UNIT-I

Introduction: Learning, Types of Machine Learning, Supervised Learning, The Brain and the Neuron, Design a Learning System, Perspectives and Issues in Machine Learning, Concept Learning Task, Concept Learning as Search, Finding a Maximally Specific Hypothesis, Version Spaces and the Candidate Elimination Algorithm, Linear Discriminates, Perceptron, Linear Separability, Linear Regression

Learning:

- Getting better at some task through practice is called Learning

Machine Learning:

- Machine Learning is the study of computer algorithms that allow computer programs to automatically improve through experience.
- Machine Learning is about making computers modify or adapt their actions so that these actions get more accurate.
- Machine learning is a subset of AI, which enables the machine to automatically learn from data, improve performance from past experiences, and make predictions.

Types of Machine Learning:

Supervised Learning:

- A training set of examples with correct responses is provided and based on this, the algorithm generalises to respond correctly to all possible inputs is called supervised learning.
- Works on labelled data.

Unsupervised Learning:

- Correct answers are not provided, but instead the algorithm tries to identify similarities between the inputs so that inputs that have something in common are categorised together.
- Works on unlabelled data.

Reinforcement Learning:

- Reinforcement learning works on a feedback-based process, in which an AI agent automatically explore its surrounding by hitting & trail, taking action, learning from experiences, and improving its performance.
- Agent gets rewarded for each good action and get punished for each bad action; hence the goal of reinforcement learning agent is to maximize the rewards.
- In reinforcement learning, there is no labelled data like supervised learning, and agents learn from their experiences only.

Evolutionary Learning:

- Biological evolution can be seen as a learning process.
- Biological organisms adapt to improve their survival rates and chance of having offspring in their environment.

- Uses fitness score, which corresponds to a score for how good the current solution is.

Supervised Learning:

- The algorithm should produce sensible outputs for inputs that were not encountered during learning is called generalisation
- Supervised machine learning can be classified into two types of problems
  - Classification
  - Regression
- Classification algorithms are used to solve the classification problems in which the output variable is categorical, such as "Yes" or “No”.
- The classification algorithms predict the categories present in the dataset.
- Example: Spam Detection, Email Filtering
- Regression algorithms are used to solve regression problems in which there is a linear relationship between input and output variables.
- These are used to predict continuous output variables, such as market trends, weather prediction, etc.
- The algorithm should produce sensible outputs for inputs that were not encountered during learning is called generalisation.

The Machine Learning Process:

1. Data Collection and Preparation
   - Machine learning algorithms need significant amounts of data preferably without too much noise.
   - It is hard to collect data because they are in a variety of a places and formats and merging it appropriately is difficult.
   - We should ensure data should be clean that is it does not have significant errors, missing data.

2. Feature Selection
   - It consists of identifying the features that are most useful for the problem
3. Choose Appropriate Algorithm
   - The knowledge of the underlying principles of each algorithm and examples of their use is required.
4. 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.
5. Training
   - Given the dataset, algorithm, and parameters, training should be simply the use of computational resources in order to build a model
6. Evaluation
   - Before a system can be deployed it needs to be tested and evaluated for accuracy on data that it was not trained on.

The Brain and the Neuron:

- In animals, learning occurs within the brain.

- If we can understand how the brain works, then there might be things in there for us to copy and use for our machine learning systems.
- The brain is an impressively powerful and complicated system.
- It deals with noisy and even inconsistent data, and produces answers that are usually correct from very high dimensional data (such as images) very quickly.
- It weighs about 1.5 kg and is losing parts of itself all the time (neurons die as you age at impressive/depressing rates), but its performance does not degrade appreciably.
- The processing unit of the brain is called neuron (nerve cell)
- Signals can be received from dendrites, and sent down the axon once enough signals were received.
- We can mimic most of this process by coming up with a function that receives a list of weighted input signals and outputs some kind of signal if the sum of these weighted inputs reach a certain bias.
- Each neuron can be viewed as a separate processor, performing a very simple computation: deciding whether or not to fire.
- This makes the brain a massively parallel computer made up of 10^{11} processing elements.

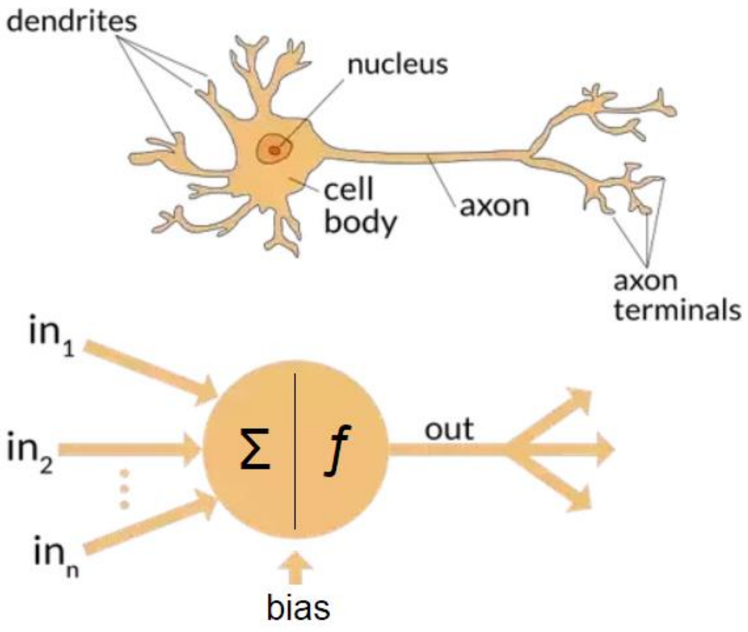


Fig.1 Biological Neuron Vs Artificial Neuron

Hebb's Rule:

- Hebb's rule says that the changes in the strength of synaptic connections are proportional to the correlation in the firing of the two connecting neurons.
- if two neurons consistently fire simultaneously, then any connection between them will change in strength, becoming stronger.
- if the two neurons never fire simultaneously, the connection between them will die away.
- The idea is that if two neurons both respond to something, then they should be connected.

McCulloch and Pitts Neurons:

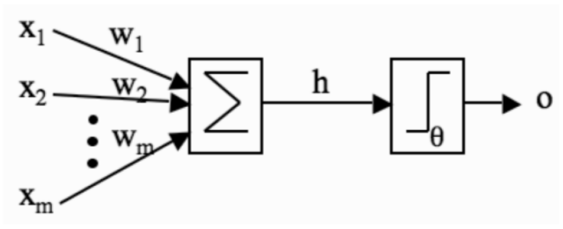


Fig.2 McCulloch and Pitts mathematical model of neuron

- The inputs x_i are multiplied by the weights w_i, and the neurons sum their values.
- If this sum is greater than the threshold then the neuron fires; otherwise it does not.
- a set of weighted inputs wi that correspond to the synapses
- an adder that sums the input signals (equivalent to the membrane of the cell that collects electrical charge)
- an activation function (initially a threshold function) that decides whether the neuron fires for the current inputs.

$$h = \sum_{i=1}^{m} w_i x_i,$$

$$o = g(h) = \begin{cases} 1 & \text{if } h > \theta \\ 0 & \text{if } h \leq \theta. \end{cases}$$

Limitations of McCulloch and pitts model:

- Inability to handle non-boolean inputs.
- The requirement to manually set thresholds.
- All inputs are treated equally; no weighting mechanism.
- Cannot handle functions that are not linearly separable

Design a Learning System:

1. Choosing the Training Experience
2. Choosing the Target Function
3. Choosing a Representation for the Target Function
4. Choosing a Function Approximation Algorithm
    a. Estimating Training Values
    b. Adjusting The Weights
5. The Final Design

Example:

checkers learning program

- Task T: playing checkers
- Performance measure ***P:*** percent of games won in the world tournament
- Training experience E: games played against itself
- Target function: V:Board →value
- Target function representation

  $$\hat{V}(b) = w_0 + w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + w_5x_5 + w_6x_6$$
- Least Squared Error Approximation Algorithm
- The final design of the checkers learning system can be described by four distinct program modules. These are:
  1. Performance System
  2. Critic
  3. Generalizer
  4. Experiment Generator

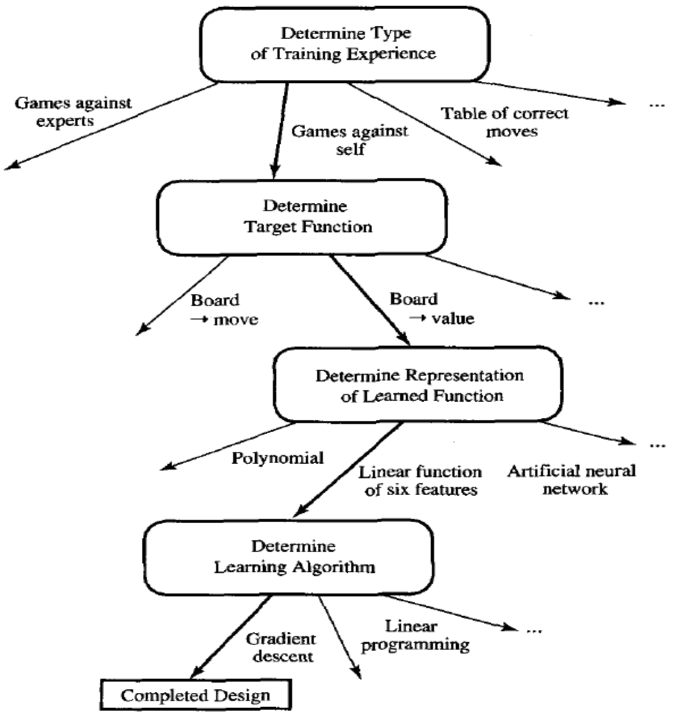

Determine Type of Training Experience
Games against experts
Games against self
Table of correct moves
...
Determine Target Function
Board → move
Board → value
...
Determine Representation of Learned Function
Polynomial
Linear function of six features
Artificial neural network
...
Determine Learning Algorithm
Gradient descent
Linear programming
...
Completed Design

Fig.3. Summary of choices in designing the checkers learning program

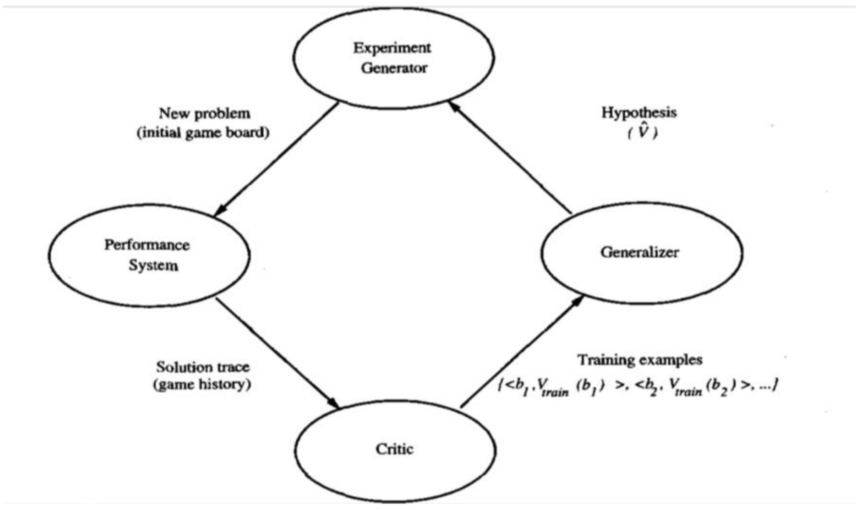

Experiment Generator
New problem (initial game board)
Hypothesis ($\hat{V}$)
Performance System
Generalizer
Solution trace (game history)
Training examples $\{<b_1, V_{train}(b_1)>, <b_2, V_{train}(b_2)>, ...\}$
Critic

Fig.4. Final design of the checkers learning program.

Perspectives and Issues in Machine Learning:

Perspectives:

- It involves searching a very large space of possible hypothesis to determine one that best fits the observed data and any prior knowledge held by the learner.
- Search a hypothesis space defined by some underlying representations (linear function, decision tree, ANN).
- These different hypothesis representations are appropriate for learning different kind of target functions.
- Learning as search problem where analysing the relationship between the size of hypothesis space to be searched, the number of training examples available, the confidence we can have that a hypothesis consistent with the training data will correctly generalize to unseen examples.

Generic Issues:

The generic issues machine are:

- What algorithms exist for learning general target functions from specific training examples?
- In what settings will particular algorithms converge to the desired function, given sufficient training data?
- Which algorithms perform best for which types of problems and representations?
- How much training data is sufficient?
- What general bounds can be found to relate the confidence in learned hypotheses to the amount of training experience and the character of the learner's hypothesis space?
- When and how can prior knowledge held by the learner guide the process of generalizing from examples?
- Can prior knowledge be helpful even when it is only approximately correct?
- What is the best strategy for choosing a useful next training experience, and how does the choice of this strategy alter the complexity of the learning problem?

- What is the best way to reduce the learning task to one or more function approximation problems?
- What specific functions should the system attempt to learn? Can this process itself be automated?
- How can the learner automatically alter its representation to improve its ability to represent and learn the target function?

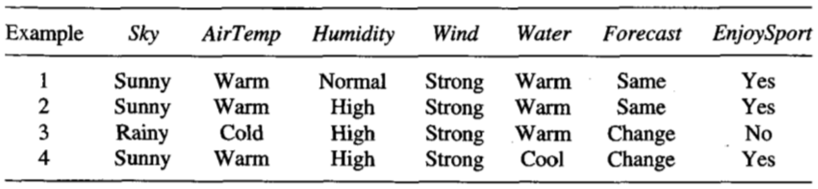
The answers to the above questions will solve the most of the issues in machine learning.

Specific issues in Machine Learning:

1. Inadequate Training Data
2. Poor quality of data
3. Non-representative training data
4. Overfitting and Underfitting
5. Monitoring and Maintenance
6. Getting Bad Recommendations
7. Lack of Skilled Resources
8. Customer Segmentation
9. Process Complexity of Machine Learning
10. Data Bias

Concept Learning Task:

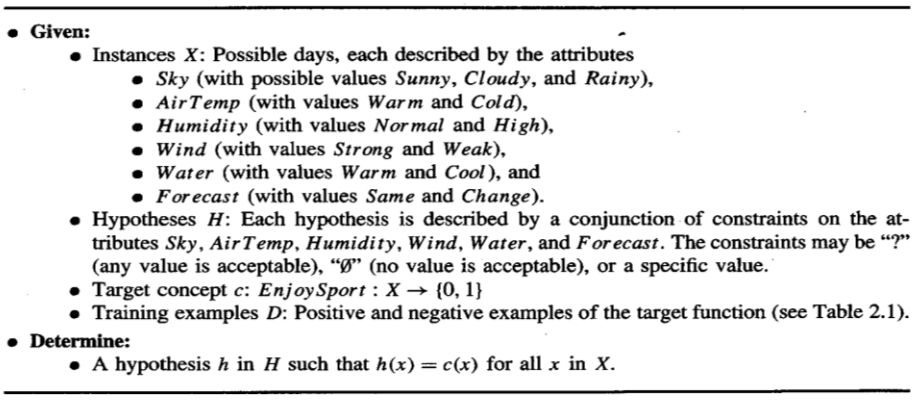
- Inferring a Boolean-valued function from training examples of its input and output is called concept learning.
- Example Problem: Days on which my friend enjoys his favourite water sport.



Example	*Sky*	*AirTemp*	*Humidity*	*Wind*	*Water*	*Forecast*	*EnjoySport*
1	Sunny	Warm	Normal	Strong	Warm	Same	Yes
2	Sunny	Warm	High	Strong	Warm	Same	Yes
3	Rainy	Cold	High	Strong	Warm	Change	No
4	Sunny	Warm	High	Strong	Cool	Change	Yes

Table: Positive and Negative Training examples for the target concept EnjoySport

- The most general hypothesis –that every day is a positive example is represented by <?, ?, ?, ?, ?, ?>
- The most specific possible hypothesis – that no day is a positive example. < ∅,∅,∅,∅,∅,∅>



- **Given:**
  - Instances X: Possible days, each described by the attributes
    - *Sky* (with possible values *Sunny*, *Cloudy*, and *Rainy*),
    - *AirTemp* (with values *Warm* and *Cold*),
    - *Humidity* (with values *Normal* and *High*),
    - *Wind* (with values *Strong* and *Weak*),
    - *Water* (with values *Warm* and *Cool*), and
    - *Forecast* (with values *Same* and *Change*).
  - Hypotheses H: Each hypothesis is described by a conjunction of constraints on the attributes *Sky*, *AirTemp*, *Humidity*, *Wind*, *Water*, and *Forecast*. The constraints may be "?" (any value is acceptable), "∅" (no value is acceptable), or a specific value.
  - Target concept c: $EnjoySport : X \rightarrow \{0, 1\}$
  - Training examples D: Positive and negative examples of the target function (see Table 2.1).
- **Determine:**
  - A hypothesis h in H such that $h(x) = c(x)$ for all x in X.

Table: The EnjoySport Concept Learning Task

Concept Learning as Search:

- Concept learning can be viewed as the task of searching through a large space of hypotheses implicitly defined by the hypothesis representation.
- The goal of this search is to find the hypothesis that best fits the training examples
- If we view learning as a search problem, then it is natural that our study of learning algorithms will examine different strategies for searching the hypothesis space.
- We will be particularly interested in algorithms capable of efficiently searching very large or infinite hypothesis spaces, to find the hypotheses that best fit the training data.

General-to-specific ordering of hypotheses:

- Many algorithms for concept learning organize the search through the hypothesis space by relying on a very useful structure that exists for any concept learning problem: a general-to-specific ordering of hypotheses.
- we can design learning algorithms that exhaustively search even infinite hypothesis spaces without explicitly enumerating every hypothesis.
- To illustrate the general-to-specific ordering, consider the two hypotheses.
  h1=<Sunyy,?,?,Strong,?,?>
  h2=<Sunny,?,?,?,?,?>
- Now consider the sets of instances that are classified positive by hl and by h2.
- Because h2 imposes fewer constraints on the instance, it classifies more instances as positive.
- In fact, any instance classified positive by hl will also be classified positive by h2. Therefore, we say that h2 is more general than hl

FIND-S: Finding a Maximally Specific Hypothesis:

```
1. Initialize h to the most specific hypothesis in H
2. For each positive training instance x
      • For each attribute constraint $a_i$ in h
            If the constraint $a_i$ is satisfied by x
            Then do nothing
            Else replace $a_i$ in h by the next more general constraint that is satisfied by x
3. Output hypothesis h
```

Example: EnjoySport

Example	*Sky*	*AirTemp*	*Humidity*	*Wind*	*Water*	*Forecast*	*EnjoySport*
1	Sunny	Warm	Normal	Strong	Warm	Same	Yes
2	Sunny	Warm	High	Strong	Warm	Same	Yes
3	Rainy	Cold	High	Strong	Warm	Change	No
4	Sunny	Warm	High	Strong	Cool	Change	Yes

- The first step of FIND-S algorithm is to initialize h to the most specific hypothesis in H.
  h← < Ø,Ø,Ø,Ø,Ø,Ø>

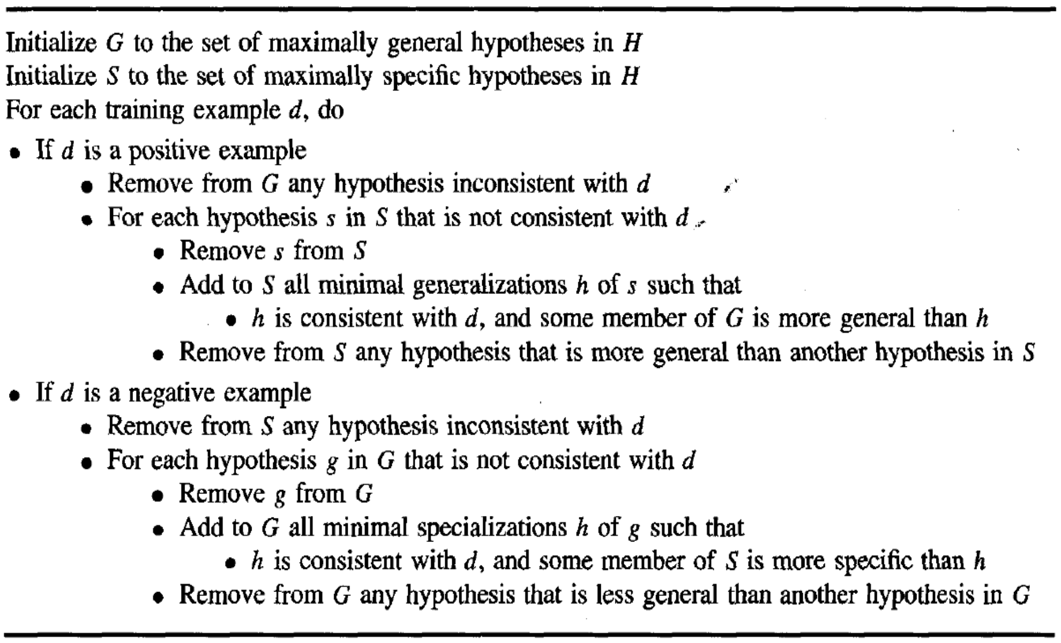
- Consider first training example
  h🡐 <Sunny, Warm, Normal, Strong, Warm, Same>
- Here h is still very specific.
- Next consider the second training example forces the algorithm to further generalize h.
  h🡐 <Sunny, Warm, ?, Strong, Warm, Same>
- Upon encountering the third training example-in this case a negative example- the algorithm makes no change to h.
- In fact, the FIND-S algorithm simply ignores every negative example.
- Consider the fourth training example leads to a further generalization of h.
  h🡐 <Sunny, Warm, ?, Strong, ?, ?>

Version Spaces:

- The version space represents the set of all hypotheses that are consistent with the training examples.
- It includes all hypotheses that correctly classify all the training examples as belonging to their respective categories.
- It is possible to represent the version space by two sets of hypotheses:
  (1) the most specific consistent hypotheses
  (2) the most general consistent hypotheses

The Candidate Elimination Algorithm:

- The key idea in the Candidate Elimination Algorithm is to output a description of the set of all hypotheses consistent with the training examples.
- Surprisingly, this algorithm computes the description of the set without explicitly enumerating all of its members



```
Initialize $G$ to the set of maximally general hypotheses in $H$
Initialize $S$ to the set of maximally specific hypotheses in $H$
For each training example $d$, do
• If $d$ is a positive example
    • Remove from $G$ any hypothesis inconsistent with $d$
    • For each hypothesis $s$ in $S$ that is not consistent with $d$
        • Remove $s$ from $S$
        • Add to $S$ all minimal generalizations $h$ of $s$ such that
            • $h$ is consistent with $d$, and some member of $G$ is more general than $h$
        • Remove from $S$ any hypothesis that is more general than another hypothesis in $S$
• If $d$ is a negative example
    • Remove from $S$ any hypothesis inconsistent with $d$
    • For each hypothesis $g$ in $G$ that is not consistent with $d$
        • Remove $g$ from $G$
        • Add to $G$ all minimal specializations $h$ of $g$ such that
            • $h$ is consistent with $d$, and some member of $S$ is more specific than $h$
        • Remove from $G$ any hypothesis that is less general than another hypothesis in $G$
```

Example: EnjoySport

Example	*Sky*	*AirTemp*	*Humidity*	*Wind*	*Water*	*Forecast*	*EnjoySport*
1	Sunny	Warm	Normal	Strong	Warm	Same	Yes
2	Sunny	Warm	High	Strong	Warm	Same	Yes
3	Rainy	Cold	High	Strong	Warm	Change	No
4	Sunny	Warm	High	Strong	Cool	Change	Yes

Initially : G = [[?,?,?,?,?,?],[?,?,?,?,?,?],[?,?,?,?,?,?],[?,?,?,?,?,?],[?,?,?,?,?,?],[?,?,?,?,?,?]]

S = [Null, Null, Null, Null, Null, Null]

For instance 1: <'sunny','warm','normal','strong','warm ','same'> and positive output.

G1 = G

S1 = ['sunny','warm','normal','strong','warm ','same']

For instance 2 : <'sunny','warm','high','strong','warm ','same'> and positive output.

G2 = G

S2 = ['sunny','warm',?,'strong','warm ','same']

For instance 3 : <'rainy','cold','high','strong','warm ','change'> and negative output.

G3 = [['sunny', ?, ?, ?, ?, ?], [?, 'warm', ?, ?, ?, ?], [?, ?, ?, ?, ?, ?],

[?, ?, ?, ?, ?, ?], [?, ?, ?, ?, ?, ?], [?, ?, ?, ?, ?, 'same']]

S3 = S2

For instance 4 : <'sunny','warm','high','strong','cool','change'> and positive output.

G4 = G3

S4 = ['sunny','warm',?,'strong', ?, ?]

At last, by synchronizing the G4 and S4 algorithm produce the output.

G = [['sunny', ?, ?, ?, ?, ?], [?, 'warm', ?, ?, ?, ?]]

S = ['sunny','warm',?,'strong', ?, ?]

Linear Discriminates:

- Linear Discriminants is a statistical method of dimensionality reduction that provides the highest possible discrimination among various classes.
- It is used in machine learning to find the linear combination of features, which can separate two or more classes of objects with best performance.
- It has been widely used in many applications, such as pattern recognition, image retrieval, speech recognition, among others.

- The method is based on discriminant functions that are estimated based on a set of data called training set.
- These discriminant functions are linear with respect to the characteristic vector, and usually have the form
  $f(t)=w^t x+b_0$,

  where w represents the weight vector, x the input vector, and b_0 a threshold.

Perceptron:

- The Perceptron is nothing more than a collection of McCulloch and Pitts neurons together with a set of inputs and some weights to fasten the inputs to the neurons

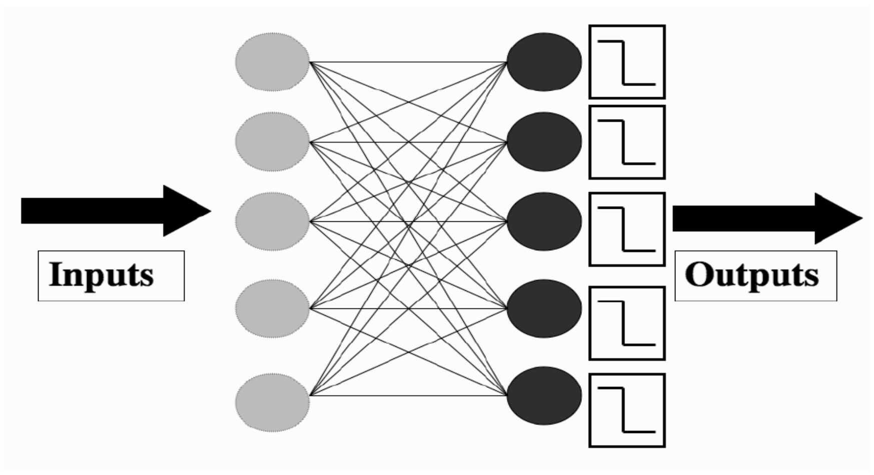


FIGURE 3.2 The Perceptron network, consisting of a set of input nodes (left) connected to McCulloch and Pitts neurons using weighted connections.

- Notice that the neurons in the Perceptron are completely independent of each other:
- It doesn't matter to any neuron what the others are doing
- It works out whether or not to fire by multiplying together its own weights and the input, adding them together, and comparing the result to its own threshold.
- The result is a pattern of firing and non-firing neurons, which looks like a vector of 0s and 1s
  Example: (0,1,0,0,1)

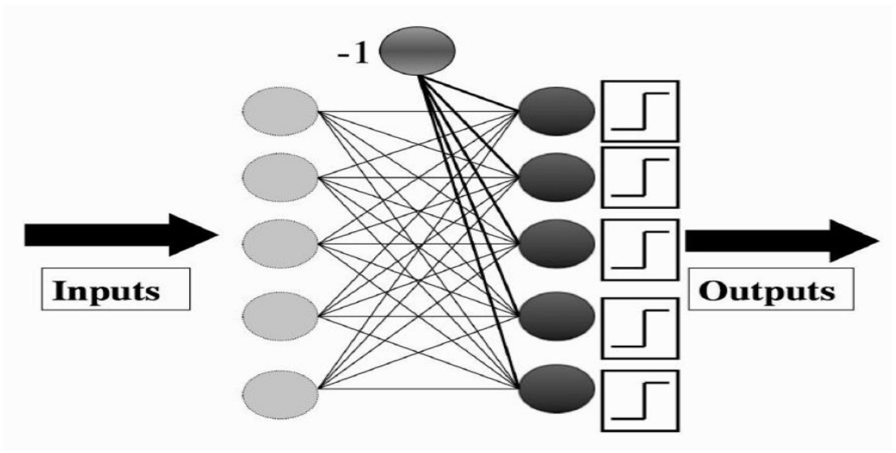


FIGURE 3.3 The Perceptron network again, showing the bias input.

The Learning Rate:

- This parameter learning rate determines **how fast or slow we will move towards the optimal weights**.
- If the learning rate is very large we will skip the optimal solution.
- If it is too small we will need too many iterations to converge to the best values. So using a good learning rate is crucial.
- We therefore use a moderate learning rate, typically $0.1 < \eta < 0.4$, depending upon how much error we expect in the inputs.

The Bias Input:

- A bias term is added to the input layer to provide the perceptron with additional flexibility in modeling complex patterns in the input data.

  ∑wi*xi = x1*w1 + x2*w2 + x3*w3+……..x4*w4

  Add a term called bias ‘b’ to this weighted sum to improve the model’s performance.

  Y=f(∑wi*xi + b)

  Where b is bias

There are three main reasons why we need to add bias:

1. It assists in achieving a better data fit and learning complex patterns
2. Handling zero inputs and mitigating the problem of vanishing gradient
3. Prevents underfitting and overfitting. Improves generalization

The Perceptron Learning Algorithm:

The Perceptron Algorithm

- **Initialisation**
  - set all of the weights w_{ij} to small (positive and negative) random numbers
- **Training**
  - for T iterations or until all the outputs are correct:
    - for each input vector:
      - compute the activation of each neuron j using activation function g:

$$y_j = g\left(\sum_{i=0}^{m} w_{ij}x_i\right) = \begin{cases} 1 & \text{if } \sum_{i=0}^{m} w_{ij}x_i > 0 \\ 0 & \text{if } \sum_{i=0}^{m} w_{ij}x_i \leq 0 \end{cases} \tag{3.4}$$

      - update each of the weights individually using:

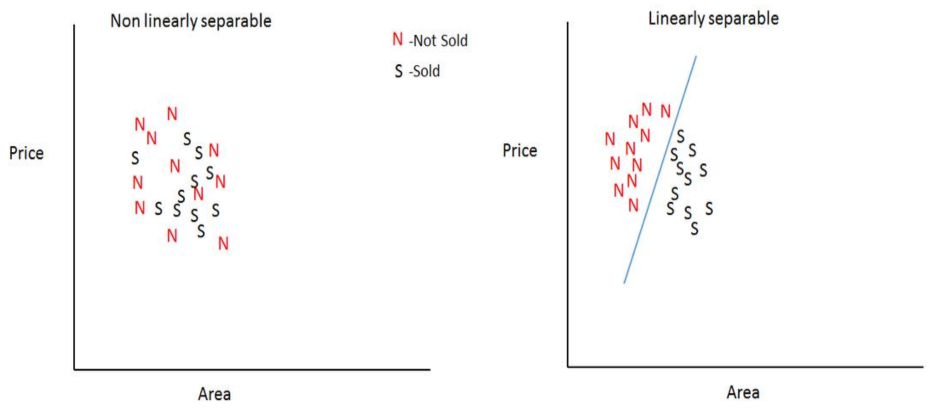
$$w_{ij} \leftarrow w_{ij} - \eta(y_j - t_j) \cdot x_i \tag{3.5}$$

- **Recall**
  - compute the activation of each neuron j using:

$$y_j = g\left(\sum_{i=0}^{m} w_{ij}x_i\right) = \begin{cases} 1 & \text{if } w_{ij}x_i > 0 \\ 0 & \text{if } w_{ij}x_i \leq 0 \end{cases} \tag{3.6}$$

Linear Seperability:

- Linear separability implies that if there are two classes then there will be a point, line, plane, or hyperplane that splits the input features in such a way that all points of one class are in one-half space and the second class is in the other half-space.
- For example, here is a case of selling a house based on area and price. We have got a number of data points for that along with the class, which is house Sold/Not Sold:



Linear Regression:

- Linear regression is a type of supervised machine learning algorithm that computes the linear relationship between the dependent variable and one or more independent features by fitting a linear equation to observed data.
- When there is only one independent feature, it is known as Simple Linear Regression, and when there are more than one feature, it is known as Multiple Linear Regression.
- Similarly, when there is only one dependent variable, it is considered Univariate Linear Regression, while when there are more than one dependent variables, it is known as Multivariate Regression.

Types of Linear Regression:

- There are two main types of linear regression
  1. Simple Linear Regression
  2. Multiple Linear Regression

Simple Linear Regression:

- This is the simplest form of linear regression.
- It involves only one independent variable and one dependent variable.
- The equation for simple linear regression is:
  $y=\beta_0+\beta_1X$

where:

- Y is the dependent variable

- X is the independent variable
- β0 is the intercept
- β1 is the slope

Multiple Linear Regression:

- This involves more than one independent variable and one dependent variable.
- The equation for multiple linear regression is:
  y=β0+β1X1+β2X2+………βnXn
  where:
  Y is the dependent variable
  X1, X2, …, Xn are the independent variables
  β0 is the intercept
  β1, β2, …, βn are the slopes
- The goal of the algorithm is to find the best Fit Line equation that can predict the values based on the independent variables.

Best Fit Line:

- The best Fit Line equation provides a straight line that represents the relationship between the dependent and independent variables.
- The slope of the line indicates how much the dependent variable changes for a unit change in the independent variable(s).
- Our primary objective while using linear regression is to locate the best-fit line, which implies that the error between the predicted and actual values should be kept to a minimum.
- There will be the least error in the best-fit line.

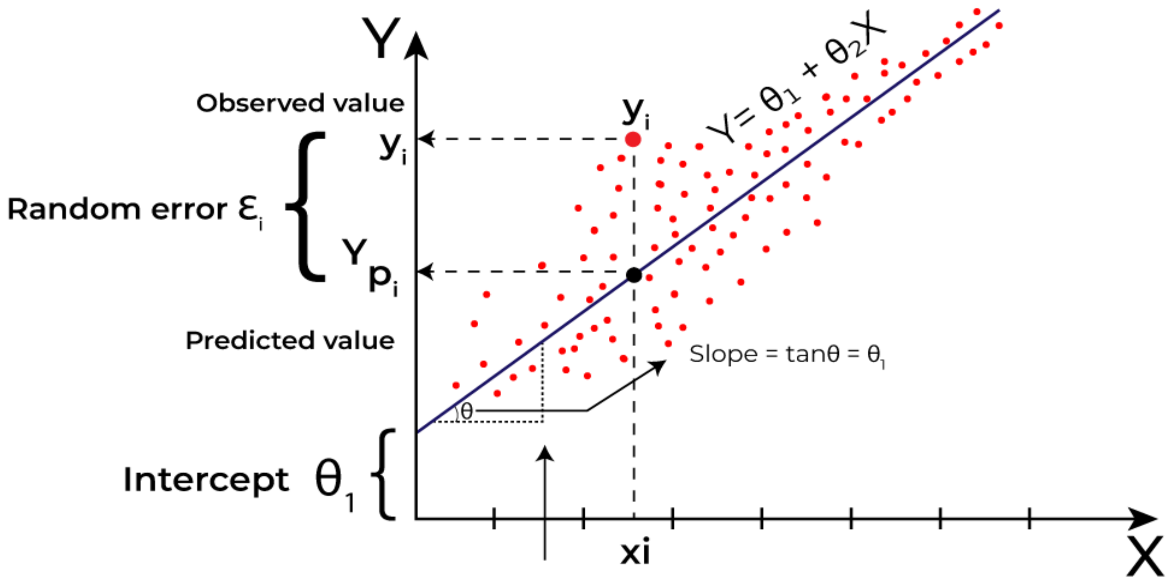

Y
Observed value
y_i
Random error ε_i
Y_{p_i}
Predicted value
Intercept θ_1
y_i
$Y = \theta_1 + \theta_2 X$
θ
Slope = tanθ = θ_1
xi
X

Fig: Linear Regression

Cost function for Linear Regression:

- The cost function or the loss function is nothing but the error or difference between the predicted value $\hat{Y}$ and the true value Y.
- In Linear Regression, the Mean Squared Error (MSE) cost function is employed, which calculates the average of the squared errors between the predicted values $\hat{y}_i$ and the actual values y_i.
- The purpose is to determine the optimal values for the intercept θ_1 and the coefficient of the input feature θ_2 providing the best-fit line for the given data points.
- The linear equation expressing this relationship is
  $\hat{y}_i = \theta_1 + \theta_2 x_i$
- MSE function can be calculated as:

$$\text{Cost function}(J) = \frac{1}{n}\sum_{n}^{i}(\hat{y}_i - y_i)^2$$

*****