Deep reinforcement learning

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.

E2.1: Variance of a value (old exam question) (voluntary)

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

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

E2.2: Values for policy gradient (old exam question) (voluntary)

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

E2.3: Expectation of a Markov chain (voluntary)

Given a Markov chain with 2 transitions (from s_0 to s_2), show analytically that $\mathbb{E}_\pi[f(s_1)] = \int \xi_1^\pi(s)\, f(s)\, ds$, where ξ_1^π is the state distribution after one step (from the lecture slides). How exactly is $\xi_1^\pi(s)$ defined in this special case?

E2.4: Belief-MDP of sufficient POMDP statistics (old exam question) (voluntary)

Given a POMDP $M := \langle \mathcal{S}, \mathcal{A}, \mathcal{O}, \rho, P, R, O \rangle$, and a belief distribution $b(s|\tau_t)$, with τ_t denoting the observation-action history at time t, define the MDP $M' := \langle \mathcal{S}', \mathcal{A}', \rho', P', r' \rangle$ over the sufficient statistics of the belief b, where r' is the average reward function of M' (the full reward distribution R' can be omitted). Make sure you define *all* the components of M'.

Hint: the MDP M' is defined in terms of observation-action histories, as in DRQN.

E2.5: Expected loss of a random function (old exam question) (voluntary)

Let $f : \mathbb{R} \to \mathbb{R}$ denote a random function, where the output $f(x) \sim \mathcal{N}(\mu(x), \sigma^2)$ is drawn i.i.d. for each input $x \in \mathbb{R}$. Prove analytically that the expected *mean squared error* $\mathbb{E}[\mathcal{L}^{\text{mse}}]$ for a given data-set $\{x_i, y_i\}_{i=1}^n$ is:

$$\mathbb{E}[\mathcal{L}^{\text{mse}}] \quad = \quad \tfrac{1}{n}\sum_{i=1}^{n}\big(\mu(x_i) - y_i\big)^2 + \sigma^2$$

Note that for each index i, the data tuple $\langle x_i, y_i \rangle$ is fixed, whereas the function output $f(x_i)$ is random!

E2.6: Entropy of a Gaussian (voluntary)

A univariate Gaussian distribution p is defined as $p(x) := \mathcal{N}(x|\mu, \sigma^2) := \frac{1}{\sqrt{2\pi\sigma^2}} \exp\big(-\frac{1}{2\sigma^2}(x-\mu)^2\big)$.

(a) Show analytically that the entropy of p is
$\mathcal{H}[p] := -\int p(x) \, \ln p(x) \, dx = \frac{1}{2}\big(\ln(2\pi\sigma^2) + 1\big)$.

(b) Show analytically that the derivative of the entropy w.r.t. σ is $\frac{\partial}{\partial\sigma}\mathcal{H}[p] = \frac{1}{\sigma}$.

E2.7: Implement another 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 L_1 loss to learn the state-value v_θ in the given `MyLearner` class **efficiently**:

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

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()
```