2. Assignment 1: Q-Learning - Tabular & Deep

2.1. Motivation

You have so far worked with tabular RL algorithms. However, if the state-space is very large, storing values in a table becomes infeasible, and if the state-space is infinitely large (continuous) it is even impossible.

This can be overcome by *function approximation*, essentially parameterizing the action-value function with some parameters such that $Q_\theta(s, a) \approx Q(s, a)$ for any state action pair $(s, a) \in \mathcal{S} \times \mathcal{A}$. Technically, tabular methods are a special case of function approximation (cf. lecture, practicals or (Sutton, 2018), Exercise 9.1), where the number of tunable parameters equals the size of the state-action space, i.e. $|\theta| = |\mathcal{S} \times \mathcal{A}|$. If we use less parameters, i.e. $|\theta| < |\mathcal{S} \times \mathcal{A}|$, the resulting function approximation requires less memory, than its tabular counterpart, at the cost, that we can not store an *independent* action-value for each state-action pair, sometimes referred to as generalization.

The most prominent function approximator are Neural Networks, which will be the main subject of this assignment. Mnih et al. (Mnih, 2013; Mnih et al., 2015) were the first to report a successful implementation of neural networks as function approximators for Q-Learning, called Deep Q-Learning (DQN). The environment we will focus on, will be Cartpole (Towers et al., 2024).

2.2. Tasks

The assignment will resolve around two major parts: A more theoretical component and some experiments.

Theory

In the realm of tabular RL, we have update rules, inspired by the Bellman Equation. In contrast, to optimize the parameters of neural networks, we require a loss function. In this part of the assignment you are asked to explain how the Loss function of DQN is derived/motivated, and moreover, to highlight, why we could not use the update rules, employed in tabular RL.

Experiments

Implement a Q-Learning agent with (naive) function approximation by neural networks and plot the resulting learning curves. Keep in mind, that challenges that arise in tabular RL (for instance exploration strategies) still exist with function approximation. How does this Agent perform? Does it reach the optimal performance in Cartpole? As a result of function approximation, we have next to the RL specific hyperparameters (exploration, ...) also network specific hyperparameters (network architecture, learning rate, ...). Implement an ablation study to test the effect of hyperparameters and discuss the results.

In general: You may compare your results to the provided baseline implementation. However, don't interpret this as a target, it is only designed to provide you a way to put your results into perspective.

The reality is, that deep neural networks seldom work out of the box, and some more engineering tricks are required. For instance, Mnih et al. (Mnih et al., 2015) used the following tricks

(i) **Target network:** During the update, we try to push the q values towards the target q values which is the immediate reward plus the bootstrapped q values. Since a single update will affect the whole network, and it will cause the target q values to become non-stationary. In order to make stationary target q values, we could use another network for providing target q values and update it using the weights of the original network periodically.

(ii) **Experience Replay:** By maintaining a replay buffer, we are able to reuse collected data multiple times. Meanwhile, sampling batches randomly from the buffer can break the correlation between consecutive data, which can make the training more stable.

To improve the performance of your (naive) Q-Agent with deep function approximation, implement these two features, and compare the performances of all four possible configurations: Naive - Only TN - Only ER - TN & ER. We recommend to fix the hyperparameters for this ablation study, you may simply select the set that worked best for the previous part.

Experiment Design The number of environment steps per run should be around 10^6, which is also the amount of steps that we used to obtain the baseline. Moreover, for every configuration/setup you should do roughly five repetitions, to make sure to get a more reliable estimate. If you don't manage to run that many environment steps we encourage you to have a look into how to vectorize your environment. If this doesn't work out, you may cut down the number of steps, but we encourage you to incorporate that into your discussion, especially as you are provided with a baseline. Lastly, please note, that the baseline should not be seen as optimal results for DQN, these are just results from some RL algorithm, that is considered "solving" CartPole in a reasonable amount of time.

BONUS

These instructions are enough to pass the assignment with good grade. Should you however want to impress us, then you are welcome to show additional experiments. In general you are free, but as some initial inspiration you may extend the study to other gymnasium environments, or add some more engineering tricks to boost the performance (inspiration could for instance be found in (Hessel et al., 2018)).

2.3. Task-Checklist

1.1 Explain how the loss function in Q-Learning is derived

1.2 Highlight why we cannot use update rules as in tabular RL

1.3 Pseudo Code for DQN with ER and TN; should be around 10 algorithmic steps

2.1 Implement Q learning in Cartpole. Plot the learning curves (Return over environment steps). Does it reach optimal performance?

2.2 Implement an ablation study for the hyperparameters. The minimum amount of required hyperparameters is the learning rate, the update-to-data ratio (how many environment steps per update), the networksize and the exploration factor. For each of these you should test at least 3 values (small, medium, high)

2.3 Add a target network and a replay buffer

2.4 Plot results for Naive - Only TN - Only ER - TN & ER in a single graph (for a fixed set of hyperparamters)

2.5 Reflect on your results.

Bonus To obtain a bonus of up to 1 point, you can do either perform additional experiments, as outlined in the "BONUS" section above. Alternatively we reward concise (yet complete) reports, where we reward half a point per page below the 6 page limit (up to a full point). Please note that this should not come at the cost of the completeness of your assignment, writing concisely is challenging and requires work.

2.4. Submission

Make sure to nicely document everything that you do. Your final submission consists of:

(i) Your source code, together with a requirements file and a README with instructions that allows us to easily (single command per experiment / sub task) rerun your experiments on a university machine booted into Linux (DSLab or computer lab). To generate a requirements file boot up a terminal, activate your virtual/conda environment (if you used one), navigate to the folder in which you want to generate the file and run the following command: `python -m pip freeze > requirements.txt`.

(ii) You have to use the ICML Latex template for your Report. (Obtainable here)

(iii) Chapter Structure: Abstract (at most quarter of a page), Introduction, Theory (Theory only, around one page), Experiments (setup & results; explaining environment should take at most quarter of a page), Discussion (discussion of results & weaknesses), Conclusion (reflect own work & outline future work).

(iv) A self-contained scientific pdf report of 4-6 pages with figures etc. This report contains an explanation of the techniques, your experimental design, results (performance statistics, other measurements,...), and overall conclusions, in which you briefly summarize the goal of your experiments, what you have done, and what you have observed/learned.

If you have any questions about this assignment, please visit our lab sessions on Friday where we can help you out. In case you cannot make it, you can post questions about the contents of the course on the Brightspace discussion forums, where other students can also read and reply to your questions.

2.5. Deadline

March 14, 23:59:59

References

Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor. In *Proceedings of the 35th International Conference on Machine Learning*, pp. 1861–1870. PMLR, July 2018.

Haarnoja, T., Zhou, A., Hartikainen, K., Tucker, G., Ha, S., Tan, J., Kumar, V., Zhu, H., Gupta, A., Abbeel, P., and Levine, S. Soft Actor-Critic Algorithms and Applications, January 2019.

Hessel, M., Modayil, J., Van Hasselt, H., Schaul, T., Ostrovski, G., Dabney, W., Horgan, D., Piot, B., Azar, M., and Silver, D. Rainbow: Combining improvements in deep reinforcement learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

Mnih, V. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal Policy Optimization Algorithms, August 2017.

Sutton, R. S. Reinforcement learning: An introduction. *A Bradford Book*, 2018.

Towers, M., Kwiatkowski, A., Terry, J., Balis, J. U., De Cola, G., Deleu, T., Goulao, M., Kallinteris, A., Krimmel, M., KG, A., et al. Gymnasium: A standard interface for reinforcement learning environments. *arXiv preprint arXiv:2407.17032*, 2024.