

94-775 Unstructured Data Analytics

Lecture 5: PCA (cont'd), manifold learning

Slides by George H. Chen

Principal Component Analysis (PCA)

Demo

PCA Recap

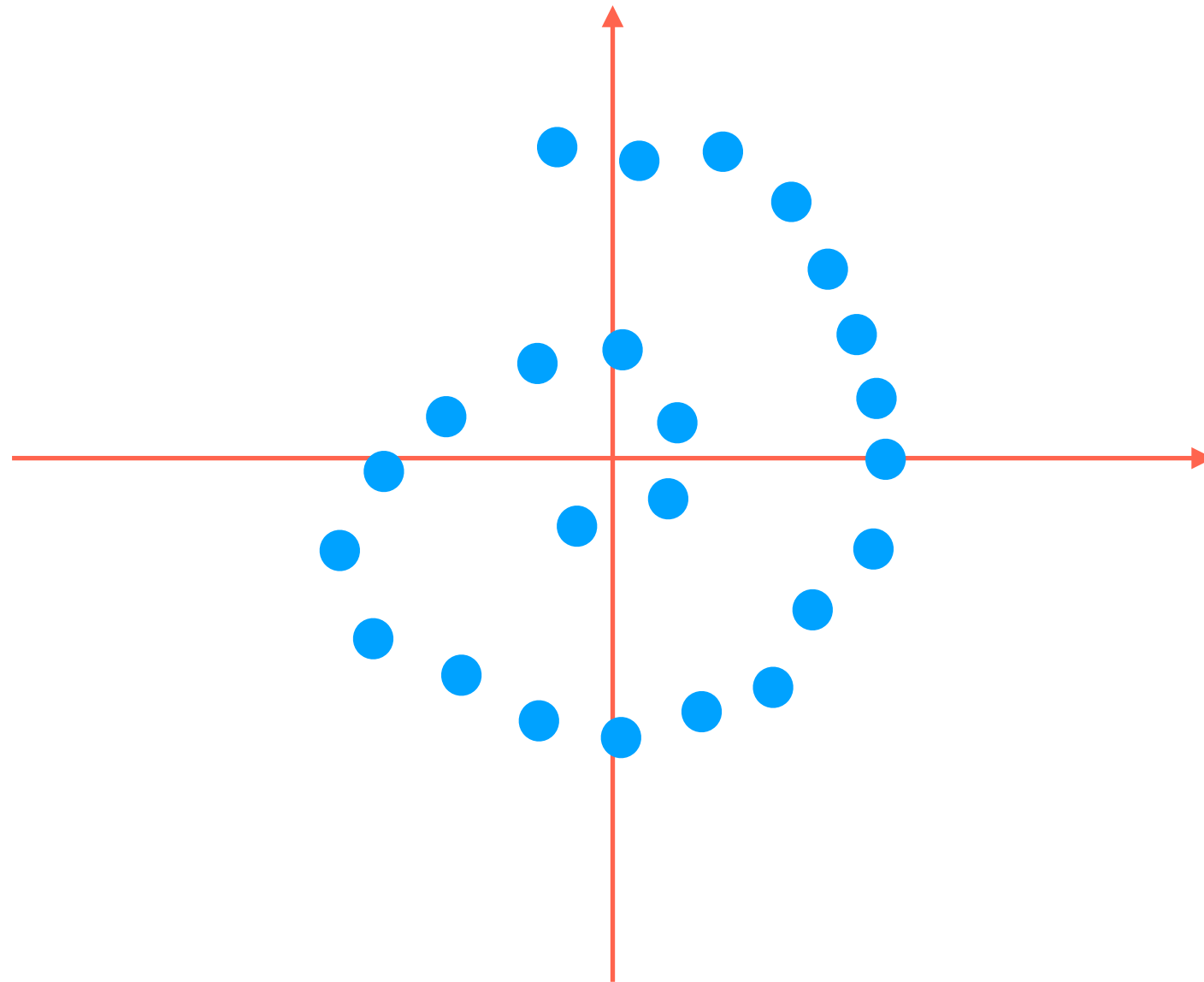
- PCA reduces high-dimensional data to k dimensions
 - These k dimensions (the principal component directions) are the k orthogonal directions that explain the most variance in the data
- Fitting a PCA model means:
 - Figuring out the center of mass of the data we're fitting the model to
 - Figuring out "weights" for each principal component direction
 - We saw how to compute the PCA coordinates by taking an inner product (also called a dot product)
- After fitting a PCA model, we can also compute the fraction of variance explained by each principal component
- Reminder: a 3D PCA model contains the solution to a 2D PCA model as well as a 1D PCA model
 - More generally: if you have a k -dimensional PCA model, then we also have PCA models for number of dimensions from 1 up to k

When does PCA not work well?



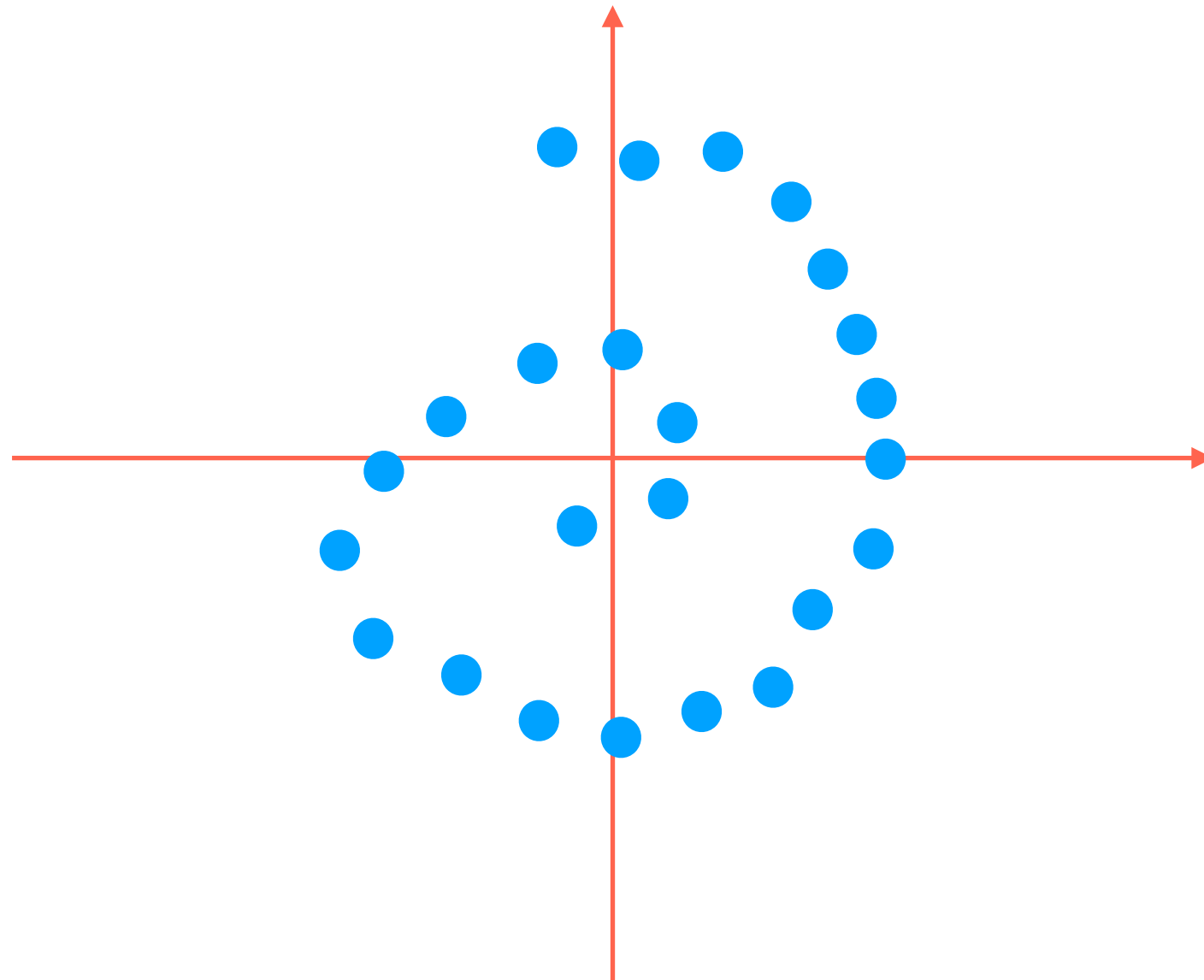
Image source: http://4.bp.blogspot.com/-USQEgoh1jCU/VfncdNOETcl/AAAAAAAAAGp8/Hea8UtE_1c0/s1600/Blog%2B1%2BIMG_1821.jpg

2D Swiss Roll

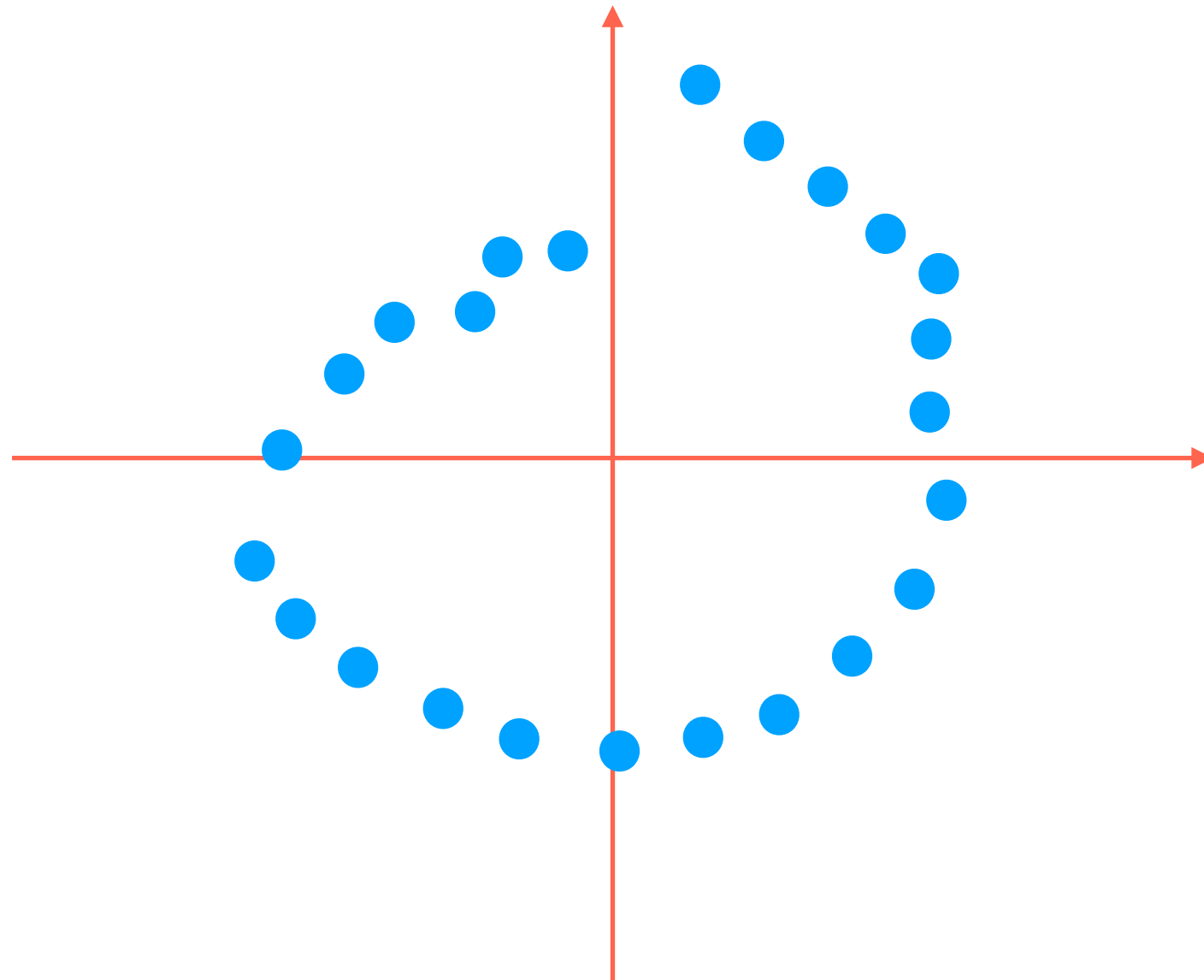


PCA would just flatten this thing and
*lose the information that the data actually lives
on a 1D line that has been curved!*

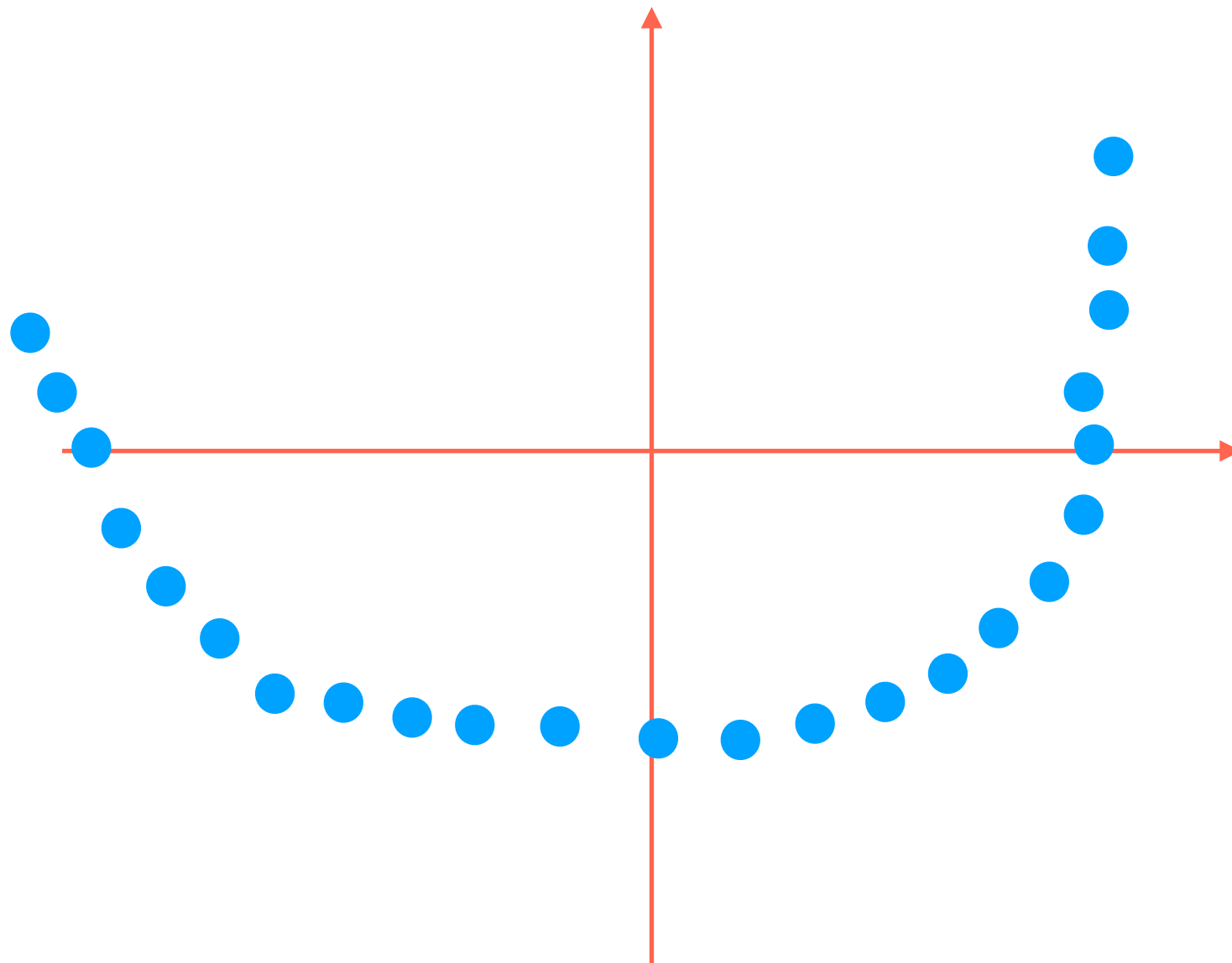
2D Swiss Roll



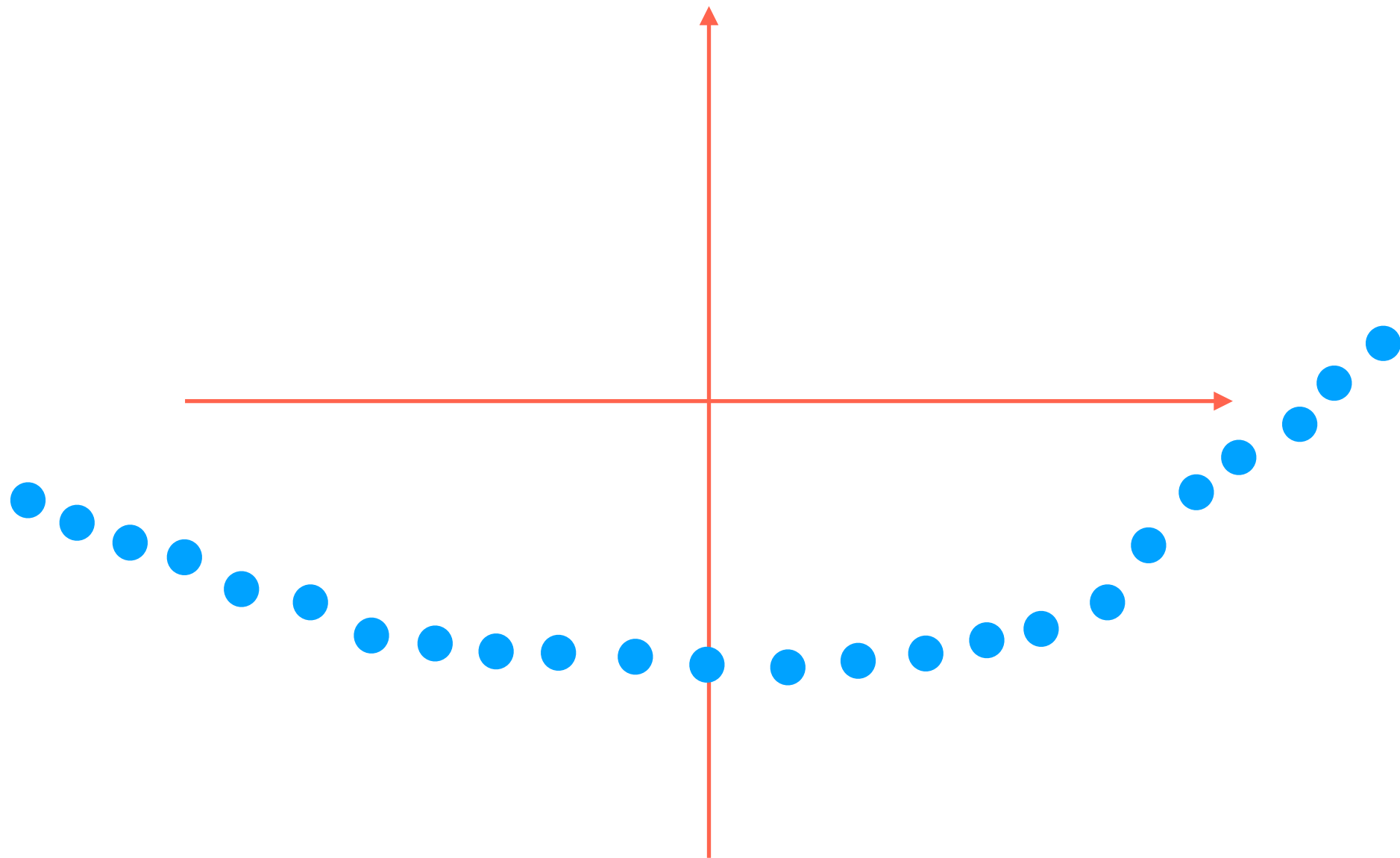
2D Swiss Roll



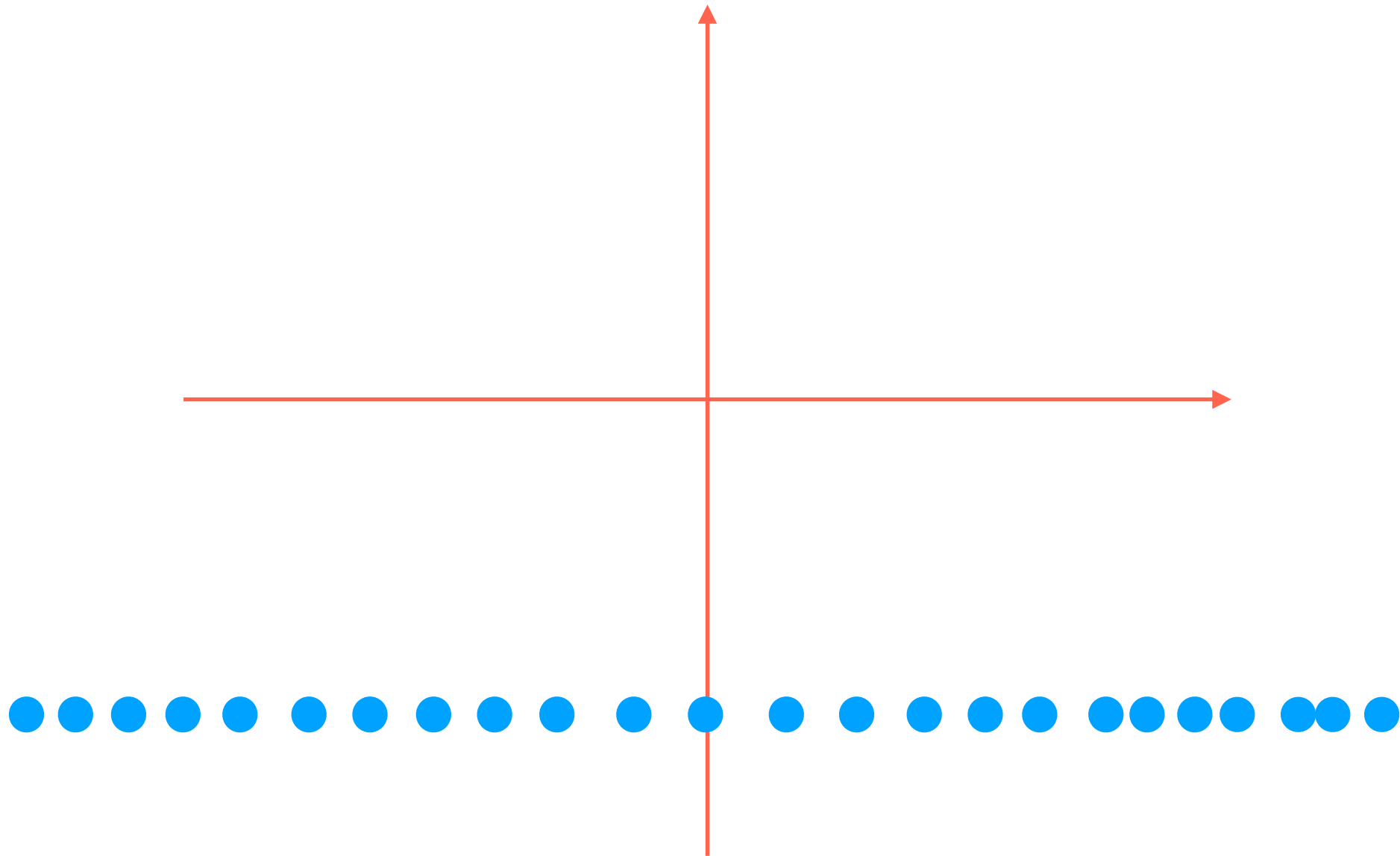
2D Swiss Roll



2D Swiss Roll



2D Swiss Roll

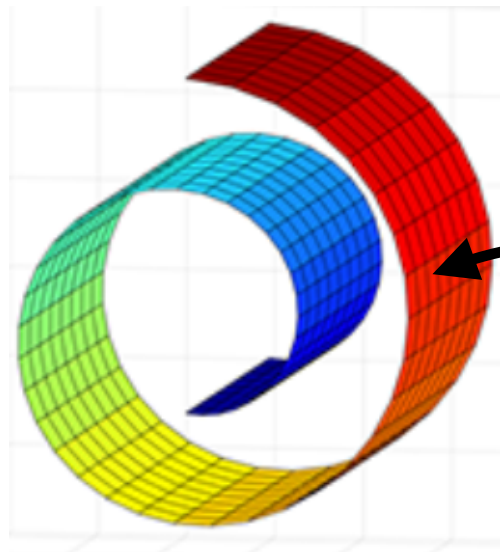


2D Swiss Roll



This is the desired result

Manifold Learning



The dataset here is clearly 3D

But when we zoom in a lot on any point, around the point it looks like a flat 2D sheet!

Another example: Earth is approximately a 3D sphere, but zooming a lot on any point, around the point it's approximately a 2D sheet

In general: if we have d -dimensional data where when you zoom in a lot, the data dimensionality is smaller than d , then the lower-dimensional object is called a **manifold**

- We have the data's high-dim. coordinates, but we want to find the low-dim. coordinates (on the manifold) → this is manifold learning
- Manifold learning is *nonlinear* whereas PCA is linear (this will make more sense after we see code demos)

Image source: "Head Pose Estimation via Manifold Learning" (Wang et al 2017)

Do Data Actually Live on Manifolds?

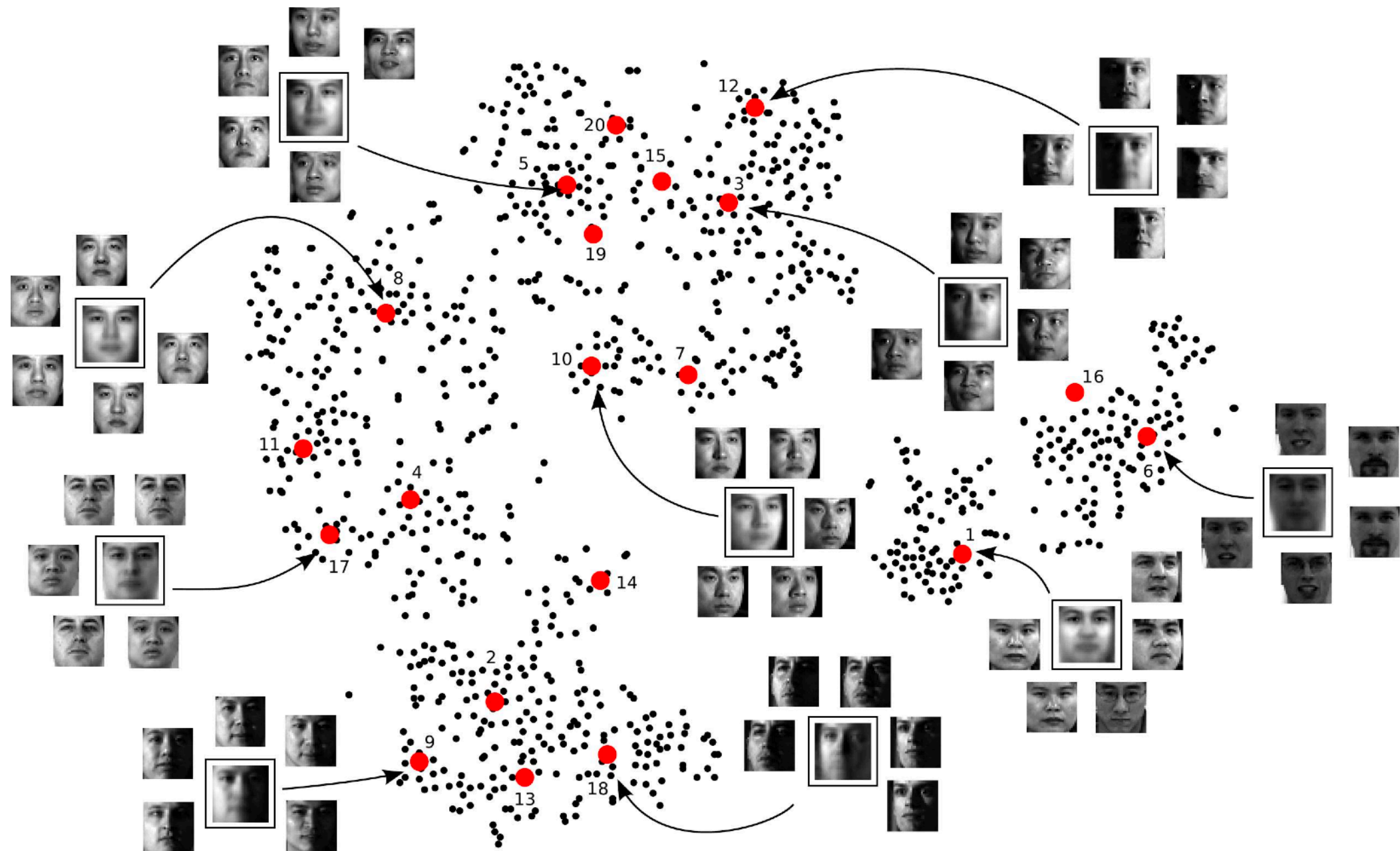


Image source: <http://www.columbia.edu/~jwp2128/Images/faces.jpeg>

Do Data Actually Live on Manifolds?

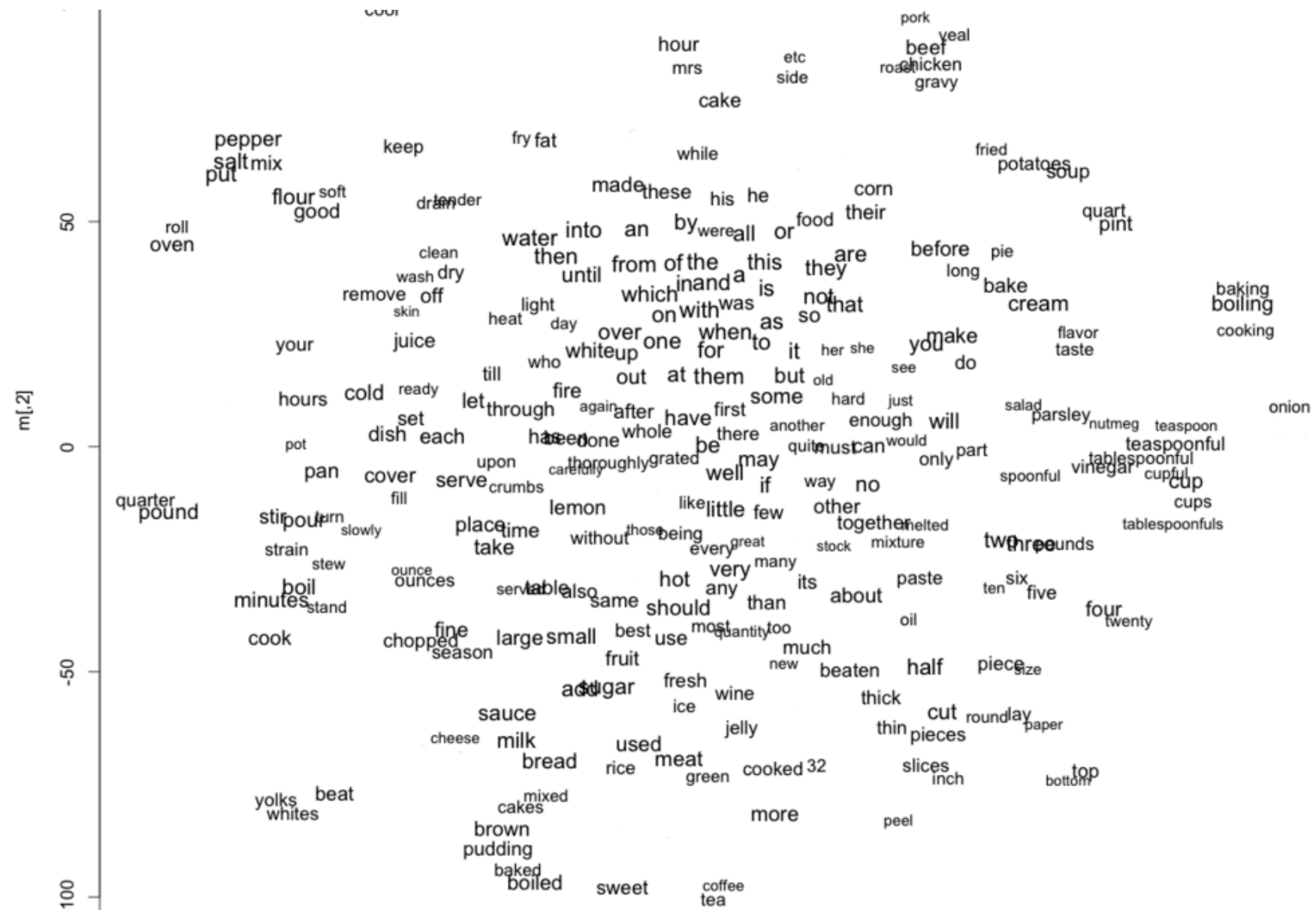
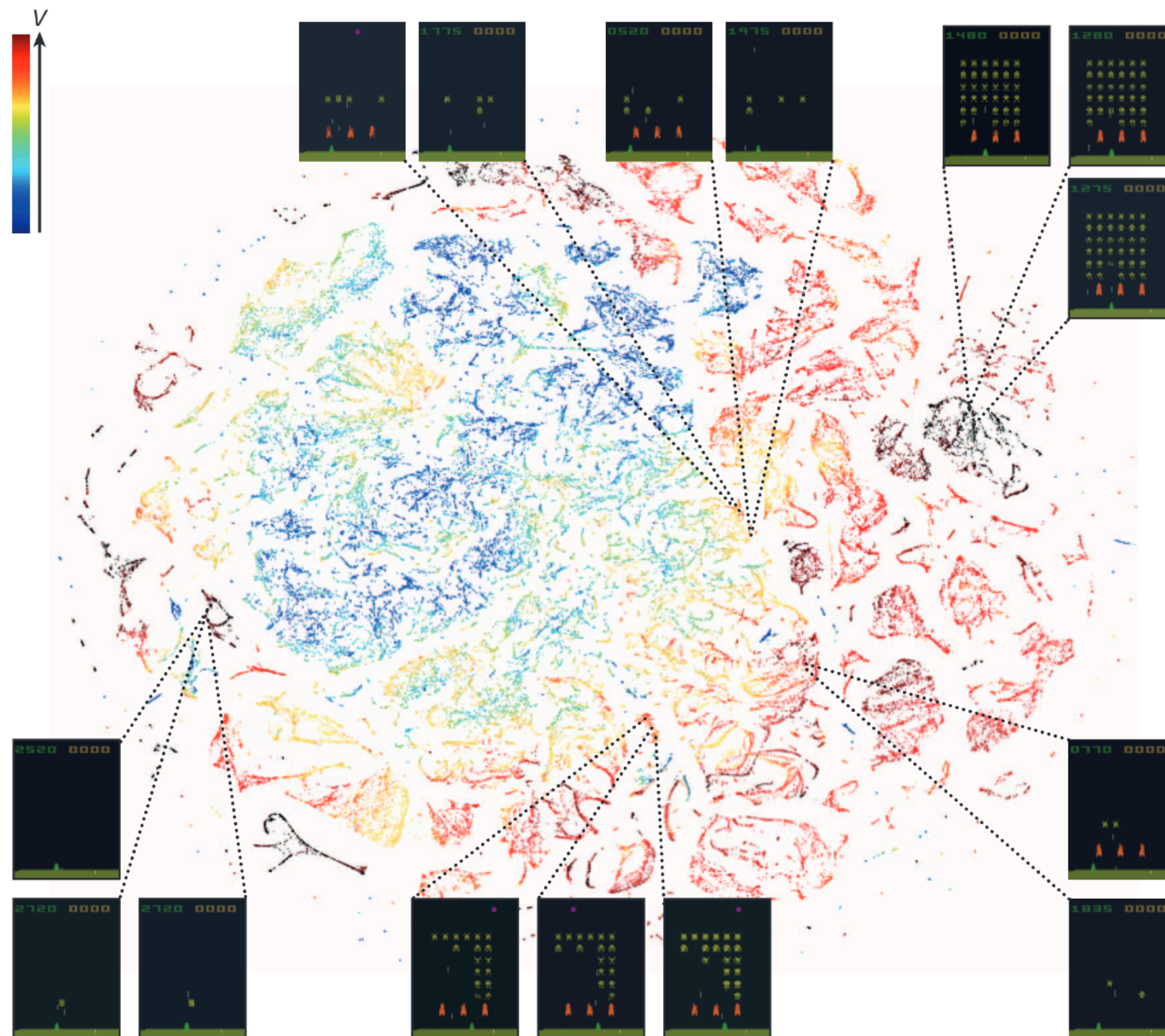


Image source: <http://www.adityathakker.com/wp-content/uploads/2017/06/word-embeddings-994x675.png>

Do Data Actually Live on Manifolds?



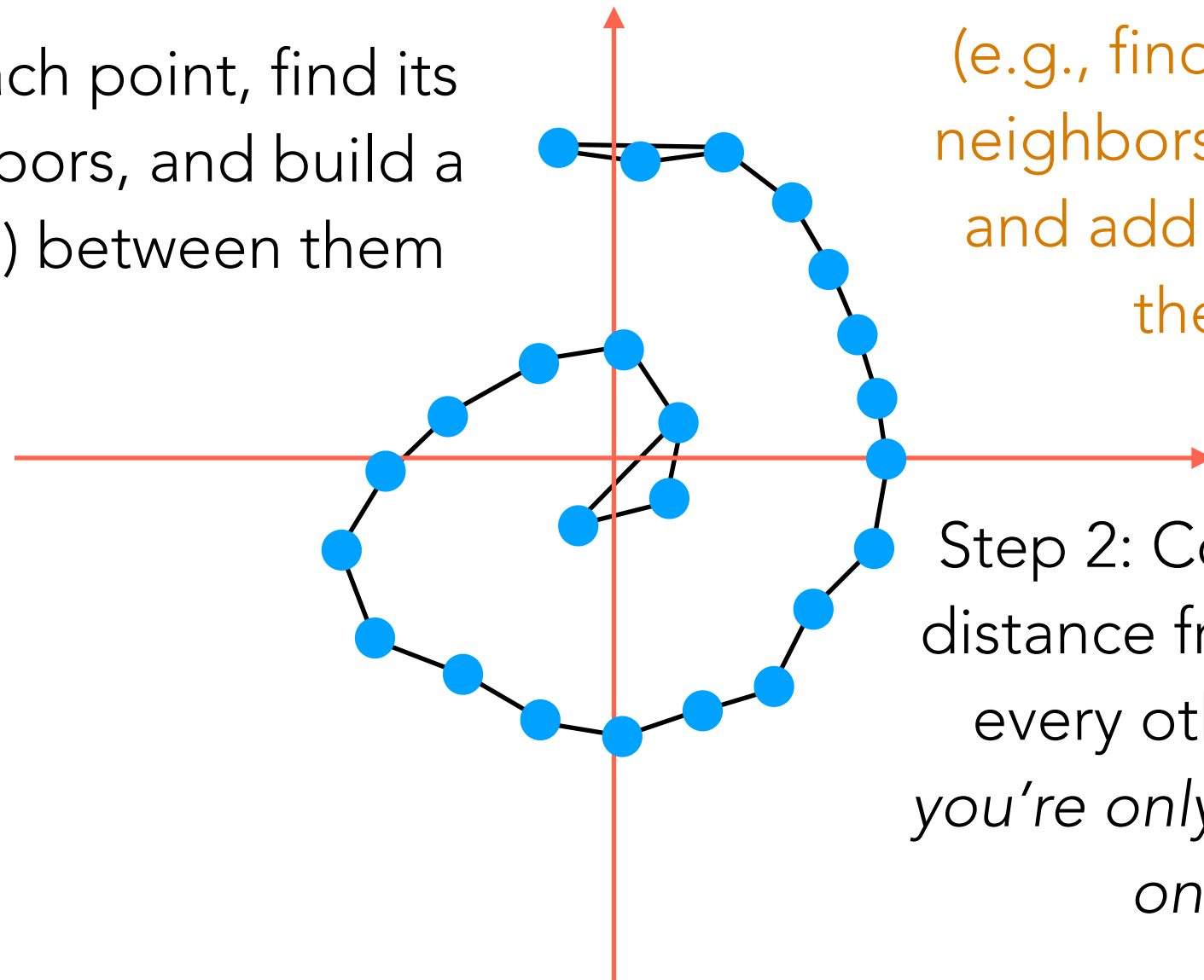
Mnih, Volodymyr, et al. Human-level control through deep reinforcement learning. Nature 2015.

There are many manifold learning methods

We begin with one that's easy to describe
(but it often doesn't work well in practice...)

Manifold Learning with Isomap

Step 1: For each point, find its nearest neighbors, and build a road ("edge") between them



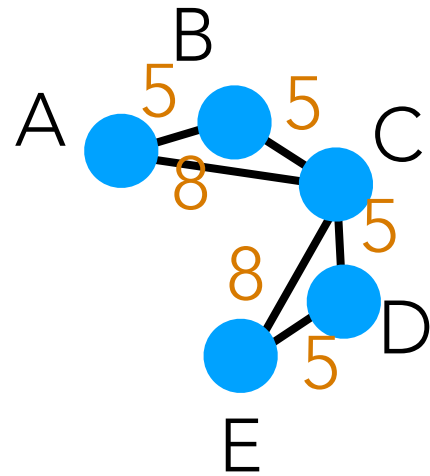
(e.g., find closest 2 neighbors per point and add edges to them)

Step 2: Compute shortest distance from each point to every other point *where you're only allowed to travel on the roads*

Step 3: It turns out that given all the distances between pairs of points, we can compute what the low-dimensional points should be (the algorithm for this is called *multidimensional scaling*)

Isomap Calculation Example

In orange: road lengths



2 nearest neighbors of A: B, C

2 nearest neighbors of B: A, C

2 nearest neighbors of C: B, D

2 nearest neighbors of D: C, E

2 nearest neighbors of E: C, D

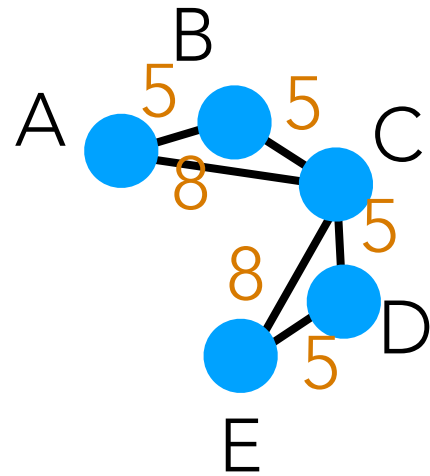
Build "symmetric 2-NN" graph
(add edges for each point to its 2
nearest neighbors)

Shortest distances between
every point to every other point
*where we are only allowed to
travel along the roads*

	A	B	C	D	E
A	0	5	8	13	16
B	5	0	5	10	13
C	8	5	0	5	8
D	13	10	5	0	5
E	16	13	8	5	0

Isomap Calculation Example

In orange: road lengths



2 nearest neighbors of A: B, C

2 nearest neighbors of B: A, C

2 nearest neighbors of C: B, D

2 nearest neighbors of D: C, E

2 nearest neighbors of E: C, D

Build "symmetric 2-NN" graph
(add edges for each point to its 2
nearest neighbors)

Shortest distances between
every point to every other point
*where we are only allowed to
travel along the roads*

	A	B	C	D	E
A	0	5	8	13	16
B	5	0	5	10	13
C	8	5	0	5	8
D	13	10	5	0	5
E	16	13	8	5	0

This matrix gets fed into
multidimensional scaling to get 1D
version of A, B, C, D, E

The solution is not unique!

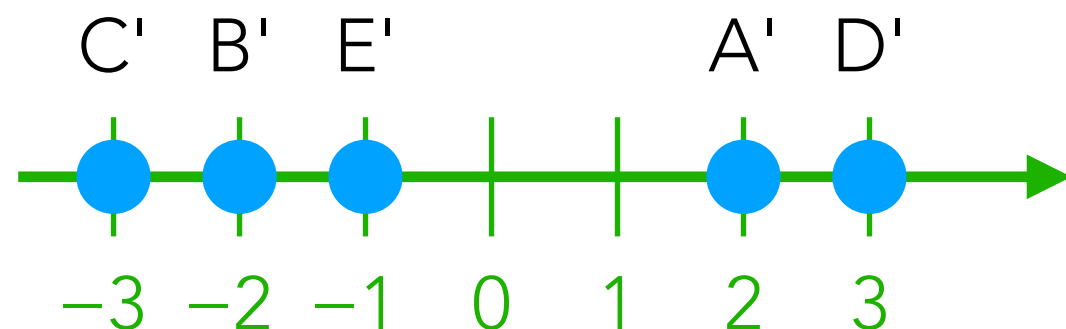
Multidimensional Scaling (MDS)

High-dimensional land

	A	B	C	D	E
A	0	5	8	13	16
B	5	0	5	10	13
C	8	5	0	5	8
D	13	10	5	0	5
E	16	13	8	5	0

Low-dimensional land

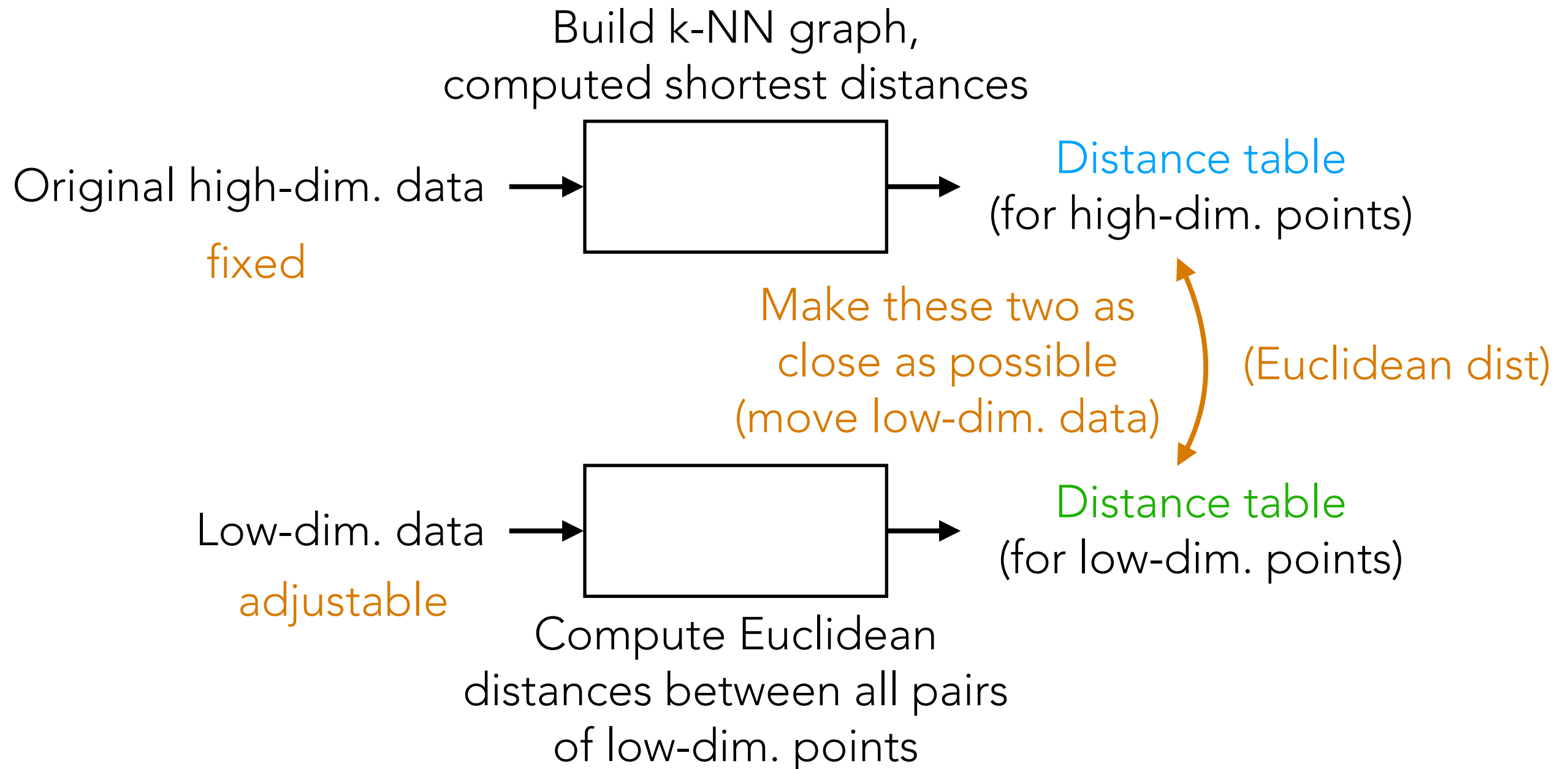
Suppose we have a guess for where the low-dimensional points are



	A'	B'	C'	D'	E'
A'	0	4	5	1	3
B'	4	0	1	5	1
C'	5	1	0	6	2
D'	1	5	6	0	4
E'	3	1	2	4	0

MDS moves the low-dim. points to make the 2 tables as close as possible

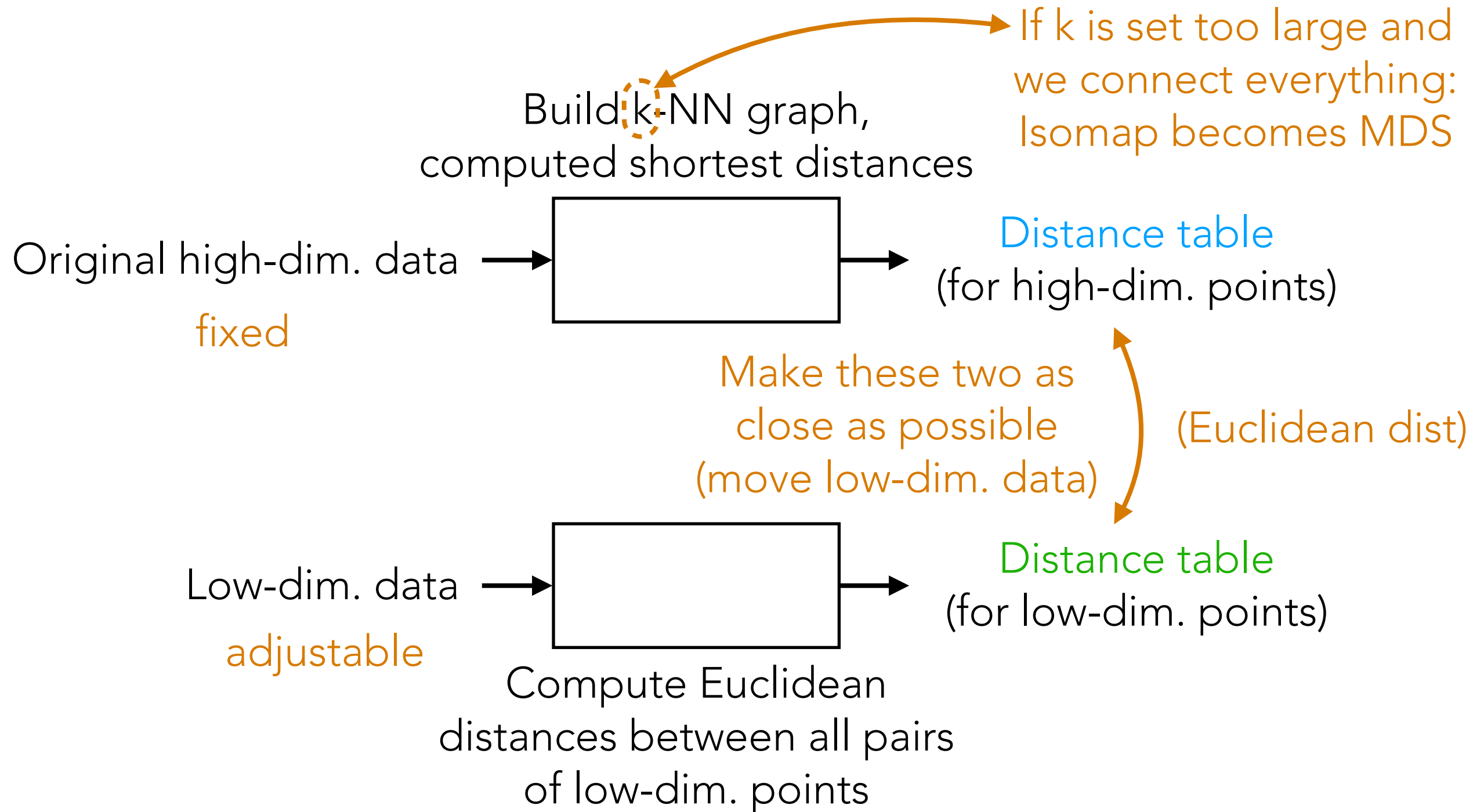
Isomap



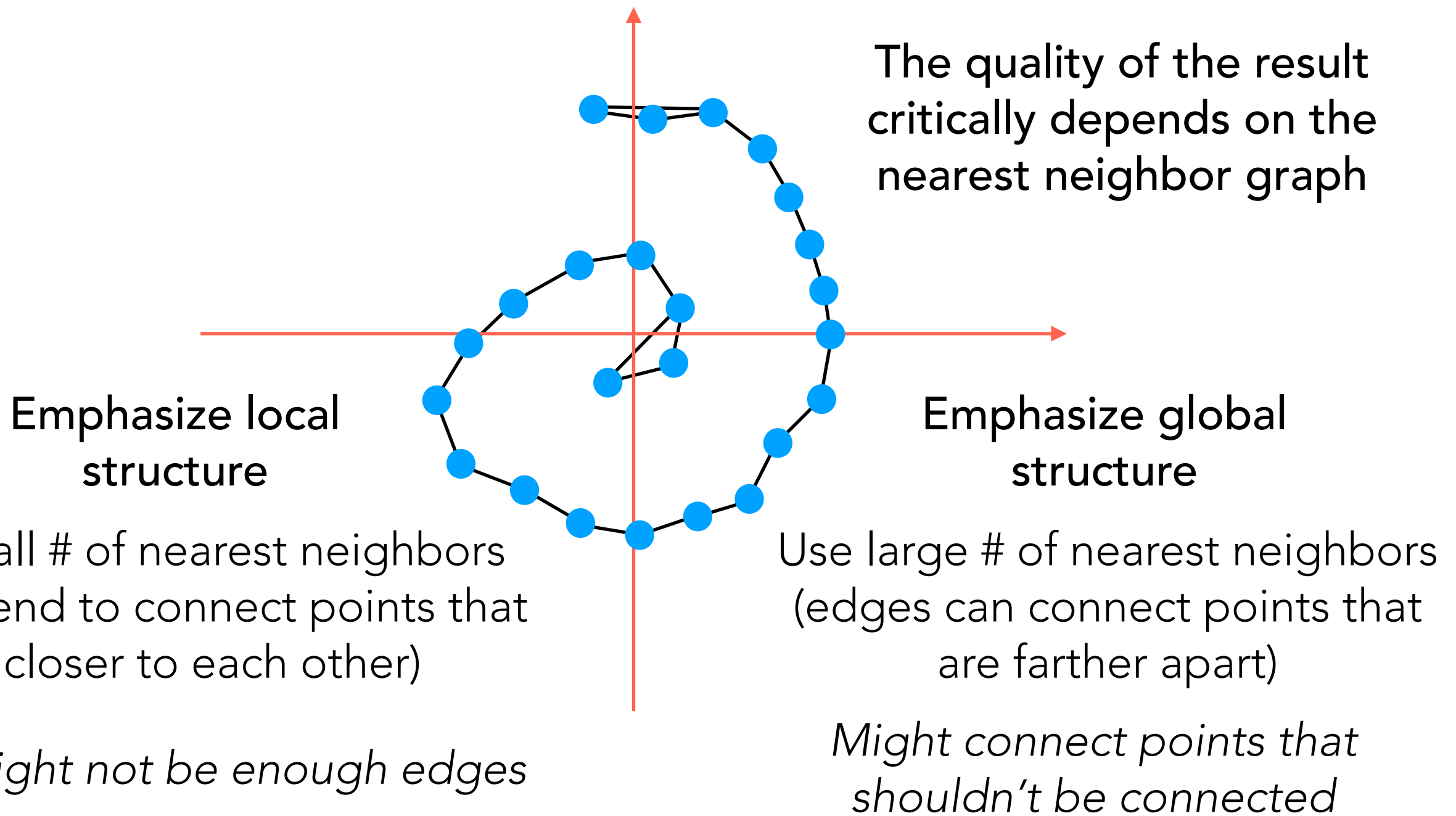
Isomap Calculation Example

Demo

Isomap



Some Observations on Isomap



In general: try different parameters for nearest neighbor graph construction when using Isomap + visualize