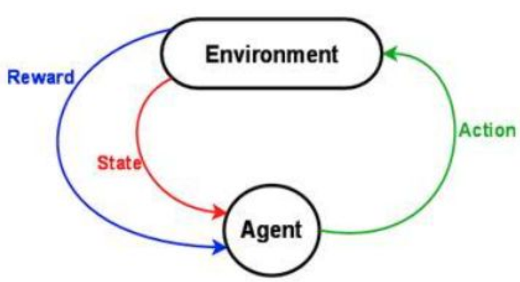
High Level Applications for RL

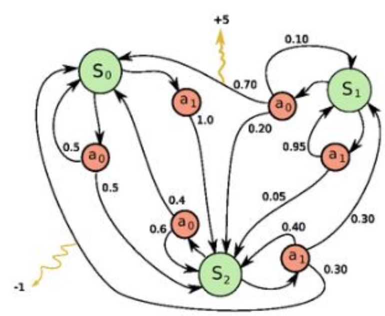
- Currently Proficient: Video Game AI's and basic reward tasks
- In Progress:
    - Self driving cars
    - Autonomous robots (manufacturing)
    - Finance
    - Various other autonomy tasks

Markov Decision Process

- **Markov Decision Process (MDP).** We assume that reward function and transition probabilities are known and available to the robot. More specifically, we are provided with
    - a set of states S
    - a set of actions A
    - a transition probability function $T(s, a, s') = p(s'|a, s)$
    - a reward function $R(s, a, s')$ (or just $R(s')$)

- **Reinforcement Learning** The reward function and transition probabilities are unknown (except for their form), and the robot only knows
    - a set of states S
    - a set of actions A





3 states, S1, S2, S3 (overheat)
$A = \{A_{safe,} A_{risk}\}$
$S = \{S_{cold}, S_{warm}, S_{hot}\}$
Transition probabilities: always half and half
Reward Function

S, a S'	S_cold	S_warm	S_hot
S_cold, A_safe	+1		
S_cold, A_risk	+3	+3	
S_warm, A_safe	+1	+1	
S_warm, A_risk		+3	-10

• If T is deterministic, this is just a graph problem
• Present each of the varaibles in the context of a simple example, maybe a grid
• Various formulations of MDPs

Utility Function/Discounted Utility/Policy

- *Method of aggregating rewards in order to quantify success*
- "What we want to optimize"
- "Why discounted utility necessary?"
    - To make an infinite sum finite

$$U([s_0, s_1, s_2 \dots]) = R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \dots$$
$$= \sum_{t=0}^{\infty} \gamma^t R(s_t) \leq \sum_{t=0}^{\infty} \gamma^t R_{max} = \frac{R_{max}}{1-\gamma}$$

Value Iteration Algorithm

- $V^*(s)$ – The value of state s, i.e., the expected utility of starting in state s and acting optimally thereafter.
- $Q^*(s, a)$ – The Q value of state s and action s. It is the expected utility of starting in state s, taking action a and acting optimally thereafter.
- $\pi^*(s)$ –The optimal policy. $\pi^*(s)$ specifies the action we should take in state s. Following policy π^* would, in expectation, result in the highest expected utility (see Figure 44).

The equations:

$$V^*(s) = \max_a Q^*(s,a) = Q^*(s,\pi^*(s)) \tag{313}$$

$$Q^*(s,a) = \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V^*(s')] \tag{314}$$

$$V^*(s) = \max_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V^*(s')] \tag{315}$$

$$= \sum_{s'} T(s,\pi^*(s),s')[R(s,\pi^*(s),s') + \gamma V^*(s')] \tag{316}$$

The algorithm:

- Start with $V_0^*(s) = 0$, for all $s \in S$
- Given V_i^*, calculate the values for all states $s \in S$ (depth $i+1$):

$$V_{i+1}^*(s) = \max_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V_i^*(s')]$$

Value Iteration Example:

0 (P1)	End (+100)
0 (P2)	Lava (-10)
0 (P3)	Start (0)

Each square is $V_i(s)$

Discount factor = 1 (no discount)

Iteration (i)	Start	P1	P2	P3	Lava
0	0	0	0	0	0
1	0	+100	0	0	+100
2	+90 (to Lava)	+100	+100	0	+100
3	+90 (to Lava)	+100	+100	+100	+100
4	+100 (to P3)	+100	+100	+100	+100

Discount factor 0.5:

Iteration (i)	Start	P1	P2	P3	Lava
0	0	0	0	0	0
1	0	+100	0	0	+100
2	+40 (to Lava) [+50 - 10]	+100	+50	0	+100
3	+40	+100	+50	+25	+100
4	+40 **(Lava)**	+100	+50	+25	+100

• Q_4*(Start, West) would be 12.5 so we go through lava

• Q-value iteration, same concept

The Q-Value Iteration Algorithm

- Start with $Q_0^*(s,a) = 0$ for all $s \in S$, $a \in A$.
- Given $Q_i^*(s,a)$, calculate the q-values for all states (depth $i+1$) and for all actions a:

$$Q_{i+1}^*(s,a) = \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma \max_{a'} Q_i^*(s',a')]$$

Reinforcement Learning

• Don't know R or T (or impossible to evaluate V* immediately)

• Model-based vs model-free, learning T/R vs not

• Sample empirically for model-based

$$T(s,a,s') = \frac{count(s,a,s')}{\sum_{s'} count(s,a,s')}$$

$$R(s,a,s') = \frac{\sum_t R_t(s,a,s')}{count(s,a,s')}$$

,

Model-Free Stuff

• Sample-based estimation suffers from low sample sizes
• Solution: maintaining exponential running average

$$\bar{x}_n = \frac{x_n + (1-\alpha) * x_{n-1} + (1-\alpha)^2 * x_{n-2}}{1 + (1-\alpha) + (1-\alpha^2) + \dots}$$

$$\bar{x}_n = \alpha * x_n + (1-\alpha) * \bar{x}_{n-1}$$

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha[sample]$$

$$sample = R(s,a,s') + \gamma \max_{a'} Q_i(s',a')$$

• Overall Q-learning algorithm and relationship to gradient descent (convergence requirements)

Exploration/Exploitation

• ε randomization, ε probability of taking random action, (1-ε) probability following our Q
• ε high to account for noisy estimates at the beginning, decrease over time

Deep Learning

• Playing Breakout, pixel representation (need 2 frames for directional velocity comp.)
• Can't represent in table, too many states

Useful Links:

Atari Paper
Helpful Deep Learning Introduction

Videos:
https://youtu.be/xOCurBYI_gY?t=15m10s "Tetris AI gets philosophical"
https://www.youtube.com/watch?v=V1eYniJ0Rnk "Google AI is *smart*"
https://www.youtube.com/watch?v=6QuMyq85__A "Go to the pink dot"