Machine Learning Nanodegree
Capstone Proposal

Marcelo Roger García Escalante

July 30, 2018

1 Domain Background

Machine learning, in layman terms, consists of programming the computer to learn from previous data. A more detail description could be given as stated in [Wang and Tao(2008)]: "Machine learning is the process (algorithm) of estimating a model that's true to the real-world problem with a certain probability from a dataset (or sample) generated by finite observations in a noisy environment."

Today Machine Learning's method Deep Reinforcement Learning is gaining popularity since lots of applications has been given to it, such as Video Games, unmanned vehicles, and control systems. The latter is a well-known field where the goal is to make dynamical systems behave in a desired manner, to achieve this it is required to obtain the mathematical model of the system as explained in [Simrock()]

Since control systems are dynamic, all its environments are modeled as a continuous state space and most of the times as a continuous action space as well. Therefore, it signifies a challenging task to solve by machine learning, some of the algorithms able to tackle these problems are deep Q learning (for infinite state spaces and discrete action spaces), and actor-critic methods (for infinite state and action spaces).

Most of the times it is not possible to know all the system features thus it is not possible to get the mathematical model of the system, that is the main reason why a Reinforcement Learning approach could be useful to solve the problem. One approach of Reinforcement Learning applied to Control systems is presented in "A heuristic approach to reinforcement learning control systems" [Waltz and Fu(1965)]

1.1 Motivation

The motivation behind this project is because control is one of the required fields for robotics, in more specific for self-driving cars, where I would love to be involved in. Finally, I am convinced that my previous knowledge of control theory will be enhanced by relating Machine Learning with Control Systems, where there is a lot of room to work in. Therefore my main goal is to enlarge my experience in the combination of Deep Reinforcement Learning with Control Systems.

2 Problem Statement

The project aims to solve an OpenAI gym environment from the control section that can be seen here Control Environments. The environment selected is the MountainCar-v0 because it has a continuous state space with a deterministic action space being a good candidate for applying deep q learning. The details of the problem are as follows:

- **Task:** MountainCar-v0
- **Category:** Classic Control
- **Goal:** Get an under powered car to the top of a hill (top = 0.5 position) by creating momentum as shown in Fig.1

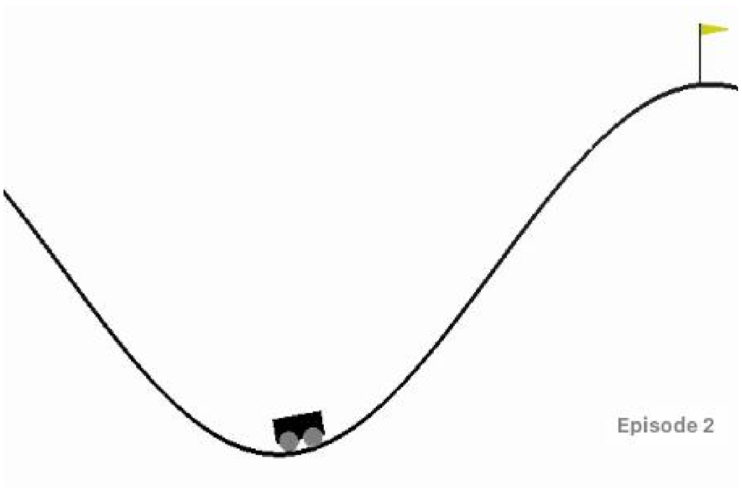

Figure 1: Mountain Car environment

- **State Space:**

Num	Observation	Min	Max
0	position	-1.2	0.6
1	velocity	-0.07	0.07

Table 1: State Space

- **Action Space:**

Num	Observation
0	push left
1	no push
2	push right

Table 2: Action Space

- **Termination Criteria:** The episode ends when either the goal is reached or 200 actions are made.
- **Rewards:** -1 penalty for each environment step until termination criteria is met.
- **Potential Solution:** Since the environment has continuous state space and discrete action space a good candidate would be a **Deep Q Network**

3 Datasets and Inputs

There is no dataset for this project since it involves a Deep Reinforcement Learning approach. However, the inputs for the model will be the states of the environment which in this case are two continuous measurements: position and velocity of the car as shown in 1.

4 Solution Statement

As explained in Section 2 The problem's best candidate is a Deep Q Network, which will have a Circular Memory (Deque) or a tree Memory(sum-tree) in case a prioritize Memory is required. Since the initial state is different in each iteration it may require a tweak of the state obtained from the environment before processing it to the deep network.

The deep Q network will be in charge of getting the best function to map the state input to one of the three actions, that is seeking for the optimal policy.

5 Benchmark Model

The project will be compared against three benchmarks:

- **Random approach**, to ensure the model is better than the most basic model.
- **-110 average rewarding**, as stated in OpenAI's documentation for the MountainCar-v0 the problem pass criteria is a reward of -110 in 100 consecutive episodes
- **Leader board**, as explained in the same documentation there are two algorithms that met the pass criteria as shown in Table3.

User	**Episodes before solve**
jing582	1119
DaveLeongSingapore	1967

Table 3: Leader board for the MountainCar-V0 problem

6 Evaluation Metrics

Average Reward The main metric for the project will be the average reward in 100 consecutive episodes.
Number of episodes The secondary metric is the number of episodes in which the problem converges to a solution.

7 Project Design

In this section, an explanation of the programming language, libraries and the selected algorithm to tackle the problem is given.

7.1 Programming Language and Libraries

- **Python 3.**
- **Scikit-learn**. Open source machine learning library for Python.
- **Keras.** Open source neural network library written in Python. It is capable of running on top of either Tensorflow or Theano.
- **TensorFlow.** Open source software libraries for deep learning.
- **OpenAI gym.** Open source framework for testing reinforcement learning projects

7.2 Deep Q Learning algorithm

As stated in DeepMind's paper the algorithm for the DQN should have the following structure:

```
• Initialize replay Memory with capacity N
• Initialize Main Q-network weights $W$ with random uniform distribution.
• Initialize Target Q-network weights with the main Q-network weights. $W^- \leftarrow W$
• For the episode in maximum number of (Episodes):
    – Reset environment
    – Prepare initial state: $S$
    – For time step t in maximum number of (Steps):
          Observation stage (sample to memory)
        ∗ Act in the environment based on state $S$ and get action $A$ using an epsilon-greedy algorithm
        ∗ Take action $A$ and make an environment action-step, get from the output of the environment the new state $S'$ and reward $R$
        ∗ Store the tuple $(S, A, R, S')$ in memory
        ∗ update state. $S \leftarrow S'$
          Learning stage ...
```

- Obtain random mini-batch of tuples from memory. list of (S, A, R, S').
- Use Target Q-Network to predict the target Q for Main Q-Network for each tuple of the mini-batch.
  $\hat{Q}$=predict(S)
  $\hat{Q}[A] = R + \gamma argMax_a(Q[S', W^-])$
- Update weights in the Main Q-Network based on the target $\hat{Q}$ as follows:
  $\Delta W = \alpha(Q - \hat{Q})\nabla_W(\hat{Q})$
- Update Target Q-network weights with the Main Q-network weights every C steps. $W^- \leftarrow W$

References

[Simrock()] S. Simrock. Control theory.

[Waltz and Fu(1965)] M. Waltz and K. Fu. A heuristic approach to reinforcement learning control systems. *IEEE Transactions on Automatic Control*, 10(4):390–398, Oct 1965. ISSN 0018-9286. doi: 10.1109/TAC.1965.1098193.

[Wang and Tao(2008)] J. Wang and Q. Tao. Machine learning: The state of the art. *IEEE Intelligent Systems*, 23(6):49–55, Nov 2008. ISSN 1541-1672. doi: 10.1109/MIS.2008.107.