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 12 (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?

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

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*?

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

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?

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

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

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

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.

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:

$$r(a^1, a^2) \quad := \quad \begin{array}{|c|c|c|c|} \hline \backslash & a^2_1 & a^2_2 & a^2_3 \\ \hline a^1_1 & & & \\ \hline a^1_2 & & & \\ \hline a^1_3 & & & \\ \hline \end{array}$$

(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.

Question 4:

(6 points)

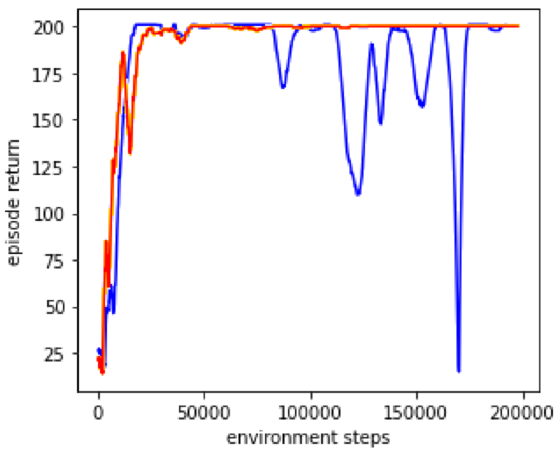
Which deep reinforcement learning *algorithm* **and** *neural network architecture* from the lectures would you use to train two robots, with cameras as eyes, playing *ping-pong* against each other? Give at least one argument to justify **each** of your choices.

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.

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.

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

(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.

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 $N(s)$.

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 $\mathbb{E}'[\cdot] := \mathbb{E}\Big[\cdot\Big|_{\substack{s'\sim P(\cdot|s,a) \\ a'\sim\pi(\cdot|s')}}\Big]$.

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**:

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

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

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 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
        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()
```

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!

You can use the next page to write down your answer as well!

Question 9 (continuation):

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