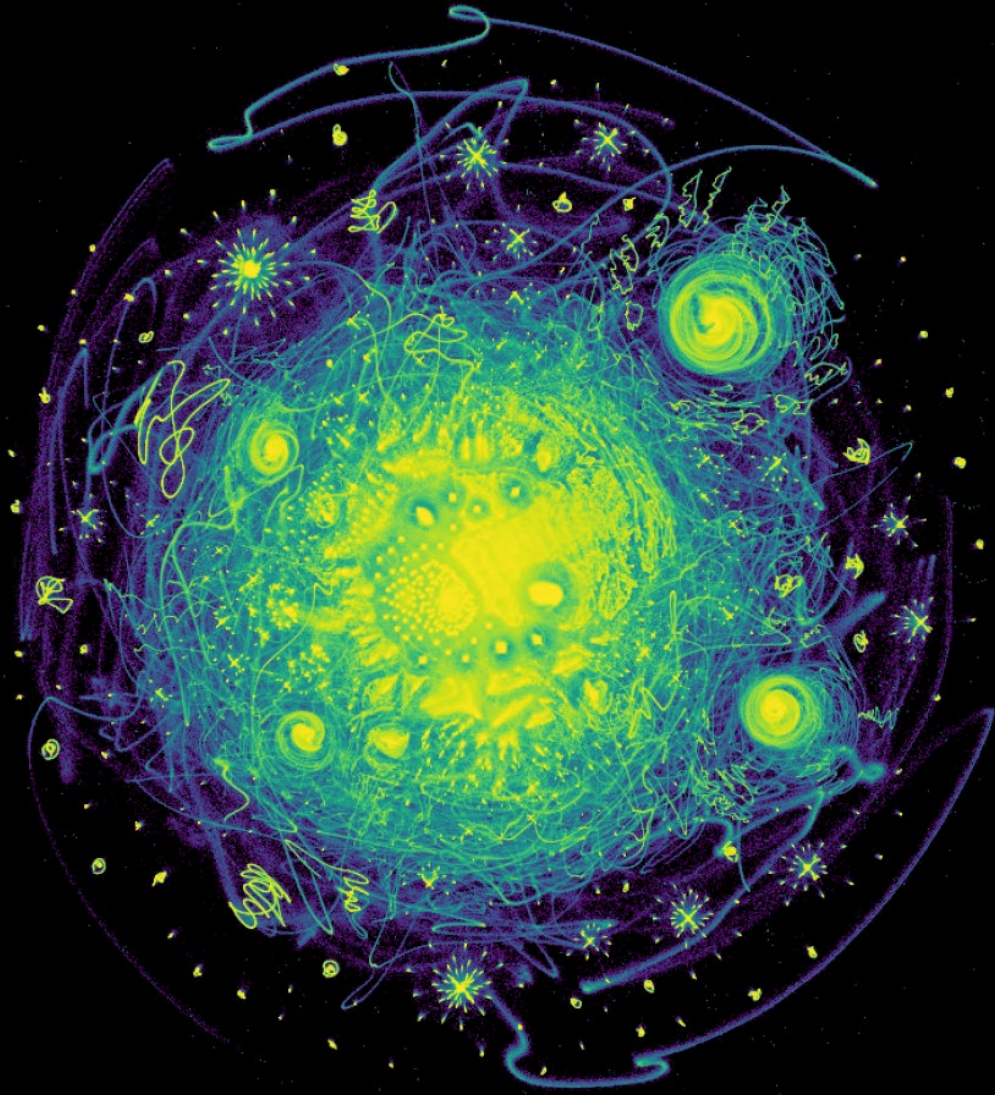


Dimensionality Reduction: PCA and low-D embeddings

Applied Machine Learning
Derek Hoiem

McInnes et al. (UMAP, 2000): Visualization of 30,000,000 integers as represented by binary vectors of prime divisibility, colored by integer value of the point

Go to tinyurl.com/cs441chat to ask questions during lecture (CampusWire in-lecture chat room)

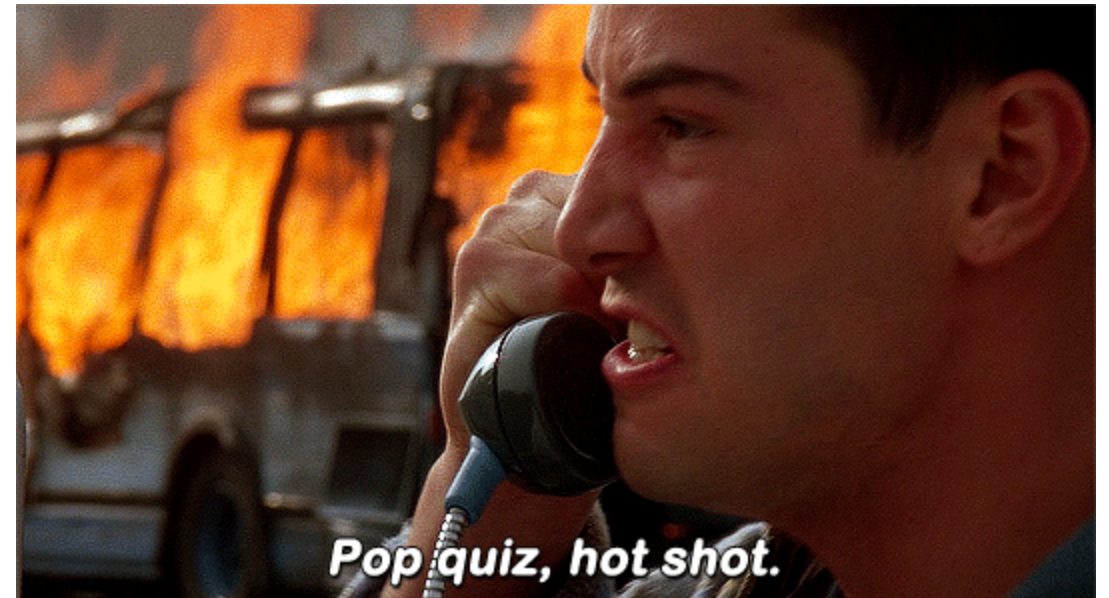


Last Class - KNN

- K-nearest neighbor classification and regression
- Measuring and understanding error

Which of these tend to be decreased as the number of training examples increases?

- a) Training error
- b) Test error
- c) Generalization error

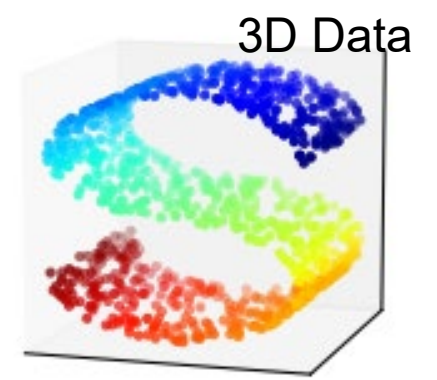


This class – dimensionality reduction

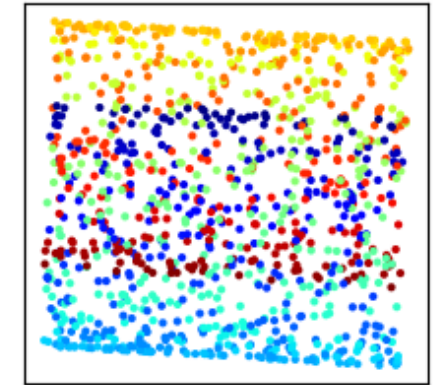
- Goal: We want to represent high dimensional data with fewer dimensions, e.g. for:
 - Compression: reduced storage, faster retrieval
 - Visualization: plot in two dimensions
- A good dimensionality reduction can be defined in different ways
 - Be able to reproduce the original data
 - Preserve discriminative features
 - Preserve the neighborhood structure
- One main strategy is linear projection, e.g. a street map projects everything onto the 2D ground dimensions and ignores height
- Another strategy is embedding or manifold fitting, where we try to preserve relationships in the data

This class – PCA and manifolds

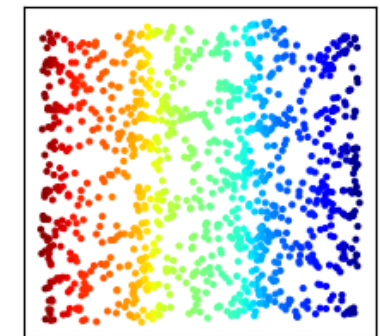
- Linear projection
 - PCA: Principal Components Analysis
 - Reduce dimension while preserving variance of data
- Embedding/manifold learning
 - MDS (multidimensional scaling), IsoMap and t-SNE
 - Preserve point distances and/or local structure



PCA projection



IsoMap projection



Key terms

- Vectors and matrices
- Translation
- Projection
- Scaling
- Rotation
- Rank
- Eigenvectors/eigenvalues
- SVD

Key terms

Vector can represent a data point \mathbf{x} or a **projection** \mathbf{w} onto a **coordinate**

- $\mathbf{w}^T \mathbf{x}$ projects data point \mathbf{x} onto the axis defined by \mathbf{w}
- E.g. suppose $d = 2$
 - $\mathbf{w} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$ selects the first value of \mathbf{x}
 - $\mathbf{w} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$ adds the two values of \mathbf{x} together

Matrix can represent a set of data points or a set of projection vectors

Key terms

Translation is a transformation that adds a constant value to each coordinate

- E.g. centering is $\mathbf{x}_c = \mathbf{x} - \boldsymbol{\mu}_x$, where $\boldsymbol{\mu}_x$ is the mean of \mathbf{x}

Scaling is a transformation that multiplies each coordinate by a constant value

- E.g. $\mathbf{W} = \begin{bmatrix} 2 & & \\ & 1 & \\ & & 1 \end{bmatrix}$ will double the value of the first coordinate of \mathbf{x} when applied as $\mathbf{W}\mathbf{x}$ (blanks are assumed to be zero)

Key terms

Rotation is a set of projections that preserves the distances between points and distances to the origin

- Each row and column represents a **basis vector**
- The basic vectors must have a unit norm and be orthogonal to each other: $\mathbf{r}_i^T \mathbf{r}_i = 1, \mathbf{r}_i^T \mathbf{r}_j = 0, \forall (i, j \neq i)$

Key terms

Rank of matrix \mathbf{M} is the number of linearly independent vectors in the rows or columns of \mathbf{M}

- $\text{Rank}(\mathbf{M})$ is at most the smaller of the number of rows and columns in \mathbf{M}
- $\text{Rank} \left(\begin{bmatrix} \mathbf{1} & \mathbf{0} \\ \mathbf{0} & \mathbf{1} \end{bmatrix} \right) = 2$
- $\text{Rank} \left(\begin{bmatrix} \mathbf{1} & \mathbf{2} \\ \mathbf{2} & \mathbf{4} \end{bmatrix} \right) = 1$ because one of the rows (or columns) can be composed of a weighted sum of the others

Key terms

Eigenvector \boldsymbol{v} and corresponding **eigenvalue** λ of matrix \boldsymbol{M} are defined by having the special property $\boldsymbol{M}\boldsymbol{v} = \lambda\boldsymbol{v}$

- Eigenvectors characterize matrices, and appear as part of a solution to many linear algebra problems

SVD (singular value decomposition) is a factorization of a matrix \boldsymbol{A} into $\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T$, where:

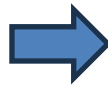
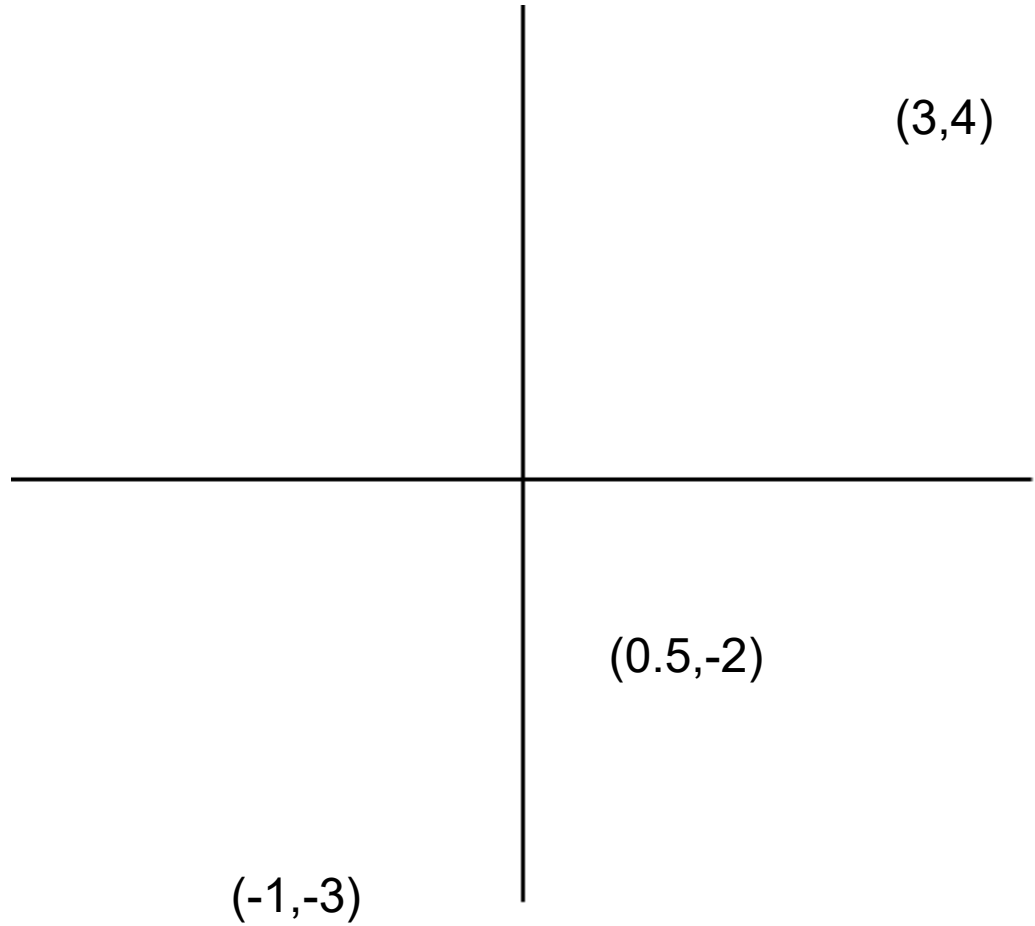
- Columns of \boldsymbol{U} are eigenvectors of $\boldsymbol{A}\boldsymbol{A}^T$
- Columns of \boldsymbol{V} are eigenvectors of $\boldsymbol{A}^T\boldsymbol{A}$
- $\boldsymbol{\Sigma}$ is a diagonal (scaling) matrix of singular values, which are square roots of the eigenvalues of both $\boldsymbol{A}\boldsymbol{A}^T$ and $\boldsymbol{A}^T\boldsymbol{A}$

PCA

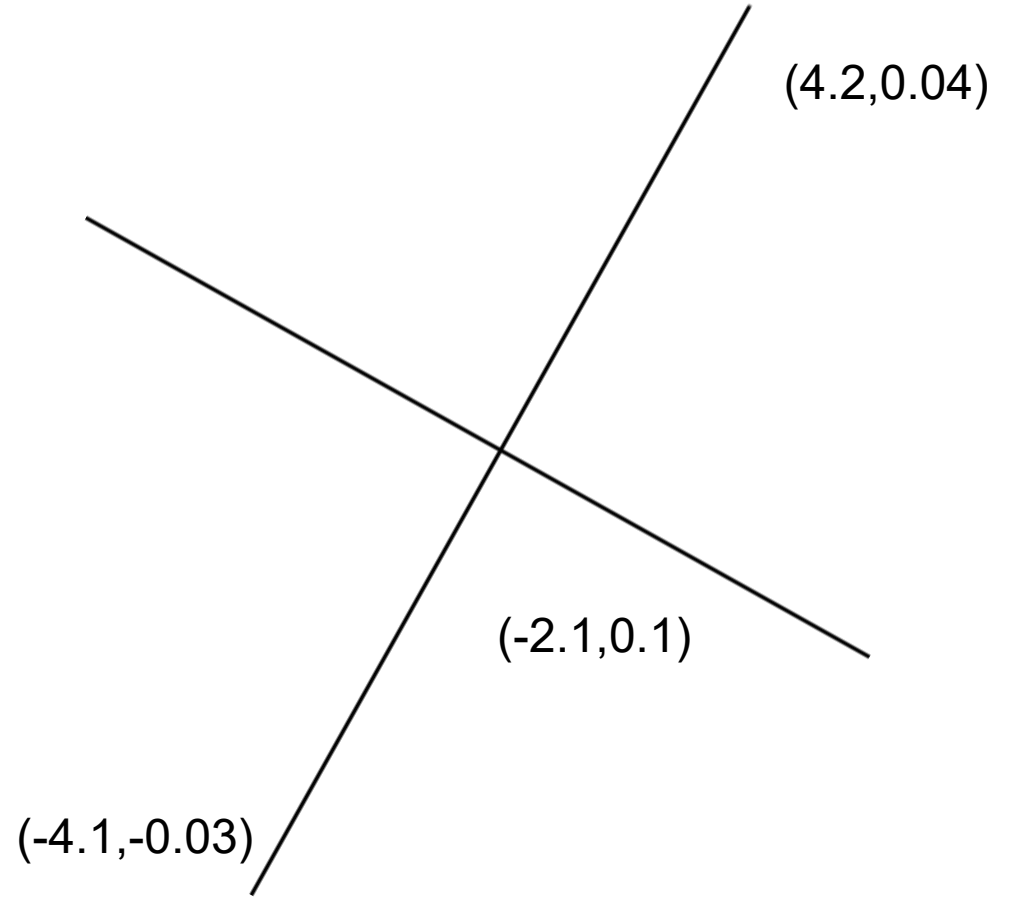
- General dimensionality reduction technique
 - Finds major orthogonal directions of variation
- Preserves most of variance with a much more compact representation
 - Lower storage requirements
 - Faster matching/retrieval
 - Easier to work in low dimensions, e.g. for probability estimation

Center and rotate the axes

Original 2D Representation

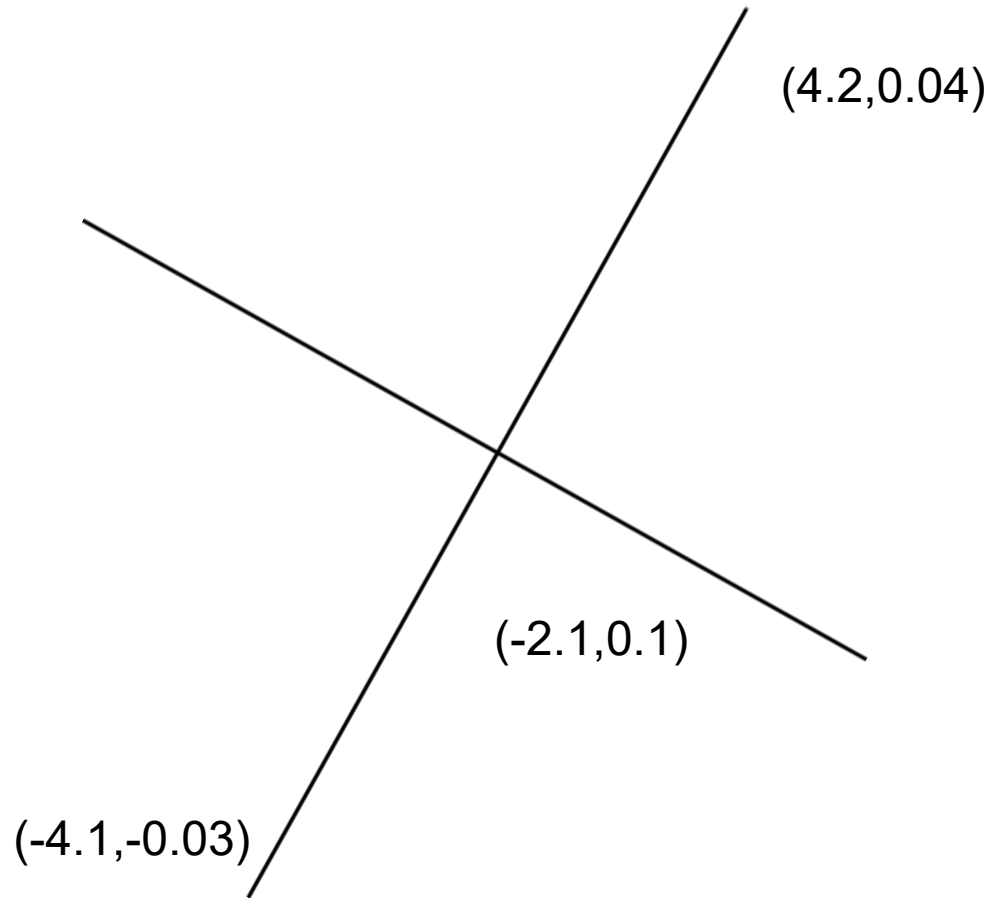


New 2D Representation



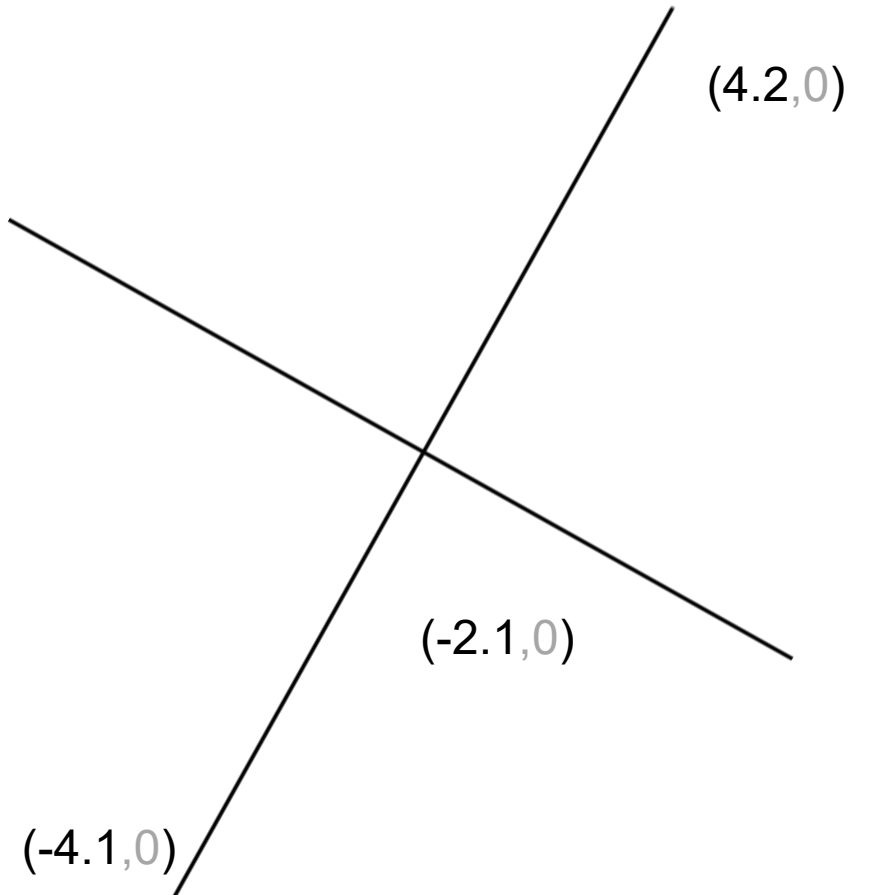
Now data is well-approximated by dropping the coordinates with the least variance

New 2D Representation



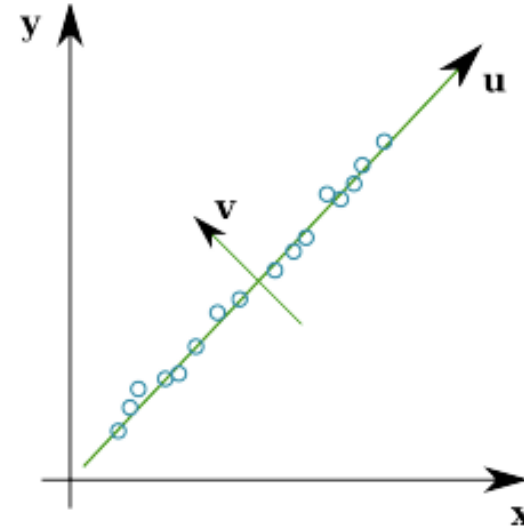
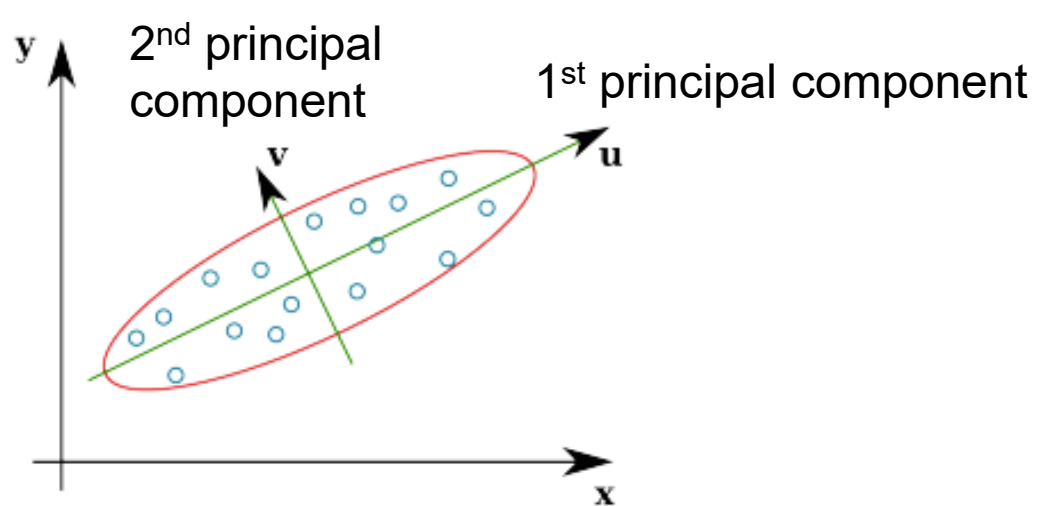
\approx

New 1D Representation



Principal Component Analysis

- Given a point set $\{\vec{p}_j\}_{j=1\dots P}$, in an M -dim space, PCA finds a basis such that
 - The most variation is in the first basis vector
 - The second most, in the second vector that is orthogonal to the first vector
 - The third...



Principal Component Analysis (PCA)

- Given: N data points $\mathbf{x}_1, \dots, \mathbf{x}_N$ in \mathbb{R}^d
- We want to find a new set of features that are linear combinations of original ones:

$$u(\mathbf{x}_i) = \mathbf{u}^T(\mathbf{x}_i - \boldsymbol{\mu})$$

($\boldsymbol{\mu}$: mean of data points)

- Choose unit vector \mathbf{u} in \mathbb{R}^d that captures the most data variance

Principal Component Analysis

Direction that maximizes the variance of the projected data:

Maximize $\frac{1}{N} \sum_{i=1}^N \underbrace{\mathbf{u}^T (\mathbf{x}_i - \mu)}_{\text{Projection of data point}} (\mathbf{u}^T (\mathbf{x}_i - \mu))^T$ subject to $\|\mathbf{u}\|=1$

$$= \mathbf{u}^T \left[\underbrace{\frac{1}{N} \sum_{i=1}^N (\mathbf{x}_i - \mu)(\mathbf{x}_i - \mu)^T}_{\text{Covariance matrix of data}} \right] \mathbf{u}$$

$$= \mathbf{u}^T \Sigma \mathbf{u}$$

The direction that maximizes the variance is the eigenvector associated with the largest eigenvalue of Σ (can be derived using Raleigh's quotient or Lagrange multiplier)

PCA in Python

```
print(X.shape)
x_mu = np.mean(X, axis=0)
X_cov = np.matmul((X-x_mu).transpose(), X-x_mu)/X.shape[0]
[lam, v] = np.linalg.eig(X_cov)
print(lam[:3])
v = v.transpose()
from sklearn.decomposition import PCA
pca_transform = PCA().fit(X)
print(np.max(np.abs(v[0]-pca_transform.components_[0])))
print(pca_transform.explained_variance_[:3])
```

```
(50000, 784)
[5.0695131  3.7301149  3.25871794]
3.9811903773667723e-16
[5.06961449  3.7301895  3.25878311]
```

Compute PCA components as eigenvectors of covariance matrix

Compute PCA using the PCA function

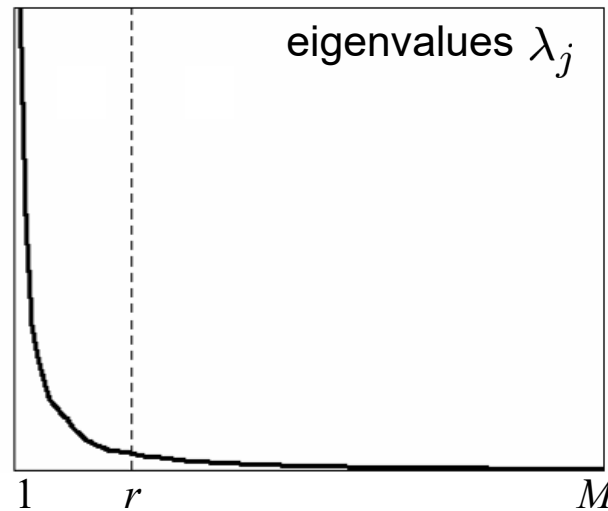
Printout shows that eigenvalues are the same as the explained variance, and the first component is identical (up to numerical precision)

Principal Component Analysis

First $r < M$ principal component vectors provide an approximate basis that minimizes the mean-squared-error (MSE) of reconstructing the original points

Choosing subspace dimension r :

- look at decay of the eigenvalues as a function of r
- Larger r means lower expected error in the subspace data approximation



Example on aligned faces

$\mathbf{x}_1, \dots, \mathbf{x}_N$ are pixel values of each face

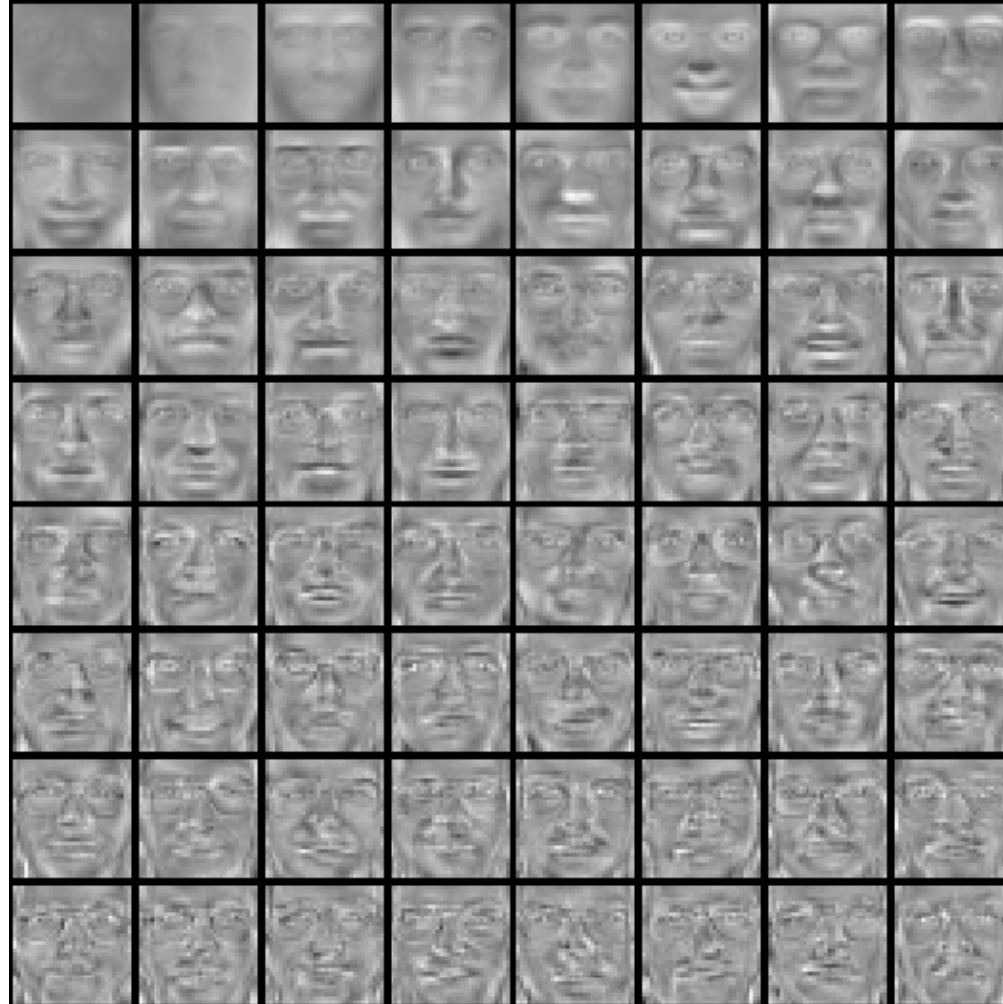


PCA of aligned face images (called “eigenfaces”)

Mean: μ



Top eigenvectors: u_1, \dots, u_k



Visualization of eigenfaces (appearance variation)

Principal component (eigenvector) u_k



$\mu + 3\sigma_k u_k$



$\mu - 3\sigma_k u_k$



Representation and reconstruction


- Face \mathbf{x} in “face space” coordinates:



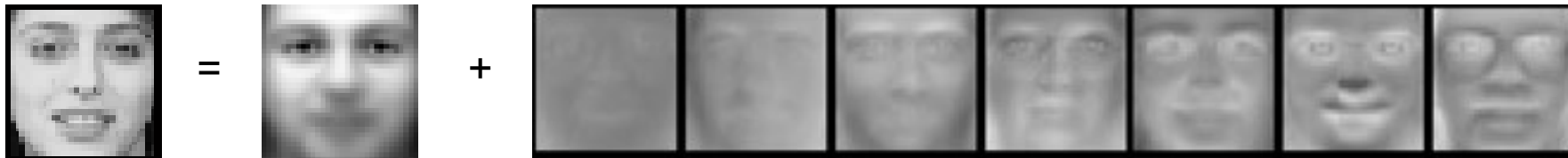
$$\begin{aligned}\mathbf{x} &\longrightarrow [\mathbf{u}_1^T (\mathbf{x} - \mu), \dots, \mathbf{u}_k^T (\mathbf{x} - \mu)] \\ &= w_1, \dots, w_k\end{aligned}$$

Representation and reconstruction

- Face \mathbf{x} in “face space” coordinates:


$$\mathbf{x} \rightarrow [\mathbf{u}_1^T (\mathbf{x} - \mu), \dots, \mathbf{u}_k^T (\mathbf{x} - \mu)]$$
$$= w_1, \dots, w_k$$

- Reconstruction:


$$\hat{\mathbf{x}} = \mu + w_1 \mathbf{u}_1 + w_2 \mathbf{u}_2 + w_3 \mathbf{u}_3 + w_4 \mathbf{u}_4 + \dots$$

Reconstruction

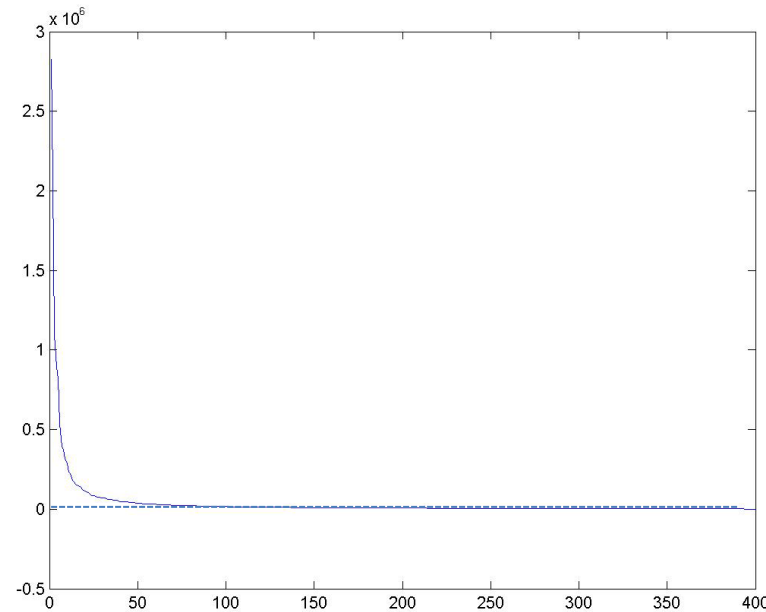


After computing eigenfaces using 400 face images from ORL face database

Note

Preserving variance (minimizing MSE) does not necessarily lead to perceptually good reconstruction.

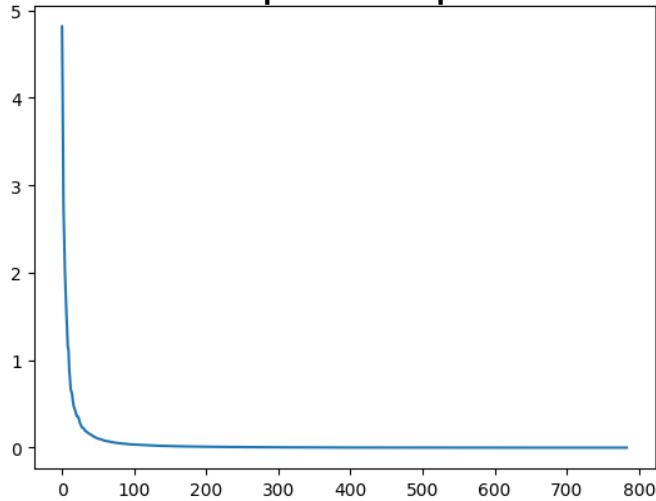
$P = 200$



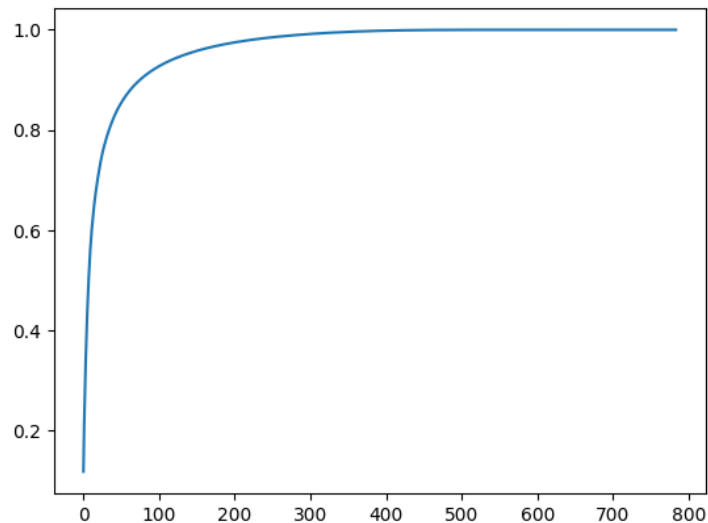
Plot of eigenvalues for each eigenvector of the covariance matrix, equivalent to the variance contained along each principal component

Another example, representing MNIST 4's

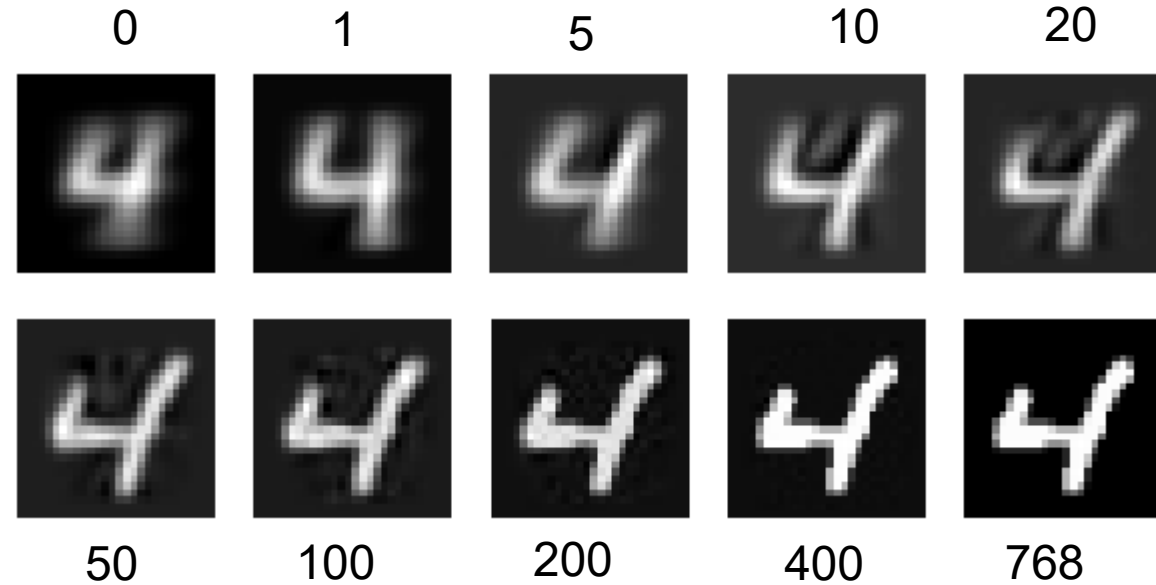
Variance per component



Cumulative % variance explained



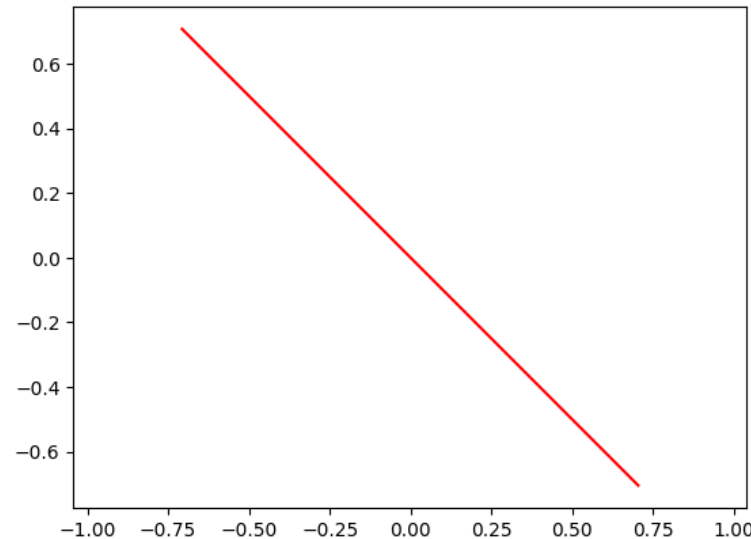
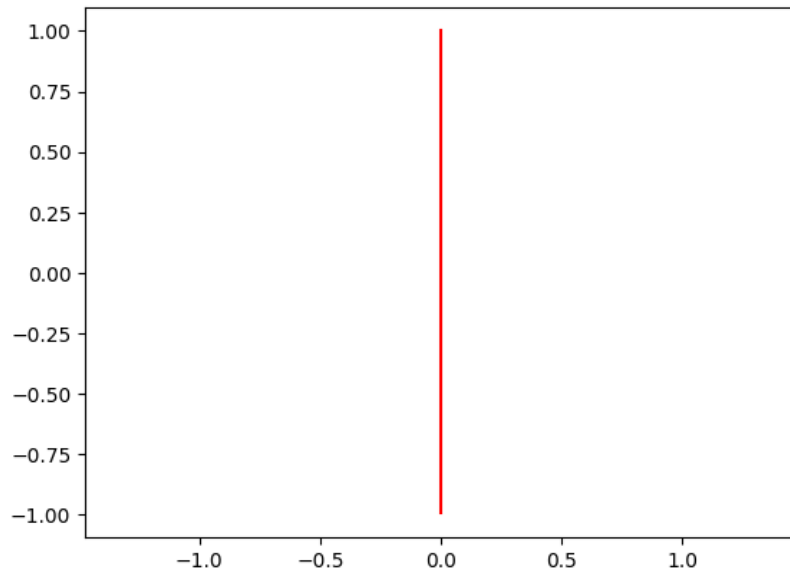
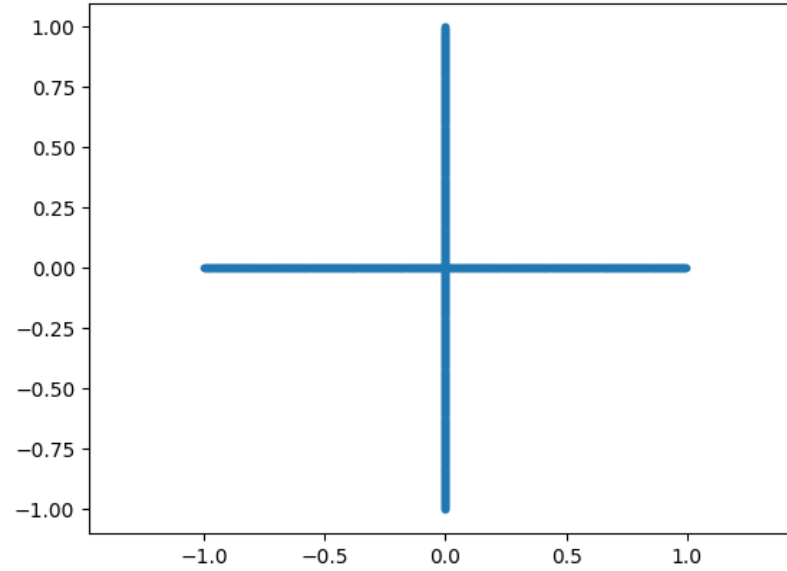
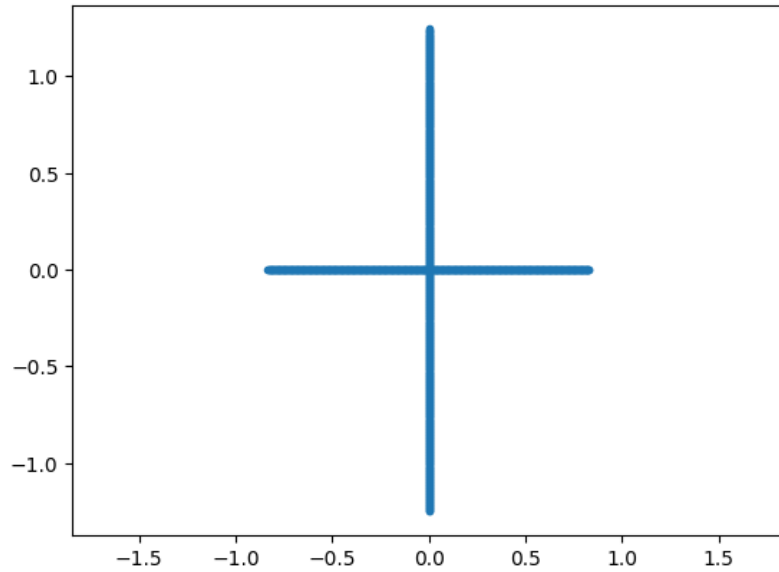
Reconstructions with varying # PCs



```
from sklearn.decomposition import PCA
x4 = x_train[y_train==4]
pca = PCA().fit(x4)
pca_x4 = pca.transform(x4)
plt.plot(pca.explained_variance_)
plt.show()
plt.plot(np.cumsum(pca.explained_variance_ratio_))
plt.show()
ims = np.zeros((10, 28*28))
nc = np.int32([0, 1, 5, 10, 20, 50, 100, 200, 400, 768])
for i in range(len(nc)):
    c = nc[i]
    ims[i] = pca.mean_ + np.matmul(pca_x4[0][:c], pca.components_[0:c])
display_mnist(ims, 1, 10)
```

Two minute break (3 questions)

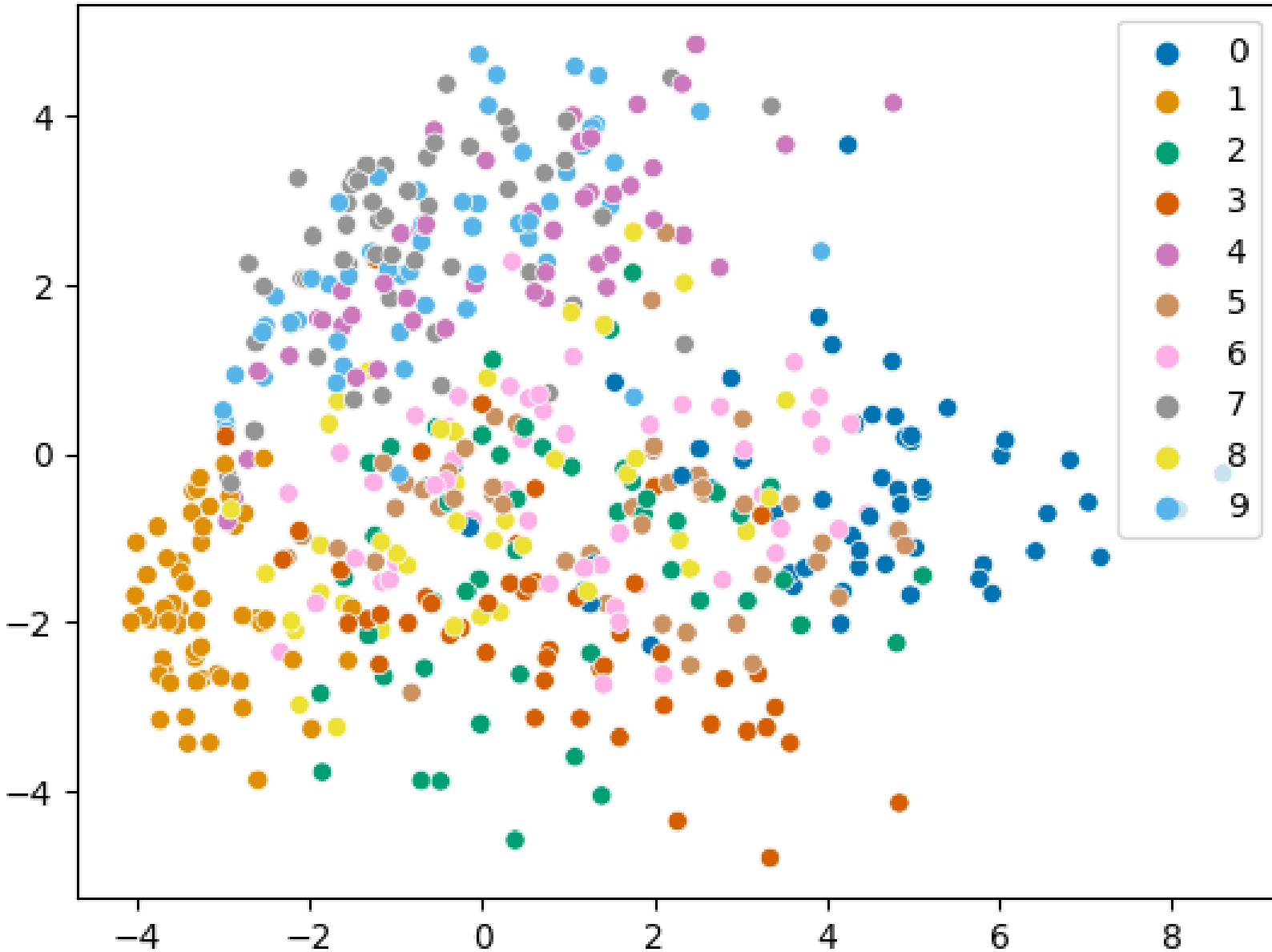
For each plot of data, what is the direction of the first principal component?



If X consists of N data points of d dimensions, what is the maximum number of PCA components that would be needed to perfectly reconstruct the data?

$$\text{Rank}(X) \leq \min(N, d)$$

PCA: MNIST at 2 dimensions



Note: I'm only plotting a subset

Non-Linear Scaling and Manifold Estimation

We may care less about being able to reconstruct each data point than representing the relationships between data points

- MDS: Preserve Euclidean pairwise distances
- Non-metric MDS: Preserve distance orderings
- ISOMAP: Define distances in terms of “geodesic” (graph-based) similarity

MultiDimensional Scaling (MDS)

- For all data points, solve for new coordinate positions that preserve some input set of pairwise distances
- Classic case (equations on right) uses Euclidean distance and has closed form solution
- More generally, distance can be defined arbitrarily (e.g. from user surveys)
- Major drawback is that the solution is most influenced by points that are far from each other

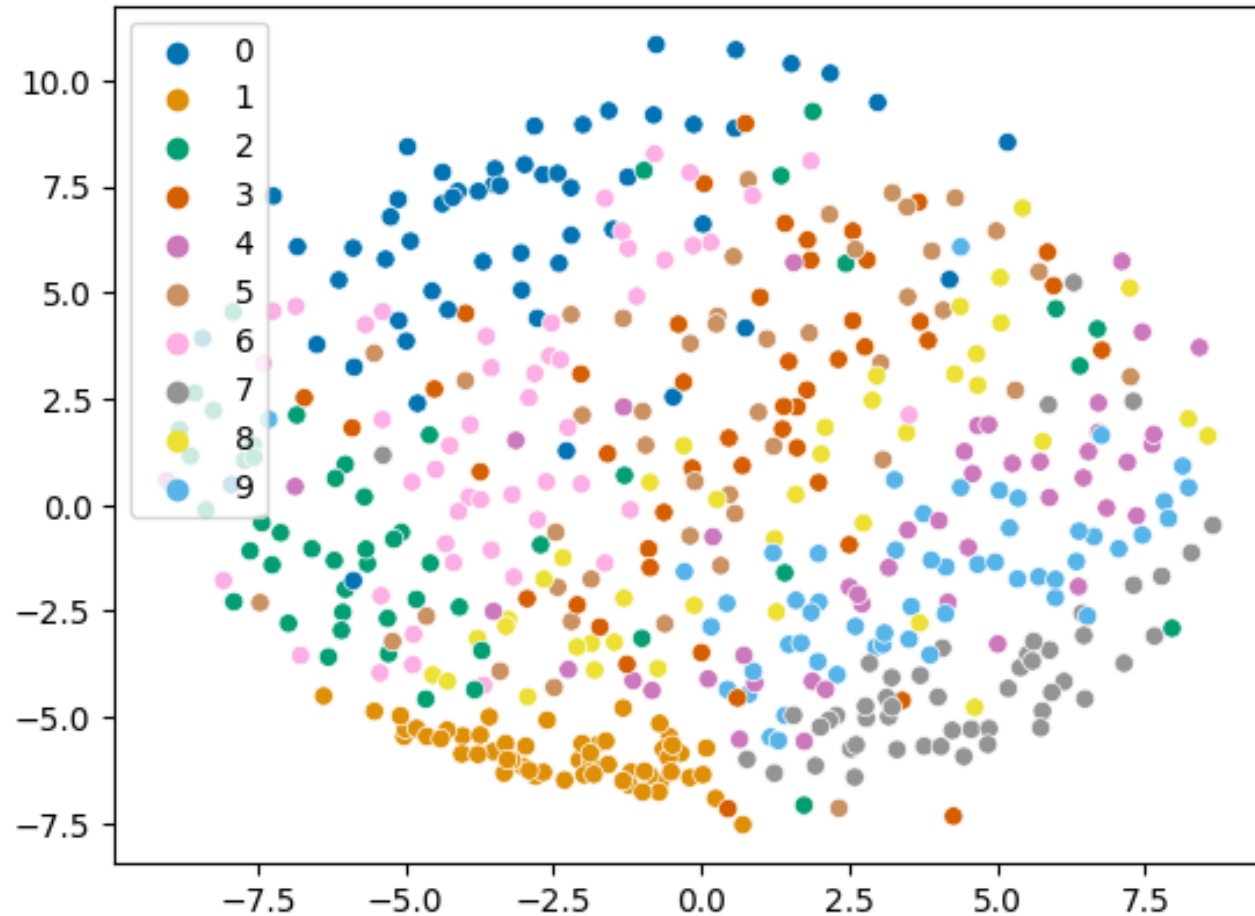
Solve for \mathbf{y} that minimizes
$$\sum_{ij} \left(D_{ij}^{(2)}(\mathbf{x}) - D_{ij}^{(2)}(\mathbf{y}) \right)^2$$

where
$$D_{ij}^{(2)}(\mathbf{x}) = (\mathbf{x}_i - \mathbf{x}_j)^T (\mathbf{x}_i - \mathbf{x}_j)$$

\mathbf{x} are old coordinates, \mathbf{y} are new coordinates

MDS on MNIST

- 30 PCA components
- MDS to 2 dimensions



```
from sklearn.manifold import MDS
```

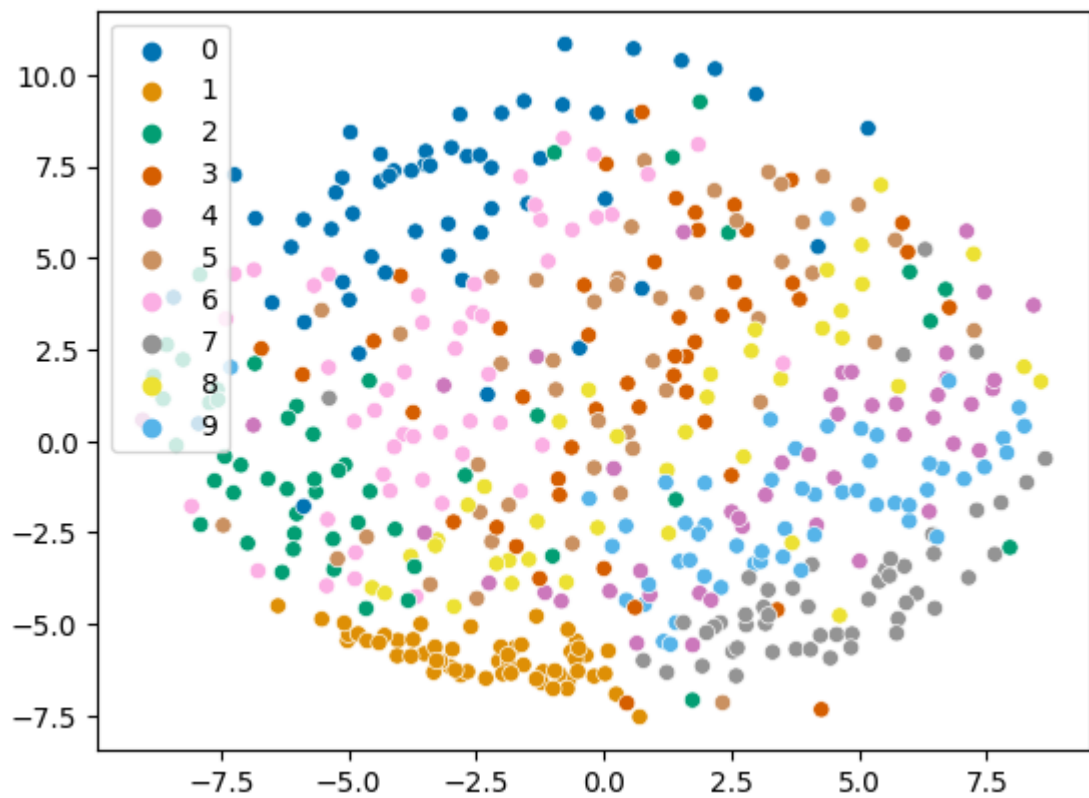
```
pca = PCA()  
pca.fit(x_train)  
x_pca = pca.transform(x_train)
```

```
ind = train_indices['s']  
x_mds = MDS(n_components=2, normalized_stress='auto').fit_transform(x_pca[ind, :30])  
sns.scatterplot(x=x_mds[ind,0], y=x_mds[ind,1], hue=y_train[ind], palette="colorblind")
```

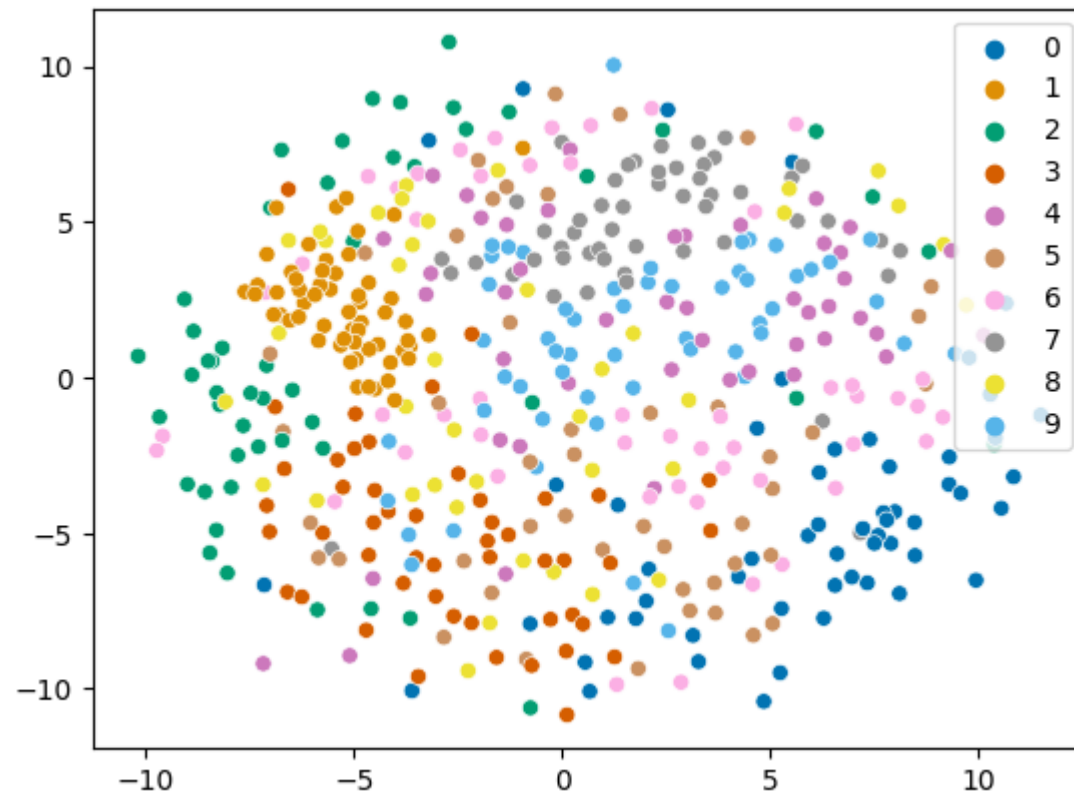
Note: For MDS and others, manifolds are fit on 500 pts for speed

MDS on MNIST

Pre-process with PCA to 30 dim



No PCA



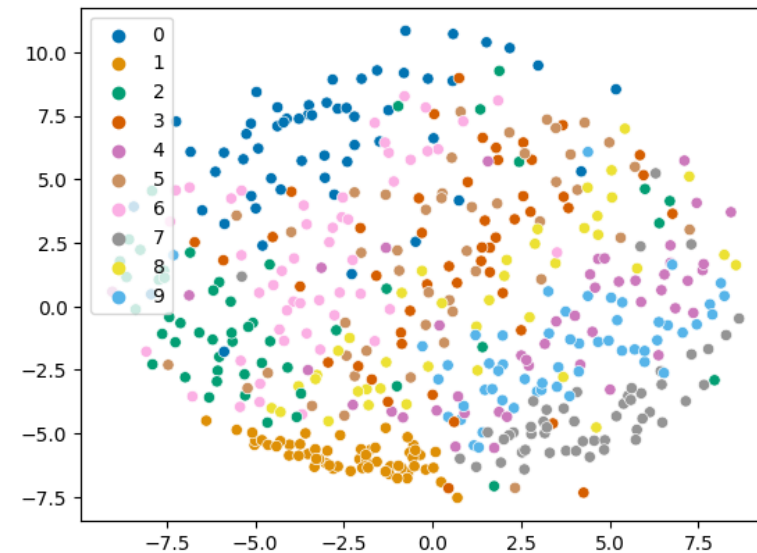
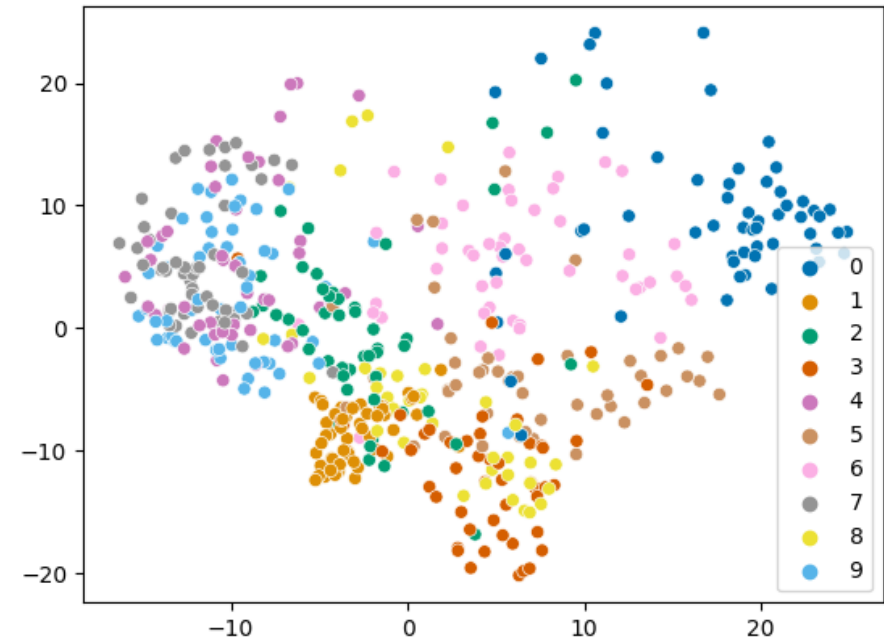
Non-metric MDS

- Optimize position of data points so that Euclidean distance preserves the ordering of input pairwise distances
- Requires only an order of dissimilarities
- Slow because this is a complicated optimization

ISOMAP

- Same as MDS but define distance in a graph
 - Compute adjacency graph (e.g. 5 nearest neighbors)
 - Distance is shortest path in graph between two points

MNIST ISOMAP (PCA: 30)



MNIST MDS
(PCA: 30)

t-SNE

Map to 2 or 3 dimensions while preserving similarities of nearby points

1. Assign probability p_{ij} that pairs of points are similar, e.g. with Gaussian weighted distance
2. Define similarity q_{ij} in new coordinates
3. Minimize KL divergence between p and q (i.e. they should have similar distributions) using gradient descent

For $i \neq j$, define

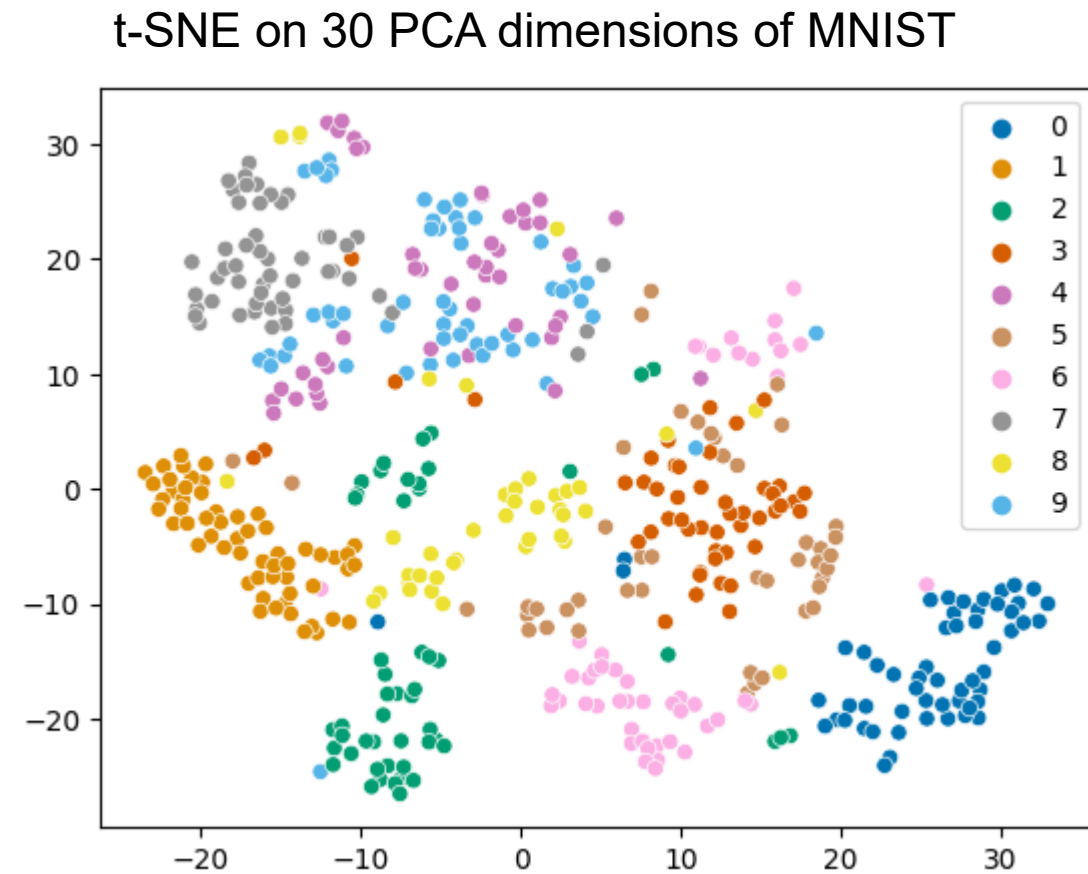
$$p_{j|i} = \frac{\exp(-\|\mathbf{x}_i - \mathbf{x}_j\|^2 / 2\sigma_i^2)}{\sum_{k \neq i} \exp(-\|\mathbf{x}_i - \mathbf{x}_k\|^2 / 2\sigma_i^2)}$$
$$p_{ij} = \frac{p_{j|i} + p_{i|j}}{2N}$$

$$q_{ij} = \frac{(1 + \|\mathbf{y}_i - \mathbf{y}_j\|^2)^{-1}}{\sum_k \sum_{l \neq k} (1 + \|\mathbf{y}_k - \mathbf{y}_l\|^2)^{-1}}$$

$$\text{KL}(P \parallel Q) = \sum_{i \neq j} p_{ij} \log \frac{p_{ij}}{q_{ij}}$$

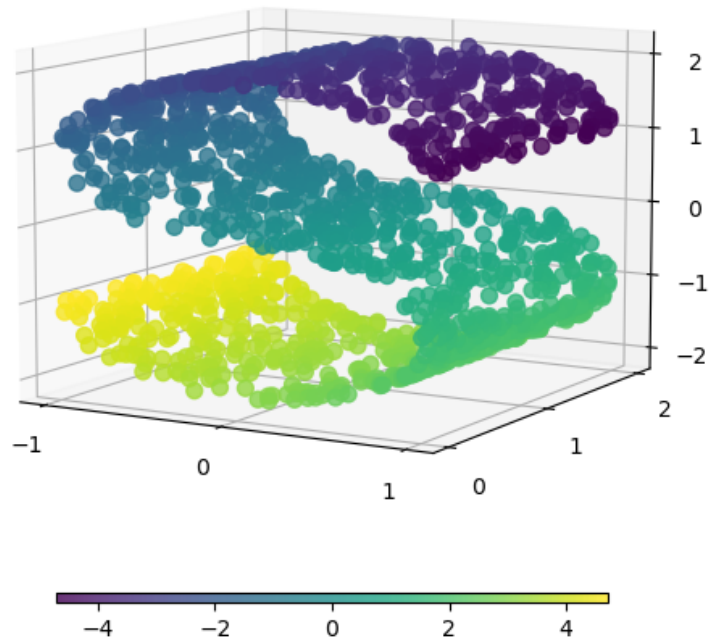
t-SNE

- In high dimensions, each point tends to be similarly distant to many points, so we often use PCA before applying t-SNE

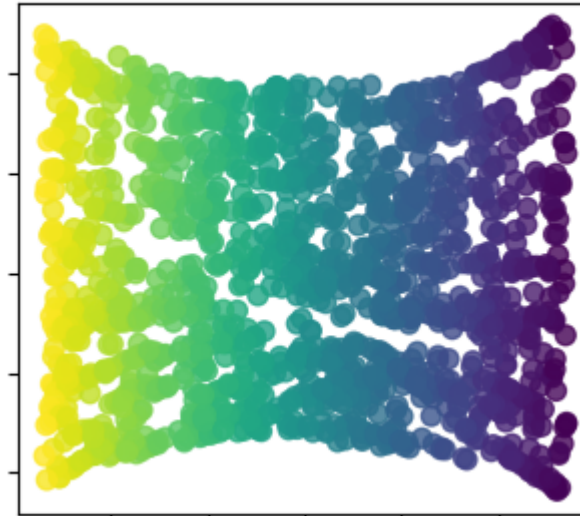


Comparison

Original S-curve samples



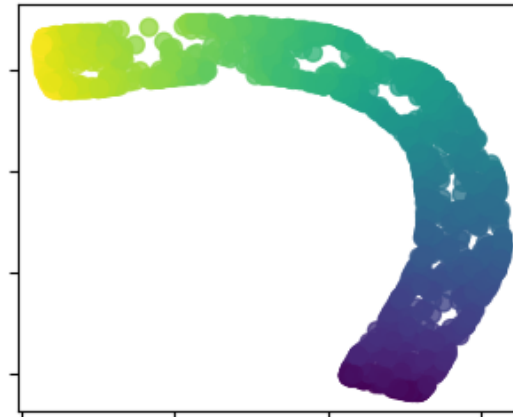
Isomap Embedding



Multidimensional scaling

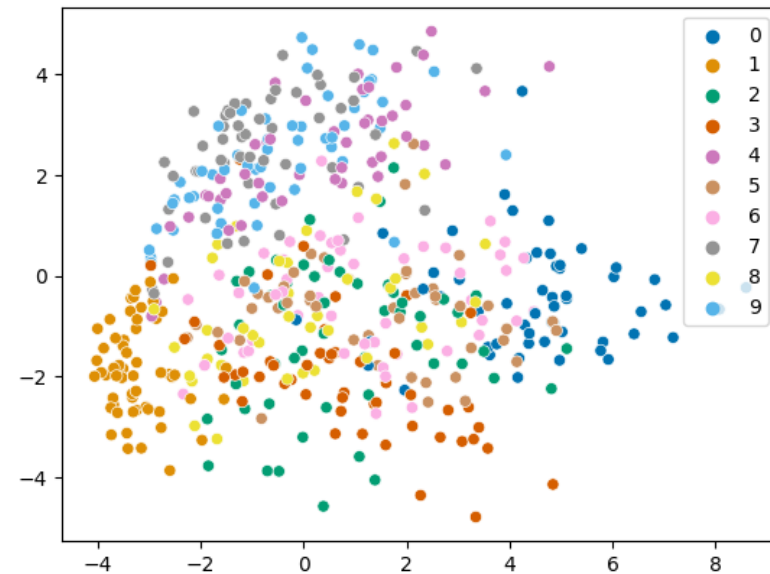


T-distributed Stochastic Neighbor Embedding

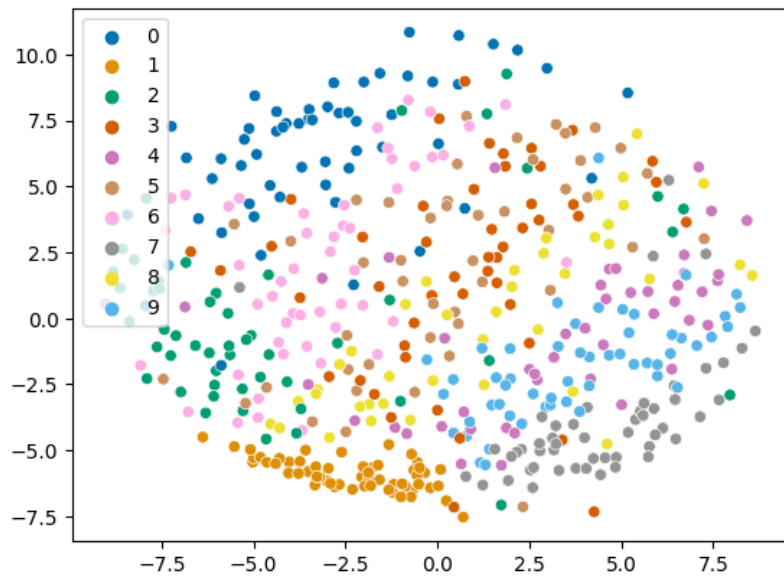


Comparison on MNIST

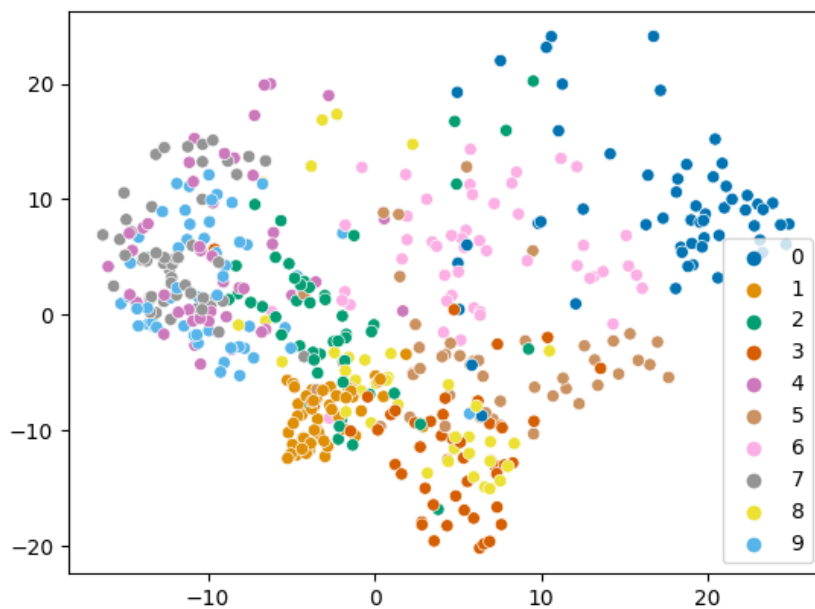
PCA



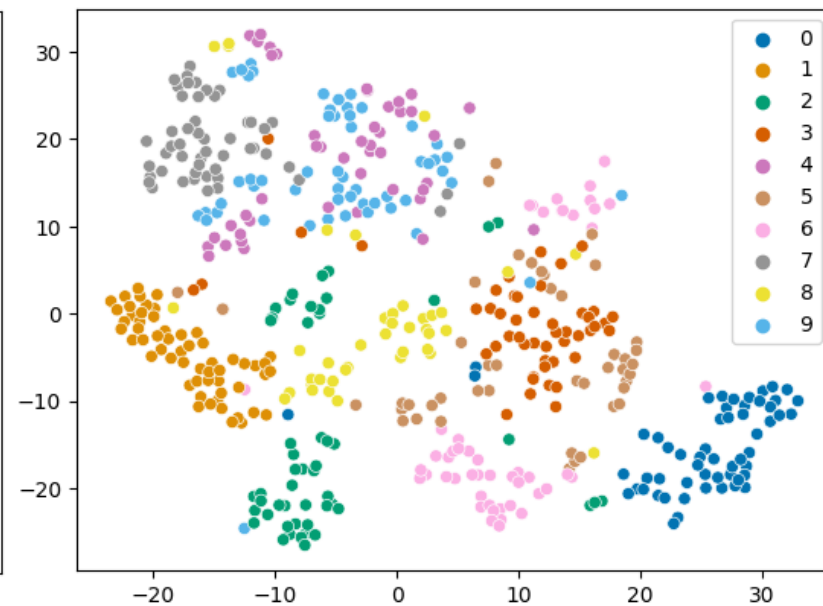
MDS



IsoMap

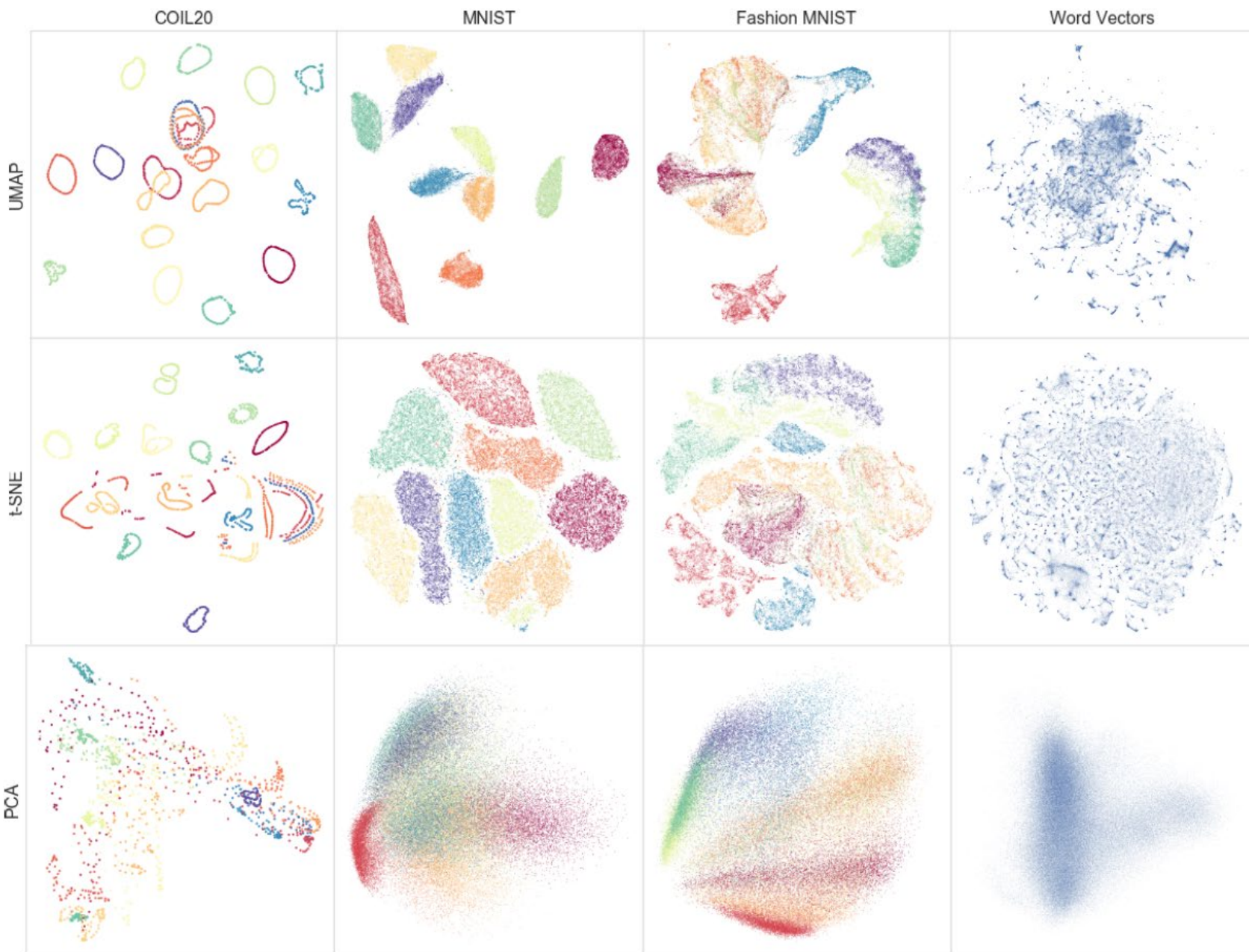


tSNE



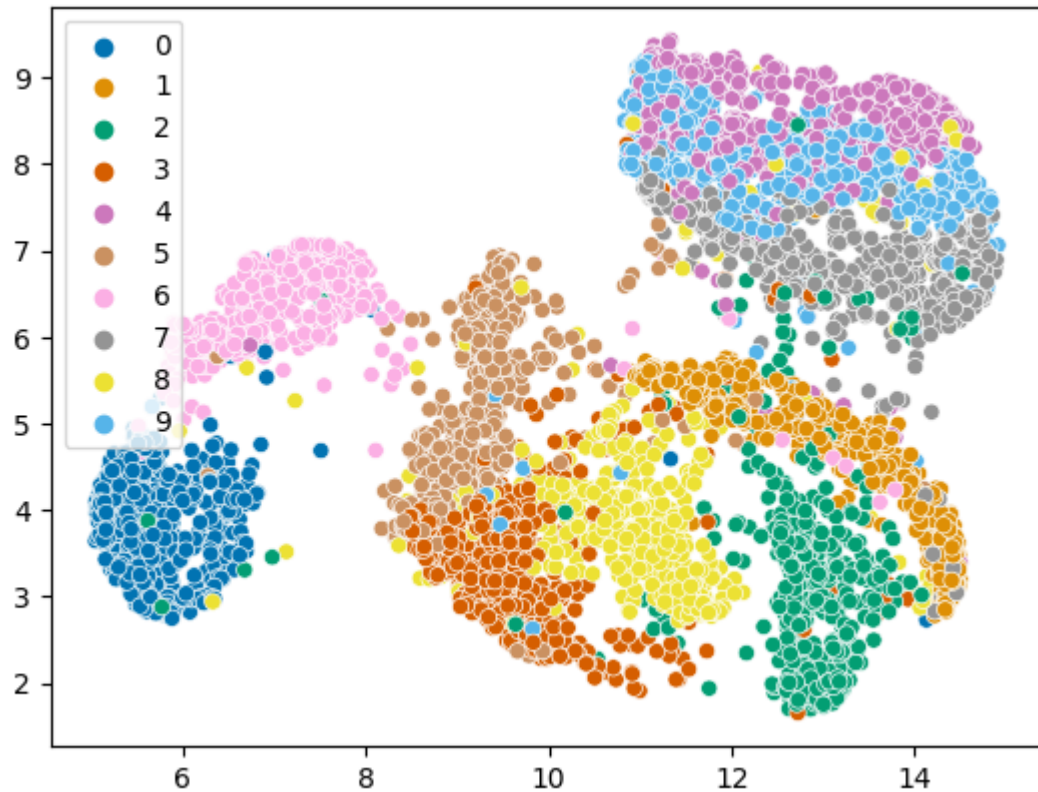
UMAP (McInnes, 2020)

- Assumes data is uniformly distributed on an underlying manifold that is locally connected. Goal is to preserve that local structure.
- Algorithm: relatively complicated, incorporates many ideas from other methods, has strong mathematical foundations
- Hyperparameters:
 - number of neighbors to consider
 - dimension of target embedding
 - desired separation between close points
 - number of training epochs

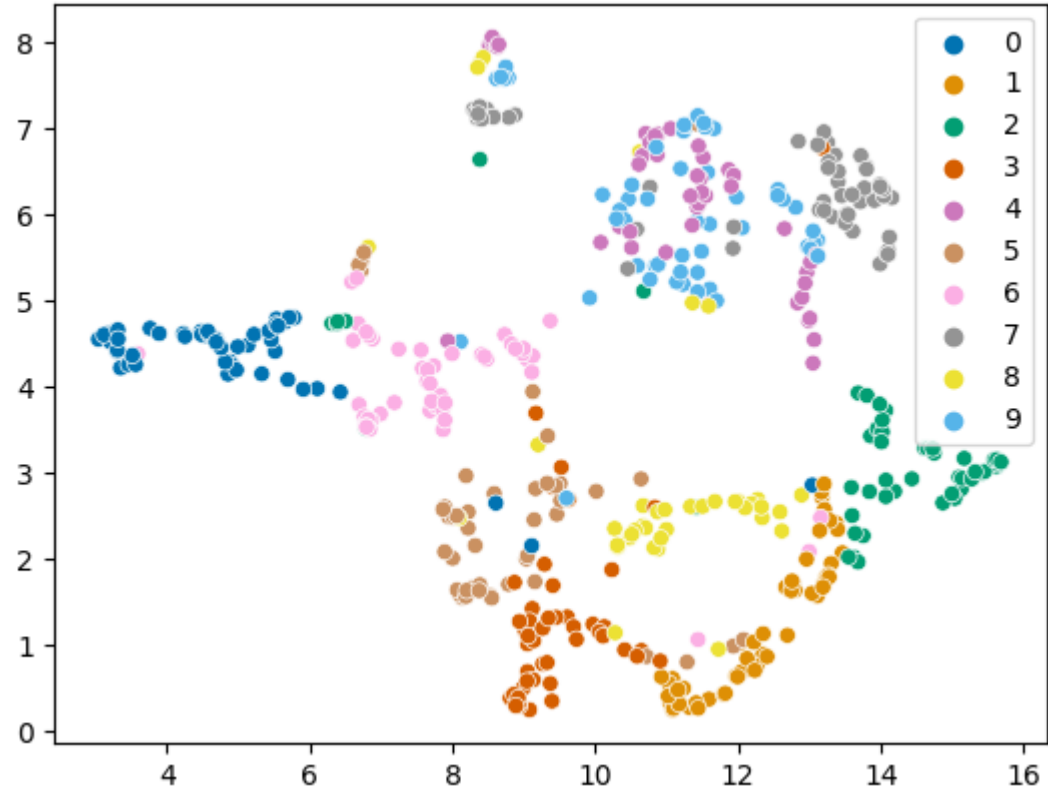


| | UMAP | t-SNE | Isomap |
|------------------------------------|------|--------|--------|
| Pen Digits (1797x64) | 9s | 17s | 2s |
| COIL20 (1440x16384) | 12s | 22s | 58s |
| COIL100 (7200x49152) | 85s | 810s | 3210s |
| scRNA (21086x1000) | 28s | 258s | 923s |
| Shuttle (58000x9) | 94s | 714s | - |
| MNIST (70000x784) | 87s | 1450s | - |
| F-MNIST (70000x784) | 65s | 934s | - |
| Flow (100000x17) | 102s | 1135s | - |
| Google News (200000x300) | 361s | 16906s | - |

UMAP on MNIST



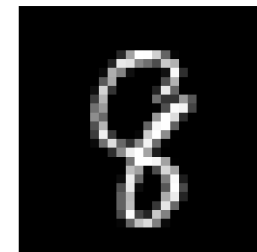
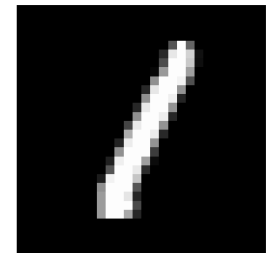
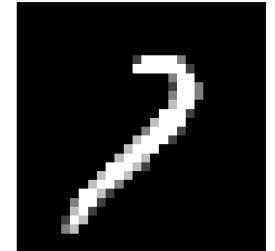
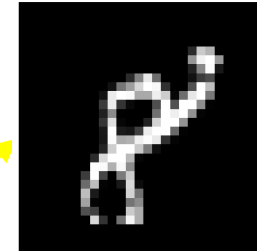
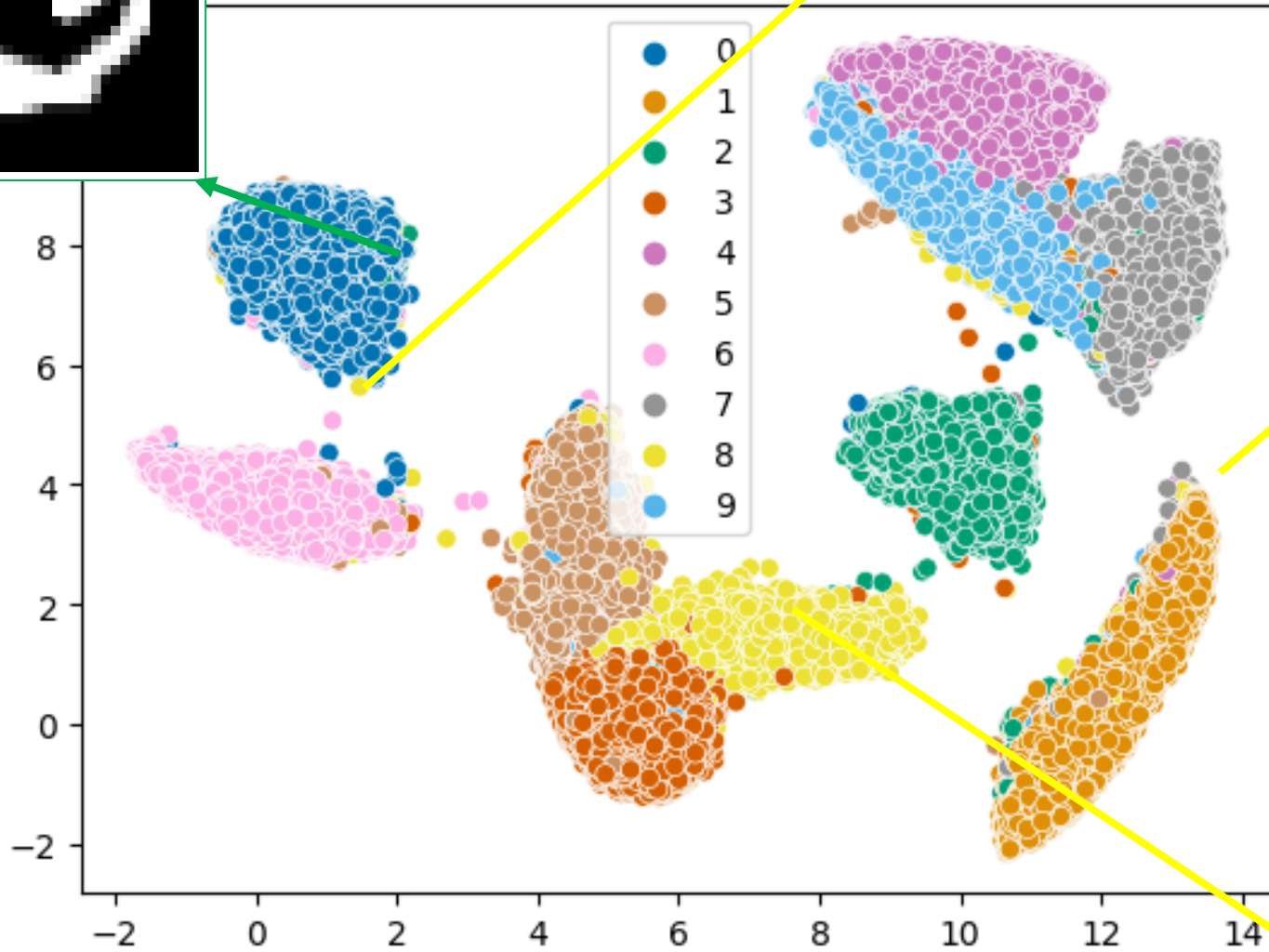
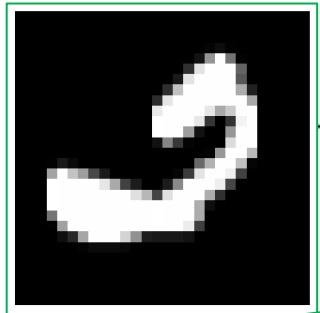
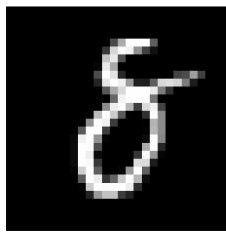
5000 points, 100 neighbors, 34s



500 pts, 10 neighbors

```
#!/pip install umap-learn
from umap import UMAP
ind = train_indices['s']
x_umap = UMAP(n_components=2, n_neighbors=100).fit_transform(x[ind, :30])
sns.scatterplot(x=x_umap[ind,0],y=x_umap[ind,1], hue=y_train[ind], palette="colorblind")
```

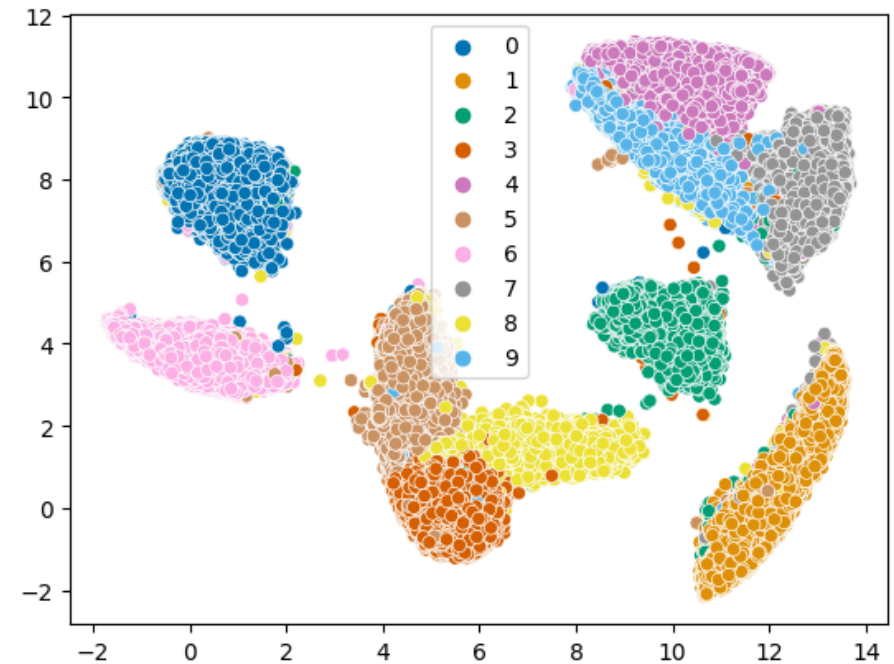
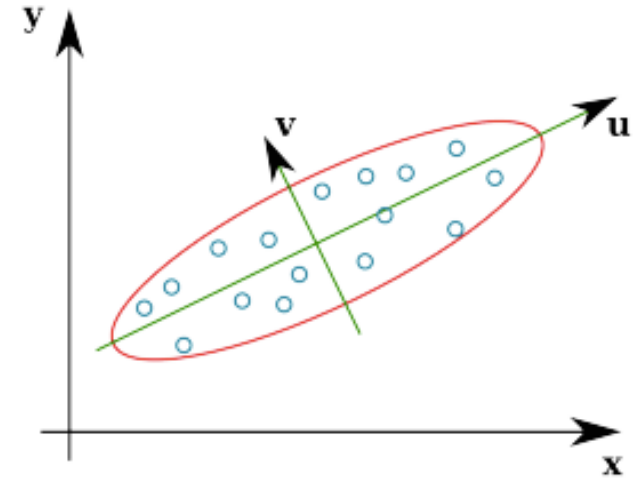
UMAP on MNIST



50,000 points, 100 neighbors, 200s

Things to remember

- PCA reduces dimensions by linear projection
 - Preserves variance to reproduce data as well as possible, according to mean squared error
 - May not preserve local connectivity structure or discriminative information
- Other methods try to preserve relationships between points
 - MDS: preserve pairwise distances
 - IsoMap: MDS but using a graph-based distance
 - t-SNE: preserve a probabilistic distribution of neighbors for each point (also focusing on closest points)
 - UMAP: incorporates k-nn structure, spectral embedding, and more to achieve good embeddings relatively quickly



Next class: Linear Regression and Regularization