Authors:
Ardit Ymeri (201657885) - sgaymeri@liverpool.ac.uk
Devarghya Chakraborty (201646435) - sgdchakr@liverpool.ac.uk
Anushka Shukla (201654836) - sgashukl@liverpool.ac.uk
Darshil Bagadia (201652473) - sgdbagad@liverpool.ac.uk
Aman Ajith Prasad (201680988) - sgamanaj@liverpool.ac.uk

This document is a report for the second Machine Learning and BioInspired Optimisation (Comp532) class. It is split into two sections, one for each problem presented in the coursework handout.

1 Q1 - Deep reinforcement learning

OpenAI's Gymnasium is a collection of environments and games on which machine learning algorithms can be tested on. The environment that we picked for this assignment was the Acrobot environment. The goal of the Acrobot game is to swing a pendulum in such a way that its second segment crosses a specific height.

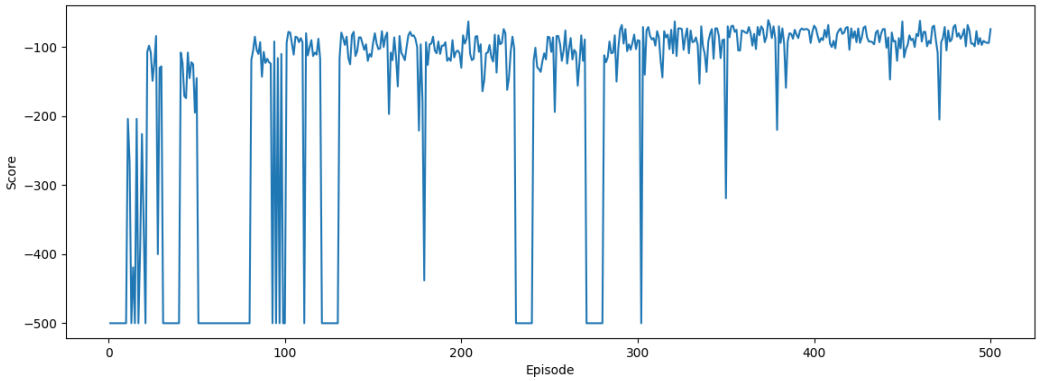
For every action taken in the environment, the environment deducts a point from the reward. If an agent takes random actions, it will continue to lose points until it reaches the maximum limit, which is -500 points. It is possible for a random agent to also win by chance, although this is an anomaly and occurs very rarely. As such, the median of N trials for an agent that acts randomly is -500.

For this problem, we will be using a Deep-Q Network (DQN). A Deep-Q Network is a type of reinforcement learning algorithm that uses a neural network to approximate the Q-value function, which represents the expected future rewards for each possible action in a given state.

Such a network is comprised of an input layer, hidden layers, and an output layer. Each layer has a number of neurons depending on the problem and hyperparameter testing. The input layer represents the state that the model will observe, and as such the number of neurons corresponds to the number of dimensions or features that our input has. In the case of Acrobot, this is the sine, cosine and angular velocity of theta1 and theta2, for a total of 6 features. The number of neurons on the hidden layers and the number of these hidden layers are obtained through experimentation, as they are hyperparameters that depend on the problem. We found good results using 2 layers of 64 neurons each, although different variations also worked fine. Lastly, the output layer represents the number of answers that we would like our model to put out. In other words, this is the number of actions that we can take in the environment, which in our case is 3.

We implemented the above model using Tensorflow's Keras library. Keras also comes with a class for implementing a DQN Agent. This class allows for the specification of a learning policy. For our solution, we used the BoltzmannQPolicy, which provides a similar idea to the greedy-ϵ approach used in the first assignment. In other words, it uses a combination of exploration and exploitation based on the "temperature" of each action, and therefore it does not get stuck on a solution like a greedy algorithm would. We also input some parameters based on recommendations by the documentation, such as the memory limit, window length, step warmup etc.

Once the model is defined, it is time to train it against our environment. For our testing, we trained the model for 10000 steps and then tested it for 5 episodes per iteration. We ran this training for 10 iterations, with various different configurations. This allowed us to see how the model improved over time. After not much training, the model was already able to reach scores of nearly -100, which meant that it was completing the task in around 100 steps. Based on other comparisons online of people solving this problem, it would appear that this score is considered a successful baseline. With that said, the model tends to hit cold streaks in earlier stages of testing, and gets more and more consistent as training progresses. Below is a visual representation of the scores achieved in the training performed by the above-mentioned settings.



For the actual submission, we have lowered the number of training steps to 1000, which should make it weaker but easier to run and verify the results. The line responsible for this is line 51, if any changes need to be made to verify the above-mentioned results.