DL 3 - Deep Learning Module 3 Semester 7

Computer Science and Engineering (AIML) (University of Mumbai)

3

Module 3

Autoencoders : Unsupervised Learning

Syllabus

3.1 Introduction, Linear Autoencoder, Undercomplete Autoencoder, Overcomplete Autoencoders, Regularization in Autoencoders

3.2 Denoising Autoencoders, Sparse Autoencoders, Contractive Autoencoders

3.3 Application of Autoencoders: Image Compression

3.1 Introduction

- Machine Learning algorithms are often classified into two categories: Supervised Learning and Unsupervised Learning. The difference between these categories largely lies in the type of data that we deal with: Supervised Learning deals with labelled data, while Unsupervised Learning deals with unlabelled data.
- An autoencoder is a class of neural network that uses unsupervised learning, and applies backpropagation, setting the target values to be equal to the inputs.
- An autoencoder is a neural network that is trained to learn the identity function and attempts to copy its input to output. While doing this, the network will be able to learn useful and interesting properties in the data and learns a representation of the data, which might be helpful in Classification, image recovery, or any other application where good feature set is needed.
- An autoencoder learns this data encodings in an unsupervised manner.
- The aim of an autoencoder is to learn a lower-dimensional representation (encoding) for higher-dimensional data
- It is typically used for dimensionality reduction, by training the network to capture the most important parts of the input image.

Architecture of Autoencoder

An autoencoder is composed of three parts : An encoder, a bottleneck (or code) , and a decoder.

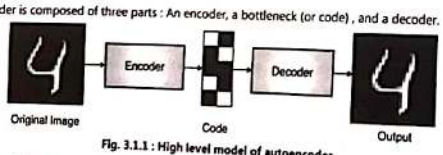

Fig. 3.1.1 : High level model of autoencoder

1. **Encoder :** A module that compresses the input data into an encoded representation that is typically several orders of magnitude smaller than the input data.

2. **Bottleneck :** A module that contains the compressed knowledge representations and is therefore the most important part of the network.
3. **Decoder :** A module that helps the network "decompress" the knowledge representations and reconstructs the data back from its encoded form. The output is then compared with ground truth.

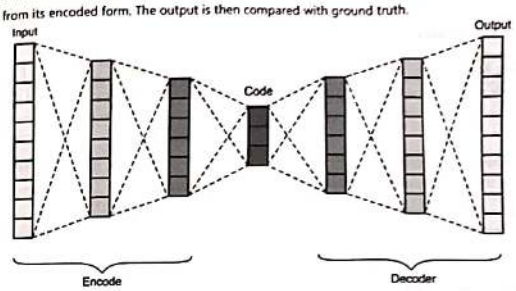

Fig. 3.1.2 : Architectural components of autoencoder

The encoder compresses the input and the decoder attempts to recreate the input from the compressed version provided by the encoder. Autoencoders compress the input into a lower-dimensional code (Encoded representation of data) and then reconstruct the output from this representation. The code is a compact "summary" or "compression" of the input, also called the latent-space representation. After training, the encoder model is saved and the decoder is discarded

Why autoendcoders ?

Autoencoders are used to

1. map high-dimensional data to lower dimensions.
2. to compress data or images
3. learn abstract features present in data in an unsupervised manner. These features can be then used for other supervised tasks such as classification.

How to train autoencoders ?

For training autoencoders we need to set 4 hyperparameters before training an autoencoder:

1. **Code size :** The code size or the size of the bottleneck is the most important hyperparameter used to tune the autoencoder. The bottleneck size decides how much the data has to be compressed. This can also act as a regularization term.
2. **Number of layers :** Like all neural networks, an important hyperparameter to tune autoencoders is the depth of the encoder and the decoder. While a higher depth increases model complexity, a lower depth is faster to process.
3. **Number of nodes per layer :** The number of nodes per layer defines the weights we use per layer. Typically, the number of nodes decreases with each subsequent layer in the autoencoder as the input to each of these layers becomes smaller across the layers.

4. **Reconstruction Loss :** The loss function we use to train the autoencoder is highly dependent on the type of input and output we want the autoencoder to adapt to. If we are working with image data, the most popular loss functions for reconstruction are MSE Loss and L1 Loss. In case the inputs and outputs are within the range [0,1], we can also make use of Binary Cross Entropy as the reconstruction loss.

3.1.1 Important Properties of Autoencoders

1. **Data-specific :** Autoencoders are only able to meaningfully compress data similar to what they have been trained on. Since they learn features specific to the given training data, they are different from a standard data compression algorithm like gzip. So we can't expect an autoencoder trained on handwritten digits to compress landscape photos.
2. **Lossy :** The output of the autoencoder will not be exactly the same as the input, it will be a close but degraded representation. If you want lossless compression, they are not the way to go.
3. **Unsupervised :** Autoencoders are considered an *unsupervised* learning technique since they don't need explicit labels to train on. But to be more precise they are *self-supervised* because they generate their own labels from the training data.

3.2 Linear Autoencoders

- An **autoencoder** is a type of Feedforward used to learn efficient data coding in an unsupervised manner.
- The aim of an autoencoder is to learn a representation (encoding) for a set of data, typically for dimensionality reduction, by training the network to ignore signal "noise".

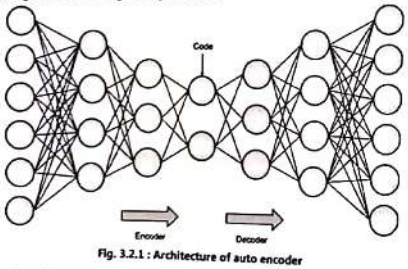

Fig. 3.2.1 : Architecture of auto encoder

- Autoencoders consists of two main parts: encoder and decoder (Fig 3.2.1).
- They work by automatically encoding data based on input values, then performing an activation function, and finally decoding the data for output.
- A bottleneck layer imposed on the input features, compresses input into fewer categories. Thus, if some inherent structure exists within the data, the autoencoder model will identify and leverage it to get the output.
- A linear autoencoder uses zero or more linear activation function in its layers.

- The components of autoencoders: encoder , code and decoder have been explained below :
  - **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.
- A linear autoencoders is the simplest auto encoder with relatively very simple structure say single layer encoder and single layer decoder.
- Let us consider an autoencoder with just one hidden layer (i.e. the bottleneck layer).
- We can prove that a single layer auto encoder with linear transfer function is nearly equivalent to PCA (Principal Component Analysis).
- Here x is the original data, z is the reduced data and X^ is the reconstructed data from the reduced representation.
- Then we can write PCA as:

$$z = B^T x$$

$$x^{\wedge} = Bz$$

- Now consider an autoencoder architecture shown in Fig. 3.2.2.

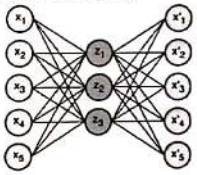

Fig. 3.2.2 : Auto encoder with one hidden layer

Which is basically x->z->x^.

- Since we have assumed that the activation functions are linear transfer functions $\sigma(x)=x$, then we can write the autoencoder as :

$$x^{\wedge} = W_1 W_2 x$$

- Where W_1 and W_2 are the weights of the first and second layer.
- Now if we set $W_1=B$ and $W_2=B^T$ we have :

$$x^{\wedge} = W_1(W_2 x)$$

$$x^{\wedge} = W_1 z$$

$$x^{\wedge} = Bz$$

which is the same solution that we had for PCA.

- Note that the equivalence is valid only for autoencoders that have a bottleneck layer smaller than the input layer.

3.2.1 Difference between PCA and Auto Encoders

1. PCA is essentially a linear transformation but Auto-encoders are capable of modelling complex nonlinear functions
2. PCA features are totally linearly uncorrelated with each other since features are projections onto the orthogonal basis. But autoencoder features might have correlations since they are just trained for accurate reconstruction.
3. PCA is faster and computationally cheaper than autoencoders
4. A single layered autoencoder with a linear activation function is very similar to PCA.
5. Autoencoder is prone to overfitting due to high number of parameters. (though regularization and careful design can avoid this)

3.3 Under Complete Auto Encoder

- In under complete autoencoder, the dimensionality of hidden layers is smaller than the input layer.
- In these types of encoders we constrain the number of nodes present in the hidden layer(s) of the network, thus limiting the amount of information that can flow through the network.
- So, basically networks learn to compress high dimensional input into lower dimensional representation forcing the network to capture only the most important features.
- An under complete autoencoder has no explicit regularization term - we simply train the model according to the reconstruction loss.
- Thus, the only way to ensure that the model isn't memorizing the input data (overfitting) is the ensure that we sufficiently restrict the number of nodes in the hidden layer(s).
- Here by penalizing the network according to the reconstruction error, the model can learn the most important attributes of the input data and how to best reconstruct the original input from an "encoded" state. Ideally, this encoding will learn and describe latent attributes of the input data.
- The hidden layer is under complete means it is smaller than the input layer.
- The hidden layer compresses the input.
- Will compress well only on the training distribution.

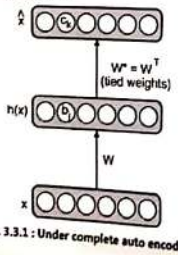

Fig. 3.3.1 : Under complete auto encoders

3.4 Overcomplete Autoencoder

- In overcomplete autoencoder, hidden layer has more neurons than the input.
- This type of network architecture gives the possibility of learning greater number of features, but on the other hand, it has potential to learn the identity function (meaning that the output equals the input) and become useless.
- One way of handling this threat is implementing regularization.
- Regularized autoencoder, uses a loss function that encourages the network to prevent from just copy its input to its output.
- The hidden layer is over complete means it is larger than the input layer.
- No compression in hidden layer
- Each hidden unit could copy a different input component.
- No guarantee that the hidden unit will extract meaningful structure

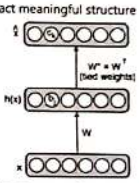

Fig. 3.4.1 : Over complete auto encoders

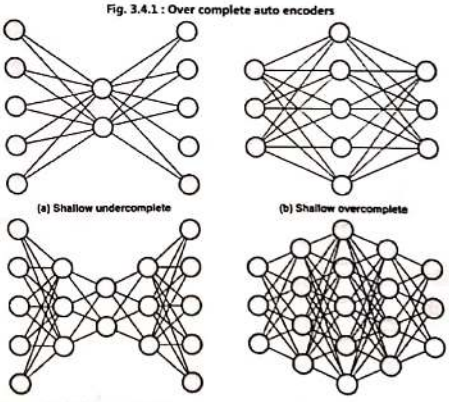

Fig. 3.4.2 : Different types of under complete and over complete auto encoders

3.5 Regularization in Auto Encoder

- Rather than limiting the model capacity by keeping the encoder and decoder shallow and the code size small, regularized autoencoders use a loss function that encourages the model to have other properties besides the ability to copy its input to its output.
- In practice, we usually find two types of regularized autoencoder : the sparse autoencoder and the denoising autoencoder.

3.5.1 Denoising Auto Encoders

- Autoencoders are Neural Networks that are commonly used for feature selection and extraction. However, when there are more nodes in the hidden layer than there are inputs, the Network is risking to learn the so-called "Identity Function", also called the "Null Function", meaning that the output equals the input, marking the Autoencoder useless.
- Denoising Autoencoders solve this problem by corrupting the data on purpose by randomly turning some of the input values to zero. In general, the percentage of input nodes which are being set to zero is about 30% to 50%.
- The denoising autoencoders...
  1. The hidden layers of the autoencoder learn more robust filters
  2. Reduce the risk of overfitting in the autoencoder
  3. Prevent the autoencoder from learning a simple identify function (to solve the problem of overcomplete AE)
- Noise was stochastically (i.e., randomly) added to the input data, and then the autoencoder was trained to recover the original, nonperturbed signal.
- One of the common applications of such auto encoders is to pre-process an image to improve the accuracy of an optical character recognition (OCR) algorithm.
- We know that just a little bit of the *wrong type of noise* (ex., printer ink smudges, poor image quality during the scan, etc.) can *dramatically* hurt the performance of your OCR method.
- Using denoising autoencoders, we can automatically pre-process the image, improve the quality, and therefore increase the accuracy of the downstream OCR algorithm.

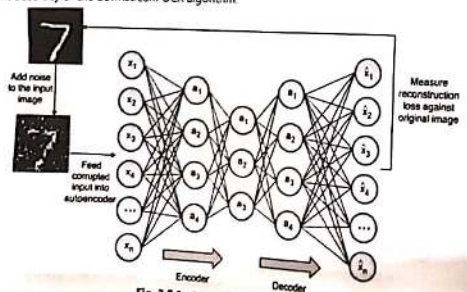

Fig. 3.5.1 : Denoising autoencoders

3.5.2 Sparse Auto Encoders

- The undercomplete autoencoders are regulated and fine-tuned by regulating the size of the bottleneck.
- The sparse autoencoder is regulated by changing the number of nodes at each hidden layer.
- This is done by activating fewer neurons at the same time, creating an information bottleneck similar to that of Under complete AE.
- Since it is not possible to design a neural network that has a flexible number of nodes at its hidden layers, sparse autoencoders work by penalizing the activation of some neurons in hidden layers.
- In other words, the loss function has a term that calculates the number of neurons that have been activated and provides a penalty that is directly proportional to that.
- This penalty, called the sparsity function, prevents the neural network from activating more neurons and serves as a regularize.

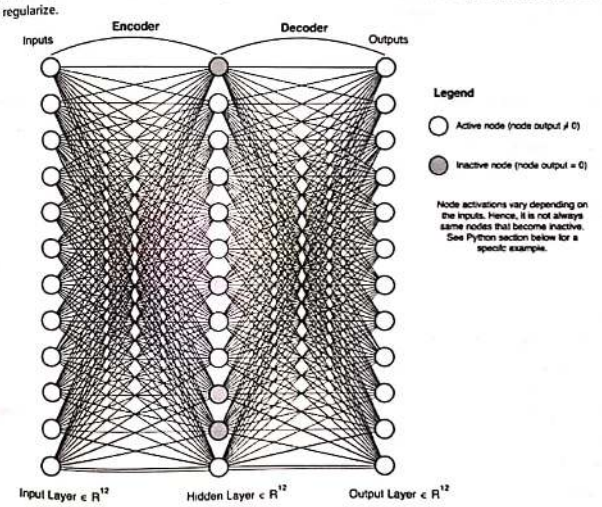

Fig. 3.5.2 : Sparse auto encoders

- We can think of each neuron in hidden layer as filter which will be activated for certain input configuration.
- We would construct our loss function by penalizing activations of hidden layers so that only a few nodes are encouraged to activate when a single sample is fed into the network.
- There are actually two different ways to construct our sparsity penalty: L1 regularization and KL-divergence.

- Most commonly used is L1 regularization.
- L1 regularization adds "absolute value of magnitude" of coefficients as penalty term while L2 regularization adds "squared magnitude" of coefficient as a penalty term.
- Although L1 and L2 can both be used as regularization term, the key difference between them is that L1 regularization tends to shrink the penalty coefficient to zero while L2 regularization would move coefficients towards zero but they will never reach. Thus L1 regularization is often used as a method of feature extraction.

$$\text{Obj} = L(x, \hat{x}) + \text{regularization} + \lambda \sum_i |a_i^{(h)}|$$

- Except for the first two terms, we add the third term which penalizes the absolute value of the vector of activations a in layer h for sample i. Then we use a hyperparameter to control its effect on the whole loss function. And in this way, we do build a sparse autoencoder.

3.5.3 Contractive Auto Encoders

- The goal of Contractive Autoencoder is to reduce the representation's sensitivity towards the training input data.
- In other words we strive to make the autoencoders robust of small changes in the training dataset.
- In order to achieve this, we must add a regularizer or penalty term to the cost function that the autoencoder is trying to minimize.
- It adds an extra term in the loss function of auto encoder , it is given by

$$\|J_h(x)\|_F^2 = \sum_{ij}\left(\frac{\partial h_j(x)}{\partial x_i}\right)^2$$

- The above penalty term is the Frobinious Norm of the encoder, the Frobinious norm is just a generalization of Euclidean norm.
- In other words, this sensitivity penalization term is the "sum of squares of all partial derivatives of the extracted features with respect to input dimensions".
- Mathematics of contractive auto encoders
- We need to understand the Frobenius norm of the Jacobian matrix before we dive into mathematics.
- The Frobenius norm, also called the Euclidean norm, is matrix norm of an mxn matrix A defined as the "square root of the sum of the absolute squares of its elements".
- The Jacobian matrix is the matrix of "all first-order partial derivatives of a vector-valued function. So when the matrix is a square matrix, both the matrix and its determinant are referred to as the Jacobian.
- Combining these two definitions gives us the meaning of Frobenius norm of the Jacobian matrix.
- The loss function is:

$$L = |X - g(f(x))| + \lambda \|J_f(x)\|_F^2$$

$$\|J_f(x)\|_F^2 = \sum_{ij}\left(\frac{\partial h_j(x)}{\partial x_i}\right)^2$$

- where the penalty term, $\lambda(J(x))^2$, is the squared Frobenius norm of the Jacobian matrix of partial derivatives associated with the encoder function and is defined in the second line.

3.6 Application of Autoencoders : Image Compression

- Digitizing the image consumes more bandwidth which means more cost for the transmission of the data. So, for the efficient use of bandwidth the images are compressed before transmission over the internet.
- We discuss the image compression using Convolutional Auto encoder.
- In order to explain how image compression can be achieved using auto encoders , we use MNIST image dataset.
- The dataset consists of various types of images, including digits, fashion and etc.
- We would be dealing with grayscale images available in the dataset. The dataset contains following train-test split.

No. of training dataset images = 60,000
No. of testing dataset images = 10,000
No. of classes/labels = 10 (0–9)
Dimension of each image = 28 X 28

- Following Fig. 3.6.1 shows the high-level architecture of the Image compression model using convolution autoencoders.

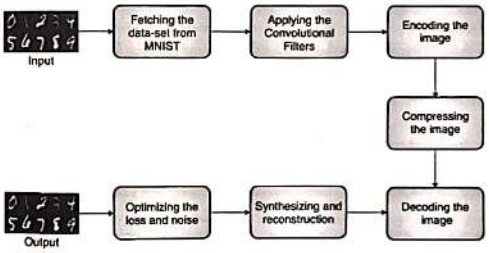

Fig. 3.6.1 : Image compression using Convolution Autoencoders

- A convolutional autoencoder with 20 layers with different filter sizes can be used.
- Unlike traditional auto encoders where we need to flatten the image and feed it as an input to autoencoder, convolutional autoencoder directly takes the whole image matrix as an input. This maintains the data spatial relationship.
- This convolutional layer is responsible for the **image compression**. It has the potential of learning the non-linear transformation through the multilayers and the linear activation function.
- Different nodes are used in each layer. In the encoder, the number of nodes decreases from **128 to 1** and in decoder increases from **1 to 128**.
- Both the encoder and the decoder consist of 10 layers each. Convolutional layers are responsible for successful compression of an image with high level of accuracy.

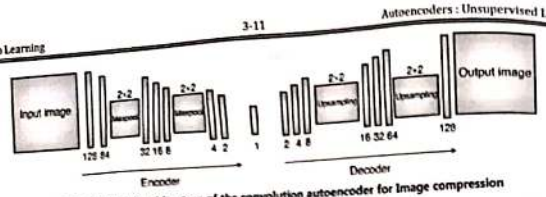

Fig. 3.6.2 : Detailed architecture of the convolution autoencoder for image compression

- The autoencoder layer starts with an input layer, whose main function is to take the input image into the further layers of the autoencoder
- In the convolution layers, the 3x3 filter having the same padding and ReLU activation function can be used.
- Following the input layer are two convolutional layers whose main function is to compress the image.
- Following the two convolutional layers there is a maxpool layer. Max pool layers are also used in the encoder part to select the more robust features from the input image or dataset. In the maxpooling layer, 2x2 size filters are used.
- After the Maxpool layer there are three convolutional layers, which are again responsible for performing the convolution function.
- Following the maxpool layer there are two convolutional layer and one code layer.
- This code layer is basically the bottleneck consisting of the lowest possible compressed data with the minimum data which is 7x7x1.
- Now decoder module starts. After the code layer there are three convolutional layers, responsible for slowly restoring the compressed image from the code layer.
- Upsampling layer (opposite of Maxpooling operation) is also included in this model in the decoder part to increase the dimension of the image from the encoded representation having filter size 2x2.
- The upsampling layer is further followed by three convolutional layer and then again by one upsampling layer. After the upsampling layer there is one final convolutional layer, followed by the final output layer that shows the output function. For the output layer, tanh activation function can be used.

3.7 Application of Autoencoders: Image Denoising (Additional)

- Image Denoising is the process of removing noise from the images.
- The noise present in the images may be caused by various intrinsic or extrinsic conditions which are practically hard to deal with.
- The problem of Image Denoising is a very fundamental challenge in the domain of Image processing and Computer vision. Therefore, it plays an important role in a wide variety of domains where getting the original image is really important for robust performance.
- While solving the denoising problem, our goal is to make a model that is capable of performing noise removal on images. To be able to do this, we need to add random noise to the existing images. So, we design one models that takes the original images as input and we get the noisy images as output. Autoencoder will learn the relationship between a clean image and a noisy image and learn how to clean a noisy image.

- We explain Image denoising using MNIST image dataset.
- The dataset consists of images of handwritten digits in the form of a greyscale. It also includes labels for each image, telling us which digit it is (i.e., output for each image). It means we have a labelled data in our hands to work with. Each image in the MNIST dataset is 28 pixels by 28 pixels.

Fig. 3.7.1 : Original clean images from MNIST

- We split the dataset into 2 parts:
- **Training Dataset:** 60,000 data points belong to the training dataset, and
- **Testing Dataset:** 10,000 data points belongs to the test dataset.

Fig. 3.7.2 : Noisy images

Encoder-Decoder Network

It will have an input layer of 784 neurons since we have an image size of 784 due to 28 by 28 pixels present.

```
#create model
model = Sequential()
# Encoder
model.add(Dense(500, input_dim=num_pixels, activation='relu'))
model.add(Dense(300, activation='relu'))
# Code
model.add(Dense(100, activation='relu'))
# Decoder
model.add(Dense(300, activation='relu'))
model.add(Dense(500, activation='relu'))
model.add(Dense(784, activation='sigmoid'))
```

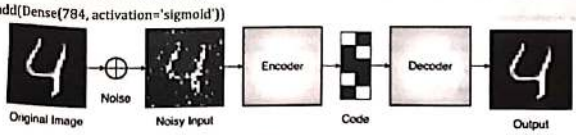

Fig. 3.7.3

3.8 Other Applications of Autoencoders

1. **Dimension reductionality**
   - The autoencoders reduce the input to a reduced representation stored in the middle layer called code.

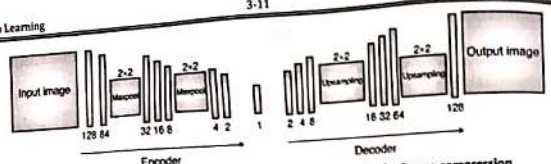

Fig. 3.6.2 : Detailed architecture of the convolution autoencoder for Image compression

- The autoencoder layer starts with an input layer, whose main function is to take the input image into the furthe layers of the autoencoder.
- In the convolution layers, the 3x3 filter having the same padding and ReLU activation function can be used.
- Following the input layer are two convolutional layers whose main function is to compress the image.
- Following the two convolutional layers there is a maxpool layer. Max pool layers are also used in the encoder pa to select the more robust features from the input image or dataset. In the maxpooling layer, 2x2 size filters ar used.
- After the Maxpool layer there are three convolutional layers, which are again responsible for performing th convolution function.
- Following the maxpool layer there are two convolutional layer and one code layer.
- This code layer is basically the bottleneck consisting of the lowest possible compressed data with the minimun data which is 7x7x1.
- Now decoder module starts. After the code layer there are three convolutional layers, responsible for slowl restoring the compressed image from the code layer.
- Upsampling layer (opposite of Maxpooling operation) is also included in this model in the decoder part to increas the dimension of the image from the encoded representation having filter size 2x2.
- The upsampling layer is further followed by three convolutional layer and then again by one upsampling laye After the upsampling layer there is one final convolutional layer, followed by the final output layer that shows th output function. For the output layer, tanh activation function can be used.

3.7 Application of Autoencoders: Image Denoising (Additional)

- Image Denoising is the process of removing noise from the Images.
- The noise present in the images may be caused by various intrinsic or extrinsic conditions which are practically hard to deal with.
- The problem of Image Denoising is a very fundamental challenge in the domain of Image processing and Computer vision. Therefore, it plays an important role in a wide variety of domains where getting the original image is really important for robust performance.
- While solving the denoising problem, our goal is to make a model that is capable of performing noise removal on images. To be able to do this, we need to add random noise to the existing images. So, we design one models that takes the original images as input and we get the noisy images as output. Autoencoder will learn the relationship between a clean image and a noisy image and learn how to clean a noisy image.

- We explain Image denoising using MNIST image dataset.
- The dataset consists of images of handwritten digits in the form of a greyscale. It also includes labels for each image, telling us which digit it is (i.e., output for each image). It means we have a labelled data in our hands to work with. Each image in the MNIST dataset is 28 pixels by 28 pixels.

Fig. 3.7.1 : Original clean images from MNIST

- We split the dataset into 2 parts:
- **Training Dataset:** 60,000 data points belong to the training dataset, and
- **Testing Dataset:** 10,000 data points belongs to the test dataset.

Fig. 3.7.2 : Noisy images

Encoder-Decoder Network

It will have an input layer of 784 neurons since we have an image size of 784 due to 28 by 28 pixels present.

```
#create model
model = Sequential()
# Encoder
model.add(Dense(500, input_dim=num_pixels, activation='relu'))
model.add(Dense(300, activation='relu'))
# Code
model.add(Dense(100, activation='relu'))
# Decoder
model.add(Dense(300, activation='relu'))
model.add(Dense(500, activation='relu'))
model.add(Dense(784, activation='sigmoid'))
```

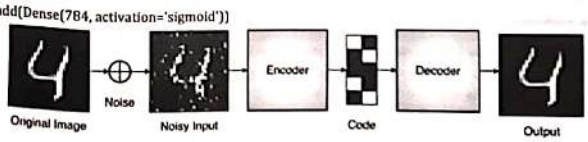

Fig. 3.7.3

3.8 Other Applications of Autoencoders

1. **Dimension reductionality**
   - 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.

2. Recommendation Systems

- Deep Autoencoders can be used to understand user preferences to recommend movies, books or other items. Consider the case of YouTube, the idea is:
- The input data is the clustering of similar users based on interests
- Interests of users are denoted by videos watched, watch time for each, interactions (like commenting) with the video
- Above data is captured by clustering content
- Encoder part will capture the interests of the user
- Decoder part will try to project the interests on two parts:
  - existing unseen content
  - new content from content creators

3. Anomaly detection

- Another application for autoencoders is anomaly detection. By learning to replicate the most salient features in the training data, the model is encouraged to learn to precisely reproduce the most frequently observed characteristics.
- When facing anomalies, the model should worsen its reconstruction performance.
- In most cases, only data with normal instances are used to train the autoencoder; in others, the frequency of anomalies is small compared to the observation set so that its contribution to the learned representation could be ignored. After training, the autoencoder will accurately reconstruct "normal" data, while failing to do so with unfamiliar anomalous data.
- Reconstruction error (the error between the original data and its low dimensional reconstruction) is used as an anomaly score to detect anomalies.

4. Image Production

- The VAE (Variational Autoencoder) is a generative model used to produce images that the model has not yet seen. The concept is that the system will generate similar images based on input photographs such as faces or scenery. The purpose is to:
  - Create new animated characters
  - Create fictitious human images
  - Colourization of an image
- One of the purposes of autoencoders is to convert a black-and-white image to a colored image. A colorful image can also be converted to grayscale

5. Image colorization

- The idea is to distill the salient features of an image and then regenerate the image based on these learned features. Traditionally, by using the output of the *encoder*, autoencoders are used to filter noise, compress data, and conduct dimensional reduction.
- More recently, however, there is increased use of the *decoder* to generate novel data (generative model). Since, if a well-trained decoder can generate accurate data from a set of representations, it should be able to generate data from representations it has never seen before.
- As an example, if the decoder can generate images of "cloud" and "pigs" from their representations, then when given the representation of "cloud + pigs", it should be able to generate pig-shaped clouds. Therefore, applications such as image colorization and style transfer, where the style of one painting is transferred to another, often have autoencoders embedded inside a deep neural network

Review Questions

Q. 1 What are autoencoders? Explain each component of autoencoder in details.

Q. 2 What are autoencoders? Discuss a few applications of autoencoders.

Q. 3 Explain linear autoencoders?

Q. 4 Write a short-note on sparse and contractive auto encoders.

Q. 5 Differentiate between under-complete and over-complete autoencoders.

Q. 6 Explain how autoencoders can be used for image compression.

Q. 7 Discuss autoencoder for image denoising.

Q. 8 Why regularization in the autoencoder is required? Explain any two Regularized autoencoders.

Q. 9 List and explain any five applications of autoencoders.

Q. 10 Explain de-noising autoencoders.

□□□