Configuration Manual

MSc Research Project
MSc in Data Analytics

Simanta Sarkar
Student ID: x18201148

School of Computing
National College of Ireland

Supervisor: Dr. Muhammad Iqbal

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Simanta Sarkar		
Student ID:	18201148		
Programme:	MSc in Data Analytics	**Year:**	2019-2020
Module:	MSc Research Project		
Lecturer:	Dr. Muhammad Iqbal		
Submission Due Date:	17/08/2020		
Project Title:	Optimisation of Actor-Critic model in Continuous Action space …………………………………………………………………………………………………		
Word Count:	844……………………………… **Page Count: 9**…………………………………………		

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: Simanta Sarkar ………………………………………………………………………………

Date: 17/08/2020………………………………………………………………………………………

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

Simanta sarkar
Student ID: 18201148

1 Introduction

This configuration manual states the software and hardware used and explains the codes used in course of the research thesis “Optimisation of Actor-Critic model in Continuous Action space”.

2 System Configurations

2.1 Hardware

Processor: Intel(R) Core(TM) i5-7200U CPU @ 2.50GHz 2.70 GHz
RAM: 12 GB
System Type: Windows OS, 64-bit
GPU: Intel(R) UHD Graphics Family, NVIDEA Gforce 940mx 2GB
Storage: 256 GB SSD

2.2 Software

Anaconda Distribution:
The open source software which supports multiple platforms like spyder, jupyter notebooks, R studio and many more is downloaded and installed from Anaconda website. It provides a conda interface through which all package dependencies can be managed and installed.

Spyder: This is a open source IDE for the development of Python and scripted in Python. It provides an environment for scientific programming. Spyder is included in the Anaconda environment.

3 Project Development

3.1 Package Dependencies

Anaconda Installation[1] :

All packages required for the project implementation is handled by Anaconda which is installed from the Anaconda website.

[1] https://repo.anaconda.com/archive/Anaconda3-5.2.0-Linux-x86_64.sh

3.2 Gym Environment

Gym provides an environment for development of artificial intelligent agents. The Gym is developed and maintained by OpenAI(Brockman et al., 2016; Henderson et al., 2017). OpenAI is a research company which primarily focuses on AI development.

3.2.1 Cartpol

This is a Gym environment consisting of a cart and a pendulum connected through a moveable joint.

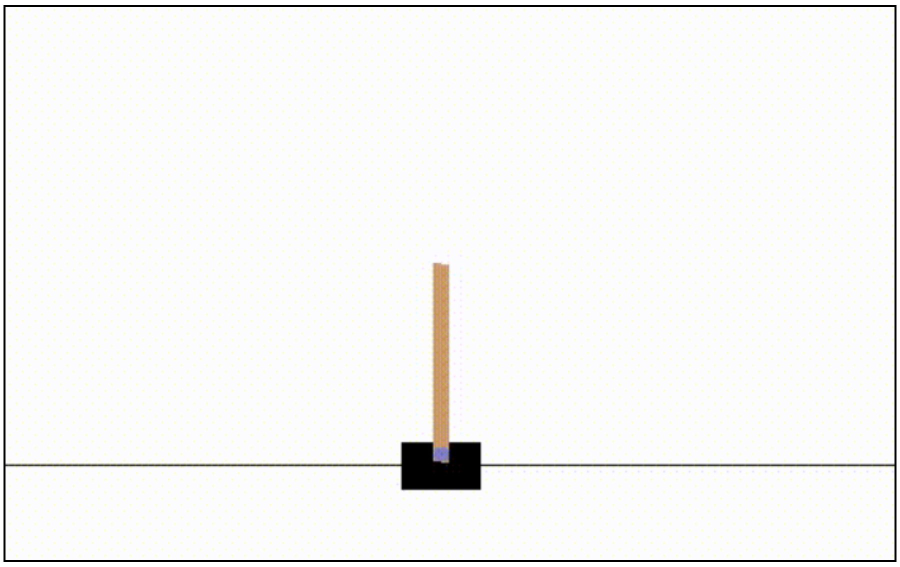
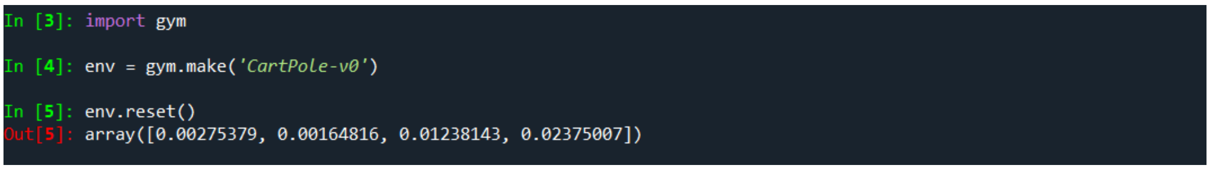


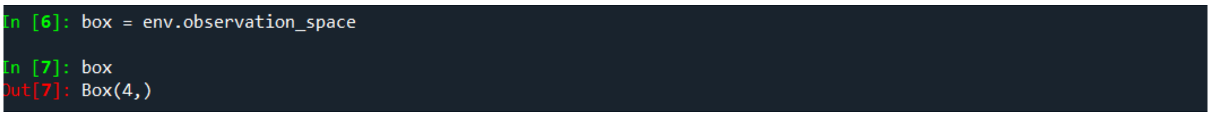
Figure 1: Cartpole environment

Different properties of Cartpole environment is illustrated below:



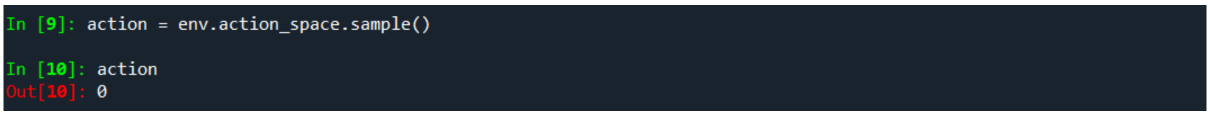
```
In [3]: import gym

In [4]: env = gym.make('CartPole-v0')

In [5]: env.reset()
Out[5]: array([0.00275379, 0.00164816, 0.01238143, 0.02375007])
```

Here Gym package is used to load the Cartpole environment. The reset() function is implemented to reset the initial cart position in the environment.



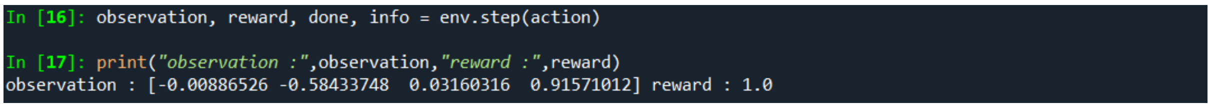
```
In [6]: box = env.observation_space

In [7]: box
Out[7]: Box(4,)
```

The the observation space of the cart is checked which is nothing but the state of the cart in the environment. Initial position gives four values.



```
In [9]: action = env.action_space.sample()

In [10]: action
Out[10]: 0
```

The random sampling from the action space is performed.



```
In [16]: observation, reward, done, info = env.step(action)

In [17]: print("observation :",observation,"reward :",reward)
observation : [-0.00886526 -0.58433748  0.03160316  0.91571012] reward : 1.0
```

The step() function is used to select a particular action which is used by the agent to interact with the environment.

3.2.2 MountainCar

MountainCar is a popularly used environment for testing reinforcement learning agent with an aim to climb the mountain top to reach the goal.

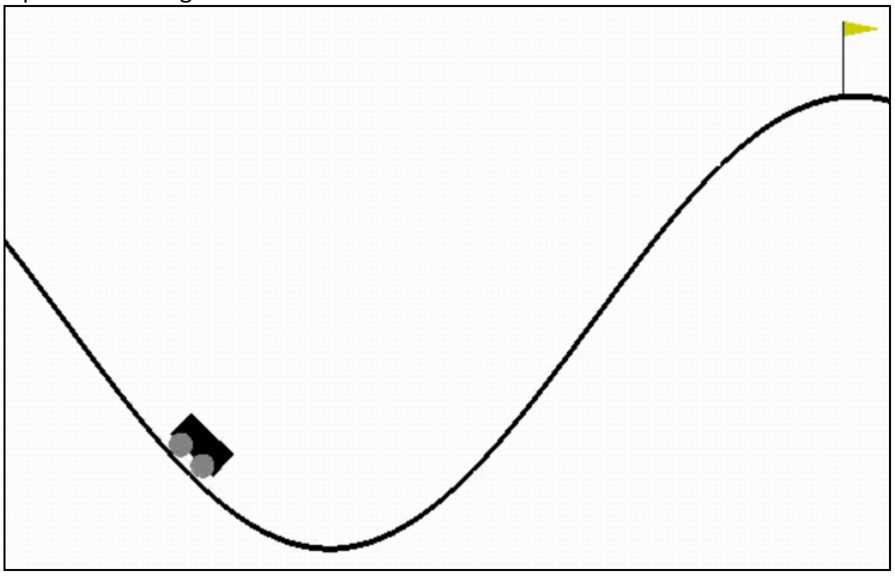
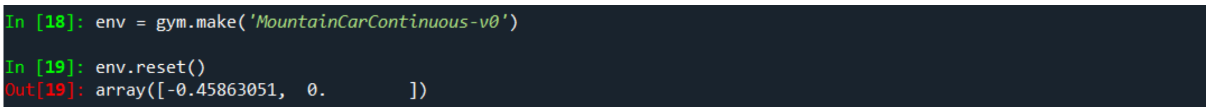
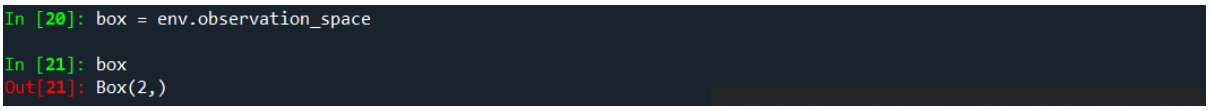


Figure 2: MountainCarContinuous environment



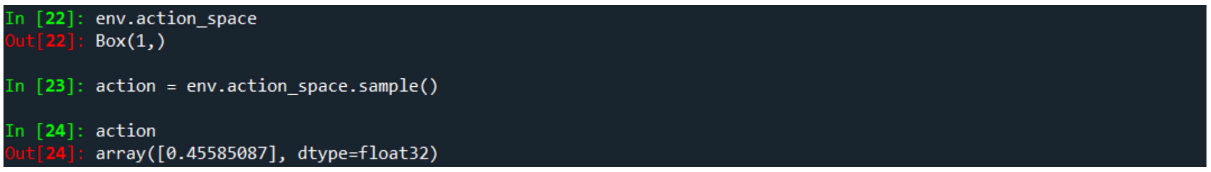
```
In [18]: env = gym.make('MountainCarContinuous-v0')

In [19]: env.reset()
Out[19]: array([-0.45863051,  0.        ])
```

Here Gym package is used to load the MountainCarContinuous environment. The reset() function is implemented to reset the initial car position in the environment.



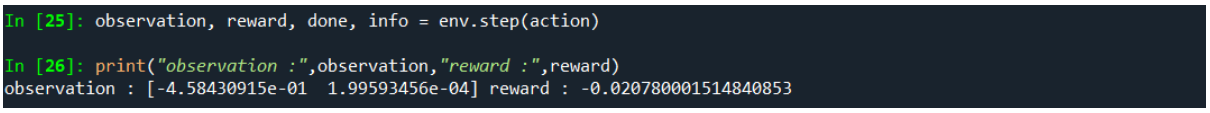
```
In [20]: box = env.observation_space

In [21]: box
Out[21]: Box(2,)
```

The the observation space of the Car is checked which is nothing but the state of the Car in the environment. Initial position gives two values.



```
In [22]: env.action_space
Out[22]: Box(1,)

In [23]: action = env.action_space.sample()

In [24]: action
Out[24]: array([0.45585087], dtype=float32)
```

The action values are printed to check if the values are continuous.

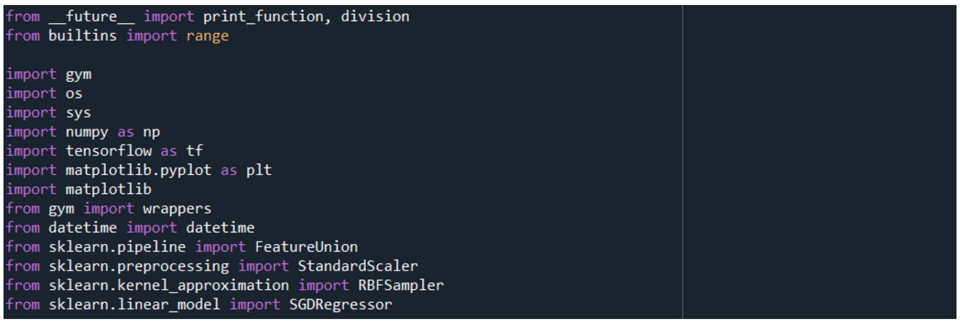


```
In [25]: observation, reward, done, info = env.step(action)

In [26]: print("observation :",observation,"reward :",reward)
observation : [-4.58430915e-01  1.99593456e-04] reward : -0.020780001514840853
```

The step() function is used to select a particular action which is used by the Car to interact with the environment.

3.3 Implementation:

3.3.1 packages used

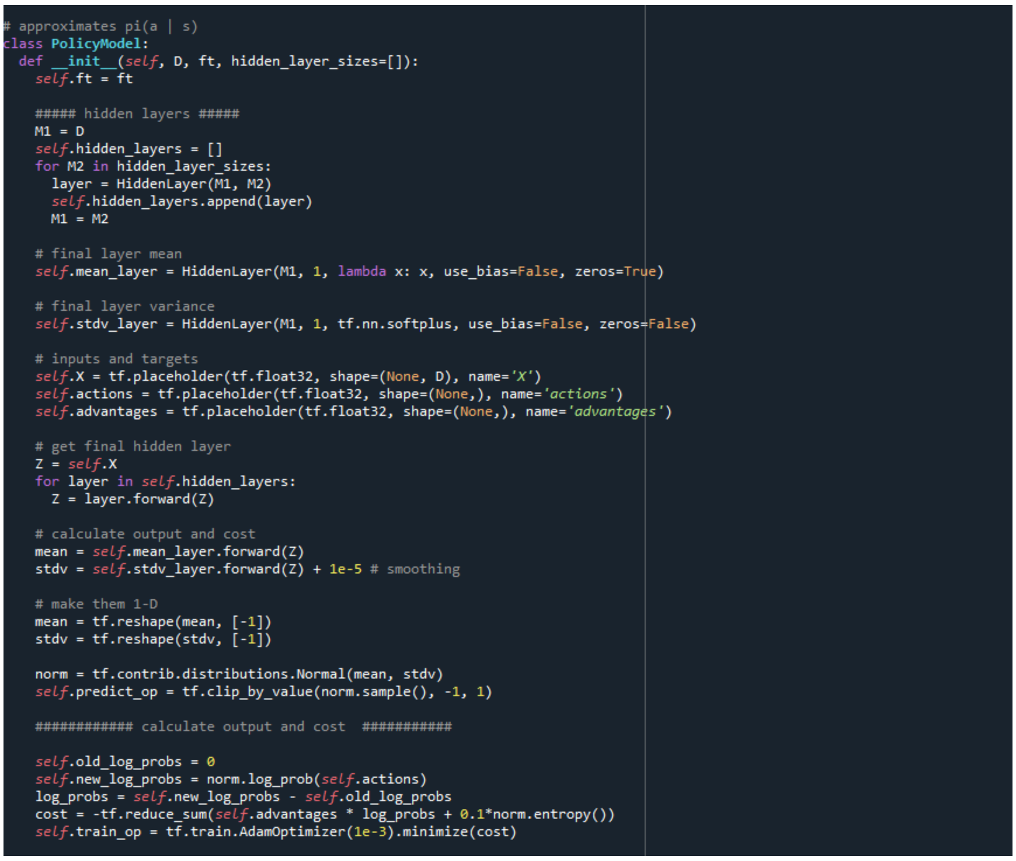


```
from __future__ import print_function, division
from builtins import range

import gym
import os
import sys
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
import matplotlib
from gym import wrappers
from datetime import datetime
from sklearn.pipeline import FeatureUnion
from sklearn.preprocessing import StandardScaler
from sklearn.kernel_approximation import RBFSampler
from sklearn.linear_model import SGDRegressor
```

All necessary packages for implementing actor-critic and modified actor-critic algorithm in Gym environment is installed.

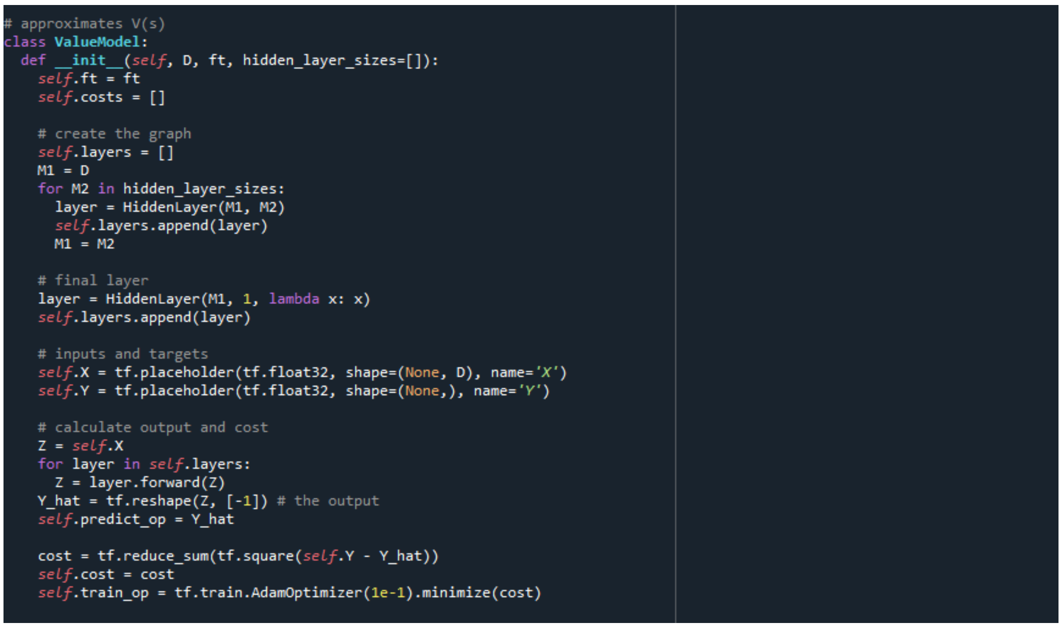
3.3.2 Actor Critic

For implementation of Actor-critic algorithm in Gym environment the coding section is broadly segmented in four sections namely policy model class, value model class, play 1TD and the main function.



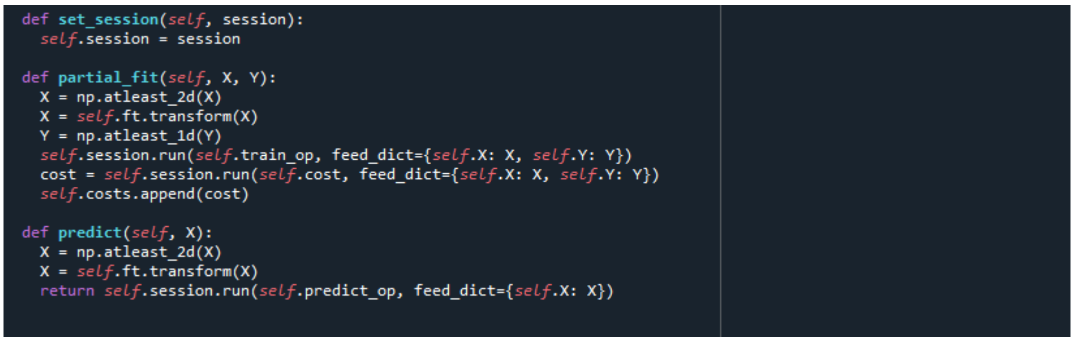
```
# approximates pi(a | s)
class PolicyModel:
  def __init__(self, D, ft, hidden_layer_sizes=[]):
    self.ft = ft

    ##### hidden layers #####
    M1 = D
    self.hidden_layers = []
    for M2 in hidden_layer_sizes:
      layer = HiddenLayer(M1, M2)
      self.hidden_layers.append(layer)
      M1 = M2

    # final layer mean
    self.mean_layer = HiddenLayer(M1, 1, lambda x: x, use_bias=False, zeros=True)

    # final layer variance
    self.stdv_layer = HiddenLayer(M1, 1, tf.nn.softplus, use_bias=False, zeros=False)

    # inputs and targets
    self.X = tf.placeholder(tf.float32, shape=(None, D), name='X')
    self.actions = tf.placeholder(tf.float32, shape=(None,), name='actions')
    self.advantages = tf.placeholder(tf.float32, shape=(None,), name='advantages')

    # get final hidden layer
    Z = self.X
    for layer in self.hidden_layers:
      Z = layer.forward(Z)

    # calculate output and cost
    mean = self.mean_layer.forward(Z)
    stdv = self.stdv_layer.forward(Z) + 1e-5 # smoothing

    # make them 1-D
    mean = tf.reshape(mean, [-1])
    stdv = tf.reshape(stdv, [-1])

    norm = tf.contrib.distributions.Normal(mean, stdv)
    self.predict_op = tf.clip_by_value(norm.sample(), -1, 1)

    ############ calculate output and cost  ###########

    self.old_log_probs = 0
    self.new_log_probs = norm.log_prob(self.actions)
    log_probs = self.new_log_probs - self.old_log_probs
    cost = -tf.reduce_sum(self.advantages * log_probs + 0.1*norm.entropy())
    self.train_op = tf.train.AdamOptimizer(1e-3).minimize(cost)
```

In the beginning of the clsass the int() function is called which initilises when the object of the class is created. First the neural network is loaded with m layers depending upon the input. Then the two output layers are added one for the mean and the other for the approximating the standard deviation. These parameters are used to build the normal distribution function. It is followed by sampling from the normal distribution which is used to select an action in the environment by the agent. This results in a new state and correspinding reward for taking that action.



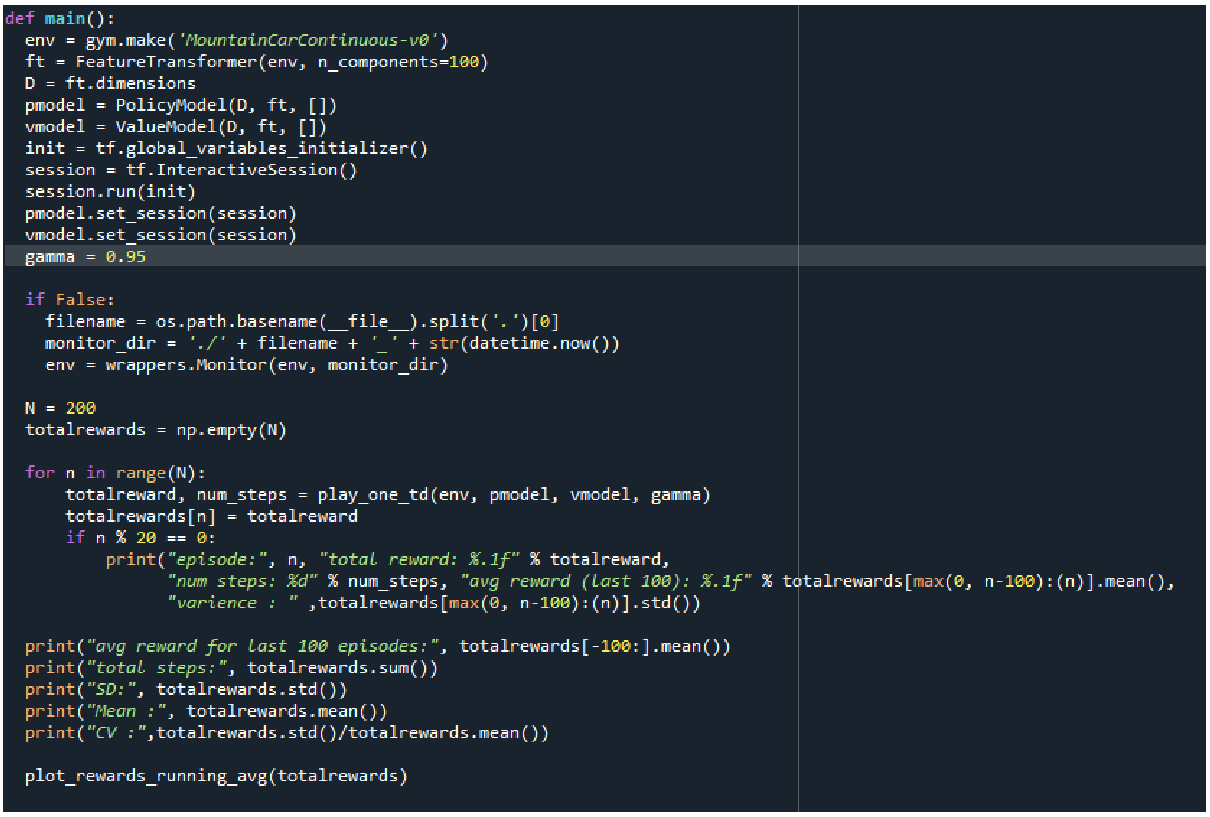
```
# approximates V(s)
class ValueModel:
  def __init__(self, D, ft, hidden_layer_sizes=[]):
    self.ft = ft
    self.costs = []

    # create the graph
    self.layers = []
    M1 = D
    for M2 in hidden_layer_sizes:
      layer = HiddenLayer(M1, M2)
      self.layers.append(layer)
      M1 = M2

    # final layer
    layer = HiddenLayer(M1, 1, lambda x: x)
    self.layers.append(layer)

    # inputs and targets
    self.X = tf.placeholder(tf.float32, shape=(None, D), name='X')
    self.Y = tf.placeholder(tf.float32, shape=(None,), name='Y')

    # calculate output and cost
    Z = self.X
    for layer in self.layers:
      Z = layer.forward(Z)
    Y_hat = tf.reshape(Z, [-1]) # the output
    self.predict_op = Y_hat

    cost = tf.reduce_sum(tf.square(self.Y - Y_hat))
    self.cost = cost
    self.train_op = tf.train.AdamOptimizer(1e-1).minimize(cost)
```

The class represents Critic netwrok in the Actor-critic algorithm which tell the policy netwrok how good a selected action is. The int() function of the code is similar to the policy network. The only difference is that it has only one output layer which is used to approximate the state value. For optimisation of the value network squared error is calculated for the predicted and the actual state values. The Adam optimiser is used to perform optimisation.



```
def set_session(self, session):
  self.session = session

def partial_fit(self, X, Y):
  X = np.atleast_2d(X)
  X = self.ft.transform(X)
  Y = np.atleast_1d(Y)
  self.session.run(self.train_op, feed_dict={self.X: X, self.Y: Y})
  cost = self.session.run(self.cost, feed_dict={self.X: X, self.Y: Y})
  self.costs.append(cost)

def predict(self, X):
  X = np.atleast_2d(X)
  X = self.ft.transform(X)
  return self.session.run(self.predict_op, feed_dict={self.X: X})
```

The above three functions are coomon to both the classes. Set_session() function is used to assign a tensorflow interactive session. The partial_fit() function fits the model for one update of the weights. The predict() function selects a particular action a state is provided.

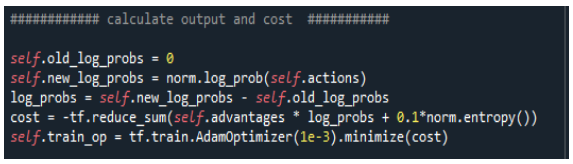


```python
def main():
  env = gym.make('MountainCarContinuous-v0')
  ft = FeatureTransformer(env, n_components=100)
  D = ft.dimensions
  pmodel = PolicyModel(D, ft, [])
  vmodel = ValueModel(D, ft, [])
  init = tf.global_variables_initializer()
  session = tf.InteractiveSession()
  session.run(init)
  pmodel.set_session(session)
  vmodel.set_session(session)
  gamma = 0.95

  if False:
    filename = os.path.basename(__file__).split('.')[0]
    monitor_dir = './' + filename + '_' + str(datetime.now())
    env = wrappers.Monitor(env, monitor_dir)

  N = 200
  totalrewards = np.empty(N)

  for n in range(N):
      totalreward, num_steps = play_one_td(env, pmodel, vmodel, gamma)
      totalrewards[n] = totalreward
      if n % 20 == 0:
          print("episode:", n, "total reward: %.1f" % totalreward,
                "num steps: %d" % num_steps, "avg reward (last 100): %.1f" % totalrewards[max(0, n-100):(n)].mean(),
                "varience : " ,totalrewards[max(0, n-100):(n)].std())

  print("avg reward for last 100 episodes:", totalrewards[-100:].mean())
  print("total steps:", totalrewards.sum())
  print("SD:", totalrewards.std())
  print("Mean :", totalrewards.mean())
  print("CV :",totalrewards.std()/totalrewards.mean())

  plot_rewards_running_avg(totalrewards)
```

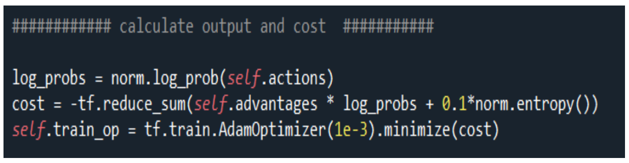
Inside the main() function the environment is initialised followed by creation of the interactive tensorflow session so that both the neural network can share the same session. Two objects namely PolicyModel() and ValueModel() class is created using which the agent tries to solve the environment.

3.3.3 Comparison between policy updates

Continuous:



```python
############ calculate output and cost  ###########

self.old_log_probs = 0
self.new_log_probs = norm.log_prob(self.actions)
log_probs = self.new_log_probs - self.old_log_probs
cost = -tf.reduce_sum(self.advantages * log_probs + 0.1*norm.entropy())
self.train_op = tf.train.AdamOptimizer(1e-3).minimize(cost)
```

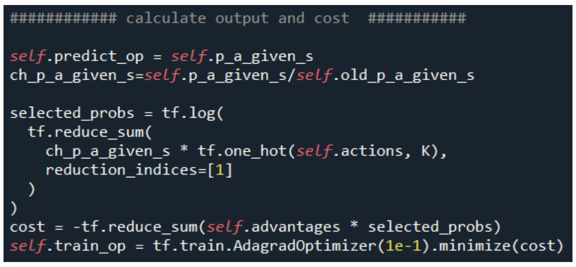


```python
############ calculate output and cost  ###########

log_probs = norm.log_prob(self.actions)
cost = -tf.reduce_sum(self.advantages * log_probs + 0.1*norm.entropy())
self.train_op = tf.train.AdamOptimizer(1e-3).minimize(cost)
```

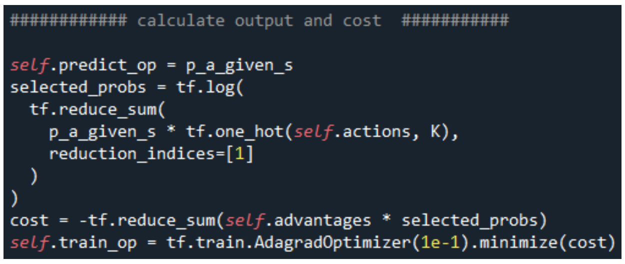
Figure 3 A) Modified Actor-Critic (left) B) Actor-Critic (right)

In continuous action space the difference between the cost calculation by actor-critic and modified actor-critic. Here, instead of using a ratio value of the old and current policy in the log function differences of the log value is taken as it represents the same.

Discreate:



```
############ calculate output and cost  ###########

self.predict_op = self.p_a_given_s
ch_p_a_given_s=self.p_a_given_s/self.old_p_a_given_s

selected_probs = tf.log(
  tf.reduce_sum(
    ch_p_a_given_s * tf.one_hot(self.actions, K),
    reduction_indices=[1]
  )
)
cost = -tf.reduce_sum(self.advantages * selected_probs)
self.train_op = tf.train.AdagradOptimizer(1e-1).minimize(cost)
```



```
############ calculate output and cost  ###########

self.predict_op = p_a_given_s
selected_probs = tf.log(
  tf.reduce_sum(
    p_a_given_s * tf.one_hot(self.actions, K),
    reduction_indices=[1]
  )
)
cost = -tf.reduce_sum(self.advantages * selected_probs)
self.train_op = tf.train.AdagradOptimizer(1e-1).minimize(cost)
```

Figure 4 A) Modified Actor-Critic (left) B) Actor-Critic (right)

In discrete action space the difference between the cost calculation by actor-critic and modified actor-critic. Here, the ratio value of the old and current policy in the log function is used.

References

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., & Zaremba, W. (2016). *OpenAI Gym*. 1–4. http://arxiv.org/abs/1606.01540

Henderson, P., Chang, W.-D., Shkurti, F., Hansen, J., Meger, D., & Dudek, G. (2017). *Benchmark Environments for Multitask Learning in Continuous Domains*. 1–6. http://arxiv.org/abs/1708.04352