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. Solutions are available on Brightspace.

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

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

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

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

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?

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.

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

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

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 $\mathbb{P}(\hat{y}|\mathcal{D}) = \int \mathbb{P}(\hat{y}|\theta)\,\mathbb{P}(\theta|\mathcal{D})\,d\theta$.

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.

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:

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

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.

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.

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

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

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**:

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

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):
        return self.value_model(torch.cat([states, actions], dim=-1))

    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['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()
```

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**:

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

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 $\boldsymbol{\pi}_\theta$ 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()
```