CS7643: Deep Learning
Fall 2019
HW4 Solutions

Nicolas Six

November 8, 2019

1 Optimal Policy and Value Function

1.1 Always stay policy

In this context:

$$r_t(S_1, \text{"stay"}) = -1$$
$$r_t(S_2, \text{"stay"}) = -1$$

In other word, no matter the first state, we can simplify the equation in the following way:

$$\begin{aligned}\sum_{t=0}^{\infty} \gamma^t r_t(s_t, a_t) &= \sum_{t=0}^{\infty} -\gamma^t \\ &= -lim_{t\to\infty}\left(\frac{\gamma^t - 1}{\gamma - 1}\right) \text{ if } \gamma \neq 1\end{aligned}$$

In this question γ is described as a discount, which is usually so $\gamma \in [0, 1[$. If it's the case you can directly look in Section 1.1.1 for the result using γ in the classical range. But this range is not precised on the question we will so explore all the different cases.

1.1.1 $\gamma \in]-1, 1[$

$$\sum_{t=0}^{\infty} \gamma^t r_t(s_t, a_t) = \frac{1}{\gamma - 1}$$

1.1.2 $\gamma = 1$

$$\begin{aligned}\sum_{t=0}^{\infty} \gamma^t r_t(s_t, a_t) &= \sum_{t=0}^{\infty} -1^t \\ &= -\infty\end{aligned}$$

1.1.3 $\gamma \in]1, +\infty[$

$$\begin{aligned}\sum_{t=0}^{\infty} \gamma^t r_t(s_t, a_t) &= -lim_{t\rightarrow\infty}\left(\frac{\gamma^t - 1}{\gamma - 1}\right)\\ &= -\infty\end{aligned}$$

1.1.4 $\gamma = -1$

In this case $\sum_{t=0}^{\infty} \gamma^t r_t(s_t, a_t)$ does not converge and just alternate between 0 and 1.

1.1.5 $\gamma \in] - \infty, -1[$

$$\begin{aligned}\sum_{t=0}^{\infty} -\gamma^t &= -\sum_{t=0}^{\infty} \gamma^{2t} + \gamma^{2t+1}\\ &= -lim_{t\rightarrow\infty}\left(\frac{\gamma^{2^t} - 1}{\gamma^2 - 1} + \gamma\frac{\gamma^{2^t} - 1}{\gamma^2 - 1}\right)\\ &= -lim_{t\rightarrow\infty}\left((\gamma + 1)\frac{\gamma^{2^t} - 1}{\gamma^2 - 1}\right)\\ &= +\infty\end{aligned}$$

1.2 Optimal policy

The optimal policy is (”*go*”, ”*go*”). Lets name this policy π^*. Now we can compute the value of each state following this policy.

$$\begin{aligned} V^*(S_2) &= r(S_2, ”go”) \\ &= 3 \\ V^*(S_1) &= r(S_1, ”go”) + \gamma V^*(S_2) \\ &= -2 + 3\gamma \end{aligned}$$

1.3 Value function

$$
\begin{aligned}
V^0 &= [0, 0] \\
V^1 &= [-1, 3] \\
V^2 &= [1, 3] \\
V^3 &= [1, 3] \\
V^* &= [1, 3]
\end{aligned}
$$

2 Value Iteration Convergence

2.1 Error decrease

$$\begin{aligned}
\left\|V^0 - V^*\right\|_\infty &= 3 \\
\left\|V^1 - V^*\right\|_\infty &= 2 \\
\left\|V^2 - V^*\right\|_\infty &= 0 \\
\left\|V^3 - V^*\right\|_\infty &= 0
\end{aligned}$$

We can see that the error of the value iteration is decreasing monotonically in this example.

2.2 Proof of decrease over iterations

$$\begin{aligned}
\left\|T(V) - T(V')\right\|_\infty &= \left\|max_a \sum_{s'} p(s'|s,a)[r(s,a)+\gamma V] - max_a \sum_{s'} p(s'|s,a)[r(s,a)+\gamma V']\right\|_\infty \\
&\leq \left\|max_a \left(\sum_{s'} p(s'|s,a)[r(s,a)+\gamma V] - \sum_{s'} p(s'|s,a)[r(s,a)+\gamma V']\right)\right\|_\infty \\
&\leq \left\|max_a \left(\sum_{s'} \big(p(s'|s,a)[r(s,a)+\gamma V] - p(s'|s,a)[r(s,a)+\gamma V']\big)\right)\right\|_\infty \\
&\leq \left\|max_a \left(\sum_{s'} \big(p(s'|s,a)[r(s,a)+\gamma V - r(s,a)+\gamma V']\big)\right)\right\|_\infty \\
&\leq \left\|max_a \left(\sum_{s'} \big(p(s'|s,a)[\gamma V - \gamma V']\big)\right)\right\|_\infty \\
&\leq \gamma \left\|max_a \left(\sum_{s'} \big(p(s'|s,a)[V - V']\big)\right)\right\|_\infty \\
&\leq \gamma \left\|max_a \big(V - V'\big)\right\|_\infty \text{ as } \sum_{s'} p(s'|s,a) = 1 \\
&\leq \gamma \left\|V - V'\right\|_\infty
\end{aligned}$$

2.3 Proof of bound

Starting with the previous results, we have:

$$\begin{aligned}
\left\|T(V^n) - T(V^{n+1})\right\|_\infty &\leq \gamma \left\|V^n - V^{n+1}\right\|_\infty \\
\Leftrightarrow \left\|V^{n+1} - V^{n+2}\right\|_\infty &\leq \gamma \left\|V^n - V^{n+1}\right\|_\infty \\
\Leftrightarrow \left\|V^{n+1} - V^*\right\|_\infty - \left\|V^{n+2} - V^*\right\|_\infty &\leq \gamma \left\|V^n - V^{n+1}\right\|_\infty \\
\Leftrightarrow \left\|V^{n+1} - V^*\right\|_\infty - \left\|T(V^{n+1}) - T(V^*)\right\|_\infty &\leq \gamma \left\|V^n - V^{n+1}\right\|_\infty \\
\Leftrightarrow \left\|V^{n+1} - V^*\right\|_\infty - \gamma \left\|V^{n+1} - V^*\right\|_\infty &\leq \gamma \left\|V^n - V^{n+1}\right\|_\infty \\
\Leftrightarrow (1-\gamma) \left\|V^{n+1} - V^*\right\|_\infty &\leq \gamma \left\|V^n - V^{n+1}\right\|_\infty \\
\Leftrightarrow \left\|V^{n+1} - V^*\right\|_\infty &\leq \frac{\gamma}{1-\gamma} \left\|V^n - V^{n+1}\right\|_\infty \\
\Leftrightarrow \left\|V^{n+1} - V^*\right\|_\infty &\leq \frac{\gamma}{1-\gamma} \epsilon \text{ for } n > N
\end{aligned}$$

2.4 Unique fixed point

$$\|T(x_1) - T(x_2)\|_\infty \leq \alpha \, \|x_1 - x_2\|_\infty \tag{1}$$

From equation 1 we know that T is continuous and monotonic. In addition this equation give us a bound of it's derivative, by definition $\frac{dT(x)}{dx_i} \in [-\alpha, \alpha]$ For all x_i dimension of x.
So the hyper surface designed by T must intersect with the hyper plan designed by the identity function, which is continuous, monotonic and increasing with slope $1 > \alpha$. So there must be at least one fixed point for T.
In addition, if there is two different fixed point, x^{*1} and x^{*2}, then:

$$\begin{aligned} \left\|T(x^{*1}) - T(x^{*2})\right\|_\infty &= \left\|x^{*1} - x^{*2}\right\|_\infty \\ &> \alpha \left\|x^{*1} - x^{*2}\right\|_\infty \text{ as } \alpha < 1 \text{ and } \left\|x^{*1} - x^{*2}\right\|_\infty \neq 0 \end{aligned}$$

So we can't have two distinct fixed points for T.
In conclusion, there is one unique fixed point where $T(x) = x$.

3 Learning the Model

3.1 Error bound

$$
\begin{aligned}
&\left\|V^{\pi}_{\hat{M}} - V^{\pi}_{M}\right\|_{\infty} = max_s\left|\left(V_{\hat{M}}(s) - V_M(s)\right)\right| \\
&= max_s\left|\left(max_a \sum_{s'} \hat{\mathbb{T}}(s,a)(s')\left(\hat{\mathcal{R}}(s,a) + \gamma V_{\hat{M}}(s')\right) - max_a \sum_{s'} \mathbb{T}(s,a)(s')\left(\mathcal{R}(s,a) + \gamma V_M(s')\right)\right)\right| \\
&\leq max_s max_a\left|\left(\sum_{s'} \hat{\mathbb{T}}(s,a)(s')\left(\hat{\mathcal{R}}(s,a) + \gamma V_{\hat{M}}(s')\right) - \sum_{s'} \mathbb{T}(s,a)(s')\left(\mathcal{R}(s,a) + \gamma V_M(s')\right)\right)\right| \\
&\leq max_{s,a}\left|\sum_{s'}\left(\hat{\mathbb{T}}(s,a)(s')\left(\hat{\mathcal{R}}(s,a) + \gamma V_{\hat{M}}(s')\right) - \mathbb{T}(s,a)(s')\left(\mathcal{R}(s,a) + \gamma V_M(s')\right)\right)\right| \\
&\leq max_{s,a}\left|\sum_{s'}\left(\hat{\mathbb{T}}(s,a)(s')\hat{\mathcal{R}}(s,a) + \hat{\mathbb{T}}(s,a)(s')\gamma V_{\hat{M}}(s') - \mathbb{T}(s,a)(s')\mathcal{R}(s,a) - \mathbb{T}(s,a)(s')\gamma V_M(s')\right)\right| \\
&\leq max_{s,a}\left|\left\|\hat{\mathbb{T}}(s,a)\right\|_1 \hat{\mathcal{R}}(s,a) + \gamma\left\|\hat{\mathbb{T}}(s,a)V_{\hat{M}}\right\|_1 - \left\|\mathbb{T}(s,a)\right\|_1 \mathcal{R}(s,a) - \gamma\left\|\mathbb{T}(s,a)V_M\right\|_1\right| \\
&\leq max_{s,a}\left|\left\|\hat{\mathbb{T}}(s,a)\right\|_1 \hat{\mathcal{R}}(s,a) + \gamma\left\|\hat{\mathbb{T}}(s,a)\right\|_1\left\|V_{\hat{M}}\right\|_{\infty} - \left\|\mathbb{T}(s,a)\right\|_1 \mathcal{R}(s,a) - \gamma\left\|\mathbb{T}(s,a)\right\|_1\left\|V_M\right\|_{\infty}\right| \\
&\leq max_{s,a}\left|\left\|\hat{\mathbb{T}}(s,a)\right\|_1 \hat{\mathcal{R}}(s,a) + \gamma\left\|V_{\hat{M}}\right\|_{\infty} - \left\|\mathbb{T}(s,a)\right\|_1 \mathcal{R}(s,a) - \gamma\left\|V_M\right\|_{\infty}\right| \\
&\leq max_{s,a}\left|\left\|\hat{\mathbb{T}}(s,a)\right\|_1 \hat{\mathcal{R}}(s,a) - \left\|\mathbb{T}(s,a)\right\|_1 \mathcal{R}(s,a) + \gamma\left\|V_{\hat{M}} - V_M\right\|_{\infty}\right| \\
&\leq max_{s,a}\left|\left\|\hat{\mathbb{T}}(s,a)\right\|_1 \hat{\mathcal{R}}(s,a) - \left\|\mathbb{T}(s,a)\right\|_1 \mathcal{R}(s,a)\right| + \gamma\left\|V_{\hat{M}} - V_M\right\|_{\infty} \\
&\leq \frac{1}{1-\gamma} max_{s,a}\left|\left\|\hat{\mathbb{T}}(s,a)\right\|_1 \hat{\mathcal{R}}(s,a) - \left\|\mathbb{T}(s,a)\right\|_1 \mathcal{R}(s,a)\right| \\
&\leq \frac{1}{1-\gamma} max_{s,a}\left|\hat{\mathcal{R}}(s,a) - \mathcal{R}(s,a)\right| \text{ as } \left\|\mathbb{T}(s,a)\right\|_1 = 1 \\
&\leq \frac{\epsilon_R}{1-\gamma} \\
&\leq \frac{\epsilon_R}{1-\gamma} + \frac{\epsilon_P}{(1-\gamma)^2}
\end{aligned}
$$

3.2 Error of approximate policy on real word

3.3 devlopement

3.4 Expend ϵ_R and ϵ_R

4 Policy Gradients Variance Reduction

4.1 Gradient offset

$$\begin{aligned}\nabla_\theta \mathbb{E}_{\tau\sim\pi_\theta}[\mathcal{R}(\tau)+b] &= \nabla_\theta\left(\mathbb{E}_{\tau\sim\pi_\theta}[\mathcal{R}(\tau)]+b\right)\\ &= \nabla_\theta \mathbb{E}_{\tau\sim\pi_\theta}[\mathcal{R}(\tau)]+\nabla_\theta b\\ &= \nabla_\theta \mathbb{E}_{\tau\sim\pi_\theta}[\mathcal{R}(\tau)]\end{aligned}$$

In conclusion, changing the reward by a constant doesn't change the gradient of J.

4.2 Variance

With $J_b(\theta)$ the $J(\theta)$ function with $\mathcal{R}(\tau) := \mathcal{R}(\tau) + b$.

$$\begin{aligned}
Var(\nabla_\theta J_b(\theta)) &= Var([\mathcal{R}(\tau)+b]\nabla_\theta log\pi_\theta(\tau)) \\
&= \mathbb{E}\left(([\mathcal{R}(\tau)+b]\nabla_\theta log\pi_\theta(\tau))^2\right) - \mathbb{E}\left([\mathcal{R}(\tau)+b]\nabla_\theta log\pi_\theta(\tau)\right)^2 \\
&= \mathbb{E}\left(([\mathcal{R}(\tau)^2 + 2\mathcal{R}(\tau)b + b^2](\nabla_\theta log\pi_\theta(\tau))^2\right) - \left(\mathbb{E}\left(\mathcal{R}(\tau)\nabla_\theta log\pi_\theta(\tau)\right) + \mathbb{E}\left(b\nabla_\theta log\pi_\theta(\tau)\right)\right)^2 \\
&\text{with } f(\tau) = \nabla_\theta log\pi_\theta(\tau) \\
&= \mathbb{E}\left(\mathcal{R}(\tau)^2 f(\tau)^2\right) + 2b\mathbb{E}\left(\mathcal{R}(\tau)f(\tau)\right) + b^2\mathbb{E}\left(f(\tau)^2\right) \\
&\ - \mathbb{E}\left(\mathcal{R}(\tau)f(\tau)\right)^2 - 2\mathbb{E}\left(\mathcal{R}(\tau)f(\tau)\right)\mathbb{E}\left(bf(\tau)\right) - b^2\mathbb{E}\left(f(\tau)\right)^2 \\
&= Var(\nabla_\theta J(\theta)) + b^2 Var(f(\tau)) + 2b\mathbb{E}\left[\mathcal{R}(\tau)f(\tau)\right]\left(1 - \mathbb{E}\left[f(\tau)\right]\right)
\end{aligned}$$

We are looking for the point where:

$$\begin{aligned}
2bVar(f(\tau)) + 2\mathbb{E}\left[\mathcal{R}(\tau)f(\tau)\right]\left(1-\mathbb{E}\left[f(\tau)\right]\right) = 0 \\
\Leftrightarrow bVar(f(\tau)) = -\mathbb{E}\left[\mathcal{R}(\tau)f(\tau)\right]\left(1-\mathbb{E}\left[f(\tau)\right]\right) \\
\Leftrightarrow b = -\frac{\mathbb{E}\left[\mathcal{R}(\tau)f(\tau)\right]\left(1-\mathbb{E}\left[f(\tau)\right]\right)}{Var(f(\tau))}
\end{aligned}$$

For $Var(f(\tau)) \neq 0$.

In this case the variance is minimized when the rewards is being subtracted by $\frac{\mathbb{E}[\mathcal{R}(\tau)f(\tau)](1-\mathbb{E}[f(\tau)])}{Var(f(\tau))}$ Please note that as defined before, we use here $\mathcal{R}(\tau) := \mathcal{R}(\tau) + b$, and so have a sign difference with the equation proposed in the homework. This allowed to prevent useless sign error during the development.

In addition, we can note that this value is difficult to compute and is going to evolve during training. So it would be impossible to get the minimum variance possible during all the training with a constant b. However it shows that using an approximation of b will help to stabilise the training.

5 Coding: Dynamic Programming and Deep Q-Learning

5.1 Dynamic Programming

Dynamic Programming (20 points + 10 bonus points)

In this assignment, we will implement a few dynamic programming algorithms, namely, policy iteration and value iteration and run them on a simple MDP - the Frozen Lake environment.

The sub-routines for these algorithms are present in `vi_and_pi.py` and must be filled out to test your implementation.

The deliverables are located at the end of this notebook and show the point distrbution for each part.

Value iteration is worth 20 points of regular credit and policy iteration is worth 10 points of bonus credit for both sections of this course CS 7643 and CS 4803.

In [1]:

```
%load_ext autoreload
%autoreload 2

import numpy as np
import gym
import time

from IPython.display import clear_output

from lake_envs import *
from vi_and_pi import *

np.set_printoptions(precision=3)

env_d = gym.make("Deterministic-4x4-FrozenLake-v0")
env_s = gym.make("Stochastic-4x4-FrozenLake-v0")
```

```
/home/nicolas/.local/lib/python3.6/site-packages/gym/envs/registration.py:14: PkgResourcesDeprecationWarning: Parameters to load ar
e deprecated.  Call .resolve and .require separately.
  result = entry_point.load(False)
```

Render Mode

The variable `RENDER_ENV` is set `True` by default to allow you to see a rendering of the state of the environment at every time step. However, when you complete this assignment, you must set this to `False` and re-run all blocks of code. This is to prevent excessive amounts of rendered environments from being included in the final PDF.

IMPORTANT: SET `RENDER_ENV` TO FALSE BEFORE SUBMISSION!

In [2]:

```
RENDER_ENV = False
```

Part 1: Value Iteration

For the first part, you will implement the familiar value iteration update from class.

In `vi_and_pi.pi` and complete the `value_iteration` function.

In [3]:

```
###################################################
# Use this space for debugging                    #
# Make sure to delete this code before submission #
###################################################
pass
###################################################
```

Run the cell below to train value iteration and render a single episode of following the policy obtained at the end of value iteration.

You should expect to get an Episode reward of `1.0`.

In [4]:

```
print("\n" + "-"*25 + "\nBeginning Value Iteration\n" + "-"*25)

V_vi, p_vi = value_iteration(env_d.P, env_d.nS, env_d.nA, gamma=0.9, tol=1e-3)
render_single(env_d, p_vi, 100, show_rendering=RENDER_ENV)
```

```
-------------------------
Beginning Value Iteration
-------------------------
Episode reward: 1.000000
```

[BONUS] Part 2: Policy Iteration

This is a bonus question in which you will implement policy iteration. If you do not wish to attempt this bonus quesiton, skip to the next part.

In class, we studied the value iteration update:

$$V_{t+1}(s) \leftarrow \max_{a} \sum_{s'} p(s' \mid s, a)\Big[r(s, a) + \gamma V_t(s')\Big]$$

This is used to compute the value function V^* corresponding to the optimal policy π^*. We can alternatively compute the value function V^π corresponding to an arbitrary policy π, with a similar update loop:

$$V^\pi_{t+1}(s) \leftarrow \sum_{a} \pi(a \mid s) \sum_{s'} p(s' \mid s, a)\Big[r(s, a) + \gamma V^\pi_t(s')\Big]$$

On convergence, this will give us V^π, which is the first step of a policy iteration update.

The second step involves policy refinement, which will update the policy to take actions greedily with respect to V^π:

$$\pi_{new} \leftarrow \arg\max_{a} \left[r(s, a) + \gamma \sum_{s'} p(s' \mid s, a) V^\pi(s') \right]$$

A single update of policy iteration involves the two above steps: (1) policy evaluation (which itself is an inner loop which will converge to V^π and (2) policy refinement. In the first part of assignment, you will implement the functions for policy evaluation, policy improvement (refinement) and policy iteration.

In `vi_and_pi.pi` and complete the `policy_evaluation`, `policy_improvement` and `policy_iteration` functions. Run the blocks below to test your algorithm.

In [5]:

```
###################################################
# Use this space for debugging                    #
# Make sure to delete this code before submission #
###################################################
pass
###################################################
```

In [6]:

```
print("\n" + "-"*25 + "\nBeginning Policy Iteration\n" + "-"*25)

V_pi, p_pi = policy_iteration(env_d.P, env_d.nS, env_d.nA, gamma=0.9, tol=1e-3)
render_single(env_d, p_pi, 100, show_rendering=RENDER_ENV)
```

```
-------------------------
Beginning Policy Iteration
-------------------------
Episode reward: 1.000000
```

Part 3: VI on Stochastic Frozen Lake

Now we will apply our implementation on an MDP where transitions to next states are stochastic. Modify your implementation of value iteration as needed so that policy iteration and value iteration work for stochastic transitions.

In [7]:

```
###################################################
# Use this space for debugging                    #
# Make sure to delete this code before submission #
###################################################
pass
###################################################
```

In [8]:

```
print("\n" + "-"*25 + "\nBeginning Value Iteration\n" + "-"*25)

V_vi, p_vi = value_iteration(env_s.P, env_s.nS, env_s.nA, gamma=0.9, tol=1e-3)
render_single(env_s, p_vi, 100, show_rendering=RENDER_ENV)
```

```
-------------------------
Beginning Value Iteration
-------------------------
Episode reward: 1.000000
```

[BONUS] Part 4: PI on Stochastic Frozen Lake

This is a bonus question to run policy iteration on stochastic frozen lake.

Now we will apply our implementation on an MDP where transitions to next states are stochastic. Modify your implementation of value iteration as needed so that policy iteration and value iteration work for stochastic transitions.

In [9]:

```
###################################################
# Use this space for debugging                    #
# Make sure to delete this code before submission #
###################################################
pass
###################################################
```

In [10]:

```
print("\n" + "-"*25 + "\nBeginning Policy Iteration\n" + "-"*25)

V_pi, p_pi = policy_iteration(env_s.P, env_s.nS, env_s.nA, gamma=0.9, tol=1e-3)
render_single(env_s, p_pi, 100, show_rendering=RENDER_ENV)
```

```
-------------------------
Beginning Policy Iteration
-------------------------
Episode reward: 1.000000
```

Evaluate All Policies

Now, we will first test the value iteration implementation on two kinds of environments - the dererministic FrozenLake and the stochastic FrozenLake. We will also run the same for policy iteration

Deliverable 1 (10 points)

Run value iteration on deterministic FrozenLake. You should get a reward of 1.0 for full credit.

In [11]:

```
print("\nValue Iteration on Deterministic FrozenLake:")
V_vi, p_vi = value_iteration(env_d.P, env_d.nS, env_d.nA, gamma=0.9, tol=1e-3)
evaluate(env_d, p_vi, max_steps=100, max_episodes=2)
```

```
Value Iteration on Deterministic FrozenLake:
> Average reward over 2 episodes:                        1.0
> Percentage of episodes goal reached:                   100%
```

Deliverable 2 (10 points)

Run value iteration on stochastic FrozenLake. Note that this time, running the same policy over multiple episodes will result in different outcomes (final reward) due to stochastic transitions in the environment, and even the optimal policy may not succeed in reaching the goal state 100% of the time.

You should get a reward of 0.7 or higher over 1000 episodes for full credit.

In [12]:

```
print("\nValue Iteration on Stochastic FrozenLake:")
V_vi, p_vi = value_iteration(env_s.P, env_s.nS, env_s.nA, gamma=0.9, tol=1e-3)
evaluate(env_s, p_vi, max_steps=100, max_episodes=1000)
```

```
Value Iteration on Stochastic FrozenLake:
> Average reward over 1000 episodes:                     0.701
> Percentage of episodes goal reached:                   94%
```

Deliverable 3 (5 bonus points)

Run policy iteration on deterministic FrozenLake. You should get a reward of 1.0 for full credit.

In [13]:

```
print("Policy Iteration on Deterministic FrozenLake:")
V_pi, p_pi = policy_iteration(env_d.P, env_d.nS, env_d.nA, gamma=0.9, tol=1e-3)
evaluate(env_d, p_pi, max_steps=100, max_episodes=2)
```

```
Policy Iteration on Deterministic FrozenLake:
> Average reward over 2 episodes:                        1.0
> Percentage of episodes goal reached:                   100%
```

Deliverable 4 (5 bonus points)

Run policy iteration on stochastic FrozenLake.

You should get a reward of 0.7 or higher over 1000 episodes for full credit.

In [14]:

```
print("Policy Iteration on Stochastic FrozenLake:")
V_pi, p_pi = policy_iteration(env_s.P, env_s.nS, env_s.nA, gamma=0.9, tol=1e-3)
evaluate(env_s, p_pi, max_steps=100, max_episodes=1000)
```

```
Policy Iteration on Stochastic FrozenLake:
> Average reward over 1000 episodes:                     0.714
> Percentage of episodes goal reached:                   93%
```

Submission Reminder

PLEASE RE-RUN THE NOTEBOOK WITH `RENDER_ENV` SET TO FALSE BEFORE SUBMISSION!

5.2 Deep Q-Learning

Q-Learning & DQNs (30 points + 5 bonus points)

In this section, we will implement a few key parts of the Q-Learning algorithm for two cases - (1) A Q-network which is a single linear layer (referred to in RL literature as "Q-learning with linear function approximation") and (2) A deep (convolutional) Q-network, for some Atari game environments where the states are images.

Optional Readings:

- **Playing Atari with Deep Reinforcement Learning**, Mnih et. al., https://www.cs.toronto.edu/~vmnih/docs/dqn.pdf (https://www.cs.toronto.edu/~vmnih/docs/dqn.pdf)
- **The PyTorch DQN Tutorial** https://pytorch.org/tutorials/intermediate/reinforcement_q_learning.html (https://pytorch.org/tutorials/intermediate/reinforcement_q_learning.html)

Note: **The bonus credit for this question applies to both sections CS 7643 and CS 4803**

In [1]:

```
%load_ext autoreload
%autoreload 2

import numpy as np
import gym

import torch
import torch.nn as nn
import torch.optim as optim

from core.dqn_train import DQNTrain
from utils.test_env import EnvTest
from utils.schedule import LinearExploration, LinearSchedule
from utils.preprocess import greyscale
from utils.wrappers import PreproWrapper, MaxAndSkipEnv

from linear_qnet import LinearQNet
from cnn_qnet import ConvQNet

if torch.cuda.is_available():
    device = torch.device('cuda', 0)
else:
    device = torch.device('cpu')
```

Part 1: Setup Q-Learning with Linear Function Approximation

Training Q-networks using (Deep) Q-learning involves a lot of moving parts. However, for this assignment, the scaffolding for the first 3 points listed below is provided in full and you must only complete point 4. You may skip to point 4 if you only care about the implementation required for this assignment.

1. **Environments**: We will use the standardized OpenAI Gym framework for environment API calls (read through http://gym.openai.com/docs/ (http://gym.openai.com/docs/) if you want to know more details about this interface). Specifically, we will use a custom Test environment defined in `utils/test_env.py` for initial sanity checks and then Gym-Atari environments later on.

1. **Exploration**: In order to train any RL model, we require experience or "data" gathered from interacting with the environment by taking actions. What policy should we use to collect this experience? Given a Q-network, one may be tempted to define a greedy policy which always picks the highest valued action at every state. However, this strategy will in most cases not work since we may get stuck in a local minima and never explore new states in the environment which may lead to a better reward. Hence, for the purpose of gathering experience (or "data") from the environment, it is useful to follow a policy that deviates from the greedy policy slightly in order to explore new states. A common strategy used in RL is to follow an ϵ-greedy policy which with probability $0 < \epsilon < 1$ picks a random action instead of the action provided by the greedy policy.

1. **Replay Buffers**: Data gathered from a single trajectory of states and actions in the environment provides us with a batch of highly correlated (non IID) data, which leads to high variance in gradient updates and convergence. In order to ameliorate this, replay buffers are used to gather a set of transitions i.e. (state, action, reward, next state) tuples, by executing multiple trajectories in the environment. Now, for updating the Q-Network, we will first wait to fill up our replay buffer with a sufficiently large number of transitions over multiple different trajectories, and then randomly sample a batch of transitions to compute loss and update the models.

1. **Q-Learning network, loss and update**: Finally, we come to the part of Q-learning that we will implement for this assignment -- the Q-network, loss function and update. In particular, we will implement a variant of Q-Learning called "Double Q-Learning", where we will maintain two Q networks -- the first Q network is used to pick actions and the second "target" Q network is used to compute Q-values for the picked actions. Here is some referance material on the same - Blog 1 (https://towardsdatascience.com/double-q-learning-the-easy-way-a924c4085ec3), Blog 2 (https://medium.com/@ameetsd97/deep-double-q-learning-why-you-should-use-it-bedf660d5295), but we will not need to get into the details of Double Q-learning for this assignment. Now, let's walk through the steps required to implement this below.
    - **Linear Q-Network**: In `linear_qnet.py`, define the initialization and forward pass of a Q-network with a single linear layer which takes the state as input and outputs the Q-values for all actions.
    - **Setting up Q-Learning**: In `core/dqn_train.py`, complete the functions `process_state`, `forward_loss` and `update_step` and `update_target_params`. The loss function for our Q-Networks is defined for a single transition tuple of (state, action, reward, next state) as follows. $Q(s_t, a_t)$ refers to the state-action values computed by our first Q-network at the current state and and for the current actions, $Q_{target}(s_{t+1}, a_{t+1})$ refers to the state-action values for the next state and all possible future actions computed by the target Q-Network

$$Q_{sample}(s_t) = r_t \text{ if done}$$

$$= r_t + \gamma \max_{a_{t+1}} Q_{target}\left(s_{t+1}, a_{t+1}\right) \text{ otherwise}$$

$$\text{Loss} = \left(Q_{sample}(s_t) - Q(s_t, a_t)\right)^2$$

Deliverable 1 (15 points)

Run the following block of code to train a Linear Q-Network. You should get an average reward of ~4.0, full credit will be given if average reward at the final evaluation is above 3.5

In [69]:

```
from configs.p1_linear import config as config_lin

env = EnvTest((5, 5, 1))

# exploration strategy
exp_schedule = LinearExploration(env, config_lin.eps_begin,
        config_lin.eps_end, config_lin.eps_nsteps)

# learning rate schedule
lr_schedule  = LinearSchedule(config_lin.lr_begin, config_lin.lr_end,
        config_lin.lr_nsteps)

# train model
model = DQNTrain(LinearQNet, env, config_lin, device)
model.run(exp_schedule, lr_schedule)
```

```
Evaluating...
Average reward: -1.00 +/- 0.00

 1001/10000 [==>...........................] - ETA: 5s - Loss: 6.4078 - Avg_R: 1.0150 - Max_R: 4.0000 - eps: 0.8020 - Grads: 12.177
2 - Max_Q: 0.6513 - lr: 0.0042

Evaluating...
Average reward: 3.80 +/- 0.00

 2001/10000 [=====>........................] - ETA: 5s - Loss: 5.8750 - Avg_R: 1.7150 - Max_R: 4.1000 - eps: 0.6040 - Grads: 19.473
4 - Max_Q: 1.7326 - lr: 0.0034

Evaluating...
Average reward: 3.90 +/- 0.00

 3001/10000 [========>.....................] - ETA: 4s - Loss: 13.2152 - Avg_R: 2.2650 - Max_R: 4.1000 - eps: 0.4060 - Grads: 52.37
76 - Max_Q: 2.3394 - lr: 0.0026

Evaluating...
Average reward: 3.80 +/- 0.00

 4001/10000 [===========>..................] - ETA: 3s - Loss: 7.9353 - Avg_R: 2.8200 - Max_R: 4.1000 - eps: 0.2080 - Grads: 25.314
4 - Max_Q: 2.6468 - lr: 0.0018

Evaluating...
Average reward: 4.10 +/- 0.00

 5001/10000 [==============>...............] - ETA: 3s - Loss: 3.5959 - Avg_R: 4.0650 - Max_R: 4.1000 - eps: 0.0100 - Grads: 32.987
3 - Max_Q: 2.5674 - lr: 0.0010

Evaluating...
Average reward: 4.10 +/- 0.00

 6001/10000 [=================>............] - ETA: 2s - Loss: 0.3704 - Avg_R: 3.9150 - Max_R: 4.1000 - eps: 0.0100 - Grads: 9.9973
- Max_Q: 2.8478 - lr: 0.0010  ETA: 2s - Loss: 0.7227 - Avg_R: 4.0850 - Max_R: 4.1000 - eps: 0.0100 - Grads: 13.4264 - Max_Q: 2.7099
- lr:

Evaluating...
Average reward: 4.10 +/- 0.00

 7001/10000 [====================>.........] - ETA: 2s - Loss: 1.0530 - Avg_R: 4.0500 - Max_R: 4.1000 - eps: 0.0100 - Grads: 18.250
4 - Max_Q: 2.7377 - lr: 0.0010

Evaluating...
Average reward: 4.10 +/- 0.00

 8001/10000 [=======================>......] - ETA: 1s - Loss: 0.0247 - Avg_R: 4.1000 - Max_R: 4.1000 - eps: 0.0100 - Grads: 3.1597
- Max_Q: 2.6737 - lr: 0.0010

Evaluating...
Average reward: 4.10 +/- 0.00

 9001/10000 [==========================>...] - ETA: 0s - Loss: 0.0435 - Avg_R: 4.1000 - Max_R: 4.1000 - eps: 0.0100 - Grads: 4.7094
- Max_Q: 2.6844 - lr: 0.0010

Evaluating...
Average reward: 4.10 +/- 0.00

10001/10000 [==============================] - 6s - Loss: 0.3452 - Avg_R: 4.1000 - Max_R: 4.1000 - eps: 0.0100 - Grads: 5.2634 - Ma
x_Q: 2.6558 - lr: 0.0010

- Training done.
Evaluating...


Average reward: 4.10 +/- 0.00
```

You should get a final average reward of over 4.0 on the test environment.

Part 2: Q-Learning with Deep Q-Networks

In `cnn_qnet.py`, implement the initialization and forward pass of a convolutional Q-network with architecture as described in this DeepMind paper:

"Playing Atari with Deep Reinforcement Learning", Mnih et. al. (https://www.cs.toronto.edu/~vmnih/docs/dqn.pdf (https://www.cs.toronto.edu/~vmnih/docs/dqn.pdf))

Deliverable 2 (10 points)

Run the following block of code to train our Deep Q-Network. You should get an average reward of ~4.0, full credit will be given if average reward at the final evaluation is above 3.5

In [80]:

```
from configs.p2_cnn import config as config_cnn

env = EnvTest((80, 80, 1))

# exploration strategy
exp_schedule = LinearExploration(env, config_cnn.eps_begin,
        config_cnn.eps_end, config_cnn.eps_nsteps)

# learning rate schedule
lr_schedule  = LinearSchedule(config_cnn.lr_begin, config_cnn.lr_end,
        config_cnn.lr_nsteps)

# train model
model = DQNTrain(ConvQNet, env, config_cnn, device)
model.run(exp_schedule, lr_schedule)
```

```
Evaluating...
Average reward: -0.10 +/- 0.00

Populating the memory 150/200...

Evaluating...


Average reward: -0.50 +/- 0.00

 301/1000 [========>.....................] - ETA: 1s - Loss: 3.1996 - Avg_R: 0.3350 - Max_R: 2.2000 - eps: 0.4060 - Grads: 62.2550
- Max_Q: 0.1038 - lr: 0.0002

Evaluating...
Average reward: 0.50 +/- 0.00

 401/1000 [===========>..................] - ETA: 1s - Loss: 2.2810 - Avg_R: 0.1600 - Max_R: 2.1000 - eps: 0.2080 - Grads: 50.1883
- Max_Q: 0.1335 - lr: 0.0001

Evaluating...
Average reward: 0.50 +/- 0.00

 501/1000 [==============>...............] - ETA: 1s - Loss: 2.8834 - Avg_R: 0.2150 - Max_R: 2.3000 - eps: 0.0100 - Grads: 65.2602
- Max_Q: 0.1339 - lr: 0.0001

Evaluating...
Average reward: 0.50 +/- 0.00

 601/1000 [=================>............] - ETA: 0s - Loss: 4.3118 - Avg_R: 1.8450 - Max_R: 4.0000 - eps: 0.0100 - Grads: 59.2287
- Max_Q: 0.2047 - lr: 0.0001

Evaluating...
Average reward: 4.00 +/- 0.00

 701/1000 [====================>.........] - ETA: 0s - Loss: 3.1365 - Avg_R: 3.9900 - Max_R: 4.0000 - eps: 0.0100 - Grads: 31.6353
- Max_Q: 0.3318 - lr: 0.0001

Evaluating...
Average reward: 3.90 +/- 0.00

 801/1000 [=======================>......] - ETA: 0s - Loss: 1.0304 - Avg_R: 3.9550 - Max_R: 4.0000 - eps: 0.0100 - Grads: 28.7879
- Max_Q: 0.4492 - lr: 0.0001

Evaluating...
Average reward: 2.10 +/- 0.00

 901/1000 [==========================>...] - ETA: 0s - Loss: 0.3808 - Avg_R: 3.6600 - Max_R: 4.1000 - eps: 0.0100 - Grads: 34.4124
- Max_Q: 0.5452 - lr: 0.0001

Evaluating...
Average reward: 4.10 +/- 0.00

1001/1000 [==============================] - 2s - Loss: 0.3321 - Avg_R: 4.1000 - Max_R: 4.1000 - eps: 0.0100 - Grads: 71.5977 - Max
_Q: 0.6279 - lr: 0.0001

- Training done.
Evaluating...
Average reward: 4.10 +/- 0.00
```

You should get a final average reward of over 4.0 on the test environment, similar to the previous case.

Part 3: Playing Atari Games from Pixels - using Linear Function Approximation

Now that we have setup our Q-Learning algorithm and tested it on a simple test environment, we will shift to a harder environment - an Atari 2600 game from OpenAI Gym: Pong-v0 (https://gym.openai.com/envs/Pong-v0/ (https://gym.openai.com/envs/Pong-v0/)), where we will use RGB images of the game screen as our observations for state.

No additional implementation is required for this part, just run the block of code below (will take around 1 hour to train). We don't expect a simple linear Q-network to do well on such a hard environment - full credit will be given simply for running the training to completion irrespective of the final average reward obtained.

You may edit `configs/p3_train_atari_linear.py` if you wish to play around with hyperparamters for improving performance of the linear Q-network on Pong-v0, or try another Atari environment by changing the `env_name` hyperparameter. The list of all Gym Atari environments are available here: https://gym.openai.com/envs/#atari (https://gym.openai.com/envs/#atari)

Deliverable 3 (5 points)

Run the following block of code to train a linear Q-network on Atari Pong-v0. We don't expect the linear Q-Network to learn anything meaingful so full credit will be given for simply running this training to completion (without errors), irrespective of the final average reward.

In [78]:

```
from configs.p3_train_atari_linear import config as config_lina

# make env
env = gym.make(config_lina.env_name)
env = MaxAndSkipEnv(env, skip=config_lina.skip_frame)
env = PreproWrapper(env, prepro=greyscale, shape=(80, 80, 1),
                    overwrite_render=config_lina.overwrite_render)

# exploration strategy
exp_schedule = LinearExploration(env, config_lina.eps_begin,
        config_lina.eps_end, config_lina.eps_nsteps)

# learning rate schedule
lr_schedule  = LinearSchedule(config_lina.lr_begin, config_lina.lr_end,
        config_lina.lr_nsteps)

# train model
model = DQNTrain(LinearQNet, env, config_lina, device)
print("Linear Q-Net Architecture:\n", model.q_net)
model.run(exp_schedule, lr_schedule)
```

```
/home/nicolas/.local/lib/python3.6/site-packages/gym/envs/registration.py:14: PkgResourcesDeprecationWarning: Parameters to load ar
e deprecated.  Call .resolve and .require separately.
  result = entry_point.load(False)
Evaluating...

Linear Q-Net Architecture:
 LinearQNet(
  (linear): Linear(in_features=25600, out_features=6, bias=True)
)

Average reward: -21.00 +/- 0.00

250001/500000 [==============>..............] - ETA: 698s - Loss: 0.9539 - Avg_R: -20.3600 - Max_R: -18.0000 - eps: 0.7750 - Grads
: 95.4523 - Max_Q: 9.9121 - lr: 0.0001

Evaluating...


Average reward: -20.96 +/- 0.03

500001/500000 [==============================] - 1454s - Loss: 0.7359 - Avg_R: -20.5200 - Max_R: -19.0000 - eps: 0.5500 - Grads: 12
5.4631 - Max_Q: 9.9664 - lr: 0.0001

- Training done.
Evaluating...


Average reward: -20.92 +/- 0.04
```

Part 4: [BONUS] Playing Atari Games from Pixels - using Deep Q-Networks

This part is extra credit and worth 5 bonus points. We will now train our deep Q-Network from Part 2 on Pong-v0.

Again, no additional implementation is required but you may wish to tweak your CNN architecture in `cnn_qnet.py` and hyperparameters in `configs/p4_train_atari_cnn.py` (however, evaluation will be considered at no farther than the default 5 million steps, so you are not allowed to train for longer). Please note that this training may take a very long time (we tested this on a single GPU and it took around 6 hours).

The bonus points for this question will be allotted based on the best evaluation average reward (EAR) before 5 million time stpes:

1. EAR >= 0.0 : 4/4 points
2. EAR >= -5.0 : 3/4 points
3. EAR >= -10.0 : 3/4 points
4. EAR >= -15.0 : 1/4 points

Deliverable 4: (5 bonus points)

Run the following block of code to train your DQN:

In [3]:

```
from configs.p4_train_atari_cnn import config as config_cnna


# make env
env = gym.make(config_cnna.env_name)
env = MaxAndSkipEnv(env, skip=config_cnna.skip_frame)
env = PreproWrapper(env, prepro=greyscale, shape=(80, 80, 1),
                    overwrite_render=config_cnna.overwrite_render)

# exploration strategy
exp_schedule = LinearExploration(env, config_cnna.eps_begin,
        config_cnna.eps_end, config_cnna.eps_nsteps)

# learning rate schedule
lr_schedule  = LinearSchedule(config_cnna.lr_begin, config_cnna.lr_end,
        config_cnna.lr_nsteps)

# train model
model = DQNTrain(ConvQNet, env, config_cnna, device)
print("CNN Q-Net Architecture:\n", model.q_net)
model.run(exp_schedule, lr_schedule)
```

```
Evaluating...

CNN Q-Net Architecture:
 ConvQNet(
  (conv1): Conv2d(4, 16, kernel_size=(8, 8), stride=(4, 4))
  (relu1): ReLU()
  (conv2): Conv2d(16, 32, kernel_size=(4, 4), stride=(2, 2))
  (relu2): ReLU()
  (advantage1): Linear(in_features=2048, out_features=128, bias=True)
  (relu3): ReLU()
  (advantage2): Linear(in_features=128, out_features=6, bias=True)
  (value1): Linear(in_features=2048, out_features=128, bias=True)
  (relu4): ReLU()
  (value2): Linear(in_features=128, out_features=1, bias=True)
)

Average reward: -21.00 +/- 0.00

 250301/5000000 [>.............................] - ETA: 49772s - Loss: 9.5663 - Avg_R: -20.1400 - Max_R: -17.0000 - eps: 0.7747 - G
rads: 65.2361 - Max_Q: 0.1439 - lr: 0.0002

Evaluating...
```

```
Average reward: -17.10 +/- 0.22

 500501/5000000 [==>...........................] - ETA: 52177s - Loss: 7.1902 - Avg_R: -18.8600 - Max_R: -15.0000 - eps: 0.5496 - G
rads: 65.6029 - Max_Q: 0.8861 - lr: 0.0002

Evaluating...


Average reward: -15.72 +/- 0.30

 750801/5000000 [===>..........................] - ETA: 50891s - Loss: 9.9263 - Avg_R: -17.2000 - Max_R: -10.0000 - eps: 0.3243 - G
rads: 75.6627 - Max_Q: 1.3700 - lr: 0.0002

Evaluating...


Average reward: -12.86 +/- 0.43

1001201/5000000 [=====>........................] - ETA: 48637s - Loss: 5.2073 - Avg_R: -13.6800 - Max_R: -7.0000 - eps: 0.1000 - Gr
ads: 55.8515 - Max_Q: 1.6066 - lr: 0.0002

Evaluating...


Average reward: -13.38 +/- 0.41

1251601/5000000 [======>.......................] - ETA: 45873s - Loss: 7.0315 - Avg_R: -13.0400 - Max_R: -2.0000 - eps: 0.1000 - Gr
ads: 73.1891 - Max_Q: 1.6714 - lr: 0.0001

Evaluating...


Average reward: -11.00 +/- 0.52

1501601/5000000 [========>.....................] - ETA: 42893s - Loss: 7.1460 - Avg_R: -11.5400 - Max_R: -3.0000 - eps: 0.1000 - Gr
ads: 79.0992 - Max_Q: 1.5813 - lr: 0.0001

Evaluating...


Average reward: -11.88 +/- 0.50

1752401/5000000 [=========>....................] - ETA: 39737s - Loss: 8.2967 - Avg_R: -11.3200 - Max_R: 5.0000 - eps: 0.1000 - Gra
ds: 69.5188 - Max_Q: 1.4486 - lr: 0.0001

Evaluating...


Average reward: -9.34 +/- 0.56

2002701/5000000 [===========>..................] - ETA: 36592s - Loss: 10.7280 - Avg_R: -11.4600 - Max_R: -3.0000 - eps: 0.1000 - G
rads: 125.6370 - Max_Q: 1.3241 - lr: 0.0001

Evaluating...


Average reward: -9.24 +/- 0.64

2253201/5000000 [============>.................] - ETA: 33457s - Loss: 5.9579 - Avg_R: -10.2400 - Max_R: -1.0000 - eps: 0.1000 - Gr
ads: 85.9777 - Max_Q: 1.2528 - lr: 0.0001

Evaluating...


Average reward: -7.16 +/- 0.78

2504201/5000000 [==============>...............] - ETA: 30348s - Loss: 7.7309 - Avg_R: -9.9000 - Max_R: 2.0000 - eps: 0.1000 - Grad
s: 77.5510 - Max_Q: 1.1142 - lr: 0.0001

Evaluating...


Average reward: -8.24 +/- 0.66

2754701/5000000 [===============>..............] - ETA: 27258s - Loss: 5.2205 - Avg_R: -8.6200 - Max_R: 5.0000 - eps: 0.1000 - Grad
s: 85.7707 - Max_Q: 1.0944 - lr: 0.0001

Evaluating...
```

```
Average reward: -3.60 +/- 1.02

3005601/5000000 [=================>............] - ETA: 24151s - Loss: 5.7598 - Avg_R: -5.3200 - Max_R: 6.0000 - eps: 0.1000 - Grad
s: 75.9029 - Max_Q: 0.9720 - lr: 0.0001

Evaluating...


Average reward: -3.08 +/- 0.95

3256501/5000000 [==================>...........] - ETA: 21103s - Loss: 5.4483 - Avg_R: -7.2200 - Max_R: 4.0000 - eps: 0.1000 - Grad
s: 88.8399 - Max_Q: 0.8678 - lr: 0.0001

Evaluating...


Average reward: -3.08 +/- 1.02

3506801/5000000 [====================>.........] - ETA: 18171s - Loss: 9.5677 - Avg_R: -5.6600 - Max_R: 13.0000 - eps: 0.1000 - Gra
ds: 88.4731 - Max_Q: 0.9018 - lr: 0.0001

Evaluating...


Average reward: -2.38 +/- 0.85

3757201/5000000 [=====================>........] - ETA: 15329s - Loss: 5.2974 - Avg_R: -6.2400 - Max_R: 7.0000 - eps: 0.1000 - Grad
s: 74.1779 - Max_Q: 0.8903 - lr: 0.0001

Evaluating...


Average reward: -1.90 +/- 0.84

4007501/5000000 [=======================>......] - ETA: 12301s - Loss: 6.3289 - Avg_R: -5.0200 - Max_R: 10.0000 - eps: 0.1000 - Gra
ds: 75.6850 - Max_Q: 0.9433 - lr: 0.0001

Evaluating...


Average reward: 0.16 +/- 0.97

4257501/5000000 [========================>.....] - ETA: 9181s - Loss: 8.3986 - Avg_R: -4.5600 - Max_R: 12.0000 - eps: 0.1000 - Grad
s: 108.1021 - Max_Q: 0.8888 - lr: 0.0001

Evaluating...


Average reward: -0.78 +/- 0.86

4507701/5000000 [==========================>...] - ETA: 6076s - Loss: 6.1040 - Avg_R: -3.8200 - Max_R: 9.0000 - eps: 0.1000 - Grads
: 80.0519 - Max_Q: 0.8612 - lr: 0.0001

Evaluating...


Average reward: -0.94 +/- 0.88

4758701/5000000 [===========================>..] - ETA: 2972s - Loss: 8.7027 - Avg_R: -4.3600 - Max_R: 12.0000 - eps: 0.1000 - Grad
s: 93.0876 - Max_Q: 0.9098 - lr: 0.0001

Evaluating...


Average reward: 0.76 +/- 0.94

5000001/5000000 [==============================] - 61477s - Loss: 9.3504 - Avg_R: -6.0000 - Max_R: 10.0000 - eps: 0.1000 - Grads: 9
4.6308 - Max_Q: 0.9095 - lr: 0.0001

- Training done.
Evaluating...


Average reward: -0.28 +/- 0.87
```