> Note: This report was made on `Hackmd.io` and restricted by the `.pdf` format, the `.gif` animation would not display. Please view it on https://hackmd.io/@swchiu/SJQehejA8, thanks.

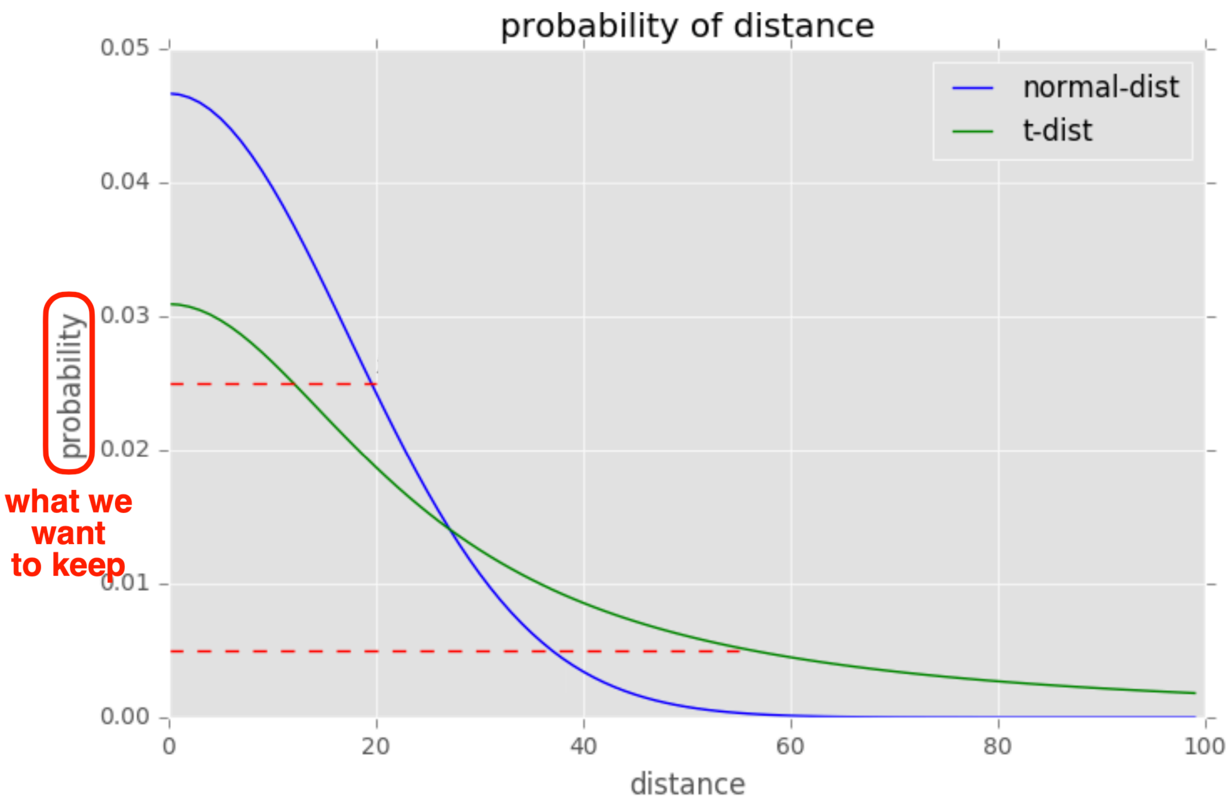
t-SNE / Symmetric SNE

Referrence to the https://lvdmaaten.github.io/tsne/code/tsne_python.zip and modify from it.

Modify t-SNE into symmetric SNE

`t-SNE` using a different distribution on the reducted data (i.e, `student t-distribution`, which is a more widly distribution than `Gaussion`).

Because `t-SNE` use a more widly distribution, the crowded problem would be sloved.



> We could see that when the data is more closed, the distance in `t-SNE` is more far.

The crowded problem which `symmetric SNE` would face, I will show at the result part below.

`SNE`'s concept is that it want to preserve the probabilitic distribution after the reduction.

Look at the definition of probability of `symmetreic SNE` and `t-SNE`,

$$symmetric\ SNE:$$
$$p_{ij} = \frac{exp(-||x_i - x_j||^2/(2\sigma^2))}{\sum_{k\neq i} exp(-||x_l - x_k||^2/(2\sigma^2))}$$
$$q_{ij} = \frac{exp(-||y_i - y_j||^2)}{\sum_{k\neq l} exp(-||y_l - y_k||^2)}$$
$$t - SNE:$$
$$p_{ij} = \frac{exp(-||x_i - x_j||^2/(2\sigma^2))}{\sum_{k\neq l} exp(-||x_l - x_k||^2/(2\sigma^2))}$$
$$q_{ij} = \frac{(1 + ||y_i - y_j||^2)^{-1}}{\sum_{k\neq l} (1 + ||y_l - y_k||^2)^{-1}}$$

We could see that the different is at q_{ij} part, `t-SNE` using an different distribution to describe it.

The gradient descent derivation is also different due to different q_{ij}

$$C = KL(P||Q) = \sum_i \sum_{j\neq i} p_{ij} \log \frac{p_{ij}}{q_{ij}}$$

$$symmetric\ SNE:$$

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

$$t - SNE:$$

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

So, we only need to find out the specific code segment related to those part and modify it into `symmetric SNE` form.

> Look at C, we could observ that when the data is more closed in higher dimension, then the data would not be sparse, too.
> But if datas are sparse in higher dimension, datas in lower dimension might be closed!
> [name=Shao-Wei Chiu]

The first part is shown below(`line 14 to line 15`):

```
...
# Compute pairwise affinities
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 sym_sne:
```

```
            sum_Y_ssne = np.sum(np.square(Y_ssne), 1)
            num_ssne = -2. * np.dot(Y_ssne, Y_ssne.T)
            # different between t-sne region
            num_ssne = np.add(np.add(num_ssne, sum_Y_ssne).T, sum_Y_ssne)
            num_ssne = np.exp(-1 * num_ssne)
            # different between t-sne region
            num_ssne[range(n), range(n)] = 0.
            Q_ssne = num_ssne / np.sum(num_ssne)
            Q_ssne = np.maximum(Q_ssne, 1e-12)
        ...
```

And the second part is `line 9 to line 10`:

```
    ...

    dY[i, :] = np.sum(
        np.tile(PQ[:, i] * num[:, i], (no_dims, 1)).T * (Y[i, :] - Y),
        0)
    if sym_sne:
        # different between t-sne
        dY_ssne[i, :] = np.sum(
            np.tile(PQ_ssne[:, i],
                    (no_dims, 1)).T * (Y_ssne[i, :] - Y_ssne), 0)
    ...
```

I modify it to perform `t-SNE` and `symmetric SNE` by calling `tsen()` method only, so I copy some parameter of `t-SNE` for running `symmetric SNE` at the same time.
And I stored the `Y` into a list at each ten steps for visualization.
I implement some useful method for visualization (e.g., `make_gif(...)`, `show_similarity(...)`), too.

```
def make_gif(record, labels, method, perplexity):
    camera = Camera(plt.figure())
    plt.title(method + ' with Perplexity=' + str(perplexity))
    for i in range(len(record)):
        img = plt.scatter(record[i][:, 0], record[i][:, 1], 20, labels)
        camera.snap()
    anim = camera.animate(interval=5, repeat_delay=20)
    anim.save(
        'output/' + method + '_' + str(perplexity) + '.gif', writer='pillow')
    plt.scatter(record[-1][:, 0], record[-1][:, 1], 20, labels)
    plt.savefig('output/' + method + '_' + str(perplexity) + '.png')


def show_similarity(S, labels, title, filename, perplexity):
    n = len(S)
    sort_idx = np.concatenate(
        [np.where(labels == l)[0] for l in np.unique(labels)])
```

```python
    plt.figure(figsize=(10 * n, 7.5))
    S = [np.log(p[:, sort_idx][sort_idx, :]) for p in S]
    all_min = min([np.min(p) for p in S])
    all_max = max([np.max(p) for p in S])
    for i in range(n):
        plt.subplot(1, n, i + 1)
        plt.title(title[i])
        im = plt.imshow(S[i], cmap='gray', vmin=all_min, vmax=all_max)
        plt.colorbar(im)
    plt.savefig('output/' + filename + '_' + str(perplexity) + '.png')
```

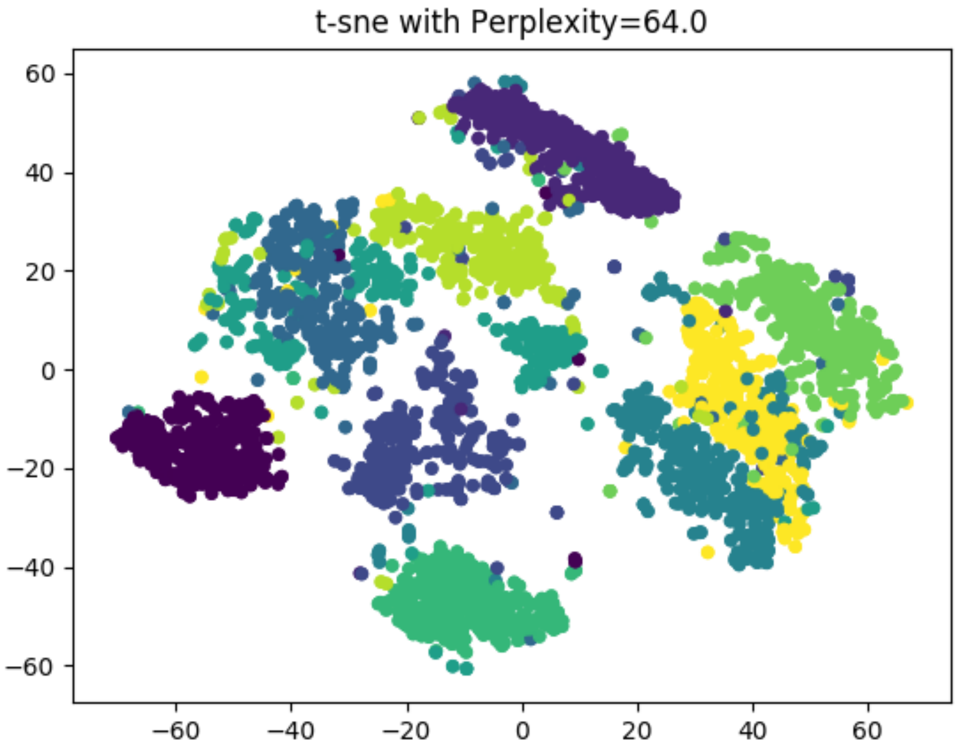
```python
def gradient_descend(iter, Y, iY, dY, gains, min_gain, momentum, eta):
    gains = (gains + 0.2) * ((dY > 0.) != (iY > 0.)) + (gains * 0.8) * (
        (dY > 0.) == (iY > 0.))
    gains[gains < min_gain] = min_gain
    iY = momentum * iY - eta * (gains * dY)
    Y = Y + iY
    Y = Y - np.tile(np.mean(Y, 0), (Y.shape[0], 1))

    return Y, iY, gains
def tsne(
        X=np.array([]), no_dims=2, initial_dims=50, perplexity=30.0,
        sym_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 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
    (n, d) = X.shape
    max_iter = 1000
    initial_momentum = 0.5
    final_momentum = 0.8
    eta = 500
    min_gain = 0.01

    record = []
    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))

    if sym_sne:
        record_ssne = []
        Y_ssne = Y.copy()
        dY_ssne = np.zeros((n, no_dims))
        iY_ssne = np.zeros((n, no_dims))
        gains_ssne = np.ones((n, no_dims))
        Q_ssne = 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.  # early exaggeration
    P = np.maximum(P, 1e-12)

    # Run iterations
    for iter in range(max_iter):

        # Compute pairwise affinities
        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 sym_sne:
            sum_Y_ssne = np.sum(np.square(Y_ssne), 1)
            num_ssne = -2. * np.dot(Y_ssne, Y_ssne.T)
            # different between t-sne region
            num_ssne = np.add(np.add(num_ssne, sum_Y_ssne).T, sum_Y_ssne)
            num_ssne = np.exp(-1 * num_ssne)
            # different between t-sne region
            num_ssne[range(n), range(n)] = 0.
            Q_ssne = num_ssne / np.sum(num_ssne)
            Q_ssne = np.maximum(Q_ssne, 1e-12)

        # Compute gradient
        PQ = P - Q
        if sym_sne:
            PQ_ssne = P - Q_ssne
        for i in range(n):
            dY[i, :] = np.sum(
                np.tile(PQ[:, i] * num[:, i], (no_dims, 1)).T * (Y[i, :] - Y),
                0)
            if sym_sne:
                # different between t-sne
```

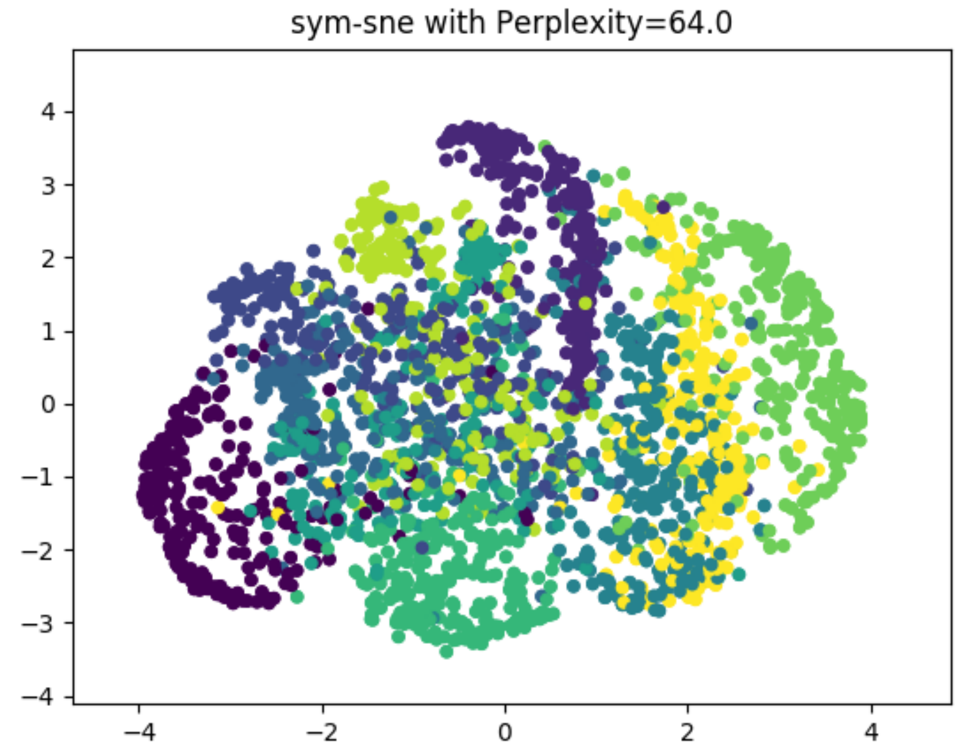
```
                dY_ssne[i, :] = np.sum(
                    np.tile(PQ_ssne[:, i],
                            (no_dims, 1)).T * (Y_ssne[i, :] - Y_ssne), 0)

        # Perform the update
        if iter < 20:
            momentum = initial_momentum
        else:
            momentum = final_momentum

        Y, iY, gains = gradient_descend(iter, Y, iY, dY, gains, min_gain,
                                        momentum, eta)
        if sym_sne:
            Y_ssne, iY_ssne, gains_ssne = gradient_descend(
                iter, Y_ssne, iY_ssne, dY_ssne, gains_ssne, min_gain,
momentum,
                eta)

        # Compute current value of cost function
        if (iter + 1) % 10 == 0:
            C = np.sum(P * np.log(P / Q))
            record.append(Y)
            if sym_sne:
                C_ssne = np.sum(P * np.log(P / Q_ssne))
                record_ssne.append(Y_ssne)
                print("Iter %4d: tsne error = %f, sym-sne error = %f" %
                      (iter + 1, C, C_ssne))
            else:
                print("Iter %4d: tsne error = %f" % (iter + 1, C))

        # Stop lying about P-values
        if iter == 100:
            P = P / 4.

    # Return solution
    if not sym_sne:
        return record, P, Q
    else:
        return (record, record_ssne), P, (Q, Q_ssne)
```

Result and Discussion

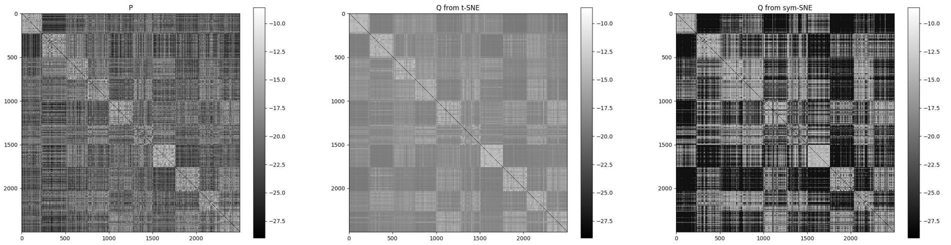
1. Visualize the embedding of both `t-SNE` and `symmetric SNE`
    - t-sne with preplexity=64



- sym-sne with preplexity=64

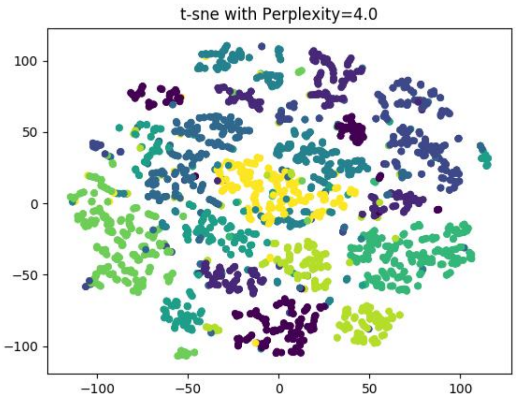


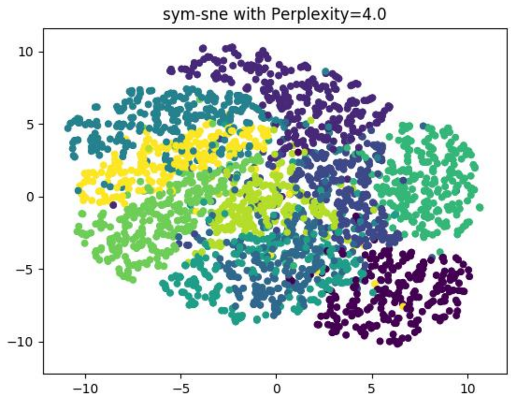
2. Visualize the distribution of pairwise similarities in both high-dimensional space and low-dimensional space, based on both `t-SNE` and `symmetric SNE`
   (P, Q with `t-SNE`, `sym-SNE`, respectively)

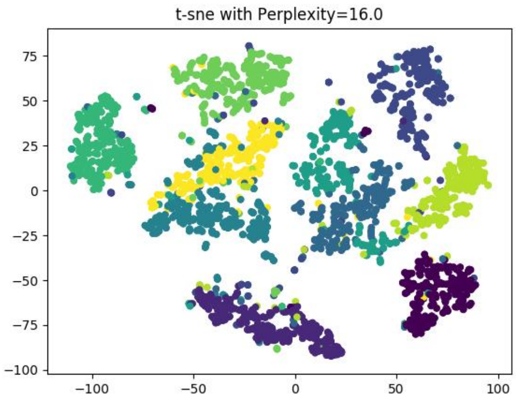


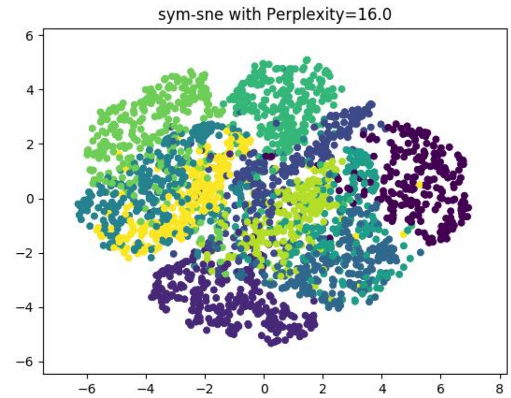
> We could see that `t-SNE`'s is more light, this is because t-distribution is more widly, when x is lower ,its probability is lower.

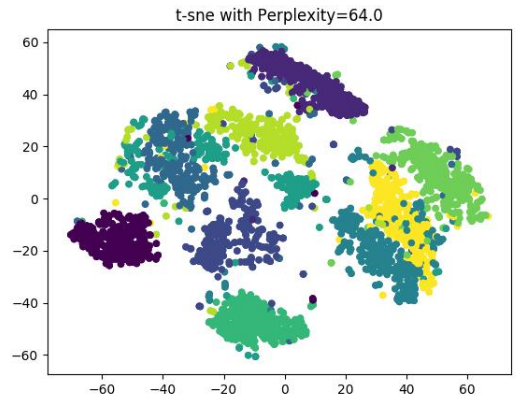
3. Play with different perplexity values

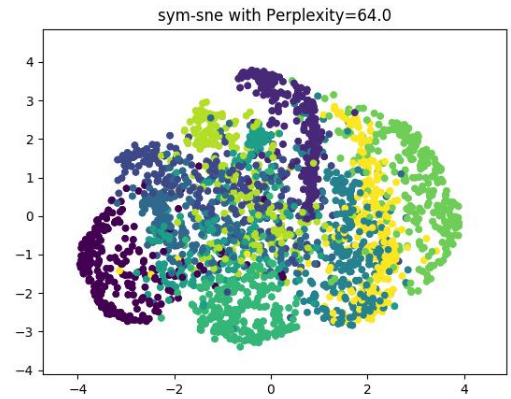


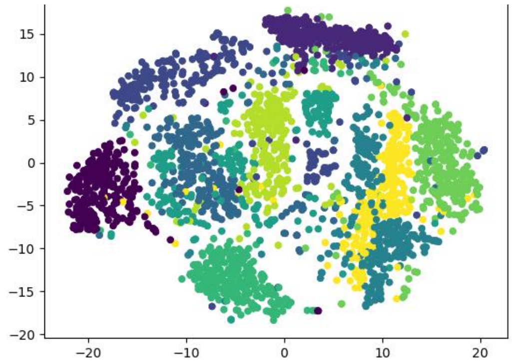


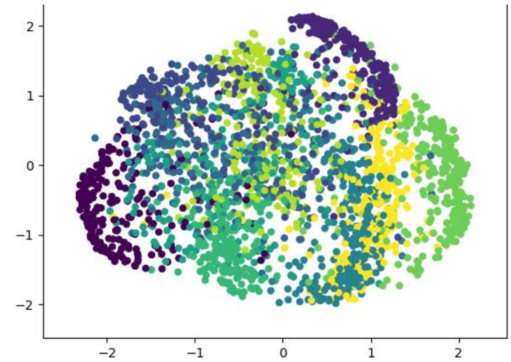




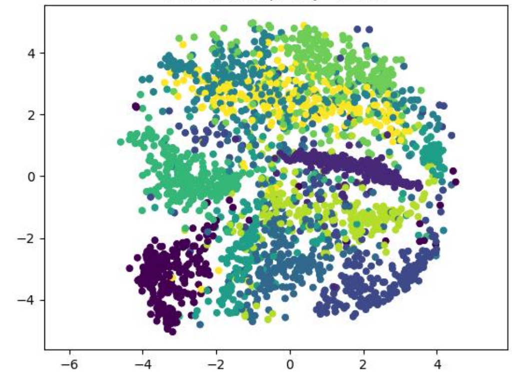




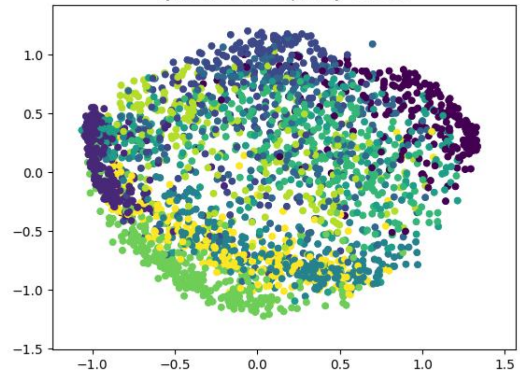




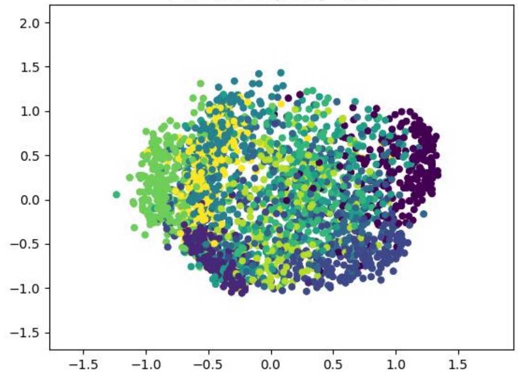
t-sne with Perplexity=1024.0



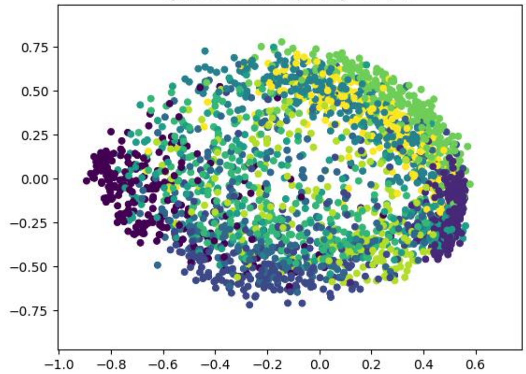
sym-sne with Perplexity=1024.0



t-sne with Perplexity=2048.0



sym-sne with Perplexity=2048.0



We could see that, when `preplexity` become large, `t-SNE` is more similar to `symmetric SNE`, I think this is related to entropy.
When `preplexity` is large, hints that the entropy is large, too. So the orignal distribution is more uniform.
Otherwise, when `preplexity` is small, hints that the entropy is small, and the original distribution is more similar to delta distribution.

And `t-SNE` is hard to tell the different in uniform distribution, because of the higer dimension data is more similar, and the result would be crowded.

For the small `preplexity` case, I observ that the `t-SNE` seems to split some labels.
I think this is because the higer dimension data become more dissimilar.