Machine Learning in Business
John C. Hull

Chapter 7

Reinforcement Learning

Reinforcement Learning

- Reinforcement learning is concerned with finding a strategy for taking a series of decisions rather than just one
- The environment is usually changing unpredictably

Rewards and costs

- There are rewards and costs and the algorithm tries to maximize expected rewards (net of costs) in its interaction with the environment
- Exploitation vs. Exploration (Should you choose best decision based on evidence to date or try something new?)
- Action evaluated by the sum of expected rewards (net of costs) that come after it (possibly discounted)

A Simple example: K-armed bandits

- This is like a one-armed bandit except that you have to choose between K levers.
- Lever k provides a return from a normal distribution with mean m_k and standard deviation 1
- Objective is to maximize return over a large number of trials

Strategy

- You keep records of the average return from choosing each lever
- At each turn you have to decide between
    - Choose the lever that has given you the best average return so far (the "greedy action")
    - Try out a new action
- The first choice is exploitation; the second is exploration.
- We can choose a parameter ε equal to the probability of exploration
- Exploitation maximizes the immediate expected return but exploration may do better in the long run

The Math (page 149)

- Suppose that lever k has been chosen $n-1$ times and the total reward on the jth time it is chosen is R_j
- Expected reward is

$$Q_k^{old} = \frac{1}{n-1}\sum_{j=1}^{n-1} R_j$$

- If kth lever is chosen for the nth time and produces a reward R_n

$$Q_k^{new} = \frac{1}{n}\sum_{i=1}^{n} R_i = Q_k^{old} + \frac{1}{n}\left(R_n - Q_k^{old}\right)$$

Example: Table 7.1, 4 bandits (m_1=1.2, m_2=1, m_3=0.8, m_4=1.4); ε = 0.1

Trial	Decision	Chosen Action	Value	Action 1 (stats) Ave	Action 1 (stats) Nobs	Action 2 (stats) Ave	Action 2 (stats) Nobs	Action 3 (stats) Ave	Action 3 (stats) Nobs	Action 4 (stats) Ave	Action 4 (stats) Nobs	Gain per trial
				0		0		0		0		
1	Exploit	1	1.293	1.293	1	0.000	0	0.000	0	0.000	0	1.293
2	Exploit	1	0.160	0.726	2	0.000	0	0.000	0	0.000	0	0.726
3	Exploit	1	0.652	0.701	3	0.000	0	0.000	0	0.000	0	0.701
4	Explore	2	0.816	0.701	3	0.816	1	0.000	0	0.000	0	0.730
50	Exploit	1	0.113	1.220	45	-0.349	3	0.543	2	0.000	0	1.099
100	Exploit	4	2.368	1.102	72	0.420	6	0.044	3	1.373	19	1.081
500	Explore	3	1.632	1.124	85	1.070	17	0.659	11	1.366	387	1.299
1000	Exploit	4	2.753	1.132	97	0.986	32	0.675	25	1.386	846	1.331
5000	Exploit	4	1.281	1.107	206	0.858	137	0.924	130	1.382	4527	1.345

When ε = 0.01 (Table 7.2)

Trial	Decision	Chosen Action	Value	Action 1 (stats) Ave	Action 1 (stats) Nobs	Action 2 (stats) Ave	Action 2 (stats) Nobs	Action 3 (stats) Ave	Action 3 (stats) Nobs	Action 4 (stats) Ave	Action 4 (stats) Nobs	Gain per trial
				0		0		0		0		
1	Exploit	1	1.458	1.458	1	0.000	0	0.000	0	0.000	0	1.458
2	Exploit	1	0.200	0.829	2	0.000	0	0.000	0	0.000	0	0.829
3	Exploit	1	2.529	1.396	3	0.000	0	0.000	0	0.000	0	1.396
4	Exploit	1	-0.851	0.834	4	0.000	0	0.000	0	0.000	0	0.834
50	Exploit	1	1.694	1.198	49	0.000	0	-0.254	1	0.000	0	1.169
100	Exploit	1	0.941	1.132	99	0.000	0	-0.254	1	0.000	0	1.118
500	Exploit	1	0.614	1.235	489	0.985	6	-0.182	2	0.837	3	1.224
1000	Exploit	1	1.623	1.256	986	0.902	7	-0.182	2	0.749	5	1.248
5000	Exploit	1	1.422	1.215	4952	1.022	18	0.270	8	1.148	22	1.213

When ε = 0.5 (Table 7.3)

Trial	Decision	Chosen Action	Value	Action 1 (stats) Ave	Action 1 (stats) Nobs	Action 2 (stats) Ave	Action 2 (stats) Nobs	Action 3 (stats) Ave	Action 3 (stats) Nobs	Action 4 (stats) Ave	Action 4 (stats) Nobs	Gain per trial
				0		0		0		0		
1	Exploit	1	0.766	0.766	1	0.000	0	0.000	0	0.000	0	0.766
2	Explore	1	1.257	1.011	2	0.000	0	0.000	0	0.000	0	1.011
3	Exploit	1	-0.416	0.536	3	0.000	0	0.000	0	0.000	0	0.536
4	Explore	3	0.634	0.536	3	0.000	0	0.634	1	0.000	0	0.560
50	Explore	4	0.828	1.642	17	1.140	9	0.831	9	1.210	15	1.276
100	Explore	3	2.168	1.321	47	0.968	15	0.844	16	1.497	22	1.231
500	Explore	1	0.110	1.250	86	0.922	65	0.636	72	1.516	277	1.266
1000	Explore	4	1.815	1.332	154	1.004	129	0.621	131	1.394	586	1.233
5000	Explore	3	2.061	1.265	666	0.953	623	0.797	654	1.400	3057	1.247

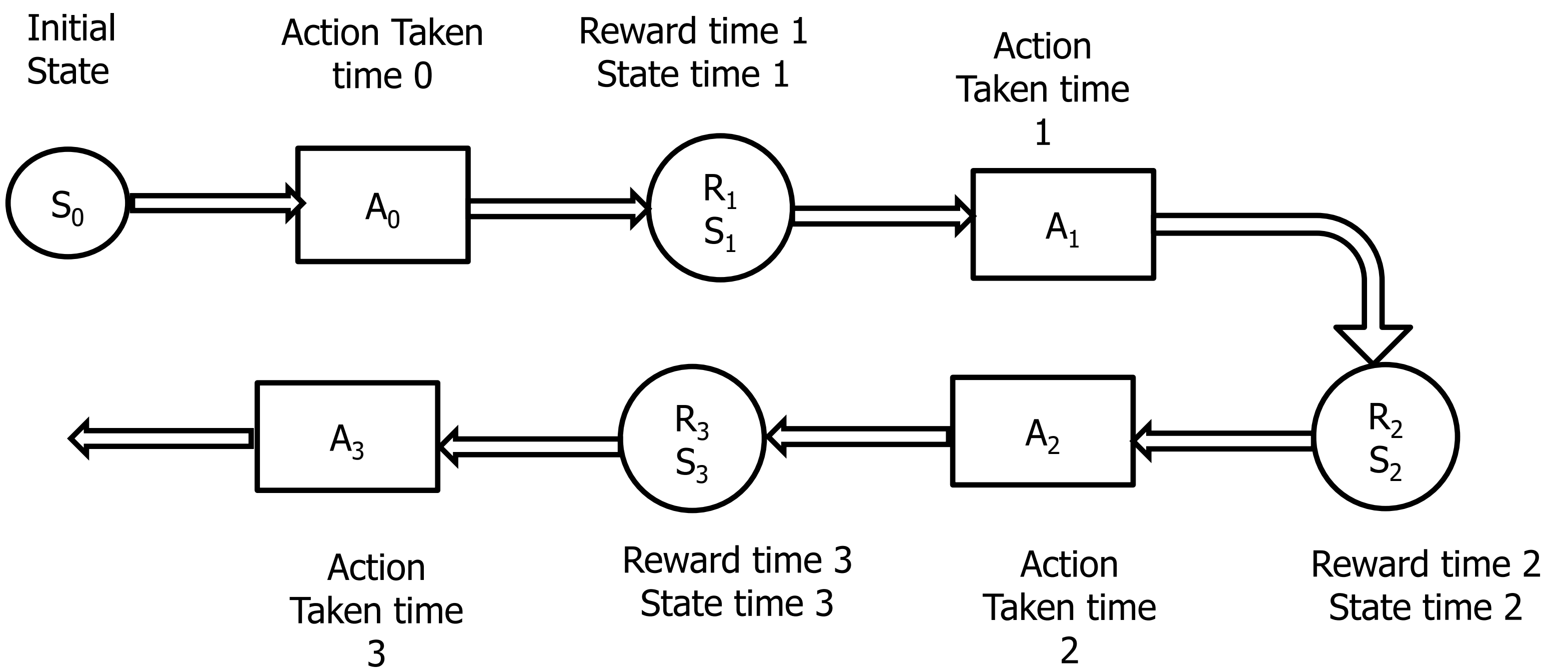
The exploration parameter ε and initial values

- It makes sense to reduce ε over time. For example we can let it decline exponentially
- The initial *Q*-values make a difference. For example if they all set equal to 2 instead of 0 there would be early exploration
- If the standard deviation of the payoff is increased, a higher value of ε would be appropriate

ε starts at 1 but has a decay factor of 0.995

Trial	Decision	Lever Chosen	Payoff	Lever 1 (stats) Q-val	Nobs	Lever 2 (stats) Q-val	Nobs	Lever 3 (stats) Q-val	Nobs	Lever 4 (stats) Q-val	Nobs	Ave Gain per trial
1	Explore	2	1.4034	0	0	1.403	1	0	0	0.000	0	1.403
2	Explore	1	0.796	0.796	1	1.403	1	0.000	0	0.000	0	1.100
3	Explore	2	0.499	0.796	1	0.951	2	0.000	0	0.000	0	0.900
4	Explore	1	0.407	0.601	2	0.951	2	0.000	0	0.000	0	0.776
5	Explore	4	0.743	0.601	2	0.951	2	0.000	0	0.743	1	0.770
50	Explore	3	-1.253	0.719	8	1.640	18	0.729	11	1.698	13	1.308
100	Explore	1	0.100	0.852	19	1.326	31	0.681	20	1.391	30	1.126
500	Exploit	4	-0.448	1.148	37	1.184	51	0.815	51	1.349	361	1.263
1000	Exploit	4	2.486	1.174	44	1.225	53	0.819	53	1.387	850	1.339
5000	Exploit	4	3.607	1.148	45	1.225	53	0.819	53	1.391	4849	1.381

The More General Model: Actions and States (Figure 7.1)



Objective

- Objective could be to maximize the expected value of rewards

$$G = R_1 + R_2 + R_3 + \ldots\ldots + R_T$$

- In some cases, particularly when there is a long or infinite horizon, the objective is to maximize the expected value of discounted rewards:

$$G = R_1 + \gamma\, R_2 + \gamma^2 R_3 + \ldots\ldots..$$

where γ is a discount factor

Updating

- Define $Q(S,A)$ as the value of being in state S and taking action A
- Suppose we have just completed the nth trial for state S and action A. Instead of

$$Q(S,A)^{new} = Q(S,A)^{old} + \frac{1}{n}\left[G - Q(S,A)^{old}\right]$$

we usually set

$$Q(S,A)^{new} = Q(S,A)^{old} + \alpha\left[G - Q(S,A)^{old}\right]$$

for some parameter α (e.g. 0.1)

A Simple Example: the game of Nim

- There is a pile of N matches. Two players take it in turns to pick up one, two, or three matches.
- Last player to pick up loses
- State is the number of matches. Action is number picked up. Reward occurs at the end (=+1 if you win and -1 if you lose)
- What is the optimal strategy?
- We can work back from the end of the game.
- What if it is your turn and there are 2, 3, 4, 5, 6,… matches?

Using Monte Carlo

- This works similarly to the 4-armed bandit example
- In that example we had 4 actions
- Here we consider state-action combinations
- Define $Q(S,A)$ as the value of being in state S when action A is taken
- We might initially set all the $Q(S,A)$ to zero
- We simulate a set of actions

Using Monte Carlo *continued*

- There is a chance (1-ε) that we choose the best action so far for any given state and ε that we randomly choose an action
- Define G as the total reward (possibly with discounting) for the complete set of actions taken in one trial
- For each $\{S,A\}$ combination that is encountered on the trial we update as follows

$$Q(S,A)^{new} = Q(S,A)^{old} + \alpha\left[G - Q(S,A)^{old}\right]$$

Nim with 8 matches in the pile initially

- Here are the initial values for the state/action combinations

Matches picked up	State (=number of matches left)						
	2	3	4	5	6	7	8
1	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0
3		0	0	0	0	0	0

Nim continued

- Opponent behaves randomly
- Reward for winning is 1; reward for losing is −1

The matches picked up on first simulation (with opponent's choice in brackets) is

1,[3],1,[3] (win)

We assign a gain, G, of 1 to:

  state 8, action 1

  state 4, action 1

and apply updating formula, setting α = 0.05

$$Q(S,A)^{new} = Q(S,A)^{old} + \alpha\left[G - Q(S,A)^{old}\right]$$

Nim after one trial (1,[3],1,[3])

Matches picked up	State (=number of matches left)						
	2	3	4	5	6	7	8
1	0	0	0.05	0	0	0	0.05
2	0	0	0	0	0	0	0
3		0	0	0	0	0	0

Next trials

Second trial
1,[2],1,[3],1 (lose)

Matches picked up	State (=number of matches left)						
	2	3	4	5	6	7	8
1	0	0	0.05	-0.05	0	0	-0.025
2	0	0	0	0	0	0	0
3		0	0	0	0	0	0

Example of convergence, ε=0.1 (Tables 7.8 to 7.10)

After 1000 trials

Matches picked up	State (= number of matches left)						
	2	3	4	5	6	7	8
1	0.999	-0.141	0.484	-0.122	0.155	0.000	0.272
2	-0.994	0.999	-0.108	-0.276	-0.171	0.000	0.252
3	0.000	-0.984	1.000	-0.070	-0.080	0.000	0.426

After 5,000 trials

Matches picked up	State (= number of matches left)						
	2	3	4	5	6	7	8
1	1.000	-0.127	0.382	0.069	0.898	0.000	0.786
2	-1.000	1.000	0.222	0.297	-0.059	0.000	0.683
3	0.000	-1.000	1.000	-0.106	0.041	0.000	0.936

After 25,000 trials

Matches picked up	State (= number of matches left)						
	2	3	4	5	6	7	8
1	1.000	0.080	0.104	0.069	0.936	0.000	0.741
2	-1.000	1.000	0.103	0.412	-0.059	0.000	0.835
3	0.000	-1.000	1.000	-0.106	0.041	0.000	1.000

Dynamic Programming

- Simple sequential decision problems can be solved by dynamic programming where we work back from the end to the beginning calculating the optimal action in each state. The recursion formula for the value $V(S)$ of being in state S is

$$V_t(S) = \max_{A} E\left[R_{t+1} + V_{t+1}(S^{new}) \right]$$

- An alternative to Monte Carlo simulation is temporal difference learning which uses the ideas underlying dynamic programming

Temporal difference learning

Instead of using G (the total, possibly discounted, future rewards) to update, we can use the current value at the next step. The updating formula becomes

$$Q(S,A)^{new} = Q(S,A)^{old} + \alpha\left[R + V - Q(S,A)^{old}\right]$$

where V is the current estimate of the value of being in the state reached at the end the next step. If S^* is this state, V is the current maximum value of $Q(S^*,A)$ across all actions A that can be taken in the state.

This is referred to a Q-learning

Example (page 158-159)

- Suppose that the current Q values are

Matches picked up	State (= number of matches left) 2	3	4	5	6	7	8
1	1.000	-0.127	0.382	0.069	0.898	0.000	0.786
2	-1.000	1.000	0.222	0.297	-0.059	0.000	0.683
3	0.000	-1.000	1.000	-0.106	0.041	0.000	0.936

- The next trial is 1,[1],1,[3],1,[1] (explore, exploit, exploit)

$$Q^{new}(8,1) = Q^{old}(8,1) + 0.05[V(6) - Q^{old}(8,1)]$$
$$= 0.786 + 0.05 \times (0.898 - 0.786) = 0.792$$
$$Q^{new}(6,1) = Q^{old}(6,1) + 0.05[V(2) - Q^{old}(6,1)]$$
$$= 0.898 + 0.05 \times (1.000 - 0.898) = 0.903$$
$$Q^{new}(2,1) = 1.000 + 0.05 \times (1.000 - 1.000) = 1.000$$

n-step bootstrapping

- Monte Carlo bases updates on what happens over the complete life of the trial
- Temporal difference bases updates on what happens over the next period
- n-step bootstrapping algorithms is between the two. It bases updates on what happens over the next n periods

When there are many states or actions (or both)

- The cells of the state/action table do not get filled in very quickly
- It becomes necessary to estimate the $Q(S,A)$ function from observed values.
- As this function is in general non-linear a natural approach is to use artificial neural networks (ANNs).
- We use an ANN to minimize the sum of squared errors between the estimates and the target
- This is known as deep Q-learning or deep reinforcement learning

Applications

- Games such as Go and chess
- Driverless cars
- Programming of traffic lights
- Healthcare
- Finance applications
  - Optimally selling a large block of shares
  - Portfolio management where there are transaction costs
  - Hedging