Unsupervised Machine Learning

Section 9.1: Dimension Reduction

Dimension Reduction

- Machine learning problems may involve datasets in 100s or 1000s of dimensions
- More dimensions generally means slower computation
- Dimension Reduction attempts to map feature vectors into a lower dimensional space while retaining as much information as possible

Dimension Reduction

Two Approaches in this Section:

- Principal Component Analysis (PCA):
  - “Linear” approach
  - Cool application of singular value decomposition
- Autoencoding:
  - “Nonlinear” approach
  - Uses techniques from supervised learning to learn new representation of data

Section 9.2: Principal Component Analysis: Algorithm

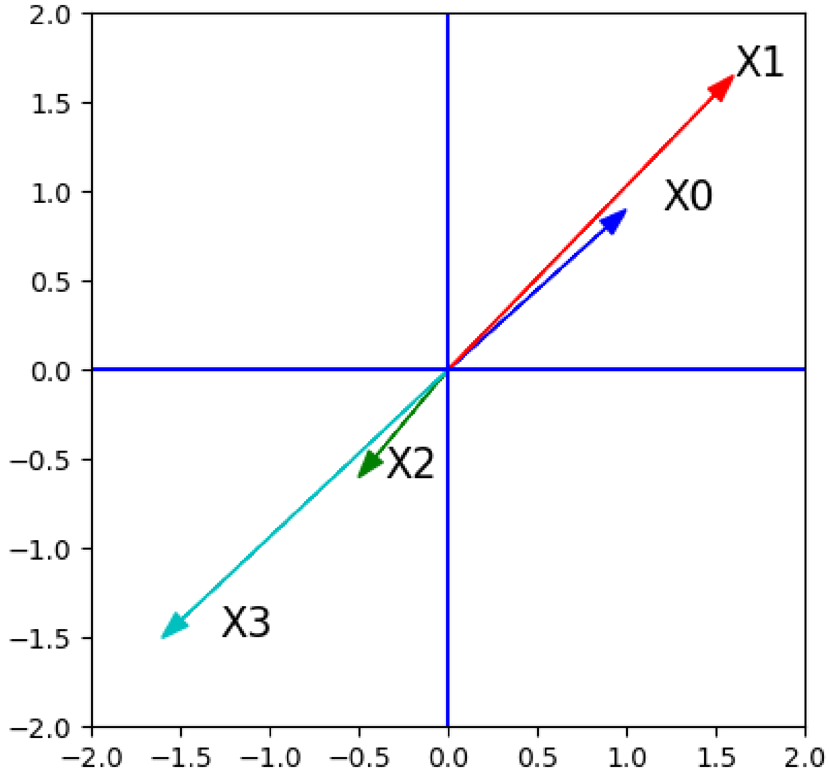
PCA and SVD

- SVD is used to find a new basis $u_0, u_1, \ldots, u_{d-1}$
- Relevance of each basis vector (or component) is determined by its corresponding singular value, which are ordered in decreasing value
- Idea is to project data onto space spanned by first K basis vectors $u_0, u_1, \ldots, u_{K-1}$, choosing K to retain sufficient information

Example: 4 Data Points in 2D

- Consider 4 data points in 2 dimensions

$$X = [X_0 \quad X_1 \quad X_2 \quad X_3] = \begin{bmatrix} 1 & 1.6 & -0.5 & -1.6 \\ 0.9 & 1.65 & -0.6 & -1.5 \end{bmatrix}$$



SVD of Data Set

- Compute SVD of X

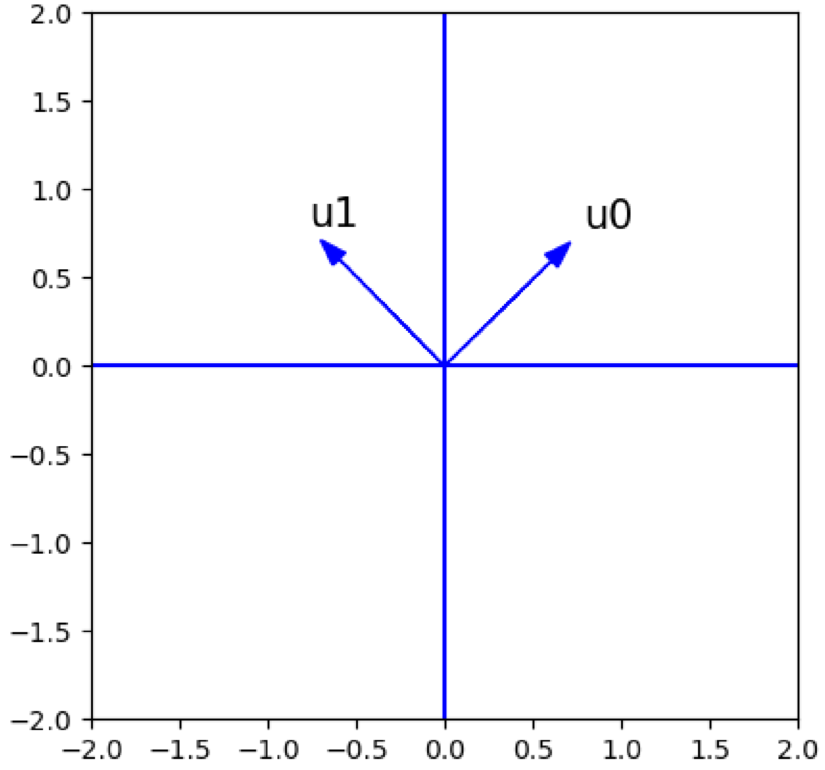
$$X = U\Sigma V^T$$

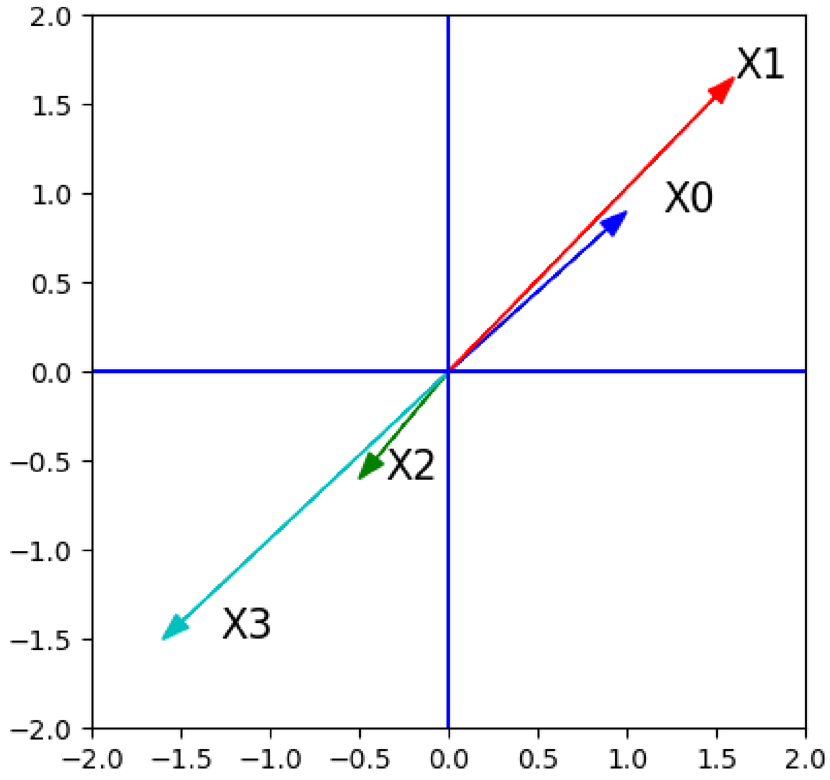
$$X = U\Sigma V^T = \begin{bmatrix} u_0 & u_1 \end{bmatrix} \begin{bmatrix} \sigma_0 & 0 & 0 & 0 \\ 0 & \sigma_1 & 0 & 0 \end{bmatrix} \begin{bmatrix} v_0^T \\ v_1^T \\ v_2^T \\ v_3^T \end{bmatrix}$$

$$X = \begin{bmatrix} 0.71 & -0.70 \\ 0.70 & -0.71 \end{bmatrix} \begin{bmatrix} 3.54 & 0 & 0 & 0 \\ 0 & 0.12 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0.38 & 0.65 & -0.22 & -062 \\ -0.47 & 0.46 & -0.63 & 0.41 \\ -0.26 & 0.58 & 0.74 & 0.19 \\ 0.75 & 0.16 & -0.03 & 0.64 \end{bmatrix}$$

New Coordinate system with basis u0 and u1

- Consider new coordinate system using u0 and u1 as basis





Data points in u0-u1 Coordinate System

- Original X

$$X = \begin{bmatrix} 1 & 1.6 & -0.5 & -1.6 \\ 0.9 & 1.65 & -0.6 & -1.5 \end{bmatrix}$$

- Data points in u0 and u1 coordinate system

$$X_U = \Sigma V^T = \begin{bmatrix} 3.54 & 0 & 0 & 0 \\ 0 & 0.12 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0.38 & 0.65 & -0.22 & -0.62 \\ -0.47 & 0.46 & -0.63 & 0.41 \\ -0.26 & 0.58 & 0.74 & 0.19 \\ 0.75 & 0.16 & -0.03 & 0.64 \end{bmatrix}$$

$$X_U = \begin{bmatrix} 1.34 & 2.30 & -0.78 & -2.20 \\ -0.06 & -0.06 & -0.08 & 0.05 \end{bmatrix}$$

Row0 gives units of u0 for each data point

Row1 gives units of u1 for each data point

Reduce Dimensions

- Only keep information in u0 direction (1 dimension)
- Data points in reduced number of dimensions

Units of u0 for each data point

$$R = [\sigma_0][v_0^T] = [3.54][0.38 \quad 0.65 \quad -0.22 \quad -0.62] = [1.34 \quad 2.30 \quad -078 \quad -2.20]$$

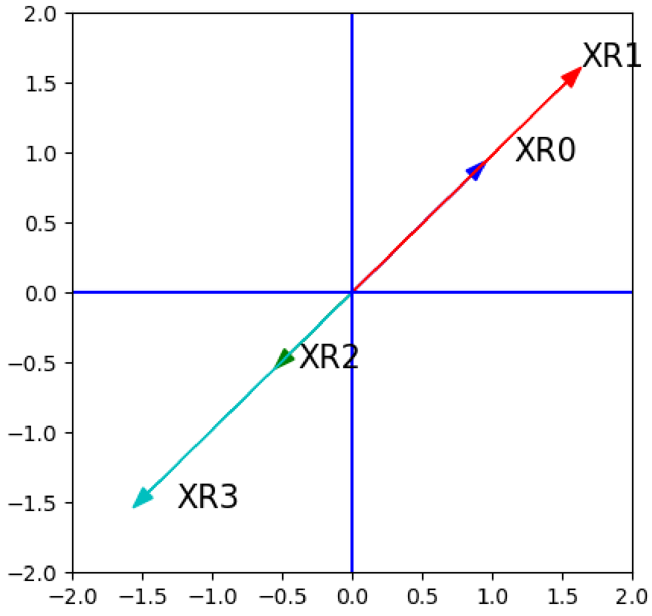
Reconstruct in Original Space

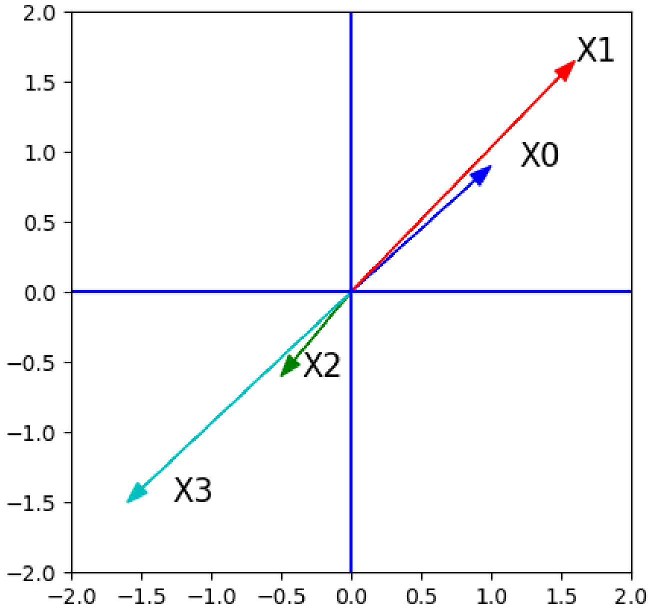
- Can reconstruct data points in 2d space using information in u0 direction only

$$X_R = [u_0][\sigma_0][v_0^T]$$

$$X_R = \begin{bmatrix} 0.71 \\ 0.70 \end{bmatrix} [3.54][0.38 \quad 0.65 \quad -0.22 \quad -0.62]$$

$$X_R = \begin{bmatrix} 0.96 & 1.64 & -0.55 & -1.56 \\ 0.94 & 1.61 & -0.54 & -1.54 \end{bmatrix} \qquad X = \begin{bmatrix} 1 & 1.6 & -0.5 & -1.6 \\ 0.9 & 1.65 & -0.6 & -1.5 \end{bmatrix}$$





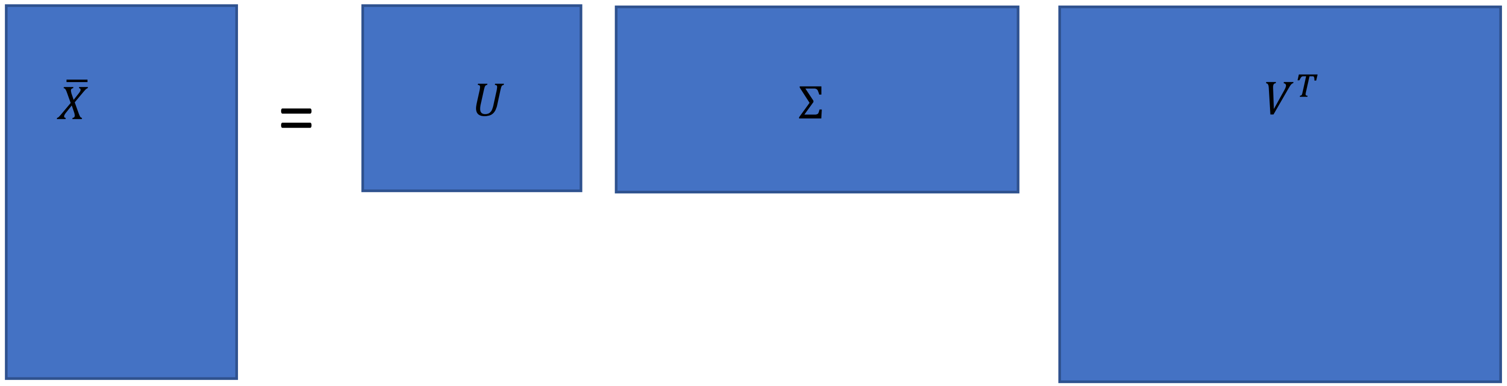
PCA Algorithm

PCA Algorithm:

- Start with data matrix X (number of features x number of samples)

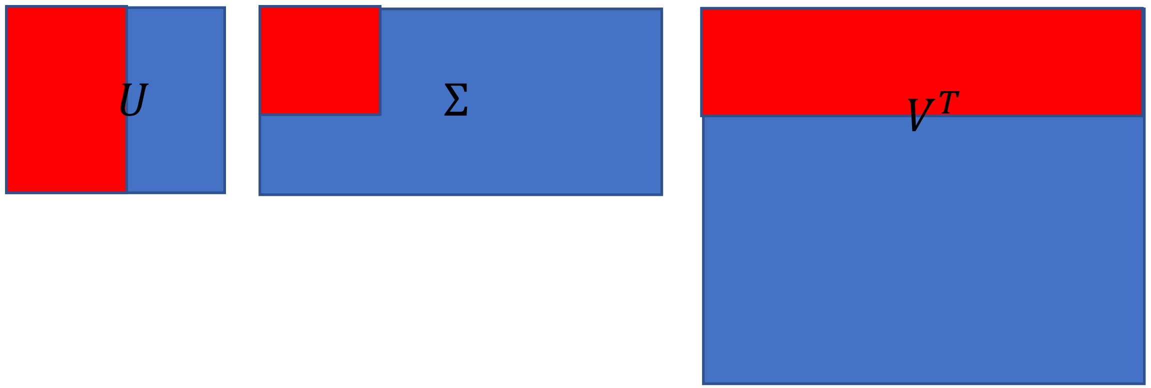
(1) Subtract mean of data $X - X_{mean}$ (translate data points by same amount)

(2) Compute the singular value decomposition of $\bar{X} = X - X_{mean} = U\Sigma V^T$



PCA Algorithm

(3) Pick K principal components



(first K columns of U, first K diagonal entries of Σ, first K rows of V^T)

(4) Data points in K dimensional $(u_0, \dots, u_{K-1})$ coordinate system given by:

$$R = \begin{bmatrix} \sigma_0 & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & \sigma_{K-1} \end{bmatrix} \begin{bmatrix} v_0^T \\ \vdots \\ v_{K-1}^T \end{bmatrix}$$

PCA Algorithm

(5) Reconstructed $\bar{X}$ (keeping only K principal components) in original space

$$\bar{X}_R = \begin{bmatrix} u_0 & \cdots & u_{K-1} \end{bmatrix} \begin{bmatrix} \sigma_0 & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & \sigma_{K-1} \end{bmatrix} \begin{bmatrix} v_0^T \\ \vdots \\ v_{K-1}^T \end{bmatrix}$$

(6) Add back mean to get reconstructed X (keeping only K principal components)

$$X_R = \bar{X}_R + X_{mean}$$

Principal Component Analysis: Choosing K

- Covariance matrix for data set:

$$Cov = \frac{1}{M}(X - X_{mean})(X - X_{mean})^{T}$$

- Total variance is sum of eigenvalues of Covariance matrix, which is same as sum of square of singular values of $\frac{1}{\sqrt{M}}(X - X_{mean})$
- Define

$$Variance(K) = \frac{1}{M}\sum_{k=0}^{K-1}\sigma_k^2$$

- Instead of specifying K directly, choose smallest number of principal components so that Variance(K)/Variance(M) is at least as large as specified percentage

Section 9.3: PCA for MNIST Dataset

MNIST Digits Dataset

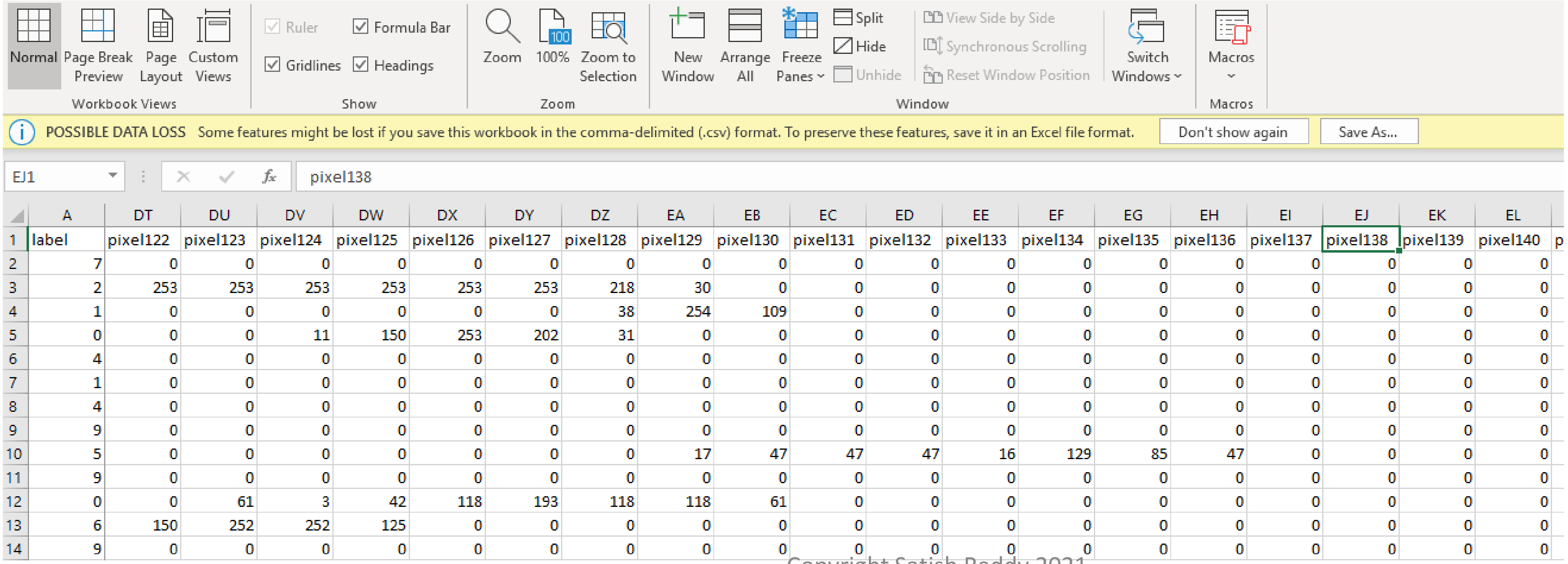
- Thousands of handwritten digit images with 28x28 resolution
- Data Source: http://yann.lecun.com/exdb/mnist/
- Used extensively for testing machine learning algorithms

Collage of 160 individual digit images

By Josef Steppan - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=64810040

MNIST Digits – Format of Data Files

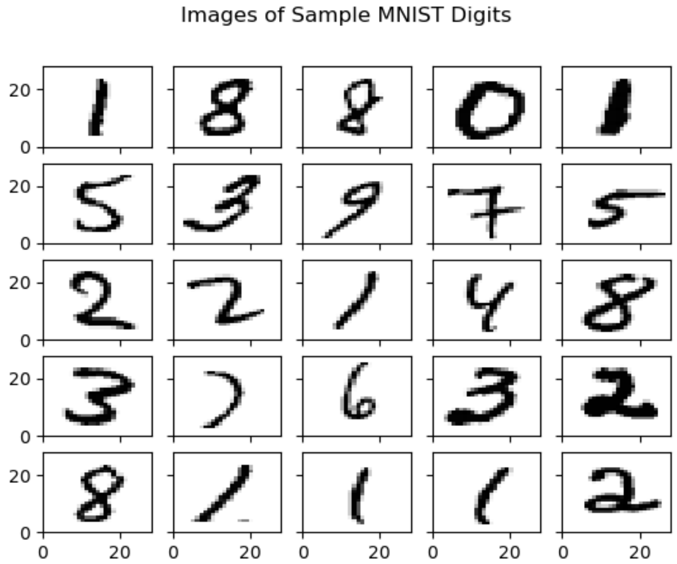
- Each row represents label and intensities for one image
  - First column is the digit label (0,1,…,9)
  - Columns 2 – 785 are the intensities
  - Take transpose to convert feature matrix and to correct format
  - Standard practice is to divide pixel values by 255 so between 0 and 1



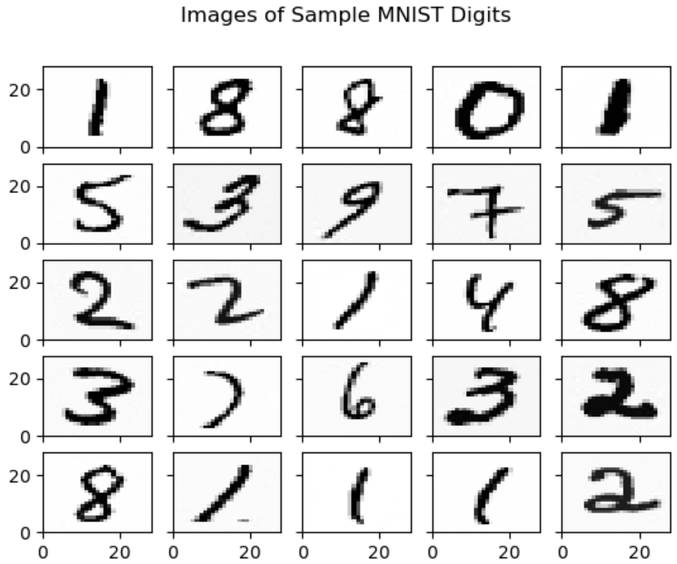
	A	DT	DU	DV	DW	DX	DY	DZ	EA	EB	EC	ED	EE	EF	EG	EH	EI	EJ	EK	EL
1	label	pixel122	pixel123	pixel124	pixel125	pixel126	pixel127	pixel128	pixel129	pixel130	pixel131	pixel132	pixel133	pixel134	pixel135	pixel136	pixel137	pixel138	pixel139	pixel140
2	7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	2	253	253	253	253	253	253	218	30	0	0	0	0	0	0	0	0	0	0	0
4	1	0	0	0	0	0	0	38	254	109	0	0	0	0	0	0	0	0	0	0
5	0	0	0	11	150	253	202	31	0	0	0	0	0	0	0	0	0	0	0	0
6	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	5	0	0	0	0	0	0	0	17	47	47	47	16	129	85	47	0	0	0	0
11	9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
12	0	0	61	3	42	118	193	118	118	61	0	0	0	0	0	0	0	0	0	0
13	6	150	252	252	125	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
14	9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

PCA for MNIST: Original and Reconstructed

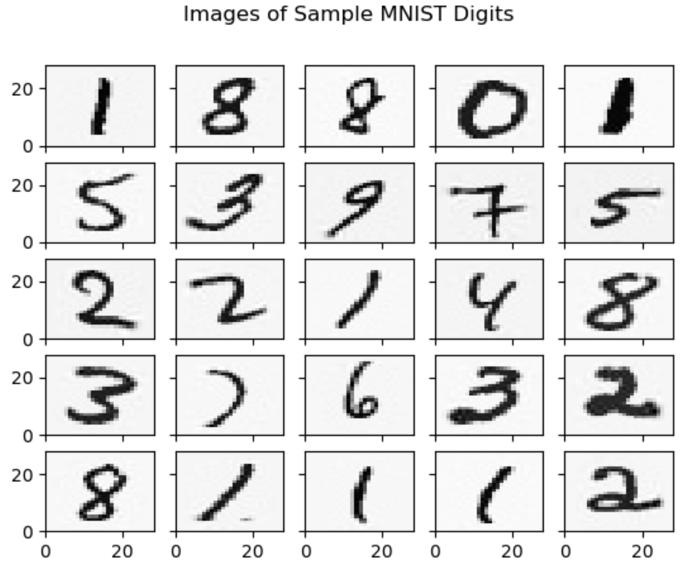
Original
Dataset: 6000 image
784 dimensions
Plot of 25 images



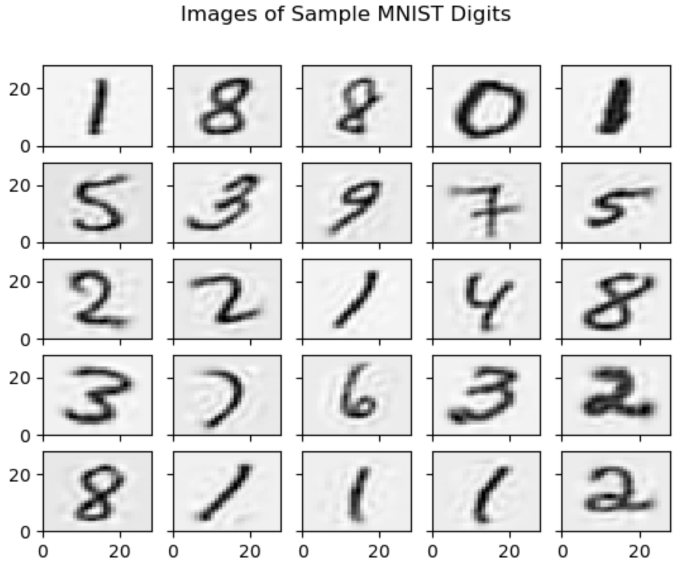
Reconstructed
99.9% Variance
476 dimensions



Reconstructed
99% Variance
323 dimensions



Reconstructed
90% Variance
84 dimensions



Section 9.4: PCA: Code Design

PCA Code Design

- This section contains design information for the PCA Code
- Design based on theory in Section 10.1
- Code located at UnsupervisedML/Code/Clustering/pca.py
- Stop video here, if you would like to do code design yourself

PCA Code Design: To Do

Component	Description
class pca	class for Principal Component Analysis
driver_pca	driver for pca example
load_mnist	Function for loading mnist_data set

pca class: Principal Variables

Variable	Type	Description
self.U	2d numpy array	U from SVD of $X - X_{mean}$
self.Sigma	1d numpy array	Singular values of $X - X_{mean}$
self.Vh	2d numpy array	V^T from SVD of $X - X_{mean}$

pca class – Key Methods

Method	Input	Description
__init__		Constructor for pca class
fit	X (2d np.array)	Computes SVD of X – Xmean and stores results in self.U, self.Sigma, and self.vh Return: nothing
get_dimension	variance_capture (float)	Computes number of principal components to capture specified proportion of variance Return: number of principal components K
data_reduced_ dimension	**kwargs reduced_dim (integer) variance_capture (float)	Computes the dataset in the reduced number of dimensions. User specifies either reduced_dim directly or the proportion of variance to be captured. Return: R (2d numpy array)
data_reconstructed	**kwargs reduced_dim (integer) variance_capture (float)	Computes the reconstructed dataset using only K principal components. User specifies either reduce_dim directly or the proportion of variance to be captured. Return: X_R

Section 9.5: PCA: Code Walkthrough

PCA: Code Walkthrough

- Code is located in:

  Folder: UnsupervisedML/Code/Clustering

  Files: pca.py, driver_pca.py, load_mnist.py

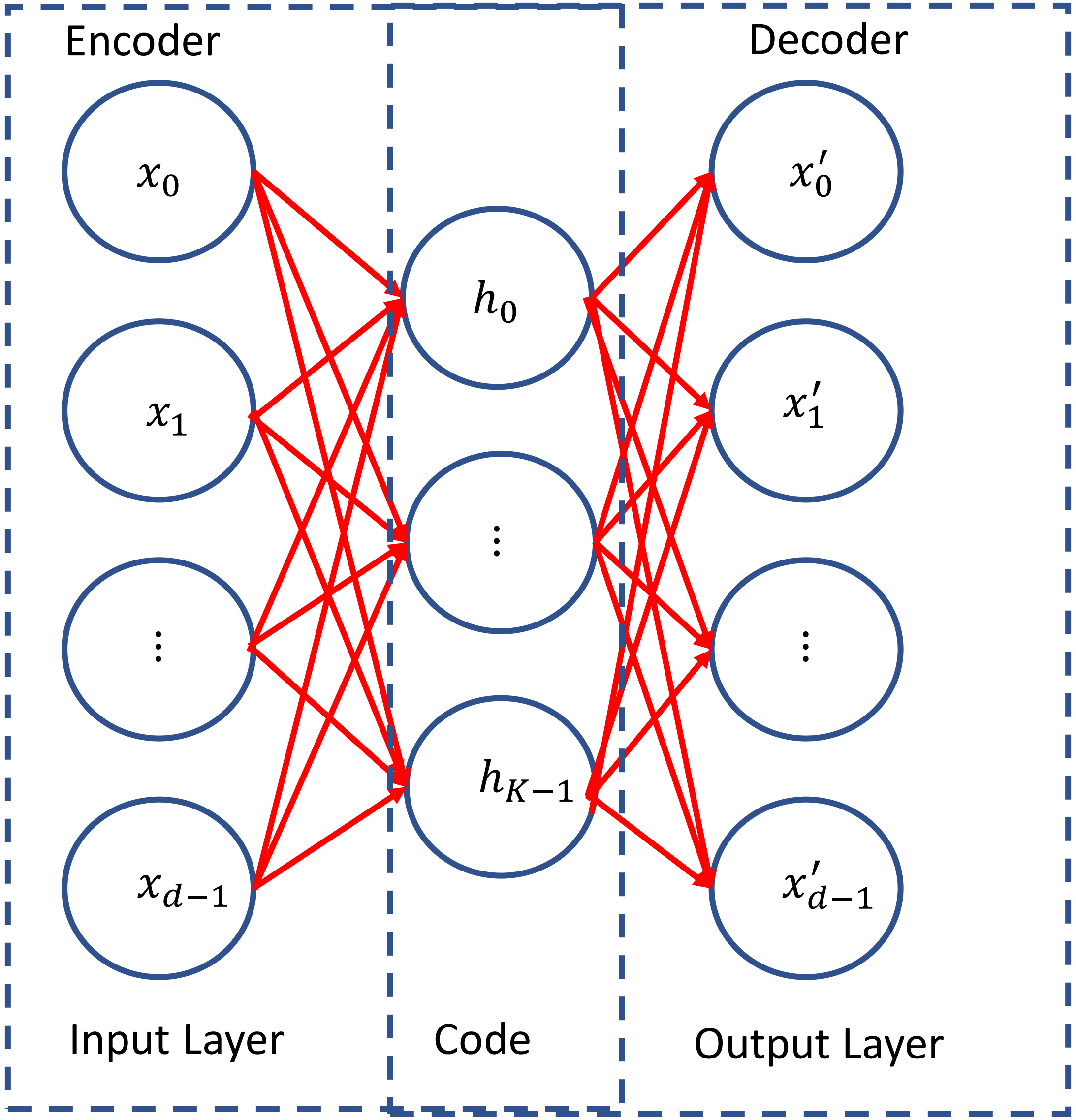
- Stop video here, if you would like to do coding yourself before seeing my implementation

Section 9.6: Autoencoders

What is an Autoencoder?

- An Autoencoder is type of Artificial Neural Network for learning efficient representations of the feature vector in unsupervised manner
- Can be used to reduce dimensions
- Whereas PCA is based on SVD is linear, Autoencoder is a nonlinear approach

Sample Autoencoder Neural Network



- Input: (d dimensions)

$$X = \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_{d-1} \end{bmatrix}$$

- Encoder: Map X to H (K dimensions). “Code” H is a new feature vector representation of X

$$H = \begin{bmatrix} h_0 \\ \vdots \\ h_{K-1} \end{bmatrix}$$

- Decoder: Map H to X’ (d dimensions). X’ is a reconstructed version of X

$$X' = \begin{bmatrix} x'_0 \\ x'_1 \\ \vdots \\ x'_{d-1} \end{bmatrix}$$

Autoencoder: Determining Mappings

- Encoder:

$$H = f(WX + b)$$

W is Kxd matrix, b is Kx1 vector, f is activation function

- Decoder:

$$X' = g(W'H + b')$$

W’ is dxK matrix, b’ is dx1 vector, g is activation function

- Given a dataset of M-1 datapoints: $X_0, X_1, \ldots, X_{M-1}$ for fixed activation functions f, and g, determine unknown W, b, W’, b’ by minimizing mean squared error loss function:

$$Loss = \frac{1}{M} \sum_{i=0}^{M-1} dist(X_i', X_i)^2$$

Autoencoder: Notes

- Autoencoder set up is similar to Supervised Learning
  - Key difference is that there is no label Y for each data point
- Determine W, b, W’, b’ using optimizer such as Gradient Descent and backpropagation to determine derivatives
- Can use familiar activation functions such as sigmoid, Relu for f and g
- “Code” representation H is related nonlinearly to X (if nonlinear activation functions used)
- Use code representation H in unsupervised learning algorithm instead of original X