5-1 Assignment: Cartpole Problem

Joel De Alba

Southern New Hampshire University

Professor Tim Alexander

08 / 05 / 23

The initial practice implementations resulted in little to no success when running the provided document from within the Aporto Access. After un-zipping and uploading python files there where several requirements to be met. These requirements included the packages needed to the code to function correctly such as the following libraries:

* ***Keras***
* ***Numpy***
* ***TensorFlow***
* ***Gym***
* ***Pygame --pre***

*For Different forms of implementation that were used:*

* ***Pyglet***
* ***Tensolflow-gpu***
* ***Tensorflow 2.12***

*Separate OS and Technologies Used*

* ***Aporto***
* ***Windows / Anaconda***

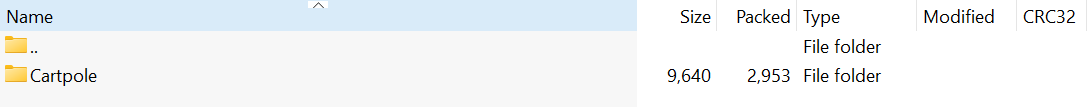
Packages were installed using ‘pip’ within the Ubuntu terminal and all showed to be installed fully and globally. There were still runtime issues that could not be resolved after several hours of research and study.

At first the instructions were followed from the given modules as follows:

* **Download the Zip File Provided “Cartpole.zip”**

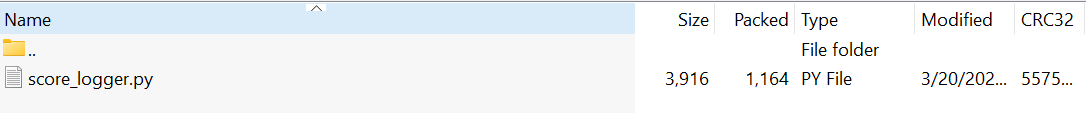
A black square with white text

Description automatically generated with medium confidence



A white and blue background

Description automatically generated with medium confidence



* **The next step was to extract and import the files into Aporto Virtual Lab under the Documents folder while maintaining the current folder structure including the scores folder.**

**A screenshot of a computer

Description automatically generated**

* **From here we are to review and modify the “Cartpole.ipynb” by performing maintenance on the provided code and renaming or creating a new file with the last name of the student and then Assignment5 at the end of the title. This new Jupyter Notebook file is to hold the provided code and any future developments made by the student.**

**A screenshot of a computer

Description automatically generated**

Following the installation of the required libraries, my initial challenge arose when I couldn't detect the 'score\_logger.py' file. Despite meticulously following the instructions and attempting several notebook restarts, Jupyter Notebook remained unable to locate the file within the designated folder. This complication became a significant hurdle, leading to multiple uploads and restructurings of the CartPole files in pursuit of successful 'scores' code file detection.

Upon successful detection of the 'score\_logger.py' file, it became evident that numerous variables and functionalities had become deprecated. Additionally, compatibility issues emerged due to the latest TensorFlow library update, specifically version 2.13.0.

Subsequently, the approach shifted towards uninstalling TensorFlow to install an older version that would ensure compatibility with both Keras and Gym frameworks. However, during this attempt, I encountered a roadblock. The console only permitted the installation of TensorFlow 2.12, which unfortunately also led to installation failure.

**Error Example 1 (Windows 11 Anaconda/Jupyter Notebook & Ubuntu Linux Aporto Virtual Lab)**

**A close-up of a computer screen

Description automatically generated**

**Error Example 2 (Ubuntu Linux Aporto Virtual Lab)**

**A screenshot of a computer error

Description automatically generated**

**A screenshot of a computer program

Description automatically generated**

After extensive research and employing various comprehension tactics, I encountered three major setbacks during the implementation of the code. These setbacks primarily revolved around the time taken for implementation, compatibility issues between libraries, and missing packages that failed to install correctly.

Through extensive trial and error, coupled with thorough investigation of effective documentation, I discovered that the version of 'pip' I was utilizing was contributing to the encountered issues. Upon uninstalling the current version of 'pip' and reverting to the prior version, several issues within my system were rectified. One notable issue was the initialization failure of pygame, which crucially impacted the successful execution of the code.

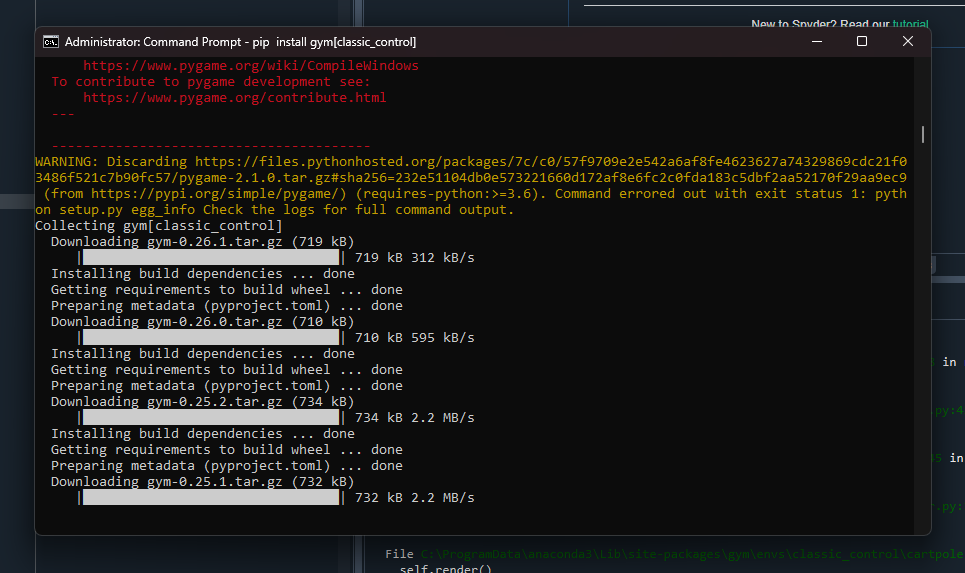
Compatibility issues can indeed lead to significant time wastage and concerns. Initially, I held the assumption that ensuring all libraries and packages were updated to the latest version would guarantee optimal performance. However, this experience served as a valuable lesson, highlighting the importance of acknowledging that not all libraries and packages will be compatible with each other. This lesson underscores the necessity of understanding compatibility limitations.

At this juncture, initial success was achieved in running the Python code, providing a functional template to work from. The CartPole problem was successfully rendered and displayed. This successful implementation was carried out using the Spyder Python 3.1.1 Text Editor, which offered the least number of errors. The main objective of this endeavor was to gain a comprehensive understanding of the capabilities and functionalities of Reinforcement Learning techniques and algorithms, including Markov Decision Process and Q-Learning. The pursuit of this goal remained steadfast and preserved the intent of achieving full completion.

**Theory and Understanding**

This project delves into the realm of reinforcement learning, employing a variety of contemporary methodologies and tutorials to tackle the 'CartPole' problem. The primary objective is to instruct the agent – represented by the 'Cart' – to execute appropriate actions such as 'Left,' or 'Right' thereby effectively maintaining the 'Pole' in an upright position within the designated environment. The agent's success is determined by its ability to prevent the pole from falling below a critical angle, securing maximum rewards. This endeavor revolves around conditioning an AI to optimize its actions through the Markov Decision Process framework, encompassing states, actions, transition probabilities, rewards, and a policy.

Before we begin explanation on the development of the CartPole Problem, the following images are provided to depict successful installation of the packages that were required to make this practice functional. Some Windows Updates and Visual Studio Installations were also a necessity for this process and code to be functional.

****

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

From the running application we can review the output received from the state of which returns an array of four values as shown in the following image:

**A computer screen with white text

Description automatically generated**

The initial state is defined by four values:

1. The first value corresponds to the x-axis coordinate, representing the cart position (x).
2. The second value denotes the cart velocity (x dot).
3. The third value represents the angle of rotation (theta) for the pole.
4. The fourth value signifies the angular velocity (theta dot) of the pole, which is being trained to maintain balance.

These values collectively establish the starting conditions within the environment.

The following image represents the values rendered from the solution within the Spyder Editor.

A screenshot of a computer

Description automatically generated

The image that follows depicts the program running and adapting accordingly:

A screenshot of a computer

Description automatically generated

In these image examples we can review the code structure as well as the runtime console print data that shows the runtime environment values. The Cart has only two possible actions, either a 0 or a 1. These values represent the direction the cart is pushed as action 0 is to push left and action 1 is to push right. The CartPole problem is often tackled using the Q-learning algorithm, which is a model-free reinforcement learning algorithm. Q-learning learns an action-value function that estimates the expected cumulative reward when taking a specific action in each state.

Experience replay is a technique used to improve the efficiency and stability of Q-learning. In CartPole, experience replay involves storing agent experiences (state, action, reward, next state) in a replay buffer. During learning, batches of experiences are randomly sampled from the buffer, which reduces the correlations between consecutive experiences and stabilizes learning.

The discount factor (gamma) determines the importance of future rewards in Q-learning. Higher gamma places more emphasis on future rewards, leading the agent to make decisions that consider long-term rewards. Lower gamma values give more importance to immediate rewards.

Deep Q-learning uses neural networks to approximate the action-value function (Q-values). The neural network takes the state as input and produces Q-values for each possible action as output. This allows handling high-dimensional state spaces efficiently. Neural network architecture forms the backbone of the Deep Q-Learning algorithm applied to the CartPole problem. It processes the input state information, learns patterns through the hidden layers, and provides actionable decisions through the output layer, effectively allowing the agent to learn how to maintain balance and achieve the goal of keeping the pole upright.

The learning rate determines how much the Q-values are updated based on new experiences. Increasing the learning rate may lead to faster convergence, but too high a learning rate can cause instability or overshooting. Decreasing the learning rate may lead to slower but more stable learning.

In conclusion this project was completed as indicated in this document in order to further aid complete and adequate comprehension of the concepts, the documents provided by the course modules aided and resulted in poor results, so this method was implemented to achieve a successful run of the RL scenario.

Sources

Gulli, A., & Pal, S. (2017). *Deep learning with keras : Get to grips with the basics of keras to implement fast and efficient deep-learning models*. Packt Publishing, Limited.

Beysolow, I. T. (2019). *Applied reinforcement learning with python : With openai gym, tensorflow, and keras*. Apress L. P..

Lamba, A. (2018b, September 3). An introduction to Q-Learning: reinforcement learning. Medium. <https://medium.com/free-code-camp/an-introduction-to-q-learning-reinforcement-learning-14ac0b4493cc>

Düğmeci, D. (2022, July 6). Finding Shortest Path using Q-Learning Algorithm - Towards Data Science. Medium. <https://towardsdatascience.com/finding-shortest-path-using-q-learning-algorithm-1c1f39e89505>

Surma, G. (2021, October 13). Cartpole - Introduction to Reinforcement Learning (DQN - Deep Q-Learning). Medium. <https://gsurma.medium.com/cartpole-introduction-to-reinforcement-learning-ed0eb5b58288>

Manrique Escobar, C. A., Pappalardo, C. M., & Guida, D. (2020). A parametric study of a deep reinforcement learning control system applied to the swing-up problem of the cart-pole. Applied Sciences, 10(24), 9013.

Kumar, S. (2020). Balancing a CartPole System with Reinforcement Learning--A Tutorial. arXiv preprint arXiv:2006.04938.

Van Hasselt, H., Guez, A., & Silver, D. (2016, March). Deep reinforcement learning with double q-learning. In Proceedings of the AAAI conference on artificial intelligence (Vol. 30, No. 1).

Watkins, C. J., & Dayan, P. (1992). Q-learning. Machine learning, 8, 279-292.

Dearden, R., Friedman, N., & Russell, S. (1998). Bayesian Q-learning. Aaai/iaai, 1998, 761-768.

Aleksandar Haber. (2023, February 5). Introduction to OpenAI Gym (Gymnasium): Cart-Pole Environment - Reinforcement Learning Tutorial [Video]. YouTube. <https://www.youtube.com/watch?v=2sp_eucoX2I>

Admin. (2023, January 31). Cart Pole Control environment in OpenAI Gym (Gymnasium)- Introduction to OpenAI Gym – fusion of engineering, control, coding, machine learning, and science. <https://aleksandarhaber.com/cart-pole-control-environment-in-openai-gym-gymnasium-introduction-to-openai-gym/>

Admin. (2023b, February 10). Deep Q Networks (DQN) in Python from scratch by using OpenAI GyM and TensorFlow- Reinforcement Learning tutorial – fusion of engineering, control, coding, machine learning, and science. <https://aleksandarhaber.com/deep-q-networks-dqn-in-python-from-scratch-by-using-openai-gym-and-tensorflow-reinforcement-learning-tutorial/>