**DEEP LEARNING ASSIGNMENT\_9**

**1.What are the main tasks that autoencoders are used for?**

Autoencoders are a type of neural network architecture that are primarily used for unsupervised learning tasks. Some of the main tasks that autoencoders are used for include:

Dimensionality reduction: Autoencoders can be used to learn a compact representation, or encoding, of high-dimensional data, such as images or text. This encoding can then be used for other tasks, such as visualization or classification.

Anomaly detection: Autoencoders can be trained to reconstruct normal data, and then used to detect instances that are significantly different from the normal data.

Generative modeling: Autoencoders can be used to generate new data samples that are similar to the training data. This can be achieved by sampling from the latent space learned by the autoencoder, or by using the decoder component of the autoencoder to generate new data.

Image denoising: Autoencoders can be trained to denoise images by reconstructing a clean version of an image that has been corrupted with noise.

Pre-training: Autoencoders can be used as a pre-training step for supervised learning tasks, such as classification or regression. The learned encodings can be used as input features for the supervised task, potentially improving performance.

Overall, autoencoders are a versatile tool that can be used for a variety of unsupervised learning tasks, including dimensionality reduction, anomaly detection, generative modeling, image denoising, and pre-training.

**2. Suppose you want to train a classifier, and you have plenty of unlabeled training data but only a few thousand labeled instances. How can autoencoders help? How would you proceed?**

Autoencoders can be used to help train a classifier when there is limited labeled training data available. Here is one possible way to proceed:

Train an autoencoder on the unlabeled training data: The autoencoder can be trained to reconstruct the input data, using the encoded representations as the hidden layer activations.

Use the encoded representations as features for a classifier: The encoded representations learned by the autoencoder can be used as features for a classifier, such as a logistic regression or support vector machine. The classifier can then be trained on the few thousand labeled instances.

Fine-tune the autoencoder: Once the classifier has been trained, the autoencoder can be fine-tuned to improve the quality of the encoded representations, using the labeled data as a supervision signal.

By using the autoencoder to learn useful features from the large amount of unlabeled data, it's possible to train a classifier even when there is limited labeled data available. The fine-tuning step can further improve the quality of the encoded representations, potentially improving the performance of the classifier.

It's worth noting that this approach is not limited to classifiers, and can also be used for other supervised learning tasks, such as regression or sequence labeling. The general idea is to use the autoencoder to learn useful representations from the large amount of unlabeled data, which can then be used as features for a supervised learning task.

**3. If an autoencoder perfectly reconstructs the inputs, is it necessarily a good autoencoder? How can you evaluate the performance of an autoencoder?**

Autoencoders are a common tool for training neural network algorithms, but developers need to be mindful of the challenges that come with using them skillfully.

Autoencoders are additional neural networks that work alongside machine learning models to help data cleansing, denoising, feature extraction and dimensionality reduction.

An autoencoder is made up by two neural networks: an encoder and a decoder. The encoder works to code data into a smaller representation (bottleneck layer) that the decoder can then convert into the original input. Autoencoders distill inputs into the densest amount of data necessary to re-create a similar output. This removes data noise, transforms raw files into clean machine learning data and detects anomalies.

While autoencoders have data-cleansing power, they are not a one-size-fits-all tool and come with a lot of applicational errors. Data scientists using autoencoders for machine learning should look out for these eight specific problems.

1. Insufficient training data

Autoencoders are an unsupervised technique that learns from its own data rather than labels created by humans. This often means that autoencoders need a considerable amount of clean data to generate useful results. They can deliver mixed results if the data set is not large enough, is not clean or is too noisy.

"To maintain a robust autoencoder, you need a large representative data set and to recognize that training a robust autoencoder will take time," said Pat Ryan, chief architect at SPR, a digital tech consultancy.

Autoencoders help data scientists automate the organization of data for the machine learning pipeline.

2. Training the wrong use case

Not only do autoencoders need a comprehensive amount of training data, they also need relevant data. Like many algorithms, autoencoders are data-specific and data scientists must consider the different categories represented in a data set to get the best results.

"If one trains an autoencoder in a compression context on pictures of dogs, it will not generalize well to an application requiring data compression on pictures of cars," said Nathan White, lead consultant of data science and machine learning at AIM Consulting Group.

It is vital to make sure the available data matches the business or research goal; otherwise, valuable time will be wasted on the training and model-building processes. In some cases, it may be useful to segment the data first using other unsupervised techniques before feeding each segment into a different autoencoder.

3. Too lossy

Additionally, autoencoders are lossy, which limits their use in applications when compression degradation affects system performance in a significant way. White said there is no way to eliminate the image degradation, but developers can contain loss by aggressively pruning the problem space. In this case, the autoencoder would be more aligned with compressing the data relevant to the problem to be solved. For example, in a predictive analytics application, the resulting encodings would be scored on how well they align with predictions related to common business problems in a domain.

4. Imperfect decoding

Semirelated to being lossy, the decoder process is never perfect. In some circumstances, Ryan said it becomes a business decision to decide how much loss is tolerable in the reconstructed output. This can be important in applications such as anomaly detection. In these cases, data scientists need to continually monitor the performance and update it with new samples. He stressed that anomalies are not necessarily problems and sometimes represent new business opportunities.

5. Misunderstanding important variables

The biggest challenge with autoencoders is understanding the variables that are relevant to a project or model, said Russ Felker, CTO of GlobalTranz, a logistics service and freight management provider.

Developing a good autoencoder can be a process of trial and error, and, over time, data scientists can lose the ability to see which factors are influencing the results.

Felker recommended thinking about autoencoders as a business and technology partnership to ensure there is a clear and deep understanding of the business application. For example, implementing an image recognition algorithm might be easy in a small-scale application, but it can be a very different process in a different business context. Data scientists need to work with business teams to figure out the application, perform appropriate tests and determine the value of the application.

6. Better alternatives

Data scientists must evaluate data characteristics to deem data sets fit for the use of autoencoders, said CG Venkatesh, global head of data science, AI, machine learning and cognitive practice at Larsen and Toubro Infotech Ltd., a global IT services provider. While the use of autoencoders is attractive, use cases like image compression are better suited for other alternatives. Alternatively, data scientists need to consider implementing autoencoders as part of a pipeline with complementary techniques. If there is a large number of variables, autoencoders can be used for dimension reduction before the data is processed by other algorithms.

Venkatesh recommended doing trial runs with various alternatives to get a sense of whether to use autoencoders or explore how they might work alongside other techniques. If autoencoders show promise, then data scientists can optimize them for a specific use case.

7. Algorithms become too specialized

Training autoencoders to learn and reproduce input features is unique to the data they are trained on, which generates specific algorithms that don't work as well for new data. The network can simply remember the inputs it was trained on without necessarily understanding the conceptual relations between the features, said Sriram Narasimhan, vice president for AI and analytics at Cognizant. This problem can be overcome by introducing loss regularization using contractive autoencoder architectures. Another approach is to introduce a small amount of random noise during training to improve the sturdiness of the algorithm.

8. Bottleneck layer is too narrow

A typical autoencoder consists of multiple layers of progressively fewer neurons for encoding the original input called a bottleneck layer. One danger is that the resulting algorithms may be missing important dimensions for the problem if the bottleneck layer is too narrow. This problem can be avoided by testing reconstruction accuracy for varying sizes of the bottleneck layer, Narasimhan said. Narrow layers can also make it difficult to interpret the dimensions embedded in the data. When this becomes a problem, he recommended increasing the bottleneck layer, even if there is a minor trade-off in reproduction loss.

Sparse Overcomplete Autoencoder

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.

Advantages-

Sparse autoencoders have a sparsity penalty, a value close to zero but not exactly zero. Sparsity penalty is applied on the hidden layer in addition to the reconstruction error. This prevents overfitting.

They take the highest activation values in the hidden layer and zero out the rest of the hidden nodes. This prevents autoencoders to use all of the hidden nodes at a time and forcing only a reduced number of hidden nodes to be used.

Drawbacks-

For it to be working, it's essential that the individual nodes of a trained model which activate are data dependent, and that different inputs will result in activations of different nodes through the network.

**4. What are undercomplete and overcomplete autoencoders? What is the main risk of an excessively undercomplete autoencoder? What about the main risk of an overcomplete autoencoder?**

Undercomplete Autoencoder

The objective of undercomplete autoencoder is to capture the most important features present in the data. Undercomplete autoencoders have a smaller dimension for hidden layer compared to the input layer. This helps to obtain important features from the data. It minimizes the loss function by penalizing the g(f(x)) for being different from the input x.

Advantages-

Undercomplete autoencoders do not need any regularization as they maximize the probability of data rather than copying the input to the output.

Drawbacks-

Using an overparameterized model due to lack of sufficient training data can create overfitting.

Sparse Overcomplete Autoencoder

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.

Advantages-

Sparse autoencoders have a sparsity penalty, a value close to zero but not exactly zero. Sparsity penalty is applied on the hidden layer in addition to the reconstruction error. This prevents overfitting.

They take the highest activation values in the hidden layer and zero out the rest of the hidden nodes. This prevents autoencoders to use all of the hidden nodes at a time and forcing only a reduced number of hidden nodes to be used.

Drawbacks-

For it to be working, it's essential that the individual nodes of a trained model which activate are data dependent, and that different inputs will result in activations of different nodes through the network.

**5. How do you tie weights in a stacked autoencoder? What is the point of doing so?**

Tying weights

To understand the concept of tying weights we need to find the answers of three questions about it. what , why and when. Lets start with when to use it? So when the autoencoder is typically symmetrical, it is a common practice to use tying weights . Now what is it? This is nothing but tying the weights of the decoder layer to the weights of the encoder layer. Next is why we need it? This reduces the number of weights of the model almost to half of the original, thus reducing the risk of over-fitting and speeding up the training process.

Implementation of Tying Weights: To implement tying weights, we need to create a custom layer to tie weights between the layer using keras. This custom layer acts as a regular dense layer, but it uses the transposed weights of the encoder’s dense layer, however having its own bias vector.

**6. What is a generative model? Can you name a type of generative autoencoder?**

A generative model includes the distribution of the data itself, and tells you how likely a given example is. For example, models that predict the next word in a sequence are typically generative models (usually much simpler than GANs) because they can assign a probability to a sequence of words.

Autoencoders are also generative models which can randomly generate new data that is similar to the input data (training data)

**7. What is a GAN? Can you name a few tasks where GANs can shine?**

Generative Adversarial Networks, or GANs for short, are an approach to generative modeling using deep learning methods, such as convolutional neural networks.

Generative modeling is an unsupervised learning task in machine learning that involves automatically discovering and learning the regularities or patterns in input data in such a way that the model can be used to generate or output new examples that plausibly could have been drawn from the original dataset.

GANs are a clever way of training a generative model by framing the problem as a supervised learning problem with two sub-models: the generator model that we train to generate new examples, and the discriminator model that tries to classify examples as either real (from the domain) or fake (generated). The two models are trained together in a zero-sum game, adversarial, until the discriminator model is fooled about half the time, meaning the generator model is generating plausible examples.

GANs are an exciting and rapidly changing field, delivering on the promise of generative models in their ability to generate realistic examples across a range of problem domains, most notably in image-to-image translation tasks such as translating photos of summer to winter or day to night, and in generating photorealistic photos of objects, scenes, and people that even humans cannot tell are fake.

Application:

* Generate Examples for Image Datasets
* Generate Photographs of Human Faces
* Generate Realistic Photographs
* Generate Cartoon Characters
* Image-to-Image Translation
* Text-to-Image Translation
* Semantic-Image-to-Photo Translation
* Face Frontal View Generation
* Generate New Human Poses
* Photos to Emojis
* Photograph Editing
* Face Aging
* Photo Blending
* Super Resolution
* Photo Inpainting
* Clothing Translation
* Video Prediction
* 3D Object Generation

**8. What are the main difficulties when training GANs?**

GANs are difficult to train.

The reason they are difficult to train is that both the generator model and the discriminator model are trained simultaneously in a game. This means that improvements to one model come at the expense of the other model.