Exam CS4400: Deep Reinforcement Learning

19-04-2023 | 13:30–16:30

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 loss computation considered *efficient* for deep learning?

```
loss = ((f(x) - y) ** 2).mean()
```

- ☐ because the `mean()` function automatically masks out non-existing values.
- ☒ because the entire batch is computed in one forward pass.
- ☐ because the mean-squared error minimizes the difference between `f(x)` and `y`.
- ☐ because all operations are differentiable and can therefore be used in gradient descent.

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

- ☒ `hidden_size`
- ☐ `output_size`
- ☐ `num_heads`
- ☐ `transposed`

1.3: What is the input to a *Multi-headed Attention Layer* (MHA)?

- ☐ a tensor one vector per sample
- ☐ a tensor with an oredered set of vectors per sample
- ☒ a tensor with an 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 **not** been discussed in the lecture?

- ☒ $-\mathbb{E}[Q(s,a) \mid (s,a) \sim \mathcal{D}]$
- ☐ $-\mathbb{E}[\ln \pi(a|s) \mid (s,a) \sim \mathcal{D}]$
- ☐ $-\mathbb{E}[Q(s,a)\ \ln \pi(a|s) \mid (s,a) \sim \mathcal{D}]$
- ☐ $-\mathbb{E}[Q(s,\pi(s,a)) \mid s \sim \mathcal{D}, a \sim \mathcal{N}(0,1)]$

1.5: Which of the following approaches update the *target network* the **slowest**?

- ☐ Semi-gradient TD-learning
- ☐ Soft target updates
- ☐ Deep Q-networks
- ☒ Neural Fitted Q-iteration

1.6: Which *property* of RL does **not** violate classical ML assumptions?

- ☐ regression targets are non-stationary
- ☒ states have to fulfill the Markov assumption
- ☐ neural networks forget values of unsampled states
- ☐ transitions within episodes are not sampled i.i.d.

1.7: Changing which parameter makes the DQN algorithm **not** more *sample efficient*?

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

1.8: Which class is part of the standard RL *software architecture* from the lectures?

- ☐ Trainer
- ☐ Executor
- ☐ Explorer
- ☒ Controller

1.9: Which of the following definitions is a TD(λ) *value target*?

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

1.10: Which term from the *on-policy-gradient loss* $\mathcal{L}_\pi$ did we **not** ignore or approximate to derive the *off-policy actor-critic loss* $\mathcal{L}_\mu$ in the lecture?

$$\mathcal{L}_\pi[\theta] \quad := \quad -\sum_{t=0}^{n-1}\iint \xi_t^\pi(s_t)\,\gamma^t\,\big(Q^\pi(s_t,a_t) - V^\pi(s_t)\big)\,\nabla_\theta\pi_\theta(a_t|s_t)\,ds_t\,da_t$$

- ☐ $\frac{\xi_t^\pi(s_t)}{\xi_t^\mu(s_t)}$
- ☒ $\frac{\pi_\theta(a_t|s_t)}{\mu(a_t|s_t)}$
- ☐ $V^\pi(s_t)$
- ☐ $Q^\pi(s_t,a_t)$

1.11: Which algorithm can **not** work with *discrete actions* without the reparametrization trick?

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

1.12: How does SAC *differ* from TD3?

- ☐ SAC can use an experience replay buffer, TD3 can not
- ☒ SAC uses stochastic policies via the reparametrization trick, TD3 does not
- ☐ SAC trains two Q-value functions to be pessimistic, TD3 does not
- ☐ SAC regularizes actions with clipped noise, TD3 does not

1.13: Which of the following uncertainties is *irreducible*?

- ☒ Aleatoric
- ☐ Epistemic
- ☐ Posterior variance
- ☐ Upper confidence bounds

1.14: What is the *value of the initial state* of a Markov chain with 10 transitions 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 transitions with a reward of 1 each, summed up $\sum_{t=0}^{9} \gamma^t = \frac{1-\gamma^{10}}{1-\gamma}$.

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

- ☐ non-stationarity
- ☒ catastrophic forgetting
- ☒ no exploration
- ☒ learning stability

Remark: Many students gave the answer "learning stability" amd "catastrophic forgetting", which is less true, but one can interpret it as correct.

1.16: In offline RL, *restricting available actions* performs 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 **outputs** does a LSTM network with h memory cells require to represent a *stochastic Gaussian policy* in a *partially observable* environment with continuous actions $\boldsymbol{a} \in \mathbb{R}^k$?

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

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

- ☐ games with sequential moves
- ☐ games with continuous actions
- ☒ games with non-stationary rules
- ☐ games with stochastic moves

1.19: Which of the following MARL algorithms can overcome *relative overgeneralization*?

- ☐ VDN
- ☐ MADDPG
- ☒ DCG
- ☐ QMIX

1.20: In which of the following *multi-task* techniques does the policy have access to the *task-id*?

- ☐ DQN
- ☒ HER
- ☐ DR
- ☐ Sim2Real

Question 2: (6 points)

In 6 sentences or less, explain *active domain randomization* and name one approach to do it.

Solution

In domain randomization, random environmental parameters are drawn every episode. Active domain randomization tries to choose the parameters that are most informative for the learning process. An

example Bayesian optimization, which learns the Bayesian posterior over the task space and samples parameters with high probability. Another example is adversarial interference, where another agent is trained, like in robust RL, to choose the task/parameters that are hardest for the current agent.

Rubrik:

- 2 points for mentioning randomly drawn environments/tasks/parameters. Only 1 point if just noise is added to the state.
- 2 point for mentioning that one wants to choose the most informative. Only 1 point if standard domain randomization is described.
- 2 points for either Bayesian optimization or adversarial interference, even if they are called slightly differently.

Question 3: **(6 points)**

(a) *[3 points]* Design a *cooperative* two-payer normal-form game, that exhibits on average *relative overgeneralization*, by defining the collaborative reward:

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

$\setminus$	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 games exhibiting relative overgeneralization 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 predator-prey:

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

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

The same game with a punishment below -1 is an example where both players *almost* exhibit RO.

Once can solve relative overgeneralization (RO) with *optimistic return methods*. Here agents remember the best past reward for each action or ignore rewards that are lower than the current Q-value. As a result, each agent always remembers the *best* action of the other agent and RO cannot occur any more.

Another methods are *higher order factorization* like Deep Coordination Graphs. Here the joint q-value function can be learned by a pairwise function, which does not exhibit RO. The agents can agree decentralized on the best joint act by using the max-sum algorithm with message passing.

Rubrik:

- 2 point for a game where at least one agent exhibits RO, or where both agents *almost* exhibit RO

- 1 point for a game where both agents exhibit RO, including *on* the boundary (-1 above)
- 1 point for mentioning *either* optimistic return or higher-order factorization, or any method that works against RO in the example.
- 2 point for a good explanation of the mentioned concept

Question 4: (6 points)

Which deep reinforcement learning *algorithm* **and** *neural network architecture* from the lectures would you use to train two robots, with cameras as eyes, playing *ping-pong* against each other? Give at least one argument to justify **each** of your choices.

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

Solution

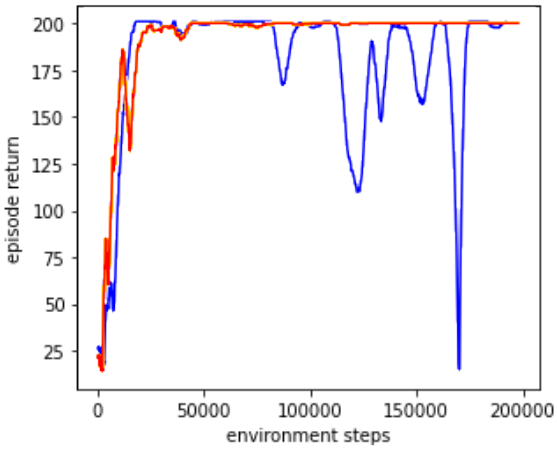
The environment is partially observable (camera eyes), the observations are images (require CNNs), and the actions are continuous (motors). The two robots play aginst each other (zero-sum MARL game) and can/will therefore not communicate.

Rubrik:

- 3 points for a MARL algorithms for continuous actions: MADDPG or independent learning with any single-agent algorithm for continuous actions (TRPO, PPO, DDPG, TD3, SAC). Only 2 point for MARL algorithms for discrete actions (IQL, COMA, VDN, QMIX, QTRAN) or for not mentioning MARL. 1 point for DCG (no communication) or single-agent DQN.
- 3 points for CNN (for the images), 2 points for LSTM (against partial observability). No points for Linear, GNN, MHA or others, unless justified (e.g. with slot attention).
- 1 point less if the choice is suboptimal or no justification (minimally 0 points per category).
- No points if the choice is totally wrong or the justification unconnected.

Question 5: (6 points)

The figure to the right plots the training return for Online Q-learning with a *replay buffer* of the last $100k$ transitions (blue, dark) and the same algorithm with soft-updated *target networks* (red, bright), learning the `CartPole-v1` task.



(a) *[2 points]* Explain why the blue agent becomes instable in the second half of training.

(b) *[2 points]* Explain how the target network stabilizes training.

(c) *[2 points]* Explain the difference between *soft* and *hard* target updates.

Solution

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

(a) In the second half of training, the initial exploration transitions leave the replay buffer and catastrophic forgetting leads to uncontrollable drifting of unexectured actions. Once the wrong action is selected greedily, it takes a while to unlearn it again. Random deviations, for example due to the ϵ-greedy policy, can also lead to states that have been affected by catastrophic forgetting.

(b) Soft-updated target networks slow the change in the network that is used for bootstrapping. The above problems are therefore kept *local* and propagations are *delayed* long enough for the sampling policy is not affected (that much).

(c) Soft target updates change the target parameters every gradient update *slightly* towards the current network parameters, whereas hard software updates do not change the target network most of the time but every now and then replace the target parameters completely with the the current parameters.

Rubrik:

- 2 points for a coherent explanation of (a).
- no points for explanations that would also apply to target networks.
- 2 points for a coherent explanation of (b).
- no points for explanantions that would also apply without target networks.
- 2 points for a coherent explanation of (c).
- only 1 point in either cathegory if the explanation is not coherent or the answer otherwise unclear.

Question 6: **(6 points)**

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

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

and the policies of agents i: $\pi^i_{\theta_i}(a) = \begin{cases} \theta_i & \text{if } a = X \\ 1-\theta_i & \text{if } a = Y \end{cases}$, $\theta_i \in [0, 1]$.

(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) *[4 points]* Derive the *mixed Nash equilibrium* for $\mathcal{G}$.

Hint: it suffices to derive the exteme point, you do not have to prove that it is a saddle-point.

Solution

(a) None exist. **Rubrik:**

- 2 points for the correct answer.

(b) The mixed Nash equilibrium of a zero-sum game is the tuple of stochastic policies (π^A, π^B) that maximize the expected return Q w.r.t. π^A and minimizes Q w.r.t. π^B. Let in the following $\pi' := \pi^A_{\theta_A}$

and $\pi := \pi^B_{\theta_B}$:

$$\begin{aligned} Q &:= \mathbb{E}[r(a,b)|a \sim \pi', b \sim \pi] \\ &= \pi'(X)\pi(X)r(X,X) + \pi'(X)\pi(Y)r(X,Y) + \pi'(Y)\pi(X)r(Y,X) + \pi'(Y)\pi(Y)r(Y,Y) \\ &= 4\theta_A\theta_B - 4\theta_A(1-\theta_B) + 2(1-\theta_A)(1-\theta_B) \\ &= 4\theta_A\theta_B + 4\theta_A\theta_B - 4\theta_A + 2\theta_A\theta_B - 2\theta_A - 2\theta_B + 2 \\ &= 10\,\theta_A\theta_B - 6\theta_A - 2\theta_B + 2\,. \end{aligned}$$

Setting the derivatives to zero reveals the saddle point of Q in parameter space (θ_A, θ_B):

$$\begin{aligned} \tfrac{\partial Q}{\partial \theta_B} &= 10\,\theta_A - 2 \stackrel{!}{=} 0 \quad \Rightarrow \quad \theta_A = \tfrac{2}{10}\,, \\ \tfrac{\partial Q}{\partial \theta_A} &= 10\,\theta_B - 6 \stackrel{!}{=} 0 \quad \Rightarrow \quad \theta_B = \tfrac{6}{10}\,. \end{aligned}$$

The mixed Nash equilibrium is therefore $\boldsymbol{\theta} = [\frac{2}{10}, \frac{6}{10}]$, or a pair of stochastic policies in which agent A plays X 20% of the time and agent B plays X 60% of the time.

Rubrik:

- 1 point for the correct ansatz of Q
- 1 point for the correct Q
- 1 point for setting the derivatives to zero
- 1 point for the correct policy parameters

Question 7: **(6 points)**

To implement *pseudo-counts*, you are given a generative model that would produce a given state s with probability $p(s)$ *before* and $p'(s)$ *after* updating the generative model with an observed s.

(a) *[2 points]* Explain how $p(s)$ and $p'(s)$ are related to the count $N(s)$ of past observations of s.

(b) *[2 points]* Derive the *pseudo-count* $N(s) = \frac{p(s)\,(1-p'(s))}{p'(s)-p(s)}$.

(c) *[2 points]* Define a sensible intrinsic reward $r_i(s, a, s')$ for *deep exploration* based on pseudo-count $N(s)$.

Solution

(a) For countable states, $p(s) = \frac{N(s)}{n}$ and $p'(s) = \frac{N(s)+1}{n+1}$, where $n := \sum_s N(s)$ denotes the total number of observations before observing s again.

Rubrik:

- 1 point per correct definition of $p(s)$ and $p(s')$, or an equivalent description (without math)
- 1 point if $p(s)$ and $p'(s)$ are not defined but the basic idea of the relationship is communicated

(b) From above we can deduce $n = \frac{N(s)}{p(s)}$ and $n = \frac{N(s)+1}{p'(s)} - 1$. Putting those equalities together yields

$$\tfrac{N(s)}{p(s)} = \tfrac{N(s)}{p'(s)} + \tfrac{1}{p'(s)} - 1 \quad \Leftrightarrow \quad N(s)\big(\tfrac{1}{p(s)} - \tfrac{1}{p'(s)}\big) = \tfrac{1-p'(s)}{p'(s)} \quad \Leftrightarrow \quad N(s)\tfrac{p'(s)-p(s)}{p(s)\,p'(s)} = \tfrac{1-p'(s)}{p'(s)}$$

$$\Leftrightarrow \quad N(s) = \frac{p(s)\,p'(s)\,(1-p'(s))}{p'(s)(p'(s)-p(s))} = \frac{p(s)\,(1-p'(s))}{p'(s)-p(s)}\,.$$

Rubrik:

- 1 point for an ansatz that removes the unknown n
- 1 point for the correct derivation of pseudo counts $N(s)$

(c) The reward should create an upper confidence bound on the value, i.e., should be an analogue to the standard deviation, which for averages is proportionally to $1/\sqrt{N(s)}$. As we only measure $N(s)$, and not $N(s,a)$, we can only see how often we visited the *next state s'*. The intrinsic reward for the transition $s, a \to s'$ is therefore (for some factor c):

$$r_i(s,a,s') \quad := \quad \frac{c}{\sqrt{N(s')}}\,.$$

Rubrik:

- 1 point for using $1/\sqrt{N}$, even if $\sqrt{}$ is missing, but no point if fraction is missing
- 1 point for using $N(s')$ or redefining the generative model to deduce $N(s,a)$.

Question 8: **(10 points)**

Prove that the variance $\mathbb{V}[Q^\pi(s,a)]$ of the Q-value $Q^\pi(s,a)$ of stochastic policy π, in a MDP with deterministic rewards and stochastic transitions, is bounded from above by

$$\mathbb{V}[Q^\pi(s,a)] \quad \leq \quad \gamma^2\,\mathbb{E}\Big[\mathbb{V}[Q^\pi(s',a')]\Big|\substack{s'\sim P(\cdot|s,a)\\ a'\sim\pi(\cdot|s')}\Big]\,, \qquad \forall s\in\mathcal{S}, \quad \forall a\in\mathcal{A}\,.$$

Hint: you can use Jensen's inequality $\mathbb{E}[x]^2 \leq \mathbb{E}[x^2]$. You may also abbreviate $\mathbb{E}'[\cdot] := \mathbb{E}\Big[\cdot\Big|\substack{s'\sim P(\cdot|s,a)\\ a'\sim\pi(\cdot|s')}\Big]$.

Solution

Note that the reward function $r(s,a)$ is deterministic, and thus $\mathbb{E}\big[r(s,a)\big] = r(s,a)$.

$$\begin{aligned}
\mathbb{V}[Q^\pi(s,a)] &= \mathbb{E}\Big[\Big(\overbrace{r(s,a)+\gamma\mathbb{E}'[Q^\pi(s',a')]}^{Q^\pi(s,a)} - \overbrace{\mathbb{E}\big[r(s,a)+\gamma\mathbb{E}'[Q^\pi(s',a')]\big]}^{\mathbb{E}[Q^\pi(s,a)]}\Big)^2\Big]\\
\text{(det. reward)} &= \mathbb{E}\Big[\Big(\underbrace{r(s,a)-\mathbb{E}\big[r(s,a)\big]}_{0}+\gamma\mathbb{E}'[Q^\pi(s',a')] - \gamma\mathbb{E}\big[\mathbb{E}'[Q^\pi(s',a')]\big]\Big)^2\Big]\\
\text{(swap expectations)} &= \mathbb{E}\Big[\Big(\gamma\mathbb{E}'\big[Q^\pi(s',a') - \mathbb{E}[Q^\pi(s',a')]\big]\Big)^2\Big]\\
\text{(Jensen's inequality)} &\leq \gamma^2\,\mathbb{E}\Big[\mathbb{E}'\big[\big(Q^\pi(s'a') - \mathbb{E}[Q^\pi(s',a')]\big)^2\big]\Big]\\
\text{(swap expectations)} &= \gamma^2\,\mathbb{E}'\Big[\underbrace{\mathbb{E}\big[\big(Q^\pi(s'a') - \mathbb{E}[Q^\pi(s',a')]\big)^2\big]}_{\mathbb{V}[Q^\pi(s',a')]}\Big] \quad = \quad \gamma^2\,\mathbb{E}'\big[\mathbb{V}[Q^\pi(s',a')]\big]\,.
\end{aligned}$$

Rubrik:

- 1 point for the correct definition of the Q-value

- 2 point for the correct definition of the variance
- 2 point for canceling out the reward functions
- 2 points for using Jensen's inequality correctly
- 2 point for resubstituting the variance of the next state-action
- 1 point for putting it all correctly together

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 *DQN with intrinsic reward* in the given `MyLearner` class **efficiently**:

$$\min_{\{\theta_i\}_{i=1}^k} \mathbb{E}\Big[\tfrac{1}{k}\sum_{i=1}^{k}\tfrac{1}{n}\sum_{t=1}^{n}\Big(r_t + \sigma(s_t,a_t) + \gamma \max_{a'\in\mathcal{A}} \mu(s'_t,a') - Q_{\theta_i}(s_t,a_t)\Big)^2 \,\Big|\, \langle s_t,a_t,r_t,s'_t\rangle \sim \mathcal{D}\Big],$$

where $\sigma(s,a) := \sqrt{\tfrac{1}{k}\sum_{j=1}^{k}\Big(Q_{\theta'_j}(s,a) - \mu(s,a)\Big)^2}$, and $\mu(s,a) := \tfrac{1}{k}\sum_{j=1}^{k} Q_{\theta'_j}(s,a)$.

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.

```
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
```

```
21          values = stack([m(batch['states']) for m in self.models], dim=0)
22          # YOUR CODE HERE
23          self.optimizer.zero_grad()
24          loss.backward()
25          self.optimizer.step()
26          return loss.item()
```

Hint: you can use the functions `Tensor.mean(dim)` to compute μ and `Tensor.std(dim)` for σ. Be careful about the order in which `.mean()` and `.max()` are applied!

Solution

Here are two variants how to implement this loss:

```
 1 # Variant A: list comprehensions
 2 values = values.gather(dim=-1, index=batch['actions'].unsqueeze(dim=0))
 3 futures = stack([m(batch['next_states']) for m in self.models], dim=0)
 4 futures = futures.mean(dim=0).max(dim=-1)[0]
 5 targets = batch['rewards'] + values.std(dim=0) \
 6           + self.gamma * ~batch['terminals'] * futures
 7 loss = mse_loss(values, targets.unsqueeze(dim=0).detach())
 8
 9 # Variant B: for loops over models
10 futures = []
11 for m in self.models:
12     futures.append(m(batch['next_states']))
13 futures = stack(futures, dim=0).mean(dim=0).max(dim=-1)
14 targets = batch['rewards'] + values.std(dim=0) \
15           + self.gamma * ~batch['terminals'] * futures
16 for i in range(values.shape[0]):
17     v = values[i].gather(dim=-1,index=batch['actions'])
18     loss = loss + mse_loss(v, targets.detach()) / values.shape[90]
```

Both variants are fine. The $\max - \text{mean}$ order is important, due to Jensen's inequality.

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]`)
- 2 points for taking the mean over future values (only 1 point if the mean-max 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, including value (only 1 point for the mean current future values)
- no point deduction for double-Q-learning or for 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.**