# CS601PC: Machine Learning

## Course objectives:

* To Explain machine learning techniques like - decision tree learning
  - Bayesian learning etc.
* To understand computational learning theory.
* To study pattern comparison techniques.
* To understand concepts of ANN & Evaluation Hypothesis
* To study genetic Algorithms and understand the concepts of Reinforcement & Analytical learning.

## Course Outcomes:

* Students will be able to understand the concepts of computational intelligence like machine learning (ML).
* Ability to get the skill to apply ML techniques to address the real time problems in different areas.
* Understand Neural networks & its usage in ML applications.
* Understand the basic concepts of Reinforcement learning, analytical learning, genetic Algorithms etc.

# UNIT-I

## CHAPTER 1 :-

Introduction

- Well posed learning problems
- Designing a learning system
- perspectives and Issues in machine learning

## CHAPTER 2 :-

Concept learning & the general-to-specific ordering

- Introduction
- A concept learning task
- Concept learning as search
- Find S : Finding a maximally specific Hypothesis.
- Version spaces & "The candidate Elimination" Algorithm.
- Remarks on version spaces & "candidate Elimination".
- Inductive bias.

## CHAPTER 3 :-

Decision Tree learning

- Introduction
- Decision tree representation
- Appropriate problems for decision tree learning
- Basic decision tree learning algorithm.
- Hypothesis space search in decision Tree learning
- Inductive bias in Decision tree learning.
- Issues in Decision tree learning

# CHAPTER-1

## INTRODUCTION:

- A successful understanding of how to make computers learn would open up many new uses of computers & new levels of competence & customisation.
- A detail understanding of info. based processing algorithms for ML leads to a better understanding of human learning abilities.
- Ex:- computers learning from medical records, which treatment is effective for a new disease.
- ML algorithms are most effective compared to any other approaches in case of speech recognition, Data Mining etc.

## 1. Well posed learning problems:

**Definition:-**

A computer program is said to "learn" from an experience E w.r.t some class of tasks T & performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.

Ex:- Assume a prog. which "learns to play checkers" so the Task "T" is Playing checkers game.
So now, a computer prog. that learns "to play checkers" will improve its performance (performance is "ability to win") at the class of tasks involving "playing checkers game" through an experience E (E = playing game with itself) which is obtained by "playing games against itself".

- To have a Well defined problem, look for 3 features.
  * The class of tasks (T)
  * The measure of performance to be improved (P)
  * Source of Experience (E).

- For the above "checker game" example.
  T = Playing checker
  P = % of games won against opponents
  E = % of games practiced against itself.

- Following are other examples that can be given to specify ML problems:

Ex2:- Hand Writing Recognition Learning problem.
  T = Recognising & classifying handwritten words within images
  P = % of words correctly specified
  E = Training experience i.e. database of Handwritten words with given classification

Ex3:- Robot Driving Learning problem.
  T = Driving on 4 lane Highway using Vision Sensor
  P = Avg. distance travelled before an error
  E = Seq. of images & steering commands recorded while observing human driver.

2. Designing a Learning System:
  - Before understanding design issues & design approaches, first, we shall understand, to design a problem
  - Examples used: GOAL: Entering world checkers tournament
    TASK: Learn to play checkers.
    PERFORMANCE MEASURE: % of games it wins in the world tournament

→ Choosing the Training Experience :-

- Choose a type of training experience from which our sys will learn. Its v. imp for the success/failure of the learner.

① A key feature to select it will be to see if training experience will provide direct/indirect feedback.

ex:- Direct - Individual checkers board states & correct move for each.

Indirect - Move seqs. & final outcomes of various games played

- The disadv. of this is "credit assignment".
- Learning from direct feedback is easier.

② Degree to which learner controls the seq. of training examples.

- ex: (i) learner dependent on teacher to select informative boards & to provide correct move for each.

(ii) learner can propose board states & if finds a particular move confusing then it can ask teacher.

(iii) Learner can have control on both board states & training classifications & there is no teacher present

③ How well it rep. distribution of examples over which the final sys. performance P must be measured:

- generally, if future test example & distribution of learning examples (train) are similar then learning is more reliable.

- Example of checker game, if training experience is E, i.e. games is played against itself & performance measure P, is % of games won in world tournament then there is a danger that training experience will not be similar to the situation where it will be tested in future.

- But for simplicity, we still assume a fully specified Tasks:

(Task) T: playing checkers

(Performance Measure) P: % of games won in world Tournament

(Train experience) E: games played against itself

- Foll. steps are needed next to complete the design:
  (i) exact type of knowledge to be learned.
  (ii) Rep" for this target knowledge
  (iii) learning mechanism.

→ Choosing the Target function :-

- We shall now determine the Target knowledge & how will it be used.
- Ex; take a checkers playing prog. Assume, it can generate the "legal" moves from any board state. Lets say, prog needs to learn only the "best" move among them.
- Assume "ChooseMove" be a func. which chooses best move for any given board state.
- "ChooseMove : B→M" is the notation used to accept any board as i/p from a set of "legal board states" B and produces "M", which is an o/p move from a set of legal move M. But this func. is V. difficult to learn since it is an "indirect" training experience.
- Alternative Target func.: Let say, an evaluation func. 'V' assigns a numerical score to any board state.
  ie $V: B \to R$ ⇒ V will map any legal board state from set B to some real value.
  ⇒ The better board states the greater shall be "R" value. This method is comparatively easier for the sys. to learn and select best move from any board state.

Example:- if b is a final board state i.e won ⇒ V(b) = 100
if b " " " " i.e lost ⇒ V(b) = -100
if b " " " " i.e Drawn ⇒ V(b) = 0
if b is not a final state in game ⇒ V(b) = V(b')
where b' = best final board state that can be achieved

→ Choosing a repⁿ for Target func:-

- We've got the ideal target func. "V". Now we must choose $\hat{V}$ i.e function approximation
Because learning algo. can't acquire ideal or perfect target func. "V". Rather, they acquire only some approxⁿ to the target func. so that's why we shall choose $\hat{V}$ now.
- For the case of checkers game $\hat{V}$ can be:
  (i) collection of rules that match against features of board state
  (ii) Large table with distinct entry specifying the value for each distinct board state.
- For expressing $\hat{V}$ which is more similar to V, one needs a large amount of training data.
- $\hat{V}$ will be calculated as a linear combⁿ of foll. board features:
  (i) $x_1$ : no. of black pieces on board
  (ii) $x_2$ : " " red " "
  (iii) $x_3$ : " " black kings " "
  (iv) $x_4$ : " " Red kings " "
  (v) $x_5$ : no. of black pieces threatened by red.
  (vi) $x_6$ : " " red " " " black.

- So, learning prog. is
$$\hat{V}(b) = w_0 x_0 + w_1 x_1 + w_2 x_2 + w_3 x_3 + w_4 x_4 + w_5 x_5 + w_6 x_6$$
where, $w_0, w_1 \ldots w_6$ = weights chosen be algo.
- weights $w_1$ to $w_6$ determine the relative importance of various board features in finding out value of board.
$w_0$ adds an additive constant to the board value.
- After this step now, we have a partial design of checkers learning program.

Specification of Learning Task:
① Task T: playing checkers
② performance measure P: % of games won in World tournament
③ Training Experience E: games played against itself.

implementation of learning prog:
④ Target func.: V: Board → R
⑤ Target func. Repn: $\hat{V}(b) = w_0 + w_1 x_1 + w_2 x_2 + w_3 x_3 + w_4 x_4 + w_5 x_5 + w_6 x_6$

→ choosing a func. Approx$^n$ Algo.:-

- For learning the Target func. $\hat{V}$, a set of training examples is needed, i.e $V_{train}(b)$.
- Each example will describe a specific board state "b" & training value for it i.e $V_{train}(b)$.
- Consider an example, where, black has won game. then $x_2 = 0 \Rightarrow$ There are no red pieces on board & value of $V_{train}(b) = +100$.
→ So the ordered pair of training example will be in the form $\langle b, V_{train}(b) \rangle$.
For above example taken it is
$\langle x_1 = 3, x_2 = 0, x_3 = 1, x_4 = 0, x_5 = 0, x_6 = 0, +100 \rangle$

→ Estimating Training Values:-

- Only Training info. thats available is whether game was won/lost So we can assign Training values v. easily.
- But, some times there are "Intermediate" board states that occurs before game ends. i.e we cant say whether game path was good/bad or whether a subsequent poor move made the game to lose it.
→ There is a simple approach for such

Intermediate board states:

"Assign training value of $V_{train}(b)$ for any intermediate board state 'b' to be $\hat{V}(successor(b))$"

i.e $$V_{train}(b) \leftarrow \hat{V}(successor(b)).$$

where $\hat{V}$ = Learner's current approx$^n$ to V.

Successor(b) = next board state following b for which it is again the prog's turn to move.

→ Adjusting Weights:-

- we must choose weights $w_i$ so that it fits best to the set of training examples $\{\langle b, V_{train}(b)\rangle\}$
- Best fit:- First define best hypothesis as a set of weights which minimises E (squared error) b/w training values & predicted values by $\hat{V}$.

$$\Rightarrow E \equiv \sum_{\langle b, V_{train}(b)\rangle \in \text{Training examples}} (V_{train}(b) - \hat{V}(b))^2 .$$

- There are many Algos. for finding weights of a linear func. that minimises E. One of them is LMS (Least Mean Squares) Training Rule.

- LMS Weight update Rule:-

For each training example $\langle b, V_{train}(b)\rangle$

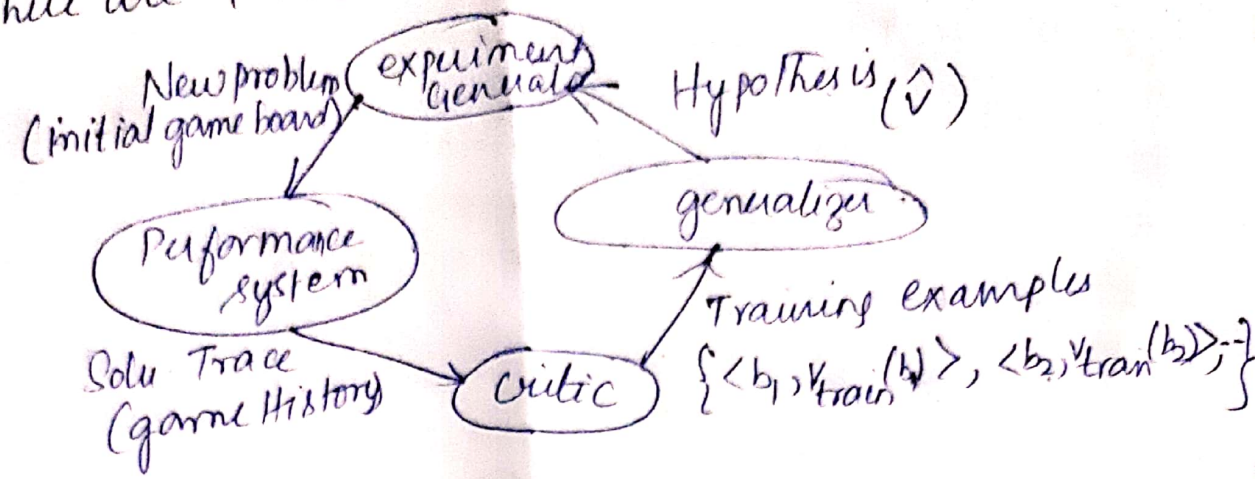
- use current wt. to calculate $\hat{V}(b)$.
- For each wt $w_i$, update

$$w_i \leftarrow w_i + \eta (V_{train}(b) - \hat{V}(b)) x_i$$

where $\eta$ = small const (0.1), which moderates size of wt. update

if $V_{train}(b) - \hat{V}(b)$ = { +ve ⇒ each wt. is ↑ in prop. to value of its feature
-ve ⇒ " " ↓ " "
0 ⇒ No wt. change.

→ The Final Design:-
There are 4 modules to describe the final Design viz:




(i) Performance sys:-
- Solves given performance Task
- In this ex., it is playing the checkers game, using Learned Target function.
- Takes new game as i/p & produces a part of its solu. i.e game History as o/p
- Strategy for performance sys:
  - Select its next move at each step determined by its evaluation func. $\hat{V}$.
  - As $\hat{V}$ ↑ses ⇒ performance ↑ses

(ii) Critic:-
- i/p: History / part of Trace of solu.
- o/p: Set of Training examples of the Target func.
- Each Training ex., is nothing but a game state 'b' and its estimate of Target func. $V_{train}(b)$.

(iii) Generalizer:-
- i/p: Training examples
- o/p: Hypothesis $\hat{V}$ = estimate of Target func.
- It takes specific training examples & generalises a general func. (Hypothesis) that covers these examples.
- In this ex., it has followed LMS algo. ∴ It uses Learned weights $w_0, w_1, \ldots, w_6$ to describe the o/p hypothesis $\hat{V}$.

(iv) Experiment Generator:-

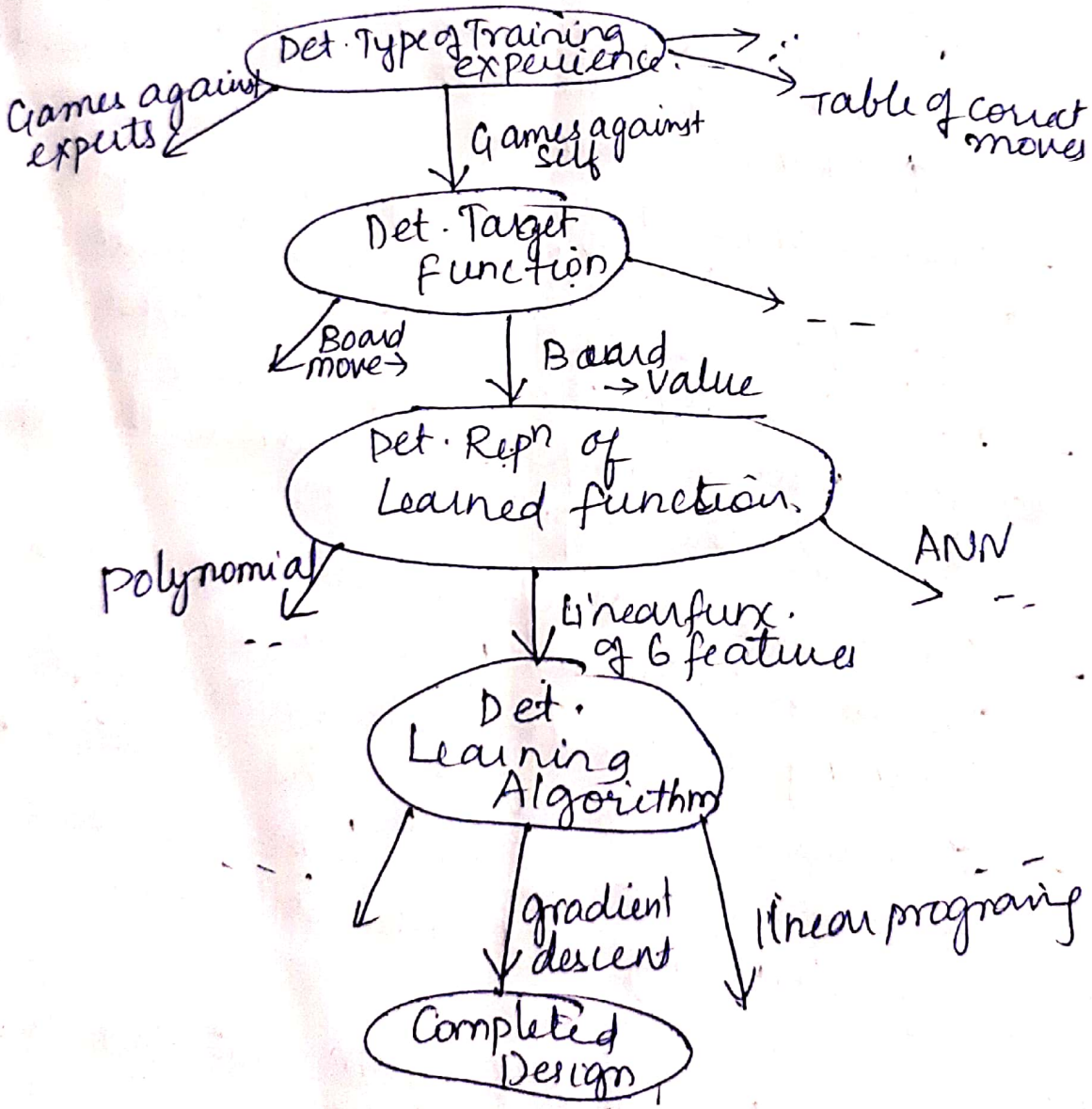
i/p:- Current Hypothesis

o/p:- new problem (i.e. initial board state) for the performance sys. to explore.

Its role is to pick new practice problems in order to maximize learning rate of overall system.

→ The seq. of design choices made for the checkers prog. is summarised below:

Assumptions:- — Single linear evaluation fun.
— 6 specific board features



# Perspectives & Issues in Machine Learning (ML):

## Perspectives :-

- Involves searching a v. large space of hypotheses to determine the one which fits the best on the observed data & any prior knowledge held by the learner.

example:- In checkers example, the hypothesis space is all the evaluation func. that can be rep. by some choice of values for weights $w_0$ to $w_6$.
The learner searches through this vast space & finds hypothesis that is best fit in available training example using LMS algorithm.

- There can be different hypothesis rep$^{n}$ using diff. algos. like linear func, logical descriptions decision Trees, Artificial Neural N/ws (ANN) etc.
- For each Hypothesis rep$^{n}$, there is a Corresponding learning algo. which helps to organise the search through the hypothesis space.
- perspective of Machine learning is taken as a "search problem" where in we characterise Learning methods by their "search strategy".
- This also helps to analyze the relationship b/w size of hypothesis space to be searched and no. of training examples available.

## Issues :-

- which Algorithm performs best for which type of problems & rep$^{n}$s.?
- Under what settings an algo. will converge to desired func? (given sufficient Training data).

- How much training data is sufficient?
- can prior knowledge be helpful even when it is only approximately correct?
- what is the best strategy for choosing a useful next training experience & how will it effect complexity of learning problem.
- what specific funcs. should the sys. attempt to learn? can this process itself be automated?
- How can a learner automatically improve its ability to rep. & learn target func.