Artificial Neural Networks (Gerstner). Solutions for week 7

Deep Reinforcement Learning

Exercise 1. Uncorrelated mini-batches in A2C.

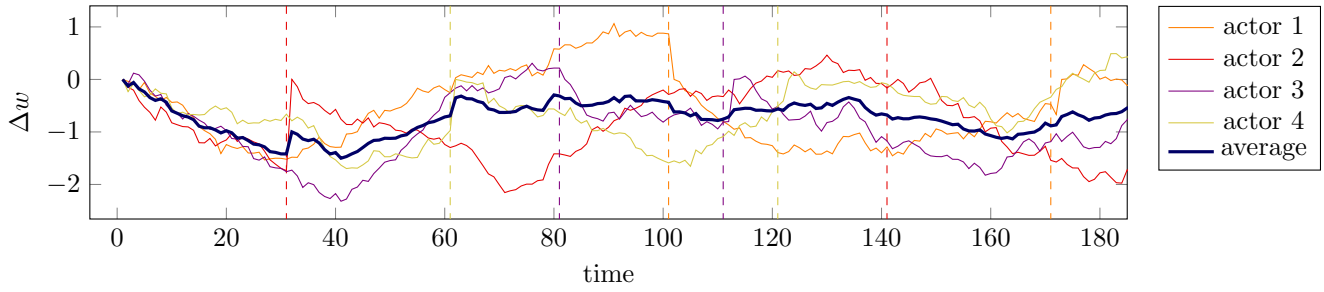
In the lecture you have seen a simple example of a single weight w that changes with temporally correlated updates Δw (red dots and red curve on slide 6). Reshuffling the updates in time led to a more stable learning dynamics (blue dots and blue curve). This example illustrates the effect of sampling 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.

Solution:



In this example the updates Δw tend to become more and more negative at the beginning of each episode while they tend to grow again later during each episode. If the episodes of all agents start at the same time, this is also reflected in the average weight update (see first 60 time steps), even though averaging helps already to avoid the most extreme updates. After time step 60, the average weight update is more balanced, because the different agents are in different parts of their episode.

Exercise 2. Q–learning with function approximation

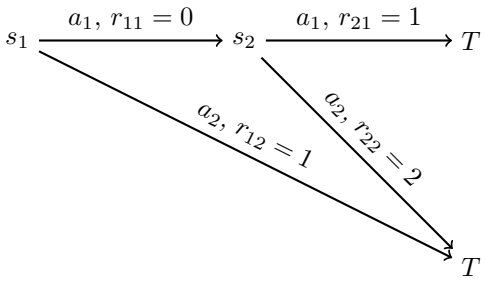


Figure 1: MDP in Exercise 2

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.

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

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

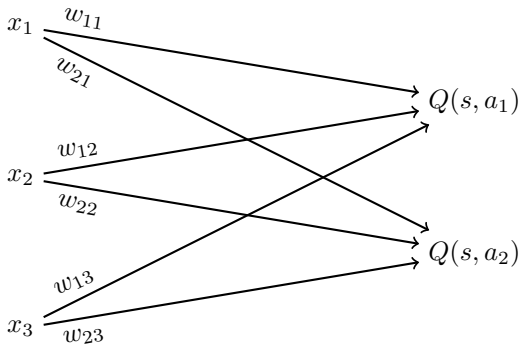


Figure 2: Neural network for function approximation in Exercise 2

Solution:

a. (i)

$$\begin{bmatrix} Q(s_1, a_1) & Q(s_1, a_2) \\ Q(s_2, a_1) & Q(s_2, a_2) \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \quad \text{(Start)}$$

$$\begin{bmatrix} 0 & 0 \\ 1 & 2 \end{bmatrix} \quad \text{(After } s_2 \text{ convergence)}$$

$$\begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix} \quad \text{(After } s_1 \text{ convergence)}$$

(ii) With a discount factor of 1, the optimal policy is to take a_1 in s_1 and a_2 in s_2, achieving a reward of 2. After s_2 convergence, the Q–values in s_1 are equal, so they do not represent an optimal policy. The final Q–values do represent the optimal policy.

b. (i) Noting that the updates should result in $w_{12} = w_{13}$ and $w_{22} = w_{23}$, we can expect the weights to converge to

$$\begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad \text{(Start)}$$

$$\begin{bmatrix} 0 & 0.5 & 0.5 \\ 0 & 1 & 1 \end{bmatrix} \quad \text{(After } s_2 \text{ convergence)}$$

(ii) Unlike the tabular case, the Q–values for s_1 are now $Q(s_1, a_1) = 0.5$ and $Q(s_1, a_2) = 1$, resulting in a policy that does not give equivalent weight to the two actions (and in this case is suboptimal). This occurs because $x_2 = 1$ for both states, so their updates are correlated.

(iii) One possible solution is to fix $w_{12} = w_{22} = 0$. In this case, the value of x_2 is irrelevant and the state representations become effectively orthogonal. The optimal weights are given by

$$\begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \end{bmatrix} = \begin{bmatrix} 2 & 0 & 1 \\ 1 & 0 & 2 \end{bmatrix}$$

which we can see as equivalent to the tabular solution given above.
In general, we can notice that the weights for $Q(s, a_2)$ can be solved without using the values of any other states, and amount to solving a system of 2 equations with 3 unknowns. Since this solution will not be unique, the optimal weights are not unique either.

(iv) The semi–gradient descent rule for the network above gives the weight update

$$\Delta w_{ji} = \alpha\Big(r_{t+1} + \gamma \max_{a_i} Q(s_{t+1}, a_i) - Q(s_t, a_t)\Big) \frac{\partial Q(s_t, a_t)}{\partial w_{ji}}$$

applied only for the observed action $a_t = j$. Note that this is equivalent to tabular Q–learning with $w_{ji} = Q(i, j)$ if

$$\frac{\partial Q(s_t, a_t)}{\partial w_{ji}} = \begin{cases} 1, & \text{if } s_t = i \text{ and } a_t = j \\ 0, & \text{otherwise.} \end{cases}$$

Since $\frac{\partial Q(s_t, a_t)}{\partial w_{ji}} = x_i$, this can be enforced by using a one–hot input representation with N inputs for N states.

Exercise 3. 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.

Solution:

a. Using an expanded form of the expectation similar to the one in slide 21, we have

$$\begin{aligned} E_{s_t, a_t \sim p_{\theta'}, \pi_{\theta'}}\Big[\sum_{t=0}^{\infty} \gamma^t A_\theta(s_t, a_t)\Big] &= \sum_{t=0}^{\infty} E_{s_t, a_t \sim p_{\theta'}, \pi_{\theta'}}\big[\gamma^t A_\theta(s_t, a_t)\big] \\ &= \sum_{t=0}^{\infty} \sum_{s_t, a_t} \gamma^t A_\theta(s_t, a_t) \pi_{\theta'}(a_t; s_t) p_{\theta'}(s_t) \\ &= \sum_{t=0}^{\infty} \sum_{s_t, a_t} \gamma^t A_\theta(s_t, a_t) \frac{\pi_{\theta'}(a_t; s_t)}{\pi_\theta(a_t; s_t)} \pi_\theta(a_t; s_t) p_{\theta'}(s_t) \qquad (1) \\ &= \sum_{t=0}^{\infty} \sum_{s_t, a_t} \gamma^t A_\theta(s_t, a_t) r_{\theta'}(a_t; s_t) \pi_\theta(a_t; s_t) p_{\theta'}(s_t) \\ &= \sum_{t=0}^{\infty} E_{s_t, a_t \sim p_{\theta'}, \pi_\theta}\big[\gamma^t A_\theta(s_t, a_t) r_{\theta'}(a_t; s_t)\big]. \end{aligned}$$

b. The goal is to show the identity

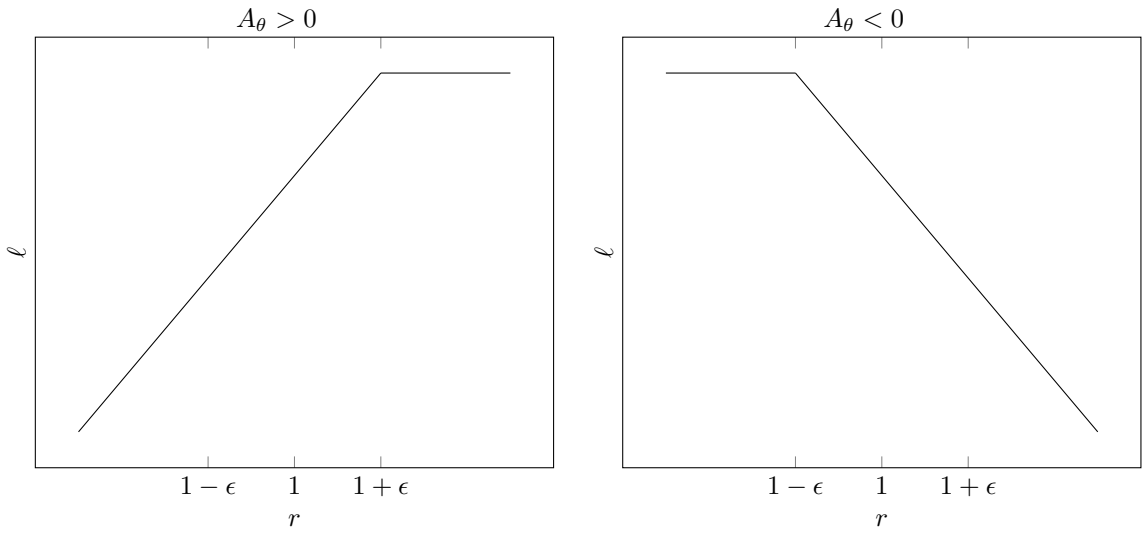
$$\ell_1(r, A) = \ell_2(r, A), \tag{2}$$

where

$$\begin{aligned} \ell_1(r, A) &= \min(rA, \text{clip}\,(r, 1-\epsilon, 1+\epsilon)A), \\ \ell_2(r, A) &= \min(rA, g(\epsilon, A)). \end{aligned} \tag{3}$$

To do so, we show that for different values of r and A, two functions are equal:
Case 1. When $1 - \epsilon \leq r \leq 1 + \epsilon$: For this case, r is not clipped for ℓ_1, and we have $\ell_1 = rA$. Then, if $A \geq 0$, we have $\ell_2(r, A) = \min(r, 1 + \epsilon)A = rA$, and if $A < 0$, we have $\ell_2(r, A) = \min(rA, 1 - \epsilon A) = \max(r, (1 - \epsilon))A = rA$.
Case 2. When $1 + \epsilon < r$: Then we have $\ell_1(r, A) = \min(rA, (1 + \epsilon)A)$, which is equal to $(1 + \epsilon)A$ if $A \geq 0$ and is equal to rA if $A < 0$. At the same time, when $A \geq 0$, we have $\ell_2(r, A) = \min(r, 1+\epsilon)A = (1+\epsilon)A$, and when $A < 0$, we have $\ell_2(r, A) = \min(rA, (1 - \epsilon)A) = \max(r, 1 - \epsilon)A = rA$.
Case 3. When $r < 1 - \epsilon$: It is very similar to case 2 - note that r cannot be negative.

c.



d. If $A_\theta > 0$, gradient ascent on ℓ increases r until it reaches $1 + \epsilon$. For $r > 1 + \epsilon$ the gradient is zero. If $A_\theta < 0$, gradient ascent on ℓ decreases r until it reaches $1 - \epsilon$. For $r < 1 - \epsilon$ the gradient is zero.

Exercise 4. 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.

Solution:

a. Input of DQN is the state, and its out put is a set of values for each action, then we need 100 input neurons and 10 output neurons.

b. Input of DDPG is a pair of state and action, and its output is the value corresponding to that pair. So, we need $100 + 10 = 110$ input neurons and only 1 output neuron.

c. Like DQN, DDPG is an off-policy method, i.e. the learning rule should update the greedy policy. Let us see what happens if we take the action a_{j+1}, which is equal to $\pi_{\psi_{j+1}}(s_{j+1}) + \epsilon$, where by ψ_{j+1} we mean the parameters of the policy at time $j + 1$, which is changing through time. We can see three problems with using this on-policy action in the update rule. First, we should not use on-policy next actions with replay buffers as in DQN or DDQP, because $\hat{Q}(s_{j+1}, a_{j+1})$ may become very different from $\arg\max_a \hat{Q}(s_{j+1}, a)$ when θ and ψ are changing over time. Second, since ψ is changing we loose the stabilizing effect of the target network. Third, we do not want our updates to depend on exploration through the dependence on the random variable ϵ (see d.).

d. In DDPG, the Gaussian random samples ϵ are used for exploration, similarly as we may choose a random action with probability ϵ in ϵ-greedy exploration when there is a countable number of actions. Ideally, $\arg\max_a \hat{Q}(s_{j+1}, a) = \hat{Q}(s_{j+1}, \hat{\pi}(s_{j+1}))$, but most likely $\arg\max_a \hat{Q}(s_{j+1}, a) \neq \hat{Q}(s_{j+1}, \hat{\pi}(s_{j+1}) + \epsilon)$, which is why we do not want to use $\hat{\pi}(s_{j+1}) + \epsilon$ to update the policy.