Exam CS4400: Deep Reinforcement Learning

practice early! | anytime

Student name: ____

Student number: ____________________

- This is a closed-book individual examination with **9 questions** and a total of **42 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 **150 minutes** after the official start.
- This gives you roughly 3 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 8 (excluding this front page).
- Exam prepared by Wendelin Böhmer. ©2022 TU Delft.

Question 1 (multiple choice): (10 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 1 point per correct answer, except if multiple squares are marked. Wrong answers yield no points, but are also not punished. Good luck!

1.1: Recurrent neural networks *without* gates can **not** have:

□ exploding gradients
□ vanishing gradients
☒ memory cells
□ non-linear transfer-functions

1.2: Which of the following is **not** a normalization layer module in PyTorch?

□ `torch.nn.LayerNorm`
☒ `torch.nn.TemporalNorm`
□ `torch.nn.InstanceNorm2d`
□ `torch.nn.BatchNorm1d`

1.3: We use double Q-learning to counter which problem for bootstrapping?

□ $\mathbb{E}[\max_a Q(s,a)] \leq \max_a \mathbb{E}[Q(s,a)]$
☒ $\mathbb{E}[\max_a Q(s,a)] \geq \max_a \mathbb{E}[Q(s,a)]$
□ $\mathbb{E}[Q(s, \arg\max_a Q'(s,a))] \leq \max_a \mathbb{E}[Q(s,a)]$
□ $\mathbb{E}[Q(s, \arg\max_a Q'(s,a))] \geq \max_a \mathbb{E}[Q(s,a)]$

1.4: In which of the following cases is a *dueling network architecture* useful?

□ When the value of two actions are too similar.
□ When maximization introduces overestimation bias.
□ When a pessimistic value estimate is needed.
☒ When many actions have the same successor state.

1.5: What is the correct normalization factor x for $Q(\lambda)$ targets $Q^*_\lambda(s_t,a_t) := x\sum_{n=0}^{m} \lambda^n Q^*_{n+1}(s_t,a_t)$?

□ $x = 1 - \lambda^{m+1}$
□ $x = \frac{1}{1-\lambda^{m+1}}$
□ $x = \frac{1-\lambda^{m+1}}{1-\lambda}$
☒ $x = \frac{1-\lambda}{1-\lambda^{m+1}}$

1.6: Which uncertainty measure can **not** be used as an estimate of the value function's variance?

☒ random network distillation
☐ pseudo-counts
☐ random hash functions
☐ ensembles

1.7: Which of the following statements about deep exploration is correct?

☐ deep exploration converges faster to the optimal policy than ϵ-greedy
☒ deep exploration requires to propagate future uncertainty
☐ deep exploration requires to count how often a specific state has been seen
☐ deep exploration requires deep neural networks

1.8: Which value learning method is the most *stable*?

☐ independent learning
☒ on-policy learning
☐ off-policy learning
☐ offline learning

1.9: Which algorithm uses methods from offline RL to stabilize learning?

☐ DQN
☐ REINFORCE
☐ TRPO
☒ SAC

1.10: Which MARL algorithm does **not** use centralized training?

☒ IQL
☐ MADDPG
☐ COMA
☐ QMIX

Question 2: **(2 points)**

Describe in 4 sentences or less **two** examples where a value function helps a policy gradient algorithm. The examples must come from *different* algorithms.

Solution

1 Point for any of the following (up to 2 points). **Rubrik:**

- Actor-critic/Off-PAC/TRPO/PPO: reduce variance with bias

- Actor-critic/Off-PAC/TRPO/PPO: replace rollouts with bootstrapping
- DDPG/TD3/SAC: optimize policy for estimated Q-value

Question 3: (2 points)

In 4 senteces or less, name and explain **two** examples where *robust reinforcement learning* is useful.

Solution

Any example that contains one of the following points (or any other that makes sense). **Rubrik:**

- Stay away from dangerous states (e.g. not bumping into obstacles)
- Avoid states in which small changes can destabilize learning
- Become robust against environmental disturbances like wind

Question 4: (2 points)

Explain in 4 sentences or less the difference between *Thompson sampling* and *optimistic exploration*.

Solution

The difference are in *what* is explored (parameter or action space) and in *how* it is explored (sample model or overestimate action). **Rubrik:**

- 1 point for *what* is explored: Thompson sampling explores the parameter space of models, optimistic exploration the action space in a state.
- 1 point for *how* it is explored: Thompson sampling chooses the action with the largest Q-value of a randomly drawn model, optimistic exploration overestimates action values before choosing the maximum.

Question 5: (3 points)

Give the joint actions of all Nash-equilibria for a two-player single-state general-sum game with the following reward matrix, where entry x/y denotes the reward x for player 1 and y for player 2:

P1/P2	a_1^2	a_2^2	a_3^2	a_4^2
a_1^1	-2/1	4/-2	2/3	0/-1
a_2^1	-1/5	3/1	0/4	-3/2
a_3^1	1/2	2/-2	-1/1	2/1
a_4^1	0/2	-3/3	1/4	5/1

Which Nash equilibrium would player 1 prefer? Which would be better for player 2?

Solution

Rubrik:

- 1 point for $a_3^1, a_1^2 : 1/2$
- 1 point for $a_1^1, a_3^2 : 2/3$
- -1 point for any wrong Nash equilibrium (minimum 0 points)
- 1 point for: both players prefer $2/3$
- no point if either player picks the wrong (or ambiguous) equilibrium

Question 6: (4 points)

Let $v := \sum_{t=0}^{\infty} \gamma^t r_t$ denote the value in a MDP with a single state and action, where the reward $r_t \sim \mathcal{N}(\mu, \sigma^2)$ is drawn i.i.d. from a normal distribution and $\gamma \in (0, 1)$ denotes the discount factor. *Without* using the fact the variance of a sum of independent variables is the sum of the variables' variances, prove analytically that the variance of v is

$$\mathbb{V}[v] \quad = \quad \frac{\sigma^2}{1-\gamma^2} \,.$$

Solution

The major insight here is that $\mathbb{E}[r_t] = \mu$, that $\mathbb{E}[(r_i - \mu)(r_j - \mu)] = (\mathbb{E}[r_i] - \mu)(\mathbb{E}[r_j] - \mu) = 0, \forall i \neq j$, and that $\mathbb{E}[(r_i - \mu)^2] = \sigma^2$. We also need the geometric series $\sum_{t=0}^{\infty} \gamma^t = \frac{1}{1-\gamma}$.

$$\begin{aligned}
\mathbb{E}[v] &= \sum_{t=0}^{\infty} \gamma^t \mathbb{E}[r_t] &&= \sum_{t=0}^{\infty} \gamma^t \mu \\
\mathbb{V}[v] &= \mathbb{E}\Big[\Big(\sum_{t=0}^{\infty} \gamma^t r_t - \mathbb{E}[v]\Big)^2\Big] &&= \mathbb{E}\Big[\Big(\sum_{t=0}^{\infty} \gamma^t (r_t - \mu)\Big)^2\Big] \\
&= \sum_{i=0}^{\infty}\sum_{j=0}^{\infty} \gamma^{i+j} \underbrace{\mathbb{E}\big[(r_i - \mu)(r_j - \mu)\big]}_{\sigma^2\,\delta(i=j)} &&= \sigma^2 \sum_{i=0}^{\infty} (\gamma^2)^i \quad = \quad \frac{\sigma}{1-\gamma^2}
\end{aligned}$$

Rubrik:

- 1 point for the correct definition of variance $\mathbb{V}$
- 1 point for the use of independent samples
- 1 point for the use of the definition of σ^2
- 1 point for putting it correctly together

Question 7: **(5 points)**

Derive the vanilla actor-critic policy gradient $\nabla_\theta \mathcal{L}_\pi[\theta]$ of policy π_θ from the off-policy gradient $\nabla_\theta \mathcal{L}_\mu[\theta]$:

$$\nabla_\theta \mathcal{L}_\mu[\theta] \;:=\; -\nabla_\theta \mathbb{E}_\mu\Big[\sum_{t=0}^{n-1}\gamma^t A_t^\pi \tfrac{\pi_\theta(a_t|s_t)}{\mu(a_t|s_t)}\Big] \;=\; \underbrace{\ldots}_{\text{derivation}} \;\approx\; -\mathbb{E}_{\pi_\theta}\Big[\sum_{t=0}^{n-1}\gamma^t \, A_t^\pi \, \nabla_\theta \ln \pi_\theta(a_t|s_t)\Big] \;=:\; \nabla_\theta \mathcal{L}_\pi[\theta]\,,$$

where $A_t^\pi := r_t + \gamma V^\pi(s_{t+1}) - V^\pi(s_t)$ is the advantage at time t, based on the state-value function $V^\pi(s)$ that does not depend on θ. Which approximation do you need to make? Make sure every step can be followed easily!

Solution

We have to make the approximation $\frac{\xi_t^\pi(s_t)}{\xi_t^\mu(s_t)} \overset{(1)}{\approx} 1$. We also use $\mathbb{E}_\mu[\frac{\xi_t^{\pi_\theta}(s_t)}{\xi_t^\mu(s_t)} \frac{\pi_\theta(a_t|s_t)}{\mu(a_t|s_t)} A_t^\pi] \overset{(2)}{=} \mathbb{E}_{\pi_\theta}[A_t^\pi]$. Lastly, we use the log-trick: $\nabla_\theta \pi_\theta(a_t|s_t) \overset{(3)}{=} \pi_\theta(a_t|s_t) \nabla_\theta \ln \pi_\theta(a_t|s_t)\,.$

$$\begin{aligned}
\nabla_\theta \mathcal{L}_\mu[\theta] \;&:=\; -\nabla_\theta \mathbb{E}_\mu\Big[\sum_{t=0}^{n-1}\gamma^t A_t^\pi \tfrac{\pi_\theta(a_t|s_t)}{\mu(a_t|s_t)}\Big] \\
&\overset{(1)}{\approx}\; -\sum_{t=0}^{n-1}\gamma^t \nabla_\theta \mathbb{E}_\mu\Big[A_t^\pi \tfrac{\xi_t^\pi(s_t)}{\xi_t^\mu(s_t)} \tfrac{\pi_\theta(a_t|s_t)}{\mu(a_t|s_t)}\Big] \\
&\overset{(2)}{=}\; -\nabla_\theta \sum_{t=0}^{n-1}\gamma^t \mathbb{E}_{\pi_\theta}[A_t^\pi] \\
&=\; -\sum_{t=0}^{n-1}\gamma^t \iiint \xi_t^\pi(s_t)\, \nabla_\theta \pi(a_t|s_t)\, P(s_{t+1}|s_t,a_t)\, A_t^\pi \, ds_t\, da_t\, ds_{t+1} \\
&\overset{(3)}{=}\; -\sum_{t=0}^{n-1}\gamma^t \iiint \xi_t^\pi(s_t)\, \pi_\theta(a_t|s_t)\, P(s_{t+1}|s_t,a_t)\, A_t^\pi \, \nabla_\theta \ln \pi_\theta(a_t|s_t)\, ds_t\, da_t\, ds_{t+1} \\
&=\; -\mathbb{E}_{\pi_\theta}\Big[\sum_{t=0}^{n-1}\gamma^t \, A_t^\pi \, \nabla_\theta \ln \pi_\theta(a_t|s_t)\Big] \quad =: \quad \nabla_\theta \mathcal{L}_\pi[\theta]
\end{aligned}$$

Rubrik:

- 1 point for the correct approximation (1)
- 2 point for some form of (2)
- 1 point for the log-trick (3)
- 1 point for making it all work together
- no point deduction for shortcuts, unless they become excessive
- e.g. "jumping" over (1) and (2) without explanation, but then using (3) do derive the right gradient would be worth 2 points if everything else is correct.

Question 8: (programming) **(6 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 DDPG policy objective efficiently in the given `MyLearner` class:

$$\max_{\theta} \quad \mathbb{E}\Big[\frac{1}{m}\sum_{i=1}^{m}\frac{1}{n_i}\sum_{t=0}^{n_i-1}\gamma^t Q_\phi\big(s^i_t, \pi_\theta(s^i_t)\big)\,\Big|\,\tau^i_{n_i}\!\sim\!\mathcal{D}\Big],$$

where $\tau^i_{n_i} := \{s^i_t, a^i_t\}_{t=0}^{n_i-1}$ is a history of states $s^i_t \in \mathbb{R}^d$ and actions $a^i_t \in \mathbb{R}^b$ up to time $n_i - 1$.

Hint: The Q-value module `value` (computes the Q-values Q_ϕ for a minibatch) is given and takes the concatenation of a state- and an action-tensor of equal size (except for the last dimension) as input. The policy module `policy` (computes the policy π_θ for a minibatch) is given, takes a state-tensor as input and returns an action tensor of the same size (except for the last dimension).

```
import torch

class MyLearner:
    def __init__(self, value, policy, gamma=0.99):
        self.value_model = value
        self.policy_model = policy
        self.gamma = gamma
        self.optimizer = torch.optim.Adam(policy.parameters())

    def train(self, batch):
        """ Performs one gradient update step on the loss defined above.
            "batch" is a dictionary of equally sized tensors
            (except for last dimension):
                - batch['states'][i, t, :] = $s^i_t$
                - batch['actions'][i, t, :] = $a^i_t$
                - batch['mask'][i, t] = $t < n_i$   """
        loss = 0
        # YOUR CODE HERE
        self.optimizer.zero_grad()
        loss.backward()
        self.optimizer.step()
        return loss.item()
```

Solution

```
pol = self.policy_model(batch['states'])
val = self.value_model(torch.cat([batch['states'], pol], dim=-1))
gam = torch.zeros(1, val.shape[1])  # broadcast dim=0
for t in range(gam.shape[1]):       # for loops over inputs are efficient
    gam[0, t] = self.gamma ** t     # gamma^t for each t
loss = -(gam * val * batch['mask']).sum() / batch['mask'].sum()
```

Rubrik:

- 1 point for computing the policy actions correctly
- 1 point for computing the value correctly (must be `cat` not `stack`)
- 1 point for attempting to compute γ^t

- 1 point for computing γ^t correctly (broadcasting takes care of the first dimension)
- 1 point for an efficient loss (no point for loops), including the minus sign
- 1 point for correct masking with `batch['mask']` and normalization

Question 9: **(8 points)**

A given linear recurrent neural network takes a time series of n input vectors $\boldsymbol{x}_t \in \mathbb{R}^d$ and outputs a time-series of n vectors $\boldsymbol{y}_t \in \mathbb{R}^b$. The RNN computes hidden outputs $\boldsymbol{h}_t \in \mathbb{R}^q$ without non-linearities:

$$\boldsymbol{h}_t := \mathbf{W}\boldsymbol{h}_{t-1} + \mathbf{U}\boldsymbol{x}_t \in \mathbb{R}^q \,, \qquad \boldsymbol{y}_t := \mathbf{V}\boldsymbol{h}_t \in \mathbb{R}^b \,, \qquad 1 \leq t \leq n \,, \qquad \boldsymbol{h}_0 := \mathbf{0} \,.$$

(a) *[4 points]* Prove by induction that the above update equations are equivalent to the multivariate function $g : \{\mathbb{R}^d\}_{t=1}^n \times \mathbb{R}^{q\times q} \times \mathbb{R}^{q\times d} \times \mathbb{R}^{b\times q} \to \mathbb{R}^{b\times n}$:

$$g(\{\boldsymbol{x}_t\}_{t=1}^n, \mathbf{W}, \mathbf{U}, \mathbf{V})_{k,m} := \sum_{t=1}^{M} (\mathbf{V}\mathbf{W}^{m-t}\mathbf{U}\boldsymbol{x}_t)_k = (\boldsymbol{y}_m)_k \,, \qquad 1 \leq k \leq b \,, \quad 1 \leq m \leq n \,.$$

Hint: start your proof with showing by induction that $\boldsymbol{h}_m \overset{(ind)}{=} \sum_{t=1}^m \mathbf{W}^{m-t}\mathbf{U}\boldsymbol{x}_t$.

(b) *[4 points]* Define the *linear function* $f : \mathbb{R}^{\mathcal{J}} \to \mathbb{R}^{\mathcal{I}}$ that is equivalent to $g(\{\boldsymbol{x}_t\}_{t=1}^n, \mathbf{W}, \mathbf{U}, \mathbf{V})$:

$$f(\boldsymbol{z})_i \; := \; \sum_{j\in\mathcal{J}} \Theta_{i,j}\, z_j \,, \qquad \forall \boldsymbol{z} \in \mathbb{R}^{\mathcal{J}} \,, \quad \forall i \in \mathcal{I} \,,$$

by defining the index sets $\mathcal{J}$ and $\mathcal{I}$, and by constructing the inputs $\boldsymbol{z} \in \mathbb{R}^{\mathcal{J}}$ from $\{\boldsymbol{x}_t\}_{t=1}^n$ and the parameter matrix/tensor $\boldsymbol{\Theta} \in \mathbb{R}^{\mathcal{I}\times\mathcal{J}}$ from the g's parameters $\mathbf{U}, \mathbf{V}$ and $\mathbf{W}$. Make sure f outputs exactly the same as g!

Solution

(a) We prove by induction $\boldsymbol{h}_m \overset{(ind)}{=} \sum\limits_{t=1}^m \mathbf{W}^{m-t}\mathbf{U}\boldsymbol{x}_t$. Induction beginning $m = 1$:

$$\boldsymbol{h}_1 \overset{(def)}{=} \mathbf{W}\boldsymbol{h}_0 + \mathbf{U}\boldsymbol{x}_1 = \mathbf{U}\boldsymbol{x}_1 = \sum_{t=1}^{1} \mathbf{W}^{1-t}\mathbf{U}\boldsymbol{x}_t \,. \qquad \checkmark$$

Induction step for $m+1$:

$$\boldsymbol{h}_{m+1} \overset{(def)}{=} \mathbf{W}\boldsymbol{h}_m + \mathbf{U}\boldsymbol{x}_{m+1} \overset{(ind)}{=} \mathbf{W}\sum_{t=1}^{m} \mathbf{W}^{m-t}\mathbf{U}\boldsymbol{x}_t + \mathbf{U}\boldsymbol{x}_{m+1} = \sum_{t=1}^{m+1} \mathbf{W}^{m+1-t}\mathbf{U}\boldsymbol{x}_t \,. \qquad \checkmark$$

Lastly we can use the induction of $\boldsymbol{h}_m$ to show that:

$$g(\{\boldsymbol{x}_t\}_{t=1}^n, \mathbf{U}, \mathbf{V}, \mathbf{W})_{k,m} = \Big(\mathbf{V} \underbrace{\sum_{t=1}^{m} \mathbf{W}^{m-t}\mathbf{U}\boldsymbol{x}_t}_{\boldsymbol{h}_m}\Big)_k \overset{(def)}{=} (\boldsymbol{y}_m)_k \,. \qquad \square$$

Rubrik:

- 1 point for the induction beginning

- 2 points for the induction step:
  - 1 point for application of definition and induction assumption
  - 1 point for construction the correct sum over $m+1$
- 1 point for the final equality of g and $\boldsymbol{y}$

(b) Because we need $\boldsymbol{z} \equiv \{\boldsymbol{x}_t\}_{t=1}^n$, we define:

$$\mathcal{J} := \{(u,v) \,|\, 1 \leq u \leq d, 1 \leq v \leq n\} \qquad \text{and} \qquad z_{(u,v)} := (\boldsymbol{x}_v)_u \,.$$

Similarly, the output of f needs to be the same as that of g, so we define

$$\mathcal{I} := \{(k,m) \,|\, 1 \leq k \leq b, 1 \leq m \leq n\} \,.$$

Lastly we only need to extend the sum over t until n by zeroing out non-existing summands, i.e. $\sum_{t=1}^m \phi_t = \sum_{t=1}^n \delta(t \leq m)\, \phi_t, \forall m \leq n$:

$$g(\{\boldsymbol{x}_t\}_{t=1}^n, \mathbf{U}, \mathbf{V}, \mathbf{W})_{k,m} = \sum_{t=1}^{m}\sum_{l=1}^{d} \big(\mathbf{V}\mathbf{W}^{m-t}\mathbf{U}\big)_{k,l} (\boldsymbol{x}_t)_l = \underbrace{\sum_{t=1}^{n}\sum_{l=1}^{d}}_{\sum\limits_{(l,t)\in\mathcal{J}}} \underbrace{\big(\delta(t \leq m)\, \mathbf{V}\mathbf{W}^{m-t}\mathbf{U}\big)_{k,l}}_{\Theta_{(k,m),(l,t)}} \underbrace{(\boldsymbol{x}_t)_l}_{z_{(l,t)}} \,.$$

□

Rubrik:

- 1 point for identifying $\mathcal{J}$ correctly
- 1 point for identifying $\mathcal{I}$ correctly
- 1 point for extending the sum to n, e.g. by using a Kronecker-δ function
- 1 point for defining the correct parameter matrix $\boldsymbol{\Theta}$

End of exam. **Total 42 points.**