Machine Learning

Lesson 10

PCA & Autoencoders,

Lecturer: Dr. Msagha J Mbogholi, PhD

Flashback from Lesson 9

- By using a sample that is a small subset of the larger population of interest we can calculate probabilities which will be fairly accurate; thus it can be fairly stated that the results of the sample are representative of the population
- Probabilistic classification algorithms will use an inferential way to determine the class of a given example
- A conditional probability denoted P(y | x) is the probability of y happening given that x has happened.
- Bayes theorem states as follows: P (B|E) = P(E|B) * P(B) / P(E)
- It is important to note that naïve Bayes classifiers is the name used by all those that use Bayes theorem in classification. They are called naïve since they assume that each input variable is independent. Further each input variable contributes equally to the outcome. The independence means that no input variable depends on any other.
- All naïve Bayes algorithms fall under 3 groups: Multinomial naive Bayes, Gaussian naive Bayes and Bernoulli naive Bayes

Content

- Principal component Analysis(PCA)
- Autoencoders
- PCA VS Autoencoders

Part 1

Principal Component Analysis(PCA)

1.1 Principal component analysis

- Principal Component Analysis, or PCA, is a dimensionality-reduction method that is often used to reduce the dimensionality of large data sets, by transforming a large set of variables into a smaller one that still contains most of the information in the large set[3].
- It allows us to take an n-dimensional feature-space and reduce it to
- a kk-dimensional feature-space while maintaining as much information from the original dataset as possible in the reduced dataset.
- Specifically, PCA will create a new feature-space that aims to capture
- as much variance as possible in the original dataset[3].
- So to sum up, the idea of PCA is simple — reduce the number of variables of a data set, while preserving as much information as possible.

1.2 Dimensionality Reduction

- Dimensionality reduction is the process of taking data in a high dimensional space and mapping it into a new space whose dimensionality is much smaller.
- This process is closely related to the concept of (lossy) compression in information theory. There are several reasons to reduce the dimensionality of the data. [1]
- These are some of the reasons why dimensionality reduction is useful. However, it can also remove noise, significantly improve the results of the learning algorithm, make the dataset easier to work with, and make the results easier to understand. [2]

1.3 Steps by step explanation of PCA

1. Standardize the range of continuous initial variables
2. Compute the covariance matrix to identify correlations
3. Compute the eigenvectors and eigenvalues of the covariance matrix to identify the principal components
4. Create a feature vector to decide which principal components to keep
5. Recast the data along the principal components axes

1.3.1. Standardization of Linear variables

- The aim of this step is to standardize the range of the continuous initial variables so that each one of them contributes equally to the analysis.
- More specifically, the reason why it is critical to perform standardization prior to PCA, is that the latter is quite sensitive regarding the variances of the initial variables. That is, if there are large differences between the ranges of initial variables, those variables with larger ranges will dominate over those with small ranges (For example, a variable that ranges between 0 and 100 will dominate over a variable that ranges between 0 and 1), which will lead to biased results. So, transforming the data to comparable scales can prevent this problem.
- Mathematically, this can be done by subtracting the mean and dividing by the standard deviation for each value of each variable as shown in Equation 1.

$$z = \frac{\text{value-mean}}{\text{standard deviation}} \quad \text{.....Equation 1}$$

- Once the standardization is done, all the variables will be transformed to the same scale.

1.3.2 Compute the covariance matrix

- The aim of this step is to understand how the variables of the input data set are varying from the mean with respect to each other, or in other words, to see if there is any relationship between them, because sometimes, variables are highly correlated in such a way that they contain redundant information. So, in order to identify these correlations, we compute the covariance matrix.
- The covariance matrix is a $p \times p$ symmetric matrix (where p is the number of dimensions) that has as entries the covariances associated with all possible pairs of the initial variables. For example, for a 3-dimensional data set with 3 variables x, y, and z, the covariance matrix is a 3×3 matrix of this from:

$$\begin{matrix} Cov(x,y) & Cov(x,y) & Cov(x,z) \\ Cov(y,x) & Cov(y,y) & Cov(y,z) \\ Cov(z,x) & Cov(z,y) & Cov(z,z \end{matrix}$$

Covariance Matrix for 3- Dimensional data

- if positive then : the two variables increase or decrease together (correlated)
- if negative then : One increases when the other decreases (Inversely correlated)

1.3.4 Compute the eigenvectors and eigenvalues of the covariance matrix to identify the principal components

- Eigenvectors and eigenvalues are the linear algebra concepts that we need to compute from the covariance matrix in order to determine the ***principal components*** of the data.
- Before getting to the explanation of these concepts, let's first understand what do we mean by principal components.
- Principal components are new variables that are constructed as linear combinations or mixtures of the initial variables.
- These combinations are done in such a way that the new variables (i.e., principal components) are uncorrelated and most of the information within the initial variables is squeezed or compressed into the first components.
- So, the idea is 10-dimensional data gives you 10 principal components, but PCA tries to put maximum possible information in the first component, then maximum remaining information in the second and so on

1.3.5 Create a feature vector to decide which principal components to keep

- As we saw in the previous step, computing the eigenvectors and ordering them by their eigenvalues in descending order, allows us to find the principal components in order of significance.
- In this step, what we do is, to choose whether to keep all these components or discard those of lesser significance (of low eigenvalues), and form with the remaining ones a matrix of vectors that we call Feature vector.
- So, the feature vector is simply a matrix that has as columns the eigenvectors of the components that we decide to keep. This makes it the first step towards dimensionality reduction, because if we choose to keep only p eigenvectors (components) out of n, the final data set will have only p dimensions.

1.3.6. Recast the data along the principal components axes

- In the previous steps, apart from standardization, you do not make any changes on the data, you just select the principal components and form the feature vector, but the input data set remains always in terms of the original axes (i.e, in terms of the initial variables).
- In this step, which is the last one, the aim is to use the feature vector formed using the eigenvectors of the covariance matrix, to reorient the data from the original axes to the ones represented by the principal components (hence the name Principal Components Analysis). This can be done by multiplying the transpose of the original data set by the transpose of the feature vector.

FinalDataset=Feature Vector T*.* StandardizedOriginalDataSet (Mbogho, 2019)

An example of PCA

- The next few slides are based on an example that can be found in https://medium.com/analytics-vidhya/understanding-principle-component-analysis-pca-step-by-step-e7a4bb4031d9.
- All the tables used in the example are from the same source.

1.4 An Example of PCA

1. Standardize the Dataset

Assume we have the below dataset which has 4 features and a total of 5 training examples.

F_1	F_2	F_3	F_4
1	2	3	4
5	5	6	7
1	4	2	3
5	3	2	1
8	1	2	2

Table 1 : Un-standardized Dataset [7]

First, we need to standardize the dataset and for that, we need to calculate the mean and standard deviation for each feature as shown in Equation 2.

$$X_{new} = \frac{x-\mu}{\alpha} \quad \text{.....Equation 2}$$

1.4 An Example of PCA (cont'd)

	F1	F2	F3	F4
μ	4	3	3	3.4
α	3	1.58114	1.73205	2.30217

Table 2 : Mean and Standard Deviation before standardization [7]

After applying the formula for each feature in the dataset is transformed as below:

F1	F2	F3	F4
-1	-0.63245	0	0.26062
0.33333	1.26491	1.73205	1.56374
-1	0.63246	-0.57735	-0.17375
0.33333	0	-0.57735	-1.04249
1.33333	-1.26491	-0.57735	-0.60812

Table 3 : Standardized Dataset [7]

1.4 An Example of PCA (cont'd)

- **Calculate the covariance matrix for the whole dataset**
- The formula to calculate the covariance matrix:

For Population

$$Cov(x,y) = \frac{\sum (x_i - \bar{x}) * (y_i - \bar{y})}{N}$$

For Sample

$$Cov(x,y) = \frac{\sum (x_i - \bar{x}) * (y_i - \bar{y})}{(N-1)}$$

Covariance Formula

the covariance matrix for the given dataset will be calculated as below

	F1	F2	F3	F4
F1	Var(F1)	Cov(F1,F2)	Cov(F1,F3)	Cov(F1,F4)
F2	Cov(F2,F1)	Var(F2)	Cov(F2,F3)	Cov(F2,F4)
F3	Cov(F3,F1)	Cov(F3,F2)	Var(F3)	Cov(F3,F4)
F4	Cov(F4,F1)	Cov(F4,F2)	Cov(F4,F3)	Var(F4)

Table 4 : Covariance Matrix [7]

1.4 An Example of PCA (cont'd)

- Since we have standardized the dataset, so the **mean for each feature is o** and the standard deviation is 1.
- var(f1) = ((-1.0-0)² + (0.33-0)² + (-1.0-0)² +(0.33–0)² +(1.33–0)²)/5
  var (f1) = 0.8

- cov(f1,f2) =
  ((-1.0–0)*(-0.632456-0) +
  (0.33–0)*(1.264911-0) +
  (-1.0–0)* (0.632456-0)+
  (0.33–0)*(0.000000 -0)+
  (1.33–0)*(-1.264911–0))/5
  cov(f1,f2 = -0.25298

- In the similar way be can calculate the other covariances and which will result in the below covariance matrix

	f1	f2	f3	f4
f1	0.8	-0.25298	0.03849	-0.14479
f2	-0.25298	0.8	0.51121	0.4945
f3	0.03849	0.51121	0.8	0.75236
f4	-0.14479	0.4945	0.75236	0.8

covariance matrix (population formula)

Table 5 : covariances Matrix [7]

1.4 An Example of PCA (cont'd)

- **3. Calculate eigenvalues and eigen vectors.**
- An **eigenvector** is a nonzero vector that changes at most by a scalar factor when that linear transformation is applied to it. The corresponding **eigenvalue** is the factor by which the eigenvector is scaled.
- Let A be a square matrix (in our case the covariance matrix), ν a vector and λ a scalar that satisfies $Aν = λν$, then λ is called eigenvalue associated with eigenvector ν of A.
  Rearranging the above equation

$Aν-λν = o$; $(A-λI)ν = o$

- Since we have already know ν is a non- zero vector, only way this equation can be equal to zero, if

$det(A-λI) = o$

	f1	f2	f3	f4
f1	0.8 - λ	-0.25298	0.03849	-0.14479
f2	-0.25298	0.8- λ	0.51121	0.4945
f3	0.03849	0.51121	0.8 - λ	0.75236
f4	-0.14479	0.4945	0.75236	0.8 - λ

A-λI = o

Table 6 : $det(A-λI) = o$ [7]

1.4 An Example of PCA (cont'd)

- Solving the above equation = 0
- ***λ = 2.51579324 , 1.0652885 , 0.39388704 , 0.02503121***
- Solving the (A-λI)v = 0 equation for v vector with different λ values as shown in Table 7:

$$\begin{pmatrix} 0.800000-\lambda & -(0.252982) & 0.038490 & -(0.144791) \\ -(0.252982) & 0.800000-\lambda & 0.511208 & 0.494498 \\ 0.038490 & 0.511208 & 0.800000-\lambda & 0.752355 \\ -(0.144791) & 0.494498 & 0.752355 & 0.800000-\lambda \end{pmatrix} \times \begin{pmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \end{pmatrix} = 0$$

Table 7 :v vector with different λ values [7]

For λ = *2.51579324, solving the above equation using Cramer's rule, the values for v vector are*

v1 = 0.16195986
v2 = -0.52404813
v3 = -0.58589647
v4 = -0.59654663

1.4 An Example of PCA (cont'd)

- Going by the same approach, we can calculate the eigen vectors for the other eigen values. We can from a matrix using the eigen vectors.

E1	E2	E3	E4
0.161960	-0.917059	-0.37071	0.196162
-0.524048	0.206922	-0.817319	0.120610
-0.585896	-0.320539	0.188250	-0.720099
-0.596547	-0.115935	0.44933	-.65447

Table 8 : Eigen Vector(4*4) Matrix [7]

4. **Sort eigenvalues and their corresponding eigenvectors.**

Since eigenvalues are already sorted in this case so no need to sort them again.

1.4 An Example of PCA (cont'd)

5. Pick k eigenvalues and form a matrix of eigenvectors

- If we choose the top 2 eigenvectors, the matrix will look like this:

E1	E2
0.161960	-0.917059
-0.524048	0.206922
-0.585896	-0.320539
-0.596547	-0.115935

Table 9 : top 2 Eigen Vectors (4*2 Matrix) [7]

6. Transform the original matrix.

Feature matrix * top k eigenvectors = Transformed Data

f1	f2	f3	f4
-1.000000	-0.632456	0.000000	0.260623
0.333333	1.264911	1.732051	1.563740
-1.000000	0.632456	-0.577350	-0.173749
0.333333	0.000000	-0.577350	-1.042493
1.333333	-1.264911	-0.577350	-0.608121

(5,4)

*

e1	e2
0.161960	-0.917059
-0.524048	0.206922
-0.585896	-0.320539
-0.596547	-0.115935

(4,2)

=

nf1	nf2
0.014003	0.755975
-2.556534	-0.780432
-0.051480	1.253135
1.014150	0.000239
1.579861	-1.228917

(5,2)

Table 10 : Data Transformation [7]

1.4 An Example of PCA (cont'd)

Sample code for the PCA example [7]

```
import numpy as np
import pandas as pd
A = np.matrix([[1,2,3,4],
               [5,5,6,7],
               [1,4,2,3],
               [5,3,2,1],
               [8,1,2,2]])

df = pd.DataFrame(A,columns  = ['f1','f2','f3','f4'])
df_std  = (df - df.mean()) / (df.std())
n_components = 2
from sklearn.decomposition import PCA
pca = PCA(n_components=n_components)
principalComponents = pca.fit_transform(df_std)
principalDf = pd.DataFrame(data=principalComponents,columns=['nf'+str(i+1) for i in range(n_components)])
print(principalDf)
```

```
        nf1       nf2
0 -0.014003  0.755975
1  2.556534 -0.780432
2  0.051480  1.253135
3 -1.014150  0.000239
4 -1.579861 -1.228917
```

code snippet for PCA using Sklearn

Part 2

Autoencoders

2.1 Introduction

- Artificial intelligence encircles a wide range of technologies and techniques that enable computer systems to solve problems like Data Compression which is used in computer vision, computer networks, computer architecture, and many other fields.
- Compression reduces the cost of storage, increases the speed of algorithms, and reduces the transmission cost. Compression is achieved by removing redundancy, that is repetition of unnecessary data.
- Autoencoders are surprisingly simple neural architectures. They are basically a form of compression, similar to the way an audio file is compressed using MP3, or an image file is compressed using JPEG.
- **Autoencoders** are *unsupervised neural networks* that use machine learning to do this compression for us.
- Autoencoders are used to reduce complexity of input data, for unsupervised learning problems and anomaly detection in data.

.

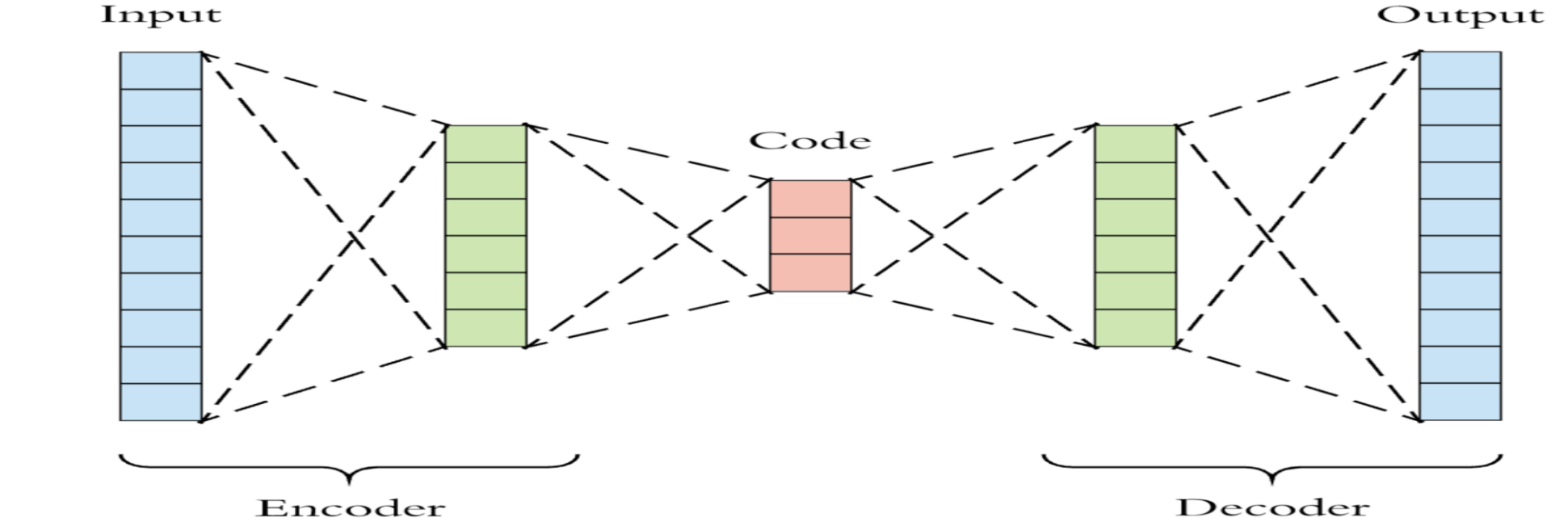
2.2 Autoencoders

- Autoencoders are used to help reduce noise in data. Through the process of compressing input data, encoding it, and then reconstructing it as an output, autoencoders allow you to reduce dimensionality and focus only on areas of real value[4].
- It is used to efficiently learn the data representation or representation space in an unsupervised manner. The main purpose is to learn a reduced representation of the input data.
- Autoencoders are closely related to principal component analysis (PCA). In fact, if the activation function used within the autoencoder is linear within each layer, the latent variables present at the bottleneck (the smallest layer in the network, aka. code) directly correspond to the principal components from PCA.
- Autoencoders work well if correlation exists between input data(performs poorly if all the input data is independent)
- Formally, we can say, an autoencoders describes a nonlinear relationship of an input to an output through an intermediate representation called code(aka bottleneck) as shown in Figure 1 .

2.2 Autoencoders (cont'd)

Autoencoders use the same input data for the same input and output.

Figure 1 : Autoencoder(Mathew,2019)



Source: https://towardsdatascience.com/generating-images-with-autoencorders

2.3 Architecture of Autoencoders

- **Architecture of Autoencoders**

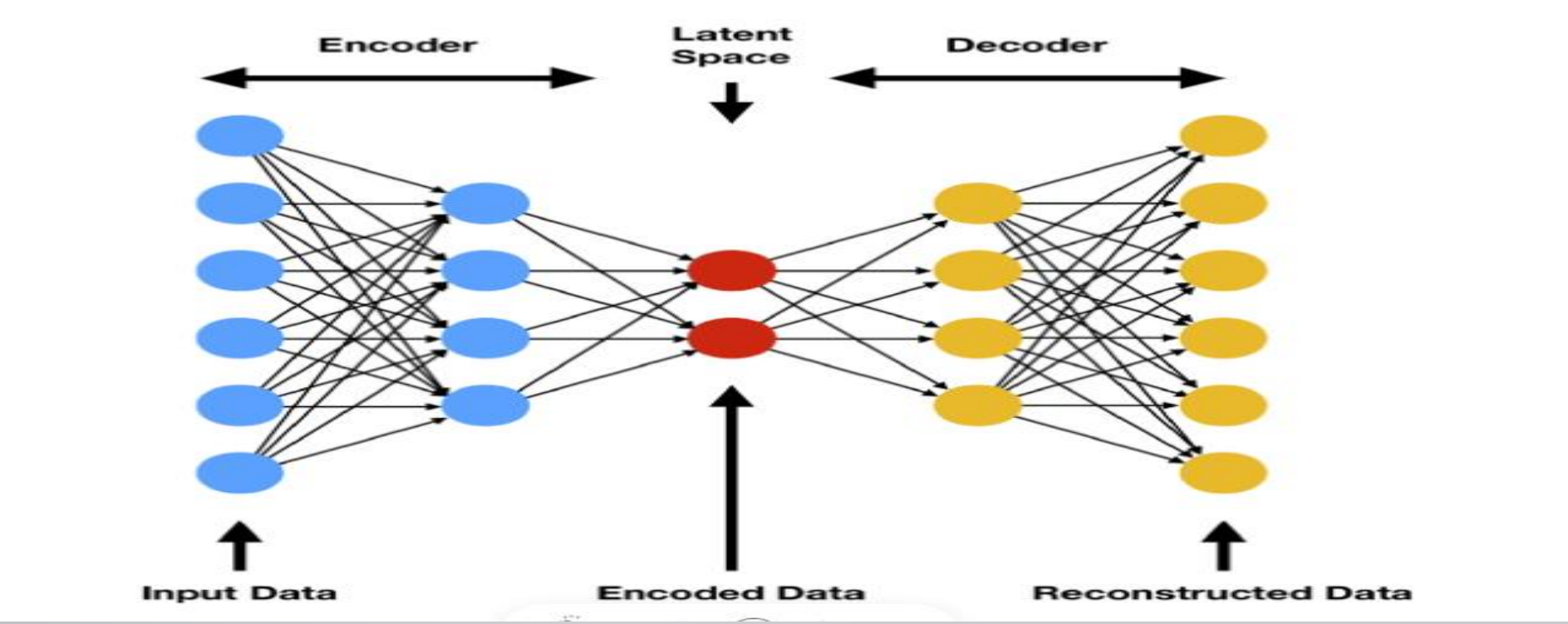
An Autoencoder consist of three layers as shown in Figure 2:

1. **Encoder**
2. **Code**
3. **Decoder**

- **Encoder:** This part of the network compresses the input into a **latent space representation**. The encoder layer **encodes** the input image as a compressed representation in a reduced dimension. The compressed image is the distorted version of the original image.
- **Code:** This part of the network represents the compressed input which is fed to the decoder.
- **Decoder:** This layer **decodes** the encoded image back to the original dimension. The decoded image is a lossy reconstruction of the original image and it is reconstructed from the latent space representation.

2.3 Architecture of Autoencoders

Figure 2: Architecture of Autoencoders (Emma,2021)



Source : microsoftbing.com

2.2 Applications of Autoencoders

- **Image Denoising: I**t is a process to reserve the details of an **image** while removing the random noise from the **image** as far as possible. Instead of feeding in the same input and output data, a noisy image is fed in and then set the target to be the original image.
- Image compression : autoencoders decompose data(images) into fairly small bits and then using that representation, reconstruct the original data as closely as it can to the original[5].
- Anomaly detection: an autoencoders is trained on non fraudulent transactions. If a fraudulent transaction is fed then the reconstruction loss will be large. A threshold can now be set to perform anomaly detection.
- **Reduction of Dimensionality**
- The autoencoders reduce the input to a reduced representation stored in the middle layer called code. By separating this layer from the model, the information from the input has been compressed, and each node can now be handled as a variable. As a result, we may determine that by deleting the decoder, an autoencoder with the coding layer as the output can be used for dimensionality reduction[6].
- **Extraction of Features**

Autoencoders' encoding segment aids in the learning of critical hidden features present in the input data, reducing the reconstruction error. A new set of unique feature combinations is formed during the encoding process[6].

2.3 why Autoencoders?

- An autoencoder can learn **non-linear transformations** with a **non-linear activation function** and multiple layers.
- It doesn't have to learn dense layers. It can use **convolutional layers** to learn which is better for video, image and series data.
- It is more efficient to learn several layers with an autoencoder rather than learn one huge transformation with PCA.
- An autoencoder provides a representation of each layer as the output.
- It can make use of **pre-trained layers** from another model to apply transfer learning to enhance the encoder/decoder.

2.4 : Types of autoencoders

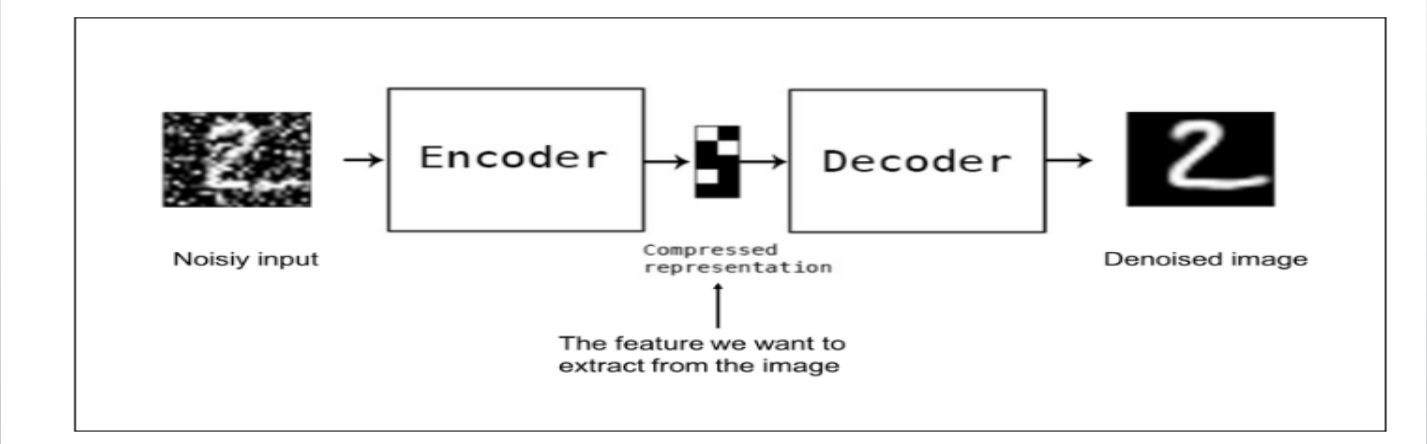
- There are 7 types of Autoencoders:

1. Denoising Autoencoder
2. Sparse Autoencoder
3. Deep Autoencoder
4. Contractive Autoencoder
5. Undercomplete Autoencoder
6. Convolutional Autoencoder
7. Variational Autoencoder

2.3.1 Denoising autoencoder

- The input seen by the autoencoder is not the raw input but a stochastically corrupted version. A denoising autoencoder is thus trained to reconstruct the original input from the noisy version.

Figure 3 : Denoising autoencoder(Sayantini,2018)



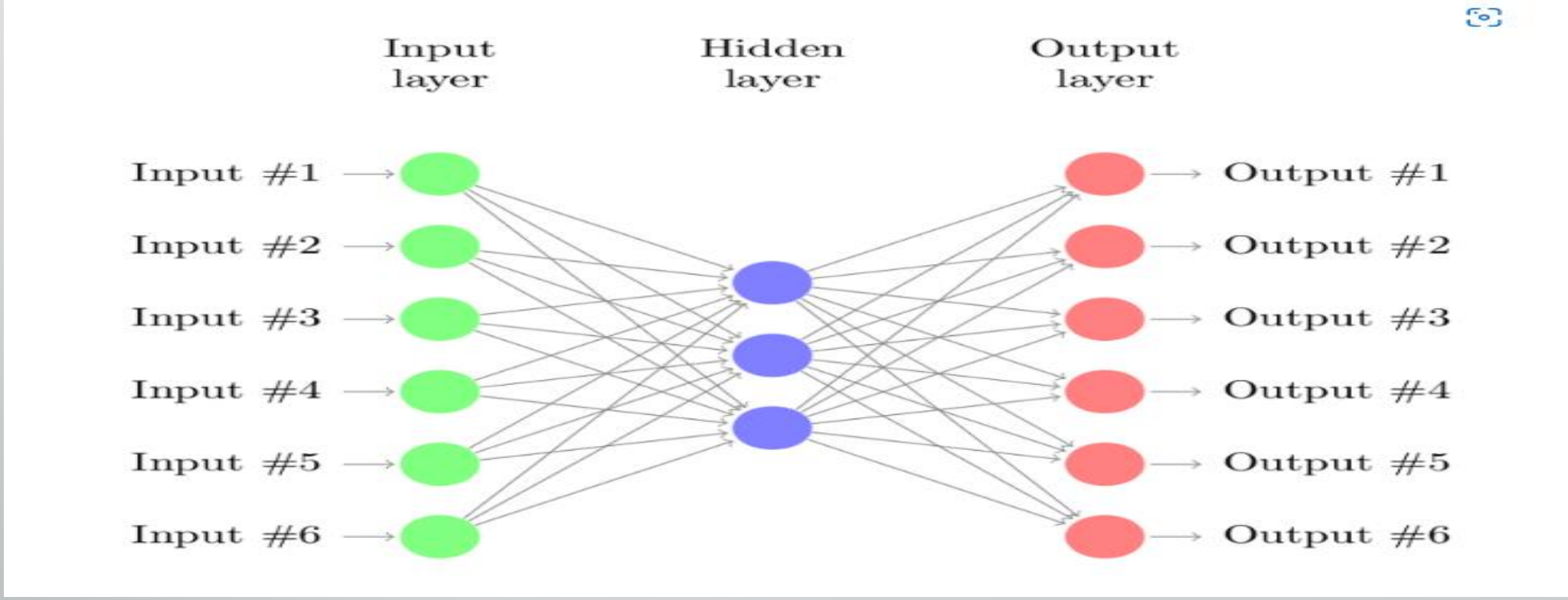
Source: https://medium.com/edureka/autoencoders-tutorial

2.3.2 Sparse autoencoders

- Sparse autoencoders have hidden nodes greater than input nodes. They can still discover important features from the data.
- A generic sparse autoencoder is visualized where the obscurity of a node corresponds with the level of activation.
- Sparsity constraint is introduced on the hidden layer. This is to prevent output layer copy input data.
- Sparsity may be obtained by additional terms in the loss function during the training process, either by comparing the probability distribution of the hidden unit activations with some low desired value, or by manually zeroing all but the strongest hidden unit activations.
- Some of the most powerful AIs in the 2010s involved sparse autoencoders stacked inside of deep neural networks.

2.3.2 Sparse autoencoders

Figure 4 : Sparse autoencoder(Sayantini,2018)

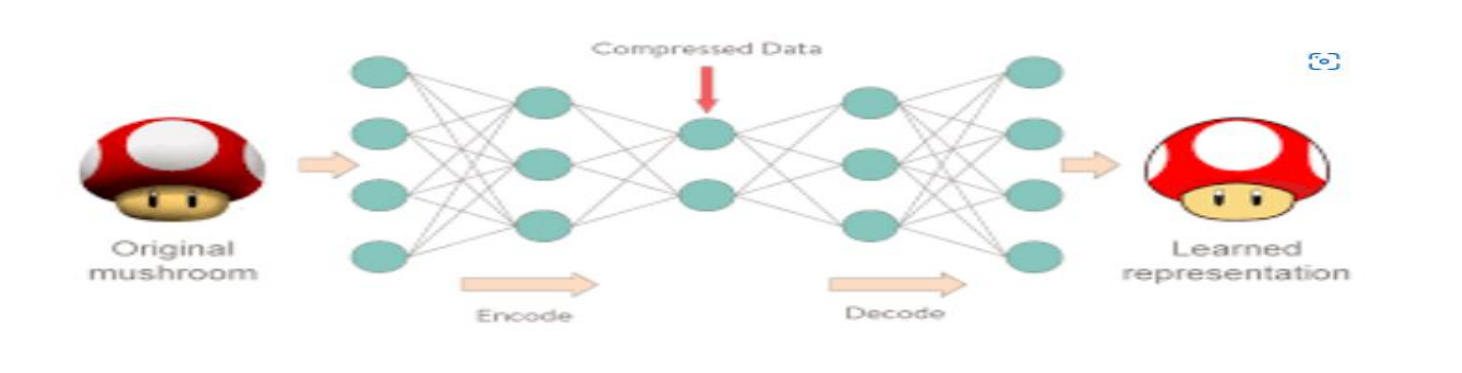


Source: https://medium.com/edureka/autoencoders-tutorial

2.3.3: Deep Autoencoders

- Deep Autoencoders consist of two identical deep belief networks, One network for encoding and another for decoding.
- Typically deep autoencoders have 4 to 5 layers for encoding and the next 4 to 5 layers for decoding. We use unsupervised layer by layer pre-training for this model.
- The layers are Restricted Boltzmann Machines which are the building blocks of deep-belief networks.
- Deep autoencoders are useful in topic modeling, or statistically modeling abstract topics that are distributed across a collection of documents. They are also capable of compressing images into 30 number vectors.

Figure 5 : Deep autoencoder(Sayantini,2018)



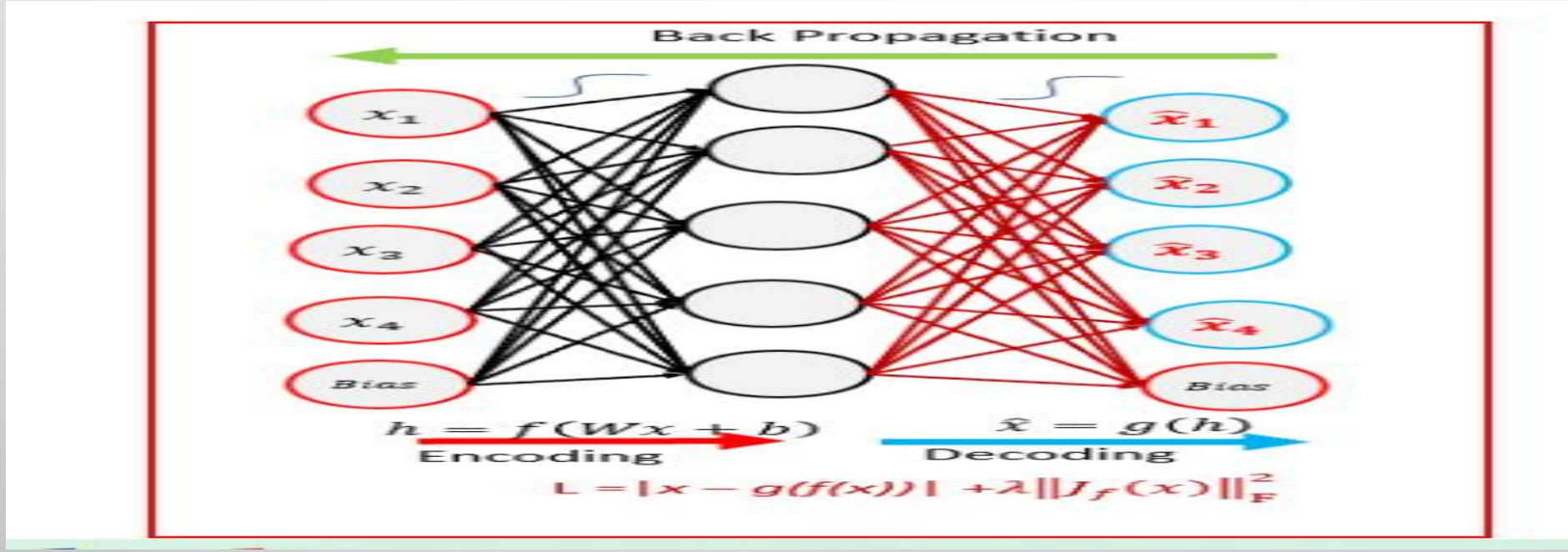
Source: https://medium.com/edureka/autoencoders-tutorial

2.3.4 Contractive autoencoders

- The objective of a contractive autoencoder is to have a robust learned representation which is less sensitive to small variation in the data.
- Robustness of the representation for the data is done by applying a penalty term to the loss function.
- Contractive autoencoder is another regularization technique just like sparse and denoising autoencoders. However, this regularizer corresponds to the Frobenius norm of the Jacobian matrix of the encoder activations with respect to the input. Frobenius norm of the Jacobian matrix for the hidden layer is calculated with respect to input and it is basically the sum of square of all elements.

2.3.4 Contractive autoencoders

Figure 6 : Contractive autoencoder(Sayantini,2018)



Source: https://medium.com/edureka/autoencoders-tutorial

2.3.5 Undercomplete autoencoder

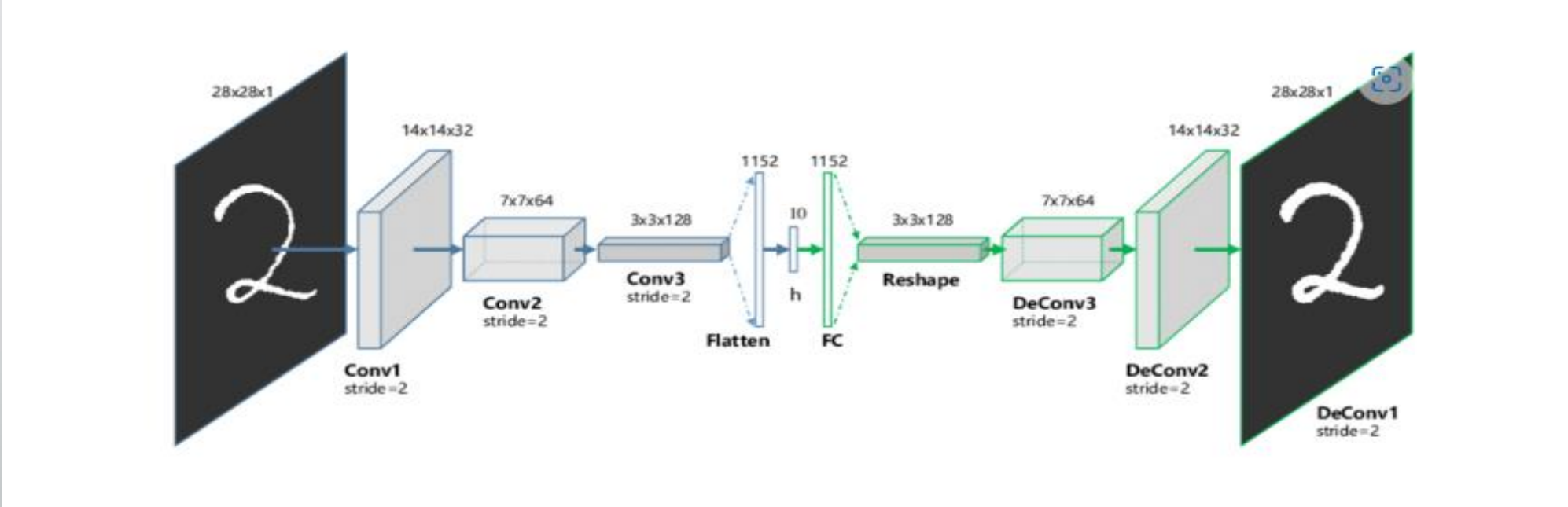
- Under complete Autoencoder is a type of Autoencoder. Its goal is to capture the important features present in the data.
- It has a small hidden layer hen compared to Input Layer.
- This Autoencoder do not need any regularization as they maximize the probability of data rather copying the input to output.
- An autoencoder whose code dimension is less than the input dimension is called undercomplete.
- Learning an undercomplete representation forces the autoencoder to capture the most salient features of the training data.

2.3.6 : Convolutional autoencoder

- Autoencoders in their traditional formulation does not take into account the fact that a signal can be seen as a sum of other signals. Convolutional Autoencoders use the convolution operator to exploit this observation. They learn to encode the input in a set of simple signals and then try to reconstruct the input from them, modify the geometry or the reflectance of the image. They are the state-of-art tools for unsupervised learning of convolutional filters. Once these filters have been learned, they can be applied to any input in order to extract features. These features, then, can be used to do any task that requires a compact representation of the input, like classification.

2.3.6 : Convolutional autoencoder

Figure 7 : Convolutional (Sayantini,2018)



Source: https://medium.com/edureka/autoencoders-tutorial

2.3.7 : Convolutional autoencoder

In this study We will look at one type of autoencoders knows as Convolutional autoencoder.

Convolutional Autoencoders(CAE) are the state of art tools for unsupervised learning of convolutional filters. Once these filters have been learned, they can be applied to any input to extract features. These features can be used to do any task that requires a compact representation of the input, like classification.

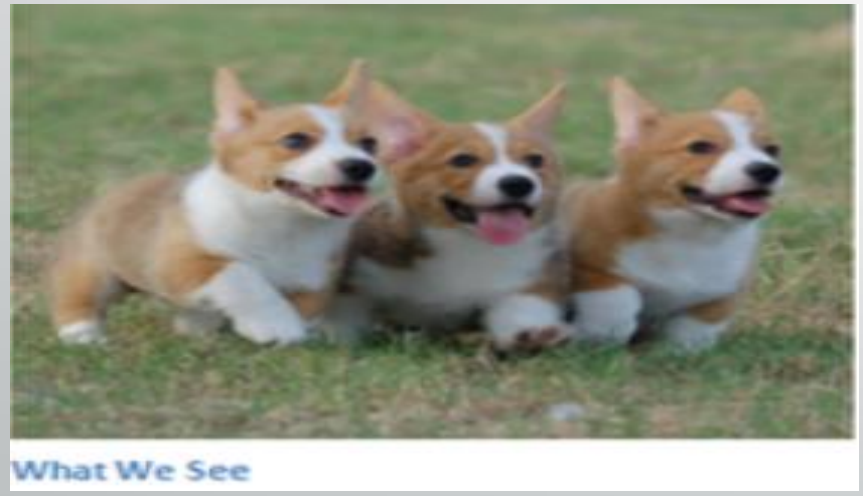
CAEs are a type of Convolutional Neural Networks (CNNs). The main difference between the common interpretation of CNN and CAE is that the formers are trained end to end to learn filters and combine features with the aim of classifying their input. The latter are trained only to learn filters able to extract features that can be used to reconstruct the input.

CAEs, due to their convolutional nature, scale well to accurate sized high dimensional images as the number of parameters required to produce an activation map is always the same, no matter what the size of the input is.

2.3.7 : Convolutional autoencoder(cont'd)

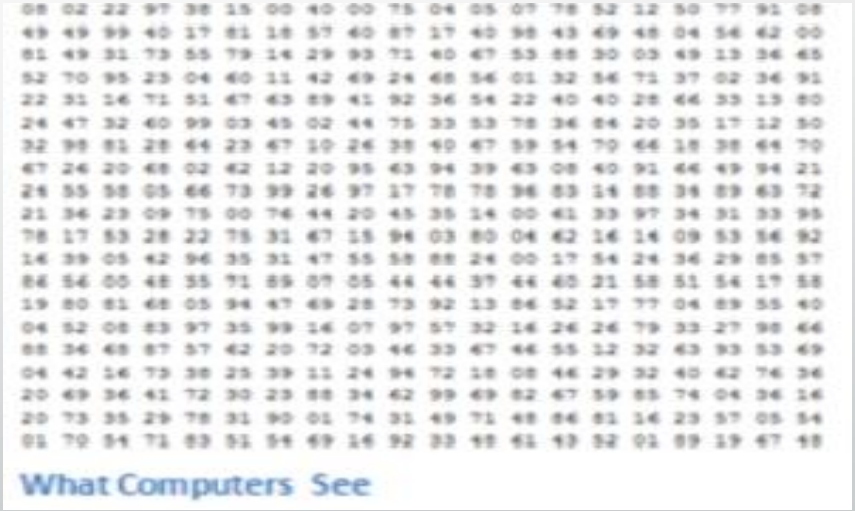
When a computer sees an image (takes an image as input), it will see an array of pixel values. Depending on the resolution and size of the image, it will see a 32 x 32 x 3 array of numbers (The 3 refers to RGB values). Just to drive home the point, let's say we have a color image in JPG form and its size is 480 x 480. The representative array will be 480 x 480 x 3. Each of these numbers is given a value from 0 to 255 which describes the pixel intensity at that point.

Figure 8 : What we see(Amir, 2019)



htpps://medium.com/machine-learning-researcher

Figure 10 : What computers see(Amir, 2019)

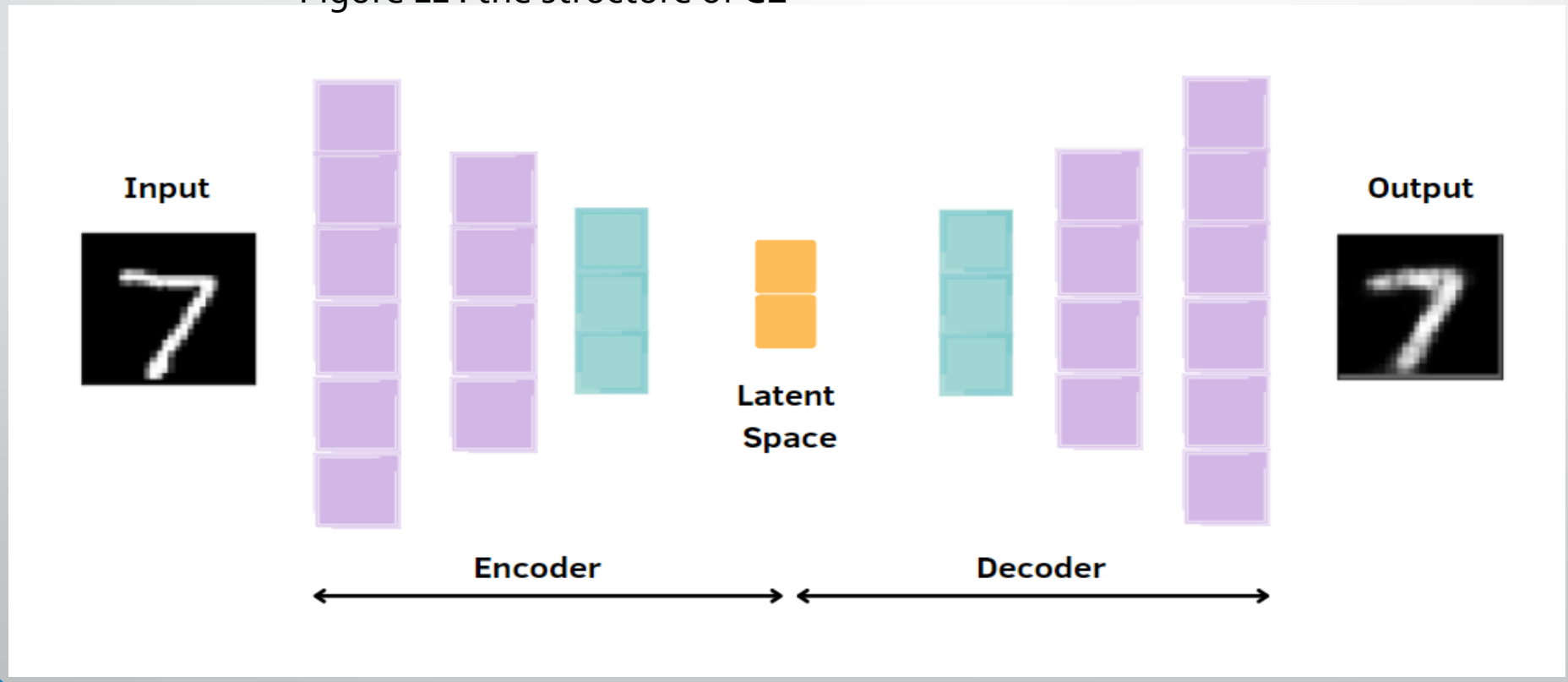


htpps://medium.com/machine-learning-researcher

2.3.7 : Convolutional autoencoder(cont'd)

The autoencoder takes the input decodes using the Encoder portion and provides an output which is similar to the input using a Decoder portion of the autoencoder as shown in Figure 11.

Figure 11 : the structure of CE



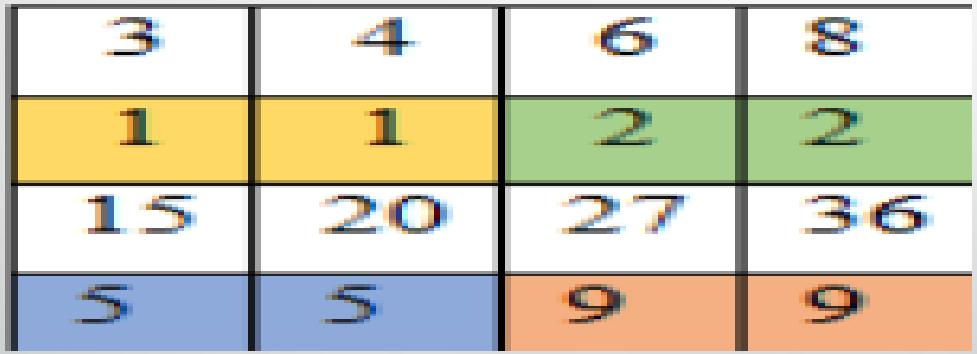
htpps://medium.com/machine-learning-researcher

2.3.7 : Convolutional autoencoder(cont'd)

1. The Encoding Portion

Suppose we have an image's convolutional layer of 4x4 pixel matrix as shown in Figure 12 with some simple sort of patterns highlighted with colors in the image given below. Each box represents a pixel and its value.

Figure 12 : convolution Layer(Amir,2019)



htpps://medium.com/machine-learning-researcher

2.3.7 : Convolutional autoencoder(cont'd)

Now we will apply the pulling process on the convolutional layer to convert the image into a compressed form with useful extracted features. We apply the 2x2 matrix with a stride length of 2 on the convolutional layer to compress the image to obtain pooling layers as shown in figure 13-16.

Figure 13 : Pooling layer1 (amir,2019)

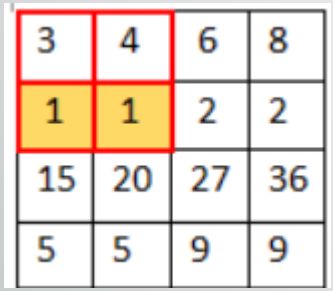


Figure 14 : Pooling layer2 (amir,2019)

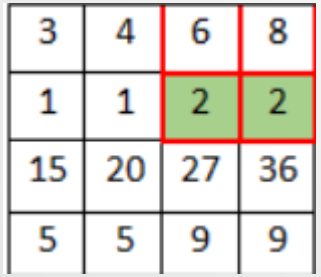


Figure 15 : Pooling layer3 (amir,2019)

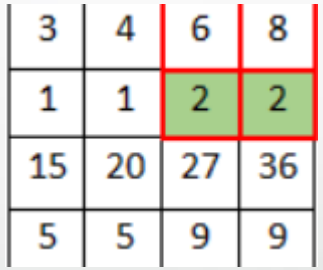
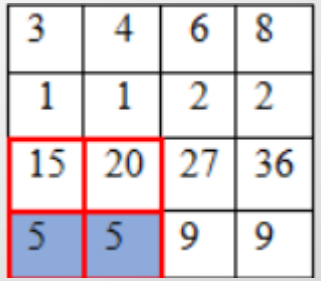


Figure 16 : Pooling layer4 (amir,2019)



We take these important and useful high-level feature values representing a specific pattern available in the convolutional layer of the image and compressed the convolutional layer into a 2x2 matrix during the pulling process from the 4x4 matrix to 2x2 matrix known as the feature map as shown in Figure 17 .

Figure 17 : Feature Map(amir,2019)

htpps://medium.com/machine-learning-researcher

2.3.7 : Convolutional autoencoder(cont'd)

Our feature detector, also called filter or kernel is

Figure 18 : Feature Detector(amir,2019)

3	4
1	1

2. Decoding Portion

In the decoding portion, we will unspool the 2x2 compressed form matrix extracted from the convolutional layer to reconstruct the image. To do this, take the kernel and multiply every single location with the pixel value of the 2x2 compressed matrix that we are deconvolving. Where there are overlaps, we sum the values.

Step 1:

We take the filter and the output feature map as shown in figure and figure and multiply them . After multiplication we get

Figure 19 : Feature Map(amir,2019)

3x1	4x1
1x1	1x1

htpps://medium.com/machine-learning-researcher

2.3.7 : Convolutional autoencoder(cont'd)

We use switches represented by number o to remember the original location of each pixel

Our first unpooled matrix;

Figure 20 : First unpooled matrix(amir,2019)

3x1	4x1	0	0
1x1	1x1	0	0
0	0	0	0
0	0	0	0

Step 2: The feature detector shown in figure 18 is multiplied with the matrix on figure 21 and the resultant matrix is shown in figure 22.

Figure 21 : Feature Map(amir,2019)

1	2
5	9

Our second unpooled matrix

Figure 22 : Second unpooled matrix(amir,2019)

0	0	3x2	4x2
0	0	2x1	2x1
0	0	0	0
0	0	0	0

2.3.7 : Convolutional autoencoder(cont'd

Step 3: the same process is repeated and the third and fourth unpooled matrices are shown in figure 23

Figure 23 : Third unpooled matrix(amir,2019)

0	0	0	0
0	0	0	0
3x5	4x5	0	0
1x5	1x5	0	0

Step 4:

the same process is repeated and the third and fourth unpooled matrices are shown in figure 24

Figure 24 : Fourth unpooled matrix(amir,2019)

0	0	0	0
0	0	0	0
0	0	3x9	4x9
0	0	1x9	1x9

htpps://medium.com/machine-learning-researcher

2.3.7 : Convolutional autoencoder(cont'd

As a result of overlaps, we sum all the unpooled matrix to reconstruct the final image;

Figure 25 : sum of unpooled matrices(amir,2019)

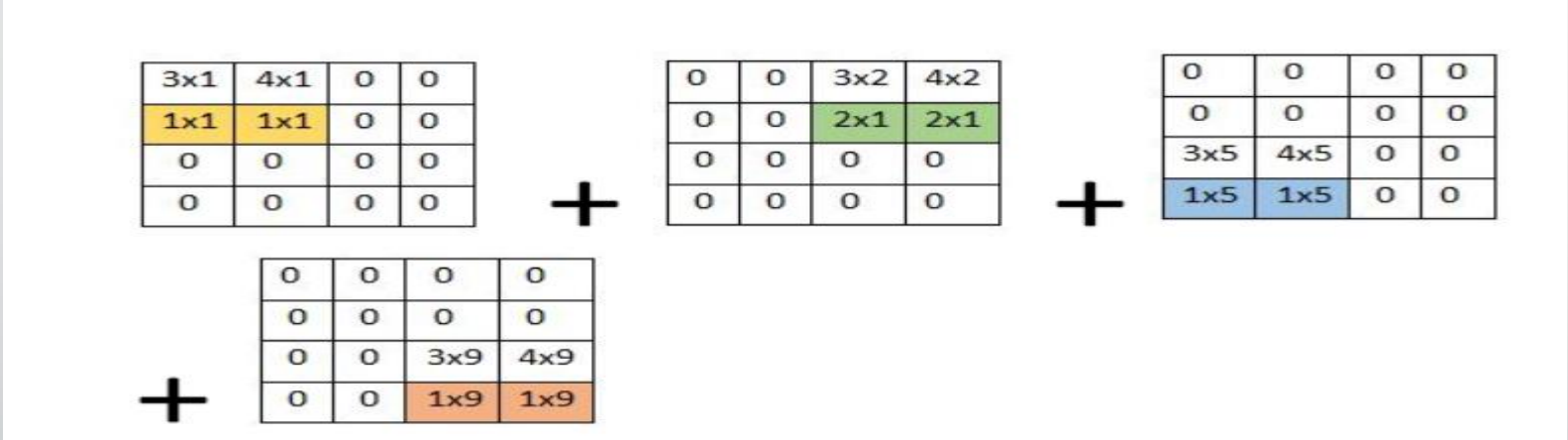


Figure 24 : Fourth unpooled matrix(amir,2019)

So our final output matrix is

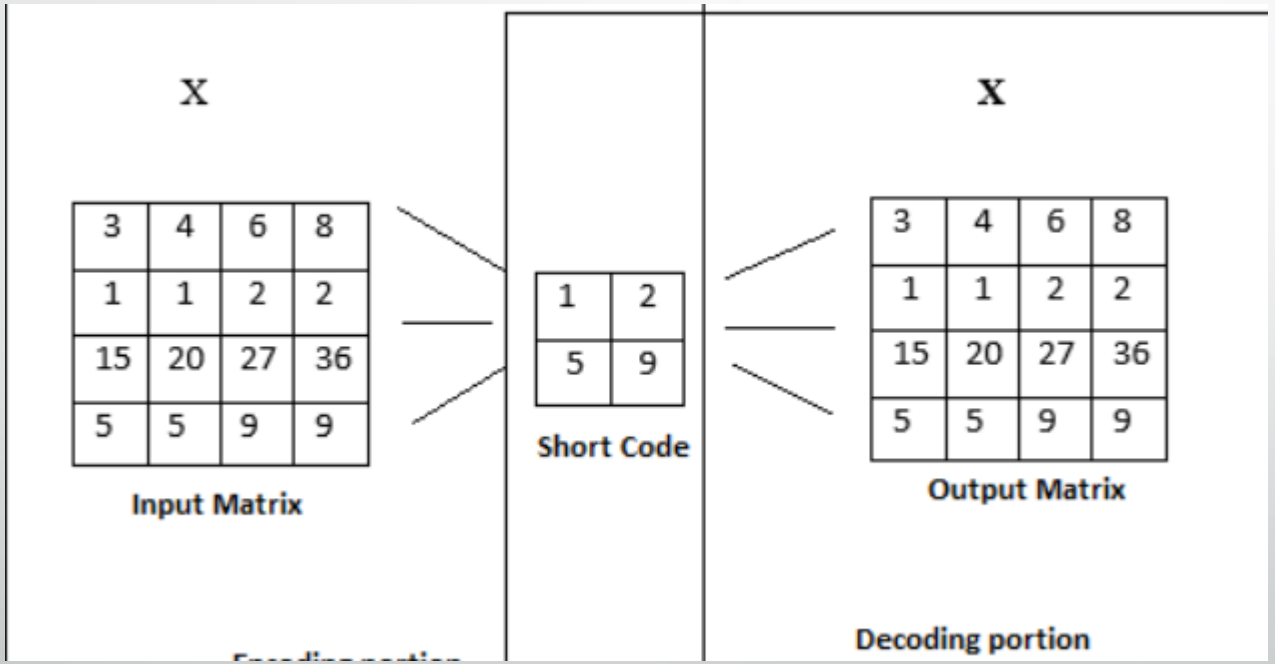
Figure 26 : Final output(amir,2019)

3	4	6	8
1	1	2	2
15	20	27	36
5	5	9	9

2.3.7 : Convolutional autoencoder(cont'd

So our final output matrix is exactly the same as the input matrix

Figure 27 : comparison of input and output(amir,2019)



htpps://medium.com/machine-learning-researcher

Part 3

PCA vs Autoencoders

3.1 : PCA vs Autoencoders

- **PCA vs Autoencoder**
- Although PCA is fundamentally a linear transformation, auto-encoders may describe complicated non-linear processes.
- Because PCA features are projections onto the orthogonal basis, they are completely linearly uncorrelated. However, since autoencoder features are only trained for correct reconstruction, they may have correlations.
- PCA is quicker and less expensive to compute than autoencoders.
- PCA is quite similar to a single layered autoencoder with a linear activation function.
- Because of the large number of parameters, the autoencoder is prone to overfitting. (However, regularization and proper planning might help to prevent this).

3.2 How to select the models

- Aside from processing computational resources, the choice of approach is influenced by the features of the feature space itself. If the features have a non-linear connection, the autoencoder may compress the data more efficiently into a low-dimensional latent space by utilizing its capacity to represent complicated non-linear processes.
- Researchers created a two-dimensional feature space with linear and non-linear relationships between them (x and y are two features) (with some added noise). After projecting the input into latent space, we can compare the capabilities of autoencoders and PCA to properly reconstruct the input. PCA is a linear transformation with a well-defined inverse transform, and the reconstructed input comes from the autoencoder's decoder output. For both PCA and autoencoders, we employ a one-dimensional latent space.
- Autoencoder latent space may be employed for more accurate reconstruction if there is a nonlinear connection (or curvature) in the feature space. PCA, on the other hand, only keeps the projection onto the first principal component and discards any information that is perpendicular to it.

Summary

- In machine learning projects we often run into curse of dimensionality problem where the number of records of data are not a substantial factor of the number of features. This often leads to a problems since it means training a lot of parameters using a scarce data set, which can easily lead to overfitting and poor generalization.
- For dimensionality reduction to be effective, there needs to be underlying low dimensional structure in the feature space. I.e the features should have some relationship with each other.
- If there is non-linearity or curvature in low dim structure than autoencoders can encode more information using less dimensions. So they are a better dimensionality reduction technique in these scenarios.
- Autoencoders are neural networks that can be used to reduce the data into a low dimensional latent space by stacking multiple non-linear transformations(layers).
- PCA essentially learns a linear transformation that projects the data into another space, where vectors of projections are defined by variance of the data

References

1. Ben-David, S. (2014). *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press.
2. Marsland, S. (2015). *Machine Learning, An algorithmic Perspective* .Chapman and Hall/CRC.
3. What is Principal component Analysis. Retrieved April , 1,2021. From https://builtin//data-science //towardsdatascience.com/A step by step explanation of Principal component analysis
4. Autoencoders: What They Are & When to Use Them. Retrieved December, 21,2021. From https://rapidminer.com/blog/autoencoders.
5. Autoencoders for image reconstruction in Python and Keras. Retrieved 2022. From http ://stackabuse.com/autoenders-for-image-reconstruction-in-python-and-Keras.
6. What are the applications of autoencoders?(nd). From htpps://tutorialspoint.com/what are the applications of autoencoders.
7. Understanding Principal component analysis (PCA) step by step. From https://medium.com/analytics-vidhya/understanding-principle-component-analysis-pca-step-by-step-e7a4bb4031d9