INTRODUCTION

OVERVIEW

- **Well Posed learning Problems**
- **Choices in Designing the Checkers Learning Program**
- **Designing a Learning System**
- **Perspectives & Issues in machine Learning**

LEARNING

- **Learning is at the core of**
  - Understanding High Level Cognition
  - Performing knowledge intensive inferences
  - Building adaptive, intelligent systems
  - Dealing with messy, real world data
  - Analytics
- **Learning has multiple purposes**
  - Knowledge Acquisition
  - Integration of various knowledge sources to ensure robust behavior
  - Adaptation (human, systems)
  - Decision Making (Predictions)

WHY "LEARN" ?

Learning is used when:

- Human expertise does not exist. – Ex: Navigating on Mars
- Humans are unable to explain their expertise. – Ex. speech recognition
- Solution changes in time. – Ex. routing on a computer network
- Solution needs to be adapted to particular cases . – Ex. user biometrics

DEFINITION OF LEARNING

- **Definition:** A computer program which learns from experience is called a ***machine learning program*** or simply a ***learning program***. Such a program is sometimes also referred to as a ***learner.***

- **Tom Mitchell provides a more modern 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.**"

WELL POSED LEARNING PROBLEMS

- In general to have a Well Posed learning Problems, we must identify 3 features:
  - The class of tasks**(T)**
  - The measure of performance to be improved**(P)**
  - The source of experience**(E)**

EXAMPLES OF LEARNING

Handwriting recognition learning problem

- **Task T:**
  Recognising and classifying handwritten words within images
- **Performance P:**
  Percent of words correctly classified
- **Training experience E:**
  A dataset of handwritten words with given classifications

A robot driving learning problem

- **Task T:**
  Driving on highways using vision sensors
- **Performance measure P:**
  Average distance traveled before an error
- **Training experience E:**
  A sequence of images and steering commands recorded while observing a human driver

EXAMPLES OF LEARNING

A checkers learning problem

- **Task T:**
  Playing checkers
- **Performance measure P:**
  Percent of games won against opponents
- **Training experience E:**
  Playing practice games against itself /opponents/computer

Spam Mail detection learning problem

- **Task T:**
  To recognize and classify mails into 'spam' or 'not spam'.
- **Performance measure P:**
  Total percent of mails being correctly classified as 'spam' (or 'not spam') by the program.
- **Training experience E:**
  A set of mails with given labels ('spam' / 'not spam').

EXAMPLES OF LEARNING

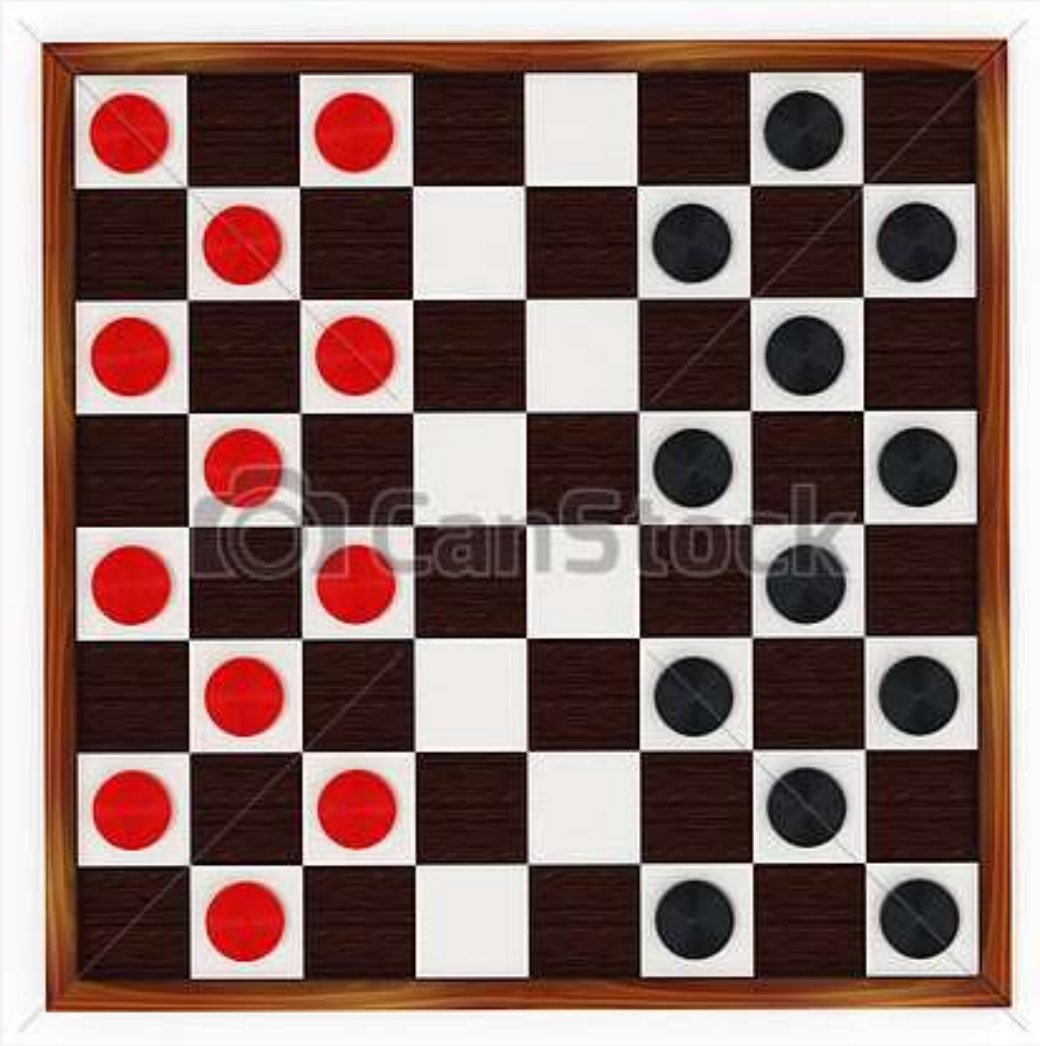
Learning to recognize faces

- **Task T:**
  Recognixe faces
- **Performance measure P:**
  Percent of correct recognitions
- **Training experience E:**
  Oppurtunity to make guesses and being told what the truth is.

Learning to find clusters in data

- **Task T:**
  Finding Clusters.
- **Performance measure P:**
  Compactness of groups detected.
- **Training experience E:**
  Analysis of a growing set of data

CHECKERS BOARD



A Glimpse of Checkers Game

CHECKERS
GAME INSTRUCTIONS

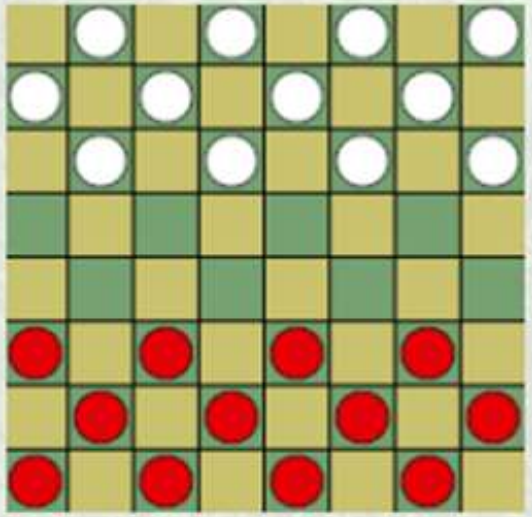
Capture all of your opponents pieces to win the game!

ITEMS NEEDED

- Two Players
- One checker board
- 12 pieces for each player

SET-UP

- Each player places their pieces on the 12 dark squares in the first three rows closest to him or her.
- Each of these three rows should have a total of 4 checkers.



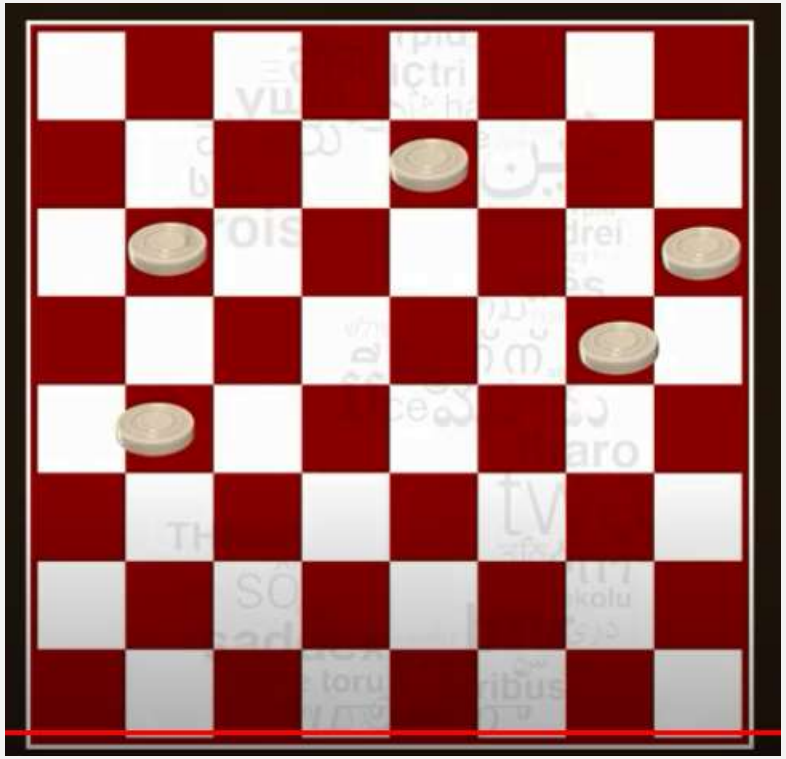
CHECKERS

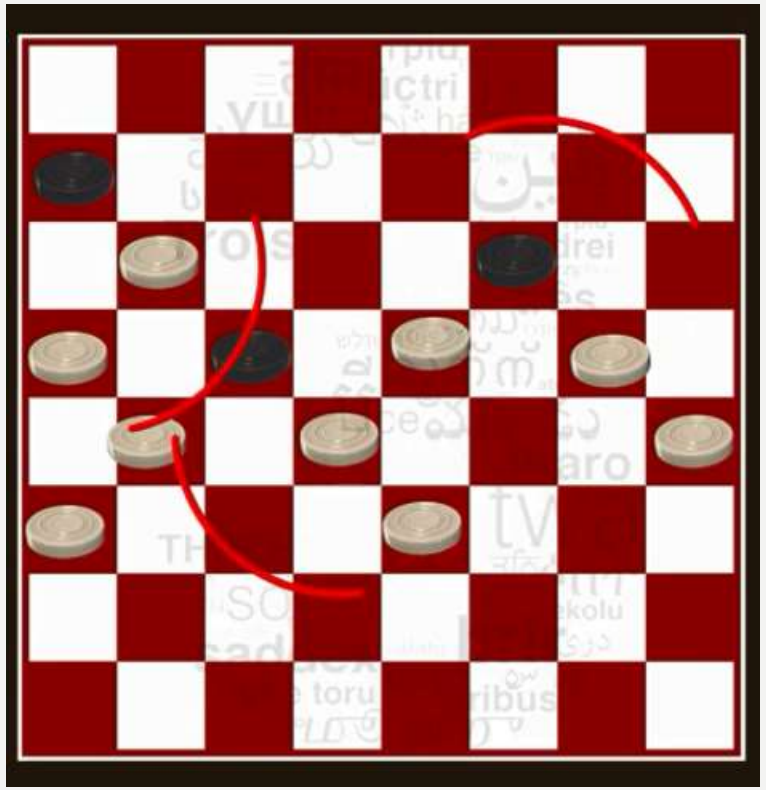
RULES:

1. Always move the checker diagonally forward towards their opponents.
2. To start, move the checker diagonally one place ahead.
3. To capture an opponent,"jump" over it by moving two diagonal spaces in the direction of the opponent’s checker.
4. As soon as the checker moves to the end row of the opponent side the checker becomes the king.
5. The king can move both the directions diagonally forward or backward.
6. When all the opponent checkers are captured or not able to move then, the player has won

WINNING STATES

When no opponents checker is left on the board.





When all opponent checkers are surrounded.

WELL POSED LEARNING PROBLEMS

- **Tom Mitchell provides a more modern 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.**"

- Let's consider a checkers learning problem:, ***a computer program that learns to play checkers might improve its performance as measured by its ability to win at the class of tasks involving playing checkers games, through experience obtained by playing games against itself***.

- In general, to have a well-defined learning problem,we must identify 3 features: the class of **T**asks, the measure of **P**erformance to be improved, the source of **E**xperience.

  - **Task T:** Playing checkers
  - **Performance measure P:** Percent of games won against opponents
  - **Training experience E:** Playing practice games against itself

DESIGNING A LEARNING SYSTEM

1. Choosing the Training Experience
2. Choosing the Target function
3. Choosing the representation for Target function
4. Choosing a Function Approximation Algorithm
    a. Estimating Training Values
    b. Adjusting the weights
5. Final Design

1. CHOOSING THE TRAINING EXPERIENCE

- The first design choice is to choose the type of training experience from which the system will learn.
- The type of training experience available can have a significant impact on success or failure of the learner.
- 3 Key attributes which impact on success or failure of the learner are:
  1) Whether the training experience provides **direct or indirect** feedback regarding the choices made by the performance system.
  2) The degree to which the **learner controls the sequence of training examples**.
  3) How well it represents the **distribution of examples** over which the final system performance P must be measured.

1. CHOOSING THE TRAINING EXPERIENCE

A) DIRECT OR INDIRECT FEEDBACK

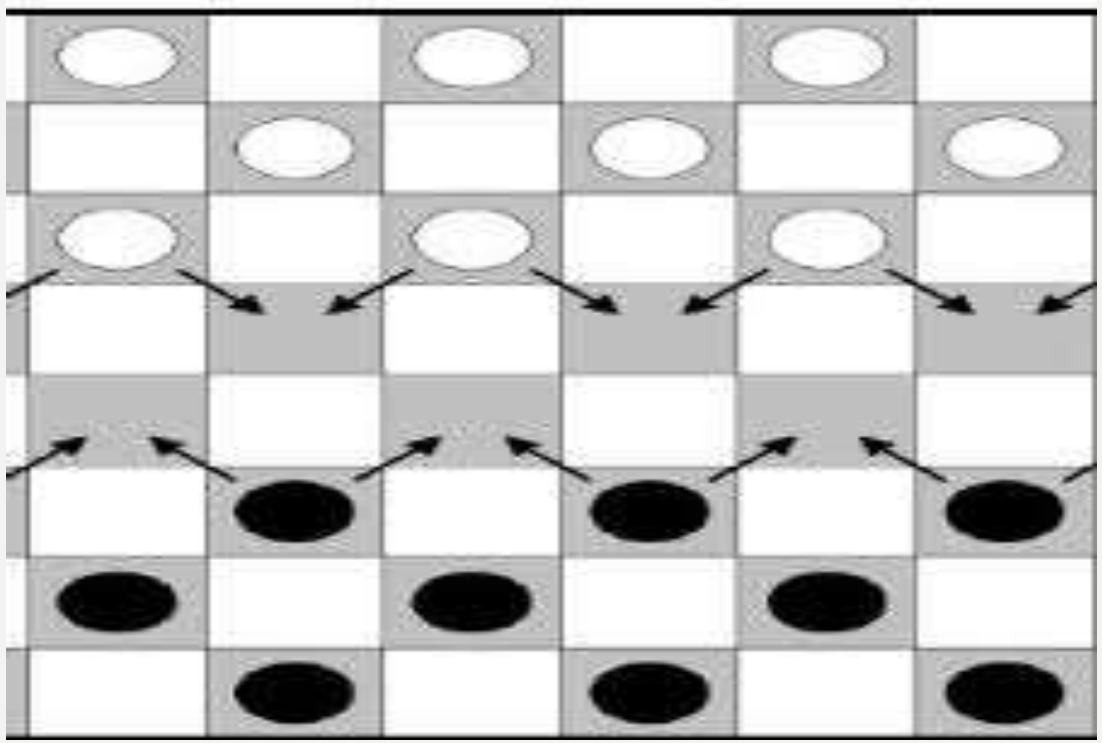
- One key attribute is whether the training experience provides **direct or indirect** feedback regarding the choices made by the performance system.

- Mostly, learning from direct training feedback is typically easier than learning from indirect feedback.

- In learning to play checkers, the system might learn from **direct training examples** consisting of **individual checkers board states** and **the correct move for each.**

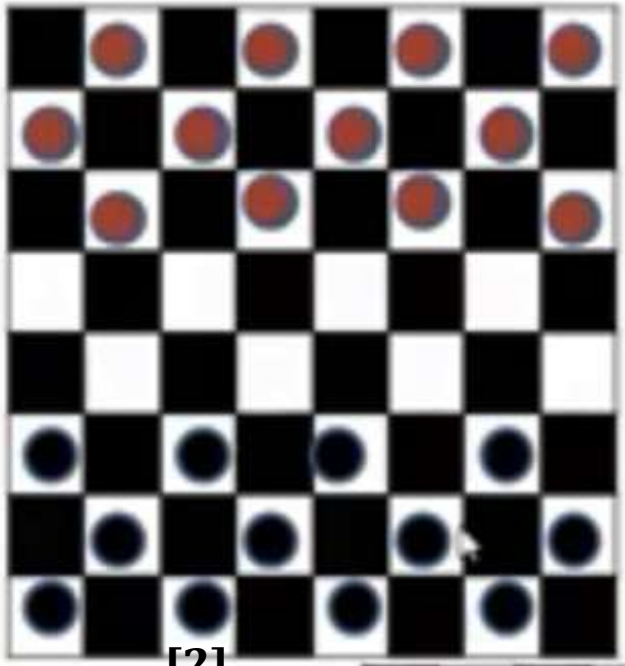
DIRECT TRAINING EXAMPLE

- For each board state, set of possible moves are given

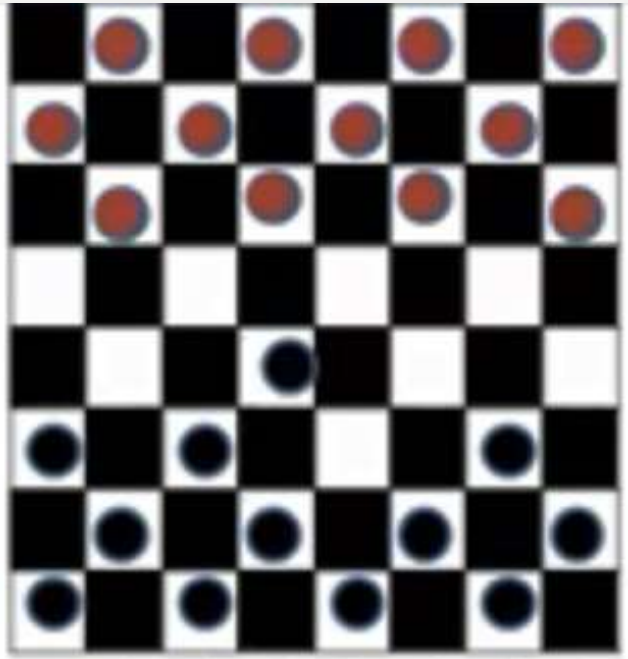


- In the above example, Black and White piecies facing each other can only make moves.

DIRECT TRAINING EXAMPLE

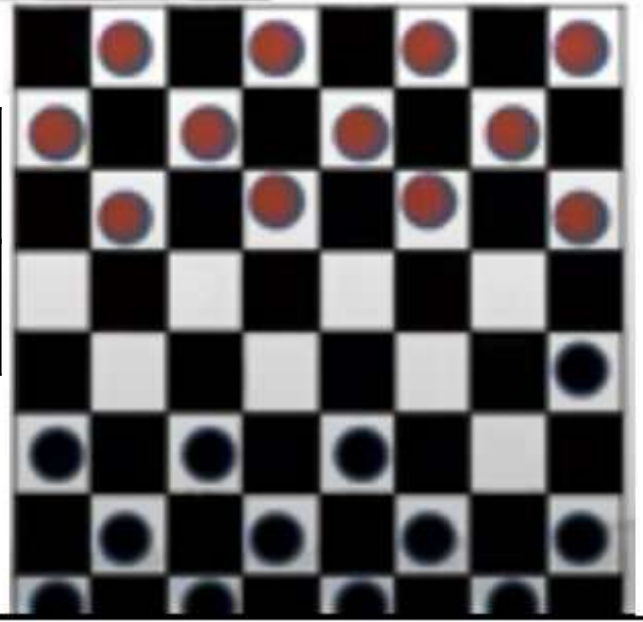


[2]

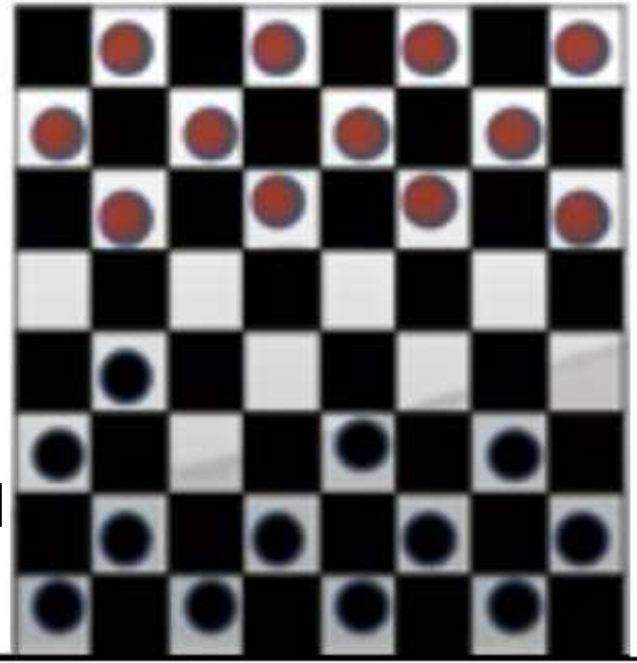


[3]

Present States	Possible States
2	3,7,...10



[10]



[7]

INDIRECT TRAINING EXAMPLE

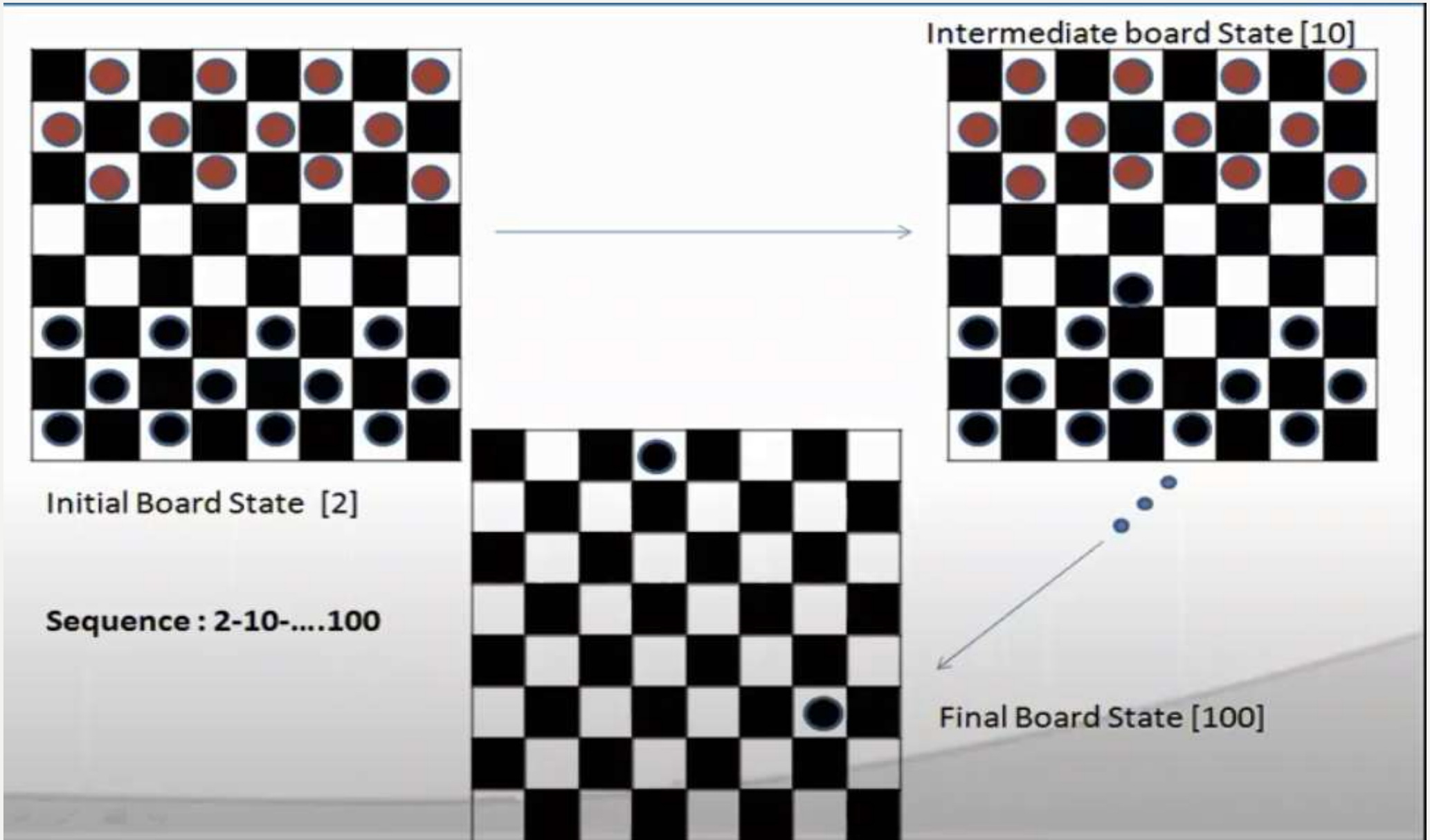
- ***Indirect training examples*** consisting of the ***move sequences*** and ***final outcomes*** of various games played.

- The information about the correctness of specific moves early in the game must be inferred indirectly from the fact that the game was eventually won or lost.

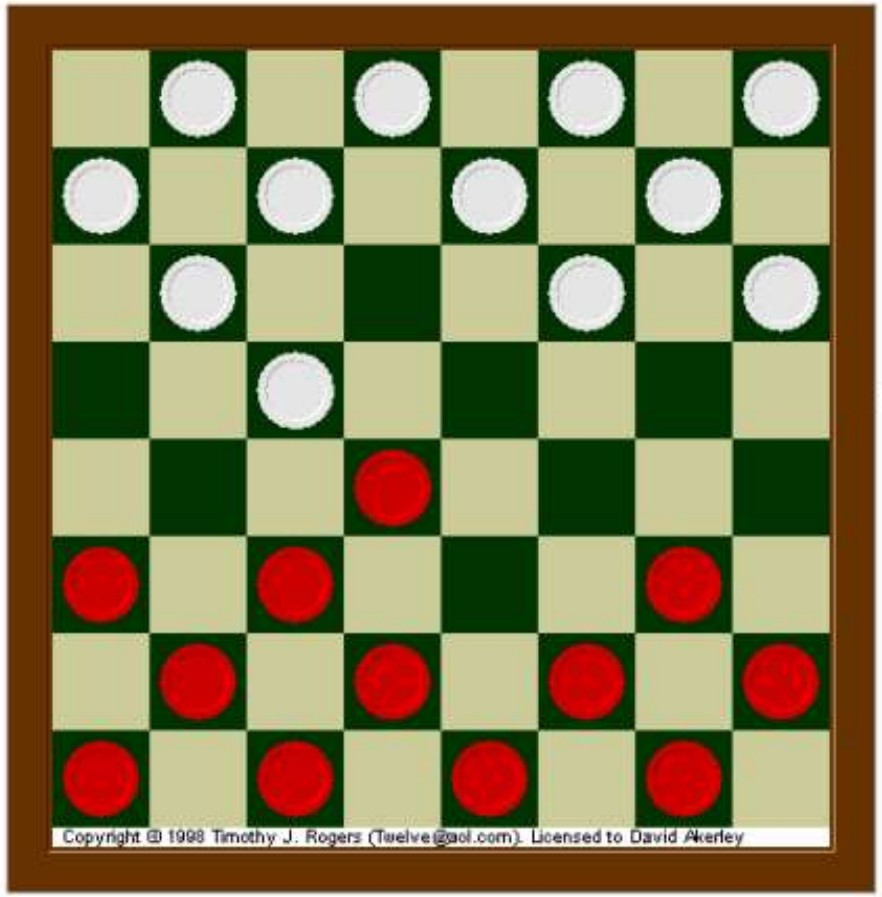
- Here the learner faces an additional problem of **credit assignment**, or determining the degree to which each move in the sequence deserves credit or blame for the final outcome.

- Credit assignment can be a particularly difficult problem because the game can be lost even when early moves are optimal, if these are followed later by poor moves.

INDIRECT TRAINING EXAMPLE



INDIRECT TRAINING EXAMPLE



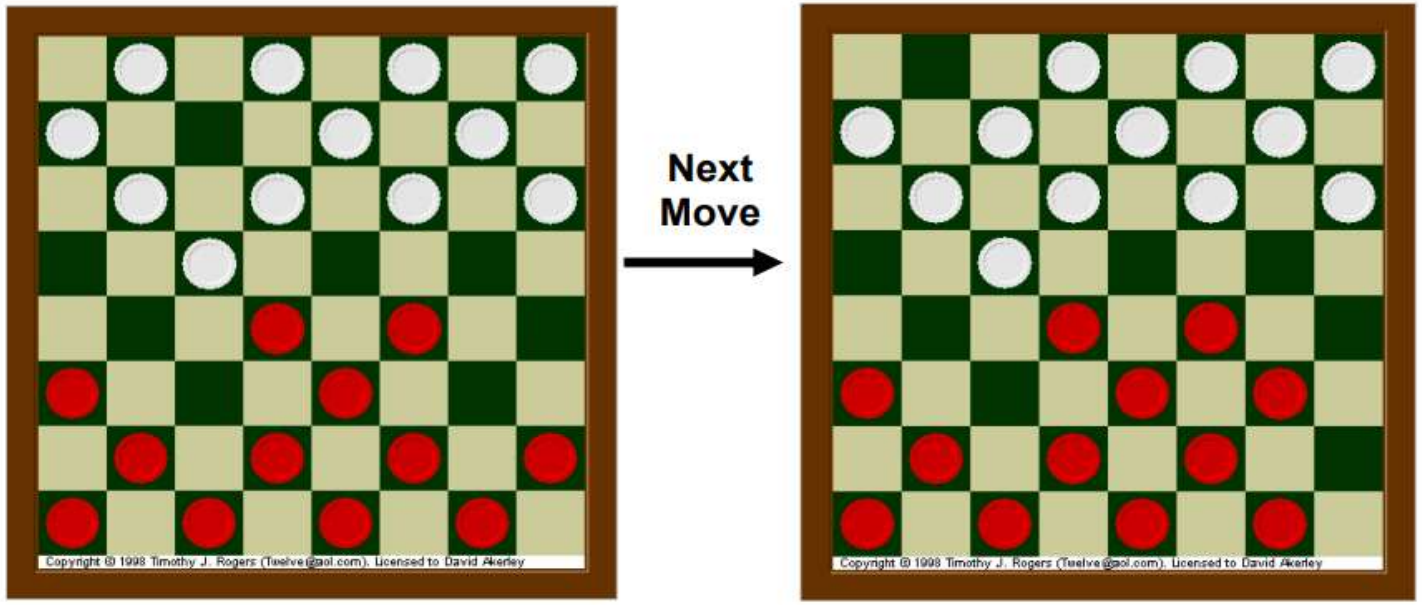
Next Move

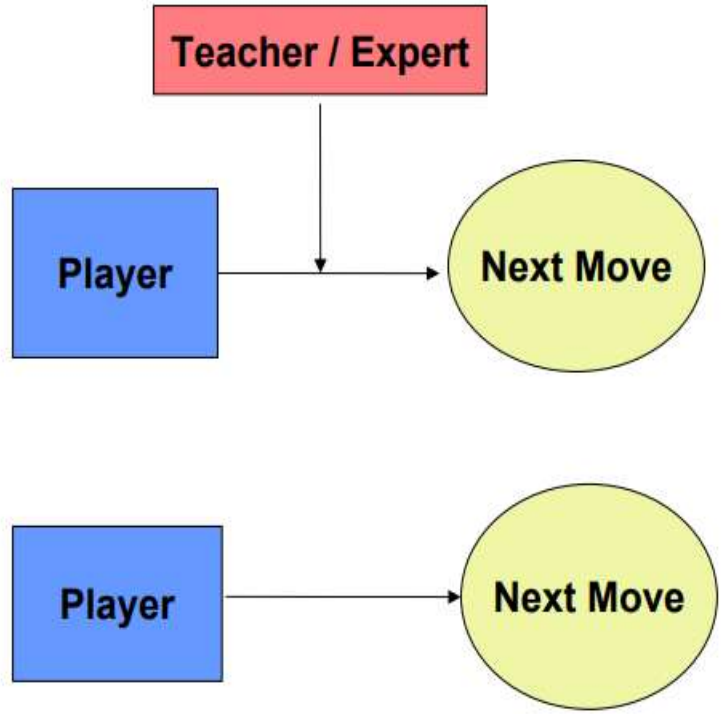
Win / Loss

1. CHOOSING THE TRAINING EXPERIENCE

B) DEGREE TO WHICH THE LEARNER CONTROLS THE SEQUENCE OF TRAINING EXAMPLES

- The learner might depends on the ***teacher*** to select informative board states and to provide the correct move for each.;
- The learner might itself propose board states that it finds particularly confusing and ask the teacher for the correct move.
- The learner can be completely on its own with **no teacher present**





1. CHOOSING THE TRAINING EXPERIENCE

C) DISTRIBUTION OF EXAMPLES

- A third important attribute of the training experience is how well it represents the distribution of examples over which the final system performance P must be measured.

- In checkers learning scenario, the performance metric P is the percent of games the system wins in the world tournament.

- If its training experience E consists only of games played against itself, there is a danger that this training experience might not be fully representative of the distribution of situations over which it will later be tested.

- It is necessary to learn from a distribution of examples that is different from those on which the final system will be evaluated.

SUMMARY (SO FAR)

- Decisions made
  - T: playing checkers
  - P: percent of games won in the world tournament
  - E: games played against itself/opponent/computer

  --DESIGNING A LEARNING SYSTEM

  1. Choosing the Training Experience

     a) Direct or Indirect feedback

     b) Degree to which the learner controls the sequence of training examples

     c) Distribution of examples

- Decisions yet to be made
  - The exact type of knowledge to be learned
  - A representation for this target knowledge
  - A learning mechanism

2. CHOOSING THE TARGET FUNCTION

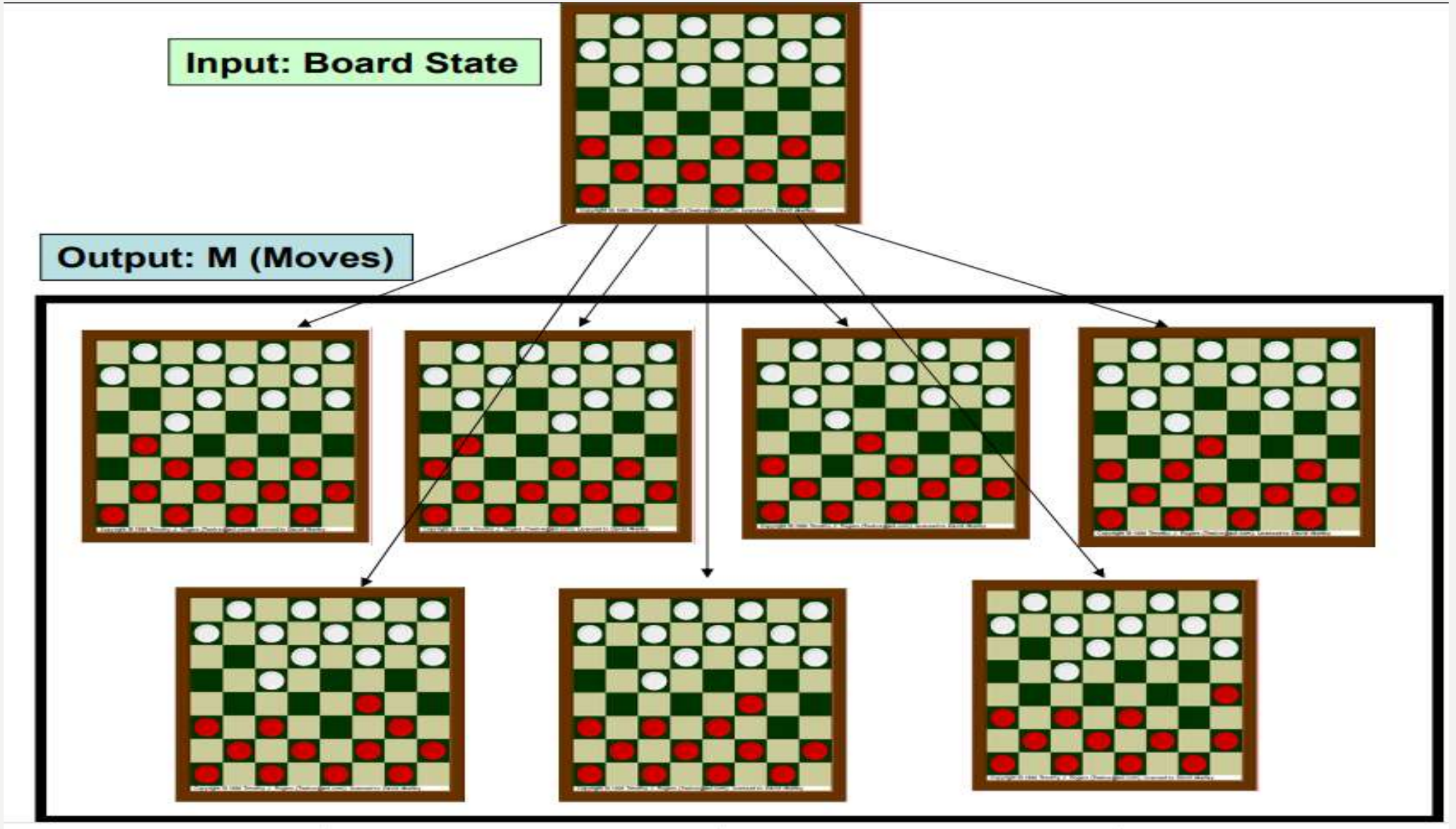
- The next design choice is to determine exactly what type of knowledge will be learned and how this will be used by the performance program.

- The program needs only to learn how to choose the best move from among these legal moves.

- Let us call this function ***ChooseMove*** and use the notation **ChooseMove : B → M** to indicate that this function accepts as input any board from the set of legal board states B and produces as output some move from the set of legal moves M.

2. CHOOSING THE TARGET FUNCTION

- **ChooseMove : B → M**



2. CHOOSING THE TARGET FUNCTION

Similar to

```
Fun( )
{
Y=f(x)
}
```

We can relate,

ChooseMove : B → M

Function to **choosemove**

x to board state

Y to a leagal move

i.e., **x** is similar to **B** and **Y** is similar to **M**

M consists of (left, right, left jump, right jump)

B

Board State (B)	Legal Move (M)
[2]	3,7,...10
[5]	11,17,..6
....	
[33]	17,44,...8

M

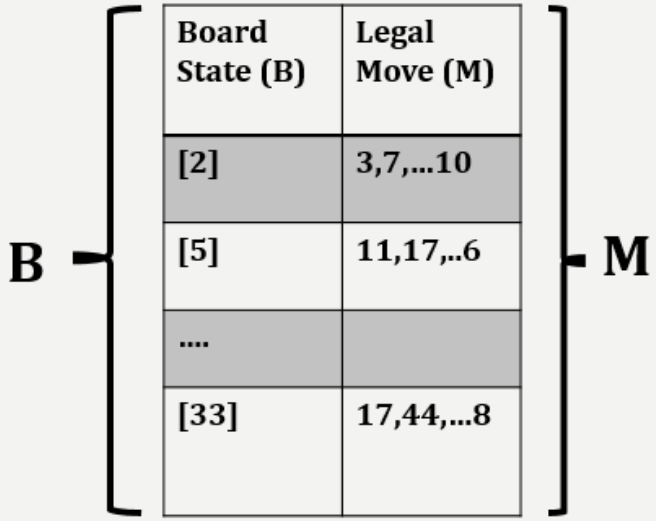
2. CHOOSING THE TARGET FUNCTION

- Although ChooseMove is an obvious choice for the target function in our example, this function will turn out to be very difficult to learn given the kind of indirect training experience available to our system.

- An **alternative target function** and one that will turn out to be easier to learn in this setting-is an **evaluation function** that assigns a numerical score to any given board state.

2. CHOOSING THE TARGET FUNCTION

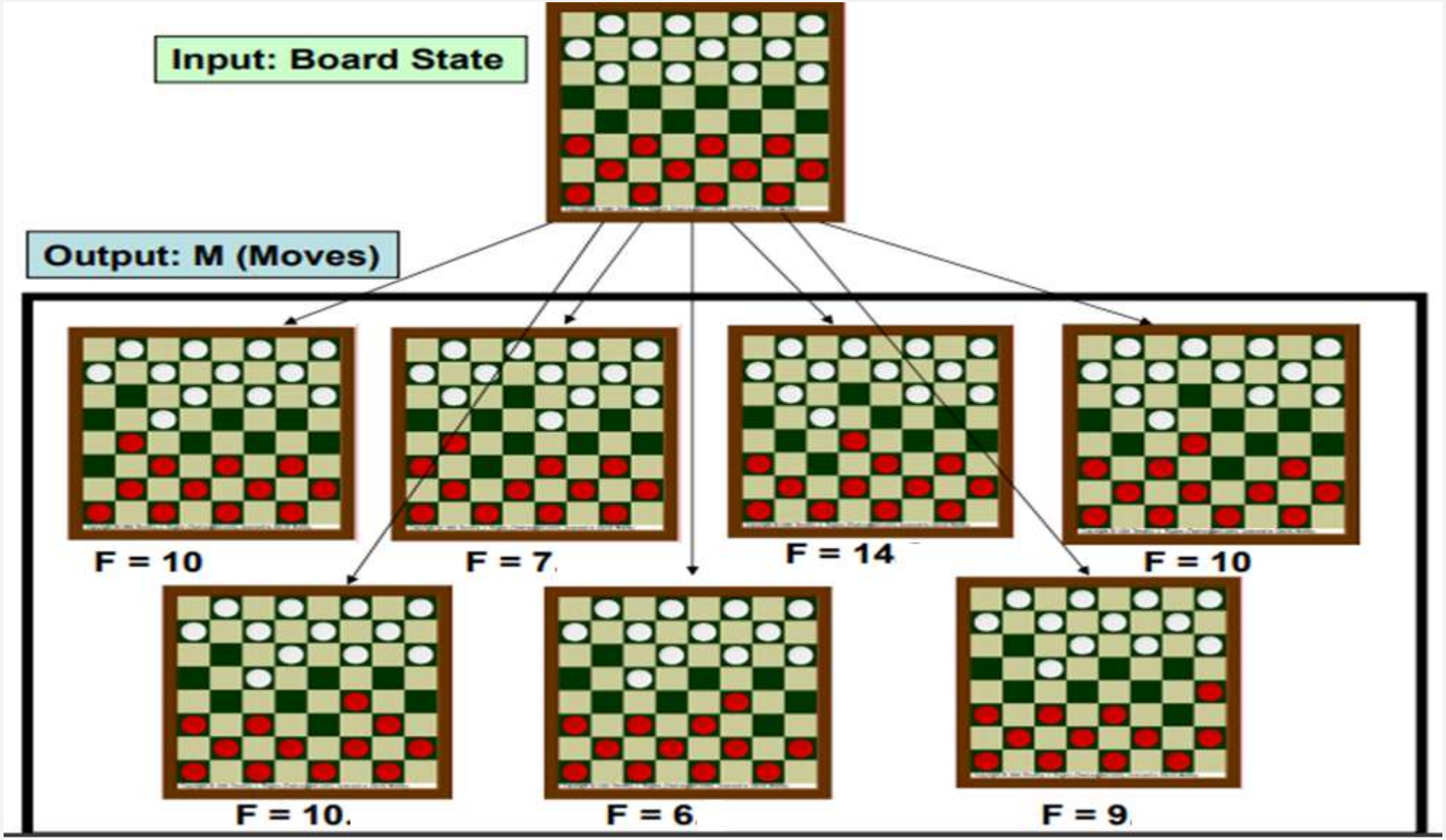
- Let us call this **target function V** and again use the notation **V : B →R** to denote that V maps any legal board state from the set B to some real value (we use R to denote the set of real numbers).



- If the system can successfully learn such a target function V, then it can easily use it to select the best move from any current board position.

2. CHOOSING THE TARGET FUNCTION

- **V : B → R or F:B → R**



2. CHOOSING THE TARGET FUNCTION

- Let us therefore define the target value ***V(b)*** for an arbitrary board state ***b*** in *B,* ***as*** follows:

  1. if ***b*** is a final board state that is won, then ***V(b)*** = 100

  2. if b is a final board state that is lost, then ***V(b)*** = -100

  3. if b is a final board state that is drawn, then ***V(b)*** = *0*:

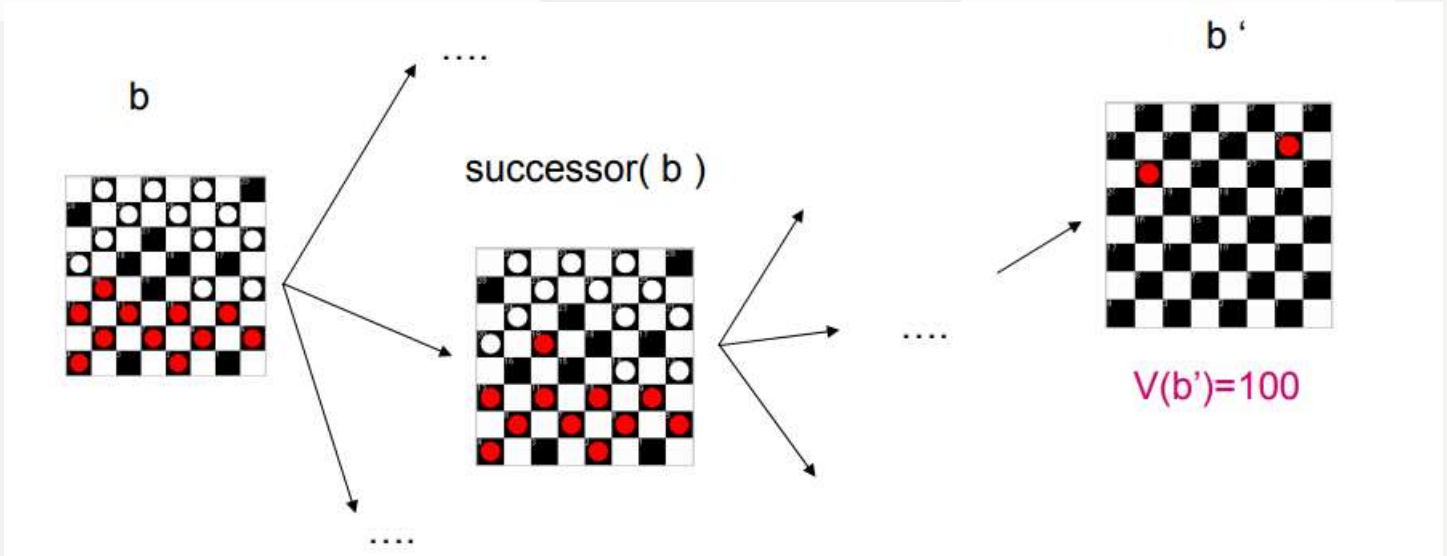
  4. if b is a not a final state in the game, then **V(b)** = V(b'), where b' is the best final board state that can be achieved starting from b and playing optimally until the end of the game.

- Except for the trivial cases (cases 1-3) in which the game has already ended, determining the value of V(b) for a particular board state requires (case 4) searching ahead for the optimal line of play, all the way to the end of the game.

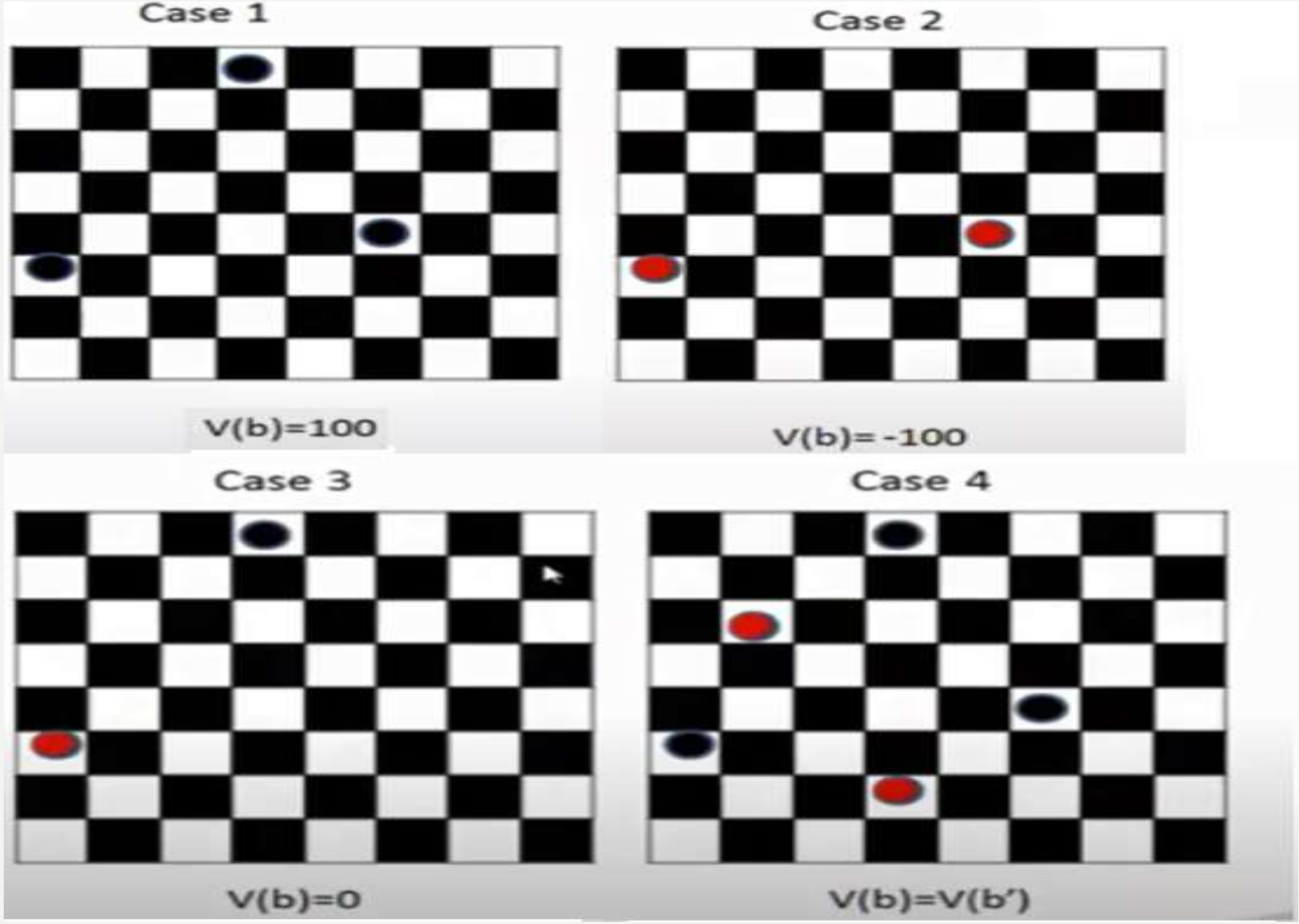
- Because (case 4) this definition is not efficiently computable by our checkers playing program, we say that it is a ***nonoperational*** definition.

2. CHOOSING THE TARGET FUNCTION

- Case-4 can be accomplished by generating the successor board state produced by every legal move, then using V to choose the best successor state and therefore the best legal move.



2. CHOOSING THE TARGET FUNCTION



2. CHOOSING THE TARGET FUNCTION

- The goal of learning in this case is to discover an operational description of V ; that is, a description that can be used by the checkers-playing program to evaluate states and select moves within realistic time bounds.

- We often expect learning algorithms to acquire only some ***approximation*** to the target function, and for this reason the process of learning the target function is often called ***function approximation.***

$\hat{V}$ -function learned by our program

V- ideal target function

3. CHOOSING A REPRESENTATION FOR THE TARGET FUNCTION

FEATURES

- Each feature, or column, represents a measurable piece of data that can be used for analysis.
- **Example:** Textbook
- **Features of Textbook:** Name, Publisher, Author, Edition, Year of Publication, Price…
- Consider a library Database, TexBook table like,

Sl. No.	Name	Publisher	Author	Edition	Year of Publication	Price

3. CHOOSING A REPRESENTATION FOR THE TARGET FUNCTION

Consider an example of -- **Choosing best student of the year in a class**
For parameters (features) are MarksObtained, CulturalActivities, Attendance etc.,

x1= MarksObtained
x2=CulturalActivities
x3=Attendance

Importance (weight) given to these features varies.
BestStudent = □x1 + □x2 + □x3

x1 importance (weight) given is 70 = w1
x2 weight assigned is 20 = w2
x3 weight assigned is 10 = w3

BestStudent = w1 x1+ w2 x2+ w3 x3

3. CHOOSING A REPRESENTATION FOR THE TARGET FUNCTION

Consider an example of -- **Choosing best student of the year in a class**
For parameters (features) are MarksObtained, CulturalActivities, Attendance etc.,

x1= MarksObtained
x2=CulturalActivities
x3=Attendance

Importance (weight) given to these features varies.
BestStudent = □x1 + □x2 + □x3

x1 importance (weight) given is 70 = w1
x2 weight assigned is 20 = w2
x3 weight assigned is 10 = w3

BestStudent = w1 x1+ w2 x2+ w3 x3

FEATURES

[2]

1. No. of Black pieces.
2. No. of Red pieces.
3. No. of Black kings.
4. No. of Red kings.
5. No. of Black pieces threatened by Red pieces.
6. No. of Red pieces threatened by Black pieces.

Board. No.	No. of Black pieces	No. of Red pieces	No. of Black kings	No. of Red kings	No. of Black pieces threatened by Red pieces	No. of Red pieces threatened by Black pieces
2	12	12	0	0	0	0

FEATURES

[98]

Board. No.	No. of Black pieces	No. of Red pieces	No. of Black kings	No. of Red kings	No. of Black pieces threatened by Red pieces	No. of Red pieces threatened by Black pieces
98	3	0	1	0	0	0

3. CHOOSING A REPRESENTATION FOR THE TARGET FUNCTION

- For any given board state, the function $\hat{V}$ will be calculated as a linear combination of the following board features:
  - x1: No. of Black pieces.
  - x2: No. of Red pieces.
  - x3: No. of Black kings.
  - x4: No. of Red kings.
  - x5: No. of Black pieces threatened by Red pieces.
  - x6: No. of Red pieces threatened by Black pieces.
- Thus, our learning program will represent $\hat{V}(b)$ as a linear function of the form

$$\hat{V}(b) = w_0 + w_1 x_1 + w_2 x_2 + w_3 x_3 + w_4 x_4 + w_5 x_5 + w_6 x_6$$

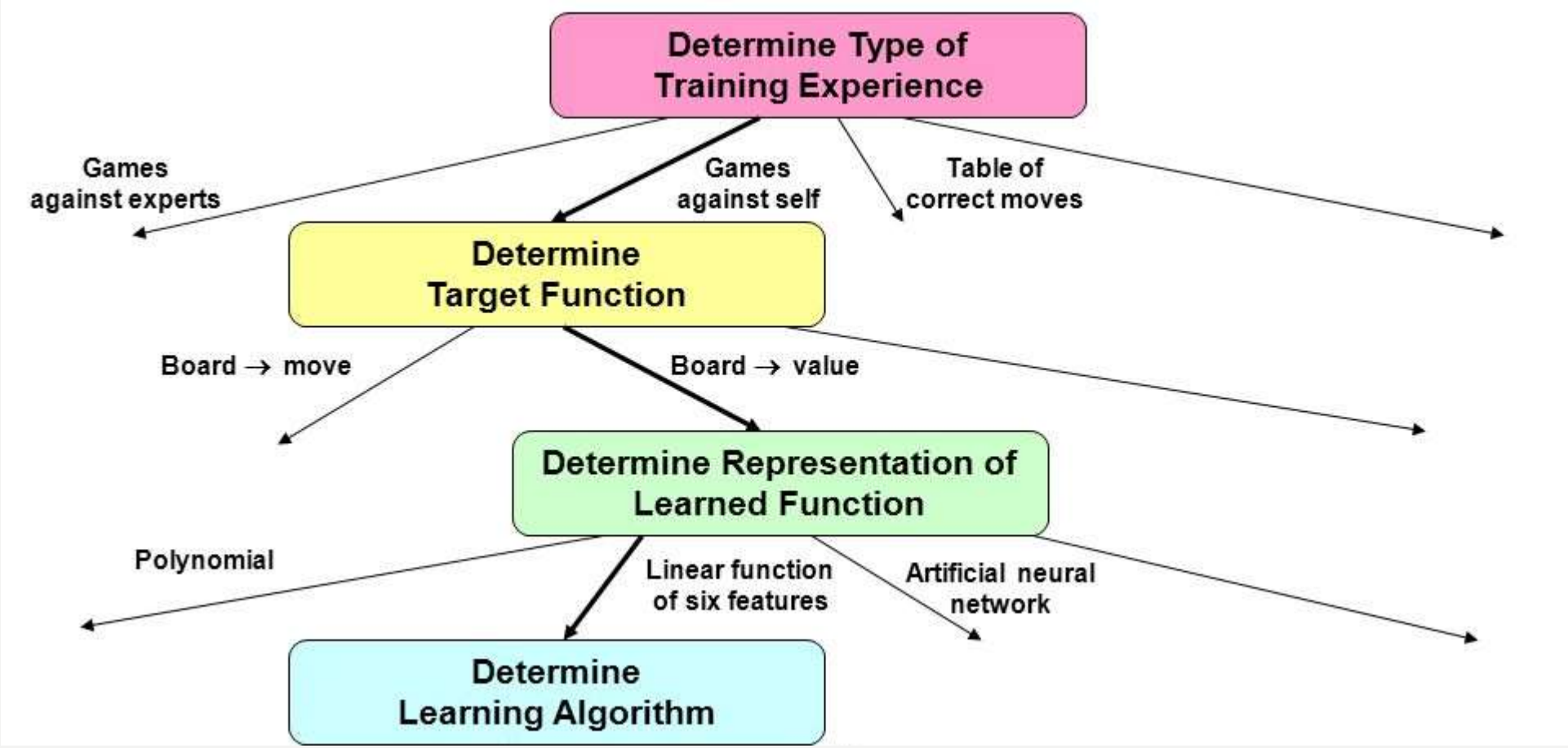
  - where $\mathbf{w_0}$ through $\mathbf{w_6}$ are numerical coefficients, or weights
  - $\mathbf{w_0}$ an additive constant to the board value.

Partial design of a 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:* $V : Board \rightarrow \mathbb{R}$
- *Target function representation*

$$\hat{V}(b) = w_0 + w_1 x_1 + w_2 x_2 + w_3 x_3 + w_4 x_4 + w_5 x_5 + w_6 x_6$$

SO FAR..

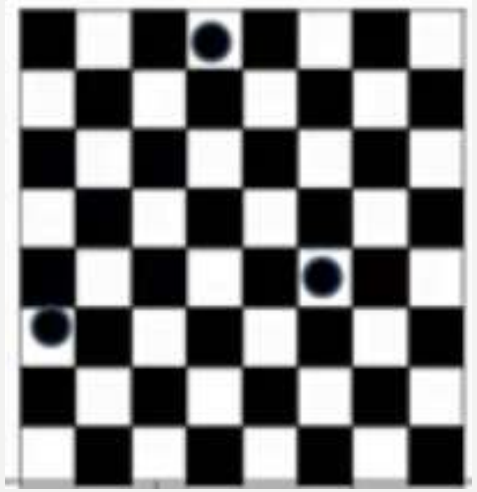

Determine Type of Training Experience
Games against experts
Games against self
Table of correct moves
Determine Target Function
Board → move
Board → value
Determine Representation of Learned Function
Polynomial
Linear function of six features
Artificial neural network
Determine Learning Algorithm

4. CHOOSING A FUNCTION APPROXIMATION ALGORITHM

- In order to learn the target function $\hat{V}$ we require a set of training examples, each describing a specific board state **b** and the training value $V_{train}(b)$ for **b.**
- In other words, each training example is an ordered pair of the form $(b, V_{train}(b))$.
- For instance, the following training example describes a board state b in which black has won the game (note x2 = 0 indicates that red has no remaining pieces) and for which the target function value $V_{train}(b)$ is therefore +100.

((x1=3, x2=0, x3=1, x4=0, x5=0, x6=0), +100)

Board. No.	No. of Black pieces	No. of Red pieces	No. of Black kings	No. of Red kings	No. of Black pieces threatened by Red pieces	No. of Red pieces threatened by Black pieces
98	3	0	1	0	0	0



4. CHOOSING A FUNCTION APPROXIMATION ALGORITHM

Function Approximation Procedure:

1. Derive training examples from the indirect training experience available to the Learner

2. Adjusts the weights w_i to best fit these training examples

4. CHOOSING A FUNCTION APPROXIMATION ALGORITHM

1. Estimating training values

- A simple approach for estimating training values for intermediate board states is to assign the training value of $V_{train}(b)$ for any intermediate board state b to be $\hat{V}(Successor(b))$
- Where , $\hat{V}$ is the learner's current approximation to V
- Successor(b) denotes the next board state following b for which it is again the program's turn to move

Rule for estimating training values

$$\mathbf{V_{train}(b) \leftarrow \hat{V}\ (Successor(b))}$$

4. CHOOSING A FUNCTION APPROXIMATION ALGORITHM

2. Adjusting the weights

- Specify the learning algorithm for choosing the weights w_i to best fit the set of training examples $\{(b, V_{train}(b))\}$
- A first step is to define what we mean by the bestfit to the training data.
- One common approach is to define the best hypothesis, or set of weights, as that which minimizes the squared error E between the training values and the values predicted by the hypothesis.

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

- Several algorithms are known for finding weights of a linear function that minimize E.

4. CHOOSING A FUNCTION APPROXIMATION ALGORITHM

- In our case, we require an ***algorithm*** that will incrementally refine the weights as new training examples become available and that will be robust to errors in these estimated training values

- One such algorithm is called the ***least mean squares***, or ***LMS training rule***. For each observed training example it adjusts the weights a small amount in the direction that reduces the error on this training example

- ***LMS weight update rule*** :-

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

  - Use the current weights to calculate $\hat{v}(b)$
  - For each weight w_i, update it as
    - $w_i \leftarrow w_i + \eta\,(Vtrain(b) - \hat{v}(b))\,x_i$

Here η is a small constant (e.g., 0.1) that moderates the size of the weight update.

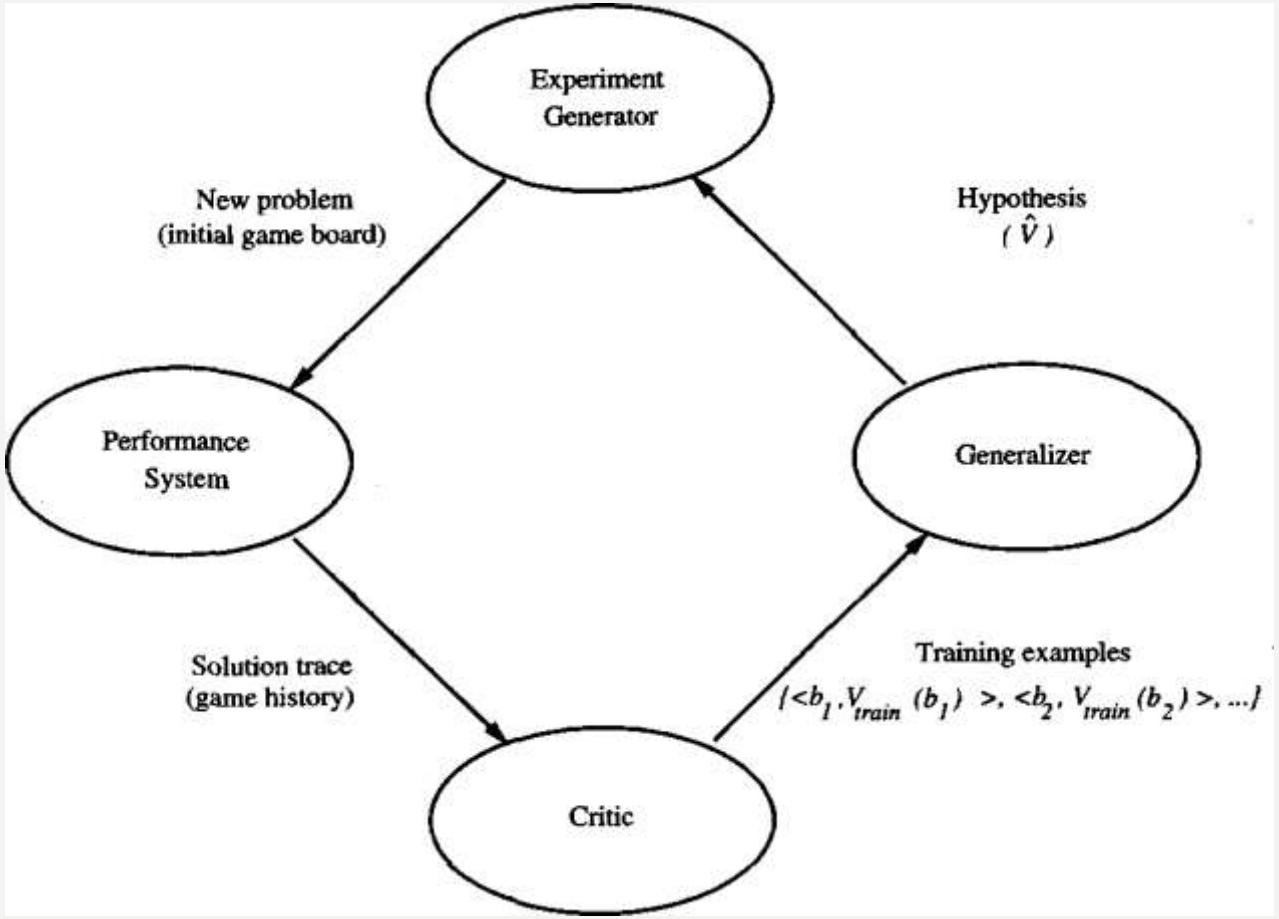
4. CHOOSING A FUNCTION APPROXIMATION ALGORITHM

Weight Update Rule:

- When the **error** ($V_{train}(b)$- $\hat{V}(b)$) is **zero**, **no weights are changed.**
- When **error** is **positive** (i.e., when $\hat{V}(b)$ is too low), then **each weight is increased in proportion** to the value of its corresponding feature. This will raise the value of $\hat{V}(b)$, reducing the error.
- If the value of some feature $\mathbf{x_i}$ **is zero**, then its **weight is not altered** regardless of the error, so that the only weights updated are those whose features actually occur on the training example board.

5. THE FINAL DESIGN

- The final design of checkers learning system can be described by four distinct program modules that represent the central components in many learning systems



5. THE FINAL DESIGN

1.The Performance System:

- It is the module that must solve the given performance task by using the learned target function(s).
- It takes an instance of a new problem **(new game)** as input and produces a trace of its solution **(game history)** as output.
- In checkers game, the strategy used by the Performance System to select its next move at each step is determined by the learned $\hat{V}$ evaluation function.
- Therefore, we expect its performance to improve as this evaluation function becomes increasingly accurate.

2.The Critic

- It takes as input the **history** or **trace of the game** and produces as output a set of training examples of the target function. **(b, V_{train})**
- As shown in the diagram, each training example in this case corresponds to some game state in the trace, along with an estimate V_{train} of the target function value for this example.

5. THE FINAL DESIGN

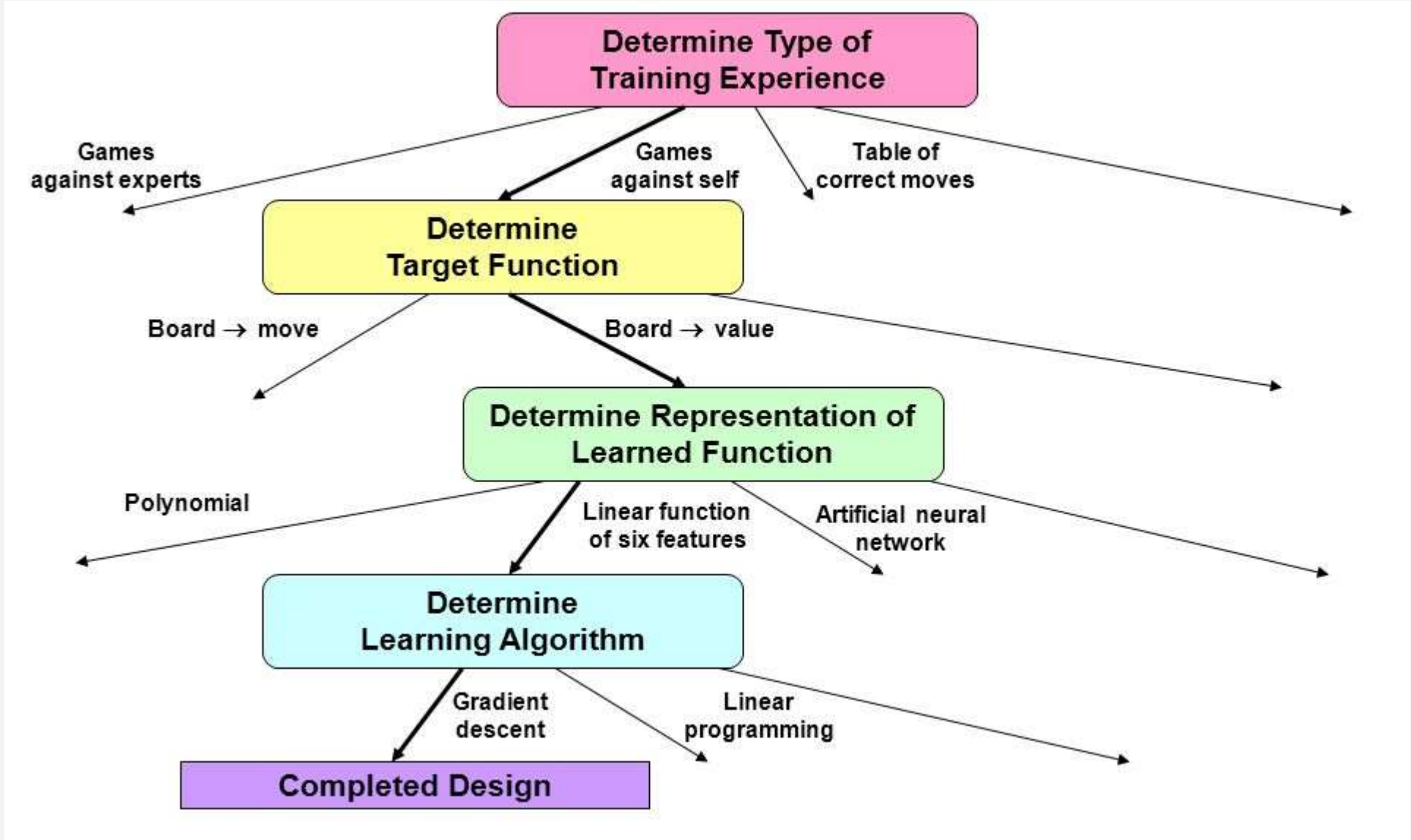
3.The Generalizer

- This module takes as input the training examples and produces an output hypothesis that is its estimate of the target function.
- It generalizes from the specific training examples, hypothesizing a general function that covers these examples and other cases beyond the training examples.
- In our example, the **Generalizer corresponds to the LMS algorithm**, and the output **hypothesis is the function $\hat{V}$** described by the learned weights $w_0, \ldots, w_6$.

4.The Experiment Generator

- This module takes as input the **current hypothesis** and outputs a **new problem** (i.e., initial board state) for the Performance System to explore.
- Its role is to pick new practice problems that will maximize the learning rate of the overall system.
- In our example, the Experiment Generator always proposes the same initial game board to begin a new game.

CHOICES IN DESIGNING THE CHECKERS LEARNING PROGRAM.



ISSUES IN MACHINE LEARNING

- 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?

ISSUES IN MACHINE LEARNING

- 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? Put another way, 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?

SUMMARY OF CHAPTER-1

What is the Learning Problem?

Learning = Improving with experience at some task

- Improve over task T,
- with respect to performance measure P,
- based on experience E.

E.g., Learn to play checkers

- T: Play checkers
- P: % of games won in world tournament
- E: opportunity to play against self

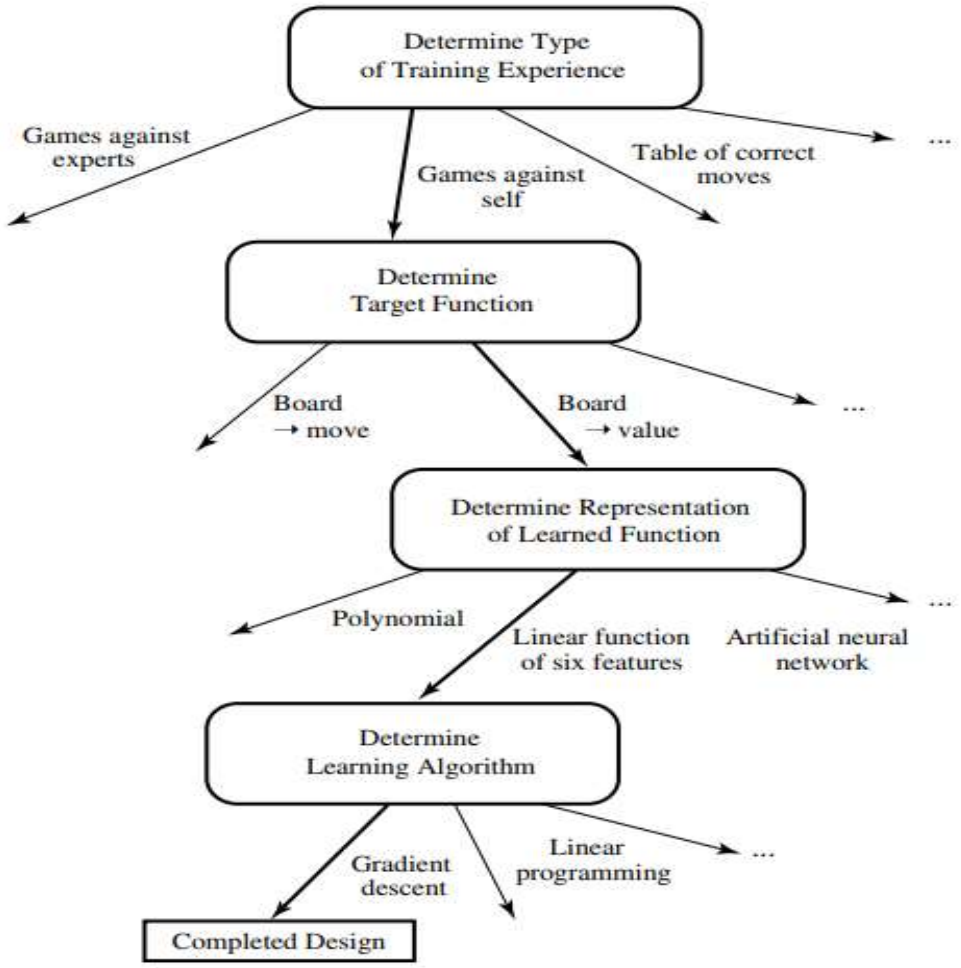
SUMMARY OF CHAPTER-1

Learning to Play Checkers

- T: Play checkers
- P: Percent of games won in world tournament
- What experience?
- What exactly should be learned?
- How shall it be represented?
- What specific algorithm to learn it?

SUMMARY OF CHAPTER-1

Design Choices



SUMMARY OF CHAPTER-1

Some Issues in Machine Learning

- What algorithms can approximate functions well (and when)?
- How does number of training examples influence accuracy?
- How does complexity of hypothesis representation impact it?
- How does noisy data influence accuracy?

- What are the theoretical limits of learnability?
- How can prior knowledge of learner help?
- What clues can we get from biological learning systems?
- How can systems alter their own representations?

QUESTIONS ON CHAPTER-1

1. **Define Machine Learning. Explain with examples why machine learning is important.**
2. **Discuss some applications of machine learning with examples.**
3. **Explain how some disciplines have influenced the machine learning.**
4. **What is well- posed learning problems.**
5. **Describe the following problems with respect to Tasks, Performance and Experience:**
    a. **A Checkers learning problem**
    b. **A Handwritten recognition learning problem**
    c. **A Robot driving learning problem**
6. **Explain the steps in designing a learning systems in detail.**
7. **Explain different perspective and issues in machine learning.**