COMP 579 - Assignment 2

Andreas Enzenhoefer - 260701043

Mostafa Dadkhah - 261097757

Submission Deadline - March 11, 2022

1 Problem formulation

1.1 State and action space, reward function

The state space is given by the remaining ingredients in stock after each sandwich purchase. The shop can hold K units of bread (N types available) and L units of fillings (M types available). Ingredients which are more than 3 days old must be discarded.

The action space is given by the ingredients to be purchased by the shop owner. They can purchase at most up to the point that the maximum stock capacity is reached (K units of bread, L units of fillings).

The reward function is given by the sales revenue made through sandwich sales minus a penalty if a customer cannot get the desired sandwich. This penalty has to be sufficiently severe since the customer will never come back again and the town's population is limited.

1.2 Size of the state space

The size of the state space is given by all possible combinations of remaining ingredients, after every purchase until expiration duration. There are N types of bread, and up to K units stored and up to t days until the expiration date. Additionally, a storage unit could be empty which increases the options to $N+1$, i.e. $(N+1)^t \cdot (N+1)^t \cdot \ldots \cdot (N+1)^t = (N+1)^{tK}$. The same applies to the fillings. Thus, the total state space considering $t = 3$ is given by $S = (N+1)^{3K}(M+1)^{3L}$.

1.3 Dynamic programming or reinforcement learning

Dynamic programming computes backup diagrams for all possible actions and states in an episode (full-width). Finding the optimal policy is polynomial in the number of states and, thus, only suitable for a relatively small state space (in practice up to 1 million states). We would choose reinforcement learning if this state-space size is exceeded.

1.4 Monte Carlo vs temporal-difference learning

The difference between Monte Carlo (MC) and temporal-difference (TD) learning is that TD bootstraps whereas MC does not. Bootstrapping means that the current estimate is updated based on previous estimates. In truly Markov scenarios, TD will learn faster than MC as there is no dependency between the previous states/actions and the next ones. For the sandwich shop, it is unlikely that the same customer comes multiple times a day whereas other do not come at all. Thus, we cannot assume that it is truly a Markov process and would favor MC. However, an intermediate n-step TD method could even be a better choice.

1.5 Function approximation

Depending on the values of N, M, K, and L, the state space can become quite large. If the state space is too large, we need to use function approximation to solve the problem efficiently (curse of dimensionality).

2 Bellman equations and dynamic programming

This question requires investigating the addition property in q_π, q_* and π_* values. For each value we first assume they have the property and then we will survey whether the assumption holds or not.

2.1 Action-Value Functions

The Bellman equation for state-value function is as follows:

$$q_\pi(s,a) = r(s,a) + \gamma \sum_{s' \in S} P^a_{ss'} \sum_{a' \in A} \pi(a'|s') q_\pi(s',a')$$

The linear combination of the two bellman equations of each reward function:

$$\underbrace{aq^1_\pi(s,a) + bq^2_\pi(s,a)}_{q_\pi(s,a)} = \underbrace{ar^1(s,a) + br^2(s,a)}_{\text{r(s,a)}} + \gamma \sum_{s' \in S} P^a_{ss'} \sum_{a' \in A} \pi(a'|s') \underbrace{(aq^1_\pi(s',a') + bq^2_\pi(s',a')))}_{q_\pi(s',a')}$$

Therefore, a linear combination of state-values with different reward functions could obtain the state-value that of the same linear combination of rewards.

2.2 Optimal Action-Value Functions

The Bellman equation for optimal state-value function is:

$$q_*(s,a) = r(s,a) + \gamma \sum_{s' \in S} P^a_{ss'} \max_{a'} q_*(s',a')$$

$$\underbrace{aq^1_*(s,a) + bq^2_*(s,a)}_{q_*(s,a)} = \underbrace{ar_1(s,a) + br_2(s,a)}_{\text{r(s,a)}} + \gamma \sum_{s' \in S} P^a_{ss'} \underbrace{(a \max_{a'} q^1_*(s',a') + b \max_{a'} q^2_*(s',a')}_{\neq \max_{a'} q_*(s',a') = \max_{a'}[aq^1_*(s',a') + bq^2_*(s',a')]})$$

Though, the Non-Linearity of maximization prevents the Bellman equation for the optimal value function from having the additionality.

2.3 Optimal policies

Optimal Policy requires meeting the optimal action-value condition, while as we have shown in section 2.2, the solution for combined rewards in this situation couldn't be obtained from each solution. Therefore, optimal policies couldn't obtain from suboptimal policies. In other words:

$$\pi_*(a|s) = \begin{cases} 1 & \text{if } a = \text{argmax}_a q_*(s,a) \neq \text{argmax}_a [aq^1_*(s,a) + bq^2_*(s,a)] \\ 0 & \text{otherwise} \end{cases}$$

3 An alternative learning algorithm

3.1 On-policy or off-policy

The algorithm is a combination of an off-policy Q-learning $(1-\varepsilon)\max_a Q(s',a)$ and an expected Sarsa $\varepsilon \sum_a \pi(s',a)Q(s',a)$. The policy in Expected Sarsa is softmax and not greedy which makes the Expected Sarsa to be on-policy. Thus, the algorithm can be classified as on-policy altogether since the Q values are always influenced by the agents current action a.

3.2 Two-step version of the learning algorithm

Algorithm 1 Two-step version of the algorithm

Require: Initialize $Q(s,a), \forall s \in S, a$ in $A(s)$, arbitrarily, and $Q(terminal-state,) = 0$
  repeat(for each episode):
    Initialize S
    repeat(for each step of the episode):
      Choose A from S using policy derived from Q
      Take action A, observe R, S'
      $G \leftarrow R_{t-1} + \gamma R_t + \gamma^2((1-\varepsilon).max_a Q(S',a) + \varepsilon \sum_a \pi(a|S')Q(S',a))$
      $Q(S_{t-1}, A_{t-1}) \leftarrow Q(S_{t-1}, A_{t-1}) + \alpha[G - Q(S_{t-1}, A_{t-1})]$
      $S \leftarrow S'$
    until S is terminal

3.3 Single-step function approximation version

Algorithm 2 Episodic mix Q-Learning+Sarsa

  Initialization: $Q_{QL}(s,a)$ and $Q_{Sarsa}(s,a), \forall s \in S, a \in A(s)$
  Initialization: $Q_{QL}(\text{terminal-state}, a) = Q_{Sarsa}(\text{terminal-state}, a) = 0$
  repeat(for each episode):
    Initialize S
    repeat(for each step of the episode):
      Choose A from S using $softmax(Q_{QL}(s,a) + Q_{Sarsa}(s,a))$
      Take action A, observe R, S'
      With ϵ Probability:
        $Q_{Sarsa}(s,a) \leftarrow Q_{Sarsa}(s,a) + \alpha[R + \gamma \sum_a \pi(a|S')Q_{QL}(S', \text{argmax}_a\, Q_{Sarsa}(s,a))]$
      Else:
        $Q_{QL}(s,a) \leftarrow Q_{QL}(s,a) + \alpha[R + \gamma max_a Q_{Sarsa}(S', \text{argmax}_a\, Q_{QL}(s,a))]$
      $S \leftarrow S'$
    until S is terminal

3.4 Algorithm convergence in the limit

Both algorithms, off-policy Q-learning and on-policy Expected Sarsa converge in the limit if all state-action pairs are visited and updated for a soft, greedifying policy. Since the proposed algorithm is a linear combination of the other two, we can conclude that it will converge in the same cases.

3.5 Expected performance

The algorithm is a linear combination of Expected Sarsa and Q-learning. Sarsa and Q-Learning are biased on initial values, leading to a bias maximization. Also, Q-Learning is expected to have a more variance and less convergence speed since the off-policy methods use the other policy than the agent's behavior. The Double algorithm is able to use expected sarsa to explore and Q-learning to exploit; Therefore has an in-between bias and variance. However, using two different functions prevents the problem of bias maximization as it has access to two different datasets.

4 Function approximation

Function approximation combined with TD bootstrapping and on-policy learning is in principle stable (as opposed to combined with off-policy learning). A higher degree of boostrapping ($\lambda \approx 0.9$) can reduce the steps per episode needed while keeping the cost comparable to pure bootstrapping ($\lambda = 0$). These algorithms use linear function approximators (state agreggation) which are guaranteed to converge for a on-policy learning. The algorithm producing a larger feature vector will generally take more computational effort, however, if the number of samples n is too small, it may not cover the entire state space.

5 k-order Markovian assumption

5.1 Algorithm

In this assignment, the space value function considered as be the expected value of rewards after a given trajectory of $\tau_{i:j}$ that is $s_i, a_i, s_{i+1}, a_{i+1}, ..., s_j$ [ref.]

$$V_\pi(\tau_{t:t+k}) = \mathbb{E}\left[\sum_{m=0}^{\infty} \gamma^m R_{t+k+m+1} | \tau_{t:t+k}\right]$$

$$V_\pi(\tau_{t:t+k}) = \lim_{T\to\infty} \sum_{(r_{t+k+1:T})} p(r_{t+k+1:T}|\tau_{t:t+k}) \sum_{m=0}^{T} \gamma^m r_{t+k+m+1}$$

Using chain rule and re-ordering probabilities obtain:

$$\begin{aligned} V_\pi(\tau_{t:t+k}) = \sum_{(a_{t+k})} \pi(a_{t+k}|\tau_{t:t+k}) \sum_{(s_{t+k+1})} p(s_{t+k+1}|\tau_{t:t+k}, a_{t+k}). \\ \lim_{T\to\infty} \sum_{(r_{t+k+1:T})} p(r_{t+k+1:T}|\tau_{t:t+k}, a_{t+k}, s_{t+k+1}) \sum_{m=0}^{T} \gamma^m r_{t+k+m+1} \end{aligned} \tag{1}$$

With the definition of trajectory, $\tau_{t:t+k}, a_{t+k}, s_{t+k+1} = \tau_{t:t+k+1}$, the last limit with isolating on the first step would be:

$$E = \lim_{T\to\infty} \sum_{r_{(t+k+1:T)}} p(r_{t+k+1:T}|\tau_{t:t+k+1})[r_{t+k+1} + \sum_{m'=1}^{T} \gamma^{m'} r_{t+k+m'+1}]$$

By linearity of expectations the above equation would be $E = E_1 + E_2$ where the first part is the expected reward of $(R_{\tau_{t:t+k+1}})$ in the trajectory. The second part, with factoring one γ can be written as follows:

$$E_1 = \lim_{T\to\infty} \sum_{(r_{t+k+1:T})} p(r_{t+k+1:T}|\tau_{t:t+k+1}).r_{t+k+1} = R_{\tau_{t:t+k+1}}$$

$$E_2 = \gamma. \lim_{T\to\infty} \sum_{(r_{t+k+1:T})} p(r_{t+k+1:T}|s_t, a_t, \tau_{t+1:t+k+1})[\sum_{m'=0}^{T} \gamma^{m'} r_{t+k+m'+2}]$$

Since the rest of trajectory is independent of s_t, a_t, we can ignore them in the probability.

$$E_2 = \gamma. \lim_{T\to\infty} \mathbb{E}\left[\sum_{m'=0}^{T-k\approx T} \gamma^{m'} r_{t+m+2} | \tau_{t+1:t+k+1}\right] = \gamma V_\pi(\tau_{t:t+k})$$

By using E_1 and E_2 in (1) the state-value of policy π would be:

$$V_\pi(\tau_{t:t+k}) = \sum_{(a_{t+k})} \pi(a_{t+k}|\tau_{t:t+k}) \sum_{(s_{t+k+1})} p(s_{t+k+1}|\tau_{t:t+k}, a_{t+k})[R_{\tau_{t:t+k+1}} + \gamma V_\pi(\tau_{t+1:t+k+1})]$$

With maximazing over all current actions we can obtain the optimal policy which is a γ-contraction and it will converge to a unique, fixed point. However, since the state space is in a high order of $|S|^k$, the algorithm might not visit adequately and not learn for a long time.

$$V^*(\tau_{t:t+k}) = \max_{(a_{t+k})} \sum_{(s_{t+k+1})} p(s_{t+k+1}|\tau_{t:t+k}, a_{t+k})[R_{\tau_{t:t+k+1}} + \gamma V^*(\tau_{t+1:t+k+1})]$$

$$\begin{aligned} ||T^\pi(u) - T^\pi(v)||_\infty &= ||\max P(R^\pi + \gamma.u) - \max P(R^\pi + \gamma.v)||_\infty \\ &\leq ||(||P(R^\pi + \gamma.u) - P(R^\pi + \gamma.v)||_\infty)||_\infty \\ &= ||P\gamma(u - v)||_\infty \\ &\leq ||\gamma(u - v)||_\infty \end{aligned}$$

To obtain the transition probabilities in this question, we can simply use the equation bellow:

$$p(s_{t+k+1}|\tau_{t:t+k}, a_{t+k}) = \sum_{r_{t+k+1}} p(s_{t+k+1}, r_{t+k+1}|\tau_{t:t+k}, a_{t+k})$$

If $k = 0$, then the trajectory $\tau_{t:t}$ and $R_{\tau_{t:t+1}}$ are respectvely equal to s_t and $R^{a'}_{s_t,s_{t+1}}$ that makes the formulation the same with the original form of optimal state value function.

$$V^*(\tau_{t:t}) = \max_{(a_t)} \sum_{(s_{t+1})} p(s_{t+1}|\tau_{t:t}, a_t)[R_{\tau_{t:t+1}} + \gamma V^*(\tau_{t+1:t+1})]$$

Algorithm 3 Value iteration algorithm for k-order MDP

Define: $T = [\tau]$ as all possible k-order trajectories τ of states $s \in S$)
Initialize: V arbitrarily (e.g., $V(\tau), \forall \tau \in T$)
repeat:
    $\Delta \leftarrow 0$
    for for each trajectory $\tau_{s:s'} \in T$ **do**
        $v \leftarrow V(\tau_{s:s'})$
        $V(\tau_{s:s'}) \leftarrow \max_{(a')} \sum_{(s'+1,r'+1)} p(s'+1, r'+1|\tau_{s:s'}, a')[R_{s+1:s'+1} + \gamma V(\tau_{s+1:s'+1})]$
        $\Delta \leftarrow \max(\Delta, |v - V(\tau_{s:s'})|)$
until:$\Delta < \theta$
output: a deterministic policy π, such that:
    $\pi(\tau_{s:s'}) = \text{argmax}_{a'} \sum_{(s'_{+1},r'_{+1})} p(s'_{+1}, r'_{+1}|\tau_{s:s'}, a')[R_{s_{+1}:s'_{+1}} + \gamma V^*(\tau_{s_{+1}:s'_{+1}})]$

5.2 Effect of unknown k:

In the case of the unknown order of MDP, the environment model has a different distribution of rewards and therefore learning might take place with a lower speed. In the limit, we can reach to the optimal value but in practice, it might have instability in the answer, a higher level of variance, and lots unvisited trajectories. if we have a sequence of $S_{t-k:t+1} = (s_{t+1}, a_t, s_t, a_{t-1}, ..., s_{t-k})$

5.3 verbose

In this section we construct another value function. The probability of a section is:

$$P(S_{t-k:t}) = P(s_t|S_{t-k-1:t-1}, a_{t-1}).P(a_{t-1}|S_{t-k-2:t-2})...P(s_{t-k+1}|s_{t-k}, a_{t-k})$$

$$= \prod_{m=t-k}^{t-1} P(s_{m+1}|S_{t-k:m}, a_m). \prod_{m=t-k+1}^{t-1} \pi(a_m|S_{t-k:m-1})$$

One approximation for this could be Bigram such that:

$$\prod_{m=t-k+1}^{t-1} P(a_m|S_{t-k:m-1}) = \prod_{m=t-k+1}^{t-1} \pi(a_m|s_{m-1})$$

We define the state value as bellow, conditioned on k further steps because of the assumption of the question.

$$V(s_t) = \mathbb{E}\left[\sum_{m=t+1}^{\infty} r_m\right] = \sum_{(t+1:t+k+1)} P(S_{t+1:t+k+1}).(R_{t+1:t+k+1}) + P(S_{t+k+2:\infty}).(R_{t+k+2:\infty}))$$

$$= \sum_{(t+1:t+k+1)} P(S_{t+1:t+k+1}).(R_{t+1:t+k+1}) + V(s_{t+1})$$

With the approximation we have:

$$V(s_t) = \mathbb{E}\left[\sum_{m=t+1}^{\infty} r_m\right] = \sum_{(t+1:t+k+1)} .(R_{t+1:t+k+1}) + P(S_{t+k+2:\infty}).(R_{t+k+2:\infty}))$$

$$= \sum_{((t+1):(t+k+1))} (\prod_{m=(t+1)}^{(t+k+1)} P(s_m|S_{t+1:m}, a_m).[\prod_{m=(t+1)}^{(t+k+1)} \pi(a_m|s_{(m-1)}).R_{(t+1):(t+k+1)} + V(s_{t+k+1})]$$

The optimal version of the last value could be maximized on the policy π. In other words, it only updates the value function when the greedy policy has chosen on the whole trajectory.