Exam CS4400: Deep Reinforcement Learning

13-04-2022 | 13:30–16:30

Student name: ____

Student number: ____________________

- This is a closed-book individual examination with **9 questions** and a total of **50 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 10 (excluding this front page).
- Exam prepared by Wendelin Böhmer.

Question 1 (multiple choice): (20 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: Which of the following definitions is called the *cross entropy loss* over samples x and labels y?

☒ $-\mathbb{E}[\ln p(y|x)]$
☐ $-\mathbb{E}[p(y|x)\, \ln p(y|x)]$
☐ $\mathbb{E}[(f(x)-y)^2]$
☐ $\sqrt{\mathbb{E}[(f(x)-y)^2]}$

Remark: Many students choose the 2nd answer, but this is objectively wrong, as the distribution of y is handled by the expectation (Lecture 2, slide 12).

1.2: Which of the following is **not** a parameter of the pytorch `Conv2d` constructor?

☐ `stride`
☐ `kernel_size`
☒ `return_indices`
☐ `padding`

1.3: How many parameters has a convolutional layer with a $4 \times 3 \times 5 \times 5$ kernel when applied to a 30×20 RGB image?

☐ 75
☒ 300
☐ 416
☐ 600

1.4: Which of the following modifications of the loss $\mathcal{L}[\theta]$ changes the *direction* of the gradient w.r.t. θ?

☐ $\mathcal{L}'[\theta] := \mathcal{L}[\theta] + c, \quad c > 0$
☐ $\mathcal{L}'[\theta] := \mathcal{L}[\theta] + c, \quad c < 0$
☐ $\mathcal{L}'[\theta] := c\,\mathcal{L}[\theta], \quad c > 0$
☒ $\mathcal{L}'[\theta] := c\,\mathcal{L}[\theta], \quad c < 0$

1.5: Which of the following is **not** a value target definition from the lecture?

☐ n-step targets
☐ Monte-Carlo targets
☒ Residual targets
☐ Eligibility-traces targets

1.6: Which of the following is the *on-policy* state-action value target when $a_t \sim \pi(\cdot|s_t)$?

- ☐ $r_t + \gamma \max_a Q(s_{t+1}, a)$
- ☐ $r_t + \gamma Q(s_{t+1}, \arg\max_a Q'(s_{t+1}, a))$
- ☒ $r_t + \gamma Q(s_{t+1}, a_{t+1})$
- ☐ $r_t + \gamma \mathbb{E}[Q(s_{t+1}, a) \,|\, a \sim \pi'(\cdot|s_{t+1})]$

  Remark: π' is another policy and the 4th option is therefore the *off-policy* value target.

1.7: Which of the following statements about target networks is **not** true?

- ☐ Online Q-learning updates the target network after every gradient descent step
- ☐ the original DQN updated the target network after a fixed number of gradient descent steps
- ☒ Double Q-learning updates the target network twice as often as DQN
- ☐ Neural-fitted Q-learning updates the target network after the value function has converged

1.8: Which of the following is called the *Boltzmann policy* of Q-value function $Q^\pi(s, a)$?

- ☐ $\pi'(a|s) = (1-\epsilon)\,\delta(a = \arg\max_{a'} Q(s,a)) + \epsilon\frac{1}{|\mathcal{A}|}$
- ☒ $\pi'(a|s) = \frac{\exp(\epsilon Q(s,a))}{\sum_{a'} \exp(\epsilon Q(s,a'))}$
- ☐ $\pi'(a|s) = \exp(\frac{1}{\epsilon}Q(s,a) - \frac{1}{\epsilon}V(s))$
- ☐ $\pi'(a|s) = \arg\max_\pi Q^\pi(s,a) \quad \text{s.t.} \quad D_{\text{KL}}[\mu(\cdot|s) \| \pi(\cdot|s)] \leq \epsilon$

1.9: Which of the following justifications allows us to add a bias in actor-critic algorithms?

- ☐ the advantage function is sufficient for policy improvement
- ☒ the derivative of a constant bias w.r.t. parameters θ is zero
- ☐ the variance reduction justifies the statistical bias
- ☒ the gradient of the average bias is zero

  Remark: the 2nd answer was supposed to be wrong, but seems to be ambiguously formulated. I decided to give a full point for both the 2nd and 4th answer.

1.10: Which algorithm uses an experience replay buffer?

- ☒ Deep deterministic policy gradients (DDPG)
- ☐ Asynchronous advantage actor-critic (A3C)
- ☐ Trust-region policy optimization (TRPO)
- ☐ Proximal policy optimization (PPO)

  Remark: TRPO and PPO work on minibatches of samples until the policy changed too much, then they resample. This is not an experience replay buffer.

1.11: Which of the following is **not** a condition for local convergence of actor-critic methods?

☐ the learning rate of the critic is higher than that of the actor
☐ the critic is trained by TD(λ) with sufficient large λ
☐ the actor and critic are linear with suitable basis functions
☒ the action space is continuous

1.12: Which of the following algorithms or techniques uses the *reparameterization trick*?

☐ Twin delayed DDPG (TD3)
☒ Soft actor-critic (SAC)
☐ Bootstrapping error accumulation reduction (BEAR)
☐ Counterfactual multi-agent learning (COMA)

1.13: Which exploration method is best suited to learn navigation in a maze (labyrinth)?

☐ Boltzmann sampling of Q-values
☐ Thompson sampling of Q-values
☐ optimistic exploration of Q-values
☒ deep exploration with intrinsic reward

Remark: A maze has many arms in which the agent must explore deeply into. Thompson sampling and optimistic exploration *of Q-values* is shallow and not well suited for a maze.

1.14: Which of the following uncertainty estimates can be **not** be used for Thompson sampling?

☐ dropout
☐ noisy nets
☒ random network distillation
☐ ensembles

Remark: RND cannot be interpreted as variance of the value function (see Lecture 5, slide 13).

1.15: What is the *DAgger* (online) error bound for

$$\epsilon = \mathbb{E}_{\mathcal{D}}\big[\delta(a_t = a_t^*)\big|_{a_t\sim\pi_\theta(\cdot|s_t)}\big] \quad \text{and} \quad f(H,\theta) := \mathbb{E}_{\pi_\theta}\Big[\sum_{t=0}^{H-1}\delta(a_t = a_t^*)\Big|_{\substack{a_t\sim\pi_\theta(\cdot|s_t)\\ a_t^*\sim\pi^*(s_t)}}\Big] \;?$$

☒ $f(H,\theta) \leq C + H\epsilon$
☐ $f(H,\theta) \leq C + H^2\epsilon$
☐ $f(H,\theta) \leq C + H\epsilon^2$
☐ $f(H,\theta) \leq C + H^2\epsilon^2$

1.16: Which of the following is a possible value target $\underline{Q}(s,a)$ for *pessimistic offline* DQN with an ensemble of Q-value functions Q_{θ_i}?

- ☐ $\underline{Q}(s,a) := Q_{\theta_1}(s, \arg\min_{a'} Q_{\theta_2}(s,a'))$
- ☐ $\underline{Q}(s,a) := Q_{\theta_1}(s, \arg\max_{a'} Q_{\theta_2}(s,a'))$
- ☒ $\underline{Q}(s,a) := \min_i Q_{\theta_i}(s,a)$
- ☐ $\underline{Q}(s,a) := \max_i Q_{\theta_i}(s,a)$

1.17: Which property can lead to *cyclic games*?

- ☐ sequential moves
- ☒ simultaneous moves
- ☐ stochastic moves
- ☐ deterministic moves

1.18: Which effect can lead to accidents when two autonomously driving cars try to cross an intersection? You can assume that the cars are from different manufacturers and have been trained using IQL without explicit punishment for crashes.

- ☐ centralized training
- ☒ zero-shot coordination
- ☐ relative overgeneralization
- ☐ value factorization

Remark: The cars suffer from *zero-shot coordination* because they have been trained with IQL by "different manufacturers". Each car can therefore find another Nash-equilibrium. Many students thought the answer was *relative overgeneralization*, which only appears in the presence of strong punishment, but the cars have been trained with IQL "without explicit punishment for crashes".

1.19: In a Dec-POMDP with 2 agents and $|\mathcal{A}^i| = 3$, $\forall i$, how many heads are on an *independent* Q-value function with a DQN architecture?

- ☐ 1
- ☒ 3
- ☐ 8
- ☐ 9

1.20: Which of the following algorithms uses *value factorization*?

- ☐ IQL
- ☐ MADDPG
- ☐ COMA
- ☒ QMIX

Remark: IQL does not factorize the value into independent utilities, like QMIX does, but estimates *independent values*.

Question 2: (2 points)

Explain in no more that 4 sentences the difference between DDPG and TD3.

Solution

Rubrik:

- 0.5 points for "TD3 extends DDPG" or "both are (deterministic) policy gradient methods"
- 1 point for pessimistic values of two critics
- only 0.5 points for "double Q-learning"
- 1 point for regularization with clipped noise
- no point for "performs better"
- max. 2 points

Question 3: (3 points)

Describe in no more that 6 sentences why it could be advantageous to use *robust reinforcement learning* to train an autonomous car. How would robustness affect the transition and reward model of the car?

Solution

This question could be answered in a lot of ways. 1 point for each of the following (up to 3 points):

Rubrik:

- a good motivation to use robust RL
- disruption δ can make collisions happening earlier
- disruption δ can make other cars/pedestrians act more chaotic
- disruption δ can introduce sensor noise
- disruption δ can make the environment act up (e.g. storms)
- the reward model adds reward for large δ
- the transition model must be able to be disturbed by δ
- half points for ambiguous or unclear arguments going in the right direction

Question 4: (3 points)

Name one environment from the OpenAI gym library (which includes the ATARI game library) that is easy to explore and one environment that is hard hard to explore. Explain in 5 sentences or less for both cases what makes exploration easy or hard.

Solution

Rubrik:

- $\frac{1}{2}$ point for an easy environments, and 1 point for a good justification:
  - Cartpole: failures end episode
  - LunarLander: failures end episode
  - Pong: small state-space
- $\frac{1}{2}$ point for an easy environments, and 1 point for a good justification:
  - Mountaincar: small state space but highly adversarial dynamics
  - MontezumasRevenge: large diverse state-space
- depending on the explanation medium environments could be either:
  - Acrobot: small state space but somewhat adversarial dynamic
  - Breakout: large state space but many states with similar values
- $-\frac{1}{2}$ if an environment name has been forgotten or significantly altered (like "MountainClimber"), but only once.

Question 5: (2 points)

In 4 sentences or less, name and explain **two** challenges of *offline* reinforcement learning in comparison to *online* reinforcement learning.

Solution

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

- no exploration: unknown state-actions remain unknown
- distribution shift: policy and state distribution changes
- learning stability: errors cannot be detected/corrected
- while not strictly a challenge, offline RL methods also count, e.g.:
  - constrain the policy
  - limit available actions
  - make the values pessimistic
- adding incorrect arguments to support correct points can lead to $-\frac{1}{2}$
- $\frac{1}{2}$ point if the argument is not correct, but goes into the right direction
- $\frac{1}{2}$ point if the same point is made twice, but only if the second time shows another perspective

Question 6: (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	0/2	-1/-1	1/0	3/1
a_2^1	2/0	0/0	3/1	0/-1
a_3^1	0/0	-1/-2	1/3	2/0
a_4^1	1/0	1/2	1/1	2/1

You do not need to justify your answer. Which Nash equilibrium would player 1 prefer? Which would be better for player 2?

Solution

Rubrik:

- 1 point for $a_4^1, a_2^2 : 1/2$
- 1 point for $a_2^1, a_3^2 : 3/1$
- -1 point for any wrong Nash equilibrium (minimum 0 points)
- 1 point for: player 1 prefers 3/1 but player 2 prefers 1/2
- no point if either player picks the wrong (or ambiguous) equilibrium

Question 7: (5 points)

Prove that the variance of the empirical mean $f_n := \frac{1}{n}\sum_{i=1}^n x_i$, based on n samples $x_i \in \mathbb{R}$ drawn i.i.d. from the Gaussian distribution $\mathcal{N}(\mu, \sigma^2)$, is $\mathbb{V}[f_n] = \frac{\sigma^2}{n}$, *without* using the fact the variance of a sum of independent variables is the sum of the variables' variances.

Solution

The major insights are that $\mathbb{E}[x_i] = \mu, \forall i$, $\mathbb{E}[x_i x_j] = \mathbb{E}[x_i]\mathbb{E}[x_j]$ if $i \neq j$ due to i.i.d. sampling and that $\mathbb{E}[(x_i - \mu)^2] = \sigma^2$.

$$\begin{aligned}
\mathbb{V}[f_n] &= \mathbb{E}\Big[\big(\tfrac{1}{n}\sum_{i=1}^n x_i - \mu\big)^2\Big] &= \tfrac{1}{n^2}\sum_{i=1}^n\sum_{j=1}^n \mathbb{E}[(x_i-\mu)(x_j-\mu)] \\
&= \tfrac{1}{n^2}\sum_{i\neq j}\underbrace{\mathbb{E}[(x_i-\mu)]}_{0}\underbrace{\mathbb{E}[(x_j-\mu)]}_{0} + \tfrac{1}{n^2}\sum_{i=1}^n \underbrace{\mathbb{E}[(x_i-\mu)^2]}_{\sigma^2} &= \frac{\sigma^2}{n}\,.
\end{aligned}$$

Rubrik:

- 1 point for the correct definition of variance $\mathbb{V}$
- 1 point for using $\mathbb{E}[x_i] = \mu$
- 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
- $-\frac{1}{2}$ points for minor mistakes (e.g. $\mathbb{E}[x_i x_j] = 0$ for $i \neq j$)
- but no point loss for forgetting little things like one or two $\pm$ mistakes

Question 8: (5 points)

Let $\mathbf{X} \in \mathbb{R}^{d \times n}$ denote a time series of n (d-dimensional) samples and $g(\mathbf{X}, \mathbf{K})$ denote a one-dimensional *convolutional layer*, with given kernel $\mathbf{K} \in \mathbb{R}^{b \times d \times n_K}$, defined as

$$g(\mathbf{X}, \mathbf{K})_{k,m} := \sum_{l=1}^{n_K} \sum_{p=1}^{d} K_{k,p,l} \, X_{p,m+l-1} \,, \qquad 1 \leq k \leq b \,, \qquad 1 \leq m \leq n - n_K + 1 \,.$$

Define the equivalent *linear function* $f : \mathbb{R}^{\mathcal{J}} \to \mathbb{R}^{\mathcal{I}}$ (and the corresponding index sets $\mathcal{J}$ and $\mathcal{I}$):

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

by constructing $\boldsymbol{z}$ from $\mathbf{X}$ and the parameter matrix/tensor $\boldsymbol{\Theta} \in \mathbb{R}^{\mathcal{I} \times \mathcal{J}}$ from g's kernel $\mathbf{K}$.
Hint: start by defining $\mathcal{J}$ and $\mathcal{I}$.

Solution

The input $\boldsymbol{z}$ of the linear function f shall be euqvialent to $\mathbf{X}$, so we define

$$\mathcal{J} := \{(u, v) \,|\, 1 \leq u \leq d, 1 \leq v \leq n\} \quad \text{and} \quad z_{(u,v)} := X_{u,v} \,.$$

Conversely, the output of f must be in $\mathbb{R}^{b \times n - n_K + 1}$, like g's output, so we define

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

Next we rewrite the g as sum over v ($v{=}m{+}l{-}1 \;\Rightarrow\; l{=}v{-}m{+}1$, note that $v{\in}\{m,\ldots,m{+}n_K{-}1\}$) and then extend v to $1 \ldots n$ by zeroing out impossible summands with $\delta(\, m{\leq}v{<}m{+}n_K)$:

$$g(\mathbf{X}, \mathbf{K})_{k,m} = \sum_{v=m}^{m+n_K-1} \sum_{u=1}^{d} K_{k,u,v-m+1} X_{u,v} = \underbrace{\sum_{v=1}^{n} \sum_{u=1}^{d}}_{\sum\limits_{(u,v) \in \mathcal{J}}} \underbrace{\delta(\, m{\leq}v{<}m{+}n_K) \, K_{k,u,v-m+1}}_{\Theta_{(k,m),(u,v)}} \underbrace{X_{u,v}}_{z_{(u,v)}} = f(\boldsymbol{z})_{(k,m)} \,.$$

Rubrik:

- 1 point for identifying $\mathcal{J}$ correctly
- 1 point for identifying $\mathcal{I}$ correctly
- only $\frac{1}{2}$ point for the right dimensions, but not a correct set (second occurrence is follow-up)
- only $\frac{1}{2}$ point if $\mathcal{J}$ or $\mathcal{I}$ is described correctly, but not defined mathematically
- 1 point for using (an equivalent of) a δ function
- $\frac{1}{2}$ point for the correct indices in the δ function
- $\frac{1}{2}$ point for the definition of $\boldsymbol{z}$ (even indirectly)
- 1 point for defining otherwise the correct $\boldsymbol{\Theta}$, i.e. the correct kernel for (u, v)
- $-\frac{1}{2}$ point deduction for each "flipped" index pair (no deduction for follow-up flips)
- $-\frac{1}{2}$ point deduction for general math errors (only if they simplify the solution, no deduction for follow-up errors)

Question 9: (programming) **(7 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 L_1 loss to learn the state-value v_θ in the given `MyLearner` class:

$$\min_\theta \mathbb{E}\Big[\frac{1}{m}\sum_{i=1}^{m}\frac{1}{n_i}\sum_{t=0}^{n_i-1}\big|r_t^i + \gamma v_{\theta'}(s_{t+1}^i) - v_\theta(s_t^i)\big|\,\Big|\,\tau_{n_i}^i \sim \mathcal{D}\Big],$$

where $\tau_{n_i}^i := \{s_t^i, r_t^i\}_{t=0}^{n_i-1} \cup \{s_{n_i}^i\}$ are m trajectories of states $s_t^i \in \mathbb{R}^d$ and rewards $r_t^i \in \mathbb{R}$. The last state $s_{n_i}^i$ is always terminal. $|x|$ denotes the absolute value of x and θ' the target network parameters which shall not change during gradient descent.

Hint: The given `model` computes the values v_θ for a minibatch of states s_t^i, that is, a tensor `S` of arbitrary dimensionality, except for `S.shape[-1]`$=d$. `gamma`$=\gamma$. Do not to bootstrap from $v_{\theta'}(s_{n_i}^i)$. You can ignore problems with singleton dimensions, e.g. whether `x.sum(dim=2)` removes the `dim=2`.

```
class MyLearner:
    def __init__(self, model, gamma=0.99):
        self.model = model
        self.target_model = deepcopy(model)
        self.gamma = gamma
        self.optimizer = torch.optim.Adam(model.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, :] = $s_t^i$
               - batch['rewards'][i, t] = $r_t^i$
               - batch['mask'][i, t] = $t < n_i$
               - batch['terminals'][i, t] = $s_t^i$ is terminal """
        loss = 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!

Solution

```
values = self.model(batch['states'])
t_values = ~batch['terminals'] * self.target_model(batch['states'])
targets = batch['rewards'][:, :-1] + self.gamma * t_values[:, 1:]
td = targets.detach() - values[:, :-1]
mask = batch['mask'][:, :-1]
loss = (td * mask / mask.sum(dim=1, keepdim=True)).sum() / mask.shape[0]
```

Rubrik:

- 1 point for computing the values in one call

- 1 point for computing the target values in one call
- Computing the values in loops only yields $\frac{1}{2}$ point per value
- Ignore all problems stemming from left-over singleton dimensions (e.g. in `dim=2`)
- 1 point for correct use of time (`[:, :-1]` vs. `[:, 1:]`) for values
- In case of loops, using `t` and `t+1` correctly counts as follow-up error
- No point deduction for incorrect timing in `batch['rewards']` or `batch['terminals']`
- 1 point for using `batch['terminals']` correctly
- 1 point for correct TD-error (including the `.abs()`)
- 1 point for detaching the targets correctly
- 1 point for masking with `batch['mask']` and normalizing w.r.t. the mask
- Two correct **`for`** loops (second holds on `mask`/`terminal`) count as masking!
- No point deduction for `loss = (td * mask).abs().sum() / mask.sum()`
- No point deduction for forgetting to time the mask `batch['mask'][:, :-1]`
- Ignore small Python/PyTorch errors, but punish those which seriously affect the loss (e.g. `[:-1]` vs. `[:, :-1]`)

End of exam. **Total 50 points.**