CS4287 Neural Computing - Assignment 3 - Deep Reinforcement Learning

[**Why RL is the ML paradigm of choice for this task 1**](#_pcpu11o8zg7f)

[Gym Environment 2](#_m369yaj2f8d0)

[**Implementation 3**](#_tbano8i5e32p)

[Capture and Sample The Data 3](#_ozy1ev7eeh6d)

[Network Structure 4](#_yrezrcnk1rqk)

[Q Learning 5](#_ljcf3vbp87n8)

[Independently Researched Concepts 5](#_xarjeyveqemx)

[Results 7](#_lw9rp7mawc8a)

[Evaluation of Results 8](#_69e1ayc3nzl)

[References 8](#_f5ijdsuhkt10)

## Why RL is the ML paradigm of choice for this task

Reinforcement Learning(RL) is useful for classic control problems such as Mountain Car,CartPole and Acrobot due to its ability in these four instances.

* Handle Sequential Decision Making
  + Allows the agent to learn optimal policies over time
* Continuous Action Spaces
  + RL can handle continuous action spaces, a common issue in control problems using algorithms such as Deep Deterministic Policy Gradients(DPPG) and Trust Region Policy Optimisation(TRPO),
* Exploration-Exploitation trade-offs
  + Agent needs to explore different actions to discover different strategies, for example in Acrobot, where it needs to find different strategies for swinging the pendulum while exploiting known successful actions.
* Model Free Learning
  + Doesn’t require a precise model of the system dynamics.Instead, it learns from interacting with the environment.

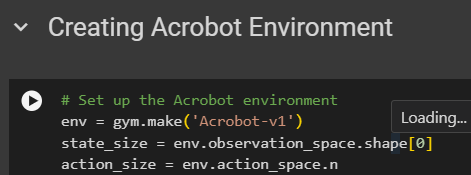
RL allows agents to learn from effective control strategies directly from interactions with the environment which make it a suitable paradigm for dealing with classic control problems.

## Gym Environment

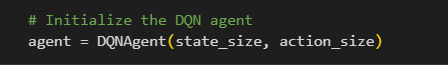
The OpenAI Gym environment is an interface used for developing and testing reinforcement learning algorithms.It provides environments which represent a different task and problem. The aim for this is to create a framework that allows for developers to easily compare and evaluate the performance of learning algorithms.

In this project we used the gym environment to create an instance of the Acrobot-V1 environment.

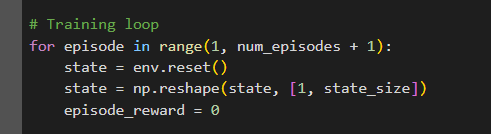
* We then defined the dimensions of the observation space through using the *env.observation\_space* attribute We also defined the number of actions in the action space space by using the env.action\_space attribute



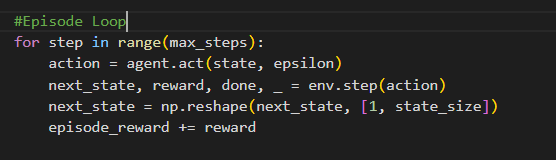
* We then created then initialized the DQNAgent with the action space and observation space information



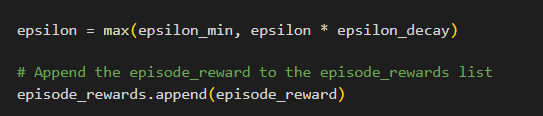
* We created a training loop that would run for the specified number of episodes in a variable named *num\_episodes* defined earlier in the code



* In the episode loop we had, it would iterate over each episode and the agent would interact with the environment of a specified number of steps(*max\_steps*). The agent then selects an action using *act* based on the current state and the exploration-exploitation tradeoff(*epsilon*) and then the environment would move forward using *env.step(action).*



* The agent is then trained based on the current state,action,rewards,next state and done flag.The epsilon value is decayed and updated, then the total reward for an episode is recorded

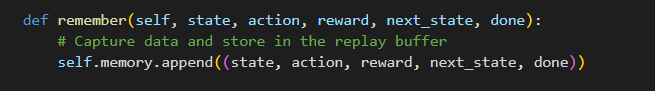


## Implementation

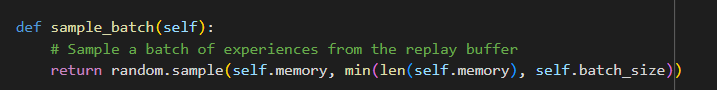
### Capture and Sample The Data

The data in this project is captured and sampled by using an experience replay buffer.

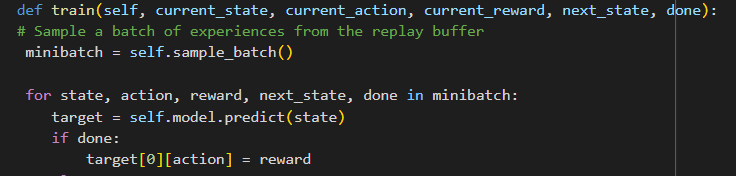
The data capture occurs in the *remember* method.After each step in the environment the parameters are added to the replay buffer and represent a single experience in the environment.



Data Sampling is done in the sample\_batch method, which is responsible for randomly sampling a batch of experiences from the replay buffer.This is done by using random.sample to select a set of experiences with a size equal to batch\_size and this is then used to train the DQN model.

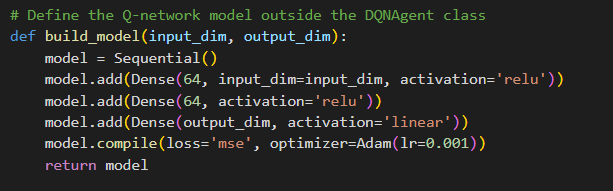


The sample\_batch is shown to be used in the train method for the DQN model. It takes a batch of training experiences and iterates over them to perform Q-Learning updates This helps to break temporal correlation between samples and improve the stability of Learning



### Network Structure

The Neural Network(NN) of the implementation has 3 fully connected layer. The model has an input layer with 64 units and uses the ReLu activation function with the input dimension being the size of the input state. This is similar to the second layer of the NN minus the input dimension as this is not the input layer. The Output Layer has units that match the dimensions of the action space and has the linear activation function since its a Q-value estimation task.The model is compiled using Mean Squared Error(MSE) and uses the Adam Optimiser with a learning rate of 0.001.



### Q Learning

The Q Learning occurs in the *train* method for the DQNAgent agent.

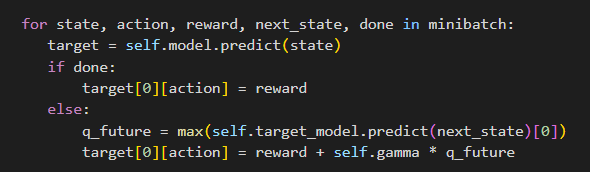
In this method, the first thing that happens is that the code samples experiences from the replay buffer. This replay buffer allows for the agent to learn from a mix of recent and older experiences.



The model then predicts the current Q-values for the current state using the model.This initializes the target values before they are updated based on the Q-Learning Update Rule.



If the episode is done, the q value for the chosen action is set to the immediate reward but if the episode is not finished then the chosen action for the Q value is chosen using the Q-Learning Update Rule. This is done by using the target model to estimate the best action in the next state. This is done by using the immediate reward and the discounted maximum Q-value of the next state, which follows the Q-Learning Equation

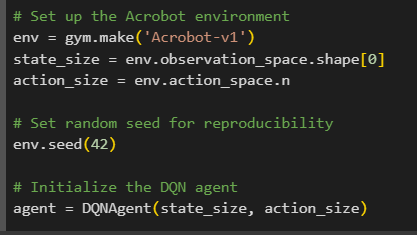


The Model is then trained using the updated Q values and the current state.This is done for a single epoch of training, updating the model weights to minimize the MSE between predicted Q-values and target values.

### Independently Researched Concepts

**Random Seed Initialisation**

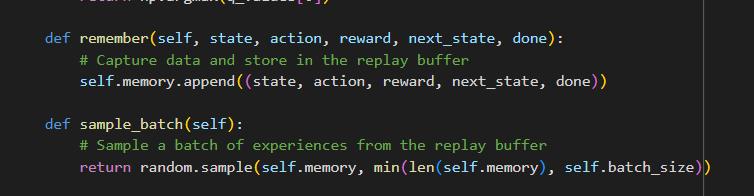
OpenAI Gym environments often involve some degree of randomness, such as initial conditions, noise in observations, or stochasticity in the environment dynamics. By setting a random seed, you make the environment's behavior deterministic. This is particularly important for reproducibility. When the seed is set, it ensures that any random process or condition is the same when you run the code.Using this parameter ensures that everybody who runs the code gets the same result. In our code we used a seed when setting the Acrobot environment.



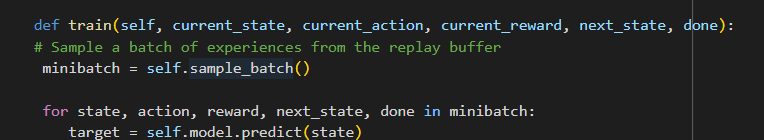
**Catastrophic Forgetting**

Catastrophic forgetting is when a NN forgets previously learned information. A common technique to counter this is using a replay buffer.

In our implementation we utilize a replay buffer to allow our DQN Agent to learn from a diverse set of experience to try and avoid the issue of catastrophic forgetting.It stores past experiences using the *remember* method, in the replay buffer and then the experiences are sampled in a batch using the *sample\_batch* method



These are then sampled in the train method and Q Learning updates are then applied to the NN



## Results

HYPERPARAMETERS FOR RESULTS

buffer\_size = 16

gamma = 0.95

epsilon = 1.0

epsilon\_decay = 0.995

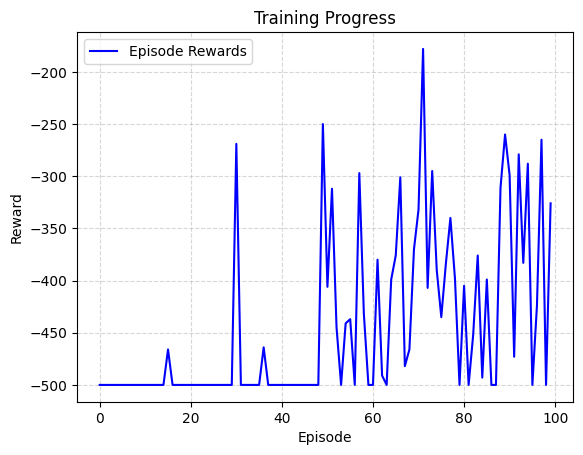
epsilon\_min = 0.01

alpha = 0.001

batch\_size = 32

learning\_rate = 0.001

update\_target\_frequency = 100 # Update target model every 100 steps



X-axis (Episodes): This axis represents the number of episodes the agent has gone through during training. Each episode is an instance where the agent starts the environment from a beginning state and continues until a terminal state or the end of the episode is reached.

Y-axis (Reward): The y-axis shows the total reward the agent received during an episode. The reward is typically a numerical value that the agent receives from the environment as a result of its actions. The goal of the agent is to maximize this reward over time.

## Evaluation of Results

The graph shows volatility in the rewards, without a clear upward trend. This could mean that the agent is still exploring or that it has not started to converge towards an optimal policy.

The fluctuations also suggest that the training process might not be stable. This could be due to many factors such as the complexity of the task.We explored varying the hyperparameters such as the learning rate and batch size but the trend for the graph was similar.

This shows that we may need to run a greater amount of episodes when training the DQNAgent to reach the optimal policy for the Acrobot environment, in order to see a consistent upward trend in the graph.We tried running a thousand episodes for the training of the agent but found that the system would consistently crash around the 250th episodes as a result of exhausting the resources available.

## References

Random Seed Initialisation - [https://towardsdatascience.com/how-to-use-random-seeds-effectively](https://towardsdatascience.com/how-to-use-random-seeds-effectively-54a4cd855a79#:~:text=A%20random%20seed%20is%20used,data%20science%20and%20other%20fields.)

Replay Buffer Catasrophic Forgetting -

<https://openreview.net/pdf?id=SyeCmVSl8r>