Logo, company name

Description automatically generated

Text

Description automatically generated

Group (6)

Text

Description automatically generated with medium confidence

**Ahmed Mohamed Gaber** [**21AMGA@queensu.ca**](mailto:21AMGA@queensu.ca)

**Karim Gamal Mahmoud** [**21KGMM@queensu.ca**](mailto:21KGMM@queensu.ca)

**Mostafa Awad** [**21MMAA1@queensu.ca**](mailto:21MMAA1@queensu.ca) **(Leader)**

The Mountain Car

The Mountain car is an environment where a car must climb a mountain. Because gravity is stronger than the car's engine, it cannot merely accelerate up the steep slope even with full throttle. The vehicle is situated in a valley and must learn to utilize potential energy by driving up the opposite hill before the car can make it to the goal at the top of the rightmost hill.

The mountain car environment provides the following discrete actions:

* 0 - Apply left force
* 1 - Apply no force
* 2 - Apply right force

The mountain car environment is made up of the following continuous values:

* state[0] - Position
* state[1] – Velocity

Programmed Car:

The programmed car always applies force in one direction or another. It does not break. Whatever direction the vehicle is currently rolling, the agent uses power in that direction. Therefore, the car begins to climb a hill, is overpowered, and turns backward. However, once it starts to roll backward, force is immediately applied in this new direction.

Reinforcement Learning

Q-Learning is a system of rewards that the algorithm gives an agent for successfully moving the environment into a state considered successful. These rewards are the Q-values from which this algorithm takes its name. The final output from the Q-Learning algorithm is a table of Q-values that indicate the reward value of every action that the agent can take, given every possible environment state. The agent must bin continuous state values into a fixed finite number of columns.

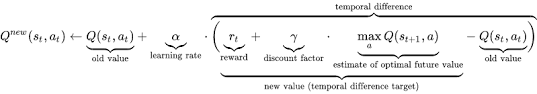
Learning occurs when the algorithm runs the agent and environment through episodes and updates the Q-values based on the rewards received from actions taken; Figure 1.REINF provides a high-level overview of this reinforcement or Q-Learning loop.

Diagram

Description automatically generated

Figure 1.REINF:Reinforcement/Q Learning

The Q-values can dictate action by selecting the action column with the highest Q-value for the current environment state. The choice between choosing a random action and a Q-value-driven action is governed by the epsilon ( ϵ ) parameter, the probability of random action.

Each time through the training loop, the training algorithm updates the Q-values according to the following equation.

There are several parameters in this equation:

* alpha ( α ) - The learning rate, how much should the current step cause the Q-values to be updated.
* lambda ( λ ) - The discount factor is the percentage of future reward that the algorithm should consider in this update.

This equation modifies several values:

* Q(st,at) - The Q-table. For each combination of states, what reward would the agent likely receive for performing each action?
* St - The current state.
* Rt - The last reward received.
* At - The action that the agent will perform.

The equation works by calculating a delta (temporal difference) that the equation should apply to the old state. This learning rate ( α ) scales this delta. A learning rate of 1.0 would fully implement the temporal difference in the Q-values each iteration and would likely be very chaotic.

Several hyperparameters are very important for Q-Learning. These parameters will likely need adjustment as you apply Q-Learning to other problems. Because of this, it is crucial to understand the role of each parameter.

* **LEARNING\_RATE** The rate at which previous Q-values are updated based on new episodes run during training.
* **DISCOUNT** The amount of significance to give estimates of future rewards when added to the reward for the current action taken. A value of 0.99 would indicate a discount of 1% on the future reward estimates.
* **EPISODES** The number of episodes to train over. Increase this for more complex problems; however, training time also increases.
* **SHOW\_EVERY** How many episodes to allow to elapse before showing an update.
* **DISCRETE\_GRID\_SIZE** How many buckets to use when converting each continuous state variable. For example, [10, 10] indicates that the algorithm should use ten buckets for the first and second state variables.
* **START\_EPSILON\_DECAYING** Epsilon is the probability that the agent will select a random action over what the Q-Table suggests. This value determines the starting probability of randomness.
* **END\_EPSILON\_DECAYING** How many episodes should elapse before epsilon goes to zero and no random actions are permitted. For example, EPISODES//10 means only the first 1/10th of the episodes might have random actions.

# Deep Q-Network (DQN) stable baselines3

Our environment is deterministic, so all equations presented here are also formulated deterministically for the sake of simplicity. In the reinforcement learning literature, they would also contain expectations over stochastic transitions in the environment.

Our aim will be to train a policy that tries to maximize the discounted, cumulative reward , where Rt0 is also known as the return. The discount, γ , should be a constant between 0 and 1 that ensures the sum converges. It makes rewards from the uncertain far future less important for our agent than the ones in the near future that it can be fairly confident about.

The main idea behind Q-learning is that if we had a function Q∗:State×Action→R , that could tell us what our return would be, if we were to take an action in a given state, then we could easily construct a policy that maximizes our rewards:



However, we don't know everything about the world, so we don't have access to Q∗ . But, since neural networks are universal function approximators, we can simply create one and train it to resemble Q∗ .

[Deep Q Network (DQN)](https://arxiv.org/abs/1312.5602) builds on [Fitted Q-Iteration (FQI)](http://ml.informatik.uni-freiburg.de/former/_media/publications/rieecml05.pdf) and make use of different tricks to stabilize the learning with neural networks: it uses a replay buffer, a target network and gradient clipping.

## Hyperparameters:

Using DQN with hyperparameter values chosen according to rl-baselines3-zoo which contains parameters optimized using Optuna.

The hyperparameters as follows:

Text

Description automatically generated

## Training and results:

after training for 1.2x10^5 timesteps we got the following results:

Chart, bar chart

Description automatically generated

The model took nearly 420 episode to reach the optimal solution with nearly 80,000 timesteps.

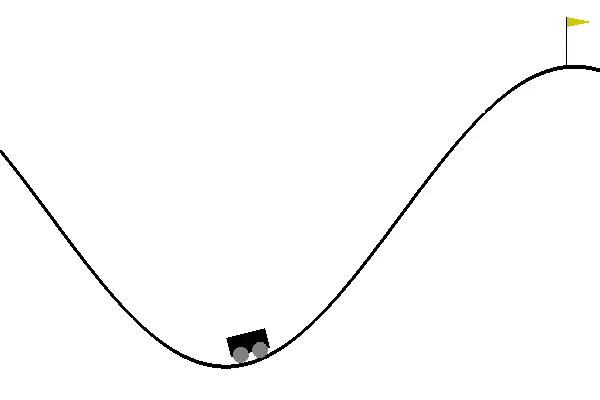
The optimal model has a mean reward of -92.6 according to the testing plot

Chart

Description automatically generated

## Validation:

the validation walk (using the optimal deterministic policy):



## Hyperparameters tuning:

We Tried to tune the hyperparameters more but changing the values of gamma and alpha (the learning rate)

gamma  = [0.99 , 0.97]

exploration\_fraction =  [0.3,0.1]

and fixing all other hyperparameters as the first trial.

We obtained the results then compared the performance with the first trial.

Chart, histogram

Description automatically generated

The purple line which is gamma = 0.99, exploration\_rate = 0.1 achieved better results than the first trial and faster converge with only 390 episode.

**The second model : Q-Learning Algorithm:**

**These are the hyperparameters we used for learning that give us the best performance:**

LEARNING\_RATE = 0.1 DISCOUNT = 0.99

EPISODES = 1000 SHOW\_EVERY = 100

DISCRETE\_GRID\_SIZE = [10, 10] START\_EPSILON\_DECAYING = 0.7

END\_EPSILON\_DECAYING = EPISODES//10

**The test performance that we had , after every 10 episodes of training. We ran the estimated policy in the environment for 5 test episodes:**

A screenshot of a computer

Description automatically generated with medium confidence**In the beginning of the training :**

Shape, rectangle

Description automatically generated

**In the middle of the training :**

A screenshot of a computer

Description automatically generated with medium confidence

Chart

Description automatically generated

**Note:** Optimal performance for training and testing in this section is highlighted by a dotted line.

A screenshot of a computer

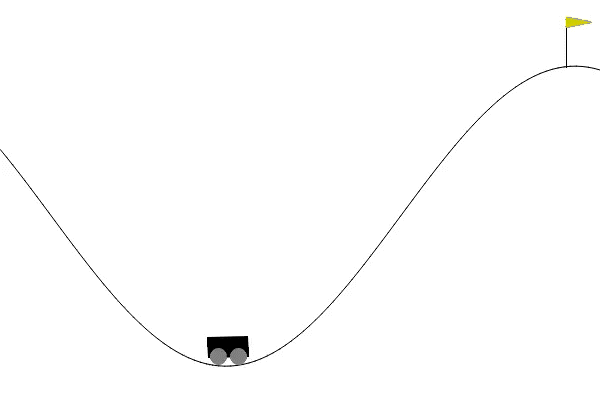
Description automatically generated with medium confidence**In the end of the training :**

Chart, bar chart

Description automatically generated

**Note:** Optimal performance for training and testing in this section is highlighted by a dotted line.

**Running and Observing the Agent:**



Background pattern

Description automatically generated with medium confidence**Visualize best Q-Table for each state and corresponding action:**

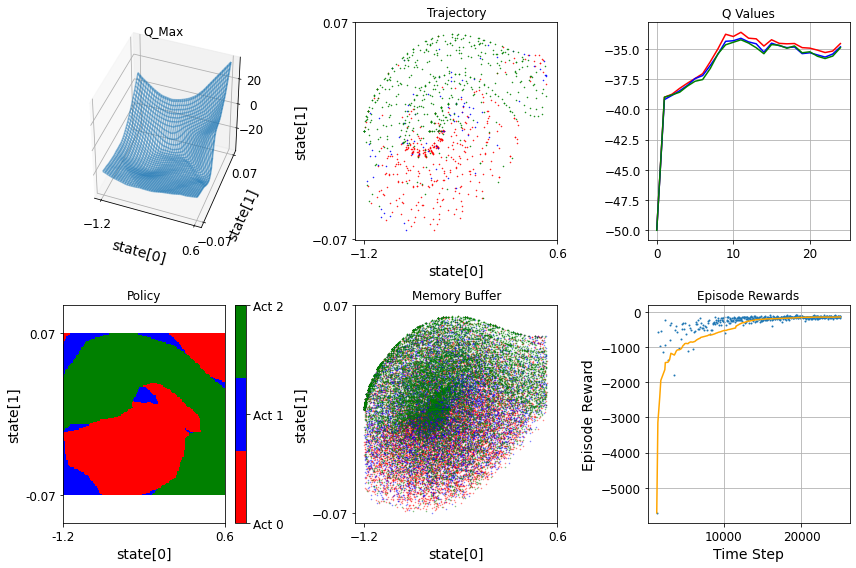
**The third model : Deep Q-Network Algorithm from scratch:**

**These are the hyperparameters we used for learning that give us the best performance:**

LEARNING\_RATE = 0.1 DISCOUNT/gamma = 0.9

A picture containing table

Description automatically generatedeps\_decay\_steps=20000, eps\_target=0.2, batch\_size=42

**plot the agent state:**