Homework# 8

0756079 陳冠聞

1. Symmetric-SNE

t-SNE 與 symmetric SNE 的差異為 low-dim similarity Q 的計算，
symmetric SNE 採用與 high-dimension 一樣的高斯分布，
t-SNE 採用與 t-student 分布，因此將 t-SNE 修改為 symmetric SNE 主要需要修改 (1) 計算 Q (2) 計算 Gradient

公式如下圖

(1) 計算 Q value

$$q_{ij} = \frac{(1+ \| y_i - y_j \|^2)^{-1}}{\sum_{k\neq l}(1+ \| y_i - y_j \|^2)^{-1}}$$

T-SNE Q value

$$q_{ij} = \frac{\exp(- \| y_i - y_j \|^2)}{\sum_{k\neq l}\exp(- \| y_l - y_k \|^2)}$$

Symmetric -SNE Q value

(2) 計算 Gradient

$$\frac{\delta C}{\delta y_i} = 4\sum_j (p_{ij} - q_{ij})(y_i - y_j)(1+ \| y_i - y_j \|^2)^{-1}$$

T-SNE Gradient

$$\frac{\partial C}{\partial y_i} = 2\sum_i (p_{ij} - q_{ij})(y_i - y_j)$$

Symmetric SNE Gradient

修改的 code 如下

(1) 計算 Q value

```
num = np.exp(-1*np.add(np.add(num, sum_Y).T, sum_Y))
```

改成

```
num = 1. / (1. + np.add(np.add(num, sum_Y).T, sum_Y))
```

(2) 計算 Gradient

```
dY[i, :] = np.sum(np.tile(PQ[:, i] * num[:, i], (no_dims, 1)).T * (Y[i, :] - Y), 0)
```

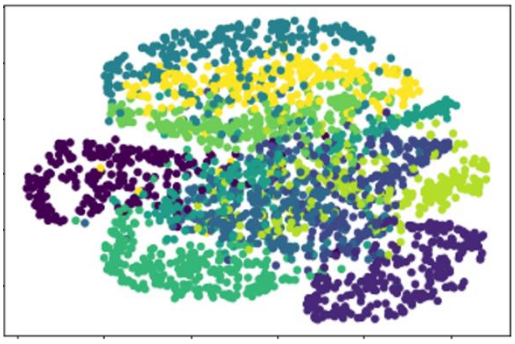
改成

```
dY[i, :] = np.sum(np.tile(PQ[:, i], (no_dims, 1)).T * (Y[i, :] - Y), 0)
```

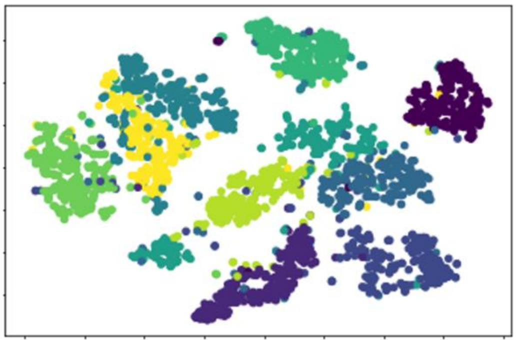
2. Visualize the embedding

使用 default parameters (initial_dims=50, perplexity=30.0)

可看出比起 Symmetric-SNE, t-SNE 的 crowded problem 改善了很多，因為 t-SNE 使用的 t 分布相較於常態分佈，對於高相似度的點在低維空間的距離較小，而對於相似度低的點在低維空間的距離則較大，因此可以減少 crowded problem 的問題。



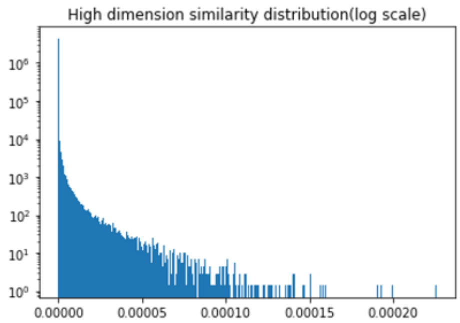
Symmetric-SNE



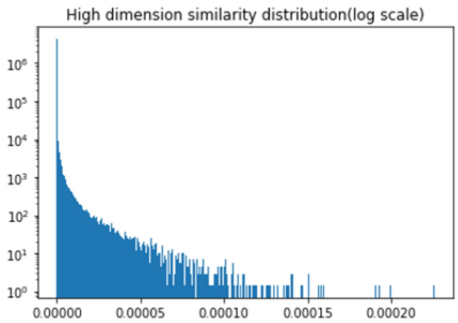
t-SNE

3. Visualize the distribution

High-dimension

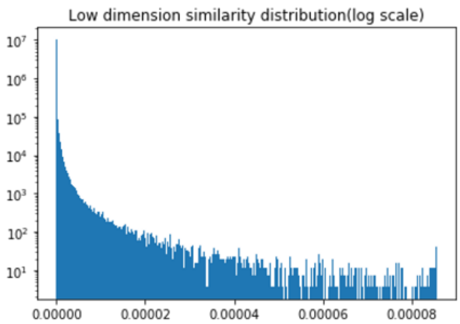


t-SNE

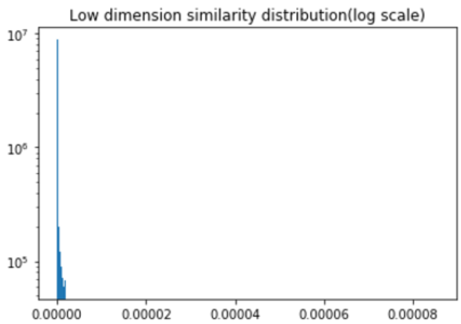


Symmetric-SNE

low-dimension



t-SNE

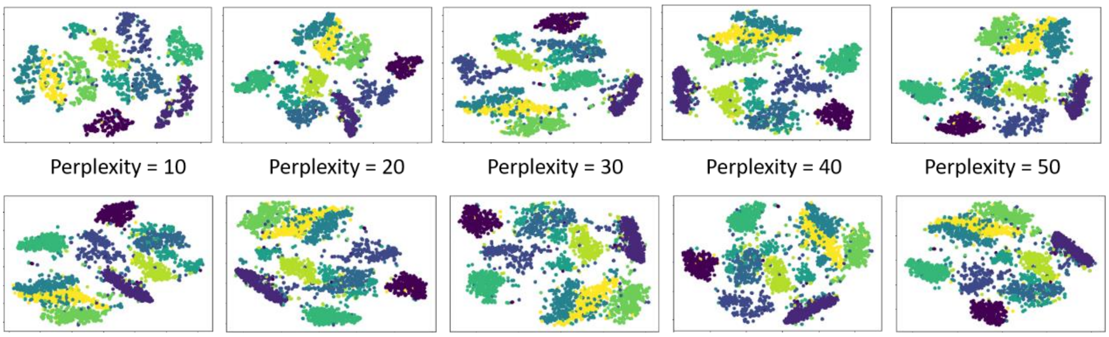


Symmetric-SNE

從上圖可以看出 t-SNE 相比 Symmetric-SNE，在 low-dimension 的 similarity distribution 的 range 較大，這是因為 t 分布比起常態分布較為寬矮，而這也是為何 t-SNE 較能減輕 crowded problem 的原因。

4. Different settings of perplexity

Perplexity 決定了 effective nearest neighbors，在本次作業中實驗了 t-SNE Perplexity 從 10 到 100 (間隔為 10)，結果如下



Perplexity = 10 Perplexity = 20 Perplexity = 30 Perplexity = 40 Perplexity = 50

Perplexity = 60 Perplexity = 70 Perplexity = 80 Perplexity = 90 Perplexity = 100

可看到當 Perplexity 逐漸變大時，會逐漸有 crowded problem，因為當依賴的鄰居變多時，很容易造成彼此分不開的情況。

5. Code Detail

只放上與原 reference 不同的程式碼，其他詳見 SNE.py

SymSNE 的 function

```python
def symSNE(X=np.array([]), no_dims=2, initial_dims=50, perplexity=30.0):
    """
        Runs t-SNE on the dataset in the NxD array X to reduce its
        dimensionality to no_dims dimensions. The syntaxis of the function is
        `Y = tsne.tsne(X, no_dims, perplexity), where X is an NxD NumPy array.
    """

    # Check inputs
    if isinstance(no_dims, float):
        print("Error: array X should have type float.")
        return -1
    if round(no_dims) != no_dims:
        print("Error: number of dimensions should be an integer.")
        return -1

    # Initialize variables
    X = pca(X, initial_dims).real
    (n, d) = X.shape
    max_iter = 1000
    initial_momentum = 0.5
    final_momentum = 0.8
    eta = 500
    min_gain = 0.01
    Y = np.random.randn(n, no_dims)
    dY = np.zeros((n, no_dims))
    iY = np.zeros((n, no_dims))
    gains = np.ones((n, no_dims))

    # Compute P-values
    P = x2p(X, 1e-5, perplexity)
    P = P + np.transpose(P)
    P = P / np.sum(P)

    P_copy = np.copy(P)

    P = P * 4.                                  # early exaggeration
    P = np.maximum(P, 1e-12)
    pbar = tqdm(total = max_iter)
    for iter in range(max_iter):

        # Compute pairwise affinities (### MODIFY !!)
        sum_Y = np.sum(np.square(Y), axis=1) # yi^T * yi
        num = -2. * np.dot(Y, Y.T)  # -2 *  yj^T * yj
        num = np.exp(-1*np.add(np.add(num, sum_Y).T, sum_Y))
        num[range(n), range(n)] = 0. # dia set to zeros
        Q = num / np.sum(num)
        Q = np.maximum(Q, 1e-12)
        Q_copy = np.copy(Q)

        # Compute gradient
        PQ = P - Q
        for i in range(n):
            dY[i, :] = np.sum(np.tile(PQ[:, i], (no_dims, 1)).T * (Y[i, :] - Y), 0)
        if iter < 20:
            momentum = initial_momentum
        else:
            momentum = final_momentum
        gains = (gains + 0.2) * ((dY > 0.) != (iY > 0.)) +             (gains * 0.8) * ((dY > 0.) == (iY > 0.))
        gains[gains < min_gain] = min_gain
        iY = momentum * iY - eta * (gains * dY)
        Y = Y + iY
        Y = Y - np.tile(np.mean(Y, 0), (n, 1))

        # Compute current value of cost function
        C = np.sum(P * np.log(P / Q))
        pbar.set_postfix(Error=C)
        pbar.update()
#        if (iter + 1) % 10 == 0:
#            C = np.sum(P * np.log(P / Q))
#            print("Iteration %d: error is %f" % (iter + 1, C))

        # Stop lying about P-values
        if iter == 100:
            P = P / 2.

    # Return solution
    pbar.close()
    return Y, P_copy, Q_copy
```

與 t-SNE 相異處

回傳 similarity matrix 方便畫 distribution

畫 t-SNE high-dim 及 low-dim 的 similarity distribution

```
X = np.loadtxt("mnist2500_X.txt")
labels = np.loadtxt("mnist2500_labels.txt")
perplexity_range = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
Y, P, Q = symSNE(X, 2, initial_dims=50, perplexity=30.0)
pylab.scatter(Y[:, 0], Y[:, 1], 20, labels)
pylab.show()


high_distribution = P.reshape((2500*2500))
low_distribution = Q.reshape((2500*2500))


plt.hist(high_distribution[:], bins=1000, density=True, log=True)
plt.title("High dimension similarity distribution(log scale)")
plt.show()

plt.hist(low_distribution[:], bins=100, density=True, log=True, range=low_range)
plt.title("Low dimension similarity distribution(log scale)")
plt.show()
```

畫 sym-SNE high-dim 及 low-dim 的 similarity distribution

```
X = np.loadtxt("mnist2500_X.txt")
labels = np.loadtxt("mnist2500_labels.txt")
perplexity_range = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
Y, P, Q = tSNE(X, 2, initial_dims=50, perplexity=30.0)
pylab.scatter(Y[:, 0], Y[:, 1], 20, labels)
pylab.show()


high_distribution = P.reshape((2500*2500))
low_distribution = Q.reshape((2500*2500))

low_range = (low_distribution.min(), low_distribution.max())

plt.hist(high_distribution[:], bins=1000, density=True, log=True)
plt.title("High dimension similarity distribution(log scale)")
plt.show()

plt.hist(low_distribution[:], bins=1000, density=True, log=True, range=low_range)
plt.title("Low dimension similarity distribution(log scale)")
plt.show()
```

畫不同 perplexity 的 t-SNE

```
X = np.loadtxt("mnist2500_X.txt")
labels = np.loadtxt("mnist2500_labels.txt")
perplexity_range = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
for perplexity in perplexity_range:
    print("perplexity=", perplexity)
    Y = tSNE(X, 2, 50, perplexity=perplexity)
    pylab.scatter(Y[:, 0], Y[:, 1], 20, labels)
    pylab.show()
```