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.

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

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

<u>**Solution**</u> The major insight here is that $\mathbb{E}[r_t] = \mu$, that $\mathbb{E}[(r_i - \mu)(r_j - \mu)] = (\mathbb{E}[r_i] - \mu)(\mathbb{E}[r_j] - \mu) = 0$, $\forall i \neq j$, and that $\mathbb{E}[(r_i - \mu)^2] = \sigma^2$. We also need the geometric series $\sum_{t=0}^{\infty} \gamma^t = \frac{1}{1-\gamma}$.

$$\begin{aligned}
\mathbb{E}[v] &= \sum_{t=0}^{\infty} \gamma^t \mathbb{E}[r_t] \quad = \quad \sum_{t=0}^{\infty} \gamma^t \mu \\
\mathbb{V}[v] &= \mathbb{E}\Big[\Big(\sum_{t=0}^{\infty} \gamma^t r_t - \mathbb{E}[v]\Big)^2\Big] \quad = \quad \mathbb{E}\Big[\Big(\sum_{t=0}^{\infty} \gamma^t (r_t - \mu)\Big)^2\Big] \\
&= \sum_{i=0}^{\infty}\sum_{j=0}^{\infty} \gamma^{i+j} \underbrace{\mathbb{E}\big[(r_i - \mu)(r_j - \mu)\big]}_{\sigma^2\,\delta(i=j)} \quad = \quad \sigma^2 \sum_{i=0}^{\infty} (\gamma^2)^i \quad = \quad \frac{\sigma^2}{1-\gamma^2}
\end{aligned}$$

Rubrik:

- 1 point for the correct definition of variance $\mathbb{V}$
- 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

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.

Solution 1 Point for any of the following (up to 2 points). **Rubrik:**

- Actor-critic/Off-PAC/TRPO/PPO: reduce variance with bias
- Actor-critic/Off-PAC/TRPO/PPO: replace rollouts with bootstrapping
- DDPG/TD3/SAC: optimize policy for estimated Q-value

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?

Solution One only needs to write the expectation as sequence of integrals, and remember that, like sums, one can exchange the order of integrals arbitrarily:

$$\begin{aligned}
\mathbb{E}_\pi[f(s_1)] &= \int \rho(s_0) \int \pi(a_0|s_0) \int P(s_1|s_0,a_0) f(s_1) \underbrace{\int \pi(a_1|s_1) \overbrace{\int P(s_2|s_1,a_1)\, ds_2}^{1}\, da_1}_{1}\, ds_1\, da_0\, ds_0 \\
&= \int \overbrace{\iint \rho(s_0)\, \pi(a_0|s_0)\, P(s_1|s_0,a_0)\, ds_0\, da_0}^{\xi_1^\pi(s_1)}\, f(s_1)\, ds_1 \,.
\end{aligned}$$

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.

Solution Following the lecture's notation, we have $\mathcal{S}' := (\mathcal{O} \times \mathcal{A})^* \times \mathcal{O}$, $\mathcal{A}' := \mathcal{A}$, and:

$$\begin{aligned}
\rho'(\tau_0) &:= \int O(o_0|s)\, \rho(s)\, ds \overset{\text{or}}{=} \int O(o_0|s)\, b(s|\tau_0)\, ds \\
r'(\tau_t, a) &:= \iint r\, R(r|s,a,s')\, P(s'|s,a)\, b(s|\tau_t)\, ds\, ds'\, dr \\
P'(\tau_{t+1}|\tau_t, a_t) &:= \iint O(o_{t+1}|s')\, P(s'|s,a_t)\, b(s|\tau_t)\, ds\, ds' \\
\text{or} \quad P'(\tau_{t+1}|\tau_t, a) &:= \delta(a = a_t) \iint O(o_{t+1}|s')\, P(s'|s,a)\, b(s|\tau_t)\, ds\, ds'
\end{aligned}$$

Rubrik:

- 1 point for the definition of $\mathcal{S}'$ and $\mathcal{A}'$.
- 1 point for *one* of the definitions of $\rho'(\tau_0)$.

- 1 point for the definition of $r'(\tau_t, a)$. Only $\frac{1}{2}$ point for $r'(\tau_t)$.
- 1 point for *either* the definitions of $P'(\tau_{t+1}|\tau_t, a_t)$ or $P'(\tau_{t+1}|\tau_t, a)$.
- Arguing that any of these terms are not explicitly defined in the question is not permissive.

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!

Solution Let $\mu_i := \mu(x_i), 1 \leq i \leq n$. The main insight is that $\mathbb{E}[f(x_i) - \mu_i] = 0$ and that $\mathbb{E}[(f(x_i) - \mu_i)^2] = \sigma^2$.

$$\begin{aligned}
\mathbb{E}[\mathcal{L}^{\text{mse}}] \quad &:= \quad \mathbb{E}\Big[\tfrac{1}{n}\sum_{i=1}^n \big(f(x_i) - y_i\big)^2\Big] \quad = \quad \tfrac{1}{n}\sum_{i=1}^n \mathbb{E}\Big[\Big(\big(f(x_i) - \mu_i\big) - \big(y_i - \mu_i\big)\Big)^2\Big] \\
&= \quad \tfrac{1}{n}\sum_{i=1}^n \Big(\underbrace{\mathbb{E}\big[\big(f(x_i) - \mu_i\big)^2\big]}_{\sigma^2} - 2\,\underbrace{\mathbb{E}\big[f(x_i) - \mu_i\big]}_{0}\,\big(y_i - \mu_i\big) + \big(y_i - \mu_i\big)^2 \Big) \\
&= \quad \sigma^2 + \tfrac{1}{n}\sum_{i=1}^n (y_i - \mu_i)^2 \,.
\end{aligned}$$

Rubrik:

- 1 point for the correct defintion of the expected mean suqared error (only $\frac{1}{2}$ for minor deviations)
- 1 point for the correct use $\mathbb{E}[f(x_i)] = \mu_i$ (only $\frac{1}{2}$ point for definition, but incorrect use and vice versa)
- 1 point for the correct use of the varianvce σ^2 of $f(x_i)$ (only $\frac{1}{2}$ point for definition, but incorrect use and vice versa)
- 1 points for putting it correctly together

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 $\frac{\partial}{\partial\sigma}\mathcal{H}[p] = \frac{1}{\sigma}$.

Solution

(a) Here the important insight is the definition of variance $\sigma^2 = \int p(x)(x-\mu)^2 dx$. We also use properties of the logarithm: $\ln(xy) = \ln x + \ln y$, $\ln \frac{1}{x} = -\ln x$, $\ln x^y = y \ln x$ and $\ln \exp(x) = x$.

$$\begin{aligned} \mathcal{H}[p] &= -\int p(x) \ln p(x)\, dx \quad = \quad -\int p(x) \Big(\ln \tfrac{1}{\sqrt{2\pi\sigma^2}} + \ln \exp\big(-\tfrac{1}{2\sigma^2}(x-\mu)^2\big)\Big) dx \\ &= \ln \sqrt{2\pi\sigma^2} \underbrace{\int p(x)\, dx}_{1} + \tfrac{1}{2\sigma^2} \underbrace{\int p(x)\,(x-\mu)^2\, dx}_{\sigma^2} \quad = \quad \tfrac{1}{2}\ln(2\pi\sigma^2) + \tfrac{1}{2}\,. \end{aligned}$$

(b) $\frac{\partial}{\partial\sigma}\mathcal{H}[p] \quad = \quad \frac{\partial}{\partial\sigma}\frac{1}{2}\ln(2\pi\sigma^2) \quad = \quad \frac{1}{2}\frac{1}{2\pi\sigma^2}2\pi\frac{\partial}{\partial\sigma}\sigma^2 \quad = \quad \frac{1}{2}\frac{1}{\sigma^2}2\sigma \quad = \quad \frac{1}{\sigma}\,.$

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**:

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

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()
```

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]).abs()
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]`)