ETD 403: Machine Learning

(1) INTRODUCTION

Machine Learning, in one form or the other, has become intrinsic in everyone's life recently. This course introduces the basic concepts and algorithms of machine learning which help provide intelligent solutions to real world problems.

Welcome to the course, Guys!

Well-posed learning problems

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.

Eg:

Task T	Experience E	Performance P
① A computer program learning to play checkers.	Experience obtained by playing games against itself.	The program's ability to win the game of checkers (Total % of games won)
② Learning to recognize spoken words	Speaker-specific strategies to recognize phonemes	Recognition Rate
③ Recognizing and classifying hand-written character recognition	a database of handwritten words and their corresponding classifications	% of words correctly classified

Task T	Training Experience E	Performance P Measure
④ Robot driving learning problem (T- driving on public 4-lane highways using vision sensors)	a sequence of images and steering commands recorded while observing a human driver.	Average distance travelled before an error (as judged by a human overseer.)

Once the problem is formally defined, the next step is to design a learning system by exploring algorithms that solve such problems.

Designing a Learning System

① choosing the training experience

The type of training experience chosen has a direct effect on the level of success of the learning system.

- Does the training experience provide direct or indirect feedback about the choices made by the performance element?

Ex1: training experience providing direct feedback
⇒ Computer program learning to play checkers system learns from direct training examples mapping board states to correct moves.

Ex2: training experience providing indirect feedback
⇒ correctness of a move can be inferred by the final outcome of a game.

- Degree to which the learner controls the sequence of training examples.
  ① The learner can rely on a teacher to select correct move at each state.
  (or)
  ② The learner might itself propose board states and ask teacher for the current move
  (or)
  ③ The learner has complete control over board states and training classifications

- How well the training experience represents the distribution of examples over which 'P' is measured.

  In checkers game, P = % of games won
  if E consists of only games played by the program against itself, the performance measures might be great but the program may fail against a human contender.

  ∴ distribution of training examples = distribution of test examples

For checkers learning problem, we must choose
1, the exact type of knowledge to be learnt
2, a representation for this target knowledge.
3, a learning mechanism.

② Choosing the Target Function

i.e., What type of knowledge will be learnt? How will this be used by the performance element?

For checkers problem, generating legal moves from any board state is easy but selecting the best move is difficult.

- **ChooseMove : B → M**

"ChooseMove" is a function which maps the set of legal moves B to the best move M.

- An alternative is to design a target function

$$V : B \rightarrow \mathbb{R}$$

V maps a board state to a real number, with higher scores for better board positions.

This notation is better.

V(b) where $b \in B$ can be defined as:

(trivial states) game ended

- $V(b) = 100$ if b is final winning state
- $V(b) = -100$ if b is final but losing state
- $V(b) = 0$ if b is final state & match is drawn
- If b is not final state, then $V(b) = V(b')$ where b' is best final state obtained when starting to play from b

Usually, learning algorithms obtain only an approximation to the target function ($\hat{V} \approx V$)

- $\hat{V}$: target function learnt by program
- V : ideal target function

③ choosing a representation for target function

$\hat{V}$ can be represented as

- a large table ~~in~~ with one entry per board position

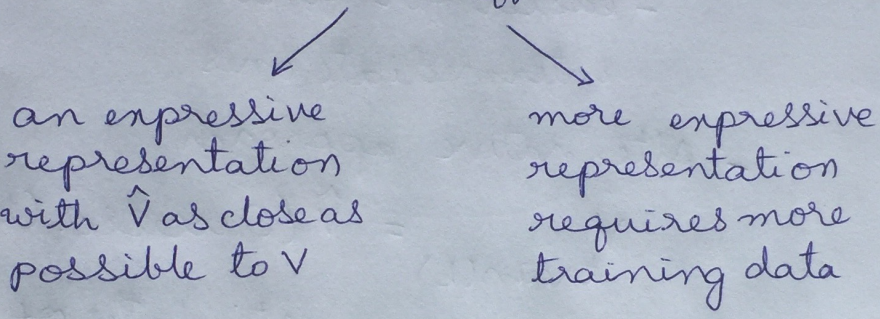
(or)

- Set of rules to match against the features of the board position

(or) a quadratic polynomial function

(or) an artificial neural network

choice of representation tradeoffs



Eg: $\hat{V}$ can be represented as linear combination of following board features

$x_1 \rightarrow$ no. of black pieces on the board

$x_2 \rightarrow$ no. of red pieces on the board

$x_3 \rightarrow$ no of black kings on the board

$x_4 \rightarrow$ no. of red kings on the board

$x_5 \rightarrow$ no of black pieces threatened by red

$x_6 \rightarrow$ no. of red pieces threatened by black

∴ The learning program

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

where weights $w_1 - w_6$ will determine the relative importance of board features $x_1 - x_6$

④ choosing a function approximation algorithm

To learn target function $\hat{V}$, we need training examples.

Each training example is an ordered pair

$$(b, V_{train}(b))$$

- Estimating training values

only information available to learner ⇒ match won or lost

- scores can be assigned to training examples
  - easier for final board states than intermediate ones
- Alternative approach

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

we are using estimates of Successor(b) to estimate value of board state b

- Adjusting the weights
  - first approach: define best hypothesis or set of weights that minimizes square error E

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

Eg: Least Mean Squares (LMS) training rule

LMS weight update rule

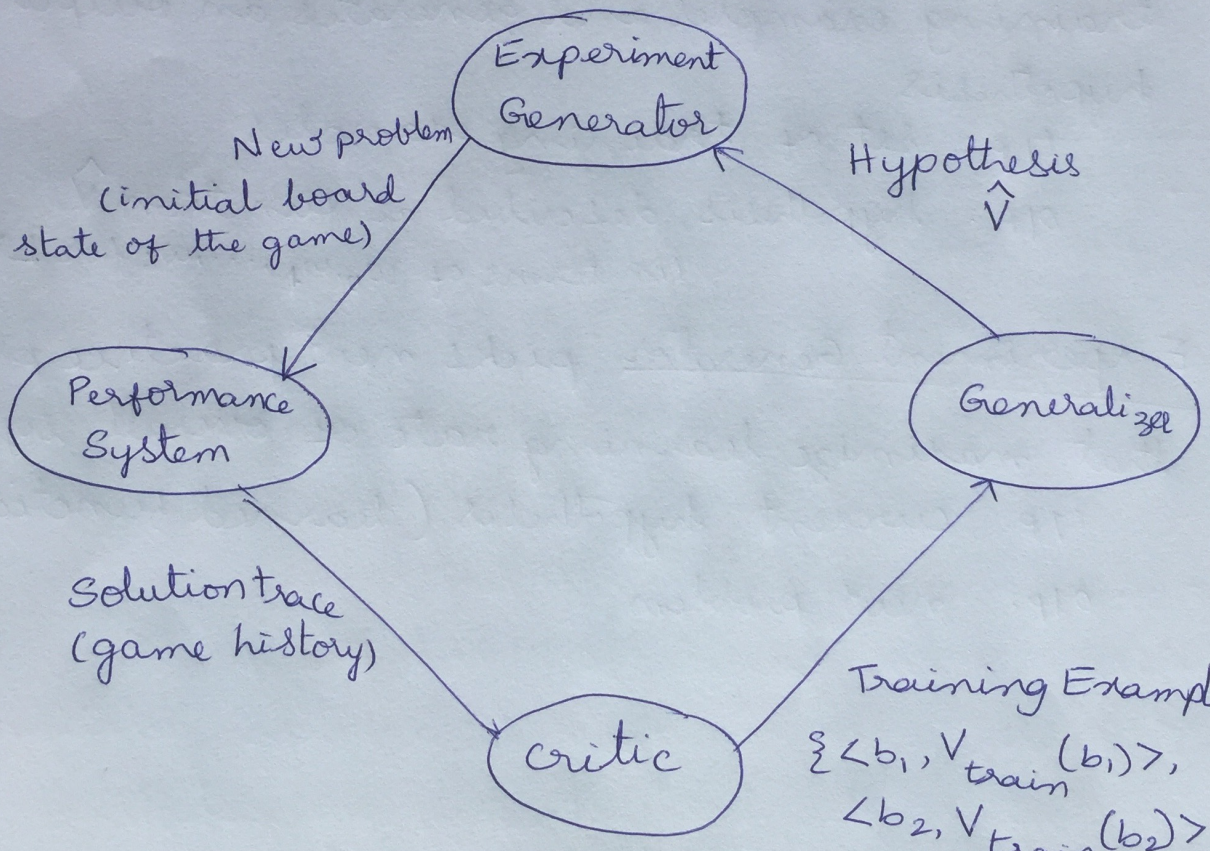
for each training example $\langle b, V_{train}(b) \rangle$

- use current weights to calculate $\hat{V}(b)$
- update each weight w_i

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

This weight-tuning method ~~work~~ converges mostly in least square error methodology.

(5) Final Design



Final design of checkers learning game

- Performance system solves performance task by using learned target function.
  i/p: new problem (or new game)
  o/p: solution trace (or game history)
- Critic module generates the training experience
  i/p: history or trace of the game
  o/p: set of training examples of the target function
- Generalizer: The algorithm which takes specific training examples and generates an output hypothesis
  i/p: set of training examples
  o/p: hypothesis described as function $\hat{V}$ (in terms of $w_0, w_1, \ldots, w_6$ for LMS)
- Experiment Generator picks new practice problems that maximize learning rate of overall system
  i/p: current hypothesis (learned function, $\hat{V}$)
  o/p: new problem

Issues in Machine Learning

Many methods in machine learning involve searching through a huge set of possible hypothesis for the best fit hypothesis for a given training data.

- LMS tunes the weights every time the predicted value is far from the actual value
- Different hypothesis representations are appropriate for learning different kinds of target functions.

Issues

- What algorithms exist to generalize target function from specific training examples? How do these algorithms converge to desired function?
  What is the best algorithm for given problem or representation?
- Is training data sufficient?
- can prior knowledge about the problem guide the learner towards a better solution?
- what is the best strategy for selecting next training experience?
- can the learning task of approximating target functions be automated
- can a learner automatically change its representation to learn a target function?