we only compare
t-SNE with: (1) Sammon mapping, (2) **Isomap**, and (3) **LLE**. In the supporting material, we also compare t-SNE with: (4) CCA, (5) **SNE**, (6) MVU, and (7) Laplacian Eigenmaps.

4.1 Data Sets

The five data sets we employed in our experiments are: (1) the MNIST data set, (2) the Olivetti faces data set, (3) the COIL-20 data set, (4) the word-features data set, and (5) the Netflix data set.

Simple Algorithm

Algorithm 1: Simple version of t-Distributed Stochastic Neighbor Embedding.

Data: data set $\mathcal{X} = \{x_1, x_2, \ldots, x_n\}$,
cost function parameters: perplexity *Perp*,
optimization parameters: number of iterations T, learning rate η, momentum $\alpha(t)$.
Result: low-dimensional data representation $\mathcal{Y}^{(T)} = \{y_1, y_2, \ldots, y_n\}$.
begin
  compute pairwise affinities $p_{j|i}$ with perplexity *Perp* (using Equation 1)
  set $p_{ij} = \frac{p_{j|i} + p_{i|j}}{2n}$
  sample initial solution $\mathcal{Y}^{(0)} = \{y_1, y_2, \ldots, y_n\}$ from $\mathcal{N}(0, 10^{-4}I)$
  for $t=1$ **to** T **do**
    compute low-dimensional affinities q_{ij} (using Equation 4)
    compute gradient $\frac{\delta C}{\delta \mathcal{Y}}$ (using Equation 5)
    set $\mathcal{Y}^{(t)} = \mathcal{Y}^{(t-1)} + \eta \frac{\delta C}{\delta \mathcal{Y}} + \alpha(t)\left(\mathcal{Y}^{(t-1)} - \mathcal{Y}^{(t-2)}\right)$
  end
end

```
def equation4(params, samples, dimensions):
    """
    :param params: array, shape (n_params,)
        Unraveled embedding.
    :param samples: [int] sample 의 갯수
   :param dimensions: [int] 공간의 차수
   :return: [float] joint probabilities Q
    """

    X = params.reshape(samples, dimensions)

    from scipy.spatial.distance import pdist
    import numpy as np
    org_distance = pdist(X, "sqeuclidean")
    comput_distance = org_distance ** 2.0 + 1.0
    comput_distance = np.invert(comput_distance)
    Q = comput_distance / np.sum(comput_distance)

    return Q
```

```
def equation5(P, Q, org_distance, comput_distance):
    """
    :param P: [float] equation 1 의 결과인 joint probabilities
    :param Q: [float] equation 4 의 결과인 joint probabilities
    :param org_distance: [float] equation 4 에서 구한 원거리
    :param comput_distance: [float] equation 4 에서 구한 계산된 거리
    :return: [float] the gradient of the Kullback-Leibler divergence
    """
    return  4 * np.sum((P - Q) * org_distance * comput_distance)
```

```
from __future__ import print_function
import time
import numpy as np
import pandas as pd
from sklearn.datasets import fetch_openml
from sklearn.manifold import TSNE, LocallyLinearEmbedding, Isomap
from sklearn.cross_decomposition  import CCA
%matplotlib inline
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import seaborn as sns
```