UNIT V

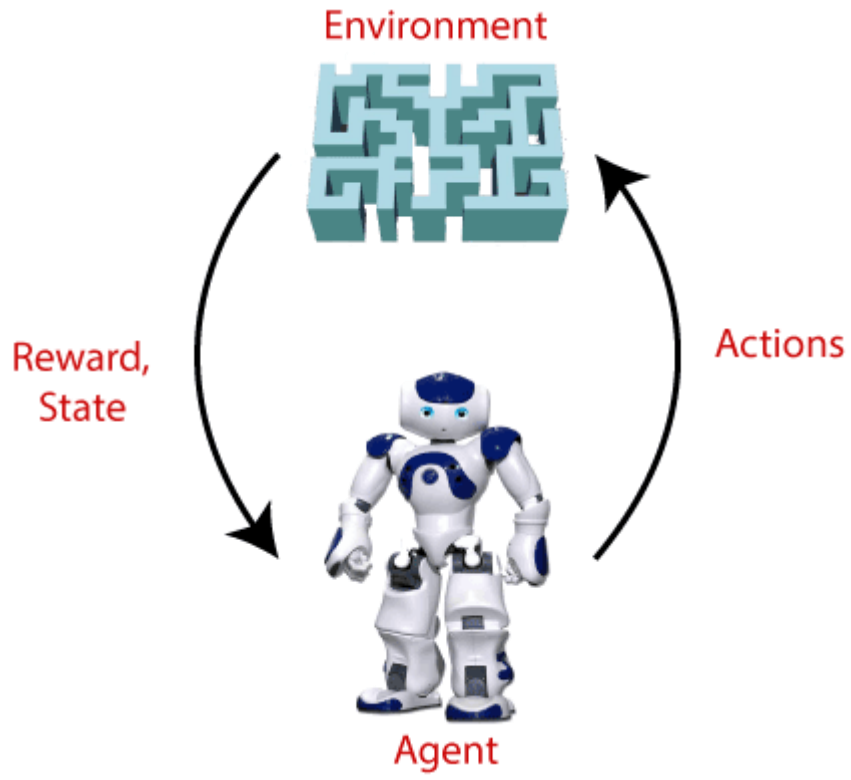
REINFORCEMENT LEARNING

INTRODUCTON REINFORCEMENT LEARNING

What is Reinforcement Learning?

- Reinforcement Learning is a feedback-based Machine learning technique in which an agent learns to behave in an environment by performing the actions and seeing the results of actions. For each good action, the agent gets positive feedback, and for each bad action, the agent gets negative feedback or penalty.
- In Reinforcement Learning, the agent learns automatically using feedbacks without any labelled data, unlike supervised learning.
- Since there is no labelled data, so the agent is bound to learn by its experience only.
- RL solves a specific type of problem where decision making is sequential, and the goal is long-term, such as game-playing, robotics, etc.
- The agent interacts with the environment and explores it by itself. The primary goal of an agent in reinforcement learning is to improve the performance by getting the maximum positive rewards.
- The agent learns with the process of hit and trial, and based on the experience, it learns to perform the task in a better way. Hence, we can say that "Reinforcement learning is a type of machine learning method where an intelligent agent (computer program) interacts with the environment and learns to act within that." How a Robotic dog learns the movement of his arms is an example of Reinforcement learning.
- It is a core part of Artificial intelligence, and all AI agent works on the concept of reinforcement learning. Here we do not need to pre-program the agent, as it learns from its own experience without any human intervention.

- Example: Suppose there is an AI agent present within a maze environment, and his goal is to find the diamond. The agent interacts with the environment by performing some actions, and based on those actions, the state of the agent gets changed, and it also receives a reward or penalty as feedback.
- The agent continues doing these three things (take action, change state/remain in the same state, and get feedback), and by doing these actions, he learns and explores the environment.
- The agent learns that what actions lead to positive feedback or rewards and what actions lead to negative feedback penalty. As a positive reward, the agent gets a positive point, and as a penalty, it gets a negative point.



Terms used in Reinforcement Learning:

- Agent(): An entity that can perceive/explore the environment and act upon it.
- Environment(): A situation in which an agent is present or surrounded by. In RL, we assume the stochastic environment, which means it is random in nature.
- Action(): Actions are the moves taken by an agent within the environment.
- State(): State is a situation returned by the environment after each action taken by the agent.

- Reward(): A feedback returned to the agent from the environment to evaluate the action of the agent.
- Policy(): Policy is a strategy applied by the agent for the next action based on the current state.
- Value(): It is expected long-term retuned with the discount factor and opposite to the short-term reward.
- Q-value(): It is mostly similar to the value, but it takes one additional parameter as a current action (a).

Key Features of Reinforcement Learning:

- In RL, the agent is not instructed about the environment and what actions need to be taken.
- It is based on the hit and trial process.
- The agent takes the next action and changes states according to the feedback of the previous action.
- The agent may get a delayed reward.
- The environment is stochastic, and the agent needs to explore it to reach to get the maximum positive rewards.

Approaches to implement Reinforcement Learning:

There are mainly three ways to implement reinforcement-learning in ML, which are:

1. **Value-based:**

   The value-based approach is about to find the optimal value function, which is the maximum value at a state under any policy. Therefore, the agent expects the long-term return at any state(s) under policy π.

2. **Policy-based:**

   Policy-based approach is to find the optimal policy for the maximum future rewards without using the value function. In this approach, the agent tries to apply such a policy that the action performed in each step helps to maximize the future reward. **The policy-based approach has mainly two types of policy:**

   - **Deterministic:** The same action is produced by the policy (π) at any state.

- **Stochastic**: In this policy, probability determines the produced action.

3. **Model-based:** In the model-based approach, a virtual model is created for the environment, and the agent explores that environment to learn it. There is no particular solution or algorithm for this approach because the model representation is different for each environment.

Single State Case in Reinforcement Learning

- One major difference between reinforcement learning and supervised learning is that a reinforcement-learner must explicitly explore its environment.
- In order to highlight the problems of exploration, we treat a very simple case in this section.
- The fundamental issues and approaches described here will, in many cases, transfer to the more complex instances of reinforcement learning discussed later in the paper.
- The simplest possible reinforcement-learning problem is known as the k-armed bandit problem, which has been the subject of a great deal of study in the statistics and applied mathematics literature .
- The agent is in a room with a collection of k gambling machines (each called a ``one-armed bandit'' in colloquial English).
- The agent is permitted a fixed number of pulls, h.
- Any arm may be pulled on each turn. The machines do not require a deposit to play; the only cost is in wasting a pull playing a suboptimal machine.
- When arm i is pulled, machine i pays off 1 or 0, according to some underlying probability parameter p_i , where payoffs are independent events and the p_i s are unknown.
- This problem illustrates the fundamental tradeoff between exploitation and exploration. The agent might believe that a particular arm has a fairly high payoff probability; should it choose that arm all the time, or should it choose another one that it has less information about, but seems to be worse? Answers to these questions depend on how long the agent is expected to play the game; the longer the game lasts, the worse the consequences of prematurely converging on a sub-optimal arm, and the more the agent should explore.

- There is a wide variety of solutions to this problem.
- We will consider a representative selection of them, but for a deeper discussion and a number of important theoretical results, see the book by Berry and Fristedt .
- We use the term ``action'' to indicate the agent's choice of arm to pull. This eases the transition into delayed reinforcement models in Section 3.
- It is very important to note that bandit problems fit our definition of a reinforcement-learning environment with a single state with only self transitions.
- Section 2.1 discusses three solutions to the basic one-state bandit problem that have formal correctness results.
- Although they can be extended to problems with real-valued rewards, they do not apply directly to the general multi-state delayed-reinforcement case.
- Section 2.2 presents three techniques that are not formally justified, but that have had wide use in practice, and can be applied (with similar lack of guarantee) to the general case.

Elements of Reinforcement Learning

There are four main elements of Reinforcement Learning, which are given below:

1. Policy
2. Reward Signal
3. Value Function
4. Model of the environment

1) Policy:

- A policy can be defined as a way how an agent behaves at a given time. It maps the perceived states of the environment to the actions taken on those states.
- A policy is the core element of the RL as it alone can define the behavior of the agent. In some cases, it may be a simple function or a lookup table, whereas, for other cases, it may involve general computation as a search process.
- It could be deterministic or a stochastic policy:

For deterministic policy: $a = \pi(s)$

For stochastic policy: $\pi(a \mid s) = P[At = a \mid St = s]$

2) Reward Signal:

- The goal of reinforcement learning is defined by the reward signal.
- At each state, the environment sends an immediate signal to the learning agent, and this signal is known as a reward signal.
- These rewards are given according to the good and bad actions taken by the agent.
- The agent's main objective is to maximize the total number of rewards for good actions.
- The reward signal can change the policy, such as if an action selected by the agent leads to low reward, then the policy may change to select other actions in the future.

3) Value Function:

- The value function gives information about how good the situation and action are and how much reward an agent can expect.
- A reward indicates the immediate signal for each good and bad action, whereas a value function specifies the good state and action for the future.
- The value function depends on the reward as, without reward, there could be no value. The goal of estimating values is to achieve more rewards.

4) Model:

- The last element of reinforcement learning is the model, which mimics the behavior of the environment.
- With the help of the model, one can make inferences about how the environment will behave.
- Such as, if a state and an action are given, then a model can predict the next state and reward.

- The model is used for planning, which means it provides a way to take a course of action by considering all future situations before actually experiencing those situations.
- The approaches for solving the RL problems with the help of the model are termed as the model-based approach.
- Comparatively, an approach without using a model is called a model-free approach.

How does Reinforcement Learning Work:

To understand the working process of the RL, we need to consider two main things:

- Environment: It can be anything such as a room, maze, football ground, etc.
- Agent: An intelligent agent such as AI robot.

MODEL BASED REINFORCEMENT LEARNING

- A machine learning model is defined as a mathematical representation of the output of the training process.
- Machine learning is the study of different algorithms that can improve automatically through experience & old data and build the model.
- A machine learning model is similar to computer software designed to recognize patterns or behaviors based on previous experience or data.
- The learning algorithm discovers patterns within the training data, and it outputs an ML model which captures these patterns and makes predictions on new data.

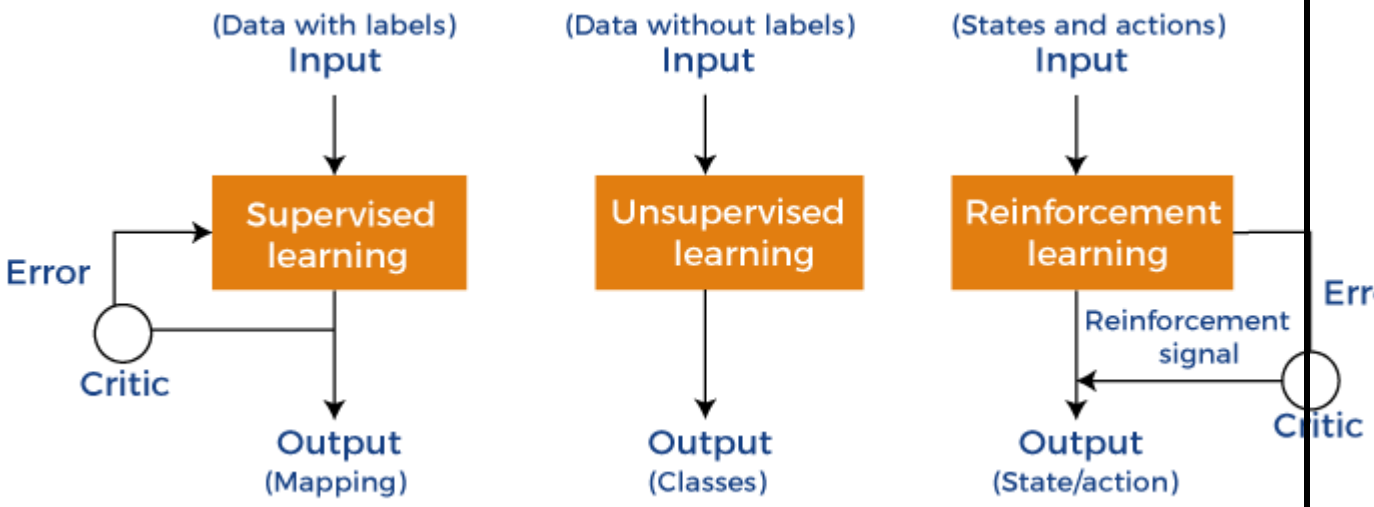
What is Machine Learning Model:

- Machine Learning models can be understood as a program that has been trained to find patterns within new data and make predictions.
- These models are represented as a mathematical function that takes requests in the form of input data, makes predictions on input data, and then provides an output in response.
- First, these models are trained over a set of data, and then they are provided an algorithm to reason over data, extract the pattern from feed data and learn from those data.
- Once these models get trained, they can be used to predict the unseen dataset.
- There are various types of machine learning models available based on different business goals and data sets.

Classification of Machine Learning Models:

Based on different business goals and data sets, there are three learning models for algorithms. Each machine learning algorithm settles into one of the three models:

- Supervised Learning
- Unsupervised Learning
- Reinforcement Learning



Supervised Learning is further divided into two categories:

- Classification
- Regression

Unsupervised Learning is also divided into below categories:

- Clustering
- Association Rule
- Dimensionality Reduction

1. Supervised Machine Learning Models:

- Supervised Learning is the simplest machine learning model to understand in which input data is called training data and has a known label or result as an output.
- So, it works on the principle of input-output pairs. It requires creating a function that can be trained using a training data set, and then it is applied to unknown data and makes some predictive performance.
- Supervised learning is task-based and tested on labeled data sets.
- We can implement a supervised learning model on simple real-life problems. For example, we have a dataset consisting of age and height; then, we can build a supervised learning model to predict the person's height based on their age.

Supervised Learning models are further classified into two categories:

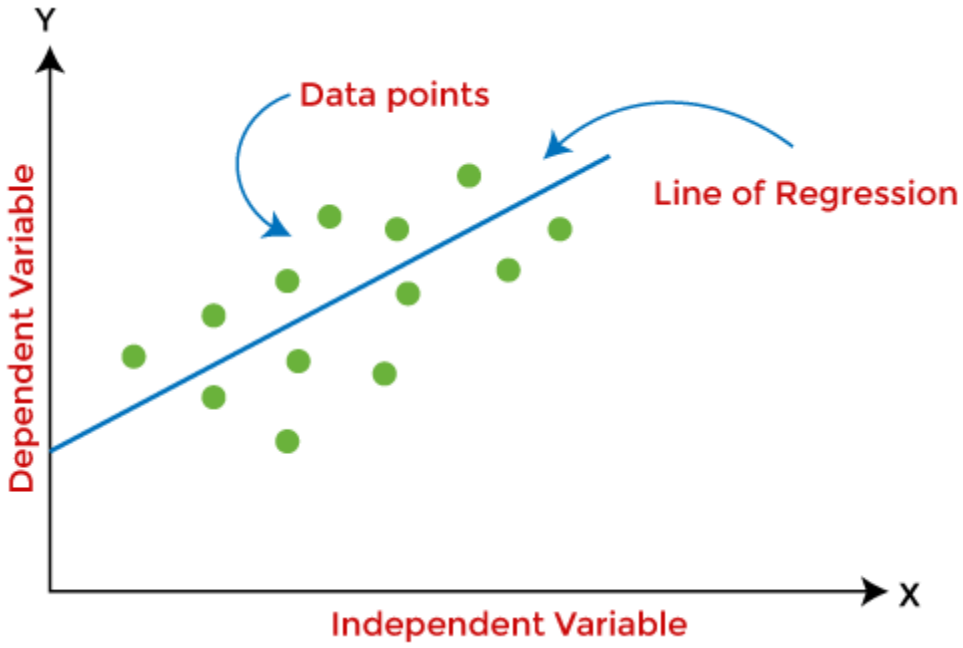
Regression

In regression problems, the output is a continuous variable. Some commonly used Regression models are as follows:

a) Linear Regression

- Linear regression is the simplest machine learning model in which we try to predict one output variable using one or more input variables.
- The representation of linear regression is a linear equation, which combines a set of input values(x) and predicted output(y) for the set of those input values.
- It is represented in the form of a line:

Y = bx+ c.



- The main aim of the linear regression model is to find the best fit line that best fits the data points.
- Linear regression is extended to multiple linear regression (find a plane of best fit) and polynomial regression (find the best fit curve).

b) Decision Tree

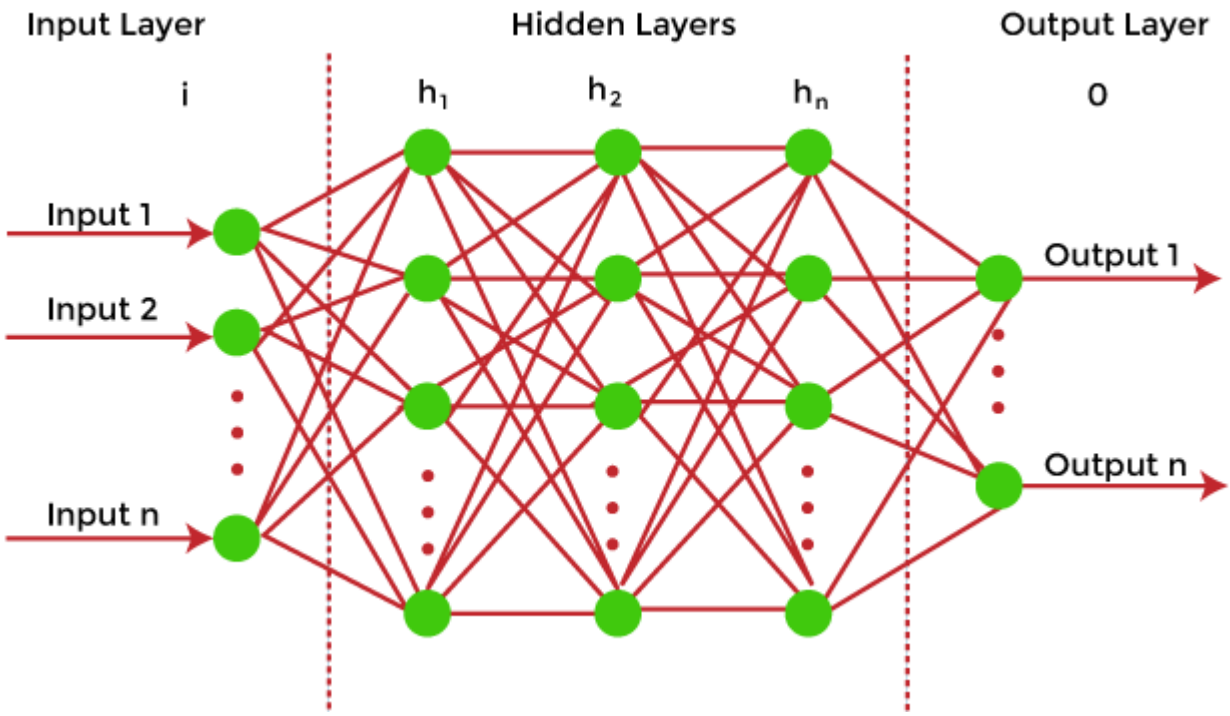
- Decision trees are the popular machine learning models that can be used for both regression and classification problems.
- A decision tree uses a tree-like structure of decisions along with their possible consequences and outcomes. In this, each internal node is used to represent a test on an attribute; each branch is used to represent the outcome of the test. The more nodes a decision tree has, the more accurate the result will be.
- The advantage of decision trees is that they are intuitive and easy to implement, but they lack accuracy.
- Decision trees are widely used in operations research, specifically in decision analysis, strategic planning, and mainly in machine learning.

c) Random Forest

- Random Forest is the ensemble learning method, which consists of a large number of decision trees.
- Each decision tree in a random forest predicts an outcome, and the prediction with the majority of votes is considered as the outcome.
- A random forest model can be used for both regression and classification problems.
- For the classification task, the outcome of the random forest is taken from the majority of votes. Whereas in the regression task, the outcome is taken from the mean or average of the predictions generated by each tree.

d) Neural Networks

- Neural networks are the subset of machine learning and are also known as artificial neural networks.
- Neural networks are made up of artificial neurons and designed in a way that resembles the human brain structure and working.
- Each artificial neuron connects with many other neurons in a neural network, and such millions of connected neurons create a sophisticated cognitive structure.



- Neural networks consist of a multilayer structure, containing one input layer, one or more hidden layers, and one output layer.
- As each neuron is connected with another neuron, it transfers data from one layer to the other neuron of the next layers.
- Finally, data reaches the last layer or output layer of the neural network and generates output.
- Neural networks depend on training data to learn and improve their accuracy.
- However, a perfectly trained & accurate neural network can cluster data quickly and become a powerful machine learning and AI tool.
- One of the best-known neural networks is Google's search algorithm.

Classification

- Classification models are the second type of Supervised Learning techniques, which are used to generate conclusions from observed values in the categorical form.
- For example, the classification model can identify if the email is spam or not; a buyer will purchase the product or not, etc.
- Classification algorithms are used to predict two classes and categorize the output into different groups.

- In classification, a classifier model is designed that classifies the dataset into different categories, and each category is assigned a label.

There are two types of classifications in machine learning:

- **Binary classification:** If the problem has only two possible classes, called a binary classifier. For example, cat or dog, Yes or No,
- **Multi-class classification:** If the problem has more than two possible classes, it is a multi-class classifier.

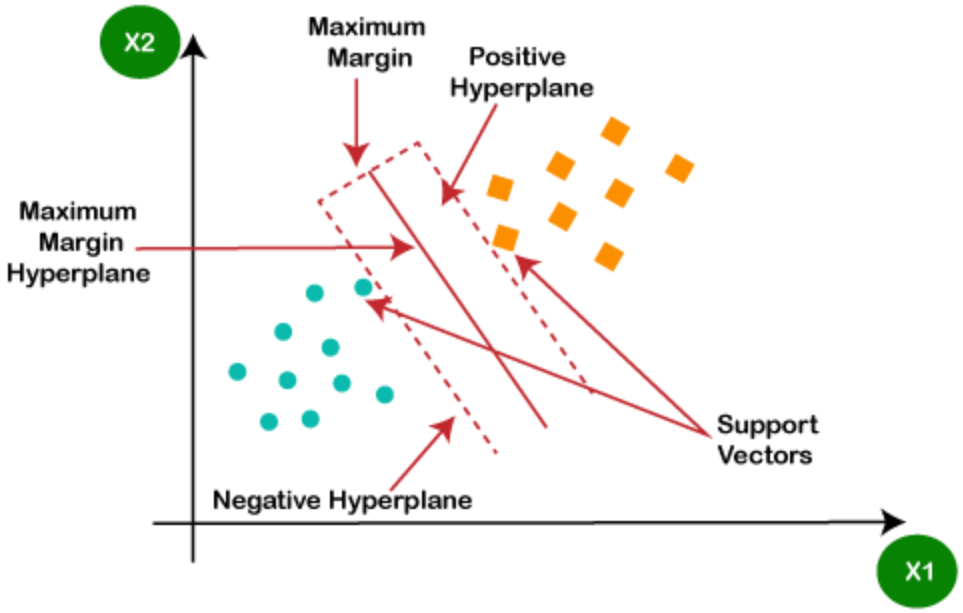
Some popular classification algorithms are as below:

a) Logistic Regression

- Logistic Regression is used to solve the classification problems in machine learning. They are similar to linear regression but used to predict the categorical variables.
- It can predict the output in either Yes or No, 0 or 1, True or False, etc. However, rather than giving the exact values, it provides the probabilistic values between 0 & 1.

b) Support Vector Machine

- Support vector machine or SVM is the popular machine learning algorithm, which is widely used for classification and regression tasks.
- However, specifically, it is used to solve classification problems.
- The main aim of SVM is to find the best decision boundaries in an N-dimensional space, which can segregate data points into classes, and the best decision boundary is known as Hyperplane.
- SVM selects the extreme vector to find the hyperplane, and these vectors are known as support vectors.



c) Naïve Bayes

- Naïve Bayes is another popular classification algorithm used in machine learning.
- It is called so as it is based on Bayes theorem and follows the naïve(independent) assumption between the features which is given as:

$$P(y|X) = \frac{P(X|y) * P(y)}{P(X)}$$

- Each naïve Bayes classifier assumes that the value of a specific variable is independent of any other variable/feature.
- For example, if a fruit needs to be classified based on color, shape, and taste. So yellow, oval, and sweet will be recognized as mango. Here each feature is independent of other features.

2. Unsupervised Machine learning models

- Unsupervised Machine learning models implement the learning process opposite to supervised learning, which means it enables the model to learn from the unlabeled training dataset.

- Based on the unlabeled dataset, the model predicts the output. Using unsupervised learning, the model learns hidden patterns from the dataset by itself without any supervision.

Unsupervised learning models are mainly used to perform three tasks, which are as follows:

Clustering

- Clustering is an unsupervised learning technique that involves clustering or groping the data points into different clusters based on similarities and differences.
- The objects with the most similarities remain in the same group, and they have no or very few similarities from other groups.
- Clustering algorithms can be widely used in different tasks such as Image segmentation, Statistical data analysis, Market segmentation, etc.
- Some commonly used Clustering algorithms are K-means Clustering, hierarchal Clustering, DBSCAN, etc.

Association Rule Learning:

- Association rule learning is an unsupervised learning technique, which finds interesting relations among variables within a large dataset.
- The main aim of this learning algorithm is to find the dependency of one data item on another data item and map those variables accordingly so that it can generate maximum profit.
- This algorithm is mainly applied in Market Basket analysis, Web usage mining, continuous production, etc.
  Some popular algorithms of Association rule learning are Apriori Algorithm, Eclat, FP-growth algorithm.

Dimensionality Reduction:

- The number of features/variables present in a dataset is known as the dimensionality of the dataset, and the technique used to reduce the dimensionality is known as the dimensionality reduction technique.
- Although more data provides more accurate results, it can also affect the performance of the model/algorithm, such as overfitting issues.
- In such cases, dimensionality reduction techniques are used.
  "It is a process of converting the higher dimensions dataset into lesser dimensions dataset ensuring that it provides similar information."
- Different dimensionality reduction methods such as PCA(Principal Component Analysis), Singular Value Decomposition, etc.

Reinforcement Learning

- In reinforcement learning, the algorithm learns actions for a given set of states that lead to a goal state.
- It is a feedback-based learning model that takes feedback signals after each state or action by interacting with the environment.
- This feedback works as a reward (positive for each good action and negative for each bad action), and the agent's goal is to maximize the positive rewards to improve their performance.

- The behavior of the model in reinforcement learning is similar to human learning, as humans learn things by experiences as feedback and interact with the environment.

Below are some popular algorithms that come under reinforcement learning:

- **Q-learning:** Q-learning is one of the popular model-free algorithms of reinforcement learning, which is based on the Bellman equation. It aims to learn the policy that can help the AI agent to take the best action for maximizing the reward under a specific circumstance. It incorporates Q values for each state-action pair that indicate the reward to following a given state path, and it tries to maximize the Q-value.

- **State-Action-Reward-State-Action (SARSA):** SARSA is an On-policy algorithm based on the Markov decision process. It uses the action performed by the current policy to learn the Q-value. The SARSA algorithm stands for State Action Reward State Action, which symbolizes the tuple (s, a, r, s', a').
- **Deep Q Network**: DQN or Deep Q Neural network is Q-learning within the neural network. It is basically employed in a big state space environment where defining a Q-table would be a complex task. So, in such a case, rather than using Q-table, the neural network uses Q-values for each action based on the state.

Training Machine Learning Models:

- Once the Machine learning model is built, it is trained in order to get the appropriate results. To train a machine learning model, one needs a huge amount of pre-processed data.
- Here pre-processed data means data in structured form with reduced null values, etc. If we do not provide pre-processed data, then there are huge chances that our model may perform terribly.

How to choose the best model:

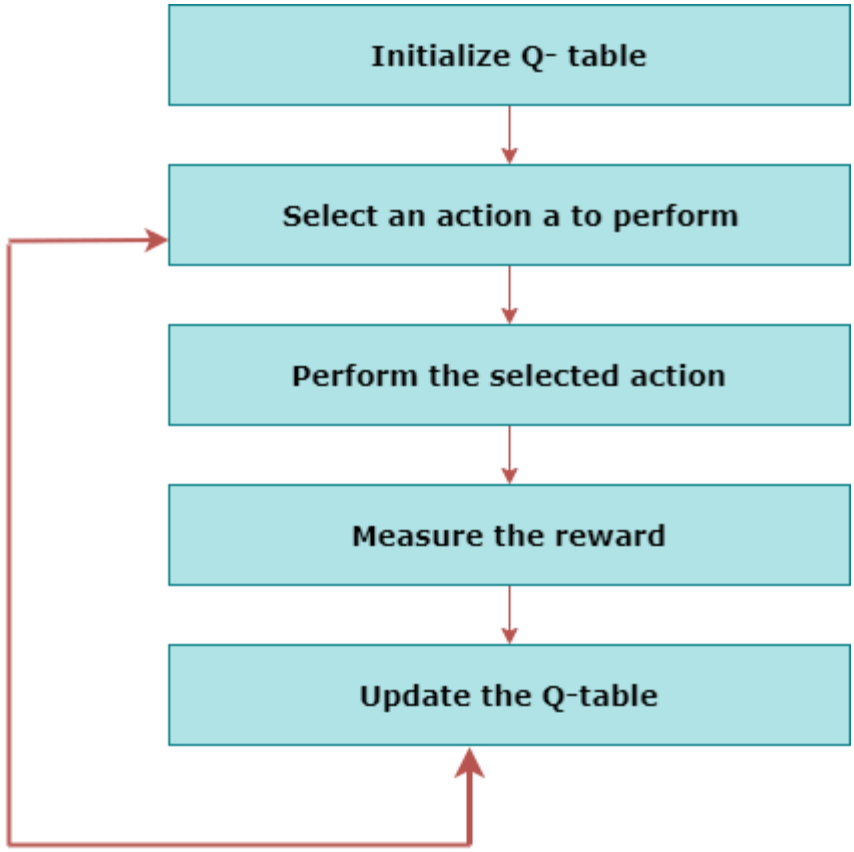
- In the above section, we have discussed different machine learning models and algorithms. But one most confusing question that may arise to any beginner that "which model should I choose?".

- So, the answer is that it depends mainly on the business requirement or project requirement. Apart from this, it also depends on associated attributes, the volume of the available dataset, the number of features, complexity, etc.
- However, in practice, it is recommended that we always start with the simplest model that can be applied to the particular problem and then gradually enhance the complexity & test the accuracy with the help of parameter tuning and cross-validation.

Reinforcement Learning Algorithms

Reinforcement learning algorithms are mainly used in AI applications and gaming applications. The main used algorithms are:

- **Q-Learning:**
  - Q-learning is an Off policy RL algorithm, which is used for the temporal difference Learning. The temporal difference learning methods are the way of comparing temporally successive predictions.
  - It learns the value function Q (S, a), which means how good to take action "a" at a particular state "s."
  - The below flowchart explains the working of Q- learning:



- **State Action Reward State action (SARSA):**
  - SARSA stands for State Action Reward State action, which is an on-policy temporal difference learning method. The on-policy control method selects the action for each state while learning using a specific policy.
  - The goal of SARSA is to calculate the Q π (s, a) for the selected current policy π and all pairs of (s-a).
  - The main difference between Q-learning and SARSA algorithms is that unlike Q-learning, the maximum reward for the next state is not required for updating the Q-value in the table.
  - In SARSA, new action and reward are selected using the same policy, which has determined the original action.
  - The SARSA is named because it uses the quintuple Q(s, a, r, s', a'). Where,

    s: original state

    a: Original action

    r: reward observed while following the states

    s' and a': New state, action pair.

- **Deep Q Neural Network (DQN):**
  - As the name suggests, DQN is a Q-learning using Neural networks.
  - For a big state space environment, it will be a challenging and complex task to define and update a Q-table.
  - To solve such an issue, we can use a DQN algorithm. Where, instead of defining a Q-table, neural network approximates the Q-values for each action and state.

Now, we will expand the Q-learning.

Q-Learning Explanation:

- Q-learning is a popular model-free reinforcement learning algorithm based on the Bellman equation.
- The main objective of Q-learning is to learn the policy which can inform the agent that what actions should be taken for maximizing the reward under what circumstances.
- It is an off-policy RL that attempts to find the best action to take at a current state.
- The goal of the agent in Q-learning is to maximize the value of Q.
- The value of Q-learning can be derived from the Bellman equation. Consider the Bellman equation given below:

$$V(s) = \max [R(s,a) + \gamma \sum_{s'} P(s,a,s')V(s`)]$$

Over fitting and Under fitting in Machine Learning

What is generalization in machine learning?

Generalization refers to your model's ability to adapt properly to new, previously unseen data, drawn from the same distribution as the one used to create the model.

Over fitting and Under fitting 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 under fitting and over fitting are the two terms that need to be checked for the performance of the model and whether the model is generalizing well or not.

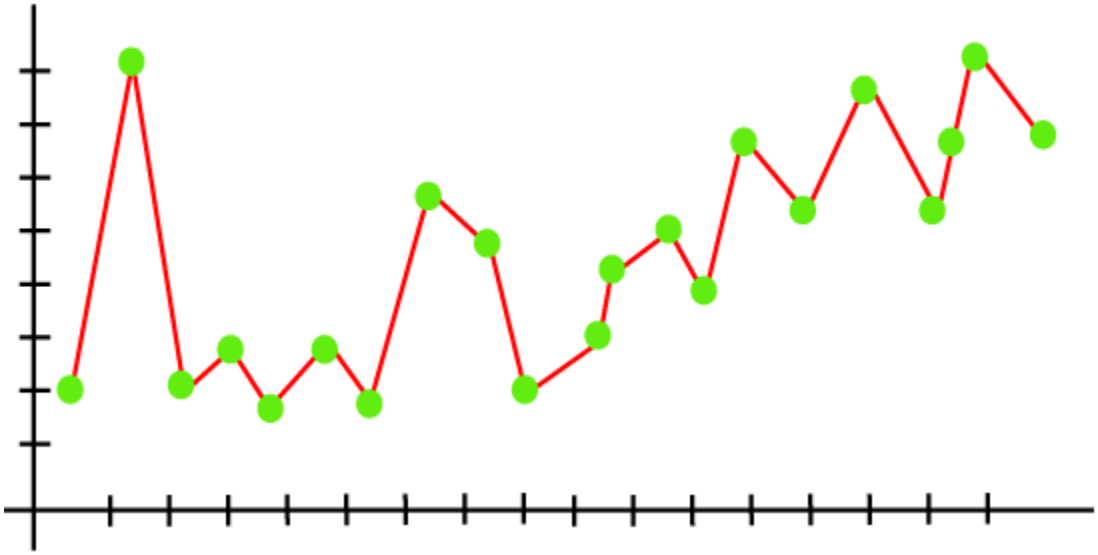
Before understanding the over fitting and under fitting, let's understand some basic term that will help to understand this topic well:

- **Signal:** It refers to the true underlying pattern of the data that helps the machine learning model to learn from the data.
- **Noise:** Noise is unnecessary and irrelevant data that reduces the performance of the model.
- **Bias:** Bias is a prediction error that is introduced in the model due to oversimplifying the machine learning algorithms. Or it is the difference between the predicted values and the actual values.
- **Variance:** If the machine learning model performs well with the training dataset, but does not perform well with the test dataset, then variance occurs.

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 supervised learning.

Example: The concept of the overfitting can be understood by the below graph of the linear regression output:



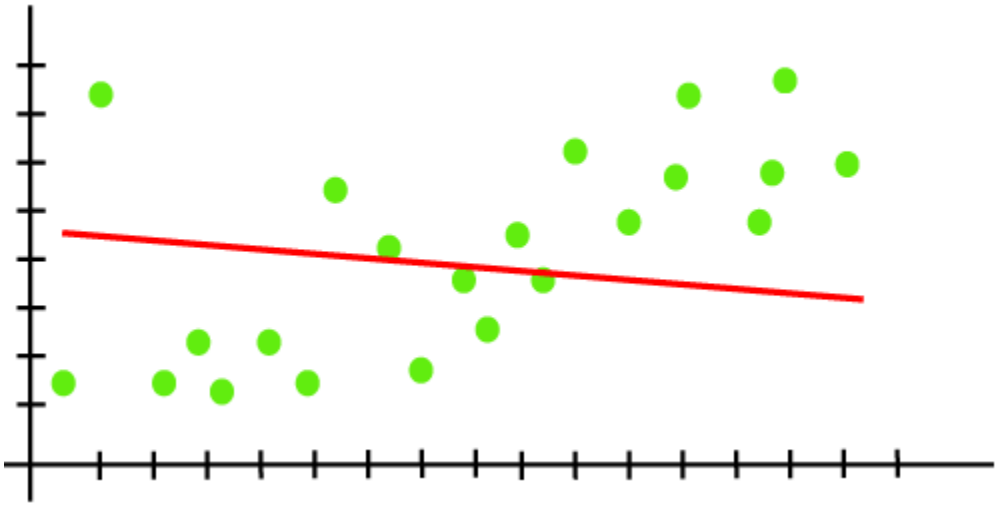
- As we can see from the above graph, the model tries to cover all the data points present in the scatter plot.
- It may look efficient, but in reality, it is not so. Because the goal of the regression model to find the best fit line, but here we have not got any best fit, so, it will generate the prediction 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 :

- Underfitting occurs when our machine learning model is not able to capture the underlying trend of the data.
- To avoid the overfitting in the model, the fed of training data can be stopped at an early stage, due to which the model may not learn enough from the training data.
- As a result, it may fail to find the best fit of the dominant trend in the data.
- In the case of underfitting, the model is not able to learn enough from the training data, and hence it reduces the accuracy and produces unreliable predictions.
- An underfitted model has high bias and low variance.



As we can see from the above diagram, the model is unable to capture the data points present in the plot.

How to avoid underfitting:

- By increasing the training time of the model.
- By increasing the number of features.

Goodness of Fit

- The "Goodness of fit" term is taken from the statistics, and the goal of the machine learning models to achieve the goodness of fit.

- In statistics modeling, it defines how closely the result or predicted values match the true values of the dataset.
- The model with a good fit is between the underfitted and overfitted model, and ideally, it makes predictions with 0 errors, but in practice, it is difficult to achieve it.
- As when we train our model for a time, the errors in the training data go down, and the same happens with test data.
- But if we train the model for a long duration, then the performance of the model may decrease due to the overfitting, as the model also learn the noise present in the dataset.
- The errors in the test dataset start increasing, so the point, just before the raising of errors, is the good point, and we can stop here for achieving a good model.
- There are two other methods by which we can get a good point for our model, which are the resampling method to estimate model accuracy and validation dataset.

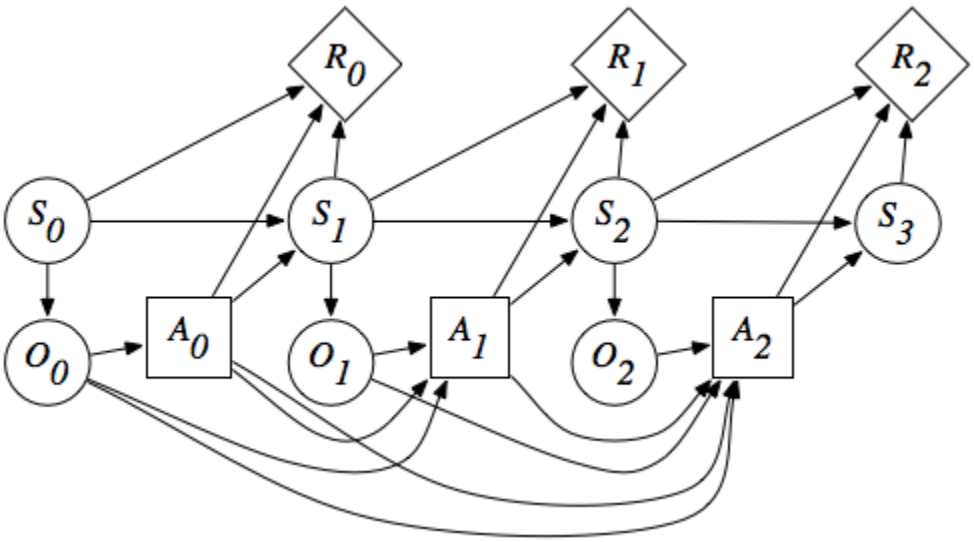
Partially Observable Decision Processes

- A partially observable Markov decision process (POMDP) is a combination of an MDP and a hidden Markov model.
- Instead of assuming that the state is observable, we assume that there are some partial and/or noisy observations of the state that the agent gets to observe before it has to act.

A POMDP consists of the following:

- S, a set of states of the world;
- A, a set of actions;
- O, a set of possible observations;
- $P(S_0)$, which gives the probability distribution of the starting state;
- P(S'|S,A), which specifies the dynamics - the probability of getting to state S' by doing action A from state S;
- R(S,A,S'), which gives the expected reward of starting in state S, doing action A, and transitioning to state S'; and
- P(O|S), which gives the probability of observing O given the state is S.

- A **partially observable system** is one in which the entire state of the system is not fully visible to an external sensor.
- In a partially observable system the observer may utilise a memory system in order to add information to the observer's understanding of the system.
- An example of a partially observable system would be a card game in which some of the cards are discarded into a pile face down.
- In this case the observer is only able to view their own cards and potentially those of the dealer.
- They are not able to view the face-down (used) cards, nor the cards that will be dealt at some stage in the future.
- A memory system can be used to remember the previously dealt cards that are now on the used pile. This adds to the total sum of knowledge that the observer can use to make decisions.
- In contrast, a fully observable system would be that of chess.
- In chess (apart from the 'who is moving next' state, and minor subtleties such as whether a side has castled, which may not be clear) the full state of the system is observable at any point in time.
- Partially observable is a term used in a variety of mathematical settings, including that of artificial intelligence and partially observable Markov decision processes.



Case study

Samuel's Checkers Player

An important precursor to Tesauro's TD-Gammon was the seminal work of Arthur Samuel (1959, 1967) in constructing programs for learning to play checkers. Samuel was one of the first to make effective use of heuristic search methods and of what we would now call temporal-difference learning. His checkers players are instructive case studies in addition to being of historical interest. We emphasize the relationship of Samuel's methods to modern reinforcement learning methods and try to convey some of Samuel's motivation for using them.

Samuel first wrote a checkers-playing program for the IBM 701 in 1952. His first learning program was completed in 1955 and was demonstrated on television in 1956. Later versions of the program achieved good, though not expert, playing skill. Samuel was attracted to game-playing as a domain for studying machine learning because games are less complicated than problems "taken from life" while still allowing fruitful study of how heuristic procedures and learning can be used together. He chose to study checkers instead of chess because its relative simplicity made it possible to focus more strongly on learning.

Samuel's programs played by performing a lookahead search from each current position. They used what we now call heuristic search methods to determine how to expand the search tree

and when to stop searching. The terminal board positions of each search were evaluated, or "scored," by a value function, or "scoring polynomial," using linear function approximation. In this and other respects Samuel's work seems to have been inspired by the suggestions of Shannon (1950). In particular, Samuel's program was based on Shannon's minimax procedure to find the best move from the current position. Working backward through the search tree from the scored terminal positions, each position was given the score of the position that would result from the best move, assuming that the machine would always try to maximize the score, while the opponent would always try to minimize it. Samuel called this the backed-up score of the position. When the minimax procedure reached the search tree's root--the current position--it yielded the best move under the assumption that the opponent would be using the same evaluation criterion, shifted to its point of view. Some versions of Samuel's programs used sophisticated search control methods analogous to what are known as "alpha-beta" cutoffs (e.g., see Pearl, 1984).

Samuel used two main learning methods, the simplest of which he called rote learning. It consisted simply of saving a description of each board position encountered during play together with its backed-up value determined by the minimax procedure. The result was that if a position that had already been encountered were to occur again as a terminal position of a search tree, the depth of the search was effectively amplified since this position's stored value cached the results of one or more searches conducted earlier. One initial problem was that the program was not encouraged to move along the most direct path to a win. Samuel gave it a "a sense of direction" by decreasing a position's value a small amount each time it was backed up a level (called a ply) during the minimax analysis. "If the program is now faced with a choice of board positions whose scores differ only by the ply number, it will automatically make the most advantageous choice, choosing a low-ply alternative if winning and a high-ply alternative if losing" (Samuel, 1959, p. 80). Samuel found this discounting-like technique essential to successful learning. Rote learning produced slow but continuous improvement that was most effective for opening and endgame play. His program became a "better-than-average novice" after learning from many games against itself, a variety of human opponents, and from book games in a supervised learning mode.

Rote learning and other aspects of Samuel's work strongly suggest the essential idea of temporal-difference learning--that the value of a state should equal the value of likely following states. Samuel came closest to this idea in his second learning method, his "learning by

generalization" procedure for modifying the parameters of the value function. Samuel's method was the same in concept as that used much later by Tesauro in TD-Gammon. He played his program many games against another version of itself and performed a backup operation after each move. The idea of Samuel's backup is suggested by the diagram in Figure 11.3. Each open circle represents a position where the program moves next, an on-move position, and each solid circle represents a position where the opponent moves next. A backup was made to the value of each on-move position after a move by each side, resulting in a second on-move position. The backup was toward the minimax value of a search launched from the second on-move position. Thus, the overall effect was that of a backup consisting of one full move of real events and then a search over possible events, as suggested by Figure 11.3. Samuel's actual algorithm was significantly more complex than this for computational reasons, but this was the basic idea.

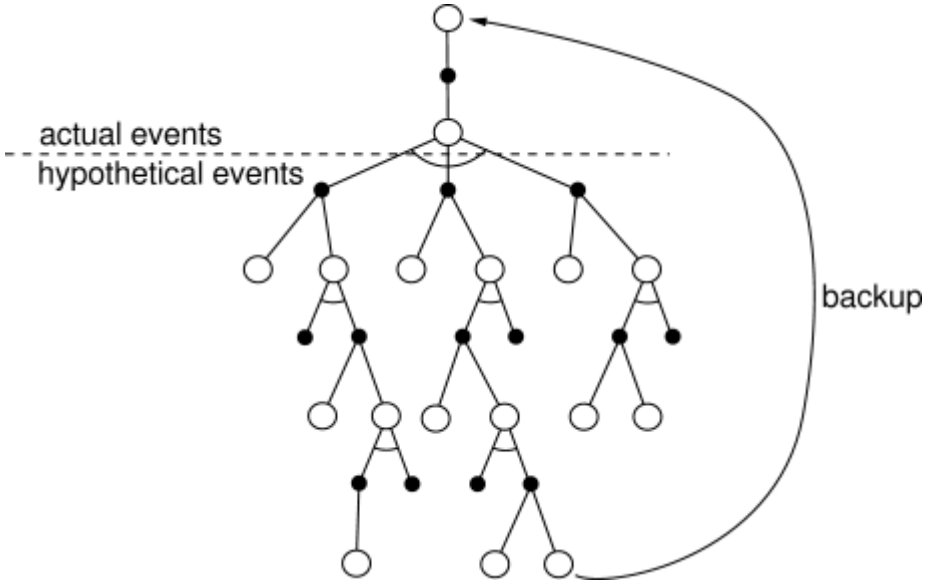


Figure 11.3: The backup diagram for Samuel's checkers player.

Samuel did not include explicit rewards. Instead, he fixed the weight of the most important feature, the piece advantage feature, which measured the number of pieces the program had relative to how many its opponent had, giving higher weight to kings, and including refinements so that it was better to trade pieces when winning than when losing. Thus, the goal

of Samuel's program was to improve its piece advantage, which in checkers is highly correlated with winning.

However, Samuel's learning method may have been missing an essential part of a sound temporal-difference algorithm. Temporal-difference learning can be viewed as a way of making a value function consistent with itself, and this we can clearly see in Samuel's method. But also needed is a way of tying the value function to the true value of the states. We have enforced this via rewards and by discounting or giving a fixed value to the terminal state. But Samuel's method included no rewards and no special treatment of the terminal positions of games. As Samuel himself pointed out, his value function could have become consistent merely by giving a constant value to all positions. He hoped to discourage such solutions by giving his piece-advantage term a large, nonmodifiable weight. But although this may decrease the likelihood of finding useless evaluation functions, it does not prohibit them. For example, a constant function could still be attained by setting the modifiable weights so as to cancel the effect of the nonmodifiable one.

Since Samuel's learning procedure was not constrained to find useful evaluation functions, it should have been possible for it to become worse with experience. In fact, Samuel reported observing this during extensive self-play training sessions. To get the program improving again, Samuel had to intervene and set the weight with the largest absolute value back to zero. His interpretation was that this drastic intervention jarred the program out of local optima, but another possibility is that it jarred the program out of evaluation functions that were consistent but had little to do with winning or losing the game.

Despite these potential problems, Samuel's checkers player using the generalization learning method approached "better-than-average" play. Fairly good amateur opponents characterized it as "tricky but beatable" (Samuel, 1959). In contrast to the rote-learning version, this version was able to develop a good middle game but remained weak in opening and endgame play. This program also included an ability to search through sets of features to find those that were most useful in forming the value function. A later version (Samuel, 1967) included refinements in its search procedure, such as alpha-beta pruning, extensive use of a supervised learning mode called "book learning," and hierarchical lookup tables called signature tables (Griffith, 1966) to represent the value function instead of linear function approximation. This version learned to play much better than the 1959 program, though still not at a master level. Samuel's checkers-

playing program was widely recognized as a significant achievement in artificial intelligence and machine learning.