Exam CS4400: Deep Reinforcement Learning

26-01-2024 | 9:00–12:00

Student name: ____

Student number: ____________________

- This is a closed-book individual examination with **9 questions** and a total of **94 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): (38 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: What is called a *Markov chain*?

☐ A set of observed states, actions and rewards, that are drawn by executing a policy in a MDP.
☐ A set of observed actions and observations, that are drawn by executing a policy in a POMDP.
☒ A set of random variables of states, actions and rewards, that are induced by a MDP and a policy.
☐ A set of random variables of actions and observations, that are induced by a POMDP and a policy.

1.2: Which relationship between state value $V^\pi(s)$ and state-action value $Q^\pi(s,a)$ is correct?

☐ $V^\pi(s) = \max_a Q^\pi(s,a)$
☒ $V^\pi(s) = \mathbb{E}[Q^\pi(s,a) \,|\, a \sim \pi(\cdot|s)]$
☐ $Q^\pi(s,a) = r(s,a) + \gamma \max_{a'} \mathbb{E}[V^\pi(s') \,|\, s' \sim P(\cdot|s,a')]$
☐ $Q^\pi(s,a) = \mathbb{E}[r(s',a) + \gamma V^\pi(s') \,|\, s' \sim P(\cdot|s,a)]$

1.3: Which PyTorch function collects the entries of tensor `a` at the indices `b` of dimension `d`?

☐ `torch.cat([a, b], dim=d)`
☐ `torch.stack([a, b], dim=d)`
☒ `a.gather(dim=d, index=b)`
☐ `a.scatter(dim=d, index=b)`

1.4: Which algorithm propagates newly observed reward the *fastest* to earlier states?

☐ Neural-fitted Q-learning
☐ DQN with hard target updates
☐ DQN with soft target updates
☒ DQN without a target network

1.5: What causes *catastrophic forgetting*?

☒ distribution shift of input samples
☐ distribution shift of output labels
☐ experience replay of old input samples
☐ experience replay of old output labels

1.6: Which actor samples the transitions in *on-policy* sampling?

- ☐ a fixed behavior policy π_β
- ☐ an older sampling policy μ
- ☒ the current policy π_θ
- ☐ any policy

1.7: What makes the DQN algorithm more *sample efficient*?

- ☒ sampling less transitions between gradient updates
- ☐ sampling more transitions between gradient updates
- ☐ decreasing the maximum length of episodes
- ☐ increasing the maximum length of episodes

1.8: *Pooling layers* are in practice used to ...

- ☒ decrease the size of the output tensor relative to the input tensor
- ☐ increase the size of the output tensor relative to the input tensor
- ☐ normalize the input tensor across the feature dimension
- ☐ normalize the input tensor across the batch dimension

1.9: *Relational graph convolutional networks* are ...

- ☐ a simplification of *convolutional neural networks*
- ☐ a simplification of *graph convolutional networks*
- ☒ a generalization of *convolutional neural networks*
- ☒ a generalization of *graph convolutional networks*

  Note: the intention of this question was to probe the insight that R-GCN are GCN, however, the formulation *a generalization of* can be interpreted as “can be made into a”, which is also true for CNN. Both answers are therefore technically correct.

1.10: Which architecture should be used when training ATARI games with the the DRQN algorithm?

- ☐ a number of convolutional layers, followed by a number of linear layers
- ☒ a number of convolutional layers, followed by a LSTM and a final linear layer
- ☐ a number of multi-headed attention layers, followed by max-pooling and a final linear layer
- ☐ a number of graph neural network layers, followed by max-pooling and a final lienar layer

1.11: When learning a POMDP, a *belief* denotes...

- ☒ a distribution over states given a history
- ☐ a distribution over histories given a state
- ☐ a sequence of past observations and actions
- ☐ a sequence of past states, actions and rewards

1.12: When training the REINFORCE algorithm, which information is **not** required?

- ☐ the current state of a transition
- ☐ the action of a transition
- ☒ the next state of a transition
- ☐ the return of the remaining episode

1.13: Which λ has the *smallest variance* for TD(λ) targets?

- ☒ $\lambda = 0$
- ☐ $\lambda = \frac{1}{\sqrt{2}}$
- ☐ $\lambda = 1$
- ☐ $\lambda = \sqrt{2}$

1.14: Why do we prefer PPO over TRPO in practice?

- ☐ because policy ratio clipping is more precise than gradient descent with constraints
- ☒ because gradient descent with constraints is more computationally expensive
- ☐ because TRPO requires to estimate Q^π, but PPO only V^π
- ☐ because PPO can use an experience replay buffer and TRPO can not

1.15: Formally, the reparametrization trick $\boldsymbol{a} = f_\theta(s, \boldsymbol{\epsilon}) \sim \pi_\theta(\cdot|s), \boldsymbol{\epsilon} \sim N(\cdot|\mathbf{0}, \mathbf{I})$, requires:

- ☐ f_θ needs to be injective in $\boldsymbol{\epsilon}$
- ☐ f_θ needs to be surjective in $\boldsymbol{\epsilon}$
- ☒ f_θ needs to be bijective in $\boldsymbol{\epsilon}$
- ☐ f_θ needs to be monotonic in $\boldsymbol{\epsilon}$

1.16: Which of the following is *reducible*?

- ☐ aleatoric uncertainty
- ☒ epistemic uncertainty
- ☐ model-bias
- ☐ stochastic transitions

1.17: Which of the following can **not** be used as *posterior* for model parameters?

- ☐ diagonal Gaussian
- ☐ dropout layers
- ☐ ensemble of networks
- ☒ random network distillation

1.18: What property of the reward function defines a *two-player zero-sum* normal-form game?

- ☐ $r^1(s, \boldsymbol{a}) = r^2(s, \boldsymbol{a}) + c$
- [✓ ☒ $r^1(s, \boldsymbol{a}) = -r^2(s, \boldsymbol{a}) + c$]
- ☐ $r^1(s, \boldsymbol{a}) \leq r^2(s, \boldsymbol{a}) + c$
- ☐ $r^1(s, \boldsymbol{a}) \geq r^2(s, \boldsymbol{a}) + c$

1.19: Which decision affects generalization in *multi-task RL* the most?

- ☐ the regularization constant
- ☐ the discount factor
- [✓ ☒ the network architecture]
- ☐ using stochastic policies

Question 2: (6 points)

Explain in 6 sentences or less why the DDPG algorithm requires **two** separate Optimizer objects. Which part of the loss makes this necessary and what are the implications for the network architecture?

Hint: it suffices to describe the parts of the loss that make this necessary, you do not need to define the complete DDPG loss.

Solution

As all actor-critic algorithms, DDPG has two losses, one for the value function and one for the policy. The policy loss contains the Q-value of the action that the deterministic policy would select, so performing gradient descent on that loss with one optimizer that changes both neural nets would change the value as well. We do not want that (DDPG has a separate value loss). The two functions need therefore to be represented by two networks, and we need to optimize the value with another optimizer object than the policy. **Rubrik:**

- 2 points for mentioning 2 neural networks, one for policy and one for Q-values. No points for describing DQN, A2C or PPO.
- 2 points for mentioning that the policy is an input to the Q-value in the policy loss. Only 1 point if deterministic policy is mentioned.
- 2 points for mentioning that the Q-value parameters should not be changed by the policy loss. Only 1 point if the idea is there, but insufficiently explained.

Question 3: (6 points)

Note: Looking at the answers, this question has been deemed too hard and not precisely enough formulated to be fair. Any of the 6 possible points that can be earned here have therefore be tuned into **extra-points**, that do not count towards the total of the exam (but you still get to keep them).

In 6 sentences or less, explain why MADDPG uses the *observed actions* $\boldsymbol{a}_t^{-i}$ of all other agents (except of agent i) in i's policy loss, instead of using the actions $\boldsymbol{\pi}^{-i}(\boldsymbol{s}_t)$ that their *policies would have chosen*.

Solution

Like DDPG, MADDPG estimates the Q-value function with a Bellman error loss from an experience replay buffer. This loss only computes gradients for actions that have been executed in the environment and can be found in the replay buffer. If the current policy of other agents would choose actions that are not present in the replay buffer, their estimate can be arbitrarily bad. To avoid this wherever possible (in DDPG the action a^i of agent i needs to be chosen by the policy), MADDPG uses the corresponding action to the state sampled from the replay buffer, which minimizes this potential error. **Rubrik:**

- 2 points for mentioning the use of a centralized Q-value. Only 1 point if only CTDE is mentioned. Only 1 point if only the connection to the policy loss is pointed out.
- 2 points for mentioning that both Q-value and policy are trained on the actions in the replay buffer. Only 1 point to notice that the policy may differ from the actions in the replay buffer. Only 1 point for mentioning the replay buffer.
- 2 points for explaining how using actions the Q-value is trained on decreases out-of-distribution errors. Only 1 point if only stability is mentioned.

Question 4: **(6 points)**

In 6 sentences or less, explain the difference between *aleatoric* and *epistemic* uncertainty. Speculate how the exploration behavior of a RL agent would change if it would use aleatoric (instead of epistemic) uncertainty to compute intrinsic reward. Also name a counter-example, which demonstrates the possible failure of this approach.

Solution

Aleatoric uncertainty is caused by irreducible stochasticity of the environment, whereas epistemic uncertainty is caused by the agent's lack of knowledge, which is reducible by exploration. An agent using aleatoric intrinsic reward would be attracted to random outcomes (both in reward and in transitions). While in the beginning this randomness can be caused by changing policies, which would vanish once they have converged, in the long run this attraction would not diminish. This can lead to suboptimal behavior if the intrinsic reward in some states is higher than the true optimal reward. An example is the "noisy TV room", where in one room without reward is a TV, which shows white noise and is therefore a new observation every time it is visited. If the intrinsic reward caused by this aleatoric uncertainty is high, the agent might go for the room with the TV, and since this uncertainty will never go away, this effect will never vanish. **Rubrik:**

- 2 points for a good contrast between aleatoric and epistemic uncertainty.
- 2 points for a good explanation of the effect on exploration. Only 1 point for the correct conclusion or for the correct effect without a conclusion.
- 2 points for a good example.

Question 5: **(6 points)**

In 6 sentences or less, explain the difference between *Double Q-learning* in DQN and *Twin Q-learning* (also known as *Clipped Double Q-learning*) in TD3. What is the underlying problem that both aim to solve, and how do the two methods differ?

Solution

Both aim to avoid the overestimation of Q-values $\mathbb{E}[\max_a Q(s,a)] \geq \max_a \mathbb{E}[Q(s,a)]$ due to Jensen's inequality. However, Double Q-learning learns a second Q-value function that is statistically independent (or use the target net in Deep Double Q-learning as a proxy) to choose the actions $\mathbb{E}[Q(s, \arg\max_a Q'(s,a))] \approx \max_a \mathbb{E}[Q(s,a)]$, whereas Twin Q-learning estimates a second Q-value function based on the same data (so it is not statistically independent) and performs the maximum on the *smaller* of the two value estimation, i.e., $\mathbb{E}[\max_a \min_i Q_i(s,a)] \approx \max_a \mathbb{E}[Q(s,a)]$. Both approaches are imperfect but work better than DQN. **Rubrik:**

- 2 points for an explanation of overestimation in DQN.
- 2 points for an explanation how Double Q-learning solves it.
- 2 points for an explanation how Twin Q-learning solves it.

Question 6: **(10 points)**

You want to train an RL agent to solve the classical jump-and-run game **Super Mario Brothers** based on the input and the output of the game console. Assume very simple levels that scroll from left to right, such that no two locations yield the same image.

(a) *[3 points]* Name a suitable deep reinforcement learning algorithm that can learn to play the game as well as a human could, and name at least one argument to justify your answer.

(b) *[3 points]* Name the module-classes you would use in the agent's network architecture and describe how they would be connected (in words, drawing a diagram is not necessary).

Hint: linear layers are always present, so they do not have to be specifically mentioned.

(c) *[4 points]* Which changes to the inputs/outputs, algorithm, architecture and/or training procedure would be necessary to allow the agent to also perform well if the *background of each level can be customized* with private photographs of the player.

Solution

(a) The controller has discrete actions, so DQN, REINFORCE, TRPO, PPO and SAC would all work. **Rubrik:**

- 2 points for an algorithm with discrete actions. Only 1 point for MARL algorithms for discrete actions. Only 1 point for *multi-task algorithm*. No points for algorithms for continuous actions.
- 1 point for a good justification. No point for talking about *tasks* without specifying it.
- if multiple algorithms/justifications are given, give points for the worst answer.

(b) The input are images, so a CNN is necessary. Since no two locations look the same, an RNN is not needed. **Rubrik:**

- 2 points for mentioning CNN. Only 1 point for over-complicated answers like GNN. No point for RNN/LSTM. E.g. CNN+LSTM yields 0 points, but often 1 point for the justification of CNNs with images.
- 1 point for a good justification or a reasonable order.
- if multiple architectures/justifications are given, give points for the worst answer.

(c) We need to use multi-task learning that ignore the background. This can be achieved by training with random background images (so the neural net cannot attach meaning to background pixels). However, other changes are not necessary, as the images are still unique and the task does not change. **Rubrik:**

- 2 point for mentioning or describing multi-task RL. Only 1 point if they want to use LSTMs.
- 2 points for describing the changed training procedure.
- Point deductions for any unnecessary changes (e.g. adding the task as an extra input).

Question 7: **(8 points)**

Let the transition model $P(\boldsymbol{s}'|\boldsymbol{s},a) := \mathcal{N}(\boldsymbol{s}'|\boldsymbol{\mu}(\boldsymbol{s},a),\sigma^2\mathbf{I}), \forall \boldsymbol{s} \in \mathcal{S} \subset \mathbb{R}^m, \forall a \in \mathcal{A}$, be a Gaussian with constant covariance matrix $\sigma^2\mathbf{I} \in \mathbb{R}^{m\times m}$ and conditional mean $\boldsymbol{\mu}(\boldsymbol{s},a) \in \mathbb{R}^m$. We consider an affine value function $v_\theta(\boldsymbol{s}) := \boldsymbol{\theta}^\top \boldsymbol{s} + b$, with parameters $\boldsymbol{\theta} \in \mathbb{R}^m, b \in \mathbb{R}$. Let in the following $\bar{v} := \frac{1}{n}\sum_{i=1}^n v_\theta(\boldsymbol{s}'_i)$ denote the empirical average of the future value for some given state-action pair $(\boldsymbol{s},a) \in \mathcal{S}\times\mathcal{A}$, based on n next states $\{\boldsymbol{s}'_i\}_{i=1}^n$ sampled i.i.d. from model P.

(a) *[4 points]* Prove that the empirical average $\bar{v}$ is an *unbiased* estimator of $\mathbb{E}[v_\theta(\boldsymbol{s}')|\boldsymbol{s}' \sim P(\cdot|\boldsymbol{s},a)]$.

(b) *[4 points]* Prove that the variance of $\bar{v}$ is $\mathbb{V}[\bar{v}] = \frac{\sigma^2}{n}\|\boldsymbol{\theta}\|^2$.

Solution

(a) Let $\boldsymbol{\mu} := \boldsymbol{\mu}(\boldsymbol{s},a)$ and let $\hat{\mathbb{E}}[\cdot] := \mathbb{E}[\cdot|\boldsymbol{s}' \sim P(\cdot|\boldsymbol{s},a)]$. An estimator is unbiased if it returns the correct result in expectation:

$$\hat{\mathbb{E}}[\bar{v}] = \tfrac{1}{n}\sum_{i=1}^n \hat{\mathbb{E}}[v_\theta(\boldsymbol{s}'_i)] = \tfrac{1}{n}\sum_{i=1}^n \boldsymbol{\theta}^\top \underbrace{\hat{\mathbb{E}}[\boldsymbol{s}'_i]}_{\boldsymbol{\mu}} + b = \boldsymbol{\theta}^\top\boldsymbol{\mu} + b = \boldsymbol{\theta}^\top \underbrace{\hat{\mathbb{E}}[\boldsymbol{s}']}_{\boldsymbol{\mu}} + b = \hat{\mathbb{E}}[v_\theta(\boldsymbol{s}')]\,.$$

Both terms are identical, and $\bar{v}$ is therefore an unbiased estimator. **Rubrik:**

- 1 point for using the correct definition of *unbiased estimator*
- 1 point for the correct ansatz
- 1 point for the use of $\mathbb{E}[\boldsymbol{s}'_i] = \boldsymbol{\mu}, \forall i$
- 1 point for putting it all correctly together

(b) Note that $\boldsymbol{s}'_i$ are i.i.d., and that thus $\hat{\mathbb{E}}[(\boldsymbol{s}'_i - \boldsymbol{\mu})(\boldsymbol{s}'_j - \boldsymbol{\mu})^\top] = \delta_{ij}\sigma\mathbf{I}$, as for all $i \neq j$ the expectation can be factorized into $(\mathbb{E}[\boldsymbol{s}'_i] - \boldsymbol{\mu})(\mathbb{E}[\boldsymbol{s}'_j] - \boldsymbol{\mu})^\top = \mathbf{0}$:

$$\begin{aligned}
\mathbb{V}[\bar{v}] &= \hat{\mathbb{E}}\big[(\bar{v} - \hat{\mathbb{E}}[\bar{v}])^2\big] &&= \hat{\mathbb{E}}\Big[\Big(\tfrac{1}{n}\sum_{i=1}^n v_\theta(\boldsymbol{s}'_i) - \boldsymbol{\theta}^\top\boldsymbol{\mu} - b)\Big)^2\Big] \\
&= \hat{\mathbb{E}}\Big[\Big(\tfrac{1}{n}\sum_{i=1}^n \boldsymbol{\theta}^\top(\boldsymbol{s}'_i - \boldsymbol{\mu})\Big)^2\Big] &&= \tfrac{1}{n^2}\sum_{i=1}^n\sum_{j=1}^n \boldsymbol{\theta}^\top \underbrace{\hat{\mathbb{E}}\big[(\boldsymbol{s}'_i - \boldsymbol{\mu})(\boldsymbol{s}'_j - \boldsymbol{\mu})^\top\big]}_{\delta_{ij}\sigma^2\mathbf{I}}\boldsymbol{\theta} \\
&= \tfrac{\sigma^2}{n^2}\underbrace{\boldsymbol{\theta}^\top\boldsymbol{\theta}}_{\|\boldsymbol{\theta}\|^2} \underbrace{\sum_{i=1}^n \overbrace{\sum_{j=1}^n \delta_{ij}}^{1}}_{n} &&= \tfrac{\sigma^2}{n}\|\boldsymbol{\theta}\|^2\,.
\end{aligned}$$

Rubrik:

- 1 point for correct definition of the variance
- 1 point for taking the two sums correctly out of the square
- 1 point for the use of $\mathbb{E}[(\boldsymbol{s}'_i - \boldsymbol{\mu})(\boldsymbol{s}'_i - \boldsymbol{\mu})^\top] = \sigma\mathbf{I}$
- 1 point for putting it all correctly together

An alternative solution exist, using (i) *constant factors can be squared and taken outside a variance*, (ii) *the variance of a sum of independent variables is the sum of the variables' variances*, and (iii) *adding a constant does not change the variance* (which follows from (ii)):

$$\begin{aligned}
\mathbb{V}[\bar{v}] &= \mathbb{V}\Big[\tfrac{1}{n}\sum_{i=1}^n v_\theta(\boldsymbol{s}'_i)\Big] \overset{\text{(i)}}{=} \tfrac{1}{n^2}\mathbb{V}\Big[\sum_{i=1}^n \underbrace{(\boldsymbol{\theta}^\top\boldsymbol{s}'_i + b)}_{\text{i.i.d.}}\Big] \overset{\text{(ii)}}{=} \tfrac{1}{n^2}\sum_{i=1}^n \mathbb{V}\Big[\boldsymbol{\theta}^\top\boldsymbol{s}'_i + b\Big] \\
&\overset{\text{(iii)}}{=} \tfrac{1}{n^2}\sum_{i=1}^n \mathbb{V}\Big[\sum_{k=1}^m \underbrace{\theta_k(\boldsymbol{s}'_i)_k}_{\text{decorrelated}}\Big] \overset{\text{(ii)+(i)}}{=} \tfrac{1}{n^2}\sum_{i=1}^n\sum_{k=1}^m \theta_k^2 \underbrace{\mathbb{V}[(\boldsymbol{s}'_i)_k]}_{\sigma^2} = \tfrac{\sigma^2}{n}\|\boldsymbol{\theta}\|^2\,.
\end{aligned}$$

Note that the second use of (ii) holds because the state dimensions are decorrelated. **Rubrik:**

- 1 point for using (i)
- 1 point for the use of (ii) due to i.i.d. sampling
- 1 point for the use of (ii) due to decorrelation
- 1 point for putting it all correctly together

Question 8: **(6 points)**

Let $\Phi := \{\phi_1(\mathbf{X}) = \mathbf{X}, \phi_2(\mathbf{X}) = \mathbf{X}^\top\}$ denote a *group of mappings* for matrices $\mathbf{X} \in \mathbb{R}^{2\times 2}$. Let further $f(\mathbf{X}) := \sum_{i=1}^{2}\sum_{j=1}^{2} W_{ij}X_{ij} + b$ be an affine function of input matrices $\mathbf{X} \in \mathbb{R}^{2\times 2}$, with parameters $\mathbf{W} \in \mathbb{R}^{2\times 2}$ and $b \in \mathbb{R}$. Prove that the function f is **invariant** under Φ if and only if $W_{12} = W_{21}$.

Solution

Invariance under Φ requires that for each input $\mathbf{X}$, all mappings in Φ have the same output, i.e., $f(\phi_1(\mathbf{X})) = f(\phi_2(\mathbf{X})), \forall \mathbf{X} \in \mathbb{R}^{2\times 2}$.

$$\begin{aligned} 0 \overset{!}{=} f(\phi_1(\mathbf{X})) - f(\phi_2(\mathbf{X})) &= \sum_{i=1}^{n}\sum_{j=1}^{n} W_{ij}X_{ij} + b - \Big(\sum_{i=1}^{n}\sum_{j=1}^{n} W_{ij}X_{ji} + b\Big) \\ = W_{12}X_{12} + W_{21}X_{21} - W_{12}X_{21} - W_{21}X_{12} &= \underbrace{(W_{12} - W_{21})}_{0}X_{12} + \underbrace{(W_{21} - W_{12})}_{0}X_{21}\,. \end{aligned}$$

The equation holds for all $\mathbf{X}$ if an only if $W_{12} = W_{21}$. If this constraint is violated, there would exist a $\mathbf{X}$ for which the invariance does not hold. **Rubrik:**

- 2 points for the correct definition of invariance under Φ. Note that we used $f(\mathbf{X}) = f(\phi(\mathbf{X})), \forall \phi \in \Phi, \forall \mathbf{X}$ in the lecture, which yields an identical ansatz here. Only 1 point for $\phi_i(f(\mathbf{X})) = f(\phi_i(\mathbf{X}))$.
- 2 points for the correct ansatz of $f(\phi_1(\mathbf{X})) = f(\phi_2(\mathbf{X})), \forall \mathbf{X} \in \mathbb{R}^{2\times 2}$. Only 1 point if one $W_{12} = W21 \Rightarrow$ invariance is proven.
- 2 points for deriving that $W_{12} - W_{21} \overset{!}{=} 0$ from the ansatz. Only 1 point if one $W_{12} = W21 \Rightarrow$ invariance is proven.

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 *n-step* loss for *on-policy values* in the given `MyLearner` class **efficiently**:

$$\min_\theta \mathbb{E}\Big[\Big(\sum_{k=0}^{n-1}\gamma^k r^i_{t+k} + \gamma^n v_{\theta'}(\boldsymbol{s}^i_{t+n}) - v_\theta(\boldsymbol{s}^i_t)\Big)^2 \,\Big|\, \langle \boldsymbol{s}^i_t, r^i_t, \ldots, \boldsymbol{s}^i_{t+n-1}, r^i_{t+n-1}, \boldsymbol{s}^i_{t+n}\rangle \sim \mathcal{D}\Big].$$

The loss trains a state-value function v_θ, that takes a variable size mini-batch of states $\boldsymbol{s}^i_{t+k} \in \mathbb{R}^d$ as input, and outputs the network's value estimate for each state $\boldsymbol{s}^i_{t+k}$. Note you can call v_θ, i.e. **`self`**`.value_model`, with any tensor of size $m \times d, \forall m \in \mathbb{N}$, which will result in an output tensor of size $m \times 1$. Each semi-gradient descent step, i.e. each call to `MyLearner.train`, receives a mini-batch of m sub-trajectories $\langle \boldsymbol{s}^i_t, r^i_t, \ldots, \boldsymbol{s}^i_{t+n-1}, r^i_{t+n-1}, \boldsymbol{s}^i_{t+n}\rangle$ of length n. The number of subtajectories m can vary, but the length n is constant. The parameters θ' of the target network shall be identical to those of the current model, i.e. $\theta' = \theta$, but shall not be changed by gradient descent. `batch['terminals']` also encodes whether the last state $\boldsymbol{s}^i_{t+n}$ of each sub-trajectory i is terminal. Ensure that terminal states are handled properly.

```python
import torch
from torch.nn.functional import mse_loss                  # MSE loss

class MyLearner:
    def __init__(self, model, gamma=0.99, n=10):
        self.value_model = model
        self.gamma = gamma
        self.n = n
        self.optimizer = torch.optim.Adam(model.parameters())

    def train(self, batch):
        """ Performs one gradient descent step on the loss defined above.
            "batch" is a dictionary where the first dimension or each
            tensor has the same (batch-)size $m$:
            - batch['states'][i, k, :] = $s^i_{t+k} \in \mathbb{R}^d$,   $0 \le k \le n$,   $0 \le i < m$
            - batch['rewards'][i, k] = $r^i_{t+k} \in \mathbb{R}$,       $0 \le k < n$,   $0 \le i < m$
            - batch['terminals'][i] = $s^i_{t+n}$ is terminal          $0 \le i < m$ """
        assert batch['rewards'].shape[1] == self.n,
            "The batch must contain sequences of $n$ time steps"
        # all_gammas is a (1, n) dimensional tensor containing all $\gamma^k$
        all_gammas = torch.tensor([self.gamma**k for k in range(self.n)])
        all_gammas = all_gammas.unsqueeze(dim=0)
        # Define the loss
        loss = 0
        # YOUR CODE HERE
        # Gradient descent step on the loss
        self.optimizer.zero_grad()
        loss.backward()
        self.optimizer.step()
        return loss.item()
```

Solution

```
n_step_rewards = (batch['rewards'] * all_gammas).sum(dim=1)
end_values = self.value_model(batch['states'][:, -1])
bootstraps = ~batch['terminals'] * end_values.squeeze()
n_step_targets = n_step_rewards + (self.gamma**self.n) * bootstraps
values = self.value_model(batch['states'][:, 0]).squeeze()
loss = mse_loss(values, n_step_targets.detach())
```

Rubrik:

- 2 points for computing the n-step rewards (-1 point if the sum is normalized, -1 point if the rewards are not summed up)
- 2 points for computing the value of s^i_{t+n} (-1 point for computing the values for all states, no deduction for missing squeeze, no point deduction for `[:, self.n]`)
- 2 points for correct use of the terminals (-1 point if not negated)
- 2 points for the correct n-step targets (-1 point for forgetting/faulty γ^n)
- 2 points for detaching the n-step targets (-1 point for trying to use the index t)
- 2 points for the correct value of s^i_t (no deduction for missing squeeze), -1 point for computing the values
- 2 points the correct MSE loss (self-definition possible, -1 point for missing normalization). No points for for loop

End of exam. **Total 94 points.**