a. code with detailed explanations

- t-SNE
  - **Part 1 :**

    In Part 1, we modify the origin tsne.py file, and make it back to symmetric SNE. In following part, we just show the code where tsne and symmetric SNE are different.

```
def tsne(X=np.array([]), no_dims=2, initial_dims=50, perplexity=30.0, istsne = True):
    """
```

In our function, we add a bool parameter “istsne”. If “istsne” is True, that’s mean we are running tsne algorithm, otherwise we are running symmetric SNE.

```
sum_Y = np.sum(np.square(Y), 1)
num = -2. * np.dot(Y, Y.T)
if istsne :
    num = 1. / (1. + np.add(np.add(num, sum_Y).T, sum_Y))
else:
    num = np.exp(-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)
```

Above code show how we implement “Q” (the probabilty in low dimension) both in tsne and symmetric SNE.

In tsne:

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

In symmetric SNE:

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

```
# Compute gradient
PQ = P - Q
for i in range(n):
    if istsne :
        dY[i, :] = np.sum(np.tile(PQ[:, i] * num[:, i], (no_dims, 1)).T * (Y[i, :] - Y), 0)
    else:
        dY[i, :] = np.sum(np.tile(PQ[:, i], (no_dims, 1)).T * (Y[i, :] - Y), 0)
```

Above code show the implement of calculate cost function gradient in both tsne and symmetric SNE.

In tsne:

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

In symmetric SNE:

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

- **Part2 :**

```
def draw_img(Y, labels, idx):
    pylab.scatter(Y[:, 0], Y[:, 1], 20, labels)
    pylab.savefig('GIF/it' + str(idx) + '.png')
```

```
def compose_gif():
    path_idx = [100, 200, 300, 400, 500, 600, 700, 800, 900, 1000]
    all_path = []
    for i in path_idx:
        all_path.append("GIF/it" + str(i) + ".png")
    gif_images = []
    for path in all_path:
        gif_images.append(imageio.imread(path))
    imageio.mimsave("test.gif",gif_images,fps=1)
```

To visualize the embedding result, we first use function “draw_img” to store the embedding result (we will store image in each 100 iteration). Then we use function “compose_gif” to make a GIF file. Note that in we will color the data points depend on their real label, which have been required in spec.

- **Part 3 :**

```
img_x = np.arange(2500)
plt.figure(figsize=(10,7),dpi=100,linewidth = 2)
plt.plot(img_x, Q[0], label="Q")
plt.plot(img_x, P[0], label="P")
plt.legend(loc = "best", fontsize=20)
plt.show()
```

Above code how we visualize the distribution in both high-dimensional space and low-dimensional space. The “P” and “Q” in the code are 2500*2500 matrix, which are return from our embedding algorithm.

- **Part 4 :**

```
# test different perplexity

all_perplexity = [10, 25, 50, 100]
all_perplexity_C = []

for i in range(len(all_perplexity)):
    Y, P, Q, all_C = tsne(X, 2, 50, all_perplexity[i], True)
    all_perplexity_C.append(all_C)
    pylab.scatter(Y[:, 0], Y[:, 1], 20, labels)
    pylab.savefig('perplexity=' + str(all_perplexity[i]) + '.png')
```

In part 4, we test four level of perplexity (10, 25, 50, 100) and observe the visualize result by storing the result image.

b. experiments settings and results & discussion

- t-SNE
  - **Part 1 :**

We run tsne and symmetric SNE separately:

tsne

```
print("Run tsne")
Y, P, Q, all_C = tsne(X, 2, 50, 20.0, True)
pylab.scatter(Y[:, 0], Y[:, 1], 20, labels)
pylab.show()
```

symmetric SNE

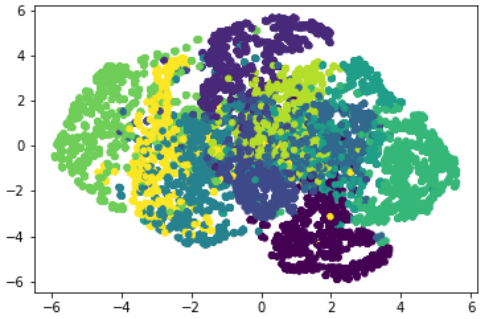
```
print("Run symmetric sne")
Y, P, Q, all_C = tsne(X, 2, 50, 20.0, False)
pylab.scatter(Y[:, 0], Y[:, 1], 20, labels)
pylab.show()
```

We set the iteration both in tsne and symmetric SNE are 10000, and their perplexity are 20 in this part.

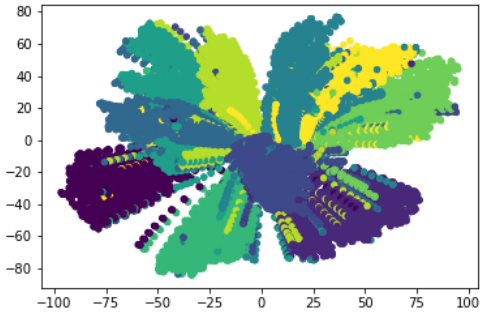
Note that the different between symmetric SNE and tsne is that there is a crowded problem in symmetric SNE, which mean each cluster may crowd together and it may be hard to separate them in low dimension in symmetric SNE. To handle crowded problem, tsne use t-distribution to model the data points in low dimension instead of gaussian distribution. T-distribution is a distribution with longer-tail, such that data should be further away in low dimension.

- **Part 2 :**

Symmetric SNE



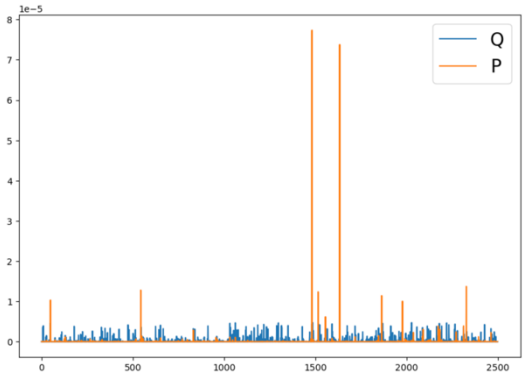
tsne



Obviously, there is some crowded problem in symmetric SNE as we expect.

- **Part 3 :**

Symmetric SNE

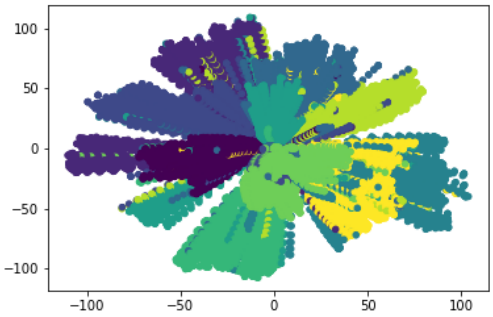


tsne

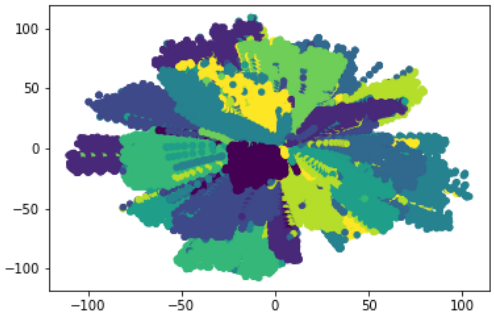
According to the chart, the result of tsne is better than symmetric SNE. Because the distribution of Q is more similar with P in tsne.

- **Part 4 :**

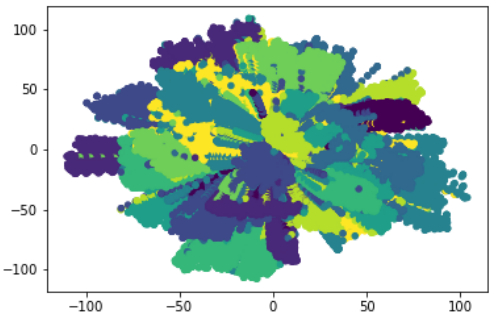
perplexity = 10



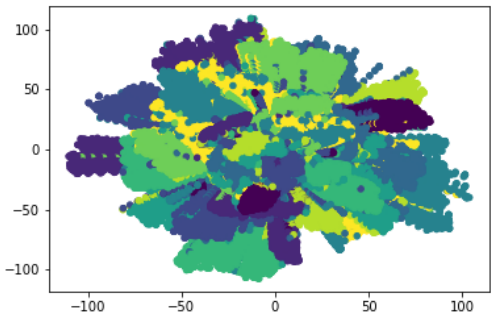
perplexity = 25



perplexity = 50



perplexity = 100



The number of perplexity means the effective number of neighbors of each point that we would consider during running our algorithm. If the perplexity is too high, our data points may mix together. Above charts show this phenomenon. When perplexity is 10, each cluster showed in visualization is the most separatist. When we set perplexity higher, the data points in the visualize charts are obviously muddier.

c. observations and discussion

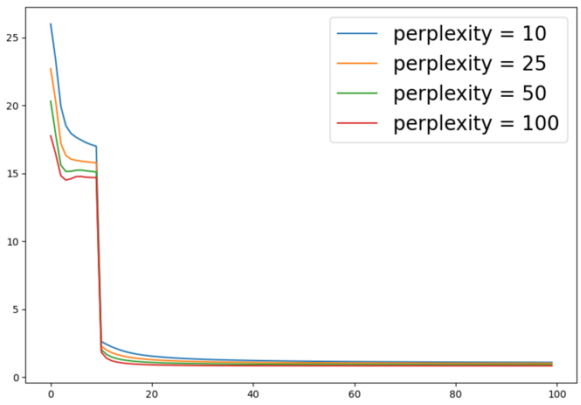
In this section, we discuss the “Error” of our model in both symmetric SNE and tsne.:

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

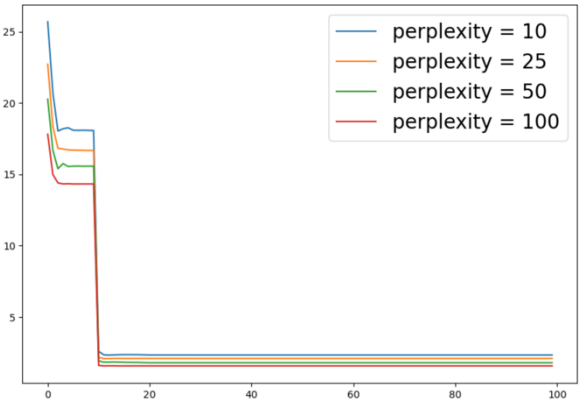
Which actually means the KL divergence between P and Q, Moreover, we compare the error in different perplexity.

We select four kinds of perplexity, 10, 25, 50 and 100, which is same as part 4. Following is the result:

tsne



symmetric SNE



y-axis is the KL divergence, and x-axis represents iteration (we extract KL divergence each 10 iteration, so there is total 1000/10 =100 points in the each hart)

According to above charts, it is clear that the KL divergence goes lower while perplexity goes higher. Moreover, in both tsne and symmetric SNE, the KL divergence will drop extremely at iteration 100. Note that the KL divergence of tsne will about to drop to 0.82, and the KL divergence of symmetric will drop to about 1.5. It means the KL divergence of tsne is a little bit lower than symmetric SNE.