Artificial Neural Networks (Gerstner). Exercises for week 8

Deep Reinforcement Learning

Exercise 1. Uncorrelated mini-batches in A2C.

In the lecture you have seen a simple example of a single weight w that changes with temporally correlated updates Δw (red dots and red curve on slide 6). Reshuffling the updates in time led to a more stable learning dynamics (blue dots and blue curve). This example illustrates the effect of sampling iid from the replay buffer for off-policy methods like DQN. For on-policy methods, the proposed solution is to run multiple actors in parallel.

a. Sketch a figure similar to the one on slide 6 for 4 parallel actors and 2 to 3 episodes per actor. Mark the starts of new episodes for each actor with vertical lines.
Hint: the episodes can have different lengths.

b. Draw in the same figure approximately the values $\frac{1}{4}\sum_{k=1}^{4}\Delta w^{(k)}$ for all time points.

c. Write a caption to the figure that explains, why the proposed solution of A2C helps to stabilize learning.

Exercise 2. Importance sampling.

Let us assume we would like to evaluate a policy $\pi(a|s)$, but we can only obtain episodes

$$(S_0, A_0, R_1, S_1, \ldots, S_{T-1}, A_{T-1}, R_T, S_T)$$

with policy $b(a|s)$. We will use importance weights C_t to correct for the mismatch between the two policies, i.e. we will compute

$$\tilde{V}_\gamma^{(T)}(b,s) := \mathbb{E}_b\left[\sum_{t=1}^{T}\gamma^{t-1}C_tR_t\Big|S_0=s\right]$$

where the expectation is taken over actions sampled from policy b. How should the importance weights C_t be chosen to have $V_\gamma^{(T)}(\pi,s)=\tilde{V}_\gamma^{(T)}(b,s)$?

Hint: Importance weights are themselves random variable, i.e., they depends on $(S_0, A_0, R_1, \ldots)$.

Exercise 3. Q–learning with function approximation

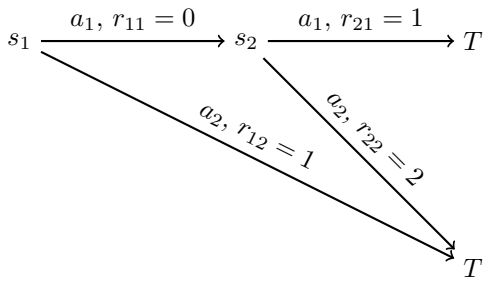

Figure 1: MDP in Exercise 3

Consider the MDP shown in Figure 1, with two states, two actions and deterministic rewards (where T represents the terminal state). We want to learn the Q–values associated with the states using Q–learning, with discount factor $\gamma = 1$.

a. (**Tabular Case**) The agent starts with all Q–values equal to 0. We assume that the agent can store observed transitions in memory (similar to a replay buffer). The agent observes all 4 possible transitions, then updates the Q–values for s_2 by alternating between observations of (s_2, a_1, r_{21}) and (s_2, a_2, r_{22}) until learning converges. The agent then similarly alternates between observations of (s_1, a_1, r_{11}) and (s_1, a_2, r_{12}) until learning converges.

(i) What are the Q–values after convergence in s_2, and finally after convergence in s_1?

(ii) Do the Q–values after each stage result in the optimal policy?

b. (**Function Approximation**) Now assume that the states are given to us with the vector–based observations shown in Figure 2 left. We will learn the Q–values using the linear network shown in Figure 2 right.

As before, assume a replay-buffer-style learning where the agent learns the weights after observing all transitions. Start with $w_{11} = w_{12} = w_{13} = w_{21} = w_{22} = w_{23} = 0$.

(i) What will the converged weights be after alternating between the two possible s_2 observations?
Hint: Note that certain weights will always be updated in exactly the same way and should, therefore, converge to the same value.

(ii) After s_2 convergence, what is the policy in s_1? How does this differ from the tabular case after s_2 convergence, and why?

(iii) What weights would result in the correct Q–value predictions for all (s, a) pairs? Are they unique?

(iv) How can an arbitrary tabular Q–learning problem be represented using a simple linear neural network like the one shown on the right?
Hint: consider how the input space could be represented such that semi–gradient descent results in each weight converging exactly to $Q(s, a)$ for some (s, a) pair.

$$s_1 = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix}$$

$$s_2 = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix}$$

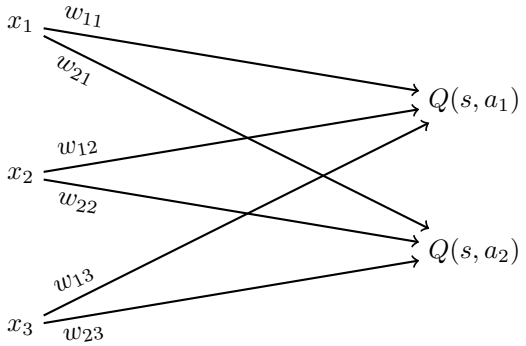

Figure 2: Neural network for function approximation in Exercise 3

Exercise 4. Proximal Policy Optimization.

a. In the derivation of Proximal Policy Optimization methods the ratio $r_{\theta'}(s_t, a_t) = \frac{\pi_{\theta'}(a_t; s_t)}{\pi_\theta(a_t; s_t)}$ appeared on the last line on slide 22. Convince yourself that the equality on the last line of slide 22 is correct by explicitly writing out the expectations in the same way as we did on slide 21.
Hint: Write something like $E_{s_t, a_t \sim p_{\theta'}, \pi_{\theta'}} \left[\sum_{t=0}^{\infty} \gamma^t A_\theta(s_t, a_t)\right] = \sum \ldots = \sum \ldots = E_{s_t, a_t \sim p_{\theta'}, \pi_\theta} \ldots$

b. Show that the summands in the loss function of PPO-CLIP can also be written in the form

$$\ell(r_{\theta'}) = \min(r_{\theta'} \gamma^t A_\theta, g(\epsilon, \gamma^t A_\theta)), \quad \text{with} \quad g(\epsilon, A) = \begin{cases} (1+\epsilon)A & A \geq 0 \\ (1-\epsilon)A & A < 0 \end{cases}$$

c. Sketch $\ell(r)$ as a function of r in two figures: one where A_θ is positive and one where A_θ is negative.

d. Write a caption to your figures that explains, why one can safely run a few steps of gradient ascent on $\hat{L}^{\text{CLIP}}(\theta')$ without risking that $\pi_{\theta'}$ would move too far away from π_θ.

Exercise 5. Deep Deterministic Policy Gradient.

a. How many input and output neurons does the Q-network of DQN have, if the input consists of 100-dimensional vectors and there are 10 possible actions?

b. How many input and output neurons does the Q-network of DDPG have, if the input consists of 100-dimensional vectors and the action space is 10 dimensional?

c. Explain, why it would not be a good idea to use $\hat{Q}(s_{j+1}, a_{j+1})$ in line 7 of the DDPG algorithm (slide 20).

d. Explain, why it would not be a good idea to use $\hat{Q}(s_{j+1}, \hat{\pi}(s_{j+1}) + \epsilon)$ in line 7 of the DDPG algorithm.