

## Reinforcement Learning

## Semi-Supervised Learning

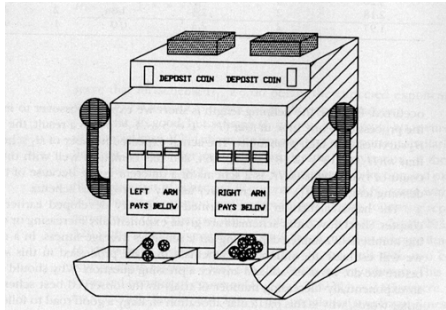
- We've focused so far on supervised learning:
  - Training examples:  $(x_1, y_1), (x_2, y_2), \dots$
- But consider a different type of learning problem, in which a robot has to learn to do tasks in a particular environment.
  - E.g.,
    - Navigate without crashing into anything
    - Locate and retrieve an object
    - Perform some multi-step manipulation of objects resulting in a desired configuration (e.g., sorting objects)

- This type of problem doesn't typically provide clear "training examples" with detailed feedback at each step.
- Rather, robot gets intermittent "rewards" for doing "the right thing", or "punishment" for doing "the wrong thing".
- Goal: To have robot (or other learning system) learn, based on such intermittent rewards, what action to take in a given situation.
- Ideas for this have been inspired by "reinforcement learning" experiments in psychology literature.

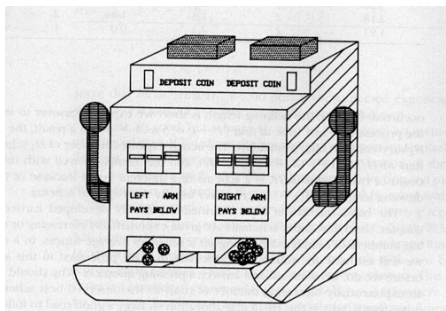
## Exploitation vs. Exploration

- *On-line* versus *off-line* learning
- On-line learning requires the correct balance between "exploitation" and "exploration"
- **Exploitation**
  - Exploit current good strategies to obtain known reward
- **Exploration**
  - Explore new strategies in hope of finding ways to increase reward

## Two-armed bandit model for exploitation and exploration with non-deterministic rewards



(From D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley 1989.)



(From D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley 1989.)

- You are given  $n$  quarters to play with, and don't know the probabilities of payoffs of the respective arms.
- What is the optimal way to allocate your quarters between the two arms so as to maximize your earnings (or minimize your losses) over the  $n$  arm-pulls ?

- Each arm roughly corresponds with a possible “strategy” to test.
- The question is, how should you allocate the next sample between the two different arms, given what rewards you have obtained so far?
- Let  $Q(a)$  = expected reward of arm  $a$ .

- Here is one algorithm for estimating  $Q(a)$ :

$$Q_0(a) = 0 \text{ for } a_1, a_2$$

Repeat for  $t = 1$  to  $n$ :

Choose arm  $a^*$  if  $Q(a^*) = \max_a Q(a)$

$$Q_{t+1}(a^*) \leftarrow Q_t(a^*) + \eta [r_{t+1}(a^*) - Q_t(a^*)]$$

where  $r_{t+1}(a)$  is the reward observed after pulling arm  $a$ , and  $\eta$  is the learning rate. (cf. perceptron learning rule)

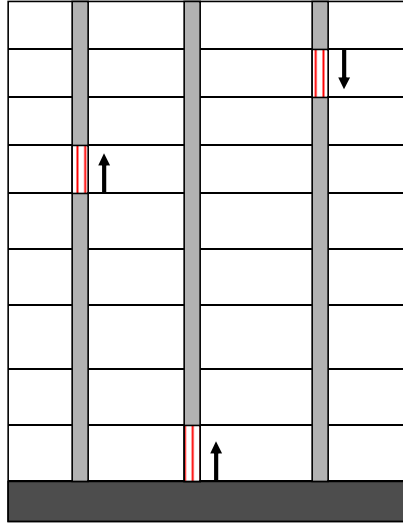
- Can generalize to “ $k$ -armed bandit”, where each arm is a possible “strategy”.
- This model was used both for GAs and for reinforcement learning.

### Applications of reinforcement learning: A few examples

- Learning to play backgammon
- Robot arm control (juggling)
- Robo-soccer
- Elevator dispatching
- Power systems stability control
- Job-shop scheduling

### Example

#### Controlling a bank of elevators



#### Sensor information:

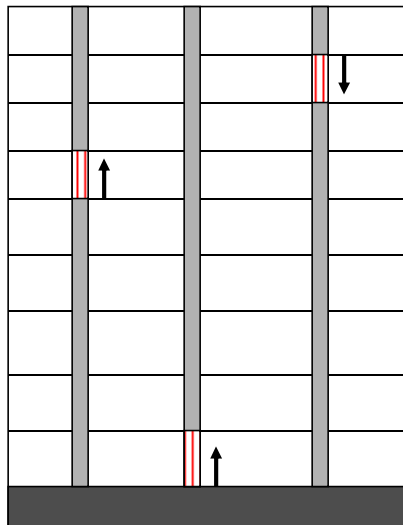
- State of up and down buttons outside the elevator on each floor (pressed or not-pressed)
- State of buttons inside elevator (what floor buttons are pressed)

#### Possible actions:

- Stop
- Go up
- Go down

### Example

#### Controlling a bank of elevators



**Constraint:** Non-empty elevator must continue in current direction and stop at all floors requested by its passengers before it reverses direction

- **Control program:**
  - At each iteration (e.g., every .5 seconds)
    1. **Examine sensor readings and memory contents**  
 (“global state” s)
    2. **Perform action for each elevator**
    3. **Receive reward**
      - -50 for every floor that has passengers waiting  
 (minimize time spent waiting for elevators)
      - -1 for each elevator that is not stopped  
 (minimize distance travelled by elevators)

- **Policy:** Mapping from space of states to space of actions.  
 Could be represented as a look-up table.
- **Optimal policy:** Policy that maximizes expected sum of  
 rewards received over lifetime of system

### Formalization

- This is a Markov decision processes:
  - Agent  $L$  only knows current state and actions available from that state.
- Agent  $L$  can
  - perceive a set  $S$  of distinct states of an environment
  - perform a set  $A$  of actions.
- Let  $r$  be a reward function, unknown to  $L$ , such that
$$r : S \times A \rightarrow \mathfrak{R}$$
$$r(s_t, a_t) = r_t$$
- Let  $\delta$  be a state transition function, unknown to  $L$ , such that
$$\delta : S \times A \rightarrow S$$
$$\delta(s_t, a_t) = s_{t+1}$$

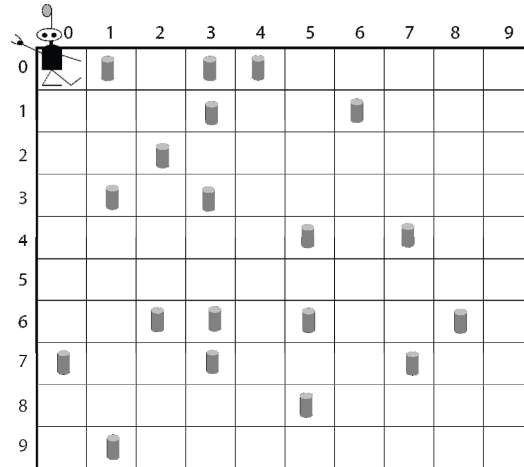
- Goal is for  $L$  to learn policy  $\pi$ ,  $\pi : S \rightarrow A$ 
$$\pi(s_t) = a_t$$
such that  $\pi$  maximizes cumulative reward.

## Example: Robby the robot

Sensors:  
N,S,E,W,C(current)

Actions:  
Move N  
Move S  
Move E  
Move W  
Move random  
Stay put  
Try to pick up can

Rewards/Penalties (points):  
Picks up can: 10  
Tries to pick up can on  
empty site: -1  
Crashes into wall: -5



## Example: Robby the robot

Sensors:  
N,S,E,W,C(current)

Example policy:

Actions:  
Move N  
Move S  
Move E  
Move W  
Move random  
Stay put  
Try to pick up can

State                      Action

Rewards/Penalties (points):  
Picks up can: 10  
Tries to pick up can on  
empty site: -1  
Crashes into wall: -5

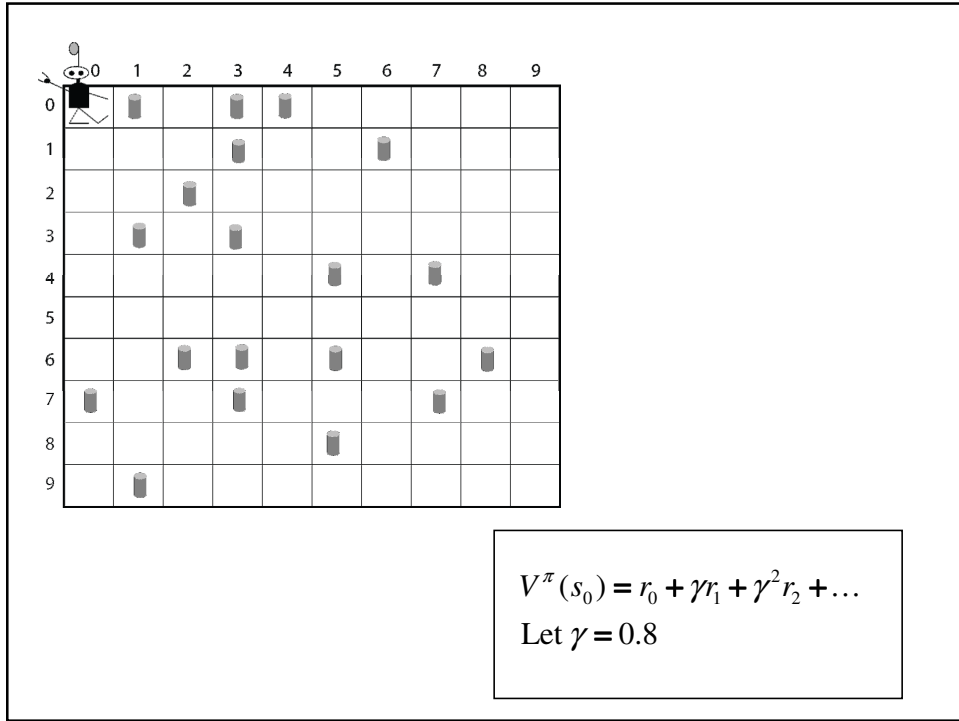
- Let  $V^\pi(s_t)$  denote the “cumulative value” of  $\pi$  starting from initial state  $s_t$ :

$$\begin{aligned} V^\pi(s_t) &= r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots \\ &= \sum_{i=0}^{\infty} \gamma^i r_{t+i} \end{aligned}$$

where  $0 \leq \gamma < 1$  is a constant that determines the relative value of delayed versus immediate rewards.

$V^\pi(s_t)$  is called the “value function” for policy  $\pi$

- Note that rewards received  $i$  times steps into the future are discounted exponentially ( by a factor of  $\gamma^i$ ).
- If  $\gamma = 0$ , we only care about immediate reward.
- The closer  $\gamma$  is to 1, the more we care about future rewards, relative to immediate reward.



- Precise specification of learning task:**

We require that the agent learn policy  $\pi$  that maximizes  $V^\pi(s)$  for all states  $s$ .

We call such a policy an *optimal policy*, and denote it by  $\pi^*$ :

$$\pi^* = \operatorname{argmax}_\pi V^\pi(s), \forall s$$
- To simplify notation, let
 
$$V^*(s) \equiv V^{\pi^*}(s)$$

## $Q$ Learning

- Hard to learn  $\pi^* : S \rightarrow A$  directly, since we don't have training data of form  $(s, a)$ .
- Only training data available to learner is sequence of rewards:

$$r(s_0, a_0), r(s_1, a_1), \dots$$

- So what should the learner  $L$  learn?

## $Q$ Learning, continued

- One possibility: learn evaluation function  $V^*(s)$ .
- Then  $L$  would know what action to take next:
  - $L$  should prefer state  $s_1$  over  $s_2$  whenever  $V^*(s_1) > V^*(s_2)$
  - Optimal action  $a$  in state  $s$  is the one that maximizes sum of  $r(s, a)$  (immediate reward) and  $V^*$  of the immediate successor state, discounted by  $\gamma$ :

$$\pi^*(s) = \operatorname{argmax}_a [r(s, a) + \gamma V^*(\delta(s, a))]$$

## $Q$ Learning, continued

- However, using  $V^*$  to obtain optimal policy  $\pi^*$  requires perfect knowledge of  $\delta$  and  $r$ , which we earlier said are unknown to  $L$ .

## $Q$ Learning, continued

- Alternative: learn evaluation function  $Q: S \times A \rightarrow \mathfrak{R}$ :

$$Q(s, a) \equiv r(s, a) + \gamma V^*(\delta(s, a))$$

- $Q(s, a)$  is the maximum discounted cumulative reward that can be achieved starting from state  $s$  and applying action  $a$  as the first action.
- In other words,  $Q(s, a)$  is the immediate reward gained when action  $a$  is taken from state  $s$ , plus the value of following the optimal policy thereafter.

## $Q$ Learning, continued

- Note that

$$V^*(s) = \max_a Q(s, a')$$

(Why is this is true?)

## $Q$ Learning, continued

- What's the difference between learning  $Q$  and learning  $V^*$ ?
- Recapping what we have discussed:

$$\pi^*(s) = \operatorname{argmax}_a [r(s, a) + \gamma V^*(\delta(s, a))]$$

$$Q(s, a) \equiv r(s, a) + \gamma V^*(\delta(s, a))$$

so,

$$\pi^*(s) = \operatorname{argmax}_a Q(s, a)$$

## $Q$ Learning, continued

- In short, if  $L$  can learn  $Q$  function, it can find optimal policy without knowledge of  $\delta$  and  $r$  !

- Since

$$\pi^*(s) = \operatorname{argmax}_a Q(s, a)$$

all  $L$  needs to do is consider each available action  $a$  in current state  $s$  and choose the action that maximizes  $Q(s, a)$ .

- This is similar to how some computer game-playing algorithms work, e.g., chess.

## $Q$ Learning, continued

- How does a learner learn this magical  $Q$  function?

- Recall that

$$V^*(s) = \max_{a'} Q(s, a')$$

- Thus we can write  $Q(s, a)$  as follows:

$$Q(s, a) = r(s, a) + \gamma \max_{a'} Q(\delta(s, a), a')$$

- Now, using this recursive definition, we can estimate  $Q(s, a)$  via iterative approximation.

## How to learn $Q$

- Let  $\hat{Q}$  denote  $L$ 's current hypothesis—an estimate for target function  $Q$ .
- $\hat{Q}$  is represented by table, whose entries are  $\hat{Q}(s,a)$
- Assume deterministic rewards and actions.
- Choose  $0 \leq \gamma < 1$ .

## Q learning algorithm

- For each  $(s, a)$ , initialize  $\hat{Q}(s,a)$  to be zero.
- Observe the current state  $s$ .
- Do forever:
  - Select an action  $a$  and execute it.
  - Receive immediate reward  $r$
  - Learn:
    - Observe the new state  $s'$
    - Update the table entry for  $\hat{Q}(s,a)$  as follows:

$$\hat{Q}(s, a) \leftarrow r + \gamma \max_{a'} \hat{Q}(s', a')$$

- $s \leftarrow s'$

## Simple illustrations of Q learning

R		
		C

C gives reward of 5 points

Wall gives reward of  
-1 points

1. What is an optimal policy?
2. Compute  $V^*(s)$  of this state, with  $\gamma = 0.8$
3. Apply Q-learning over several trials to compute  $\hat{Q}(s,a)$

Actions are selected at random.

Current state  $s$ : Trial 1, Step 1

N S E W C  
W E E W E

R		
		C

C gives reward of 5 points

Wall gives reward of  
-1 points

Trial 1, Step 1

Current state  $s$ :

NSEWC  
WEEWE

Select action  $a = \textit{Move South}$

R		
		C

C gives reward of 5 points

Wall gives reward of -1 points

Trial 1, Step 1

Current state  $s$ :

NSEWC  
WEEWE

Select action  $a = \textit{Move South}$

R		C

C gives reward of 5 points

Wall gives reward of -1 points

Trial 1, Step 1

Current state  $s$ :

NSEWC  
WEEWE

Select action  $a = \textit{Move South}$

Reward  $r = 0$

R		C

C gives reward of 5 points

Wall gives reward of -1 points

Trial 1, Step 1

Current state  $s$ :

NSEWC  
WEEWE

Select action  $a = \textit{Move South}$

Reward  $r = 0$

New State:

NSEWC  
EWEWE

R		C

C gives reward of 5 points

Wall gives reward of -1 points

**Trial 1, Step 1**

Current state  $s$ :

NSEWC  
WEEWE

Select action  $a = \text{Move South}$

Reward  $r = 0$

New State:

NSEWC  
EWEWE

R		C

C gives reward of 5 points

Wall gives reward of -1 points

Learn:

$Q(\text{WEEWE}, \text{Move South}) =$

$$r + \gamma \max_{a'} \hat{Q}(s', a') = 0 + 0.8 \times 0 = 0$$

**Trial 1, Step 2**

Current state  $s$ :

NSEWC  
EWEWE

R		C

C gives reward of 5 points

Wall gives reward of -1 points

Trial 1, Step 2

Current state  $s$ :

NSEWC  
EWEWE

Select action  $a = \textit{Move West}$

R		C

C gives reward of 5 points

Wall gives reward of -1 points

Trial 1, Step 2

Current state  $s$ :

NSEWC  
EWEWE

Select action  $a = \textit{Move West}$

R		C

C gives reward of 5 points

Wall gives reward of -1 points

Trial 1, Step 2

Current state  $s$ :

NSEWC  
EWEWE

Select action  $a = \textit{Move West}$

R		C

C gives reward of 5 points

Wall gives reward of -1 points

Trial 1, Step 2

Current state  $s$ :

NSEWC  
EWEWE

Select action  $a = \textit{Move West}$

Reward  $r = -1$

New State:

NSEWC  
EWEWE

R		C

C gives reward of 5 points

Wall gives reward of -1 points

Trial 1, Step 2

Current state  $s$ :

NSEWC  
EWEWE

Select action  $a = \text{Move West}$

Reward  $r = -1$

New State:

NSEWC  
EWEWE

R		C

C gives reward of 5 points

Wall gives reward of -1 points

Learn:

$Q(\text{EWEWE}, \text{Move West}) =$

$$r + \gamma \max_{a'} \hat{Q}(s', a') = -1 + 0.8 \times 0 = -1$$

Trial 1, Step 3

Current state  $s$ :

NSEWC  
EWEWE

R		C

C gives reward of 5 points

Wall gives reward of -1 points

Trial 1, Step 3

Current state  $s$ :

NSEWC  
EWEWE

Select action  $a = \textit{Move West}$

R		C

C gives reward of 5 points

Wall gives reward of -1 points

Trial 1, Step 3

Current state  $s$ :

NSEWC  
EWEWE

Select action  $a = \textit{Move West}$

R		C

C gives reward of 5 points

Wall gives reward of -1 points

Trial 1, Step 3

Current state  $s$ :

NSEWC  
EWEWE

Select action  $a = \textit{Move West}$

R		C

C gives reward of 5 points

Wall gives reward of -1 points

Trial 1, Step 3

Current state  $s$ :

NSEWC  
EWEWE

Select action  $a = \textit{Move West}$

Reward  $r = -1$

New State:

NSEWC  
EWEWE

R		C

C gives reward of 5 points

Wall gives reward of -1 points

Trial 1, Step 3

Current state  $s$ :

NSEWC  
EWEWE

Select action  $a = \text{Move West}$

Reward  $r = -1$

New State:

NSEWC  
EWEWE

R		C

C gives reward of 5 points

Wall gives reward of -1 points

Learn:

$Q(\text{EWEWE}, \text{Move West}) =$

$$r + \gamma \max_{a'} \hat{Q}(s', a') = -1 + 0.8 \times 0 = -1$$

Trial 1, Step 4

Current state  $s$ :

NSEWC  
EWEWE

R		C

C gives reward of 5 points

Wall gives reward of -1 points

Current state  $s$ : **Trial 1, Step 4**

NSEWC  
EWEWE

Select action  $a = \textit{Move East}$

R		C

C gives reward of 5 points

Wall gives reward of  
-1 points

Current state  $s$ : **Trial 1, Step 4**

NSEWC  
EWEWE

Select action  $a = \textit{Move East}$

	R	C

C gives reward of 5 points

Wall gives reward of  
-1 points

Trial 1, Step 4

Current state  $s$ :

NSEWC  
EWEWE

Select action  $a = \text{Move East}$

Reward  $r = 0$

	R	C

C gives reward of 5 points

Wall gives reward of -1 points

Trial 1, Step 4

Current state  $s$ :

NSEWC  
EWEWE

Select action  $a = \text{Move East}$

Reward  $r = 0$

New State:

NSEWC  
EWCCE

	R	C

C gives reward of 5 points

Wall gives reward of -1 points

Trial 1, Step 4

Current state  $s$ :

NSEWC  
EWEWE

Select action  $a = \text{Move East}$

Reward  $r = 0$

New State:

NSEWC  
EWC EE

	R	C

C gives reward of 5 points

Wall gives reward of -1 points

Learn:

$Q(\text{EWEWE}, \text{Move East}) =$

$$r + \gamma \max_{a'} \hat{Q}(s', a') = 0 + 0.8 \times 0 = 0$$

Trial 1, Step 5

Current state  $s$ :

NSEWC  
EWC EE

	R	C

C gives reward of 5 points

Wall gives reward of -1 points

Trial 1, Step 5

Current state  $s$ :

NSEWC  
EWCEE

Select action  $a = \textit{Move East}$

	R	C

C gives reward of 5 points

Wall gives reward of -1 points

Trial 1, Step 5

Current state  $s$ :

NSEWC  
EWCEE

Select action  $a = \textit{Move East}$

		C R

C gives reward of 5 points

Wall gives reward of -1 points

Trial 1, Step 5

Current state  $s$ :

<u>N</u>	<u>S</u>	<u>E</u>	<u>W</u>	<u>C</u>
E	W	C	E	E

Select action  $a = \text{Move East}$

Reward  $r = 5$

New State:

<u>N</u>	<u>S</u>	<u>E</u>	<u>W</u>	<u>C</u>
E	W	W	E	C

		C <b>R</b>

C gives reward of 5 points

Wall gives reward of -1 points

Trial 1, Step 5

Current state  $s$ :

<u>N</u>	<u>S</u>	<u>E</u>	<u>W</u>	<u>C</u>
E	W	C	E	E

Select action  $a = \text{Move East}$

Reward  $r = 5$

New State:

<u>N</u>	<u>S</u>	<u>E</u>	<u>W</u>	<u>C</u>
E	W	W	E	C

		C <b>R</b>

C gives reward of 5 points

Wall gives reward of -1 points

Learn:

$Q(\text{EWCEE}, \text{Move East}) =$

$$r + \gamma \max_{a'} \hat{Q}(s', a') = 5 + 0.8 \times 0 = 5$$

Current state  $s$ :

### Trial 2, Step 1

NSEWC  
WEEWE

R		
		C

C gives reward of 5 points

Wall gives reward of  
-1 points

Current state  $s$ :

### Trial 2, Step 1

NSEWC  
WEEWE

Select action  $a = \textit{Move South}$

R		
		C

C gives reward of 5 points

Wall gives reward of  
-1 points

Trial 2, Step 1

Current state  $s$ :

NSEWC  
WEEWE

Select action  $a = \textit{Move South}$

R		C

C gives reward of 5 points

Wall gives reward of -1 points

Trial 2, Step 1

Current state  $s$ :

NSEWC  
WEEWE

Select action  $a = \textit{Move South}$

Reward  $r = 0$

R		C

C gives reward of 5 points

Wall gives reward of -1 points

Trial 2, Step 1

Current state  $s$ :

NSEWC  
WEEWE

Select action  $a = \text{Move South}$

Reward  $r = 0$

New State:

NSEWC  
EWEWE

R		C

C gives reward of 5 points

Wall gives reward of -1 points

Trial 2, Step 1

Current state  $s$ :

NSEWC  
WEEWE

Select action  $a = \text{Move South}$

Reward  $r = 0$

New State:

NSEWC  
EWEWE

R		C

C gives reward of 5 points

Wall gives reward of -1 points

Learn:

$Q(\text{WEEWE}, \text{Move South}) =$

$r + \gamma \max_{a'} \hat{Q}(s', a') = 0 + 0.8 \times 0 = 0$

Current state  $s$ :

### Trial 2, Step 2

NSEWC  
EWEWE

R		C

C gives reward of 5 points

Wall gives reward of  
-1 points

Current state  $s$ :

### Trial 2, Step 2

NSEWC  
EWEWE

Select action  $a = \textit{Move East}$

R		C

C gives reward of 5 points

Wall gives reward of  
-1 points

Current state  $s$ :

### Trial 2, Step 2

NSEWC  
EWEWE

Select action  $a = \textit{Move East}$

	R	C

C gives reward of 5 points

Wall gives reward of  
-1 points

Current state  $s$ :

### Trial 2, Step 2

NSEWC  
EWEWE

Select action  $a = \textit{Move East}$

Reward  $r = 0$

	R	C

C gives reward of 5 points

Wall gives reward of  
-1 points

Trial 2, Step 2

Current state  $s$ :

NSEWC  
EWEWE

Select action  $a = \text{Move East}$

Reward  $r = 0$

New State:

NSEWC  
EWC EE

	R	C

C gives reward of 5 points

Wall gives reward of -1 points

Trial 2, Step 2

Current state  $s$ :

NSEWC  
EWEWE

Select action  $a = \text{Move East}$

Reward  $r = 0$

New State:

NSEWC  
EWC EE

	R	C

C gives reward of 5 points

Wall gives reward of -1 points

Learn:

$Q(\text{EWEWE}, \text{Move East}) =$

$r + \gamma \max_{a'} \hat{Q}(s', a') = 0 + 0.8 \times 5 = 4$

And so on...

- **Convergence theorem for  $Q$ -learning:**

Let  $\hat{Q}_n(s,a)$  denote the agent's hypothesis,  $\hat{Q}(s,a)$ , after the  $n$ th update in the  $Q$ -learning algorithm.

If each state-action pair is visited infinitely often, then

$\hat{Q}(s,a)$  converges to  $Q(s,a)$  as  $n \rightarrow \infty$ .

## How to choose actions?

- Naïve strategy: at each time step, choose action that maximizes  $\hat{Q}(s,a)$ .
- This **exploits** current  $\hat{Q}$  but doesn't further **explore** the state-action space (in case  $\hat{Q}$  is way off).
- Also, convergence theorem assumes that, in the limit, each state-action transition occurs infinitely often.
- Common in Q learning to use probabilistic approach:

$$P(a_i | s) = \frac{e^{\hat{Q}(s,a_i)/T}}{\sum_j e^{\hat{Q}(s,a_j)/T}}, \quad T > 0$$

- This balances exploitation and exploration in a tunable way:
  - high  $T$ : more exploration (more random)
  - low  $T$ : more exploitation (more deterministic)
- Can start with high  $T$ , and decrease it as  $\hat{Q}$  improves.

## Representation of $\hat{Q}(s,a)$

- Note that in all of the above discussion,  $\hat{Q}(s,a)$  was assumed to be a look-up table, with a distinct table entry for each distinct  $(s,a)$  pair.
- More commonly,  $\hat{Q}(s,a)$  is represented as a function (e.g., as a neural network), and the function is estimated (e.g., through back-propagation).

## Recap on Reinforcement Learning

$r : S \times A \rightarrow \mathfrak{R}$  (reward function)

$\delta : S \times A \rightarrow S$  (transition function)

$\pi : S \rightarrow A$  (policy)

$V^\pi(s_t) = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots = \sum_{i=0}^{\infty} \gamma^i r_{t+i}$  (cumulative value of  $\pi$ )

$\pi^* = \operatorname{argmax}_\pi V^\pi(s), \forall s$  (optimal policy)

$V^*(s) \equiv V^{\pi^*}(s)$  (cumulative value of optimal policy)

$Q(s,a) \equiv r(s,a) + \gamma V^*(\delta(s,a))$  Evaluation function

## Temporal Difference Learning

- *Q*-learning: Reduces difference between estimated values of a state and its immediate successor.  $\hat{Q}$

$$\hat{Q}(s, a) \leftarrow r + \gamma \max_{a'} \hat{Q}(s', a')$$

- TD-learning: Reduces difference between estimated values of a state and  $m$  descendants.

$$\hat{Q}(s, a) \leftarrow r_t + \gamma r_{t+1} + \dots + \gamma^n \max_{a'} \hat{Q}(s_{t+n}, a')$$

## Example: Learning to play backgammon

[Rules of backgammon](#)

## Complexity of Backgammon

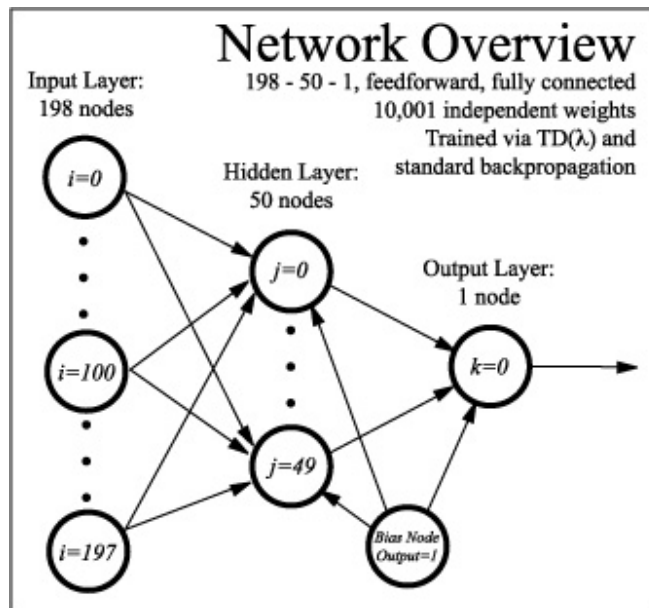
- Over  $10^{20}$  possible states.
- At each ply, 21 dice combinations, with average of about 20 legal moves per dice combination. Result is branching ratio of several hundred per ply.
- Chess has branching ratio of about 30-40 per ply.
- Brute-force look-ahead search is not practical!

## Neurogammon (Tesauro, 1989)

- Used supervised learning approach: multilayer NN trained by back-propagation on data base of recorded expert games.
- Input: raw board information (number of pieces at each location), and a few hand-crafted features that encoded important expert concepts.
- Neurogammon achieved strong intermediate level of play.
- Won backgammon championship at 1989 International Computer Olympiad. But not close to beating best human players.

## TD-Gammon (G. Tesauro, 1994)

- Program had two main parts:
  - **Move Generator:** Program that generates all legal moves from current board configuration.
  - **Predictor network:** multi-layer NN that predicts probability of winning the game from the current board configuration.
- Predictor network scores all legal moves. Highest scoring move is chosen.
- **Rewards:** Zero for all time steps except those on which game is won or lost.



- Input: 198 units
  - 24 positions, 8 input units for each position (192 input units)
    - First 4 input units of each group of 8 represent # white pieces at that position,
    - Second 4 represent # black units at that position
  - Two inputs represent who is moving (white or black)
  - Two inputs represent pieces on the bar
  - Two inputs represent number of pieces borne off by each player.

- 50 hidden units
- 1 output unit (activation represents probability that white will win from given board configuration)

Program plays against itself.

On each turn:

- Use network to evaluate all possible moves from current board configuration. Choose the move with the highest (lowest as black) evaluation. This produces a new board configuration.
- If this is end of game, run back-propagation, with target output activation of 1 or 0 depending on whether white won or lost.
- Else evaluate new board configuration with neural network. Calculate difference between current evaluation and previous evaluation.
- Run back-propagation, using the current evaluation as desired output, and the board position previous to the current move as the input.

Program	Training Games	Opponents	Results
TDG 1.0	300,000	Robertie, Davis, Magrie	-13 pts/51 games (-0.25 pfg)
TDG 2.0	800,000	Goulding, Woolsey, Snellings, Russell, Sylvester	-7 pts/38 games (-0.18 pfg)
TDG 2.1	1,500,000	Robertie	1 pt/10 games (-0.02 pfg)

**Table 1.** Results of testing TD-Cammon 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 60 hidden units.

- From Sutton & Barto, *Reinforcement Learning: An Introduction*:

“After playing about 300,000 games against itself, TD-Gammon 0.0 as described above learned to play approximately as well as the best previous backgammon computer programs.”

“TD-Gammon 3.0 appears to be at, or very near, the playing strength of the best human players in the world. It may already be the world champion. These programs have already changed the way the best human players play the game. For example, TD-Gammon learned to play certain opening positions differently than was the convention among the best human players. Based on TD-Gammon’s success and further analysis, the best human players now play these positions as TD-Gammon does (Tesauro, 1995).”

- TD-gammon is probably the most famous success in reinforcement learning to date.
- Why hasn’t its success translated into widespread application of TD-learning?
- Why hasn’t TD-learning worked well even on other complex games?

## “Why did TD-gammon work” (Pollack and Blair, 1997)

- Tried same setup as Tesauro, but instead of TD-learning, used simple “hill-climbing” to train weights:
  1. Create a “mutant” network by adding Gaussian noise to the current network’s weights.
  2. Play the current network against the mutant network for a number of games.
  3. If the mutant wins more than half the games, select it for the next generation.

## Refinement of algorithm

- Play game in pairs, with each player (current and mutant) going first on one of the games in the pair.
- If mutant wins more than 3/4 of the games, then (rather than replacing current by mutant):
$$current \leftarrow 0.95 * current + 0.05 * mutant$$
(Idea is to change the weights more slowly.)
- “Anneal” as follows:
  - After 10,000 generations, require 5/6 wins from mutant for weights to change.
  - After 70,000 generations, require 7/8 wins from mutant

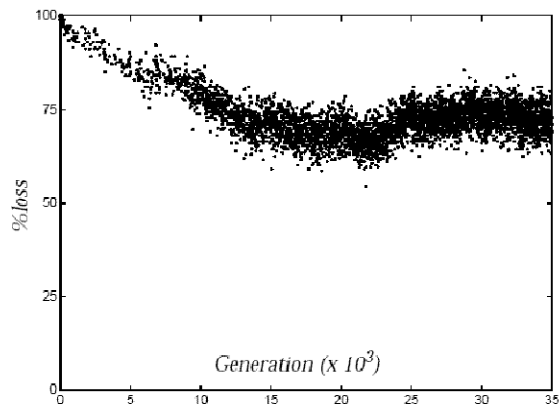


Figure 1. Percentage of losses of our first 35,000 generation players against PUBEVAL. Each match consisted of 200 games.

Best players win 45% of the time against PUBEVAL (a program comparable with TD-gammon).

### Their interpretation of these results

- TD-learning not necessary for success of TD-gammon
- Success came from “set-up of co-evolutionary self-play biased by the dynamics of backgammon”.
- “The learner is embedded in an environment which responds to its own improvements in a never-ending spiral.”
- You can play with their program at <http://www.demo.cs.brandeis.edu>

## Applying RL to Robocup Soccer

Robocup soccer video (non-learning robots):

<http://www.youtube.com/watch?v=zXnJQRAnZT0&mode=related&search=>

Learning (simulated) robots for “Keepaway” (Stone, Sutton, and Kuhlmann, 2005):

<http://www.cs.utexas.edu/~AustinVilla/sim/keepaway/>

“Keepaway is a challenging machine learning task for several reasons:

- The state space is far too large to explore exhaustively;
- Each agent has only partial (and noisy) state information;
- The action space is continuous;
- Multiple teammates need to learn simultaneously.”

## Keepaway

- Two teams: *Keepers* and *Takers*
  - Keepers learn, Takers have fixed policy
- Keepers try to keep ball for as long as possible
- Takers try to take ball
- When Takers take ball, episode ends, and a new episode (with random positions) is started.

## States

- Location of ball
- $dist(K_1, C); dist(K_2, C); dist(K_3, C);$
- $dist(T_1, C); dist(T_2, C);$
- $dist(K_1, K_2); dist(K_1, K_3);$
- $dist(K_1, T_1); dist(K_1, T_2);$
- $Min(dist(K_2, T_1), dist(K_2, T_2));$
- $Min(dist(K_3, T_1), dist(K_3, T_2));$
- $Min(ang(K_2, K_1, T_1), ang(K_2, K_1, T_2));$
- $Min(ang(K_3, K_1, T_1), ang(K_3, K_1, T_2)).$

## Primitive Actions

- HoldBall
- PassBall(k)
- GetOpen
- GoToBall
- BlockPass

## Macro Actions

- Receive
- Pass*k*ThenReceive
- Holdball

## RL Algorithm

Initialize  $Q(s, a)$  arbitrarily and  $e(s, a) = 0$  for all  $s, a$ .  
Repeat (for each episode):

  Initialize  $s$

  Choose  $a$  from  $s$  using policy derived from  $Q$

  Repeat (for each step of episode):

    Take action  $a$ , observe reward  $r, s'$

    Choose  $a'$  from  $s'$  using policy derived from  $Q$

$\delta \leftarrow r + \gamma Q(s', a') - Q(s, a)$

$e(s, a) \leftarrow e(s, a) + 1$

    For all  $s, a$ :

$Q(s, a) \leftarrow Q(s, a) + \alpha \delta e(s, a)$

$e(s, a) \leftarrow \gamma \lambda e(s, a)$

$s \leftarrow s'; a \leftarrow a';$

  until  $s$  is terminal

$\alpha$  = learning rate

**Eligibility Trace:**

$e(s, a)$  encodes how often and how recently  $(s, a)$  has been visited in the past.

If  $e(s, a)$  is high, give more “credit” to  $(s, a)$  for the error  $\delta$ .

## Nondeterministic Rewards and Actions

- $r(s, a)$  and  $\delta(s, a)$  may have probabilistic outcomes.
- Nondeterministic Markov decision process

- Redefine  $V^\pi$  as expected value:

$$V^\pi(s_t) = E\left[\sum_{i=0}^{\infty} \gamma^i r_{t+i}\right]$$

- Redefine  $Q$  as expected value:

$$\begin{aligned} Q(s, a) &= E[r(s, a) + \gamma V^*(\delta(s, a))] \\ &= E[r(s, a)] + \gamma E[V^*(\delta(s, a))] \\ &= E[r(s, a)] + \gamma \sum_{s'} P(s' | s, a) V^*(s') \end{aligned}$$

So we have, as before:

$$Q(s, a) = E[r(s, a)] + \gamma \sum_{s'} P(s' | s, a) \max_{a'} Q(s', a')$$

Now: how to estimate  $Q$ ?

Old rule:

$$\hat{Q}(s, a) \leftarrow r(s, a) + \gamma \max_{a'} \hat{Q}(s', a')$$

New rule:

$$\hat{Q}(s, a) = \hat{Q}(s, a) + \eta [r(s, a) + \gamma \max_{a'} \hat{Q}(s', a') - \hat{Q}(s, a)]$$