2018 ML HW8 t-SNE

0756110 李東霖

This document made by HackMD, you can view here https://hackmd.io/s/HJAAYE8ZV
(https://hackmd.io/s/HJAAYE8ZV)

Table of Contents

Modify the code for Symmetric SNE

```
def tsne(
    X=np.array([]),
    no_dims=2,
    initial_dims=50,
    perplexity=30.0,
    SNE=False
):
    """
        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 not X.dtype == 'float':
        raise AttributeError("array X should have type float")
    if round(no_dims) != no_dims:
        raise AttributeError("number of dims must be an integer")
```

```
    # initialize variables

    # reduce X to initial_dims dimensions
    # because eigen vector is complex, just get real part
    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))
    C = []
    Q = np.zeros((n, n))

    if SNE:
        # use same random initial values
        Y_SNE = Y.copy()
        dY_SNE = np.zeros((n, no_dims))
        iY_SNE = np.zeros((n, no_dims))
        gains_SNE = np.ones((n, no_dims))
        C_SNE = []
        Q_SNE = np.zeros((n, n))

    # compute P-values
    P = x2p(X, 1e-5, perplexity)
    P = P + np.transpose(P)
    P = P / np.sum(P)
    P = P * 4.
    P = np.maximum(P, 1e-12)

    # run iterations
    for iter in range(max_iter):

        # compute pairwise affinities Q
        sum_Y = np.sum(np.square(Y), 1)
        num = -2. * np.dot(Y, Y.T)
        num = 1. / (1. + np.add(np.add(num, sum_Y).T, sum_Y))
        num[range(n), range(n)] = 0.
        Q = num / np.sum(num)
        Q = np.maximum(Q, 1e-12)
        if SNE:
            sum_Y_SNE = np.sum(np.square(Y_SNE), 1)
            num_SNE = -2. * np.dot(Y_SNE, Y_SNE.T)
            num_SNE = np.add(np.add(num_SNE, sum_Y_SNE).T, sum_Y_SNE)
            num_SNE = np.exp(-1*num_SNE)
            num_SNE[range(n), range(n)] = 0.
            Q_SNE = num_SNE / np.sum(num_SNE)
            Q_SNE = np.maximum(Q_SNE, 1e-12)
```

```
        # compute gradient
        PQ = P - Q
        if SNE:
            PQ_SNE = P - Q_SNE
        for i in range(n):
            dY[i, :] = np.sum(
                np.tile(
                    PQ[:, i] * num[:, i],
                    (no_dims, 1)
                ).T * (Y[i, :] - Y),
                0
            )
            if SNE:
                dY_SNE[i, :] = np.sum(
                    np.tile(
                        PQ_SNE[:,i],
                        (no_dims, 1)
                    ).T * (Y_SNE[i, :] - Y_SNE),
                    0
                )

        # perform the update
        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))

        if SNE:
            Y_SNE, iY_SNE, gains_SNE = __tsne_gradient_descent(
                Y_SNE,
                dY_SNE,
                iY_SNE,
                gains_SNE,
                min_gain,
                momentum,
                eta
            )

        # compute current value of cost function
        # use KL divergense
        if (iter + 1) % 10 == 0:
            C += [np.sum(P * np.log(P / Q))]
            if SNE:
                C_SNE += [np.sum(P * np.log(P / Q_SNE))]

        # stop lying about P-values
        if iter == 100:
```

```
            P = P / 4.

    # return solution
    if not SNE:
        return Y, P, Q, C
    else:
        return (Y, Y_SNE), P, (Q, Q_SNE), (C, C_SNE)
```

將 symmetric SNE 一起實作進去，並且在回傳時一起丟出

接下來，逐步講解程式碼
首先輸入資料預處理，對 `X` 做 PCA 降至 `initial_dims` 維度

```
def pca(X=np.array([]), no_dims=50):
    """
        Runs PCA on the NxD array X in order to
        reduce its dimensionality to no_dims dimensions.
    """

    # get n and d from shape
    (n, d) = X.shape

    # become X as centered data
    # use mean get 1xd array from n datas
    # and then expend to n*d array
    # sub with X
    X = X - np.tile(np.mean(X, axis=0), (n,1))

    # maximize W^T S W, solve eigen problem
    # S is covariance matrix
    (l, M) = np.linalg.eig(np.matmul(X.T, X))

    # use max no_dims principal compoment to reduce dimensions
    Y = np.matmul(X, M[:, 0:no_dims])
    return Y
```

先將 `X` 減去其平均，使用 `np.dot(X.T, X)` 做出 covariance matrix
並解出其矩陣的 eigen vector，最後使用前面 `no_dims` 個 eigen vector 將 `X` 降維
降維完成的 `Y` 回傳回去

`pca(X, initial_dims).real`
因為 eigen vector 是複數，因此只取實數

接下來初始化必要參數

- `max_iter = 1000`
    - 設定要跑幾個 iteration，並一定會跑到這個迭代數
    - 也因此沒有設定特別的收斂條件

底下參數都是在設定 梯度 如何更新參數的設定值

- `initial_momentum = 0.5`
- `final_momentum = 0.8`
- `eta = 500`
- `min_gain = 0.01`

準備需梯度更新的參數們

- `Y = np.random.randn(n, no_dims)`
    - 用隨機的方式初始化 `Y`
    - 也因為如此，降維結果會不一樣
    - 因為要比較 t-SNE 與 symmetric SNE，讓初始化的 `Y` 相同
    - `Y_SNE = Y.copy()`
- `dY = np.zeros((n, no_dims))`
    - 藉由 $\frac{\delta C}{\delta y_i}$ 求得的梯度
- `iY = np.zeros((n, no_dims))`
- `gains = np.ones((n, no_dims))`

如果需要計算 symmetric SNE 也準備一份參數

```
if SNE:
    # use same random initial values
    Y_SNE = Y.copy()
    dY_SNE = np.zeros((n, no_dims))
    iY_SNE = np.zeros((n, no_dims))
    gains_SNE = np.ones((n, no_dims))
```

`P = x2p(X, 1e-5, perplexity)` 依照 perplexity ，計算高維資料的 pairwise similarities P

$$p_{j|i} = \frac{exp(-||x_i - x_j||^2/(2\sigma_i^2))}{\sum_{k\neq i} exp(-||x_i - x_k||^2/(2\sigma_i^2))}$$

```
def x2p(X=np.array([]), tol=1e-5, perplexity=30.0):
    """
        Performs a binary search to get P-values in such a way that each
        conditional Gaussian has the same perplexity.
    """

    # initialize some variables

    # compute pairwise distances
    (n, d) = X.shape
    sum_X = np.sum(np.square(X), 1)
    D = np.add(np.add(-2 * np.dot(X, X.T), sum_X).T, sum_X)


    P = np.zeros((n, n))
    beta = np.ones((n, 1))
    logU = np.log(perplexity)

    # loop over all datapoints
    for i in range(n):

        # compute the Gaussian kernel and entropy for the current precision
        betamin = -np.inf
        betamax = np.inf
        Di = D[i, np.concatenate((np.r_[0:i], np.r_[i+1:n]))]
        (H, thisP) = Hbeta(Di, beta[i])

        # evaluate whether the perplexity is within tolerance
        Hdiff = H - logU
        tries = 0
        while np.abs(Hdiff) > tol and tries < 50:

            # if not, increase or decrease precision
            if Hdiff > 0:
                betamin = beta[i].copy()
                if betamax == np.inf or betamax == -np.inf:
                    beta[i] = beta[i] * 2.
                else:
                    beta[i] = (beta[i] + betamax) / 2.
            else:
                betamax = beta[i].copy()
                if betamin == np.inf or betamin == -np.inf:
                    beta[i] = beta[i] / 2.
                else:
                    beta[i] = (beta[i] + betamin) / 2.

            # recompute the value
            (H, thisP) = Hbeta(Di, beta[i])
            Hdiff = H - logU
            tries += 1

        # set the final row of P
        P[i, np.concatenate((np.r_[0:i], np.r_[i+1:n]))] = thisP
```

```
    # return final P-matrix
    return P
```

因為在高維資料計算 pairwise similarities 時，有一個 σ_i 需要調整
SNE 設定一個超參數叫 perplexity 去決定 σ_i

$$\text{perplexity}(P_i) = 2^{H(P_i)}, H(P_i) = -\sum_j p_{j|i} log_2 p_{j|i}$$

可以發現整個公式跟 Shannon entropy 有關係，同時發現以下關係

- perplexity 越大，entropy 越大，σ_i 越大
- perplexity 越小，entropy 越小，σ_i 越小

但最後要如何藉由 perplexity 找到 σ_i，則是使用二元搜尋
觀察當前算出的 perplexity 與設定的 perplexity 差距方向
並決定增加 σ 或 σ ，中止條件為小於容忍度或執行到特定迭代次數

在計算當前的 perplexity ，參考實作的程式碼使用了一點 trick

```
def Hbeta(D=np.array([]), beta=1.0):
    """
        Compute the perplexity and the P-row for a specific value of the
        precision of a Gaussian distribution.
    """

    # compute P-row and corresponding perplexity
    P = np.exp(-D.copy() * beta)
    sumP = sum(P)
    H = np.log(sumP) + beta * np.sum(D * P) / sumP
    P = P / sumP

    return H, P
```

以下是推導

$$D_{j|i} = -||x_i - x_j||^2, \text{beta} = \frac{1}{2\sigma_i^2}, p_{j|i} = \frac{exp(-D_{j|i} * beta)}{\sum_j exp(-D_{j|i} * beta)}$$

$$H(P_i) = -\sum_j p_{j|i} log_2 p_{j|i} = \sum_j \frac{exp(-D_{j|i} * beta)}{\sum_j D_{j|i} * beta} log_2 \frac{\sum_j exp(-D_{j|i} * beta)}{exp(-D_{j|i} * beta)}$$

$$= \sum_j \frac{exp(-D_{j|i} * beta)}{\sum_j exp(-D_{j|i} * beta)} (log(\sum_j exp(-D_{j|i} * beta)) - log(exp(-D_{j|i} * beta)))$$

$$= \sum_j \frac{exp(-D_{j|i} * beta) log(\sum_j D_{j|i} * beta)}{\sum_j exp(-D_{j|i} * beta)} + \frac{exp(-D_{j|i} * beta) D_{j|i} * beta)}{\sum_j exp(-D_{j|i} * beta)}$$

$$= log(sumP) + \frac{\sum_j p_j D_{j|i} beta}{sumP}, sumP = \sum_j p_j, p_j = exp(-D_{j|i} * beta)$$

因此 `H` 代表 Shannon entropy， `P` 代表得到的高維資料的 pairwise similarities

到此以得到 P ，但這個 P 會導致 Asymmetric KL divergence
因為在不同 row 的 σ 是不同的
這邊將 P 轉換成 $p_{j|i} = \frac{exp(-||x_i - x_j||^2/(2\sigma^2))}{\sum_{k \neq l} exp(-||x_l - x_k||^2/(2\sigma^2))}$ 藉由以下程式

```
P = P + np.transpose(P)
P = P / np.sum(P)
P = P * 4.
P = np.maximum(P, 1e-12)
```

接下來要開始進行學習，也就是利用梯度去更新 Y
而目標就是讓 cost C 最小，換句話說就是高維 pairwise similarities P 與 低維 pairwise similarities Q 分佈盡可能相似

$$C = \sum_i KL(P_i || Q_i) = \sum_i \sum_j p_{j|i} log \frac{p_{j|i}}{q_{j|i}}$$

先計算 Q
這邊也是 t-SNE 與 symmetric 關鍵不同之一
在低維空間使用的 分佈 公式不同

- t-SNE : $q_{ij} = \frac{(1+||y_i - y_j||^2)^{-1}}{\sum_{k \neq l}(1+||y_l - y_j||^2)^{-1}}$
- symmetric SNE : $q_{ij} = \frac{exp(-||y_i - y_j||^2)}{\sum_{k \neq l} exp(-||y_l - y_k||^2)}$

```
# compute pairwise affinities Q
sum_Y = np.sum(np.square(Y), 1)
num = -2. * np.dot(Y, Y.T)
num = 1. / (1. + np.add(np.add(num, sum_Y).T, sum_Y))
num[range(n), range(n)] = 0.
Q = num / np.sum(num)
Q = np.maximum(Q, 1e-12)
if SNE:
    sum_Y_SNE = np.sum(np.square(Y_SNE), 1)
    num_SNE = -2. * np.dot(Y_SNE, Y_SNE.T)
    num_SNE = np.add(np.add(num_SNE, sum_Y_SNE).T, sum_Y_SNE)
    num_SNE = np.exp(-1*num_SNE)
    num_SNE[range(n), range(n)] = 0.
    Q_SNE = num_SNE / np.sum(num_SNE)
    Q_SNE = np.maximum(Q_SNE, 1e-12)
```

之後計算梯度 gradient
這邊 t-SNE 與 symmetric SNE

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

```
# compute gradient
PQ = P - Q
if SNE:
    PQ_SNE = P - Q_SNE
for i in range(n):
    dY[i, :] = np.sum(
        np.tile(
            PQ[:, i] * num[:, i],
            (no_dims, 1)
        ).T * (Y[i, :] - Y),
        0
    )
    if SNE:
        dY_SNE[i, :] = np.sum(
            np.tile(
                PQ_SNE[:,i],
                (no_dims, 1)
            ).T * (Y_SNE[i, :] - Y_SNE),
            0
        )
```

取得梯度 gradient 之後，接下來就是更新參數 Y

```
    # perform the update
    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))

    if SNE:
        Y_SNE, iY_SNE, gains_SNE = __tsne_gradient_descent(
            Y_SNE,
            dY_SNE,
            iY_SNE,
            gains_SNE,
            min_gain,
            momentum,
            eta
        )
```

`__tsne_gradient_descent` 是將更新 t-SNE 的部份複製一樣過程更新給 symmetric SNE

接下來，每 10 個 iterations
將 cost C 記錄下來，方便之後比較

```
    # compute current value of cost function
    # use KL divergense
    if (iter + 1) % 10 == 0:
        C += [np.sum(P * np.log(P / Q))]
        if SNE:
            C_SNE += [np.sum(P * np.log(P / Q_SNE))]
```

當跑完設定的 `max_iter` 次數，將更新完的 Y 與 P、Q、C 都回傳

```
    # return solution
    if not SNE:
        return Y, P, Q, C
    else:
        return (Y, Y_SNE), P, (Q, Q_SNE), (C, C_SNE)
```

t-SNE and Symmetric SNE

設定 perplexity 為 20.0 ，先用 pca 到 50 維最後用 t-SNE 與 symmetric SNE 到 2 維

```
Ys, P, Qs, Cs = tsne(X, 2, 50, 20.0, SNE=True)
```

並使用下列程式視覺化

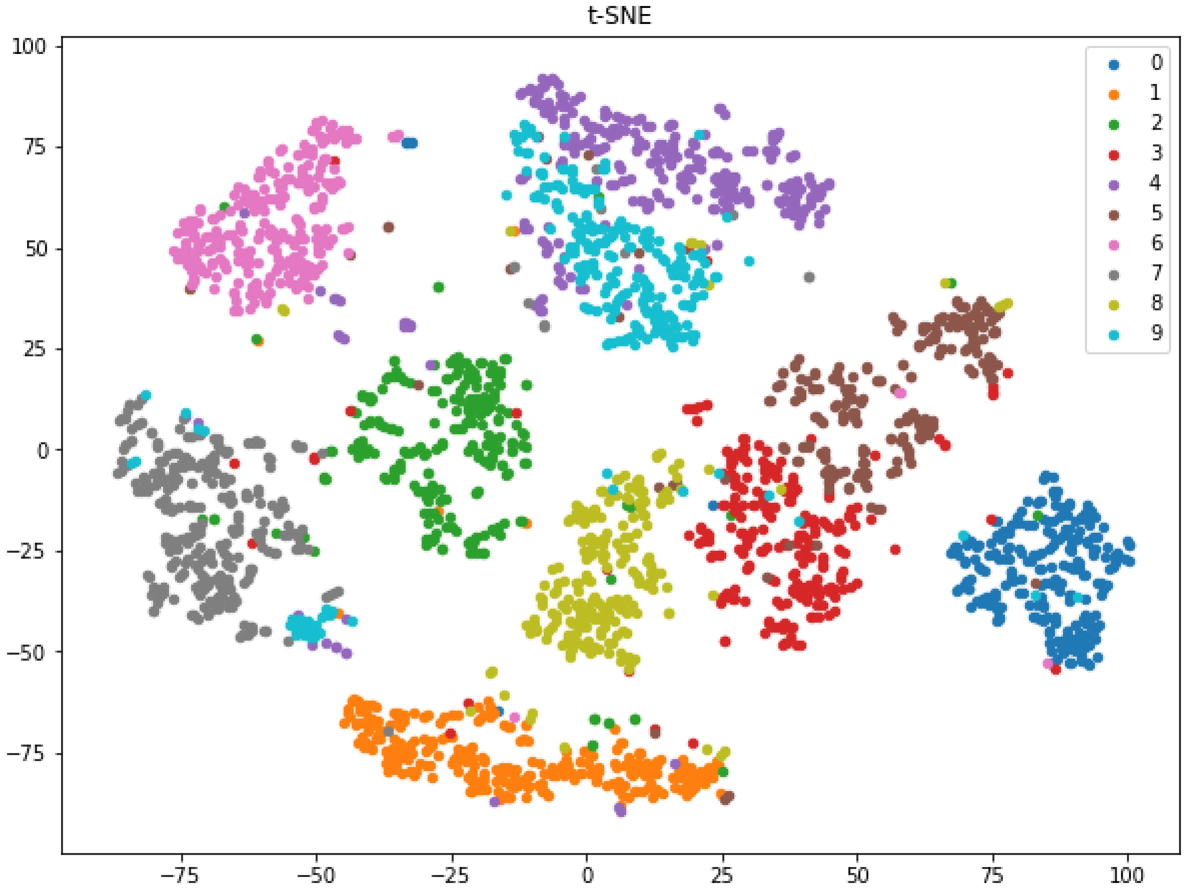
```
def showByClustering(Y, labels, size=20, title=''):
    if not isinstance(Y, list):
        Y = [Y]
    if not isinstance(title, list):
        title = [title]
    if len(Y) != len(title):
        raise AttributeError('number is different')

    n = len(Y)

```

```
    plt.figure(figsize=(10*n,7.5))
    for i in range(n):
        plt.subplot(1,n,i+1)
        plt.title(title[i])
        y = Y[i]
        for l in np.unique(labels):
            plt.scatter(
                y[labels==l,0],
                y[labels==l,1],
                s=size,
                label=str(int(l))
            )
        plt.legend()
    plt.show()
def showCostCurve(Cs, labels=None, title=''):
    plt.figure(figsize=(10,7.5))
    plt.title(title)
    if type(labels) == type(None):
        labels = ['']*len(Cs)

    for idx, C in enumerate(Cs):
        plt.plot(C, label=labels[idx])

    plt.xlabel('Iterations per 10')
    plt.ylabel('KL divergence (Cost)')
    plt.legend()
    plt.show()
```

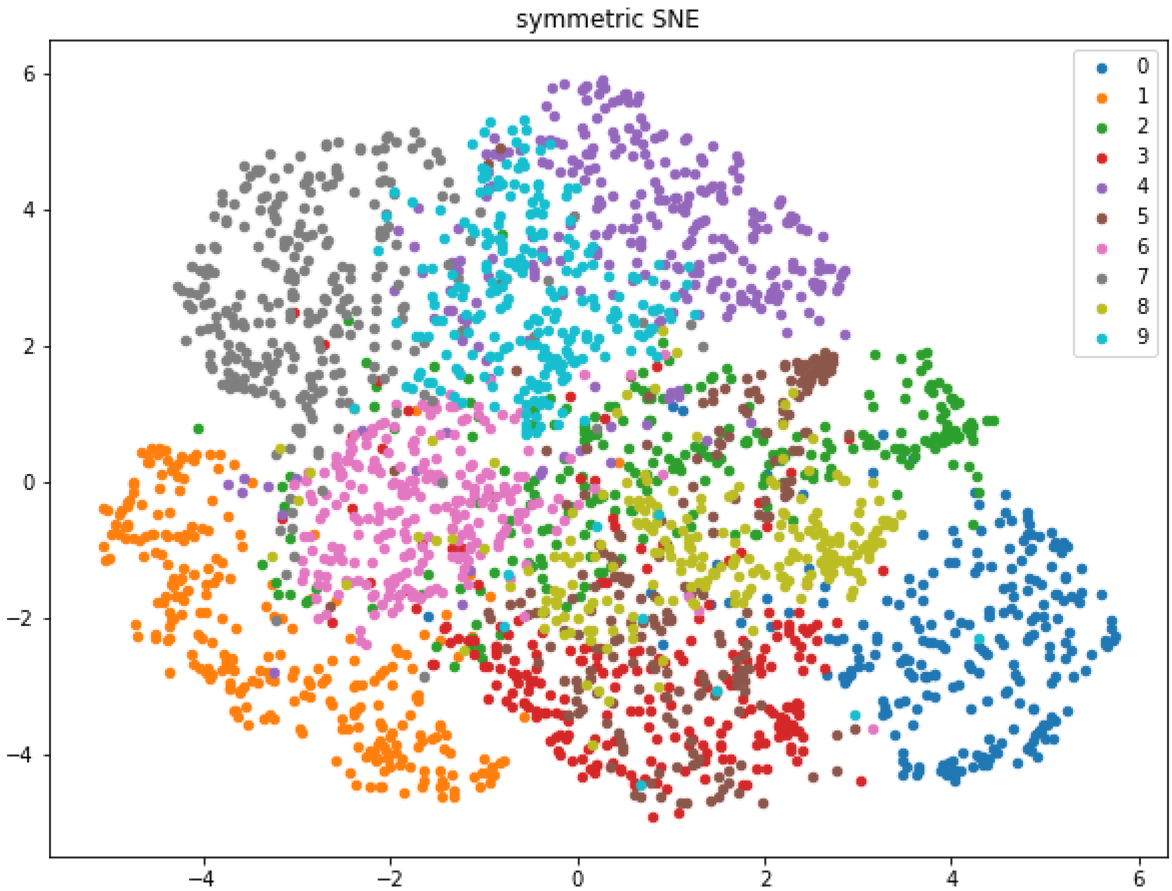
這邊直接使用散點圖將降維完的空間呈現出來，另外為了看出分的好壞與差異，將 labels 標記出來
最後希望了解優化過程的 cost 變化，因此也講 cost 經由線條圖呈現

visualization

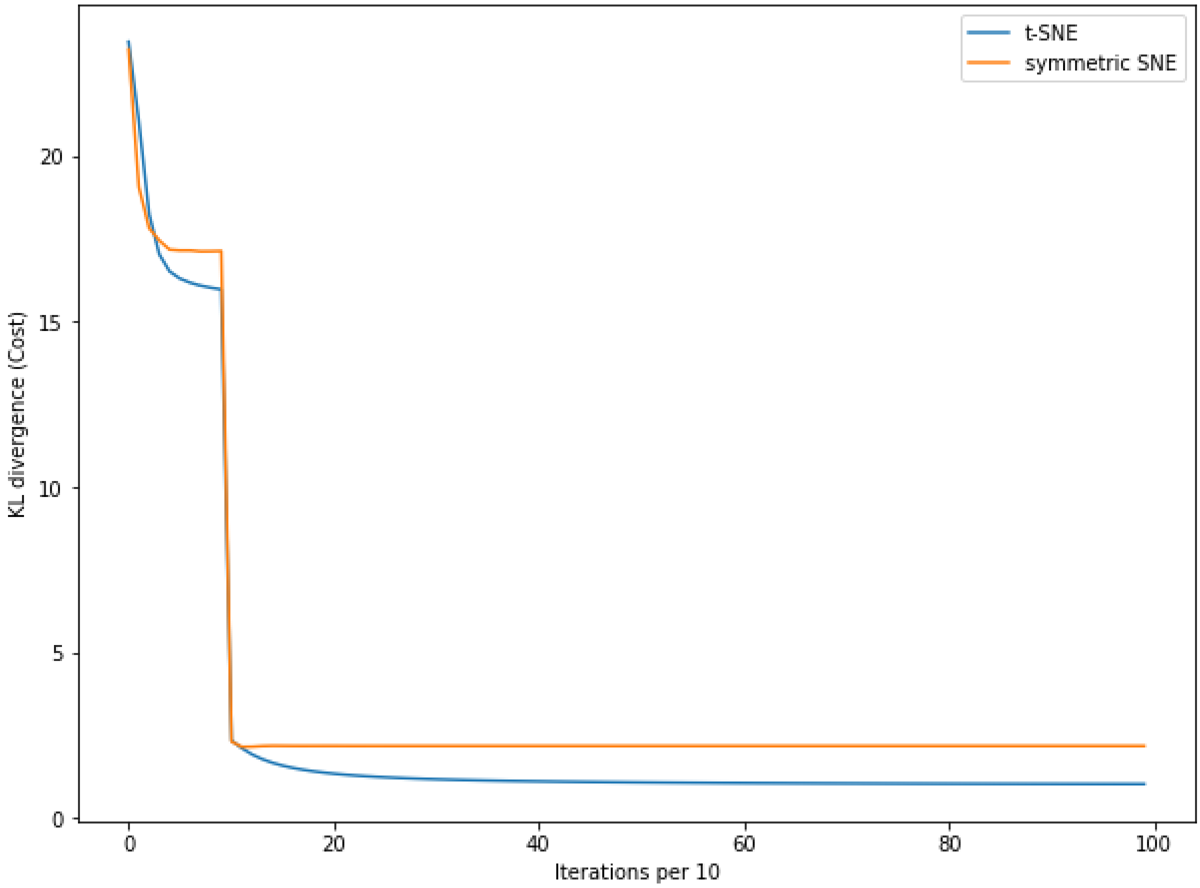
- t-SNE



- symmetric SNE



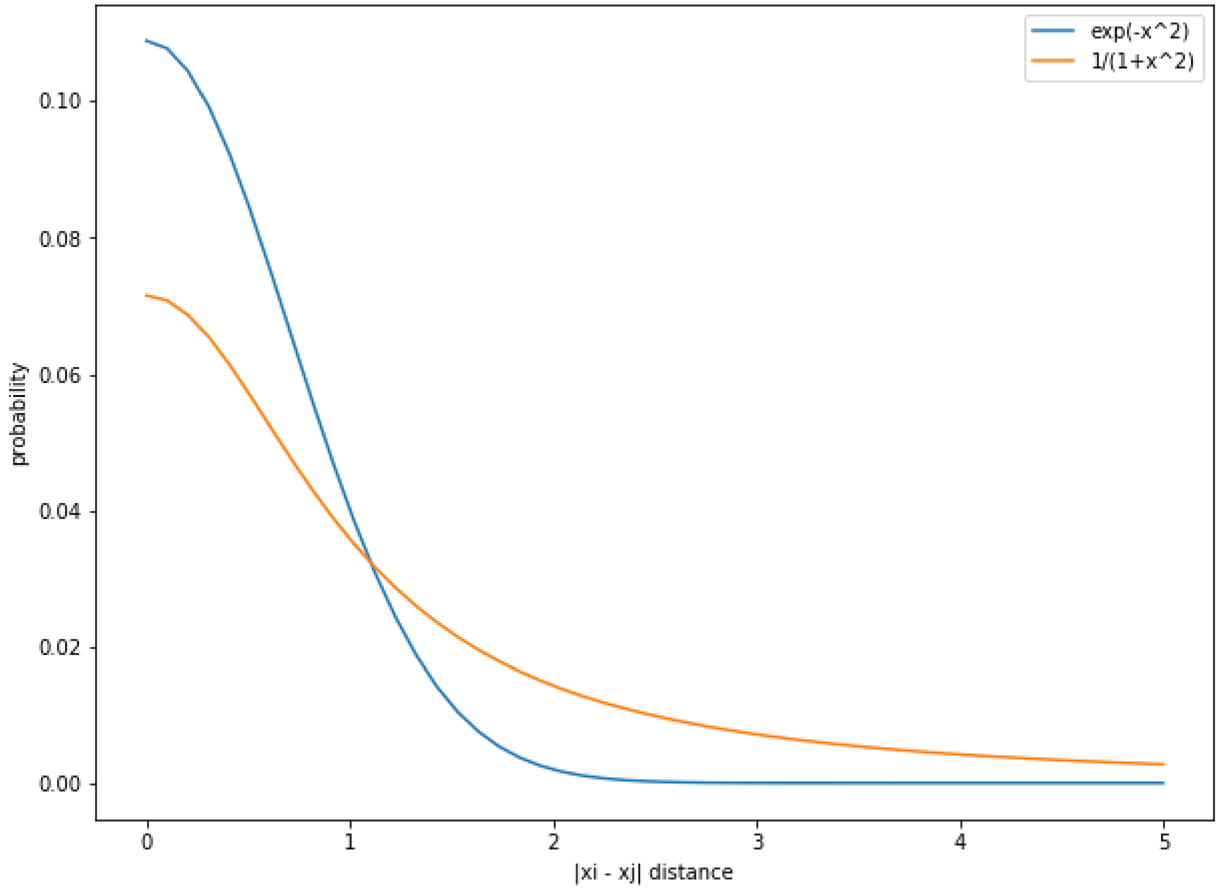
- cost curve with both



my observation and discuss

經由著色後的分群，可以看到兩個方法降維都不錯
可以盡可能把相似的靠近，並且不相似的分開

另外也可以看到 t-SNE 很好的解決了 symmetric SNE 會遇到的擁擠問題
雖然 symmetric SNE 的確可以把相遇的拉近，但無法把不相似盡可能的拉遠
導致如果沒有 labels 進行著色，根本不知道降維的好壞



但 t-SNE 因為 低維空間分佈 較 SNE分佈 來的長尾

- t-SNE
  - $(1 + ||y_i - y_j||^2)^{-1}$
- symmetric SNE
  - $exp(-||y_i - y_j||^2)$

因此在與高維空間相同的 p_{ij} 之下

- 機率高的點(越相似的點)，在 T分佈 會比 常態分佈 更近
- 機率低的點(越不相似的)，在 T分佈 會比 常態分佈 更遠

另外觀察 cost curve ，可以發現有一個非常明顯的斷層
那是因為在優化過程一開始有誇大 P 分佈，導致 cost 異常的大

之後有將 P 變回來，因此 cost curve 有一個明顯斷層

這邊也可以看到 t-SNE 優於 symmetric SNE 的另一個地方
就是在相同 iteration 下，t-SNE 的 cost 大多都優於 symmetric SNE

而且到最終的 cost 也可以來得比 symmetric SNE 低

這邊猜測是 T分佈 可以保留距離很遠的機率
因此異常值也可以有貢獻，並且發揮作用

Distribution of pairwise similarities

```
def __mybins(X, bins=10, ds=None):
    if type(ds) == type(None):
        m = np.min(X)
        d = (np.max(X) - np.min(X))/bins
        ds = np.array([d*(i+1) for i in range(bins)]) + m
    else:
        bins = len(ds)
    bin_count_t = np.zeros(bins)
    bin_count = np.zeros(bins)
    for i,d in enumerate(ds):
        bin_count_t[i] = np.count_nonzero(X < d)
    bin_count[0] = bin_count_t[0]
    for i in range(bins-1):
        bin_count[i+1] = bin_count_t[i+1] - bin_count_t[i]

    return bin_count, ds

def showPairwiseSimilarities(dis, labels, title='', rn=1):
    if not isinstance(dis, list):
        dis = [dis]
    if not isinstance(title, list):
        title = [title]
    if len(dis) != len(title):
        raise AttributeError('number is different')

    n = len(dis)
    sort_idx = np.concatenate(
        [np.where(labels==l)[0] for l in np.unique(labels)]
    )

    plt.figure(figsize=(10*n,7.5))

    dis = [np.log(d[:, sort_idx][sort_idx, :]) for d in dis]
    all_min = min([np.min(d) for d in dis])
    all_max = max([np.max(d) for d in dis])


    rnd_idx = np.random.randint(0, labels.shape[0], size=rn*3)

    for i in range(n):
        plt.subplot(1,n,i+1)
        plt.title(title[i])
        im = plt.imshow(dis[i], cmap='gray',
                        vmin=all_min, vmax=all_max)
        plt.colorbar(im)

    plt.show()
```

```
    bin_nu = 10
    bin_w = 0.3

    for i in range(rn):
        plt.figure(figsize=(10*n,7.5))
        for j in range(n):
            plt.subplot(1,n,j+1)
            idx = rnd_idx[i*rn + j]
            plt.title('{} idx distribution'.format(idx))
            tmp_idx = np.concatenate(
                [np.r_[:idx], np.r_[idx+1:labels.shape[0]]]
            )

            _, bin_x = __mybins(dis[0][idx, tmp_idx], bins=bin_nu)
            bin_x_start = np.arange(bin_nu) - ((bin_w*n)/2.0)

            for k in range(n):
                bin_y, _ = __mybins(dis[k][idx, tmp_idx], ds=bin_x)
                plt.bar(
                    bin_x_start + ((k+0.5)*bin_w),
                    bin_y,
                    width=bin_w,
                    label=title[k]
                )

            plt.xticks(range(bin_nu),
            ['{:.2f}'.format(x) for x in bin_x])
            plt.legend()

        plt.show()
```

首先，為了方便觀看，將 data 根據 label 重新排序
另外視覺化的目標是分佈，加上差距極大，因此取 log

```
sort_idx = np.concatenate(
    [np.where(labels==l)[0] for l in np.unique(labels)]
)
dis = [np.log(d[:, sort_idx][sort_idx, :]) for d in dis]
```

然後使用 `plt.imshow` ，將 2500x2500 的分佈當成一張圖，其中 p_{ij} 的數值則是灰階值，但是範圍不是 0 到 1
並且為了比較差異，高維與低維的 pairwise similarities 使用相同範圍去呈現

```
all_min = min([np.min(d) for d in dis])
all_max = max([np.max(d) for d in dis])
for i in range(n):
    plt.subplot(1,n,i+1)
    plt.title(title[i])
    im = plt.imshow(dis[i], cmap='gray', vmin=all_min, vmax=all_max)
    plt.colorbar(im)
plt.show()
```

另外也好奇 pairwise similarities 的數值分佈情形
t-distribution 有沒有所謂的較平滑的分佈
因此隨機取個 data point，將他們與其他所有資料產生的分佈分進 10 個桶

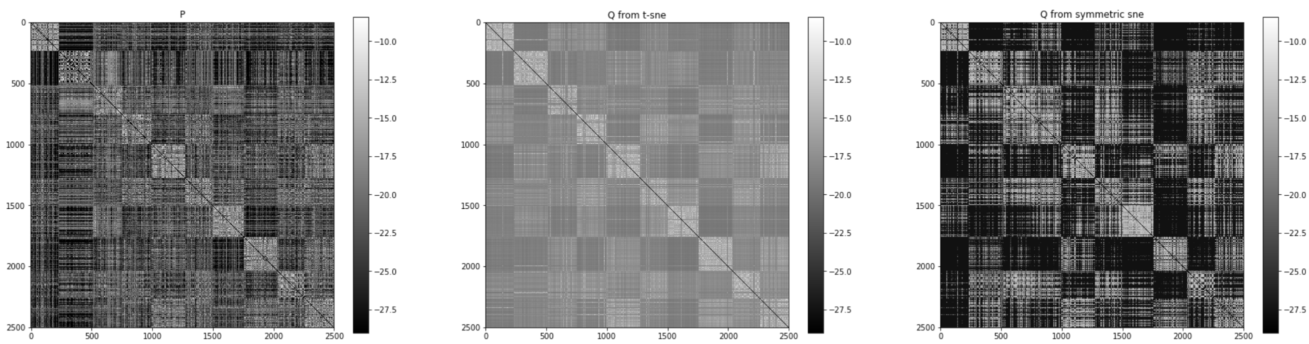
```
rnd_idx = np.random.randint(0, labels.shape[0], size=rn*3)
```

當然，為了互相比較，高維與低維使用同一種桶的範圍
最後將需要比較的分佈使用柱狀圖，全部畫到同一張圖上

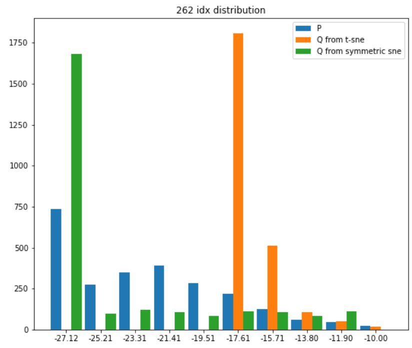
```
bin_nu = 10
_, bin_x = __mybins(dis[0][idx, tmp_idx], bins=bin_nu)
for k in range(n):
       bin_y, _ = __mybins(dis[k][idx, tmp_idx], ds=bin_x)
       plt.bar(
           bin_x_start + ((k+0.5)*bin_w),
           bin_y,
           width=bin_w,
           label=title[k]
       )
```

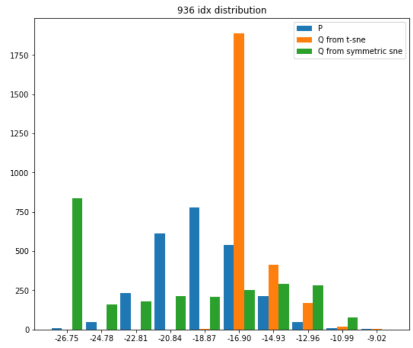
visualization

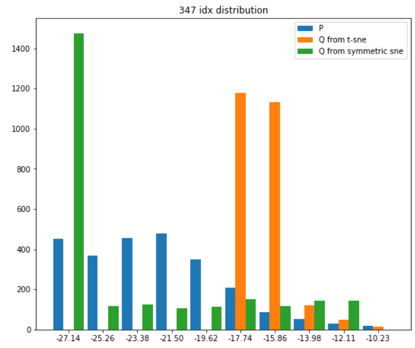
- 整體分佈
  左到右分別為 高維空間、t-SNE 的低維分佈、symmetric SNE 的低維分佈



- 隨機取點數值分佈







my observation and discuss

在整體分佈可以看到高維空間與低維空間分佈是相似的
深淺的位置幾乎都類似，這也是我們希望優化的目標
讓高維空間的機率分佈與低維空間的機率分佈一樣，去進行降維的動作

另外也可以看到 t-SNE 與 symmetric SNE 的差別
t-SNE 低維空間的分佈整體顏色偏淺，symmetric SNE 則顏色變化較劇烈

這可以從 常態分佈 與 T分佈 的不同得到該結論
可以看到在較短的距離(x 軸)之下，T 分佈都會比常態分佈機率來得低
較長的距離則是常態分佈比較低

也因此造成在低維空間分佈，t-SNE 整體數值偏淺

另外在數值分佈也可以看到這個現象
橘色長條柱是 t-SNE 的 Q ，可以看到數值大多落在較高的地方
綠色長條柱是 symmetric SNE 的 Q，可以看到數值大多落在較低的地方
而藍色是 高維空間的 P ，可以看到落在兩者之間

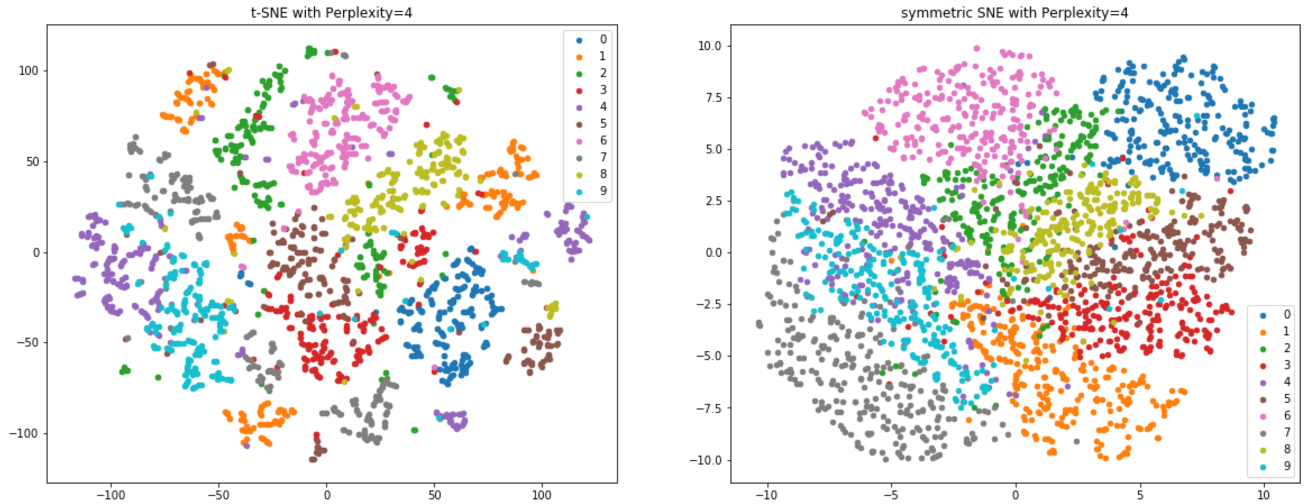
這樣的關係同時也反應在圖片的顏色深淺變化上

Different perplexity

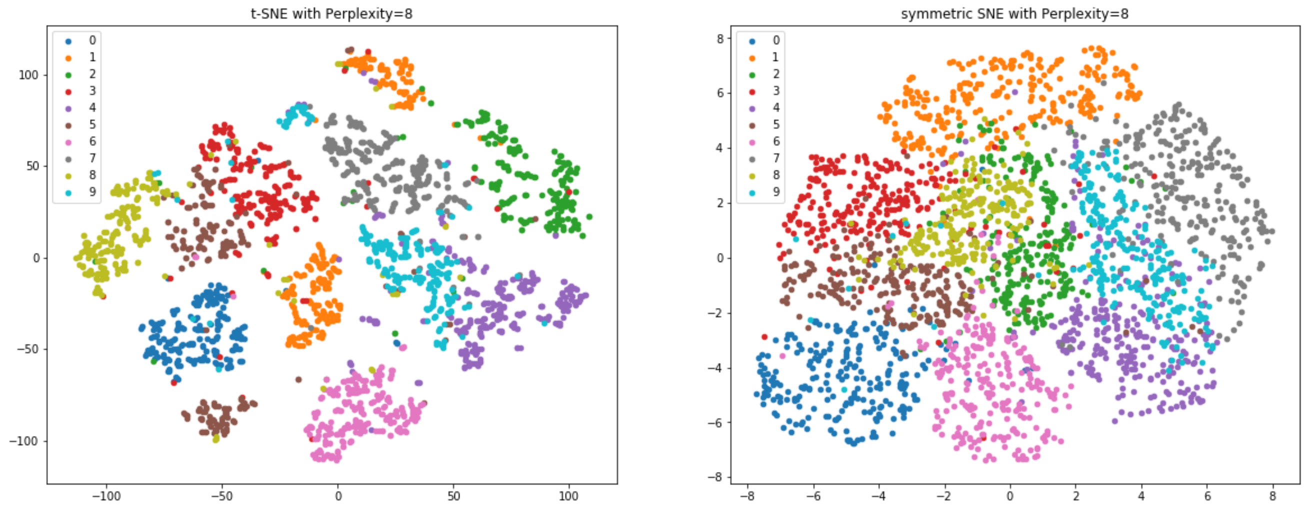
perplexity 跑以下數值

```
perplexitys = 2**np.arange(2,12)
# array([   4,    8,   16,   32,   64,  128,  256,  512, 1024, 2048])
```

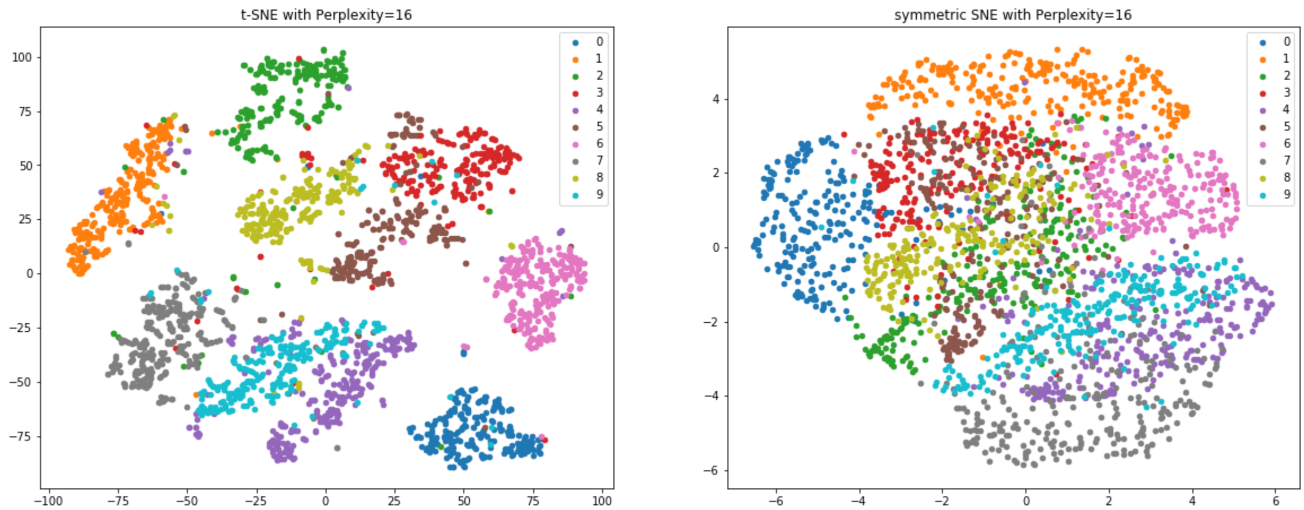
- 4


t-SNE with Perplexity=4
symmetric SNE with Perplexity=4
0
1
2
3
4
5
6
7
8
9

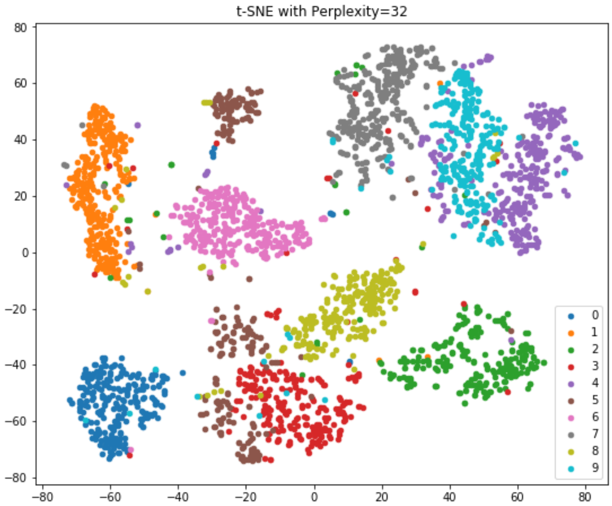
- 8

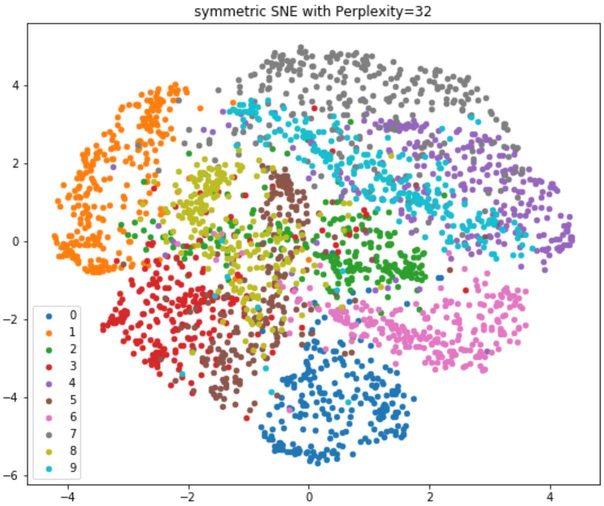

t-SNE with Perplexity=8
symmetric SNE with Perplexity=8
0
1
2
3
4
5
6
7
8
9

- 16

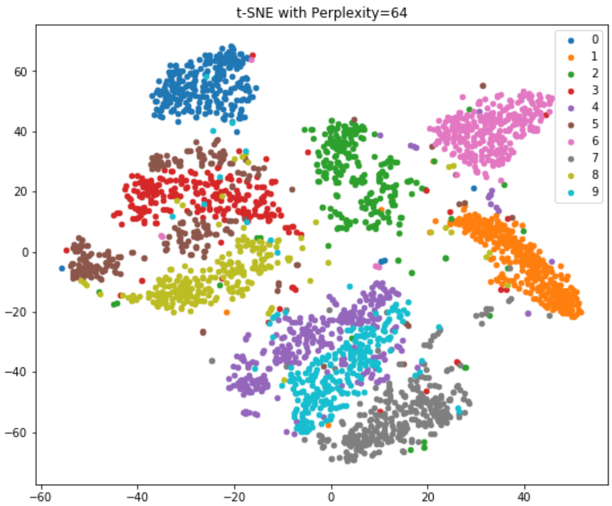

t-SNE with Perplexity=16
symmetric SNE with Perplexity=16
0
1
2
3
4
5
6
7
8
9

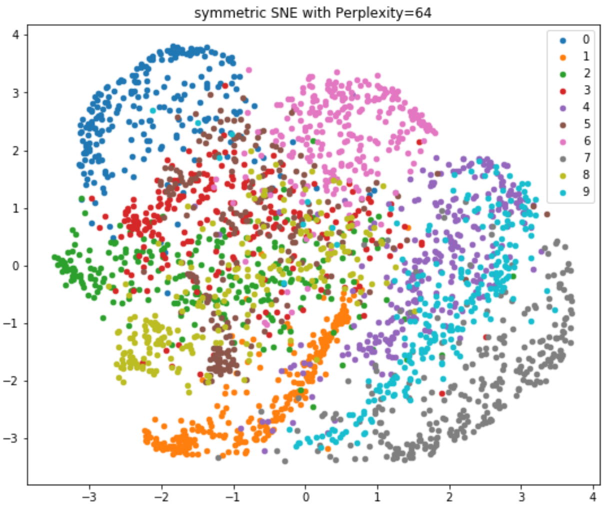
- 32


t-SNE with Perplexity=32
0
1
2
3
4
5
6
7
8
9

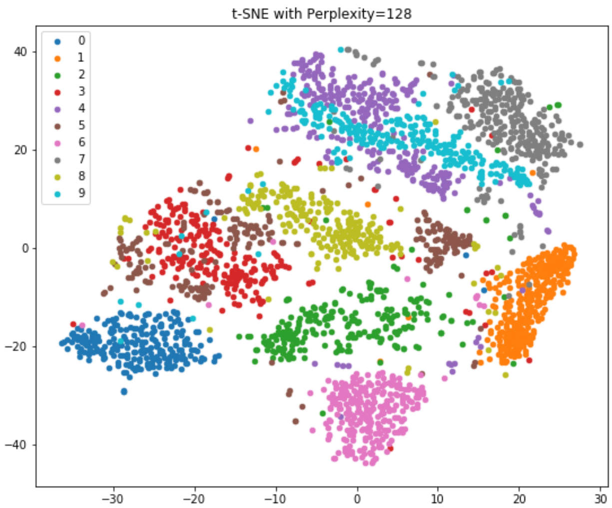

symmetric SNE with Perplexity=32
0
1
2
3
4
5
6
7
8
9

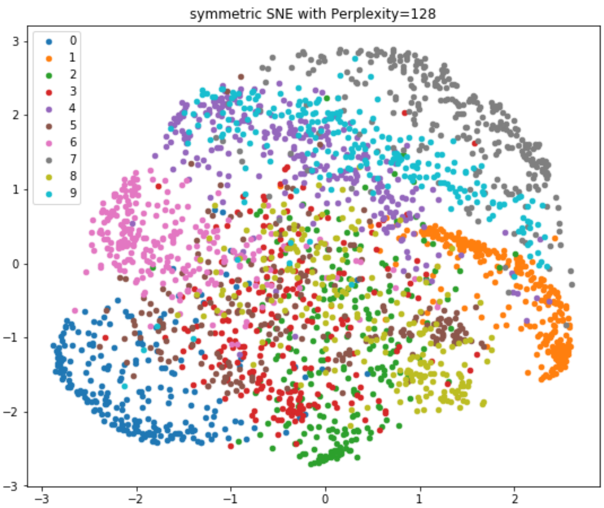
- 64


t-SNE with Perplexity=64
0
1
2
3
4
5
6
7
8
9

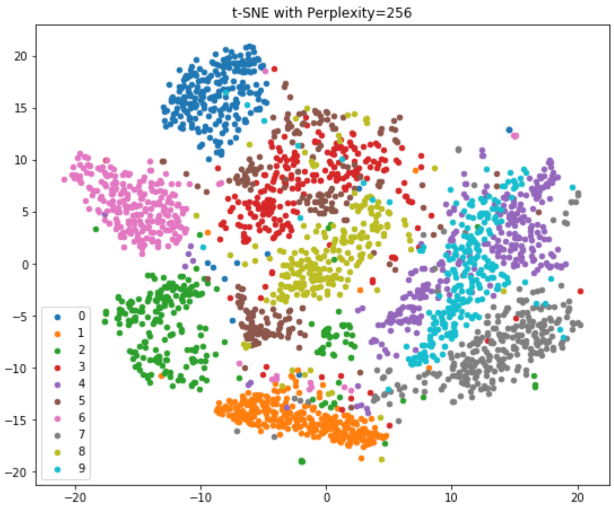

symmetric SNE with Perplexity=64
0
1
2
3
4
5
6
7
8
9

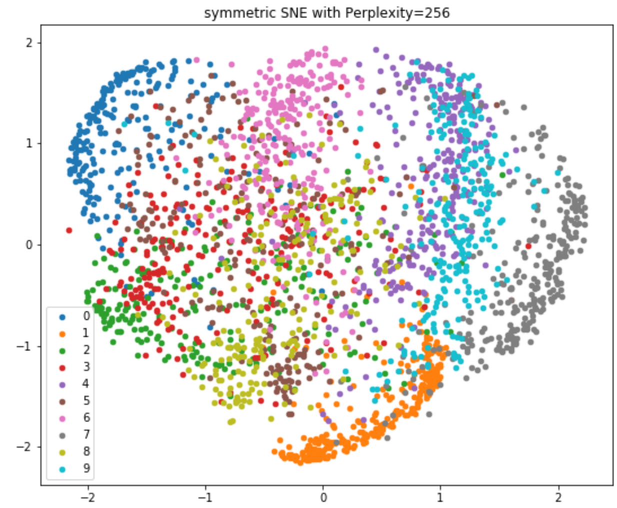
- 128


t-SNE with Perplexity=128
0
1
2
3
4
5
6
7
8
9

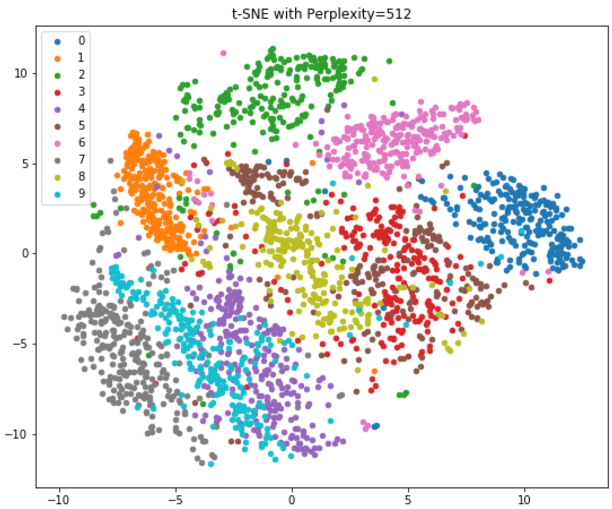

symmetric SNE with Perplexity=128
0
1
2
3
4
5
6
7
8
9

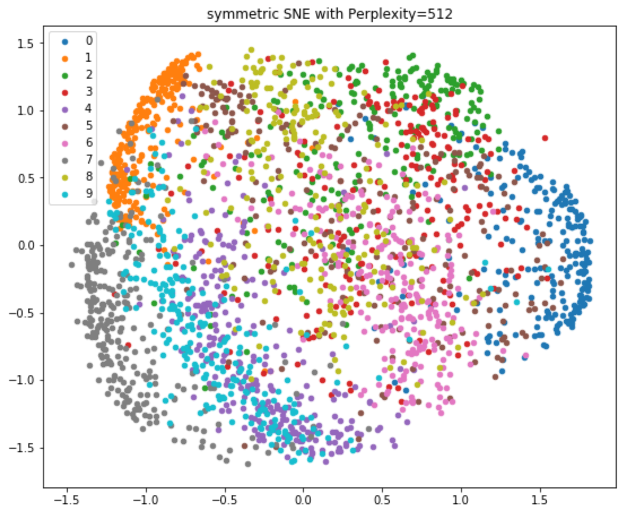
- 256


t-SNE with Perplexity=256
0
1
2
3
4
5
6
7
8
9

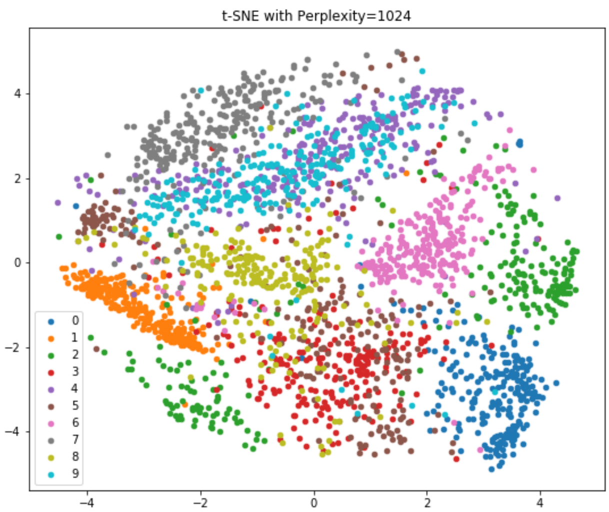

symmetric SNE with Perplexity=256
0
1
2
3
4
5
6
7
8
9

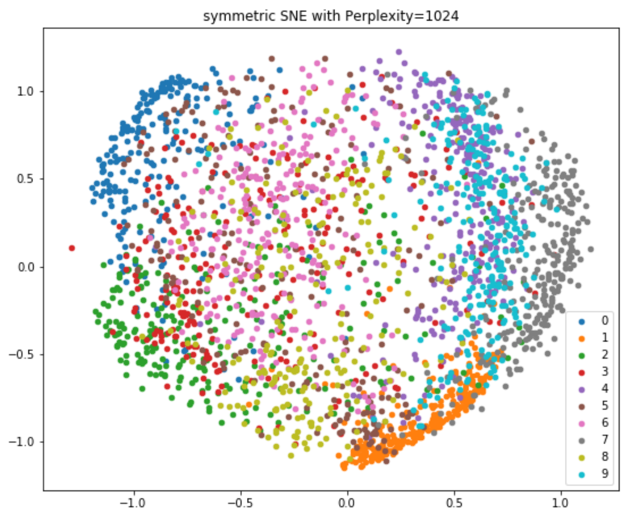
- 512


t-SNE with Perplexity=512
0
1
2
3
4
5
6
7
8
9

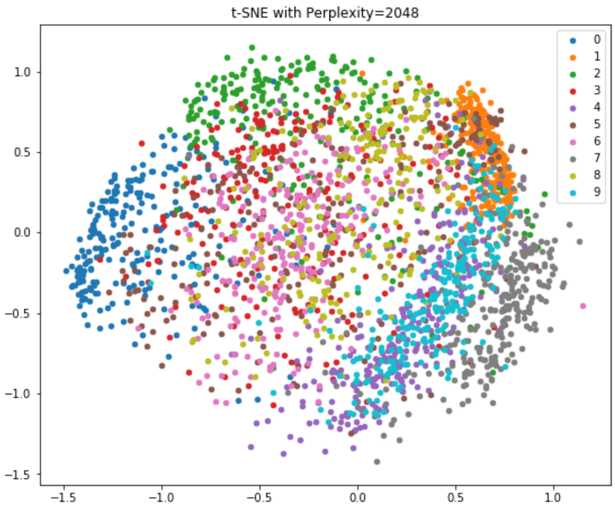

symmetric SNE with Perplexity=512
0
1
2
3
4
5
6
7
8
9

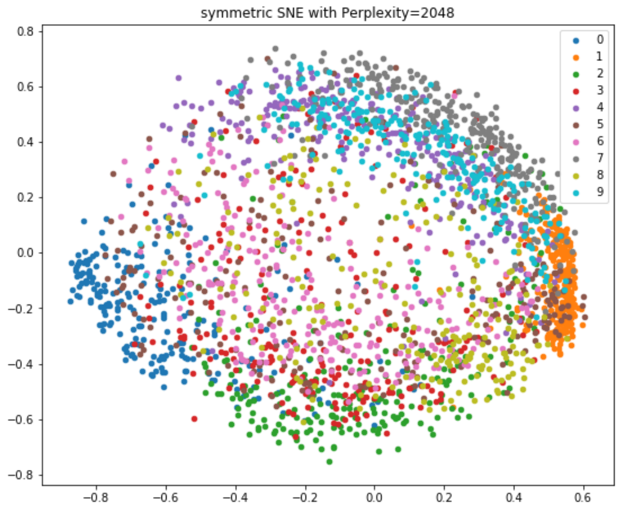
- 1024


t-SNE with Perplexity=1024
0
1
2
3
4
5
6
7
8
9


symmetric SNE with Perplexity=1024
0
1
2
3
4
5
6
7
8
9

- 2048





my observation and discuss

這邊觀察到當 perplexity 越大，t-SNE 有種退步的趨勢
漸漸也產生擁擠問題，差異很大，反倒是 symmetric SNE 變化都不大

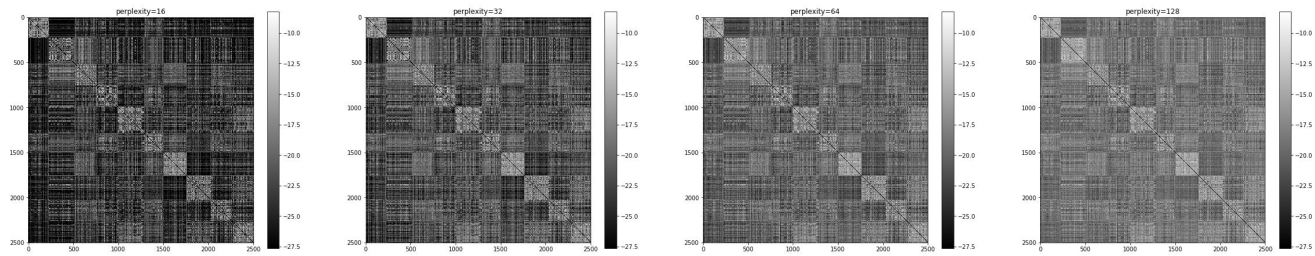
那為什麼，perplexity 越大結果 t-SNE 都擠在一起
這是因為在高維空間上的 P 使用的 σ 太大

導致高維分佈所有點都很相似，因此 t-SNE 擠在一起是因為要去符合這樣的分佈
讓相似的點靠在一起，也幾乎讓所有點靠在一起

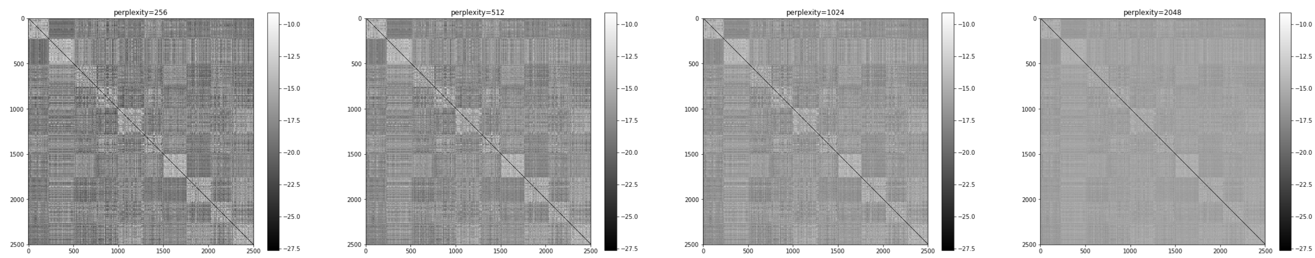
反過來 perplexity 太小，讓高維空間分佈覺得點都很不相似
因此可以看到 t-SNE 甚至讓同一群的點被拆散，分成了好幾群

也可以從 P 的分佈情形略知一二

- 從 16 到 128



- 從 256 到 2048



越往右邊是越高的 perplexity，可以看到越來越接近 uniform distribution
但在 2048 之下還是可以看深淺變化