LESSON 2
DESIGNING A LEARNING SYSTEM

To illustrate some of the basic design issues and approaches to machine learning. We can consider designing a program to learn to play checkers, with the goal of entering it in the world checkers tournament.

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

The obvious performance measure is adopted: i.e, the percent of games it wins in this world tournament.

To design a learning system we consider the following

- Determine the type of training experience
- Determine the exact type of knowledge to be, learned
- Determine a representation for this target knowledge
- Determine a learning mechanism (algorithm)

Choosing the Training Experience

The first design choice is to determine 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.

We consider the following attributes

1. Is 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.

Alternatively, it might have available only ***indirect*** information consisting of the move sequences and final outcomes of various games played. 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

2. 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 particularly confusing and ask the teacher for the correct move. ***Or*** 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.

3. **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 general, 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.

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.

In checkers-playing program it generates 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

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 us call this function ***ChooseMove*** and use the notation ***ChooseMove*** : B $\rightarrow$ 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 therefore reduce the problem of improving performance P at task T to the problem of learning some particular ***targetfunction*** such as ***ChooseMove.*** These becomes a key design choice as the target function.

If we consider a ***ChooseMove*** as our ***c***hoice for the target function it poses a difficulty in to learn given the kind of indirect training experience available to our system

An alternative target function is an evaluation function that assigns a numerical score to any given board state. Assume the target function ***V*** : B + $\Re$ to denote that ***V*** maps any legal board state from the set B to some real value ($\Re$ is used to denote the set of real numbers). 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.

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.

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

Choosing a Representation for the Target Function
The ideal target function V has been specified, now we must choose a representation that the learning program will use to describe the function V' that it will learn. As with earlier design choices, we again have many options. We could, for example, allow the program to represent V' using a large table with a distinct entry specifying the value for each distinct board state. Or we could allow it to represent V' using a collection of rules that match against features of the board state, or a quadratic polynomial function of predefined board features, or an artificial neural network.

In general, this choice of representation involves a crucial tradeoff. On one hand, we wish to pick a very expressive representation to allow representing as close an approximation as possible to the ideal target function V.

On the other hand, the more expressive the representation, the more training data the program will require in order to choose among the alternative hypotheses it can represent. To keep the discussion brief, let us choose a simple representation: for any given board state, the function $\acute{V}$ 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 $\acute{V}$ (b) 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 ***w0*** through ***W6*** are numerical coefficients, or weights, to be chosen by the learning algorithm

thus far

Partial design of a checkers learning program:
1. Task T: playing checkers
2. Performance measure ***P:*** percent of games won in the world tournament
3. Training experience E: games played against itself
4. Target function: V:Board + $\Re$
5. 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, 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 ***w0*** through ***w6*** in the target function representation

Choosing a Function Approximation Algorithm

In order to learn the target function 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.**

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

Estimating the training values

The only training information available so far to our learner is whether the game was eventually won or lost. On the other hand, we require training examples that assign specific scores to specific board states. While it is easy to assign a value to board states that correspond to the end of the game, it is less obvious how to assign training values to the more numerous ***intermediate*** board states that occur before the game's end.

This approach is to assign the training value of V***train(b)*** or 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

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

Adjusting the weights

A 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 ***V.***

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

Least Mean Squares 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

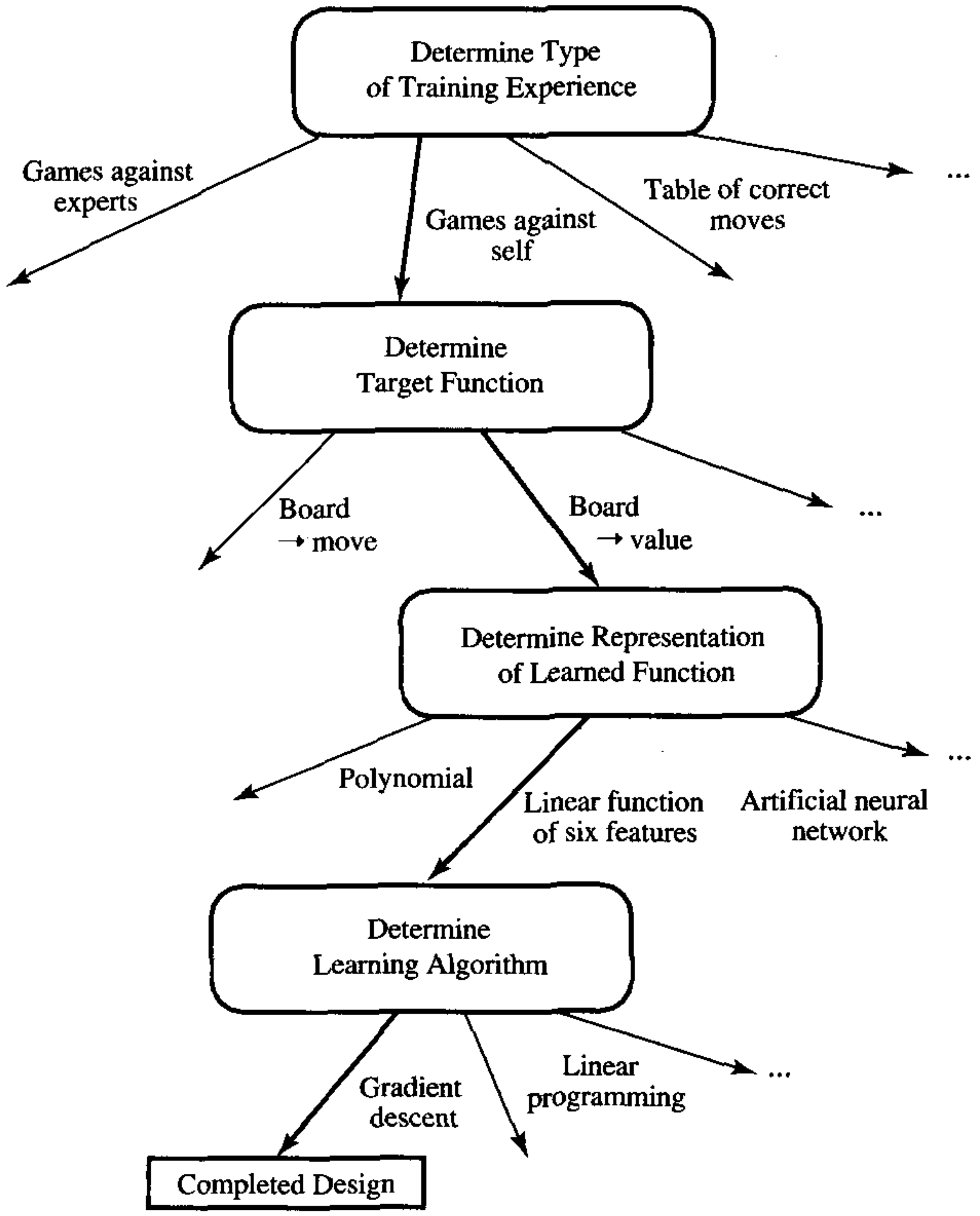
$$w_i \leftarrow w_i + \eta \; (V_{train}(b) - \hat{V}(b)) \; x_i$$

Here η 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) - V' (b)) is zero, no weights are changed.

When (Vtrain(b) - V' (b)) is positive (i.e., when 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 V' (b), reducing the error. Notice that if the value of some feature ***xi*** 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.

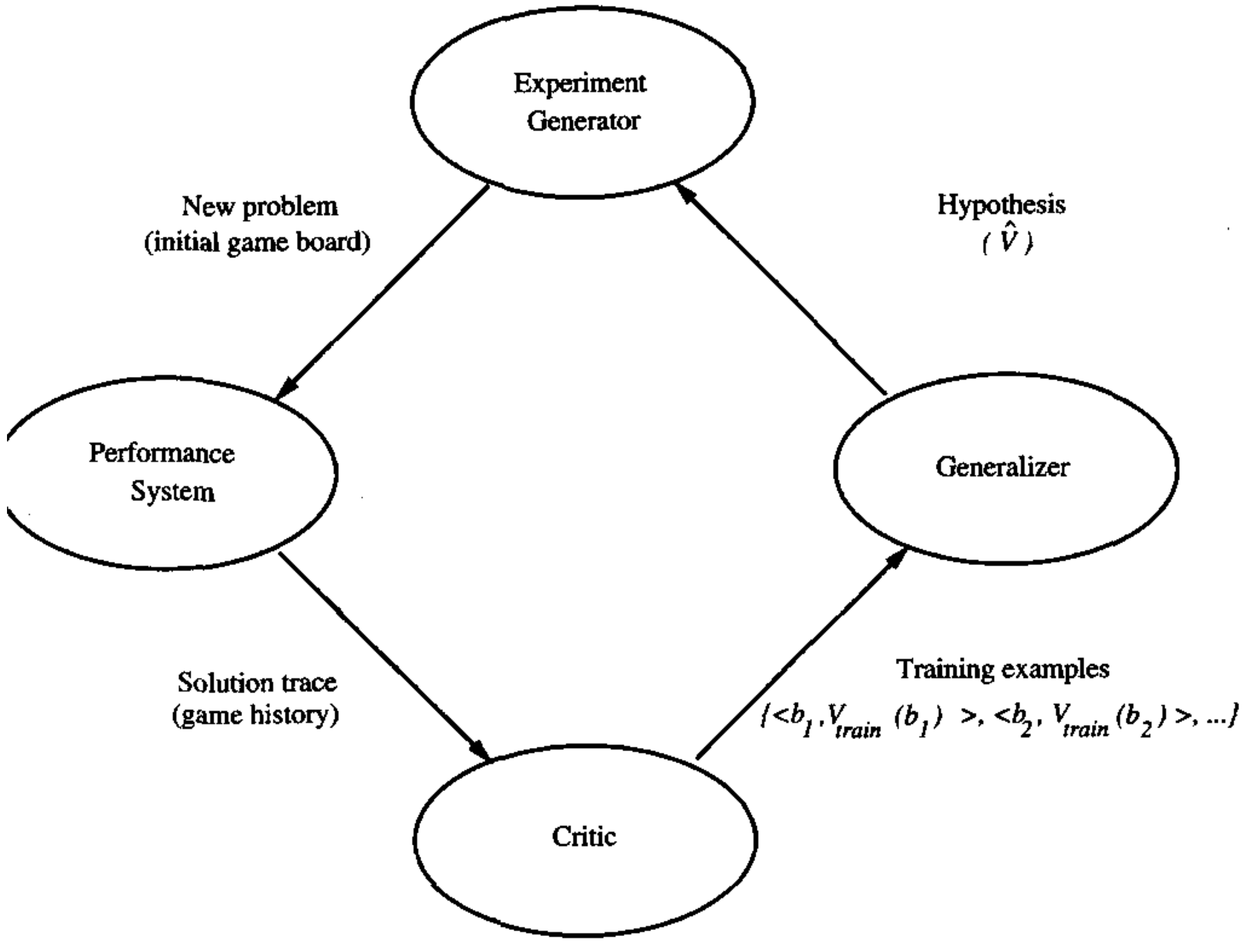

Determine Type of Training Experience
Games against experts
Games against self
Table of correct moves
...
Determine Target Function
Board → move
Board → value
...
Determine Representation of Learned Function
Polynomial
Linear function of six features
Artificial neural network
...
Determine Learning Algorithm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

The Final Design
Proram modules
1. 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

2. 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.

3. 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

4. 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.


Experiment Generator
New problem (initial game board)
Hypothesis ($\hat{V}$)
Performance System
Generalizer
Solution trace (game history)
Training examples
$\{<b_1, V_{train}(b_1)>, <b_2, V_{train}(b_2)>, \ldots\}$
Critic

Problems in machine learning

Optical character recognition:
- Categorize images of handwritten characters by the letters represented

Face detection:
- Find faces in images (or indicate if a face is present)

Spam filtering:
- identify email messages as spam or non-spam

Topic spotting:
- Categorize news articles (say) as to whether they are about politics, sports, entertainment, etc.

Spoken language understanding:
- Within the context of a limited domain, determine the meaning of something uttered by a speaker to the extent that it can be classified into one of a fixed set of categories

Choosing the Training Experience
- Sometimes straightforward
    - Text classification, disease diagnosis
- Sometimes not so straightforward
    - Chess playing

Solving Real World Problems
What Is the Input?
- Features representing the real world data

What Is the Output?
- Predictions or decisions to be made

What Is the Intelligent Program?

- Types of classifiers (input-features, output-single decision), value functions(input-feature, output-values), etc.

How to Learn from experience?

- Learning algorithms

Applications of machine learning

Data mining

- In data mining, machine learning algorithms are used routinely to discover valuable knowledge from large commercial databases containing records such as equipment maintenance, loan applications, financial transactions, medical records, etc.

Learning associations

- In supermarkets for example, ML is applied to basket analysis, which is to find an association among the products a customer buys.
- In finding an association rule, association rule we are interested in learning a conditional probability of the form P(Y|X) where Y is the product we would like to condition on X, which is the product or the set of products which we know that the customer has already purchased. Let us say, going over available data, we calculate that P(chips|beer) = 0.7. Then, we can define the rule: 70 percent of customers who buy beer also buy chips.

Classification

- In financial institutions, may want to classify the type of customers into two or more classes, say low-risk and high risk (for the sake of loans) based on credit scoring calculated from the past data.

- Class 1: low-risk (income>P1 and savings>p2)
- Class 2: high-risk (vice versa)
- Therefore ML classifies candidates into various classes using predictions based on experience.

Unsupervised learning

- In supervised learning, the aim is to learn a mapping from the input to an output whose correct values are provided by a supervisor.
- While in unsupervised learning we have only input. Its aim is to find regularities in the input data. This means that there is a structure in the input space forming certain patterns.
- Clustering or groupings is used to find clusters/groupings of inputs that form a specific pattern, example demographic data of patients to an hospital.

Successful application of machine learning

Recognize spoken words.

- All of the most successful speech recognition systems employ machine learning in some form. For example, the SPHINX system learns speaker-specific strategies for recognizing the primitive sounds (phonemes) and words from the observed speech signal. Neural network learning methods and methods for learning hidden Markov models are effective for automatically customizing to, individual speakers, vocabularies, microphone characteristics, background noise, etc. Similar techniques have potential applications in many signal-interpretation problems.

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. Similar techniques have possible applications in many sensor-based control problems.

Classify new astronomical structures.

- Machine learning methods have been applied to a variety of large databases to learn general regularities implicit in the data. For example, decision tree learning algorithms have been used by NASA to learn how to classify celestial objects from the second Palomar Observatory Sky Survey. This system is now used to automatically classify all objects in the Sky Survey, which consists of three terabytes of image data.

Play world-class games (such as backgammon).

- The most successful computer programs for playing games such as backgammon are based on machine learning algorithms. For example, the world's top computer program for backgammon, TD-GAMMON. learned its strategy by playing over one million practice games against itself. It now plays at a level competitive with the human world champion. Similar techniques have applications in many practical problems where very large search spaces must be examined efficiently.