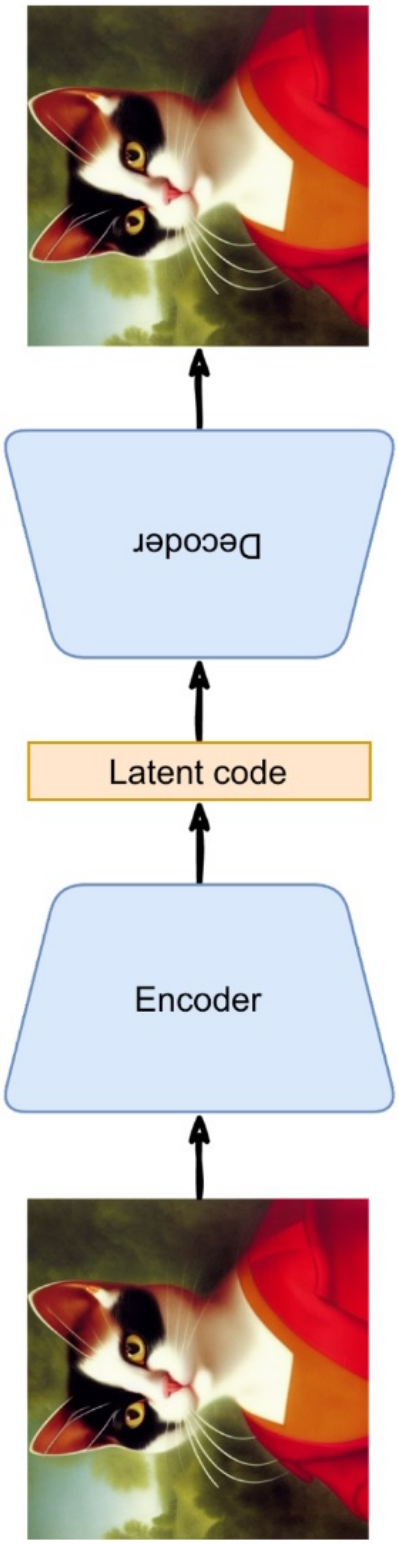
Autoencoders and Variational Autoencoder

Asir Intesar Tushar, Alexander Brooks, and Tahmina Farhin

Math 526, Department of Mathematics.

April 16th, 2024


Decoder
Latent code
Encoder

Overview:

- Introduction to Autoencoders
- Brief Introduction to PCA
- PCA vs Linear AE
- PCA vs Nonlinear AE
- Denoising AE
- Outlier/ Anomaly Detection

Unsupervised learning

Unsupervised learning is a type of machine learning where the algorithm learns patterns and structures from unlabeled data without explicit guidance or supervision [1].
Primary tasks of unsupervised learning:

1. Clustering.
2. Dimensionality Reduction.

[1] https://www.ibm.com/topics/unsupervised-learning

Encoder

In the context of neural networks, an encoder is a component responsible for transforming input data into a different representation, often of lower dimensionality, which captures important features of the input.
The encoder function $f(x)$ learns to extract meaningful features from the input data, capturing the most salient information while discarding irrelevant details.

AutoEncoder

Autoencoding is training a neural network to replicate its input to its output. The network (a single hidden layer) may be viewed as consisting of two parts: an encoder function

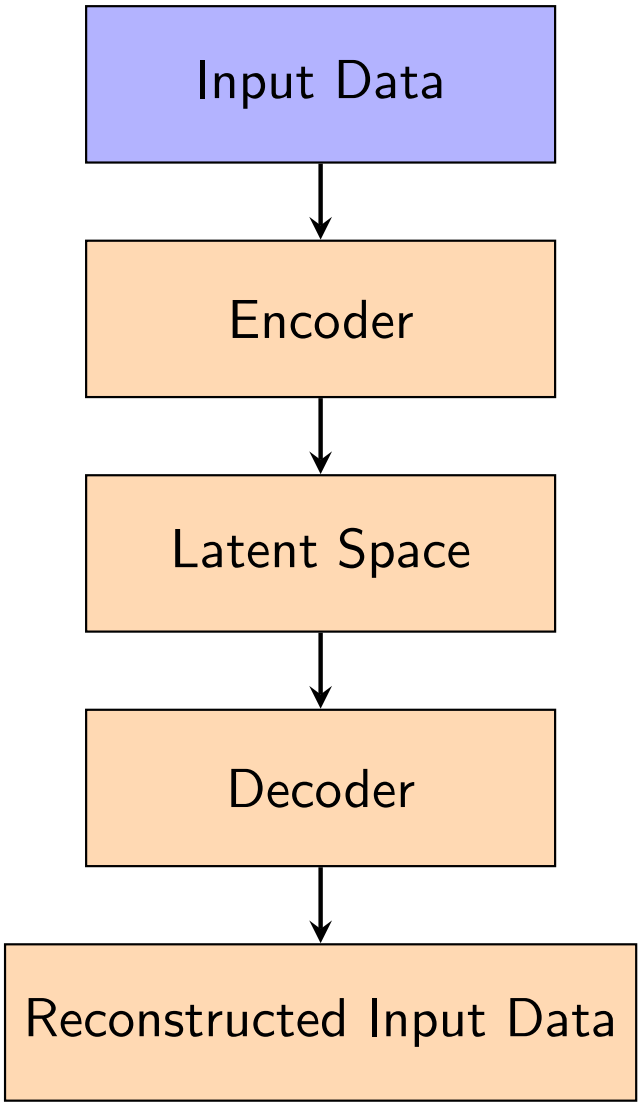
$$Z = f(X) = f_1(W_1 \cdot X + b_1)$$

and a decoder that produces a reconstruction

$$X' = h(Z) = f_2(W_2 \cdot Z + b_2)$$

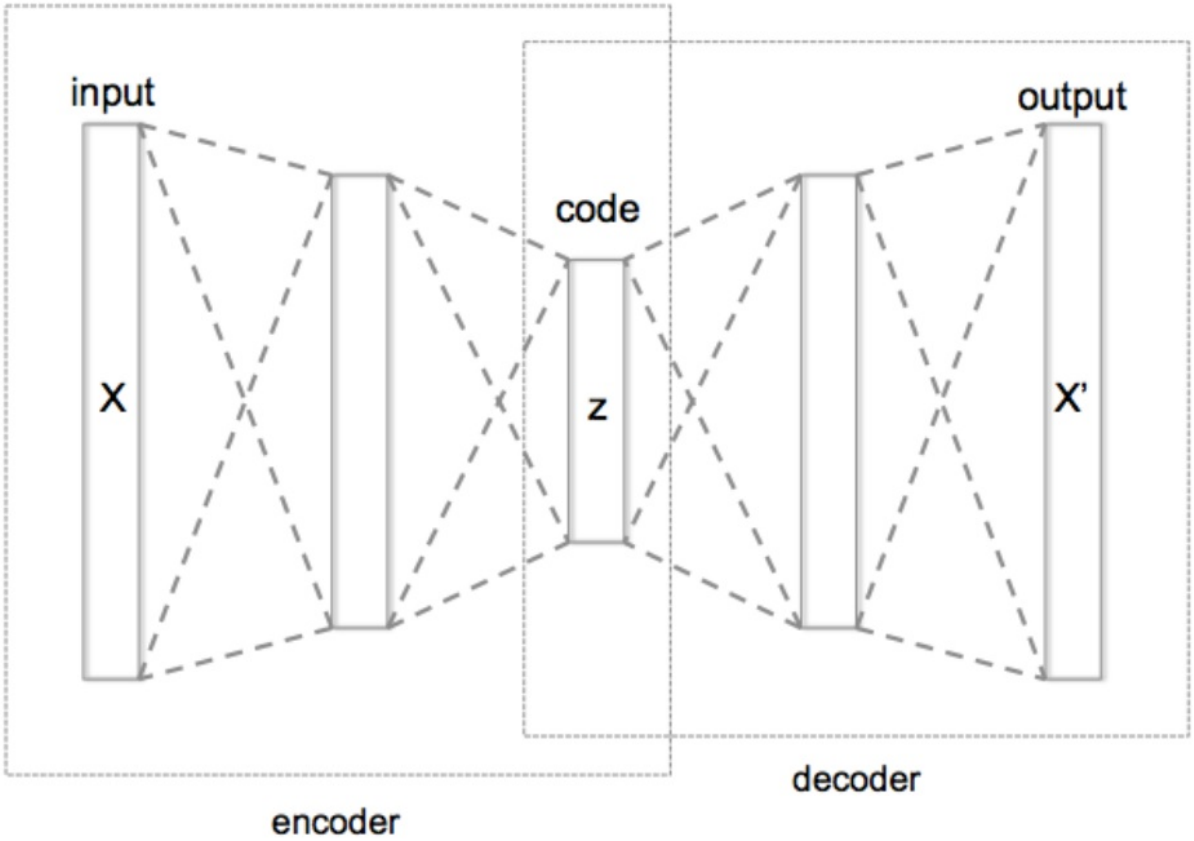
Where W_1, W_2 are weight matrices and b_1, b_2 are bias vectors. f_1 and f_2 are activation functions applied to the hidden layer and output layer respectively. During training, the autoencoder adjusts these weights and biases iteratively through optimization techniques like gradient descent to minimize the reconstruction error.

Flowchart of AutoEncoder Components



1. **Input Data:** The autoencoder takes raw input data, which could be images,text, or any other form of structured data.
2. **Encoder:** Takes input data and maps it to a lower-dimensional latent space.
3. **Latent Space:**Encoded representation of the input data, capturing essential features.
4. **Decoder:** Reconstructs the input data from the latent space representation.
5. **Reconstructed Input Data:** The output of the decoder is a reconstructed version of the input data which could be images,text, or any other form of structured data.

Autoencoder Architecture



Autoencoder seeks to minimize a loss function.

Activation Functions and Loss Functions

Commonly used activation functions:

$$\text{Linear Identity Function}: f(x) = x$$
$$\text{Relu: Rectified Linear Unit:} f(x) = max(0, x)$$
$$\text{Sigmoid:} f(x) = \frac{1}{1 + e^{-x}}$$
$$\text{Hyperbolic Tangent:} f(x) = \tanh x = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

Typical Loss Functions (Reconstruction Error):

$$\text{MSE} = \frac{1}{M} \sum_{i}^{M} |x_i - x_i'|^2$$

$$\text{Binary Cross Entropy loss} = -\frac{1}{M} \sum_{i}^{M} \left(x_i \log\left(x_i'\right) + (1 - x_i) \log(1 - x_i')\right)$$

Example: MNIST (Handwritten digits)

Let us now see how an autoencoder performs with a real example, using the MNIST dataset. This dataset contains hand-written digits from 0 to 9. Each image is 28×28 pixels with only gray values, that means that we have 784 features (the pixel gray values) as inputs. For this example, we used the cross-entropy as loss function and we trained the model for 10 epochs.
The MNIST database contains 60,000 training images and 10,000 testing images.

MNIST

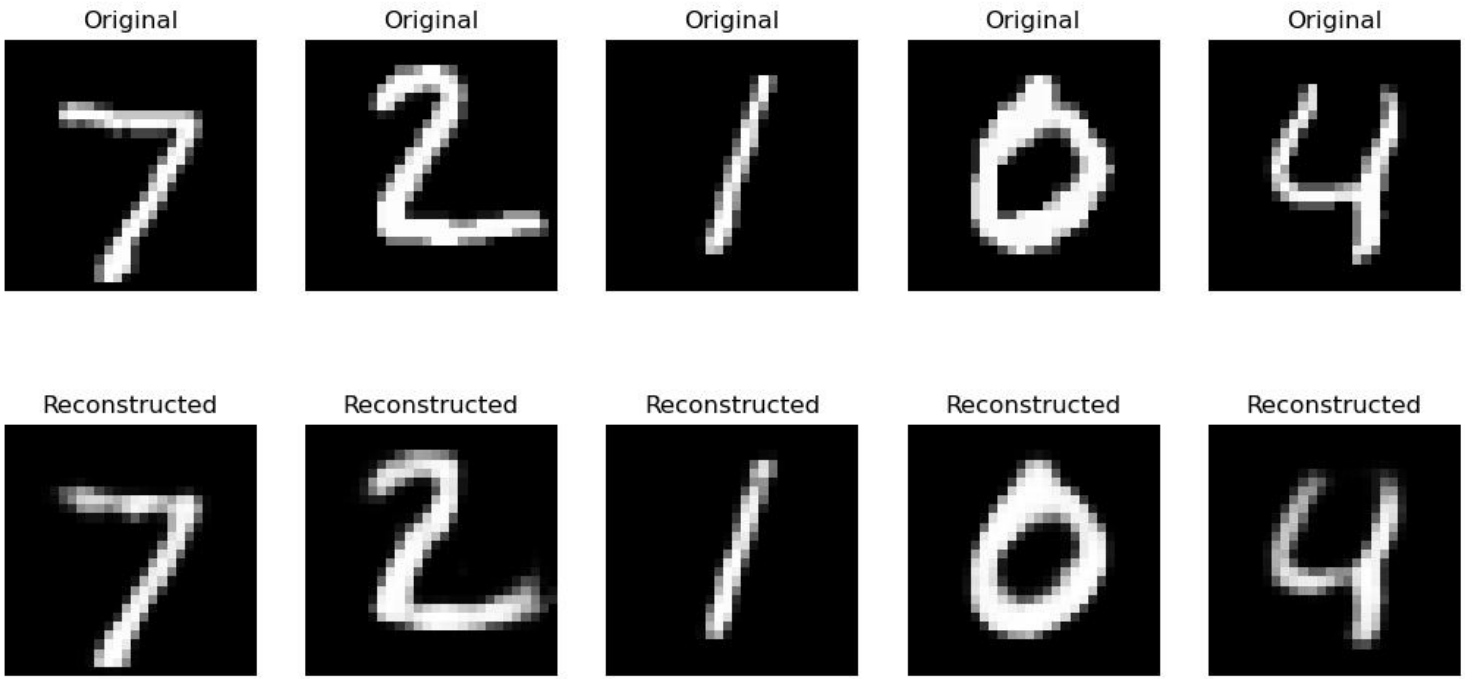


Figure: 784 to 64 to 784 dimensions. And The MNIST database is a large database of handwritten digits that is commonly used for training various image processing systems

Feature Reduction

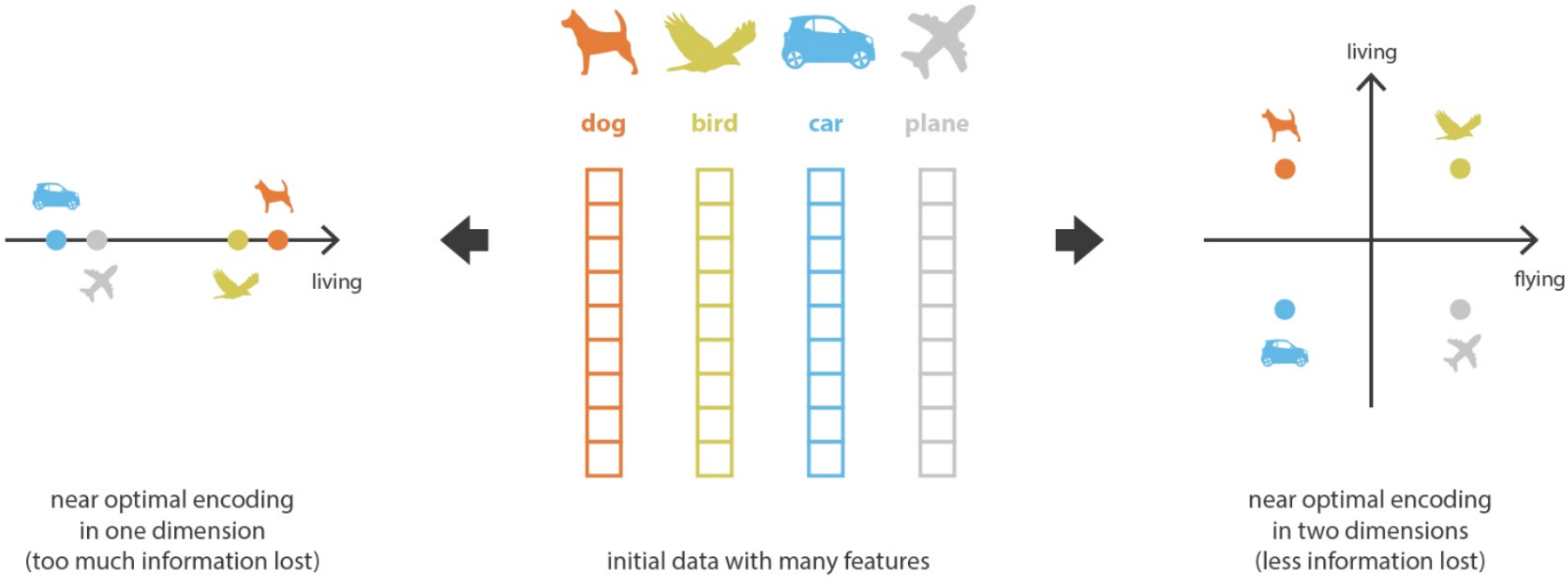


Figure: A dummy example of Feature/Dimension Reduction.
Techniques such as regularization, careful architecture design can help strike a balance and determine an optimal level of dimensionality reduction for a given task and dataset.

MNIST Example Again

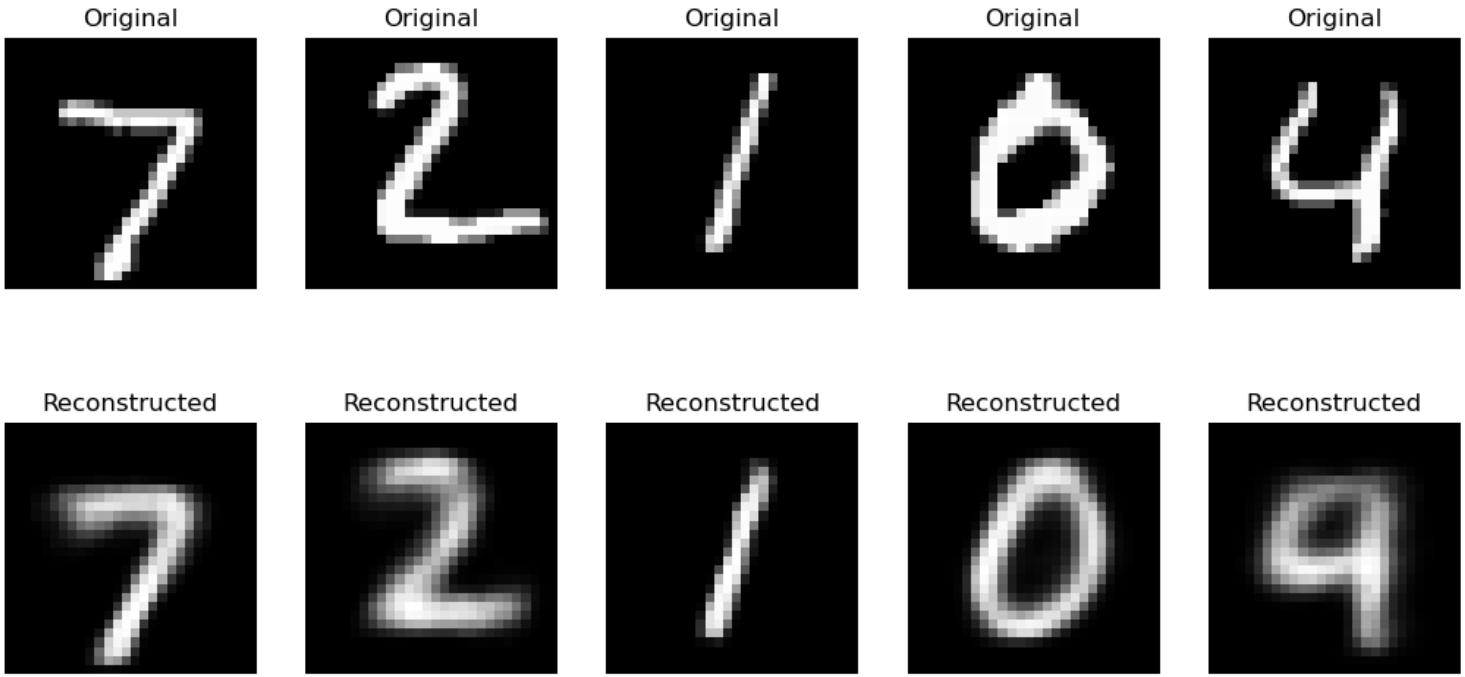
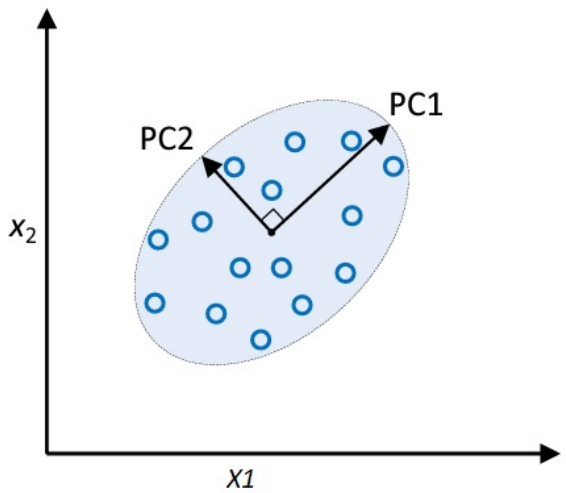


Figure: 784 to 8 to 784 dimensions. It has bigger REs; one totally incorrect reconstructed image.

PCA-Introduction

1) Find directions of maximum variance



The blue dots are our data sets, and we have two features x_1 and x_2. And PC_1 and PC_2 represents the Principal Components. We can think of the data set as two-dimensional Matrix, then this PC_1 and PC_2 are eigenvectors associated with the largest eigenvalue of the Matrix and the second largest eigenvalue of the matrix.

PCA-intro Cont.

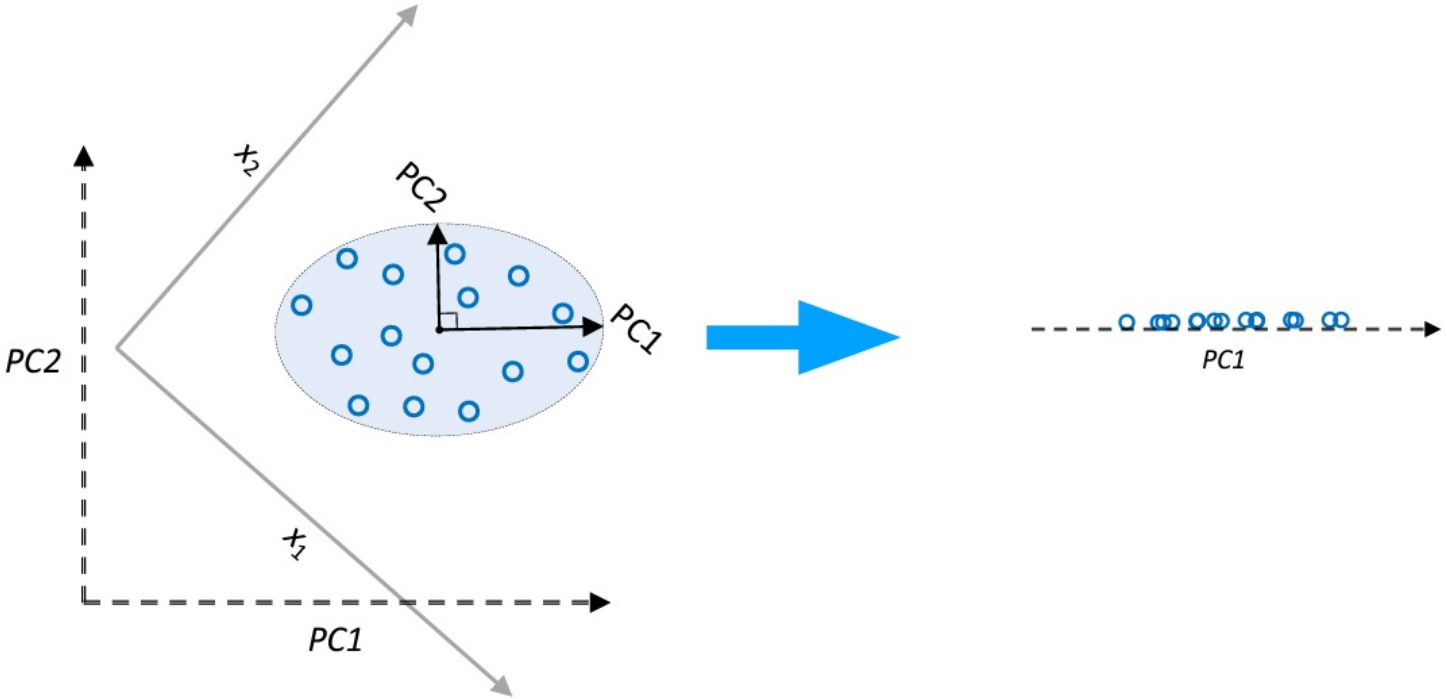


Figure: Transforming(Linear Transform) onto directions of max. variance (New Feature Axis). For Dimensionality Reduction: $2D \rightarrow 1D$. Projecting the data onto One-D (PC1).

PCA introduction

Principal component analysis is a technique that extracts the best orthogonal subspace in which we can project our data points by maximizing the variance. This method uses a linear transformation to create a new data representation, yielding a set of "principal components."
What can we say about features in the subspace of PCA ?

PCA introduction

Principal component analysis is a technique that extracts the best orthogonal subspace in which we can project our data points by maximizing the variance. This method uses a linear transformation to create a new data representation, yielding a set of "principal components."
What can we say about features in the subspace of PCA ?

- An orthogonal set of vectors (features) is linearly independent.

PCA introduction

Principal component analysis is a technique that extracts the best orthogonal subspace in which we can project our data points by maximizing the variance. This method uses a linear transformation to create a new data representation, yielding a set of "principal components."
What can we say about features in the subspace of PCA ?

- An orthogonal set of vectors (features) is linearly independent.
- *proof:* if $v_1, v_2, \cdots v_j$ are orthogonal set of vectors. Let, $c_1 v_1 + c_2 v_2 + \cdots + c_k v_k = 0$. Then, multiplying v_j on both sides: $c_j \|v_j\|^2 = 0$ implies $c_j = 0$. Therefore, they are linearly independent.

PCA-Linear AutoEncoder

A linear autoencoder is a neural network composed by an encoder (single layer) that compresses our space in a new subspace through linear transformation, which is not necessarily orthogonal[2] , and of a decoder that reconstruct our data with less information loss possible.

[2] Elad Plaut.'*From Principal Subspaces to Principal Components with Linear Autoencoders*'

Linear Autoencoder

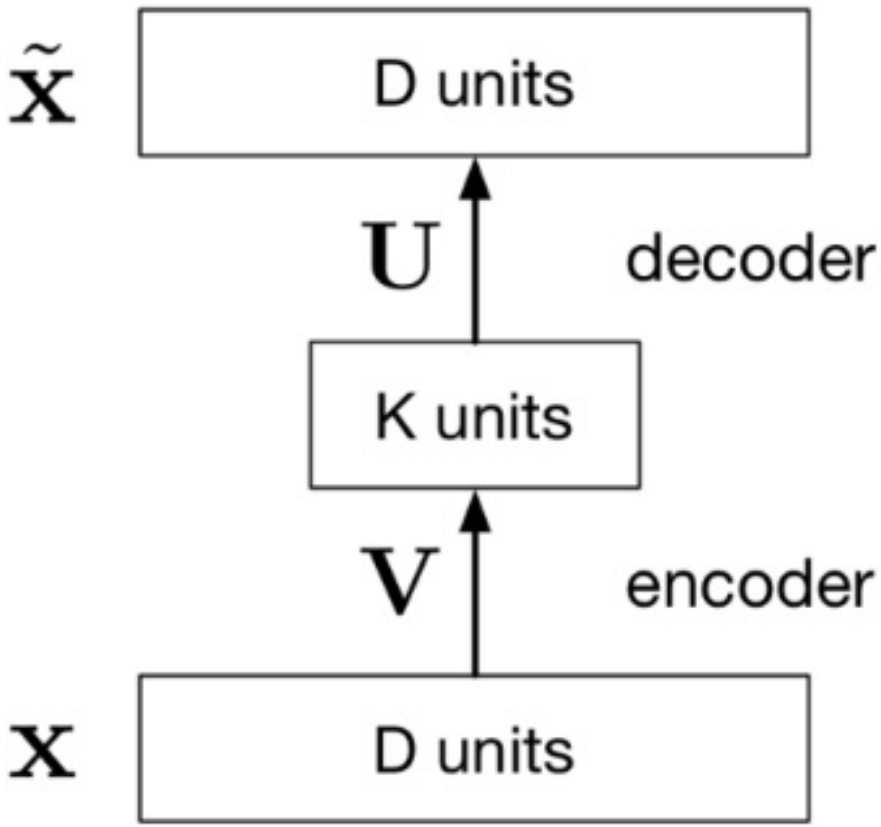
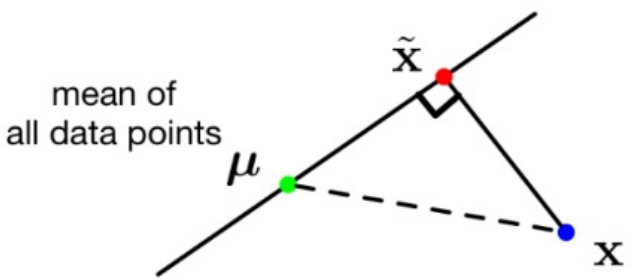


Figure: The simplest kind of autoencoder has one hidden layer, linear activations, and squared error loss. The network computes $\tilde{x} = UVx$.

Linear Autoencoder

- The autoencoder should learn to choose the subspace which minimizes the squared distance from the data to the projections.
- This is equivalent to the subspace which maximizes the variance of the projections.
- The best possible K dimensional subspace in terms of reconstruction error is the PCA subspace[3]. Therefore, PCA is equivalent to Linear Autoencoder.



By the Pythagorean Theorem,

$$\underbrace{\frac{1}{N}\sum_{i=1}^{N}\|\tilde{\mathbf{x}}^{(i)}-\boldsymbol{\mu}\|^2}_{\text{projected variance}}+\underbrace{\frac{1}{N}\sum_{i=1}^{N}\|\mathbf{x}^{(i)}-\tilde{\mathbf{x}}^{(i)}\|^2}_{\text{reconstruction error}}$$

$$=\underbrace{\frac{1}{N}\sum_{i=1}^{N}\|\mathbf{x}^{(i)}-\boldsymbol{\mu}\|^2}_{\text{constant}}$$

[3]Elad Plaut,'*From Principal Subspaces to Principal Components with Linear Autoencoders*'

PCA vs Lin AE

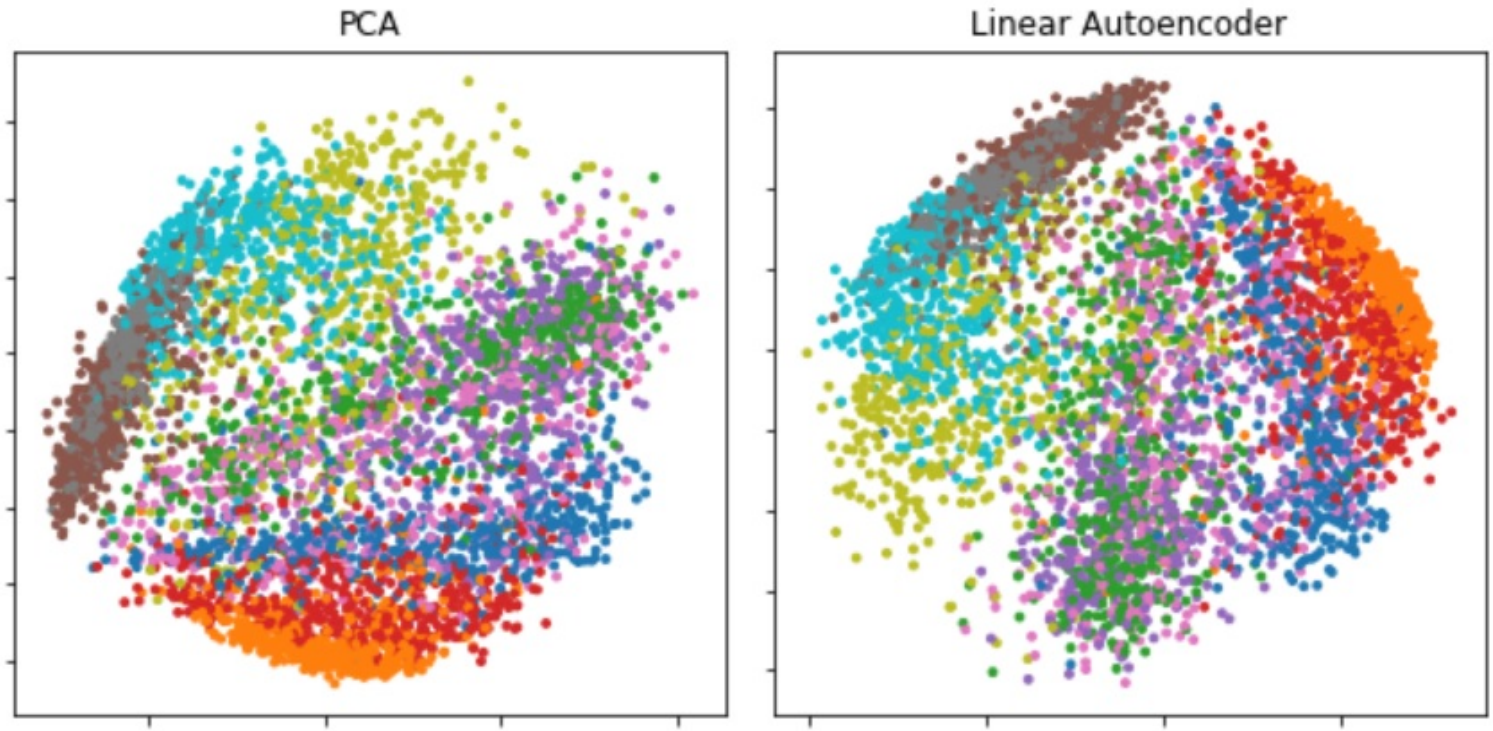


Figure: MNIST data:Plot of the first two Principal Components (left) and a two-dimension hidden layer of a Linear Autoencoder (Right) applied to the MNIST dataset. The two models being both linear learn to span the same subspace. The projection of the data points is indeed identical, apart from rotation of the subspace.

PCA vs Non-lin. AE

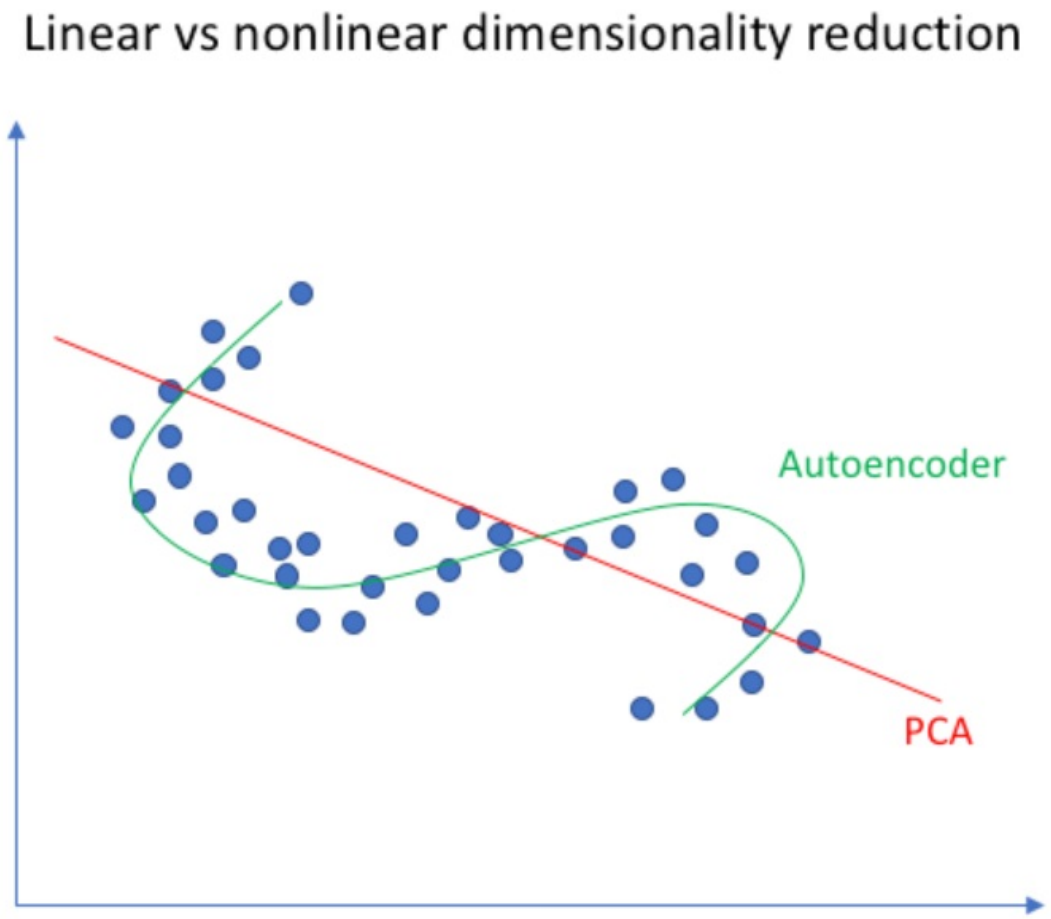


Figure: PCA vs nonlinear Autoencoder

PCA-NonLinear AutoEncoder

Figure: The figure illustrates how the nonlinearity of autoencoders can help to isolate the signals in the features better than PCA using MNIST data [4]. 2 epochs

Denoising-AutoEncoder

We modify the basic autoencoder. We will train it to construct a clean "repaired" input from a corrupted , partially destroyed one.[4]

1. We need to first corrupt the initial input x to get a partially destroyed version $\widetilde{x}$ by means of a stochastic mapping $\widetilde{x} \sim q_D(\widetilde{x}|x)$
2. We considered the following corrupting process, parameterized by the desired proportion ν of destruction: for each input x, a fixed number of ν components are chosen at random and gets destroyed, while others are left untouched.
3. All information about the chosen components is thus removed from the particular input pattern, and the autoencoder will be trained to "fill-in" these artificially introduced "blanks".

[4] Pascel Vincent,Yoshua Bengio, Hugo Larochelle,*'Extracting and Composing Robust Features with Denoising Autoencoder'*

DAE Cont.

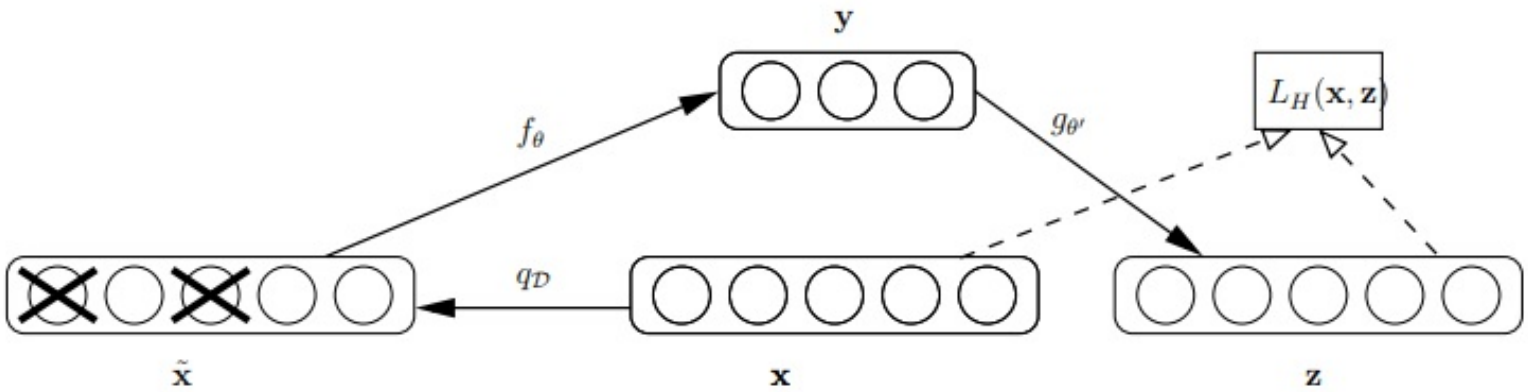


Figure 1. An example $\mathbf{x}$ is corrupted to $\tilde{\mathbf{x}}$. The autoencoder then maps it to $\mathbf{y}$ and attempts to reconstruct $\mathbf{x}$.

1. $\widetilde{x} \sim q_D(\widetilde{x}|x)$
2. $y = f_\theta(\widetilde{x})$
3. $z = g_{\theta'}(y)$
4. Error function: $L(x, z)$

DAE

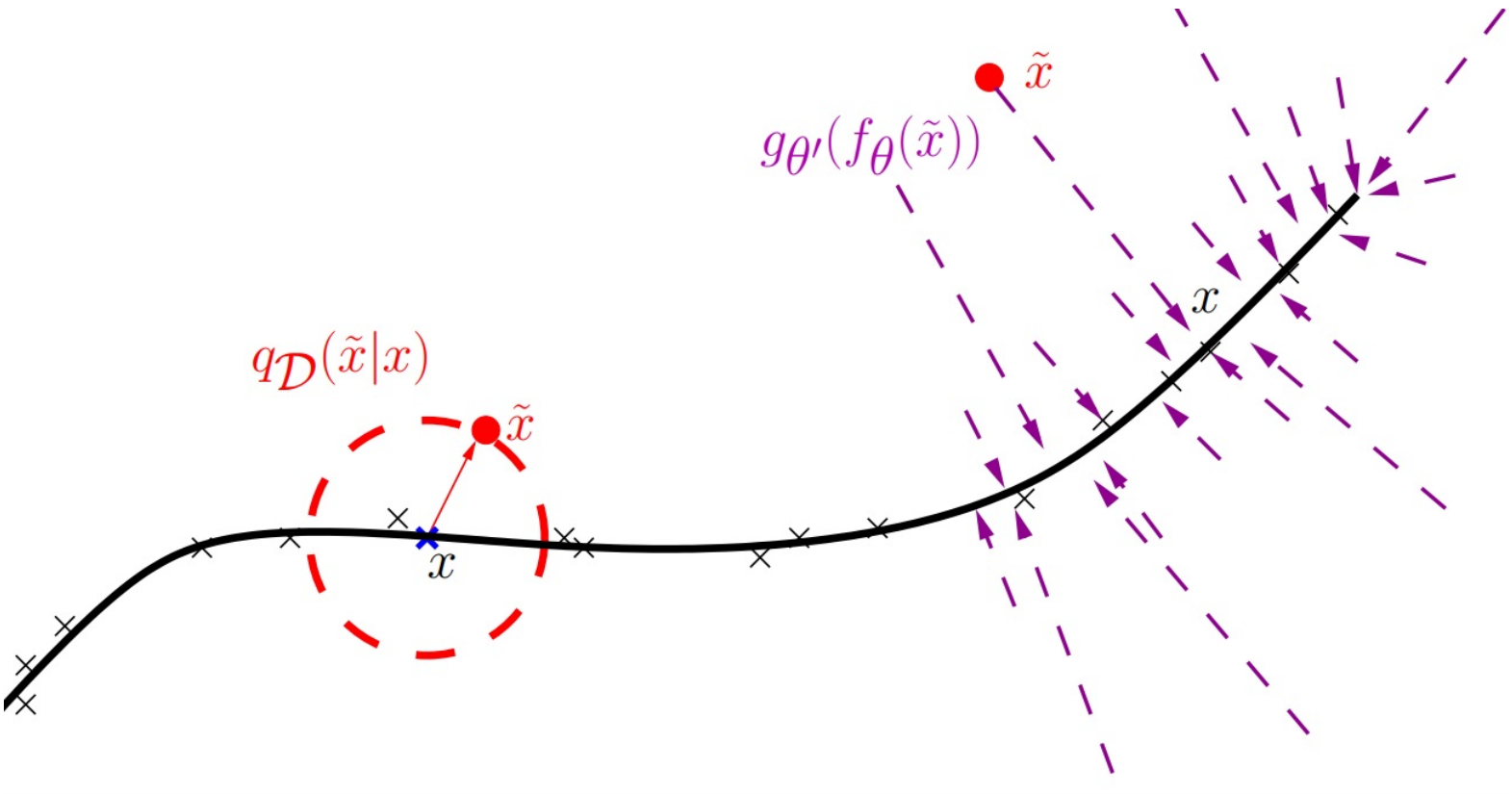


Figure: Suppose training data (x) concentrate near a low dimensional manifold . Corrupted examples (·) obtained by applying a corruption process $q_D(\widetilde{x}|x)$ will lie farther from the manifold. The model learns to project corrupted data back onto the manifold. [1]

DAE: Example

As a concrete example, consider the MNIST dataset. We can add to each pixel a random value generated by a standard normal distribution scaled by a factor. We can train an autoencoder using as input the noisy images, and as output the original images. The model should learn to remove the noise, since it is random in nature and have no relationship with the images.

DAE: Example Cont.

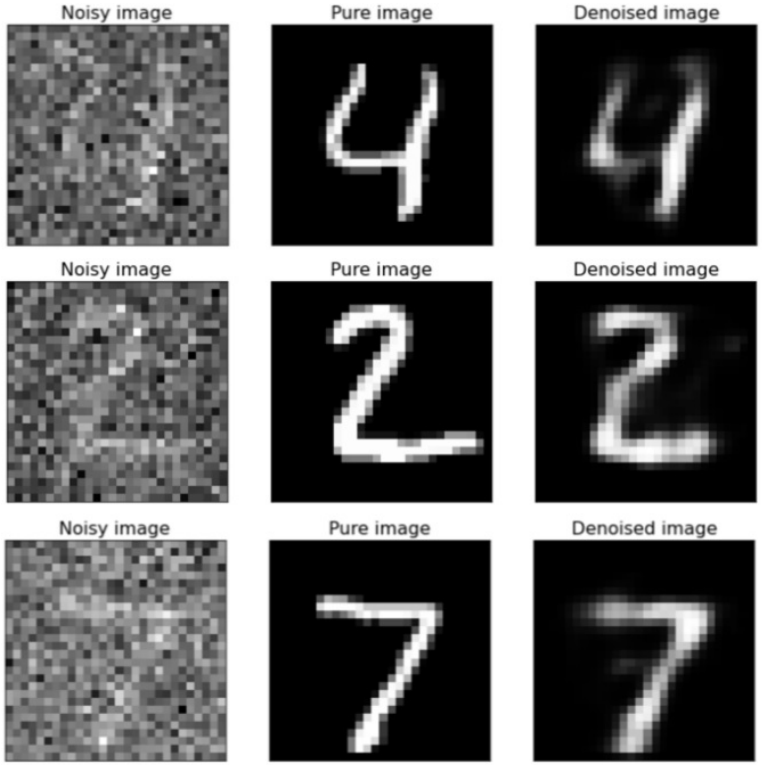


Figure: The noise taken from a standard normal distribution has been added to the testing inputs(image) [2].

Anomaly Detection Algorithm

1. Train an autoencoder on the entire dataset (or if possible, on a portion of the dataset known not to have any outlier)
2. Calculate RE.
3. Sort the observations by the RE.
4. classify the observations with the highest RE as outliers. Note that how many observations are outliers will depend on the problem at hand and require an analysis of the results and usually lot of knowledge of the data and the problem

Anomaly Detection Example

1. We consider autoencoder with only three layers with 784 neurons in the first, 64 in the latent feature generation layer, and again 784 neurons in the output layer;
2. We will train it with the MNIST dataset (60000 training images).
3. Let us choose an image of a shoe from Fashion MNIST and add it to the testing portion of the MNIST dataset (10,001 testing images).

Anomaly Detection: Image

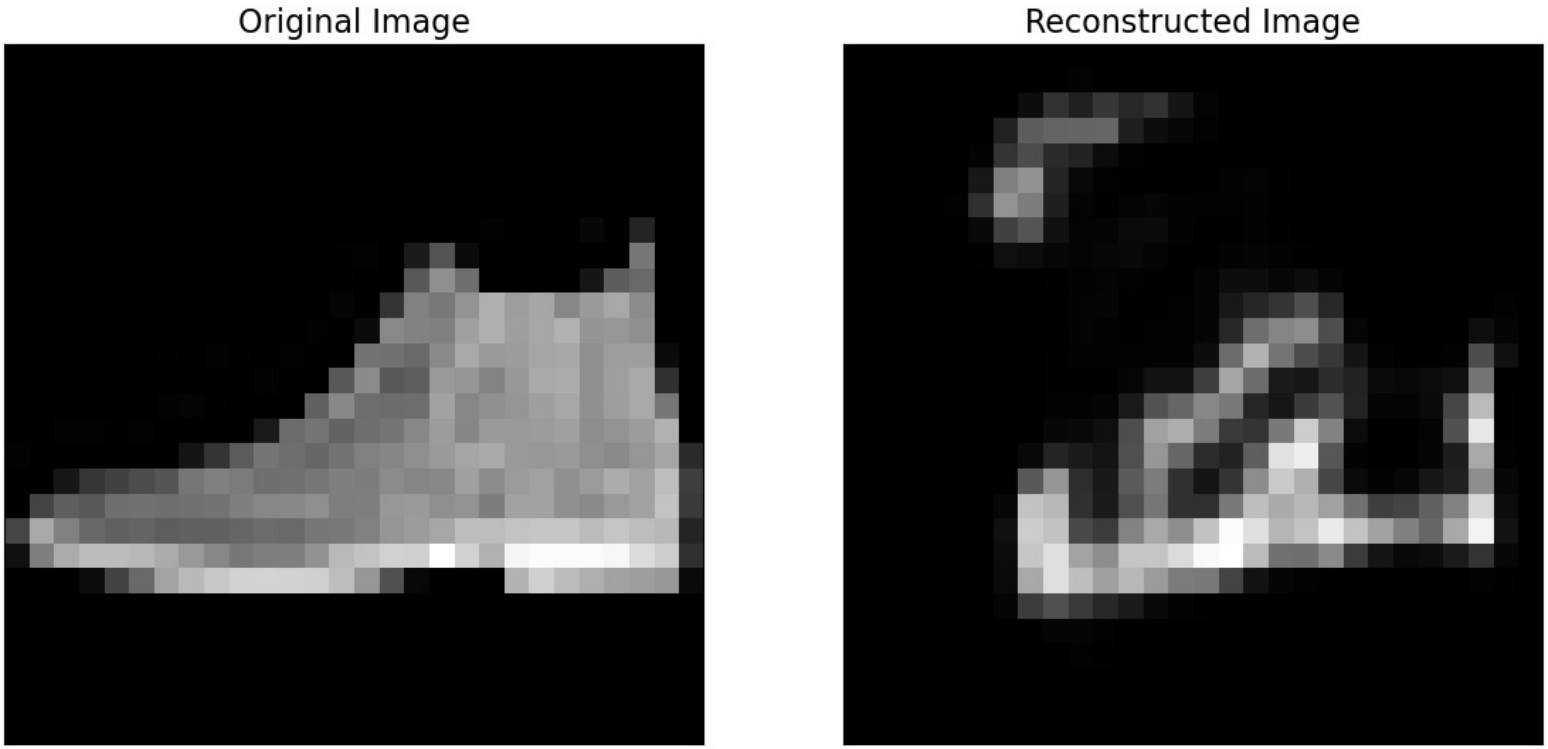


Figure: For this Fashion MNIST shoe RE: 0.054 . The second highest reconstruction error 0.021 is less than half of the RE for the added image.

Anomaly Detection: RE table

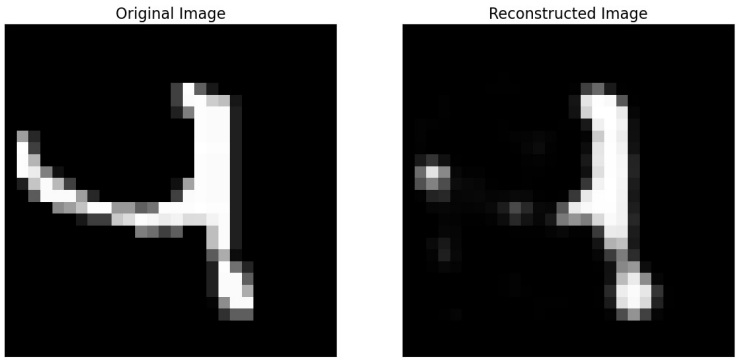


Figure: The image shown (and its reconstructed version) is the one with the second highest reconstruction error.

Sorted RE index	RE:
1	.0542
2	.0214
3	.0204
4	.0201
5	.0200
6	.0190

Table: The sorted Reconstruction Errors

Anomaly Detection: A better Example:

- We will train an autoencoder to detect anomalies on the ECG5000 dataset.
- This dataset contains 5,000 Electrocardiograms, each with 140 data points. Most of them will be normal rhythms.
- Our hypothesis is that the abnormal rhythms will have higher reconstruction error.
- We will then classify a rhythm as an anomaly if the reconstruction error surpasses a fixed threshold.

Example Cont.

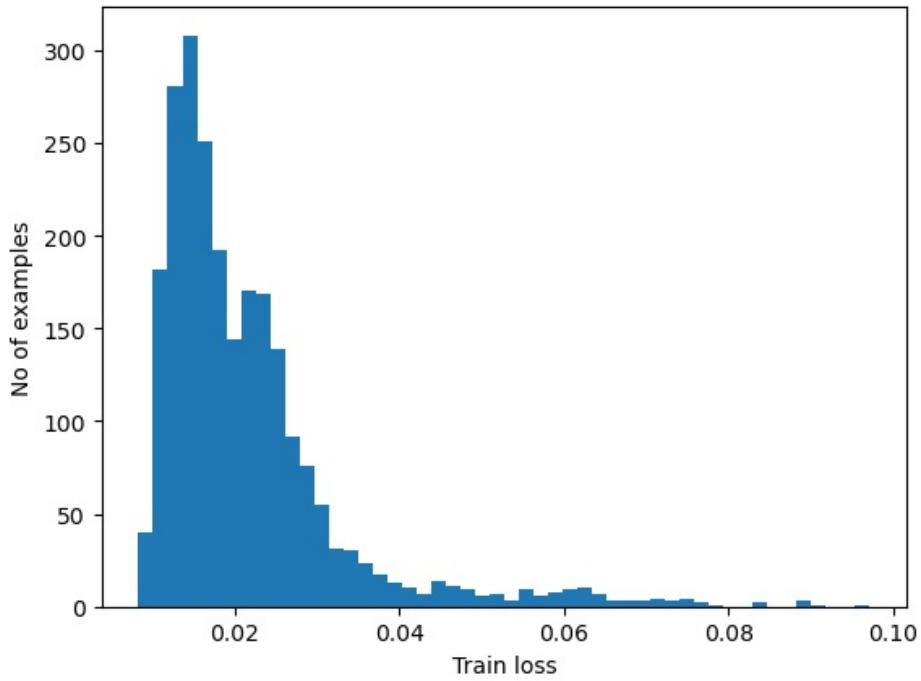


Figure: Histogram of Error Loss vs Number of Training Examples

- We will calculate the mean average error for normal examples from the training set. Mean $= .21$
- then classify future examples as anomalous if the reconstruction error is higher than one standard deviation from the training set. SD $= .32$

Code

Codes are available on Github
https://github.com/AsirTushar/Math526Project

References

1. Pascal Vincent, Yoshua Bengio ,*'Extracting and Composing Robust Features with Denoising Autoencoders.'*
2. Umberto Michelucci, *'An Introduction to Autoencoders.'*
3. Sebastian Raschka, Lecture 16, *'Introduction to Deep Learning and Generative Models'*
4. https://bradleyboehmke.github.io/HOML/autoencoders.html
5. https://blog.roboflow.com/what-is-an-autoencoder-computer-vision/
6. H. Bourlard and Y. Kamp , *'Auto-Association by Multilayer Perceptrons and Singular Value Decomposition '*