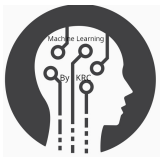
Machine Learning: Basic Neural Nets to Deep Learning

Prof K R Chowdhary

Former Professor & Head, CSE, MBMEC

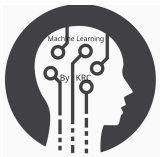
July 9, 2025



Supervised Learning

Most successful kinds of machine learning algorithms are: those that automate decision-making by generalizing from known examples. In *supervised learning*, the user provides the algorithm with pairs of: *inputs* and *desired* outputs, and the algorithm finds a way to produce the desired output given an input.

The algorithm is able to create an output for an input it has never seen before without any help from a human. Considering that pair of inputs are: $\{(1,2),(2,4),(3,9),(4,16),...,(10,100)\}$, the system learns that the relation is square. Now, this relation (of square) is used to produce output for any input x, and the output will be x^2.



Supervised Learning ...

Considering the example of email classification using machine learning, a user provides the algorithm with a large number of emails as input, together with information about whether the email is spam (i.e., the desired output).
Given a future new email, the algorithm will then produce a prediction as to whether the new email is a spam (see Fig. 1).

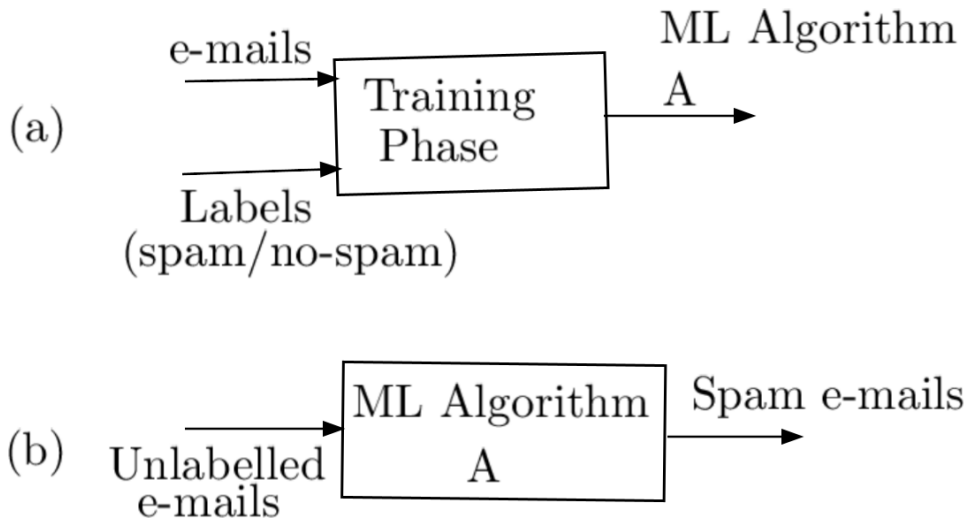


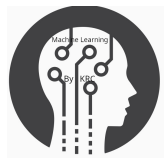
Figure 1: Machine Learning Algorithm: (a) Training phase, (b) Testing Phase

Supervised ML

1. *Identifying zip code from handwritten digits*: Input is a scan of the handwriting, and the desired output is the actual digits in the zip code.

2. *Determining whether a tumor is benign*: Input is the medical image, and the output is whether the tumor is benign.

3. *Detecting fraudulent credit card transaction*: Input is a record of the credit card transaction, and the output is whether it is likely to be fraudulent.



Mathematical definition of Supervised Learning

Example: Given an input or *feature vector* **x**, one of the main goals of machine learning is to predict an output variable y. For example, **x** could be a digitized signature and y, a binary variable that indicates whether the signature is genuine or not.

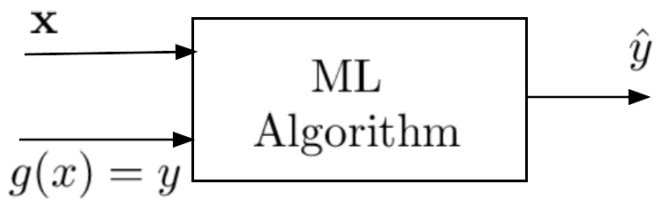


Figure 2: Defining Machine Learning

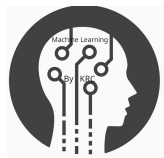
The g comprises all the information about the relationship between the variables **x** and y, excluding the random and chance cases.

Unsupervised Learning

• As an example, consider the task of learning to detect spam e-mail versus the task of *anomaly detection*.

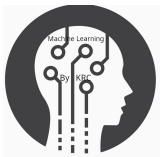
• For the first, we consider a setting in which the learner receives e-mails to be trained, and each of it is manually labeled as "spam" or "not-spam". On the basis of such training, the learner figures out a rule for labeling a newly arriving e-mail messages as spam or no-spam.

• In contrast, for the task of anomaly detection, all that the learner gets as training is a large body of e-mail messages (with no labels) and the learner's task is to detect "unusual" messages.



Unsupervised algorithms

- In unsupervised learning, only the input data is known, and no known output data or labels are provided. Some examples are:
- *Identifying topics in a set of blog posts*: If you have a large collection of text data, you might want to summarize it and find prevalent themes in it.
- *Segmenting customers into groups*: Given a set of customer records, you might want to identify which customers are similar, and whether there are groups of customers with similar preferences.
- *Detecting abnormal access patterns to a website*: To identify abuse or bugs, it is often helpful to find access patterns that are different from the norms.
- All these processes are *clustering algorithms*.



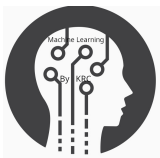
Data for Supervised and Unsupervised Learning

- Both supervised and unsupervised learning tasks requires representation of input data, in the form of a table.

- Image can be described by, e.g., image of a tumor, by the gray-scale values of each pixel, or maybe by using the size, shape, and color of the tumor.

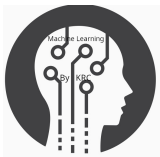
- Each *entity* or *row*, is known as a sample (or data point) in machine learning, while the columns – the properties that describe these entities –are called *features*.

- Building a good representation of data is called *feature extraction* or feature engineering.



Learning through Neural Networks

- An average five-year-old child can easily recognize difference between his teacher madam's face and that of his mother. But, a computer with conventional programming, cannot do it.
- A neural network consists of two or more layers: an input layer, zero or more hidden layers, and an output layer.
- Using an iterative approach, a neural network continuously adjusts and makes inferences until some stopping point is reached.
- The Neural Networks have applications in image recognition, natural language and speech processing, and computer vision.

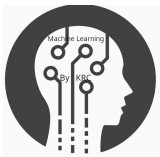


Neural Networks (NNs) vs. Deep Learning (DL) Networks

- DL networks are complex NNs, with many layers, designed to emulate human brain. Computers using these can be trained to deal with *abstractions* and problems that are *poorly defined*.
- A typical NN may consist of thousands or even millions of simple processing nodes that are densely interconnected.

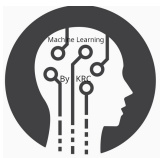
They are useful in learning patterns from unstructured data.

- DL uses hierarchical neural networks to learn from a combination of *unsupervised* and *supervised* algorithms. Typically, DL networks learns from unlabeled and unstructured data.



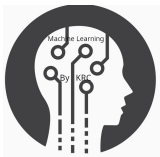
Neural Networks

- Artificial Neural Networks (ANN) is a brain model.
- The science of machine learning is mostly experimental as there is a no universal learning algorithm yet. Given a number of tasks, none can make a computer to learn every task well.
- We are fairly good at general learning abilities due to which we are able to master number of tasks, like playing chess and playing cards.
- A knowledge-acquisition algorithm is always required to be tested on learning tasks and data.
- There is no method to prove that the given algorithm will be consistently better for all the situations.
- These arguments suggest and might serve as inspirations for building machines with some form of general intelligence.

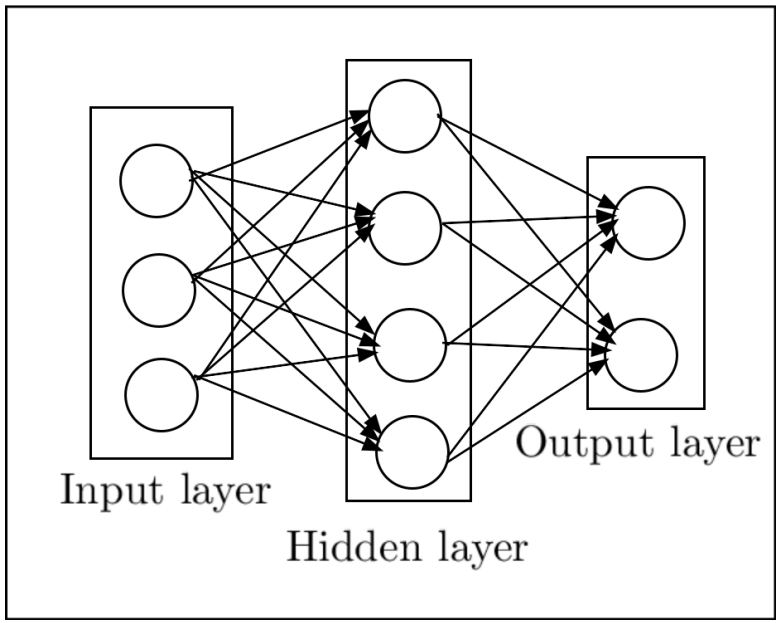


Neural Networks...

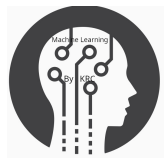
- Basic unit of brain for performing the computation is a cell, called *neuron*, each one of them sends a signal to other neurons through very small gaps between the cells, called *synaptic clefts*.
- The property of any neuron of sending a signal through this gap, and the amplitude of the signal together, is called as *synaptic strength*.
- As a neuron learns enough, its synaptic strength increases, and in that situation, if it is stimulated by an electrical impulse, there are better chances that it would send messages to its neighboring neurons.
- Using *iterations*, a neural network continuously adjusts and makes inferences until some stopping pt. reached.



Architecture of a simple NN



- Majority of these algorithms are based on *supervised learning*.
- As an example, let picture of sunrise is associated with caption: "Sunrise". Hence, goal of learning algorithm is: take the photo as an input, and produce name of object in the image, as output, i.e., "sunrise."
- The process of transforming an input to the output is math *function*.
- Synaptic strength (a numerical value) produce this function, which is the solution to the learning through ANN.
- In fact, when we want to teach to the algorithm "what the sunrise is", then the algorithm should recognize any sunrise, even the one for which we have not trained it! This is the goal of ML.



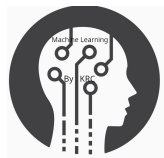
NN: Properties

Important properties:

- Learning ability,
- Massive parallelism,
- Adaptability,
- Fault tolerance,
- Distributed representation and computation,
- Generalization ability, and
- Low energy consumption.

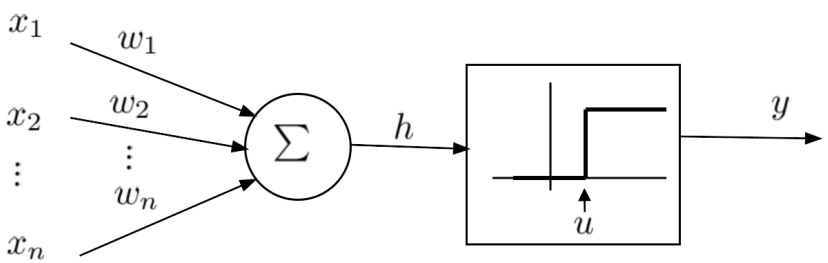
Thus, ANN has many applications. Although their models vary, the most common is in learning and computation, where ANN is the basic processing unit. Each processing unit has following properties:

- an activity-level to represent neuron's polarization state,
- an output value represents firing rate,
- a set of input connections,
- synapses on the cell and its dendrite,
- a bias value to represent an internal resting level of a neuron, and
- output connections to represent neuron's *axonal* projections.



Basic model of an Artificial Neuron

• Each connection of a neuron has an associated weight, called *synaptic strength* or weight w_i of *ith* input, that influences the effect of the incoming input on the activity of the unit. The weight is either +ve (excitatory) or -ve (inhibitory).



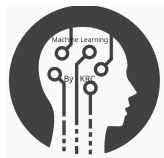
• The basic model for artificial neuron with binary threshold is shown above.

• The mathematical neuron computes the weight as the *sum* of its n-input signals $x_1, ..., x_n$.

• The generated output is 1, if the *sum* > *threshold* u, else, output is 0. The o/p is:

$$y = \theta\Big(\sum_{i=1}^{n} w_i x_i - u\Big), \qquad (1)$$

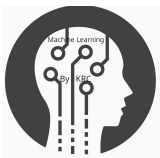
• The $\theta(.)$ is *unit step* function at 0. Threshold u is other weight; $w_0 = -u$ is attached to the neuron with a constant input $x_0 = 1$.



An overly simplified one of true biological neuron

- A properly chosen weights allows a synchronous arrangement of such neurons to perform universal computations.
- There is a crude analogy of this neuron model to biological neurons as follows: wires and interconnections model the *axons* and *dendrites*, respectively, in the biological neuron.
- The *connection weights* in here correspond to *synapses* in biological neuron, and threshold function approximates the activity in biological neuron.
- The ANNs can be considered as weighted directed graphs, where artificial neurons act as nodes, and directed edges with weights are connections between neurons, and between outputs and inputs of neuron.
- Based on the connection pattern an ANN can be classified as:

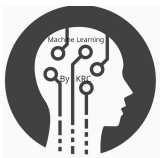
- *Feed-forward networks:* The direct graphs have no loops.
- *Recurrent feedback networks:* Have loops, due to feedback connections.



Linear Classifiers

- Examples with n-dimensional instance space, positive and negative examples tend to cluster in different regions.
- This observation motivates us to use another approach to classification where we identify decision surfaces that separates the two classes.
- A very simple approach is to use a linear function.

- The goal of predictive modeling is to build a model that predicts some specified attribute(s) value from the values of the other attributes.
- We will elaborate on linear classifier in general.
- We shell use a domain with attributes as real numbers. To use these attributes in a algebraic function shown in Fig. 3



Linear Classifier

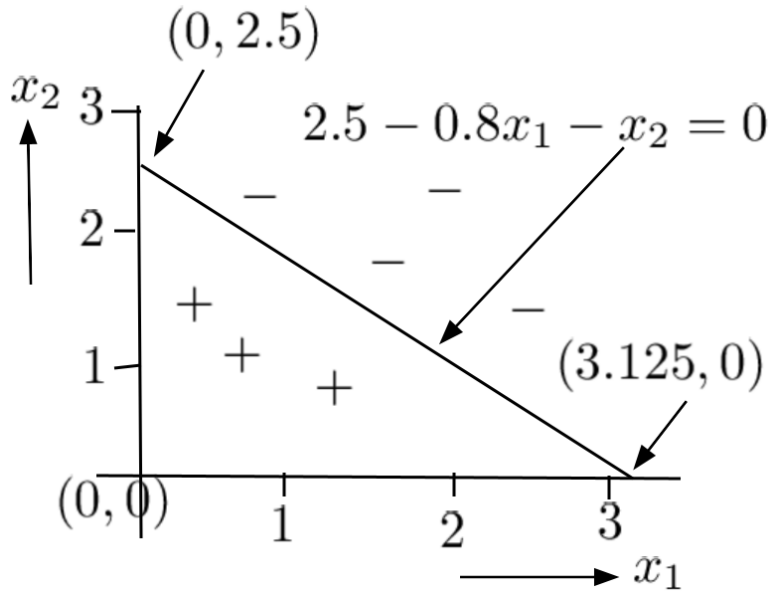


Figure 3: A linear classifier in a domain of two real valued attributes x_1, x_2

- Examples are labeled as positive (+) and negative (-), and two classes are separated by a linear function:

$$2.5 - 0.8x_1 - x_2 = 0 \qquad (2)$$

- Equation (2)[1], the variables x_1 and x_2 are real numbers.

Exercise: Given the graph in Fig. (3), construct the eq. (2).

Hint: Extend line to -x_1 direction, and the angle m in $y = mx + c$ is $-\frac{2.5}{3.125} = 0.8$

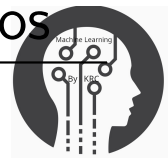
[1] This equation is a standard line equation $y = mx + c$, where m is slope and c is point where this line intersects on x axis. We have used coordinates x_1, x_2, which can be extended to n coordinates x_1 ... x_n.

Linear Classifier...

- Table 1 shows seven examples of attributes (x_1, x_2), value of classifier "$2.5 - 0.8x_1 - x_2$," and class of the example.
- When value of classifier is *neg*, point or the coordinate is falling above the straight line in Fig. 3, when classifier returns positive, the example is taken as belonging to pos class.
- Hence, given a classifier like this, we are able to classify any attribute set, which is a two dimensional vector.

Table 1: Set of attributes (x_1, x_2) and their classes

x_1	x_2	$2.5-0.8x_1-x_2$	Class
1.0	2.3	-0.6	neg
1.6	1.8	-0.58	neg
2.1	2.7	-1.88	neg
2.4	1.4	-0.82	neg
0.8	1.1	$+0.76$	pos
0.8	1.8	$+0.06$	pos
1.4	0.8	$+0.58$	pos



Linear Classifier...

• Note only the linear classifier (2) classifies examples as *pos* and *neg*, but any classifier, e.g., $1.5 + 2.1x_1 - 1.1x_2$, will classify infinitely large number of examples as *pos*/*neg*. Generic form is:

$$w_0 + w_1 x_1 + w_2 x_2 = 0. \quad (3)$$

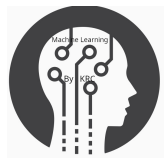
And, for a domains with n attributes is:

$$w_0 + w_1 x_1 + ... + w_n x_n = 0. \quad (4)$$

In eq. (4), if $n = 2$, it a line, if $n = 3$, it is a plane, for $n > 3$, it is a *hyperplane*. If 0th attribute $x_0 = 1$, eq. (4) becomes:

$$\sum_{i=0}^{n} w_i x_i = 0. \quad (5)$$

Classifier's behavior is decided by coefficients w_i (weights). Task of ML is: find out w_i's values. In equ. $y = mx + c$, the m is angle w.r.t. x axis, and in (4), coefficients $w_1, ..., w_n$ define angle of hyperplane, w.r.t. system of coordinates, w_0 is *bias* or offset – the hyperplane has distance from the system coordinates.



Linear Classifier...

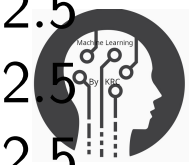
- *Bias versus Threshold*: Bias is amount of error introduced by approximating real-world phenomena with a simplified model.
- *Bias* in Fig. 3 is $w_0 = 2.5$, lower the bias, classifier shifts closer to origin $[0, 0]$, higher value shifts it away from origin. At, $w_0 = 0$, the classifier intersects the origin of the coordinate system. Equation (4) can also be written as:

$$w_1 x_1 + w_2 x_2 + ... + w_n x_n = \theta, \quad (6)$$

here, $\theta = -w_0$. This θ is called *threshold* that weighted sum has to exceed it, if the example is to be positive.

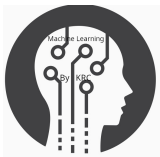
Table 2: Attributes (x_1, x_2), their weighted sum and threshold

x_1	x_2	$(-0.8x_1 - x_2)$	θ
1.0	2.3	−3.3	−2.5
1.6	1.8	−3.8	−2.5
2.1	2.7	−4.26	−2.5
2.4	1.4	−3.52	−2.5
0.8	1.1	−1.74	−2.5
0.8	1.8	−2.24	−2.5
1.4	0.8	−1.92	−2.5



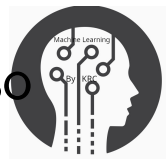
Perceptron Learning

- Last 3 examples (table 2): weighted sum in third column exceeds θ, so they have *pos* labels. First 4 examples: weighted sum $< \theta$, so label=*neg*.
- *Perceptron Learning*: To simplify linear classifier, we assume that training example $\mathbf{x}$ is described by n binary attributes for n dimensions, $x_i \in \mathbf{x}$ is binary, i.e., 0 or 1.
- For $c(\mathbf{x}) = 1$, class=*pos*, for $c(\mathbf{x}) = 0$, it is *neg*. Real class is c, and hypothesized class is $h(\mathbf{x})$, (h = hypothesis). If, $\sum_{i=0}^{n} w_i x_i > 0$, classifier hypothesizes $\mathbf{x}$ as *pos*, so $h(\mathbf{x}) = 1$.
- When, $\sum_{i=0}^{n} w_i x_i \leq 0$, label is *neg* and $h(\mathbf{x}) = 0$.
- Examples with $c(\mathbf{x}) = 1$ are linearly separable from those with $c(\mathbf{x}) = 0$. So, there exists a linear classifier that can label correctly all the training examples $\mathbf{x}$, and for each $h(\mathbf{x}) = c(\mathbf{x})$. Task of ML: find weights w_i that correctly classifies all $\mathbf{x}$.



Inducing the Linear Classifier

- Objective: for any attribute example $\mathbf{x}$ with real class $c(\mathbf{x}) = 1$, the classifier must hypothesize the example as positive, i.e. $h(\mathbf{x}) = 1$, and when $c(\mathbf{x}) = 0$, it must hypothesize $\mathbf{x}$ as negative, i.e. $h(\mathbf{x}) = 0$. We can do this by adjusting the weight w_i.
- When classifier is presented with training example $\mathbf{x}$, it must return its label as $h(\mathbf{x})$. If $c(\mathbf{x}) \neq h(\mathbf{x})$, weights w_i are not perfect, so they must be modified so that $c(\mathbf{x}) = h(\mathbf{x})$.
- Assume that $c(\mathbf{x}) = 1$ and $h(\mathbf{x}) = 0$. This happens only if $\sum_{i=0}^{n} w_i x_i < 0$: an indication that the weights are too small. So, weights must be increased so that $\sum_{i=0}^{n} w_i x_i > 0$, (So, $h(\mathbf{x}) = 1$.
- It is simple to understand that only the weight of w_i be increased for which $x_i = 1$, (when $x_i = 0$, $w_i.x_i = w_i.0 = 0$). (This is the reason we chose binary attributes!!).
- Similarly, when $c(\mathbf{x}) = 0$ and $h(\mathbf{x}) = 1$, we decrease the weights w_i for which x_i are 1, so that $\sum_{i=0}^{n} w_i x_i \leq 0$.



Weight adjustment in Perceptron

Weight adjustment Summary:

- Hypothesized label, $h(\mathbf{x}) = 1$ when real class label $c(\mathbf{x}) = 0$: decrease w_i for attribute $x_i = 1$,
- Hypothesized label, $h(\mathbf{x}) = 0$ when real class label $c(\mathbf{x}) = 1$: increase w_i for attribute $x_i = 1$,
- Both labels same, $c(\mathbf{x}) = h(\mathbf{x})$: no wt. adjustment reqd.

Regulate the weights by:

$$w_i = w_i + \eta.[c(\mathbf{x}) - h(\mathbf{x})].x_i \quad (7)$$

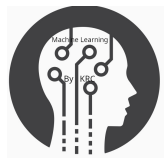
$\eta \in (0, 1]$, called *learning rate*.

- Checking validity of equation (7): (i) When $c(\mathbf{x}) = h(\mathbf{x})$: w_i remains unchanged.

(*ii*) When $c(\mathbf{x}) = 1$ and $h(\mathbf{x}) = 0$: RHS of equ. (7) is: $w_i + \eta.1.x_i = w_i + \eta$, as $x_i = 1$. This increases w_i, so it is ≥ 1, hence perceptron fires, and makes $h(\mathbf{x}) = 1$.

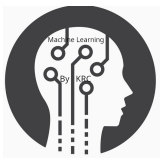
(*iii*) When $c(\mathbf{x}) = 0$ but $h(\mathbf{x}) = 1$: RHS of equ. (7) is: $w_i + \eta.[-1].1 = w_i - \eta$, as $x_i = 1$. This decreases w_i to ≤ 0, it stops the perceptron from firing, and makes $h(\mathbf{x}) = 0$.

This concludes how perceptron hypothesizes the same label as the label of $c(\mathbf{x})$.



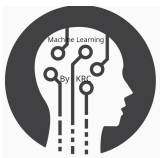
Perceptron Learning Algorithm

- To start with, weights w_i of perceptron are initialized to some random values. Next, each training example $\mathbf{x}$ with attributes $x_1, ..., x_i, ..., x_n$, is presented to the classifier, one at a time. Each time, every weight of the classifier is subjected to equation (7).
- The training for last example $\mathbf{x}$ shows that one *epoch* (round) of training is complete. If all the labels are correctly hypothesized, indicated by $h(\mathbf{x}) = c(\mathbf{x})$, the training process is terminated, else it repeats from first example again. Usually, many such rounds are needed to train the perceptron. The corresponding algorithm is shown as algorithm 1.



Algorithm 1 Perceptron learning Algorithm

1: % Let two classes be $c(\mathbf{x}) = 1$ and $c(\mathbf{x}) = 0$, and they are linearly separable.
2: Initialize weights w_i to some small random numbers.
3: Choose some suitable learning rate $\eta \in (0, 1]$.
4: **while** $c(\mathbf{x}) \neq h(\mathbf{x})$ for all training examples **do**
5: **for** each training example $\mathbf{x} = (x_1, ..., x_n)$, having class $c(\mathbf{x})$ **do**
6: $h(\mathbf{x}) = 1$ if $\sum_{i=0}^{n} w_i x_i > 0$, otherwise $h(\mathbf{x}) = 0$.
7: Update each weight using the formula, (7)
8: **end for**
9: **end while**



Example on Perceptron Learning Algorithm

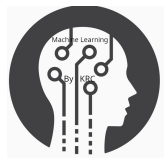
We are given a table of examples as 3, with three examples Ex1 to Ex3, each having three binary attributes.

Table 3: Examples for perceptron learning

Example	x_1	x_2	x_3	$c(\mathbf{x})$
Ex1	1	1	0	1
Ex2	0	0	1	0
Ex3	1	0	1	0

We consider that learning rate $\eta = 0.6$, and randomly generated initial weights w_0, w_1, w_2, w_3 are $[0.15, 0.2, 0.1, 0.25]$ and $x_0 = 1$. Given these, our objective is to separate the "+" examples (Ex1) from "-" examples (Ex2, Ex3).

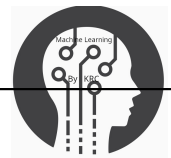
The classifier's hypothesis about class $\mathbf{x}$: $h(\mathbf{x}) = 1$ if $\sum_{i=0}^{n} w_i x_i > 0$, and $h(\mathbf{x}) = 0$, otherwise. After each example is presented to the classifier, all the weights are adjusted through formula (7), as table 5 shows.



Example on Perceptron Learning Algorithm...

Table 4: Weight adjustments for perceptron learning

Var.→ Examples ↓	x_1	x_2	x_3	w_0	w_1	w_2	w_3	$c(\mathbf{x})$	$h(\mathbf{x})$	$c(\mathbf{x})$-$h(\mathbf{x})$
Random classifier				0.15	0.2	0.1	0.25			
Ex1→ New Classifier:	1	1	0	0.15	0.2	0.1	0.25	1	1	0
Ex2→ New Classifier:	0	0	1	−0.45	0.8	0.7	−0.35	0	1	−1
Ex3→ New Classifier:	1	0	1	−0.45	0.8	0.7	−0.35	0	0	0



Example on Perceptron Learning Algorithm...

• Let us see how the computation in table 5 are computed: First, we find out hypothesized class $h(\mathbf{x})$ in Ex1:

$$\sum_{i=0}^{n} w_i x_i$$

$$= w_0 x_0 + w_1 x_1 + w_2 x_2 + w_3 x_3$$

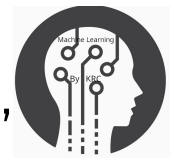
$$= 0.15 \times 1 + 0.2 \times 1 + 0.1 \times 1 + 0.25 \times 0$$

$$= 0.45$$

Hence, $h(\mathbf{x}) = 1$. Since, $c(\mathbf{x}) = 1$ and $h(\mathbf{x}) = 1$, their difference is zero. So, no weight will change, and the final weights remains the same as it is for $Ex2$.

• Since $c(\mathbf{x}) - h(\mathbf{x}) \neq 0$ in Ex1 to Ex3, we make 2nd round, where weights of Ex3 are used in place of Random classifier at top of the table, and calculate the table again.

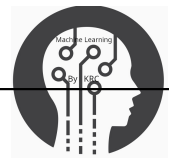
• This also does not make $c(\mathbf{x}) = h(\mathbf{x})$, we make one more round, resulting table is given in next slide.



Example on Perceptron Learning Algorithm...

Table 5: Weight adjustments for perceptron learning

Var.→ Examples ↓	x_1	x_2	x_3	w_0	w_1	w_2	w_3	$c(\mathbf{x})$	$h(\mathbf{x})$	$c(\mathbf{x})$-$h(\mathbf{x})$
Random classifier				−0.45	0.2	0.7	−0.95			
Ex1→ New Classifier:	1	1	0	−0.45	0.2	0.7	−0.95	1	1	0
Ex2→ New Classifier:	0	0	1	−0.45	0.2	0.7	−0.95	0	0	0
Ex3→ New Classifier:	1	0	1	−0.45	0.2	0.7	−0.95	0	0	0



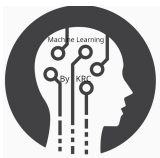
Conclusion: Perceptron Learning Algorithm • So, the classifier is trained in three steps. It classifies Ex1 in one class (+) and Ex2, Ex3 in "−" class, as seen in the column $c(\mathbf{x}) - h(\mathbf{x}$ of the table. Final version of classifier: $-0.45+0.2x_1+0.7x_2-0.95x_3 = 0$, no longer classifies wrongly. The training has thus been completed in three epochs (rounds).

• Note: Learning has taken place using perceptron, and have obtained a linear classifier:

$$-0.45+0.2x_1+0.7x_2-0.95x_3 = 0, \quad (8)$$

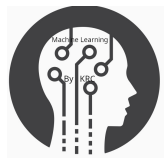
Now, it classify any amount of data as either "+" or "-", in single step. So, classifier has been *induced*.

• Important: Irrespective of he initial weights $(w_0..w_n)$, size n of attribute vector, and learning rate η, if the "+" and "-" classes are linearly separable, this algorithm is guaranteed to find a version of hyperplane in finite number of steps, that separates the classes.



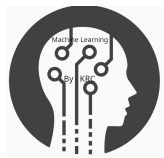
Perceptron learning Program

```
import numpy as np

# Input data: 3 examples with 4 features (x0 to x3)
X = np.array([
    [1, 1, 1, 0],
    [1, 0, 0, 1],
    [1, 1, 0, 1]
])

# Real class labels
c = np.array([1, 0, 0])

# Initial weights
weights = np.array([0.15, 0.2, 0.1, 0.25])

# Parameters
learning_rate = 0.6
epochs = 3
```



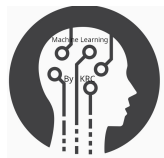
Perceptron...

```
# Perceptron Learning
for epoch in range(epochs):
  print(f"\nEpoch {epoch + 1}")
  for i in range(len(X)):
    x_i = X[i]
    target = c[i]
    activation = np.dot(weights, x_i)
    h_x = 1 if activation > 0 else 0  # Binary threshold at 0
    error = target - h_x

    print(f"  Input: {x_i}, Target: {target}, h(x): {h_x},
                                          Error: {error}")

    # Weight update rule
    weights = weights + learning_rate * error * x_i

    print("  Updated Weights:", weights)
```



Perceptron...

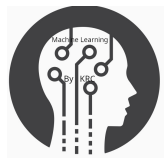
```
# Final model output
print("\nFinal Weights after Training:", weights)

# Prediction function
def predict(x):
    return 1 if np.dot(weights, x) > 0 else 0

# Test all inputs
print("\nPredictions:")
for x in X:
    print(f"  Input: {x}, Predicted Class: {predict(x)}")
```



Perceptron...Results

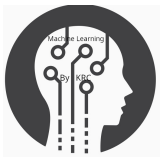
The results on running this program are shown below:

```
Epoch 1
  Input: [1 1 1 0], Target: 1, h(x): 1, Error: 0
  Input: [1 0 0 1], Target: 0, h(x): 1, Error: -1
  Input: [1 1 0 1], Target: 0, h(x): 0, Error: 0
  Updated Weights: [-0.45  0.2   0.1  -0.35]

Epoch 2
  Input: [1 1 1 0], Target: 1, h(x): 0, Error: 1
  Input: [1 0 0 1], Target: 0, h(x): 0, Error: 0
  Input: [1 1 0 1], Target: 0, h(x): 1, Error: -1
  Updated Weights: [-0.45  0.2   0.7  -0.95]

Epoch 3
  Input: [1 1 1 0], Target: 1, h(x): 1, Error: 0
  Input: [1 0 0 1], Target: 0, h(x): 0, Error: 0
  Input: [1 1 0 1], Target: 0, h(x): 0, Error: 0
  Updated Weights: [-0.45  0.2   0.7  -0.95]
```



The results on running this program are shown below:

```
Final Weights after Training: [-0.45  0.2   0.7  -0.95]

Predictions:
  Input: [1 1 1 0], Predicted Class: 1
  Input: [1 0 0 1], Predicted Class: 0
  Input: [1 1 0 1], Predicted Class: 0
```



Support Vector Machine (SVM)

The support vector machine (SVM) is a parametric model, which typically solves the same problem of classification with two classes – and yields the similar performance.

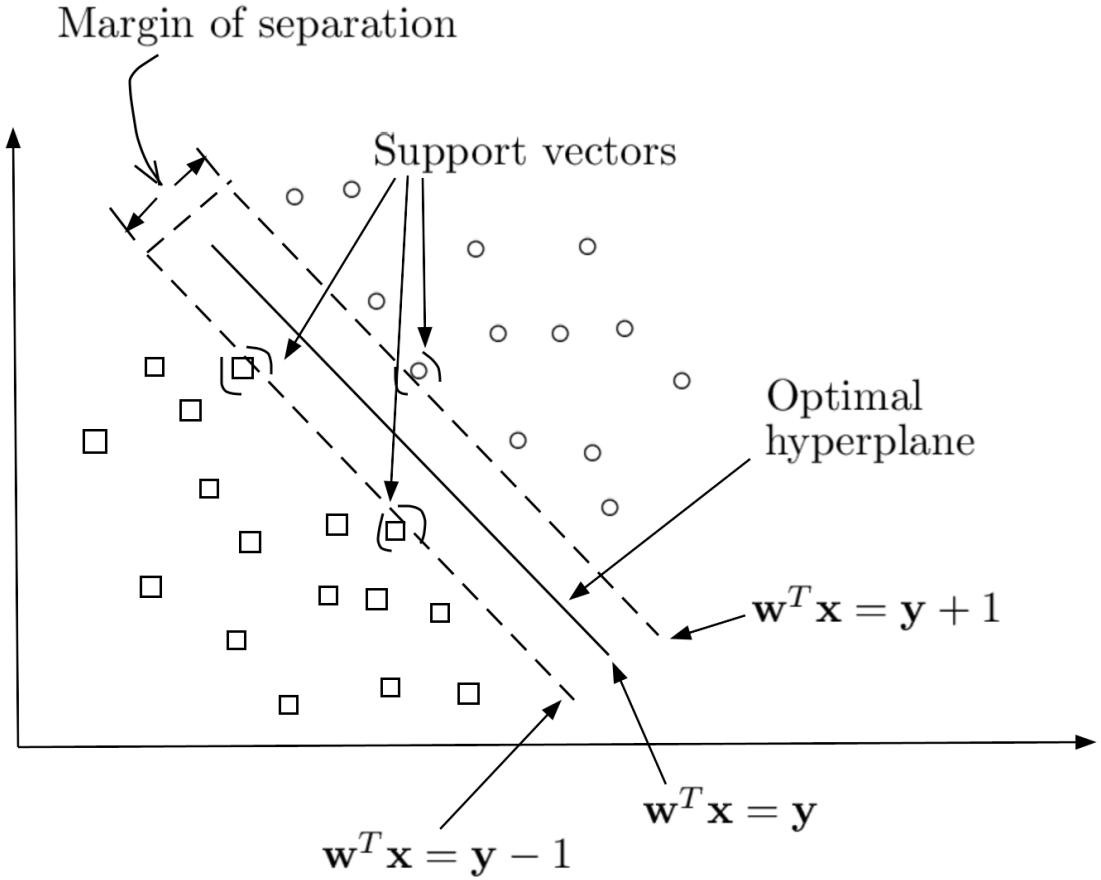
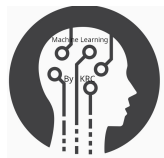


Figure 4: Classification using SVM

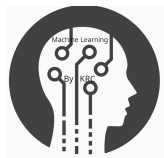
SVMs ...

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.svm import SVC
from sklearn.preprocessing import LabelEncoder

# Input data
weight = [45, 50, 60, 60, 70, 70, 50, 60, 60, 70, 80, 80]
height = [1.5, 1.7, 1.8, 2.0, 1.9, 2.0, 1.4, 1.5, 1.55, 1.6,
                                                    1.6, 1.7]
labels = ['No','No','No','No','No','No','Yes','Yes','Yes',
                                          'Yes','Yes','Yes']

# Combine features into a 2D array
X = np.column_stack((weight, height))

# Encode labels to 0 and 1
encoder = LabelEncoder()
y = encoder.fit_transform(labels)  # No -> 0, Yes -> 1
```



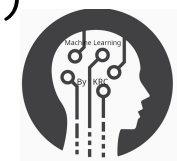
SVM ...

```
# Train hard-margin SVM by setting C to a large value
model = SVC(kernel='linear', C=1e6)
model.fit(X, y)

# Print model parameters
print("Support Vectors:\n", model.support_vectors_)
print("Coefficients (w):", model.coef_)
print("Intercept (b):", model.intercept_)

# Visualize the decision boundary
def plot_svm(model, X, y):
    plt.figure(figsize=(8, 6)) # in Inches

    # Plot data points
    for label, marker, color in zip([0, 1], ['o', 's'],
    ['blue', 'red']): plt.scatter(X[y == label, 0],
    X[y == label, 1], marker=marker, color=color,
     label=f"{encoder.inverse_transform([label])[0]}")
```



SVM ...

```
# Plot decision boundary and margins
ax = plt.gca()
xlim = ax.get_xlim()
ylim = ax.get_ylim()

xx = np.linspace(xlim[0], xlim[1], 100)
yy = np.linspace(ylim[0], ylim[1], 100)
YY, XX = np.meshgrid(yy, xx)
xy = np.vstack([XX.ravel(), YY.ravel()]).T
Z = model.decision_function(xy).reshape(XX.shape)

plt.contour(XX, YY, Z, levels=[0], linewidths=2,
                                        colors='black')
plt.contour(XX, YY, Z, levels=[-1, 1], linestyles=
                                ['--', '--'], colors='gray')
```



```
    plt.xlabel('Weight (kg)')
    plt.ylabel('Height (m)')
    plt.title('Hard-Margin SVM Classification')
    plt.legend()
    plt.grid(True)
    plt.show()
# Call the plot function
plot_svm(model, X, y)
```

Support vectors: [[45.0 1.5][70.0 1.9][50.0 1.4]],
The coefficients: (w): [[0.17905957 -11.15761197]],
Intercept (b): [7.67388049].
Coefficients represents the linear decision boundary line in 2D:

$$f(x) = w_1.x_1 + w_2.x_2 + b, \tag{9}$$

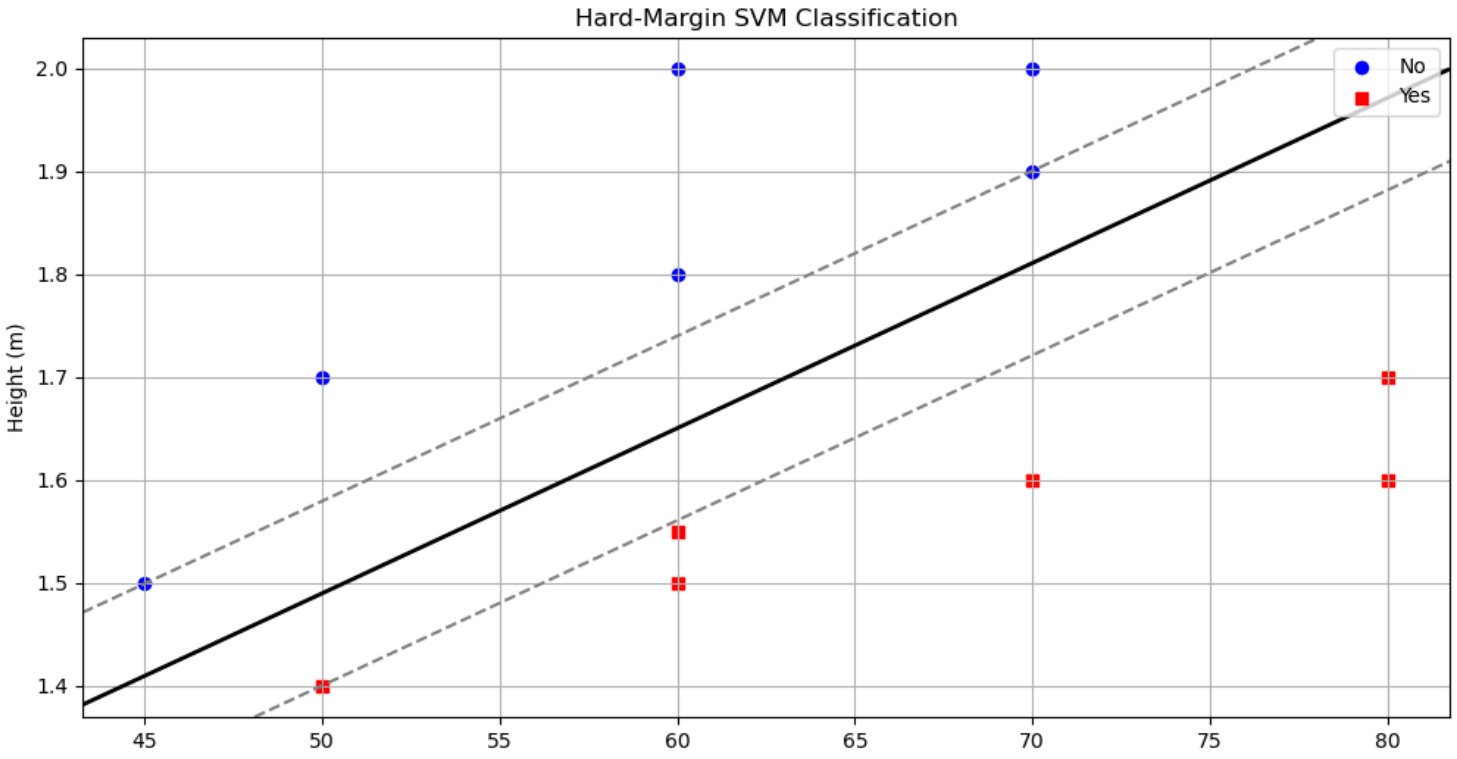
where x_1 is weight, x_2 is height, and b is intercept.

SVM ...

SVM decides class of (x_1, x_2):

- if $f(x) > 0$: Class $= 1$ =Yes
- if $f(x) < 0$: Class =0=No
- $f(x) = 0$ Point lies on decision boundary.

Hence, the decision boundary equation is: $0.179\ weight - 11.1576\ height + 7.6739 = 0$.

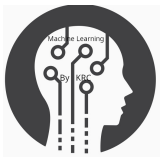


Deep Learning (DL): Intro

- The ML (particularly, the DL) has become popular in research, and it is being used in large number of applications, including multimedia concept retrieval, image classification, video recommendation systems, social network analysis, natural language processing,

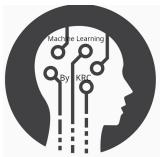
- The DL networks are complex neural networks, that are often used in image recognition, natural language speech processing, computer vision, and many well-known applications.

- DL is a machine learning technique that uses hierarchical neural networks to learn from a combination of unsupervised and supervised algorithms.



Deep Learning (DL): Basic Principles

- The basic technique of DL relies on ANNs, they are based on general principles of mathematics that allow them to learn from examples, to recognize people or objects in a picture, and translate the spoken language from one language to another.
- To be successful in generalizing, having observed number of examples, the deep learning network needs more than just the examples, but depends on hypotheses about the data and assumptions about what can be a possible solution of a given problem.
- It is essentially useful when we are trying to learn patterns from unstructured data.



Deep Learning Architecture

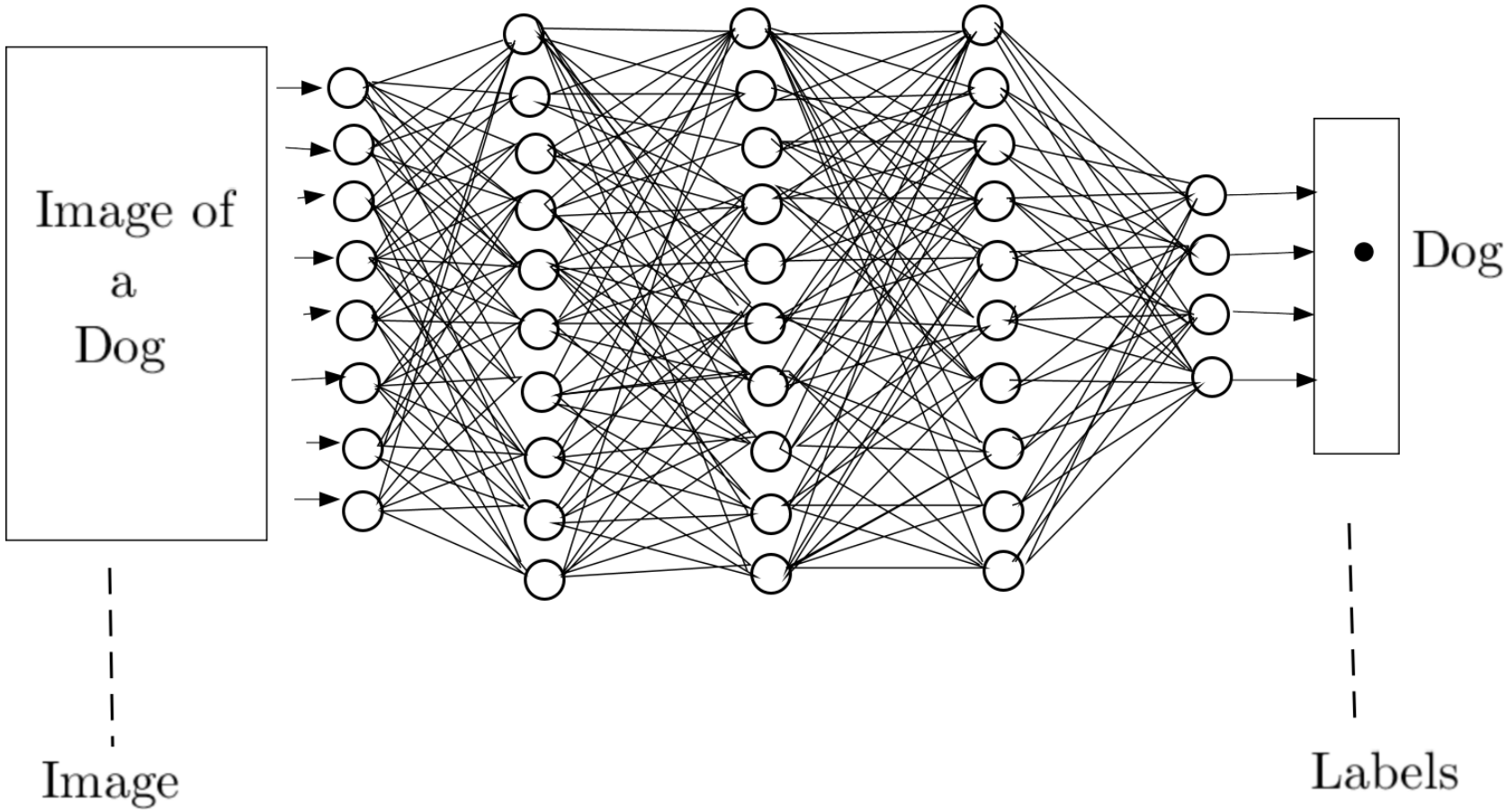


Figure 6: Deep Learning example

Deep Learning (DL): Learning Process

- Mathematically, using DL, we learn a function to map input X to output Y, with minimal loss on the test data.

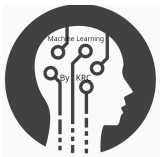
$$Y = f(X) + \varepsilon. \qquad (10)$$

- The real world is messy, hence the f may be complicated. In natural language problems, large vocabulary sizes mean lots of features. Vision problems involves lots of visual information in the form of pixels, and, ...

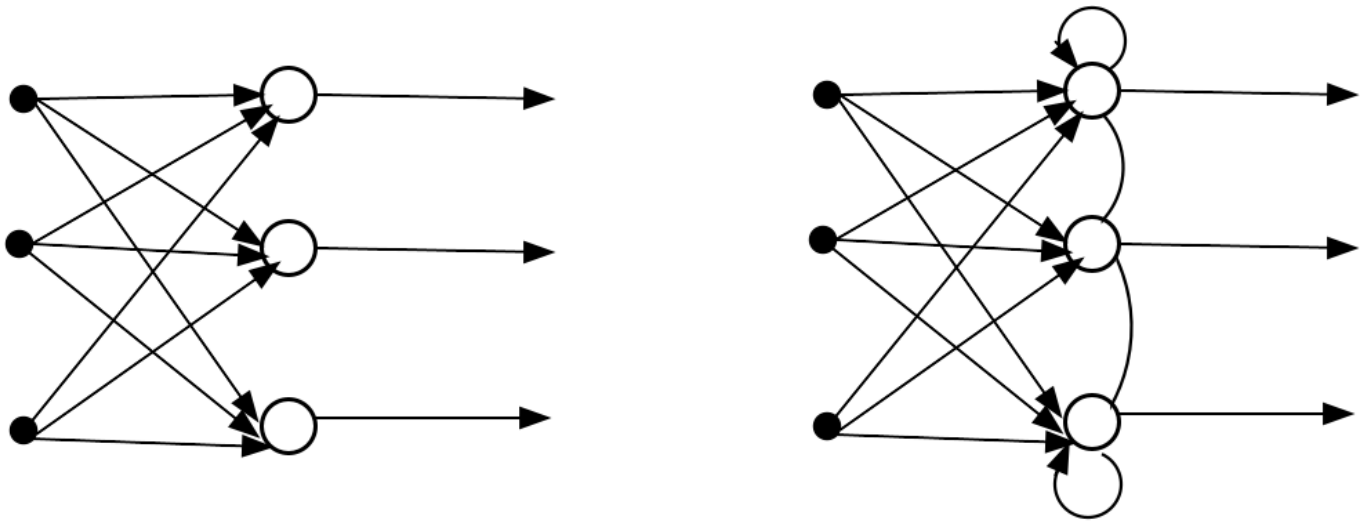
- The DL function f is good when data is complex.

- The neural networks are known as *universal function approximators* because they are able to learn any function, no matter how complex it is, with a single hidden layer.

- In fact, the equation 10 is a giant mathematical equation with millions of terms and equally large parameters.



Deep Learning: Feed foreward and Recurrent Networks



(a) Feedforward network

(b) Feedback network

Figure 7: Single layer feed-forward and Recurrent feedback networks

Deep Learning (DL): 1. Convolutional neural network

• CNN: The deep neural network that consists of hypotheses about images. These networks have many layers, organized in such a way that the output is less sensitive to the deviation from the original object, due to changes in input image.

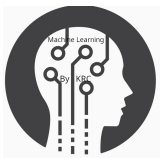
• In a typical CNN, there are several convolution layers, each followed by a pooling (subsampling) layer, and in the final stage a fully connected layer, identical to multilayer perceptron are common.

• The layers in the CNN have inputs $\mathbf{x}$ arranged in three dimensions, $m \times m \times r$

• An activation function f or a nonlineariy, applied to the output of the convolution layers (k) is:

$$h^k = f(W^k * \mathbf{x} + b^k). \quad (11)$$

where, W^k is weights, and b^k are biases.

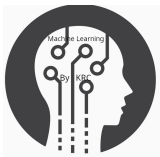


- These are widely used algorithms in deep learning, specially in NLP and speech processing. Unlike the traditional neural networks, RNN utilize the sequential information in the network.

- This property is essential in many applications where embedded structure in the data sequence convey useful knowledge.

- For example, to understand a word in a sentence, it is necessary to know its context, therefore, RNN can be seen as a short-term memory unit that include the input layer x, hidden layer h and output layer y.

- In a RNN, three deep RNN approaches include "Input-to-Hidden," "Hidden-to-Output," and "Hidden-to-Hidden" connections.



Deep Learning (DL): NN Learning

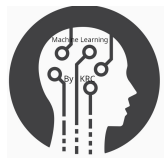
- All the three paradigms of learning exists in the neural networks, i.e, *supervised learning*, *unsupervised learning*, and *hybrid learning*.

- The unsupervised learning based neural networks explores the underlying structure in the data, or correlations between patterns in the data, and organizes patterns into categories based on the correlations.

The learning rules in the neural networks are of four basic types.

- *Error-Correction Rules*: During the training, the input $x_1, ..., x_n$ is applied to the network, and the flows through the network generates a set of values in the units of output y. Next step: actual output y is compared with the desired target d.

- *Boltzmann learning algorithm*: adjusts the connection weights in such a way that the desired probability distribution is satisfied by the states of the visible units.



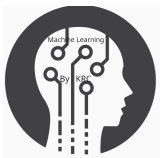
Deep Learning (DL): NN Learning ...

- *Hebbian Learning Rule*: Specifies the magnitude of the weight by which the connection between two units be increased/ decreased in proportion to the product of their activation.

$$w_{ij}(t+1) = w_{ij}(t) + \eta y_j(t).x_i(t) \quad (12)$$

- *Competitive Learning Rules*: The competitive-learning units compete among themselves for activation, which is in contrast to the Hebbian learning, where multiple output units can be fired together.

- Hence, only one output unit is active at any given time. The biological neurons follow this type of learning.



Deep Learning Applicationsg

- *Natural Language Processing*: Sentimental analysis, Machine Translation, Machine Translation, Summarization, Question Answering.
- *Visual Data Processing*: Image Classification, Object Detection and Semantic Segmentation, Video Processing, Visual Datasets.
- *Speech and Audio Processing*: Speech Emotion Recognition, Speech Enhancement.
- *Other Applications*: Information Retrieval, Transportation Prediction, Autonomous Driving, Biomedicine, Disaster Management Systems.