# DEEP LEARNING
# LINEAR ALGEBRA

M E Palanivel

## Introduction

- Linear algebra is a branch of mathematics that is widely used throughout science and engineering.
- A good understanding of linear algebra is essential for understanding and working with many machine learning algorithms, especially deep learning algorithms.

## Scalars, Vectors, Matrices and Tensors

- *Scalars*: A scalar is just a single number, in contrast to most of the other objects studied in linear algebra, which are usually arrays of multiple numbers.
- Example, we might say “Let $s \in$ R be the slope of the line,” while defining a real-valued scalar, or “Let $n \in$ N be the number of units,” while defining a natural number scalar.

## Vectors

- *Vectors*: A vector is an array of numbers. The numbers are arranged in order. We can identify each individual number by its index in that ordering.
- Typically we give vectors lower case names written in bold typeface, such as *x*.
- The elements of the vector are identified by writing its name in italic typeface, with a subscript.
- The first element of *x* is *x*1 , the second element is *x*2 and so on. We also need to say what kind of numbers are stored in the vector.
- If each element is in R, and the vector has *n* elements, then the vector lies in the set formed by taking the Cartesian product of R *n* times, denoted as R*n*. When we need to explicitly identify the elements of a vector, we write them as a column enclosed in square brackets:

$$\boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}.$$

- Sometimes we need to index a set of elements of a vector. In this case, we define a set containing the indices and write the set as a subscript.
- For example, to access $x1$ , $x3$ and $x6$, we define the set $S = \{1, 3, 6\}$ and write $x_S$.
- We use the − sign to index the complement of a set.
- For example $x_{-1}$ is the vector containing all elements of $x$ except for $x_1$ , and $x_{-S}$ is the vector containing all of the elements of $x$ except for $x_1$, $x_3$ and $x_6$ .

## MATRICES

- A matrix is a 2-D array of numbers, so each element is identified by

two indices instead of just one.

- We usually give matrices upper-case variable names with bold typeface, such as *A*.
- If a real-valued matrix $A$ has a height of $m$ and a width of $n$, then we say that $A \in$ R$m \times n$.
- We usually identify the elements of a matrix using its name in italic but not bold font, and the indices are listed with separating commas.
- For example, $A_{1,1}$ is the upper left entry of $A$ and $A_{m,n}$ is the bottom right entry.

## Tensors

- *Tensors*: In some cases we will need an array with more than two axes.
- In the general case, an array of numbers arranged on a regular grid with a variable number of axes is known as a *tensor.*
- We denote a tensor named “A” with this typeface: A.
- We identify the element of A at coordinates ($i$, $j$, $k$ ) by writing $A$ $i$,$j$,$k$.

## Linear Dependence and Span

- In order for $A^{-1}$ to exist, Eq. $Ax=b$ must have exactly one solution for every value of b.
- However, it is also possible for the system of equations to have no solutions or infinitely many solutions for some values of b.
- If both x and y are solutions then $z = \alpha x + (1 - \alpha)y$ is also a solution for any real $\alpha$.
- $$Ax = \sum x_i A_{:,i}.$$
- In general, this kind of operation is called a linear combination.
- Formally, a linear combination of some set of vectors $\{v^{(1)}, \ldots, v^{(n)}\}$ is given by multiplying each vector $v^{(i)}$ by a corresponding scalar coefficient and adding the results:

$$\sum c_i v^{(i)}$$

- The span of a set of vectors is the set of all points obtainable by linear combination of the original vectors.

## Cont.

- Determining whether $Ax = b$ has a solution thus amounts to testing whether b is in the span of the columns of A. This particular span is known as the column space or the range of A.
- In order for the system $Ax = b$ to have a solution for all values of $b \in R^m$, we therefore require that the column space of A be all of $R^m$ .
- If any point in $R^m$ is excluded from the column space, that point is a potential value of b that has no solution.
- The requirement that the column space of A be all of $R^m$ implies immediately that A must have at least m columns, i.e., $n \geq m$. Otherwise, the dimensionality of the column space would be less than m.
-

## Norms

- In machine learning, we usually measure the size of vectors using a function called a norm. Formally, the L p norm is given by

  $$||x||_p = [\sum |xi|^p]^{(1/p)} \quad \text{for } p \in R, p \geq 1.$$

- Norms, including the L p norm, are functions mapping vectors to non-negative values.
- A norm is any function f that satisfies the following properties:
- $f(x) = 0 \Rightarrow x = 0$
- $f(x + y) \leq f(x) + f(y)$ (the triangle inequality)
- $\forall \alpha \in R, f(\alpha x) = |\alpha| f(x)$
- The $L_2$ norm, with p = 2, is known as the Euclidean norm.
- It is simply the Euclidean distance from the origin to the point identified by x. The $L_2$ norm is used so frequently in machine learning that it is often denoted simply as $||x||$, with the subscript 2 omitted.

## Cont.

- It is also common to measure the size of a vector using the squared L 2 norm, which can be calculated simply as $x^T x$.
- The $L_1$ norm may be simplified to $||x||_1 = \sum |x_i|$
- One other norm that commonly arises in machine learning is the L∞ norm, also known as the max norm.
- This norm simplifies to the absolute value of the element with the largest magnitude in the vector, $||x||_\infty = \max |xi|$.
- In the context of deep learning, the most common way to do this is with the otherwise obscure Frobenius norm $||A||_F = \sqrt{\sum A_{i,j}^2}$, which is analogous to the $L_2$ norm of a vector.
- The dot product of two vectors can be rewritten in terms of norms.

$$x^T y = ||x||_2 ||y||_2 \cos\theta$$, where $\theta$ is the angle between x and y.

## Special Kinds of Matrices and Vectors

- A symmetric matrix is any matrix that is equal to its own transpose:
- $A = A^T$
- A unit vector is a vector with unit norm: $||x||2 = 1$.
- A vector x and a vector y are orthogonal to each other if $x^T y = 0$
- If the vectors are not only orthogonal but also have unit norm, we call them orthonormal.
- An orthogonal matrix is a square matrix whose rows are mutually orthonormal and whose columns are mutually orthonormal:
- $$A^T A = A A^T = I$$
- This implies that $A^{-1} = A^T$

## Eigen decomposition

- One of the most widely used kinds of matrix decomposition is called eigen decomposition, in which we decompose a matrix into a set of eigenvectors and eigenvalues.
- An eigenvector of a square matrix A is a non-zero vector v such that multiplication by A alters only the scale of v: $Av = \lambda v$.
- The scalar $\lambda$ is known as the eigenvalue corresponding to this eigenvector. (One can also find a left eigenvector such that $v^T A = \lambda v^T$ , but we are usually concerned with right eigenvectors).

## Example : Eigen Decomposition of a 2×2 Matrix

- **Problem**

  Find the eigenvalues and eigenvectors of

$$A = \begin{bmatrix} 4 & 1 \\ 2 & 3 \end{bmatrix}$$

**Step 1: Characteristic Equation**

$$\begin{vmatrix} 4-\lambda & 1 \\ 2 & 3-\lambda \end{vmatrix} \quad = (4-\lambda)(3-\lambda) - 2$$

$$= \lambda^2 - 7\lambda + 10 = 0$$

**Step 2: Eigenvalues**

$$(\lambda - 5)(\lambda - 2) = 0$$

$$\lambda_1 = 5, \lambda_2 = 2$$

**Step 3: Eigenvectors**

**For $\lambda = 5$**

$$(A - 5I)x = 0 \Rightarrow \begin{bmatrix} -1 & 1 \\ 2 & -2 \end{bmatrix}$$

$$x = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

- **For $\lambda = 2$**

$$(A - 2I)x = 0 \Rightarrow \begin{bmatrix} 2 & 1 \\ 2 & 1 \end{bmatrix}$$

$$x = \begin{bmatrix} 1 \\ -2 \end{bmatrix}$$

- **Final Eigen Decomposition**

$$A = Q\Lambda Q^{-1}$$

$$Q = \begin{bmatrix} 1 & 1 \\ 1 & -2 \end{bmatrix}, \Lambda = \begin{bmatrix} 5 & 0 \\ 0 & 2 \end{bmatrix}$$

**Apply Formula**

$$Q^{-1} = \frac{1}{-3}\begin{bmatrix} -2 & -1 \\ -1 & 1 \end{bmatrix}$$

## CONT.

- If v is an eigenvector of A, then so is any rescaled vector sv for $s \in R$, $s \neq 0$. Moreover, sv still has the same eigenvalue. For this reason, we usually only look for unit eigenvectors.
- Suppose that a matrix A has n linearly independent eigenvectors, $\{v^{(1)}, \ldots, v^{(n)}\}$, with corresponding eigenvalues $\{\lambda_1, \ldots, \lambda_n\}$.
- We may concatenate all eigenvectors to form a matrix V with one eigenvector per column:

  $V = [v^{(1)}, \ldots, v^{(n)}]$. Likewise, we can concatenate the eigenvalues to form a vector $\lambda = [\lambda_1, \ldots, \lambda_n]$

- $$A = V \operatorname{diag}(\lambda) V^{-1}$$

## Cont.

- We have seen that constructing matrices with specific eigenvalues and eigenvectors allows us to stretch space in desired directions.
- However, we often want to decompose matrices into their eigenvalues and eigenvectors.
- Doing so can help us to analyze certain properties of the matrix, much as decomposing an integer into its prime factors can help us understand the behavior of that integer.
- Every real symmetric matrix can be decomposed into an expression using only real-valued eigenvectors and eigenvalues:

$$A = Q \Lambda Q^T$$

where Q is an orthogonal matrix composed of eigenvectors of A, and $\Lambda$ is a diagonal matrix.

## CONT.

- The eigenvalue $\Lambda_{i,i}$ is associated with the eigenvector in column i of Q, denoted as Q:,i. Because Q is an orthogonal matrix, we can think of A as scaling space by $\lambda_i$ in direction v(i).
- The eigen decomposition of a matrix tells us many useful facts about the matrix.
- The matrix is singular if and only if any of the eigenvalues are zero.
- The eigen decomposition of a real symmetric matrix can also be used to optimize quadratic expressions of the form $f(x) = x^{T}Ax$ subject to $\|x\|_2 = 1$. Whenever x is equal to an eigenvector of A, f takes on the value of the corresponding eigenvalue.
- The maximum value of f within the constraint region is the maximum eigenvalue and its minimum value within the constraint region is the minimum eigenvalue.

## Singular Value Decomposition

- we saw how to decompose a matrix into eigenvectors and eigenvalues.
- The singular value decomposition (SVD) provides another way to factorize a matrix, into singular vectors and singular values.
- The SVD allows us to discover some of the same kind of information as the eigen decomposition.
- However, the SVD more generally applicable.
- Every real matrix has a singular value decomposition, but the same is not true of the eigenvalue decomposition.
- For example, if a matrix is not square, the eigen decomposition is not defined, and we must use a singular value decomposition instead.

## Example for Singular Value Decomposition

- Find the SVD of $A = \begin{bmatrix} 3 & 0 \\ 0 & 4 \end{bmatrix}$
- **step 1: Compute $A^T A$**

$$A^T A = \begin{bmatrix} 3 & 0 \\ 0 & 4 \end{bmatrix} \begin{bmatrix} 3 & 0 \\ 0 & 4 \end{bmatrix} = \begin{bmatrix} 9 & 0 \\ 0 & 16 \end{bmatrix}$$

- **Step 2: Eigenvalues of $A^T A$**

$$\lambda_1 = 16, \lambda_2 = 9$$

- **Step 3: Singular Values**

$$\sigma_1 = \sqrt{16} = 4, \sigma_2 = \sqrt{9} = 3$$

- **Step 4: Right Singular Vectors ($V$)**

  Eigenvectors of $A^T A$:

$$V = I$$

- **Step 5: Left Singular Vectors ($U$)**
- $U = AV\Sigma^{-1} = I$
- **Final SVD**
- $\boxed{A = U\Sigma V^T = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 4 & 0 \\ 0 & 3 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}}$

- **Given Matrix**

$$A = \begin{bmatrix} 1 & 2 \\ 2 & 1 \end{bmatrix}$$

- **Step 1: Compute $A^T A$**

$$A^T A = \begin{bmatrix} 5 & 4 \\ 4 & 5 \end{bmatrix}$$

- **Step 2: Eigenvalues & Singular Values**

$$\lambda_1 = 9, \lambda_2 = 1$$
$$\sigma_1 = 3, \sigma_2 = 1$$

- **Step 3: Right Singular Vectors ($V$)**

$$\text{Solve } (A^T A - 9I)v = 0:$$
$$\Rightarrow v_1 = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

Solve $(A^T A - I)v = 0$:

$$\Rightarrow v_2 = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ -1 \end{bmatrix}$$
$$V = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{bmatrix}$$

- **Step 4: Left Singular Vectors ($U$)**

**For $\sigma_1 = 3$**

$$u_1 = \frac{1}{3}Av_1 = \frac{1}{\sqrt{2}}\begin{bmatrix}1\\1\end{bmatrix}$$

**For $\sigma_2 = 1$**

$$u_2 = Av_2 = \frac{1}{\sqrt{2}}\begin{bmatrix}1\\-1\end{bmatrix}$$

$$U = \begin{bmatrix}\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}}\\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}}\end{bmatrix}$$

- **Final SVD**

$$\boxed{A = U\Sigma V^T}$$

## Cont.

- Recall that the eigen decomposition involves analyzing a matrix A to discover a matrix V of eigenvectors and a vector of eigenvalues λ such that we can rewrite A as

$$A = V \operatorname{diag}(\lambda) V^{-1} .$$

- The singular value decomposition is similar, except this time we will write A as a product of three matrices:

$$A = UDV^{T}.$$

- Suppose that A is an $m \times n$ matrix. Then U is defined to be an $m \times m$ matrix, D to be an $m \times n$ matrix, and V to be an $n \times n$ matrix.
- The elements along the diagonal of D are known as the singular values of the matrix A.
- The columns of U are known as the left-singular vectors.
- The columns of V are known as the right-singular vectors.

- We can actually interpret the singular value decomposition of A in terms of the eigen decomposition of functions of A .
- The left-singular vectors of A are the eigenvectors of $AA^T$.
- The right-singular vectors of A are the eigenvectors of $A^TA$.
- The non-zero singular values of A are the square roots of the eigenvalues of $A^T A$.
- The same is true for $AA^T$.

## The Trace Operator

- The trace operator gives the sum of all of the diagonal entries of a matrix:
- $\mathrm{Tr}(A) = \sum A_{i,i}$.
- The trace operator is useful for a variety of reasons.
- Some operations that are difficult to specify without resorting to summation notation can be specified using matrix products and the trace operator.
- example, the trace operator provides an alternative way of writing the Frobenius norm of a matrix: $\|A\|_F = \sqrt{\mathrm{Tr}(AA^T)}$.
- For example, the trace operator is invariant to the transpose operator: $\mathrm{Tr}(A) = \mathrm{Tr}(A^T)$.
- , if the shapes of the corresponding matrices allow the resulting product to be defined:

## Cont.

- $\mathrm{Tr}(ABC) = \mathrm{Tr}(CAB) = \mathrm{Tr}(BCA)$ or more generally,
- $\mathrm{Tr}(\prod F^{(i)}) = \mathrm{Tr}(F^{(n)} \prod F^{(i)})$.

This invariance to cyclic permutation holds even if the resulting product has a different shape.

- For example, for $A \in Rm\times n$ and $B \in Rn\times m$, we have $\mathrm{Tr}(AB) = \mathrm{Tr}(BA)$ even though $AB \in R^{m\times m}$ and $BA \in R^{n\times n}$ .

## The Determinant

- The determinant of a square matrix, denoted det(A), is a function mapping matrices to real scalars.
- The determinant is equal to the product of all the eigenvalues of the matrix. The absolute value of the determinant can be thought of as a measure of how much multiplication by the matrix expands or contracts space.
- If the determinant is 0, then space is contracted completely along at least one dimension, causing it to lose all of its volume. If the determinant is 1, then the transformation is volume-preserving.

## Probability and Information Theory

- Probability theory is a mathematical framework for representing uncertain statements.
- In artificial intelligence applications, we use probability theory in two major ways.

1. The laws of probability tell us how AI systems should reason, so we design our algorithms to compute or approximate various expressions derived using probability theory.
2. we can use probability and statistics to theoretically analyze the behavior of proposed AI systems.

- Probability theory is a fundamental tool of many disciplines of science and engineering.
- Why Probability?

## Cont.

- Many branches of computer science deal mostly with entities that are entirely deterministic and certain. A programmer can usually safely assume that a CPU will execute each machine instruction flawlessly.
- It can be surprising that machine learning makes heavy use of probability theory.
- This is because machine learning must always deal with uncertain quantities, and sometimes may also need to deal with stochastic (non-deterministic) quantities.
- Uncertainty and stochasticity can arise from many sources.

## Cont.

- There are three possible sources of uncertainty:
- 1.Inherent stochasticity in the system being modeled.
- For example, most interpretations of quantum mechanics describe the dynamics of subatomic particles as being probabilistic. We can also create theoretical scenarios that we postulate to have random dynamics, such as a hypothetical card game where we assume that the cards are truly shuffled into a random order.
- 2. Incomplete observability. Even deterministic systems can appear stochastic when we cannot observe all of the variables that drive the behavior of the system.

## Cont.

- For example, in the Monty Hall problem, a game show contestant is asked to choose between three doors and wins a prize held behind the chosen door. Two doors lead to a goat while a third leads to a car. The outcome given the contestant's choice is deterministic, but from the contestant's point of view, the outcome is uncertain.
- 3. Incomplete modeling. When we use a model that must discard some of the information we have observed, the discarded information results in uncertainty in the model's predictions. For example, suppose we build a robot that can exactly observe the location of every object around it.

## Cont.

- If the robot discretizes space when predicting the future location of these objects, then the discretization makes the robot immediately become uncertain about the precise position of objects: each object could be anywhere within the discrete cell that it was observed to occupy.

## Random Variables

- A random variable is a variable that can take on different values randomly.
- x1 and x2 are both possible values that the random variable x can take on.
- For vector-valued variables, we would write the random variable as x and one of its values as x.
- A random variable is just a description of the states that are possible; it must be coupled with a probability distribution that specifies how likely each of these states are.
- Random variables may be discrete or continuous. A discrete random variable is one that has a finite or countably infinite number of states.
- Note that these states are not necessarily the integers; they can also just be named states that are not considered to have any numerical value.
- A continuous random variable is associated with a real value.

## Probability Distributions

- A probability distribution is a description of how likely a random variable or set of random variables is to take on each of its possible states. The way we describe probability distributions depends on whether the variables are discrete or continuous.
- Discrete Variables and Probability Mass Functions
- A probability distribution over discrete variables may be described using a probability mass function (PMF). We typically denote probability mass functions with a capital P.
- Often we associate each random variable with a different probability mass function and the reader must infer which probability mass function to use based on the identity of the random variable, rather than the name of the function; P(x) is usually not the same as P(y).
- To be a probability mass function on a random variable x, a function P must satisfy the following properties:

• The domain of P must be the set of all possible states of x.

• $\forall x \in x, 0 \leq P(x) \leq 1$. An impossible event has probability 0 and no state can be less probable than that. Likewise, an event that is guaranteed to happen has probability 1, and no state can have a greater chance of occurring.

## Cont.

- $\sum P(x) = 1$. We refer to this property as being normalized. Without this property, we could obtain probabilities greater than one by computing the probability of one of many events occurring.

Example: consider a single discrete random variable x with k different states. We can place a uniform distribution on x make each of its states equally likely by setting its probability mass function to $P(x = x_i) = 1/k$ for all i.

We can see that this fits the requirements for a probability mass function.

The value $1/k$ is positive ,because k is a positive integer.

$\sum P(x = x_i) = \sum 1/k = k/k = 1$,

so the distribution is properly normalized.

## Continuous Variables and Probability Density Functions

- When working with continuous random variables, we describe probability distributions using a probability density function (PDF) rather than a probability mass function.
- To be a probability density function, a function p must satisfy the following properties:

• The domain of p must be the set of all possible states of x.

• $\forall x \in x$, $p(x) \geq 0$. Note that we do not require $p(x) \leq 1$.

• $\int p(x)dx = 1$.

- A probability density function p(x) does not give the probability of a specific state directly, instead the probability of landing inside an infinitesimal region with volume $\delta x$ is given by $p(x)\delta x$.

## Marginal Probability

- we know the probability distribution over a set of variables and we want to know the probability distribution over just a subset of them. The probability distribution over the subset is known as the marginal probability distribution.
- Example: suppose we have discrete random variables x and y, and we know
- P(x, y).
- We can find P(x) with the sum rule: $\forall x \in x$, $P(x = x) = \sum P(x = x, y = y)$.
- For continuous variables, we need to use integration instead of summation: $p(x) = \int p(x, y)dy$.

## Conditional Probability

- we are interested in the probability of some event, given that some other event has happened. This is called a conditional probability. We denote the conditional probability that y = y given x = x as P(y = y | x = x). This conditional probability can be computed with the formula
- $P(y = y \mid x = x) = P(y = y, x = x)/ P(x = x)$ .
- The conditional probability is only defined when $P(x = x) > 0$

## The Chain Rule of Conditional Probabilities

- Any joint probability distribution over many random variables may be decomposed into conditional distributions over only one variable:

$$P(x^{(1)}, \ldots, x^{(n)}) = P(x^{(1)})\Pi\, P(x^{(i)} \mid x^{(1)}, \ldots, x^{(i-1)}).$$

This is known as the chain rule or product rule of probability.

$$P(a, b, c) = P(a \mid b, c)P(b, c)$$

$$P(b, c) = P(b \mid c)P(c)$$

$$P(a, b, c) = P(a \mid b, c)P(b \mid c)P(c).$$

## Independence and Conditional Independence

- Two random variables x and y are independent if their probability distribution can be expressed as a product of two factors, one involving only x and one involving only y:

$$\forall x \in x, y \in$$

- Two random variables x and y are conditionally independent given a random variable z if the conditional probability distribution over x and y factorizes in this way for every value of z:

$$\forall x \in x, y \in y, z \in z, p(x = x, y = y \mid z = z) = p(x = x \mid z = z)p(y = y \mid z = z)$$

## Expectation, Variance and Covariance

- The expectation or expected value of some function f(x) with respect to a probability distribution P(x) is the average or mean value that f takes on when x is drawn from P. For discrete variables this can be computed with a summation: $E_{x\sim P}\,[f(x)] = \sum\; P(x)f(x)$,
- while for continuous variables, it is computed with an integral:
- $E_{x\sim p}[f(x)] = \int p(x)f(x)dx$.
- Expectations are linear, for example, $Ex[\alpha f(x) + \beta g(x)] = \alpha Ex[f(x)] + \beta Ex[g(x)]$, when $\alpha$ and $\beta$ are not dependent on x. The variance gives a measure of how much the values of a function of a random variable x vary as we sample different values of x from its probability distribution:

## CONT.

- $\mathrm{Var}(f(x)) = E[(f(x) - E[f(x)])^2]$.
- Expectations are linear, for example,
- $E_x[\alpha f(x) + \beta g(x)] = \alpha E_x[f(x)] + \beta E_x[g(x)]$,
- when $\alpha$ and $\beta$ are not dependent on x. The variance gives a measure of how much the values of a function of a random variable x vary as we sample different values of x from its probability distribution:
- $\mathrm{Var}(f(x)) = E[(f(x) - E[f(x)])^2]$.

## Common Probability Distributions

- **Bernoulli Distribution**
- The Bernoulli distribution is a distribution over a single binary random variable. It is controlled by a single parameter $\varphi \in [0, 1]$, which gives the probability of the random variable being equal to .
- properties:
- $P(x = 1) = \varphi$
- $P(x = 0) = 1 - \varphi$
- $P(x = x) = \varphi^{x} (1 - \varphi)^{(1-x)}$

- **Bernoulli Random Variable in ML**
- In ML, we often model a **binary label** as a Bernoulli random variable:
- Y∈{0,1},Y~Bernoulli(p)
- Where:
- $Y = 1 \rightarrow$ positive class (spam, disease present, fraud)
- $Y = 0 \rightarrow$ negative class (not spam, healthy, normal)
- $p = P(Y = 1 \mid X)$ →model's predicted probability
- **Bernoulli in Logistic Regression**
- **Model**
- Logistic regression directly models the **Bernoulli parameter** $p$:
- $p = \sigma(z) = \frac{1}{1+e^{-z}}, z = w^T x + b$
- Thus:
- $Y \mid X \sim Bernoulli(p)$

- **Bernoulli in Generative Models**
- **Bernoulli Likelihood**
- Used when data is binary:
- Pixel on/off (e.g., MNIST binarized images)
- Presence/absence features
- Example: **Variational Autoencoders (VAEs)**
- $p(x \mid z) = \prod_i Bernoulli\,(x_i \mid p_i)$
- Each pixel is modeled as a Bernoulli variable.
- **Bernoulli in Naive Bayes (Bernoulli NB)**
- Used when features are binary:
- $P(x_j \mid y) \sim Bernoulli(p_{jy})$
- Applications:
- Text classification (word present / not present)
- Spam filtering

- **Bernoulli in Regularization & Dropout**
- **Dropout**
- Dropout uses **Bernoulli random variables:**
- $d_i \sim Bernoulli(p)$
- $d_i = 1$ →neuron kept
- $d_i = 0$ →neuron dropped

**Bernoulli vs Gaussian in ML**

| Aspect | Bernoulli | Gaussian |
|---|---|---|
| Data type | Binary | Continuous |
| Output activation | Sigmoid | Linear |
| Loss | Binary Cross-Entropy | MSE |
| Use case | Classification | Regression |

## Real-World ML Examples

| Task | Bernoulli Usage |
|---|---|
| Spam detection | Email is spam / not spam |
| Medical diagnosis | Disease present / absent |
| Fraud detection | Fraud / not fraud |
| Click prediction | Click / no-click |

## Cont.

- $E_x[x] = \varphi$
- $\mathrm{Var}_x(x) = \varphi(1 - \varphi)$
- **Multinoulli Distribution**
- The multinoulli or categorical distribution is a distribution over a single discrete variable with k different states, where k is finite. The multinoulli distribution is parametrized by a vector $p \in [0, 1]^{k-1}$, where $p_i$ gives the probability of the $i^{-th}$ state.

## Gaussian Distribution

- The most commonly used distribution over real numbers is the normal distribution, also known as the Gaussian distribution:

  $$N(x; \mu, \sigma^2) = \sqrt{1/2\pi\sigma^2} \exp(-1/2\sigma^2)(x-\mu)^2 .$$

- The two parameters $\mu \in R$ and $\sigma \in (0,\infty)$ control the normal distribution. The parameter $\mu$ gives the coordinate of the central peak. This is also the mean of the distribution: $E[x] = \mu$. The standard deviation of the distribution is given by $\sigma$, and the variance by $\sigma^2$ .

- The normal distribution generalizes to $R_n$ , in which case it is known as the multivariate normal distribution. It may be parametrized with a positive definite symmetric matrix $\Sigma$:

  $$N(x; \mu, \Sigma) = \sqrt{(1/(2\pi)^n \det(\Sigma))} \exp((-½)(x-\mu)^T\Sigma^{-1}(x-\mu)) .$$

- We can instead use a precision matrix $\beta$:

  $$N(x; \mu, \beta -1) = \sqrt{(\det(\beta) / (2\pi)^n)} \exp((-1/2)(x-\mu)^T\beta(x-\mu)) .$$

## Exponential and Laplace Distributions

- In the context of deep learning, we often want to have a probability distribution with a sharp point at $x = 0$.
- To accomplish this, we can use the exponential distribution:
- $p(x; \lambda) = \lambda 1_{x\geq 0} \exp(-\lambda x)$.
- The exponential distribution uses the indicator function $1_{x\geq 0}$ to assign probability zero to all negative values of $x$.
- A closely related probability distribution that allows us to place a sharp peak of probability mass at an arbitrary point $\mu$ is the Laplace distribution
- $\text{Laplace}(x;\mu, \gamma) – (1/ 2\gamma) \exp(-|x - \mu| / \gamma)$ .

## The Dirac Distribution and Empirical Distribution

- we wish to specify that all of the mass in a probability distribution clusters around a single point. This can be accomplished by defining a PDF using the Dirac delta function , $\delta(x)$:

$$p(x) = \delta(x - \mu).$$

- A common use of the Dirac delta distribution is as a component of an empirical distribution,

$$\hat{p}(x) = (1/m) \sum \delta(x - x^{(i)})$$

- which puts probability mass 1/m on each of the m points $x^{(1)}, \ldots, x^{(m)}$ forming a given data set or collection of samples.
- Mixtures of Distributions
- One common way of combining distributions is to construct a mixture distribution. A mixture distribution is made up of several component distributions. On each trial, the choice of which component distribution generates the sample is determined by sampling a component identity from a multinoulli distribution:

$$P(x) = P(c = i)P(x \mid c = i)$$

- where P(c) is the multinoulli distribution over component identities.

## Useful Properties of Common Functions

- Certain functions arise often while working with probability distributions, especially the probability distributions used in deep learning models.
- One of these functions is the logistic sigmoid:
- $\sigma(x) = 1/ (1 + \exp(-x))$ .
- The logistic sigmoid is commonly used to produce the $\varphi$ parameter of a Bernoulli distribution because its range is (0,1), which lies within the valid range of values for the $\varphi$ parameter.
- Another commonly encountered function is the softplus function (Dugas et al., 2001):
- $\zeta(x) = \log (1 + \exp(x))$.
- The softplus function can be useful for producing the $\beta$ or $\sigma$ parameter of a normal distribution because its range is $(0,\infty)$.
- The name of the softplus function comes from the fact that it is a smoothed or “softened” version of $x^+ = \max(0, x)$.

## Cont.

- The following properties are all useful enough that you may wish to memorize them:

$$\sigma(x) = \exp(x) / (\exp(x) + \exp(0))$$

$$d/dx\,(\sigma(x)) = \sigma(x)(1 - \sigma(x))$$

$$1 - \sigma(x) = \sigma(-x)$$

$$\log \sigma(x) = -\zeta(-x)$$

$$d/dx\,\zeta(x) = \sigma(x)$$

$$\forall x \in (0, 1),\ \sigma^{-1}(x) = \log (x / 1 - x)$$

$$\forall x > 0,\qquad \zeta^{-1}(x) = \log (\exp(x) - 1)$$

$$\zeta(x) = \int \sigma(y)dy$$

$$\zeta(x) - \zeta(-x) = x$$

- The function σ−1(x) is called the logit in statistics, but this term is more rarely used in machine learning.

## Bayes' Rule

- We often find ourselves in a situation where we know $P(y \mid x)$ and need to know $P(x \mid y)$. Fortunately, if we also know $P(x)$,
- we can compute the desired quantity using Bayes' rule: $P(x \mid y) = P(x)P(y \mid x) P(y)$ .
- Note that while $P(y)$ appears in the formula, it is usually feasible to compute
- $P(y) = \sum P(y \mid x)P(x)$, so we do not need to begin with knowledge of $P(y)$.

# Machine Learning Basics

M E PALANIVEL
Professor

## Introduction

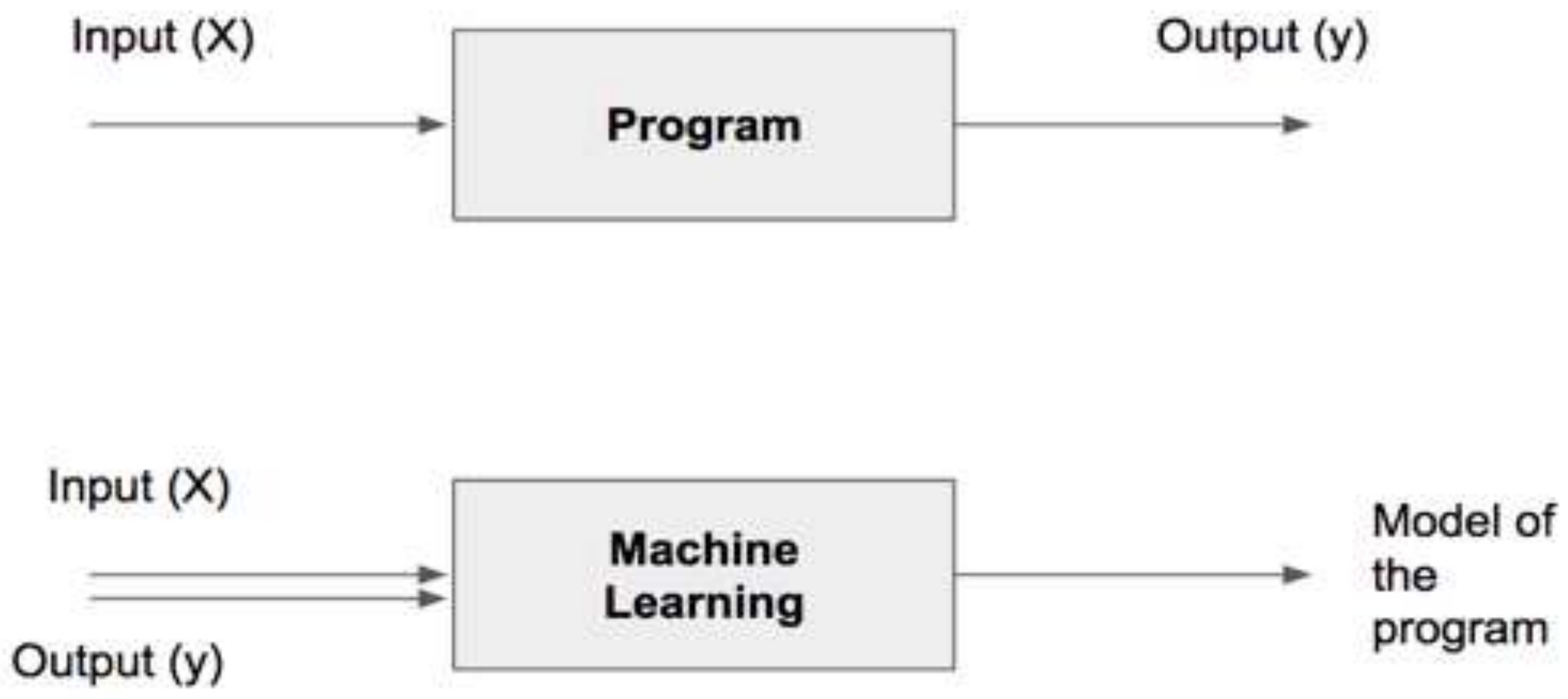
- Deep learning is a specific kind of machine learning.
- In order to understand deep learning well, one must have a solid understanding of the basic principles of machine learning.
- A machine learning algorithm is an algorithm that is able to learn from data.
- what do we mean by learning? Mitchell (1997) provides the definition
- There are multiple ways to define machine learning. But the one which is perhaps most relevant, concise and accepted universally is the one stated by Tom M. Mitchell, Professor of Machine Learning Department, School of Computer Science, Carnegie Mellon University. Tom M. Mitchell has defined machine learning as:

## Definition

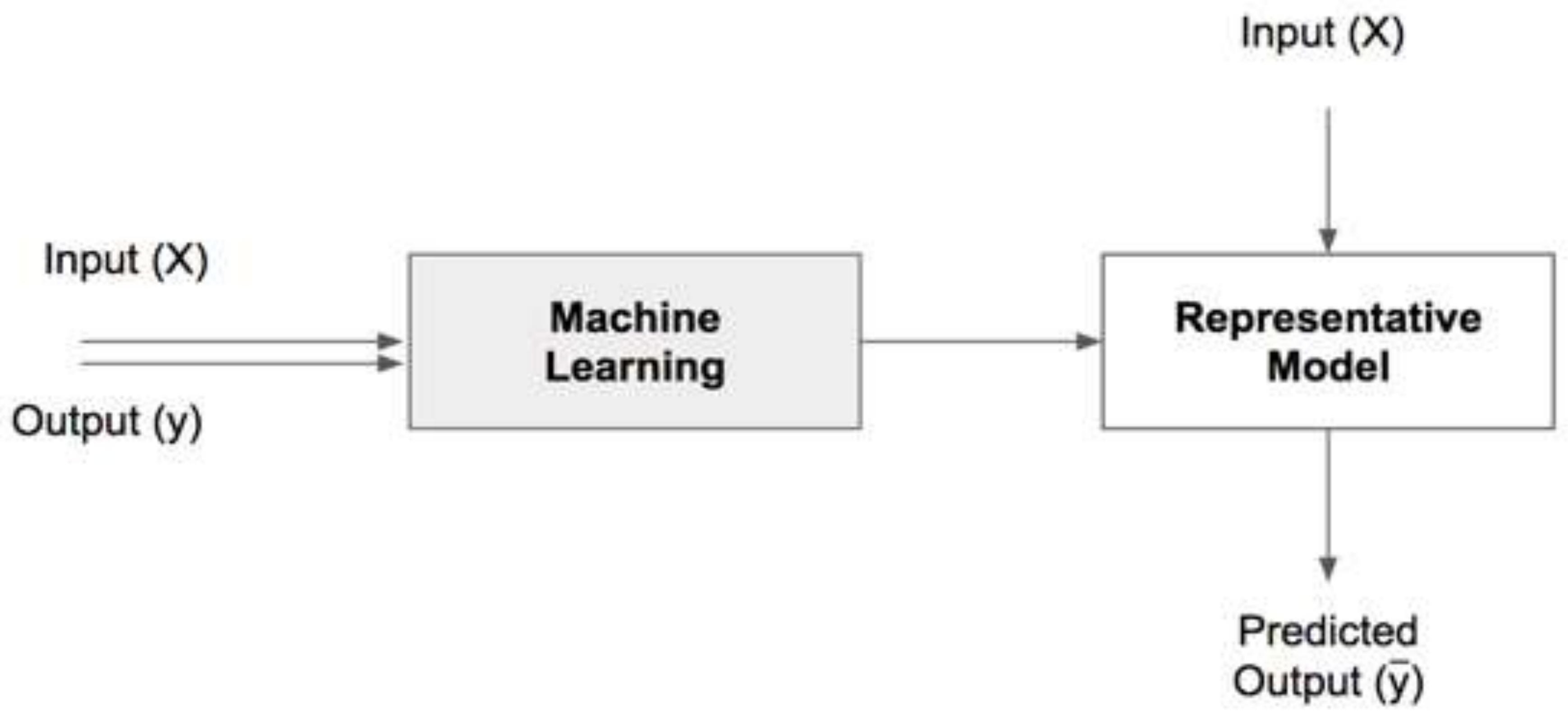
- "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E."
- A formal definition of the word "task," the process of learning itself is not the task. Learning is our means of attaining the ability to perform the task. For example, if we want a robot to be able to walk, then walking is the task.
- We could program the robot to learn to walk

- In order to evaluate the abilities of a machine learning algorithm, we must design a quantitative measure of its performance.
- Usually this performance measure *P* is specific to the task *T* being carried out by the system.
- Machine learning algorithms can be broadly categorized as *unsupervised* or *supervised* by what kind of experience they are allowed to have during the learning process.
- Most of the learning algorithms in this book can be understood as being allowed to experience an entire *dataset*.
- It means a machine can be considered to learn if it is able to gather experience by doing a certain task and improve its performance in doing the similar tasks in the future.

# Traditional programming VS machine learning.



## Data science models



## THE MACHINE LEARNING PROCESS

- **Data Collection and Preparation :** To solve a problem we have to collect the from scratch.
- If the problem is completely new, so that appropriate data can be chosen, then this process should be merged with the next step of feature selection, so that only the required data is collected.
- For supervised learning, target data is also needed, which can require the involvement of experts in the relevant field and significant investments of time.
- Finally, the quantity of data needs to be considered. Machine learning algorithms need significant amounts of data, preferably without too much noise, but with increased dataset size comes increased computational costs, and the sweet spot at which there is enough data without excessive computational overhead is generally impossible to predict.

## Cont.

- **Feature Selection :**when  we looked at possible features that might be useful for coin recognition.
- It consists of identifying the features that are most useful for the problem under examination.
- This invariably requires prior knowledge of the problem and the data; our common sense was used in the coins example above to identify some potentially useful features and to exclude others.
- As well as the identification of features that are useful for the learner, it is also necessary that the features can be collected without significant expense or time, and that they are robust to noise and other corruption of the data that may arise in the collection process.

## Cont.

- **Algorithm Choice :** Given the dataset, the choice of an appropriate algorithm (or algorithms) is what this book should be able to prepare you for, in that the knowledge of the underlying principles of each algorithm and examples of their use is precisely what is required for this.
- **Parameter and Model Selection :** For many of the algorithms there are parameters that have to be set manually, or that require experimentation to identify appropriate values.
- **Training:** Given the dataset, algorithm, and parameters, training should be simply the use of computational resources in order to build a model of the data in order to predict the outputs on new data.

## Cont.

- **Evaluation :**Before a system can be deployed it needs to be tested and evaluated for accuracy on data that it was not trained on.
- This can often include a comparison with human experts in the field, and the selection of appropriate metrics for this comparison.

## MODEL REPRESENTATION AND INTERPRETABILITY

- The goal of supervised machine learning is to learn or derive a target function which can best determine the target variable from the set of input variables.
- It is an extent of generalization.
- The input data is just a limited, specific view and the new, unknown data in the test data set may be differing quite a bit from the training data.
- Fitness of a target function approximated by a learning algorithm determines how correctly it is able to classify a set of data it has never seen.

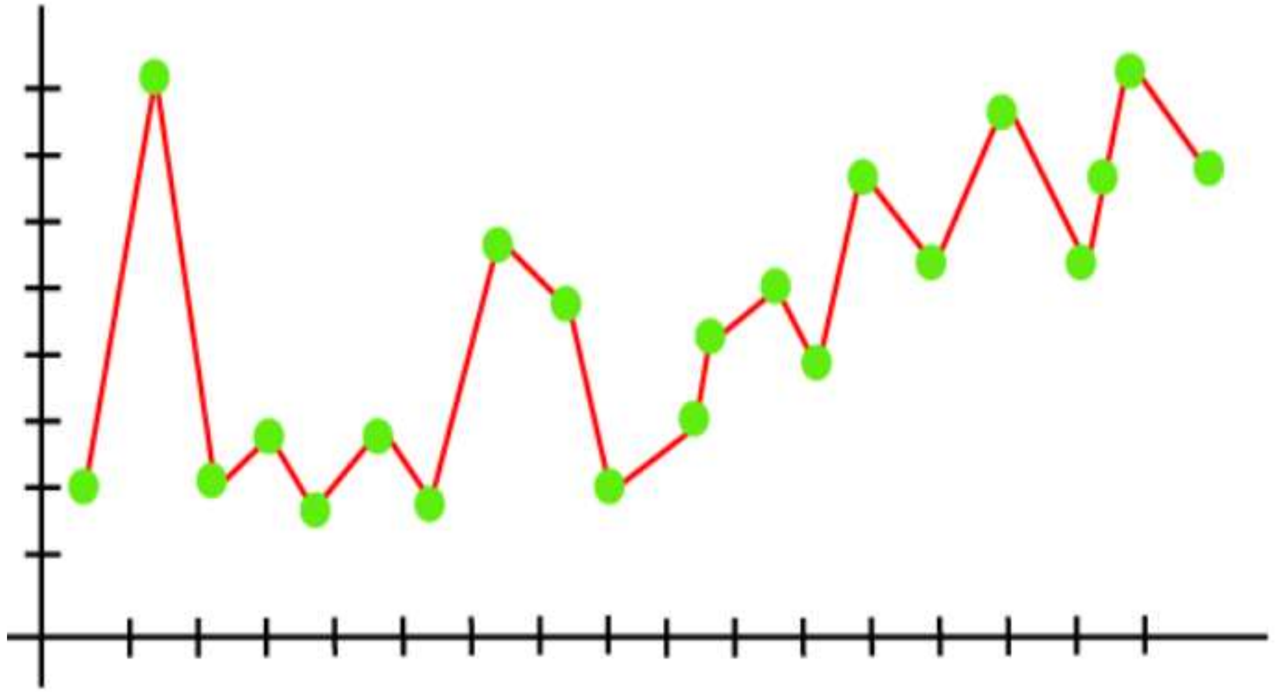
## Fitting: Overfitting and Underfitting in ML

- Overfitting and Underfitting are the two main problems that occur in machine learning and degrade the performance of the machine learning models.
- The main goal of each machine learning model is **to generalize well**. Here **generalization** defines the ability of an ML model to provide a suitable output by adapting the given set of unknown input.
- It means after providing training on the dataset, it can produce reliable and accurate output.
- Hence, the **underfitting** and **overfitting** are the two terms that need to be checked for the performance of the model and whether the model is generalizing well or not.

## Overfitting

- Overfitting occurs when our machine learning model **tries to cover all the data points or more than the required data points present in the given dataset.**
- Because of this, the model starts **caching noise and inaccurate values present in the dataset**, and all these factors reduce the efficiency and accuracy of the model.
- The overfitted model has **low bias** and **high variance.**
- The chances of occurrence of overfitting increase as much we provide training to our model. It means the more we train our model, the more chances of occurring the overfitted model.
- Overfitting is the main problem that occurs in S-L.

## Ex: Linear Regression O/P



As we can see from the above graph,
the model tries to cover all the data points
present in the scatter plot. It may look
efficient, but in reality, it is not so.

Because the goal of the regression model
to find the best fit line,

but here we have not got any best fit,
so, it will generate the prediction errors

## How to avoid the overfitting in model

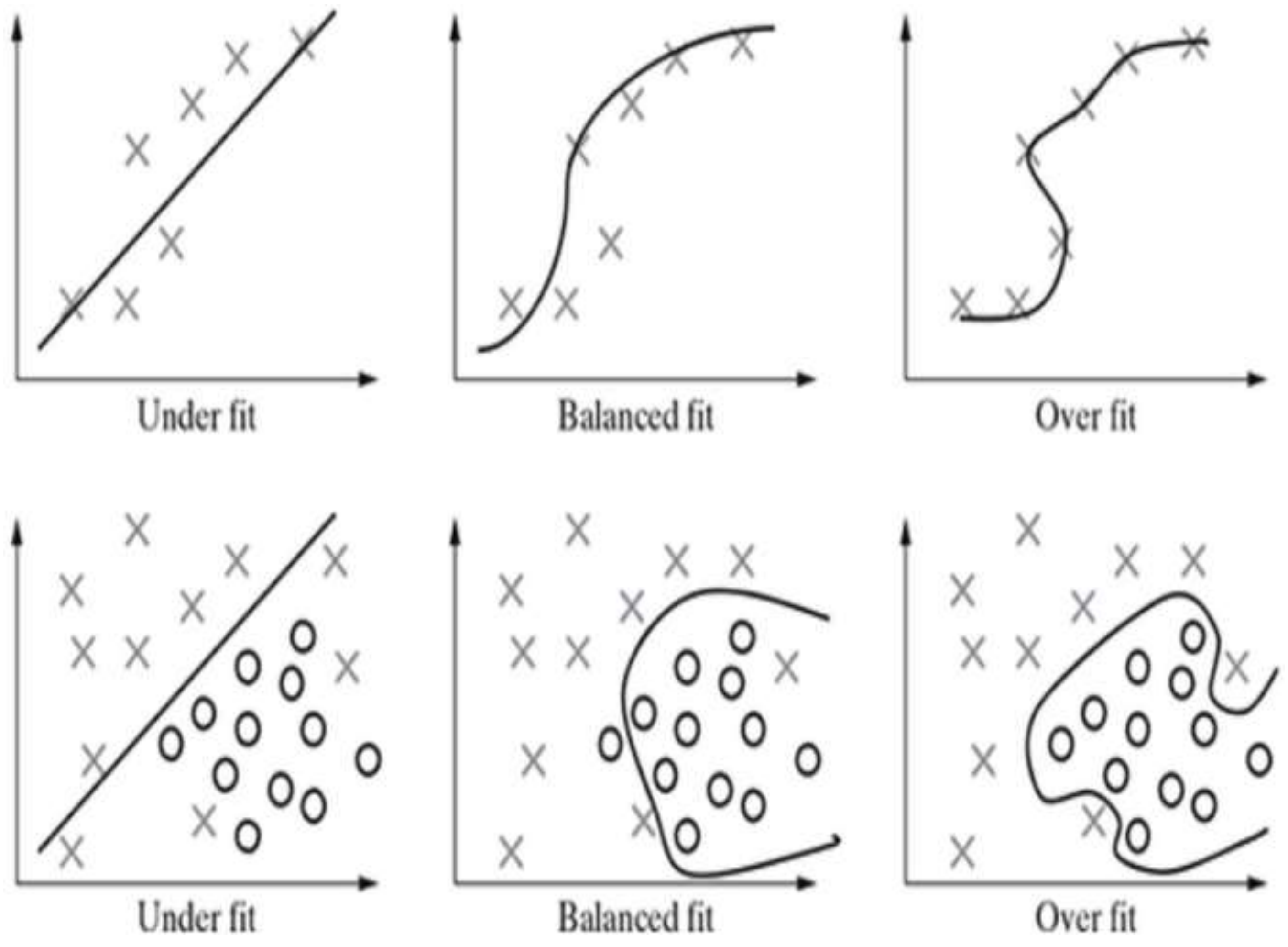
- Both overfitting and underfitting cause the degraded performance of the machine learning model. But the main cause is overfitting, so there are some ways by which we can reduce the occurrence of overfitting in our model.
- **Cross-Validation**
- **Training with more data**
- **Removing features**
- **Early stopping the training**
- **Regularization**
- **Ensembling**

## Underfitting

- A typical case of underfitting may occur when trying to represent a non-linear data with a linear model
- Many times underfitting happens due to unavailability of sufficient training data.
- Underfitting results in both poor performance with training data as well as poor generalization to test data.
- Underfitting can be avoided by

1. using more training data
2. reducing features by effective feature selection

## Underfitting and Overfitting of models

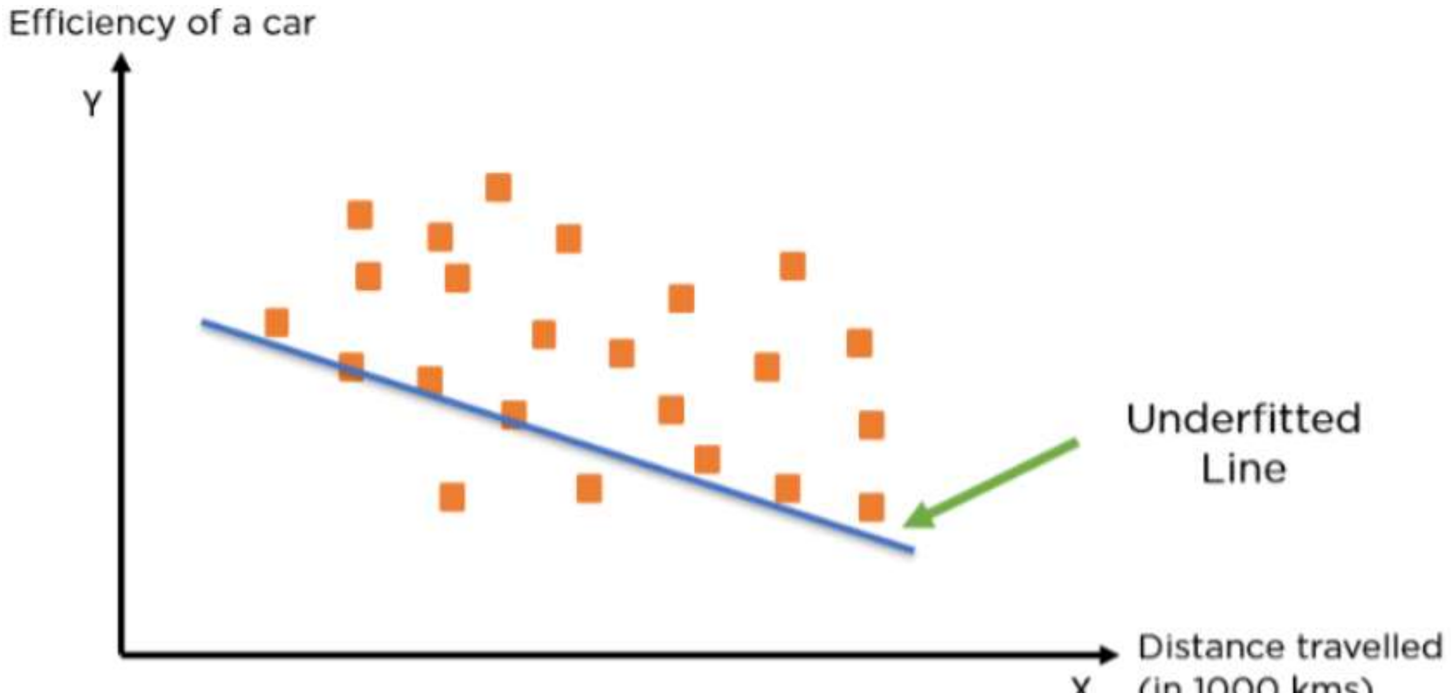


## Cont.

- Overfitting refers to a situation where the model has been designed in such a way that it emulates the training data too closely.
- any specific deviation in the training data, like noise or outliers, gets embedded in the model.
- It adversely impacts the performance of the model on the test data.
- Overfitting, in many cases, occur as a result of trying to fit an excessively complex model to closely match the training data.
- The target function, in these cases, tries to make sure all training data points are correctly partitioned by the decision boundary.
- This exact nature is not replicated in the unknown test data set.
- The target function results in wrong classification in the test data set.

## What is Underfitting?

- When a model has not learned the patterns in the training data well and is unable to generalize well on the new data, it is known as underfitting.
- An underfit model has poor performance on the training data and will result in unreliable predictions.
- Underfitting occurs due to **high bias and low variance.**



## Reasons for Underfitting

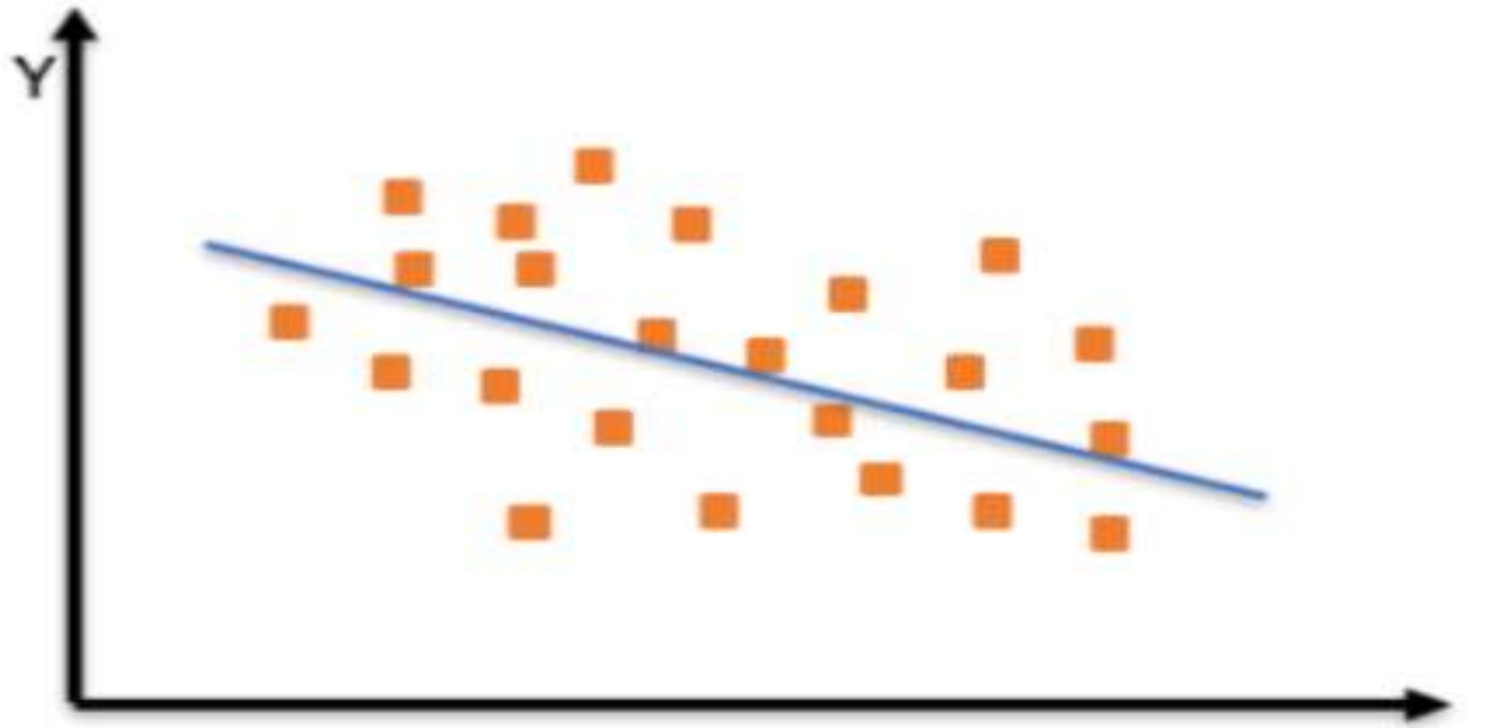
- Data used for training is not cleaned and contains noise (garbage values) in it
- The model has a high bias
- The size of the training dataset used is not enough
- The model is too simple

**Ways to Tackle Underfitting**

- Increase the number of features in the dataset
- Increase model complexity
- Reduce noise in the data
- Increase the duration of training the data

## What Is a Good Fit In Machine Learning?

- To find the good fit model, you need to look at the performance of a machine learning model over time with the training data.
- As the algorithm learns over time, the error for the model on the training data reduces, as well as the error on the test dataset.
- If you train the model for too long, the model may learn the unnecessary details and the noise in the training set and hence lead to overfitting.
- In order to achieve a good fit, you need to stop training at a point where the error starts to increase.



Good Fit

## Overfitting

- Overfitting can be avoided by

1. using re-sampling techniques like k-fold cross validation
2. hold back of a validation data set
3. remove the nodes which have little or no predictive power for the given machine learning problem.

- Both underfitting and overfitting result in poor classification quality which is reflected by low classification accuracy.

## Estimat ors

- An estimator is a **function or algorithm** that tries to estimate a target variable from given input features.
- Point Estimation is the attempt to provide the single best prediction of some quantity of interest.
- Quantity of interest can be:
  - A single parameter
  - A vector of parameters — e.g., weights in linear regression
  - A whole function

## Point estimator

- A point estimator is just a **formula or function** that gives you a best guess for an unknown value (parameter) based on data.
- We write this guess as:

  q = true value

  q̂ (read as "q hat") = estimated value (from data)

**Example:**

- You want to estimate the **average exam score** of all students in a school.
  But you only collect the scores of **5 students**:
- Scores: 78, 85, 90, 82, 75
- Point estimate of the average score (mean):

$$\hat{\mu} = \frac{78+85+90+82+75}{5} = 82$$

- So, $\hat{\mu} = 82$ is your **point estimate** of the true average score.

- ✓ **Example 2:**
- ✓ Out of the past **30 days**, it rained on **12 days**.
- ✓ Point estimate of the **probability of rain**:
- ✓ p^=12/30=0.4
- ✓ So, $\hat{p} = 0.4$ is your **point estimate** of the true probability of rain on a given day.

## Bias and Variance

- These are one of the ways to evaluate a machine-learning model.
- There are two types of error in machine learning:
- **1.Reducible error and 2.Irreducible error.**
- **Bias and Variance come under reducible error**.
- Bias is the error occurring between the model's **predicted value** and the **actual value.**
- Let *Y* be the true value of a parameter, and
- let Y^ be an estimator of Y based on a sample of data.
- Then, the bias of the estimator Y^ is given by: Bias(Y^)=E(Y^)-Y

- where E(Y^)   is the expected value of the estimator *Y*^.
- It is the measurement of the model that how well it fits the data.
- **Low Bias:** Low bias value means **fewer assumptions** are taken to build the target function. In this case, the model will closely **match the training dataset.**
- **High Bias:** High bias value means **more assumptions** are taken to build the target function. In this case, the model will **not match the training dataset closely.**

## To Reduce High Bias

- Use a more complex model
  - Polynomial Regression, CNN, RNN & Image Processing
- Increase the number of features
  - adding more features to train the dataset will increase the complexity of the model.
- Reduce Regularization of the model
  - To prevent overfitting problem and to increase generalization ability of the model
- Increase the size of the training data

- where E[Y^] is the expected value of the predicted values. Here expected value is averaged over all the training data.
- Variance errors are either low or high-variance errors
- **Low variance**: Low variance means that the model is less sensitive to changes in the training data. This is the case of **underfitting** when the model fails to generalize on both training and test data.
- **High variance**: High variance means that the model is very sensitive to changes in the training data. This is the case of **overfitting** when the model performs well on the training data but poorly on test data

| **Type** | **Meaning** | **Result** |
|---|---|---|
| **Low Variance** | Model doesn't change much with different training data | **Underfitting** — poor on both training and test data |
| **High Variance** | Model changes too much with different training data | **Overfitting** — good on training, bad on test data |

## Maximum Likelihood

- Maximum Likelihood Estimation (MLE) is a method for estimating model parameters by choosing the values that make the observed data most probable under the assumed statistical model.

- Likelihood vs Probability
- Probability: data is random, parameters fixed
- Likelihood: data is fixed, parameters vary
- $p(x \mid \theta)$ *viewed as a function of* $\theta$

- Why MLE is Central in ML
- MLE underlies:
- Linear regression
- Logistic regression
- Neural networks
- Naive Bayes
- Gaussian Mixture Models
- Hidden Markov Models
- Training a model = **maximizing likelihood**

- Likelihood for i.i.d. Data
- Assume $n$independent samples:
- $\mathcal{L}(\theta) = \prod_{i=1}^{n} p\,(x_i \mid \theta)$
- Taking logs (for numerical stability):
- $\log \mathcal{L}\,(\theta) = \sum_{i=1}^{n} \log p\,(x_i \mid \theta)$
- **MLE in Deep Learning**
- Neural networks are trained by:
- $\theta := \arg\min_{\theta} \quad \sum_{i=1}^{n} -\log p\,(y_{i'} \mid x_i \theta)$
- SGD ≈ stochastic MLE optimization

- Properties of MLE
- Advantages
- Consistent (converges to true parameters)

  Efficient (low variance asymptotically)

  Simple and general
- **Limitations**
- Overfits with small data

  Sensitive to outliers

  No uncertainty estimates

## Bayesian Statistics

- Why Bayesian Statistics in ML
- Classical (frequentist) machine learning:
- Learns **point estimates** of parameters
- Treats parameters as **fixed but unknown**
- Often gives **no uncertainty measure**
- Bayesian machine learning:
- Treats parameters as **random variables**
- Learns **full probability distributions**
- Naturally models **uncertainty, noise, and prior knowledge**

- Bayes' Theorem
- $P(\theta|D) = \frac{P(D|\theta)P(\theta)}{P(D)}$

- θ: model parameters
- D: observed data
- p(θ) : **prior**
- p(D|θ): **likelihood**
- p(θ|D): **posterior**
- p(D) : **evidence (marginal likelihood)**

- Bayesian Learning Framework
- Choose a **prior** $p(\theta)$
- Define a **likelihood** $p(D \mid \theta)$
- Compute the **posterior**
- Make predictions by **integrating over parameters**
- Bayesian Prediction
- Instead of using a single parameter value:
- $p(y^*|x^*,D)=\int p(y^*|x^*,\theta)p(\theta|D)d\theta$
- Bayesian Neural Networks
- Weights are **probability distributions**, not scalars
- $w \sim p(w)$
- **Predictive distribution**
- $p(y' \mid x, D) = \int p\,(y' \mid x.w)\,p(w \mid D)\,dw$

- Strengths of Bayesian ML
- Handles uncertainty

  Incorporates prior knowledge

  Prevents overconfidence

  Works well with small data
- Limitations
- Computational cost

  Approximation errors

  Difficult posterior design
- When to Use Bayesian Methods

•Data is scarce
•Interpretability is needed
•Decisions are high-risk
•Uncertainty matters

## Bias – variance trade-off

- In supervised learning, the class value assigned by the learning model built based on the training data may differ from the actual class value.
- This error in learning can be of two types – errors due to ‘bias’ and error due to ‘variance’.

**Errors due to ‘Bias’:**

- ***While making predictions, a difference occurs between prediction values made by the model and actual values/expected values, and this difference is known as bias errors or Errors due to bias***.

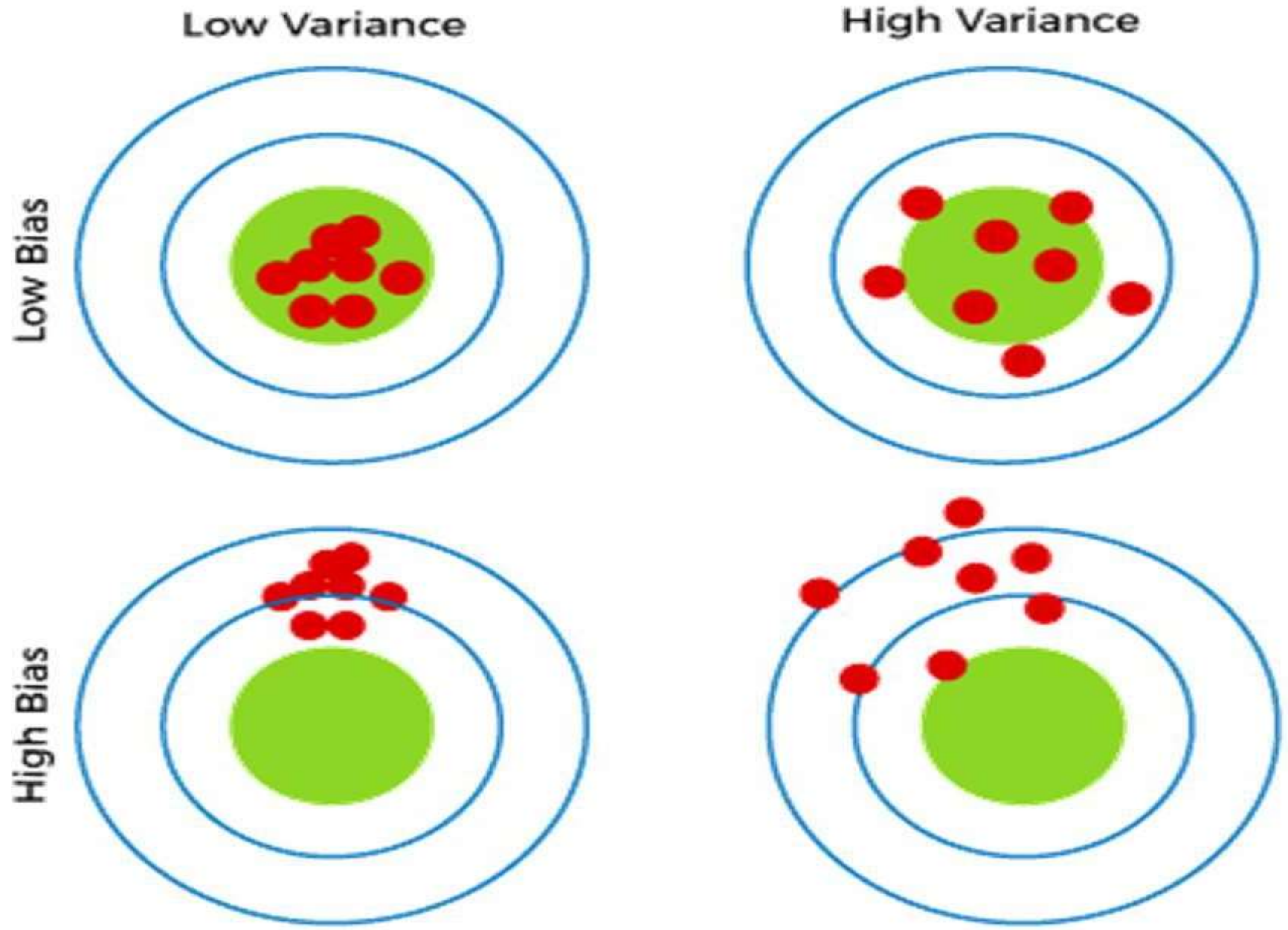
## Cont.

- Errors due to bias arise from simplifying assumptions made by the model to make the target function less complex or easier to learn.
- It is due to underfitting of the model.
- Parametric models generally have high bias
- These algorithms have a poor performance on data sets which are complex in nature
- Underfitting results in high bias.

## Errors due to ‘Variance’

- ***variance tells that how much a random variable is different from its expected value.***
- Ideally the difference in the data sets should not be significant and the model trained using different training data sets should not be too different.
- in case of overfitting, since the model closely matches the training data, even a small difference in training data gets magnified in the model.
- A high variance model leads to overfitting.
- ***It*** Increase model complexities.

# Combinations of Bias-Variance



## Combinations of Bias-Variance

1. **Low-Bias,Low-Variance:**The combination of low bias and low variance shows an ideal machine learning model. However, it is not possible practically.

2. **Low-Bias, High-Variance:** With low bias and high variance, model predictions are inconsistent and accurate on average. This case occurs when the model learns with a large number of parameters and hence leads to an **overfitting.**

## Cont.

3. **High-Bias, Low-Variance:** With High bias and low variance, predictions are consistent but inaccurate on average. This case occurs when a model does not learn well with the training dataset or uses few numbers of the parameter. It leads to **underfitting** problems in the model.

4. **High-Bias, High-Variance:** With high bias and high variance, predictions are inconsistent and also inaccurate on average.

Ways to reduce High Bias:

- Increase the input features as the model is underfitted.
- Decrease the regularization term.
- Use more complex models, such as including some polynomial features.

## Cont.

### Ways to Reduce High Variance:

- Reduce the input features or number of parameters as a model is overfitted.
- Do not use a much complex model.
- Increase the training data.
- Increase the Regularization term.

## Bias-Variance

- If the model is very simple with fewer parameters, it may have low variance and high bias.
- If the model has a large number of parameters, it will have high variance and low bias.
- It is required to make a balance between bias and variance errors, and this balance between the bias error and variance error is known as **the Bias-Variance trade-off.**

## TYPES OF MACHINE LEARNING

- Machine learning can be classified into three broad categories.

**1.Supervised learning -** Also called predictive learning. A machine predicts the class of unknown objects based on prior class related information of similar objects.

**2.Unsupervised learning** – Also called descriptive learning. A machine finds patterns in unknown objects by grouping similar objects together.

**3.Reinforcement learning** – A machine learns to act on its own to achieve the given goals.

Cont.

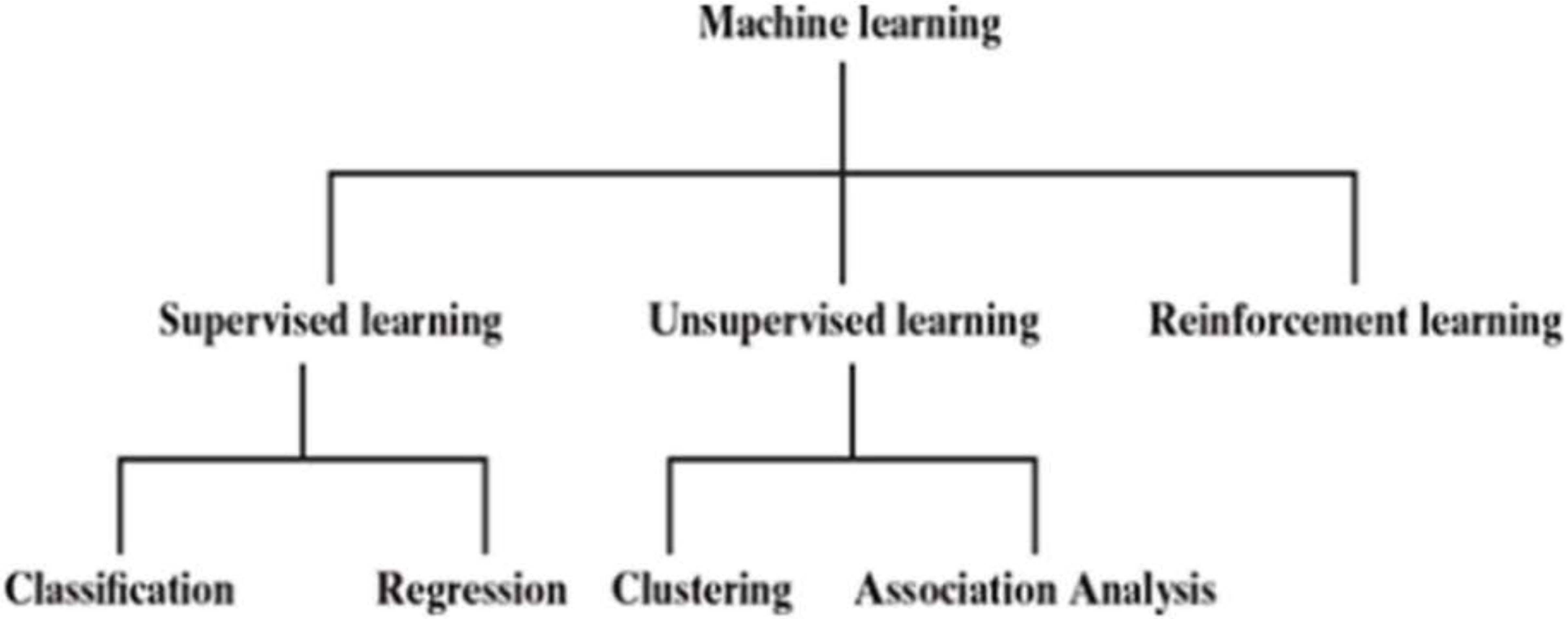


**FIG. 1.3** Types of machine learning

## Supervised learning

- Supervised learning is the machine learning task of learning a function that maps an input to an output based on example input-output pairs.
- $f(X) = Y$
- $X = \{ (X_1,Y_1), (X_2,Y_2), \ldots\ldots, (X_n,Y_n)\}$
- Where $X_1$, $X_2,\ldots,X_n$ are inputs and $Y_1,Y_2\ldots,Y_n$ are target outputs
- In supervised learning, each example in the training set is a pair consisting of an input object (typically a vector) and an output value.

## Cont.

- A supervised learning algorithm analyzes the training data and produces a function, which can be used for mapping new examples.
- In the optimal case, the function will correctly determine the class labels for unseen instances.
- Both classification and regression problems are supervised learning problems.
- A wide range of supervised learning algorithms are available, each with its strengths and weaknesses. There is no single learning algorithm that works best on all supervised learning problems.

## Cont.

- There is a set of data (the training data) that consists of a set of input data that has target data, which is the answer that the algorithm should produce, attached.
- This is usually written as a set of data ($\mathbf{x}_i$, $\mathbf{t}_i$), where the inputs are $\mathbf{x}_i$, the targets are $\mathbf{t}_i$, and the $i$ index suggests that we have lots of pieces of data, indexed by $i$ running from 1 to some upper limit $N$ .
- If we had examples of every possible piece of input data, then we could put them together into a big look-up table, and there would be no need for machine learning at all.
- The thing that makes machine learning better than that is generalization: the algorithm should produce sensible outputs for inputs that weren’t encountered during learning.

## Cont.

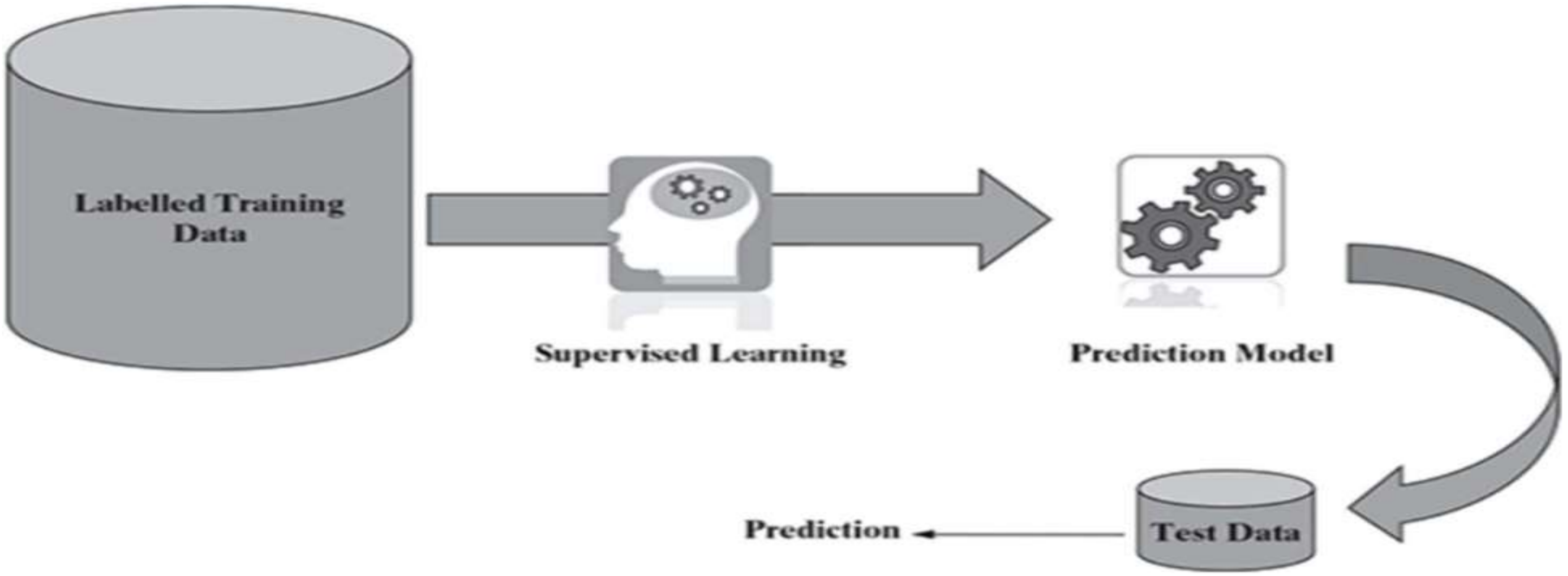
Given:

- a set of input features $X_1, \ldots, X_n$
- A target feature $Y$
- a set of training examples where the values for the input features and the target features are given for each  example
- a new example, where only the values for the input features are given

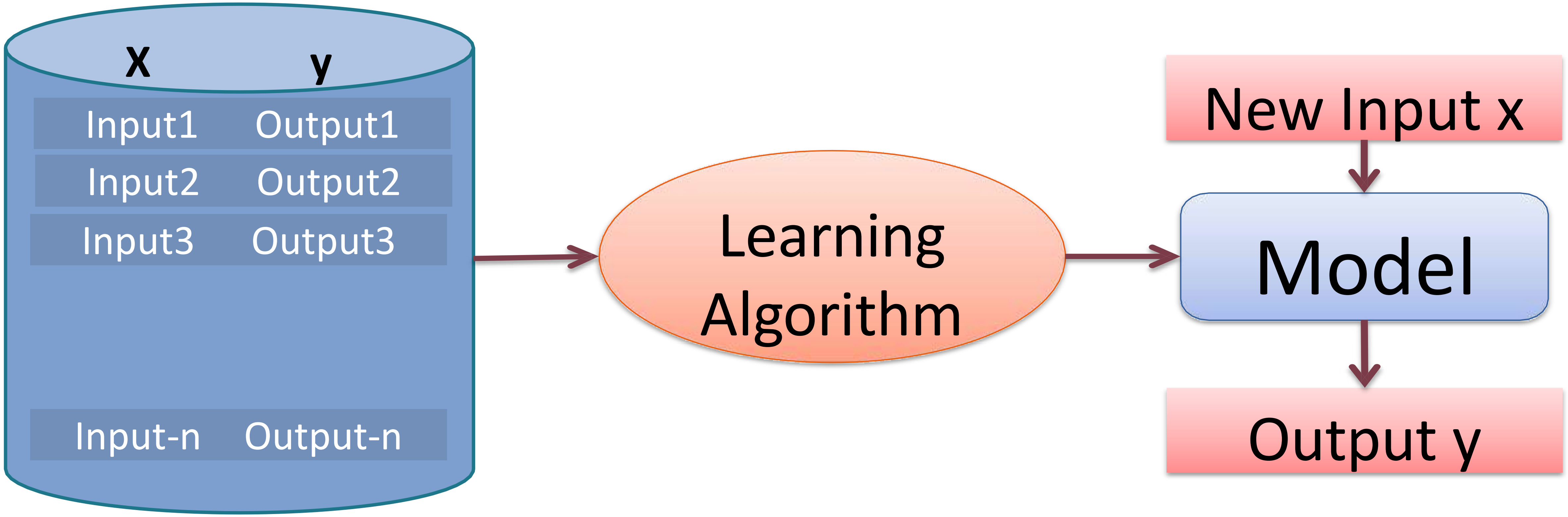
Predict the values for the target features for the new example.

- classification when Y is discrete
- regression when **Y** is continuous

Cont.

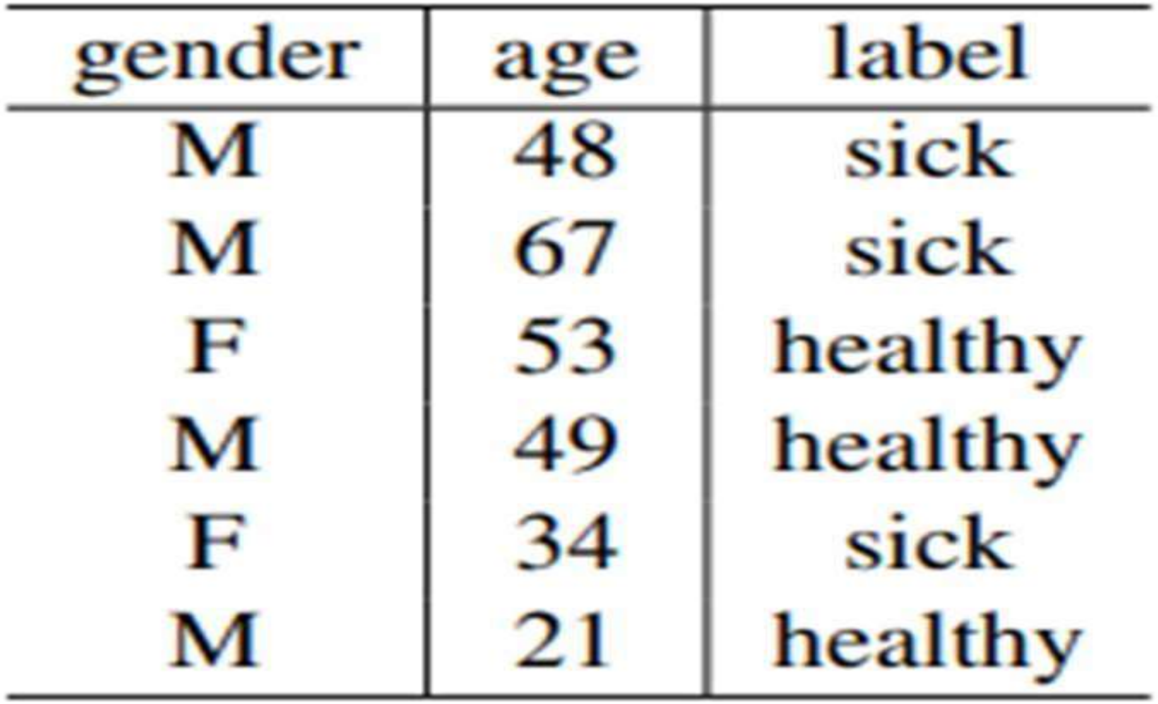

Labelled Training Data
Supervised Learning
Prediction Model
Test Data
Prediction

## Supervised Learning



## Example

Consider the following data regarding patients entering a clinic. The data consists of the gender and age of the patients and each patient is labeled as “healthy” or “sick”.



| gender | age | label |
|---|---|---|
| M | 48 | sick |
| M | 67 | sick |
| F | 53 | healthy |
| M | 49 | healthy |
| F | 34 | sick |
| M | 21 | healthy |

## Cont.

- Based on this data, when a new patient enters the clinic, how can one predict whether he/she is healthy or sick?

Classification

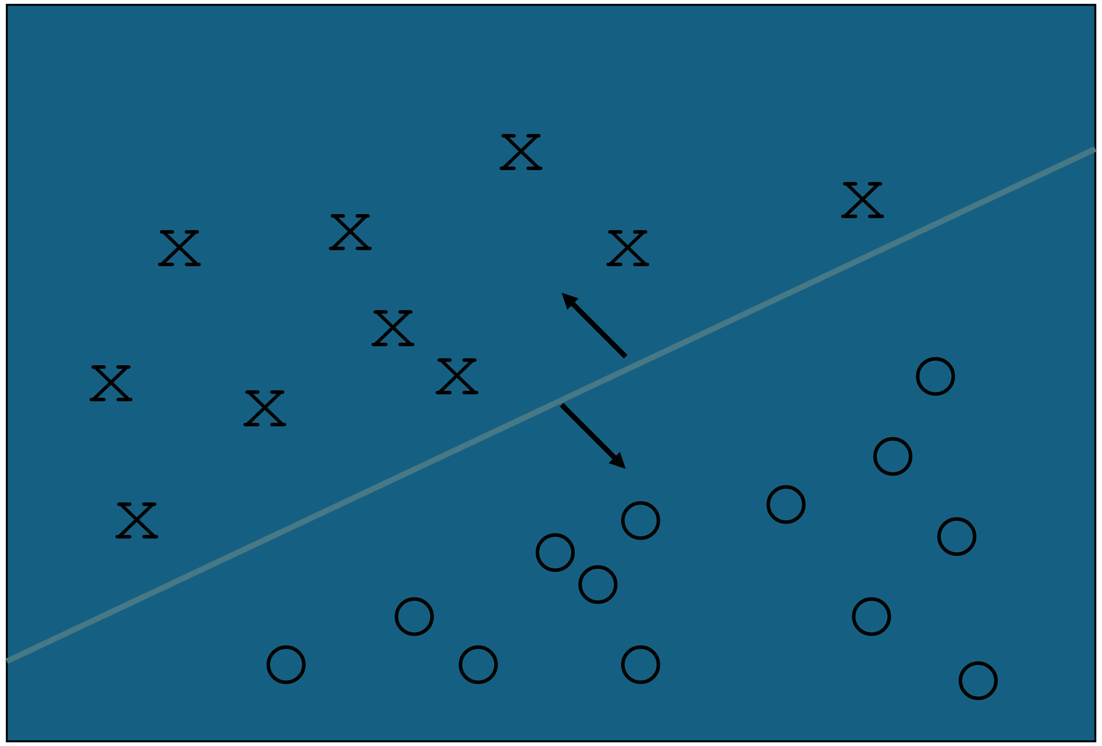
- The Classification algorithm is a Supervised Learning technique that is used to identify the category of new observations on the basis of training data.

- In Classification, a program learns from the given dataset or observations and then classifies new observation into a number of classes or groups Such as, Yes or No, 0 or 1, Spam or Not Spam, cat or dog, etc. Classes can be called as targets/labels or categories.

Cont.

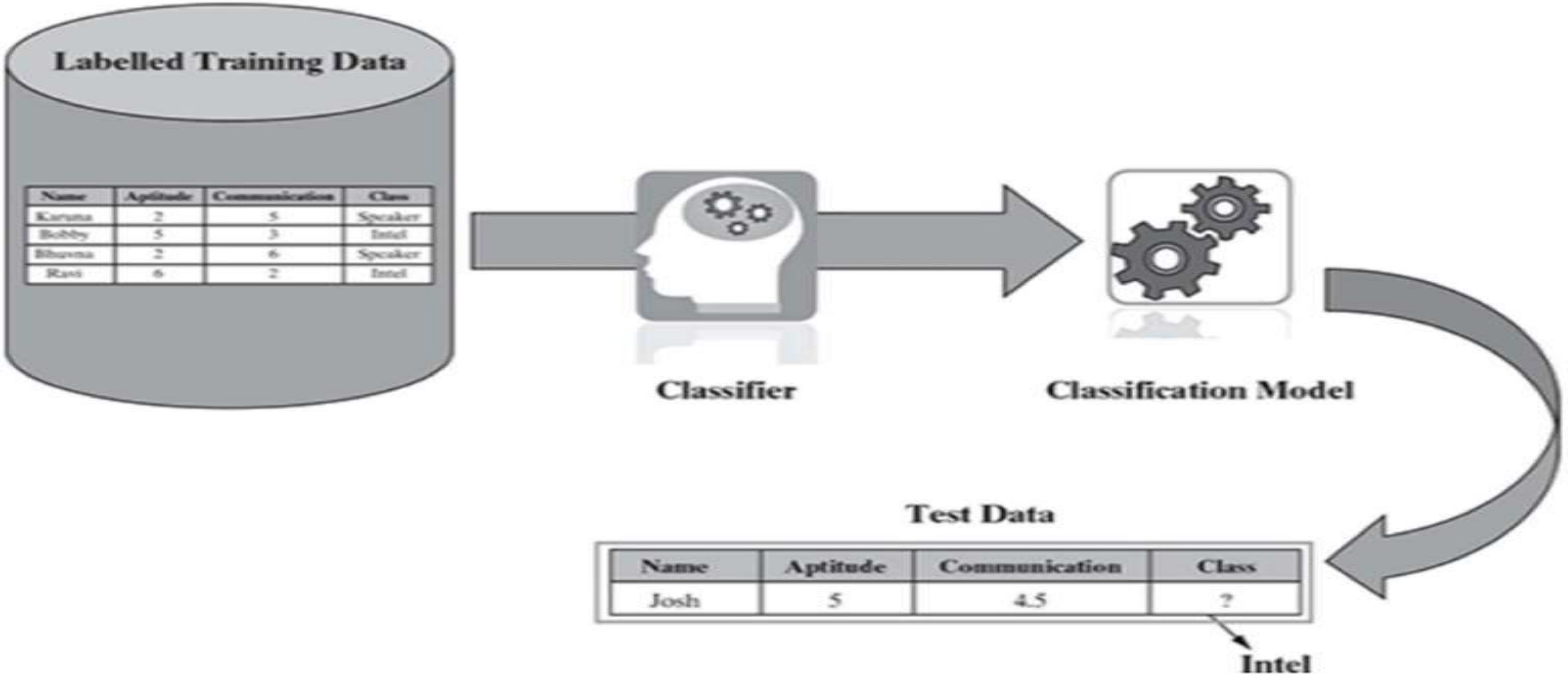
- There are mainly four types of classification tasks that one may come across, these are:
  - Binary Classification
  - Multi-Class Classification
  - Multi-Label Classification
  - Imbalanced Classification
  - **Binary Classification-** This type of classification involves separating the dataset into two categories. It means that the output variable can only take two values.
  - The task of labeling an e-mail as "spam" or "not spam." The input variable here will be the content of the e-mail that we are trying to classify. The output variable is represented by 0 for "not spam" and 1 for "spam”.
  - **Multi-Class Classification-** In multi-class classification, the output variable can have more than two possible values.
  - Example-Email received from friend, relative, unknown contact etc.,

# Linear Classification



- Binary Classification problem
- The data above the red line belongs to class 'x'
- The data below red line belongs to class 'o'
- Examples: SVM, Perceptron, Probabilistic Classifiers

## Cont.



| Name | Aptitude | Communication | Class |
|---|---|---|---|
| Karuna | 2 | 5 | Speaker |
| Bobby | 5 | 3 | Intel |
| Bhuvna | 2 | 6 | Speaker |
| Ravi | 6 | 2 | Intel |

| Name | Aptitude | Communication | Class |
|---|---|---|---|
| Josh | 5 | 4.5 | ? |

**FIG. 1.5** Classification

Cont.

- If the image is of a round object, it is put under one category, while if the image is of a triangular object, it is put under another category.
- In which category the machine should put an image of unknown category, also called a test data in machine learning parlance, depends on the information it gets from the past data, which we have called as training data.
- The whole problem revolves around assigning a label or category or class to a test data based on the label or category or class information that is imparted by the training data.

## CONT.

- There are number of popular machine learning algorithms which help in solving classification problems.
- To name a few, Naïve Bayes, Decision tree, and k-Nearest Neighbour algorithms are adopted by many machine learning practitioners.
- Some typical classification problems include: Image classification, Prediction of disease, Win– loss prediction of games, Prediction of natural calamity like earthquake, flood, etc., Recognition of handwriting.

## Unsupervised learning

- Def: **Unsupervised learning is a type of machine learning in which models are trained using unlabeled dataset and are allowed to act on that data without any supervision.**

- Unsupervised learning is a type of machine learning algorithm used to draw inferences from datasets consisting of input data without labeled responses.

- In unsupervised learning algorithms, a classification or categorization is not included in the observations.

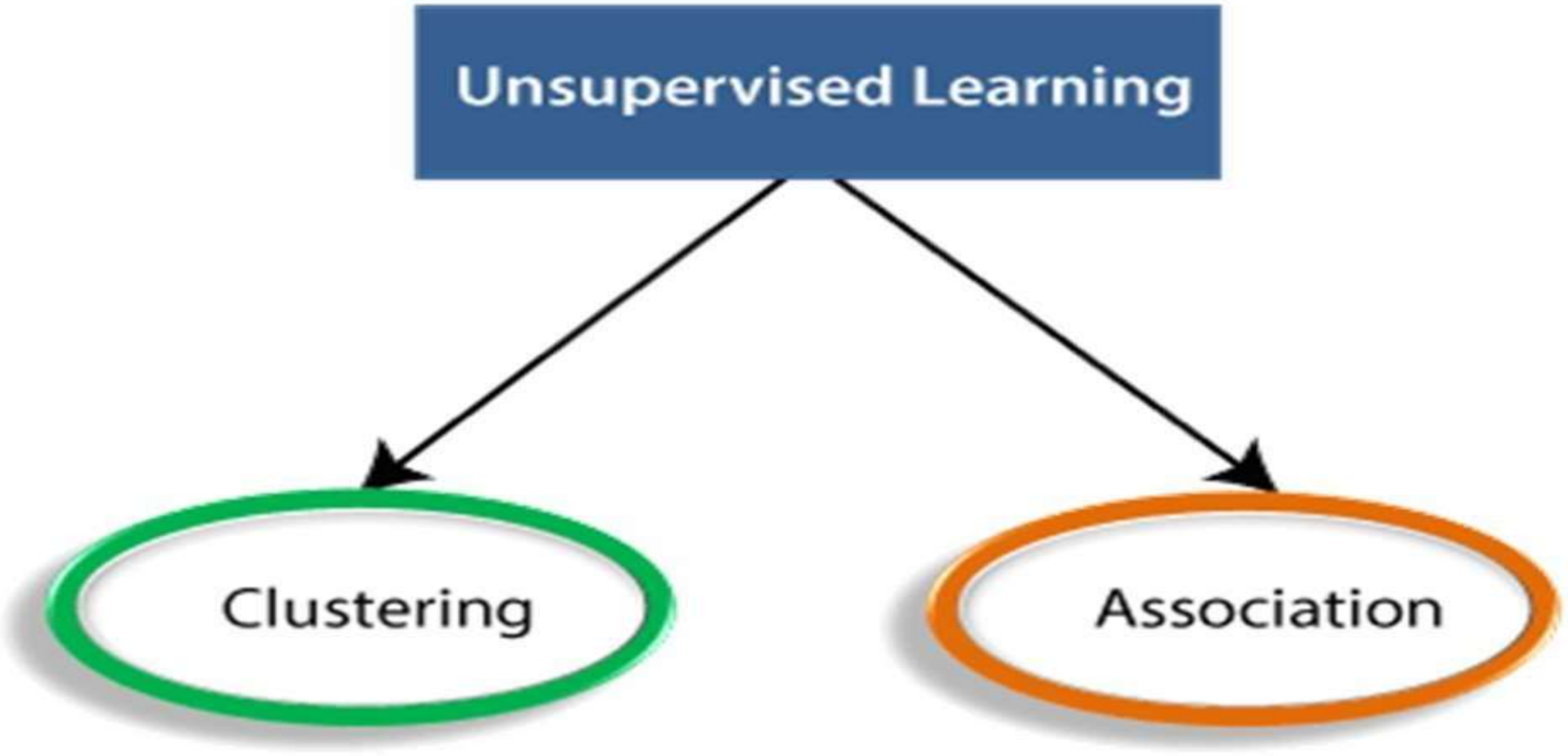
## Cont.

- There are no output values and so there is no estimation of functions. Since the examples given to the learner are unlabeled, the accuracy of the structure that is output by the algorithm cannot be evaluated.
- The most common unsupervised learning method is cluster analysis, which is used for exploratory data analysis to find hidden patterns or grouping in data.
- In unsupervised learning, the objective is to take a dataset as input and try to find natural groupings or patterns within the data elements or records.
- Therefore, unsupervised learning is often termed as descriptive model and the process of unsupervised learning is referred as pattern discovery or knowledge discovery.
- One critical application of unsupervised learning is customer segmentation.

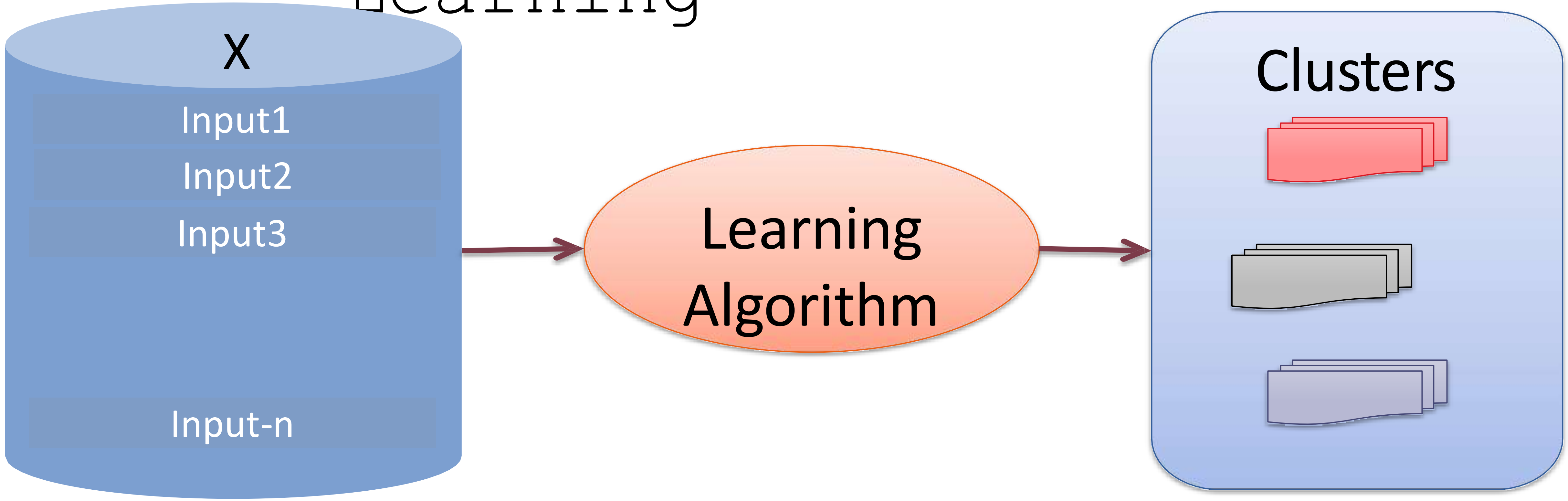
Cont
.

- **Reasons for unsupervised learning:**
  - Unsupervised learning is helpful for finding useful insights from the data.
  - Unsupervised learning is much similar as a human learns to think by their own experiences, which makes it closer to the real AI.
  - Unsupervised learning works on unlabeled and uncategorized data which make unsupervised learning more important.
  - In real-world, we do not always have input data with the corresponding output so to solve such cases, we need unsupervised learning.

## Types of Unsupervised Learning Algorithm



## Unsupervised Learning



## Clustering

- Different measures of similarity can be applied for clustering. One of the most commonly adopted similarity measure is distance.
- Two data items are considered as a part of the same cluster if the distance between them is less.
- In the same way, if the distance between the data items is high, the items do not generally belong to the same cluster. This is also known as distance-based clustering.

Cont

.

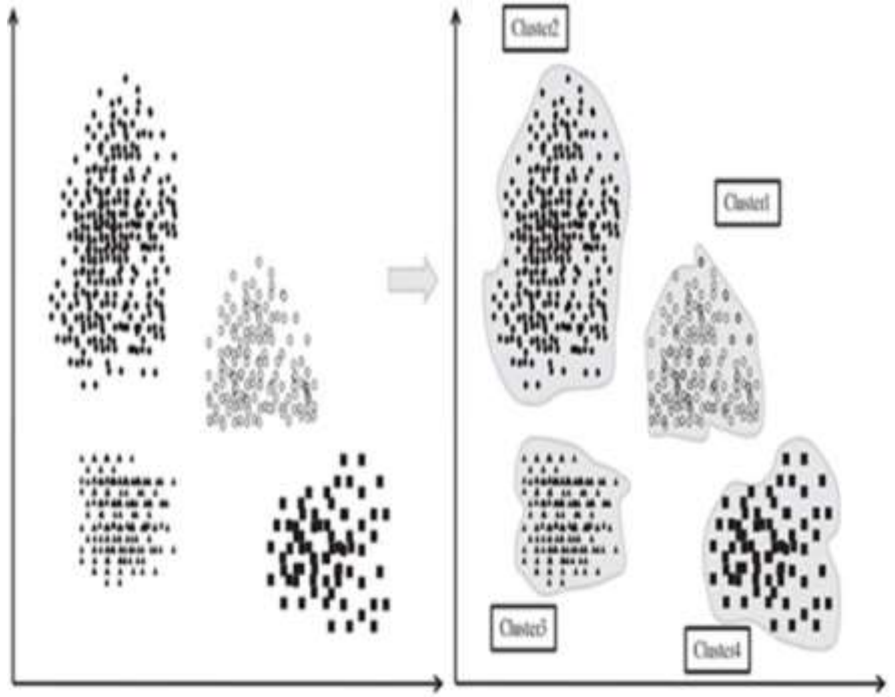


**FIG. 1.7** Distance-based clustering

cont.

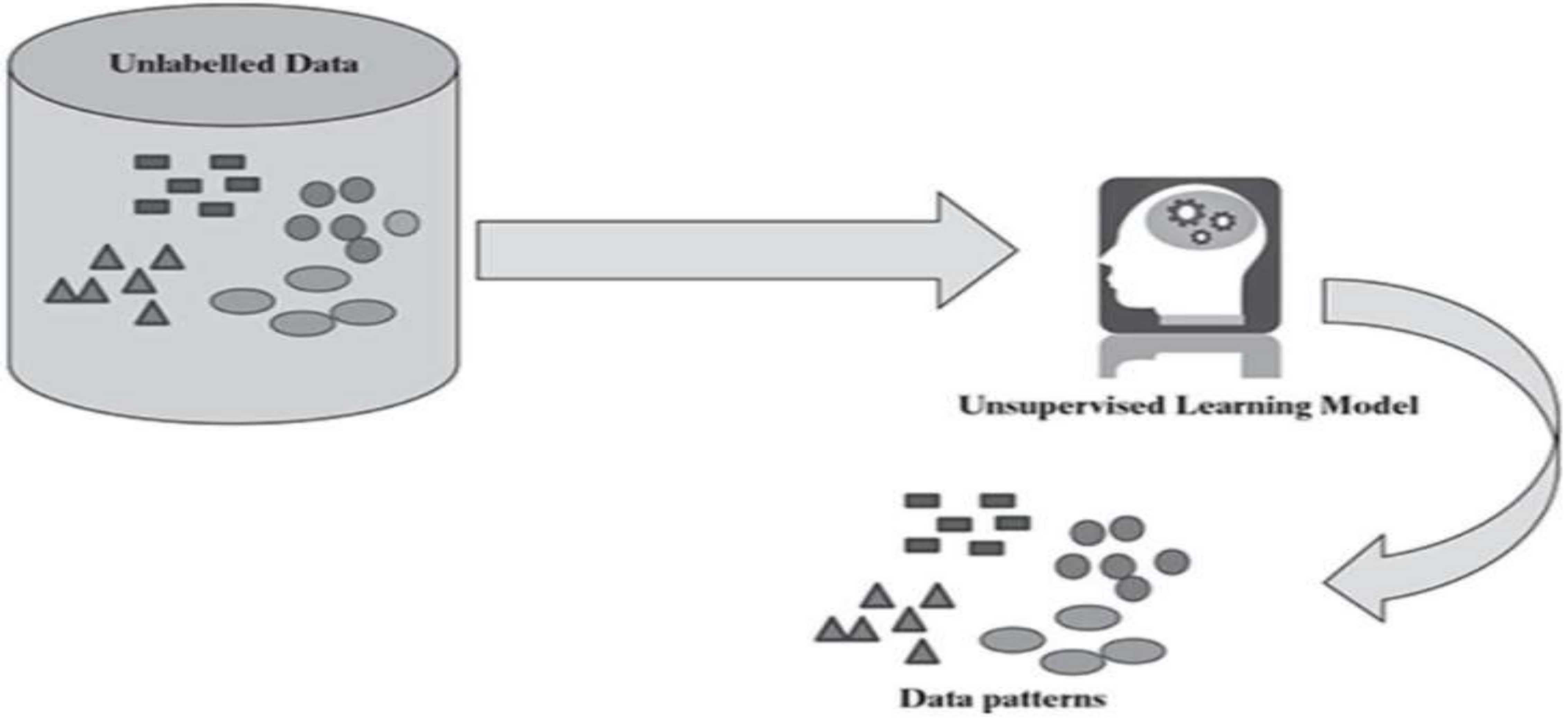


FIG. 1.8 Unsupervised learning

# Cont.

- **Unsupervised Learning algorithms:**
- K-means clustering
- Hierarchal clustering
- Anomaly detection
- Neural Networks
- Principle Component Analysis
- Independent Component Analysis
- Apriori algorithm
- Singular value decomposition

Cont.

- **Advantages**
- Unsupervised learning is used for more complex tasks as compared to supervised learning because, in unsupervised learning, we don't have labeled input data.
- Unsupervised learning is preferable as it is easy to get unlabeled data in comparison to labeled data.
- **Disadvantages**
- Unsupervised learning is intrinsically more difficult than supervised learning as it does not have corresponding output.
- The result of the unsupervised learning algorithm might be less accurate as input data is not labeled, and algorithms do not know the exact output in advance.

## Reinforcement learning

- It tries to improve its performance of doing the task.
- When a sub-task is accomplished successfully, a reward is given.
- When a sub-task is not executed correctly, obviously no reward is given.
- This continues till the machine is able to complete execution of the whole task.
- This process of learning is known as reinforcement learning.

# Cont.

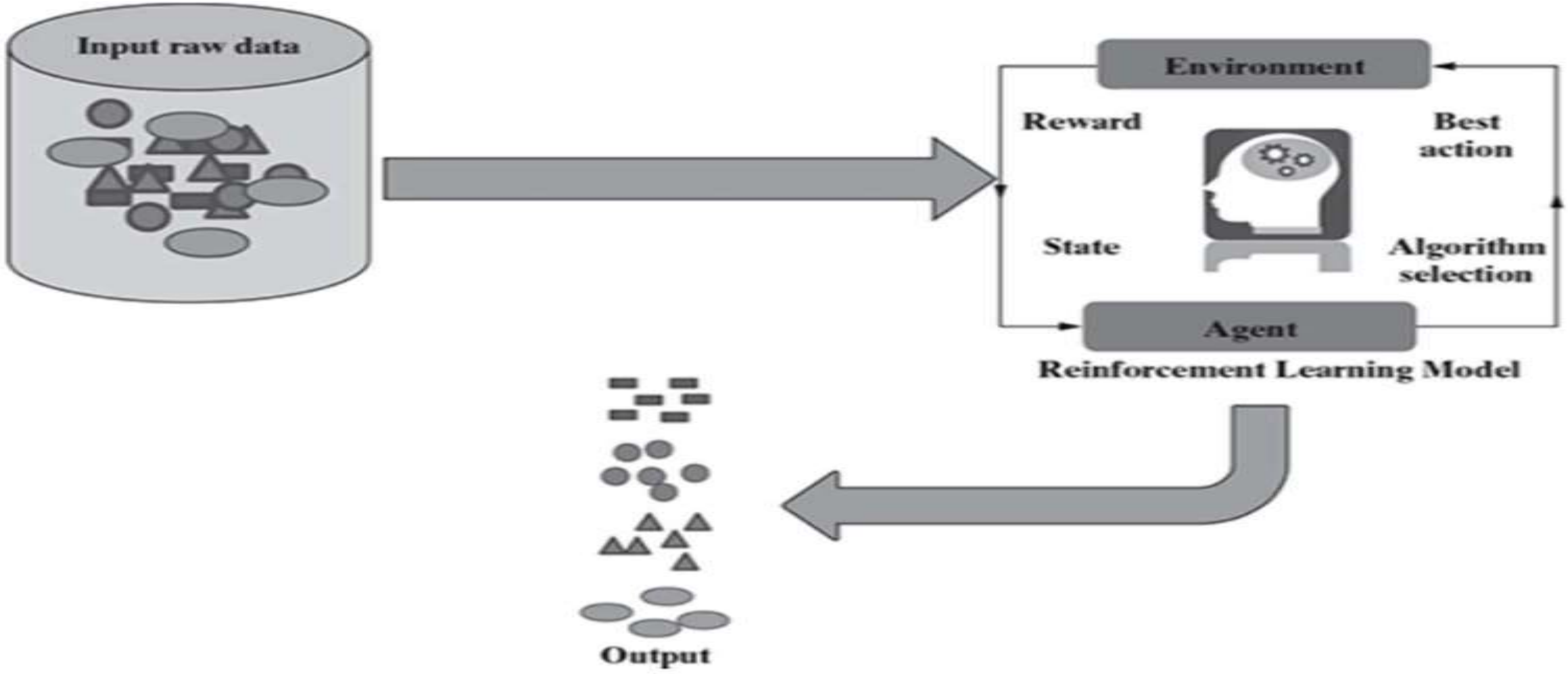
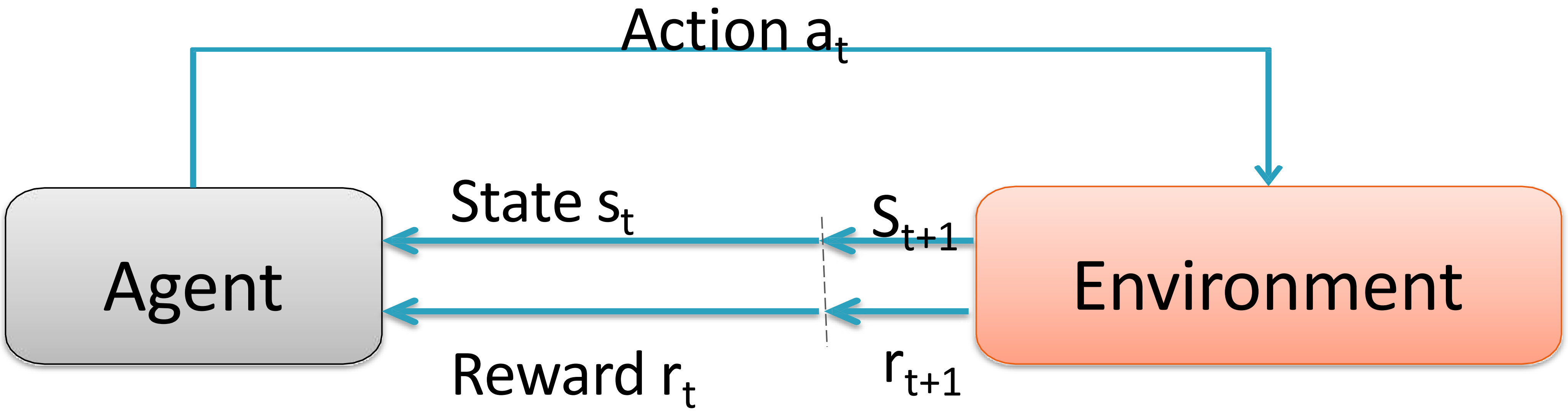
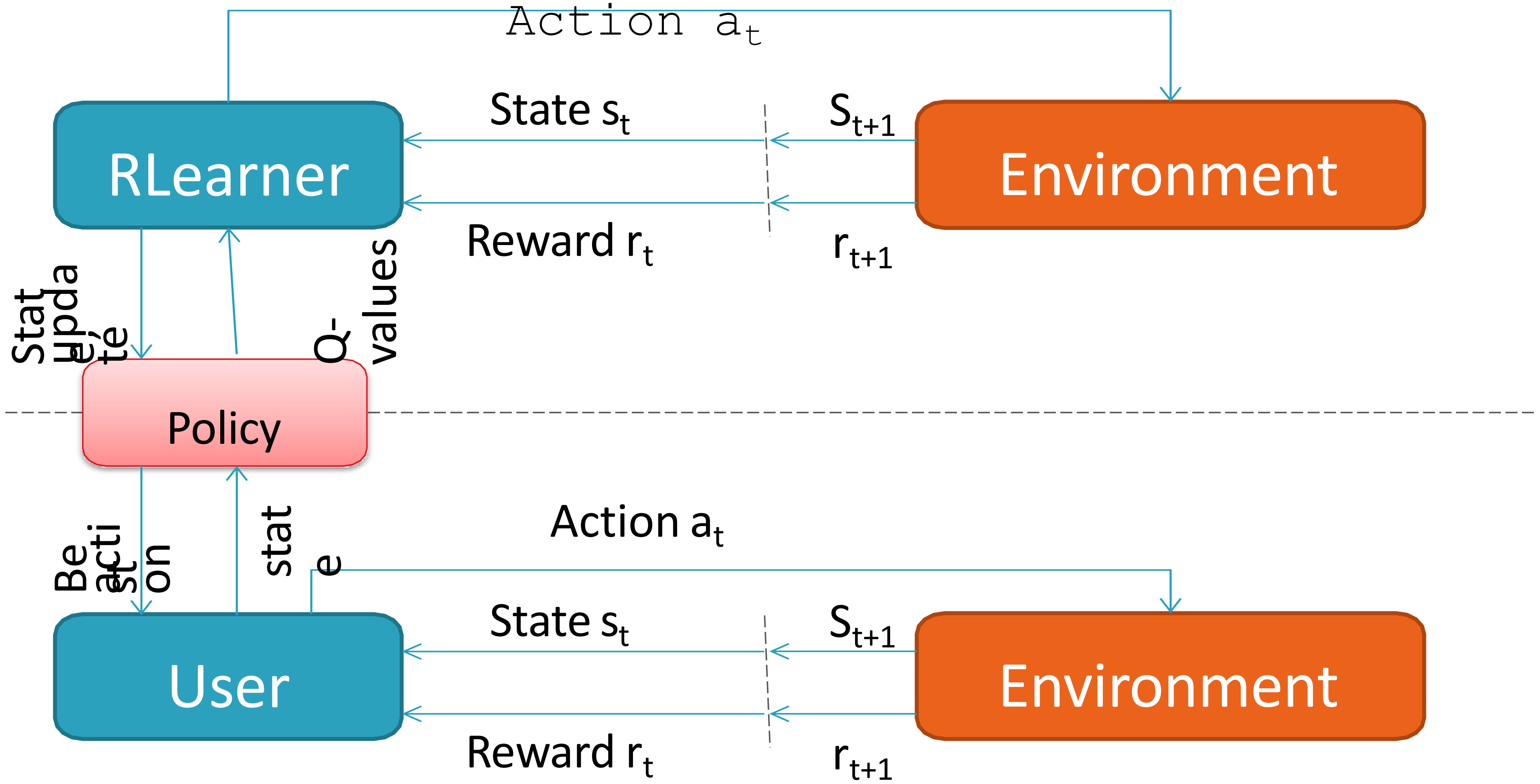


FIG. 1.10 Reinforcement learning

## Reinforcement Learning



# Reinforcement Learning



## Building a Machine Learning Algorithm

- Most ML algorithms follow a simple recipe:

  **Dataset + Model + Cost Function + Optimization Procedure**

- Example: Linear regression is one such instance.

- Understanding this recipe allows creating a wide variety of algorithms.

## Flexibility of Components

- Dataset, model, cost, and optimizer are mostly independent.
- Changing any component leads to **new algorithms**.
- This modularity supports a **wide variety of machine learning methods**.

## Cost Functions and Statistical Estimation

- Cost functions often encode **statistical objectives**.
- Most common: **negative log-likelihood** → equivalent to **maximum likelihood estimation (MLE)**.
- Additional terms may be added for regularization:
  - Example: Weight decay in linear regression
- $J_\lambda(w, b) = \|w\|_2^2 - \mathbb{E}_{x,y\sim p_{data}} \log p_{model}(y \mid x)$

## Nonlinear Models and Numerical Optimization

- Nonlinear models usually **cannot be optimized in closed form**.
- Require **iterative numerical optimization** methods:
- Gradient Descent
- Stochastic Gradient Descent
- Variants like Adam, RMSProp

## • PCA Example

- Loss function for first PCA vector:
- $J(w) = \mathbb{E}_{x \sim p_{data}} || x - r(x; w) ||_2^2$
- Model constraints: $|| w ||= 1, r(x) = (w^\top x)w$
- Optimization finds principal components without labels.
- **Cost Functions That Are Hard to Evaluate**
- Some cost functions cannot be evaluated exactly.
- Approximate gradients can still be used for **iterative optimization**.
- Allows **flexible learning even in computationally difficult scenarios**.

- **Special-Case Optimizers**
- Some algorithms require specialized optimization:
  - Decision Trees
  - k-Means
- Reason: Their cost functions have **flat regions** unsuitable for gradient-based methods.
- **Understanding ML Algorithms as a Recipe**
- Recognizing the common recipe helps:
  - See ML algorithms as a **taxonomy** of methods.
  - Understand why similar methods work for similar tasks.
  - Avoid thinking of algorithms as unrelated or ad hoc.

## Stochastic Gradient Descent

- Deep learning is powered by one very important algorithm: *stochastic gradient descent* or *SGD*.
- Stochastic gradient descent is an extension of the gradient descent algorithm.
- **Stochastic Gradient Descent (SGD)** is an optimization algorithm used to **minimize a loss function** by iteratively updating model parameters using gradients computed from **one training example (or a small batch)** at a time.
- A recurring problem in machine learning is that large training sets are necessary for good generalization, but large training sets are also more computationally expensive.
- The cost function used by a machine learning algorithm often decomposes as a sum over training examples of some per-example loss function.

- **Idea of Stochastic Gradient Descent**
- Instead of using the full dataset, SGD:
- Uses **one randomly chosen data point** (or a mini-batch)
- Makes **frequent noisy updates**
- Reaches good solutions much faster
- **SGD Update Rule**
- For a training example $(x_i, y_i)$:
- $\boxed{\theta := \theta - \eta \nabla_\theta \ell(\theta; x_i; y_i)}$

•θ: model parameters

•η: learning rate

•ℓ: loss for one data point

- Variants

| Type | Data set |
|---|---|
| Batch GD | Entire data |
| SGD | One data point |
| Mini-batch SGD | Small batch |

## Challenges Motivating Deep Learning

- The simple machine learning algorithms work very well on a wide variety of important problems.
- However, they have not succeeded in solving the central problems in AI, such as recognizing speech or recognizing objects.
- The development of deep learning was motivated in part by the failure of traditional algorithms to generalize well on such AI tasks.

## The Curse of Dimensionality

- Many machine learning problems become exceedingly difficult when the number of dimensions in the data is high.
- This phenomenon is known as the *curse of dimensionality*.
- Of particular concern is that the number of possible distinct configurations of a set of variables increases exponentially as the number of variables increases.
- The curse of dimensionality arises in many places in computer science, and especially so in machine learning

## Local Constancy and Smoothness Regularization

- The most widely used of these implicit “priors” is the *smoothness prior* or *local constancy prior*.
- This prior states that the function we learn should not change very much within a small region.
- Many simpler algorithms rely exclusively on this prior to generalize well, and as a result they fail to scale to the statistical challenges involved in solving Ailevel tasks.
- we explain why the smoothness prior alone is insufficient for these tasks.
- There are many different ways to implicitly or explicitly express a prior belief that the learned function should be smooth or locally constant.
- All of these different methods are designed to encourage the learning process to learn a function $f*$ that satisfies the condition
- $f*(x) \approx f*(x + \varepsilon))$ for most configurations $x$ and small change $\varepsilon$ .

## Manifold Learning

- An important concept underlying many ideas in machine learning is that of a manifold.
- A is a connected region. Mathematically, it is a set *manifold* of points, associated with a neighborhood around each point.
- From any given point, the manifold locally appears to be a Euclidean space. In everyday life, we experience the surface of the world as a 2-D plane, but it is in fact a spherical manifold in 3-D space.
- The definition of a neighborhood surrounding each point implies the existence of transformations that can be applied to move on the manifold from one position to a neighboring one.
- In the example of the world's surface as a manifold, one can walk north, south, east, or west.

- Although there is a formal mathematical meaning to the term “manifold,” in machine learning it tends to be used more loosely to designate a connected set of points that can be approximated well by considering only a small number of degrees of freedom, or dimensions, embedded in a higher-dimensional space.
- Each dimension corresponds to a local direction of variation.
- In the context of machine learning, we allow the dimensionality of the manifold to vary from one point to another.
- This often happens when a manifold intersects itself.
- For example, a figure eight is a manifold that has a single dimension in most places but two dimensions at the intersection at the center.

# DEEP FEED FORWARD NEURAL NTWORK

M E PALANIVEL

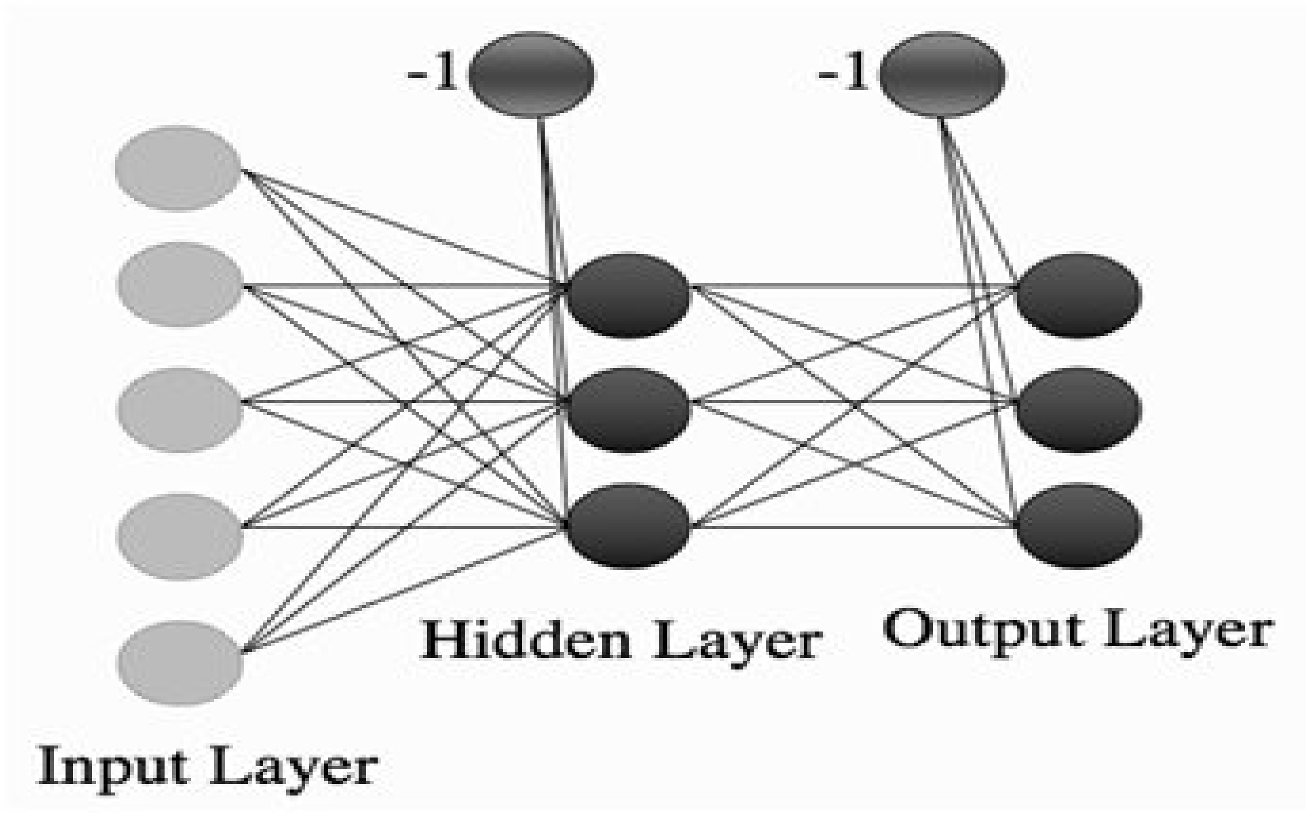
Professor

SITAMS

## Introduction

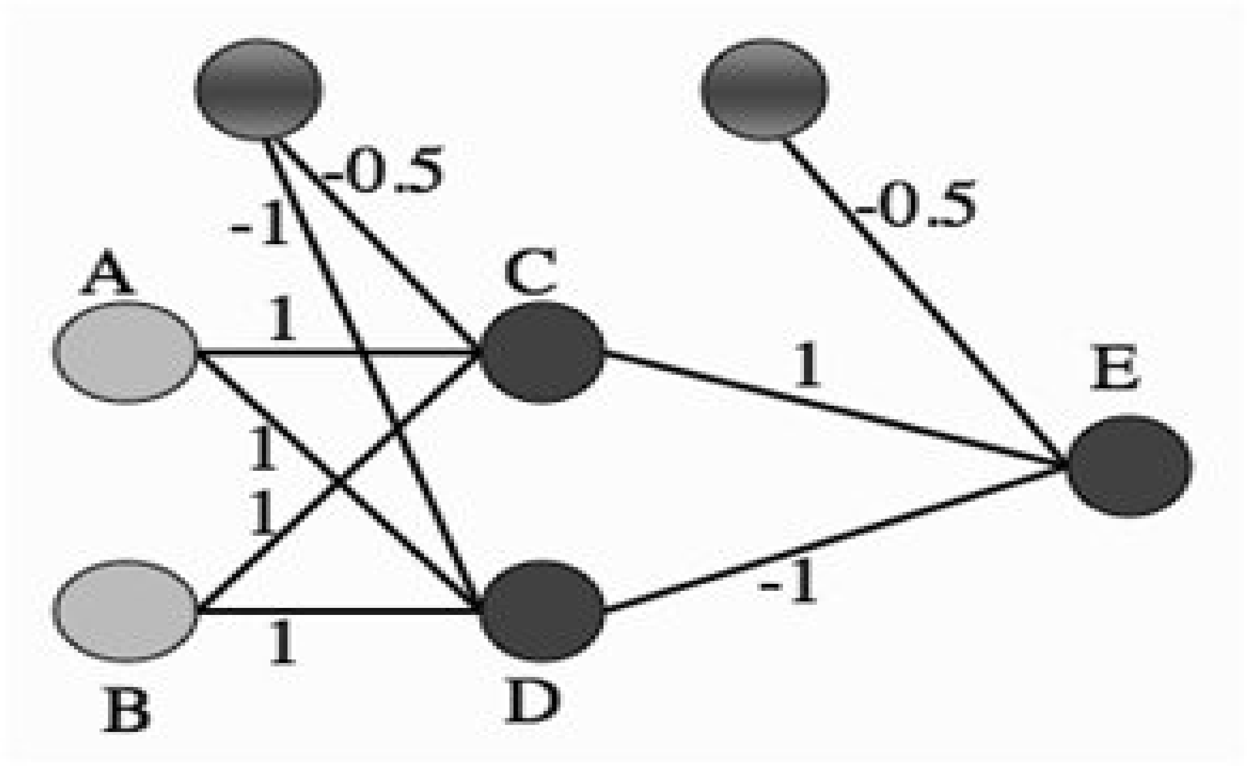
- In the last chapter we saw that while linear models are easy to understand and they can only identify straight lines, planes, or hyperplanes.
- This is not usually enough, because the majority of interesting problems are not linearly separable.
- We have pretty much decided that the learning in the neural network happens in the weights. So, to perform more computation it seems sensible to add more weights.
- There are two things that we can do: add some backwards connections, so that the output neurons connect to the inputs again, or add more neurons.
- The first approach leads into recurrent networks.

- We will instead consider the second approach.
- We can add neurons between the input nodes and the outputs, and  this will make more complex neural networks, such as the one shown in Figure

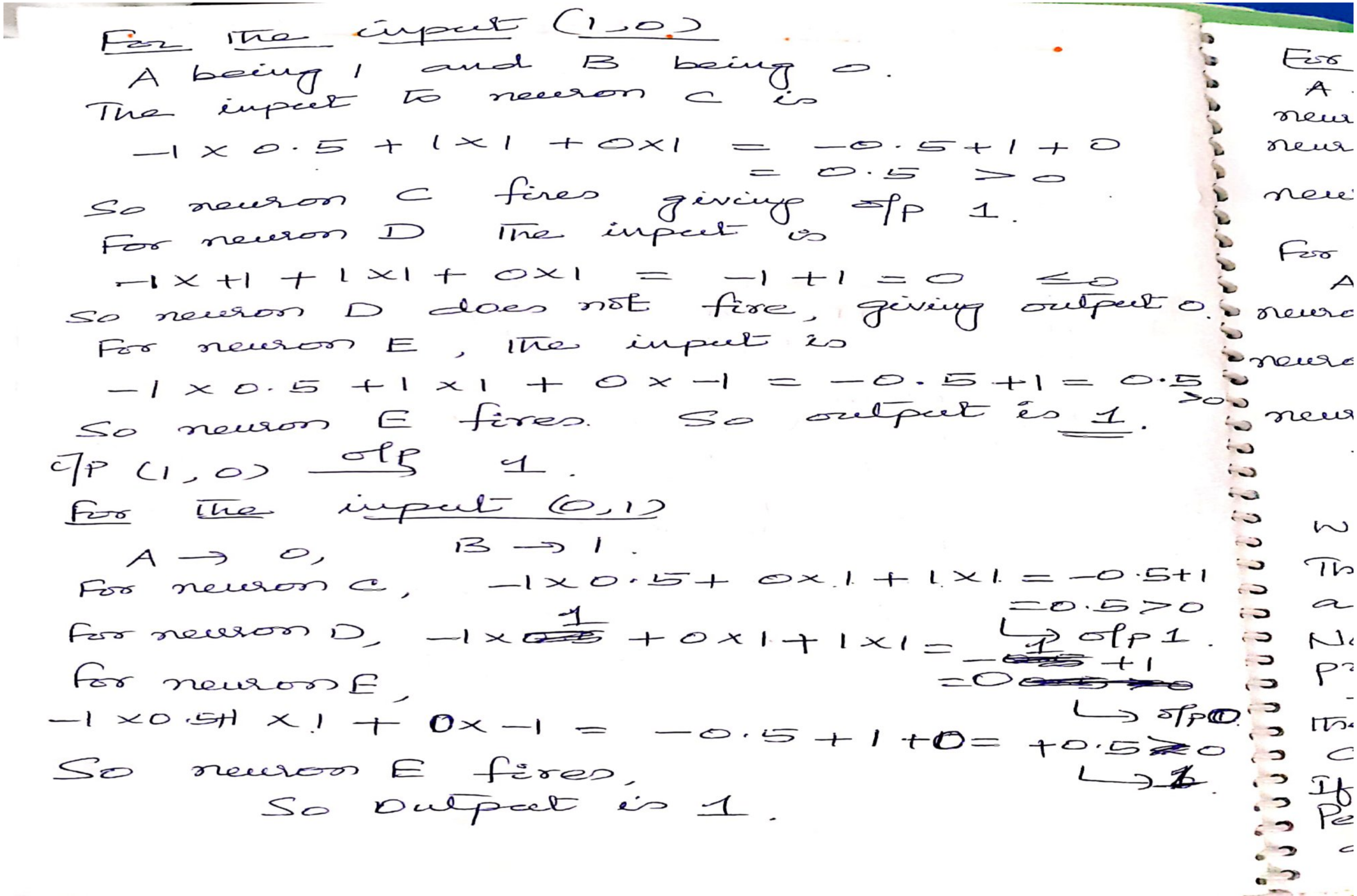


## Learning XOR Problem

- we can check that a prepared network can solve the two-dimensional XOR problem, something that we have seen is not possible for a linear model like the Perceptron.
- A suitable network is shown in Figure



•



<u>For the input (1, 0)</u>

A being 1 and B being 0.

The input to neuron C is

$-1 \times 0.5 + 1 \times 1 + 0 \times 1 = -0.5 + 1 + 0$

$= 0.5 > 0$

So neuron C fires giving o/p 1.

For neuron D the input is

$-1 \times +1 + 1 \times 1 + 0 \times 1 = -1 + 1 = 0 \le 0$

So neuron D does not fire, giving output 0.

For neuron E, the input is

$-1 \times 0.5 + 1 \times 1 + 0 \times -1 = -0.5 + 1 = 0.5 > 0$

So neuron E fires. So output is 1.

i/p (1, 0) $\xrightarrow{\text{o/p}}$ 1.

<u>For the input (0, 1)</u>

A → 0, B → 1.

For neuron C, $-1 \times 0.5 + 0 \times 1 + 1 \times 1 = -0.5 + 1$

$= 0.5 > 0$

↳ o/p 1.

For neuron D, $-1 \times \overset{1}{\cancel{0.5}} + 0 \times 1 + 1 \times 1 = -\overset{1}{\cancel{0.5}} + 1$

$= 0 \cancel{.5 > 0}$

↳ o/p 0.

For neuron E,

$-1 \times 0.5 + 1 \times 1 + 0 \times -1 = -0.5 + 1 + 0 = +0.5 > 0$

↳ 1.

So neuron E fires,

So output is 1.

For the input (0, 0).

A → 0, B → 0.

neuron C, $-1 \times 0.5 + 0 \times 1 + 0 \times 1 = -0.5 < 0$

$\hookrightarrow$ o/p → 0

neuron D, $-1 \times 1 + 0 \times 1 + 0 \times 1 = -1 < 0$

$\hookrightarrow$ o/p → 0.

neuron E, $-1 \times 0.5 + 0 \times 1 + 0 \times -1 = -0.5 < 0$

$\hookrightarrow$ o/p → $\underline{\underline{0}}$

For the input (1, 1)

A → 1, B → 1.

neuron C, $-1 \times 0.5 + 1 \times 1 + 1 \times 1 = -0.5 + 2$

$= 1.5 > 0$

$\hookrightarrow$ o/p → 1

neuron D, $-1 \times 1 + 1 \times 1 + 1 \times 1 = -1 + 1 + 1$

$= 1 > 0$

$\hookrightarrow$ o/p → 1.

neuron E,

$-1 \times 0.5 + 1 \times 1 + 1 \times -1 = -0.5 + 1 - 1$

$= -0.5 < 0$

$\hookrightarrow$ o/p → $\underline{\underline{0}}$.

Which is required o/p.

Therefore the above network can solve a problem that the perceptron cannot

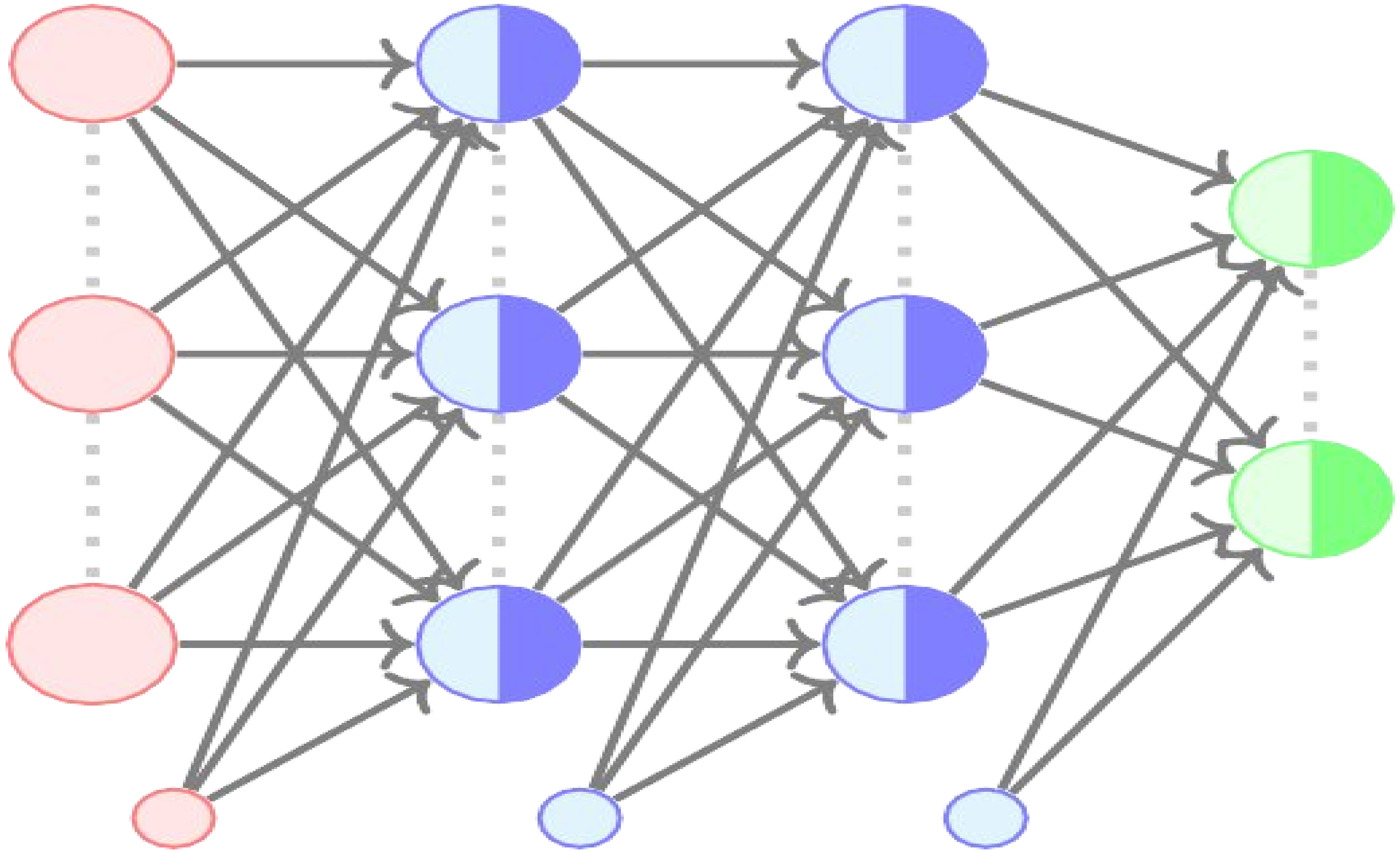
Now we have got a much more interesting problem to solve,

How can we train this network so that the weights are adapted to generate the correct answers? (target)

If we try the method that we used for the Perceptron we need to compute the <u>error</u> at the output.

## Deep Feed forward Neural Network model

- The input to the network is an **n**-dimensional vector
- The network contains **L −1** hidden layers (2, in this case) having **n** neurons each Finally, there is one output layer containing **k** neurons (say, corresponding to **k** classes)
- Each neuron in the hidden layer and output layer can be split into two parts : pre-activation and activation ($a_i$ and $h_i$ are vectors)
- The input layer can be called the 0-th layer and the output layer can be called the (L)-th layer $W_i \in R^{n\times n}$ and $b_i \in R^n$ are the weight and bias between layers i −1 and i ($0 < i < L$) $W_L \in R^{n\times k}$ and $b_L \in R^k$ are the weight and bias between the last hidden layer and



- A DFFNN Model consists of three layers. They are
- 1. Input layer
- 2. Hidden layer
- 3. Output layer
- Each layer is made of units or neurons.
- The inputs to the model corresponds to the  features measured to each training tuple.
- The input are feed simultaneously into the units making up the input layer.
- They are then weighted and fed simultaneously to a  second  layer of hidden neurons known as hidden layer.
- The output of hidden layer units can be input to another hidden layer and so on.

- The number of hidden layers is arbitrary.
- The weighted outputs of the last hidden layer are input to units making up the output layer, which emits the DFFNN MODEL prediction output.
- The units in the input layer are called input units.
- The units in the hidden layer are called hidden units.
- The units in the output layer are called output units.
- The number of hidden layers in the above model is L-1 and one output layer, therefore we call it as L layered DFFNN Model.
- The input layer is not counted because it serves  only to pass the input values to the next layer.
- The network is **feed-forward** in that none of the weights cycles back to an input unit or to an output unit of a previous layer

- The training the DFFNN Model consists of two phases:
- 1. Forward phase
- 2.  Backward phase

## Forward Phase

The pre-activation at layer *i* is given by

$$a_i(x) = b_i + W_i h_{i-1}(x)$$

The activation at layer *i* is given by

$$h_i(x) = g(a_i(x))$$

where $g$ is called the activation function example, logistic, tanh, linear, *etc.*
The activation at the output layer is given by

$$f(x) = h_L(x) = O(a_L(x))$$

where $O$ is the output activation function (for example, softmax, linear, *etc.*)

**Data:** $\{x_i, y_i\}_{i=1}^{N}$

**Model:**

$$\hat{y}_i = f(x_i) = O(W_3 g(W_2 g(W_1 x + b_1) + b_2) + b_3)$$

**Parameters:**

$\theta = W_1, .., W_L, b_1, b_2, ..., b_L \ (L = 3)$

**Algorithm:** Gradient Descent with Back-propagation (we will see soon)

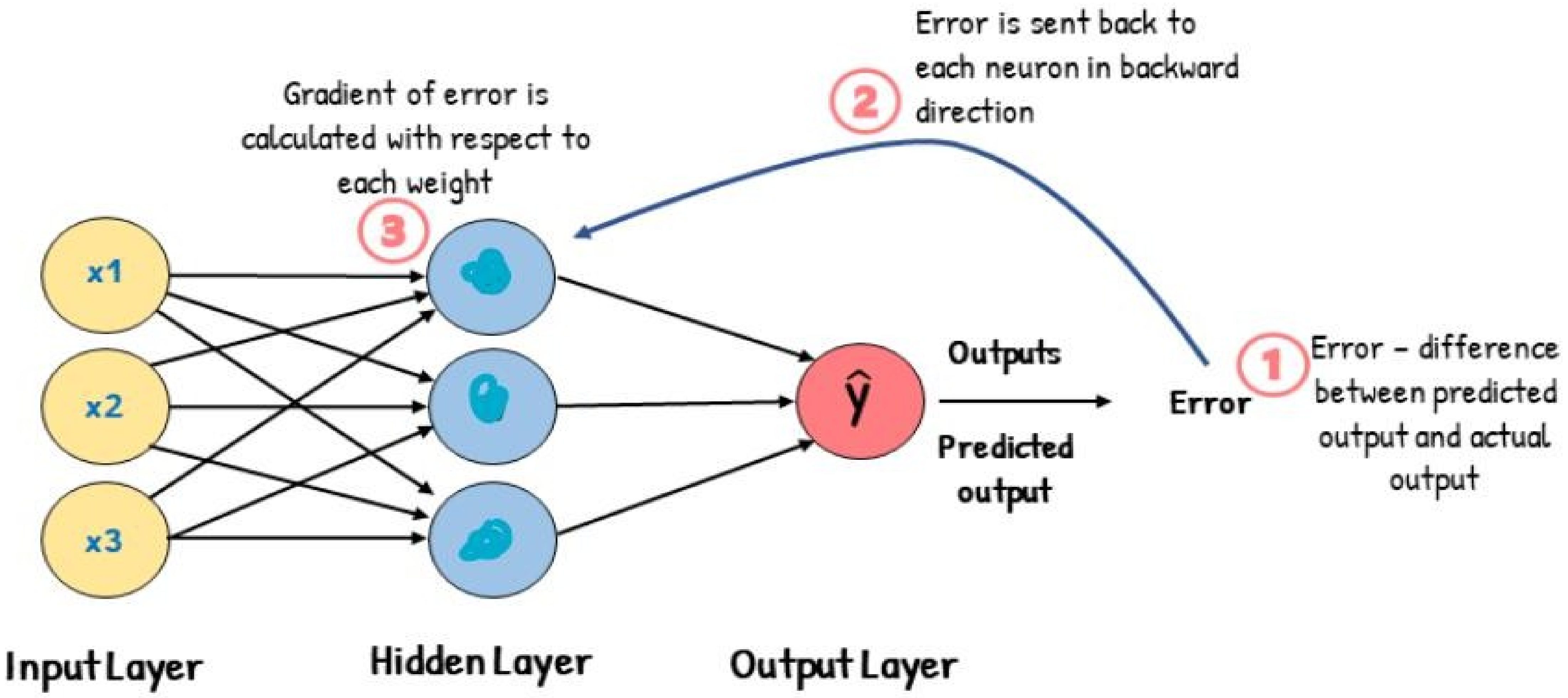
**Objective/Loss/Error function:** Say,

$$\min \frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{k} (\hat{y}_{ij} - y_{ij})^2$$

$In\ general, \min L(\theta)$

where $L(\theta)$ is some function of the parameters

# Backpropagation


Error is sent back to each neuron in backward direction
2
Gradient of error is calculated with respect to each weight
3
x1
x2
x3
ŷ
Outputs
Predicted output
Error
1
Error – difference between predicted output and actual output
Input Layer
Hidden Layer
Output Layer

## • Backward phase

- Backpropagation allows us to readjust our weights to reduce output error.
- The error is propagated backward during backpropagation from the output to the input layer.
- This error is then used to calculate the gradient of the cost function with respect to each weight.
- Essentially, backpropagation aims to calculate the negative gradient of the cost function.
- This negative gradient is what helps in adjusting of the weights.
- It gives us an idea of how we need to change the weights so that we can reduce the cost function.

❖ To find this weight, we must navigate down the cost function until we find its minimum point.

## Gradient Descent

- ***The weights are adjusted using a process called gradient descent.***
- Gradient descent is an optimization algorithm that is used to find the weights that minimize the cost function.
- Minimizing the cost function means getting to the minimum point of the cost function.
- So, gradient descent aims to find a weight corresponding to the cost function’s minimum point.

**Navigating towards minimum cost function depends on 2 things**

**Direction** - Gradient Descent or Gradient Ascent

Direction is determined by calculating the **Gradients.**

Specifically, we aim to find the negative gradient.

This is because a negative gradient indicates a decreasing slope.

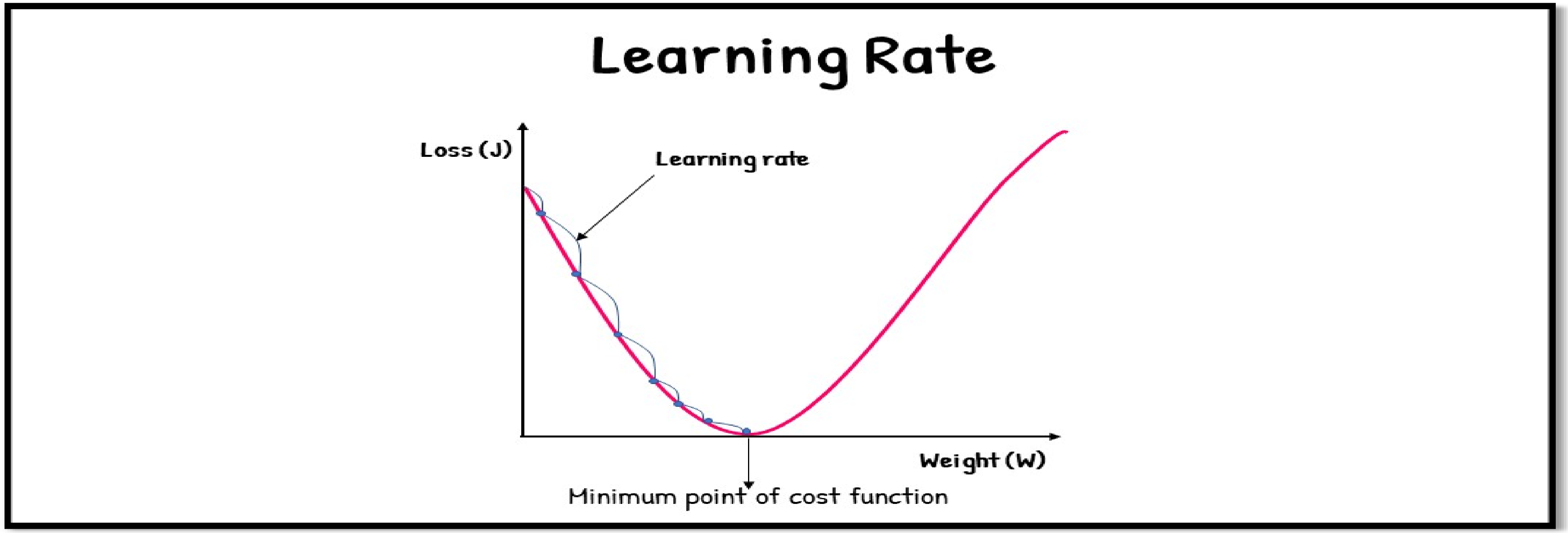
A decreasing slope means that moving downward will lead us to the minimum point.

**Step Size** – Learning Rate

Step Size is determined by **Learning Rate.**

The learning rate is a tuning parameter that determines the step size at each iteration of gradient descent.

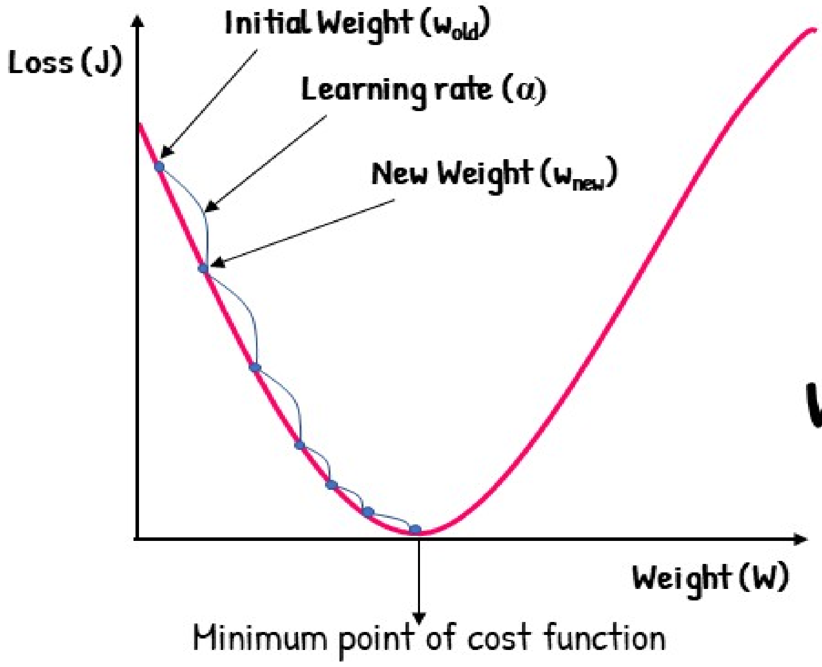
1t determines the speed at which we move down the slope.



- Let’s say you are playing a game where the players are at the top of a mountain, and they are asked to reach the lowest point of the mountain. Additionally, they are blindfolded. So, what approach do you think would make you reach the lake?
- The best way is to observe the ground and find where the land descends. From that position, take a step in the descending direction and iterate this process until we reach the lowest point.

Using the initial weight and the gradient and learning rate, we can determine the subsequent weights.

## Gradient Descent



$$w_{new} = w_{old} - \alpha \frac{\delta J}{\delta w}$$

From the graph of the cost function, we can see that:

1. To start descending the cost function, we first initialize a random weight.
2. Then, we take a step down and obtain a new weight using the gradient and learning rate. With the gradient, we can know which direction to navigate. We can know the step size for navigating the cost function using the learning rate.
3. We are then able to obtain a new weight using the gradient descent formula.
4. We repeat this process until we reach the minimum point of the cost function.
5. Once we’ve reached the minimum point, we find the weights that correspond to the minimum of the cost function.

- **Algorithm:** gradient descent()

```
$t \leftarrow 0;$
$max\ iterations \leftarrow 1000;$
$Initialize\ w_0, b_0;$
while $t{+}{+} < max\ iterations$ do
    $w_{t+1} \leftarrow w_t - \eta \nabla w_t;$
    $b_{t+1} \leftarrow b_t - \eta \nabla b_t;$
end
```

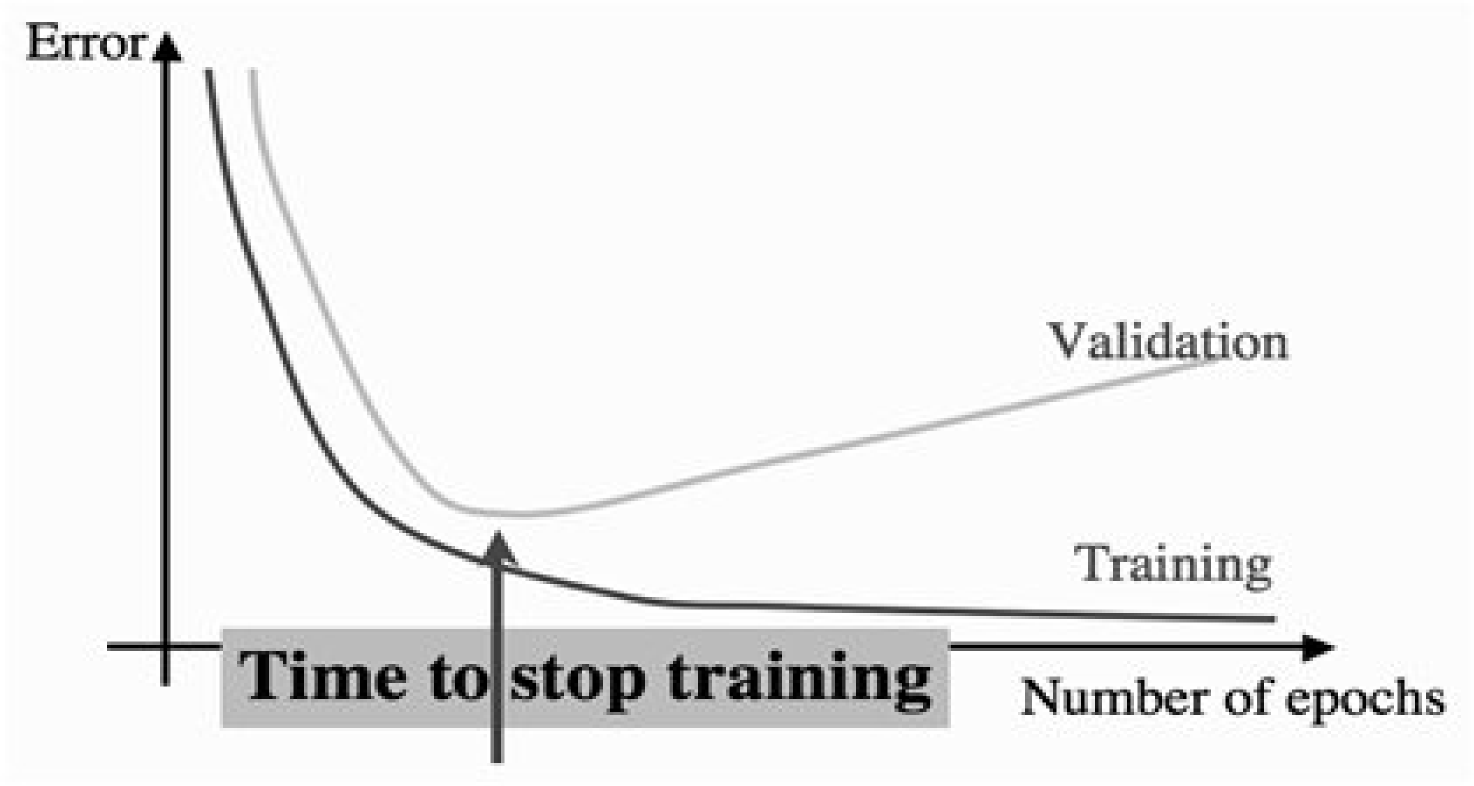
# The DFFNN Model in Practice

- We apply these ideas to using the MLP to find solutions to four different types of problem: regression, classification, time-series prediction, and data compression.
- For the MLP with one hidden layer there are $(L + 1) \text{ X } M + (M + 1)\text{X } N$ weights, where $L$, $M$, $N$ are the number of nodes in the input, hidden, and output layers, respectively.
- The extra +1s come from the bias nodes, which also have adjustable weights.
- This is a potentially huge number of adjustable parameters that we need to set during the training phase.
- the more training data there is, the better for learning, although the time that the algorithm takes to learn increases.

- Number of Hidden Layers
- There are two other considerations, which is the choice of the number of hidden nodes, and the number of hidden layers.
- We can use the back-propagation algorithm for a network with as many layers as we like, although it gets progressively harder to keep track of which weights are being updated at any given time.
- When to Stop Learning
- The training of the MLP requires that the algorithm runs over the entire dataset many times, with the weights changing as the network makes errors in each iteration.
- Setting some predefined number *N* of iterations, and running until that is reached runs the risk that the network has overfitted by then, or not learnt sufficiently, and only stopping when some predefined minimum error is reached might mean the algorithm never terminates, or that it overfits

- Using both of these options together can help, as can terminating the learning once the error stops decreasing.
- If we plot the sum-of-squares error during training, it typically reduces fairly quickly during the first few training iterations, and then the reduction slows down as the learning algorithm performs small changes to find the exact local minimum.
- We don't want to stop training until the local minimum has been found, but, as we've just discussed, keeping on training too long leads to overfitting of the network.
- This is where the validation set comes in useful. We train the network for some predetermined amount of time, and then use the validation set to estimate how well the network is generalizing. At some stage the error on the validation set will start increasing again, because the network has stopped learning about the function that generated the data, and started to learn about the noise that is in the data itself as shown in Figure .

At this stage we stop the training. This technique is called early stopping.



## Back Propagation Algorithm

- Back Propagation algorithm is a learning or Training algorithm works on Deep Feed Forward Neural Network model to solve  Classification or Prediction problems.
- Input :   a) A training data set D, consisting of Training tuples and their associated target values.

  b) the learning rate η

  c)  A Deep Feed Forward Neural Network model
- Output :    A trained Deep Feed Forward Neural Network model

## Method :

**1. Initialization :** Initialize all Weights and Biases in the DFFNN Model to small random values ( i.e [0,1] or [-1,+1])

### 2. Training phase :

#### i) Forward phase :

For each training tuple or input vector X in D , propagate the inputs in the forward

Input layer : for each input layer unit or node j

$O_j = I_j$ i.e output of an input node is its actual input value

Hidden layers : for each Hidden layer unit or node j , Compute the net input of neuron j with respect to the previous layer

$$\boxed{I_j = \sum_i w_{ij} O_i + \theta_j}$$

- Compute the activation or output of each neuron j in the hidden layers using the sigmoid activation function
- $O_j = f(I_j) = \boxed{\dfrac{1}{1 + e^{-I_j}}}$
- Output layer : For Output layer unit or node j , Compute the net input of neuron j with respect to the previous layer

$$\boxed{I_j = \sum_i w_{ij} O_i + \theta_j}$$

- Compute the activation or output of each neuron j in the Output layer using the sigmoid activation function

$$O_j = \boxed{\frac{1}{1+e^{-I_j}}}$$

- Backward Phase:
- Compute Error at each neuron j in the output layer

$$Err_j = O_j(1-O_j)\,(T_j - O_j)$$

- For each neuron j in the Hidden layer

$$Err_k = O_j(1-O_j) \sum Err_k\, w_{jk}$$

- For each weight $w_{ij}$ in the Neural Network
- The weight increment equation is

$$\Delta w_{ij} = \eta \, Err_j \, O_j$$

- The weight update equation

$$w_{ij} = w_{ij} + \Delta w_{ij}$$

- For each bias $\theta_j$ in the Neural Network
- The bias increment equation is

$$\Delta \theta_j = \eta \, Err_j$$

- The bias update equation

$$\theta_j = \theta_j + \Delta \theta_j$$

## EXAMPLES OF USING THE DFFNN Model

- **A Regression Problem**

- We will take a set of samples generated by a simple mathematical function, and try to learn the generating function (that describes how the data was made) so that we can find the values of any inputs, not just the ones we have training data for.

- We can now train an MLP on the data. There is one input value, x and one output value t, so the neural network will have one input and one output. Also, because we want the output to be the value of the function, rather than 0 or 1, we will use linear neurons at the output. We don’t know how many hidden neurons we will need yet, so we’ll have to experiment to see what works.

- Before getting started, we need to normalise the data using the method and then separate the data into training, testing, and validation sets.

## Classification with the DFFNN Model

- Using the DFFNNM for classification problems is not radically different once the output encoding has been worked out.
- The inputs are easy: they are just the values of the feature measurements (suitably normalized). There are a couple of choices for the outputs. The first is to use a single linear node for the output, $y$, and put some thresholds on the activation value of that node.
- For example, for a four-class problem
- However, this gets impractical as the number of classes gets large, and the boundaries are artificial; what about an example that is very close to a boundary, say $y = 0.5$?
- We arbitrarily guess that it belongs to class $C_3$, but the neural network doesn't give us any information about how close it was to the boundary in the output, so we don't know that this was a difficult example to classify.

Cont.

- A more suitable output encoding is called 1-of-*N* encoding. A separate node is used to represent each possible class, and the target vectors consist of zeros everywhere except for in the one element that corresponds to the correct class, e.g., (0, 0, 0, 1, 0, 0) means that the correct result is the 4th class out of 6. We are therefore using binary output values (we want each output to be either 0 or 1).

- Once the network has been trained, performing the classification is easy:

- simply choose the element $y_k$ of the output vector that is the largest element of $\mathbf{y}$ (in mathematical notation, pick the $y_k$ for which $y_k > y_j$ $j = k$; means for all, so this statement says pick the $y_k$ that is bigger than all other possible values $y_j$).

- This generates an unambiguous decision, since it is very unlikely that two output neurons will have identical largest output values.

- This is known as the hard-max activation function (since the neuron with the highest activation is chosen to fire and the rest are ignored).
- An alternative is the soft-max function, which  and which has the effect of scaling the output of each neuron according to how  large it is in comparison to the others, and making the total output sum  to 1.
- So if there is one clear winner, it will have a value near 1, while if there are several *P* values that are close to each other, they will each have a value of about $\underline{1}$ , where *p* is the number of output neurons that have similar values.

## A RECIPE FOR USING THE MLP

Here we are going to discuss about how to use the Multi-layer Perceptron with a dataset.

1. Select inputs and outputs for your problems:

* First think about the problem to solve, and make sure that you have data for the problem, both input vectors and target outputs.

* Now you need to choose what features are suitable for the problem and decide on the output encoding that you will use standard neurons (or) linear nodes.

* These things are decided by the input features and targets to solve the problem.

* Later on in the learning it can also be useful to re-evaluate the choice by training networks with some input feature missing to see if it improves the results at all.

2. Normalise Inputs:

Rescale the data by subtracting the mean value from each element of the input vector and divide by the variance.

(or) Apply any other ~~Dat~~ Normalization Techniques.

Split the data into training, testing and Validation Sets:

* You cannot test the learning ability of the network on the same data that you trained it on, Since it will generally fit That data very well.
* We generally split the data into three Sets, one for training, one for testing and Then a third set for Validation, which is testing how well the network is learning during training.
* The ratio between the Size of the three groups depends on how much data you have, but is often around 50 : 25 : 25.
* If you do not have enough data for this, use cross-Validation instead.

4. Select a network architecture:

You already know how many input nodes there will be and how many output neurons. You need to consider whether you will need a hidden layer at all and if so how many neurons it should have in it. You might want to consider more than one hidden layer.

* The more complex the network, the more data it will need to be trained on and the longer it will take.
It might also be more subject to overfitting.
* The usual method of selecting a network architecture is to try several with different numbers of hidden nodes and see which works best.

5. Train a network :
* The training of the neural network consists of applying the Multi-layer Perceptron algorithm to the training data.
* This is usually run in conjunction with early stopping, where after a few iterations of the algorithm through all of the training data, the generalization ability of the network is tested by using the validation set.
* The neural network is very likely to have far too many degrees of freedom for the problem, and so after some amount of learning it will stop modelling the generating function of the data and start to fit the noise and inaccuracies inherent in the training data.
* At this stage the error on the validation set will start to increase and learning should be stopped.

6. Test the network :

Once you have a trained network that you are happy with, it is time to use the test data for the first time.

This will enable you to see how well the network performs on some data that it has not seen before, and will tell you whether this network is likely to be usable for the other data, for which you do not have targets.

# Information Theory

M E Palanivel

## Information Theory

- Information theory is a branch of applied mathematics that revolves around quantifying how much information is present in a signal.
- It was originally invented to study sending messages from discrete alphabets over a noisy channel, such as communication via radio transmission.
- In this context, information theory tells how to design optimal codes and calculate the expected length of messages sampled from specific probability distributions using various encoding schemes.
- In the context of machine learning, we can also apply information theory to continuous variables where some of these message length interpretations do not apply.

- We would like to quantify information in a way that formalizes this intuition.
- Likely events should have low information content, and in the extreme case, events that are guaranteed to happen should have no information content whatsoever.
- Less likely events should have higher information content.
- Independent events should have additive information.
- For example, finding out that a tossed coin has come up as heads twice should convey twice as much information as finding out that a tossed coin has come up as heads once.

- In order to satisfy all three of these properties, we define the *self-information* of an event x = *x* to be

$$I(x) = -\log P(x).$$

- Our definition of $I(x)$ is therefore written in units of *nats*. One nat is the amount of information gained by observing an event of probability $1/e$ . Other texts use base-2 logarithms and units called *bits* or *shannons*; information measured in bits is just a rescaling of information measured in nats.
- When x is continuous, we use the same definition of information by analogy, but some of the properties from the discrete case are lost.
- For example, an event with unit density still has zero information, despite not being an event that is guaranteed to occur.

## Structured Probabilistic Models

- Machine learning algorithms often involve probability distributions over a very large number of random variables.
- Often, these probability distributions involve direct interactions between relatively few variables. Using a single function to describe the entire joint probability distribution can be very inefficient (both computationally and statistically).
- Instead of using a single function to represent a probability distribution, we can split a probability distribution into many factors that we multiply  together.
- For example, suppose we have three random variables: a, b and c. Suppose that a influences the value of b and b influences the value of c, but that a and c are independent given b.

- We can represent the probability distribution over all three variables as a product of probability distributions over two variables:

$$p(\mathrm{a}, \mathrm{b}, \mathrm{c}) = p(\mathrm{a})\, p(\mathrm{b} \mid \mathrm{a})\, p(\mathrm{c} \mid \mathrm{b}).$$

- These factorizations can greatly reduce the number of parameters needed to describe the distribution. Each factor uses a number of parameters that is exponential in the number of variables in the factor.

- This means that we can greatly reduce the cost of representing a distribution if we are able to find a factorization into distributions over fewer variables.

- We can describe these kinds of factorizations using graphs. Here we use the word "graph" in the sense of graph theory: a set of vertices that may be connected to each other with edges. When we represent the factorization of a probability distribution with a graph, we call it a *structured probabilistic model* or *graphical model*.

- There are two main kinds of structured probabilistic models: directed and  undirected.
- Both kinds of graphical models use a graph *G* in which each node in the graph corresponds to a random variable, and an edge connecting two random variables means that the probability distribution is able to represent direct interactions between those two random variables.
- *Directed* models use graphs with directed edges, and they represent factorizations into conditional probability distributions, as in the example above. Specifically, a directed model contains one factor for every random variable x*i* in the distribution, and that factor consists of the conditional distribution over x*i* given the parents of x*i*, denoted *P aG*(x*i* ):

$$p(\mathbf{x}) = \prod_i p\left(\mathrm{x}_i \mid Pa_{\mathcal{G}}(\mathrm{x}_i)\right).$$

- *Undirected* models use graphs with undirected edges, and they represent factorizations into a set of functions; unlike in the directed case, these functions are usually not probability distributions of any kind.
- Any set of nodes that are all connected to each other in $G$ is called a clique. Each clique $C(i)$ in an undirected model is associated with a factor $\varphi(i)(C(i)\ )$.
- These factors are just functions, not probability distributions. The output of each factor must be non-negative, but there is no constraint that the factor must sum or integrate to 1 like a probability distribution.
- The probability of a configuration of random variables is *proportional* to the product of all of these factors

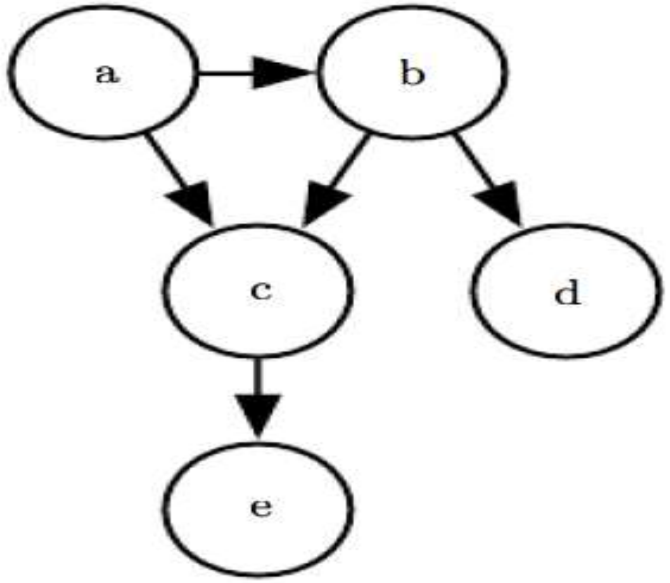


Figure :A directed graphical model over random variables a, b, c, d and e. This graph corresponds to probability distributions that can be factored as

$p(\mathrm{a}, \mathrm{b}, \mathrm{c}, \mathrm{d}, \mathrm{e}) = p(\mathrm{a})p(\mathrm{b} \mid \mathrm{a})p(\mathrm{c} \mid \mathrm{a}, \mathrm{b})p(\mathrm{d} \mid \mathrm{b})p(\mathrm{e} \mid \mathrm{c})$.

- This graph allows us to quickly see some properties of the distribution.
- For example, a and c interact directly, but a and e interact only indirectly via c.
- there is no guarantee that this product will sum to 1.
- We therefore divide by a normalizing constant *Z*, defined to be the sum or integral over all states of the product of the *φ* functions, in order to obtain a normalized probability distribution:

$$p(\mathbf{x}) = \frac{1}{Z} \prod_i \phi^{(i)} \left( \mathcal{C}^{(i)} \right).$$

See Fig.  for an example of an undirected graph and the factorization of probability distributions it represents.

- Keep in mind that these graphical representations of factorizations are a language for describing probability distributions.
- They are not mutually exclusive families of probability distributions.
- Being directed or undirected is not a property of a probability distribution; it is a property of a particular *description* of a probability distribution, but any probability distribution may be described in both ways.

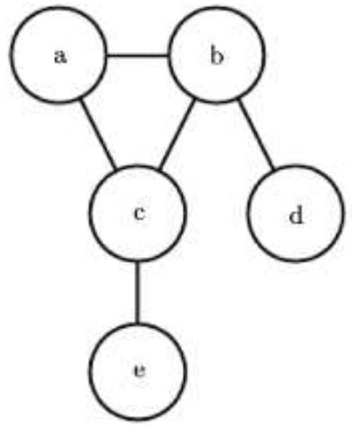


Figure : An undirected graphical model over random variables a, b, c, d and e.

- This graph corresponds to probability distributions that can be factored as
- $p$(a, b, c, d, e) = 1 $Z$ $\varphi$(1) (a, b, c)$\varphi$(2) (b, d)$\varphi$(3)(c, e).
- This graph allows us to quickly see some properties of the distribution.
- For example, a and c interact directly, but a and e interact only indirectly via c.

## Numerical Computation

- Machine learning algorithms usually require a high amount of numerical computation.
- This typically refers to algorithms that solve mathematical problems by methods that update estimates of the solution via an iterative process, rather than analytically deriving a formula providing a symbolic expression for the correct solution.
- Common operations include optimization (finding the value of an argument that minimizes or maximizes a function) and solving systems of linear equations.
- Even just evaluating a mathematical function on a digital computer can be difficult when the function involves real numbers, which cannot be represented precisely using a finite amount of memory.

## Overflow and Underflow

- The fundamental difficulty in performing continuous math on a digital computer is that we need to represent infinitely many real numbers with a finite number of bit patterns.
- This means that for almost all real numbers, we incur some approximation error when we represent the number in the computer.
- In many cases, this is just rounding error. Rounding error is problematic, especially when it compounds across many operations, and can cause algorithms that work in theory to fail in practice if they are not designed to minimize the accumulation of rounding error.
- One form of rounding error that is particularly devastating is *underflow*. Underflow occurs when numbers near zero are rounded to zero. Many functions behave qualitatively differently when their argument is zero rather than a small positive number.

- Overflow in Deep Learning
- Definition :Overflow occurs when a computed value exceeds the maximum representable floating-point number.
- For float32: max $\approx 3.4 \times 10^{38}$
- Example 1: Overflow in Exponential (Softmax)
- Problem
- $x =$[1000,1001,1002]
- Softmax(xi)=$e^{xi}$ / ∑$e^{xj}$
- e1002→ ∞
- ∞ / ∞ =NaN
- Overflow → NaN outputs

- **Stable Solution (Used in DL Libraries)**

- softmax(xi) $= \dfrac{e^{x\ i-\max(x)}}{\sum e^{xj-\max(x)}}$
- $x-\max(x)=[-2,-1,0]$
- $e^{0}=1, e^{-1}, e^{-2}$ (safe)
- Prevents overflow

- Underflow in Deep Learning
- Definition Underflow occurs when a value is too small to be represented and is rounded to zero.
- For float32: min positive≈1.18×10−38
- Example 3: Underflow in Sigmoid
- Sigmoid: $\sigma(x) = \frac{1}{1+e^{-x}}$
- If: $x = -1000$, $e^{-(-1000)} = e^{1000} \to \infty \quad \Rightarrow \sigma(x) \approx 0$
- Output becomes exact zero
- Effect on Backpropagation
- Gradient: $\sigma'(x) = \sigma(x)\big(1 - \sigma(x)\big)$
- $\sigma'(x) = 0 \times 1 = 0$
- Vanishing gradient due to underflow

- For example, we usually want to avoid division by zero or taking the logarithm of zero
- Another highly damaging form of numerical error is *overflow*. Overflow occurs when numbers with large magnitude are approximated as $\infty$ or $-\infty$. Further arithmetic will usually change these infinite values into not-a-number values.
- One example of a function that must be stabilized against underflow and
- overflow is the softmax function.
- The softmax function is often used to predict the probabilities associated with a multinoulli distribution.
- The softmax function is defined to be

$$\text{softmax}(\boldsymbol{x})_i = \frac{\exp(x_i)}{\sum_{j=1}^{n} \exp(x_j)}.$$

- Consider what happens when all of the $x_i$ are equal to some constant $c$. Analytically, we can see that all of the outputs should be equal to $\frac{1}{n}$.
- Numerically, this may not occur when $c$ has large magnitude. If $c$ is very negative, then $\exp(c)$ will underflow.
- This means the denominator of the softmax will become 0, so the final result is undefined. When $c$ is very large and positive, $\exp(c)$ will overflow, again resulting in the expression as a whole being undefined.
- Both of these difficulties can be resolved by instead evaluating $\text{softmax}(z)$ where $z = x - \max_i x_i$.
- Simple algebra shows that the value of the softmax function is not changed analytically by adding or subtracting a scalar from the input vector.
- Subtracting $\max_i x_i$ results in the largest argument to exp being 0, which rules out the possibility of overflow.

- Likewise, at least one term in the denominator has a value of 1, which rules out the possibility of underflow in the denominator leading to a division by zero.
- There is still one small problem. Underflow in the numerator can still cause the expression as a whole to evaluate to zero.
- This means that if we implement log softmax($x$) by first running the softmax subroutine then passing the result to the log function, we could erroneously obtain $-\infty$.
- Instead, we must implement  a separate function that calculates log softmax in a numerically stable way.
- The log softmax function can be stabilized using the same trick as we used to stabilize the softmax function.
- In some cases, it is possible to implement a new algorithm and have the new implementation automatically stabilized.

## Gradient Descent

- Cauchy discovered Gradient Descent motivated by the need to compute the orbit of heavenly bodies.

## Gradient-Based Optimization

- Most deep learning algorithms involve optimization of some sort. Optimization refers to the task of either minimizing or maximizing some function f (x) by altering x.
- We usually phrase most optimization problems in terms of minimizing f (x).
- Maximization may be accomplished via a minimization algorithm by minimizing −f(x).
- The function we want to minimize or maximize is called the objective function

or criterion. When we are minimizing it, we may also call it the cost function, loss function, or error function.

- We often denote the value that minimizes or maximizes a function with a superscript ∗. For example, we might say $x^* = \text{argmin } f(x)$.

- Suppose we have a function $y = f(x)$, where both $x$ and $y$ are real numbers.
- The *derivative* of this function is denoted as $f'(x)$ or as $dy / dx$ .
- The derivative $f$ ‘$(x)$ gives the slope of $f(x)$ at the point $x$.
- In other words, it specifies how to scale a small change in the input in order to obtain the corresponding change in the output: $f(x + \square) \approx f(x) + \square f \square(x)$.
- The derivative is therefore useful for minimizing a function because it tells us how to change $x$ in order to make a small improvement in $y$.
- For example, we know that $f(x - \square \operatorname{sign}(f\square(x)))$ is less than $f(x)$ for small enough $\square$. We can thus reduce $f(x)$ by moving $x$ in small steps with opposite sign of the derivative.
- This technique is called *gradient descent* (Cauchy, 1847). See Fig. 4.1 for an example of this technique.

- When $f'(x) = 0$, the derivative provides no information about which direction to move.
- Points where $f'(x) = 0$ are known as *critical points* or *stationary points*.
- A *local minimum* is a point where $f(x)$ is lower than at all neighboring points, so it is no longer possible to decrease $f(x)$ by making infinitesimal steps.
- A *local maximum* is a point where $f(x)$ is higher than at all neighboring points, so it isnot possible to increase $f(x)$ by making infinitesimal steps.
- Some critical points are neither maxima nor minima. These are known as *saddle points*.
- See Fig. 4.2 for examples of each type of critical point.

- A point that obtains the absolute lowest value of $f(x)$ is a *global minimum*.
- It is possible for there to be only one global minimum or multiple global minima of the function.
- It is also possible for there to be local minima that are not globally optimal.
- In the context of deep learning, we optimize functions that may have many local minima that are not optimal, and many saddle points surrounded by very flat regions.
- All of this makes optimization very difficult, especially when the input to the function is multidimensional.
- We therefore usually settle for finding a value of $f$ that is very low, but not necessarily minimal in any formal sense. SeeFig. 4.3 for an example.

- We often minimize functions that have multiple inputs: $f : \mathrm{R}^n \to \mathrm{R}$. For the concept of "minimization" to make sense, there must still be only one (scalar) output.
- For functions with multiple inputs, we must make use of the concept of *partial derivatives*. The partial derivative $\partial\ \partial x\ I\ f(x)$ measures how $f$ changes as only the variable $xi$ increases at point $x$.
- The *gradient* generalizes the notion of derivative to the case where the derivative is with respect to a vector: the gradient of $f$ is the vector containing all of the partial derivatives, denoted $\nabla xf\ (x)$.
- Element $i$ of the gradient is the partial derivative of $f$ with respect to $xi$.
- In multiple dimensions, critical points are points where every element of the gradient is equal to zero.

- The *directional derivative* in direction $u$ (a unit vector) is the slope of the function $f$ in direction $u$.
- In other words, the directional derivative is the derivative of the function $f(x + \alpha u)$ with respect to $\alpha$ , evaluated at $\alpha = 0$.
- Using the chain rule, we can see that $\partial\ \partial\alpha\ f(x + \alpha u) = u\square\ \nabla x f(x)$.
- To minimize $f$ , we would like to find the direction in which $f$ decreases the fastest.
- We can do this using the directional derivative:
- Min $u,u\square u$=1
- $u\square\ \nabla x f(x)$ (4.3)
- = min

- where $\theta$ is the angle between $u$ and the gradient. Substituting in $||u||2 = 1$ and ignoring factors that do not depend on $u$, this simplifies to $\text{min}u \cos \theta$.
- This is minimized when $u$ points in the opposite direction as the gradient.
- In other words, the gradient points directly uphill, and the negative gradient points directly downhill.
- We can decrease $f$ by moving in the direction of the negative gradient.
- This is known as the *method of steepest descent* or *gradient descent*.
- Steepest descent proposes a new point
- $x\square = x - \square \nabla x f(x)$ (4.5)

- where  is the *learning rate*, a positive scalar determining the size of the step.
- We  can choose  in several different ways.  A popular approach is to set  to a small constant.
- Sometimes, we can solve for the step size that makes the directional derivative vanish. Another approach is to evaluate $f\,(x - \square\, \nabla x f(x))$ for several values of $\square$ and choose the one that results in the smallest objective function value. This last strategy is called a *line search*.
- Steepest descent converges when every element of the gradient is zero (or, in practice, very close to zero).
- In some cases, we may be able to avoid running this iterative algorithm, and just jump directly to the critical point by solving the equation $\nabla x f(x) = 0$ for $x$.
- Although gradient descent is limited to optimization in continuous spaces, the   general concept of making small moves (that are approximately the best small move)   towards better configurations can be generalized to discrete spaces.
- Ascending an objective function of discrete parameters is called *hill climbing*

## Constrained Optimization

- Sometimes we wish not only to maximize or minimize a function $f(x)$ over all possible values of $x$.
- Instead we may wish to find the maximal or minimal value of $f(x)$ for values of $x$ in some set S. This is known as *constrained optimization*.
- Points $x$ that lie within the set S are called *feasible* points in constrained optimization terminology
- One simple approach to constrained optimization is simply to modify gradient descent taking the constraint into account
- If we use a small constant step size

- we can make gradient descent steps, then project the result back into S. If we use a line search, we can search only over step sizes □ that yield new *x* points that are feasible, or we can project each point on the line back into the constraint region.
- When possible, this method can be made more efficient by projecting the gradient into the tangent space of the feasible region before taking the step or beginning the line search.
- A more sophisticated approach is to design a different, unconstrained optimization problem whose solution can be converted into a solution to the original, constrained optimization problem.

- For example, if we want to minimize $f(x)$ for $x \in$ R2 with $x$ constrained to have exactly unit $L2$ norm, we can instead minimize $g(\theta) = f([\cos\theta, \sin\theta]$□) with respect to $\theta$, then return $[\cos\theta, \sin\theta]$ as the solution to the original problem.
- This approach requires creativity; the transformation between optimization problems must be designed specifically for each case we encounter.

- The Karush–Kuhn–Tucker (KKT) approach1 provides a very general solution to constrained optimization. With the KKT approach, we introduce a new function called the generalized Lagrangian or generalized Lagrange function.
- To define the Lagrangian, we first need to describe S in terms of equations and inequalities. We want a description of S in terms of m functions g(i) and n functions h(j) so that S= $\{x \mid \forall i, g^{(i)}(x) = 0 \text{ and } \forall j, h^{(j)}(x) \leq 0\}$.
- The equations involving g(i) are called the equality constraints and the inequalities involving h(j) are called inequality constraints.

- We introduce new variables $\lambda i$ and $\alpha j$ for each constraint, these are called the KKT multipliers.
- The generalized Lagrangian is then defined as $L(x, \lambda,\alpha) = f(x) + I\ \lambda ig(i)(x) + j\ \alpha jh(j)(x)$.
- We can now solve a constrained minimization problem using unconstrained optimization of the generalized Lagrangian.
- Observe that, so long as at least one feasible point exists and $f(x)$ is not permitted to have value $\infty$, then

- Min $x$
- Max $\lambda$
- Max $\alpha,\alpha \geq 0$ $L(x, \lambda, \alpha)$. (
- has the same optimal objective function value and set of optimal points $x$ as
- Min $x \in S$  $f(x)$. This follows because any time the constraints are satisfied,
- Max $\lambda$
- Max $\alpha,\alpha \geq 0$
- $L(x, \lambda, \alpha) = f(x)$, while any time a constraint is violated, Max $\lambda$
- Max $\alpha,\alpha \geq 0$
- $L(x, \lambda, \alpha) = \infty$.

- To perform constrained maximization, we can construct the generalized Lagrange function of , which leads to this $-f(x)$ optimization problem:
- Min $x$
- Max $\lambda$
- Max $\alpha,\alpha\geq 0$ $-f(x)$ + □ $i$
- $\lambda ig(i)(x)$ + $j$
- $\alpha jh(j)(x)$. (4.19)
- We may also convert this to a problem with maximization in the outer loop:
- Max $x$
- Min $\lambda$
- Min $\alpha,\alpha\geq 0$
- $f(x)$ + $i$

- The sign of the term for the equality constraints does not matter; we may define it with addition or subtraction as we wish, because the optimization is free to choose any sign for each $\lambda i$.
- The inequality constraints are particularly interesting. We say that a constraint $h(i)(x)$ is *active* if $h(i)(x*) = 0$.
- If a constraint is not active, then the solution to the problem found using that constraint would remain at least a local solution if that constraint were removed.
- It is possible that an inactive constraint excludes other solutions.

- For example, a convex problem with an entire region of globally optimal points (a wide, flat, region of equal cost) could have a subset of this region eliminated by constraints, or a non-convex problem could have better local stationary points excluded by a constraint that is inactive at convergence.
- However, the point found at convergence remains a stationary point whether or not the inactive constraints are included. Because an inactive $h(i)$ has negative value, then the solution to $\min_x \max_\lambda \max_{\alpha,\alpha \geq 0} L(x, \lambda, \alpha)$ will have $\alpha_i = 0$.
- We can thus least one of the constraints $\alpha_i \geq 0$ and $h(i)(x) \leq 0$ must be active at the solution.

- To gain some intuition for this idea, we can say that either the solution is on the boundary imposed by the inequality and we must use its KKT multiplier to influence the solution to $x$, or the inequality has no influence on the solution and we represent this by zeroing out its KKT multiplier.
- The properties that the gradient of the generalized Lagrangian is zero, all constraints on both $x$ and the KKT multipliers are satisfied, and $\alpha \,\square\, h(x) = 0$ are called the Karush-Kuhn-Tucker (KKT) conditions
- Together, these properties describe the optimal points of constrained optimization problems.
- For more information about the KKT approach, see Nocedal and Wright

# Linear Least Squares

- Suppose we want to find the value of $x$ that minimizes
- $f(x) = 1\ 2\ ||-||\ Ax\ b\ 22$
- There are specialized linear algebra algorithms that can solve this problem efficiently.
- However, we can also explore how to solve it using gradient-based optimization as a simple example of how these techniques work.
- First, we need to obtain the gradient:

$$\nabla x f(x) = A□\ (Ax - b) = A□Ax - A□b.$$

- We can then follow this gradient downhill, taking small steps. See Algorithm 4.1 for details.

- Algorithm 4.1 An algorithm to minimize $f(x) = 1$
- $2 \, ||Ax - b||22$
- with respect to $x$
- using gradient descent.
- Set the step size (□) and tolerance ($\delta$) to small, positive numbers.
- while $||A□Ax - A□b||2 > \delta$ do
- $x \leftarrow x -$ □
- □
- $A□Ax - A□b$
- □
- end whil

- One can also solve this problem using Newton’s method. In this case, because

  the true function is quadratic, the quadratic approximation employed by Newton’s

  method is exact, and the algorithm converges to the global minimum in a single

  step.
- Now suppose we wish to minimize the same function, but subject to the constraint $x$□$x \leq 1$. To do so, we introduce the Lagrangian $L(x, \lambda) = f(x) + \lambda$
- $x$□ $x - 1$
- We can now solve the problem
- Min $x$
- Max $\lambda, \lambda \geq 0$

- The smallest-norm solution to the unconstrained least squares problem may be found using the Moore-Penrose pseudoinverse: $x = A^+ b$.
- If this point is feasible, then it is the solution to the constrained problem. Otherwise, we must find asolution where the constraint is active.
- By differentiating the Lagrangian with respect to $x$, we obtain the equation

$$A^\top Ax - A^\top b + 2\lambda x = 0.$$

- This tells us that the solution will take the form

$$x = (A^\top A + 2\lambda I)^{-1} A^\top b.$$

- The magnitude of $\lambda$ must be chosen such that the result obeys the constraint. We can find this value by performing gradient ascent on $\lambda$. To do so, observe
- $\partial \ \partial\lambda$
- $L(x, \lambda) = x \square x - 1$. When the norm of $x$ exceeds 1, this derivative is positive, so to follow the derivative uphill and increase the Lagrangian with respect to $\lambda$, we increase $\lambda$ .
- Because the coefficient on the $x \square x$ penalty has increased, solving the linear equation for $x$ will now yield a solution with smaller norm.
- The process of solving the linear equation and adjusting $\lambda$ continues until $x$ has the correct norm and the derivative on $\lambda$ is 0.