Exploration and uncertainty

Voluntary exercises

The following exercises do not have to be submitted as homework, but might be helpful to practice the required math and prepare for the exam. Some questions are from old exams and contain the used rubrik. You do not have to submit these questions and will not receive points for them.

E3.1: TD3 vs. DDPG (old exam question) **(voluntary)**

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

E3.2: TD3 vs. SAC (old exam question) **(voluntary)**

Explain in no more that 5 sentences at least **three** differences between TD3 and SAC.

Solution 1 point for each of the following points (up to 3 points total):

Rubrik:

- SAC does not regularize with clipped noise
- SAC learns stochastic policy with the reparameterization trick
- SAC adds maximum entropy term and/or automatic entropy adjustment
- SAC performs actions squashing for bounded action space
- SAC uses KL-divergence for policy update
- no point for "performs better/worse"

E3.3: Robust RL (old exam question) **(voluntary)**

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

E3.4: Different uncertainties (old exam question) **(voluntary)**

Define in 4 sentences or less the causes of *aleatoric* and *epistemic* uncertainty, and the difference between between.

Solution **Rubrik:**

- 1 point for aleatoric uncertainty comes from a stochastic environment and is irreducible
- 1 point for epistemic uncertainty comes from our knowledge of the environment and reduces to zero in the limit

E3.5: Bayesian inference **(voluntary)**

Bayesian inference is a very important part of machine learning. While this course does not directly uses it, we discuss posteriors a lot in the context of exploration. This exercise will give you a better understanding of how posteriors are defined and how they are used in Bayesian inference.

Let the data set $\mathcal{D} := \{y_t\}_{t=1}^n, y_t \in \mathbb{R}$, of labels be i.i.d. drawn from the Gaussian distribution $\rho(y_t) := \mathcal{N}(y_t|\mu_\rho, \sigma_\rho^2)$. In Bayesian inference we want to infer the true distribution ρ from observed data $\mathcal{D}$, i.e. find[1] $\mathbb{P}(\hat{y}|\mathcal{D})$. Note that we denote here $y \sim \rho(\cdot)$ and $\hat{y} \sim \mathbb{P}(\cdot|\mathcal{D})$, to avoid confusion in statements like $\mathbb{E}[y] \neq \mathbb{E}[\hat{y}|\mathcal{D}]$. To keep track over our belief, we define a (potentially infinite) set of hypotheses $\{\theta\}$, where θ refers to the parameters of a candidate *likelihood function* $\mathbb{P}(\hat{y}|\theta)$. Given a *prior* distribution over hypotheses $\mathbb{P}(\theta)$, we can use *Bayes theorem* to deduce the *posterior* distribution $p(\theta|\mathcal{D}) = \frac{\mathbb{P}(\mathcal{D}|\theta)\,\mathbb{P}(\theta)}{\mathbb{P}(\mathcal{D})} \propto \mathbb{P}(\mathcal{D}|\theta)\,\mathbb{P}(\theta)$ over hypotheses after we have seen the data. Finally, Bayesian inference uses this posterior for the best guess of the data distribution $\mathbb{P}(\hat{y}|\mathcal{D}) = \int \mathbb{P}(\hat{y}|\theta)\,\mathbb{P}(\theta|\mathcal{D})\,d\theta$.

[1] Note that $\mathbb{P}$ with different arguments denotes here different probability distributions.

In this exercise we will consider Gaussian likelihoods $\mathbb{P}(y|\theta) = \mathcal{N}(y|\theta, \sigma^2)$, with varying means θ but fixed variances σ^2, and the prior $\mathbb{P}(\theta) = \mathcal{N}(\theta|0, \sigma^2)$.

(a) Prove that $\mathbb{P}(\theta|\mathcal{D}) = \mathcal{N}\Big(\theta\Big|\frac{1}{n+1}\sum_{t=1}^{n} y_t, \frac{\sigma^2}{n+1}\Big)$.

Hint: (i) Gaussian PDFs are symmetric, i.e. $\mathcal{N}(a|b, \sigma^2) = \mathcal{N}(b|a, \sigma^2)$, and (ii) the product of two Gaussian PDFs is $\mathcal{N}(y|\mu, \sigma^2)\,\mathcal{N}(y|\mu', \sigma'^2) \propto \mathcal{N}\big(y\big|\frac{\sigma'^2\mu+\sigma^2\mu'}{\sigma^2+\sigma'^2}, (\frac{1}{\sigma^2}+\frac{1}{\sigma'^2})^{-1}\big)$.

(b) Using Bayesian inference, prove that $\mathbb{E}[\hat{y}|\mathcal{D}] = \frac{1}{n+1}\sum_{t=1}^{n} y_t$, and that $\mathbb{V}[\hat{y}|\mathcal{D}] = \frac{n+2}{n+1}\sigma^2$.

(c) Prove that the epistemic uncertainty from Question A3.5 in assignment sheet 3 is here $\mathbb{V}_{\mathcal{D}}\big[\mathbb{E}_{\hat{y}}[\hat{y}|\mathcal{D}]\big] = \frac{n}{(n+1)^2}\,\sigma_\rho^2$.

Note that both the posterior variance $\mathbb{V}[\theta|\mathcal{D}]$ and this definition of epistemic uncertainty $\mathbb{V}_{\mathcal{D}}[\mathbb{E}_{\hat{y}}[\hat{y}|\mathcal{D}]]$ (but not the predicted variance $\mathbb{V}_{\hat{y}}[\hat{y}|\mathcal{D}]$) shrink with roughly $\frac{1}{n+1}$. At least in this example one could therefore use the posterior variance as a (scaled) approximation of the epistemic uncertainty.

Solution

(a) We show by induction that $\mathbb{P}(\theta|\{y_t\}_{t=1}^m) \overset{ind}{=} \mathcal{N}(\theta|\frac{1}{m+1}\sum_{t=1}^m y_t, \frac{\sigma^2}{m+1})$. Induction start for $m = 1$:

$$\begin{aligned}\mathbb{P}(\{y_t\}_{t=1}^1|\theta) &\propto \mathbb{P}(\theta)\,\mathbb{P}(y_1|\theta) &&= \mathcal{N}(\theta|0,\sigma^2)\,\mathcal{N}(\theta|y_1,\sigma^2)\\ &\propto \mathcal{N}(\theta|\tfrac{\sigma^2 y_1+\sigma^2 0}{\sigma^2+\sigma^2}, [\tfrac{1}{\sigma^2}+\tfrac{1}{\sigma^2}]^{-1}) &&= \mathcal{N}(\theta|\tfrac{y_1}{2},\tfrac{\sigma^2}{2})\,. \qquad \checkmark\end{aligned}$$

Induction step for $m+1$, starting with Bayes' theorem (and remember the hints):

$$\begin{aligned}\mathbb{P}(\theta|\{y_t\}_{t=1}^{m+1}) &= \frac{\mathbb{P}(y_{m+1}|\theta)\,\mathbb{P}(\theta|\{y_t\}_{t=1}^m)}{\mathbb{P}(y_{m+1}|\{y_t\}_{t=1}^m)} \propto \mathbb{P}(y_{m+1}|\theta)\,\mathbb{P}(\theta|\{y_t\}_{t=1}^m)\\ \text{(symmetric)} &\overset{ind}{=} \mathcal{N}(\theta|y_{m+1},\sigma^2)\,\mathcal{N}\big(\theta|\tfrac{\sum_{t=1}^m y_t}{m+1},\tfrac{\sigma^2}{m+1}\big)\\ \text{(product)} &\propto \mathcal{N}\Big(\theta\Big|\frac{\frac{\sigma^2}{m+1}y_{m+1}+\sigma^2\frac{1}{m+1}\sum_{t=1}^m y_t}{\sigma^2+\frac{\sigma^2}{m+1}}, \big[\tfrac{1}{\sigma^2}+\tfrac{m+1}{\sigma^2}\big]^{-1}\Big)\\ &= \mathcal{N}\Big(\theta\Big|\frac{\frac{\sigma^2}{m+1}\sum_{t=1}^{m+1} y_t}{(m+2)\frac{\sigma^2}{m+1}}, \big[\tfrac{m+2}{\sigma^2}\big]^{-1}\Big)\\ &= \mathcal{N}\Big(\theta\Big|\tfrac{1}{m+2}\textstyle\sum_{t=1}^{m+1} y_t, \tfrac{\sigma^2}{m+2}\Big)\,. \qquad \checkmark\end{aligned}$$

Since both sides are properly normalized probability distributions, we have thus

$$\mathbb{P}(\theta|\{y_t\}_{t=1}^n) = \mathcal{N}\Big(\theta\Big|\tfrac{1}{n+1}\textstyle\sum_{t=1}^n y_t, \tfrac{\sigma^2}{n+1}\Big)\,.$$

(b) Using the above posterior, Bayesian inference has the following expectation:

$$\mathbb{E}[\hat{y}|\mathcal{D}] = \int \hat{y}\,\mathbb{P}(\hat{y}|\mathcal{D})\,d\hat{y} = \int \underbrace{\textstyle\int \hat{y}\,\mathbb{P}(\hat{y}|\theta)\,d\hat{y}}_{\theta}\,\mathbb{P}(\theta|\mathcal{D})\,d\theta = \mathbb{E}[\theta|\mathcal{D}] = \tfrac{1}{n+1}\sum_{t=1}^n y_t\,.$$

Similarly for the variance, where we abbreviate $\bar{y} := \mathbb{E}[\hat{y}|\mathcal{D}] = \mathbb{E}[\theta|\mathcal{D}]$:

$$\begin{aligned}\mathbb{V}[\hat{y}|\mathcal{D}] &= \int (\hat{y}-\bar{y})^2\,\mathbb{P}(\hat{y}|\mathcal{D})d\hat{y} = \iint (\hat{y}-\theta+\theta-\bar{y})^2\,\mathbb{P}(\hat{y}|\theta)\,d\hat{y}\,\mathbb{P}(\theta|\mathcal{D})\,d\theta\\ &= \int \Big(\underbrace{\textstyle\int(\hat{y}-\theta)^2\,\mathbb{P}(\hat{y}|\theta)\,d\hat{y}}_{\mathbb{V}[\hat{y}|\theta]=\sigma^2} + 2\underbrace{\textstyle\int(\hat{y}-\theta)\,\mathbb{P}(\hat{y}|\theta)\,d\hat{y}}_{\mathbb{E}[\hat{y}|\theta]-\theta=0}(\theta-\bar{y}) + \underbrace{\textstyle\int\mathbb{P}(\hat{y}|\theta)\,d\hat{y}}_{1}(\theta-\bar{y})^2\Big)\mathbb{P}(\theta|\mathcal{D})\,d\theta\\ &= \sigma^2 + \underbrace{\textstyle\int(\theta-\bar{y})^2\,\mathbb{P}(\theta|\mathcal{D})\,d\theta}_{\mathbb{V}[\theta|\mathcal{D}]=\frac{\sigma^2}{n+1}} = \tfrac{n+2}{n+1}\sigma^2\,.\end{aligned}$$

(c) It is easy to show that $\mathbb{E}\big[\mathbb{E}[\hat{y}|\mathcal{D}]\big|\mathcal{D} \sim \rho^n\big] = \frac{n}{n+1}\mu_\rho$. The rest of the proof runs analogue to Question (A3.5c) from assignment sheet 3, but to mix things up, we will use integral notation here. For the variance we have therefore:

$$\begin{aligned}
\mathbb{V}[\mathbb{E}[\hat{y}|\mathcal{D}]] &= \int \big(\mathbb{E}[\hat{y}|\mathcal{D}] - \tfrac{n}{n+1}\mu_\rho\big)^2 \rho^n(\mathcal{D})\, d\mathcal{D} = \int\cdots\int \Big(\tfrac{1}{n+1}\sum_{t=1}^{n}\big(y_t - \mu_\rho\big)\Big)^2 \prod_{t=1}^{n}\rho(y_t)\, dy_t \\
&= \tfrac{1}{(n+1)^2}\int\cdots\int \Big(\sum_{i=1}^{n}\sum_{j=1}^{n}(y_i-\mu_\rho)(y_j-\mu_\rho)\Big)\prod_{t=1}^{n}\rho(y_t)\, dy_t \\
&= \tfrac{1}{(n+1)^2}\sum_{i=1}^{n}\sum_{j=1}^{n}\overbrace{\Big(\int\cdots\int \prod_{\substack{t=1\\ t\neq i\\ t\neq j}}^{n}\rho(y_t)\, dy_t\Big)}^{1}\underbrace{\iint (y_i-\mu_\rho)(y_j-\mu_\rho)\,\rho(y_i)\,\rho(y_j)\, dy_i\, dy_j}_{\mathbb{E}[(y_i-\mu_\rho)(y_j-\mu_\rho)] = \sigma_\rho^2\,\delta(i=j)} \\
&= \tfrac{n}{(n+1)^2}\,\sigma_\rho^2\,.
\end{aligned}$$

E3.6: Bayes-by-backpropagation with an ensemble **(voluntary)**

In this exercise you will derive the Bayes-by-backpropagation objective for a posterior q_ϕ that is represented by an ensemble. As in Lecture 8.2, assume that the likelihood of the data $\mathcal{D}$ given neural-network parameters θ is proportional to the exponential of some given negative loss $\mathbb{P}(\mathcal{D}|\theta) \propto \exp(-\mathcal{L}_{[\theta]})$.

(a) Prove that $D_{\mathrm{KL}}\big(q_\phi(\cdot)\big\|\mathbb{P}(\cdot|\mathcal{D})\big) = \mathbb{E}[\mathcal{L}_{[\theta]}|\,\theta\sim q_\phi] + D_{\mathrm{KL}}\big(q_\phi(\cdot)\big\|\mathbb{P}(\cdot)\big) + c$, where $\mathbb{P}(\theta)$ denotes the prior over θ and c is some constant that does not depend on θ.

(b) Prove that for ensemble $\phi = \{\theta^k\}_{k=1}^m$, with posterior $q_\phi(\theta) = \frac{1}{m}\sum_{k=1}^m \delta(\theta=\theta^k)$, Bayes-by-backpropagation with a Gaussian prior $\mathbb{P}(\theta) = \mathcal{N}(\theta|\mathbf{0},\sigma^2\mathbf{I})$ is equivalent to minimizing the L_2 regularized loss for each ensemble member individually, that is:

$$\nabla_{\theta^i} D_{\mathrm{KL}}\big(q_\phi(\cdot)\big\|\mathbb{P}(\cdot|\mathcal{D})\big) = \tfrac{1}{m}\nabla_{\theta^i}\mathcal{L}_{[\theta^i]} + \tfrac{1}{m}\tfrac{1}{2\sigma^2}\nabla_{\theta^i}\|\theta^i\|_2^2\,, \qquad \forall i\in\{1,\dots,m\}\,.$$

Solution

(a) Here we use the definition of the KL-divergence, Bayes' theorem, the definition of the likelihood $\mathbb{P}(\mathcal{D}|\theta)$ and the properties of the logarithm.

$$\begin{aligned}
D_{\mathrm{KL}}\big(q_\phi(\cdot)\big\|\mathbb{P}(\cdot|\mathcal{D})\big) &= \int q_\phi(\theta)\ln\tfrac{q_\phi(\theta)}{\mathbb{P}(\theta|\mathcal{D})}\, d\theta = \int q_\phi(\theta)\ln\tfrac{q_\phi(\theta)\,\mathbb{P}(\mathcal{D})}{\mathbb{P}(\mathcal{D}|\theta)\,\mathbb{P}(\theta)}\, d\theta \\
&= \ln\mathbb{P}(\mathcal{D})\underbrace{\int q_\phi(\theta)\, d\theta}_{1} - \int q_\phi(\theta)\ln\mathbb{P}(\mathcal{D}|\theta)\, d\theta + D_{\mathrm{KL}}\big(q_\phi(\cdot)\big\|\mathbb{P}(\cdot)\big) \\
&= \underbrace{\ln\mathbb{P}(\mathcal{D})}_{c} + \int q_\phi(\theta)\,\mathcal{L}_{[\theta]}\, d\theta + D_{\mathrm{KL}}\big(q_\phi(\cdot)\big\|\mathbb{P}(\cdot)\big) \\
&= c + \mathbb{E}[\mathcal{L}_{[\theta]}|\theta\sim q_\phi] + D_{\mathrm{KL}}\big(q_\phi(\cdot)\big\|\mathbb{P}(\cdot)\big)
\end{aligned}$$

(b) We need to show that the gradients for the above Bayes-by-backpropagation loss for each ensemble member's model parameters $\theta^i \in \mathbb{R}^d$ are equivalent to the member's loss plus some L_2 regularization term. The trick is that, by construction of the posterior, the integral

$\int q_\phi(\theta)\, f(\theta)\, d\theta = \frac{1}{m}\sum_{k=1}^{m} f(\theta^k), \forall f$, and that the ensemble members have independent parameters, i.e., $\nabla_{\theta^i} \frac{1}{m}\sum_{k=1}^{m} f(\theta^k) = \frac{1}{m}\nabla_{\theta^i} f(\theta^i)$.

$$
\begin{aligned}
\nabla_{\theta^i} D_{\mathrm{KL}}\big(q_\phi(\cdot)\big\|\mathbb{P}(\cdot|\mathcal{D})\big) &= \nabla_{\theta^i}\mathbb{E}[\mathcal{L}_{[\theta]}|\theta\sim q_\phi] + \nabla_{\theta^i} D_{\mathrm{KL}}\big(q_\phi(\cdot)\big\|\mathbb{P}(\cdot)\big) \\
&= \nabla_{\theta^i}\frac{1}{m}\sum_{k=1}^{m}\mathcal{L}_{[\theta^k]} + \nabla_{\theta^i}\frac{1}{m}\sum_{k=1}^{m}\ln\frac{\sum_{j=1}^{m}\delta(\theta^k=\theta^j)}{m\,\mathcal{N}(\theta^k|\mathbf{0},\sigma^2\mathbf{I})} \\
&= \tfrac{1}{m}\nabla_{\theta^i}\mathcal{L}_{[\theta^i]} + \underbrace{\nabla_{\theta^i}\ln(1)}_{0} - \underbrace{\nabla_{\theta^i}\ln(m)}_{0} - \nabla_{\theta^i}\tfrac{1}{m}\sum_{k=1}^{m}\ln(\underbrace{\mathcal{N}(\theta^k|\mathbf{0},\sigma^2\mathbf{I})}_{(2\pi)^{-\frac{d}{2}}\sigma^{-1}\exp\left(-\frac{1}{2\sigma^2}\|\theta^k\|_2^2\right)}) \\
&= \tfrac{1}{m}\nabla_{\theta^i}\mathcal{L}_{[\theta^i]} + \tfrac{1}{m}\tfrac{1}{2\sigma^2}\nabla_{\theta^i}\|\theta^i\|_2^2 - \underbrace{\nabla_{\theta^i}\ln\left((2\pi)^{-\frac{d}{2}}\sigma^{-1}\right)}_{0}
\end{aligned}
$$

E3.7: Exploration with ϵ-greedy (old exam question) **(voluntary)**

Describe in 4 sentences or less at least **two** different things that can hurt learning when exploring with ϵ-greedy.

Solution 1 point for each of the following arguments (up to 2 points):

Rubrik:

- large state spaces (due to exponential decay of random exploration)
- nearby sub-optimal rewards that outshine far away optimal rewards
- rewarded action-sequences are rare under random exploration (e.g. due to adversarial dynamics)
- wrong hyper-parameters can explore too long or not long enough

E3.8: Exploration environments (old exam question) **(voluntary)**

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.

E3.9: Thompson vs. optimisitc exploration (old exam question) **(voluntary)**

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.

E3.10: Implement DDPG value loss (old exam question) **(voluntary)**

In the exam you must be able to solve simple programming questions *without a computer*. Try to solve this one on paper, by writing the missing code (`YOUR CODE HERE`), to prepare yourself for the exam!

Implement the following *value loss* for the DDPG algorithm in the given `MyLearner` class **efficiently**:

$$\min_{\phi} \quad \mathbb{E}\left[\frac{1}{\sum_{i=1}^{m} n_i}\sum_{i=1}^{m}\sum_{t=0}^{n_i-1}\left(r_t^i + \gamma Q_{\phi'}\left(s_{t+1}^i, \boldsymbol{\pi}_\theta(s_{t+1}^i)\right) - Q_\phi(s_t^i, a_t^i)\right)^2 \,\middle|\, \tau_{n_i}^i \sim \mathcal{D}\right],$$

where $\tau_{n_i}^i := \{s_t^i, a_t^i, r_t^i\}_{t=0}^{n_i-1} \cup \{s_{n_i}^i\}$, denotes m trajectories of states $s_t^i \in \mathbb{R}^d$, actions $a_t^i \in \mathbb{R}^b$ and rewards $r_t^i \in \mathbb{R}$. The last state $s_{n_i}^i$ is always terminal. The target network parameters shall be $\phi' := \phi$ at all times, but no gradients shall flow into ϕ'!

Hint: Efficient implementations produce minimal computation graphs. You can use the given function `self.values()` to compute the Q-values $Q_\phi(s, a)$ for a mini-batch (tensor) of states s_t^i and a mini-batch (tensor) of the corresponding actions a_t^i, with the same size (except for the last dimension). The deterministic policy $\boldsymbol{\pi}_\theta$ is computed by another given module `policy` that takes a mini-batch (tensor) of states s_t^i as input and outputs an equally sized (except in the last dimension) tensor of actions $a_t'^i = \boldsymbol{\pi}_\theta(s_t^i)$.

```python
import torch

class MyLearner:
    def __init__(self, model, policy, gamma=0.99):
        self.value_model = model
        self.policy_model = policy
        self.gamma = gamma
        self.all_parameters = [p for m in self.value_models for p in m.parameters()]
        self.optimizer = torch.optim.Adam(self.all_parameters)

    def values(self, states, actions):
```

```
12          return self.value_model(torch.cat([states, actions], dim=-1))
13
14      def train(self, batch):
15          """ Performs one gradient update step on the loss defined above.
16              "batch" is a dictionary of equally sized tensors
17              (except for last dimension):
18                  - batch['states'][i, t, :] = s_t^i
19                  - batch['actions'][i, t, :] = a_t^i
20                  - batch['rewards'][i, t] = r_t^i
21                  - batch['mask'][i, t] = t < n_i
22                  - batch['terminals'][i, t] = s_t^i is terminal """
23          loss = 0
24          # YOUR CODE HERE
25          self.optimizer.zero_grad()
26          loss.backward()
27          self.optimizer.step()
28          return loss.item()
```

Solution

```
1 val = self.values(batch['states'], batch['actions'])
2 pol = self.policy_model(batch['states'])
3 tar = ~batch['terminals'] * self.values(batch['states'], pol)
4 tar = batch['rewards'][:, :-1] + self.gamma * tar[:, 1:]
5 td = (tar.detach() - val[:, :-1]) ** 2
6 mask = batch['mask'][:, :-1]
7 loss = (td * mask).sum() / mask.sum()
```

Rubrik:

- 1 point for computing actions of the policy in one call (not for loops!)
- 1 point for computing the target values in one call
- 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
- 1 point for correct timing in other fields (`batch['rewards']`, `batch['terminals']`, `batch['mask']` and `val`)
- 1 point for using `batch['terminals']` correctly
- 1 point for correct TD-error, including correct detaching
- 1 point for masking with `batch['mask']` and normalizing w.r.t. the mask
- Ignore small Python/PyTorch errors

E3.11: Implement DDPG policy loss (old exam question) **(voluntary)**

In the exam you must be able to solve simple programming questions *without a computer*. Try to solve this one on paper, by writing the missing code (`YOUR CODE HERE`), to prepare yourself for the exam!

Implement the following DDPG policy objective efficiently in the given `MyLearner` class **efficiently**:

$$\max_{\theta} \quad \mathbb{E}\left[\frac{1}{\sum_{i=1}^{m} n_i} \sum_{i=1}^{m} \sum_{t=0}^{n_i-1} \gamma^t Q_\phi\big(s_t^i, \boldsymbol{\pi}_\theta(s_t^i)\big) \,\middle|\, \tau_{n_i}^i \sim \mathcal{D}\right],$$

where $\tau_{n_i}^i := \{s_t^i, a_t^i\}_{t=0}^{n_i-1}$ is a history of states $s_t^i \in \mathbb{R}^d$ and actions $a_t^i \in \mathbb{R}^b$ 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).

```python
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_t^i
                - batch['actions'][i, t, :] = a_t^i
                - 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

```python
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