Tom Mitchell: Machine Learning

# Chapter 1: Introduction

## Well-posed learning problems

Task – T

Performance measure – P

Training experience – E

*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.*

## Designing a learning system

Example in playing checkers

### Choosing the training experience

Direct training – Examining specific board states and giving feedback according to the move.

Indirect training – Examining a full game only giving feedback after a win/loss

Direct training is usually easier to learn from than indirect training.

Teacher can present interesting board states or the learner can present board states which it is undecisive on having a teacher confirm the correct move. The learner may even compete 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

A checkers learning problem:

0 Task T: playing checkers

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

0 Training experience E: games played against itself

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

1. the exact type of knowledge to be,learned

2. a representation for this target knowledge

3. a learning mechanism

### Choosing the target function

* All the **legal** moves
* Choose the **best** moves of the legal moves

Thes learning task in representative of many tasks which contains legal moves but needs to find an optimal strategy.

We need a function **Choose move: B -> M** – Any legal boardstate B can lead to any legal move M. This is what is to be learned making the choice of the function a key design choice.

In order to **evaluate boardstates** we make a function, **V,** which learns to score each board state taking a board state B and generating and scoring all legal subsequent boardstates and returns the one with the highest score. The optimal function V is not computationally efficient hence an approximation will be used, **Vhat**. The function maps a board state to a real number: **B -> R**

### Choosing a representation for the target function

To keep the discussion brief, let us choose a simple representation:

for any given board state, the function c will be calculated as a linear combination

of the following board features:

* x1: 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 Vhat(b) as a linear function of the form

|  |  |  |
| --- | --- | --- |
|  |  | **(1.1)** |

where w0 through w6 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
* Targetfunction: V:Board -> R
* Targetfunction representation
  + See function above

First the points is specifikation of learning task and the last two is design choices for implementation. We now have a reduced the task of learning the game to optimizing the values w1 through w6.

### Choosing a function approximation algorithm

#### Estimating training values

In order to evaluate the current board states value we use the approximation of the successor (when it is the algorithm’s turn again) and assign this to the current board states training value. This is a sort of back tracking in order to counter the opponents move.

### Adjusting the weights

We need to define the best fit to the training data, commonly defined as:

The error, **E**, is to be minimized.

To minimize E we may use **LMS training rule**. Can be described as performing a stochastic gradient-descent search through the space of possible hypotheses to minimize the value E.

#### LMS weight update rule.

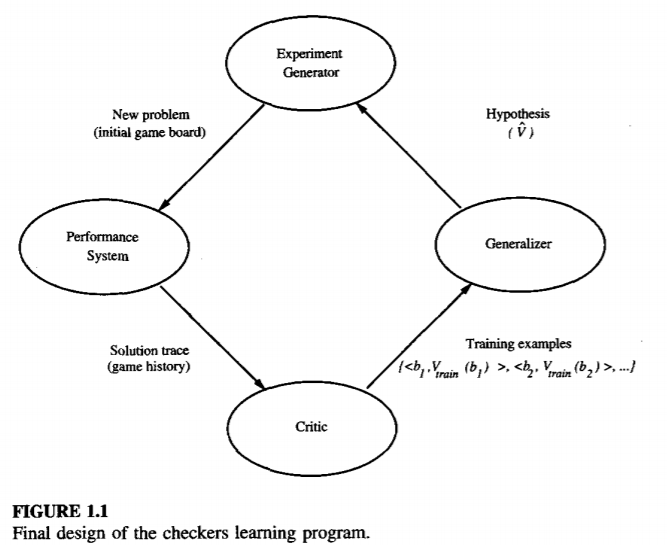
For each training example [b, Vtrain(b)]

* Use the current weights to calculate Vhat(b)
* For each weight wi, update it as

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

## The final design

The final design can, just like many other learning systems, be described in four distinct program modules.



* The **Performance system** is the module that must solve the given performance task. Using a board and Vhat it will become increasingly accurate given enough games.
* 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. It corresponds to the equation (1.1)
* The **Generalizer** takes as input the training examples and produces an output hypothesis that is its estimate of the target function. The Generalizer corresponds to the LMS algorithm, and the output hypothesis is the function Vhat described by the learned weights w0, … , w6.
* The **Experiment Generator** takes as input the current hypothesis (currently learned function) and outputs a new problem for the performance system to explore. In our case only a new game. However one could present specific boards to explore particular regions of the state space.

