Temporal-Difference Methods

In this notebook, you will write your own implementations of many Temporal-Difference (TD) methods.

While we have provided some starter code, you are welcome to erase these hints and write your code from scratch.

Part 0: Explore CliffWalkingEnv

We begin by importing the necessary packages.

In [1]:
```
%load_ext autoreload
%autoreload 2
import numpy as np
from collections import defaultdict, deque
import matplotlib.pyplot as plt
%matplotlib inline

import check_test
from plot_utils import plot_values
```

Use the code cell below to create an instance of the CliffWalking environment.

In [2]:
```
env = gym.make('CliffWalking-v0')
```

The agent moves through a 4×12 gridworld, with states numbered as follows:

```
[[ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11],
 [12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23],
 [24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35],
 [36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47]]
```

At the start of any episode, state `36` is the initial state. State `47` is the only terminal state, and the cliff corresponds to states `37` through `46`.

The agent has 4 potential actions:

```
UP = 0
RIGHT = 1
DOWN = 2
LEFT = 3
```

Thus, $\mathcal{S}^+ = \{0, 1, \ldots, 47\}$, and $\mathcal{A} = \{0, 1, 2, 3\}$. Verify this by running the code cell below.

In [3]:
```
print(env.action_space)
print(env.observation_space)
```

```
Discrete(4)
Discrete(48)
```

In this mini-project, we will build towards finding the optimal policy for the CliffWalking environment. The optimal state-value function is visualized below. Please take the time now to make sure that you understand why this is the optimal state-value function.

Note *: You can safely ignore the values of the cliff "states" as these are not true states from which the agent can make decisions. For the cliff "states", the state-value function is not well-defined.*

In [4]:
```
# define the optimal state-value function
V_opt = np.zeros((4,12))
V_opt[0:13][0] = -np.arange(3, 15)[::-1]
V_opt[0:13][1] = -np.arange(3, 15)[::-1] + 1
V_opt[0:13][2] = -np.arange(3, 15)[::-1] + 2
V_opt[3][0] = -13

plot_values(V_opt)
```

```
/home/jeffmacklin/anaconda3-5.1/lib/python3.7/site-packages/matplotlib/cbook/deprecation.py:107: MatplotlibDeprecationWarning: Passing one of 'on' or 'off', 'true', 'false' as a boolean is deprecated; use an actual boolean (True/False) instead.
  warnings.warn(message, mplDeprecation, stacklevel=1)
/home/jeffmacklin/anaconda3-5.1/lib/python3.7/site-packages/matplotlib/cbook/deprecation.py:107: MatplotlibDeprecationWarning: Passing one of 'on' or 'off', 'true', 'false' as a boolean is deprecated; use an actual boolean (True/False) instead.
  warnings.warn(message, mplDeprecation, stacklevel=1)
/home/jeffmacklin/anaconda3-5.1/lib/python3.7/site-packages/matplotlib/cbook/deprecation.py:107: MatplotlibDeprecationWarning: Passing one of 'on' or 'off', 'true', 'false' as a boolean is deprecated; use an actual boolean (True/False) instead.
  warnings.warn(message, mplDeprecation, stacklevel=1)
/home/jeffmacklin/anaconda3-5.1/lib/python3.7/site-packages/matplotlib/cbook/deprecation.py:107: MatplotlibDeprecationWarning: Passing one of 'on' or 'off', 'true', 'false' as a boolean is deprecated; use an actual boolean (True/False) instead.
  warnings.warn(message, mplDeprecation, stacklevel=1)
```

Part 1: TD Control: Sarsa

In this section, you will write your own implementation of the Sarsa control algorithm.

Your algorithm has four arguments:

- `env`: This is an instance of an OpenAI Gym environment.
- `num_episodes`: This is the number of episodes that are generated through agent environment interaction.
- `alpha`: This is the step-size parameter for the update step.
- `gamma`: This is the discount rate. It must be a value between 0 and 1, inclusive (default value: `1`).

The algorithm returns as output:

- `Q`: This is a dictionary (of one-dimensional arrays) where `Q[s][a]` is the estimated action value corresponding to state `s` and action `a`.

Please complete the function in the code cell below.

(Feel free to define additional functions to help you to organize your code.)

In [5]:
```
def policy(state, nps, ns):
    prob = np.ones(ns)* nps/ns
    action = np.argmax(state)
    prob[action] = 1 - nps + nps/ns
    return prob

def update_Q(s, state, action, reward, new_state, done, alpha, gamma, ns, nps):
    new_action = np.random.choice(np.arange(ns), p = policy(s[1][new_state], nps, ns))
                                                    if new_state == 36 else new_action_space.sample()
    if not done:
        s.append((new_state,new_action))
    else:
        s.append((-1, -1))
    return s[-2][1] + alpha * (s[-1][1] - s[-2][1] + reward + gamma * Q_next - s[-2][1])[action]
    return Q, new_action
```

In [6]:
```
def run_sarsa(env, num_episodes, alpha, gamma):
    state = env.reset()
    eps = 1.0
    rewards = []
    done = False
    nps = np.inf
    ns = np.shape(state)[-1]
    while not done:
        new_state, reward, done, _ = env.step(action)
        if done:
            Q = update_Q(s, state, action, reward, new_state,done, alpha, gamma, ns, nps)
            break
        else:
            new_action = update_Q(s, state, action, reward, new_state,done, alpha, gamma, ns, nps)
        rewards.append(reward)
        state = new_state
        action = new_action
    return Q, rewards
```

In [8]:
```
def sarsa(env, num_episodes, eps_init = 1.0, eps_decay = 0.9, eps_min = 0.0001, alpha = 0.009, gamma = 1.0):
    # initialize action-value function (empty dictionary of arrays)
    nA = env.nA
    Q = defaultdict(lambda: np.zeros(nA))
    # initialize performance monitor
    # loop over episodes
    rewards = []
    eps = eps_init
    for i_episode in range(1, num_episodes+1):
        if i_episode % 100 == 0:
            print("\rEpisode {}/{}".format(i_episode, num_episodes), end="")
            sys.stdout.flush()
        rewards.append(np.mean(r))
    plot_plots(rewards)
    print("Estimated Optimal Policy:")
    pick_states("Rewards")
    return Q
```

Use the next code cell to visualize the ***estimated*** optimal policy and the corresponding state-value function.

If the code cell returns **PASSED**, then you have implemented the function correctly! Feel free to change the `num_episodes` and `alpha` parameters that are supplied to the function. However, if you'd like to ensure the accuracy of the unit test, please do not change the value of `gamma` from the default.

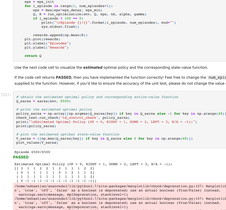
In [9]:
```
# obtain the estimated optimal policy and corresponding action-value function
Q_sarsa = sarsa(env, 4500)

# print the estimated optimal policy
policy_sarsa = np.array([np.argmax(Q_sarsa[key]) if key in Q_sarsa else -1 for key in np.arange(48)].reshape(4,12)
check_test.run_check('td_control_check', policy_sarsa)
print("Estimated Optimal Policy UP = 0, RIGHT = 1, DOWN = 2, LEFT = 3, N/A = -1:")
print(policy_sarsa)

# plot the estimated optimal state-action function
plot_values(np.max(Q_sarsa,axis=1).reshape(4,12))
```

```
Episode 4500/4500
PASSED

Estimated Optimal Policy UP = 0, RIGHT = 1, DOWN = 2, LEFT = 3, N/A = -1:
[[ 1 1 1 1 1 1 1 1 1 1 1 1]
 [ 1 1 1 1 1 1 1 1 1 1 1 1]
 [ 1 1 1 1 1 1 1 1 1 1 1 1]
 [ 1 2 2 2 2 2 2 2 2 2 2 0]
/home/jeffmacklin/anaconda3-5.1/lib/python3.7/site-packages/matplotlib/cbook/deprecation.py:107: MatplotlibDeprecationWarning: Passing one of 'on' or 'off', 'true', 'false' as a boolean is deprecated; use an actual boolean (True/False) instead.
  warnings.warn(message, mplDeprecation, stacklevel=1)
/home/jeffmacklin/anaconda3-5.1/lib/python3.7/site-packages/matplotlib/cbook/deprecation.py:107: MatplotlibDeprecationWarning: Passing one of 'on' or 'off', 'true', 'false' as a boolean is deprecated; use an actual boolean (True/False) instead.
  warnings.warn(message, mplDeprecation, stacklevel=1)
/home/jeffmacklin/anaconda3-5.1/lib/python3.7/site-packages/matplotlib/cbook/deprecation.py:107: MatplotlibDeprecationWarning: Passing one of 'on' or 'off', 'true', 'false' as a boolean is deprecated; use an actual boolean (True/False) instead.
  warnings.warn(message, mplDeprecation, stacklevel=1)
/home/jeffmacklin/anaconda3-5.1/lib/python3.7/site-packages/matplotlib/cbook/deprecation.py:107: MatplotlibDeprecationWarning: Passing one of 'on' or 'off', 'true', 'false' as a boolean is deprecated; use an actual boolean (True/False) instead.
  warnings.warn(message, mplDeprecation, stacklevel=1)
```



Part 2: TD Control: Q-learning

In this section, you will write your own implementation of the Q-learning control algorithm.

Your algorithm has four arguments:

- `env`: This is an instance of an OpenAI Gym environment.
- `num_episodes`: This is the number of episodes that are generated through agent environment interaction.
- `alpha`: This is the step-size parameter for the update step.
- `gamma`: This is the discount rate. It must be a value between 0 and 1, inclusive (default value: `1`).

The algorithm returns as output:

- `Q`: This is a dictionary (of one-dimensional arrays) where `Q[s][a]` is the estimated action value corresponding to state `s` and action `a`.

Please complete the function in the code cell below.

(Feel free to define additional functions to help you to organize your code.)

In [10]:
```
def policy(state, nps, ns):
    prob = np.ones(ns)* nps/ns
    action = np.argmax(state)
    prob[action] = 1 - nps + nps/ns
    return prob

def update_Q(s, state, action, reward, new_state, alpha, gamma, ns, nps):
    new_action = np.argmax(s[1][new_state])
    s[1][action] = s[1][action] + alpha * (reward + gamma * s[1][new_state][new_action] - s[1][action])
    return s

def run_qlearning(env, num_episodes, nps):
    state = env.reset()
    eps = 1.0
    rewards = []
    done = False
    ns = np.shape(state)[-1]
    for i_episode in range(1, num_episodes+1):
        if i_episode % 100 == 0:
            print("\rEpisode {}/{}".format(i_episode, num_episodes), end="")
            sys.stdout.flush()
        rewards.append(np.mean(r))
    plot_plots(rewards)
    print("Estimated Optimal Policy:")
    pick_states("Rewards")
    return Q
```

Use the next code cell to visualize the ***estimated*** optimal policy and the corresponding state-value function.

If the code cell returns **PASSED**, then you have implemented the function correctly! Feel free to change the `num_episodes` and `alpha` parameters that are supplied to the function. However, if you'd like to ensure the accuracy of the unit test, please do not change the value of `gamma` from the default.

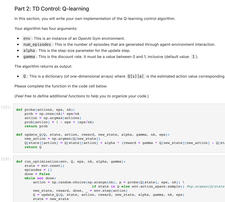
In [11]:
```
# obtain the estimated optimal policy and corresponding action-value function
Q_qlearning = qlearning(env, 5000, .01)

# print the estimated optimal policy
policy_qlearning = np.array([np.argmax(Q_qlearning[key]) if key in Q_qlearning else -1 for key in np.arange(48)].reshape(4,12)
check_test.run_check('td_control_check', policy_qlearning)
print("Estimated Optimal Policy UP = 0, RIGHT = 1, DOWN = 2, LEFT = 3, N/A = -1:")
print(policy_qlearning)

# plot the estimated optimal state-action function
plot_values([np.max(Q_qlearning,axis=1).reshape(4,12)])
```

```
Episode 5000/5000
PASSED

Estimated Optimal Policy UP = 0, RIGHT = 1, DOWN = 2, LEFT = 3, N/A = -1:
[[ 1 1 1 1 1 1 1 1 1 1 1 1]
 [ 1 1 1 1 1 1 1 1 1 1 1 1]
 [ 1 1 1 1 1 1 1 1 1 1 1 1]
 [ 1 2 2 2 2 2 2 2 2 2 2 0]
/home/jeffmacklin/anaconda3-5.1/lib/python3.7/site-packages/matplotlib/cbook/deprecation.py:107: MatplotlibDeprecationWarning: Passing one of 'on' or 'off', 'true', 'false' as a boolean is deprecated; use an actual boolean (True/False) instead.
  warnings.warn(message, mplDeprecation, stacklevel=1)
/home/jeffmacklin/anaconda3-5.1/lib/python3.7/site-packages/matplotlib/cbook/deprecation.py:107: MatplotlibDeprecationWarning: Passing one of 'on' or 'off', 'true', 'false' as a boolean is deprecated; use an actual boolean (True/False) instead.
  warnings.warn(message, mplDeprecation, stacklevel=1)
/home/jeffmacklin/anaconda3-5.1/lib/python3.7/site-packages/matplotlib/cbook/deprecation.py:107: MatplotlibDeprecationWarning: Passing one of 'on' or 'off', 'true', 'false' as a boolean is deprecated; use an actual boolean (True/False) instead.
  warnings.warn(message, mplDeprecation, stacklevel=1)
/home/jeffmacklin/anaconda3-5.1/lib/python3.7/site-packages/matplotlib/cbook/deprecation.py:107: MatplotlibDeprecationWarning: Passing one of 'on' or 'off', 'true', 'false' as a boolean is deprecated; use an actual boolean (True/False) instead.
  warnings.warn(message, mplDeprecation, stacklevel=1)
```



Part 3: TD Control: Expected Sarsa

In this section, you will write your own implementation of the Expected Sarsa control algorithm.

Your algorithm has four arguments:

- `env`: This is an instance of an OpenAI Gym environment.
- `num_episodes`: This is the number of episodes that are generated through agent environment interaction.
- `alpha`: This is the step-size parameter for the update step.
- `gamma`: This is the discount rate. It must be a value between 0 and 1, inclusive (default value: `1`).

The algorithm returns as output:

- `Q`: This is a dictionary (of one-dimensional arrays) where `Q[s][a]` is the estimated action value corresponding to state `s` and action `a`.

Please complete the function in the code cell below.

(Feel free to define additional functions to help you to organize your code.)

In [12]:
```
def policy(state, nps, ns):
    prob = np.ones(ns)* nps/ns
    action = np.argmax(state)
    prob[action] = 1 - nps + nps/ns
    return prob

def update_Q(s, state, action, reward, new_state, alpha, gamma, ns, nps):
    pol = policy(s[1][new_state], nps, ns)
    s[1][action] = s[1][action] + alpha * (reward + gamma * (s[1][new_state] * pol).sum() - s[1][action])
    return s

def run_expectedsarsa(env, num_episodes, nps):
    state = env.reset()
    eps = 1.0
    rewards = []
    done = False
    ns = np.shape(state)[-1]
    while not done:
        new_state, reward, done, _ = env.step(action)
        if done:
            s = update_Q(s, state, action, reward, new_state, alpha, gamma, ns, nps)
            break
        else:
            new_action = np.random.choice(np.arange(ns), p = policy(s[1][new_state], nps, ns)) \
                                                    if new_state == 36 else env.action_space.sample()
        s = update_Q(s, state, action, reward, new_state, alpha, gamma, ns, nps)
        state = new_state
        action = new_action
        if done:
            break
    return s, rewards
```

In [13]:
```
def run_expectedsarsa(env, num_episodes, nps):
    state = env.reset()
    eps = 1.0
    rewards = []
    done = False
    ns = np.shape(state)[-1]
    while not done:
        new_state, reward, done, _ = env.step(action)
        if done:
            s = update_Q(s, state, action, reward, new_state, alpha, gamma, ns, nps)
            break
        else:
            new_action = np.random.choice(np.arange(ns), p = policy(s[1][new_state], nps, ns))
                                                    if new_state == 36 else env.action_space.sample()
        s = update_Q(s, state, action, reward, new_state, alpha, gamma, ns, nps)
        state = new_state
        action = new_action
        if done:
            break
    return s, rewards
```

In [14]:
```
def expected_sarsa(env, num_episodes, eps_init = 1.0, eps_decay = 0.9, eps_min = 0.0001, alpha = 0.01, gamma=1.0):
    # initialize action-value function (empty dictionary of arrays)
    nA = env.nA
    Q = defaultdict(lambda: np.zeros(nA))
    # initialize performance monitor
    # loop over episodes
    rewards = []
    eps = eps_init
    for i_episode in range(1, num_episodes+1):
        if i_episode % 100 == 0:
            print("\rEpisode {}/{}".format(i_episode, num_episodes), end="")
            sys.stdout.flush()
        rewards.append(np.mean(r))
    plot_plots(rewards)
    print("Estimated Optimal Policy:")
    pick_states("Rewards")
    return Q
```

Use the next code cell to visualize the ***estimated*** optimal policy and the corresponding state-value function.

If the code cell returns **PASSED**, then you have implemented the function correctly! Feel free to change the `num_episodes` and `alpha` parameters that are supplied to the function. However, if you'd like to ensure the accuracy of the unit test, please do not change the value of `gamma` from the default.

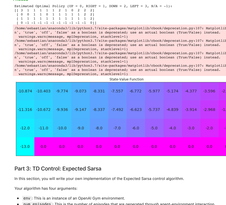
In [15]:
```
# obtain the estimated optimal policy and corresponding action-value function
Q_expected_sarsa = expected_sarsa(env, 5000)

# print the estimated optimal policy
policy_expected_sarsa = np.array([np.argmax(Q_expected_sarsa[key]) if key in Q_expected_sarsa else -1 for key in np.arange(48)].reshape(4,12)
check_test.run_check('td_control_check', policy_expected_sarsa)
print("Estimated Optimal Policy UP = 0, RIGHT = 1, DOWN = 2, LEFT = 3, N/A = -1:")
print(policy_expected_sarsa)

# plot the estimated optimal state-action function
plot_values([np.max(Q_expected_sarsa,axis=1).reshape(4,12)])
```

```
Episode 5000/5000
PASSED

Estimated Optimal Policy UP = 0, RIGHT = 1, DOWN = 2, LEFT = 3, N/A = -1:
[[ 1 1 1 1 1 1 1 1 1 1 1 1]
 [ 1 1 1 1 1 1 1 1 1 1 1 1]
 [ 1 1 1 1 1 1 1 1 1 1 1 1]
 [ 1 2 2 2 2 2 2 2 2 2 2 0]
/home/jeffmacklin/anaconda3-5.1/lib/python3.7/site-packages/matplotlib/cbook/deprecation.py:107: MatplotlibDeprecationWarning: Passing one of 'on' or 'off', 'true', 'false' as a boolean is deprecated; use an actual boolean (True/False) instead.
  warnings.warn(message, mplDeprecation, stacklevel=1)
/home/jeffmacklin/anaconda3-5.1/lib/python3.7/site-packages/matplotlib/cbook/deprecation.py:107: MatplotlibDeprecationWarning: Passing one of 'on' or 'off', 'true', 'false' as a boolean is deprecated; use an actual boolean (True/False) instead.
  warnings.warn(message, mplDeprecation, stacklevel=1)
/home/jeffmacklin/anaconda3-5.1/lib/python3.7/site-packages/matplotlib/cbook/deprecation.py:107: MatplotlibDeprecationWarning: Passing one of 'on' or 'off', 'true', 'false' as a boolean is deprecated; use an actual boolean (True/False) instead.
  warnings.warn(message, mplDeprecation, stacklevel=1)
/home/jeffmacklin/anaconda3-5.1/lib/python3.7/site-packages/matplotlib/cbook/deprecation.py:107: MatplotlibDeprecationWarning: Passing one of 'on' or 'off', 'true', 'false' as a boolean is deprecated; use an actual boolean (True/False) instead.
  warnings.warn(message, mplDeprecation, stacklevel=1)
```



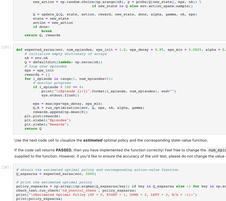
In [16]:
```
# obtain the estimated optimal policy and corresponding action-value function
Q_expected_sarsa = expected_sarsa(env, 5000)

# print the estimated optimal policy
policy_expected_sarsa = np.array([np.argmax(Q_expected_sarsa[key]) if key in Q_expected_sarsa else -1 for key in np.arange(48)].reshape(4,12)
check_test.run_check('td_control_check', policy_expected_sarsa)
print("Estimated Optimal Policy UP = 0, RIGHT = 1, DOWN = 2, LEFT = 3, N/A = -1:")
print(policy_expected_sarsa)

# plot the estimated optimal state-action function
plot_values([np.max(Q_expected_sarsa,axis=1).reshape(4,12)])
```

```
Episode 5000/5000
PASSED

Estimated Optimal Policy UP = 0, RIGHT = 1, DOWN = 2, LEFT = 3, N/A = -1:
[[ 1 1 1 1 1 1 1 1 1 1 1 1]
 [ 1 1 1 1 1 1 1 1 1 1 1 1]
 [ 1 1 1 1 1 1 1 1 1 1 1 1]
 [ 1 2 2 2 2 2 2 2 2 2 2 0]
/home/jeffmacklin/anaconda3-5.1/lib/python3.7/site-packages/matplotlib/cbook/deprecation.py:107: MatplotlibDeprecationWarning: Passing one of 'on' or 'off', 'true', 'false' as a boolean is deprecated; use an actual boolean (True/False) instead.
  warnings.warn(message, mplDeprecation, stacklevel=1)
/home/jeffmacklin/anaconda3-5.1/lib/python3.7/site-packages/matplotlib/cbook/deprecation.py:107: MatplotlibDeprecationWarning: Passing one of 'on' or 'off', 'true', 'false' as a boolean is deprecated; use an actual boolean (True/False) instead.
  warnings.warn(message, mplDeprecation, stacklevel=1)
/home/jeffmacklin/anaconda3-5.1/lib/python3.7/site-packages/matplotlib/cbook/deprecation.py:107: MatplotlibDeprecationWarning: Passing one of 'on' or 'off', 'true', 'false' as a boolean is deprecated; use an actual boolean (True/False) instead.
  warnings.warn(message, mplDeprecation, stacklevel=1)
/home/jeffmacklin/anaconda3-5.1/lib/python3.7/site-packages/matplotlib/cbook/deprecation.py:107: MatplotlibDeprecationWarning: Passing one of 'on' or 'off', 'true', 'false' as a boolean is deprecated; use an actual boolean (True/False) instead.
  warnings.warn(message, mplDeprecation, stacklevel=1)
```



In [ ]:
```
```