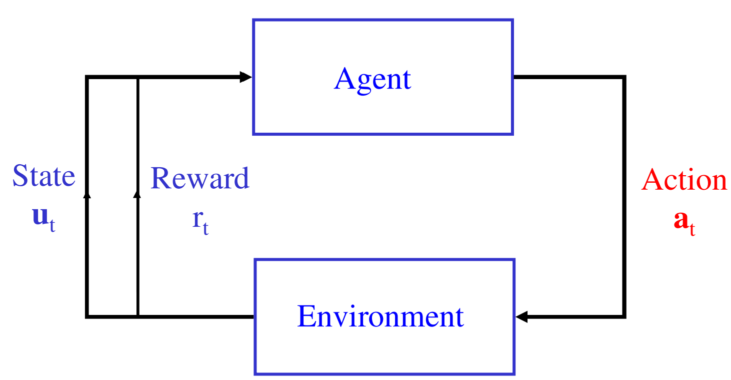
CSE 473

Chapter 21

Reinforcement Learning

The Reinforcement Learning "Agent"



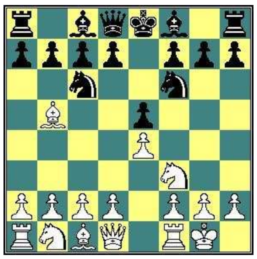
Why reinforcement learning?

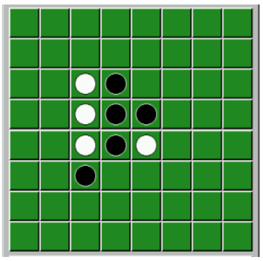
Programming an agent to drive a car or fly a helicopter is very hard!

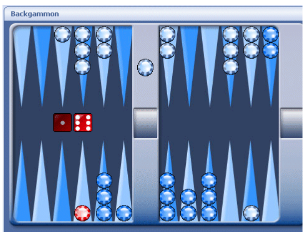
Can an agent learn to drive or fly through positive/negative rewards?

Why reinforcement learning?

Can an agent learn to win at board games through rewards?



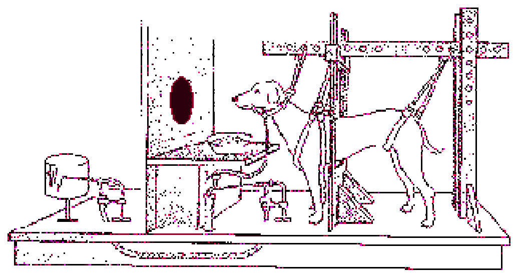




Win = large positive reward, Lose = negative
Learn evaluation function for different board positions?
Play games against itself?

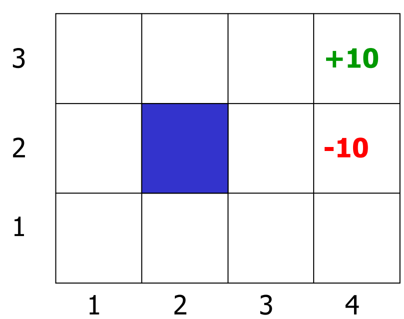
Why reinforcement learning?

Humans and animals learn through rewards
– Reinforcement learning as a model of brain function?



Pavlov's dog
Training: Bell ⇒ Food
After: Bell ⇒ Salivate

Toy Example: Agent in a Maze



Reward

Punishment

States = Maze locations (1,1), (1,2),…
Actions = Move forward, left, right, back
Rewards = +10 at (3,4), -10 at (2,4)
-1 at others (cost of moving)

Actions might be noisy

- An action may not always succeed

  E.g. 0.9 probability of moving forward, 0.1 probability divided equally among other neighboring locations

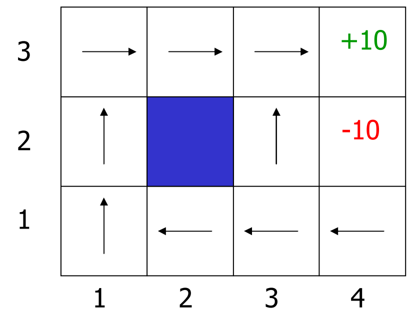
- Characterized by transition probabilities: P(next state | current state, action)

Goal: Learn a "Policy"

3	?	?	?	+10
2	?		?	-10
1	?	?	?	?
	1	2	3	4

Policy = for each state, what is the best action that maximizes my expected reward?

Goal: Learn a "Policy"



The Optimal Policy

A central problem in all these cases is learning to predict future reward

How do we do it?

Can we use supervised learning??

Predicting Delayed Rewards

- Time: $0 \leq t \leq T$ with input $u(t)$ and reward $r(t)$ (possibly 0) at each time step t
- Key Idea: Make the output $v(t)$ of supervised learner predict *total expected future reward* starting from time t

$$v(t) \approx \left\langle \sum_{\tau=0}^{T-t} r(t+\tau) \right\rangle$$

< > denotes average

Learning to Predict Delayed Rewards

- Use a set of modifiable weights $w(t)$ and predict based on all past inputs $u(t)$:

$$v(t) = \sum_{\tau=0}^{t} w(\tau)u(t-\tau)$$

(Linear neural network)

- Would like to find $w(\tau)$ that minimize:

$$\left(\sum_{\tau=0}^{T-t} r(t+\tau) - v(t) \right)^2$$

(Can we minimize this using gradient descent and delta rule?)

Yes, BUT…not yet available are future rewards

Temporal Difference (TD) Learning

- **Key Idea**: Rewrite squared error to get rid of future terms:

$$\left(\sum_{\tau=0}^{T-t} r(t+\tau) - v(t)\right)^2 = \left(r(t) + \sum_{\tau=0}^{T-t-1} r(t+1+\tau) - v(t)\right)^2$$

$$\approx \left(r(t) + v(t+1) - v(t)\right)^2$$

Temporal Difference (TD) Learning

- **TD Learning**:
  For each time step t, do:
    For all τ ($0 \leq \tau \leq t$), do:

$$v(t) = \sum_{\tau=0}^{t} w(\tau) u(t-\tau)$$

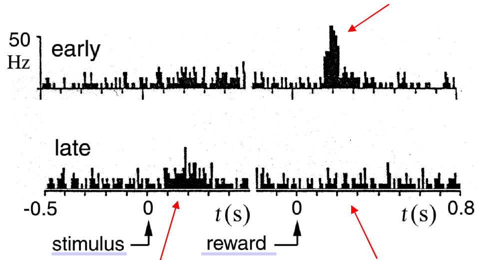
$$w(\tau) \rightarrow w(\tau) + \varepsilon [\underbrace{r(t) + v(t+1)}_{\text{Expected future reward}} - v(t)] u(t-\tau)$$

Prediction

Temporal Difference Learning in the Brain?

Activity of a Dopaminergic cell in Ventral Tegmental Area

Reward Prediction error $[r(t)+v(t+1)-v(t)]$



Before Training

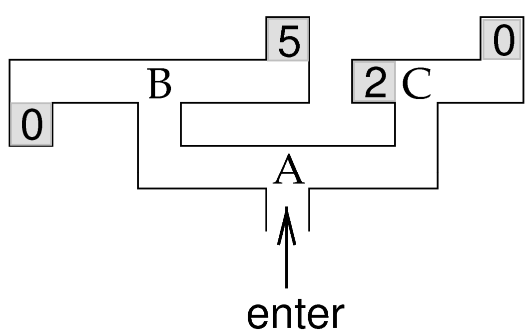
After Training

$[0+v(t+1)-v(t)]$

No error
$v(t) \approx r(t)+v(t+1)$

Selecting Actions when Reward is Delayed

Can we learn the optimal policy for this maze?

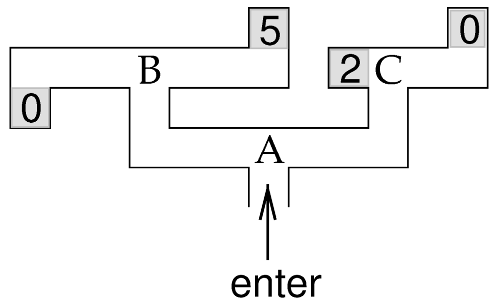


States: A, B, or C

Possible actions at any state: Left (L) or Right (R)

If you randomly choose to go L or R (random "policy"), what is the *value v of each state*?

Policy Evaluation



For random policy:

$$v(B) = \frac{1}{2}\cdot 0 + \frac{1}{2}\cdot 5 = 2.5$$

$$v(C) = \frac{1}{2}\cdot 2 + \frac{1}{2}\cdot 0 = 1$$

$$v(A) = \frac{1}{2}\cdot v(B) + \frac{1}{2}\cdot v(C) = 1.75$$

(Location, action) $\rightarrow$ new location

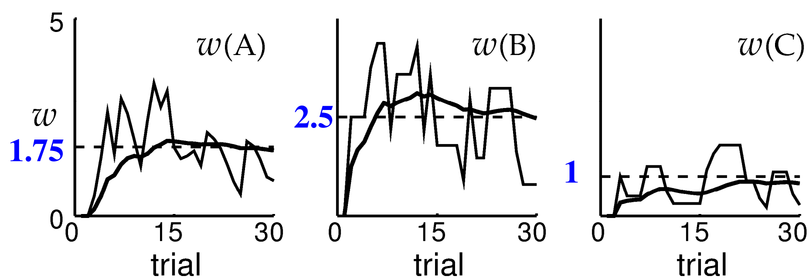
$(u,a) \rightarrow u'$

Use output $v(u) = w(u)$

Can learn this using TD learning:

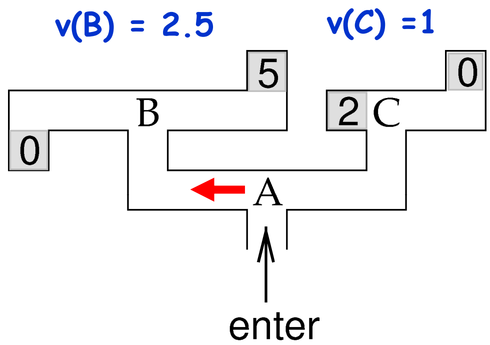
$$w(u) \rightarrow w(u) + \varepsilon[r_a(u) + v(u') - v(u)]$$

Maze Value Learning for Random Policy



Once I know the values, I can pick the action that leads to the higher valued state!

Selecting Actions based on Values



Values act as surrogate immediate rewards ⇒ Locally optimal choice leads to globally optimal policy

Related to *Dynamic Programming*

Q learning

Simple method for action selection based on action values (or Q values) Q(*u*,*a*) where *u* is a state and *a* is an action

1. Let *u* be the current state. Select an action *a* according to:

$$P(a) = \frac{\exp(\beta Q(u,a))}{\sum_{a'} \exp(\beta Q(u,a'))}$$

2. Execute *a* and record new state *u'* and reward *r*. Update Q:

$$Q(u,a) \rightarrow Q(u,a) + \varepsilon(r + \max_{a'} Q(u',a') - Q(u,a))$$

3. Repeat until an end state is reached

Reinforcement Learning Applications

Example: Flying a helicopter via reinforcement learning (videos)
(work of Andrew Ng, Stanford)

http://ai.stanford.edu/~ang/