AMRITA VISHWA VIDYAPEETHAM

DEEMED TO BE UNIVERSITY

Introduction to Deep Learning

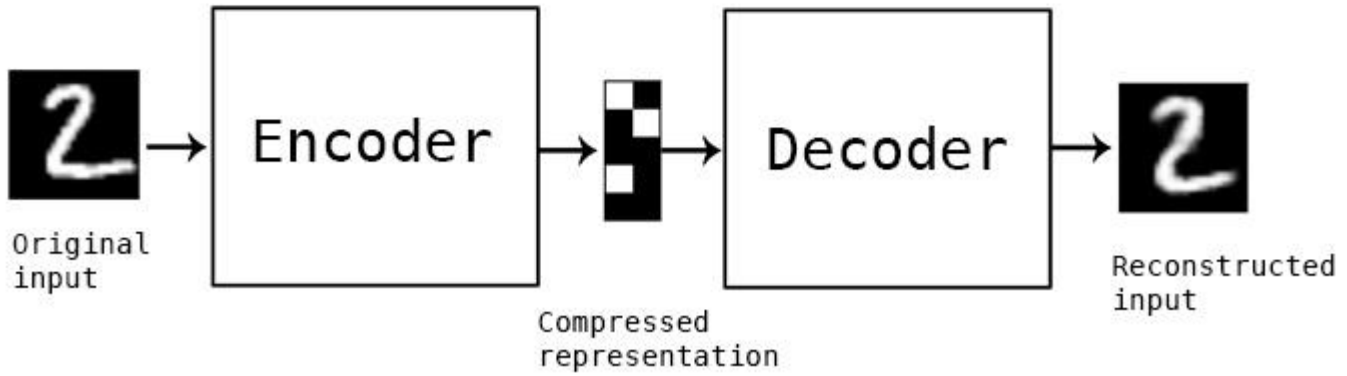
AutoEncoders

Courtesy:, analytics vidya,fast.ai, coursera: AndrewNG,

What is an Autoencoder?

Autoencoders are neural network-based models that are used for unsupervised learning purposes to discover underlying correlations among data and represent data in a smaller dimension. The autoencoders frame unsupervised learning problems as supervised learning problems to train a neural network model.

Dimensionality Reduction



latent feature space

The reason behind using an autoencoder is that we want to understand and represent only the deep correlations and relationships among data

The input is squeezed down to a lower encoded representation using an encoder network, then a decoder network decodes the encoding to recreate back the input.

The encoding produced by the encoder layer has a lower-dimensional representation of the data and shows several interesting complex relationships among data.

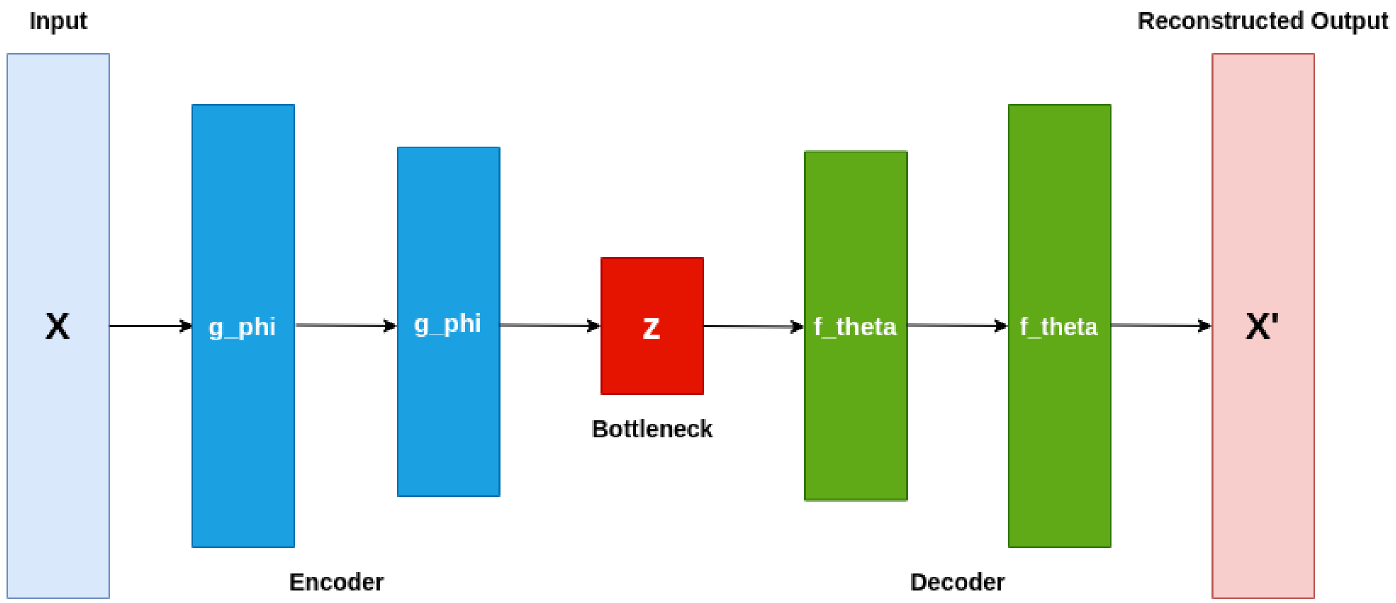
Auto-encoder- Parts

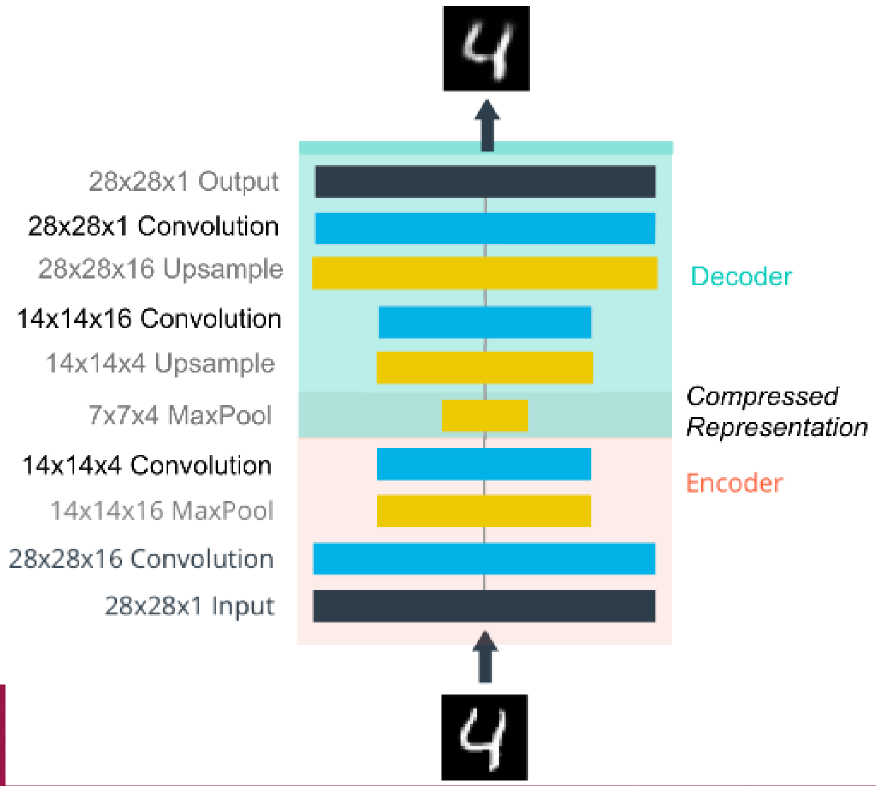
An Autoencoder has the following parts:

1.Encoder: The encoder is the part of the network which takes in the input and produces a lower Dimensional encoding

2.Bottleneck: It is the lower dimensional hidden layer where the encoding is produced. The bottleneck layer has a lower number of nodes and the number of nodes in the bottleneck layer also gives the dimension of the encoding of the input.

3.Decoder: The decoder takes in the encoding and recreates back the input.

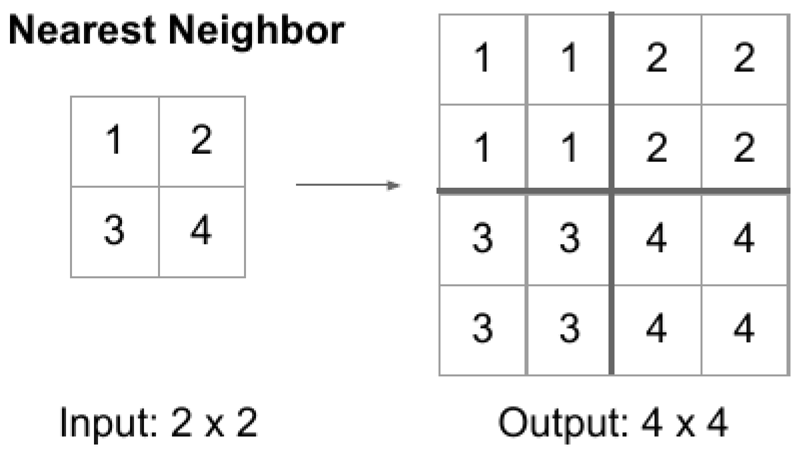


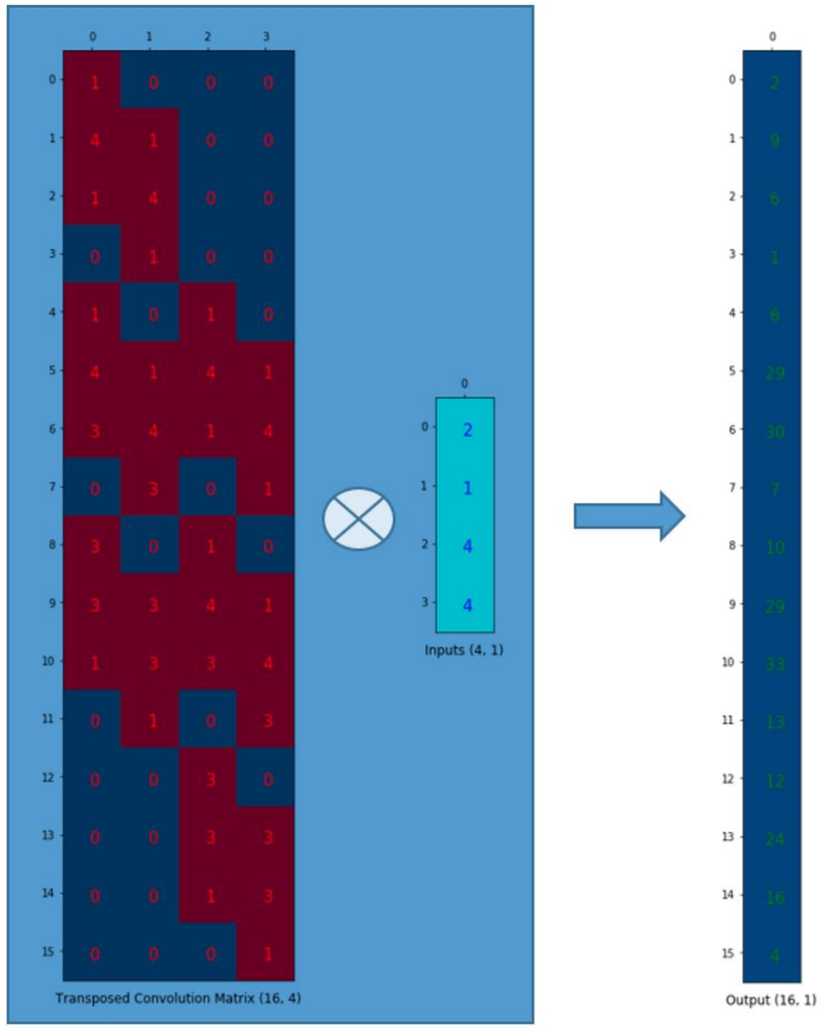


Courtesy : https://towardsdatascience.com/

Upsampling

Upsample the compressed image is by **Unpooling** *(the reverse of pooling)* using Nearest Neighbor or by max unpooling.

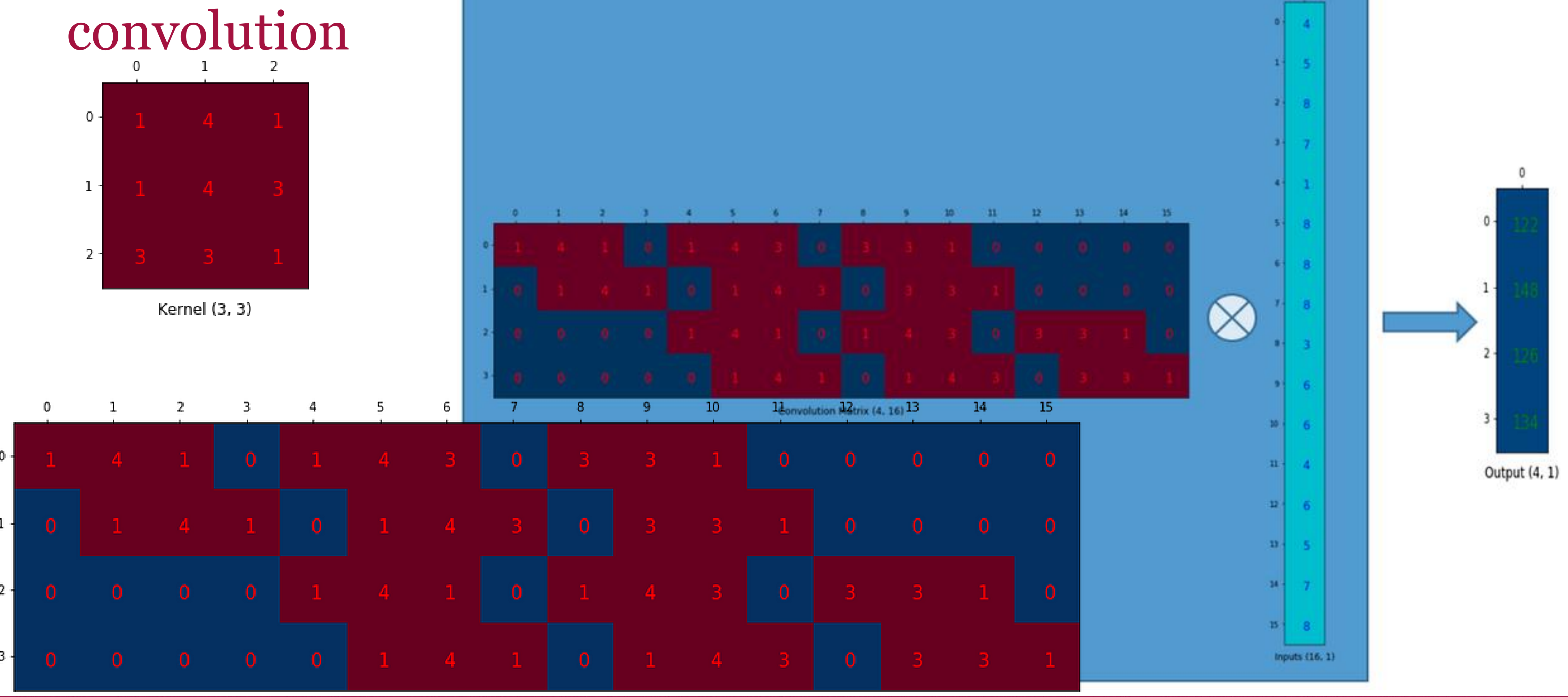




Another way is to use **transposed convolution**. (De-convolution)The transpose convolution is reverse of the convolution operation.

Convolution Matrix

We can express a convolution operation using a matrix. It is nothing but a kernel matrix rearranged so that we can use a matrix multiplication to conduct convolution operations.



	0	1	2
0	1	4	1
1	1	4	3
2	3	3	1

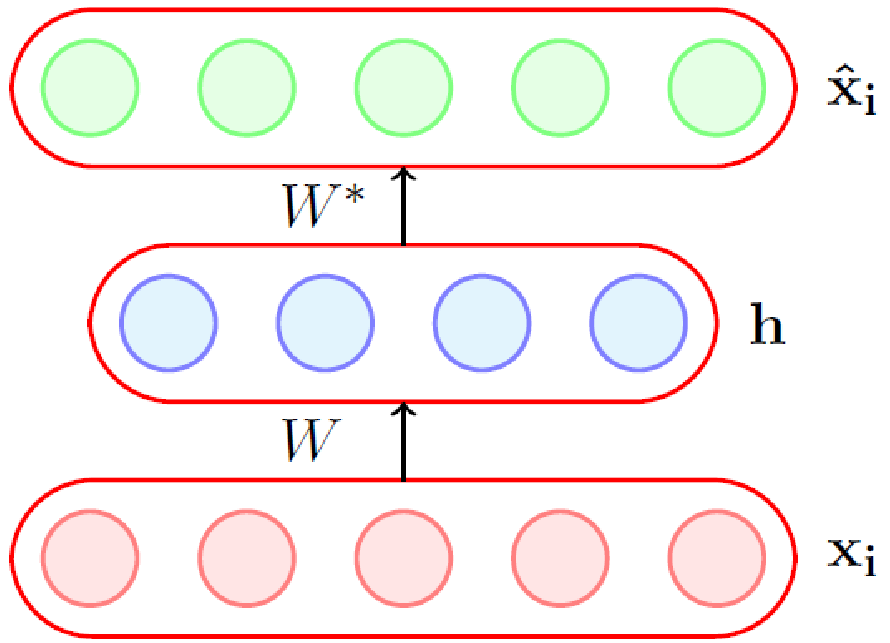
Kernel (3, 3)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	4	1	0	1	4	3	0	3	3	1	0	0	0	0	0
1	0	1	4	1	0	1	4	3	0	3	3	1	0	0	0	0
2	0	0	0	0	1	4	1	0	1	4	3	0	3	3	1	0
3	0	0	0	0	0	1	4	1	0	1	4	3	0	3	3	1

Convolution Matrix (4, 16)

latent features

Autoencoder

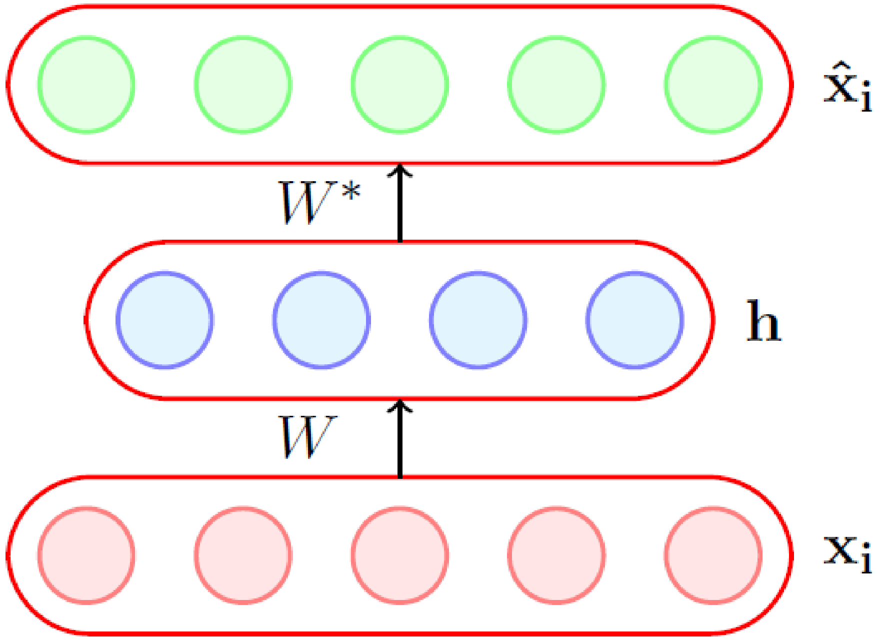


$$\mathbf{h} = g(W\mathbf{x}_\mathbf{i} + \mathbf{b})$$
$$\hat{\mathbf{x}}_\mathbf{i} = f(W^*\mathbf{h} + \mathbf{c})$$

- An autoencoder is a special type of feed forward neural network which does the following
- Encodes its input $\mathbf{x}_\mathbf{i}$ into a hidden representation $\mathbf{h}$
- Decodes the input again from this hidden representation
- The model is trained to minimize a certain loss function which will ensure that $\hat{\mathbf{x}}_\mathbf{i}$ is close to $\mathbf{x}_\mathbf{i}$ (we will see some such loss functions soon)

Autoencoder

latent features



$$\mathbf{h} = g(W\mathbf{x}_\mathbf{i} + \mathbf{b})$$
$$\hat{\mathbf{x}}_\mathbf{i} = f(W^*\mathbf{h} + \mathbf{c})$$

- Consider the case when the inputs are real valued
- The objective of the autoencoder is to reconstruct $\hat{\mathbf{x}}_\mathbf{i}$ to be as close to $\mathbf{x}_\mathbf{i}$ as possible
- This can be formalized using the following objective function:

$$\min_{W,W^*,\mathbf{c},\mathbf{b}} \frac{1}{m}\sum_{i=1}^{m}\sum_{j=1}^{n}(\hat{x}_{ij} - x_{ij})^2$$

$$i.e., \min_{W,W^*,\mathbf{c},\mathbf{b}} \frac{1}{m}\sum_{i=1}^{m}(\hat{\mathbf{x}}_\mathbf{i} - \mathbf{x}_\mathbf{i})^T(\hat{\mathbf{x}}_\mathbf{i} - \mathbf{x}_\mathbf{i})$$

- We can then train the autoencoder just like a regular feedforward network using back-propagation

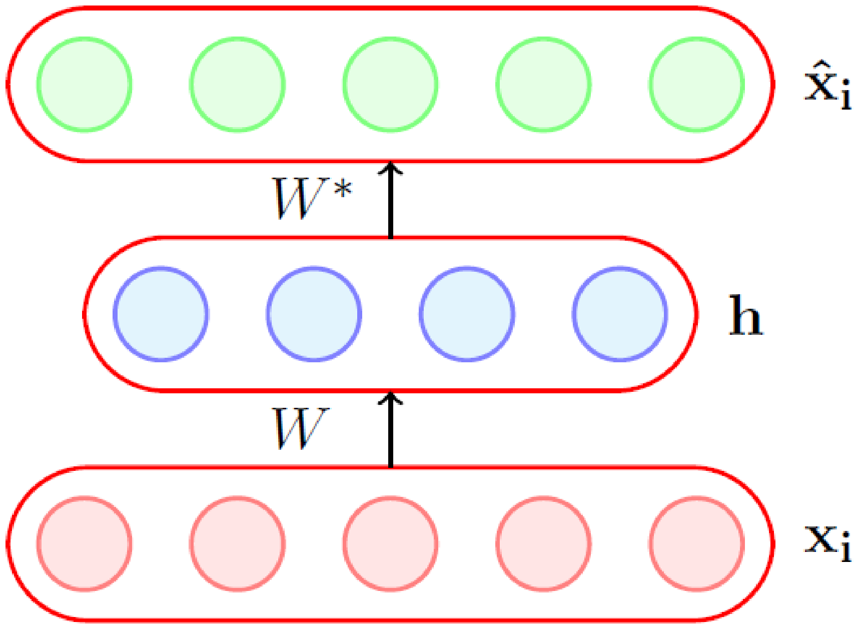
Types of Autoencoders

- Over Complete Autoencoder
- Under Complete Autoencoder
- Denoising Autoencoder
- Sparse Autoencoder
- Variational Autoencoder

Variations of autoencoders based on the dimensionality of their latent (bottleneck) layer compared to the input layer.

Undercomplete Autoencoder:

- an undercomplete autoencoder has a lower dimensionality in the latent layer compared to the input layer.
- The number of neurons in the latent layer is smaller than the number of neurons in the input layer.
- By limiting the capacity of the latent layer, undercomplete autoencoders are encouraged to learn a compressed representation of the input data, capturing the most salient features.
- Undercomplete autoencoders are often used for dimensionality reduction, feature extraction, and anomaly detection tasks.



if we use a shallow network with a very less number of nodes, it will be very hard to capture all the relationships.

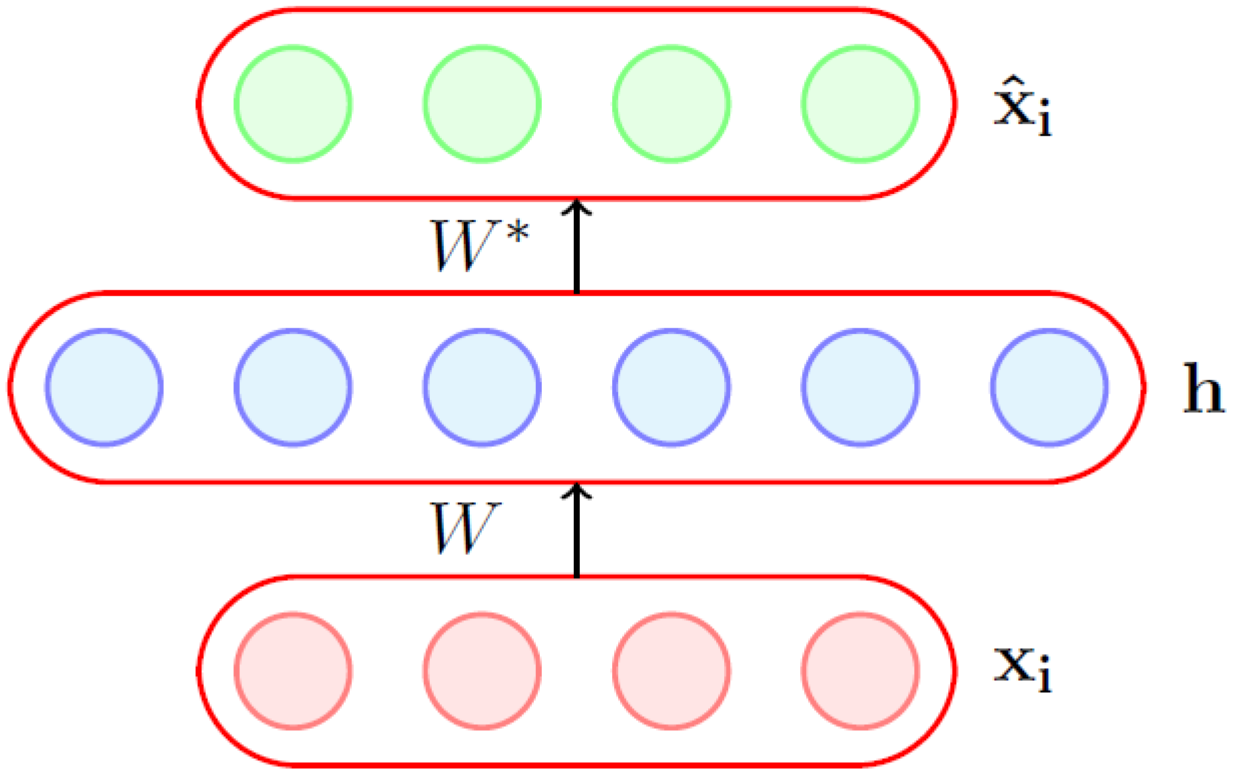
$$\dim(\mathbf{h}) < \dim(\mathbf{x_i})$$

Variations of autoencoders based on the dimensionality of their latent (bottleneck) layer compared to the input layer.

Larger Number of Parameters : Overfit

Overcomplete Autoencoder:

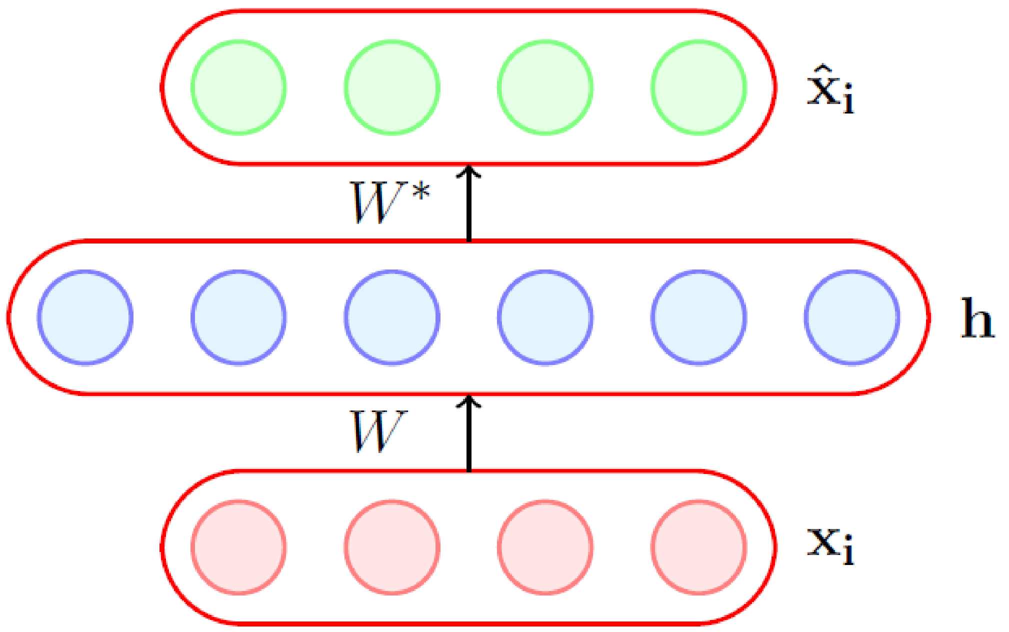
- In an overcomplete autoencoder, the dimensionality of the latent layer is higher than the dimensionality of the input layer.
- This means that the number of neurons in the latent layer is larger than the number of neurons in the input layer.
- Overcomplete autoencoders have more parameters and capacity to learn than the input data, which can potentially lead to overfitting.
- However, they can still be useful in certain scenarios, such as feature extraction, where the autoencoder is forced to learn a representation of the input data.



$$\dim(\mathbf{h}) \geq \dim(\mathbf{x}_\mathbf{i})$$

Network might cheat and overfit to the input data by simply remembering the input data. In this case, we will not be able to get the correct relationships in our encodings.

Regularisation of Autoencoders

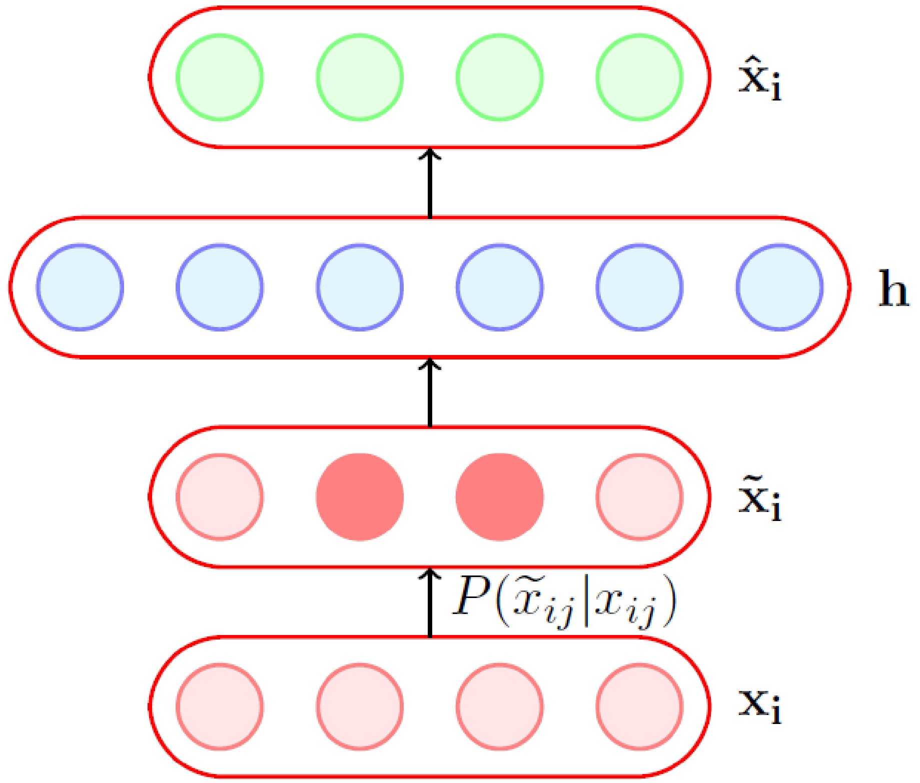


- While poor generalization could happen even in undercomplete autoencoders it is an even more serious problem for overcomplete auto encoders
- Here, (as stated earlier) the model can simply learn to copy $\mathbf{x_i}$ to $\mathbf{h}$ and then $\mathbf{h}$ to $\hat{\mathbf{x}}_{\mathbf{i}}$
- To avoid poor generalization, we need to introduce regularization

- The simplest solution is to add a L_2-regularization term to the objective function

$$\min_{\theta, w, w^*, \mathbf{b}, \mathbf{c}} \frac{1}{m} \sum_{i=1}^{m} \sum_{j=1}^{n} (\hat{x}_{ij} - x_{ij})^2 + \lambda \|\theta\|^2$$

Denoising AutoEncoder

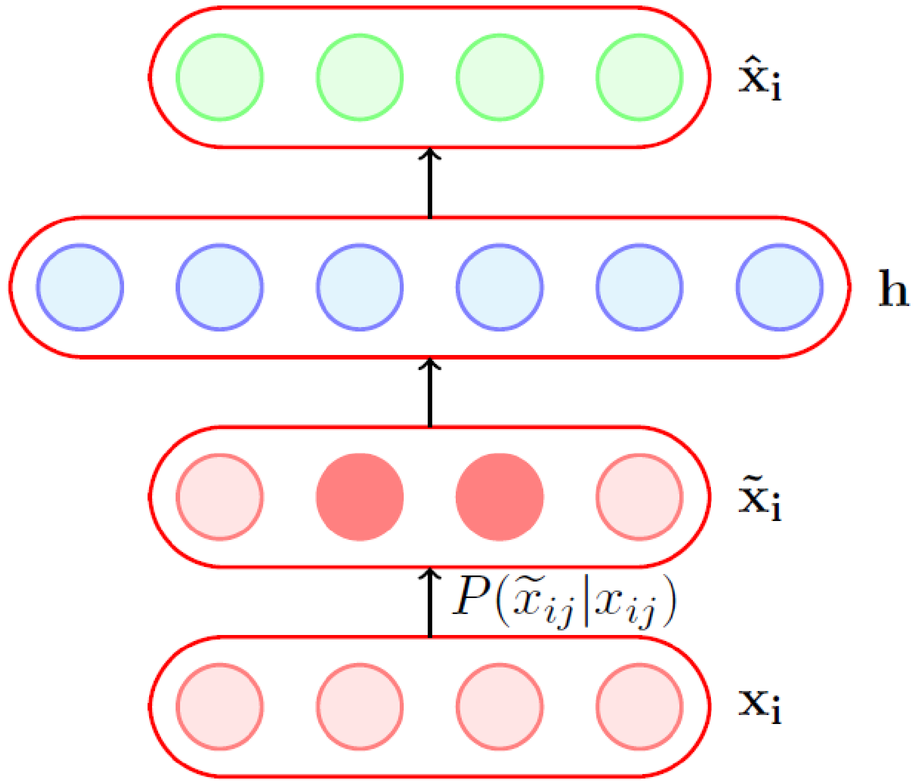

$\hat{\mathbf{x}}_\mathbf{i}$
$\mathbf{h}$
$\tilde{\mathbf{x}}_\mathbf{i}$
$P(\widetilde{x}_{ij}|x_{ij})$
$\mathbf{x}_\mathbf{i}$

- A denoising encoder simply corrupts the input data using a probabilistic process ($P(\widetilde{x}_{ij}|x_{ij})$) before feeding it to the network
- A simple $P(\widetilde{x}_{ij}|x_{ij})$ used in practice is the following

$$P(\widetilde{x}_{ij} = 0|x_{ij}) = q$$
$$P(\widetilde{x}_{ij} = x_{ij}|x_{ij}) = 1 - q$$

- In other words, with probability q the input is flipped to 0 and with probability $(1 - q)$ it is retained as it is

Denoising AutoEncoder



- This helps because the objective is still to reconstruct the original (uncorrupted) $\mathbf{x}_i$

$$\arg\min_{\theta} \frac{1}{m} \sum_{i=1}^{m} \sum_{j=1}^{n} (\hat{x}_{ij} - x_{ij})^2$$

- It no longer makes sense for the model to copy the corrupted $\widetilde{\mathbf{x}}_i$ into $h(\widetilde{\mathbf{x}}_i)$ and then into $\hat{\mathbf{x}}_i$ (the objective function will not be minimized by doing so)
- Instead the model will now have to capture the characteristics of the data correctly.

Application

Task: Hand-written digit recognition

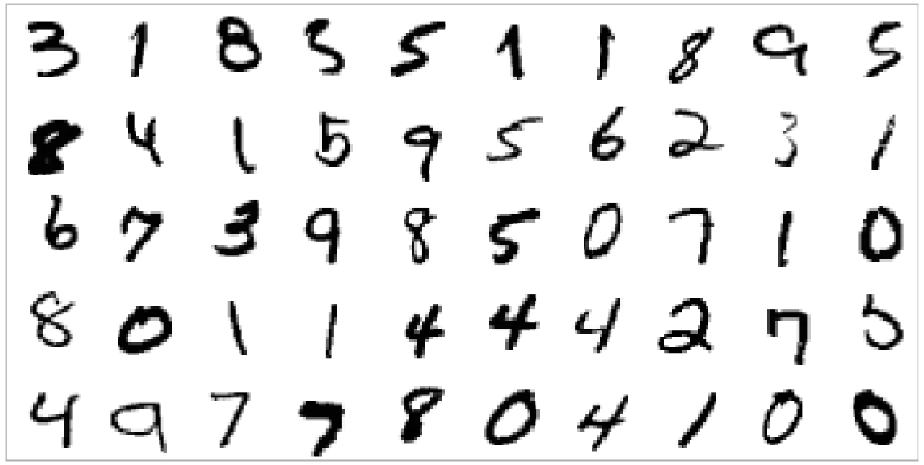


Figure: MNIST Data

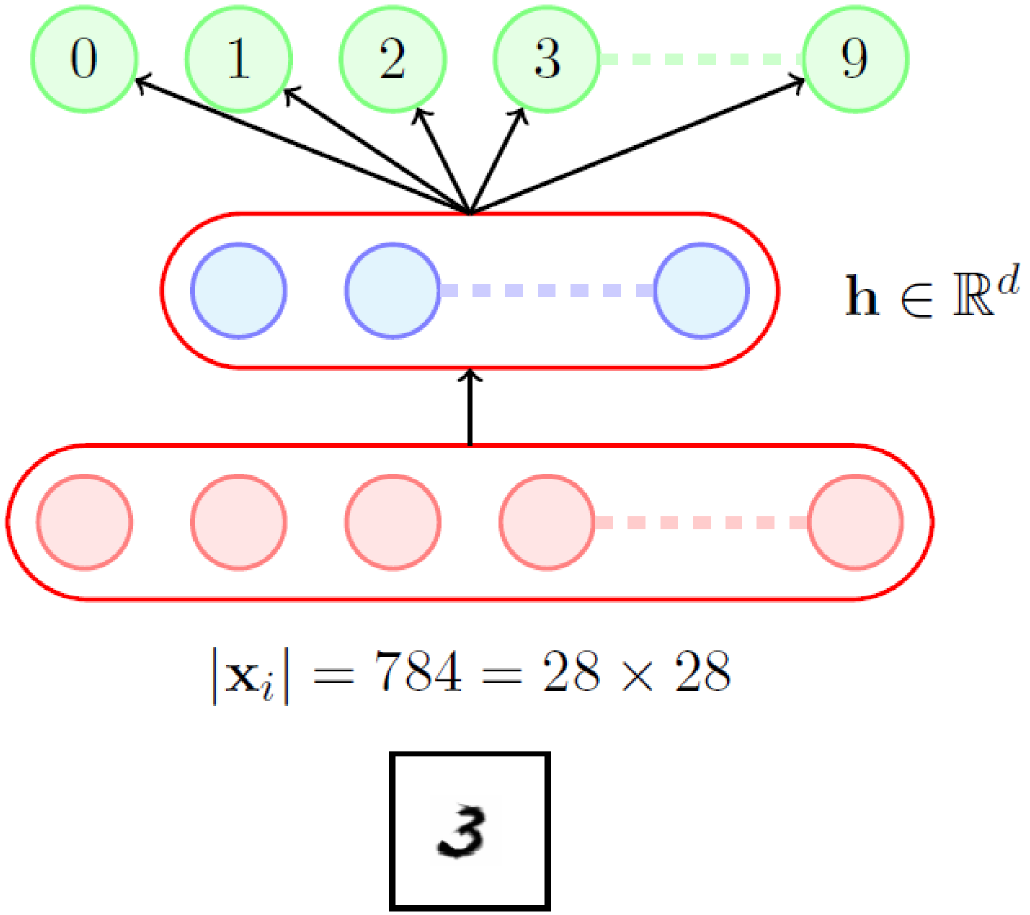
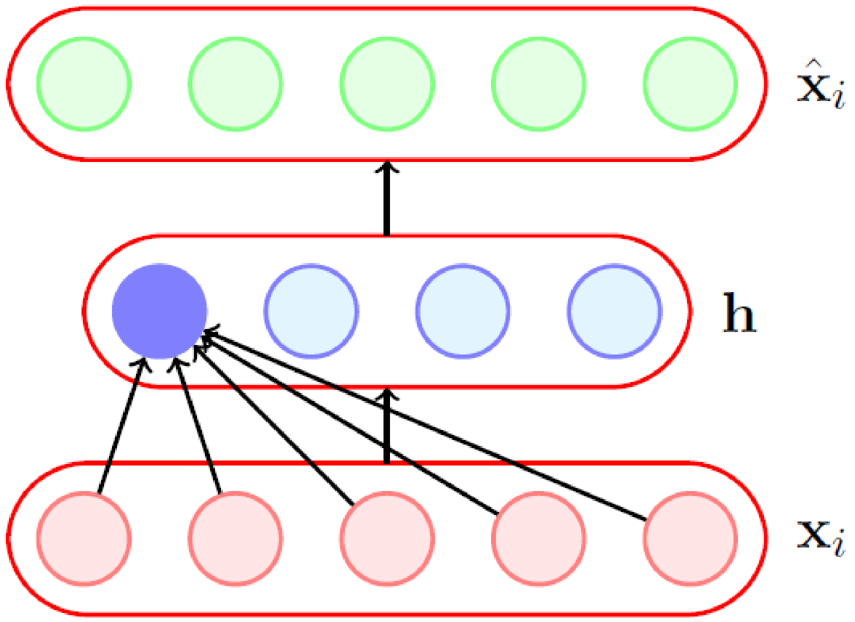

0
1
2
3
9
$\mathbf{h} \in \mathbb{R}^d$
$|\mathbf{x}_i| = 784 = 28 \times 28$
3

Figure: AE approach (and then train a classifier on top of this hidden representation)



$$\max_{\mathbf{x}_i} \quad \{W_1^T \mathbf{x}_i\}$$

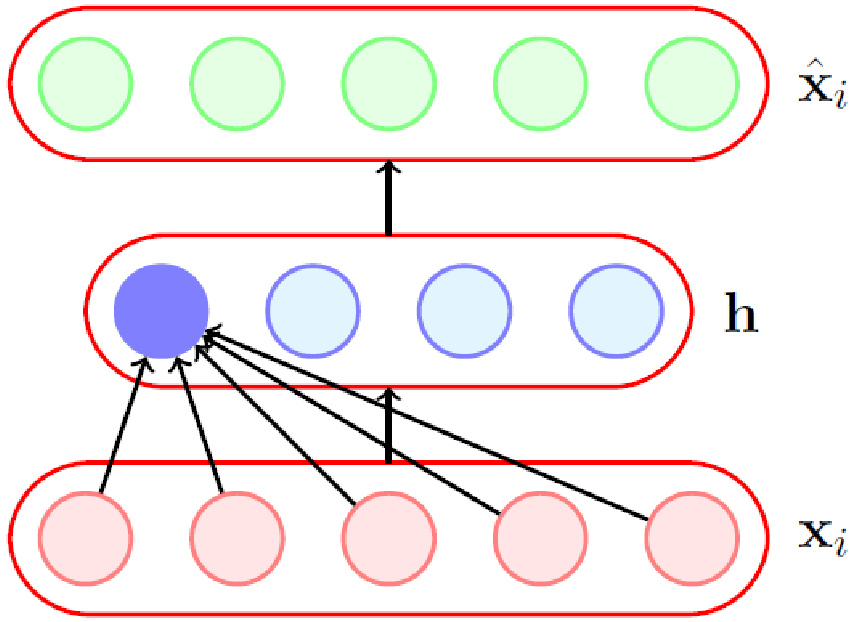
$$s.t. \quad ||\mathbf{x}_i||^2 = \mathbf{x}_i^T \mathbf{x}_i = 1$$

$$\text{Solution:} \quad \mathbf{x}_i = \frac{W_1}{\sqrt{W_1^T W_1}}$$

- We can think of each neuron as a filter which will fire (or get maximally) activated for a certain input configuration $\mathbf{x}_i$
- For example,

$$\mathbf{h}_1 = \sigma(W_1^T \mathbf{x}_i) \ [ignoring \ bias \ b]$$

  Where W_1 is the trained vector of weights connecting the input to the first hidden neuron
- What values of $\mathbf{x}_i$ will cause $\mathbf{h}_1$ to be maximum (or maximally activated)
- Suppose we assume that our inputs are normalized so that $\|\mathbf{x}_i\| = 1$



$$\max_{\mathbf{x}_i} \quad \{W_1^T \mathbf{x}_i\}$$

$$s.t. \quad ||\mathbf{x}_i||^2 = \mathbf{x}_i^T \mathbf{x}_i = 1$$

$$\text{Solution:} \quad \mathbf{x}_i = \frac{W_1}{\sqrt{W_1^T W_1}}$$

- Thus the inputs

$$\mathbf{x}_i = \frac{W_1}{\sqrt{W_1^T W_1}}, \frac{W_2}{\sqrt{W_2^T W_2}}, \ldots \frac{W_n}{\sqrt{W_n^T W_n}}$$

  will respectively cause hidden neurons 1 to n to maximally fire
- Let us plot these images ($\mathbf{x}_i$'s) which maximally activate the first k neurons of the hidden representations learned by a vanilla autoencoder and different denoising autoencoders
- These $\mathbf{x}_i$'s are computed by the above formula using the weights $(W_1, W_2 \ldots W_k)$ learned by the respective autoencoders

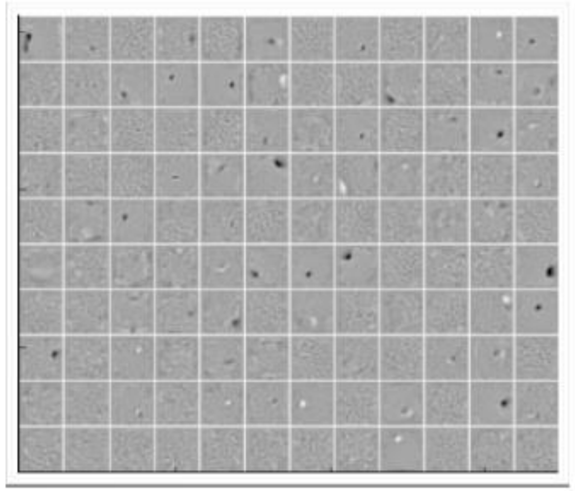


Figure: Vanilla AE (No noise)

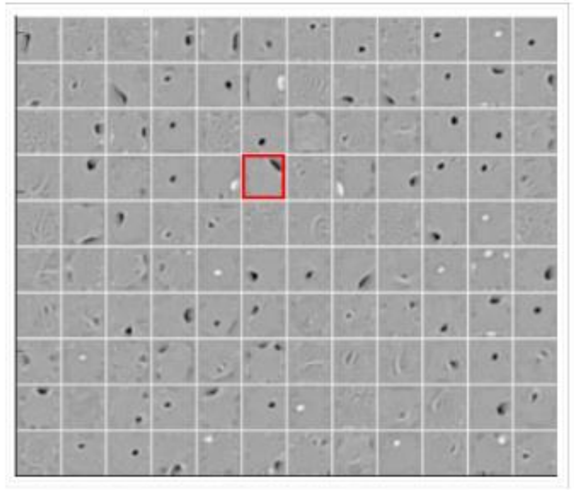


Figure: 25% Denoising AE (q=0.25)

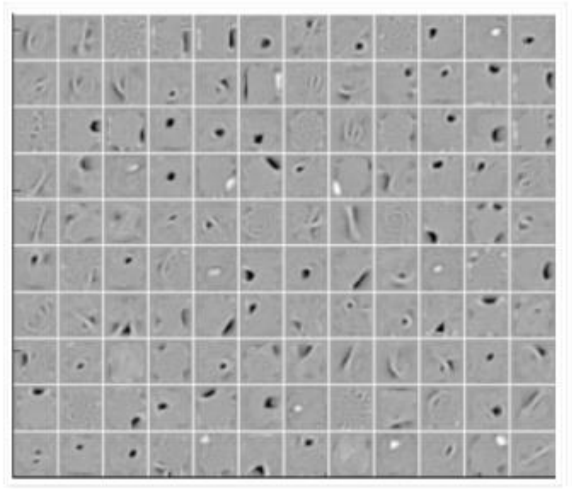
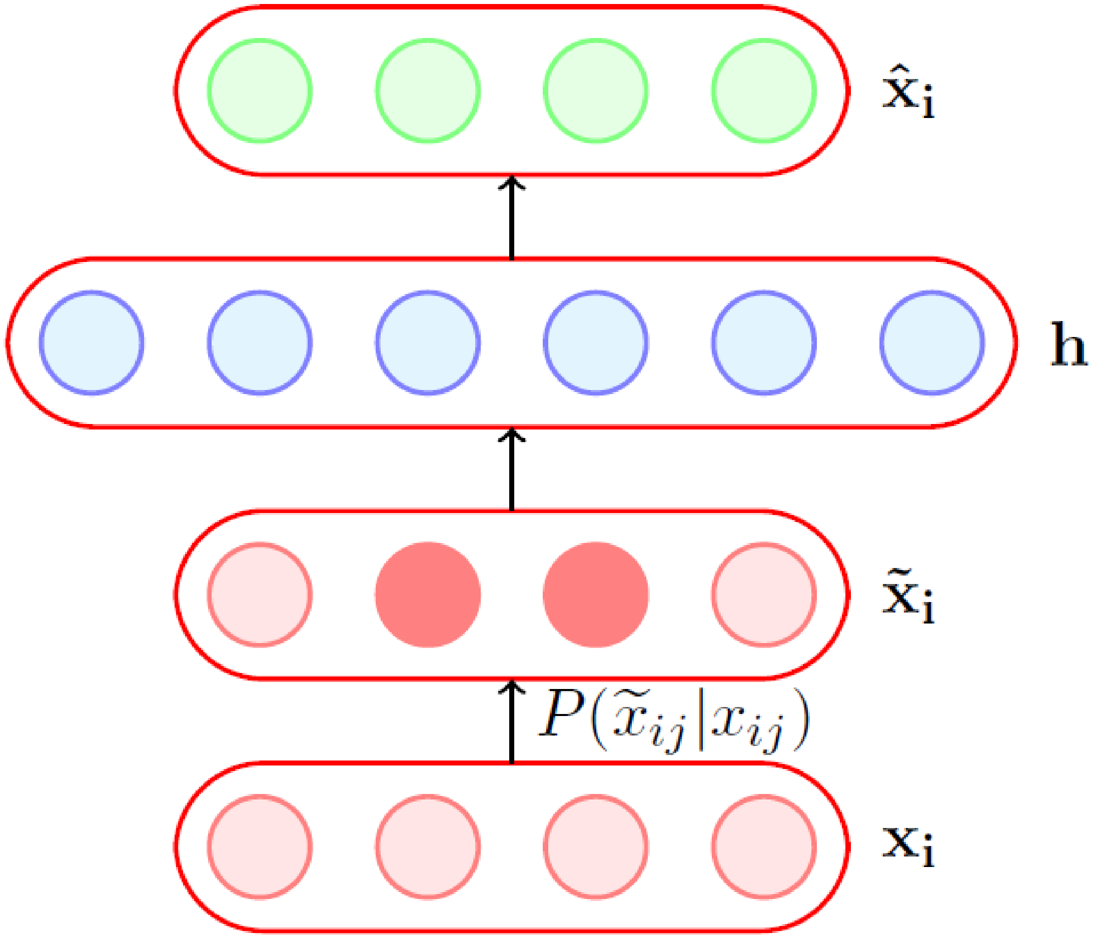


Figure: 50% Denoising AE (q=0.5)

- The vanilla AE does not learn many meaningful patterns
- The hidden neurons of the denoising AEs seem to act like pen-stroke detectors (for example, in the highlighted neuron the black region is a stroke that you would expect in a '0' or a '2' or a '3' or a '8' or a '9')
- As the noise increases the filters become more wide because the neuron has to rely on more adjacent pixels to feel confident about a stroke

Gaussian Noise corruption



$\hat{\mathbf{x}}_\mathbf{i}$
$\mathbf{h}$
$\tilde{\mathbf{x}}_\mathbf{i}$
$P(\widetilde{x}_{ij}|x_{ij})$
$\mathbf{x}_\mathbf{i}$

- We saw one form of $P(\widetilde{x}_{ij}|x_{ij})$ which flips a fraction q of the inputs to zero
- Another way of corrupting the inputs is to add a Gaussian noise to the input

$$\widetilde{x}_{ij} = x_{ij} + \mathcal{N}(0, 1)$$

- We will now use such a denoising AE on a different dataset and see their performance

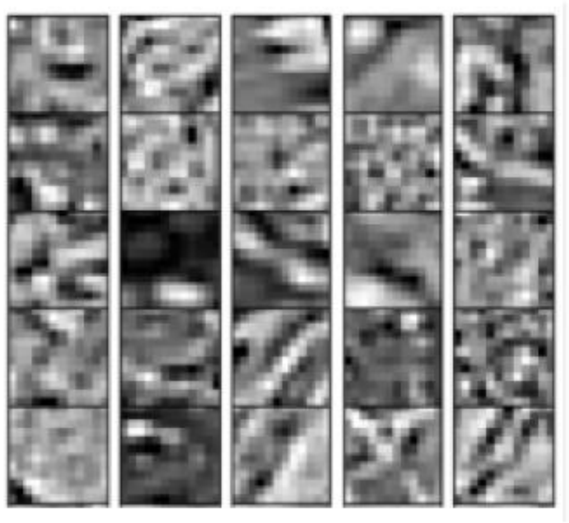


Figure: Data

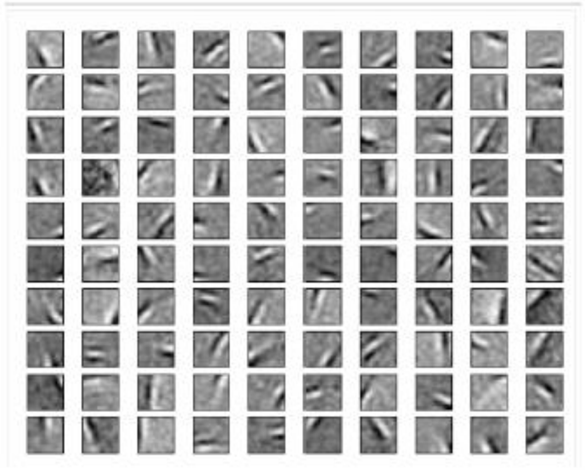


Figure: AE filters

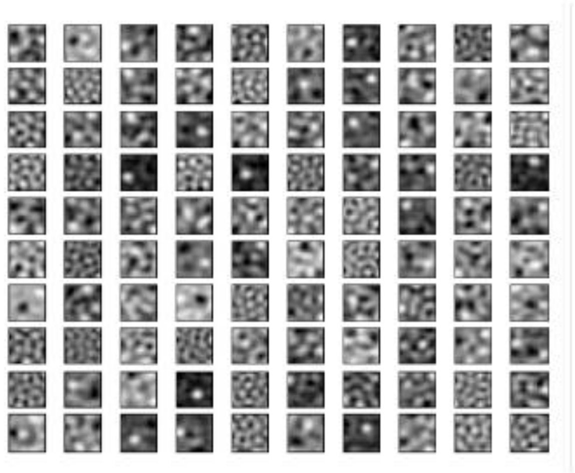
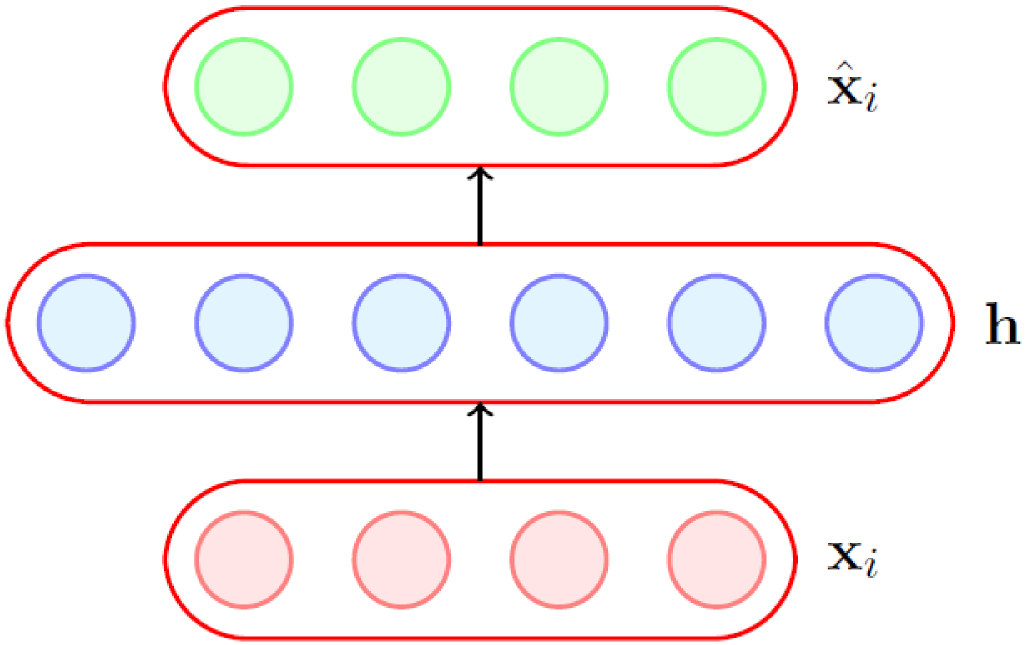


Figure: Weight decay filters

- The hidden neurons essentially behave like edge detectors
- PCA does not give such edge detectors

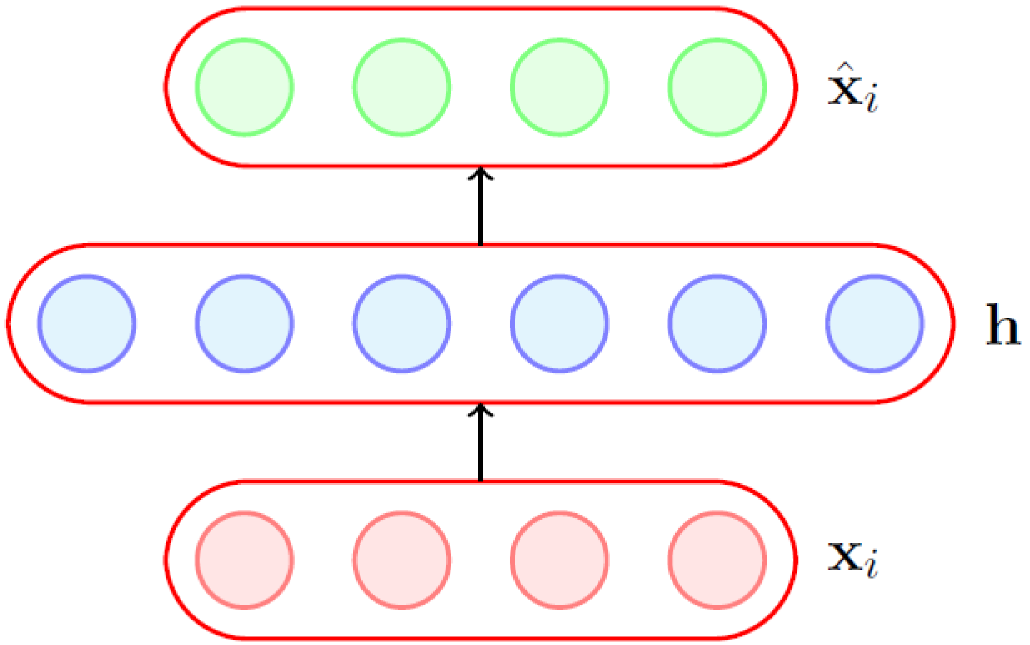
Sparse Autoencoder



- A hidden neuron with sigmoid activation will have values between 0 and 1
- We say that the neuron is activated when its output is close to 1 and not activated when its output is close to 0.
- A sparse autoencoder tries to ensure the neuron is inactive most of the times.

Sparse autoencoders are a type of autoencoder that incorporates a sparsity constraint in the learning process. The sparsity constraint encourages the autoencoder to have a small number of active neurons, meaning only a subset of neurons are activated for each input sample. :
sparse autoencoders are useful for unsupervised feature learning tasks, such as dimensionality reduction or pretraining deep neural networks.

Sparse Autoencoder



The average value of the activation of a neuron l is given by

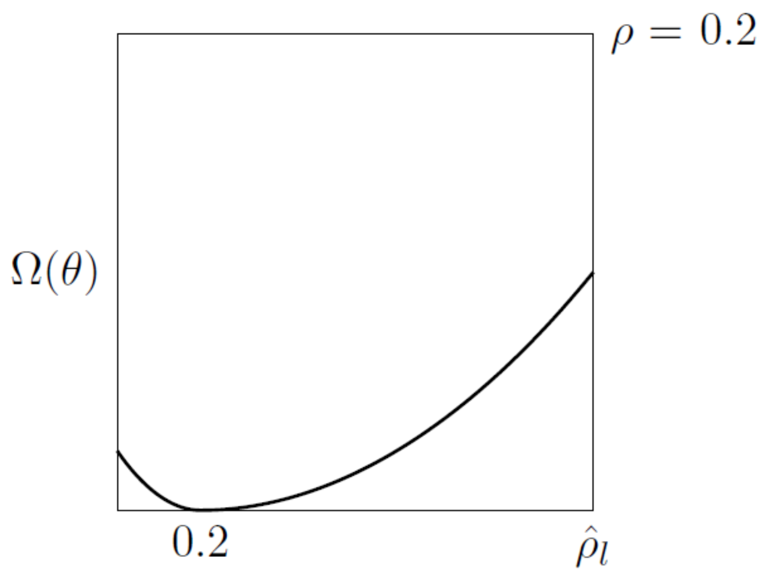
$$\hat{\rho}_l = \frac{1}{m}\sum_{i=1}^{m} h(\mathbf{x}_i)_l$$

- If the neuron l is sparse (i.e. mostly inactive) then $\hat{\rho}_l \rightarrow 0$
- A sparse autoencoder uses a sparsity parameter ρ (typically very close to 0, say, 0.005) and tries to enforce the constraint $\hat{\rho}_l = \rho$
- One way of ensuring this is to add the following term to the objective function

$$\Omega(\theta) = \sum_{l=1}^{k} \rho \log \frac{\rho}{\hat{\rho}_l} + (1-\rho)\log\frac{1-\rho}{1-\hat{\rho}_l}$$

- When will this term reach its minimum value and what is the minimum value? Let us plot it and check.

Sparse Autoencoder



The average value of the activation of a neuron l is given by

$$\hat{\rho}_l = \frac{1}{m}\sum_{i=1}^{m} h(\mathbf{x}_i)_l$$

- If the neuron l is sparse (i.e. mostly inactive) then $\hat{\rho}_l \to 0$
- A sparse autoencoder uses a sparsity parameter ρ (typically very close to 0, say, 0.005) and tries to enforce the constraint $\hat{\rho}_l = \rho$
- One way of ensuring this is to add the following term to the objective function

$$\Omega(\theta) = \sum_{l=1}^{k} \rho \log \frac{\rho}{\hat{\rho}_l} + (1-\rho)\log\frac{1-\rho}{1-\hat{\rho}_l}$$

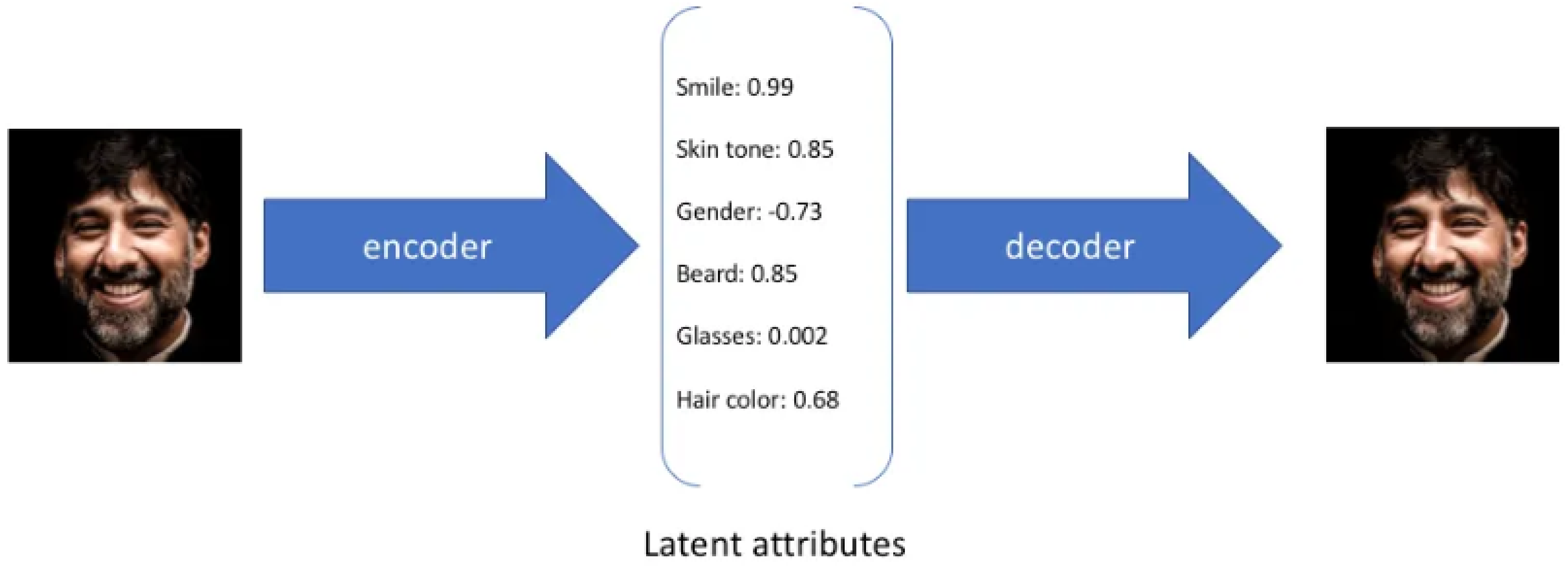
- When will this term reach its minimum value and what is the minimum value? Let us plot it and check.

By enforcing sparsity, the autoencoder learns to identify and capture the most salient features in the data, discarding irrelevant or redundant information. This makes sparse autoencoders useful for unsupervised feature learning tasks, such as dimensionality reduction or pretraining deep neural networks.

The sparsity constraint in sparse autoencoders can enhance their robustness to noisy or corrupted input data. By encouraging only a subset of neurons to be activated, the autoencoder learns to focus on the most informative features and filter out irrelevant or noisy information. This can improve the autoencoder's ability to denoise or reconstruct corrupted input samples.

Variational Autoencoder

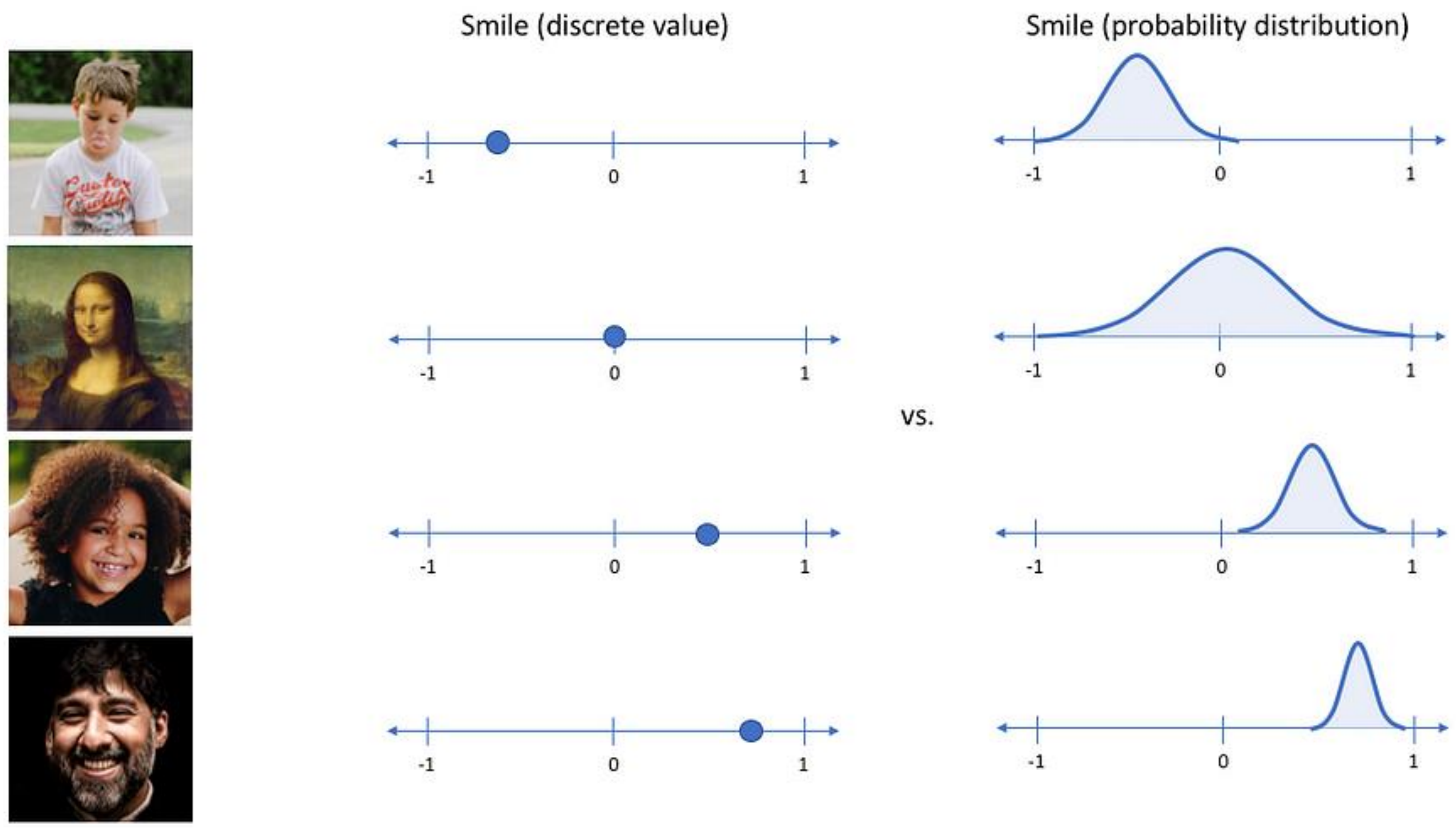
Basics : Lower dimensional encoding



Lower dimensional encoding? when we encode 784x1 or 28x28 dimension to a say much smaller 32x1 dimension, we basically mean that now we have 32 features which are the most important features and reflect most of the information in the data, or image. So, say for a face, when we encode a face image of say 32x32 dimension, it has the full facial two-dimensional image, now, if we encode it to 6x1 dimension, i.e, send it through a bottleneck layer of 6 nodes, we will basically get 6 features which contribute most or the major information about the facial image. Say, for the 6 features we have **a smile, skin tone, gender, beard, wears glasses, and hair color**.

Variational Autoencoder: Latent attribute as probability distribution

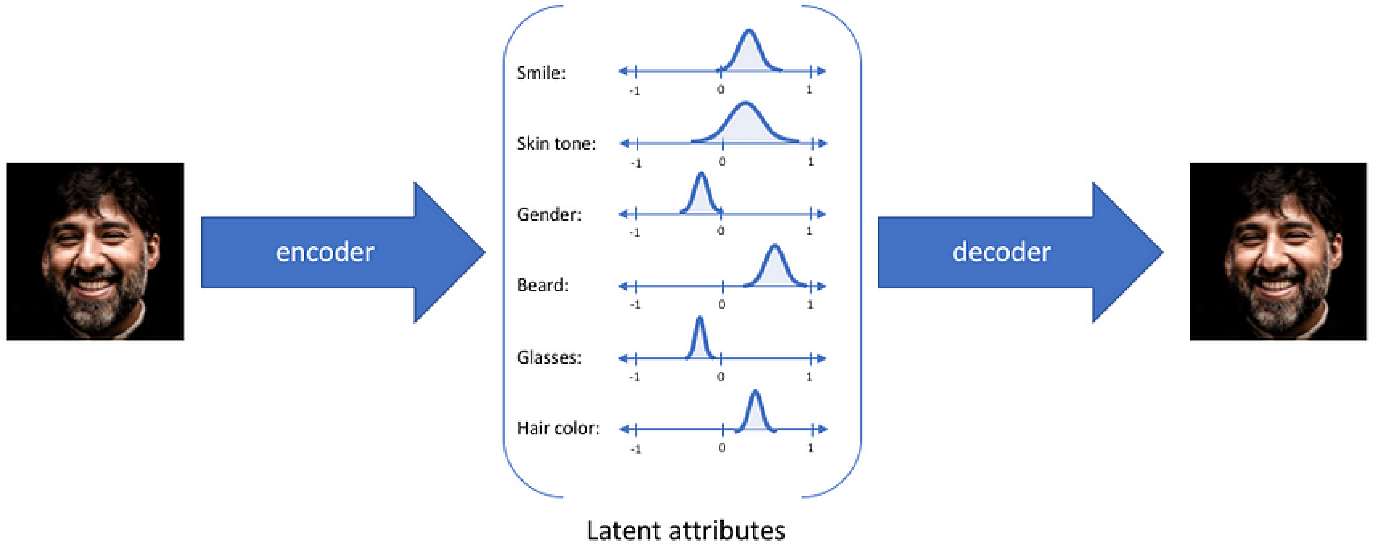
We have seen that the values of the latent attributes are always discrete. This is where the variational autoencoders are different. Instead of considering to pass discrete values, the variational autoencoders pass each latent attribute as a probability distribution.



Variational Autoencoder: Latent attribute as probability distribution

We have seen that the values of the latent attributes are always discrete. This is where the variational autoencoders are different. Instead of considering to pass discrete values, the variational autoencoders pass each latent attribute as a probability distribution.

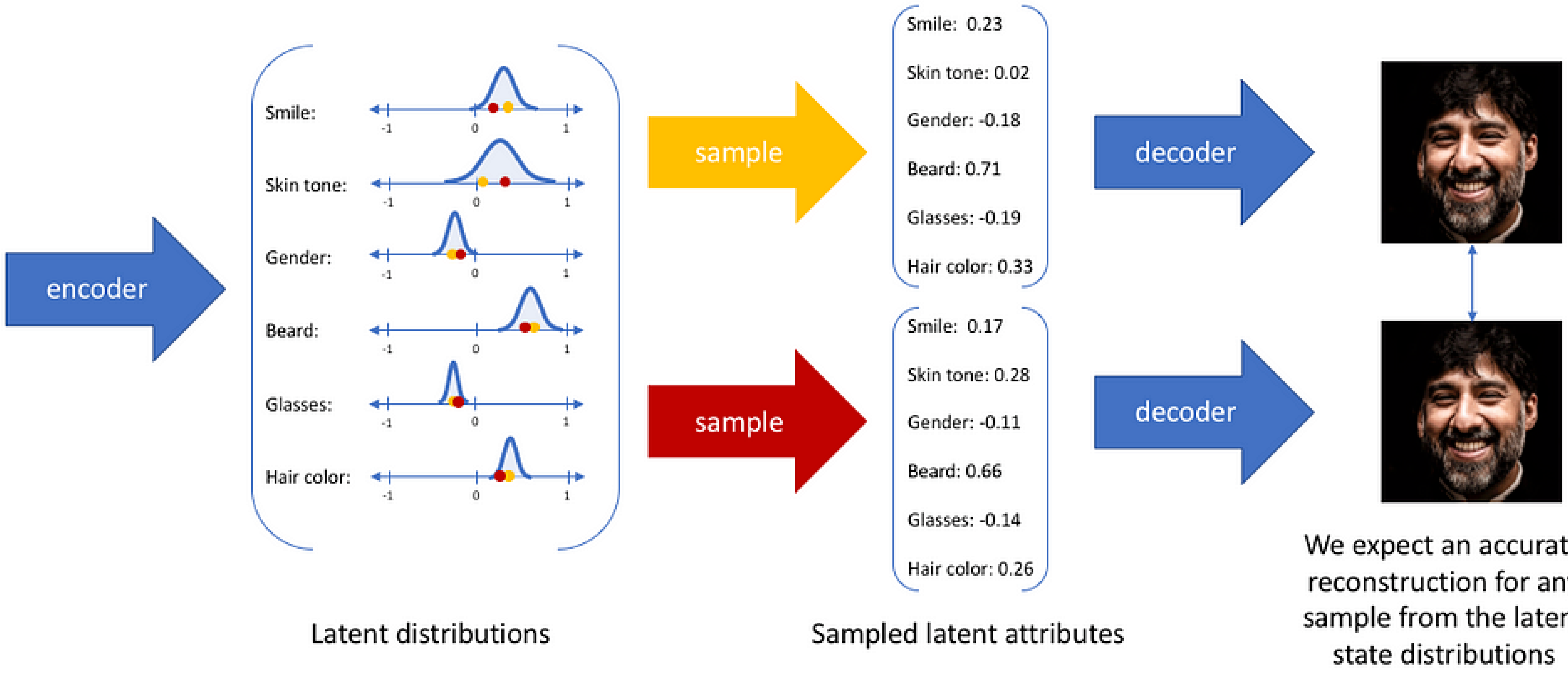
Now the representation →



Variational autoencoders as a generative network

Now, we can see each latent attribute is passed as a probability distribution. The decoder samples from each latent distribution and decodes to reconstruct the image. So, as the sampling is random and not backpropagated the reconstructed image is similar to the input but is not actually present in the input set. This is the reason for variational autoencoders to be known as a generative network.

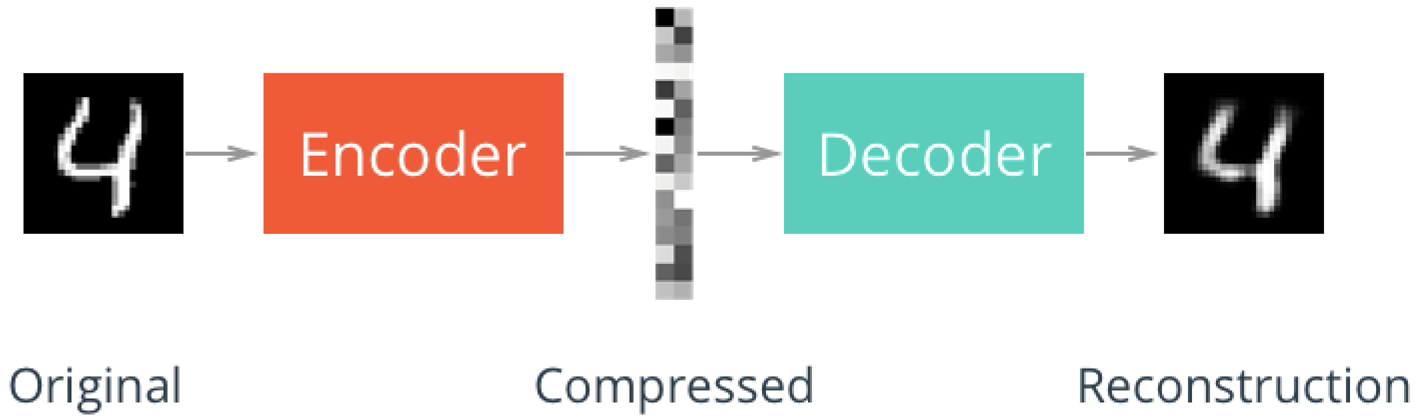
Now, to create a distribution for each latent vector, the encoder in place of passing the value, pass the mean and standard deviation of the distribution, which is used to create construct the normal distribution.



Applications

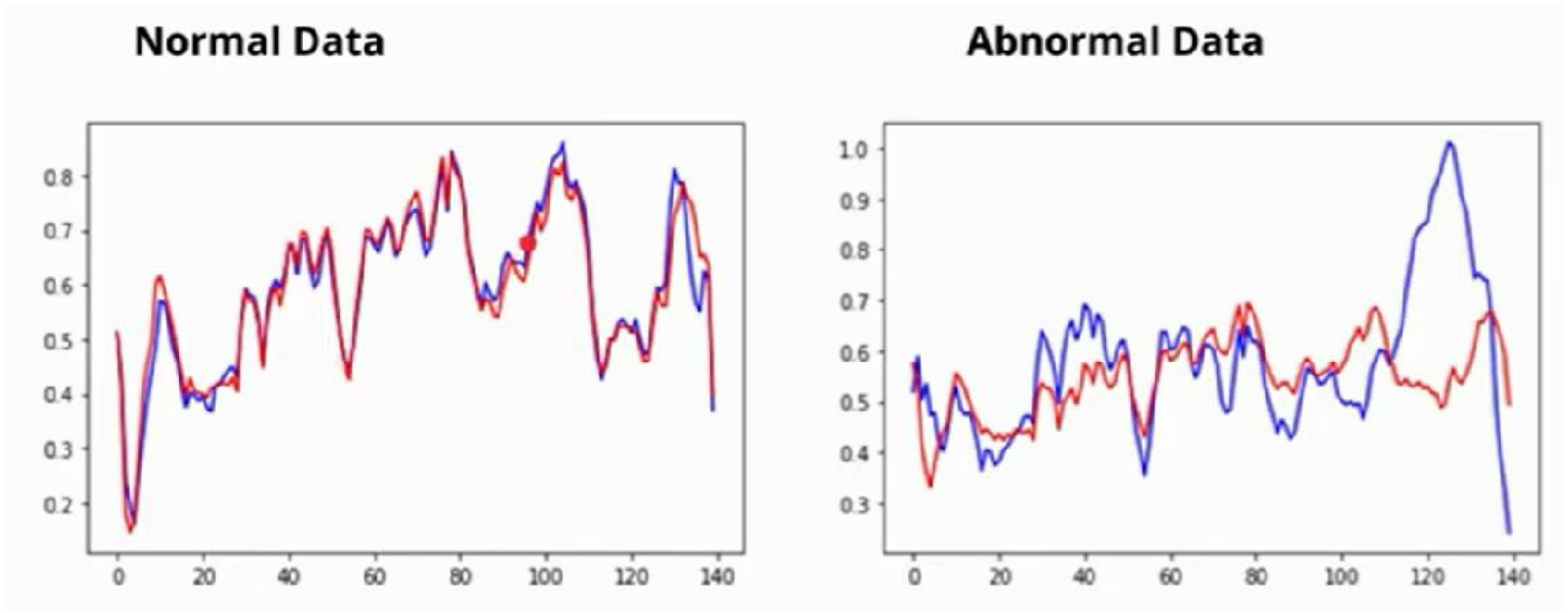
Autoencoder for Image File Compression (Dimensionality Reduction)

Using Autoencoders we can compress image files which are great for sharing and saving in a faster and more memory efficient way. These compressed images contain the key information same as in original images but in a compressed format that can be used further for other reconstructions and transformations.



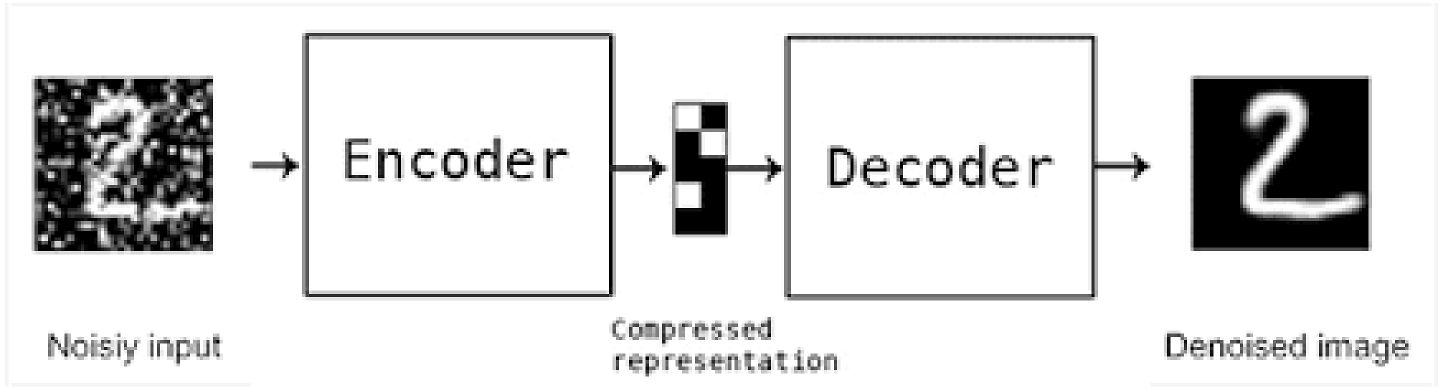
Autoencoders are used largely for anomaly detection:

Autoencoders create encodings that basically captures the relationship among data. Now, if we train our autoencoder on a particular dataset, the encoder and decoder parameters will be trained to represent the relationships on the datasets the best way. Thus, will be able to recreate any data given from that kind of dataset in the best way. So, if data from that particular dataset is sent through the autoencoder, the reconstruction error is less. But if some other kind of data is sent through the autoencoder it will generate a huge reconstruction error. If we are able to apply a correct cutoff we will be able to create an anomaly detector.



Autoencoders are used for Noise Removal

If we can pass the noisy data as input and clean data as output and train an autoencoder on such given data pairs, trained Autoencoders can be highly useful for noise removal. This is because noise points usually do not have any correlations. Now, as the autoencoders need to represent the data in the lowest dimensions, the encodings usually have only the important relations there exists, rejecting the random ones. So, the decoded data coming out as output of an autoencoder is free of all the extra relations and hence the noise.



Autoencoders as Generative models:

Before the GAN’s came into existence, Autoencoders were used as generative models. One of the modified approaches of autoencoders, variational autoencoders are used for generative purposes.

Image Reconstruction

The convolutional Autoencoder learns to remove noise from a picture or reconstruct the missing parts, so the input noisy version becomes the clean output version. The network also fills the gap in the image.

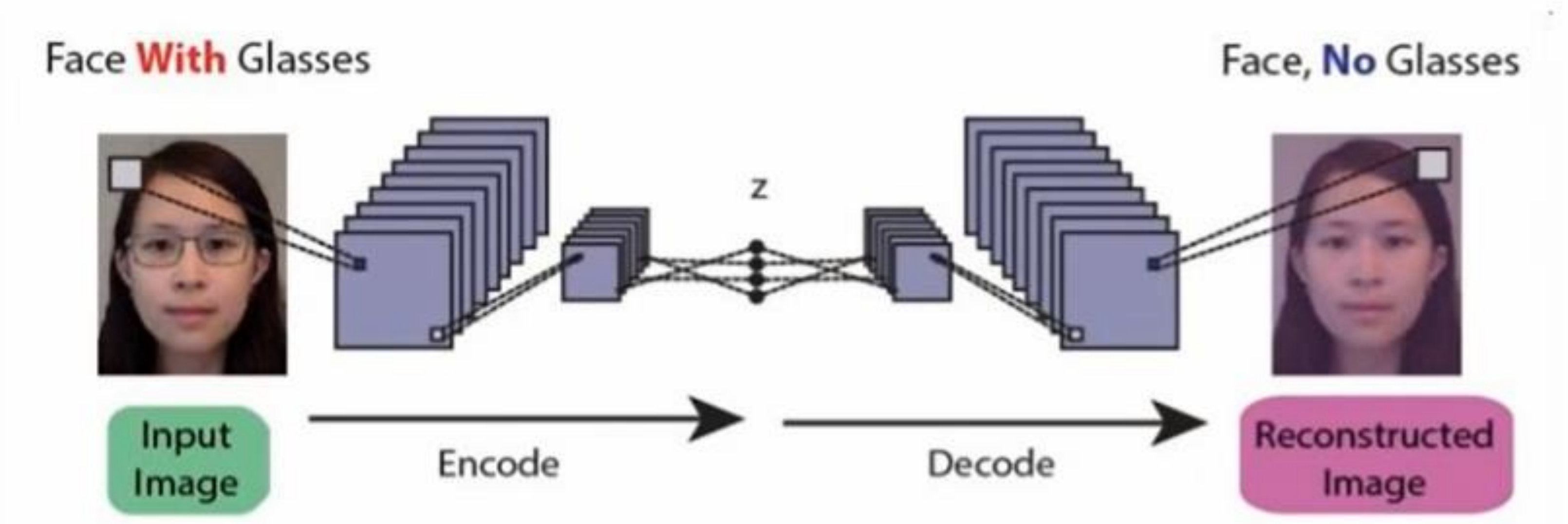
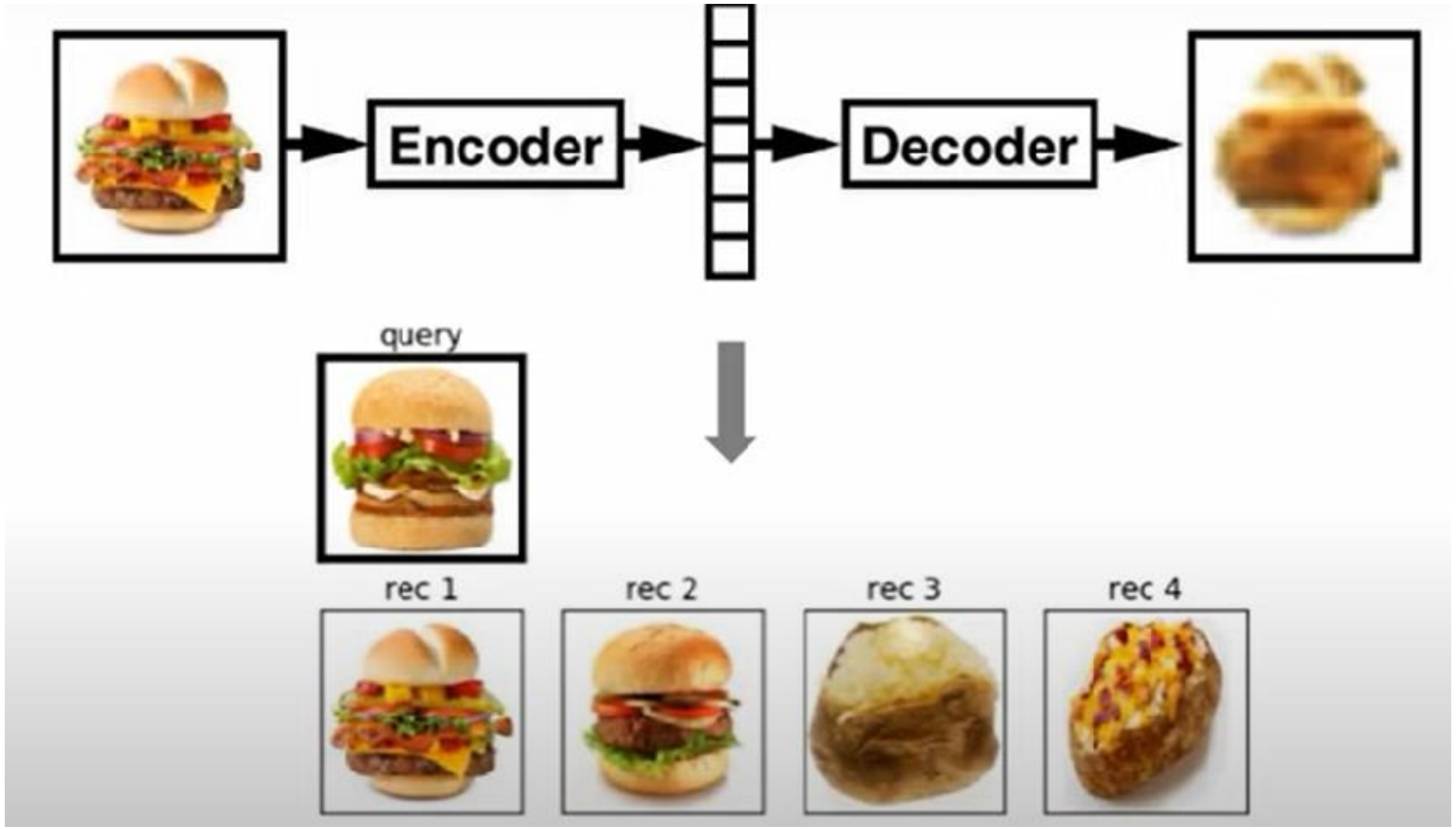


Image Search

Image Search – Deep autoencoders can compress images into 30 number vector images. Image search becomes a matter of application where a search engine will compress an image and compare the vector to index and translate to a matching image.



Training of AutoEncoders- Hyper parameters

- There are four significant hyperparameters that we need to set before training them.
  1) **<u>Code Size</u>** represents the number of nodes in the middle layer—the smaller size results in more compression.
  2) **<u>Number of Layers</u> –** The Autoencoder can be as deep as we want to be.
  3) **<u>Loss function</u> –** To update the weights, we must calculate the loss, which we need to minimize using optimizer and weight updation. Mainly mean squared error and binary cross-entropy. The input value is 0-1; then, we use cross-entropy. Otherwise, we use mean squared error.
  4) **<u>Number of nodes per layer</u> –** the number of nodes decreases with each subsequent encoder layer and increases back in the decoder.

Namah Shivaya