# TDT4113 Project 6: Nonlinear Dimensionality Reduction for Visualization

# Purpose of the project

- Gain a basic appreciation of
  - data science programming
  - data visualization
- Implement a data science algorithm with given pseudocode
- Be aware of efficiency of the implemented code

# Practicals of Project 6

- The project will be done in groups of size four.
- This project is compulsory
  - If you fail the project, you fail the whole course
- Your program can use only the allowed packages
  - cheating will fail your project
- Your code must be uploaded to BLACKBOARD before 8:00am on 7 April, 2021 and demonstrated before 8pm on 7 April, 2021.
- Your code needs to have a Pylint level of at least 8.0

# Dimensionality Reduction (DR) for Visualization

- Visualization helps digestion of massive data quickly
- Digital objects are often high-dimensional
- But human can see only 2D or 3D
- Dimensionality reduction $\mathbb{R}^D \mapsto \mathbb{R}^d$ $(D \gg d)$is needed
  - Input: $x_i \in \mathbb{R}^D$, $i = 1, \ldots, N$
  - Output: $y_i \in \mathbb{R}^d$, $i = 1, \ldots, N$
  - $d = 2$ or $d = 3$ for visualization
- Numerous DR methods in the literature
- In this project
  - Principal Component Analysis (PCA)
  - Isometric Mapping (Isomap)
  - Student t-Distributed Stochastic Neighbor Embedding (t-SNE)

# Related mathematics

Brief recap of some university math:

- *Matrix*, e.g. $\begin{pmatrix} 3.0 & -5.3 & 2.1 \\ 12.4 & -1.9 & -7.6 \end{pmatrix}$
- *Matrix product* of $C = AB$, where $A \in \mathbb{R}^{m \times r}$, $B \in \mathbb{R}^{r \times n}$, $C \in \mathbb{R}^{m \times n}$, and $C_{ij} = \sum_{k=1}^{r} A_{ik} B_{kj}$.
- A pair of *eigenvalue* $\lambda$ and *eigenvector* $u$ for a matrix $A$ fulfills $Au = \lambda u$.
- *covariance matrix* of a multidimensional random variable $x$: $\Sigma = \mathbb{E}\left[(x - \mu)(x - \mu)^T\right]$, where $\mu = \mathbb{E}[x]$.
- Covariance matrices are symmetric and positive-semidefinite.

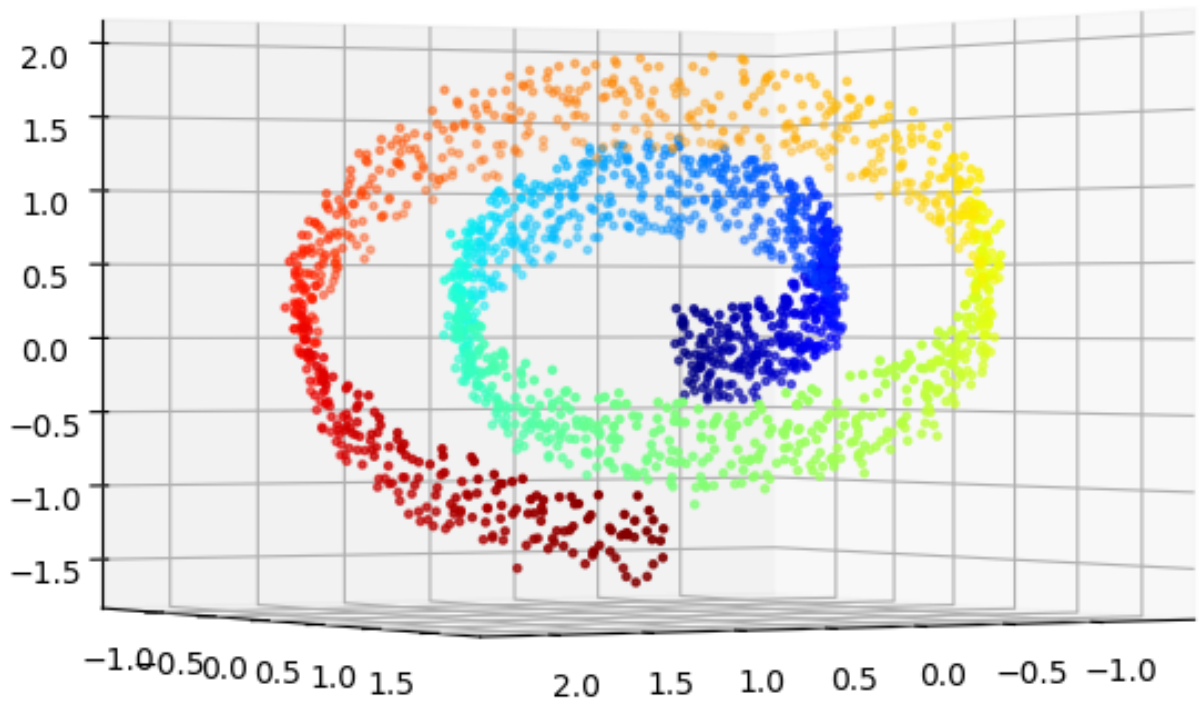
# Data sets

In the project you will use two data sets

- a synthetic data sets `SWISS-ROLL`
- a real-world data set `OPTDIGITS`

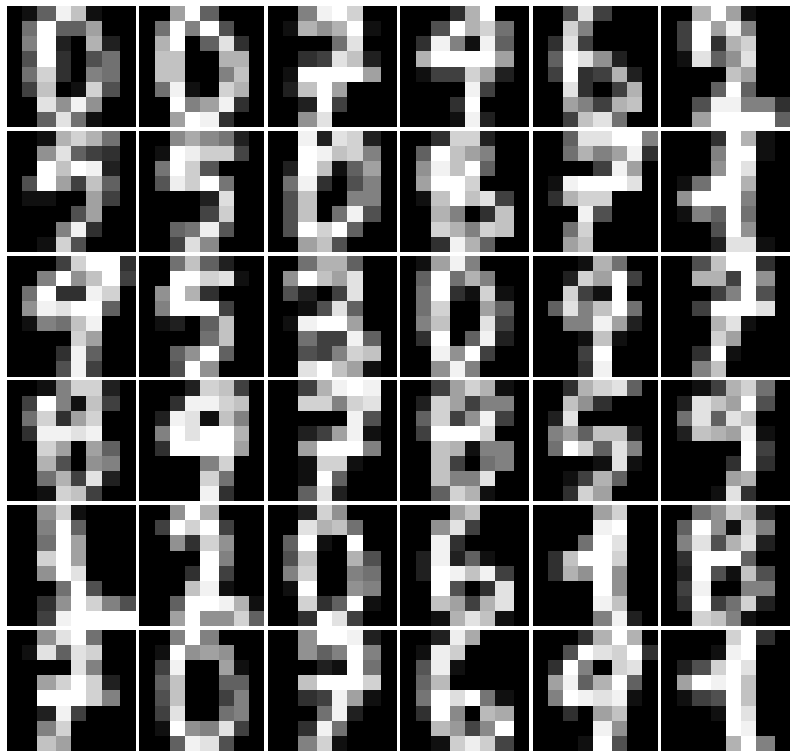
# Data set: SWISS-ROLL

- $N = 2000$ and $D = 3$
- Task: unfold the roll to 2D space
- Expected output: roughly a rectangle (color preserved)



# Data set: OPTDIGITS

- 5620 grayscale $8 \times 8$ images of handwritten digits
- Comes from UCI repository
- Task: the scatter plot should basically show the digit clusters



# Numpy

- Numpy is a Python library
  - dedicated for large, multi-dimensional arrays
  - with many built-in functions for these arrays
- Create from list of lists

```
>>> import numpy as np
>>> A = np.array([[1,3,5], [2,4,6]])
>>> A
array([[1, 3, 5],
       [2, 4, 6]])
```

- Read from CSV file into `numpy.array`

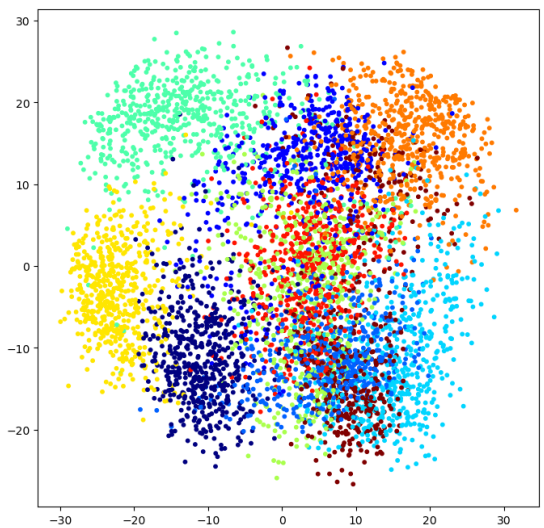
```
>>> X = np.genfromtxt('swiss_data.csv', delimiter=',')
>>> X.shape
(2000, 3)
```

- I suggest you read the quick start at
  https://numpy.org/doc/stable/user/quickstart.html

# Scatter plots in Python

Scatter plots show data points as dots in the visualization

- `import matplotlib.pyplot as plt`
- Example:
  - `plt.scatter(Y[:,0], Y[:,1], s=10, c=C, marker=".")`
  - `Y` is an $N \times 2$ array of the coordinates of the $N$ data points to be visualized
  - `s=10` specifies the marker size to 10
  - `c=C` specifies the colors of the dots, where `C` is an array of size $N$ integers
  - `marker="."` specifies solid dots as the markers



# Three DR methods in Project 6

- Principal Component Analysis (PCA)
- Isometric Mapping (Isomap)
- Student t-Distributed Stochastic Neighbor Embedding (t-SNE)

# Principal Component Analysis (PCA)

PCA projects the data to a *linear* subspace such that the sum of squared distances between the projected points and the original points is minimum.

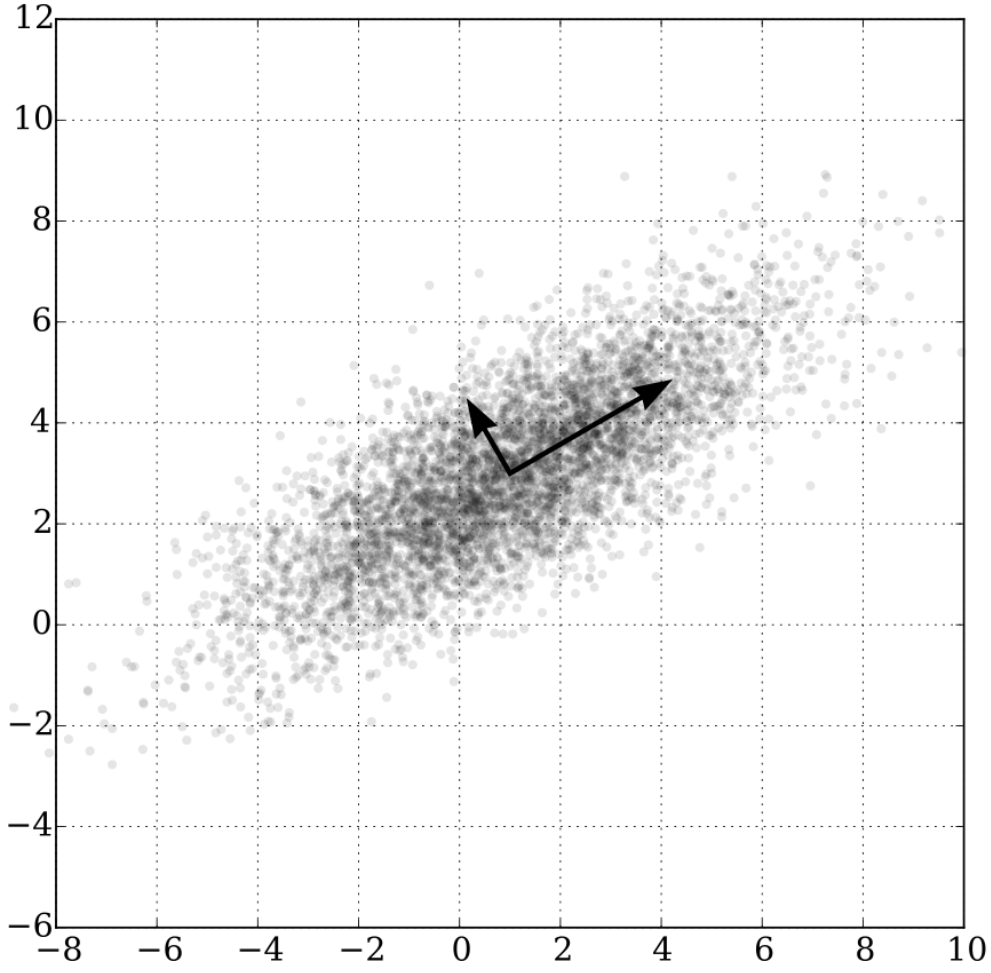



image source: Wikipedia

# Principal Component Analysis (PCA)

- Derivation (not required for the project)

$$\begin{aligned}
\min_{w:\|w\|=1} \quad & \mathbb{E}\left\{\|x - ww^T x\|^2\right\} \\
&= \mathbb{E}\left\{x^T x - 2x^T ww^T x + x^T ww^T ww^T x\right\} \\
&= \mathbb{E}\left\{x^T x - 2x^T ww^T x + x^T ww^T x\right\} \\
&= \mathbb{E}\left\{x^T x - x^T ww^T x\right\} \\
&= \mathbb{E}\left\{x^T x - w^T xx^T w\right\}
\end{aligned}$$

- This is equivalent to

$$\max_{w:\|w\|=1} w^T \mathbb{E}\left\{xx^T\right\} w = \mathbb{E}\left\{(w^T x)^2\right\}$$

- If $x$ is centered (i.e. mean removed), $\mathbb{E}\left\{xx^T\right\}$ is the covariance matrix $\Sigma$ of $x$

# Principal Component Analysis (PCA)

- Least squared error is equivalent to finding the directions with maximal variances after centering

$$\max_{w:\|w\|=1} w^T \mathbb{E}\left\{xx^T\right\} w = \mathbb{E}\left\{(w^T x)^2\right\}$$

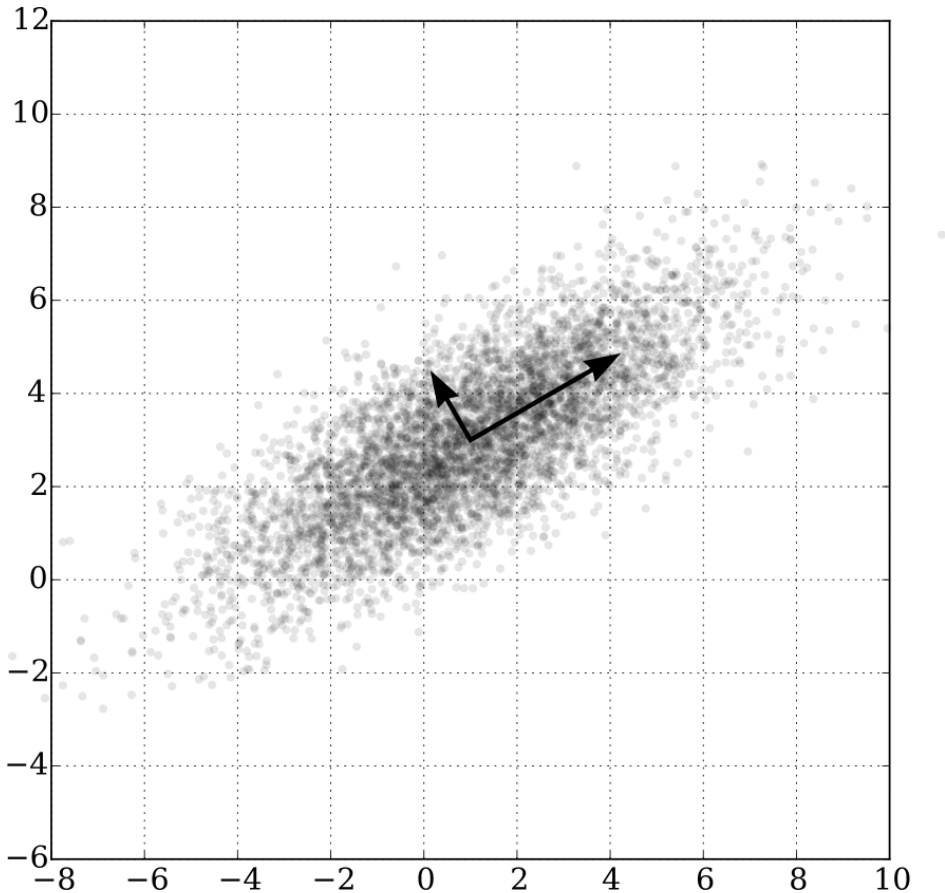



image source: Wikipedia

# PCA Steps

- Centering
- Find the directions of maximal variances

Centering is easy. For $i = 1, \ldots, N$

$$x_i \leftarrow x_i - \mu,$$

where (you can use `numpy.mean`)

$$\mu = \frac{1}{N} \sum_i x_i$$

# Find the directions of maximal variances (in math)

- These directions are identified by the eignvectors which correspond to the largest eigenvalues of the covariance matrix.
- Derivation (not required for the project)

$$\max_{w} \quad w^T \Sigma w \qquad \text{subject to } \|w\|^2 = 1$$

This is equivalent to optimizing the Lagrangian

$$\max_{w,\lambda} \quad \mathcal{L}(w) = w^T \Sigma w - \lambda(\|w\|^2 - 1)$$

with the Lagrangian multiplier $\lambda$. Zeroing the gradient

$$\frac{\partial \mathcal{L}}{\partial w} = 2\Sigma w - 2\lambda w = 0$$

gives $\Sigma w = \lambda w$, with the corresponding $w^T \Sigma w = \lambda$.

# Find the directions of maximal variances (in programming)

- `numpy.cov` calculates the covariance matrix `Sigma`
- If $D - 1 > d$, use `numpy.eigs` to find the largest eigenvalues and their corresponding eigenvectors
  - `from scipy.sparse.linalg import eigs`
  - `[eigenvalues, eigenvectors] = eigs(Sigma, k=d)`
- If $D - 1 = d$, use `numpy.linalg.eigh`
  - first find all eigenvalues and eigenvectors
  - then sort the eigenvalues
  - keep the largest $d$ eigenvalues and their corresponding eigenvectors

# PCA Transform

- The eigenvectors form the projection matrix $F \in \mathbb{R}^{D \times d}$
- The PCA transform for a data point $x$ is

$$y \leftarrow F^T(x - \mu)$$

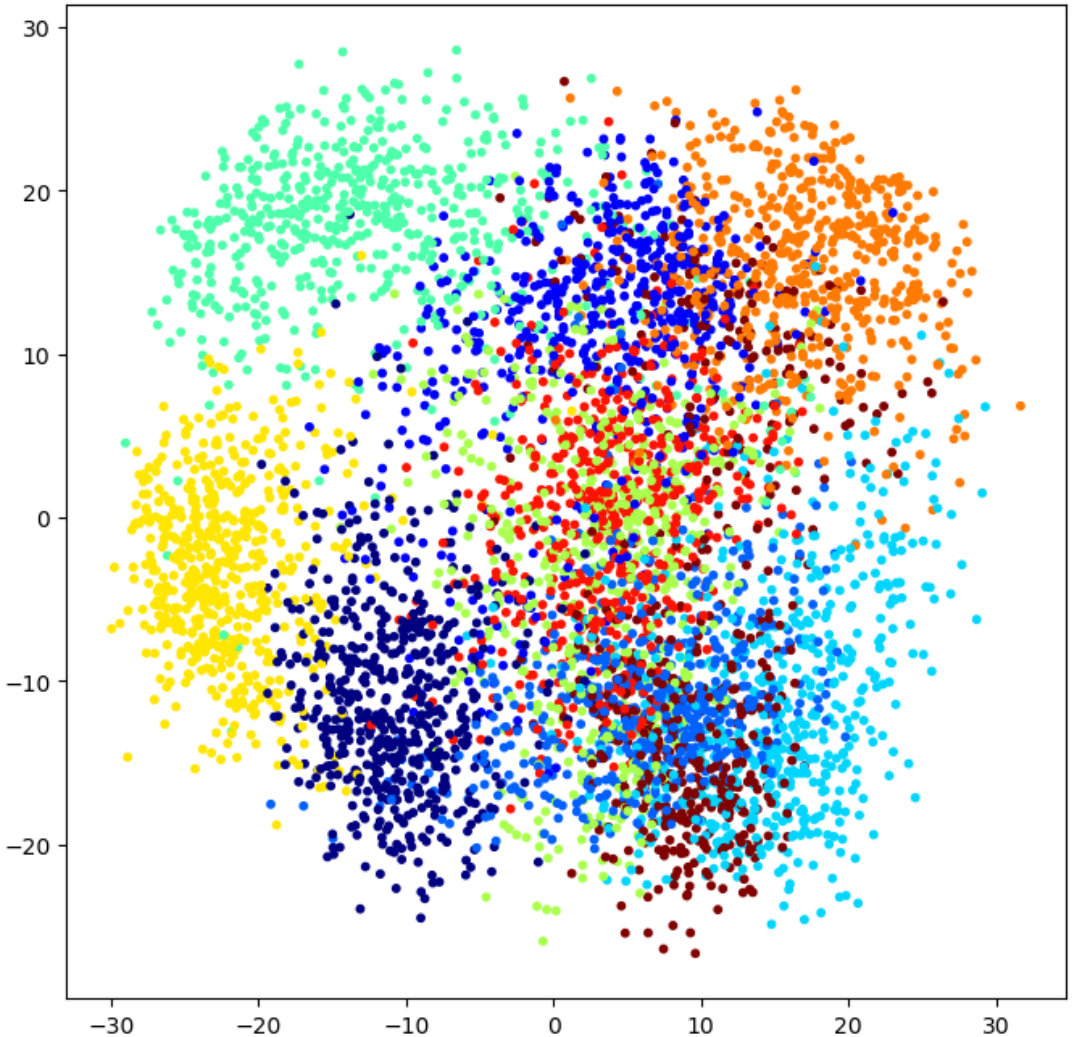
- Note:
  - $\mu$ and $F$ are obtained from the data set
  - $x$ can be in the training set or beyond the training set

Implementation - Part 1:

- Implement the PCA class which contains at least two functions `fit` and `transform`. Or you can combine the two functions into a single function `fit_transform` which needs not center the data twice.
- Apply your PCA method to the two data sets. Visualize the mapped points using scatter plots.
- You cannot import any existing packages or code that contains PCA. You can use only `numpy`, `scipy` and `matplotlib`.

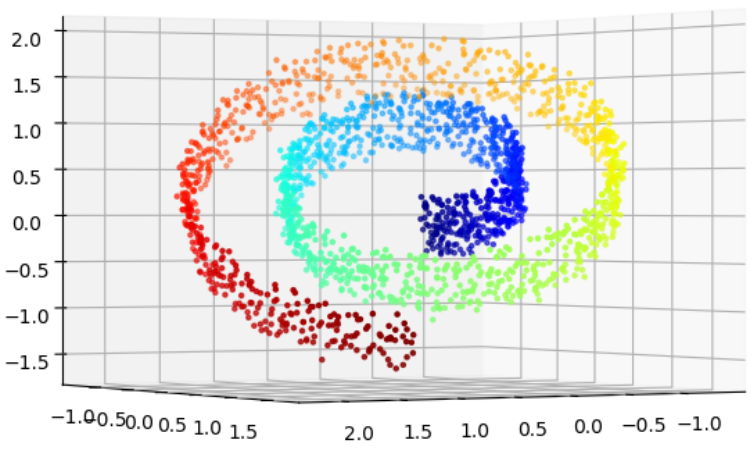
# PCA visualization for OPTDIGITS

Using `cmap='jet'`

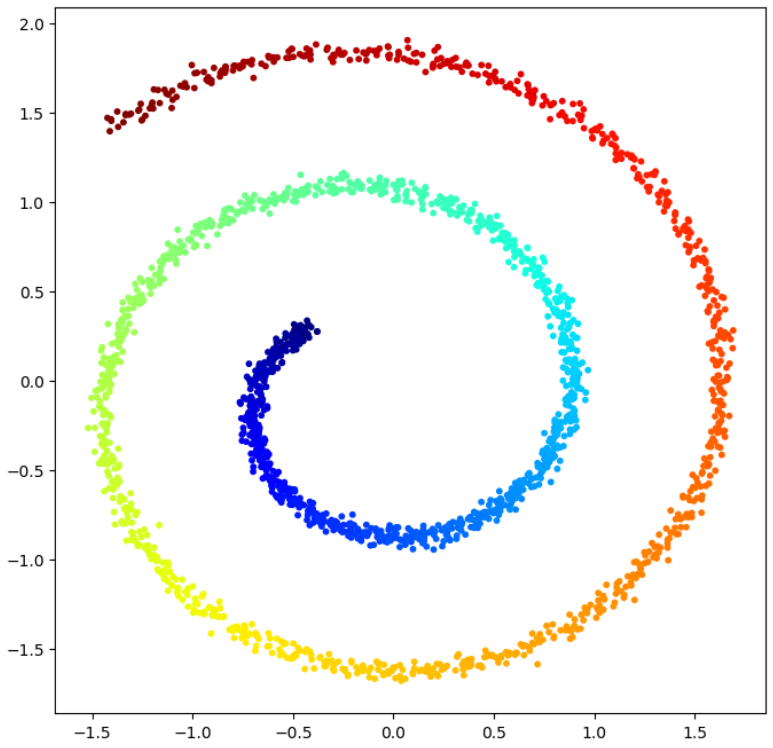


# PCA visualization for SWISS-ROLL

- PCA works mediocrely for OPTDIGITS
- but only gives an overlook of SWISS-ROLL




original data




PCA result

# Three DR methods in Project 6

- Principal Component Analysis (PCA)
- Isometric Mapping (Isomap)
- Student t-Distributed Stochastic Neighbor Embedding (t-SNE)

# Manifold

- PCA can only find linear subspace
- It cannot handle curved intrinsic subspace
- Curved manifold = topological space where Euclidean distances only locally works
- Geodesic distance = distances along the manifold

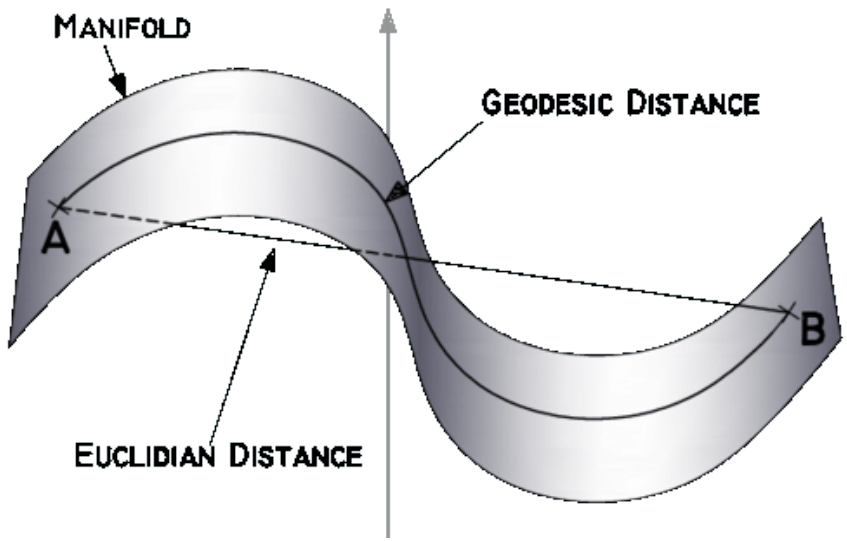



image source: *Graph embedding for speaker recognition*

# Isometric Mapping (ISOMAP)

ISOMAP tries to preserve the geodesic distances

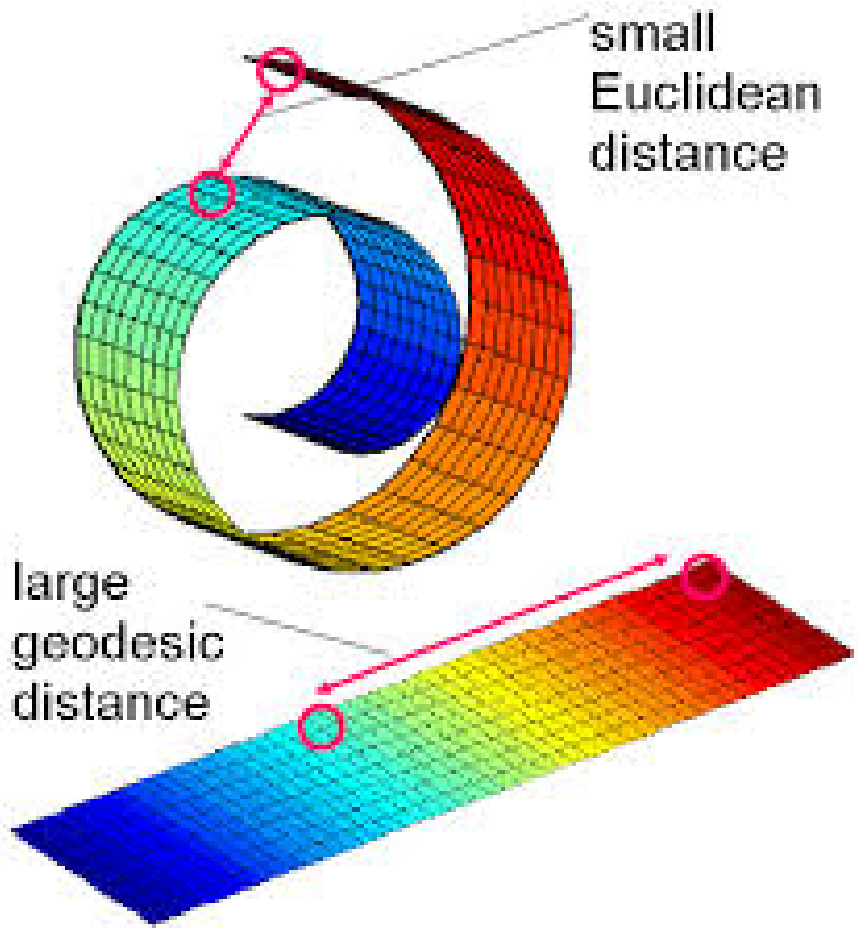



image source: *Deep Generative Models for Knowledge Transfer*

# ISOMAP major steps

1. Approximately calculate the geodesic distances
2. Map to 2D space, trying to preserved geodesic distances

# Approximately calculate the geodesic distances

1. Compute all pairwise Euclidean distances
   - Do not just use nested loops; that is slow
   - Write the square of difference ($X \in \mathbb{R}^{N \times D}$)

$$\sum_a (X_{ia} - X_{ja})^2 = \sum_a X_{ia}^2 + \sum_a X_{ja}^2 - 2 \sum_a X_{ia} X_{ja}$$

   - All three terms can be obtained with built-in matrix operations
2. Keep only the distances of the $k$-nearest neighbors (e.g. $k \in [10, 50]$)
   - Set the others to 0 (in programming)
   - This gives a distance matrix $D^{k\text{NN}}$ of the $k$NN graph
3. Perform a shortest-path graph search using $D^{k\text{NN}}$
   - This can be done by the Dijkstra's algorithm
   - You can use `graph_shortest_path` from `sklearn.utils.graph_shortest_path`
   - This returns the approximated geodesic distance matrix $D^{\text{geodesic}}$

# Multidimensional Scaling (MDS) derivation (optional)

- Input: the geodesic distance matrix $D^{\text{geodesic}}$
- Output: coordinates of the low-dimensional mapped points $Y = [y_1, \ldots, y_N]^T$, where each $y_i \in \mathbb{R}^d$ (in the project $d = 2$)
- There are several MDS variants; we use *classical MDS* here
- There is a sufficiently high-dimensional Euclidean vector space such that for all $i, j$

$$\|z_i - z_j\| = D_{ij}^{\text{geodesic}}$$

- We can further assume the vectors are centered, i.e. $\sum_i z_i = 0$
- Denote $Z = [z_1, \ldots, z_N]^T$
- Classical MDS finds $Y$ such that

$$ZZ^T \approx YY^T$$

by minimizing $\sum_{ij} \left(ZZ^T - YY^T\right)_{ij}^2$ over $Y$

# Multidimensional Scaling (continued)

- Note we don't need $Z$, but only $ZZ^T$
- We can write $B = ZZ^T$ in terms of $\boxed{D_{ij}^{(2)} = \left(D_{ij}^{\text{geodesic}}\right)^2}$

$$D_{ij}^{(2)} = \|z_i - z_j\|^2 = z_i^T z_i + z_j^T z_j - 2z_i^T z_j = B_{ii} + B_{jj} - 2B_{ij}$$

- With the assumption $\sum_i z_i = 0$, we have $\sum_i B_{ij} = 0$ and $\sum_j B_{ij} = 0$. So

$$2B_{ij} = -D_{ij}^{(2)} + \underbrace{\left(B_{ii} + \frac{1}{N}\sum_j B_{jj}\right)}_{\frac{1}{N}\sum_j D_{ij}^{(2)}} + \underbrace{\left(B_{jj} + \frac{1}{N}\sum_i B_{ii}\right)}_{\frac{1}{N}\sum_i D_{ij}^{(2)}} - \underbrace{\left(\frac{1}{N}\sum_i B_{ii} + \frac{1}{N}\sum_j B_{jj}\right)}_{\frac{1}{N^2}\sum_{ij} D_{ij}^{(2)}}$$

- In matrix notation ($\mathbf{1}$ is a column vector of 1's)

$$\boxed{B = -\frac{1}{2} J D^{(2)} J, \text{ with } J = I - \frac{1}{N}\mathbf{1}\mathbf{1}^T}$$

- This is called double-centering in mathematics

# Multidimensional Scaling (continued)

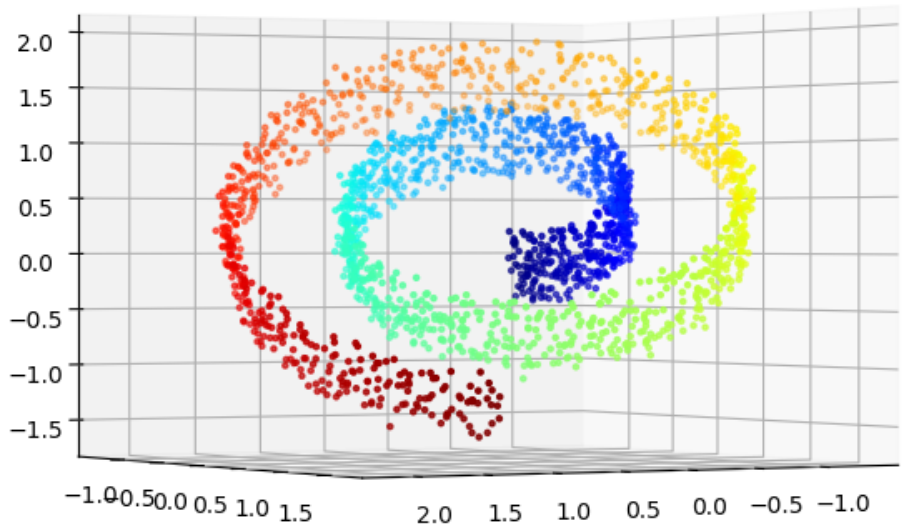
- Minimizing $\sum_{ij} \left(B - YY^T\right)_{ij}^2$ over $Y$ can be solved by truncated eigendecomposition (Eckart-Young-Mirsky theorem)
- Determine the two largest eigenvalues $\lambda_1$, $\lambda_2$ of $B$ and corresponding eigenvectors $e_1$ and $e_2$
- $Y \leftarrow E\Lambda^{1/2}$, where $E = [e_1, e_2]$ and $\Lambda = \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix}$
- The resulting matrix $Y \in \mathbb{R}^{N \times 2}$, where each row gives the coordinates of the mapped points
- For programming, you can just focus on the boxed steps

Implementation - Part 2:

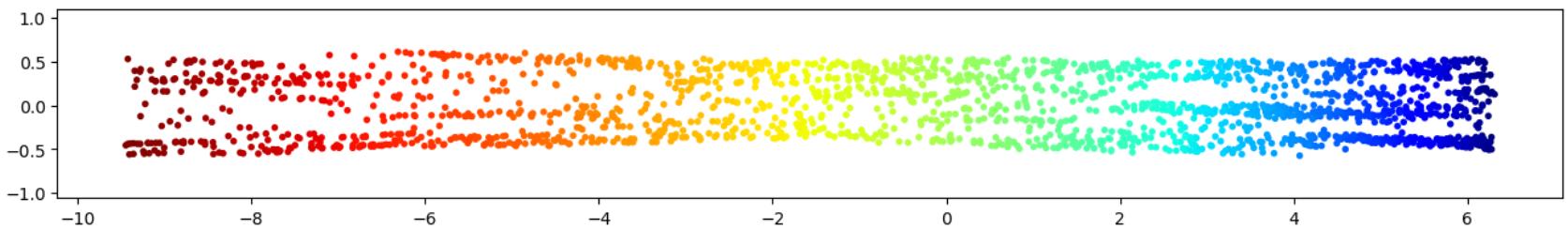
- Implement the Isomap method for dimensionality reduction.
- Apply your Isomap method to the two data sets. Visualize the mapped points using scatter plots.
- You can use only `numpy`, `scipy`, `matplotlib` and `graph_shortest_path` from `sklearn.utils.graph_shortest_path`. You cannot import other packages or functionalities in Scikit-Learn (sklearn). You cannot use other packages or code that contains Isomap.

# Isomap visualization for SWISS-ROLL

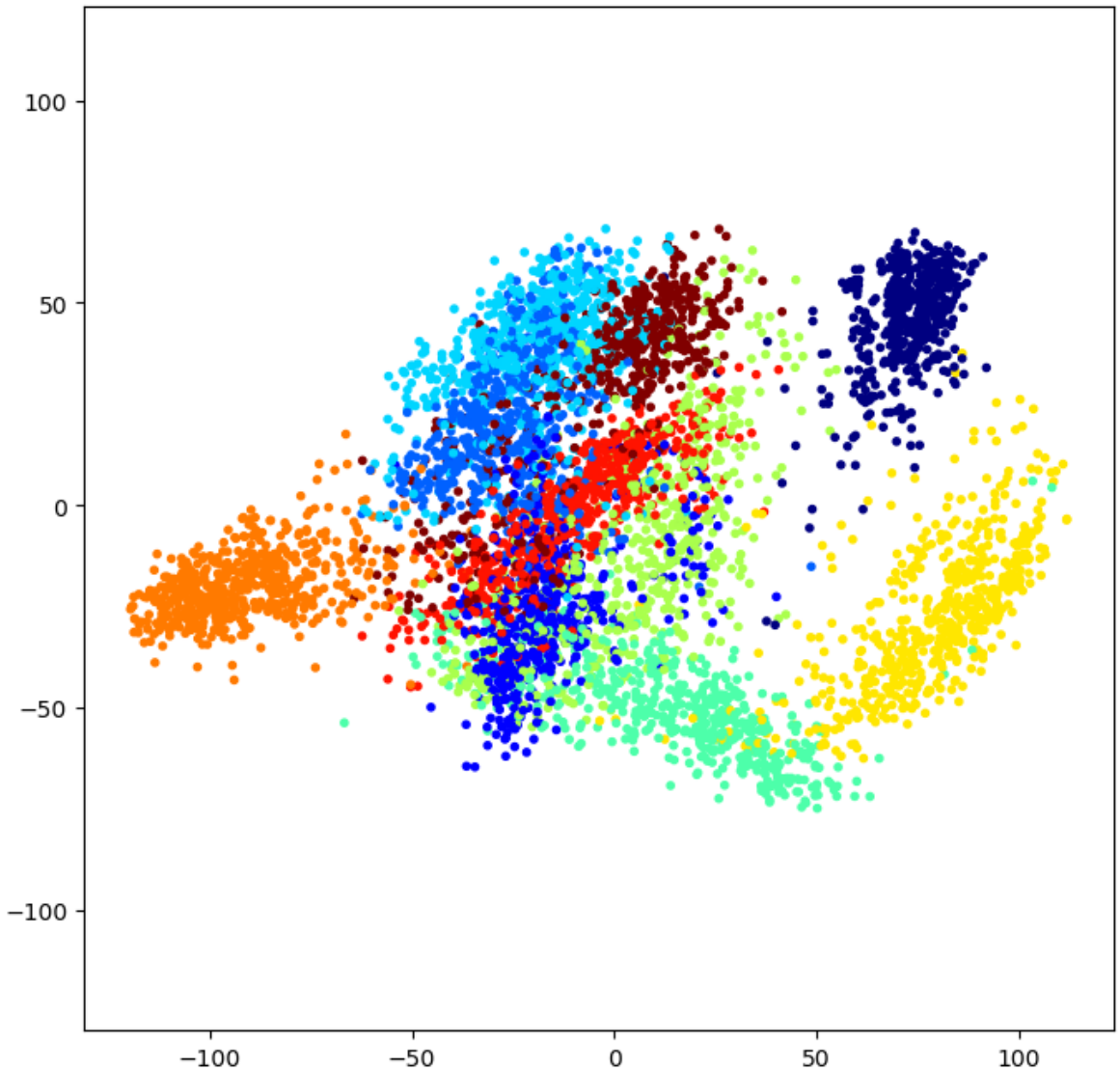
Original data




Visualization (using colors `c=np.arange(N)`)



# Isomap visualization for OPTDIGITS



# Three DR methods in Project 6

- Principal Component Analysis (PCA)
- Isometric Mapping (Isomap)
- Student t-Distributed Stochastic Neighbor Embedding (t-SNE)

# Student t-Distributed Stochastic Neighbor Embedding (t-SNE)

- ISOMAP works well for `SWISS-ROLL`, but still not good for `OPTDIGITS`
- t-SNE works much better in showing clusters
- Neighbor Embedding

| | input HD space | output 2D space |
|---|---|---|
| vectorial data | $X = \{x_i\}_{i=1}^N$ | $Y = \{y_i\}_{i=1}^N$ |
| similarities | $p_{ij} = \text{sim}(x_i, x_j)$ | $q_{ij} = \text{sim}(y_i, y_j)$ |

- Objective: $p \approx q$, by minimizing a divergence between $p$ and $q$
- Originally t-SNE employs the entropic affinities
- In the project you will use the $k$-NN graph

$$p_{ij} = \begin{cases} 1 & \text{if } i \neq j \text{ and } (x_i \text{ is among kNNs of } x_j \text{ or vice versa}) \\ 0 & \text{otherwise} \end{cases}$$

# t-SNE steps

- t-SNE uses $q_{ij} = \dfrac{1}{1 + \|y_i - y_j\|^2}$
- It has the tendency to separate clusters
- So t-SNE comprises two steps
  1. Compute the input similarity matrix $p$
  2. Minimize $D(P, Q)$ over $Y$, where

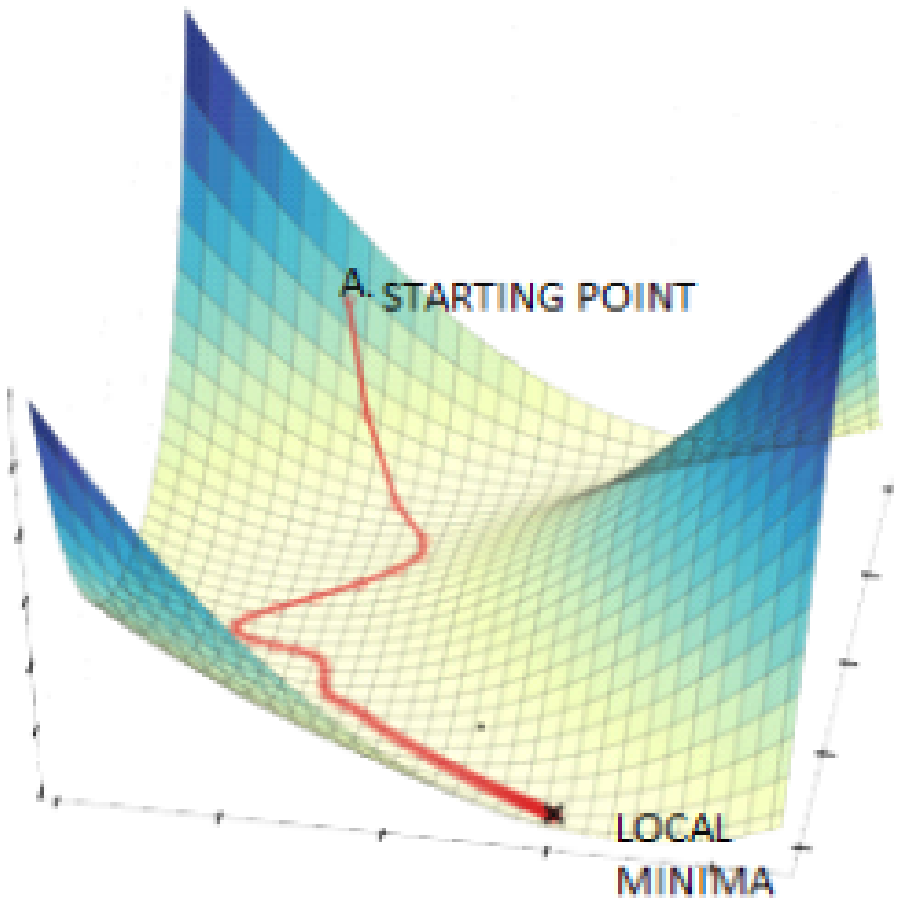
$$D(P, Q) = \sum_{i=1}^{N} \sum_{j=1}^{N} P_{ij} \ln \frac{P_{ij}}{Q_{ij}}$$

with

$$P_{ij} = \frac{p_{ij}}{\sum_{i=1}^{N} \sum_{j=1}^{N} p_{ij}} \qquad \text{and} \qquad Q_{ij} = \frac{q_{ij}}{\sum_{i=1}^{N} \sum_{j=1}^{N} q_{ij}}.$$

# Minimization by gradient descent

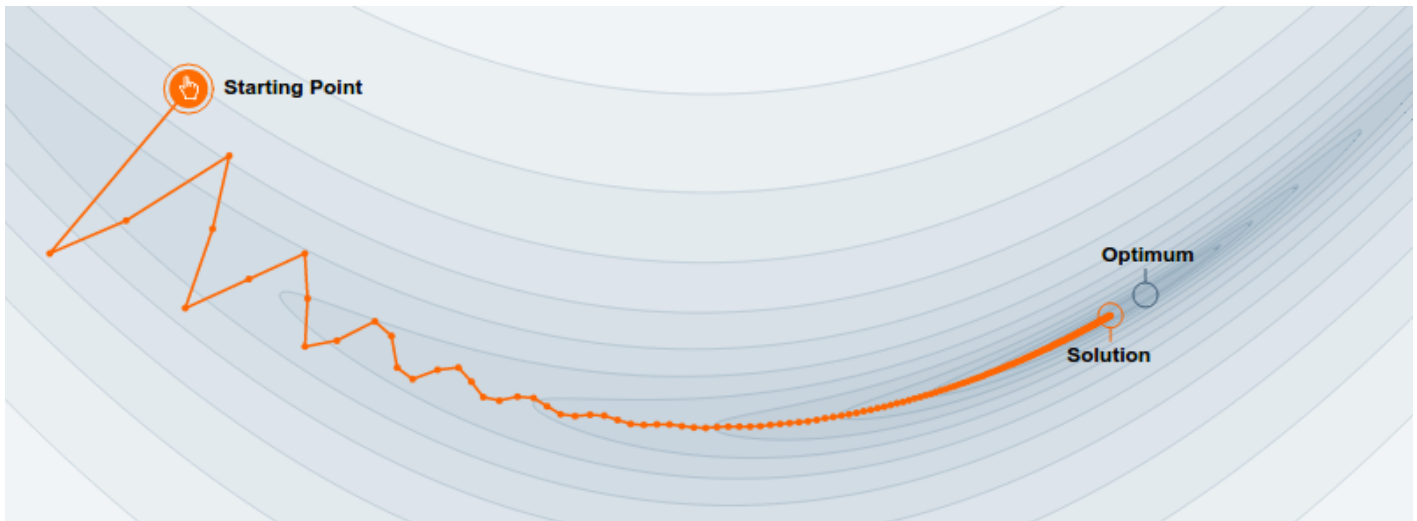
- Gradient is the vector of partial derivatives
- Gradient points to the steepest ascent direction
- So iteratively take small steps along the negative gradients
- Towards a local minima



# Gradient descent with momentum

- The surface is more complicated in practice
- Quite often there are long and narrow valleys
- Momentum is a technique to avoid many zigzags
- At the $t$th iteration

$$y_i^{(t+1)} \leftarrow y_i^{(t)} - \epsilon \frac{\partial D(P,Q)}{\partial y_i} + \alpha \left( y_i^{(t)} - y_i^{(t-1)} \right)$$



# The t-SNE algorithm

1. For $i = 1, \ldots, N$ and $d = 1, 2$
   - sample $y_{id}$ from Normal distribution $\mathcal{N}(0, 10^{-4})$
   - initialize the gain $g_{id} = 1$ and the change $\Delta_{id} = 0$
2. For iteration=1 to maximum_iteration

   2.1 Calculate $q$ and $Q$ using the current $y_i$'s

   2.2 calculate the gradient over each $y_i$ (denote by $\nabla_i$):

$$\nabla_i \stackrel{\text{def}}{=} \frac{\partial D(P, Q)}{\partial y_i} = 4 \sum_j (P_{ij} - Q_{ij})\, q_{ij}\, (y_i - y_j)$$

   2.3 update the gain $g_i$: (for $d = 1, 2$)

$$g_{id} \leftarrow \begin{cases} g_{id} + 0.2 & \text{if } \text{sign}(\nabla_{id}) \neq \text{sign}(\Delta_{id}) \\ g_{id} \times 0.8 & \text{if } \text{sign}(\nabla_{id}) = \text{sign}(\Delta_{id}) \end{cases}$$

   If $g_{id} < 0.01$, set $g_{id} = 0.01$.

   2.4 update the change $\Delta_i \leftarrow \alpha \Delta_i - \epsilon g_i \nabla_i$

   2.5 update $y_i \leftarrow y_i + \Delta_i$ where $\epsilon > 0$ is the learning step size

# Implementation details

- suggested maximum iteration 500 or 1000 and $\epsilon = 500$
- "lying" $P$ at beginning
  - use $4P$ in place of $P$ in the first 100 iterations
  - and change back to normal $P$ afterwards
- smaller momentum at beginning
  - set $\alpha = 0.5$ in the first 250 iterations and $\alpha = 0.8$ afterwards
- fix the random seed
  - use `numpy.random.seed` to set a constant seed before sampling $\{y_i\}_{i=1}^{N}$;
  - This can help you repeat the results and debug your program.

# Use matrix operations

- Loops in Python are slow; so avoid loops over the data points
- Try to implement with matrix operations
- Example:

$$\nabla_i \stackrel{\text{def}}{=} \frac{\partial D(P,Q)}{\partial y_i} = 4 \sum_j (P_{ij} - Q_{ij})\, q_{ij}\, (y_i - y_j)$$

  - nested loops over $i$ and $j$ are slow
  - A significantly faster version
    - Let $G_{ij} = (P_{ij} - Q_{ij})\, q_{ij}$ and $s_i = \sum_j G_{ij}$

$$\nabla_i = 4 \sum_j G_{ij}(y_i - y_j) = 4 \left[ \sum_j G_{ij} y_i - \sum_j G_{ij} y_j \right] = 4 \left[ s_i y_i - \sum_j G_{ij} y_j \right]$$

    - Let $Y = [y_1, \ldots, y_N]^T$ and $S = \text{diag}(s) = \begin{bmatrix} s_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & s_N \end{bmatrix}$
    - `G = (P - Q) * q`
    - `S = np.diag(np.sum(G, axis=1))`
    - $\nabla = 4$ `* (S - G) @ Y`

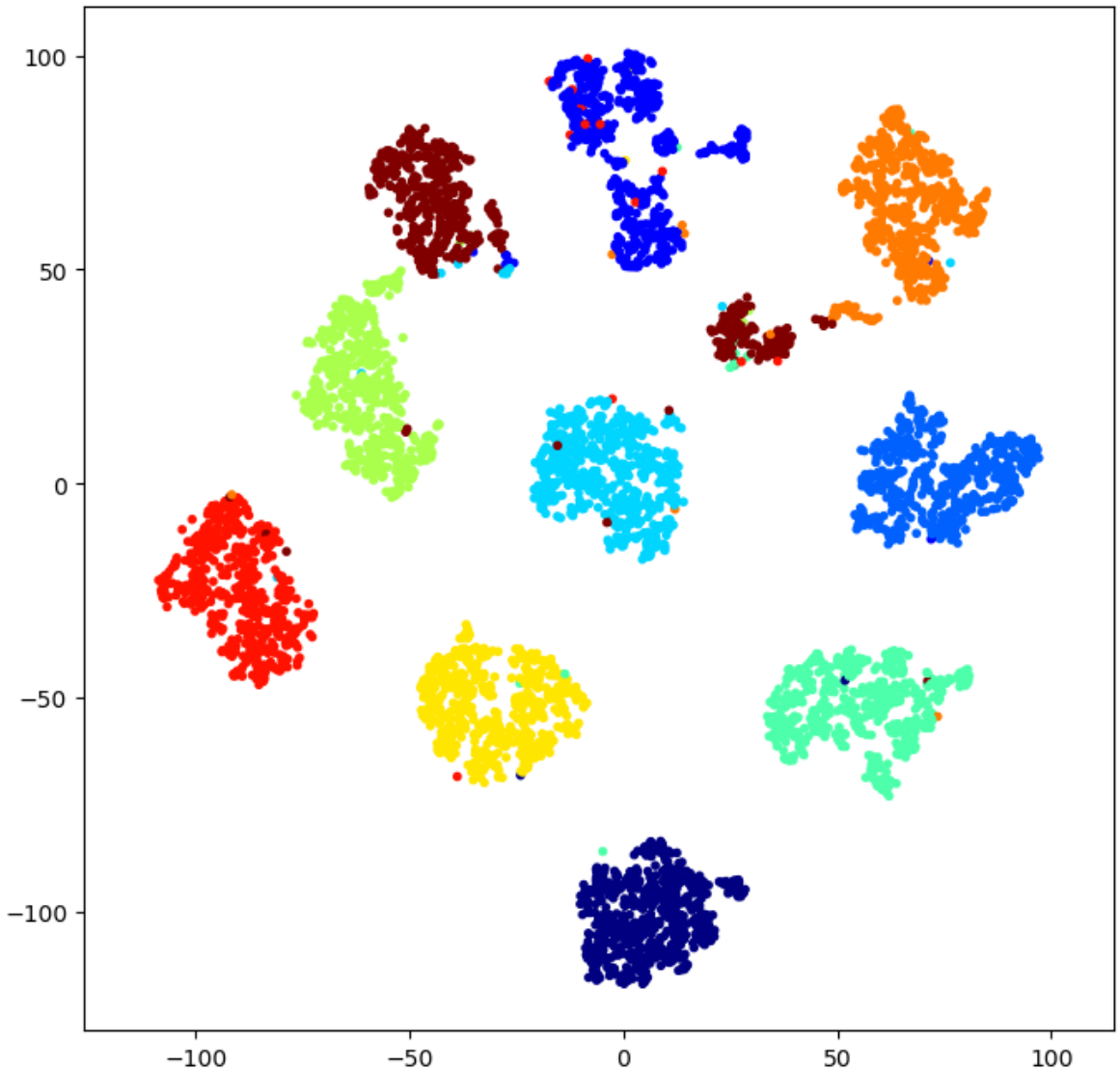
# About the demo time

- You must run the t-SNE iterations during the demo
- It is your responsibility to finish the running in time
- The whole demo has 15 minutes, including PCA, ISOMAP and t-SNE
- For t-SNE, your program should print the progress (i.e. the number of iterations)
- If the TA feels that your program is much longer than the demo time
  - the TA will fail your demo
  - and you must reschedule a time before the deadline
- So try to make your program efficient
- Otherwise you have to use fewer iterations which lead to a worse visualization; that can also cause failure

Implementation - Part 3:

- Implement the `tSNE` algorithm for dimensionality reduction.
- Apply your t-SNE method to the `OPTDIGITS` data set. Visualize the mapped points using scatter plot. Visualizing `SWISS-ROLL` is optional.
- You can use only `numpy` and `matplotlib`. You cannot use other packages or code that contains t-SNE.

# t-SNE visualization for OPTDIGITS



# Extra challenges

This is an optional part and not needed for passing the demo.

- Try to speed up the $k$NN graph construction. You may try `sklearn.neighbors.NearestNeighbors`. Then compare its running time with your own implementation.
- In t-SNE, you may record $Y$ after each iteration. Then you can visualize the sequence of mapped points as an video, with each scatter plot as a frame.
- Consider an even larger data set, for example the MNIST data set. You may find the original data set in http://yann.lecun.com/exdb/mnist/. Or you can download the .csv files from https://www.python-course.eu/neural_network_mnist.php. For such a large data set, you cannot store the $N \times N$ dense matrices in memory. Think about how to overcome this scalability problem.

# A remark on Pylint

- We often use capital letters for matrices in mathematical notation
- But Pylint hates capital letters as variable names
- So you must rename the variables to satisfy Pylint
- Write down (e.g. by comments) the correspondence between the program variables and their corresponding mathematical notations; This is important for teamwork

# Demonstration

- Run PCA to visualize the `SWISS-ROLL` data set
- Run PCA to visualize the `OPTDIGITS` data set
- Run Isomap to visualize the `SWISS-ROLL` data set
- Run Isomap to visualize the `OPTDIGITS` data set
- Run t-SNE to visualize the `OPTDIGITS` data set

Your visualizations will be compared with the answers. You will get pass if all of them are close to the answers.