UNIT - 3

Introduction

Introduction

- Ever since computers were invented, we have wondered whether they might be made to learn.
- If we could understand how to program them to learn-to improve automatically with experience-the impact would be dramatic.

Imagine computers learning from the following:

- Medical records which treatments are most effective for new diseases
- Houses learning from experience to optimize energy costs based on the particular usage patterns of their occupants.
- Personal software assistants learning the evolving interests of their users in order to highlight especially relevant stories from the online morning newspaper

A successful understanding of how machines learn will open several applications and might also open up the abilities and disabilities of human learning.

Few Successful Applications of Machine Learning

- Learning to recognize spoken words
- Learning to drive an autonomous vehicle
- Learning to classify new astronomical structures
- Learning to play world-class backgammon
- Learning to play chess
- Learning to recognize face
- Learning to generate contents

Importance of Machine Learning

- Some tasks cannot be defined well, except by examples (e.g., recognizing people).
- Relationships and correlations can be hidden within large amounts of data. Machine Learning/Data Mining may be able to find these relationships.
- Human designers often produce machines that do not work as well as desired in the environments in which they are used.
- The amount of knowledge available about certain tasks might be too large for explicit encoding by humans (e.g., medical diagnostic).
- Environments change over time.
- New knowledge about tasks is constantly being discovered by humans. It may be difficult to continuously re-design systems “by hand”.

Disciplines that influence ML

- ***Statistics:*** How best to use samples drawn from unknown probability distributions to help decide from which distribution some new sample is drawn? Characterization of errors, statistical tests

- ***Brain Models***: Non-linear elements with weighted inputs (Artificial Neural Networks) have been suggested as simple models of biological neurons.

- ***Adaptive Control Theory***: How to deal with controlling a process having unknown parameters that must be estimated during operation?

- ***Psychology***: How to model human performance on various learning tasks?

- ***Artificial Intelligence***: How to write algorithms to acquire the knowledge humans are able to acquire, at least, as well as humans?

- ***Evolutionary Models***: How to model certain aspects of biological evolution to improve the performance of computer programs?

Few other disciplines

- Bayesian methods
- Computational complexity theory
- Control theory
- Information theory
- Philosophy

Machine learning: 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.

Need of learning

- Human expertise does not exist (navigating on Mars)
- Humans are unable to explain their expertise (speech recognition)
- Solution changes in time (routing on a computer network)
- Solution needs to be adapted to particular cases (user biometrics)

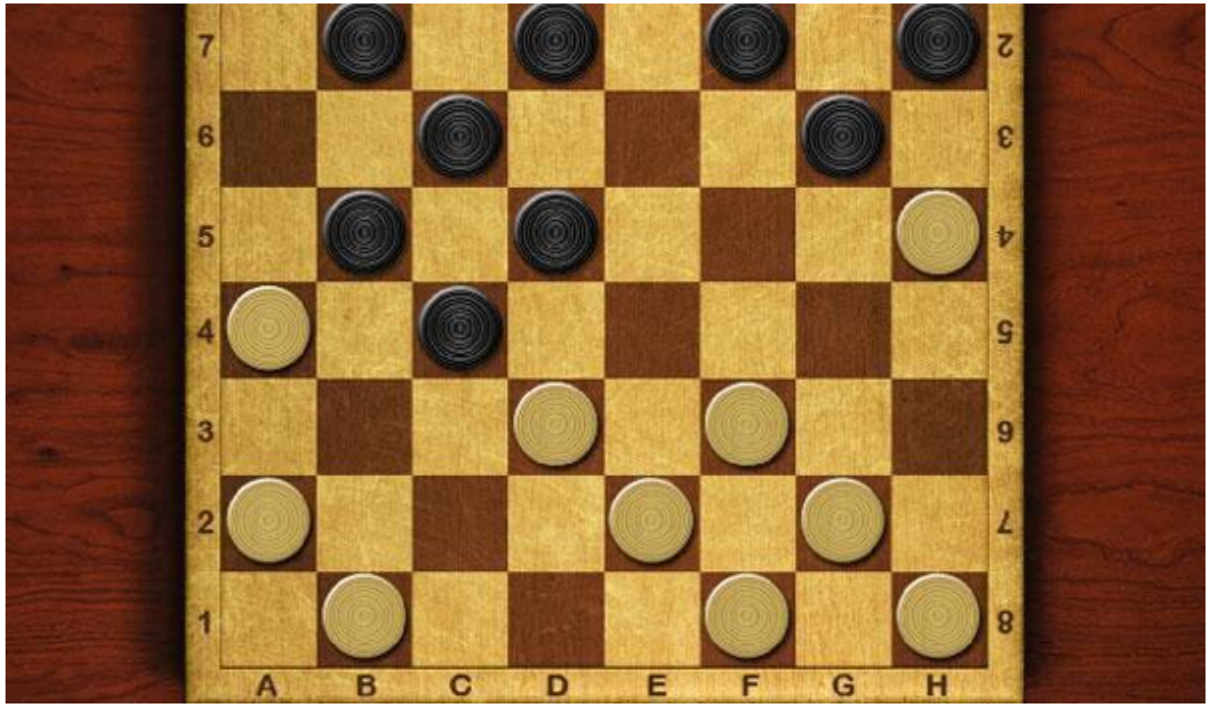
Well-Posed Learning Problem

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.

To have a well-defined learning problem, three features needs to be identified:

1. The class of tasks
2. The measure of performance to be improved
3. The source of experience

Checkers Game Board



https://www.google.com/search?sca_esv=570601344&q=checkers+game&tbm=vid&source=lnms&sa=X&sqi=2&ved=2ahUKEwje1I3hg9yBAxWWbGwGHWVRBQ4Q0pQJegQICxAB&biw=1707&bih=830&dpr=1.13#fpstate=ive&vld=cid:1bd1a206,vid:ScKIdStgAfU,st:0

Rules to play Checkers Game

- Checkers is played by two players. Each player begins the game with 12 colored discs. (One set of pieces is black and the other red.) Each player places his or her pieces on the 12 dark squares closest to him or her. Black moves first. Players then alternate moves.
- The board consists of 64 squares, alternating between 32 dark and 32 light squares.
- It is positioned so that each player has a light square on the right side corner closest to him or her.
- A player wins the game when the opponent cannot make a move. In most cases, this is because all of the opponent's pieces have been captured, but it could also be because all of his pieces are blocked in.

Rules to play Checkers Game

- Moves are allowed only on the dark squares, so pieces always move diagonally. Single pieces are always limited to forward moves (toward the opponent).
- A piece making a non-capturing move (not involving a jump) may move only one square.
- A piece making a capturing move (a jump) leaps over one of the opponent's pieces, landing in a straight diagonal line on the other side. Only one piece may be captured in a single jump; however, multiple jumps are allowed during a single turn.
- When a piece is captured, it is removed from the board.
- If a player is able to make a capture, there is no option; the jump must be made.
- If more than one capture is available, the player is free to choose whichever he or she prefers.

Rules to play Checkers Game

- When a piece reaches the furthest row from the player who controls that piece, it is crowned and becomes a king. One of the pieces which had been captured is placed on top of the king so that it is twice as high as a single piece.
- Kings are limited to moving diagonally but may move both forward and backward. (Remember that single pieces, i.e. non-kings, are always limited to forward moves.)
- Kings may combine jumps in several directions, forward and backward, on the same turn. Single pieces may shift direction diagonally during a multiple capture turn, but must always jump forward (toward the opponent).

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

A handwriting recognition learning problem:

- Task T: recognizing and classifying handwritten words within images
- Performance measure P: percent of words correctly classified
- Training experience E: a database of handwritten words with given classifications

***A robot driving learning problem*:**

- Task T: driving on public four-lane highways using vision sensors
- Performance measure P: average distance travelled before an error (as judged by human overseer)
- Training experience E: a sequence of images and steering commands recorded while observing a human driver

Designing a Learning System

The basic design issues and approaches to machine learning is illustrated by considering designing a program to learn to play checkers, with the goal of entering it in the world checkers tournament

1. Choosing the Training Experience
2. Choosing the Target Function
3. Choosing a Representation for the Target Function
4. Choosing a Function Approximation Algorithm
    1. Estimating training values
    2. Adjusting the weights
5. The Final Design

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.

<u>There are three attributes which impact on success or failure of the learner</u>

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. Whether the training experience provides ***direct or indirect feedback*** regarding the choices made by the performance system.

Direct 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***.

Indirect Feedback

- 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.

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

Direct or Indirect Feedback contd…

- 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.

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

2. A second important attribute of the training experience *is the* ***degree to which the learner controls*** *the sequence of training examples*

With dependence on teacher

- The learner might depends on the teacher to select informative board states and to provide the correct move for each.
- Alternatively, the learner might itself propose board states that it finds particularly confusing and ask the teacher for the correct move.

With-out dependence on teacher

- The learner may have complete control over both the board states and (indirect) training classifications, as it does when it learns by playing against itself with no teacher present.
- Notice in this last case the learner may choose between experimenting with novel board states that it has not yet considered, or honing its skill by playing minor variations of lines of play it currently finds most promising.

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

Learning is most reliable when the training examples follow a distribution similar to that of future test examples.

- 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. For example, the learner might never encounter certain crucial board states that are very likely to be played by the human checkers champion.
- It is necessary to learn from a distribution of examples that is somewhat different from those on which the final system will be evaluated. Such situations are problematic because mastery of one distribution of examples will not necessary lead to strong performance over some other distribution.

Designing a Learning System

The basic design issues and approaches to machine learning is illustrated by considering designing a program to learn to play checkers, with the goal of entering it in the world checkers tournament

1. Choosing the Training Experience
2. **Choosing the Target Function**
3. Choosing a Representation for the Target Function
4. Choosing a Function Approximation Algorithm
    1. Estimating training values
    2. Adjusting the weights
5. The Final Design

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.

- Lets begin with a checkers-playing program that can generate the legal moves from any board state.

- The program needs only to learn how to choose the best move from among these legal moves. This learning task is representative of a large class of tasks for which the legal moves that define some large search space are known a priori, but for which the ***best search strategy is not known.***

Target Function:1

Given this setting where we must learn to choose among the legal moves, the most obvious choice for the type of information to be learned is a program, or function, that chooses the best move for any given board state.

Let ***ChooseMove*** be the target function and the notation is

ChooseMove : B→M

which 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.

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

Target Function:2

An alternative target function is an ***evaluation function*** that assigns a ***numerical score*** to any given board state

Let the target function V and the notation

$$\boldsymbol{V : B \longrightarrow R}$$

which denote that V maps any legal board state from the set B to some real value

We intend for this target function V to assign higher scores to better board states. 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.

- Let us 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

Designing a Learning System

The basic design issues and approaches to machine learning is illustrated by considering designing a program to learn to play checkers, with the goal of entering it in the world checkers tournament

1. Choosing the Training Experience
2. Choosing the Target Function
3. Choosing a Representation for the Target Function
4. Choosing a Function Approximation Algorithm
    1. Estimating training values
    2. Adjusting the weights
5. The Final Design

3. Choosing a Representation for the Target Function

Let us choose a simple representation - for any given board state, the function will be calculated as a linear combination of the following board features:

- xl: the number of black pieces on the board
- x2: the number of red pieces on the board
- x3: the number of black kings on the board
- x4: the number of red kings on the board
- x5: the number of black pieces threatened by red (i.e., which can be captured on red's next turn)
- x6: the number of red pieces threatened by black

Thus, learning program will represent as a linear function of the form

$$\hat{V}(b) = w_0 + w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + w_5x_5 + w_6x_6$$

Where,

- w_0 through w_6 are numerical coefficients, or weights, to be chosen by the learning algorithm.
- Learned values for the weights w_1 through w_6 will determine the relative importance of the various board features in determining the value of the board
- The weight w_0 will provide 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→R
- Target function representation

$$\hat{V}(b) = w_0 + w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + w_5x_5 + w_6x_6$$

- The first three items above correspond to the specification of the learning task.
- The final two items constitute design choices for the implementation of the learning program.
- The net effect of this set of design choices is to reduce the problem of learning a checkers strategy to the problem of learning values for the coefficients w_0 through w_6 in the target function representation.

Designing a Learning System

The basic design issues and approaches to machine learning is illustrated by considering designing a program to learn to play checkers, with the goal of entering it in the world checkers tournament

1. Choosing the Training Experience
2. Choosing the Target Function
3. Choosing a Representation for the Target Function
4. **Choosing a Function Approximation Algorithm**
    1. Estimating training values
    2. Adjusting the weights
5. The Final Design

4. Choosing a Function Approximation Algorithm

- In order to learn the target function ***f*** we require a set of training examples, each describing a specific board state b and the training value $V_{train}(b)$ for b.

- 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 $x_2 = 0$ indicates that red has no remaining pieces) and for which the target function value $V_{train}(b)$ is therefore +100.

  - $((x_1=3, x_2=0, x_3=1, x_4=0, x_5=0, x_6=0), +100)$

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

$$V_{train}(b) \leftarrow \hat{v}(\text{Successor}(b))$$

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.

- In this 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.

- <u>Working of weight update rule</u>

- When the error (Vtrain(b)- V̂(b)) is zero, no weights are changed.
- When (Vtrain(b) - V̂(b)) is positive (i.e., when V̂(b) is too low), then each weight is increased in proportion to the value of its corresponding feature. This will raise the value of V̂(b), reducing the error.
- If the value of some feature 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.

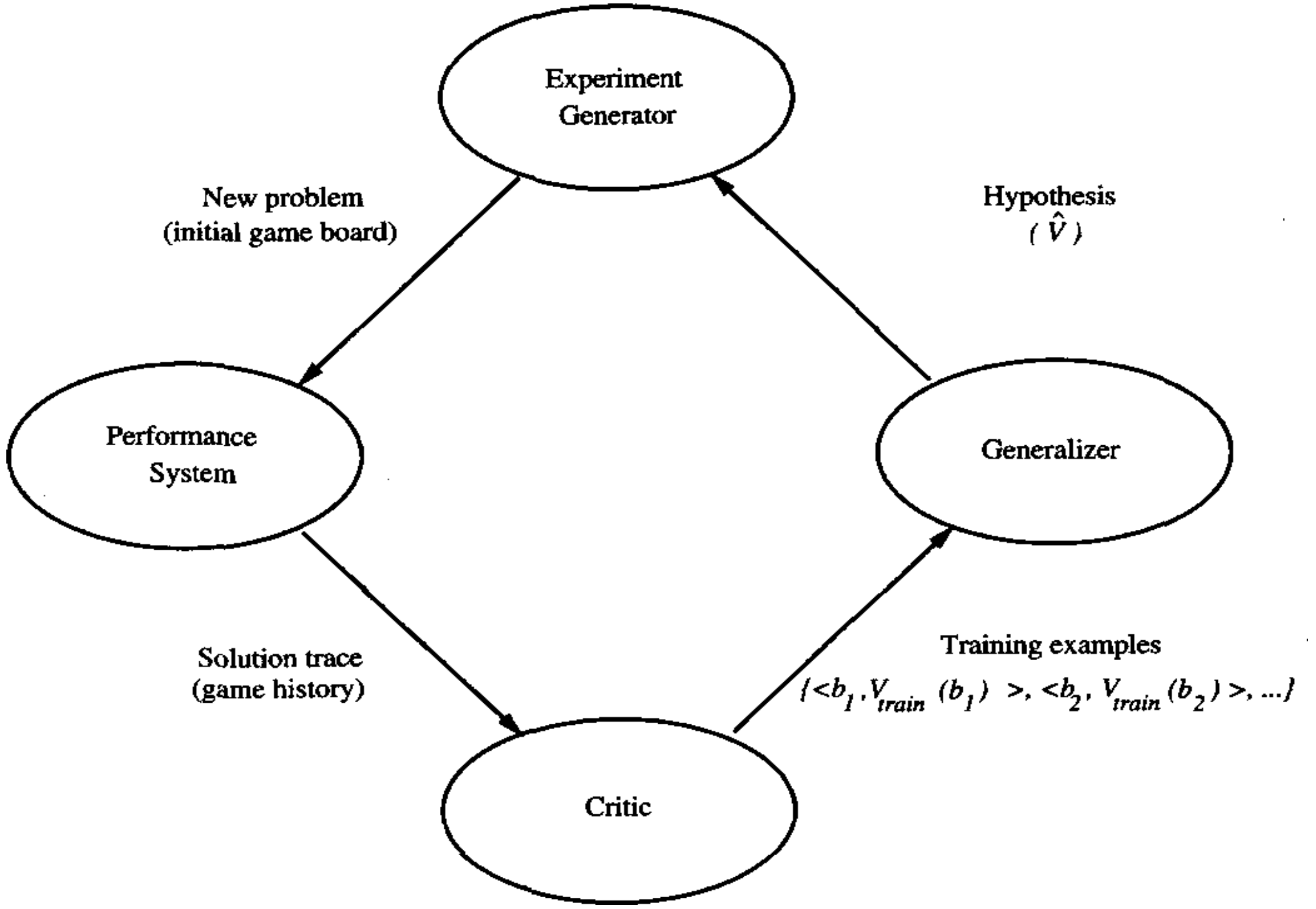
Designing a Learning System

The basic design issues and approaches to machine learning is illustrated by considering designing a program to learn to play checkers, with the goal of entering it in the world checkers tournament

1. Choosing the Training Experience
2. Choosing the Target Function
3. Choosing a Representation for the Target Function
4. Choosing a Function Approximation Algorithm
    1. Estimating training values
    2. Adjusting the weights
5. **The Final Design**

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


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

The Performance System 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 which is determined by the learned $\hat{V}$ evaluation function. Therefore, we expect its performance to improve as this evaluation function becomes increasingly accurate.

The Critic takes as input the history or trace of the game and produces as output a set of training examples of the target function. 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.

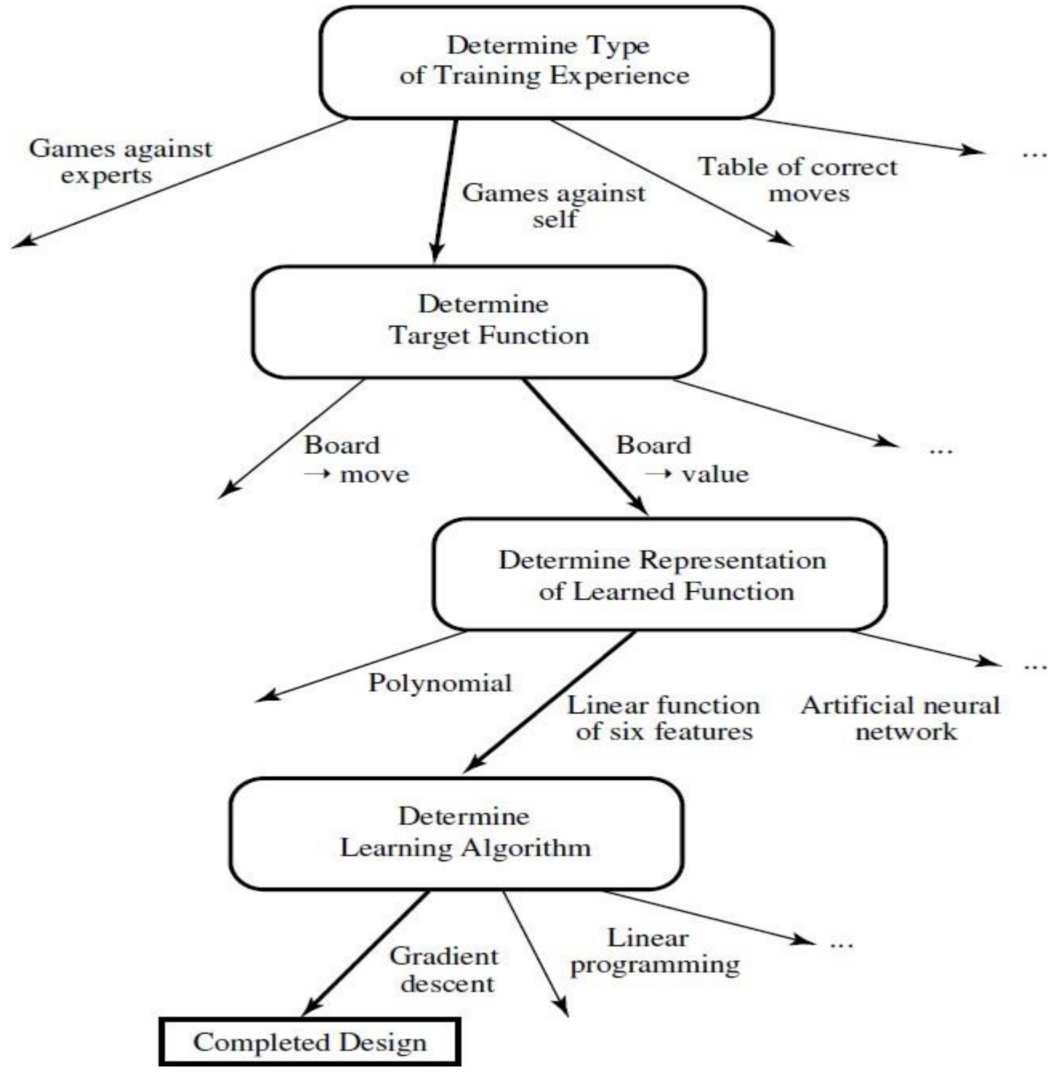
The Generalizer takes as input the training examples and produces an output hypothesis that is its estimate of the target function.

- 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 $w_0, \ldots, w_6$.

The Experiment Generator 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.

The sequence of design choices made for the checkers program is summarized in below figure


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

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?

- 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?

Concept Learning

- Learning involves acquiring general concepts from specific training examples.
- Each such concept can be viewed as describing some subset of objects or events defined over a larger set
- Alternatively, each concept can be thought of as a Boolean-valued function defined over this larger set. (Example: A function defined over all animals, whose value is true for birds and false for other animals).
- Concept learning can be formulated as a problem of searching through a predefined space of potential hypotheses for the hypothesis that best fits the training examples
- **Concept learning** - Inferring a Boolean-valued function from training examples
of its input and output

A Concept Learning Task

- Consider the example task of learning the target concept
  - "Days on which Aldo enjoys his favorite 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

Positive and negative training examples for the target concept ***EnjoySport***

The attribute ***EnjoySport*** indicates whether or not a Person enjoys his favorite water sport on this day.

The task is to learn to predict the value of ***EnjoySport*** for an arbitrary day, based on the values of its other attributes ?

What hypothesis representation is provided to the learner?

Let’s consider a simple representation in which each hypothesis consists of a conjunction of constraints on the instance attributes.

Let each hypothesis be a vector of six constraints, specifying the values of the six attributes Sky, AirTemp, Humidity, Wind, Water, and Forecast.

For each attribute, the hypothesis will either

- Indicate by a "?' that any value is acceptable for this attribute,
- Specify a single required value (e.g., Warm) for the attribute, or
- Indicate by a "Φ" that no value is acceptable

If some instance x satisfies all the constraints of hypothesis h, then h classifies x as a positive example (h(x) = 1).

The hypothesis that PERSON enjoys his favorite sport only on cold days with high humidity (independent of the values of the other attributes) is represented by the expression

(?, Cold, High, ?, ?, ?)

(?, Warm,?, ?, ?, ?)

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-is represented by

(Φ , Φ, Φ, Φ, Φ, Φ)