Definition

Project Overview

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

Autonomous vehicles is one of the most direct applications of AI in robotics. Generally this is broken down into two primary components: perception and planning and control. This project will demonstate agent planning and control capabilities that could protentially be applied to the planning and control part of an autonomous vehicle system.

This project will 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 building a framework for OpenAI's gym to solve a variety of control problems.

Initially, the agent will be trained and improved based on simplier control tasks in "classic control", such as Mountain Car. Once it performs well, the model and paramters will be tested on more complicated environments in the Box2D section of OpenAI's gym, to see how well agents can transfer between environments in OpenAI. If generalization can be show, then it seems plausiable that these agents could also be used in a larger autonomy problem like that of autonomous vehicles.

- http://gym.openai.com/envs/#classic_control (http://gym.openai.com/envs/#classic_control)
- http://gym.openai.com/envs/#box2d (http://gym.openai.com/envs/#box2d)

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], [22], [23]).

Solution Statement

```
– a the solution proposed for the problem given
```

A solution will depend on the environment in question

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

Methodology

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.

Data Preprocessing

Not applicable to reinforcement learning. See handling different environment types below for something similar.

Implementation

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.

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([-0.4708512,  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)
```

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 [4]:
```
# 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: |
+---------+
  (North)
+---------+
|R: | : :G|
| : : : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
  (Pickup)
+---------+
|R: | : :G|
| : : : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
  (North)
+---------+
|R: | : :G|
| : : : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
  (North)
+---------+
|R: | : :G|
| : : : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
  (South)
+---------+
|R: | : :G|
| : : : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
  (East)
+---------+
|R: | : :G|
| : : : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
```

In [5]:
```
# 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: -10.0
State space: Box(2,)
State space samples:
[[ 0.075 -0.061]
 [ 0.47   0.062]
 [ 0.281 -0.038]
 [-0.034  0.047]
 [ 0.438  0.052]
 [-0.085  0.021]
 [-0.074  0.056]
 [-1.092  0.031]
 [-0.924  0.05 ]
 [ 0.213  0.012]]
Action space: Discrete(3)
Action space samples:
[2 1 2 2 2 1 0 0 0 1]
```

In [6]:
```
# 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: -0.20560413343981743
State space: Box(2,)
State space samples:
[[-0.844 -0.045]
 [-0.351 -0.035]
 [-0.179 -0.063]
 [-0.145  0.022]
 [ 0.26   0.032]
 [ 0.021  0.063]
 [-1.07  -0.019]
 [-0.686 -0.019]
 [-0.77  -0.062]
 [ 0.459  0.018]]
Action space: Box(1,)
Action space samples:
[[-0.288]
 [ 0.136]
 [-0.264]
 [-0.96 ]
 [-0.539]
 [-0.689]
 [-0.287]
 [-0.575]
 [-0.856]
 [ 0.512]]
```

In [7]:
```
# 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: 1012..1279 -> 267-tiles track
Final score: 10.278195488721808
State space: Box(96, 96, 3)
State space samples:
[[[[245  61 250]
   [187   6 165]
   [ 30 164  43]
   ...
   [247 218 235]
   [ 69 161  55]
   [ 35 193 173]]

  [[177 201  86]
   [126  59 186]
   [204  30 132]
   ...
   [112 213 149]
   [176 157 240]
   [ 34   7 185]]

  [[101 186  64]
   [  8 254 195]
   [226 211 217]
   ...
   [202 137  27]
   [ 71   6  73]
   [140  45 132]]

  ...

  [[ 36 150  91]
   [143   9 149]
   [ 76 118 159]
   ...
   [207 226  59]
   [167  96 171]
   [182 191  38]]

  [[ 98  18 121]
   [251 114 184]
   [245 113  41]
   ...
   [228 200  45]
   [ 57  30  36]
   [241  75 102]]

  [[ 60 248 142]
   [ 84  21 229]
   [ 26  45 113]
   ...
   [164 176  91]
   [ 37 223  90]
   [ 18 231 245]]]


 [[[248 208 204]
   [ 19 173  11]
   [ 73  88 156]
   ...
   [139 113  76]
   [187 202  94]
   [215 126  25]]
```

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 [8]:
```
# see agents/QLearningAgent.py
```

In [9]:
```
# see agents/QLearningAgentDisTiles.py
```

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 [10]:
```
# see agents/QNetwork.py
```

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.

The primary network used for most experiments (such as Mountain Car) in this report was "QuadCopterBig", listed in actor/critic sections of agents/DDPG.py.

Actor

- 4 dense layers of 128, 256, 256, 128, ReLU activation, 20% dropout
- [alternative for image state] 3 CNN layers (32, size 8x8, stride 4x4), (64, size 4x4, stride 2x2), (64, size 3x3, stride 1x1), one dense layer 512, RelU activations, 20% dropout
- Final softmax activation

Critic

- 2 dense state layers 128, 256, ReLU activation, 20% dropout
- [alternative for image state] 3 CNN layers (32, size 8x8, stride 4x4), (64, size 4x4, stride 2x2), (64, size 3x3, stride 1x1), one dense layer 512, RelU activations, 20% dropout
- 1 dense action laters 512 ReLU activation, 20% dropout
- 1 combined state-action layer, with additional 256 dense layer, ReLU activation, 20% dropout

DDPG

- Actor-critic
- Exploration-Exploit with percentage decay per episode
- Soft-update
- Random memory replay buffer for learning with 10,000 step warm-up
- Action Repeat

The image version of the DPPG agent (used for Car Racing) can be found under "imageStateContinuousAction" section of the DDPG agent, using the "imageInputGrayscale" network architecture for the actor and critic networks.

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 [11]:
```
# replay buffer (see agents/DDPG.py)
```

In [12]:
```
# Actor (see agents/DDPG.py)
```

In [13]:
```
# Critic (see agents/DDPG.py)
```

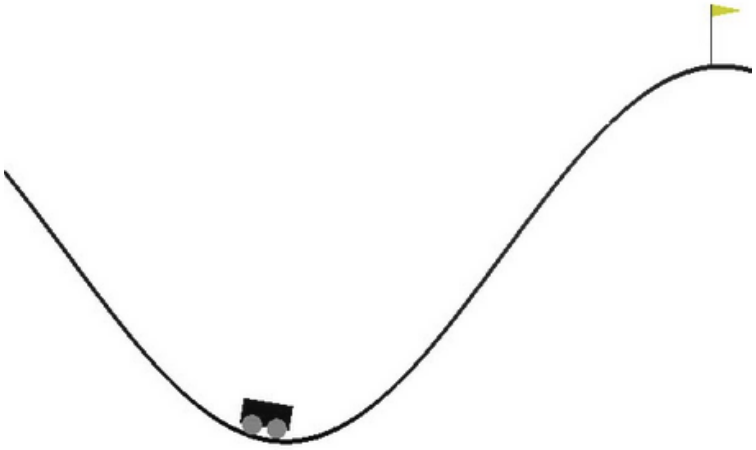
In [14]:
```
# DDPG Agent (see agents/DDPG.py)
```

Results & Refinement

Mountain Car

In order to compare different networks for initial experimentation, the Mountain Car 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.



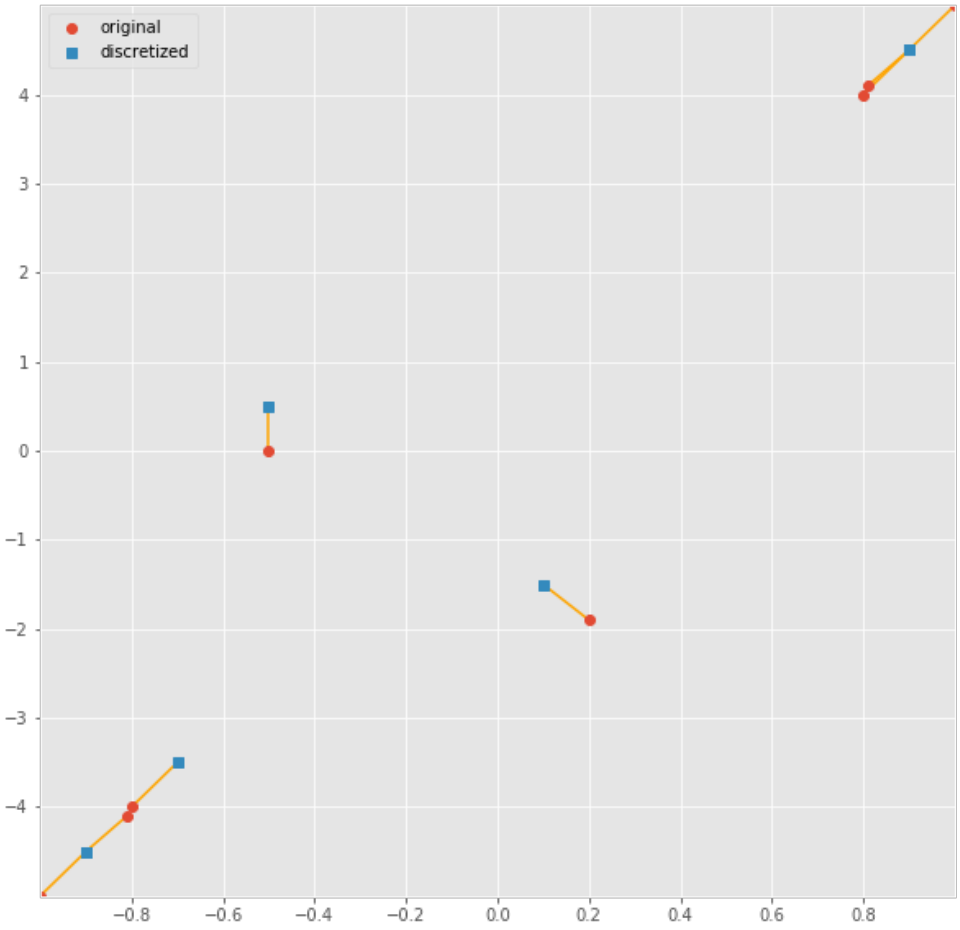
Simple Q Learning Agent

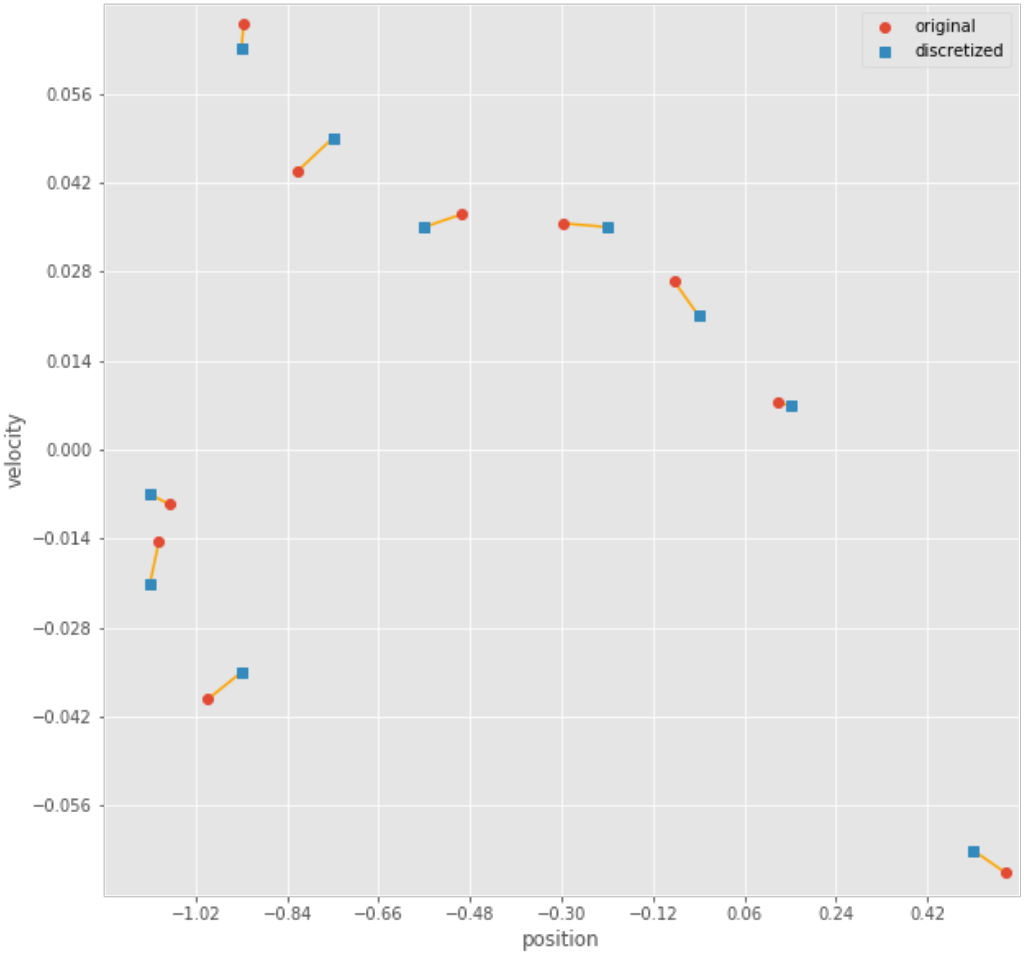
The results of experiments on mountain car for Q Learning Agent are below.

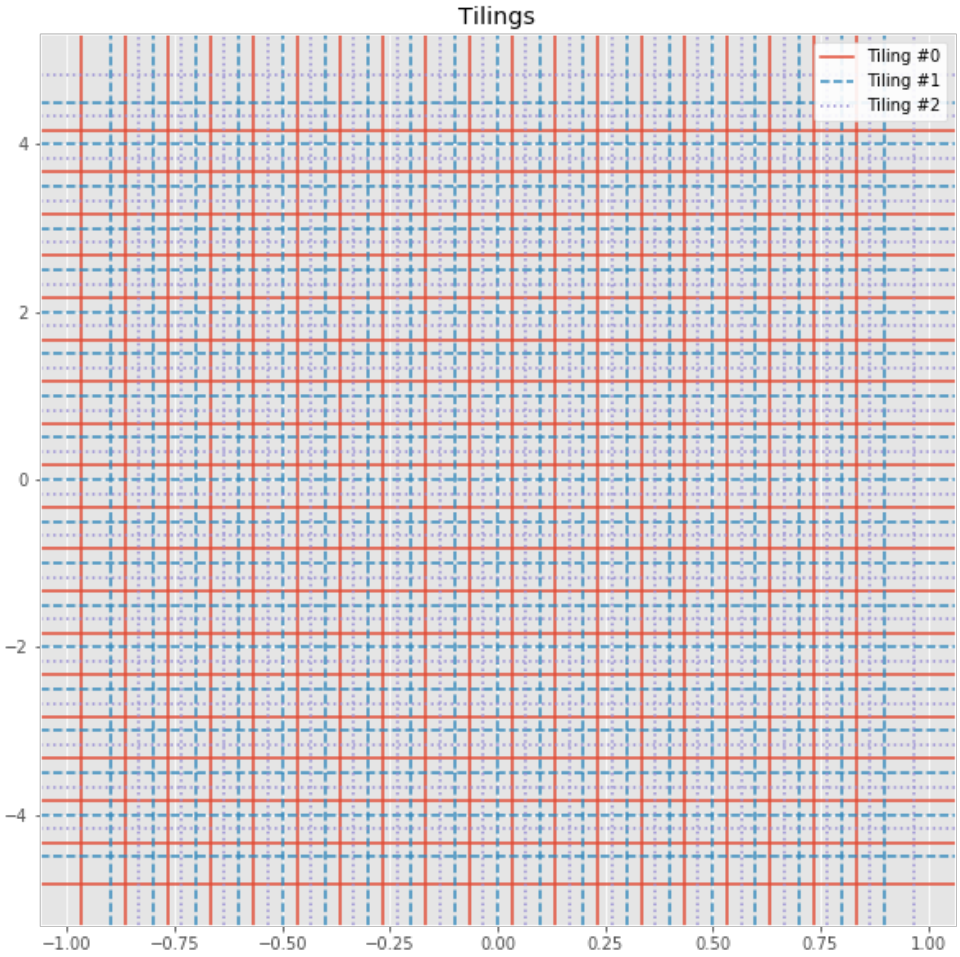
In [15]:
```
# First, let's take a look at the basic and tiled state discretization

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)
```

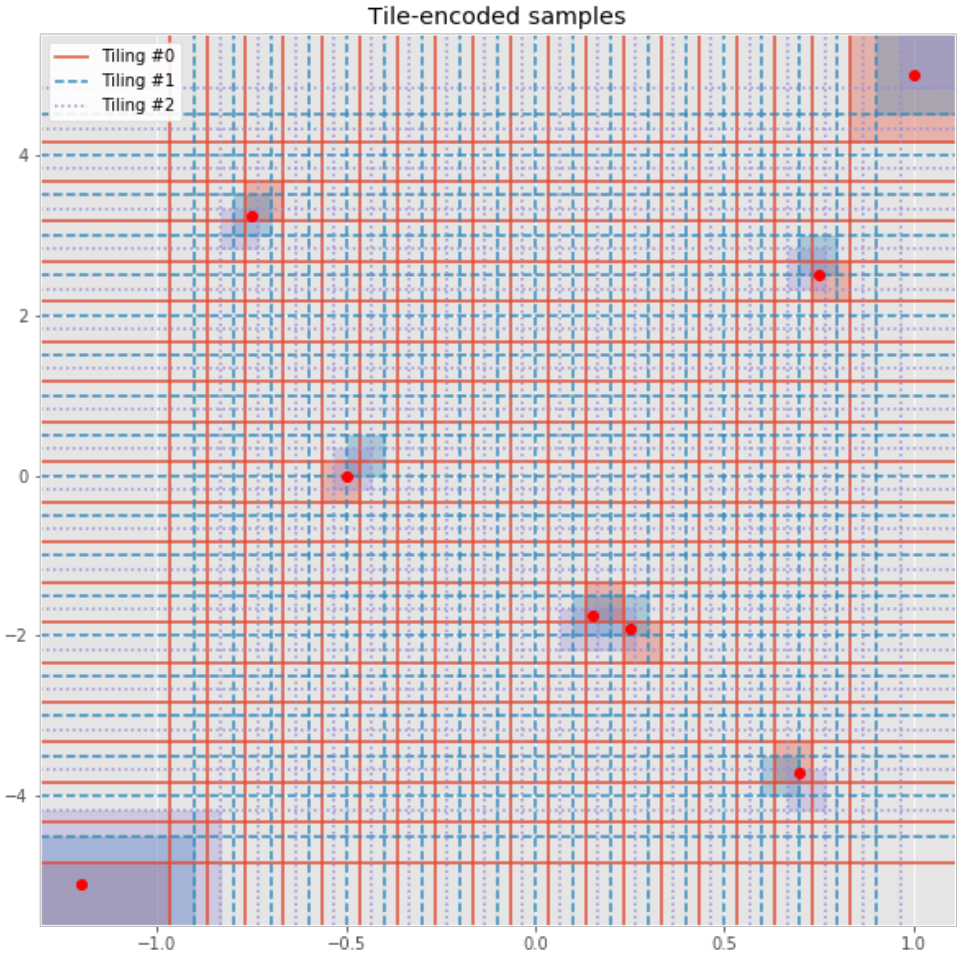
```
State space: Box(2,)
- low: [-1.2  -0.07]
- high: [0.6  0.07]
State space samples:
[[-0.28   0.005]
 [ 0.294  0.056]
 [-1.066  0.039]
 [ 0.401 -0.028]
 [-1.03  -0.027]
 [-1.026 -0.057]
 [-0.179  0.007]
 [-0.443  0.053]
 [-1.037  0.025]
 [ 0.035 -0.059]]
Action space: Discrete(3)
Action space samples:
[2 2 1 0 2 0 0 2 1 1]
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
```




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

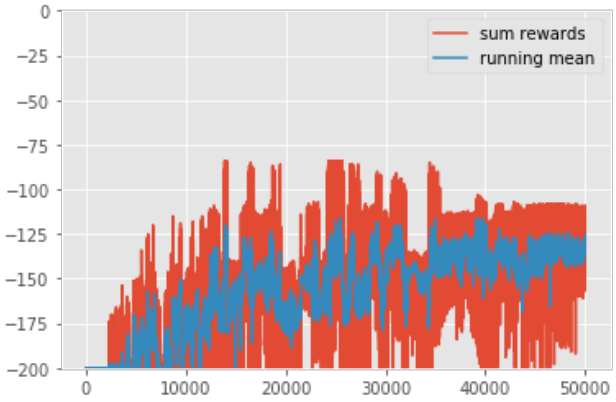

Tilings
Tiling #0
Tiling #1
Tiling #2
4
2
0
−2
−4
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00



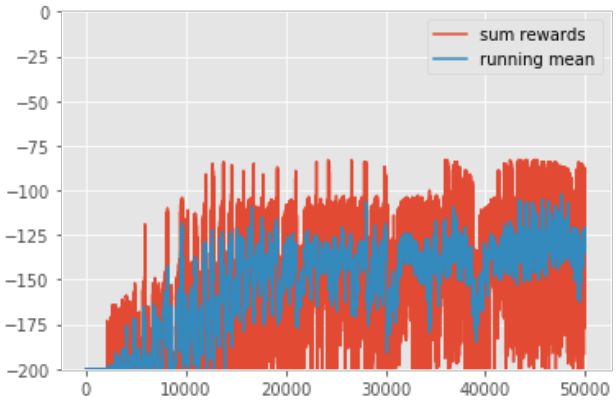
Results show both discretized and disretized with tiling agents able to solve mountain car, though not quite reliably by the solve definition of -110 over 100 trials. The tiled approach seems to have a smoother, more stable, and overall higher reward profile.

In [16]:
```
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
```



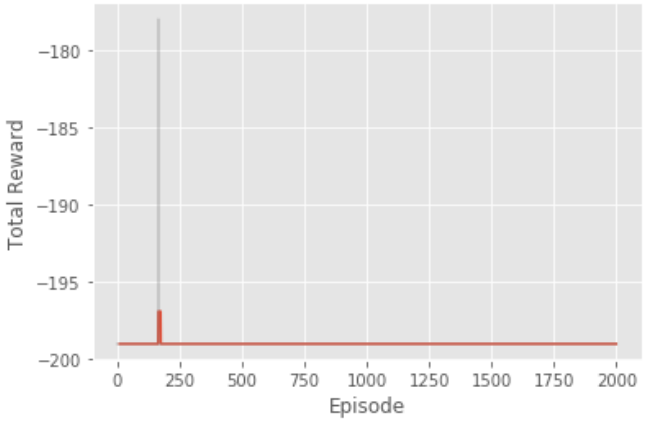

```
Tiled Discretization
```



Deep Q Network

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

It was not able to successfully solve the environment.



DDPG

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

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

Activation Function

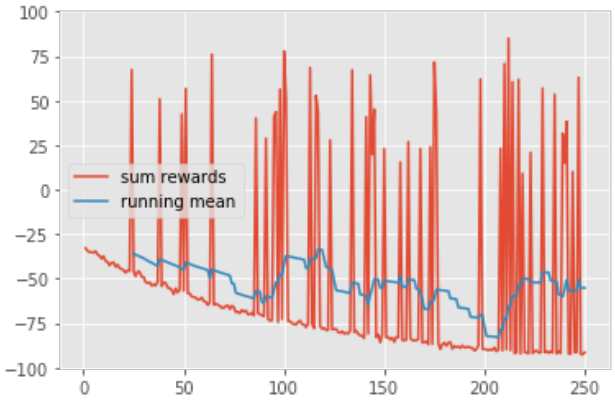
" In contrast to ReLUs, ELUs have negativevalues which allows them to push mean unit activations closer to zero like batchnormalization but with lower computational complexity" [5]

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 car using RELU activation function was changed to use ELU.

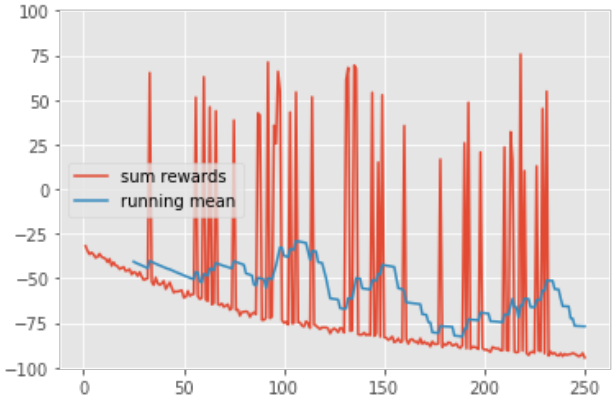
The network normally solves this environment between 50 an 100 episodes. However, with ELU, the network was unable to solve even after 250 episodes.

In [17]:
```
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
```

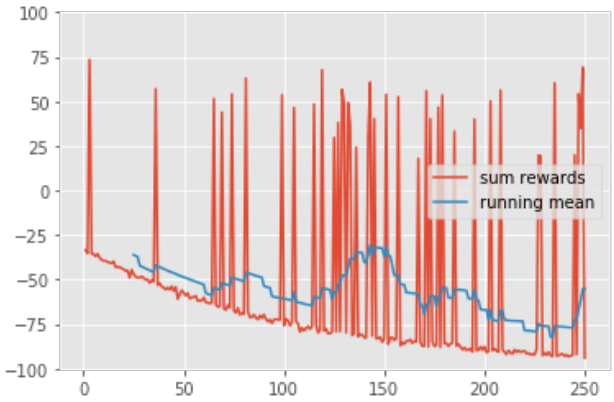


Initializers

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 [18]:
```
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
```



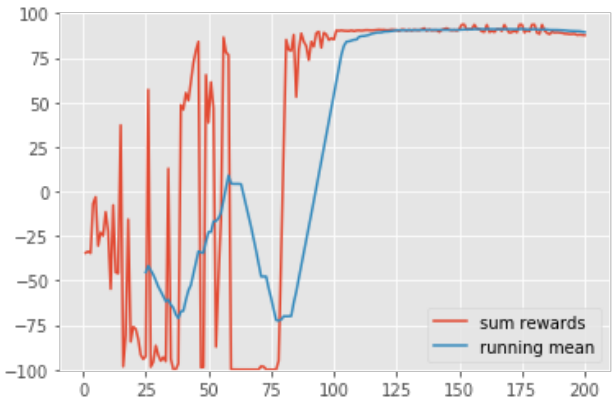
Network Size

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

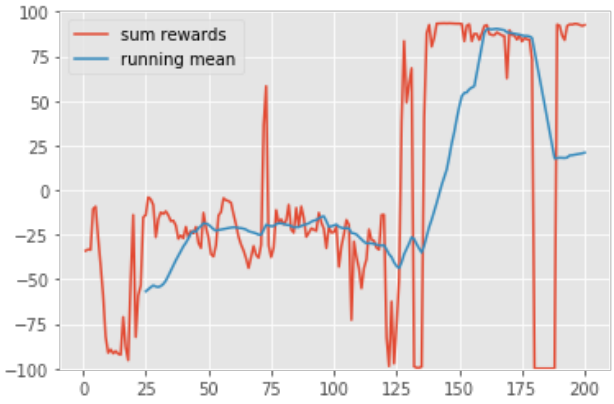
They all seem to be able to solve the environment. Generally there is an increase in training time for the larger networks to converge to a solution, though the actions by the network seems to have a more variety and sophisticated.

In [19]:
```
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)
```

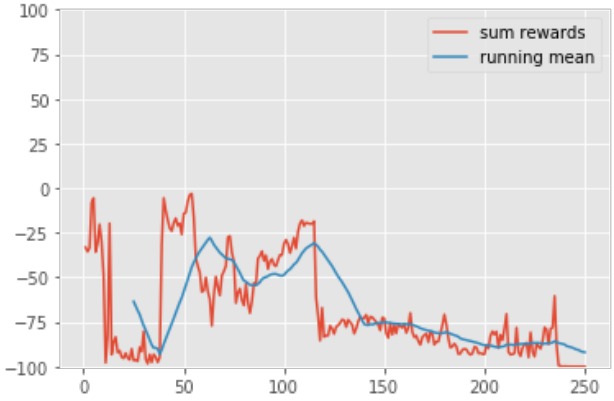
```
Copter
```




```
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.

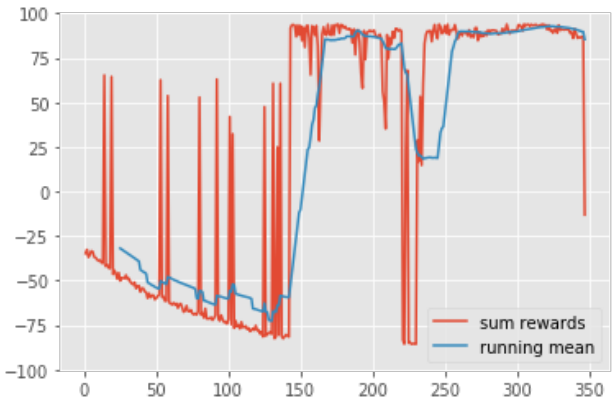
Network size when using dropout should also be considered, though was not adjusted for here.

"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]

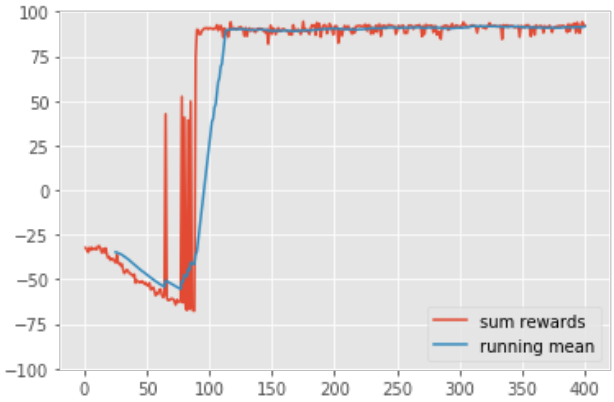
The following settings were used:

In [20]:
```
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)
```

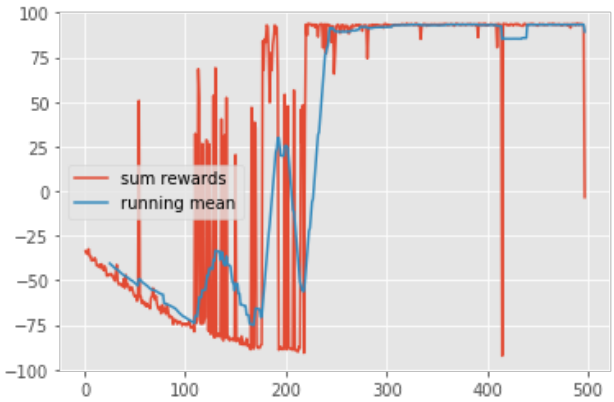
```
Dropout 0.0
```



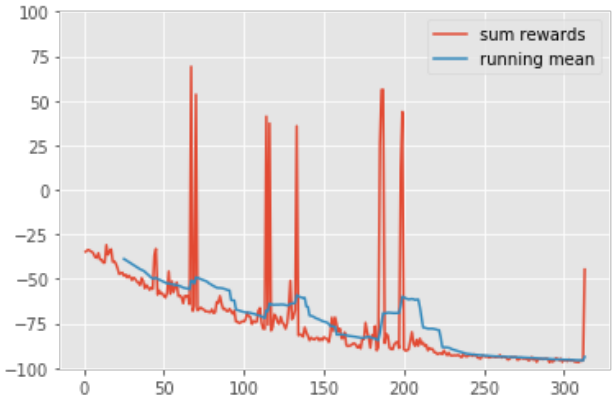

```
Dropout 0.2 (note QuadcoperBig network instead of Quadcopter Max)
```



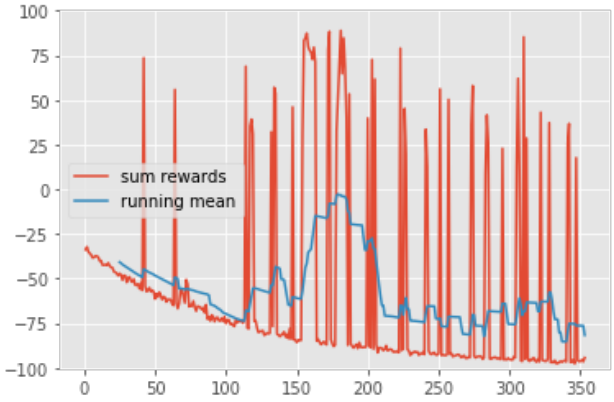

```
Dropout 0.3
```




```
Dropout 0.5
```



Dropout 0.7



Regularizers

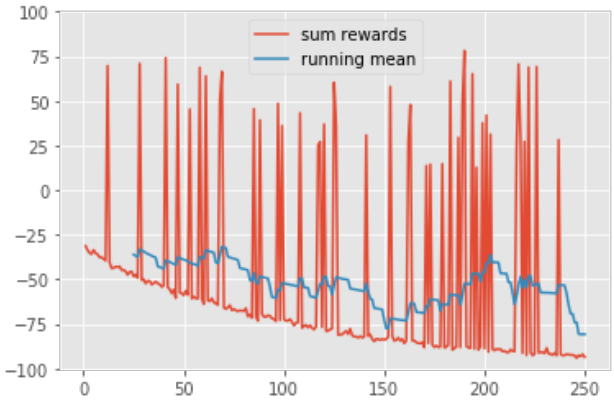
Regularizers are another common technique in superised learning.

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.

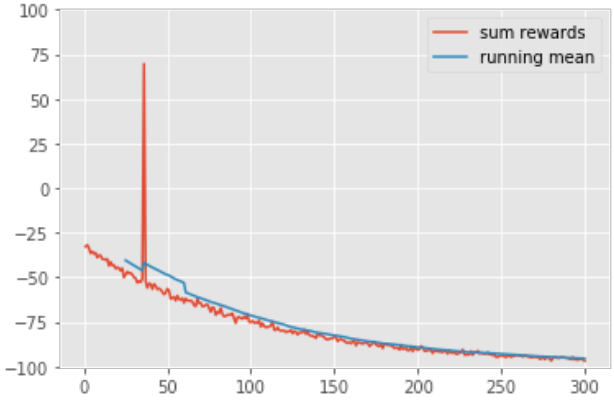
In [21]:
```
from visuals import plot_score_from_file

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

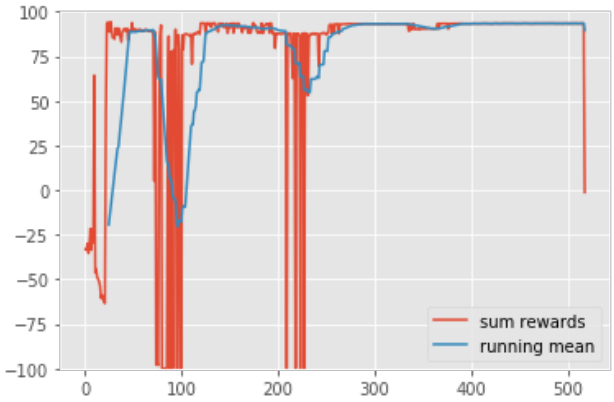
For the DDPG agent the Adam optimiser was used.

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 [22]:
```
print("ams grad")
dir = "results/MtClimberContinuous/amsgrad/"
plot_score_from_file(dir + "20190630112548MountainCarContinuous-v0_train.txt",
-100, 100, 1)
```

```
ams grad
```

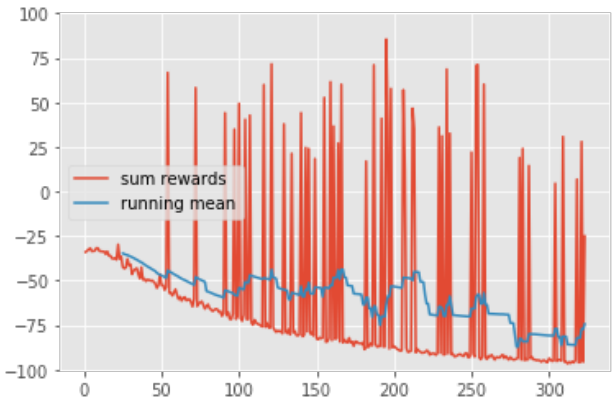


Learning Rate

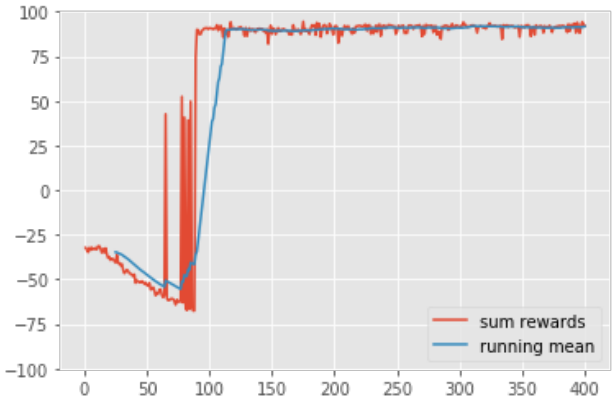
The learning rate is one of the most fundamental and difficult parts to get right in any network. For mountain car environment, a learning rate of 0.0001 seems to produce reasonable results.

In [23]:
```
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)
```

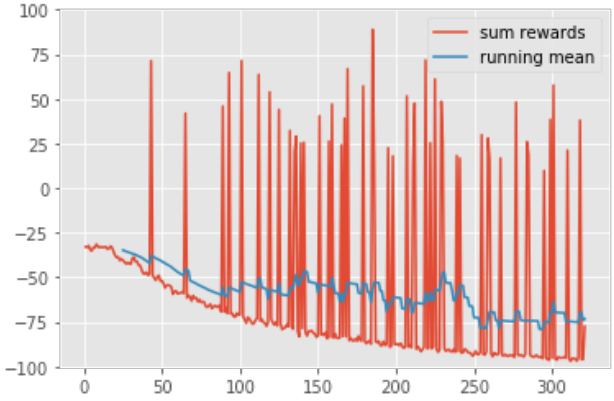
```
Learning Rate 0.001
```




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




```
Learning Rate 0.00001
```



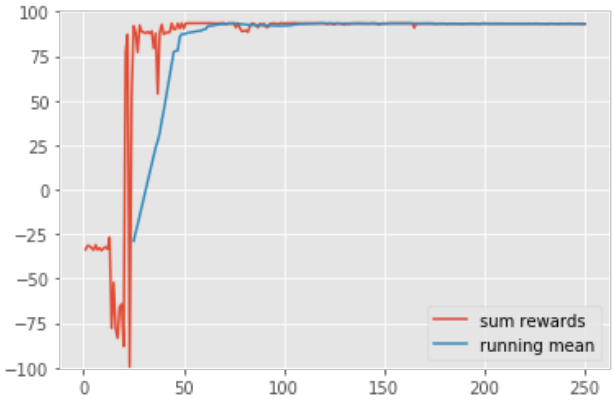
Action Repeat

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.

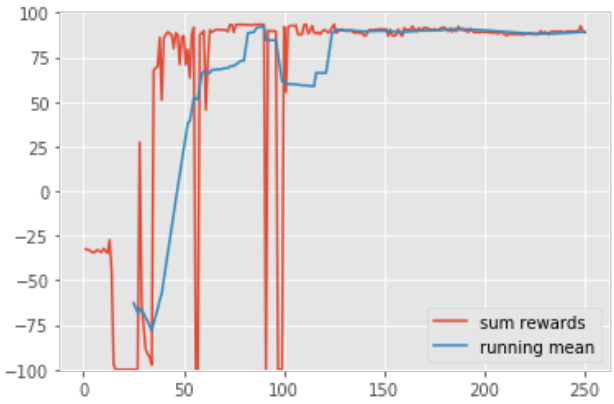
It would probably be more beneficial to just feed the agents previous state as a network input.

In [24]:
```
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).

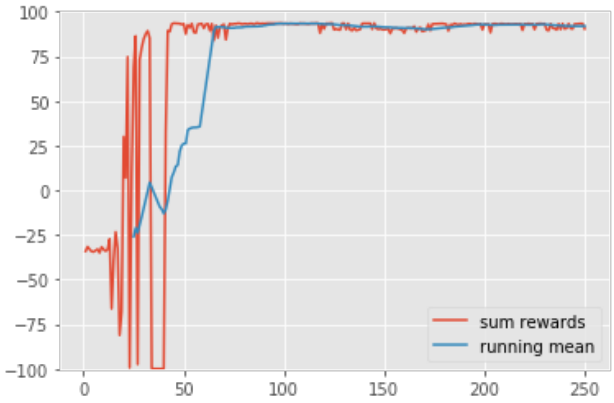
Pure exploitation seems to be important to allow the agent to get the precision on its actions, as correlated to its network weights, to continue the learning process.

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 [25]:
```
def sample_memory_full_enough(self):
    return self.batch_size * 100 * self.action_size * self.action_size
```

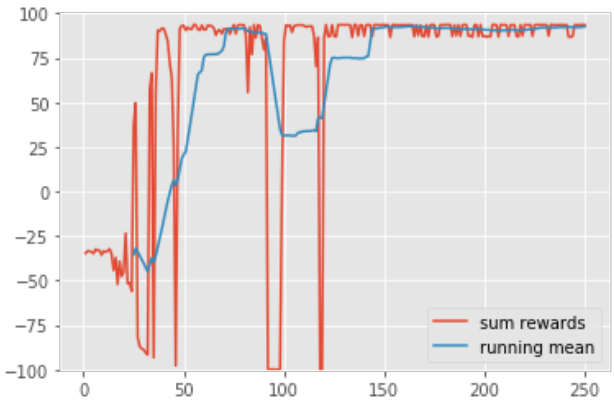
In [26]:
```
from visuals import plot_score_from_file

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

```
no exploration
```

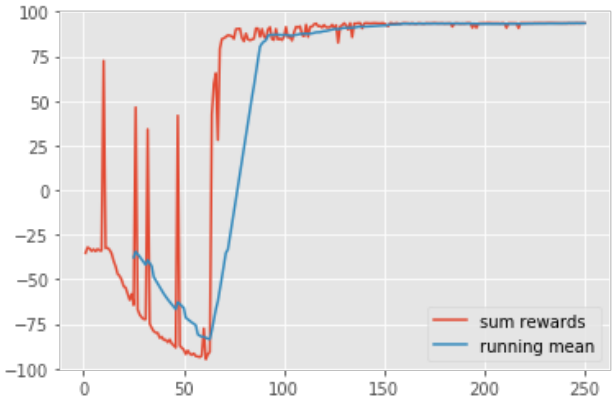


In [27]:
```
# uncorrelated exploration - 0.9 (0.1 decay per episode)
print("0.1 uncorrelated exploration decay")
dir = "results/MtClimberContinuous/exploration/uncorrelated/0.9/"
plot_score_from_file(dir + "20190628200353MountainCarContinuous-v0_train.txt",
-100, 100, 1)

# 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
```



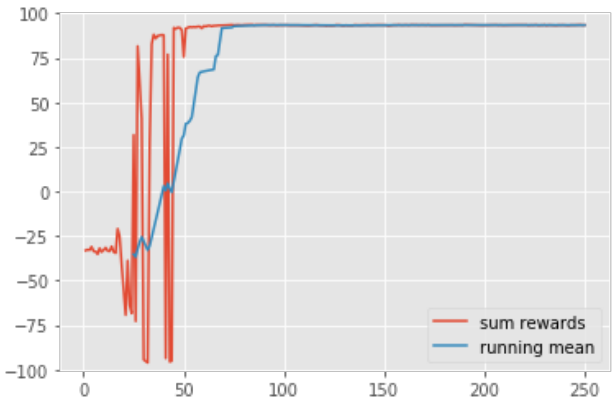

```
0.05 uncorrelated exploration decay
```



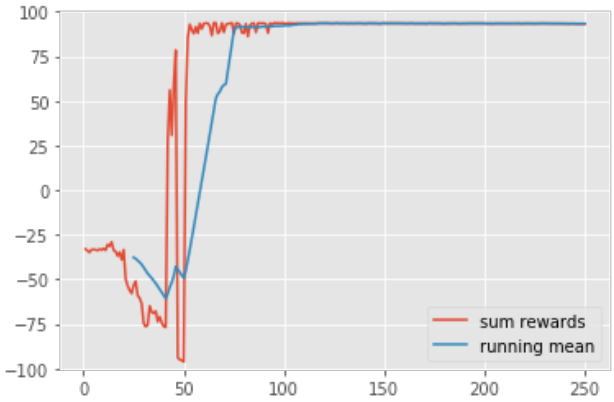

```
0.01 uncorrelated exploration decay
```

In [28]:
```
# 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)
```

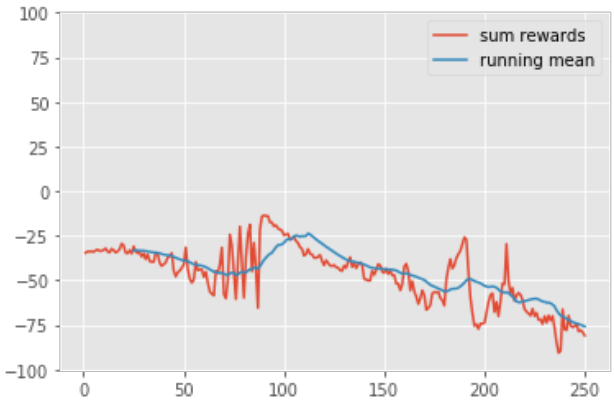
```
0.1 correlated exploration decay
```




```
0.05 correlated exploration decay
```

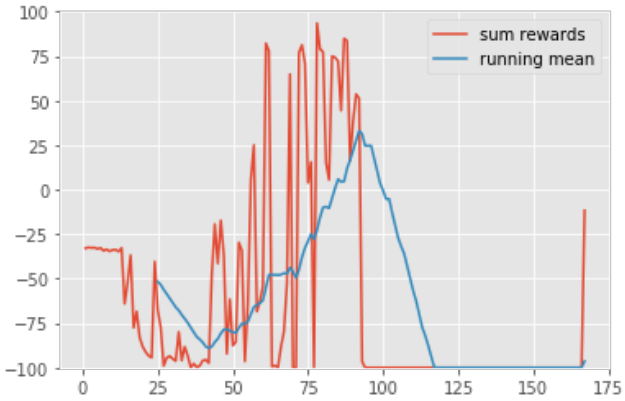



```
0.01 correlated exploration decay
```

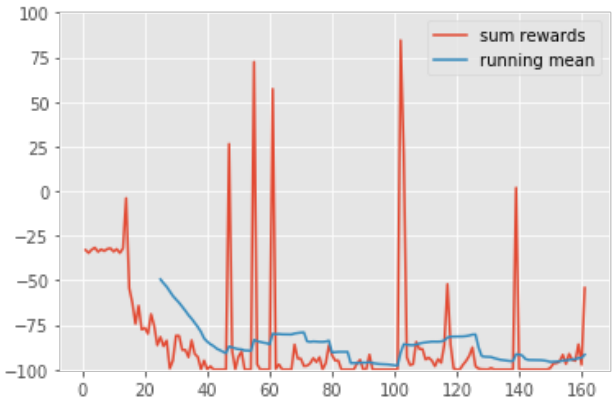


In [29]:
```
# parameter noise on the network - std dev 0.01
print("paramnoise exploration decay - 0.01")
dir = "results/MtClimberContinuous/exploration/paramnoise/0.01/"
plot_score_from_file(dir + "20190628212025MountainCarContinuous-v0_train.txt",
-100, 100, 1)

# 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

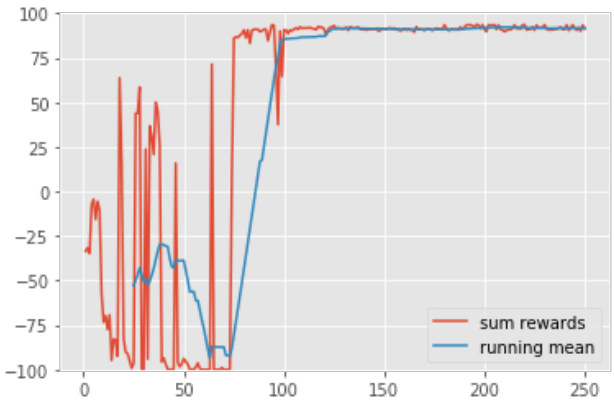
The idea with random replay is to de-couple the actions from the learning, and is widely used in both DQN and DDPG. Instead of learning from the current state, prior state, and action, which tend to be highly correlated from one step to the next, the agent instead learns randomly from memory. This allows 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.

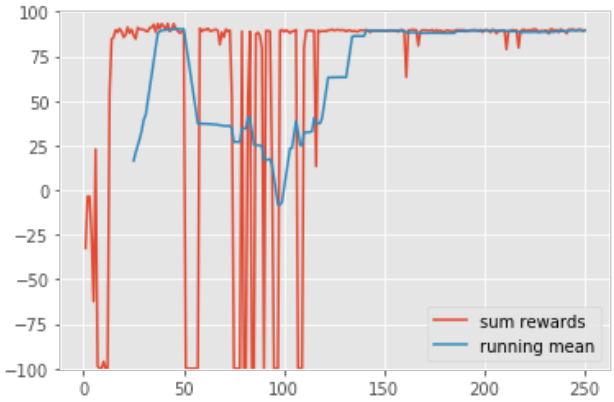
Learning rate appeared coupled to the batch size. While larger batch sizes seem to make the agent learn faster, it also appeared that if the batch size got too large, the learning rate should be decrease to compensate for the increased learning that happens each episode as a result of the larger batch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

In [30]:
```
# 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)
```

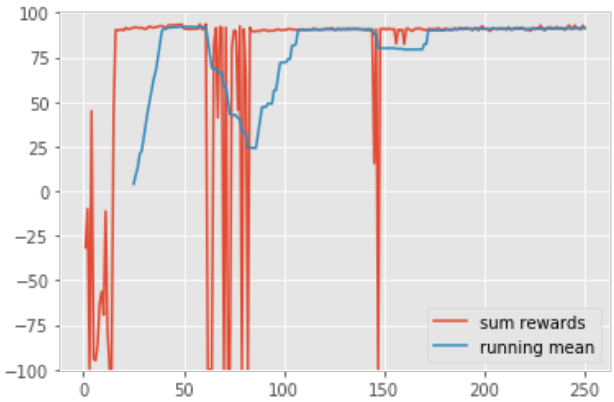
```
batch size 32
```



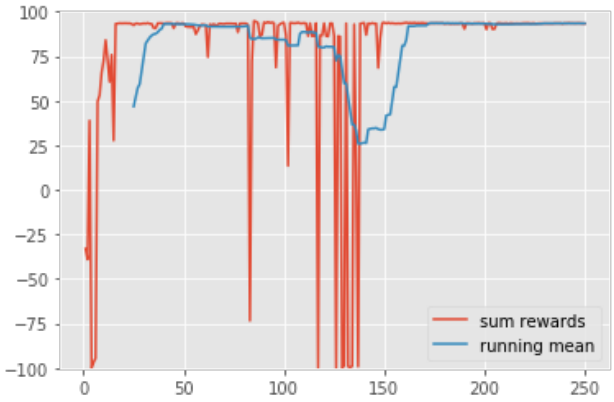

```
batch size 128
```



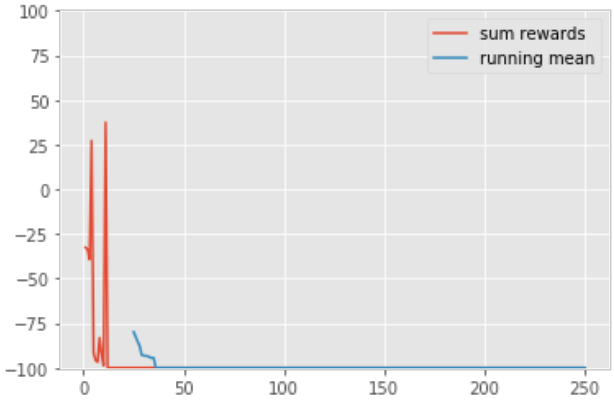

```
batch size 256
```




```
batch size 512
```




```
batch size 1024
```

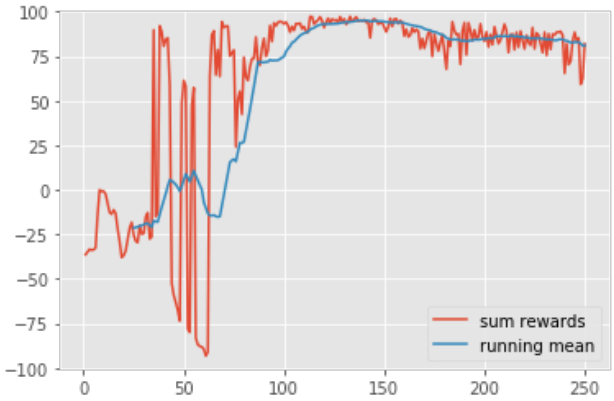


Batch Normalization

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 [31]:
```
# 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.

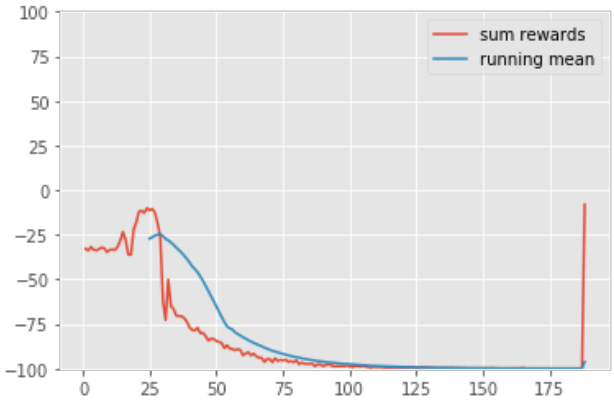
Unfortunately this approach did not seem to really help the learning process, and was hence abandoned.

Soft Update

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

In [32]:
```
print("no soft update")
dir = "results/MtClimberContinuous/noSoftUpdate/"
plot_score_from_file(dir + "20190629182336MountainCarContinuous-v0_train.txt",
-100, 100, 1)
```

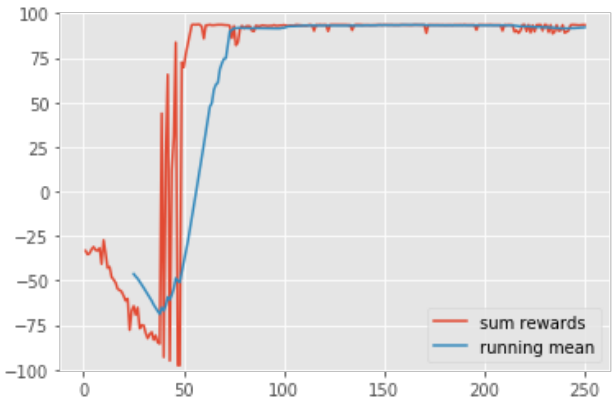
```
no soft update
```



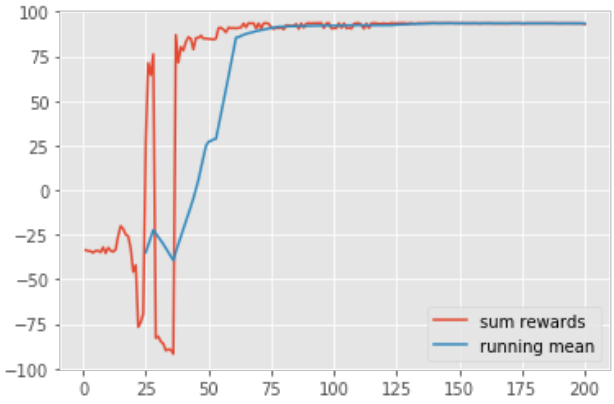
Below are results for different values of the soft update tau parameter.

In [33]:
```
# 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)
```

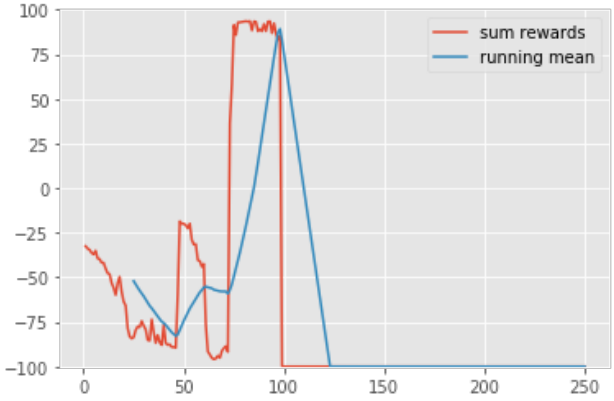
```
tau 0.1
```




```
tau 0.01
```




```
tau 0.001
```



Justification

Lessons Learned

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.

Comparing different agents

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.

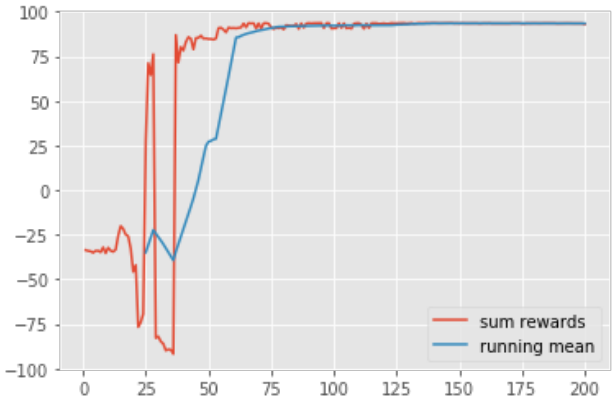
Final Agent Settings

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

In [34]:
```
print("final")
dir = "results/MtClimberContinuous/final/"
plot_score_from_file(dir + "20190629195726MountainCarContinuous-v0_train.txt",
-100, 100, 1)
```

```
final
```



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."

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 [35]:
```
print("lunar lander with simple discretized Q learning agent")
dir = "results/lunarlander/"
plot_score_from_file(dir + "20190629195846LunarLander-v2_train.txt", -500, 100,
1)
```

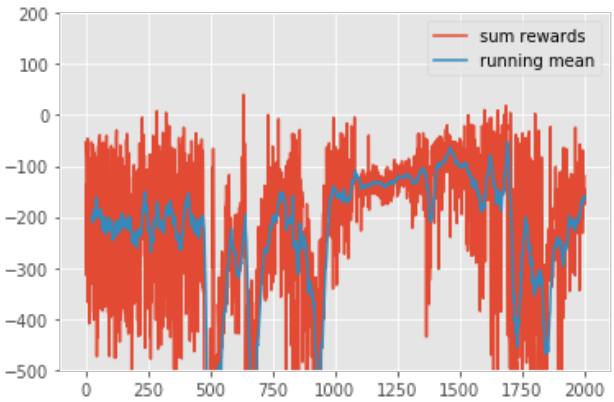
```
lunar lander with simple discretized Q learning agent
```

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 [36]:
```
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

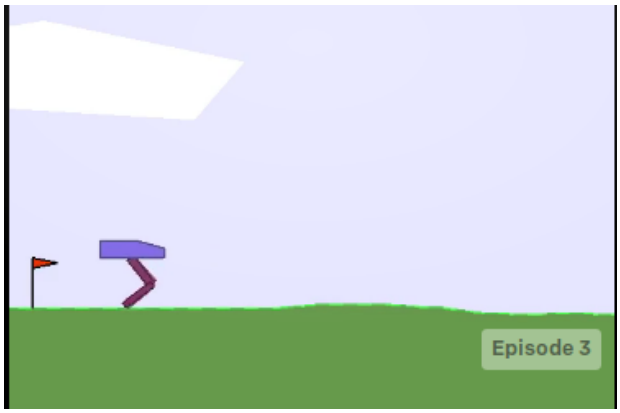
LR 0.0003

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."

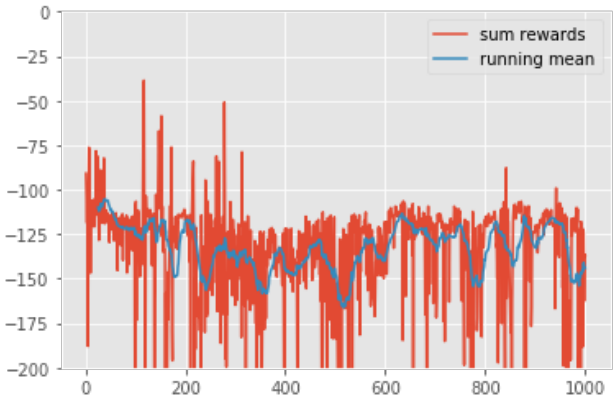


DDPG

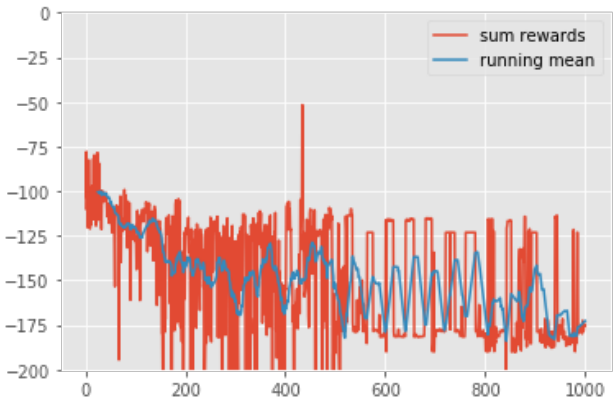
The agent was able to demonstate walking, however was not able to solve the environment.

In [37]:
```
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
```



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 [38]:
```
"""
# 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 [39]:
```
"""
# 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 CNN stride sizes taken from the DDQN with Dropout OpenAI leaderboard implementation [14].

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 (see code example below).

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 [40]:
```
import gym
import numpy as np

"""
# Create an environment and set random seed
"""
envName = 'CarRacing-v0'      #  image input, actions [steer, gas, brake]
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
"""
agent = 0
selectedAgent = 3

# Create DDPG network agent
obsSpace = env.observation_space.shape
print("env.observation_space: ", obsSpace)
from agents.DDPG import DDPG
agent = DDPG(env, "imageStateContinuousAction")  # continousStateAction imageSt
ateContinuousAction

"""
# run the simulation
"""
import interact as sim
num_episodes=2
sim.interact(agent, env, num_episodes, mode='train', file_output=file_output_tr
ain, renderSkip=100)

"""
# Exit Environment
"""
env.close()
del agent
```

```
Track generation: 1156..1449 -> 293-tiles track
----------------------------------------------------------
New Experiment, training output file name:  20190705135744CarRacing-v0_train.tx
t
env.observation_space:  (96, 96, 3)

Using TensorFlow backend.
```

```
************************************
************************************
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.]
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. 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_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
_________________________________________________________________
```

Conclusion

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 Car (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.

A lot of this can probably be attibuted to the increase in complexity in the environments in which the agent was attempted to be transferred into. The Mountain Car environment only had a single continuous action, while the Lunar Lander had two, and the Walker had four, so the range of possible action combinations is much larger. The agent might perform better it it had started in a more complex environment then tried a simplier one, however the more complex enviroments also take far longer to train, which makes agent experimentation more time consuming.

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 Car, 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

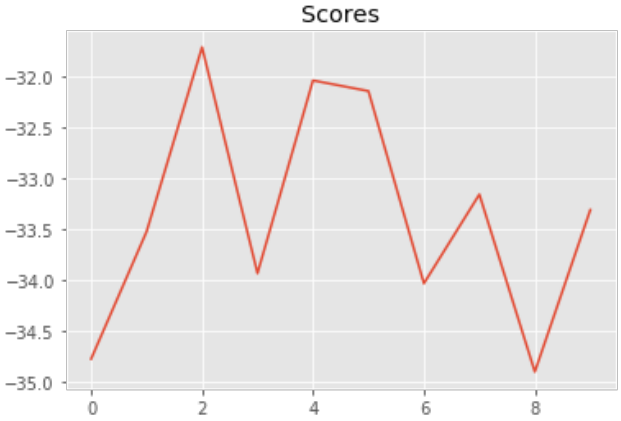
- L2 regularization. This appears to make the agents actions too in-precise to properly learn
- ELU activation function instead of RELU. Did not appear to provide any benefit and made performance worse.
- Learning Frequency. Didn't appear to add any value.
- Large amount of exploration (1% decay). Generally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

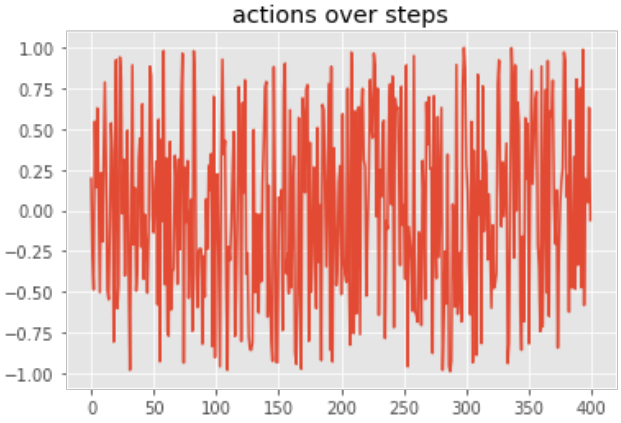
Free-Form Visualization (Mountain Car)

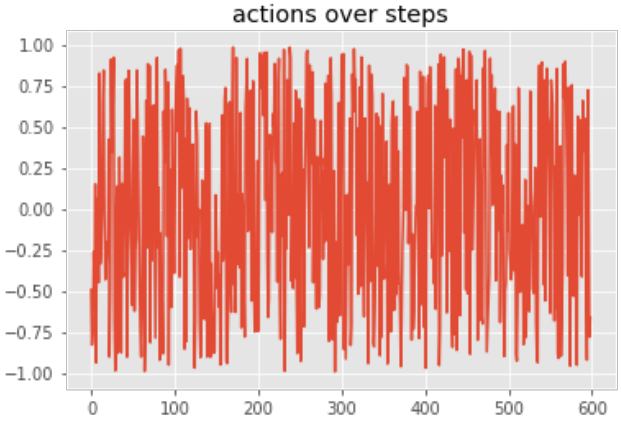
The following section shows learning progression for the final version of the mountain car DDPG network. This network was able to solve the environment quite quickly. Progression of learning can be seen in action plots for select episodes, as well as in the progression of rewards from the environment.

In [41]:
```
import gym
import numpy as np

"""
# Create an environment and set random seed
"""
envName = 'MountainCarContinuous-v0' # continuous only
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
"""
# Create DDPG network agent
obsSpace = env.observation_space.shape
print("env.observation_space: ", obsSpace)
from agents.DDPG import DDPG
agent = DDPG(env, "continousStateAction")  # continousStateAction imageStateCon
tinuousAction

"""
# run the simulation
"""
import interact as sim
num_episodes=100
sim.interact(agent, env, num_episodes, mode='train', file_output=file_output_tr
ain, renderSkip=10000)

"""
# Run in test mode and analyze scores obtained
"""
print("[TEST] Training Done, now running tests...")
test_scores = sim.interact(agent, env, num_episodes=3, mode='test', file_outpu
t=file_output_test)
plot_score_from_file(file_output_test, -300, 300, 1)

"""
# Watch Agent
"""
state = env.reset()
score = 0
for t in range(5000):
    # get action from agent
    action = agent.act(state, mode='test')

    # show environment and step it forward
    env.render()
    state, reward, done, _ = env.step(action)
    score += reward
```

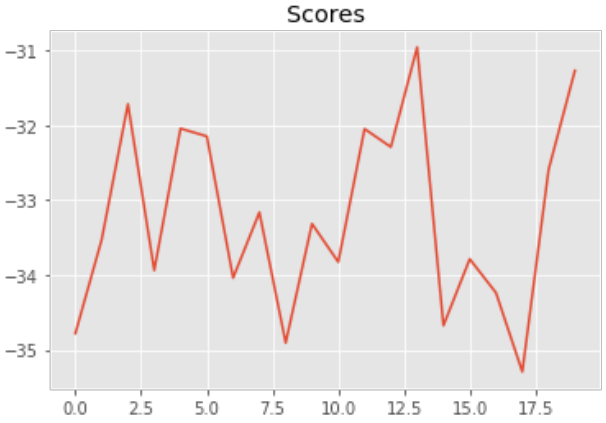
```
------------------------------------------------------------
New Experiment, training output file name:  20190705135811MountainCarContinuous
-v0_train.txt
env.observation_space:  (2,)
**********************************
**********************************
Initializing DDPG Agent
        Environment:  <TimeLimit<Continuous_MountainCarEnv<MountainCarContinuou
s-v0>>>
env.action_space.shape (1,)
env.action_space.low [-1.]
env.action_space.high [1.]
*** 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, 4)                 0
_________________________________________________________________
dense_9 (Dense)              (None, 128)               640
_________________________________________________________________
dropout_19 (Dropout)         (None, 128)               0
_________________________________________________________________
dense_10 (Dense)             (None, 256)               33024
_________________________________________________________________
dropout_20 (Dropout)         (None, 256)               0
_________________________________________________________________
dense_11 (Dense)             (None, 256)               65792
_________________________________________________________________
dropout_21 (Dropout)         (None, 256)               0
_________________________________________________________________
dense_12 (Dense)             (None, 128)               32896
_________________________________________________________________
dropout_22 (Dropout)         (None, 128)               0
_________________________________________________________________
raw_actions (Dense)          (None, 1)                 129
=================================================================
Total params: 132,481
Trainable params: 132,481
Non-trainable params: 0
_________________________________________________________________
[ CRITIC MODEL SUMMARY ]
__________________________________________________________________________________
_____________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================
==================
states (InputLayer)             (None, 4)            0
__________________________________________________________________________________
_____________________
actions (InputLayer)            (None, 1)            0
__________________________________________________________________________________
_____________________
dense_17 (Dense)                (None, 128)          640         states[0][0]
__________________________________________________________________________________
_____________________
dense_19 (Dense)                (None, 128)          256         actions[0][0]
```

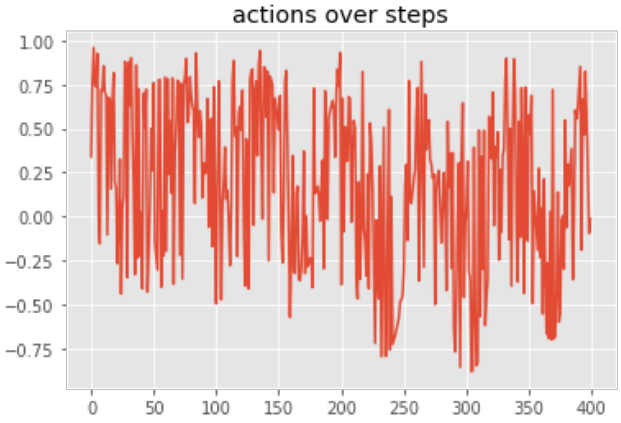

Scores
−32.0
−32.5
−33.0
−33.5
−34.0
−34.5
−35.0
0
2
4
6
8

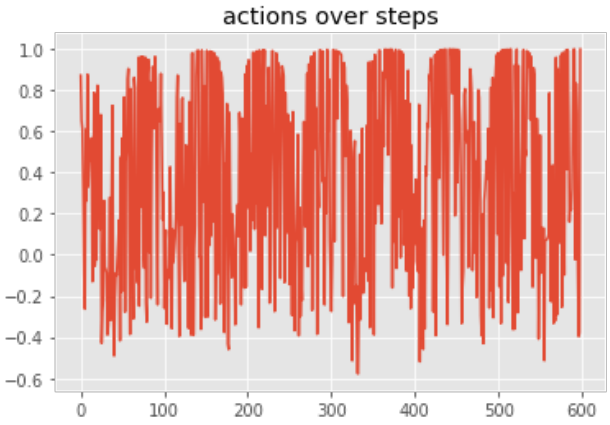

actions over steps
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
50
100
150
200
250
300
350
400


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

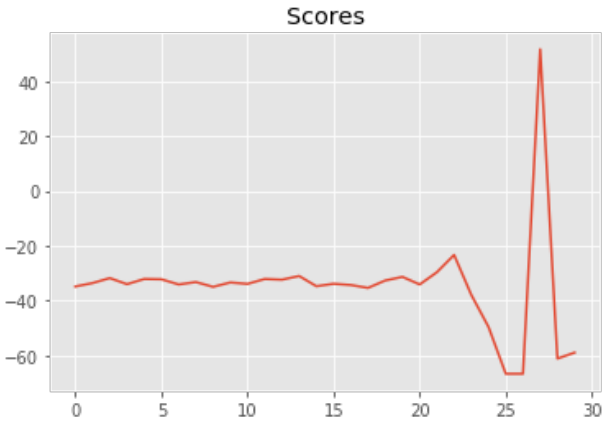
```
Episode =   10 (duration of 999 steps); Reward = -33.317 (best = -31.719, in ep
i 3)
        Episode training time:  0.755040168762207
        resetting episode... next explore_p:  1.0
Garbage collector: collected 8096 objects.
        step:  0 , action: [0.246]
Episode =   11 (duration of 999 steps); Reward = -33.828 (best = -31.719, in ep
i 3)
        Episode training time:  0.18204116821289062
        resetting episode... next explore_p:  1.0
Garbage collector: collected 0 objects.
        step:  0 , action: [-0.716]
Episode =   12 (duration of 999 steps); Reward = -32.050 (best = -31.719, in ep
i 3)
        Episode training time:  0.18336105346679688
        resetting episode... next explore_p:  1.0
Garbage collector: collected 0 objects.
        step:  0 , action: [0.927]
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.
Episode =   13 (duration of 999 steps); Reward = -32.292 (best = -31.719, in ep
i 3)
        Episode training time:  7.618906736373901
        resetting episode... next explore_p:  0.95
Garbage collector: collected 0 objects.
        step:  0 , action: [0.739]
Episode =   14 (duration of 999 steps); Reward = -30.959 (best = -30.959, in ep
i 14)
        Episode training time:  29.003148078918457
        resetting episode... next explore_p:  0.9025
Garbage collector: collected 0 objects.
        step:  0 , action: [-0.691]
Episode =   15 (duration of 999 steps); Reward = -34.677 (best = -30.959, in ep
i 14)
        Episode training time:  29.213695287704468
        resetting episode... next explore_p:  0.8573749999999999
Garbage collector: collected 0 objects.
        step:  0 , action: [-0.541]
Episode =   16 (duration of 999 steps); Reward = -33.788 (best = -30.959, in ep
i 14)
        Episode training time:  28.93559718132019
        resetting episode... next explore_p:  0.8145062499999999
Garbage collector: collected 0 objects.
        step:  0 , action: [-0.505]
Episode =   17 (duration of 999 steps); Reward = -34.240 (best = -30.959, in ep
i 14)
        Episode training time:  28.700440168380737
        resetting episode... next explore_p:  0.7737809374999999
Garbage collector: collected 0 objects.
        step:  0 , action: [0.999]
Episode =   18 (duration of 999 steps); Reward = -35.294 (best = -30.959, in ep
i 14)
        Episode training time:  28.792741775512695
        resetting episode... next explore_p:  0.7350918906249998
Garbage collector: collected 0 objects.
        step:  0 , action: [0.461]
Episode =   19 (duration of 999 steps); Reward = -32.595 (best = -30.959, in ep
i 14)
        Episode training time:  28.116634845733643
        resetting episode... next explore_p:  0.6983372960937497
```

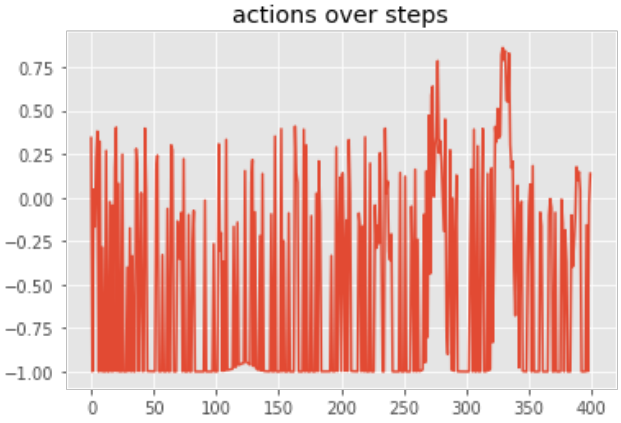

Scores
−31
−32
−33
−34
−35
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5


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


actions over steps
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
0
100
200
300
400
500
600

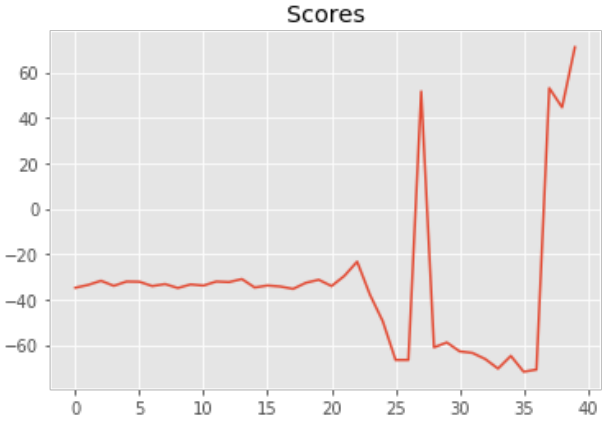
```
Episode =   20 (duration of 999 steps); Reward = -31.272 (best = -30.959, in ep
i 14)
        Episode training time:  29.671430110931396
        resetting episode... next explore_p:  0.6634204312890623
Garbage collector: collected 8096 objects.
        step:  0 , action: [0.983]
Episode =   21 (duration of 999 steps); Reward = -34.041 (best = -30.959, in ep
i 14)
        Episode training time:  28.70863914489746
        resetting episode... next explore_p:  0.6302494097246091
Garbage collector: collected 0 objects.
        step:  0 , action: [0.45]
Episode =   22 (duration of 999 steps); Reward = -29.583 (best = -29.583, in ep
i 22)
        Episode training time:  29.02830672264099
        resetting episode... next explore_p:  0.5987369392383786
Garbage collector: collected 0 objects.
        step:  0 , action: [-0.28]
Episode =   23 (duration of 999 steps); Reward = -23.279 (best = -23.279, in ep
i 23)
        Episode training time:  29.006455659866333
        resetting episode... next explore_p:  0.5688000922764596
Garbage collector: collected 0 objects.
        step:  0 , action: [0.556]
Episode =   24 (duration of 999 steps); Reward = -37.854 (best = -23.279, in ep
i 23)
        Episode training time:  29.106515169143677
        resetting episode... next explore_p:  0.5403600876626365
Garbage collector: collected 0 objects.
        step:  0 , action: [0.921]
Episode =   25 (duration of 999 steps); Reward = -49.530 (best = -23.279, in ep
i 23)
        Episode training time:  28.884387493133545
        resetting episode... next explore_p:  0.5133420832795047
Garbage collector: collected 0 objects.
        step:  0 , action: [0.034]
Episode =   26 (duration of 999 steps); Reward = -66.572 (best = -23.279, in ep
i 23)
        Episode training time:  29.16269850730896
        resetting episode... next explore_p:  0.48767497911552943
Garbage collector: collected 0 objects.
        step:  0 , action: [0.054]
Episode =   27 (duration of 999 steps); Reward = -66.570 (best = -23.279, in ep
i 23)
        Episode training time:  29.020633697509766
        resetting episode... next explore_p:  0.46329123015975293
Garbage collector: collected 0 objects.
        step:  0 , action: [1.]
Episode =   28 (duration of 684 steps); Reward =  51.671 (best =  51.671, in ep
i 28)
        Episode training time:  19.92894744873047
        resetting episode... next explore_p:  0.44012666865176525
Garbage collector: collected 0 objects.
        step:  0 , action: [1.]
Episode =   29 (duration of 999 steps); Reward = -61.047 (best =  51.671, in ep
i 28)
        Episode training time:  28.99922251701355
        resetting episode... next explore_p:  0.41812033521917696
Garbage collector: collected 0 objects.
        step:  0 , action: [0.343]
```


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


actions over steps
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

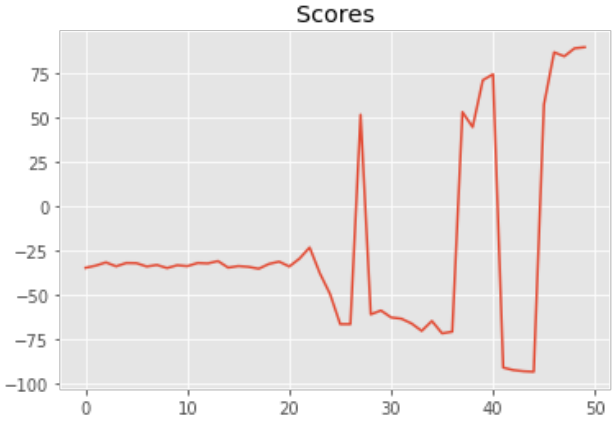

actions over steps
0.4
0.2
0.0
−0.2
−0.4
−0.6
−0.8
−1.0
0
100
200
300
400
500
600

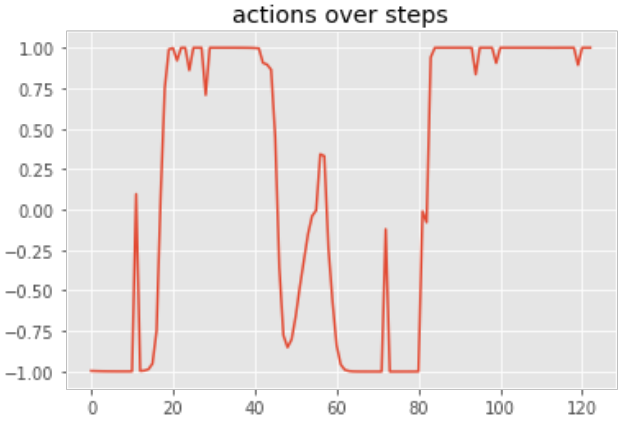
```
Episode =   30 (duration of 999 steps); Reward = -58.865 (best =  51.671, in ep
i 28)
        Episode training time:  29.43665623664856
        resetting episode... next explore_p:  0.3972143184582181
Garbage collector: collected 8031 objects.
        step:  0 , action: [-0.232]
Episode =   31 (duration of 999 steps); Reward = -62.788 (best =  51.671, in ep
i 28)
        Episode training time:  28.723541259765625
        resetting episode... next explore_p:  0.37735360253530714
Garbage collector: collected 0 objects.
        step:  0 , action: [-0.335]
Episode =   32 (duration of 999 steps); Reward = -63.486 (best =  51.671, in ep
i 28)
        Episode training time:  27.722180128097534
        resetting episode... next explore_p:  0.35848592240854177
Garbage collector: collected 0 objects.
        step:  0 , action: [-1.]
Episode =   33 (duration of 999 steps); Reward = -66.178 (best =  51.671, in ep
i 28)
        Episode training time:  28.701806783676147
        resetting episode... next explore_p:  0.34056162628811465
Garbage collector: collected 0 objects.
        step:  0 , action: [-1.]
Episode =   34 (duration of 999 steps); Reward = -70.389 (best =  51.671, in ep
i 28)
        Episode training time:  28.601864099502563
        resetting episode... next explore_p:  0.3235335449737089
Garbage collector: collected 0 objects.
        step:  0 , action: [-0.16]
Episode =   35 (duration of 999 steps); Reward = -64.789 (best =  51.671, in ep
i 28)
        Episode training time:  28.677013158798218
        resetting episode... next explore_p:  0.30735686772502346
Garbage collector: collected 0 objects.
        step:  0 , action: [-1.]
Episode =   36 (duration of 999 steps); Reward = -71.773 (best =  51.671, in ep
i 28)
        Episode training time:  28.641079902648926
        resetting episode... next explore_p:  0.2919890243387723
Garbage collector: collected 0 objects.
        step:  0 , action: [-1.]
Episode =   37 (duration of 999 steps); Reward = -70.814 (best =  51.671, in ep
i 28)
        Episode training time:  28.553797006607056
        resetting episode... next explore_p:  0.27738957312183365
Garbage collector: collected 0 objects.
        step:  0 , action: [-1.]
Episode =   38 (duration of 603 steps); Reward =  53.113 (best =  53.113, in ep
i 38)
        Episode training time:  17.090879678726196
        resetting episode... next explore_p:  0.263520094465742
Garbage collector: collected 0 objects.
        step:  0 , action: [0.883]
Episode =   39 (duration of 690 steps); Reward =  44.647 (best =  53.113, in ep
i 38)
        Episode training time:  19.730556964874268
        resetting episode... next explore_p:  0.25034408974245487
Garbage collector: collected 0 objects.
        step:  0 , action: [0.824]
```


Scores
60
40
20
0
−20
−40
−60
0
5
10
15
20
25
30
35
40

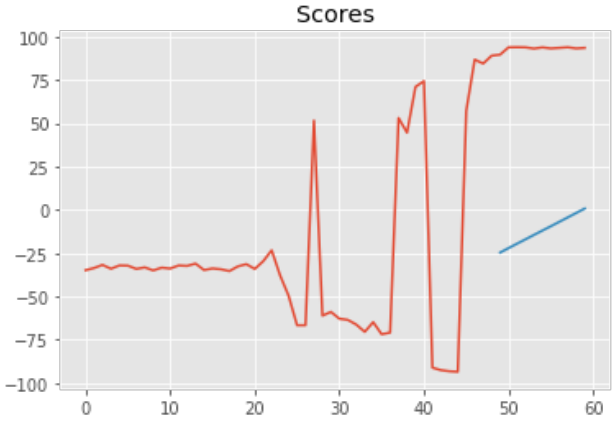

actions over steps
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
50
100
150
200
250
300
350

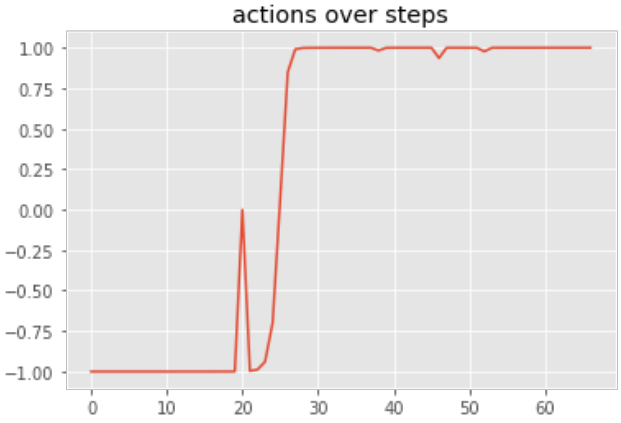
```
Episode =   40 (duration of 379 steps); Reward =  71.108 (best =  71.108, in ep
i 40)
        Episode training time:  11.397584676742554
        resetting episode... next explore_p:  0.2378268852553321
Garbage collector: collected 5579 objects.
        step:  0 , action: [0.907]
Episode =   41 (duration of 324 steps); Reward =  74.460 (best =  74.460, in ep
i 41)
        Episode training time:  9.29276180267334
        resetting episode... next explore_p:  0.2259355409925655
Garbage collector: collected 0 objects.
        step:  0 , action: [1.]
Episode =   42 (duration of 999 steps); Reward = -91.078 (best =  74.460, in ep
i 41)
        Episode training time:  28.634819507598877
        resetting episode... next explore_p:  0.2146387639429372
Garbage collector: collected 0 objects.
        step:  0 , action: [1.]
Episode =   43 (duration of 999 steps); Reward = -92.497 (best =  74.460, in ep
i 41)
        Episode training time:  28.77960515022278
        resetting episode... next explore_p:  0.20390682574579033
Garbage collector: collected 0 objects.
        step:  0 , action: [1.]
Episode =   44 (duration of 999 steps); Reward = -93.138 (best =  74.460, in ep
i 41)
        Episode training time:  28.47392725944519
        resetting episode... next explore_p:  0.1937114844585008
Garbage collector: collected 0 objects.
        step:  0 , action: [1.]
Episode =   45 (duration of 999 steps); Reward = -93.433 (best =  74.460, in ep
i 41)
        Episode training time:  28.518723487854004
        resetting episode... next explore_p:  0.18402591023557577
Garbage collector: collected 0 objects.
        step:  0 , action: [1.]
Episode =   46 (duration of 466 steps); Reward =  57.293 (best =  74.460, in ep
i 41)
        Episode training time:  13.519705295562744
        resetting episode... next explore_p:  0.17482461472379698
Garbage collector: collected 0 objects.
        step:  0 , action: [-0.084]
Episode =   47 (duration of 166 steps); Reward =  86.838 (best =  86.838, in ep
i 47)
        Episode training time:  4.850905895233154
        resetting episode... next explore_p:  0.16608338398760714
Garbage collector: collected 0 objects.
        step:  0 , action: [0.992]
Episode =   48 (duration of 182 steps); Reward =  84.560 (best =  86.838, in ep
i 47)
        Episode training time:  5.284491300582886
        resetting episode... next explore_p:  0.15777921478822676
Garbage collector: collected 0 objects.
        step:  0 , action: [-0.999]
Episode =   49 (duration of 122 steps); Reward =  89.095 (best =  89.095, in ep
i 49)
        Episode training time:  3.6622378826141357
        resetting episode... next explore_p:  0.14989025404881542
Garbage collector: collected 0 objects.
        step:  0 , action: [-0.996]
```


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

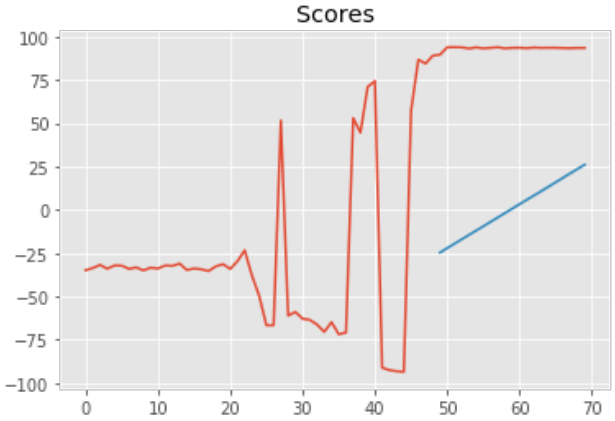

actions over steps
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
20
40
60
80
100
120

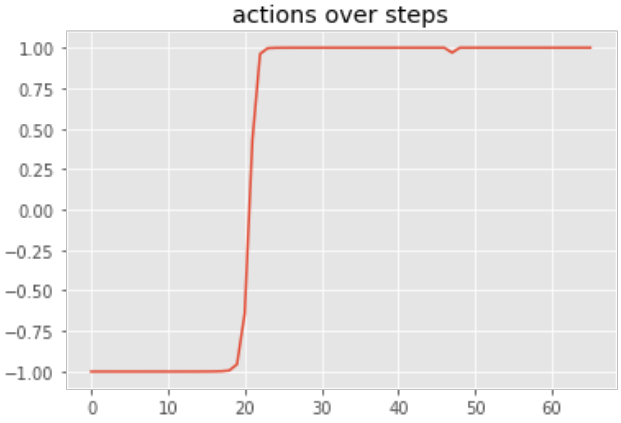
```
Episode =   50 (duration of 123 steps); Reward =  89.712 (best =  89.712, in ep
i 50)
        Episode training time:  3.9312186241149902
        resetting episode... next explore_p:  0.14239574134637464
Garbage collector: collected 5384 objects.
        step:  0 , action: [-0.999]
Episode =   51 (duration of 69 steps); Reward =  93.916 (best =  93.916, in epi
51)
        Episode training time:  2.1173481941223145
        resetting episode... next explore_p:  0.1352759542790559
Garbage collector: collected 0 objects.
        step:  0 , action: [-1.]
Episode =   52 (duration of 70 steps); Reward =  93.997 (best =  93.997, in epi
52)
        Episode training time:  2.1678466796875
        resetting episode... next explore_p:  0.1285121565651031
Garbage collector: collected 0 objects.
        step:  0 , action: [-1.]
Episode =   53 (duration of 68 steps); Reward =  93.921 (best =  93.997, in epi
52)
        Episode training time:  2.0086591243743896
        resetting episode... next explore_p:  0.12208654873684793
Garbage collector: collected 0 objects.
        step:  0 , action: [-1.]
Episode =   54 (duration of 76 steps); Reward =  93.221 (best =  93.997, in epi
52)
        Episode training time:  2.3112144470214844
        resetting episode... next explore_p:  0.11598222130000553
Garbage collector: collected 0 objects.
        step:  0 , action: [-0.999]
Episode =   55 (duration of 66 steps); Reward =  93.895 (best =  93.997, in epi
52)
        Episode training time:  2.0410687923431396
        resetting episode... next explore_p:  0.11018311023500525
Garbage collector: collected 0 objects.
        step:  0 , action: [-1.]
Episode =   56 (duration of 76 steps); Reward =  93.313 (best =  93.997, in epi
52)
        Episode training time:  2.358534574508667
        resetting episode... next explore_p:  0.10467395472325498
Garbage collector: collected 0 objects.
        step:  0 , action: [-1.]
Episode =   57 (duration of 68 steps); Reward =  93.629 (best =  93.997, in epi
52)
        Episode training time:  1.950880765914917
        resetting episode... next explore_p:  0.09944025698709223
Garbage collector: collected 0 objects.
        step:  0 , action: [-1.]
Episode =   58 (duration of 70 steps); Reward =  93.964 (best =  93.997, in epi
52)
        Episode training time:  2.159595489501953
        resetting episode... next explore_p:  0.09446824413773762
Garbage collector: collected 0 objects.
        step:  0 , action: [-1.]
Episode =   59 (duration of 75 steps); Reward =  93.308 (best =  93.997, in epi
52)
        Episode training time:  2.292975664138794
        resetting episode... next explore_p:  0.08974483193085074
Garbage collector: collected 0 objects.
        step:  0 , action: [-1.]
```


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

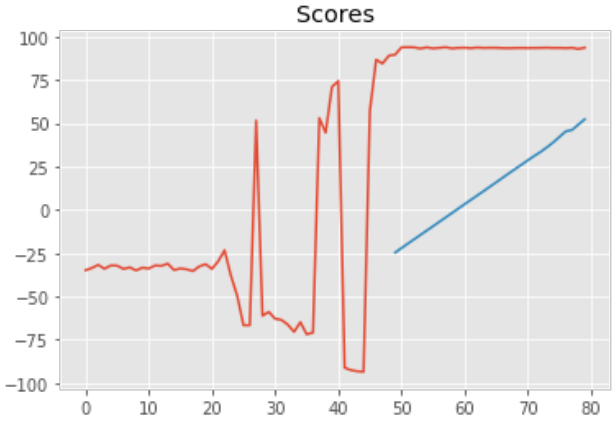

actions over steps
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
10
20
30
40
50
60

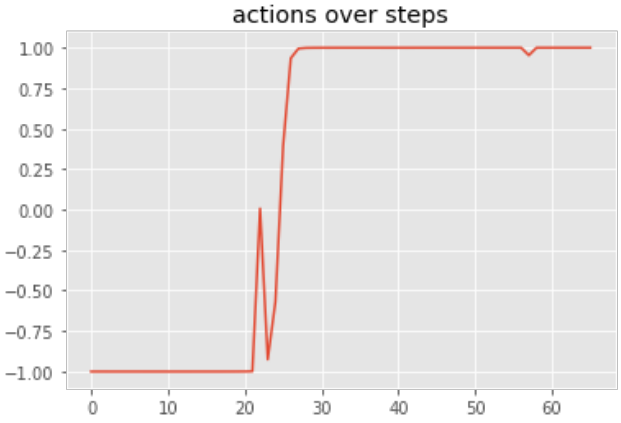
```
Episode =   60 (duration of 67 steps); Reward =  93.617 (best =  93.997, in epi
52)
        Episode training time:  2.3561136722564697
        resetting episode... next explore_p:  0.0852575903343082
Garbage collector: collected 5514 objects.
        step:  0 , action: [-1.]
Episode =   61 (duration of 68 steps); Reward =  93.691 (best =  93.997, in epi
52)
        Episode training time:  2.138866424560547
        resetting episode... next explore_p:  0.08099471081759278
Garbage collector: collected 0 objects.
        step:  0 , action: [-0.999]
Episode =   62 (duration of 72 steps); Reward =  93.442 (best =  93.997, in epi
52)
        Episode training time:  2.2136378288269043
        resetting episode... next explore_p:  0.07694497527671314
Garbage collector: collected 0 objects.
        step:  0 , action: [-1.]
Episode =   63 (duration of 67 steps); Reward =  93.801 (best =  93.997, in epi
52)
        Episode training time:  2.0557548999786377
        resetting episode... next explore_p:  0.07309772651287748
Garbage collector: collected 0 objects.
        step:  0 , action: [-1.]
Episode =   64 (duration of 67 steps); Reward =  93.589 (best =  93.997, in epi
52)
        Episode training time:  1.9376323223114014
        resetting episode... next explore_p:  0.0694428401872336
Garbage collector: collected 0 objects.
        step:  0 , action: [-1.]
Episode =   65 (duration of 66 steps); Reward =  93.667 (best =  93.997, in epi
52)
        Episode training time:  2.027273654937744
        resetting episode... next explore_p:  0.0659706981778719
Garbage collector: collected 0 objects.
        step:  0 , action: [-1.]
Episode =   66 (duration of 67 steps); Reward =  93.644 (best =  93.997, in epi
52)
        Episode training time:  2.0871872901916504
        resetting episode... next explore_p:  0.0626721632689783
Garbage collector: collected 0 objects.
        step:  0 , action: [-1.]
Episode =   67 (duration of 68 steps); Reward =  93.497 (best =  93.997, in epi
52)
        Episode training time:  2.088264226913452
        resetting episode... next explore_p:  0.059538555105529384
Garbage collector: collected 0 objects.
        step:  0 , action: [-1.]
Episode =   68 (duration of 70 steps); Reward =  93.430 (best =  93.997, in epi
52)
        Episode training time:  1.9435906410217285
        resetting episode... next explore_p:  0.05656162735025291
Garbage collector: collected 0 objects.
        step:  0 , action: [-1.]
Episode =   69 (duration of 66 steps); Reward =  93.554 (best =  93.997, in epi
52)
        Episode training time:  1.9996497631072998
        resetting episode... next explore_p:  0.053733545982740265
Garbage collector: collected 0 objects.
        step:  0 , action: [-1.]
```


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

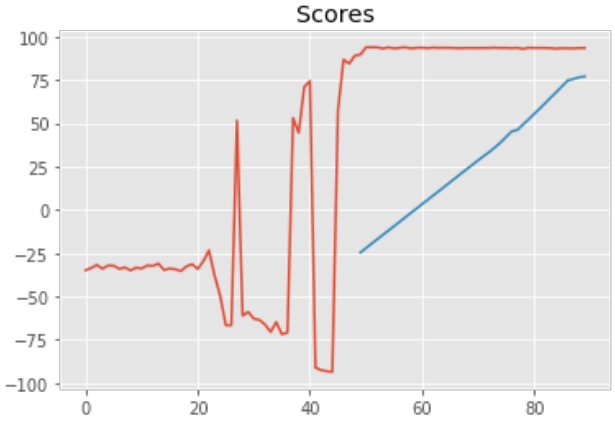

actions over steps
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
10
20
30
40
50
60

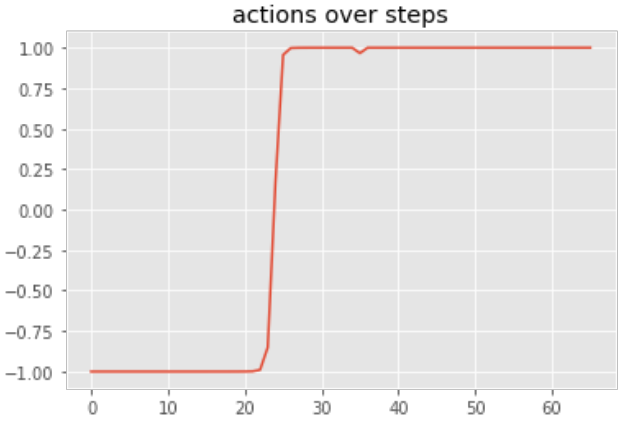
```
Episode =   70 (duration of 66 steps); Reward =  93.566 (best =  93.997, in epi
52)
        Episode training time:  2.3160619735717773
        resetting episode... next explore_p:  0.05104686868360325
Garbage collector: collected 5564 objects.
        step:  0 , action: [-1.]
Episode =   71 (duration of 66 steps); Reward =  93.534 (best =  93.997, in epi
52)
        Episode training time:  1.9383699893951416
        resetting episode... next explore_p:  0.04849452524942309
Garbage collector: collected 0 objects.
        step:  0 , action: [-1.]
Episode =   72 (duration of 66 steps); Reward =  93.561 (best =  93.997, in epi
52)
        Episode training time:  2.0041120052337646
        resetting episode... next explore_p:  0.04606979898695193
Garbage collector: collected 0 objects.
        step:  0 , action: [-1.]
Episode =   73 (duration of 67 steps); Reward =  93.599 (best =  93.997, in epi
52)
        Episode training time:  2.0504586696624756
        resetting episode... next explore_p:  0.04376630903760433
Garbage collector: collected 0 objects.
        step:  0 , action: [-0.999]
Episode =   74 (duration of 65 steps); Reward =  93.710 (best =  93.997, in epi
52)
        Episode training time:  2.0131146907806396
        resetting episode... next explore_p:  0.041577993585724116
Garbage collector: collected 0 objects.
        step:  0 , action: [-1.]
Episode =   75 (duration of 70 steps); Reward =  93.553 (best =  93.997, in epi
52)
        Episode training time:  1.9787070751190186
        resetting episode... next explore_p:  0.03949909390643791
Garbage collector: collected 0 objects.
        step:  0 , action: [-1.]
Episode =   76 (duration of 66 steps); Reward =  93.598 (best =  93.997, in epi
52)
        Episode training time:  2.06886887550354
        resetting episode... next explore_p:  0.03752413921111601
Garbage collector: collected 0 objects.
        step:  0 , action: [-1.]
Episode =   77 (duration of 69 steps); Reward =  93.461 (best =  93.997, in epi
52)
        Episode training time:  2.1457903385162354
        resetting episode... next explore_p:  0.03564793225056021
Garbage collector: collected 0 objects.
        step:  0 , action: [-1.]
Episode =   78 (duration of 65 steps); Reward =  93.637 (best =  93.997, in epi
52)
        Episode training time:  1.9992430210113525
        resetting episode... next explore_p:  0.0338655356380322
Garbage collector: collected 0 objects.
        step:  0 , action: [-1.]
Episode =   79 (duration of 78 steps); Reward =  93.002 (best =  93.997, in epi
52)
        Episode training time:  2.2186591625213623
        resetting episode... next explore_p:  0.032172258856130585
Garbage collector: collected 0 objects.
        step:  0 , action: [-1.]
```


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

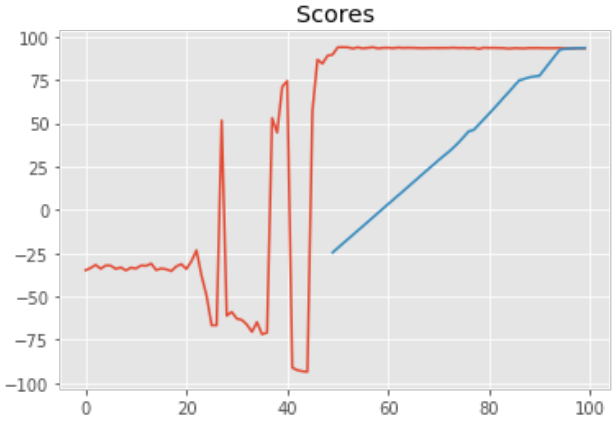

actions over steps
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
10
20
30
40
50
60

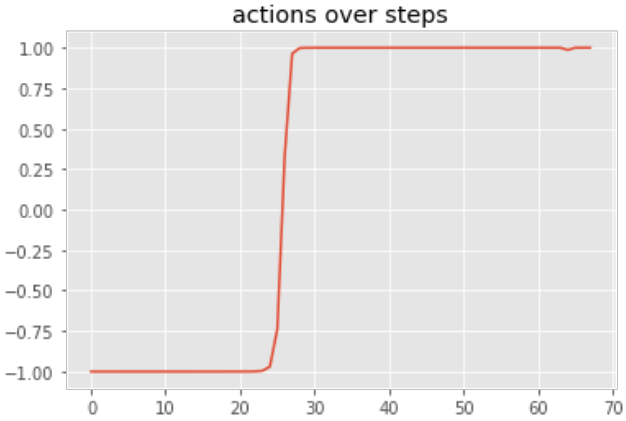
```
Episode =   80 (duration of 66 steps); Reward =  93.690 (best =  93.997, in epi
52)
        Episode training time:  2.2825353145599365
        resetting episode... next explore_p:  0.030563645913324056
Garbage collector: collected 5629 objects.
        step:  0 , action: [-1.]
Episode =   81 (duration of 66 steps); Reward =  93.573 (best =  93.997, in epi
52)
        Episode training time:  1.6299962997436523
        resetting episode... next explore_p:  0.029035463617657853
Garbage collector: collected 0 objects.
        step:  0 , action: [-1.]
Episode =   82 (duration of 66 steps); Reward =  93.620 (best =  93.997, in epi
52)
        Episode training time:  1.9522051811218262
        resetting episode... next explore_p:  0.027583690436774957
Garbage collector: collected 0 objects.
        step:  0 , action: [-1.]
Episode =   83 (duration of 66 steps); Reward =  93.533 (best =  93.997, in epi
52)
        Episode training time:  1.9783222675323486
        resetting episode... next explore_p:  0.02620450591493621
Garbage collector: collected 0 objects.
        step:  0 , action: [-1.]
Episode =   84 (duration of 68 steps); Reward =  93.461 (best =  93.997, in epi
52)
        Episode training time:  1.82802152633667
        resetting episode... next explore_p:  0.0248942806191894
Garbage collector: collected 0 objects.
        step:  0 , action: [-1.]
Episode =   85 (duration of 70 steps); Reward =  93.175 (best =  93.997, in epi
52)
        Episode training time:  1.979407787322998
        resetting episode... next explore_p:  0.023649566588229927
Garbage collector: collected 0 objects.
        step:  0 , action: [-1.]
Episode =   86 (duration of 67 steps); Reward =  93.463 (best =  93.997, in epi
52)
        Episode training time:  1.902723789215088
        resetting episode... next explore_p:  0.022467088258818428
Garbage collector: collected 0 objects.
        step:  0 , action: [-1.]
Episode =   87 (duration of 68 steps); Reward =  93.360 (best =  93.997, in epi
52)
        Episode training time:  2.069056272506714
        resetting episode... next explore_p:  0.021343733845877507
Garbage collector: collected 0 objects.
        step:  0 , action: [-1.]
Episode =   88 (duration of 69 steps); Reward =  93.320 (best =  93.997, in epi
52)
        Episode training time:  2.0173256397247314
        resetting episode... next explore_p:  0.02027654715358363
Garbage collector: collected 0 objects.
        step:  0 , action: [-1.]
Episode =   89 (duration of 66 steps); Reward =  93.521 (best =  93.997, in epi
52)
        Episode training time:  2.0447168350219727
        resetting episode... next explore_p:  0.019262719795904448
Garbage collector: collected 0 objects.
        step:  0 , action: [-1.]
```


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

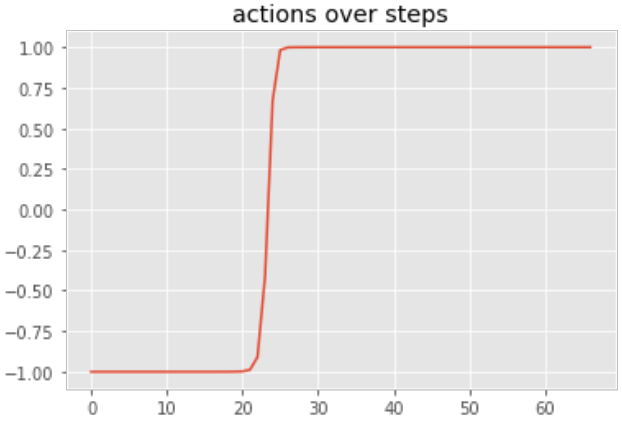

actions over steps
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
10
20
30
40
50
60

```
Episode =   90 (duration of 66 steps); Reward =  93.543 (best =  93.997, in epi
52)
        Episode training time:  2.3657758235931396
        resetting episode... next explore_p:  0.018299583806109226
Garbage collector: collected 5384 objects.
        step:  0 , action: [-1.]
Episode =   91 (duration of 68 steps); Reward =  93.527 (best =  93.997, in epi
52)
        Episode training time:  2.0191895961761475
        resetting episode... next explore_p:  0.017384604615803764
Garbage collector: collected 0 objects.
        step:  0 , action: [-1.]
Episode =   92 (duration of 67 steps); Reward =  93.435 (best =  93.997, in epi
52)
        Episode training time:  1.8538141250610352
        resetting episode... next explore_p:  0.016515374385013576
Garbage collector: collected 0 objects.
        step:  0 , action: [-1.]
Episode =   93 (duration of 68 steps); Reward =  93.443 (best =  93.997, in epi
52)
        Episode training time:  2.0606887340545654
        resetting episode... next explore_p:  0.015689605665762895
Garbage collector: collected 0 objects.
        step:  0 , action: [-1.]
Episode =   94 (duration of 66 steps); Reward =  93.497 (best =  93.997, in epi
52)
        Episode training time:  1.8788363933563232
        resetting episode... next explore_p:  0.01490512538247475
Garbage collector: collected 0 objects.
        step:  0 , action: [-1.]
Episode =   95 (duration of 66 steps); Reward =  93.527 (best =  93.997, in epi
52)
        Episode training time:  1.9727513790130615
        resetting episode... next explore_p:  0.014159869113351011
Garbage collector: collected 0 objects.
        step:  0 , action: [-1.]
Episode =   96 (duration of 69 steps); Reward =  93.301 (best =  93.997, in epi
52)
        Episode training time:  2.081477165222168
        resetting episode... next explore_p:  0.01345187565768346
Garbage collector: collected 0 objects.
        step:  0 , action: [-1.]
Episode =   97 (duration of 71 steps); Reward =  93.246 (best =  93.997, in epi
52)
        Episode training time:  2.0722436904907227
        resetting episode... next explore_p:  0.012779281874799287
Garbage collector: collected 0 objects.
        step:  0 , action: [-0.007]
Episode =   98 (duration of 68 steps); Reward =  93.433 (best =  93.997, in epi
52)
        Episode training time:  1.9968807697296143
        resetting episode... next explore_p:  0.012140317781059323
Garbage collector: collected 0 objects.
        step:  0 , action: [-1.]
Episode =   99 (duration of 66 steps); Reward =  93.505 (best =  93.997, in epi
52)
        Episode training time:  1.981321096420288
        resetting episode... next explore_p:  0.011533301892006355
Garbage collector: collected 0 objects.
        step:  0 , action: [-1.]
```

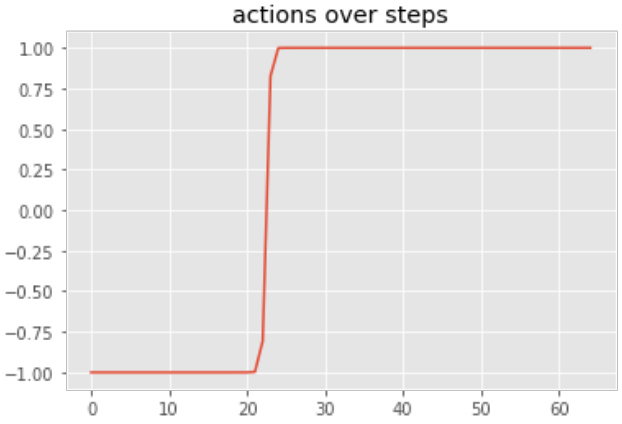




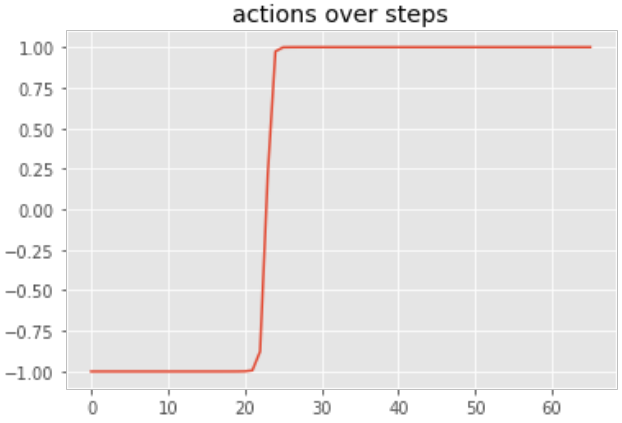

```
Episode =  100 (duration of 68 steps); Reward =  93.352 (best =  93.997, in epi
52)
        Episode training time:  2.4613993167877197

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

*** All episodes training time (HH:MM:SS):  0:16:40.446298
        Average training time per episode:  10.004462978839875
*****************************
[TEST] Training Done, now running tests...
        resetting episode... next explore_p:  0.010956636797406038
Garbage collector: collected 5514 objects.
        step:  0 , action: [-1.]
        step:  50 , action: [1.]
```



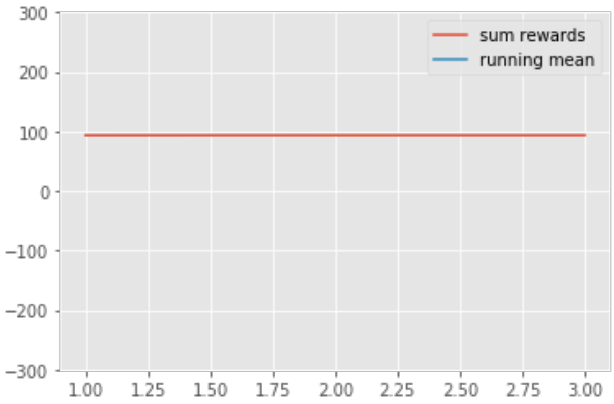

```
        resetting episode... next explore_p:  0.010408804957535735
Garbage collector: collected 2712 objects.
        step:  0 , action: [-1.]
        step:  50 , action: [1.]
```




```
        resetting episode... next explore_p:  0.009888364709658948
Garbage collector: collected 2712 objects.
        step:  0 , action: [-1.]
        step:  50 , action: [1.]
```



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

*** All episodes training time (HH:MM:SS):  0:00:01.174453
        Average training time per episode:  0.39148418108622235
******************************
```




```
Final score: 93.37505513843622
```

Reflection

Reinforcement learning is hard. It is certainly not as clean and well understood at supervised learning approaches [21]. Probably one of the reasons for this is that the reward function provided to agents has a much poorer signal to noise ration than cleaner labelled supervised learning data. Additionally, labels make it very explicit what the network is supposed to do, while in reinforcement learning the agent has to figure that out itself.

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). It was hard to know how much explore and exploit to perform, and some way of getting a metric from the agent to decide whether to explore or exploit, rather than just doing it randomly, would be very helpful.

Compute speed is important. Training takes a long time in the real world. For my i7-2600, a single experiment used about 20-30% of all cores, so around 2 experiments could run without too much slowdown for the given PCIe lanes in the i7. The GPU 1080ti usually had more than enough memory to handle the data sizes of the reinforcement learning environments and their batch sizes, except for the case of the image environemt space of Car Racing. In this image case only a single experiment could fit on GPU memory at once. This is kind of practical training knowledge would have been something that would have been very helpful to figure out early on, instead of later.

Creating a reinforcement learning framework is also a significant effort, particularly one that can interchange different agents and environments. Experiments on the relatively simple Mountain Car network took a long time (so many knobs to play with), and a lot of basic coding work was involved to build the training pipeline and convert the DDPG agent to work in image state coordinates (not to mention the unresolved memory leak when training on images).

Given these factors, this project scope was probably a bit too ambituous given the small timeline for completion, and my own limited experience in reinforcement learning.

That said, I definitely learned a lot.

Improvement

More experiments with Walker and Lunar lander would be informative. Due to time contraints, unforuntately there was not the same rigor of experimentation on those environments as on Mountain Car. Better performance on Lunar Lander and Walker would be desired before attempting the Car Racing environment again, since that agent really takes much longer to train (though the memory leak of in the Car Racing Agent implementation should be investigated and dealt with).

Following up the car racing game with Atari games would also be of interest, as the Car Racing environment, which also uses pixels as input state space, is largely similar and should not take too much modification to make work.

Different forms of exploration could be further investigated, though it did seem that allowing the agent some amount of pure exploitation, without action noise, helped convergence to an optimal policty. There is no direct feedback from the agent about whether it needs to explore or exploit. Rather, like many things in Machine Learning, this is still a hard coded parameter and algorithm. Reseach into the agent deciding when it needs to explore of exploit, and by how much, would be very interesting.

A kind of parameter search for reinforcement learning would be of interest, as was manually done in this project for Mountain Car. However, due to the sheen number of options in the current DDPG agent, it seems like it would be pretty difficult to implement using standard libraries.

DDQN, Double Deep Q Network, seems to be the state of the art for Reinforcement Learning recently. Implementing some of the innovations in "Rainbow: Combining Improvements in Deep Reinforcement Learning" [17], such as prioritized replay, dueling networks, multi-step learning, noisy nets, and distribuitional RL would be of interest. It would be good to directly compare the latest state of the art in DDQN with DDPG directly.

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)
- [21] Reinforcement Learning Doesn't Work Yet - https://www.alexirpan.com/2018/02/14/rl-hard.html (https://www.alexirpan.com/2018/02/14/rl-hard.html)
- [22] 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)
- [23] Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor. TuomasHaarnoja, Aurick Zhou, Pieter Abbeel, Sergey Levine