Machine Learning Nanodegree
Capstone Project

Marcelo Roger García Escalante

August 14, 2018

1 Definition

1.1 Project Overview

In layman terms, Machine learning consists of programming the computer to learn from previous data. 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), both of them being from Deep Reinforcement Learning field.

In many cases it is not possible to know all the system's features thus it is not possible to get the mathematical model of the system, that is where Deep Reinforcement Learning came in place. There are lots of works developed in Control Systems related to Reinforcement learning. For instance, one approach of Reinforcement Learning applied to Control systems is presented in "A heuristic approach to reinforcement learning control systems" [Waltz and Fu, 1965]

1.2 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

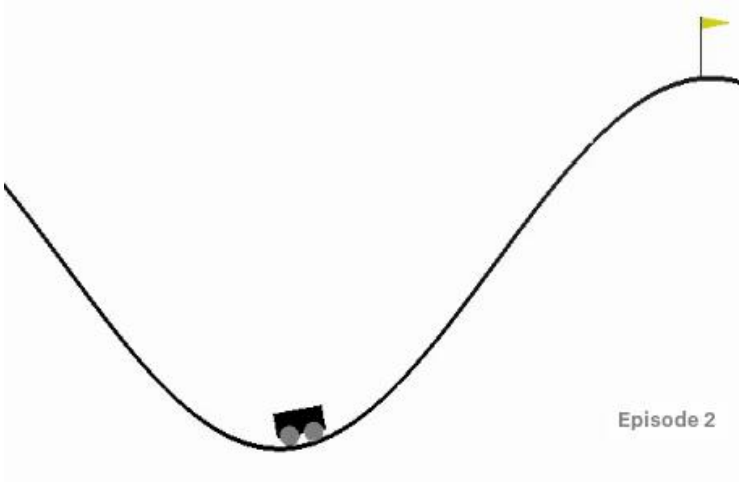

Figure 1: Mountain Car environment

- **Termination Criteria:** The episode ends when either the goal is reached or 200 actions are made.
- **Rewards:** -1 penalty for each environment step until the termination criteria is met.
- **Initial state:** The environment starts with the car randomly positioned between -0.6 and -0.4 and always with a velocity of 0.
- **Potential Solution:** Since the environment has continuous state space and discrete action space, a good candidate would be a **Deep Q Network**
- **Solution Concerns:** The car will eventually find a solution, however, in order to get an optimal policy in fewer episodes, it might be required to implement Double Deep Q Learning or prioritized memory.

1.3 Metrics

Average Reward The main metric for the project will be the average reward in 100 consecutive episodes. The reason to take 100 rewards averaged is to avoid the model to work just for certain specific states in the environment.
Number of episodes The secondary metric is the number of episodes in which the problem converges to a solution.

2 Analysis

In this section, an analysis of the data, the algorithms related to the solution and the benchmarks to compare its performance are discussed.

2.1 Data Exploration

There is no dataset for this project since it involves a Deep Reinforcement Learning approach and learning takes place while running. However, the inputs

for the model are the states of the environment which in this case are two continuous measurements: position and velocity of the car as shown in Table1, and three actions that are part of the discrete action space shown in Table2.

- **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

2.2 Exploratory Visualization

As shown in the image below Fig2 the left edge of the environment is at -1.2 units, the goal of the environment is at 0.5 units represented by the flag, and finally the bottom of the environment is at -0.5 units.

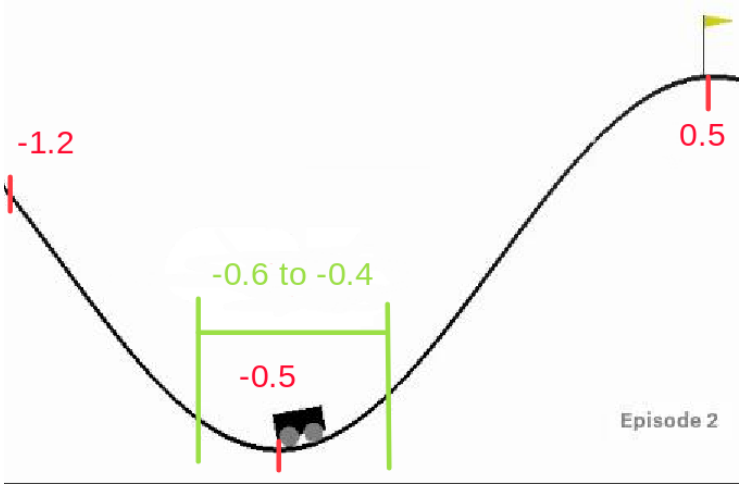

Figure 2: Mountain Car environment key positions

On the other hand, as shown in Table1 velocity's range for the car is from -0.07 and 0.07 thus we can spot out an evident possible issue. Since both the velocity and position are the two variables fed to the deep neural network it is important that both are normalized as explained in this video. Normalizing

the data input fed to the deep neural network will enhance optimality of the algorithm since fewer steps for the gradient descent in the loss function will be required as illustrated in Figure3. Where on the left we can note the function when a data input is not normalized and where is harder for the gradient descent to reach the global minimum, whereas at the right we could observe the function of the same data but this time normalized and in which clearly the gradient descent reaches the minimum smoothly.

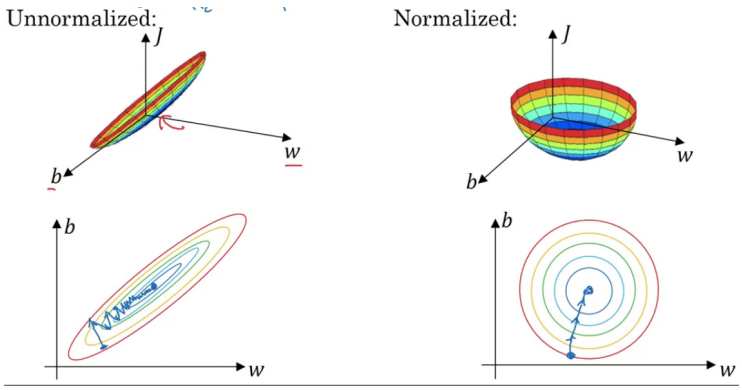

Figure 3: Unnormalized vs Normalized data [Andrew,]

2.3 Deep Q Learning algorithm

As stated in DeepMind's paper the Deep Q Network algorithm should be adequate for any problem where the state space is continuous and the action space discrete. The DQN should follow 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 a maximum number of (Episodes):
  - Reset environment
  - Prepare initial state: S
  - **For** time step t in a 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. $W^- \leftarrow W$

2.4 Benchmark

The project has been compared against three benchmarks:

- **Random approach**, to ensure the model is better than the most basic model.
- **Leader board**, as explained in OpenAI's documentation there are two algorithms that met the -110 average rewarding criteria as shown in Table3.

User	Episodes before solve
jing582	1119
DaveLeongSingapore	1967

Table 3: Leader board for the MountainCar-V0 problem

3 Methodology

3.1 Data Preprocessing

As spotted out in subsectionExploratory Visualization, an issue with the data (our state space) input for the deep neural network is that it is not normalized. This problem could shrink the speed of convergence of our model. Therefore, a pre-process of the data was required, where the input data was converted to a -1 to 1 range instead of the -1.2 to 0.6 for the position, and from -0.07 to 0.07 to -1 to 1 for the velocity. The process is illustrated in the following code snipped:

```
def normalize_data(self,state):
     position = state[0][0] #getting position from state
     velocity = state[0][1] #getting velocity from state
     state[0][0] = (position + 0.3)/0.9
     state[0][1] = (velocity)/0.07
     return self.state
```

As we can note from the code snippet above, the position is added by 0.3 and divided by 0.9 in order to get a -1 to 1 range. On the other hand, the velocity is just divided by its maximum 0.07 value to get a -1 to 1 range as well.

Another featured addressed in the pre-processing stage was the accumulation of previous states into a single one, this idea was borrowed again from DeepMind's paper, where the idea was to stack more states into a single one to obtain additional patterns or relationships between states. The code snippet below shows how this is done.

```
def stateProcessing(self,new_state):
     #This function pre-processes and concatenates last observation
         with a stack of previous states
     state_history=self.state
     new_state=new_state.reshape(1,2)
     new_state=self.normalize_data(new_state)
     self.state=np.concatenate((state_history[1:],new_state),axis=0)
     return self.state
```

3.2 Implementation

The process of implementation of the deep Q learning was built following the steps stated in the Deep Q Learning algorithm section with the following parameters introduced:

- **Number of episodes:** 4000
- **Maximum number of steps per episode:** 200 (limited by the environment)
- **Number of stacked states:** None
- **Number of episodes to update target model's weights:** 6
- **Replay memory structure type:** Deque
- **Memory size:** 5000
- **Mini-batch size for training:** 128
- γ**:** 0.99
- **Initial** ϵ**:** 1.0
- ϵ **decay factor:** 0.99
- **Minimum** ϵ **value:** 0.01
- **Learning rate:** 0.005
- **Deep Q Network implemented:**
  - **input shape** = (number of stacked states)+state space size
  - A fully connected **hidden layer** (Dense layer) with **200 nodes**.
  - **Uniform** initialization of weights
  - **L2 regularizer** with a 0.01 weight

- **Batch normalization** layer
- **Relu** activation layer
- **Output layer** with shape = action space size with a **linear** activation
- **Loss function:** mean square error
- **Optimizer:** Adam

3.3 Refinement

After implementation of the Q learning algorithm the main problem faced was the long training required for first convergence in a solution. The problem was due to the configuration of the environment since the most visited states were at the bottom because of gravity pulling the car down. Therefore, a lot of the episodes were occupied just on the exploration stage, otherwise, the algorithm would not find a solution and if it founds it, it diverged again after a couple of episodes later.

1. An alternative solution implemented to catalyze the convergence of the algorithm was to use a custom reward function, as shown in the following pseudo code:

IF current position of the car = Goal position

Reward event

IF current position of the car = Left edge of the environment

Punish event

IF car ascending and velocity produced is contrary to the action. (E.g. action applied is right but the car goes left due to gravity)

Punish event

IF car's speed is high and suddenly drops due to an action

Punish event

The result of this custom reward system was a significant increase in the speed of convergence of the algorithm from more than 1000 to 400 episodes.

2. Another important achievement in velocity of convergence was the implementation of a skipping frames system, which was inspired from the DeepMind's paper, where they skipped frames to speed up the training process of the Atari images fed to its neural network. The skipped frames were set to 4, this means that the same action was applied to the next four states. The result was a huge increase in the speed of convergence, that in conjunction with the custom reward system dropped the 400 episodes to 200 episodes.

4 Results

In this section, the results of the model applied to the environment are illustrated. The first sub-section shows an evaluation of the model based on the metrics stated in Metrics. The last sub-section shows a Justification of the model, which is a comparison of its performance against the Benchmark.

4.1 Model Evaluation and Validation

The reason to implement a custom reward (CR) + skipped frames (SF) system as explained in Refinement was to favor the creation of momentum. Momentum is the key physical attribute to solve this environment which at the same time is proportional to the speed of the car and its mass. However, since we do not have control over its mass, the only variable to favor in the environment is velocity, that is why all CR + SF focus on increasing the speed of the car. The results compared to a normal environment reward (ER) after the implementation of the DQN are shown in Table4.

	Environment's Reward (ER)	**ER+Custom Reward (CR)**	**CR+Skipped Frames (SF)**
Episodes required to find a solution	>2000	>400	>200
Best 100 average Reward in 2000 episodes	< -190	-160 to -150	-130 to -120

Table 4: ER vs CR+SF

.

4.2 Justification

The results of the model against the benchmarks stated for this projects are presented in Table5

	Random algorithm	**First in Leader Board (jing582)**	**Second in Leader Board (DaveLeong)**	**DQN Model**
Episodes required to converge to a solution	No solution Found	1119	1967	<1000
100 average range in Testing the algorithm	-200	>-110	>-110	-130 to -120

Table 5: Deep Q learning model against Benchmarks

As it can be seen in the table above, our model is much better than a random algorithm where the 100 average in testing is -200, whereas the DQN model result is between -130 and -120. On the other hand, both leaders got better 100 average results than the DQN model. However, the speed of convergence of the DQN was faster, since it required less than 1000 episodes to converge

to a solution. Overall we can state that the DQN is a good candidate to solve the Mountain Car problem, and to strengthen the statement we could find the graph of the learning process of the model in Figure4

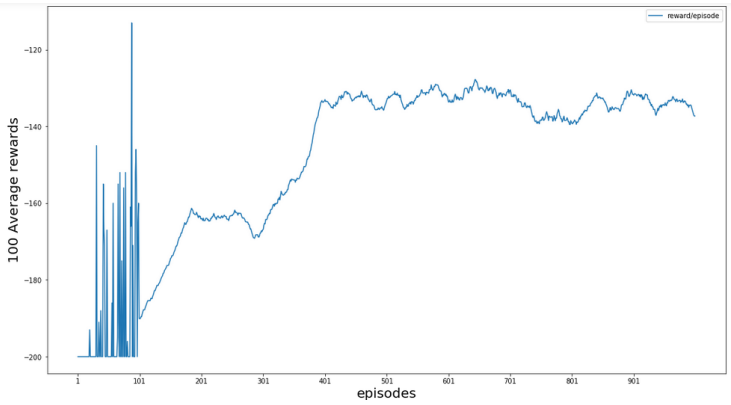

Figure 4: Learning process of the Deep Q learning approach to the Mountain Car problem

.

5 Conclusion

This project has turned out to be more challenging than expected. The first problem to overcome was the lack of a GPU to work with the neural network which delayed the training process. Therefore, the work done in this project focused more on enhancing the speed of convergence of the algorithm rather than getting a better 100 reward average. The other main conflict faced in this project was the lack of states visited at the beginning of the training process, since the car tended to visit more the bottom of the mountain due to gravity pulling down the car, that was what delayed the training process the most. At the end, generating an additional custom reward of the environment plus skipping frames turned to be the key features to overcome these problems, where the approach was to build a Deep Q Learning model the more simplistic possible to solve the problem since a more complex model should lead to a better 100 average reward value but at the cost of more computational resources. Overall the project highlighted the importance of an optimal algorithm, where the goal was to learn more from less information of the environment (i.e. fewer episodes required). Finally, to visualize the outcome of the last 100 average reward training of the model a video was generated which can be seen Here.

5.1 Future Improvements

There is plenty of space to enhance the performance of this project. Some of the possible ideas are the following:

- Try training the model with a more complex deep neural network, perhaps one more hidden layer.

- Try a DDQN, the model created for this project already has implemented an option for a Double Deep Q Learning process. It will demand more computational resources but it must perform better, especially at the beginning of the training process.
- Try to implement prioritized memory since one of the main problems stated in the training process was the lack of new states to learn from, a prioritized memory should help overcome this issue since it will tend to select the new states or the states from which it will learn more.
- Try stacked states, the model created has already implemented the option to stack more states into the Deep Neural Network. This idea was brought from DeepMind's paper where four Atari frames were stacked into a single state to learn additional patterns and features from the environment. [Mnih et al., 2015]

References

[Andrew,] Andrew, N. Improving deep neural networks: Hyperparameter tuning, regularization and optimization. `https://www.coursera.org/lecture/deep-neural-network/normalizing-inputs-lXv6U`. MOOC presented at Coursera.

[Mnih et al., 2015] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., and Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533.

[Simrock,] Simrock, S. Control theory.

[Waltz and Fu, 1965] Waltz, M. and Fu, K. (1965). A heuristic approach to reinforcement learning control systems. *IEEE Transactions on Automatic Control*, 10(4):390–398.