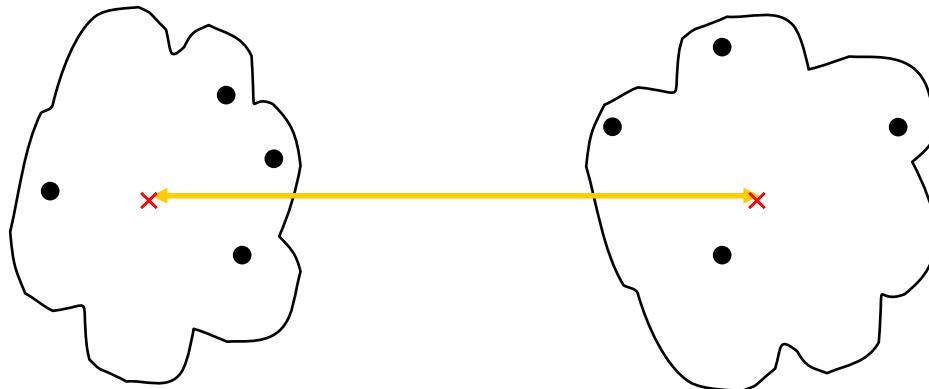
Dendrogram

Group Average Clustering

- Compromise between Single and Complete Link
- Strengths
 - Less susceptible to noise and outliers
 - Noises' impact is mitigated by average
- Limitations
 - Biased towards convex clusters

Centroid Distance Clustering

- Inter-Cluster Distance
 - The distance between clusters is represented by the distance between the centers of the clusters
 - Determined by cluster centroids



$$d_{mean}(C_i, C_j) = d(m_i, m_j)$$

Cluster Distance: Ward's Method

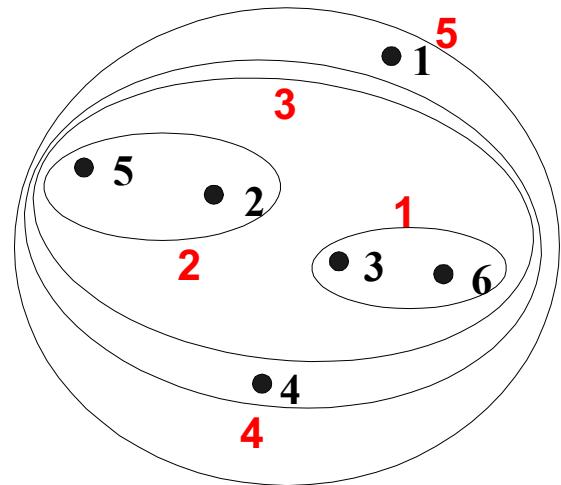
- Ward's method says that the distance between two clusters A and B is how much the sum of squares will **increase** when we merge them

$$\Delta(A, B) = \sum_{i \in A \cup B} \|\vec{x}_i - \vec{m}_{A \cup B}\|^2 - \sum_{i \in A} \|\vec{x}_i - \vec{m}_A\|^2 - \sum_{i \in B} \|\vec{x}_i - \vec{m}_B\|^2$$

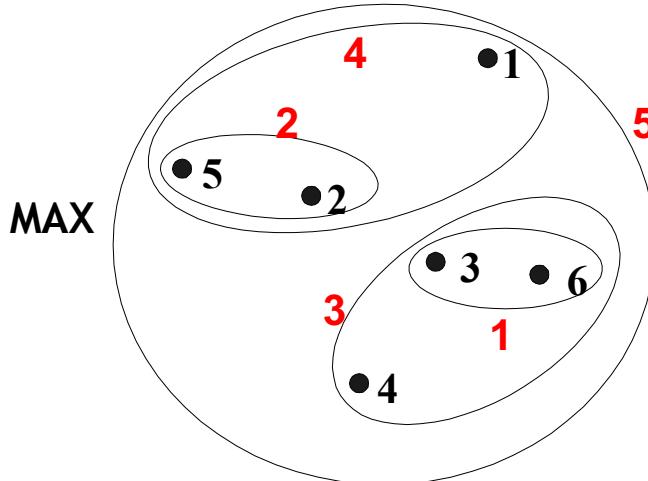
\vec{m}_j is the center of cluster j, n_j is the number of points in it, Δ is called the merging cost of combining A and B

- Less susceptible to noise and outliers
- Biased towards convex clusters
- Hierarchical analogue of K-means
 - Can be used to initialize K-means

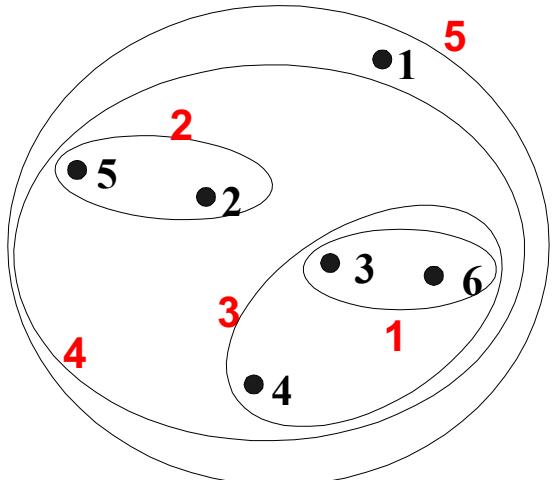
Hierarchical Clustering: Comparison



MIN

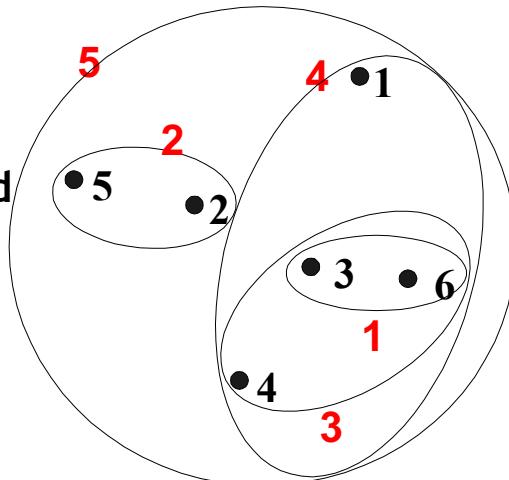


MAX



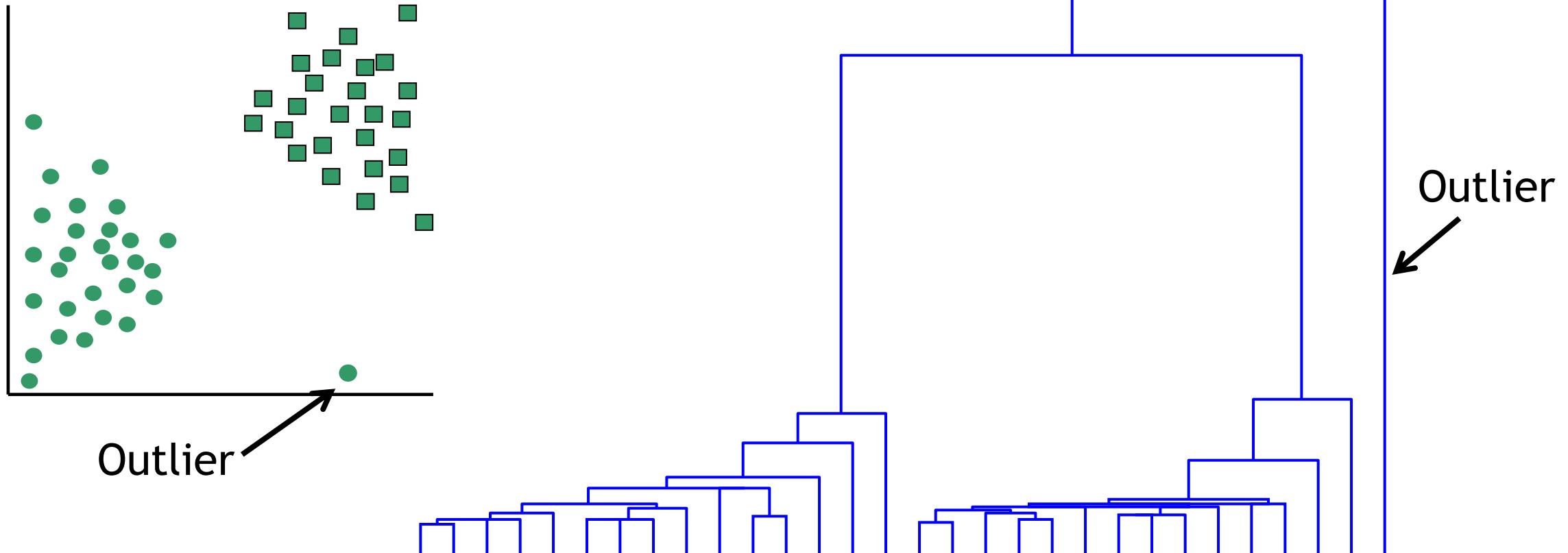
Group Average

Ward's Method



Detecting Outliers Using Dendrogram

- The single isolated branch is suggestive of a data point that is very different to all others

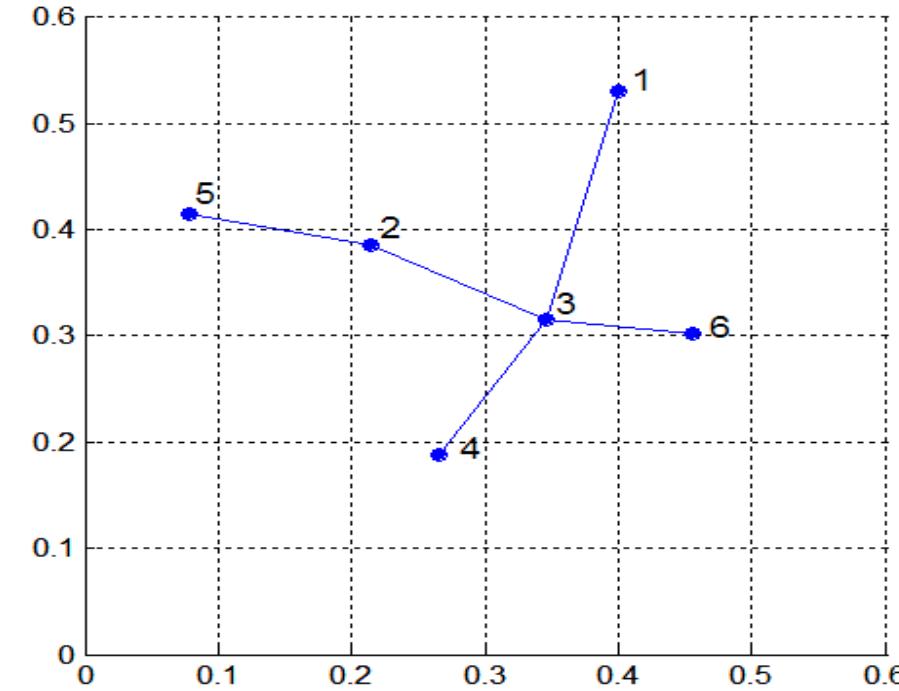
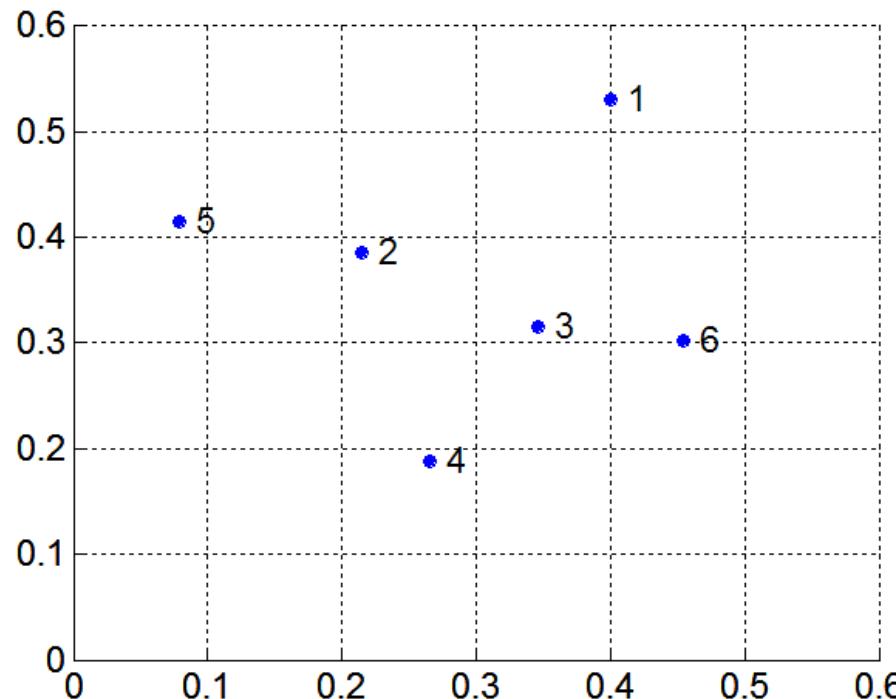


Time and Space Requirement

- $O(N^2)$ space since it uses the distance matrix
 - N is the number of data points
- $O(N^3)$ time in many cases
 - There are N steps and at each step the size, N^2 , distance matrix must be updated and searched
 - Complexity can be reduced to $O(N^2\log(N))$ for some approaches

Divisive Hierarchical Clustering

- Build MST (Minimum Spanning Tree)
 - Start with a tree that consists of any point
 - Create subclusters by breaking links in the minimum spanning tree



Divisive Hierarchical Clustering

- Use MST for constructing hierarchy of clusters

Algorithm 7.5 MST Divisive Hierarchical Clustering Algorithm

- 1: Compute a minimum spanning tree for the proximity graph.
 - 2: **repeat**
 - 3: Create a new cluster by breaking the link corresponding to the largest distance
 (smallest similarity).
 - 4: **until** Only singleton clusters remain
-

Remark: Selects Splits which maximize inter-cluster distance with respect to C1 and C2 (used to split C)

Strengths

- Do not have to assume any particular number of clusters
 - Any desired number of clusters can be obtained by ‘cutting’ the dendrogram at the proper level
- They may correspond to meaningful taxonomies
 - e.g., shopping websites—electronics (computer, camera, ...), furniture, groceries

Problems and Limitations

- Once a decision is made to combine two clusters, it cannot be undone → greedy algorithm
- No global objective function is directly minimized
- Different schemes have problems with one or more of the following:
 - Sensitivity to noise and outliers
 - Difficulty handling clusters of different sizes and non-globular shapes
 - Breaking large clusters
- In comparison to other methods: does not search a lot of clusters and therefore sometimes misses promising clusters

Agglomerative Clustering in Sklearn

```
class sklearn.cluster.AgglomerativeClustering(n_clusters=2,  
*, metric='euclidean', memory=None, connectivity=None, com  
pute_full_tree='auto', linkage='ward', distance_threshold=N  
one, compute_distances=False)
```

Recursively merges pair of clusters of sample data; uses linkage distance.

n_clusters: int or None, default=2

metric: str or callable, default="euclidean"

Metric used to compute the linkage. Can be "euclidean", "l1", "l2", "manhattan", "cosine", or "precomputed". If linkage is "ward", only "euclidean" is accepted.

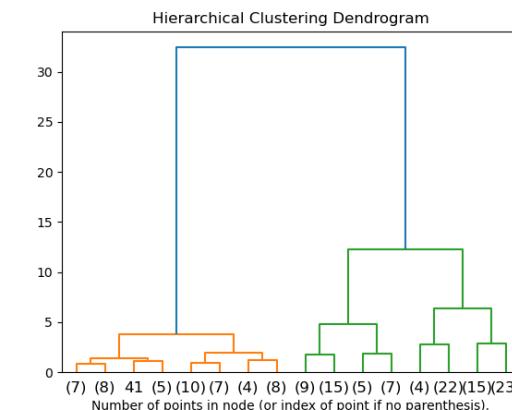
compute_full_tree: 'auto' or bool, default='auto'

Stop early the construction of the tree at n_clusters. This is useful to decrease computation time if the number of clusters is not small compared to the number of samples.

linkage: {'ward', 'complete', 'average', 'single'}, default='ward'

Which linkage criterion to use. The linkage criterion determines which distance to use between sets of observation. The algorithm will merge the pairs of cluster that minimize this criterion.

```
>>> from sklearn.cluster import AgglomerativeClustering  
>>> import numpy as np  
>>> X = np.array([[1, 2], [1, 4], [1, 0], ... [4, 2],  
[4, 4], [4, 0]])  
>>> clustering = AgglomerativeClustering().fit(X)  
>>> clustering = AgglomerativeClustering()  
>>> clustering.labels_ array([1, 1, 1, 0, 0, 0])  
https://scikit-learn.org/stable/auto\_examples/cluster/plot\_agglomerative\_dendrogram.html#sphx-glr-auto-examples-cluster-plot-agglomerative-dendrogram-py  
>>> plt.title("Hierarchical Clustering Dendrogram")  
>>> plot_dendrogram(model, truncate_mode="level", p=3)  
>>> plt.xlabel("Number of points in node (or index of point if no parenthesis).")  
>>> plt.show()
```



Summary

- Agglomerative and divisive hierarchical clustering
- Several ways of defining inter-cluster distance
- The properties of clusters outputted by different approaches based on different inter-cluster distance definition
- Pros and cons of hierarchical clustering



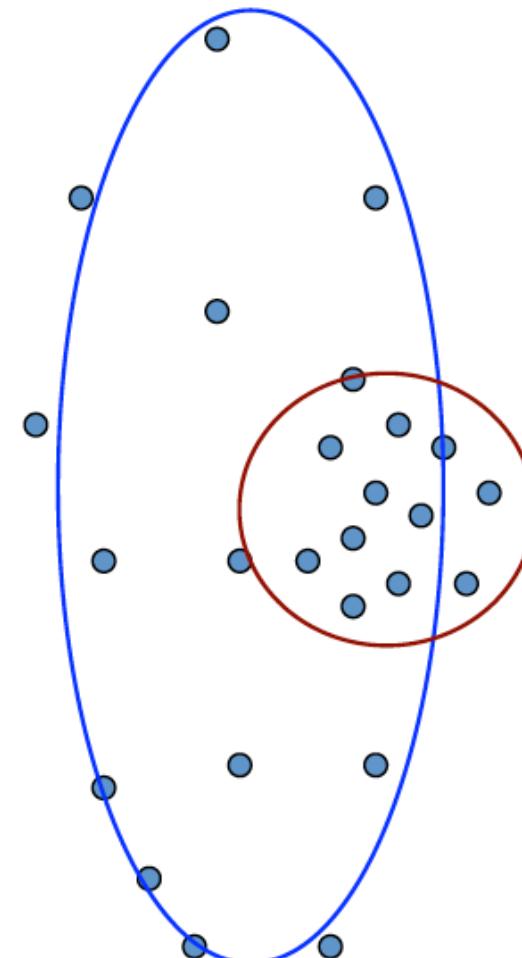
COSC 3337
Data Science I
Section 14623

Clustering (contd.)

Instructor: Jingchao Ni
Fall 2024

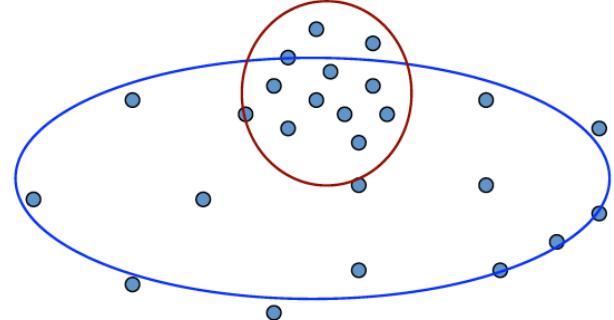
Problems of “Hard Assignments”

- Clusters may partially overlap
 - Different from hierarchical clustering
- Some clusters may be “wider” than others
- Distances can be deceiving



Probabilistic Clustering

- Try a probabilistic model
 - Allow overlaps, clusters of different sizes, densities, etc.
- Can tell a generative story for data
 - $P(X,Y) = P(X|Y)p(Y)$
- Challenge: we need to estimate model parameters without labels “Y”’s

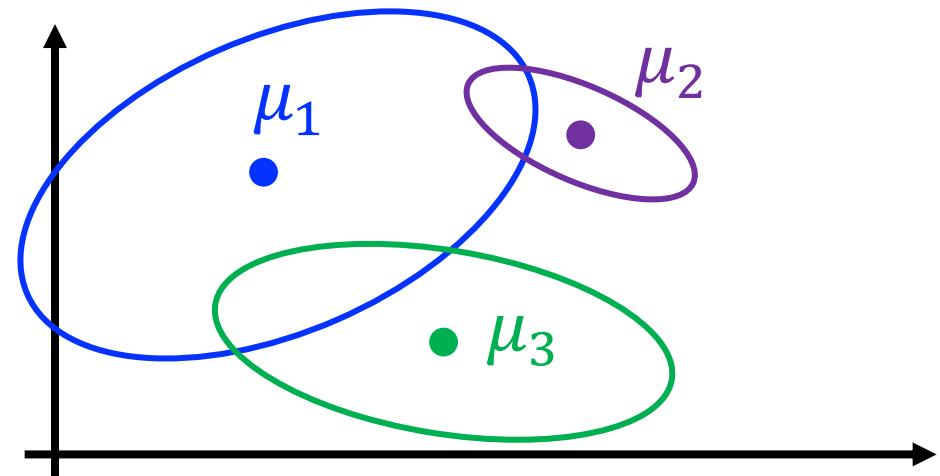


Y	X ₁	X ₂
??	0.1	2.1
??	0.5	-1.1
??	0.0	3.0
??	-0.1	-2.0
??	0.2	1.5
...

Gaussian Mixture Model: Assumption

- $P(Y)$: There are K components (clusters)
- $P(X|Y)$: Each component generates data from a **multivariate Gaussian distribution** with mean μ_i and covariance matrix Σ_i
 - Range of i : $[1, K]$
- **Each data point is sampled from a generative process:**
 1. Choose a component i with probability $P(y=i)$
 2. Generate data point from $N(\mu_i, \Sigma_i)$

Gaussian Mixture Model
(GMM) with 3 components



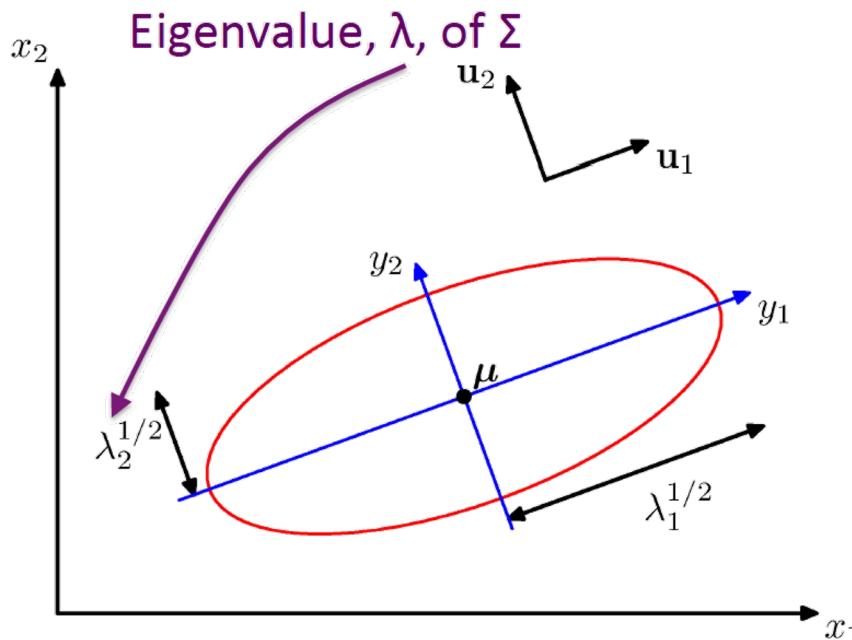
Multivariate Gaussian Distribution

- Univariate Gaussian distribution

$$P(X_i | Y_j) = \frac{1}{\sqrt{2\pi\sigma_{ij}^2}} e^{-\frac{(X_i - \mu_{ij})^2}{2\sigma_{ij}^2}}$$

- Multivariate Gaussian distribution

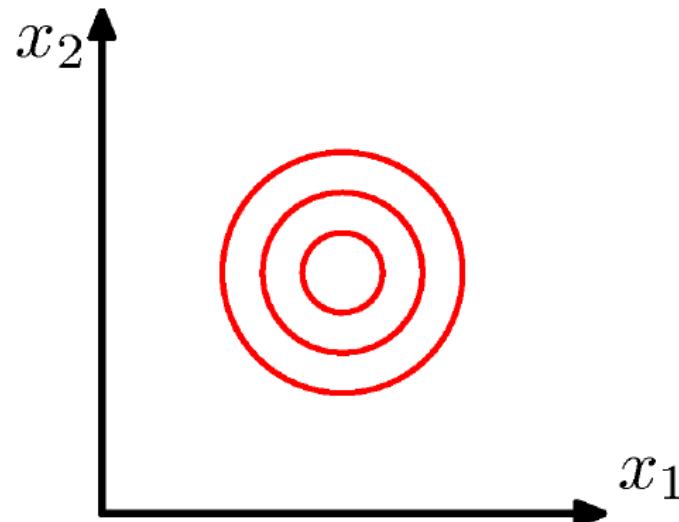
$$P(\mathbf{X} = \mathbf{x}_j | Y = i) = \frac{1}{(2\pi)^{d/2} \|\Sigma_i\|^{1/2}} \exp \left[-\frac{1}{2} (\mathbf{x}_j - \boldsymbol{\mu}_i)^T \boldsymbol{\Sigma}_i^{-1} (\mathbf{x}_j - \boldsymbol{\mu}_i) \right]$$



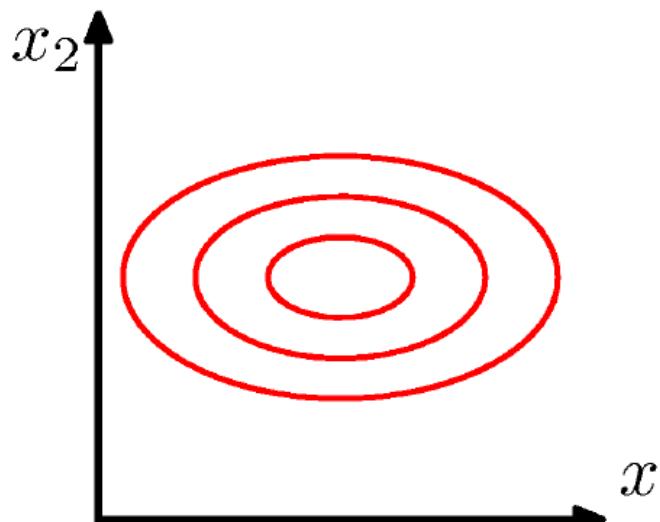
Covariance matrix, Σ ,
represents the degree to
which x_1, \dots, x_d vary together

Multivariate Gaussian Distribution

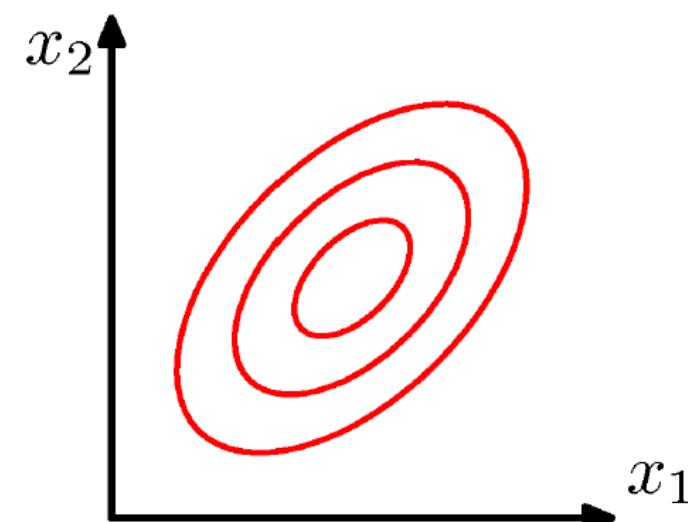
$$P(X = \mathbf{x}_j | Y = i) = \frac{1}{(2\pi)^{d/2} \|\Sigma_i\|^{1/2}} \exp \left[-\frac{1}{2} (\mathbf{x}_j - \boldsymbol{\mu}_i)^T \Sigma_i^{-1} (\mathbf{x}_j - \boldsymbol{\mu}_i) \right]$$



Σ is identity matrix



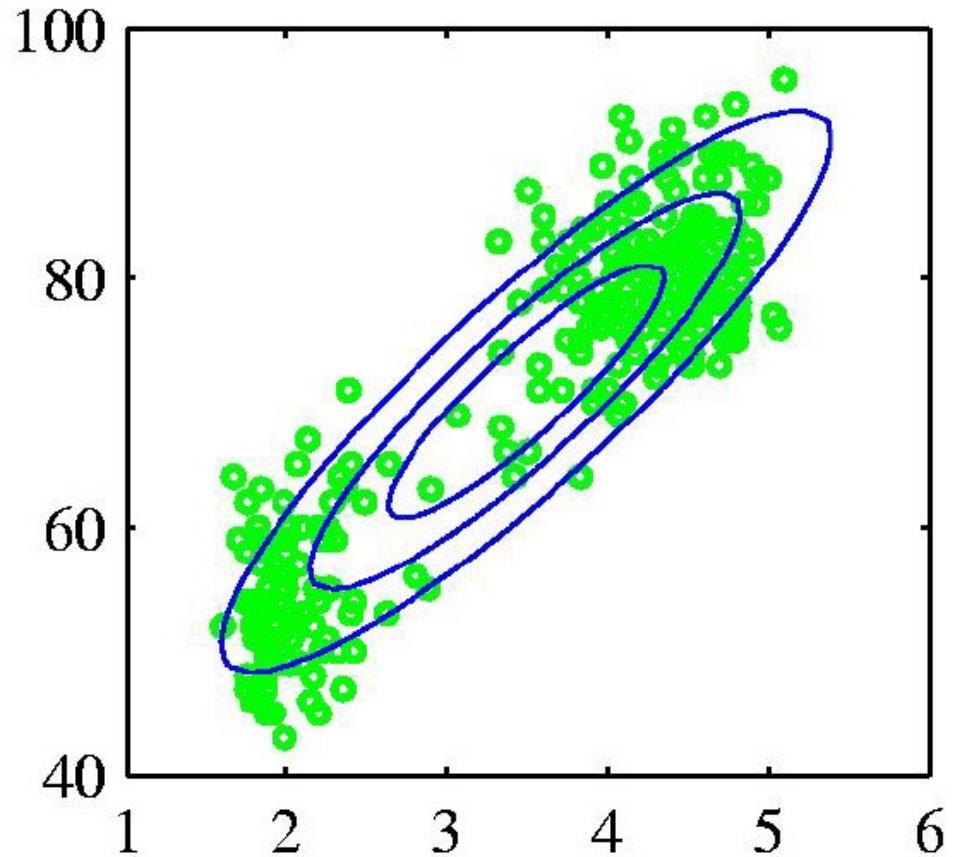
Σ is diagonal matrix
Xi are independent
from each other



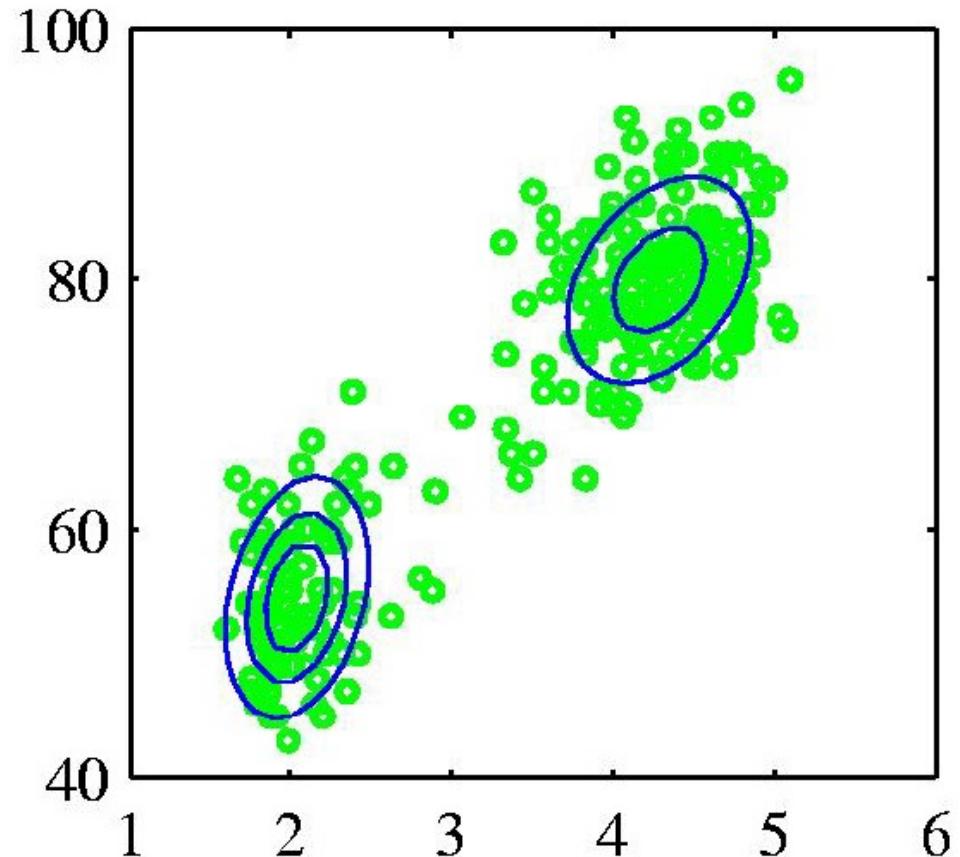
Σ is arbitrary (semidefinite) matrix

- Specifies rotation (change of basis)
- Eigenvalues specific relative elongation

Fitting GMM with Data: Example



Single Gaussian Component



Mixture of Two Gaussian Components

Density Function

- Combine simple models into a complex model

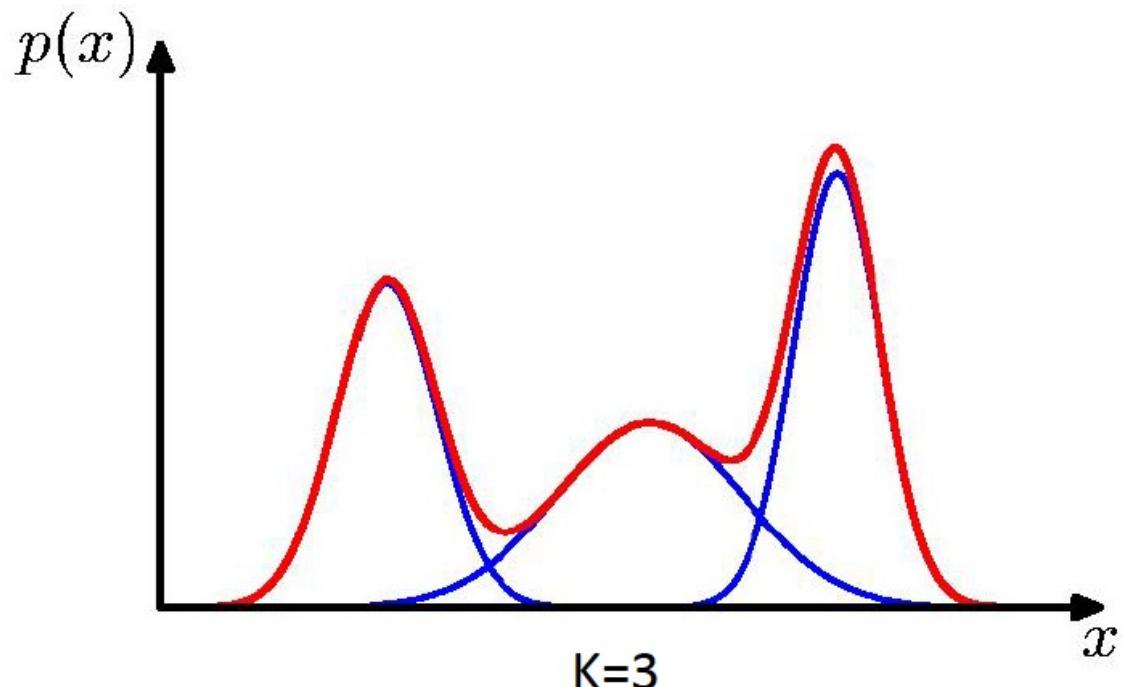
$$P(X) = \sum_{i=1}^K P(X, Y = i) = \sum_{i=1}^K P(X|Y = i)P(Y = i)$$

$$\Rightarrow P(\mathbf{x}) = \sum_{i=1}^K \pi_i \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)$$

$$\forall i: \pi_i \geq 0$$

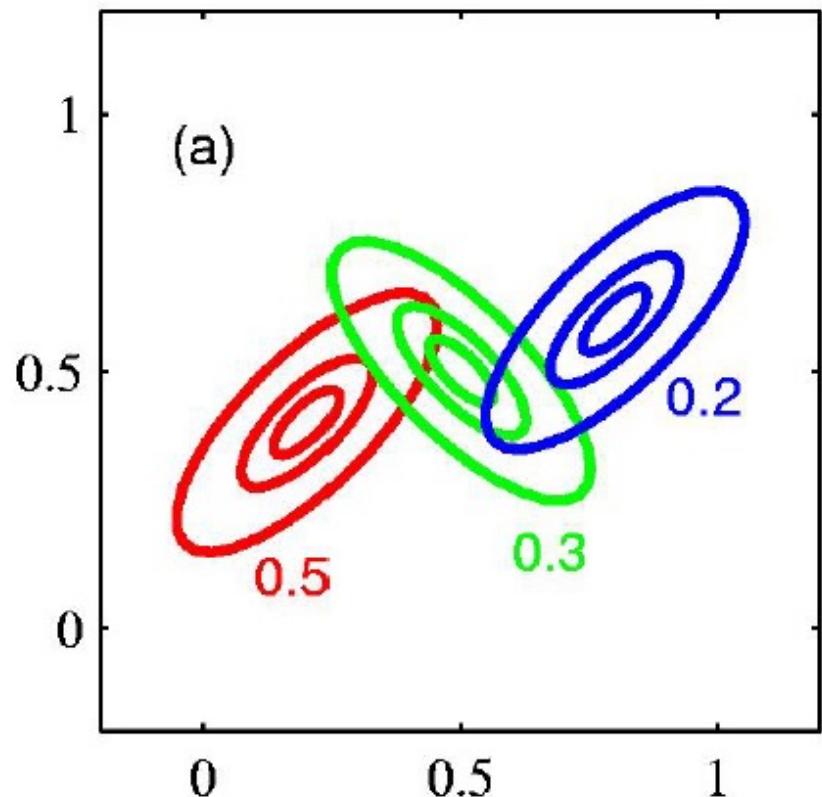
$$\sum_{i=1}^K \pi_i = 1$$

The i -th Gaussian component
The i -th mixture coefficient

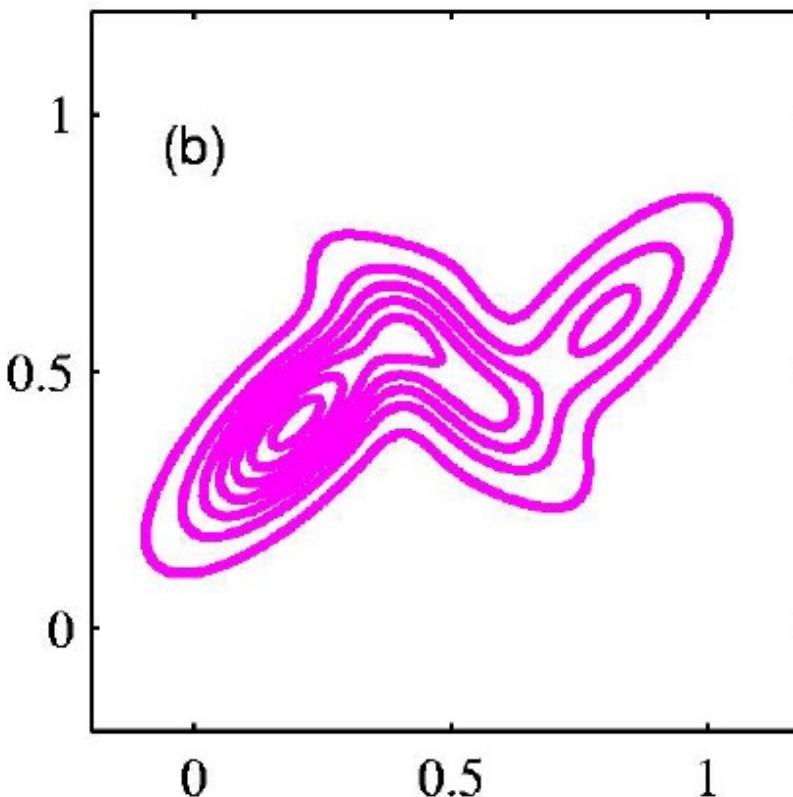


K=3

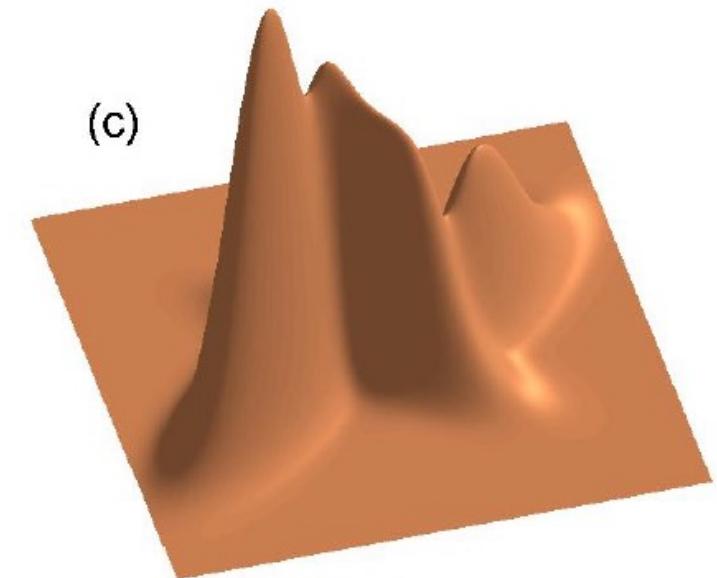
Density Function



Three Gaussian components



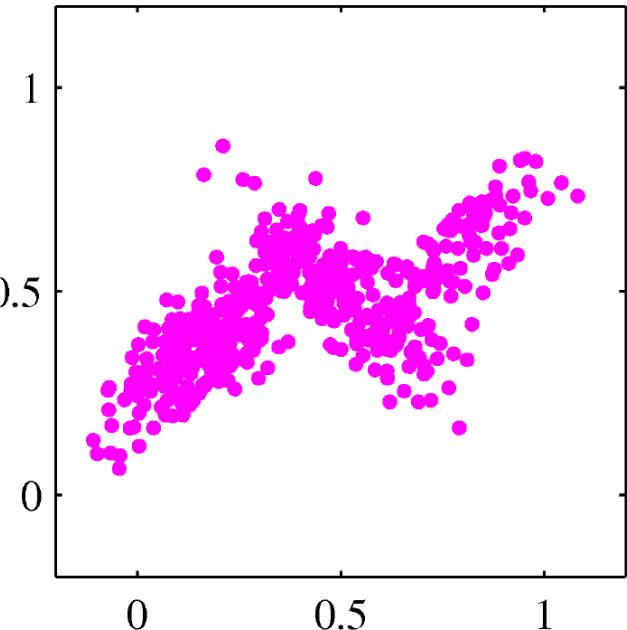
Mixture distribution of three Gaussian components



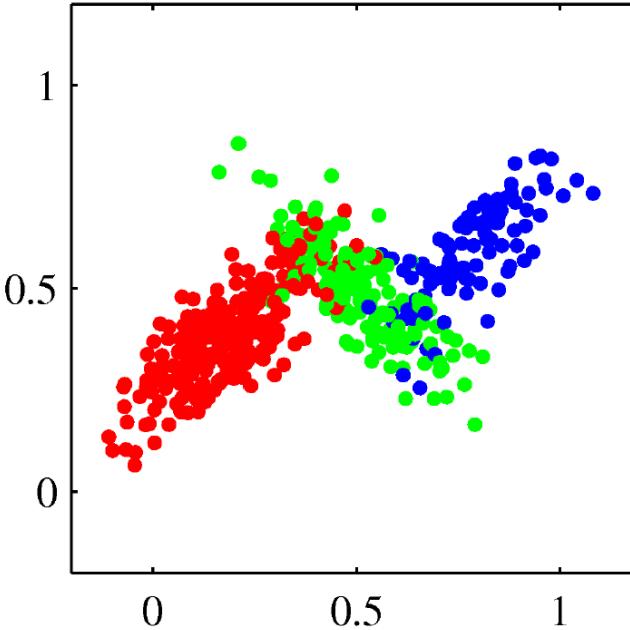
3D view

Soft Assignments to Clusters

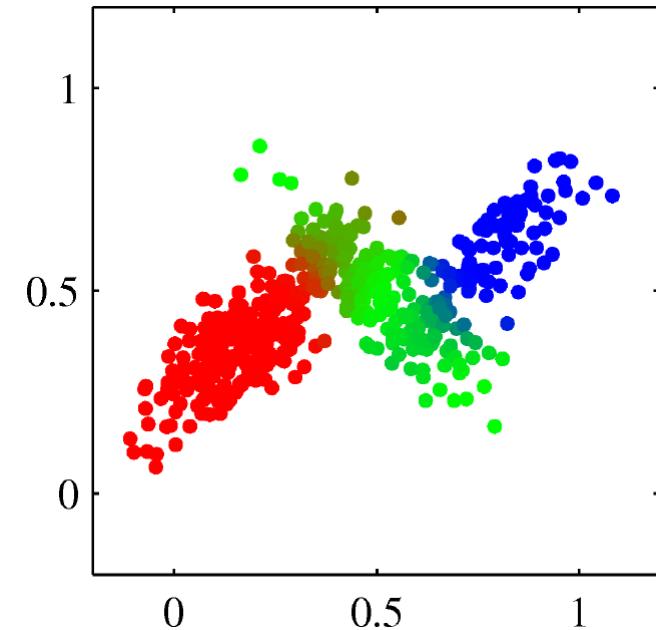
- Model data as mixture of multivariate Gaussian distributions



Original data



Hard cluster assignments



Soft cluster assignments

- Shown is the posterior probability that a point was generated from the i -th component: $P(Y=i|x)$

MLE in Supervised Setting

- Maximum Likelihood Estimation Solution
 - Univariate Gaussian distribution

$$\mu_{MLE} = \frac{1}{N} \sum_{i=1}^N x_i \quad \sigma_{MLE}^2 = \frac{1}{N} \sum_{i=1}^N (x_i - \hat{\mu})^2$$

- Mixture of Multivariate Gaussian distribution

$$\mu_{ML}^k = \frac{1}{n} \sum_{j=1}^n x_n \quad \Sigma_{ML}^k = \frac{1}{n} \sum_{j=1}^n (\mathbf{x}_j - \mu_{ML}^k)(\mathbf{x}_j - \mu_{ML}^k)^T$$

Sum over x generated from the k -th Gaussian component

How about Unsupervised Setting?

- Maximum Likelihood Estimation

$$\operatorname{argmax}_{\theta} \prod_j P(y_j, x_j)$$

- θ : all model parameters
 - E.g., class probs, means, variances
- But we don't know y_j 's!
- Maximize **marginal likelihood**

$$\operatorname{argmax}_{\theta} \prod_j P(x_j) = \operatorname{argmax} \prod_j \sum_{k=1}^K P(Y_j=k, x_j)$$

How about Unsupervised Setting?

- Maximize marginal likelihood

$$\operatorname{argmax}_{\theta} \prod_j P(x_j) = \operatorname{argmax} \prod_j \sum_{k=1}^K P(Y_j=k, x_j)$$

- Almost always a hard problem!
 - Usually no closed form solution
 - Even when $\log P(X, Y)$ is convex, $\log P(X)$ generally isn't ...
 - For all but the simplest $P(X)$, we will have to do gradient ascent/descent, in a big messy space with lots of local optimum

Learning General Mixtures of Gaussian

- Cluster membership probability (posterior)

$$P(y = i | \mathbf{x}_j) \propto \frac{1}{(2\pi)^{\frac{d}{2}} \|\Sigma_i\|^{\frac{1}{2}}} \exp \left[-\frac{1}{2} (\mathbf{x}_j - \boldsymbol{\mu}_i)^T \Sigma_i^{-1} (\mathbf{x}_j - \boldsymbol{\mu}_i) \right] P(y = i)$$

- Marginal likelihood:

$$\begin{aligned} \prod_{j=1}^n P(\mathbf{x}_j) &= \prod_{j=1}^n \sum_{i=1}^K P(\mathbf{x}_j, y = i) \\ &= \prod_{j=1}^n \sum_{i=1}^K \frac{1}{(2\pi)^{\frac{d}{2}} \|\Sigma_i\|^{\frac{1}{2}}} \exp \left[-\frac{1}{2} (\mathbf{x}_j - \boldsymbol{\mu}_i)^T \Sigma_i^{-1} (\mathbf{x}_j - \boldsymbol{\mu}_i) \right] P(y = i) \end{aligned}$$

- Need to differentiate and solve for $\boldsymbol{\mu}_i$, Σ_i , and $P(Y=i)$ for $i=1, \dots, K$
- There will be no closed form solution, gradient is complex, lots of local optimum

The Expectation-Maximization Algorithm

- A clever method for maximizing marginal likelihood

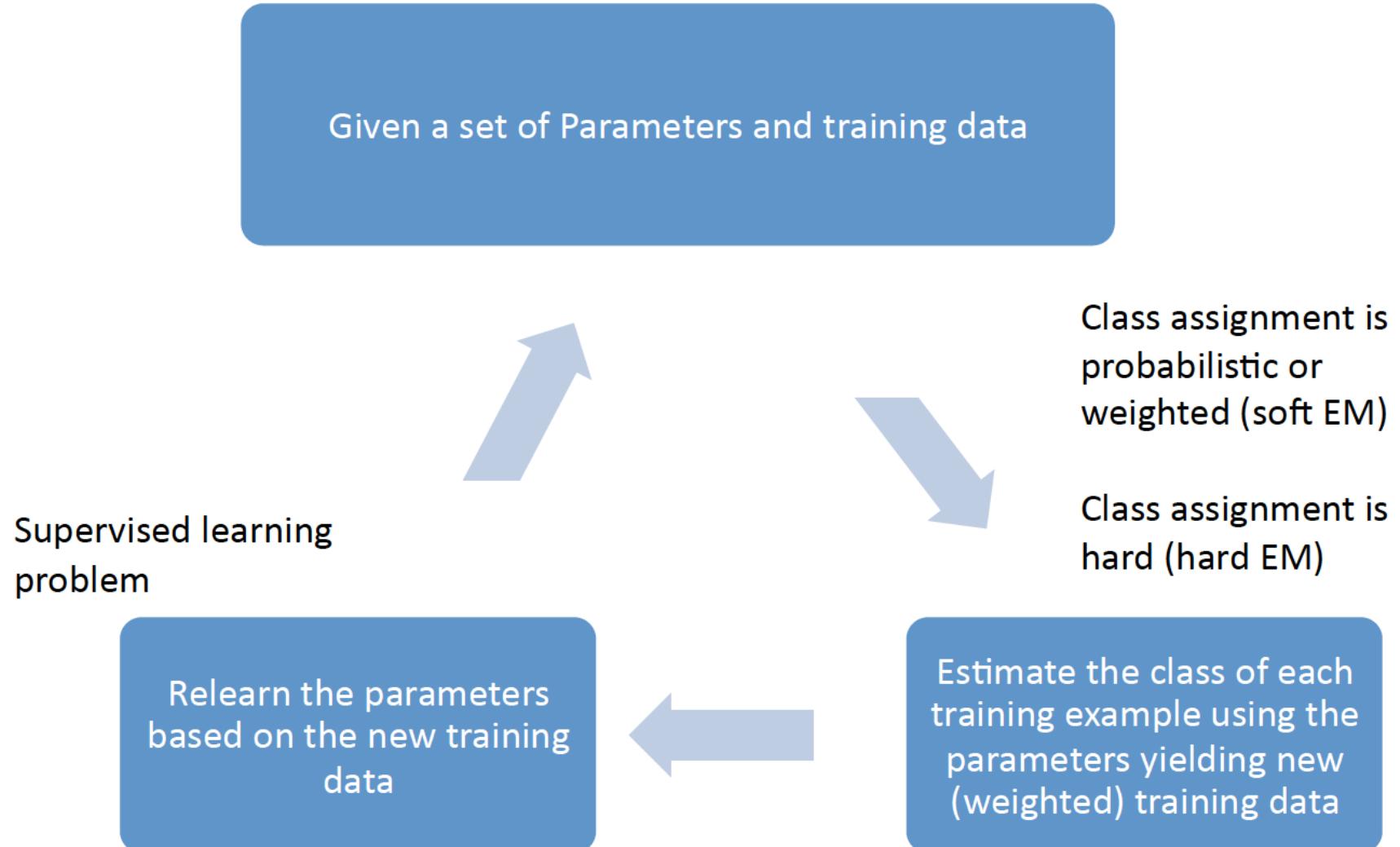
$$\operatorname{argmax}_{\theta} \prod_j P(x_j) = \operatorname{argmax} \prod_j \sum_{k=1}^K P(Y_j=k, x_j)$$

- A type of gradient ascent that can be easy to implement
 - E.g., no line search, learning rates, etc.
- Alternate between two steps:
 - Compute an expectation
 - Compute a maximization
- Not magic: still optimizing a non-convex function with lots of local optima
 - The computations are just easier

EM: Two Easy Steps

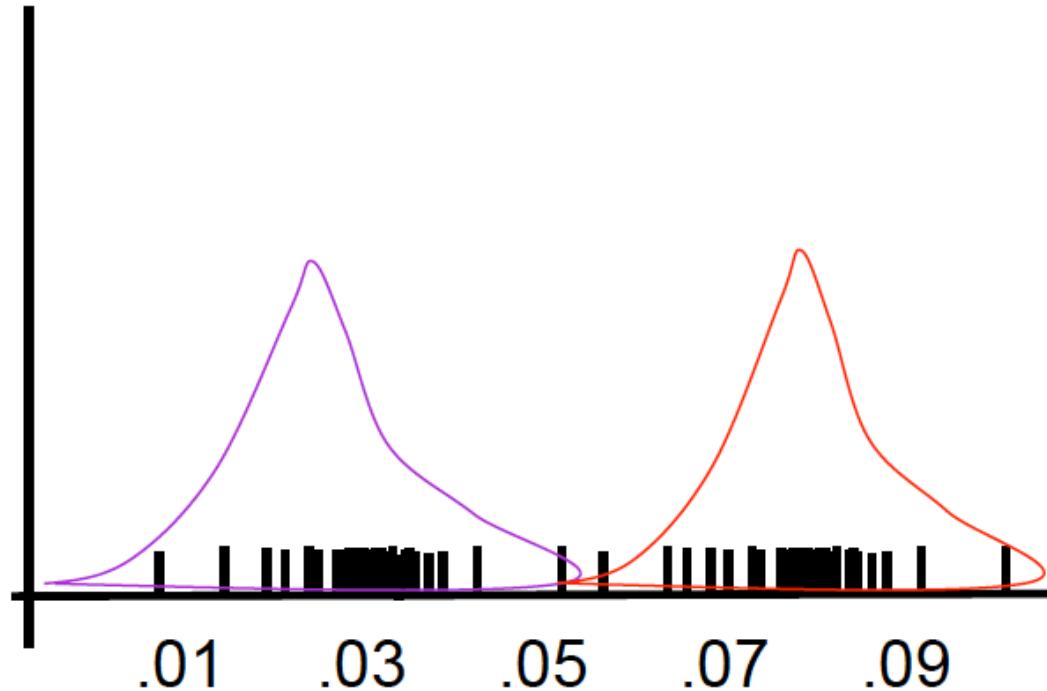
- Objective: $\operatorname{argmax}_{\theta} \lg \prod_j \sum_{k=1}^K P(Y_j=k, x_j | \theta) = \sum_j \lg \sum_{k=1}^K P(Y_j=k, x_j | \theta)$
- Data: $\{x_j \mid j=1, \dots, n\}$
- **E-Step:** Compute expectations to “fill in” missing y values according to the current parameters, θ
 - For all examples j and values k for Y_j , compute $P(Y_j=k \mid x_j, \theta)$
- **M-Step:** Re-estimate the parameters with “weighted” MLE estimates
 - Set $\theta = \operatorname{argmax}_{\theta} \sum_j \sum_k P(Y_j=k \mid x_j, \theta) \log P(Y_j=k, x_j | \theta)$

EM Algorithm: Overview



Simple Example

- Consider:
 - 1D data
 - Mixture of $k=2$ components
 - Variances fixed to $\sigma = 1$
 - Distribution over classes is uniform
 - Just need to estimate μ_1 and μ_2



$$\prod_{j=1}^n \sum_{k=1}^K P(x, Y_j = k) \propto \prod_{j=1}^n \sum_{k=1}^{K=2} \exp\left[-\frac{1}{2\sigma^2} \|x - \mu_k\|^2\right] P(Y_j = k)$$

EM for GMMs: Only Learning Means

- Iterate: on the t-th iteration, let our estimates be

$$\theta^{(t)} = \{\mu_1^{(t)}, \mu_2^{(t)}, \dots, \mu_K^{(t)}\}$$

- E-Step

- Compute “expected” classes of all data points

$$P(Y_j = k | x_j, \mu_1, \dots, \mu_K) \propto \exp\left(-\frac{1}{2\sigma^2} \|x_j - \mu_k\|^2\right) P(Y_j = k)$$

- M-Step

- Compute most likely new μ 's given class expectations

$$\mu_k = \frac{\sum_{j=1}^m P(Y_j = k | x_j) x_j}{\sum_{j=1}^m P(Y_j = k | x_j)}$$

EM for General GMMs

- Iterate: on the t-th iteration, let our estimates be

$$\theta^{(t)} = \{\mu_1^{(t)}, \dots, \mu_K^{(t)}, \Sigma_1^{(t)}, \dots, \Sigma_K^{(t)}, p_1^{(t)}, \dots, p_K^{(t)}\}$$

- E-Step

- Compute “expected” classes of all data points for each class

$$P(Y_j = k | x_j, \lambda_t) \propto p_k^{(t)} p(x_j | \mu_k^{(t)}, \Sigma_k^{(t)})$$

Just evaluate a Gaussian at x_j

- M-Step

- Compute weighted MLE for θ

$$\mu_k^{(t+1)} = \frac{\sum_j P(Y_j = k | x_j, \lambda_t) x_j}{\sum_j P(Y_j = k | x_j, \lambda_t)}$$

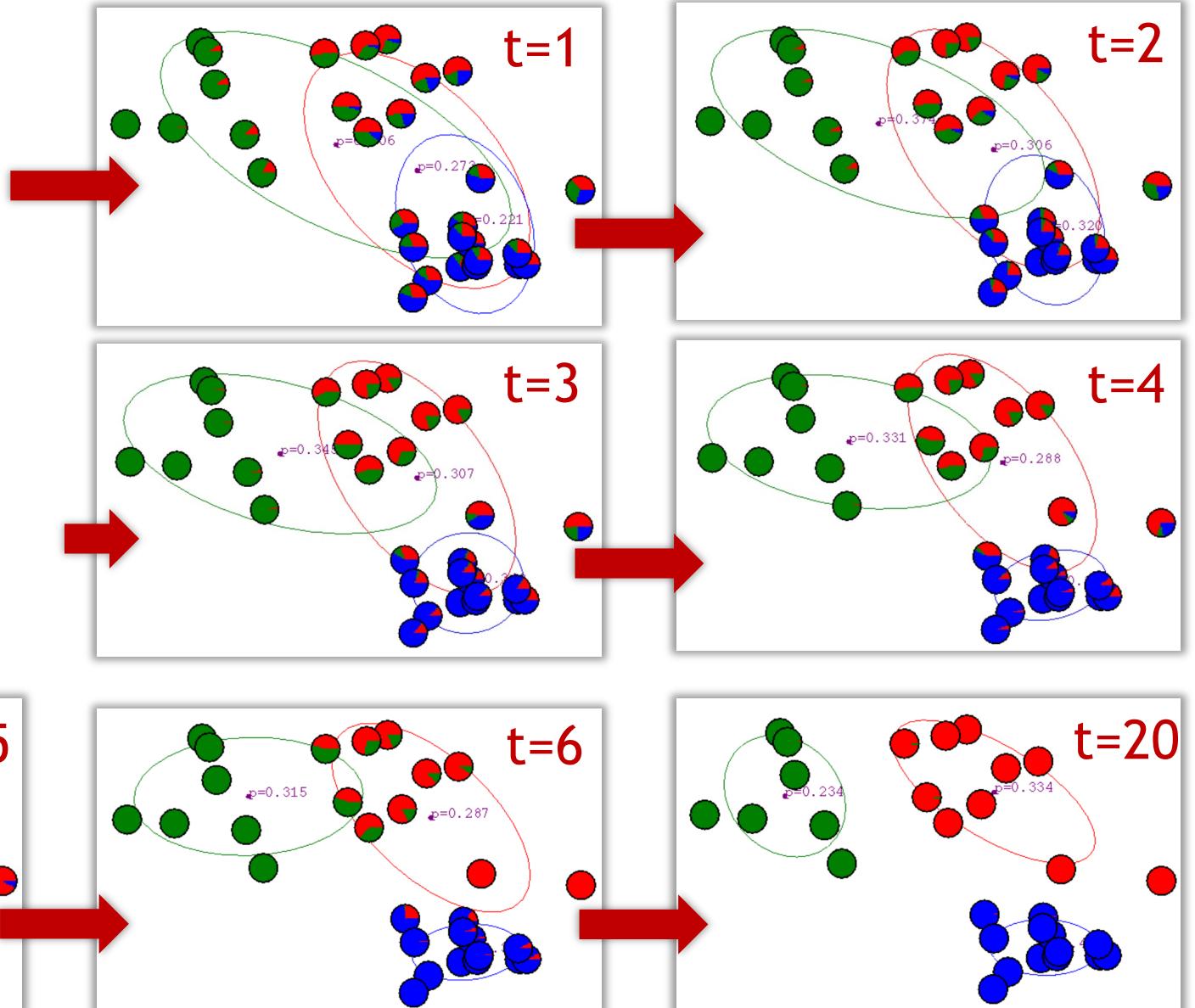
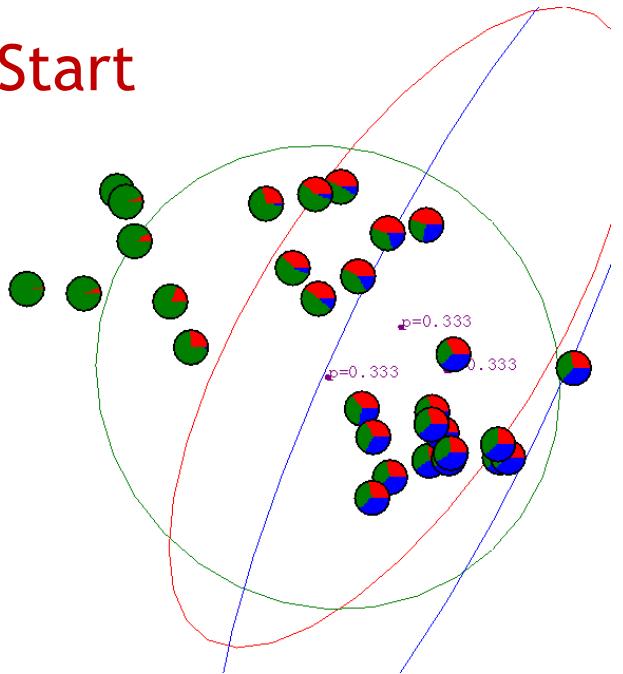
$$\Sigma_k^{(t+1)} = \frac{\sum_j P(Y_j = k | x_j, \lambda_t) [x_j - \mu_k^{(t+1)}] [x_j - \mu_k^{(t+1)}]^T}{\sum_j P(Y_j = k | x_j, \lambda_t)}$$

$$p_k^{(t+1)} = \frac{\sum_j P(Y_j = k | x_j, \lambda_t)}{n}$$

$p_k^{(t)}$ is shorthand for estimate of $P(y=k)$ on t'th iteration

GMM Example

Start



What If We Do Hard Assignments?

- Iterate: on the t-th iteration, let our estimates be

$$\theta^{(t)} = \{\mu_1^{(t)}, \mu_2^{(t)}, \dots, \mu_K^{(t)}\}$$

- E-Step

- Compute “expected” classes of all data points

$$P(Y_j = k | x_j, \mu_1, \dots, \mu_K) \propto \exp\left(-\frac{1}{2\sigma^2} \|x_j - \mu_k\|^2\right) P(Y_j = k)$$

- M-Step

- Compute most likely new μ 's given class expectations

$$\mu_k = \frac{\sum_{j=1}^m \delta(Y_j = k, x_j) x_j}{\sum_{j=1}^m \delta(Y_j = k, x_j)} = \frac{\delta(Y_j = k, x_j) x_j}{\sum_{j=1}^m \delta(Y_j = k, x_j)}$$

Equivalent to k-means clustering!

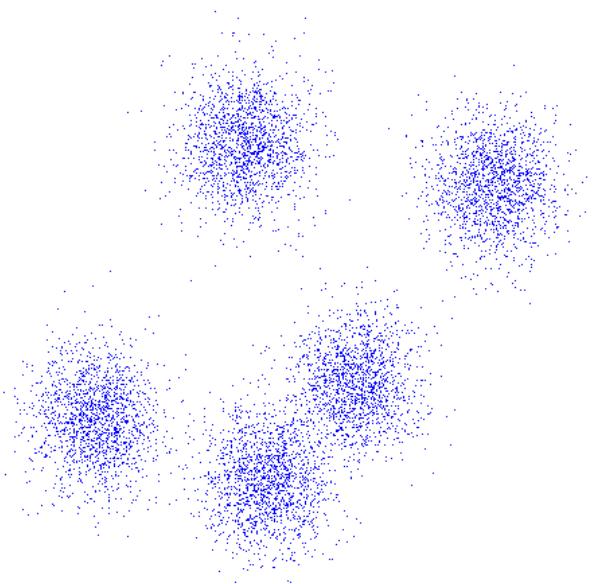
δ : hard assignment to “most likely” or nearest cluster

Summary of GMM

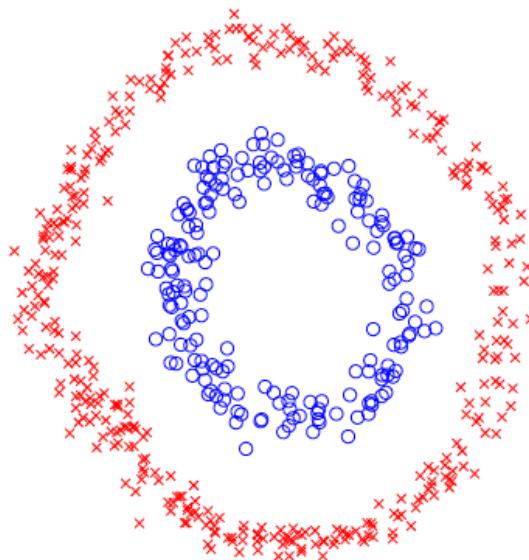
- Combination of simple models
- A probabilistic clustering model, formulated as a maximum likelihood estimation problem
- The optimization problem is solved by EM algorithm
 - E-step: compute the expectation of cluster assignment
 - M-step: maximize likelihood by solving model parameters
- Advantage: fits clusters with overlapped boundaries, different sizes, different densities
- Disadvantage: EM algorithm can get stuck in local minima
 - GMM is usually initialized by firstly running k-means

Spectral Clustering

- Two different clustering criteria
 - Compactness, e.g., k-means, mixture models
 - Connectivity, e.g., spectral clustering

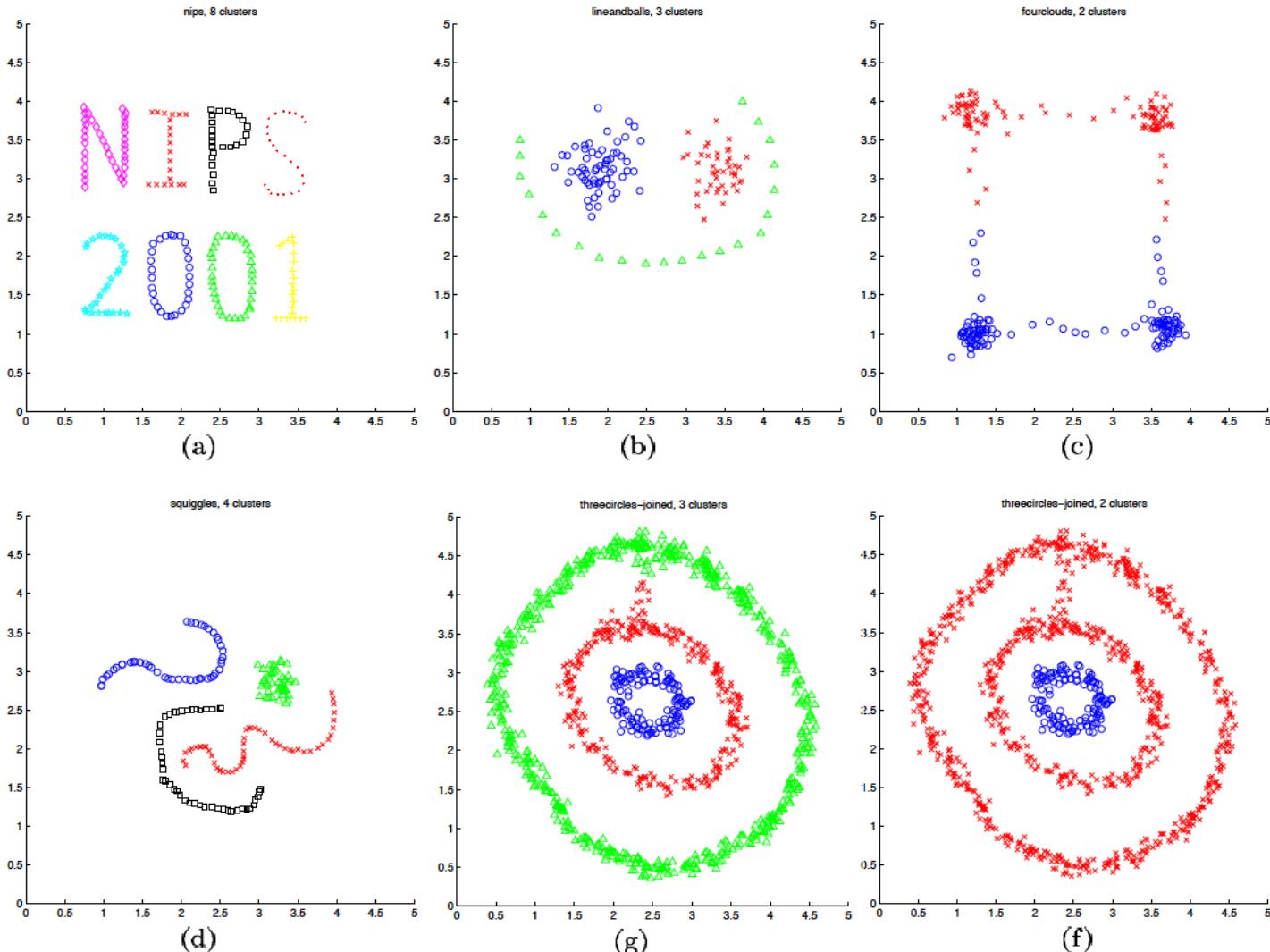


Compactness



Connectivity

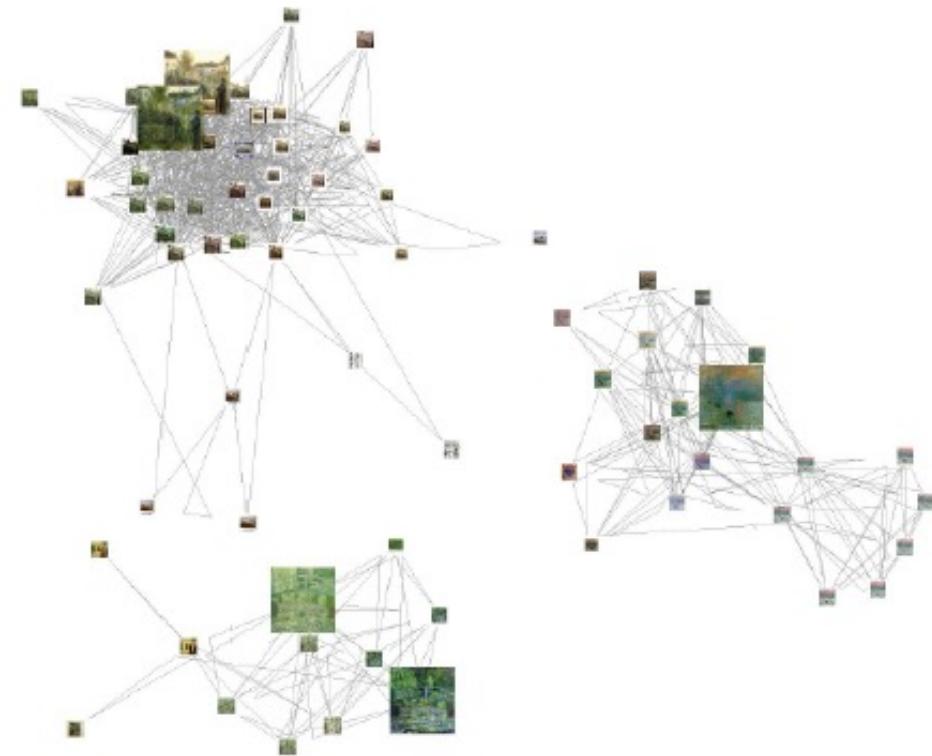
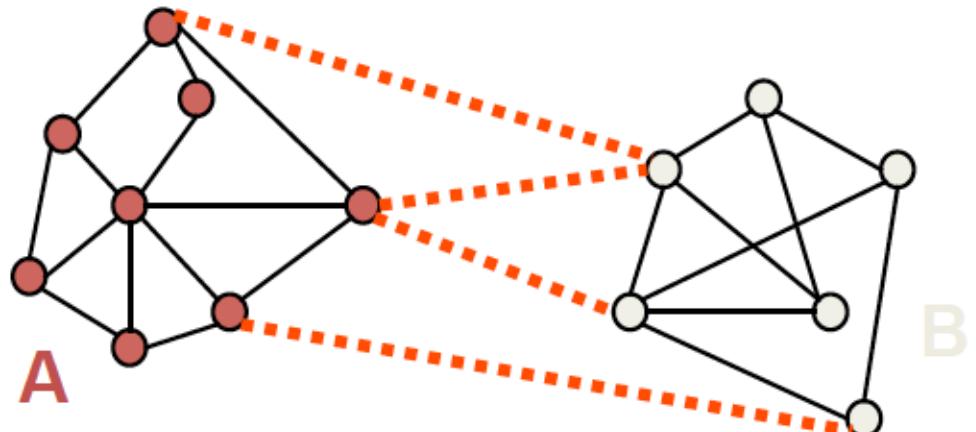
Spectral Clustering: Examples



[Figures from Ng,
Jordan, Weiss NIPS'01]

Spectral Clustering

- Group data points based on links in a graph

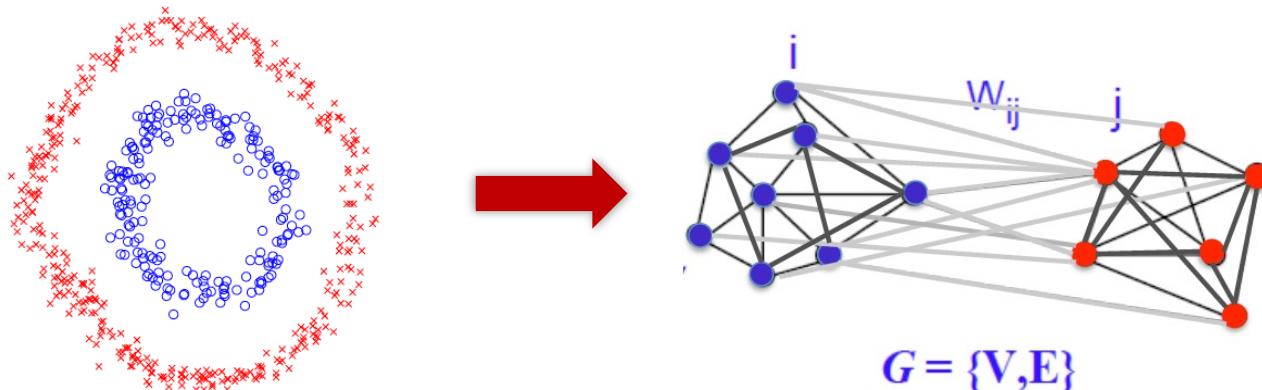


How to Create Graphs?

- It is common to use a Gaussian Kernel to compute similarity between objects

$$W(i, j) = \exp \frac{-|x_i - x_j|^2}{\sigma^2}$$

- One could create
 - A fully connected graph
 - K-nearest neighbor graph (each node is only connected to its K-nearest neighbors)





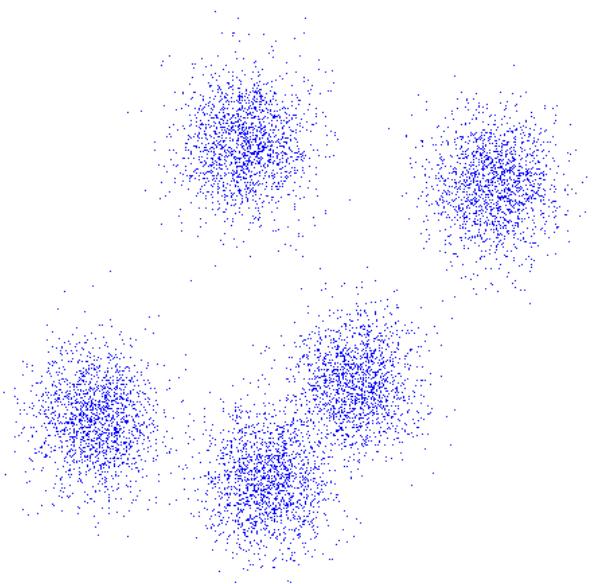
COSC 3337
Data Science I
Section 14623

Clustering (contd.)

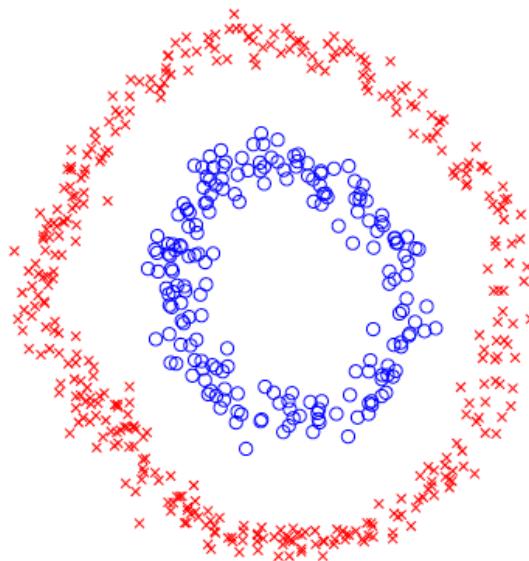
Instructor: Jingchao Ni
Fall 2024

Spectral Clustering

- Two different clustering criteria
 - Compactness, e.g., k-means, mixture models
 - Connectivity, e.g., spectral clustering

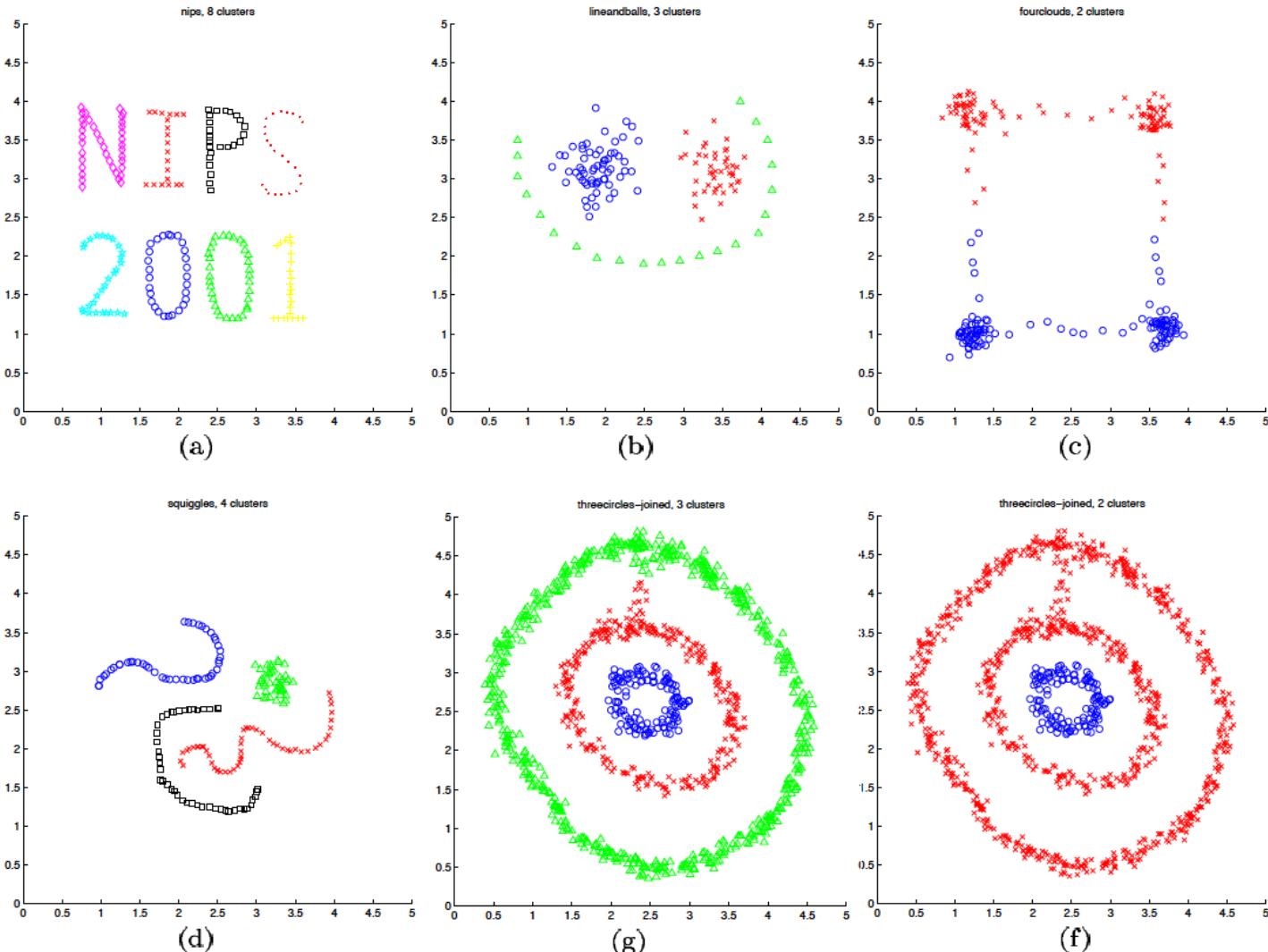


Compactness



Connectivity

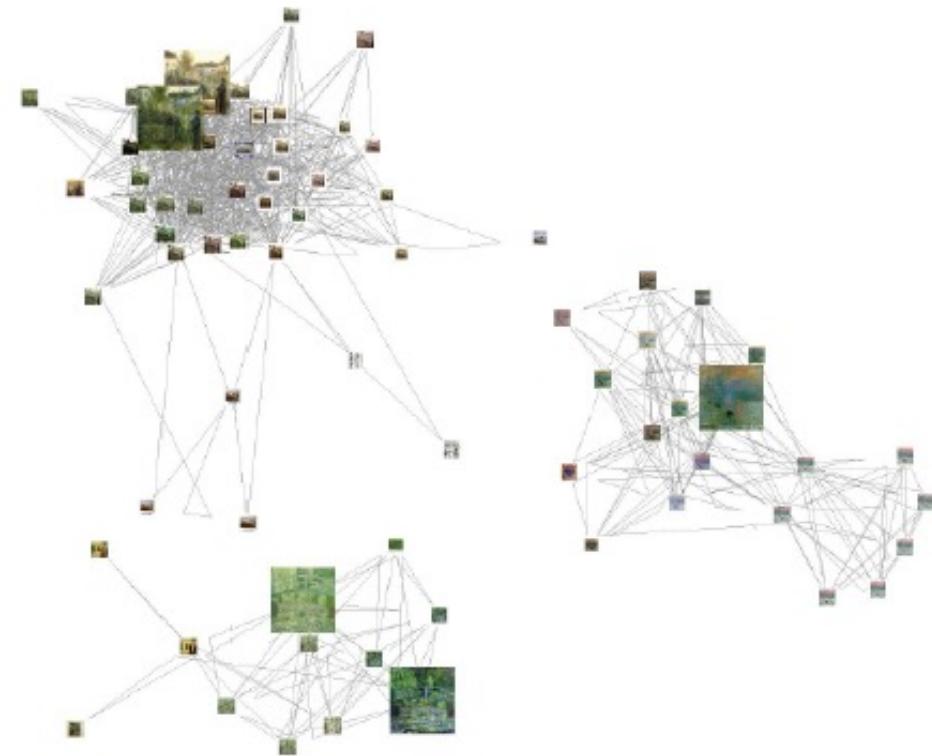
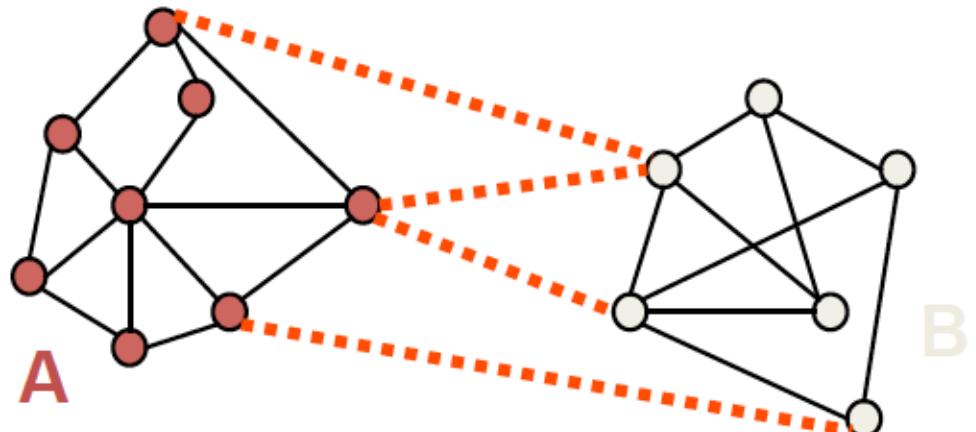
Spectral Clustering: Examples



[Figures from Ng,
Jordan, Weiss NIPS'01]

Spectral Clustering

- Group data points based on links in a graph

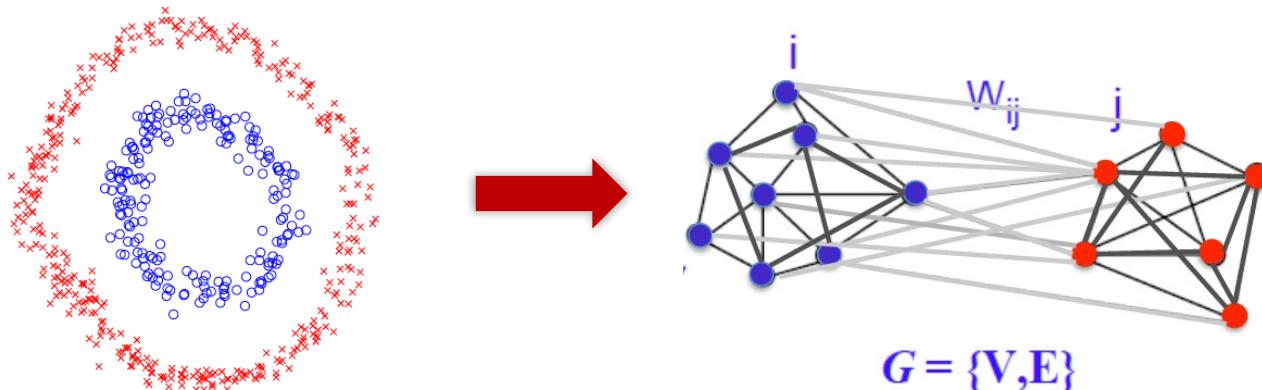


How to Create Graphs?

- It is common to use a Gaussian Kernel to compute similarity between objects

$$W(i, j) = \exp \frac{-|x_i - x_j|^2}{\sigma^2}$$

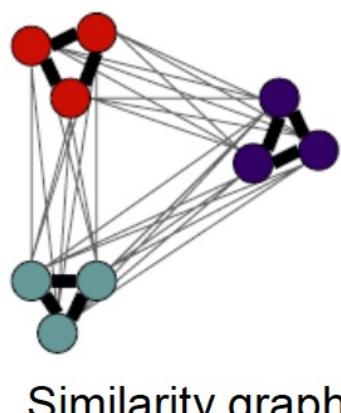
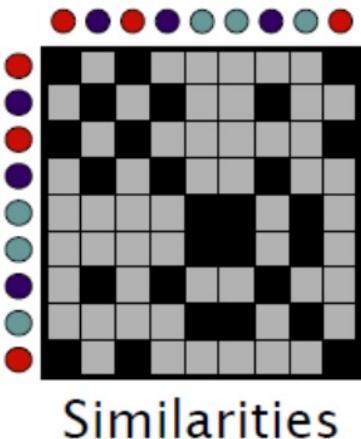
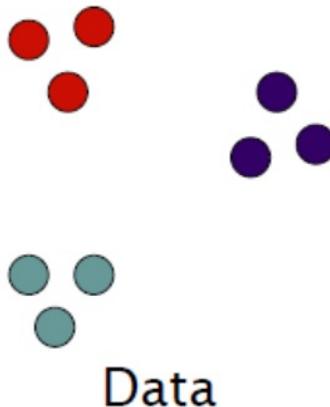
- One could create
 - A fully connected graph
 - K-nearest neighbor graph (each node is only connected to its K-nearest neighbors)



Graph Clustering

- **Goal:** Given data points X_1, \dots, X_n and similarities $W(X_i, X_j)$, partition the data into groups so that points in a group are similar and points in different groups are dissimilar
- Similarity Graph: $G(V, E, W)$

V: Vertices (Data Points)
E: Edge (If similarity > 0)
W: Edge weights (similarities)

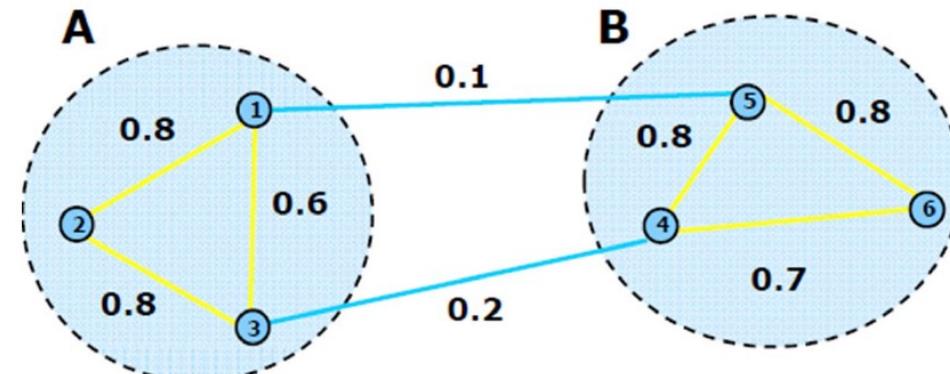


Partition the graph so that edges within a group have large weights and edges across groups have small weights

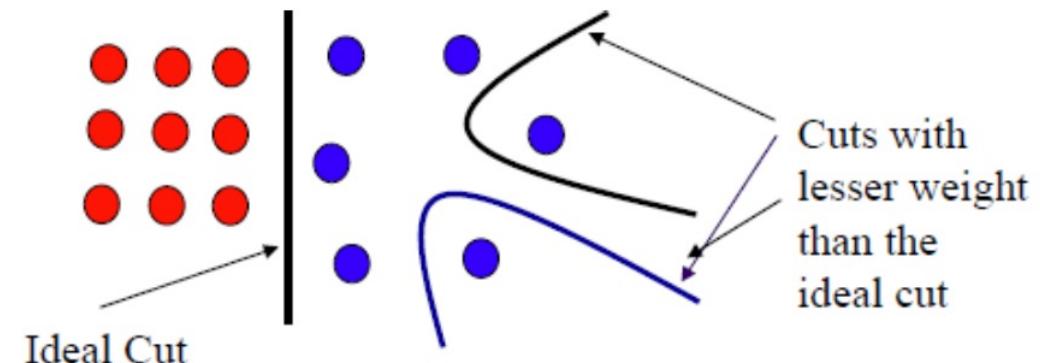
Partitioning A Graph into Two Clusters

- **Min-Cut:** Partition a graph into two sets A and B such that the weights of edges connecting the vertices in A and the vertices in B is minimum
 - $\text{Cut}(A, B)$: sum of the weights of the set of edges that connect the vertices in the two groups

$$\text{cut}(A, B) = \sum_{i \in A, j \in B} w_{ij} = 0.3$$



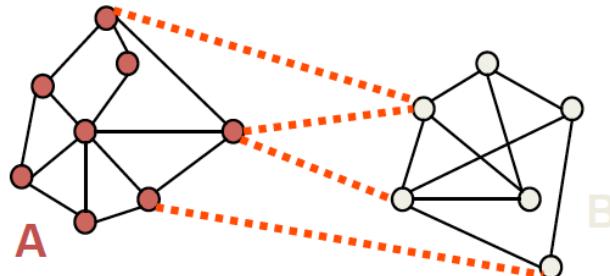
- Problem: often isolates vertices



Partitioning A Graph into Two Clusters

- **Min-Cut:** Partition a graph into two sets A and B such that the weights of edges connecting the vertices in A and the vertices in B is minimum & the sizes of A and B are large

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



- Balanced Min-Cut

$$\min_{A,B} Cut(A, B) \quad \text{s.t. } |A| = |B| \quad (\text{or } |A|, |B| \geq \delta)$$

- Ratio Cut

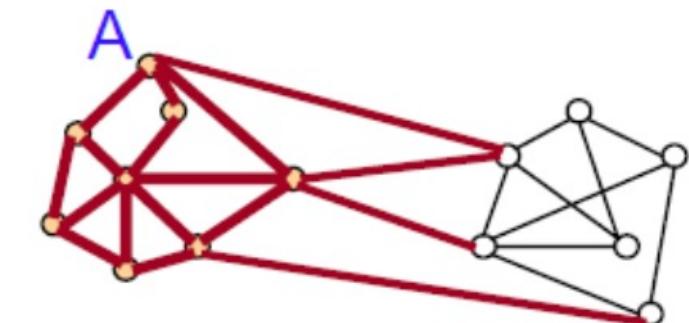
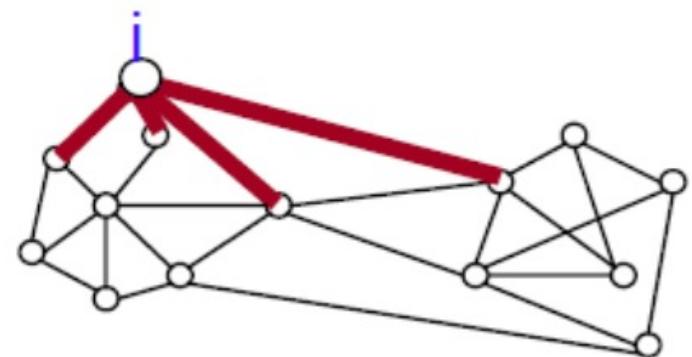
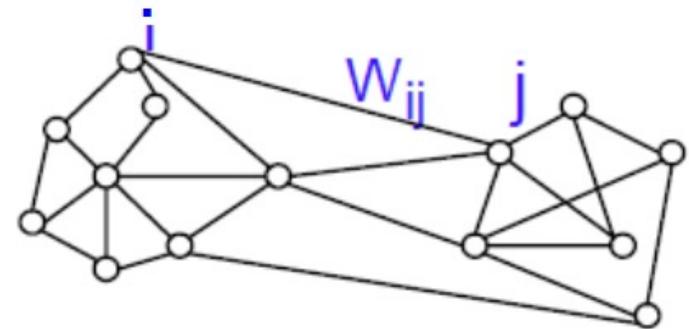
$$\text{RatioCut}(A, B) = Cut(A, B) \left(\frac{1}{|A|} + \frac{1}{|B|} \right)$$

NP-Hard to solve
Spectral clustering is
a relaxation of these

- Normalized Cut $\text{Ncut}(A, B) = Cut(A, B) \left(\frac{1}{\text{Vol}(A)} + \frac{1}{\text{Vol}(B)} \right)$

Graph Terminologies

- $W=(w_{ij})$: **adjacency matrix** of the graph
- $d_i = \sum_j w_{ij}$: **degree** of a vertex i
- $D=\text{diag}(d_1, \dots, d_n)$: **degree matrix** of the graph
- $|A|$ = number of vertices in set A
- $\text{Vol}(A)= \sum_{i \in A} d_i$: **volume** of set A

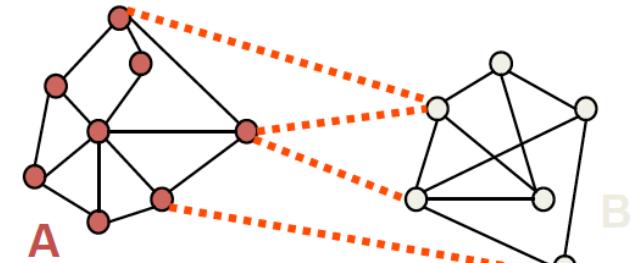


Graph Cut

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

Choose $f = [f_1, \dots, f_n]^T$ with $f_i = \begin{cases} 1 & \text{if } x_i \in A \\ -1 & \text{if } x_i \in B \end{cases}$

$$Cut(A, B) = \sum_{i \in A, j \in B} w_{ij} = \frac{1}{4} \sum_{i,j} w_{ij} (f_i - f_j)^2 = \frac{1}{2} f^T (D - W) f$$



$$\begin{aligned} 2\text{RHS} &= f^T (D - W) f = f^T D f - f^T W f = \sum_i d_i f_i^2 - \sum_{i,j} f_i f_j w_{ij} \\ &= \frac{1}{2} \left(\sum_i \left(\sum_j w_{ij} \right) f_i^2 - 2 \sum_{ij} f_i f_j w_{ij} + \sum_j \left(\sum_i w_{ij} \right) f_j^2 \right) \\ &= \frac{1}{2} \sum_{ij} w_{ij} (f_i - f_j)^2 = 2\text{LHS} \end{aligned}$$

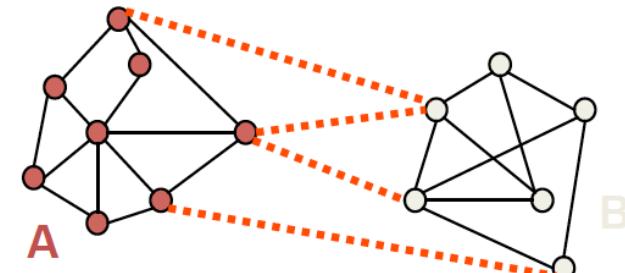
Graph Cut and Graph Laplacian

$$\begin{aligned}Cut(A, B) &= \sum_{i \in A, j \in B} w_{ij} \\&= \frac{1}{2} f^T (D - W) f = \frac{1}{2} f^T L f\end{aligned}$$

$L = D - W$: Un-normalized Graph Laplacian

- Spectral properties of L
 - Smallest eigenvalue of L is 0, corresponding eigenvector is $\mathbf{1}$
 - Eigenvalues and eigenvectors: $A\mathbf{v} = \lambda\mathbf{v}$

$$L\mathbf{1} = D\mathbf{1} - W\mathbf{1} = \begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_n \end{bmatrix} - \begin{bmatrix} \sum_j w_{1j} \\ \sum_j w_{2j} \\ \vdots \\ \sum_j w_{nj} \end{bmatrix} = 0$$



Balanced Min-Cut

$$\min_{A,B} \text{Cut}(A, B) \quad \text{s.t. } |A| = |B|$$



$$\begin{array}{ll} \min & f^T L f \\ f \in \{-1,1\}^n & \text{s.t. } f^T \mathbf{1} = 0 \end{array}$$

(since $\sum f_i = \sum \mathbf{1}_{i \in A} - \mathbf{1}_{i \in B} = 0$)

- The above formulation is NP-Hard, so we relax f not to be binary

$$\begin{array}{ll} \min & f^T L f \\ f \in \mathbb{R}^n & \text{s.t. } f^T \mathbf{1} = 0, \quad f^T f = n \end{array}$$

$$\begin{array}{ll} \min & \frac{f^T L f}{f^T f} \\ f \in \mathbb{R}^n & \text{s.t. } f^T \mathbf{1} = 0 \end{array}$$

Relaxation of Balanced Min-Cut

$$\min_{f \in R^n} \frac{f^T L f}{f^T f} \quad \text{s.t.} \quad f^T 1 = 0$$

||

$\lambda_{\min}(L)$ - smallest eigenvalue of L (Rayleigh-Ritz theorem)

- If f is an eigenvector of L , then

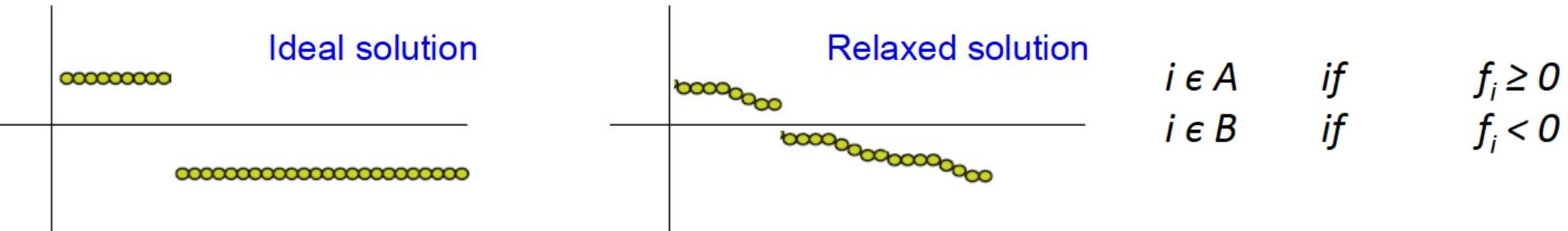
$$\frac{f^T L f}{f^T f} = \frac{f^T \lambda f}{f^T f} = \lambda$$

- Recall that the smallest eigenvalue of L is 0 with corresponding eigenvector 1, but f can't be 1 according to $f^T 1 = 0$
- Therefore, solution f is the eigenvector of L corresponding to the second smallest eigenvalue, a.k.a. second eigenvector

Approximation of Balanced Min-Cut

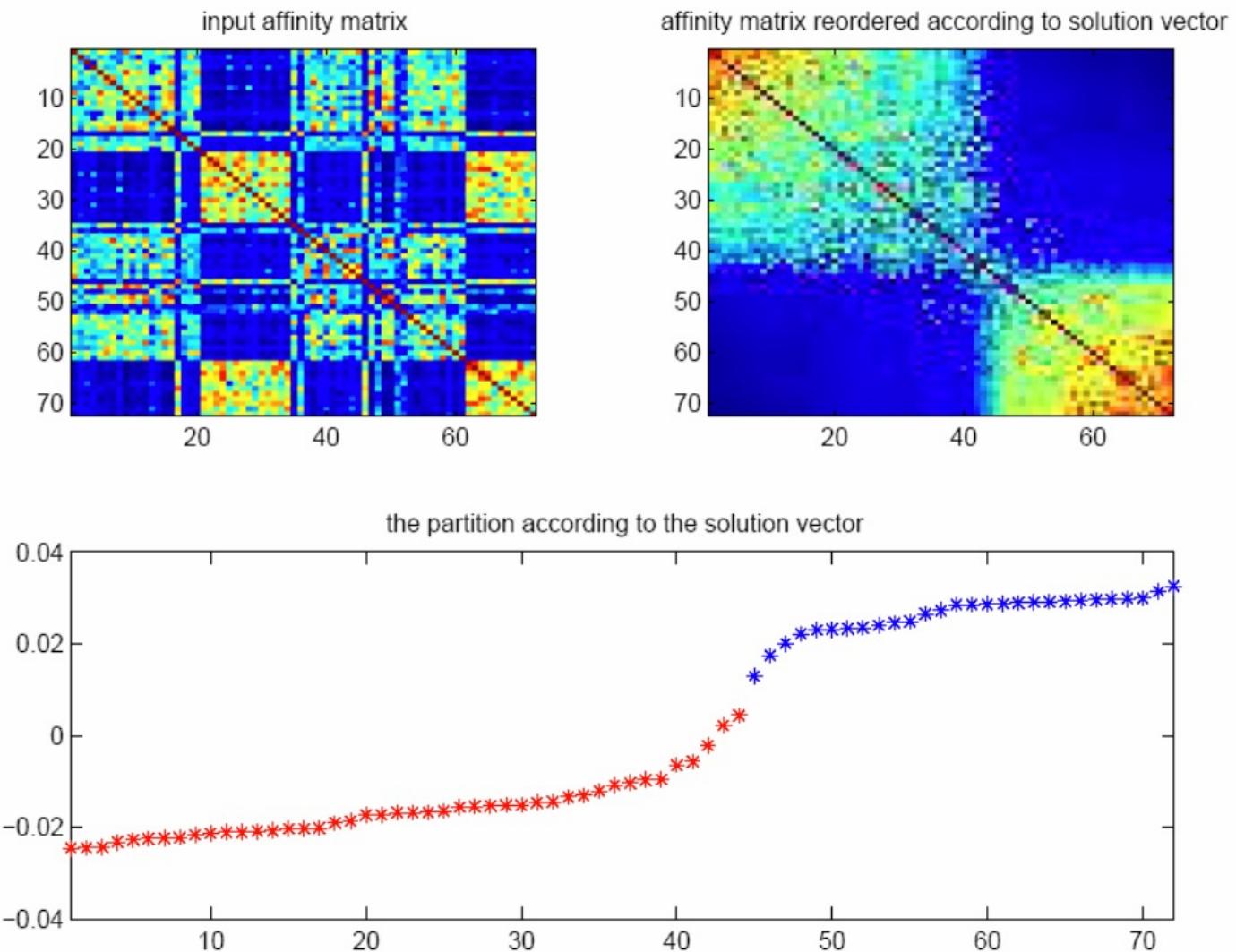
$$\min_{A,B} \text{Cut}(A, B) \quad \text{s.t. } |A| = |B|$$

- Let f be the second eigenvector of the unnormalized graph Laplacian L .
- Recover binary partition as follows:



- Similar relaxations work for other cut problems:
 - RatioCut: second eigenvector of unnormalized graph Laplacian $L=D-W$
 - NCut: second eigenvector of normalized Laplacian $L'=I - D^{-1}W$

Example



[Xing et al. 2001]

Spectral Clustering Algorithm

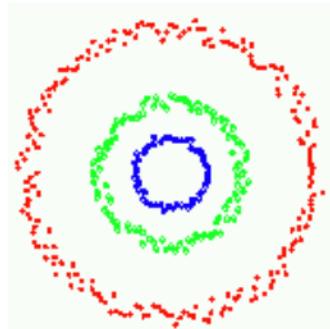
- Input: an N by N similarity matrix W , number of clusters K
 - Compute the first K eigenvectors v_1, \dots, v_k for the Laplacian matrix
 - L : for unnormalized spectral clustering
 - L' : for normalized spectral clustering
 - Build a matrix $V \in \mathbb{R}^{N \times K}$ with the first K eigenvectors as columns
 - Interpret the rows of V as new data points $V_{i:} \in \mathbb{R}^K$
 - Cluster the points $V_{i:}$ with the k-means clustering algorithm in \mathbb{R}^K

Dimension Reduction: $N \times N \rightarrow N \times K$

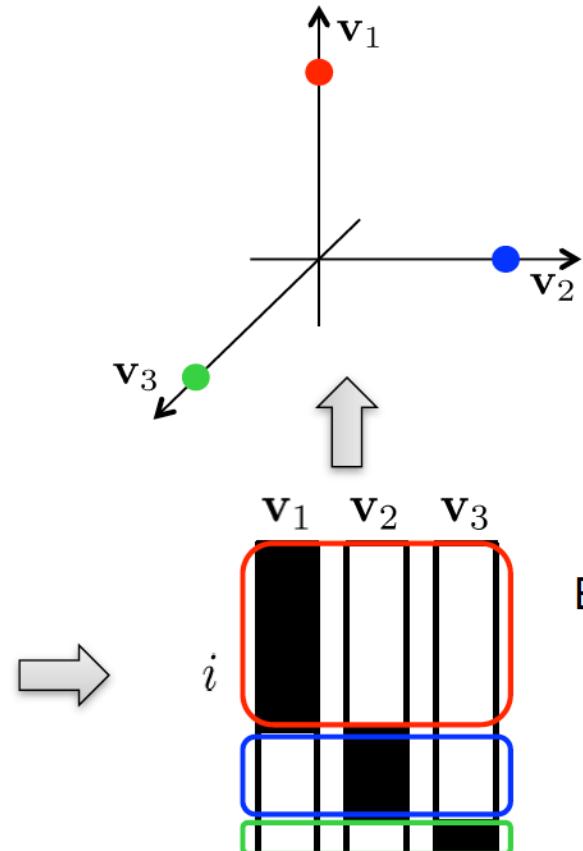
Spectral Clustering Intuition

- Eigenvectors of the Laplacian matrix provide an embedding of the data based on similarity.

Disconnected subgraphs



$$L = \begin{bmatrix} \text{gray} & 0 \\ 0 & \text{white} \end{bmatrix}$$

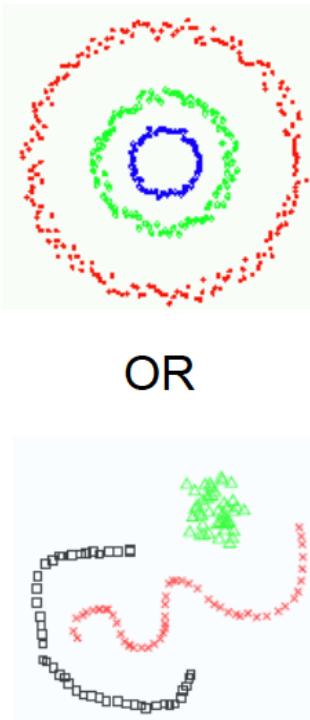


Points are easy to cluster in embedded space
e.g. using k-means

Embedding of point i
 $[v_1(i), v_2(i), v_3(i)]$

Spectral Clustering: Understanding

- If a graph is connected, the first eigenvector of L is constant (all 1s)
- If a graph is disconnected (K connected components), L is block diagonal and the first K eigenvectors are as following.



$$L = \begin{pmatrix} L_1 & & & \\ & \ddots & & 0 \\ & & L_2 & \\ & \ddots & & \\ 0 & & & L_3 \end{pmatrix}$$

First three eigenvectors

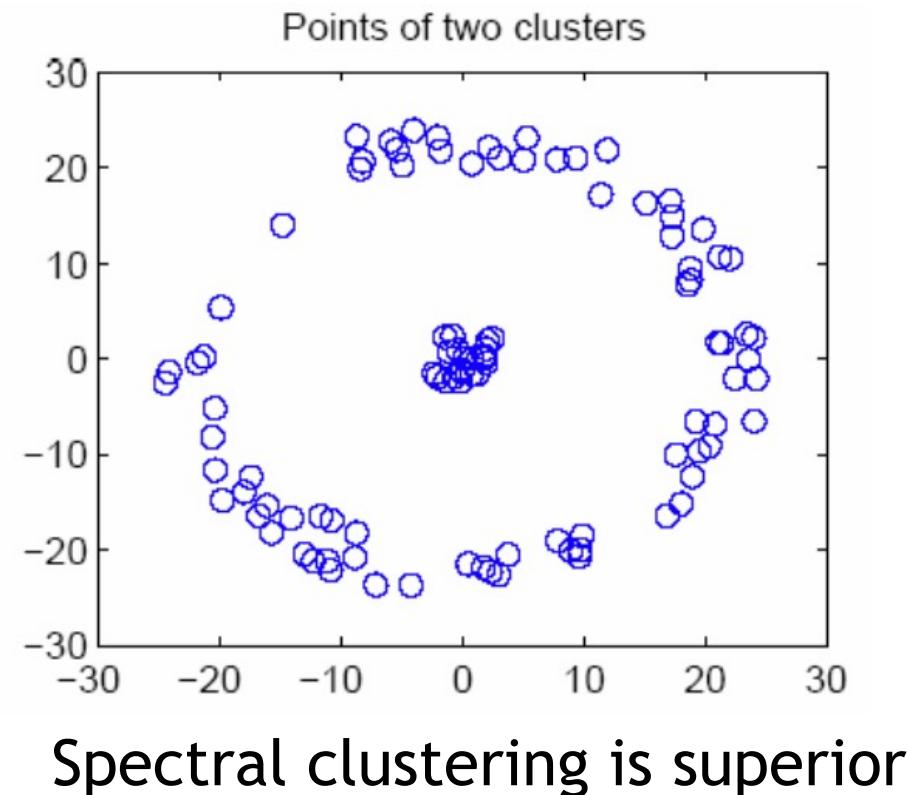
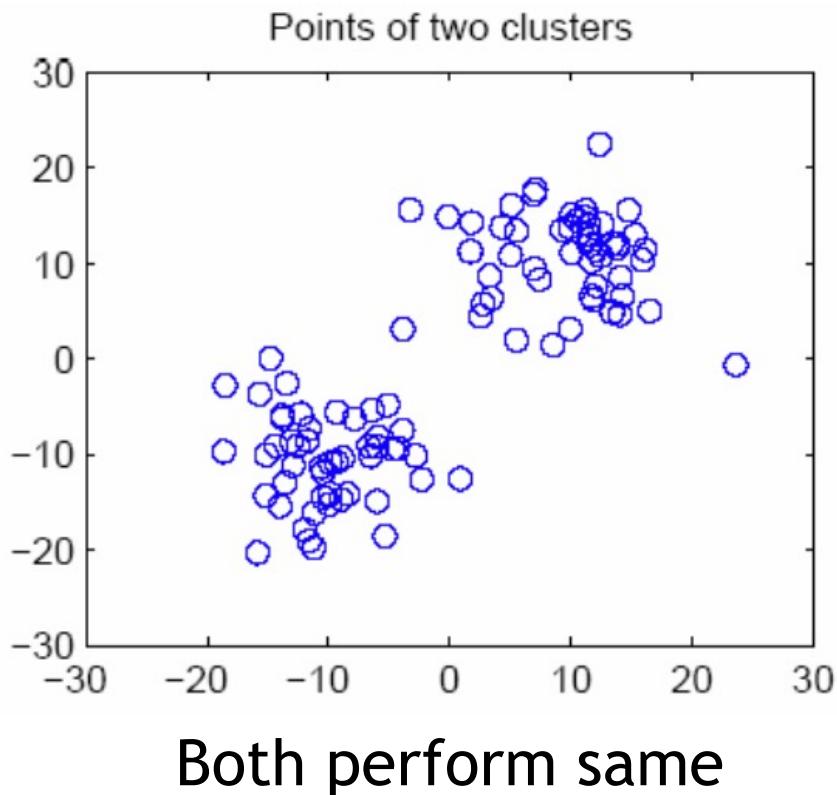
1	0	0
0	1	0
0	0	0

0	0	0
0	1	0
0	0	0

0	0	0
0	0	1
0	1	0

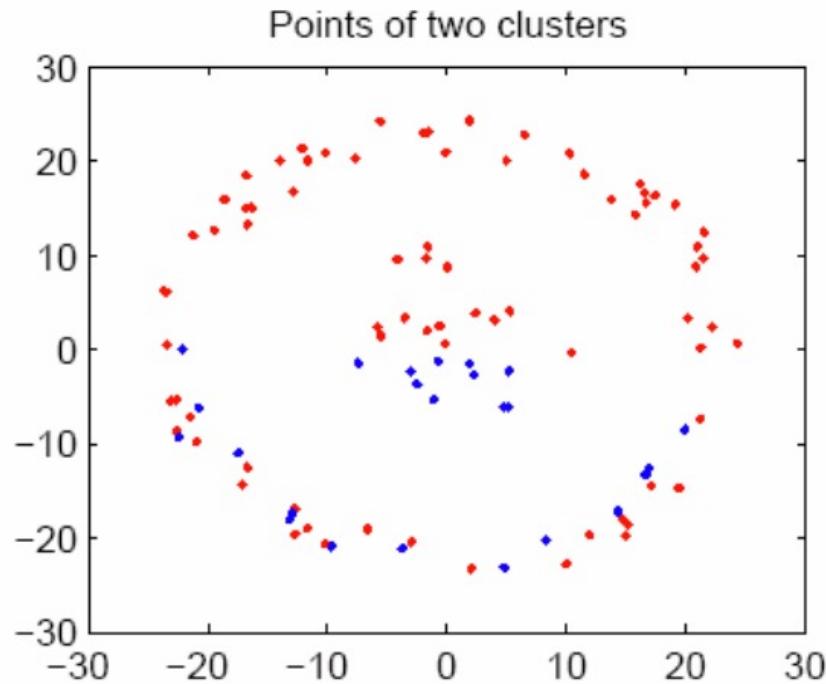
Spectral Clustering with K-Means

- Applying k-means to Laplacian eigenvectors allows us to find clusters with non-convex boundaries.

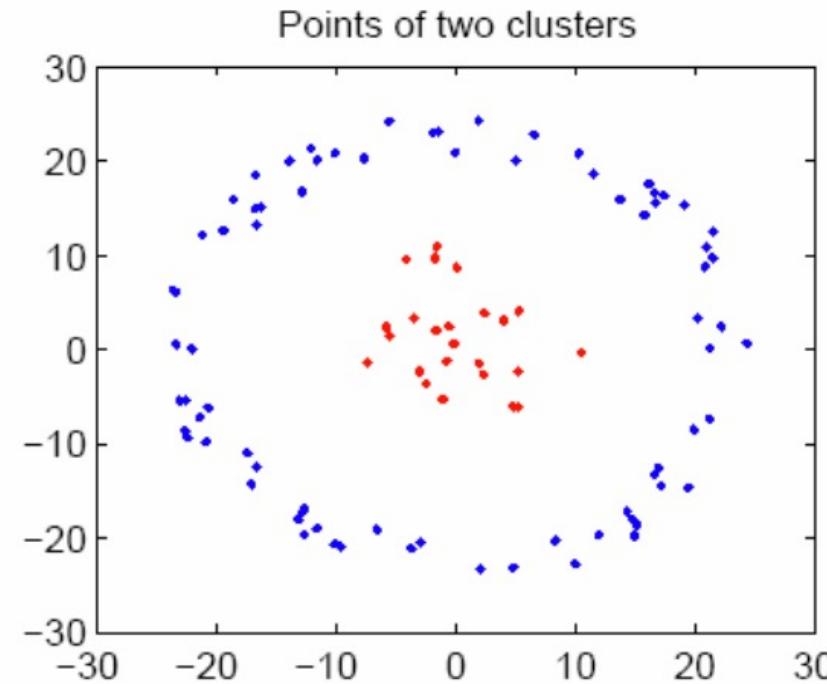


Spectral Clustering with K-Means

- Applying k-means to Laplacian eigenvectors allows us to find clusters with non-convex boundaries.



K-means output

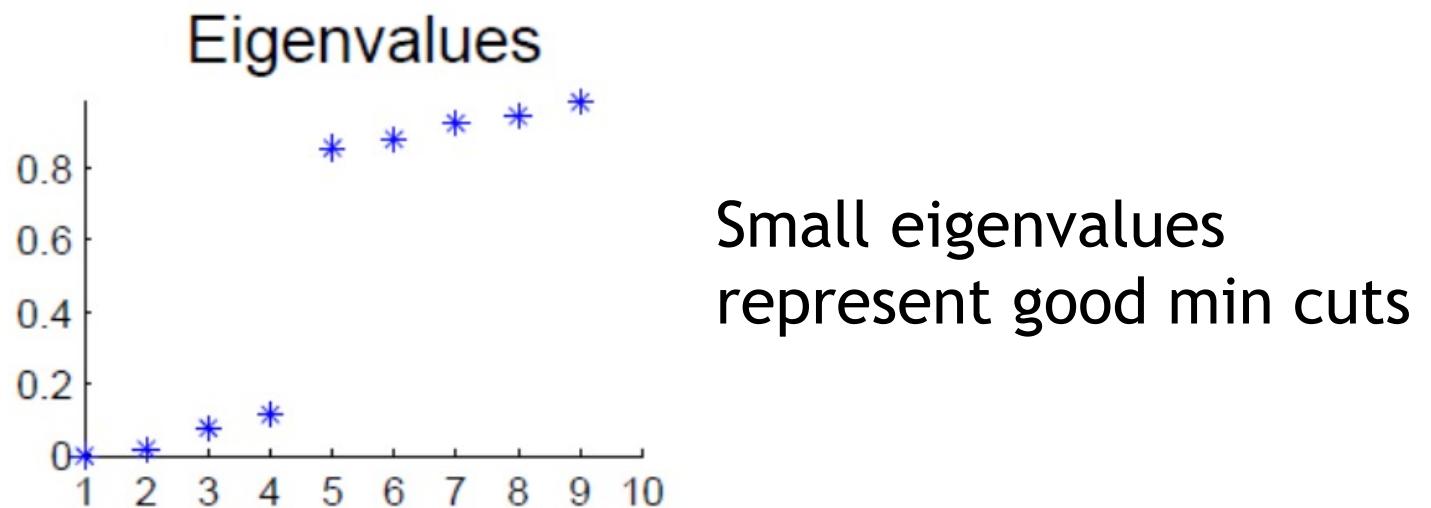


Spectral clustering output

Spectral Clustering: Choice of K

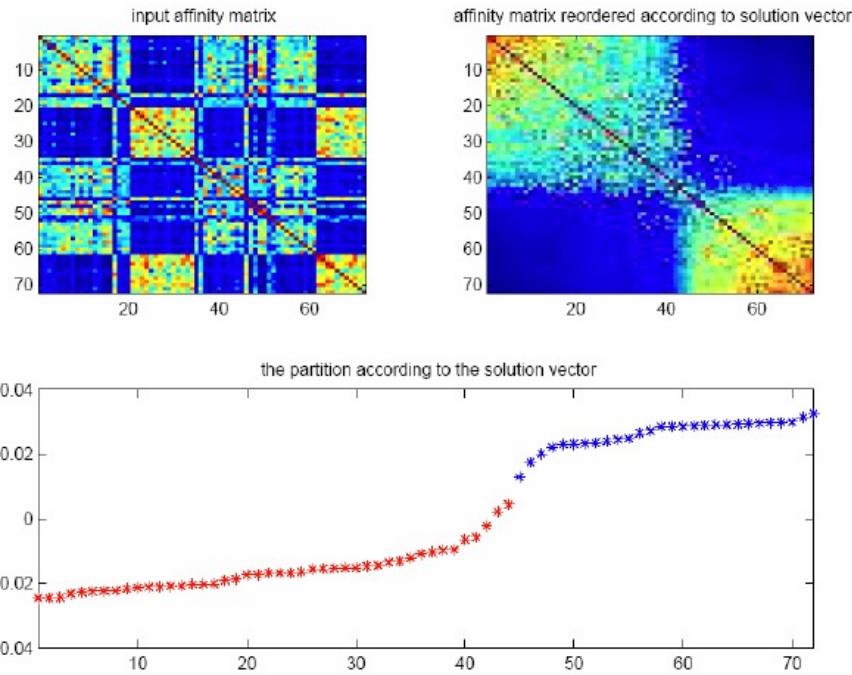
- Choice of the number of clusters K
 - Most stable clustering is usually given by the value of K that maximizes the eigengap (difference between two consecutive eigenvalues)

$$\Delta_k = |\lambda_{k+1} - \lambda_k|$$

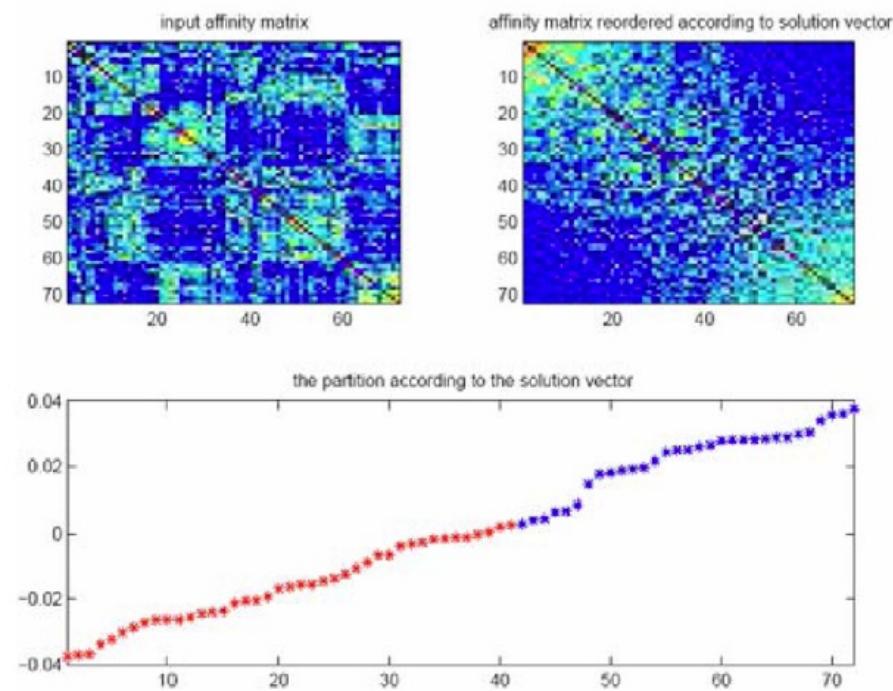


Spectral Clustering: Similarity

- Choice of similarity measure
 - Choice of kernel function
 - For Gaussian kernel, choice of σ



Good similarity measure



Poor similarity measure

Summary of Spectral Clustering

- Spectral clustering uses connectivity as its clustering criteria
- Spectral clustering builds a graph using some similarity measure (e.g., kernel functions) on the sample-feature matrix
- Spectral clustering minimizes the cut between groups of data points while maximizes the volume of each group
- The algorithm clusters data points using eigenvectors of Laplacian matrix
- It is useful in hard non-convex clustering problems
- The obtained data embedding in the low-dimensional space can be easily clustered
- Different spectral clustering methods use eigenvectors of unnormalized or normalized Laplacian matrix

GMM and Spectral Clustering in Sklearn

```
class sklearn.mixture.GaussianMixture(n_components=1, *,  
covariance_type='full', tol=0.001, reg_covar=1e-06, max_iter=100, n_init=1,  
init_params='kmeans', weights_init=None, means_init=None, precisions_init=None,  
random_state=None, warm_start=False, verbose=0, verbose_interval=10)
```

<https://scikit-learn.org/1.5/modules/generated/sklearn.mixture.GaussianMixture.html#sklearn.mixture.GaussianMixture>

```
class sklearn.cluster.SpectralClustering(n_clusters=8, *, eigen_solver=None,  
n_components=None, random_state=None, n_init=10, gamma=1.0, affinity='rbf',  
n_neighbors=10, eigen_tol='auto', assign_labels='kmeans', degree=3, coef0=1,  
kernel_params=None, n_jobs=None, verbose=False)
```

<https://scikit-learn.org/1.5/modules/generated/sklearn.cluster.SpectralClustering.html>



COSC 3337
Data Science I
Section 14623

Clustering (contd.) and Anomaly Detection

Instructor: Jingchao Ni
Fall 2024

Cluster Validation and Evaluation

- For supervised classification we have a variety of measures to evaluate how good our model is
 - Accuracy, precision, recall
- For cluster analysis, the analogous question is how to evaluate the “goodness” of the resulting clusters?
- But “clusters are in the eye of the beholder”!
- Then why do we want to evaluate them?
 - To avoid finding patterns in noise
 - To compare clustering algorithms
 - To compare two sets of clusters
 - Ultimately, to chose the best clustering result

Measures of Cluster Validity

- Numerical measures that are applied to judge various aspects of cluster validity are classified into the following two types
 - **External Index:** Used to measure the extent to which cluster labels match externally supplied class labels
 - Entropy
 - **Internal Index:** Used to measure the goodness of a clustering structure without respect to external information
 - Sum of Squared Error (SSE)
- Sometimes these are referred to as criteria instead of indices

Measuring Cluster Validity via Correlation

- Two matrices ($n \times n$ matrices)
 - Proximity Matrix (of pairwise distances)
 - Cluster Co-occurrence Matrix
 - One row and one column for each data point
 - An entry is 1 if the associated pair of points belong to the same cluster
 - An entry is 0 if the associated pair of points belongs to different clusters
- Compute the correlation between the two matrices
 - Since the matrices are symmetric, only the correlation between $n(n-1) / 2$ entries needs to be calculated
- High negative correlations indicate that points that belong to the same cluster are close to each other
- Not a good measure for some density or contiguity based clusters
 - E.g., DBSCAN, spectral clustering

Example Cluster Validity Via Correlation

- We assume we have 5 objects and a clustering is $\{1,2\} \{3,4,5\}$. The cluster co-occurrence matrix for this clustering and the 5-object distance matrix are:

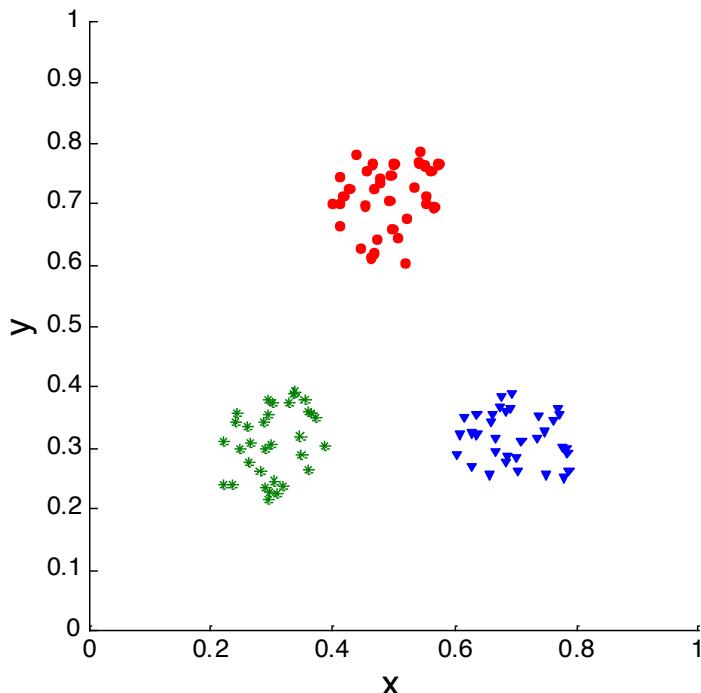
1	0	0	0	
	0	0	0	
		1	1	
			1	
				0

0	2	6	6	7
	0	5	6	2
		0	3	2
			0	1
				0

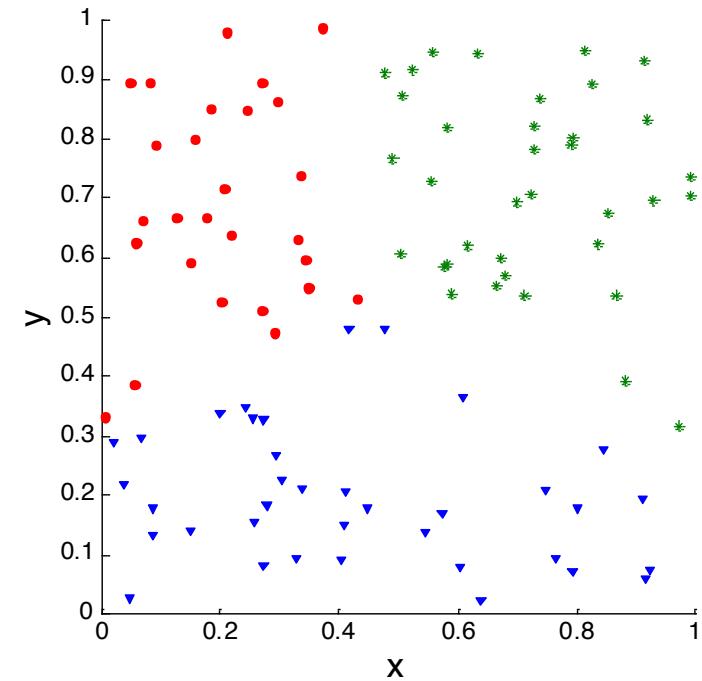
- We compute the correlation of the following vectors:
 - Correlation($(1,0,0,0,0,0,0,1,1,1)$, $(2,6,6,7,5,6,2,3,2,1)$)=-0.78
 - $a <- c(1,0,0,0,0,0,0,1,1,1)$
 - $b <- c(2,6,6,7,5,6,2,3,2,1)$
 - $> cor(a,b)$
 - [1] -0.7784989

Example Cluster Validity Via Correlation

- Correlation of ideal similarity and proximity matrices for the K-means clustering of the following two data sets



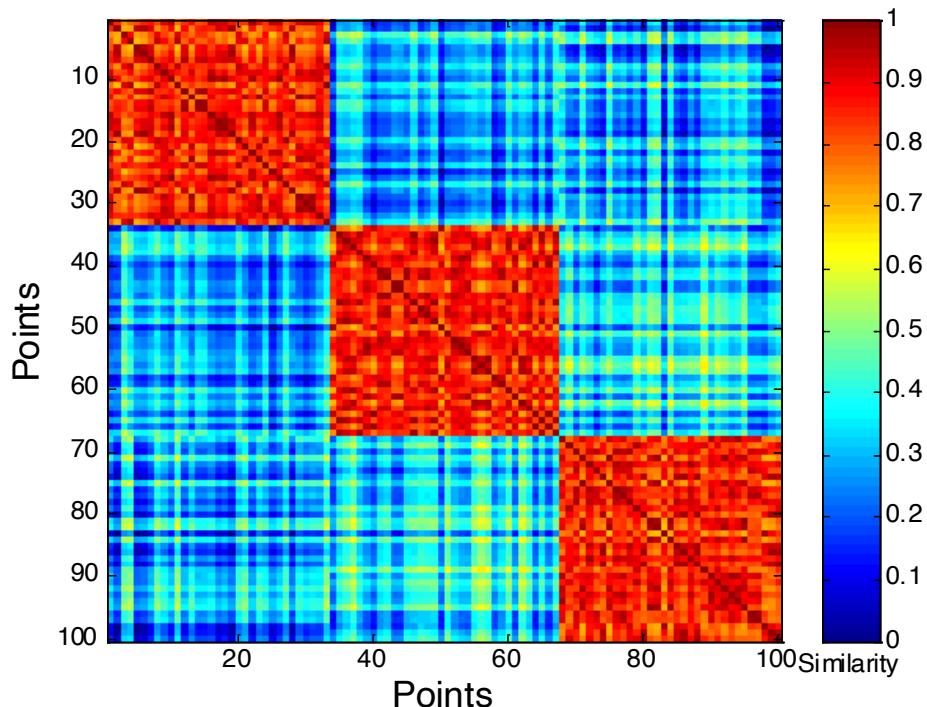
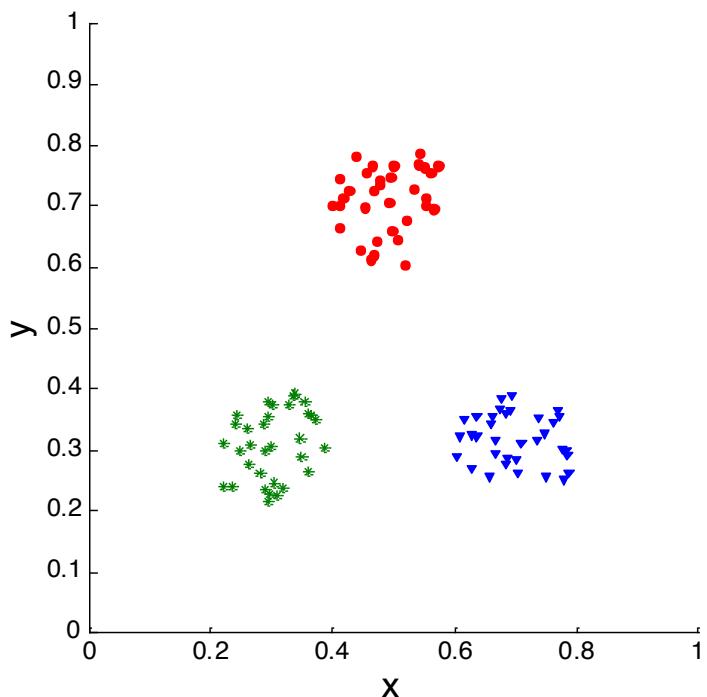
Corr = -0.9235



Corr = -0.5810

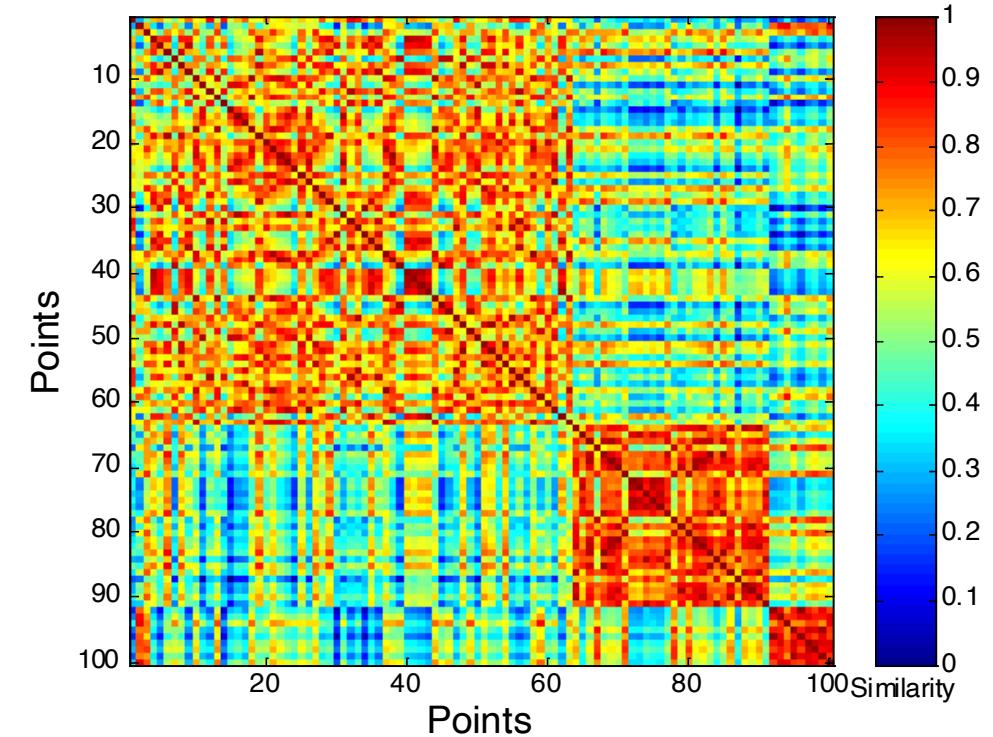
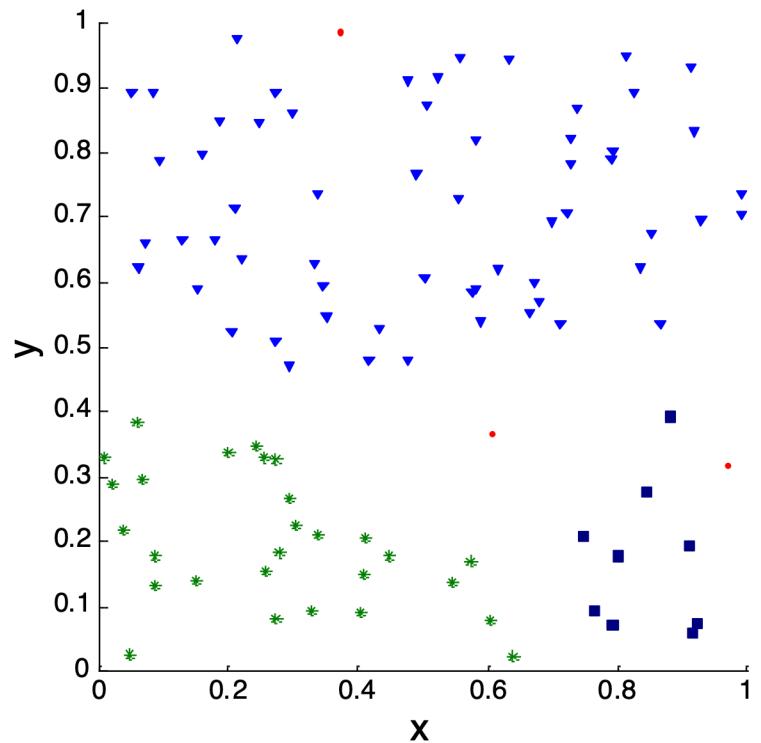
Similarity Matrix for Cluster Validation

- Order the similarity matrix (of pairwise similarities) with respect to cluster labels and inspect visually



Similarity Matrix for Cluster Validation

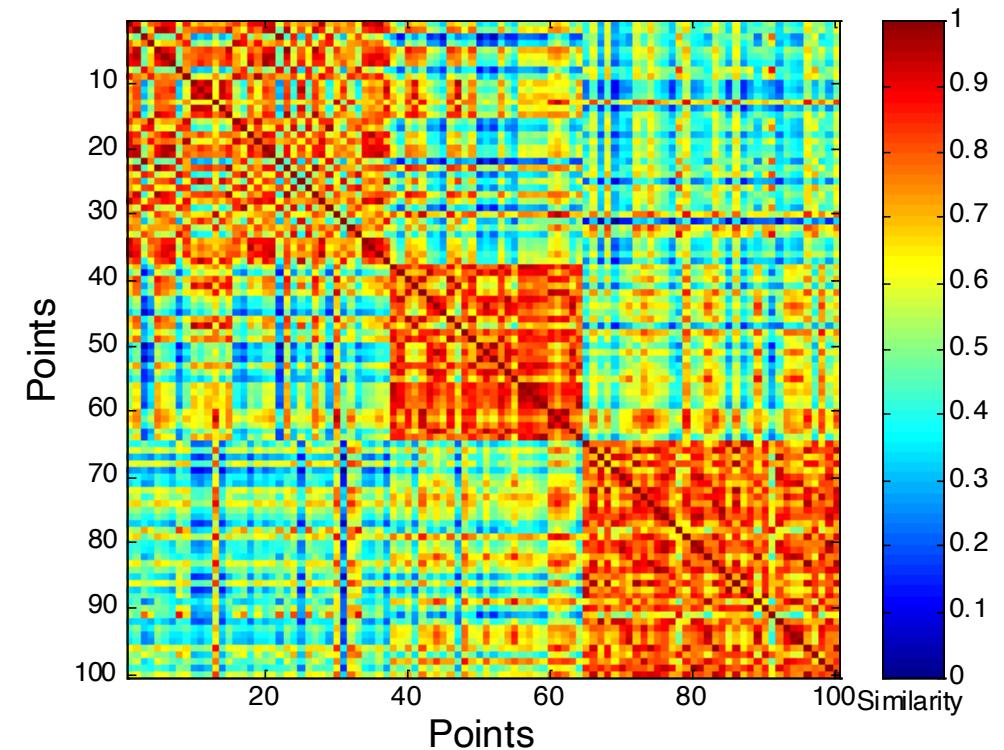
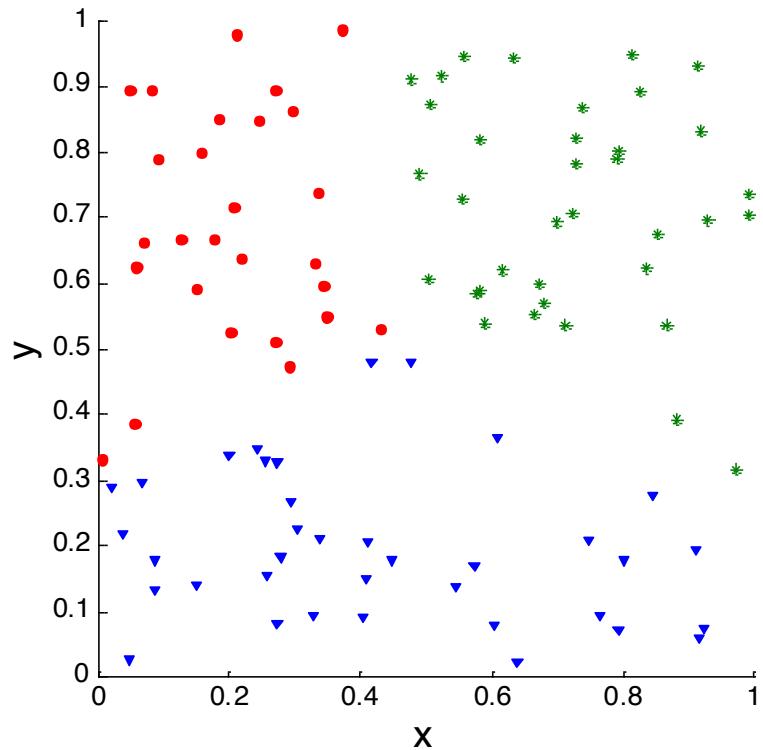
- Clusters in random data are not so clean



DBSCAN

Similarity Matrix for Cluster Validation

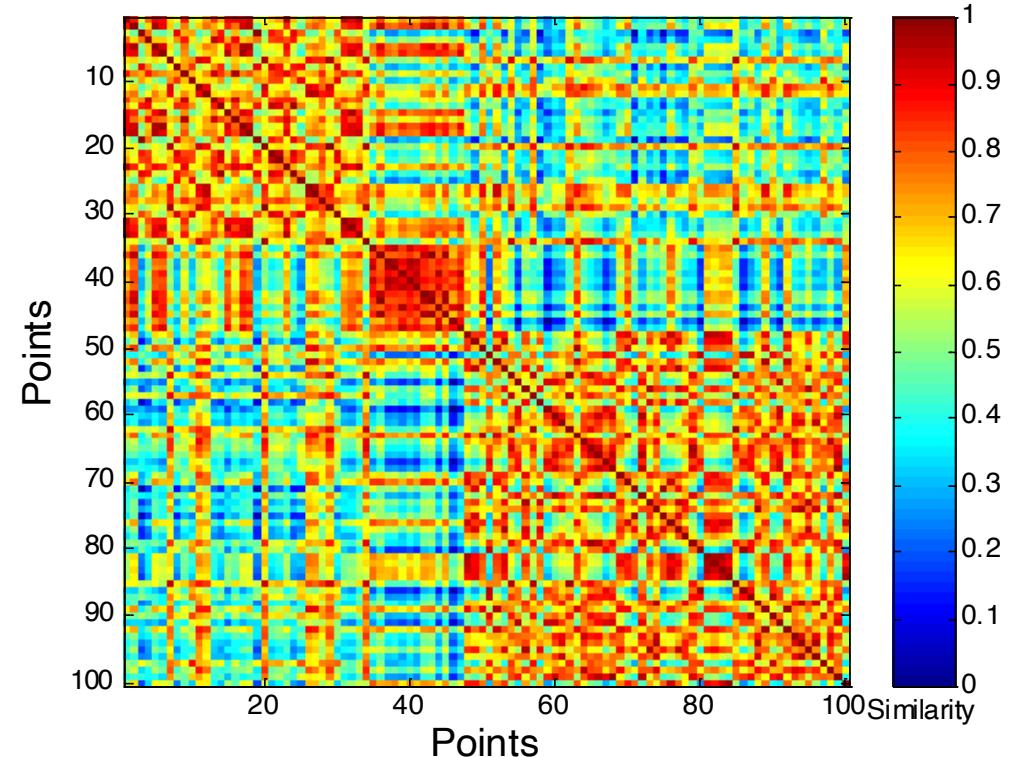
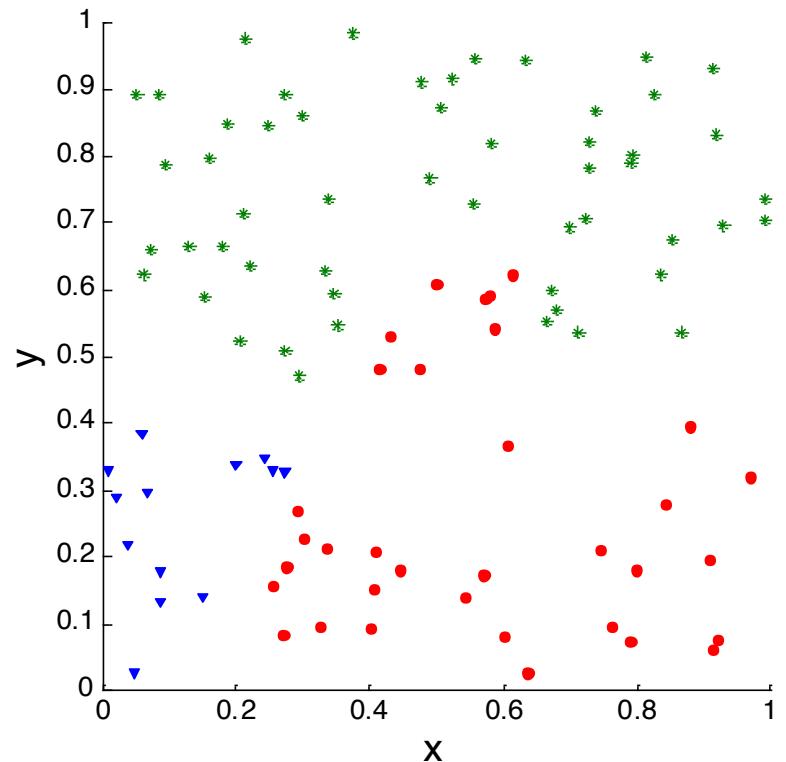
- Clusters in random data are not so clean



K-means

Similarity Matrix for Cluster Validation

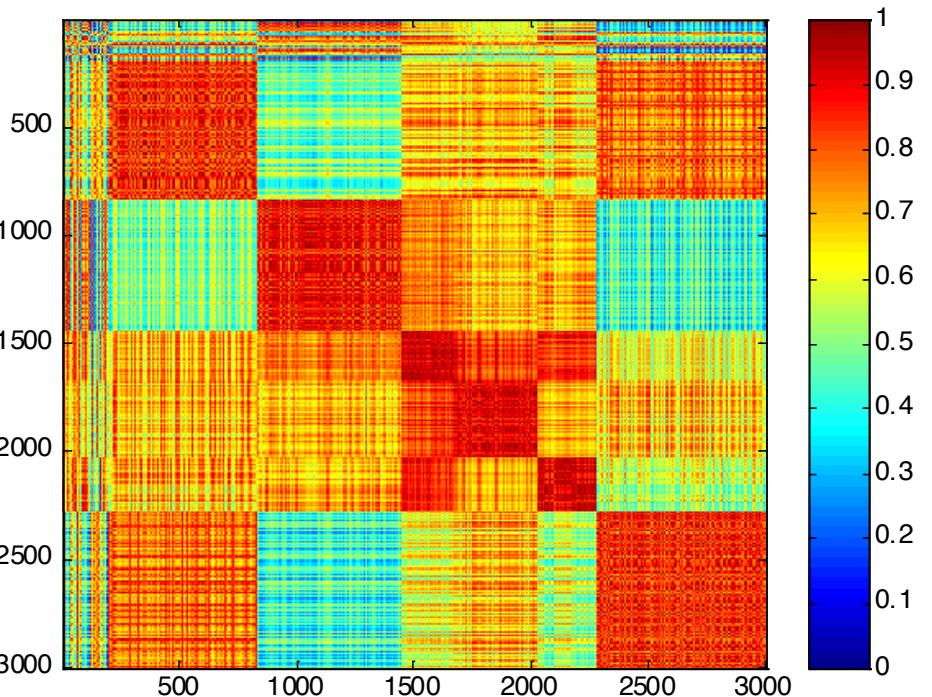
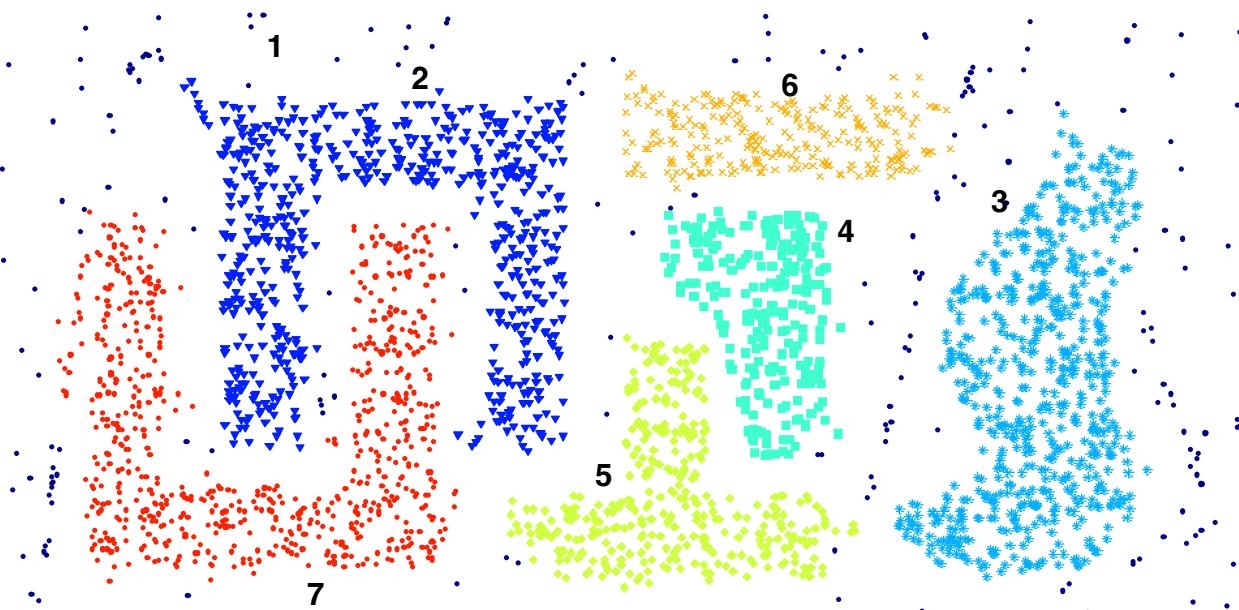
- Clusters in random data are not so clean



Complete Link

Similarity Matrix for Cluster Validation

- Visually inspecting re-ordered similarity matrix does not work for non-convex shape clusters



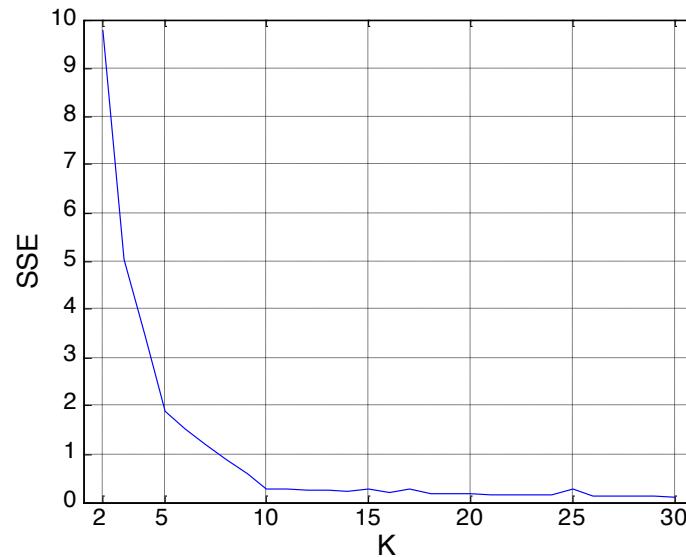
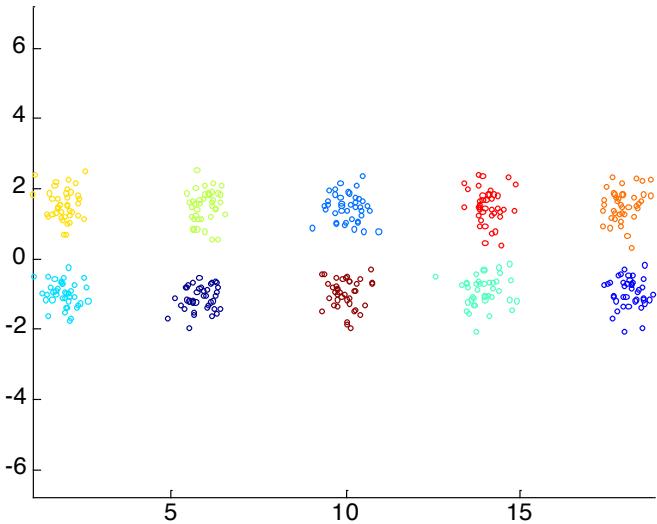
DBSCAN

Internal Measures: SSE

- Internal Index: Used to measure the goodness of a clustering structure without respect to external information

$$SSE(X) = \sum_{i=1}^K \sum_{x \in C_i} dist^2(m_i, x)$$

- SSE is good for comparing two clusters or two clusterings (average)
- Can also be used to determine the number of clusters

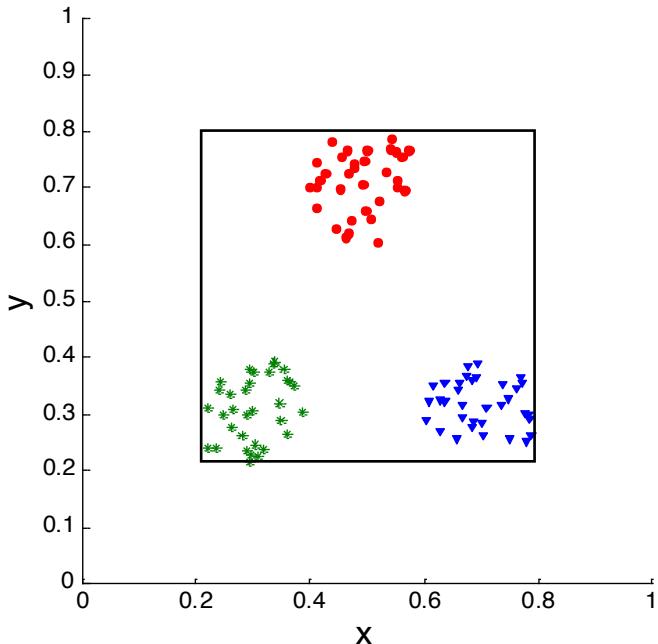


Framework for Cluster Validation

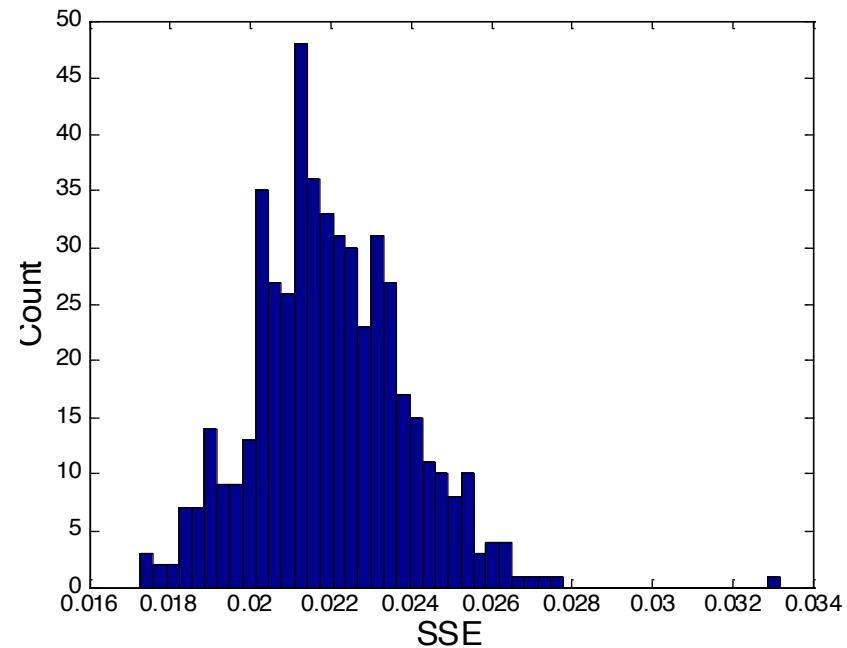
- Need a framework to interpret any measure
 - For example, if our measure of evaluation has the value, 10, is that good, fair, or poor?
- Statistics provide a framework for cluster validity
 - The more “atypical” a clustering result is, the more likely it represents valid structure in the data
 - Can compare the values of an index that result from random data to those of a clustering result
 - If the value of the index is unlikely, then the cluster results are valid

Statistical Framework for SSE

- Example
 - Compare SSE of three cohesive clusters against three clusters in random data



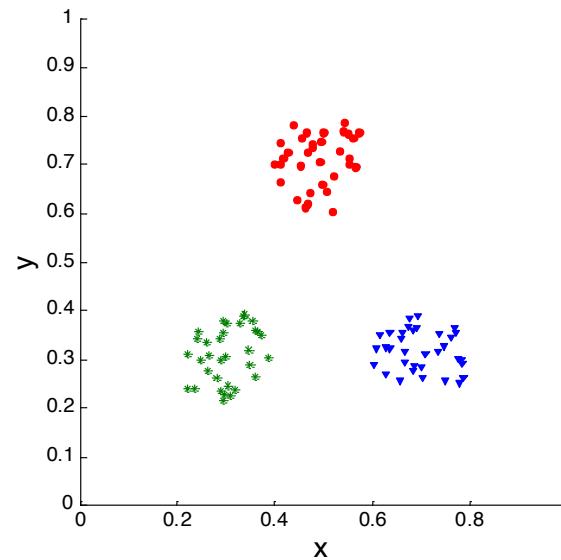
SSE = 0.005



Histogram shows SSE of three clusters in 500 sets of random data points of size 100 distributed over the range 0.2 - 0.8 for x and y values

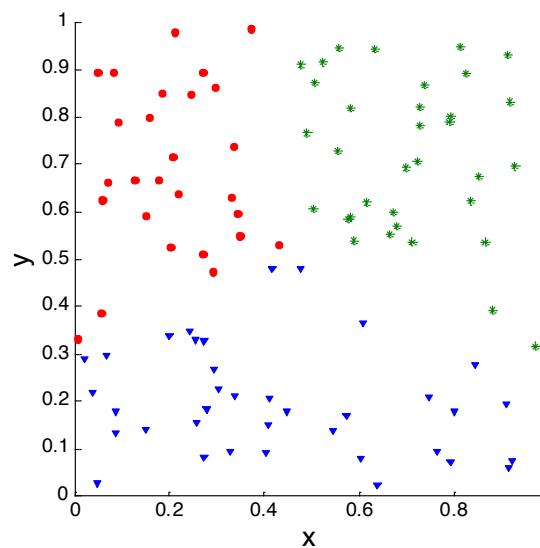
Statistical Framework for Correlation

- Example
 - Compare SSE of three cohesive clusters against three clusters in random data

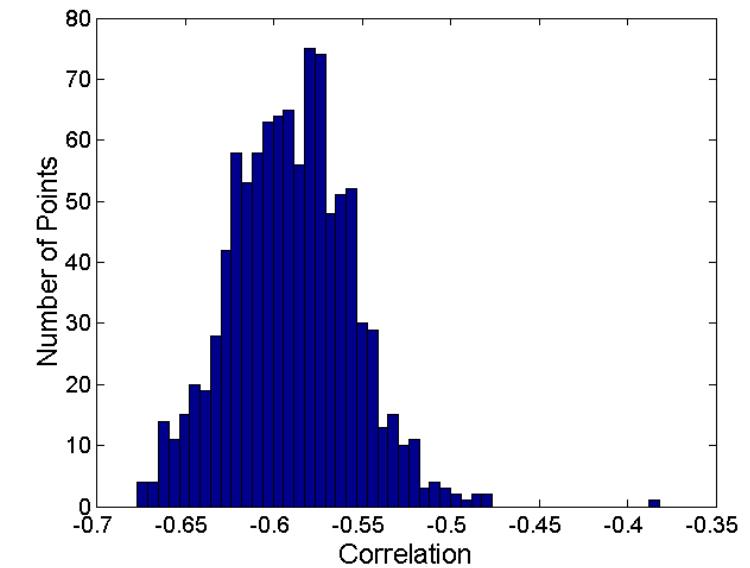


Corr = -0.9235

Correlation is negative because it is calculated between a distance matrix and the cluster co-occurrence matrix. Higher magnitude is better



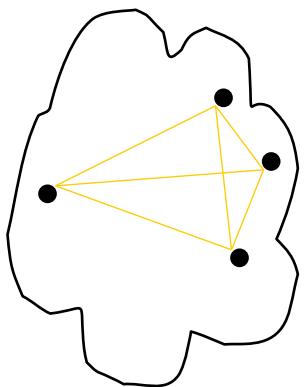
Corr = -0.5810



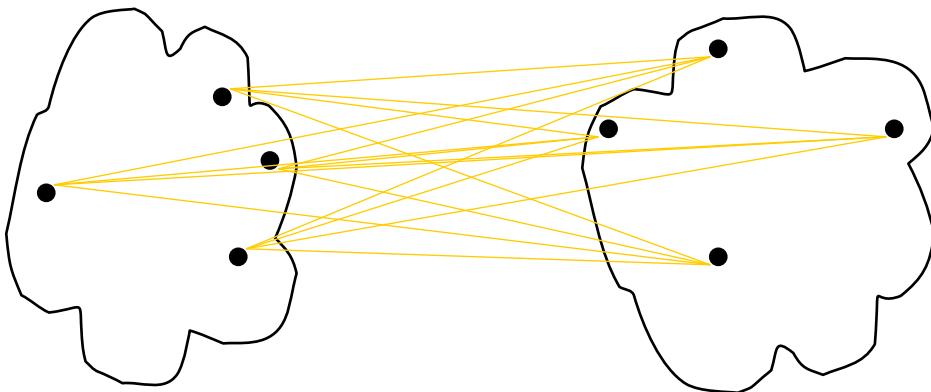
Histogram of correlation for 500 random data sets of size 100 with x and y values of points between 0.2 and 0.8

Internal Measures

- Cohesion and Separation



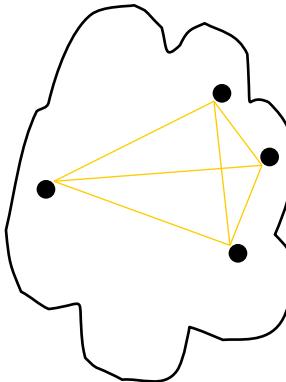
cohesion



separation

Internal Measures

- Cluster Cohesion: Measures how closely related are objects in a cluster
 - Methods to measure cohesion: Average intra cluster distance (takes the average distance of the 4 distances in the figure below), SSE,...



- Cluster Separation: Measures how distinct or well-separated a cluster is from other clusters
 - Measuring method: Average inter cluster distance (same as group average in Hierarchical Clustering)

Example: Cohesion and Separation

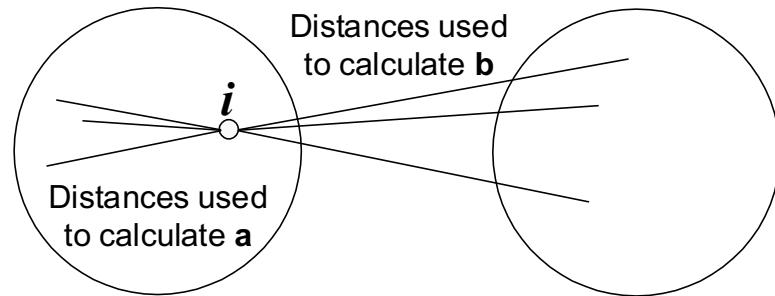
- Assume the following 7-object distance function is given and we have 2 clusters: $C1=\{1,2,3\}$, $C2=\{4,5,6,7\}$

	1	2	5	5	8	9
		3	3	4	7	8
			6	9	9	9
				3	5	2
					2	3
						1

- $\text{Cohesion}(C1) = (1+2+3)/3=2$ “based on average intra cluster distance”
- $\text{Cohesion}(C2) = (3+5+2+2+3+1)/6=2.66$
- $\text{Separation}(C1,C2) = (5+5+8+9+3+4+7+8+6+9+9+9)/12=6.84$

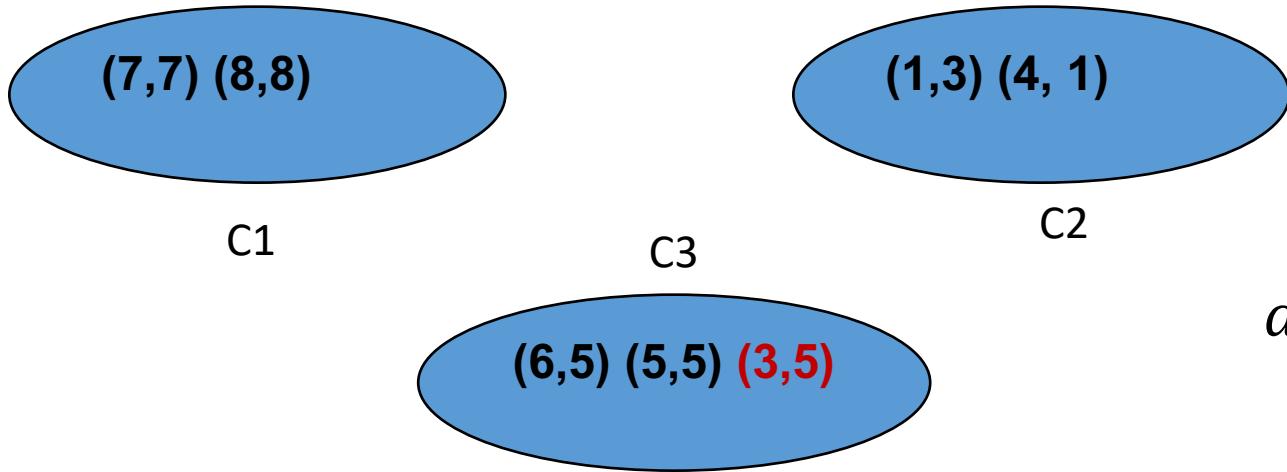
Internal Measures: Silhouette Coefficient

- **Silhouette coefficient** combines ideas of both cohesion and separation, but for individual points, can be extended to clusters and clusterings
- For an individual point, i
 - Calculate $a(i)$ = average distance of i to the points in its cluster
 - Calculate $b(i)$ = min (average distance of i to points in another cluster)
 - min is taken over all other clusters than i 's cluster
 - The silhouette coefficient for a point is then given by
- $$s(i) = \frac{b(i) - a(i)}{\max\{a(i), b(i)\}}$$
- $$s(i) = \begin{cases} 1 - a(i)/b(i), & \text{if } a(i) < b(i) \\ 0, & \text{if } a(i) = b(i) \\ b(i)/a(i) - 1, & \text{if } a(i) > b(i) \end{cases}$$
- Typically between -1 and 1
 - The closer to 1 the better
- Can calculate the average silhouette coefficient for a cluster or a clustering



Example: Silhouette Coefficient

- Remark: we use **Manhattan Distance** for distance computations
- 3 clusters with their points are depicted below:

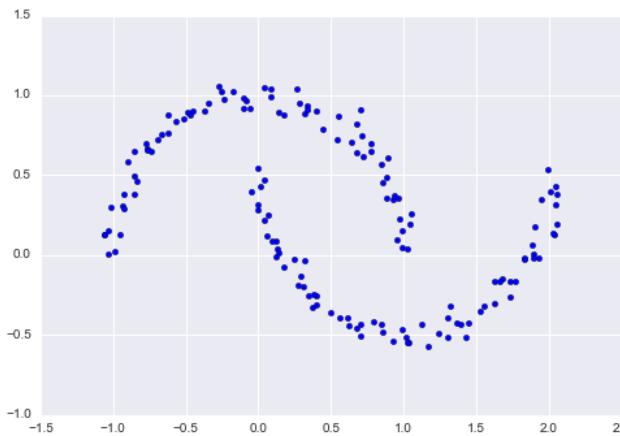


$$d_{\text{Manhattan}}(x, y) = \sum_{i=1}^d |x_i - y_i|$$

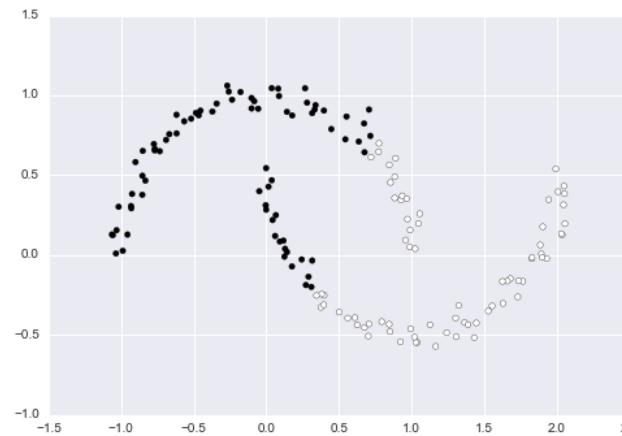
- **Task:** Compute Silhouette for point (3,5)
 - Average Distance of (3,5) to the other points in C3: $(3+2)/2=2.5$
 - Average Distance of (3,5) to points in cluster C1: $(6+8)/2=7$
 - Average Distance of (3,5) to points in cluster C2: $(4+5)/2=4.5$
 - $s = (b - a) / \max(a, b)$
 - $\text{Silhouette}((3,5))=(4.5-2.5)/4.5=0.44$ “okay but not that great”

Internal Measures

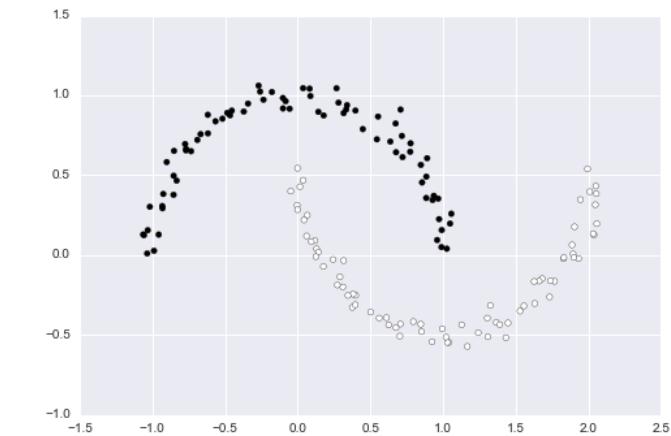
- Silhouette Coefficient may not work for non-convex clusters
- Density-Based Clustering Validation (DMCV)
 - DBCV works for non-convex clustering
 - D. Moulavi, et al. “Density-based clustering validation.” In SDM, 2014.



Raw data



K-means
(DMCV=-0.71)



DBSCAN
(DMCV=0.6)

External Measures: Purity and Entropy

- External measures use cluster/class labels of the test set for evaluating the quality of clusterings
- Suppose a clustering $\{C_1, C_2, \dots, C_K\}$
- From the class labels of data points, we can derive a set of true classes $\{Y_1, Y_2, \dots, Y_M\}$, where K and M may be different
- **Can we use Accuracy, Precision, Recall, and F1-score?**

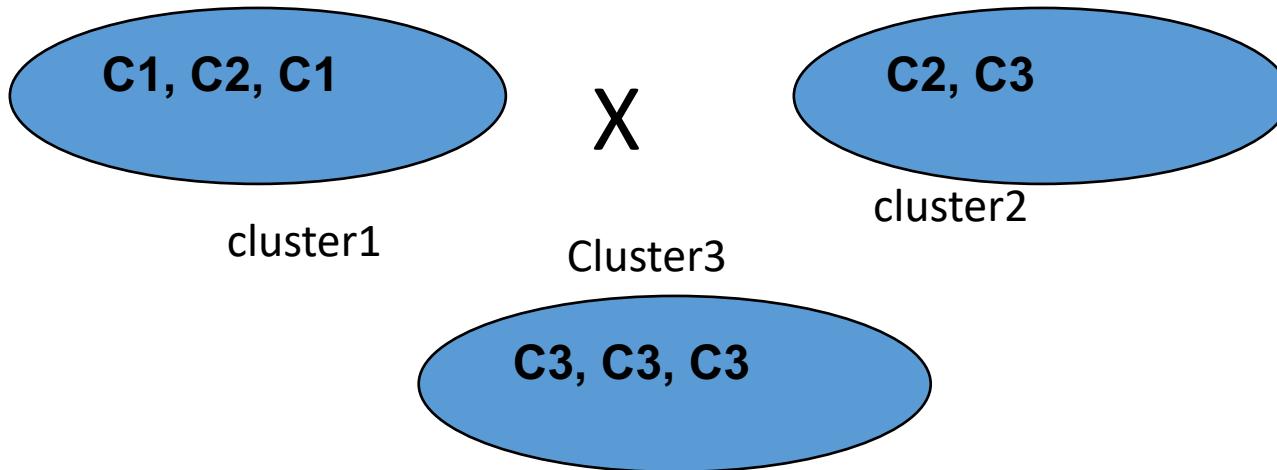
External Measures: Purity and Entropy

- External measures use cluster/class labels of the test set for evaluating the quality of clusterings
- Suppose a clustering $\{C_1, C_2, \dots, C_K\}$
- From the class labels of data points, we can derive a set of true classes $\{Y_1, Y_2, \dots, Y_M\}$, where K and M may be different
- Purity = $\frac{1}{N} \sum_{i=1}^K \max_j |C_i \cap Y_j|$, N is the total number of data points
 - Majority class examples / total number of examples
 - Large purity indicates good clustering
- Entropy = $\sum_{i=1}^K \frac{N_i}{N} \left(- \sum_{j=1}^M \frac{|C_i \cap Y_j|}{|C_i|} \log_2 \left(\frac{|C_i \cap Y_j|}{|C_i|} \right) \right)$
 - Small entropy indicates good clustering

The same as Entropy for a split in a Decision Tree

Example: Purity and Entropy

- Assume the ground truth consists of 3 Classes C1, C2, and C3



- Purity(X)=(2+1+3)/8=0.75
- Entropy(X)= $3/8*H(2/3,1/3,0)+2/8*H(0,1/2,1/2)+3/8*H(0,0,1)$
 $=3/8*[-(2/3)\log_2(2/3)-(1/3)\log_2(1/3)]$
 $+2/8*[-(1/2)\log_2(1/2)-(1/2)\log_2(1/2)] + 3/8*[-1\log_2(1)]$
 $=0.594$

Cluster Measures in Sklearn

Internal
Measures

```
sklearn.metrics.silhouette_score(X, labels, *, metric='euclidean',  
sample_size=None, random_state=None, **kwds)
```

https://scikit-learn.org/1.5/modules/generated/sklearn.metrics.silhouette_score.html

```
sklearn.metrics.davies_bouldin_score(X, labels)
```

https://scikit-learn.org/1.5/modules/generated/sklearn.metrics.davies_bouldin_score.html

```
sklearn.metrics.calinski_harabasz_score(X, labels)
```

https://scikit-learn.org/1.5/modules/generated/sklearn.metrics.calinski_harabasz_score.html#sklearn.metrics.calinski_harabasz_score

```
sklearn.metrics.homogeneity_score(labels_true, labels_pred)
```

https://scikit-learn.org/1.5/modules/generated/sklearn.metrics.homogeneity_score.html#sklearn.metrics.homogeneity_score

```
sklearn.metrics.normalized_mutual_info_score(labels_true, labels_pred, *, average_method='arithmetic')
```

https://scikit-learn.org/stable/modules/generated/sklearn.metrics.normalized_mutual_info_score.html

```
sklearn.metrics.v_measure_score(labels_true, labels_pred, *, beta=1.0)
```

https://scikit-learn.org/stable/modules/generated/sklearn.metrics.v_measure_score.html#sklearn.metrics.v_measure_score

```
sklearn.metrics.adjusted_rand_score(labels_true, labels_pred)
```

https://scikit-learn.org/1.5/modules/generated/sklearn.metrics.adjusted_rand_score.html

External
Measures



Summary of Cluster Validation

- Measuring clustering by correlation
 - Compute the correlation between proximity matrix and cluster co-occurrence matrix → large (absolute) correlation indicates better clustering
- Statistic framework for clustering
 - Compare certain measure of a clustering with the clusterings on random data
- Internal index: measure the goodness of a clustering structure without respect to external information
 - E.g., SSE, Cohesion, Separation, Silhouette Coefficient
- External index: measure the extent to which cluster labels match externally supplied class labels
 - E.g., Purity, Entropy, Normalized Mutual Information, Adjusted Rand Index

Anomaly/Outlier Detection

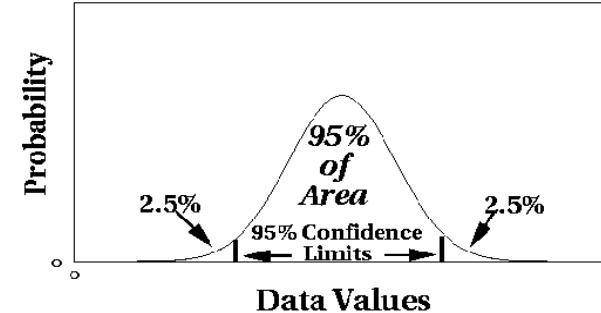
- What are anomalies/outliers?
 - The set of data points that are considerably different than the remainder of the data
- Variants of Anomaly/Outlier Detection Problems
 - Given a dataset D, find all the data points $x \in D$ with anomaly scores greater than some threshold t
 - Given a dataset D, find all the data points $x \in D$ having the top-n largest anomaly scores $f(x)$
 - Given a dataset D, containing mostly normal (but unlabeled) data points, and a test point x , compute the anomaly score of x with respect to D
- Applications:
 - Credit card fraud detection, telecommunication fraud detection, network intrusion detection, system fault detection, data cleaning, sensor fusion

Anomaly/Outlier Detection

- Historically, the field of statistics tried to find and remove outliers as a way to improve analyses
- There are now many fields where anomalies are the objects of great interests
 - The rare events may be the ones with the greatest impact

Causes of Anomalies

- Data from different classes
 - Measuring the weights of oranges, but a few grapefruit are mixed in
 - Fraud activities that pretend to be safe
- Natural variation
 - Tails of a Gaussian distribution
 - Unusually Tall people
- Data errors (during acquisition, transmission, recording, etc.)
 - 200 pound 2 year old



Anomaly Detection

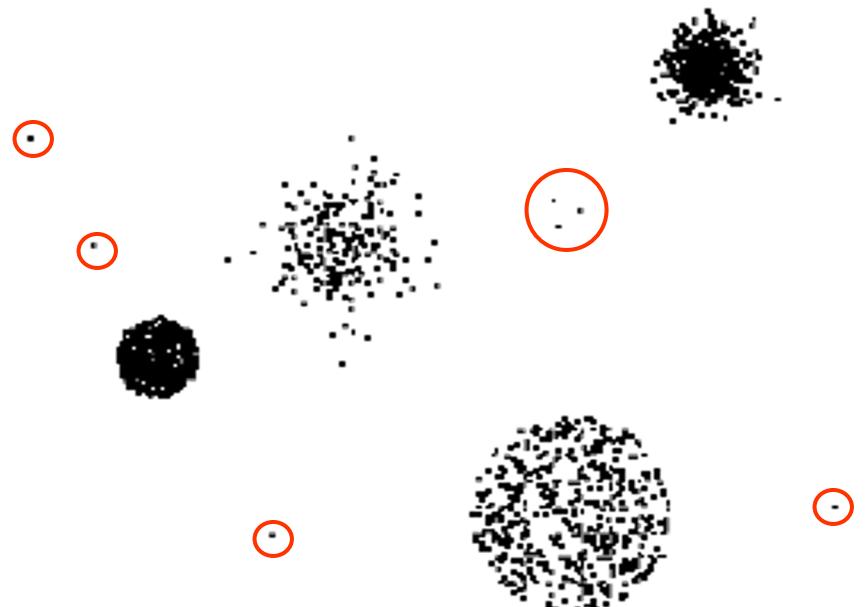
- Assumption:
 - There are considerably more “normal” observations than “abnormal” observations (outliers/anomalies) in the dataset
- Challenges
 - How many anomalies are there in the data?
 - Usually, method is unsupervised
 - Validation can be quite challenging (just like for clustering)
 - Labels of anomalies are difficult to obtain because anomalies are rare patterns
 - Finding needle in a haystack

Object vs Attribute Anomalies

- Some anomalies are defined in terms of a single attribute
 - Height
 - Shape
 - Color
- Object anomalies are harder to identify as objects are usually described by multiple attributes
- Can be hard to find an anomaly using all attributes
 - Noisy or irrelevant attributes
 - Object is only anomalous with respect to some attributes
- An object may not be anomalous in any individual attribute

Anomaly Detection Schemes

- General Steps
 - Build a profile of the “normal” behavior
 - Profile can be patterns or summary statistics for the overall population
 - Use the “normal” profile to detect anomalies
 - Anomalies are observations whose characteristics differ significantly from the normal profile
- Types of anomaly detection schemes
 - Graphical
 - Statistical
 - Proximity-based
 - Clustering-based
 - Density-based
 - Reconstruction-based





COSC 3337
Data Science I
Section 14623

Anomaly Detection

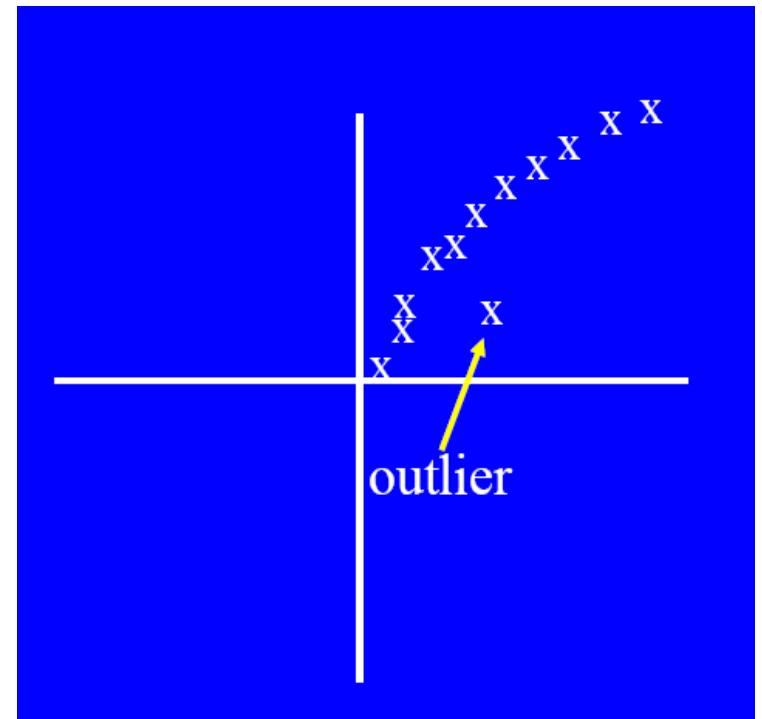
Instructor: Jingchao Ni
Fall 2024

Model-based vs Model-free

- Model-based Approaches
 - Model can be parametric or non-parametric
 - Anomalies are those points that don't fit well
 - Anomalies are those points that distort the model
- Model-free Approaches
 - Anomalies are identified directly from the data without building a model
- Often the underlying assumption is that the most of the points in the data are normal

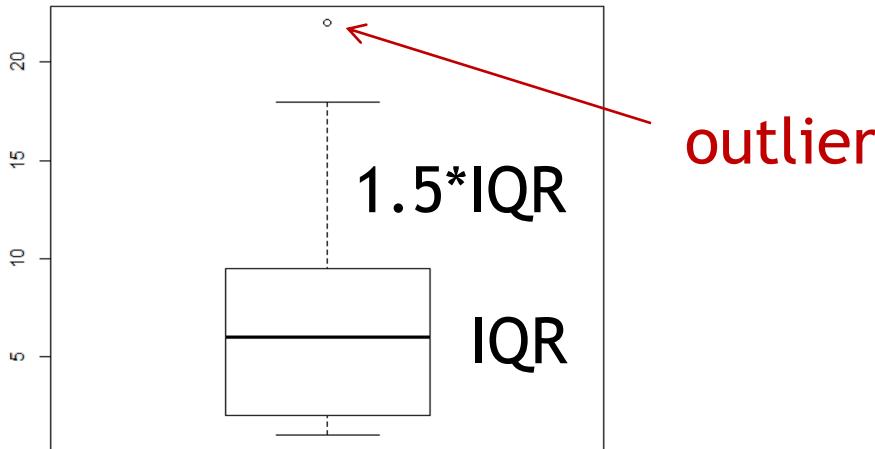
Graphical Approaches

- Idea: user identifies outliers by visual inspection
- Scatter plot (2-D, 3-D)
- Limitations
 - Time consuming
 - Subjective
 - Data with higher dimensions



Box-Plot Approach for Outlier Detection

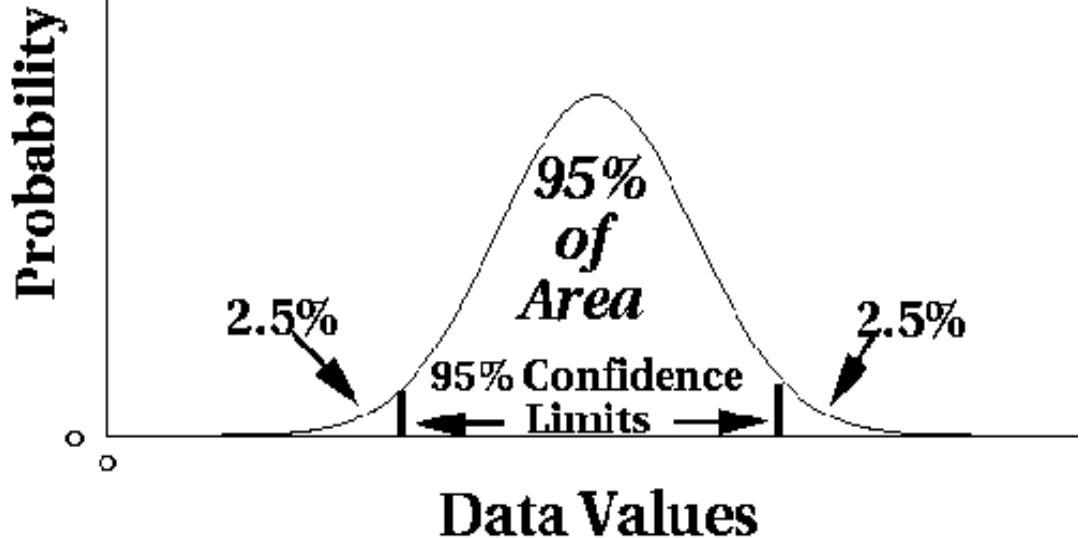
- Mixture of a graphical and a statistical approach
- Observations that are more than $\lambda * \text{IQR}$ (e.g. $\lambda=1.5$) above or below the inter-quantile range are outliers
- Decent approach for 1D/single attribute outlier detection
- Cannot be used for multi-variate data



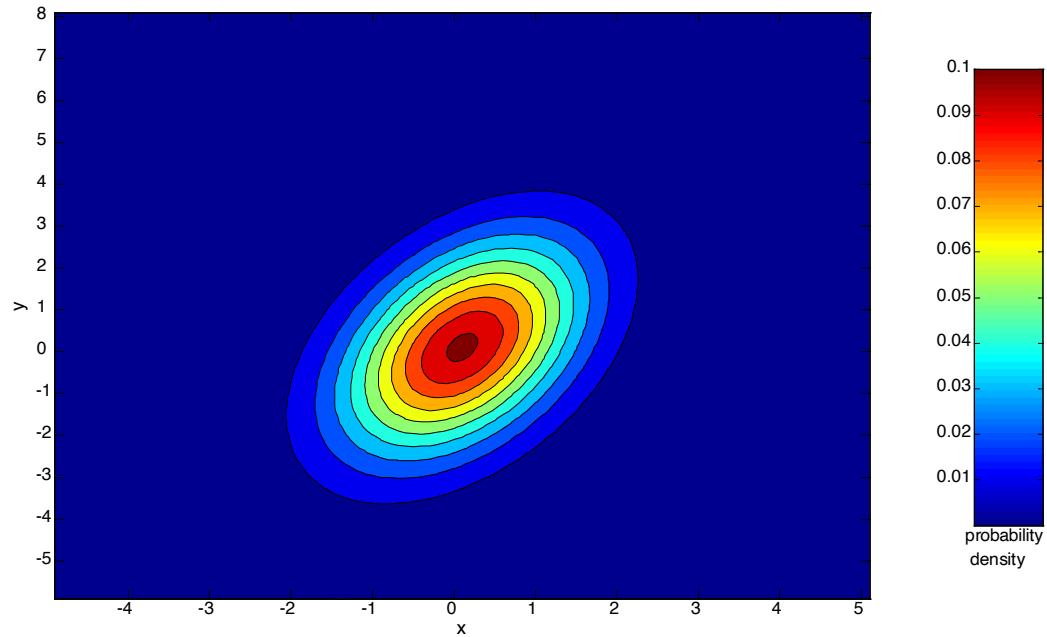
Statistical Approaches

- Probabilistic definition of anomalies
 - An anomaly is an object that has a low probability with respect to a probability distribution model of data
- Usually assume (or estimate) a parametric model describing the distribution of the data (e.g., normal distribution)
- Apply a statistical test that depends on
 - Data distribution
 - Parameters of distribution (e.g., mean, variance)
 - Number of expected outliers (confidence limit)

Normal Distributions



One-dimensional Gaussian



Two-dimensional Gaussian

Likelihood Approach

- Assume the data set D contains samples from a mixture of two probability distributions:
 - M (majority distribution)
 - A (anomalous distribution)
- General Approach:
 - Initially, assume all the data points belong to M
 - Let $L_t(D)$ be the log likelihood of D at time t
 - For each point x_t that belongs to M , move it to A
 - Let $L_{t+1}(D)$ be the new log likelihood
 - Compute the difference, $\Delta = L_{t+1}(D) - L_t(D)$
 - If $\Delta > c$ (some threshold), then x_t is declared as an anomaly and moved permanently from M to A

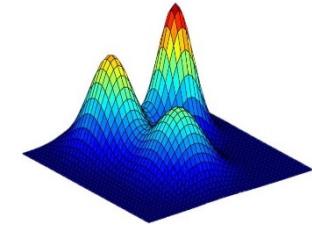
Likelihood Approach

- Data Distribution: $D = \begin{cases} (1 - \lambda)M & x \in M \\ \lambda A & x \in A \end{cases}$
- M is a probability distribution estimated from data
 - Can be based on any modeling method (naïve Bayes, maximum entropy, etc.)
- A is initially assumed to be uniform distribution
- Likelihood at time t:

$$L_t(D) = \prod_{i=1}^N P_D(x_i) = \left((1 - \lambda)^{|M_t|} \prod_{x_i \in M_t} P_{M_t}(x_i) \right) \left(\lambda^{|A_t|} \prod_{x_i \in A_t} P_{A_t}(x_i) \right)$$

$$LL_t(D) = |M_t| \log(1 - \lambda) + \sum_{x_i \in M_t} \log P_{M_t}(x_i) + |A_t| \log \lambda + \sum_{x_i \in A_t} \log P_{A_t}(x_i)$$

Likelihood Approach: GMM



- Fit a model M to the dataset D ; e.g.
 - A multivariate Gaussian Mixture Model Mixture model by running the EM clustering algorithm
- Plug each point x into the density function d_M of model M and compute $d_M(x)$ or preferably $\log(d_M(x))$, called the **log likelihood** of x , and add this value as in a new column ols (“outlier score”) to D , obtaining D' : the smaller this value is the more likely x is an outlier with respect M
- Sort D' in ascending order, the first record is the record with the smallest value for $\log(d_M(x))$
- Remove the top k records from D'

Summary of Statistical Approaches

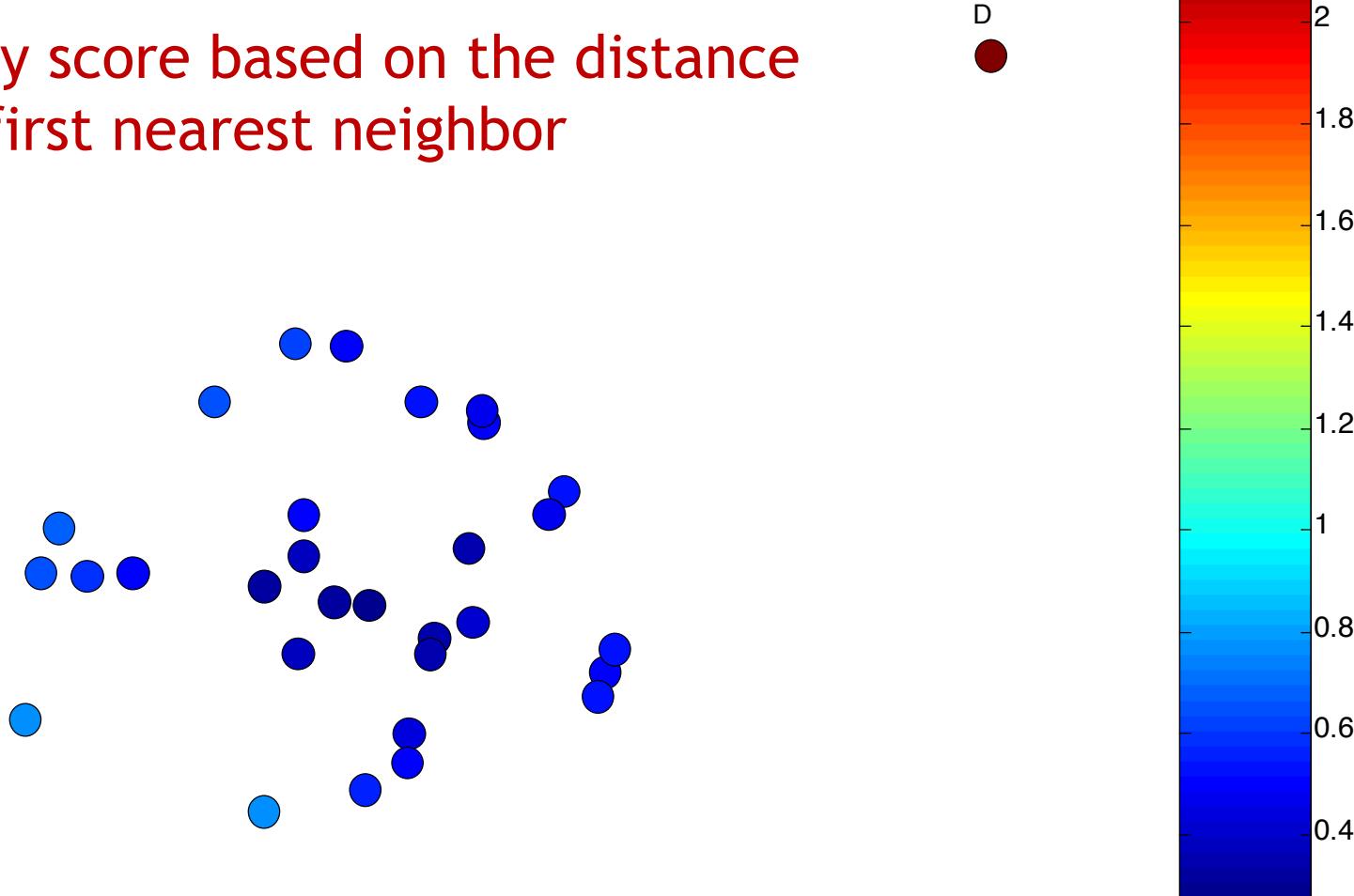
- Strengths
 - Solid mathematical foundation
 - Can be very efficient
 - Good results if distribution is known
- Weaknesses
 - In many cases, data distribution may not be known
 - For high dimensional data, data may be insufficient to estimate the true distribution
 - Anomalies can distort the parameters of the distribution

Proximity-based Approaches

- Intuition:
 - Anomalies are data points far away from other data points
- Approach:
 - Compute the distance between every pair of data points
 - There are various ways to define anomalies:
 - The top n data points whose distance to their respective k th nearest neighbor is greatest
 - The anomaly score of a data point is the distance to its k -th nearest neighbor
 - Data points for which there are fewer than p neighboring points within a distance r

Proximity-based Approaches

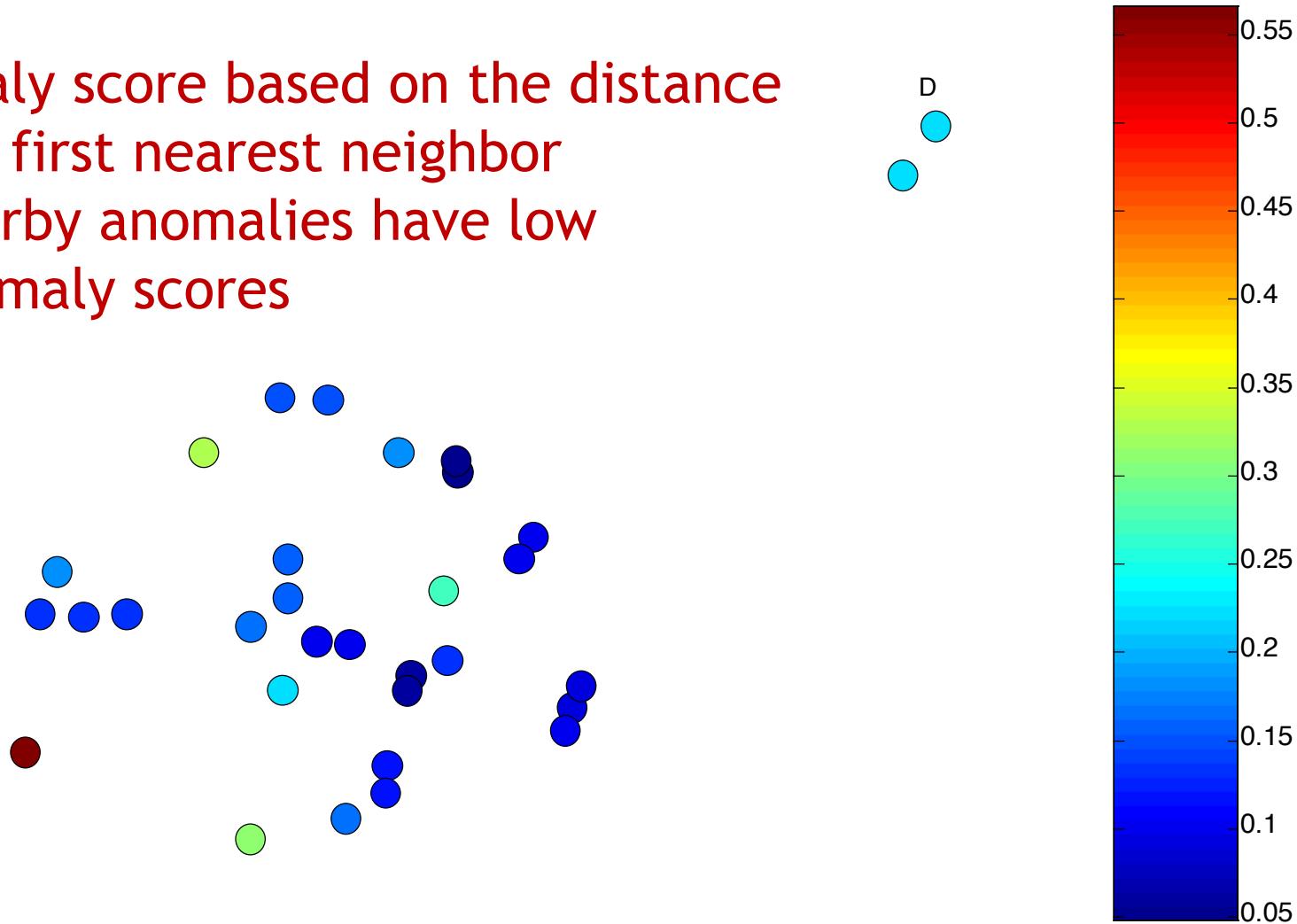
Anomaly score based on the distance
to the first nearest neighbor



Proximity-based Approaches

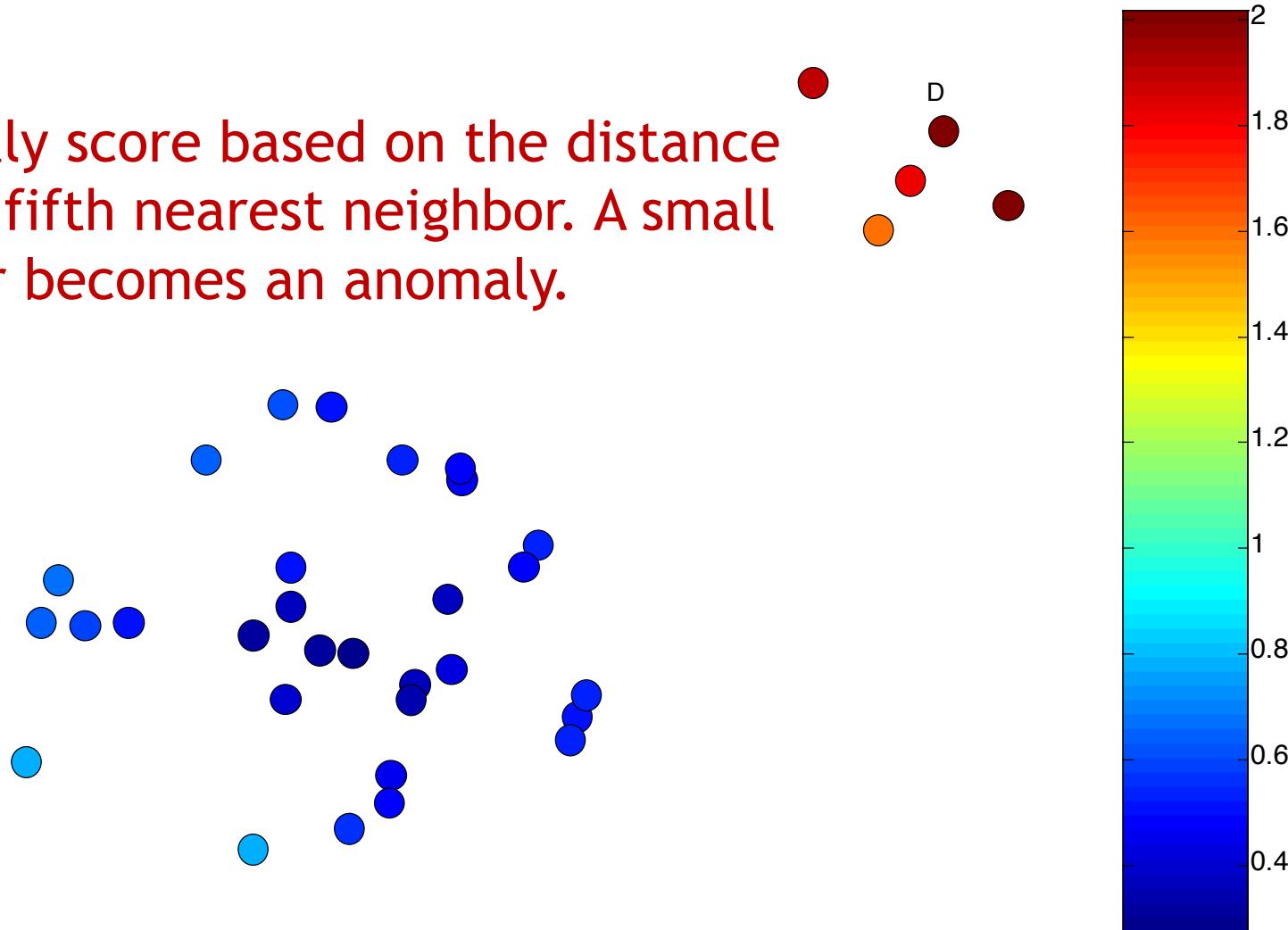
Anomaly score based on the distance
to the first nearest neighbor

- Nearby anomalies have low
anomaly scores



Proximity-based Approaches

Anomaly score based on the distance to the fifth nearest neighbor. A small cluster becomes an anomaly.



Proximity-based Approaches

Anomaly score based on the distance
to the fifth nearest neighbor.
Clusters of differing density.



Summary of Proximity-based Approaches

- Strengths
 - Easier to define a proximity measure for a dataset than determine its statistical distribution
 - Quantitative measure of degree to which a data point is an anomaly
 - Deals naturally with multiple modes (or clusters)
- Weaknesses
 - Expensive computation, $O(n^2)$ complexity
 - Score sensitive to choice of k
 - Does not work well if data has widely variable density

Density-based Approaches

- Intuition:
 - Anomalies are data points in regions of low density
- Anomaly score is inverse of density around the data point
- Scores are usually based on proximities
- Example Scores:
 - Reciprocal of average distance to k nearest neighbors

$$\text{density}(\mathbf{x}, k) = \left(\frac{1}{k} \sum_{\mathbf{y} \in N(\mathbf{x}, k)} \text{distance}(\mathbf{x}, \mathbf{y}) \right)^{-1}$$

- Number of data points within a fixed radius (DBSCAN)
- If there are regions of different density, this approach may have problems

Density-based Approaches

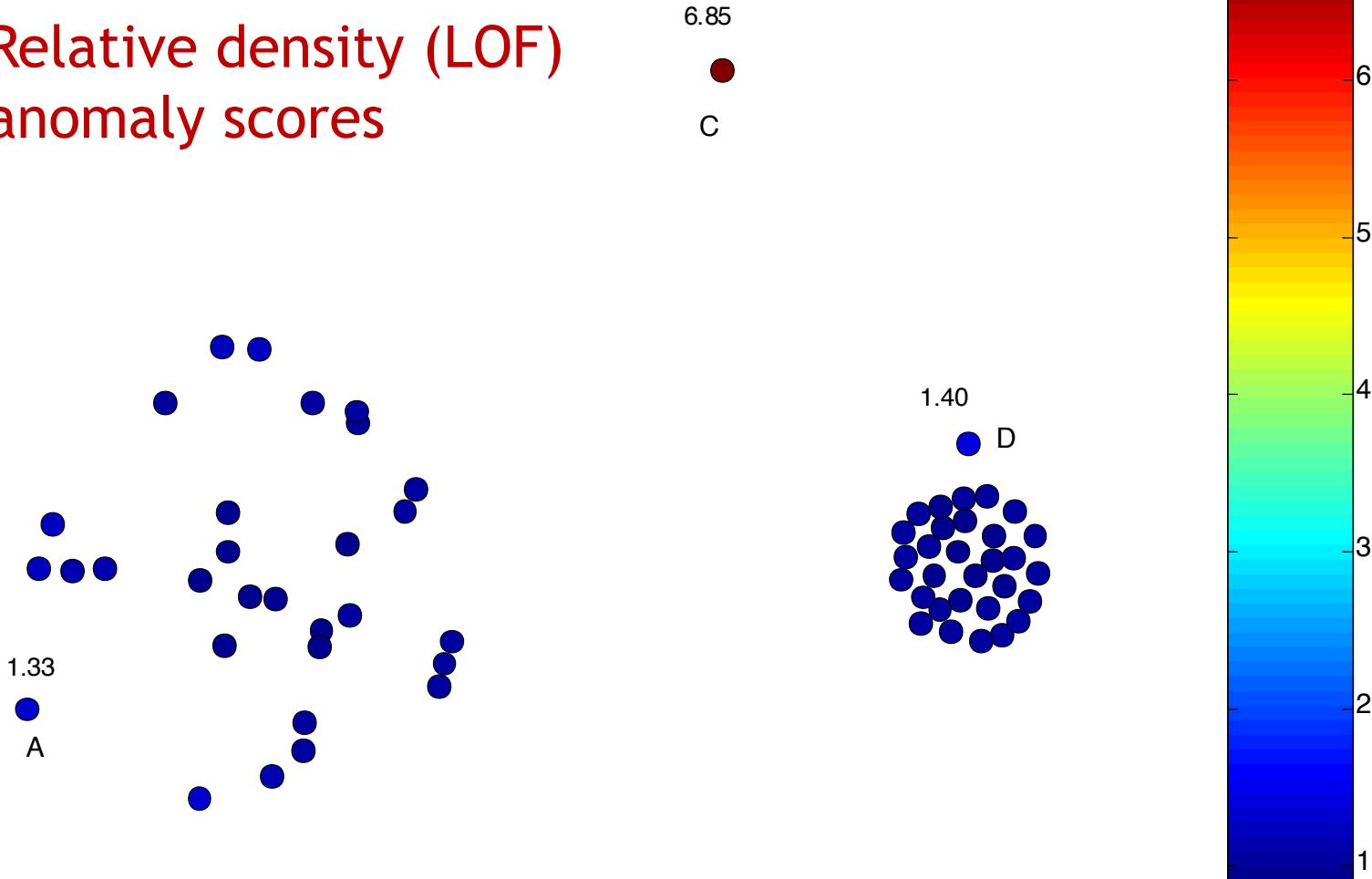
- Relative density outlier score (Local Outlier Factor, LOF)
 - Reciprocal of average distance to k nearest neighbors, relative to that of those k neighbors

$$\text{relative density}(\mathbf{x}, k) = \frac{\text{density}(\mathbf{x}, k)}{\frac{1}{k} \sum_{\mathbf{y} \in N(\mathbf{x}, k)} \text{density}(\mathbf{y}, k)}$$

A data point \mathbf{x} is anomalous if its local region has small density but its neighbors' local regions have high density

Density-based Approaches

Relative density (LOF)
anomaly scores



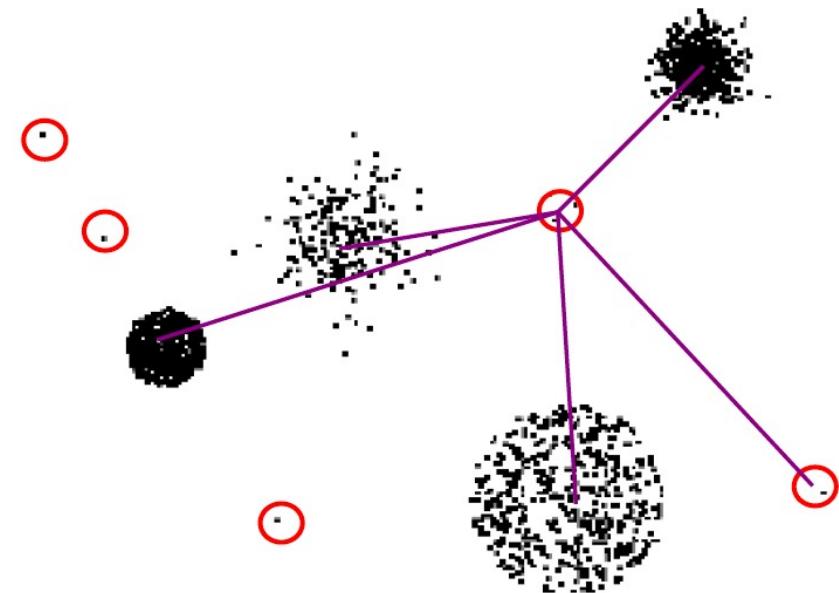
Summary of Density-based Approaches

- Strengths
 - Easier to define a density measure for a dataset than determine its statistical distribution
 - Quantitative measure of degree to which a data point is an anomaly
- Weaknesses
 - Expensive computation, $O(n^2)$ complexity
 - Must choose parameters:
 - K for nearest neighbor
 - Distance function
 - Density becomes less meaningful in high dimensional space

Cluster-based Approaches

- Intuition:

- Anomalies are data points that do not belong strongly to any cluster
 - For **representative-based clusters**, a data point is an anomaly if it is not close enough to any cluster center
 - For **density-based clusters**, a data point is an anomaly if its density is low
 - For **graph-based clusters**, a data point is an anomaly if it is not well connected to other data points
- Anomalies may affect initial clusters



Cluster-based Approaches

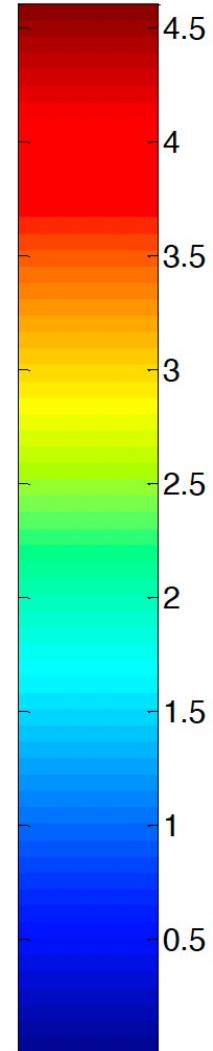
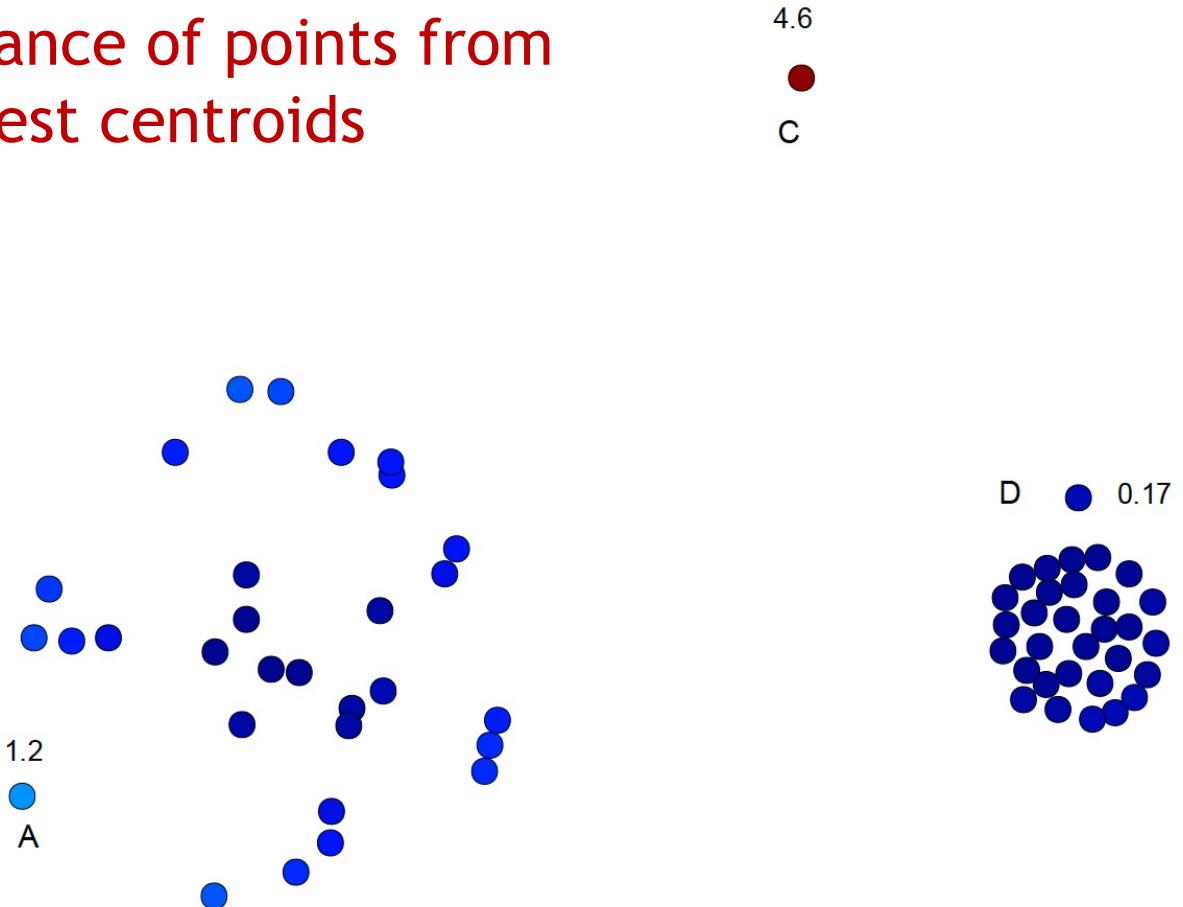
- For representative-based clustering (e.g., k-means), use distance to cluster centers
 - To deal with variable density clusters, use relative distance

$$\frac{\text{distance}(\mathbf{x}, \text{centroid}_c)}{\text{median}(\{\forall_{x' \in C} \text{distance}(\mathbf{x}', \text{centroid}_c)\})}$$

- Similar concepts for density-based or graph-based clusters

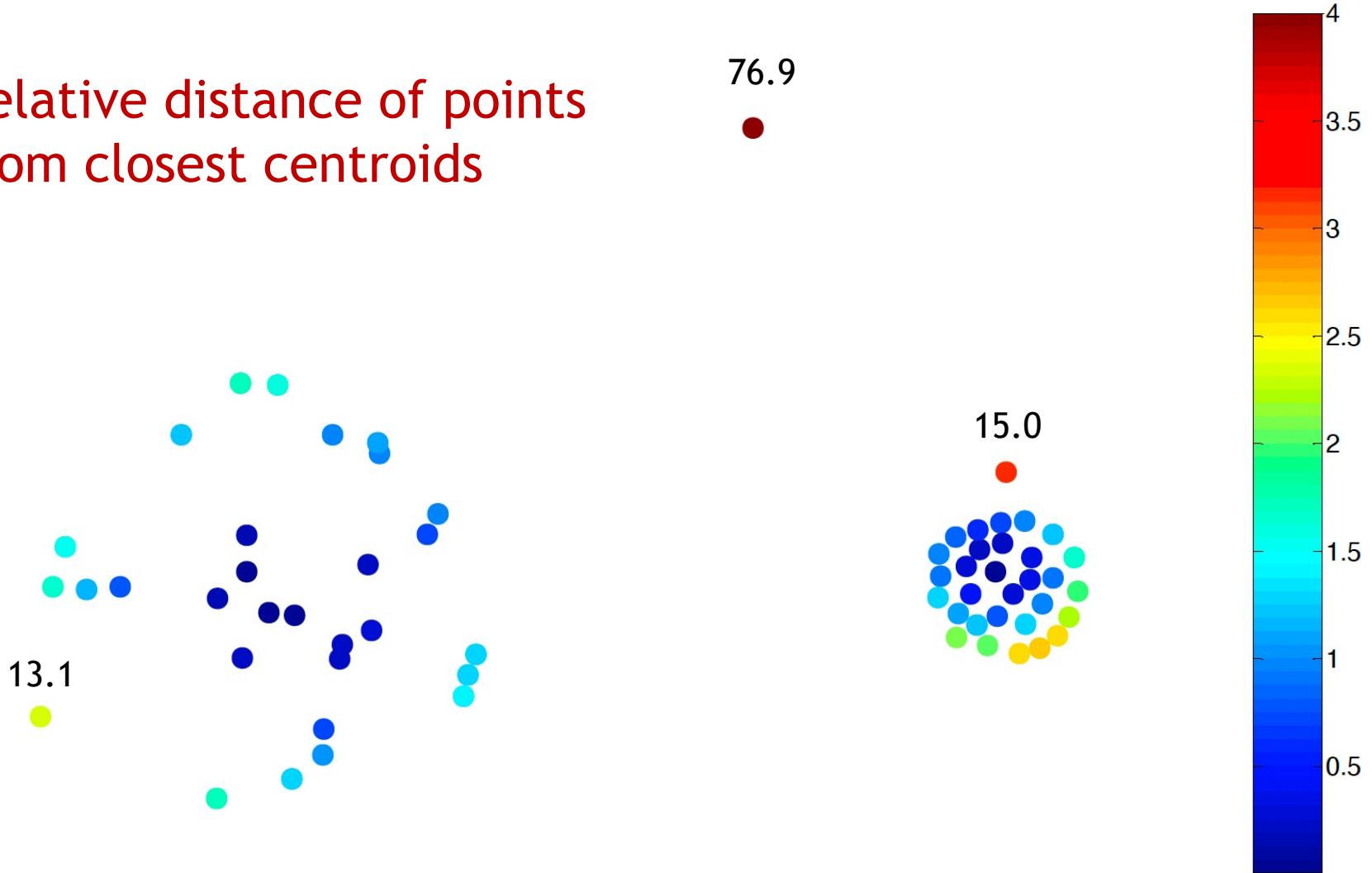
Cluster-based Approaches

Distance of points from
closest centroids



Cluster-based Approaches

Relative distance of points
from closest centroids



Summary of Cluster-based Approaches

- Strengths
 - Many clustering algorithms can be used
 - Some clustering algorithm have $O(n)$ complexity
 - Easier to define an anomaly for a dataset than determine its statistical distribution
- Weaknesses
 - Requires a proper choice of clustering algorithms
 - Requires parameters for clustering algorithms
 - E.g., number of clusters k
 - Anomalies can distort the initial formation of clusters

Reconstruction-based Approaches

- Intuition:
 - Based on assumptions there are patterns in the distribution of the normal class that can be captured using lower-dimensional representations
- Approach:
 - Reduce data to lower dimensional data
 - E.g. Use Principal Components Analysis (PCA), matrix factorization, or Auto-encoders
 - Measure the reconstruction error for each data point
 - The difference between the original and the reconstructed version using the low dimensional representations

Reconstruction-based Approaches

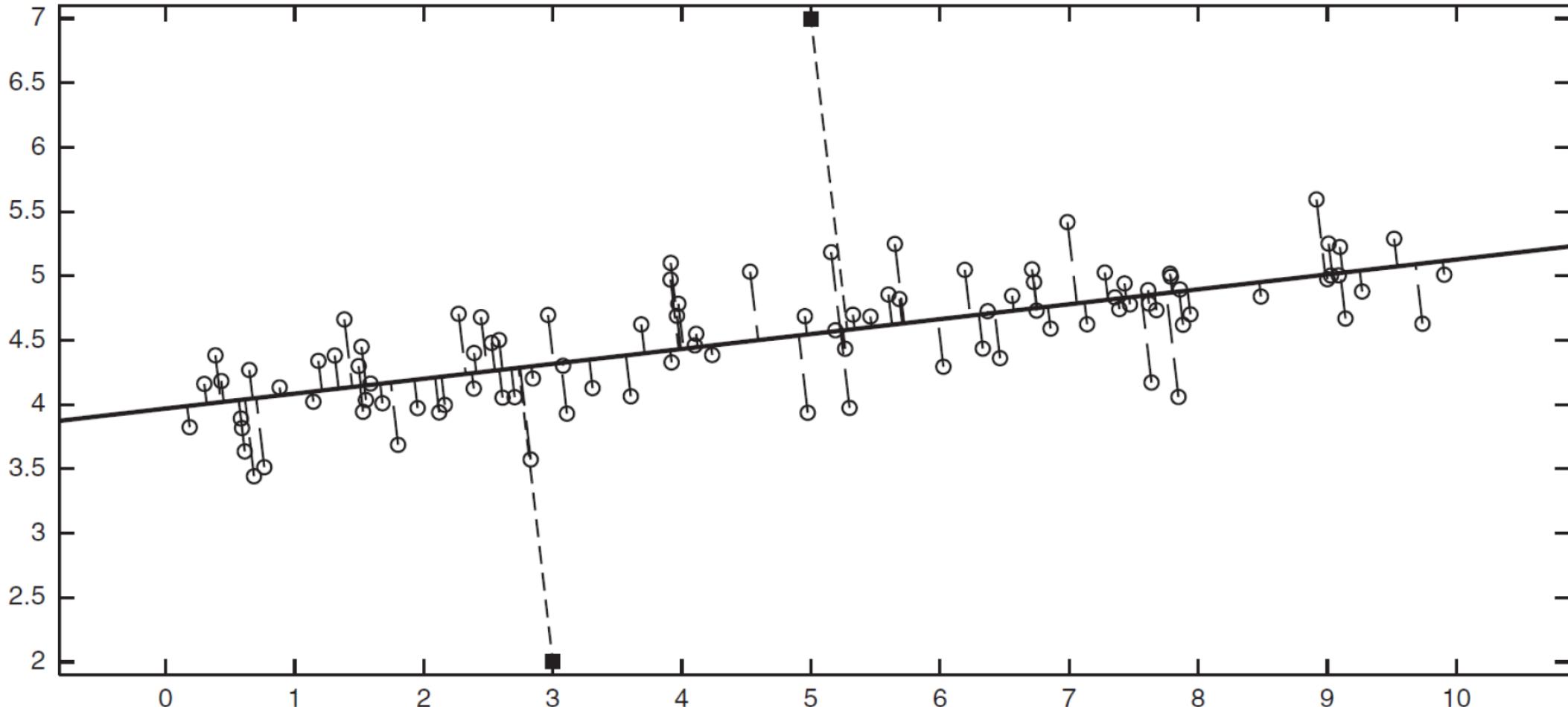
- Reconstruction Error
 - Let x be the original data point
 - Find the representation of the data point in a lower dimensional space
 - Project the object back to the original space
 - Call this object \hat{x}

$$\text{Reconstruction Error}(x) = \|x - \hat{x}\|$$

- Data points with large reconstruction errors are anomalies

Reconstruction-based Approaches

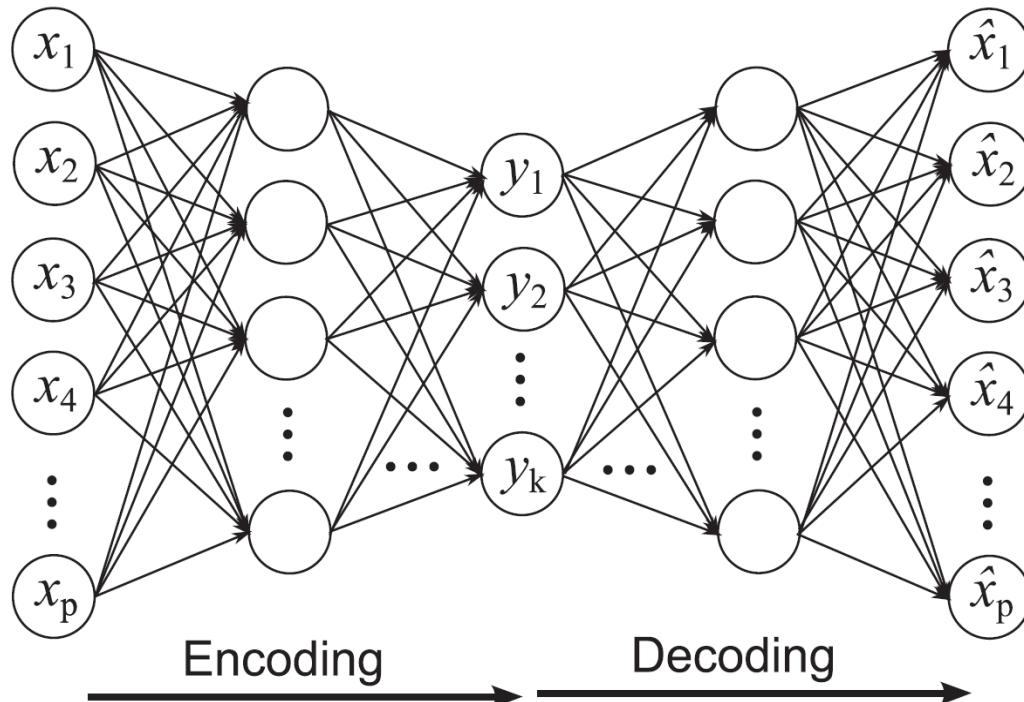
- Reconstruction of two-dimensional data



Reconstruction-based Approaches

- Basic Architecture of an Autoencoder

- An autoencoder is a multi-layer neural network
- The number of input and output neurons is equal to the number of original attributes



Summary of Reconstruction Approaches

- Strengths
 - It does not require assumptions about distribution of normal class
 - Many dimensionality reduction approaches can be used
- Weaknesses
 - The reconstruction error is computed in the original space
 - This can be a problem if dimensionality is high

One-Class SVM for Anomaly Detection

- Data is unlabeled, unlike the usual SVM setting
- Goal: find hyperplane (in high-dimensional kernel space) which encloses as much data as possible with minimum volume
 - Tradeoff amount of data enclosed and tightness of enclosure; controlled by regularization of slack variables

One-Class SVM for Anomaly Detection

- Consider a sphere with center a and radius R
- Minimize R and the error resulting from points outside the sphere: their error is their distance to the sphere

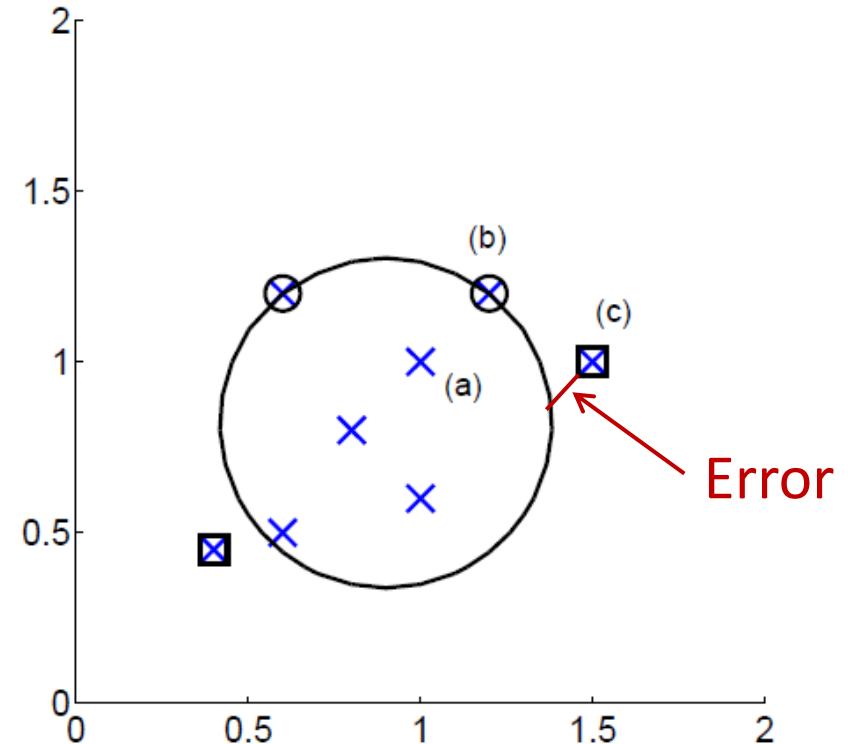
$$\min R^2 + C \sum_t \xi^t$$

Slack variable

subject to

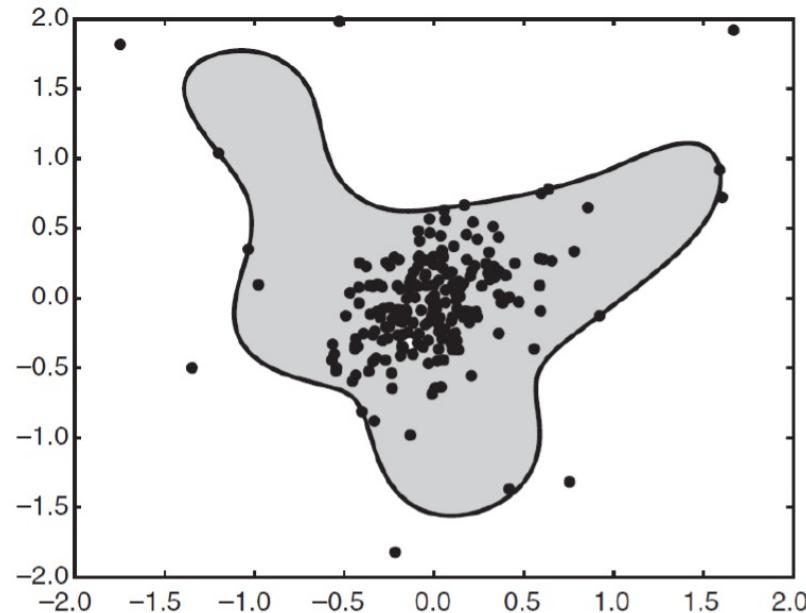
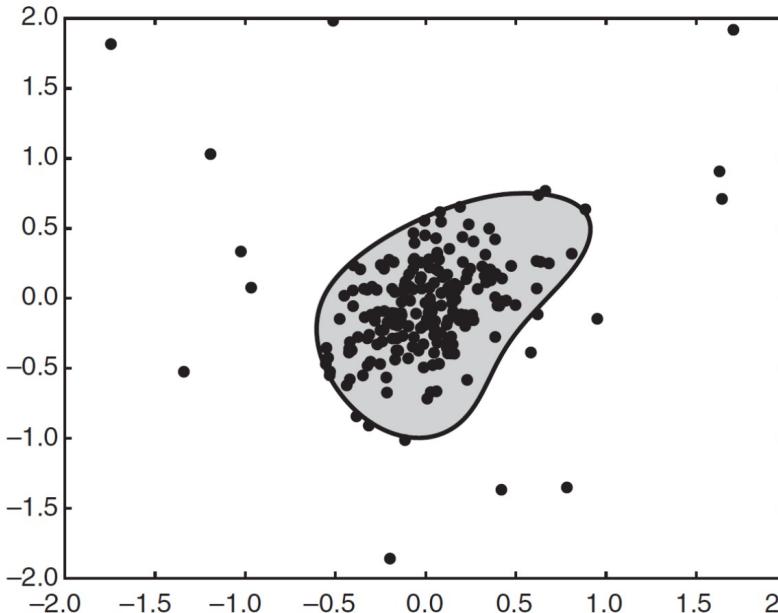
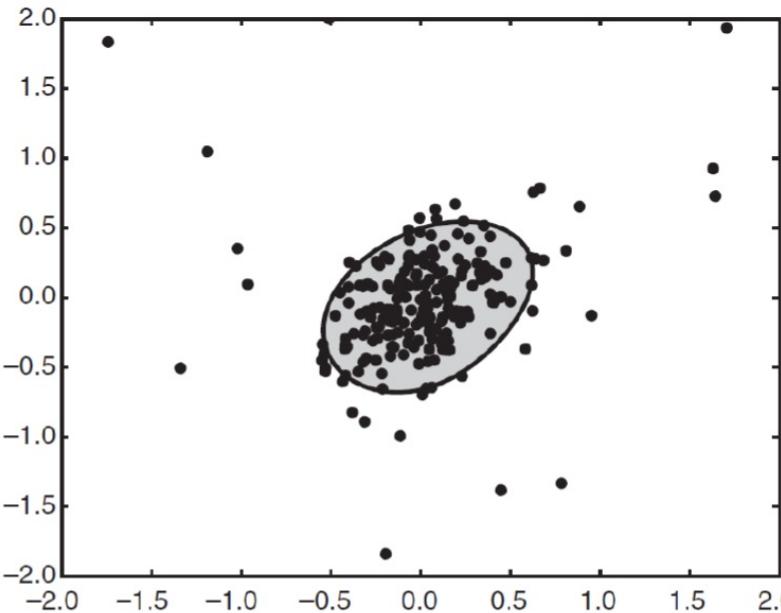
Feature Map

$$\|\mathbf{x}^t - a\| \leq R^2 + \xi^t, \xi^t \geq 0$$
$$\|\phi(\mathbf{x}^t - a)\|$$



One-Class SVM for Anomaly Detection

- Example
 - Similar to SVM, tune σ and C for different boundaries



Summary of One-Class SVM

- Strengths
 - It has a strong theoretical foundation
- Weaknesses
 - Tuning parameters may be difficult
 - The algorithm is computationally expensive

Evaluation of Anomaly Detection

- If labels of anomalies are available for evaluation, then use standard evaluation approaches for rare class such as precision, recall, F1-score, or false positive rate
 - FPR is also known as false alarm rate
 - In particular, precision @ top-K, recall @ top-K
- For evaluation without using labels, use measures provided by the anomaly method
 - E.g., likelihood, reconstruction error, etc.

Summary of Anomaly Detection

- Anomalies: The set of data points that are considerably different than the remainder of the data
- Assumption of Anomaly Detection: There are considerably more “normal” data than “abnormal” data in the dataset
- General Steps: (1) Build a profile of the “normal” behavior; (2) Use the “normal” profile to detect anomalies
- Types of Anomaly Detection Methods: (1) Graphical; (2) Statistical; (3) Proximity-based; (4) Clustering-based; (5) Density-based; (6) Reconstruction-based methods

Anomaly Detection Methods in Python

```
class sklearn.mixture.GaussianMixture(n_components=1, *, covariance_type='full',
tol=0.001, reg_covar=1e-06, max_iter=100, n_init=1, init_params='kmeans',
weights_init=None, means_init=None, precisions_init=None, random_state=None,
warm_start=False, verbose=0, verbose_interval=10)
```

<https://scikit-learn.org/1.5/modules/generated/sklearn.mixture.GaussianMixture.html#sklearn.mixture.GaussianMixture>

```
class sklearn.neighbors.LocalOutlierFactor(n_neighbors=20, *, algorithm='auto',
leaf_size=30, metric='minkowski', p=2, metric_params=None, contamination='auto',
novelty=False, n_jobs=None)
```

<https://scikit-learn.org/1.5/modules/generated/sklearn.neighbors.LocalOutlierFactor.html#sklearn.neighbors.LocalOutlierFactor>

```
class sklearn.svm.OneClassSVM(*, kernel='rbf', degree=3, gamma='scale',
coef0=0.0, tol=0.001, nu=0.5, shrinking=True, cache_size=200, verbose=False,
max_iter=-1)
```

<https://scikit-learn.org/stable/modules/generated/sklearn.svm.OneClassSVM.html>



COSC 3337
Data Science I
Section 14623

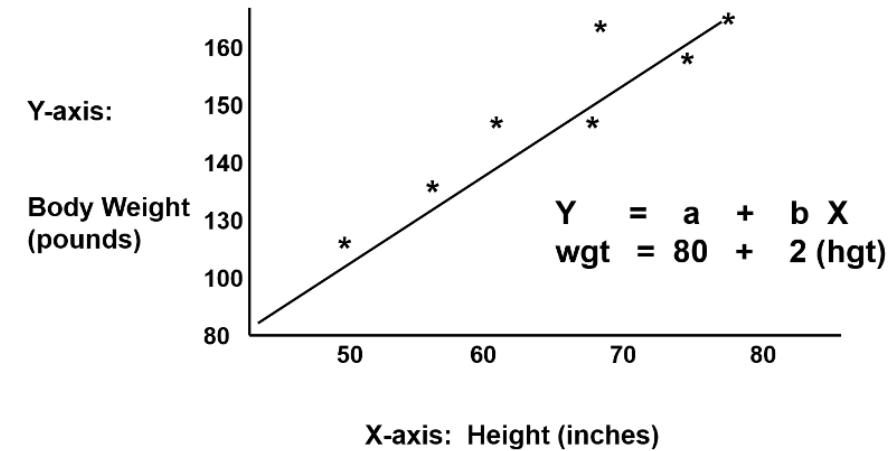
Deep Learning

Instructor: Jingchao Ni
Fall 2024

Linear Regression

- Predict the value of a **response variable y** based on a **linear combination** of the given values of the **predictor variable(s) x**

$$\hat{y} = w_0 + \sum_{i=1}^d w_j x_i$$



- Simple regression: one predictor variable ($d=1$) → regression line
- Multiple regression: multiple predictor variables → regression plane
- Residuals:** differences between observed y and predicted values \hat{y}

$$e = y - \hat{y}$$

- Use the residuals to measure the model fit

Multivariate Linear Regression

- Multiple Predictors (x_1, x_2, \dots, x_d)

$$y = w_0 + w_1x_1 + w_2x_2 + \dots + w_dx_d + \epsilon$$

- Usually, there are $n \geq 1$ data points in a dataset, each data point is represented by (x_i, y_i) , x_i is a vector of d predictors $(x_{i1}, x_{i2}, \dots, x_{id})$

$$y_1 = w_0 + w_1x_{11} + w_2x_{12} + \dots + w_dx_{1d} + \epsilon$$

...

$$y_i = w_0 + w_1x_{i1} + w_2x_{i2} + \dots + w_dx_{id} + \epsilon$$

...

$$y_n = w_0 + w_1x_{n1} + w_2x_{n2} + \dots + w_dx_{nd} + \epsilon$$

Example

- Predict credit score

Tid	Credit History	Marital Status	Taxable Income	Credit
1	7 yr	Single (1)	125K	610
2	20 yr	Married (2)	100K	750
3	5 yr	Single (1)	70K	605
4	15 yr	Married (2)	120K	670
5	7 yr	Divorced (3)	95K	620
6	10 yr	Married (2)	60K	650
7	15 yr	Divorced (3)	220K	790
8	10 yr	Single (1)	85K	670
9	15 yr	Married (2)	75K	690
10	16 yr	Single (1)	90K	730

$$\text{Credit} = w_0 + x_{CH}w_{CH} + x_{MS}w_{MS} + x_{TI}w_{TI}$$

Matrix Representation

- Suppose there are n data points, d+1 dimension

$$\begin{pmatrix} y_1 \\ y_2 \\ \dots \\ y_n \end{pmatrix} = \begin{pmatrix} 1 & x_{11} & \dots & x_{1d} \\ 1 & x_{21} & \dots & x_{2d} \\ \dots & \cdot & \dots & \cdot \\ 1 & x_{n1} & \dots & x_{nd} \end{pmatrix} \times \begin{pmatrix} w_0 \\ w_1 \\ \dots \\ w_d \end{pmatrix} + \varepsilon$$

$n \times 1$

$n \times (d + 1)$

$(d + 1) \times 1$

$$\text{Matrix Representation: } \mathbf{Y} = \mathbf{XW} + \boldsymbol{\varepsilon}$$

Multivariate Linear Regression

- Goal: minimize sum of squared error

$$E = (Y - XW)^T(Y - XW) = Y^T Y - 2X^T W Y + W^T X^T X W$$

- Derivative

$$\frac{\partial E}{\partial W} = -2X^T Y + 2X^T X W = 0$$

- Solution

$$W = (X^T X)^{-1} X^T Y$$

- Therefore, we can solve linear regression using matrix inversion, transpose, and multiplication

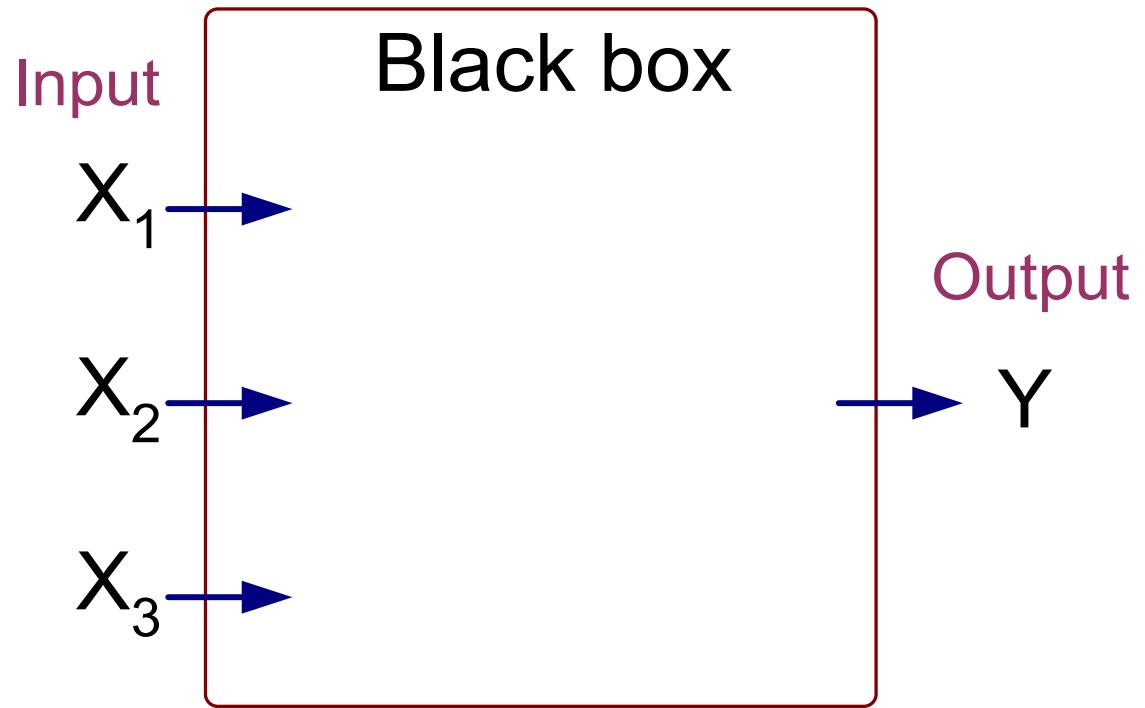
Computational Issues

$$W = (X^T X)^{-1} X^T Y$$

- When there is a lot data points or the dimension is high, matrix inversion is infeasible
- Complexity $O(d^3)$
- $X^T X$ must be invertible (i.e., it is full rank)
 - Problems if linear dependencies between predictor variables
- Solution may be unstable
 - If predictor variables almost linear dependent
- How to handle non-linear data?
 - Neural networks and iterative learning algorithm

Artificial Neural Networks

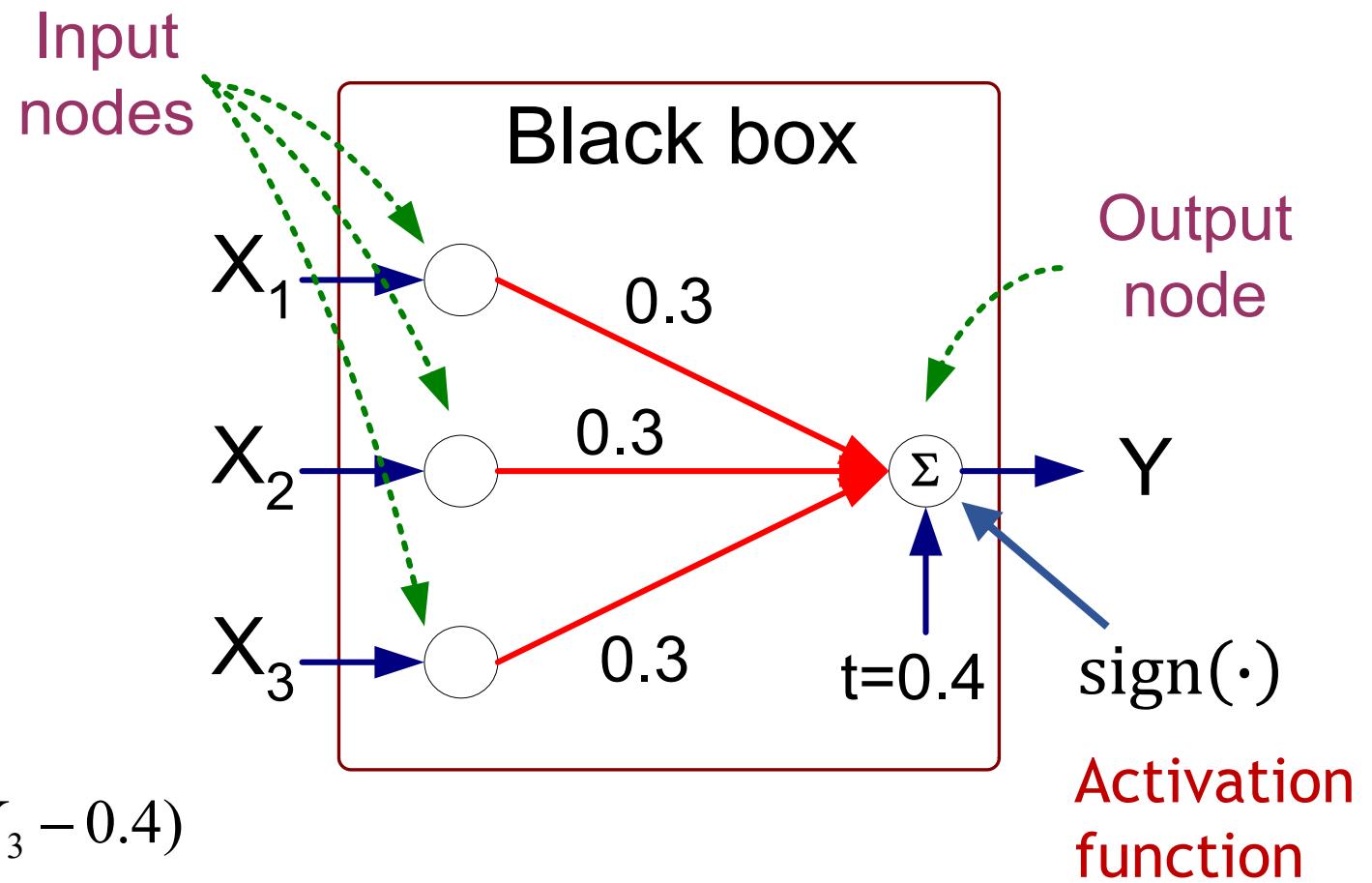
X_1	X_2	X_3	Y
1	0	0	-1
1	0	1	1
1	1	0	1
1	1	1	1
0	0	1	-1
0	1	0	-1
0	1	1	1
0	0	0	-1



Output Y is 1 if at least two of the three inputs are equal to 1.

Artificial Neural Networks

X_1	X_2	X_3	Y
1	0	0	-1
1	0	1	1
1	1	0	1
1	1	1	1
0	0	1	-1
0	1	0	-1
0	1	1	1
0	0	0	-1

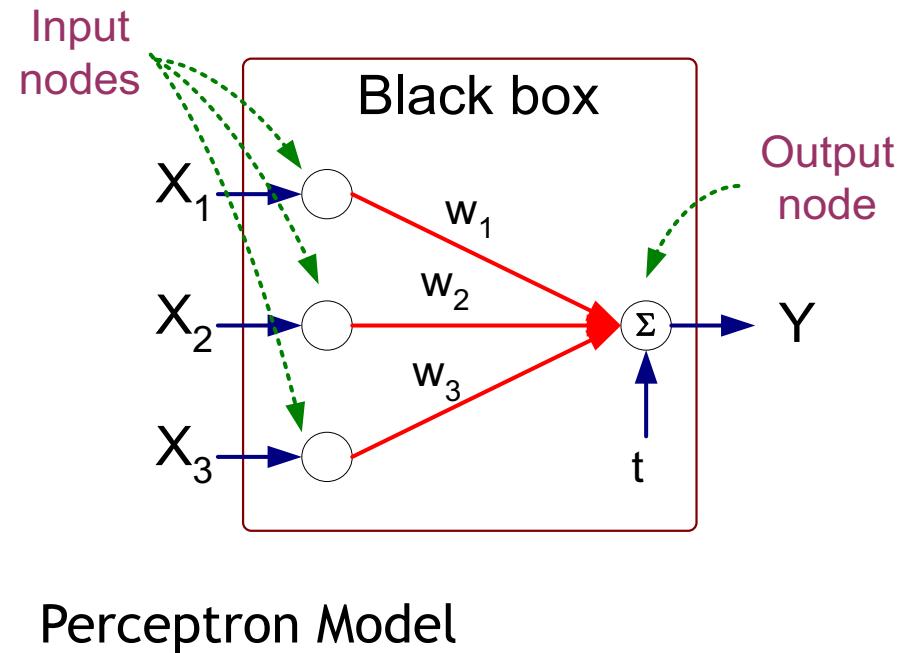


$$Y = \text{sign}(0.3X_1 + 0.3X_2 + 0.3X_3 - 0.4)$$

where $\text{sign}(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ -1 & \text{if } x < 0 \end{cases}$

Artificial Neural Networks

- Model is an assembly of **inter-connected nodes** and **weighted links**
- Output node sums up each of its input value according to the weights of its links
- Compare output node against some threshold t ($-t \rightarrow w_0$)
- Apply activation function on the output



Perceptron Model

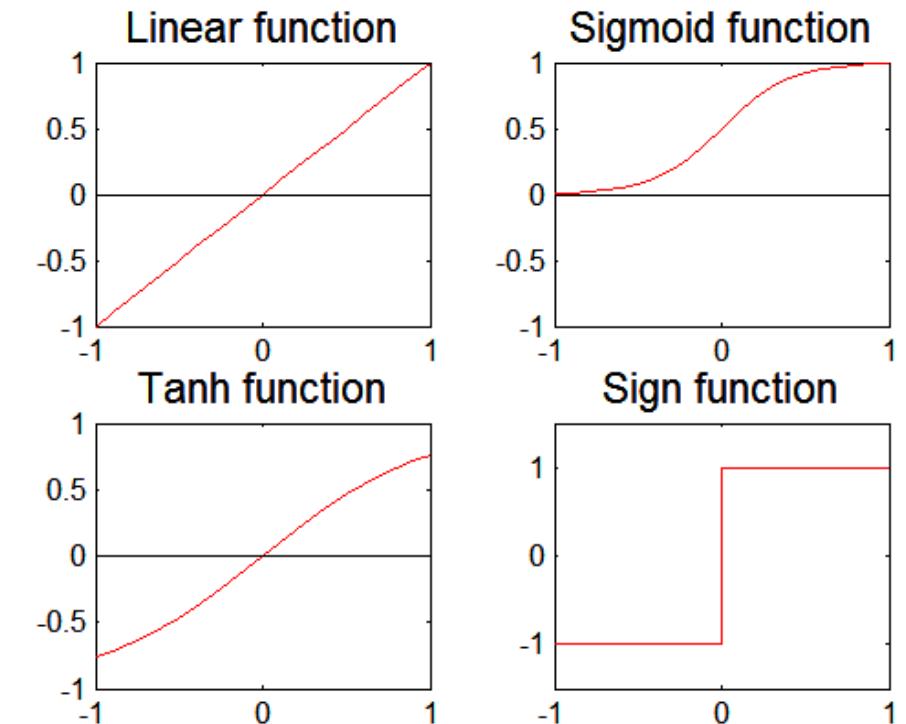
$$Y = \text{sign}(\sum_{i=1}^d w_i X_i - t)$$

$$= \text{sign}(\sum_{i=0}^d w_i X_i)$$

Activation Function

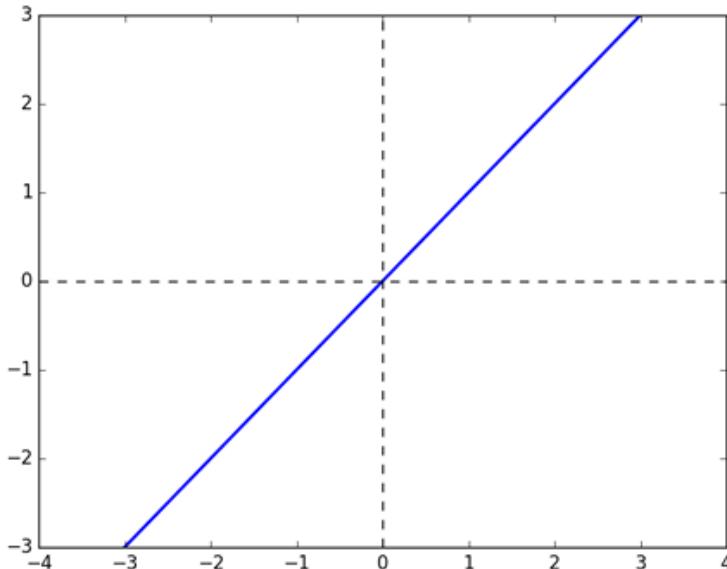
- Activation functions are important for Artificial Neural Network to learn and understand complex patterns
- Its main function is to introduce non-linear properties into the network
- Various types of activation functions

$$Y = f(\sum_i w_i X_i)$$



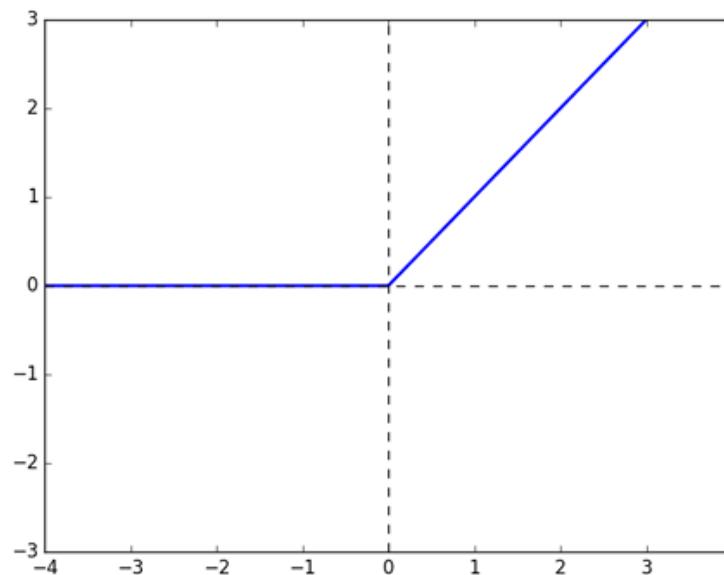
Activation Function

- Some activation functions



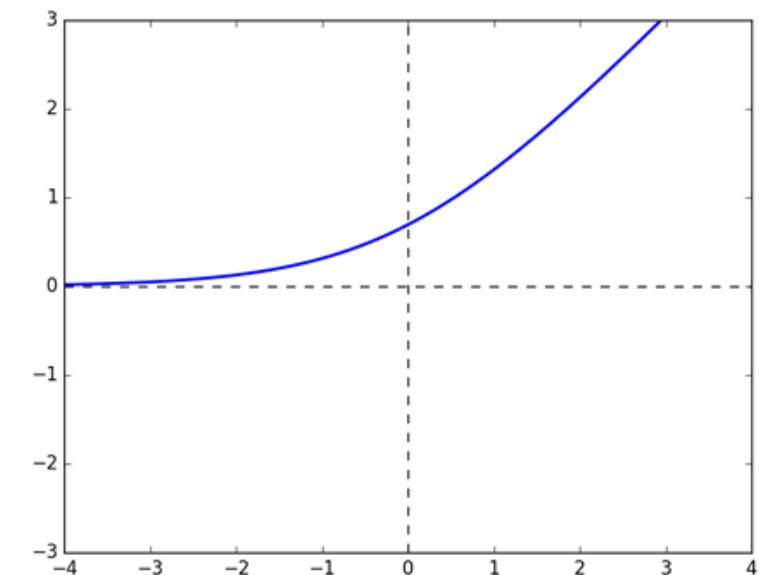
Linear

$$y = z$$



**Rectified Linear Unit
(ReLU)**

$$y = \max(0, z)$$

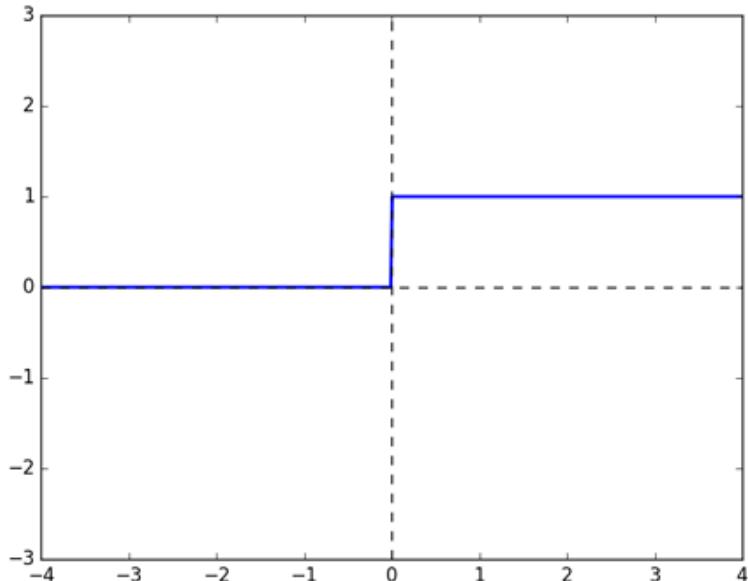


Soft ReLU

$$y = \log(1 + e^z)$$

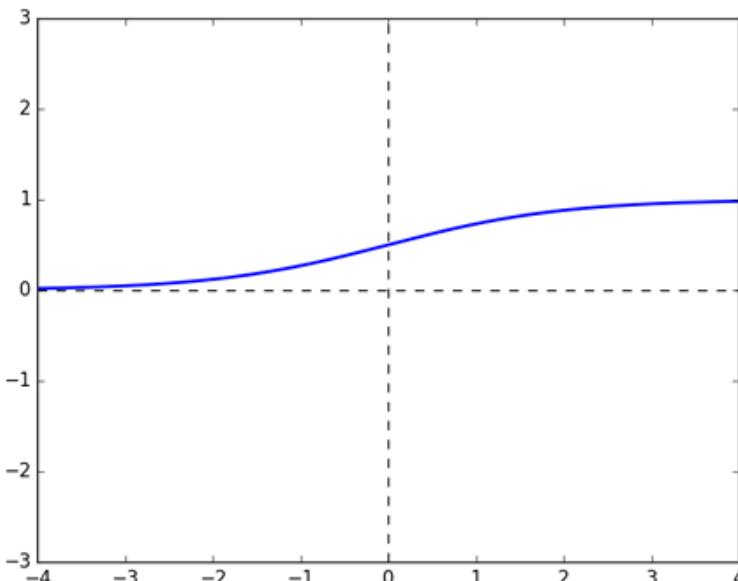
Activation Function

- Some activation functions



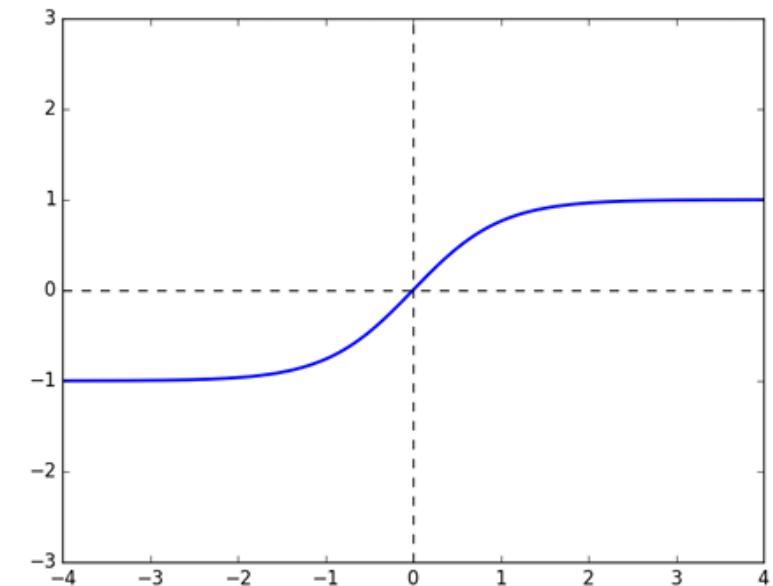
Hard Threshold

$$y = \begin{cases} 1 & \text{if } z > 0 \\ 0 & \text{if } z \leq 0 \end{cases}$$



Sigmoid

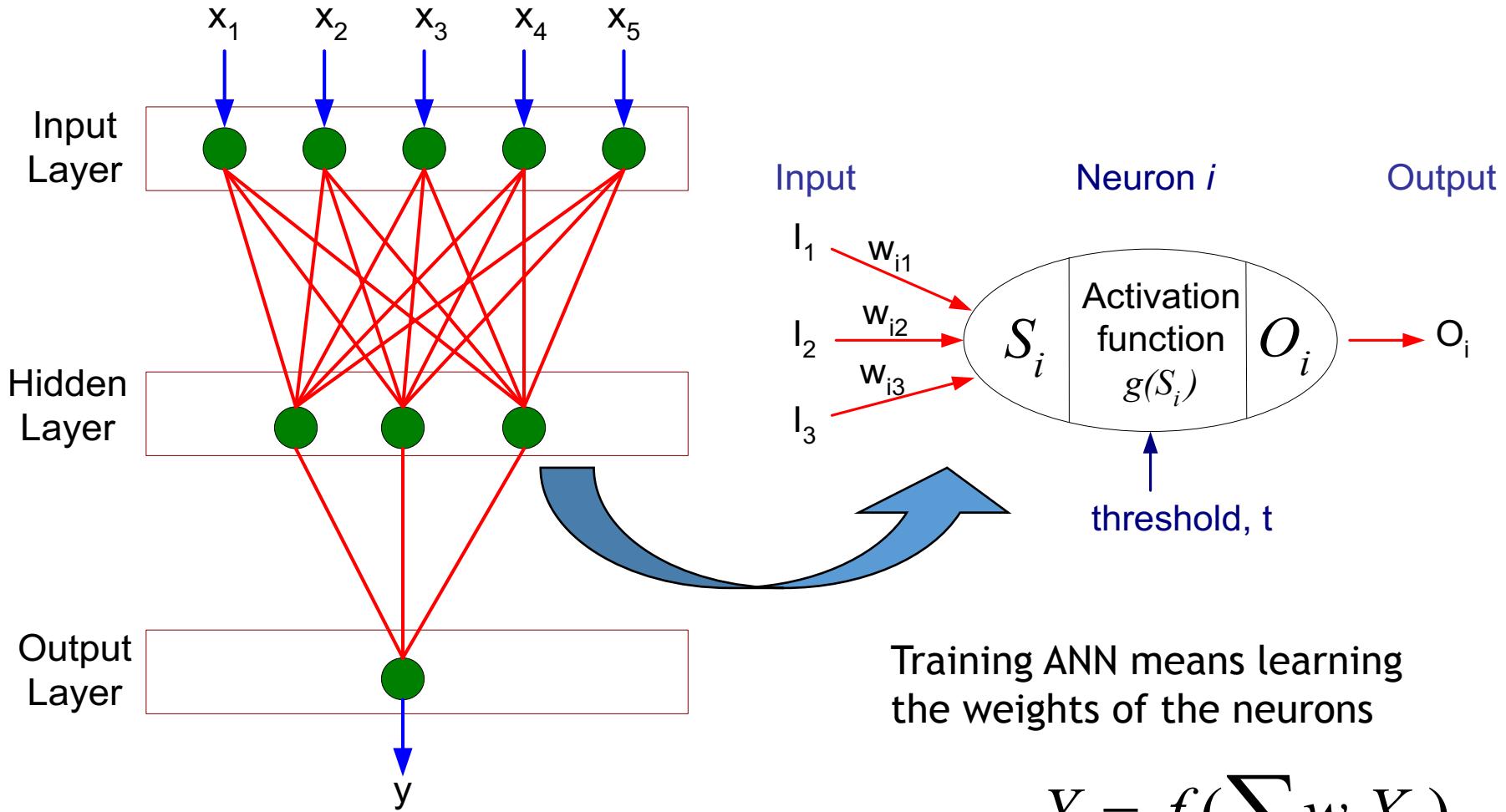
$$y = \frac{1}{1 + e^{-z}}$$



Hyperbolic Tangent (tanh)

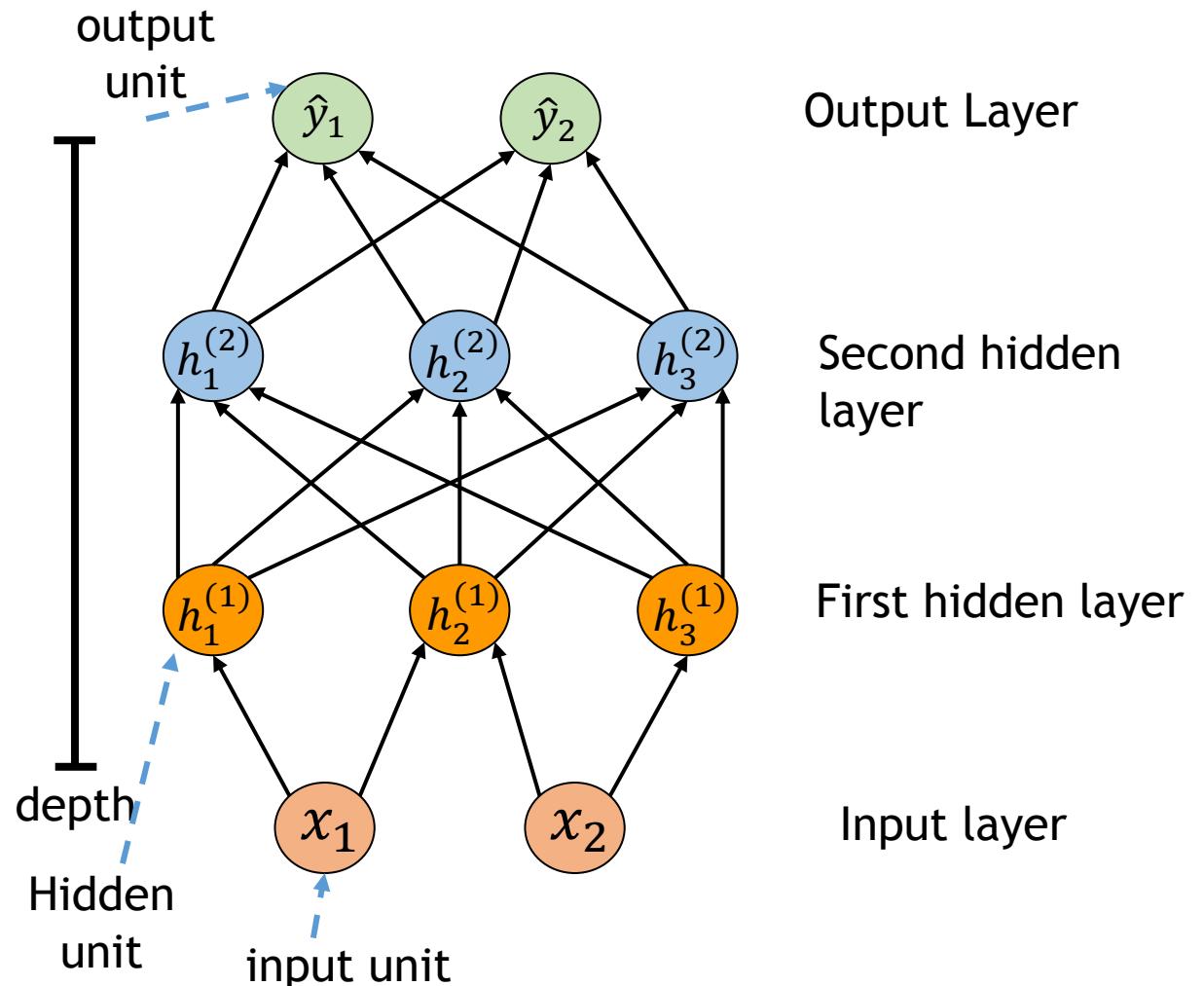
$$y = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$

General Structure of ANN



General Structure of ANN

- These units are much more powerful if we connect many of them into a neural network
- In addition to input nodes and output nodes, some hidden nodes in-between are introduced
- Use hidden units to learn internal features to represent data. Hidden nodes can learn internal representation of data that are not explicit in the input features
- Transfer function of hidden units are non-linear function



Neural Network Terminology

- A neural network is composed of a number of **units (nodes)** that are connected by **links**. Each link has a **weight** associated with it. Nodes have **biases** associated with them. Each unit has an **activation level** and a means to compute the activation level at the next step
- Most neural networks are decomposed of a linear component called **input function**, and a non-linear component call **activation function**. Popular activation functions include: relu, sign-function, and sigmoid function
- The **architecture** of a neural network determines how units are connected and what activation function are used for the network computations. Architectures are subdivided into **feed-forward**, **recurrent networks**, etc. Moreover, **single layer** and **multi-layer** neural networks (that contain **hidden units**) are distinguished
- **Learning in the context of neural networks** centers on finding “good” weights (and biases) for a given architecture so that the error in performing a particular task is minimized. Most approaches center on learning a function from a set of training examples and use hill-climbing and steepest decent hill-climbing approaches to find the best values for the weights

Multilayer Perceptron (MLP)

- Each layer connects N input units to M output units
 - In the simplest case, all input units are connected to all output units. We call this a **fully connected layer**
 - Note: the inputs and outputs for a layer are distinct from the inputs and outputs to the network
 - This means we need an $N \times M$ weight matrix
 - The output units are a function of the input units:

$$\mathbf{y} = f(\mathbf{x}) = \phi(\mathbf{w}\mathbf{x} + b)$$

- A multilayer network consisting of fully connected layers is called a **multilayer perceptron (MLP)**

Multilayer Perceptron (MLP)

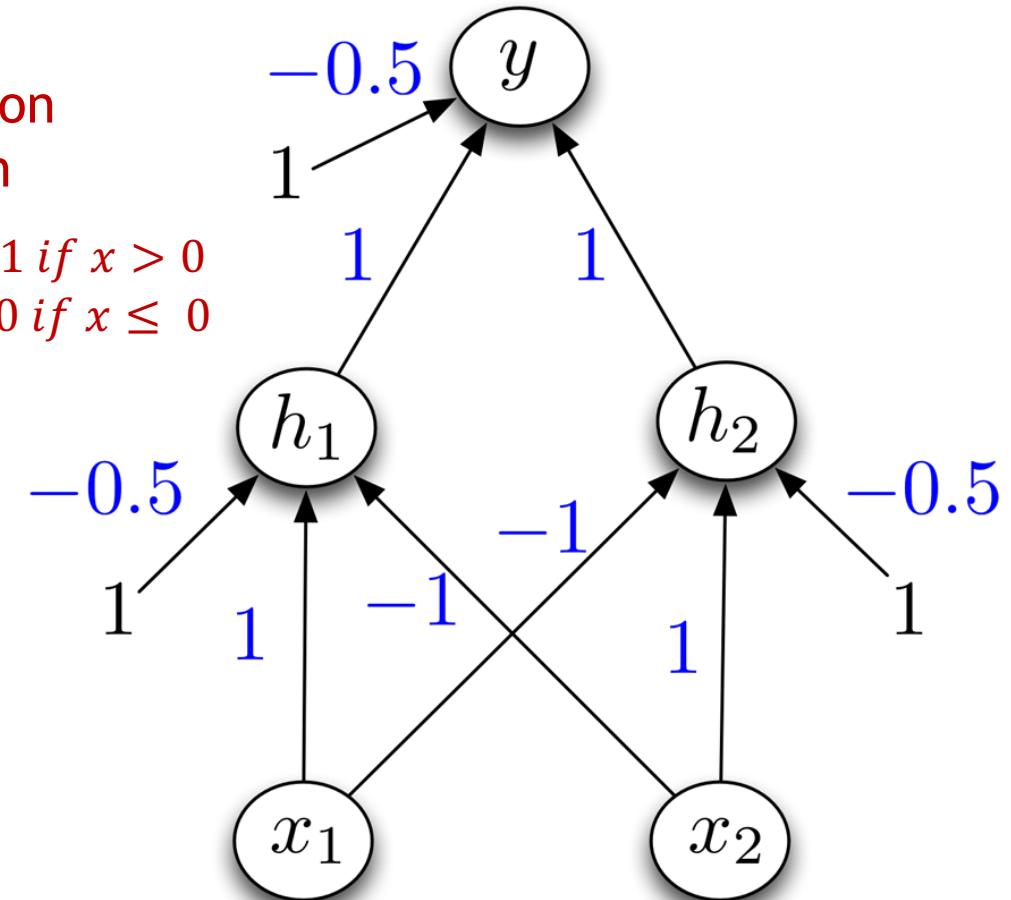
- Designing a network to compute XOR

x_1	x_2	$\text{XOR}(x_1, x_2)$
0	0	0
0	1	1
1	0	1
1	1	0

h_1	h_2	y
0	0	0
0	1	1
1	0	1
0	0	0

Activation function

$$\phi(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{if } x \leq 0 \end{cases}$$



Multilayer Perceptron (MLP)

- Each layer computes a function, so the network computes a composition of functions

$$h^{(1)} = f^{(1)}(x)$$

$$h^{(2)} = f^{(2)}(h^{(1)})$$

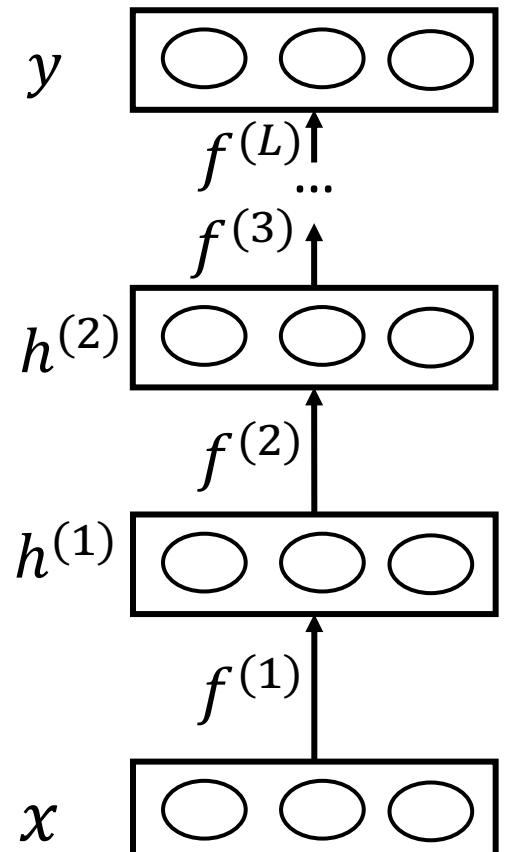
...

$$y = f^{(L)}(h^{(L-1)})$$

- Or more simply

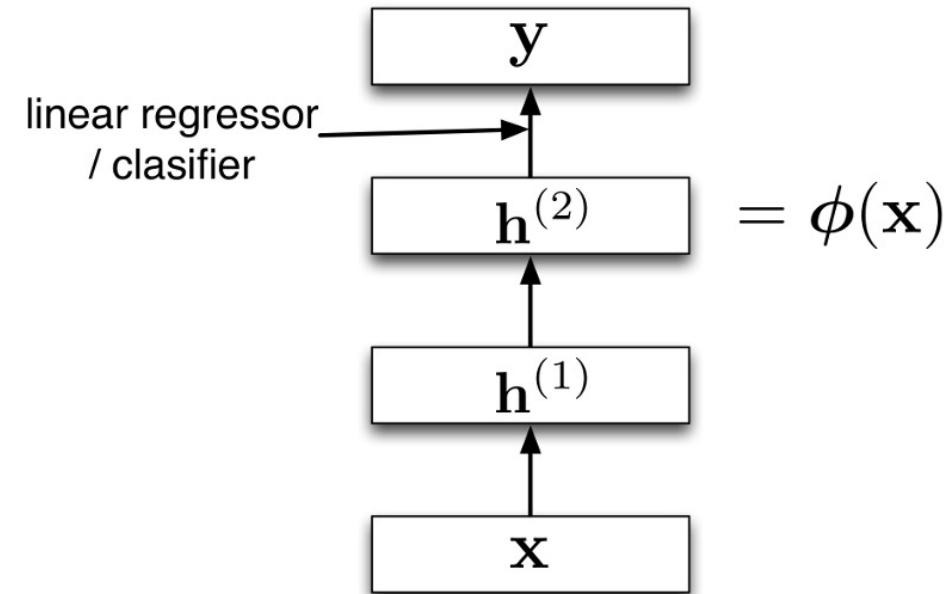
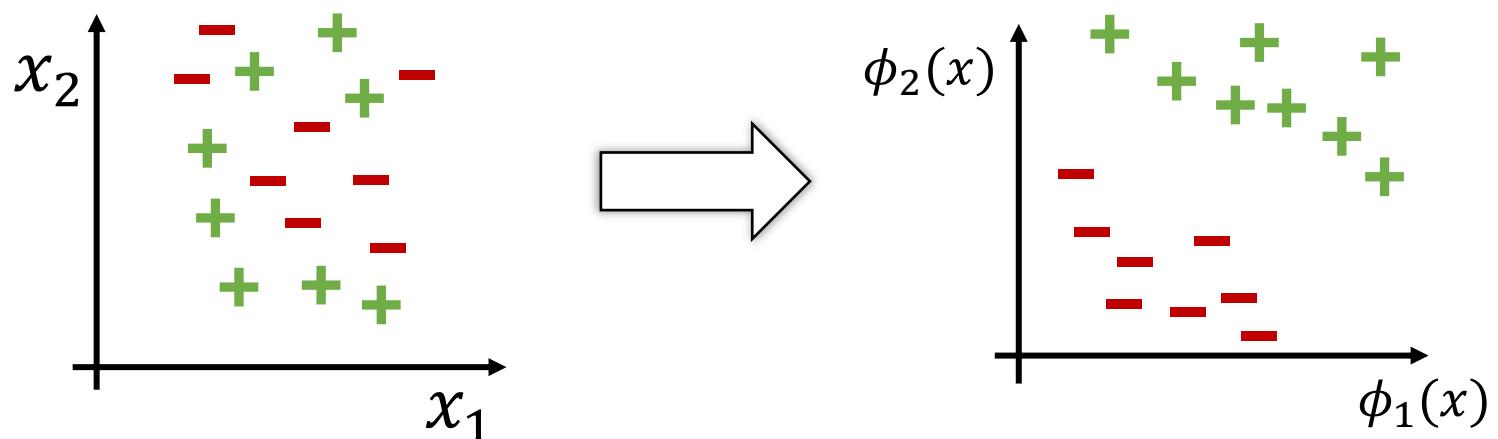
$$y = f^{(L)} \circ \dots \circ f^{(1)}(x)$$

- Neural nets provide modularity: we can implement each layer's computation as a black box



Feature Learning

- Neural nets can be viewed as a way of learning features
- In the original space, data objects x are difficult to separate
- With non-linear transformation, we learn a representation for each data object. In this space, data objects are easier to separate



Expressive Power

- We've seen that there are some functions that linear classifiers can't represent. Are deep networks any better?
- Any sequence of **linear layers** can be equivalently represented with a single linear layer

$$\mathbf{y} = \underbrace{\mathbf{w}^{(3)} \mathbf{w}^{(2)} \mathbf{w}^{(1)}}_{\triangleq \mathbf{w}'} \mathbf{x}$$

- Deep linear networks are no more expressive than linear regression

Expressive Power

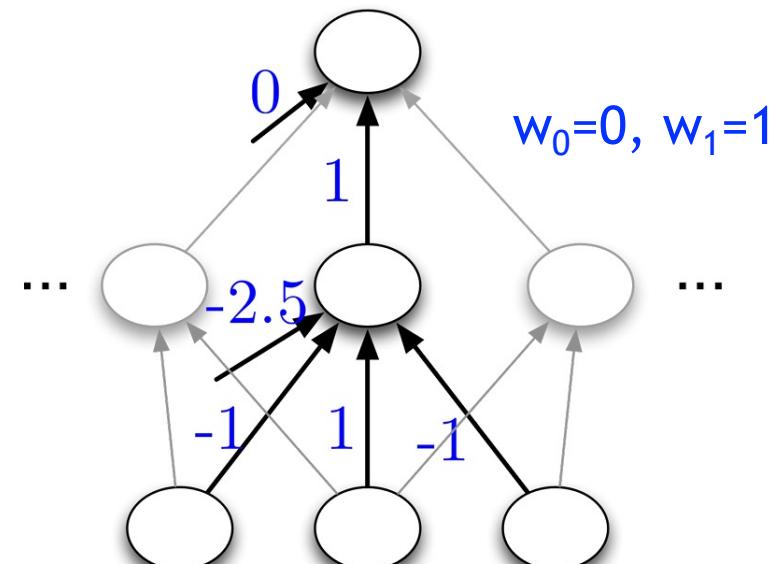
- Multilayer feed-forward neural nets with nonlinear activation functions are **universal approximators**: they can approximate any function arbitrarily well
- This has been shown for various activation functions (thresholds, logistic, ReLU, etc.)
 - Even though ReLU is “almost” linear, it’s nonlinear enough

Expressive Power

- Universality for binary inputs and targets
 - Hard threshold hidden units, linear output
 - Strategy: 2^D hidden units, each of which responds to one particular input configuration
 - Only requires one hidden layer, though it needs to be extremely wide

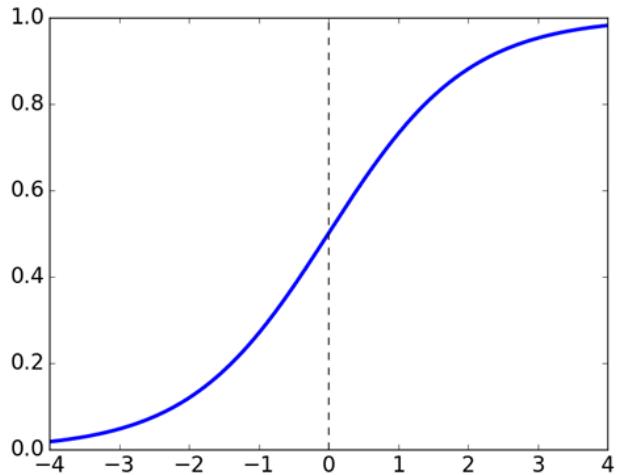
x_1	x_2	x_3	y
:	:	:	:
-1	-1	1	-1
-1	1	-1	1
-1	1	1	1
:	:	:	:

For the i -th config,
 $w_{i0} = -1 \times D + 0.5$,
 $w_{i1} = x_1$,
 $w_{i2} = x_2$,
 $w_{i3} = x_3$

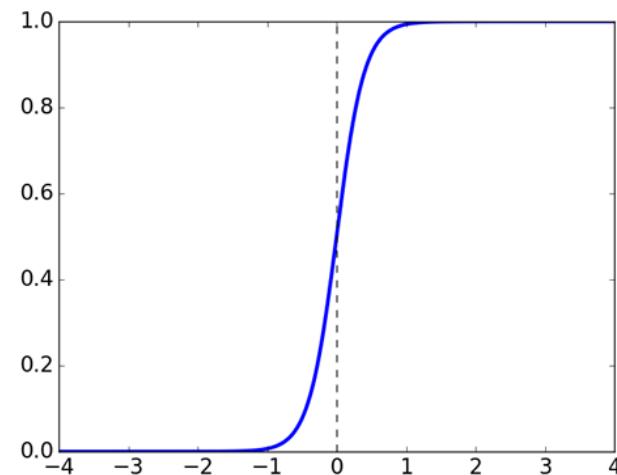


Expressive Power

- What about the sigmoid activation function?
- You can approximate a hard threshold by scaling up the weights and biases:



$$y = \sigma(x)$$



$$y = \sigma(5x)$$

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

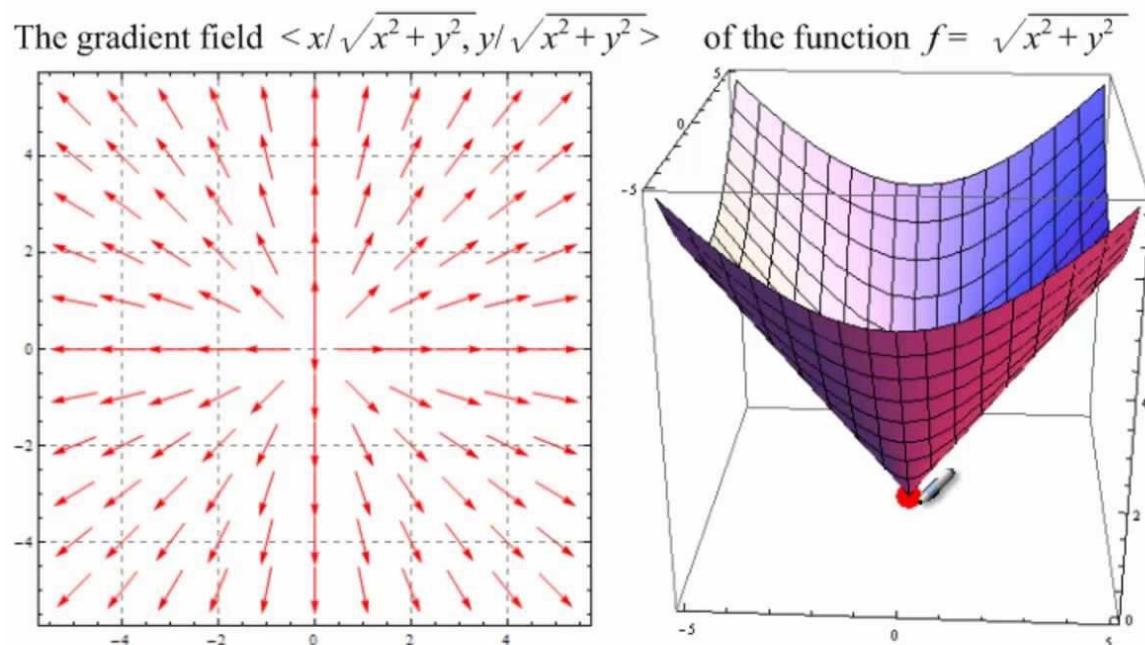
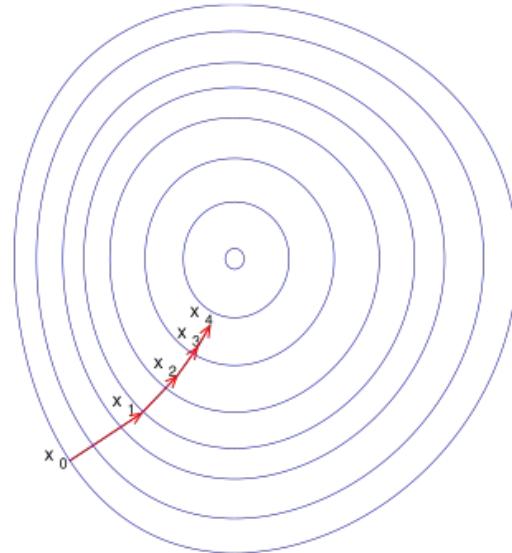
- This is good: logistic units are differentiable, so we can tune them with gradient descent

Backpropagation

- We've seen that multilayer neural networks are powerful. But how can we actually learn them?
- Backpropagation is the central algorithm in neural networks
 - It's an algorithm for computing gradients
 - Really it's an instance of reverse mode automatic differentiation, which is much more broadly applicable than just neural nets
 - This is "just" a clever and efficient use of the **Chain Rule** for derivatives

Gradient Descent

- Gradient descent is a method for unconstrained mathematical optimization. It is a first-order iterative algorithm for minimizing a differentiable multivariate function
 - Gradient descent moves **opposite** the gradient (the direction of steepest descent)



More details: https://en.wikipedia.org/wiki/Gradient_descent

Univariate Chain Rule

- Recall: if $f(x)$ and $x(t)$ are univariate functions, then

$$\frac{d}{dt} f(x(t)) = \frac{df}{dx} \cdot \frac{dx}{dt}.$$

- Example: univariate logistic least squares model

$$z = wx + b$$

$$y = \sigma(z)$$

$$\mathcal{L} = \frac{1}{2}(y - t)^2$$

$$\frac{d\mathcal{L}}{dy} = y - t \quad \frac{d\mathcal{L}}{dz} = \frac{d\mathcal{L}}{dy} \frac{dy}{dz} = \frac{d\mathcal{L}}{dy} \sigma'(z) = (y - t)\sigma'(z)$$

$$\frac{\partial \mathcal{L}}{\partial w} = \frac{d\mathcal{L}}{dy} \frac{dy}{dz} \frac{dz}{dw} = (y - t)\sigma'(z)x \quad \frac{\partial \mathcal{L}}{\partial b} = \frac{d\mathcal{L}}{dy} \frac{dy}{dz} \frac{dz}{db} = (y - t)\sigma'(z)$$

- The goal isn't to obtain closed-form solutions, but to be able to write a program that efficiently computes the derivatives

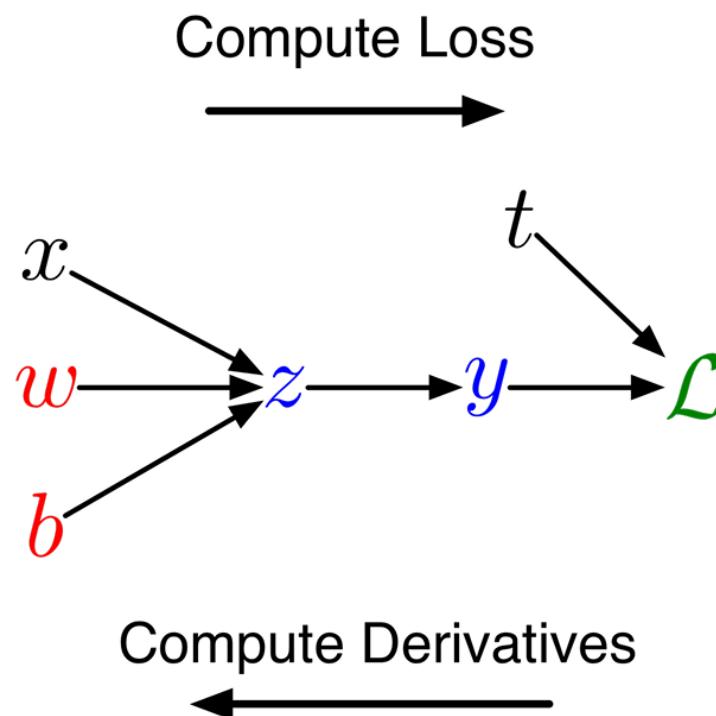
Univariate Chain Rule

- We can diagram out the computations using a **computation graph**
- The nodes represent all the inputs and computed quantities, and the edges represent which nodes are computed directly as a function of which other nodes

$$z = wx + b$$

$$y = \sigma(z)$$

$$\mathcal{L} = \frac{1}{2}(y - t)^2$$



$$\frac{d\mathcal{L}}{dy} = y - t$$

$$\frac{d\mathcal{L}}{dz} = \frac{d\mathcal{L}}{dy} \frac{dy}{dz}$$

$$\frac{d\mathcal{L}}{dw} = \frac{d\mathcal{L}}{dy} \frac{dy}{dz} \frac{dz}{dw}$$

$$\frac{d\mathcal{L}}{db} = \frac{d\mathcal{L}}{dy} \frac{dy}{dz} \frac{dz}{db}$$



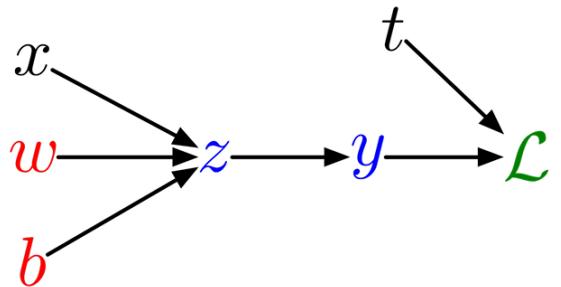
COSC 3337
Data Science I
Section 14623

Deep Learning (contd.)

Instructor: Jingchao Ni
Fall 2024

Multivariate Chain Rule

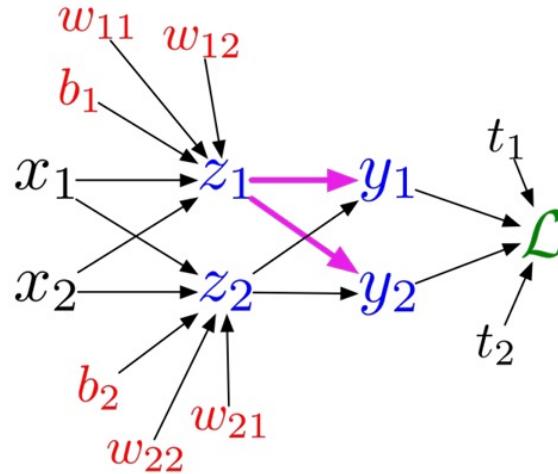
- Problem: what if the computation graph has fan-out > 1?
- This requires the **multivariate Chain Rule**



$$z = wx + b$$

$$y = \sigma(z)$$

$$\mathcal{L} = \frac{1}{2}(y - t)^2$$



$$z_l = \sum_{j=1}^2 w_{lj} x_j + b_l \quad y_k = \frac{e^{z_k}}{\sum_{l=1}^2 e^{z_l}} \quad \text{Softmax}$$

$$\mathcal{L} = - \sum_{k=1}^2 t_k \log y_k \quad \text{Cross-entropy}$$

Multivariate Chain Rule

- Suppose we have a function $f(x, y)$ and functions $x(t)$ and $y(t)$ (all the variables here are scalar-valued). Then

$$\frac{d}{dt} f(x(t), y(t)) = \frac{\partial f}{\partial x} \frac{dx}{dt} + \frac{\partial f}{\partial y} \frac{dy}{dt}$$

- Example

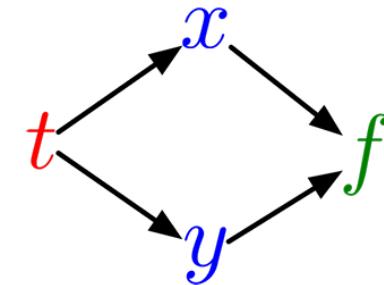
$$f(x, y) = y + e^{xy}$$

$$x(t) = \cos t$$

$$y(t) = t^2$$

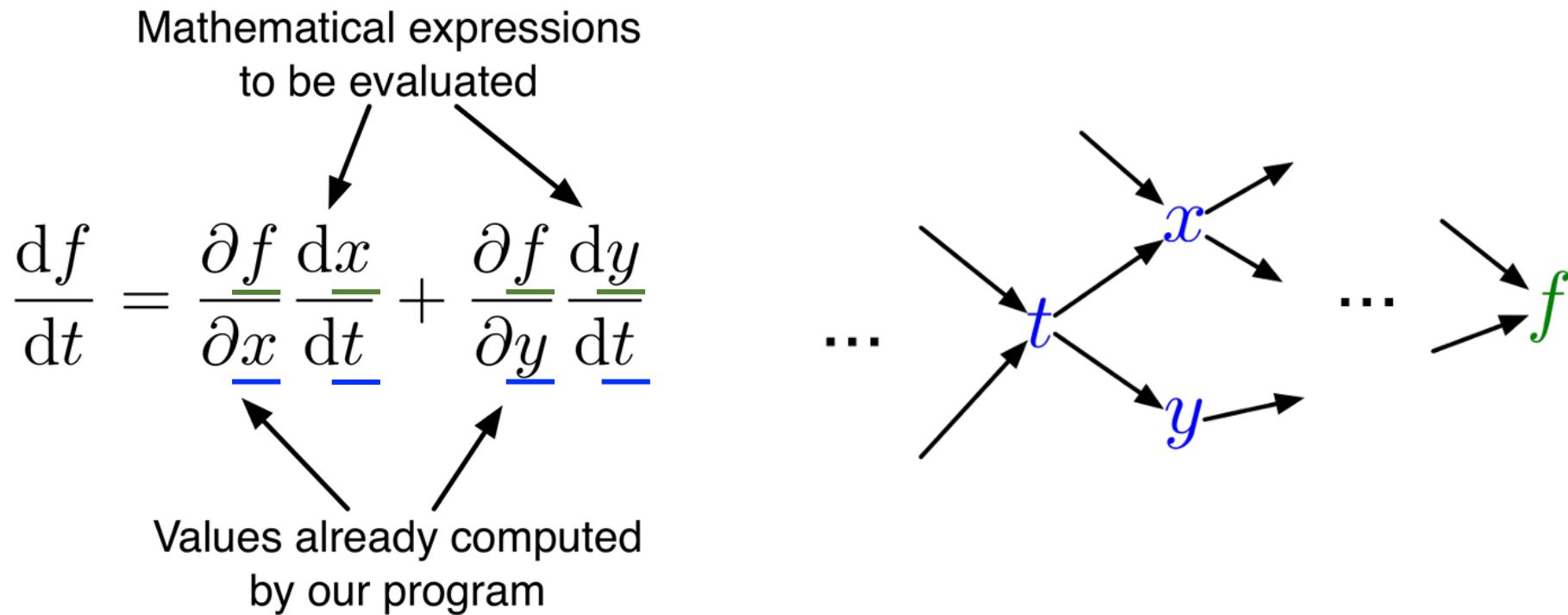
- Plug in to Chain Rule

$$\begin{aligned}\frac{df}{dt} &= \frac{\partial f}{\partial x} \frac{dx}{dt} + \frac{\partial f}{\partial y} \frac{dy}{dt} \\ &= (ye^{xy}) \cdot (-\sin t) + (1 + xe^{xy}) \cdot 2t\end{aligned}$$



Multivariate Chain Rule

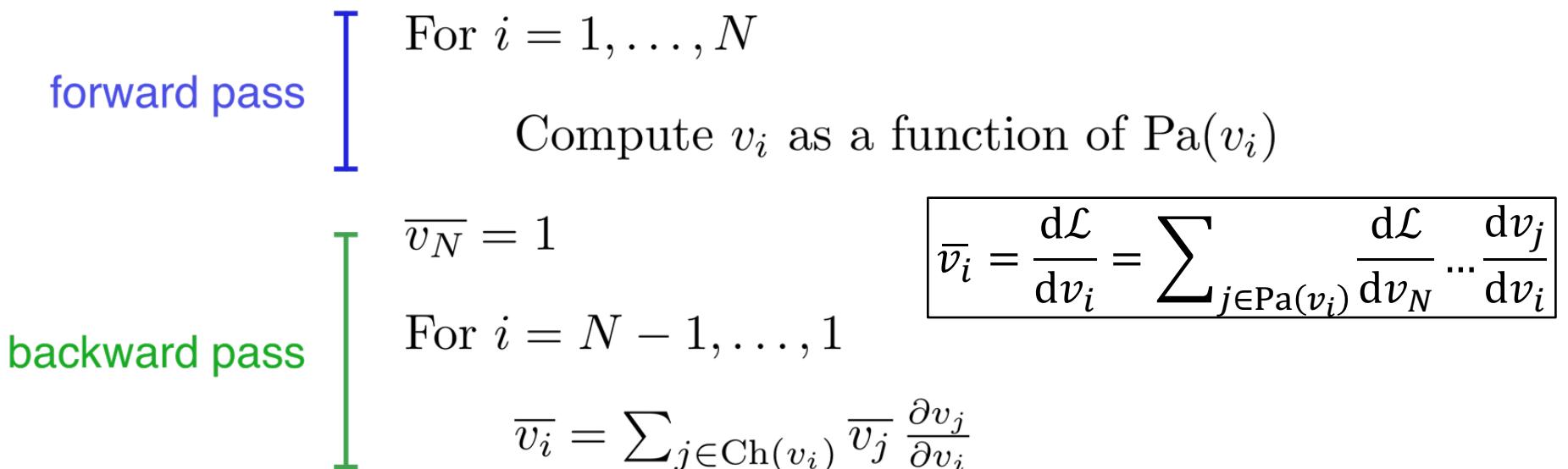
- In the context of backpropagation



Backpropagation

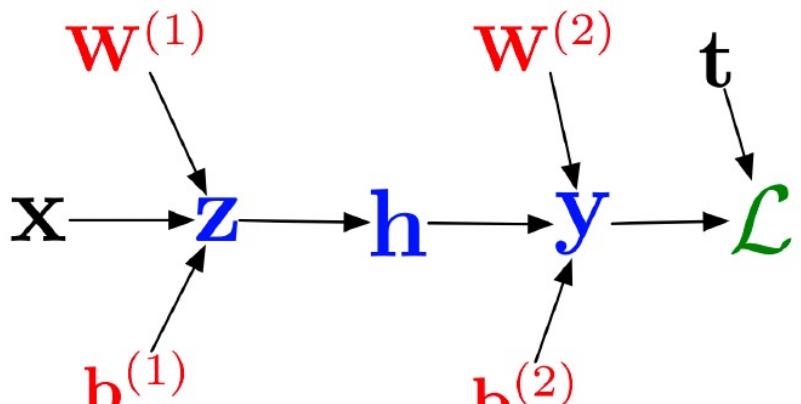
- Full backpropagation algorithm

- Let v_1, v_2, \dots, v_N be a **topological ordering** of the computation graph (i.e., parents come before children)
- v_N denotes the variable we're trying to compute derivatives of (e.g., loss function)



Backpropagation

- Example: MLP



Forward pass: $z = \mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}$

$$\mathbf{h} = \sigma(z)$$

$$\mathbf{y} = \mathbf{W}^{(2)}\mathbf{h} + \mathbf{b}^{(2)}$$

$$\mathcal{L} = \frac{1}{2}\|\mathbf{t} - \mathbf{y}\|^2$$

Backward pass:

$$\frac{d\mathcal{L}}{dy} = y - t$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}^{(2)}} = \frac{d\mathcal{L}}{dy} \mathbf{h}^T$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{b}^{(2)}} = \frac{d\mathcal{L}}{dy}$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{h}} = (\mathbf{W}^{(2)})^T \frac{d\mathcal{L}}{dy}$$

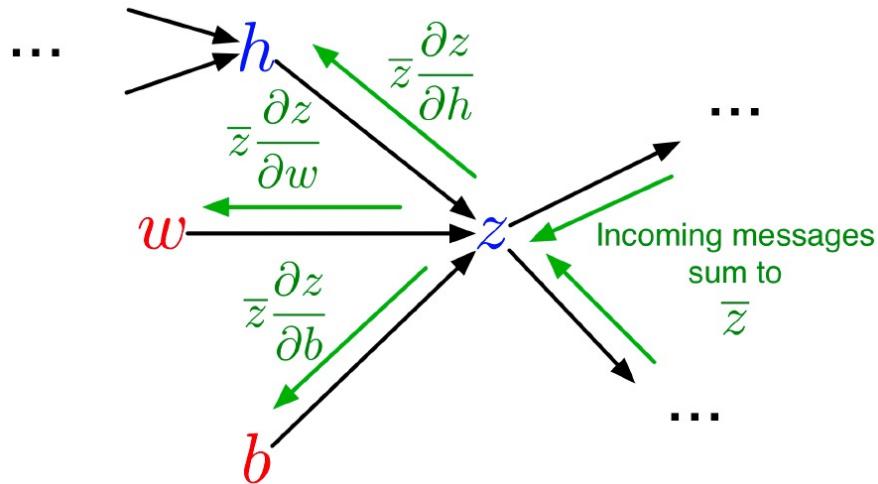
$$\frac{d\mathcal{L}}{dz} = \frac{\partial \mathcal{L}}{\partial \mathbf{h}} \sigma'(z)$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}^{(1)}} = \frac{d\mathcal{L}}{dz} \mathbf{x}^T$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{b}^{(1)}} = \frac{d\mathcal{L}}{dz}$$

Backpropagation

- Backpropagation as message passing



- Each node receives a bunch of messages from its children, which it aggregates to get its error signal. It then passes messages to its parents
- This provides modularity, since each node only has to know how to compute derivatives with respect to its arguments, and doesn't have to know anything about the rest of the graph

Computational Cost

- Computational cost of forward pass: one add-multiply operation per weight

$$z = \mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}$$

- Computational cost of backward pass: two add-multiply operations per weight

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}^{(2)}} = \frac{d\mathcal{L}}{dy} \mathbf{h}^T \quad \frac{\partial \mathcal{L}}{\partial \mathbf{h}} = (\mathbf{W}^{(2)})^T \frac{d\mathcal{L}}{dy}$$

- The backward pass is about as expensive as two forward passes
- For a multilayer perceptron, this means the cost is linear in the number of layers, quadratic in the number of units per layer

Backpropagation

- Backprop is used to train the overwhelming majority of neural nets today
 - Even optimization algorithms much fancier than gradient descent (e.g. second-order methods) use backprop to compute the gradients
- Despite its practical success, backprop is believed to be neurally implausible
 - No evidence for biological signals analogous to error derivatives

Backpropagation

- We've seen three different ways of looking at gradients
 - Geometric: visualization of gradient in weight space
 - Algebraic: mechanics of computing the derivatives
 - Implementational: efficient implementation on the computer
- When thinking about neural nets, it's important to be able to shift between these different perspectives

Some Design Issues in ANN

- Number of nodes in input layer
 - One input node per binary/**continuous attribute**
 - k nodes for each **categorical attribute** with k values (using one-hot representation)
- Number of nodes in output layer
 - One output for binary class problem (for probability)
 - k for k -class problem (one probability per class)
 - Many other possibilities
- Number of nodes in hidden layer
- Initial weights and biases: usually random values
 - Gradient descent may converge to local minimum
 - Model training can be very time consuming, but testing can be fast

More NN Terminology

- **Batch size:** Size of the training set that is used in each iteration. For example, if batch size is 4, weights are adjusted using 4 training examples in each iteration of the weight learning process. Sometimes called mini-batch size
- **Epoch** is 1 complete cycle where Neural network has seen all the data. That is if the batch size is 128 and training set size 2048, one epoch is completed after 16 iterations. That is, if in the above example the NN learning used 256 iterations that would be 16 epochs as $16 \times 16 = 256$

```
For epoch = 1 to 16 do:
```

```
    For iteration = 1 to 16 do:
```

```
        Batch = Get_Next_Batch(Data, Batch_Size=128)
```

```
        ...
```

More NN Terminology

- In mathematics, statistics, finance, computer science, particularly in machine learning and inverse problems, **regularization** is the process of adding information in order to solve an ill-posed problem or to prevent overfitting, usually to an objective / cost function. Example L1 regularization:

$$\sum_{i=1}^n (Y_i - \sum_{j=1}^p X_{ij}\beta_j)^2 + \lambda \sum_{j=1}^p |\beta_j|$$

- In the case of neural networks β_i are the p weights w_i of the neural network and the first term is the squared prediction error and λ is a parameter called **regularization rate**

More NN Terminology

- **Batch normalization** (also known as **batch norm**) is a method used to make artificial neural networks more stable and less sensitive to overfitting through normalization of the layers' inputs by re-centering and re-scaling. It was proposed by Sergey Ioffe and Christian Szegedy in 2015.
 - Details: https://en.wikipedia.org/wiki/Batch_normalization#cite_note-1
- **Dropout**: this technique consists of removing some nodes so that the NN is not too heavy. This can be implemented during the training phase. The idea is that we do not want our NN to be overwhelmed by information, especially if we consider that some nodes might be redundant and useless. So, while building our algorithm, we can decide to keep, for each training stage, each node with probability p (called ‘keep probability’) or drop it with probability $1-p$ (called ‘drop probability’)
- Remarks: Both techniques are used to alleviate overfitting

Backpropagation in PyTorch

```
import numpy as np
import torch
from torch import nn

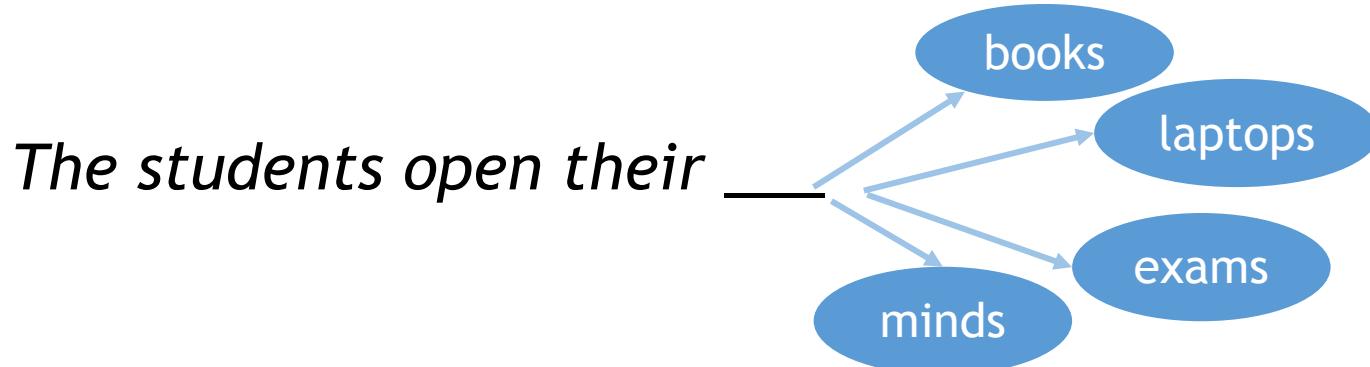
class SequentialFC(nn.Module):
    def __init__(self, in_dim, dim1, dim2):
        super(SequentialFC, self).__init__()
        self.fc1 = nn.Linear(in_dim, dim1)
        self.fc2 = nn.Linear(dim1, dim2)

    def forward(self, inputs):
        out1 = self.fc1(inputs)
        out2 = self.fc2(nn.functional.relu(out1))
        return out1, out2

mlp = SequentialFC(1024, 256, 32)
mlp.to(torch.device('cuda'))
opt = torch.optim.SGD(mlp.parameters(), lr=0.01, momentum=0.9, weight_decay=1e-4)
# opt = torch.optim.Adam(mlp.parameters(), lr=0.01, weight_decay=1e-4)
...
opt.zero_grad()
train_loss.backward()
opt.step()
...
```

Recurrent Neural Networks (RNNs)

- **Language Modeling** is the task of predicting what word comes next



- More formally: given a sequence of words $x^{(1)}, x^{(2)}, \dots, x^{(t)}$, compute the probability distribution of the next word $x^{(t+1)}$:

$$p(x^{(t+1)} | x^{(t)}, \dots, x^{(1)})$$

where $x^{(t+1)}$ can be any word in the vocabulary $V = \{w_1, \dots, w_{|V|}\}$

- A system that does this is called a Language Model

Language Modeling

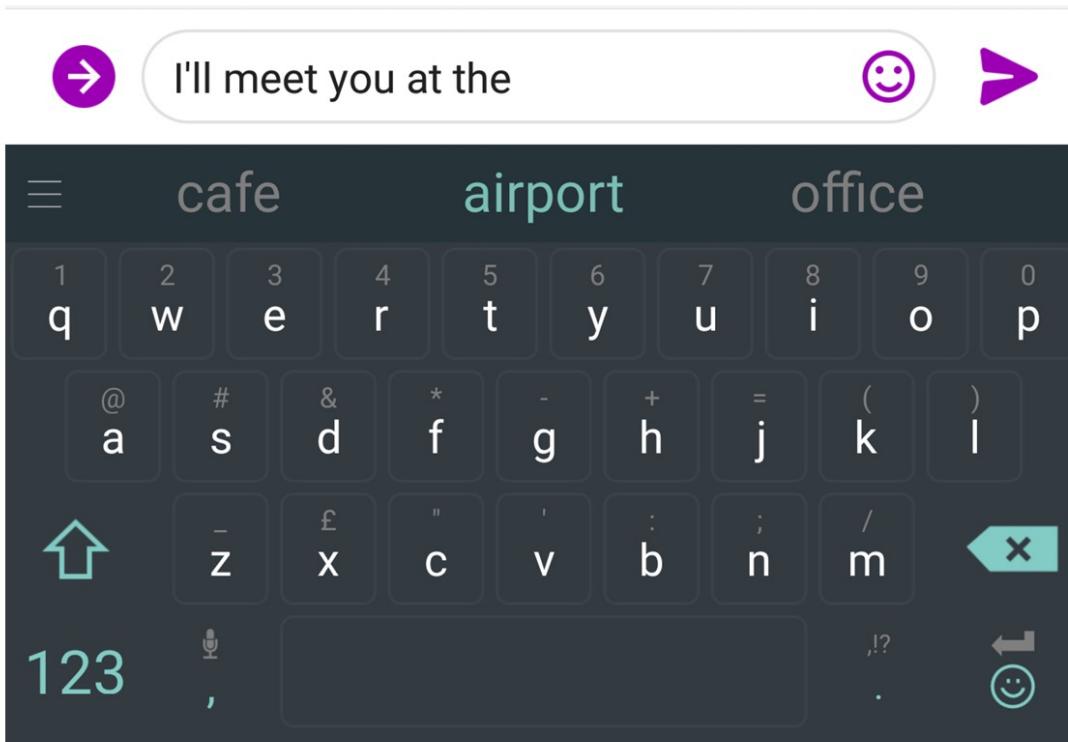
- You can also think of a Language Model as a system that **assigns probability to a piece of text**
- For example, if we have some text $x^{(1)}, x^{(2)}, \dots, x^{(T)}$, then the probability of this text (according to the Language Model) is:

$$P(x^{(1)}, \dots, x^{(T)}) = P(x^{(1)})P(x^{(2)}|x^{(1)}) \dots P(x^{(T)}|x^{(T-1)}, \dots, x^{(1)})$$

$$= \prod_{t=1}^T P(x^{(t)}|x^{(t-1)}, \dots, x^{(1)})$$

This is what an LM provides

Language Models are Used Everyday



Google

what is the |

what is the **weather**

what is the **meaning of life**

what is the **dark web**

what is the **xfl**

what is the **doomsday clock**

what is the **weather today**

what is the **keto diet**

what is the **american dream**

what is the **speed of light**

what is the **bill of rights**

Google Search I'm Feeling Lucky

n-gram Language Models

the students opened their _____

- Question: How to learn a Language Model?
- Answer (pre- Deep Learning): learn a *n-gram Language Model*
- Definition: A **n-gram** is a chunk of **n** consecutive words
 - **unigrams**: “the”, “students”, “opened”, “their”
 - **bigrams**: “the students”, “students opened”, “opened their”
 - **trigrams**: “the students opened”, “students opened their”
 - **4-grams**: “the students opened their”
- Idea: Collect statistics about how frequent different n-grams are, and use these to predict next word

n-gram Language Models

- First we make a **simplifying assumption**: $x^{(t+1)}$ depends only on the preceding $n-1$ words

$$P(x^{(t+1)} | x^{(t)}, \dots, x^{(1)}) = P(x^{(t+1)} | \underbrace{x^{(t)}, \dots, x^{(t-n+2)}}_{n-1 \text{ words}}) \quad (\text{Assumption})$$

$$\begin{aligned} \text{Prob of n-gram} &\rightarrow P(x^{(t+1)}, x^{(t)}, \dots, x^{(t-n+2)}) \\ &= \frac{P(x^{(t+1)}, x^{(t)}, \dots, x^{(t-n+2)})}{P(x^{(t)}, \dots, x^{(t-n+2)})} \\ \text{Prob of (n-1)-gram} &\rightarrow \end{aligned} \quad (\text{definition of conditional prob})$$

- Question: How do we get these n -gram and $(n-1)$ -gram probabilities?
- Answer: By **counting** them in some large corpus of text

$$\approx \frac{\text{count}(x^{(t+1)}, x^{(t)}, \dots, x^{(t-n+2)})}{\text{count}(x^{(t)}, \dots, x^{(t-n+2)})} \quad (\text{statistical approximation})$$

n-gram Language Models: Example

- Suppose we are learning a 4-gram Language Model

~~as the proctor started the clock, the~~ students opened their
discard condition on this

$$P(w|\text{students opened their}) = \frac{\text{count(students opened their } w\text{)}}{\text{count(students opened their)}}$$

- For example, suppose that in the corpus:

- “students opened their” occurred 1000 times
 - “students opened their books” occurred 400 times
 - $\rightarrow P(\text{books} \mid \text{students opened their}) = 0.4$
 - “students opened their exams” occurred 100 times
 - $\rightarrow P(\text{exams} \mid \text{students opened their}) = 0.1$

Should we have discarded the “proctor” context?





COSC 3337
Data Science I
Section 14623

Deep Learning (contd.)

Instructor: Jingchao Ni
Fall 2024

n-gram Language Models: Sparsity Problem

Sparsity Problem 1

Problem: What if “students opened their w ” never occurred in data? Then w has probability 0.

(Partial) Solution: Add small δ to the count for every $w \in V$. This is called *smoothing*.

$$P(w|\text{students opened their}) = \frac{\text{count(students opened their } w\text{)}}{\text{count(students opened their)}}$$

Sparsity Problem 2

Problem: What if “students opened their” never occurred in data? Then we can’t calculate probability for any w !

(Partial) Solution: Just condition on “opened their” instead. This is called *backoff*.

Note: Increasing n makes sparsity problems worse.
Typically we can’t have n bigger than 5.

Storage of n-gram Language Models

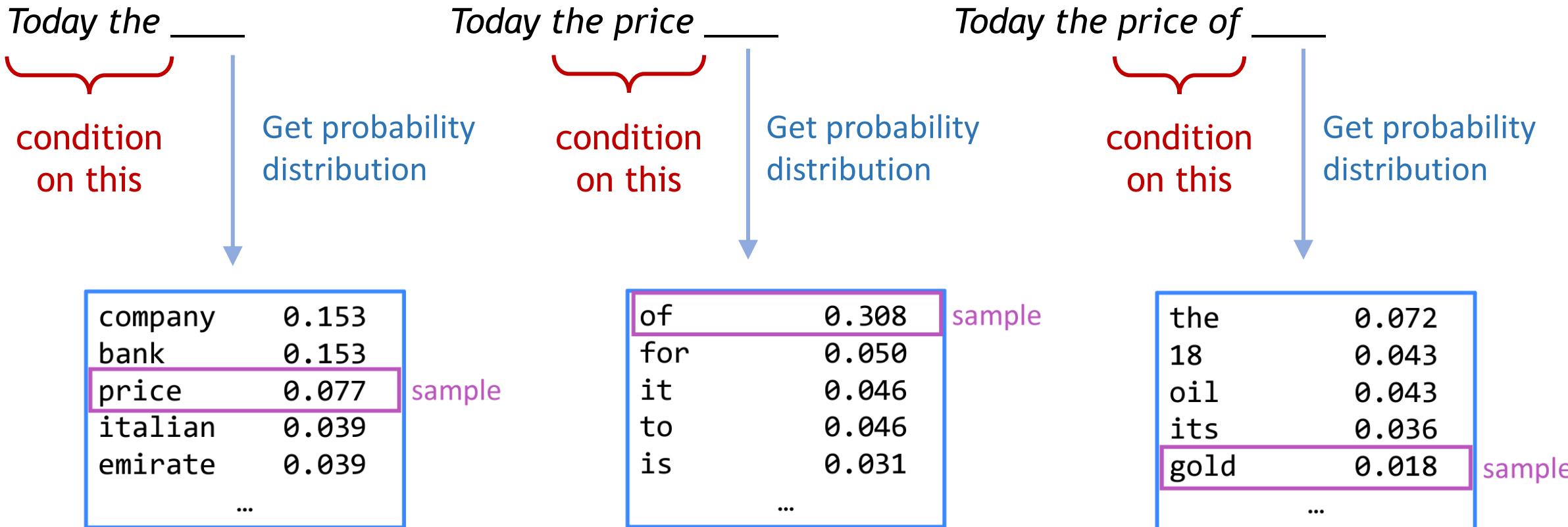
Storage: Need to store count for all n-grams you saw in the corpus.

$$P(w|\text{students opened their}) = \frac{\text{count(students opened their } w\text{)}}{\text{count(students opened their)}}$$

Note: Increasing n or increasing corpus will increase the model size.

Generating Text with an n-gram LM

■ Trigram LM



Generating Text with an n-gram LM

“today the price of gold per ton, while production of shoe lasts and
shoe industry, the bank intervened just after it considered and
rejected an imf demand to rebuild depleted european stocks, sept
30 end primary 76 cts a share.”

Surprisingly grammatical!

... but **incoherent**. We need to consider more than three words
at a time if we want to model language well.

But increasing n worsens sparsity problem, and increases
model size ...

Neural Language Model

- Recall the Language Modeling task:
 - Input: a sequence of words $x^{(1)}, x^{(2)}, \dots, x^{(t)}$
 - Output: prob distribution of the next word $p(x^{(t+1)}|x^{(t)}, \dots, x^{(1)})$
- A fixed-window neural Language Model

~~as the proctor started the clock, the students opened their~~ _____
discard 
condition on this

A Fixed-Window Neural Language Model

output distribution

$$\hat{\mathbf{y}} = \text{softmax}(\mathbf{U}\mathbf{h} + \mathbf{b}_2) \in \mathbb{R}^{|V|}$$

hidden layer

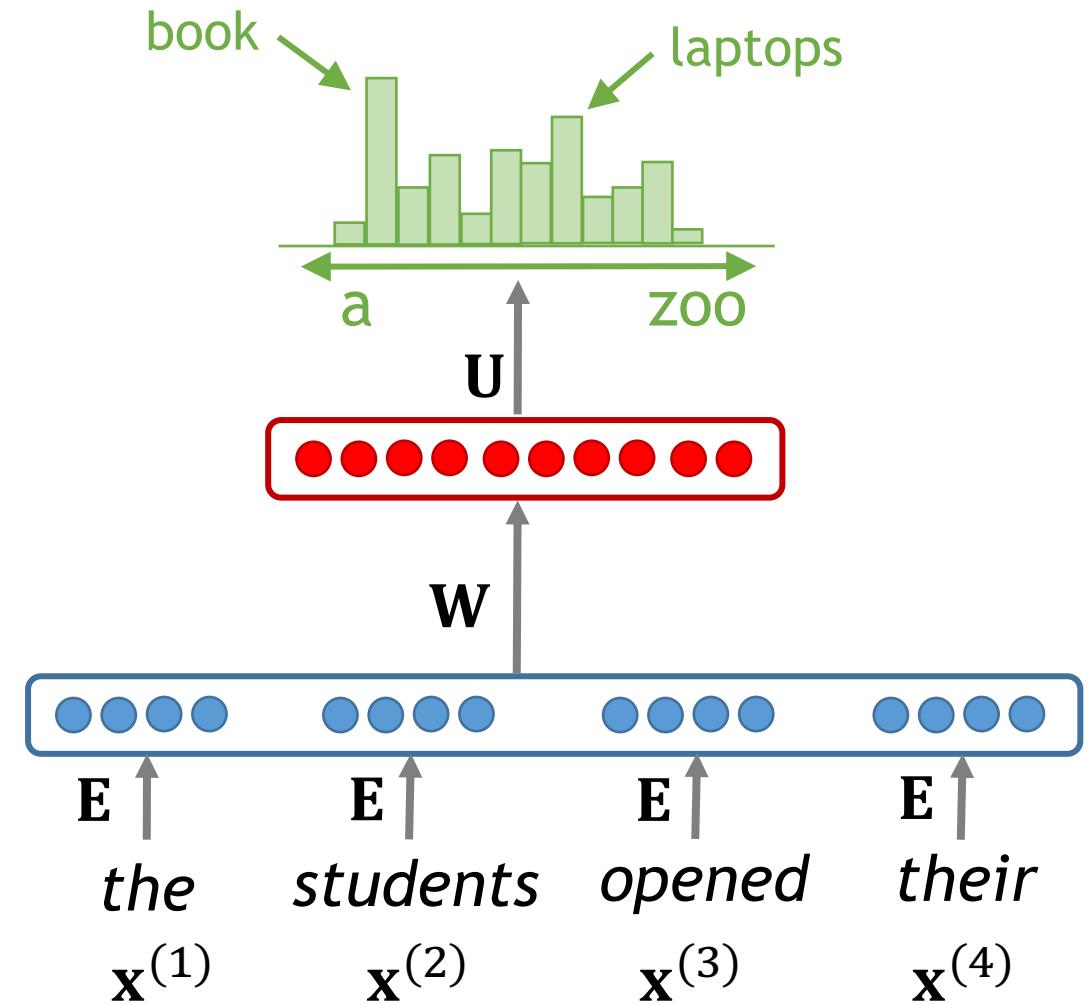
$$\mathbf{h} = f(\mathbf{W}\mathbf{e} + \mathbf{b}_1)$$

concatenated word embeddings

$$\mathbf{e} = [\mathbf{e}^{(1)}; \mathbf{e}^{(2)}; \mathbf{e}^{(3)}; \mathbf{e}^{(4)}]$$

words / one-hot vectors

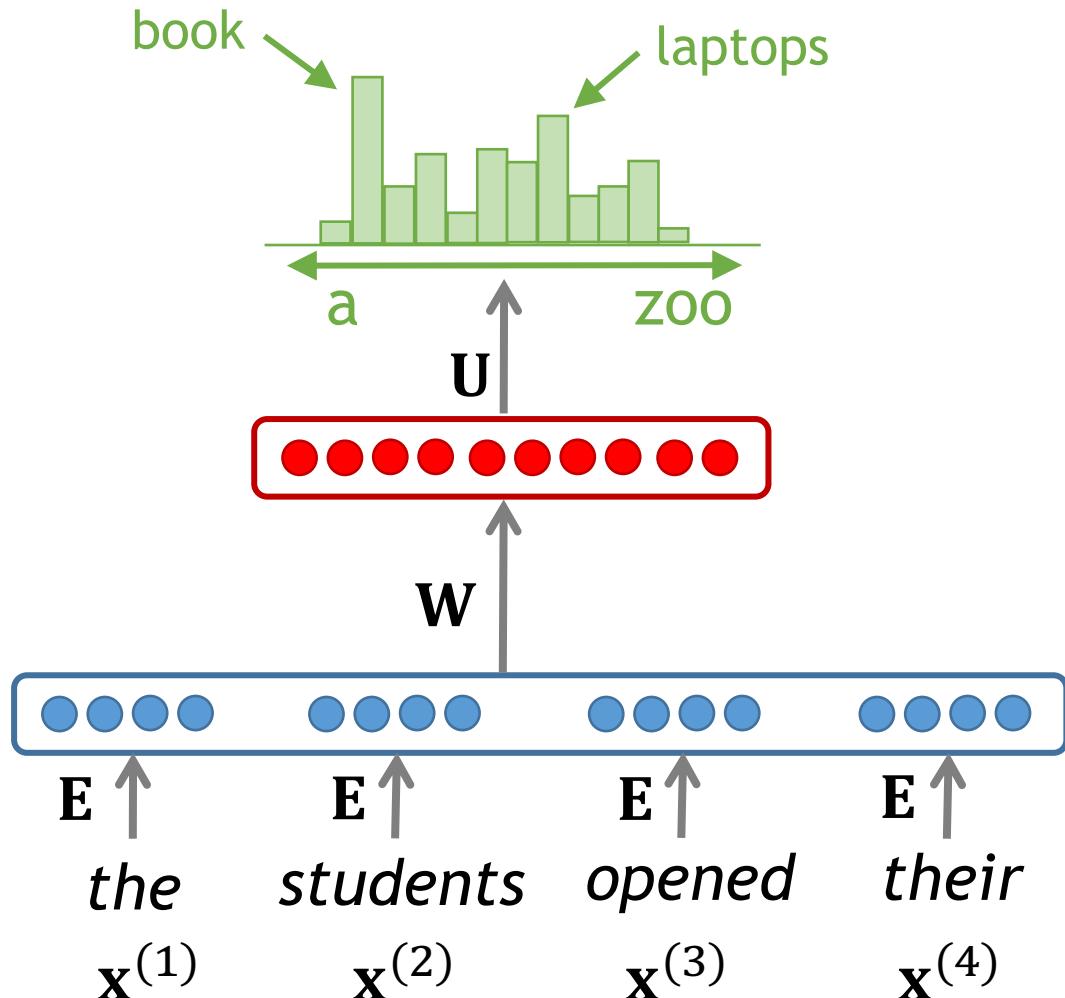
$$\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \mathbf{x}^{(3)}, \mathbf{x}^{(4)}$$



A Fixed-Window Neural Language Model

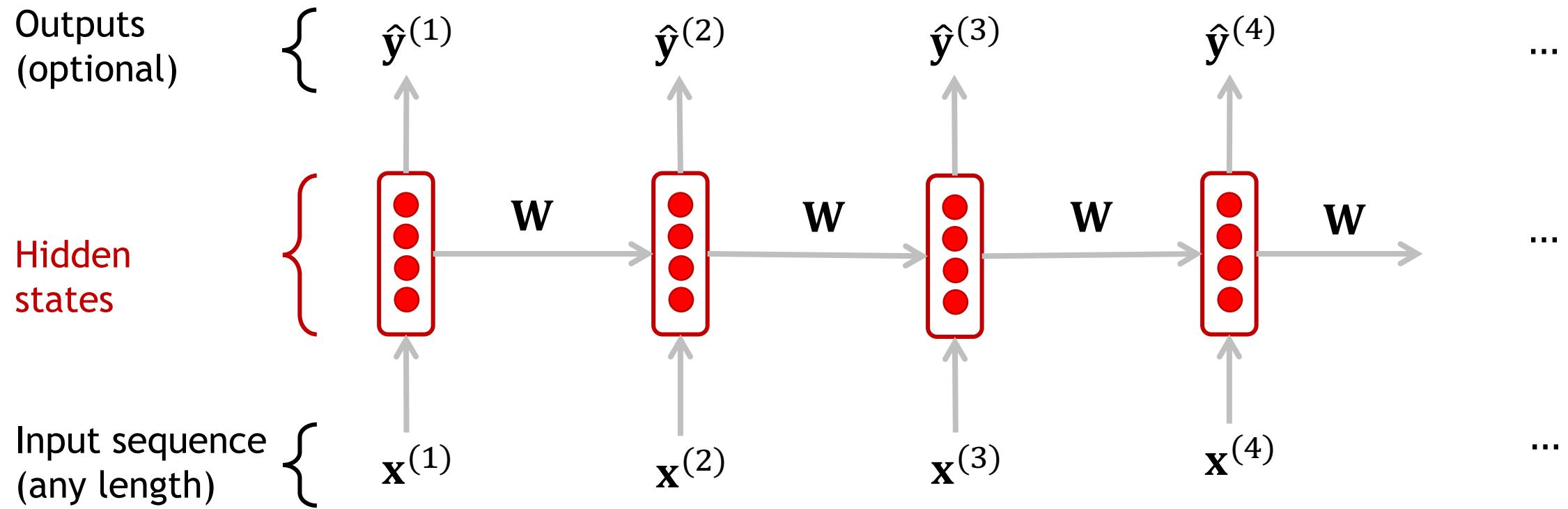
- Improvements over n -gram LM:
 - No sparsity problem
 - The input is a concatenation of words
 - Don't need to store all observed n -grams
 - The model stores parameters E , W , U
- Remaining problems:
 - Fixed window is **too small**
 - Enlarging window enlarge W
 - Window can never be large enough
 - $x^{(1)}$ and $x^{(2)}$ are multiplied by completely different weights in W . **No symmetry** in how the inputs are processed.

We need a neural architecture
that can process **any length** input



Recurrent Neural Networks (RNN)

- A family of neural architectures
 - **Core idea:** apply the same weights W repeatedly



A RNN Language Model

output distribution

$$\hat{y}^{(t)} = \text{softmax}(\mathbf{U}\mathbf{h}^{(t)} + \mathbf{b}_2) \in \mathbb{R}^{|V|}$$

hidden states

$$\mathbf{h}^{(t)} = \sigma(\mathbf{W}_h \mathbf{h}^{(t-1)} + \mathbf{W}_e \mathbf{e}^{(t)} + \mathbf{b}_1)$$

$\mathbf{h}^{(0)}$ is the initial hidden state

word embeddings

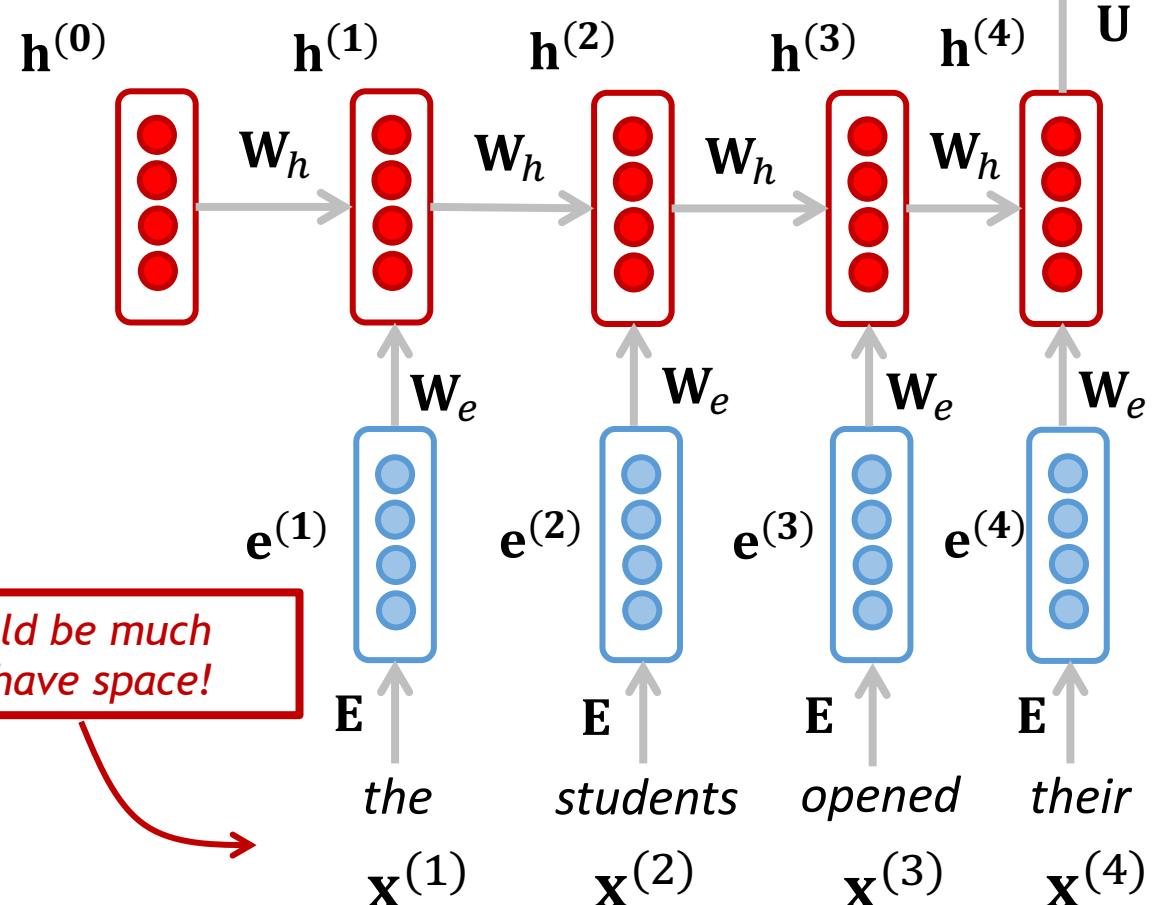
$$\mathbf{e}^{(t)} = \mathbf{E}\mathbf{x}^{(t)}$$

Note: this input sequence could be much longer, but this slide doesn't have space!

words / one-hot vectors

$$\mathbf{x}^{(t)} \in \mathbb{R}^{|V|}$$

$$\hat{y}^{(4)} = P(\mathbf{x}^{(4)} | \text{the students opened their})$$



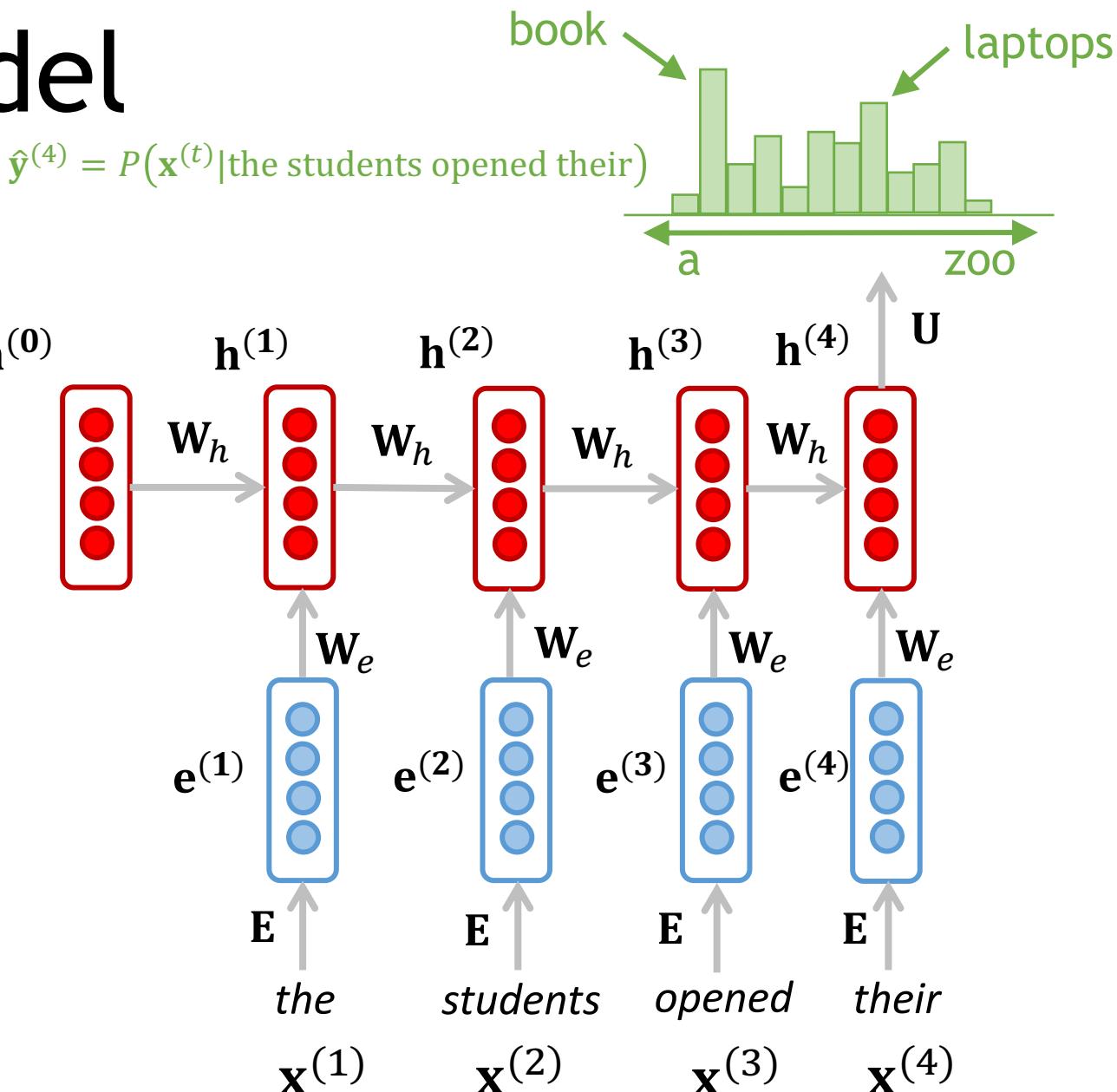
A RNN Language Model

RNN Advantages:

- Can process **any length** input
- Computation for step t can (in theory) use information from **many steps back**
- **Model size doesn't increase** for longer input
 - Parameters: E , W_e , W_h , U , b_1 , b_2
- Same weights applied on every timestep, so there is **symmetry** in how inputs are processed

RNN Disadvantages:

- Recurrent computation is **slow**
- In practice, difficult to access information from **many steps back**



Training a RNN Language Model

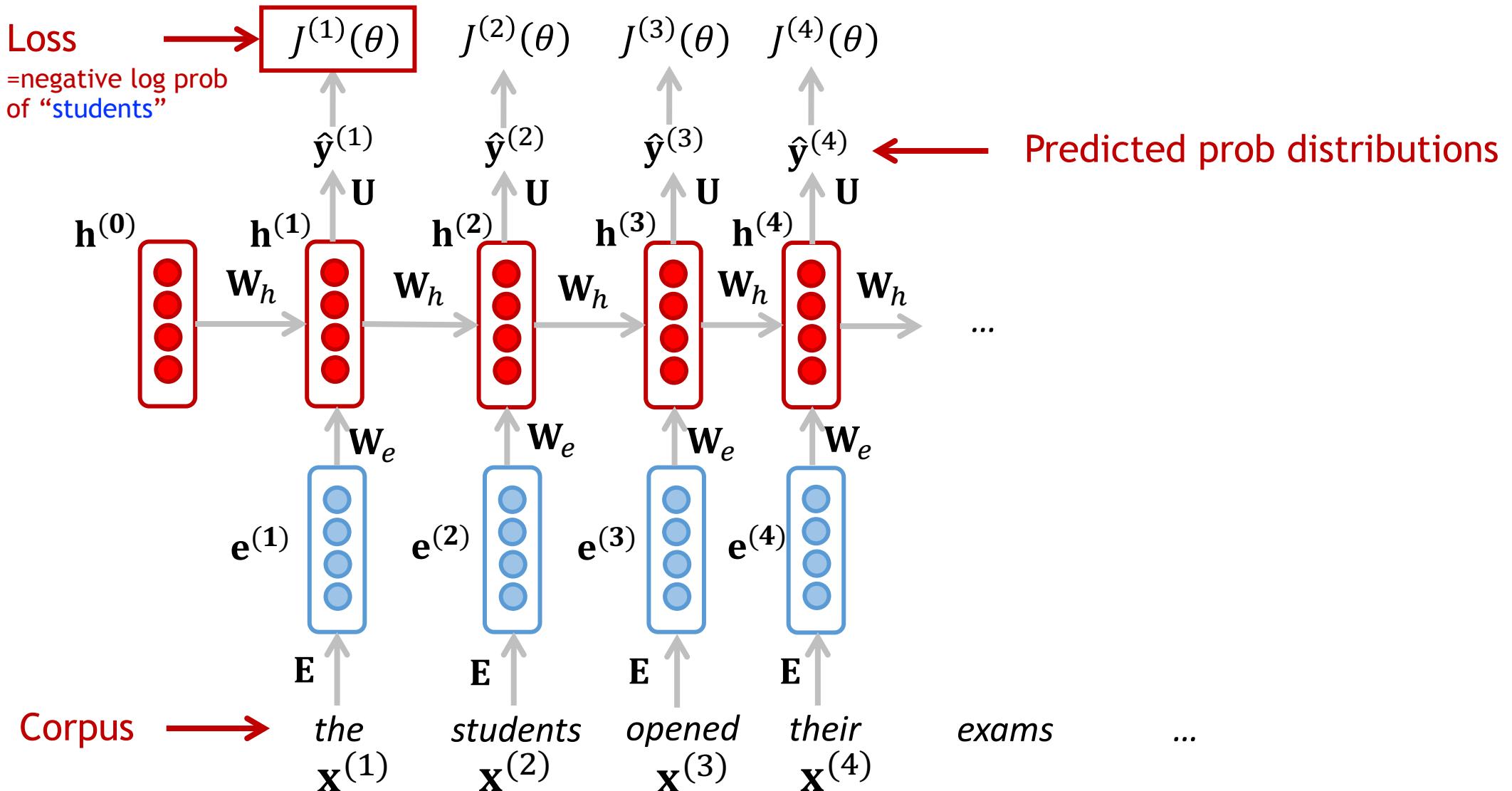
- Get a **big corpus** of text which is a sequence of words $\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \dots, \mathbf{x}^{(T)}$
- Feed into RNN-LM; compute output distribution $\hat{\mathbf{y}}^{(t)}$ for **every step t**
 - i.e. predict probability distribution of every word, given words so far
- **Loss function** on step t is **cross-entropy** between predicted probability distribution $\hat{\mathbf{y}}^{(t)}$ and the true next word $\mathbf{y}^{(t)}$ (one-hot for $\mathbf{x}^{(t+1)}$):

$$J^{(t)}(\theta) = CE(\mathbf{y}^{(t)}, \hat{\mathbf{y}}^{(t)}) = - \sum_{w \in V} \mathbf{y}_w^{(t)} \log \hat{\mathbf{y}}_w^{(t)} = - \log \hat{\mathbf{y}}_{\mathbf{x}_{t+1}}^{(t)}$$

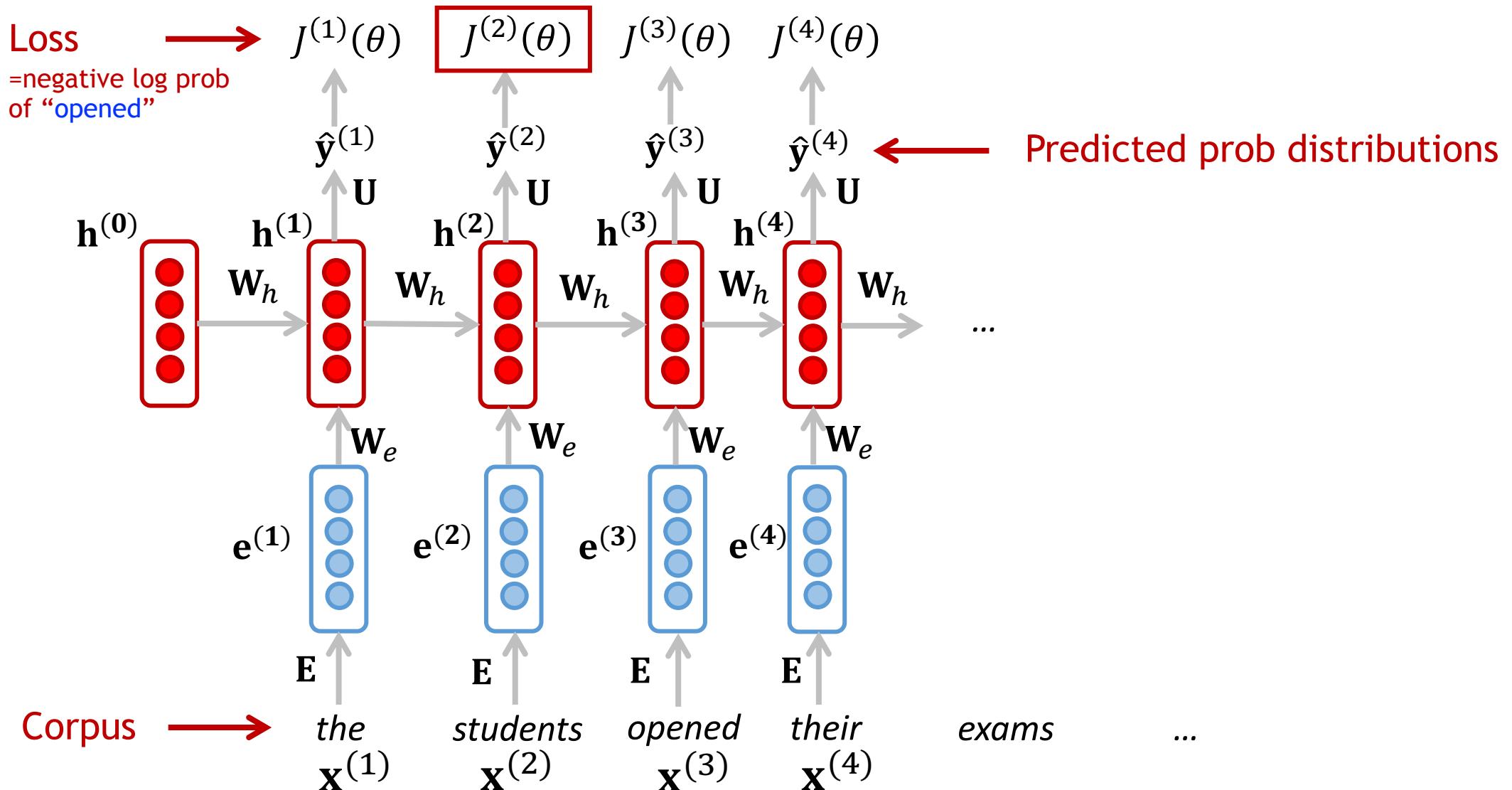
- Average this to get **overall loss** for entire training set:

$$J(\theta) = \frac{1}{T} \sum_{t=1}^T J^{(t)}(\theta) = - \frac{1}{T} \sum_{t=1}^T \log \hat{\mathbf{y}}_{\mathbf{x}_{t+1}}^{(t)}$$

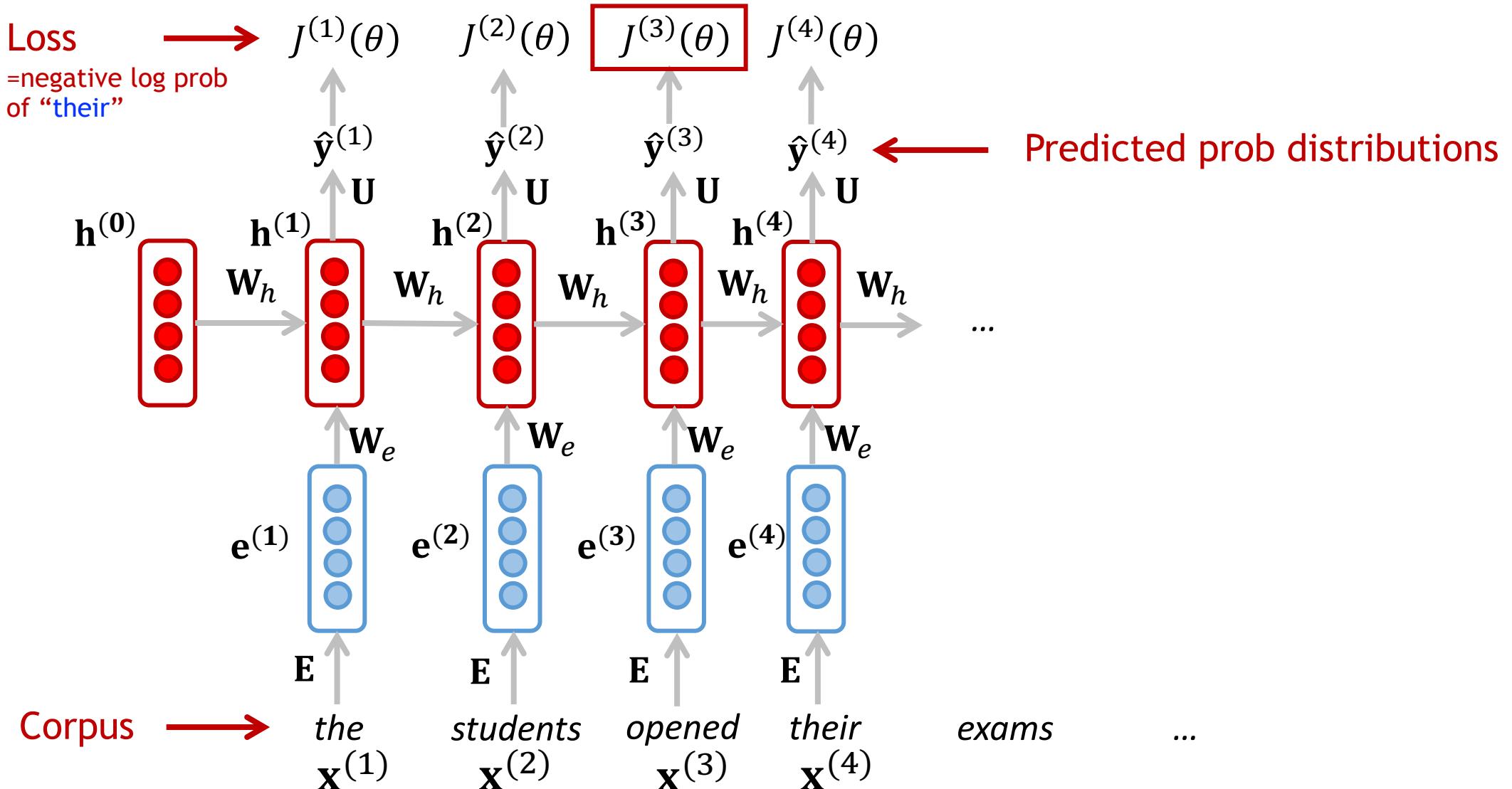
Training a RNN Language Model



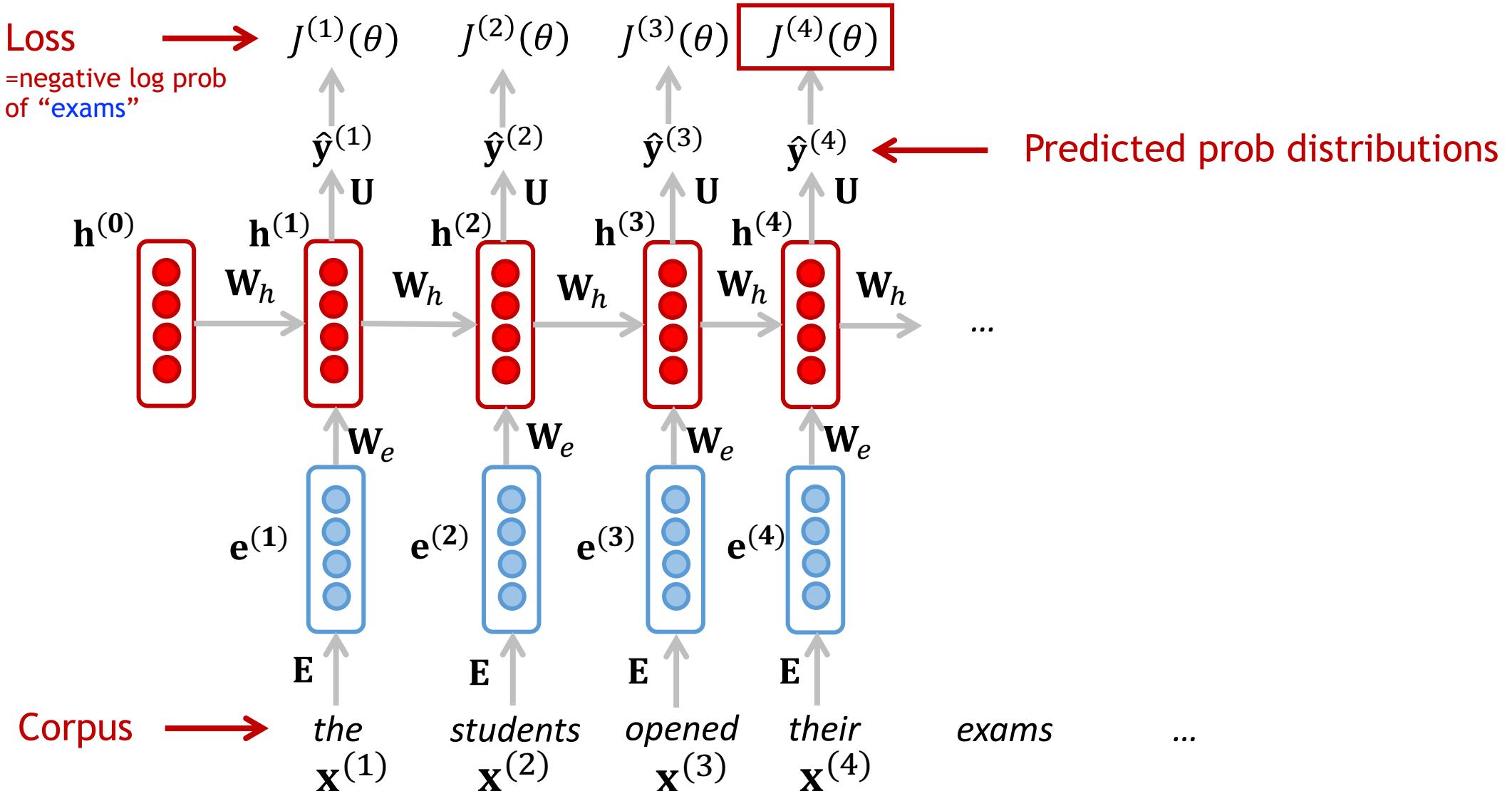
Training a RNN Language Model



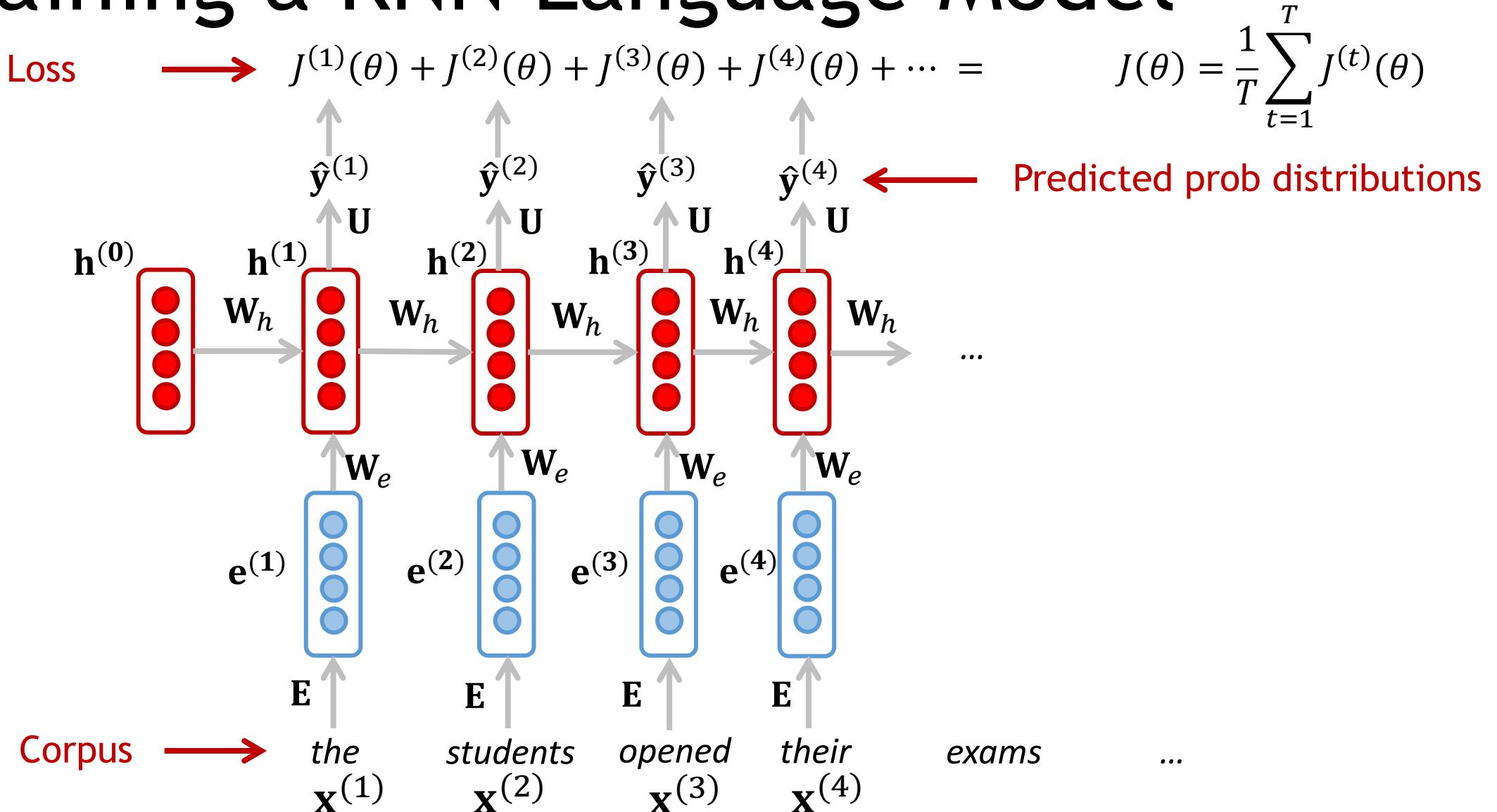
Training a RNN Language Model



Training a RNN Language Model



Training a RNN Language Model



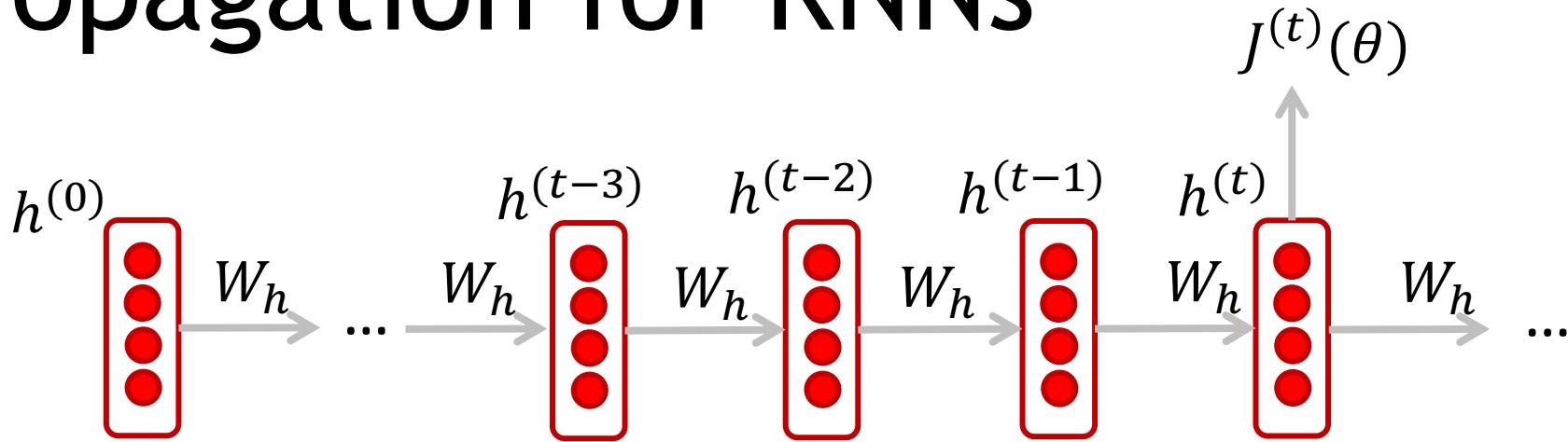
Training a RNN Language Model

- However: Computing loss and gradients across **entire corpus** $x^{(1)}, x^{(2)}, \dots, x^{(T)}$ **is too expensive**

$$J(\theta) = \frac{1}{T} \sum_{t=1}^T J^{(t)}(\theta)$$

- In practice, consider $x^{(1)}, x^{(2)}, \dots, x^{(T)}$ as a **sentence** (or a **document**)
- Recall: **Stochastic Gradient Descent** allows us to compute loss and gradients for small chunk of data, and update
- Compute loss $J(\theta)$ for a sentence (actually a batch of sentences), compute gradients and update weights. Repeat

Backpropagation for RNNs



- **Question:** What's the derivative of $J^{(t)}(\theta)$ w.r.t. **repeated** weight matrix \mathbf{W}_h ?

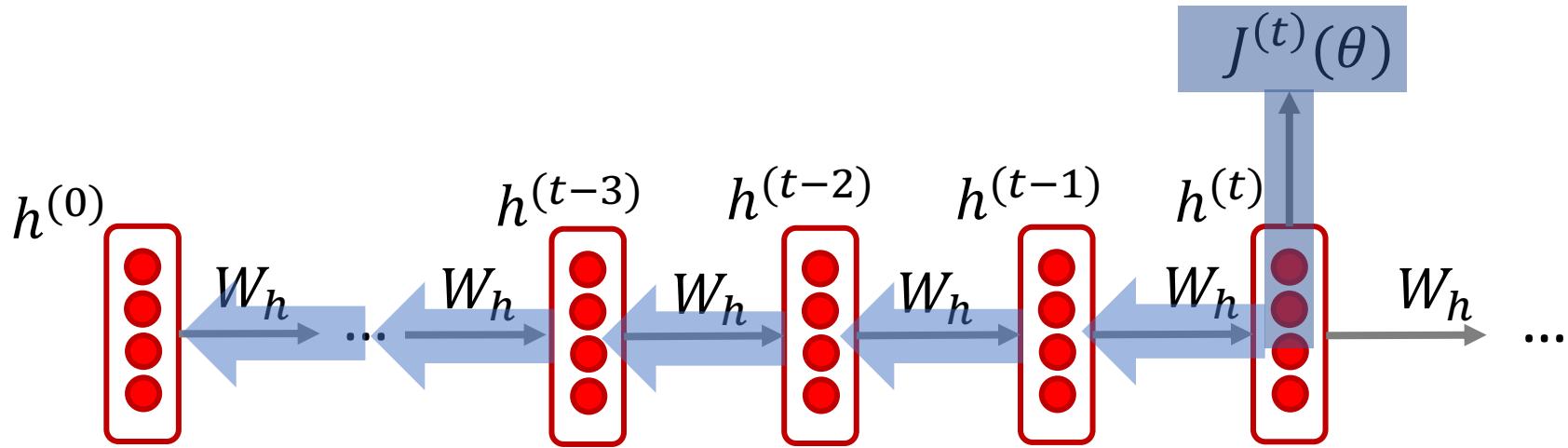
- **Answer:**
$$\frac{\partial J^{(t)}}{\partial \mathbf{W}_h} = \sum_{i=1}^t \frac{\partial J^{(t)}}{\partial \mathbf{W}_h} \Big|_{(i)}$$

“The gradient w.r.t. a repeated weight is the sum of the gradient w.r.t. each time it appears”

According to multivariate chain rule:

$$\frac{d}{dt} f(x(t), y(t)) = \frac{\partial f}{\partial x} \frac{dx}{dt} + \frac{\partial f}{\partial y} \frac{dy}{dt}$$

Backpropagation for RNNs



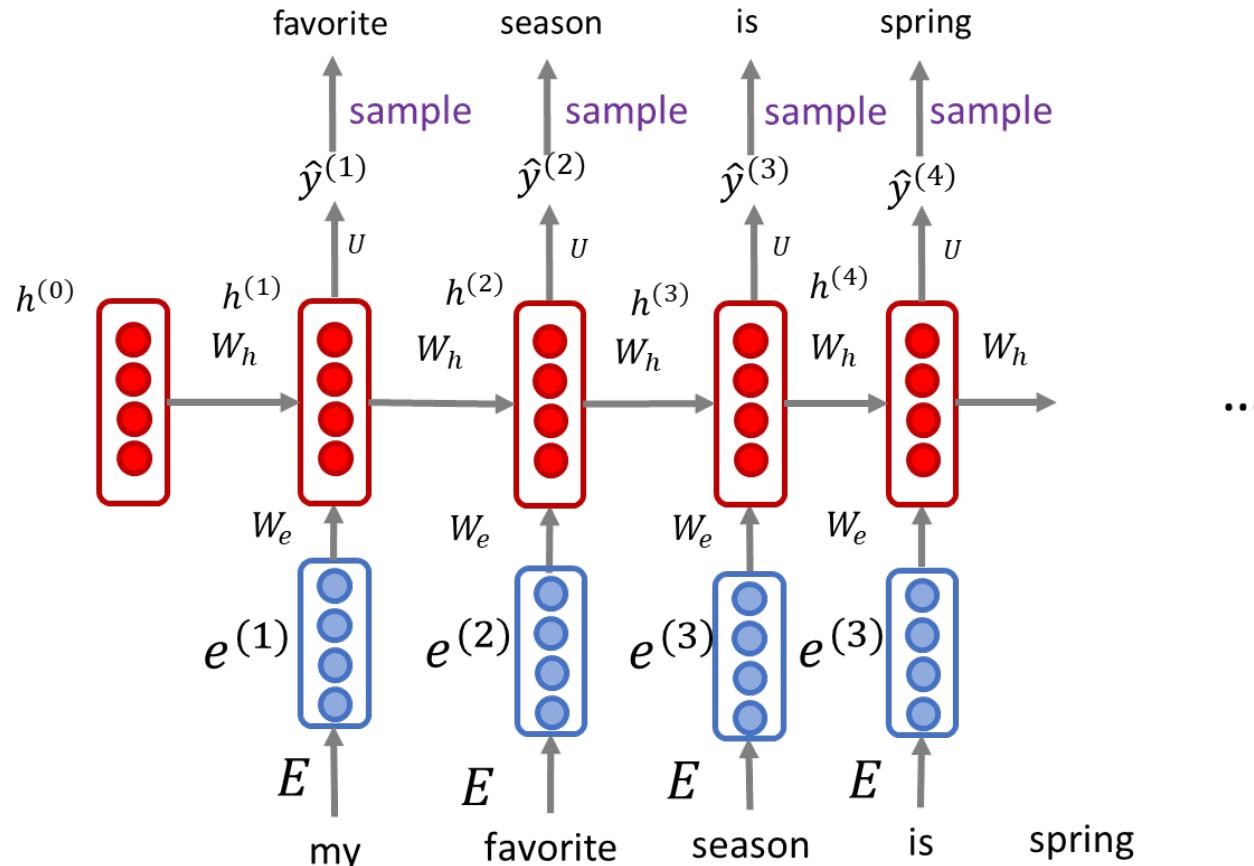
$$\frac{\partial J^{(t)}}{\partial W_h} = \boxed{\sum_{i=1}^t \frac{\partial J^{(t)}}{\partial W_h}|_{(i)}}$$

Question: How do we calculate this?

Natural Sciences and Mathematics

Generating Text with RNN LM

- Just like a n-gram Language Model, you can use a RNN Language Model to generate text by **repeated sampling**. Sampled output is next step's input



Generating Text with RNN LM

- You can train a RNN-LM on any kind of text, then generate text in that style.
- Example: RNN-LM trained on Obama speeches:



The United States will step up to the cost of a new challenges of the American people that will share the fact that we created the problem. They were attacked and so that they have to say that all the task of the final days of war that I will not be able to get this done.

<https://medium.com/@samim/obama-rnn-machine-generated-political-speeches-c8abd18a2ea0>

<https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1194/slides/cs224n-2019-lecture06-rnnlm.pdf>

Generating Text with RNN LM

- You can train a RNN-LM on any kind of text, then generate text in that style.
- Example: RNN-LM trained on recipes:

Title: CHOCOLATE RANCH BARBECUE

Categories: Game, Casseroles, Cookies, Cookies

Yield: 6 Servings

2 tb Parmesan cheese -- chopped

1 c Coconut milk

3 Eggs, beaten

Place each pasta over layers of lumps. Shape mixture into the moderate oven and simmer until firm. Serve hot in bodied fresh, mustard, orange and cheese.

Combine the cheese and salt together the dough in a large skillet; add the ingredients and stir in the chocolate and pepper.



<https://gist.github.com/nylki/1efbaa36635956d35bcc>

<https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1194/slides/cs224n-2019-lecture06-rnnlm.pdf>

Try An Online Demo

- <https://deepai.org/machine-learning-model/text-generator>
- <https://watt-ai.github.io/demos/gpt2>

Training Text Source

- ✓ GPT-2 Base Model
- Elon Musk Tweets
- Song Lyrics
- Action Movie Scripts
- Trump Tweets
- Neil deGrasse Tyson Tweets

Generated text will appear here! Use the form to configure GPT-2 and press **Generate Text** to get your own text!

Seeds the generated text with the inputted sample. (Optional: max 500 characters)

Generated Text Length

Length of the text in tokens to generate. (max: 1023)

Temperature

Controls the generated text "creativity." (the higher the temperature, the more creative)

Top k

Constrains the generated text tokens to the top k possibilities. (set to 0 to disable)

 **Generate Text!**

 **Save Image**

 **Clear Texts**

Why to Care About Language Modeling?

- Language Modeling is a **benchmark task** that helps us **measure our progress** on understanding language
- Language Modeling is a **subcomponent** of many NLP tasks, especially those involving **generating text** or **estimating the probability of text**
 - Predictive typing
 - Speech recognition
 - Handwriting recognition
 - Spelling/grammar correction
 - Authorship identification
 - Machine translation
 - Summarization
 - Dialogue
 - etc.

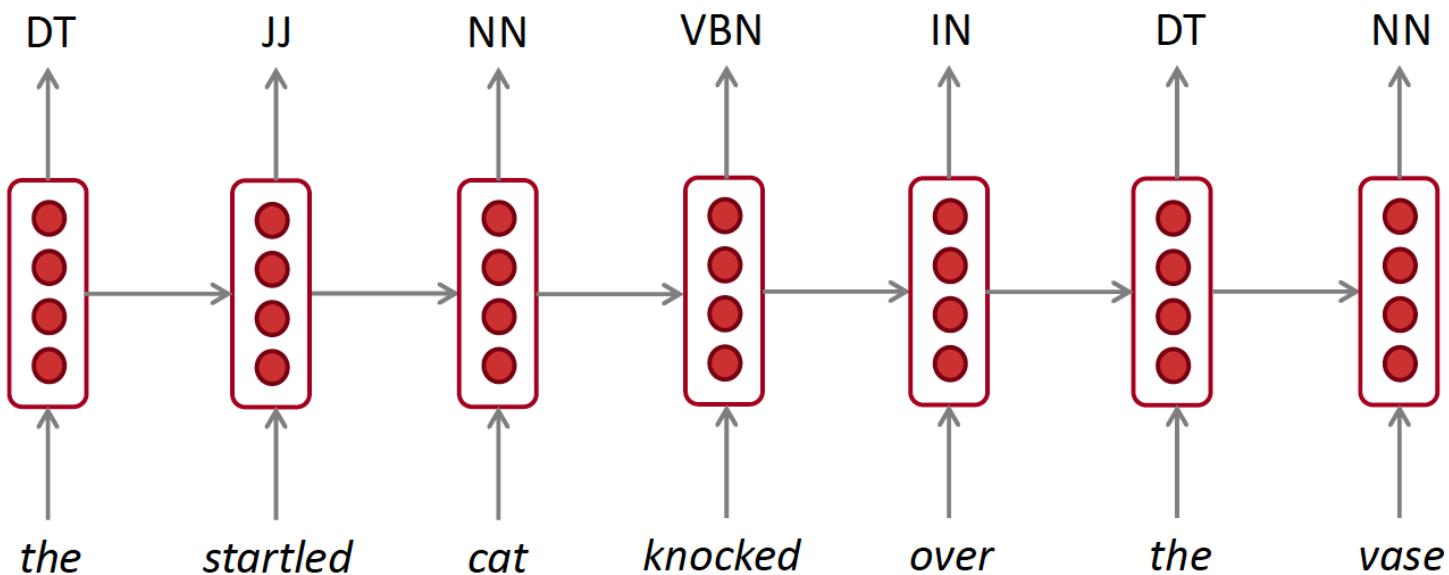
Recap

- Language Model: A system that **predicts the next word**
- Recurrent Neural Network: A family of neural networks that
 - Take **sequential input of any length**
 - Apply the **same weights on each step**
 - Can optionally produce output on each step
- Recurrent Neural Network \neq Language Model
- We've shown that RNNs are a great way to build a LM
- But RNNs are useful for much more!

RNNs can be Used for Tagging

- Part-of-speech tagging, named entity recognition

Why not tell someone ?
adverb adverb verb noun punctuation mark,
sentence closer



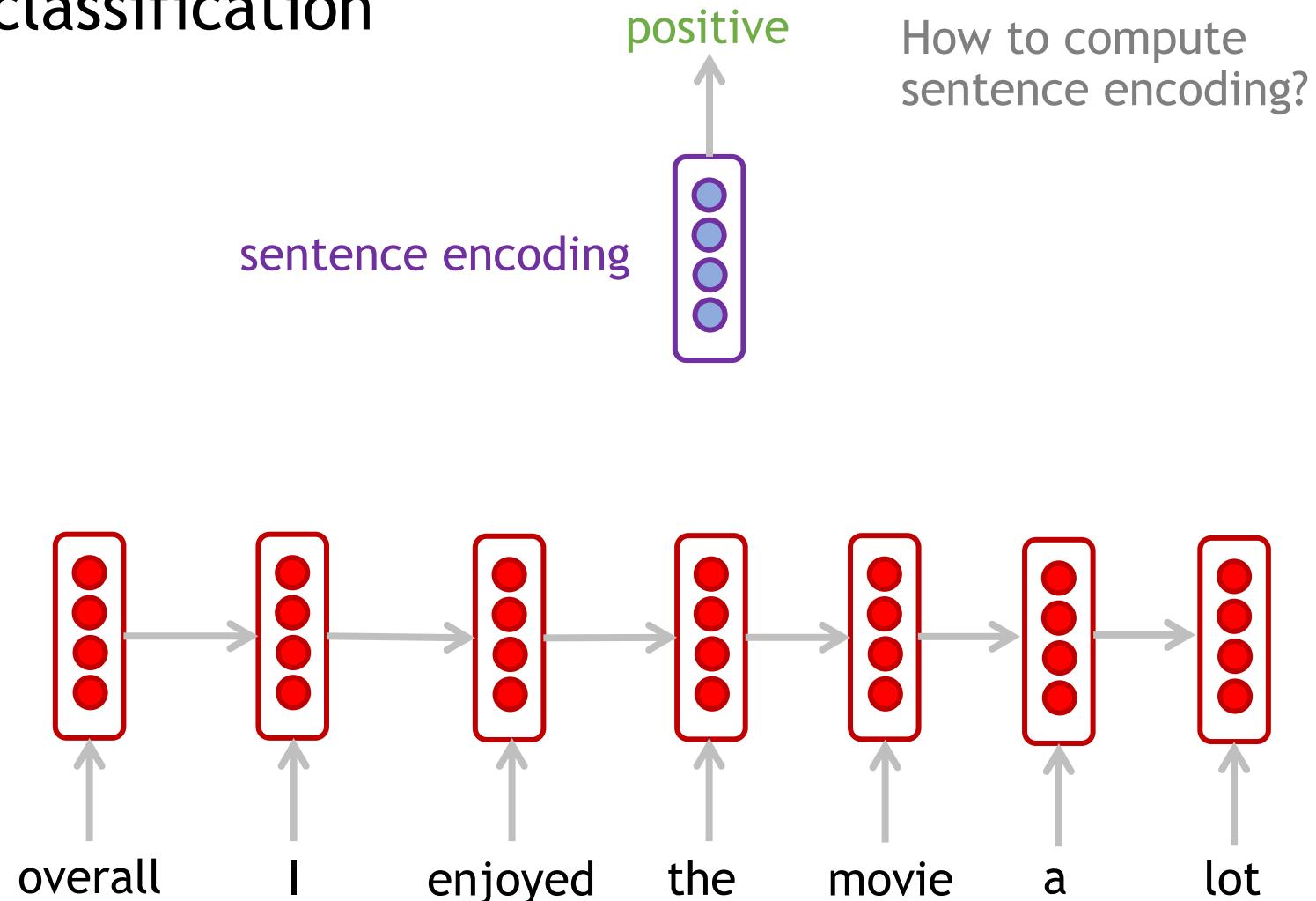
Lexical Term	Tag	Example
Noun	NN	Paris, France, Someone, Kurtis
Verb	VB	work, train, learn, run, skip
Determiner	DT	the, a
...	...	

<https://towardsdatascience.com/part-of-speech-tagging-for-beginners-3a0754b2ebba>

<https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1194/slides/cs224n-2019-lecture06-rnnlm.pdf>

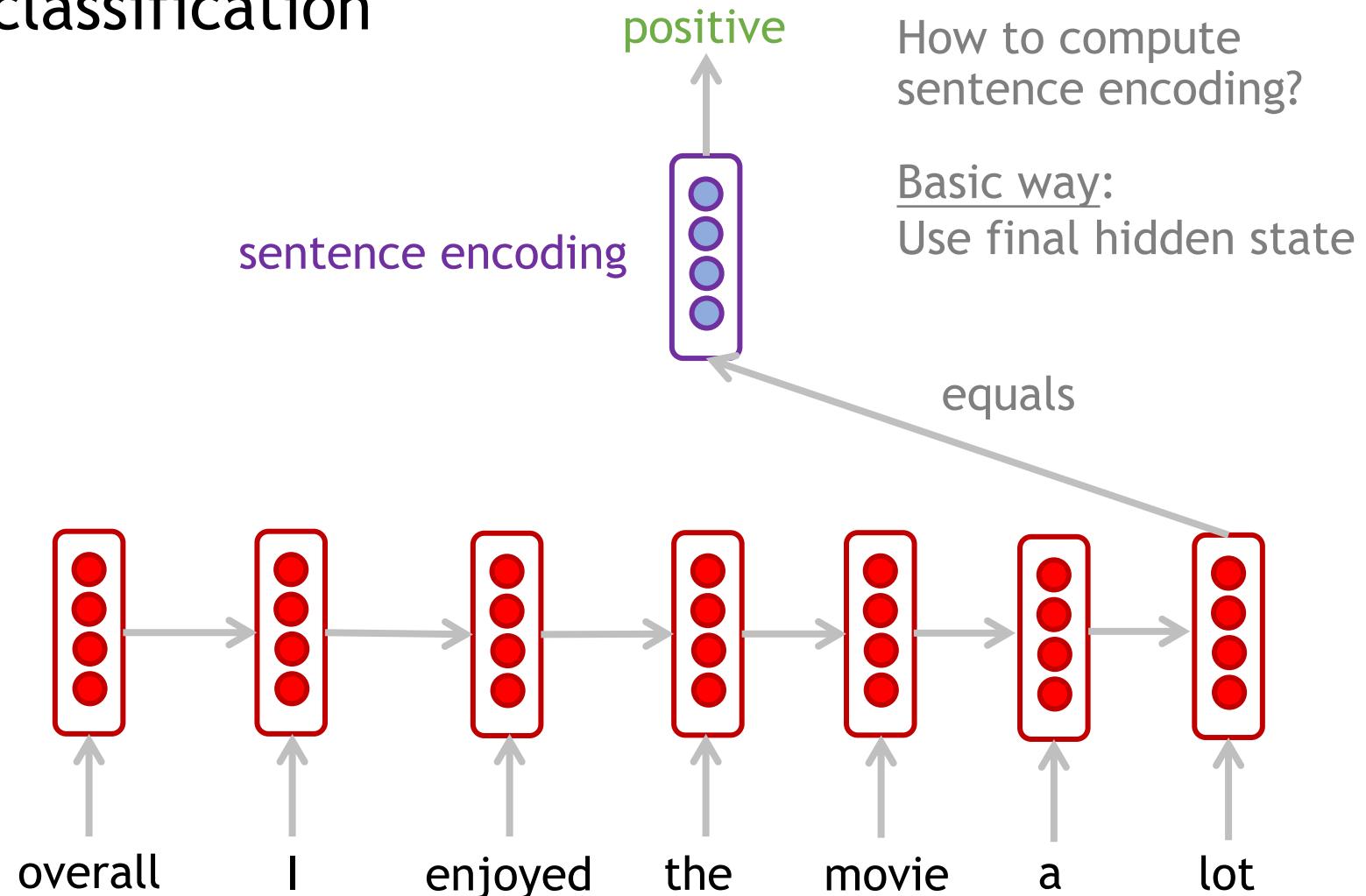
RNNs for Sentence Classification

- Sentiment classification



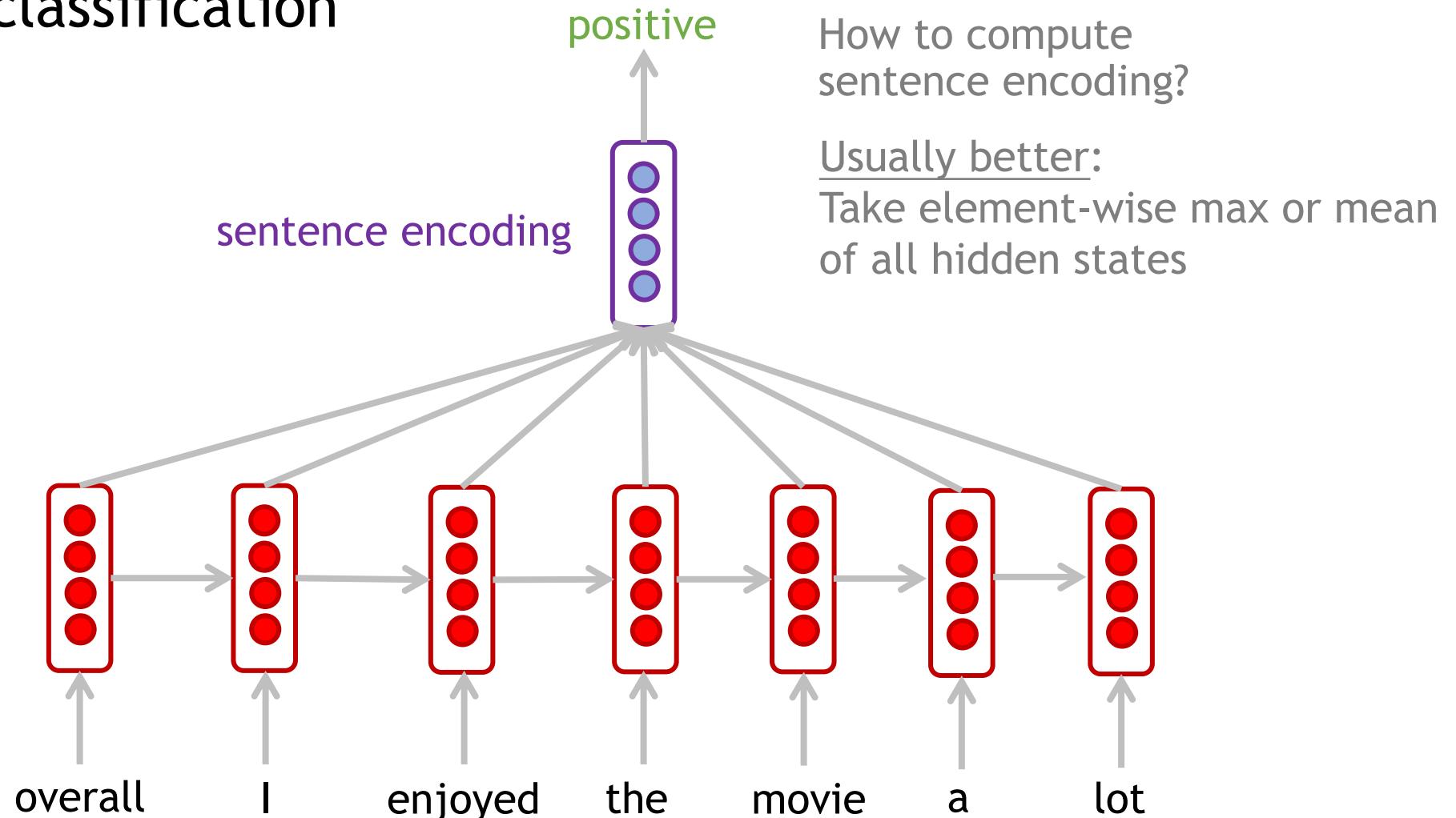
RNNs for Sentence Classification

- Sentiment classification



RNNs for Sentence Classification

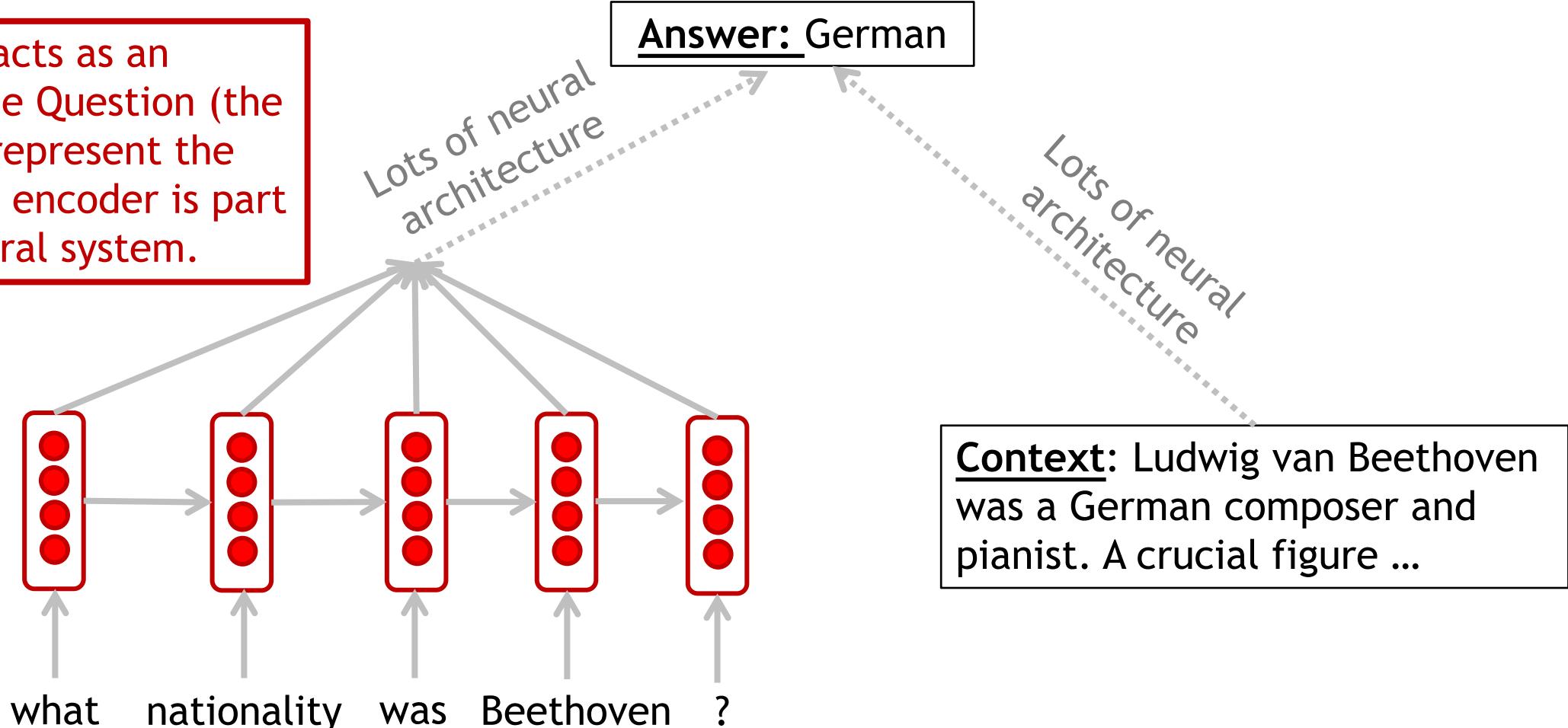
- Sentiment classification



RNNs can be Used as An Encoder Module

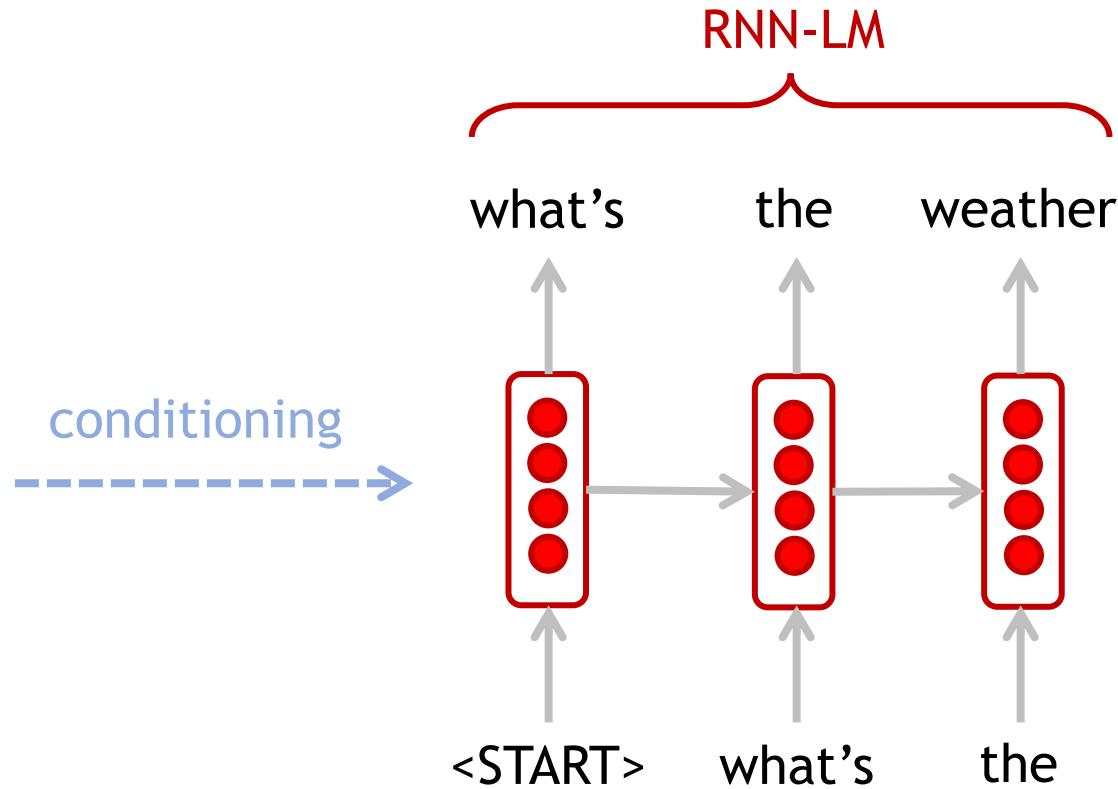
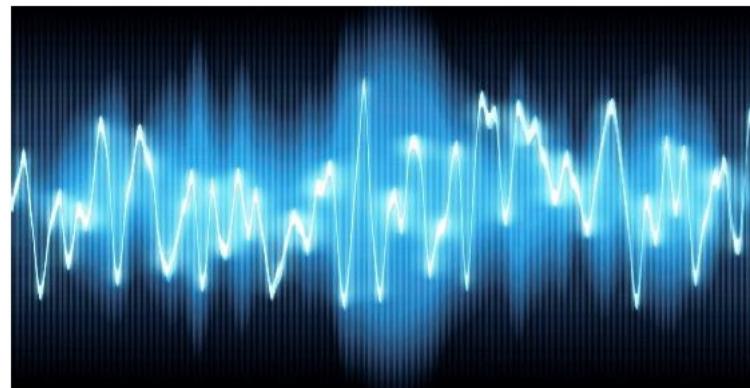
- Question answering, machine translation, many other tasks

Here the RNN acts as an **encoder** for the Question (the hidden states represent the Question). The encoder is part of a larger neural system.



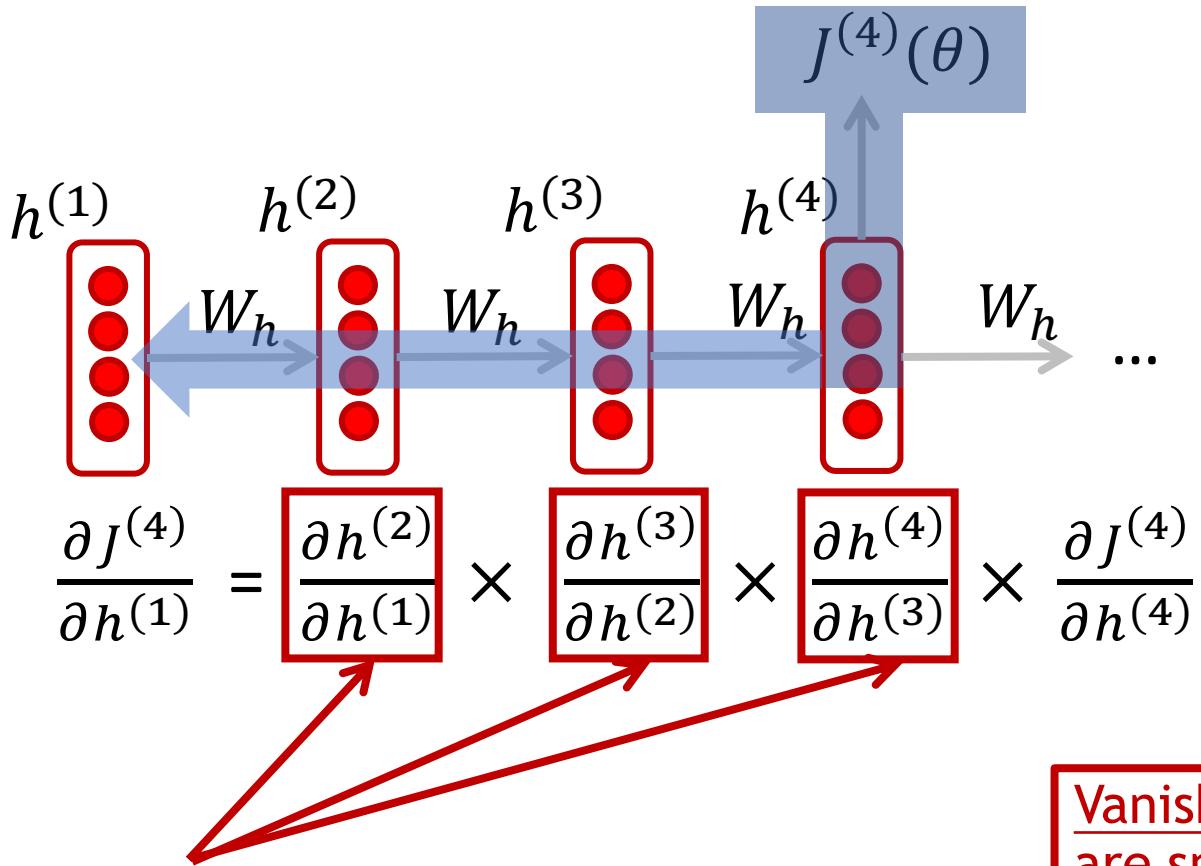
RNNs can be Used to Generate Text

- Speech recognition, machine translation, summarization



This is an example of a conditional language model.

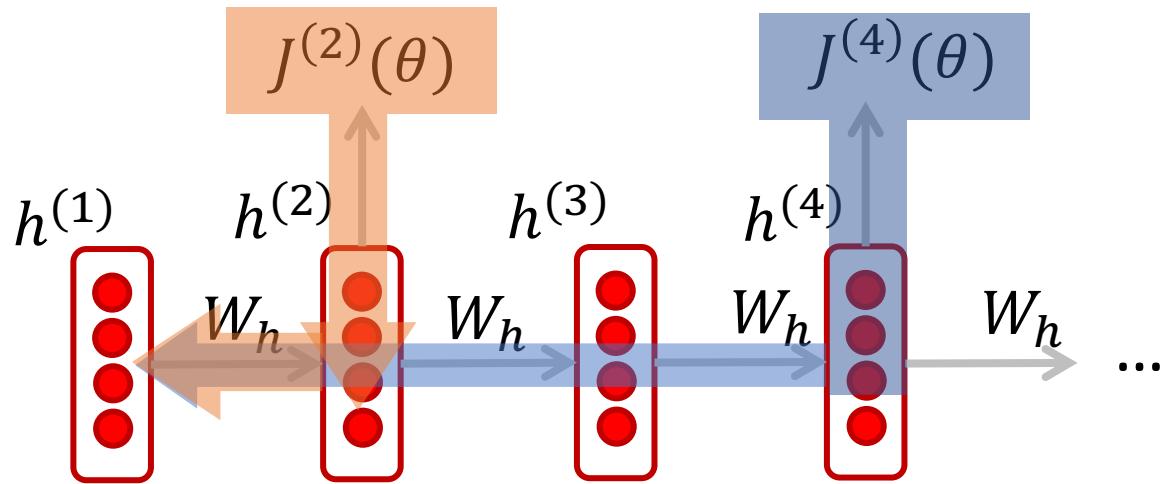
Vanishing Gradient Intuition



What happens if these are small?

Vanishing gradient problem: When these are small, the gradient signal gets smaller and smaller as it backpropagates further

Why is Vanishing Gradient A Problem?



Gradient signal from faraway is lost because it's much smaller than gradient signal from close-by.

So, model weights are updated only with respect to near effects, not long-term effects.

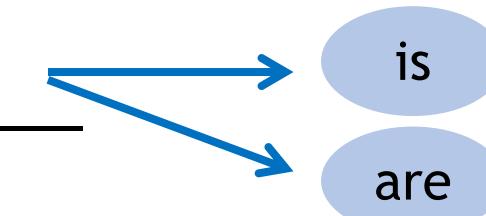
Why is Vanishing Gradient A Problem?

- Another explanation: Gradient can be viewed as a measure of the effect of the past on the future
- If the gradient becomes vanishingly small over longer distances
- (step t to step $t+n$), then we can't tell
 1. There's no dependency between step t and $t+n$ in the data
 2. We have wrong parameters to capture the true dependency between t and $t+n$

Effect of Vanishing Gradient on RNN-LM

- LM task: *When she tried to print her tickets, she found that the printer was out of toner. She went to the stationery store to buy more toner. It was very overpriced. After installing the toner into the printer, she finally printed her _____*
- To learn from this training example, the RNN-LM needs to **model the dependency** between “tickets” on the 7th step and the target word “tickets” at the end
- But if the gradient is small, the model **can't learn this dependency**
 - So, the model is **unable to predict similar long-distance dependencies** at test time

Effect of Vanishing Gradient on RNN-LM

- LM task: *The writer of the books* _____
- Correct answer: *The writer of the books is planning a sequel*
- Syntactic recency: *The writer of the books is* (correct)
- Sequential recency: *The writer of the books are* (incorrect)
- Due to vanishing gradient, RNN-LMs are better at learning from **sequential recency** than **syntactic recency**, so they make this type of error more often than we'd like [Linzen et al, 2016]

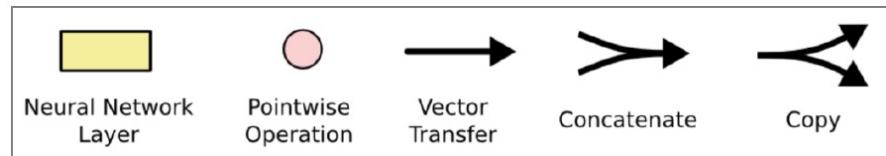
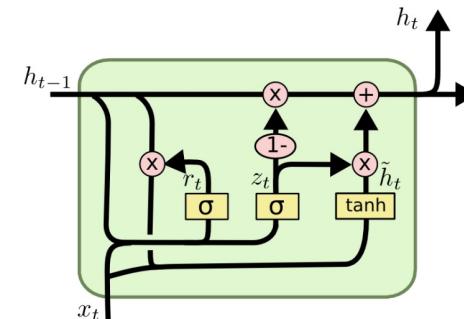
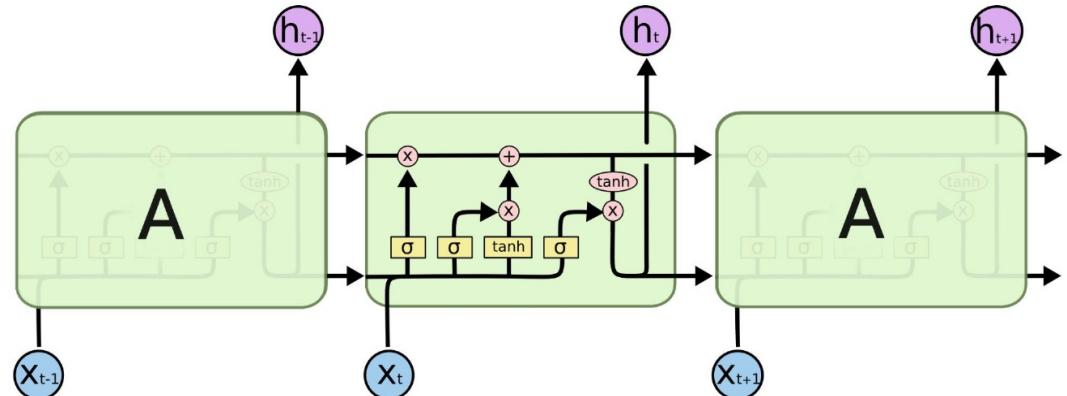
Linzen et al. "Assessing the Ability of LSTMs to Learn Syntax Sensitive Dependencies", 2016. <https://arxiv.org/pdf/1611.01368.pdf>

RNN Variants

- **Problems** with RNNs
 - Vanishing gradients

Motivates
↓

- **Fancy RNN** variants
 - LSTM (Long Short-Term Memory)
 - GRU (Gated Recurrent Unit)
 - Multi-layer
 - Bidirectional





COSC 3337
Data Science I
Section 14623

Deep Learning (contd.)

Instructor: Jingchao Ni
Fall 2024

How to Fix Vanishing Gradient Problem?

- The main problem is that *it's too difficult for the RNN to learn to preserve information over many timesteps*
- In a regular RNN, the hidden state is constantly being **rewritten**

$$\mathbf{h}^{(t)} = \sigma(\mathbf{W}_h \mathbf{h}^{(t-1)} + \mathbf{W}_x \mathbf{x}^{(t)} + \mathbf{b})$$

- How about an RNN with separate **memory** which is added to?

Long Short-Term Memory (LSTM)

- LSTM is **a type of RNN** proposed by Hochreiter and Schmidhuber in 1997 as a solution to the vanishing gradient problem
- At step t , there is a **hidden state $h^{(t)}$** and a **cell state $c^{(t)}$**
 - Both are vectors of length n
 - The cell stores **long-term information**
 - The LSTM can **read**, **erase**, and **write** information from the cell
- The selection of which information is erased/written/read is controlled by three corresponding **gates**
 - The gates are also vectors of length n
 - At each timestep, each element of the gates can be **open** (1), **closed** (0), or somewhere in-between
 - The gates are **dynamic**: their value is computed based on the current context

Long Short-Term Memory (LSTM)

- We have a sequence of inputs $\mathbf{x}^{(t)}$, and we will compute a sequence of hidden states $\mathbf{h}^{(t)}$ and cell states $\mathbf{c}^{(t)}$. At timestep t :

Forget gate: controls what is kept vs forgotten, from previous cell state

Input gate: controls what parts of the new cell content are written to cell

Output gate: controls what parts of cell are output to hidden state

New cell content: this is the new content to be written to the cell

Cell state: erase (“forget”) some content from last cell state, and write (“input”) some new cell content

Hidden state: read (“output”) some content from the cell

Sigmoid function: all gate values are between 0 and 1

$$\mathbf{f}^{(t)} = \sigma(\mathbf{W}_f \mathbf{h}^{(t-1)} + \mathbf{U}_f \mathbf{x}^{(t)} + \mathbf{b}_f)$$

$$\mathbf{i}^{(t)} = \sigma(\mathbf{W}_i \mathbf{h}^{(t-1)} + \mathbf{U}_i \mathbf{x}^{(t)} + \mathbf{b}_i)$$

$$\mathbf{o}^{(t)} = \sigma(\mathbf{W}_o \mathbf{h}^{(t-1)} + \mathbf{U}_o \mathbf{x}^{(t)} + \mathbf{b}_o)$$

$$\tilde{\mathbf{c}}^{(t)} = \tanh(\mathbf{W}_c \mathbf{h}^{(t-1)} + \mathbf{U}_c \mathbf{x}^{(t)} + \mathbf{b}_c)$$

$$\mathbf{c}^{(t)} = \mathbf{f}^{(t)} \circ \mathbf{c}^{(t-1)} + \mathbf{i}^{(t)} \circ \tilde{\mathbf{c}}^{(t)}$$

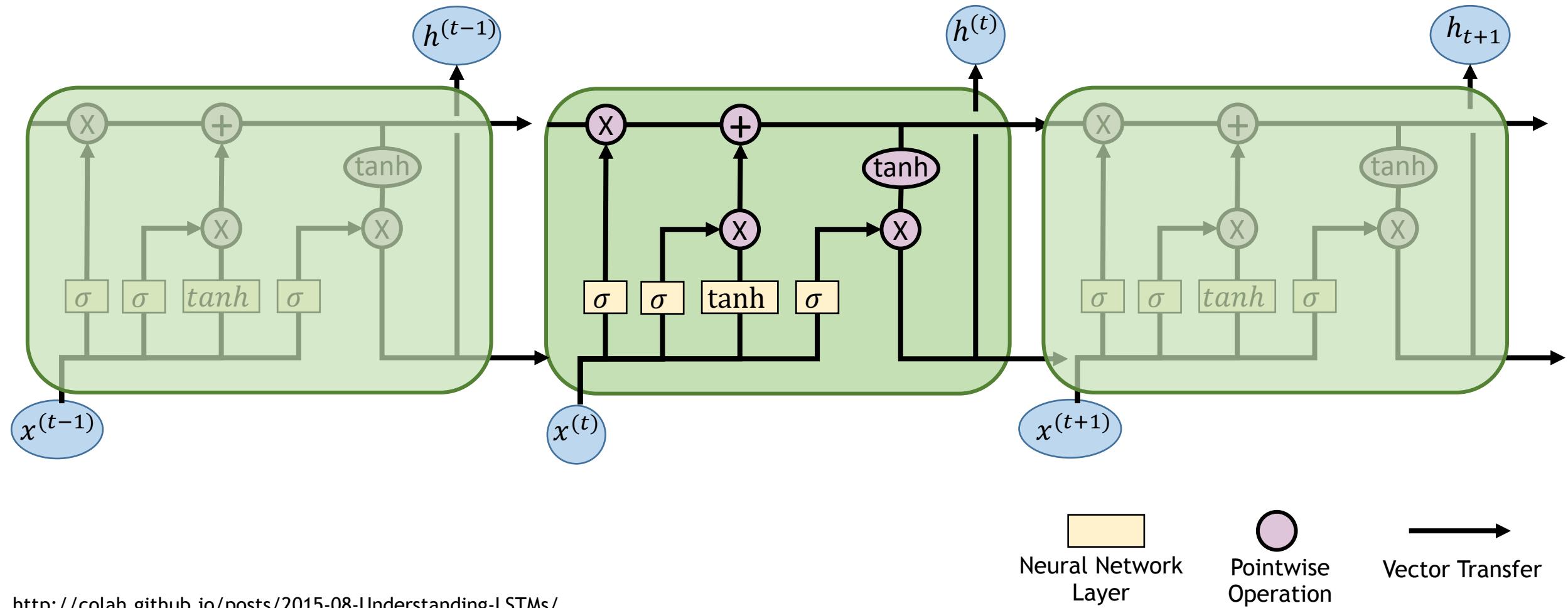
$$\mathbf{h}^{(t)} = \mathbf{o}^{(t)} \circ \tanh \mathbf{c}^{(t)}$$

Gates are applied using element-wise product

All these are vectors of same length n

Long Short-Term Memory (LSTM)

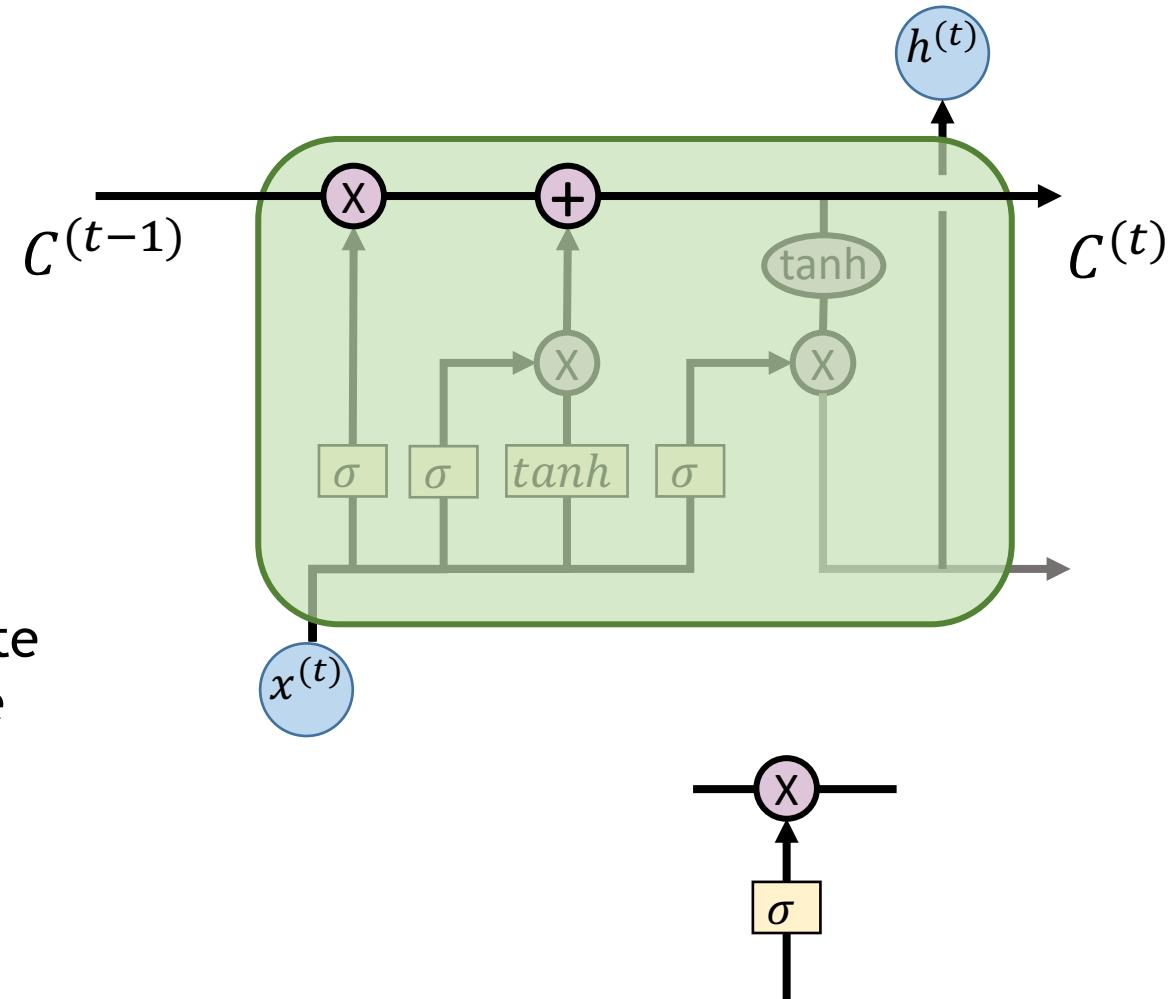
- You can think of the LSTM equations visually like this



<http://colah.github.io/posts/2015-08-Understanding-LSTMs/>

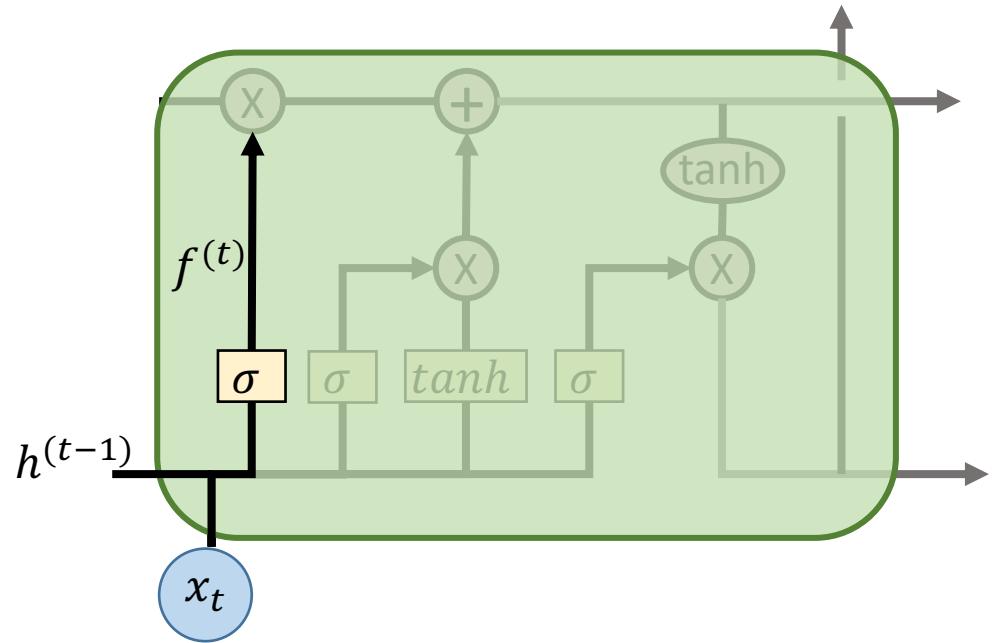
LSTM Cell State

- Top of the diagram
- Some minor linear interactions
 - Easy for information to flow along it unchanged
 - Capture long range dependency
- Gates
 - Remove or add information to the cell state
 - A sigmoid neural net layer and a pointwise multiplication operation
 - 0 means “let nothing go through”
 - 1 means “let everything go through”



Step-by-Step Walk Through

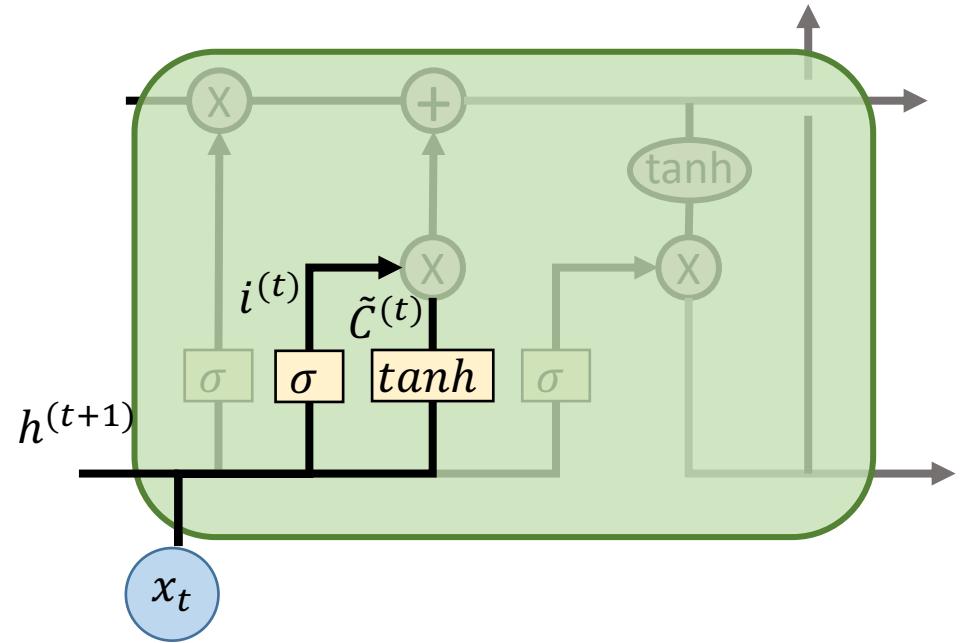
- Forget gate layer
 - Decides what information we're going to throw away from the cell state
- Input: $h^{(t-1)}, x^{(t)}$
- Output: $\in [0, 1]$
 - 1 represents "completely keep this"
 - 0 represents "completely get rid of this"



$$f^{(t)} = \sigma(\mathbf{W}_f h^{(t-1)} + \mathbf{U}_f x^{(t)} + \mathbf{b}_f)$$

Step-by-Step Walk Through

- Decide what new information we're going to store in the cell state
- Input gate layer
 - Decides which values we'll update
- A tanh layer creates a vector of new candidate values, $\tilde{c}^{(t)}$, that could be added to the cell state

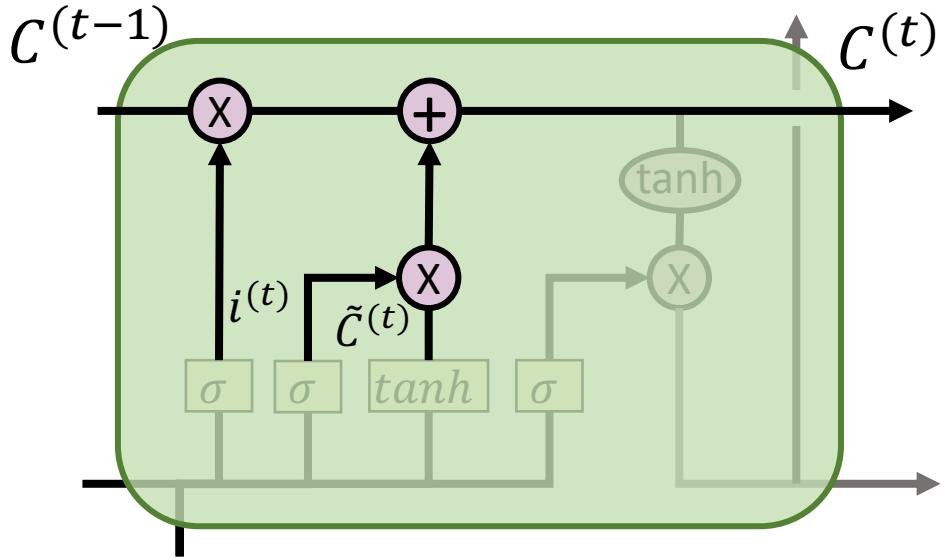


$$i^{(t)} = \sigma(\mathbf{W}_i \mathbf{h}^{(t-1)} + \mathbf{U}_i \mathbf{x}^{(t)} + \mathbf{b}_i)$$

$$\tilde{\mathbf{c}}^{(t)} = \tanh(\mathbf{W}_c \mathbf{h}^{(t-1)} + \mathbf{U}_c \mathbf{x}^{(t)} + \mathbf{b}_c)$$

Step-by-Step Walk Through

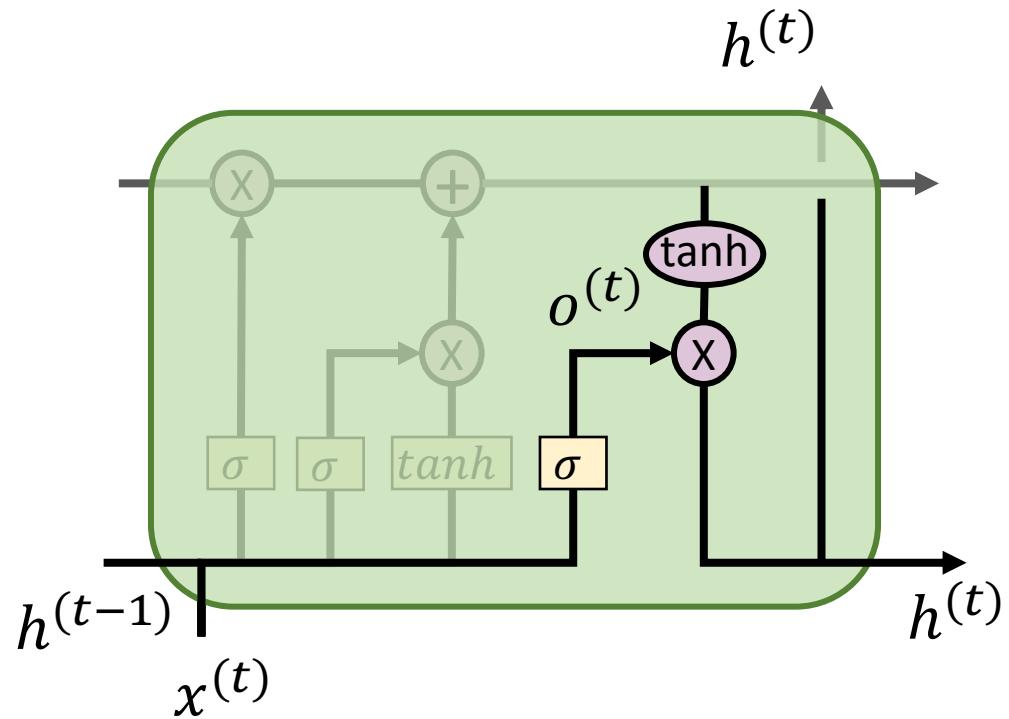
- Update the old cell state, $c^{(t-1)}$, into the new cell state $c^{(t)}$
 - Multiply the old state by $f^{(t)}$, forgetting the things we decided to forget earlier
 - Add $i^{(t)} \circ \tilde{c}^{(t)}$. This is the new candidate values, scaled by how much we decided to update each state value



$$c^{(t)} = f^{(t)} \circ c^{(t-1)} + i^{(t)} \circ \tilde{c}^{(t)}$$

Step-by-Step Walk Through

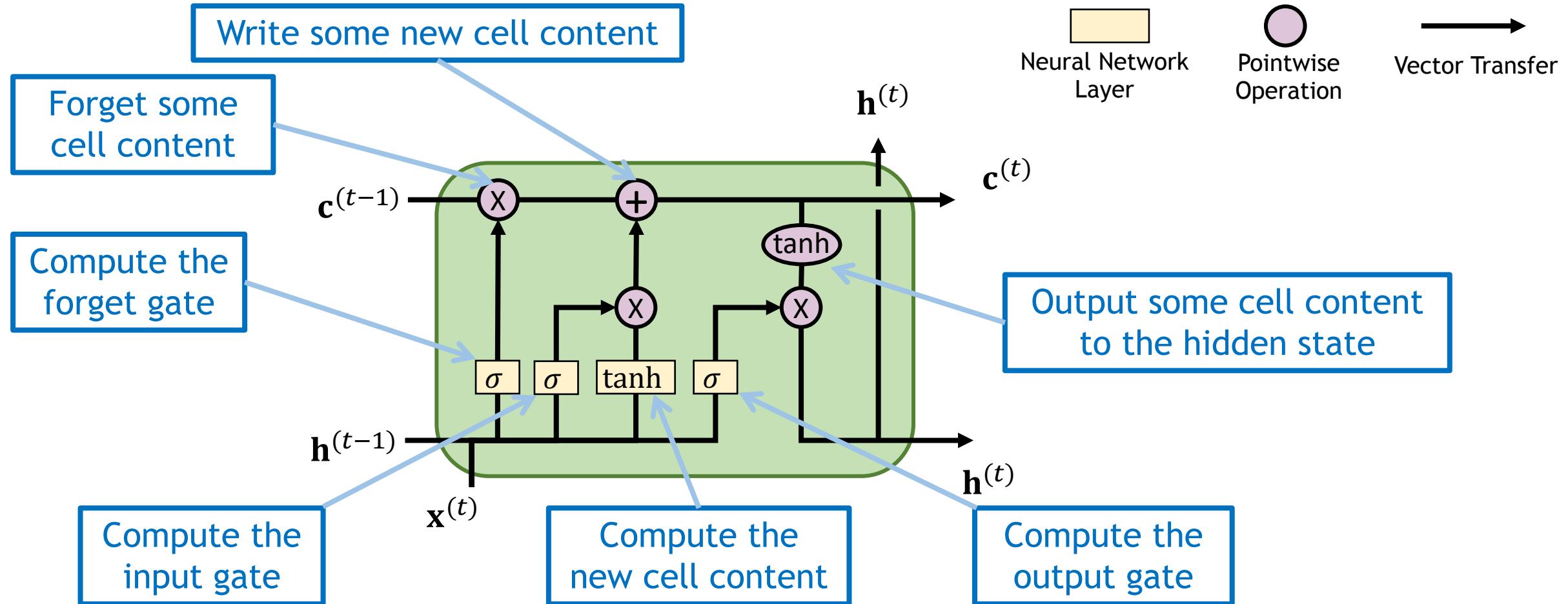
- Decide what we're going to output
 - Based on cell state
 - With a gate
- A sigmoid layer decides what parts of the cell state we're going to output
- We put the cell state through tanh to push the values to be within [-1, 1]
- Multiply it by the output of the sigmoid gate, so that we only output the parts we decided to



$$\mathbf{o}^{(t)} = \sigma(\mathbf{W}_o \mathbf{h}^{(t-1)} + \mathbf{U}_o \mathbf{x}^{(t)} + \mathbf{b}_o)$$

$$\mathbf{h}^{(t)} = \mathbf{o}^{(t)} \circ \tanh \mathbf{c}^{(t)}$$

Long Short-Term Memory (LSTM)



How Does LSTM Solve Vanishing Gradients?

- The LSTM architecture makes it **easier** for the RNN to **preserve information over many timesteps**
 - e.g., if the forget gate is set to 1 for a cell dimension and the input gate set to 0, then the information of that cell is preserved indefinitely
 - In contrast, it's harder for a vanilla RNN to learn a recurrent weight matrix W_h that preserves info in the hidden state
 - In practice, you get about 100 timesteps rather than about 10
- LSTM doesn't *guarantee* that there is no vanishing gradient, but it does provide an easier way for the model to learn long-distance dependencies

LSTMs: Real-World Success

- In 2013-2015, LSTMs started achieving state-of-the-art results
 - Successful tasks include handwriting recognition, speech recognition, machine translation, parsing, and image captioning
 - LSTM became the dominant approach for most NLP tasks
- Now, other approaches (e.g., Transformers) have become dominant for many tasks
 - For example, in WMT (a Machine Translation conference + competition):
 - In WMT 2014, there were 0 neural machine translation systems
 - In WMT 2016, the summary report contains “RNN” 44 times
 - In WMT 2019: “RNN” 7 times, “Transformer” 105 times

"Findings of the 2016 Conference on Machine Translation (WMT16)", Bojar et al. 2016, <http://www.statmt.org/wmt16/pdf/W16-2301.pdf>

"Findings of the 2018 Conference on Machine Translation (WMT18)", Bojar et al. 2018, <http://www.statmt.org/wmt18/pdf/WMT028.pdf>

"Findings of the 2019 Conference on Machine Translation (WMT19)", Barrault et al. 2019, <http://www.statmt.org/wmt18/pdf/WMT028.pdf>

Gated Recurrent Units (GRU)

- Proposed by Cho et al. in 2014 as a **simpler alternative** to the LSTM
- On each timestep t we have input and hidden state (no cell state)

Update gate: controls what parts of hidden state are updated vs preserved

$$\mathbf{u}^{(t)} = \sigma(\mathbf{W}_u \mathbf{h}^{(t-1)} + \mathbf{U}_u \mathbf{x}^{(t)} + \mathbf{b}_u)$$

Reset gate: controls what parts of previous hidden state are used to compute new content

$$\mathbf{r}^{(t)} = \sigma(\mathbf{W}_i \mathbf{h}^{(t-1)} + \mathbf{U}_r \mathbf{x}^{(t)} + \mathbf{b}_r)$$

New hidden state content: reset gate selects useful parts of prev hidden state. Use this and current input to compute new hidden content

$$\tilde{\mathbf{h}}^{(t)} = \tanh(\mathbf{W}_h (\mathbf{r}^{(t)} \circ \mathbf{h}^{(t-1)}) + \mathbf{U}_h \mathbf{x}^{(t)} + \mathbf{b}_h)$$

Hidden state: update gate simultaneously controls what is kept from previous hidden state, and what is updated to new hidden state content

$$\mathbf{h}^{(t)} = (1 - \mathbf{u}^{(t)}) \circ \mathbf{h}^{(t-1)} + \mathbf{u}^{(t)} \circ \tilde{\mathbf{h}}^{(t)}$$

How does this solve vanishing gradient?

Like LSTM, GRU makes it easier to retain info long-term (e.g., by setting update gate to 0)

LSTM vs GRU

- Researchers have proposed many gated RNN variants, but LSTM and GRU are the most widely-used
- The biggest difference is that **GRU is quicker to compute** and has fewer parameters
- There is no conclusive evidence that one consistently performs better than the other
- **LSTM is a good default choice** (especially if your data has particularly long dependencies, or you have lots of training data)
- **Rule of thumb**: start with LSTM, but switch to GRU if you want something more efficient

Is Vanishing Gradient Just A RNN Problem?

- No! It can be a problem for all neural architectures (including **feed-forward** and **convolutional**), especially **deep** ones
 - Due to chain rule / choice of nonlinearity function, gradient can become vanishingly small as it backpropagates
 - Thus, lower layers are learned very slowly (hard to train)
 - Solution: lots of new deep feedforward/convolutional architectures **add more direct connections** (thus allowing the gradient to flow)

For example:

- **Residual connections** aka “ResNet”
- Also known as **skip-connections**
- The **identity connection preserves information** by default
- This makes **deep networks** much **easier to train**

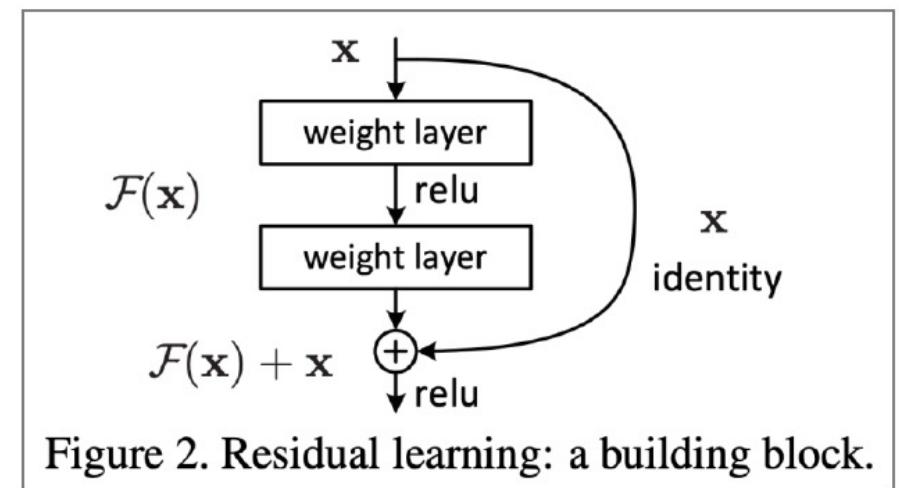


Figure 2. Residual learning: a building block.

“Deep Residual Learning for Image Recognition”, He et al, 2015. <https://arxiv.org/pdf/1512.03385.pdf>

Is Vanishing Gradient Just A RNN Problem?

- No! It can be a problem for all neural architectures (including **feed-forward** and **convolutional**), especially **deep** ones
 - Due to chain rule / choice of nonlinearity function, gradient can become vanishingly small as it backpropagates
 - Thus, lower layers are learned very slowly (hard to train)
 - Solution: lots of new deep feedforward/convolutional architectures **add more direct connections** (thus allowing the gradient to flow)

For example:

- **Dense connections** aka “DenseNet”
- Directly connect every layer to all other layers

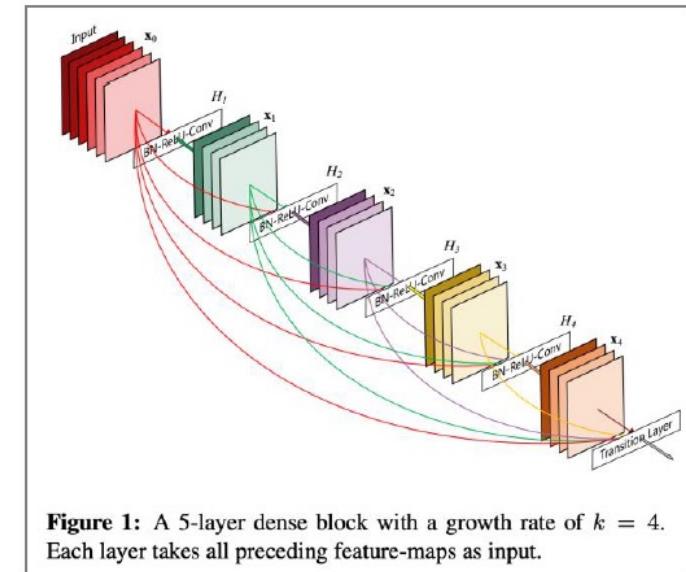


Figure 1: A 5-layer dense block with a growth rate of $k = 4$. Each layer takes all preceding feature-maps as input.

“Densely Connected Convolutional Networks”, Huang et al, 2017. <https://arxiv.org/pdf/1608.06993.pdf>

Is Vanishing Gradient Just A RNN Problem?

- No! It can be a problem for all neural architectures (including **feed-forward** and **convolutional**), especially **deep** ones
 - Due to chain rule / choice of nonlinearity function, gradient can become vanishingly small as it backpropagates
 - Thus, lower layers are learned very slowly (hard to train)
 - Solution: lots of new deep feedforward/convolutional architectures **add more direct connections** (thus allowing the gradient to flow)
- Conclusion: Though vanishing gradients are a general problem, **RNNs are particularly unstable** due to the repeated multiplication by the **same** weight matrix [Bengio et al, 1994]

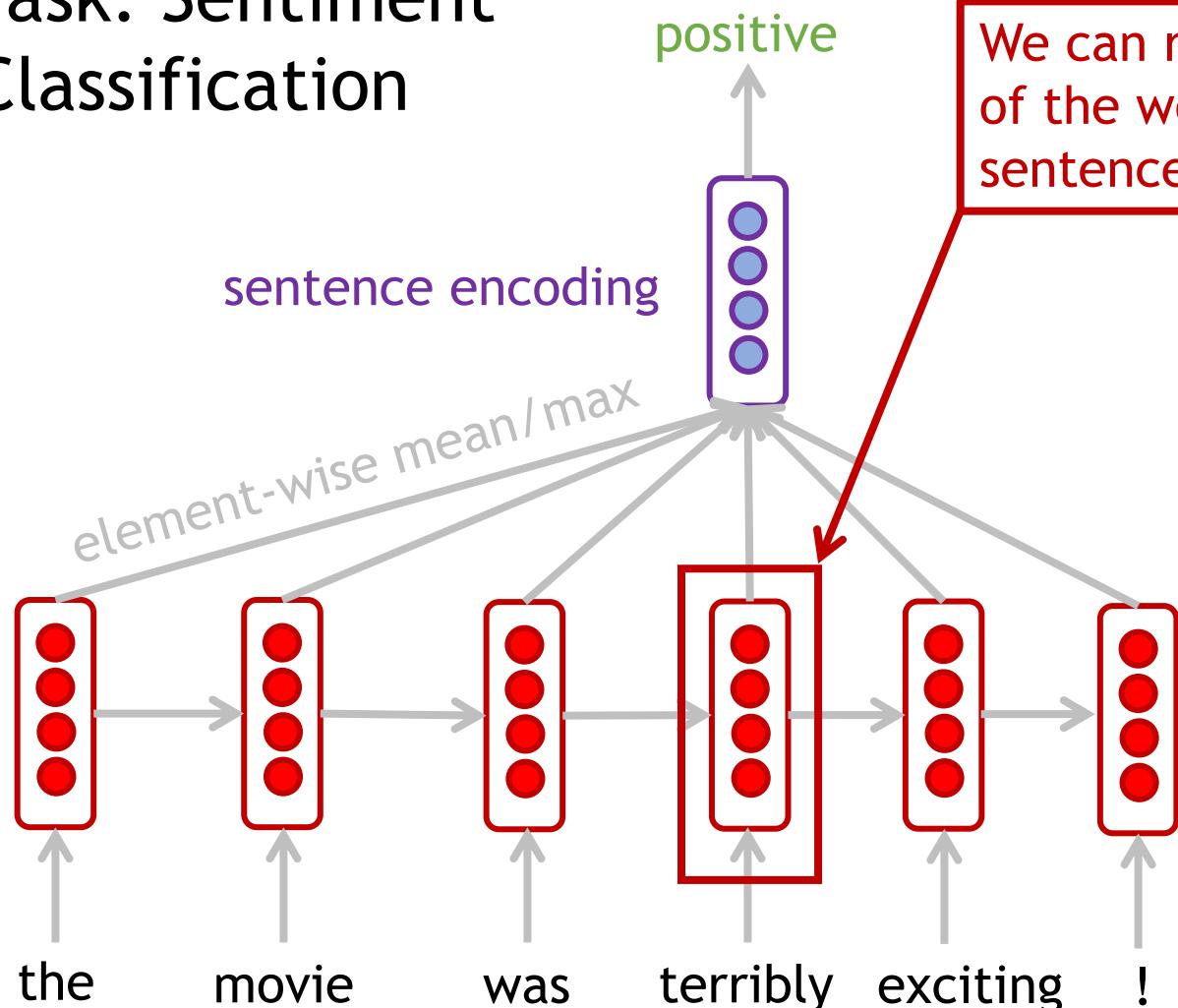
“Learning Long-Term Dependencies with Gradient Descent is Difficult”, Bengio et al. 1994, <http://ai.dinfo.unifi.it/paolo//ps/tnn-94-gradient.pdf>

Recap

- So far, we've learnt
 - **Vanishing gradient problem**: what it is, why it happens, and why it's bad for RNNs
 - **LSTMs and GRUs**: more complicated RNNs that use **gates** to control information flow; they are more resilient to vanishing gradients
 - Next
 - **Bidirectional RNNs**
 - **Multi-layer RNNs**
- } Both of these are pretty simple

Bidirectional RNNs: Motivation

- Task: Sentiment Classification



We can regard this hidden state as a representation of the word “terribly” in the context of this sentence. We call this a contextual representation.

These contextual representations only contain information about the left context (e.g. “the movie was”).

What about right context?

In this example, “exciting” is in the right context and this modifies the meaning of “terribly” (from negative to positive)

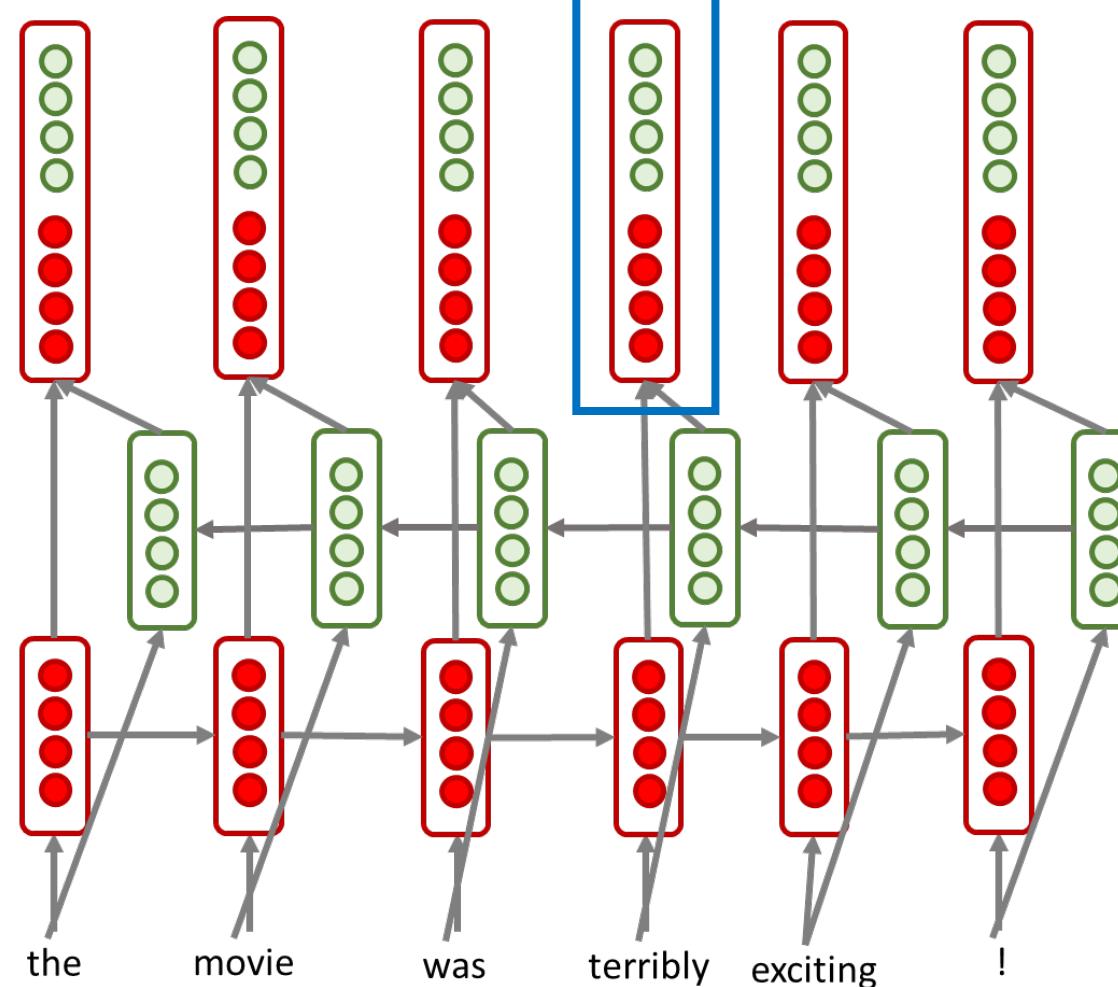
Bidirectional RNNs

This contextual representation of “terribly” has both left and right context!

Concatenated
hidden states

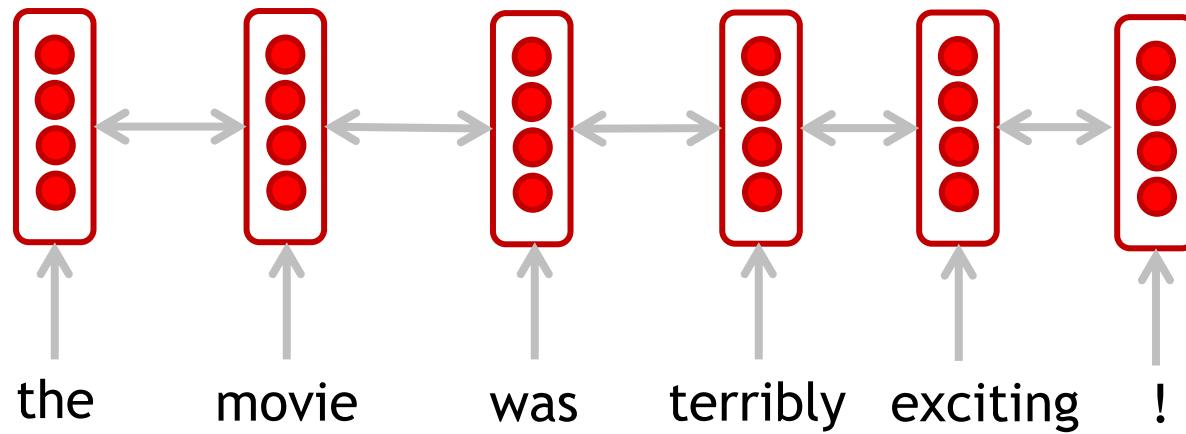
Backward RNN

Forward RNN



Bidirectional RNNs: Simplified Diagram

- The two-way arrows indicate bidirectionality and the depicted hidden states are assumed to be the concatenated forwards+backwards states



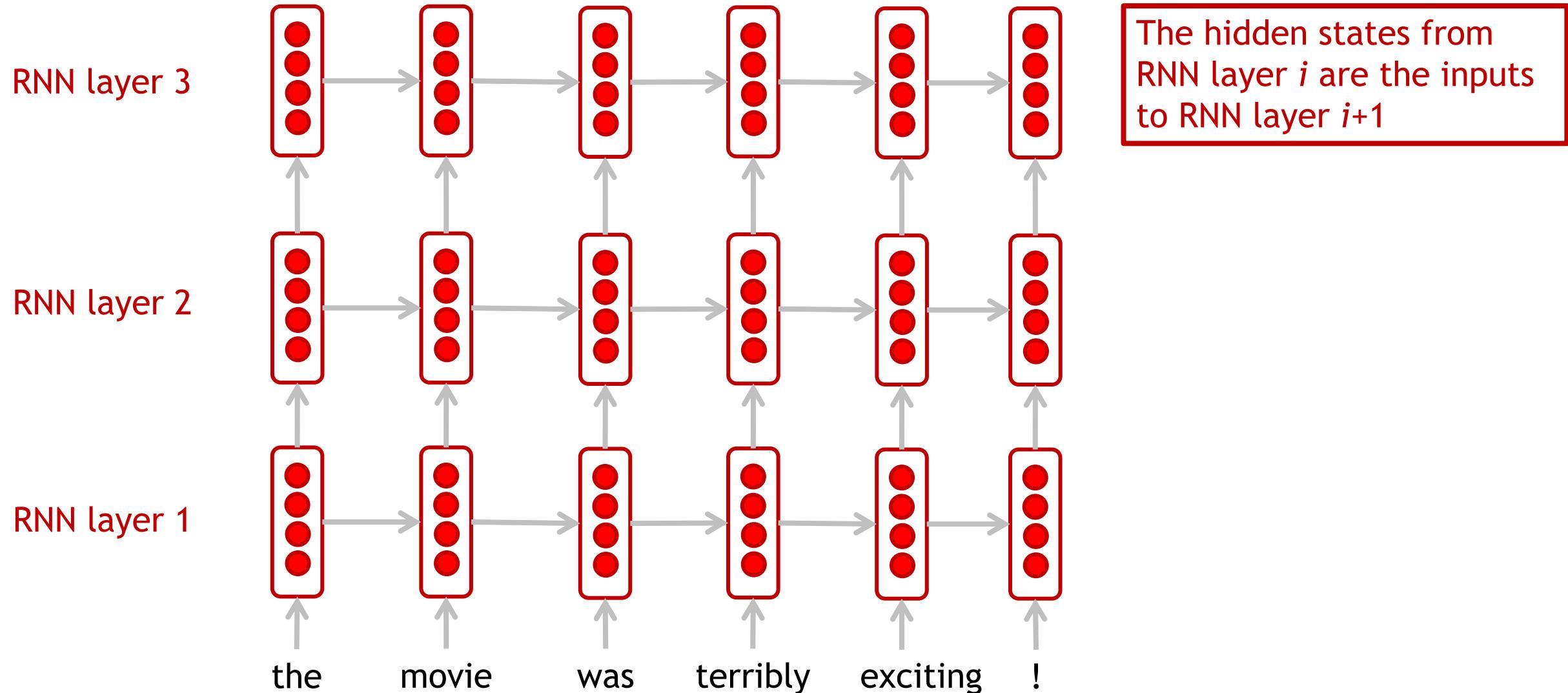
Bidirectional RNNs

- Note: bidirectional RNNs are only applicable if you have access to the **entire input sequence**
 - They are **not** applicable to Language Modeling, because in LM you only have left context available
- If you do have entire input sequence (e.g., any kind of encoding), **bidirectionality is powerful** (you should use it by default)
- For example, **BERT** (Bidirectional Encoder Representations from Transformers) is a powerful pretrained contextual representation system **built on bidirectionality**

Multi-Layer RNNs

- RNNs are already “deep” on one dimension (they unroll over many timesteps)
- We can also make them “deep” in another dimension by **applying multiple RNNs**: this is a multi-layer RNN
- This allows the network to compute **more complex representations**
 - The **lower RNNs** should compute **lower-level features** and the **higher RNNs** should compute **higher-level features**
- Multi-layer RNNs are also called ***stacked RNNs***

Multi-Layer RNNs



Multi-Layer RNNs

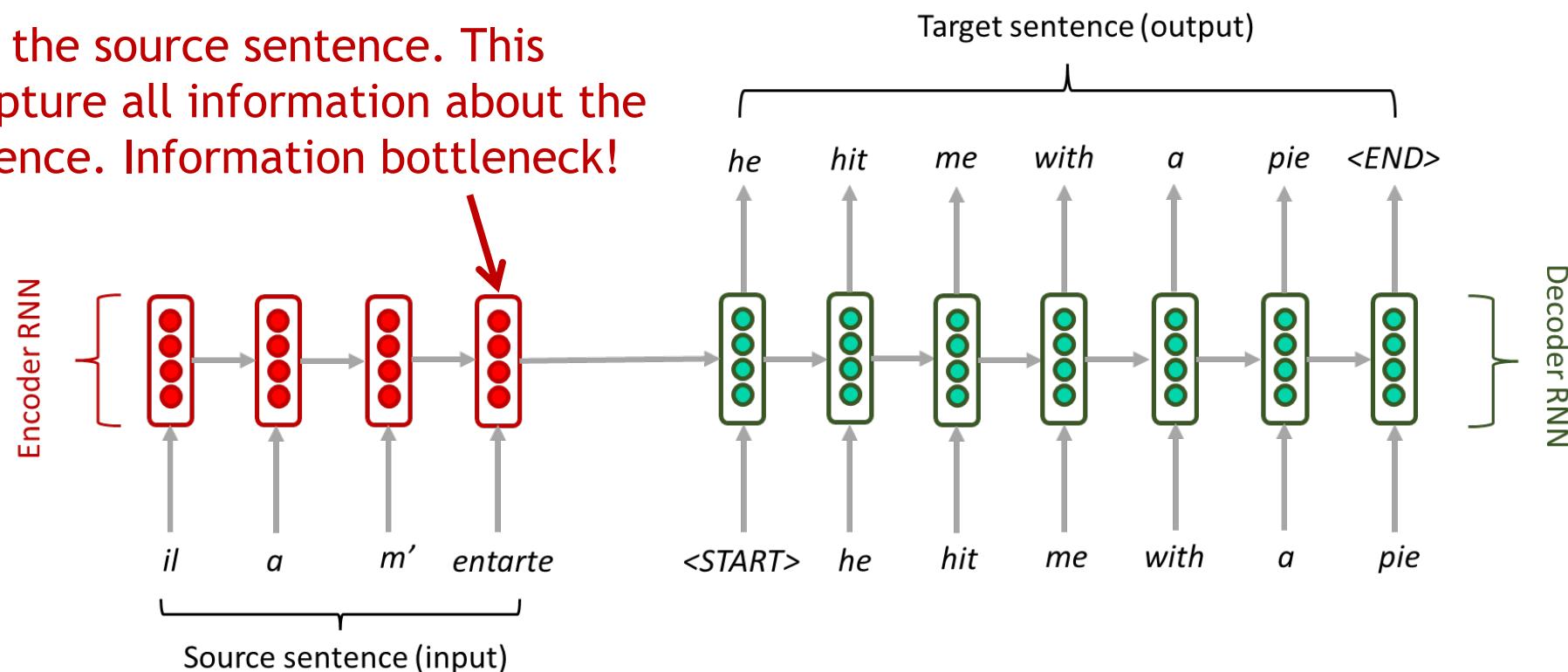
- High-performing RNNs are usually multi-layer (but aren't as deep as convolutional or feed-forward networks)
- For example: In a 2017 paper, Britz et al. find that for Neural Machine Translation, **2 to 4 layers** is best for the encoder RNN, and **4 layers** is best for the decoder RNN
 - However, **skip-connections/dense-connections** are needed to train deeper RNNs (e.g., 8 layers)
- Transformer-based networks (e.g., BERT) are usually deeper (e.g., **24 layers**)
 - They have a lot of skipping-like connections

“Massive Exploration of Neural Machine Translation Architectures”, Britz et al, 2017. <https://arxiv.org/pdf/1703.03906.pdf>

Attention: Seq-to-Seq Bottleneck Problem

- An **encoder-decoder** architecture is built with RNN
 - Neural machine translation (NMT) , Sequence to sequence (Seq2Seq) prediction
 - Detach the encoder and the decoder, so they have different lengths

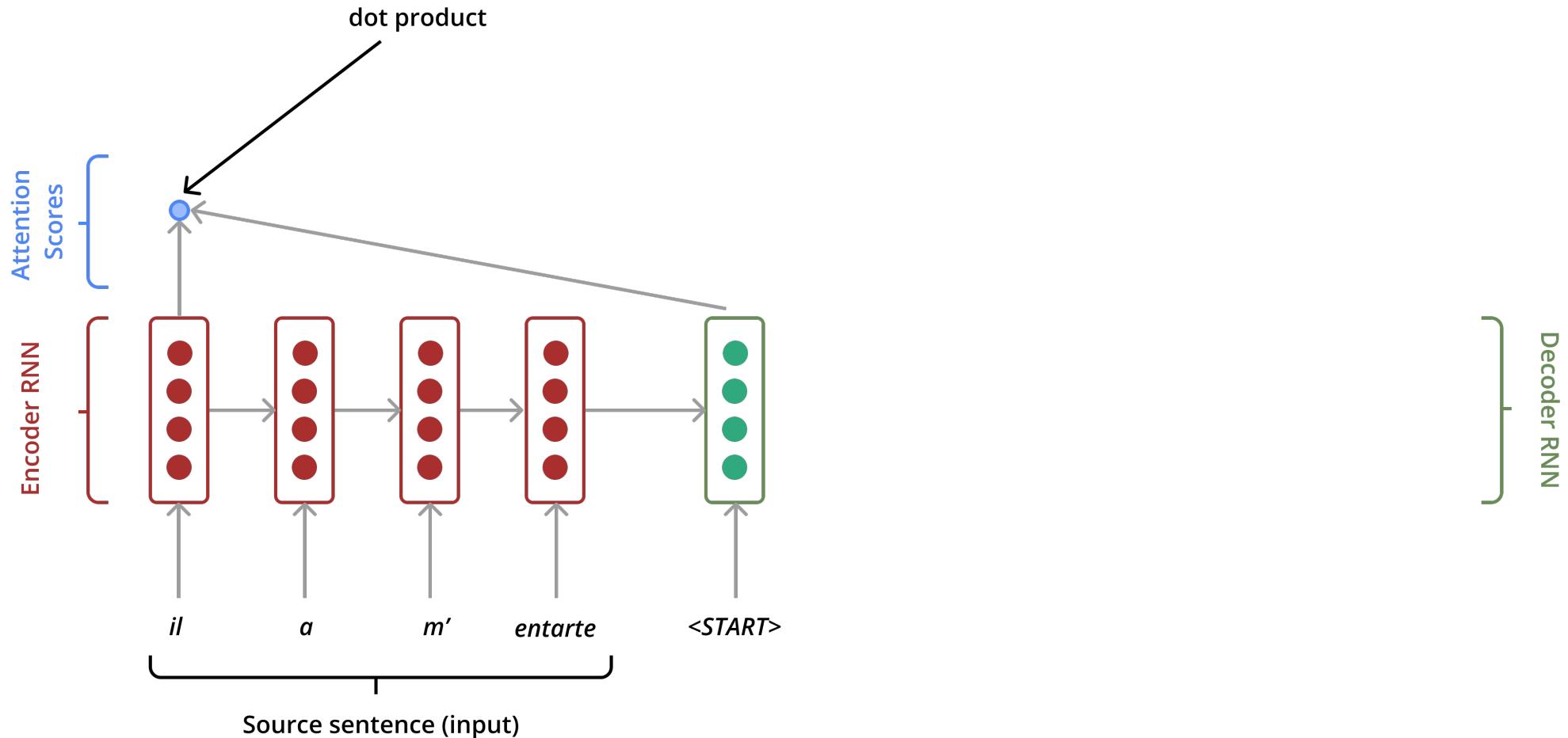
Encoding of the source sentence. This needs to capture all information about the source sentence. Information bottleneck!



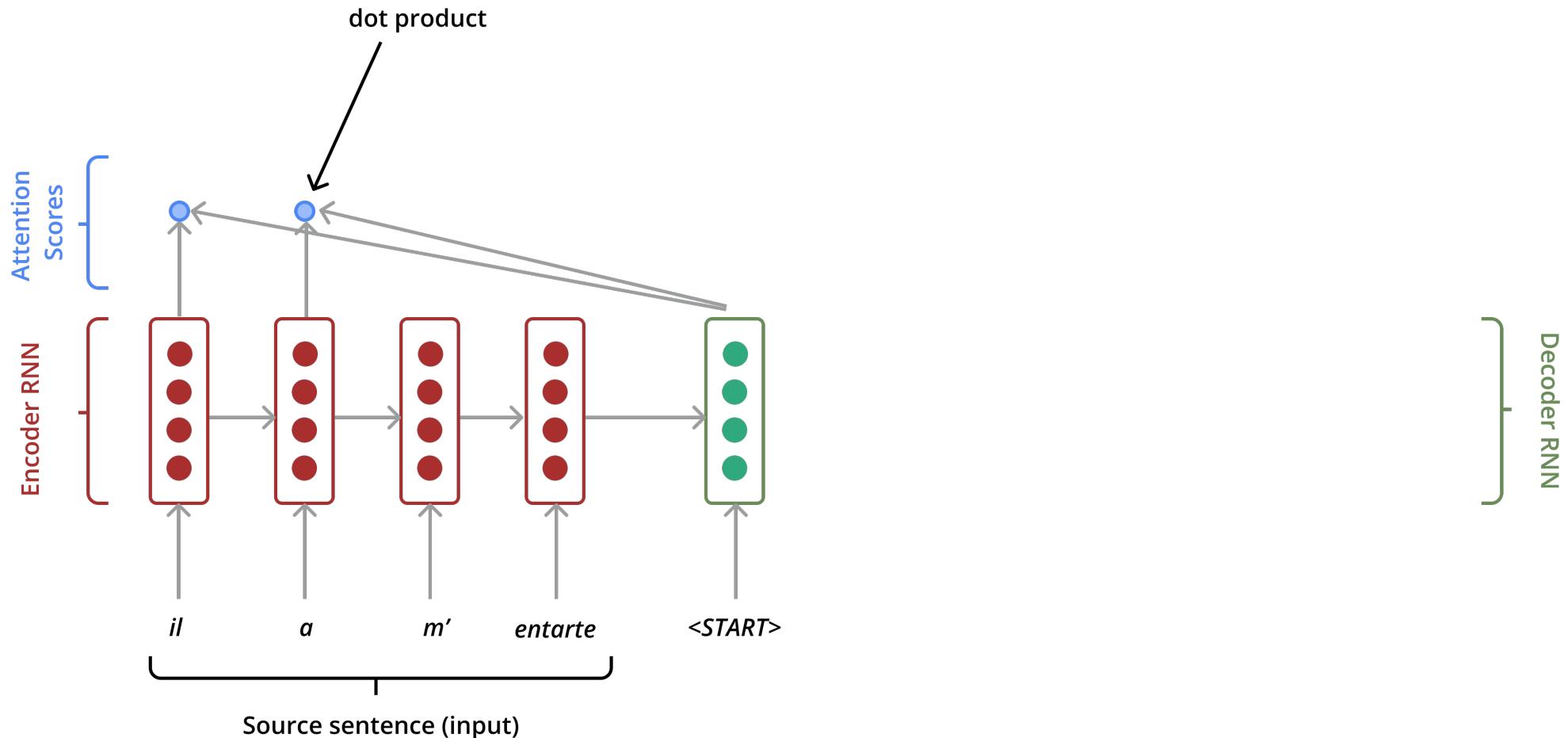
Attention

- **Attention** provides a solution to the bottleneck problem
- **Core idea:** at each step of the decoder, **use direct connection to the encoder to focus on a particular part** of the source sequence
- First, we will show via diagram (no equations), then we will show with equations

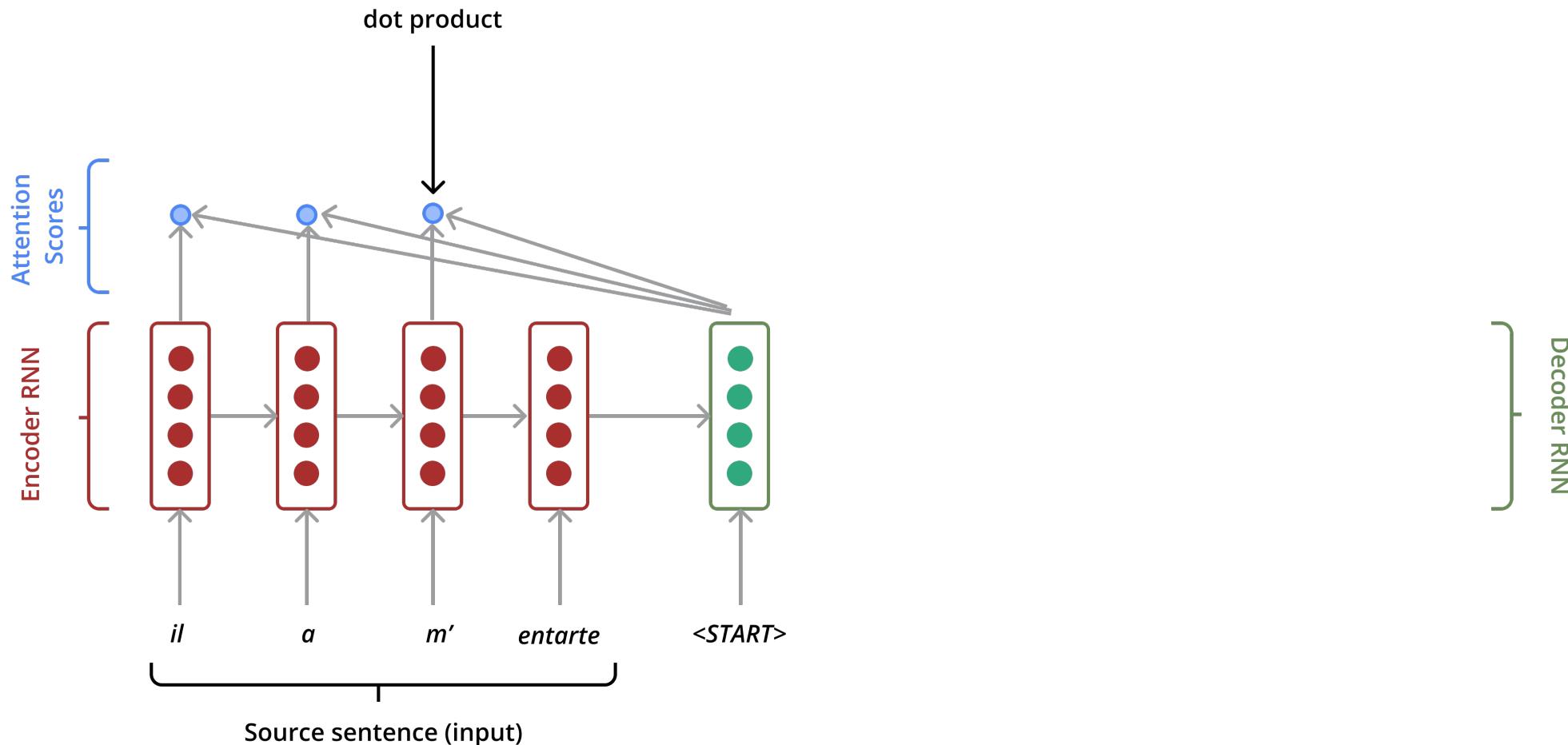
Sequence-to-Sequence with Attention



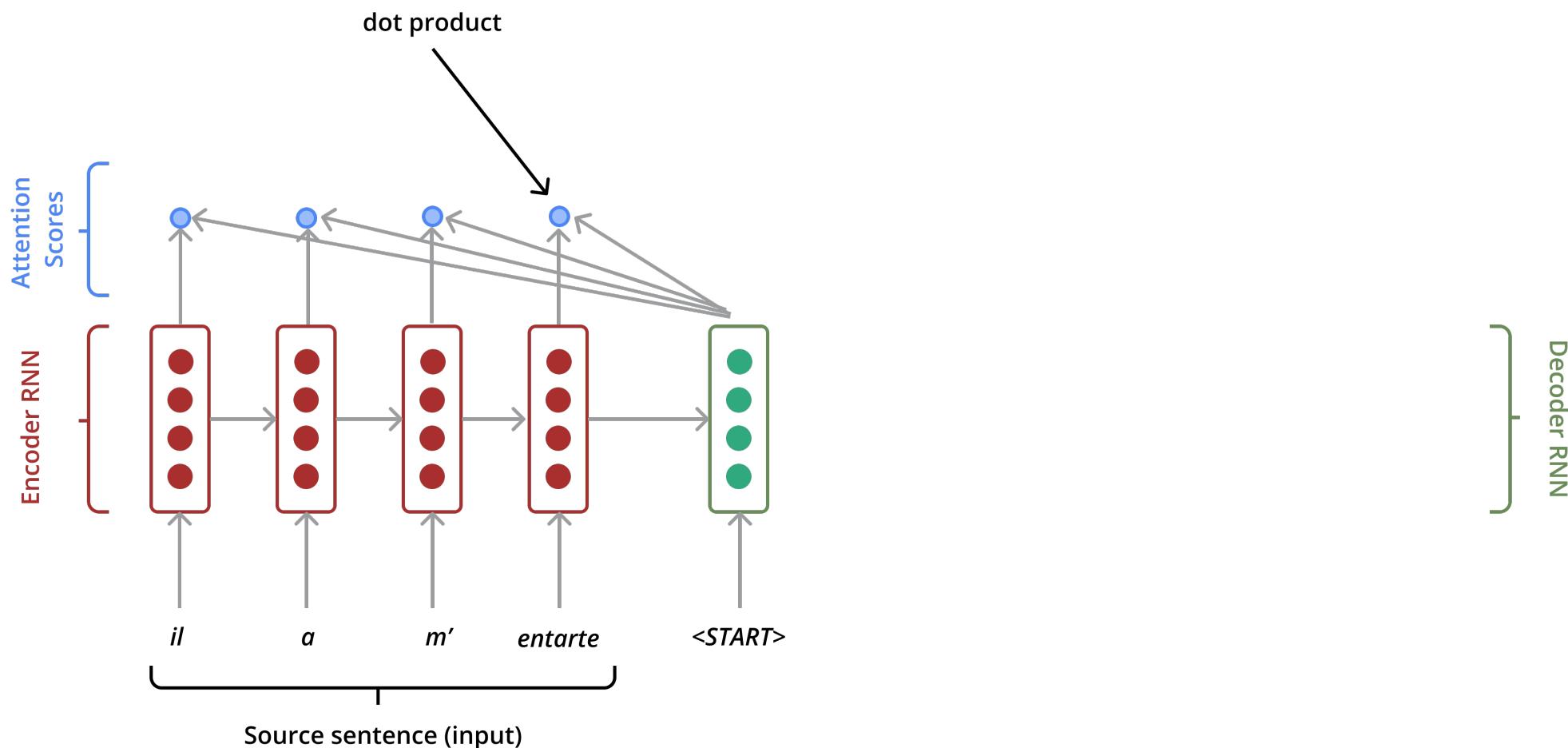
Sequence-to-Sequence with Attention



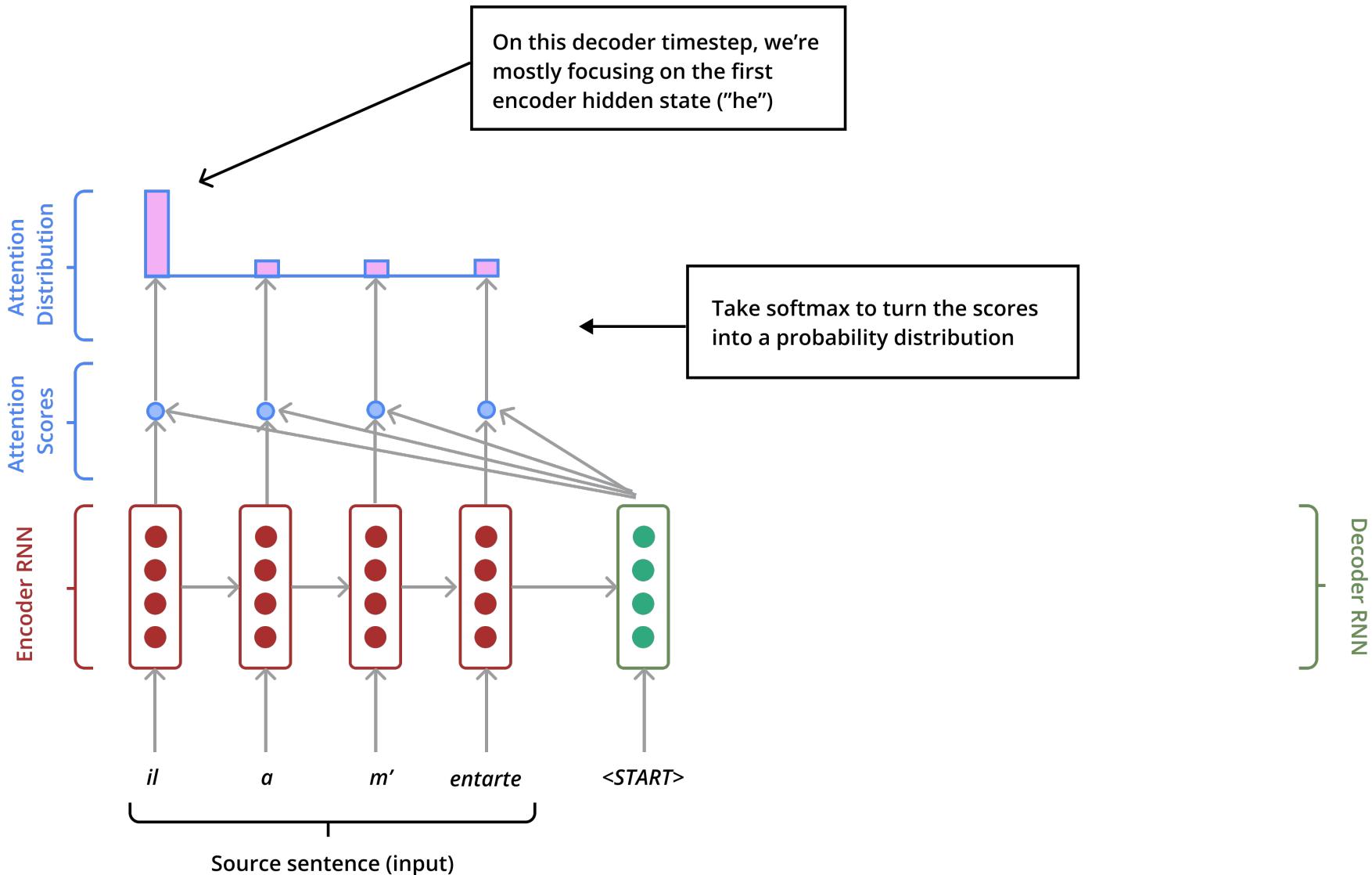
Sequence-to-Sequence with Attention



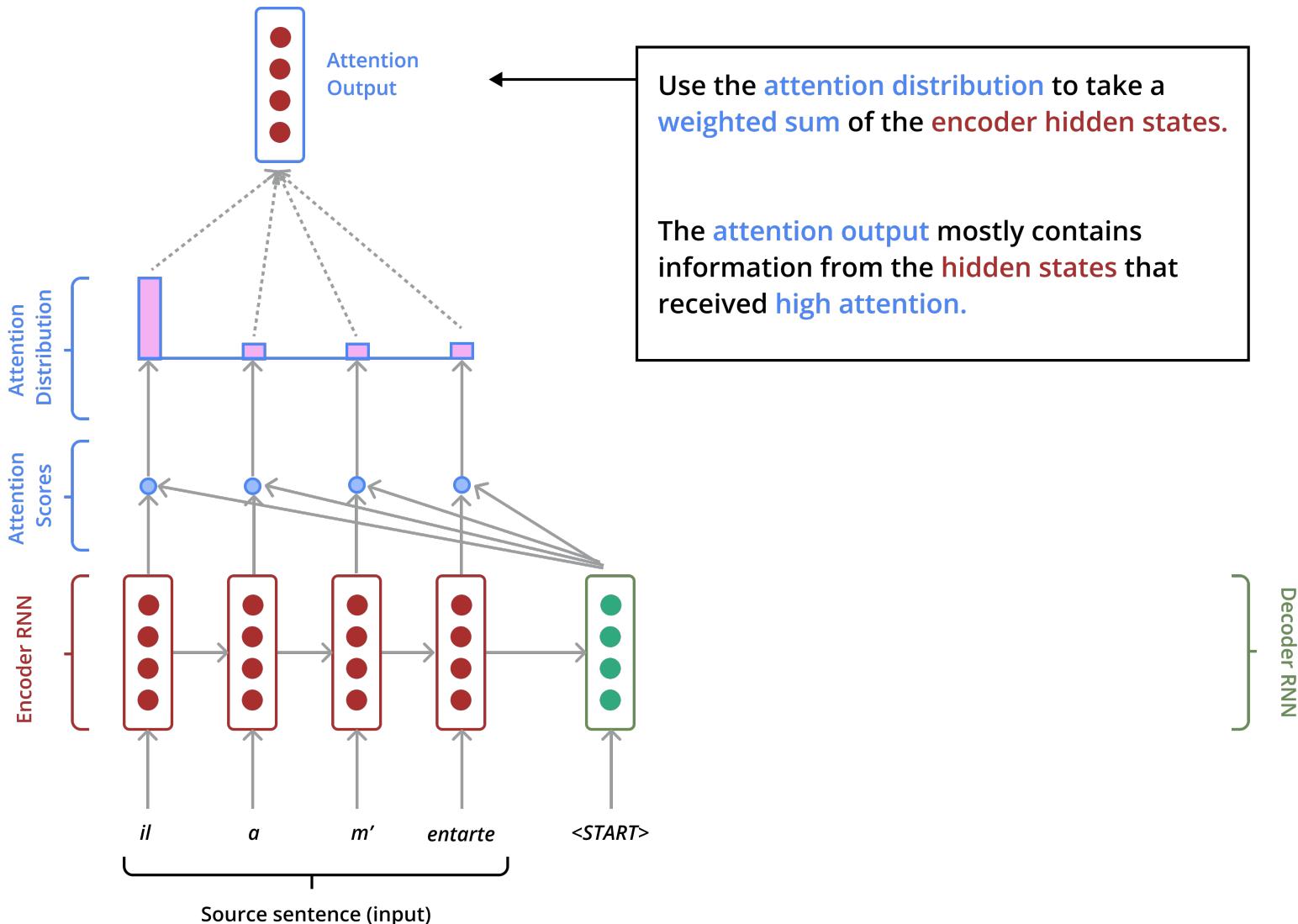
Sequence-to-Sequence with Attention



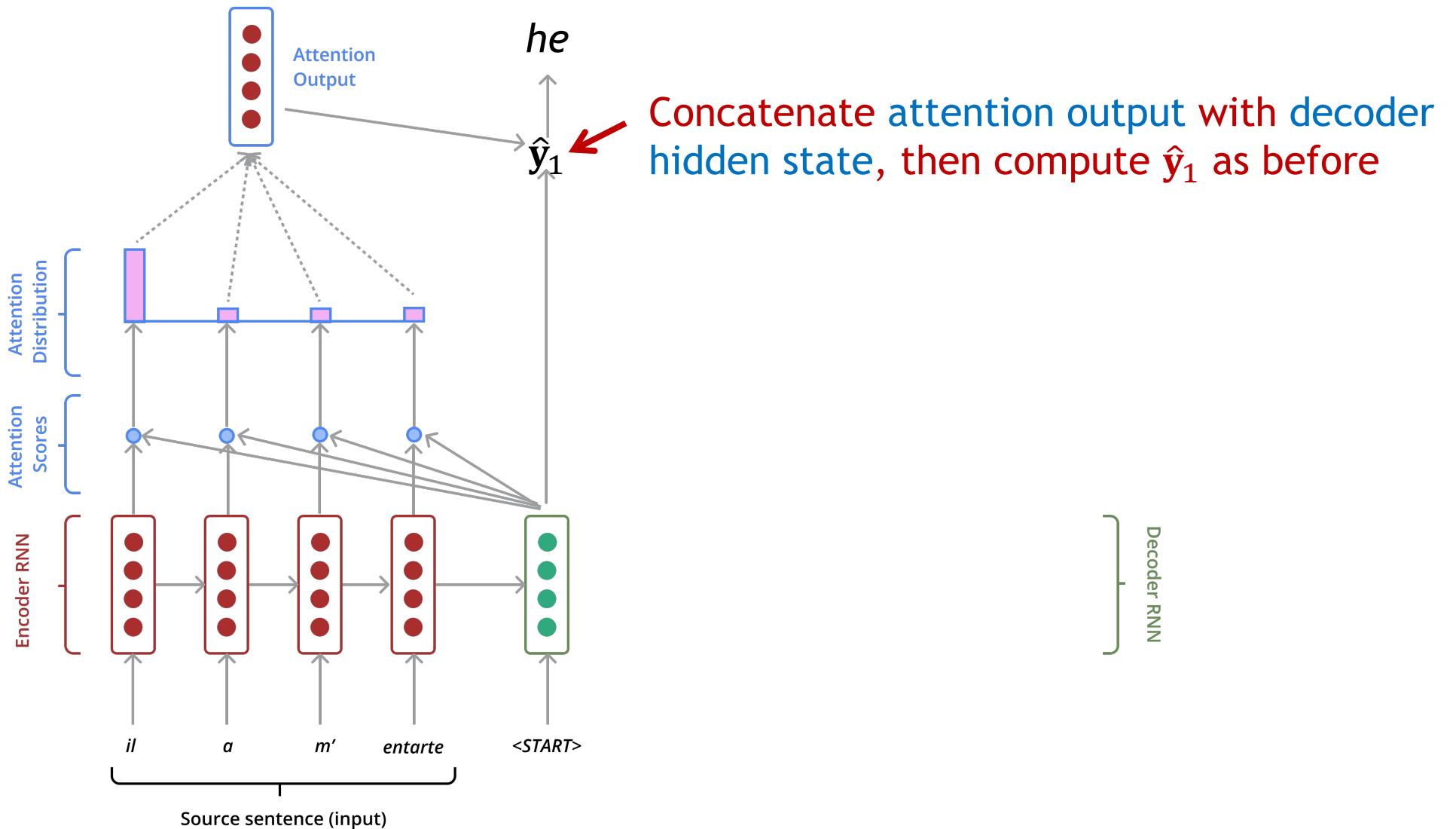
Sequence-to-Sequence with Attention



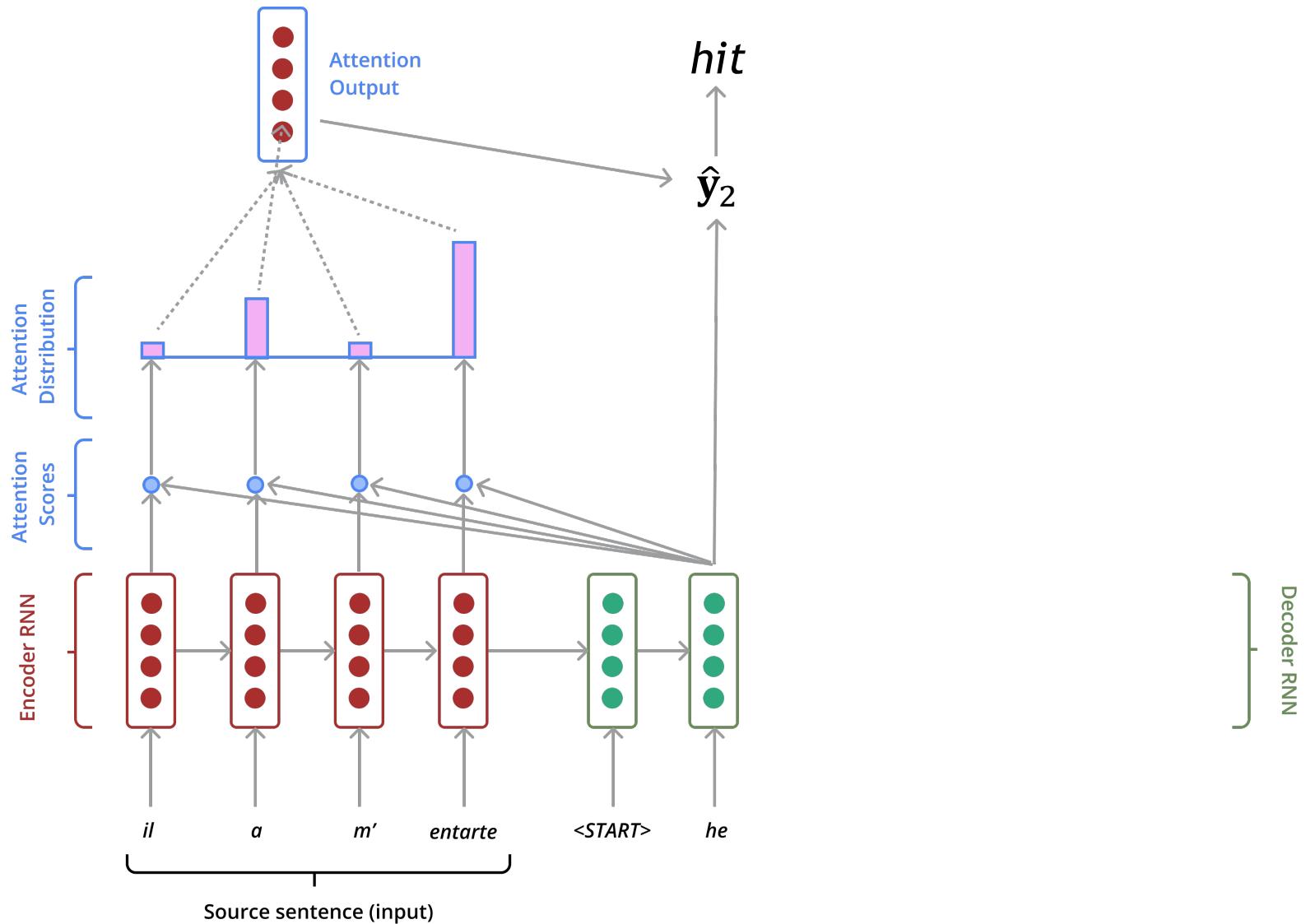
Sequence-to-Sequence with Attention



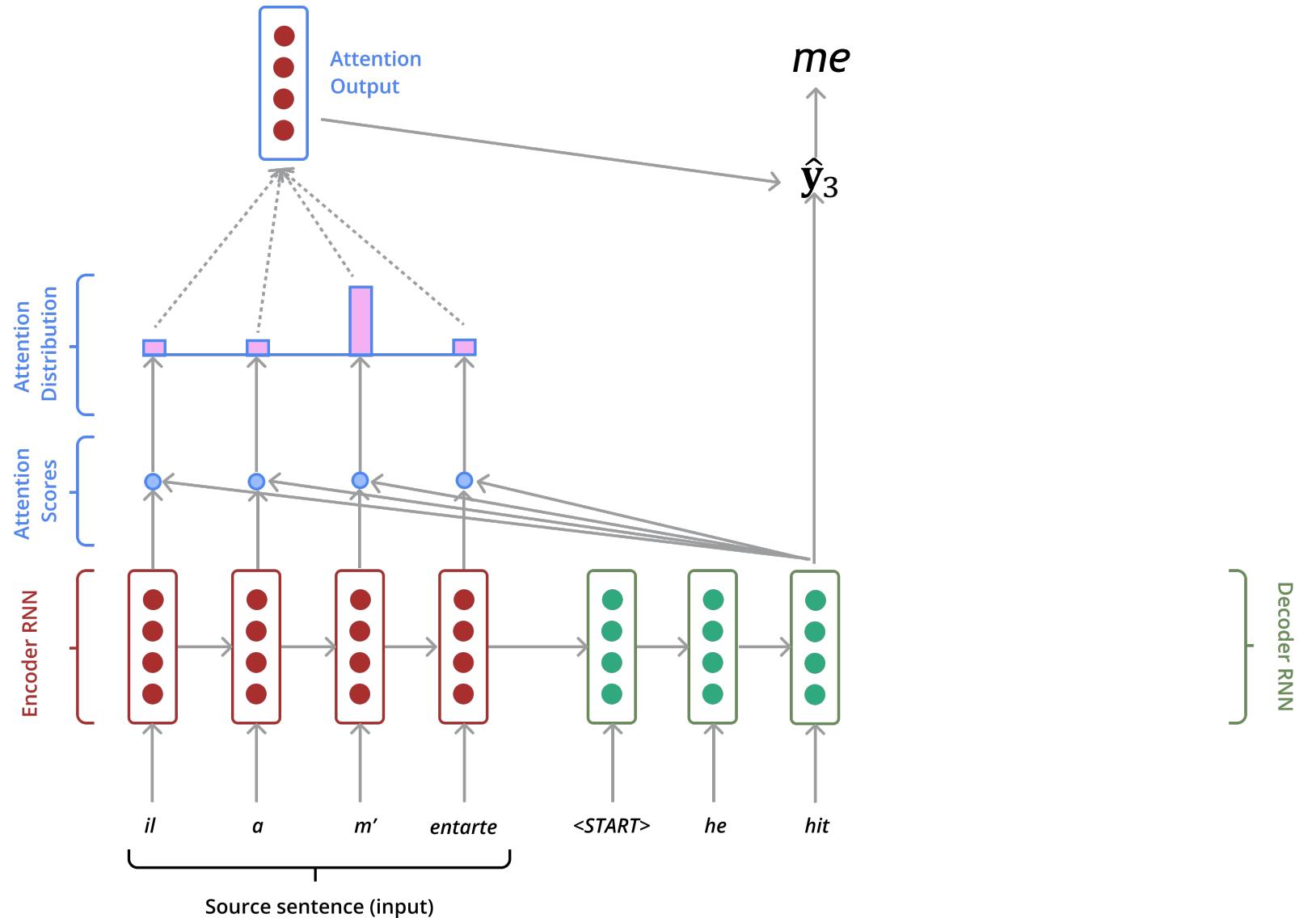
Sequence-to-Sequence with Attention



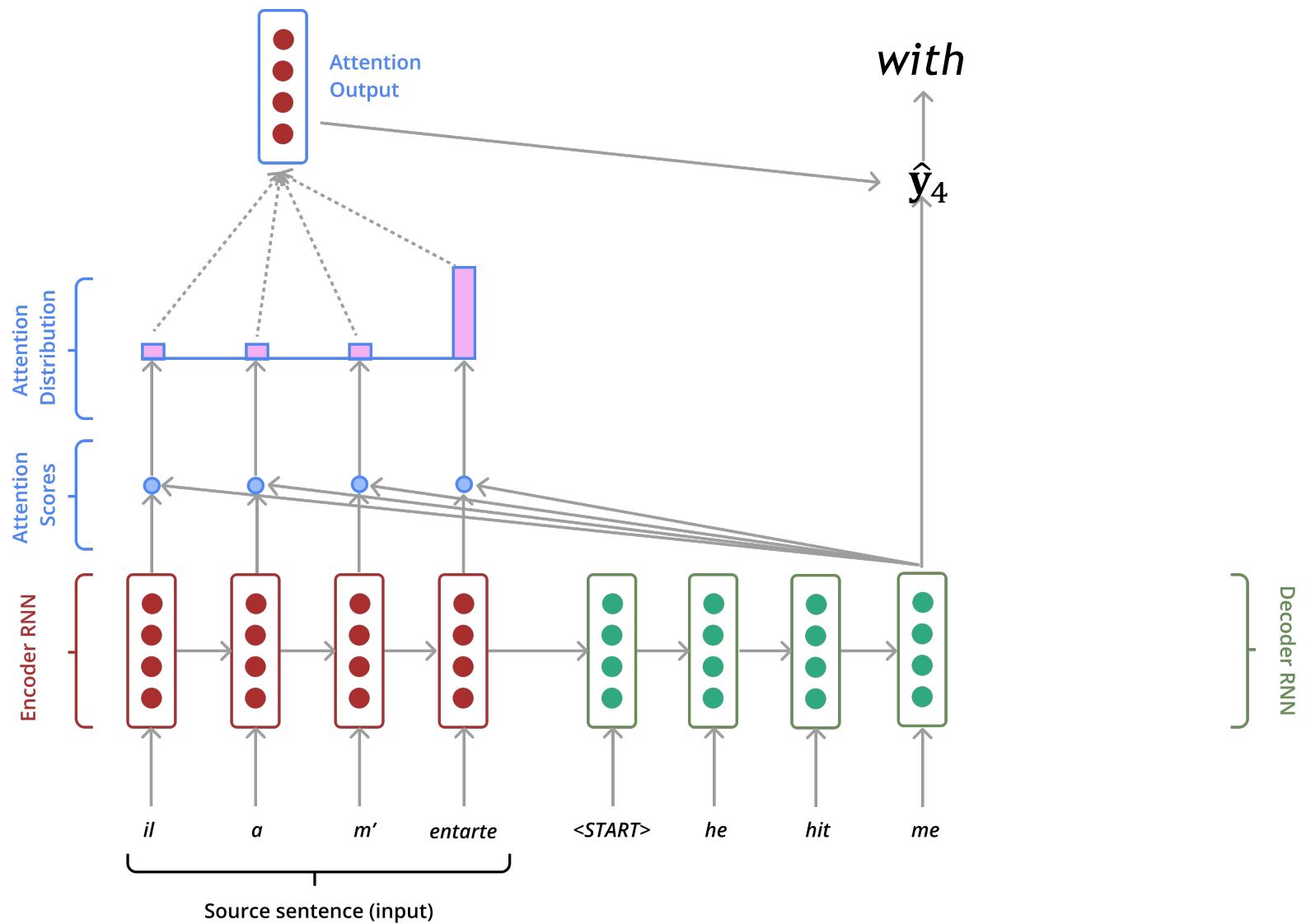
Sequence-to-Sequence with Attention



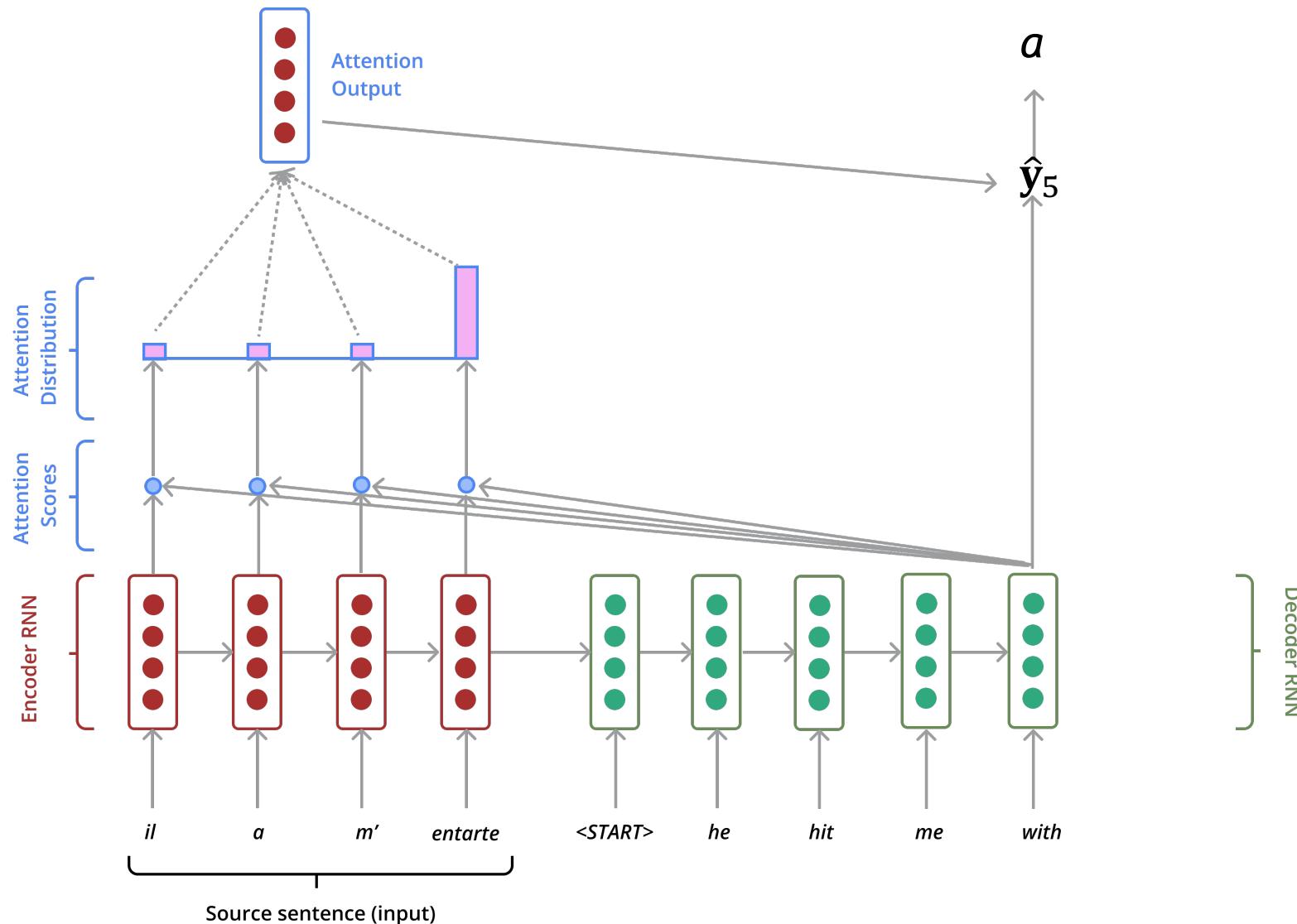
Sequence-to-Sequence with Attention



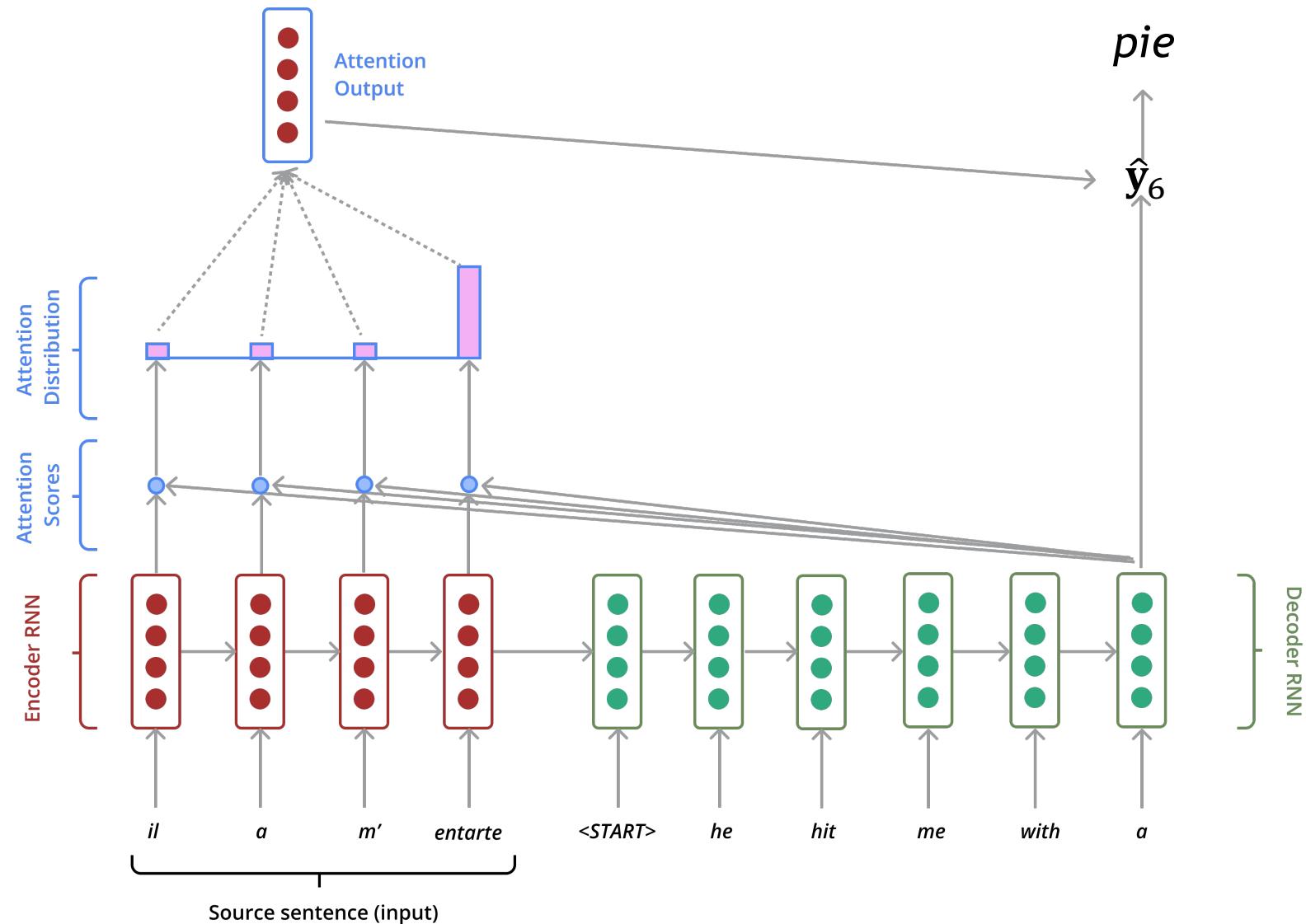
Sequence-to-Sequence with Attention



Sequence-to-Sequence with Attention



Sequence-to-Sequence with Attention



Attention: Equations

- We have encoder hidden states $\mathbf{h}_1, \dots, \mathbf{h}_N \in \mathbb{R}^d$
- At timestep t , we have decoder hidden state $\mathbf{s}_t \in \mathbb{R}^d$
- We get the attention scores at this step:

$$\mathbf{e}^t = [\mathbf{s}_t^T \mathbf{h}_1, \dots, \mathbf{s}_t^T \mathbf{h}_N] \in \mathbb{R}^N$$

- We take softmax to get the attention distribution α^t for this step (this is a probability distribution and sums to 1)
- We use α^t to take a weighted sum of the encoder hidden states to get the attention output \mathbf{a}_t

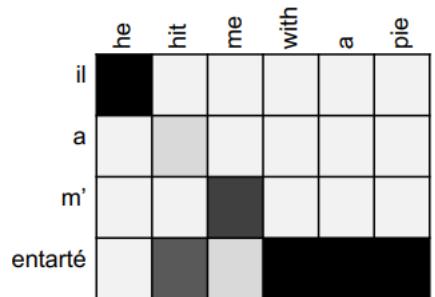
$$\mathbf{a}_t = \sum_{i=1}^N \alpha_i^t \mathbf{h}_i \in \mathbb{R}^d$$

- Finally we concatenate the attention output with the decoder hidden state \mathbf{s}_t and proceed as in the non-attention seq2seq model

$$[\mathbf{a}_t; \mathbf{s}_t] \in \mathbb{R}^{2d}$$

Attention is Useful

- Attention significantly **improves NMT performance**
 - It's very useful to allow decoder to focus on certain parts of the source
- Attention provides **more “human-like” model of the MT process**
 - You can look back at the source sentence while translating, rather than needing to remember it all
- Attention **solves the bottleneck problem**
 - Attention allows decoder to look directly at source; bypass bottleneck
- Attention **alleviate the vanishing gradient problem**
 - Provides shortcut to faraway states
- Attention provides **some interpretability**
 - By inspecting attention distribution, we see what the decoder was focusing on
 - We get (soft) **alignment for free!**
 - This is cool because we never explicitly trained an alignment system
 - The network just learned alignment by itself



Variants of Attention

- We have some **values** $\mathbf{h}_1, \dots, \mathbf{h}_N \in \mathbb{R}^{d_1}$ and a **query** $\mathbf{s} \in \mathbb{R}^{d_2}$
- Attention computation:
 - Computing the **attention scores** $\mathbf{e} \in \mathbb{R}^N$ (**there are multiple ways to do this**)
 - Taking softmax to get **attention distribution** α
$$\alpha = \text{softmax}(\mathbf{e}) \in \mathbb{R}^N$$
 - Using attention distribution to take weighted sum of values
$$\mathbf{a} = \sum_{i=1}^N \alpha_i \mathbf{h}_i \in \mathbb{R}^{d_1}$$
- Finally obtaining the **attention output** \mathbf{a} (sometimes called the context vector)

Variants of Attention

- There are **several ways** to compute $e \in \mathbb{R}^N$ from $\mathbf{h}_1, \dots, \mathbf{h}_N \in \mathbb{R}^{d_1}$ and $\mathbf{s} \in \mathbb{R}^{d_2}$
- Basic **dot-product attention**: $e_i = \mathbf{s}^T \mathbf{h}_i \in \mathbb{R}$
 - Note: this assumes $d_1=d_2$
- **Multiplicative attention**: $e_i = \mathbf{s}^T \mathbf{W} \mathbf{h}_i \in \mathbb{R}$
 - $\mathbf{W} \in \mathbb{R}^{d_1 \times d_2}$ is a weight matrix (called bilinear attention)
- **Additive attention**: $e_i = \tanh(\mathbf{W}_1 \mathbf{h}_i + \mathbf{W}_2 \mathbf{s}) \in \mathbb{R}$
 - Weight matrices $\mathbf{W}_1 \in \mathbb{R}^{d_3 \times d_1}$, $\mathbf{W}_2 \in \mathbb{R}^{d_3 \times d_2}$
 - d_3 (attention dimensionality) is a hyperparameter
 - It actually uses a feed-forward neural network layer

Attention is A General DL Technique

- We've seen that attention is a great way to improve the sequence-to-sequence model for Machine Translation (MT)
- However: attention can be used in **many architectures** (not just seq2seq) and **many tasks** (not just NMT)
 - Graph attention networks
- More general definition of attention:
 - Given a set of vector **values**, and a vector **query**, **attention** is a technique to compute a **weighted sum of the values**, dependent on the query
- We sometimes say that the **query attends to the values**
 - For example, in the seq2seq + attention model, each decoder hidden state (query) attends to all the encoder hidden states (values)

Attention is A General DL Technique

- Intuition:
 - The weighted sum is a selective summary of the information contained in the values, where the query determines which values to focus on
 - Attention is a way to obtain a **fixed-size representation of an arbitrary set of representations** (the **values**), dependent on some other representation (the **query**)
- Attention has become the powerful, flexible, general way pointer and memory manipulation in all deep learning models

Summary

- Language Modeling is the task of predicting what word comes next
- n-gram Language Models
 - Simple, storage problem, sparsity problem, fixed input size
- RNN Advantages:
 - Can process any length input
 - Computation for step t can (in theory) use information many steps back
 - Model size doesn't increase for longer input
 - Same weights applied on every timestep, so there is symmetry in how inputs are processed
- RNN Disadvantages:
 - Recurrent computation is slow
 - In practice, difficult to access information from many steps back

Summary

- LSTM and GRU
 - Design gates for capturing **long range dependencies**
- Use **bidirectionality** when possible
- Multi-layer RNNs are more powerful, but you might need **skip connections** if it's deep
- Attention mechanism helps alleviate the information bottleneck problem in Seq2seq problem



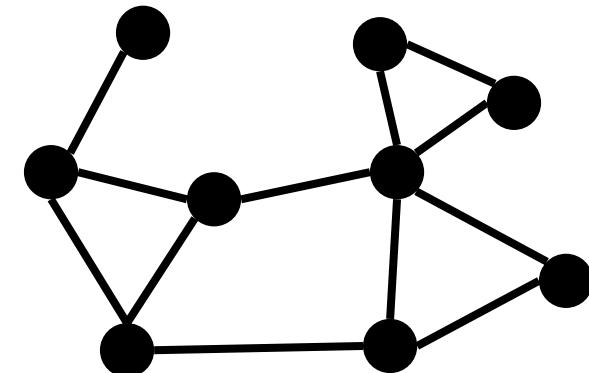
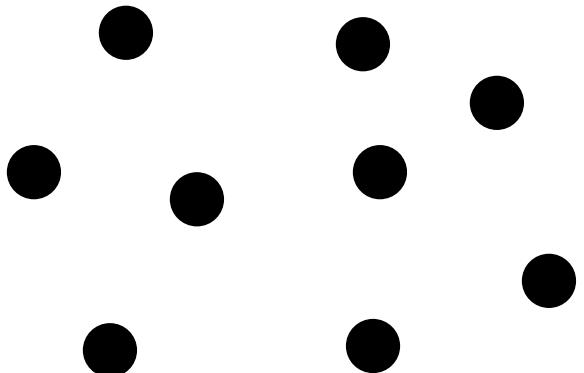
COSC 3337
Data Science I
Section 14623

Graph Data Analysis

Instructor: Jingchao Ni
Fall 2024

Why Graphs?

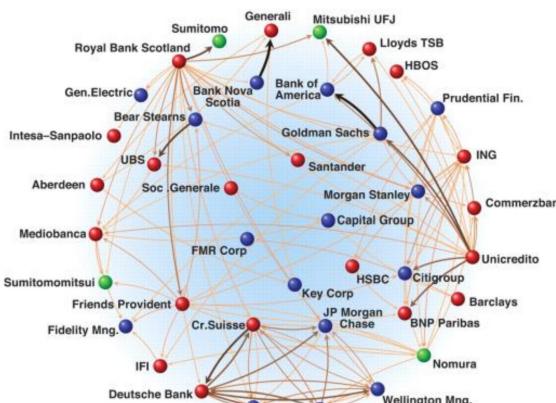
- Graphs are a general language for describing and analyzing entities with relations/interactions



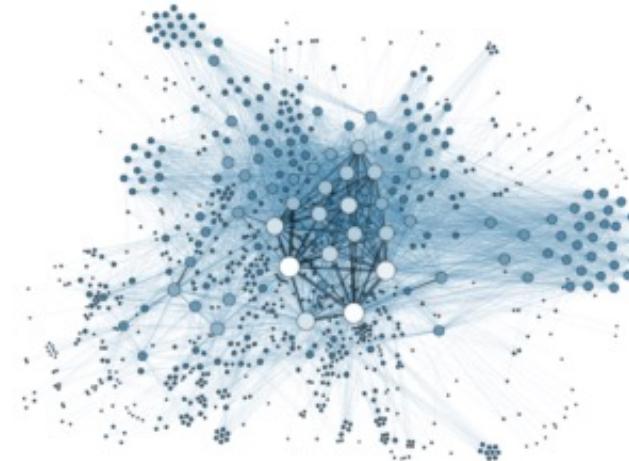
Many Types of Data are Graphs



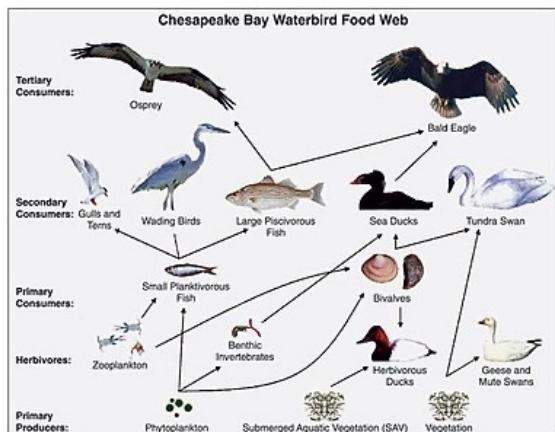
Social Networks



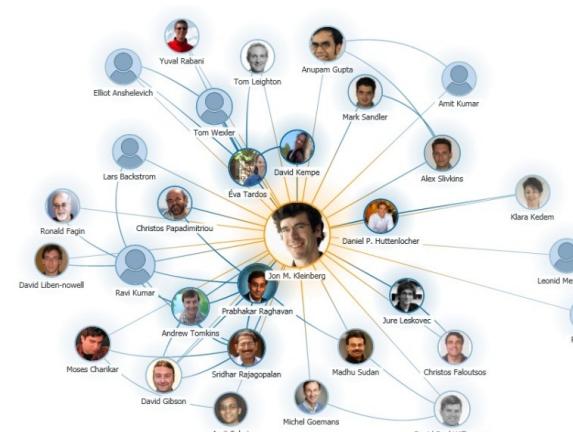
Economic Networks



Communication Networks



Food Webs

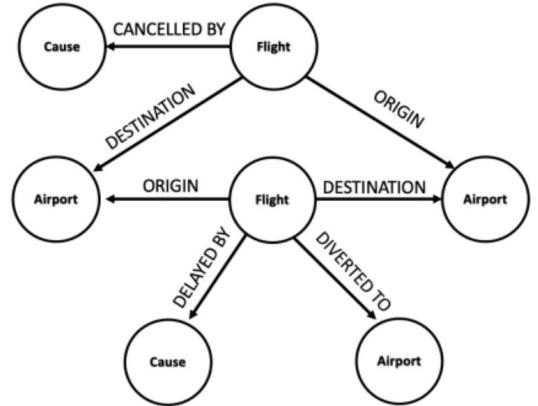


Collaboration Networks



Underground Networks

Many Types of Data are Graphs

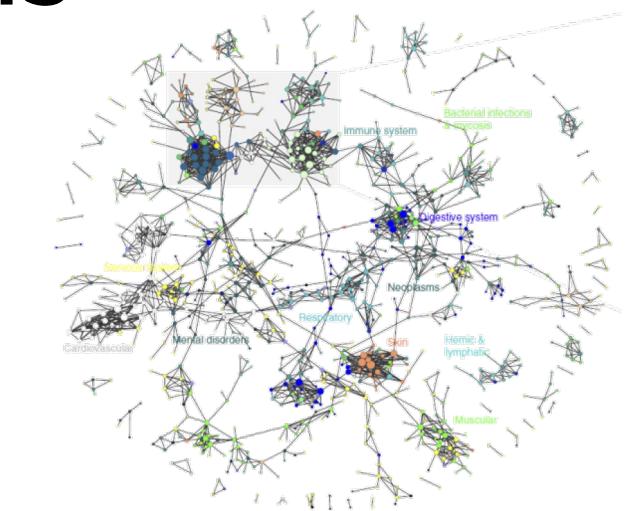


Event Graphs



Image credit: [SalientNetworks](#)

Computer Networks



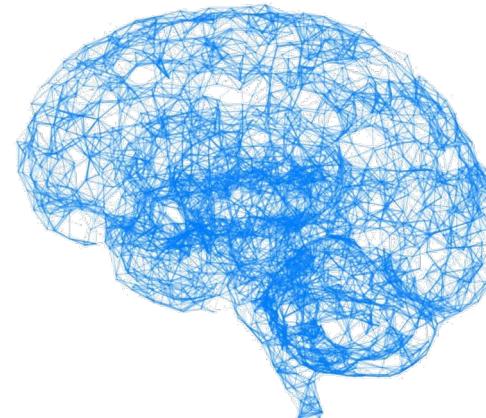
Disease Similarity Networks



Citation Networks

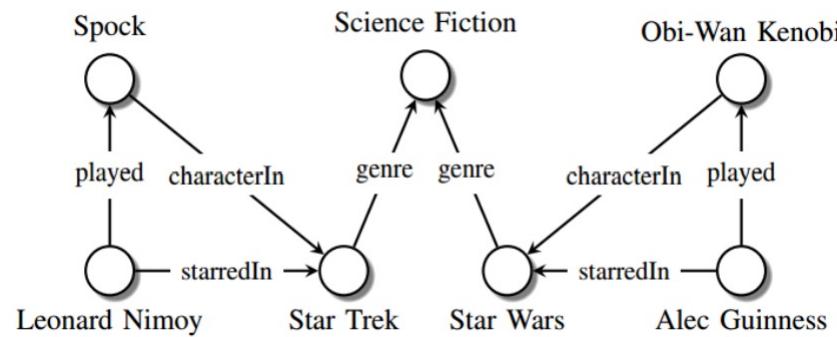


Internet

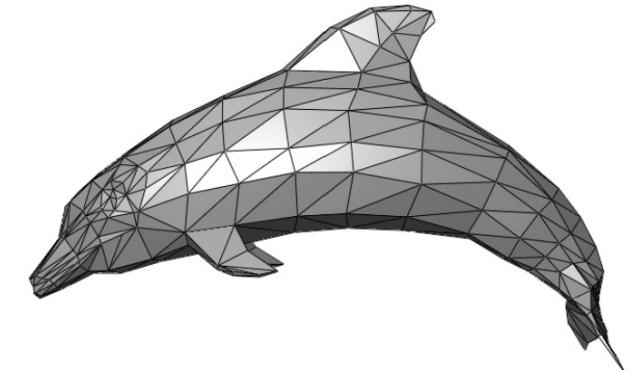
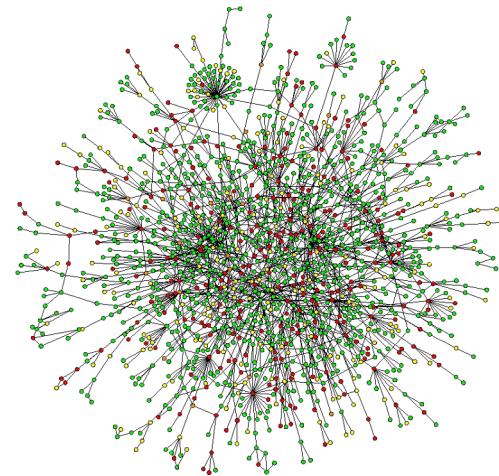


Networks of Neurons

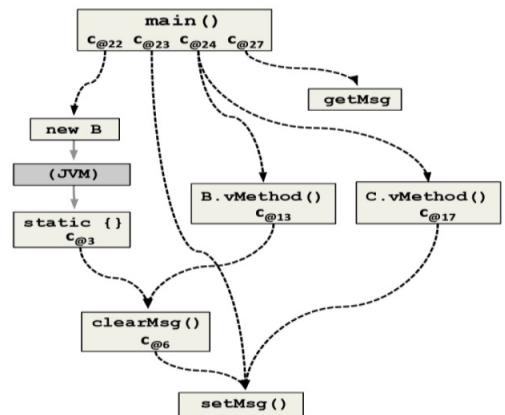
Many Types of Data are Graphs



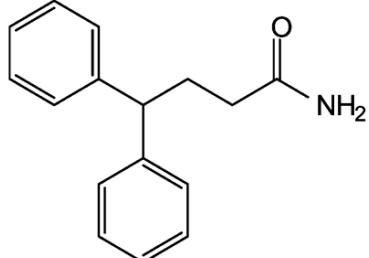
Knowledge Graphs



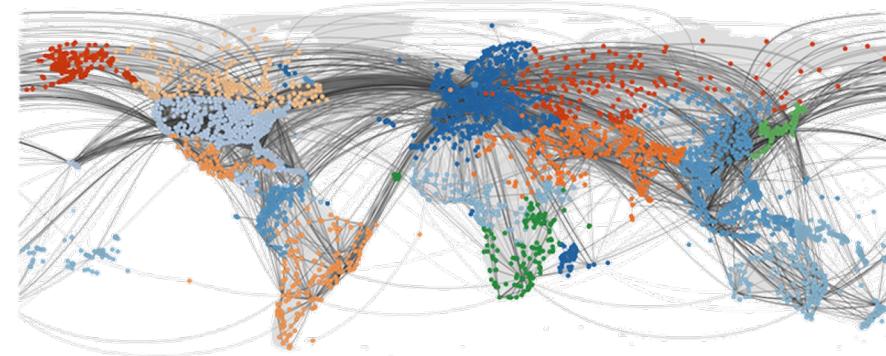
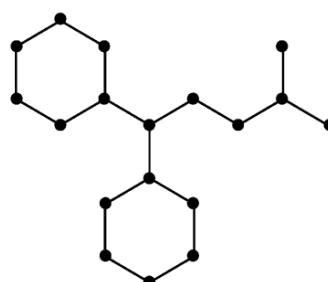
3D Shapes



Code Graphs

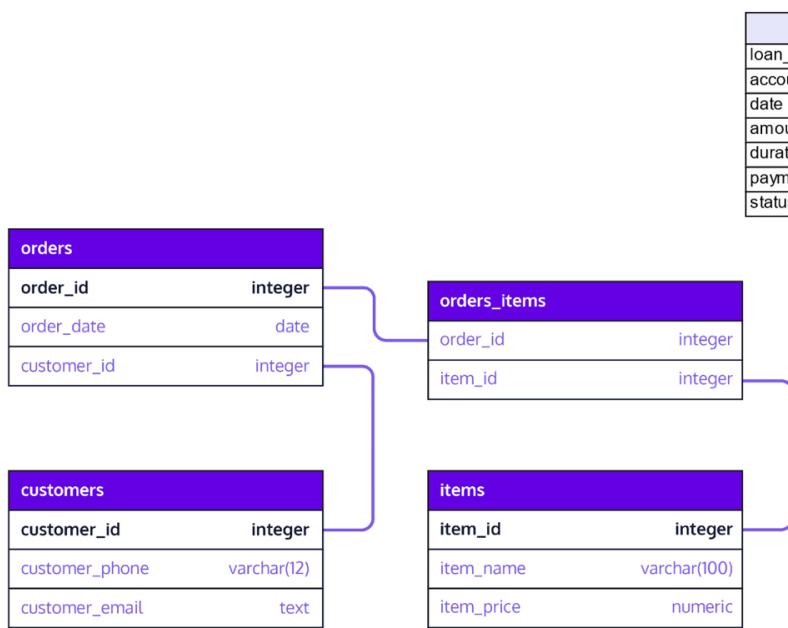


Molecular Structures

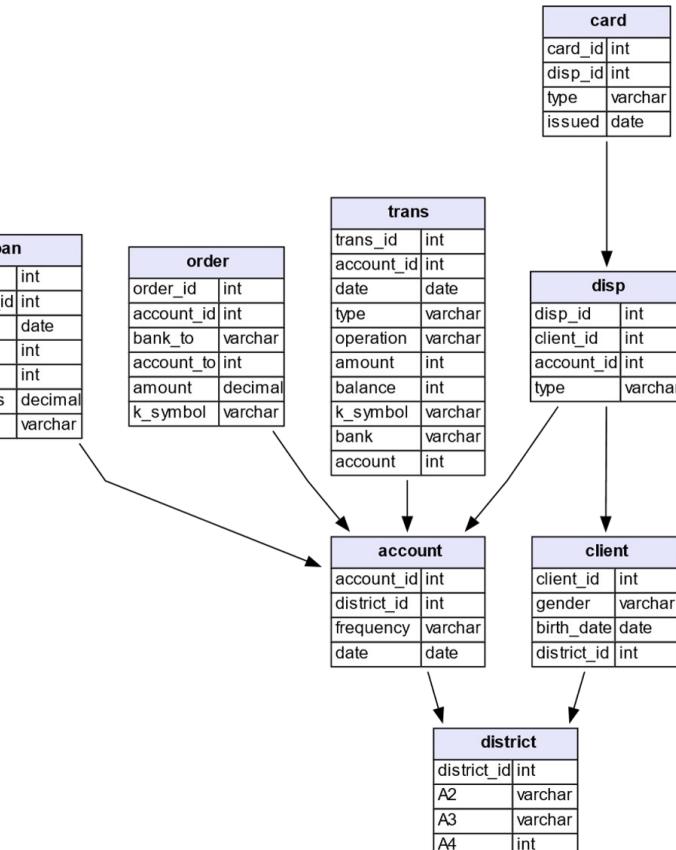


Transportation Networks

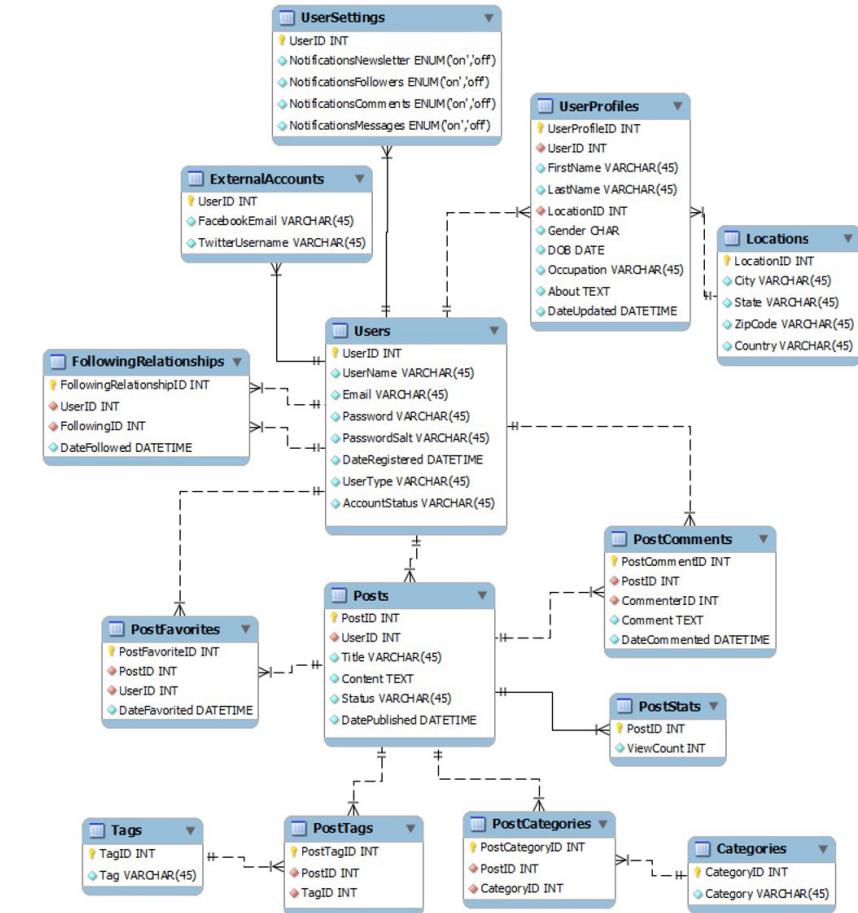
Databases are Graphs



Commerce

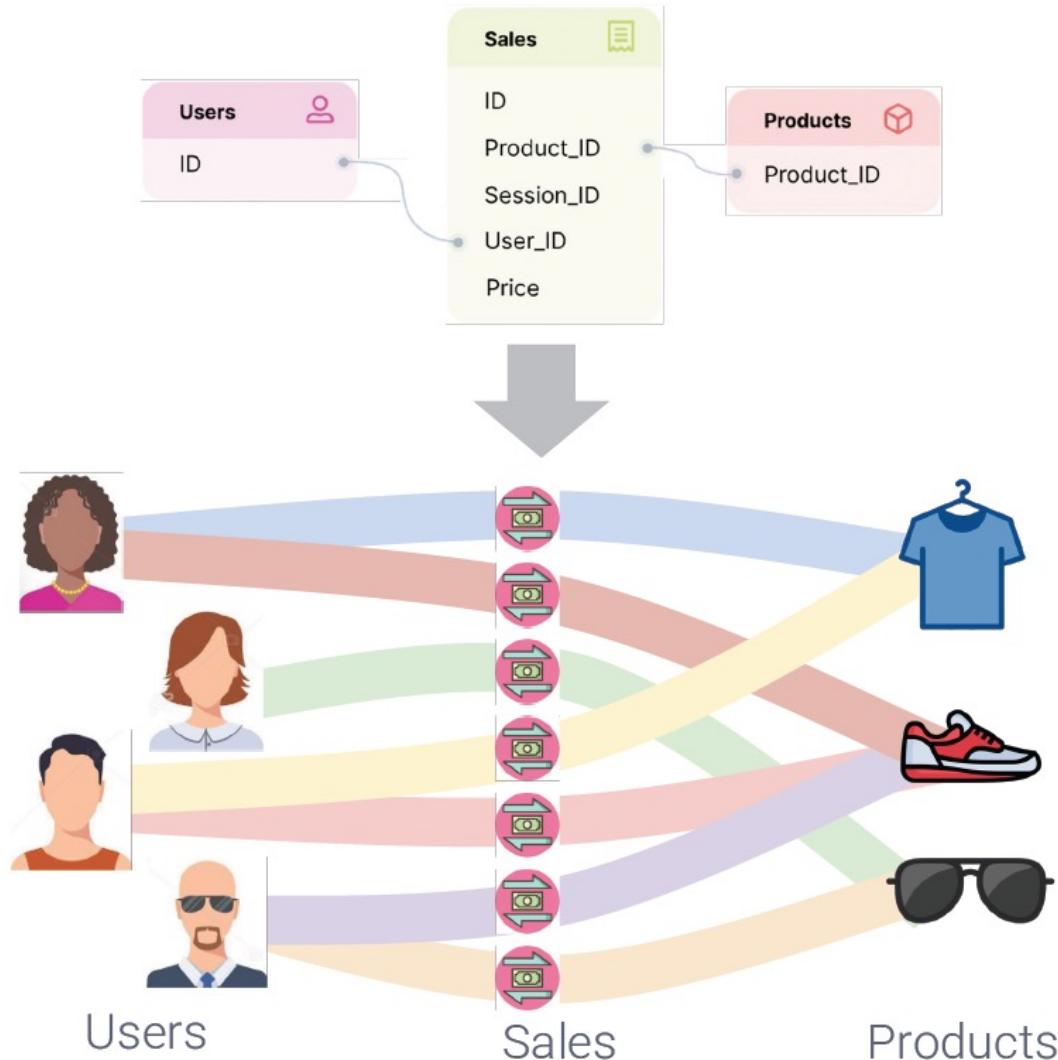


Finance



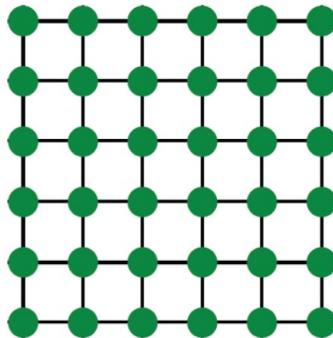
Social Media

Relational Machine Learning

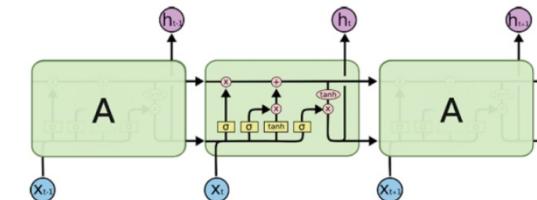
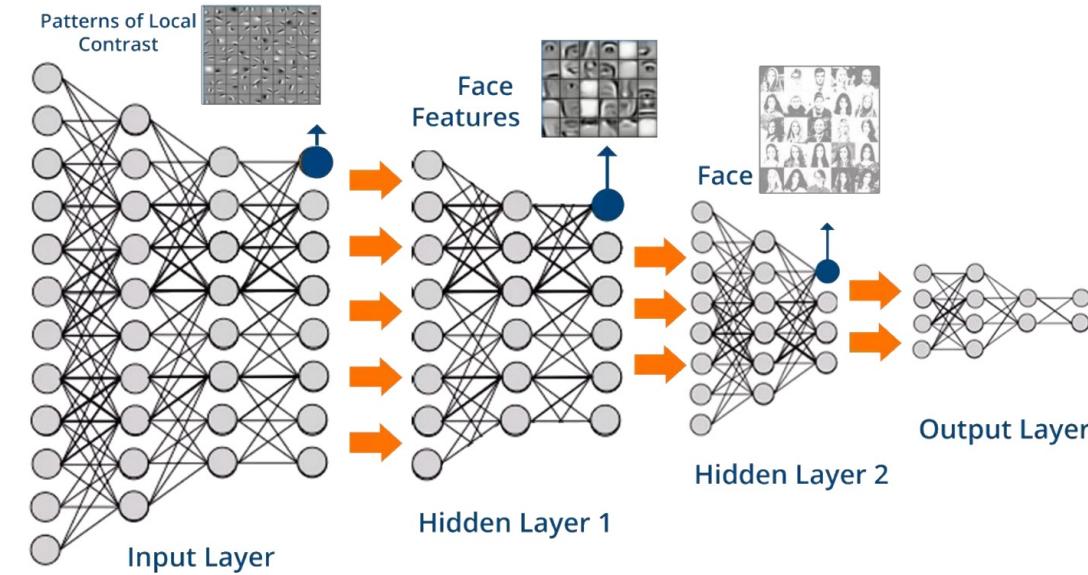


Machine Learning Toolbox

- Modern deep learning toolbox is designed for simple sequences and grids

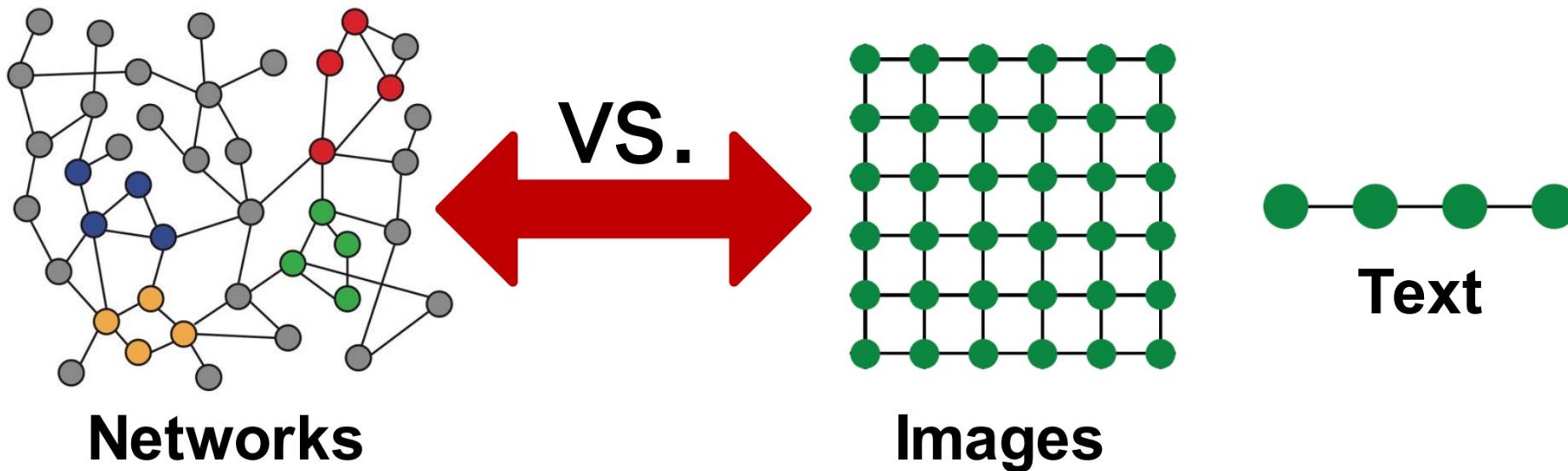


Images



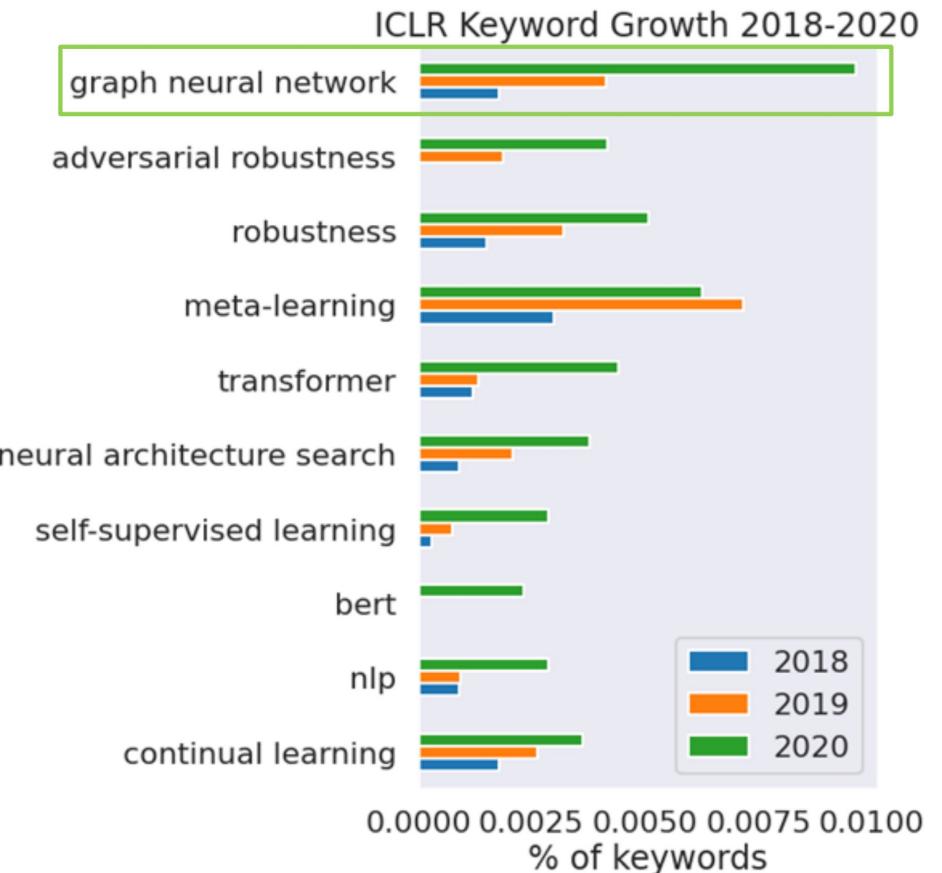
Why is Graph Machine Learning Hard?

- Networks are complex
 - Arbitrary size and complex topological structure (i.e., no spatial locality like grids)
 - No fixed node ordering or reference points
 - Often dynamic and have multimodal features



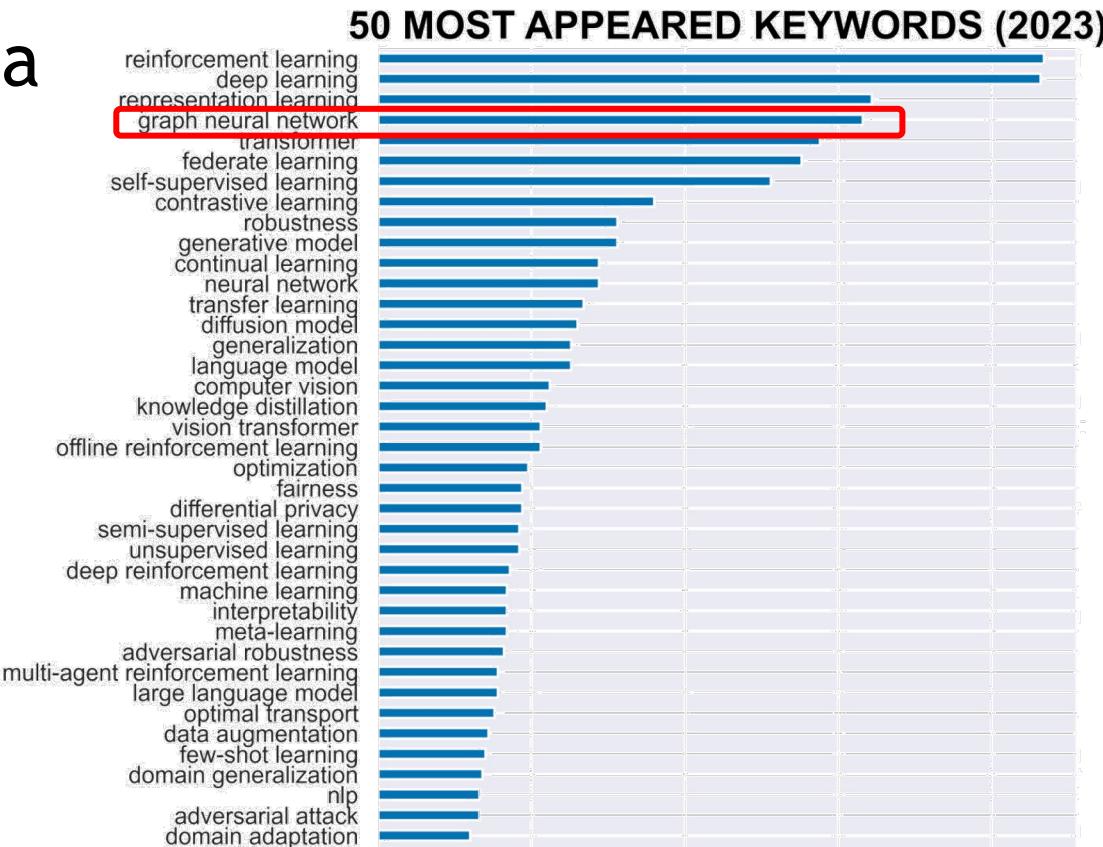
Important Subfield in Machine Learning

- Not everything can be represented as a sequence or a grid
- How can we develop neural networks that are much more broadly applicable?
- Graphs are the new frontier of deep learning



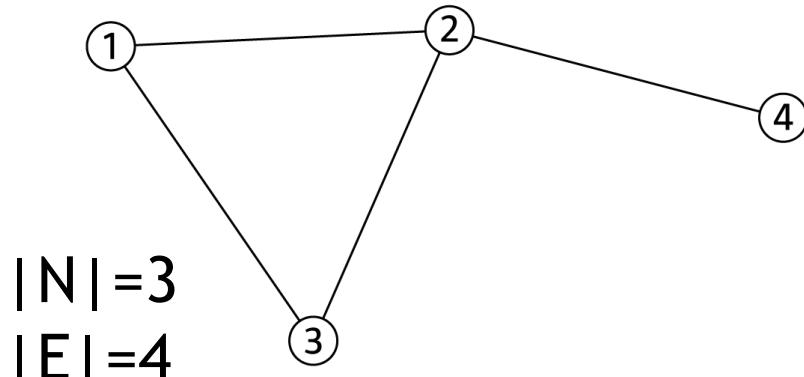
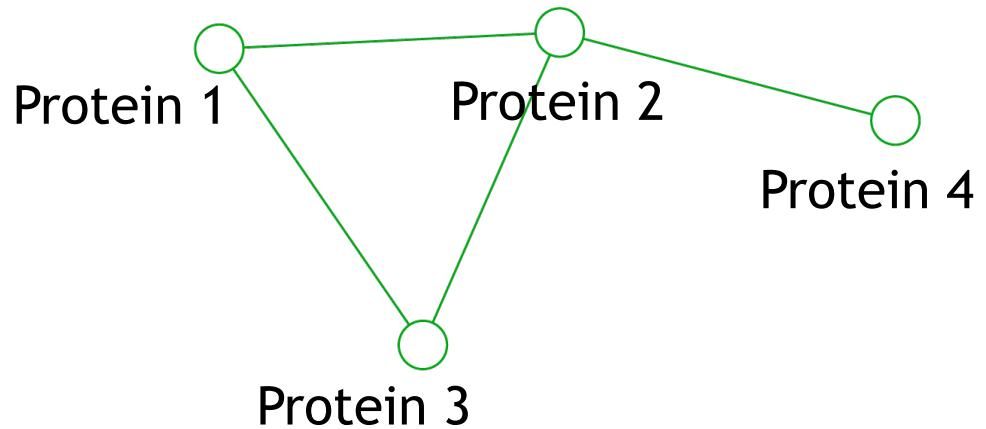
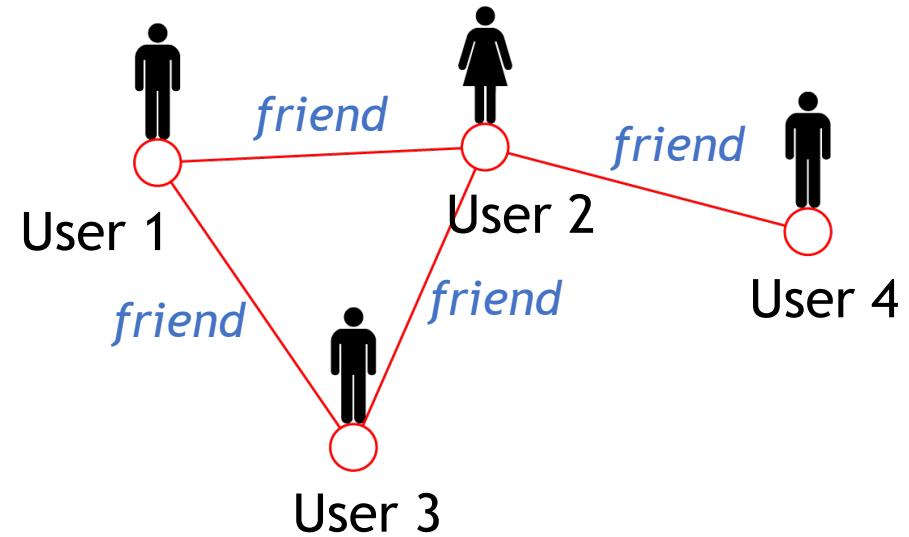
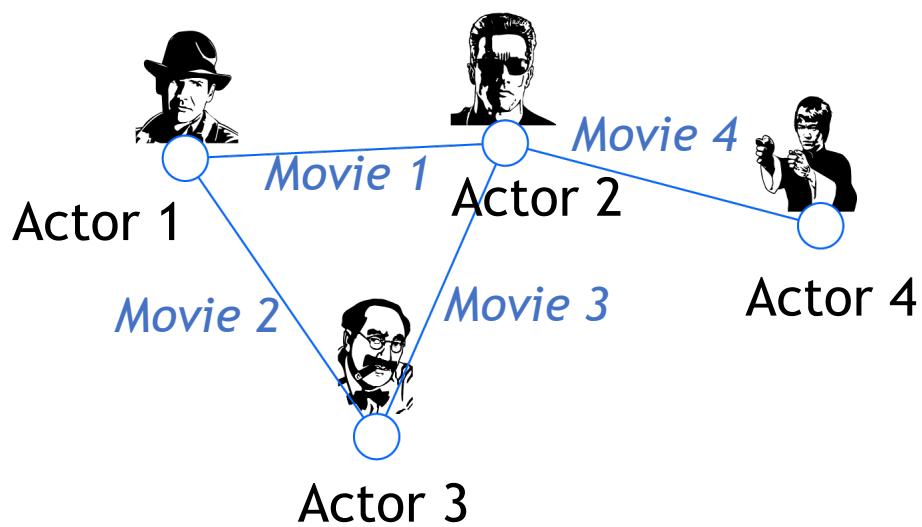
Important Subfield in Machine Learning

- Not everything can be represented as a sequence or a grid
- How can we develop neural networks that are much more broadly applicable?
- Graphs are the new frontier of deep learning



ICLR 2023 Keywords

Homogeneous Graphs



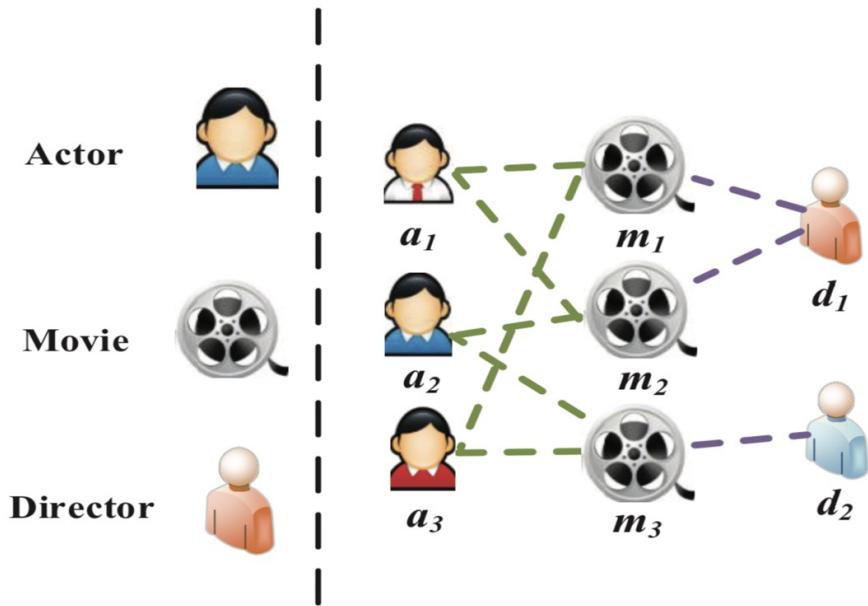
Heterogenous Graphs

- A heterogeneous graph is defined as

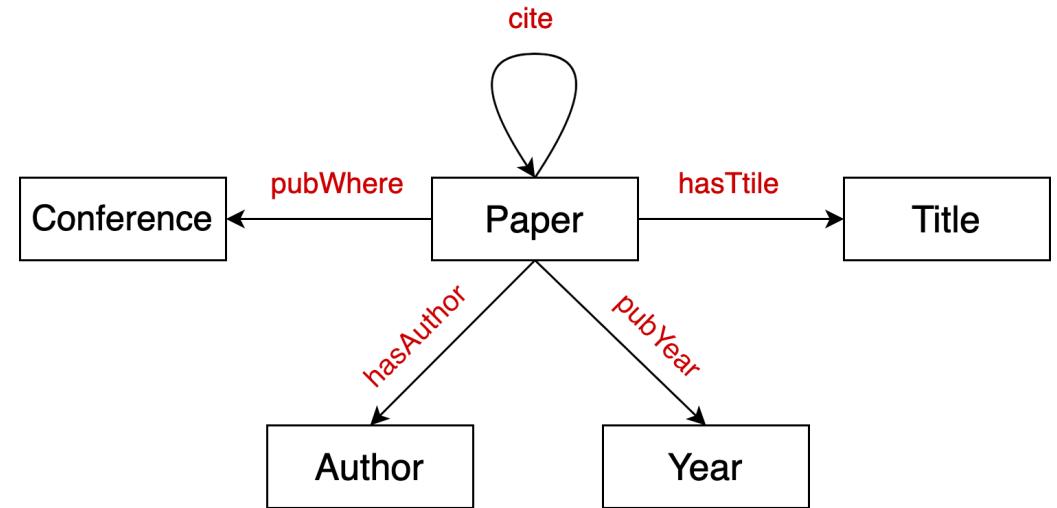
$$G = (V, E, R, T)$$

- Node set $v_i \in V$
- Edge set $(v_i, r, v_j) \in E$
- Nodes with node types $T(v_i)$
- Edges with relation types $r \in E$
- Nodes and edges have **attributes/features**

Many Graphs are Heterogeneous



Movie Actor Graphs



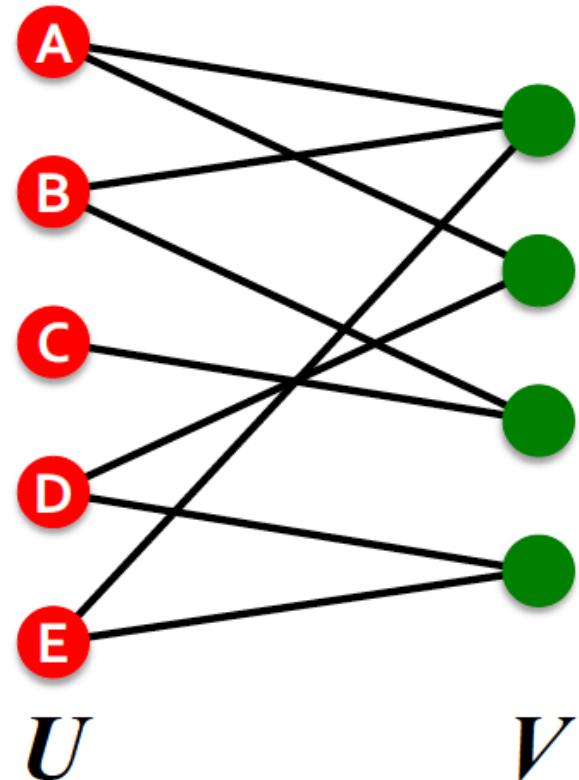
Academic Graphs

- Example node: ICML
- Example edge: (GraphSAGE, NeurIPS)
- Example node type: author
- Example edge type (relation): pubYear

Wang, et al. "Heterogeneous graph attention network." In WWW, 2019.

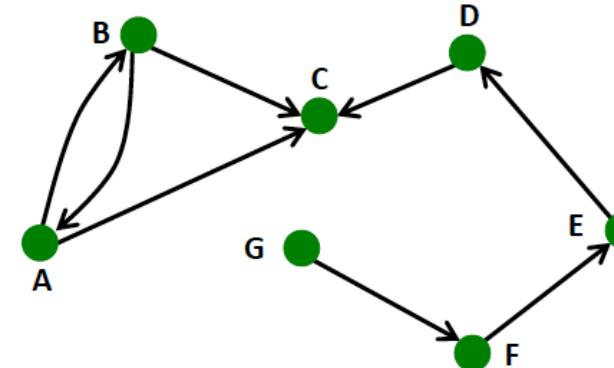
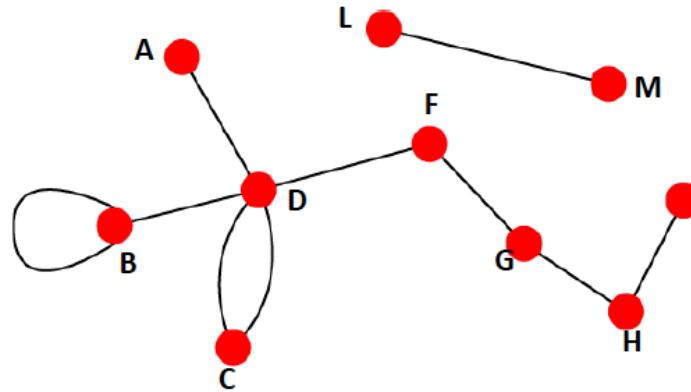
Bipartite Graphs

- Bipartite graph is a graph whose nodes can be divided into two disjoint sets U and V such that every link connects a node in U to one in V ; that is, U and V are independent sets
- Examples:
 - Authors-to-Papers (they authored)
 - Actors-to-Movies (they appeared in)
 - Users-to-Movies (they rated)
 - Recipes-to-Ingredients (they contain)
- “Folded” homogeneous networks:
 - Author collaboration networks
 - Movie co-rating networks



Directed vs Undirected Graphs

- Undirected
 - Links: undirected (symmetrical reciprocal)
- Directed
 - Links: directed



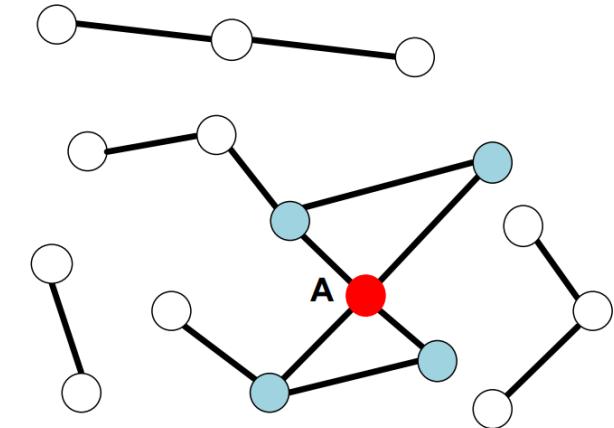
- Other considerations on links
 - Weight, types, attributes/features

Node Degrees

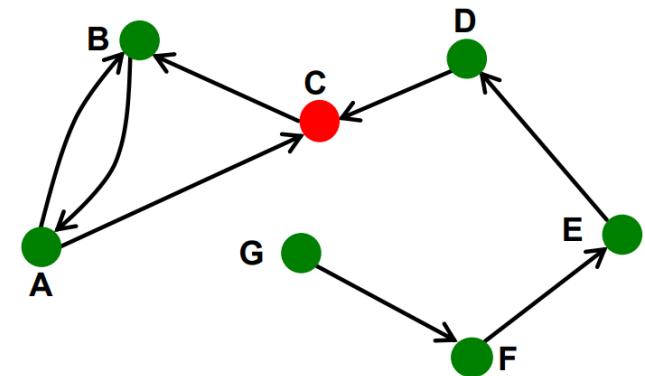
- Node degree k_i : the number of edges adjacent to node i
 - $k_A=4$
- In directed networks we define an **in-degree** and **out-degree**
- The (total) degree of a node is the sum of in- and out-degree

$$k_c^{in} = 2, \ k_c^{out} = 1, \ k_c = 3$$

Undirected



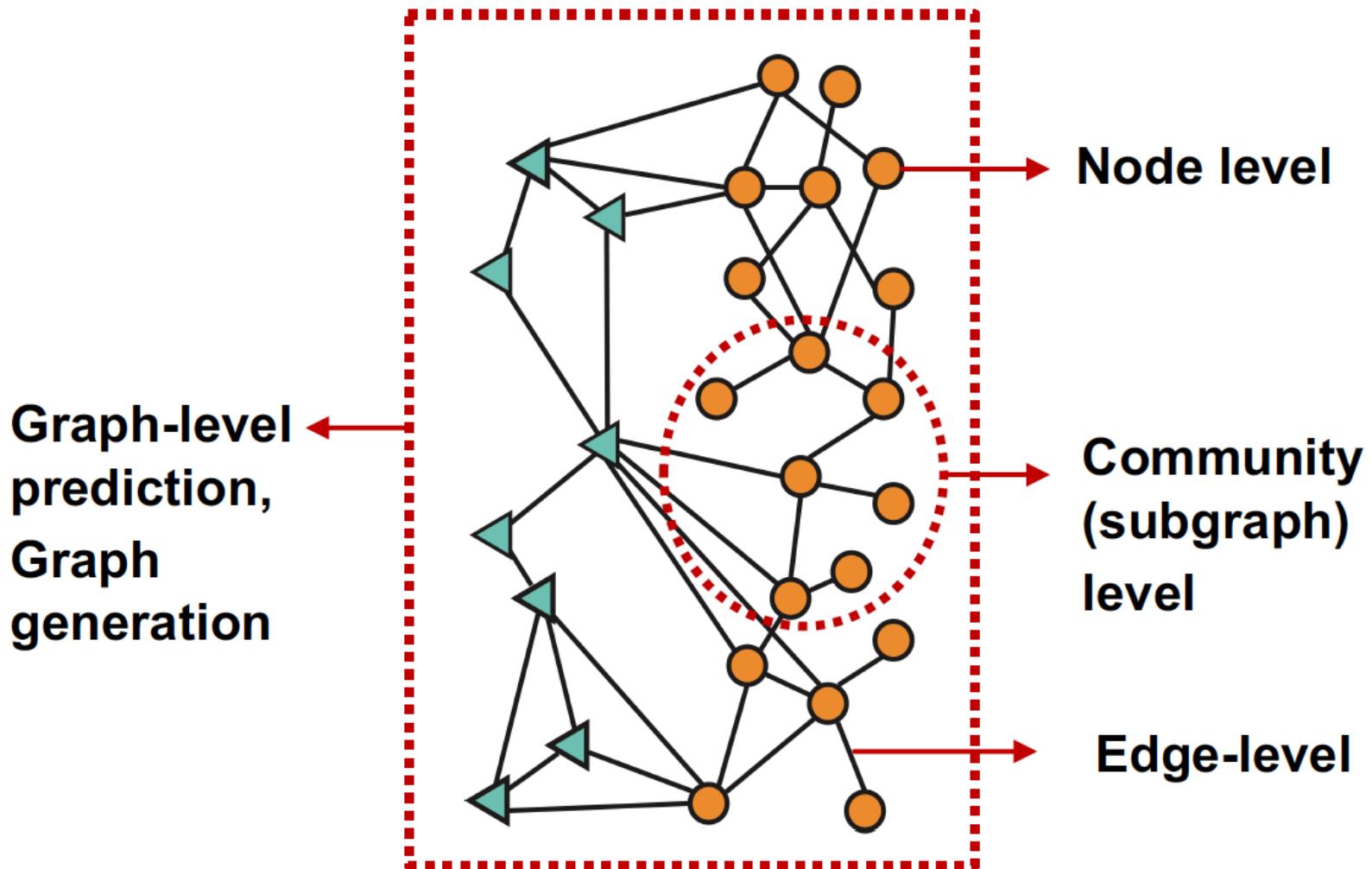
Directed



Choosing a Proper Representation

- How to build a graph:
 - What are nodes?
 - What are edges?
- Choice of the proper network representation of a given domain/problem determines our ability to use networks successfully:
 - In some cases, there is a unique, unambiguous representation
 - In other cases, the representation is by no means unique
 - The way you assign links will determine the nature of the question you can study

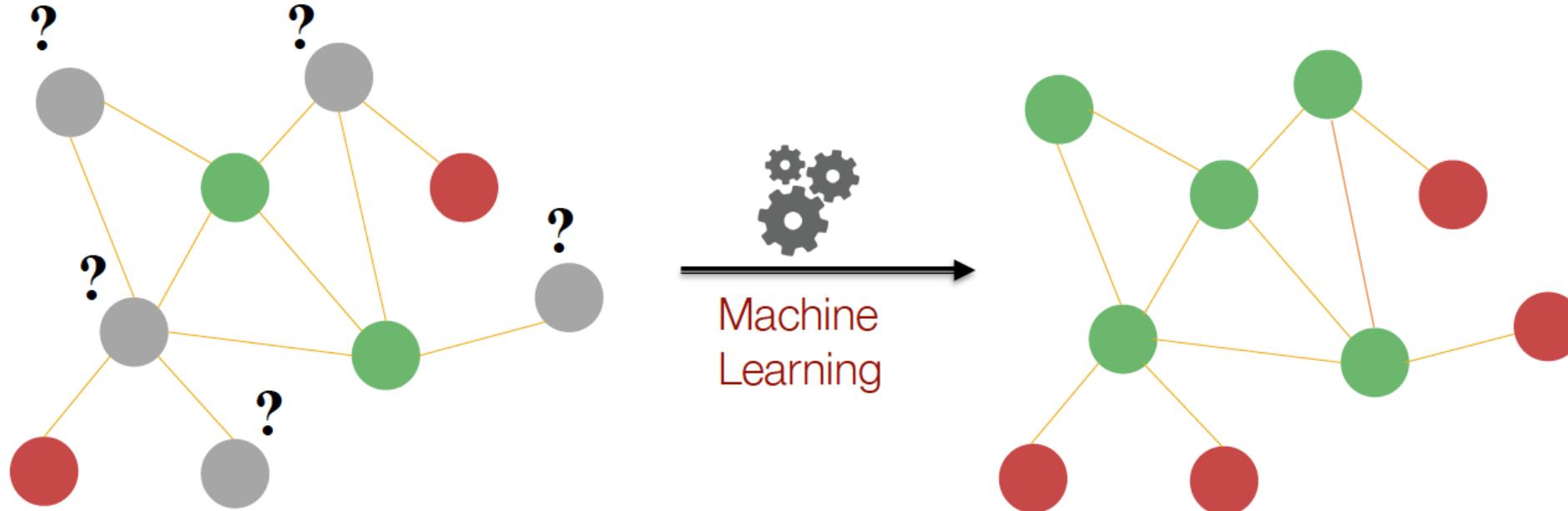
Different Types of Tasks



Graph Analysis Tasks

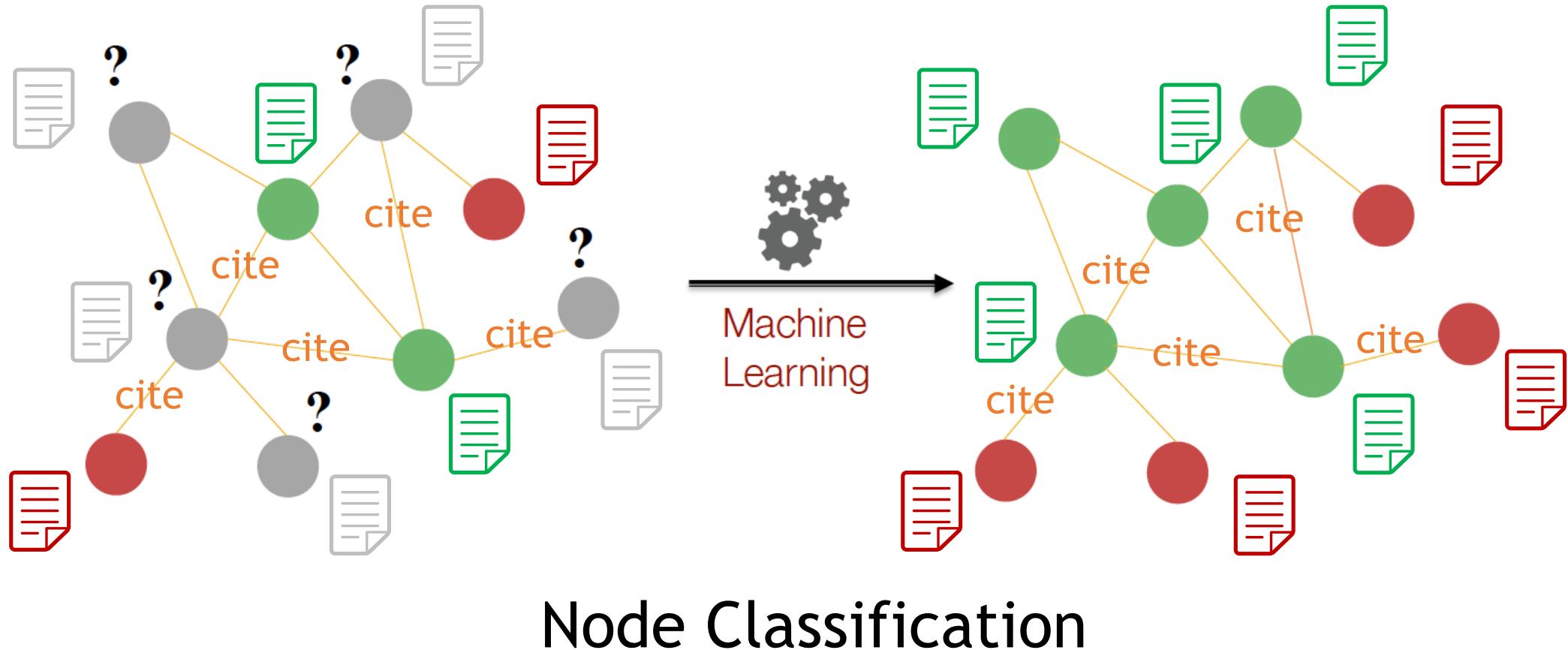
- **Node classification:** predict a property of a node
 - Example: predict the research area of a paper
- **Link prediction:** predict whether there is a link between two nodes
 - Example: friendship prediction in social networks
- **Graph classification:** categorize different graphs
 - Example: Molecule property prediction
- **Clustering:** detect if some nodes form a community
 - Example: social circle detection
- Other tasks:
 - Graph generation: drug discovery
 - Graph evolution: physical simulation

Node-Level Task



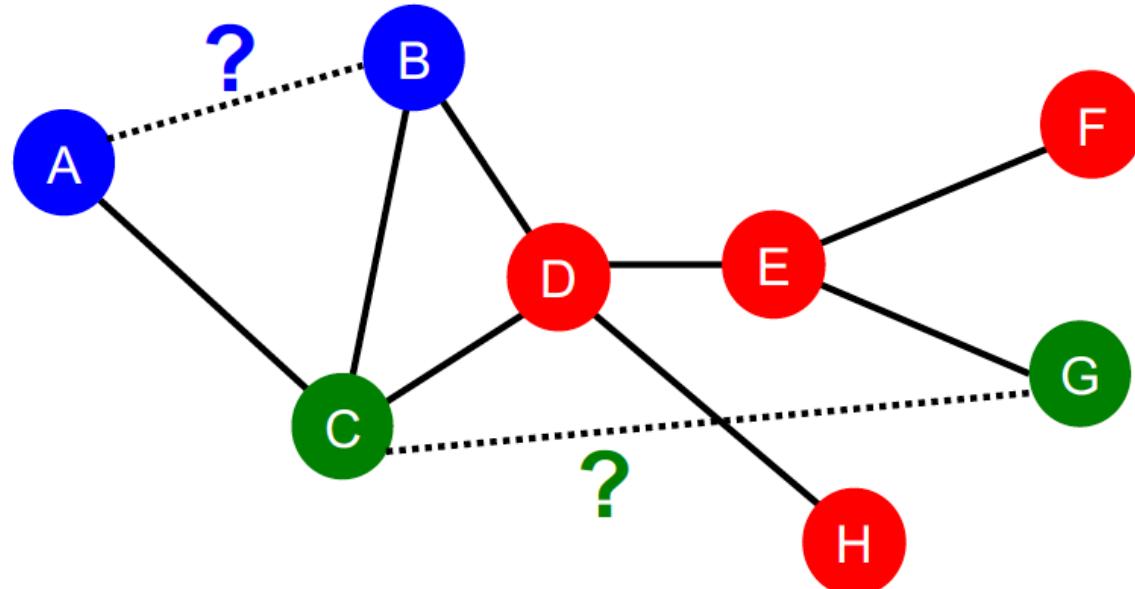
Node Classification

Example: Document Classification



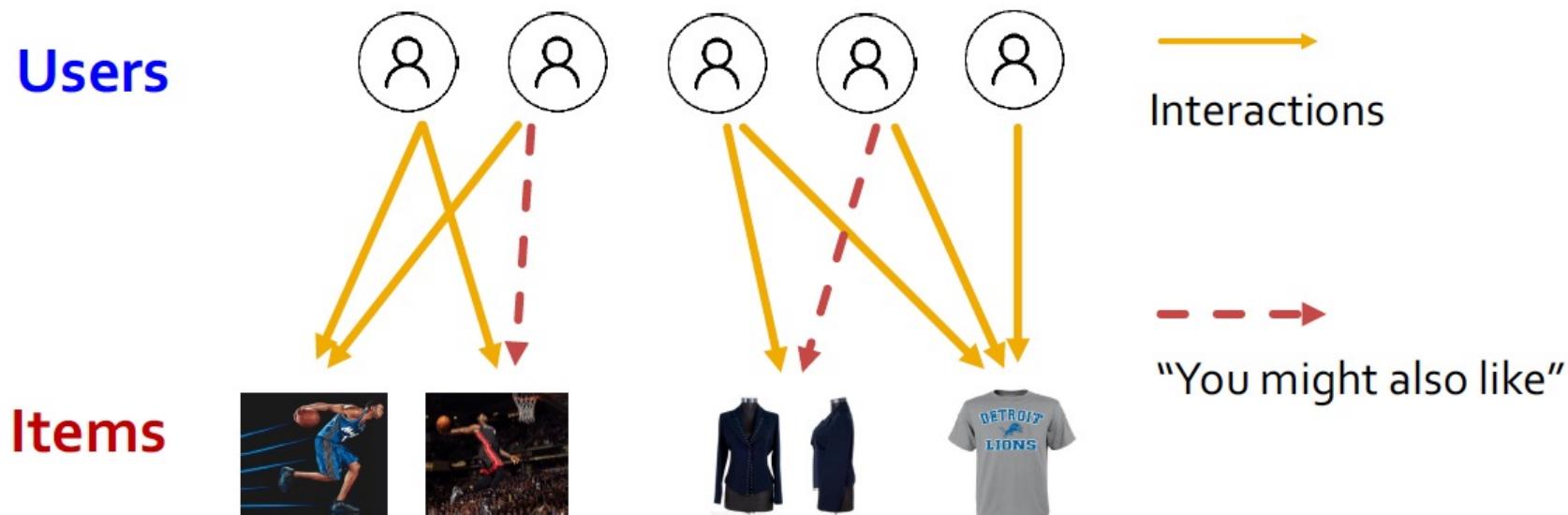
Link-Level Task

- The task is to predict new/missing/unknown links based on the existing links
- At test time, node pairs (with no existing links) are ranked, and top K node pairs are predicted
- Task: Make a prediction for **a pair of nodes**



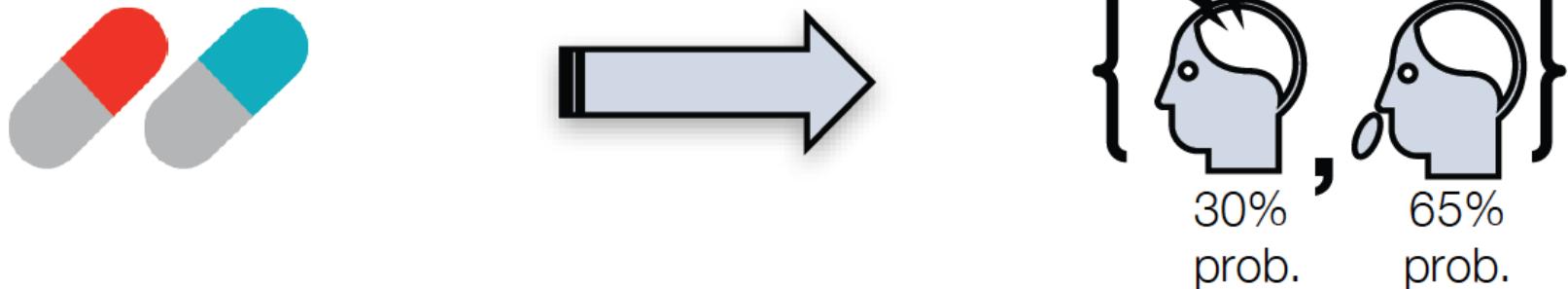
Example: Recommender Systems

- Users interacts with items
 - Watch movies, buy merchandise, listen to music
 - **Nodes**: Users and items
 - **Edges**: User-item interactions
- **Goal**: Recommend items users might like



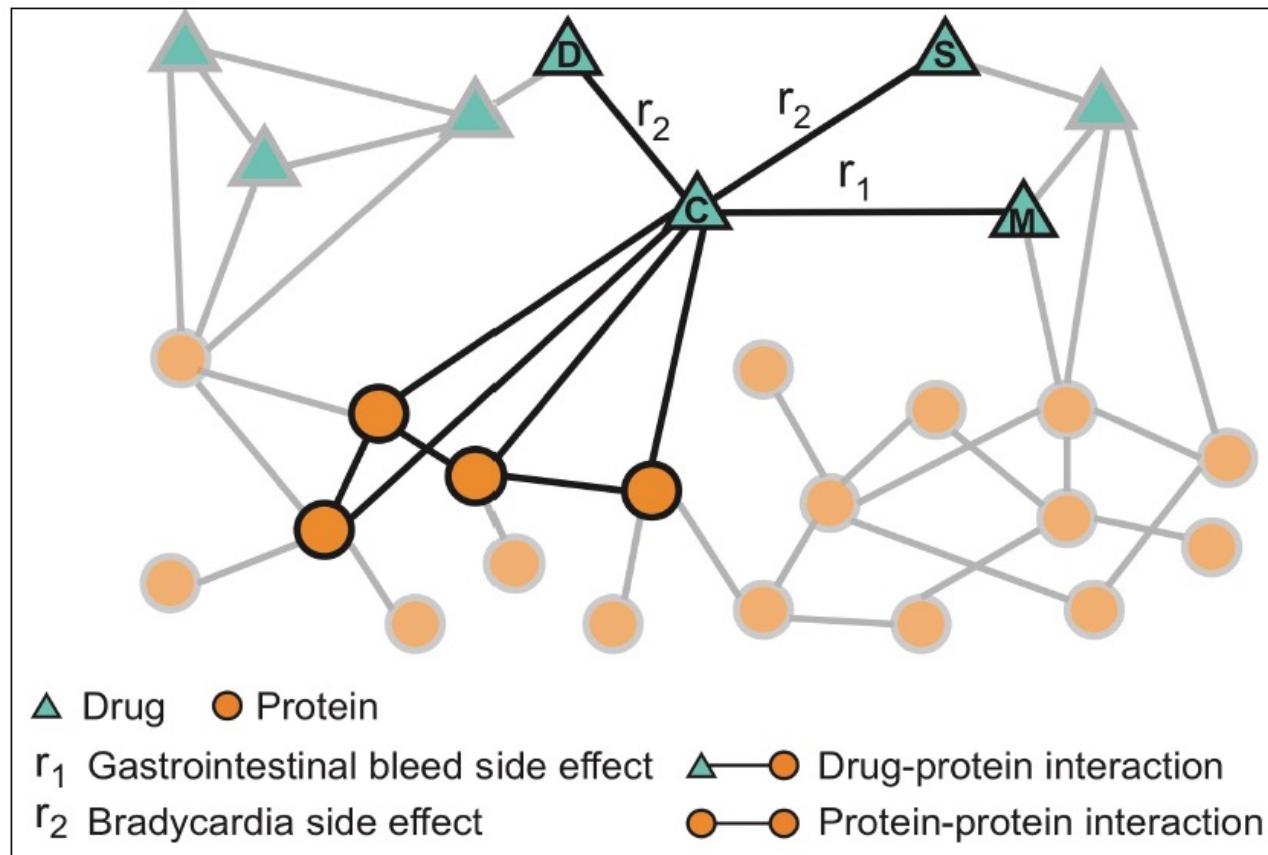
Example: Drug Side Effects

- Many patients **take multiple drugs** to treat **complex or co-existing diseases**:
 - 46% of people ages 70-79 take more than 5 drugs
 - Many patients take more than 20 drugs to treat heart disease, depression, insomnia, etc.
- **Task:** Given a pair of drugs predict adverse side effects

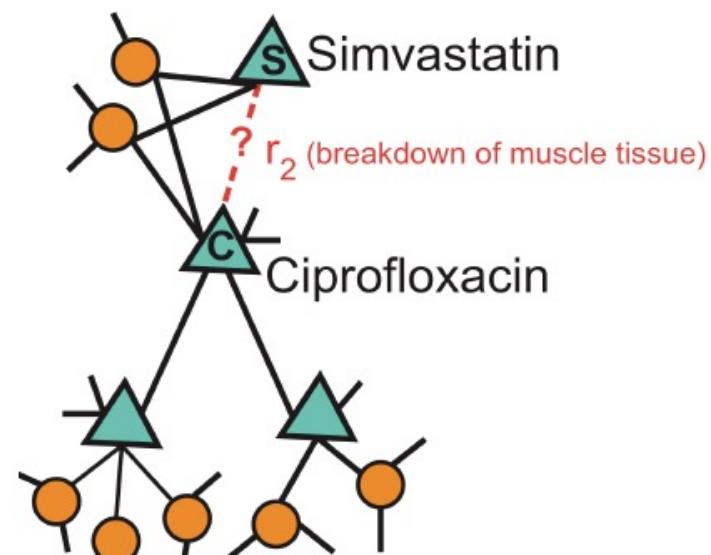


Example: Drug Side Effects

- **Nodes:** Drugs and Proteins
- **Edges:** Interactions

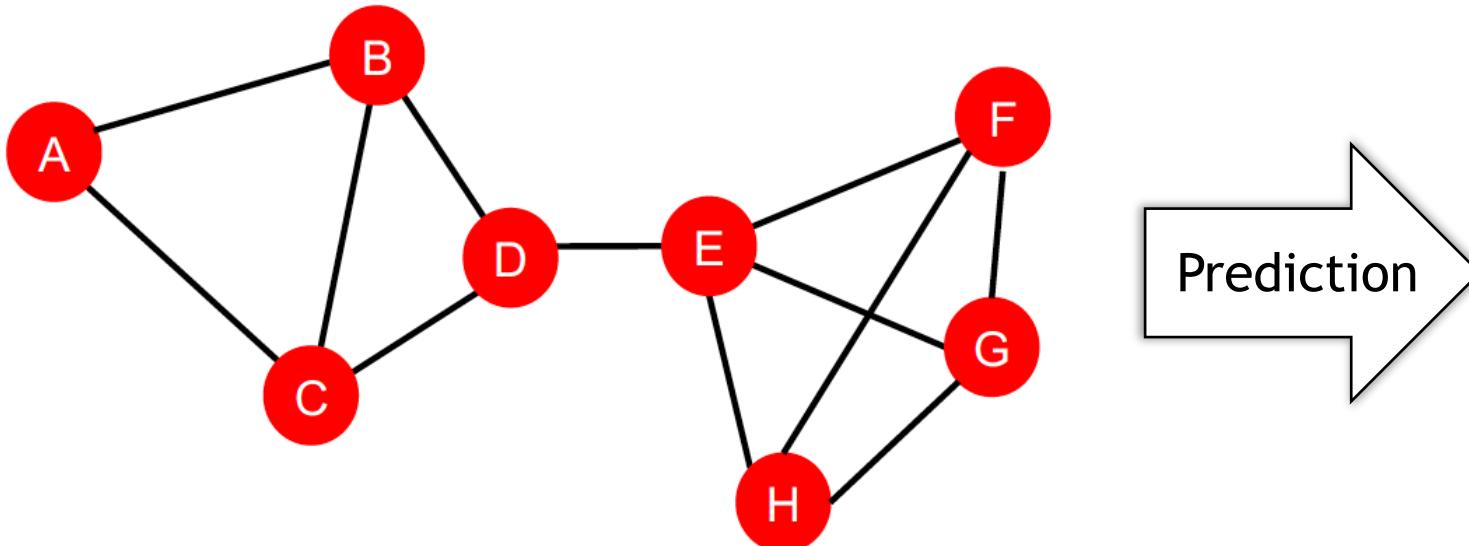


- **Query:** How likely will Simvastatin and Ciprofloxacin, when taken together, break down muscle tissue?



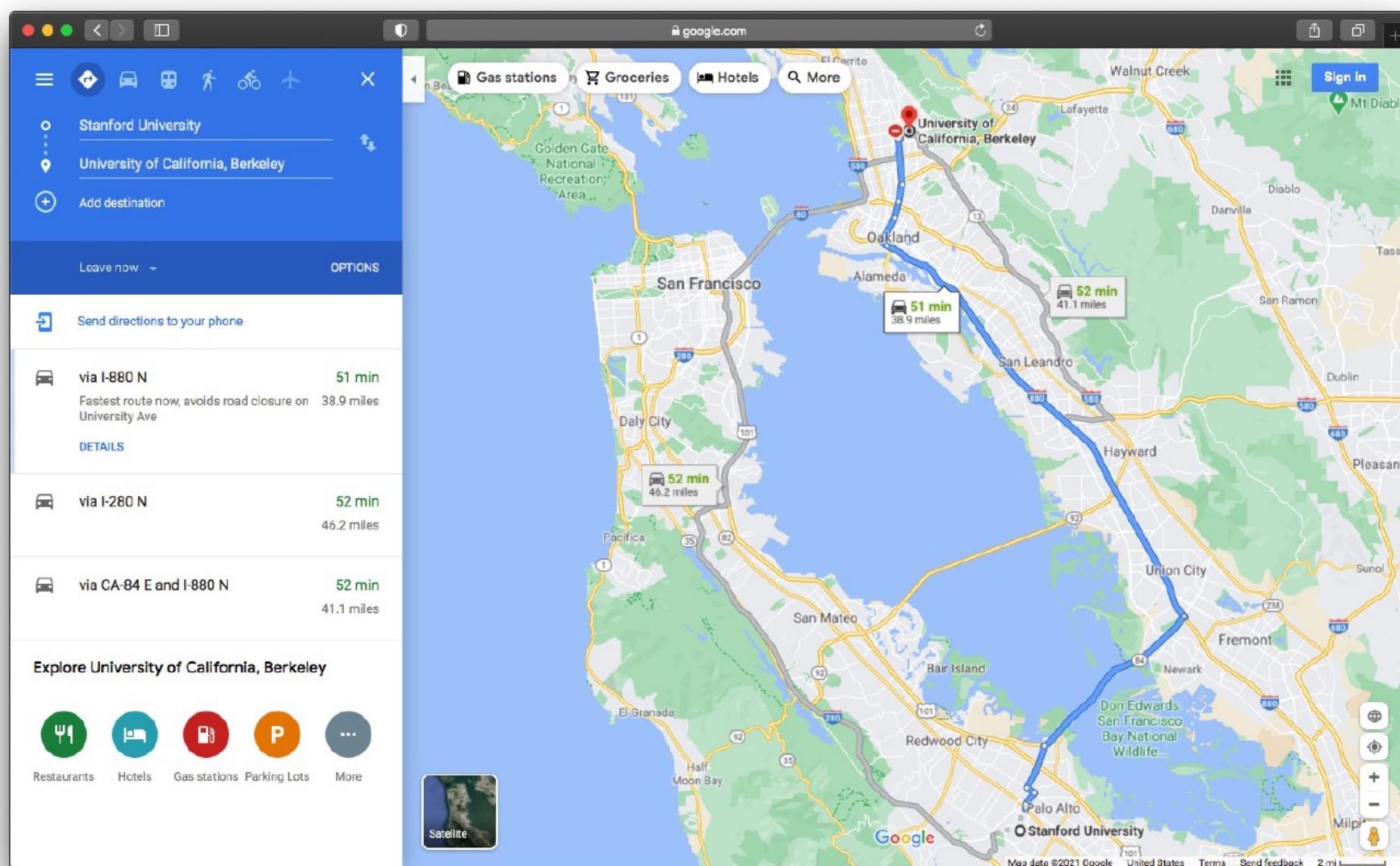
Graph-Level Task

- Goal: make a prediction for an entire graph or a subgraph of the graph



Property/Class of
the Graph

Example: Traffic Prediction



Example: Traffic Prediction

- **Nodes:** road segments
- **Edges:** connectivity between road segments
- **Prediction:** time of arrival (ETA)

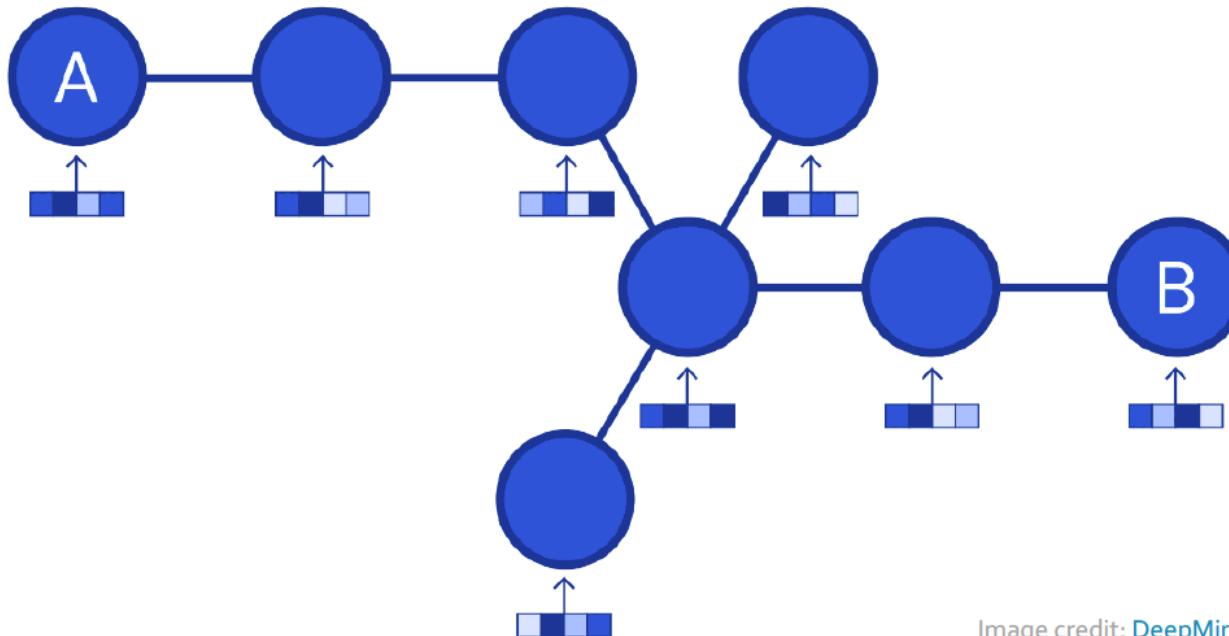


Image credit: [DeepMind](#)

Example: Drug Classification

- Chemical compounds are small molecular graphs
- Nodes: atoms, Edges: chemical bonds
- Classes: for example, mutagenic effect on a bacterium

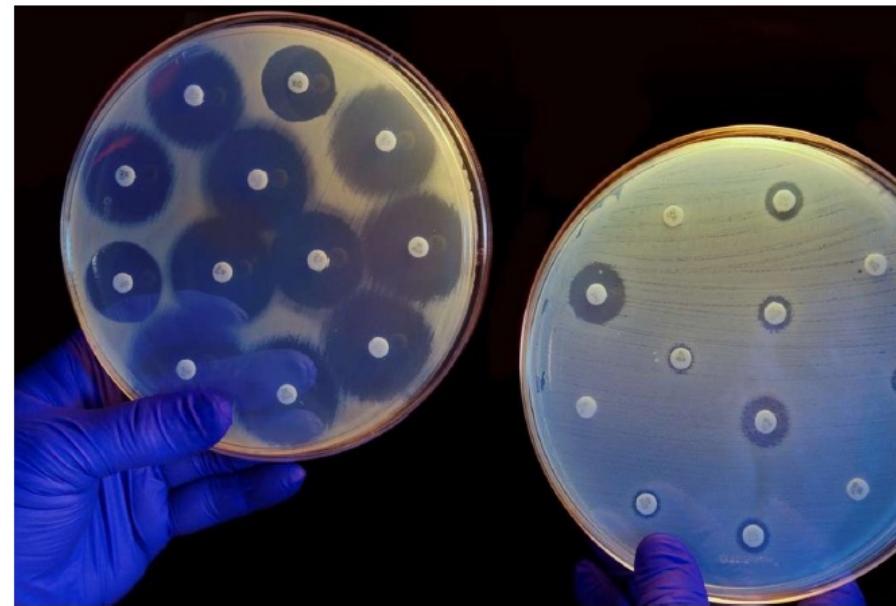
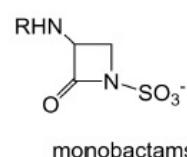
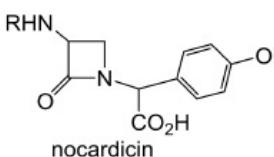
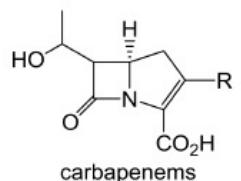
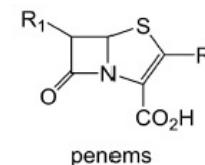
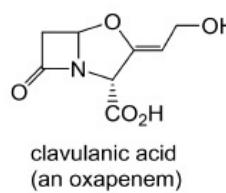
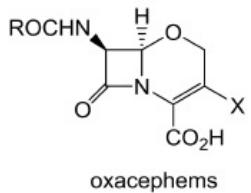
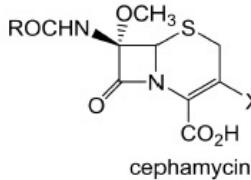
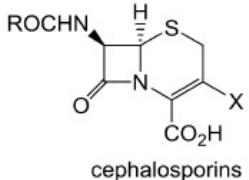
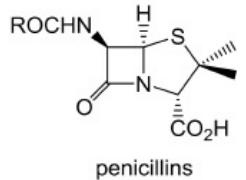


Image credit: [CNN](#)

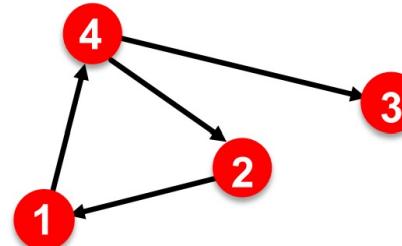
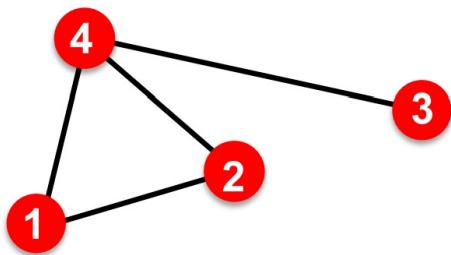
Konaklieva, Monika I. "Molecular targets of β -lactam-based antimicrobials: beyond the usual suspects." *Antibiotics* 3.2 (2014): 128-142.

How to Represent a Graph

- Adjacency Matrix
- Edge List
- Adjacency List

Adjacency Matrix

- $A_{ij}=1$ if there is a link from node i to node j
- $A_{ij}=0$ otherwise



$$A = \begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix}$$

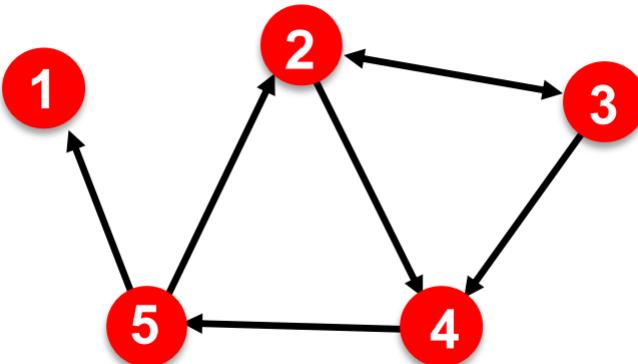
$$A = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \end{pmatrix}$$

Note that for a directed graph (right) the matrix is not symmetric

Edge List

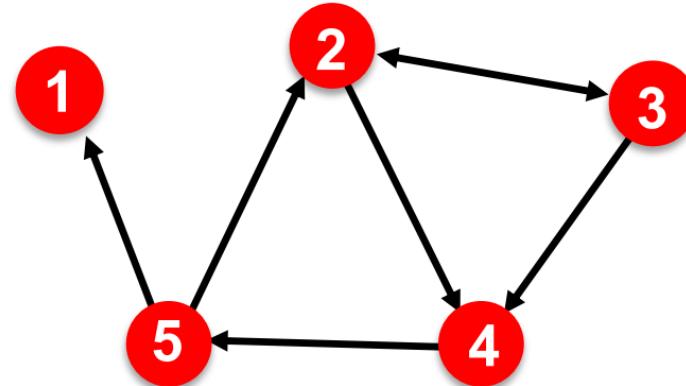
- Represent graph as a list of edges:

- (2,3)
- (2, 4)
- (3, 2)
- (3, 4)
- (4, 5)
- (5, 2)
- (5, 1)



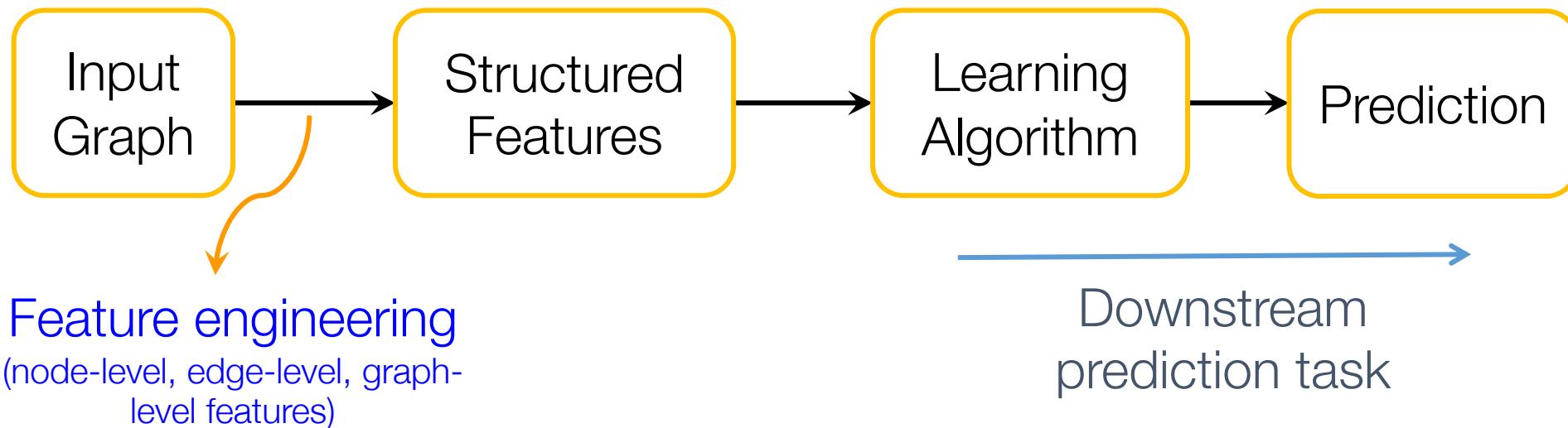
Adjacency List

- Easier to work with if network is
 - Large
 - Sparse
- Allows us to quickly retrieve all neighbors of a given node
 - Node: neighboring nodes
 - 1:
 - 2: 3, 4
 - 3: 2, 4
 - 4: 5
 - 5: 1, 2



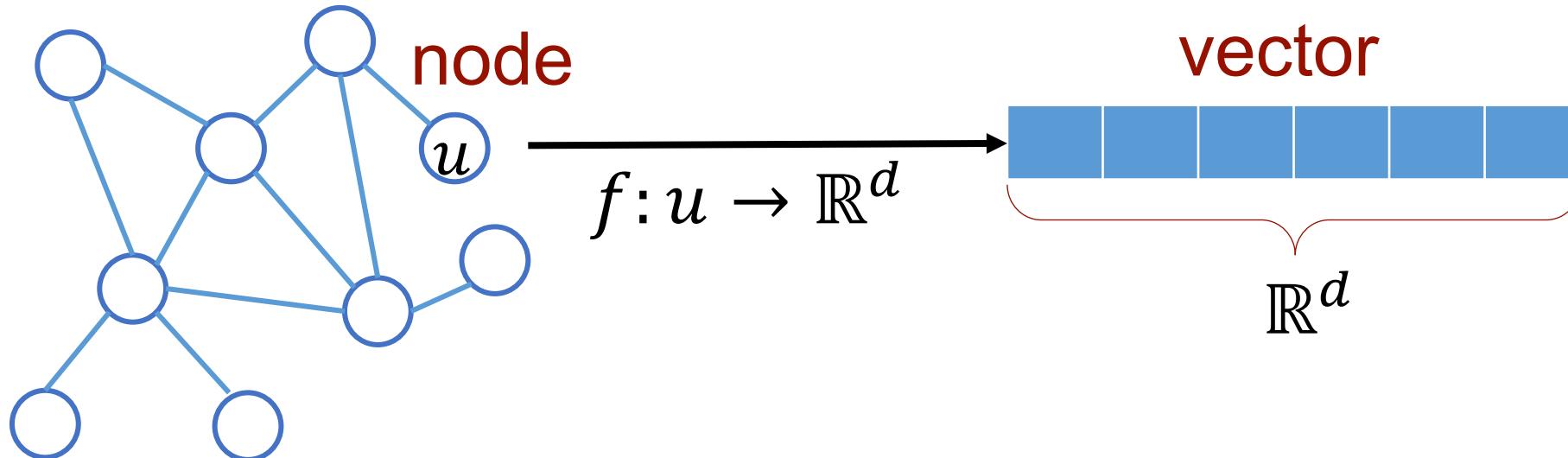
Traditional ML for Graphs

- Given an input graph, extract node, link and graph-level features, then learn a model (SVM, neural network, etc.) that maps features to labels



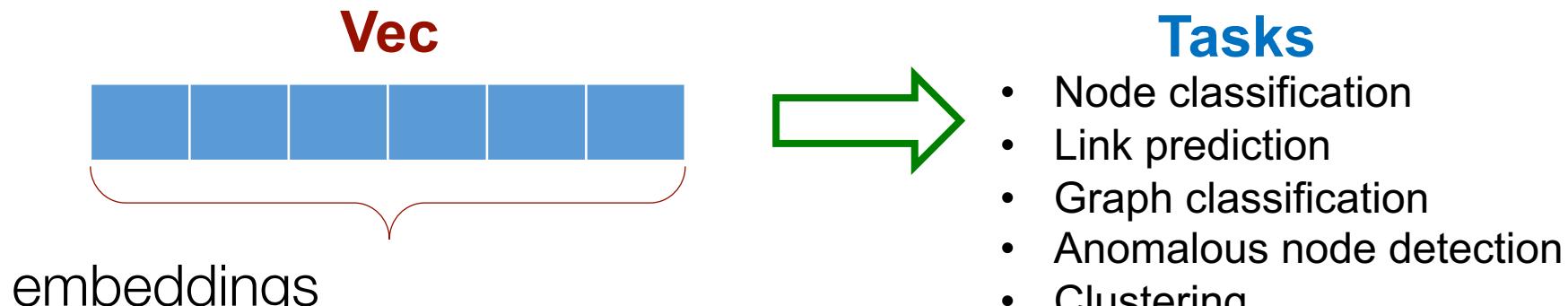
Graph Representation Learning

- Goal: Efficient task-independent feature learning for machine learning with graphs!



Why Embedding?

- Task: Map nodes into an embedding space
 - Similarity of embeddings between nodes indicates their similarity in the network. For example:
 - Both nodes are close to each other (connected by an edge)
 - Encode network information
 - Potentially used for many downstream predictions



Node Embedding Example

- 2D embedding of nodes of the Zachary’s Karate Club network:
 - Social relationships among the 34 individuals in the karate club studied by Zachary
 - Links represent pairs of members who interacted outside the club

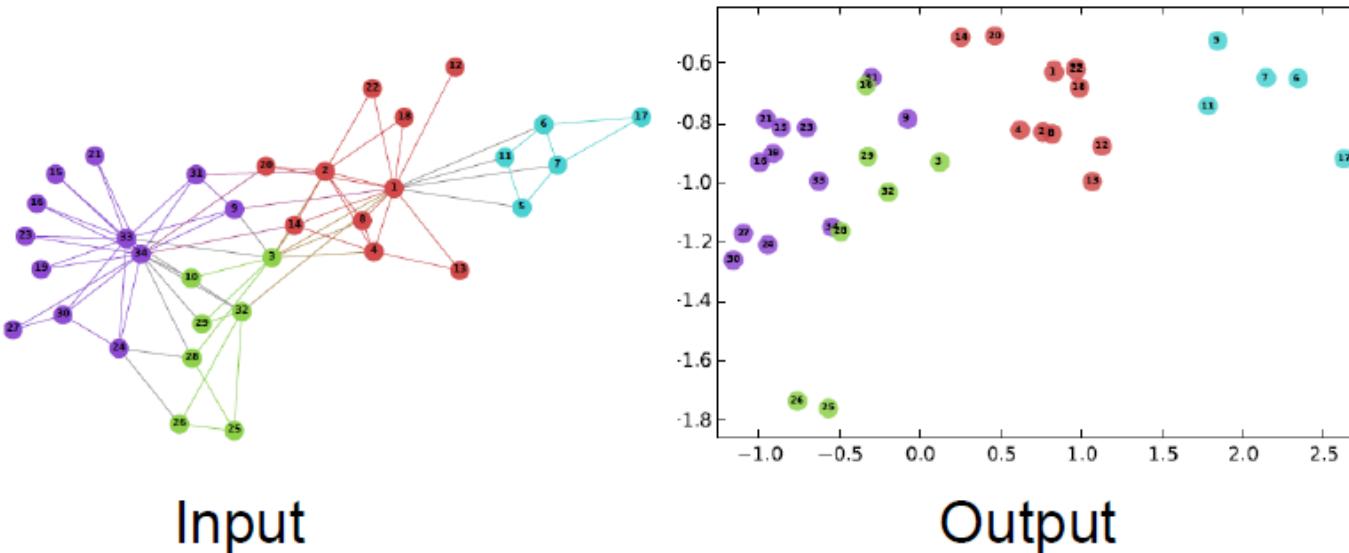
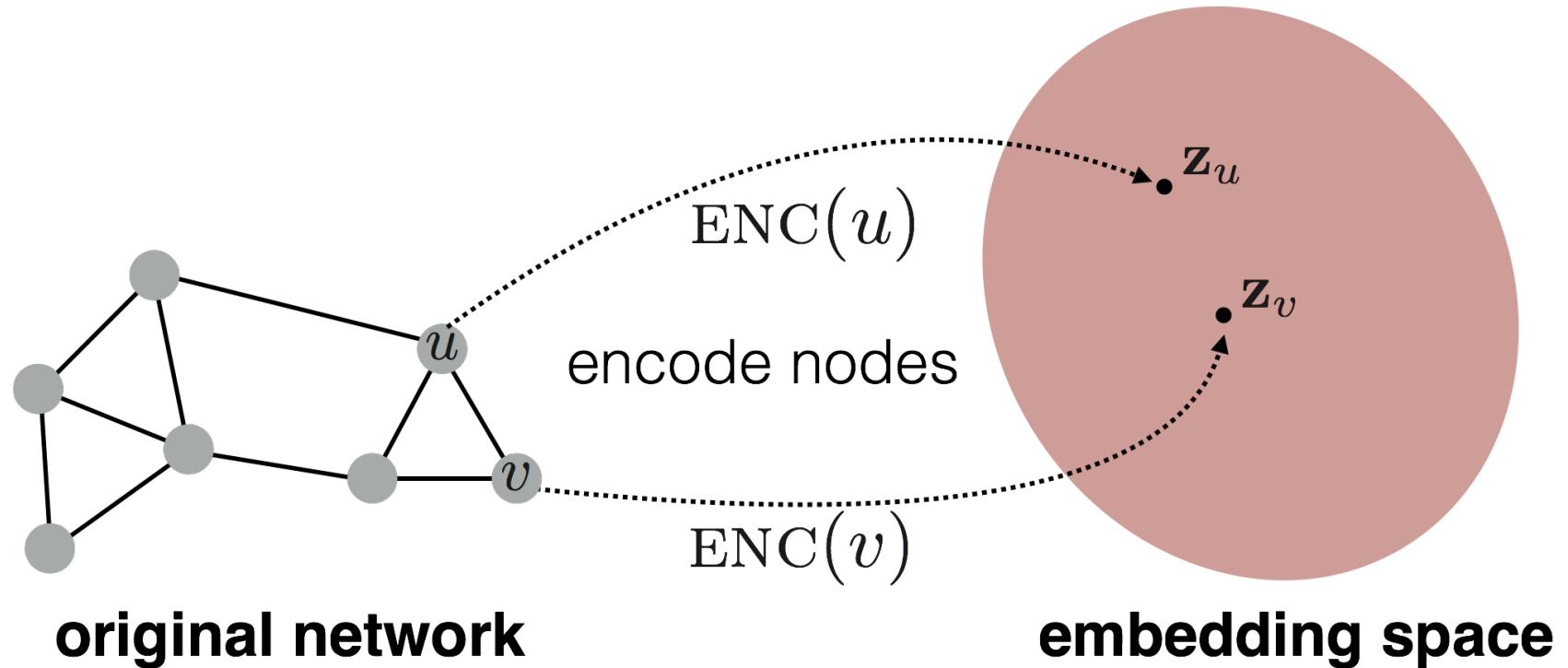


Image from: Perozzi et al. DeepWalk: Online Learning of Social Representations. KDD 2014.

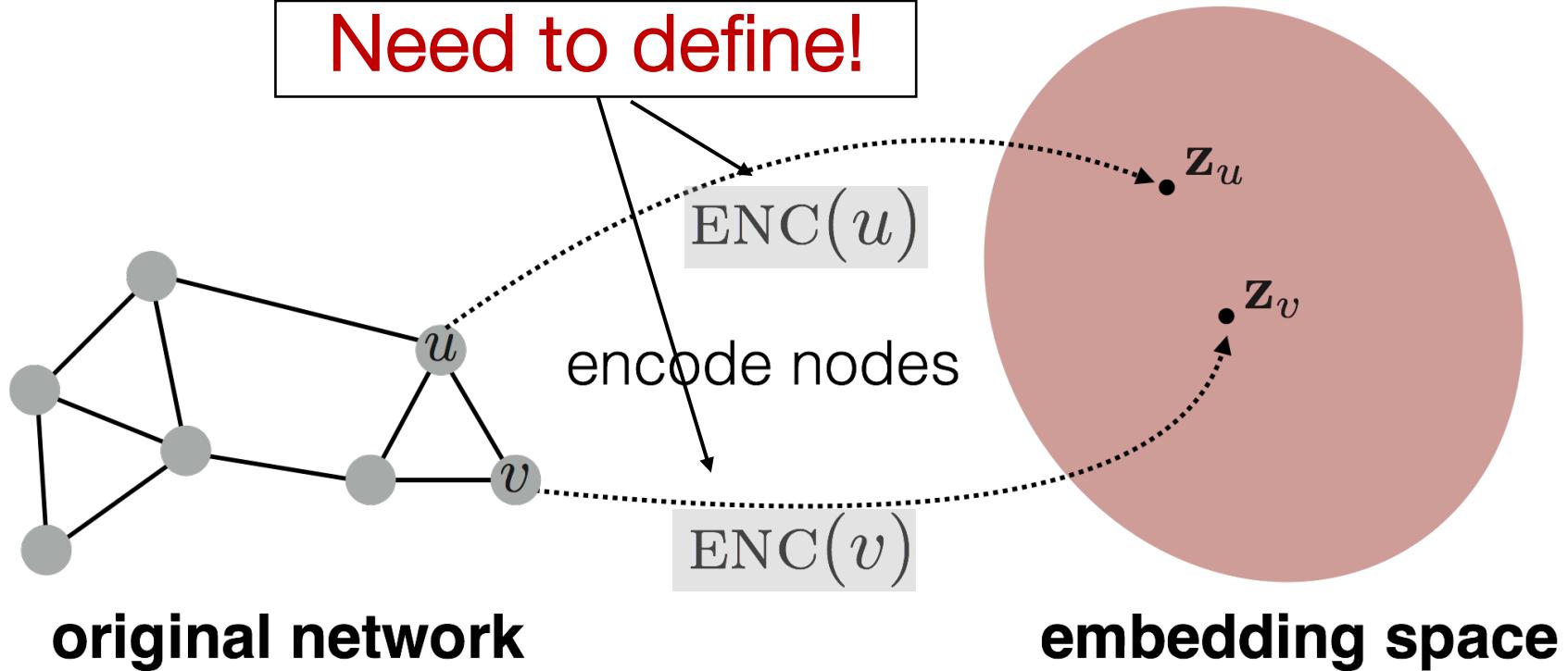
Embedding Nodes

- Goal is to encode nodes so that **similarity in the embedding space** (e.g., dot product) approximates **similarity in the graph**



Embedding Nodes

Goal: $\text{similarity}(u, v) \approx \mathbf{z}_v^T \mathbf{z}_u$
in the original network Similarity of the embedding



Learning Node Embeddings

1. Encoder maps from nodes to embeddings
2. Define a node similarity function
 - i.e., a measure of similarity in the original network
3. Decoder DEC maps from embeddings to the similarity score
4. Optimize the parameters of the encoder so that:

DEC($\mathbf{z}_v^T \mathbf{z}_u$)

$$\text{similarity}(u, v) \approx \mathbf{z}_v^T \mathbf{z}_u$$

in the original network Similarity of the embedding

Two Key Components

- **Encoder:** maps each node to a low-dimensional vector

$\text{ENC}(v) = \boxed{\mathbf{z}_v}$ *d*-dimensional embedding
node in the input graph

- **Similarity function:** specifies how the relationships in vector space map to the relationships in the original network

$\text{similarity}(u, v) \approx \mathbf{z}_v^T \mathbf{z}_u$ **Decoder**
Similarity of u and v in the original network
dot product between node embeddings

“Shallow” Encoding

- Simplest encoding approach: Encoder is just an embedding-lookup

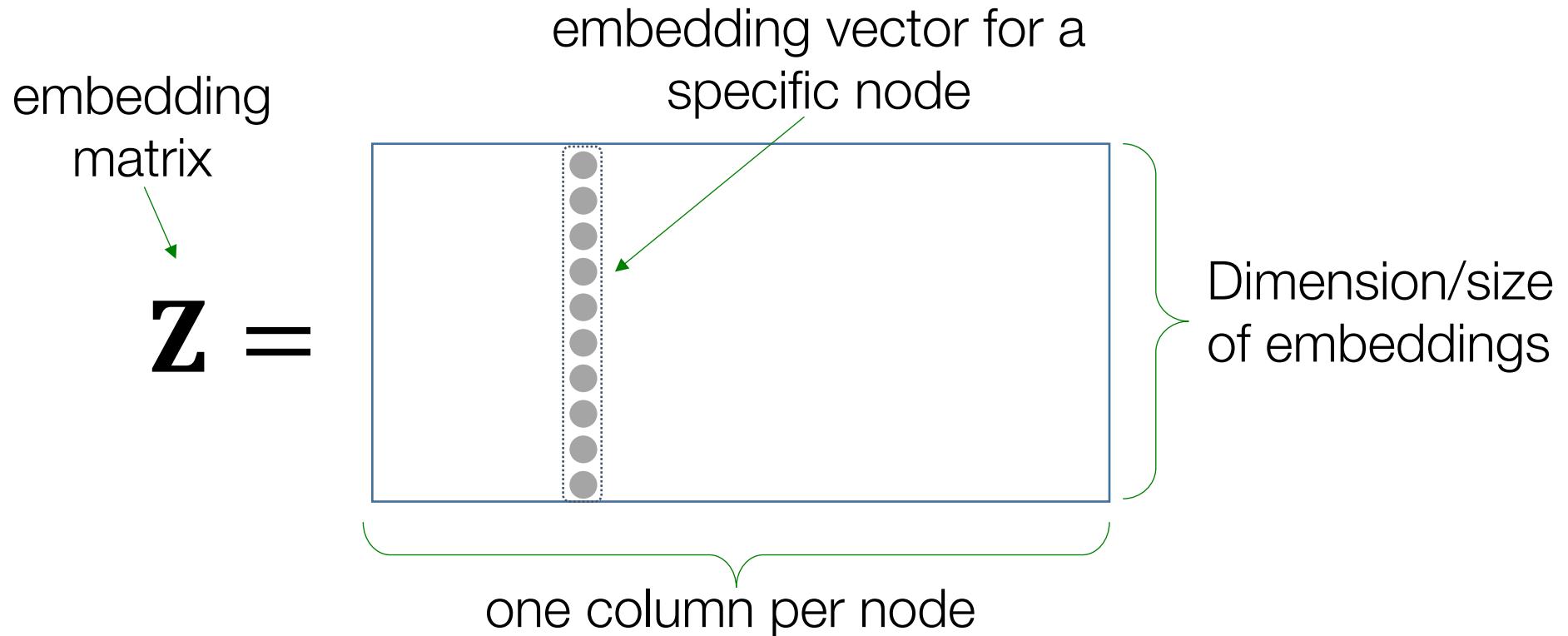
$$\text{ENC}(v) = \mathbf{z}_v = \mathbf{Z} \cdot v$$

$\mathbf{Z} \in \mathbb{R}^{d \times |\mathcal{V}|}$ matrix, each column is a node embedding [what we learn / optimize]

$v \in \mathbb{I}^{|\mathcal{V}|}$ indicator vector, all zeroes except a one in column indicating node v
(One-hot representation of a node)

“Shallow” Encoding

- Simplest encoding approach: Encoder is just an embedding-lookup



“Shallow” Encoding

- Simplest encoding approach: Encoder is just an embedding-lookup
- Each node is assigned a unique embedding vector
 - i.e., we directly optimize the embedding of each node
- Many methods: DeepWalk, node2vec

Framework Summary

- Encoder + Decoder Framework
 - Shallow encoder: Embedding lookup
 - Parameters to optimize: Z which contains node embeddings z_u for all nodes $u \in V$
 - We will cover deep encoders in the GNNs
- Decoder: based on node similarity.
- Objective: maximize $z_v^T z_u$ for node pairs (u, v) that are **similar**

How to Define Node Similarity?

- Key choice of methods is **how they define node similarity**
- Should two nodes have a similar embedding if they ...
 - **are linked?**
 - **share neighbors?**
 - **have similar “structural roles”?**
- We will now learn node similarity definition that uses **random walks**, and how to optimize embeddings for such a similarity measure

Note on Node Embeddings

- This is **unsupervised/self-supervised** way of learning node embeddings
 - We are **not** utilizing node labels
 - We are **not** utilizing node features
 - The goal is to directly estimate a set of coordinates (i.e., the embedding) of a node so that some aspect of the network structure (captured by DEC) is preserved
- These embeddings are **task independent**:
 - They are not trained for a specific task but can be used for any task (e.g., classification, link prediction, clustering, etc.)



COSC 3337
Data Science I
Section 14623

Graph Data Analysis

Instructor: Jingchao Ni
Fall 2024

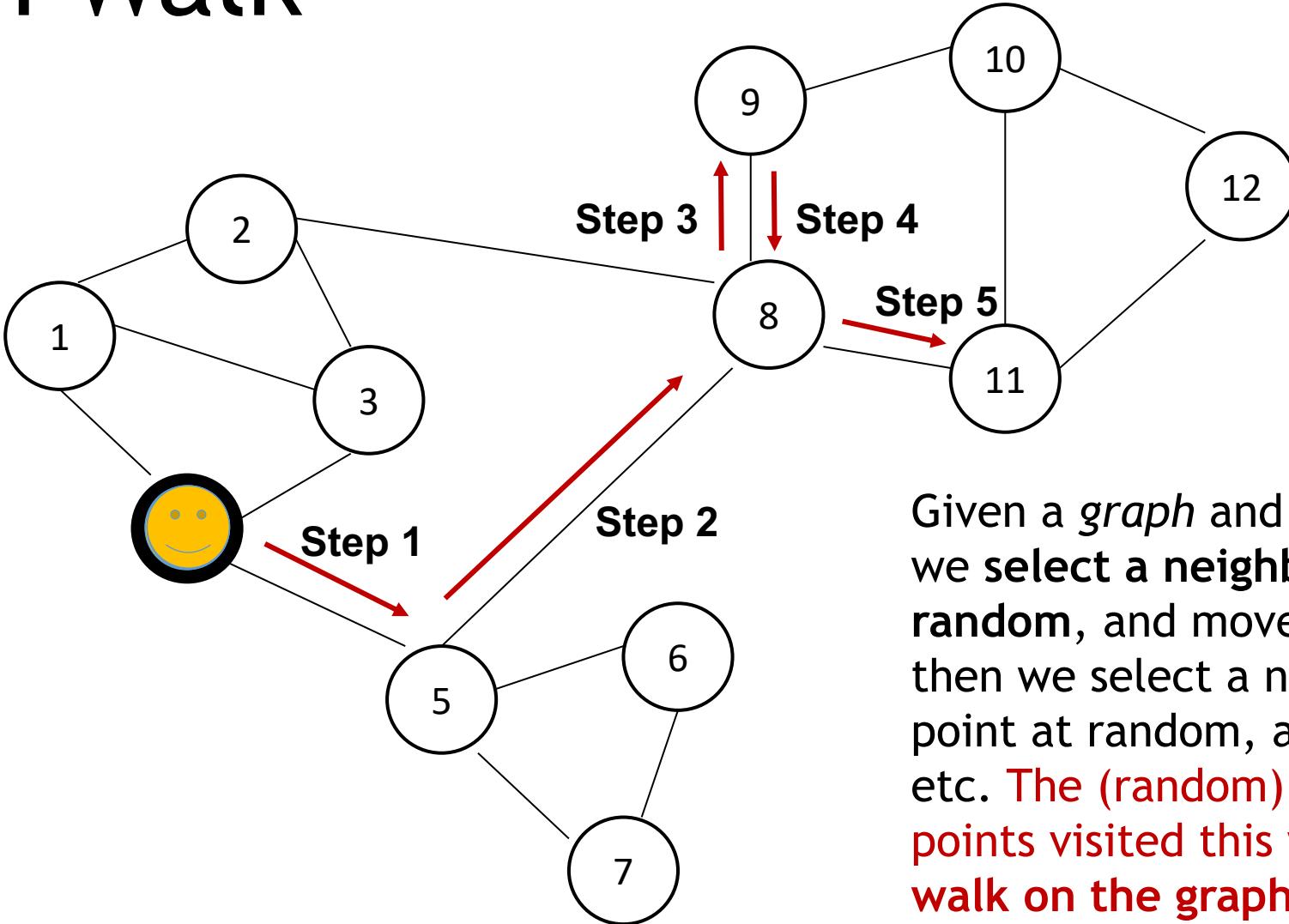
Random Walk Method: Notation

- **Vector \mathbf{z}_u :**
 - The embedding of node u (what we aim to find)
 - **Probability $P(v \mid \mathbf{z}_u)$:**  Our model prediction based on \mathbf{z}_u
 - The (predicted) probability of visiting node v on random walks starting from node u
-

Non-linear functions are used to produce predicted probabilities

- **Softmax** function:
 - Turns vector of K real values (model predictions) into K probabilities that sum to 1: $S(\mathbf{z})[i] = \frac{e^{\mathbf{z}[i]}}{\sum_{j=1}^K e^{\mathbf{z}[j]}}$
- **Sigmoid** function:
 - S-shaped function that turns real values into range of $(0, 1)$: $\sigma(x) = \frac{1}{1+e^{-x}}$

Random Walk



Given a *graph* and a *starting point*, we select a **neighbor** of it at **random**, and move to this neighbor; then we select a neighbor of this point at random, and move to it, etc. **The (random) sequence of points visited this way is a random walk on the graph.**

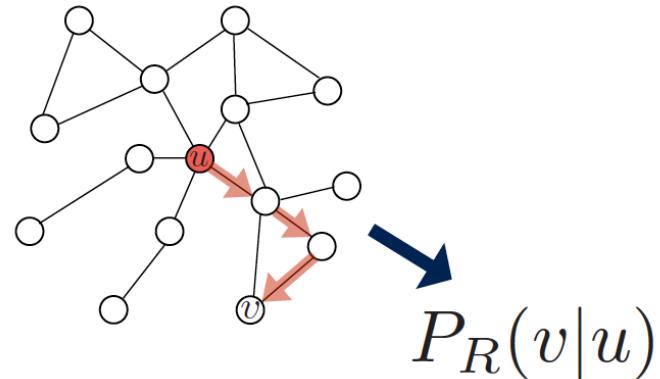
Random-Walk Embeddings

$$\mathbf{z}_v^T \mathbf{z}_u \approx \text{similarity}(u, v)$$

$\mathbf{z}_u^T \mathbf{z}_v \approx$ probability that u and v co-occur on a random walk over the graph

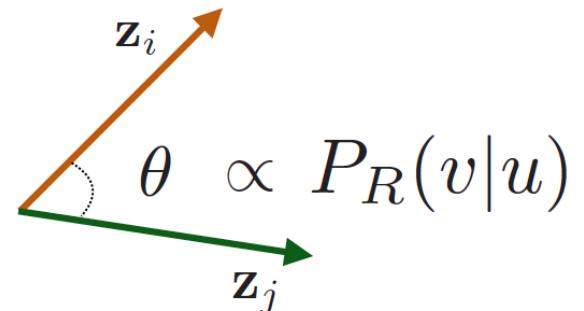
Random-Walk Embeddings

- Estimate probability of visiting node v on a random walk starting from node u using some random walk strategy R



- Optimize embeddings to encode these random walk statistics:

Similarity in embedding space (Here:
dot product= $\cos(\theta)$) encodes random walk “similarity”



Why Random Walks?

- **Expressivity:** Flexible stochastic definition of node similarity that incorporates both local and higher-order neighborhood information
 - Idea: if random walk starting from node u visits v with high probability, u and v are similar (high-order multi-hop information)
- **Efficiency:** Do not need to consider all node pairs when training; only need to consider pairs that co-occur on random walks

Unsupervised Feature Learning

- **Intuition:** Find embedding of nodes in d -dimensional space that preserves similarity
- **Idea:** Learn node embedding such that **nearby** nodes are close together in the network
- Given a node u , how do we define nearby nodes?
 - $N_R(u)$... neighbourhood of u obtained by some **random walk strategy R**

Feature Learning as Optimization

- Given $G = (V, E)$
- Our goal is to learn a mapping $f: u \rightarrow \mathbb{R}^d$, i.e., $f(u) = \mathbf{z}_u$
- Log-likelihood objective:

$$\arg \max_z \sum_{u \in V} \log P(N_R(u) | \mathbf{z}_u)$$

- $N_R(u)$ is the neighborhood of node u by strategy R
- Given node u , we want to learn feature representations that are predictive of the nodes in its random walk neighborhood $N_R(u)$

Random Walk Optimization

1. Run **short fixed-length random walks** starting from each node u in the graph using some random walk strategy R
2. For each node u collect $N_R(u)$, the multiset of nodes visited on random walks starting from u
 - $N_R(u)$ can have repeat elements since nodes can be visited multiple times on random walks
3. Optimize embeddings according to: Given node u , predict its neighbors

$$\arg \max_z \sum_{u \in V} \log P(N_R(u) | \mathbf{z}_u) \quad \Rightarrow \text{Maximum likelihood objective}$$

Random Walk Optimization

Equivalently

$$\arg \max_z \sum_{u \in V} \log P(N_R(u) | z_u) \implies \text{Maximum likelihood objective}$$
$$\arg \min_z \mathcal{L} = \sum_{u \in V} \sum_{v \in N_R(u)} -\log(P(v | z_u))$$

- **Intuition:** Optimize embeddings to minimize the negative log-likelihood of random walk neighborhoods $N(u)$
- Parameterize $P(v | z_u)$ using softmax: $P(v|z_u) = \frac{\exp(z_u^T z_v)}{\sum_{n \in V} \exp(z_u^T z_n)}$

Why softmax?

We want node v to be most similar to node u (out of all nodes)

Random Walk Optimization

- Putting it all together

$$\mathcal{L} = \sum_{u \in V} \left[\sum_{v \in N_R(u)} - \log\left(\frac{\exp(\mathbf{z}_u^T \mathbf{z}_v)}{\sum_{n \in V} \exp(\mathbf{z}_u^T \mathbf{z}_n)}\right) \right]$$

sum over all nodes u

sum over nodes v seen on random walks starting from u

predicted probability of u and v co-occurring on random walk

Optimizing random walk embeddings =

Finding embeddings \mathbf{z}_u that minimize \mathcal{L}

Random Walk Optimization

- But doing this naively is too expensive

$$\mathcal{L} = \sum_{u \in V} \sum_{v \in N_R(u)} -\log\left(\frac{\exp(\mathbf{z}_u^T \mathbf{z}_v)}{\sum_{n \in V} \exp(\mathbf{z}_u^T \mathbf{z}_n)}\right)$$



Nested sum over nodes gives
 $O(|V|^2)$ complexity!

Random Walk Optimization

- But doing this naively is too expensive

$$\mathcal{L} = \sum_{u \in V} \sum_{v \in N_R(u)} -\log\left(\frac{\exp(\mathbf{z}_u^\top \mathbf{z}_v)}{\sum_{n \in V} \exp(\mathbf{z}_u^\top \mathbf{z}_n)}\right)$$

The normalization term from the softmax is the problem ... can we approximate it?

Negative Sampling

- Solution: Negative Sampling

$$-\log\left(\frac{\exp(\mathbf{z}_u^\top \mathbf{z}_v)}{\sum_{n \in V} \exp(\mathbf{z}_u^\top \mathbf{z}_n)}\right)$$

$$\approx \log\left(\sigma(\mathbf{z}_u^\top \mathbf{z}_v)\right) + \sum_{i=1}^k \log\left(\sigma(-\mathbf{z}_u^\top \mathbf{z}_{n_i})\right), n_i \sim P_V$$

sigmoid function $\sigma(x) = \frac{1}{1 + e^{-x}}$
(makes each term a “probability” between 0 and 1)

Why is the approximation valid?

Technically, this is a different objective. But Negative Sampling is a form of Noise Contrastive Estimation (NCE) which approx. maximizes the log probability of softmax.

New formulation corresponds to using a logistic regression (sigmoid func.) to distinguish the target node v from nodes n_i sampled from background distribution P_V .

More at <https://arxiv.org/pdf/1402.3722.pdf>

random distribution over nodes

Instead of normalizing w.r.t. all nodes, just normalize against k random “negative samples” n_i ,
Negative sampling allows for quick likelihood calculation

Negative Sampling

$$-\log\left(\frac{\exp(\mathbf{z}_u^T \mathbf{z}_v)}{\sum_{n \in V} \exp(\mathbf{z}_u^T \mathbf{z}_n)}\right)$$

random distribution
over nodes

$$\approx \log\left(\sigma(\mathbf{z}_u^T \mathbf{z}_v)\right) + \sum_{i=1}^k \log\left(\sigma(-\mathbf{z}_u^T \mathbf{z}_{n_i})\right), n_i \sim P_V$$

- Sample k negative nodes n_i with probability proportional to its degree
- Two considerations for k (# negative samples):
 - Higher k gives more robust estimates
 - Higher k corresponds to higher bias on negative events
 - In practice, $k=5$ to 20

Can negative sample be any node or only the nodes not on the walk?
People often sample any node (for efficiency).

Stochastic Gradient Descent

- After we obtained the objective function, how do we optimize (minimize) it?

$$\mathcal{L} = \sum_{u \in V} \sum_{v \in N_R(u)} -\log(P(v|\mathbf{z}_u))$$

- Gradient Descent: a simple way to minimize \mathcal{L}

- Initialize \mathbf{z}_u at some randomized value for all nodes u
- Iterate until convergence:

- For all u , compute the derivative $\frac{\partial \mathcal{L}}{\partial z_u}$

η : learning rate

- For all u , make a step in reverse direction of derivative $z_u \leftarrow z_u - \eta \frac{\partial \mathcal{L}}{\partial z_u}$

Stochastic Gradient Descent

- **Stochastic Gradient Descent:** Instead of evaluating gradients over all examples, evaluate it for each **individual** training example
 - Initialize \mathbf{z}_u at some randomized value for all nodes u
 - Iterate until convergence:

$$\mathcal{L}^{(u)} = \sum_{v \in N_R(u)} -\log(P(v|\mathbf{z}_u))$$

- Sample a node u , for all v calculate the gradient $\frac{\partial \mathcal{L}^{(u)}}{\partial z_v}$
- For all v , update $z_v \leftarrow z_v - \eta \frac{\partial \mathcal{L}^{(u)}}{\partial z_v}$

Random Walks: Summary

- Run short fixed-length random walks starting from each node on the graph
- For each node u , collect $N_R(u)$, the multiset of nodes visited on random walks starting from u
- Optimize embeddings \mathbf{Z} using Stochastic Gradient Descent

$$\mathcal{L} = \sum_{u \in V} \sum_{v \in N_R(u)} -\log(P(v|\mathbf{z}_u))$$

We can efficiently approximate this
using negative sampling

How Should We Randomly Walk?

- So far we have described how to optimize embeddings given a random walk strategy R
- **What strategies should we use to run these random walks?**
 - Simplest idea: Just run fixed-length, unbiased random walks starting from each node (i.e., DeepWalk from Perozzi et al.)
 - The issue is that such notion of similarity is too constrained
- How can we generalize this?

Perozzi et al. DeepWalk: Online Learning of Social Representations. In KDD, 2014.

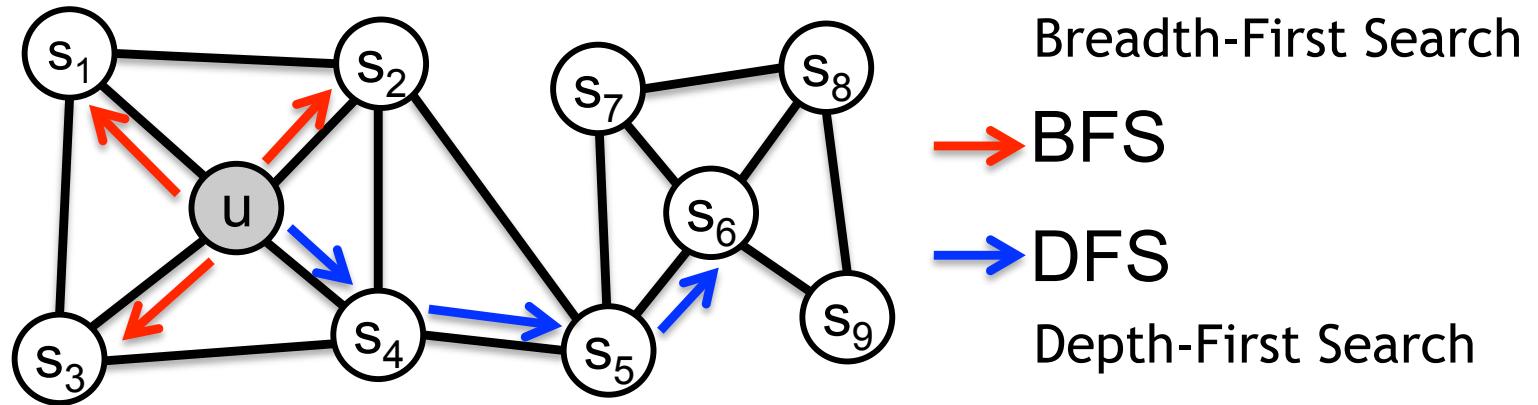
Overview of node2vec

- **Goal:** Embed nodes with similar network neighborhoods close in the feature space
- This goal is framed as a maximum likelihood optimization problem, independent to the downstream prediction task
- **Key observation:** Flexible notion of network neighborhood $N_R(u)$ of node u leads to rich node embeddings
- Develop biased 2nd order random walk R to generate network neighborhood $N_R(u)$ of node u

Grover et al. node2vec: Scalable Feature Learning for Networks. In KDD, 2016.

node2vec: Biased Walks

- Idea: use flexible, biased random walks that can trade off between **local** and **global** views of the network (Grover and Leskovec, 2016).



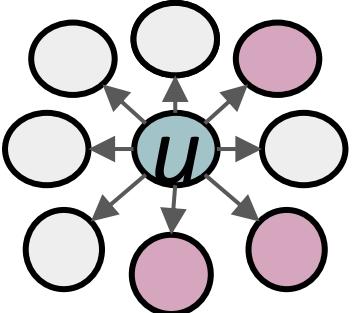
Walk of length 3 ($N_R(u)$ of size 3):

$$N_{BFS}(u) = \{ s_1, s_2, s_3 \} \quad \text{Local microscopic view}$$

$$N_{DFS}(u) = \{ s_4, s_5, s_6 \} \quad \text{Global macroscopic view}$$

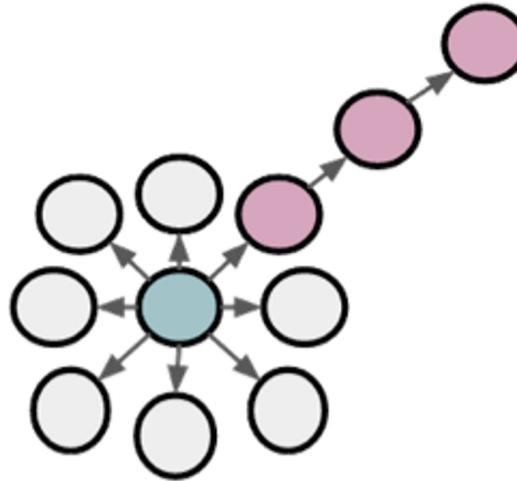
Grover et al. node2vec: Scalable Feature Learning for Networks. In KDD, 2016.

BFS vs DFS



BFS:

$N_R(\cdot)$ will provide
a micro-view of
neighbourhood



DFS:

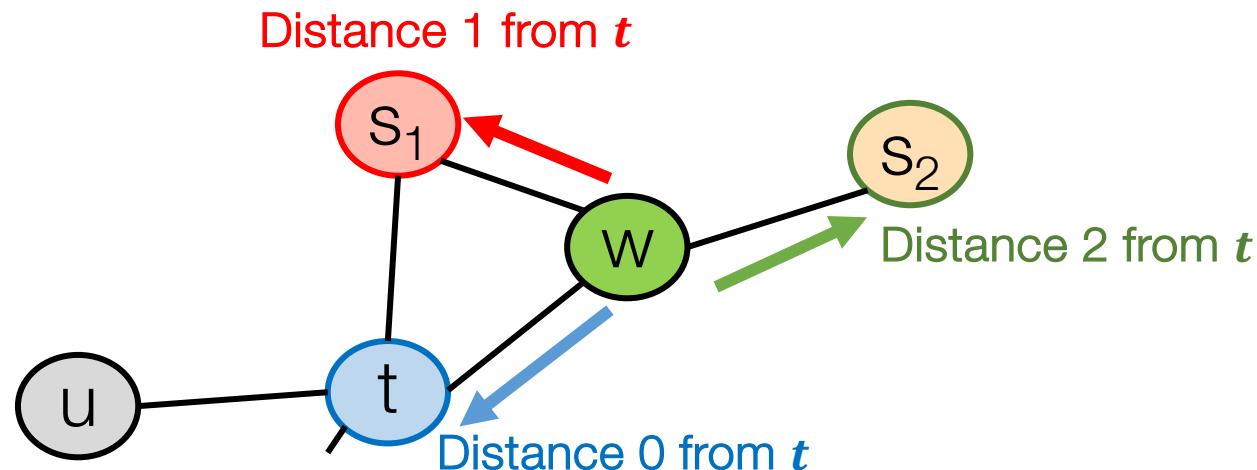
$N_R(\cdot)$ will provide a
macro-view of
neighbourhood

Interpolating BFS and DFS

- Biased fixed-length random walk R that given a node u generates neighborhood $N_R(u)$
 - Random walk has two parameters
 - Return parameter p
 - Return back to the previous node
 - In-out parameter q
 - Moving outwards (DFS) vs inwards (BFS) from the previous node
 - Intuitively, q is the “ratio” of BFS vs DFS
 - Next, we specify how a **single step** of biased random walk is performed
 - Random walk is then just a sequence of these steps

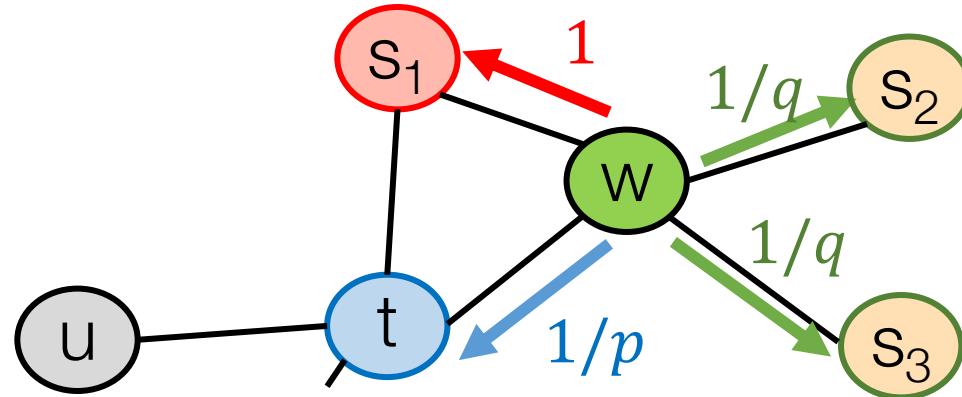
One Step of the Biased Random Walk

- Define the random walk by specifying the walk transition probabilities on edges adjacent to the current node w :
 - Random walk **just traversed edge** (t, w) and **is now at** w
 - We specify edge transition probabilities out of node w
 - **Insight:** Neighbors of w can only be:



One Step of the Biased Random Walk

- Walker came over edge (t, w) and is now at w .
 - How to set edge transition probabilities?

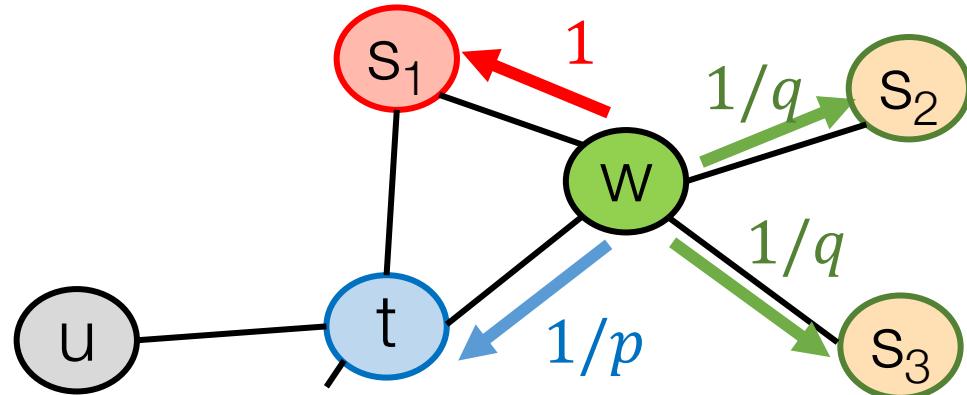


$1/p, 1/q, 1$ are
unnormalized
probabilities

- p, q model transition probabilities
 - p : return parameter
 - q : “walk away” parameter

One Step of the Biased Random Walk

- Walker came over edge (t, w) and is now at w .
 - How to set edge transition probabilities?



Target x	Prob.	Dist. (t, x)
t	$1/p$	0
s_1	1	1
s_2	$1/q$	2
s_3	$1/q$	2

Unnormalized
transition prob.
segmented based
on distance from t

- BFS-like walk: low value of p
- DFS-like walk: low value of q
- $N_R(u)$ are the nodes visited by the biased walk

node2vec algorithm

- Compute edge transition probabilities:
 - For each edge (t, w) , compute edge walk probabilities (based on p, q) of edge (w, \cdot)
- Simulate r random walks of length l starting from each node
- Optimize the node2vec objective using Stochastic Gradient Descent
- **Linear-time complexity**
- All 3 steps are **individually parallelizable**

Summary So Far

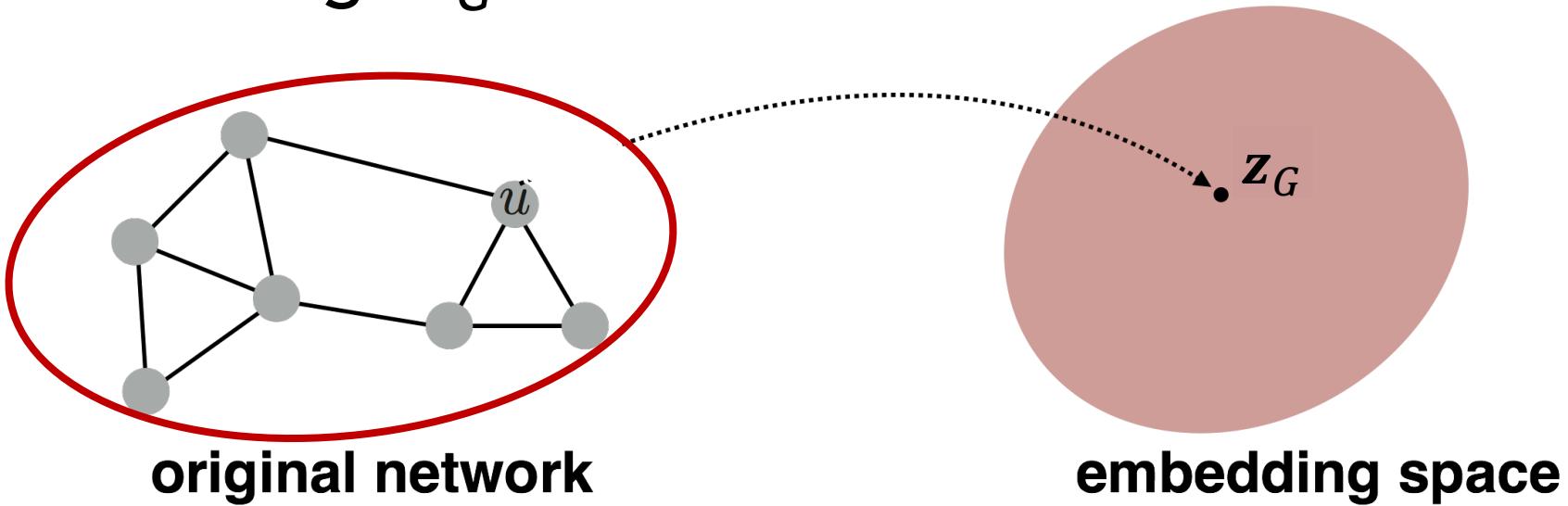
- Core idea: Embed nodes so that distances in embedding space reflect node similarities in the original network
- Different notions of node similarity:
 - Naïve: Similar if two nodes are connected
 - Random walk approaches (covered today)

Summary So Far

- So, what method should we use?
- No one method wins in all cases
 - E.g., node2vec performs better on node classification while alternative methods perform better on link prediction (Goyal and Ferrara, survey)
- Random walk approaches are generally more efficient
- **In general:** Must choose definition of node similarity that matches your application

Embedding Entire Graphs

- Goal: Want to embed a subgraph or an entire graph G .
Graph embedding: \mathbf{z}_G



- Tasks:
 - Classifying toxic vs non-toxic molecules
 - Identifying anomalous graphs

Approach 1

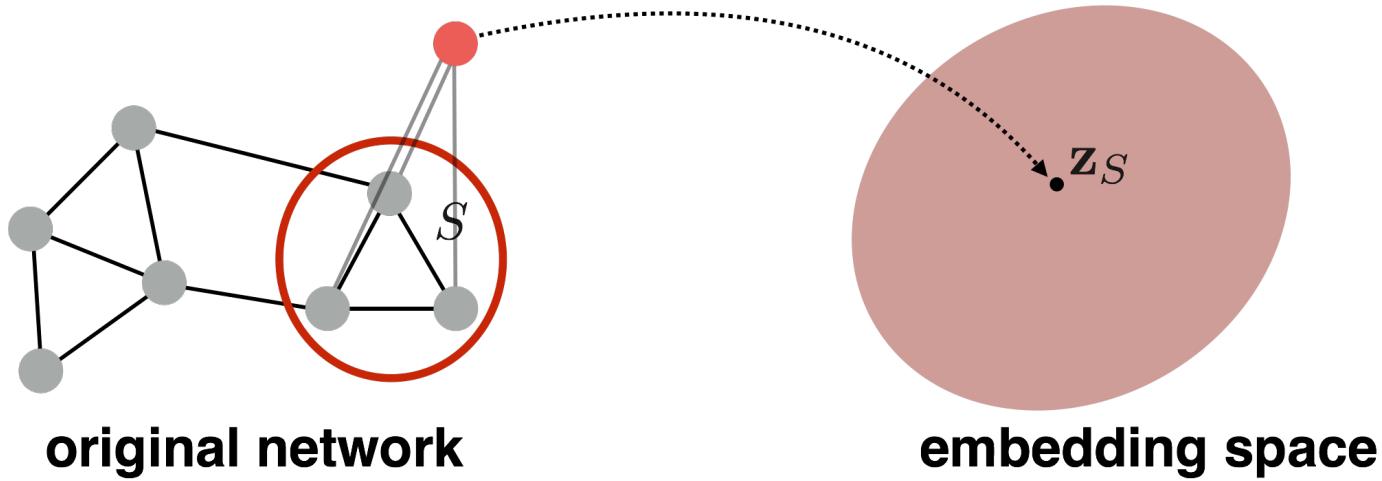
- Simple (but effective) approach 1:
 - Run a standard node embedding technique on the (sub)graph G
 - Then just sum (or average) the node embeddings in the (sub)graph G

$$\mathbf{z}_G = \sum_{v \in G} \mathbf{z}_v$$

- Used by Duvenaud et al. to classify molecules based on their graph structure

Approach 2

- Approach 2: Introduce a “**virtual node**” to represent the (sub)graph and run a standard graph embedding technique



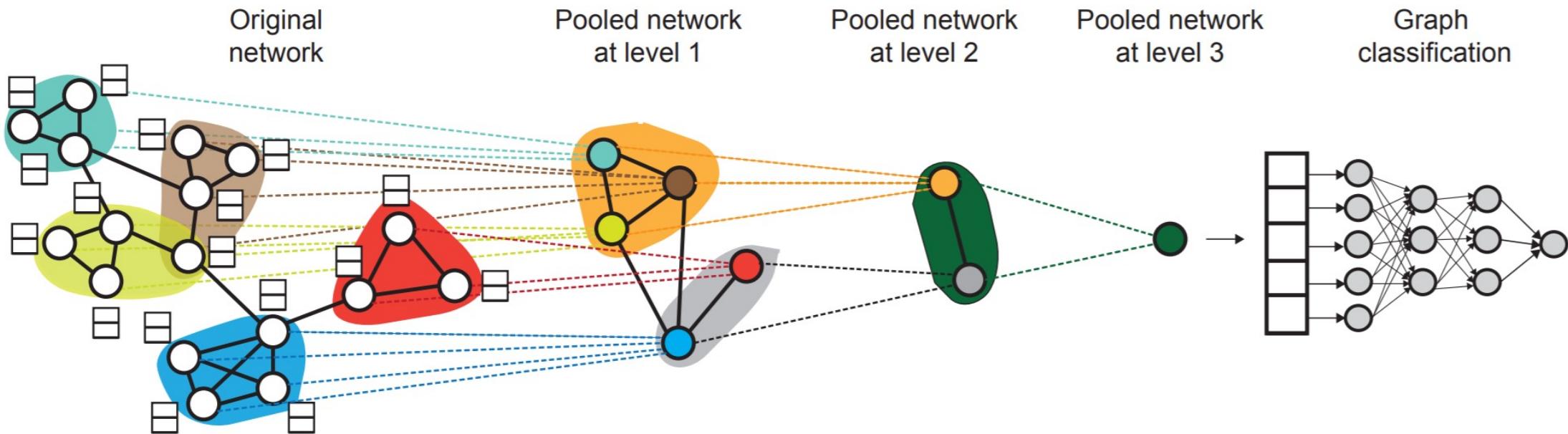
- Proposed by Li et al. as a general technique for subgraph embedding

Summary

- We discussed 2 ideas to graph embeddings:
 - **Approach 1:** Embed nodes and sum/avg them
 - **Approach 2:** Create super-node that spans the (sub) graph and then embed that node

Preview: Hierarchical Embeddings

- **DiffPool:** We can also **hierarchically** cluster nodes in graphs, and **sum/avg** the node embeddings according to these clusters



How to Use Embeddings?

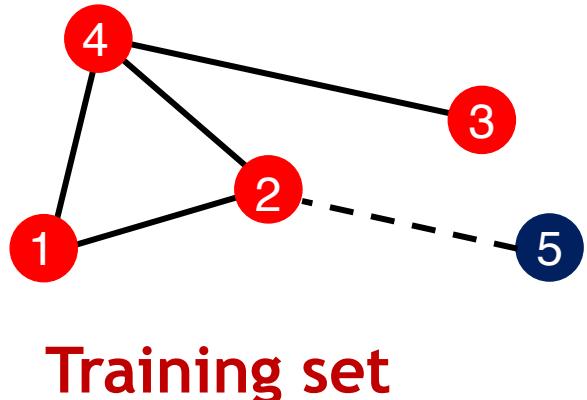
- How to use embeddings \mathbf{z}_i of nodes
 - **Clustering/community detection:** cluster points \mathbf{z}_i
 - **Node classification:** predict label of node i based on \mathbf{z}_i
 - **Link prediction:** predict edge (i, j) based on $(\mathbf{z}_i, \mathbf{z}_j)$
 - where we can: concatenate, avg, product, or take difference between embeddings:
 - Concatenate: $f(\mathbf{z}_i, \mathbf{z}_j) = g([\mathbf{z}_i, \mathbf{z}_j])$
 - Hadamard: $f(\mathbf{z}_i, \mathbf{z}_j) = g(\mathbf{z}_i * \mathbf{z}_j)$ (per coordinate product)
 - Sum/avg: $f(\mathbf{z}_i, \mathbf{z}_j) = g(\mathbf{z}_i + \mathbf{z}_j)$
 - Distance: $f(\mathbf{z}_i, \mathbf{z}_j) = g(\|\mathbf{z}_i - \mathbf{z}_j\|_2)$
 - **Graph classification:** graph embedding \mathbf{z}_G via aggregating node embeddings or virtual node
 - Predict label based on graph embedding \mathbf{z}_G

Summary

- We discussed **graph representation learning**, a way to learn **node** and **graph embeddings** for downstream tasks, **without feature engineering**
 - **Encoder-decoder framework:**
 - Encoder: embedding lookup
 - Decoder: predict score based on embedding to match node similarity
 - **Node similarity measure:** (biased) random walk
 - Examples: DeepWalk, Node2Vec
 - **Extension to Graph embedding:** Node embedding aggregation

Limitations (1)

- Limitations of node embeddings via random walks
 - Transductive (not inductive) method:
 - Cannot obtain embeddings for nodes not in the training set
 - Cannot apply to new graphs
 - If you apply DeepWalk to the same graph multiple times, each time you'll get a different embedding each time. Why?

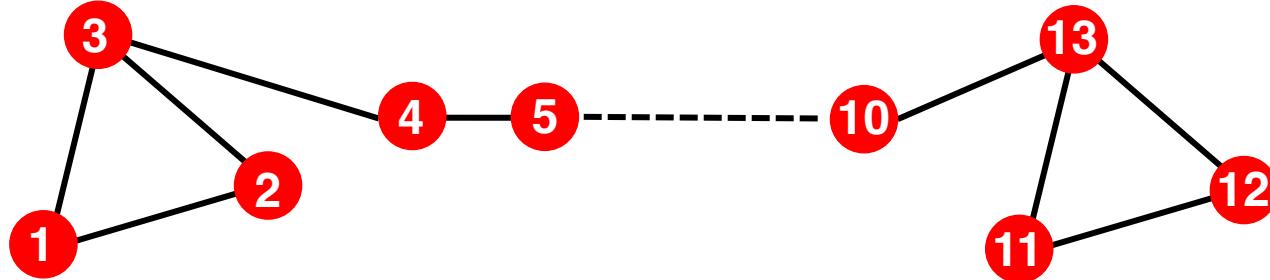


A newly added node 5 at test time
(e.g., new user in a social network)

Cannot compute its embedding
with DeepWalk / node2vec. Need to
recompute all node embeddings

Limitations (2)

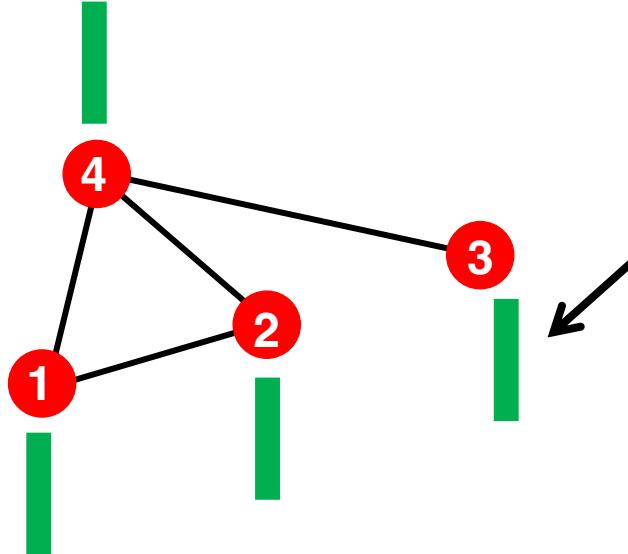
- Cannot capture **structural similarity**:



- Node 1 and 11 are **structurally similar**: part of one triangle, degree 2, ...
- However, they have very **different** embeddings
 - It's unlikely that a random walk will reach node 11 from node 1
- DeepWalk and node2vec do not capture structural similarity

Limitations (3)

- Cannot utilize node, edge and graph features



Feature vector
(e.g. protein properties in a protein-protein interaction graph)

DeepWalk / node2vec
embeddings do not incorporate
such node features

Solution to these limitations: Deep Representation Learning and Graph Neural Networks
(To be covered in depth next)



COSC 3337
Data Science I
Section 14623

Graph Data Analysis

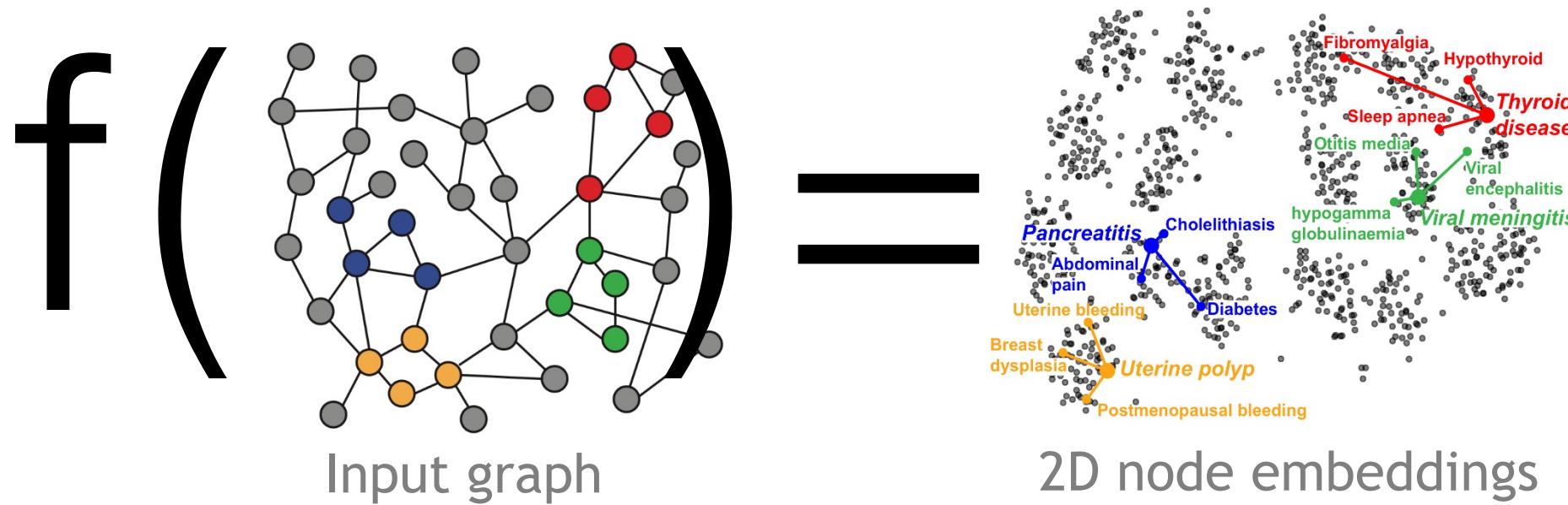
Instructor: Jingchao Ni
Fall 2024

About Exam II

- Time and location: Nov. 25 (Monday) 2:30PM, SEC 104
- Length: 90 Minutes
- Exam Materials:
 - You can use the lectures notes/slides
 - You can also reference reading material distributed as part of your reading assignment or suggested textbooks for the course
 - **Use calculator or python for some computations**
 - NO INTERNET AND GENERAL AI RESOURCES SHOULD BE USED
- Exam Contents: 100 points
 - Clustering, Anomaly Detection, Deep Learning, Graph Data Analysis

Recap: Node Embeddings

- Intuition: Map nodes to d -dimensional embeddings such that similar nodes in the graph are embedded close together

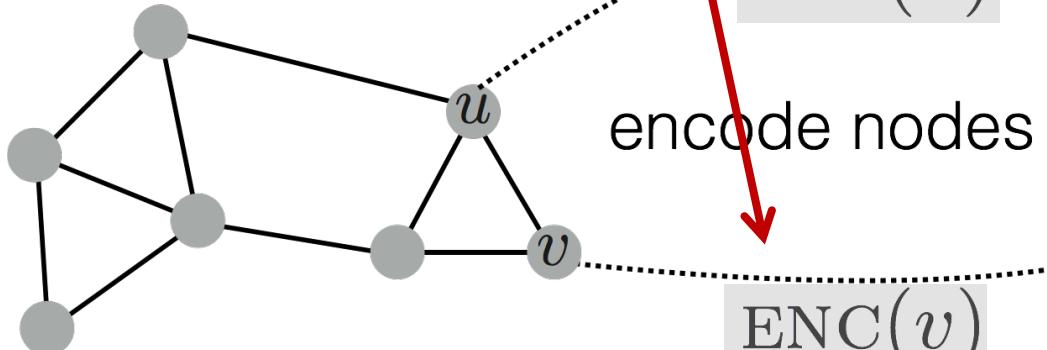


How to learn mapping function f ?

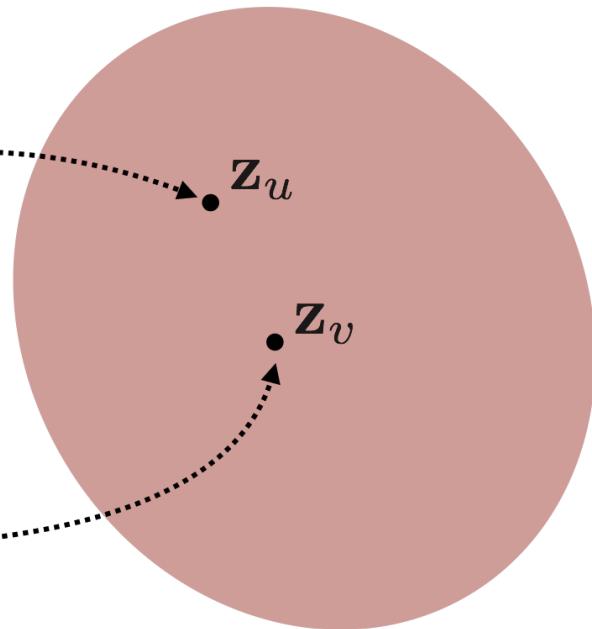
Recap: Node Embeddings

Goal: $\text{similarity}(u, v) \approx \mathbf{z}_v^T \mathbf{z}_u$
in the original network Similarity of the embedding

Need to define!



Input network



d -dimensional
embedding space

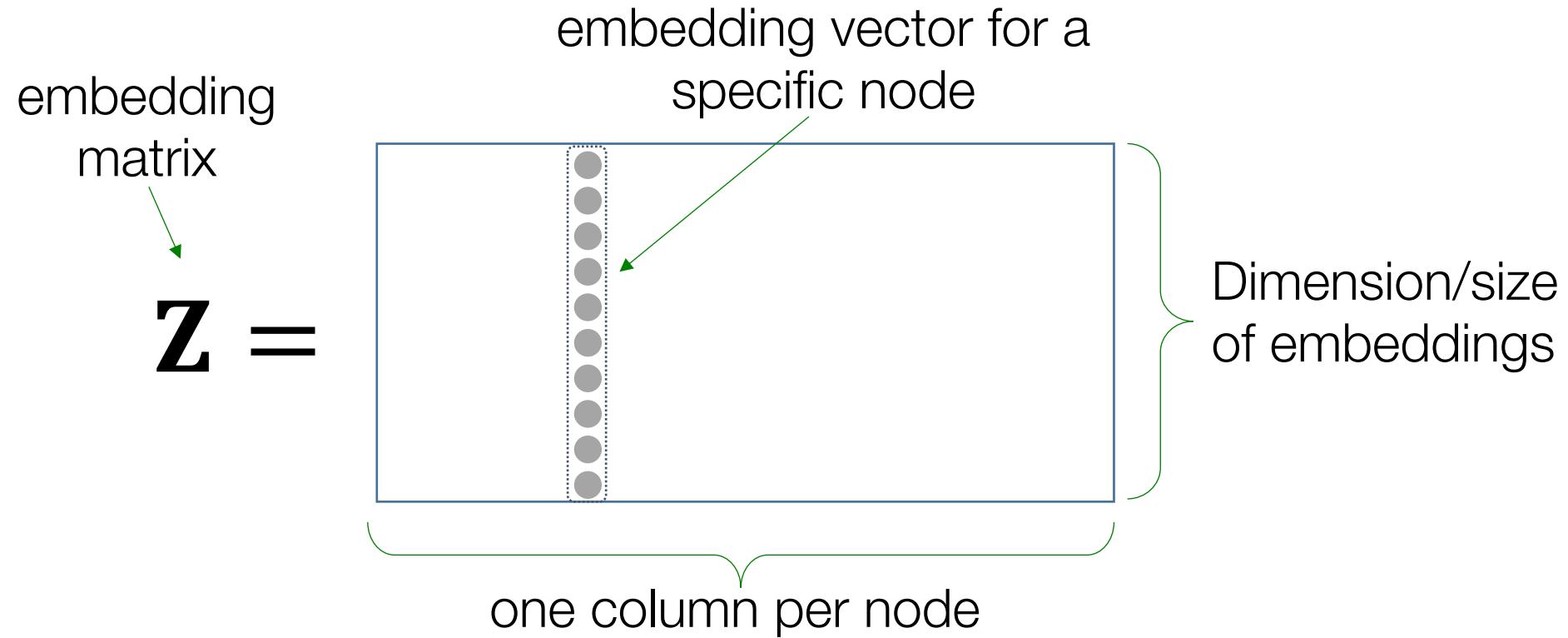
encode nodes

$\text{ENC}(u)$

$\text{ENC}(v)$

Recap: “Shallow” Encoding

- Simplest encoding approach: **Encoder is just an embedding-lookup**



Recap: “Shallow” Encoders

- Limitations of shallow embedding methods:
 - **$O(|V|d)$ parameters are needed:**
 - No sharing of parameters between nodes
 - Every node has its own unique embedding
 - **Inherently “transductive”:**
 - Cannot generate embeddings for nodes that are not seen during training
 - **Do not incorporate node features:**
 - Nodes in many graphs have features that we can and should leverage

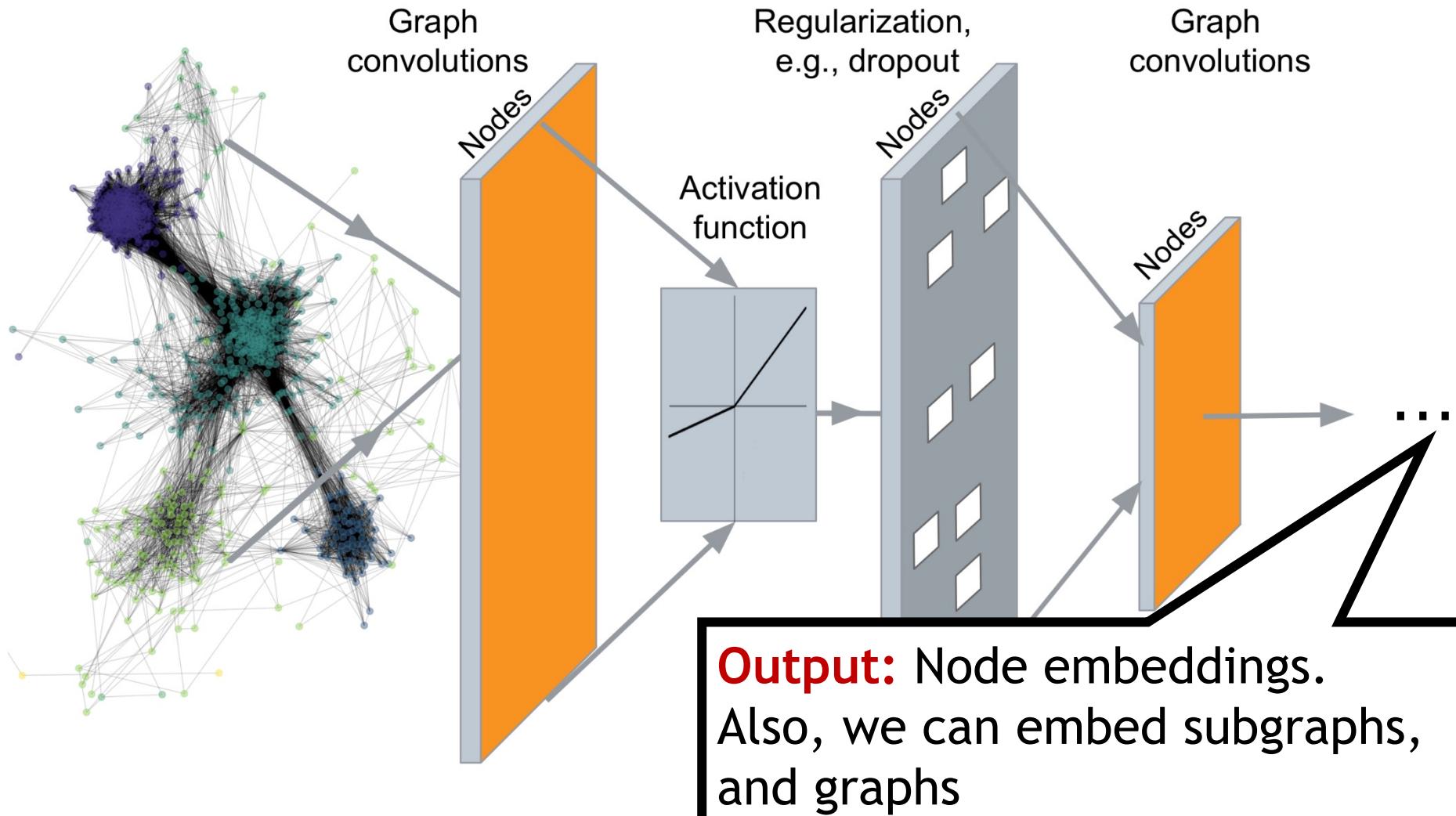
Today: Deep Graph Encoders

- We will discuss deep learning methods based on **graph neural networks (GNNs)**:

$\text{ENC}(v) =$ **multiple layers of
non-linear transformations
based on graph structure**

- Note: All these deep encoders can be **combined with node similarity functions** defined in the previous lecture

Deep Graph Encoders

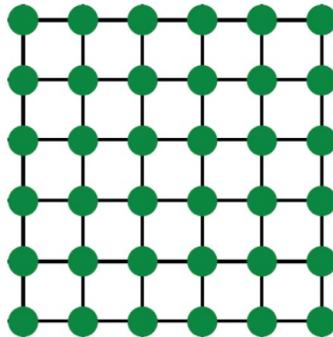


Tasks on Graphs

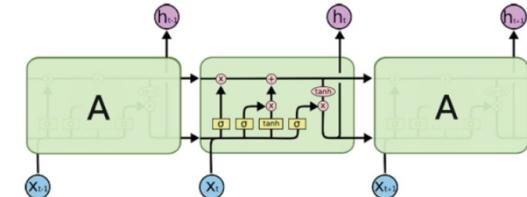
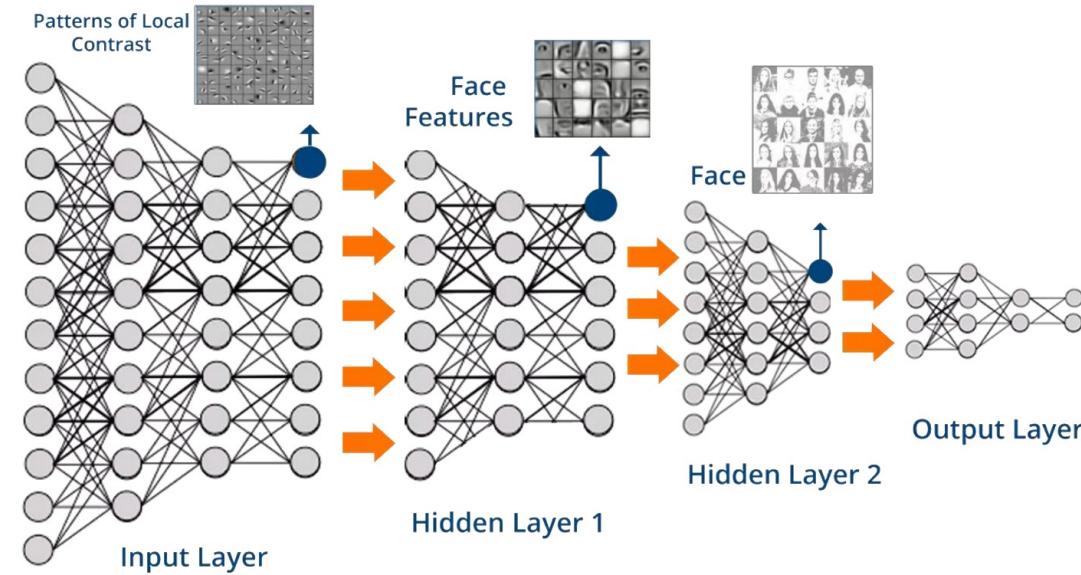
- Tasks we will be able to solve:
 - **Node classification**
 - Predict the type/class of a given node
 - **Link prediction**
 - Predict whether two nodes are linked
 - **Clustering/Community detection**
 - Identify densely linked clusters of nodes
 - **Graph similarity**
 - How similar are two (sub)graphs

Machine Learning Toolbox

- Modern deep learning toolbox is designed for simple sequences and grids

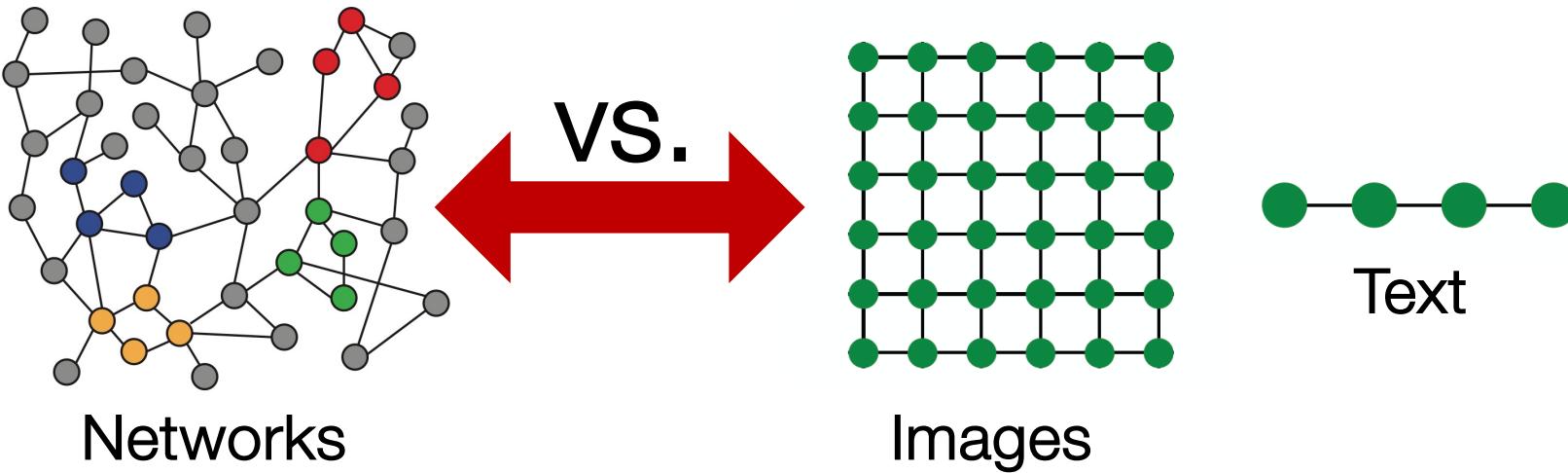


Images



Why is it Hard?

- But graphs are far more complex!
 - Arbitrary size and complex topological structure (i.e., no spatial locality like grids)



- No fixed node ordering or reference point
- Often dynamic and have multimodal features

Recap: Basics of Deep Learning

- Loss function \mathcal{L} :

$$\min_{\Theta} \mathcal{L}(y, f_{\Theta}(\mathbf{x}))$$

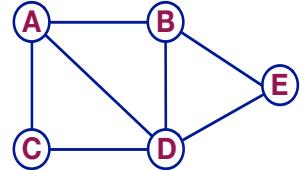
- f can be a simple linear layer, an MLP, or other neural networks (e.g., a GNN later)
- Sample a **minibatch** of input data \mathbf{x}
- **Forward propagation:** Compute \mathcal{L} given \mathbf{x}
- **Back-propagation:** Obtain gradient $\nabla_{\Theta} \mathcal{L}$ using a chain rule
- Use **stochastic gradient descent (SGD)** to optimize \mathcal{L} for weights Θ over many iterations

Terminology

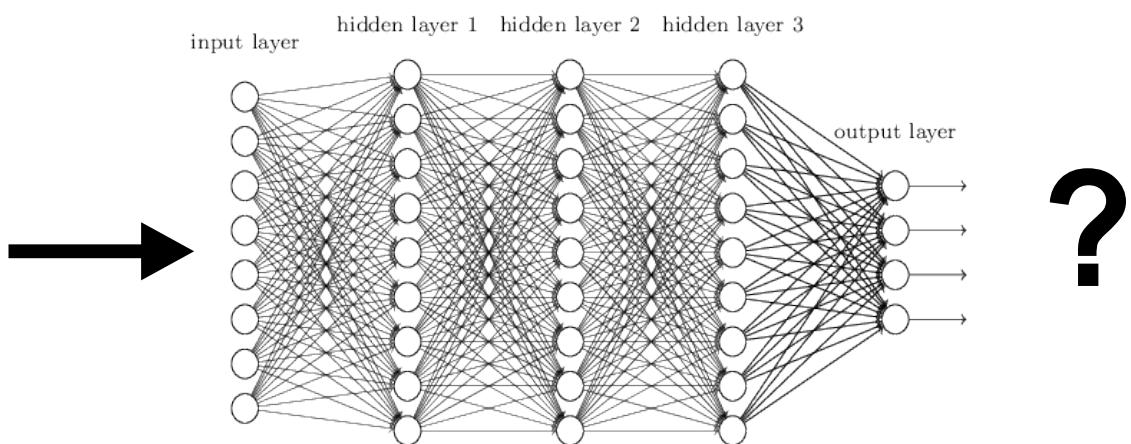
- Assume we have a graph G :
 - V is the **vertex/node set**
 - A is the **adjacency matrix** (assume binary)
 - $X \in \mathbb{R}^{|V| \times m}$ is a matrix of **node features**
 - v : a node in V ; $N(v)$: the set of neighbors of v .
 - **Node features**:
 - Social networks: User profile, User image
 - Biological networks: Gene expression profiles, gene functional information
 - When there is no node feature in the graph dataset:
 - Indicator vectors (one-hot encoding of a node)
 - Vector of constant 1: $[1, 1, \dots, 1]$

A Naïve Approach

- Join adjacency matrix and features
- Feed them into a deep neural net:



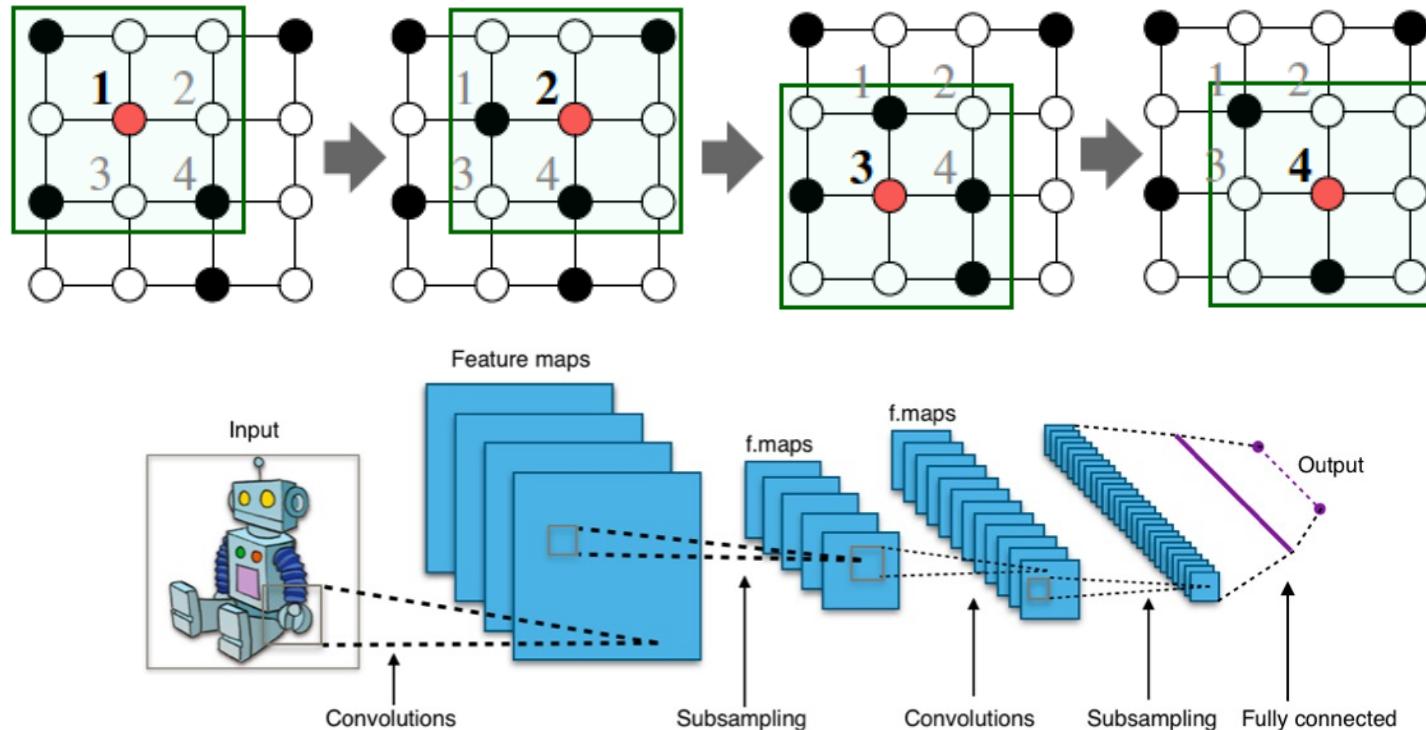
	A	B	C	D	E	Feat
A	0	1	1	1	0	1 0
B	1	0	0	1	1	0 0
C	1	0	0	1	0	0 1
D	1	1	1	0	1	1 1
E	0	1	0	1	0	1 0



- Issues with this idea:
 - $\mathcal{O}(|V|)$ parameters
 - Not applicable to graphs of different sizes
 - Sensitive to node ordering

Idea: Convolutional Neural Networks

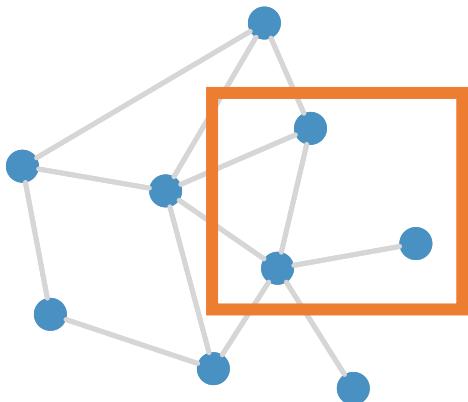
- CNN on an image



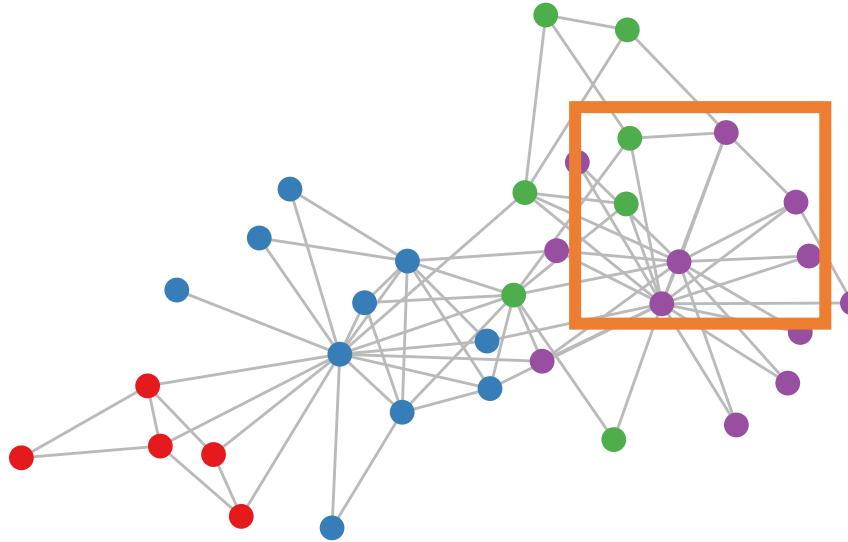
Goal is to generalize convolutions beyond simple lattices
Leverage node features/attributes (e.g., text, images)

Real-World Graphs

- But graphs look like this



or this:



- There is no fixed notion of locality or sliding window on the graph
- Graph is **permutation invariant**

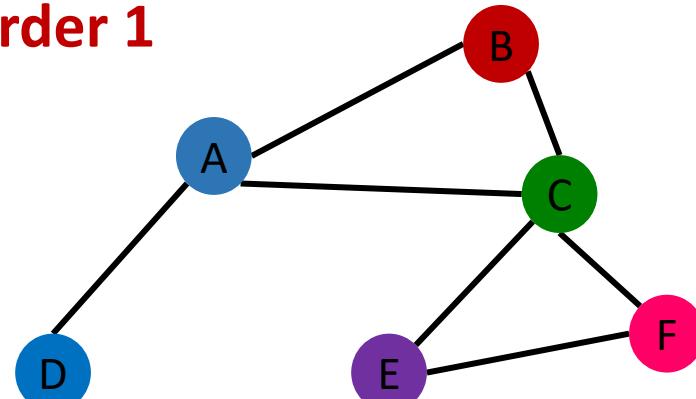
Permutation Invariance

- Consider we want to embed an entire graph
- Observation: A graph does not have a canonical ordering of its nodes
 - We can have many different node orderings of the same graph
- What do we want: If we learn an embedding function over a graph, we should get the same result (the same embedding) regardless of how the nodes are numbered

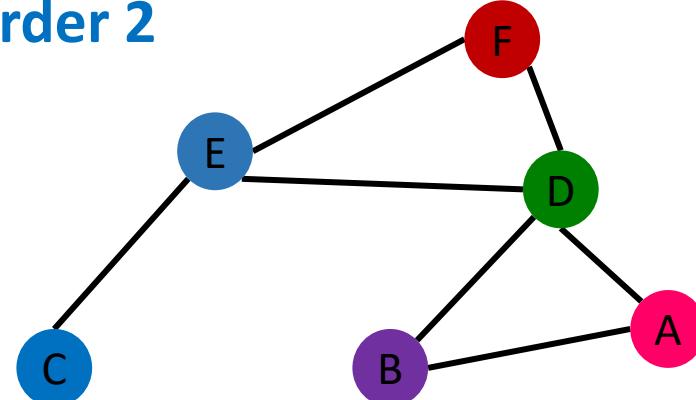
Permutation Invariance

- Graph does not have a canonical ordering of the nodes

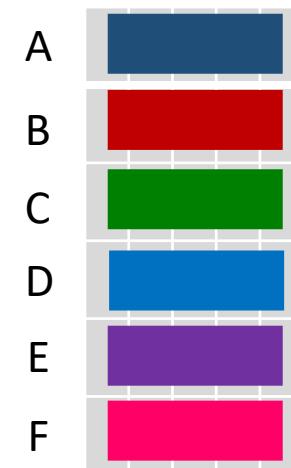
Order 1



Order 2



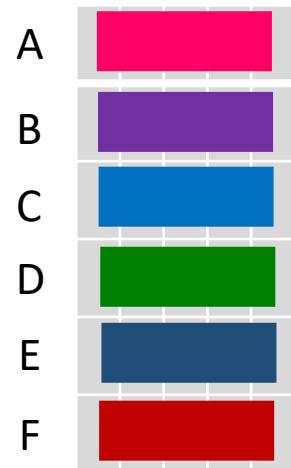
Node features X_1



Adjacency matrix A_1

	A	B	C	D	E	F
A	Gray	Light Blue	Light Blue	Light Blue	Light Blue	Gray
B	Light Blue	Gray	Light Blue	Light Blue	Light Blue	Gray
C	Light Blue	Light Blue	Gray	Light Blue	Light Blue	Light Blue
D	Light Blue	Light Blue	Light Blue	Gray	Light Blue	Light Blue
E	Light Blue	Light Blue	Light Blue	Light Blue	Gray	Light Blue
F	Light Blue	Gray				

Node features X_2

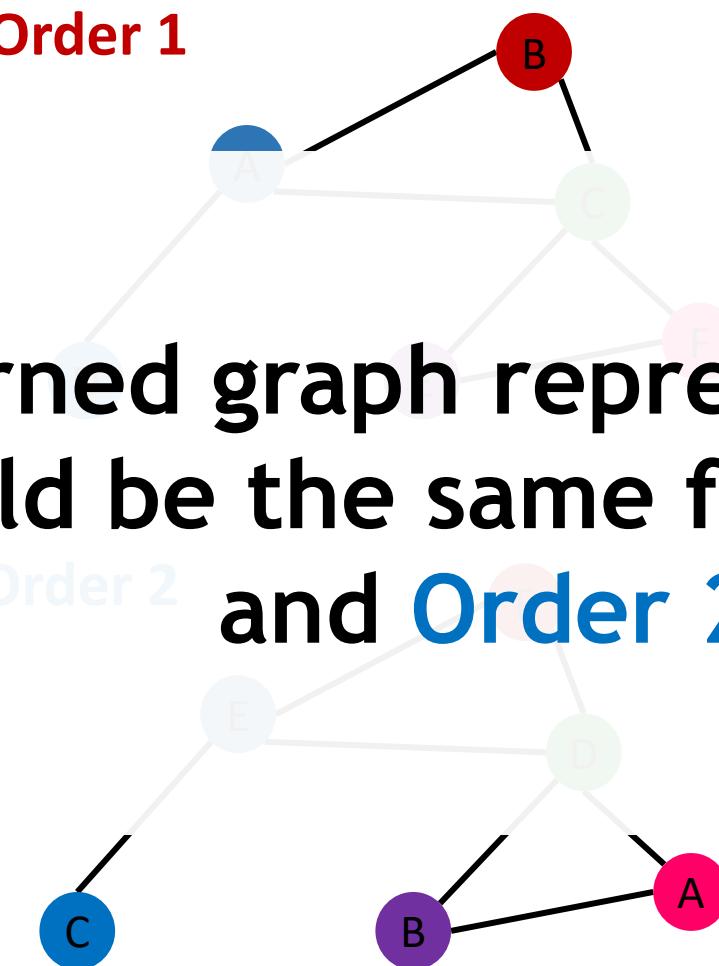


Adjacency matrix A_2

	A	B	C	D	E	F
A	Gray	Light Blue	Gray	Light Blue	Light Blue	Gray
B	Light Blue	Gray	Gray	Light Blue	Light Blue	Gray
C	Gray	Gray	Gray	Light Blue	Light Blue	Light Blue
D	Light Blue	Light Blue	Light Blue	Gray	Light Blue	Light Blue
E	Gray	Light Blue	Light Blue	Light Blue	Gray	Light Blue
F	Gray	Light Blue	Light Blue	Light Blue	Light Blue	Gray

Permutation Invariance

- Graph does not have a canonical ordering of the nodes

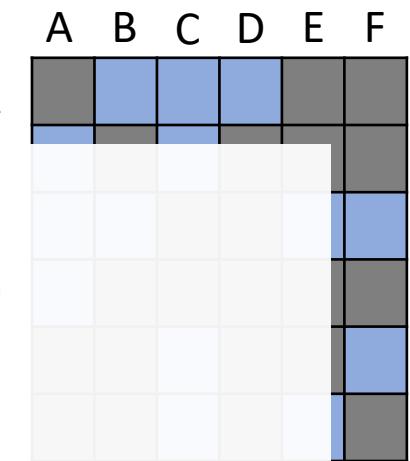


Learned graph representation should be the same for **Order 1** and **Order 2**

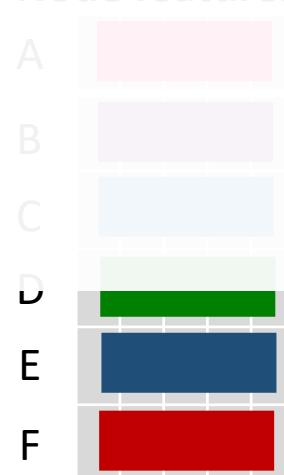
Node features X_1



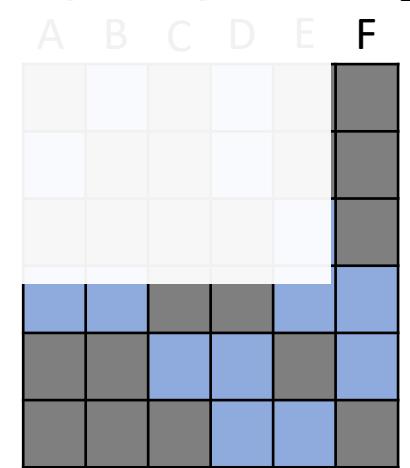
Adjacency matrix A_1



Node features X_2



Adjacency matrix A_2



Permutation Invariance

- What do we mean by “graph representation is the same for two orderings”?
 - Consider we learn a function f that maps a graph $G = (A, X)$ to a vector in \mathbb{R}^d then

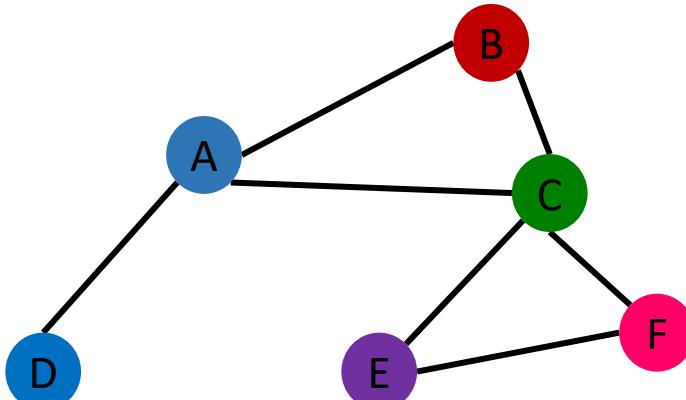
In other words, f maps a graph to a d -dim embedding

$$f(A_1, X_1) = f(A_2, X_2)$$

A is the adjacency matrix

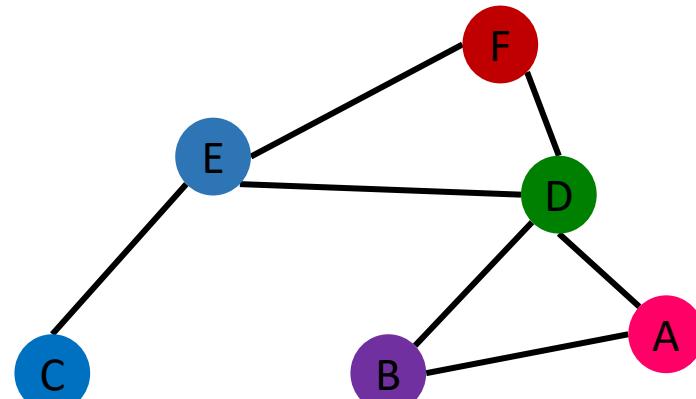
X is the node feature matrix

Order 1: A_1, X_1



For two orders,
output of f should
be the same!

Order 2: A_2, X_2



Permutation Invariance

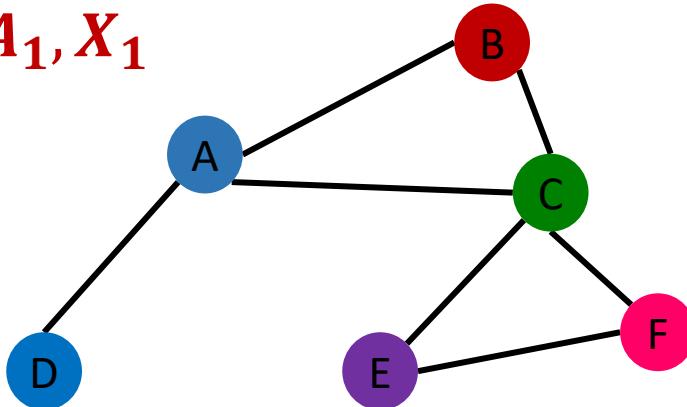
- What do we mean by “graph representation is the same for two orderings”?
 - Consider we learn a function f that maps a graph $G = (A, X)$ to a vector in \mathbb{R}^d
 A is the adjacency matrix, X is the node feature matrix
 - Then, if $f(A_i, X_i) = f(A_j, X_j)$ for any ordering i and j , we formally say f is a **permutation invariant function**
For a graph with $|V|$ nodes, there are $|V|!$ different orderings
 - **Definition:** For any **graph** function: $f: \mathbb{R}^{|V| \times |V|} \times \mathbb{R}^{|V| \times m} \rightarrow \mathbb{R}^d$, f is **permutation invariant** if $f(A, X) = f(PAP^T, PX)$ for any permutation matrix P
Permutation matrix P : a shuffle of the node order
Example: $(A, B, C) \rightarrow (B, C, A)$
 m ... each node has a m -dim feature vector associated with it

Permutation Equivariance

- For node representation: we learn a function f that maps nodes of G to a matrix in $\mathbb{R}^{|V| \times d}$

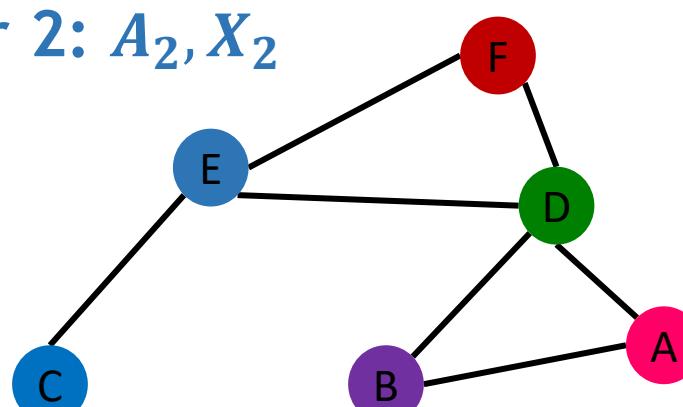
In other words, each node in V is mapped to a d -dim embedding

Order 1: A_1, X_1



$$f(A_1, X_1) = \begin{matrix} & \text{A} & \text{B} & \text{C} & \text{D} & \text{E} & \text{F} \\ \text{A} & \text{G} & \text{G} & & & & \\ \text{B} & \text{G} & \text{H} & \text{H} & & & \\ \text{C} & \text{H} & \text{H} & \text{I} & \text{I} & & \\ \text{D} & \text{I} & \text{I} & \text{I} & \text{J} & \text{J} & \\ \text{E} & \text{J} & \text{J} & \text{J} & \text{J} & \text{K} & \text{K} \\ \text{F} & \text{K} & \text{K} & \text{K} & \text{K} & \text{K} & \text{L} \end{matrix}$$

Order 2: A_2, X_2

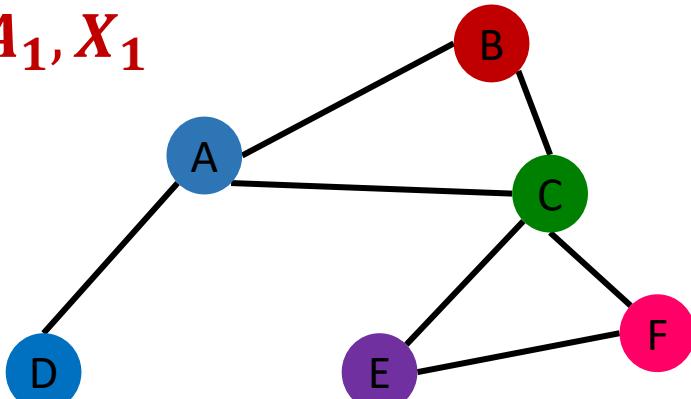


$$f(A_2, X_2) = \begin{matrix} & \text{A} & \text{B} & \text{C} & \text{D} & \text{E} & \text{F} \\ \text{A} & \text{L} & \text{L} & & & & \\ \text{B} & \text{L} & \text{M} & \text{M} & & & \\ \text{C} & \text{M} & \text{M} & \text{N} & \text{N} & & \\ \text{D} & \text{N} & \text{N} & \text{N} & \text{O} & \text{O} & \\ \text{E} & \text{O} & \text{O} & \text{O} & \text{O} & \text{P} & \text{P} \\ \text{F} & \text{P} & \text{P} & \text{P} & \text{P} & \text{P} & \text{L} \end{matrix}$$

Permutation Equivariance

- For node representation: we learn a function f that maps nodes of G to a matrix in $\mathbb{R}^{|V| \times d}$

Order 1: A_1, X_1



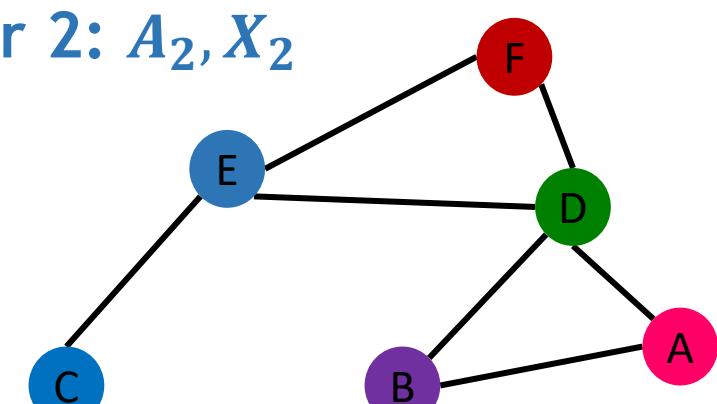
Representation vector
of the blue node A

A		
B		
C		
D		
E		
F		

$$f(A_1, X_1) =$$

For two orderings, the vector
of node at the same position in
the graph is the same!

Order 2: A_2, X_2



A		
B		
C		
D		
E		
F		

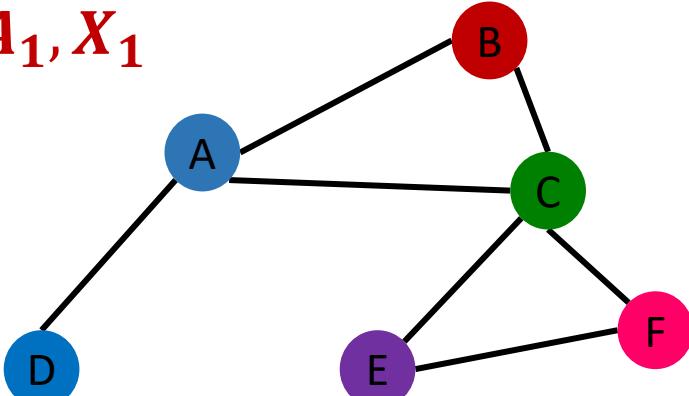
Representation vector
of the blue node E

$$f(A_2, X_2) =$$

Permutation Equivariance

- For node representation: we learn a function f that maps nodes of G to a matrix in $\mathbb{R}^{|V| \times d}$

Order 1: A_1, X_1

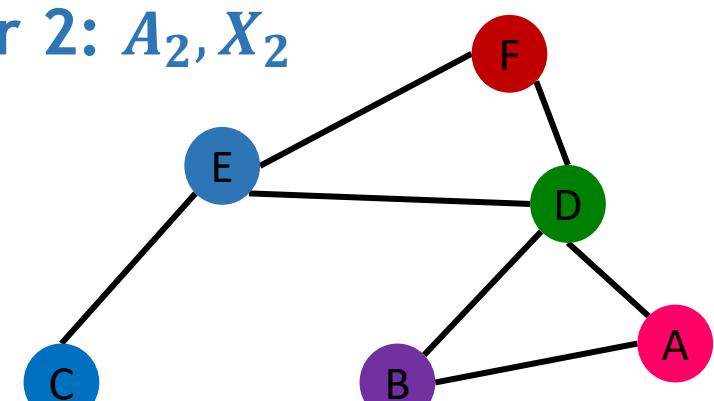


Representation vector
of the green node C

$$f(A_1, X_1) = \begin{matrix} & \text{A} & \text{B} & \text{C} & \text{D} & \text{E} & \text{F} \\ \text{A} & \text{Gold} & & & & & \\ \text{B} & & \text{Red} & & & & \\ \text{C} & & & \text{Green} & & & \\ \text{D} & & & & \text{Blue} & & \\ \text{E} & & & & & \text{Purple} & \\ \text{F} & & & & & & \text{Red} \end{matrix}$$

For two orderings, the vector
of node at the same position in
the graph is the same!

Order 2: A_2, X_2



$$f(A_2, X_2) = \begin{matrix} & \text{A} & \text{B} & \text{C} & \text{D} & \text{E} & \text{F} \\ \text{A} & \text{Red} & & & & & \\ \text{B} & & \text{Purple} & & & & \\ \text{C} & & & \text{Blue} & & & \\ \text{D} & & & & \text{Green} & & \\ \text{E} & & & & & \text{Gold} & \\ \text{F} & & & & & & \text{Red} \end{matrix}$$

Representation vector
of the green node D

Permutation Equivariance

- For node representation:

- Consider we learn a function f that maps a graph $G = (A, X)$ to a matrix in $\mathbb{R}^{|V| \times d}$
- If the output vector of a node at the same position in the graph remains unchanged for any ordering, we say f is **permutation equivariant**

m: each node has a m -dim feature vector associated with it

- Definition: For any **node** function $f: \mathbb{R}^{|V| \times |V|} \times \mathbb{R}^{|V| \times m} \rightarrow \mathbb{R}^{|V| \times d}$, f is **permutation equivariant** if $Pf(A, X) = f(PAP^T, PX)$ for any permutation P

f maps each node in V to a d -dim embedding

Summary: Invariance and Equivariance

- Permutation invariant

$$f(A, X) = f(PAP^T, PX)$$

- Permutation equivariant

$$Pf(A, X) = f(PAP^T, PX)$$

- Examples

- $f(A, X) = \mathbf{1}^T X$: permutation invariant

- Reason: $f(PAP^T, PX) = \mathbf{1}^T P X = \mathbf{1}^T X = f(A, X)$

- $f(A, X) = X$: permutation equivariant

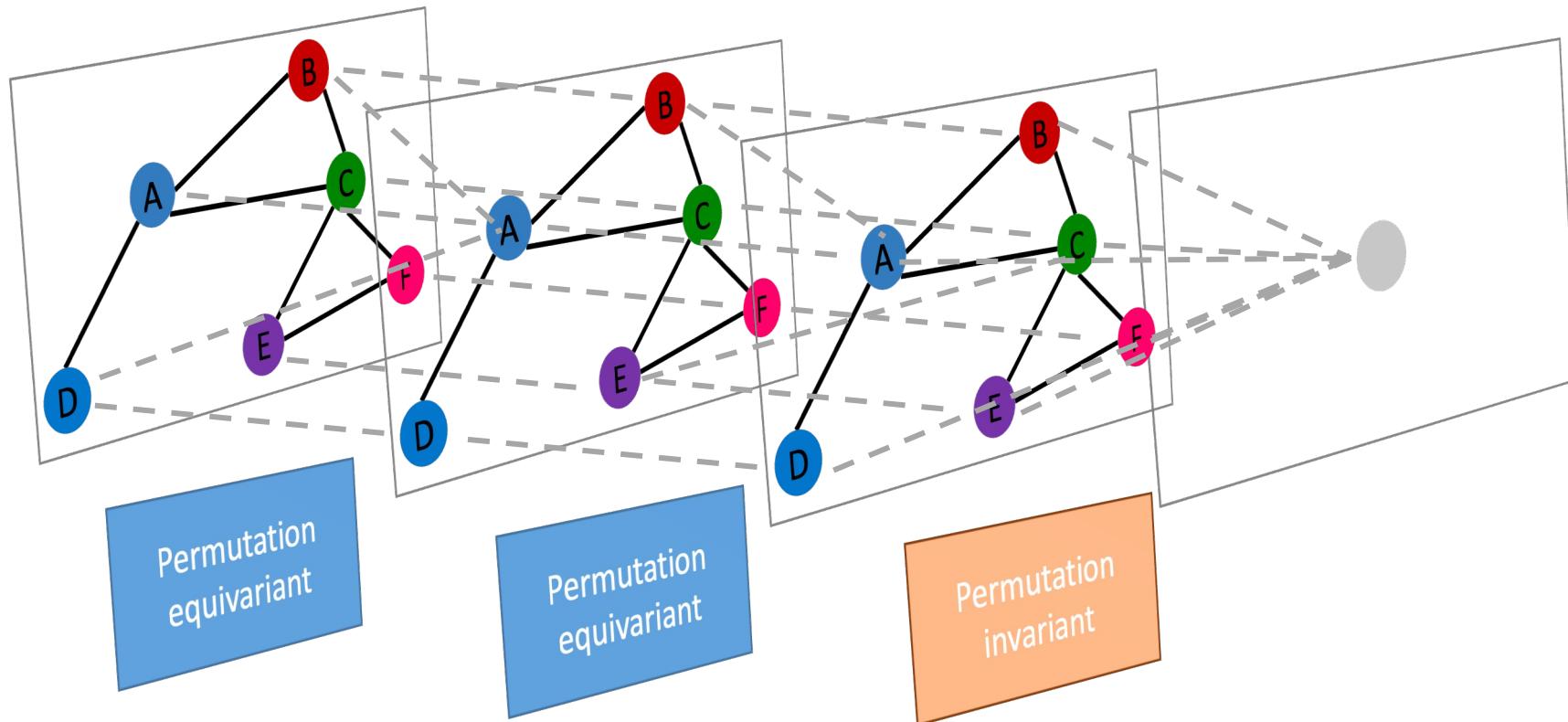
- Reason: $f(PAP^T, PX) = P X = Pf(A, X)$

- $f(A, X) = A X$: permutation equivariant

- Reason: $f(PAP^T, PX) = P A P^T P X = P A X = Pf(A, X)$

Graph Neural Network Overview

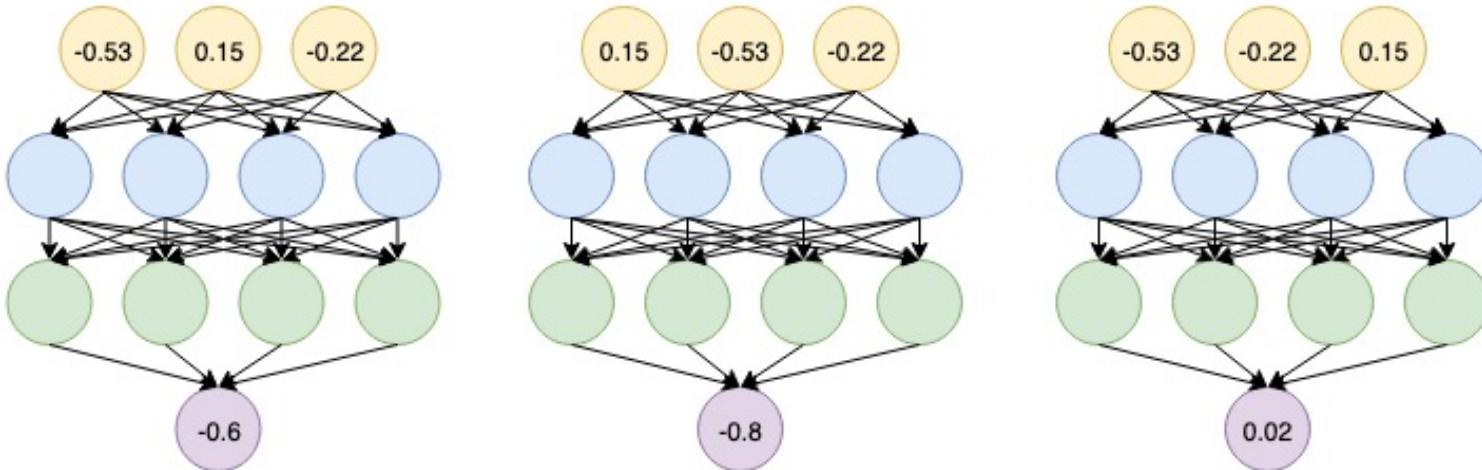
- Graph neural networks consist of multiple permutation equivariant/invariant functions



Graph Neural Network Overview

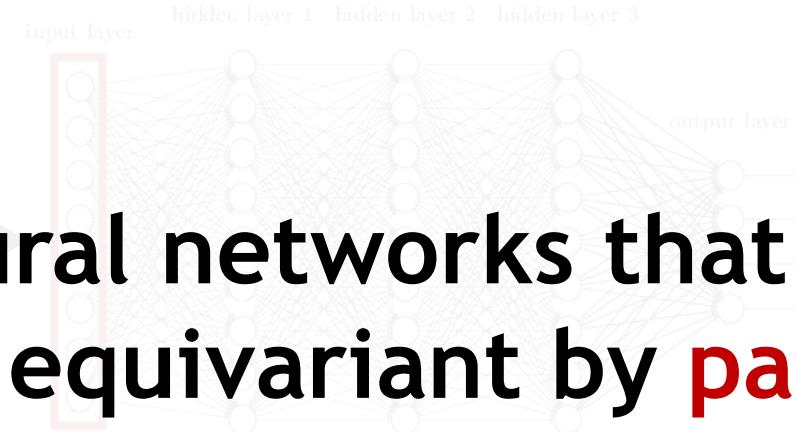
- Are other neural network architectures, e.g., MLPs, permutation invariant/equivariant?
 - No

Switching the order of the input leads to different outputs!



Graph Neural Network Overview

- Are other neural network architectures, e.g., MLPs, permutation invariant/equivariant?
 - No

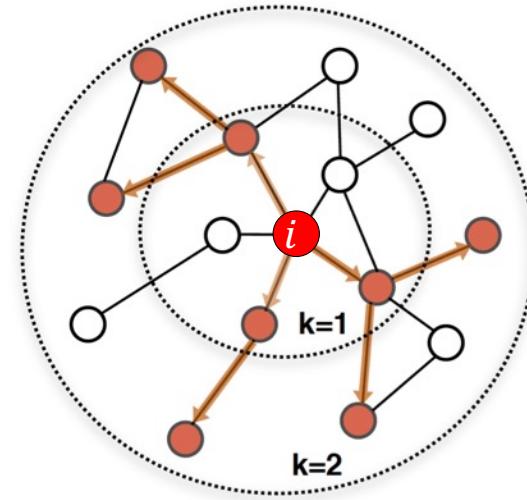


Next: Design graph neural networks that are permutation invariant / equivariant by **passing and aggregating information from neighbors**

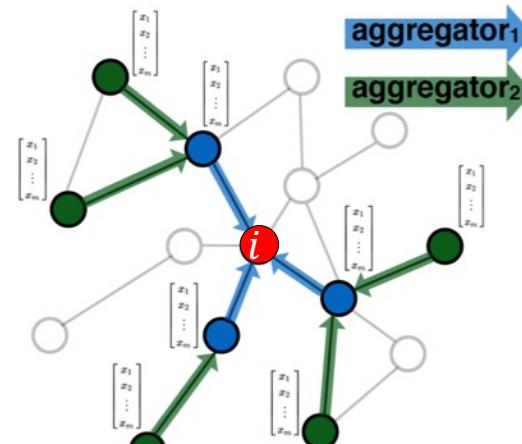
This explains why the naïve MLP approach cannot be used for graphs

Graph Convolutional Networks

- Idea: Node's neighborhood defines a computation graph



Determine node
computation graph

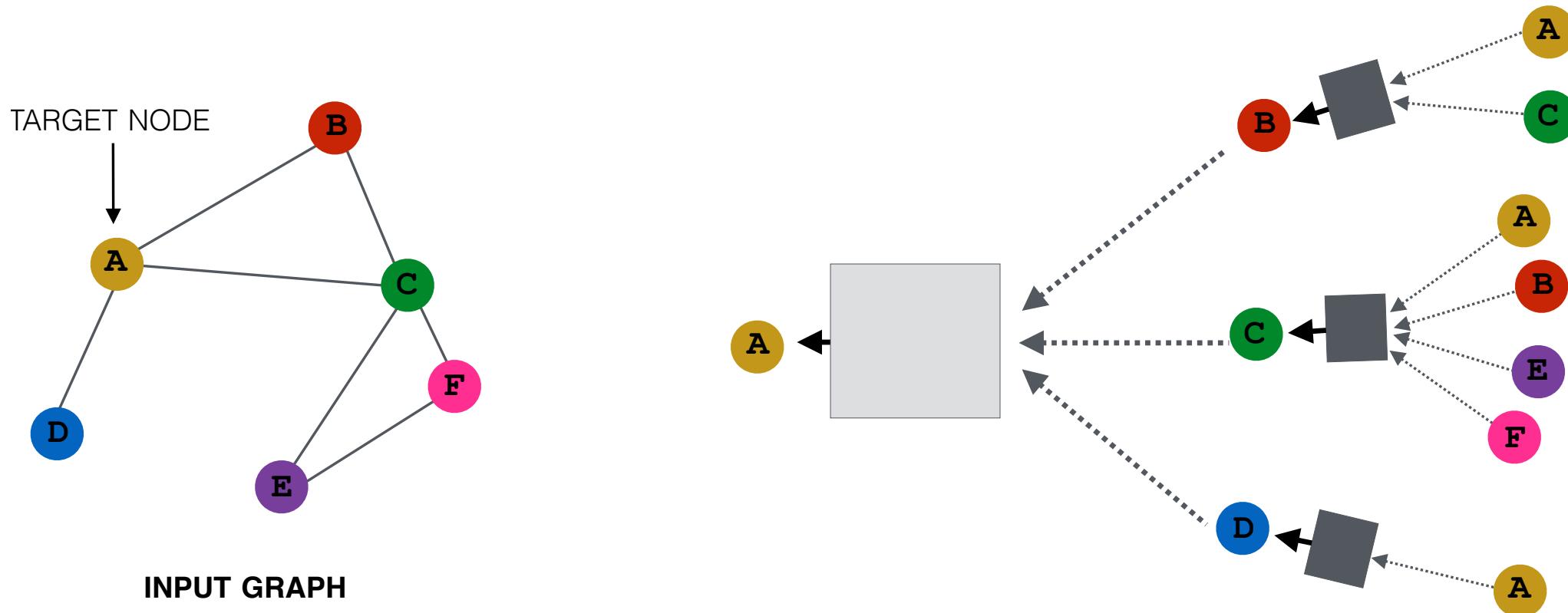


Propagate and
transform information

- Learn how to propagate information across the graph to compute node features

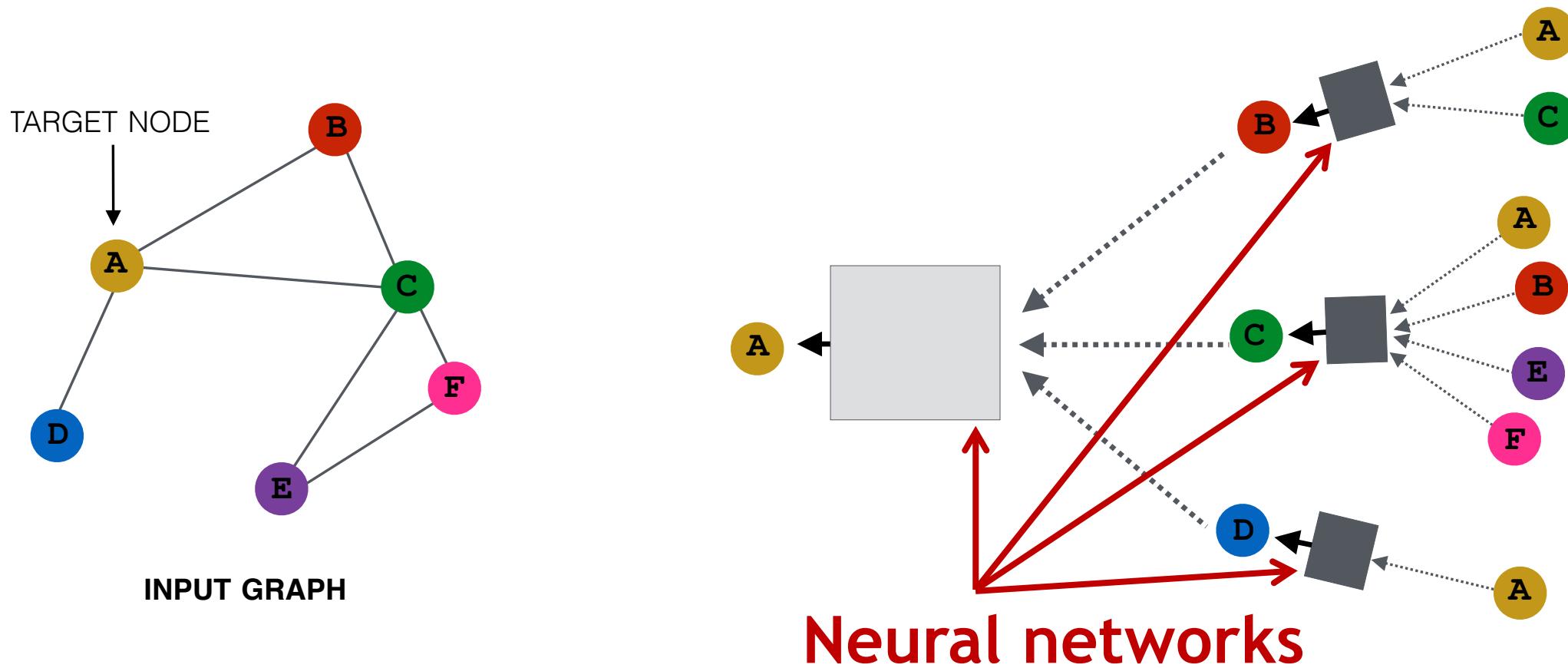
Idea: Aggregate Neighbors

- Key idea: generate node embeddings based on **local graph neighborhoods**



Idea: Aggregate Neighbors

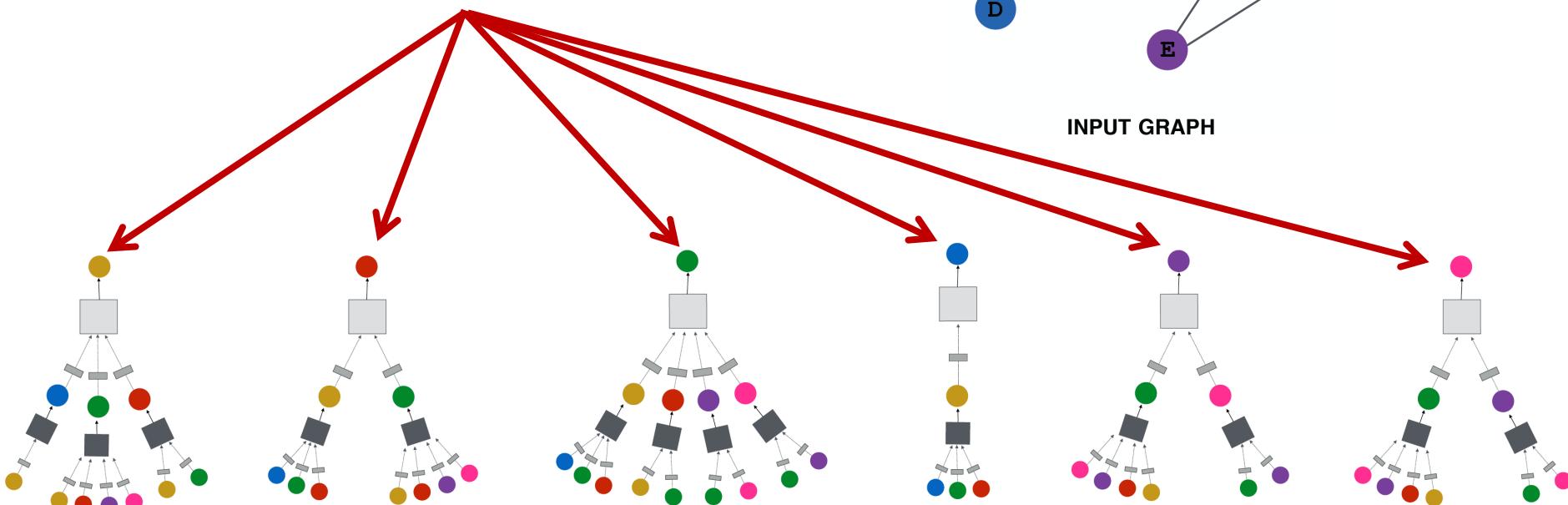
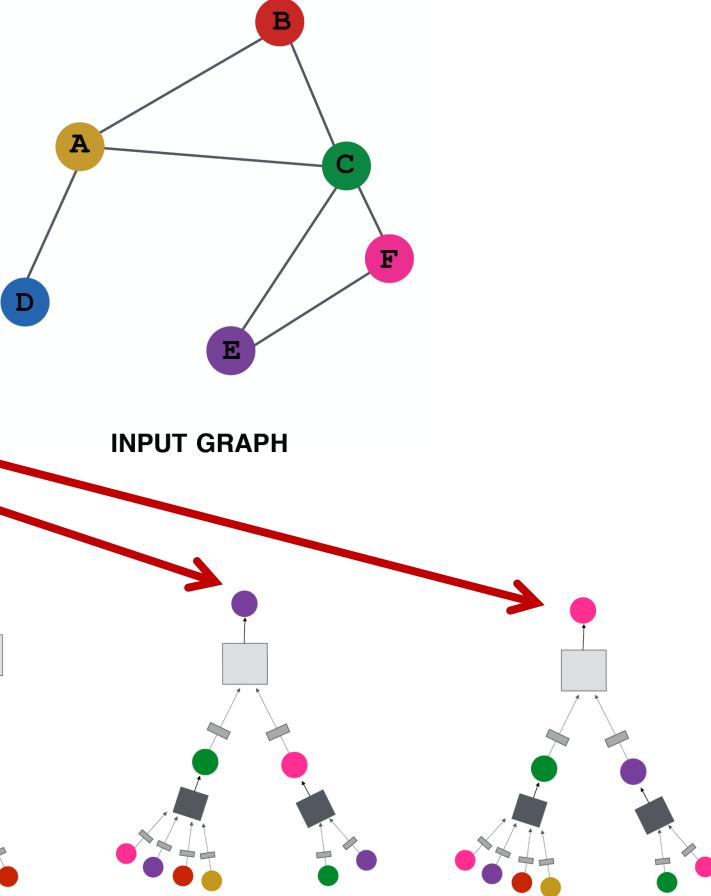
- Nodes aggregate information from their neighbors using neural networks



Idea: Aggregate Neighbors

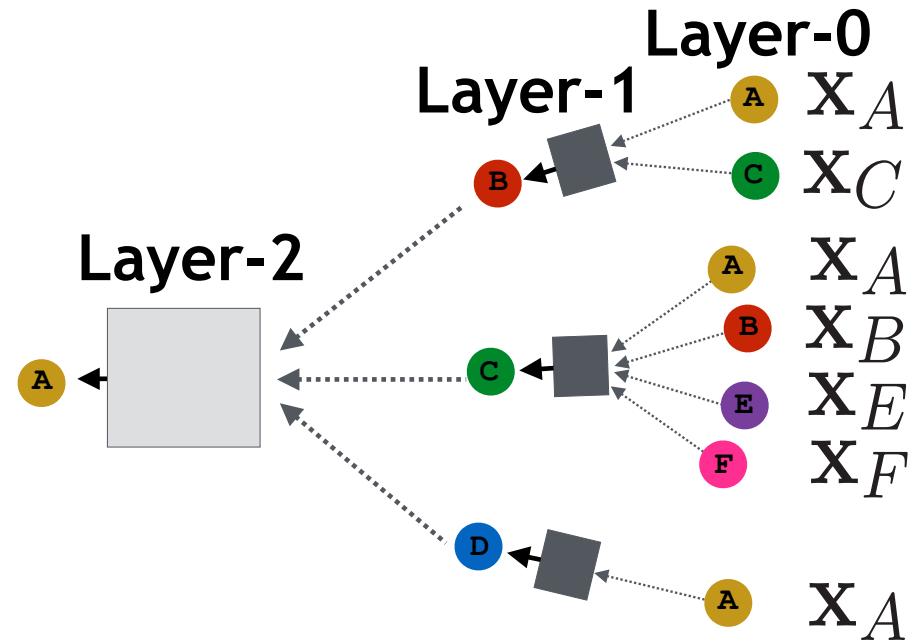
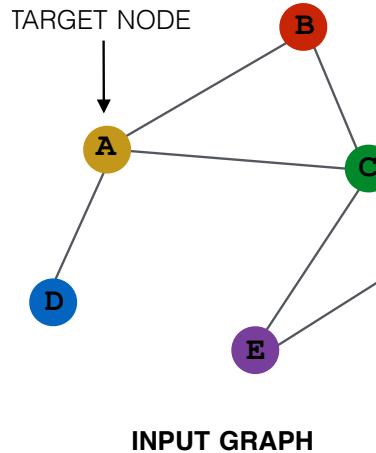
- Graph neighborhood defines a **computation graph**

Every node defines a computation graph based on its neighborhood!



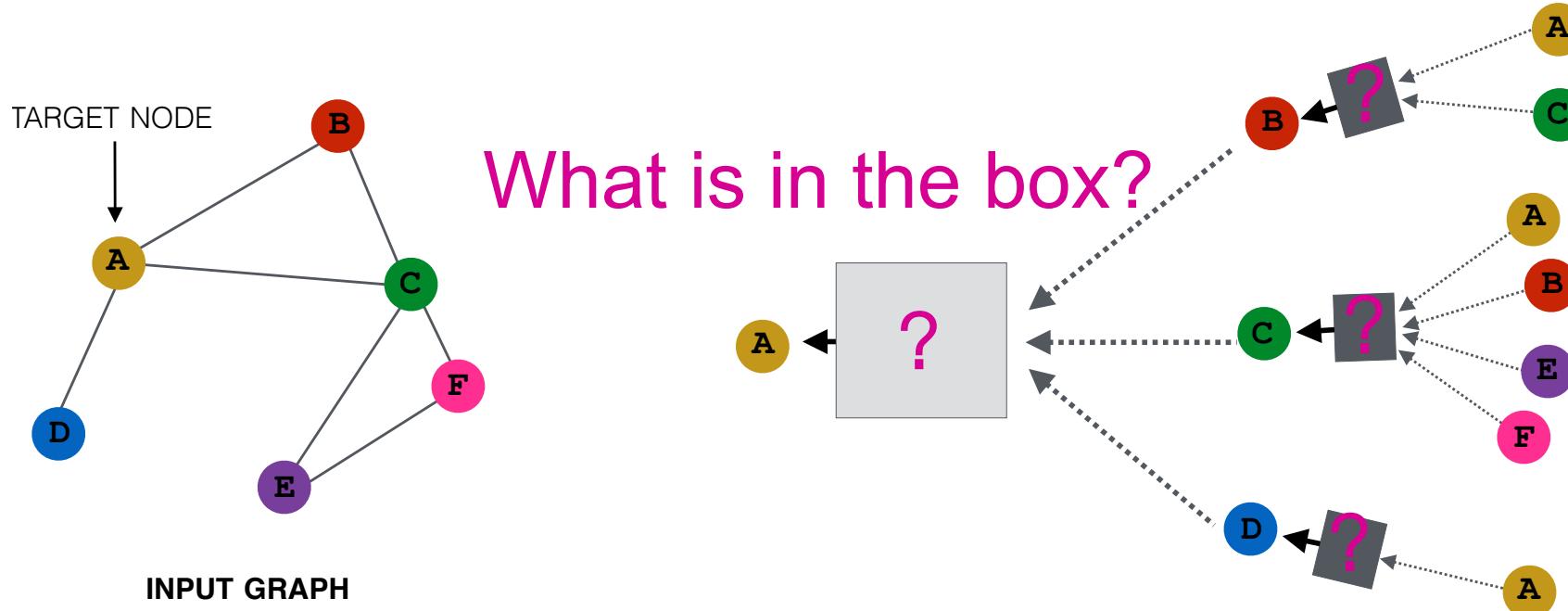
Deep Model: Many Layers

- Model can be of **arbitrary depth**:
 - Nodes have **embeddings** at each layer
 - Layer-0 embedding of node v is its input feature x_v
 - Layer- k embedding gets information from nodes that are **k hops** away



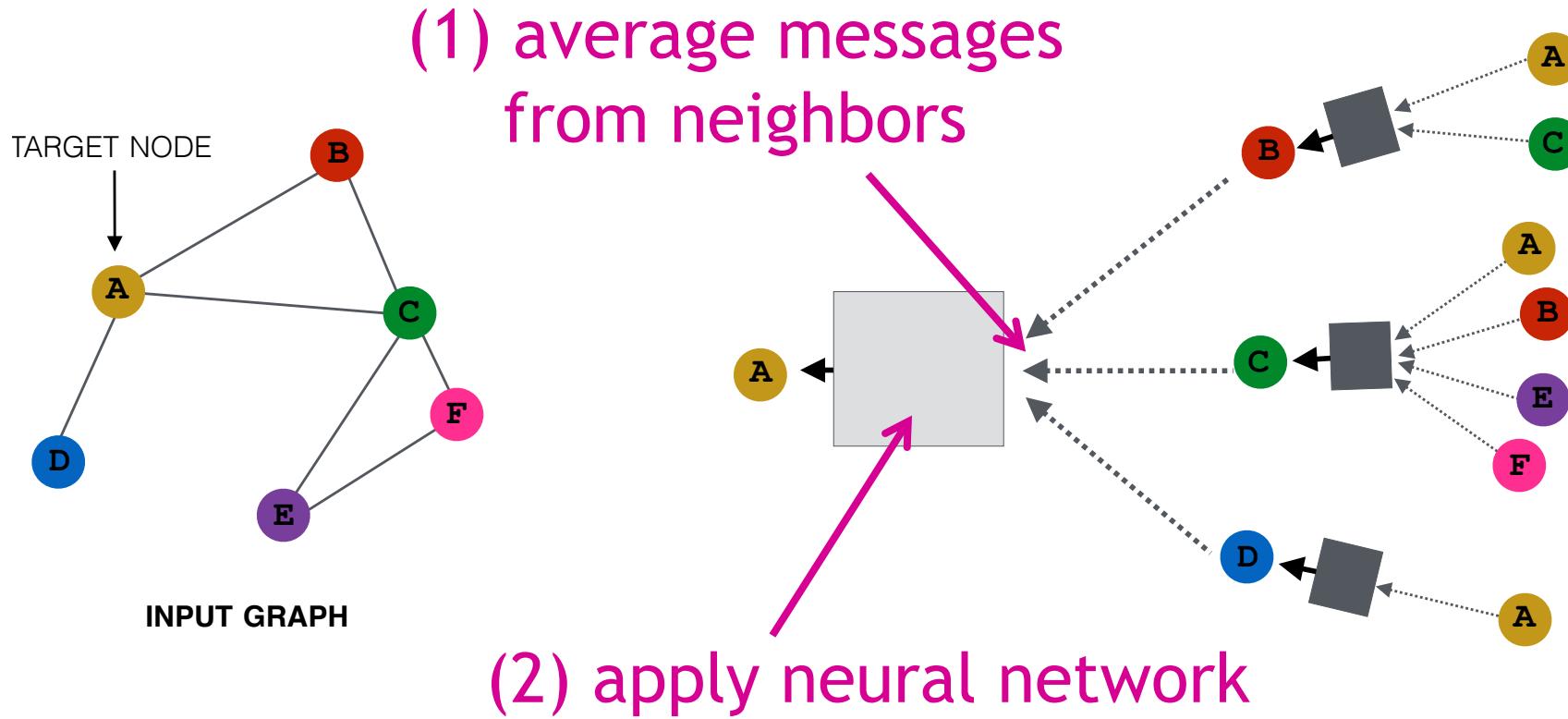
Neighborhood Aggregation

- Neighborhood aggregation: Key distinctions are in how different approaches aggregate information across the layers



Neighborhood Aggregation

- **Basic approach:** average information from neighbors and apply a neural network



The Math: Deep Encoder of a GCN

- **Basic approach:** average information from neighbors and apply a neural network

$$h_v^0 = x_v$$

Initial 0-th layer embeddings are equal to node features

$$h_v^{(k+1)} = \sigma(W_k \sum_{u \in N(v)} \frac{h_u^{(k)}}{|N(v)|} + B_k h_v^{(k)}), \forall k \in \{0, \dots, K-1\}$$

$$z_v = h_v^{(K)}$$

Embedding after K layers of neighborhood aggregation

The Math: Deep Encoder of a GCN

- **Basic approach:** average information from neighbors and apply a neural network

$$h_v^0 = x_v$$

$$h_v^{(k+1)} = \sigma(W_k \sum_{u \in N(v)} \frac{h_u^{(k)}}{|N(v)|} + B_k h_v^{(k)}), \forall k \in \{0, \dots, K-1\}$$

$$z_v = h_v^{(K)}$$

Average of neighbor's previous layer embeddings

embedding of v at layer k

Total number of layers

Notice summation is a permutation invariant pooling/aggregation

The Math: Deep Encoder of a GCN

- **Basic approach:** average information from neighbors and apply a neural network

$$h_v^0 = x_v$$
$$h_v^{(k+1)} = \sigma(W_k \left(\sum_{u \in N(v)} \frac{h_u^{(k)}}{|N(v)|} + B_k h_v^{(k)} \right)), \forall k \in \{0, \dots, K-1\}$$

embedding of v at layer k

Average of neighbor's previous layer embeddings

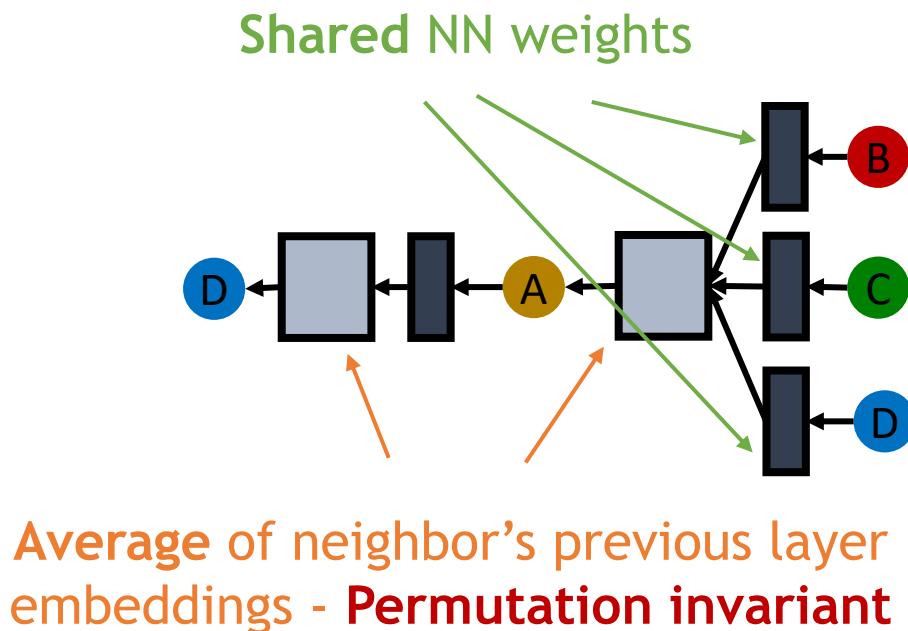
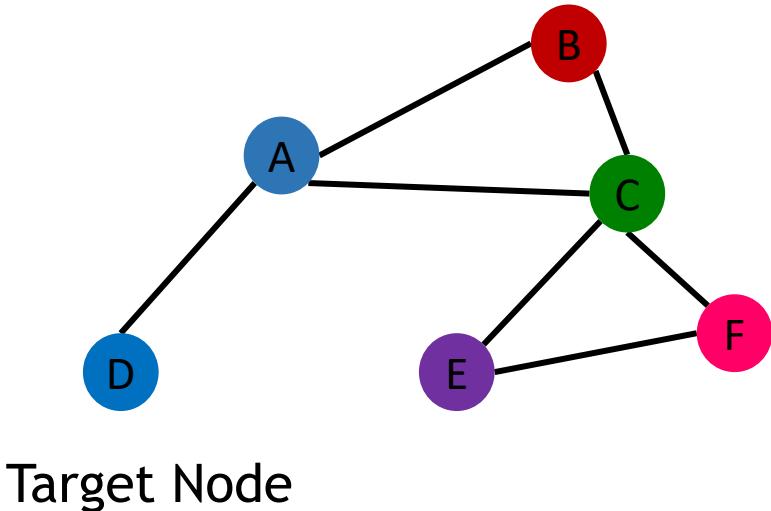
Non-linearity (e.g., ReLU)

Notice summation is a permutation invariant pooling/aggregation

Total number of layers

GCN: Invariance and Equivariance

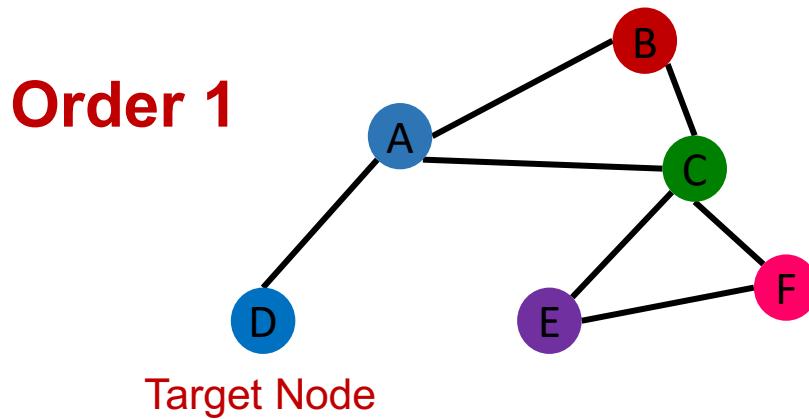
- What are the **invariance** and **equivariance** properties for a GCN?
 - Given a node, the GCN that computes its embedding is **permutation invariant**



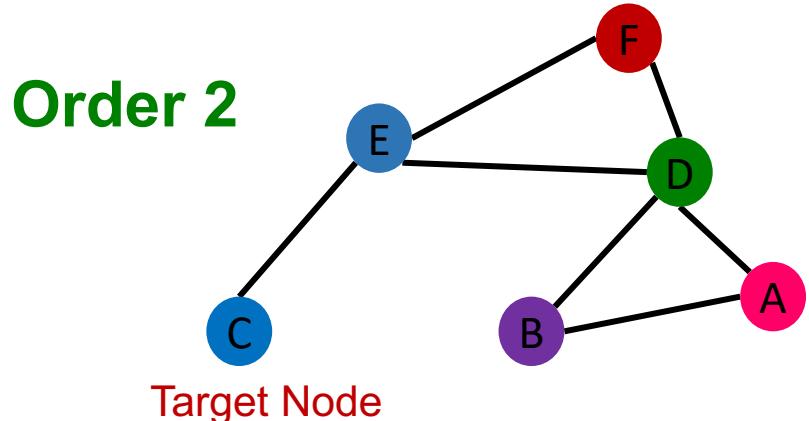
GCN: Invariance and Equivariance

- Considering all nodes in a graph, GCN computation is permutation equivariant

Order 1



Order 2



Node feature X_1

A	[Yellow]
B	[Red]
C	[Green]
D	[Blue]
E	[Purple]
F	[Pink]

Adjacency matrix A_1

A		B	C	D	E	F
A						
B						
C						
D						
E						
F						

Embeddings H_1

A	[Yellow]	
B	[Red]	
C	[Green]	
D	[Blue]	
E	[Purple]	
F	[Pink]	

Permute the input, the output also permutes accordingly - permutation equivariant

Node feature X_2

A	[Pink]
B	[Purple]
C	[Blue]
D	[Green]
E	[Yellow]
F	[Red]

Adjacency matrix A_2

A		B	C	D	E	F
A						
B						
C						
D						
E						
F						

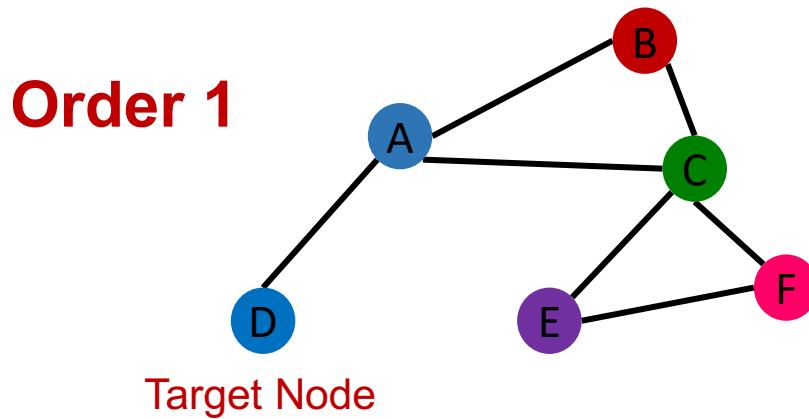
Embeddings H_2

A	[Red]	
B	[Purple]	
C	[Blue]	
D	[Green]	
E	[Yellow]	
F	[Red]	

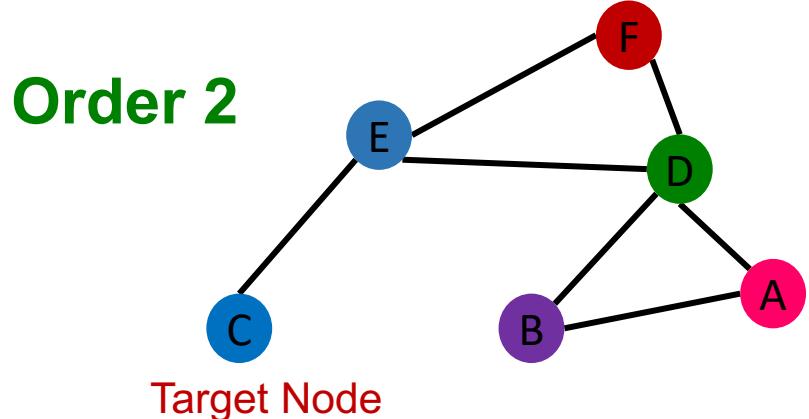
GCN: Invariance and Equivariance

- Considering all nodes in a graph, GCN computation is permutation equivariant

Order 1



Order 2



Node feature X_1

A	[Color Bar]
B	[Color Bar]
C	[Color Bar]
D	[Color Bar]
E	[Color Bar]
F	[Color Bar]

Adjacency matrix A_1

A	B	C	D	E	F
B					
C					
D					
E					
F					

Embeddings H_1

A	[Color Bar]
B	[Color Bar]
C	[Color Bar]
D	[Color Bar]
E	[Color Bar]
F	[Color Bar]

Permute the input, the output also permutes accordingly - permutation equivariant

Node feature X_2

A	[Color Bar]
B	[Color Bar]
C	[Color Bar]
D	[Color Bar]
E	[Color Bar]
F	[Color Bar]

Adjacency matrix A_2

A	B	C	D	E	F
B					
C					
D					
E					
F					

Embeddings H_2

A	[Color Bar]
B	[Color Bar]
C	[Color Bar]
D	[Color Bar]
E	[Color Bar]
F	[Color Bar]

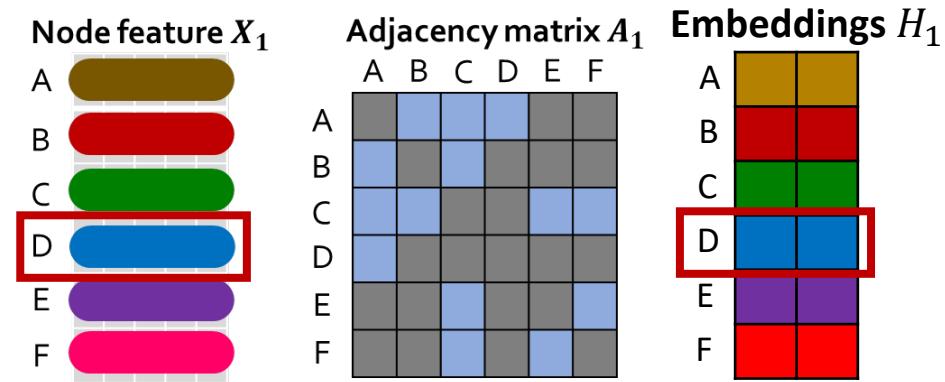
GCN: Invariance and Equivariance

- Considering all nodes in a graph, GCN computation is permutation equivariant

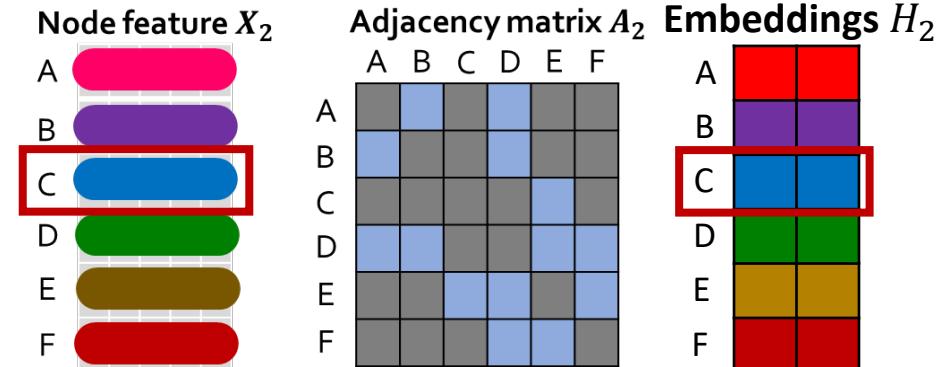
Detailed reasoning:

- The rows of **input node features** and **output embeddings** are aligned
- We know computing the embedding of **a given node** with GCN is **invariant**
- So, after permutation, the **location** of **a given node** in the **input node feature matrix** is changed, and the **the output embedding of a given node stays the same** (**the colors of node feature and embedding are matched**)

This is **permutation equivariant**

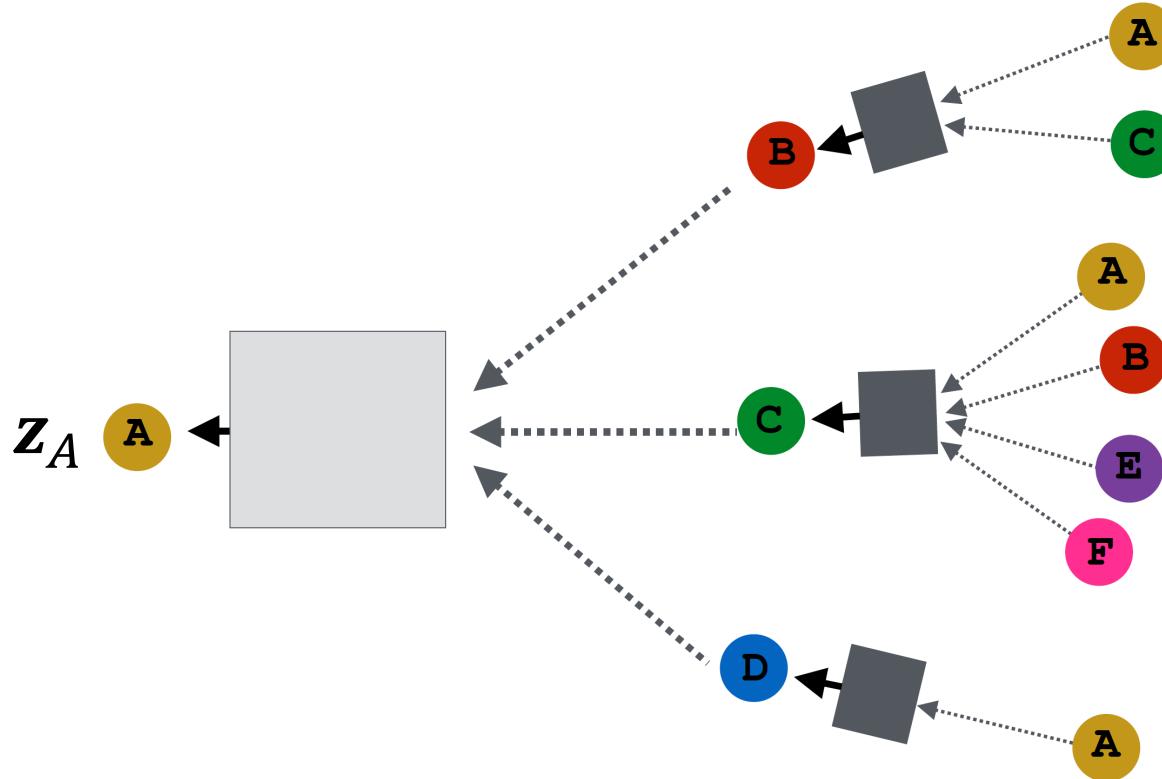


Permute the input, the output also permutes accordingly - **permutation equivariant**



Training the Model

- How do we train the GCN to generate embeddings?



Need to define a loss function on the embeddings

Model Parameters

Trainable weight matrices
(i.e., what we learn)

$$h_v^{(0)} = x_v$$
$$h_v^{(k+1)} = \sigma(W_k \sum_{u \in N(v)} \frac{h_u^{(k)}}{|N(v)|} + B_k h_v^{(k)}), \forall k \in \{0..K-1\}$$
$$z_v = h_v^{(K)}$$

Final node embedding

- We can feed these **embeddings** into any **loss function** and run SGD to **train the weight parameters**
 - h_v^k : the hidden representation of node v at layer k
 - W_k : weight matrix for neighborhood aggregation
 - B_k : weight matrix for transforming hidden vector of self

Matrix Formulation (1)

- Many aggregations can be performed efficiently by (sparse matrix operations)

- Let $H^{(k)} = [h_1^{(k)} \dots h_{|V|}^{(k)}]^T$

- Then, $\sum_{u \in N_v} h_u^{(k)} = A_{v,:} H^{(k)}$

- Let D be a diagonal matrix where

- $D_{v,v} = \text{Deg}(v) = |N(v)|$

- The inverse of D: D^{-1} is also diagonal

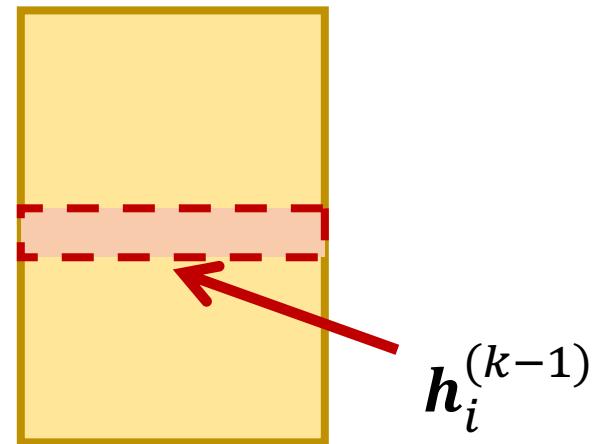
- $D_{v,v}^{-1} = 1/|N(v)|$

- Therefore

$$\sum_{u \in N(v)} \frac{h_u^{(k-1)}}{|N(v)|}$$

$$H^{(k+1)} = D^{-1} A H^{(k)}$$

Matrix of hidden embeddings $H^{(k-1)}$

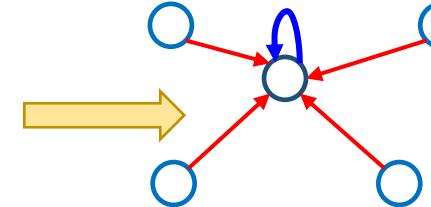


Matrix Formulation (2)

- Re-writing update function in matrix form:

$$H^{(k+1)} = \sigma(\tilde{A}H^{(k)}W_k^T + H^{(k)}B_k^T)$$

where $\tilde{A} = D^{-1}A$



$$H^{(k)} = [h_1^{(k)} \dots h_{|V|}^{(k)}]^T$$

- Red: neighborhood aggregation
- Blue: self-transformation
- In practice, this implies that efficient sparse matrix multiplication can be used (\tilde{A} is sparse)
- Note: not all GNNs can be expressed in a simple matrix form, when aggregation function is complex

How to Train A GNN

- Node embedding \mathbf{z}_v is a function of input graph
- **Supervised setting:** we want to minimize loss \mathcal{L} :

$$\min_{\Theta} \mathcal{L}(\mathbf{y}, f_{\Theta}(\mathbf{z}_v))$$

- \mathbf{y} : node label
- \mathcal{L} could be **sum of squared error (SSE)** if \mathbf{y} is a real number, or **cross-entropy** if \mathbf{y} is categorical
- **Unsupervised setting:**
 - No node label available
 - Use the graph structure as the supervision

Unsupervised Training

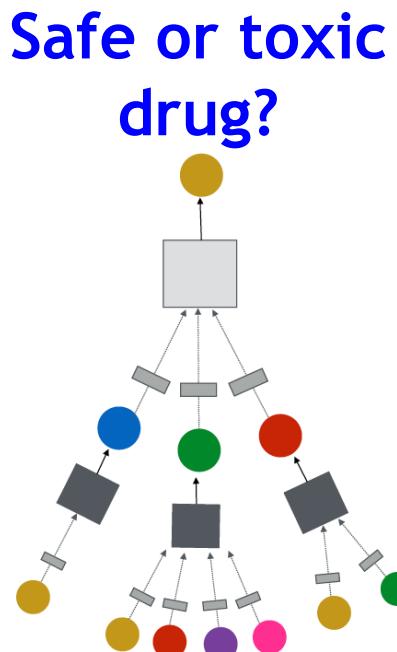
- One possible idea: “similar” nodes have similar embeddings

$$\min_{\Theta} \mathcal{L} = \sum_{z_u, z_v} \text{CE}(y_{u,v}, \text{DEC}(z_u, z_v))$$

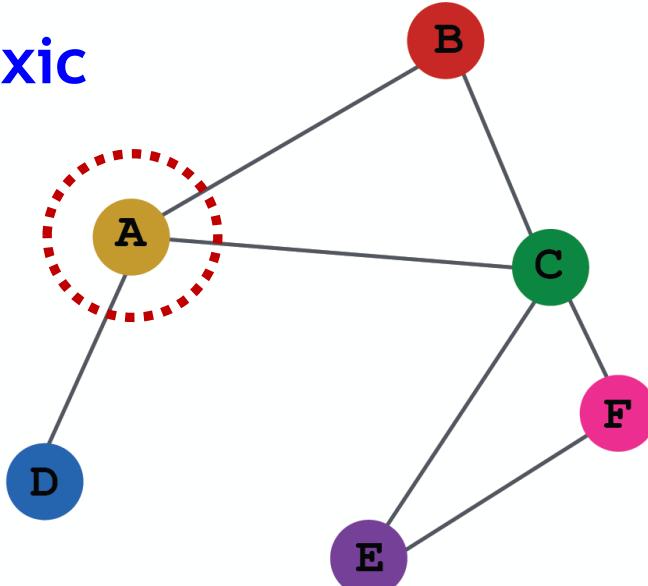
- where $y_{u,v}=1$ when node u and v are similar
- $z_u = f_{\Theta}(u)$ and $\text{DEC}(\cdot, \cdot)$ is the dot product
- CE is the cross-entropy loss $\text{CE}(y, f(x)) = -\sum_{i=1}^C (y_i \log f_{\Theta}(x)_i)$
 - y_i and $f_{\Theta}(x)_i$ are the **actual** and **predicted** values of the i -th class
 - **intuition:** the lower the loss, the closer the prediction is to one-hot
- Node similarity can be anything from the previous lecture, e.g., loss based on:
 - Random walks (node2vec, DeepWalk, struct2vec)

Supervised Training

- Directly train the model for a supervised task (e.g., node classification)



Safe or toxic drug?



E.g., a drug-drug interaction network

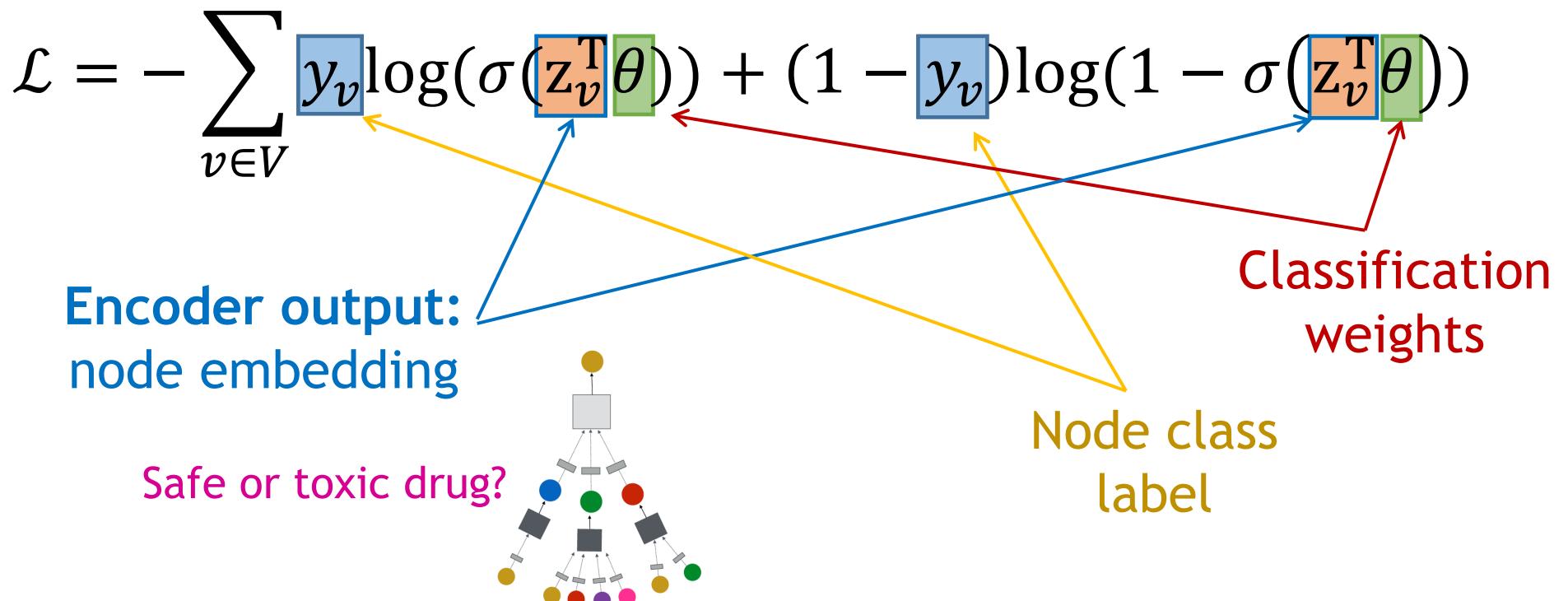
Supervised Training

- Directly train the model for a supervised task (e.g., node classification)
 - Use cross-entropy loss (e.g., binary classification)

$$\mathcal{L} = - \sum_{v \in V} y_v \log(\sigma(z_v^T \theta)) + (1 - y_v) \log(1 - \sigma(z_v^T \theta))$$

Supervised Training

- Directly train the model for a supervised task (e.g., node classification)
 - Use cross-entropy loss (e.g., binary classification)





COSC 3337
Data Science I
Section 14623

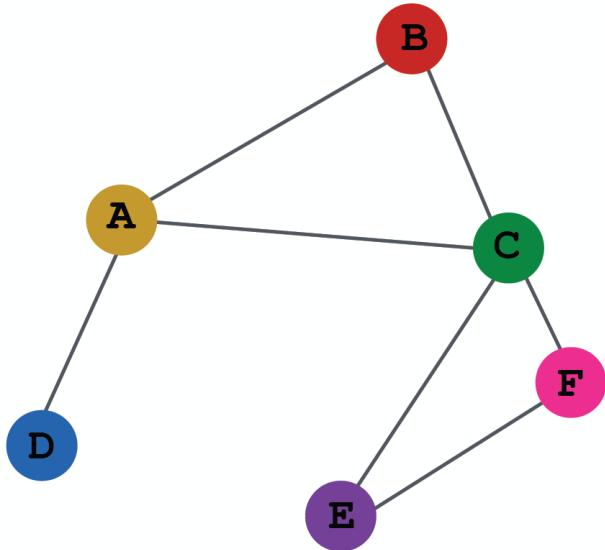
Graph Data Analysis

Instructor: Jingchao Ni
Fall 2024

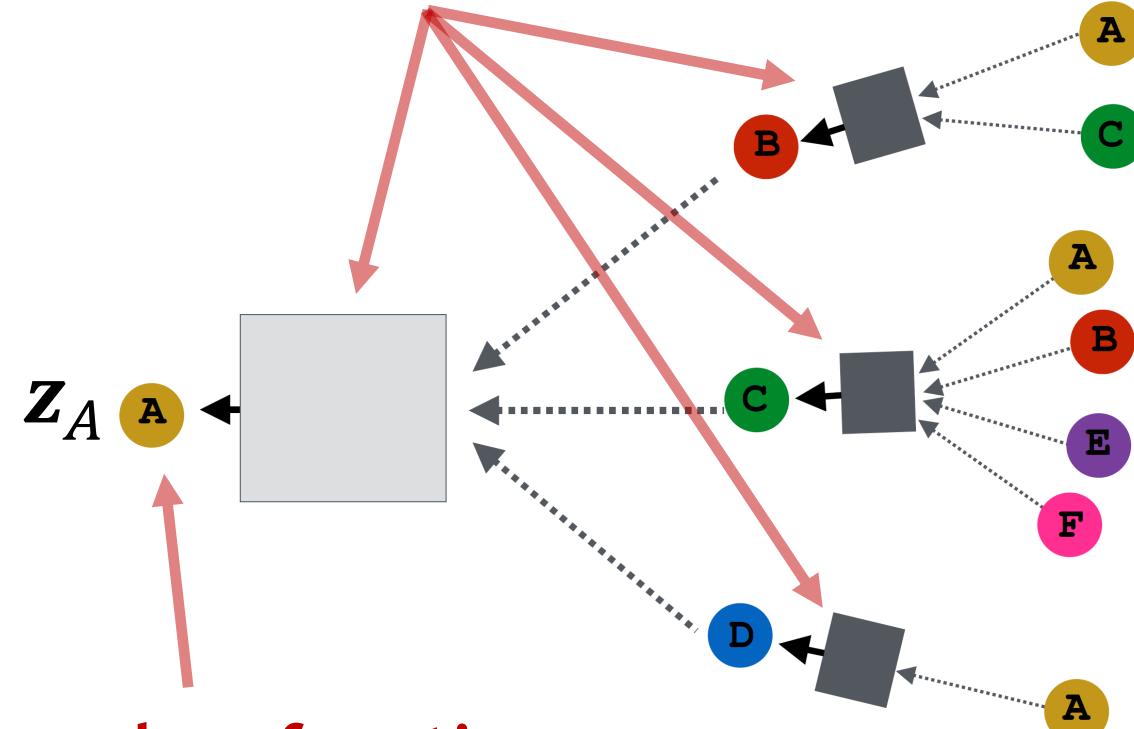
About Exam II

- Clustering
 - Hierarchical clustering, DBSCAN, k-means clustering
- Anomaly Detection
 - Proximity-based method, density-based method
- Deep Learning
 - Chain rule, backpropagation, RNN, Long Short-Term Memory (LSTM)
- Graph Data Analysis
 - “Shallow” node embedding, permutation invariance and equivariance

Model Design: Overview

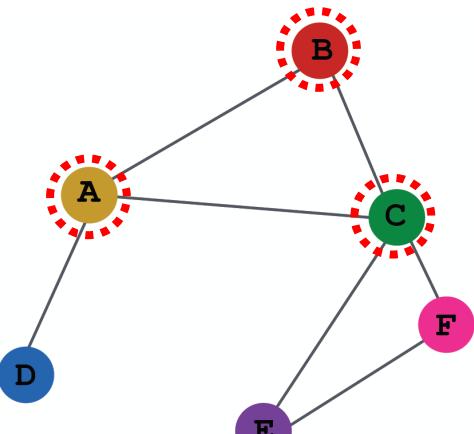


(1) Define a neighborhood aggregation function



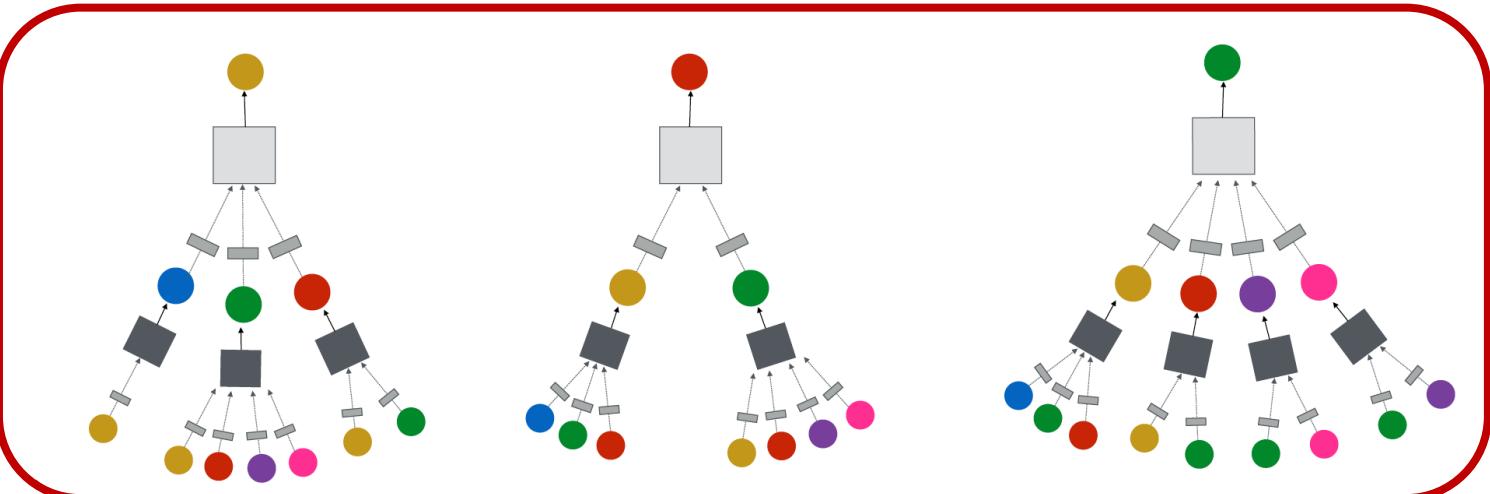
(2) Define a loss function on the embeddings

Model Design: Overview

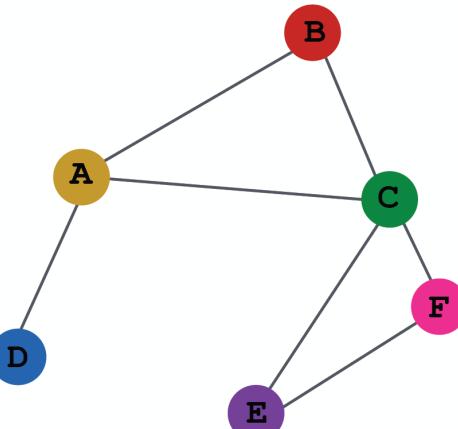


INPUT GRAPH

(3) Train on a set of nodes, i.e.,
a batch of compute graphs



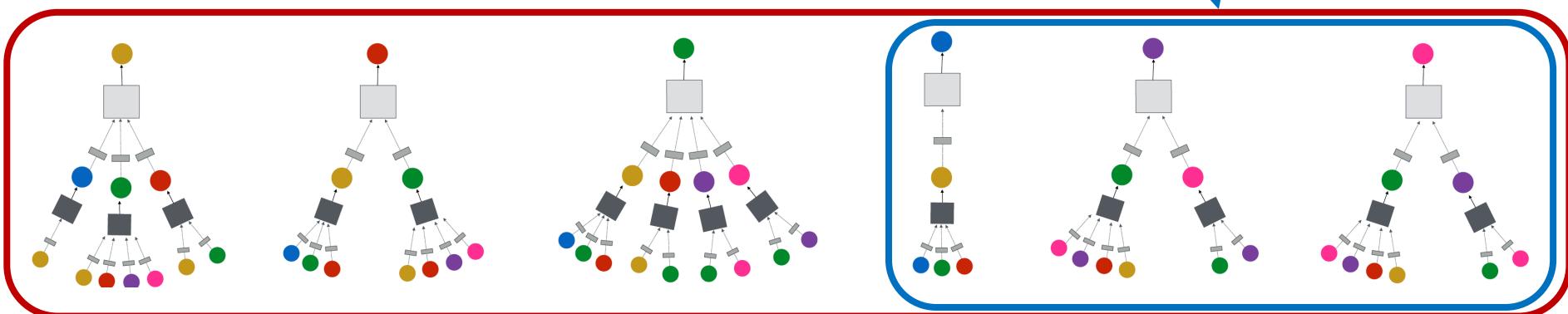
Model Design: Overview



INPUT GRAPH

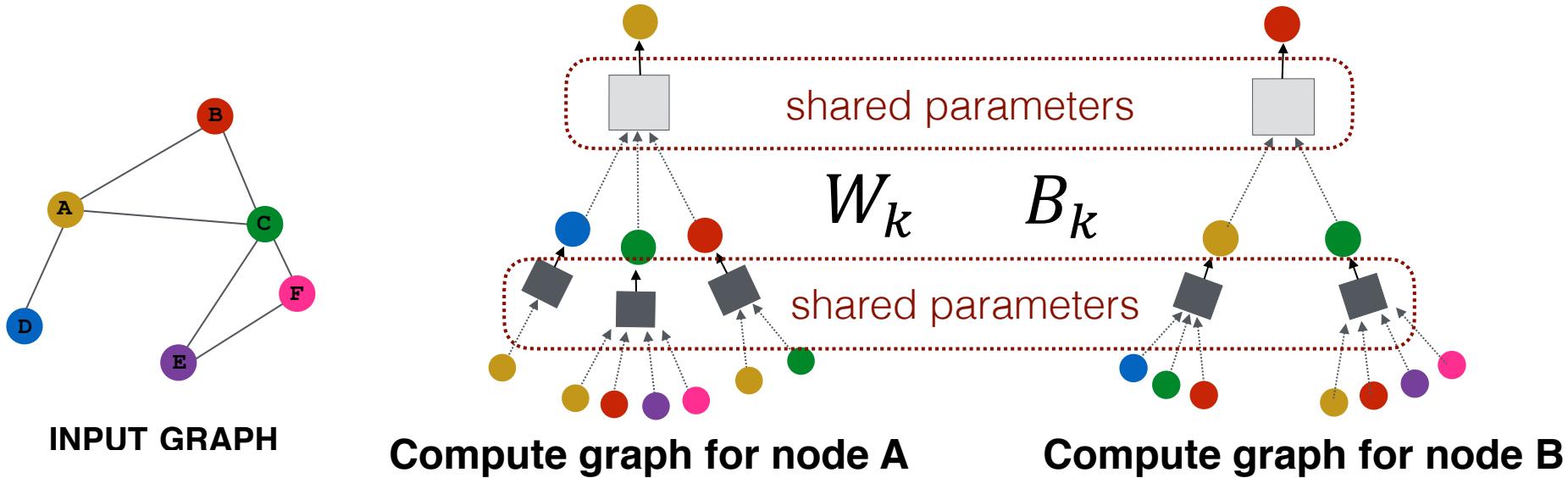
(4) Generate embeddings
for nodes as needed

Even for nodes we never
trained on

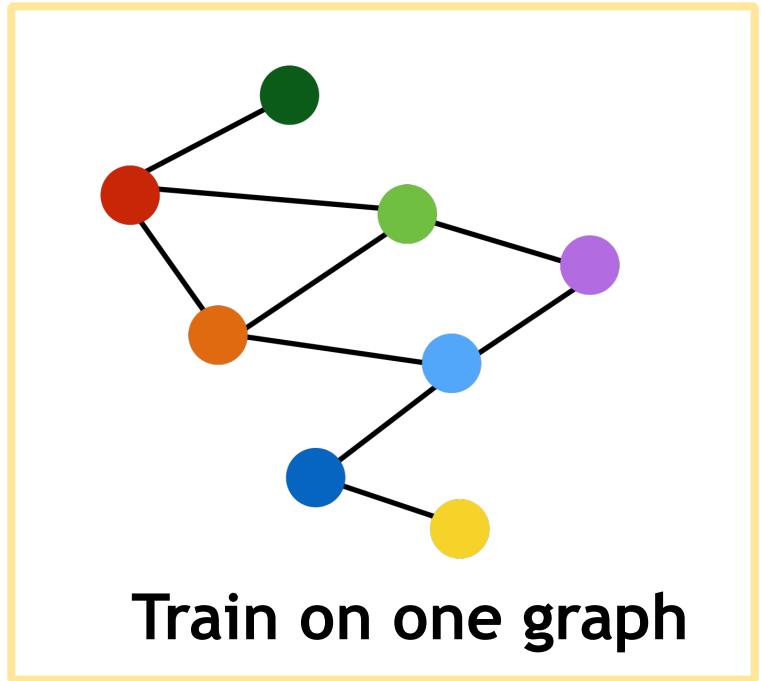


Inductive Capability

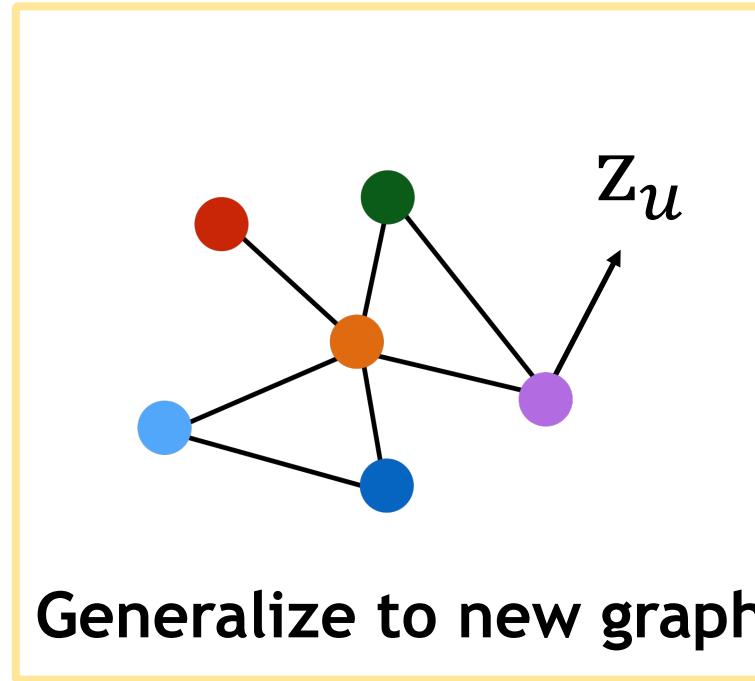
- The same aggregation parameters are shared for all nodes
 - The number of model parameters is sublinear in $|V|$ and we can generalize to unseen nodes



Inductive Capability: New Graphs



Train on one graph



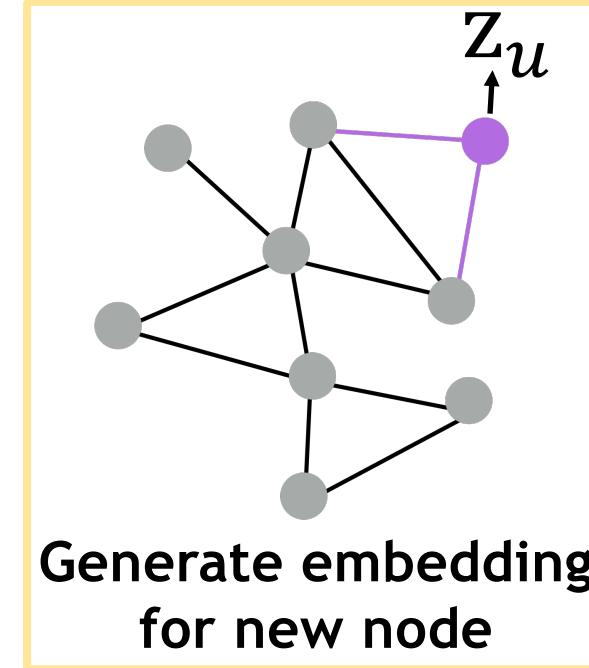
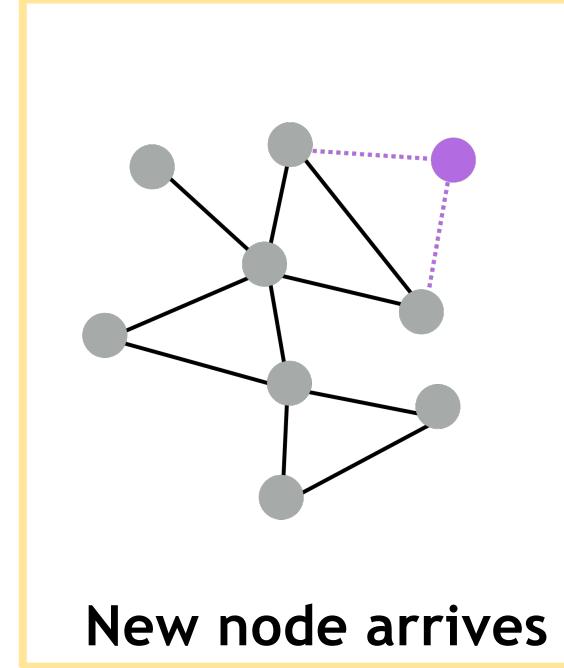
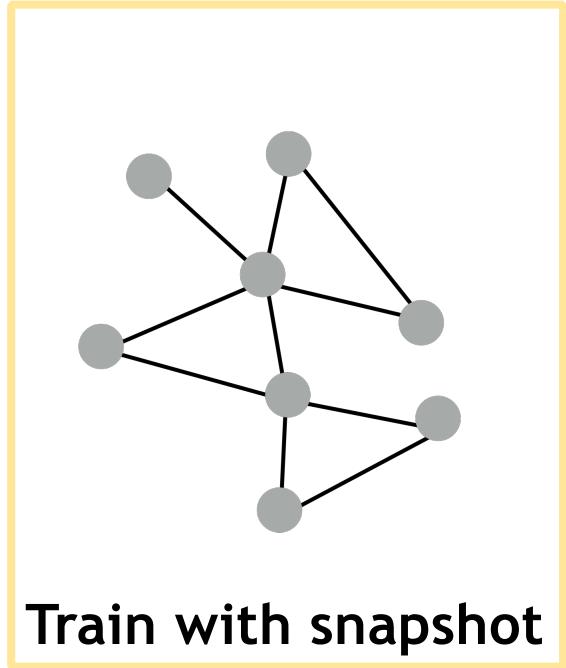
Generalize to new graph

Inductive node embedding →

Generalize to entirely unseen graphs

E.g., train on protein interaction graph from model organism A and generate embeddings on newly collected data about organism B

Inductive Capability: New Nodes



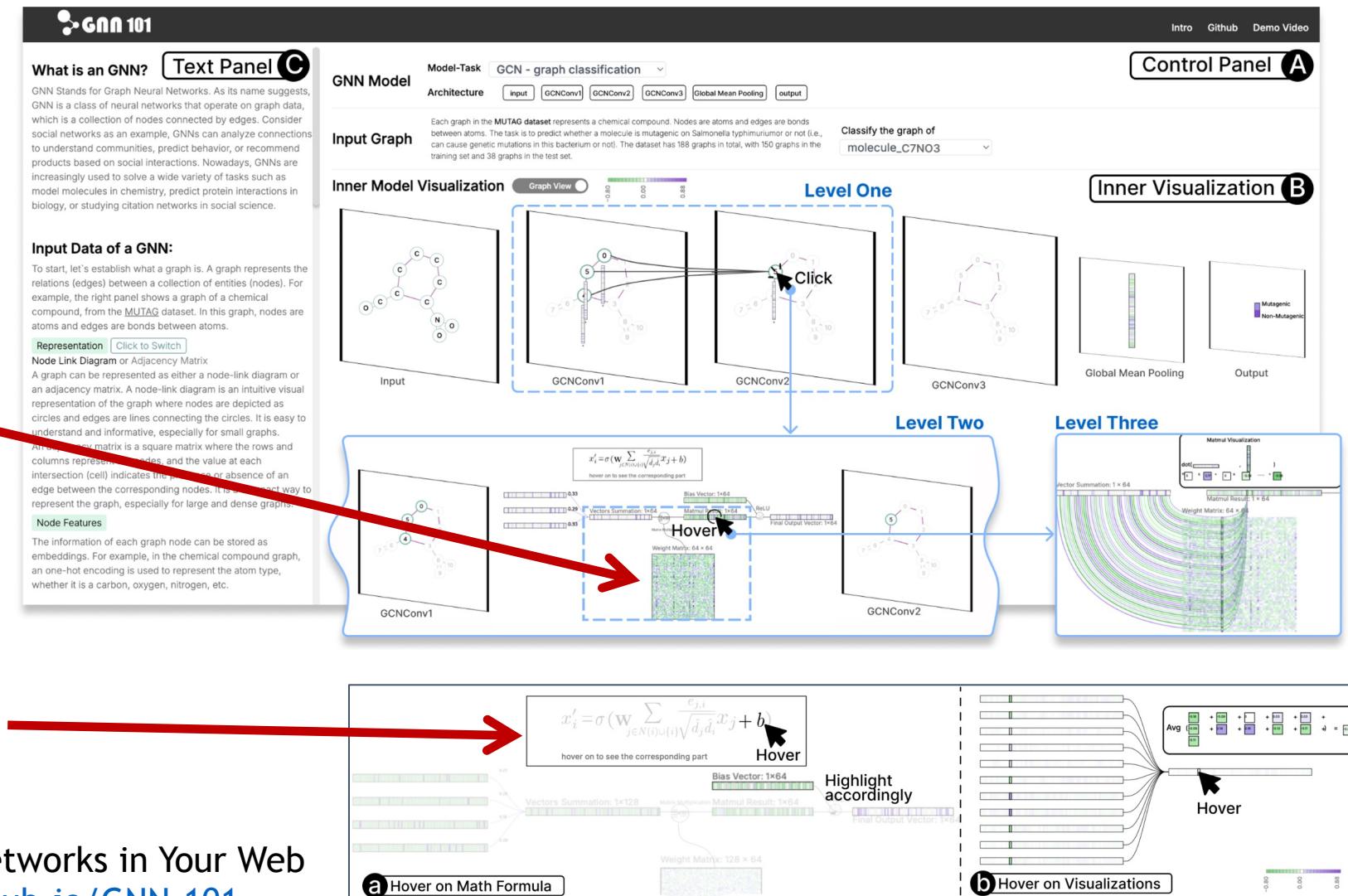
- Many application settings constantly encounter previously unseen nodes:
 - E.g., Reddit, YouTube, Google Scholar
- Need to generate new embeddings “on the fly”

Interactive Visualization of GNNs

Visualizing the inner working of a GNN with a **hierarchical level of detail**

Parameters are shown as heatmaps and curves
animations are the computation process

You can hover terms in the equations, and it will highlight the corresponding visualization



GNN 101: Visual Learning of Graph Neural Networks in Your Web Browser <https://visual-intelligence-umn.github.io/GNN-101>



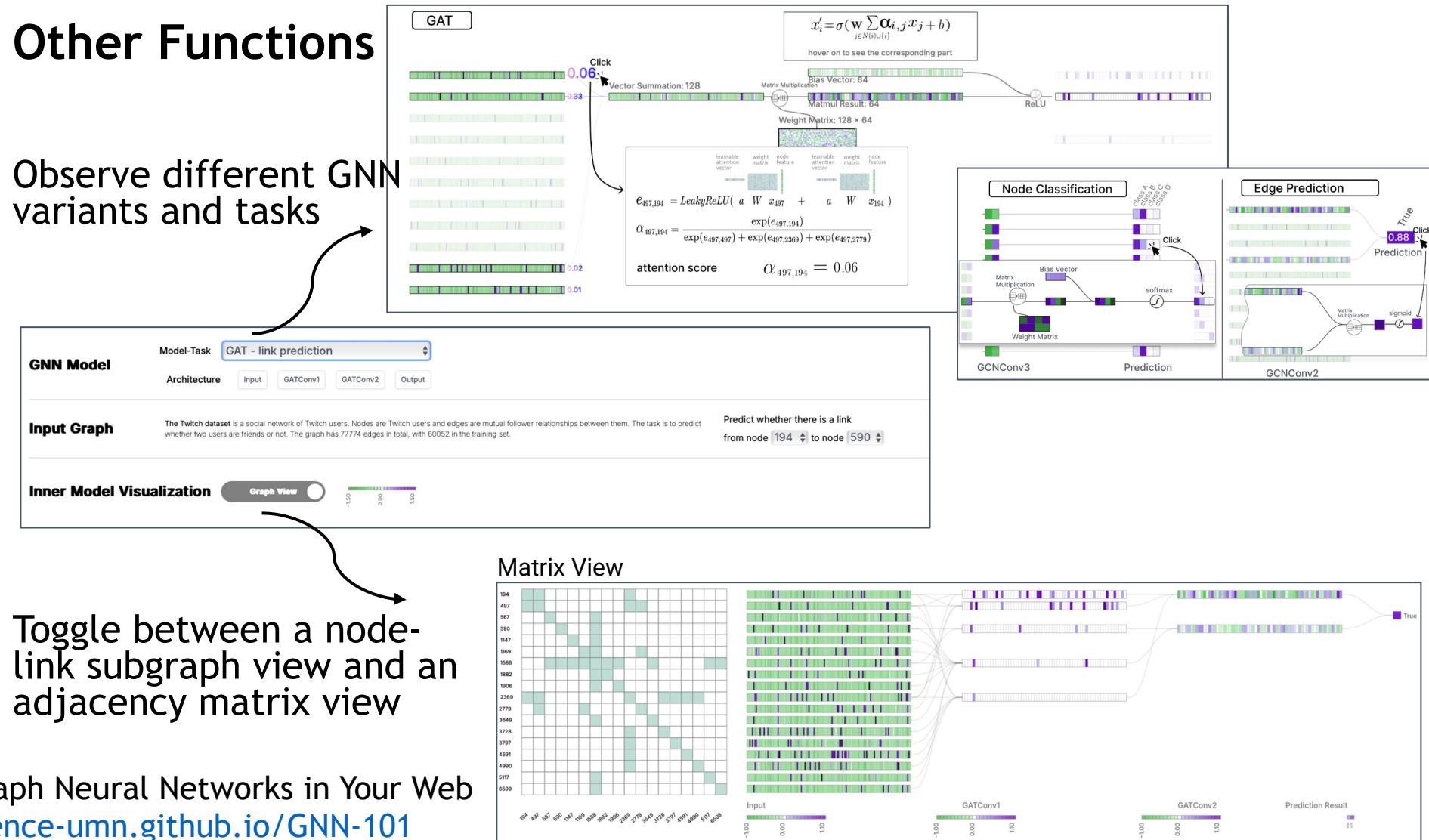
<http://cs224w.Stanford.edu>

Natural Sciences and Mathematics

Interactive Visualization of GNNs

Other Functions

Observe different GNN variants and tasks

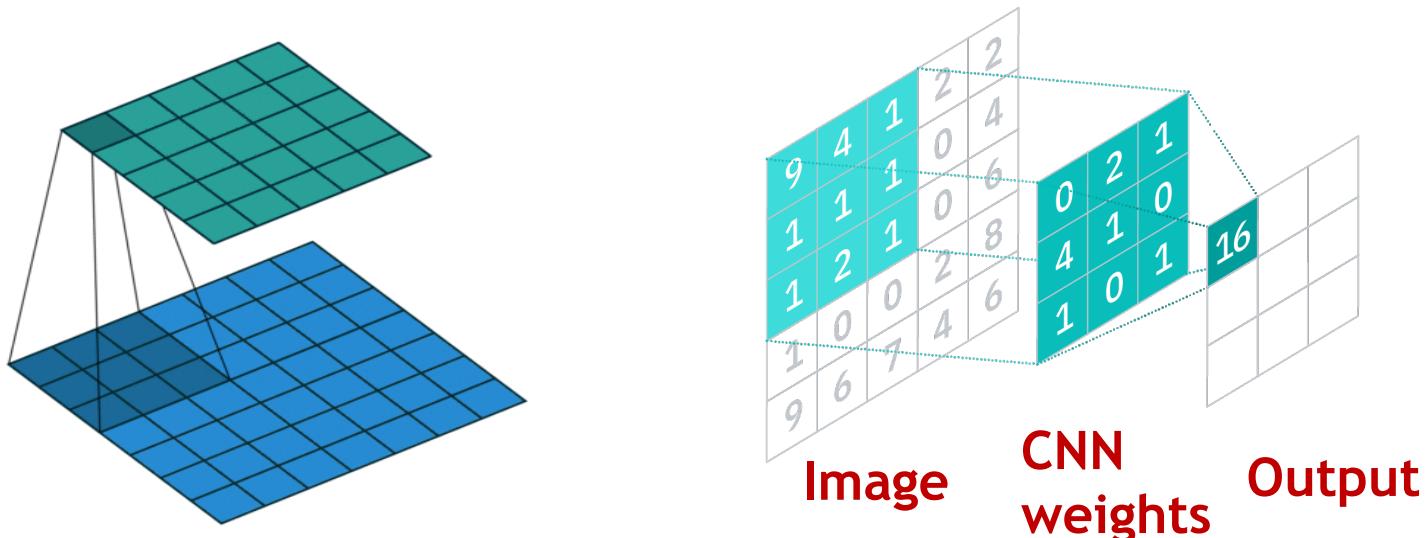


Toggle between a node-link subgraph view and an adjacency matrix view

GNN 101: Visual Learning of Graph Neural Networks in Your Web Browser <https://visual-intelligence-umn.github.io/GNN-101>

GNN vs. CNN

- How do GNNs compare to prominent architectures such as Convolutional Neural Nets (CNNs)?
- **Convolutional neural network (CNN) layer with 3×3 filter:**

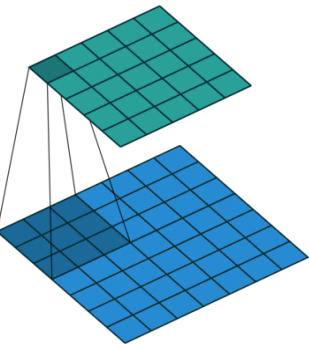


$$\text{CNN formulation: } h_v^{(l+1)} = \sigma(\sum_{u \in N(v) \cup \{v\}} W_l^u h_u^{(l)}), \quad \forall l \in \{0, \dots, L-1\}$$

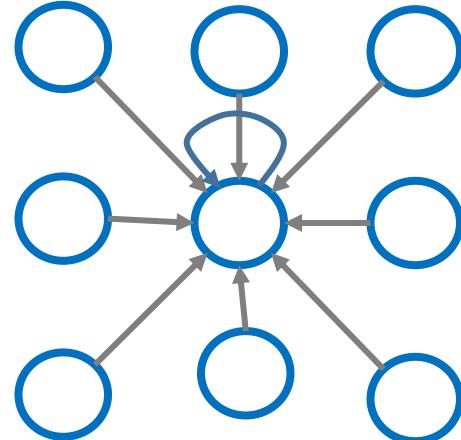
$N(v)$ represents the 8 neighbor pixels of v

GNN vs. CNN

- Convolutional neural network (CNN) layer with 3×3 filter:



Image

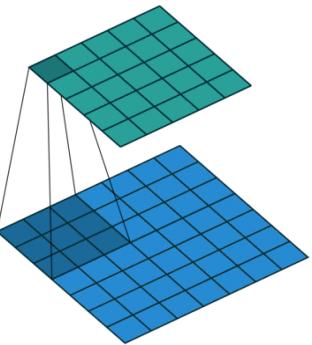


Graph

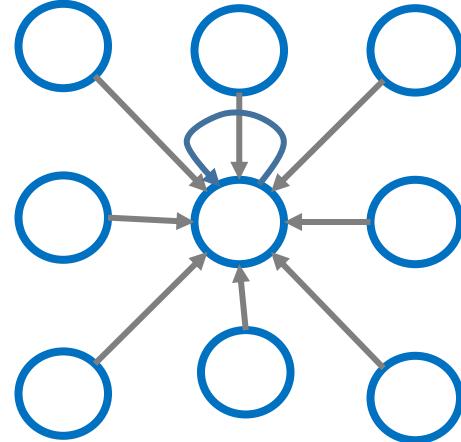
- GNN formulation: $h_v^{(l+1)} = \sigma(\mathbf{W}_l \sum_{u \in N(v)} \frac{h_u^{(l)}}{|N(v)|} + B_l h_v^{(l)})$, $\forall l \in \{0, \dots, L-1\}$
- CNN formulation: (previous slide) $h_v^{(l+1)} = \sigma(\sum_{u \in N(v) \cup \{v\}} W_l^u h_u^{(l)})$, $\forall l \in \{0, \dots, L-1\}$
if we rewrite: $h_v^{(l+1)} = \sigma(\sum_{u \in N(v)} \mathbf{W}_l^u h_u^{(l)} + B_l h_v^{(l)})$, $\forall l \in \{0, \dots, L-1\}$

GNN vs. CNN

- Convolutional neural network (CNN) layer with 3x3 filter:



Image



Graph

$$\text{GNN formulation: } h_v^{(l+1)} = \sigma(\mathbf{W}_l \sum_{u \in N(v)} \frac{h_u^{(l)}}{|N(v)|} + B_l h_v^{(l)}), \forall l \in \{0, \dots, L-1\}$$

$$\text{CNN formulation: } h_v^{(l+1)} = \sigma(\sum_{u \in N(v)} \mathbf{W}_l^u h_u^{(l)} + B_l h_v^{(l)}), \forall l \in \{0, \dots, L-1\}$$

Key difference: We can learn different W_l^u for different “neighbor” u for pixel v on the image. The reason is we can pick an order for the 9 neighbors using relative position to the center pixel: $\{(-1, -1), (-1, 0), (-1, 1), \dots, (1, 1)\}$

GNN vs. CNN

- Convolutional neural network (CNN) layer with 3x3 filter:

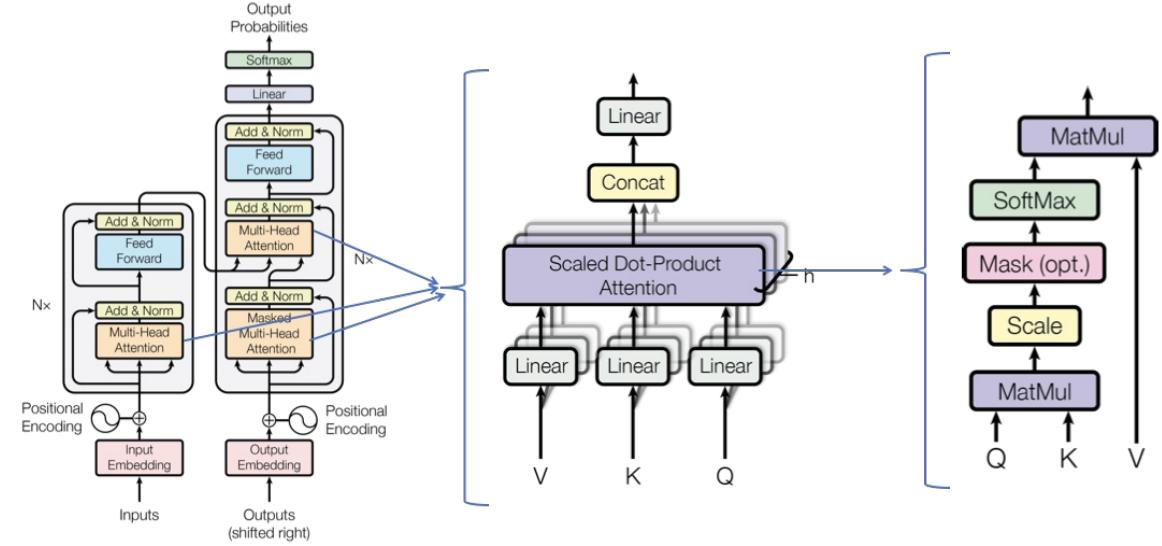


- CNN can be seen as a special GNN with fixed neighbor size and ordering:
 - The size of the filter is pre-defined for a CNN
 - The advantage of GNN is it processes arbitrary graphs with different degrees for each node
- CNN is not permutation invariant/equivariant
 - Switching the order of pixels leads to different outputs

Key difference: We can learn different W_l^u for different “neighbor” u for pixel v on the image. The reason is we can pick an order for the 9 neighbors using relative position to the center pixel: $\{(-1, -1), (-1, 0), (-1, 1), \dots, (1, 1)\}$

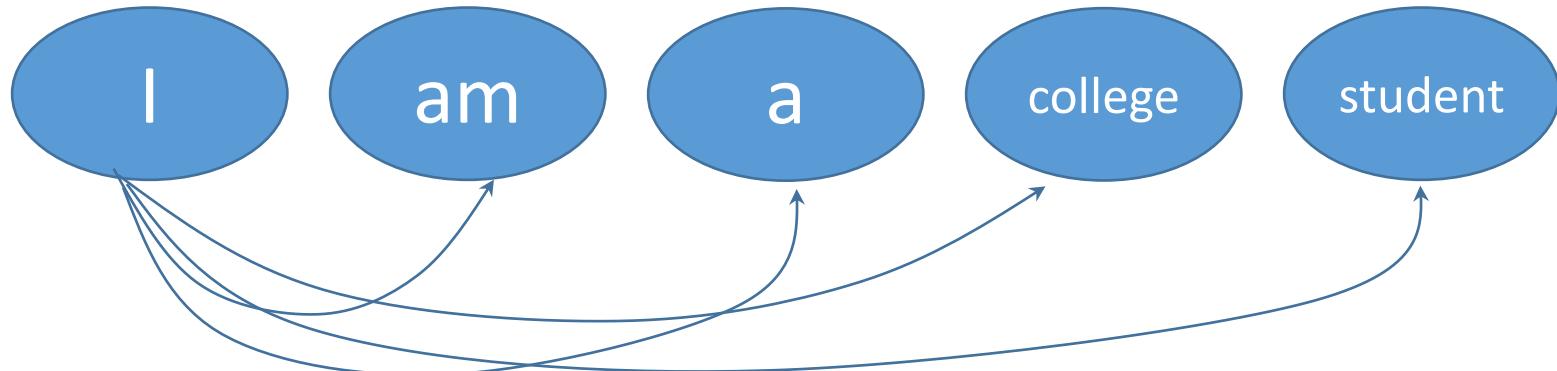
Transformer

- Transformer is one of the most popular architectures that achieves great performance in many sequence modeling tasks



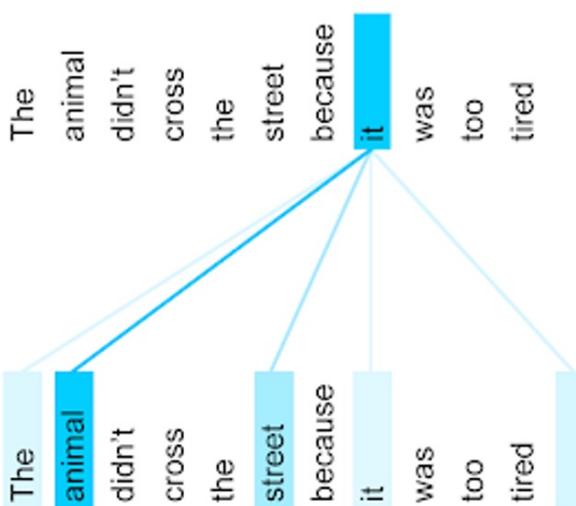
[Attention is all you need. Vaswani et al., NeurIPS 2017]

- Key component: self-attention
 - Every token/word attends to all the other tokens/words via matrix calculation.



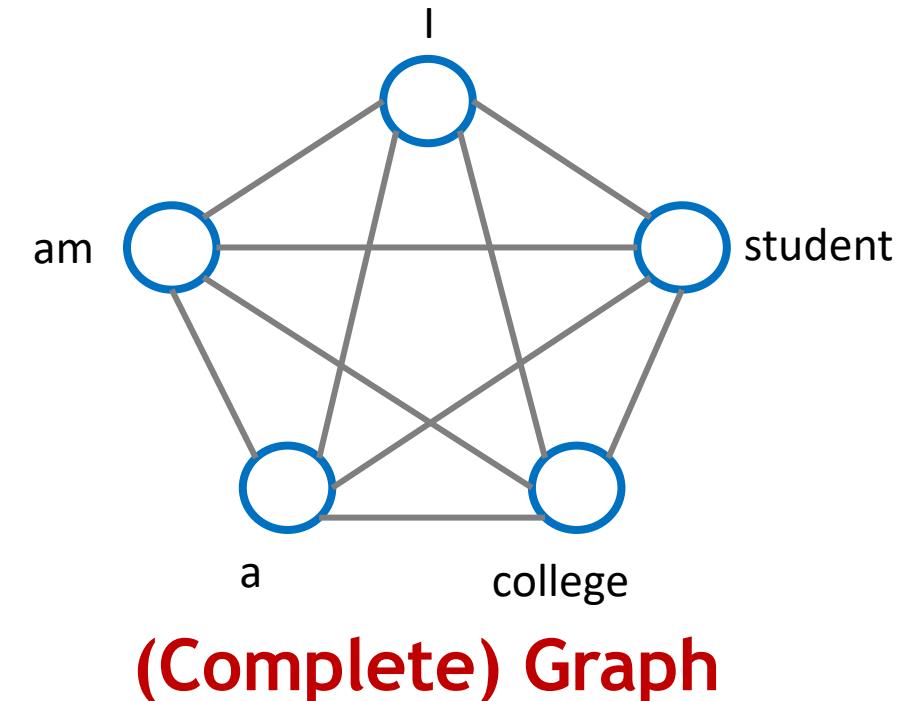
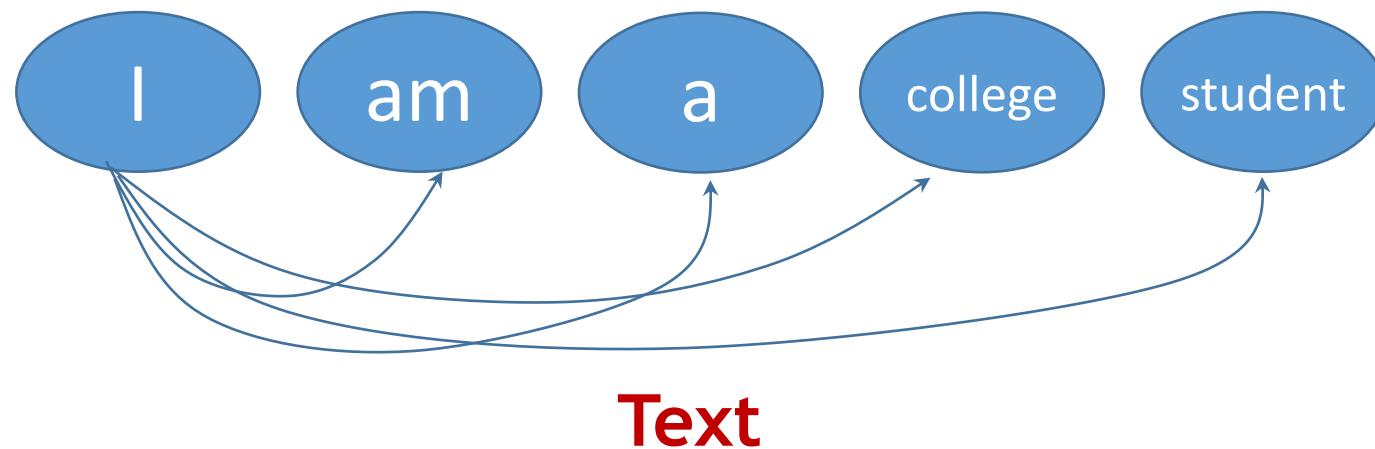
Transformer

- A general definition of attention:
 - Given a set of vector **values**, and a vector **query**, **attention** is a technique to compute a weighted sum of the values, dependent on the query
 - Each token/word has a **value vector** and a **query vector**. The value vector can be seen as the representation of the token/word. We use the query vector to calculate the attention score (weights in the weighted sum)

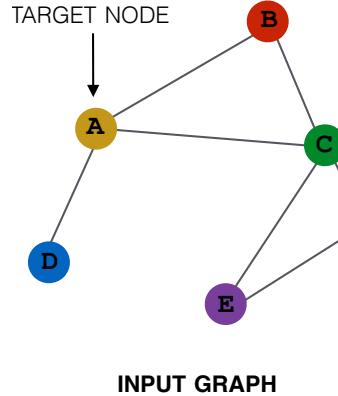


GNN vs. Transformer

- Transformer layer can be seen as a special GNN that runs on a fully-connected “word” graph
 - Since each word attends to all the other words, the computation graph of a transformer layer is identical to that of a GNN on the fully-connected “word” graph

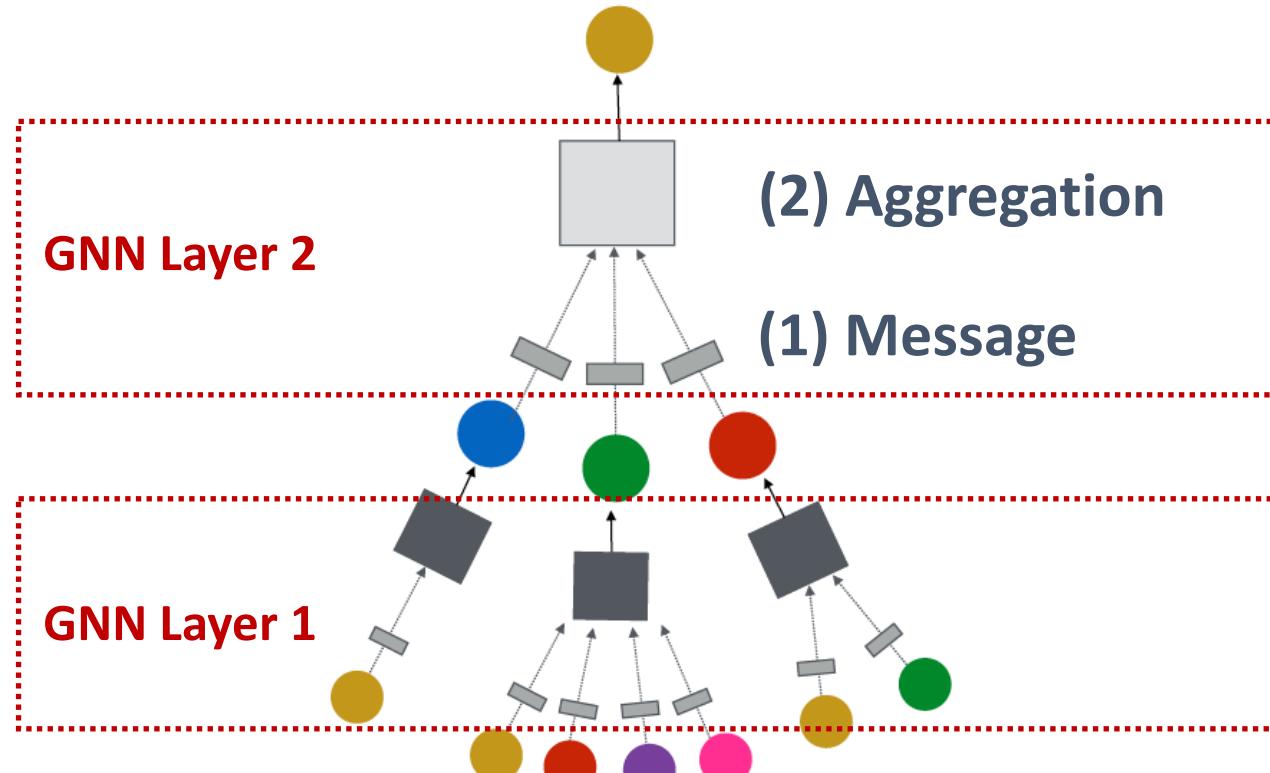


Next: A General GNN Framework



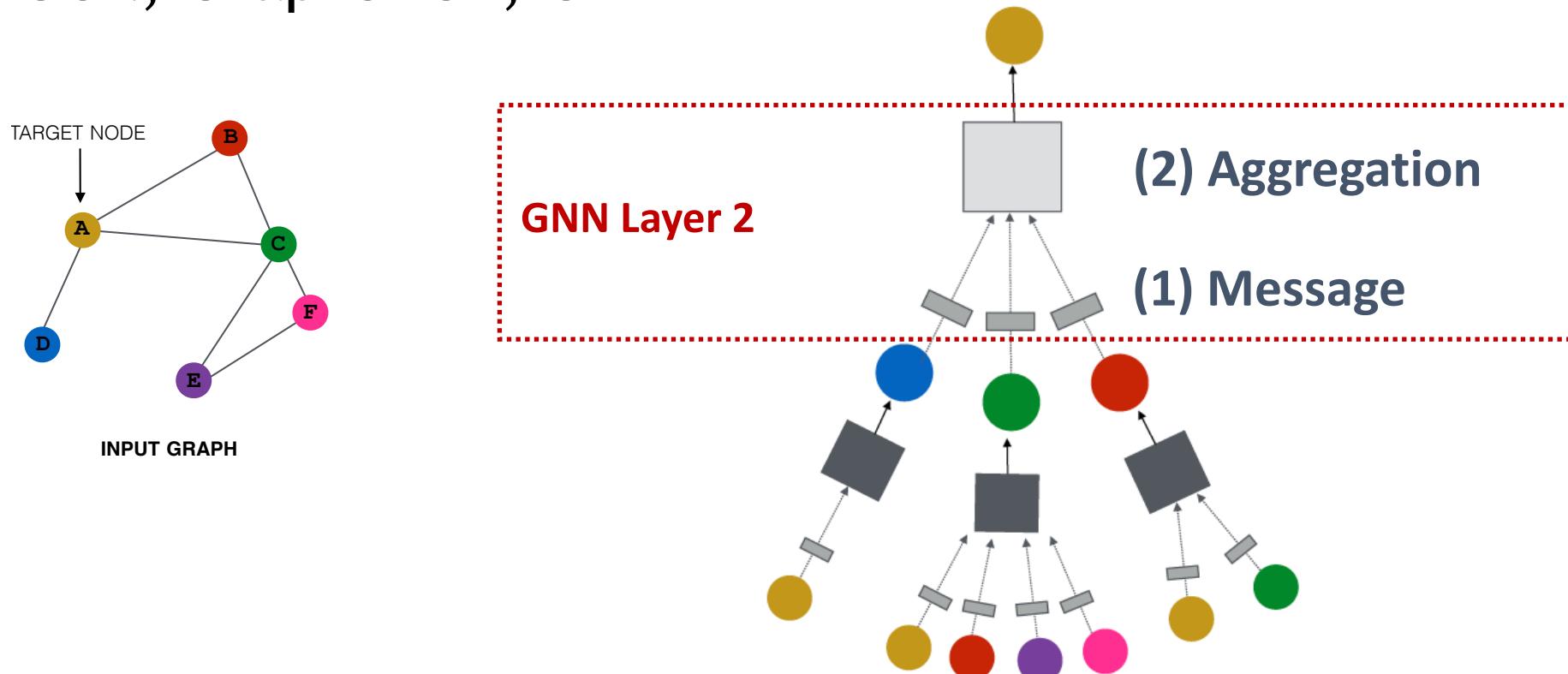
Layer connectivity

Learning objective



A GNN Layer

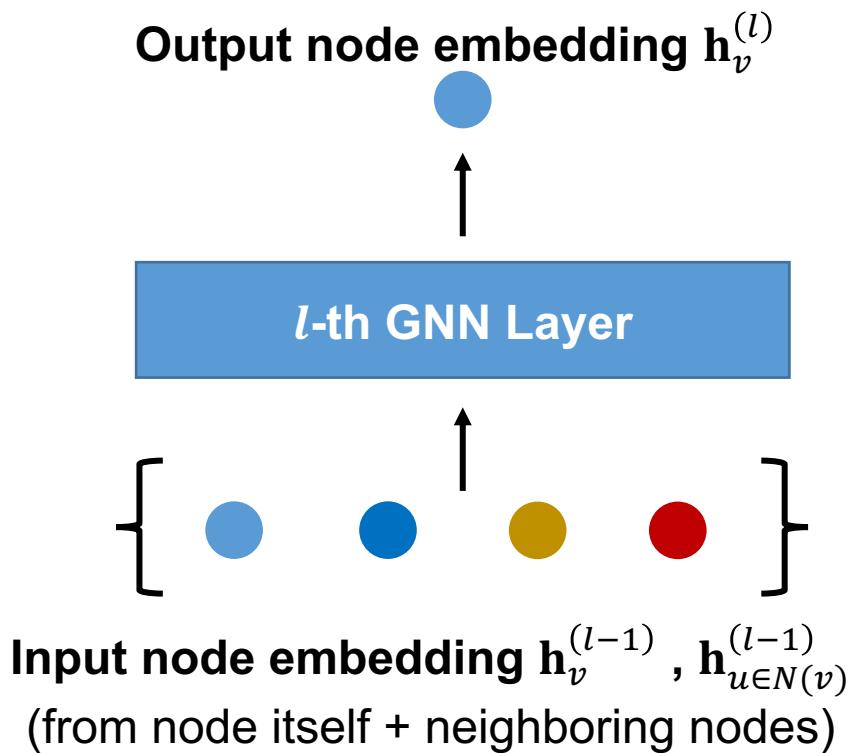
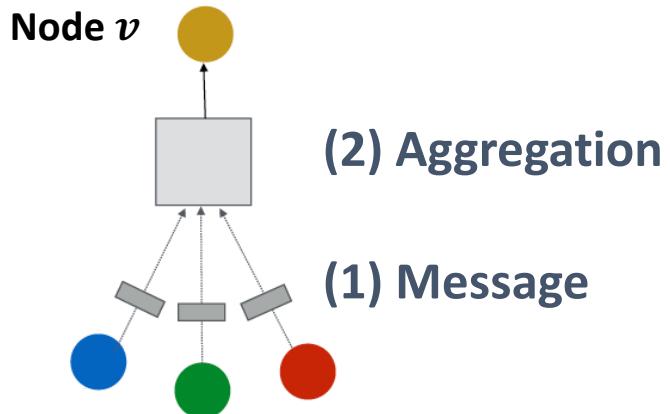
- GNN Layer = Message + Aggregation
 - Different instantiations under this perspective
 - GCN, GraphSAGE, GAT



A Single GNN Layer

- Idea of a GNN Layer:

- Compress a set of vectors into a single vector
- Two-step process:
 - (1) Message
 - (2) Aggregation



Message Computation

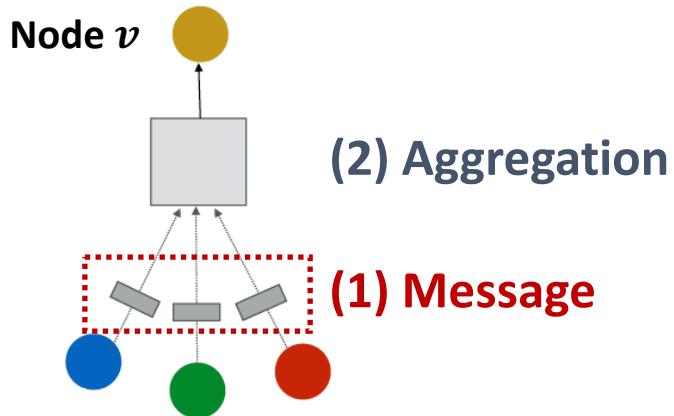
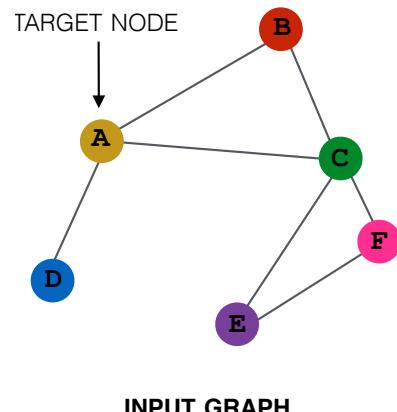
- (1) Message computation

- Message function: $\mathbf{m}_u^{(l)} = \text{MSG}^{(l)}(\mathbf{h}_u^{(l-1)})$

- **Intuition:** each node will create a message, which will be sent to other nodes later

- **Example:** a linear layer $\mathbf{m}_u^{(l)} = \mathbf{W}^{(l)} \mathbf{h}_u^{(l-1)}$

- Multiply node features with weight matrix $\mathbf{W}^{(l)}$



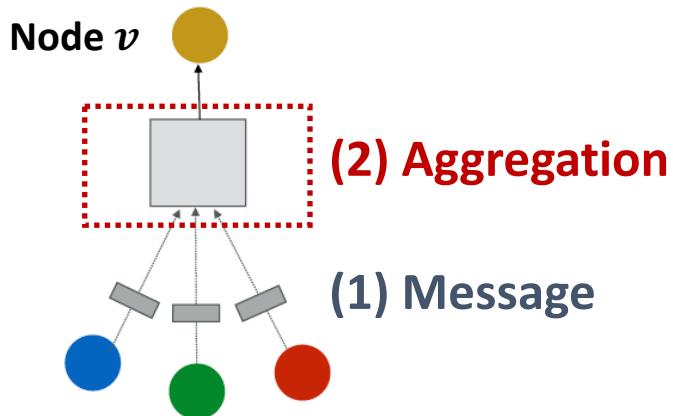
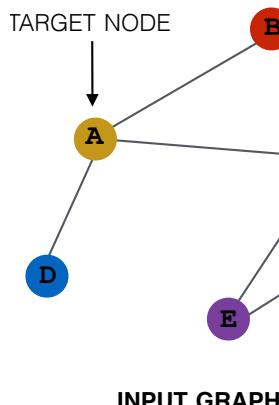
Message Aggregation

- (2) Aggregation

- Intuition: node v will aggregate the messages from its neighbors u :

$$\mathbf{h}_v^{(l)} = \text{AGG}^{(l)} \left(\left\{ \mathbf{m}_u^{(l)}, u \in N(v) \right\} \right)$$

- Example: Sum(\cdot), Mean(\cdot) or Max(\cdot) aggregator
 - $\mathbf{h}_v^{(l)} = \text{Sum}(\{\mathbf{m}_u^{(l)}, u \in N(v)\})$



Message Aggregation: Issue

- **Issue:** Information from node v itself **could get lost**
 - Computation of $h_v^{(l)}$ does not directly depend on $h_v^{(l-1)}$
- **Solution:** Include $h_v^{(l-1)}$ when computing $h_v^{(l)}$
 - (1) **Message:** compute message from node v itself
 - Usually, a different message computation will be performed

$$\bullet \bullet \bullet \quad \mathbf{m}_u^{(l)} = \mathbf{W}^{(l)} \mathbf{h}_u^{(l-1)} \quad \bullet \quad \mathbf{m}_v^{(l)} = \mathbf{B}^{(l)} \mathbf{h}_v^{(l-1)}$$

- (2) **Aggregation:** After aggregating from neighbors, we can aggregate the message from node v itself
 - Via **concatenation** or summation

$$\mathbf{h}_v^{(l)} = \text{CONCAT} \left(\text{AGG} \left(\left\{ \mathbf{m}_u^{(l)}, u \in N(v) \right\} \right), \mathbf{m}_v^{(l)} \right)$$

First aggregate from neighbors

Then aggregate from node itself

A Single GNN Layer

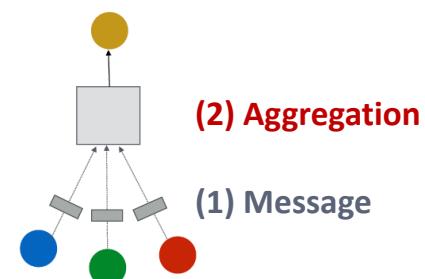
- Putting things together:
 - (1) Message: each node computes a message

$$\mathbf{m}_u^{(l)} = \text{MSG}^{(l)} \left(\mathbf{h}_u^{(l-1)} \right), u \in \{N(v) \cup v\}$$

- (2) Aggregation: aggregate messages from neighbors

$$\mathbf{h}_v^{(l)} = \text{AGG}^{(l)} \left(\left\{ \mathbf{m}_u^{(l)}, u \in N(v) \right\}, \mathbf{m}_v^{(l)} \right)$$

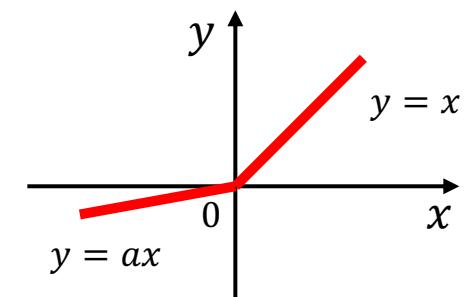
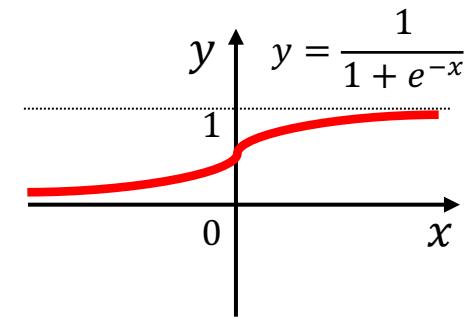
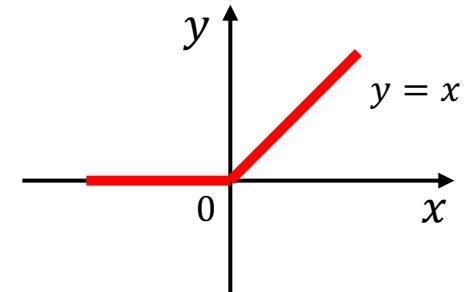
- Nonlinearity (activation): Adds expressiveness
 - Often written as $\sigma(\cdot)$. Examples: ReLU(\cdot), Sigmoid(\cdot), ...
 - Can be added to message or aggregation



ReCap: Activation (Non-Linearity)

■ Applying activation to i -th dimension of embedding x

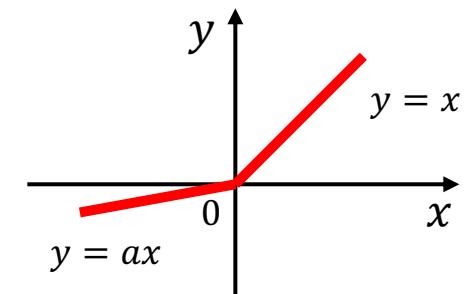
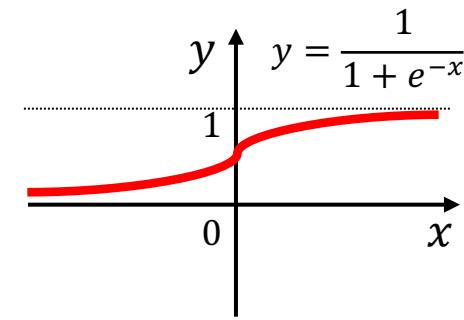
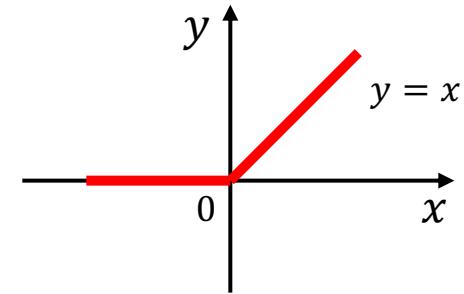
- Rectified linear unit (ReLU)
 - $\text{ReLU}(x_i) = \max(x_i, 0)$
 - Most commonly used
- Sigmoid
 - $\sigma(x_i) = \frac{1}{1+e^{-x_i}}$
 - Used only when you want to restrict the range of your embeddings
- Parametric ReLU
 - $\text{PReLU}(x_i) = \max(x_i, 0) + a_i \min(x_i, 0)$
 - a_i is a trainable parameter
 - Empirically performs better than ReLU



ReCap: Activation (Non-Linearity)

■ Applying activation to i -th dimension of embedding x

- Rectified linear unit (ReLU)
 - $\text{ReLU}(x_i) = \max(x_i, 0)$
 - Most commonly used
- Sigmoid
 - $\sigma(x_i) = \frac{1}{1+e^{-x_i}}$
 - Used only when you want to restrict the range of your embeddings
- Parametric ReLU
 - $\text{PReLU}(x_i) = \max(x_i, 0) + a_i \min(x_i, 0)$
 - a_i is a trainable parameter
 - Empirically performs better than ReLU



Classical GNN Layers: GCN

- Graph Convolutional Networks (GCN)

$$\mathbf{h}_v^{(l)} = \sigma \left(\mathbf{W}^{(l)} \sum_{u \in N(v)} \frac{\mathbf{h}_u^{(l-1)}}{|N(v)|} \right)$$

- How to write this as Message + Aggregation?

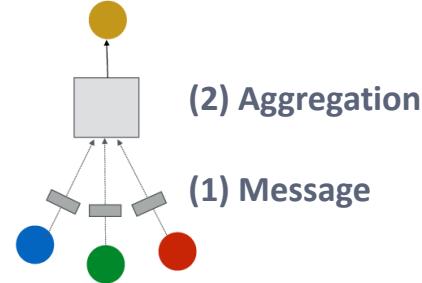
$$\mathbf{h}_v^{(l)} = \sigma \left(\sum_{u \in N(v)} \underbrace{\mathbf{W}^{(l)} \frac{\mathbf{h}_u^{(l-1)}}{|N(v)|}}_{\text{Aggregation}} \underbrace{\text{Message}}_{\text{Message}} \right)$$

The diagram illustrates the two-step process of a Graph Convolutional Network (GCN) layer. It shows a central node (yellow) connected to three other nodes (blue, green, red). Step (1) 'Message' shows dashed arrows pointing from the blue, green, and red nodes to the central yellow node. Step (2) 'Aggregation' shows a solid arrow pointing from the central yellow node upwards, representing the summation of the aggregated messages.

Classical GNN Layers: GCN

■ Graph Convolutional Networks (GCN)

$$\mathbf{h}_v^{(l)} = \sigma \left(\sum_{u \in N(v)} \mathbf{W}^{(l)} \frac{\mathbf{h}_u^{(l-1)}}{|N(v)|} \right)$$



■ Message:

- Each Neighbor: $\mathbf{m}_u^{(l)} = \frac{1}{|N(v)|} \mathbf{W}^{(l)} \mathbf{h}_u^{(l-1)}$

Normalized by node degree
(In the GCN paper they use a slightly different normalization)

■ Aggregation:

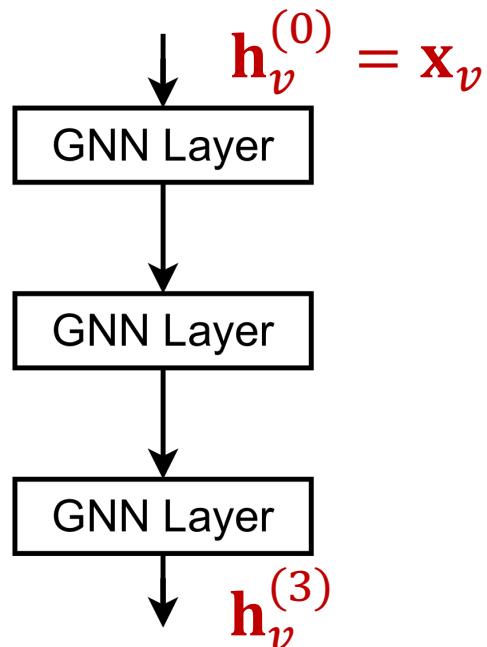
- Sum over messages from neighbors, then apply activation

$$\mathbf{h}_v^{(l)} = \sigma \left(\text{Sum} \left(\{\mathbf{m}_u^{(l)}, u \in N(v)\} \right) \right)$$

In GCN the input graph is assumed to have self-edges that are included in the summation

Stacking GNN Layers

- How to construct a Graph Neural Network?
 - **The standard way:** Stack GNN layers sequentially
 - **Input:** Initial raw node feature \mathbf{x}_v
 - **Output:** Node embeddings $\mathbf{h}_v^{(L)}$ after L GNN layers



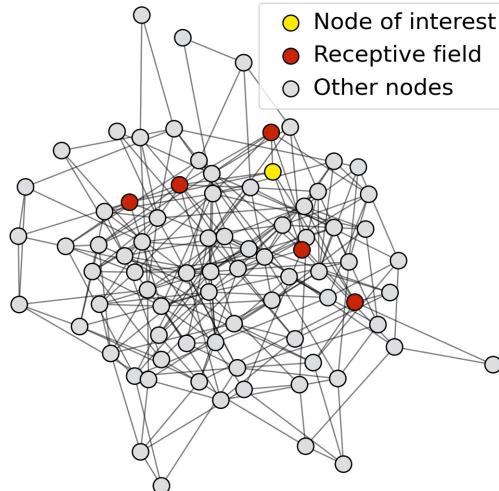
The Over-Smoothing Problem

- The issue of stacking many GNN layers
 - GNN suffers from **the over-smoothing problem**
- **The over-smoothing problem:** all the node embeddings converge to the same value
 - This is bad because we **want to use node embeddings to differentiate nodes**
- Why does the over-smoothing problem happen?

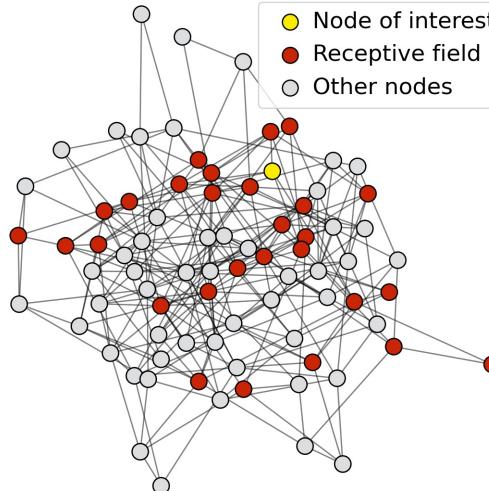
Receptive Field of A GNN

- **Receptive field:** the set of nodes that determine the embedding of a node of interest
 - In a **K-layer GNN**, each node has a **receptive field of K-hop neighborhood**

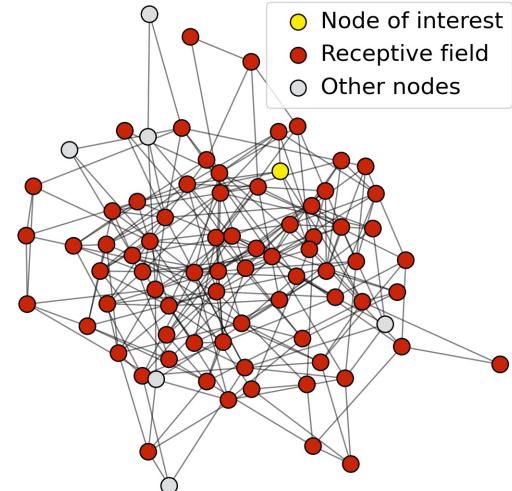
Receptive field for
1-layer GNN



Receptive field for
2-layer GNN



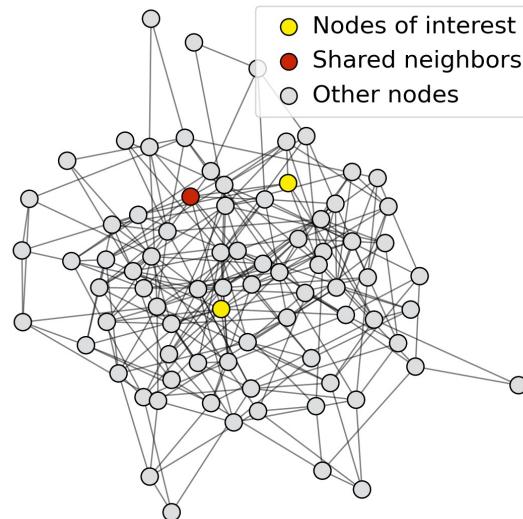
Receptive field for
3-layer GNN



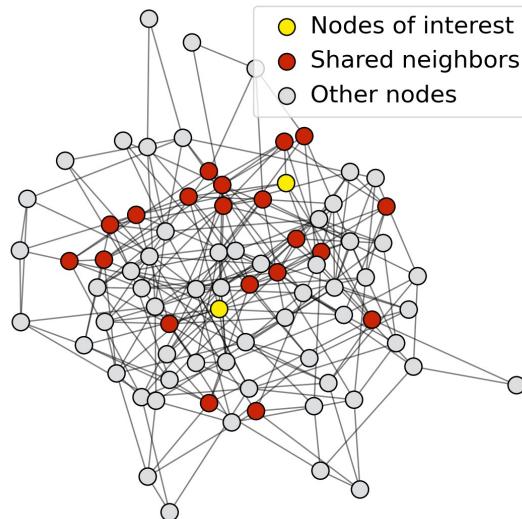
Receptive Field of A GNN

- Receptive field overlap for two nodes
 - The shared neighbors quickly grows when we increase the number of hops (number of GNN layers)

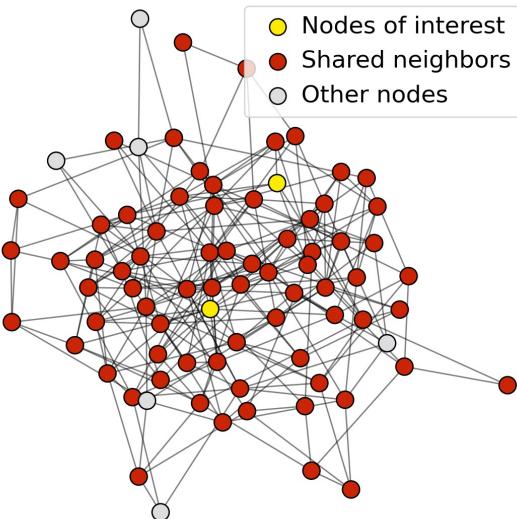
1-hop neighbor overlap
Only 1 node



2-hop neighbor overlap
About 20 nodes



3-hop neighbor overlap
Almost all the nodes!



Receptive Field and Over-Smoothing

- We can explain over-smoothing via the notion of the receptive field
 - We know **the embedding of a node is determined by its receptive field**
 - If two nodes have highly-overlapped receptive fields, then their embeddings are highly similar
 - Stack many GNN layers nodes → will have highly-overlapped receptive fields → node embeddings will be highly similar → suffer from the over-smoothing problem
 - **Next:** how do we overcome over-smoothing problem?

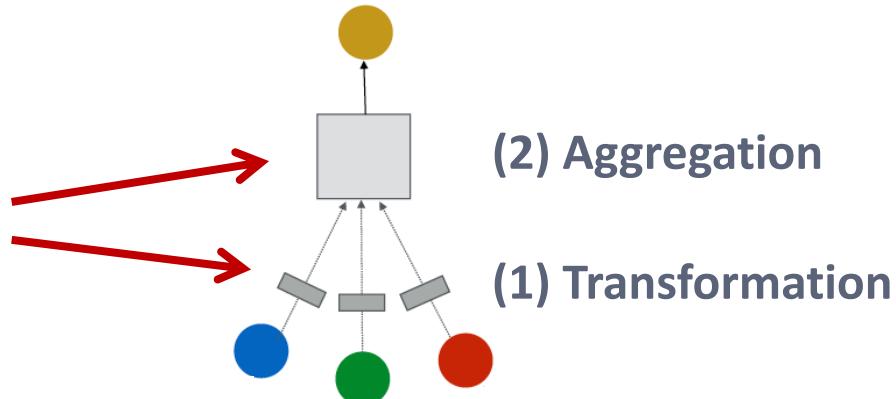
Design GNN Layer Connectivity

- Be cautious when adding GNN layers
 - Unlike neural networks in other domains (CNN for image classification), **adding more GNN layers do not always help**
 - **Step 1:** Analyze the necessary receptive field to solve your problem. E.g., by computing the diameter of the graph
 - **Step 2:** Set number of GNN layers L to be a bit more than the receptive field we like. Do not set L to be unnecessarily large
- Question: How to enhance the expressive power of a GNN, if the number of GNN layers is small?

Expressive Power for Shallow GNNs

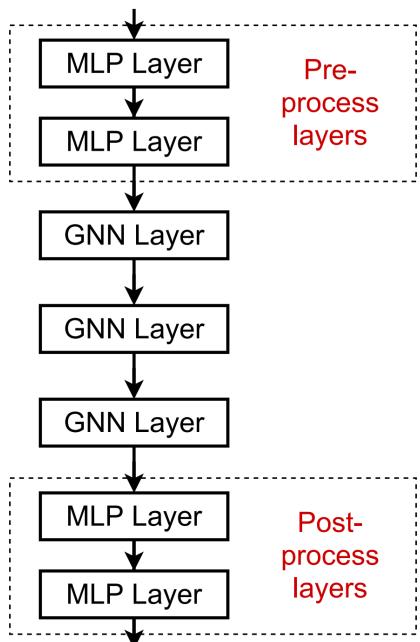
- How to make a shallow GNN more expressive?
- Solution 1: Increase the expressive power **within each GNN layer**
 - In our previous examples, each transformation or aggregation function only include one linear layer
 - We can make aggregation / transformation become a deep neural network

If needed, each box could include a **3-layer MLP**



Expressive Power for Shallow GNNs

- How to make a shallow GNN more expressive?
- **Solution 2:** Add layers that do not pass messages
 - A GNN does not necessarily only contain GNN layers
 - E.g., we can add **MLP layers** (applied to each node) before and after GNN layers, as **pre-process layers** and **post-process layers**



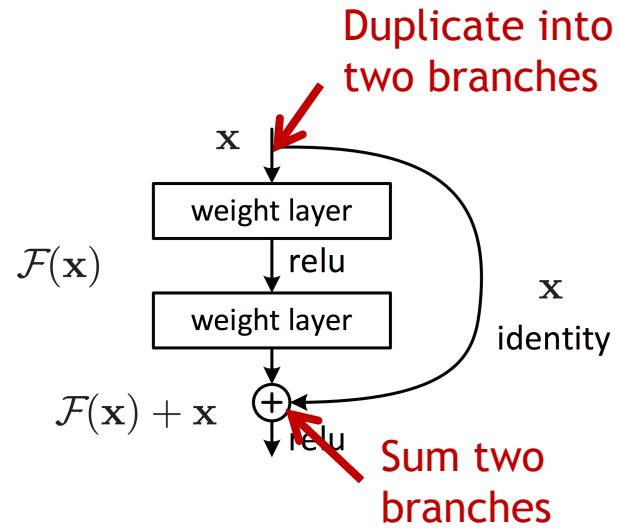
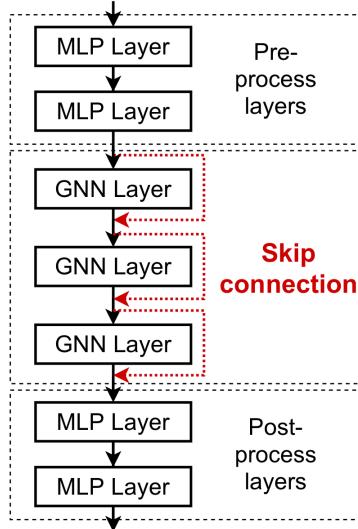
Pre-processing layers: Important when encoding node features is necessary.
E.g., when nodes represent images/text

Post-processing layers: Important when reasoning / transformation over node embeddings are needed
E.g., graph classification, knowledge graphs

In practice, adding these layers works great

Design GNN Layer Connectivity

- What if my problem still requires many GNN layers?
- Add skip connections in GNNs
 - Observation from over-smoothing: Node embeddings in earlier GNN layers can sometimes better differentiate nodes
 - Solution: We can increase the impact of earlier layers on the final node embeddings, by adding shortcuts in GNN



Idea of skip connections:
Before adding shortcuts:
 $\mathcal{F}(x)$
After adding shortcuts:
 $\mathcal{F}(x) + x$

Example: GCN with Skip Connections

- A standard GCN layer

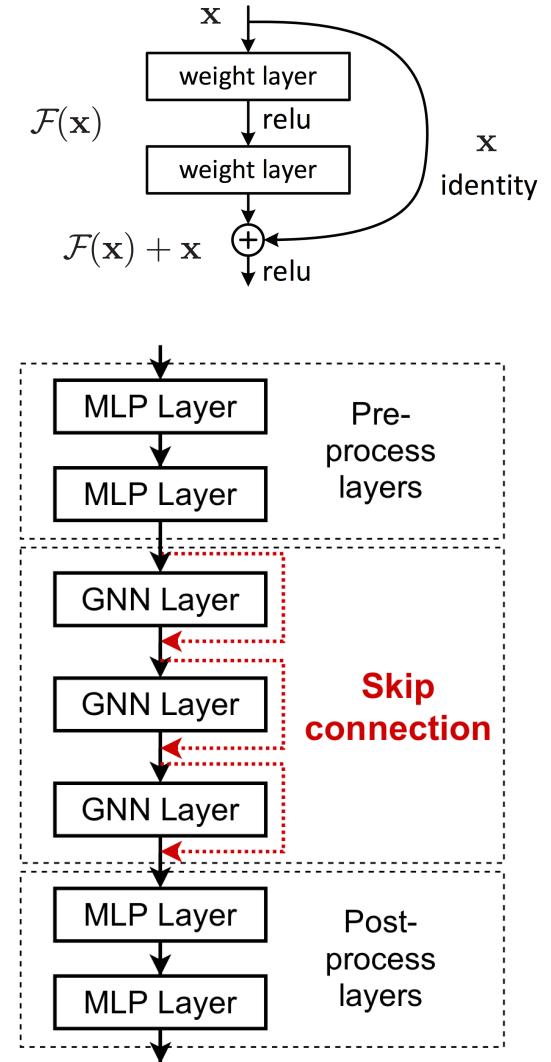
This is our $F(\mathbf{x})$

$$\mathbf{h}_v^{(l)} = \sigma \left(\sum_{u \in N(v)} \mathbf{W}^{(l)} \frac{\mathbf{h}_u^{(l-1)}}{|N(v)|} \right)$$

- A GCN layer with skip connection

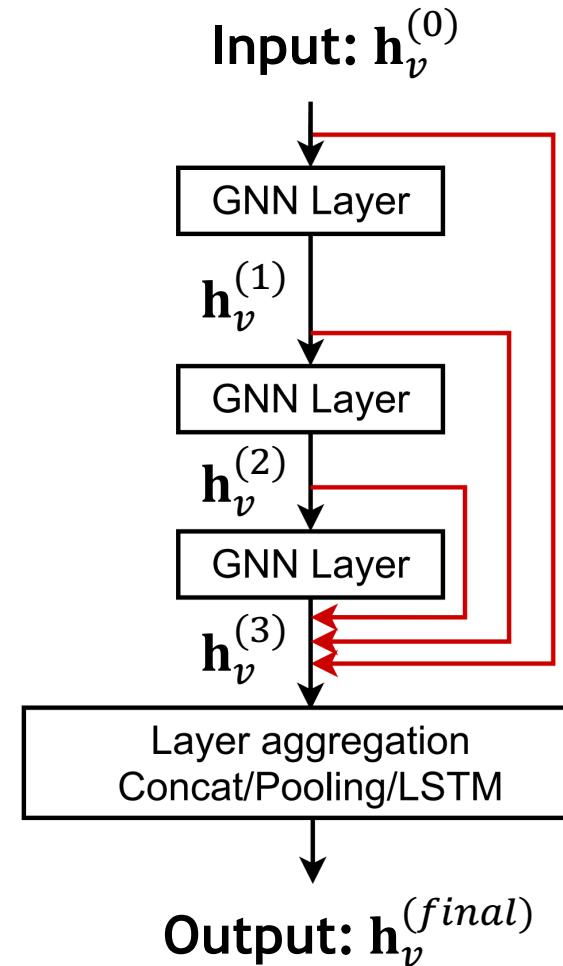
$$\mathbf{h}_v^{(l)} = \sigma \left(\sum_{u \in N(v)} \mathbf{W}^{(l)} \frac{\mathbf{h}_u^{(l-1)}}{|N(v)|} + \mathbf{h}_v^{(l-1)} \right)$$

$\mathbf{F}(\mathbf{x}) \quad + \quad \mathbf{x}$



Other Options of Skip Connections

- Other options: Directly skip to the last layer
 - The final layer directly aggregates from the all the node embeddings in the previous layers



Summary

- Different types of graphs (homogeneous, heterogeneous, directed, etc.), graph analysis tasks (node classification, node clustering, link prediction, etc.), mathematical representation of graphs (adjacency matrix, feature matrix)
- **Node embedding:** “shallow” embedding is an embedding-lookup, the goal is to preserve the similarity of nodes in the embedding space
 - Random walk similarity
 - Limitations: transductive, many parameters, cannot incorporate node features, cannot encode structural similarity
- **Graph Neural Networks:** permutation invariance and equivariance, deep encoder of a GCN, training of GNNs, relationship with CNNs and Transformer, message computation and aggregation, design of deep GNNs

Thank you!