Dynamic Programming

February 15, 2023

1 4.1 Policy Evaluation

Consider a sequence of approximate value functions $v_0, v_1, v_2....$ The initial approximation v_0 is chosen arbitrarily and each successive approximation is obtained by using the *Bellman Equation* as an update rule.

$$v_{k+1}(s) = \sum_{r,a,s'} \pi(a|s)p(s', r|s, a)[r + \gamma v_k(s')] \tag{1}$$

v_k can be shown in general to converge to v_π as $k \to \infty$. This method is called **iterative policy evaluation**.

A small threshold $\epsilon > 0$ is used to determine the accuracy of estimation. Loop: $\Delta \leftarrow 0$ Loop for each $s \in S$: $v \leftarrow V(s)$ $V(s) \leftarrow \sum_{r,a,s'} \pi(a|s)p(s', r|s, a)[r + \gamma V(s')]$ $\Delta \leftarrow max(\Delta, |V(s) - v|)$ until $\Delta < \epsilon$

2 4.2 Policy Improvement

Our reason for computing the value function for a policy is to help find better policies. For some state s we would like to know whether or not we should change the policy to determinstically choose an action $a \neq \pi(s)$. Would it be better or worse? We can use the *action-value function.*

$$q_\pi(s, a) = \sum_{r,s'} p(s', r|s, a)[r + \gamma v_\pi(s')] \tag{2}$$

if $q_\pi(s, a) > v_\pi(s)$, then taking action a at state s is better. The process of making a new policy that improves on an original policy, by making it greedy with respect to the value function of the original policy, is called **policy improvement**.

$$v_{\pi'}(s) = \max_a q_\pi(s, a) = \max_a \sum_{r,s'} p(s', r|s, a)[r + \gamma v_\pi(s')] \tag{3}$$

$$\pi'(s) = arg\max_a q_\pi(s, a) = arg\max_a \sum_{r,s'} p(s', r|s, a)[r + \gamma v_\pi(s')] \tag{4}$$

Here we are only considering the special case of determinstic policies. But the idea can be extended to stochastic policies.

3 4.3 Policy Iteration (using iterative policy evaluation)

1. Initialization Initialize $V(s)$ and $\pi(s)$ arbitrarily for all s.
2. Policy Evaluation Loop: $\Delta \leftarrow 0$ Loop for each $s \in S$: $v \leftarrow V(s)$ $V(s) \leftarrow \sum_{r,a,s'} \pi(a|s)p(s',r|s,a)[r+\gamma V(s')]$ $\Delta \leftarrow max(\Delta, |V(s)-v|)$ until $\Delta < \epsilon$
3. Policy Improvement $policystable \leftarrow true$ for each $s \in S$: $oldaction \leftarrow \pi(s)$ $\pi(s) \leftarrow arg\max_a \sum_{r,s'} p(s',r|s,a)[r+\gamma v_\pi(s')]$ if $oldaction \neq \pi(s)$, then $policystable \leftarrow false$ if $policystable$, then stop and return V and π; else go to 2.

3.1 Ex 4.2 Jack's Car Rental

[ ]:
```
import pickle
import numpy as np

with open('jcr.pickle', 'rb') as f:
    table = pickle.load(f)
for s_ in range(len(table)):
    for r in range(len(table[s_])):
        print(np.sum(table[s_, r]))
```

4 4.4 Value Iteration

Initialize $V(s)$ for all $s \in S$ arbitrarily Repeat $\Delta \leftarrow 0$ For each $s \in S$: $v \leftarrow V(s)$ $V(s) \leftarrow max_a \sum_{r,s'} p(s',r|s,a)[r+\gamma V(s')]$ $\Delta \leftarrow max(\Delta, |v-V(s)|)$ until $\Delta < \epsilon$(a small positive integer) Output a deterministic policy π, such that $\pi(s) = argmax_a \sum_{r,s'} p(s',r|s,a)[r+\gamma V(s')]$

Value iteration effectively combines, in each of its sweeps, one sweep of policy evalutaion and one sweep of policy improvement.

4.1 Ex4.3 Gambler's Problem

[ ]:
```
import matplotlib.pyplot as plt

states = [i for i in range(101)]
rewards = [0, 1]
gamma = 1
values = {s: 0 for s in states}
table = dict()
p_win = 0.4

for s_ in states:
    for s in states:
        if s == 0:
            continue
        for a in [i for i in range(min(s, 100 - s) + 1)]:
            if a == 0 and s == s_:
                table[(s_, 0, s, a)] = 1
            elif s + a == s_:
```

```
                if s_ == 100:
                    table[(s_, 1, s, a)] = p_win
                else:
                    table[(s_, 0, s, a)] = p_win
            elif s - a == s_:
                table[(s_, 0, s, a)] = 1 - p_win

delta = float('inf')
epsilon = 0.01
while delta >= epsilon:
    delta = 0
    for s in values:
        actions = [i for i in range(min(s, 100 - s) + 1)]
        old_value = values[s]
        action_values = {a: 0 for a in actions}
        max_action_value = 0
        for a in actions:
            for s_ in states:
                for r in rewards:
                    try:
                        action_values[a] += table[(s_, r, s, a)] * (r + gamma *␣
 ↪values[s_])
                    except:
                        pass
            max_action_value = max(max_action_value, action_values[a])
        values[s] = max_action_value
        delta = max(delta, abs(values[s] - old_value))

fig, ax = plt.subplots(figsize=(16, 9))
ax.plot([s for s in values], [values[s] for s in values])
```

5 4.5 Asynchronous Dynamic Programming

Asynchronous DP algorithms are in-place iterative DP algorithms that are not organized in terms of systematic sweeps of the state set. These algorithms update the values of states in any order whatsoever, usinng whatever values of other states happen to be available. The values of some states may be updated several times before the values of others are updated once. To converge correctly, an asynchronous algorithm must continue to update the values of all the states: it cacn't ignore any state after some point in the computation. Asynchronous DP algorithms allow great flexibility in selecting states to which update operations are applied. Asynchronous algorithms also make it easier to intermix computation with real-time interaction. To solve a given MDP, we can run an iterative DP algorithm *at the same time that an agent is actually experiencing the MDP*. The agent's experience can be used to determine the states to which the DP algorithm applies its updates. At the same time, the latest value and policy information from the DP algorithm can guid the agent's decision-making. For example, we can apply updates to states as the agent visits then. This makes it possible to focus the DP algorithm's updates onto parts of the state set that are most relevant to the agent.

6 4.6 Generalized Policy Iteration

We use the term *generalized policy iteration* (GPI) to refer to the general idea of letting policy evaluation and policy improvement processes interact, independent of the granularity and other details of the two processes.

[ ]: