Unit 4

Autoencoders: Unsupervised Learning with Deep Network, Autoencoders (AEs), Regularization in autoencoders, Types of autoencoders (Denoising, sparse, contractive)

Unsupervised Neural Network

An unsupervised neural network is a type of artificial neural network (ANN) used in unsupervised learning tasks. Unlike supervised neural networks, trained on labeled data with explicit input-output pairs, unsupervised neural networks are trained on unlabeled data. In unsupervised learning, the network is not under the guidance of features. Instead, it is provided with unlabeled data sets (containing only the input data) and left to discover the patterns in the data and build a new model from it. Here, it has to figure out how to arrange the data by exploiting the separation between clusters within it. These neural networks aim to discover patterns, structures, or representations within the data without specific guidance.

There are several **components** of unsupervised learning. They are:

1. **Encoder-Decoder**: As the name itself suggests that it is used to encode and decode the data. Encoder basically responsible for transforming the input data into lower dimensional representation on which the neural network works. Whereas decoder takes the encoded representation and reconstruct the input data from it. There architecture and parameters are learned during the training of the network.
2. **Latent Space**: It is the immediate representation created by the encoder. It contains the abstract representation or features that captures important information about the data's structures. It is also known as the latent space.
3. **Training algorithm**: Unsupervised neural network model use specific training algorithms to get the parameters. Some of the common optimization algorithms are Stochastic gradient descent, Adam etc. They are used depending on the type of model and loss function.
4. **Loss Function**: It is a common component among all the machine learning models. It basically calculates the model's output and the actual/measured output. It quantifies how well the model understands the data.

Now, Let's discuss about some of the types of unsupervised neural network.

Autoencoder

A neural network that learn how to compress and decompress data. They are trained to reconstruct the input data from a compressed representation, which is further learned by the network during training of the model. It can be used for tasks such as image compression, dimensionality reduction, and denoising. Autoencoders are feedforward neural networks with an encoder and a decoder. They aim to map input data to a lower-dimensional representation and then reconstruct the data.

When input data is compressed into a lower-dimensional representation called the latent space, the first part of the neural network to do this is called an encoder. In order to facilitate effective data reconstruction by the decoder, it extracts important aspects from the data. Reducing dimensionality, learning features, and denoising data are common applications for autoencoders. The encoder maps the input data x to an encoding or hidden representation h using a set of weights (We) and biases (be):

where fe is the activation function of the encoder

The neural network component that reverses the encoding process is called the decoder. By using the encoder's compact, lower-dimensional representation (latent space), it reconstructs the original input data. Autoencoders can learn to denoise data or create new data based on learnt characteristics by using the decoder's goal of closely replicating the input. The decoder maps the encoding h back to the original input x using a different set of weights (Wd) and biases (bd):

An autoencoder's loss function measures how much the input data differs from its reconstruction. It calculates the discrepancy between the original and decoder-generated data. In order to help the model acquire useful feature representations in the latent space and provide correct reconstructions, the objective is to minimize this loss. The loss function measures the difference between the input x and the reconstructed output x':

Restricted Boltzmann Machine

A probabilistic model serves as the foundation for Restricted Boltzmann Machines (RBMs), which are unsupervised nonlinear feature learners. A linear classifier, such as a linear SVM or a perceptron, can often yield strong results when given features extracted by an RBM or a hierarchy of RBMs. About how the inputs are distributed, the model makes assumptions. Only BernoulliRBM is currently available through scikit-learn, and it expects that the inputs are binary or between 0 and 1, each of which encodes the likelihood that a given feature will be enabled. The RBM uses a specific graphical model to attempt to optimize the likelihood of the data. The representations capture interesting regularities since the parameter learning approach (Stochastic Maximum Likelihood) keeps them from deviating from the input data. However, this makes the model less useful for small datasets and typically not useful for density estimation.

Each RBM has an energy function that measures the compatibility between visible and hidden unit states. The energy of an RBM is defined as follows:

where E(v,h) is the energy of RBM of given state, Wij is weight, and ai and bj are biases.

The joint probability of a visible unit configuration (v) and a hidden unit configuration (h) is calculated using the energy function:

and for the marginal probabilities for visible and hidden units are obtained by summing over the respective variables:

Self-Organizing maps (SOM)

Kohonen maps, or self-organizing maps (SOM), are an intriguing class of unsupervised neural network models. They work especially well for comprehending and interpreting complex, high-dimensional data. The capacity of SOMs to reduce dimensionality while maintaining the topological linkages in the input data is one of its distinguishing features. A SOM starts with a grid of neurons, each of which represents a particular area of the data space. Through a process of competitive learning, the neurons adjust to the distribution of the input data throughout training. Neighboring neurons change their weights in response to similar data points, making them more sensitive to these patterns. This self-organizing characteristic produces a map that

places comparable data points in close proximity to one another, making it possible to see patterns and clusters in the data.
Applications for SOMs can be found in many areas, such as anomaly detection, feature extraction, and data clustering. They play a crucial role in data mining and exploratory analysis because they make it possible to find hidden structures in intricate datasets without the need for labeled data. SOMs are an important tool for unsupervised learning and data visualization because of their capacity to condense information while maintaining linkages.

Generative Adversarial Networks (GANs)

Generative Adversarial Networks (GANs) are a novel paradigm in the field of unsupervised neural networks. The discriminator and generator neural networks that make up a GAN are always in conflict with one another. As the discriminator works to separate authentic from produced data, the generator aims to create data samples that are identical to real data. After training, the generator in a GAN learns to produce data that is more and more realistic, starting out as random noise. Image generation, style transfer, and data augmentation all benefit from the adversarial training process that pushes the generator to provide data that is frequently very convincing. Drug discovery and natural language processing are two more fields in which GANs are being used.
GANs are an essential tool for unsupervised learning because of their ability to capture complex data distributions without the need for labeled input. However, they may require complex training, and they could be vulnerable to mode collapse, in which the generator concentrates on a small number of data patterns. Notwithstanding these difficulties, GANs have completely changed the field of machine learning and had a significant influence on many artistic and scientific domains.

Advantages of Unsupervised Neural network models

- **Feature Learning:** makes the model proficient in automatic learning and extracting relevant feature.
- **Dimensionality Reduction:** Since it works on the principle of dimensionality reduction, it preserves a lot of important information.
- **Generative model**: Generates new data samples which is valuable in tasks like data argumentation and data synthesis.
- **Data Preprocessing:** It can serve as a preprocessing step for supervised learning tasks. Extracted features or reduced-dimension representations can improve the performance of subsequent supervised models.

Disadvantages of Unsupervised Neural network models

- **Lack of labels**: Unlabeled data somehow reduces the precise prediction
- **Complexity**: It is complex and computationally expensive as it needs careful tuning of hyperparameters.
- **Difficulty in Evaluation:** Evaluating the performance of unsupervised models is more challenging than in supervised learning as there is no ground truth to compare against, making it harder to measure success.
- **Limited Applicability**: It is not suitable for all types of data or task as effectiveness depends on the quality and nature of the data.

AutoEncoders:

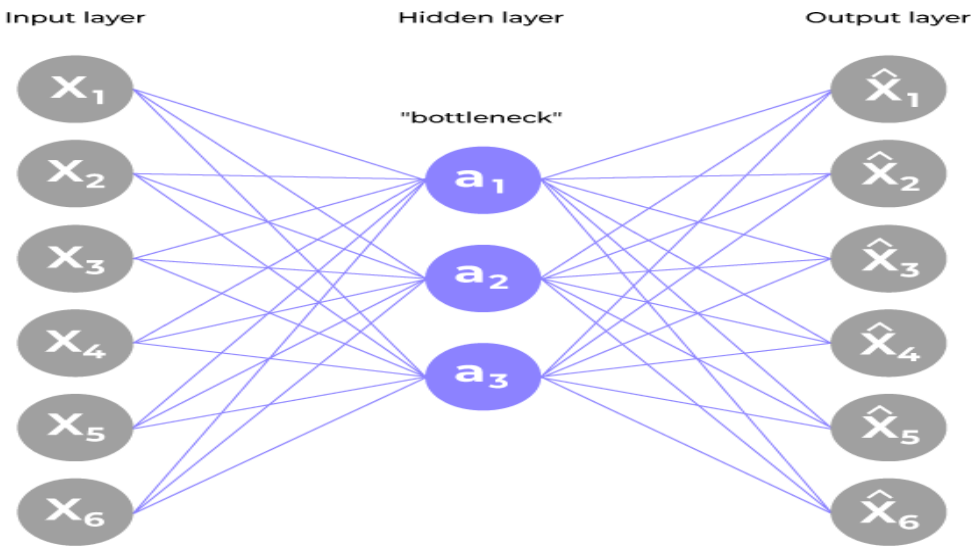
At the heart of **deep learning** lies the neural network, an intricate interconnected system of nodes that mimics the human brain's neural architecture. Neural networks excel at discerning intricate patterns and representations within vast datasets, allowing them to make predictions, classify information, and generate novel insights. **Autoencoders** emerge as a fascinating subset of neural networks, offering a unique approach to unsupervised learning. Autoencoders are an adaptable and strong class of architectures for the dynamic field of deep learning, where neural networks develop constantly to identify complicated patterns and representations. With their ability to learn effective representations of data, these unsupervised learning models have received considerable attention and are useful in a wide variety of areas, from image processing to anomaly detection.

What are Autoencoders?

Autoencoders are a specialized class of algorithms that can learn efficient representations of input data with no need for labels. It is a class of artificial neural networks designed for unsupervised learning. Learning to compress and effectively represent input data without specific labels is the essential principle of an automatic decoder. This is accomplished using a two-fold structure that consists of an encoder and a decoder. The encoder transforms the input data into a reduced-dimensional representation, which is often referred to as "latent space" or "encoding". From that representation, a decoder rebuilds the initial input. For the network to gain meaningful patterns in data, a process of encoding and decoding facilitates the definition of essential features.

Architecture of Autoencoder in Deep Learning

The general architecture of an autoencoder includes an encoder, decoder, and bottleneck layer.



1. Encoder
   - Input layer take raw input data
   - The hidden layers progressively reduce the dimensionality of the input, capturing important features and patterns. These layer compose the encoder.
   - The bottleneck layer (latent space) is the final hidden layer, where the dimensionality is significantly reduced. This layer represents the compressed encoding of the input data.

2. Decoder
    - The bottleneck layer takes the encoded representation and expands it back to the dimensionality of the original input.
    - The hidden layers progressively increase the dimensionality and aim to reconstruct the original input.
    - The output layer produces the reconstructed output, which ideally should be as close as possible to the input data.
3. The loss function used during training is typically a reconstruction loss, measuring the difference between the input and the reconstructed output. Common choices include mean squared error (MSE) for continuous data or binary cross-entropy for binary data.
4. During training, the autoencoder learns to minimize the reconstruction loss, forcing the network to capture the most important features of the input data in the bottleneck layer.

After the training process, only the encoder part of the autoencoder is retained to encode a similar type of data used in the training process. The different ways to constrain the network are: –

- **Keep small Hidden Layers:** If the size of each hidden layer is kept as small as possible, then the network will be forced to pick up only the representative features of the data thus encoding the data.
- **Regularization:** In this method, a loss term is added to the cost function which encourages the network to train in ways other than copying the input.
- **Denoising:** Another way of constraining the network is to add noise to the input and teach the network how to remove the noise from the data.
- **Tuning the Activation Functions:** This method involves changing the activation functions of various nodes so that a majority of the nodes are dormant thus, effectively reducing the size of the hidden layers.

Types of Autoencoders

There are diverse types of autoencoders and analyze the advantages and disadvantages associated with different variation:

Denoising Autoencoder

Denoising autoencoder works on a partially corrupted input and trains to recover the original undistorted image. As mentioned above, this method is an effective way to constrain the network from simply copying the input and thus learn the underlying structure and important features of the data.

Advantages

1. This type of autoencoder can extract important features and reduce the noise or the useless features.
2. Denoising autoencoders can be used as a form of data augmentation, the restored images can be used as augmented data thus generating additional training samples.

Disadvantages

1. Selecting the right type and level of noise to introduce can be challenging and may require domain knowledge.
2. Denoising process can result into loss of some information that is needed from the original input. This loss can impact accuracy of the output.

Sparse Autoencoder

This type of autoencoder typically contains more hidden units than the input but only a few are allowed to be active at once. This property is called the sparsity of the network. The sparsity of the network can be controlled by either manually zeroing the required hidden units, tuning the activation functions or by adding a loss term to the cost function.

Advantages

1. The sparsity constraint in sparse autoencoders helps in filtering out noise and irrelevant features during the encoding process.
2. These autoencoders often learn important and meaningful features due to their emphasis on sparse activations.

Disadvantages

1. The choice of hyperparameters play a significant role in the performance of this autoencoder. Different inputs should result in the activation of different nodes of the network.
2. The application of sparsity constraint increases computational complexity.

Variational Autoencoder

Variational autoencoder makes strong assumptions about the distribution of latent variables and uses the **Stochastic Gradient Variational Bayes** estimator in the training process. It assumes that the data is generated by a **Directed Graphical Model** and tries to learn an approximation to to the conditional property where and are the parameters of the encoder and the decoder respectively.

Advantages

1. Variational Autoencoders are used to generate new data points that resemble the original training data. These samples are learned from the latent space.
2. Variational Autoencoder is probabilistic framework that is used to learn a compressed representation of the data that captures its underlying structure and variations, so it is useful in detecting anomalies and data exploration.

Disadvantages

1. Variational Autoencoder use approximations to estimate the true distribution of the latent variables. This approximation introduces some level of error, which can affect the quality of generated samples.
2. The generated samples may only cover a limited subset of the true data distribution. This can result in a lack of diversity in generated samples.

Convolutional Autoencoder

Convolutional autoencoders are a type of autoencoder that use convolutional neural networks (CNNs) as their building blocks. The encoder consists of multiple layers that take a image or a grid as input and pass it through different convolution layers thus forming a compressed representation of the input. The decoder is the mirror image of the encoder it deconvolves the compressed representation and tries to reconstruct the original image.

Advantages

1. Convolutional autoencoder can compress high-dimensional image data into a lower-dimensional data. This improves storage efficiency and transmission of image data.
2. Convolutional autoencoder can reconstruct missing parts of an image. It can also handle images with slight variations in object position or orientation.

Disadvantages

1. These autoencoder are prone to overfitting. Proper regularization techniques should be used to tackle this issue.
2. Compression of data can cause data loss which can result in reconstruction of a lower quality image.

Autoencoders in Regularization:

Regularization helps with the effects of out-of-control parameters by using different methods to minimize parameter size over time.

In mathematical notation, we see regularization represented by the coefficient lambda, controlling the trade-off between finding a good fit and keeping the value of certain feature weights low as the exponents on features increase.

Regularization coefficients L1 and L2 help fight overfitting by making certain weights smaller. Smaller-valued weights lead to simpler hypotheses, which are the most generalizable. Unregularized weights with several higher-order polynomials in the feature sets tend to overfit the training set.

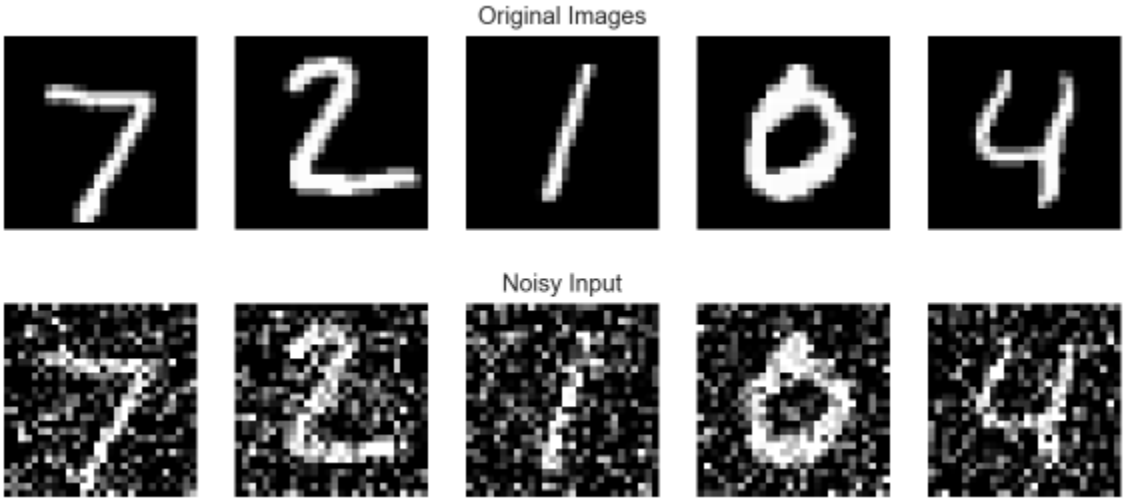
As the input training set size grows, the effect of regularization decreases, and the parameters tend to increase in magnitude. This is appropriate because an excess of features relative to training set examples leads to overfitting in the first place. Bigger data is the ultimate regularizer.

Regularized autoencoders

There are other ways to constrain the reconstruction of an autoencoder than to impose a hidden layer of smaller dimensions than the input. The regularized autoencoders use a loss function that helps the model to have other properties besides copying input to the output. We can generally find two types of regularized autoencoder: the denoising autoencoder and the sparse autoencoder.

Denoising autoencoder

We can modify the autoencoder to learn useful features is by changing the inputs; we can add random noise to the input and recover it to the original form by removing noise from the input data. This prevents the autoencoder from copying the data from input to output because it contains random noise. We ask it to subtract the noise and produce meaningful underlying data. This is called a denoising autoencoder.



In the above diagram, the first row contains original images. We can see in the second row that random noise is added to the original images; this noise is called Gaussian noise. The input of the autoencoder will not get the original images, but autoencoders are trained in such a way that they will remove noise and generate the original images.

The only difference between implementing the denoising autoencoder and the normal autoencoder is a change in input data. The rest of the implementation is the same for both the autoencoders. Below is the difference between training the autoencoder.
Training simple autoencoder:

```
autoencoder.fit(x_train, x_train)
```

Training denoising autoencoder:

```
autoencoder.fit(x_train_noisy, x_train)
```

Simple as that, everything else is exactly the same. The input to the autoencoder is the noisy image, and the expected target is the original noise-free one.

Sparse autoencoders

Another way of regularizing the autoencoder is by using a sparsity constraint. In this way of regularization, only fraction nodes are allowed to do forward and backward propagation. These nodes have non-zero values and are called active nodes.
To do so, we add a penalty term to the loss function, which helps to activate the fraction of nodes. This forces the autoencoder to represent each input as a combination of a small number of nodes and demands it to discover interesting structures in the data. This method is efficient even if the code size is large because only a small subset of the nodes will be active.
For example, add a regularization term in the loss function. Doing this will make our autoencoder learn the sparse representation of data.

```
input_size = 256
hidden_size = 32
output_size = 256
l1 = Input(shape=(input_size,))
# Encoder
h1 = Dense(hidden_size ,activity_regularizer=regularizers.l1(10e-6), activation='relu')(l1)
# Decoder
l2 = Dense(output_size, activation='sigmoid')(h1)
autoencoder = Model(input=l1, output=l2)
autoencoder.compile(loss='mse', optimizer='adam’)
```

In the above code, we have added L1 regularization to the hidden layer of the encoder, which adds the penalty to the loss function.

Training of Variational Autoencoder

To train a Variational Autoencoder (VAE) on a dataset, several modifications are needed compared to a standard autoencoder. Here's an outline of those changes and an explanation of the reparameterization trick:

Modifications for Variational Autoencoder

1. **Latent Space Representation**:
   - In a standard autoencoder, the latent space is a deterministic representation of the input. In contrast, a VAE learns to model the distribution of the latent variables. Instead of mapping inputs to a single point in the latent space, a VAE outputs parameters of a probability distribution (typically a Gaussian) for each latent variable.

2. **Encoder Output**:
   - The encoder network outputs two vectors: one for the mean (μ) and one for the log variance ($\log(\sigma^2)$). This represents the learned distribution over the latent space.

3. **Decoder Input**:
   - The decoder receives samples drawn from the latent distribution rather than a single deterministic point. This allows for generating diverse outputs from similar inputs.

4. **Loss Function**:
   - The loss function consists of two parts:
     - The reconstruction loss (similar to standard autoencoders, usually Mean Squared Error).
     - The Kullback-Leibler (KL) divergence loss, which measures how much the learned latent distribution diverges from a prior distribution (typically a standard Gaussian).

Implementing the Reparameterization Trick

The reparameterization trick is essential for enabling backpropagation through the stochastic sampling process. Here's how it works:

1. **Sampling**:
   - From the encoder, you obtain the mean (μ) and standard deviation (σ) of the latent space distribution. You want to sample z from this distribution: $z \sim \mathcal{N}(\mu, \sigma^2)$.

2. **Reparameterization**:
   - Instead of directly sampling from $\mathcal{N}(\mu, \sigma^2)$, you can rewrite this as:
   $$
   z = \mu + \sigma \cdot \epsilon
   $$
   where $\epsilon \sim \mathcal{N}(0, 1)$ is a random noise term. This formulation allows the model to be differentiable with respect to μ and σ, enabling gradient descent optimization.

Training Steps

1. **Construct the Encoder**:
   - Build a neural network to output \(\mu \) and \(\log(\sigma^2) \) from the input data.

2. **Sample from the Latent Space**:
   - Use the reparameterization trick to sample \(z \) during training.

3. **Construct the Decoder**:
   - Build a neural network to reconstruct the input from the sampled \(z \).

4. **Define the Loss Function**:
   - Combine the reconstruction loss and the KL divergence loss.

5. **Optimize**:
   - Use a suitable optimizer (like Adam) to minimize the total loss during training.

This approach allows the VAE to learn a meaningful latent space representation, facilitating tasks like data generation and interpolation between different samples.