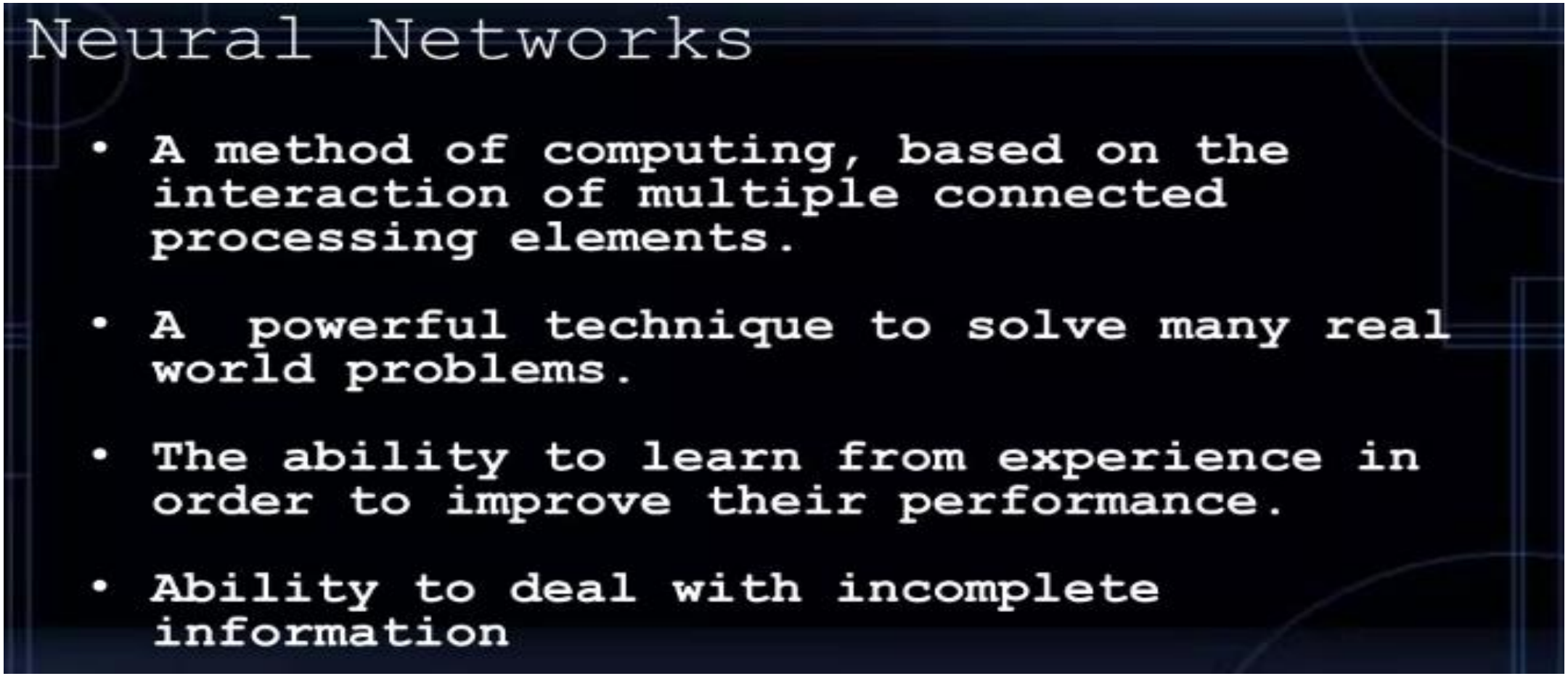
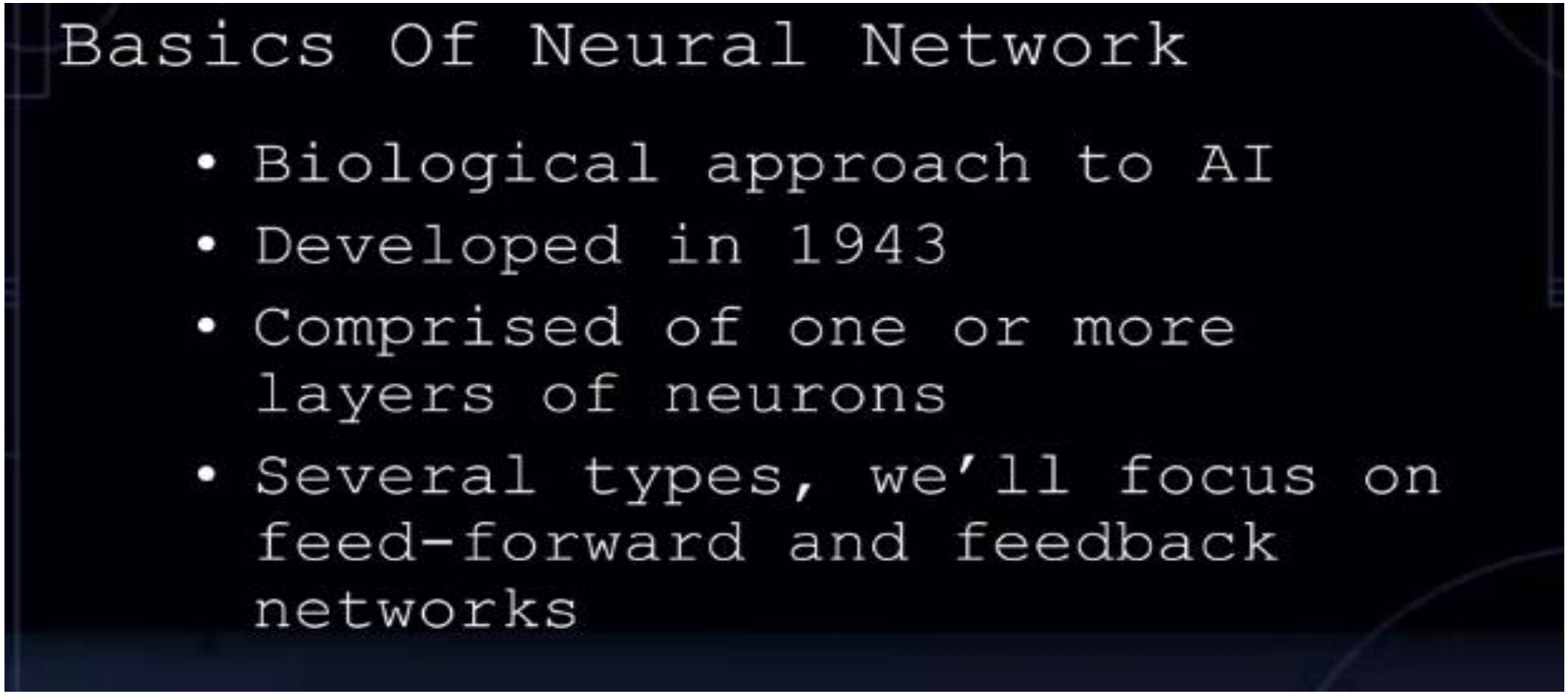
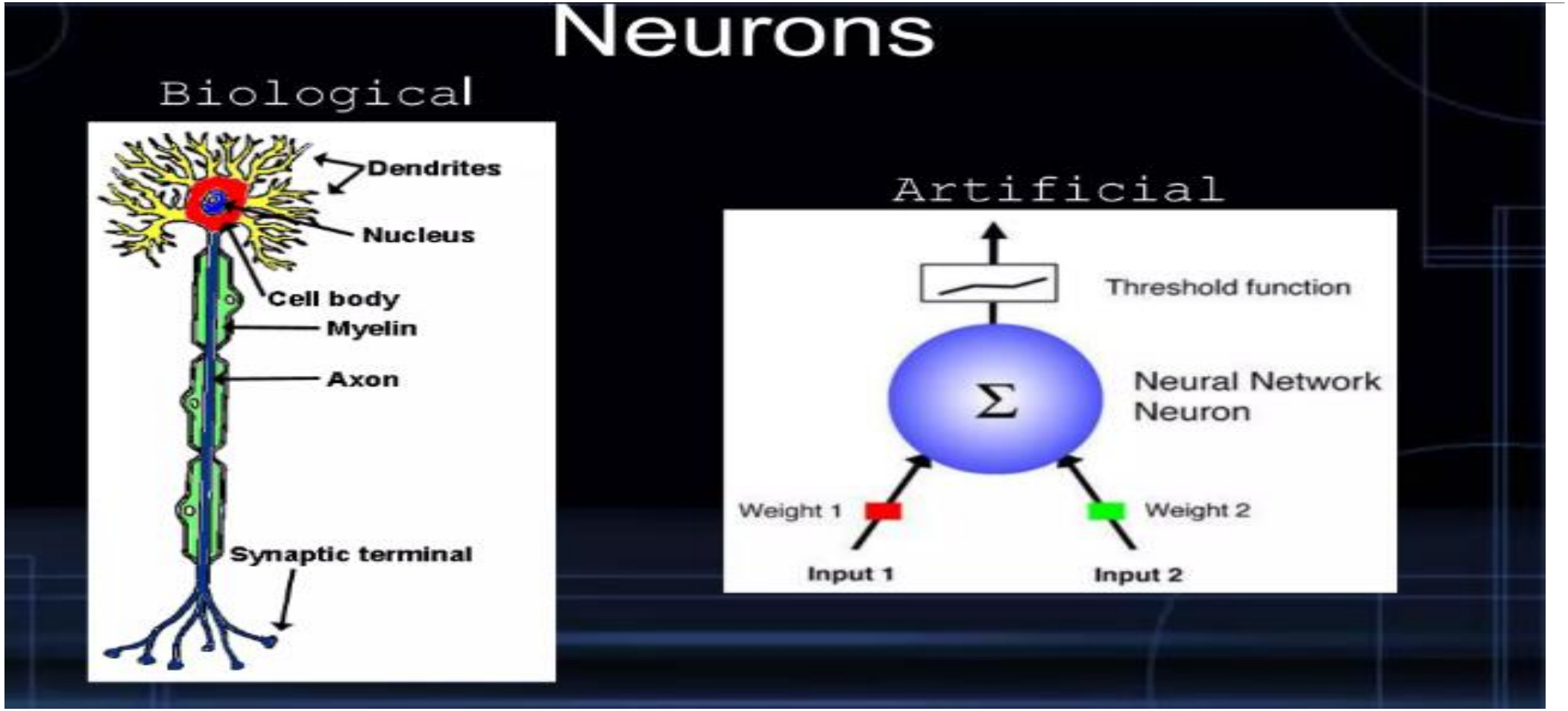
Introduction to Neural Networks



Introduction to Neural Networks….

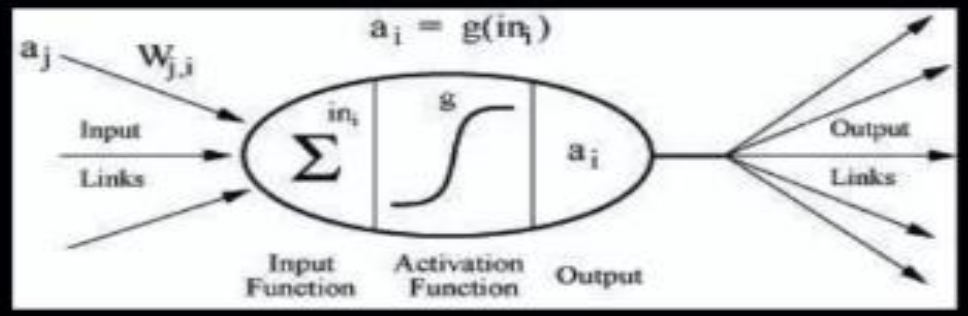


Introduction to Neural Networks….



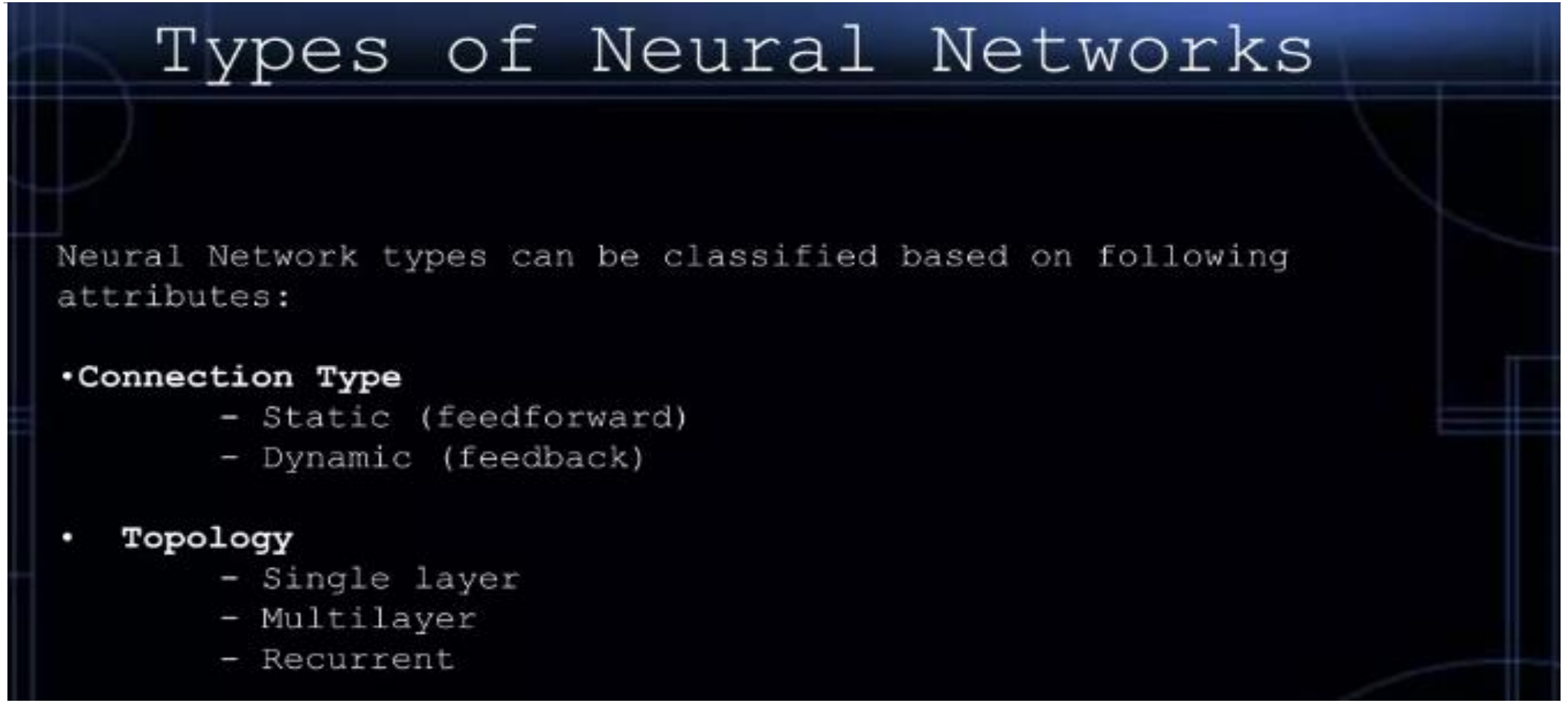
Introduction to Neural Networks….

Neural Network Neurons



- Receives n-inputs
- Multiplies each input by its weight
- Applies activation function to the sum of results
- Outputs result

Introduction to Neural Networks….



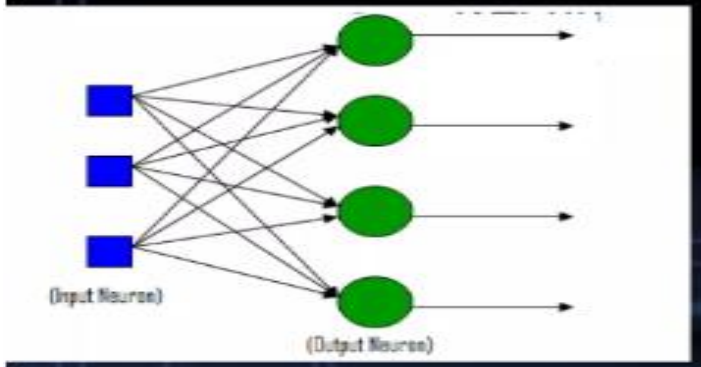
Introduction to Neural Networks….

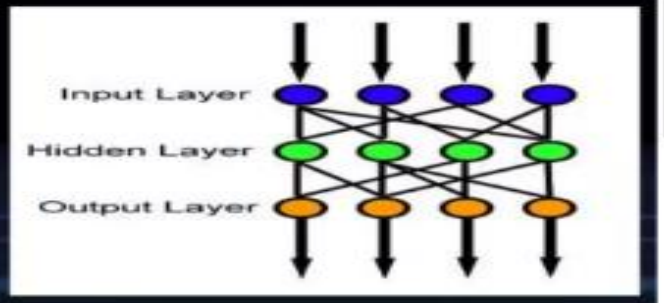
Classification Based On Connection Types

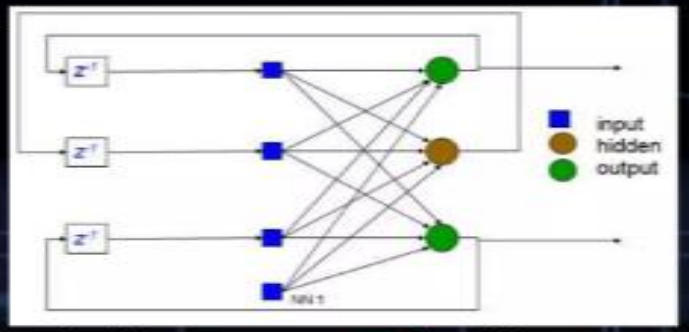
- Static(Feedforward)
- Dynamic(Feedback)

Classification Based On Topology

- Single layer
- Multilayer
- Recurrent







(unit delay operator z^{-1} implies dynamic system)

NETWORK ARCHITECTURE

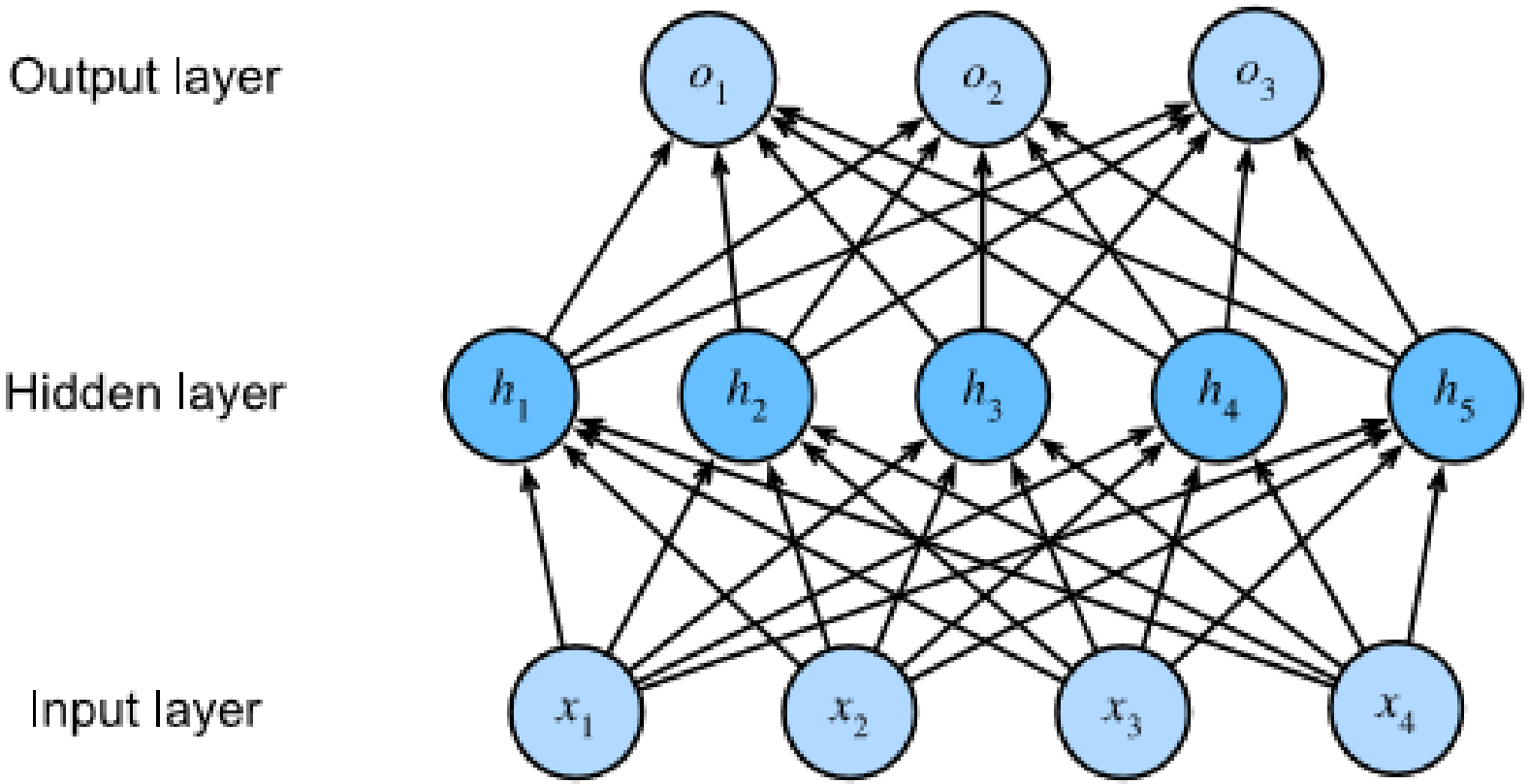
Three Fundamentally different classes of network Architectures

- ✓Single layer feedforward
- ✓Multi layer feedforward
- ✓Recurrent network

Single Layer Neural Networks

- In Single layer we have an input layer of source nodes that projects onto an output layer of neurons ,but not vice vers .
-
- This network is a **feedforward or acrylic type.**
- Consists of only the input and the output layers. Such a network is called a single-layer network.
- We do **not count the input layer of source nodes because no computation is performed there.**

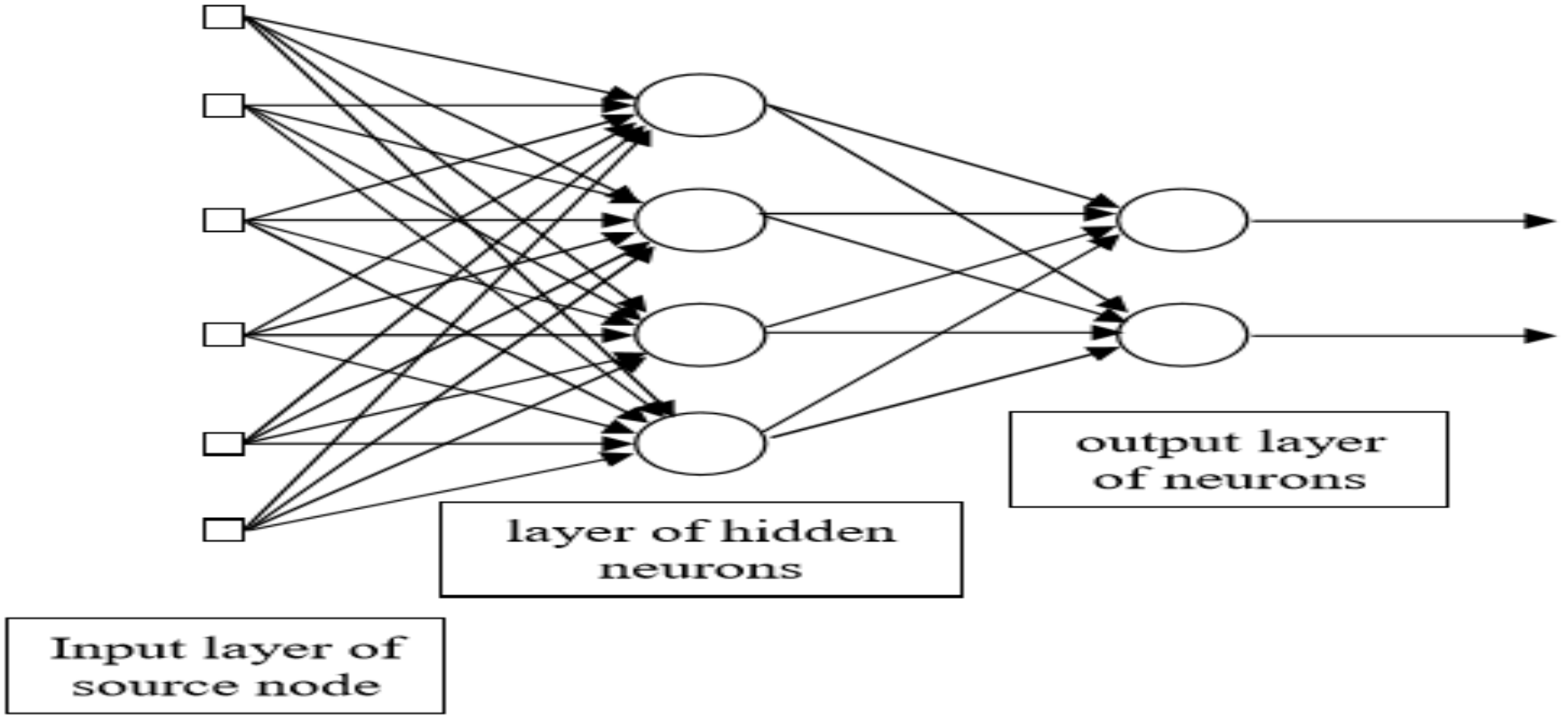
Multilayer Perceptron



Multi-layer Feedforward Networks

- The source nodes in the input layer of the network constitute the input signals applied to the neurons in the 2nd. Layer (i.e.,hidden layer).
- The output signals of the second layer are used as input to the 3rd. layer, and so on for the rest of the network.

Multi-layer Feedforward Networks…



Home Work

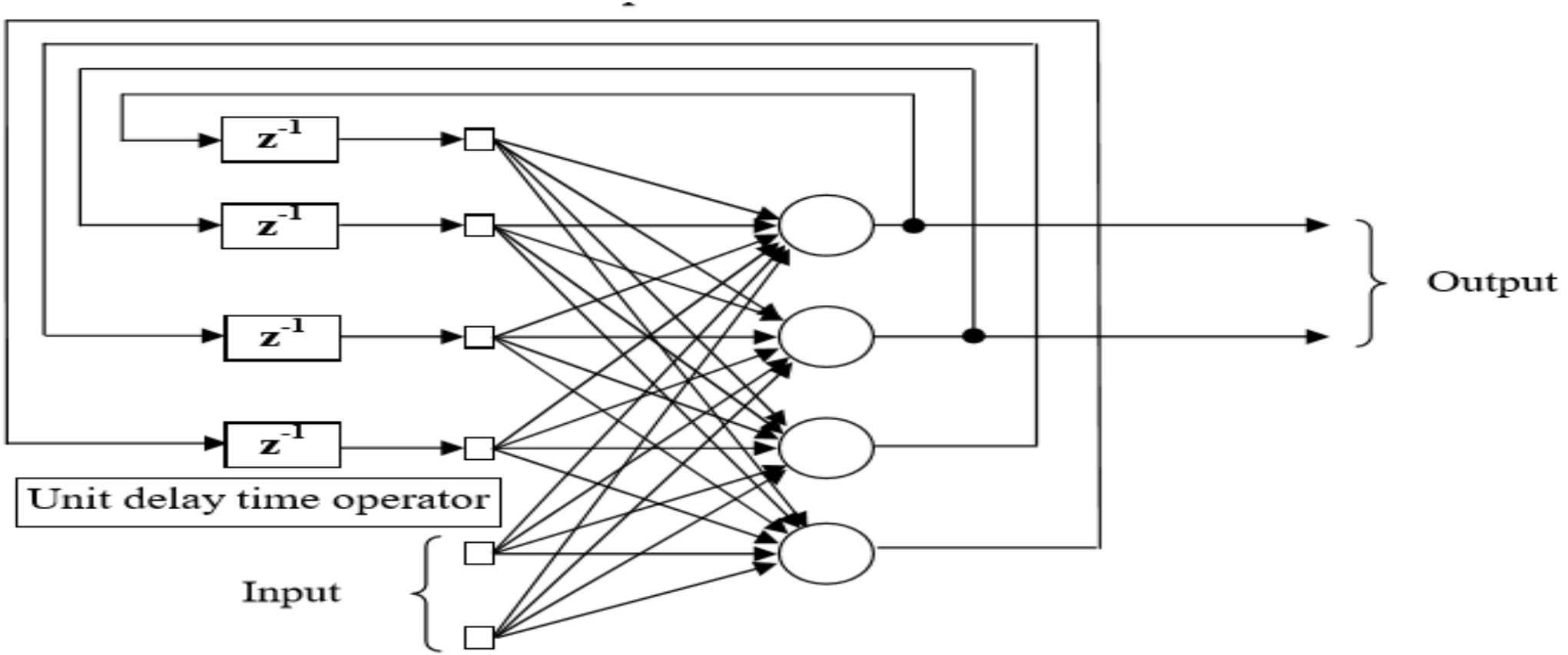
Implementation of Multilayer perceptron refer page number 138-139

- Text book -1

Recurrent Networks

- A recurrent network distinguishes it self from a feedforward neural network in that it has **at least one feedback loop.**
- The feedback connections shown in the figure originate from the hidden neurons as well as from the output neurons.

Recurrent Networks



Activation Functions

Depending on the activation function you pick, you will find that some objective functions are more appropriate for different kinds of data.

Different Types of Activation Functions:-

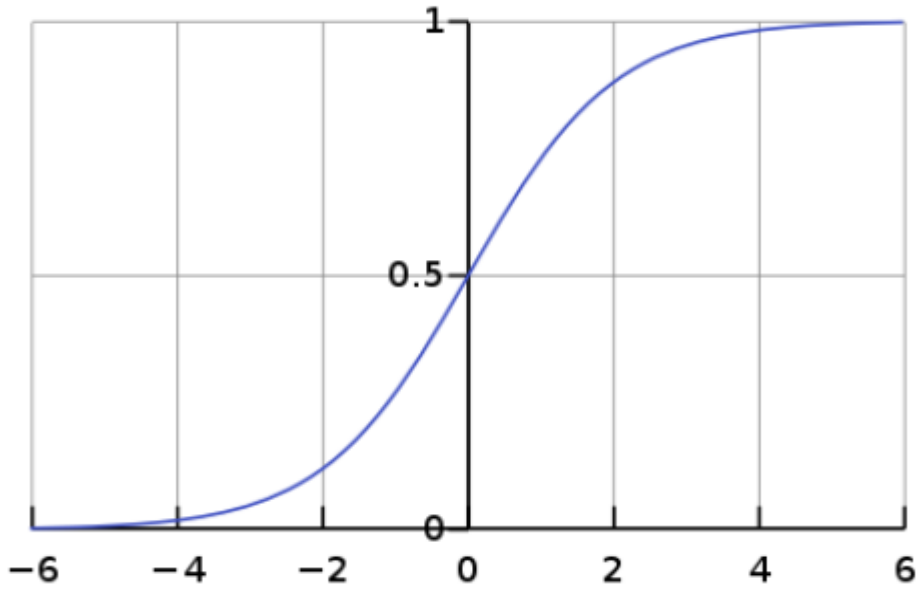
- Sigmoid
- Tanh
- Hard Tanh
- ReLu (Rectified Linear Unit)

Note: Students are requested to read the text books and reference books for detail study. Lecture slides will give only brief idea about the topics.

SIGMOID

A **sigmoid function** is a mathematical function having a characteristic "S"-shaped curve or **sigmoid curve**. A common example of a sigmoid function is the logistic function .

$$S(x) = \frac{1}{1 + e^{-x}}$$

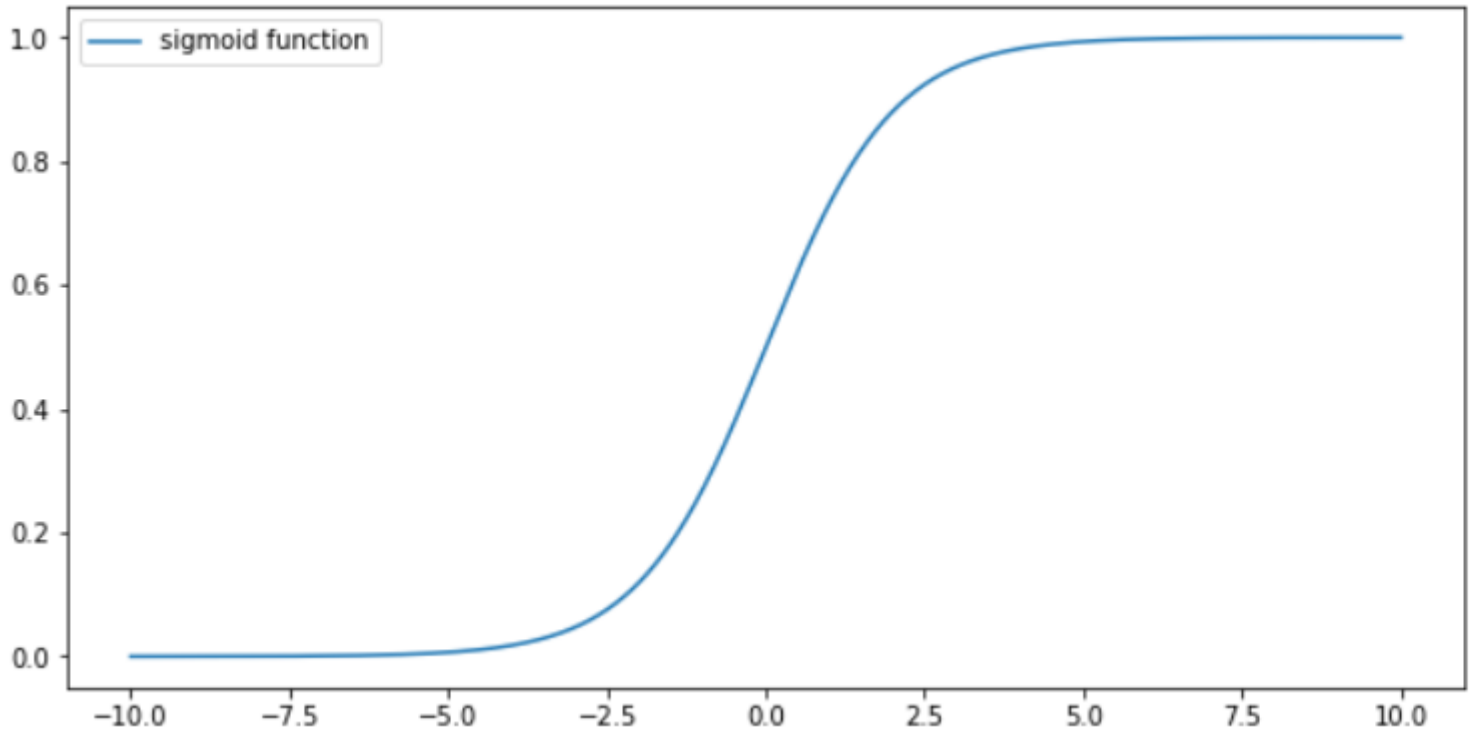


Write a python program to plot Sigmoid Function

```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
x = np.linspace(-10, 10, 1000)
y = 1 / (1 + np.exp(-x) )

plt.figure(figsize=(10, 5))
plt.plot(x, y)
plt.legend(['sigmoid function'])
plt.show()
```

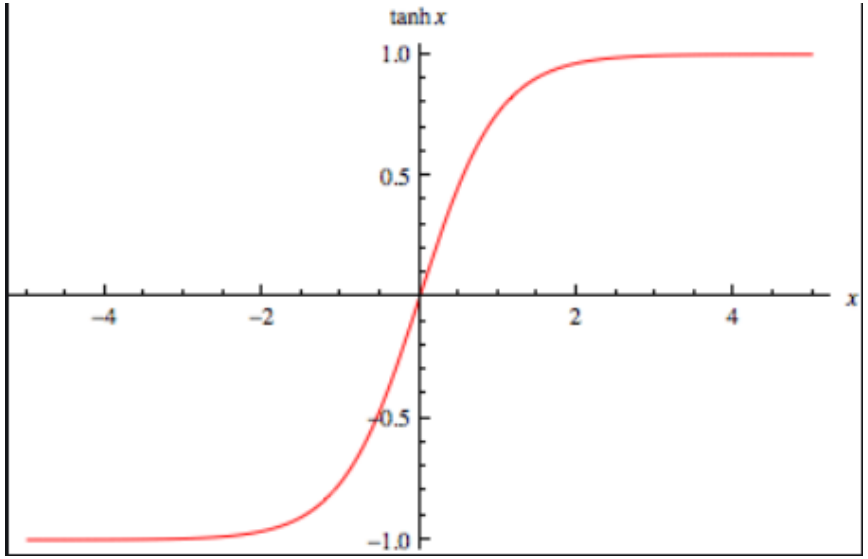
Result



Tanh

Hyperbolic tangent functions are analogues of the ordinary trigonometric functions, but defined using the hyperbola.

$$g(x) = \frac{e^x}{1 + e^x}$$



Tanh…

- Hyperbolic tangent **function**, is a rescaling of the logistic sigmoid, such that its outputs range from -1 to 1.

Hard Tanh

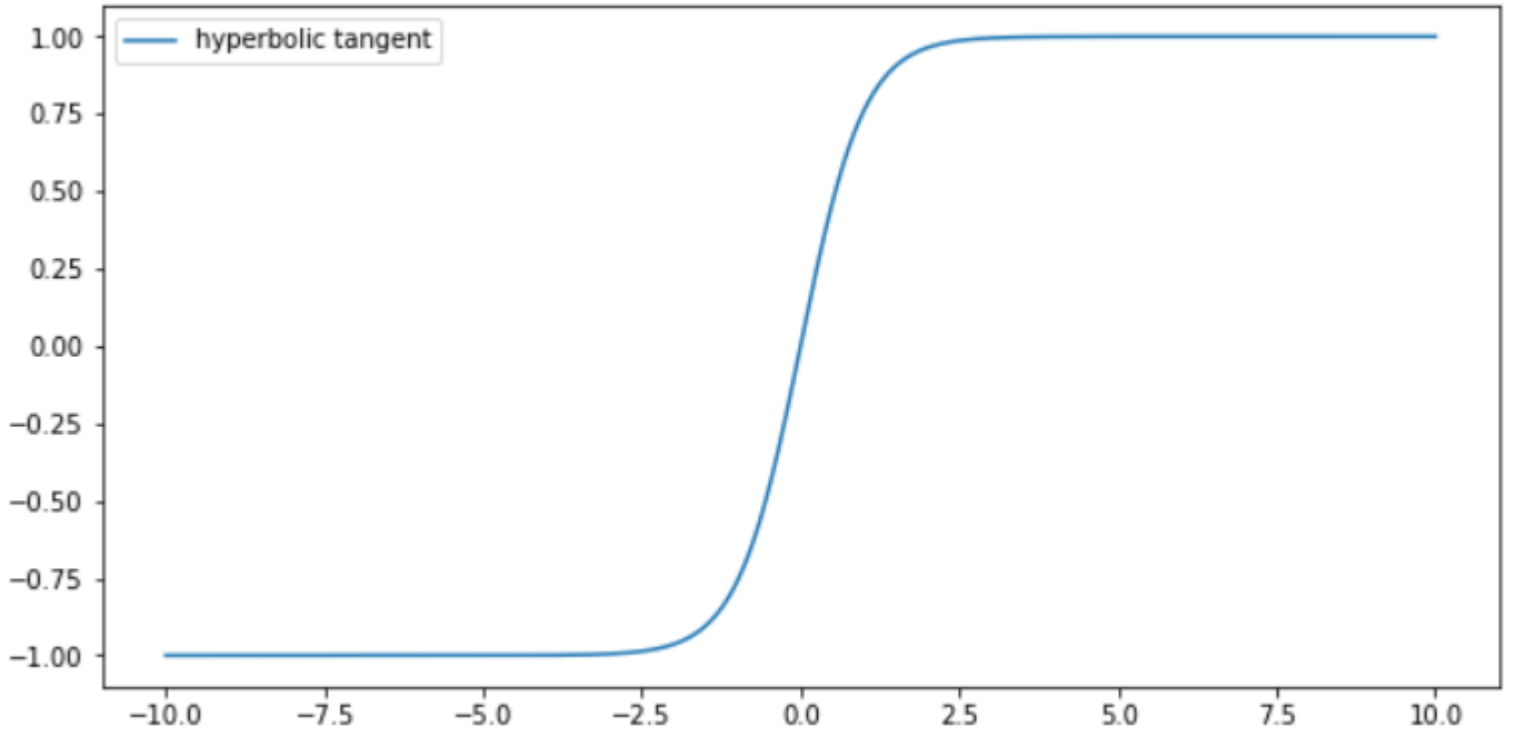
- Anything more than +1 is made to +1 and Anything less than -1 is made to -1.

Write a python program to plot tanh Function

```
import numpy as np
import matplotlib.pyplot as plt
x = np.linspace(-10, 10, 1000)
y = ( 2 / (1 + np.exp(-2*x) ) ) -1

plt.figure(figsize=(10, 5))
plt.plot(x, y)
plt.legend(['hyperbolic tangent'])
plt.show()
```

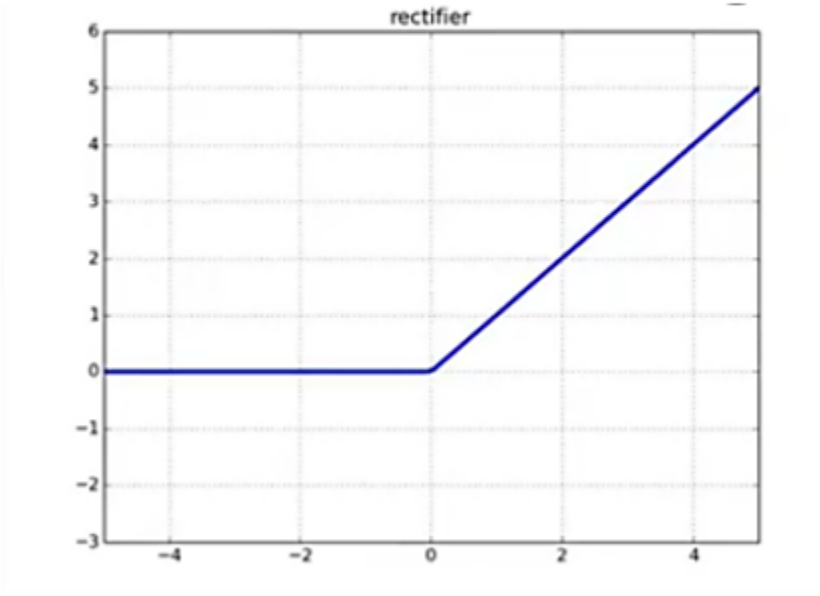
Result


hyperbolic tangent
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0

ReLU(Rectified Linear unit)

The **ReLU** is an activation function defined as the positive part of its argument:

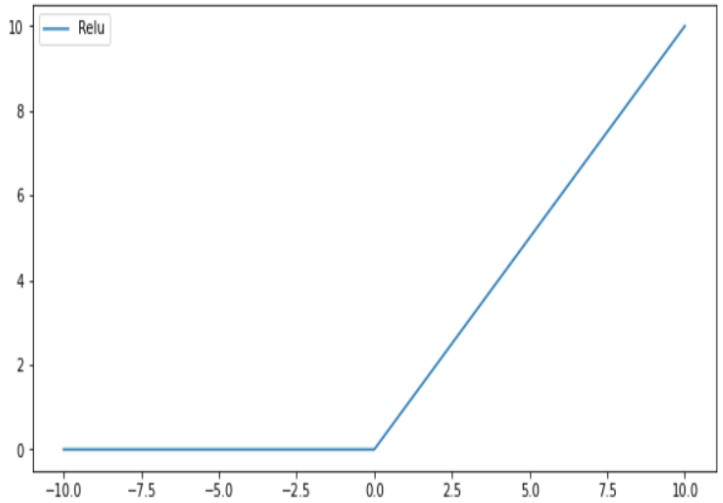
$$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \Rightarrow 0 \end{cases}$$



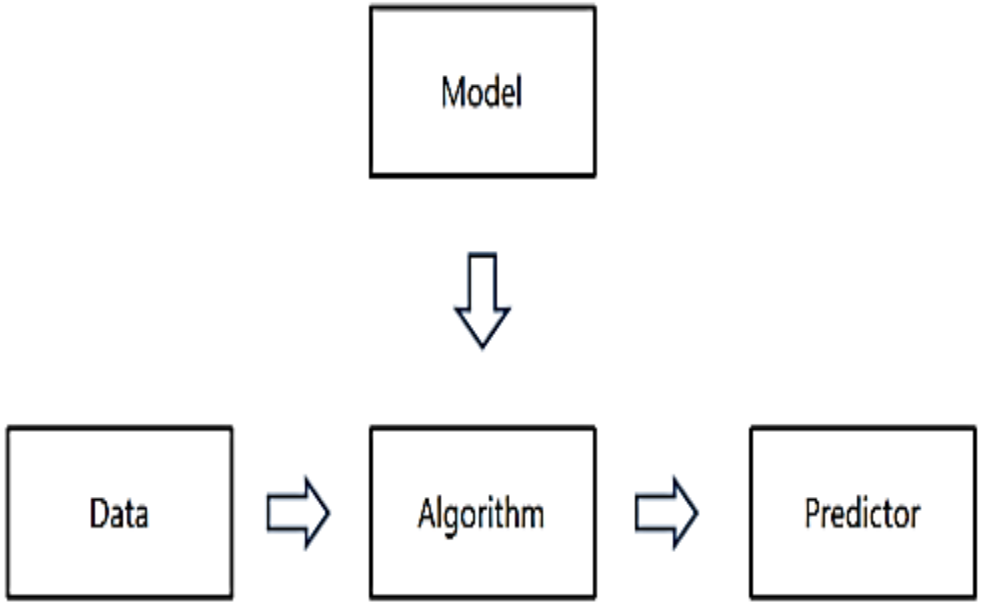
Write a python program to plot tanh Function

```
x = np.linspace(-10, 10, 1000)
y = np.maximum(0, x)
plt.figure(figsize=(10, 5))
plt.plot(x, y)
plt.legend(['Relu'])
plt.show()
```

Result



Machine Learning Framework



Model selection

We do not want our model to say“That’sBob! I remember him! He has dementia!” The reason why is simple.

When we deploy the model in the future, we will encounter patients that the model has never seen before.

Our predictions will only be useful if our model has truly discovered a general pattern.

We access just a small sample of data. The largest public image data sets contain roughly one million images.

Model selection…

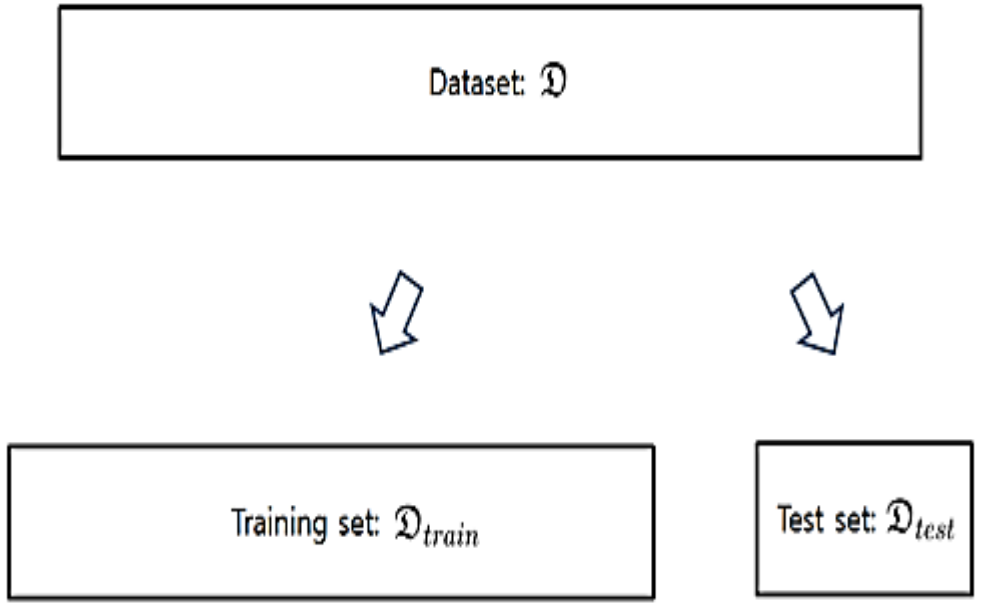
More often, we must learn from only thousands or tens of thousands of data examples.

In a large hospital system,we might access hundreds of thousands of medical records.

When working with finite samples,we run the risk that we might discover apparent associations that turn out not to hold up when we collect more data

Testing and Training the Dataset

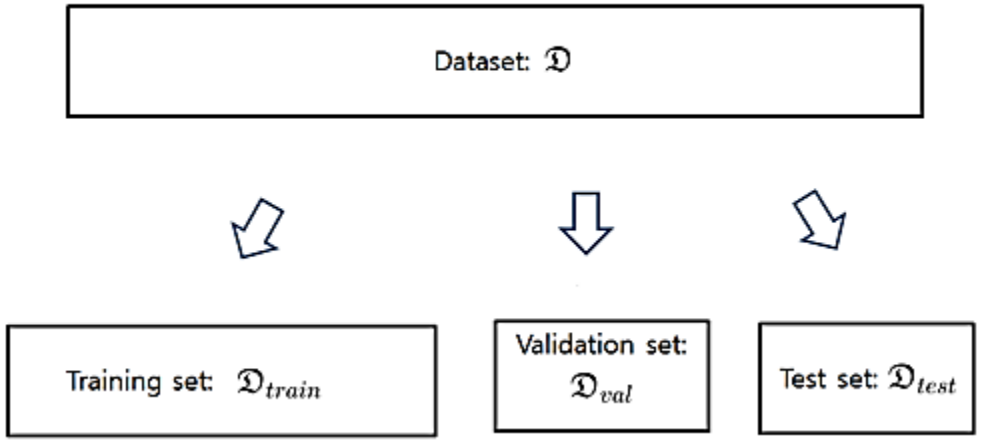
A good predictor must satisfy dual criteria. It has to perform well for the given dataset D, but it also has to work well for similar future data.



Validation and model selection

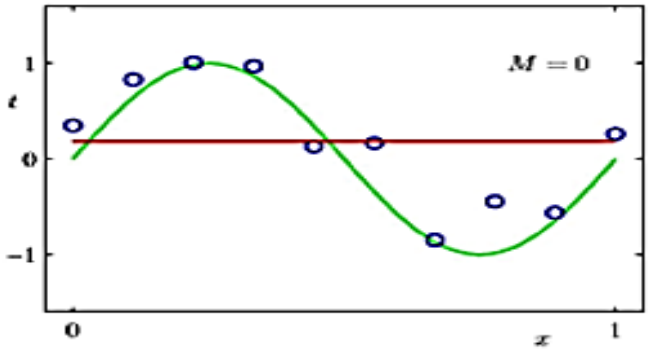
The above simple scheme of training and testing is good for estimating the performance of a predictor.

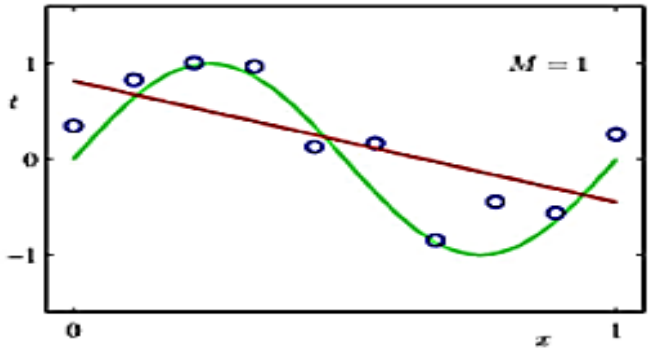
when there are several models to choose from, we need a different scheme.



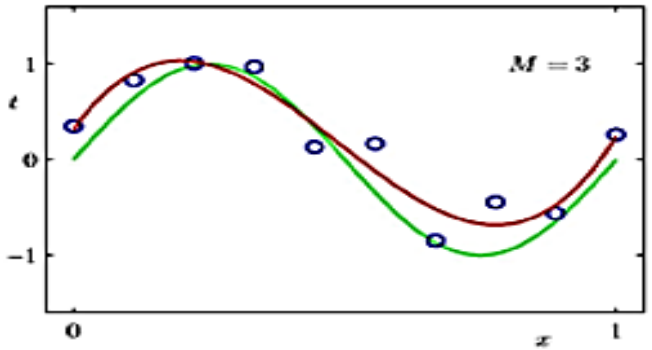
Under fit, Best fit and Overfit

– Red lines are best fits with $M = 0,1,3,9$ and $N=10$

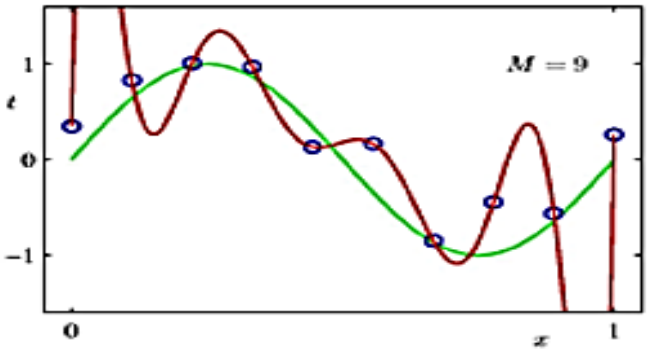




← Poor representations of $\sin(2\pi x)$



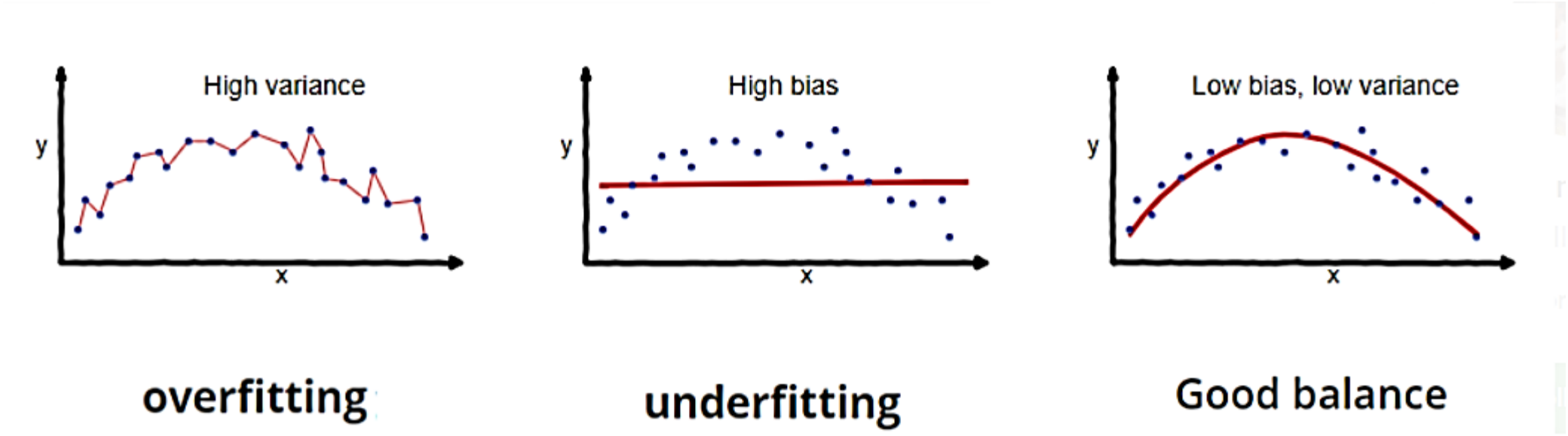
← Best Fit to $\sin(2\pi x)$



Over Fit
Poor representation of $\sin(2\pi x)$

Bias/Variance Trade -off

There are two sources of error, namely Bias and Variance, which acts as a hindrance for any algorithm to generalise.



Loss Function

Loss functions quantify how close a given neural network is to the ideal toward which it is training.

Steps to calculate the Loss:

1. Calculate a metric based on the error we observe in the network's predictions.
2. Aggregate these errors over the entire dataset and average them and compare the value with the ideal case.
3. Adjust the parameters(Weights) to reduce the Loss due to error.

Loss Function Notation….

The loss function notation indicates that its value depends only on W and b, the weights and the biases of the neural network.

Types of Loss Functions:

- Squared loss
- Logistic loss
- Hinge loss
- Negative log likelihood

Loss Function Notation

- Consider the dataset gathered to train a neural net. Let "*N*" denote the number of samples (set of inputs with corresponding outcomes) that have been gathered.

- Consider the nature of the input and output collected. Each data point records some set of unique input features and output features. Let "*P*" denote the number of input features gathered and "*M*" denote the number of output features that have been observed.

Loss Function Notation….

- We will use (X,Y) to denote the input and output data we collected. Note that there will be N such pairs where the input is a collection of P values and the output Y is a collection of M values. We will denote the *ith* pair in the dataset as X_i and Y_i.

- We will use $\hat{Y}$ to denote the output of the neural net. Of course, $\hat{Y}$ is the network's guess at Y and therefore it will also have M features.

Loss Function Notation….

- We will use (X,Y) to denote the input and output data we collected. Note that there will be N such pairs where the input is a collection of P values and the output Y is a collection of M values. We will denote the *ith* pair in the dataset as X_i and Y_i.

- We will use $\hat{Y}$ to denote the output of the neural net. Of course, $\hat{Y}$ is the network's guess at Y and therefore it will also have M features.

Loss Function

Loss Functions for Regression:

Loss function in regression is the least squares

$$L(W, b) = \frac{1}{N}\sum_{i=1}^{N}\left(\hat{Y}_i - Y_i\right)^2$$

$\hat{Y}_i$ Y_i Where is the desired output

is the predicted output

N is the size of the dataset

In above case we have considered we have only only one output feature(M=1).

Loss Function…

When We have more than one output features the loss function is:

$$L(W, b) = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{M}\sum_{j=1}^{M}(\hat{y}_{ij} - y_{ij})^2$$

When M=N

$$L(W, b) = \frac{1}{2N}\sum_{i=1}^{N}\sum_{j=1}^{M}(\hat{y}_{ij} - y_{ij})^2$$

Loss Function…

Loss Functions for Classification :

We can build neural networks to bin data points into different categories; for example, fraud or not fraud.

0= no fraud and 1 =fraud, which by convention is called a 0-1 classifier.(Binary classifier)

- Hinge loss is the most commonly used loss function when the network must be optimized for a hard classification.
- Hinge loss is also seen in a class of models called maximum-margin classification models (e.g., support vector machines)

Loss Function….

(Soft classifiers: Estimate the class conditional probabilities and then perform classification based on estimated probabilities.

Hard classifiers: Directly target on the classification decision boundary without producing the probability estimation.).

Hing Loss equation:

$$L(W, b) = \frac{1}{N}\sum_{i=1}^{N} max(0, 1 - y_{ij} \times \hat{y}_{ij})$$

Logistic Loss:

- Logistic loss functions are used when probabilities are of greater interest than hard classifications.
- Let us consider the case in which our network predicts a probability for two classes, like the fraud and not fraud 0–1 classifier
- For the given set of bias **b** and weight **w** .The probability of 1 and 0 :-

$$P(y_i = 1 \mid X_i; \mathbf{W}, \mathbf{b}) = h_{\mathbf{W},\mathbf{b}}(X_i)$$

$$P(y_i = 0 \mid X_i; \mathbf{W}, \mathbf{b}) = 1 - h_{\mathbf{W},\mathbf{b}}(X_i)$$

We can combine these equations and express them as follows:

$$P(y_i \mid X_i; \mathbf{W}, \mathbf{b}) = (h_{\mathbf{W},\mathbf{b}}(X_i))^{y_i} \times (1 - h_{\mathbf{W},\mathbf{b}}(X_i))^{1-y_i}$$

Negative log likelihood

For the sake of mathematical convenience, when dealing with the product of probabilities ,convert the probabilities into log of probabilities.

The product of the probabilities transforms to the sum of the log

of the probabilities.

$$L(W, b) = -\sum_{i=1}^{N} y_i \times \log \hat{y}_i + (1 - y_i) \times \log (1 - \hat{y}_i)$$

If there are M classes then the equation for negative log likelihood will change to :-

$$L(W, b) = -\sum_{i=1}^{N}\sum_{j=1}^{M} y_{i,j} \times \log \hat{y}_{i,j}$$

This is the cross-entropy between two probability distributions.

in this case, what we predict and what we have observed under the same criteria

Introduction to Deep Learning, Principles of Deep Networks and Building blocks of deep networks.

Building Blocks of Deep Networks

Building deep networks goes beyond basic feed-forward multilayer neural networks.

Three specific building blocks:-

- Feed-forward multilayer neural networks
- RBMs
- Autoencoders

Feed-Forward Networks

- Feed-Forward Networks are the simplest Artificial Neural Networks.

- They are composed of an input layer, one or many hidden layers, and an output layer.

Feed-Forward Deep Learning Neural Networks

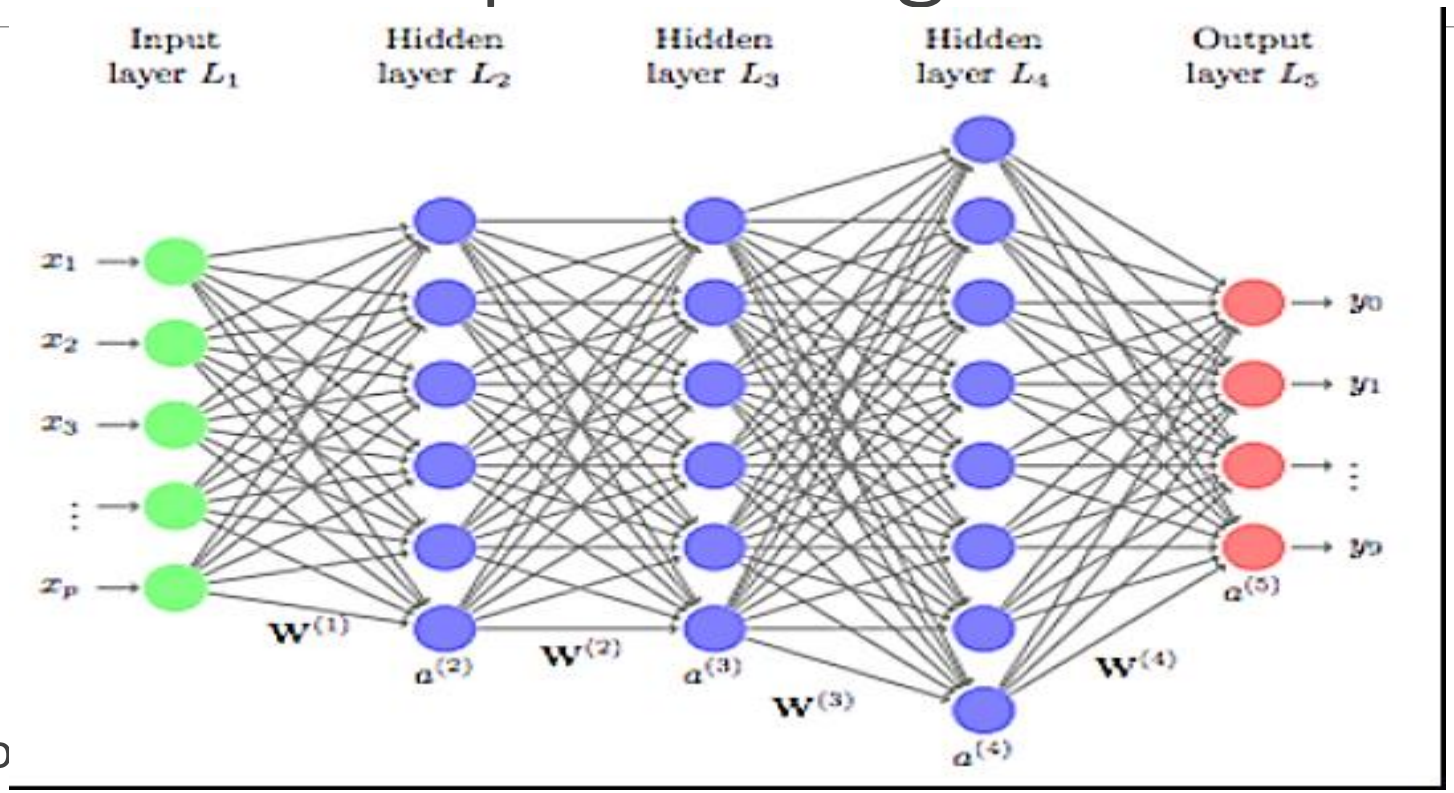


Figure: Feedfo

RBMs(Restricted Boltzmann Machines)

Both RBMs and autoencoders are characterized by an extra layer-wise step for training.

RBMs model probability and are great at feature extraction

They are feed forward networks in which data is fed through them in one direction with two biases, rather than one bias as in traditional backpropagation feed-forward networks.

History of Restricted Boltzmann Machines

RBMs were initially invented under the name **Harmonium** by Paul Smolensky in 1986.

And rose to prominence after Geoffrey Hinton and collaborators invented fast learning algorithms for them in the mid-2000.

Applications of RBMs

1. Dimensionality reduction
2. Classification
3. collaborative filtering
4. feature learning

They can be trained in either supervised or unsupervised ways, depending on the task.

RBMs are a variant of Boltzmann machines, with the restriction that their neurons must form a bipartite graph.

bipartite graph means a pair of nodes from each of the two groups of units (commonly referred to as the "visible" and "hidden" units respectively)

It is having a symmetric connection between them; and there are no connections between nodes within a group.

RBMs is a more efficient training algorithms.

It Plays major role in Deep Learning Networks.

RBMs are used in deep learning for the following:

----Feature extraction

---Dimensionality reduction

The “restricted” part of the name “Restricted Boltzmann Machines” means that connections betweennodes of the same layer are prohibited

(e.g., there are no visible-visible or hidden- hidden connections along which signal passes).

RBMs Network

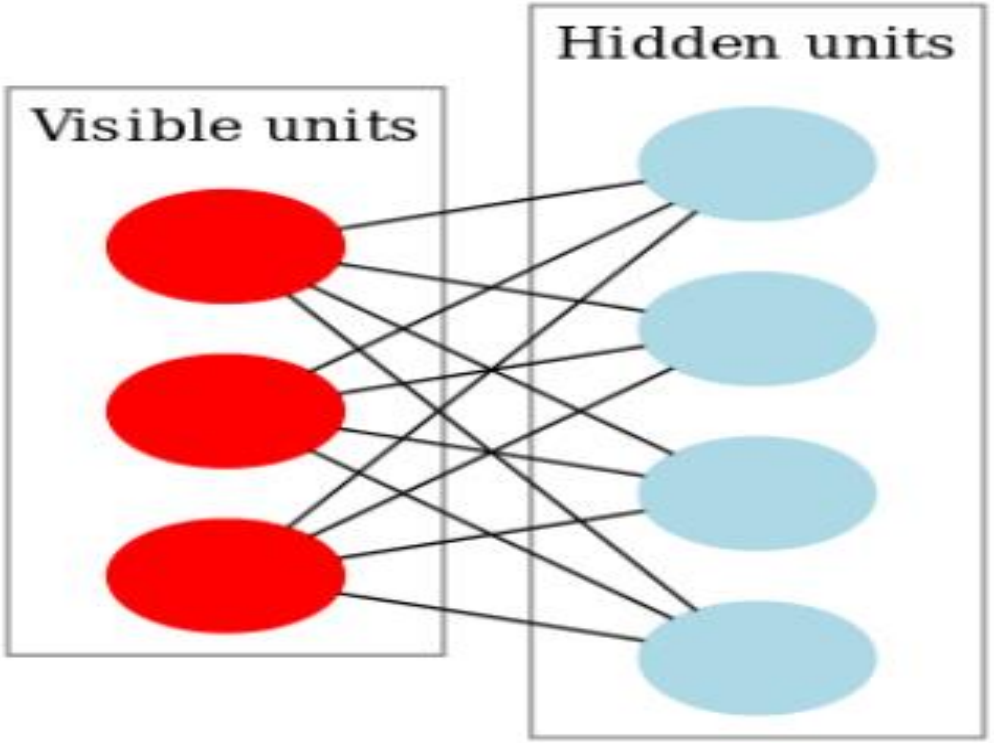


Figure : Restric and four hidden units (no bias units).

RBMs are also a type of autoencoder, which we'll talk about in a following section.

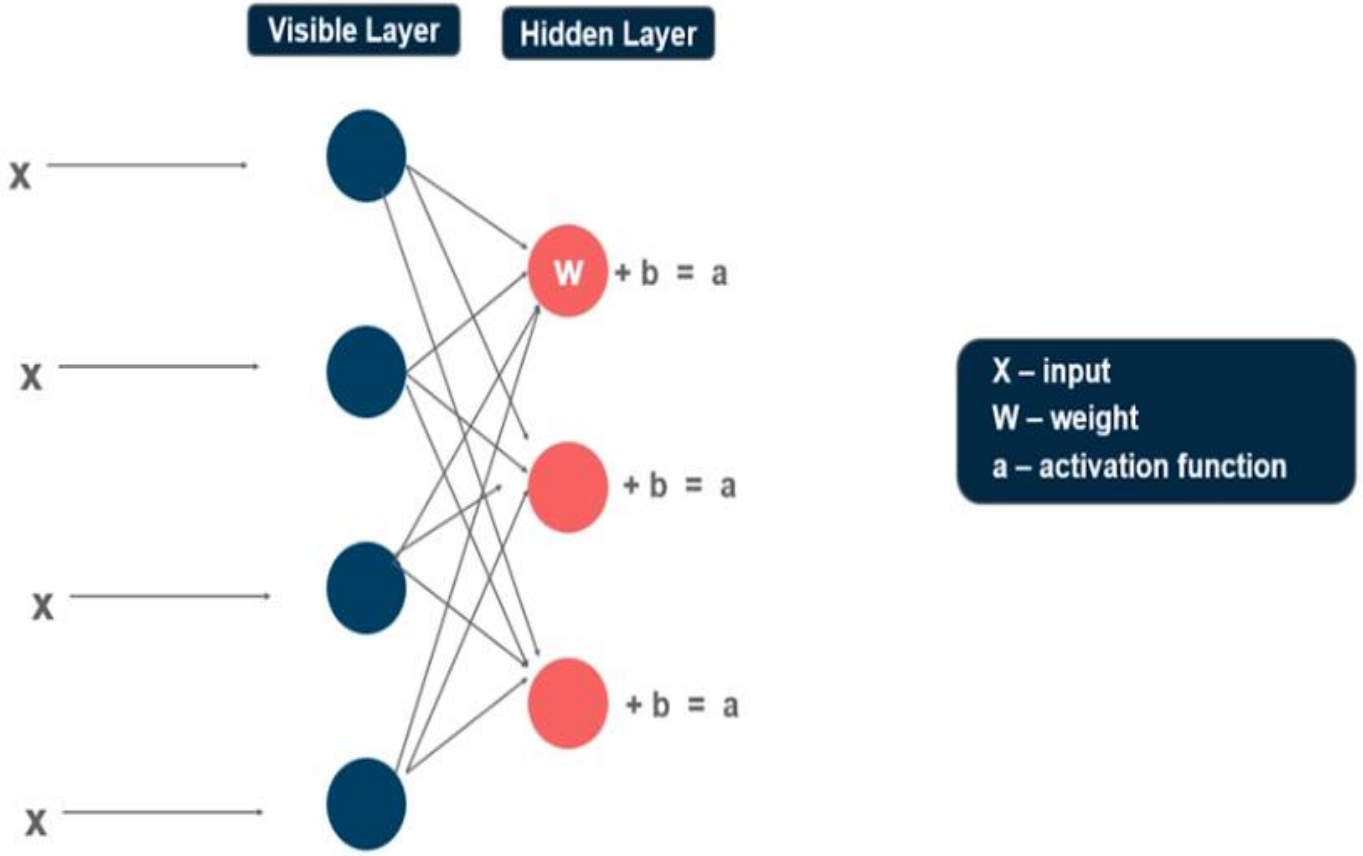
RBMs are used for pretraining layers in larger networks such as Deep Belief Networks.

Network layout

There are five main parts of a basic RBM:

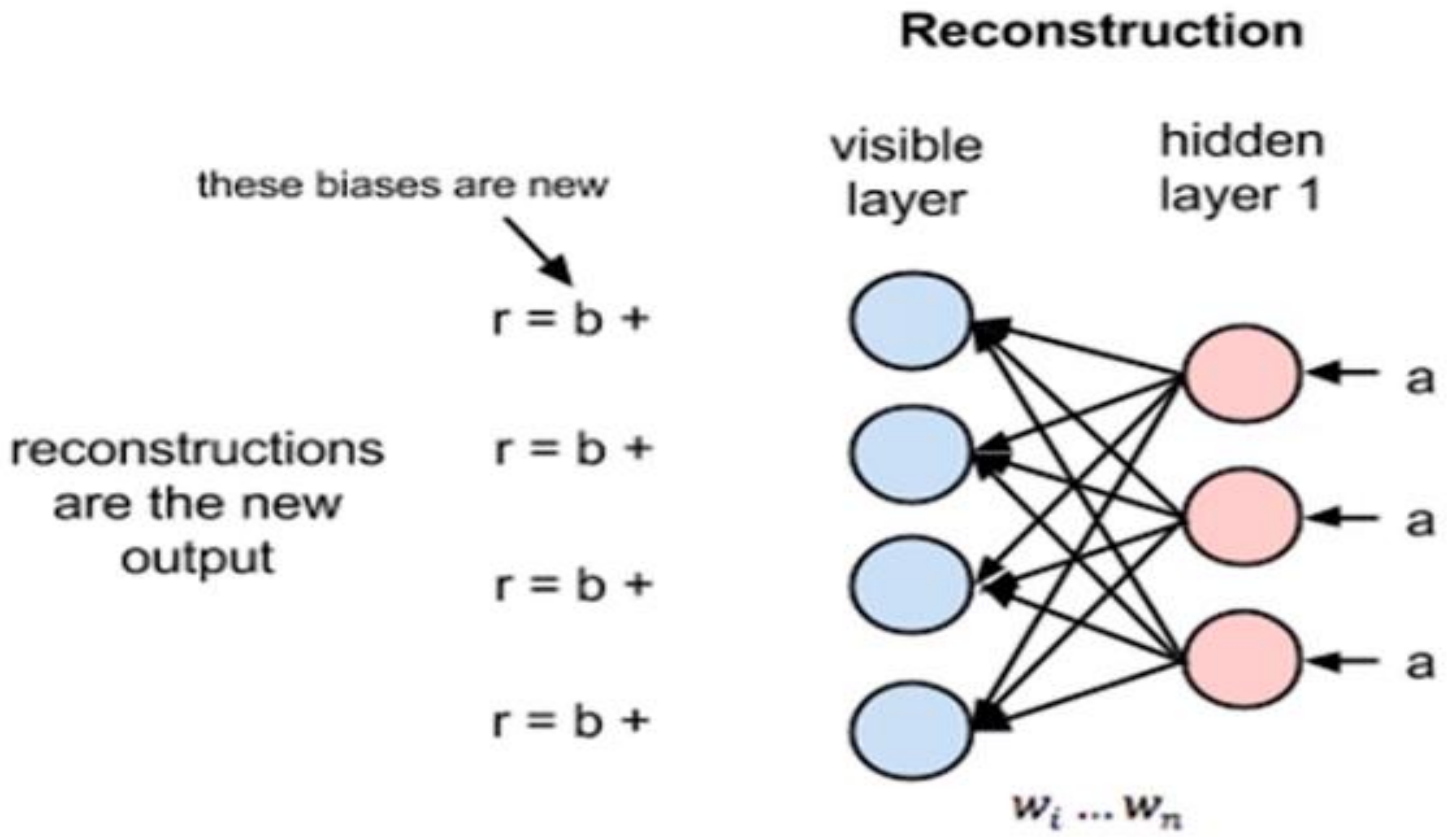
1. Visible units
2. Hidden units
3. Weights
4. Visible bias units
5. Hidden bias units

Working of RBMs

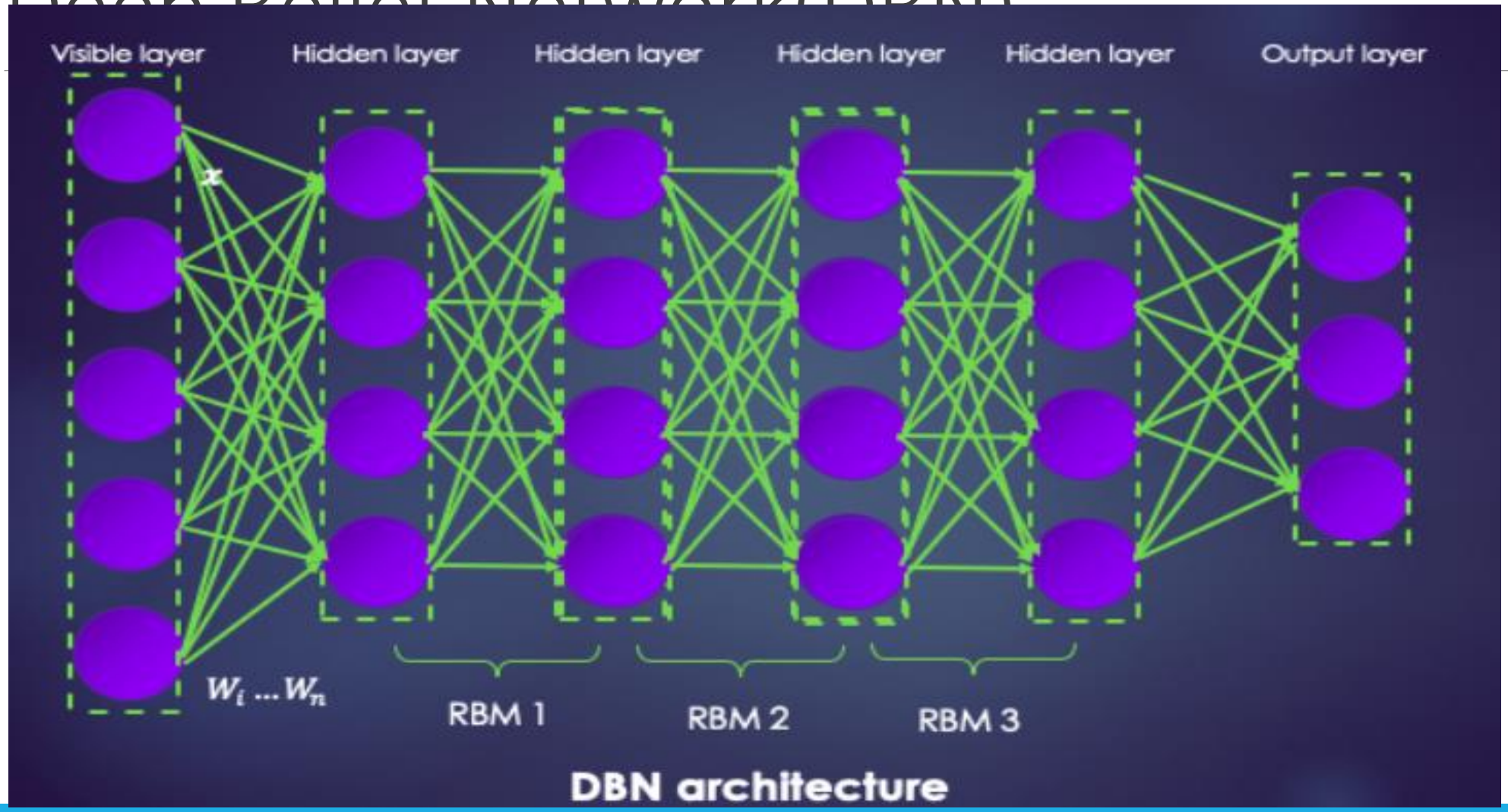


- Layer x is multiplied by a weight and added to a bias and the result of the operation is passed to the activation function which produces the output.
- Reconstruction of RBMs

Reconstruction of RBMs



Deep Belief Network(DBN)



Autoencoders

The autoencoders are used to learn compressed representations of datasets.

It reduce a dataset's dimensionality.

The output of the autoencoder network is a reconstruction of the input data in the most efficient form.

Original Lena Image (12KB size)

Lena Image, Compressed (85% less information, 1.8KB)

Lena Image, Highly Compressed (96% less information, 0.56KB)

Autoencoder Network

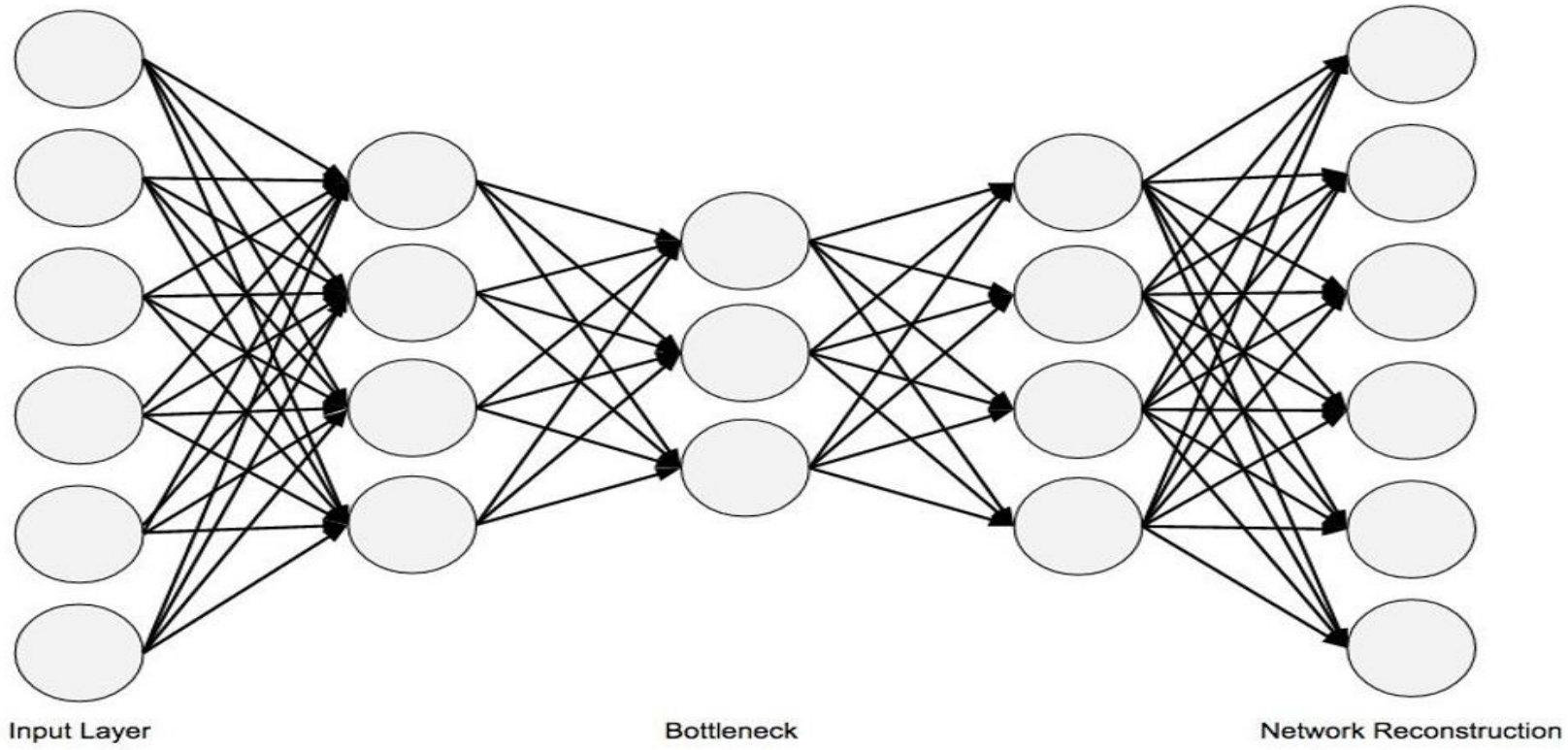


Figure: Auto Encoder Network

Autoencoders share a strong resemblance with multilayer perceptron neural networks.

The key difference to note betweena multilayer perceptron network diagram (from earlier chapters) and an autoencoder diagram is the output layer in an autoencoder has the same number of units as the input layer does.

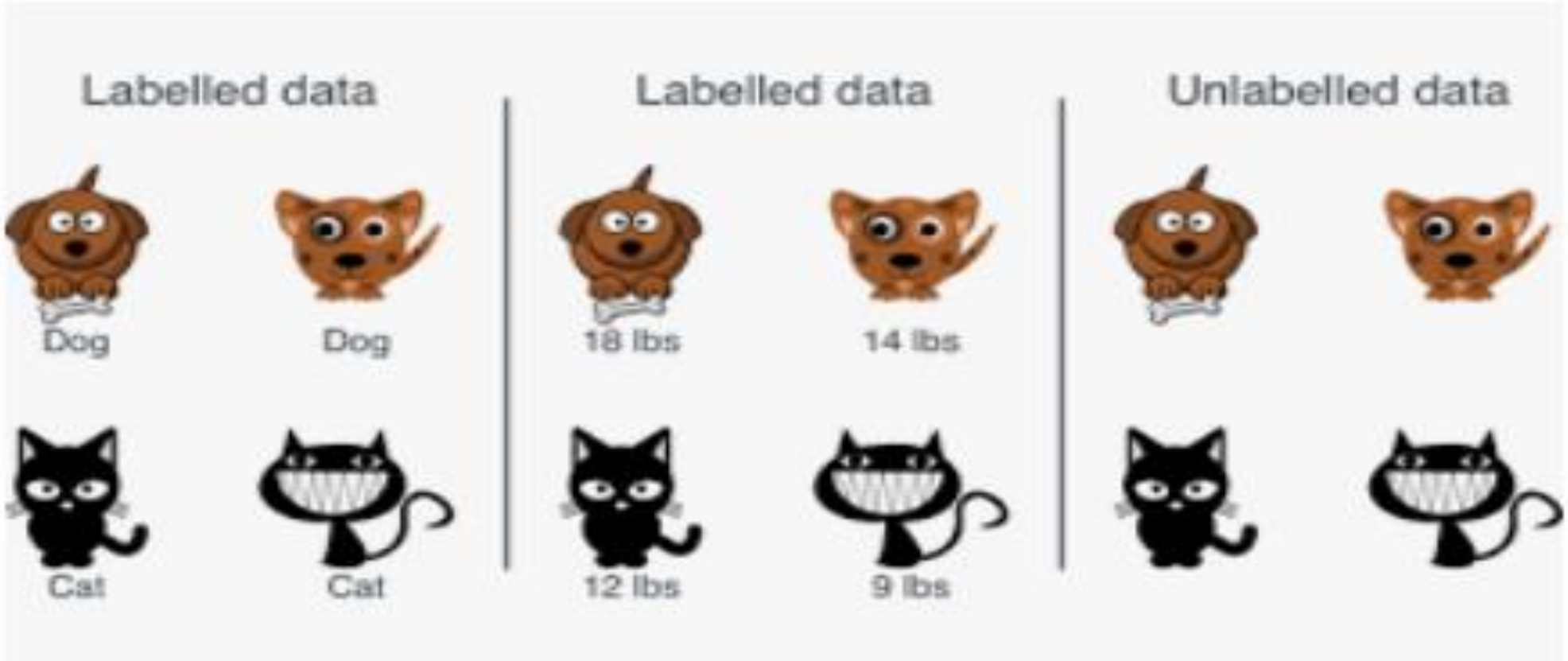
Features of autoencoders

Unlabeled data in unsupervised learning.

Compressed representation of the input data.

Unsupervised learning of unlabeled data

The autoencoder learns directly from unlabeled data. This is connected to the second major difference between multilayer perceptrons and autoencoders.


Labelled data
Dog
Dog
Cat
Cat
Labelled data
18 lbs
14 lbs
12 lbs
9 lbs
Unlabelled data

Learning to reproduce the input data

The goal of a multilayer perceptron network is to generate predictions over a class (e.g., fraud versus no fraud). An autoencoder is trained to reproduce its own input data.

Training autoencoders

Autoencoders rely on backpropagation to update their weights.

The main difference between RBMs and autoencoders is the more general class of in how they calculate the gradients.

Types of autoencoders

1. Compression autoencoders
2. Denoising autoencoders.

Compression autoencoders

The network input must pass through a bottleneck region of the network before being expanded back into the output representation.

Denoising autoencoders

It is the scenario in which the autoencoder is given a corrupted version (e.g., some features are removed randomly) of the input and the network is forced to learn the uncorrupted output.

Variational Autoencoders (VAE)

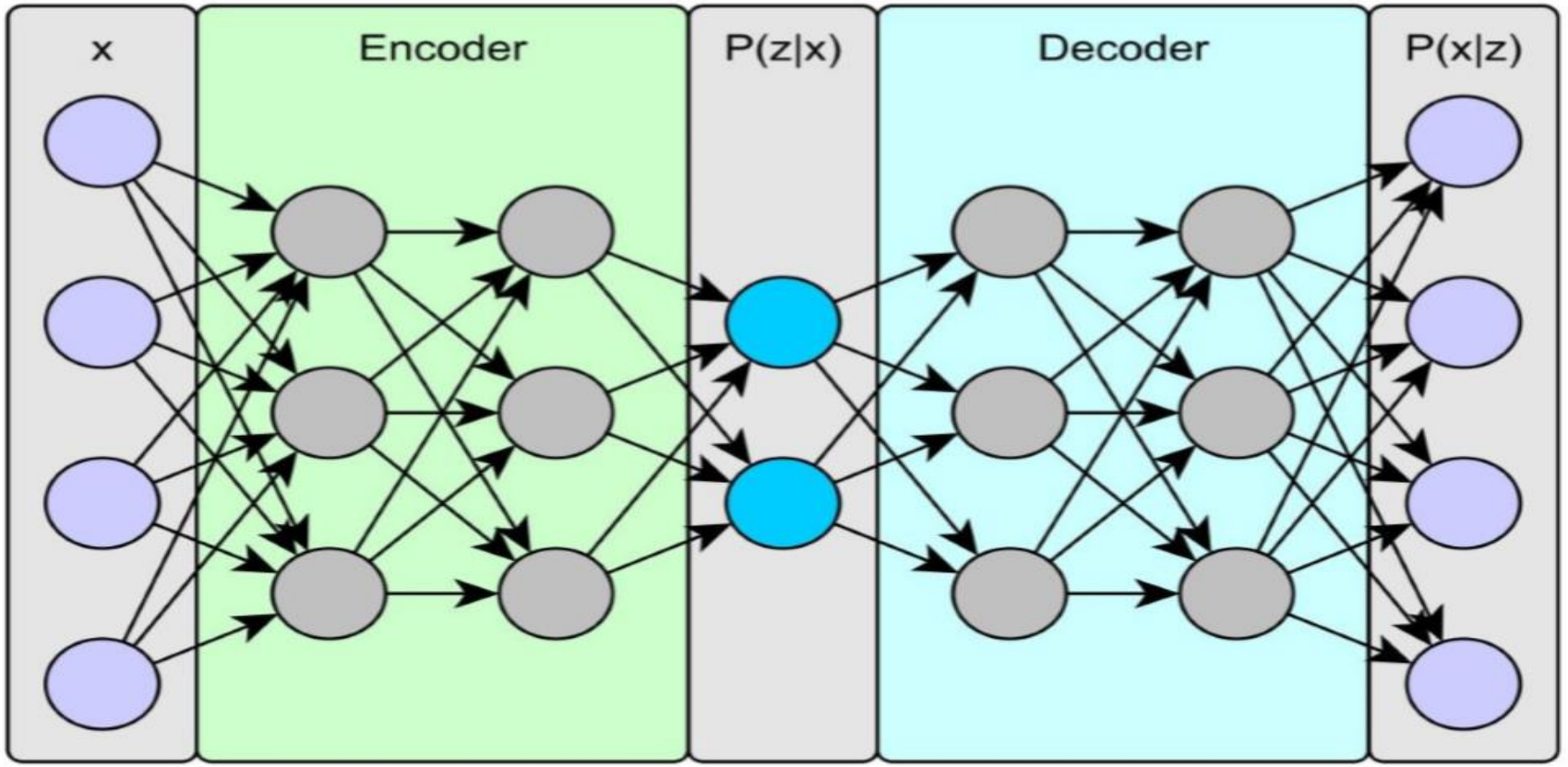


Figure: VAE Network Architecture

- ✓A more recent type of autoencoder model is the variational autoencoder (VAE)
- ✓Introduced by Kingma and Welling.
- ✓VAE uses a probabilistic approach for the forward pass.