Artificial Neural Networks (Gerstner). Solutions for week 11

Deep Reinforcement Learning 2

Exercise 1. Uncorrelated mini-batches in A2C.

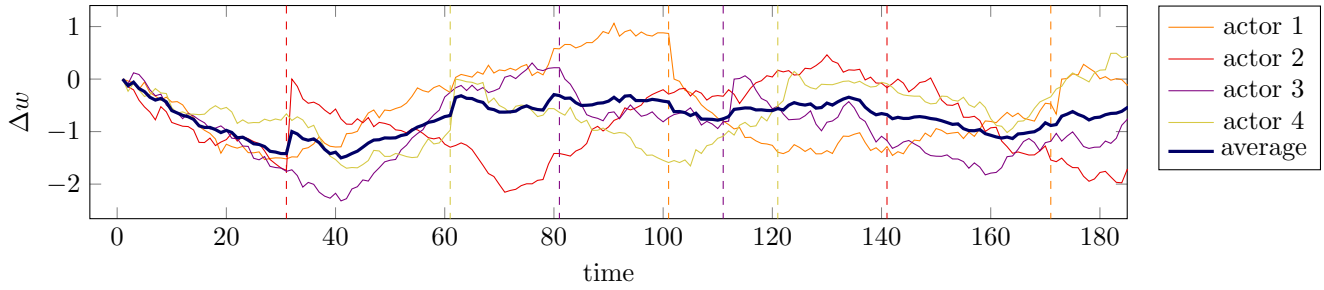
In the lecture you have seen a simple example of a single weight w that changes with temporally correlated updates Δw (red dots and red curve on slide 6). Reshuffling the updates in time led to a more stable learning dynamics (blue dots and blue curve). This example illustrates the effect of sampling 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. 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 17. Convince yourself that the equality on the last line of slide 17 is correct by explicitly writing out the expectations in the same way as we did on slide 16.
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 16, we have

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

b. The goal is to show the identity

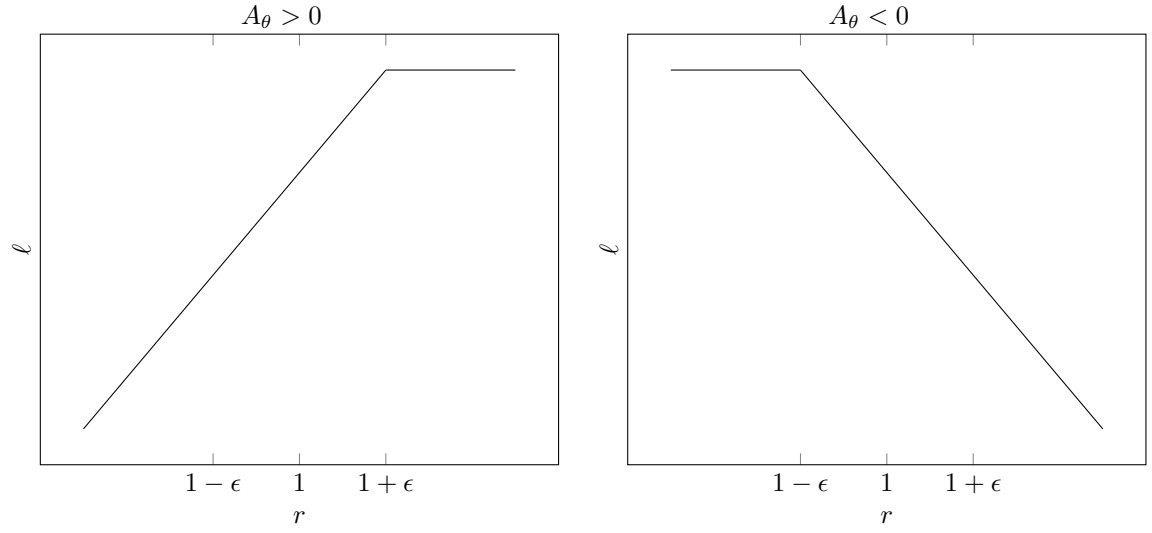
$$\ell_1(r,A) = \ell_2(r,A), \tag{2}$$

where

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

To do so, we show that for different values of r and A, two functions are equal:
Case 1. When $1-\epsilon \le r \le 1+\epsilon$: For this case, r is not clipped for ℓ_1, and we have $\ell_1 = rA$. Then, if $A \ge 0$, 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 $A \ge 0$ and is equal to rA if $A < 0$. At the same time, when $A \ge 0$, 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 3. 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 24).

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.

Exercise 4. Background Planning.

In this exercise we look again at the simple map of Europe on slide 29. You will run value iteration with goal Vienna. This means that we will keep $V(\mathsf{V}) = 0$ all the time.

a. Initialize all V- and Q-values to zero.

b. Apply the update rule of value iteration (equation 1 on slide 29) to all cities in parallel. Hint 1: keep $V(\mathsf{V}) = 0$ and don't worry, if you find e.g. $V(\mathsf{Z}) = -2$. Hint 2: "In parallel" means, that you should assume $V(s') = 0$ when running this step for the first time and take the value that you obtained in the previous iteration otherwise.

c. Repeat step b. until convergence.

d. Convince yourself that value iteration found the optimal solution. Write down, how you convinced yourself that the optimal solution was found.

Solution:

a. V(Z) = 0, V(P) = 0, V(N) = 0, V(F) = 0, V(M) = 0, V(R) = 0, V(L) = 0, V(V) = 0, V(B) = 0

b. V(Z) = -2, V(P) = -7, V(N) = -3, V(F) = -4, V(M) = -3, V(R) = -4, V(L) = -2, V(V) = 0, V(B) = -4

c. V(Z) = -4, V(P) = -11, V(N) = -6, V(F) = -6, V(M) = -5, V(R) = -8, V(L) = -4, V(V) = 0, V(B) = -8
V(Z) = -6, V(P) = -15, V(N) = -8, V(F) = -7, V(M) = -7, V(R) = -11, V(L) = -6, V(V) = 0, V(B) = -12
V(Z) = -7, V(P) = -19, V(N) = -10, V(F) = -7, V(M) = -9, V(R) = -13, V(L) = -8, V(V) = 0, V(B) = -15
V(Z) = -7, V(P) = -22, V(N) = -12, V(F) = -7, V(M) = -9, V(R) = -15, V(L) = -9, V(V) = 0, V(B) = -17
V(Z) = -7, V(P) = -24, V(N) = -12, V(F) = -7, V(M) = -9, V(R) = -15, V(L) = -9, V(V) = 0, V(B) = -19
V(Z) = -7, V(P) = -24, V(N) = -12, V(F) = -7, V(M) = -9, V(R) = -15, V(L) = -9, V(V) = 0, V(B) = -19

d. We can manually check some values. For example, the shortest path from Lausanne to Vienna goes through Zurich and costs -9, which is the final value that was found by value iteration.