Dynamic Programming and Reinforcement Learning

Week 8a: Recap, Projects

Rainer Schlosser und Alexander Kastius

Enterprise Platform and Integration Concepts

31.05.21

Past Schedule

- 12.4.: Introduction & MDPs 15.4.: Finite Time MDPs
- 19.4.: Infinite Time MDPs 22.4.: Exercise (DP, VI, PI)
- 26.4.: Approximate Dynamic Prog. 29.4.: Q-Learning
- 3.5.: Problems with Q-Learning 6.5.: Artificial Neural Networks
- 10.5.: Deep Q-Networks
- 17.5.: DQN (2) / RAINBOW 20.5.: Policy Gradients (1)
- 27.5.: Policy Gradients (2)

Solution Methods with Full Information

Backward Induction

Uses the knowledge that a finite horizon exists to compute state and action values from the end of the process.

Return:

Optimal policy

Requirements:

Full knowledge about reward function, state transitions and events. A and S need to be discrete and finite.

A decision process with finite horizon.

Value/Policy Iteration

Iteratively either optimizes the value function or the policy, until an optimal policy is found.

Return:

Optimal policy (within computational limits).

Requirements:

Full knowledge about reward function, state transitions and events. A and S need to be discrete and finite.

ADP and Monte Carlo

Approximate Dynamic Prog.

Learns a policy by observing a process and only updating those states that actually occur. Can overcome the limitations of a large state space.

Return:

Near optimal policy

Requirements:

A and S need to be discrete and finite.

Monte-Carlo Methods

Learns a policy by repeatedly performing the process and estimating Q-values. Compute the Q-value estimate by updating a table based on values computed from full trajectories.

Return:

Near optimal policy

Requirements:

A and S need to be discrete and finite.

The process needs to have a terminal state that is reached with probability 1.

Temporal Difference Learning

TD-methods

Learns a policy by repeatedly performing the process and estimating Q-values. Compute the Q-value estimate by observing a single reward and estimate future rewards.

Return:

Near optimal policy

Requirements:

A and S need to be discrete and finite.

TD with function approximation

Learns a policy, by estimating Q-values. Instead of keeping tables to store the values, a function approximator (for example an ANN) is trained to estimate those values, by minimizing the TD-error.

Return:

Near optimal policy

Requirements:

A needs to be discrete and finite.

(Some) Improvements for DQN

Double DQN

Overcomes the maximization bias of Q-learning.

Introduces a second estimator, to be able to use target values which are not by definition the maximum of the output.

PER

Highly improves data efficiency. Introduces a buffer, which stores experience tuples. Select tuples from this buffer by their TD-error.

The buffer itself needs to be well implemented to allow efficient usage of the data.

n-Step Methods

Immediately take delayed rewards into account instead of slowly propagating them through the learning process.

Can make things more complex if applied off-policy, but works well if used with greedy policies.

Distributional RL

Learns distributions of rewards instead of their expected value.

Leads to more stable estimations of the overall value by correctly representing skewed distributions.

Policy Gradients for Continuous Control

Policy gradient and AC

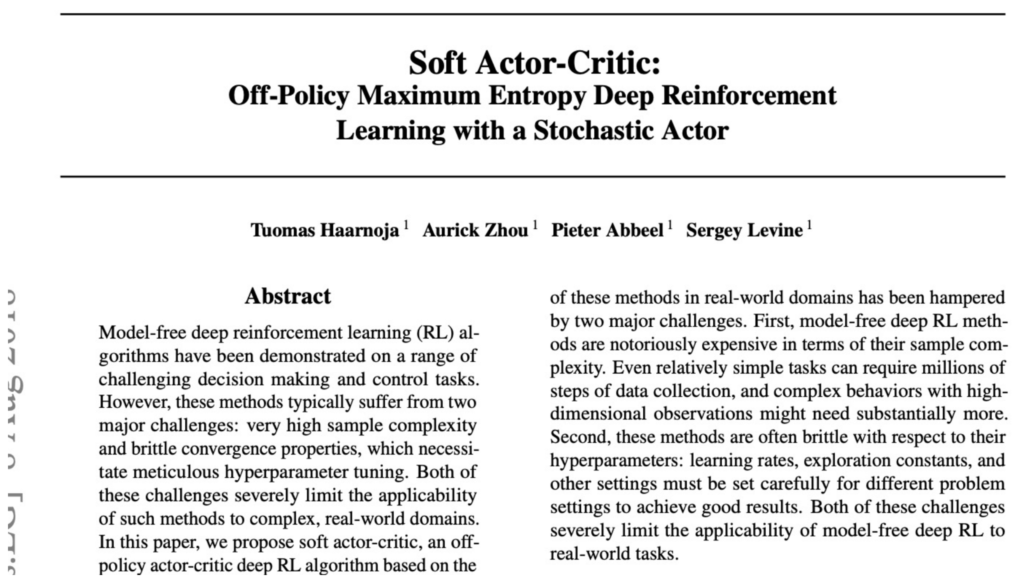
Learns a parametric representation of the policy by optimizing it according to the policy gradient theorem. Can be improved by adding a critic, which estimates s or s, a values to stabilize learning or even learn off-policy.

Return:

Near optimal policy

Requirements:

None, except that the process observed is markovian. This comes at the disadvantage of a complex system with high demand for data and many hyperparameters.



Soft Actor-Critic:
Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor

Tuomas Haarnoja [1] Aurick Zhou [1] Pieter Abbeel [1] Sergey Levine [1]

Abstract

Model-free deep reinforcement learning (RL) algorithms have been demonstrated on a range of challenging decision making and control tasks. However, these methods typically suffer from two major challenges: very high sample complexity and brittle convergence properties, which necessitate meticulous hyperparameter tuning. Both of these challenges severely limit the applicability of such methods to complex, real-world domains. In this paper, we propose soft actor-critic, an off-policy actor-critic deep RL algorithm based on the

of these methods in real-world domains has been hampered by two major challenges. First, model-free deep RL methods are notoriously expensive in terms of their sample complexity. Even relatively simple tasks can require millions of steps of data collection, and complex behaviors with high-dimensional observations might need substantially more. Second, these methods are often brittle with respect to their hyperparameters: learning rates, exploration constants, and other settings must be set carefully for different problem settings to achieve good results. Both of these challenges severely limit the applicability of model-free deep RL to real-world tasks.

Regarding The Style of Projects

Problem Oriented

We or you select a problem to work on and your challenge consists of finding the system and algorithm that yields the best performing policies, while being as efficient as possible. You have to choose your algorithm and modify it. If basic versions of the problem are solved you can look for extensions and generalizations!

Method Oriented

We or you select a special RL approach to work on. The goal consists of determining which use-cases allow efficient usage of the method under evaluation and which don’t work well. You might have to reimplement some algorithms based on their description in the respective paper. You will select the aspect under assessment beforehand.

Example Projects!

Problem oriented:

- Tetris
- Blobby Volley!
- Mau Mau/Uno
- Castle
- Flipper
- Pricing under Competition
- Pricing multiple products with substitution effects

Preferably, we can find something where you don’t have to spend 4 of 5 weeks to implement the environment.

Method/paper oriented:

- Distral (a framework for multi-task learning)
- Alpha Go (incorporates tree search and other additions)
- Eligibility traces (the final iteration of n-step learning)
- Impala/A3C (parallel learning on multiple environment instances)

This might require you to evaluate more than a single problem, to gain an understanding of what those algorithms are designed for.

Project Format

- The goal of the project is to understand the dynamics of reinforcement learning (what works, what doesn’t work and why) given a specific use-case or aspect of RL and presenting the knowledge gained, so everybody else can learn something from it as well.
- You work in groups of at most 3.
- Present your results in a documentation (ca. 15 pages, LNCS layout) and a final presentation (so everybody else can learn what you learned).
- Documentation should incorporate information about:
    - What problem or aspect did you work on?
    - What did perform well?
    - What didn’t?
    - Which tricks did you use to overcome the issues found?

Schedule

3.5.: Problems with Q-Learning	6.5.: Artificial Neural Networks
10.5.: Deep Q-Networks	
17.5.: DQN (2) / RAINBOW	20.5.: Policy Gradients (1)
	27.5.: Policy Gradients (2)
31.5.: Project Assignments	**3.6.: Project Assignments**
7.6.: Project Work/Support	10.6.: Project Work/Support
14.6.: Project Work/Support	17.6.: Project Work/Support
21.6.: Project Work/Support	24.6.: Project Work/Support
28.6.: Project Work/Support	31.6.: Project Work/Support
5.7.: Project Work/Support	8.7.: Project Work/Support
12.7.: Final Presentations	15.7.: Final Presentations
31.8.: Final Documentation	