Game Applications

Reinforcement Learning

Temporal Differences

Example: Game Playing

- How to learn to play a game, say tic-tac-toe?
- **Supervised** learning approach:

  Listen to a **teacher** indicate good moves in various situations, or

  Observe an expert player play games; learn to mimic the good player.

Problems with Supervised Learning

- Need access to **expert**/teacher or many recorded game samples.
- The teacher might not be perfect.
- There is typically **no** play-by-play **target** value; the value is only assigned to a sequence of plays, ending with +1 (win) or -1 (loss).
- The environment is **non-deterministic**, so dealing with a single sequence is not enough.
- We don't necessarily have a **model** for the environment (e.g. the opponent).

Game Applications

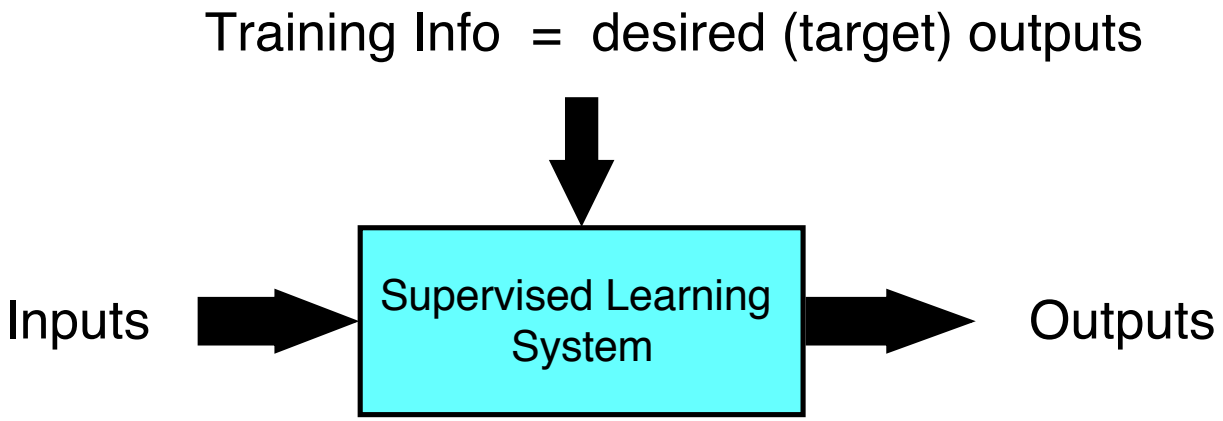
- BPPT (Backpropagation Through Time) seems potentially useable.
- Another approach is to use the “Temporal Difference” method, which is an example of “reinforcement learning”.

General Learning Types

- **Supervised learning**: Training with desired answer given for each action
- **Unsupervised learning**: No desired answer; learns *similarities* between training patterns (e.g. clustering)
- **Reinforcement learning**:
  Generalization of Supervised Learning:

  Reward given later, not necessarily tied to specific action now

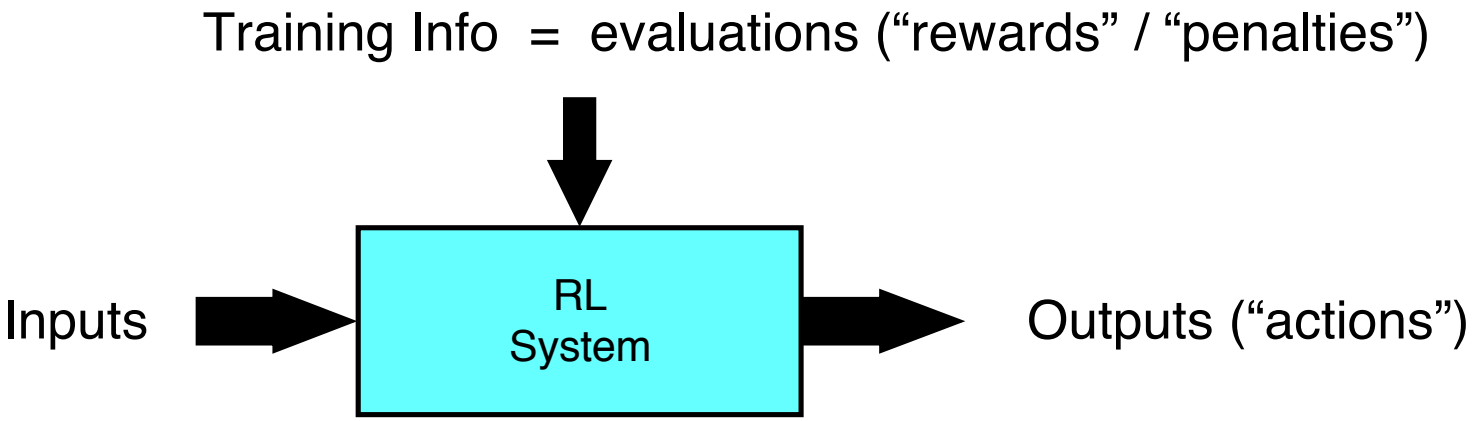
Supervised Learning



Error = (target output – actual output)

Reinforcement Learning

(slide from Sutton and Barto)



Objective: get as much reward as possible

Key Features of RL

(slide from Sutton and Barto)

- Learner is not told *which* actions to take
- Trial-and-Error search
- Possibility of delayed reward

  Sacrifice short-term gains for greater long-term gains
- Need to ***explore*** and ***exploit***
- Consider the whole problem of a goal-directed agent interacting with an uncertain environment

A Few Reinforcement Learning Examples

- **TD-Gammon**: Tesauro
  - world's best backgammon program
- **Elevator Control**: Crites & Barto
  - high performance down-peak elevator controller
- **Inventory Management**: Van Roy, Bertsekas, Lee&Tsitsiklis
  - 10–15% improvement over industry standard methods
- **Dynamic Channel Assignment**: Singh & Bertsekas, Nie & Haykin
  - high performance assignment of radio channels to mobile telephone calls

Reinforcement Learning for Games

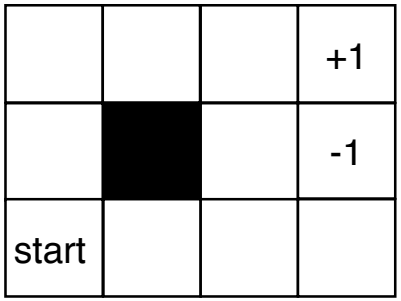
- Reward often deferred until the end of the game.
- Must deal with stochastic environment (transition probabilities).

Typical AI Model: MDP (Markov Decision Problem)

- Set of states, maybe very large
- Set of actions
- Transition probabilities between states:
  M^a_{ij} = prob[transition from state i to state j,
  given **action** *a* is taken in state i]
  [Actions are deterministic; all non-determinism is in transition probabilities.]
- Reward R(i) (positive, negative, or 0) associated with each state i.
- The objective is to accrue as much reward as possible.

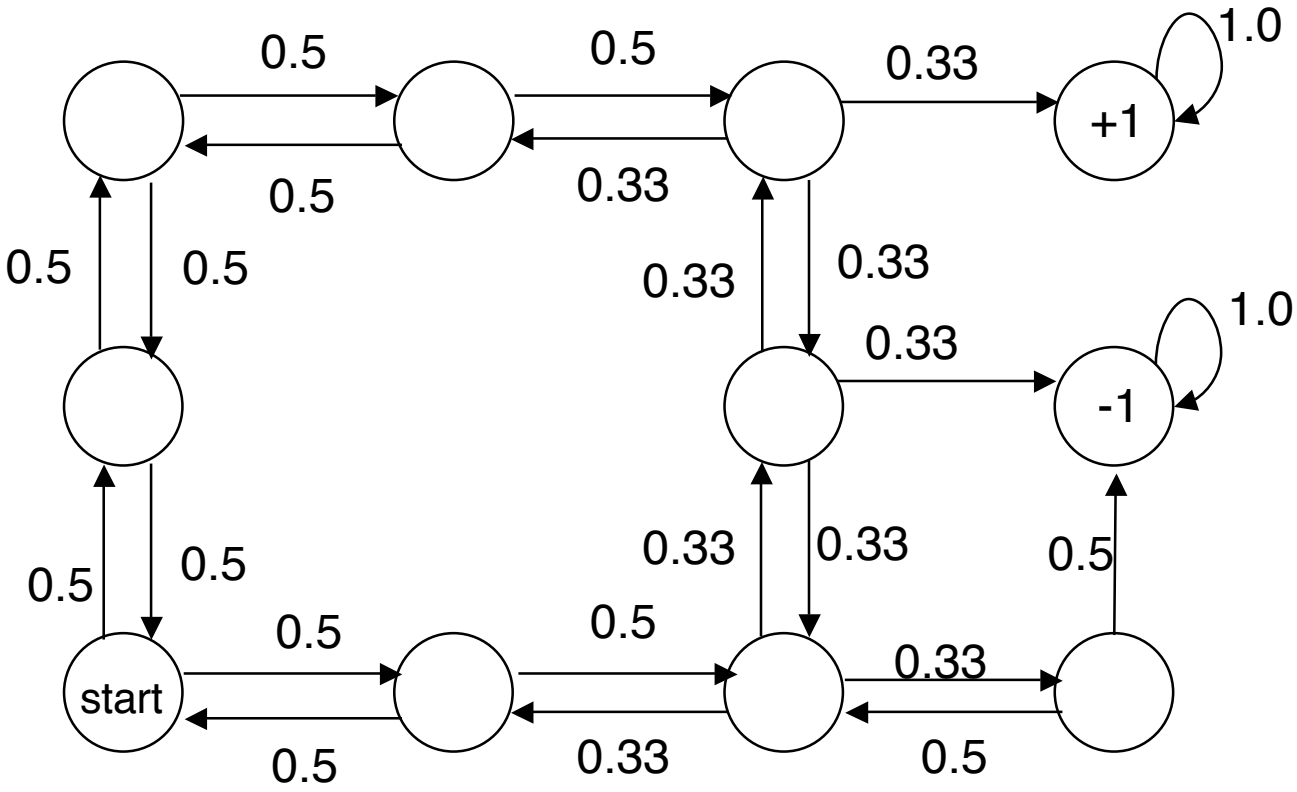
3x4 “Grid World” (Explicit States) Example (Norvig & Russell)

- Consider the following maze, with reward function 0 except as shown in the two boxes (with +1, -1).
- Assume a single action “move” with **equal probabilities** of moving any direction, except stay in +1 or -1 if reached.



Markov State-Transition Probs

based on equal probs. of any action in a state



Problem

- Which moves to make, in a given state, to get the most reward?

Utility Functions

- Desirability of moving to a given state s is expressed by a ***utility*** $V(s)$ [also called the "value" of the state s].
- The **utility is not the reward**, but rather an ***estimate*** of the award that can be accrued from that state.
- The utility is generally not given explicitly; it must be ***learned***.
- Normally the ***expected*** utility is sought, since transitions can be probabilistic.
- A general strategy: Given a choice of moves to several states, choose the state with ***highest expected utility***.

Additive Utility Function

- Assume that the sought utility function V is *additive*, in that it obeys the relationship

Utility of state i

$$V(i) = R(i) + \max_{a}\left(\sum_{j} P^{a}_{ij} \cdot V(j)\right)$$

Expected utility of next state given a.

where

- R(i) is the reward of state i, (could be 0.)
- a is an action, and
- P^{a}_{ij} is the probability of going from state i to state j with action a.

- This is the ***dynamic programming*** equation (Richard Bellman).

A conceptual way to compute V iteratively

- For each state i, initialize V(i) to R(i), the reward function.
- While(not converged)

```
{
in parallel for each state i,
    {
    Update V(i) ← R(i) + max( Σ_j P^a_ij • V( j ) )
                                a
    }
}
```

Computed Utilities for the 3x4 Grid World using the iterative method

-0.03	0.09	0.22	1.0
-0.15		-0.43	-1.0
-0.28	-0.40	-0.53	-0.76

Problem with this Method for Games

- There are generally too many (maybe infinitely many) states in a game to:
  - enumerate
  - compute all their utilities

Monte Carlo Method

- The Monte Carlo Method is another way to estimate utility V:

- Repeat over time:
  - Generate a *random* sequence from a state i at time t and compute the total reward R_{it} for that sequence.

  - Update V according to $\Delta V(i) = \alpha(R_{it} - V(i))$ where α is a constant identified as the step size.

Summary so far

- Dynamic Programming

$$V(i) \leftarrow R(i) + \max_{a}\left(\sum_{j} P^{a}_{ij} \cdot V(j)\right)$$

- Monte Carlo

$$\Delta V(i) = \alpha(R_{it} - V(i))$$

R_t is reward from a sequence from s_t.

Reinforcement Learning using Temporal Differences

Temporal Difference Learning

- **Through many trial runs**, adjust the observed values of V so that they more closely agree with the dynamic programming equation.

- If we encounter a **transition from i to j**, adjust V(i) so that it **better agrees** with V(j),
  using a **discount rate** $\gamma \leq 1$ which tempers the current utility of future reward.

- Temporal Difference updating rule (with η the **learning rate**:

$$\Delta V(i) = \eta \bullet (R(i) + \gamma V(j) - V(i))$$

New Summary

- Dynamic Programming

$$V(i) \leftarrow R(i) + \max_{a}\left(\sum_{j} P^{a}_{ij} \cdot V(j)\right)$$

- Monte Carlo

$$\Delta V(i) = \alpha(R_{it} - V(i))$$

R_t is reward from a sequence from state i at time t

- Temporal Difference

$$\Delta V(i) = \eta \bullet (R(i) + \gamma V(j) - V(i))$$

learning rate η, discount rate $\gamma \leq 1$

TD Update Rule

- What's "temporal" about this?

  $$\Delta V(i) = \eta \bullet (R(i) + \gamma V(j) - V(i))$$

- If we identify i as the "state at time t" and j as the "state at time t+1" the rule becomes

  $$V(i) \leftarrow V(i) + \eta(R(i) + \gamma V(j) - V(i))$$

- which resembles the **Monte Carlo update**:

  $$V(i) \leftarrow V(i) + \alpha(R_{it} - V(i))$$

- In effect, we are estimating the reward portion as

  $$R(i) + \gamma V(j)$$

  rather than as R_{it} as in Monte Carlo.

Utilities Computed by TD vs. by Dynamic Programming

This was after 1 million cycles. The values were not yet stable.

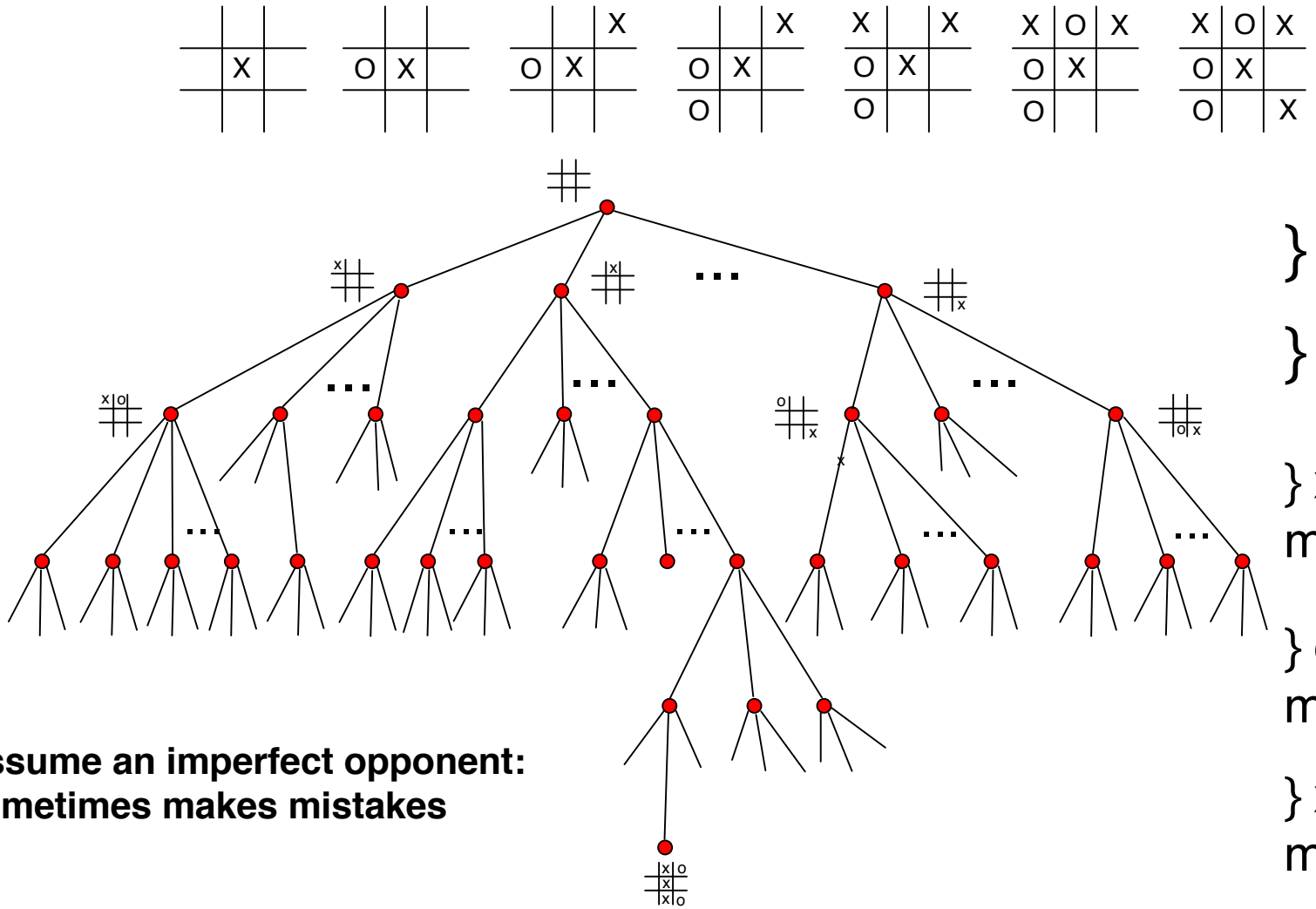
TD →	-0.01	0.05	0.14	1.0
DP →	-0.03	0.09	0.22	
	-0.07		-0.5	-1.0
	-0.15		-0.43	
	-0.11	-0.16	-0.46	-0.76
	-0.28	-0.40	-0.53	-0.76

Game Playing

- In game playing it is generally infeasible to enumerate all states.
- However, the TD updating rule can be applied to states encountered **in the course of play**.
- We can also make “exploratory moves” along the way to make the utility estimates more robust.

Example: Tic-Tac-Toe

(slide from Sutton and Barto)



} x's move

} o's move

} x's move

} o's move

} x's move

Assume an imperfect opponent: sometimes makes mistakes

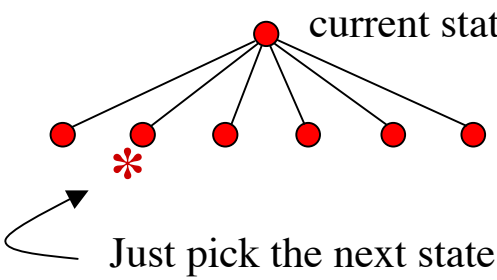
An RL Approach to Tic-Tac-Toe

(slide from Sutton and Barto)

1. Make a table with one entry per state:

State	$V(s)$ – estimated probability of winning	
	.5	?
	.5	?
⋮	⋮	
	1	win
⋮	⋮	
	0	loss
⋮	⋮	
	0	draw

2. Now play lots of games.
To pick our moves,
look ahead one step:

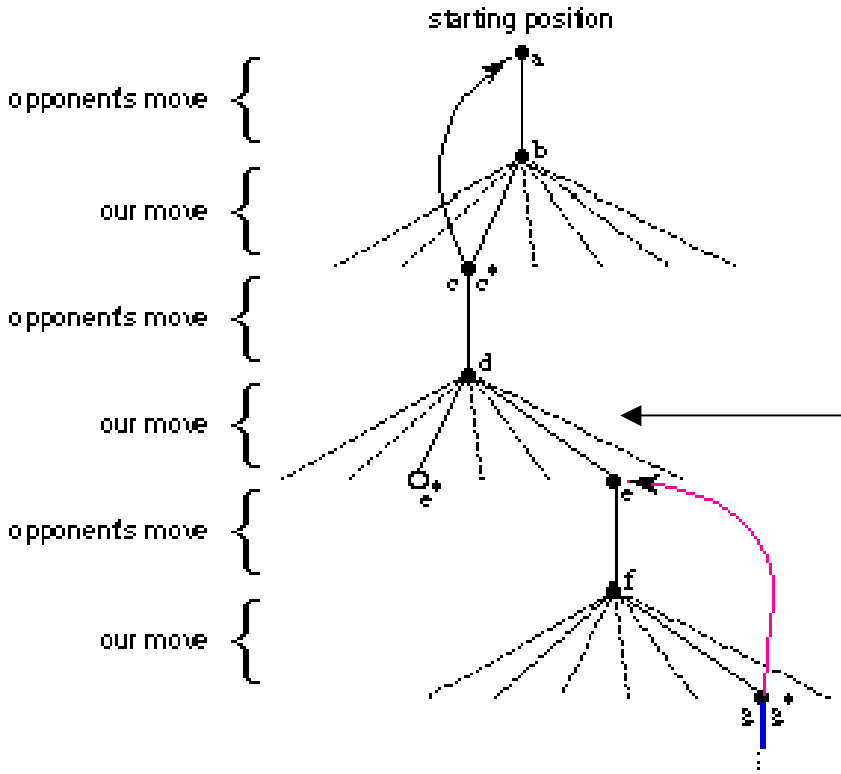


Just pick the next state with the highest estimated prob. of winning — the largest $V(s)$; a ***greedy*** move.

But 10% of the time pick a move at random; an ***exploratory move***.

RL Learning Rule for Tic-Tac-Toe

(slide from Sutton and Barto)



"Exploratory" move

s – the state before our greedy move

s' – the state after our greedy move

We increment each $V(s)$ toward $V(s')$ – a ***backup***:

$$V(s) \leftarrow V(s) + \alpha\left[V(s') - V(s)\right]$$

a small positive fraction, e.g., $\alpha = .1$

the ***step - size parameter***

More on TD method

- TD tries to train a utility function to **predict** the outcome from a state.
- The earliest known use (not by the name TD) was in Samuel's checker-playing program (1959).
- Richard Sutton expressed the general framework (1988).

Single- vs. Multi-Step Prediction

- In single-step prediction, the outcome is revealed right after the prediction.
- In multi-step prediction, the outcome is delayed for several or many steps.
- TD is applied in multi-step cases (in single-step it is identical to supervised learning).

Sutton's Derivation of TD

- Observation-outcome sequence:

  $x_1, x_2, x_3, \ldots, x_m, z$

- x_t is the observation (input) at step t
- z is the net **outcome** of the sequence
- All values are real numbers

- Say that the learner's **predictions** that estimate the final outcome z are:

  $P_1, P_2, P_3, \ldots, P_m$

Sutton's Derivation (2)

- In general, a **prediction** P_t can be a function of all preceding observations, but for simplicity it can be assumed to just depend on the current observation x_t.
- (We could always include all previous observations as "part of" the current observation.)
- P_t can be regarded as the **utility** value in the earlier slides.

Sutton's Derivation (3)

- If computed by a neural net, P_t will also depend on some **weights** w, and could be written explicitly as

$$P_t = P(x_t, w)$$

- The learning rule will indicate how to update w.

- Let Δw_t be the weight change as a result of prediction P_t .

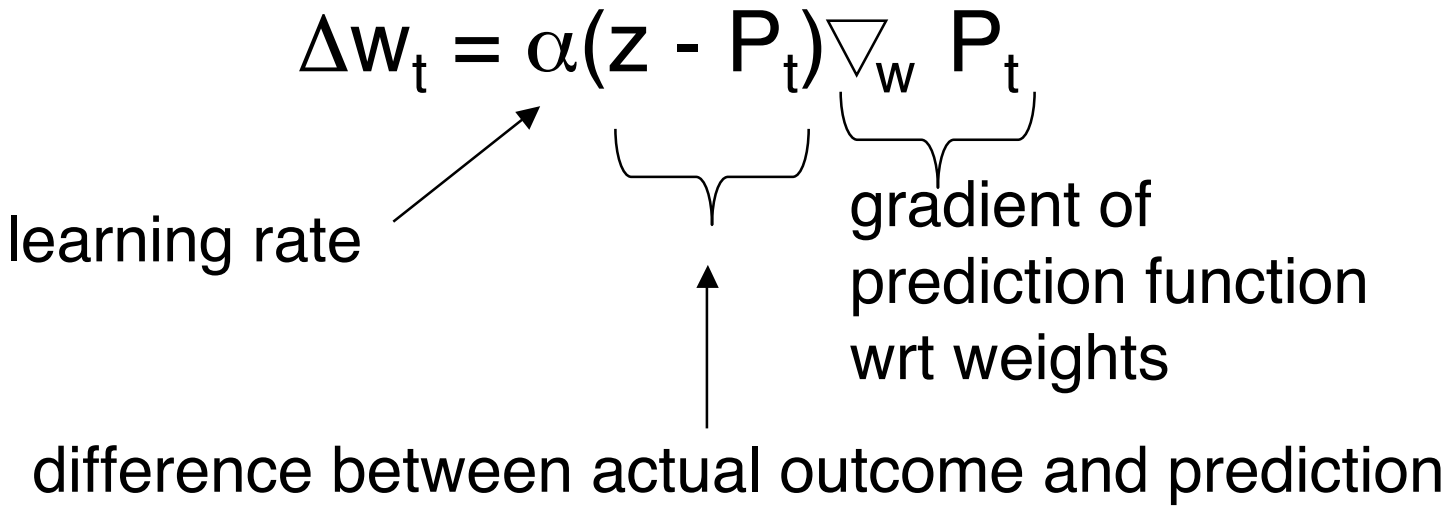
Sutton's Derivation (4)

- The **net change** in w over the entire observation sequence x_1, x_2, x_3, …., x_m, is thus:

$$\sum_t \Delta w_t$$

Sutton's Derivation (5)

- *Supervised* learning would pair each observation with the expected *final* outcome and train thus:



Sutton's Derivation (6)

- Example: If the prediction function were linear: $P_t(w, x_t) = \sum w(i) \cdot x_t(i)$ then

  $\nabla_w P_t = x_t$

  and we have the Widrow-Hoff rule:

$$\Delta w_t = \alpha(z - w^T x_t) \cdot x_t$$

$(z - w^T x_t)$: difference between outcome and prediction

x_t: gradient of prediction function

Sutton's Derivation (7)

- For an MLP network, rather than linear, the same update form can be used as with backpropagation. The gradient is just more complicated, as we know.
- The problem with supervised technique is that it **assumes knowledge of the final outcome**.
- Temporal differences remove this assumption, as shown next.

Sutton's Derivation (8)

- Represent the error in a prediction $z - P_t$ as a sum of **changes** in predictions, using "telescoping"

$$z - P_t = \sum_{k=t}^{m} \left(P_{k+1} - P_k\right) \text{ where } P_{m+1} =_{def} z.$$

(z is the final outcome)

Sutton's Derivation (9)

- Now re-express the **net weight-change** for supervised learning:

$$\sum_{t=1}^{m} \Delta w_t = \sum_{t=1}^{m} \alpha \left(z - P_t\right) \nabla_w P_t$$

Incremental change, slide 5

$$\Delta w_t = \alpha (z - P_t)\, \nabla_w P_t$$

$$= \sum_{t=1}^{m} \alpha \sum_{k=t}^{m} \left(P_{k+1} - P_k\right) \nabla_w P_t$$

from telescoping (8)

$$= \sum_{k=1}^{m} \alpha \sum_{t=1}^{k} \left(P_{k+1} - P_k\right) \nabla_w P_t$$

changing summation order,

(see next slide)

$$= \sum_{t=1}^{m} \alpha \left(P_{t+1} - P_t\right) \sum_{k=1}^{t} \nabla_w P_k$$

factoring and changing indices

The "temporal difference"

Changing Summation Order

- Outer & inner summation:

  t = 1: k = 1, 2, 3, ..., m
  t = 2: k = 2, 3, ..., m
  t = 3: k = 3, 4, ..., m
  . . .
  t = m: k = m

$$\sum_{t=1}^{m} \sum_{k=t}^{m}$$

- Same as:

  k = 1: t = 1
  k = 2: t = 1, 2
  k = 3: t = 1, 2, 3
  . . .
  k = m: t = 1, 2, 3, ..., m

$$\sum_{k=1}^{m} \sum_{t=1}^{k}$$

Sutton's Derivation (10)

- From (9), the incremental weight change can be seen as

$$\Delta w_t = \alpha\left(P_{t+1} - P_t\right) \sum_{k=1}^{t} \nabla_w P_k$$

- In other words, weight change is based on the ***difference*** between current and previous predictions, times the sum of the gradients computed at previous steps.

Sutton's Derivation (11)

- If using backprop, for example, one would need to maintain a sum of the gradient values (weight changes) from previous steps.
- The method on the previous slides is called TD(1).
- For the *linear* case, TD(1) gives the same weight changes as Widrow-Hoff would.

Sutton's Derivation (12)

- TD(1) is **generalized** to TD(λ), $0 \leq \lambda \leq 1$.
- The value of λ is a **decay factor** indicating what portion of previous weight changes are to be added in.

$$\Delta w_t = \alpha\left(P_{t+1} - P_t\right) \sum_{k=1}^{t} \lambda^{t-k} \nabla_w P_k$$

- Lower values of λ give more weight to recent predictions.

Sutton's Derivation (13)

- Of special interest is TD(0) (note $0^0 = 1$):

$$\Delta w_t = \alpha(P_{t+1} - P_t)\nabla_w P_t$$

Change in value of prediction function

Gradient of P_t

- However, Bertsekas at MIT, 1995 showed by example that the TD(0) approximation can be inferior.

Possible Use of TD in Game-Playing

- For a given state of the game, **enumerate** the possible moves.
- Evaluate P_t (prediction of a win) for **each** state resulting from a possible move.
- Choose the move for which P_t is highest.
- Occasionally choose sub-optimal moves for purposes of exploration. (Opponents do not always play optimally.)

Tic-Tac-Toe

- An example exists on knuth.cs.hmc.edu:

  /cs/cs152/ttt

  by Chen Levkovich
- It uses TD and backprop to learn to play tic-tac-toe.

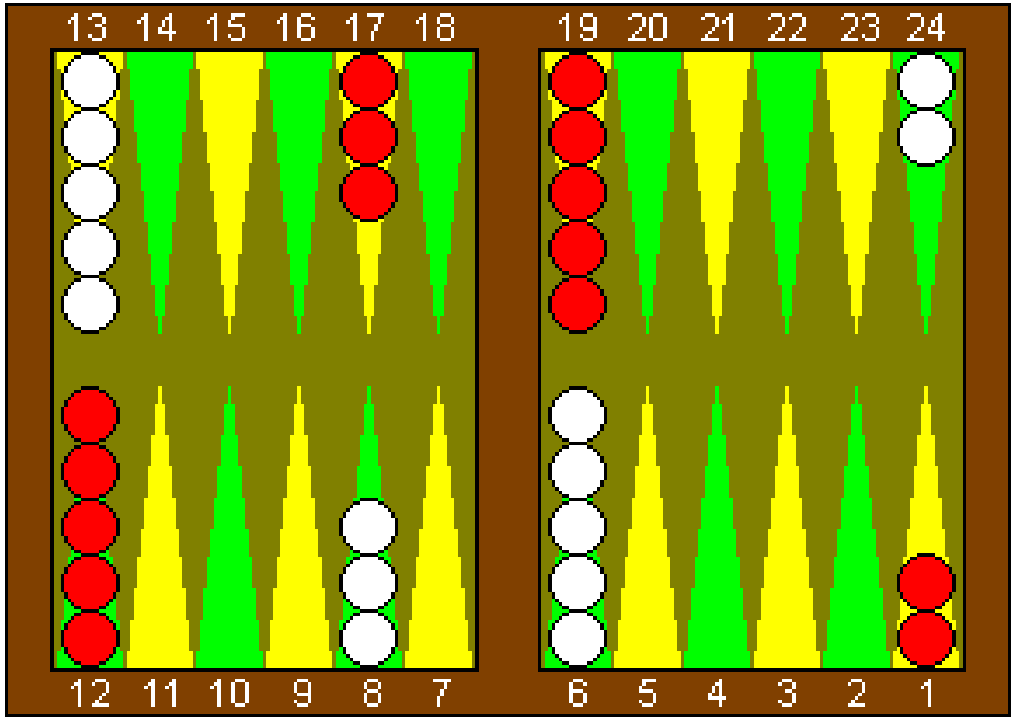
Case Study: Backgammon (Gerald Tesauro, 1995)

TD-Gammon, A Self-Teaching Backgammon Program, Achieves Master-Level Play

Gerald Tesauro
IBM Thomas J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598
(tesauro@watson.ibm.com)

See also: http://www.research.ibm.com/massive/tdl.html

Backgammon Board



The normal opening position in backgammon

Summary of Backgammon

- Players roll dice and move their checkers from points according to the numbers shown on the dice.
- The sum of the number of points moved equals the number showing on the dice.
- Landing on another player's checker captures it.

Neurogammon

- Earlier program by the same author, 1989
- Trained using supervised learning (not TD):
  - 30,000 “expert opinions”
- Eventually augmented neural network with a traditional 2-ply AI search.

TD-gammon

- 2-layer network with:
  - 1 output (whether a proposed state is good or not)
  - 198 inputs
  - 40 or 80 hidden neurons
- Weight-update rule:

$$\Delta w_t = \alpha\left(P_{t+1} - P_t\right) \sum \lambda^{t-k} \nabla_w P_k$$

Board Encoding (1)

- 4 inputs encode the number of white pieces on each of 24 board points:
  - 0000: no pieces
  - 0001: one piece
  - 0011: two pieces
  - 0111: three pieces
  - x111: >3 pieces, x = (n-3)/2 for n pieces

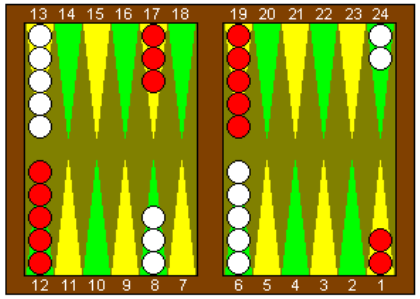


Figure 2. An illustration of the normal opening position in backgammon. TD-Gammon has sparked a near-universal conversion in the way experts play certain opening rolls. For example, with an opening roll of 4-1, most players have now switched from the traditional move of 13-9, 6-5, to TD-Gammon's preference, 13-9, 24-23. TD-Gammon's analysis is given in Table 2.

- 4x24 = 96 inputs for white + 96 for red

Board Encoding (2)

- Two more inputs encode number of pieces on the bar (n/2) for n pieces.
- Two more inputs encode the number of pieces removed (n/15).
- Two units encode whose turn to move.
- All unit inputs were roughly in the 0 to 1 range.

TD-gammon results world-class play

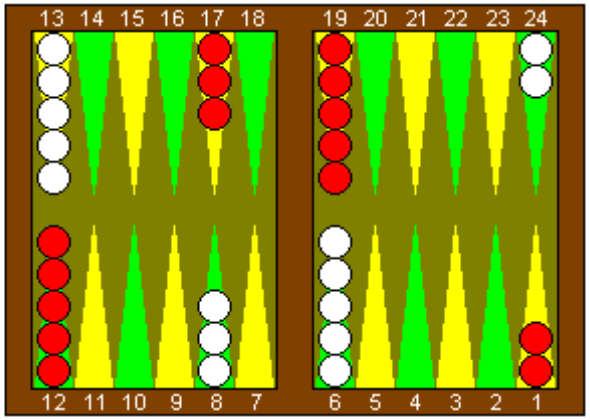
Program	Training Games	Opponents	Results
TDG 1.0	300,000	Robertie, Davis, Magriel	-13 pts/51 games (-0.25 ppg)
TDG 2.0	800,000	Goulding, Woolsey, Snellings, Russell, Sylvester	-7 pts/38 games (-0.18 ppg)
TDG 2.1	1,500,000	Robertie	-1 pt/40 games (-0.02 ppg)

Results of testing TD-gammon in play against world-class human opponents. Version 1.0 used 1-ply search for move selection; versions 2.0 and 2.1 used 2-ply search. Version 2.0 had 40 hidden units; versions 1.0 and 2.1 had 80 hidden units.

TD-gammon results

- In 1994, TD-Gammon was at the level of the best human players in the world.
- Expert players learned new strategy from the program.

Judgement superior to experts?



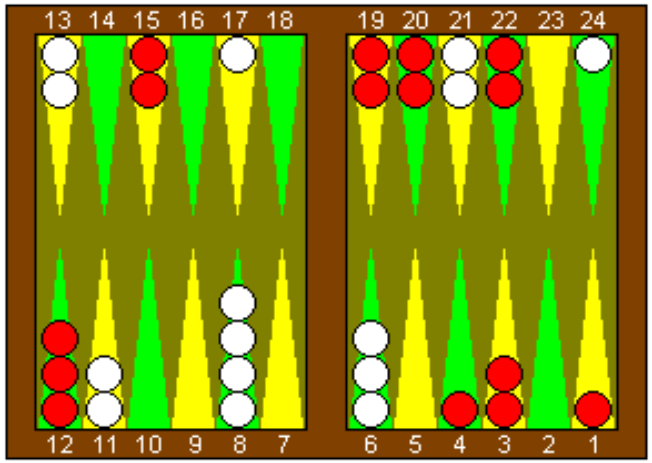
Move	Estimate	Rollout
13-9, 6-5 13-9, 24-23	−0.014 +0.005	−0.040 +0.005

An illustration of the normal opening position in backgammon.

TD-Gammon has sparked a near-universal conversion in the way experts play certain opening rolls. For example, with an opening roll of 4-1, most players have now switched from the traditional move of 13-9, 6-5 to TD-Gammon's preference, 13-9, 24-23.

TD-Gammon's analysis of the two choices: The estimated equity is the neural network's output at the 1-ply level (i.e. no lookahead). The rollout is actual outcome of playing each position out 10,000 times to completion with different random dice sequences. Standard deviation in the rollout results is approximately 0.01.

Judgement superior to experts?



Move	Estimate	Rollout
8-4*, 8-4, 11-7, 11-7	+0.184	+0.139
8-4*, 8-4, 21-17, 21-17	+0.238	+0.221

Table 3. TD-Gammon's analysis of the two choices in Figure 3. The estimated equity is the neural network's output at the 1-ply level (i.e., no lookahead). The rollout is actual outcome playing each position out 10,000 times to completion with different random dice sequences (see the appendix). Standard deviation in the rollout results is approximately 0.01.

Figure 3: A complex situation where TD-Gammon's positional judgment was apparently superior to traditional expert thinking. White is to play 4-4. The obvious human play is 8-4*, 8-4, 11-7, 11-7. (The asterisk denotes that an opponent checker has been hit.) However, TD-Gammon's choice is the surprising 8-4*, 8-4, 21-17, 21-17! TD-Gammon's analysis of the two plays is given in Table 3.

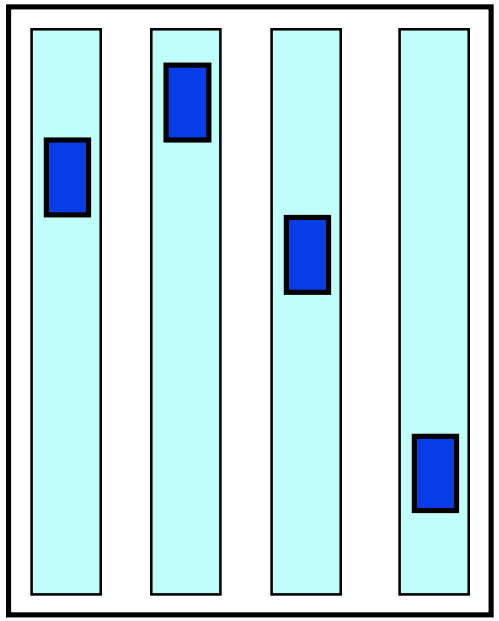
Other TD uses

- Checkers (A. Samuel)
- Go
- Othello
- Chess
- AHC (Adaptive Heuristic Critic): pole-balancing, etc. (Barto, Sutton, and Anderson)

Example: Elevator Dispatching

Crites and Barto, 1996

10 floors, 4 elevator cars



STATES: button states; positions, directions, and motion states of cars; passengers in cars & in halls

ACTIONS: stop at, or go by, next floor

REWARDS: roughly, –1 per time step for each person waiting

Conservatively about 10^{22} states

from R. S. Sutton and A. G. Barto: *Reinforcement Learning: An Introduction*

Giving Back to Biology

- A neural substrate of prediction and reward.
  Schultz W, Dayan P, Montague PR.
  Science. 1997 Mar 14; 275(5306):1593-9

Abstract: The **capacity to predict future events** permits a creature to detect, model, and manipulate the causal structure of its interactions with its environment. Behavioral experiments suggest that **learning is driven by changes in the expectations about future salient events such as rewards and punishments**. Physiological work has recently complemented these studies by identifying dopaminergic neurons in the primate whose fluctuating output apparently signals changes or errors in the predictions of future salient and rewarding events. **Taken together, these findings can be understood through quantitative theories of adaptive optimizing control** [i.e. the TD model, which the authors use].

TD Learning using a Robot

- Touretzky, Daw, and Tira-Thompson, Combining Configural and TD Learning on a Robot, Proc. 2nd Intl. Conf. on Development and Learning, 2002.
- Sony AIBO robot, model ERS-210
- Combine configural and TD learning in a classical conditioning model:
  - Solved the negative patterning problem
  - Discriminate sequences of stimuli
  - Exhibit second-order conditioning
  - Real-time interaction

Further Background of TD

- Touretzky et al. mention that, in devising TD, Barto and Sutton extended the Rescorla-Wagner model (1972) [which is the same as the Widrow-Hoff rule and thus suffers from its linear limitations].

- Rescorla and Wagner, “A Theory of **Pavlovian** conditioning”, Classical Conditioning II: Theory and Research, Black and Prokasy (eds), Appleton-Century-Crofts, 1972.