Exam CS4400: Deep Reinforcement Learning

01-02-2023 | 9:00–12:00

Student name: _____

Student number: _____

- This is a closed-book individual examination with **9 questions** and a total of **100 points**.
- Do not open the exam before the official start of the examination.
- If you feel sick or otherwise unable to take the examination, please indicate this *before* the exam starts.
- The examination lasts **180 minutes** after the official start.
- This gives you a bit under 2 minutes per point. Use your time carefully!
- You can hand in your exam solution any time until 15 minutes before the end of the exam and leave the examination room quietly. In the last 15 minutes, no one can leave the examination room to help other students concentrate on finishing their exam.
- Only one student can visit the bathroom at the same time. In the last 15 minutes, no bathroom visits are possible.
- Use of course books, readers, notes, and slides is **not** permitted
- Use of (graphical) calculators or mobile computing devices (including mobile phones) is **not** permitted.
- Write down your name and student number above.
- Write your **student number on each sheet** of the exam after the exam started.
- You can write your answer on the free space under each question.
- If you need more space, use the back of another exam-sheet and write where to find the answer under the question. Ask for additional empty pages if you need them.
- Use pens with black or blue ink. Pencils and red ink are not allowed!
- Clearly cross out invalid answers. If two answers are given, we consider the one with less points!
- Write clearly, use correct English, and avoid verbose explanations. Giving irrelevant information may lead to a reduction in your score.
- This exam covers all information on the slides of the course, the tutorials and everything discussed in lectures.
- This exam assumes a familiarity with the stated background knowledge of the course.
- The total number of pages of this exam is 11 (excluding this front page).
- Exam prepared by Wendelin Böhmer.

Question 1 (multiple choice): **(40 points)**

Please mark only the correct answers with a **cross** like this: ☒. If you wish to **unmark** a marked answer, **fill** the entire square and **draw an empty** one next to it like this: ☐ ■

Only **one answer per question** is correct. You will receive 2 points per correct answer, except when multiple squares are marked. Wrong answers yield no points, but are also not punished. Good luck!

1.1: Why is the following for-loop **not** considered *efficient* for deep learning?

```
for i in range(n): loss = loss + mse_loss(f(x[i,:]), y[i])
```

- ☐ because in a for loop we cannot mask out non-existing time-steps
- ☐ because here the loss is defined recursively and no *base case* is defined
- ☒ because the depth of the computation graph grows linearly in `n`
- ☐ because `loss=loss+...` overwrites the `loss` variable every iteration

1.2: Which of the following is a parameter of the `torch.nn.LSTM` class?

- ☐ `stride`
- ☐ `padding`
- ☐ `kernel_size`
- ☒ `input_size`

1.3: What is the input to a *Recurrent Neural Network* (RNN)?

- ☐ a tensor with one vector per i.i.d. drawn sample
- ☒ a tensor with one ordered set of vectors per sample
- ☐ a tensor with one unordered set of vectors per sample
- ☐ tensors representing one graph with nodes annotated by vectors per sample

1.4: Which of the following *losses* has been discussed in the lecture?

- ☐ $\mathbb{E}\left[\left(r + \max_{a'} Q(s', a') - \gamma Q(s, a)\right)^2 \,\middle|\, (s, a, r, s') \sim \mathcal{D}\right]$
- ☒ $\mathbb{E}\left[\left(r + \gamma \max_{a'} Q(s', a') - Q(s, a)\right)^2 \,\middle|\, (s, a, r, s') \sim \mathcal{D}\right]$
- ☐ $\mathbb{E}\left[\left(r + Q(s, a) - \gamma \max_{a'} Q(s', a')\right)^2 \,\middle|\, (s, a, r, s') \sim \mathcal{D}\right]$
- ☐ $\mathbb{E}\left[\left(r + \gamma Q(s, a) - \max_{a'} Q(s', a')\right)^2 \,\middle|\, (s, a, r, s') \sim \mathcal{D}\right]$

1.5: Which algorithm can **not** use *target networks*?

- ☒ Residual-gradient TD-learning
- ☐ Semi-gradient TD-learning
- ☐ Deep Q-networks
- ☐ Neural Fitted Q-iteration

1.6: Which *property of RL* violates classical ML assumptions, e.g. made in behavior cloning?

- ☐ the state space grows exponential with the state-dimensions
- ☐ action-spaces can be continuous
- ☒ exploration changes the state distribution
- ☐ transitions can be stochastic

1.7: Changing which parameter makes the DQN algorithm more *sample efficient*?

- ☐ decrease the gradient updates per sampled tajectory
- ☒ decrease the number of environmental steps between gradient updates
- ☒ increase the time between target number updates
- ☐ increase the time during which the exploration is decayed

Remark: there is a pretty big typo in the 3rd answer, it was supposed to say "increase the time between target network updates". As is, the answer does not make sense, but counts as correct due to ambiguity.

1.8: Which class is **not** part of the standard RL *software architecture* from the lectures?

- ☐ Learner
- ☐ Runner
- ☒ Explorer
- ☐ Model

1.9: Which of the following definitions is **not** an *on-policy value target* discussed in the course?

- ☐ $\sum_{k=0}^{\infty} \gamma^k r_{t+k}$
- ☐ $\sum_{k=0}^{n-1} \gamma^k r_{t+k} + \gamma^n V^\pi(s_{t+n})$
- ☐ $\sum_{k=0}^{\infty} (\lambda\gamma)^k \left(r_{t+k} + \gamma\,(1-\lambda)\, V^\pi(s_{t+k+1})\right)$
- ☒ $\sum_{k=0}^{n-1} (\lambda\gamma)^k r_{t+k} + \left(\gamma\,(1-\lambda)\right)^n V^\pi(s_{t+n})$

1.10: Which is the PPO loss $\mathcal{L}_\mu^{\text{clip}}[\theta]$?

- ☐ $-\sum_{t=0}^{n-1} \gamma^t \mathbb{E}_\mu \left[\min \left(A_t \frac{\mu(a_t|s_t)}{\pi_\theta(a_t|s_t)}, A_t \,\text{clip}(\frac{\mu(a_t|s_t)}{\pi_\theta(a_t|s_t)}, 1-\epsilon, 1+\epsilon) \right) \right]$
- ☐ $-\sum_{t=0}^{n-1} \gamma^t \mathbb{E}_\mu \left[\max \left(A_t \frac{\mu(a_t|s_t)}{\pi_\theta(a_t|s_t)}, A_t \,\text{clip}(\frac{\mu(a_t|s_t)}{\pi_\theta(a_t|s_t)}, 1-\epsilon, 1+\epsilon) \right) \right]$
- ☒ $-\sum_{t=0}^{n-1} \gamma^t \mathbb{E}_\mu \left[\min \left(A_t \frac{\pi_\theta(a_t|s_t)}{\mu(a_t|s_t)}, A_t \,\text{clip}(\frac{\pi_\theta(a_t|s_t)}{\mu(a_t|s_t)}, 1-\epsilon, 1+\epsilon) \right) \right]$
- ☐ $-\sum_{t=0}^{n-1} \gamma^t \mathbb{E}_\mu \left[\max \left(A_t \frac{\pi_\theta(a_t|s_t)}{\mu(a_t|s_t)}, A_t \,\text{clip}(\frac{\pi_\theta(a_t|s_t)}{\mu(a_t|s_t)}, 1-\epsilon, 1+\epsilon) \right) \right]$

1.11: Which algorithm can **not** work with *continuous actions*?

- ☒ DRQN
- ☐ Reinforce
- ☐ PPO
- ☐ DDPG

1.12: How does TD3 *differ* from DDPG?

- ☐ TD3 uses stochastic policies via the reparametrization trick, DDPG does not
- ☒ TD3 trains two Q-value functions to be pessimistic, DDPG does not
- ☐ TD3 squashes actions for bounded action spaces, DDPG does not
- ☐ TD3 requires two separate optimizers, DDPG does not

1.13: Which of the following is **not** a type of *uncertainty* derived from expected generalization errors?

- ☐ Aleatoric
- ☐ Epistemic
- ☒ Entropy regularization
- ☐ Model-bias

1.14: What is the *value of the initial state* of a Markov chain with 10 states in a row, where the last state is terminal and each transition yields a reward of 1?

- ☐ $\frac{1}{1-\gamma}$
- ☒ $\frac{1-\gamma^9}{1-\gamma}$
- ☐ $\frac{1-\gamma^{10}}{1-\gamma}$
- ☐ $\frac{1-\gamma^{11}}{1-\gamma}$

Remark: The finite geometrical series is $\sum_{t=0}^{n} \gamma^t = \frac{1-\gamma^{n+1}}{1-\gamma}$. There are 10 states, but the last is terminal, so there are only 9 transitions. Summing the rewards yields $\sum_{t=0}^{8} \gamma^t = \frac{1-\gamma^9}{1-\gamma}$

1.15: Which of the following is **not** a main challenge of *offline deep RL*?

- ☐ distribution shift
- ☐ no exploration
- ☒ catastrophic forgetting
- ☐ learning stability

1.16: In offline RL, *policy constrained methods* perform well when ...

- ☐ a suitable distance measure can be found
- ☐ a suitable epistemic uncertainty measure can be found
- ☐ the sample distribution is close to random behavior
- ☒ the sample distribution is close to optimal behavior

1.17: How many **inputs** does a LSTM network with h memory cells require to represent a *policy* in a *partially observable* environment with states $\boldsymbol{s} \in \mathbb{R}^n$, observations $\boldsymbol{o} \in \mathbb{R}^m$, and actions $\boldsymbol{a} \in \mathbb{R}^k$?

- ☐ $n + k + h$
- ☐ $m + k + h$
- ☐ $n + k + 2h$
- ☒ $m + k + 2h$

1.18: Which games **cannot** be formaluted as POSG?

- ☐ games with simultaneous moves
- ☒ games with continuous time
- ☐ games with partial information
- ☐ games with deterministic moves

1.19: Which of the following *MARL algorithms* has a neural network that represents the *policy*?

- ☐ IQL
- ☒ COMA
- ☐ QMIX
- ☐ DCG

1.20: Which of the following techniques is **not** used in *multi-task learning*?

- ☒ QTRAN
- ☐ UVF
- ☐ HER
- ☐ DR

Question 2: (6 points)

In 6 sentences or less, explain the *zero-shot (ad-hoc) cooperation* pathology and give an example task in which it prevents agents to execute an optimal solution.

Solution

Zero-shot cooperation denotes situations in which two teams of agents are trained *independently* to solve the same game. As a result, the two teams can converge to different Nash equilibria and therefore fail to cooperate when playing with members of the other team. An example is the 10-levers game, where two agents must pull one of 10 levers each, and are given a collaborative reward if they pull the same lever. All levers give the same reward, except for one, which gives a slightly smaller reward. This lever is therefore suboptimal and no team will learn to pull it. However, as all other levers yield similar reward, independently trained agents will in all likelihood converge to different levers.

Rubrik:

- 2 point for mentioning independently trained teams of agents cooperating

- 2 point for mentioning that the teams can arrive at different NE
- 2 points for an applicable example task
- only 1 point if one of the above is mentioned only very indirect
- mentioning OtherPlay out of the blue also gets 1 point as a description

Question 3: (6 points)

(a) *[3 points]* Design a *cyclic* two-payer *zero-sum* normal-form game by defining player 1's reward:

$r^1(a^1, a^2) :=$

$\setminus$	a_1^2	a_2^2	a_3^2
a_1^1			
a_2^1			
a_3^1			

(b) *[3 points]* Name a *training method* that can solve cyclic games and explain why it works at the example of your game. You do not need to solve your game, just explain how it could be done.

Solution

Many games are possible, but for simplicity, here is rock paper scissors:

$r^1(a^1, a^2) :=$

$\setminus$	a_1^2	a_2^2	a_3^2
a_1^1	0	$+1$	-1
a_2^1	-1	0	$+1$
a_3^1	$+1$	-1	0

One method to solve cyclic games are leagues, where an agent is trained against past versions of itself. This works because the agent first learns all the single-action strategies, i.e., only play a_1, a_2 and a_3, which each can be beat by another single action. When playing against all past agents, the next agent must learn to beat all single actions, which is on average possible by playing stochastically.

Another method to solve cyclic games is to compute the *mixed Nash equilibrium* by parameterizing both agents' policies, analytically formulate the expected return and finding the saddle point in parameter space. The result for rock-paper-sciessors is that both agents play uniformly random.

Rubrik:

- 1 point for a game that has at most 1 Nash equilibrium
- 2 point for a game without Nash equilibria
- one point less for a general sum game
- 1 point for mentioning *either* leagues or mixed NE's
- 2 point for a good explanation of the mentioned concept
- only 1 point if the concept is explained very vaguely

Question 4: **(6 points)**

Which deep reinforcement learning *algorithm* and *architecture* from the lectures would you use to learn the game of *chess*, where the agent plays one side and a given chess computer plays the other? Give at least one argument to justify each of your choices.

Hint: it is sufficient to name and justify the required module-types, you do not need to draw how they are connected. Linear layers are always present, so they do not have to be named.

Solution

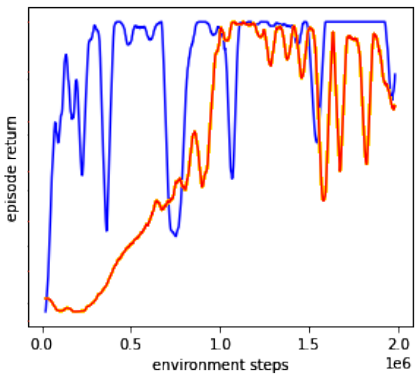
The game is *not* a multi-player task, as the opponent does not learn but is stationary. MARL algorithms are therefore not a good choice! The state is fully observable and can be learned with a CNN architecture. The action space is discrete, but does not need to be one-hot: one can e.g. encode the piece to be moved and the position it should be moved to each with 2 one-hot encoded coordinates.

Rubrik:

- 3 points for DQN, TRPO or PPO (discrete actions), only 2 points for IQL, DRQN, Reinforce or AC (inefficient or unnecessary), no points for DDPG, TD3 or SAC (require continuous actions) and no points for other MARL algorithms.
- 3 points for CNNs (input grid), 3 points for GNN or MHA with a good explanation (between pieces), 2 points for GNN or MHA with a bad explanation, no points for LSTM or any other (unless explanation works).
- 1 point less if the choice is suboptimal or the justification (minimally 0 points per category).
- 1 point less if a wrong choice is presented additionally (e.g. LSTM, but do not count follow-up errors with DRQN)
- 1 point if the algorithm/architecture is wrong, but the justification contains many good ideas
- No points if the choice is totally wrong or the justification unconnected.
- One can interpret "architecture" as software architecture, and reasonable answers should get full 3 points!

Question 5: **(6 points)**

The figure to the right plots the smoothed training returns for REINFORCE (blue, dark) and A2C (red, bright) learning the `CartPole-v1` task.



(a) *[2 points]* Explain why both algorithms take much more environmental interactions than e.g. DQN.

(b) *[2 points]* Explain why A2C takes more environemtal interactions (around 10^6) to learn a "good policy".

(c) *[2 points]* Name a related actor-ciritc algorithm that could improve sample efficiency and stability here.

Solution

There are of course multiple possible explanations, but the ones we expect here are:

(a) Both algorithms are *on-policy*, without repeated off-policy updates like in PPO, and can therefore only learn on rollouts of the current policy. This requires many rollouts and therefore many environmental interactions.

(b) A2C needs to learn a *value function* to compute the advantage, which takes more time than evaluating the policy with rollouts as in REINFORCE.

(c) TRPO or PPO are derived from A2C, but are much more sample efficient and stable.

Rubrik:

- 2 points for a coherent explanation of (a).
- no points for explanations that would also apply to DQN.
- 2 points for a coherent explanation of (b).
- no points for explanantions that would also apply to REINFORCE.
- 2 points for TRPO or PPO, but no points for DQN.
- only 1 point for DDPG, TD3 or SAC.
- only 1 point in either cathegory if the explanation is not coherent or the answer otherwise unclear.
- also only 1 point if an answer is given to the wrong subquestion

Question 6: (6 points)

Consider a *cooperative* two-player normal-form game $\mathcal{G}$ with the collaborative reward function $r(a^A, a^B)$:

A \ B	X	Y
X	4	-4
Y	0	2

and a policy for agent B: $\pi_\theta(a) = \begin{cases} \theta & \text{if } a = X \\ 1-\theta & \text{if } a = Y \end{cases}, \quad \theta \in [0,1]\,.$

(a) *[2 points]* Give the *joint actions* of all Nash-equilibria of $\mathcal{G}$ or indicate that none exist. You do not have to justify your answer.

(b) *[2 points]* Derive the *expected return* $q^A(a|\theta)$ if agent A plays action $a \in \{X, Y\}$ as function of θ.

(c) *[2 points]* Derive the range of θ for which agent A exhibits on average *relative overgeneralization*.

Solution

(a) The two Nash equilibria are (X, X) and (Y, Y).

Rubrik:

- 1 point per correct Nash equilibrium
- -1 point per wrong Nash equilibrium (minimum 0 points)

(b) The expected return of agent A given some action is $q^A(a|\theta) := \mathbb{E}[r(a, a')|a' \sim \pi_\theta]$. Thus: $q^A(X|\theta) = \sum_{a'} \pi_\theta(a') r(X, a') = 4\theta - 4(1-\theta) = 8\theta - 4$ and $q^A(Y|\theta) = 2(1-\theta) = 2 - 2\theta$.

Computing instead $\mathbb{E}[r(a, a')|a \sim U, a' \sim \pi_\theta] = \frac{1}{2}q(X|\theta) + \frac{1}{2}q(X|\theta) = 4\theta - 2 + 1 - \theta = 3\theta - 1$ is the wrong ansatz.

Rubrik:

- 1 point for a correct ansatz
- 1 point for comuting $q^A(a|\theta)$ correctly from the ansatz (correct or not)
- no points if computation has nothing to do with expected return

(c) $q^A(X|\theta) < q^A(Y|\theta) \Leftrightarrow 8\theta - 4 < 2 - 2\theta \Leftrightarrow 10\theta < 6 \Leftrightarrow \theta < \frac{6}{10} \quad \Rightarrow \quad \theta \in [0, \frac{6}{10})$.

Rubrik:

- 1 point for the correct ansatz $q^A(X|\theta) < q^A(Y|\theta)$
- 1 point computing the range of θ, or the upper bound, from the ansatz (correct or not)
- 1 point for explanaition, but no computation
- no points if computation has nothing to do with relative overgeneralization

Question 7: (6 points)

Let $f(\boldsymbol{x}) := \boldsymbol{a}^\top \boldsymbol{x} - b$ denote a linear function of $\boldsymbol{x} \in \mathbb{R}^m$, and let $\boldsymbol{\epsilon} \sim \mathcal{N}(\cdot|\mathbf{0}, \boldsymbol{\Sigma})$ denote a normal distributed *noise vector*. When $\boldsymbol{\epsilon}$ is added to the input, prove that the variance of the linear function f is

$$\mathbb{V}\big[f(\boldsymbol{x} + \boldsymbol{\epsilon})\big] \quad = \quad \boldsymbol{a}^\top \boldsymbol{\Sigma} \boldsymbol{a}\,, \qquad \forall \boldsymbol{x} \in \mathbb{R}^m\,.$$

Hint: note that $\boldsymbol{x}$ is given and therefore **not** random.

Solution

Note that $\mathbb{E}[\boldsymbol{\epsilon}] = \mathbf{0}$ and $\mathbb{E}[(\boldsymbol{\epsilon} - \mathbb{E}[\boldsymbol{\epsilon}])(\boldsymbol{\epsilon} - \mathbb{E}[\boldsymbol{\epsilon}])^\top] = \mathbb{E}[\boldsymbol{\epsilon}\boldsymbol{\epsilon}^\top] = \boldsymbol{\Sigma}$.

$$\begin{aligned}
\mathbb{E}[f(\boldsymbol{x} + \boldsymbol{\epsilon})] &= \mathbb{E}[\boldsymbol{a}^\top \boldsymbol{x} + \boldsymbol{a}^\top \boldsymbol{\epsilon} - b] \quad = \quad f(\boldsymbol{x}) + \boldsymbol{a}^\top \mathbb{E}[\boldsymbol{\epsilon}] \quad = \quad f(\boldsymbol{x})\,, \\
\mathbb{V}[f(\boldsymbol{x} + \boldsymbol{\epsilon})] &= \mathbb{E}\big[\big(f(\boldsymbol{x} + \boldsymbol{\epsilon}) - \mathbb{E}[f(\boldsymbol{x} + \boldsymbol{\epsilon})]\big)^2\big] \\
&= \mathbb{E}[(\boldsymbol{a}^\top \boldsymbol{\epsilon})^2] \quad = \quad \boldsymbol{a}^\top \mathbb{E}[\boldsymbol{\epsilon}\boldsymbol{\epsilon}^\top] \boldsymbol{a} \quad = \quad \boldsymbol{a}^\top \boldsymbol{\Sigma} \boldsymbol{a}\,.
\end{aligned}$$

Rubrik:

- 2 points for the correct definition of the variance $f(\boldsymbol{x} + \boldsymbol{\epsilon})$, even if not in vector format.
- 2 points for the correct definition of the mean of $f(\boldsymbol{x} + \boldsymbol{\epsilon})$, even if not in vector format.
- only 1 point in the two cases above for correct definition, but incorrect use and vice versa.
- 2 points for putting it all correctly together.

Alternative solution:

$$\mathbb{V}[f(\boldsymbol{x} + \boldsymbol{\epsilon})] \quad = \quad \mathbb{V}[\boldsymbol{a}^\top \boldsymbol{x} + \boldsymbol{a}^\top \boldsymbol{\epsilon} - b] \quad \overset{(1)}{=} \quad \mathbb{V}[\boldsymbol{a}^\top \boldsymbol{\epsilon}] \quad \overset{(2)}{=} \quad \boldsymbol{a}^\top \mathbb{V}[\boldsymbol{\epsilon}] \boldsymbol{a} \quad \overset{(3)}{=} \quad \boldsymbol{a}^\top \boldsymbol{\Sigma} \boldsymbol{a}\,.$$

Without having to justify this, (1) holds because constants do not affect variances, (2) because inner products multiply out this way, and (3) because $\mathbb{V}[\boldsymbol{\epsilon}] = \boldsymbol{\Sigma}$ by definition.

Rubrik:

- 2 points for some version of (1)
- 2 points for some version of (2)
- 2 points for some version of (3)

Question 8: **(10 points)**

For a given vector of values $\boldsymbol{q} \in \mathbb{R}^n$, a prior policy $\boldsymbol{\mu} \in \mathbb{R}^n$, with $\boldsymbol{\mu}^\top \mathbf{1} = 1, \mu_a \geq 0, 1 \leq a \leq n$, and a constant $\lambda > 0$, you will solve the following optimization problem, where $D_{\mathrm{KL}}[\boldsymbol{\pi} \| \boldsymbol{\mu}] = \boldsymbol{\pi}^\top \ln(\frac{\boldsymbol{\pi}}{\boldsymbol{\mu}})$:

$$\max_{\boldsymbol{\pi} \in \mathbb{R}^n} \boldsymbol{\pi}^\top \boldsymbol{q} - \lambda\, D_{\mathrm{KL}}[\boldsymbol{\pi} \| \boldsymbol{\mu}] \qquad \text{s.t.} \quad \boldsymbol{\pi}^\top \mathbf{1} = 1\,, \quad \pi_a \geq 0\,, \quad 1 \leq a \leq n\,.$$

Use the method of Lagrange multipliers (here only η), with the Lagrange function

$$L[\boldsymbol{\pi}, \eta] \quad := \quad \boldsymbol{\pi}^\top \boldsymbol{q} - \lambda\, \boldsymbol{\pi}^\top (\ln \boldsymbol{\pi} - \ln \boldsymbol{\mu}) + \eta\, (\boldsymbol{\pi}^\top \mathbf{1} - 1)\,, \qquad \text{s.t.} \quad \pi_a \geq 0\,, \quad 1 \leq a \leq n\,,$$

to prove that the solution is

$$\pi_a = \frac{\mu_a \exp(\frac{1}{\lambda} q_a)}{\sum_{b=1}^n \mu_b \exp(\frac{1}{\lambda} q_b)}\,, \qquad 1 \leq a \leq n\,.$$

Hint: You can ignore the constraints $\pi_a \geq 0$ during the derivation and then check whether they hold for your solution. This question might take you more time per point than others.

Solution

We have to compute the derivatives of $L[\boldsymbol{\pi}, \eta]$ w.r.t. the primary variables $\boldsymbol{\pi} \in \mathbb{R}^n$ and secondary variable $\eta \in \mathbb{R}$ ($\lambda \in \mathbb{R}$ is here a constant):

$$\begin{aligned}
\frac{\partial L}{\partial \pi_a} &= q_a - \lambda \ln \pi_a - \lambda \frac{\pi_a}{\pi_a} + \lambda \ln \mu_a + \eta \quad \stackrel{!}{=} \quad 0 \\
\Rightarrow \ln \pi_a &= \tfrac{1}{\lambda} q_a - 1 + \ln \mu_a + \tfrac{\eta}{\lambda} \\
\Rightarrow \pi_a &= \mu_a \, \exp(\tfrac{1}{\lambda} q_a) \, \exp(\tfrac{\eta}{\lambda} - 1) \\
\frac{\partial L}{\partial \eta} &= \boldsymbol{\pi}^\top \mathbf{1} - 1 \quad \stackrel{!}{=} \quad 0 \\
\Rightarrow 1 &= \sum_{b=1}^n \mu_b \, \exp(\tfrac{1}{\lambda} q_b) \, \exp(\tfrac{\eta}{\lambda} - 1) \\
\Rightarrow \exp(\tfrac{\eta}{\lambda} - 1) &= \frac{1}{\sum_{b=1}^n \mu_b \, \exp(\frac{1}{\lambda} q_b)} \\
\Rightarrow \pi_a &= \frac{\mu_a \, \exp(\frac{1}{\lambda} q_a)}{\sum_{b=1}^n \mu_b \, \exp(\frac{1}{\lambda} q_b)}
\end{aligned}$$

As all $\mu_a \geq 0$ and $\exp(x) \geq 0, \forall x \in \mathbb{R}$, we have $\pi_a \geq 0, 1 \leq a \leq n$.

Rubrik:

- 2 points for the correct derivative $\frac{\partial L}{\partial \pi_a}$. 1 point for small errors.
- 2 points for setting (even incorrect) $\frac{\partial L}{\partial \pi_a} = 0$. 1 point for the correct $\mathbf{1}^\top \frac{\partial L}{\partial \boldsymbol{\pi}} = 0$ or otherwise small errors.
- 2 points for the correct reformulation of $\frac{\partial L}{\partial \pi_a} = 0$ into π_a.
- 2 points for deriving the constraint $\boldsymbol{\pi}^\top \mathbf{1} = 1$ or using it to derive $\exp(\frac{\eta}{\lambda} - 1)$.
- no point deduction for using the constraint $\boldsymbol{\pi}^\top \mathbf{1} = 1$ directly (without deriving it from $\frac{\partial L}{\partial \eta}$).
- 2 points for putting it together into the correct final answer π_a. 1 point for small errors. No points if the path to the solution is not visible
- no point deduction for a missing argument why $\pi_a \geq 0$ holds.

Question 9: (programming) **(14 points)**

You only have to insert the missing code segment at the line(s) marked with *#YOUR CODE HERE*. Please use correct Python/PyTorch code. Singleton dimensions of tensors can be ignored, i.e., you do not need to (un)squeeze tensors. If you forget a specific command, you can define it first, both the signature (input/output parameters) and a short description what it does. Using your own definitions of existing PyTorch functions will not yield point deductions. If no similar PyTorch function exists, your definition will be considered as wrong code and you will not receive the corresponding points.

Implement the following loss for *pessimistic Q-learning* in the given `MyLearner` class **efficiently**:

$$\min_{\{\theta_i\}_{i=1}^k} \mathbb{E}\left[\frac{1}{k}\sum_{i=1}^{k}\frac{1}{n}\sum_{t=1}^{n}\left(r_t + \gamma \max_{a'\in\mathcal{A}} \min_{1\le j\le k} Q_{\theta'_j}(\boldsymbol{s}'_t, a') - Q_{\theta_i}(\boldsymbol{s}_t, a_t)\right)^2 \,\middle|\, \langle \boldsymbol{s}_t, a_t, r_t, \boldsymbol{s}'_t\rangle \sim \mathcal{D}\right].$$

The loss trains an ensemble of k DQN Q-value models `m = make_model(env)`, with independently initialized parameters θ_i, which each take a batch of n states of dimensionality d (as $n \times d$ tensor) and return a batch of vectors of length $|\mathcal{A}| \in \mathbb{N}$ with the predicted Q-values for all actions (as $n \times |\mathcal{A}|$ tensor). The target parameters θ'_i shall be identical to those of the current models, i.e. $\theta'_i = \theta_i, \forall i$, but shall not be changed by gradient descent. Ensure that terminal next states $\boldsymbol{s}'_t$ are handled properly.

```python
import torch
from torch import stack     # to stack a list of tensors in one dim
from torch.nn.functional import mse_loss                # MSE loss

class MyLearner:
    def __init__(self, env, k=5, gamma=0.99):
        self.gamma = gamma
        self.models = [self.make_model(env) for _ in range(k)]
        self.optimizer = torch.optim.Adam([p for m in self.models
                                               for p in m.parameters()])
    def train(self, batch):
        """ Performs one gradient update step on the above loss.
            "batch" is a dictionary of equally sized tensors
            (except for last dimension):
                - batch['states'][t, :] = s_t ∈ R^d
                - batch['actions'][t] = a_t ∈ N
                - batch['rewards'][t] = r_t ∈ R
                - batch['next_states'][t, :] = s'_t ∈ R^d
                - batch['terminals'][t] = true, iff s'_t is terminal """
        loss = 0
        values = stack([m(batch['states']) for m in self.models], dim=0)
        # YOUR CODE HERE
        self.optimizer.zero_grad()
        loss.backward()
        self.optimizer.step()
        return loss.item()
```

Solution

Here are two variants how to implement this loss:

```python
# Variant A: list comprehensions
values = values.gather(dim=-1, index=batch['actions'].unsqueeze(dim=0))
futures = stack([m(batch['next_states']) for m in self.models], dim=0)
futures = futures.min(dim=0)[0].max(dim=-1)[0]
targets = batch['rewards'] + self.gamma * ~batch['terminals'] * futures
loss = mse_loss(values, targets.unsqueeze(dim=0).detach())
```

```
 7
 8 # Variant B: for loops over models
 9 futures = []
10 for m in self.models:
11     futures.append(m(batch['next_states']))
12 futures = stack(futures, dim=0).min(dim=0)[0].max(dim=-1)[0]
13 targets = batch['rewards'] + self.gamma * ~batch['terminals'] * futures
14 for in range(values.shape[0]):
15     v = values[i].gather(dim=-1, index=batch['actions'])
16     loss = loss + mse_loss(v, targets.detach()) / len(self.models)
```

Both variants are fine. The $\max - \min$ order is important, as demonstrated in this example:

$$\mathbf{A} := \left[\begin{smallmatrix} 2 & 3 \\ 4 & 1 \end{smallmatrix}\right], \quad \max_i \min_j A_{ij} = \max(2,1) = 2 \quad \neq \quad 3 = \min(4,3) = \min_j \max_i A_{ij}\,.$$

Rubrik:

- 2 points for gathering the current values of the current actions (only 1 point for a for loop)
- 2 points for computing the future values for all models (none for treating `self.models` as one model)
- 2 points for taking the maximum over actions of future values (only 1 point for forgotten `[0]`, ignore follow up errors for min)
- 2 points for taking the minimum over future values (only 1 point if the max-min order is incorrect)
- 2 points for the correct use of terminals (only 1 point for forgotten *not*: `~`)
- 2 points for detaching the targets
- 2 points for the correct TD loss
- no point deduction for double-Q-learning, or missing loss normalization
- single point deduction for significant syntax errors that would change the behavior, but no deduction for repetitions of that error

End of exam. **Total 100 points.**