

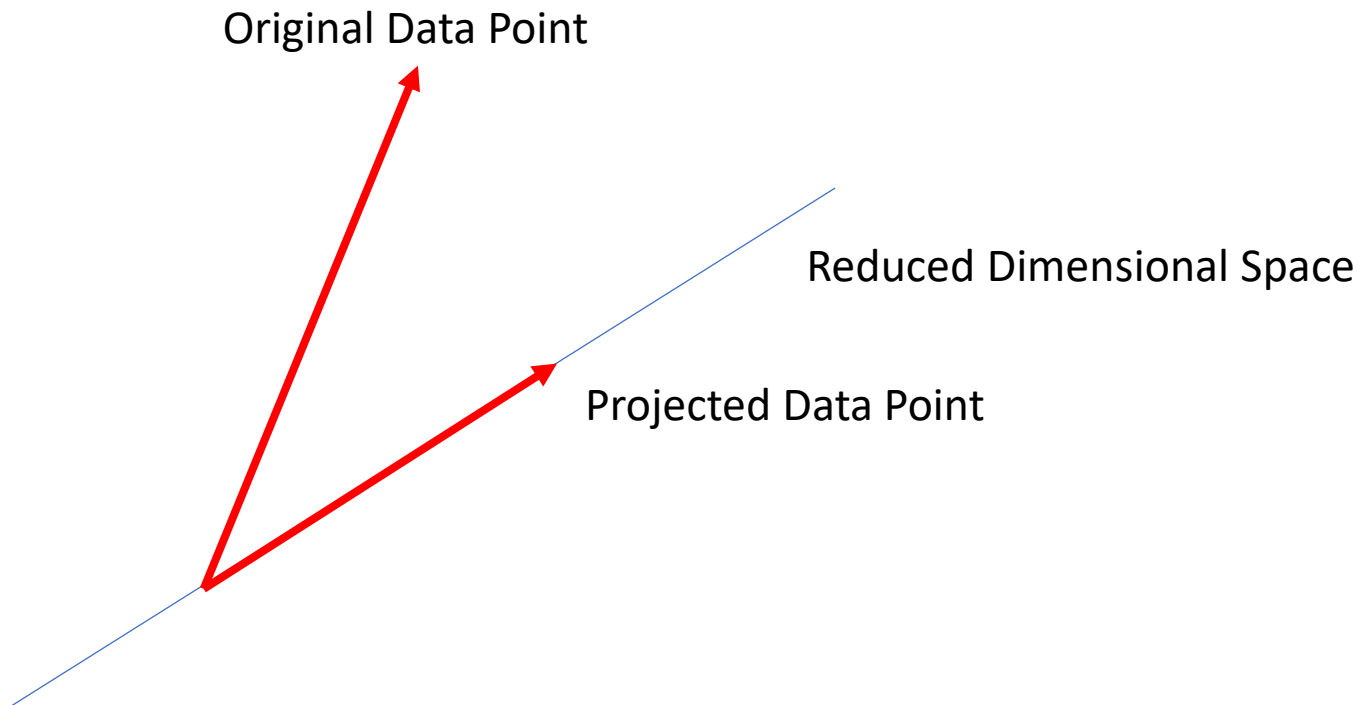
Unsupervised Machine Learning with Python

Section 9.0: Dimension Reduction Overview

Purpose of Dimension Reduction

- Machine learning problems may have datasets with 1000s of dimensions
- More dimensions generally means slower computation
- Dimension Reduction attempts to map datasets into a lower dimensional space while retaining as much information as possible
- See [UnsupervisedML_Resources.pdf](#) for links to additional resources

Dimension Reduction



- Picture often seen in Linear Algebra courses
- Use dimension reduction techniques to find Projected Data Point and Reduced Dimensional Space

Dimension Reduction for MNIST Digits

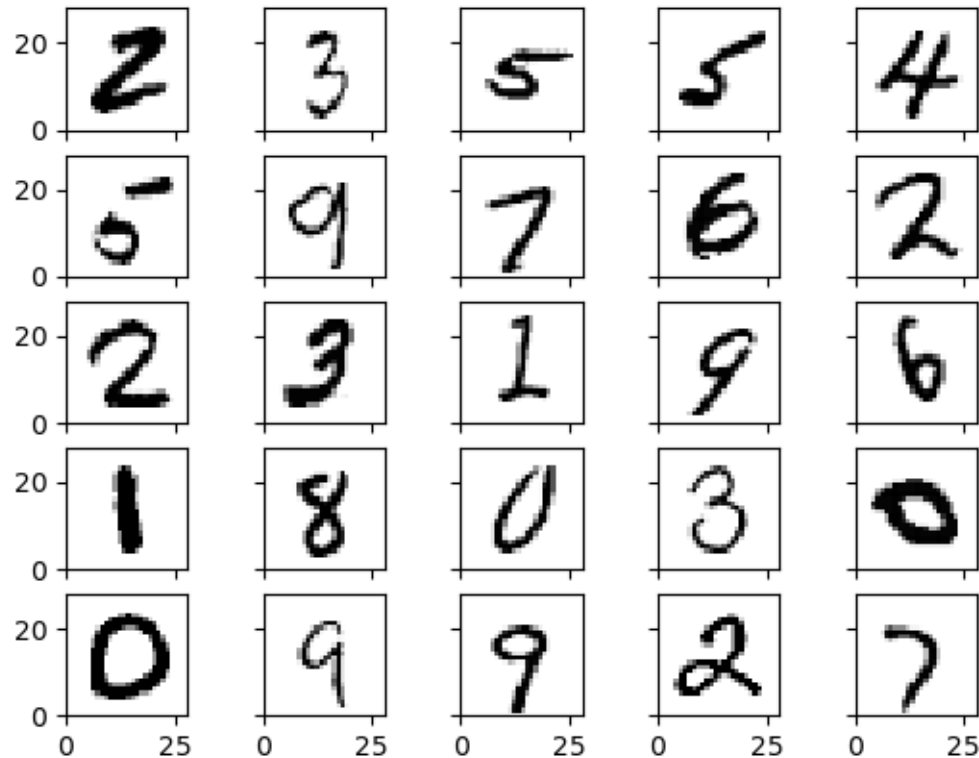
Original Dataset:

28x28 resolution = 784 pixels (dimensions)

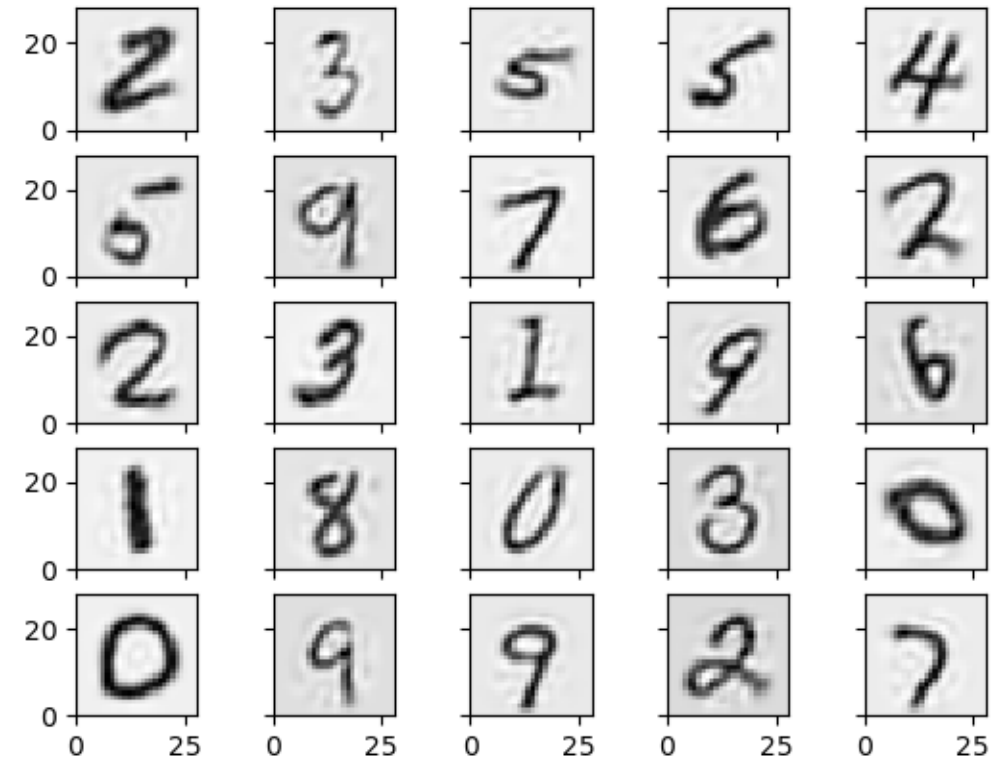
Reconstructed Dataset:

Reduce to 78 dimensions then reconstruct images

Images of Sample MNIST Digits



Images of Sample MNIST Digits



Dimension Reduction

- Principal Component Analysis (PCA):
 - “Linear” approach
 - Cool application of singular value decomposition to project a dataset onto a lower dimensional space
- Autoencoding:
 - “Nonlinear” approach
 - Use techniques from supervised learning to learn lower dimensional representation of dataset

Unsupervised Machine Learning with Python

Section 9.1: Principal Component Analysis Algorithm

Principal Component Analysis (PCA)

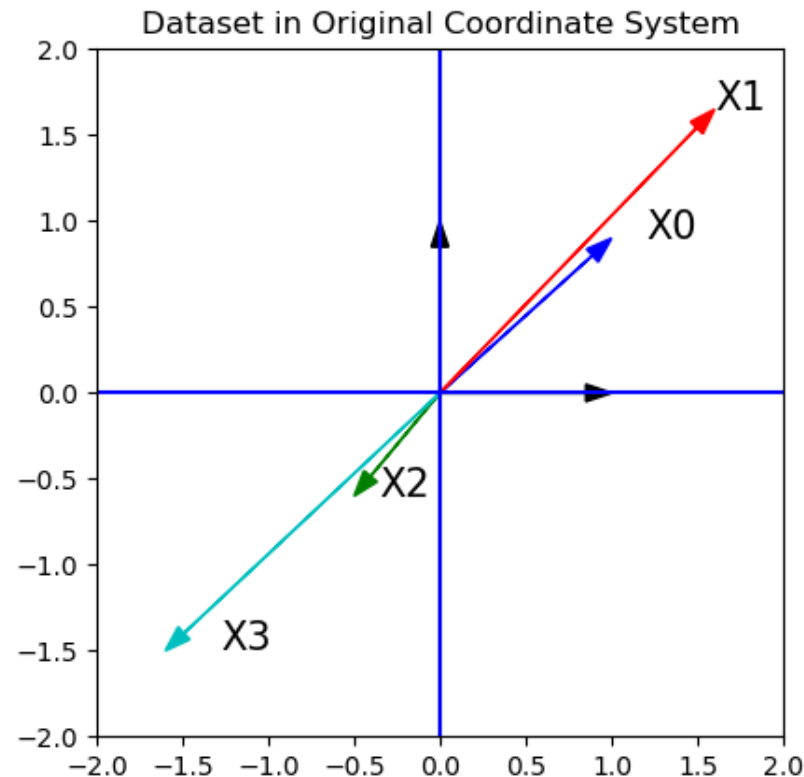
Overview:

- PCA algorithm based on SVD of dataset $X = U\Sigma V^T$
- Columns of u_0, u_1, \dots of U form an orthonormal basis for dataset
- Corresponding singular values $\sigma_0, \sigma_1, \dots$ (which are in decreasing order) indicate the relative importance of each basis vector
- Idea is project dataset onto lower dimensional subspace spanned by u_0, u_1, \dots, u_{K-1} basis vectors (or principal components) that retains sufficient amount of information of dataset

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 compact SVD:

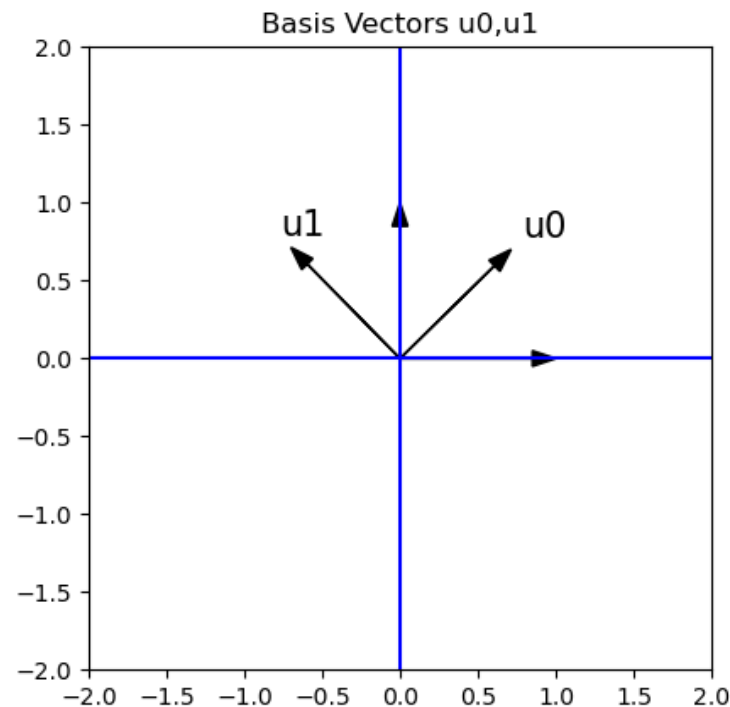
$$X = \begin{bmatrix} 1 & 1.6 & -0.5 & -1.6 \\ 0.9 & 1.65 & -0.6 & -1.5 \end{bmatrix} = U\Sigma V^T = [u_0 \quad u_1] \begin{bmatrix} \sigma_0 & 0 \\ 0 & \sigma_1 \end{bmatrix} \begin{bmatrix} v_0^T \\ v_1^T \end{bmatrix}$$

$$X = \begin{bmatrix} 0.71 & -0.70 \\ 0.70 & 0.71 \end{bmatrix} \begin{bmatrix} 3.54 & 0 \\ 0 & 0.12 \end{bmatrix} \begin{bmatrix} 0.38 & 0.65 & -0.22 & -0.62 \\ -0.47 & 0.46 & -0.63 & 0.41 \end{bmatrix}$$

New Coordinate system with basis u_0 and u_1

- Consider new coordinate system using u_0 and u_1 as basis

$$u_0 = \begin{bmatrix} 0.71 \\ 0.70 \end{bmatrix} \quad u_1 = \begin{bmatrix} -0.70 \\ 0.71 \end{bmatrix}$$



Notes:

- Basis vectors of original system typically correspond to physical or measurable quantity: (word count, pixel intensity, features of a house)
- New basis vectors u_0, u_1, \dots typically don't correspond to physical or measurable quantities

Data points in u0-u1 Coordinate System

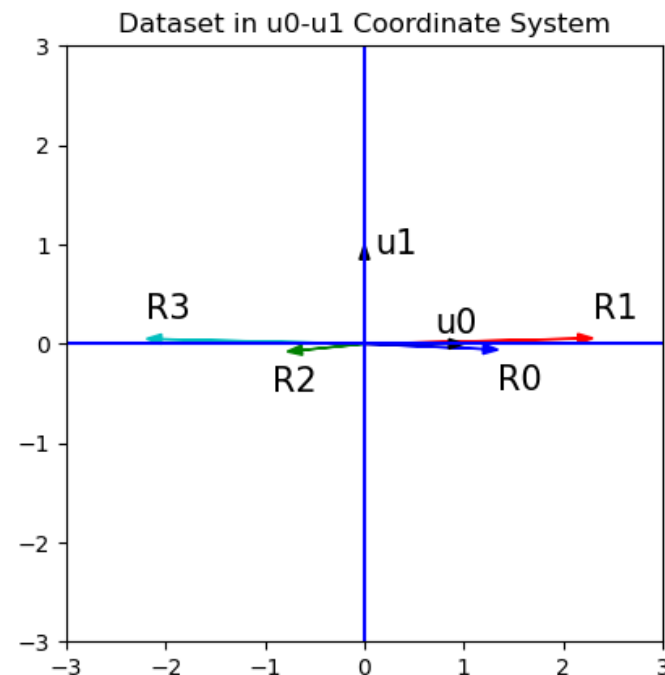
- Data points in u0 and u1 coordinate system

$$\Sigma V^T = \begin{bmatrix} 3.54 & 0 \\ 0 & 0.12 \end{bmatrix} \begin{bmatrix} 0.38 & 0.65 & -0.22 & -0.62 \\ -0.47 & 0.46 & -0.63 & 0.41 \end{bmatrix}$$

$$\Sigma V^T = [R_0 \quad R_1 \quad R_2 \quad R_3] = \begin{bmatrix} 1.34 & 2.30 & -0.78 & -2.19 \\ -0.06 & 0.06 & -0.08 & 0.05 \end{bmatrix}$$

← Units of u0 for each data point

← Units of u1 for each data point



Reduce Dimensions

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

$$R = [\sigma_0][v_0^T] = [3.54][0.38 \quad 0.65 \quad -0.22 \quad -0.62] = [1.34 \quad 2.30 \quad -0.78 \quad -2.19]$$

Units of u0 for each data point



Reconstruct in Original Space

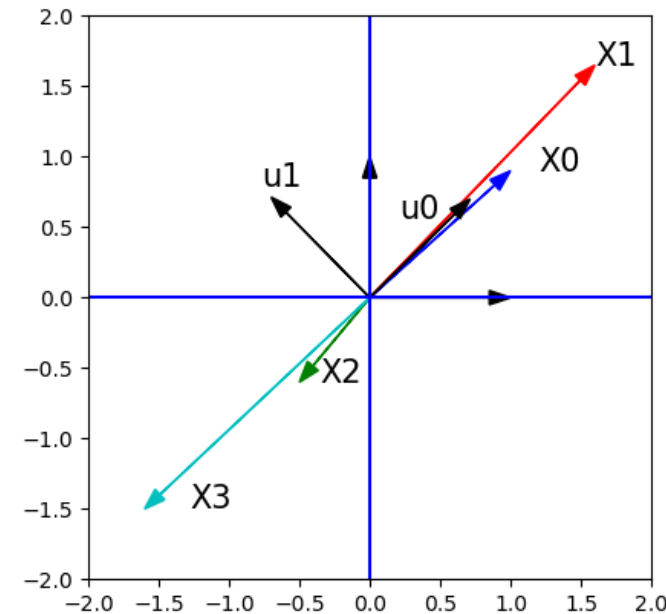
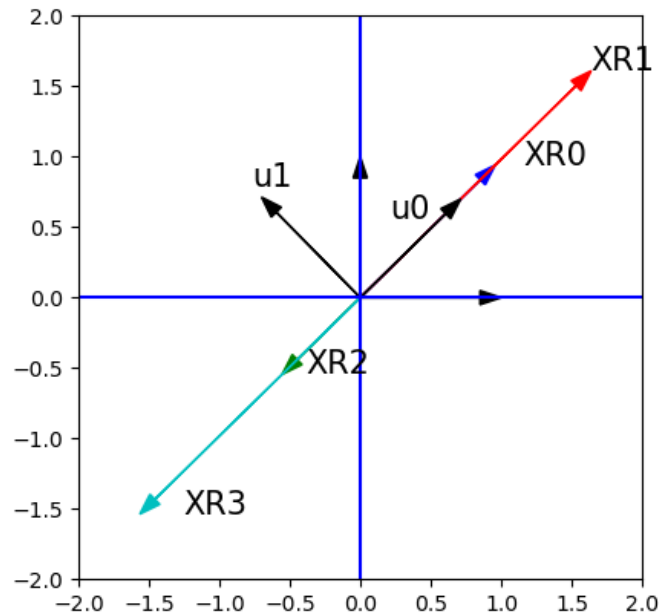
- Can reconstruct data points in original coordinate system using information in u_0 direction only

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

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

$$X_R = \begin{bmatrix} 0.96 & 1.64 & -0.55 & -1.56 \\ 0.94 & 1.61 & -0.54 & -1.54 \end{bmatrix}$$

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



Principal Component Analysis Algorithm

PCA Algorithm:

- Start with data matrix X (d features \times M samples) (here $d < M$)

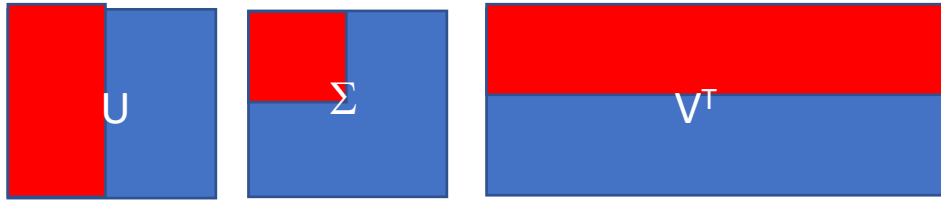
(1) Subtract mean of data $\bar{X} = X - X_{mean}$

(2) Compute the compact version of SVD of $\bar{X} = X - X_{mean} = U\Sigma V^T$

$$\bar{X} (d \times M) = U (d \times d) \Sigma (d \times d) V^T (d \times M)$$

PCA Algorithm

(3) Pick $K < d$ 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}_R in original space(keeping only K principal components)

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

(6) Add back mean to get reconstructed X (based on K principal components)

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

- SVD is connected to L2 distance measure theoretically, so use L2 distance measure when computing distances for consistency

Principal Component Analysis: Choosing K

- Covariance matrix for dataset:

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

- Total variance is sum of eigenvalues of Covariance matrix, which is same as sum of squares of singular values of $(X - X_{mean})$, divided by M-1
- Define

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

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

Example: Choosing K

- Assume $M = 4$ data points
- Assume singular values are: $\sigma_0 = 4, \sigma_1 = 3, \sigma_2 = 2, \sigma_3 = 1$
- Recall: $Variance(K) = \frac{1}{M-1} \sum_{k=0}^{K-1} \sigma_k^2$

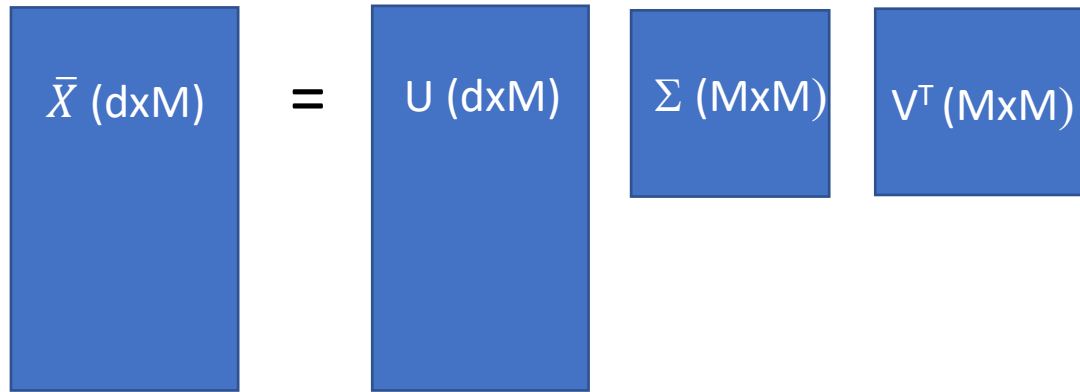
	K = 1	K = 2	K = 3	K = 4
Variance(K)	5.333	8.333	9.667	10.00
Variance(K)/Variance(M)	0.5333	0.8333	0.9667	1.0000

- In this example, only need $K=2$ principal components to capture 83% of total variance

More Dimensions than Data Points

PCA Algorithm in case number of dimensions $d >$ number of samples M

- Compute the compact version of SVD of $\bar{X} = X - X_{mean} = U\Sigma V^T$


$$\bar{X} (d \times M) = U (d \times M) \Sigma (M \times M) V^T (M \times M)$$

- Suppose we retain M principal components

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

- R is in $M < d$ dimensions and no information is lost
- (L2) distances between data points same in original and reduced dimensional space since U has orthogonal columns of length 1.

Example

- Dataset X: 2 data points in 4 dimensions

$$X_0 = \begin{bmatrix} 1 \\ 3 \\ 5 \\ 7 \end{bmatrix} \quad X_1 = \begin{bmatrix} 2 \\ 4 \\ 6 \\ 8 \end{bmatrix} \quad X = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \\ 7 & 8 \end{bmatrix}$$

- Compute L2 (Euclidean) distance between points

$$\text{dist}(X_0, X_1) = 2.646$$

Example

- SVD

$$X = U\Sigma V^T = \begin{bmatrix} -0.16 & -0.49 \\ -0.36 & -0.13 \\ -0.52 & -0.63 \\ -0.76 & 0.59 \end{bmatrix} \begin{bmatrix} 13.93 & 0 \\ 0 & 0.92 \end{bmatrix} \begin{bmatrix} -0.62 & -0.79 \\ 0.79 & -0.62 \end{bmatrix}$$

- R in 2 dimensional space (keep both components):

$$R = \Sigma V^T = \begin{bmatrix} 13.93 & 0 \\ 0 & 0.92 \end{bmatrix} \begin{bmatrix} -0.62 & -0.79 \\ 0.79 & -0.62 \end{bmatrix} = \begin{bmatrix} -8.63 & -10.94 \\ 0.72 & -0.57 \end{bmatrix}$$

$$R_0 = \begin{bmatrix} -8.63 \\ 0.72 \end{bmatrix} \quad R_1 = \begin{bmatrix} -10.94 \\ -0.57 \end{bmatrix} \quad \text{dist}(R_0, R_1) = 2.646$$

Principal Component Analysis DEMO

Jupyter Notebook for demo:

- UnsupervisedML/Examples/Section09/PCA.ipynb

Course Resources at:

- <https://github.com/satishchandrareddy/UnsupervisedML/>

Unsupervised Machine Learning with Python

Section 9.2: Principal Component Analysis Code Design

PCA Code Design

- This section contains design information for the PCA Code based on algorithm presented in Section 9.1
- Create PCA class with methods for computing compact SVD and computing reduced dimension and reconstructed versions of the dataset
- Stop video here, if you would like to do code design yourself

pca class: Principal Variables

Variable	Type	Description
self.Xmean	2d numpy array	X_{mean} of dataset (column vector of dimension $d \times 1$)
self.dimension	Integer	Number of dimensions in dataset
self.nsample	Integer	Number of data points
self.U	2d numpy array	U from SVD of $X - X_{mean}$ ($d \times N$) $N = \min(d, M)$
self.Sigma	1d numpy array	Singular values of $X - X_{mean}$ (N entries)
self.Vt	2d numpy array	V^T from SVD of $X - X_{mean}$ ($N \times M$)
self.cumulative_ variance_ proportion	1d numpy array	Cumulative variance proportion (N entries)

pca class – Key Methods

Method	Input	Description
<code>__init__</code>		Constructor for pca class
<code>fit</code>	X (2d np.array)	Computes compact SVD of X – Xmean and storing in self.U, self.Sigma, and self.Vt Return: nothing
<code>get_dimension</code>	variance_capture (float)	Computes minimum number of principal components to retain at least variance_capture proportion of total variance Return: number of principal components K
<code>data_reduced_dimension</code>	**kwargs reduced_dim (integer) variance_capture (float)	Computes the dataset in U coordinate system with reduced number of dimensions. User specifies either reduced_dim directly or the proportion of variance to be captured. Return: R (2d numpy array)
<code>data_reconstructed</code>	**kwargs reduced_dim (integer) variance_capture (float)	Computes the reconstructed dataset in original coordinate system with reduced number of dimensions. User specifies either reduced_dim directly or the proportion of variance to be captured. Return: X_R (2d numpy array)
<code>plot_cumulative_variance_proportion</code>		Plots the cumulative variance proportion Return: nothing

Unsupervised Machine Learning with Python

Section 9.3: Principal Component Analysis Code Walkthrough

PCA Code Walkthrough

Code located at:

- UnsupervisedML/Code/Programs

Files to Review	Description
pca.py	class for principal component analysis

Course Resources at:

- <https://github.com/satishchandrareddy/UnsupervisedML/>
- Stop video if you would like to implement code yourself first

Unsupervised Machine Learning with Python

Section 9.4: PCA Applied to MNIST Digits Dataset

MNIST Digits Dataset



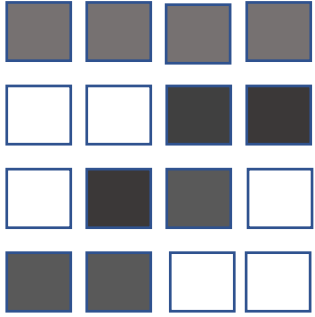
- Thousands of handwritten digit images
- 28x28=784 pixel resolution
- Data Source:
<http://yann.lecun.com/exdb/mnist/>
- Used in machine learning research and teaching
- Links in
UnsupervisedML_Resources.pdf file

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>

Converting Image to Feature Vector

Original Image:
Greyscale 4x4 =16 pixels



Intensity Matrix
4x4 (white=0 to 255=black)

$$\begin{bmatrix} 190 & 190 & 190 & 190 \\ 0 & 0 & 220 & 220 \\ 0 & 220 & 200 & 0 \\ 200 & 200 & 0 & 0 \end{bmatrix}$$



Feature Vector 16x1
Standard to divide by 255

$$\begin{bmatrix} 190 \\ 190 \\ 190 \\ 190 \\ 0 \\ 0 \\ 220 \\ 220 \\ 0 \\ 220 \\ 200 \\ 200 \\ 0 \\ 200 \\ 200 \\ 0 \end{bmatrix}$$

MNIST Digits – Format of Data Files

- Each row contains data for one image
- First column is the digit label (0,1,...,9)
- Columns 2 – 785 are the pixel intensities (integers between 0=white and 255=black)
- 784 dimensions
- We will take transpose so pixel intensities for each image are in a column
- Divide pixel intensities by 255 so feature matrix X values between 0 and 1

Normal

Page Break Preview

Page Layout

Custom Views

☒ Ruler

☒ Formula Bar

☒ Gridlines

☒ Headings

Show

Zoom

100%

Zoom to Selection

New Window

Arrange All

Freeze Panes

Split

Hide

Unhide

Window

View Side by Side

Synchronous Scrolling

Reset Window Position

Switch Windows

Macros

POSSIBLE DATA LOSS

Some features might be lost if you save this workbook in the comma-delimited (.csv) format. To preserve these features, save it in an Excel file format.

Don't show again

Save As...

EJ1

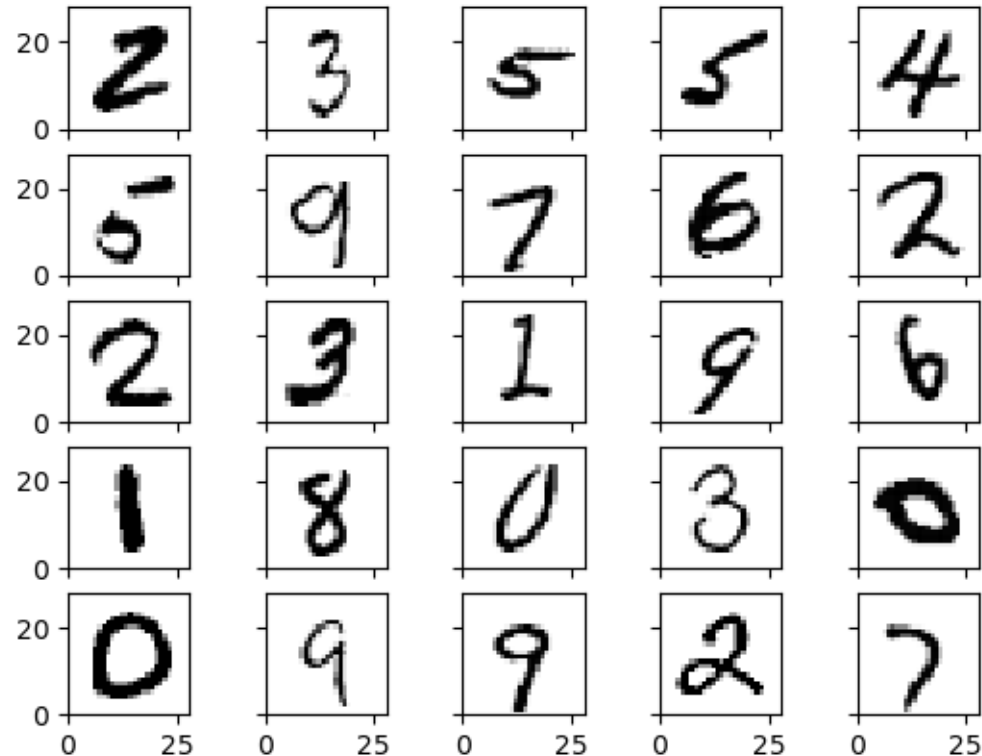
</

Example MNIST Digits

Original Dataset: 784 dimensions

Plot 25 randomly chosen images

Images of Sample MNIST Digits

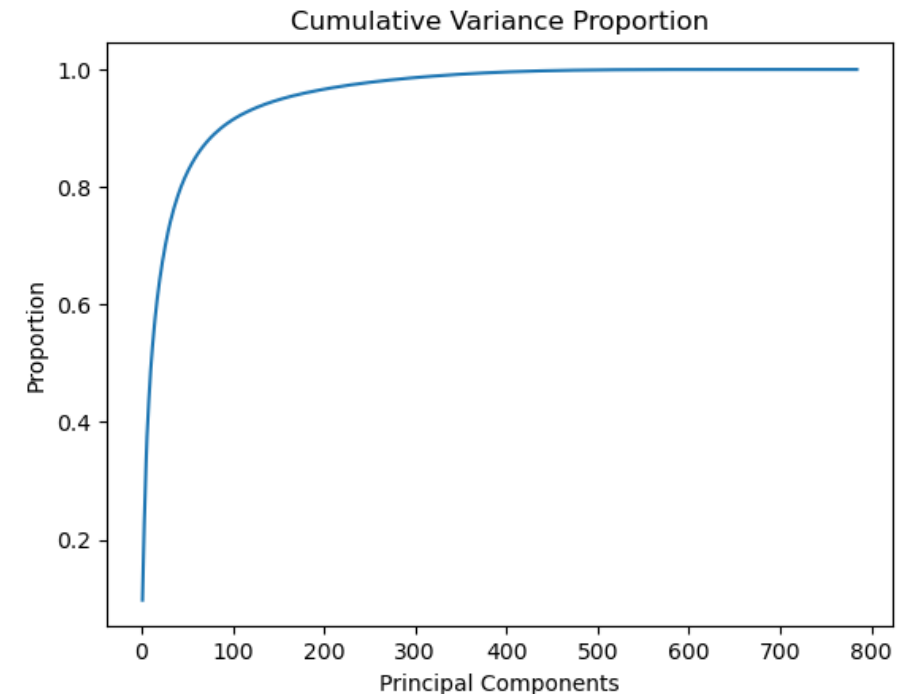


PCA for MNIST Digits Dataset

Perform PCA for MNIST Digits Dataset

- Dataset X: $d = 784$ dimensions and $M = 60000$ images
- Compute compact SVD of $\bar{X} = X - X_{mean} = U\Sigma V^T$

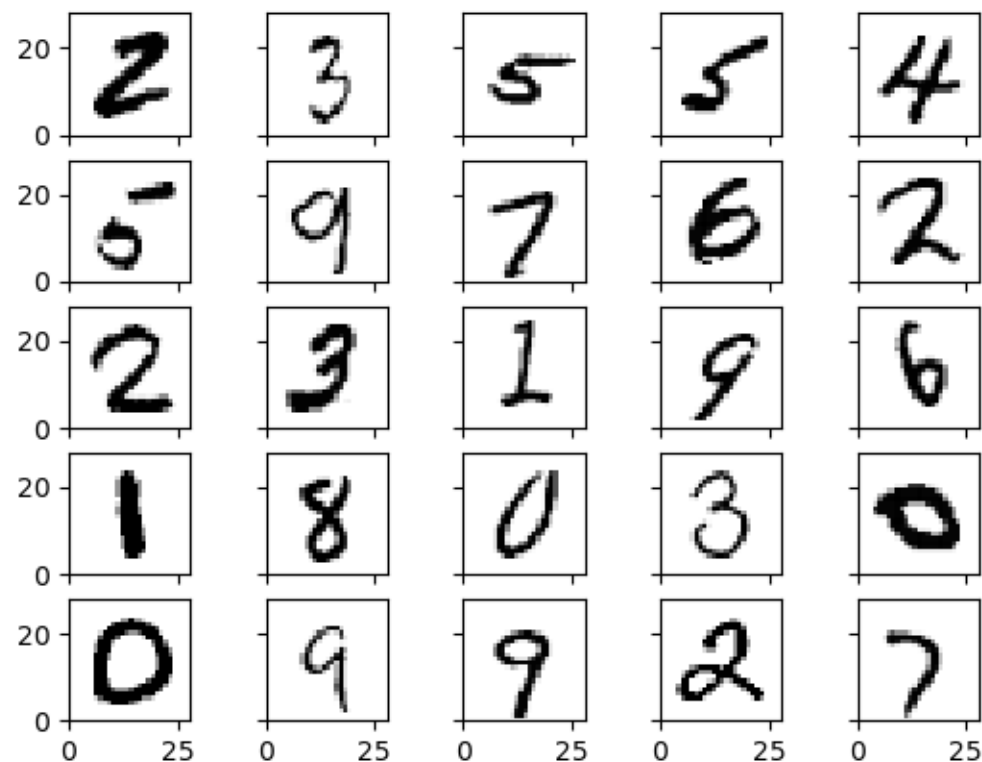
$$\bar{X} (d \times M) = U (d \times d) \Sigma (d \times d) V^T (d \times M)$$



PCA for MNIST: Original and Reconstructed

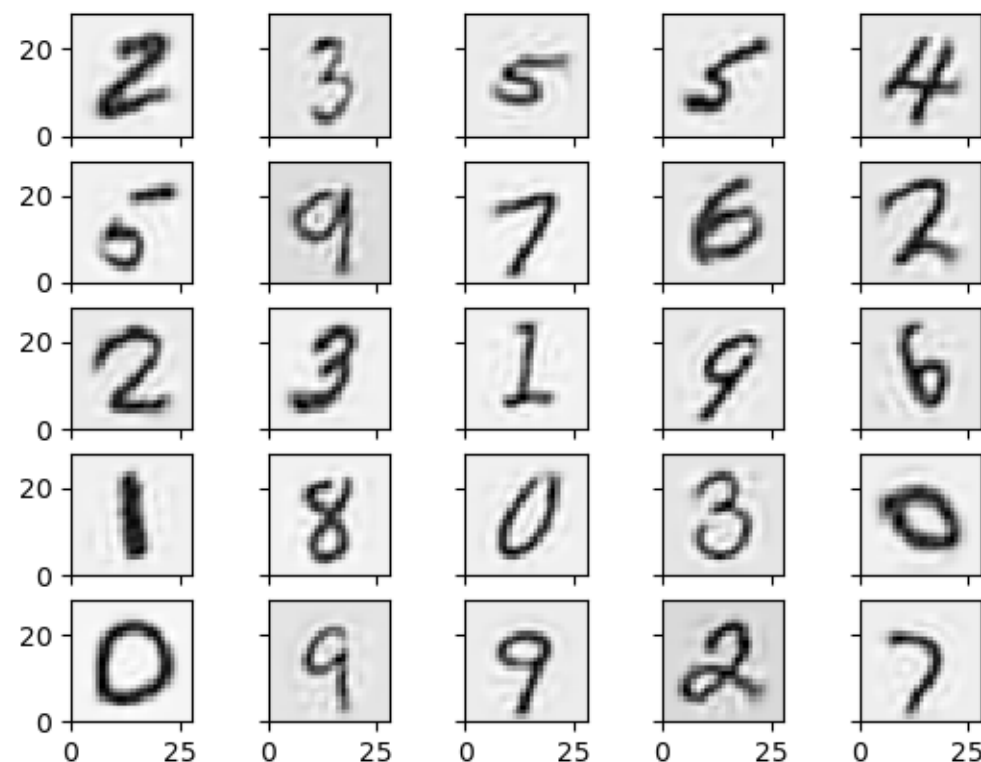
Original Dataset: 784 dimensions
Plot 25 randomly chosen images

Images of Sample MNIST Digits



Reconstructed Dataset:
90% variance capture (87 dimensions)

Images of Sample MNIST Digits



mnist class Code Design

Method	Input	Description
<code>__init__</code>		Constructor for mnist class – saves data directory Return: nothing
<code>load_train</code>	<code>nsample (integer)</code>	Loads <code>nsample</code> (maximum 60,000) images and corresponding class labels from the train dataset Return: <code>X</code> (2d numpy array), <code>class_label</code> (1d numpy array)
<code>load_valid</code>	<code>nsample (integer)</code>	Loads <code>nsample</code> (maximum 10,000) images and corresponding class labels from the validation dataset Return: <code>X</code> (2d numpy array), <code>class_label</code> (1d numpy array)
<code>plot_image</code>	<code>X</code> (2d numpy array) <code>seed (integer)</code>	Creates figure of 25=5x5 images randomly chosen from the dataset <code>X</code> . <code>seed</code> is used to set up the seed for the random number generator. Return: nothing See UnsupervisedML/Examples/Section02/MatplotlibAdvanced.ipynb

PCA for MNIST Digits Code Walkthrough

Programs and data located at

- UnsupervisedML/Code/Programs
- UnsupervisedML/Code/Data_MNIST

Files to Review	Description
Data_MNIST/MNIST_valid_10K.csv	Validation dataset (10000 images)
Programs/data_mnist.py	Class for loading and plotting mnist digits dataset
Programs/driver_pca_mnist.py	Driver for performing pca for mnist dataset

Course Resources at:

- <https://github.com/satishchandrareddy/UnsupervisedML/>
- Stop video if you would like to implement code yourself first

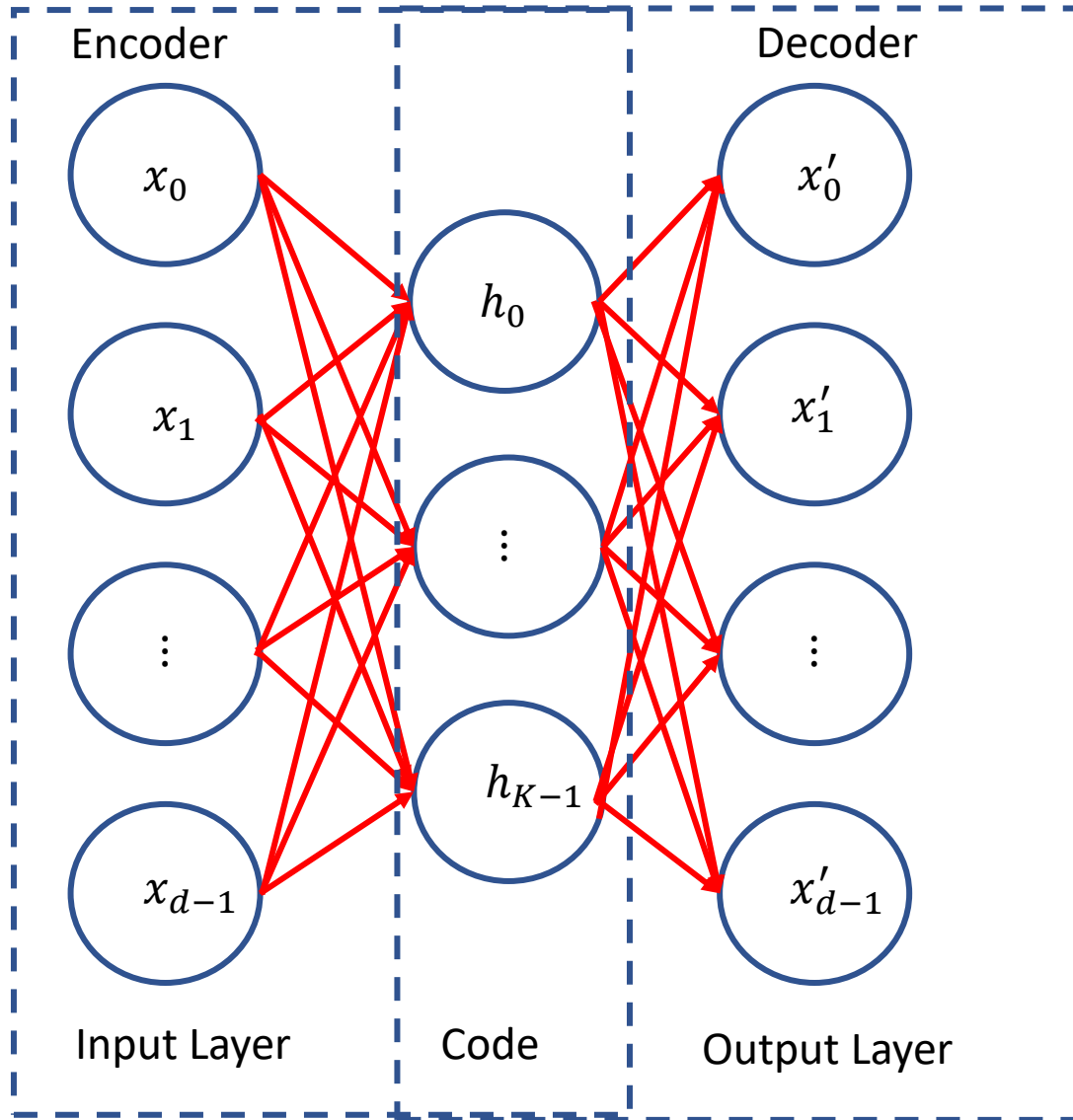
Unsupervised Machine Learning with Python

Section 9.5: Autoencoders

What is an Autoencoder?

- An Autoencoder is type of Artificial Neural Network for learning efficient representations of the feature vector
- Can be use for dimension reduction
- Whereas PCA is linear, Autoencoders can use a nonlinear approach
- Beyond the scope of this course to go through all the details, but I want to give an introduction
- Links to additional information in UnsupervisedML_Resources.pdf file

Sample Autoencoder Neural Network



- Input: (d dimensions)

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

- Encoder (1 layer example): 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 (1 layer example): 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:

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

X (d dimensions) is feature vector, W is Kxd matrix, b is Kx1 vector, f is activation function, H is (K dimensions) reduced dimension version of feature vector

- Decoder:

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

X' is (d dimensions) reconstructed feature vector, W' is dxK matrix, b' is dx1 vector, g is activation function

- Given a dataset of M vectors X_0, X_1, \dots, X_{M-1} and pre-specified activation functions f, 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 1

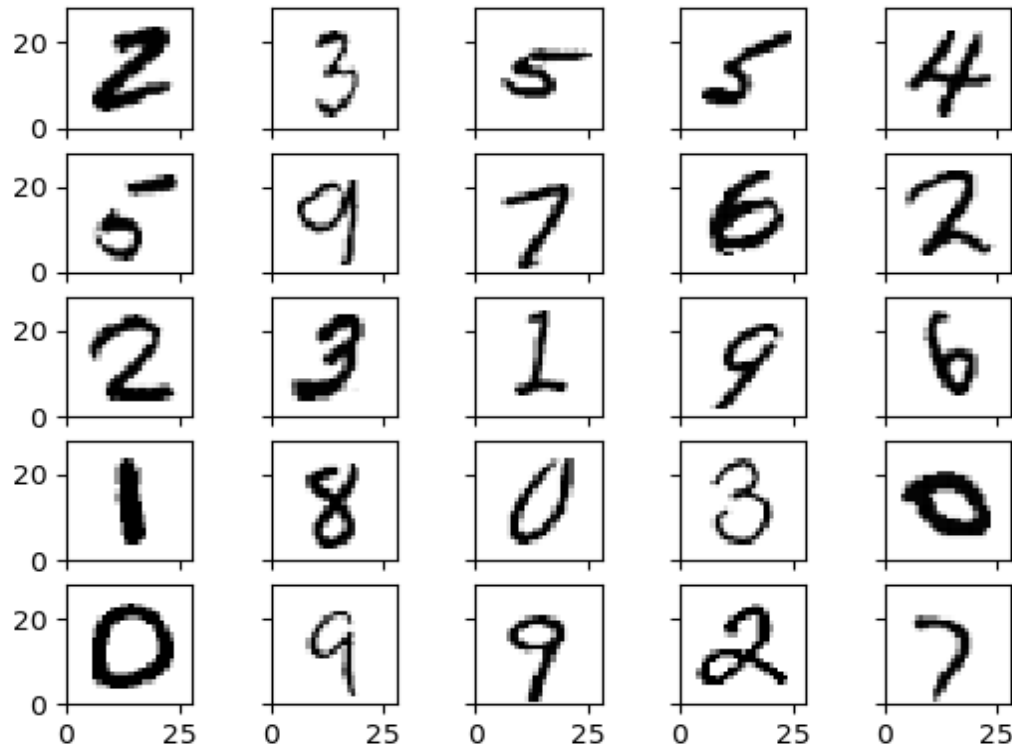
- Autoencoder set up is similar to Supervised Learning
 - Key difference is that there is no label Y for each data point
- Determine W, b, W', b' using optimizer such as Gradient Descent and backpropagation to determine derivatives
- Can use familiar activation functions such as sigmoid, Relu for f and g
- “Code” representation H is related nonlinearly to X (if nonlinear activation functions used)
- Code representation H is equivalent to reduced dimension version R from Principal Component Analysis

Example 1: Autoencoder for MNIST Digits

- Use linear activation functions $f(z) = z$ and $g(z) = z$ – results similar to PCA

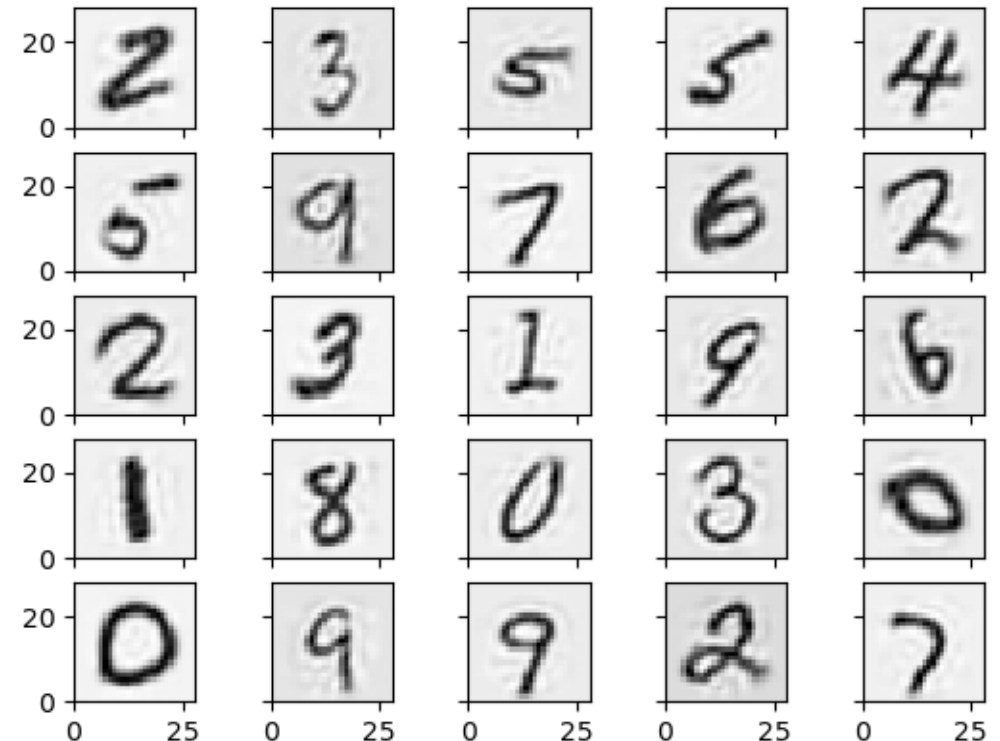
Original Dataset: 784 dimensions
Plot 25 randomly chosen images

Images of Sample MNIST Digits



Reconstructed Dataset: (87 dimensions)
Linear activation functions

Images of Sample MNIST Digits

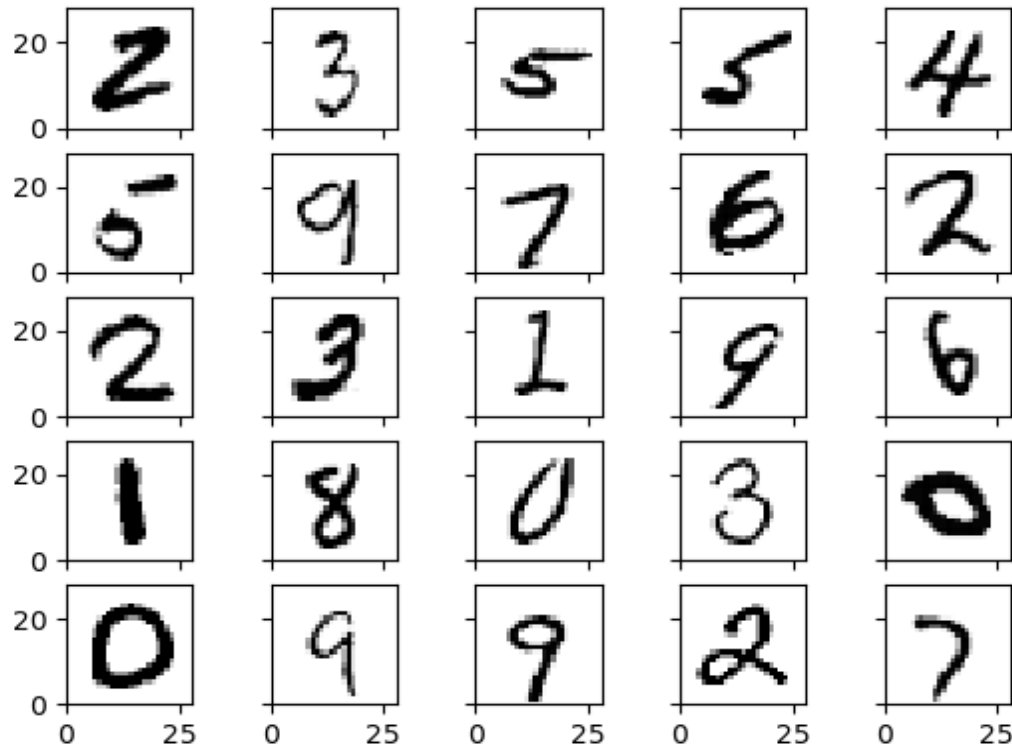


Example 2: Autoencoder for MNIST Digits

- Use nonlinear activation functions $f(z) = \text{Relu}(z)$ and $g(z) = \text{sigmoid}(z)$

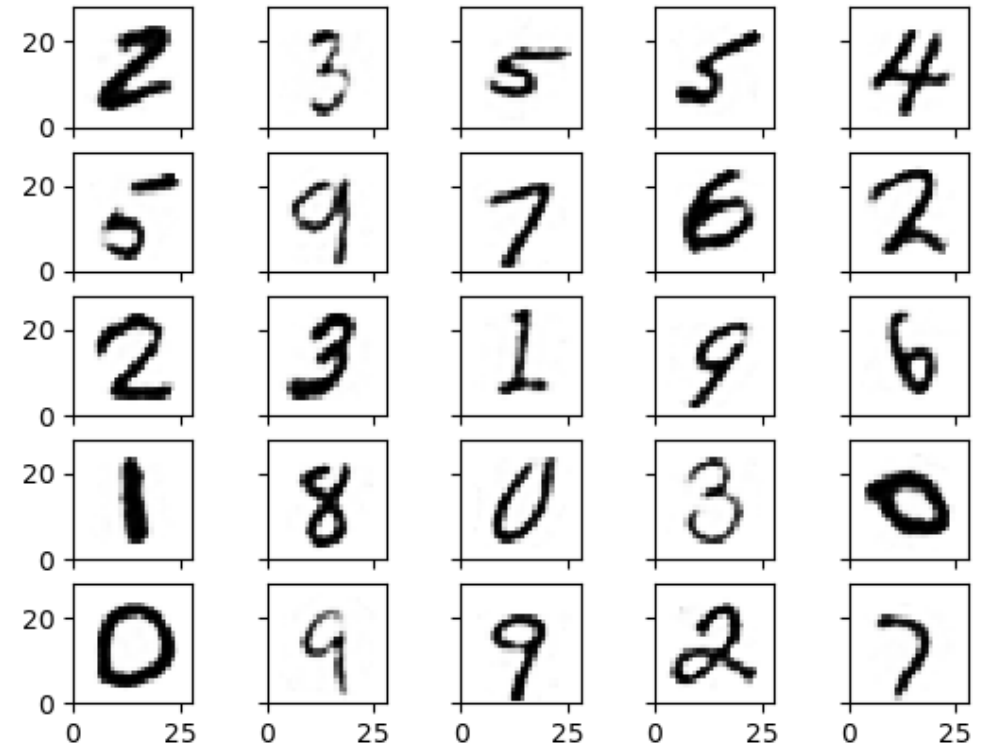
Original Dataset: 784 dimensions
Plot 25 randomly chosen images

Images of Sample MNIST Digits



Reconstructed Dataset: (87 dimensions)
Nonlinear activation functions

Images of Sample MNIST Digits



Autoencoder: Notes 2

- Autoencoder has more flexibility than PCA approach based on SVD
 - Can have multiple neural network layers in encoder and decoder
 - Can use other loss functions, such binary cross entropy
- PCA has advantage of generating cumulative variance proportion data as function of dimension which can be used to determine a suitable reduced dimension
 - For Autoencoders, must redo training for each chosen reduced dimension

Unsupervised Machine Learning with Python

Section 9.6: Autoencoder Demo (Optional)

Autoencoder Demo (Optional)

- This is an optional lecture with a demo showing an implementation of Autoencoders for dimension reduction for the MNIST digits dataset
- Demo deals with topics such as supervised learning and using the tensorflow framework. These topics are beyond the scope of this course, so I won't go into them in detail.
- I think this is a useful demo and it may be of particular interest to students who have studied supervised machine learning and have used tensorflow previously
- The concepts in this demo are not needed for the subsequent sections of this course
- Additional references listed in the UnsupervisedML_Resources.pdf file

Demo Set Up

- Need to install tensorflow package to run autoencoder demo code
- Suggested Approach: Create new conda environment to install packages
 - Within the Anaconda Prompt window
 - In base environment create tf_2.5.0 environment: **conda create -n tf_2.5.0 python=3.8.3**
 - Activate new environment: **conda activate tf_2.5.0**
 - Install tensorflow in new environment: **pip install tensorflow==2.5.0**
 - Install matplotlib in new environment: **conda install matplotlib=3.2.2**
 - Install pandas in new environment: **conda install pandas=1.0.5**
 - Deactivate new environment and return to base: **conda deactivate**
- Alternatively: Install tensorflow in base conda environment
 - Issue with this approach: tensorflow also installs other packages, which may conflict with existing versions of these packages in base environment
- Can use any installation approach as long as you have tensorflow, numpy, matplotlib, and pandas

Autoencoder Demo

Programs and data located at

- UnsupervisedML/Code/Programs
- UnsupervisedML/Code/Data_MNIST

Files to Review	Description
Programs/driver_autoencoder_mnist_tf.py	Driver for autoencoder approach for dimension reduction for MNIST dataset

Course Resources at:

- <https://github.com/satishchandrareddy/UnsupervisedML/>