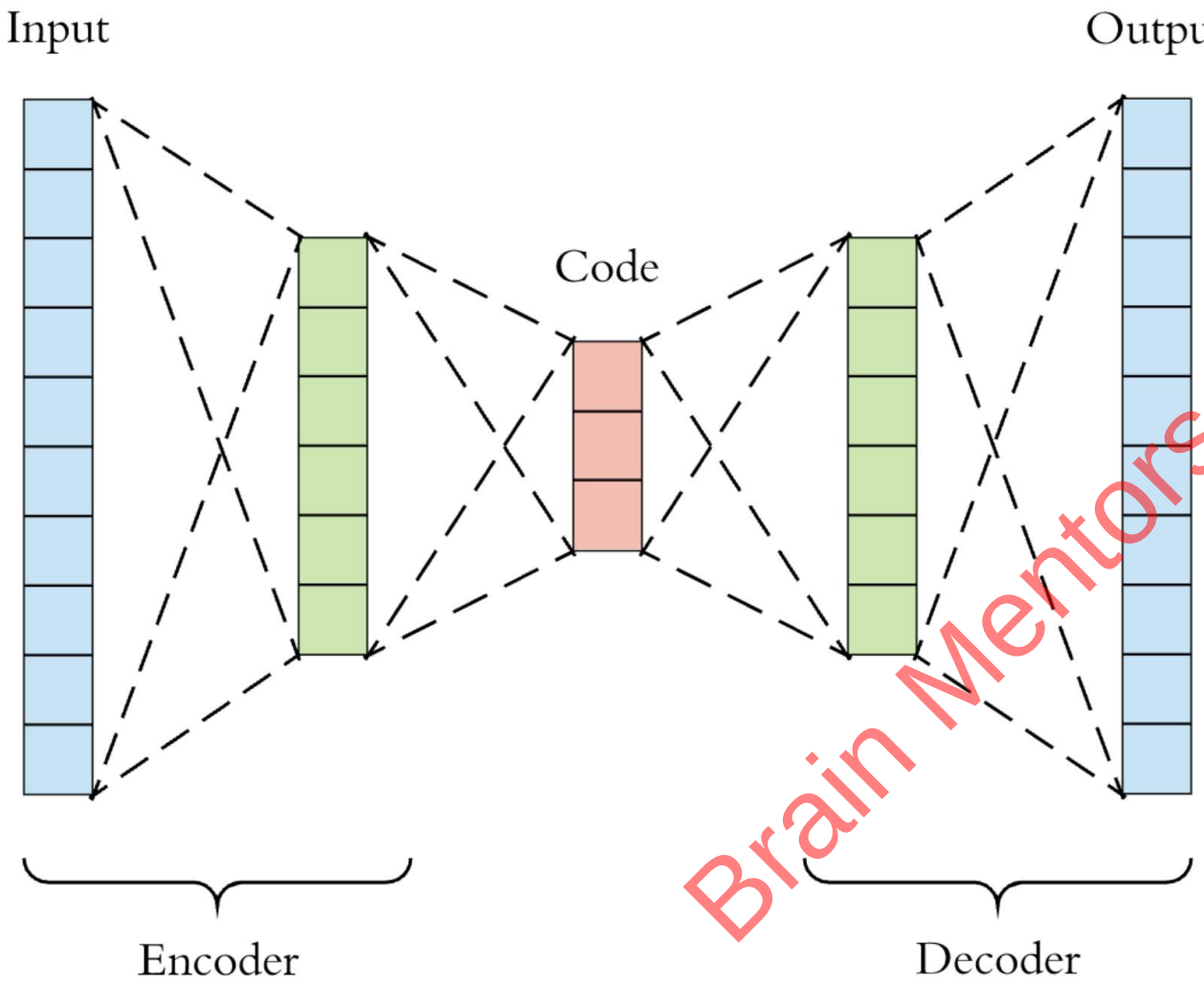


Auto Encoders

By - Ravi

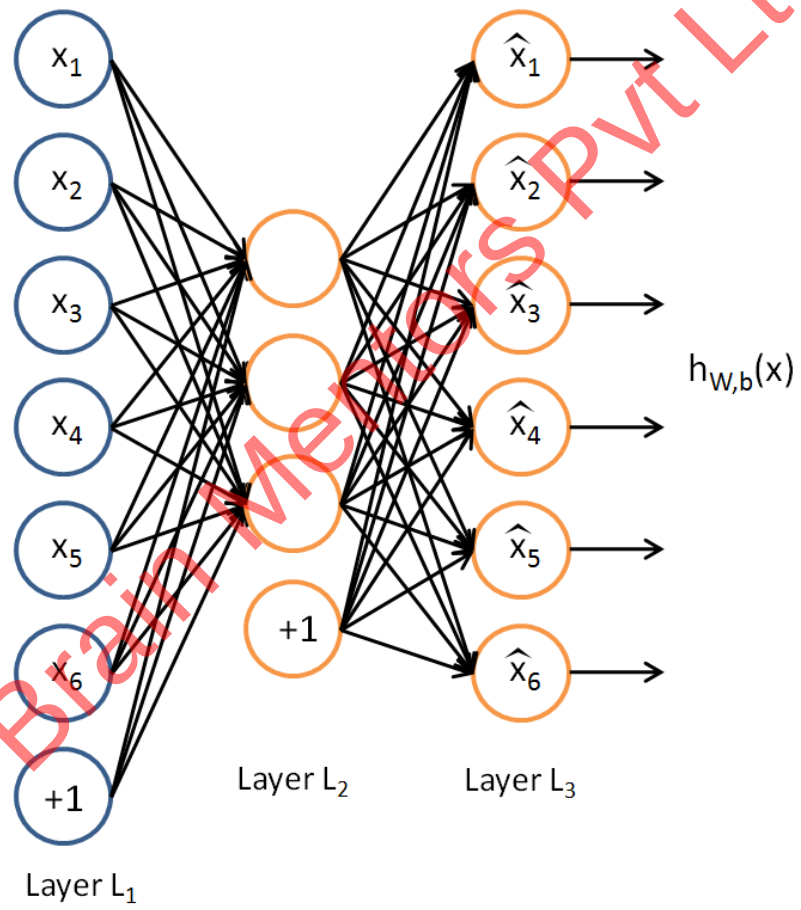
Auto Encoders

- An Autoencoder is a neural network which is an unsupervised learning algorithm which uses back propagation to generate output value which is almost close to the input value.
- It takes input such as image or vector anything with a very high dimensionality and run through the neural network and tries to compress the data into a smaller representation with two principal components.

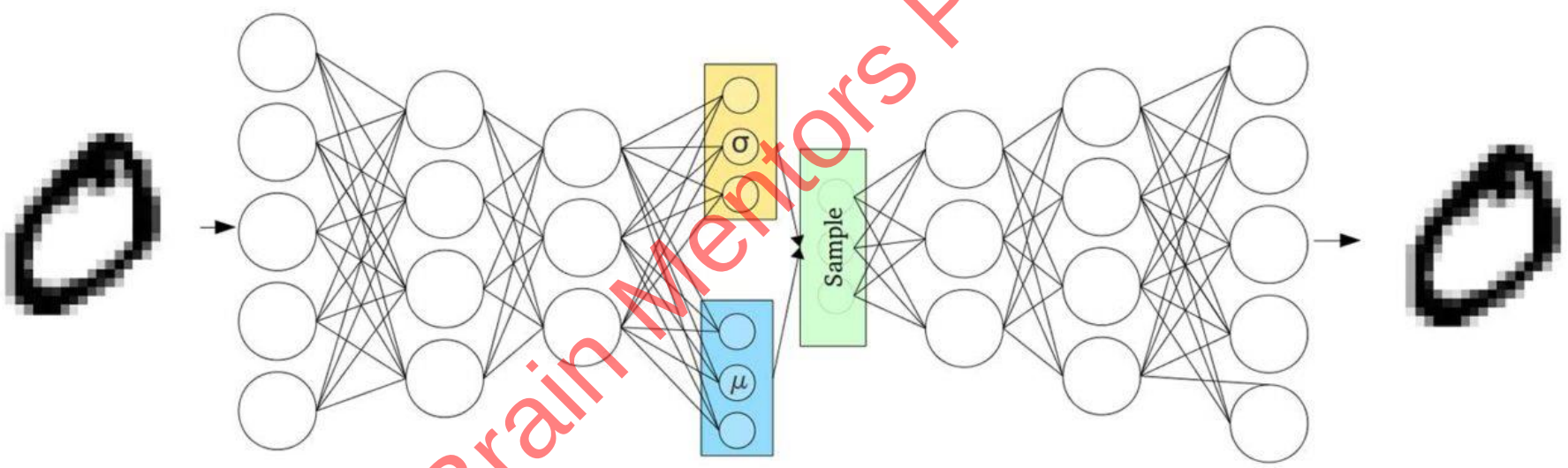
Auto Encoders

- The first one is the encoder which is simply a bunch of layers that are full connected layers or convolutional layers which are going to take the input and compress it to a smaller representation which has less dimensions then the input which is known as bottleneck.
- Now from this bottleneck it tries to reconstruct the input using full connected layers or convolutional layers.
- Autoencoders are used to reduce the size of our inputs into a smaller representation. If anyone needs the original data, they can reconstruct it from the compressed data.

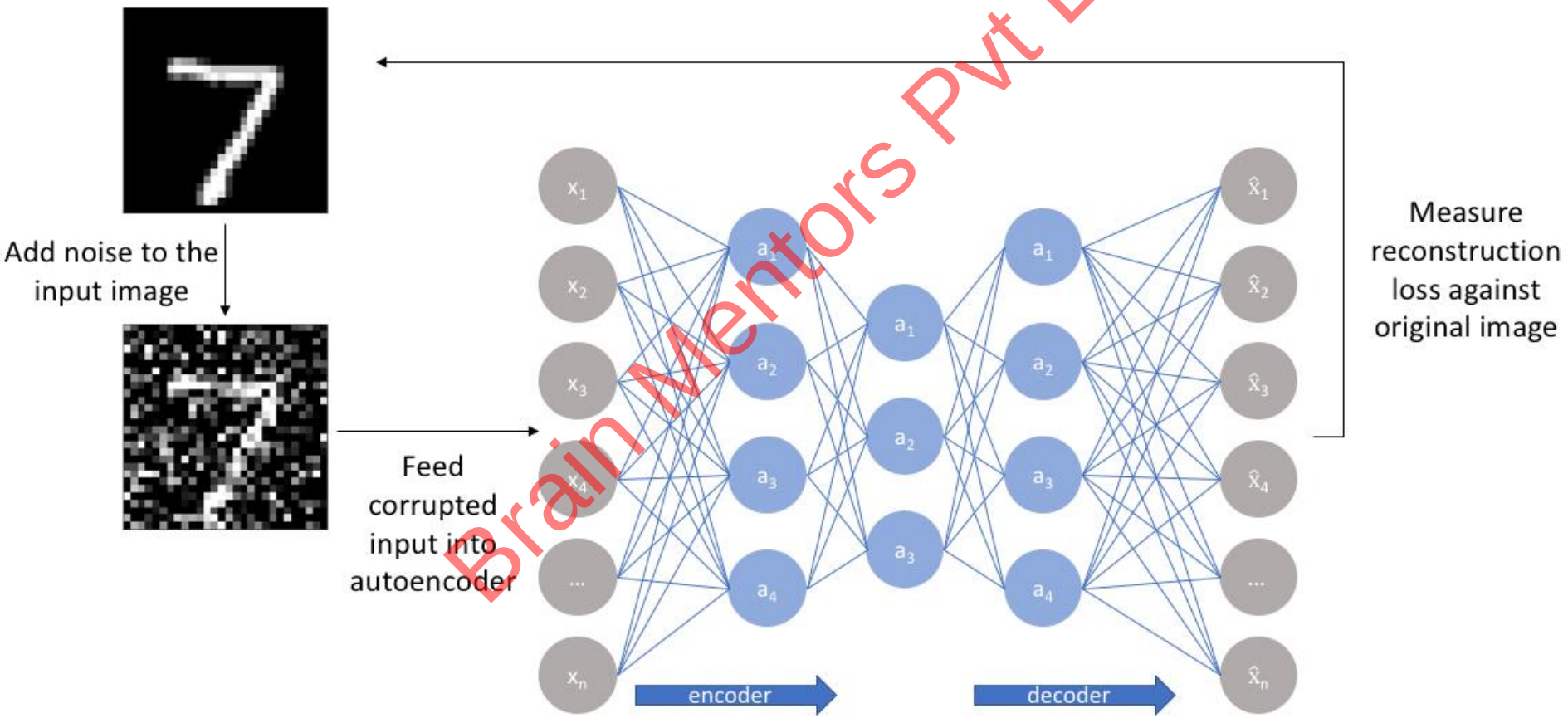
Auto Encoders



Auto Encoders



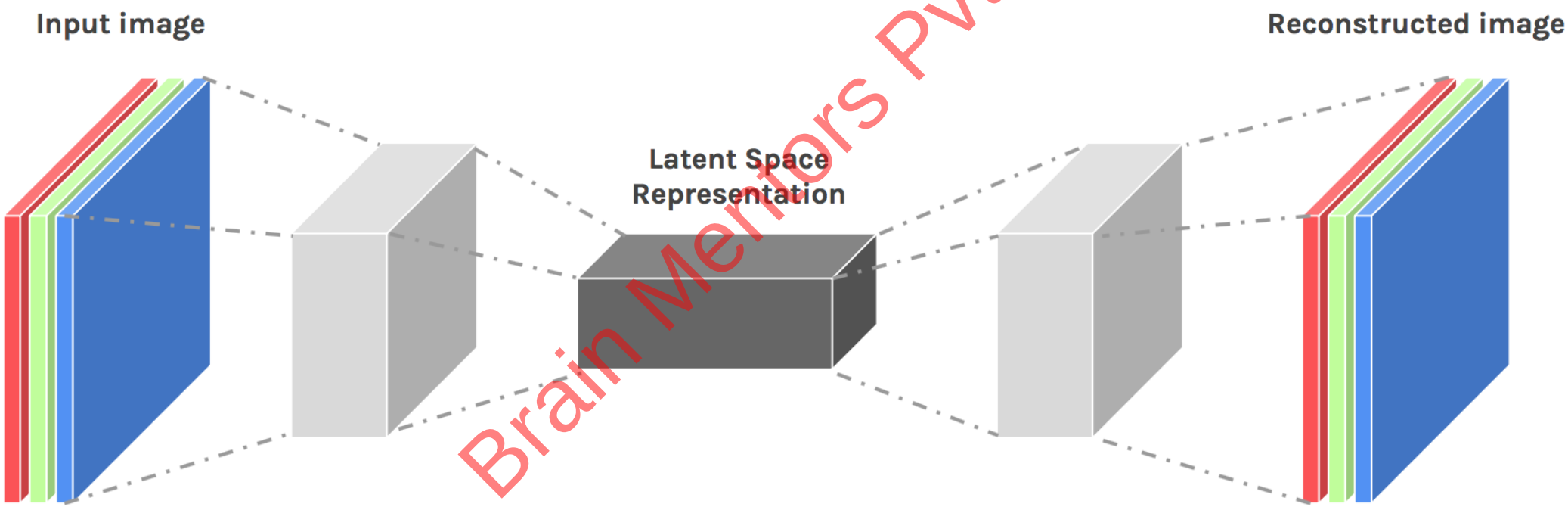
Auto Encoders



Introduction

- An *autoencoder* is a neural network that learns to copy its input to its output. It has an internal (*hidden*) layer that describes a *code* used to represent the input, and it is constituted by two main parts: an encoder that maps the input into the code, and a decoder that maps the code to a reconstruction of the original input.
- They work by compressing the input into a latent-space representation and then reconstructing the output from this representation.

Introduction



Why Autoencoders?

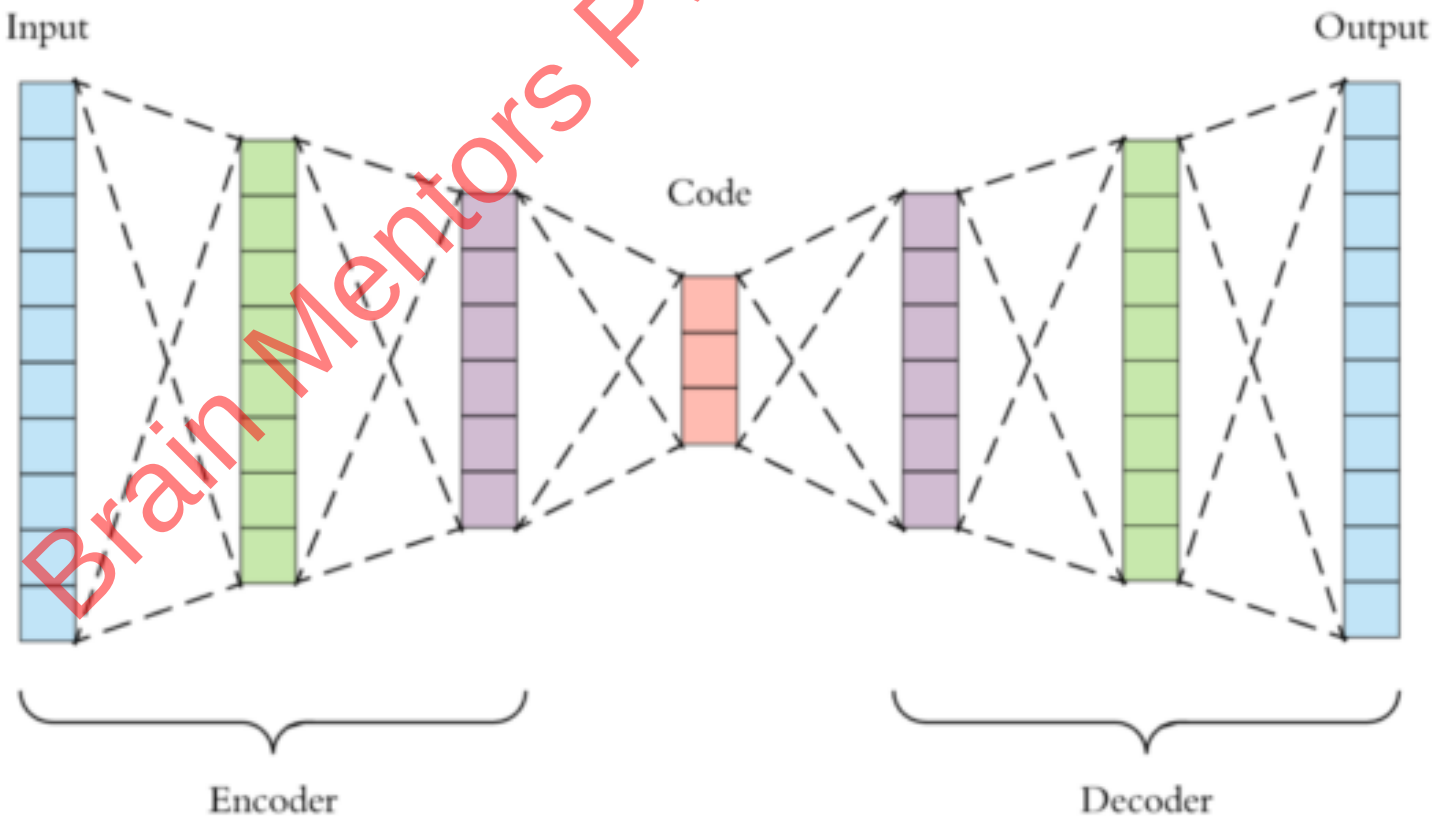
Despite the fact, the practical applications of autoencoders were pretty rare some time back, today **data denoising** and **dimensionality reduction for data visualization** are considered as two main interesting practical applications of autoencoders. With appropriate dimensionality and sparsity constraints, autoencoders can learn data projections that are more interesting than PCA or other basic techniques.

Why Autoencoders?

Architecture of Autoencoders

An Autoencoder consist of three layers:

- **Encoder**
- **Code**
- **Decoder**



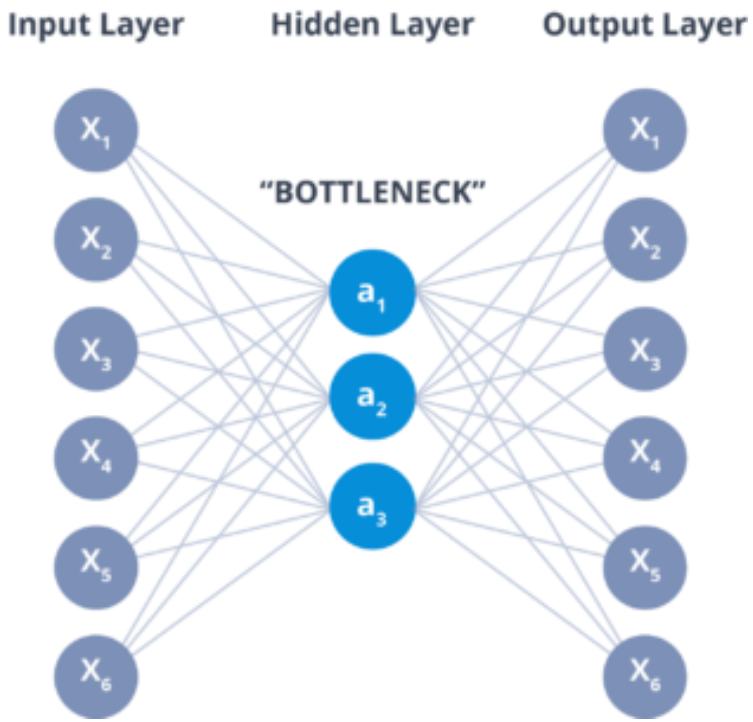
Architecture of Autoencoders

- **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.

Architecture of Autoencoders

The layer between the encoder and decoder, ie. the code is also known as **Bottleneck**. This is a well-designed approach to decide which aspects of observed data are relevant information and what aspects can be discarded. It does this by balancing two criteria :

- Compactness of representation, measured as the compressibility.
- It retains some behaviourally relevant variables from the input.



Properties and Hyperparameters

In more terms, autoencoding is a data compression algorithm where the compression and decompression functions are,

Properties of Autoencoders:

- **Data-specific:** Autoencoders are only able to compress data similar to what they have been trained on. An autoencoder which has been trained on human faces would not be performing well with images of modern buildings. This improvises the difference between autoencoders and MP3 kind of compression algorithms which only hold assumptions about sound in general, but not about specific types of sounds.

Autoencoders

- **Lossy:** This means that the decompressed outputs will be degraded compared to the original inputs. Just like what you see in JPEG or MP3.
- **Learned automatically from examples:** If you have appropriate training data, it is easy to train specialized instances of the algorithm that will perform well on a specific type of input. It doesn't require any new engineering.

Additionally, in almost all contexts where the term "autoencoder" is used, the compression and decompression functions are implemented with neural networks.

Hyperparameters of Autoencoders:

There are **4** hyperparameters that we need to set before training an autoencoder:

- **Code size**: It represents the number of nodes in the middle layer. Smaller size results in more compression.
- **Number of layers**: The autoencoder can consist of as many layers as we want.
- **Number of nodes per layer**: The number of nodes per layer decreases with each subsequent layer of the encoder, and increases back in the decoder. The decoder is symmetric to the encoder in terms of the layer structure.
- **Loss function:** We either use mean squared error or binary cross-entropy. If the input values are in the range [0, 1] then we typically use cross-entropy, otherwise, we use the mean squared error.

The problem with standard autoencoders

- Standard autoencoders learn to generate compact representations and reconstruct their inputs well, but asides from a few applications like denoising autoencoders, they are fairly limited.
- The fundamental problem with autoencoders, for generation, is that the latent space they convert their inputs to and where their encoded vectors lie, may not be continuous, or allow easy interpolation.
- For example, training an autoencoder on the MNIST dataset, and visualizing the encodings from a 2D latent space reveals the formation of distinct clusters. This makes sense, as distinct encodings for each image type makes it far easier for the decoder to decode them. This is fine if you're just *replicating* the same images.

The problem with standard autoencoders

- But when you're building a *generative* model, you **don't** want to prepare to *replicate* the same image you put in. You want to randomly sample from the latent space, or generate variations on an input image, from a continuous latent space.
- If the space has discontinuities (eg. gaps between clusters) and you sample/generate a variation from there, the decoder will simply generate an unrealistic output, because the decoder has *no idea* how to deal with that region of the latent space. During training, it *never saw* encoded vectors coming from that region of latent space.

Types of Autoencoders

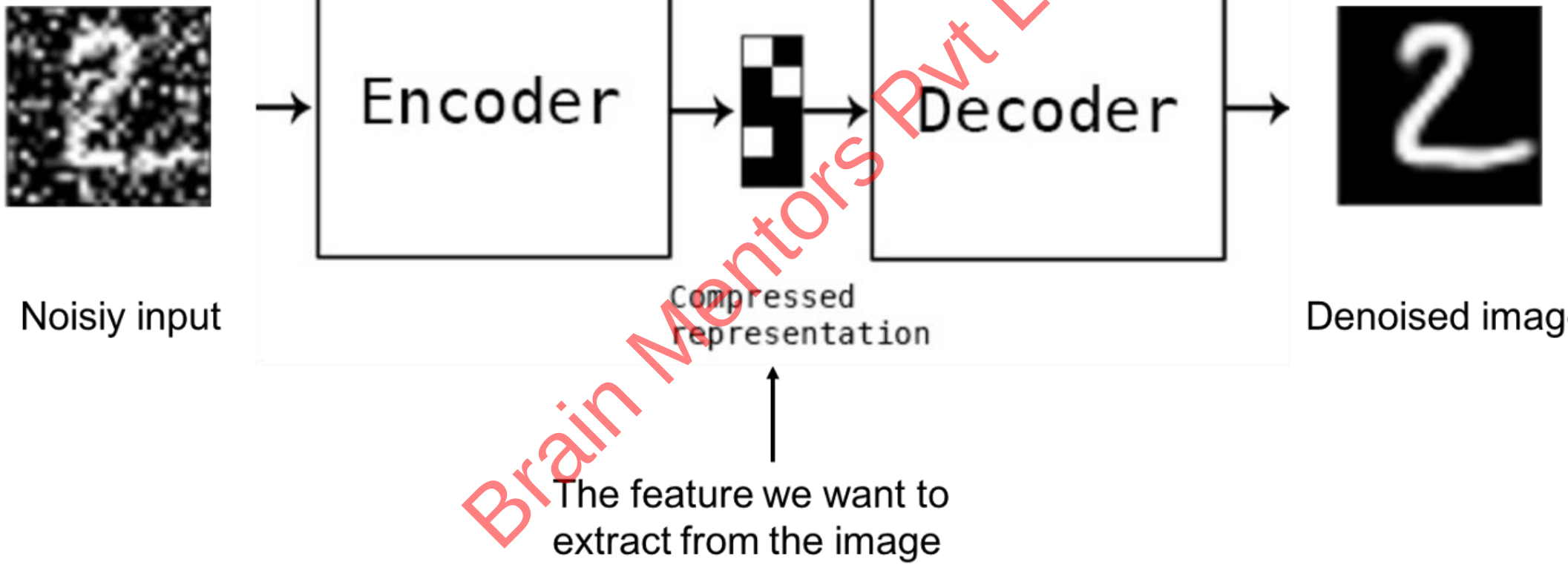
- **Denoising autoencoder**
- **Sparse Autoencoder**
- **Deep Autoencoder**
- **Contractive Autoencoder**
- **Undercomplete Autoencoder**
- **Convolutional Autoencoder**
- **Stacked Autoencoder**
- **Variational Autoencoder**
- **Convolutional Variational Autoencoder**

Denoising Autoencoder

- Denoising autoencoders create a corrupted copy of the input by introducing some noise. This helps to avoid the autoencoders to copy the input to the output without learning features about the data.
- These autoencoders take a partially corrupted input while training to recover the original undistorted input.
- The model learns a vector field for mapping the input data towards a lower dimensional manifold which describes the natural data to cancel out the added noise.

Denoising Autoencoder

- It was introduced to achieve good representation. Such a representation is one that can be obtained robustly from a corrupted input and that will be useful for recovering the corresponding clean input.
- Corruption of the input can be done randomly by making some of the input as zero. Remaining nodes copy the input to the noised input.
- Minimizes the loss function between the output node and the corrupted input.
- Setting up a single-thread denoising autoencoder is easy.

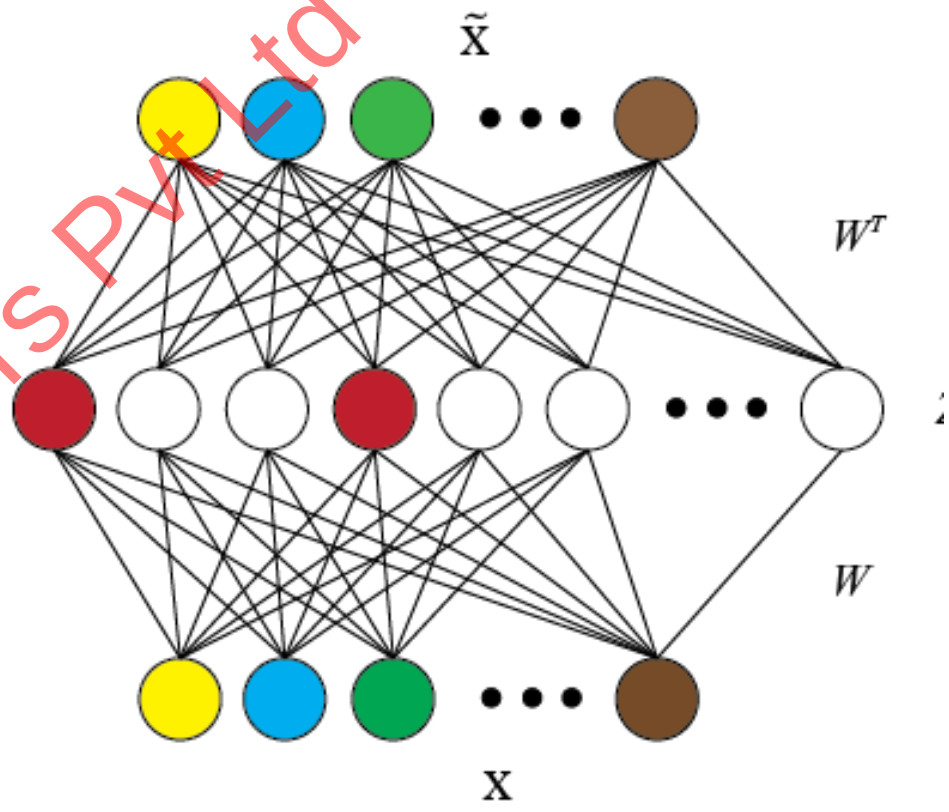


Sparse Autoencoder

- 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.

Sparse Autoencoder

- An advancement to sparse autoencoders is the k-sparse autoencoder. Here we choose k neurons with highest activation functions ignoring other activation functions using ReLU activation functions and adjusting the threshold to find the largest neurons. This tune the value of k to obtain sparsity level best suited for the dataset



Deep Autoencoder

- 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.

Contractive Autoencoder

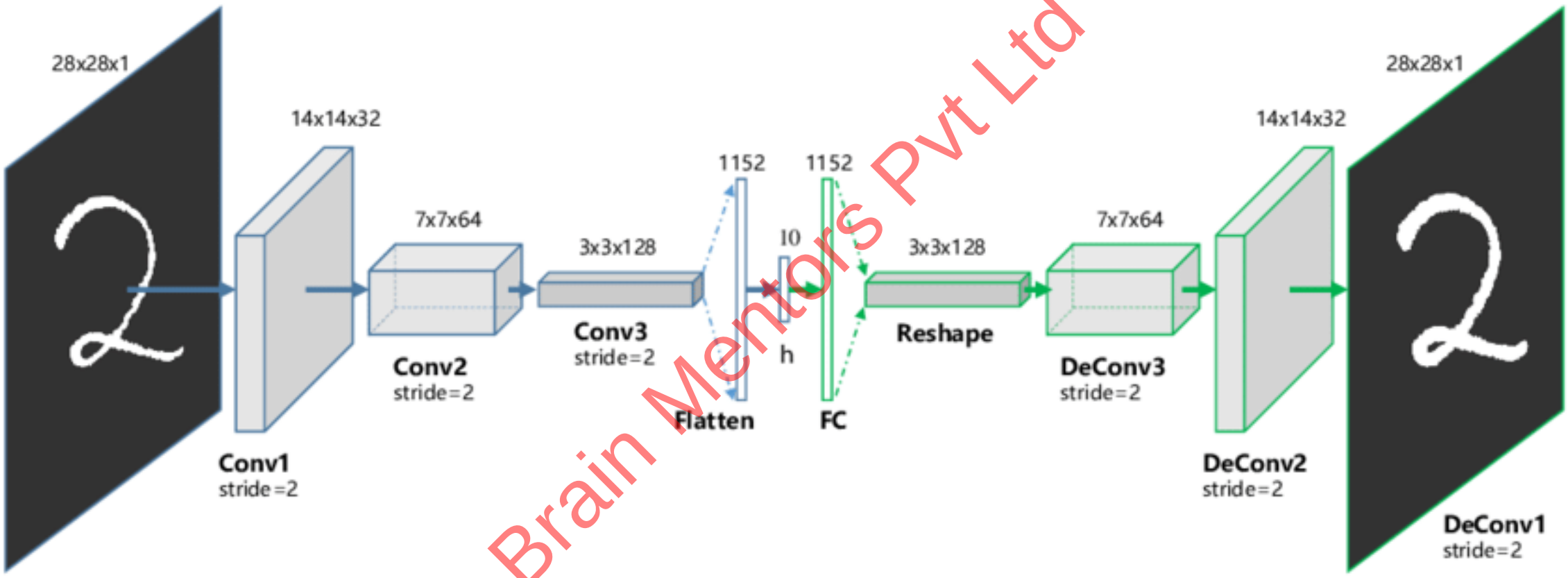
- The objective of a contractive autoencoder is to have a robust learned representation which is less sensitive to small variation in the data.
- Robustness of the representation for the data is done by applying a penalty term to the loss function. Contractive autoencoder is another regularization technique 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.

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

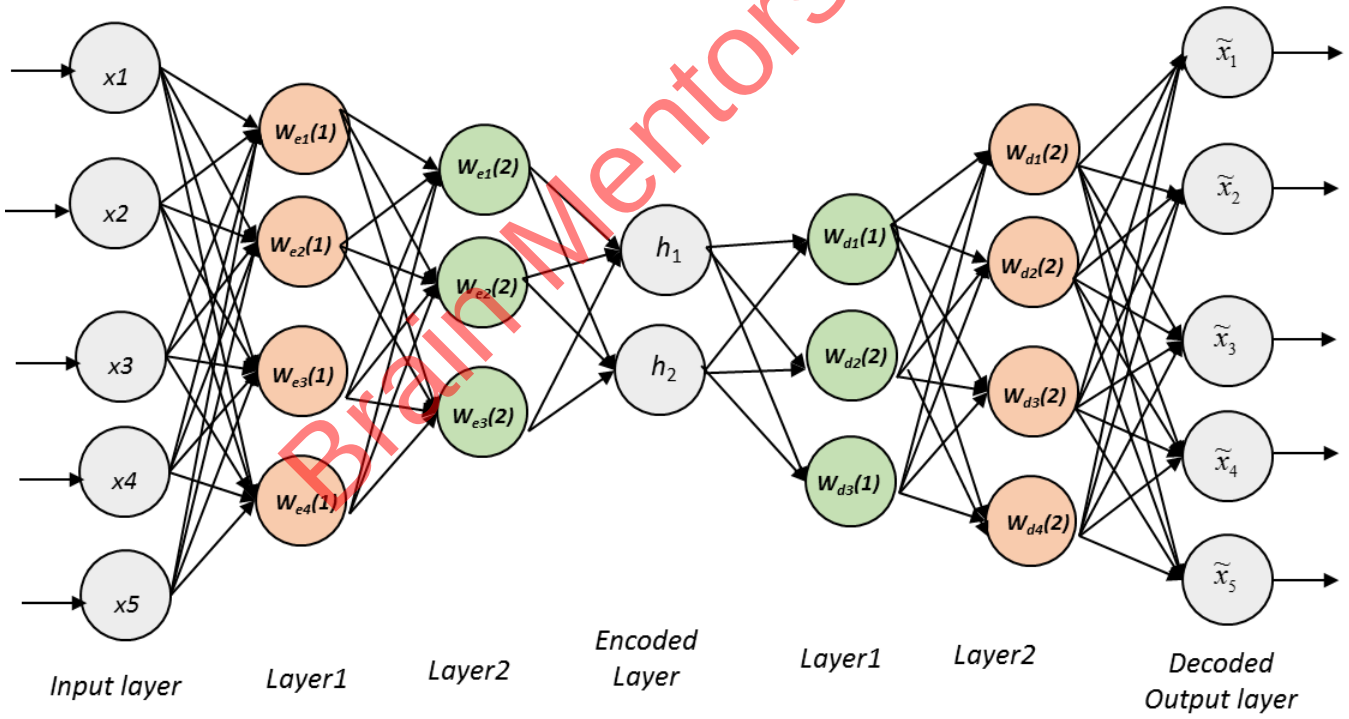
Convolutional Autoencoder

- 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.



Stacked Autoencoder

- A stacked autoencoder is a neural network consist several layers of sparse autoencoders where output of each hidden layer is connected to the input of the successive hidden layer.



Stacked Autoencoder

The hidden layers are trained by an unsupervised algorithm and then fine-tuned by a supervised method. Stacked autoencoder mainly consists of three steps

- Train autoencoder using input data and acquire the learned data.
- The learned data from the previous layer is used as an input for the next layer and this continues until the training is completed.
- Once all the hidden layers are trained use the backpropagation algorithm to minimize the cost function and weights are updated with the training set to achieve fine tuning.

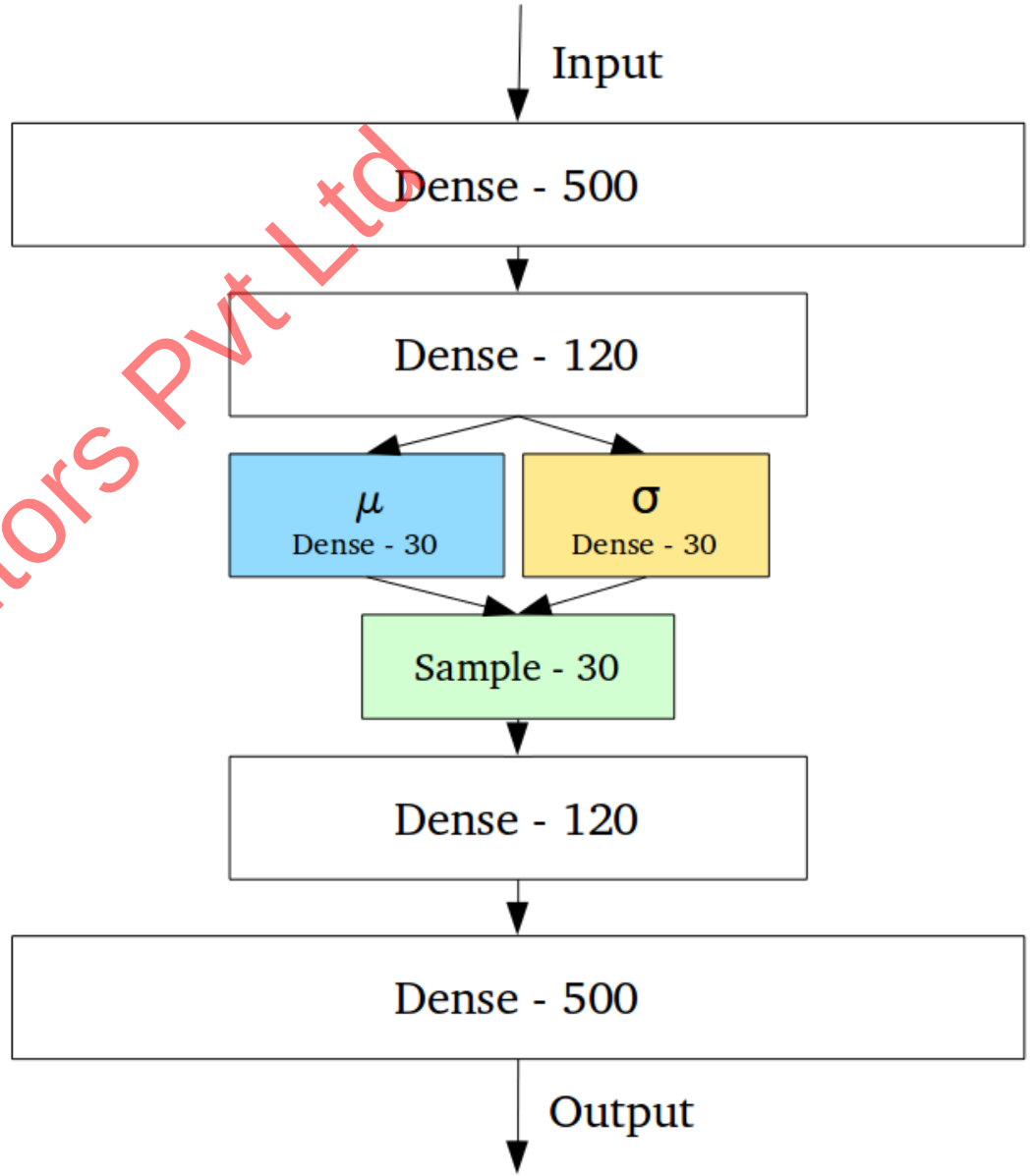
Stacked Autoencoder

- The recent advancements in Stacked Autoencoder is it provides a version of raw data with much detailed and promising feature information, which is used to train a classier with a specific context and find better accuracy than training with raw data.
- Stacked autoencoder improving accuracy in deep learning with noisy autoencoders embedded in the layers

Variational Autoencoder

- Variational Autoencoders (VAEs) have one fundamentally unique property that separates them from vanilla autoencoders, and it is this property that makes them so useful for generative modeling: their latent spaces are, *by design,* continuous, allowing easy random sampling and interpolation.
- It achieves this by doing something that seems rather surprising at first: making its encoder not output an encoding vector of size *n,* rather, outputting two vectors of size n: a vector of means**, μ**, and another vector of standard deviations, **σ**.

Variational Autoencoder



Variational Autoencoder

- Intuitively, the mean vector controls where the encoding of an input should be centered around, while the standard deviation controls the “area”, how much from the mean the encoding can vary.
- As encodings are generated at random from anywhere inside the “circle” (the distribution), the decoder learns that not only is a single point in latent space referring to a sample of that class, but all nearby points refer to the same as well.
- This allows the decoder to not just decode single, specific encodings in the latent space (leaving the decodable latent space discontinuous), but ones that slightly vary too, as the decoder is exposed to a range of variations of the encoding of the same input during training.

Variational Autoencoder

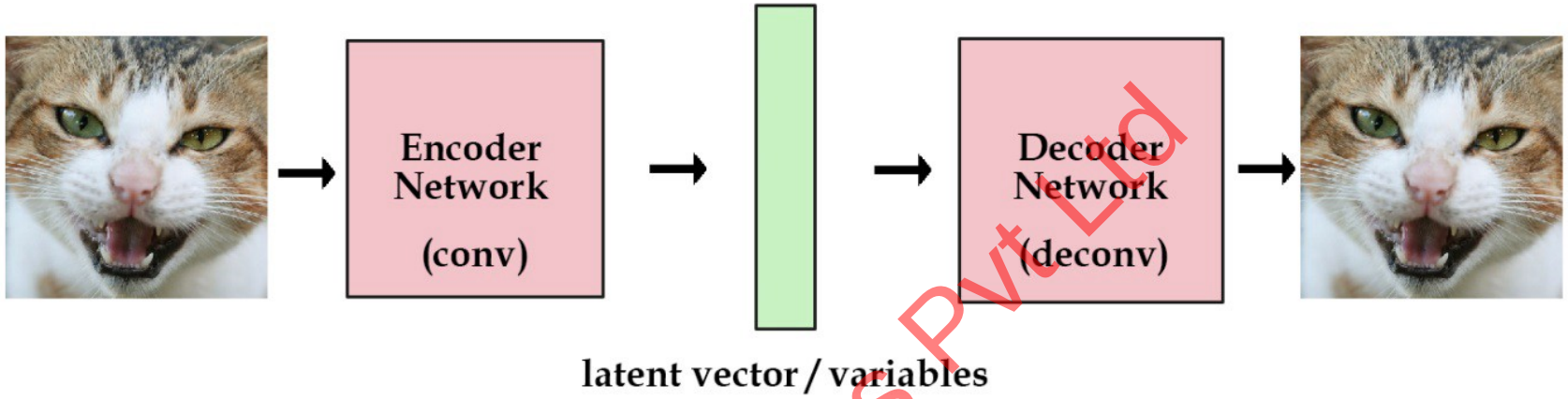
- The basic idea behind a variational autoencoder is that instead of mapping an input to fixed vector, input is mapped to a distribution. The only difference between the autoencoder and variational autoencoder is that bottleneck vector is replaced with two different vectors one representing the mean of the distribution and the other representing the standard deviation of the distribution.

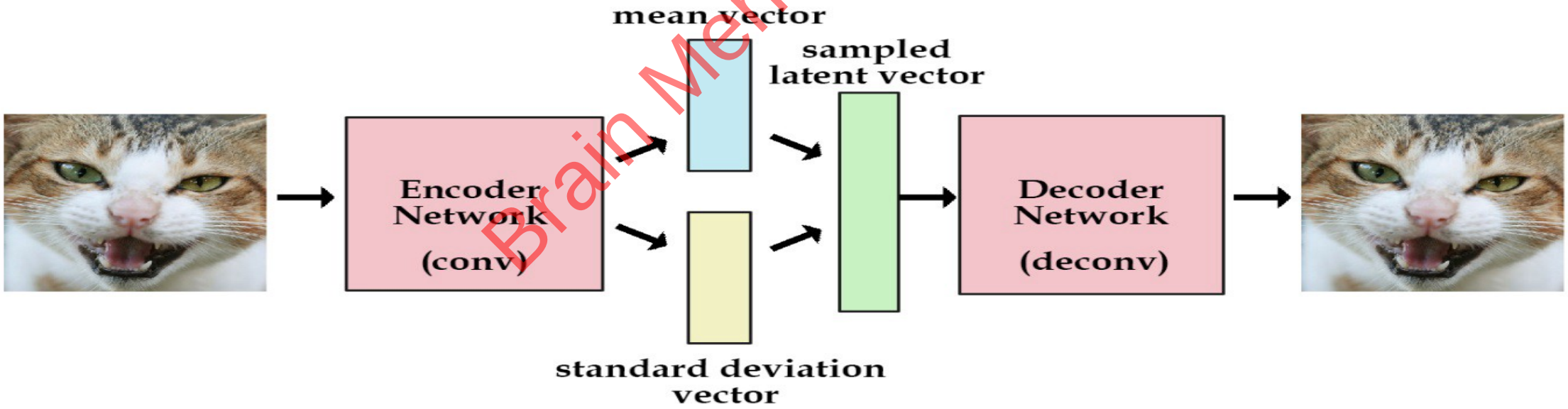
Loss Function

Loss function for variational autoencoder

$$li(\vartheta,\phi) = -Ez \sim q\vartheta\ (\ z|xi\ )\ [\ \log p\ \phi\ (\ xi|z\ )\ ] + KL\ (\ q\vartheta\ (\ z|xi\ )\ ||\ p(\ z\ )\ )$$

The loss function in variational autoencoder consists of two terms. First, one represents the reconstruction loss and the second term is a regularizer and KL means Kullback-Leibler divergence between the encoder's distribution qθ (z|x) and p (z). This divergence measures how much information is lost when using q to represent p.





Variational Autoencoder

- Recent advancements in VAE as mentioned in which improves the quality of VAE samples by adding two more components. Firstly, a pre-trained classifier as extractor to input data which aligns the reproduced images.
- Secondly, a discriminator network for additional adversarial loss signals.
- Other significant improvement in VAE is Optimization of the Latent Dependency Structure.
- In this VAE parameters, network parameters are optimized with a single objective.
- Interference is formed through sampling which produces expectations over latent variable structures and incorporates top-down and bottom-up reasoning over latent variable values.

Autoencoders Applications

- Today data denoising and dimensionality reduction for data visualization are the two major applications of autoencoders. With dimensionality and sparsity constraints, autoencoders can learn data projections which is better than PCA.
- Previously Autoencoders are used for dimensionality reduction or feature learning. In recent developments with connection with the latent variable models have brought autoencoders to forefront of the generative modelling.

Autoencoders Applications

- Autoencoders or its variants such as stacked, sparse or VAE are used for compact representation of data. For example a 256x256 pixel image can be represented by 28x28 pixel. Google is using this type of network to reduce the amount band width you use it on your phone. If you download an image the full resolution of the image is downscaled and then sent to you via wireless internet and then in your phone a decoder that reconstructs the image to full resolution.
- Autoencoders are used in Natural Language Processing, where NLP enclose some of the most difficult problems in computer science. With advancement in deep learning and indeed, autoencoders are been used to overcome some of these problems

Autoencoders Applications

- Document Clustering: classification of documents such as blogs or news or any data into recommended categories. The challenge is to accurately cluster the documents into categories where there actually fit. Hinton used autoencoder to reduce the dimensionality vectors to represent the word probabilities in newswire stories.
- Machine translation: it has been studied since late 1950s and an incredibly a difficult problem to translate text from one human language to another human language. With the use of autoencoders machine translation has taken a huge leap forward to accurately translate text from one language to another.

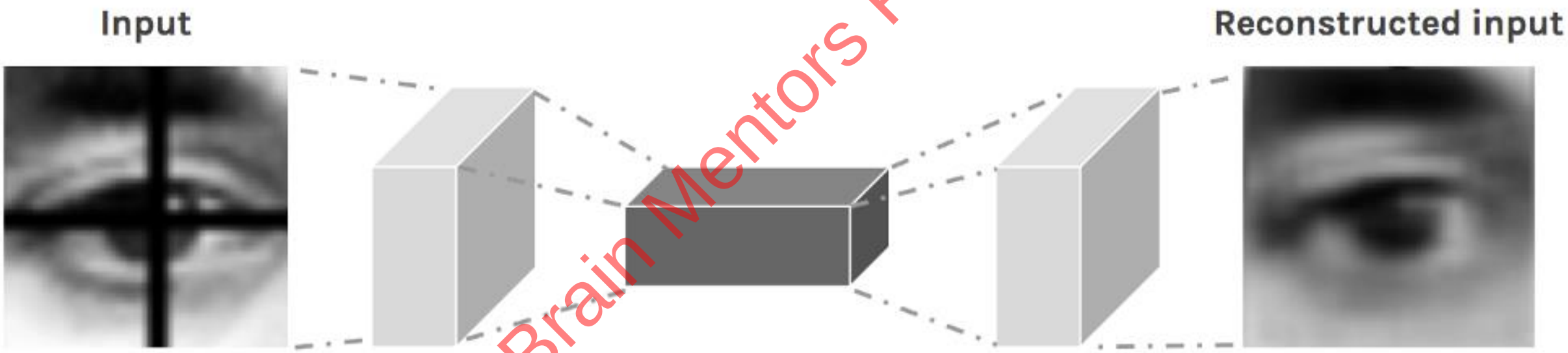
Autoencoders Applications

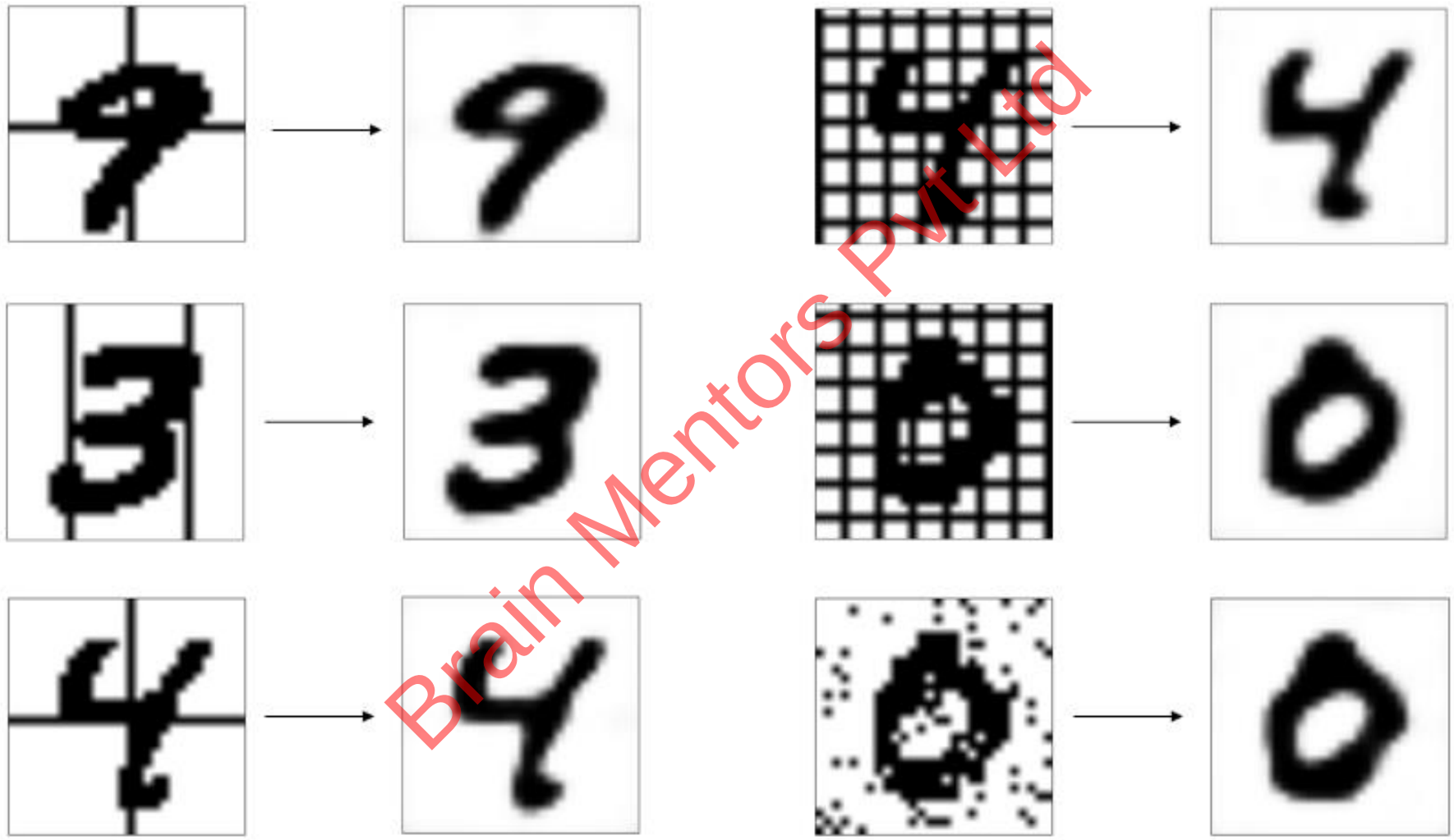
- Autoencoders to extract speech: A deep generative model of spectrograms containing 256 frequency bins and 1,3,9 or 13 frames has been created . This model has one visible layer and one hidden layer of 500 to 3000 binary latent variables.
- With Deep Denoising Autoencoders(DDAE) which has shown drastic improvement in performance has the capability to recognize the whispered speech which has been a problem for a long time in Automatic Speech Recognition(ASR). This has been implemented in various smart devices such as Amazon Alexa.

Autoencoders Applications

- Reconstruction image using Convolutional Autoencoders: CAE are useful in reconstruction of image from missing parts. For this the model has to be trained with two different images as input and output. The input image can rather be a noisy version or an image with missing parts and with a clean output image. During training process the model learns and fills the gaps in the input and output images. Here is an example below how CAE replace the missing part of the image.

Autoencoders Applications





Autoencoders Applications

- Paraphrase Detection: in many languages two phrases may look differently but when it comes to the meaning they both mean exactly same. Deep learning autoencoders allow us to find such phrases accurately.
- Many other advanced applications includes full image colorization, generating higher resolution images by using lower resolution as input.

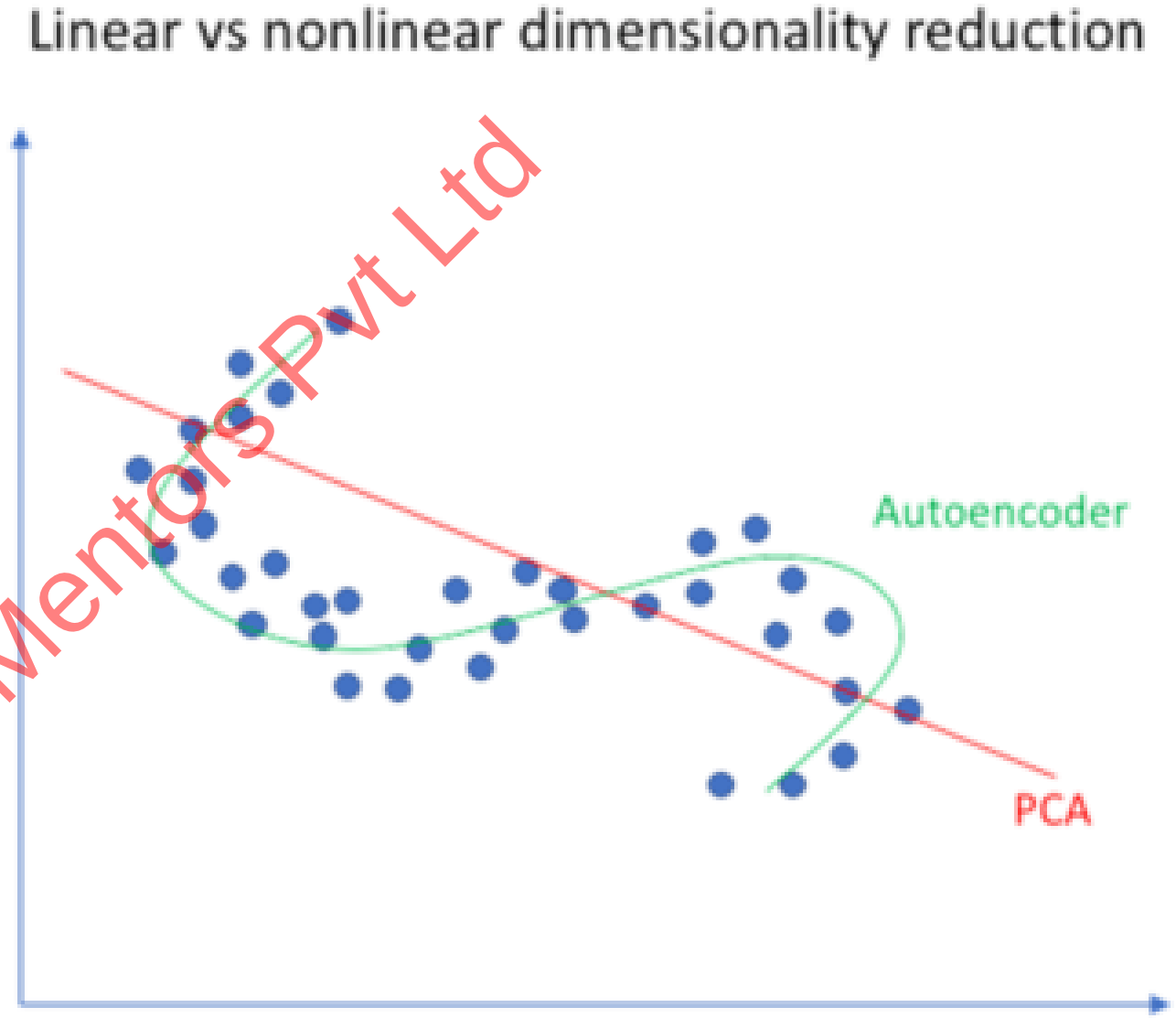
Autoencoders Applications

Autoencoder Over PCA

Autoencoders are preferred over PCA because:

- 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.

Autoencoder Over PCA



Working of Autoencoder

We buy a service or an item on the internet. we ensure that the site is secure by checking they use https protocol. We enter our credit card details for the purchase. Our credit card details are encoded over the network using some encoding algorithm. Encoded credit card detail is decoded to generate the original credit card number for validation.

In our credit card example, we took the credit card details, encoded it using some function. Later decoded it using another function to reproduce the output identical to the input. This is how autoencoders work.

Working of Autoencoder

- **Autoencoders encodes the input values *x*, using a function *f*. It then decodes the encoded values *f(x)*, using a function *g*, to create output values identical to the input values.**
- **Autoencoder ‘s objective is to minimize reconstruction error between the input and output. This helps autoencoders to learn important features present in the data. When a representation allows a good reconstruction of its input, then it has retained much of the information present in the input.**

Working of Autoencoder

We take the input, encode it to identify latent feature representation. Decode the latent feature representation to recreate the input. We calculate the loss by comparing the input and output. To reduce the reconstruction error we back propagate and update the weights. Weight is updated based on how much they are responsible for the error.

In this example, we have taken the dataset for products bought by customers

Working of Autoencoder

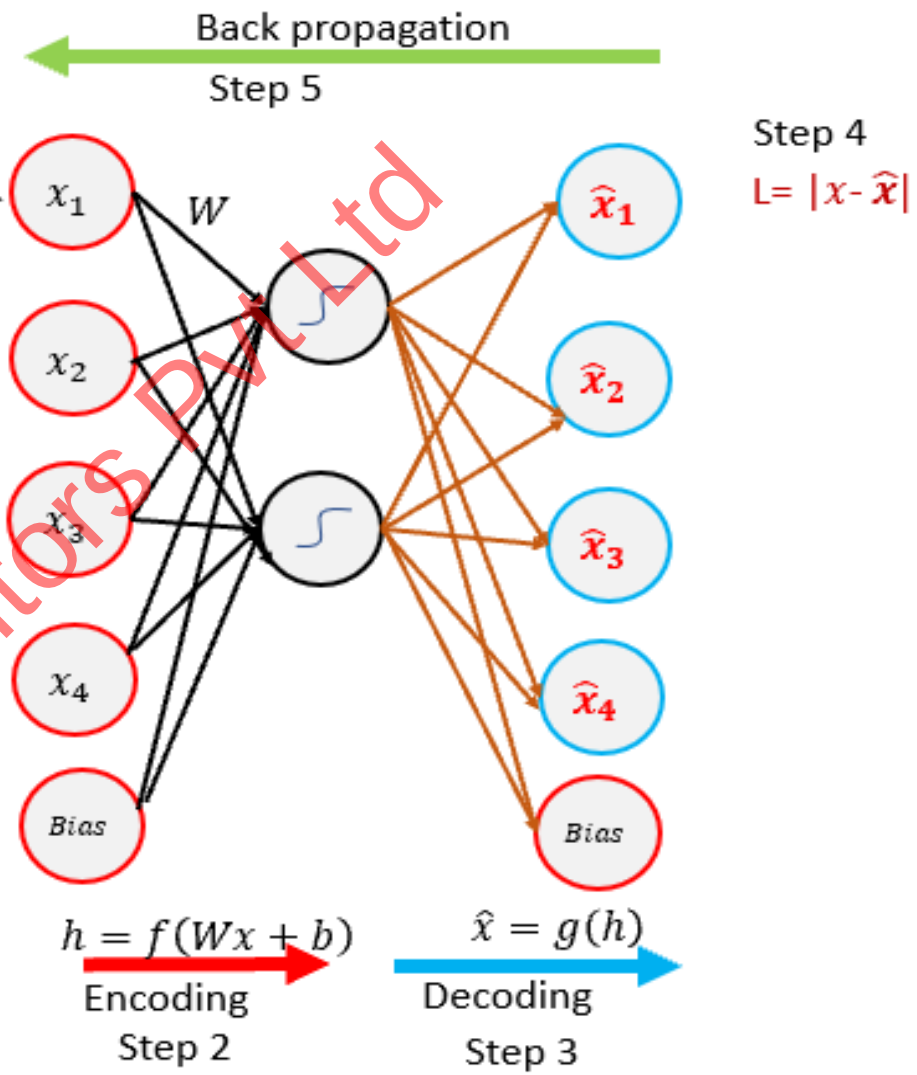
Step 1: Take the first row from the customer data for all products bought in an array as the input. 1 represent that the customer bought the product. 0 represents that the customer did not buy the product.

Step 2: Encode the input into another vector *h*. *h* is a lower dimension vector than than the input. We can use sigmoid activation function for h as the it ranges from 0 to 1. W is the weight applied to the input and b is the bias term.

$$h=f(Wx+b)$$

	Product1	Product2	Product3	Product4	Product5	Product6
Customer1	1	1	0	0	0	0
Customer2	1	0	0	0	1	0
Customer3	0	1	1	1	0	0
Customer4	1	1	0	0	1	0
Customer5	0	0	0	1	0	1

Step 1



Step 4

L= $|x - \hat{x}|$

Working of Autoencoder

Step 3: Decode the vector *h* to recreate the input. Output will be of same dimension as the input.

Step 4 : Calculate the reconstruction error L. Reconstruction error is the difference between the input and output vector. Our goal is to minimize the reconstruction error so that output is similar to the input vector.

Reconstruction error= input vector — output vector

Step 5: Back propagate the error from output layer to the input layer to update the weights. Weights are updated based on how much they were responsible for the error.