UNIT-5
AUTOENCODER

B. Keerthana

Assistant Professor

CSE department

GVPCE (A)

Contents

- Introduction,
- Features of Autoencoder,
- Applications of autoencoder.
- Types of Autoencoder,
- Restricted Boltzmann Machine: Boltzmann Machine,
- RBM Architecture, Types of RBM.
- Generative Adversarial Networks

Introduction to Autoencoder

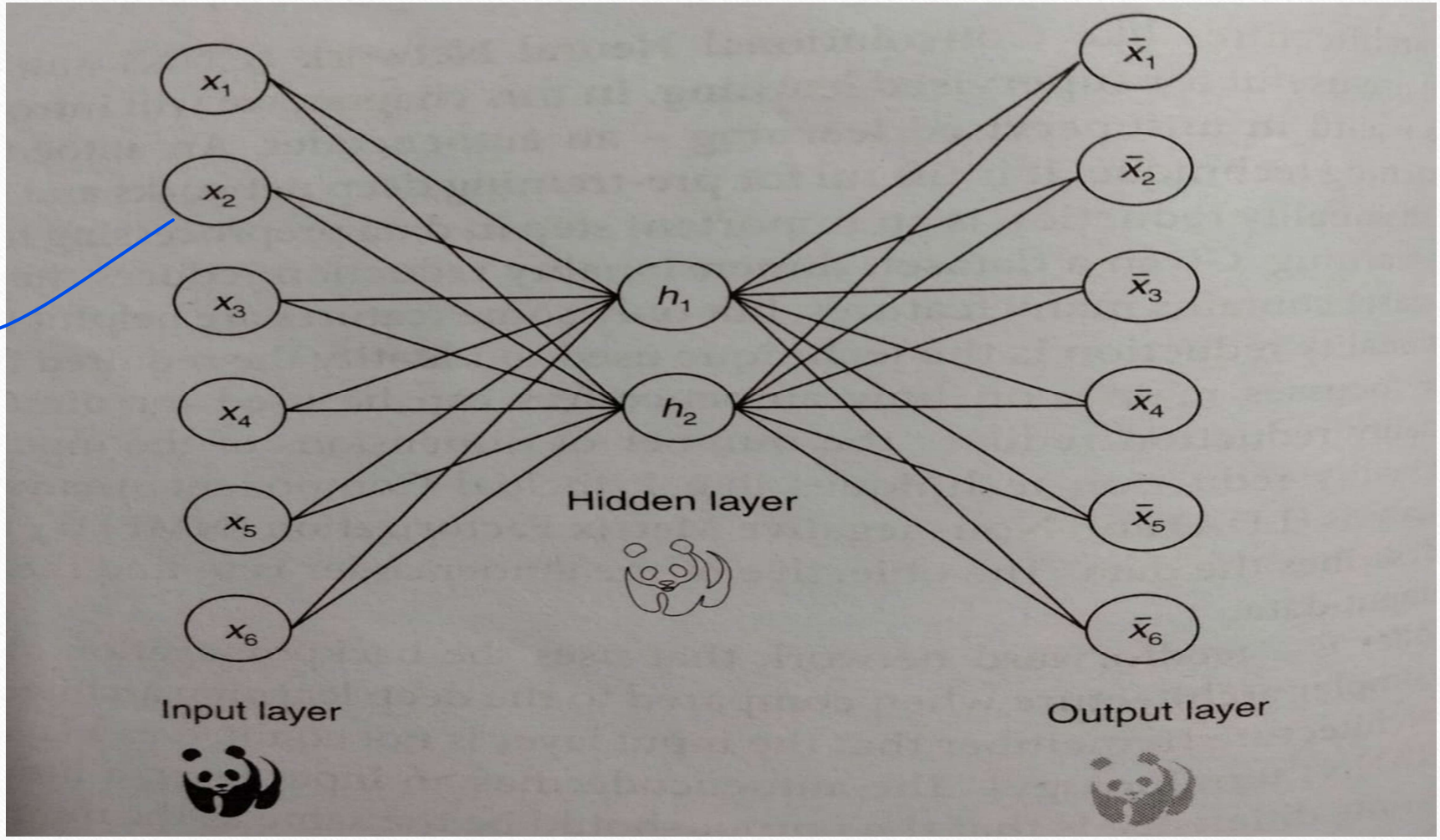
- An **autoencoder** neural network is an **unsupervised learning** algorithm that applies backpropagation, setting the target values to be equal to the inputs.
- They compress the input into a lower-dimensional *code* and then reconstruct the output from this representation.
- Autoencoder is useful for pre-training deep networks and for ***dimensionality reduction.***
- The objective of the autoencoder is to find the best latent representation of input data.

Introduction to Autoencoder

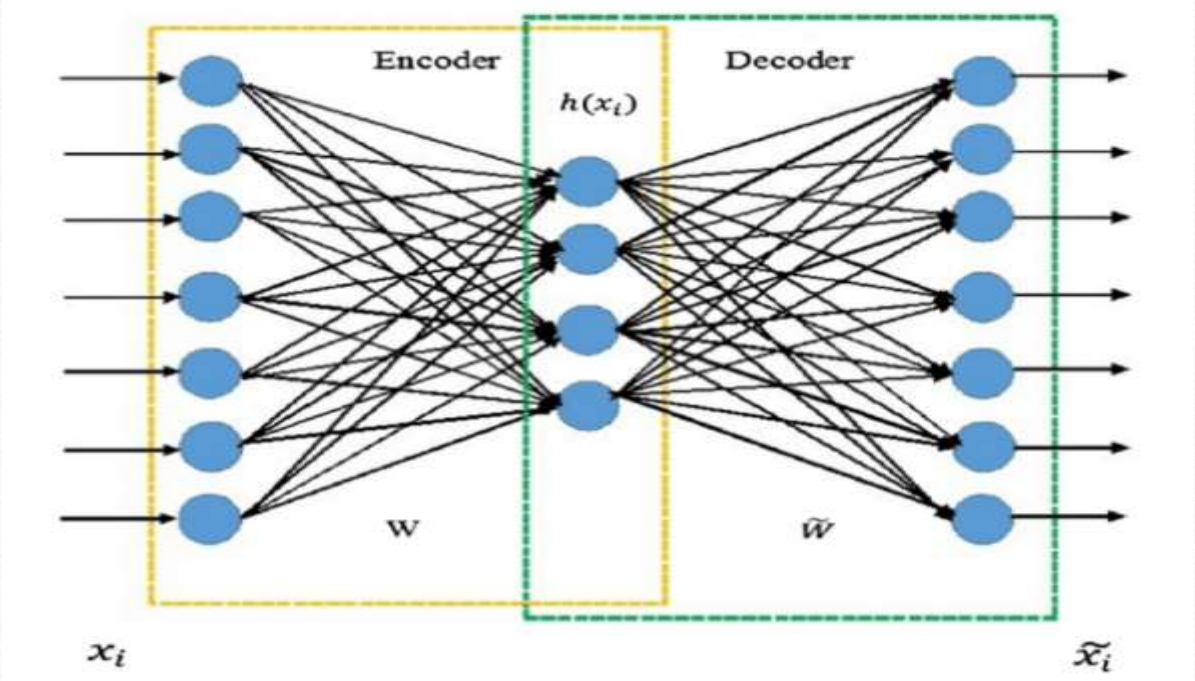
- Autoencoder are ***2 layered*** architecture which contains input layer, hidden layer and output layer. input layer not considered
- The hidden layer captures the best representative features of the input.
- Let us assume that number of hidden nodes are much less than the number of input nodes then it is called ***undercomplete autoencoder***. h<i
- If the number of hidden nodes are more than the number of input nodes then it is called ***overcomplete autoencoder.*** h>=i

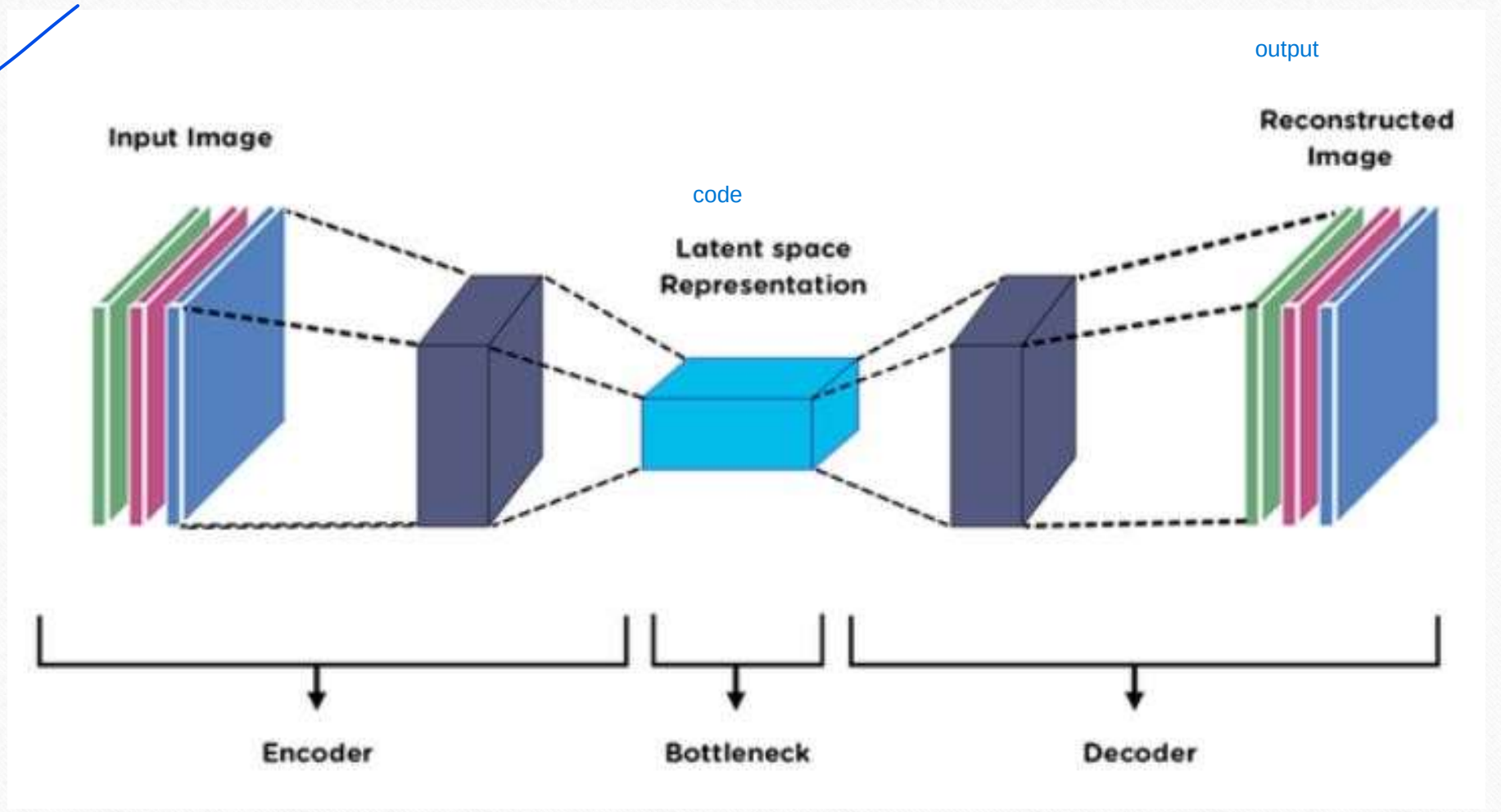
Example

- Let the number of input and output neurons be 6. The hidden layer has smaller number of neurons than the number of neurons in the input layer.
- Let us have 2 neurons in the hidden layer, the autoencoder take 6 features and encodes it using just 2 features.
- These 2 features are enough to reconstruct the 6 features.
- The dimension of the dataset is reduced from 6 to 2; this is called **dimensionality reduction**.


x_1
x_2
x_3
x_4
x_5
x_6
h_1
h_2
$\bar{x}_1$
$\bar{x}_2$
$\bar{x}_3$
$\bar{x}_4$
$\bar{x}_5$
$\bar{x}_6$
Hidden layer
Input layer
Output layer

Architecture of Autoencoder




output
Input Image
Reconstructed Image
code
Latent space Representation
Encoder
Bottleneck
Decoder

An **Autoencoder** architecture consists of three layers:

1. **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.

2. **Code:** This part of the network represents the compressed input which is fed to the decoder. This layer is also called as "**Bottleneck**" layer.

3. **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.

Features of Autoencoder

1. **Data dependent**: autoencoders are compression techniques in which they are only able to compress data similar to what they have been trained on.

Example: The model of an autoencoder used to compress house images cannot be used to compress human faces

2. **Lossy compression**: reconstruction of the original data from the compressed representation would result in a degraded output

Hyperparameters of Autoencoder

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

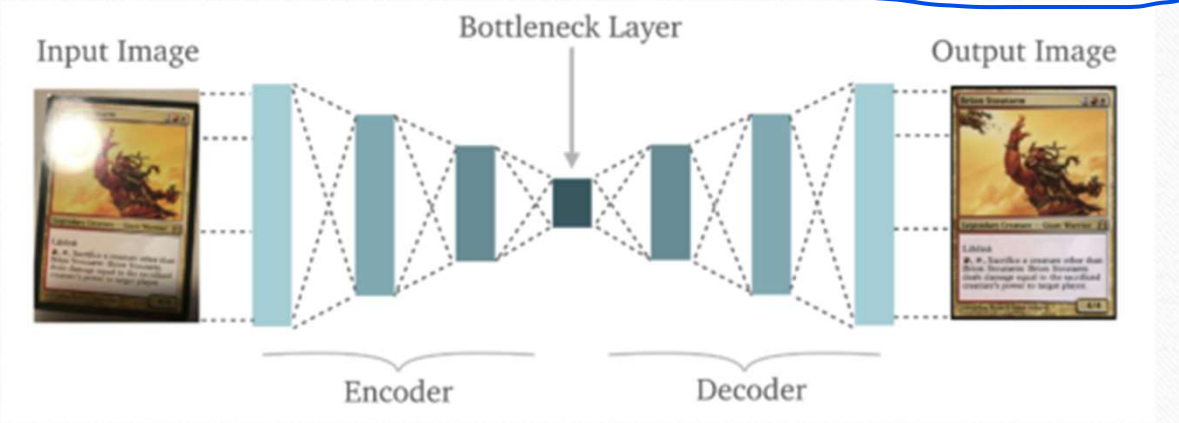
- **Code size**: It represents the number of nodes in the middle layer. Smaller size results in more compression.
- **Number of layers**: The autoencoder can consist of as many layers as we want.
- **Number of nodes per layer**: The number of nodes per layer decreases with each subsequent layer of the encoder, and increases back in the decoder.
- **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.

Applications of Autoencoder

1. **Image colouring:** Autoencoders are used for converting **any black** and **white picture** into a colored image. Depending on what is in the picture, it is possible to tell what the color should be.

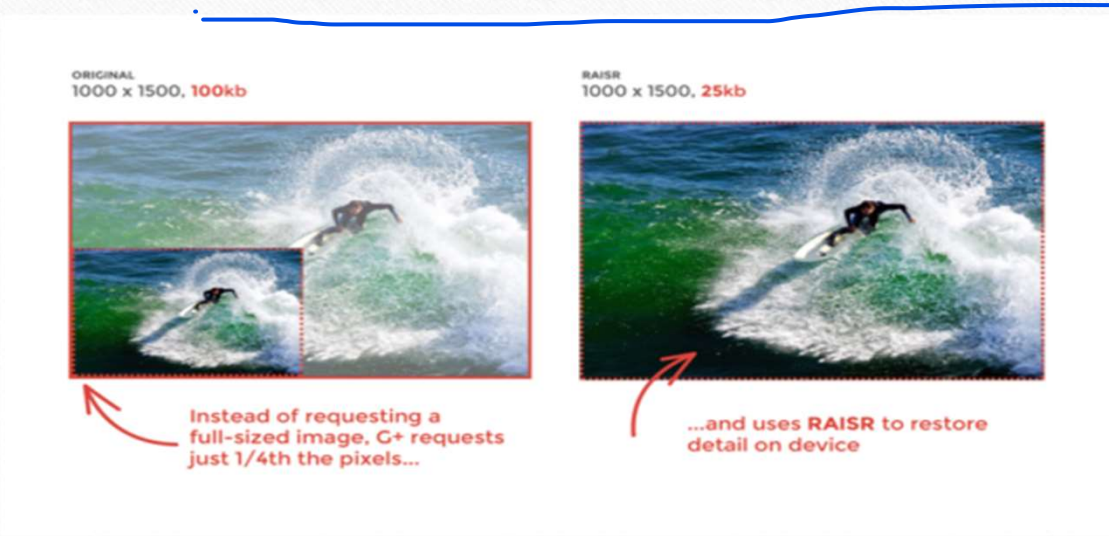
Applications of Autoencoder

2. Feature variation: It extracts only the required features of an image and generates the output by removing any noise or unnecessary interruption.



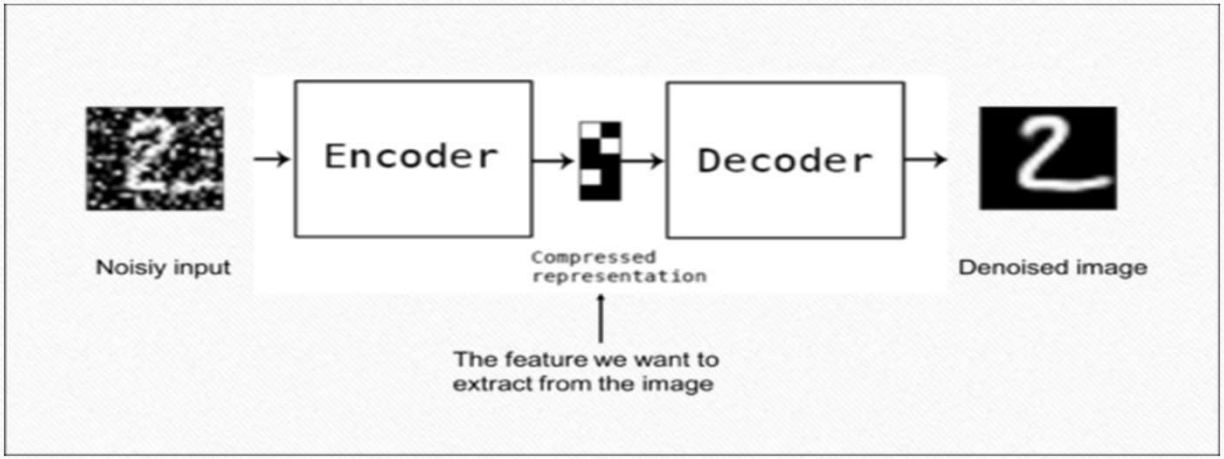
Applications of Autoencoder

3. Dimensionality Reduction: The reconstructed image is the same as our input but with reduced dimensions. It helps in providing the similar image with a reduced pixel value.



Applications of Autoencoder

4.Denoising Image: The input seen by the autoencoder is not the raw input but a stochastically corrupted version. A denoising autoencoder is thus trained to reconstruct the original input from the noisy version.



Applications of Autoencoder

5. Watermark Removal: It is also used for removing watermarks from images or to remove any object while filming a video or a movie.


Types of Autoencoder

1. Vanilla autoencoder
2. Multilayer autoencoder
3. Stacker autoencoder
4. Denoising autoencoder
5. Convolutional autoencoder
6. Regularized autoencoder
7. Deep autoencoder

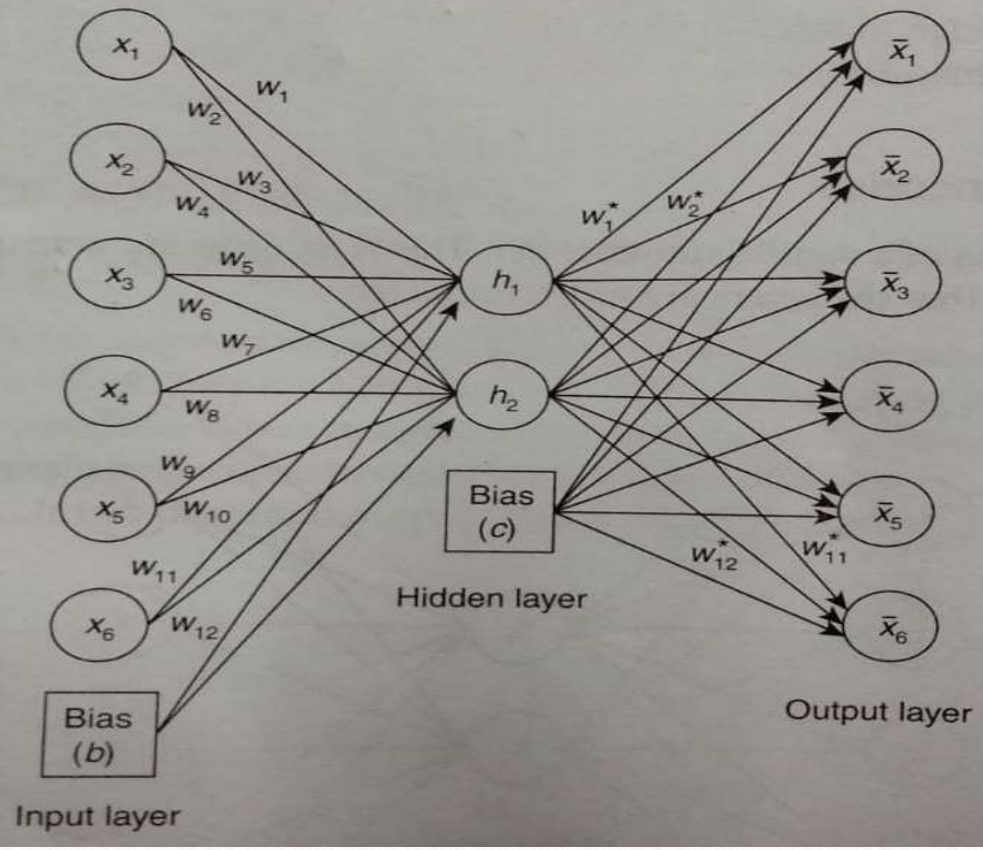
Vanilla autoencoder

- This is an ordinary autoencoder with no added features. It may be noted that the layers are fully connected.

Structure of Vanilla Autoencoder:

1. The encoder encompasses of the input layer and one hidden layer. It compresses the input data.
2. The code is a hidden layer with reduced required number of nodes.
3. The decoder decodes the output of the code to produce a lossy reconstruction of the input and produces the final output.

The encoder takes (1, 1, 0, 0, 0, 1) as input and outputs (h_1, h_2). Code has the output (h_1, h_2). In an ideal case, the decoder takes (h_1, h_2) as input and outputs (1, 1, 0, 0, 0, 1). The vanilla autoencoder with weights and bias is shown in Fig. 4.3.



Let us assume there are n input nodes and d hidden nodes. The encoder is the function

$$f(wx + b)$$

where w is the real-valued vector $(w_1, w_2, \ldots, w_{12})$ and can be generalized as $w \in R^{d \times n}$; b is the real-valued vector (b_1, b_2) and can be generalized as $b \in R^d$; x is the input vector and can be generalized as $x \in R^n$.

Here $wx + b$ is a linear transformation followed by a non-linear transformation produced by the activation function f.

Similarly, the decoder is the function

$$g(w * \bar{x} + c)$$

where $w*$ is the real-valued vector $w_1*, w_2*, \ldots, w_{12}*$ and can be generalized as $w* \in R^{n \times d}$; c is the real-valued vector $(c_1, c_2 \ldots, c_6)$ and can be generalized as $c \in R^n$; y is the output vector and can be generalized as $\bar{x} \in R^n$.

Here $w * \bar{x} + c$ is a linear transformation followed by a non-linear transformation produced by the activation function g.

The choice of the activation function can differ depending on the input. For example, if the input values are binary then the activation function can be a logistic function.

4.3.1.2 Loss Function for Vanilla Autoencoder

The loss function is written in such a way that the output is the same as the input. The loss function used here is the mean squared error. For an autoencoder, it is the average of the squared error along all the input dimensions. If there are n input dimensions, then the mean squared error is

$$\min_{w,w*,B,C} \frac{1}{n} \sum_{i=1}^{n} (\bar{x}_i - x_i)^2$$

For a lossless compression, the loss function would be closer to 0. If the activation function is logistic, then the output would be in the range [0, 1]. Here, the cross-entropy loss function can be used, which is given by

$$-\sum_{i=1}^{n} x_i \log(\bar{x}_i) + (1 - x_i) \log(1 - \bar{x}_i)$$

Multilayer autoencoder

- When the vanilla autoencoder has more than one hidden layer it is termed the multilayer autoencoder.
- Multilayer autoencoder is a multilayer perceptron with symmetry in the encoder and decoder sides
- The value in any layer can be used as an intermediate representation of features. But usually it is symmetric and the middle layer is used for feature representation.
- The loss function is like that of vanilla autoencoder.

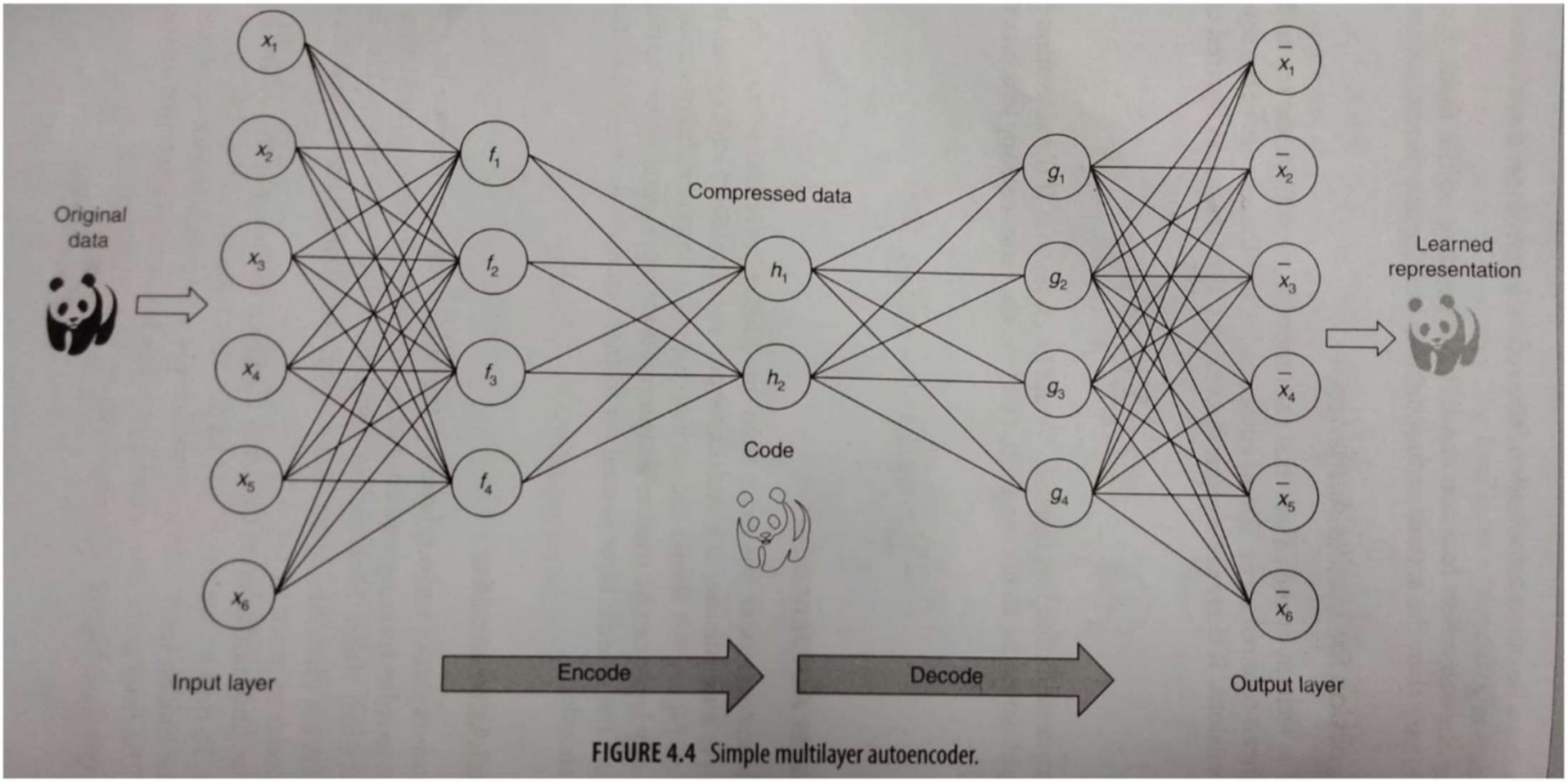


FIGURE 4.4 Simple multilayer autoencoder.

Stacked autoencoder

- Stacked autoencoders stacks various layers of hidden layers in the encoder and the decoder.
- The training does not involve training end to end as in MLP using the backpropagation algorithm.
- Rather, when there are multiple hidden layers, the first hidden layer (h^1) is trained and the parameters are identified using backpropagation.
- To train the second hidden layer (h^2), h^1 is the given in input and output layers and h^2 is used as code layer. To find h^3, we use h^2 and so on.

multi layer perceptron

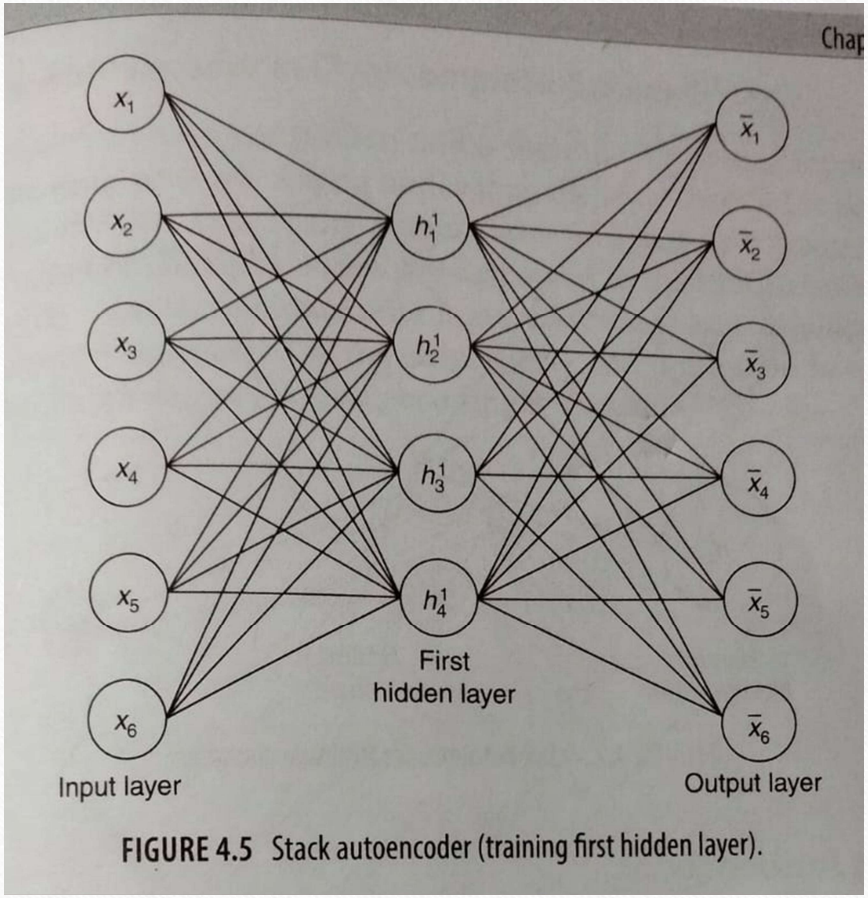


FIGURE 4.5 Stack autoencoder (training first hidden layer).

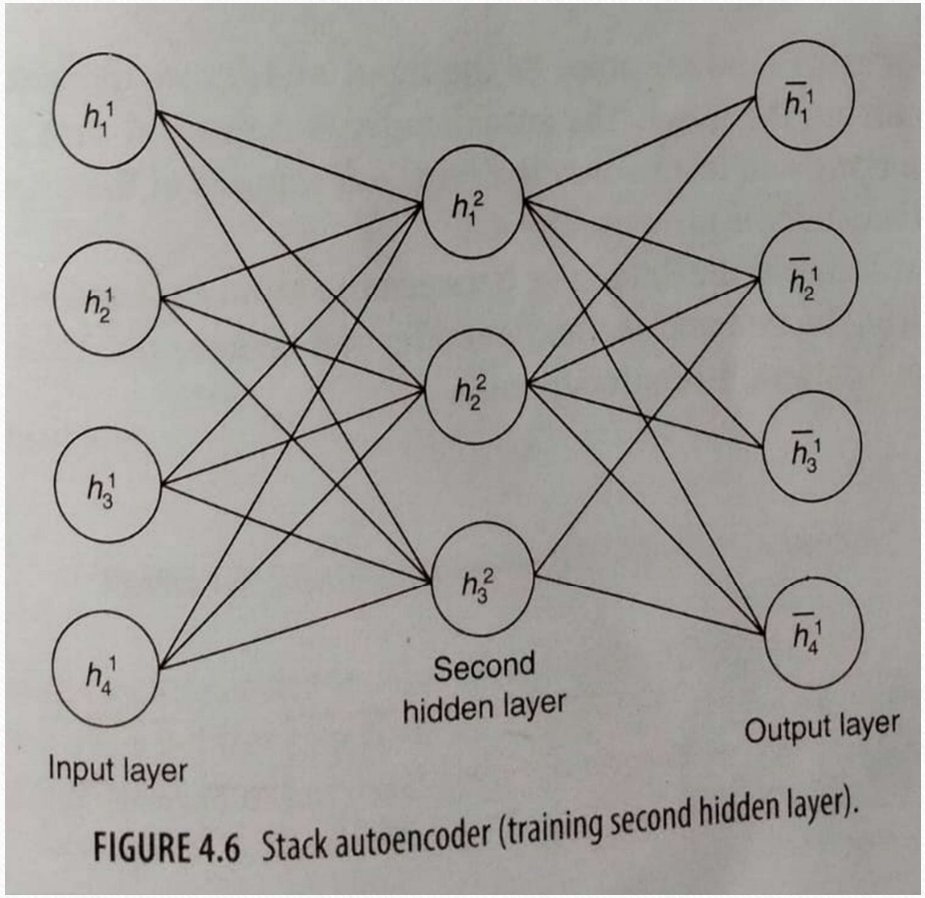


FIGURE 4.6 Stack autoencoder (training second hidden layer).

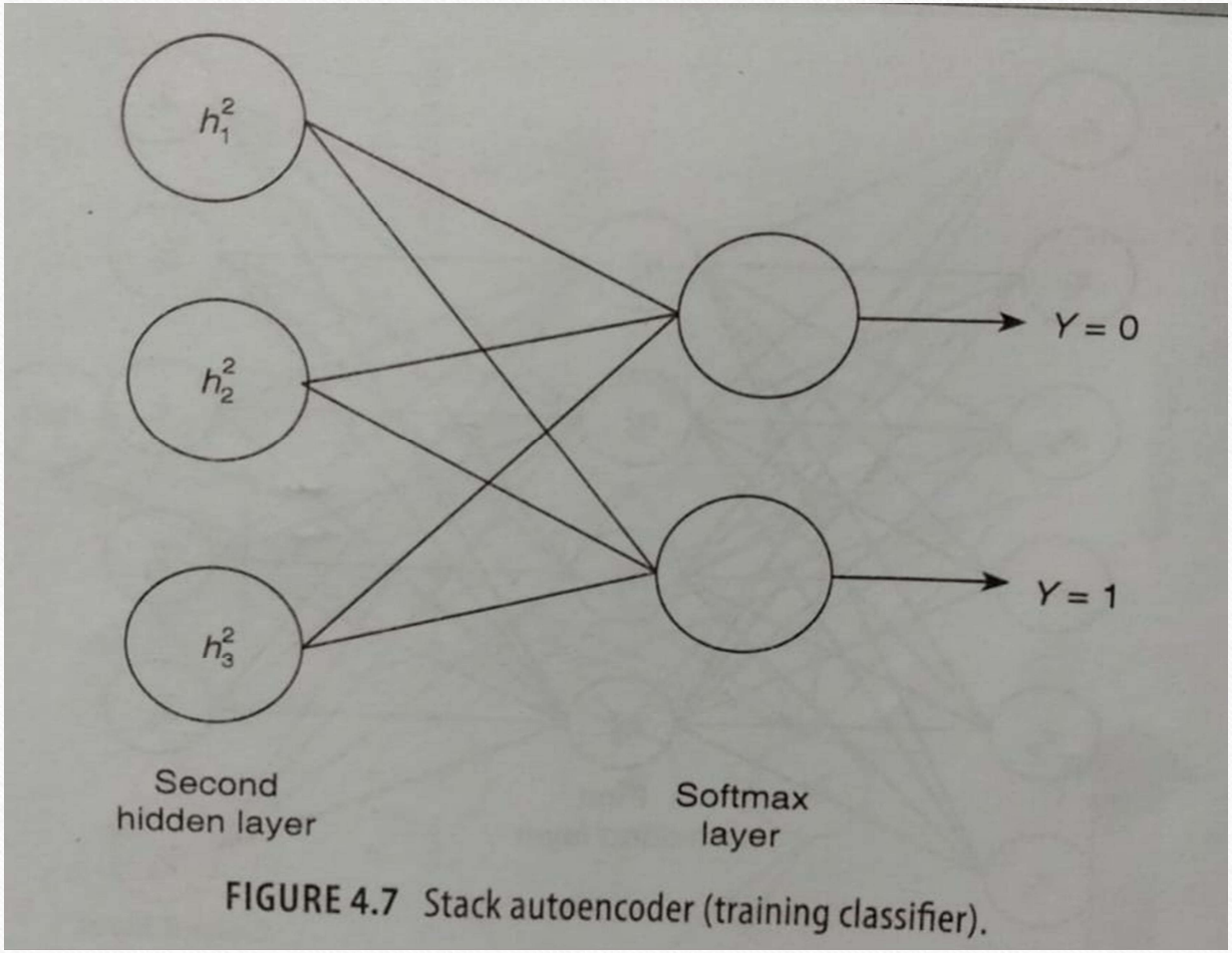


FIGURE 4.7 Stack autoencoder (training classifier).

Denoising autoencoder

- Denoising autoencoder adds random noise to the input and forces the autoencoder to learn the original data after removing the noise.
- The autoencoder is trained in such a way that it identifies the noise, remove the noise and learns only the required features of the original data.
- The loss function still checks the difference between the input data and output data.
- This ensures that there is no overfitting of data and the autoencoder can remove the noise and learn the important features of the input data after removing noise.

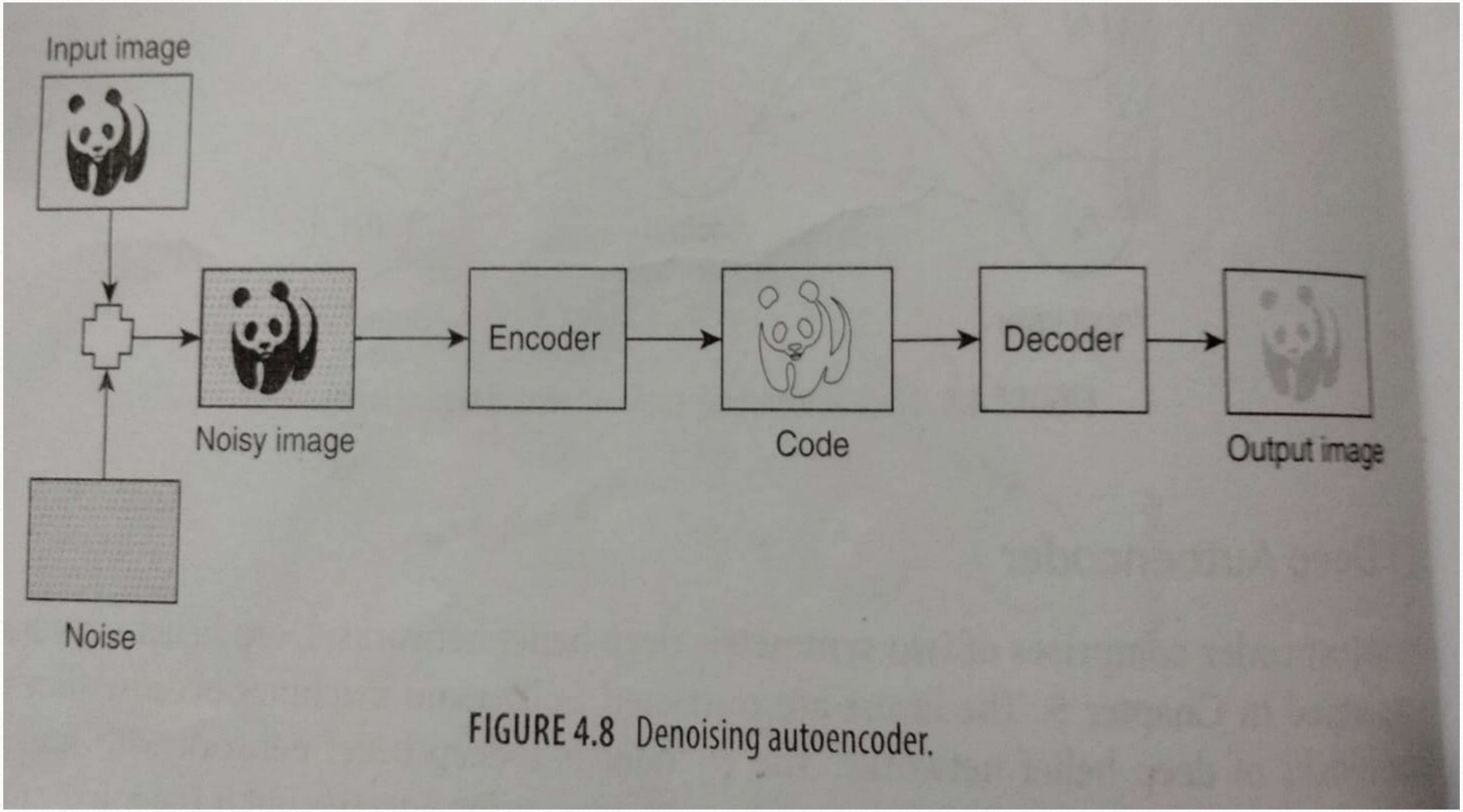


FIGURE 4.8 Denoising autoencoder.

Denoising Autoencoder in overcomplete Autoencoder

- It is always not necessary to have less hidden nodes than the number of input nodes in the code layer, may also have equal or more number of nodes in code layer.
- We expect the autoencoder to learn new features, but what might happen is that the values in the input nodes will be copied to the hidden nodes without learning any useful information.
- The input data is stored without any modification in the hidden nodes and is subsequently transferred to the output layer and found to have learnt the identity function.

one problem that can be solved using denoising auto encoders is avoiding identity encoding

identity encoding

just as an undercomplete autoencoder is used to compress input data by extracting useful features , an overcomplete ae is used to seperate jumbled features in an input data.

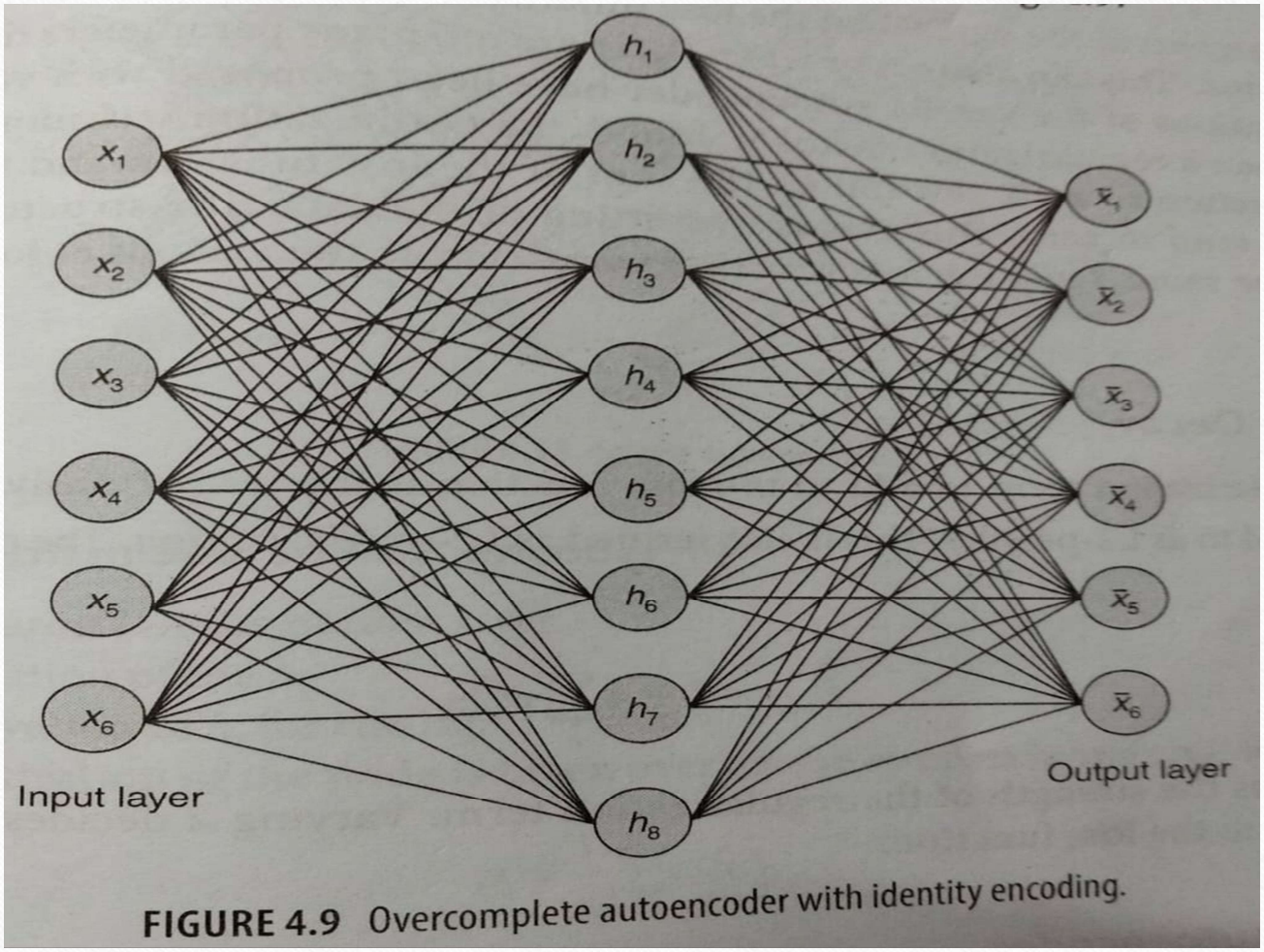


FIGURE 4.9 Overcomplete autoencoder with identity encoding.

Convolutional autoencoder

- It combines the architecture of CNN and the Vanilla autoencoders.
- The encoder is combination of the convolution layers and pooling layers as in a normal CNN.
- The pooling layers perform Downsampling, while the decoder performs deconvolution and up-sampling.
- The final layer gives the output which is expected to be the same as the input.

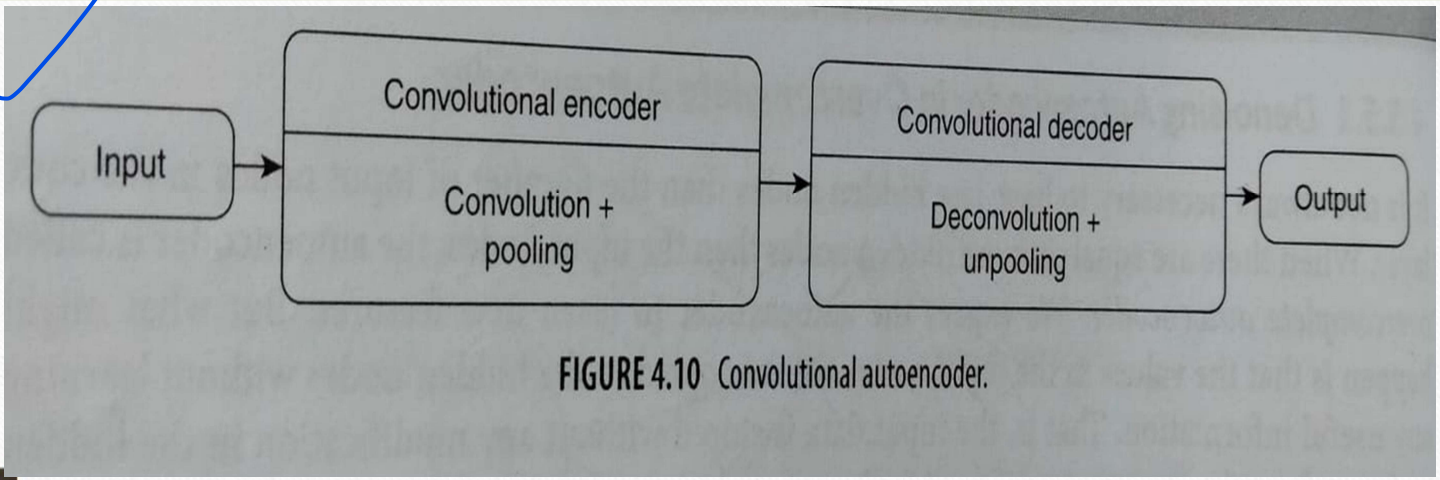
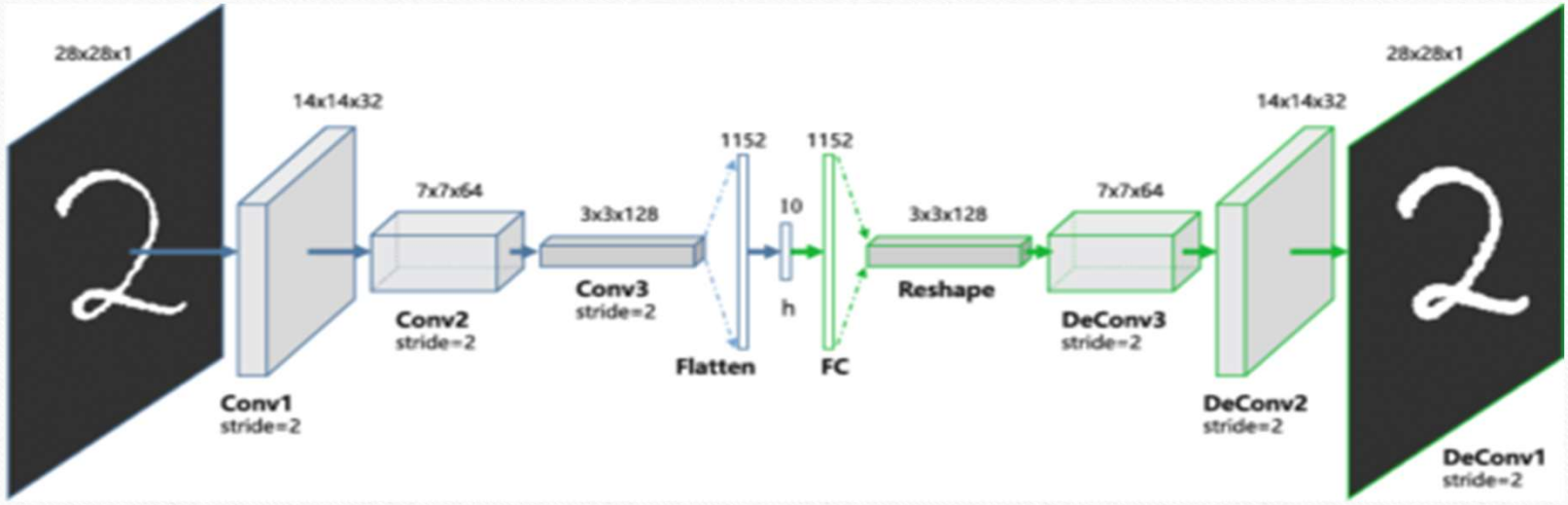


FIGURE 4.10 Convolutional autoencoder.



Regularized autoencoder

- Autoencoder can overfit the data when loss function is close to zero. Overfitting can lead to poor generalization.
- When a regularization factor is added, the optimization technique does not try to take the loss function to zero. This avoids overfitting of the data.
- The structure of the autoencoder is remains same. Only the loss function varies across the various autoencoders that use regularization.

Regularized autoencoder (cont..)

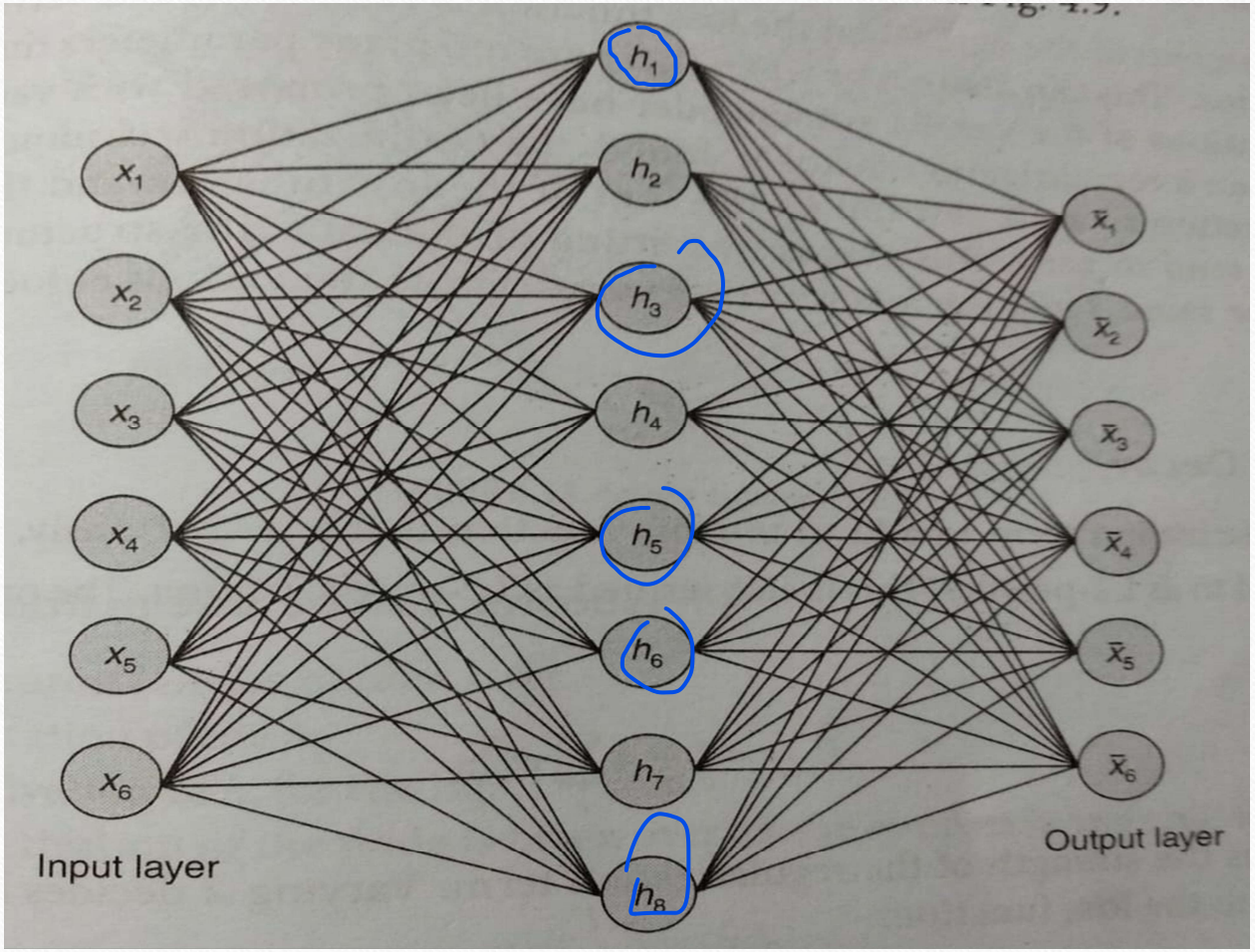
- Weight Decay: The simplest regularization term to add to the loss function is the weigh decay, which is given by $\sum_{ij} w_{ij}^2$ and referred to as L-2 penalty. The overall function becomes.

$$L_{new} = L + \lambda \sum_{ij} w_{ij}^2$$

- Where λ determines the strength of the regularization term and decides how much importance can be given to the loss function

Sparse Autoencoder

- 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.


x_1
x_2
x_3
x_4
x_5
x_6
h_1
h_2
h_3
h_4
h_5
h_6
h_7
h_8
$\bar{x}_1$
$\bar{x}_2$
$\bar{x}_3$
$\bar{x}_4$
$\bar{x}_5$
$\bar{x}_6$
Input layer
Output layer

Regularization Term in Sparse Autoencoder

Let h_l represent a neuron l in the hidden layer.
Let $a(h_l)$ be the activation of the h_l.
Let $a(h_l)_{x_i}$ be the activation of h_l for the input vector x_i.
Then, the average activation of the node is taken over all the samples in training set and is given by

$$\rho_1 = \frac{1}{n}\sum_{i=1}^{n} a(h_l)_{x_i}$$

where n is the number of samples.
Sparse autoencoders make the average activation of the node tend to a value closer to zero. This value near zero is given by the sparsity parameter ρ. If $\rho = 0.03$, then the sparse autoencoder tries to make $\rho_1 = \rho$.

To ensure this, the regularization term is chosen in such a way that ρ_1 does not deviate much from ρ. The KL divergence is used for this purpose. Based on the KL divergence, the regularization term added to the loss function is given by

$$\sum_{j=1}^{m} \mathrm{KL}(\rho \,\|\, \rho_j) = \rho \log \frac{\rho}{\rho_j} + (1-\rho)\log \frac{1-\rho}{1-\rho_j}$$

where m is the number of neurons in the hidden layer.

Loss Function of Sparse Autoencoder

The overall loss function including the regularization term now becomes

$$L_{new} = L + \lambda \sum_{j=1}^{m} \mathrm{KL}(\rho \| \rho_j)$$

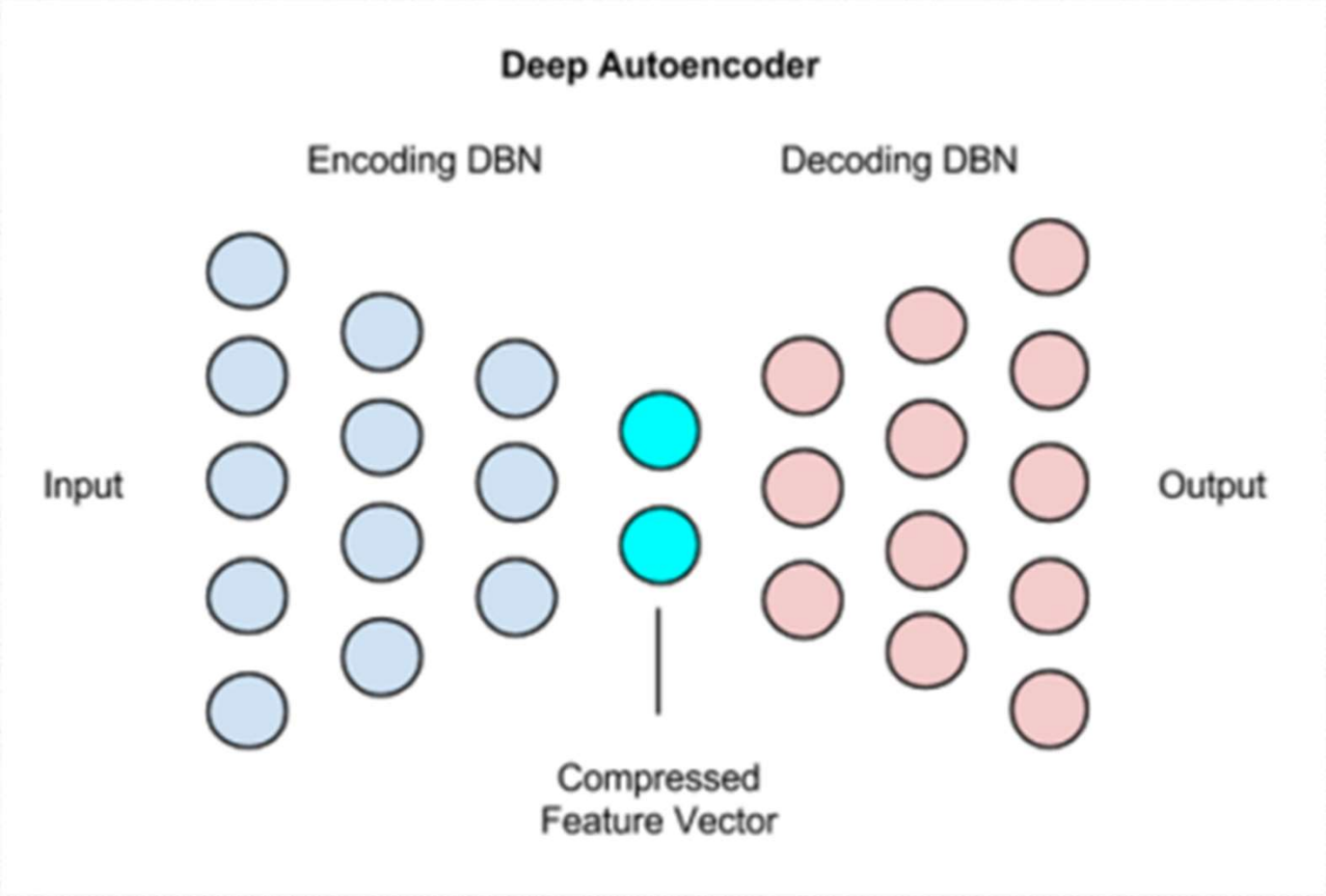
where λ is the weight parameter that controls the importance of the sparse regularization te
Here, loss function can be the mean squared error or the cross-entropy loss.

Deep Autoencoder

- The extension of the simple Autoencoder is the **Deep Autoencoder**.
- The first layer of the Deep Autoencoder is used for first-order features in the **raw input**. The second layer is used for second-order features corresponding to **patterns** in the appearance of first-order features. Deeper layers of the Deep Autoencoder tend to learn even higher-order features.

A **deep autoencoder** is composed of two, symmetrical deep-belief networks-

1. First four or five shallow layers representing the encoding half of the net.
2. The second set of four or five layers that make up the decoding half.

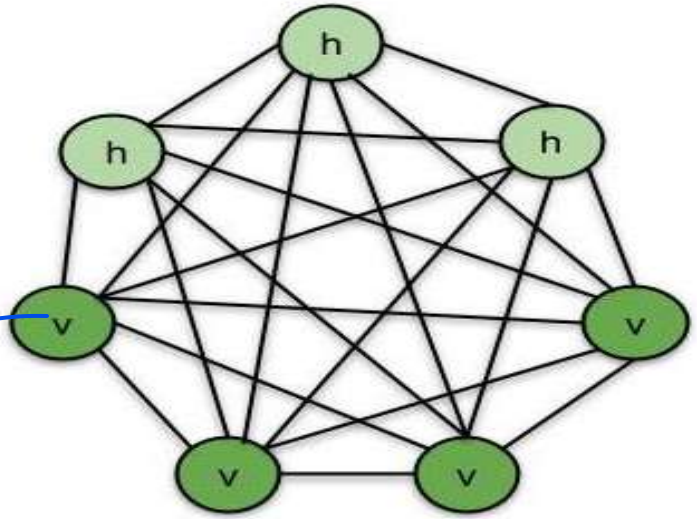

Deep Autoencoder
Encoding DBN
Decoding DBN
Input
Output
Compressed
Feature Vector

They are said to be symmetrically connected because they are undirected edges between the nodes and the weight from node i to node j is the same as the weight from nodej to node i. The input vectors in Boltzmann machines is binary vectors. They are said to be recurrent neural networks because they apply the same learning algorithm between the input nodes and the hidden nodes

What is Boltzmann Machine?

- **Boltzmann Machine** is a network of symmetrically connected, neuron like units that make stochastic decisions about whether to turn on or off.
- It is unsupervised DL model in which every node is connected to every other node. That is, unlike the ANNs, CNNs, RNNs the Boltzmann Machines are **undirected.**
- The main purpose of Boltzmann Machine is to optimize the solution of a problem. This method is used when the main objective is to create mapping and learn from the attributes and target variables in the data.
- The input vector of Boltzmann machines is binary vectors.

- There are two types of nodes in the Boltzmann Machine — **Visible nodes** — those nodes which we can and do measure, and the **Hidden nodes** – those nodes which we cannot or do not measure.
- Boltzmann machines help us understand abnormalities by learning about the working of the system in normal conditions.
- During the training phase, the visible neurons are clamped, the hidden layers always operate freely, they are used to explain underlying constraints in the environmental input vectors



v - visible nodes, h - hidden nodes

Disadvantages of Boltzmann machine

- The main disadvantage is that Boltzmann learning is significantly slower than backpropagation.
- Weight adjustment
- The time needed to collect statistics in order to calculate probabilities,
- How many weights change at a time

Restricted Boltzmann Machine

- RBM's are used to improve the accuracy in **collaborative filtering**.
- RBM is a special case of Boltzmann machine in which we have two layers, namely visible and hidden layer.
- The input vectors are not connected to each other and are connected to a set of hidden nodes.
- Similarly, the hidden nodes are not connected to one another. This is termed as ***Restricted.***

What is a collaborative filtering?

- Collaborative filtering is the predictive process behind recommendation engines.
- Recommendation engines analyze information about users with similar tastes to assess the probability that a target individual will enjoy something, such as a video, a book or a product.
- Collaborative filtering uses algorithms to filter data from user reviews to make personalized recommendations for users with similar preferences.
- Collaborative filtering is also used to select content and advertising for individuals on social media.

Example

- Boltzmann machine with four input node (v1,v2,v3,v4) and hidden nodes (h1,h2,h3)

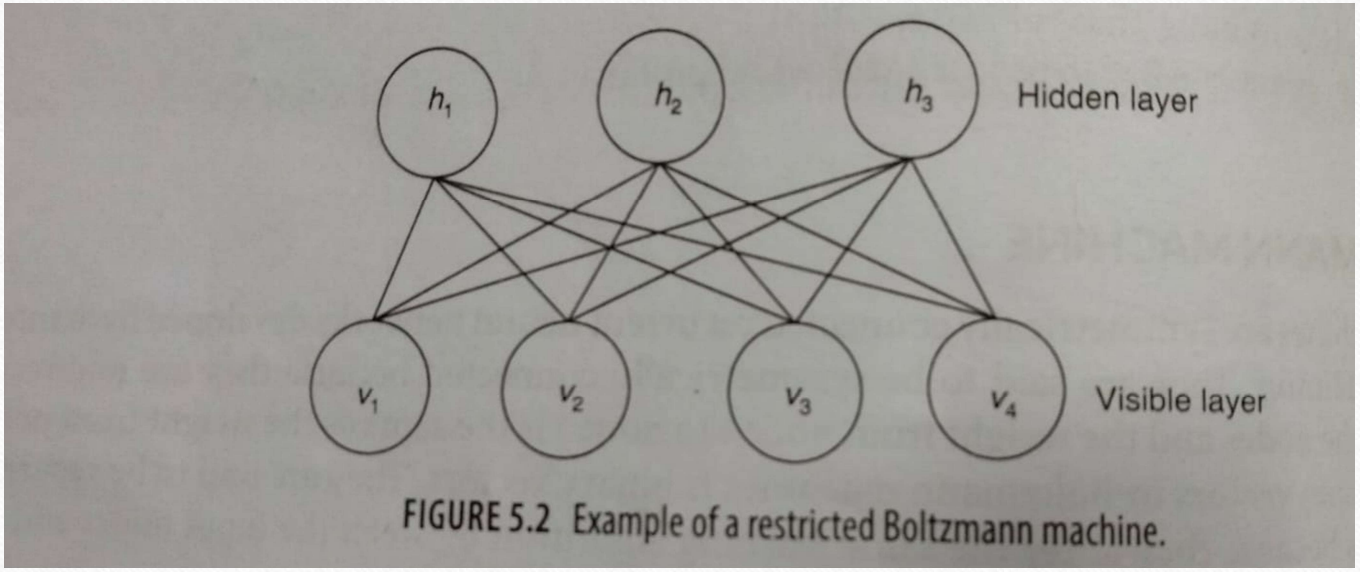


FIGURE 5.2 Example of a restricted Boltzmann machine.

Energy based model

- An energy-based model is **a probabilistic model governed by an energy function that describes the probability of a certain state.**
- The state of entire network is now given in terms of the energy of network, termed as **energy function**.
- Energy is determined by configuration of the variables. The variables associated with the RBM are the values in the visible node and values in the hidden node**.**
- The nature of the energy function is such that the energy becomes large when the compatibility between the variables is low.
- The objective is to increase the compatibility between the variables. That is the energy value is decreased iteratively till it reaches the local minima

Gibs Distribution

5.2.2 | Gibbs Distribution

The output of an RBM is a Boltzmann distribution also called as Gibbs distribution. The probability distribution function gives the probability with which a system given by x is in a certain state as a function of that state's energy and the temperature of the system. The probability distribution is given by

$$P(x) = \frac{e^{-E(x)}}{Z}$$

Z is the normalizing factor and is called the partition function. It is given by

$$Z = \sum_x e^{-E(x)}$$

In the case of an RBM, the input are the values in the visible and hidden neurons given by v and h, respectively. The probability that the system is in the state given by v and h is a joint probability and can be written as

$$P(v, h) = \frac{e^{-E(v,h)}}{Z}$$

where $Z = \sum_x e^{-E(v,h)}$. Here v is the input vector and h is the hidden layer vector. We consider all possible pairs between the visible and hidden units to calculate Z. The joint probability $P(v, h)$ is difficult to compute because the number of possible pairs between v and h would be $v \times h$. The Gibbs sampler is used to sample from $P(v, h)$ during the training phase.

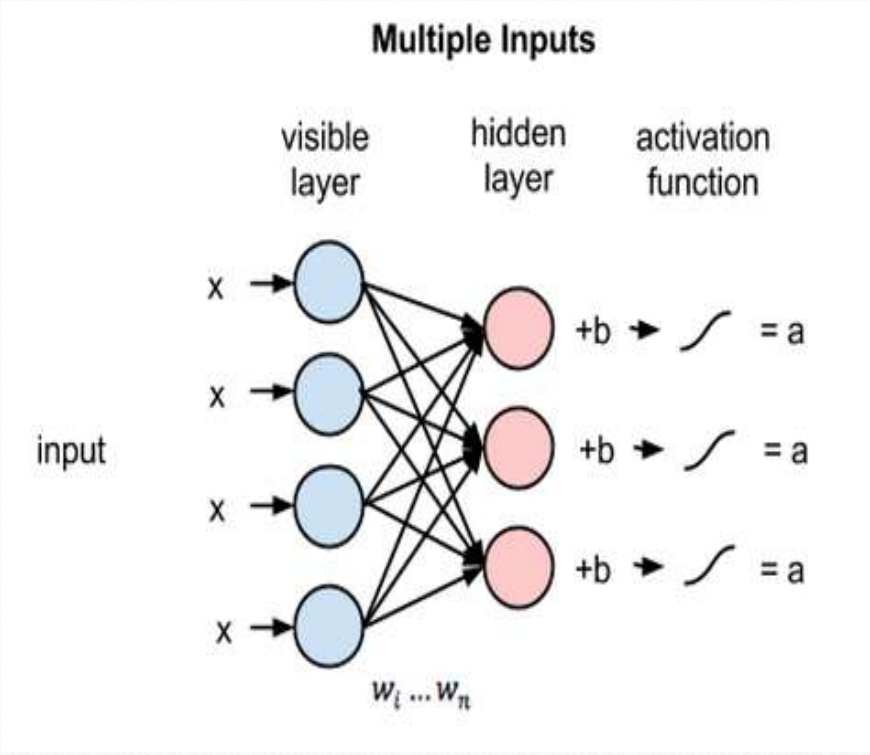
How do Restricted Boltzmann Machines work? – Gibbs Sampler

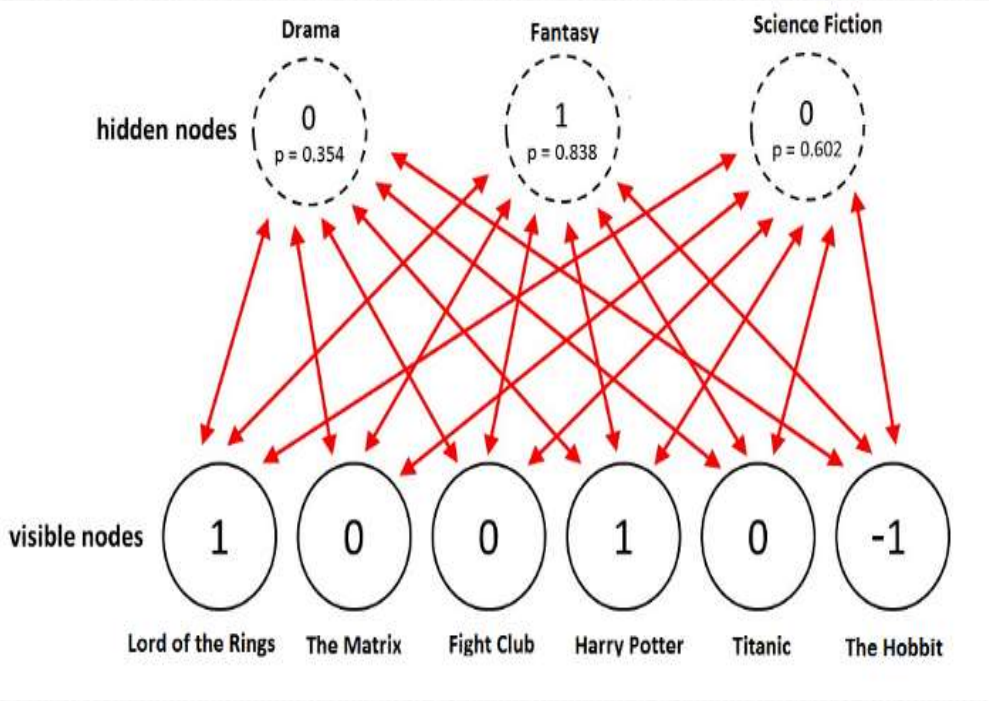
- **1st Phase:** we take the input layer and using the concept of weights and biased we are going to activate the hidden layer. This process is said to be Feed Forward Pass
- **2nd Phase:** As we don't have any output layer. Instead of calculating the output layer, we are reconstructing the input layer through the activated hidden state. This process is said to be Feed Backward Pass. We are just backtracking the input layer through the activated hidden neurons.

Feed Backward Equation:

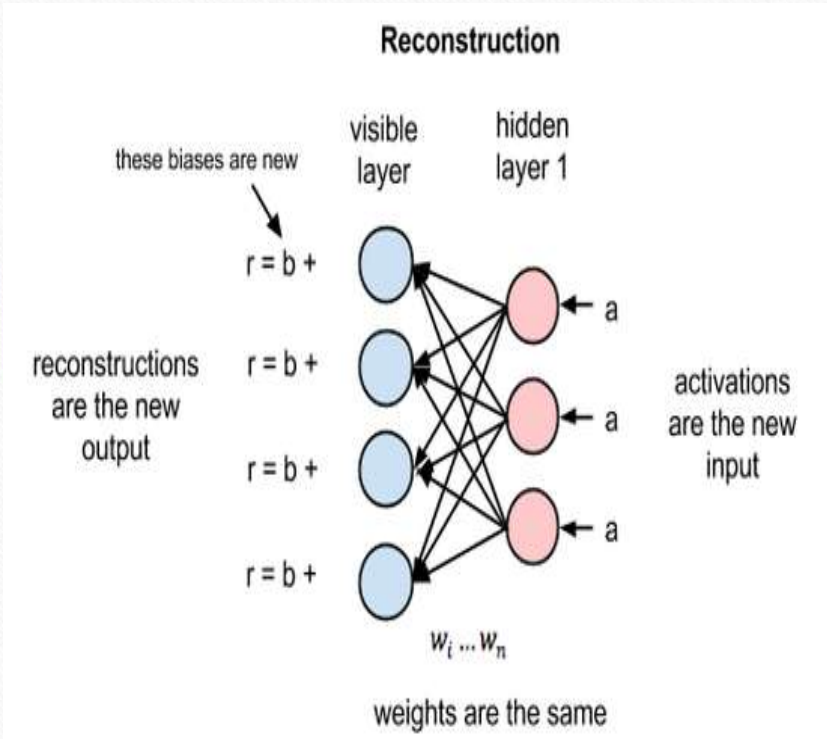
- **Error** = Reconstructed Input Layer-Actual Input layer
- **Adjust Weight** = Input*error*learning rate (0.1)

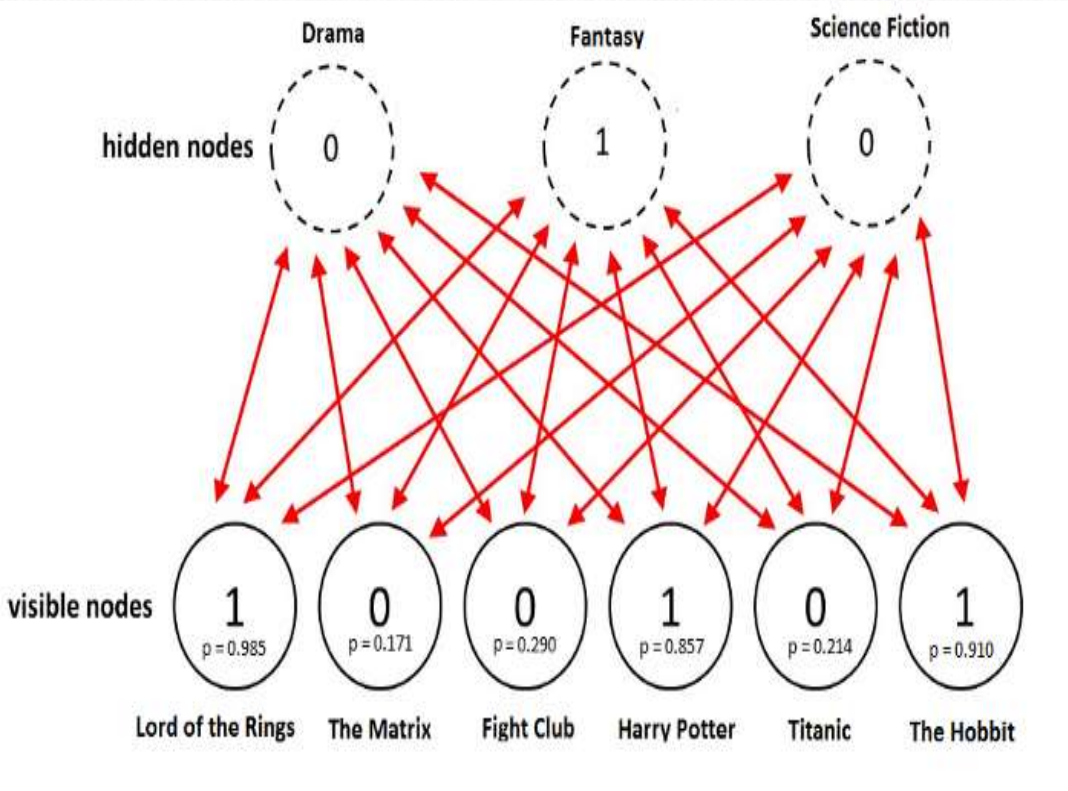
FEED FORWARD PASS





BACKWARD PASS





Gibs Sampler

The new vector created in the kth iteration is a recreation of the original input v_0. The structure of the RBM is such that there is no connection between the units of the visible layer or between the units in the hidden layer. So, the conditional probability can be written as

$$P(h \mid v) = \prod_j P(h_j \mid v)$$

$$P(v \mid h) = \prod_i P(v_i \mid h)$$

The vanilla RBM assumes the visible and hidden units to exist in one of the two states, namely, 0 and 1. That is, a neuron is said to be active if it is in state 1. The input vector (v) can be of dimension N and each dimension can take the binary value 0 or 1. Every hidden unit has a bias b_j associated with it. The weight between unit i and unit j is given by w_{ij}. The weights between the input vector and hidden vector (h) are initialized randomly. The probability that a hidden layer is active is given by

$$P(h_j = 1|v) = \frac{1}{1+e^{-\left(\sum_{i=1}^{N} w_{ij} v_i + b_j\right)}} = \sigma\left(\sum_{i=1}^{N} w_{ij} v_i + b_j\right)$$

where σ is the sigmoid function. The hidden layer is of dimension M and every visible unit has a bias c_i associated with it. Similarly, the probability that a visible node is active is given by

$$P(v_i = 1|h) = \frac{1}{1+e^{-\left(\sum_{j=1}^{M} w_{ij} h_j + c_i\right)}} = \sigma\left(\sum_{j=1}^{M} w_{ij} h_j + c_i\right)$$

Contrastive Divergence

- Contrastive Divergence is an algorithm to calculate maximum likelihood in order to estimate the slope of the graphical representation showing the relationship between the weights and its errors.
- Contrastive divergence approximates the gradient achieved when we take the difference in the probability distribution in the initial step and the probability distribution after k iterations
- Without contrastive divergence the hidden nodes in the restricted boltzmann machine can never learn to activate properly, this method is also used in the cases where the direct probability can’t be evaluated,

The output in the visible units is a probability distribution which may or may not reflect the actual input. The difference between the estimated probability distribution (Q) and the original probability distribution (P) is given by the Kullback–Leibler divergence. The similarity of the distribution P to the distribution Q is given by $D_{KL}(P||Q)$. Contrastive divergence approximates the gradient achieved when we take the difference in the probability distribution in the initial step and the probability distribution after k iterations. This is given as

$$CD_k = KL(P_0||P_\infty) - KL(P_k||P_\infty)$$

It was shown that the result would be the same even when k was as small as $k = 1$. The update matrix is given as the outer product of two probabilities.

$$\nabla w = v_0 \otimes P(h_0|v_0) - v_k \otimes P(h_k|v_k)$$

The new_weight is calculated as

$$\text{new\_weight} = \text{old\_weight} + \nabla w$$

5.3 | EXAMPLE

A group of dog lovers were asked to give their preference for the dog that they would like to buy. The preference is given as 0s and 1s for different dogs such as poodle, mastiff, bull dog, hound, terrier, shih tzu. Of these poodle and shih tzu are companion dogs, mastiff and bull dog are guard dogs, while hounds and terriers are hunting dogs. If a person gives the input vector (1, 0, 0, 0, 0, 1) the preference for companion dogs is evident. Let us see how RBM is used to find this person's latent preference for companion dogs.

We construct an RBM with six visible units representing the different dogs (poodle, mastiff, bull dog, hound, terrier, shih tzu) and three hidden units. An RBM is created for each user separately. A sample RBM state is given in Fig. 5.4.

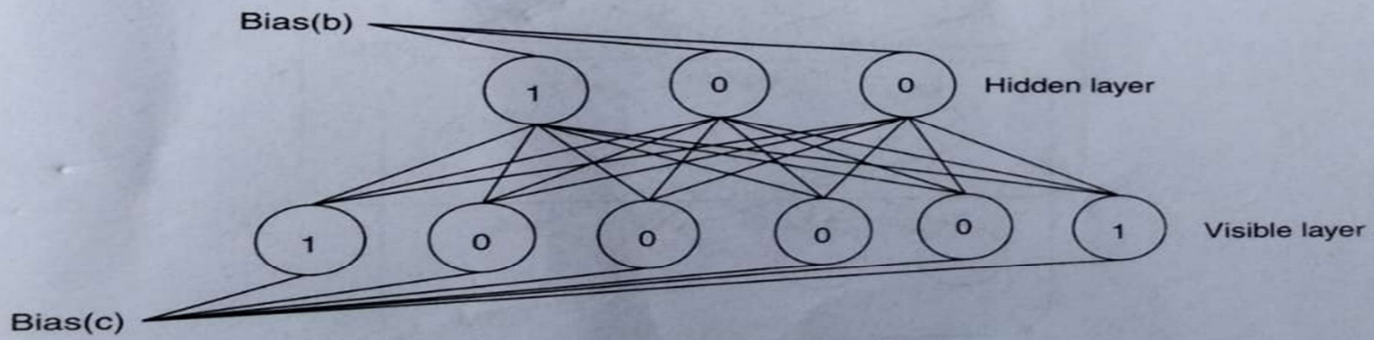


FIGURE 5.4 RBM example.

The final activation in the hidden layer is given as 0 or 1. But, it would be a probability value between 0 and 1. The probability value decides whether a node is activated or not. The feature latent in his/her selection of dog is activated in the hidden layer.

Types of RBM

- Based on the distribution used and the structure of the hidden layers, many types of RBM are possible

1. Bernoulli – Bernoulli RBM: the units in RBM considered so far are random variables taking by binary values. The probability density function is conditioned to Bernoulli distribution. This is known as Bernoulli – Bernoulli RBM where the visible and hidden units are modelled using the Bernoulli distribution

Types of RBM

- 2. Gaussian – Bernoulli RBM: In this type Visible units as Gaussian and hidden units are Bernoulli. This allows the visible units to take real valued input that are modelled using a normal distribution.
- 3. Conditional RBM: in this, the visible units are modelled using the Gaussian distribution and the hidden units are rectified linear unit transformation. Using binary values in the hidden layer restricts the number of latent features that can be represented.

Types of RBM

Deep Belief Network:

- When the features can be represented as a hierarchy DBN is used. They stack RBMs to represent the features of the training data as a hierarchy.
- The training in DBN is a greedy layer wise unsupervised training.
- During training the first RBM trained has the input vector in the visible layer and the first hidden layer. The process is repeated till the desired condition is met. Each layer has a higher level representation when compare to previous layer

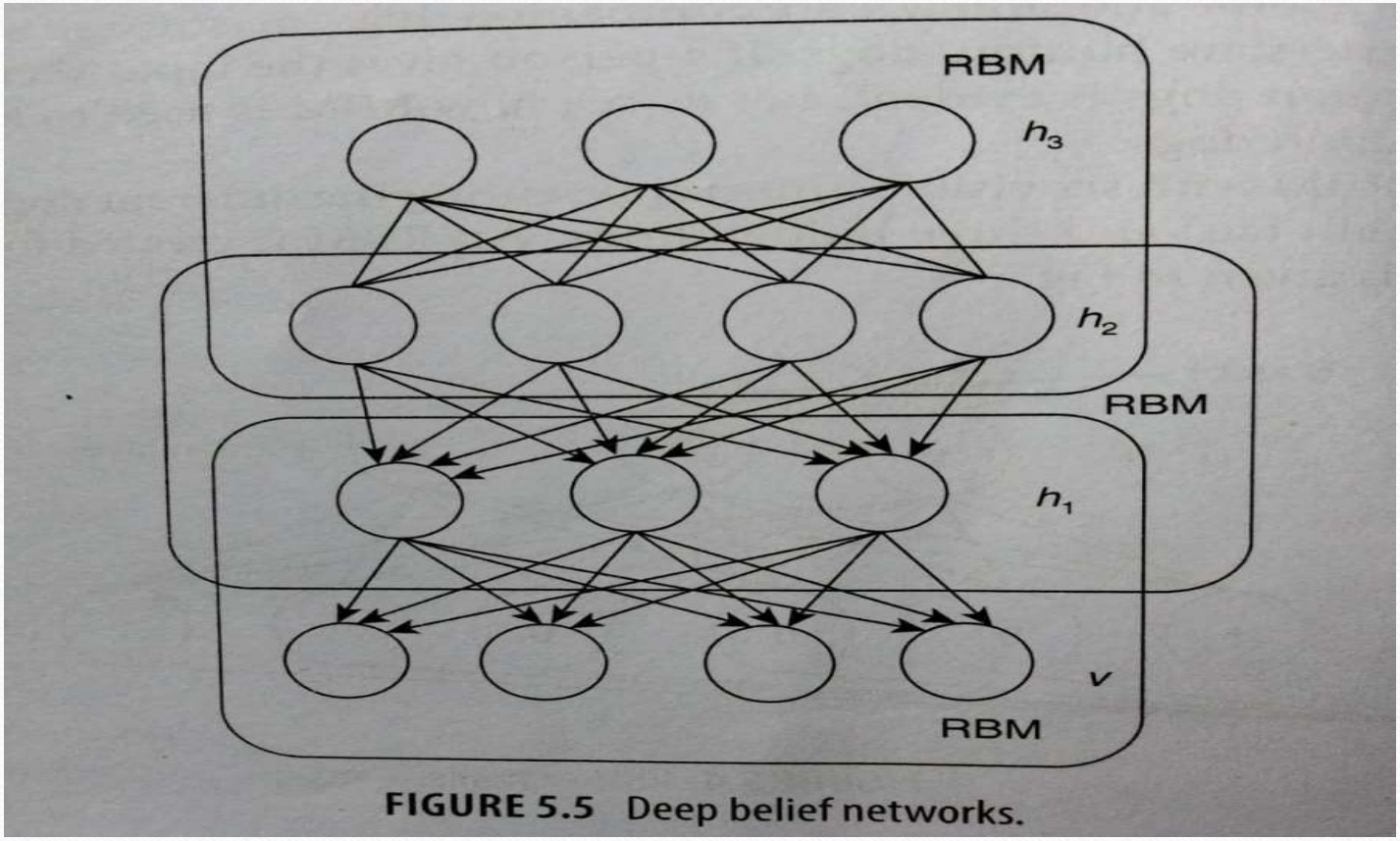


FIGURE 5.5 Deep belief networks.

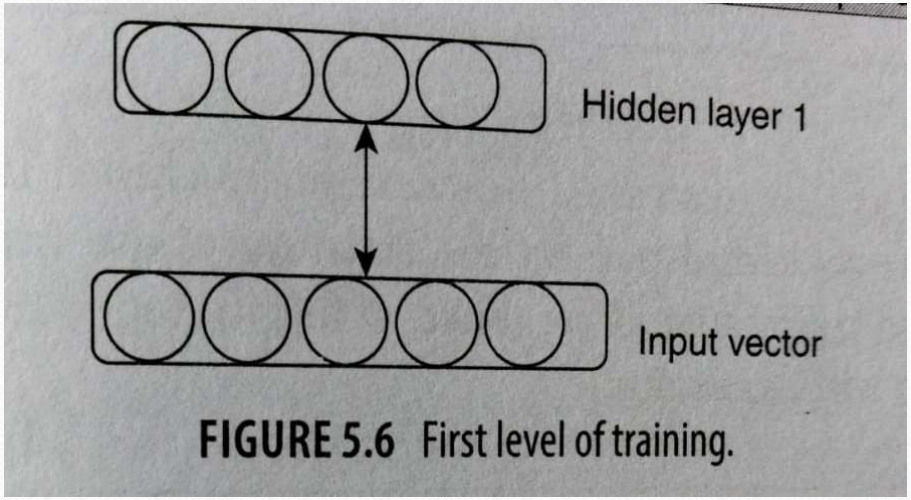


FIGURE 5.6 First level of training.

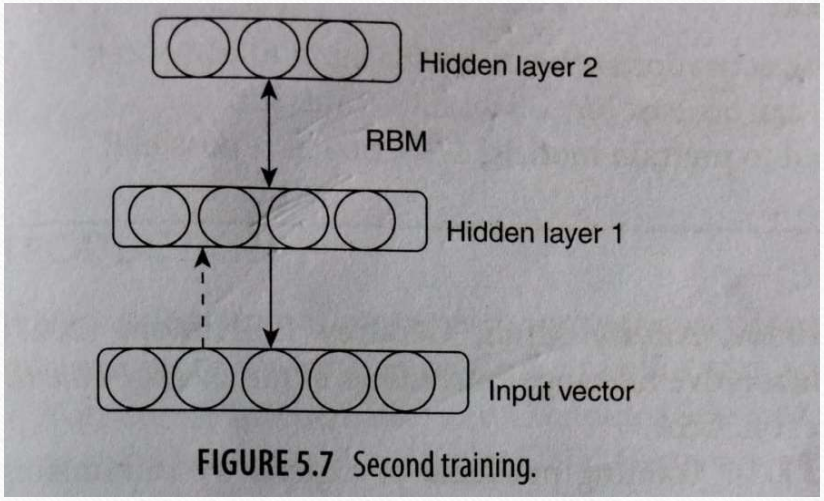


FIGURE 5.7 Second training.

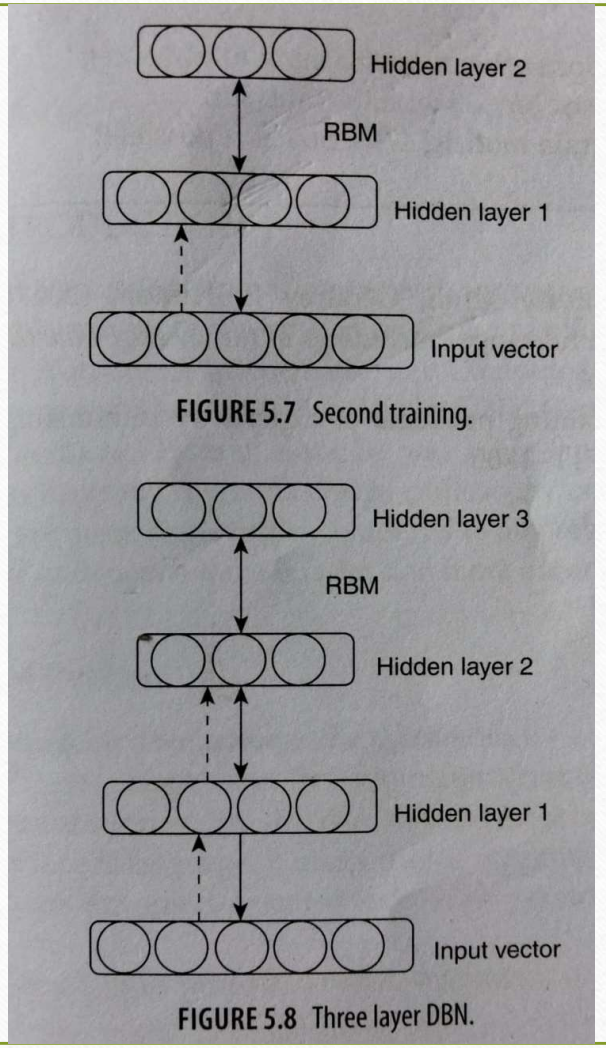


FIGURE 5.7 Second training.

FIGURE 5.8 Three layer DBN.