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 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 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?

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

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

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?

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

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

- ☐ $-\sum\limits_{t=0}^{n-1} \gamma^t\, \mathbb{E}_\mu\Big[\min\Big(A_t \frac{\mu(a_t|s_t)}{\pi_\theta(a_t|s_t)}, A_t \operatorname{clip}(\frac{\mu(a_t|s_t)}{\pi_\theta(a_t|s_t)}, 1-\epsilon, 1+\epsilon)\Big)\Big]$
- ☐ $-\sum\limits_{t=0}^{n-1} \gamma^t\, \mathbb{E}_\mu\Big[\max\Big(A_t \frac{\mu(a_t|s_t)}{\pi_\theta(a_t|s_t)}, A_t \operatorname{clip}(\frac{\mu(a_t|s_t)}{\pi_\theta(a_t|s_t)}, 1-\epsilon, 1+\epsilon)\Big)\Big]$
- ☐ $-\sum\limits_{t=0}^{n-1} \gamma^t\, \mathbb{E}_\mu\Big[\min\Big(A_t \frac{\pi_\theta(a_t|s_t)}{\mu(a_t|s_t)}, A_t \operatorname{clip}(\frac{\pi_\theta(a_t|s_t)}{\mu(a_t|s_t)}, 1-\epsilon, 1+\epsilon)\Big)\Big]$
- ☐ $-\sum\limits_{t=0}^{n-1} \gamma^t\, \mathbb{E}_\mu\Big[\max\Big(A_t \frac{\pi_\theta(a_t|s_t)}{\mu(a_t|s_t)}, A_t \operatorname{clip}(\frac{\pi_\theta(a_t|s_t)}{\mu(a_t|s_t)}, 1-\epsilon, 1+\epsilon)\Big)\Big]$

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?

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

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.

Question 3: (6 points)

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

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

$\backslash$	a^2_1	a^2_2	a^2_3
a^1_1			
a^1_2			
a^1_3			

(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.

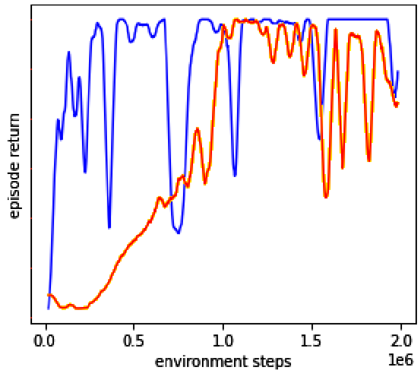
Question 4: **(6 points)**

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.

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.

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*.

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.

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 $D_{\text{KL}}[\boldsymbol{\pi} \| \boldsymbol{\mu}] = \boldsymbol{\pi}^\top \ln(\frac{\boldsymbol{\pi}}{\boldsymbol{\mu}})$:

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

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.

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**:

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

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()
```

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

Question 9 (continuation):

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