Configuration Manual

MSc Research Project
Data Analytics

Chetan Bhardwaj
Student ID: 20176724

School of Computing
National College of Ireland

Supervisor: Dr. Mohammed Hasanuzzaman

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Chetan Bhardwaj
Student ID:	20176724
Programme:	Data Analytics
Year:	2021
Module:	MSc Research Project
Supervisor:	Dr. Mohammed Hasanuzzaman
Submission Due Date:	16/12/2021
Project Title:	Configuration Manual
Word Count:	722
Page Count:	8

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	Chetan Bhardwaj
Date:	30th January 2022

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Chetan Bhardwaj
20176724

1 Introduction

This configuration manual is a formal document that contains parts that explain the hardware and software requirements, design details, operational information, implementation phases, and project settings.

2 System Configuration

2.1 Hardware Configuration

The research was conducted on local machine with the following hardware specifications AMD Ryzen 9 5900HS with Radeon Graphics 3.30 GHz, RAM 16.0 GB, System type 64-bit operating system, x64-based processor. Storage 1TB SSD.

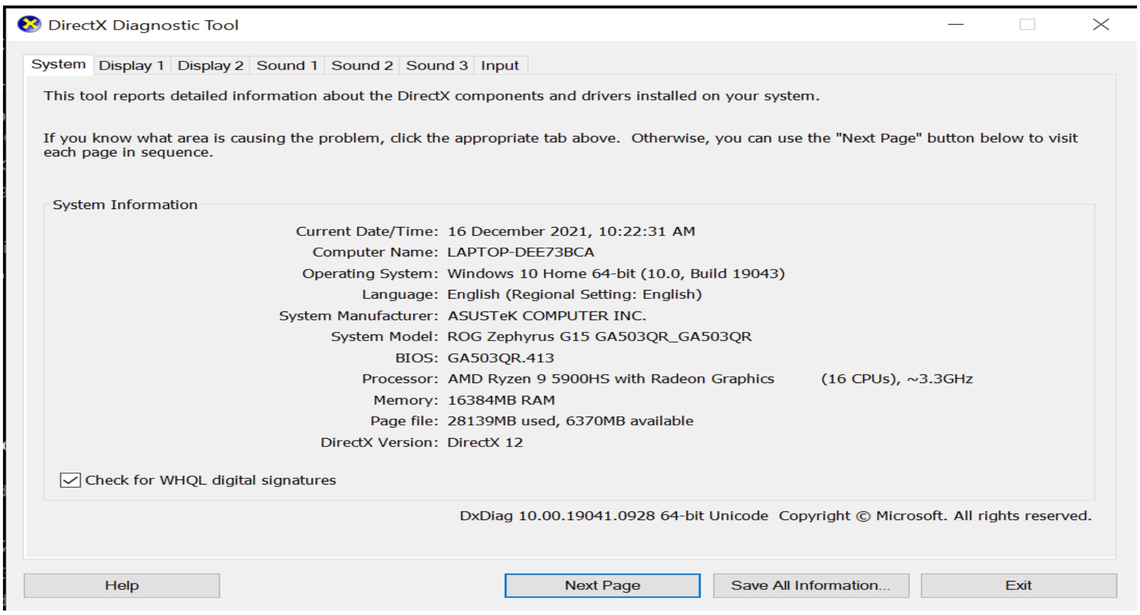

Figure 1: Hardware Configuration

2.2 Software Configuration

The research was conducted on Windows 10 Home, with Anaconda environment setup. Complete list of software is listed below.

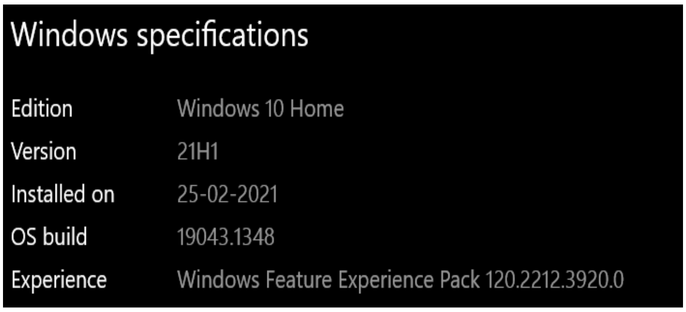

Figure 2: Windows Specification

- Anaconda for Windows (Version 2.0.1)
- Jupyter Notebook (Version 6.4.0)
- Python (Version 3.7.0)
- Tensorflow (Version 1.15)
- Stable Baselines(Version 2.10.2)
- OpenAI Gym (Version 0.21.0)

3 Setting up Environment

Installing the libraries for this research is a tricky task to accomplish. The version for Stable baselines 2.10.2 only works with tensorflow 1.x which was supported till Python 3.7.x hence the specified versions of the libraries were used. To install box2D-py on windows, it is essential to first build wheel for swig library, then only we can install box2d which is a dependency for gym library. This was done using the commands mentioned below:

- conda install swig
- pip install box2d-py

4 Open AI Gym Environment

4.1 Imports

To get started with the development, following modules from respective libraries were imported.

1 Imports

```
import time

import gym
import numpy as np

from stable_baselines import DQN, DDPG, SAC
from stable_baselines.common.vec_env import DummyVecEnv, VecFrameStack, SubprocVecEnv
from stable_baselines.common import set_global_seeds
from stable_baselines.common.evaluation import evaluate_policy
from stable_baselines.common.cmd_util import make_vec_env
from stable_baselines.bench import Monitor
from stable_baselines.results_plotter import load_results, ts2xy
from stable_baselines.common.noise import AdaptiveParamNoiseSpec
from stable_baselines.common.cmd_util import make_atari_env
from stable_baselines.common.policies import CnnPolicy, MlpPolicy, FeedForwardPolicy
from stable_baselines.common.noise import NormalActionNoise, OrnsteinUhlenbeckActionNoise, AdaptiveParamNoiseSpec
import os
```

executed in 4.32s, finished 14:50:41 2021-12-14

Figure 3: Required Imports

The environment used for simulation was taken from OpenAI Gym library, one of the biggest open-source library that provides simulation environments for Reinforcement Learning. To setup the environment we used below code, the environment used for DDPG and SAC were continuous state environments while it was a discrete state environment for DQN model.

```
env_2 = gym.make(env_id)
env_2 = Monitor(env_2, log_dir, allow_early_resets=True)
env_2 = DummyVecEnv([lambda: env_2])


# the noise objects for DDPG
n_actions = env_2.action_space.shape[-1]
param_noise = None
action_noise = OrnsteinUhlenbeckActionNoise(mean=np.zeros(n_actions), sigma=float(0.5) * np.ones(n_actions))
```

executed in 14ms, finished 15:04:26 2021-12-14

Figure 4: Code snippet to define Gym Environment

The image below shows a sample visual of the Lunar Lander environment.

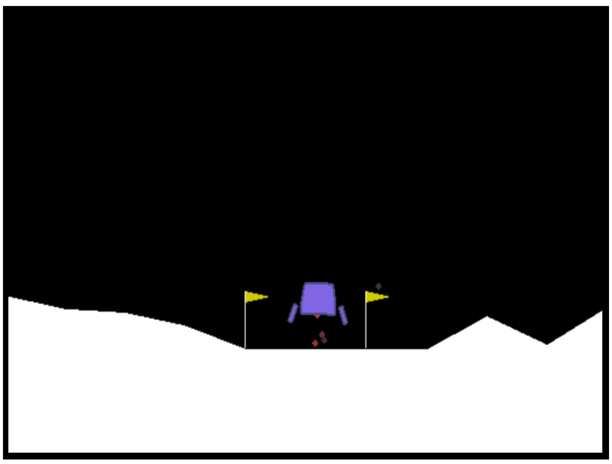

Figure 5: Lunar Lander environment of OpenAI Gym

5 Model Implementation

5.1 Implementing DQN Models

DQN class from stable_baselines was imported to create a DQN model. We used below code snippets to create models figure 6 with default hyperparameters while the model defined in figure 7 has tuned hyperparameters and an architecture of 256*256 Nodes in two hidden layers and a learning rate of 0.0001.

```
dqn_model_1 = DQN('MlpPolicy', env, verbose=1,
                  tensorboard_log=model_prog_dir, seed = 42)
```

executed in 742ms, finished 14:50:41 2021-12-14

Figure 6: DQN with default hyperparameters

```
dqn_model_2 = DQN('MlpPolicy', env,  learning_rate=0.0001, train_freq=1,
                  batch_size=256, policy_kwargs=dict(layers=[256,256]),
                  verbose=1, tensorboard_log=model_prog_dir,
                  seed = 42)
```

executed in 599ms, finished 14:50:42 2021-12-14

Figure 7: DQN with Tuned hyperparameters

5.2 Implementing DDPG Models

We used DDPG class from stable_baselines library to create an object of DDPG model. Following code snippets show definition of models with default and tuned hyperparameters.

```
ddpg_model_1 = DDPG('MlpPolicy', env_2, verbose=1, param_noise=param_noise,
                    action_noise=action_noise, tensorboard_log=model_prog_dir,
                    seed = 42)
```

executed in 453ms, finished 15:04:26 2021-12-14

Figure 8: DDPG with default hyperparameters

```
ddpg_model_2 = DDPG('MlpPolicy', env_2, param_noise=param_noise,
                    action_noise=action_noise, batch_size=256, buffer_size=50000,
                    verbose=1, tensorboard_log=model_prog_dir, seed = 42,
                    policy_kwargs=dict(layers=[256,256])
                   )
```

executed in 438ms, finished 15:04:26 2021-12-14

Figure 9: DDPG with Tuned hyperparameters

5.3 Implementing SAC Models

We used SAC class from stable_baselines library to create an object of SAC model. Following code snippets show definition of models with default and tuned hyperparameters.

```
sac_model_1 = SAC('MlpPolicy', env_2, verbose=1,
                  tensorboard_log=model_prog_dir,
                  seed = 42)
```
executed in 841ms, finished 15:18:10 2021-12-14

Figure 10: SAC with default hyperparameters

```
sac_model_2 = SAC('MlpPolicy', env_2, batch_size=256, verbose=1,
                  learning_rate=0.0001, buffer_size=50000,
                  policy_kwargs=dict(layers=[256, 256]),
                  seed=42, tensorboard_log=model_prog_dir
                 )
```
executed in 836ms, finished 15:18:11 2021-12-14

Figure 11: SAC with Tuned hyperparameters

6 Model Training

After defining the models, the models were trained for 100000 episodes. To prevent loss of training progress in models, a callback function was defined which helped in saving model checkpoints at interval of a number of episodes.

```
#Create a callback funtion to save checkpoints for models during training time
best_mean_reward, n_steps = -np.inf, 0

def callback(_locals, _globals):
    """
    Callback called at each step (for DQN an others) or after n steps (see ACER or PPO2)
    :param _locals: (dict)
    :param _globals: (dict)
    """
    global n_steps, best_mean_reward
    # Print stats every 1000 calls
    if (n_steps + 1) % 1000 == 0:
        # Evaluate policy performance
        x, y = ts2xy(load_results(log_dir), 'timesteps')
        if len(x) > 0:
            mean_reward = np.mean(y[-100:])
            print(x[-1], 'timesteps')
            print("Best mean reward: {:.2f} - Last mean reward per episode: {:.2f}".format(best_mean_reward, mean_re

            # New best model, you could save the agent here
            if mean_reward > best_mean_reward:
                best_mean_reward = mean_reward
                # Example for saving best model
                print("Saving new best model")
                _locals['self'].save(log_dir + 'best_model.pkl')
    n_steps += 1
    return True


# Create log dir
log_dir = "/tmp/gym/"
model_prog_dir = 'tmp/tensorboard/'
os.makedirs(log_dir, exist_ok=True)
os.makedirs(model_prog_dir, exist_ok=True)
```
executed in 14ms, finished 14:50:41 2021-12-14

Figure 12: Callback function

The following code snippet shows training of a DQN model for specified number of episodes. A similar code was used to train both DDPG and SAC models for same number of episodes.

```
n_timesteps = 100000
###DQN Training
start_time = time.time()
dqn_model_1.learn(n_timesteps, callback=callback)
total_time_single = time.time() - start_time

print("Took {:.2f}s for 1st DQN model - {:.2f} FPS".format(total_time_single, n_timesteps / total_time_single))
```
executed in 4m 46s, finished 14:55:29 2021-12-14

Figure 13: Training for DQN model

7 Model Evaluation and Visualizations

To evaluate the model performance in reinforcement learning, we compared mean reward earned by the models over a period of 100 episodes. The code snippet below shows the evaluation for the final three models.

```
eval_env_dqn = gym.make('LunarLander-v2')
eval_env = gym.make('LunarLanderContinuous-v2')
eval_env = Monitor(eval_env, log_dir, allow_early_resets=True)
eval_env = DummyVecEnv([lambda: eval_env])


mean_reward_dqn, std_reward_dqn = evaluate_policy(dqn_model_1, eval_env_dqn, n_eval_episodes=100)
mean_reward_ddpg, std_reward_ddpg = evaluate_policy(ddpg_model_2, eval_env, n_eval_episodes=100)
mean_reward_sac, std_reward_sac = evaluate_policy(sac_model_2, eval_env, n_eval_episodes=100)

print(f'Mean reward for DQN: {mean_reward_dqn} +/- {std_reward_dqn:.2f}')
print(f'Mean reward for DDPG: {mean_reward_ddpg} +/- {std_reward_ddpg:.2f}')
print(f'Mean reward for SAC: {mean_reward_sac} +/- {std_reward_sac:.2f}')
```

executed in 2m 42s, finished 02:13:57 2021-12-15

Figure 14: Snippet to evaluate Model Performance

We also used Tensorboard integration provided by Stable Baselines library which enabled us to monitor the training progress of models over episodes. Tensorboard provided with large number of loss visualisations for each algorithm, shown in the image below.

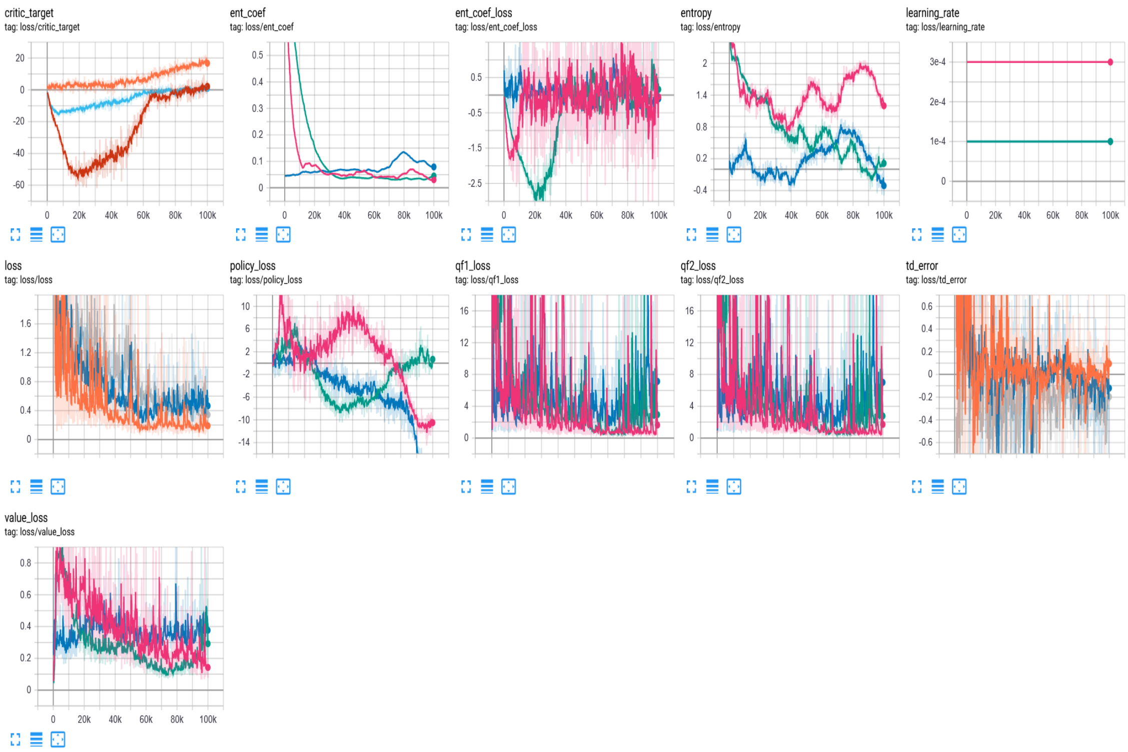

Figure 15: Loss Metrics across all models

The graph below shows the progress of every model over the training period against the reward earned by the model for every episode.

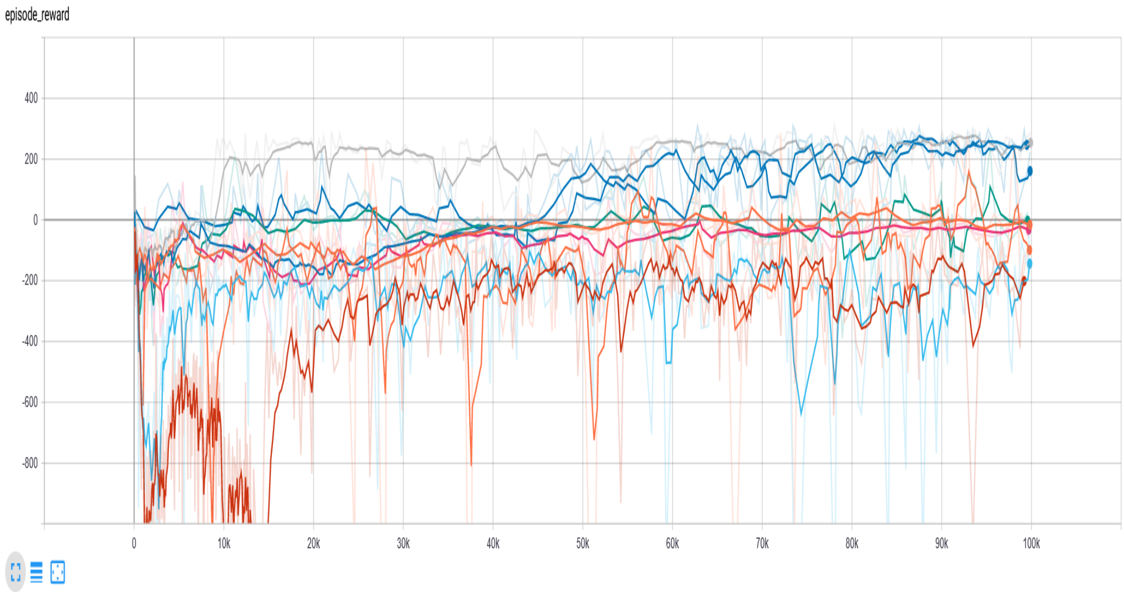

Figure 16: Rewards Earned by models during Training period

8 Saving and Simulating the model

After successful training, and selecting better performing model of the three algorithms, we first saved all three models and reloaded the models after clearing up and memory acquired by the older ones. Following code snippet depicts the same.

6 Saving the Best models

```
dqn_model_2.save('DQN_best_model')
ddpg_model_2.save('DDPG_best_model')
sac_model_2.save('SAC_best_model')
```

executed in 35ms, finished 17:20:31 2021-12-14

```
del dqn_model_1, dqn_model_2, ddpg_model_1, ddpg_model_2, sac_model_1, sac_model_2
```

Figure 17: Snippet to Save the model

After reloading the model, we simulated the agent in evaluation environment for 10000 iterations and visualize the model's performance in the environment.

7 Load Model ¶

```
sac_model_2 = SAC.load("SAC_best_model")
```

executed in 2.88s, finished 08:56:27 2021-12-16

```
Loading a model without an environment, this model cannot be trained until it has a valid environment.
```

8 SImulate Run in Environment

```
eval_env = gym.make('LunarLanderContinuous-v2')
eval_env = Monitor(eval_env, log_dir, allow_early_resets=True)
eval_env = DummyVecEnv([lambda: eval_env])
obs = eval_env.reset()
for _ in range(10000):
    action, _states = sac_model_2.predict(obs)
    obs, rewards, dones, info = eval_env.step(action)
    eval_env.render()
eval_env.close()
```

executed in 1m 7.77s, finished 00:44:49 2021-12-15

Figure 18: Snipped to Simulate Model

9 References

Code Reference of Model Implementation
https://stable-baselines.readthedocs.io/en/master/
https://gym.openai.com/envs/LunarLanderContinuous-v2/