Tabular and Approximate Q-Learning Algorithms

Dimitris Angelopoulos, Konstantinos Michalochristas

July 2024

Reinforcement Learning

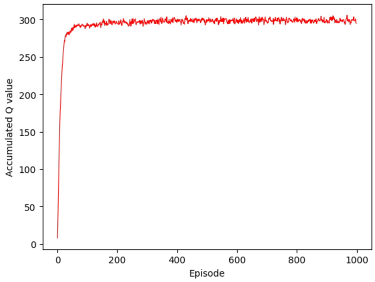
- Problem: No full environment knowledge
- Solution: Q-Learning Algorithm

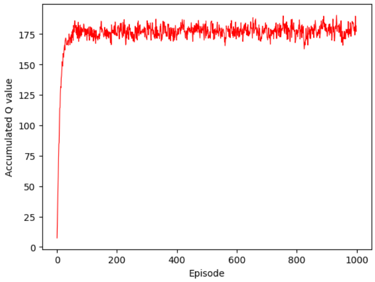
```
Initialize $Q(s, a), \forall s \in \mathcal{S}, a \in \mathcal{A}$;
**repeat**
    Initialize $s \in \mathcal{S}$;
    **repeat**
        $a \leftarrow \epsilon\text{-}greedy(s)$;
        $r, s' \leftarrow step(a, s)$;
        $Q(s, a) \leftarrow Q(s, a) + \alpha(r + \gamma \max_{a'} Q(s', a') - Q(s, a))$;
        $s \leftarrow s'$;
    **until** $t = kT_{eff}$;
**until** *last episode*;
```

Task 1 — Q1 & Q2

- For environment parameters $N = 2$, $\gamma = 0$, $r_0^H = 2r_1^H = 0.06$, $r_i^L = -r_i^H$, $c = 0.02$ and $P_{xx'}^i = 0.5$, for $i = 0, 1$.
- The optimal policy (PI algorithm) is to never switch stock.
- (tabular) Q-Learning yields the same policy !
- Hyperparameters: $\alpha = 0.2$, $T_{eff} = \lceil 1/(1-\gamma) \rceil$, $k = 10N$, $episodes = 1000$
- Q-Learning approaches optimality even for larger state spaces.
- Lets consider 2 cases with 3 stocks. The first case has one very good stock while the second has three similarly good.

Task 1 — Q1 & Q2



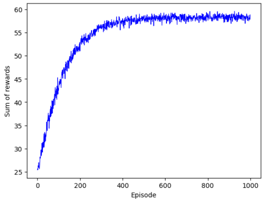


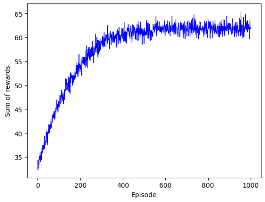
- Left Plot: One stock is the best.
- Right Plot: All 3 stocks are good enough.
- The agent switches at the most profitable stock given the markov states of the other.
- For the right case, $\hat{\pi}^*(s) \neq \pi^*(s)$. But they differ only in 2 states and yield almost the same reward !

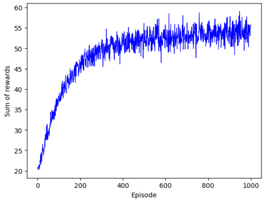
Task 2

- We run the algorithm for different N and observe that the Q-Learning policies are always almost identical even for the more complex cases (stocks similarly profitable)
- As the state space gets very large we cannot look at every element of the policies.
- Until $N = 5$ for complex cases, Q-Learning is very close to the optimal.
- We will check the convergence through the rewards the agent accumulates for each episode.
- If the cumulative reward becomes stationary, that means that the agent uses the close to optimal policy.

Task 2

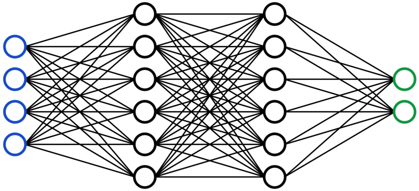






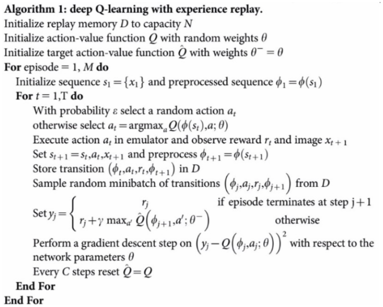
- Left Plot: Cumulative reward for $N = 6$ stocks (≈6 mins).
- Middle Plot: Cumulative reward for $N = 7$ stocks (≈13 mins).
- Right Plot: Cumulative reward for $N = 8$ stocks (≈31 mins).
- All three become stationary stochastic processes after a number of episodes passes.
- Note: It seems that we need to increase the episodes as the stocks increase.

Neural Network Architecture



- Input Layer: state $x = [a, 1, 0, \ldots, 1]$ if we have state knowledge, or $x = [a, d_1, d_2, \ldots, d_N]$ if we have continuous values, where $d_i \in \mathbb{R}$, $i = 1, \ldots, N$. a is the stock we hold.
- Output Layer: Q-values $q = [Q(x, 0), \ldots, Q(x, N-1)]$
- Hidden Layers: 2 hidden layers of size 64
- Hyperparameters: learning rate $\alpha = 0.001$, batch $|\mathcal{B}| = 64$
- Optimizer: Stochastic Gradient Descent

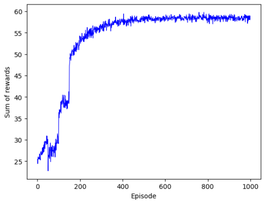
Task 3

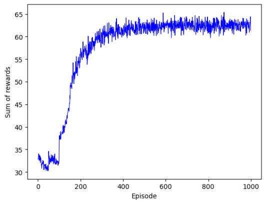


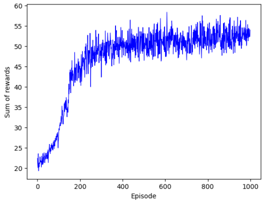
```
Algorithm 1: deep Q-learning with experience replay.
Initialize replay memory $D$ to capacity $N$
Initialize action-value function $Q$ with random weights $\theta$
Initialize target action-value function $\hat{Q}$ with weights $\theta^- = \theta$
For episode = 1, $M$ do
  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$
  For $t$ = 1,T do
    With probability $\varepsilon$ select a random action $a_t$
    otherwise select $a_t = \text{argmax}_a Q(\phi(s_t), a; \theta)$
    Execute action $a_t$ in emulator and observe reward $r_t$ and image $x_{t+1}$
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $D$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $D$
    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step } j+1 \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^-) & \text{otherwise} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ with respect to the
    network parameters $\theta$
    Every $C$ steps reset $\hat{Q} = Q$
  End For
End For
```

- We implemented the Deep Q-Learning algorithm shown above and ran it for $N = 6, 7, 8$.
- We implement the NN we developed using PyTorch.
- We get figures respective to the ones we got for tabular Q-Learning.

Task 3







- Left Plot: Cumulative reward for $N = 6$ stocks ($\approx$40 mins).
- Middle Plot: Cumulative reward for $N = 7$ stocks ($\approx$43 mins).
- Right Plot: Cumulative reward for $N = 8$ stocks ($\approx$60 mins).
- Since the cumulative reward becomes stationary and the cumulative rewards are similar to the ones of the tabular version(which is optimal), DQL is also close to optimal.
- Note: It seems that the network is very slow. But it also seems that the rate of the convergence time is less than the previous case. There might be some mistakes in this task... but the logic behind seems correct.

Conclusions

- Tabular Q-Learning is very close to to the optimal.
- Tabular methods are slow for very large state spaces.
- Approximate Q-Learning can also achieve optimality.
- Most specifically Deep Q-Learning is slower than its tabular counterpart in the cases with small state spaces.
- In cases where the state space is very large or continuous DQL is necessary.