Capstone Proposal

The project's domain background

```
– the field of research where the project is derived
```

Robotics is the application of AI that most excites me. Autonomous vehicles is one of the most direct applications of AI in robotics, and this intersection between development of real products and cutting edge research is one of the reasons I am passionate about my job in this field.

Day to day I spend most of my time on perception, mostly at the microwave wavelength at which radar sees the world. While this world hits both of my twin passions of physics and robotics, it is unfortunately hard to share, as sensor data is largely proprietary, and cannot be shared with publically.

Instead, I would like to tackle the other end of the autonomy problem, that of planning and control. To acheive this end, I plan to build upon the skills developed in the reinforcement learning section on the nanodegree for continous robotic control, specifically a DDPG agent (suited to continous control), while utilizing OpenAI's gym framework to solve a variety of control problems.

Finally, to keep the focus on something related to autonomous vehicles, once we have proven that an agent can learn simplier control tasks in "classic control", it will be unleashed on the the car racing simulation in the Box2D section of OpenAI's gym.

- https://gym.openai.com/envs/CarRacing-v0/ (https://gym.openai.com/envs/CarRacing-v0/)

A problem statement

```
– a problem being investigated for which a solution will be defined
```

How well can agents generalize between different tasks?

Specifically, how well does the DDPG agent generalize to a variety of other control tasks? What changes to reward function and agent are necessary, if any, to use the same agent to solve these tasks? Depending on the enviroment, what modifications to the agent are beneficial to solving a selection OpenAI's control environments.

Can we train an agent on other control tasks, then transfer some of this learning to a car racing environments? If possible, this will show the usefulness of the "transfer learning" approach often used in reinforcement leaning. At least, if not the weights and the input and output layers which are linked to the state and actions of the environment, can the general network architecture be shared?

If so, it allows us to train an agent in a simple, perhaps fast simulation, for many episodes, before training on a more complicated task.

At a minimum, the agent will be tested on the mountain car and cartpole simulations, before moving on to the car racing simulation.

Below are some references with OpenAI problems solved by actor-critic networks similar to the DDPG that will be attempted.

- [1] Actor-Critic Models with Keras and OpenAI: https://towardsdatascience.com/reinforcement-learning-w-keras-openai-actor-critic-models-f084612cfd69 (https://towardsdatascience.com/reinforcement-learning-w-keras-openai-actor-critic-models-f084612cfd69)
- [2] CONTINUOUS CONTROL WITH DEEP REINFORCEMENT LEARNING TimothyP.Lillicrap, JonathanJ.Hunt, AlexanderPritzel, NicolasHeess, TomErez, YuvalTassa, DavidSilver&DaanWierstra
- [3] SoftActor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor. TuomasHaarnoja, Aurick Zhou, Pieter Abbeel, Sergey Levine

Capstone Project Report

Note: "sample_" has been placed in front of classes and functions in this notebook to avoid confusing with the .py files containing the actual methods called for training agents in OpenAI.

1. Building an Open-AI Reinforcement Learning Framework

One of the goals of this project was to evaluate agent performance across different OpenAI enviroments. This was to see how well agents could generalize, as opposed to requiring re-tuning and re-architecting for every environment. In order to do this, the first challenge was decide on a software architecture for training reinforcement learning agents with OpenAI Gym.

There are many different versions of training code available for different environments and agents. Udacity provides several, and there are countless on online blogs and github repositories. Without a common design framework it is hard to combine these different resources into something that can be used for this project. Most of the source code found online seemed overly complex, brittle, hard to debug, and hard to modify.

With in mind, the following modules were developed to abstract away the reinforcement learning training process, and make it as general as possible for different agents and enviroments:

- Main: primary script. Manages agent and environment selection, examine enviroment, trigger interaction between the environment and the agent for training and testing, and plot results.
- Enviroment: In this case, OpenAI gym. The state and action sizes will be derived from here.
- Agent: The agent to train and test. The agent will need to have the ability to perform in both test and train mode. The agent will require the following functionality: act, learn, step, and reset (these are often blended together in other implementations, making switching agents and environments difficult, and modifying agents without breaking them troublesome).
- Interact: over a series of episodes, engage the agent with the enviroment in discrete time steps, in either training or test mode, compute the training time, plot actions and rewards selectively, and write the results to file.

1.1 Main Flowchat

1. Select enviroment (and optionally examine it)
2. Select agent (must be environment compatiable)
3. Interact environment and agent in training mode
4. Plot training results
5. Interact environment and agent in test mode
6. Plot test results
7. Close environment

In [1]:
```
import gym

"""
# Create an environment and set random seed
"""
selectedEnvironment = 6
env = 0
envName = 0

# Toy Text - Discrete state and action space
if selectedEnvironment == 0:
    envName = 'Taxi-v2'

# Classic Control - Continuous State and Discrete Action Spaces
elif selectedEnvironment == 1:
    envName = 'MountainCar-v0' # needs Discretized or better
elif selectedEnvironment == 2:
    envName = 'Acrobot-v1'     # needs Discretized, Tile Encoding or better
elif selectedEnvironment == 3:
    envName = 'CartPole-v1'    # needs Deep Q Learning to do well?

# Box 2D - Continuous State, Discrete Actions
elif selectedEnvironment == 4:
    envName = 'LunarLander-v2' # discrete actions, continuous state

# Classic Control - Continuous State and Action Spaces
elif selectedEnvironment == 5:
    envName = 'Pendulum-v0' # continuous only
elif selectedEnvironment == 6:
    envName = 'MountainCarContinuous-v0' # continuous only

# Box 2D - Continuous State and Action Spaces
elif selectedEnvironment == 7:
    envName = 'LunarLanderContinuous-v2' # continuous only
elif selectedEnvironment == 8:
    envName = 'BipedalWalker-v2'  # continuous only

# Box 2D - Image State and Continuous Action Spaces
elif selectedEnvironment == 9:
    envName = 'CarRacing-v0'      #  image input, actions [steer, gas, brake]

# Initialize the environment
env = gym.make(envName)
env.reset()
```

Out[1]:
```
array([-0.49785404,  0.        ])
```

In [2]:
```
# Basic inspection of the environment
def sample_examine_environment(env):

    # Run a random agent
    score = 0
    for t in range(250):
        action = env.action_space.sample()
        env.render()
        state, reward, done, _ = env.step(action)
        score += reward
        if done:
            break
    print('Final score:', score)
    env.close()

    # Explore state (observation) space
    print("State space:", env.observation_space)
    print("- low:", env.observation_space.low)
    print("- high:", env.observation_space.high)

    # Generate some samples from the state space
    print("State space samples:")
    print(np.array([env.observation_space.sample() for i in range(10)]))

    # Explore the action space
    print("Action space:", env.action_space)

    # Generate some samples from the action space
    print("Action space samples:")
    print(np.array([env.action_space.sample() for i in range(10)]))
```

1.2 Interact Flowchat

1. Initialize: setup writer, start time, best reward
2. Process episodes: agent acts, enviroment steps, agent learns (if in learn mode), agent steps
3. Monitor: print and plot select results (rewards, actions per step) to monitor progress

In [3]:
```
import sys
import numpy as np
import matplotlib.pyplot as plt
import csv
import time
import datetime

from visuals import plot_scores

def sample_interact(agent, env, num_episodes=20000, mode='train', file_output="
results.txt"):
    """Run agent in given reinforcement learning environment and return score
s."""

    # Save simulation results to a CSV file.
    labels = ['episode', 'timestep', 'reward']

    # Run the simulation, and save the results.
    with open(file_output, 'w') as csvfile:
        writer = csv.writer(csvfile)
        writer.writerow(labels)

        scores = []
        best_reward = -np.inf # keep track of the best reward across episodes
        all_start_time = time.time()

        for i_episode in range(1, num_episodes+1):
            # Initialize episode
            state = env.reset()        # reset environment
            agent.reset_episode(state) # reset agent
            episode_steps = 0 # Reset for the new episode
            episode_total_reward = 0   # total rewards per episode
            done = False
            actionList = []
            start_time = time.time()

            # Interact with the Environment in steps until done
            while not done:
                # 1. agent action given environment state
                #    assumes explore/exploit as part of agent design
                # 2. enviroment changes based on action
                # 3. (training mode) learn from environment feedback
                #    (new state, reward, done) to agent
                # 4. step the agent (forward with the new state)

                action = agent.act(state, mode)
                state, reward, done, info = env.step(action)

                if mode == 'train':
                    agent.learn(action, reward, state, done)

                agent.step(state)

                # render event 25 steps
                if(episode_steps % 25 == 0) and mode != 'train':
                    env.render()
                    print("\tstep: ", episode_steps, ", action:", action)

                # gather episode results until the end of the episode
                episode_total_reward += reward
                episode_steps += 1
                actionList.append(action)
```

In [4]:
```
def sample_plot_score_from_file(file_to_read):
    # Load simulation results from the .csv file
    results = pd.read_csv(file_to_read)

    # Total rewards for each episode
    episode_rewards_sum = results.groupby(['episode'])[['reward']].sum()
    smoothed_sum = episode_rewards_sum.rolling(25).mean()

    # plot the sum rewards
    plt.figure(4)
    plt.plot(episode_rewards_sum, label='sum rewards')
    plt.plot(smoothed_sum, label='running mean')
    plt.legend()
    axes = plt.gca()
    axes.set_ylim([-250,250])
    plt.show()
```

1.3 Handling Different Enviroment Types

The following environment types are available in OpenAI:

1. discrete states and actions.
2. discrete actions, continuous states.
3. continuous states continuous actions.
4. Image pixel states, continuous actions.

Therefore, in order to examine different agent performance across different environments, we need to ensure compatiability across these types, or at least make agents interchangable so that certain agents could be used for certain types of enviroment state and action spaces.

For example, being a discrete state space, the Taxi-v2 environment does not have the observation space low field, making it not fully compatiable with the examine_environment written from continuous state spaces and discrete/continuous action spaces.

Explore or Exploit

This decision was made part of the agent act class method to generalize across agents and enviroments, so each agent is responsible for their exploration policy as part of the agent design.

In [5]:
```
# 1. discrete states and actions

import gym

# Initialize the enviroment
env = gym.make('Taxi-v2') # continuous only
env.reset()

# Examine the environment
from visuals import examine_environment
examine_environment(env)
```

```
+---------+
|R: | : :G|
| : : : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+

+---------+
|R: | : :G|
| : : : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
  (Pickup)
+---------+
|R: | : :G|
| : : : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
  (North)
+---------+
|R: | : :G|
| : : : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
  (South)
+---------+
|R: | : :G|
| : : : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
  (Dropoff)
+---------+
|R: | : :G|
| : : : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
  (South)
+---------+
|R: | : :G|
| : : : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
  (Dropoff)
+---------+
|R: | : :G|
| : : : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
```

In [6]:
```
# 2. discrete actions, continuous states.

# Initialize the enviroment
env = gym.make('MountainCar-v0')
env.seed(505);
env.reset()

# Examine the environment
from visuals import examine_environment
examine_environment(env)
```

```
Final score: -100.0
State space: Box(2,)
State space samples:
[[-0.689  0.029]
 [-1.138 -0.061]
 [-0.135  0.04 ]
 [-0.477  0.01 ]
 [-0.635 -0.044]
 [-0.611  0.047]
 [-0.757 -0.016]
 [ 0.528  0.053]
 [-1.023 -0.005]
 [-0.451  0.063]]
Action space: Discrete(3)
Action space samples:
[1 0 2 0 0 0 0 2 2 2]
```

In [7]:
```
# 3. continuous states continuous actions.

# Initialize the enviroment
env = gym.make('MountainCarContinuous-v0') # continuous only
env.seed(505);
env.reset()

# Examine the environment
from visuals import examine_environment
examine_environment(env)
```

```
Final score: -3.76486647975512
State space: Box(2,)
State space samples:
[[-0.434 -0.066]
 [-0.911  0.055]
 [-0.03  -0.007]
 [-1.189  0.03 ]
 [-1.197 -0.068]
 [-1.028  0.044]
 [-0.59   0.05 ]
 [-0.11   0.038]
 [-0.777 -0.044]
 [ 0.165 -0.004]]
Action space: Box(1,)
Action space samples:
[[-0.851]
 [ 0.128]
 [-0.792]
 [ 0.218]
 [ 0.444]
 [-0.305]
 [ 0.001]
 [ 0.177]
 [-0.102]
 [-0.821]]
```

In [8]:
```
# 4. Image pixel states, continuous actions.

# Initialize the enviroment
env = gym.make('CarRacing-v0') # continuous only
env.reset()

# Examine the environment
from visuals import examine_environment
examine_environment(env)
```

```
Track generation: 1077..1361 -> 284-tiles track
retry to generate track (normal if there are not many of this messages)
Track generation: 1013..1270 -> 257-tiles track
Final score: 5.6250000000000355
State space: Box(96, 96, 3)
State space samples:
[[[[ 34  39  36]
   [ 88 177 110]
   [ 73  22 135]
   ...
   [100 214 161]
   [247 115  76]
   [253 105 162]]

  [[118 219  52]
   [224 234  42]
   [ 88 245  70]
   ...
   [102 142 189]
   [159 126 123]
   [  8 238   0]]

  [[231  39 200]
   [111 173 224]
   [183  83  50]
   ...
   [219  77 155]
   [ 47  21 111]
   [  2  12  62]]

  ...

  [[ 50  90  60]
   [126 177 224]
   [  2 239 164]
   ...
   [253 204 115]
   [213 204  22]
   [220  30 132]]

  [[ 52 228  58]
   [ 38 226 181]
   [ 47 150 124]
   ...
   [ 64 136 172]
   [234 138 130]
   [155 176   7]]

  [[ 76 115  84]
   [107  97 144]
   [125 210 206]
   ...
   [ 72 194   2]
   [132   8 239]
   [227  36 152]]]


 [[[210 145 197]
   [107 124 143]
   [124 241 160]
   ...
   [235  83 252]
   [ 29 204 242]
```

2. Agents Summary

2.1 Simple Q Learning Agent (Benchmark Agent)

The baseline agent for this project is modelled after the Q-Learning agent with state discretization provided by Udacity in a practice project. Small modifications have been made where necessary in order to align with the training architecture for this project.

A Q learning agent learns by comparing the expected reward with the actual reward. This approach mirrors that found in cognitive neuroscience on how learning works in biological systems [15].

This agent is designed to handle continuous state (with the agent discretizes), and discrete action spaces in OpenAI Gym, such as the discrete versions of Lunar Lander, Mountain Car, Acrobat, and Cartpole.

In [9]:
```
import numpy as np
from agents import discretize as dis

class sample_QLearningAgent:
    """Q-Learning agent that can act on a continuous state space by discretizin
g it."""

    def __init__(self, env, alpha=0.02, gamma=0.99,
                 epsilon=1.0, epsilon_decay_rate=0.9995, min_epsilon=.01, see
d=505):
        """Initialize variables, create grid for discretization."""
        # Environment info
        self.env = env
        state_grid = dis.create_uniform_grid(env.observation_space.low, env.obs
ervation_space.high, bins=(20, 20))
        self.state_grid = state_grid
        self.state_size = tuple(len(splits) + 1 for splits in self.state_grid)
# n-dimensional state space
        self.action_size = self.env.action_space.n  # 1-dimensional discrete ac
tion space
        self.seed = np.random.seed(seed)
        print("Environment:", self.env)
        print("State space size:", self.state_size)
        print("Action space size:", self.action_size)

        # Learning parameters
        self.alpha = alpha  # learning rate
        self.gamma = gamma  # discount factor
        self.epsilon = self.initial_epsilon = epsilon  # initial exploration ra
te
        self.epsilon_decay_rate = epsilon_decay_rate # how quickly should we de
crease epsilon
        self.min_epsilon = min_epsilon

        # Create Q-table
        self.q_table = np.zeros(shape=(self.state_size + (self.action_size,)))
        print("Q table size:", self.q_table.shape)

    def preprocess_state(self, state):
        """Map a continuous state to its discretized representation."""
        # TODO: Implement this
        return tuple(dis.discretize(state, self.state_grid))

    def reset_episode(self, state):
        """Reset variables for a new episode."""
        # Gradually decrease exploration rate
        self.epsilon *= self.epsilon_decay_rate
        self.epsilon = max(self.epsilon, self.min_epsilon)

        # Decide initial action
        self.last_state = self.preprocess_state(state)
        self.last_action = np.argmax(self.q_table[self.last_state])
        return self.last_action

    def reset_exploration(self, epsilon=None):
        """Reset exploration rate used when training."""
        self.epsilon = epsilon if epsilon is not None else self.initial_epsilon

    def act(self, state, mode):
        """Pick next action and update internal Q table (when mode != 'test
')."""
        state = self.preprocess_state(state)
```

In [10]:
```
import numpy as np
from agents import tile as tile

class sample_QTable:
    """Simple Q-table."""

    def __init__(self, state_size, action_size):
        """Initialize Q-table.

        Parameters
        ----------
        state_size : tuple
            Number of discrete values along each dimension of state space.
        action_size : int
            Number of discrete actions in action space.
        """
        self.state_size = state_size
        self.action_size = action_size

        # TODO: Create Q-table, initialize all Q-values to zero
        # Note: If state_size = (9, 9), action_size = 2, q_table.shape should b
e (9, 9, 2)
        self.q_table = np.zeros(shape=(self.state_size + (self.action_size,)))
        print("QTable(): size =", self.q_table.shape)


class sample_TiledQTable:
    """Composite Q-table with an internal tile coding scheme."""

    def __init__(self, low, high, tiling_specs, action_size):
        """Create tilings and initialize internal Q-table(s).

        Parameters
        ----------
        low : array_like
            Lower bounds for each dimension of state space.
        high : array_like
            Upper bounds for each dimension of state space.
        tiling_specs : list of tuples
            A sequence of (bins, offsets) to be passed to create_tilings() alon
g with low, high.
        action_size : int
            Number of discrete actions in action space.
        """
        self.tilings = tile.create_tilings(low, high, tiling_specs)
        self.state_sizes = [tuple(len(splits)+1 for splits in tiling_grid) for
tiling_grid in self.tilings]
        self.action_size = action_size
        self.q_tables = [QTable(state_size, self.action_size) for state_size in
self.state_sizes]
        print("TiledQTable(): no. of internal tables = ", len(self.q_tables))

    def get(self, state, action):
        """Get Q-value for given <state, action> pair.

        Parameters
        ----------
        state : array_like
            Vector representing the state in the original continuous space.
        action : int
            Index of desired action.

        Returns
```

2.2 Deep Q Network (Benchmark Agent)

This agent is similar to the simple Q learning agent, except that instead of a Q learning table a deep neural network is used.
The agent is based on sample code from the Reinforcement Learning section of Udacity's Machine Learning

In [11]:
```
import numpy as np
from collections import deque

class sample_Memory():
    def __init__(self, max_size=1000):
        self.buffer = deque(maxlen=max_size)

    def add(self, experience):
        self.buffer.append(experience)

    def sample(self, batch_size):
        idx = np.random.choice(np.arange(len(self.buffer)),
                               size=batch_size,
                               replace=False)
        return [self.buffer[ii] for ii in idx]


import tensorflow as tf

class sample_QNetwork:
    def __init__(self, learning_rate=0.0001, state_size=4,
                 action_size=2, hidden_size=64,
                 name='QNetwork'):

        # Memory parameters
        self.memory_size = 10000               # memory capacity
        self.batch_size = 20                   # experience mini-batch size
        self.pretrain_length = self.batch_size # number experiences to pretrain
the memory
        self.memory = Memory(max_size=self.memory_size)

        # state inputs to the Q-network
        with tf.variable_scope(name):
            # Network parameters
            self.hidden_size = hidden_size      # number of units in each Q-net
work hidden layer
            self.learning_rate = learning_rate  # Q-network learning rate

            self.inputs_ = tf.placeholder(tf.float32, [None, state_size], nam
e='inputs')

            # One hot encode the actions to later choose the Q-value for the ac
tion
            self.actions_ = tf.placeholder(tf.int32, [None], name='actions')
            one_hot_actions = tf.one_hot(self.actions_, action_size)

            # Target Q values for training
            self.targetQs_ = tf.placeholder(tf.float32, [None], name='target')

            # ReLU hidden layers
            self.fc1 = tf.contrib.layers.fully_connected(self.inputs_, hidden_s
ize)
            self.fc2 = tf.contrib.layers.fully_connected(self.fc1, hidden_size)

            # Linear output layer
            self.output = tf.contrib.layers.fully_connected(self.fc2, action_si
ze,
                                                            activation_fn=None)

            ### Train with loss (targetQ - Q)^2
            # output has length 2, for two actions. This next line chooses
            # one value from output (per row) according to the one-hot encoded
actions
```

2.3 DDPG (Primary Agent)

Reinforcement Learning agent using Deep Deterministic Policy Gradients.

This is an actor(policy)-critic(value) method, where the policy function used is deterministic, with noise added to produce the desired stochasticity in actions taken. The agent uses random memory buffer to de-correlate current actions and states from learned experiences.

DDPG agents are well suited to continuous action and state spaces, and will be the focus of this project.

Original Paper: Lillicrap, Timothy P., et al., 2015. Continuous Control with Deep Reinforcement Learning, https://arxiv.org/pdf/1509.02971.pdf (https://arxiv.org/pdf/1509.02971.pdf)

In [12]:
```
# replay buffer

import numpy as np

import random
from collections import namedtuple, deque

class sample_ReplayBuffer:
    """Fixed-size buffer to store experience tuples."""

    def __init__(self, buffer_size, batch_size):
        """Initialize a ReplayBuffer object.
        Params
        ======
            buffer_size: maximum size of buffer
            batch_size: size of each training batch
        """
        self.memory = deque(maxlen=buffer_size)  # internal memory (deque)
        self.batch_size = batch_size
        self.experience = namedtuple("Experience", field_names=["state", "actio
n", "reward", "next_state", "done"])

    def add(self, state, action, reward, next_state, done):
        """Add a new experience to memory."""
        e = self.experience(state, action, reward, next_state, done)
        self.memory.append(e)

    def sample(self, batch_size=64):
        """Randomly sample a batch of experiences from memory."""
        return random.sample(self.memory, k=self.batch_size)

    def __len__(self):
        """Return the current size of internal memory."""
        return len(self.memory)
```

In [13]:
```
# Actor

import keras
from keras import layers, models, optimizers
#from keras import backend as K
from keras import regularizers

class sample_Actor:
    """
    Actor (Policy) Model for DDPG
    """

    def __init__(self, state_size, action_size, action_low, action_high, netArc
h, learning_rate, dropout_rate):
        """Initialize parameters and build model.
        Params
        ======
            state_size (int): Dimension of each state
            action_size (int): Dimension of each action
            action_low (array): Min value of each action dimension
            action_high (array): Max value of each action dimension
        """
        self.state_size = state_size
        self.action_size = action_size
        self.action_low = action_low
        self.action_high = action_high
        self.action_range = self.action_high - self.action_low
        self.netArch = netArch # network architecture selection
        self.learning_rate = learning_rate
        self.dropout_rate = dropout_rate
        self.build_model()

        print("*** init actor ***")
        print("self.action_range: ", self.action_range)

    def build_model(self):
        """Build an actor (policy) network that maps states -> actions."""
        states = 0
        actions = 0

        if self.netArch =="Lillicrap":
            # Define input layer (states)
            states = keras.layers.Input(shape=(self.state_size,), name='states
')

            # Kernel initializer with fan-in mode and scale of 1.0
            kernel_initializer = keras.initializers.VarianceScaling(scale=1.0,
mode='fan_in', distribution='normal', seed=None)

            # Add hidden layers
            net = keras.layers.Dense(units=400, activation='elu', kernel_initia
lizer=kernel_initializer)(states)
            net = keras.layers.Dense(units=300, activation='elu', kernel_initia
lizer=kernel_initializer)(net)

            # Add final output layer with sigmoid activation
            raw_actions = keras.layers.Dense(units=self.action_size, activatio
n='sigmoid', name='raw_actions', kernel_initializer=kernel_initializer)(net)

        elif self.netArch =="UniformLayers":
            # Define input layer (states)
            states = keras.layers.Input(shape=(self.state_size,), name='states
')
```

```
Using TensorFlow backend.
```

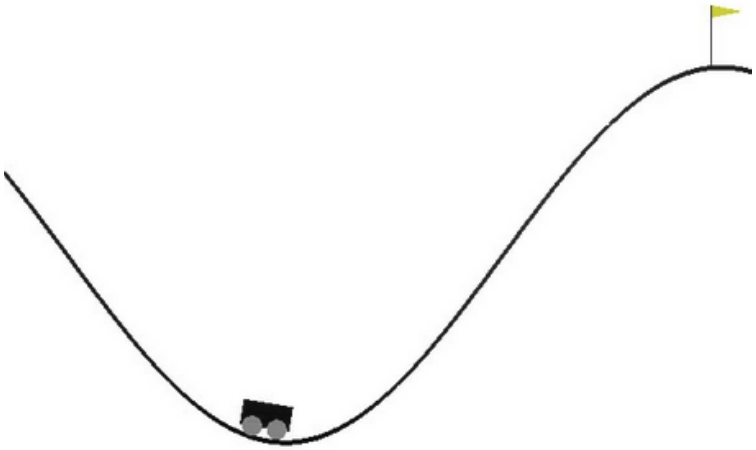
In [14]:
```
# Critic

class sample_Critic:
    """
    Critic (Value) Model for DDPG
    """

    def __init__(self, state_size, action_size, netArch, learning_rate, dropout
_rate):
        """Initialize parameters and build model.
        Params
        ======
            state_size (int): Dimension of each state
            action_size (int): Dimension of each action
        """
        self.state_size = state_size
        self.action_size = action_size
        self.netArch = netArch # network architecture selection
        self.learning_rate = learning_rate
        self.dropout_rate = dropout_rate
        self.build_model()

    def build_model(self):
        """Build a critic (value) network that maps (state, action) pairs -> Q-
values."""
        states = 0
        actions = 0
        Q_values = 0

        if self.netArch =="Lillicrap":
            # Define input layers. The critic model needs to map (state, actio
n) pairs to
            # their Q-values. This is reflected in the following input layers.
            states = keras.layers.Input(shape=(self.state_size,), name='states
')
            actions = keras.layers.Input(shape=(self.action_size,), name='actio
ns')

            # Kernel initializer with fan-in mode and scale of 1.0
            kernel_initializer = keras.initializers.VarianceScaling(scale=1.0,
mode='fan_in', distribution='normal', seed=None)

            # Add hidden layer(s) for state pathway
            net_states = keras.layers.Dense(units=400, activation='elu', kernel
_initializer=kernel_initializer)(states)

            # Add hidden layer(s) for action pathway
            net_actions = keras.layers.Dense(units=400, activation='elu', kerne
l_initializer=kernel_initializer)(actions)

            # Combine state and action pathways. The two layers can first be pr
ocessed via separate
            # "pathways" (mini sub-networks), but eventually need to be combine
d.
            net = keras.layers.Add()([net_states, net_actions])

            # Add more layers to the combined network if needed
            net = keras.layers.Dense(units=300, activation='elu', kernel_initia
lizer=kernel_initializer)(net)

        elif self.netArch =="UniformLayers":
            # Define input layers. The critic model needs to map (state, actio
n) pairs to
```

In [15]:
```
# DDPG Agent

"""
# Sets all pixel values to be between (0,1)
# Parameters:
# - image: A grayscale (nxmx1) or RGB (nxmx3) array of floats
# Outputs:
# - image rescaled so all pixels are between 0 and 1
"""
def sample_unit_image(image):
    return np.true_divide(image, 255.0)

"""
# Converts an RGB image to grayscale
# Parameters:
# - image: An RGB (nxmx3) array of floats
# Outputs:
# - A (nxmx1) array of floats in the range [0,255] representing a
#   weighted average of the color channels of 'image'
"""
def sample_grayscale_img(image):
    return np.dot(image[..., :3], [0.299, 0.587, 0.114])

"""
# scales the output actions from the network
# this is important for multi dimensional actions with different ranges and low
/hgh values
"""
def sample_scale_output(x, action_range, action_low):
    temp = (np.array(x) * np.array(action_range)) + np.array(action_low)
    return temp


class sample_DDPG():
    """
    Reinforcement Learning agent using Deep Deterministic Policy Gradients.

    This is an actor-critic method, where the policy function used is determini
stic
    in nature, with some noise added in externally to produce the desired
    stochasticity in actions taken.

    Original Paper:
    Lillicrap, Timothy P., et al., 2015. Continuous Control with Deep
    Reinforcement Learning, https://arxiv.org/pdf/1509.02971.pdf

    Actor, Critic, and ReplayBuffer are based on sample code from the Quadcopte
r Project
    in Udacity's Machine Learning Engineer nanodegree.

    Benchmark Implementation for OpenAI "MountainCarContinuous-v0" (DDPG):
    https://github.com/lirnli/OpenAI-gym-solutions/blob/master/Continuous_Deep_
Deterministic_Policy_Gradient_Net/DDPG%20Class%20ver2.ipynb

    Benchmark Implementation for OpenAI "Car Racing" (DDQN):
    https://github.com/AMD-RIPS/RL-2018/blob/master/documents/leaderboard/IPAM-
AMD-Car_Racing.ipynb

    Note that we will need two copies of each model - one local and one target.
    This is an extension of the "Fixed Q Targets" technique from Deep Q-Learnin
g,
    and is used to decouple the parameters being updated from the ones that are
    producing target values
```

3. Mountain Climber

In order to compare different networks for initial experimentation, the Mountain Climber environment was chosen. This enviroment is relatively simple, with only a single action. The action is either discretized into -1 and +1, or continuous, which will allow continuous action and discretized action agents to be compared on the same environment.

Because it is a relatively simple search space to solve (single action), it will serve a training ground for experiments of agent parameters and architectures without requiring too many episodes to solve.



3.1 Simple Q Learning Agent

The results of experiments on mountain climber for Q Learning Agent are below.

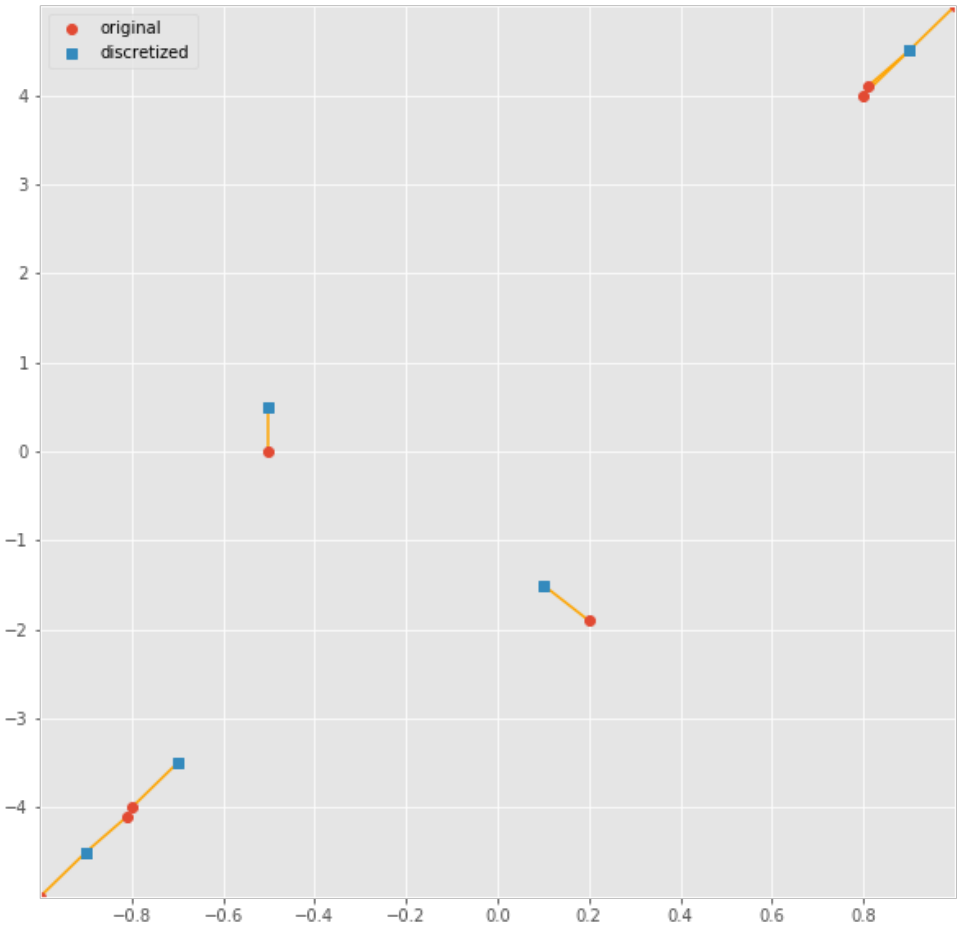
In [16]:
```
# First, let's take a look at the basic and tiled state discretization

from visuals import examine_environment_MountainCar_discretized, examine_enviro
nment_Acrobat_tiled

env = gym.make('MountainCar-v0')
env.seed(505);
env.reset()
examine_environment_MountainCar_discretized(env)

env = gym.make('Acrobot-v1')
env.seed(505);
env.reset()
examine_environment_Acrobat_tiled(env, 20)
```

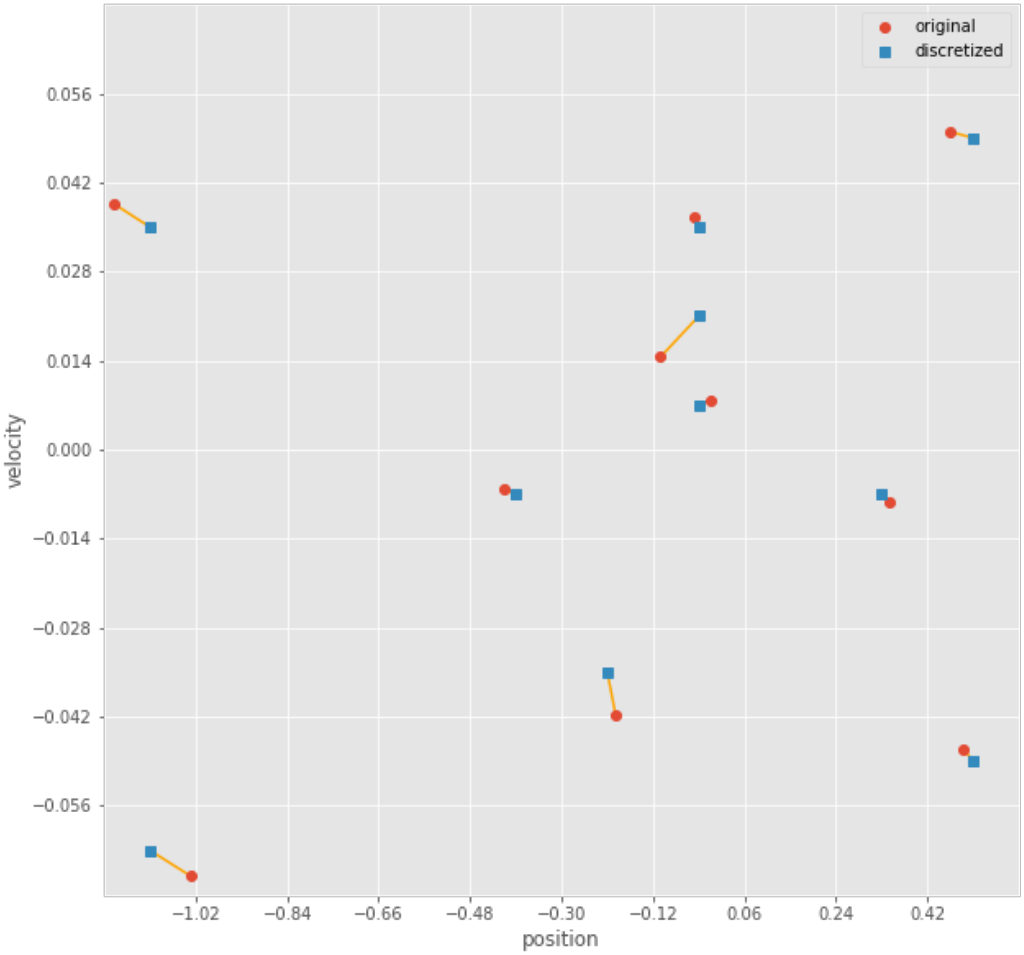
```
State space: Box(2,)
- low: [-1.2  -0.07]
- high: [0.6  0.07]
State space samples:
[[-1.189  0.013]
 [-0.191  0.024]
 [ 0.362  0.007]
 [-0.59  -0.044]
 [-0.956 -0.021]
 [-0.984 -0.011]
 [-0.455 -0.032]
 [-0.478 -0.005]
 [-0.59  -0.066]
 [-0.815  0.024]]
Action space: Discrete(3)
Action space samples:
[2 0 2 0 1 1 1 1 1 0]
Uniform grid: [<low>, <high>] / <bins> => <splits>
    [-1.0, 1.0] / 10 => [-0.8 -0.6 -0.4 -0.2  0.   0.2  0.4  0.6  0.8]
    [-5.0, 5.0] / 10 => [-4. -3. -2. -1.  0.  1.  2.  3.  4.]
Uniform grid: [<low>, <high>] / <bins> => <splits>
    [-1.0, 1.0] / 10 => [-0.8 -0.6 -0.4 -0.2  0.   0.2  0.4  0.6  0.8]
    [-5.0, 5.0] / 10 => [-4. -3. -2. -1.  0.  1.  2.  3.  4.]

Samples:
array([[-1.  , -5.  ],
       [-0.81, -4.1 ],
       [-0.8 , -4.  ],
       [-0.5 ,  0.  ],
       [ 0.2 , -1.9 ],
       [ 0.8 ,  4.  ],
       [ 0.81,  4.1 ],
       [ 1.  ,  5.  ]])

Discretized samples:
array([[0, 0],
       [0, 0],
       [1, 1],
       [2, 5],
       [5, 3],
       [9, 9],
       [9, 9],
       [9, 9]])
Uniform grid: [<low>, <high>] / <bins> => <splits>
    [-1.2000000476837158, 0.6000000238418579] / 10 => [-1.02 -0.84 -0.66 -0.48
-0.3  -0.12  0.06  0.24  0.42]
    [-0.07000000029802322, 0.07000000029802322] / 10 => [-0.056 -0.042 -0.028
-0.014  0.     0.014  0.028  0.042  0.056]
Tiling: [<low>, <high>] / <bins> + (<offset>) => <splits>
    [-1.0, 1.0] / 20 + (-0.1) => [-1.000e+00 -9.000e-01 -8.000e-01 -7.000e-01
-6.000e-01 -5.000e-01 -4.000e-01 -3.000e-01 -2.000e-01 -1.000e-01
  8.327e-17  1.000e-01  2.000e-01  3.000e-01  4.000e-01  5.000e-01  6.000e-01
7.000e-01  8.000e-01]
    [-5.0, 5.0] / 20 + (0.5) => [-4.  -3.5 -3.  -2.5 -2.  -1.5 -1.  -0.5  0.
0.5  1.   1.5  2.   2.5  3.   3.5  4.   4.5  5. ]
Tiling: [<low>, <high>] / <bins> + (<offset>) => <splits>
    [-1.0, 1.0] / 20 + (-0.066) => [-0.966 -0.866 -0.766 -0.666 -0.566 -0.466
-0.366 -0.266 -0.166 -0.066  0.034  0.134  0.234  0.334  0.434  0.534  0.634
  0.734  0.834]
    [-5.0, 5.0] / 20 + (-0.33) => [-4.83 -4.33 -3.83 -3.33 -2.83 -2.33 -1.83
-1.33 -0.83 -0.33  0.17  0.67  1.17  1.67  2.17  2.67  3.17  3.67  4.17]
Tiling: [<low>, <high>] / <bins> + (<offset>) => <splits>
    [-1.0, 1.0] / 20 + (0.0) => [-0.9 -0.8 -0.7 -0.6 -0.5 -0.4 -0.3 -0.2 -0.1
```

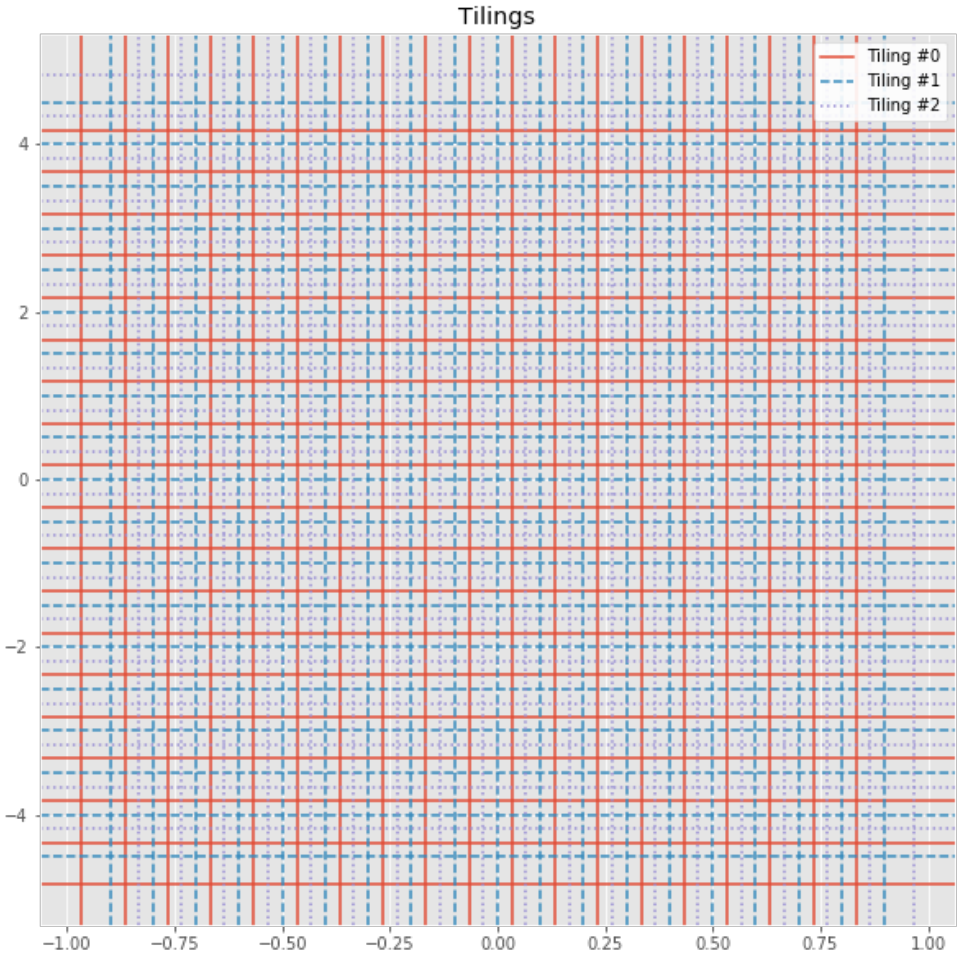
```
/home/tamanous/pj/udacityNanodegree/CapstoneProjectMLNanoDegree/agents/discreti
ze.py:80: FutureWarning: arrays to stack must be passed as a "sequence" type su
ch as list or tuple. Support for non-sequence iterables such as generators is d
eprecated as of NumPy 1.16 and will raise an error in the future.
  locs = np.stack(grid_centers[i, discretized_samples[:, i]] for i in range(len
(grid))).T  # map discretized samples
```

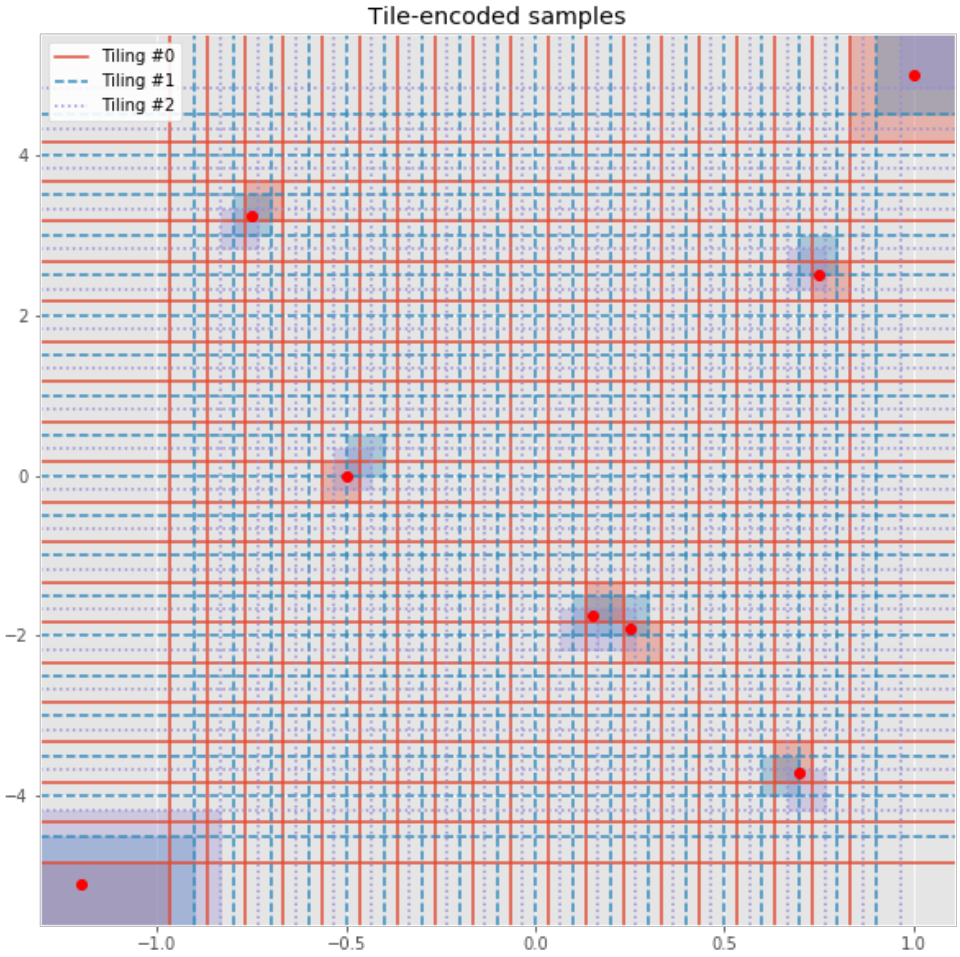
```
Samples:
[(-1.2, -5.1), (-0.75, 3.25), (-0.5, 0.0), (0.25, -1.9), (0.15, -1.75), (0.75,
2.5), (0.7, -3.7), (1.0, 5.0)]
```

```
Encoded samples:
[[(0, 0), (0, 0), (0, 0)], [(3, 17), (2, 16), (1, 15)], [(5, 10), (5, 10), (4,
9)], [(13, 6), (12, 6), (11, 5)], [(12, 7), (11, 6), (10, 5)], [(18, 15), (17,
15), (16, 14)], [(17, 3), (16, 2), (16, 1)], [(19, 19), (19, 19), (19, 19)]]
```




original
discretized
0.056
0.042
0.028
0.014
0.000
−0.014
−0.028
−0.042
−0.056
velocity
−1.02
−0.84
−0.66
−0.48
−0.30
−0.12
0.06
0.24
0.42
position

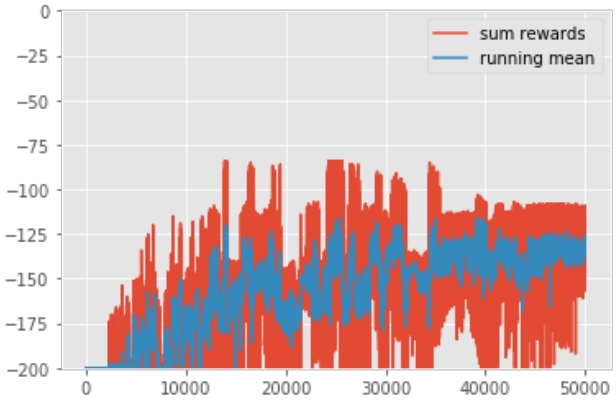

Tilings
Tiling #0
Tiling #1
Tiling #2
4
2
0
−2
−4
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00



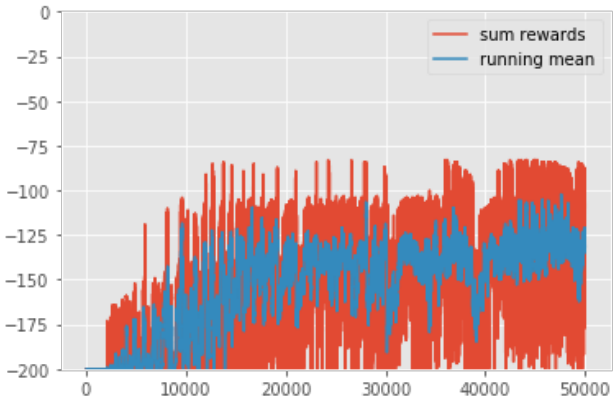
Results show both discretized and disretized with tiling agents able to solve mountain climber, though not quite reliably by the solve definition of -110 over 100 trials. The tiled approach seems to have a smoother, more stable, and overall higher reward profile.

In [17]:
```
from visuals import plot_score_from_file

# Plot Results from 50,000 Episodes, Basic Discretization
print("Basic Discretization")
dir = "results/MtClimber/discretized/"
plot_score_from_file(dir + "20190622223639MountainCar-v0_train.txt", -200, 0,
1)


# Plot Results from 50,000 Episodes, Tiled Discretization
print("Tiled Discretization")
dir = "results/MtClimber/tiled/"
plot_score_from_file(dir + "20190623105243MountainCar-v0_train.txt", -200, 0,
2)
```

```
Basic Discretization
```



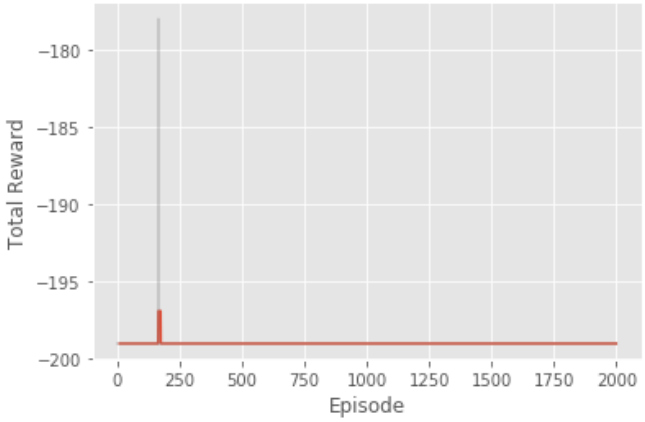

```
Tiled Discretization
```



3.2 Deep Q Network

The results of experiments on mountain climber for Q Network Agent are below.

It was not able to successfully solve the environment.



3.3 DDPG

The following section show the results of experiments for a Deep Deterministic Policy Gradient Agent to solve the Mountain Climber Continuous environment.

MountainCarContinuous-v0 defines "solving" as getting average reward of 90.0 over 100 consecutive trials.

Activation Function

" In contrast to ReLUs, ELUs have negativevalues which allows them to push mean unit activations closer to zero like batchnormalization but with lower computational complexity" [5]

ELU activation functions have gathered a lot of attention in supervised learning, with the promise of speed and better results both. A known solvable network of mountain climber using RELU activation function was changed to use ELU.

The network normally solves this environment between 50 an 100 episodes. However, with ELU, the network was unable to solve even after 250 episodes.

********** ********** Initializing DDPG Agent Environment: >> env.action_space.shape (1,) env.action_space.low [-1.] env.action_space.high [1.] *** init actor *** self.action_range: [2.] *** init actor *** self.action_range: [2.] ********** *** DDPG Agent Paramter *** - network architecture chosen: QuadCopterBigELU [ACTOR MODEL SUMMARY]
_____ Layer (type) Output Shape Param # === states (InputLayer) (None, 2) 0
_____ dense_1 (Dense) (None, 128) 384
_____ dense_2 (Dense) (None, 256) 33024
_____ dense_3 (Dense) (None, 256) 65792
_____ dense_4 (Dense) (None, 128) 32896
_____ raw_actions (Dense) (None, 1) 129
_____ actions (Lambda) (None, 1) 0
=== Total params: 132,225 Trainable params:

132,225 Non-trainable params: 0 _____ [CRITIC MODEL SUMMARY]

_____ Layer (type) Output Shape Param # Connected to

==========

states (InputLayer) (None, 2) 0

_____

actions (InputLayer) (None, 1) 0

_____

dense_9 (Dense) (None, 128) 384 states[0][0]

_____

dense_11 (Dense) (None, 128) 256 actions[0][0]

_____

dense_10 (Dense) (None, 256) 33024 dense_9[0][0]

_____

dense_12 (Dense) (None, 256) 33024 dense_11[0][0]

_____

add_1 (Add) (None, 256) 0 dense_10[0][0] dense_12[0][0]

_____

dense_13 (Dense) (None, 256) 65792 add_1[0][0]

_____

q_values (Dense) (None, 1) 257 dense_13[0][0]

==========

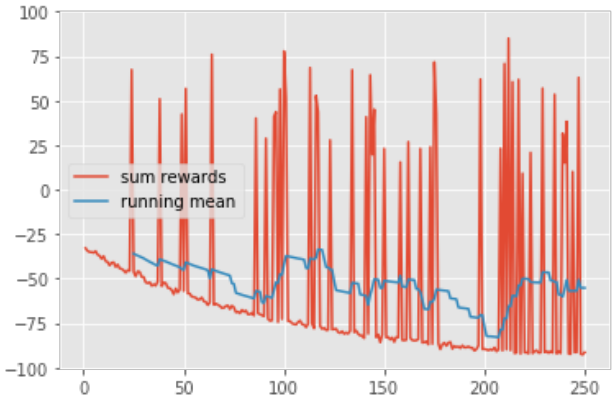
Total params: 132,737 Trainable params: 132,737 Non-trainable params: 0

_____ - action_repeat: 1 - learningRate: 0.0001 - learnFrequency: 1 - dropout_rate: 0.0 - buffer_size: 100000 - batch_size: 128 - gamma: 0.99 - useSoftUpdates: True - tau: 0.01 - explore_start: 1.0 - explore_stop: 0.01 - explore_decay_rate: 1e-05

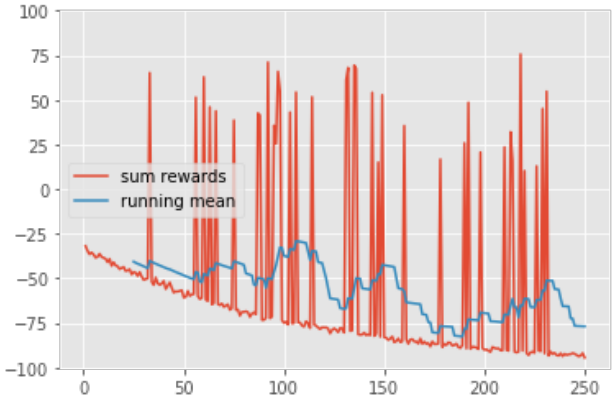
****

In [18]:
```
from visuals import plot_score_from_file

print("ELU Test 1")
dir = "results/MtClimberContinuous/elu/"
plot_score_from_file(dir + "20190622214124MountainCarContinuous-v0_train.txt",
-100, 100, 1)

print("ELU Test 2")
dir = "results/MtClimberContinuous/elu/"
plot_score_from_file(dir + "20190622214130MountainCarContinuous-v0_train.txt",
-100, 100, 1)
```

```
ELU Test 1
```




```
ELU Test 2
```

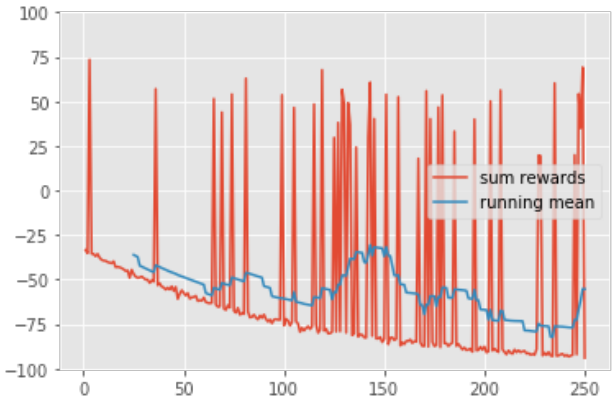


Initializers

Both Xavier and Variance scaling kernel initializers were attempted. Neither of these seemed to have a beneficial effect on learning. Therefore the random uniform initialize in Keras was maintained (https://keras.io/initializers/ (https://keras.io/initializers/)).

In [19]:
```
from visuals import plot_score_from_file

print("Xavier Initializer")
dir = "results/MtClimberContinuous/xavier/"
plot_score_from_file(dir + "20190622235529MountainCarContinuous-v0_train.txt",
-100, 100, 1)
```

```
Xavier Initializer
```



Network Size

Looking at three versions of the copter network, one original size, one 2x (big), and one 3x (max)

They all seem to be able to solve the environment. Generally there is an increase in training time for the larger networks to converge to a solution, though the actions by the network seems to have a more variety and sophisticated.

********** *** All episodes training time (HH:MM:SS): 0:30:43.334326 Average training time per episode: 9.21667163014412 ********** ---------- New Experiment, training output file name: 20190628162755MountainCarContinuous-v0_train.txt env.observation_space: (2,) ********** ********** Initializing DDPG Agent Environment: >> env.action_space.shape (1,) env.action_space.low [-1.] env.action_space.high [1.] *** init actor *** self.action_range: [2.] *** init actor *** self.action_range: [2.] 2019-06-28 16:27:56.443217: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1512] Adding visible gpu devices: 0 2019-06-28 16:27:56.443255: I tensorflow/core/common_runtime/gpu/gpu_device.cc:984] Device interconnect StreamExecutor with strength 1 edge matrix: 2019-06-28 16:27:56.443270: I tensorflow/core/common_runtime/gpu/gpu_device.cc:990] 0 2019-06-28 16:27:56.443276: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1003] 0: N 2019-06-28 16:27:56.443352: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1115] Created TensorFlow device (/job:localhost/replica:0/task:0 /device:GPU:0 with 10254 MB memory) -> physical GPU (device: 0, name: GeForce GTX 1080 Ti, pci bus id: 0000:01:00.0, compute capability: 6.1) ********** *** DDPG Agent Paramter *** - network architecture chosen: QuadCopter [ACTOR MODEL SUMMARY] _____ Layer (type) Output Shape Param # ========== states (InputLayer) (None, 2) 0 _____ dense_1 (Dense) (None, 64) 192 _____ dropout_1 (Dropout) (None, 64) 0 _____ dense_2 (Dense) (None, 128) 8320 _____ dropout_2 (Dropout) (None, 128) 0 _____ dense_3 (Dense) (None, 128) 16512 _____ dropout_3 (Dropout) (None, 128) 0 _____ dense_4 (Dense) (None, 64) 8256 _____ dropout_4 (Dropout) (None, 64) 0 _____ raw_actions (Dense) (None, 1) 65 ========== Total params: 33,345 Trainable params: 33,345 Non-trainable params: 0 _____ [CRITIC

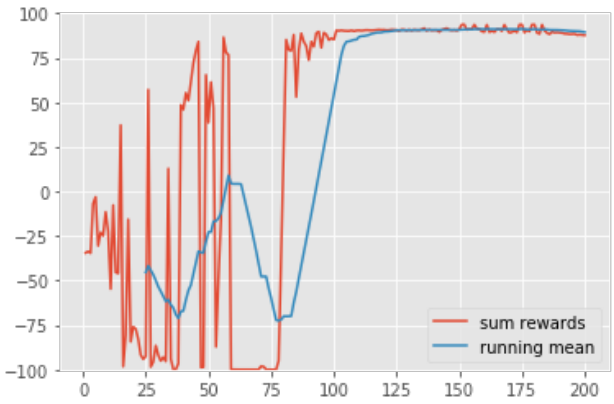
MODEL SUMMARY]
_____ Layer (type) Output Shape Param # Connected to
==========
states (InputLayer) (None, 2) 0
_____
actions (InputLayer) (None, 1) 0
_____
dense_9 (Dense) (None, 64) 192 states[0][0]
_____
dense_11 (Dense) (None, 64) 128 actions[0][0]
_____
dropout_9 (Dropout) (None, 64) 0 dense_9[0][0]
_____
dropout_11 (Dropout) (None, 64) 0 dense_11[0][0]
_____
dense_10 (Dense) (None, 128) 8320 dropout_9[0][0]
_____
dense_12 (Dense) (None, 128) 8320 dropout_11[0][0]
_____
dropout_10 (Dropout) (None, 128) 0 dense_10[0][0]
_____
dropout_12 (Dropout) (None, 128) 0 dense_12[0][0]
_____
add_1 (Add) (None, 128) 0 dropout_10[0][0] dropout_12[0][0]
_____
dense_13 (Dense) (None, 32) 4128 add_1[0][0]
_____
q_values (Dense) (None, 1) 33 dense_13[0][0]
==========
Total params: 21,121 Trainable params: 21,121 Non-trainable params: 0
_____ - action_repeat: 1 - learningRate: 0.0001 - learnFrequency: 1 - dropout_rate: 0.2 - buffer_size: 1000000 - batch_size: 32 - gamma: 0.99 - useSoftUpdates: True - tau: 0.01 - explore_start: 1.0 - explore_stop: 0.0 - explore_decay_rate: 0.0 ********** *** All episodes training time (HH:MM:SS): 0:58:43.658668 Average training time per episode: 17.618293339014052 ********** ---------- New Experiment, training output file name: 20190628162827MountainCarContinuous-v0_train.txt env.observation_space: (2,) ********** ********** Initializing DDPG Agent Environment: >> env.action_space.shape (1,) env.action_space.low [-1.] env.action_space.high [1.] *** init actor *** self.action_range: [2.] *** init actor *** self.action_range: [2.] ********** *** DDPG Agent Paramter *** - network architecture chosen: QuadCopterBig [ACTOR MODEL SUMMARY] _____ Layer (type) Output Shape Param # ========== states (InputLayer) (None, 2) 0 _____ dense_1 (Dense) (None, 128) 384 _____ dropout_1 (Dropout) (None, 128) 0 _____ dense_2 (Dense) (None, 256) 33024 _____ dropout_2 (Dropout) (None, 256) 0 _____ dense_3 (Dense) (None, 256) 65792 _____ dropout_3 (Dropout) (None, 256) 0 _____ dense_4 (Dense) (None, 128) 32896 _____ dropout_4 (Dropout) (None, 128) 0 _____ raw_actions (Dense) (None, 1) 129 ========== Total params: 132,225 Trainable params: 132,225 Non-trainable params: 0 _____ [CRITIC

MODEL SUMMARY]
_____ Layer
(type) Output Shape Param # Connected to
==========
states (InputLayer) (None, 2) 0
_____
actions (InputLayer) (None, 1) 0
_____
dense_9 (Dense) (None, 128) 384 states[0][0]
_____
dense_11 (Dense) (None, 128) 256 actions[0][0]
_____
dropout_9 (Dropout) (None, 128) 0 dense_9[0][0]
_____
dropout_11 (Dropout) (None, 128) 0 dense_11[0][0]
_____
dense_10 (Dense) (None, 256) 33024 dropout_9[0][0]
_____
dense_12 (Dense) (None, 256) 33024 dropout_11[0][0]
_____
dropout_10 (Dropout) (None, 256) 0 dense_10[0][0]
_____
dropout_12 (Dropout) (None, 256) 0 dense_12[0][0]
_____
add_1 (Add) (None, 256) 0 dropout_10[0][0] dropout_12[0][0]
_____
dense_13 (Dense) (None, 256) 65792 add_1[0][0]
_____
q_values (Dense) (None, 1) 257 dense_13[0][0]
==========
Total params: 132,737 Trainable params: 132,737 Non-trainable params: 0
_____ -
action_repeat: 1 - learningRate: 0.0001 - learnFrequency: 1 - dropout_rate: 0.2 - buffer_size: 1000000 - batch_size: 32 - gamma: 0.99 - useSoftUpdates: True - tau: 0.01 - explore_start: 1.0 - explore_stop: 0.0 - explore_decay_rate: 0.0
**********----------
---------- New Experiment, training output file name: 20190628173605MountainCarContinuous-v0_train.txt env.observation_space: (2,) **********
********** Initializing DDPG Agent Environment: >> env.action_space.shape (1,) env.action_space.low [-1.] env.action_space.high [1.] *** init actor *** self.action_range: [2.] *** init actor *** self.action_range: [2.]
********** *** DDPG Agent Paramter *** - network architecture chosen: QuadCopterMax [ACTOR MODEL SUMMARY] _____ Layer (type) Output Shape Param # ========== states (InputLayer) (None, 2) 0
_____ dense_19 (Dense) (None, 192) 576
_____ dropout_17 (Dropout) (None, 192) 0
_____ dense_20 (Dense) (None, 384) 74112
_____ dropout_18 (Dropout) (None, 384) 0
_____ dense_21 (Dense) (None, 384) 147840
_____ dropout_19 (Dropout) (None, 384) 0
_____ dense_22 (Dense) (None, 192) 73920
_____ dropout_20 (Dropout) (None, 192) 0
_____ raw_actions (Dense) (None, 1) 193
========== Total params: 296,641 Trainable params: 296,641 Non-trainable params: 0 _____ [CRITIC MODEL SUMMARY]

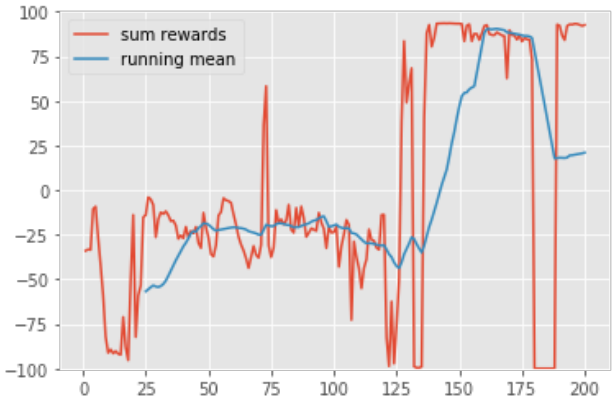
```
_________________________________________________________________ Layer
(type) Output Shape Param # Connected to
==================================================================================================
states (InputLayer) (None, 2) 0
_________________________________________________________________
actions (InputLayer) (None, 1) 0
_________________________________________________________________
dense_27 (Dense) (None, 192) 576 states[0][0]
_________________________________________________________________
dense_29 (Dense) (None, 192) 384 actions[0][0]
_________________________________________________________________
dropout_25 (Dropout) (None, 192) 0 dense_27[0][0]
_________________________________________________________________
dropout_27 (Dropout) (None, 192) 0 dense_29[0][0]
_________________________________________________________________
dense_28 (Dense) (None, 384) 74112 dropout_25[0][0]
_________________________________________________________________
dense_30 (Dense) (None, 384) 74112 dropout_27[0][0]
_________________________________________________________________
dropout_26 (Dropout) (None, 384) 0 dense_28[0][0]
_________________________________________________________________
dropout_28 (Dropout) (None, 384) 0 dense_30[0][0]
_________________________________________________________________
add_3 (Add) (None, 384) 0 dropout_26[0][0] dropout_28[0][0]
_________________________________________________________________
dense_31 (Dense) (None, 384) 147840 add_3[0][0]
_________________________________________________________________
q_values (Dense) (None, 1) 385 dense_31[0][0]
==================================================================================================
Total params: 297,409 Trainable params: 297,409 Non-trainable params: 0
_________________________________________________________________ -
action_repeat: 1 - learningRate: 0.0001 - learnFrequency: 1 - dropout_rate: 0.2 - buffer_size: 1000000 - batch_size: 32 - gamma:
0.99 - useSoftUpdates: True - tau: 0.01 - explore_start: 1.0 - explore_stop: 0.0 - explore_decay_rate: 0.0
**************************************
```

In [20]:
```
from visuals import plot_score_from_file

# Copter
print("Copter")
dir = "results/MtClimberContinuous/networkSize/copter/"
plot_score_from_file(dir + "20190628162755MountainCarContinuous-v0_train.txt",
-100, 100, 1)

# Copter Big
print("CopterBig")
dir = "results/MtClimberContinuous/networkSize/copterbig/"
plot_score_from_file(dir + "20190628162827MountainCarContinuous-v0_train.txt",
-100, 100, 1)

# Copter Max
print("CopterMax")
dir = "results/MtClimberContinuous/networkSize/coptermax/batch32/"
plot_score_from_file(dir + "20190628173605MountainCarContinuous-v0_train.txt",
-100, 100, 1)
```

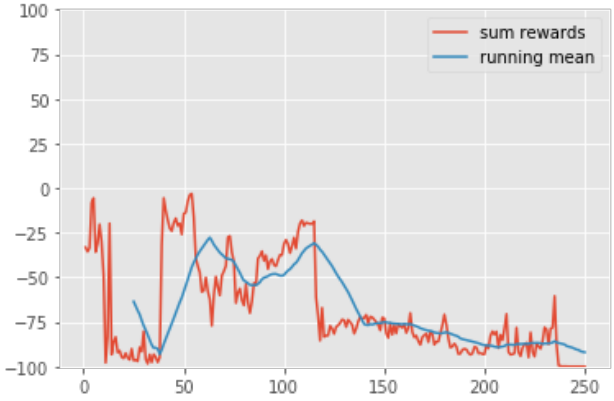
```
Copter
```




```
CopterBig
```




```
CopterMax
```



Dropout

Dropout is a long used regularization technique in DNNs [7]. It is especially common in supervised learning, but also has been used occassionally in reinforcement learning [14] .

Small levels of dropout such as 0.1 seem to neither help nor hurt learning, though performance decays after 0.3, and dropout larger than 0.5 seemed to make learning much more difficult. Dropout of 0.1 and 0.2 did seem to stabilize the learning, and make the network less vulnerable to catastophic forgetting once it had succeeded at its task.

Network size when using dropout should also be considered, though was not adjusted for here.

"If n is the number of hidden units in any layer and p is the probability of retaining a unit,then instead of n hidden units, only pn units will be present after dropout, in expectation. Moreover, this set of pn units will be different each time and the units are not allowed to build co-adaptations freely. Therefore, if an n-sized layer is optimal for a standard neuralnet on any given task, a good dropout net should have at least n/p units." [6]

The following settings were used:

---------- ********** ********** Initializing DDPG Agent Environment: >> env.action_space.shape (1,) env.action_space.low [-1.] env.action_space.high [1.] *** init actor *** self.action_range: [2.] *** init actor *** self.action_range: [2.] ********** *** DDPG Agent Paramter *** - network architecture chosen: QuadCopterMax [ACTOR MODEL SUMMARY]
_____ Layer (type) Output Shape Param # ========== states (InputLayer) (None, 2) 0 _____ dense_1 (Dense) (None, 192) 576 _____ dropout_1 (Dropout) (None, 192) 0 _____ dense_2 (Dense) (None, 384) 74112 _____ dropout_2 (Dropout) (None, 384) 0 _____ dense_3 (Dense) (None, 384) 147840 _____ dropout_3 (Dropout) (None, 384) 0 _____ dense_4 (Dense) (None, 192) 73920 _____ dropout_4 (Dropout) (None, 192) 0 _____ raw_actions (Dense) (None, 1) 193 _____ actions (Lambda) (None, 1) 0 ========== Total params: 296,641 Trainable params: 296,641 Non-trainable params: 0 _____ [CRITIC MODEL SUMMARY]
_____ Layer (type) Output Shape Param # Connected to ==========
states (InputLayer) (None, 2) 0
_____
actions (InputLayer) (None, 1) 0
_____
dense_9 (Dense) (None, 192) 576 states[0][0]
_____
dense_11 (Dense) (None, 192) 384 actions[0][0]
_____
dropout_9 (Dropout) (None, 192) 0 dense_9[0][0]
_____
dropout_11 (Dropout) (None, 192) 0 dense_11[0][0]
_____
dense_10 (Dense) (None, 384) 74112 dropout_9[0][0]
_____

dense_12 (Dense) (None, 384) 74112 dropout_11[0][0]

_____

dropout_10 (Dropout) (None, 384) 0 dense_10[0][0]

_____

dropout_12 (Dropout) (None, 384) 0 dense_12[0][0]

_____

add_1 (Add) (None, 384) 0 dropout_10[0][0] dropout_12[0][0]

_____

dense_13 (Dense) (None, 384) 147840 add_1[0][0]

_____

q_values (Dense) (None, 1) 385 dense_13[0][0]

==========

Total params: 297,409 Trainable params: 297,409 Non-trainable params: 0

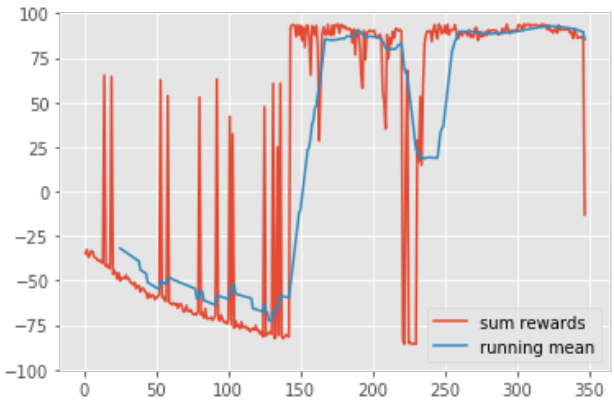
_____ -

action_repeat: 1 - learningRate: 0.0001 - learnFrequency: 1 - buffer_size: 100000 - batch_size: 64 - gamma: 0.99 - useSoftUpdates: True - tau: 0.01 - explore_start: 1.0 - explore_stop: 0.01 - explore_decay_rate: 1e-05

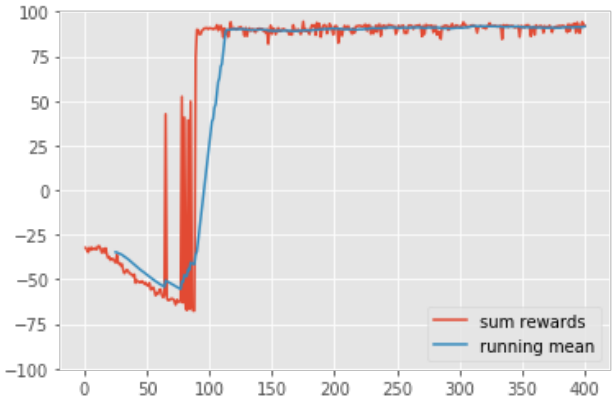
****

In [21]:
```
from visuals import plot_score_from_file

print("Dropout 0.0")
dir = "results/MtClimberContinuous/dropout/0.0/"
plot_score_from_file(dir + "20190622145327MountainCarContinuous-v0_train.txt",
-100, 100, 1)

print("Dropout 0.2 (note QuadcoperBig network instead of Quadcopter Max)")
dir = "results/MtClimberContinuous/dropout/0.2/"
plot_score_from_file(dir + "20190623181645MountainCarContinuous-v0_train.txt",
-100, 100, 1)

print("Dropout 0.3")
dir = "results/MtClimberContinuous/dropout/0.3/"
plot_score_from_file(dir + "20190622145438MountainCarContinuous-v0_train.txt",
-100, 100, 1)

print("Dropout 0.5")
dir = "results/MtClimberContinuous/dropout/0.5/"
plot_score_from_file(dir + "20190622170332MountainCarContinuous-v0_train.txt",
-100, 100, 1)

print("Dropout 0.7")
dir = "results/MtClimberContinuous/dropout/0.7/"
plot_score_from_file(dir + "20190622165506MountainCarContinuous-v0_train.txt",
-100, 100, 1)
```

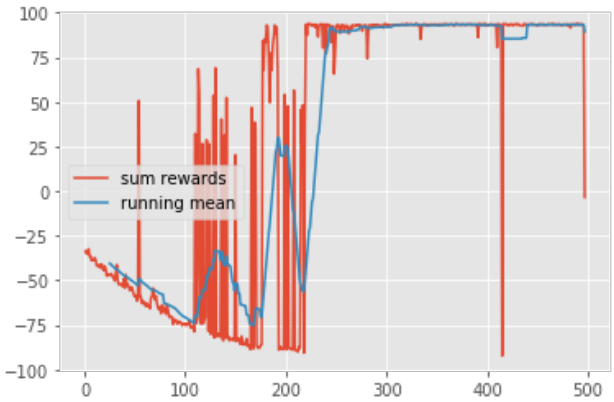
```
Dropout 0.0
```



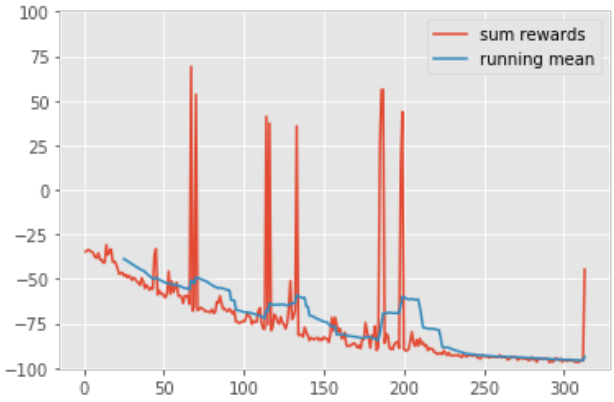

```
Dropout 0.2 (note QuadcoperBig network instead of Quadcopter Max)
```



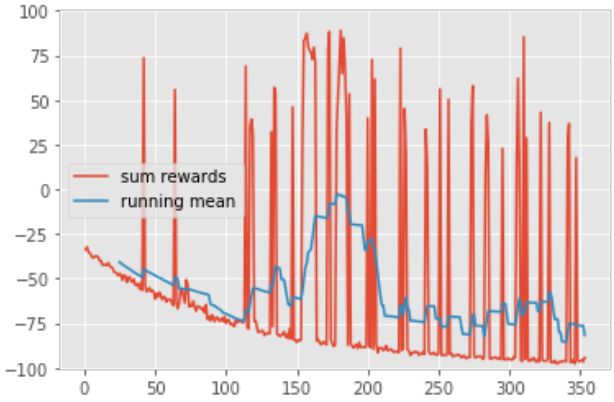

```
Dropout 0.3
```




```
Dropout 0.5
```



Dropout 0.7



Regularizers

Regularizers are another common technique in superised learning.

The application of L2 regularization however did not appear to have any benefit in reinforcement learning. While seeming to yield smoother results, it seemed to make it difficult for the agent to converge to the optimal policy. L2 levels of 0, 0.01, and 0.001 were applied to each layer of the actor and critic networks.

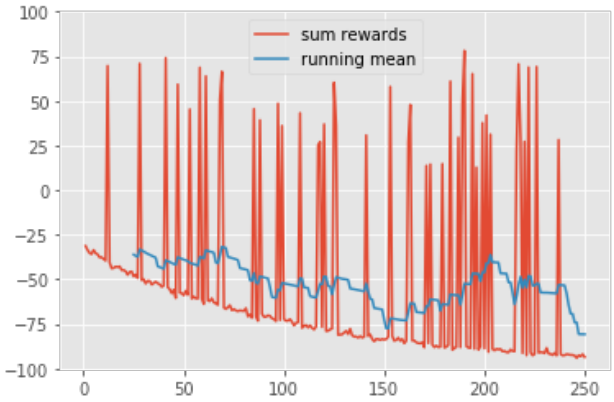
New Experiment, training output file name: 20190623113955MountainCarContinuous-v0_train.txt env.observation_space: (2,) Using TensorFlow backend. ********** ********** Initializing DDPG Agent Environment: >> env.action_space.shape (1,) env.action_space.low [-1.] env.action_space.high [1.] *** init actor *** self.action_range: [2.] *** init actor *** self.action_range: [2.] ********** *** DDPG Agent Paramter *** - network architecture chosen: QuadCopterBig [ACTOR MODEL SUMMARY]
_____ Layer (type) Output Shape Param #
========== states (InputLayer) (None, 2) 0
_____ dense_1 (Dense) (None, 128) 384
_____ dropout_1 (Dropout) (None, 128) 0
_____ dense_2 (Dense) (None, 256) 33024
_____ dropout_2 (Dropout) (None, 256) 0
_____ dense_3 (Dense) (None, 256) 65792
_____ dropout_3 (Dropout) (None, 256) 0
_____ dense_4 (Dense) (None, 128) 32896
_____ dropout_4 (Dropout) (None, 128) 0
_____ raw_actions (Dense) (None, 1) 129

_____ actions (Lambda) (None, 1) 0
=== Total params: 132,225 Trainable params: 132,225 Non-trainable params: 0 _____ [CRITIC MODEL SUMMARY]
_____ Layer (type) Output Shape Param # Connected to
==========
states (InputLayer) (None, 2) 0
_____
actions (InputLayer) (None, 1) 0
_____
dense_9 (Dense) (None, 128) 384 states[0][0]
_____
dense_11 (Dense) (None, 128) 256 actions[0][0]
_____
dropout_9 (Dropout) (None, 128) 0 dense_9[0][0]
_____
dropout_11 (Dropout) (None, 128) 0 dense_11[0][0]
_____
dense_10 (Dense) (None, 256) 33024 dropout_9[0][0]
_____
dense_12 (Dense) (None, 256) 33024 dropout_11[0][0]
_____
dropout_10 (Dropout) (None, 256) 0 dense_10[0][0]
_____
dropout_12 (Dropout) (None, 256) 0 dense_12[0][0]
_____
add_1 (Add) (None, 256) 0 dropout_10[0][0] dropout_12[0][0]
_____
dense_13 (Dense) (None, 256) 65792 add_1[0][0]
_____
q_values (Dense) (None, 1) 257 dense_13[0][0]
==========
Total params: 132,737 Trainable params: 132,737 Non-trainable params: 0
_____ - action_repeat: 1 - learningRate: 0.0001 - learnFrequency: 1 - dropout_rate: 0.0 - buffer_size: 100000 - batch_size: 128 - gamma: 0.99 - useSoftUpdates: True - tau: 0.01 - explore_start: 1.0 - explore_stop: 0.01 - explore_decay_rate: 1e-05
****

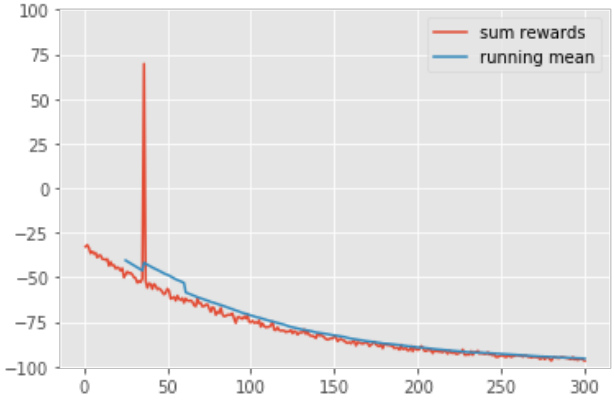
In [22]:
```
from visuals import plot_score_from_file

print("L2 0.0")
dir = "results/MtClimberContinuous/l2norm/0.000/"
plot_score_from_file(dir + "20190623105751MountainCarContinuous-v0_train.txt",
-100, 100, 1)

print("L2 0.001")
dir = "results/MtClimberContinuous/l2norm/0.001/"
plot_score_from_file(dir + "20190623074245MountainCarContinuous-v0_train.txt",
-100, 100, 1)

print("L2 0.010")
dir = "results/MtClimberContinuous/l2norm/0.010/"
plot_score_from_file(dir + "20190623113955MountainCarContinuous-v0_train.txt",
-100, 100, 1)
```

```
L2 0.0
```


```
L2 0.001
```




```
L2 0.010
```



Optimizer

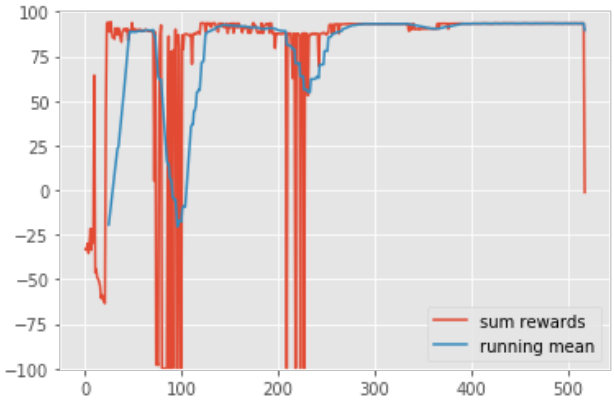
For the DDPG agent the Adam optimiser was used.

This network is widely used, and hence was not experimented with widely ([1], [16], [17]), since there were already too many parameters and architectural decisions to make.

AMS grad option [8] to the Adam optimizer showed faster convergence, but a more unstable solution.

In [23]:
```
print("ams grad")
dir = "results/MtClimberContinuous/amsgrad/"
plot_score_from_file(dir + "20190630112548MountainCarContinuous-v0_train.txt",
-100, 100, 1)
```

```
ams grad
```

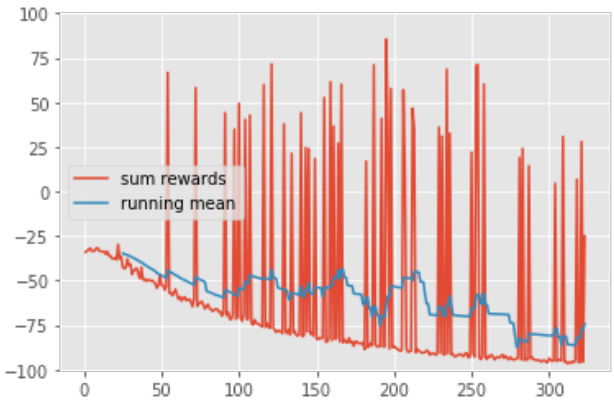


Learning Rate

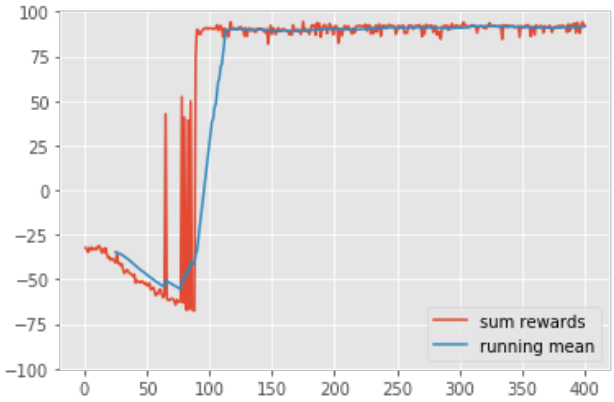
The learning rate is one of the most fundamental and difficult parts to get right in any network. For mountain climber environment, a learning rate of 0.0001 seems to produce reasonable results.

In [24]:
```
from visuals import plot_score_from_file

print("Learning Rate 0.001")
dir = "results/MtClimberContinuous/learningrate/0.001/"
plot_score_from_file(dir + "20190623190455MountainCarContinuous-v0_train.txt",
-100, 100, 1)

print("Learning Rate 0.0001 (from dropout testing)")
dir = "results/MtClimberContinuous/dropout/0.2/"
plot_score_from_file(dir + "20190623181645MountainCarContinuous-v0_train.txt",
-100, 100, 1)

print("Learning Rate 0.00001")
dir = "results/MtClimberContinuous/learningrate/0.00001/"
plot_score_from_file(dir + "20190623190506MountainCarContinuous-v0_train.txt",
-100, 100, 1)
```

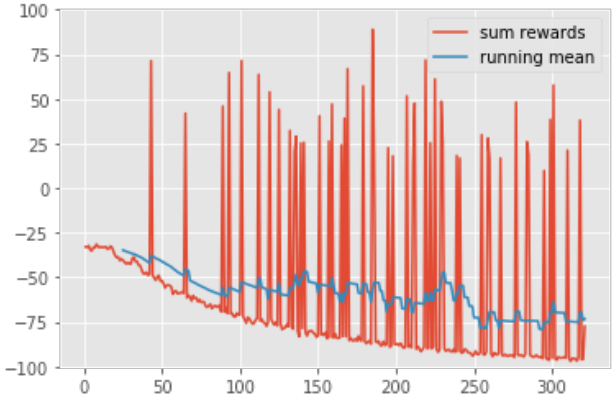
```
Learning Rate 0.001
```




```
Learning Rate 0.0001 (from dropout testing)
```




```
Learning Rate 0.00001
```



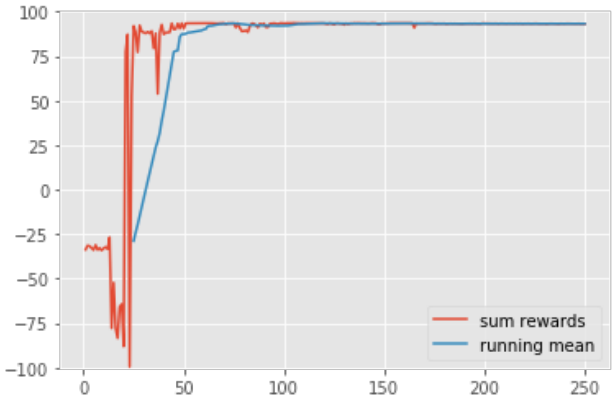
Action Repeat

Action repeat is meant to give the agent some ability to infer velocity. It did not seem to show any real benefit for this environment.

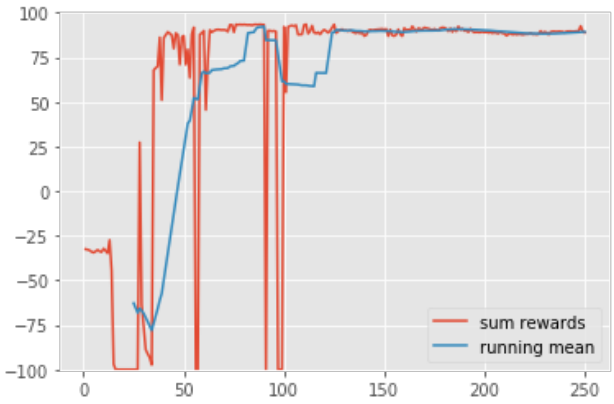
It would probably be more beneficial to just feed the agents previous state as a network input.

In [25]:
```
from visuals import plot_score_from_file

print("Action Repeat 3")
dir = "results/MtClimberContinuous/actionRepeat/3/"
plot_score_from_file(dir + "20190628174616MountainCarContinuous-v0_train.txt",
-100, 100, 1)

print("Action Repeat 5")
dir = "results/MtClimberContinuous/actionRepeat/5/"
plot_score_from_file(dir + "20190628180246MountainCarContinuous-v0_train.txt",
-100, 100, 1)
```

```
Action Repeat 3
```




```
Action Repeat 5
```



Exploration Policy

The following exploration policies were used

- Initial random exploration to fill up part of the replay buffer (used multiply by 100 of buffer size times the action size squared)
- Off after initial random actions to fill replay buffer
- Uncorrelated Noise - Decaying filter per episode
- Correleated Noise - Decaying filter per episode
- OU Noise
- Parameter noise in the network itself

"Parameter noise helps algorithms explore their environments more effectively, leading to higher scores and more elegant behaviors. We think this is because adding noise in a deliberate manner to the parameters of the policy makes an agent's exploration consistent across different timesteps, whereas adding noise to the action space leads to more unpredictable exploration which isn't correlated to anything unique to the agent's parameters." - https://openai.com/blog/better-exploration-with-parameter-noise/ (https://openai.com/blog/better-exploration-with-parameter-noise/)

Using expodentially decaying noise with a pure explore or exploit policy, resulted in more consistant and easier to tune results across agent hyper-parameter settings and environments than OU noise.

Used a decaying exploration policy (keeping a percentage of the old exploration value after each episode finished).

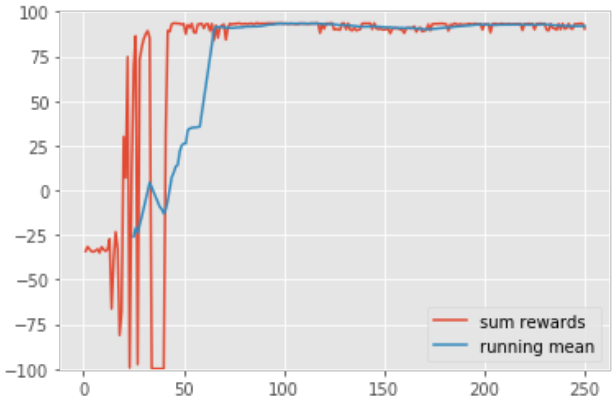
Pure exploitation seems to be important to allow the agent to get the precision on its actions, as correlated to its network weights, to continue the learning process.

Of the exploration results, correlated exploration (weighted randomy and agent agent), appeared to produce more consistant and stable actions than uncorrelated (completely random actions). Paramter noise on the state and action inputs to the actor and critic appeared to damage the learning process and was overall hard to tune.

In [26]:
```
def sample_memory_full_enough(self):
    return self.batch_size * 100 * self.action_size * self.action_size
```

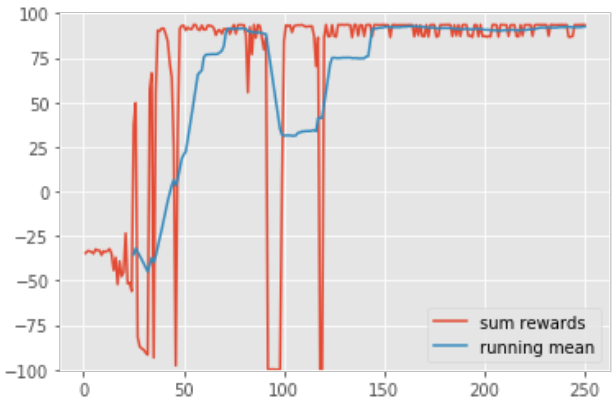
In [27]:
```
from visuals import plot_score_from_file

# no exploration
print("no exploration")
dir = "results/MtClimberContinuous/exploration/none/"
plot_score_from_file(dir + "20190628200134MountainCarContinuous-v0_train.txt",
-100, 100, 1)
```

```
no exploration
```

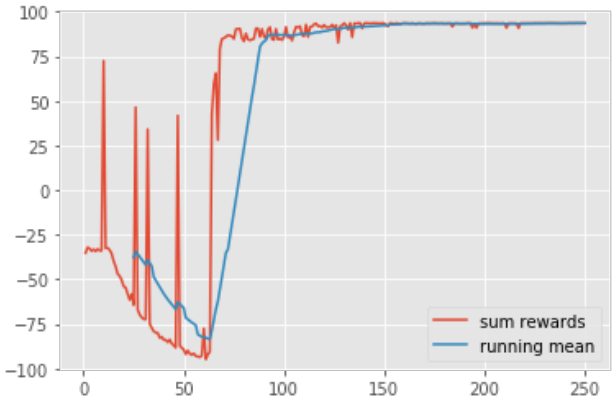


In [28]:
```
# uncorrelated exploration - 0.9 (0.1 decay per episode)
print("0.1 uncorrelated exploration decay")
dir = "results/MtClimberContinuous/exploration/uncorrelated/0.9/"
plot_score_from_file(dir + "20190628200353MountainCarContinuous-v0_train.txt",
-100, 100, 1)

# uncorrelated exploration - 0.95 (0.05 decay per episode)
print("0.05 uncorrelated exploration decay")
dir = "results/MtClimberContinuous/exploration/uncorrelated/0.95/"
plot_score_from_file(dir + "20190628202103MountainCarContinuous-v0_train.txt",
-100, 100, 1)

# uncorrelated exploration - 0.99 (0.01 decay per episode)
print("0.01 uncorrelated exploration decay")
dir = "results/MtClimberContinuous/exploration/uncorrelated/0.99/"
plot_score_from_file(dir + "20190628203409MountainCarContinuous-v0_train.txt",
-100, 100, 1)
```

```
0.1 uncorrelated exploration decay
```



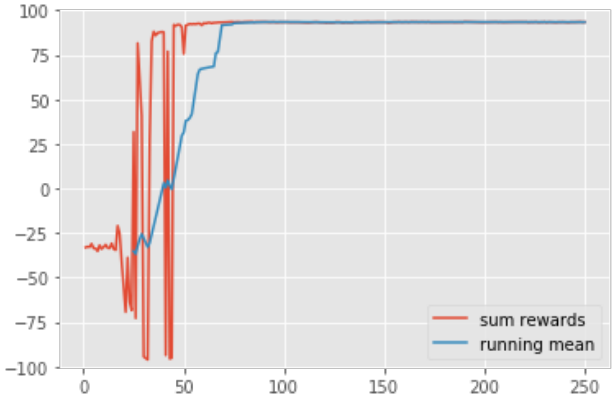

```
0.05 uncorrelated exploration decay
```



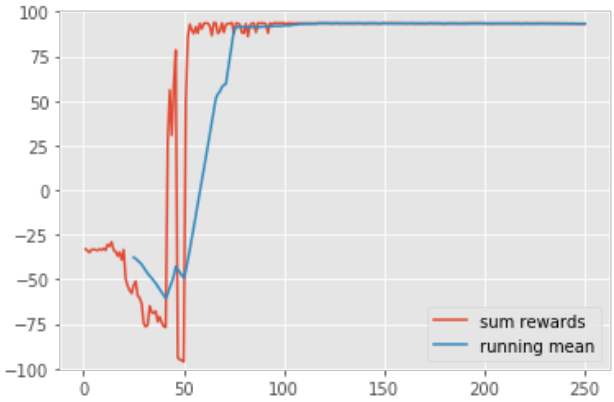

```
0.01 uncorrelated exploration decay
```

In [29]:
```
# correlated exploration - 0.9 (0.1 decay per episode)
print("0.1 correlated exploration decay")
dir = "results/MtClimberContinuous/exploration/correlated/0.9/"
plot_score_from_file(dir + "20190628221842MountainCarContinuous-v0_train.txt",
-100, 100, 1)

# correlated exploration - 0.95 (0.05 decay per episode)
print("0.05 correlated exploration decay")
dir = "results/MtClimberContinuous/exploration/correlated/0.95/"
plot_score_from_file(dir + "20190628221908MountainCarContinuous-v0_train.txt",
-100, 100, 1)

# correlated exploration - 0.99 (0.01 decay per episode)
print("0.01 correlated exploration decay")
dir = "results/MtClimberContinuous/exploration/correlated/0.99/"
plot_score_from_file(dir + "20190628223822MountainCarContinuous-v0_train.txt",
-100, 100, 1)
```

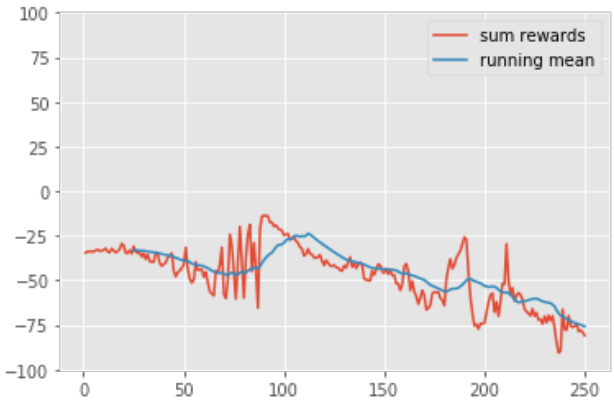
```
0.1 correlated exploration decay
```




```
0.05 correlated exploration decay
```

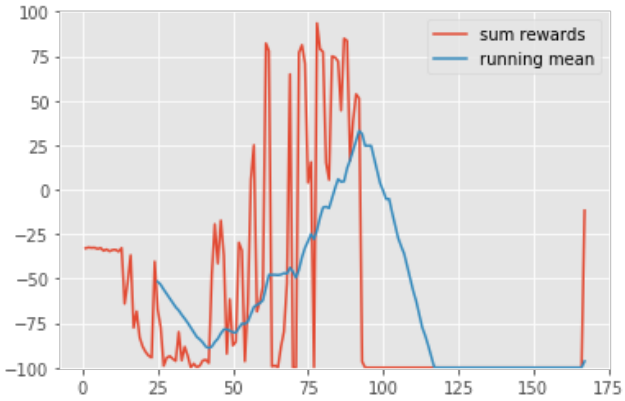



```
0.01 correlated exploration decay
```

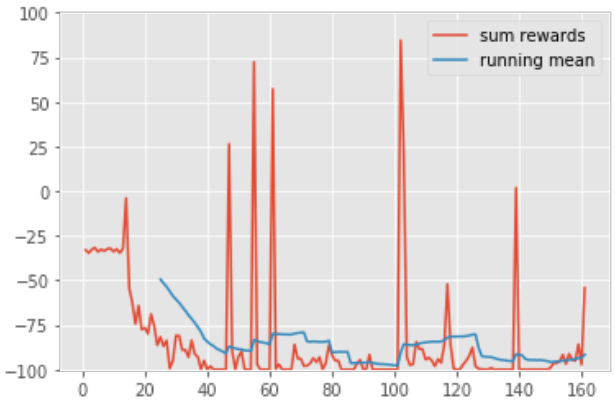


In [30]:
```
# parameter noise on the network - std dev 0.01
print("paramnoise exploration decay - 0.01")
dir = "results/MtClimberContinuous/exploration/paramnoise/0.01/"
plot_score_from_file(dir + "20190628212025MountainCarContinuous-v0_train.txt",
-100, 100, 1)

# parameter noise on the network - std dev 0.1
print("paramnoise exploration decay - 0.1")
dir = "results/MtClimberContinuous/exploration/paramnoise/0.1/"
plot_score_from_file(dir + "20190628211302MountainCarContinuous-v0_train.txt",
-100, 100, 1)
```

```
paramnoise exploration decay - 0.01
```




```
paramnoise exploration decay - 0.1
```



Random Replay Memory

The idea with random replay is to de-couple the actions from the learning, and is widely used in both DQN and DDPG. Instead of learning from the current state, prior state, and action, which tend to be highly correlated from one step to the next, the agent instead learns randomly from memory. This allows the agent to learn multiple times from the same experience in a reasonably de-correlelated way, and makes the agent less likely to get stuck in poor behavior patterns based on its recent state and action.

- Buffer Size: Typically values for this are in the range of 100,000 or 1,000,000 steps [1]. Most initial experiments were conducted with a buffer size of 100,000. Later this was increased to 1,000,000 to add more robustness against complicated enviroments. It seems plausable that an increased buffer size could make the learning process slower, as one is learning for events further in the past rather than capitalizing on the actions of more recently trained network.
- Batch Size: How many samples from the replay buffer to use. Common values are 32, 64, 128, and 256 samples [1]. Generally using a larger batch size seems to speed up the trainnig process.

Learning rate appeared coupled to the batch size. While larger batch sizes seem to make the agent learn faster, it also appeared that if the batch size got too large, the learning rate should be decrease to compensate for the increased learning that happens each episode as a result of the larger batch size. It appears that this is the reason for the poor performance of batch size 1024, when compared to 256.

---------- New Experiment, training output file name: 20190628224957MountainCarContinuous-v0_train.txt env.observation_space: (2,) ********** ********** Initializing DDPG Agent Environment: >> env.action_space.shape (1,) env.action_space.low [-1.] env.action_space.high [1.] *** init actor *** self.action_range: [2.] *** init actor *** self.action_range: [2.] ********** *** DDPG Agent Paramter *** - network architecture chosen: QuadCopterBig [ACTOR MODEL SUMMARY] _____ Layer (type) Output Shape Param # ========== states (InputLayer) (None, 2) 0 _____ dense_1 (Dense) (None, 128) 384 _____ dropout_1 (Dropout) (None, 128) 0 _____ dense_2 (Dense) (None, 256) 33024 _____ dropout_2 (Dropout) (None, 256) 0 _____ dense_3 (Dense) (None, 256) 65792 _____ dropout_3 (Dropout) (None, 256) 0 _____ dense_4 (Dense) (None, 128) 32896 _____ dropout_4 (Dropout) (None, 128) 0 _____ raw_actions (Dense) (None, 1) 129 ========== Total params: 132,225 Trainable params: 132,225 Non-trainable params: 0 _____ [CRITIC MODEL SUMMARY]

_____ Layer (type) Output Shape Param # Connected to

==========

states (InputLayer) (None, 2) 0

_____

actions (InputLayer) (None, 1) 0

_____

dense_9 (Dense) (None, 128) 384 states[0][0]

_____

dense_11 (Dense) (None, 128) 256 actions[0][0]

_____

dropout_9 (Dropout) (None, 128) 0 dense_9[0][0]

_____

dropout_11 (Dropout) (None, 128) 0 dense_11[0][0]

_____

dense_10 (Dense) (None, 256) 33024 dropout_9[0][0]

_____

dense_12 (Dense) (None, 256) 33024 dropout_11[0][0]

_____

dropout_10 (Dropout) (None, 256) 0 dense_10[0][0]

_____

dropout_12 (Dropout) (None, 256) 0 dense_12[0][0]

_____

add_1 (Add) (None, 256) 0 dropout_10[0][0] dropout_12[0][0]

_____

dense_13 (Dense) (None, 256) 65792 add_1[0][0]

_____

q_values (Dense) (None, 1) 257 dense_13[0][0]

==========

Total params: 132,737 Trainable params: 132,737 Non-trainable params: 0

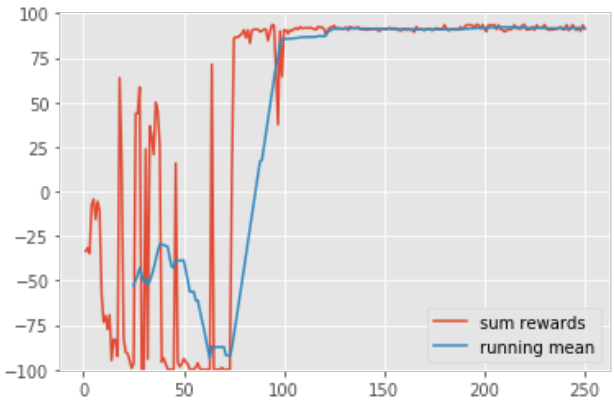
_____ -

action_repeat: 1 - learningRate: 0.0001 - learnFrequency: 1 - dropout_rate: 0.2 - buffer_size: 1000000 - batch_size: 32 - gamma: 0.99 - useSoftUpdates: True - tau: 0.01 - explore_start: 1.0 - explore_stop: 0.0 - explore_decay_rate: 0.0

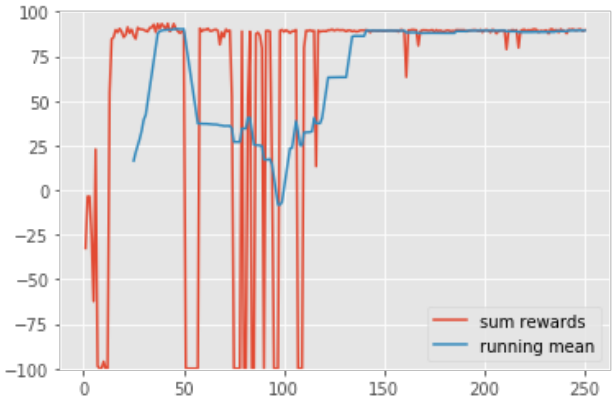
****

In [31]:
```
# Buffer Size 32
print("batch size 32")
dir = "results/MtClimberContinuous/batchsize/32/"
plot_score_from_file(dir + "20190628224957MountainCarContinuous-v0_train.txt",
-100, 100, 1)

# Buffer Size 128
print("batch size 128")
dir = "results/MtClimberContinuous/batchsize/128/"
plot_score_from_file(dir + "20190629091604MountainCarContinuous-v0_train.txt",
-100, 100, 1)

# Buffer Size 256
print("batch size 256")
dir = "results/MtClimberContinuous/batchsize/256/"
plot_score_from_file(dir + "20190629144525MountainCarContinuous-v0_train.txt",
-100, 100, 1)

# Buffer Size 512
print("batch size 512")
dir = "results/MtClimberContinuous/batchsize/512/"
plot_score_from_file(dir + "20190629003527MountainCarContinuous-v0_train.txt",
-100, 100, 1)

# Buffer Size 1024
print("batch size 1024")
dir = "results/MtClimberContinuous/batchsize/1024/"
plot_score_from_file(dir + "20190629144605MountainCarContinuous-v0_train.txt",
-100, 100, 1)
```

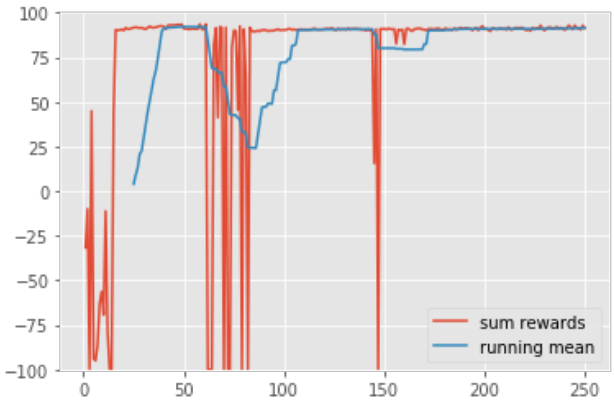
```
batch size 32
```



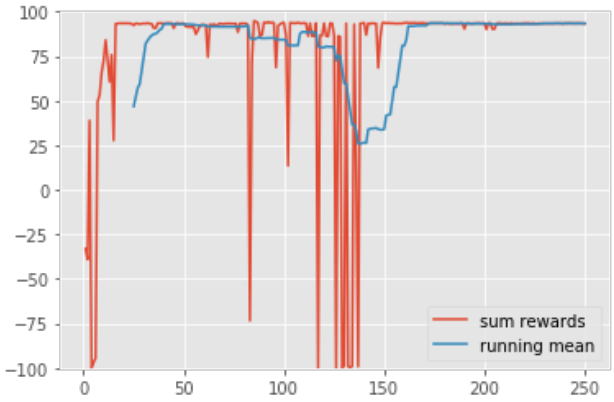

```
batch size 128
```



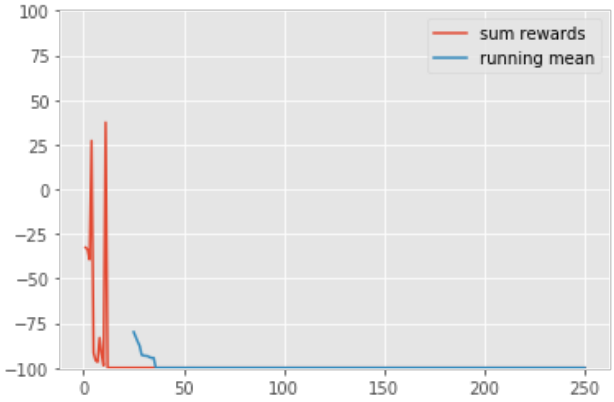

```
batch size 256
```




```
batch size 512
```




```
batch size 1024
```

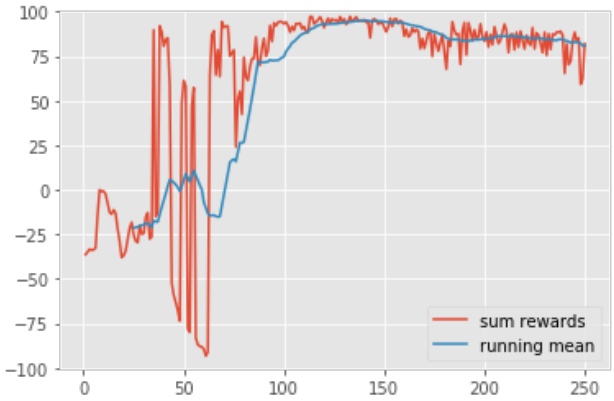


Batch Normalization

A fully batch normalized network (applied at each later), did could solve the environment. However it seems to struggle with having actions that were too sharp, preferring smooth action behaviors.

In [32]:
```
# parameter noise on the network - std dev 0.1
print("paramnoise exploration decay - 0.1")
dir = "results/MtClimberContinuous/batchnorm/all/"
plot_score_from_file(dir + "20190629152238MountainCarContinuous-v0_train.txt",
-100, 100, 1)
```

```
paramnoise exploration decay - 0.1
```



Learning Frequency

The idea here was to use larger batches, but less often, as a trade-off to avoid loading and unloading the GPU all the time. Instead of learning each step, the DDPG agent will use a 10x larger batch size, but learn 10x less frequently. While this does not reduce the number of episodes needed to train the agent, it does speed up the overall time needed to train the agent over all episodes.

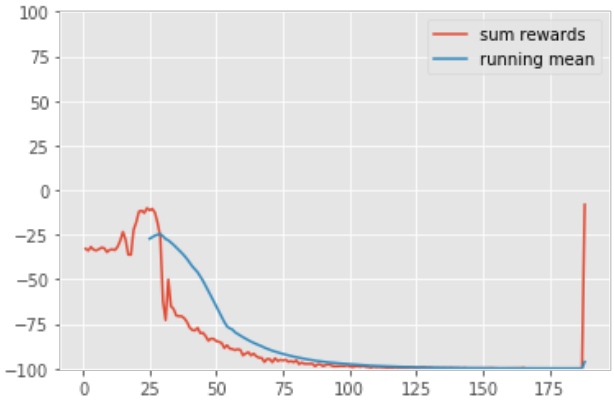
Unfortunately this approach did not seem to really help the learning process, and was hence abandoned.

Soft Update

Turning off soft updates had a destabilizing effect on the agent, as shown below

In [33]:
```
print("no soft update")
dir = "results/MtClimberContinuous/noSoftUpdate/"
plot_score_from_file(dir + "20190629182336MountainCarContinuous-v0_train.txt",
-100, 100, 1)
```

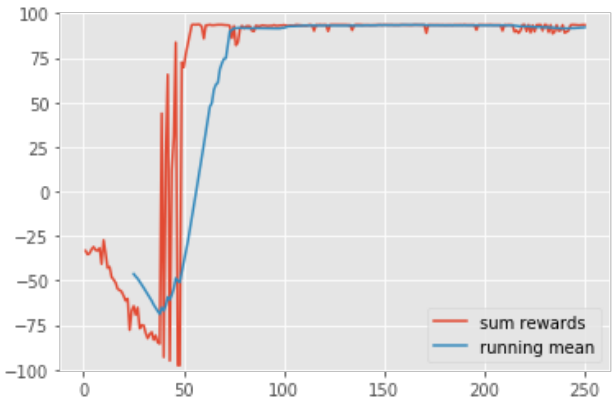
```
no soft update
```



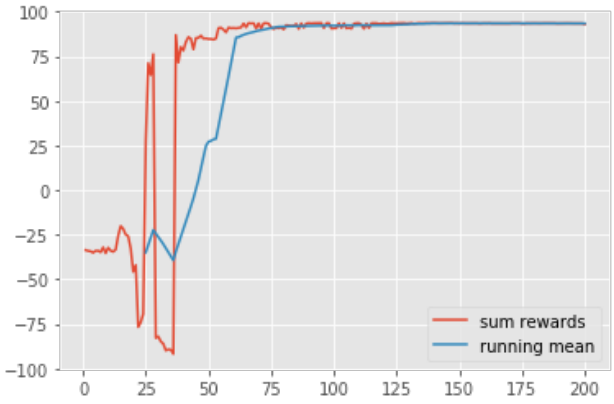
Below are results for different values of the soft update tau parameter.

In [34]:
```
# Tau 0.1
print("tau 0.1")
dir = "results/MtClimberContinuous/tau/0.1/"
plot_score_from_file(dir + "20190630133131MountainCarContinuous-v0_train.txt",
-100, 100, 1)

# Tau 0.01
print("tau 0.01")
dir = "results/MtClimberContinuous/tau/0.01/"
plot_score_from_file(dir + "20190629195726MountainCarContinuous-v0_train.txt",
-100, 100, 1)

# Tau 0.001
print("tau 0.001")
dir = "results/MtClimberContinuous/tau/0.001/"
plot_score_from_file(dir + "20190630120155MountainCarContinuous-v0_train.txt",
-100, 100, 1)
```

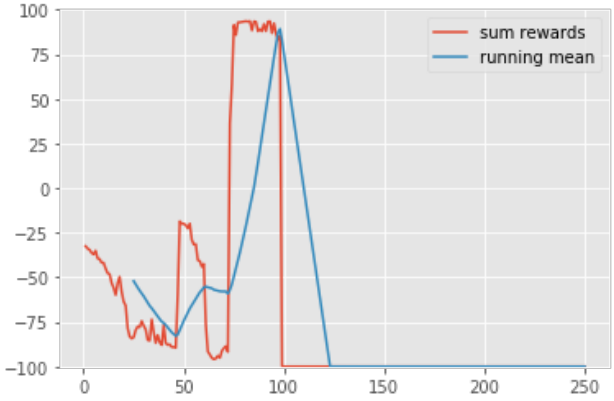
```
tau 0.1
```




```
tau 0.01
```




```
tau 0.001
```



Summary

Effective

- Larger batch size
- Tuning the learning rate
- Soft updates

Unclear

- Dropout (at least didn't seem to hurt, and provided stability and regularization against overfitting, so will be maintained)
- Small amount of exploration, correlated (seemed slightly better) or uncorrelated, with 5-10% decay per episode
- larger amount of neurons per layer. This didn't seem to hurt the ability to learn, though with more parameters learning look longer per episode, and took longer to converge to a solution
- Action repeat. Did not seem to provide any benefit, but didn't seem to hurt either. Can further evaluate on other environments
- Batch normalization. Smoothes out the agents actions, however slows down learning (both in training time per episode, and in number of episodes to find a solution), and makes the final result less stable. Batch normalization appears to make fine control more difficult.

Ineffective

- L2 regularization. This appears to make the agents actions
- ELU activation function instead of RELU. Did not appear to provide any benefit.
- Learning Frequency. Didn't appear to add any value.
- Large amount of exploration (1% decay). Generally this made the agent perform much worse.
- Network input noise as exploration.

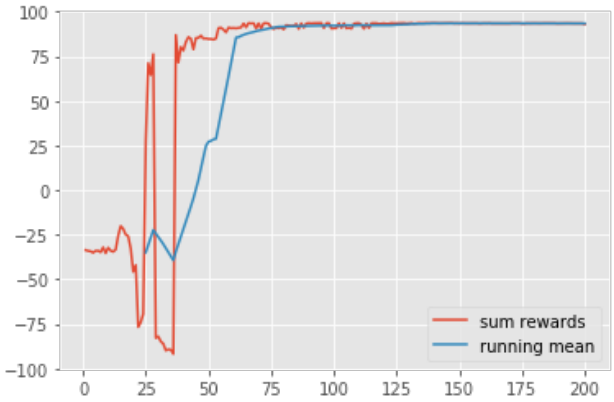
Final Agent Settings

The following agent performed well, learning fast, with performance stable after rearch the desired reward level. The agent solves in around 35 episodes, doing well compared to the leaderboard on OpenAI (https://github.com/openai/gym/wiki/Leaderboard (https://github.com/openai/gym/wiki/Leaderboard)), with even faster solution possible (~20 episodes) if the initial randomy exploration period to fill memory is shorter.

- Batch size: 128
- Memory size: 1,000,000
- Learning Rate: 0.0001
- Soft Updates: ON. Tau soft update rate: 0.01
- Batch size * 25 memory warm up with random actions, then correleated exploration with 5% decay per episode
- Network: CopterBig (will change to max if learning appears to plateau in other environments, and batch norm is still an option)
- Action Repeat: 2
- Dropout: 0.2, all hidden layers
- Discount Factor: 0.99

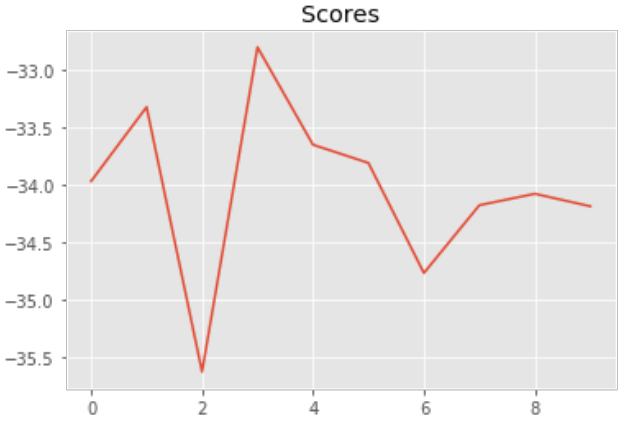
In [35]:
```
print("final")
dir = "results/MtClimberContinuous/final/"
plot_score_from_file(dir + "20190629195726MountainCarContinuous-v0_train.txt",
-100, 100, 1)
```

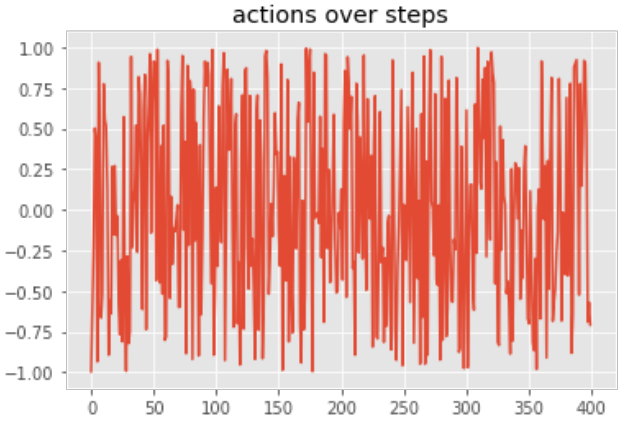
```
final
```

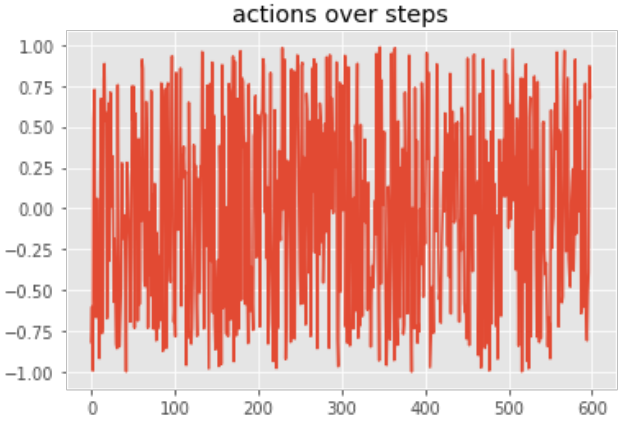


In [36]:
```
import gym
import numpy as np

"""
# Create an environment and set random seed
"""
selectedEnvironment = 6
env = 0
envName = 0

# Toy Text - Discrete state and action space
if selectedEnvironment == 0:
    envName = 'Taxi-v2'

# Classic Control - Continuous State and Discrete Action Spaces
elif selectedEnvironment == 1:
    envName = 'MountainCar-v0' # needs Discretized or better
elif selectedEnvironment == 2:
    envName = 'Acrobot-v1'     # needs Discretized, Tile Encoding or better
elif selectedEnvironment == 3:
    envName = 'CartPole-v1'    # needs Deep Q Learning to do well?

# Box 2D - Continuous State, Discrete Actions
elif selectedEnvironment == 4:
    envName = 'LunarLander-v2' # discrete actions, continuous state

# Classic Control - Continuous State and Action Spaces
elif selectedEnvironment == 5:
    envName = 'Pendulum-v0' # continuous only
elif selectedEnvironment == 6:
    envName = 'MountainCarContinuous-v0' # continuous only

# Box 2D - Continuous State and Action Spaces
elif selectedEnvironment == 7:
    envName = 'LunarLanderContinuous-v2' # continuous only
elif selectedEnvironment == 8:
    envName = 'BipedalWalker-v2'  # continuous only

# Box 2D - Image State and Continuous Action Spaces
elif selectedEnvironment == 9:
    envName = 'CarRacing-v0'      #  image input, actions [steer, gas, brake]

# Initialize the environment
env = gym.make(envName)
env.reset()

# Set output file paths based on environment
from visuals import examine_environment, examine_environment_MountainCar_discre
tized, examine_environment_Acrobat_tiled
#examine_environment(env)

from datetime import datetime
FORMAT = '%Y%m%d%H%M%S'
file_output_train = envName + '_train.txt'       # file name for saved results
file_output_test = envName + '_test.txt'       # file name for saved results
file_output_train = datetime.now().strftime(FORMAT) + file_output_train

print('------------------------------------------------------------')
print('New Experiment, training output file name: ', file_output_train)

"""
# Create Agent
"""
```

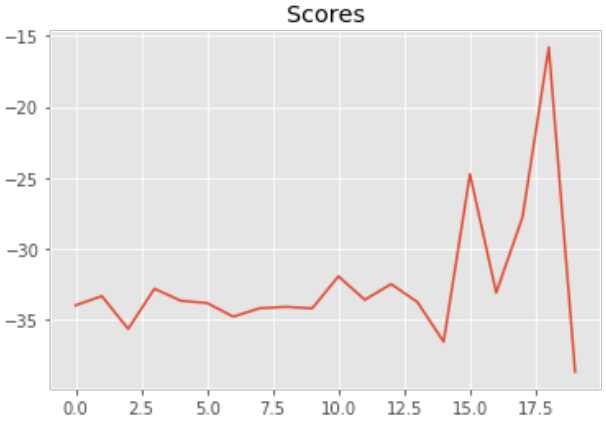
```
------------------------------------------------------------
New Experiment, training output file name:  20190702232654MountainCarContinuous
-v0_train.txt
env.observation_space:  (2,)
************************************
************************************
Initializing DDPG Agent
        Environment:  <TimeLimit<Continuous_MountainCarEnv<MountainCarContinuou
s-v0>>>
env.action_space.shape (1,)
env.action_space.low [-1.]
env.action_space.high [1.]
WARNING:tensorflow:From /home/tamanous/anaconda3/envs/capstone/lib/python3.7/si
te-packages/tensorflow/python/framework/op_def_library.py:263: colocate_with (f
rom tensorflow.python.framework.ops) is deprecated and will be removed in a fut
ure version.
Instructions for updating:
Colocations handled automatically by placer.
WARNING:tensorflow:From /home/tamanous/anaconda3/envs/capstone/lib/python3.7/si
te-packages/keras/backend/tensorflow_backend.py:3445: calling dropout (from ten
sorflow.python.ops.nn_ops) with keep_prob is deprecated and will be removed in
a future version.
Instructions for updating:
Please use `rate` instead of `keep_prob`. Rate should be set to `rate = 1 - kee
p_prob`.
*** init actor ***
self.action_range:  [2.]
*** init actor ***
self.action_range:  [2.]
************************************
*** DDPG Agent Paramter ***
- network architecture chosen:  QuadCopterBig
[ ACTOR MODEL SUMMARY ]
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
states (InputLayer)          (None, 2)                 0
_________________________________________________________________
dense_1 (Dense)              (None, 128)               384
_________________________________________________________________
dropout_1 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_2 (Dense)              (None, 256)               33024
_________________________________________________________________
dropout_2 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_3 (Dense)              (None, 256)               65792
_________________________________________________________________
dropout_3 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_4 (Dense)              (None, 128)               32896
_________________________________________________________________
dropout_4 (Dropout)          (None, 128)               0
_________________________________________________________________
raw_actions (Dense)          (None, 1)                 129
=================================================================
Total params: 132,225
Trainable params: 132,225
Non-trainable params: 0
_________________________________________________________________
[ CRITIC MODEL SUMMARY ]
_____________________________________________________________________________
____________________
```

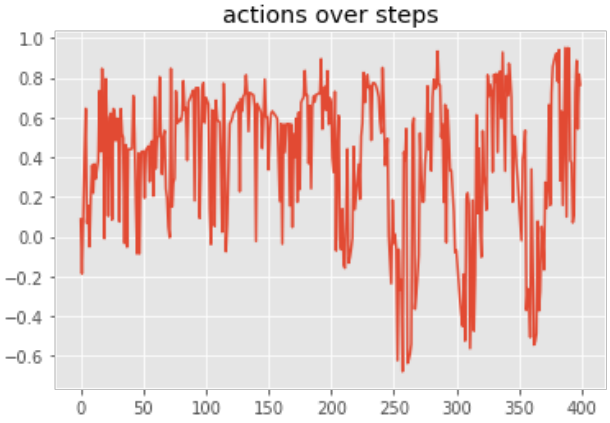

Scores
−33.0
−33.5
−34.0
−34.5
−35.0
−35.5
0
2
4
6
8

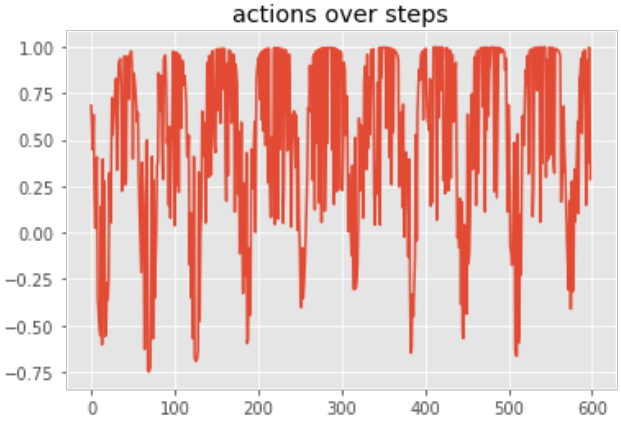

actions over steps
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
50
100
150
200
250
300
350
400


actions over steps
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
100
200
300
400
500
600

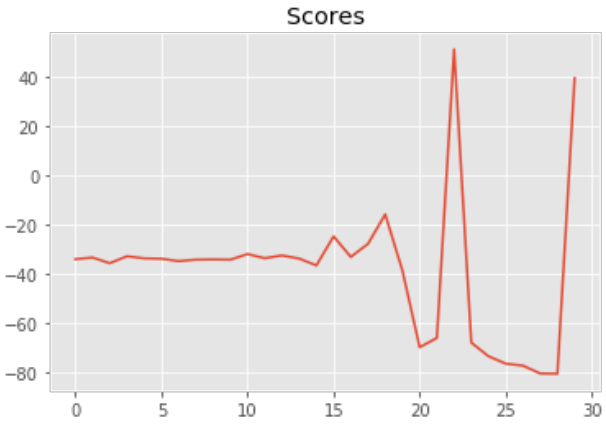
```
Episode =   10 (duration of 999 steps); Reward = -34.188 (best = -32.801, in ep
i 4)
        Episode training time:  0.8737530708312988
        resetting episode... next explore_p:  1.0
Garbage collector: collected 8016 objects.
        step:  0 , action: [-0.904]
        step:  100 , action: [-0.286]
        step:  200 , action: [-0.571]
        step:  300 , action: [-0.829]
        step:  400 , action: [0.637]
        step:  500 , action: [0.486]
        step:  600 , action: [-0.374]
        step:  700 , action: [-0.263]
        step:  800 , action: [0.747]
        step:  900 , action: [0.111]
Episode =   11 (duration of 999 steps); Reward = -31.927 (best = -31.927, in ep
i 11)
        Episode training time:  0.26170992851257324
        resetting episode... next explore_p:  1.0
Garbage collector: collected 0 objects.
        step:  0 , action: [0.24]
        step:  100 , action: [-0.826]
        step:  200 , action: [-0.965]
        step:  300 , action: [-0.789]
        step:  400 , action: [0.459]
        step:  500 , action: [-0.925]
        step:  600 , action: [-0.065]
        step:  700 , action: [0.099]
        step:  800 , action: [0.511]
        step:  900 , action: [0.695]
Episode =   12 (duration of 999 steps); Reward = -33.586 (best = -31.927, in ep
i 11)
        Episode training time:  0.26102399826049805
        resetting episode... next explore_p:  1.0
Garbage collector: collected 0 objects.
        step:  0 , action: [-0.117]
        step:  100 , action: [-0.832]
        step:  200 , action: [0.778]
        step:  300 , action: [0.961]
        step:  400 , action: [0.317]
        step:  500 , action: [-0.761]
        step:  600 , action: [0.191]
        step:  700 , action: [-0.623]
        step:  800 , action: [-0.584]
WARNING:tensorflow:From /home/tamanous/anaconda3/envs/capstone/lib/python3.7/si
te-packages/tensorflow/python/ops/math_ops.py:3066: to_int32 (from tensorflow.p
ython.ops.math_ops) is deprecated and will be removed in a future version.
Instructions for updating:
Use tf.cast instead.
        step:  900 , action: [0.279]
Episode =   13 (duration of 999 steps); Reward = -32.476 (best = -31.927, in ep
i 11)
        Episode training time:  7.167429685592651
        resetting episode... next explore_p:  0.9
Garbage collector: collected 0 objects.
        step:  0 , action: [0.494]
        step:  100 , action: [-0.641]
        step:  200 , action: [0.259]
        step:  300 , action: [-0.815]
        step:  400 , action: [-0.1]
        step:  500 , action: [-0.677]
        step:  600 , action: [-0.16]
        step:  700 , action: [0.71]
```


Scores
−15
−20
−25
−30
−35
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5

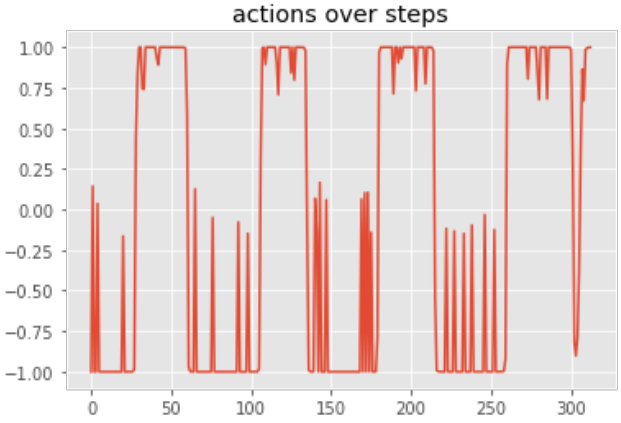

actions over steps
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
0
50
100
150
200
250
300
350
400


actions over steps
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
0
100
200
300
400
500
600

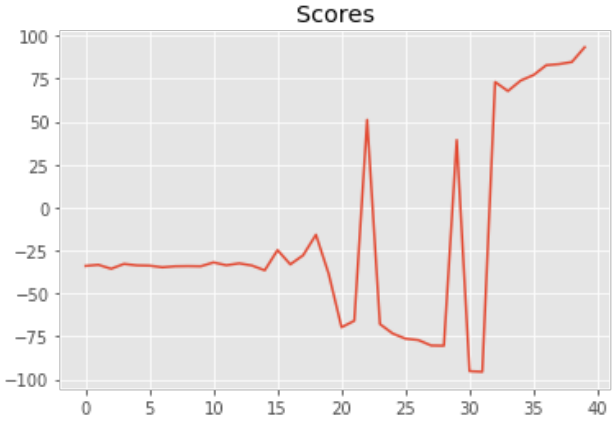
```
Episode =   20 (duration of 999 steps); Reward = -38.647 (best = -15.810, in ep
i 19)
        Episode training time:  27.989524602890015
        resetting episode... next explore_p:  0.43046721000000016
Garbage collector: collected 8096 objects.
        step:  0 , action: [0.998]
        step:  100 , action: [0.312]
        step:  200 , action: [0.999]
        step:  300 , action: [0.799]
        step:  400 , action: [0.979]
        step:  500 , action: [1.]
        step:  600 , action: [1.]
        step:  700 , action: [0.993]
        step:  800 , action: [0.747]
        step:  900 , action: [0.342]
Episode =   21 (duration of 999 steps); Reward = -69.705 (best = -15.810, in ep
i 19)
        Episode training time:  28.524041891098022
        resetting episode... next explore_p:  0.38742048900000015
Garbage collector: collected 0 objects.
        step:  0 , action: [0.743]
        step:  100 , action: [0.811]
        step:  200 , action: [0.868]
        step:  300 , action: [0.995]
        step:  400 , action: [0.548]
        step:  500 , action: [0.993]
        step:  600 , action: [0.986]
        step:  700 , action: [0.04]
        step:  800 , action: [0.999]
        step:  900 , action: [0.46]
Episode =   22 (duration of 999 steps); Reward = -65.923 (best = -15.810, in ep
i 19)
        Episode training time:  28.28426718711853
        resetting episode... next explore_p:  0.34867844010000015
Garbage collector: collected 0 objects.
        step:  0 , action: [0.969]
        step:  100 , action: [0.994]
        step:  200 , action: [0.999]
        step:  300 , action: [-0.997]
        step:  400 , action: [0.995]
        step:  500 , action: [-0.058]
        step:  600 , action: [0.342]
        step:  700 , action: [0.999]
Episode =   23 (duration of 729 steps); Reward =  51.107 (best =  51.107, in ep
i 23)
        Episode training time:  20.462514638900757
        resetting episode... next explore_p:  0.31381059609000017
Garbage collector: collected 0 objects.
        step:  0 , action: [0.993]
        step:  100 , action: [0.444]
        step:  200 , action: [0.94]
        step:  300 , action: [0.89]
        step:  400 , action: [-0.993]
        step:  500 , action: [-0.999]
        step:  600 , action: [1.]
        step:  700 , action: [-1.]
        step:  800 , action: [-1.]
        step:  900 , action: [-0.19]
Episode =   24 (duration of 999 steps); Reward = -67.821 (best =  51.107, in ep
i 23)
        Episode training time:  25.74188208580017
        resetting episode... next explore_p:  0.28242953648100017
Garbage collector: collected 0 objects.
```

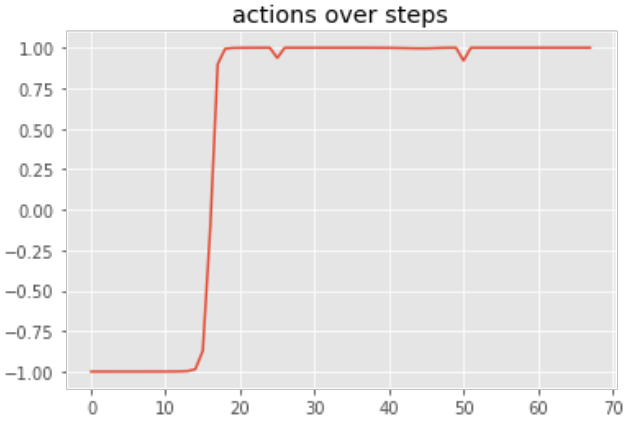

Scores
40
20
0
−20
−40
−60
−80
0
5
10
15
20
25
30


actions over steps
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
50
100
150
200
250
300
350
400

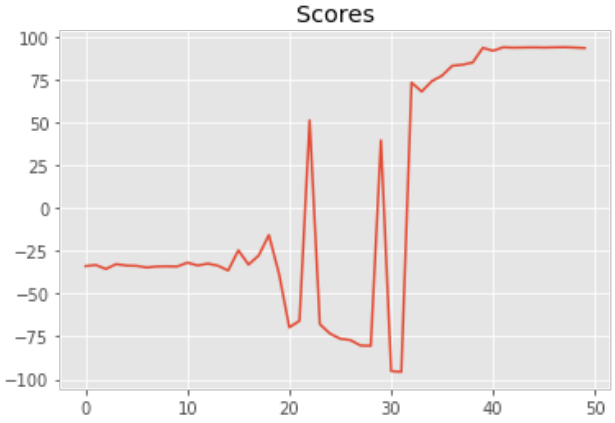

actions over steps
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
50
100
150
200
250
300

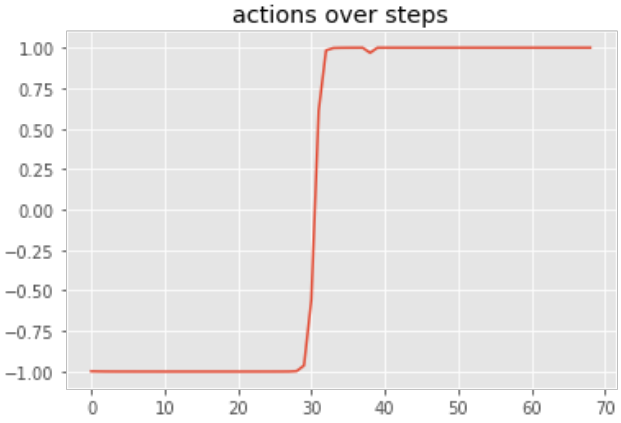
```
Episode =   30 (duration of 713 steps); Reward =  39.367 (best =  51.107, in ep
i 23)
        Episode training time:  20.833544969558716
        resetting episode... next explore_p:  0.1500946352969992
Garbage collector: collected 8211 objects.
        step:  0 , action: [1.]
        step:  100 , action: [1.]
        step:  200 , action: [0.999]
        step:  300 , action: [1.]
        step:  400 , action: [1.]
        step:  500 , action: [1.]
        step:  600 , action: [0.999]
        step:  700 , action: [0.999]
        step:  800 , action: [1.]
        step:  900 , action: [0.807]
Episode =   31 (duration of 999 steps); Reward = -95.245 (best =  51.107, in ep
i 23)
        Episode training time:  28.290242671966553
        resetting episode... next explore_p:  0.13508517176729928
Garbage collector: collected 0 objects.
        step:  0 , action: [1.]
        step:  100 , action: [0.799]
        step:  200 , action: [1.]
        step:  300 , action: [0.999]
        step:  400 , action: [0.985]
        step:  500 , action: [1.]
        step:  600 , action: [0.999]
        step:  700 , action: [0.998]
        step:  800 , action: [1.]
        step:  900 , action: [1.]
Episode =   32 (duration of 999 steps); Reward = -95.598 (best =  51.107, in ep
i 23)
        Episode training time:  28.04551124572754
        resetting episode... next explore_p:  0.12157665459056936
Garbage collector: collected 0 objects.
        step:  0 , action: [0.943]
        step:  100 , action: [0.999]
        step:  200 , action: [-0.99]
        step:  300 , action: [1.]
Episode =   33 (duration of 303 steps); Reward =  73.137 (best =  73.137, in ep
i 33)
        Episode training time:  8.64153003692627
        resetting episode... next explore_p:  0.10941898913151243
Garbage collector: collected 0 objects.
        step:  0 , action: [0.696]
        step:  100 , action: [-1.]
        step:  200 , action: [-1.]
        step:  300 , action: [-1.]
Episode =   34 (duration of 385 steps); Reward =  67.865 (best =  73.137, in ep
i 33)
        Episode training time:  11.001172304153442
        resetting episode... next explore_p:  0.0984770902183612
Garbage collector: collected 0 objects.
        step:  0 , action: [0.232]
        step:  100 , action: [-1.]
        step:  200 , action: [0.912]
        step:  300 , action: [1.]
Episode =   35 (duration of 302 steps); Reward =  73.968 (best =  73.968, in ep
i 35)
        Episode training time:  8.556604146957397
        resetting episode... next explore_p:  0.08862938119652508
Garbage collector: collected 0 objects.
        step:  0 , action: [-0.823]
```


Scores
100
75
50
25
0
−25
−50
−75
−100
0
5
10
15
20
25
30
35
40

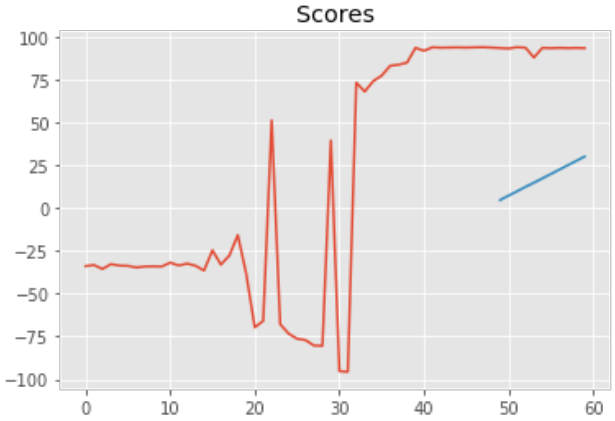

actions over steps
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
10
20
30
40
50
60
70

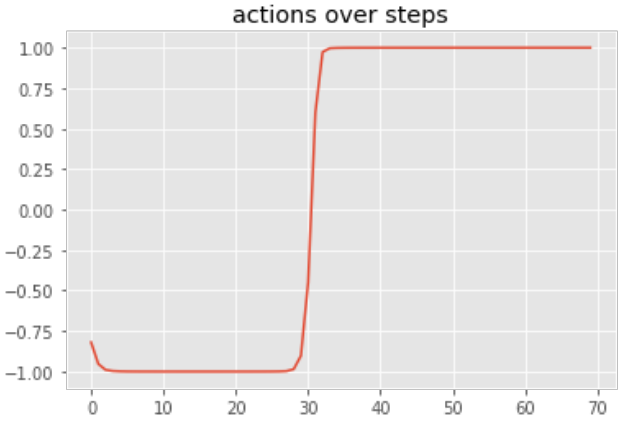
```
Episode =   40 (duration of 68 steps); Reward =  93.379 (best =  93.379, in epi
40)
        Episode training time:  2.374979019165039
        resetting episode... next explore_p:  0.0523347633027361
Garbage collector: collected 5704 objects.
        step:  0 , action: [0.984]
Episode =   41 (duration of 88 steps); Reward =  91.658 (best =  93.379, in epi
40)
        Episode training time:  2.588493585586548
        resetting episode... next explore_p:  0.04710128697246249
Garbage collector: collected 0 objects.
        step:  0 , action: [-1.]
Episode =   42 (duration of 68 steps); Reward =  93.695 (best =  93.695, in epi
42)
        Episode training time:  2.027339220046997
        resetting episode... next explore_p:  0.042391158275216244
Garbage collector: collected 0 objects.
        step:  0 , action: [-1.]
Episode =   43 (duration of 69 steps); Reward =  93.466 (best =  93.695, in epi
42)
        Episode training time:  2.051157236099243
        resetting episode... next explore_p:  0.03815204244769462
Garbage collector: collected 0 objects.
        step:  0 , action: [-1.]
Episode =   44 (duration of 66 steps); Reward =  93.560 (best =  93.695, in epi
42)
        Episode training time:  1.9166631698608398
        resetting episode... next explore_p:  0.03433683820292516
Garbage collector: collected 0 objects.
        step:  0 , action: [-1.]
Episode =   45 (duration of 67 steps); Reward =  93.630 (best =  93.695, in epi
42)
        Episode training time:  1.9816014766693115
        resetting episode... next explore_p:  0.030903154382632643
Garbage collector: collected 0 objects.
        step:  0 , action: [-0.999]
Episode =   46 (duration of 67 steps); Reward =  93.516 (best =  93.695, in epi
42)
        Episode training time:  1.9861867427825928
        resetting episode... next explore_p:  0.02781283894436938
Garbage collector: collected 0 objects.
        step:  0 , action: [-1.]
Episode =   47 (duration of 67 steps); Reward =  93.655 (best =  93.695, in epi
42)
        Episode training time:  1.9994490146636963
        resetting episode... next explore_p:  0.025031555049932444
Garbage collector: collected 0 objects.
        step:  0 , action: [0.224]
Episode =   48 (duration of 68 steps); Reward =  93.741 (best =  93.741, in epi
48)
        Episode training time:  1.9689040184020996
        resetting episode... next explore_p:  0.0225283995449392
Garbage collector: collected 0 objects.
        step:  0 , action: [-1.]
Episode =   49 (duration of 67 steps); Reward =  93.530 (best =  93.741, in epi
48)
        Episode training time:  1.9525456428527832
        resetting episode... next explore_p:  0.020275559590445278
Garbage collector: collected 0 objects.
        step:  0 , action: [-0.998]
```


Scores
100
75
50
25
0
−25
−50
−75
−100
0
10
20
30
40
50

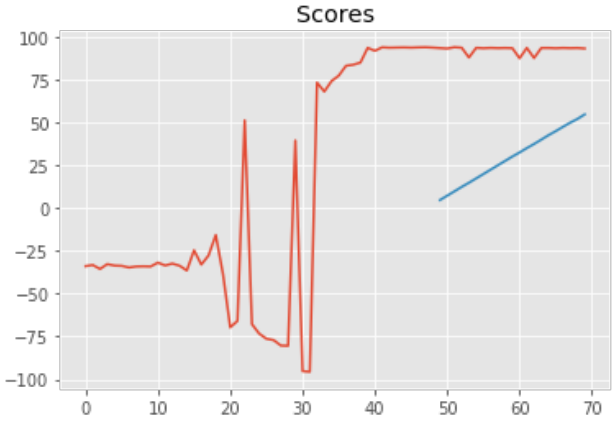

actions over steps
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
10
20
30
40
50
60
70

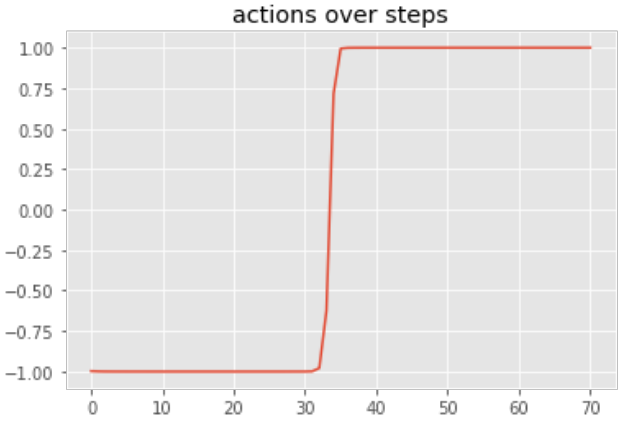
```
Episode =   50 (duration of 69 steps); Reward =  93.250 (best =  93.741, in epi
48)
        Episode training time:  2.415736675262451
        resetting episode... next explore_p:  0.01824800363140075
Garbage collector: collected 5514 objects.
        step:  0 , action: [-0.991]
Episode =   51 (duration of 72 steps); Reward =  93.001 (best =  93.741, in epi
48)
        Episode training time:  2.1458795070648193
        resetting episode... next explore_p:  0.016423203268260675
Garbage collector: collected 0 objects.
        step:  0 , action: [-0.316]
Episode =   52 (duration of 67 steps); Reward =  93.757 (best =  93.757, in epi
52)
        Episode training time:  1.9750664234161377
        resetting episode... next explore_p:  0.014780882941434608
Garbage collector: collected 0 objects.
        step:  0 , action: [-0.999]
Episode =   53 (duration of 69 steps); Reward =  93.461 (best =  93.757, in epi
52)
        Episode training time:  1.997215986251831
        resetting episode... next explore_p:  0.013302794647291147
Garbage collector: collected 0 objects.
        step:  0 , action: [0.914]
        step:  100 , action: [1.]
Episode =   54 (duration of 126 steps); Reward =  87.745 (best =  93.757, in ep
i 52)
        Episode training time:  3.64804744720459
        resetting episode... next explore_p:  0.011972515182562033
Garbage collector: collected 0 objects.
        step:  0 , action: [-1.]
Episode =   55 (duration of 68 steps); Reward =  93.396 (best =  93.757, in epi
52)
        Episode training time:  1.9690499305725098
        resetting episode... next explore_p:  0.01077526366430583
Garbage collector: collected 0 objects.
        step:  0 , action: [-1.]
Episode =   56 (duration of 70 steps); Reward =  93.172 (best =  93.757, in epi
52)
        Episode training time:  2.0765268802642822
        resetting episode... next explore_p:  0.009697737297875247
Garbage collector: collected 0 objects.
        step:  0 , action: [-0.994]
Episode =   57 (duration of 69 steps); Reward =  93.370 (best =  93.757, in epi
52)
        Episode training time:  2.0286715030670166
        resetting episode... next explore_p:  0.008727963568087723
Garbage collector: collected 0 objects.
        step:  0 , action: [-1.]
Episode =   58 (duration of 70 steps); Reward =  93.236 (best =  93.757, in epi
52)
        Episode training time:  2.0677542686462402
        resetting episode... next explore_p:  0.00785516721127895
Garbage collector: collected 0 objects.
        step:  0 , action: [-0.989]
Episode =   59 (duration of 68 steps); Reward =  93.320 (best =  93.757, in epi
52)
        Episode training time:  1.980968713760376
        resetting episode... next explore_p:  0.007069650490151055
Garbage collector: collected 0 objects.
        step:  0 , action: [-0.82]
```


Scores
100
75
50
25
0
−25
−50
−75
−100
0
10
20
30
40
50
60

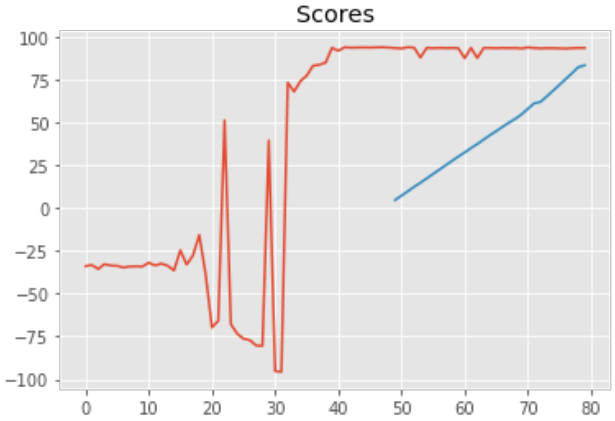

actions over steps
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
10
20
30
40
50
60
70

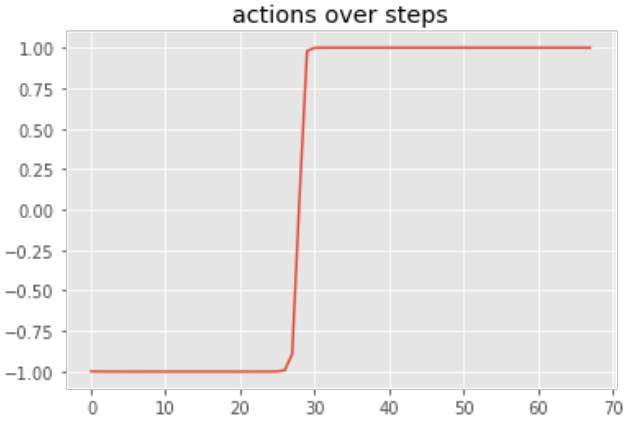
```
Episode =   60 (duration of 70 steps); Reward =  93.217 (best =  93.757, in epi
52)
        Episode training time:  2.390305995941162
        resetting episode... next explore_p:  0.00636268544113595
Garbage collector: collected 5564 objects.
        step:  0 , action: [0.978]
        step:  100 , action: [1.]
Episode =   61 (duration of 132 steps); Reward =  87.304 (best =  93.757, in ep
i 52)
        Episode training time:  3.858700752258301
        resetting episode... next explore_p:  0.005726416897022355
Garbage collector: collected 0 objects.
        step:  0 , action: [-1.]
Episode =   62 (duration of 68 steps); Reward =  93.363 (best =  93.757, in epi
52)
        Episode training time:  2.0011556148529053
        resetting episode... next explore_p:  0.00515377520732012
Garbage collector: collected 0 objects.
        step:  0 , action: [0.966]
        step:  100 , action: [1.]
Episode =   63 (duration of 130 steps); Reward =  87.430 (best =  93.757, in ep
i 52)
        Episode training time:  3.8628182411193848
        resetting episode... next explore_p:  0.004638397686588107
Garbage collector: collected 0 objects.
        step:  0 , action: [-0.999]
Episode =   64 (duration of 68 steps); Reward =  93.317 (best =  93.757, in epi
52)
        Episode training time:  2.0292861461639404
        resetting episode... next explore_p:  0.0041745579179292966
Garbage collector: collected 0 objects.
        step:  0 , action: [-0.96]
Episode =   65 (duration of 68 steps); Reward =  93.329 (best =  93.757, in epi
52)
        Episode training time:  2.0510199069976807
        resetting episode... next explore_p:  0.003757102126136367
Garbage collector: collected 0 objects.
        step:  0 , action: [-0.999]
Episode =   66 (duration of 70 steps); Reward =  93.167 (best =  93.757, in epi
52)
        Episode training time:  2.07840633392334
        resetting episode... next explore_p:  0.00338139191352273
Garbage collector: collected 0 objects.
        step:  0 , action: [-1.]
Episode =   67 (duration of 69 steps); Reward =  93.342 (best =  93.757, in epi
52)
        Episode training time:  1.8691227436065674
        resetting episode... next explore_p:  0.0030432527221704573
Garbage collector: collected 0 objects.
        step:  0 , action: [-0.841]
Episode =   68 (duration of 69 steps); Reward =  93.232 (best =  93.757, in epi
52)
        Episode training time:  2.0006282329559326
        resetting episode... next explore_p:  0.0027389274499534117
Garbage collector: collected 0 objects.
        step:  0 , action: [-1.]
Episode =   69 (duration of 68 steps); Reward =  93.309 (best =  93.757, in epi
52)
        Episode training time:  1.993483304977417
        resetting episode... next explore_p:  0.0024650347049580707
Garbage collector: collected 0 objects.
        step:  0 , action: [-0.997]
```


Scores
100
75
50
25
0
−25
−50
−75
−100
0
10
20
30
40
50
60
70

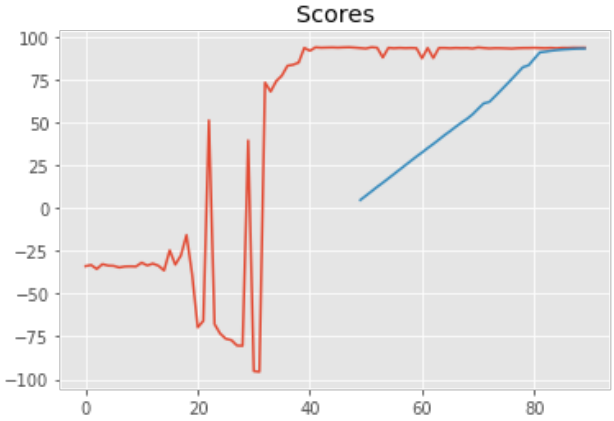

actions over steps
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
10
20
30
40
50
60
70

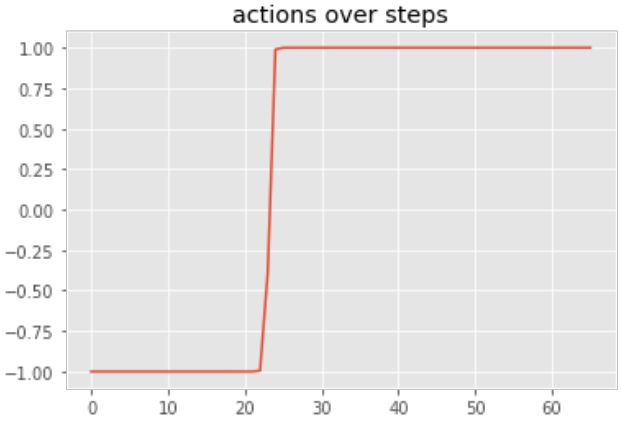
```
Episode =   70 (duration of 71 steps); Reward =  93.016 (best =  93.757, in epi
52)
        Episode training time:  2.461416244506836
        resetting episode... next explore_p:  0.0022185312344622636
Garbage collector: collected 5644 objects.
        step:  0 , action: [-1.]
Episode =   71 (duration of 67 steps); Reward =  93.612 (best =  93.757, in epi
52)
        Episode training time:  2.006866931915283
        resetting episode... next explore_p:  0.001996678111016037
Garbage collector: collected 0 objects.
        step:  0 , action: [-1.]
Episode =   72 (duration of 68 steps); Reward =  93.316 (best =  93.757, in epi
52)
        Episode training time:  2.019381284713745
        resetting episode... next explore_p:  0.0017970102999144335
Garbage collector: collected 0 objects.
        step:  0 , action: [-1.]
Episode =   73 (duration of 71 steps); Reward =  93.066 (best =  93.757, in epi
52)
        Episode training time:  2.117708206176758
        resetting episode... next explore_p:  0.0016173092699229901
Garbage collector: collected 0 objects.
        step:  0 , action: [-1.]
Episode =   74 (duration of 69 steps); Reward =  93.220 (best =  93.757, in epi
52)
        Episode training time:  2.0224270820617676
        resetting episode... next explore_p:  0.0014555783429306911
Garbage collector: collected 0 objects.
        step:  0 , action: [-1.]
Episode =   75 (duration of 69 steps); Reward =  93.199 (best =  93.757, in epi
52)
        Episode training time:  2.068068265914917
        resetting episode... next explore_p:  0.0013110020508637622
Garbage collector: collected 0 objects.
        step:  0 , action: [-1.]
Episode =   76 (duration of 70 steps); Reward =  93.094 (best =  93.757, in epi
52)
        Episode training time:  2.0547869205474854
        resetting episode... next explore_p:  0.0011790184577738598
Garbage collector: collected 0 objects.
        step:  0 , action: [-1.]
Episode =   77 (duration of 72 steps); Reward =  92.955 (best =  93.757, in epi
52)
        Episode training time:  2.1988954544067383
        resetting episode... next explore_p:  0.0010611166119964739
Garbage collector: collected 0 objects.
        step:  0 , action: [-1.]
Episode =   78 (duration of 69 steps); Reward =  93.197 (best =  93.757, in epi
52)
        Episode training time:  2.0157103538513184
        resetting episode... next explore_p:  0.0009550049507968265
Garbage collector: collected 0 objects.
        step:  0 , action: [-1.]
Episode =   79 (duration of 68 steps); Reward =  93.306 (best =  93.757, in epi
52)
        Episode training time:  2.01340651512146
        resetting episode... next explore_p:  0.0008595044557171439
Garbage collector: collected 0 objects.
        step:  0 , action: [-0.998]
```


Scores
100
75
50
25
0
−25
−50
−75
−100
0
10
20
30
40
50
60
70
80

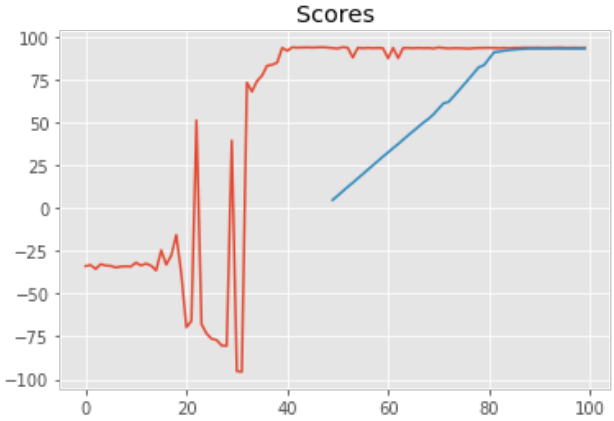

actions over steps
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
10
20
30
40
50
60
70

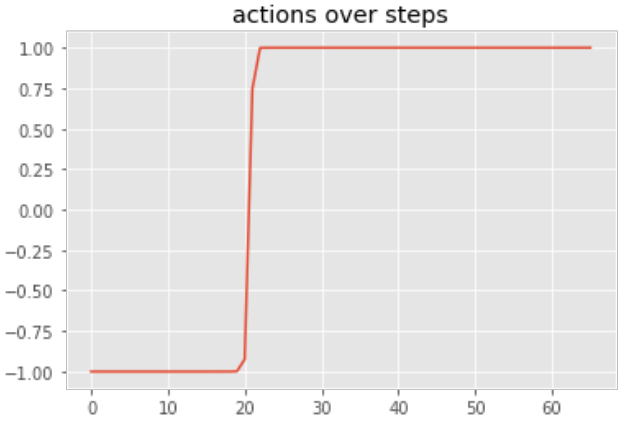
```
Episode =   80 (duration of 68 steps); Reward =  93.326 (best =  93.757, in epi
52)
        Episode training time:  2.3524293899536133
        resetting episode... next explore_p:  0.0007735540101454295
Garbage collector: collected 5704 objects.
        step:  0 , action: [-1.]
Episode =   81 (duration of 67 steps); Reward =  93.403 (best =  93.757, in epi
52)
        Episode training time:  1.9794421195983887
        resetting episode... next explore_p:  0.0006961986091308866
Garbage collector: collected 0 objects.
        step:  0 , action: [-1.]
Episode =   82 (duration of 68 steps); Reward =  93.298 (best =  93.757, in epi
52)
        Episode training time:  2.011418342590332
        resetting episode... next explore_p:  0.0006265787482177979
Garbage collector: collected 0 objects.
        step:  0 , action: [-1.]
Episode =   83 (duration of 69 steps); Reward =  93.212 (best =  93.757, in epi
52)
        Episode training time:  2.028540849685669
        resetting episode... next explore_p:  0.0005639208733960181
Garbage collector: collected 0 objects.
        step:  0 , action: [-1.]
Episode =   84 (duration of 67 steps); Reward =  93.344 (best =  93.757, in epi
52)
        Episode training time:  1.9897162914276123
        resetting episode... next explore_p:  0.0005075287860564164
Garbage collector: collected 0 objects.
        step:  0 , action: [-1.]
Episode =   85 (duration of 70 steps); Reward =  93.152 (best =  93.757, in epi
52)
        Episode training time:  2.00927472114563
        resetting episode... next explore_p:  0.0004567759074507747 6
Garbage collector: collected 0 objects.
        step:  0 , action: [-1.]
Episode =   86 (duration of 67 steps); Reward =  93.404 (best =  93.757, in epi
52)
        Episode training time:  1.9644224643707275
        resetting episode... next explore_p:  0.0004110983167056973
Garbage collector: collected 0 objects.
        step:  0 , action: [-1.]
Episode =   87 (duration of 67 steps); Reward =  93.380 (best =  93.757, in epi
52)
        Episode training time:  1.9864873886108398
        resetting episode... next explore_p:  0.0003699884850351276
Garbage collector: collected 0 objects.
        step:  0 , action: [-1.]
Episode =   88 (duration of 66 steps); Reward =  93.505 (best =  93.757, in epi
52)
        Episode training time:  1.7859289646148682
        resetting episode... next explore_p:  0.0003329896365316148 6
Garbage collector: collected 0 objects.
        step:  0 , action: [-1.]
Episode =   89 (duration of 66 steps); Reward =  93.471 (best =  93.757, in epi
52)
        Episode training time:  1.9327120780944824
        resetting episode... next explore_p:  0.0002996906728784534
Garbage collector: collected 0 objects.
        step:  0 , action: [-1.]
```


Scores
100
75
50
25
0
−25
−50
−75
−100
0
20
40
60
80

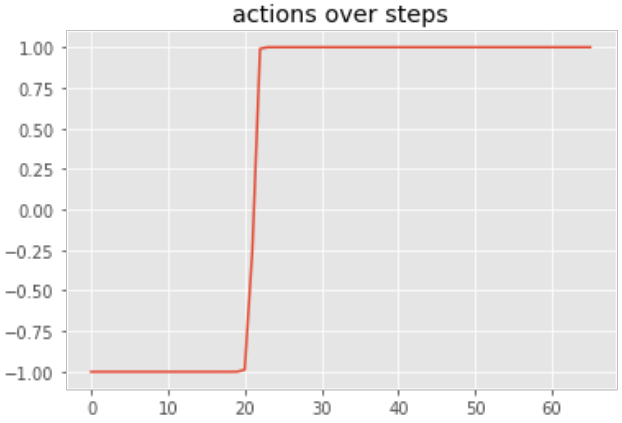

actions over steps
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
10
20
30
40
50
60

```
Episode =   90 (duration of 66 steps); Reward =  93.488 (best =  93.757, in epi
52)
        Episode training time:  2.279244899749756
        resetting episode... next explore_p:  0.00026972160559060804
Garbage collector: collected 5384 objects.
        step:  0 , action: [-1.]
Episode =   91 (duration of 66 steps); Reward =  93.499 (best =  93.757, in epi
52)
        Episode training time:  1.8783533573150635
        resetting episode... next explore_p:  0.00024274944503154723
Garbage collector: collected 0 objects.
        step:  0 , action: [-1.]
Episode =   92 (duration of 67 steps); Reward =  93.389 (best =  93.757, in epi
52)
        Episode training time:  2.0192959308624268
        resetting episode... next explore_p:  0.00021847450052839252
Garbage collector: collected 0 objects.
        step:  0 , action: [-1.]
Episode =   93 (duration of 66 steps); Reward =  93.469 (best =  93.757, in epi
52)
        Episode training time:  1.9497535228729248
        resetting episode... next explore_p:  0.00019662705047555326
Garbage collector: collected 0 objects.
        step:  0 , action: [-1.]
Episode =   94 (duration of 66 steps); Reward =  93.481 (best =  93.757, in epi
52)
        Episode training time:  1.919541597366333
        resetting episode... next explore_p:  0.00017696434542799794
Garbage collector: collected 0 objects.
        step:  0 , action: [-1.]
Episode =   95 (duration of 65 steps); Reward =  93.598 (best =  93.757, in epi
52)
        Episode training time:  1.9421331882476807
        resetting episode... next explore_p:  0.00015926791088519815
Garbage collector: collected 0 objects.
        step:  0 , action: [-1.]
Episode =   96 (duration of 66 steps); Reward =  93.450 (best =  93.757, in epi
52)
        Episode training time:  1.9798738956451416
        resetting episode... next explore_p:  0.00014334111979667834
Garbage collector: collected 0 objects.
        step:  0 , action: [-1.]
Episode =   97 (duration of 67 steps); Reward =  93.414 (best =  93.757, in epi
52)
        Episode training time:  1.9979231357574463
        resetting episode... next explore_p:  0.00012900700781701051
Garbage collector: collected 0 objects.
        step:  0 , action: [-1.]
Episode =   98 (duration of 66 steps); Reward =  93.485 (best =  93.757, in epi
52)
        Episode training time:  1.8579635620117188
        resetting episode... next explore_p:  0.00011610630703530947
Garbage collector: collected 0 objects.
        step:  0 , action: [-1.]
Episode =   99 (duration of 67 steps); Reward =  93.371 (best =  93.757, in epi
52)
        Episode training time:  1.9081997871398926
        resetting episode... next explore_p:  0.00010449567633177853
Garbage collector: collected 0 objects.
        step:  0 , action: [-1.]
```

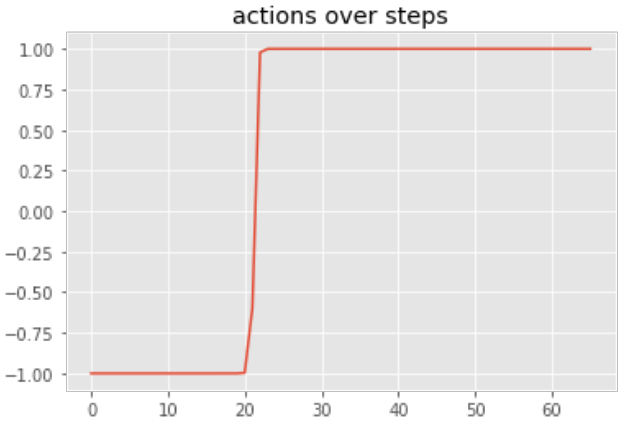




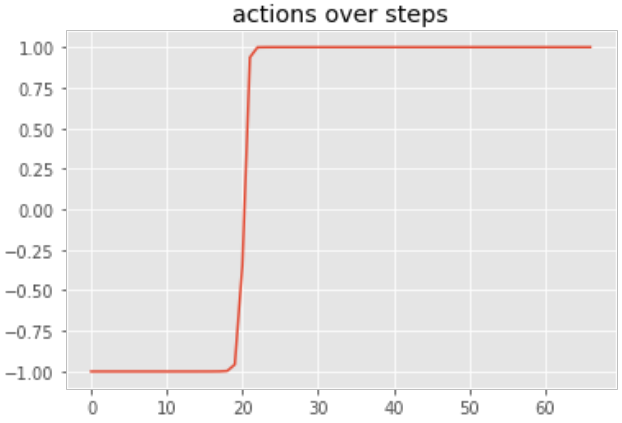

```
Episode =  100 (duration of 66 steps); Reward =  93.459 (best =  93.757, in epi
52)
        Episode training time:  2.316049575805664

*****************************

*** All episodes training time (HH:MM:SS):  0:11:45.021954
        Average training time per episode:  7.050219540596008
*****************************
[TEST] Training Done, now running tests...
        resetting episode... next explore_p:  9.404610869860067e-05
Garbage collector: collected 5449 objects.
        step:  0 , action: [-1.]
        step:  50 , action: [1.]
```



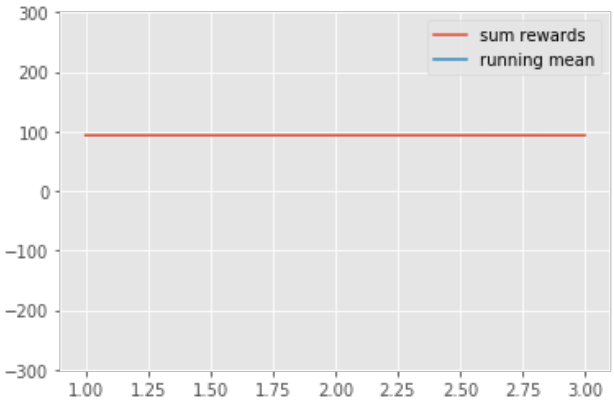

```
        resetting episode... next explore_p:  8.464149782874061e-05
Garbage collector: collected 2712 objects.
        step:  0 , action: [-1.]
        step:  50 , action: [1.]
```




```
        resetting episode... next explore_p:  7.617734804586655e-05
Garbage collector: collected 2712 objects.
        step:  0 , action: [-1.]
        step:  50 , action: [1.]
```



```
******************************

*** All episodes training time (HH:MM:SS):  0:00:01.096472
        Average training time per episode:  0.3654906749725342
******************************
```




```
Final score: 93.53995681524269
```

Comparing Q-Table, Q-Network, and DDPG

The Q table, Q Network, have the relatively simpler task of solving an environment where the action space has been discretized into two values, -1 and 1. The DDPG agent is challenged with solving the more challenging continuous action space between -1 and 1.

- Q-Table was able to get close to solving the discrete action version of Mt Climber in around 50,000 episdes
- Q-Network was unable to solve the discrete version of Mt Climber in 2000 episodes, appearing to get stuck in a bad policy
- DDPG Agent was able to solve the continuous (more challenging than discrete) version of Mt Climber in around 35 episodes.

4. Lunar Lander

"LunarLander-v2 / LunarLanderContinuous-v2

Landing pad is always at coordinates (0,0). Coordinates are the first two numbers in state vector. Reward for moving from the top of the screen to landing pad and zero speed is about 100..140 points. If lander moves away from landing pad it loses reward back. Episode finishes if the lander crashes or comes to rest, receiving additional -100 or +100 points. Each leg ground contact is +10. Firing main engine is -0.3 points each frame. Solved is 200 points. Landing outside landing pad is possible. Fuel is infinite, so an agent can learn to fly and then land on its first attempt. Action is two real values vector from -1 to +1. First controls main engine, -1..0 off, 0..+1 throttle from 50% to 100% power. Engine can't work with less than 50% power. Second value -1.0..-0.5 fire left engine, +0.5..+1.0 fire right engine, -0.5..0.5 off."

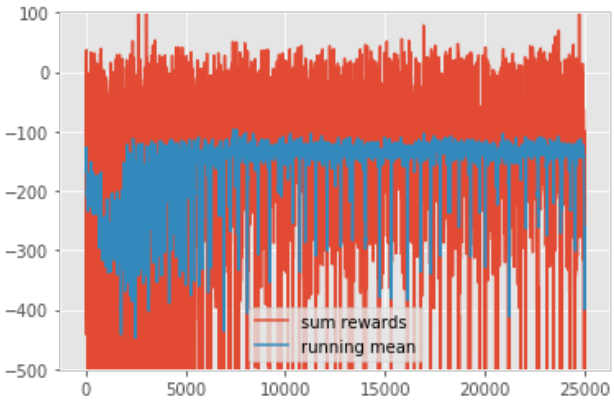
4.1 Simple Q Learning Agent

The results of 25,000 episodes of the Q learning agent are below.

Unlike Mt Climber environment, the simple Q learning agent was unable to get anywhere close to solving this environment (solve is 200 points). Additionally, the agent does not appear to be showing any upward process in running mean reward.

In [37]:
```
print("lunar lander with simple discretized Q learning agent")
dir = "results/lunarlander/"
plot_score_from_file(dir + "20190629195846LunarLander-v2_train.txt", -500, 100,
1)
```

```
lunar lander with simple discretized Q learning agent
```

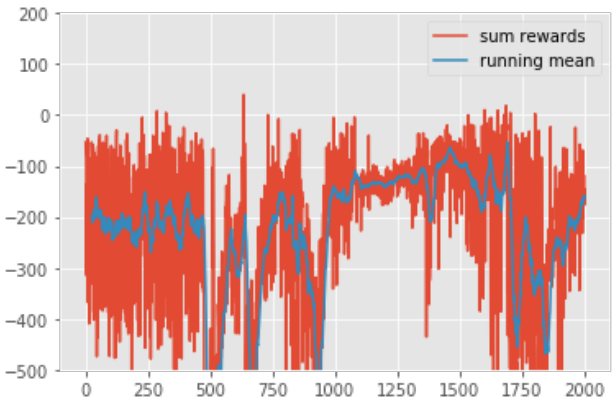


4.2 DDPG

With the Mt Climber settings this network was able to show control of the boosters to stabilize the agent flight. However, the agent was not able to solve the enviroment.

In [38]:
```
print("LR 0.0001")
dir = "results/lunarlandercontinuous/learningRate/0.0001/"
plot_score_from_file(dir + "20190629203656LunarLanderContinuous-v2_train.txt",
-500, 200, 1)

print("LR 0.0002")
dir = "results/lunarlandercontinuous/learningRate/0.0002/"
plot_score_from_file(dir + "20190629224211LunarLanderContinuous-v2_train.txt",
-500, 200, 1)

print("LR 0.0003")
dir = "results/lunarlandercontinuous/learningRate/0.0003/"
plot_score_from_file(dir + "20190630081636LunarLanderContinuous-v2_train.txt",
-500, 200, 1)
```

LR 0.0001



LR 0.0002

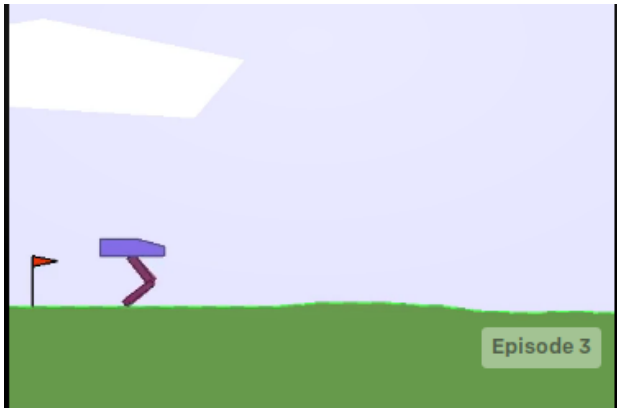
LR 0.0003

5. Bipedal Walker

Since this environment is continuous Action Spaces Only, it was only compatiable with the DDPG agent.

"BipedalWalker-v2

Reward is given for moving forward, total 300+ points up to the far end. If the robot falls, it gets -100. Applying motor torque costs a small amount of points, more optimal agent will get better score. State consists of hull angle speed, angular velocity, horizontal speed, vertical speed, position of joints and joints angular speed, legs contact with ground, and 10 lidar rangefinder measurements. There's no coordinates in the state vector."

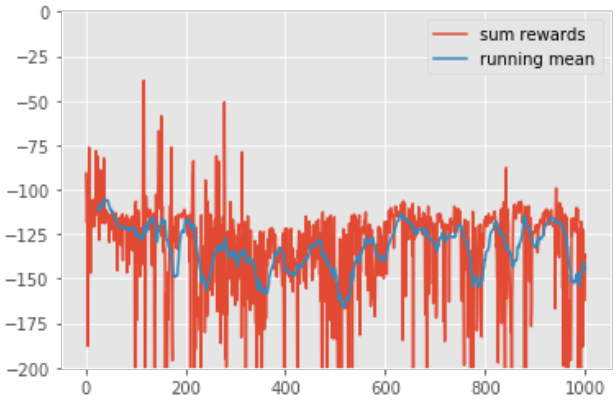


5.1 DDPG

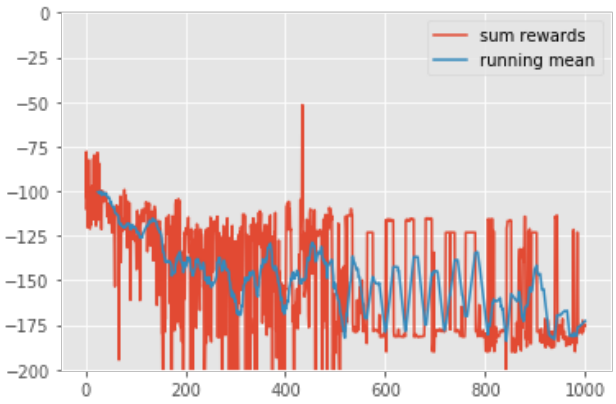
The agent was able to demonstate walking, however was not able to solve the environment.

In [39]:
```
from visuals import plot_score_from_file

print("LR 0.0001")
dir = "results/walker/learningrate/0.0001/"
plot_score_from_file(dir + "20190630222542BipedalWalker-v2_train.txt", -200, 0,
1)

print("LR 0.0005")
dir = "results/walker/learningrate/0.0005/"
plot_score_from_file(dir + "20190630222529BipedalWalker-v2_train.txt", -200, 0,
1)
```

```
LR 0.0001
```




```
LR 0.0005
```



6. Racing Car

This final agent needed some special supporting functions to pre-process image data. Functions to normalize the image to the range of 0-1, and convert to grayscale, we used based on the DDQN reference agent from the OpenAI gym leaderboard.

In [40]:
```
"""
# Sets all pixel values to be between (0,1)
# Parameters:
# - image: A grayscale (nxmx1) or RGB (nxmx3) array of floats
# Outputs:
# - image rescaled so all pixels are between 0 and 1
"""
def sample_unit_image(image):
    return np.true_divide(image, 255.0)

"""
# Converts an RGB image to grayscale
# Parameters:
# - image: An RGB (nxmx3) array of floats
# Outputs:
# - A (nxmx1) array of floats in the range [0,255] representing a
#   weighted average of the color channels of 'image'
"""
def sample_grayscale_img(image):
    return np.dot(image[..., :3], [0.299, 0.587, 0.114])
```

Furthermore, because this and other environments sometimes have action spaces of different ranges, it was necessary to replace the lambda function in the actor network with an output scaling function to interface properly with any OpenAI gym environment.

In [41]:
```
"""
# scales the output actions from the network
# this is important for multi dimensional actions with different ranges and low
/hgh values
"""
def sample_scale_output(x, action_range, action_low):
    temp = (np.array(x) * np.array(action_range)) + np.array(action_low)
    return temp
```

The following network architecture was chosen based on cifar10_cnn practice project from Udacity [13], including max pooling layers to speed up processing, and the DDQN with Dropout OpenAI leaderboard implementation [14].

********** *** DDPG Agent Paramter *** - network architecture chosen: imageInputGrayscale [ACTOR MODEL SUMMARY] _____ Layer (type) Output Shape Param # === states (InputLayer) (None, 96, 96, 1) 0 _____ conv2d_1 (Conv2D) (None, 24, 24, 32) 2080 _____ max_pooling2d_1 (MaxPooling2 (None, 12, 12, 32) 0 _____ dropout_1 (Dropout) (None, 12, 12, 32) 0 _____ conv2d_2 (Conv2D) (None, 6, 6, 64) 32832 _____ max_pooling2d_2 (MaxPooling2 (None, 3, 3, 64) 0 _____ dropout_2 (Dropout) (None, 3, 3, 64) 0 _____ conv2d_3 (Conv2D) (None, 3, 3, 64) 36928 _____ max_pooling2d_3 (MaxPooling2 (None, 1, 1, 64) 0 _____ dropout_3 (Dropout) (None, 1, 1, 64) 0 _____ flatten_1 (Flatten) (None, 64) 0 _____ dense_1 (Dense) (None, 512) 33280 _____ dropout_4 (Dropout) (None, 512) 0 _____ raw_actions (Dense) (None, 3) 1539 === Total params: 106,659 Trainable params:

106,659 Non-trainable params: 0 _____ [CRITIC MODEL SUMMARY]

_____ Layer (type) Output Shape Param # Connected to

==========

states (InputLayer) (None, 96, 96, 1) 0

_____

conv2d_7 (Conv2D) (None, 24, 24, 32) 2080 states[0][0]

_____

max_pooling2d_7 (MaxPooling2D) (None, 12, 12, 32) 0 conv2d_7[0][0]

_____

dropout_9 (Dropout) (None, 12, 12, 32) 0 max_pooling2d_7[0][0]

_____

conv2d_8 (Conv2D) (None, 6, 6, 64) 32832 dropout_9[0][0]

_____

max_pooling2d_8 (MaxPooling2D) (None, 3, 3, 64) 0 conv2d_8[0][0]

_____

dropout_10 (Dropout) (None, 3, 3, 64) 0 max_pooling2d_8[0][0]

_____

conv2d_9 (Conv2D) (None, 3, 3, 64) 36928 dropout_10[0][0]

_____

max_pooling2d_9 (MaxPooling2D) (None, 1, 1, 64) 0 conv2d_9[0][0]

_____

dropout_11 (Dropout) (None, 1, 1, 64) 0 max_pooling2d_9[0][0]

_____

flatten_3 (Flatten) (None, 64) 0 dropout_11[0][0]

_____

dense_4 (Dense) (None, 512) 33280 flatten_3[0][0]

_____

actions (InputLayer) (None, 3) 0

_____

dropout_12 (Dropout) (None, 512) 0 dense_4[0][0]

_____

dense_3 (Dense) (None, 512) 2048 actions[0][0]

_____

add_1 (Add) (None, 512) 0 dropout_12[0][0] dense_3[0][0]

_____

q_values (Dense) (None, 1) 513 add_1[0][0]

==========

Total params: 107,681 Trainable params: 107,681 Non-trainable params: 0

_____ - action_repeat: 1 - learningRate: 0.0001 - learnFrequency: 1 - dropout_rate: 0.2 - buffer_size: 100000 - batch_size: 32 - gamma: 0.99 - useSoftUpdates: True - tau: 0.01 - explore_start: 1.0 - explore_stop: 0.0 - explore_decay_rate: 0.95

****

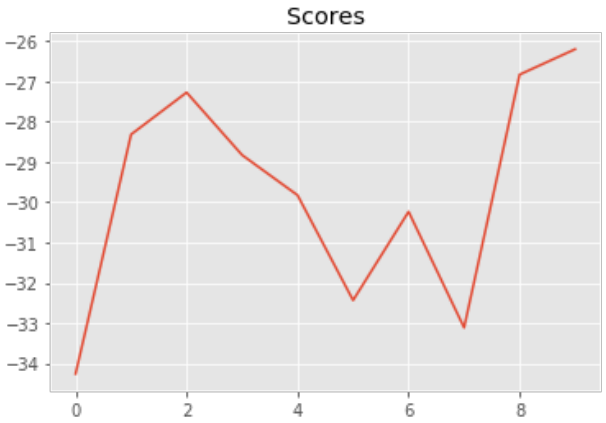
Unfortunately, there was a memory leak in this image state based implementation.

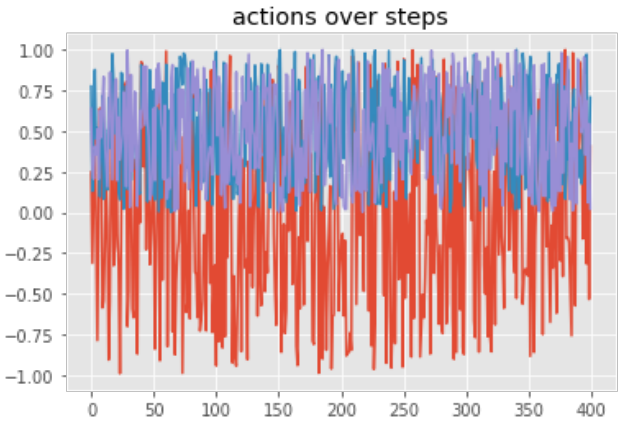
Because of this, the agent was unable to be trained for more than around 100 episodes without running out of memory and crashing the computer. Garbage collection (gc) from python returns around 300-500 unreachable garbage objects each episode of running the Car Racing environment.

Due to insufficient time remaining for the project (Mt Climber experiments and making the Car Racing envionment work at all was a significant investment), the memory leak was unable to be found and this stage of the project was forced to be abandoned.

In [43]:
```
import gym
import numpy as np

"""
# Create an environment and set random seed
"""
selectedEnvironment = 9
env = 0
envName = 0

# Toy Text - Discrete state and action space
if selectedEnvironment == 0:
    envName = 'Taxi-v2'

# Classic Control - Continuous State and Discrete Action Spaces
elif selectedEnvironment == 1:
    envName = 'MountainCar-v0' # needs Discretized or better
elif selectedEnvironment == 2:
    envName = 'Acrobot-v1'     # needs Discretized, Tile Encoding or better
elif selectedEnvironment == 3:
    envName = 'CartPole-v1'    # needs Deep Q Learning to do well?

# Box 2D - Continuous State, Discrete Actions
elif selectedEnvironment == 4:
    envName = 'LunarLander-v2' # discrete actions, continuous state

# Classic Control - Continuous State and Action Spaces
elif selectedEnvironment == 5:
    envName = 'Pendulum-v0' # continuous only
elif selectedEnvironment == 6:
    envName = 'MountainCarContinuous-v0' # continuous only

# Box 2D - Continuous State and Action Spaces
elif selectedEnvironment == 7:
    envName = 'LunarLanderContinuous-v2' # continuous only
elif selectedEnvironment == 8:
    envName = 'BipedalWalker-v2'  # continuous only

# Box 2D - Image State and Continuous Action Spaces
elif selectedEnvironment == 9:
    envName = 'CarRacing-v0'      #  image input, actions [steer, gas, brake]

# Initialize the environment
env = gym.make(envName)
env.reset()

# Set output file paths based on environment
from visuals import examine_environment, examine_environment_MountainCar_discre
tized, examine_environment_Acrobat_tiled
#examine_environment(env)

from datetime import datetime
FORMAT = '%Y%m%d%H%M%S'
file_output_train = envName + '_train.txt'       # file name for saved results
file_output_test = envName + '_test.txt'       # file name for saved results
file_output_train = datetime.now().strftime(FORMAT) + file_output_train

print('------------------------------------------------------------')
print('New Experiment, training output file name: ', file_output_train)

"""
# Create Agent
"""
```

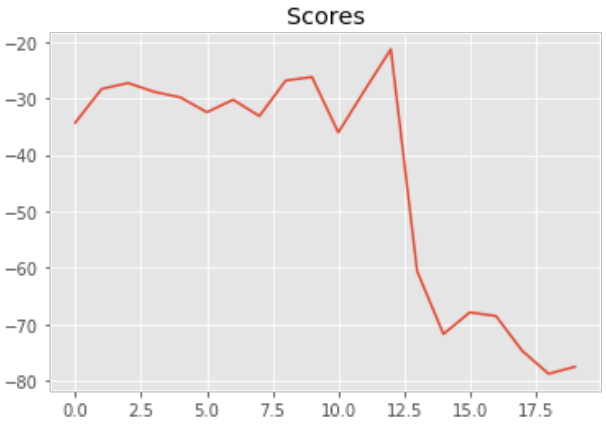
```
Track generation: 1179..1478 -> 299-tiles track
----------------------------------------------------------
New Experiment, training output file name:  20190702235525CarRacing-v0_train.tx
t
env.observation_space:  (96, 96, 3)
**************************************
**************************************
Initializing DDPG Agent
        Environment:  <TimeLimit<CarRacing<CarRacing-v0>>>
env.action_space.shape (3,)
env.action_space.low [-1.  0.  0.]
env.action_space.high [1. 1. 1.]
*** init actor ***
self.action_range:  [2. 1. 1.]
*** init actor ***
self.action_range:  [2. 1. 1.]
**************************************
*** DDPG Agent Paramter ***
- network architecture chosen:  imageInputGrayscale
[ ACTOR MODEL SUMMARY ]
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
states (InputLayer)          (None, 96, 96, 1)         0
_________________________________________________________________
conv2d_13 (Conv2D)           (None, 24, 24, 32)        2080
_________________________________________________________________
max_pooling2d_13 (MaxPooling (None, 12, 12, 32)        0
_________________________________________________________________
dropout_33 (Dropout)         (None, 12, 12, 32)        0
_________________________________________________________________
conv2d_14 (Conv2D)           (None, 6, 6, 64)          32832
_________________________________________________________________
max_pooling2d_14 (MaxPooling (None, 3, 3, 64)          0
_________________________________________________________________
dropout_34 (Dropout)         (None, 3, 3, 64)          0
_________________________________________________________________
conv2d_15 (Conv2D)           (None, 3, 3, 64)          36928
_________________________________________________________________
max_pooling2d_15 (MaxPooling (None, 1, 1, 64)          0
_________________________________________________________________
dropout_35 (Dropout)         (None, 1, 1, 64)          0
_________________________________________________________________
flatten_5 (Flatten)          (None, 64)                0
_________________________________________________________________
dense_25 (Dense)             (None, 512)               33280
_________________________________________________________________
dropout_36 (Dropout)         (None, 512)               0
_________________________________________________________________
raw_actions (Dense)          (None, 3)                 1539
=================================================================
Total params: 106,659
Trainable params: 106,659
Non-trainable params: 0
_________________________________________________________________
[ CRITIC MODEL SUMMARY ]
__________________________________________________________________________________
____________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================
==================
states (InputLayer)             (None, 96, 96, 1)    0
__________________________________________________________________________________
```

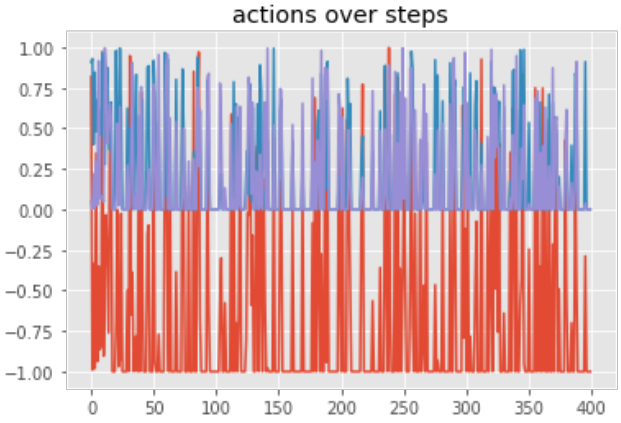

Scores
−26
−27
−28
−29
−30
−31
−32
−33
−34
0
2
4
6
8

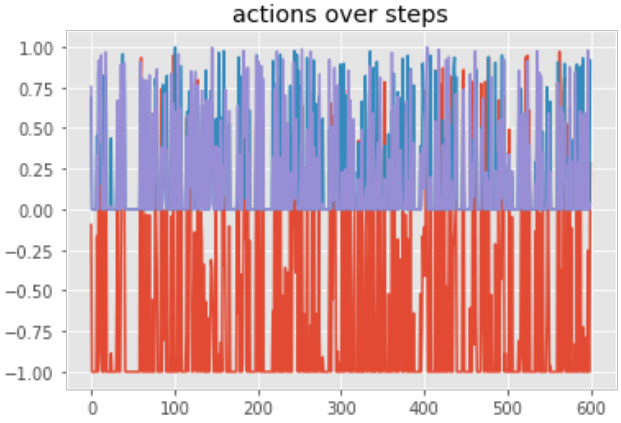

actions over steps
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
50
100
150
200
250
300
350
400


actions over steps
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
100
200
300
400
500
600

```
Episode =   10 (duration of 1000 steps); Reward = -26.199 (best = -26.199, in e
pi 10)
        Episode training time:  10.954565048217773
Track generation: 1176..1474 -> 298-tiles track
        resetting episode... next explore_p:  1.0
Garbage collector: collected 8699 objects.
        step:  0 , action: [-0.634  0.537  0.174]
        step:  100 , action: [-0.083  0.025  0.995]
        step:  200 , action: [0.284 0.34  0.363]
        step:  300 , action: [-0.297  0.929  0.929]
        step:  400 , action: [0.25  0.051 0.629]
        step:  500 , action: [-0.513  0.059  0.161]
        step:  600 , action: [0.854 0.347 0.723]
        step:  700 , action: [0.987 0.974 0.948]
        step:  800 , action: [0.363 0.219 0.308]
        step:  900 , action: [0.895 0.717 0.216]
Episode =   11 (duration of 1000 steps); Reward = -36.027 (best = -26.199, in e
pi 10)
        Episode training time:  50.252204179763794
Track generation: 1165..1460 -> 295-tiles track
        resetting episode... next explore_p:  0.9
Garbage collector: collected 173 objects.
        step:  0 , action: [0.851 0.305 0.37 ]
        step:  100 , action: [-0.036  0.295  0.675]
        step:  200 , action: [0.418 0.69  0.423]
        step:  300 , action: [-0.359  0.812  0.683]
        step:  400 , action: [0.453 0.775 0.298]
        step:  500 , action: [-0.523  0.49   0.286]
        step:  600 , action: [0.861 0.297 0.853]
        step:  700 , action: [-0.444  0.447  0.726]
        step:  800 , action: [-0.262  0.548  0.379]
        step:  900 , action: [-0.596  0.802  0.035]
Episode =   12 (duration of 1000 steps); Reward = -28.571 (best = -26.199, in e
pi 10)
        Episode training time:  48.26182556152344
Track generation: 1381..1725 -> 344-tiles track
        resetting episode... next explore_p:  0.81
Garbage collector: collected 443 objects.
        step:  0 , action: [-0.859  0.694  0.046]
        step:  100 , action: [0.709 0.259 0.609]
        step:  200 , action: [0.333 0.023 0.201]
        step:  300 , action: [0.44  0.704 0.052]
        step:  400 , action: [-0.511  0.075  0.346]
        step:  500 , action: [0.248 0.283 0.877]
        step:  600 , action: [0.431 0.652 0.802]
        step:  700 , action: [-1.000e+00  1.000e+00  4.046e-07]
        step:  800 , action: [-0.653  0.851  0.825]
        step:  900 , action: [-0.631  0.526  0.808]
Episode =   13 (duration of 1000 steps); Reward = -21.283 (best = -21.283, in e
pi 13)
        Episode training time:  49.151124238967896
Track generation: 1102..1382 -> 280-tiles track
        resetting episode... next explore_p:  0.7290000000000001
Garbage collector: collected 578 objects.
        step:  0 , action: [-9.997e-01  9.999e-01  5.798e-05]
        step:  100 , action: [-1.000e+00  1.000e+00  2.006e-07]
        step:  200 , action: [-0.286  0.556  0.984]
        step:  300 , action: [-0.861  0.49   0.931]
        step:  400 , action: [-0.353  0.771  0.178]
        step:  500 , action: [-1.00e+00  1.00e+00  1.61e-07]
        step:  600 , action: [-0.466  0.652  0.729]
        step:  700 , action: [-1.000e+00  1.000e+00  1.009e-07]
        step:  800 , action: [0.493 0.82  0.751]
```








```
Episode =   20 (duration of 1000 steps); Reward = -77.492 (best = -21.283, in e
pi 13)
        Episode training time:  50.562713384628296

******************************

*** All episodes training time (HH:MM:SS):  0:10:01.120742
        Average training time per episode:  30.05603711605072
******************************
```

Conclusions

The first question of this project, how well an agent can generalize between different tasks, was inconclusive. At least it can be said that this particular agent did not appear to generalize very well.

Through many experiments on Mountain Climber (ate up most of the project time), I was able to create an agent that performed quite well by leaderboard standards. However, when transferred to the Lunar Lander and Walker environments, while the agent was able to learn some degree of control (in flight boosters and walking joints respectively), it was unable to solve the environment even after a few small changes in parameters. This leads me to believe that the agent does not in fact generalize well.

The following lessons in agent design were learned:

Effective

- Larger batch size: provided that the learning rate was not too large, a larger batch size generally helped performance and number of episodes to find a solution.
- Tuning the learning rate: This seemed like
- Soft updates: Without this, the agent performed very poorly.

Unclear

- Dropout. At least this didn't seem to hurt for values below 0.3. It also appeared to provide some stability and regularization against overfitting.
- Explore. Small amount of exploration, correlated (seemed slightly better) or uncorrelated, with 5-10% decay per episode. Exploration is deemed important in more papers, however finding the right balance was very tricky. Too much exploration and the agent takes much longer to find a soution. Too little exploration and the agent may not find a solution at all. For Mountain Climber, this did not seem to be imporatnt besides the initial random exploration to fill a buffer of enough steps (I settled on 10,000 steps as the min frames to start learning, though other papers use much more, often up to 80,000 [17]). One thing that way clear was that pure exploitation was essential for the network to properly learn, and adding noise to network predictions made learning much harder. The lesson here was either explore or exploit, but trying to do both seems to confuse the network. Better to be decisive.
- larger amount of neurons per layer. This didn't seem to hurt the ability to learn, though with more parameters learning look longer per episode, and took longer to converge to a solution. Assuming the agent had enough capacity to learn the task, it seems that a smaller network learns faster and is generally fairly stable.
- Action repeat. Did not seem to provide any benefit, but didn't seem to hurt either. Instead of this it would probably be better to directly put the prior states and actions into the network in a RNN style architecture.
- Batch normalization. This definitely smoothed out agent actions. It also slows down learning (both in training time per episode, and in number of episodes to find a solution), and appears to makes the final result less stable (more variation between episodes, even if it is able to solve the environemnt). Batch normalization appears to make fine control more difficult, in a similar way to adding too much noise and never allowing pure exploitation makes learning hard.
- Gamma, discount factor. This hyperparameter was not investigated. Further analysis required.

Ineffective

- L2 regularization. This appears to make the agents actions too in-precise to properly learn
- ELU activation function instead of RELU. Did not appear to provide any benefit and made performance worse.
- Learning Frequency. Didn't appear to add any value.
- Large amount of exploration (1% decay). Generally this made the agent perform much worse, it seems like it makes the agent take a long time to find a solution.
- Network input noise as exploration. This destabilized learning.

Stabilizing the training was difficult, with big variations in training performance from small changes in agent networks architecture or hyperparameters. This often lead to divergence in the training and the agent unable to solve the environment. When this happens the agent gets stuck in a local minimum, repeating some undesirable action for all-time, or simply forgetting how to complete a task it could do before (often called "catastrophic forgetting" for obvious reasons).

References

- [1] CONTINUOUS CONTROL WITH DEEP REINFORCEMENTLEARNING - DDPG - https://arxiv.org/pdf/1509.02971.pdf (https://arxiv.org/pdf/1509.02971.pdf)
- [2] Batch Normalization: Accelerating Deep Network Training byReducing Internal Covariate Shift - https://arxiv.org/pdf/1502.03167.pdf (https://arxiv.org/pdf/1502.03167.pdf)
- [3] Implementing Batch Normalation with Keras - https://www.dlology.com/blog/one-simple-trick-to-train-keras-model-faster-with-batch-normalization/ (https://www.dlology.com/blog/one-simple-trick-to-train-keras-model-faster-with-batch-normalization/)
- [4] Taming the Noise inReinforcement Learning via Soft Updates - http://www.auai.org/uai2016/proceedings/papers/219.pdf (http://www.auai.org/uai2016/proceedings/papers/219.pdf)
- [5] FAST AND ACCURATE DEEP NETWORK LEARNING BY EXPONENTIAL LINEAR UNITS(ELUS) - https://arxiv.org/pdf/1511.07289.pdf (https://arxiv.org/pdf/1511.07289.pdf)
- [6] ELU as a Neural Networks Activation Function - https://sefiks.com/2018/01/02/elu-as-a-neural-networks-activation-function/ (https://sefiks.com/2018/01/02/elu-as-a-neural-networks-activation-function/)
- [7] Dropout: A Simple Way to Prevent Neural Networks fromOverfitting - http://www.cs.toronto.edu/~rsalakhu/papers/srivastava14a.pdf (http://www.cs.toronto.edu/~rsalakhu/papers/srivastava14a.pdf)
- [8] ON THE CONVERGENCE OF ADAM AND BEYOND - https://openreview.net/pdf?id=ryQu7f-RZ (https://openreview.net/pdf?id=ryQu7f-RZ)
- [9] Adam: A Method for Stochastic Optimization - https://arxiv.org/abs/1412.6980v8 (https://arxiv.org/abs/1412.6980v8)
- [10] Finding Good Learning Rate and The One Cycle Policy - https://towardsdatascience.com/finding-good-learning-rate-and-the-one-cycle-policy-7159fe1db5d6 (https://towardsdatascience.com/finding-good-learning-rate-and-the-one-cycle-policy-7159fe1db5d6)
- [11] The False Promise of Off-Policy Reinforcement Learning Algorithms - https://towardsdatascience.com/the-false-promise-of-off-policy-reinforcement-learning-algorithms-c56db1b4c79a (https://towardsdatascience.com/the-false-promise-of-off-policy-reinforcement-learning-algorithms-c56db1b4c79a)
- [12] An Overview of Regularization Techniques in Deep Learning - https://www.analyticsvidhya.com/blog/2018/04/fundamentals-deep-learning-regularization-techniques/ (https://www.analyticsvidhya.com/blog/2018/04/fundamentals-deep-learning-regularization-techniques/)
- [13] cifar10_cnn practice project from Udacity - Fork: https://github.com/smmuzza/machine-learning/blob/master/projects/practice_projects/cnn/cifar10-classification/cifar10_cnn.ipynb (https://github.com/smmuzza/machine-learning/blob/master/projects/practice_projects/cnn/cifar10-classification/cifar10_cnn.ipynb)
- [14] DDQN with Dropout: CarRacing-v0 - https://github.com/AMD-RIPS/RL-2018/blob/master/documents/leaderboard/IPAM-AMD-Car_Racing.ipynb (https://github.com/AMD-RIPS/RL-2018/blob/master/documents/leaderboard/IPAM-AMD-Car_Racing.ipynb)
- [15] "The Learning Brain" Great Courses by Prof. Thad A. Polk
- [16] Playing Atari with Deep Reinforcement Learning - https://arxiv.org/pdf/1312.5602v1.pdf (https://arxiv.org/pdf/1312.5602v1.pdf)
- [17] Rainbow: Combining Improvements in Deep Reinforcement Learning - https://arxiv.org/pdf/1710.02298.pdf (https://arxiv.org/pdf/1710.02298.pdf)
- [18] Deep Reinforcement Learning with Double Q-learning - https://arxiv.org/pdf/1509.06461.pdf (https://arxiv.org/pdf/1509.06461.pdf)
- [19] Addressing Function Approximation Error in Actor-Critic Methods - https://arxiv.org/abs/1802.09477 (https://arxiv.org/abs/1802.09477)
- [20] Deep Q Networks for Atari Games - https://github.com/danielegrattarola/deep-q-atari (https://github.com/danielegrattarola/deep-q-atari)