Learning Resource's

MACHINE LEARNING

By

Tom M. Mitchell, Carnegie Mellon University

McGraw-Hill Education (India)

Edition 2013

Example of Concept Learning: EnjoySport

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

TABLE 2.1
Positive and negative training examples for the target concept *EnjoySport*.

COURSE OBJECTIVES

To formulate machine learning problems corresponding to an application

COURSE OUTCOMES

1 Explain the basics of concept learning and inductive learning.

2 Design decision tree and neural network to solve classification problems.

3 Comprehend probabilistic methods for learning.

4 Explain the instance based learning and reinforcement learning.

5 Build optimal classifiers using Genetic Algorithm and Deep Learning.

Unit-I

- **Introduction:** Well-Posed Learning Problems, Designing a Learning System, Issues in Machine Learning.
- **The Concept Learning:** A concept Learning Task, General –to- Specific Ordering of Hypothesis (Guess, assumption), Find-S, The List-Then-Eliminate Algorithm, Candidate Elimination Learning Algorithm, Inductive bias.

Machine Learning - Chapter 1

Introduction

Artificial intelligence and Machine learning

- Artificial intelligence is a technology that enables a machine to simulate human behavior.
- Machine learning is a subset of AI which allows **a machine to automatically learn from past data without programming explicitly**.
- The goal of AI is to make a smart computer system like humans to solve complex problems.

INTRODUCTION

- A subfield of Computer Science that gives computers the ability to learn without being explicitly programmed.
- To improve automatically with experience.
- We do not yet know how to make computers learn nearly as well as people learn.
- However, algorithms have been invented that are effective for certain types of learning tasks

- Many practical computer programs have been developed to exhibit useful types of learning
- Significant commercial applications have begun to appear.
- For problems such as speech recognition, algorithms based on machine learning outperform all other approaches that have been attempted to date.
- In the field of data mining, machine learning algorithms are being used routinely to discover valuable knowledge from large commercial databases containing, equipment maintenance records, loan applications, financial transactions, medical records
- As our understanding of computers continues to mature, it **is** inevitable that machine learning will play an increasingly central role in computer science and computer technology.

- A successful understanding of how to make computers learn would open up many new uses of computers and new levels of competence and customization.
- A detailed understanding of information(data) processing(dealing) algorithms for machine learning might lead to a better understanding of human learning abilities (and disabilities) as well

Well-Posed Learning Problems

Let us begin our study of machine learning by considering a few learning tasks(responsibilities).

- We will define learning broadly, to include any computer program that improves its performance at some task through experience.
- **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.**
- For example, 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.

What is the Learning Problem

In general, to have a well-defined learning problem, we must identity the following three features:

1. The class of tasks
2. The measure of performance to be improved
3. The source of experience(knowledge or skill)

Examples of Learning Problem

Example Problem 1

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

Example Problem 2

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

Example Problem 3

A robot driving learning problem:

Task T: driving on public four-lane 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

Successful Applications of Machine Learning

- **Learning to recognize spoken words:** the SPHINX system learns speaker-specific strategies for recognizing the primitive sounds and words from the observed speech signal.
- **Learning to drive an autonomous vehicle:** Machine learning methods have been used to train computer-controlled vehicles to steer correctly when driving on a variety of road types. For example, the ALVINN system has used its learned strategies to drive unassisted at 70 miles per hour for 90 miles on public highways among other cars

- **Learning to classify new astronomical structures**: decision tree learning algorithms have been used by NASA to learn how to classify celestial objects.
- **Learning to play world-class backgammon:** the world's top computer program for backgammon, TD-GAMMON (Tesauro 1992, 1995). learned its strategy by playing over one million practice games against itself

Disciplines and Examples of their influence on Machine Learning

- **Artificial intelligence -** Learning symbolic representations of concepts. Machine learning as a search problem. Learning as an approach to improving problem solving. Using prior knowledge together with training data to guide learning.

- **Bayesian methods -** Bayes' theorem as the basis for calculating probabilities of hypotheses. The naive Bayes classifier. Algorithms for estimating values of unobserved variables.

- **Computational complexity theory-** Theoretical bounds on the inherent complexity of different learning tasks, measured in terms of the computational effort, number of training examples, number of mistakes, etc. required in order to learn.

- **Control theory -**Procedures that learn to control processes in order to optimize predefined objectives and that learn to predict the next state of the process they are controlling.

- **Information theory -** Measures of entropy and information content. Minimum description length approaches to learning.

- **Philosophy -** Occam's razor, suggesting that the simplest hypothesis is the best. Analysis of the justification for generalizing beyond observed data.

- **Psychology and neurobiology -**The power law of practice, which states that over a very broad range of learning problems, people's response time improves with practice according to a power law.

- **Statistics -** Characterization of errors (e.g., bias and variance) that occur when estimating the accuracy of a hypothesis based on a limited sample of data.

Checkers game

- https://www.youtube.com/watch?v=ScKIdStgAfU

DESIGNING A LEARNING SYSTEM

let us consider designing a program to learn to play checkers, with the goal of entering it in the world checkers tournament. We adopt the performance measure: the percent of games it wins in this world 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
5. The Final Design

1. Choosing the Training Experience

- The **first** design choice is to choose the type of training experience from which our system will learn.
- The type of training experience available can have a significant impact on success or failure of the learner
- One key attribute is whether the training experience provides direct or indirect feedback regarding the choices made by the performance system.

- For example, 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** information consisting of the move sequences and final outcomes of various games played.
- In this case, 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.

- A **second** important attribute of the training experience is the degree to which the learner controls the sequence of training examples.
- For example, the learner might rely 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 confusing and ask the teacher for the correct move.
- Or the learner may have complete control over both the board states and training classifications, as it does when it learns by playing against itself with no teacher present.

- 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.
- learning is most reliable when the training examples follow a distribution similar to that of future test examples.
- In our 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 an obvious 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

To proceed with our design, let us decide that our system will train by playing games against itself. This has the advantage that no external trainer need be present, and it therefore allows the system to generate as much training data as time permits. We now have a fully specified learning task.

- A checkers learning problem:

Task T: playing checkers

Performance measure P: percent of games won in the world tournament

Training experience E: games played against itself

In order to complete the design of the learning system, we must choose

1. the exact type of knowledge to be learned
2. a representation for this target knowledge
3. a learning mechanism

2. Choosing the Target Function

- To determine what type of knowledge will be learned and how this will be used by the performance program
- Example: 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 these legal moves.
- 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.
- we will find it useful to reduce the problem of improving performance P at task T to the problem of learning some particular ***target function*** such as ***ChooseMove.***
- The choice of the target function will therefore be a key design choice.
- Target function V use the notation, V : B -> R to 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.
- then it can easily use it to select the best move from any current board position.
- This 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

Lets 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 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 (assuming the opponent plays optimally, as well).

- we will use the symbol $\hat{V}$ to refer to the function that is actually learned by our program, to distinguish it from the ideal target function V.

- 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 this definition is not efficiently computable by our checkers playing program, we say that it is a ***nonoperational*** definition.
- 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.
- Thus, we have reduced the learning task in this case to the problem of discovering an ***operational description of the ideal targetfunction*** V.
- 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.***

3. Choosing a Representation for the Target Function

- let us choose a simple representation: for any given board state, the function $\hat{V}$ 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, 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 W_0 through W_6 are numerical coefficients, or weights, to be chosen by the learning algorithm

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:* $V : Board \rightarrow \Re$
- *Target function representation*

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

- The first three items above correspond to the specification of the learning task,
- whereas 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 ***wo*** through ***w6*** in the target function representation.

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 Vtrain(b) for b.
- each training example is an ordered pair of the form (b, Vtrain(b)).
- For instance, the following training example describes a board state b in which black has won the game (***x2*** = 0 indicates that **red** has no remaining pieces) and for which the target function value Vtrain(b) is therefore **+100.**

$$\langle\langle x_1 = 3, x_2 = 0, x_3 = 1, x_4 = 0, x_5 = 0, x_6 = 0\rangle, +100\rangle$$

Estimating Training Values

- This approach is to assign the training value of Vtrain(b) for any intermediate board state b to be $\hat{V}$(successor(b)),
- where $\hat{V}$ is the learner's current approximation to V and where Successor(b) denotes the next board state following b for which it is again the program's turn to move (i.e., the board state following the program's move and the opponent's response).
- This rule for estimating training values can be summarized as

Rule for estimating training values.

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

ADJUSTING THE WEIGHTS

- to specify the learning algorithm for choosing the weights w_i to best fit the set of training examples {<b, Vtrain(b)>}
- One common approach is to define the best hypothesis, or set of weights, as that which minimizes the square error E between the training values and the values predicted by the hypothesis $\hat{V}$

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

- Several algorithms are known for finding weights of a linear function that minimize E
- In our case, we require an algorithm that will incrementally refine the weights as new training examples become available
- 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 training rule

LMS weight update rule.

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

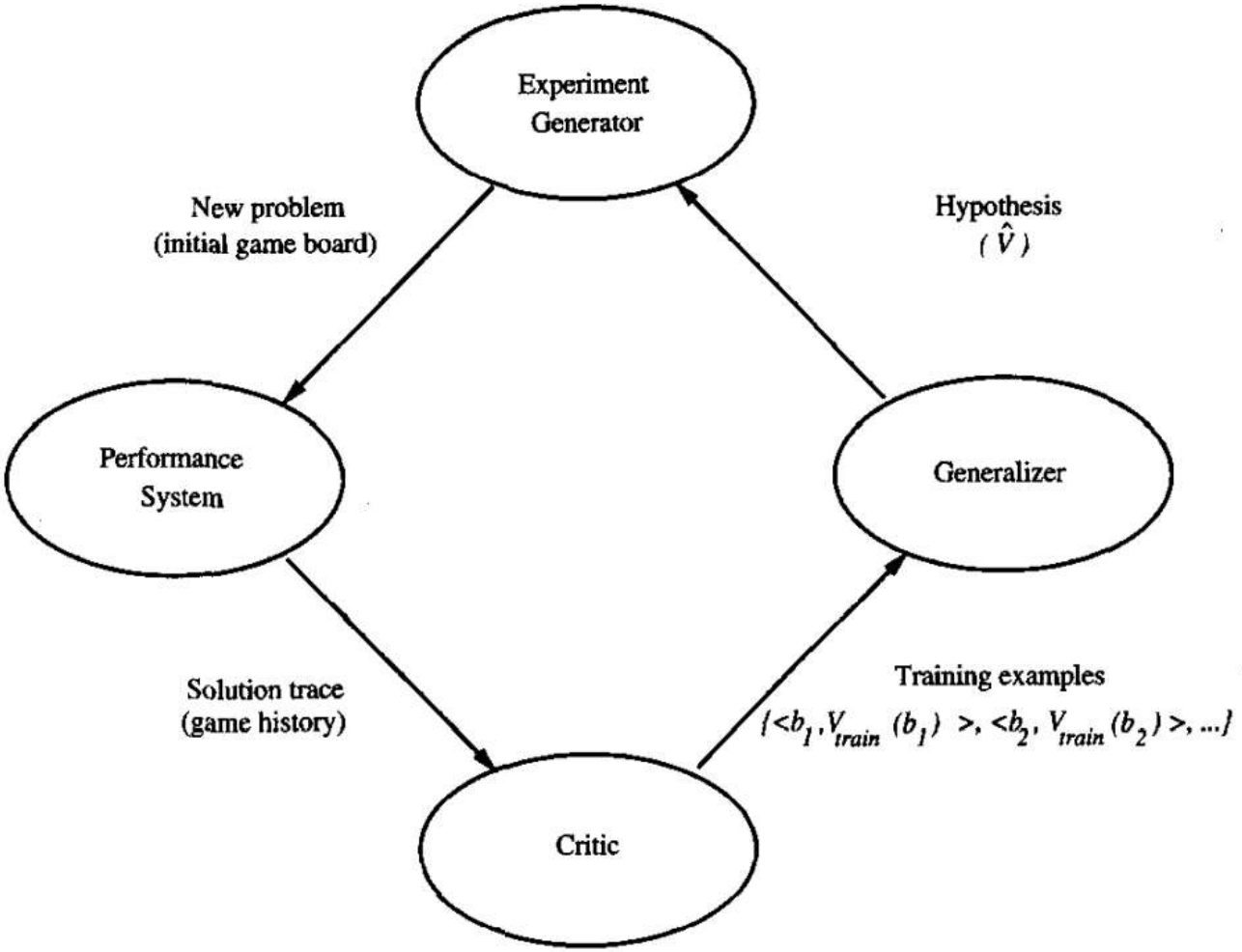
- Use the current weights to calculate $\hat{V}(b)$
- For each weight w_i, update it as

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

- Here n is a small constant (e.g., 0.1) that moderates the size of the weight update.
- To get an intuitive understanding for why this weight update rule works, notice that when the error (Vtrain(b) – $\hat{V}$ (b))is zero, no weights are changed.
- When (Vtrain(b) – $\hat{V}$ (b)) 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 ***xi*** is zero, then its weight is not altered regardless of the error,
- 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 modules that represent the central components in many learning systems.

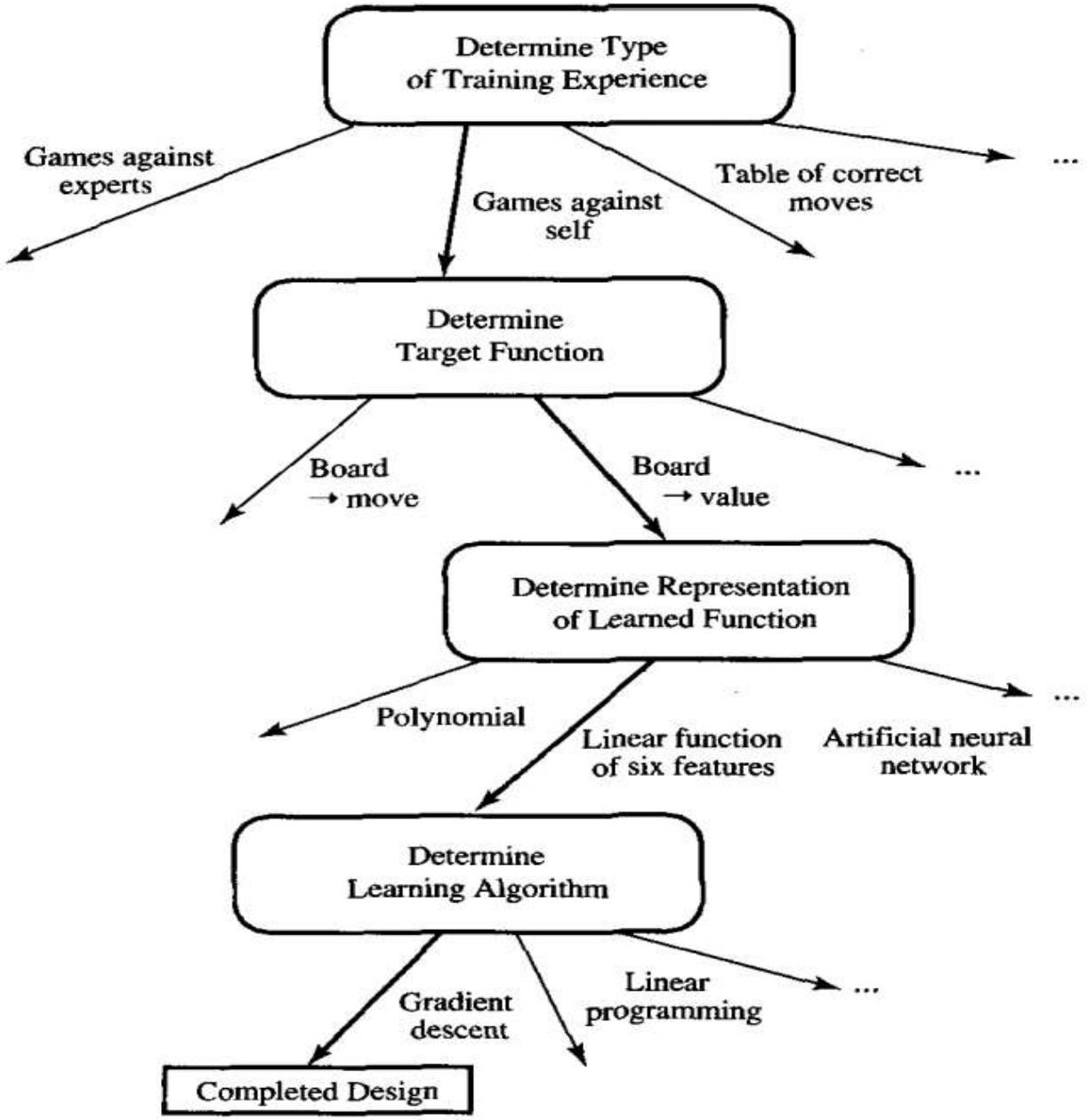


- The **Performance System** is the module that must solve the given performance task, in this case playing checkers, by using the learned target function(s). It takes an instance of a new problem (new game) as input and produces a trace of its solution (game history) as output.
- The **Critic** takes as input the history or trace of the game and produces as output a set of training examples of the target function.

- 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 **f** described by the learned weights ***wo,*** . . . , ***W6.***

- The **Experiment Generator** takes as input the current hypothesis (currently learned function) 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.

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



PERSPECTIVES AND ISSUES IN MACHINE LEARNING

- machine learning involves searching a very large space of possible hypotheses to determine one that best fits the observed data.

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
- 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 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?