AutoEncoders I (DRAFT)

Mark A. Austin

University of Maryland

austin@umd.edu

ENCE 688P, Fall Semester 2021

November 30, 2021

Overview

1. Introduction to AutoEncoders
   - Basic Idea and Applications
2. Related Concepts
   - Dimensionality Reduction
   - Principal Component Analysis
3. Examples
   - Simplest Example: PCA vs AutoEncoder
   - Linear vs Nonlinear Dimensionality Reduction
   - Handwritten Digit Recognition
   - Digit Identification

AutoEncoder

Basic Idea and Applications

Autoencoders

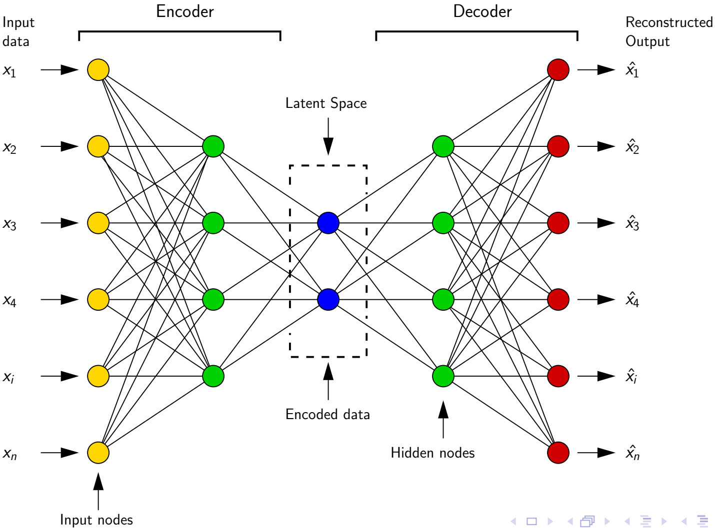
Autoencoder neural networks use unsupervised machine learning algorithms to: (1) find compressed representations of the input data (encoder), and (2) reconstruct the original data from the compressed data (decoder).

Applications:

- Dimensionality reduction.
- Image processing (compression and denoising).
- Feature extraction; anomaly detection.
- Image generation.
- Sequence-to-sequence translation.
- Recommendation systems.

AutoEncoder Architecture

AutoEncoder (Encoder-Decoder-Reconstruction)



AutoEncoder Architecture

Encoder

The encoder learns how to reduce the input dimensions and compress the input data into an encoded representation.

Decoder

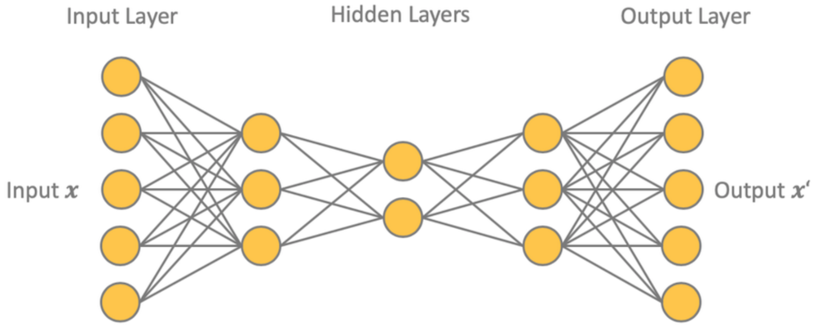
The decoder learns how to reconstruct the input data from the encoded representation and be as close to the input data as possible.

Latent Space

Latent space is simply a representation of compressed data in which similar points are closer together in space. This formalism is useful for learning data features and finding similar representations of data for analysis.

AutoEncoder Design and Analysis

Anomaly Detection ...



Execution of the Network:

$$\sqrt{(x_{new} - x'_{new})^2} > \delta \Rightarrow \text{anomaly}$$

AutoEncoder Design and Analysis

Ideal Requirements. An ideal autoencoder design should have the follow properties:

- **Tied Weights.** Weights in i-th layer of the encoder are equal to the transpose of weights in the i-th layer of the decoder, i.e., $W_i = W_i^T$.
- **Orthogonal Weights.** Weights in the encoder are orthogonal, i.e., $W_i^T . W_i = I$.
- **Uncorrelated Features.** Encoding layer outputs are not correlated.
- **Unit Norm.** The weights have unit norm, i.e.,

$$\sum_{j=1}^{p} w_{ij}^2 = 1 \text{ for } i = 1 \cdots k. \quad (1)$$

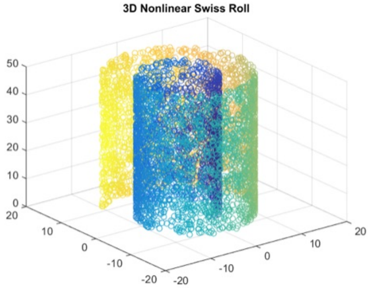
Related Concepts

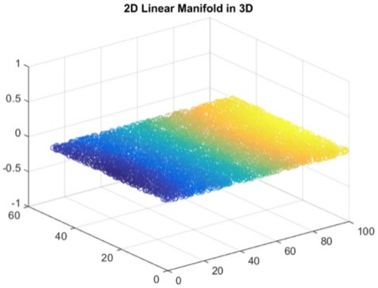
Dimensionality Reduction

Dimensionality Reduction

Strategies of dimensionality reduction involve transformation of data to new (lower) dimension in such a way that some of the dimensions can be discarded without a loss of information.

Example: Projection of Swiss Roll data in 3D to 2D ...

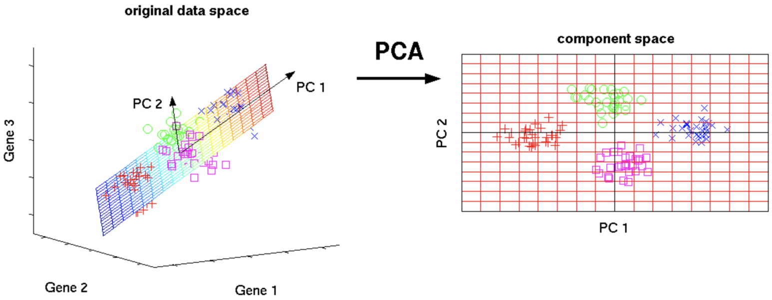




Principal Component Analysis

Principal Component Analysis

Principal component analysis is an orthogonal linear transformation that transforms mean-centered data into a new coordinate system such that the variance of the projected data is maximized along the new axes – the latter is called the first principal component.



Principal Component Analysis

Interpretation of Principal Components:

- The first principal component can be defined as the direction that maximizes the variance of the projected data.
- The i-th principal component is a direction that maximizes the variance in the projected data and is orthogonal to the first i-1 principal components.

Applications:

- Exploratory data analysis.
- Dimensionality reduction.

Note: In general, dimensionality reduction loses information. PCA-based reduction procedures tend to minimize information loss.

Principal Component Analysis

Mathematical Procedure: Suppose that our dataset comprises n m-dimensional data points:

$$X = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1m} \\ x_{21} & x_{22} & \cdots & x_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & \cdots & x_{nm} \end{bmatrix} \tag{2}$$

Step 1. Compute the mean value for each dimension in the dataset:

$$\bar{X} = \begin{bmatrix} \bar{x}_1 & \bar{x}_2 & \cdots & \bar{x}_m \end{bmatrix} \tag{3}$$

Principal Component Analysis

Step 2. Compute the $m \times m$ covariance matrix:

$$\text{Cov} = \begin{bmatrix} cov_{11} & cov_{12} & \cdots & cov_{1m} \\ cov_{21} & cov_{22} & \cdots & cov_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ cov_{m1} & cov_{m2} & \cdots & cov_{mm} \end{bmatrix} \tag{4}$$

where,

$$cov_{ij} = \frac{1}{n-1} \sum_{k=1}^{n} (x_{ki} - \bar{x}_i)(x_{kj} - \bar{x}_j) \tag{5}$$

Step 3. Compute the covariance matrix eigenvalues/eigenvectors:

$$[\text{Cov}]\, W = \lambda W. \tag{6}$$

Principal Component Analysis

Step 4. Sort eigenvalues by decreasing order.

Step 5. Choose k ($k \leq m$) largest eigenvalues/eigenvectors to form $m \times k$ matrix:

$$W = \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1k} \\ w_{21} & w_{22} & \cdots & w_{2k} \\ \vdots & \vdots & \ddots & \vdots \\ w_{m1} & w_{m2} & \cdots & w_{mk} \end{bmatrix}. \tag{7}$$

Step 6. Transform raw data X onto k-dimensional subspace Y:

$$Y = X \cdot W. \tag{8}$$

Principal Component Analysis

Example 1: Define straight line segment + noisy data:

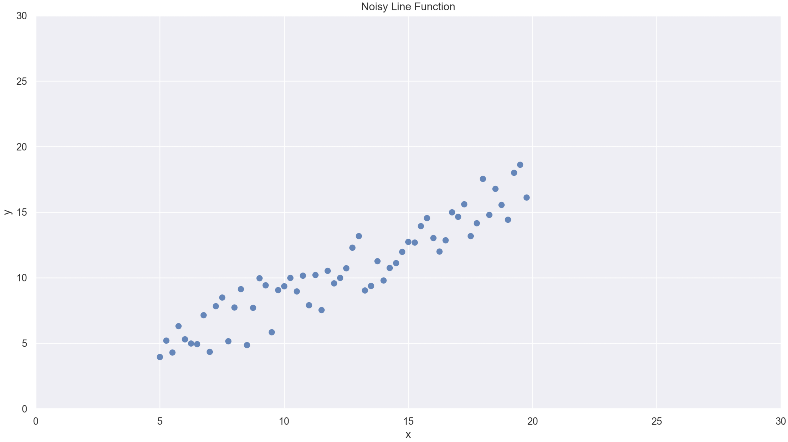
```
def NoisyLineFunction (a,b,x):
    return a + b*x + 5*(random.random() - 1.0)
```

Generate and Plot Raw Data: (x,y) coordinates:

```
    5.00     3.96
    5.25     5.22
    5.50     4.30
    5.75     6.32

    ... data values removed ...

   19.00    14.44
   19.25    18.02
   19.50    18.62
   19.75    16.12
```

Principal Component Analysis

Two-dimensional plot of raw data:



Principal Component Analysis

Compute Principal Components (No components = 2):

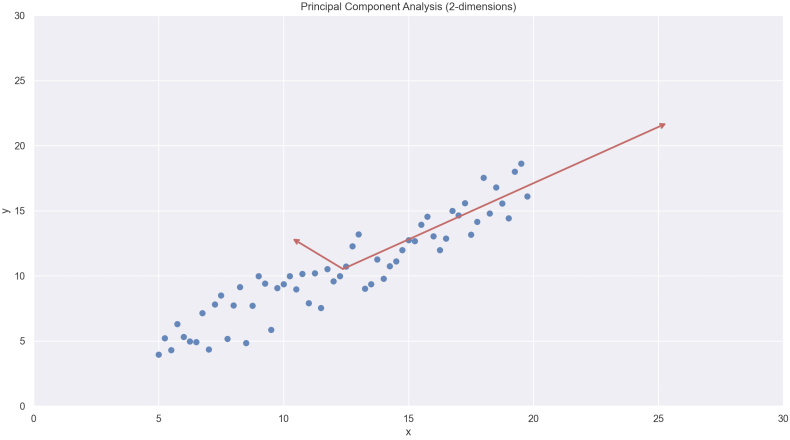
```
pca = PCA(n_components=2)
pca.fit(X)
```

Mean values; Eigenvalue and Eigenvectors

```
--- Print mean ...
[12.375     10.5313869]

--- Print components (first and second eigenvectors) ...

[ [ 0.75668904  0.65377496]     <-- first eigenvector ...
  [-0.65377496  0.75668904] ]   <-- second eigenvector ...

--- Print variance ...
[32.50694888  1.05218475]
```

Principal Component Analysis

Principal Components in Two Dimensions:



Principal Component Analysis

Dimensionality Reduction Platform (one dimension)

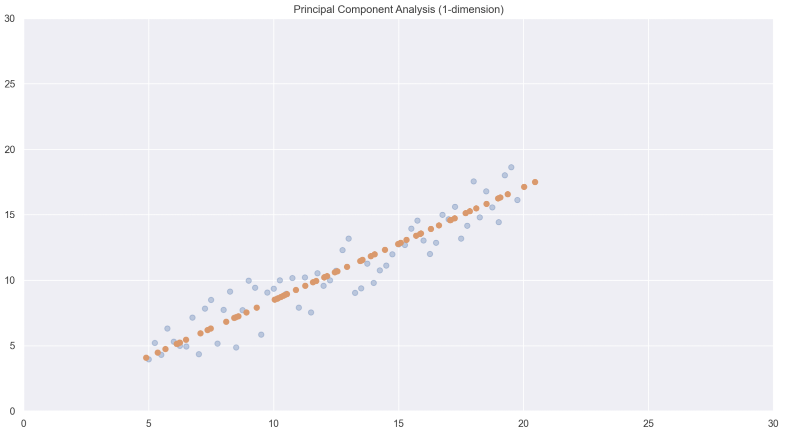
```
pca = PCA(n_components=1)
pca.fit(X)
X_pca = pca.transform(X)

print("--- original shape:    ", X.shape)
print("--- transformed shape:", X_pca.shape)

# Compute inverse transform on reduced data ...

X_new = pca.inverse_transform(X_pca)
```

Shape of Original and Transformed Data

```
--- original shape:    (60, 2)
--- transformed shape: (60, 1)
```

Principal Component Analysis

Principal Component Analysis in One Dimension:



Principal Component Analysis

Components and Variance (as above)

```
--- Print components ...
[[0.75668904 0.65377496]]
```

```
--- Print variance ...
[32.50694888]
```

Side-by-Side Comparison of Coordinates

```
2D Coords (x,y)  --> 1D Coord System --> Inverse Transform Coords
=================================================================
  5.00     3.96               -9.88                4.90     4.07
  5.25     5.22               -8.87                5.67     4.74
  5.50     4.30               -9.27                5.36     4.47
  ....     ....                ....                ....     ....
 19.00    14.44                7.57               18.10    15.48
 19.25    18.02               10.10               20.02    17.13
 19.50    18.62               10.68               20.46    17.51
 19.75    16.12                9.23               19.36    16.57
=================================================================
```

Principal Component Analysis

Strengths:

- Principal component analysis learns the linear transformation of data.
- Principal component analysis represents data in lower dimensions via an optimal orthogonal tranformation.
- Implementations are fast.

Weaknesses:

- Principal component analysis learns the linear transformation of data.
- As the number of features increases, the chances of overfitting of the model decreases significantly.

Source Code: See: python-code.d/autoencoder/

AutoEncoder vs PCA

Autoencoder vs PCA

A linearly activated autoencoder approximates principal component analysis. Mathematically, minimizing the reconstruction error in PCA modeling is the same as a single layer autoencoder.

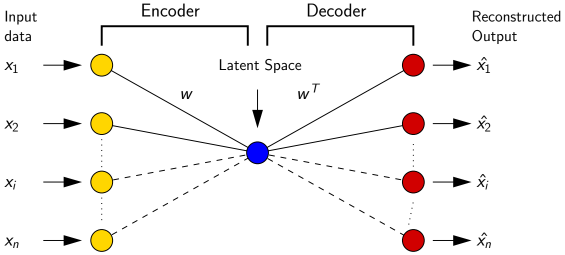
Extensions of Autoencoder to Nonlinear Spaces

Autoencoders are nonlinear extensions of PCA.

Examples

Example 1. Simplest Example

Simplest Example. No nonlinear transformation.



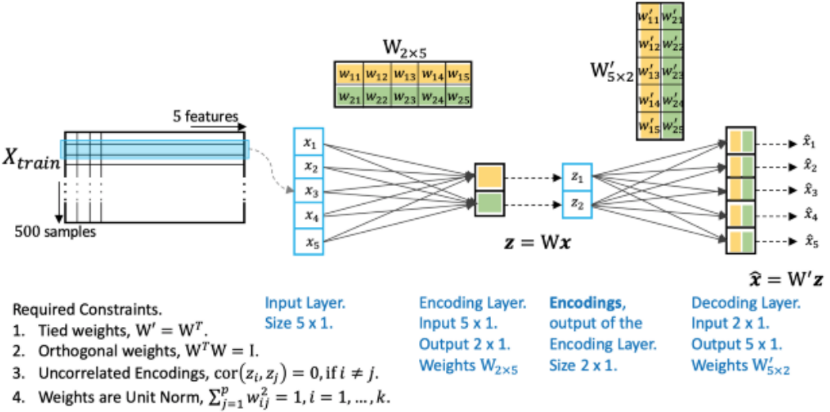
Features:

- A single hidden unit.
- Hidden unit has a **linear activation**.

Use same weight vector w for encoder/decoder. This is PCA.

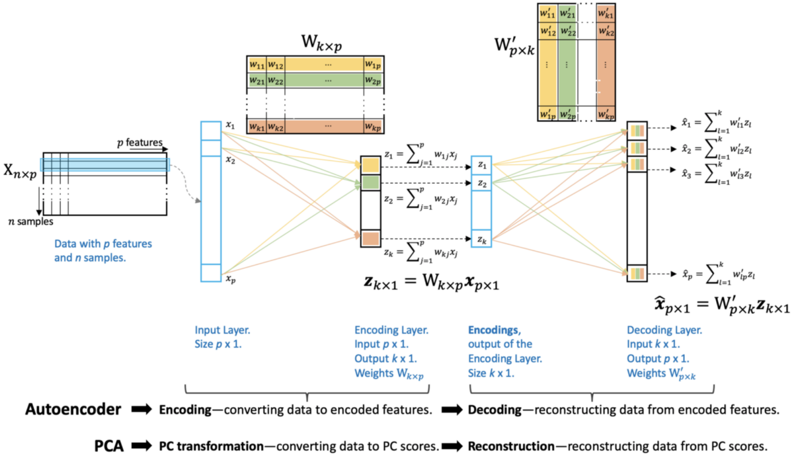
Example 1. Simplest Example

Schematic for Tied Weights:



Example 1. Simplest Example

Schematic for Tied Weights (Autoencoder vs PCA):



Example 1. Simplest Example

Training Procedure

Learn network weights by minimizing $L2$ divergence, i.e.,

$$L_2^2 = \underset{W}{arg\ min}\ E\left[\|x - \hat{x}\|^2\right] = \underset{W}{arg\ min}\ E\left[\|x - w^T wx\|^2\right] \tag{9}$$

Rewriting equation 9 as a matrix summation:

$$L_2^2 = \underset{W}{arg\ min} \sum_{i=1}^{n} \left[x_i - w^T wx_i\right]^T \left[x_i - w^T wx_i\right] \tag{10}$$

Here:

- x_i is a $(p \times 1)$ (p-dimensional) vector for the i-th data input.
- w is a $(1 \times p)$ vector of weights.

Example 1. Simplest Example

Training Procedure (Cont'd)"

$$L_2^2 = \underset{W}{arg\ min} \sum_{i=1}^{n} \left[x_i^T - x_i^T w^T w\right]\left[x_i - w^T w x_i\right].$$

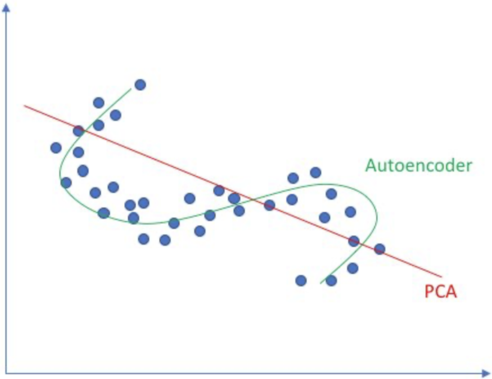
$$= \underset{W}{arg\ min} \sum_{i=1}^{n} \left[x_i^T x_i - 2x_i^T w^T w x_i + x_i^T w^T w w^T w x_i\right].$$

Equation 9 will be minimized when $w^T w$, a $(p \times p)$ matrix, equals I.

Example 1. Simplest Example

Example 2. Linear vs Nonlinear Dimensionality Reduction

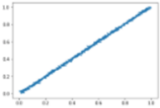
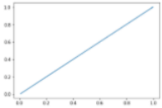
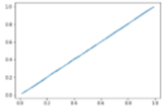
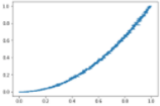
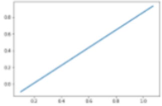
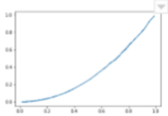
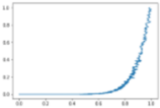
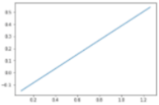
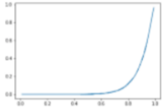
Linear vs Nonlinear Dimensionality Reduction:



Source: Nugroho H. et al., 2020

Example 2. Linear vs Nonlinear Dimensionality Reduction

Linear vs Nonlinear Dimensionality Reduction:

Function	Feature Space	PCA Reconstruction	Auto Encoder Reconstruction
y=mx+c			
y=mx²+c			
y=mx⁸+c			

Example 2. Linear vs Nonlinear Dimensionality Reduction

DL4J: Dataset

Example 2. Linear vs Nonlinear Dimensionality Reduction

DL4J: Neural Network Architecture

Example 3. Anomaly Detection

Example 4. Handwritten Digit Recognition

Problem Statement

- Demonstrate anomaly detection on MNIST using simple autoencoder.
- Goal is to identify digits that are unusual.

MNIST Handwritten Digit Dataset

- MNIST (Modified National Institute of Standards and Technology) is a database of handwritten digits provided by NIST.
- The database contains: 60,000 training images and 10,000 testing images.
- Each image is a scan of a handwritten image 0 through 9.
- Differences among images are due to variations in handwriting style.

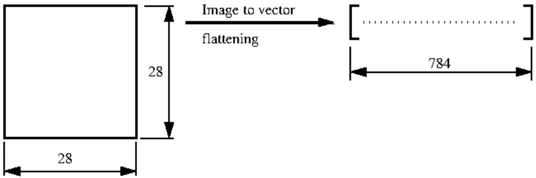
Example 4. Handwritten Digit Recognition

Sample Digits

Example 4. Handwritten Digit Recognition

Solution Procedure

- Images are centered on a 28x28 grid of pixels. Individual pixels take a value 0 through 255.
- Individual 28x28 images ⟶ 1x784 data vector.



- Create network configutation with 784 inputs/outputs, contracting down to a ten-dimensional embedding vector, i.e., 784 -> 250 -> 10 -> 250 -> 784.

Example 4. Handwritten Digit Recognition

DL4J: Training Dataset

DL4J: Testing Dataset

Example 4. Handwritten Digit Recognition

DL4J: Network Configuration: 784 -> 250 -> 10 -> 250 -> 784.

```
MultiLayerConfiguration conf = new NeuralNetConfiguration.Builder()
         .seed(12345)
         .weightInit(WeightInit.XAVIER)
         .updater(new AdaGrad(0.05))
         .activation(Activation.RELU)
         .l2(0.0001)
         .list()
         .layer(new DenseLayer.Builder().nIn(784).nOut(250)
             .build())
         .layer(new DenseLayer.Builder().nIn(250).nOut(10)
             .build())
         .layer(new DenseLayer.Builder().nIn(10).nOut(250)
             .build())
         .layer(new OutputLayer.Builder().nIn(250).nOut(784)
             .activation(Activation.LEAKYRELU)
             .lossFunction(LossFunctions.LossFunction.MSE)
             .build())
         .build();

MultiLayerNetwork net = new MultiLayerNetwork(conf);
net.setListeners(Collections.singletonList(new ScoreIterationListener(10)));
```

Example 4. Handwritten Digit Recognition

DL4J: Summary of Network Model.

```
=================================================================
LayerName (LayerType)   nIn,nOut   Params  ParamsShape
=================================================================
layer0    (DenseLayer)   784,250  196,250  W:{784,250}, b:{1,250}
layer1    (DenseLayer)    250,10    2,510  W:{250,10}, b:{1,10}
layer2    (DenseLayer)    10,250    2,750  W:{10,250}, b:{1,250}
layer3    (OutputLayer)  250,784  196,784  W:{250,784}, b:{1,784}
-----------------------------------------------------------------
Total Parameters: 398,294  Trainable Parameters: 398,294
=================================================================
```

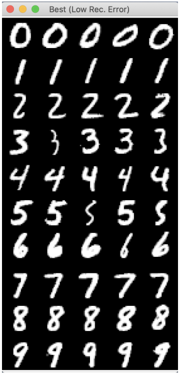
Example 4. Handwritten Digit Recognition

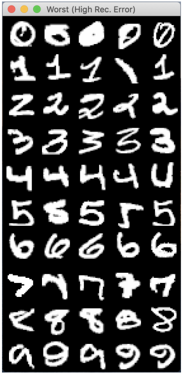
Training Procedure:

Compute Reconstruction Error:

Example 4. Handwritten Digit Recognition

Results: Best (left) and Worst (right) Results





[Best, Worst] = [low, high] reconstruction error.

Example 4. Handwritten Digit Recognition

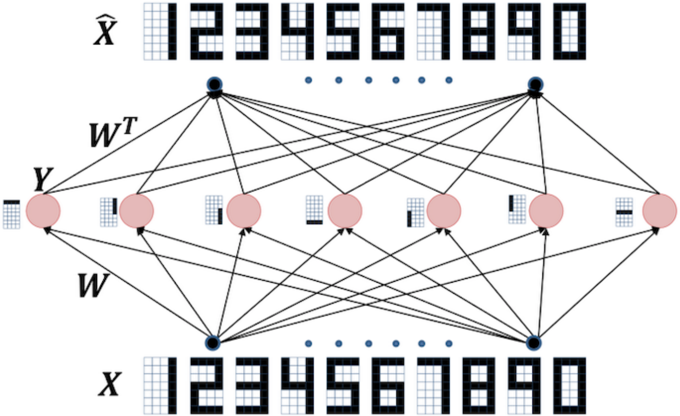
Strengths

- ...
- ...
- ...

Weaknesses

- Flattening the image looses spatial information, which, in turn, can simplify the learning process.
- ...

Example 5. Digit Identification



- A neural network can be trained to predict the input itself
- This is an *autoencoder*
- An *encoder* learns to detect all the most significant patterns in the signals

Example 5. Digit Identification

References

- Aggarwal C.C., Neural Networks and Deep Learning, Springer, 2018.
- Nugroho H., Susanty M., Irawan A., Koyimatu M., and Yunita A., Fully Convolutional Variational Autoencoder for Feature Extraction of Fire Detection System, Journal of Computer Science and Engineering, Vol. 13, No. 1, 2020.
- Watt J., Borhani R., Katsaggelos A.K., Machine Learning Refined, Second Edition, Cambridge University Press, 2020.