Dynamic Programming vs. Reinforcement Learning

Noah Ruhmer

11. October 2022

Content

1. Introduction
2. Theory
3. Example
4. Results
5. Bibliography

Motivation

- Learn theory of optimal control problems
- Compare between model-free and model-based approaches
- Apply dynamic programming and reinforcement learning in practice
- Compare these different methods

Markov Decision Process

Describes a discrete-time dynamic process [Bel57]. Useful for solving optimization problems via dynamic programming.

- States
- Actions
- Transitions (probabilistic)
- Cost/Reward

Markov Property:
"Given the present, the future does not depend upon the past."

Dynamic Programming

Model-based mathematical optimization of a MDP [Bel66]. Sub-problems solved recursively retrieving an optimal policy.

- Requires optimal substructure (e.g. Shortest Path)
- Requires overlapping sub problems (e.g. Fibonacci)

Stochastic Bellman Equation:

$$V(s) = \max_a \left(R(a, s) + \sum_{s'} P(a, s, s') V(s') \right)$$

Reinforcement Learning

Trial and Error (Explorative) approach to estimate optimal policy.

1. Observe state
2. Choose action depending on state by using the policy
3. Execute action
4. Reward or punishment from environment
5. Record information about reward for action-state pair

- Reward depending on outcome not actions
- Policy maximizes received reward

Q-Learning

Model-free reinforcement learning algorithm [Wat89].
It does not need transition rules to learn.

- Uses table assigning each action-state pair a value
- Learning by exploration in the environment
- Policy is to pick action with highest Q-Value
- Converges to optimal policy [WD92]

Q-Table update:

$$Q_{new}(a, s) = Q(a, s) + \alpha \cdot \left(R(a, s) + \gamma \cdot \max_{a} Q(a, s') - Q(a, s) \right)$$

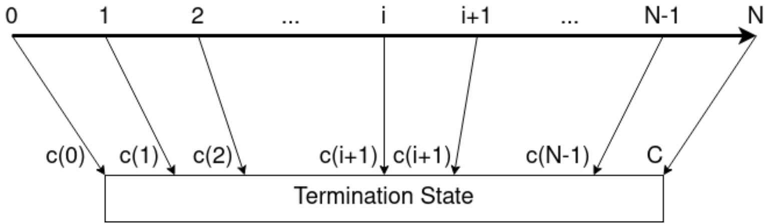
Parking Problem

N sequentially placed spaces where a driver wants to park with minimal cost. The driver is incrementally visiting the spaces observing only the current space [Ber19].

- Place is free with probability p
- The last space N (garage) has a fixed cost C and is free
- $c(i)$ decreasing from N to 0

There cost come only from transitioning to the parked state. Intuitive solution is to park as late as possible.

Parking Problem



- $2N$ states
- 2 actions

Previous states do not influence the current. Best policy to park after a threshold.

Dynamic Programming Solution

Recursive Value Function:

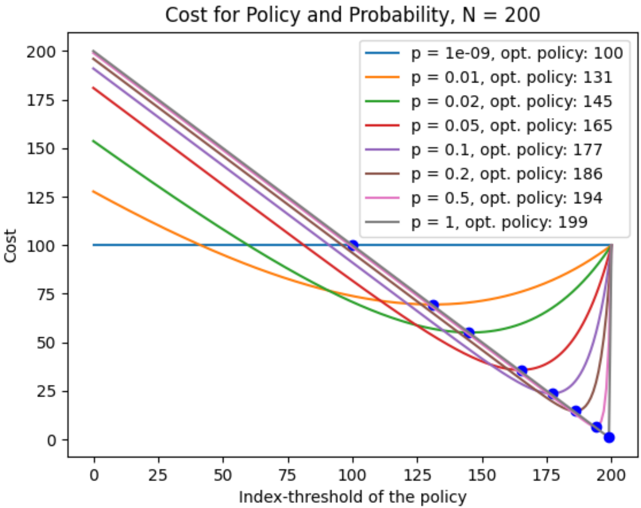
$$V(i) = p \cdot c(i) + (1-p) \cdot V(i+1)$$
$$V(N) = C, \quad \forall i: 0 \leq i < N$$

Explicit Value Function:

$$V(i) = C \cdot (1-p)^{N-i} + \sum_{j=0}^{N-i} p \cdot (N-i-j) \cdot (1-p)^j$$

Optimal Expected cost: $\max_i V(i) = V^*$

Parking at the first free space after this is optimal.


Cost for Policy and Probability, N = 200
p = 1e-09, opt. policy: 100
p = 0.01, opt. policy: 131
p = 0.02, opt. policy: 145
p = 0.05, opt. policy: 165
p = 0.1, opt. policy: 177
p = 0.2, opt. policy: 186
p = 0.5, opt. policy: 194
p = 1, opt. policy: 199
Cost
0
25
50
75
100
125
150
175
200
Index-threshold of the policy

Q-Learning Rewards

Approximate the Dynamic Programming solution.

We want minimal cost, however Q-Learning uses maximal rewards.

Total rewards:

- Negate cost function
- Shift it to use only positive values

Incremental rewards:

- reward driving instead of parking
- sum of reward needs to be equal to other reward functions $\rightarrow$ negative reward in garage

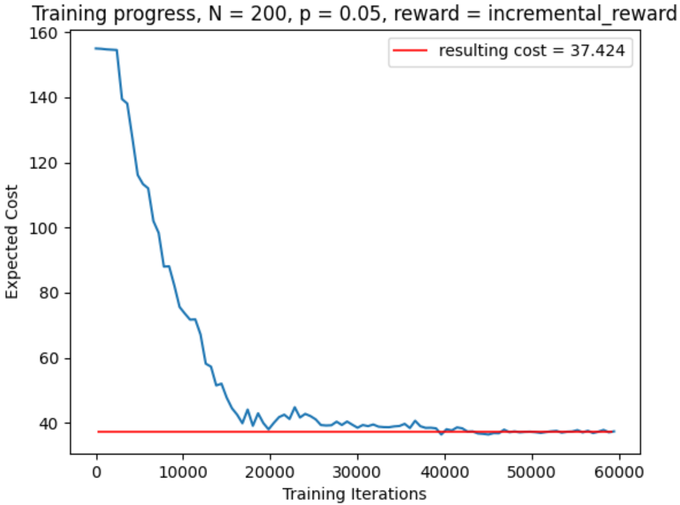
Q-Learning Training

Q-Table initialized to 0.

Parameters:

- learning rate: $\alpha = 0.05$ and descending, higher for *N* larger than 500
- exploration rate: $\epsilon = 0.05$ and descending
- discount factor: $\gamma = 0.999$

Results can be further improved by tuning these parameters and by increasing the training time.


Training progress, N = 200, p = 0.05, reward = incremental_reward
resulting cost = 37.424
Expected Cost
Training Iterations
160
140
120
100
80
60
40
0
10000
20000
30000
40000
50000
60000

Result Comparison

N	Q-learning		Dynamic Programming
-	μ	σ	-
50	17.005	0.263	16.366
100	26.251	0.519	25.140
200	37.145	0.989	35.764
500	53.973	1.591	51.663

Statistics from 20 independently learned policies

Conclusion

Dynamic Programming

- Full system model needed
- High requisites on the system
- Delivers an exact solution

Reinforcement Learning

- No system model needed
- Approximated solution
- Adaptive to similar problems

[Bel57] Richard Bellman. “A Markovian decision process”. In: Journal of mathematics and mechanics (1957), pp. 679–684.

[Bel66] Richard Bellman. “Dynamic programming”. In: Science 153.3731 (1966), pp. 34–37.

[Ber19] Dimitri Bertsekas. Reinforcement learning and optimal control. Athena Scientific, 2019.

[Wat89] Christopher John Cornish Hellaby Watkins. “Learning from delayed rewards”. In: (1989).

[WD92] Christopher JCH Watkins and Peter Dayan. “Q-learning”. In: Machine learning 8.3 (1992), pp. 279–292.