Deep Q-learning

The goal of a DQN agent is to maximize the future discounted return at each timestep t, namely

$$R_t = \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'}$$

assuming the environment episode ends at timestep T. The optimal action-value function $Q^*(s; a)$ defines the maximum discounted return achievable, i.e. when following an optimal policy π^*. This optimal action-value function satisfies a recursive relationship called the Bellman optimality Eq. (1), where $\mathcal{S}$ is the distribution over next states s' given a state s_t and action a_t:

$$Q^*(s,a) := \max_{\pi} \mathbb{E}_{\pi}\left[R_t \,\middle|\, s_t = s,\ a_t = a\right] \implies Q^*(s,a) = \mathbb{E}_{s'\sim\mathcal{S}}\left[r + \gamma \max_{a'} Q^*(s',a') \,\middle|\, s,a\right] \tag{1}$$

Generally, we can estimate this optimal Q-function by updating the Q-value function in an iterative fashion as

$$Q_{i+1}(s,a) = \mathbb{E}_{s'\sim\mathcal{S}}\left[r + \gamma \max_{a'} Q_i(s',a') \,\middle|\, s,a\right] \tag{2}$$

which ultimately converges to Q^* as the iterations i goes to infinity. In DQN we use a function approximator to represent the Q-value function. Therefore, instead of assigning values as in Eq. (2) we solve a regression problem, as detailed below in Section 2. Also, instead of trying to impose Eq. (2) in all (s, a) pairs, they are sampled from a *replay buffer* that at every iteration received new pairs obtained by executing in the environment the actions given by an "epsilon-greedy" sampling proceedure also described in Section 2.

In this assignemnt you will be asked to implement three parts:

- Define a Neural Network class that will be used as the Q-function approximator.
- Implement the epsilon-greedy sampling proceedure.
- Implement the Q-learning loss function.

Then you will be able to test your algorithm in two environments: a simple grid-world and a more complex Atari game called Pong.

In [1]:
```
# import helpers, gym environments, and other needed dependencies
from collections import deque
import time
import numpy as np
import pickle
import os.path as osp
import click
import gym

from simpledqn.replay_buffer import ReplayBuffer
import logger
from simpledqn.wrappers import NoopResetEnv, EpisodicLifeEnv
from simpledqn import gridworld_env
```

```
from simpledqn.main import assert_allclose, preprocess_obs_gridworld,
preprocess_obs_ram, LinearSchedule

nprs = np.random.RandomState
rng = nprs(42)
```

1. Construcing a Neural Network

Build a NN with **3 linear layers** (take 256 for all hidden sizes) and **relu** non-linearities at each layer output but the last.

In [2]:
```
import torch
import torch.nn as nn
import torch.optim as optim
import torch.nn.functional as F
import torch.autograd as autograd

class NN_linear(nn.Module):
    def __init__(self, obs_size, act_size):
        super(NN_linear, self).__init__()
        self.Linear = nn.Linear(obs_size, act_size)

    def forward(self, obs):
        out = self.Linear(obs)
        return out

class NN(nn.Module):
    def __init__(self, obs_size, act_size):
        super(NN, self).__init__()
        #"*** YOUR CODE HERE ***"
        self.linear1 = NN_linear(obs_size, 256)
        self.linear2 = NN_linear(256, 256)
        self.linear3 = NN_linear(256, act_size)
    def forward(self, obs):
        #"*** YOUR CODE HERE ***"
        out = F.relu(self.linear1(obs))
        out = F.relu(self.linear2(out))
```

```
        out = self.linear3(out)
        return out
```

2. Training the Q-function approximators

The function $Q(s, a; \theta)$ is trained to approximate $Q^*(s, a)$ over time using a loss function defined as:

$$\mathbb{E}_{(s,a,s') \sim \mathcal{D}}\left[(y - Q(s, a; \theta))^2\right], \qquad \text{where} \qquad y = \begin{cases} r + \gamma \max_{a'} Q(s', a'; \theta') & \text{if non-terminal trans} \\ r & \text{for terminal transitio} \end{cases}$$

where the network $Q(s; a; \theta')$ is called the target network, and its parameters θ' are updated (i.e. set to the current value of θ) at a specific interval. DQN is inherently off-policy, which means that we can update the agent towards the goal behavior through using data that is sampled from arbitrary behavior. Therefore, all sampled $(s; a; s'; r)$ tuples are stored in a replay buffer $\mathcal{D}$. The approximator $Q(s, a; \theta)$ is updated by minimizing the loss described in Eq. (3). In between updates, we add new tuples (s, a, s', r) to the replay buffer by taking actions in the environment with and **epsilon greedy** proceedure:

for t from 1 to T do:

- with probability ϵ select random action a_t, otherwise select $a_t = \max_a Q(s, a; \theta)$
- execute action a_t in environment and observe reward r_t, next state s_{t+1} and episode termination signal d_t
- store transition $(s_t, a_t, r_t, s_{t+1}, d_t)$ in $\mathcal{D}$.

end

In the next DQN class do the following:

- complete the **epsilon greedy** action sampling
- write the full `compute_q_learning_loss` function

In [3]:
```
class DQN(object):
    def __init__(self, env, obs_dim, act_dim, obs_preprocessor, replay
_buffer, NN,
                 opt_batch_size, discount, initial_step, max_steps, le
arning_start_itr, target_q_update_freq,
                 train_q_freq, log_freq, final_eps, initial_eps, fract
ion_eps, render):
        self. env = env
```

```
        self._obs_dim = obs_dim
        self._act_dim = act_dim
        self._obs_preprocessor = obs_preprocessor
        self._replay_buffer = replay_buffer
        self._initial_step = initial_step
        self._max_steps = max_steps
        self._target_q_update_freq = target_q_update_freq
        self._learning_start_itr = learning_start_itr
        self._train_q_freq = train_q_freq
        self._log_freq = log_freq
        self._opt_batch_size = opt_batch_size
        self._discount = discount
        self._render = render


        self._q = NN(self._obs_dim, self._act_dim)  # Q function which
params are optimized
        self._qt = NN(self._obs_dim, self._act_dim)  # target Q copyin
g the params in Q after several updates
        self._qt.requires_grad = False

        self.optimizer = optim.Adam(self._q.parameters(), lr=0.0001)

        self.exploration = LinearSchedule(  # gives value of eps acros
s iterations
            schedule_timesteps=int(fraction_eps * max_steps),
            initial_p=initial_eps,
            final_p=final_eps)

    def eps_greedy(self, obs, epsilon):
        # Check Q function, do argmax.
        rnd = rng.rand()
        if rnd > epsilon:
            obs = self._obs_preprocessor(obs)
            #"*** YOUR CODE HERE ***"
            # compute q_values of obs
            obs_var = autograd.Variable(torch.from_numpy(obs), require
s_grad=False)
            q_values = self._qt(obs_var) #TO DO: or self._q?
            # return the greedy action
            return np.argmax(q_values.data.numpy())
        else:
            return rng.randint(0, self._act_dim)

    def compute_q_learning_loss(self, l_obs, l_act, l_rew, l_next_obs,
l_done):
        """
        :param l_obs: A np.array holding a list of observations. Shoul
d be of shape N * |S|.
        :param l_act: A np.array variable holding a list of actions. S
hould be of shape N.
        :param l_rew: A np.array variable holding a list of rewards. S
hould be of shape N.
        :param l_next_obs: A np.array variable holding a list of obser
vations at the next time step. Should be of
        shape N * |S|.
        :param l_done: A np.array variable holding a list of binary va
lues (indicating whether episode ended after this
        time step). Should be of shape N.
        :return: A PyTorch Variable holding a scalar loss.
        """

        "*** YOUR CODE HERE ***"
        # wrap the observations into Variables
        l_next_obs_var = autograd.Variable(torch.Tensor(l_next_obs), r
equires_grad=False)
        l_obs_var = autograd.Variable(torch.Tensor(l_obs), requires_gr
```

```
ad=False)

        # compute Q values of the next_obs based on the target Q netwo
rk self._qt, and convert back to numpy
        qt_next = self._qt(l_next_obs_var).data.numpy()  # shape (N, a
ct_dim)
        qt_next = np.amax(qt_next, 1) # shape N

        # compute the target for the MSELoss (you can do it entirely i
n numpy). Use self._discount
        target = l_rew + (1-l_done) * self._discount * qt_next

        # wrap into a Variable
        target = autograd.Variable(torch.Tensor(target), requires_grad
=False)
        # compute Q values self._q of current obs and select the one c
orresponding to the action that was taken
        q_all = self._q(l_obs_var)

        l_act_tensor = torch.from_numpy(l_act).long().unsqueeze(1)
        l_act_var = autograd.Variable(l_act_tensor, requires_grad=Fals
e)
        q_sel = torch.gather(q_all, 1, l_act_var)

        # form the MSELOss and compute it
        #loss = autograd.Variable(torch.Tensor([0]), requires_grad=Fal
se)
        loss = F.mse_loss(q_sel, target)
        return loss


    def train_q(self, l_obs, l_act, l_rew, l_next_obs, l_done):
        """Update Q-value function by sampling from the replay buffer.
"""
        self._q.zero_grad()

        l_obs = self._obs_preprocessor(l_obs)
        l_next_obs = self._obs_preprocessor(l_next_obs)

        loss = self.compute_q_learning_loss(
            l_obs, l_act, l_rew, l_next_obs, l_done)

        loss.backward()
        self.optimizer.step()

        return loss.data

    def _update_target_q(self):
        """Update the target Q-value function by copying the current Q
-value function weights."""
        q_params_dict = dict(self._q.named_parameters())
        self._qt.load_state_dict(q_params_dict)


    def train(self):
        obs = self._env.reset()

        episode_rewards = []
        n_episodes = 0
        l_episode_return = deque([], maxlen=10)
        l_discounted_episode_return = deque([], maxlen=10)
        l_tq_squared_error = deque(maxlen=50)
        log_itr = -1
        for itr in range(self._initial_step, self._max_steps):
            act = self.eps_greedy(obs[np.newaxis, :],
                                  self.exploration.value(itr))
            next_obs, rew, done, _ = self._env.step(act)
```

```
            if self._render:
                self._env.render()
            self._replay_buffer.add(obs, act, rew, next_obs, float(don
e))

            episode_rewards.append(rew)

            if done:
                obs = self._env.reset()
                episode_return = np.sum(episode_rewards)
                discounted_episode_return = np.sum(
                    episode_rewards * self._discount ** np.arange(len(
episode_rewards)))
                l_episode_return.append(episode_return)
                l_discounted_episode_return.append(discounted_episode_
return)
                episode_rewards = []
                n_episodes += 1
            else:
                obs = next_obs

            if itr % self._target_q_update_freq == 0 and itr > self._l
earning_start_itr:
                self._update_target_q()

            if itr % self._train_q_freq == 0 and itr > self._learning_
start_itr:
                # Sample from replay buffer.
                l_obs, l_act, l_rew, l_obs_prime, l_done = self._repla
y_buffer.sample(
                    self._opt_batch_size)
                # Train Q value function with sampled data.
                td_squared_error = self.train_q(
                    l_obs, l_act, l_rew, l_obs_prime, l_done)
                l_tq_squared_error.append(td_squared_error)

            if (itr + 1) % self._log_freq == 0 and len(l_episode_retur
n) > 5:
                log_itr += 1
                logger.logkv('Iteration', log_itr)

                logger.logkv('Steps', itr)
                logger.logkv('Epsilon', self.exploration.value(itr))
                logger.logkv('Episodes', n_episodes)
                logger.logkv('AverageReturn', np.mean(l_episode_return
))
                logger.logkv('AverageDiscountedReturn',
                             np.mean(l_discounted_episode_return))
                logger.logkv('TDError^2', np.mean(l_tq_squared_error))
                logger.dumpkvs()
#                 self._q.dump(logger.get_dir() + '/weights.pkl')

    def test(self, epsilon):
        try:
            self._q.set_params(self._q.load(logger.get_dir() + '/weigh
ts.pkl'))
        except Exception as e:
            print(e)
        obs = self._env.reset()
        while True:
            act = self.eps_greedy(obs[np.newaxis, :], epsilon)
            obs_prime, rew, done, _ = self._env.step(act)
            self._env.render()
            if done:
                obs = self._env.reset()
                print('Done!')
                time.sleep(1)
            else:
```

```
                obs = obs_prime
```

3. Test the algorithm on grid world

Now let's train a simple GridWorld to test out our algorithm!

In [4]:
```
env = gym.make('GridWorld-v0')
test_dir = "data/local/dqn_gridworld_test"
log_dir = "data/local/dqn_gridworld2"
logger.session(log_dir).__enter__()
env.seed(42)

# Initialize the replay buffer that we will use.
replay_buffer = ReplayBuffer(max_size=10000)

# Initialize DQN training procedure.
dqn_gridworld = DQN(
    env=env,
    obs_dim=env.observation_space.n,
    act_dim=env.action_space.n,
    NN=NN_linear,
    obs_preprocessor=preprocess_obs_gridworld,
    replay_buffer=replay_buffer,
    opt_batch_size=64,
    # DQN gamma parameter
    discount=0.99,
    # Training procedure length
    initial_step=0,
    max_steps=100000,
    learning_start_itr=1000,
    # Frequency of copying the actual Q to the target Q
    target_q_update_freq=100,
    # Frequency of updating the Q-value function
    train_q_freq=4,
    # Exploration parameters
    initial_eps=1.0,
    final_eps=0.05,
    fraction_eps=0.1,
    # Logging
    log_freq=1000,
    render=False,
)

from simpledqn.main import test_loss
test_loss(dqn_gridworld, test_dir)
```

```
[2018-04-23 22:32:47,690] Making new env: GridWorld-v0

[ 2.21811724]
Test for compute_q_learning_loss passed!
```

If you passed the previous test, let's train the full policy!

In [5]:
```
dqn_gridworld.train()
```

```
-------------------------------------
| Iteration                 | 0        |
| Steps                     | 999      |
| Epsilon                   | 0.90509  |
| Episodes                  | 53       |
| AverageReturn             | 0.1      |
| AverageDiscountedReturn   | 0.091352 |
| TDError^2                 | nan      |
-------------------------------------

/Applications/anaconda/envs/deeprlbootcamp/lib/python3.5/site-packag
es/numpy/core/fromnumeric.py:2909: RuntimeWarning: Mean of empty sli
ce.
  out=out, **kwargs)
/Applications/anaconda/envs/deeprlbootcamp/lib/python3.5/site-packag
es/numpy/core/_methods.py:80: RuntimeWarning: invalid value encounte
red in double_scalars
  ret = ret.dtype.type(ret / rcount)

-------------------------------------
| Iteration                 | 1        |
| Steps                     | 1999     |
| Epsilon                   | 0.8101   |
| Episodes                  | 108      |
| AverageReturn             | 0.1      |
| AverageDiscountedReturn   | 0.077004 |
| TDError^2                 | 0.074823 |
-------------------------------------
-------------------------------------
| Iteration                 | 2        |
| Steps                     | 2999     |
| Epsilon                   | 0.7151   |
| Episodes                  | 165      |
| AverageReturn             | 0        |
| AverageDiscountedReturn   | 0        |
| TDError^2                 | 0.062754 |
-------------------------------------
-------------------------------------
| Iteration                 | 3        |
| Steps                     | 3999     |
| Epsilon                   | 0.6201   |
| Episodes                  | 234      |
| AverageReturn             | 0.1      |
| AverageDiscountedReturn   | 0.093207 |
| TDError^2                 | 0.05468  |
-------------------------------------
-------------------------------------
| Iteration                 | 4        |
| Steps                     | 4999     |
| Epsilon                   | 0.5251   |
| Episodes                  | 301      |
| AverageReturn             | 0.6      |
| AverageDiscountedReturn   | 0.51879  |
| TDError^2                 | 0.045616 |
```

```
--------------------------------------
--------------------------------------
| Iteration               | 5        |
| Steps                   | 5999     |
| Epsilon                 | 0.4301   |
| Episodes                | 378      |
| AverageReturn           | 0.2      |
| AverageDiscountedReturn | 0.18456  |
| TDError^2               | 0.040972 |
--------------------------------------
--------------------------------------
| Iteration               | 6        |
| Steps                   | 6999     |
| Epsilon                 | 0.3351   |
| Episodes                | 435      |
| AverageReturn           | 0.1      |
| AverageDiscountedReturn | 0.094148 |
| TDError^2               | 0.034993 |
--------------------------------------
--------------------------------------
| Iteration               | 7        |
| Steps                   | 7999     |
| Epsilon                 | 0.24009  |
| Episodes                | 498      |
| AverageReturn           | 0.2      |
| AverageDiscountedReturn | 0.18737  |
| TDError^2               | 0.028431 |
--------------------------------------
--------------------------------------
| Iteration               | 8        |
| Steps                   | 8999     |
| Epsilon                 | 0.14509  |
| Episodes                | 562      |
| AverageReturn           | 0        |
| AverageDiscountedReturn | 0        |
| TDError^2               | 0.023904 |
--------------------------------------
--------------------------------------
| Iteration               | 9        |
| Steps                   | 9999     |
| Epsilon                 | 0.050095 |
| Episodes                | 599      |
| AverageReturn           | 0.1      |
| AverageDiscountedReturn | 0.084294 |
| TDError^2               | 0.019751 |
--------------------------------------
--------------------------------------
| Iteration               | 10       |
| Steps                   | 10999    |
| Epsilon                 | 0.05     |
| Episodes                | 632      |
| AverageReturn           | 0.2      |
| AverageDiscountedReturn | 0.16379  |

| TDError^2               | 0.014555 |
--------------------------------------
--------------------------------------
| Iteration               | 11       |
| Steps                   | 11999    |
| Epsilon                 | 0.05     |
| Episodes                | 664      |
| AverageReturn           | 0.1      |
| AverageDiscountedReturn | 0.094148 |
| TDError^2               | 0.011821 |
--------------------------------------
--------------------------------------
| Iteration               | 12       |
```

```
| Steps                     | 12999    |
| Epsilon                   | 0.05     |
| Episodes                  | 809      |
| AverageReturn             | 1        |
| AverageDiscountedReturn   | 0.94436  |
| TDError^2                 | 0.011044 |
--------------------------------------
--------------------------------------
| Iteration                 | 13       |
| Steps                     | 13999    |
| Epsilon                   | 0.05     |
| Episodes                  | 964      |
| AverageReturn             | 1        |
| AverageDiscountedReturn   | 0.95099  |
| TDError^2                 | 0.011273 |
--------------------------------------
--------------------------------------
| Iteration                 | 14       |
| Steps                     | 14999    |
| Epsilon                   | 0.05     |
| Episodes                  | 1123     |
| AverageReturn             | 1        |
| AverageDiscountedReturn   | 0.94721  |
| TDError^2                 | 0.011528 |
--------------------------------------
--------------------------------------
| Iteration                 | 15       |
| Steps                     | 15999    |
| Epsilon                   | 0.05     |
| Episodes                  | 1284     |
| AverageReturn             | 1        |
| AverageDiscountedReturn   | 0.95099  |
| TDError^2                 | 0.011049 |
--------------------------------------
---------------------------------------
| Iteration                 | 16        |
| Steps                     | 16999     |
| Epsilon                   | 0.05      |
| Episodes                  | 1442      |
| AverageReturn             | 0.9       |

| AverageDiscountedReturn   | 0.854     |
| TDError^2                 | 0.0094323 |
---------------------------------------
--------------------------------------
| Iteration                 | 17       |
| Steps                     | 17999    |
| Epsilon                   | 0.05     |
| Episodes                  | 1604     |
| AverageReturn             | 1        |
| AverageDiscountedReturn   | 0.95099  |
| TDError^2                 | 0.010256 |
--------------------------------------
---------------------------------------
| Iteration                 | 18        |
| Steps                     | 18999     |
| Epsilon                   | 0.05      |
| Episodes                  | 1764      |
| AverageReturn             | 1         |
| AverageDiscountedReturn   | 0.94353   |
| TDError^2                 | 0.0095138 |
---------------------------------------
---------------------------------------
| Iteration                 | 19        |
| Steps                     | 19999     |
| Epsilon                   | 0.05      |
| Episodes                  | 1923      |
| AverageReturn             | 0.9       |
```

```
| AverageDiscountedReturn | 0.85494   |
| TDError^2               | 0.0089441 |
---------------------------------------
---------------------------------------
| Iteration               | 20        |
| Steps                   | 20999     |
| Epsilon                 | 0.05      |
| Episodes                | 2085      |
| AverageReturn           | 1         |
| AverageDiscountedReturn | 0.95004   |
| TDError^2               | 0.0076041 |
---------------------------------------
---------------------------------------
| Iteration               | 21        |
| Steps                   | 21999     |
| Epsilon                 | 0.05      |
| Episodes                | 2244      |
| AverageReturn           | 1         |
| AverageDiscountedReturn | 0.95004   |
| TDError^2               | 0.0064127 |
---------------------------------------
---------------------------------------
| Iteration               | 22        |
| Steps                   | 22999     |
| Epsilon                 | 0.05      |
| Episodes                | 2404      |

| AverageReturn           | 1         |
| AverageDiscountedReturn | 0.94721   |
| TDError^2               | 0.0084456 |
---------------------------------------
---------------------------------------
| Iteration               | 23        |
| Steps                   | 23999     |
| Epsilon                 | 0.05      |
| Episodes                | 2563      |
| AverageReturn           | 1         |
| AverageDiscountedReturn | 0.95099   |
| TDError^2               | 0.0064245 |
---------------------------------------
---------------------------------------
| Iteration               | 24        |
| Steps                   | 24999     |
| Epsilon                 | 0.05      |
| Episodes                | 2723      |
| AverageReturn           | 1         |
| AverageDiscountedReturn | 0.95099   |
| TDError^2               | 0.0075563 |
---------------------------------------
---------------------------------------
| Iteration               | 25        |
| Steps                   | 25999     |
| Epsilon                 | 0.05      |
| Episodes                | 2884      |
| AverageReturn           | 1         |
| AverageDiscountedReturn | 0.9491    |
| TDError^2               | 0.0059904 |
---------------------------------------
---------------------------------------
| Iteration               | 26        |
| Steps                   | 26999     |
| Epsilon                 | 0.05      |
| Episodes                | 3042      |
| AverageReturn           | 1         |
| AverageDiscountedReturn | 0.95099   |
| TDError^2               | 0.0074594 |
---------------------------------------
```

```
---------------------------------------
| Iteration               | 27        |
| Steps                   | 27999     |
| Epsilon                 | 0.05      |
| Episodes                | 3202      |
| AverageReturn           | 1         |
| AverageDiscountedReturn | 0.95099   |
| TDError^2               | 0.0054952 |
---------------------------------------
---------------------------------------
| Iteration               | 28        |
| Steps                   | 28999     |
| Epsilon                 | 0.05      |

| Episodes                | 3362      |
| AverageReturn           | 1         |
| AverageDiscountedReturn | 0.9491    |
| TDError^2               | 0.0063458 |
---------------------------------------
---------------------------------------
| Iteration               | 29        |
| Steps                   | 29999     |
| Epsilon                 | 0.05      |
| Episodes                | 3522      |
| AverageReturn           | 1         |
| AverageDiscountedReturn | 0.95004   |
| TDError^2               | 0.0056208 |
---------------------------------------
---------------------------------------
| Iteration               | 30        |
| Steps                   | 30999     |
| Epsilon                 | 0.05      |
| Episodes                | 3681      |
| AverageReturn           | 1         |
| AverageDiscountedReturn | 0.95099   |
| TDError^2               | 0.0051003 |
---------------------------------------
---------------------------------------
| Iteration               | 31        |
| Steps                   | 31999     |
| Epsilon                 | 0.05      |
| Episodes                | 3842      |
| AverageReturn           | 1         |
| AverageDiscountedReturn | 0.9491    |
| TDError^2               | 0.0049228 |
---------------------------------------
---------------------------------------
| Iteration               | 32        |
| Steps                   | 32999     |
| Epsilon                 | 0.05      |
| Episodes                | 4003      |
| AverageReturn           | 0.9       |
| AverageDiscountedReturn | 0.854     |
| TDError^2               | 0.0041008 |
---------------------------------------
---------------------------------------
| Iteration               | 33        |
| Steps                   | 33999     |
| Epsilon                 | 0.05      |
| Episodes                | 4161      |
| AverageReturn           | 1         |
| AverageDiscountedReturn | 0.94626   |
| TDError^2               | 0.0030242 |
---------------------------------------
---------------------------------------
| Iteration               | 34        |
| Steps                   | 34999     |
```

```
| Epsilon                   | 0.05      |
| Episodes                  | 4317      |
| AverageReturn             | 1         |
| AverageDiscountedReturn   | 0.94438   |
| TDError^2                 | 0.0035409 |
---------------------------------------
---------------------------------------
| Iteration                 | 35        |
| Steps                     | 35999     |
| Epsilon                   | 0.05      |
| Episodes                  | 4476      |
| AverageReturn             | 1         |
| AverageDiscountedReturn   | 0.95099   |
| TDError^2                 | 0.0029583 |
---------------------------------------
---------------------------------------
| Iteration                 | 36        |
| Steps                     | 36999     |
| Epsilon                   | 0.05      |
| Episodes                  | 4634      |
| AverageReturn             | 0.9       |
| AverageDiscountedReturn   | 0.85305   |
| TDError^2                 | 0.0029798 |
---------------------------------------
---------------------------------------
| Iteration                 | 37        |
| Steps                     | 37999     |
| Epsilon                   | 0.05      |
| Episodes                  | 4795      |
| AverageReturn             | 1         |
| AverageDiscountedReturn   | 0.94721   |
| TDError^2                 | 0.0031517 |
---------------------------------------
---------------------------------------
| Iteration                 | 38        |
| Steps                     | 38999     |
| Epsilon                   | 0.05      |
| Episodes                  | 4955      |
| AverageReturn             | 1         |
| AverageDiscountedReturn   | 0.94815   |
| TDError^2                 | 0.0019956 |
---------------------------------------
---------------------------------------
| Iteration                 | 39        |
| Steps                     | 39999     |
| Epsilon                   | 0.05      |
| Episodes                  | 5116      |
| AverageReturn             | 1         |
| AverageDiscountedReturn   | 0.9491    |
| TDError^2                 | 0.0018903 |
---------------------------------------
---------------------------------------
| Iteration                 | 40        |

| Steps                     | 40999     |
| Epsilon                   | 0.05      |
| Episodes                  | 5275      |
| AverageReturn             | 1         |
| AverageDiscountedReturn   | 0.94534   |
| TDError^2                 | 0.0018912 |
---------------------------------------
--------------------------------------
| Iteration                 | 41       |
| Steps                     | 41999    |
| Epsilon                   | 0.05     |
| Episodes                  | 5434     |
| AverageReturn             | 1        |
```

```
| AverageDiscountedReturn | 0.94156  |
| TDError^2               | 0.002019 |
-------------------------------------
--------------------------------------
| Iteration               | 42        |
| Steps                   | 42999     |
| Epsilon                 | 0.05      |
| Episodes                | 5594      |
| AverageReturn           | 1         |
| AverageDiscountedReturn | 0.95099   |
| TDError^2               | 0.0013095 |
--------------------------------------
-------------------------------------
| Iteration               | 43       |
| Steps                   | 43999    |
| Epsilon                 | 0.05     |
| Episodes                | 5754     |
| AverageReturn           | 1        |
| AverageDiscountedReturn | 0.94531  |
| TDError^2               | 0.001506 |
-------------------------------------
--------------------------------------
| Iteration               | 44        |
| Steps                   | 44999     |
| Epsilon                 | 0.05      |
| Episodes                | 5916      |
| AverageReturn           | 1         |
| AverageDiscountedReturn | 0.9491    |
| TDError^2               | 0.0012307 |
--------------------------------------
--------------------------------------
| Iteration               | 45        |
| Steps                   | 45999     |
| Epsilon                 | 0.05      |
| Episodes                | 6078      |
| AverageReturn           | 1         |
| AverageDiscountedReturn | 0.9472    |
| TDError^2               | 0.0011052 |
--------------------------------------
--------------------------------------

| Iteration               | 46        |
| Steps                   | 46999     |
| Epsilon                 | 0.05      |
| Episodes                | 6200      |
| AverageReturn           | 0.8       |
| AverageDiscountedReturn | 0.75704   |
| TDError^2               | 0.0014229 |
--------------------------------------
--------------------------------------
| Iteration               | 47        |
| Steps                   | 47999     |
| Epsilon                 | 0.05      |
| Episodes                | 6305      |
| AverageReturn           | 0.2       |
| AverageDiscountedReturn | 0.15132   |
| TDError^2               | 0.0022801 |
--------------------------------------
--------------------------------------
| Iteration               | 48        |
| Steps                   | 48999     |
| Epsilon                 | 0.05      |
| Episodes                | 6454      |
| AverageReturn           | 1         |
| AverageDiscountedReturn | 0.94721   |
| TDError^2               | 0.0024836 |
--------------------------------------
--------------------------------------
```

```
| Iteration               | 49        |
| Steps                   | 49999     |
| Epsilon                 | 0.05      |
| Episodes                | 6600      |
| AverageReturn           | 0.8       |
| AverageDiscountedReturn | 0.75701   |
| TDError^2               | 0.0019243 |
---------------------------------------
---------------------------------------
| Iteration               | 50        |
| Steps                   | 50999     |
| Epsilon                 | 0.05      |
| Episodes                | 6731      |
| AverageReturn           | 1         |
| AverageDiscountedReturn | 0.94542   |
| TDError^2               | 0.0016162 |
---------------------------------------
---------------------------------------
| Iteration               | 51        |
| Steps                   | 51999     |
| Epsilon                 | 0.05      |
| Episodes                | 6878      |
| AverageReturn           | 0.8       |
| AverageDiscountedReturn | 0.74221   |
| TDError^2               | 0.0035218 |
---------------------------------------

---------------------------------------
| Iteration               | 52        |
| Steps                   | 52999     |
| Epsilon                 | 0.05      |
| Episodes                | 7025      |
| AverageReturn           | 1         |
| AverageDiscountedReturn | 0.94817   |
| TDError^2               | 0.0020907 |
---------------------------------------
---------------------------------------
| Iteration               | 53        |
| Steps                   | 53999     |
| Epsilon                 | 0.05      |
| Episodes                | 7183      |
| AverageReturn           | 1         |
| AverageDiscountedReturn | 0.95099   |
| TDError^2               | 0.0023268 |
---------------------------------------
---------------------------------------
| Iteration               | 54        |
| Steps                   | 54999     |
| Epsilon                 | 0.05      |
| Episodes                | 7343      |
| AverageReturn           | 1         |
| AverageDiscountedReturn | 0.95099   |
| TDError^2               | 0.0022476 |
---------------------------------------
---------------------------------------
| Iteration               | 55        |
| Steps                   | 55999     |
| Epsilon                 | 0.05      |
| Episodes                | 7505      |
| AverageReturn           | 0.9       |
| AverageDiscountedReturn | 0.854     |
| TDError^2               | 0.0019097 |
---------------------------------------
---------------------------------------
| Iteration               | 56        |
| Steps                   | 56999     |
| Epsilon                 | 0.05      |
```

```
| Episodes                | 7666      |
| AverageReturn           | 1         |
| AverageDiscountedReturn | 0.95099   |
| TDError^2               | 0.0020106 |
---------------------------------------
---------------------------------------
| Iteration               | 57        |
| Steps                   | 57999     |
| Epsilon                 | 0.05      |
| Episodes                | 7827      |
| AverageReturn           | 1         |
| AverageDiscountedReturn | 0.94815   |
| TDError^2               | 0.0031277 |

---------------------------------------
----------------------------------------
| Iteration               | 58         |
| Steps                   | 58999      |
| Epsilon                 | 0.05       |
| Episodes                | 7989       |
| AverageReturn           | 0.9        |
| AverageDiscountedReturn | 0.854      |
| TDError^2               | 0.00085906 |
----------------------------------------
---------------------------------------
| Iteration               | 59        |
| Steps                   | 59999     |
| Epsilon                 | 0.05      |
| Episodes                | 8148      |
| AverageReturn           | 1         |
| AverageDiscountedReturn | 0.94721   |
| TDError^2               | 0.0011824 |
---------------------------------------
----------------------------------------
| Iteration               | 60         |
| Steps                   | 60999      |
| Epsilon                 | 0.05       |
| Episodes                | 8310       |
| AverageReturn           | 1          |
| AverageDiscountedReturn | 0.9491     |
| TDError^2               | 0.00082055 |
----------------------------------------
----------------------------------------
| Iteration               | 61         |
| Steps                   | 61999      |
| Epsilon                 | 0.05       |
| Episodes                | 8470       |
| AverageReturn           | 1          |
| AverageDiscountedReturn | 0.9491     |
| TDError^2               | 0.00048689 |
----------------------------------------
---------------------------------------
| Iteration               | 62        |
| Steps                   | 62999     |
| Epsilon                 | 0.05      |
| Episodes                | 8633      |
| AverageReturn           | 1         |
| AverageDiscountedReturn | 0.94721   |
| TDError^2               | 0.0001764 |
---------------------------------------
----------------------------------------
| Iteration               | 63         |
| Steps                   | 63999      |
| Epsilon                 | 0.05       |
| Episodes                | 8793       |
| AverageReturn           | 1          |
| AverageDiscountedReturn | 0.9491     |
```

```
| TDError^2                | 0.00013961 |
----------------------------------------
----------------------------------------
| Iteration                | 64         |
| Steps                    | 64999      |
| Epsilon                  | 0.05       |
| Episodes                 | 8928       |
| AverageReturn            | 1          |
| AverageDiscountedReturn  | 0.95099    |
| TDError^2                | 0.00049801 |
----------------------------------------
--------------------------------------
| Iteration                | 65       |
| Steps                    | 65999    |
| Epsilon                  | 0.05     |
| Episodes                 | 9064     |
| AverageReturn            | 1        |
| AverageDiscountedReturn  | 0.94721  |
| TDError^2                | 0.001038 |
--------------------------------------
----------------------------------------
| Iteration                | 66         |
| Steps                    | 66999      |
| Epsilon                  | 0.05       |
| Episodes                 | 9221       |
| AverageReturn            | 1          |
| AverageDiscountedReturn  | 0.9491     |
| TDError^2                | 0.00043546 |
----------------------------------------
----------------------------------------
| Iteration                | 67         |
| Steps                    | 67999      |
| Epsilon                  | 0.05       |
| Episodes                 | 9384       |
| AverageReturn            | 1          |
| AverageDiscountedReturn  | 0.94629    |
| TDError^2                | 0.00076797 |
----------------------------------------
---------------------------------------
| Iteration                | 68        |
| Steps                    | 68999     |
| Epsilon                  | 0.05      |
| Episodes                 | 9542      |
| AverageReturn            | 1         |
| AverageDiscountedReturn  | 0.94721   |
| TDError^2                | 0.0010483 |
---------------------------------------
----------------------------------------
| Iteration                | 69         |
| Steps                    | 69999      |
| Epsilon                  | 0.05       |
| Episodes                 | 9704       |
| AverageReturn            | 1          |

| AverageDiscountedReturn  | 0.94909    |
| TDError^2                | 0.00079955 |
----------------------------------------
----------------------------------------
| Iteration                | 70         |
| Steps                    | 70999      |
| Epsilon                  | 0.05       |
| Episodes                 | 9863       |
| AverageReturn            | 1          |
| AverageDiscountedReturn  | 0.9491     |
| TDError^2                | 0.00034144 |
----------------------------------------
---------------------------------------
```

```
| Iteration                 | 71        |
| Steps                     | 71999     |
| Epsilon                   | 0.05      |
| Episodes                  | 10005     |
| AverageReturn             | 1         |
| AverageDiscountedReturn   | 0.94435   |
| TDError^2                 | 0.0016721 |
---------------------------------------
----------------------------------------
| Iteration                 | 72         |
| Steps                     | 72999      |
| Epsilon                   | 0.05       |
| Episodes                  | 10163      |
| AverageReturn             | 1          |
| AverageDiscountedReturn   | 0.9491     |
| TDError^2                 | 0.00084058 |
----------------------------------------
----------------------------------------
| Iteration                 | 73         |
| Steps                     | 73999      |
| Epsilon                   | 0.05       |
| Episodes                  | 10322      |
| AverageReturn             | 1          |
| AverageDiscountedReturn   | 0.9491     |
| TDError^2                 | 0.00092193 |
----------------------------------------
--------------------------------------
| Iteration                 | 74       |
| Steps                     | 74999    |
| Epsilon                   | 0.05     |
| Episodes                  | 10457    |
| AverageReturn             | 0.5      |
| AverageDiscountedReturn   | 0.46795  |
| TDError^2                 | 0.001017 |
--------------------------------------
---------------------------------------
| Iteration                 | 75        |
| Steps                     | 75999     |
| Epsilon                   | 0.05      |
| Episodes                  | 10581     |

| AverageReturn             | 1         |
| AverageDiscountedReturn   | 0.95099   |
| TDError^2                 | 0.0026338 |
---------------------------------------
----------------------------------------
| Iteration                 | 76         |
| Steps                     | 76999      |
| Epsilon                   | 0.05       |
| Episodes                  | 10741      |
| AverageReturn             | 1          |
| AverageDiscountedReturn   | 0.9491     |
| TDError^2                 | 0.00078881 |
----------------------------------------
----------------------------------------
| Iteration                 | 77         |
| Steps                     | 77999      |
| Epsilon                   | 0.05       |
| Episodes                  | 10898      |
| AverageReturn             | 0.9        |
| AverageDiscountedReturn   | 0.854      |
| TDError^2                 | 0.00062791 |
----------------------------------------
----------------------------------------
| Iteration                 | 78         |
| Steps                     | 78999      |
| Epsilon                   | 0.05       |
| Episodes                  | 11004      |
```

```
| AverageReturn              | 0.9        |
| AverageDiscountedReturn    | 0.85025    |
| TDError^2                  | 0.00094903 |
---------------------------------------
-------------------------------------
| Iteration                  | 79       |
| Steps                      | 79999    |
| Epsilon                    | 0.05     |
| Episodes                   | 11164    |
| AverageReturn              | 1        |
| AverageDiscountedReturn    | 0.95099  |
| TDError^2                  | 0.001735 |
-------------------------------------
--------------------------------------
| Iteration                  | 80        |
| Steps                      | 80999     |
| Epsilon                    | 0.05      |
| Episodes                   | 11324     |
| AverageReturn              | 1         |
| AverageDiscountedReturn    | 0.95004   |
| TDError^2                  | 0.0018957 |
--------------------------------------
--------------------------------------
| Iteration                  | 81        |
| Steps                      | 81999     |
| Epsilon                    | 0.05      |

| Episodes                   | 11485     |
| AverageReturn              | 1         |
| AverageDiscountedReturn    | 0.95099   |
| TDError^2                  | 0.0008685 |
--------------------------------------
--------------------------------------
| Iteration                  | 82        |
| Steps                      | 82999     |
| Epsilon                    | 0.05      |
| Episodes                   | 11645     |
| AverageReturn              | 1         |
| AverageDiscountedReturn    | 0.94815   |
| TDError^2                  | 0.0014358 |
--------------------------------------
---------------------------------------
| Iteration                  | 83         |
| Steps                      | 83999      |
| Epsilon                    | 0.05       |
| Episodes                   | 11801      |
| AverageReturn              | 0.9        |
| AverageDiscountedReturn    | 0.854      |
| TDError^2                  | 0.00065436 |
---------------------------------------
--------------------------------------
| Iteration                  | 84        |
| Steps                      | 84999     |
| Epsilon                    | 0.05      |
| Episodes                   | 11961     |
| AverageReturn              | 1         |
| AverageDiscountedReturn    | 0.94721   |
| TDError^2                  | 0.0014733 |
--------------------------------------
---------------------------------------
| Iteration                  | 85         |
| Steps                      | 85999      |
| Epsilon                    | 0.05       |
| Episodes                   | 12118      |
| AverageReturn              | 0.9        |
| AverageDiscountedReturn    | 0.854      |
| TDError^2                  | 0.00053405 |
```

```
----------------------------------------
--------------------------------------
| Iteration               | 86       |
| Steps                   | 86999    |
| Epsilon                 | 0.05     |
| Episodes                | 12280    |
| AverageReturn           | 1        |
| AverageDiscountedReturn | 0.95004  |
| TDError^2               | 0.001025 |
--------------------------------------
---------------------------------------
| Iteration               | 87        |
| Steps                   | 87999     |

| Epsilon                 | 0.05      |
| Episodes                | 12439     |
| AverageReturn           | 1         |
| AverageDiscountedReturn | 0.94531   |
| TDError^2               | 0.0010579 |
---------------------------------------
----------------------------------------
| Iteration               | 88         |
| Steps                   | 88999      |
| Epsilon                 | 0.05       |
| Episodes                | 12574      |
| AverageReturn           | 0.6        |
| AverageDiscountedReturn | 0.55076    |
| TDError^2               | 0.00051726 |
----------------------------------------
----------------------------------------
| Iteration               | 89         |
| Steps                   | 89999      |
| Epsilon                 | 0.05       |
| Episodes                | 12707      |
| AverageReturn           | 1          |
| AverageDiscountedReturn | 0.95004    |
| TDError^2               | 0.00063419 |
----------------------------------------
---------------------------------------
| Iteration               | 90        |
| Steps                   | 90999     |
| Epsilon                 | 0.05      |
| Episodes                | 12813     |
| AverageReturn           | 1         |
| AverageDiscountedReturn | 0.9491    |
| TDError^2               | 0.0027217 |
---------------------------------------
---------------------------------------
| Iteration               | 91        |
| Steps                   | 91999     |
| Epsilon                 | 0.05      |
| Episodes                | 12973     |
| AverageReturn           | 1         |
| AverageDiscountedReturn | 0.9491    |
| TDError^2               | 0.0024985 |
---------------------------------------
---------------------------------------
| Iteration               | 92        |
| Steps                   | 92999     |
| Epsilon                 | 0.05      |
| Episodes                | 13132     |
| AverageReturn           | 1         |
| AverageDiscountedReturn | 0.94438   |
| TDError^2               | 0.0029096 |
---------------------------------------
---------------------------------------
| Iteration               | 93        |
```

```
| Steps                     | 93999     |
| Epsilon                   | 0.05      |
| Episodes                  | 13289     |
| AverageReturn             | 0.9       |
| AverageDiscountedReturn   | 0.854     |
| TDError^2                 | 0.0011467 |
---------------------------------------
---------------------------------------
| Iteration                 | 94        |
| Steps                     | 94999     |
| Epsilon                   | 0.05      |
| Episodes                  | 13448     |
| AverageReturn             | 1         |
| AverageDiscountedReturn   | 0.94721   |
| TDError^2                 | 0.0024111 |
---------------------------------------
---------------------------------------
| Iteration                 | 95        |
| Steps                     | 95999     |
| Epsilon                   | 0.05      |
| Episodes                  | 13607     |
| AverageReturn             | 1         |
| AverageDiscountedReturn   | 0.95099   |
| TDError^2                 | 0.0025553 |
---------------------------------------
---------------------------------------
| Iteration                 | 96        |
| Steps                     | 96999     |
| Epsilon                   | 0.05      |
| Episodes                  | 13766     |
| AverageReturn             | 1         |
| AverageDiscountedReturn   | 0.94341   |
| TDError^2                 | 0.0021854 |
---------------------------------------
---------------------------------------
| Iteration                 | 97        |
| Steps                     | 97999     |
| Epsilon                   | 0.05      |
| Episodes                  | 13926     |
| AverageReturn             | 0.9       |
| AverageDiscountedReturn   | 0.84465   |
| TDError^2                 | 0.0014632 |
---------------------------------------
--------------------------------------
| Iteration                 | 98       |
| Steps                     | 98999    |
| Epsilon                   | 0.05     |
| Episodes                  | 14036    |
| AverageReturn             | 1        |
| AverageDiscountedReturn   | 0.95099  |
| TDError^2                 | 0.001324 |
--------------------------------------
---------------------------------------

| Iteration                 | 99        |
| Steps                     | 99999     |
| Epsilon                   | 0.05      |
| Episodes                  | 14195     |
| AverageReturn             | 1         |
| AverageDiscountedReturn   | 0.9491    |
| TDError^2                 | 0.0010862 |
---------------------------------------
```

In [6]:

```
# visualize learned policy
dqn_gridworld.test(epsilon=0.0)
```

```
'NN_linear' object has no attribute 'set_params'
```

```
  (Down)
SFFF
FFFH
FFFF
HFFG
  (Right)
SFFF
FFFH
FFFF
HFFG
  (Right)
SFFF
FFFH
FFFF
HFFG
  (Down)
SFFF
FFFH
FFFF
HFFG
  (Right)
SFFF
FFFH
FFFF
HFFG
  (Down)
SFFF
FFFH
FFFF
HFFG
Done!
  (Down)
SFFF
FFFH
FFFF
HFFG
  (Right)
SFFF
FFFH
FFFF

HFFG
  (Right)
SFFF
FFFH
FFFF
HFFG
  (Down)
SFFF
FFFH
FFFF
HFFG
  (Right)
SFFF
FFFH
FFFF
HFFG
  (Down)
SFFF
FFFH
FFFF
HFFG
Done!
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-6-a2aac5f29ef0> in <module>()
      1 # visualize learned policy
```

```
----> 2 dqn_gridworld.test(epsilon=0.0)

<ipython-input-3-b26c52e70a62> in test(self, epsilon)
    171                 obs = self._env.reset()
    172                 print('Done!')
--> 173                 time.sleep(1)
    174             else:
    175                 obs = obs_prime

KeyboardInterrupt:
```

In [7]:
```
env.close()
```

4. Test algorithm on Pong

Now we can train for longer on a substantially more complex environment: Pong from the Atari suite. To speed up training, instead of playing from pixels we will be playing directly from the ram state.

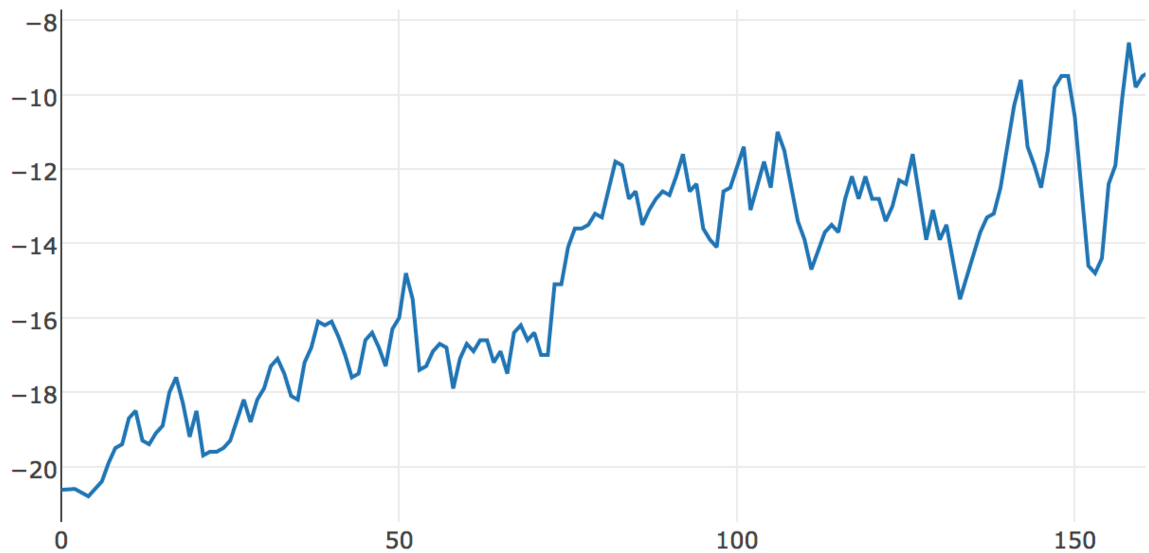
In [8]:
```
env = EpisodicLifeEnv(NoopResetEnv(gym.make('Pong-ram-v0')))
log_dir = "data/local/dqn_pong"

logger.session(log_dir).__enter__()
env.seed(42)

# Initialize the replay buffer that we will use.
replay_buffer = ReplayBuffer(max_size=10000)

# Initialize DQN training procedure.
dqn_pong = DQN(
    env=env,
    obs_dim=env.observation_space.shape[0],
    act_dim=env.action_space.n,
    NN=NN,
    obs_preprocessor=preprocess_obs_ram,
    replay_buffer=replay_buffer,
    opt_batch_size=64,
    # DQN gamma parameter
    discount=0.99,
    # Training procedure length
    initial_step=1000000,
    max_steps=10000000,
    learning_start_itr=100000,
    # Frequency of copying the actual Q to the target Q
    target_q_update_freq=1000,
    # Frequency of updating the Q-value function
    train_q_freq=4,
    # Exploration parameters
    initial_eps=1.0,
    final_eps=0.05,
    fraction_eps=0.1,
    # Logging
    log_freq=10000,
    render=False,
)
```

```
[2018-04-23 22:33:35,997] Making new env: Pong-ram-v0
```

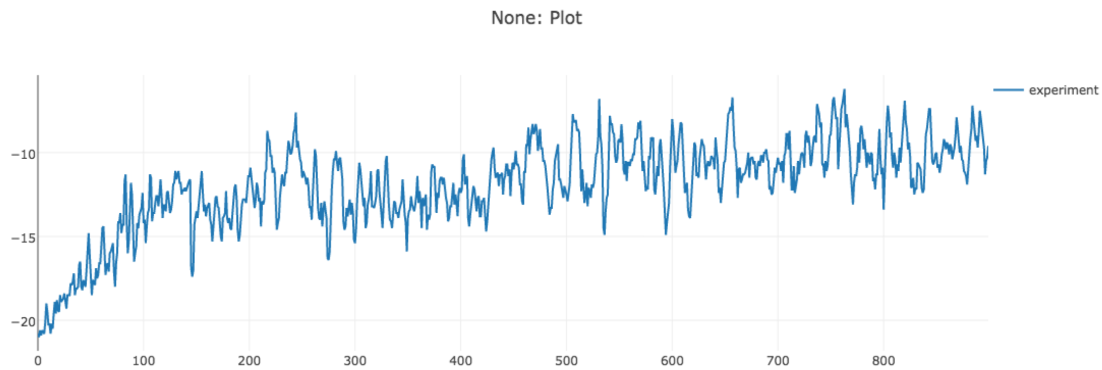
In [ ]:
```
dqn_pong.train()
```

Visualization

To visualize your learning curves, you can use the `viskit` tool by calling in a terminal: `python viskit/frontend.py path/to/log_dir` where `path/to/log_dir` is by default `data/local/exp_name`, where `exp_name` is `dqn_pong` in the case of pong for example. For this visualization to work you need to have the path to the homework directory to be added to your `$PYTHONPATH`. You should then see in your browser something like this:



Student Visualization Result



Policy Gradient methods

We will start with the standard policy gradient algorithm. This is a batch algorithm, which means that we will collect a large number of samples per iteration, and perform a single update to the policy using these samples. Recall that the formula for policy gradient is given by

$$\nabla_\theta \mathbb{E}_{\pi_\theta}\Big[\sum_{t=0}^{T}\gamma^t r_t\Big] = \mathbb{E}_{\pi_\theta}\Big[\sum_{t=0}^{T}\nabla_\theta \log \pi_\theta(a_t|s_t)\big(R_t - b(s_t)\big)\Big] \tag{1}$$

- π_θ is a stochastic policy parameterized by θ;
- γ is the discount factor;
- s_t, a_t and r_t are the state, action, and reward at time t;
- T is the length of a single episode;
- $b(s_t)$ is any funcion which does not depend on the current action a_t, and is called baseline;
- R_t is the discounted cumulative future return (already defined in the DQN exercise); Instead of optimizing this formula, we will optimize a sample-based estimation of the expectation, based on N trajectories. For this you will first implement a function that computes $\log \pi_\theta(a_t|s_t)$ given any $s,\ a$.

In [1]:
```
#!/usr/bin/env python
import numpy as np
import gym
from simplepg.simple_utils import gradient_check, log_softmax, softmax, weighted_sample, includ
e_bias, test_once, nprs
import tests.simplepg_tests
import math
```

Constructing a stochastic policy

Let's assume that π_θ is a Gaussian with unit variance $\Sigma = I$ and mean $\mu = NN_\theta(s)$, where NN_θ is a Neural Network parameterized by θ.

1. Create a Linear NN

Use two hidden linear layer with 256 hidden units and ReLu non-linearity, and use a linear output layer with no output non-linearity.

In [2]:
```
import torch
import torch.nn as nn
import torch.optim as optim
import torch.nn.functional as F
import torch.autograd as autograd

class MLP(nn.Module):
    def __init__(self, obs_size, act_size):
        super(MLP, self).__init__()
        #"*** YOUR CODE HERE ***"
        self.linear1 = nn.Linear(obs_size, 256)
        self.linear2 = nn.Linear(256,256)
        self.linear3 = nn.Linear(256, act_size)
    def forward(self, obs):
        #"*** YOUR CODE HERE ***"
        out = F.relu(self.linear1(obs))
        out = F.relu(self.linear2(out))
        out = self.linear3(out)
        return out
```

2. Create a Gaussian MLP policy

For Policy Gradient methods the policy needs to be stochastic. In our case, we will assume the distribution is a Gaussian where the mean $\mu_\theta(o)$ is the output of an MLP given the observation o and unit variance. You will need to implement the `get_action` method that, given an observation o, samples an action a from $\mathcal{N}(\mu_\theta(o), I)$; and the `get_logp_action` that gives the logprobability of a given action a under the policy when observation o is inputed. Remember the probability of a n-dimensional multivariate Gaussian with unit variance can be written as:

$$\frac{1}{\sqrt{(2\pi)^n}}\exp^{-\frac{1}{2}(a-\mu_\theta(o))^T(a-\mu_\theta(o))}$$

In [3]:
```
class GaussianMLP_Policy(object):
    def __init__(self, obs_size, act_size, NN):
        self.NN = NN(obs_size, act_size)

    def get_action(self, obs, rng=np.random):
        #"*** YOUR CODE HERE ***"
        obs_var = autograd.Variable(torch.from_numpy(obs).float(), requires_grad=False)
        mean = self.NN(obs_var).data.numpy()
        cov = np.identity(mean.shape[0])
        sampled_action = rng.multivariate_normal(mean, cov)
        return sampled_action

    def get_logp_action(self, obs, action):
        #"*** YOUR CODE HERE ***"
        # obs: Variable
        # action: Variable
        # log_p: Variable
        mean_var = self.NN(obs)
        n = action.size(1)
        diff = action - mean_var
        power = -0.5 * torch.sum(torch.pow(diff ,2.0) ,1)
        log_p = torch.log(torch.exp(power) / math.sqrt(pow(2*math.pi, float(n))))
        return log_p
```

3. Compute time-based baselines

Any function that does not depend on the action can be used as a baseline. The most usual one is to have a state-based baseline. In our case we will keep a simple time-based baseline that is the average return obtained at that particular time-step accross all paths collected in the previous iteration.

In [4]:
```
def compute_baselines(all_returns):
    baselines = np.zeros(len(all_returns))
    for t in range(len(all_returns)):
        #"*** YOUR CODE HERE ***"
        # Update the baselines
        # all_returns: list of lists. all_returns[time_step][path]
        baselines[t] = np.mean(all_returns[t]) if len(all_returns[t])>0 else 0
    return baselines
```

4. Compute returns

Given the rewards obtained in a path, return the discounted returns with the formula:

$$R_t = \begin{cases} r_t + \gamma R_{t+1} & \text{if non-terminal transition} \\ r_t & \text{for terminal transition} \end{cases} \qquad (2)$$

In [5]:
```
def compute_returns(discount, rewards):
    returns = np.zeros_like(rewards)
    #"*** YOUR CODE HERE ***"
    for i in np.arange(len(returns)-1,-1,-1):
        returns[i] = rewards[i] + (0 if i == len(returns)-1 else discount*returns[i+1])
    return returns
```

Run the algorithm

You are only asked to implement the surrogate reward that we take the gradient of. We do so by approximating the expectation in Eq. (1) by a sum over paths. In other words, the surrogate function can be written as:

$$\sum_{i=0}^{N} \sum_{t=0}^{T_i} \log \pi_\theta(a_t^i | s_t^i) \Big(R_t^i - b(t) \Big) \qquad (3)$$

If you implemented it correctly, the reward should reach arround -20 in about 50 iterations.

In [6]:

```
from simplepg import point_env
env = gym.make('Point-v0')
obs_dim = env.observation_space.shape[0]
action_dim = env.action_space.shape[0]

# Store baselines for each time step.
timestep_limit = env.spec.timestep_limit
baselines = np.zeros(timestep_limit)

# instantiate the policy
policy = GaussianMLP_Policy(obs_dim, action_dim, MLP)
```

```
[2018-04-23 22:22:39,338] Making new env: Point-v0
```

In [7]:
```
n_itrs = 50
batch_size = 2000
discount = 0.99
learning_rate = 0.1
render = False  # True  # see setup_instructions.pdf to render point-mass policy
natural_step_size = 0.01

# Policy training loop
for itr in range(n_itrs):
    # Collect trajectory loop
    n_samples = 0
    policy.NN.zero_grad()
    episode_rewards = []

    # Store cumulative returns for each time step
    all_returns = [[] for _ in range(timestep_limit)]

    all_observations = []
    all_actions = []
    all_centered_cum_rews = []

    while n_samples < batch_size:
        observations = []
        actions = []
        rewards = []
        ob = env.reset()
        done = False
        # Only render the first trajectory
        render_episode = n_samples == 0
        # Collect a new trajectory
        while not done:
            action = policy.get_action(ob)
            next_ob, rew, done, _ = env.step(action)
            observations.append(ob)
            actions.append(action)
            rewards.append(rew)
            ob = next_ob
            n_samples += 1
            if render and render_episode:
                env.render()

        # Go back in time to compute returns
        returns = compute_returns(discount, rewards)
        # center the rewards by substracting the baseline
        centered_cum_rews = returns - baselines[:len(returns)]
        # save them in all_returns to compute time-based baseline for next iteration
        for t, r in enumerate(returns):
            all_returns[t].append(r)

        episode_rewards.append(np.sum(rewards))
        all_observations.extend(observations)
        all_actions.extend(actions)
        all_centered_cum_rews.extend(centered_cum_rews)

    # autodiff loss
    obs_vars = autograd.Variable(torch.Tensor(all_observations), requires_grad=False)
    act_vars = autograd.Variable(torch.Tensor(all_actions), requires_grad=False)
    centered_cum_rews_vars = autograd.Variable(torch.Tensor(all_centered_cum_rews), requires_gr
ad=False)

    logps = policy.get_logp_action(obs_vars, act_vars)

    #"*** YOUR CODE HERE ***"
    surr_loss = torch.sum(logps * centered_cum_rews_vars)

    surr_loss.backward()

    flat_grad = np.concatenate([p.grad.data.numpy().reshape((-1)) for p in policy.NN.parameters
()])
    grad_norm = np.linalg.norm(flat_grad)
```

```python
    for p in policy.NN.parameters():
        # roughly normalize gradiend and take step
        p.data += learning_rate * p.grad.data / (grad_norm + 1e-8)

    test_once(compute_baselines)

    baselines = compute_baselines(all_returns)

    print("Iteration: %d AverageReturn: %.2f GradNorm: %.2f" % (
    itr, np.mean(episode_rewards), grad_norm))
```

```
Test for __main__.compute_baselines passed!
Iteration: 0 AverageReturn: -41.47 GradNorm: 4683.90
Iteration: 1 AverageReturn: -39.78 GradNorm: 1197.01
Iteration: 2 AverageReturn: -39.59 GradNorm: 887.91
Iteration: 3 AverageReturn: -36.59 GradNorm: 1195.10
Iteration: 4 AverageReturn: -36.08 GradNorm: 997.15
Iteration: 5 AverageReturn: -34.29 GradNorm: 1128.68
Iteration: 6 AverageReturn: -32.95 GradNorm: 1409.37
Iteration: 7 AverageReturn: -32.17 GradNorm: 1654.11
Iteration: 8 AverageReturn: -31.85 GradNorm: 2034.69
Iteration: 9 AverageReturn: -28.44 GradNorm: 1100.82
Iteration: 10 AverageReturn: -28.17 GradNorm: 1134.04
Iteration: 11 AverageReturn: -27.77 GradNorm: 1483.22
Iteration: 12 AverageReturn: -26.80 GradNorm: 839.32
Iteration: 13 AverageReturn: -25.60 GradNorm: 1296.70
Iteration: 14 AverageReturn: -23.46 GradNorm: 748.20
Iteration: 15 AverageReturn: -24.82 GradNorm: 1173.66
Iteration: 16 AverageReturn: -23.46 GradNorm: 786.92
Iteration: 17 AverageReturn: -22.39 GradNorm: 878.82
Iteration: 18 AverageReturn: -23.22 GradNorm: 1867.40
Iteration: 19 AverageReturn: -22.28 GradNorm: 518.45
Iteration: 20 AverageReturn: -21.29 GradNorm: 606.31
Iteration: 21 AverageReturn: -21.73 GradNorm: 1140.74
Iteration: 22 AverageReturn: -21.17 GradNorm: 1414.94
Iteration: 23 AverageReturn: -21.33 GradNorm: 656.82
Iteration: 24 AverageReturn: -20.71 GradNorm: 880.13
Iteration: 25 AverageReturn: -20.56 GradNorm: 614.01
Iteration: 26 AverageReturn: -20.91 GradNorm: 1366.66
Iteration: 27 AverageReturn: -20.98 GradNorm: 1031.35
Iteration: 28 AverageReturn: -20.54 GradNorm: 1023.36
Iteration: 29 AverageReturn: -19.97 GradNorm: 1108.04
Iteration: 30 AverageReturn: -19.54 GradNorm: 297.48
Iteration: 31 AverageReturn: -19.18 GradNorm: 594.79
Iteration: 32 AverageReturn: -19.92 GradNorm: 847.16
Iteration: 33 AverageReturn: -20.16 GradNorm: 1158.32
Iteration: 34 AverageReturn: -20.02 GradNorm: 434.58
Iteration: 35 AverageReturn: -20.04 GradNorm: 525.81
Iteration: 36 AverageReturn: -20.78 GradNorm: 873.64
Iteration: 37 AverageReturn: -19.45 GradNorm: 1104.83
Iteration: 38 AverageReturn: -19.80 GradNorm: 723.37
Iteration: 39 AverageReturn: -18.98 GradNorm: 817.94
Iteration: 40 AverageReturn: -19.28 GradNorm: 795.82
Iteration: 41 AverageReturn: -18.59 GradNorm: 755.74
Iteration: 42 AverageReturn: -19.32 GradNorm: 576.81
Iteration: 43 AverageReturn: -19.43 GradNorm: 633.93
Iteration: 44 AverageReturn: -18.78 GradNorm: 748.62
Iteration: 45 AverageReturn: -18.50 GradNorm: 585.67
Iteration: 46 AverageReturn: -19.06 GradNorm: 1132.68
Iteration: 47 AverageReturn: -19.14 GradNorm: 565.58
Iteration: 48 AverageReturn: -19.11 GradNorm: 1300.42
Iteration: 49 AverageReturn: -18.65 GradNorm: 942.61
```