Q1: Bandits and Reinforcement Learning

(a) Please note: I again forgot β in the Bellman optimality equations as usual! But, the overall integrity of the question is preserved even without β.

The Bellman optimality equation of the two environments are:

$$\text{Environment 1}: \; V^\star(x) = \max_{a \in A(x)} \left(r(x,a) + \sum_{x' \in S} \theta_{x,x'} V^\star\left(x'\right) \right)$$

$$\text{Environment 2}: \; V^\star(x) = \max_{a \in A(x)} \left(r(x,a) + \sum_{x' \in S} \theta_{x'} V^\star\left(x'\right) \right)$$

The main difference between Reinforcement Learning and Bandits is that in Reinforcement learning, the current action effects future rewards. The current reward is captured by $r(x,a)$ for both the environments. The future reward is captured by the second term: $\sum_{x' \in S} \theta_{x,x'} V^\star\left(x'\right)$ for environment 1 and $\sum_{x' \in S} \theta_{x'} V^\star\left(x'\right)$ for environment 2. Notice that the second term of none of these environments is dependent on the current action, a. Hence, the both the environments are Bandit setup.

Now, the Bandit setup can be either Multi-Armed Bandit or Contextual Bandit. In Multi-Armed Bandits, there is only one context in the context space. So, dependent on whether the state space, S, has one or more states (states and context are the same), both the environments are either Multi-Armed Bandit or Contextual Bandit respectively.

(b) Before we start the psuedocode, we want to point out that the explicit approach of choosing actions has not been specified. All we know is that Median elimination is a value-based policy and hence the action selection should depend on the estimate of the everage reward.

Algorithm 1: Psuedocode for Median elimination

```
1  Initialize a set $S = \{1, 2, \dots, K\}$, where $K$ is the number of actions, that stores the set of potentially "best" actions.
2  Initialize an array $Q = [0, 0, \dots, 0]$ that stores estimated average reward of each of the $K$ actions.
3  Initialize an array $N = [0, 0, \dots, 0]$ that stores the number of times each of the $K$ actions have been taken.
4  Set $counter = 0$.
5  for every time slot until $len(S) = 1$ do
6      Pick action $a = choose\_action(Q, S)$. $choose\_action()$ is a function that chooses action $a \in S$ based on the
       current estimate of the average reward, $Q$.
7      Take action $a$ and get the reward, $r$.
8      Update $Q[a]$ and $N[a]$ as follows where $Q[a]$ and $N[a]$ are the estimated average reward of action $a$ and number
       of times action $a$ is taken respectively,
           $Q[a] \leftarrow Q[a] + \frac{1}{N[a]+1}(r - Q[a])$
           $N[a] \leftarrow N[a] + 1.$
9      Set $counter \leftarrow counter + 1$.
10     if $counter = \tau$ then
11         Set $sorted\_actions = sort(Q, S)$ where $sorted\_actions$ is an array containing the actions in $S$ but in
           ascending order of $Q$.
12         Set $\overline{S} = \left\{ sorted\_actions[i] : \forall\, 1 \le i \le \left\lfloor \frac{len(S)}{2} \right\rfloor \right\}$ where $\overline{S}$ is the set of actions whose $Q[a]$ is less than the
           median. $\lfloor \cdot \rfloor$ is the floor function.
13         Update $S$ by removing the actions in $\overline{S}$ from $S$, $S \leftarrow S - \overline{S}$.
14         Set $counter \leftarrow 0$.
15 Return $S$.
```

(c) Case 2 will have a larger value of τ. This can be explained using the following flow of reasons:

1. The standard deviation of sample average of a random variable is $\frac{\sigma}{\sqrt{n}}$ where σ is the true standard deviation of the random variable, and n is the number of samples. The lesser the standard deviation of sample average, the more accurate is the average.
2. If σ is more, we need to increase n to maintain the accuracy level of sample average.
3. The larger the τ, the larger the expected number of times we take an action.
4. Since the standard deviation of rewards of different actions are more for case 2, we need a larger τ in order to get accurate estimate of average reward. An accurate estimate of average reward is required othwerwise we may eliminate actions with high reward from the set S.

Q2: Markov Decision Process

(a) At time t, any reserved instances bought before time $t-\tau+1$will expire. Hence,

$$A_t = \sum_{i=1}^{\tau-1} r_{t-i}$$

where r_t is the number of reserved instances bought at time t and $r_t = 0$ if $t < 0$.

(b) Each instance, either on-demand or reserved, can handle M units of demand. Since the maximum demand is D, the maximum number of either on-demand or reserved instance that is needed is $\lceil \frac{D}{M} \rceil$ where $\lceil \cdot \rceil$ is the ceil function.

(c) The state at time t is $\begin{bmatrix} d_t & x_t^1 & x_t^2 & \cdots & x_t^{\tau-1} \end{bmatrix}$ where d_t is the demand, and

$$x_t^i = r_{t-i} \; ; \; \forall i \in \{1, 2, \ldots, \tau-1\} \,. \tag{1}$$

Qualitatively, the state at a given time slot is the current demand and the history of the reserved instances bought in the last $\tau - 1$ time slots.

Now, we find the state space. First note that from part (b) we know that, $x_t^i \in \left\{0, 1, \ldots, \lceil \frac{D}{M} \rceil \right\}$; $\forall i$. Therefore, the state space, S, is the cartesian product,

$$S = \{0, 1, \ldots, D\} \times \underbrace{\left\{0, 1, \ldots, \left\lceil \frac{D}{M} \right\rceil \right\}}_{\text{for } x_t^1} \times \underbrace{\left\{0, 1, \ldots, \left\lceil \frac{D}{M} \right\rceil \right\}}_{\text{for } x_t^2} \times \cdots \times \underbrace{\left\{0, 1, \ldots, \left\lceil \frac{D}{M} \right\rceil \right\}}_{\text{for } x_t^{\tau-1}}$$

(d) In the first glance it seems that the action at time t, are the number of reserved instances, r_t, and the number of on-demand instances, y_t, bought at time t. But, if we think further, given r_t and the current state $\begin{bmatrix} d_t & x_t^1 & x_t^2 & \cdots & x_t^{\tau-1} \end{bmatrix}$, the number of on-demand instances,

$$y_t = \max\left(0, d_t - r_t - \sum_{i=1}^{\tau-1} x_t^i\right)$$

In other words, y_t is dependent on r_t and x_t. Hence, the action at time t is simply r_t.

From part (b), $r_t \leq \lceil \frac{D}{M} \rceil$. Hence, the action space is,

$$U = \left\{0, 1, \ldots, \left\lceil \frac{D}{M} \right\rceil \right\}$$

(e) The cost of the state-action pair $\left(x^1, x^2, \ldots, x^{\tau-1}, d, r\right)$ is,

$$\begin{aligned} c\left(x^1, x^2, \ldots, x^{\tau-1}, d, r\right) &= Pr + py \\ &= Pr + p\max\left(0, d - r - \sum_{i=1}^{\tau-1} x^i\right) \end{aligned}$$

where the first and the second terms are the cost of buying reserved and on-demand instances respectively.

(f) The Bellman optimality equation is,

$$V\left(x^1, x^2, \ldots, x^{\tau-1}, d\right) = \min_{r \in U} Q\left(x^1, x^2, \ldots, x^{\tau-1}, d, r\right) \qquad \text{where,}$$

$$Q\left(x^1, x^2, \ldots, x^{\tau-1}, d, r\right) = Pr + p\max\left(0, d - r - \sum_{i=1}^{\tau-1} x_t^i\right) + \beta \sum_{\overline{d} \in \{0,1,\ldots,D\}} V\left(\overline{x}^1, \overline{x}^2, \ldots, \overline{x}^{\tau-1}, \overline{d}\right) \quad \text{and,} \tag{2}$$

and $\overline{x}^i$ and $\overline{d}$ are the next state and,

$$\overline{x}^i = \begin{cases} r & ; i = 1 \\ x^{i-1} & ; i > 1 \end{cases} \tag{3}$$

Equation (3) can be visualized as right-shifting the current state $x^1, x^2, \ldots, x^{\tau-1}$ to get the next state $r, x^1, \ldots, x^{\tau-2}$ and filling the first entry with r. To be rigourous, (3) can be explained as follows. Using (1), $x_{t+1}^i = r_{t+1-i} = r_{t-(i-1)}$. Then,

$$x_{t+1}^i = \begin{cases} r_t & ; i = 1 \\ x_t^{i-1} & ; i > 1 \end{cases} \tag{4}$$

Equation (3) can be obtained from (4) by removing the time variable.

Q3: Reinforcement Learning

You can approach Dr. Gone for the solution.

Q4: Deep Reinforcement Learning

(a) I am filling lines 6, 8, 9, 10, 11, and 14 directly in the algorithm (check next page). The lines are highlighted in blue.

Side notes:

1. In lines 6 and 9, we can't use any policy like Deep Q-Learning. We must use ε-greedy policy for the Q-function given by the current predict DQN. This is because SARSA is an on-policy algorithm.

(b) Deep SARSA is an on-policy algorithm. Hence, it can't use replay buffer because for on-policy algorithms we can't train the agent (neural network here) using data collected from old policies (or not the current policy). Since we can't use replay buffer for Deep SARSA, the following issues arrises:

1. *Sample inefficiency:* Deep Q-Learning can use a sample multiple times to train itself while Deep SARSA can't.
2. *Correlated samples:* Deep Q-Learning addresses the issue of correlated samples by randomly sampling from replay buffer. Since SARSA can't use replay buffer, training the neural network is done on correlated data.
3. *Offline data:* Since Deep SARSA is on-policy we can't use offline data collected using another policy to train the neural network.

Algorithm 2: Psuedocode for Deep SARSA with DQN of Architecture 1

```
1  Initialize a predict DQN $\hat{Q}(\cdot\,;\phi_P)$ and target DQN $\hat{Q}(\cdot\,;\phi_T)$. Both the DQN should have the same architecture
   just different parameters.
2  Set an integer $N_u$ (predict DQN update frequency), $N_T$ (target DQN update frequency), $N_b$ (training batch size),
   and $counter = 0$.
3  for every episode until convergence do
4      Reset the environment to get the current state $x$.
5      Choose learning rate, $\alpha$, and exploration probability, $\varepsilon$, for this episode.
6      Pick action, $a$, for current state, $x$, using $\varepsilon$-greedy policy for the Q-function given by the current predict DQN
       $\hat{Q}(\cdot\,;\phi_P)$.
7      for every time slot of the episode until convergence do
8          Take action, $a$, and get reward, $r$, and next state, $x'$.
9          Pick action, $a'$, for next state, $x'$, using $\varepsilon$-greedy policy for the Q-function given by the current predict
           DQN $\hat{Q}(\cdot\,;\phi_P)$.
10         Generate a single sample of training data: set input $X = (x, a)$ and target $y = r + \beta\hat{Q}\left(x', a'\,;\phi_T\right)$ where
           $\phi_T$ implies that the target is generated using the target network.
11         Use $X$ and $y$ to update $\phi_P$ of the predict DQN by taking one gradient descent step based on the current
           learning rate, $\alpha$.
12         if $counter \% N_T == 0$ then
13             Set $\phi_T \leftarrow \phi_P$.
14         Set $x \leftarrow x'$, $a \leftarrow a'$, and $counter \leftarrow counter + 1$.
```

(c) Option 1 and Option 3. The reason is as follows. Deep Q-Learning does not calculate Q-values for each state-action pair. Hence, as far as calculating the Q-value is concerned, it can handle state and action spaces that are either discrete or continuous. However, in order to compute actions, Deep Q-Learning has to solve the following optimization problem

$$\pi(x) = \arg\max_{a \in A} \widehat{Q}(x, a)$$

which is trivial if action space A is discrete and difficult if it is continuous. Hence, in converntional sense, Deep Q-Learning can handle both discrete and continuous state spaces but only discrete action spaces.

(d) The output of the critic network is the estimated value function. The output of value function assumes continuous values. This suggests that the critic network is a regression model. The *conventional* output layer of a regression model has linear activation.

(e) The output layer of actor network generally uses softmax activation. This suggests that the actor network is a classification model. Hence, we will use categorical cross entropy loss to train actor network.

(f) The following are the steps to deduce the answer:

1. The input of both actor and critic networks are the states. Hence, input layer of critic network also has a size of 30.
2. Now, the hidden layer of critic network has 50 neurons. Hence, the number of weights of the hidden layer is $30 \cdot 50 = 1500$ and the number of biases is 50. Hence, the total number of parameters of the hidden layer is $1500 + 50 = 1550$.
3. The output of critic network is a scalar (the value function of the input state). Hence the size of the output layer is 1. Hence, the number of weights of the hidden layer is $50 \cdot 1 = 50$ and the number of biases is 1. Hence, the total number of parameters of the output layer is $50 + 1 = 51$.
4. Finally, the total number of paramters of the critic network is $1550 + 51 = 1601$.

Q5: State Estimation

(a) The question is indirectly asking us to to derive the recursive belief estimation equation of the Bayes filter. You can find it in lecture 36 notes.

(b) The equations of the system are,

$$\begin{aligned} x_t &= 0.8x_{t-1} + 2u_{t-1} + \varepsilon_{t-1} \\ z_t &= x_t + \delta_t \end{aligned}$$

Now, ε_{t-1} has a non-zero mean of 0.5. But Kalman filter assumes that ε_t has zero mean. This can be resolved by introducing a random variab$\overline{\varepsilon}_t$ that has zero mean but the same standard deviation as ε_t. Then, $\varepsilon_t = 0.5 + \overline{\varepsilon}_t$. Now consider the following,

$$\begin{aligned} x_t &= 0.8x_{t-1} + 2u_{t-1} + \varepsilon_{t-1} \\ &= 0.8x_{t-1} + 2u_{t-1} + 0.5 + \overline{\varepsilon}_t \\ &= 0.8x_{t-1} + 2\left(u_{t-1} + 0.25\right) + \overline{\varepsilon}_t \\ &= 0.8x_{t-1} + 2\overline{u}_{t-1} + \overline{\varepsilon}_t \end{aligned}$$

where $\overline{u}_t = u_t + 0.25$. Now, we can use the recursive update rule of Kalman filter. For this problem, $A_t = 0.8$, $B_t = 2$, $R_t = 4^2 = 16$, $C_t = 1$, $Q_t = 2^2 = 4$. Substituting these values in the recursive update rule of Kalman filter we get,

$$\begin{aligned} \overline{\mu}_t &= 0.8\mu_{t-1} + 2\left(u_{t-1} + 0.25\right) \\ &= 0.8\mu_{t-1} + 2u_{t-1} + 0.5 \end{aligned}$$

$$\begin{aligned} \overline{\Sigma}_t &= 0.8\Sigma_{t-1}0.8 + 16 \\ &= 0.64\Sigma_{t-1} + 16 \end{aligned}$$

$$\begin{aligned} K_t &= \overline{\Sigma}_t \cdot 1 \cdot \left(1 \cdot \overline{\Sigma}_t \cdot 1 + 4\right)^{-1} \\ &= \frac{\overline{\Sigma}_t}{\overline{\Sigma}_t + 4} \end{aligned}$$

$$\begin{aligned} \mu_t &= \overline{\mu}_t + K_t\left(z_t - 1 \cdot \overline{\mu}_t\right) \\ &= \overline{\mu}_t + K_t\left(z_t - \overline{\mu}_t\right) \end{aligned}$$

$$\begin{aligned} \Sigma_t &= \left(1 - K_t \cdot 1\right)\overline{\Sigma}_t \\ &= \left(1 - K_t\right)\overline{\Sigma}_t \end{aligned}$$

The output of the Kalman filter for every time slot is μ_t.