Lecture 17: Reinforcement Learning - Part 1

- The reinforcement learning problem
- Brief history and example applications
- What to learn: policies and value functions

Control Learning

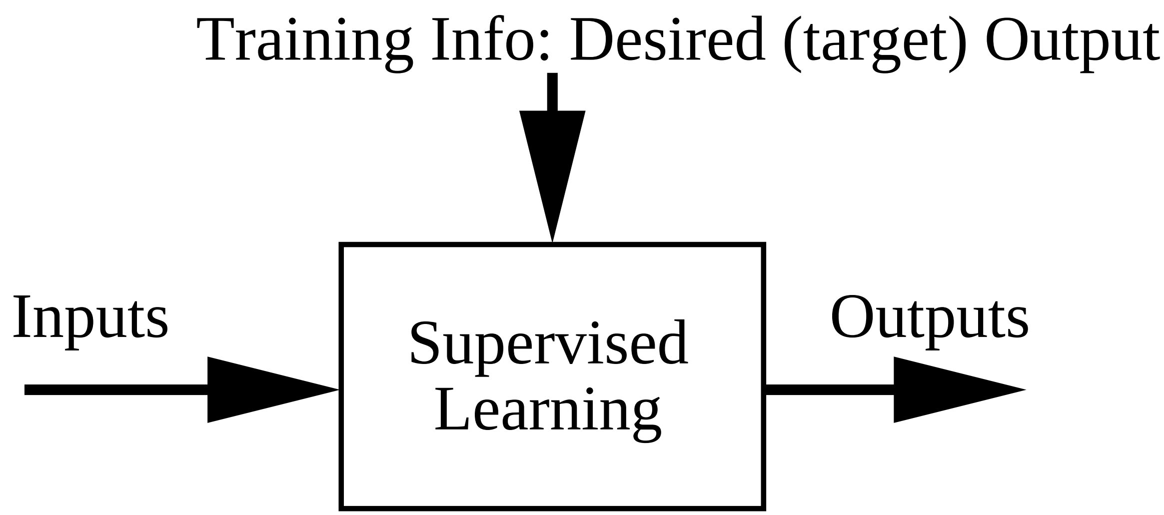
Consider *learning to choose actions*, e.g.,

- Robot learning to dock on battery charger
- Learning to choose actions to optimize factory output
- Learning to play Backgammon

Specific problem characteristics:

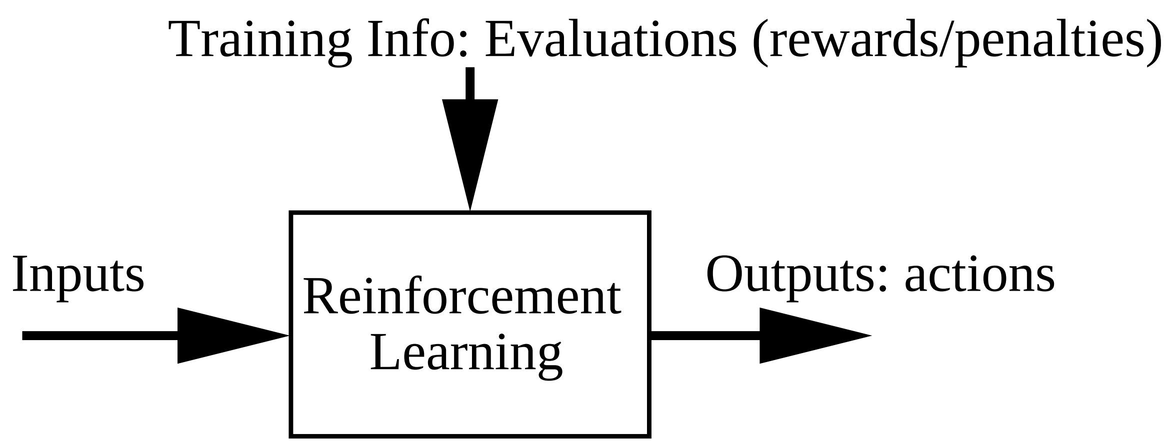
- Delayed reward
- Opportunity for active exploration
- There may not exist an adequate teacher!
- May need to learn multiple tasks using the same sensors/effectors

Supervised Learning



Error = (target output - actual output)

Reinforcement Learning (RL)



Objective: Get as much reward as possible

Key Features of RL

- The learner is not told what actions to take
- It find finds out what to do by trial-and-error search
- Possibility of delayed reward: sacrifice short-term gains for greater long-term gains
- Need to *explore* and *exploit*
- The environment is stochastic and unknown

Brief History

- Minsky's PhD thesis (1954): Stochastic Neural-Analog Reinforcement Computer
- Samuel's checkers player (1959)
- Ideas about state-action rewards from animal learning and psychology
- Dynamic programming methods developed in operations research (Bellman)
- Died down in the 70s (along with much of the learning research)
- Temporal difference (TD) learning (Sutton, 1988), for prediction
- Q-learning (Watkins, 1989), for control problems
- TD-Gammon (Tesauro, 1992) - the big success story
- Evidence that TD-like updates take place in dopamibne neurons in the brain (W.Schultz et.al, 1996)
- Currently a very active research community, with links to different fields

Success Stories

- TD-Gammon (Tesauro, 1992)
- Elevator dispatching (Crites and Barto, 1995): better than industry standard
- Inventory management (Van Roy et. al): 10-15% improvement over industry standards
- Job-shop scheduling for NASA space missions (Zhang and Dietterich, 1997)
- Dynnamic channel assignement in cellular phones (Singh and Bertsekas, 1994)
- Learning walking gaits in a legged robot (Huber and Grupen, 1997)
- Robotic soccer (Stone and Veloso, 1998) - part of the world-champion approach

All these are *large, stochastic optimal control problems*:

- Conventional methods require the problem to be simplified
- *RL just finds an approximate solution!*

An approximate solution can be better than a perfect solution to a simplified problem

Elements of RL

- *Policy*: what to do

  A mapping from states to actions, saying what action to take in each state

- *Reward*: what is good

  A numerical signal coming from the environment

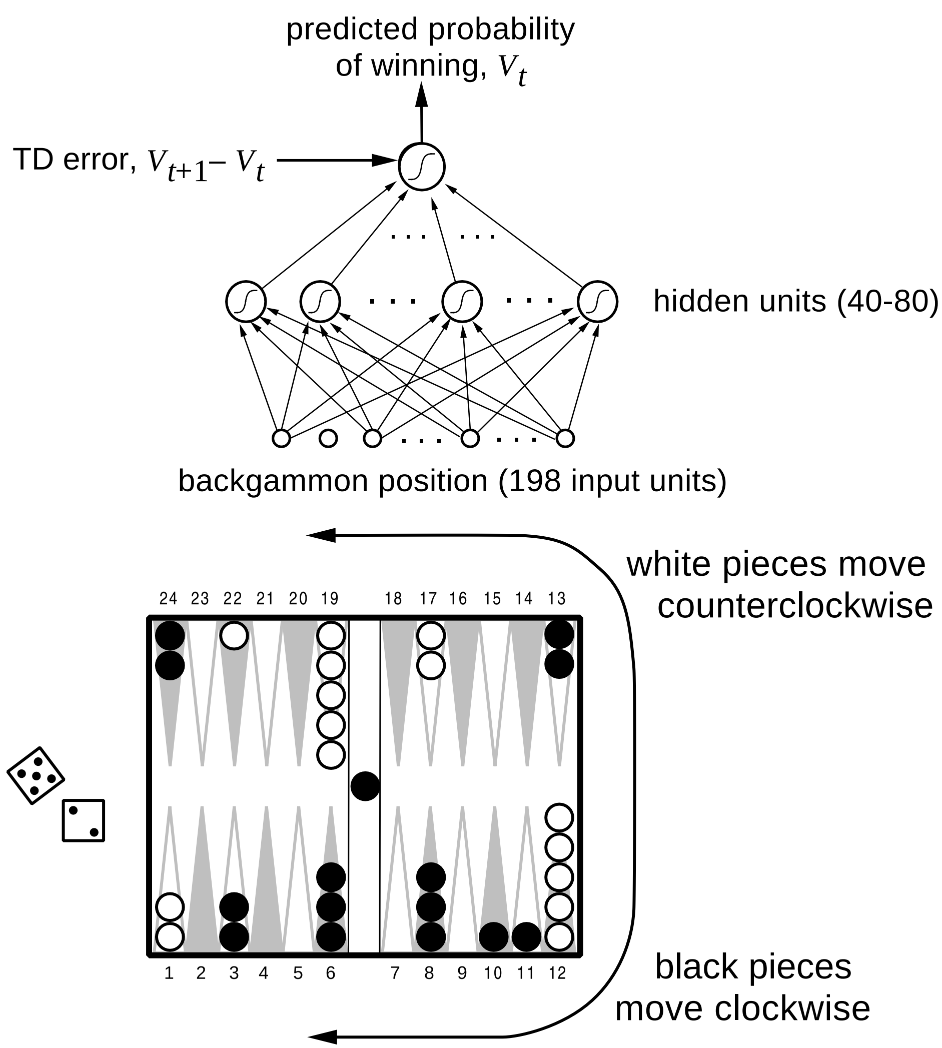
- *Value*: what is good because it *predicts* reward

  This is what we want to compute

- *Model*: what follows what

  Generally unknown, can be learned from experience

TD-Gammon (Tesauro, 1992-1995)



TD-Gammon: Training Procedure

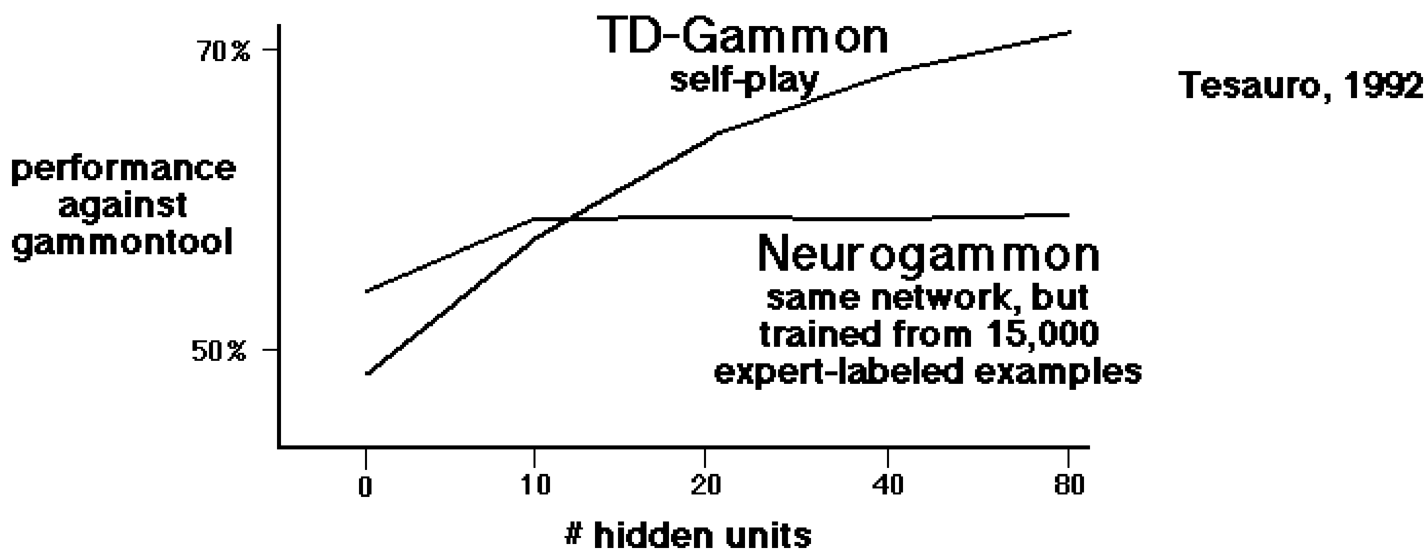
Immediate reward:

- +100 if win
- -100 if lose
- 0 for all other states

Trained by playing 1.5 million games *against itself*

Now approximately equal to best human player

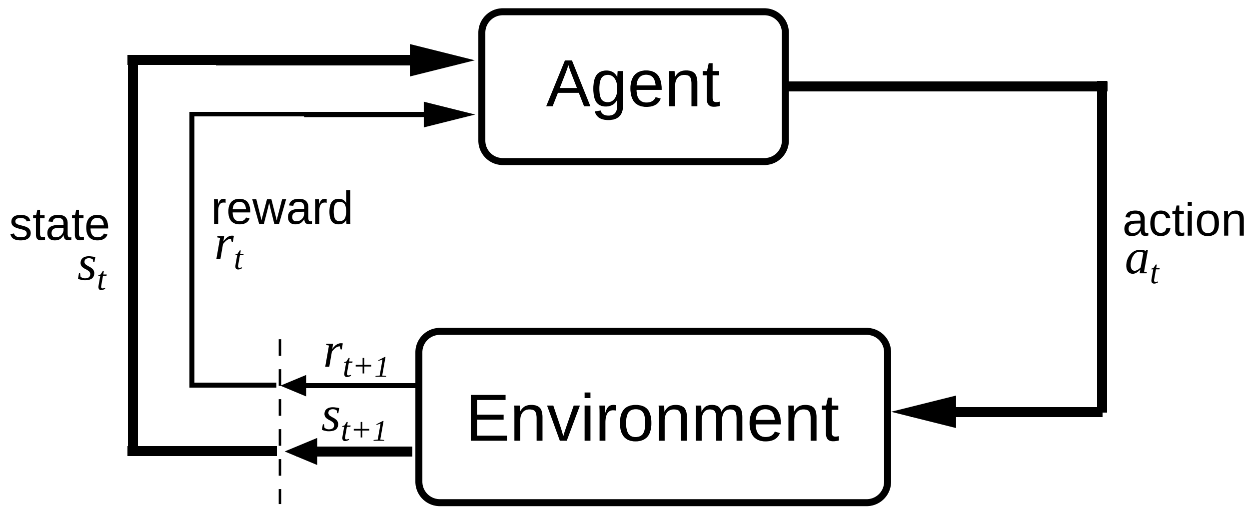
The Power of Learning from Experience



Expert examples are expensive and scarce

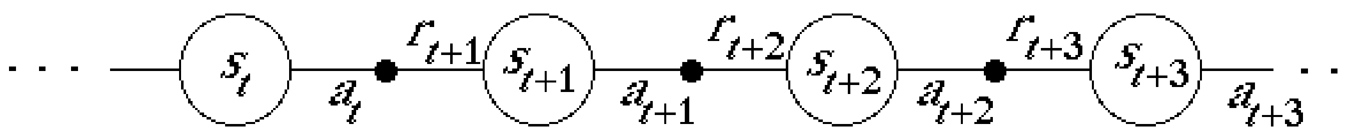
Experience is cheap and plentiful!

Reinforcement Learning Problem



- At each discrete time t, the agent observes state $s_t \in S$ and chooses action $a_t \in A$
- Then it receives an immediate reward r_{t+1} and the state changes to s_{t+1}

Markov Decision Processes (MDPs)



Assume:

- Finite set of states S (we will lift this later)
- Finite set of actions $A(s)$ available in each state s
- γ = discount factor for later rewards (between 0 and 1, usually close to 1)
- *Markov assumption:* s_{t+1} and r_{t+1} depend only on s_t, a_t and not on anything that happened before t

Models for MDPs

- r_s^a = expected value of the immediate reward if the agent is in s and does action a
- $p_{ss'}^a$ = probability of going from s to s' when doing action a

These form the *model* of the environment, and are *usually unknown*

Agent's Learning Task

Execute actions in environment, observe results, and *learn action policy* $\pi : S \rightarrow A$ that maximizes

$$E[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots]$$

from any starting state in S

where $0 \leq \gamma < 1$ is the discount factor for future rewards

Note that the target function is $\pi : S \rightarrow A$ *but we have no training examples of form* $\langle s, a \rangle$

Training examples are of form $\langle \langle s, a \rangle, r ... \rangle$

Value Function

For each possible policy π that the agent might adopt, we can define an evaluation function over states:

$$\begin{aligned} V^{\pi}(s) &= E_{\pi}\{r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots \mid s_t = s\} \\ &= E_{\pi}\left\{\sum_{i=0}^{\infty} \gamma^i r_{t+i+1} \mid s_t = s\right\} \end{aligned}$$

where $r_{t+1}, r_{t+2}, \ldots$ are generated by following policy π starting at state s

The task is to learn the *optimal policy* π^*

$$\pi^* = \arg\max_{\pi} V^{\pi}(s), (\forall s)$$

What to Learn

We might try to have agent learn the evaluation function V^{π^*} (which we write as V^*)

It could then do a lookahead search to choose best action from any state s because

$$\pi^*(s) = \arg\max_a [r(s,a) + \gamma \sum_{s'} p^a_{ss'} V^*(s')]$$

This works well if agent knows the model r, p

But when it does not know the model, it cannot choose actions this way

<u>Action-Value Function</u>

Define new function very similar to V^*

$$Q(s,a) = E_\pi\{r_{t+1} + \gamma r_{t+2} + \cdots \mid s_t = s, a_t = a\}$$

If agent learns Q, it can choose optimal action even without knowing the model!

$$\pi^*(s) = \arg\max_a Q(s,a)$$