Last Name ..... First Name.....

Artificial Neural Networks: Exam (with solutions)
June 2022

- Keep your bag next to your chair, but do not open it during the exam.
- Write your name in legible letters on top of this page.
- The exam lasts 180 min.
- Write **all** your answers in a legible way on the exam in the answer spaces (do not use additional space or additional sheets for answers).
- Colored scratch paper for calculations is available - but notes on the scratch paper will not be taken into account during the correction of the exam.
- No documentation is allowed (no textbook, no slides), except **one page A5 of handwritten notes, doublesided**.
- No calculator is allowed.
- Have your student card displayed in front of you on your desk.
- **Check that your exam has 14 pages**

Evaluation:

1. ..... / 9 pts (Section 1, BackProp)
2. ..... / 7 pts (Section 2, DQN)
3. ..... / 11 pts (Section 3, Inference Prior)
4. ..... / 6 pts (Section 4, actor-critic)
5. ..... / 7 pts (Section 5, Convergence of RL)

Total: ..... / 40 pts

Definitions and notations

RL stands for Reinforcement Learning and ANN for Artificial Neural Network

ANN: Bold face symbols refer to vectors, normal face to a single component or a single input/output. Unless noted otherwise, the input is M-dimensional: $\mathbf{x}^{(0)} \in R^M$. The total input to a unit i in an artificial neural network with standard coding is denoted by $a_i = \sum_j w_{ij} x_j$ with weights w_{ij} and the output of that same unit by $g(a_i)$. In some cases, upper indices of a_i, w_{ij}, x_j refer to the layer. The function g is called the transfer function. Unless explicitly stated otherwise, the threshold is implicit in the weights.

RL: The symbol η is reserved for the learning rate.

In the context of RL, the symbol a refers to an action; the symbols r and R to a reward; the symbol s to a discrete state; and the symbol γ to a discount rate. If the state space is continuous then states are also written as $\mathbf{x}$.

In many RL applications, there is a single goal state (terminal state). An action sequence that ends at the goal is called an episode.

How to give answers

Most sections involve calculations. **Please write the answers in the space provided for that purpose.**

We also provide some free space for calculations. We will not look at these parts for grading. You can ask for extra scratch paper. We will not look at the scratch paper.

1 BackProp for Importance evaluation (9 points)

Your friend Anna uses a network taking as input an input image $\mathbf{x}^{(0)}$ with 1024 pixels normalized to zero mean (i.e., $\sum_m x_m^{(0)} = 0$), a first hidden layer with N neurons, a second hidden layers with N neurons, and an output layer with 76 neurons representing 76 different classes. She has chosen the output units to be linear: $y_k := x_k^{(3)} = \sum_j w_{kj}^{(3)} x_j^{(2)}$.

The transfer functions of the neurons in the two hidden layers are $g(a) = \tanh(a)$ with derivative $g' = 1 - g^2$.

Anna plans to perform classification by a simple max-operation on the output, i.e. she assigns the the input to class k^* where $k^* = \text{argmax}_k y_k$.

Her aim is to find out how important a given input pixel $x_n^{(0)}$ is for determining the class label. To this end she defines, firstly, the discrimination power at the output as

(i) $P_d = y_{k^*} - \frac{1}{76} \sum_{k=1}^{76} y_k$;

and secondly, the relative importance of input pixel n as

(ii) $V_{imp}(n) = x_n^{(0)} \frac{dP_d}{x_n^{(0)}}$.

Her idea is that if a pixel has a big positive value $x_n^{(0)} > 0$ and increasing the value further increases the discrimination power, then this pixel is important. Similarly, if a pixel has a big negative value $x_n^{(0)} < 0$ and decreasing the value further increases the discrimination power, then this is also an important pixel. Pixels that are either zero, or whose exact value does not matter are not important.

Hence, the most important pixel is the pixel $n^* = \text{argmax}_n V_{imp}(n)$ and this is the one she wants to determine.

She convinces you that this is an interesting idea and wants your help with the implementation of the algorithm.

A longer calculation by hand with application of the chain rule yields

$$V_{imp}(n) = \sum_{i=1}^{N} \sum_{j=1}^{N} \sum_{k=1}^{76} x_n^{(0)} w_{in}^{(1)} w_{ji}^{(2)} w_{kj}^{(3)} \left[1 - (g(a_i^{(1)}))^2\right] \left[1 - (g(a_j^{(2)}))^2\right] \left[\delta_{kk^*} - \tfrac{1}{76}\right]$$

Your task is to propose an efficient*[1] algorithm, using ideas from the backprop algorithm, to determine the most important input pixel.

Summarize your results in pseudo-code using the layout on the next page:

[1] i.e., an algorithm with optimal scaling when changing the number N of hidden neuron

Copy of formula for convencience

$V_{imp}(n) = \sum_{i=1}^{N} \sum_{j=1}^{N} \sum_{k=1}^{76} x_n^{(0)} w_{in}^{(1)} w_{ji}^{(2)} w_{kj}^{(3)} \, [1-(g(a_i^{(1)}))^2] \, [1-(g(a_j^{(2)}))^2] \, [\delta_{kk^*} - \frac{1}{76}]$

Pseudocode

(0) Initialization

Parameters $w_{ij}^{(l)}$ are loaded and kept fixed thereafter.

The imput image $\mathbf{x}^{(0)}$ is loaded and kept fixed thereafter.

(a) Forward pass

Evaluate/iterate/loop over

$$x_i^{(1)} = g(\sum_{n=1}^{1024} w_{in}^{(1)} x_n^{(0)}) \quad \text{for} \quad 1 \le i \le N$$
$$x_j^{(2)} = g(\sum_{i=1}^{N} w_{ji}^{(2)} x_i^{(1)}) \quad \text{for} \quad 1 \le j \le N$$
$$y_k = x_k^{(3)} = \sum_{j=1}^{N} w_{kj}^{(3)} x_j^{(2)} \quad \text{for} \quad 1 \le k \le 76$$

I need to store the following variables for later use:

$$x_i^{(1)} \text{ and } x_j^{(2)} \quad \text{for} \quad 1 \le i, j \le N$$
$$x_k^{(3)} \quad \text{for} \quad 1 \le k \le 76$$

number of points: ...../ 2

(b) Backward pass

Evaluate/iterate/loop over

$$\delta_k^{(3)} = \delta_{kk^*} - 1/76 \quad \text{for} \quad 1 \le k \le 76$$
$$\delta_j^{(2)} = [1 - (x_j^{(2)})^2] \sum_{k=1}^{76} w_{kj}^{(3)} \delta_k^{(3)} \quad \text{for} \quad 1 \le j \le N$$
$$\delta_i^{(1)} = [1 - (x_i^{(1)})^2] \sum_{j=1}^{N} w_{ji}^{(2)} \delta_j^{(2)} \quad \text{for} \quad 1 \le i \le N$$
$$\delta_n^{(0)} = \sum_{i=1}^{N} w_{in}^{(1)} \delta_i^{(1)} \quad \text{for} \quad 1 \le n \le 1024$$

I need to store the following variables for later use:

$$\delta_n^{(0)} \quad \text{for} \quad 1 \le n \le 1024$$

number of points: ...../ 2

(c) Detection of the most important pixel using variables stored in (a) and (b)

$$n^* = \arg\max_n x_n^{(0)} \delta_n^{(0)}$$

number of points: ...../ 3

(d) In the first implementation of Anna, the network has two hidden layers with $N = 100$ nonlinear neurons in each hidden layer. With your algorithm defined in (a) - (c), can you predict how the algorithm will scale if you increase the number of hidden neurons to $N = 2000$ per hidden layer?

If I increase the number of hidden neurons by a factor 20, the algorithm is expected to take longer by a factor of (approximately) 50

because of $\mathcal{O}(N^2 + 1100N)$ computations, i.e. $\frac{2000^2+1100\cdot2000}{100^2+1100\cdot100} \approx 50$.

number of points: ...../ 2

Free space for your calculations, do not use to write down answers.

2 DQN analysis for 2-step Q-learning (7 points)

We approximate Q-values by function approximation in a deep neural network with weights $w_{ij}^{(k)}$. The input $\mathbf{x}$ is continuous.

(i) Write down a reasonable loss function resulting from the consistency condition of the Bellman equation for the case of **2-step Q-learning**.

Loss L $= \frac{1}{2}\delta^2$ with $\delta = (r + \gamma r' + \gamma^2 \max_{a''} Q_w(s'', a'') - Q_w(s, a))$

number of points: ...../ 0.5

(ii) Write down the online update rule derived from point (i) using **semi-gradient**:

Update $\Delta w = \eta\delta \frac{dQ_w}{dw}(s, a)$

number of points: ...../ 1.5

(iii) In class we discussed that in DQN the authors introduced two separate sets of Q-values, a fast one $Q_{fast}(s, a)$ and a slow one $Q_{slow}(s, a)$. Analogous to parts (i) and (ii) above, write down the loss function for two-step DQN and the corresponding online update rule using **full gradient**

Loss $\frac{1}{2}\delta^2$ with $\delta = (r + \gamma r' + \gamma^2 \max_{a''} Q_{slow}(s'', a'') - Q_{fast,w}(s, a))$

Update $\Delta w = \eta\delta \frac{dQ_{fast,w}}{dw}(s, a)$ and put $Q_{slow} \leftarrow Q_{fast,w}$ every C steps.

number of points: ...../ 2

(iv) In class we discussed that the set of slow Q-values is updated every C steps. Show that for $C = 1$ the **full** gradient of (iii) is equivalent to the **semi-gradient** in (i) and (ii).

For $C = 1$, we always have $Q_{slow} = Q_{fast,w}$, hence δ is the same for (i)-(ii) and (iii); at the same time, full gradient of L in (iii) ignores Q_{slow}, which is equivalent to the semi-gradient of L in (i)-(ii).

number of points: ...../ 2

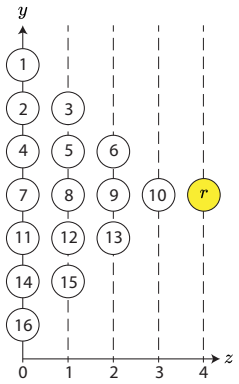
(v) Semigradient and slow-fast are both 'tricks of the trade' in RL. Comment on the insights from (iv).

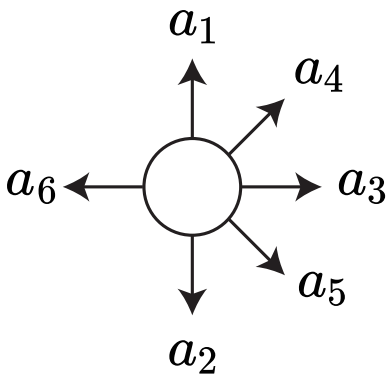
They are heuristics that help to stabilize training of the network.

number of points: ...../ 1

Free space for your calculations, do not use to write down answers.

3 Inference Prior and Reinforcement Learning (11 points)





We consider a 2-dimensional discrete environment with 16 states (Figure) plus one goal state where the agent receives a positive reward r. States are arranged in a triangular fashion in two dimensions. States are labeled as shown in the figure on the left. Available actions (figure on the right) are a1=up, a2=down, a3=right, a4=diagonally up right, a5=diagonally down right, a6= left (whenever these moves are possible). Returns are possible, e.g., the action up can be immediately followed by the action down.

Suppose that we use **function approximation for $Q(a, X) = \sum_j w_{aj} x_j$ with continuous state representation** X with the following encoding scheme: Input is encoded in 18 dimensions $X = (x_1, x_2, ...x_{16}, x_{17}, x_{18})$ where the first 16 entries are 1-hot encoded discrete states; entry 17 is $x_{17} = 0.5(z+1)$ and $x_{18} = 0.1$ where z is the horizontal coordinate of the environment (figure).

Before the first episode, we initialize all weights at zero. During the first episode, we update Q-values using the Q-learning algorithm in continuous space derived with the semi-gradient method from the Q-learning error function.

(i) Write down the quadratic loss function for 1-step Q-learning.

$\frac{1}{2}\delta^2$ with $\delta = (r + \gamma \max_{a'} Q(X', a') - Q(X, a))$
number of points: ...../ 0.5

(ii) Using the semi-gradient update rule, what are the new weight values w_{ai} and Q-values $Q(s, a)$ for **all 16 states and all actions** at the end of the first episode? **Write down all weights and Q-values that have changed.**

$$w_{aj} = \begin{cases} \eta r & \text{if } a = a_3 \text{ and } j = 10 \\ 2\eta r & \text{if } a = a_3 \text{ and } j = 17 \\ 0.1\eta r & \text{if } a = a_3 \text{ and } j = 18 \\ 0 & \text{otherwise} \end{cases} \quad \text{and} \quad Q(X, a) = \begin{cases} \eta r \left[\delta_{x_{10},1} + (z+1) + 0.01\right] & \text{if } a = a_3 \\ 0 & \text{otherwise} \end{cases}$$

number of points: ...../ 3.5

(iii) In episode 2 you use a greedy policy in which ties are broken by random search. What is the probabilty p that the agent will choose a path with a minimal number of steps to the goal? Consider two initial states:

Start at state 7. $p = 1 \cdot 1 \cdot 1 \cdot 1 = 1$

Start at state 11. $p = 1 \cdot 1 \cdot \frac{1}{3} \cdot 1 = \frac{1}{3}$
number of points: ...../ 1

(iv) Is this behavior for episode 2 typical for 1-step Q-learning? Comment on your result in (iii) in view of the no-free lunch theorem. (DO NOT write down the no-free lunch theorem, but use it in order to interpret your result.)
No, it is a consequence of the functional form of Q function: it generalizes that the good actions are similar in all states. This form is harmful in environment where the assumption is not satisfied (which is the price of the served lunch).
number of points: ...../ 2

(v) What can you say about the inference prior of the variable x_{18}? To let you focus on the role of x_{18}, consider for a moment the representation $x_{17} = \alpha[z - \beta]$ with $\alpha = 0$ (instead of $\alpha = 0.5$).

In simple words, the inference prior of variable x_{18} is that the good action is the same for all states.
number of points: ...../ 1

(vi) What can you say about the inference prior of the variable x_{17}? To answer this question consider the representation $x_{17} = \alpha[z - \beta]$ and redo on a scratch paper the calculations as in (ii). Then compare parameters $\alpha = 0.5$ and $\beta = 2$ with parameters $\alpha = 0.5$ and $\beta = -1$.

In simple words, the inference prior of variable x_{17} is that the good action is similar among all states with $z < \beta$ but different from all states with $z > \beta$. Hence, for $\beta = 2$, agents starting from $X = 7$ will never take the direct path to the goal!
number of points: ...../ 1

What happens if the sign of α is switched from +1 to -1?

The sign of α does not matter because only α^2 appears in the Q values.
number of points: ...../ 1

(vii) What would be a great choice of functional representation for input x_{17} and x_{18} if you know that the reward is located a state 6 with coordinates $(z, y) = (2, 1)$?

$x_{17} = (z - 2)$ and $x_{18} = (y - 1)$.
number of points: ...../ 1

4 4-step advantage actor-critic (6 points)

In class we have studied 1-step TD methods as well as n-step TD methods. We also have studied the advantage actor critic network. Here we bring different aspects together.

(i) Write down an error function (loss function) for 4-step V-value learning assuming that V-values are represented by function approximation

$L(w) = \frac{1}{2}\delta_t^2$ with $\delta_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 r_{t+4} + \gamma^4 V_w(s_{t+4}) - V_w(s_t)$

number of points: ...../ 0.5

(ii) Therefore, the update rule using semi-gradient for 4-step V-value learning is

$\Delta w = \eta \delta_t \frac{dV_w}{dw}(s_t)$

number of points: ...../ 1

(iii) Write down the update rule for the actor in a 1-step advantage actor-critic architecture.

$\Delta\theta = -\eta \delta'_t \frac{d \log \pi_\theta}{d\theta}(a_t|s_t)$ with $\delta'_t = r_{t+1} + \gamma V_w(s_{t+1}) - V_w(s_t)$.

number of points: ...../ 0.5

(iv) In practical applications people sometimes use n-step actor-critic. Write down the update rule for both actor and critic in the 4-step advantage actor-critic architecture.

actor $\Delta\theta = -\eta \delta_t \frac{d \log \pi_\theta}{d\theta}(a_t|s_t)$

critic $\Delta w = \eta \delta_t \frac{dV_w}{dw}(s_t)$ both with $\delta_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 r_{t+4} + \gamma^4 V_w(s_{t+4}) - V_w(s_t)$

number of points: ...../ 1

Space for calculations. Do not use to write answers

(v) Assume that we have as many inputs as states and we use for both actor and critic the same one-hot encoding of input states. V-values are inialized at zero and the actor is initialized by the random policy. Updating is done with the update rule of the 4-step advantage actor critic.

Is the following statement true for the 4-step actor-critic algorithm?

The first episode ends at the single rewarded goal state. If the neural networks for actor and critic have no hidden layer, then action policy and V-values of a state that is (via the shortest path) 5 or more steps away from the goal has, before the start of the second episode the same value as before the start of the first episode. Please answer by ticking one of the choices

[X] Yes or No [ ]

Would your answer change if we drop the condition 'no hidden layer'? Give an argument

My answer would change because there may be generalization from one state to another via the mixed representation of states in the hidden layer.

number of points: ...../ 1

(vi) As before we assume that we have as many inputs as states and we use for the actor one-hot encoding of input states. However, let us now assume that we have **no critic so that all V-values stay zero** and are never updated. Nothing else changes in the update rule. The first episode ends at the single rewarded goal state. What is different in episode 2 and 3 for the model without critic compared to the advantage-actor critic discussed before?
The update rule for the actor remains the same but with $\delta_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 r_{t+4}$ that is a truncated return. In case that we have a single rewarded goal state, all states that are more than 5 steps from the goal are NEVER updated. Hence, information never propagates back more than 4 steps.

What is the relation of the case without critic ($V = 0$) to standard policy gradient?

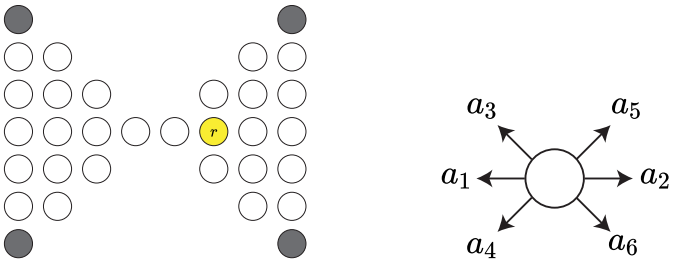
The algorithm is an approximation of the standard policy gradient. The approximation becomes more accurate as γ decreases.

number of points: ...../ 2

Space for calculations. Do not use to write answers

5 Batch-SARSA (7 points)

An agent moves in an environment of double-triangle form as indicated in the left figure



The environment contains a total of 31 unrewarded discrete states (open circles) and one rewarded states r (yellow). Available actions (right figure) are left, right, diagonally up left, diagonally down left, diagonally up right, diagonally down right (whenever these moves are possible,).

Each episode start at one of the 4 corners (upper left, uper right, lower left, lower right, grey-filled circles) and terminates at the reward location where the reward r is received. No actions are possible at the reward location.

You have already played four full episodes with **1-step SARSA using softmax policy** starting once from each of four corners and updated the Q-values online. After four episodes the values are $Q_4(s,a)$. You find that with online SARSA Q-values are noisy and converge only slowly. Therefore you want to run a batch version. You consider three different versions.

A) You generate 1860 short episode segments as follows: The agent starts from each of the 31 non-rewarded states s exactly 60 times. To reduce stochasticity of action choices, you decide that **in state s, the agent takes each of the available actions equally often** (i.e., for starting states s with 6 action choices, each action is taken 10 times; for starting states with 4 action choices, each action is taken 15 times etc). Thus, in state s action choices are unbiased. However, to reflect action choices of SARSA, you decide that **in state s' the action a' is chosen according to the softmax function evaluated with** $Q_4(s',a')$ where s, s' are the starting state and the next observed state. You collect the transition information (s,a,r,s',a') for each of the short segments starting at state s.

Finally you do a batch update. For all pairs (s,a) you set:

$$Q_5(s,a) = Q_4(s,a) + \frac{\eta}{N(s,a)} \sum_{i=1}^{N(s,a)} [r(i) - Q_4(s,a) + \gamma \sum_{a'} \pi(s'(i),a') Q_4(s'(i),a')]$$

where $N(s,a)$ is the number of times you have chosen action a in state s and $\pi(s,a)$ the softmax policy.

B) You generate 1860 short episode segments as follows: The agent starts from each of the 31 non-rewarded states 60 times. To reflect the SARSA algorithm, **the agent chooses each time in state s a short sequence of two actions according the softmax function $Q_4(s,a)$ and $Q_4(s',a')$** where s, s' are the starting state and the next observed state. You collect the transition information (s, a, r, s', a') for each of the 60 short segements starting at state s.

Finally you do a batch update. For all pairs (s,a) with $N(s,a) \neq 0$ you set:

$$Q_5(s,a) = Q_4(s,a) + \frac{\eta}{N(s,a)} \sum_{i=1}^{N(s,a)} [r(i) - Q_4(s,a) + \gamma \sum_{a'} \pi(s'(i),a') Q_4(s'(i),a')]$$

where $\pi(s,a)$ the softmax policy, $N(s,a)$ is the number of times you have chosen action a in state s, and the counter i runs over these cases. For all pairs (s,a) with $N(s,a) = 0$ you set $Q_5(s,a) = Q_4(s,a)$

C) You generate 1860 short episode segments as follows: The agent starts systematically in one of the four corners and you rotate starting states. The first episode starts in the upper-right corner, the second one in the lower-right, the third one in the lower left, the fourth one in the upper left, the fifth one again in the upper right etc. **Each episode is played using SARSA with softmax with the set of Q-values Q_4 and ends at the rewarded state** with a reward r. At every point in time, you collect the transition information (s, a, r, s', a') and store these as short episode segments. You stop the simulation when you have collected 1860 of these transitions (=episode segments).

Finally you do a batch update. For all pairs (s,a) with $N(s,a) \neq 0$ you set:

$$Q_5(s,a) = Q_4(s,a) + \frac{\eta}{N(s,a)} \sum_{i=1}^{N(s,a)} [r(i) - Q_4(s,a) + \gamma \sum_{a'} \pi(s'(i),a') Q_4(s'(i),a')]$$

where $\pi(s,a)$ the softmax policy, $N(s,a)$ is the number of times you have chosen action a in state s, and the counter i runs over these cases. For all pairs (s,a) with $N(s,a) = 0$ you set $Q_5(s,a) = Q_4(s,a)$

In all cases A,B,C, you repeat these batch updates until convergence, i.e., with $Q_5(s,a)$ you simulate another 1860 short segments which gives you Q_6 etc.

(i) For at least one of the cases A or B or C, we claim that if the batch update has converged, then the resulting Q-values solve the Bellman equation for softmax policy. Give a proof sketch for this statement! (To do so choose one of the cases from A/B/C that you believe works!)

Case A: Since $N(s,a) > 0$ for all pairs of (s,a) by construction, we have

$$Q_{n+1}(s,a) = (1-\eta)Q_n(s,a) + \eta \frac{1}{N(s,a)} \sum_{i=1}^{N(s,a)} \Big[r(i) + \gamma \sum_{a'} \pi_n(a'|s'(i)) Q_n(s'(i),a') \Big].$$

Since the environment is deterministic, $r(i) = r(s,a)$ and $s'(i) = s'(s,a)$ are always the same and independent of i; we hence have

$$Q_{n+1}(s,a) = (1-\eta)Q_n(s,a) + \Big[r(s,a) + \gamma \sum_{a'} \pi_n(a'|s'(i)) Q_n(s'(s,a),a') \Big].$$

As a result, after convergence, we have

$$Q(s,a) = r(s,a) + \gamma \sum_{a'} \pi(a'|s'(i)) Q(s'(s,a),a'),$$

which is the Bellman equation for the softmax policy in a deterministic environment.

number of points: ...../ 3

(ii) would your proof sketch in (i) also work for the other two cases? Why/Why not? Give an argument.

If we have $N(s,a) \to \infty$ for all pairs of (s,a) as $n \to \infty$, then everything in the proof above can be applied to the other cases. A finite softmax temperature guarantees this.

number of points: ...../ 1

(iii) Overall, do you think that one of the three choices is 'better'? (e.g., this could mean that only one of them works but not the other, or that one needs less data than the other, or that one is easier to implement than the other etc.). Give an argument for your choice

Version **A** (OR **C**) is 'better' than the other two because it searches more systematically the space of state-actions and is hence more data efficient for estimating Q-values (OR, for **C**, because it updates faster the (s,a) pairs that are more important for finding the reward and hence more data efficient for finding a good policy).

number of points: ...../ 1

(iv) Can you relate one of the three cases (A, B, or C) to tricks used in Deep Reinforcement Learning? If so, which case and which trick of Deep RL? Pick one relation and give an argument in favor of this relation. You should also mention remaining differences.

B/**C** is a bit like having an expected SARSA with eplay buffer when playing, but the update is at fixed intervals.

C is also a bit like running multiple agents in parallel with updates at fixed intervals.

number of points: ...../ 2