Reinforcement Learning Lab

Lesson 7: TensorFlow and Neural Networks

Davide Corsi and Alberto Castellini

University of Verona
email: davide.corsi@univr.it

Academic Year 2022-23

Environment Setup

The first step for the setup of the laboratory environment is to update the repository and load the miniconda environment.

- Update the repository of the lab:

```
cd RL-Lab
git stash
git pull
git stash pop
```

- Activate the *miniconda* environment:

```
conda activate rl-lab
```

Safe Procedure

Always back up the previous lessons' solutions before executing the repository update.

Second Tutorial

The README file on the repository (link) contains a tutorial for TensorFlow, useful to complete today's and next assignments, in particular:

Optimization

TensorFlow is a package for non-linear optimization problems. Typically used to create, train and deploy artificial neural networks. In the first part of the tutorial, there are examples of non-linear problems solved with TensorFlow.

Create a Neural Network

In the second part of the tutorial, we will learn how to create a deep neural network with TensorFlow.

Train a Neural Network

In the last part of the tutorial, we train a simple neural network to perform a simple numerical operation.

First Assignment

In today's lesson, we will implement different functions. The first one is a simple TensorFlow script to find the best assignment to minimize a two-variable function. In particular, the file to complete is:

```
RL-Lab/lessons/lesson_7_code.py
```

Inside the file, a python class and a function are partially implemented. The objective of the first assignment is to complete it.

- **def findMinimum()**

Expected results can be found in:

```
RL-Lab/results/lesson_7_results.txt
```

Non-Linear Optimization

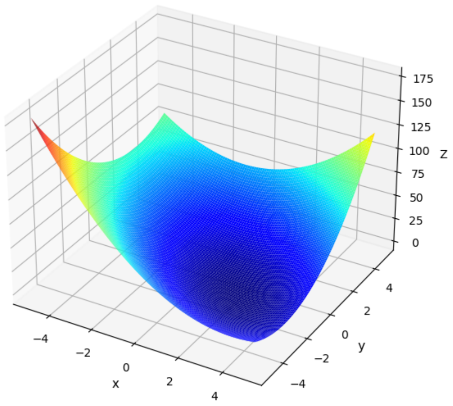

Figure: Visualization of the objective function to minimize $2x^2 + 2xy + 2y^2 - 6x$

- This function has a global minimum in -6, obtained assigning $x = 2$ and $y = -1$. The objective of the first assignment is to exploit TensorFlow to find these values.
- Remember to consult the tutorial (here) for hints and suggestions on how to solve the problem.

Second Assignment

The second assignment consists in training a neural network to predict the reward of an input state. The assignment is subdivided into three stages, corresponding to three functions to implement:

- **def createDNN()**
- **def collect_random_trajectories()**
- **def trainDNN()**

Expected results can be found in:

```
RL-Lab/results/lesson_7_results.txt
```

createDDN() and collect_random_trajectories()

1. For the first function, you should exploit the code snippet from the tutorial (here). Given the number of input, output, layers, and sizes, the function returns a neural network of the desired shape.
2. The second function, collect_random_trajectories, shows how to exploit the interactions with the environment to collect a dataset (the basic structure of reinforcement learning). You should implement a function that returns a two-dimensional array with the information of each episode, with the following structure:

```
memory_buffer = []
memory_buffer.append( [state, action, next_state, reward, done] )
```

Exploration Policy

To collect the data from the interaction with the environment, one can use different policies. In this lesson, you can use a random policy. For the implementation, you can take inspiration from lesson 4.

trainDDN()

For this function, you should exploit the code snippet from the tutorial (here, *the function requires a memory_buffer to perform the training*). The resulting DNN should be able to predict the reward from a given input state. From the given results, you may notice two main problems:

Wrong Prediction of the Goal State

The trained network correctly predicts the initial state (i.e., 0) but fails the estimation of the goal state (i.e., 48). The reason is that the goal state is rarely seen by the agent and there is not a lot of data about it.

Meaning of the Prediction

Even assuming the agent is able to predict all the rewards correctly, this information is not enough to build a policy. The DNN should predict the value (or expected reward) for choosing an action and not only the immediate reward.

In the next lesson, we will implement Deep Q-Network (DQN) to solve these problems.