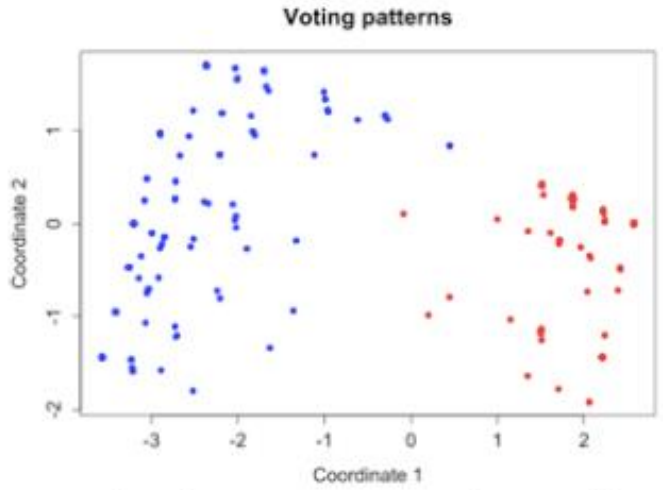
t-SNE

Outline

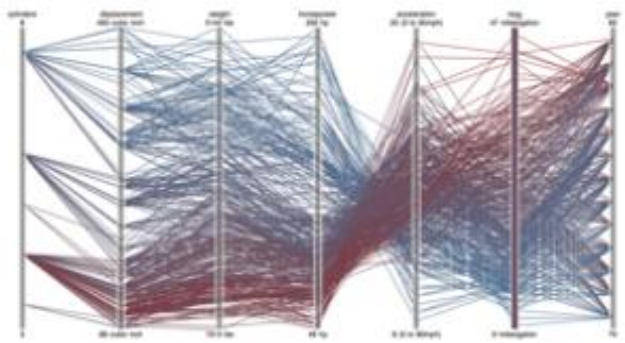
- Review Dimensionality Reduction
- Stochastic Neighbor Embedding (SNE)
  - t-Distributed Stochastic Neighbor Embedding (t-SNE)
  - Computational Complexity
- Streaming t-SNE
- Simulation Results

Visualizing many dimensions

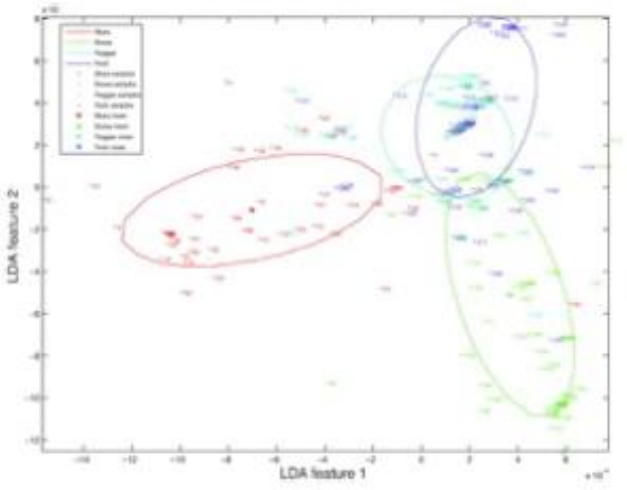
- Visualizing data of high dimension is hard!



Multidimensional Scaling



Parallel Coordinates

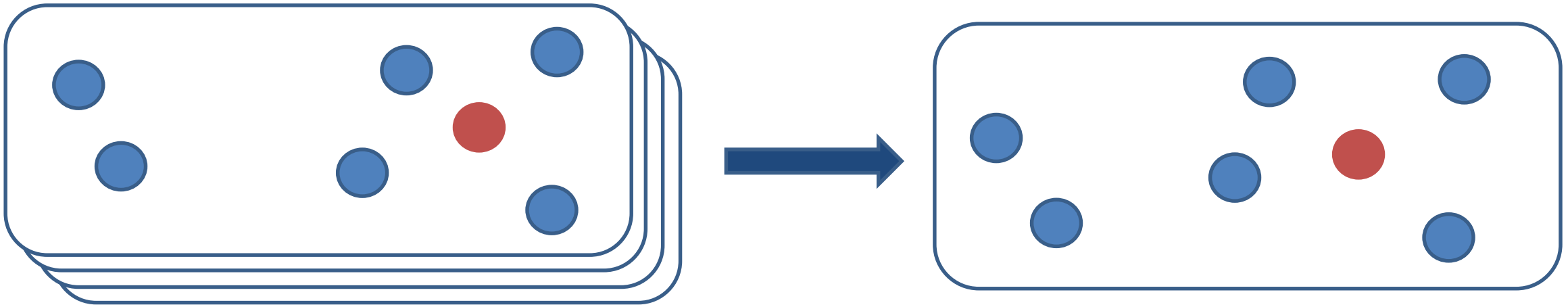


Principal Component Analysis (PCA)

Introduction

- We are given a collection of N high-dimensional objects $x_1, \ldots, x_n$
- How can we visualize these objects in a low dimensional space?

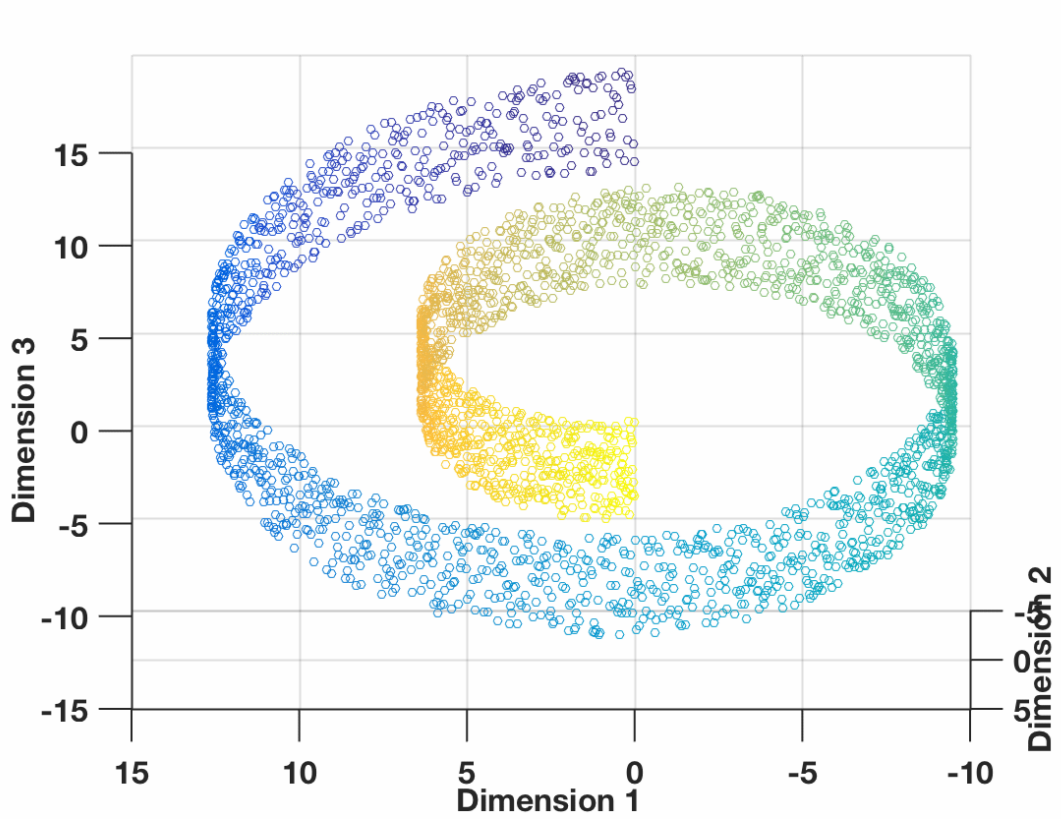
- Method
  - Build a 2-D or 3-D map, such that distances between points in the low-D map reflect similarities in the high-dimensional data.



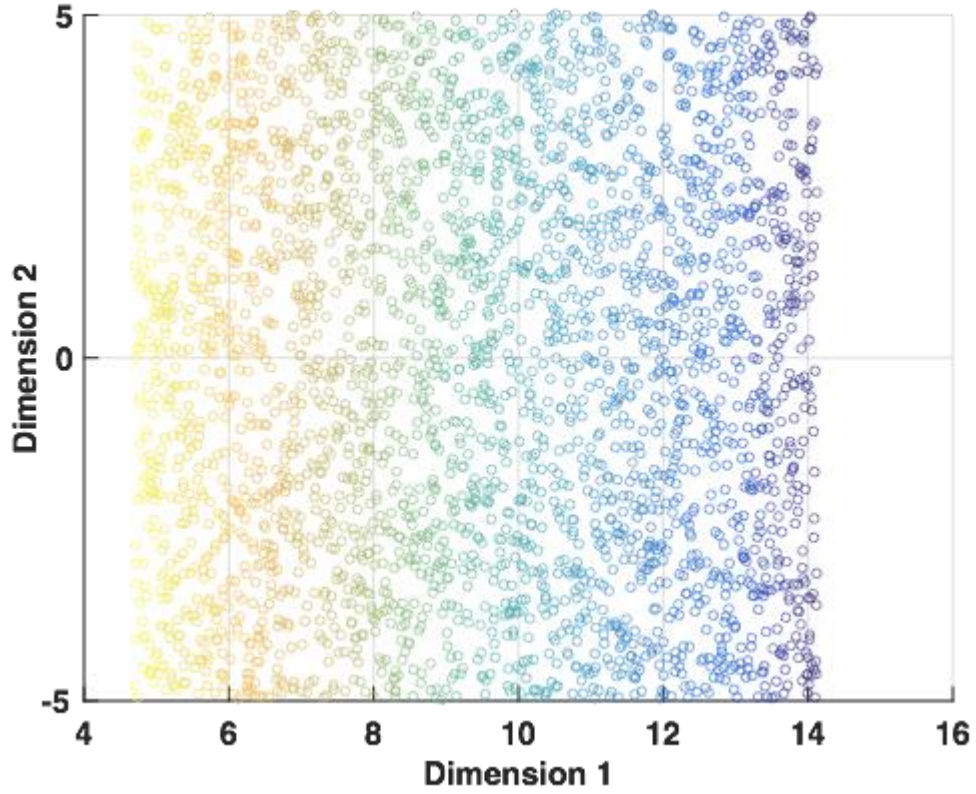
We need to minimize an objective function that measures the discrepancy between similarities in the data and similarities in the map.

Example of Dimensionality Reduction

High Order Datapoints

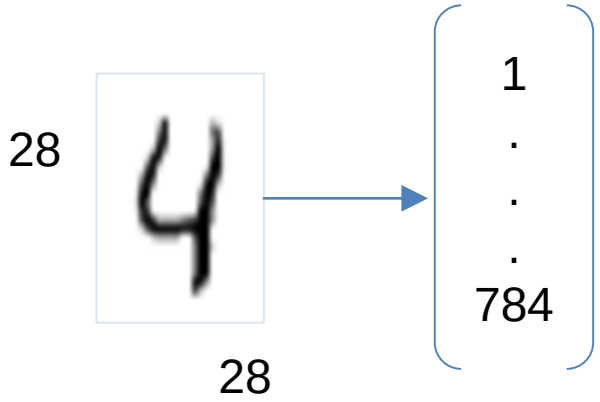


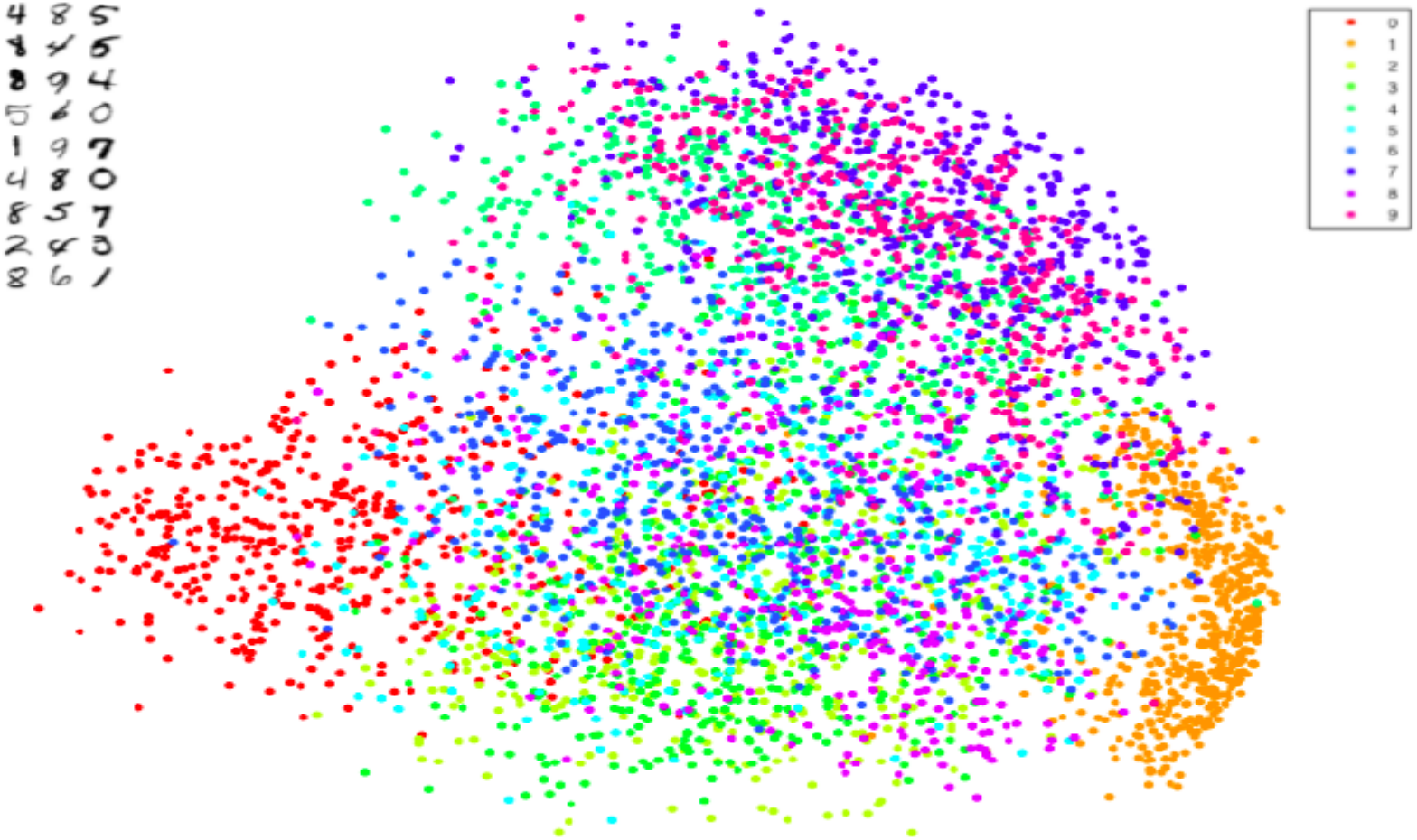
Desired Low Dimensional Embedding



What about PCA?

- (MNIST) Dataset of handwritten digits
- Each datapoint is a 28x28 grayscale image
- Can be viewed as a 784-dimensional point



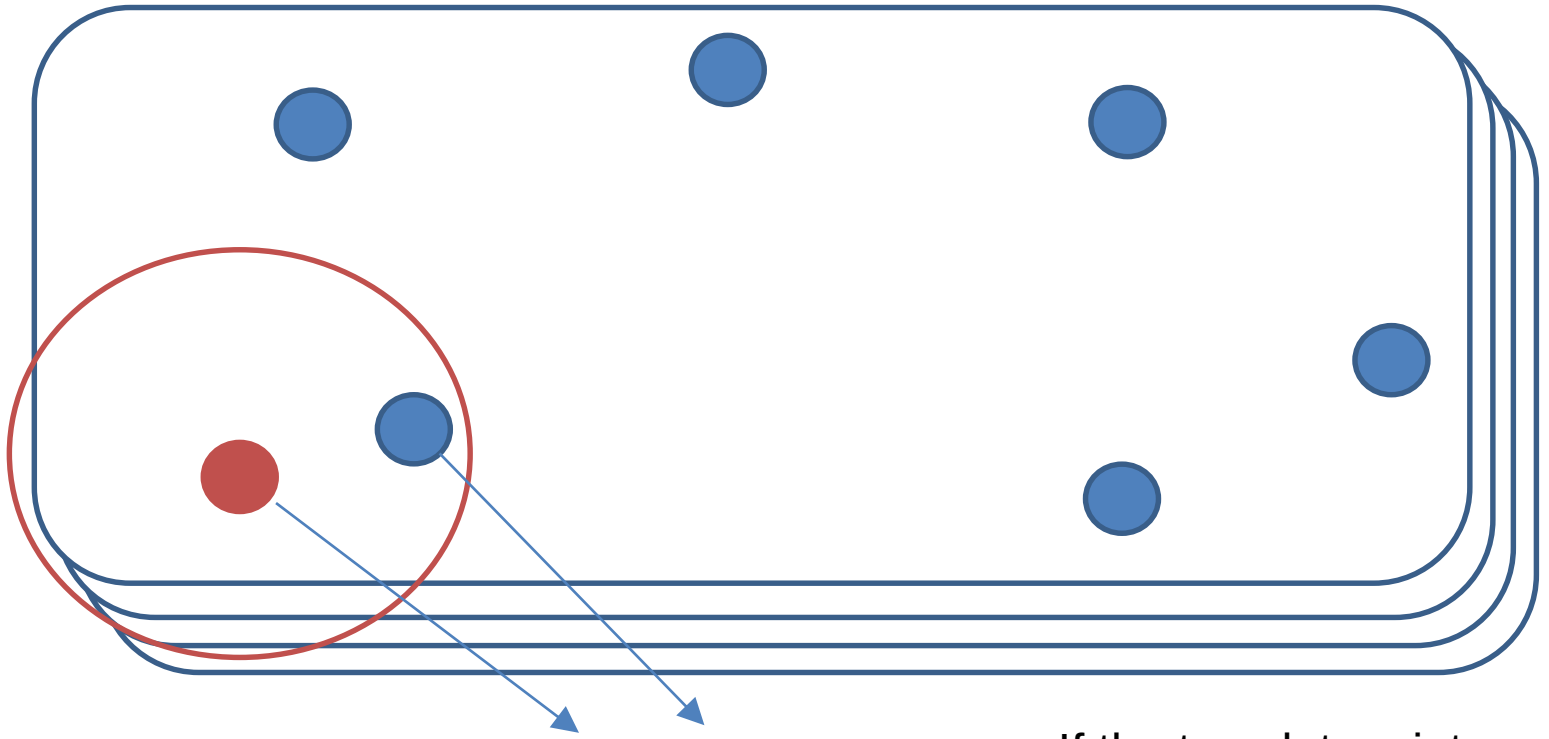


What distances to preserve?

- PCA is concerned with preserving large pairwise distances in the map.
- But are such distances very informative?
  - Points which are far apart in high-dimensional space would also appear far apart in low-dimensional subspace.
  - However, distances between similar points is not necessarily preserved.
- Our goal is to generate a low-dimensional embedded such that points that are nearest neighbor in high-dimensional map, are also neighbors in low dimensional.

Stochastic Neighbor Embedding

- Measure pairwise similarities between high-dimensional objects.

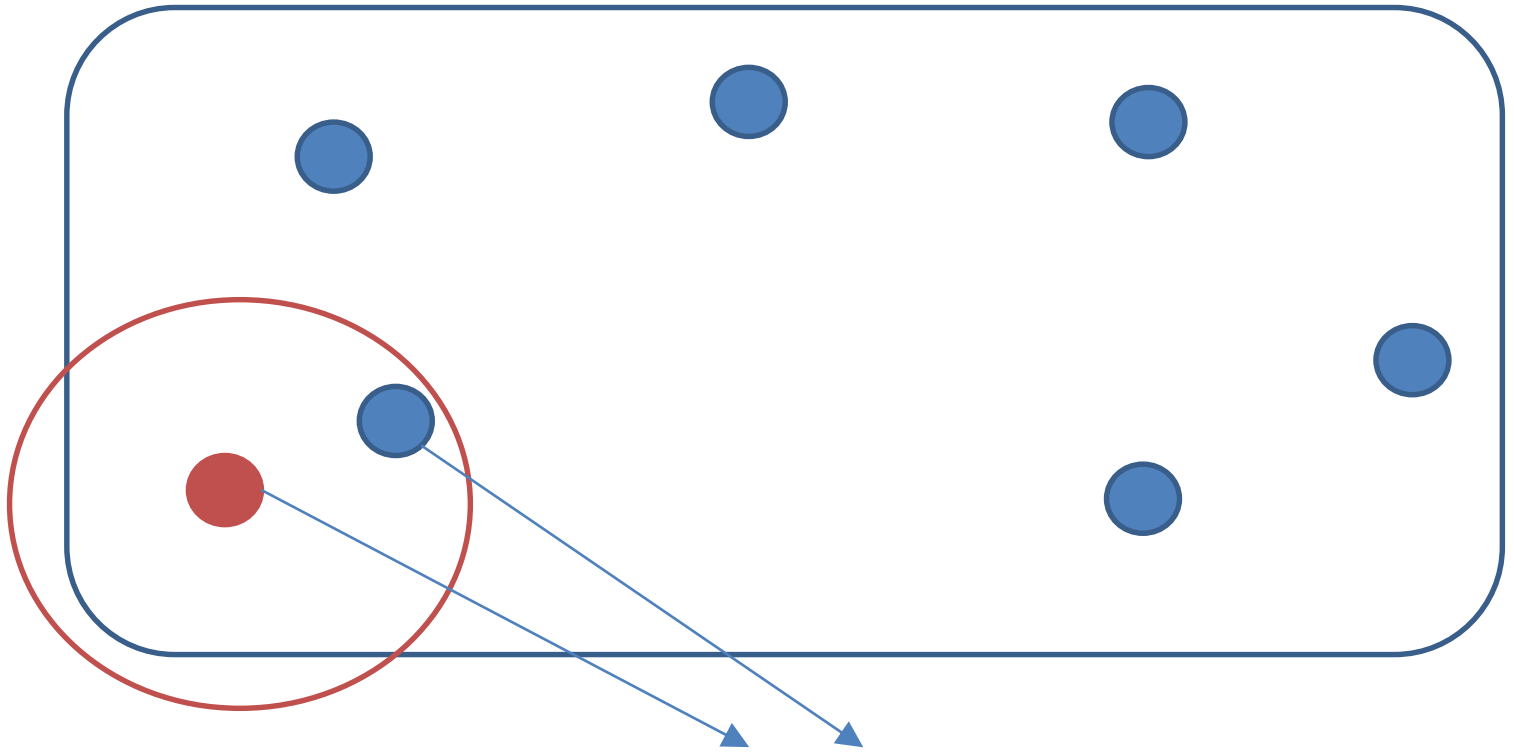


$$p_{j|i} = \frac{\exp(-\|x_i - xj\|^2/2\sigma^2)}{\sum_k \sum_{l \neq k} \exp(-\|x_k - xl\|^2/2\sigma^2)}$$

If the two datapoints x_i and x_j are relatively close to one another, the similarity should be high.

Stochastic Neighbor Embedding

- Measure pairwise similarities between low-dimensional map points.



$$q_{j|i} = \frac{(1 + \|y_i - yj\|^2)^{-1}}{\sum_k \sum_{l \neq k} (1 + \|y_k - yi\|^2)^{-1}}$$

Stochastic Neighbor Embedding (SNE)

- Converting the high-dimensional Euclidean distances into conditional probabilities that represent similarities.
- Similarity of data-points in High Dimension $p_{j|i} = \frac{\exp(-\|x_i - xj\|^2/2\sigma^2)}{\sum_k \sum_{l \neq k} \exp(-\|x_k - xl\|^2/2\sigma^2)}$
- Similarity of data-points in Low Dimension $q_{j|i} = \frac{(1 + \|y_i - yj\|^2)^{-1}}{\sum_k \sum_{l \neq k} (1 + \|y_k - yi\|^2)^{-1}}$
- Cost function $C = \sum_i KL(Pi||Q_i) = \sum_i \sum_{j \neq i} pij \log \frac{p_{ij}}{q_{ij}}$

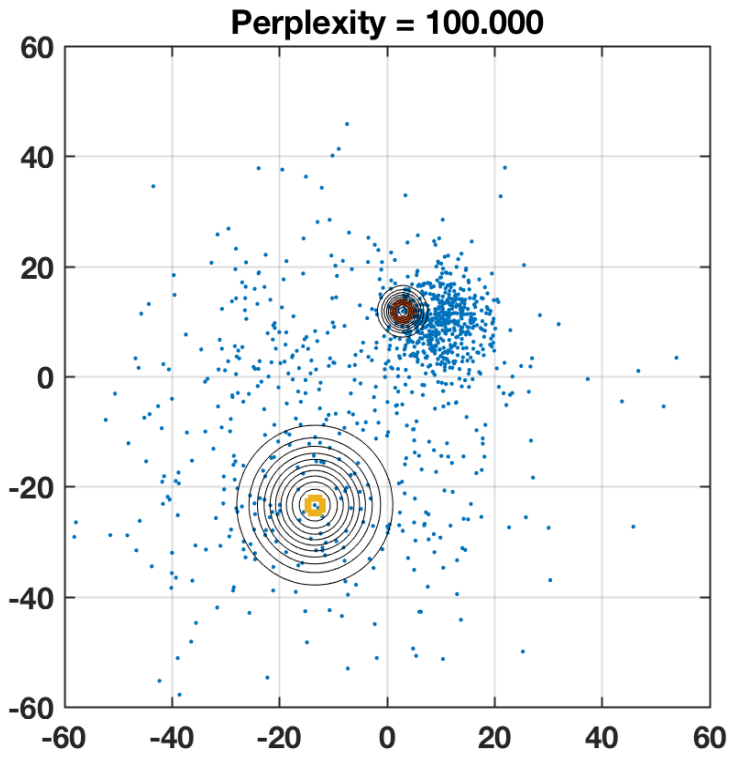
Minimize the cost function using gradient descent

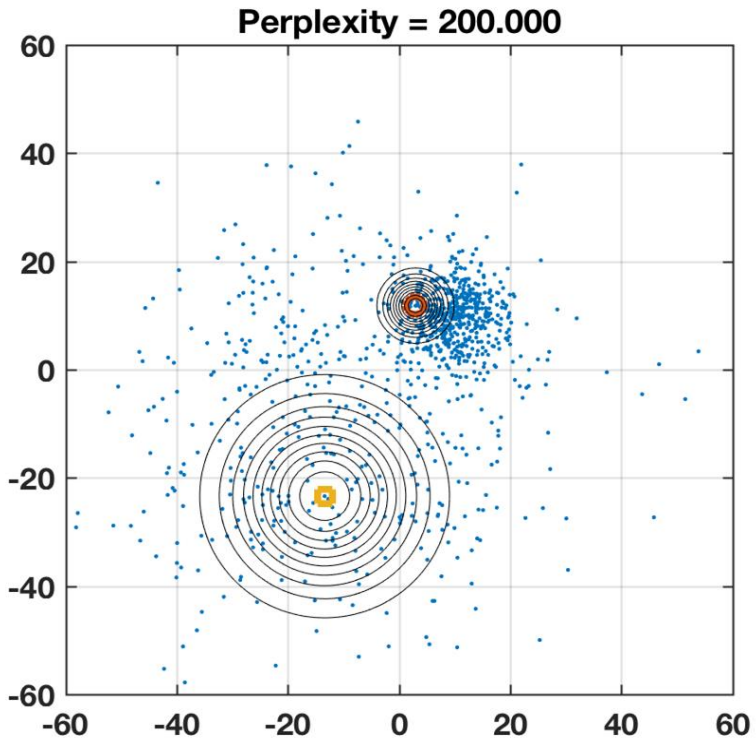
Stochastic Neighbor Embedding by Hinton & Roweis (2012)

How to choose sigma?

- Set the bandwidth such that the conditional has a fixed perplexity.

$$p_{j|i} = \frac{\exp\left(-\|x_i - x_j\|^2 / 2\sigma^2\right)}{\sum_k \sum_{k \neq i} \exp\left(-\|x_i - x_j\|^2 / 2\sigma^2\right)}$$





Issues with SNE

- SNE suffers from the crowding problem
  - Suppose data is very high – dimensional
  - If we try to model the local structure of this data in the map
  - The result is that dissimilar points have to be modeled too far part in the map

- Hence, most visualizations will display a group of points clustered in the middle.

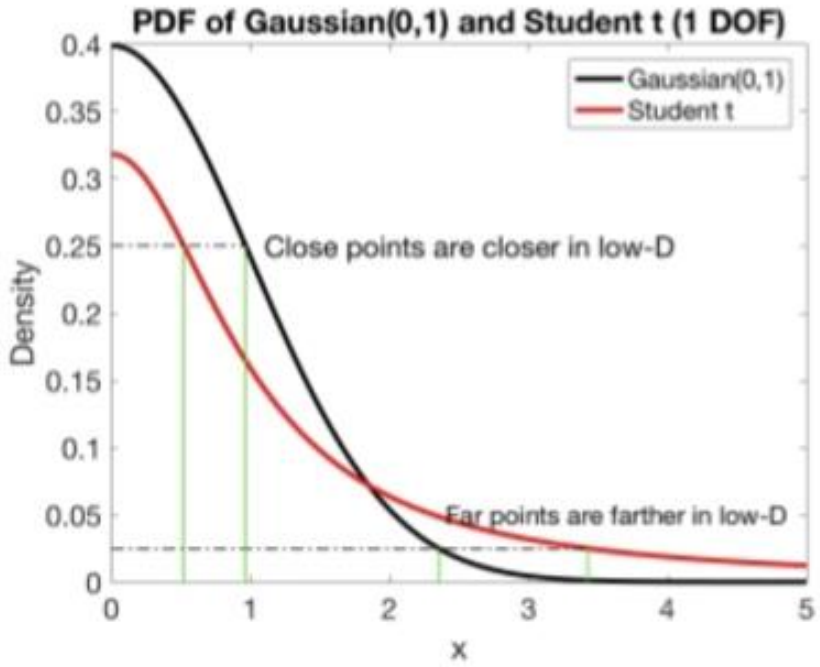
t-Distributed Stochastic Neighbor Embedding

- Use a heavy-tailed distribution to measure similarities in the map:

$$q_{j|i} = \frac{\exp\left(-\|y_i - y_j\|^2\right)}{\sum_k \sum_{k \neq i} \exp\left(-\|y_i - y_j\|^2\right)}$$

- As a result, dissimilar objects are allowed to be modeled too far apart.

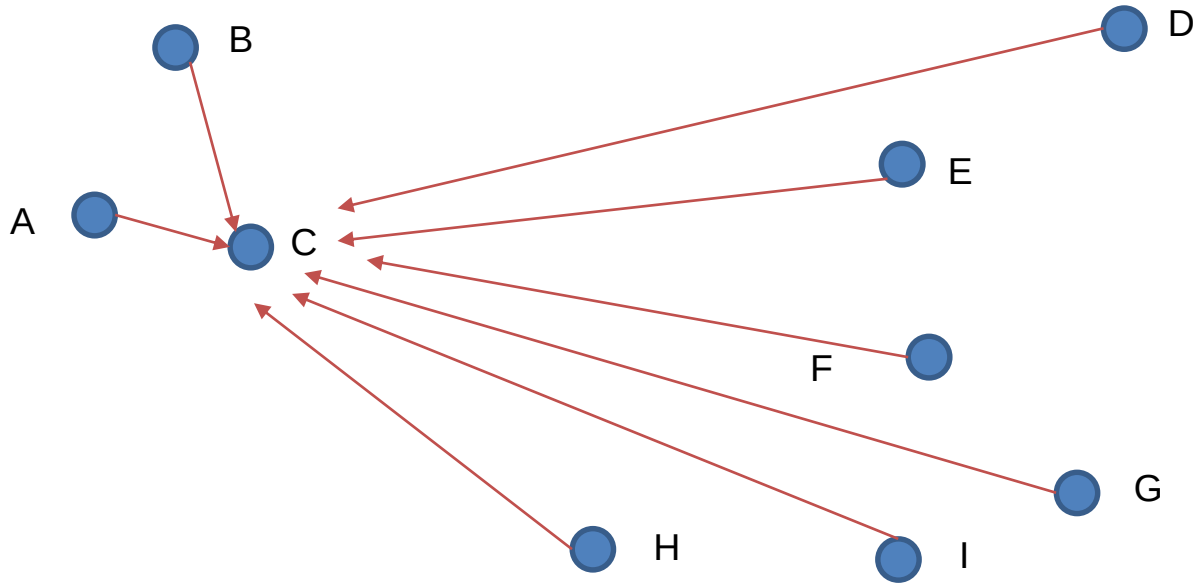
- This effectively eliminates the crowding problem.



Gradient interpretation

- We can interpret the t-SNE gradient as a simulation of an N-body system:

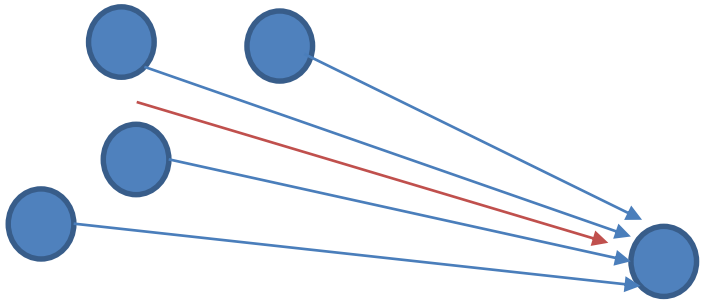
$$\frac{\partial C}{\partial y_i} = 4 \sum_{j \neq i} \frac{p_{ij} - qij}{1 + \|y_i - yj\|^2} (yi - yj)$$



Barnes-Hut t-SNE

- Original t-SNE data and computational complexity is $O(N^2)$
  - Limits visualization to thousands of points

- Observation : Many of the pairwise interactions between points are very similar
  - Idea: Approximate similar interactions by a single interaction
  - If a collection of k points are sufficiently away from x_i
  - Then approximate the force excreted by those points as f * k

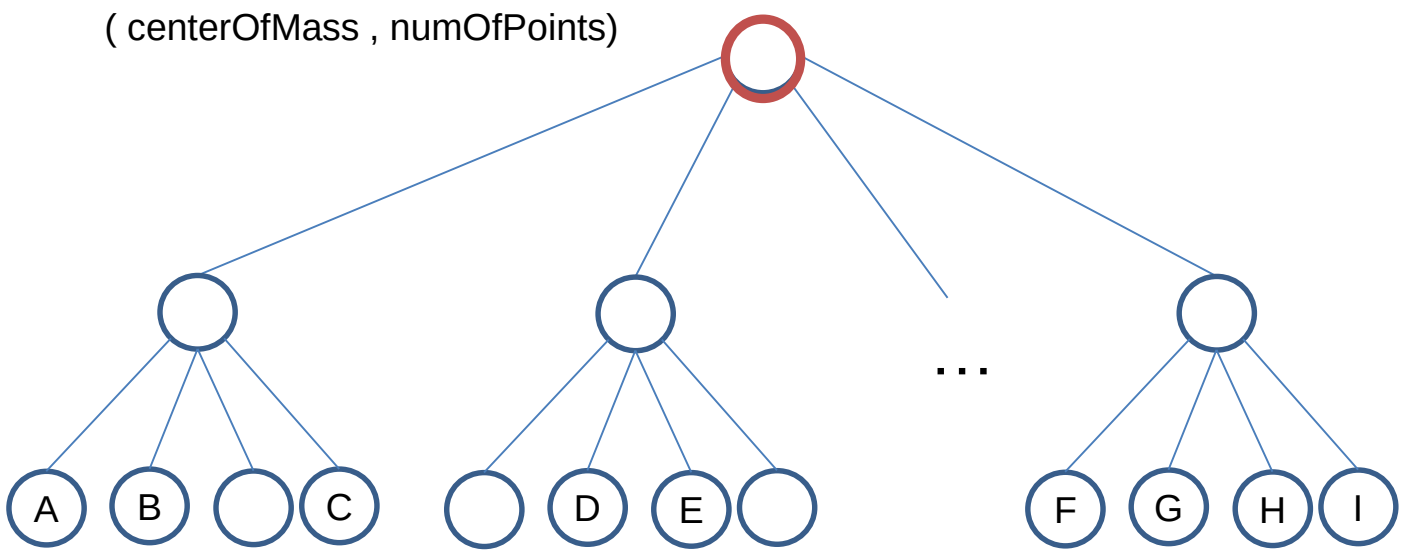


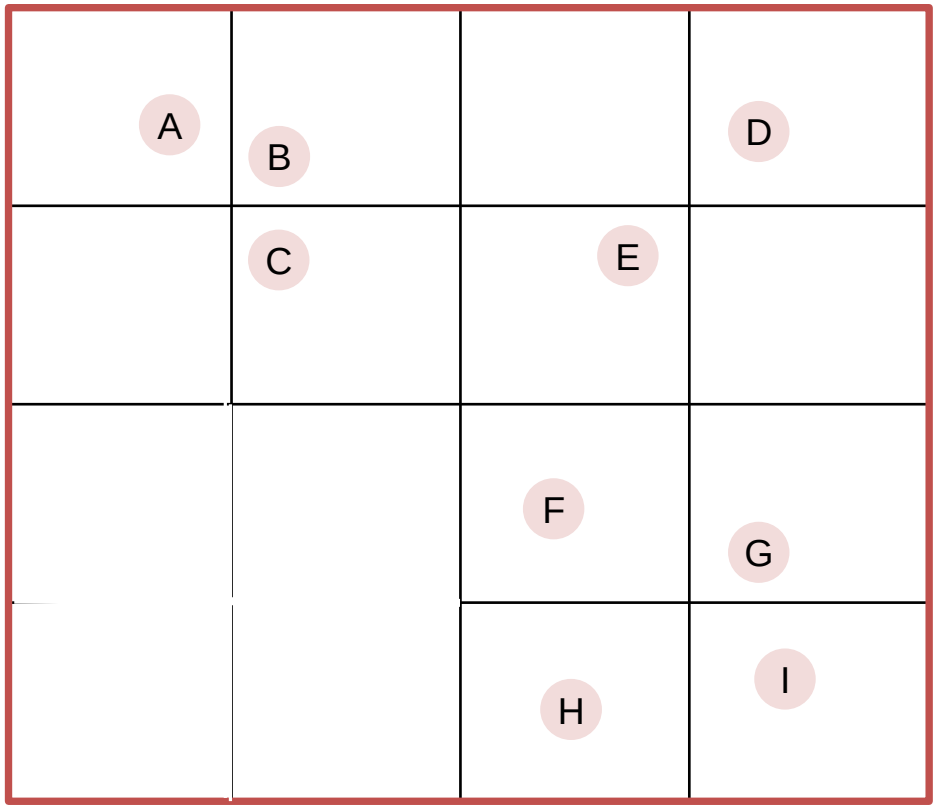
- Reduce complexity to O (N * log (N)) via Barnes-Hut t-SNE
  - Allows t-SNE to scale to millions of data points
  - Utilizes a tree - based algorithm (vantage-point trees) to perform the KNN search

Accelerating t-SNE using Tree-Based Algorithms by Maaten and Hinton (2014)

Quadtree

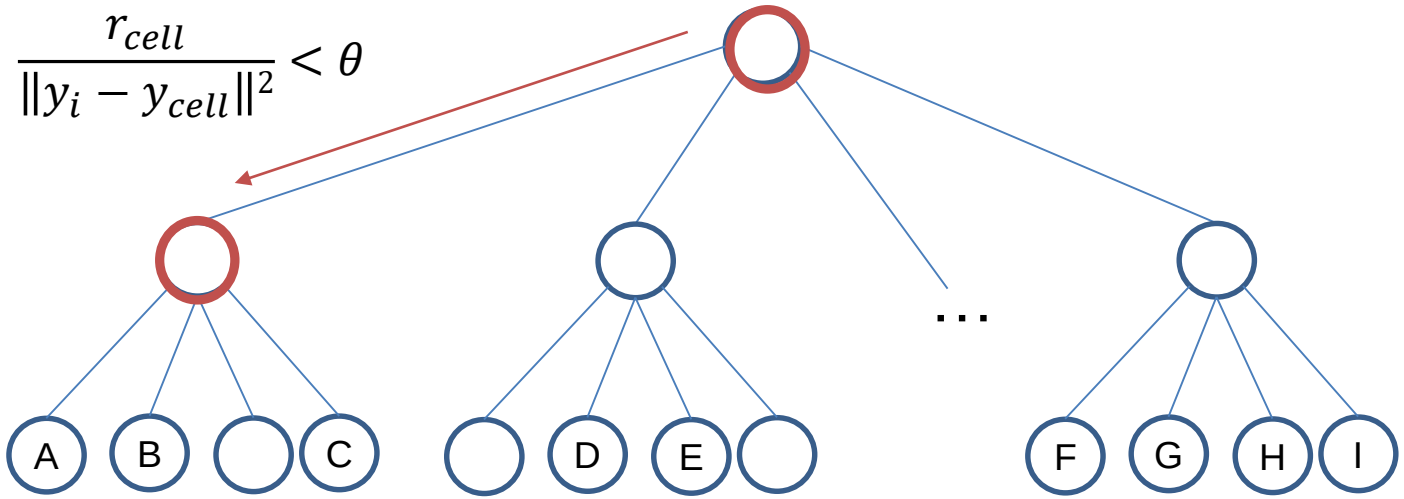
- In practice, application of Barnes-Hut optimization is performed using a tree.
- In the low-dimension, we implement this using a Quadtree.

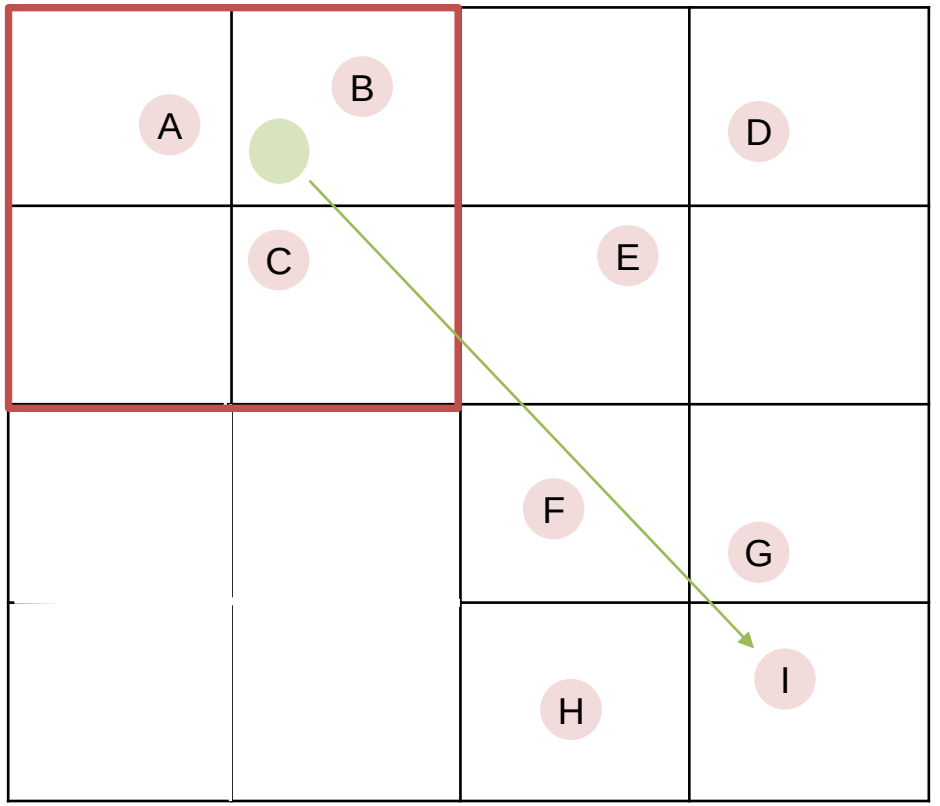




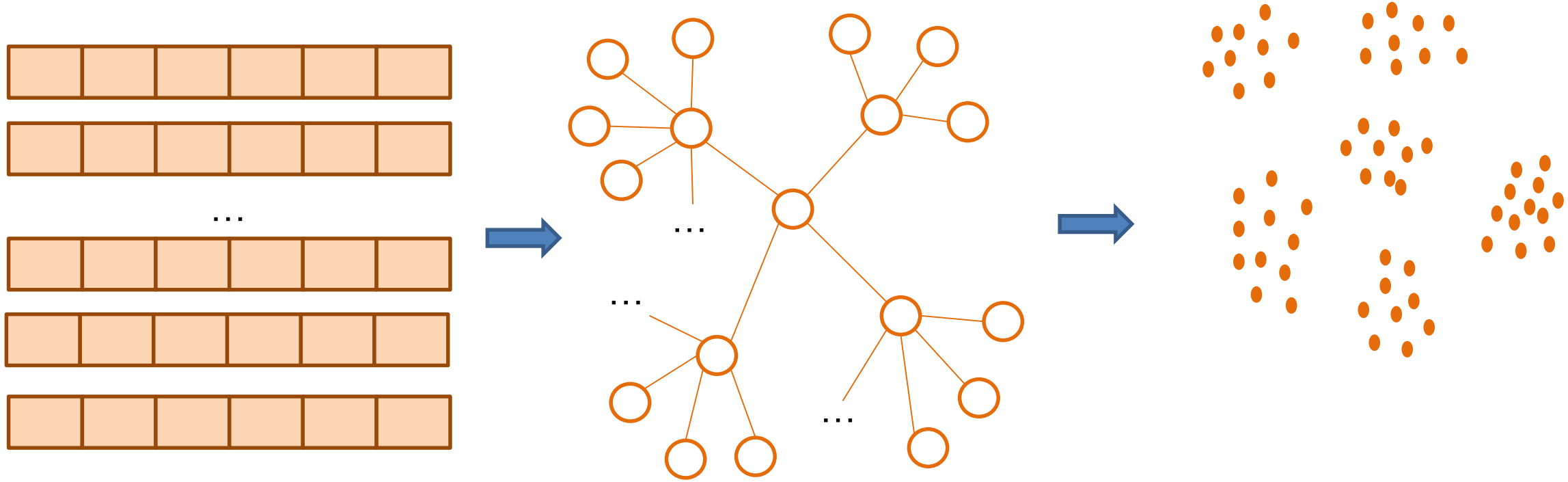
Quadtree

$$\frac{r_{cell}}{\|y_i - y_{cell}\|^2} < \theta$$





Barnes Hut Approximation



(a) High – Dimensional Feature Vectors

(b) K-nearest neighbor graph (KNN)

(c) 2 – dimensional layouts

Issues with t-SNE

- t-SNE only works in batch mode
  - Assumes all points are known up-front
  - To add new points, need to re-run t-SNE

- Tree structures don’t scale for high dimensions
  - Vantage-Point (VP) Tree are used to represent high-dimensional data
  - VP Trees do not scale for dimensions > 10

Streaming t-SNE

- Extend t-SNE to support the following operations, for a given state:
  - ***insert*** (point)
    - add a new data point to a state
  - ***update_neighborhood*** (point)
    - recompute a point's neighborhood (based on number and distances of nearby points)
  - ***remove*** (point)
    - Drop a point from the state
  - **update** ()
    - perform one step of gradient descent optimization

Batch t-SNE algorithm

- State of the t-SNE algorithm will be managed in data structure

```
def   batch_process (X) :
        state = TSNEState ( μ ,  T , θ)

        # populate state with data
        for  point  in  X:
                state.insert ( point )

        # compute all neighborhoods
        for point in X:
                state.update_neighborhood ( point )

        # gradient descent
        for  t =0  to  T:
                state.update()

        return  state
```

Parameters:

μ - perplexity

T – number of iterations

θ – degree of approximation in Barnes-Hut

Batch t-SNE algorithm

- state of the t-SNE algorithm will be managed in data structure

```
def   stream_process (X) :

        state = TSNEState ( μ ,  T , θ)

        # main processing loop
        while    stream_has_data () :
                  point  = next_point  ()
                  state.insert ( point )
                  state.update_neighborhood (point)
                  state.update()
```

Parameters:

μ - perplexity

T – number of iterations

θ – degree of approximation in Barnes-Hut

For simplicity, we assume one point is being inserted at each iteration, but inserting more points per iteration is possible.

Streaming t-SNE Demo

- Original t-SNE is batch based, hence requires all data points up-front.
- Can we lift this restriction?

- We propose a streaming t-SNE that supports
  - sliding window evaluation over a dataset
  - visualizing an evolving distribution in near real-time