PA1_working_copy-1

March 9, 2022

1 Google colab

[1]:
```
try:
  import google.colab
  IN_COLAB = True
except:
  IN_COLAB = False
```

[2]:
```
if IN_COLAB:
  from google.colab import drive
  drive.mount('/content/drive')
  import os
  os.chdir('/content/drive/MyDrive/Documents/Sem6-drive/RL/Assignments/
 ↪1Assignment')
```

```
Mounted at /content/drive
```

[ ]:
```
! sudo apt-get install texlive-latex-recommended
! sudo apt-get install dvipng texlive-latex-extra texlive-fonts-recommended
! wget http://mirrors.ctan.org/macros/latex/contrib/type1cm.zip
! unzip type1cm.zip -d /tmp/type1cm
! cd /tmp/type1cm/type1cm/ && sudo latex type1cm.ins
! sudo mkdir /usr/share/texmf/tex/latex/type1cm
! sudo cp /tmp/type1cm/type1cm/type1cm.sty /usr/share/texmf/tex/latex/type1cm
! sudo texhash
! apt install cm-super
```

2 Setup

[4]:
```
UP = 0
DOWN = 1
LEFT = 2
RIGHT = 3
NUM_CONFIG = 2000
```

[ ]:
```
from math import floor
import numpy as np
from tqdm import tqdm
import pandas as pd


from functools import partial

from time import sleep
import itertools
from collections import namedtuple

# plotting libraies
import matplotlib
matplotlib.rcParams['text.usetex'] = True
import matplotlib.pyplot as plt
from matplotlib import colors
import matplotlib.patches as mpatches

# !pip install wandb --upgrade
# import wandb
# wandb.login()

# %matplotlib inline

!pip install ipywidgets
!pip install optuna

import ipywidgets as widgets
from IPython import display
from IPython.display import clear_output


def row_col_to_seq(row_col, num_cols):  #Converts state number to row_column␣
 ↪format
    return row_col[:,0] * num_cols + row_col[:,1]

def seq_to_col_row(seq, num_cols): #Converts row_column format to state number
    r = floor(seq / num_cols)
    c = seq - r * num_cols
    return np.array([[r, c]])
class GridWorld:
    """
    Creates a gridworld object to pass to an RL algorithm.
    Parameters
    ----------
    num_rows : int
```

```
        The number of rows in the gridworld.
    num_cols : int
        The number of cols in the gridworld.
    start_state : numpy array of shape (1, 2), np.array([[row, col]])
        The start state of the gridworld (can only be one start state)
    goal_states : numpy arrany of shape (n, 2)
        The goal states for the gridworld where n is the number of goal
        states.
    """
    def __init__(self, num_rows, num_cols, start_state, goal_states, wind =␣
↪False):
        self.num_rows = num_rows
        self.num_cols = num_cols
        self.start_state = start_state
        self.goal_states = goal_states
        self.obs_states = None
        self.bad_states = None
        self.num_bad_states = 0
        self.p_good_trans = None
        self.bias = None
        self.r_step = None
        self.r_goal = None
        self.r_dead = None
        self.gamma = 1 # default is no discounting
        self.wind = wind

    def add_obstructions(self, obstructed_states=None, bad_states=None,␣
↪restart_states=None):

        self.obs_states = obstructed_states
        self.bad_states = bad_states
        if bad_states is not None:
            self.num_bad_states = bad_states.shape[0]
        else:
            self.num_bad_states = 0
        self.restart_states = restart_states
        if restart_states is not None:
            self.num_restart_states = restart_states.shape[0]
        else:
            self.num_restart_states = 0

    def add_transition_probability(self, p_good_transition, bias):

        self.p_good_trans = p_good_transition
        self.bias = bias
```

```
   def add_rewards(self, step_reward, goal_reward, bad_state_reward=None,␣
↪restart_state_reward = None):

       self.r_step = step_reward
       self.r_goal = goal_reward
       self.r_bad = bad_state_reward
       self.r_restart = restart_state_reward


   def create_gridworld(self):

       self.num_actions = 4
       self.num_states = self.num_cols * self.num_rows# +1
       self.start_state_seq = row_col_to_seq(self.start_state, self.num_cols)
       self.goal_states_seq = row_col_to_seq(self.goal_states, self.num_cols)

       # rewards structure
       self.R = self.r_step * np.ones((self.num_states, 1))
       #self.R[self.num_states-1] = 0
       self.R[self.goal_states_seq] = self.r_goal

       for i in range(self.num_bad_states):
           if self.r_bad is None:
               raise Exception("Bad state specified but no reward is given")
           bad_state = row_col_to_seq(self.bad_states[i,:].reshape(1,-1), self.
↪num_cols)
           #print("bad states", bad_state)
           self.R[bad_state, :] = self.r_bad
       for i in range(self.num_restart_states):
           if self.r_restart is None:
               raise Exception("Restart state specified but no reward is␣
↪given")
           restart_state = row_col_to_seq(self.restart_states[i,:].
↪reshape(1,-1), self.num_cols)
           #print("restart_state", restart_state)
           self.R[restart_state, :] = self.r_restart

       # probability model
       if self.p_good_trans == None:
           raise Exception("Must assign probability and bias terms via the␣
↪add_transition_probability method.")

       self.P = np.zeros((self.num_states,self.num_states,self.num_actions))
       for action in range(self.num_actions):
           for state in range(self.num_states):
```

```
                # check if the state is the goal state or an obstructed state - transition to end
                row_col = seq_to_col_row(state, self.num_cols)
                if self.obs_states is not None:
                    end_states = np.vstack((self.obs_states, self.goal_states))
                else:
                    end_states = self.goal_states

                if any(np.sum(np.abs(end_states-row_col), 1) == 0):
                    self.P[state, state, action] = 1

                # else consider stochastic effects of action
                else:
                    for dir in range(-1,2,1):

                        direction = self._get_direction(action, dir)
                        next_state = self._get_state(state, direction)
                        if dir == 0:
                            prob = self.p_good_trans
                        elif dir == -1:
                            prob = (1 - self.p_good_trans)*(self.bias)
                        elif dir == 1:
                            prob = (1 - self.p_good_trans)*(1-self.bias)

                        self.P[state, next_state, action] += prob

                # make restart states transition back to the start state with
                # probability 1
                if self.restart_states is not None:
                    if any(np.sum(np.abs(self.restart_states-row_col),1)==0):
                        next_state = row_col_to_seq(self.start_state, self.num_cols)
                        self.P[state,:,:] = 0
                        self.P[state,next_state,:] = 1
        return self

    def _get_direction(self, action, direction):

        left = [2,3,1,0]
        right = [3,2,0,1]
        if direction == 0:
            new_direction = action
        elif direction == -1:
            new_direction = left[action]
        elif direction == 1:
            new_direction = right[action]
        else:
```

```
            raise Exception("getDir received an unspecified case")
        return new_direction

    def _get_state(self, state, direction):

        row_change = [-1,1,0,0]
        col_change = [0,0,-1,1]
        row_col = seq_to_col_row(state, self.num_cols)
        row_col[0,0] += row_change[direction]
        row_col[0,1] += col_change[direction]

        # check for invalid states
        if self.obs_states is not None:
            if (np.any(row_col < 0) or
                np.any(row_col[:,0] > self.num_rows-1) or
                np.any(row_col[:,1] > self.num_cols-1) or
                np.any(np.sum(abs(self.obs_states - row_col), 1)==0)):
                next_state = state
            else:
                next_state = row_col_to_seq(row_col, self.num_cols)[0]
        else:
            if (np.any(row_col < 0) or
                np.any(row_col[:,0] > self.num_rows-1) or
                np.any(row_col[:,1] > self.num_cols-1)):
                next_state = state
            else:
                next_state = row_col_to_seq(row_col, self.num_cols)[0]

        return next_state

    def reset(self):
      return int(self.start_state_seq)

    def step(self, state, action):
        p, r = 0, np.random.random()
        for next_state in range(self.num_states):

            p += self.P[state, next_state, action]

            if r <= p:
                break

        if(self.wind and np.random.random() < 0.4):

          arr = self.P[next_state, :, 3]
          next_next = np.where(arr == np.amax(arr))
          next_next = next_next[0][0]
```

```
            return next_next, self.R[next_next]
        else:
            return next_state, self.R[next_state]
```

[6]:
```
# specify world parameters
num_cols = 10
num_rows = 10
obstructions = np.array([[0,7],[1,1],[1,2],[1,3],[1,7],[2,1],[2,3],
                         [2,7],[3,1],[3,3],[3,5],[4,3],[4,5],[4,7],
                         [5,3],[5,7],[5,9],[6,3],[6,9],[7,1],[7,6],
                         [7,7],[7,8],[7,9],[8,1],[8,5],[8,6],[9,1]])
bad_states = np.array([[1,9],[4,2],[4,4],[7,5],[9,9]])
restart_states = np.array([[3,7],[8,2]])
start_state = np.array([[3,6]])
goal_states = np.array([[0,9],[2,2],[8,7]])

# create model
gw = GridWorld(num_rows=num_rows,
               num_cols=num_cols,
               start_state=start_state,
               goal_states=goal_states, wind = False)
gw.add_obstructions(obstructed_states=obstructions,
                    bad_states=bad_states,
                    restart_states=restart_states)
gw.add_rewards(step_reward=-1,
               goal_reward=10,
               bad_state_reward=-6,
               restart_state_reward=-100)
gw.add_transition_probability(p_good_transition=0.7,
                              bias=0.5)
env = gw.create_gridworld()
```

[7]:
```
print("Number of actions", env.num_actions) #0 -> UP, 1-> DOWN, 2 -> LEFT, 3->␣
 ↪RIGHT
print("Number of states", env.num_states)
print("start state", env.start_state_seq)
print("goal state(s)", env.goal_states_seq)
```

```
Number of actions 4
Number of states 100
start state [36]
goal state(s) [ 9 22 87]
```

3 Plotting Helper Functions

[8]:
```
def plot_Q(Q, message = "Q plot"):

    # plt.figure(figsize=(10,10))
    fig, ax = plt.subplots(figsize = (10,10))

    Q_max = Q.max(-1)
    im = ax.imshow(Q_max, extent=[0, 10, 0, 10])

    cbar = ax.figure.colorbar(im, ax=ax)

    ax.set_title(message)
    #ax.grid(visible = True, which="major", color="w", linestyle='-',␣
↪linewidth=2)
    ax.set_xlim([0, 10])
    ax.set_ylim([0,10])
    ax.pcolor(Q.max(-1), edgecolors='k', linewidths=1)
    #plt.colorbar()

    # plt.colorbar()
    def x_direct(a):
        if a in [UP, DOWN]:
            return 0
        return 1 if a == RIGHT else -1
    def y_direct(a):
        if a in [RIGHT, LEFT]:
            return 0
        return 1 if a == UP else -1
    policy = Q.argmax(-1)
    policyx = np.vectorize(x_direct)(policy)
    policyy = np.vectorize(y_direct)(policy)
    idx = np.indices(policy.shape)
    ax.quiver(idx[1].ravel()+ 0.5, idx[0].ravel()+0.5, policyx.ravel(), policyy.
↪ravel(), pivot="middle", color='red')
    fig.tight_layout()
    plt.savefig(f'./plots/{NUM_CONFIG}/b3.png')
```

[26]:
```
def render_env(env, state, ax = None, render_agent = True, leg = True):
        grid = np.zeros((env.num_rows, env.num_cols))
        for start in env.start_state:
            grid[start[0], start[1]] = 1 # #0066ff blue color
        for goal in env.goal_states:
            grid[goal[0], goal[1]] = 2  #66ff66 - green color
        for obs in env.obs_states:
            grid[obs[0], obs[1]] = 3   #ff3300 - red color
        for bad in env.bad_states:
```

```
            grid[bad[0], bad[1]] = 4  #ffff66 - yellow color
        for restart in env.restart_states:
            grid[restart[0], restart[1]] = 5    #ff6600 - orange color

        if render_agent:
            grid[state[0], state[1]] = 6 #000000 - black color




        # creating legnd with color box
        start_state = mpatches.Patch(color='#0066ff', label='Start')
        goal_state = mpatches.Patch(color='#66ff66', label='Goal')
        obs_state = mpatches.Patch(color='#ff3300', label='Obstructed')
        bad_state = mpatches.Patch(color='#ffff66', label='Bad')
        restart_state = mpatches.Patch(color='#ff6600', label='Restart')
        agent = mpatches.Patch(color='#000000', label='Agent')
        #plt.legend(handles=[start_state, goal_state, obs_state, bad_state,␣
↪restart_state])
        if render_agent:
            cmap = colors.ListedColormap(['#ffffff', '#0066ff', '#66ff66',␣
↪'#ff3300', '#ffff66', '#ff6600', '#000000'])
        else:
            cmap = colors.ListedColormap(['#ffffff', '#0066ff', '#66ff66',␣
↪'#ff3300', '#ffff66', '#ff6600'])

        if ax is None:
            fig, ax = plt.subplots()
            fig.set_size_inches(10,10)

        if leg:
            ax.legend(bbox_to_anchor = (1.15, 1), handles=[start_state,␣
↪goal_state, obs_state, bad_state, restart_state, agent])

        ax.pcolor(grid, cmap=cmap, edgecolors='k', linewidths=2)
        return ax



def visualize_policy(env, Q, ax = None, plot = True):
  state_seq = env.reset()
  state = seq_to_col_row(state_seq, env.num_rows)[0]

  visited_states = [[state[1], state[0]]]

  done = False
  steps = 0
```

```
 total_reward = 0
 while not done:

     state_next_seq, reward = env.step(state_seq, Q[state[0], state[1]].
↪argmax())
     state_next = seq_to_col_row(state_next_seq, env.num_rows)[0]

     visited_states.append([state_next[1], state_next[0]])

     state = state_next
     state_seq = row_col_to_seq(np.expand_dims(state, axis = 0), env.num_cols )


     if np.any(np.sum(abs(env.goal_states - np.expand_dims(state_next, axis =␣
↪0)), 1)==0): done = True


     steps += 1
     total_reward += reward

     if steps == 500:
       done = True

 if plot :
   visited_states = np.array(visited_states, dtype = float)
   #visited_states = visited_states + ( np.random.rand(*visited_states.shape) /
↪ 3 )
   visited_states += 0.5
   dpath = visited_states[1:] -visited_states[0:-1]
   if ax == None:
     fig, ax = plt.subplots()
     fig.set_size_inches(10, 10)

   ax = render_env(env, state = None, ax = ax, render_agent = False, leg =␣
↪False)
   for start, diff in zip(visited_states, dpath):
     arrow = mpatches.Arrow(*start,  *diff, width = 0.15, color = 'k')
     ax.add_patch(arrow)
   ax.set_title("Steps: %d, Total Reward: %d"%(steps, total_reward))
 return total_reward, steps, ax
```

[10]:
```
def plot_greed_variations(alg, policy, b_alpha, l_alpha, b_beta, l_beta,␣
↪b_gamma, l_gamma, b_epsilon, l_epsilon):
```

```
    config_settings = configurations_l[NUM_CONFIG]

    # create environment
    env = create_env(**config_settings._asdict())

    rewards_l = []
    fig, ax = plt.subplots(2, 2, figsize=(7, 7))
    if policy == 'softmax':
        betas_full_list = [b_beta] + l_beta
        choose_action = choose_action_softmax
        for beta in betas_full_list:
            if alg == 'sarsa':
                Q, rewards, steps, env = sarsa(env,  episodes =2000, alpha0 =␣
↪b_alpha,  epsilon0 = b_epsilon, beta = beta, gamma = b_gamma, plot_heat =␣
↪False, choose_action = choose_action, wandb_logging = False)
                rewards_l.append(rewards)
            else:
                Q, rewards, steps, env = q_learning(env,  episodes =2000,␣
↪alpha0 = b_alpha,  epsilon0 = b_epsilon, beta = beta, gamma = b_gamma,␣
↪plot_heat = False, choose_action = choose_action, wandb_logging = False)
                rewards_l.append(rewards)
    else:
        epsilons_full_list = [b_epsilon] + l_epsilon
        choose_action = choose_action_epsilon
        for epsilon in epsilons_full_list:
            if alg == 'sarsa':
                Q, rewards, steps, env = sarsa(env,  episodes =2000, alpha0 =␣
↪b_alpha,  epsilon0 = epsilon, beta = b_beta, gamma = b_gamma, plot_heat =␣
↪False, choose_action = choose_action, wandb_logging = False)
                rewards_l.append(rewards)
            else:
                Q, rewards, steps, env = q_learning(env,  episodes =2000,␣
↪alpha0 = b_alpha,  epsilon0 = epsilon, beta = b_beta, gamma = b_gamma,␣
↪plot_heat = False, choose_action = choose_action, wandb_logging = False)
                rewards_l.append(rewards)

    for k, rewards in enumerate(rewards_l):
        i, j = (k//2), (k%2)

        if policy == 'softmax':
            parameter = r'$\beta$'
            # parameter = 'beta'
            full_list = betas_full_list
        else:
            parameter = r'$\epsilon$'
            # parameter = 'epsilon'
            full_list = epsilons_full_list
```

```
        axes = reward_curve(rewards, ax[i,j], label = fr'{parameter} =␣
↪{full_list[k]}', xy_labels = False)
        axes.legend(loc = 'lower right', prop = {'weight':'bold'})

        #axes.set_title(f'{alg}-{policy}-  ' + r'parameter' + f'={full_list[k]}␣
↪ gamma={b_gamma}  alpha={b_alpha}')
    a_parameter = r'$\alpha$'
    b_parameter = r'$\gamma$'
    # a_parameter = 'alpha'
    # b_parameter = 'gamma'
    fig.suptitle(fr'C-{NUM_CONFIG}-{alg}-{policy} {b_parameter} = {b_gamma} ␣
↪{a_parameter} = {b_alpha}', weight = 'bold')
    fig.text(0.5, 0.06, 'Episodes', ha='center')
    fig.text(0.05, 0.5, 'Rewards', va='center', rotation='vertical')

    plt.savefig(f'./plots/{NUM_CONFIG}/greed_variations_{alg}_{policy}.png')

def plot_lr_variations(alg, policy, b_alpha, l_alpha, b_beta, l_beta, b_gamma,␣
↪l_gamma, b_epsilon, l_epsilon):
    config_settings = configurations_l[NUM_CONFIG]

    # create environment
    env = create_env(**config_settings._asdict())

    rewards_l = []
    fig, ax = plt.subplots(2, 2, figsize=(7, 7))
    if policy == 'softmax':
        choose_action = choose_action_softmax
    else :
        choose_action = choose_action_epsilon

    lr_full_list = [b_alpha] + l_alpha
    for alpha in lr_full_list:
        if alg == 'sarsa':
            Q, rewards, steps, env = sarsa(env,  episodes =2000, alpha0 =␣
↪alpha,  epsilon0 = b_epsilon, beta = b_beta, gamma = b_gamma, plot_heat =␣
↪False, choose_action = choose_action, wandb_logging = False)
            rewards_l.append(rewards)
        else:
            Q, rewards, steps, env = q_learning(env,  episodes =2000, alpha0 =␣
↪alpha,  epsilon0 = b_epsilon, beta = b_beta, gamma = b_gamma, plot_heat =␣
↪False, choose_action = choose_action, wandb_logging = False)
            rewards_l.append(rewards)

    full_list = lr_full_list
    parameter = r'$\alpha$'
```

```
    a_parameter = r'$\gamma$'
    # parameter = 'alpha'
    # a_parameter = 'gamma'
    for k, rewards in enumerate(rewards_l):
        i, j = (k//2), (k%2)

        if policy == 'softmax':
            p_parameter = r'$\beta$'
            # p_parameter = 'beta'
            b_parameter = b_beta
        else:
            p_parameter = r'$/\epsilon$'
            # p_parameter = 'epsilon'
            b_parameter = b_epsilon
        axes = reward_curve(rewards, ax[i,j], label = fr'{parameter} =␣
↪{full_list[k]}', xy_labels = False)
        axes.legend(loc = 'lower right', prop = {'weight':'bold'})

        #axes.set_title(f'{alg}-{policy}-  ' + r'parameter' + f'={full_list[k]}␣
↪ gamma={b_gamma}  alpha={b_alpha}')

    fig.suptitle(fr'C-{NUM_CONFIG}-{alg}-{policy} {a_parameter} = {b_gamma} ␣
↪{p_parameter} = {b_parameter}', weight = 'bold')
    fig.text(0.5, 0.06, 'Episodes', ha='center')
    fig.text(0.05, 0.5, 'Rewards', va='center', rotation='vertical')

    plt.savefig(f'./plots/{NUM_CONFIG}/lr_variations_{alg}_{policy}.png')

def plot_gamma_variations(alg, policy, b_alpha, l_alpha, b_beta, l_beta,␣
↪b_gamma, l_gamma, b_epsilon, l_epsilon):
    config_settings = configurations_l[NUM_CONFIG]

    # create environment
    env = create_env(**config_settings._asdict())

    rewards_l = []
    fig, ax = plt.subplots(2, 2, figsize=(7, 7))
    if policy == 'softmax':
        choose_action = choose_action_softmax
    else :
        choose_action = choose_action_epsilon

    gamma_full_list = [b_gamma] + l_gamma
    for gamma in gamma_full_list:
        if alg == 'sarsa':
```

```
            Q, rewards, steps, env = sarsa(env,  episodes =2000, alpha0 =␣
↪b_alpha,  epsilon0 = b_epsilon, beta = b_beta, gamma = gamma, plot_heat =␣
↪False, choose_action = choose_action, wandb_logging = False)
            rewards_l.append(rewards)
        else:
            Q, rewards, steps, env = q_learning(env,  episodes =2000, alpha0 =␣
↪b_alpha,  epsilon0 = b_epsilon, beta = b_beta, gamma = gamma, plot_heat =␣
↪False, choose_action = choose_action, wandb_logging = False)
            rewards_l.append(rewards)

    full_list = gamma_full_list
    parameter = r'$\gamma$'
    a_parameter = r'$\alpha$'
    # parameter = 'gamma'
    # a_parameter = 'alpha'
    for k, rewards in enumerate(rewards_l):
        i, j = (k//2), (k%2)

        if policy == 'softmax':
            p_parameter = r'$\beta$'
            b_parameter = b_beta
        else:
            p_parameter = r'$\epsilon$'
            b_parameter = b_epsilon
        axes = reward_curve(rewards, ax[i,j], label = fr'{parameter} =␣
↪{full_list[k]}', xy_labels = False)
        axes.legend(loc = 'lower right', prop = {'weight':'bold'})

        #axes.set_title(f'{alg}-{policy}-  ' + r'parameter' + f'={full_list[k]}␣
↪ gamma={b_gamma}  alpha={b_alpha}')

    fig.suptitle(fr'C-{NUM_CONFIG}-{alg}-{policy} {a_parameter} = {b_alpha} ␣
↪{p_parameter} = {b_parameter}', weight = 'bold')
    fig.text(0.5, 0.06, 'Episodes', ha='center')
    fig.text(0.05, 0.5, 'Rewards', va='center', rotation='vertical')

    plt.savefig(f'./plots/{NUM_CONFIG}/gamma_variations_{alg}_{policy}.png')
```

[11]:
```
def best_plots(algorithm, policy, b_alpha,  b_epsilon, b_beta , b_gamma ):
    # create env
    config_settings = configurations_l[NUM_CONFIG]

    # create environment
    env = create_env(**config_settings._asdict())

    # train the agent in env
    if policy == 'softmax':
```

```
        choose_action = choose_action_softmax
    else:
        choose_action = choose_action_epsilon

    if algorithm == 'sarsa':
        alg = sarsa
    else:
        alg = q_learning

    Q, rewards, steps, env = alg(env,  episodes =2000, alpha0 = b_alpha, ␣
↪epsilon0 = b_epsilon, gamma = b_gamma, plot_heat = False, choose_action =␣
↪choose_action, wandb_logging = False)


    # draw reward curve
    fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(7, 14))
    ax1 = reward_curve(rewards, ax1, label = 'Reward Curve', xy_labels = True)
    ax2 = steps_taken(steps, ax2, label = 'Steps Taken', xy_labels = True)
    plt.savefig(f'./plots/{NUM_CONFIG}/b1.png')


    # draw Image of grid world with policy for 2 runs
    fig, (ax1, ax2) = plt.subplots(2, 1,figsize=(7, 14))

    _, _, ax1 = visualize_policy(env, Q, ax1, plot = True)
    _, _, ax2 = visualize_policy(env, Q, ax2, plot = True)
    plt.savefig(f'./plots/{NUM_CONFIG}/b2.png')

    # plt.clf()
    plot_Q(Q)
    return Q
```

[12]:
```
def plot_all(b_alpha, l_alpha, b_beta, l_beta, b_gamma, l_gamma, b_epsilon,␣
↪l_epsilon):

    # plot for softmax & sarsa
    # plot for epsilon greedy & softmax
        # plot for epsion / temp
        # plot for learning_rate
        # plot for gamma

    for alg in ['Q-learning', 'sarsa']:
        for policy in ['e-greedy', 'softmax']:
            plot_greed_variations(alg, policy, b_alpha, l_alpha, b_beta,␣
↪l_beta, b_gamma, l_gamma, b_epsilon, l_epsilon)
            plot_lr_variations(alg, policy, b_alpha, l_alpha, b_beta, l_beta,␣
↪b_gamma, l_gamma, b_epsilon, l_epsilon)
```

```
            plot_gamma_variations(alg, policy, b_alpha, l_alpha, b_beta,␣
 ↪l_beta, b_gamma, l_gamma, b_epsilon, l_epsilon)
```

[13]:
```
def reward_curve(episode_rewards, ax = None, label  = None, xy_labels = True):
  if ax == None:
    ax = plt.gca()
  if xy_labels:
    ax.set_xlabel('Episode')
    ax.set_ylabel('Total Reward')
  ax.plot(np.arange(len(episode_rewards)),episode_rewards, label = label)
  #plt.show()
  return ax
```

[14]:
```
def steps_taken(episode_steps, ax = None, label  = None, xy_labels = True):
  if ax == None:
    ax = plt.gca()
  if xy_labels:
    ax.set_xlabel('Episode')
    ax.set_ylabel('Steps Taken')
  ax.plot(np.arange(len(episode_steps)),episode_steps, label = label)
  #plt.show()
  return ax
```

4 Algorithms and Helper Functions

4.0.1 Exploration strategies

1. Epsilon-greedy
2. Softmax

[15]:
```
from scipy.special import softmax

seed = 42
rg = np.random.RandomState(seed)

# Epsilon greedy
def choose_action_epsilon(Q, state, epsilon, rg=rg):
    if not Q[state[0], state[1]].any() or rg.rand() < epsilon:
        return rg.choice(Q.shape[-1])
    else:
        return np.argmax(Q[state[0], state[1]])

# Softmax
def choose_action_softmax(Q, state, beta = 1, rg=rg):
    probs = np.nan_to_num(softmax( np.nan_to_num(Q[state[0], state[1]] / beta␣
 ↪)))
    probs /= probs.sum()  # normalize
    return rg.choice(Q.shape[-1], p = probs)
```

4.1 SARSA

Now we implement the SARSA algorithm.

Recall the update rule for SARSA:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)] \tag{1}$$

4.1.1 Hyperparameters

So we have som hyperparameters for the algorithm: - α - number of *episodes*. - ϵ: For epsilon greedy exploration

[16]:
```
def sarsa(env, episodes = 2000, alpha0 = 0.4,  epsilon0 = 0.1, beta = 1, gamma␣
↪= 0.9, plot_heat = False, choose_action = choose_action_softmax, print_freq␣
↪= 100, max_steps = 100, wandb_logging = False):

    Q = np.zeros((env.num_rows, env.num_cols, env.num_actions))

    # Adding code to display experiment parameters
    parameters = dict()

    parameters['Algorithm'] = 'Sarsa'
    parameters['episodes'] = episodes
    parameters['policy'] = 'Softmax' if choose_action == choose_action_softmax␣
↪else 'Epsilon Greedy'
    parameters['gamma'] = gamma
    parameters['epsilon'] = epsilon0
    parameters['alpha'] = alpha0


    df = pd.DataFrame(parameters, index=[0])

    # For epsilon greedy
    if choose_action == choose_action_epsilon:
        choose_action = partial( choose_action_epsilon, epsilon = epsilon0)
    if choose_action == choose_action_softmax:
        choose_action = partial( choose_action_softmax, beta = beta)


    ########
    episode_rewards = np.zeros(episodes)
    steps_to_completion = np.zeros(episodes)
    if plot_heat:
        clear_output(wait=True)
        plot_Q(Q)
    epsilon = epsilon0
    alpha = alpha0
    for ep in tqdm(range(episodes)):
```

```
        tot_reward, steps = 0, 0

        # Reset environment
        state_seq = env.reset()
        state = seq_to_col_row(state_seq, env.num_cols )[0]

        action = choose_action(Q, state)
        done = False
        while not done:
            state_next_seq, reward = env.step(state_seq, action)
            state_next = seq_to_col_row(state_next_seq, env.num_cols)[0]
            action_next = choose_action(Q, state_next)

            # update equation
            Q[state[0], state[1], action] += alpha*(reward +␣
↪gamma*Q[state_next[0], state_next[1], action_next] - Q[state[0], state[1],␣
↪action])

            tot_reward += reward
            steps += 1

            if steps == max_steps or np.any(np.sum(abs(env.goal_states - np.
↪expand_dims(state_next, axis = 0)), 1)==0): done = True

            state, action = state_next, action_next
            state_seq = row_col_to_seq(np.expand_dims(state, axis = 0), env.
↪num_cols )

        episode_rewards[ep] = tot_reward
        steps_to_completion[ep] = steps

        if wandb_logging:
            wandb.log({'Reward': tot_reward, 'Steps': steps})

        if (ep+1)%print_freq == 0 and plot_heat:
            clear_output(wait=True)
            plot_Q(Q, message = "Episode %d: Reward: %f, Steps: %.2f, Qmax: %.
↪2f, Qmin: %.2f"%(ep+1, np.mean(episode_rewards[ep-print_freq+1:ep]),
                                                                        np.
↪mean(steps_to_completion[ep-print_freq+1:ep]),
                                                                        Q.
↪max(), Q.min()))
    #print(df)
    return Q, episode_rewards, steps_to_completion, env
```

4.2 Q-Learning

Now, implement the Q-Learning algorithm as an exercise.

Recall the update rule for Q-Learning:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_t + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)] \tag{2}$$

Visualize and compare results with SARSA.

[17]:
```
def q_learning(env, episodes = 2000, alpha0 = 0.4,  epsilon0 = 0.1, beta = 1,␣
↪gamma = 0.9, plot_heat = False, choose_action = choose_action_softmax,␣
↪print_freq = 100, max_steps = 100, wandb_logging = False):

    Q = np.zeros((env.num_rows, env.num_cols, env.num_actions))

    # Adding code to display experiment parameters
    parameters = dict()

    parameters['Algorithm'] = 'Q-learning'
    parameters['episodes'] = episodes
    parameters['policy'] = 'Softmax' if choose_action == choose_action_softmax␣
↪else 'Epsilon Greedy'
    parameters['gamma'] = gamma
    parameters['epsilon'] = epsilon0
    parameters['alpha'] = alpha0


    df = pd.DataFrame(parameters, index=['Values'])

    # For epsilon greedy
    if choose_action == choose_action_epsilon:
        choose_action = partial( choose_action_epsilon, epsilon = epsilon0)
    if choose_action == choose_action_softmax:
        choose_action = partial( choose_action_softmax, beta = beta)

    ########
    episode_rewards = np.zeros(episodes)
    steps_to_completion = np.zeros(episodes)
    if plot_heat:
        clear_output(wait=True)
        plot_Q(Q)
    epsilon = epsilon0
    alpha = alpha0
    for ep in tqdm(range(episodes)):
        tot_reward, steps = 0, 0

        # Reset environment
        state_seq = env.reset()
        state = seq_to_col_row(state_seq, env.num_cols )[0]
```

```
        #action = choose_action(Q, state)
        done = False
        while not done:
            action = choose_action(Q, state)

            state_next_seq, reward = env.step(state_seq, action)
            state_next = seq_to_col_row(state_next_seq, env.num_cols)[0]

            action_next = np.argmax(Q[state_next[0], state_next[1]])

            # update equation
            Q[state[0], state[1], action] += alpha*(reward +␣
↪gamma*Q[state_next[0], state_next[1], action_next] - Q[state[0], state[1],␣
↪action])

            tot_reward += reward
            steps += 1

            if steps == max_steps or np.any(np.sum(abs(env.goal_states - np.
↪expand_dims(state_next, axis = 0)), 1)==0): done = True

            state, action = state_next, action_next
            state_seq = row_col_to_seq(np.expand_dims(state, axis = 0), env.
↪num_cols )

        episode_rewards[ep] = tot_reward
        steps_to_completion[ep] = steps

        if wandb_logging:
            wandb.log({'Reward': tot_reward, 'Steps': steps})

        if (ep+1)%print_freq == 0 and plot_heat:
            clear_output(wait=True)
            plot_Q(Q, message = "Episode %d: Reward: %f, Steps: %.2f, Qmax: %.
↪2f, Qmin: %.2f"%(ep+1, np.mean(episode_rewards[ep-print_freq+1:ep]),
                                                                        np.
↪mean(steps_to_completion[ep-print_freq+1:ep]),
                                                                        Q.
↪max(), Q.min()))
    #display.display( df.T )
    return Q, episode_rewards, steps_to_completion, env
```

5 Config Setup

[18]:
```
wind = [True, False]
start_states = np.array([ [[0,4]], [[3,6]] ])
p = [1.0, 0.7, 0.35]

configurations_l = list(itertools.product(wind, start_states, p))
```

[19]:
```
env_config = namedtuple("env_config", "wind start_state p")
configurations_l = [env_config(wind, start_state, p) for wind, start_state, p␣
 ↪in configurations_l]
```

[20]:
```
configurations_l
```

[20]:
```
[env_config(wind=True, start_state=array([[0, 4]]), p=1.0),
 env_config(wind=True, start_state=array([[0, 4]]), p=0.7),
 env_config(wind=True, start_state=array([[0, 4]]), p=0.35),
 env_config(wind=True, start_state=array([[3, 6]]), p=1.0),
 env_config(wind=True, start_state=array([[3, 6]]), p=0.7),
 env_config(wind=True, start_state=array([[3, 6]]), p=0.35),
 env_config(wind=False, start_state=array([[0, 4]]), p=1.0),
 env_config(wind=False, start_state=array([[0, 4]]), p=0.7),
 env_config(wind=False, start_state=array([[0, 4]]), p=0.35),
 env_config(wind=False, start_state=array([[3, 6]]), p=1.0),
 env_config(wind=False, start_state=array([[3, 6]]), p=0.7),
 env_config(wind=False, start_state=array([[3, 6]]), p=0.35)]
```

[21]:
```
def create_env(wind, start_state, p):
    # create model
    gw = GridWorld(num_rows=num_rows,
                num_cols=num_cols,
                start_state=start_state,
                goal_states=goal_states, wind = wind)
    gw.add_obstructions(obstructed_states=obstructions,
                        bad_states=bad_states,
                        restart_states=restart_states)
    gw.add_rewards(step_reward=-1,
                goal_reward=10,
                bad_state_reward=-6,
                restart_state_reward=-100)

    gw.add_transition_probability(p_good_transition=p,
                                  bias=0.5)
    env = gw.create_gridworld()
    return env
```

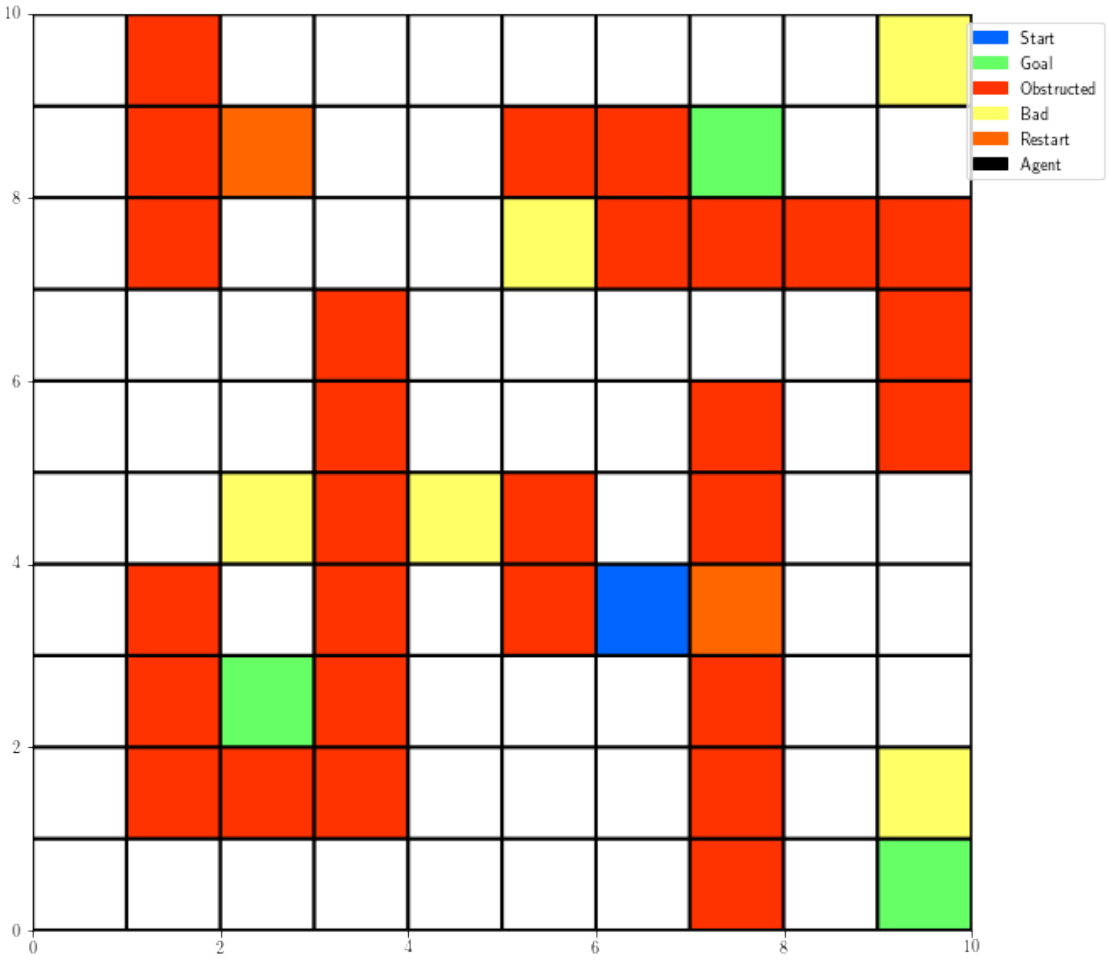
[22]:
```
def run(config= None):

    wandb_logging = True # set the global logging to true
```

```
    config_settings = configurations_1[NUM_CONFIG]

    # create environment
    env = create_env(**config_settings._asdict())

    with wandb.init(config=config,  tags = [str(NUM_CONFIG)]):
        if config == None:
            config = wandb.config

        if config.policy == 'softmax':
            choose_action = partial(choose_action_softmax, beta = config.beta)
        else:
            choose_action = partial(choose_action_epsilon, epsilon = config.
↪epsilon)

        if config.algorithm == 'sarsa':
            Q, rewards, steps, env = sarsa(env,  episodes =2000, alpha0 =␣
↪config.alpha,  epsilon0 = config.epsilon, gamma = config.gamma, plot_heat =␣
↪False, choose_action = choose_action, wandb_logging = True)
        else:
            Q, rewards, steps, env = q_learning(env,  episodes =2000, alpha0 =␣
↪config.alpha,  epsilon0 = config.epsilon, gamma = config.gamma, plot_heat =␣
↪False, choose_action = choose_action, wandb_logging = True)

        wandb.log({"avg reward(train)": np.average(rewards), "avg steps(train)"␣
↪: np.average(steps)})

        test_rewards, test_steps = [] , []
        for _ in range(100):
          reward, steps, _ = visualize_policy(env, Q, None, False)
          test_rewards.append(reward)
          test_steps.append(steps)
        wandb.log({"avg reward(test)": np.average(test_rewards), "avg␣
↪steps(test)" : np.average(test_steps)})
```

6 Experiments

6.1 Visualize Environemtn

[27]:
```
ax = render_env(env, state=None, render_agent=False)
plt.savefig('./plots/main/env_init.png')
plt.show()
```



6.2 Configurations

6.3 Config 0

[ ]:
```
NUM_CONFIG = 0

config_settings = configurations_l[NUM_CONFIG]

# create environment
env = create_env(**config_settings._asdict())

sweep_config = {
    "name" : f"{NUM_CONFIG}-config-sweep",
    "method": "random",
    "parameters": {
        "algorithm": {
            "values": ['sarsa', 'q_learning'],
```

```
        },
        "policy": {
            "values": ['softmax', 'epsilon_greedy'],
        },
        "epsilon": {
            "min": 0.0,
            "max": 1.0,
        },
        "alpha": {
            "min": 0.01,
            "max": 0.2,
        },
        "gamma": {
            "min": 0.5,
            "max": 1.0,
        },
        "beta": {
            "min": 0.5,
            "max": 1.5,
        }
    }
}
```

[ ]:
```
sweep_id = wandb.sweep(sweep_config, project='RLPA1')
```

[ ]:
```
wandb.agent(sweep_id, run, count=20)
```

6.3.1 Best HP

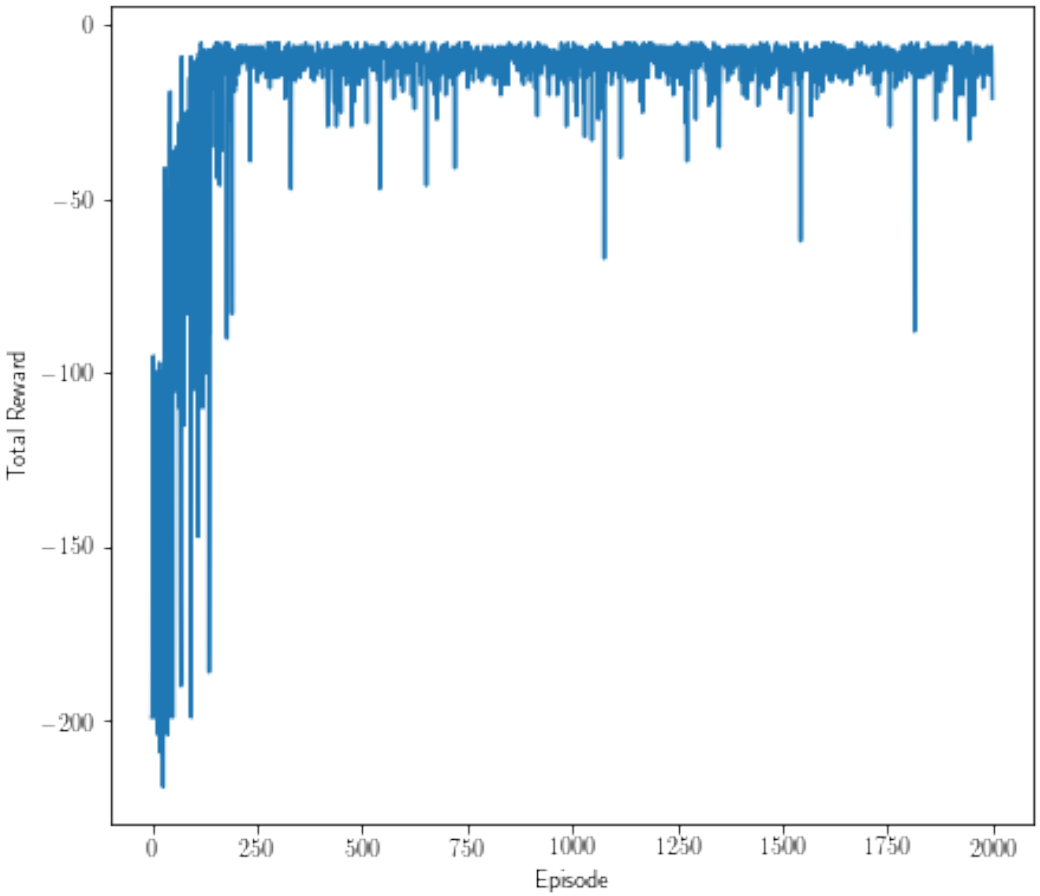
[ ]:
```
# swwep 19

alpha = 0.1022
l_alpha = [0.08, 0.11, 0.09]
beta = 0.6916
l_beta = [0.5, 0.8, 0.9]
gamma = 0.9364
l_gamma = [0.85, 0.9, 1.0]
epsilon = 0.06584
l_epsilon = [0.1, 0.2, 0.03]
policy = 'e-greedy'
algorithm = 'sarsa'
```

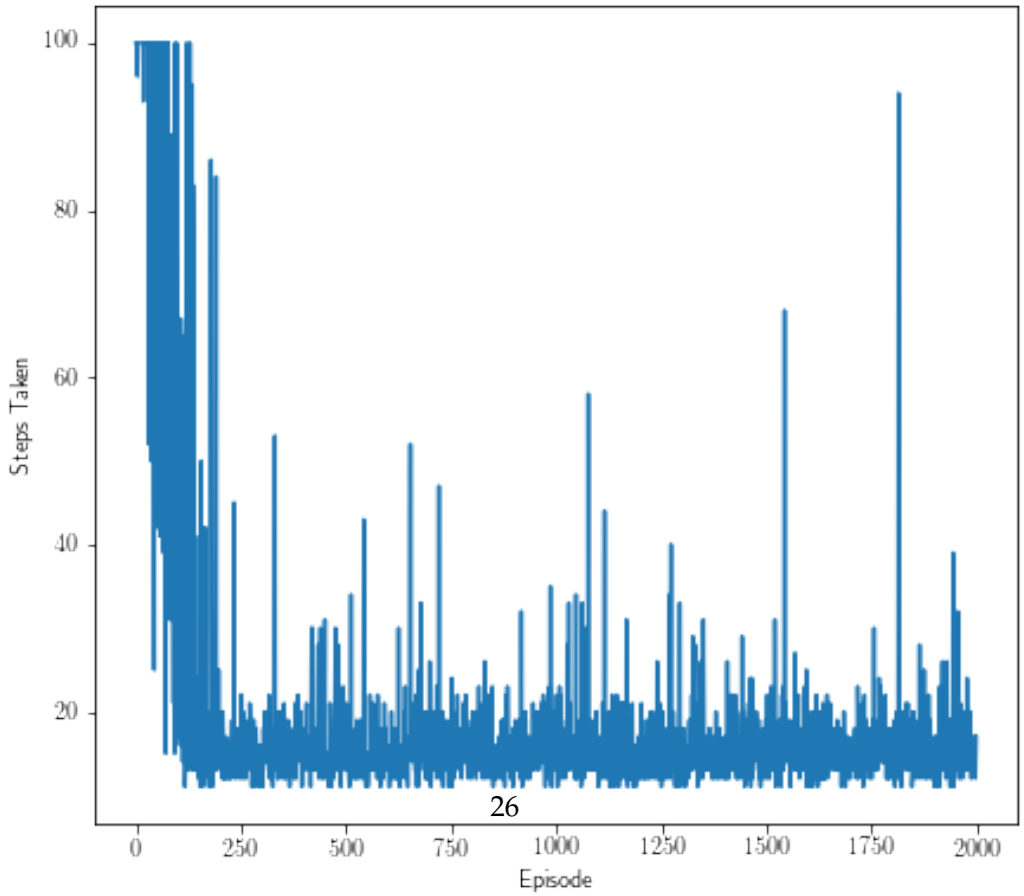
6.3.2 Plotting

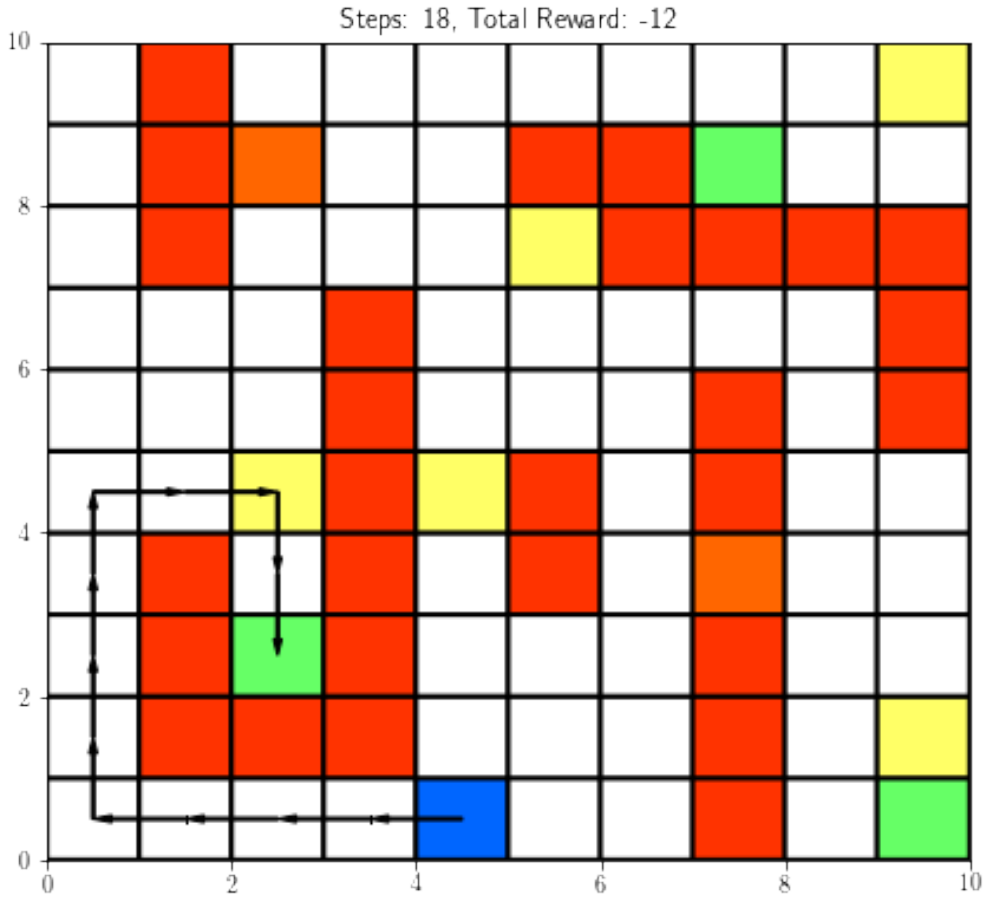
[ ]:
```
plot_all(alpha, l_alpha, beta, l_beta, gamma, l_gamma, epsilon, l_epsilon)
```

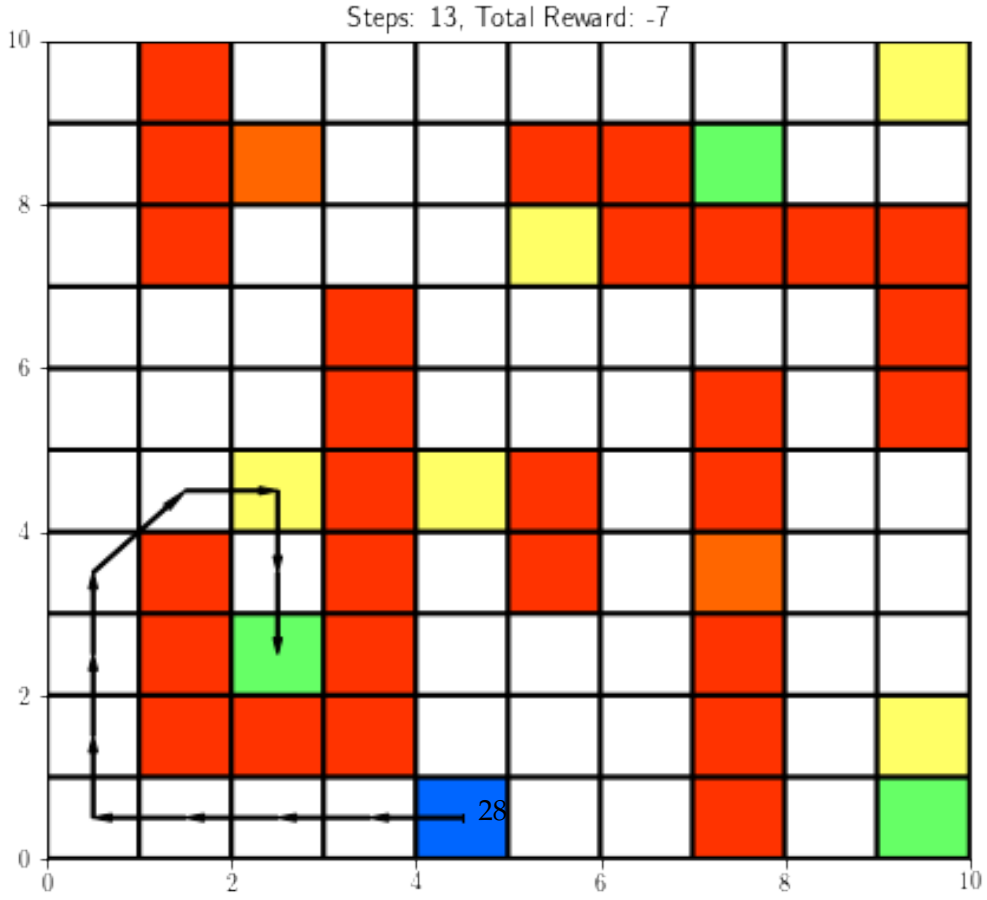
[ ]:
```
Q = best_plots(algorithm, policy, alpha,  epsilon, beta , gamma )
```

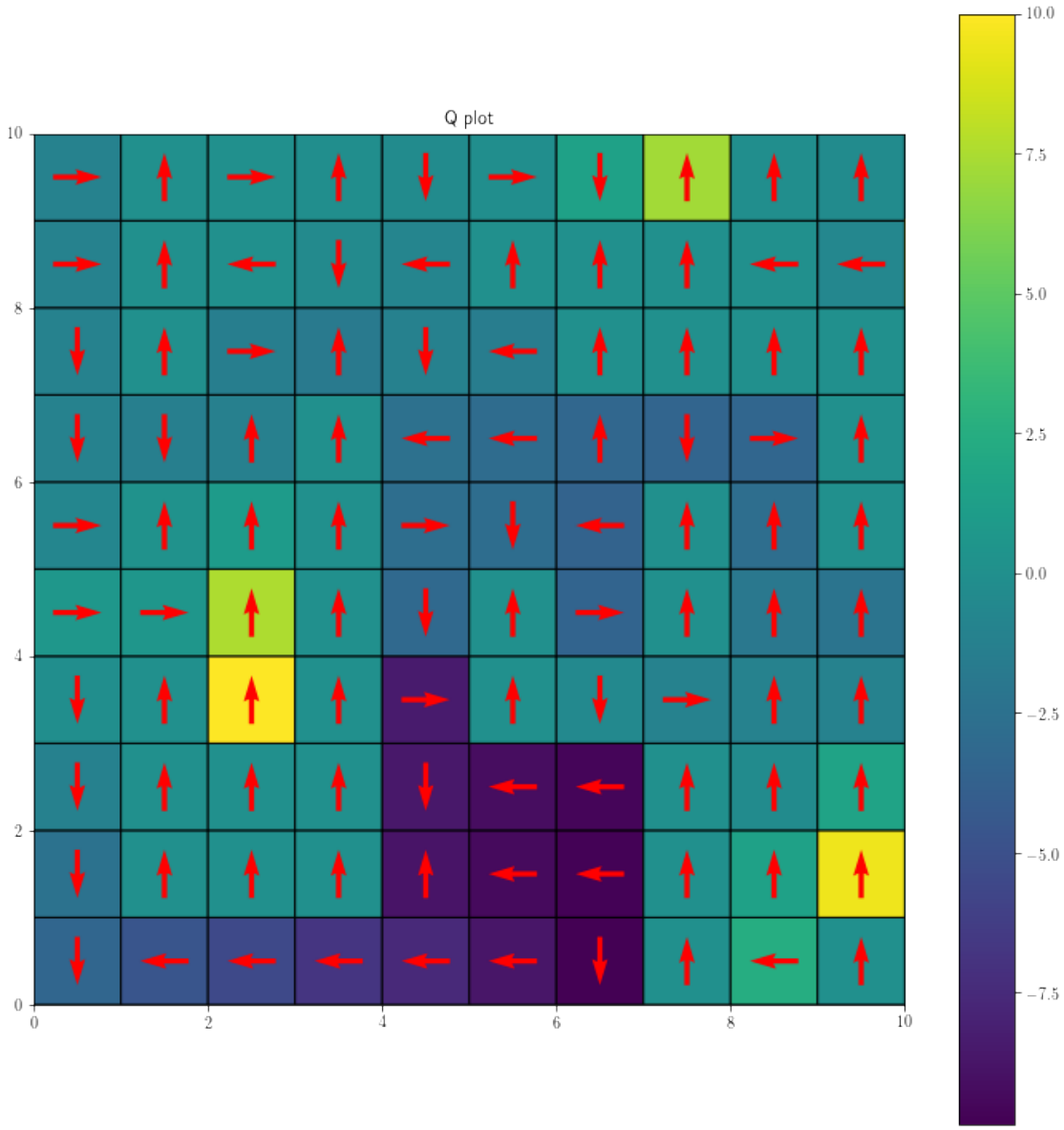
```
100%|| 2000/2000 [00:06<00:00, 327.56it/s]
```


Total Reward
0
−50
−100
−150
−200
0
250
500
750
1000
1250
1500
1750
2000
Episode


Steps Taken
100
80
60
40
20
0
250
500
750
1000
1250
1500
1750
2000
Episode


Steps: 18, Total Reward: -12
10
8
6
4
2
0
0
2
4
6
8
10


Steps: 13, Total Reward: -7
10
8
6
4
2
0
0
2
4
6
8
10



6.4 Config 1

Setting up a sweep

[ ]:
```
NUM_CONFIG = 1

config_settings = configurations_1[NUM_CONFIG]

# create environment
```

```
env = create_env(**config_settings._asdict())

sweep_config = {
    "name" : f"{NUM_CONFIG}-config-sweep",
    "method": "random",
    "parameters": {
        "algorithm": {
            "values": ['sarsa', 'q_learning'],
        },
        "policy": {
            "values": ['softmax', 'epsilon_greedy'],
        },
        "epsilon": {
            "min": 0.0,
            "max": 1.0,
        },
        "alpha": {
            "min": 0.01,
            "max": 0.2,
        },
        "gamma": {
            "min": 0.5,
            "max": 1.0,
        },
        "beta": {
            "min": 0.5,
            "max": 1.5,
        }
    }
}
```

6.4.1 wandb Sweep

[ ]:
```
sweep_id = wandb.sweep(sweep_config, project='RLPA1')
```

[ ]:
```
wandb.agent(sweep_id, run, count=20)
```

6.4.2 Plotting

best configuration

[ ]:
```
alpha = 0.1771
l_alpha = [0.20, 0.1, 0.15]
beta = 0.8304
l_beta = [1, 0.8, 0.9]
gamma = 0.9964
l_gamma = [0.95, 0.9, 1.0]
```

```
epsilon = 0.05
l_epsilon = [0.1, 0.15, 0.2]
# for epsilon we are using these 4 values only to explore
policy = 'e-greedy'
algorithm = 'sarsa'
```

Writing functions to automate plots for all the configurations

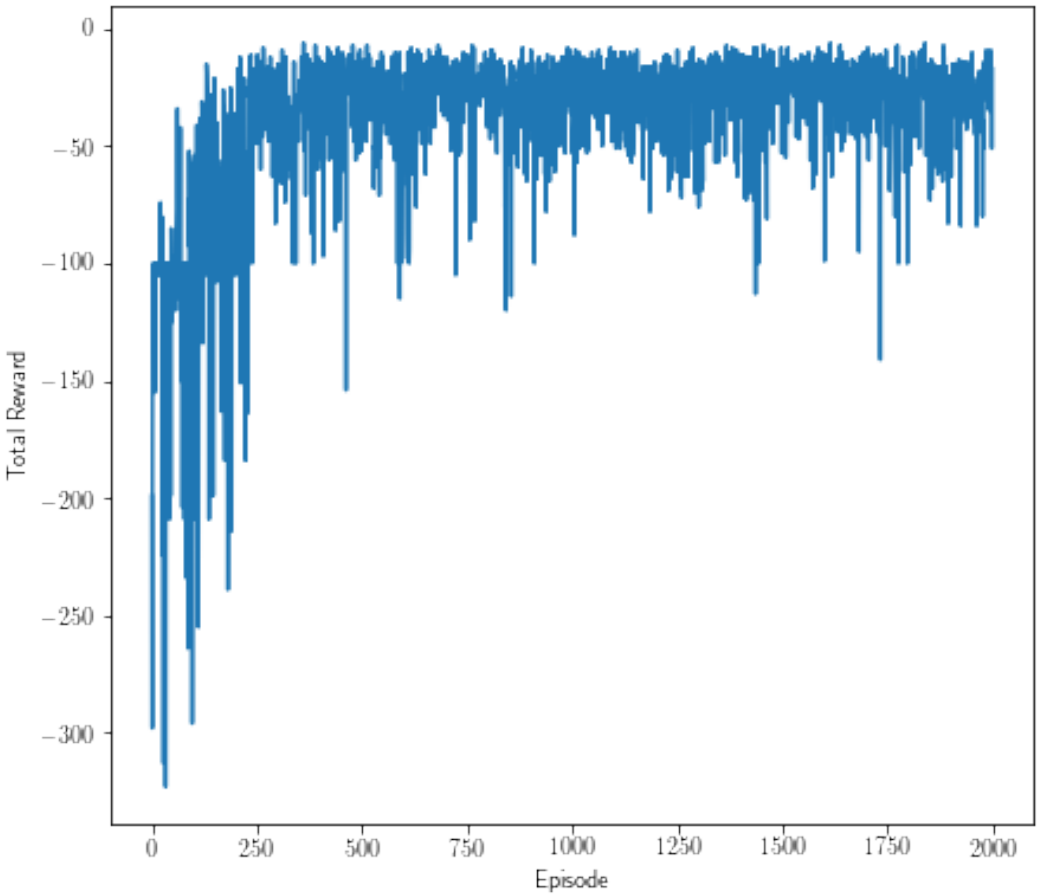
[ ]:
```
plot_all(alpha, l_alpha, beta, l_beta, gamma, l_gamma, epsilon, l_epsilon)
```

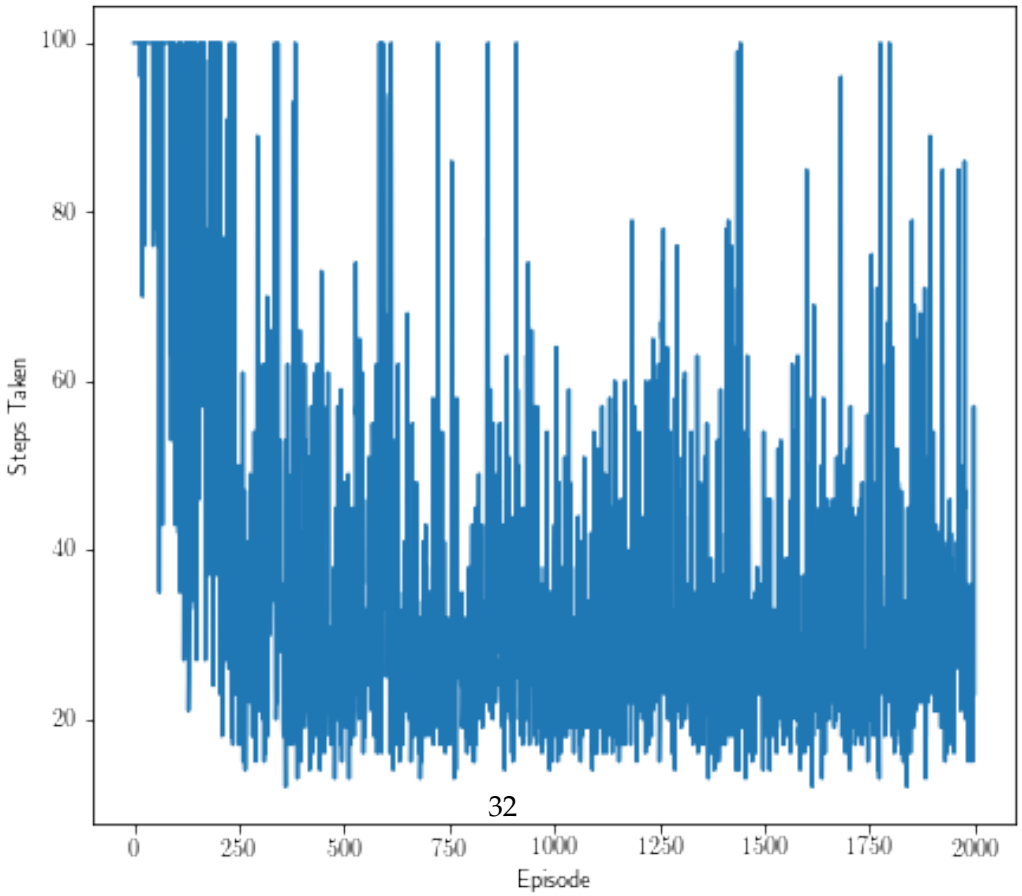
best pltos

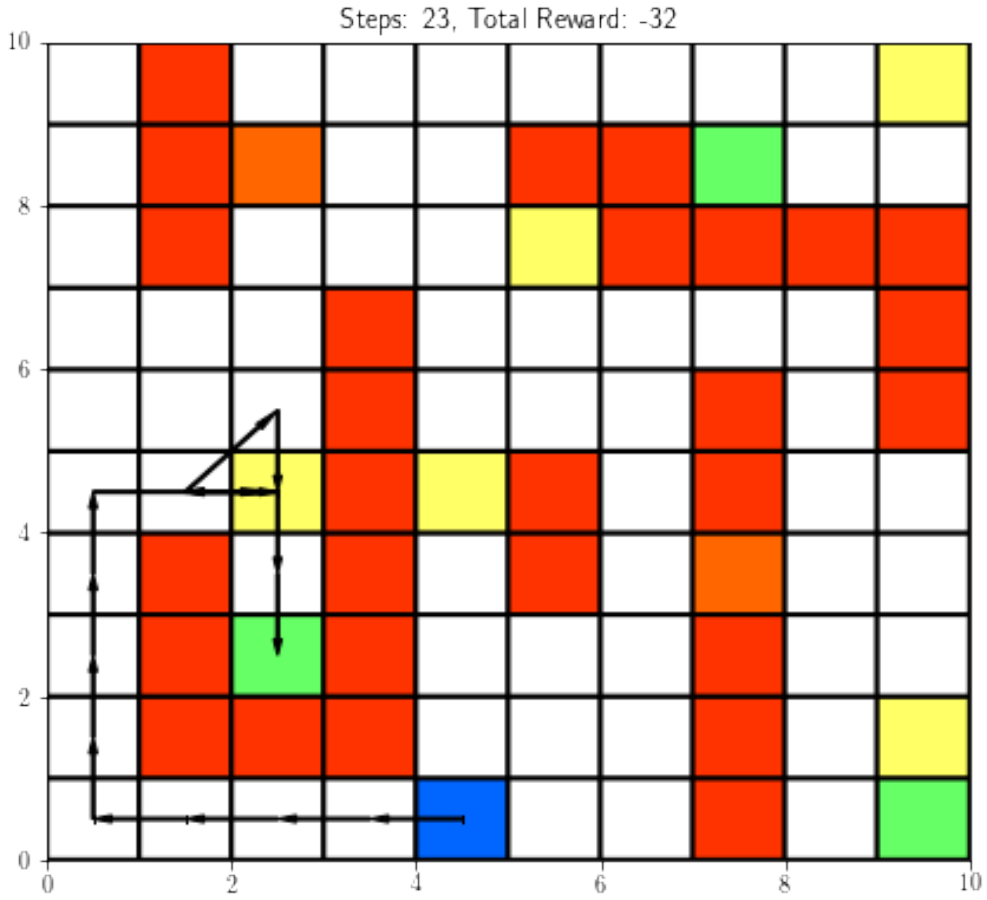
[ ]:
```
policy = 'softmax'
```

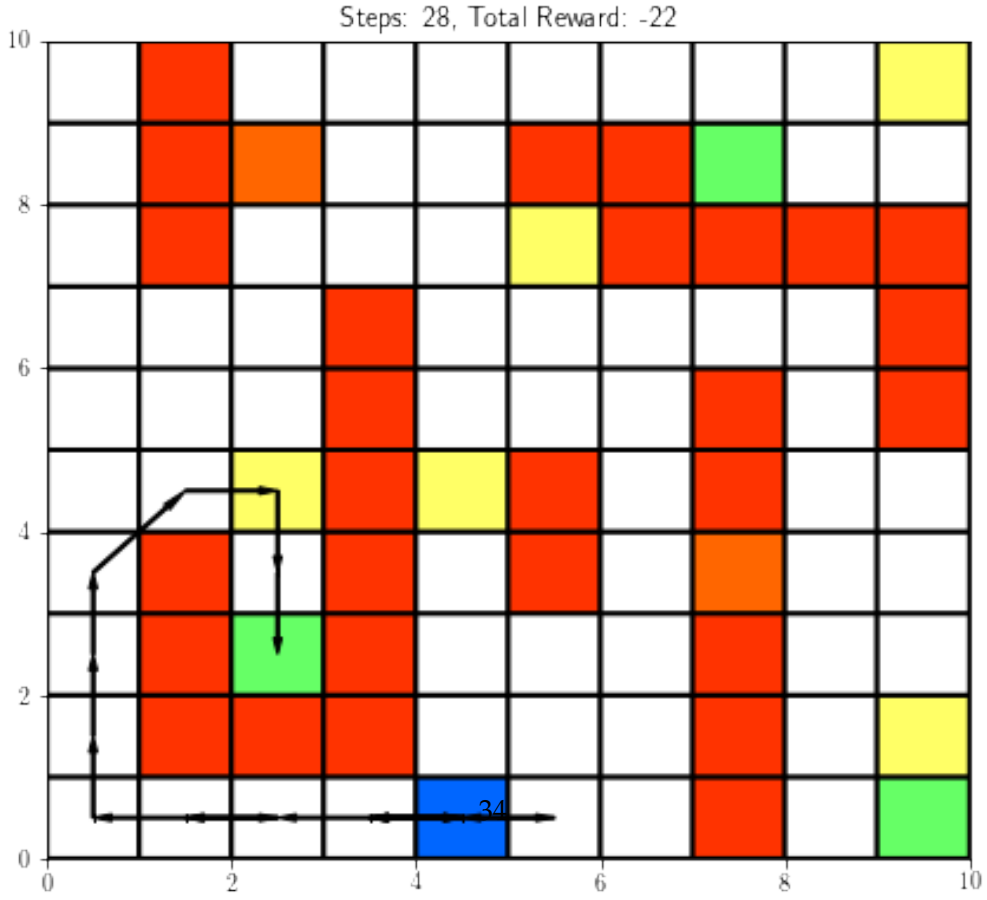
[ ]:
```
Q = best_plots(algorithm, policy, alpha,  epsilon, beta , gamma )
```

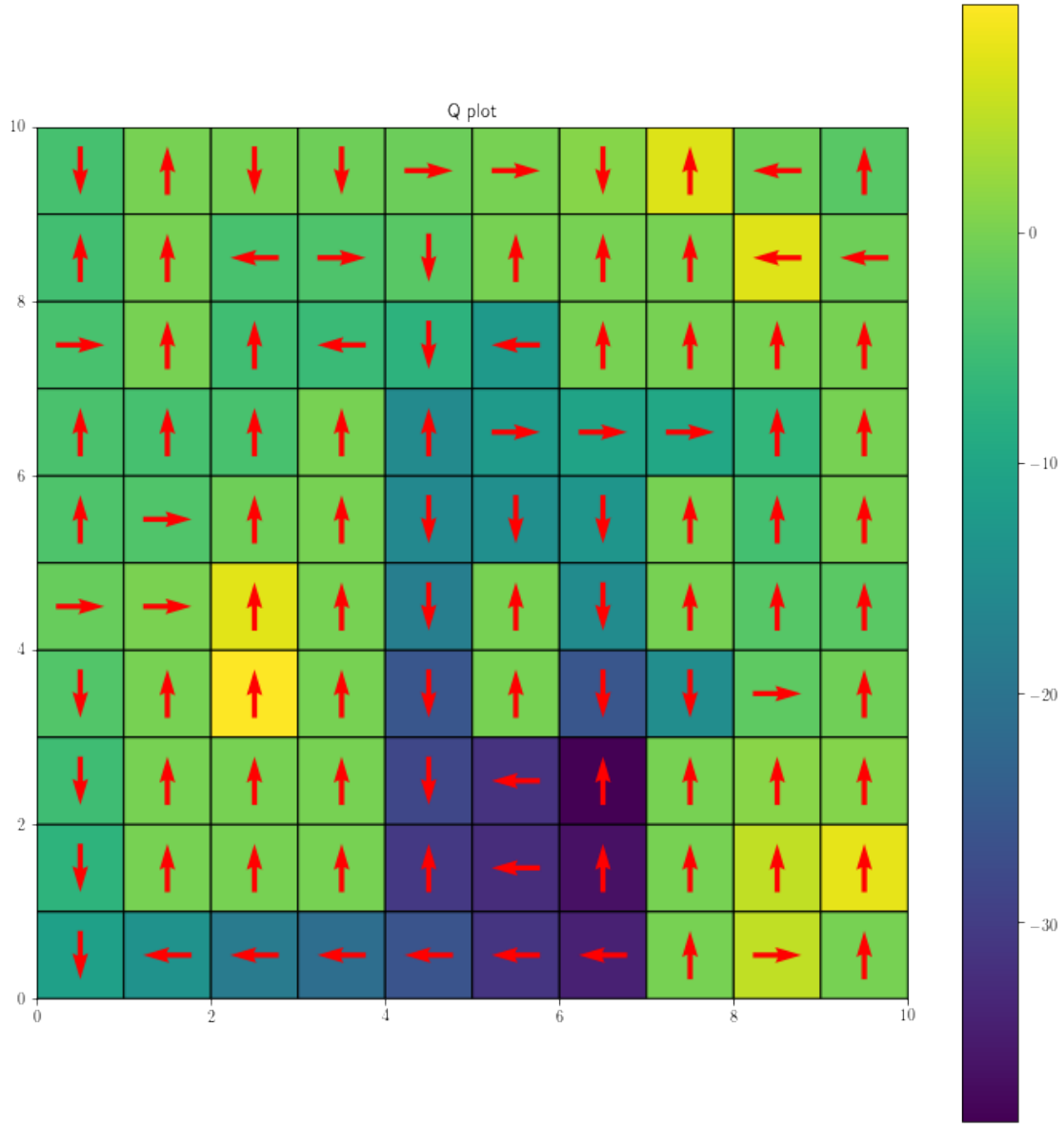
```
100%|| 2000/2000 [00:30<00:00, 65.97it/s]
```


Total Reward
0
−50
−100
−150
−200
−250
−300
0
250
500
750
1000
1250
1500
1750
2000
Episode


Steps Taken
100
80
60
40
20
0
250
500
750
1000
1250
1500
1750
2000
Episode


Steps: 23, Total Reward: -32
10
8
6
4
2
0
0
2
4
6
8
10


Steps: 28, Total Reward: -22
10
8
6
4
2
0
0
2
4
6
8
10



6.5 Config 2

[ ]:
```
NUM_CONFIG = 2

config_settings = configurations_l[NUM_CONFIG]

# create environment
env = create_env(**config_settings._asdict())
```

```
sweep_config = {
    "name" : f"{NUM_CONFIG}-config-sweep",
    "method": "random",
    "parameters": {
        "algorithm": {
            "values": ['sarsa', 'q_learning'],
        },
        "policy": {
            "values": ['softmax', 'epsilon_greedy'],
        },
        "epsilon": {
            "min": 0.0,
            "max": 1.0,
        },
        "alpha": {
            "min": 0.01,
            "max": 0.2,
        },
        "gamma": {
            "min": 0.5,
            "max": 1.0,
        },
        "beta": {
            "min": 0.5,
            "max": 1.5,
        }
    }
}
```

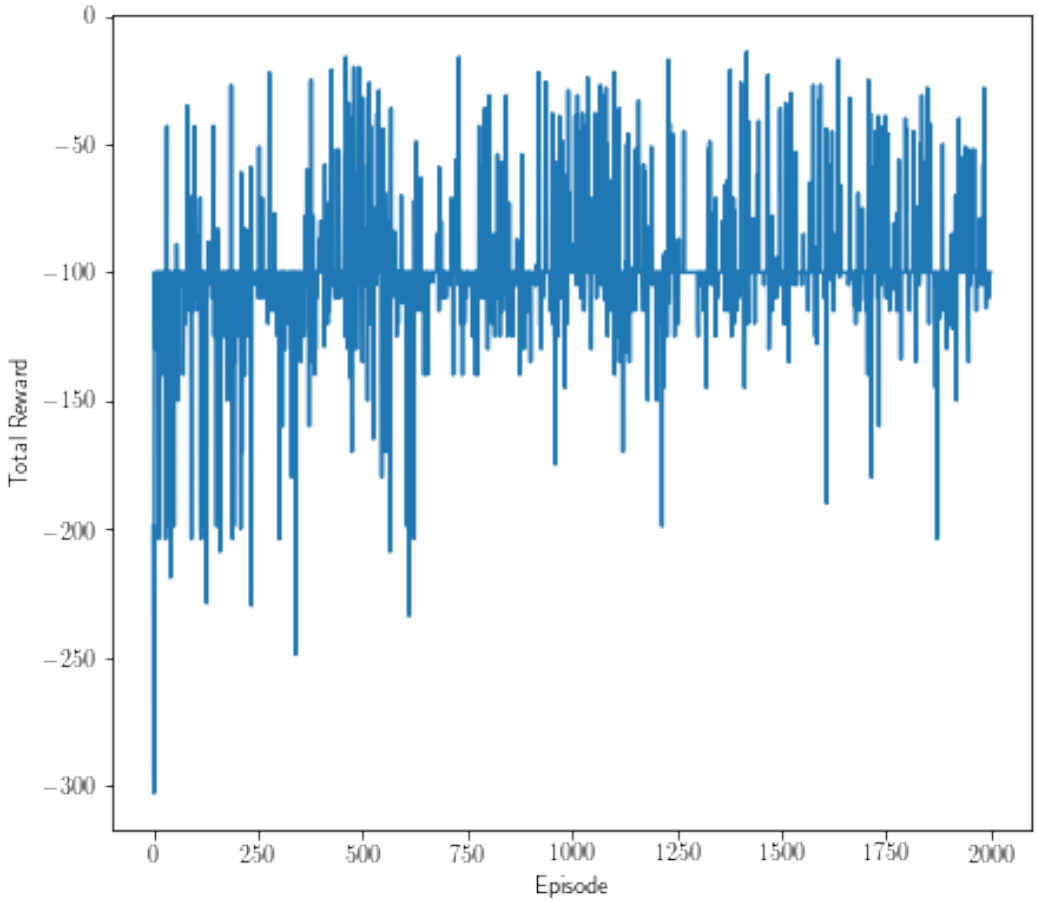
6.5.1 Plotting

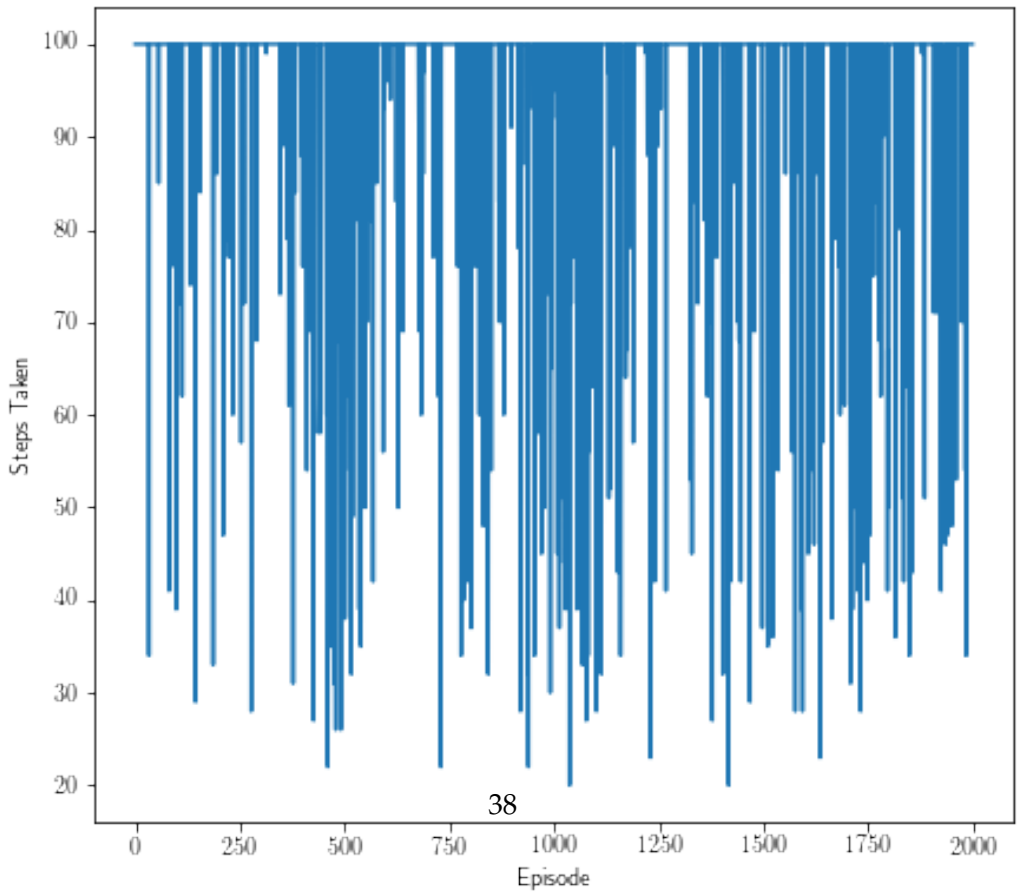
[ ]:
```
alpha = 0.1643
l_alpha = [0.20, 0.1, 0.15]
beta = 0.6187
l_beta = [0.5, 0.8, 0.7]
gamma = 0.967
l_gamma = [0.95, 0.9, 1.0]
epsilon = 0.05
l_epsilon = [0.1, 0.15, 0.2]
# for epsilon we are using these 4 values only to explore
policy = 'softmax'
algorithm = 'sarsa'
```

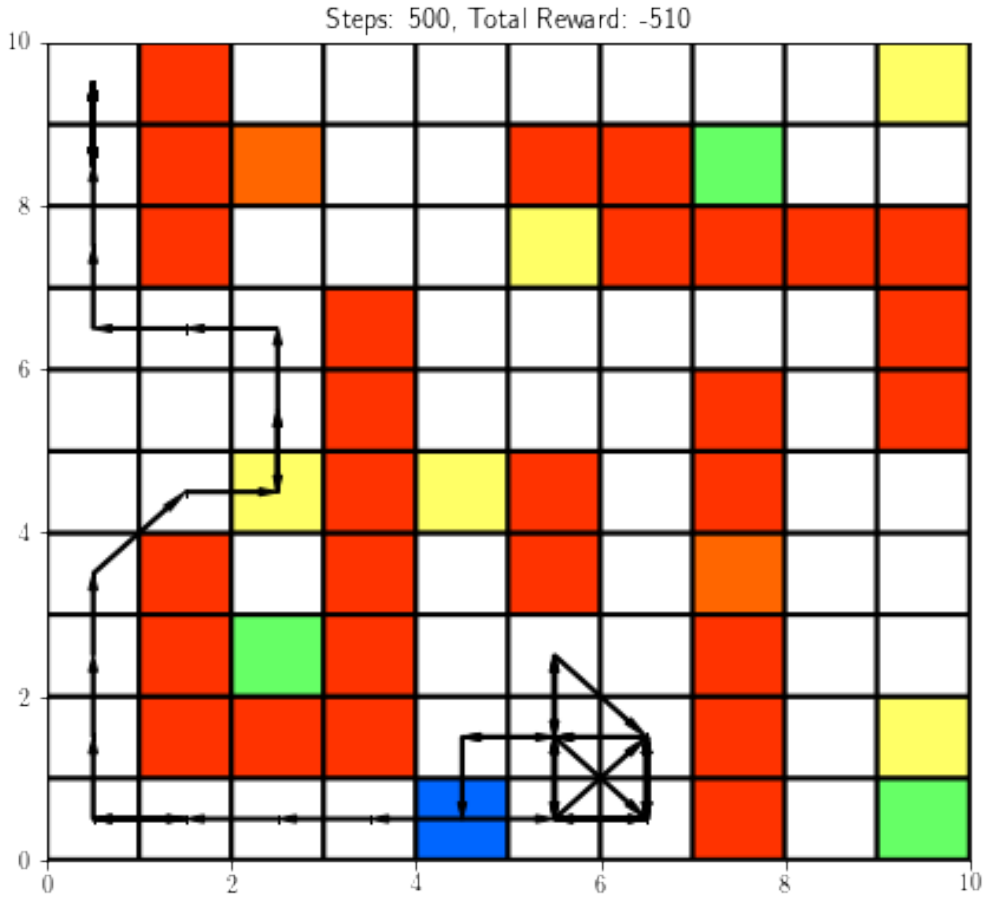
[ ]:
```
plot_all(alpha, l_alpha, beta, l_beta, gamma, l_gamma, epsilon, l_epsilon)
```

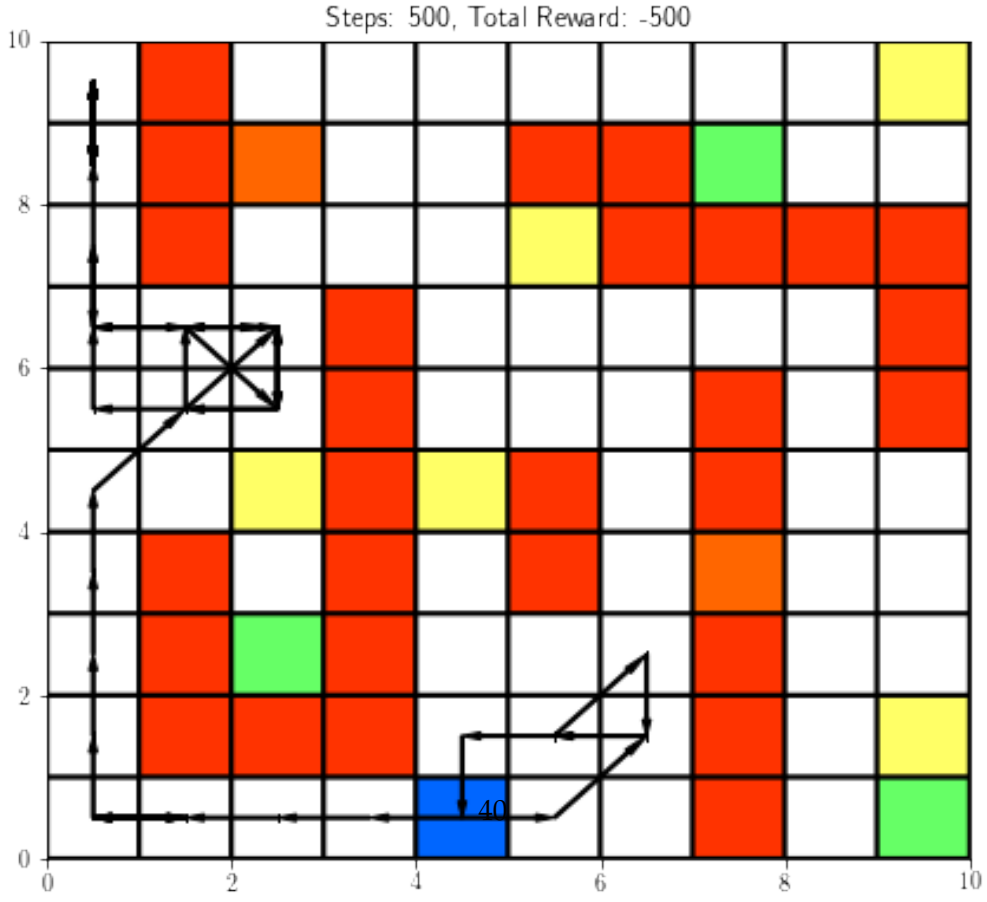
[ ]:
```
Q = best_plots(algorithm, policy, alpha,  epsilon, beta , gamma )
```

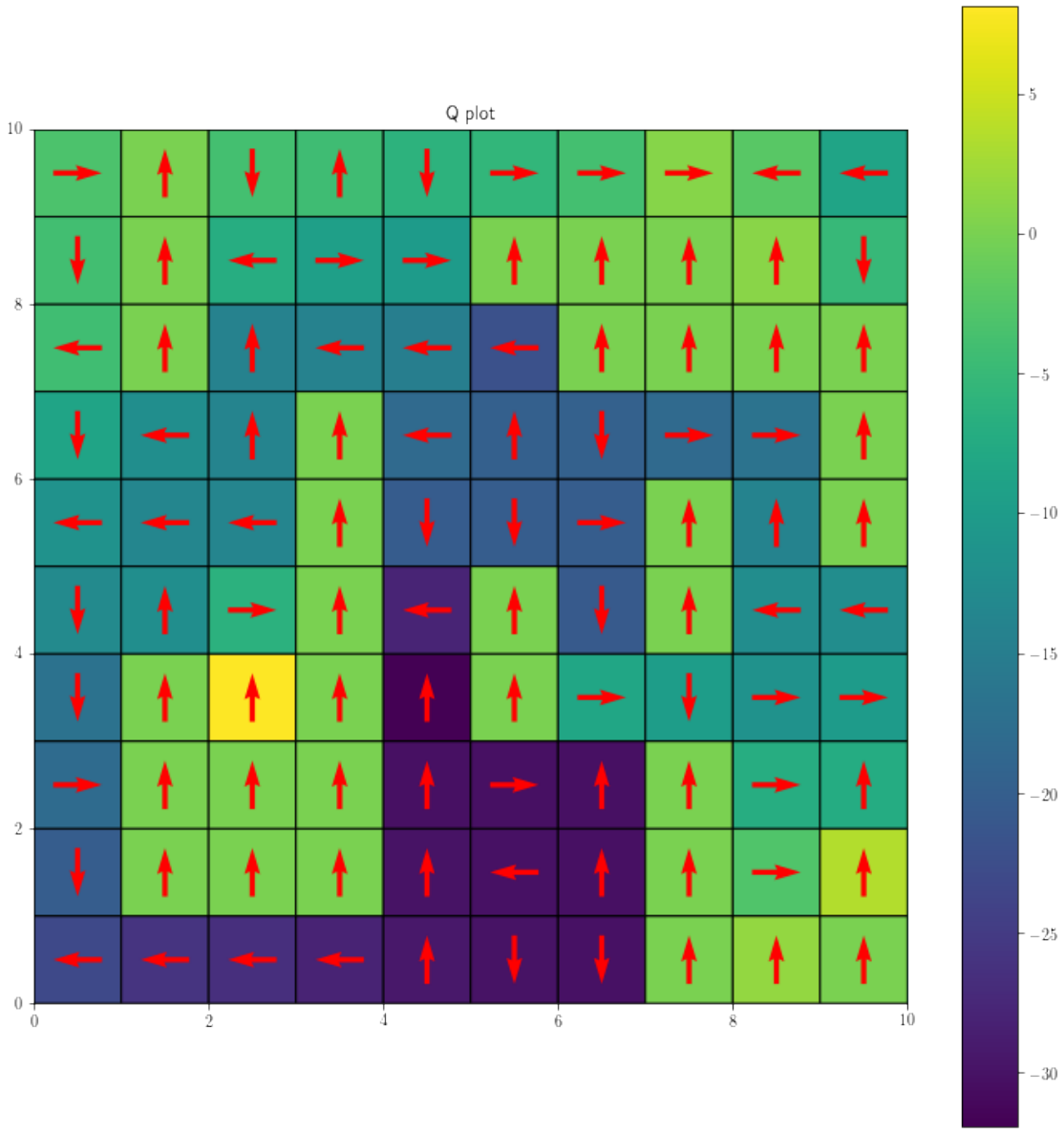
```
100%|| 2000/2000 [01:05<00:00, 30.49it/s]
```


0
−50
−100
−150
−200
−250
−300
Total Reward
0
250
500
750
1000
1250
1500
1750
2000
Episode


100
90
80
70
60
50
40
30
20
Steps Taken
0
250
500
750
1000
1250
1500
1750
2000
Episode


Steps: 500, Total Reward: -510
0
2
4
6
8
10


Steps: 500, Total Reward: -500
0
2
4
6
8
10



6.5.2 wandb sweep

[ ]:
```
sweep_id = wandb.sweep(sweep_config, project='RLPA1')
```

[ ]:
```
wandb.agent(sweep_id, run, count=20)
```

[ ]:
```

```

6.6 Confgi 3

[ ]:
```
NUM_CONFIG = 3

config_settings = configurations_1[NUM_CONFIG]

# create environment
env = create_env(**config_settings._asdict())

sweep_config = {
    "name" : f"{NUM_CONFIG}-config-sweep",
    "method": "random",
    "parameters": {
        "algorithm": {
            "values": ['sarsa', 'q_learning'],
        },
        "policy": {
            "values": ['softmax', 'epsilon_greedy'],
        },
        "epsilon": {
            "min": 0.0,
            "max": 1.0,
        },
        "alpha": {
            "min": 0.01,
            "max": 0.2,
        },
        "gamma": {
            "min": 0.5,
            "max": 1.0,
        },
        "beta": {
            "min": 0.5,
            "max": 1.5,
        }
    }
}
```

6.6.1 wandb sweep

[ ]:
```
sweep_id = wandb.sweep(sweep_config, project='RLPA1')
```

[ ]:
```
wandb.agent(sweep_id, run, count=20)
```

6.6.2 Plotting

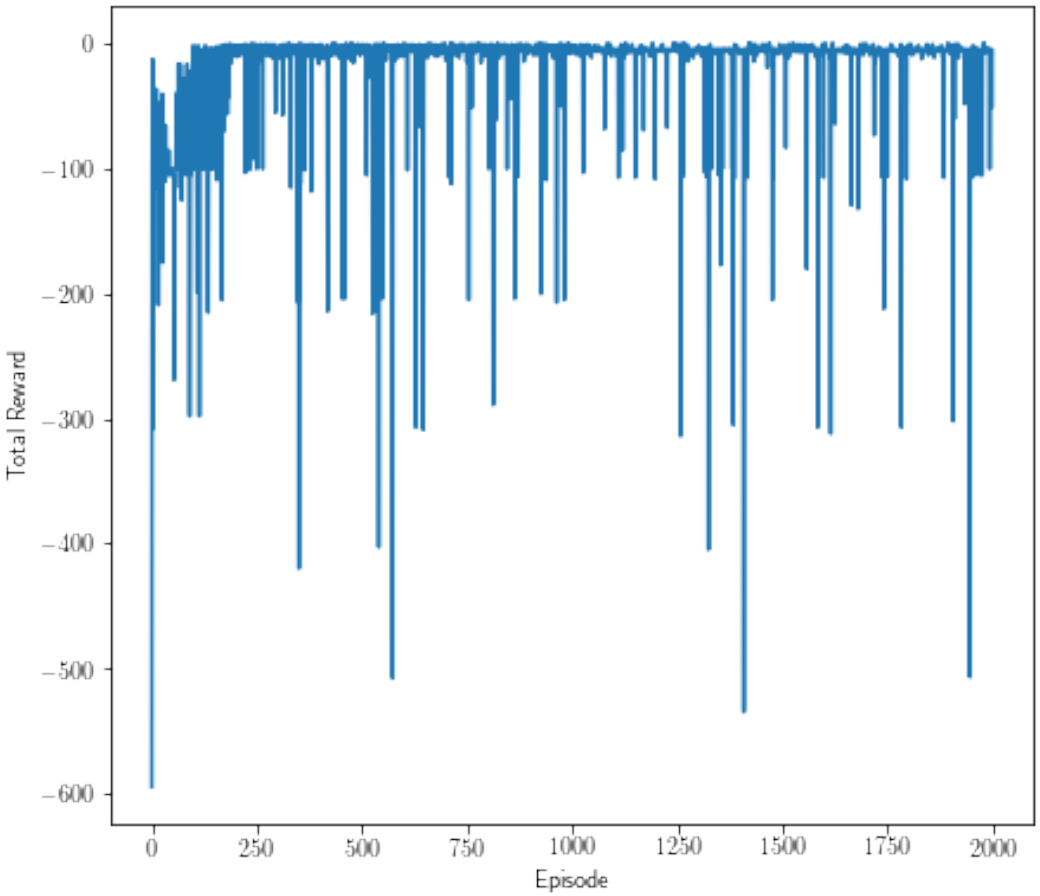
[ ]:
```
alpha = 0.1181
l_alpha = [0.10, 0.12, 0.15]
beta = 1.442
l_beta = [1.5, 1.3, 1.2]
gamma = 0.5043
l_gamma = [0.6, 0.4, 0.55]
epsilon = 0.06649
l_epsilon = [0.1, 0.05, 0.15]
policy = 'e_greedy'
algorithm = 'q-learning'
```

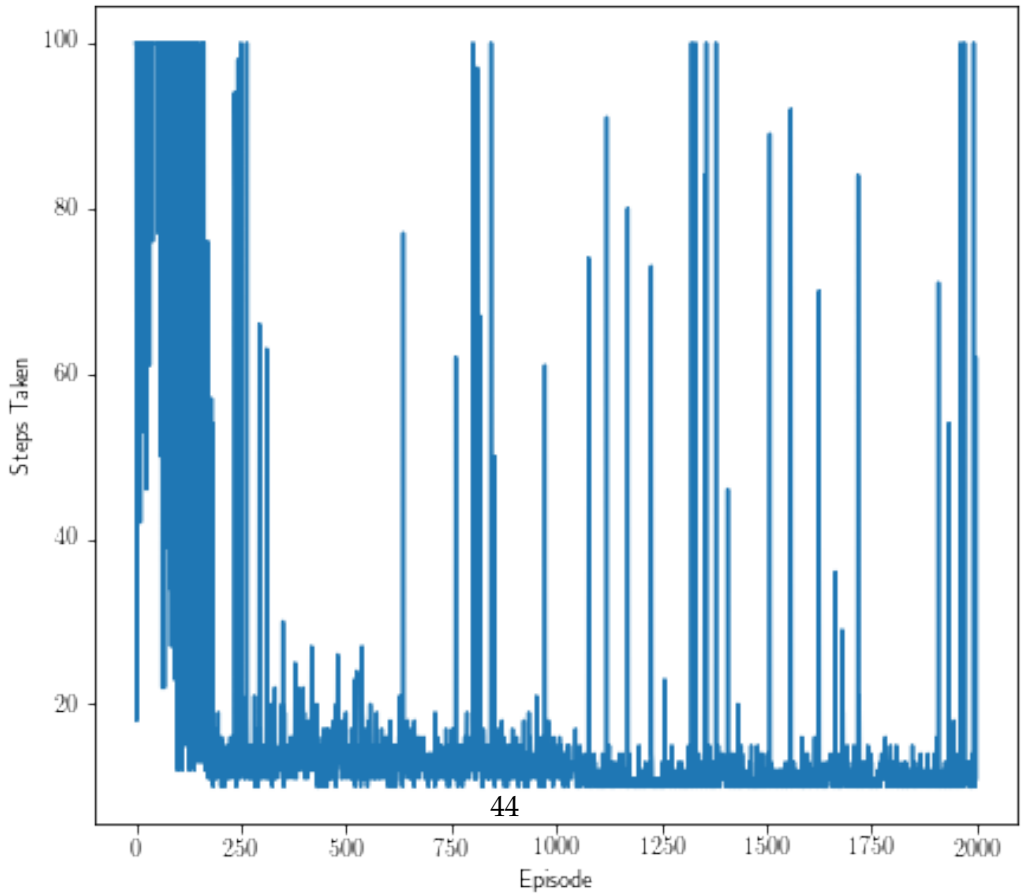
[ ]:
```
plt.style.use('seaborn')
```

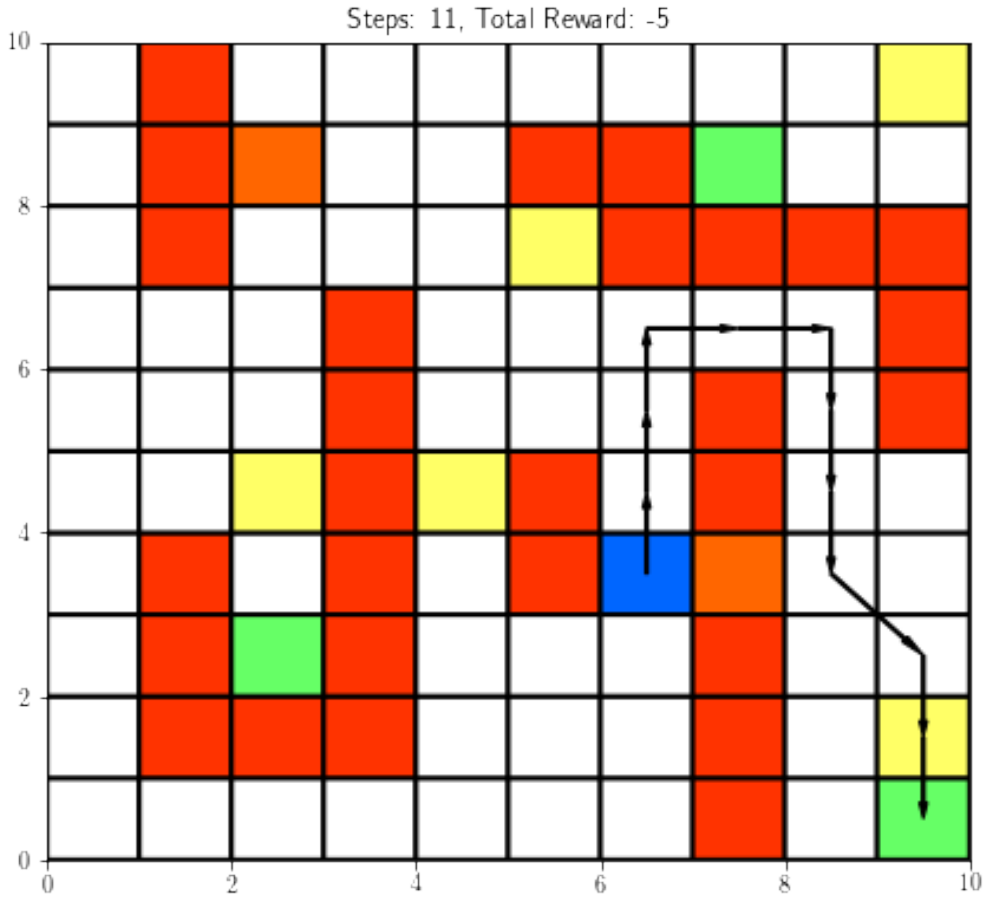
[ ]:
```
plot_all(alpha, l_alpha, beta, l_beta, gamma, l_gamma, epsilon, l_epsilon)
```

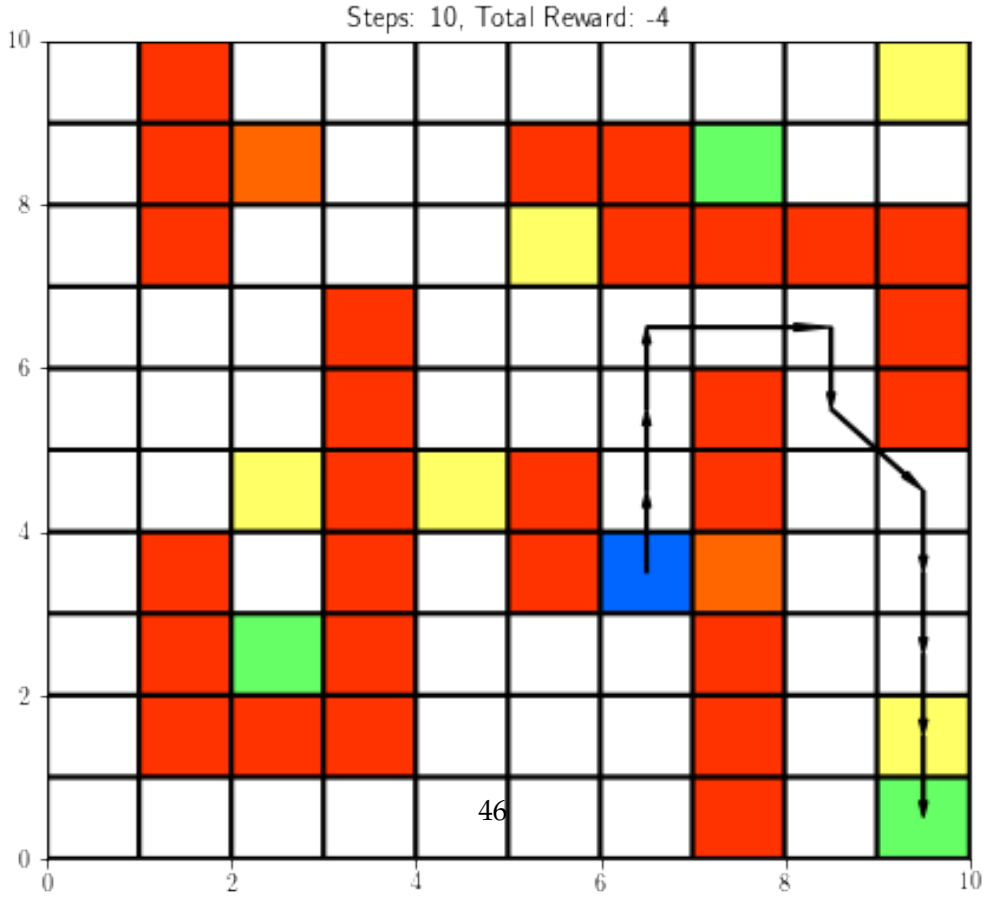
[ ]:
```
Q = best_plots(algorithm, policy, alpha,  epsilon, beta , gamma )
```

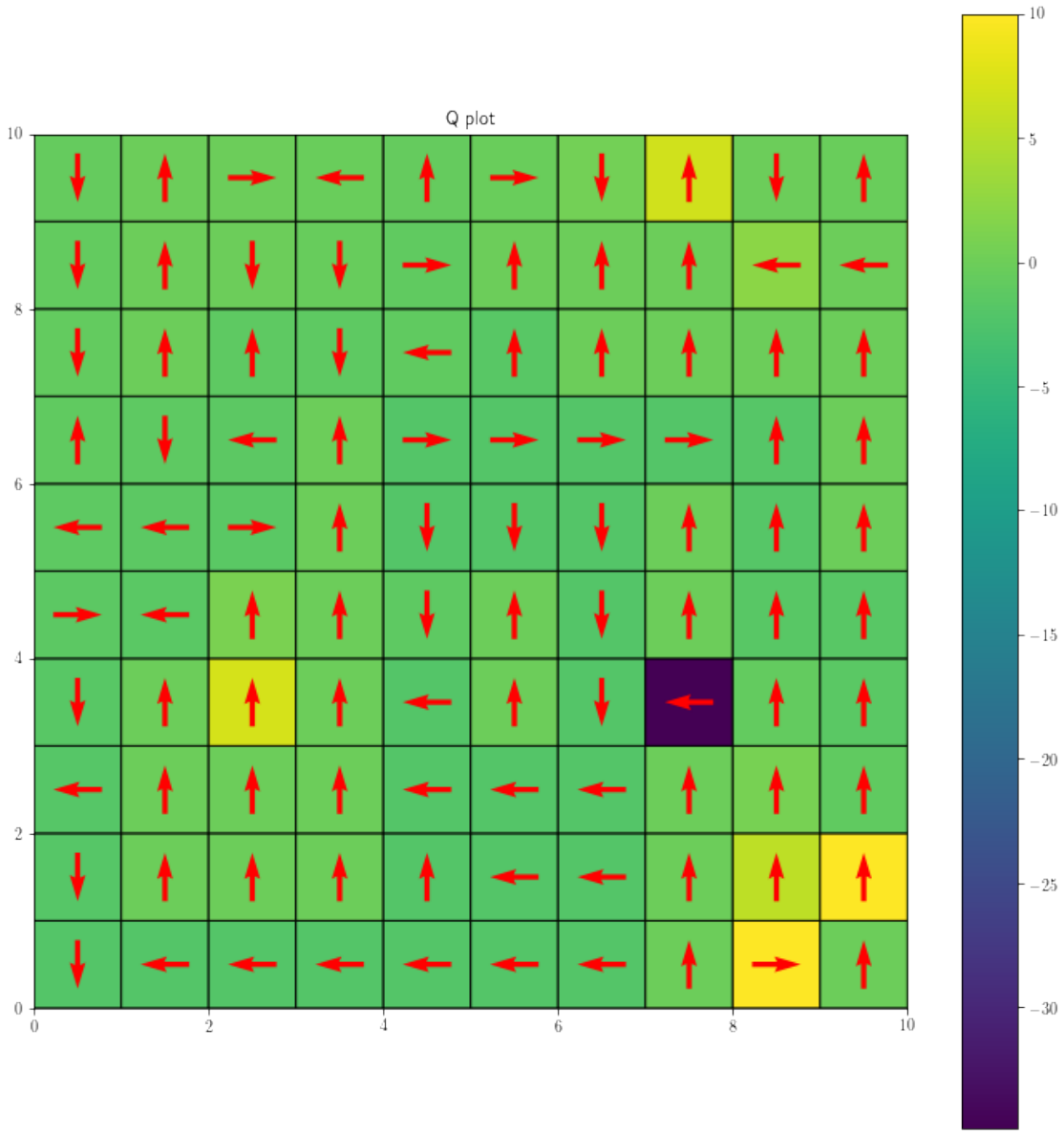
```
100%|| 2000/2000 [00:08<00:00, 233.65it/s]
```


0
−100
−200
−300
−400
−500
−600
Total Reward
0
250
500
750
1000
1250
1500
1750
2000
Episode


100
80
60
40
20
Steps Taken
0
250
500
750
1000
1250
1500
1750
2000
Episode


Steps: 11, Total Reward: -5
10
8
6
4
2
0
0
2
4
6
8
10


Steps: 10, Total Reward: -4
10
8
6
4
2
0
0
2
4
6
8
10



6.7 config 9

[22]:
```
NUM_CONFIG = 9

config_settings = configurations_l[NUM_CONFIG]

# create environment
env = create_env(**config_settings._asdict())
```

```
sweep_config = {
    "name" : f"{NUM_CONFIG}-config-sweep",
    "method": "random",
    "parameters": {
        "algorithm": {
            "values": ['sarsa', 'q_learning'],
        },
        "policy": {
            "values": ['softmax', 'epsilon_greedy'],
        },
        "epsilon": {
            "min": 0.0,
            "max": 1.0,
        },
        "alpha": {
            "min": 0.01,
            "max": 0.2,
        },
        "gamma": {
            "min": 0.5,
            "max": 1.0,
        },
        "beta": {
            "min": 0.5,
            "max": 1.5,
        }
    }
}
```

[ ]:
```
sweep_id = wandb.sweep(sweep_config, project='RLPA1')
```

[ ]:
```
wandb.agent(sweep_id, run, count=20)
```

6.7.1 Plotting

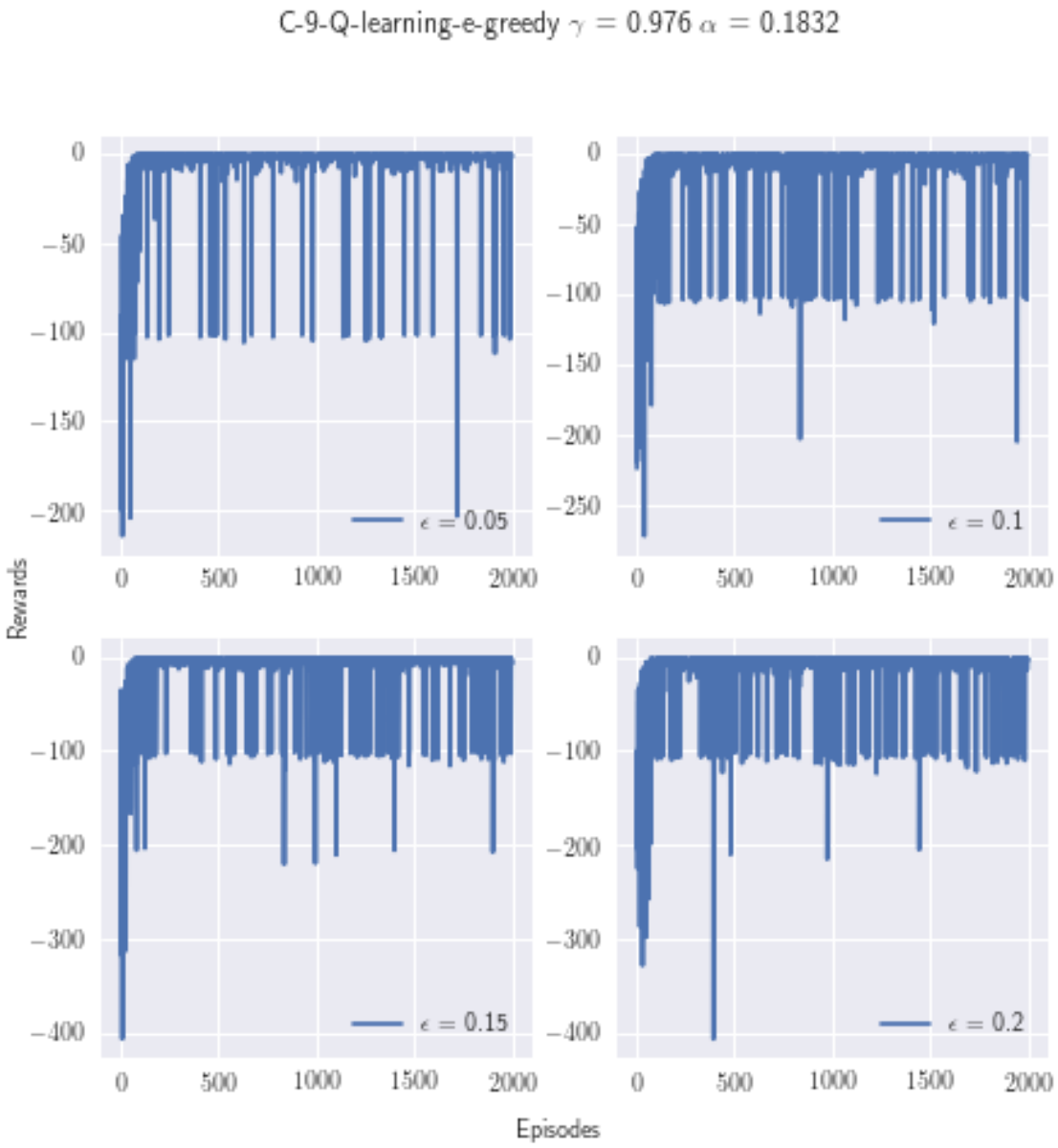
[23]:
```
alpha = 0.1832
l_alpha = [0.15, 0.12, 0.20]
beta = 0.6001
l_beta = [0.65, 0.7, 0.5]
gamma = 0.976
l_gamma = [0.99, 0.95, 0.90]
epsilon = 0.05
l_epsilon = [0.1, 0.15, 0.2]
policy = 'softmax'
algorithm = 'sarsa'
```

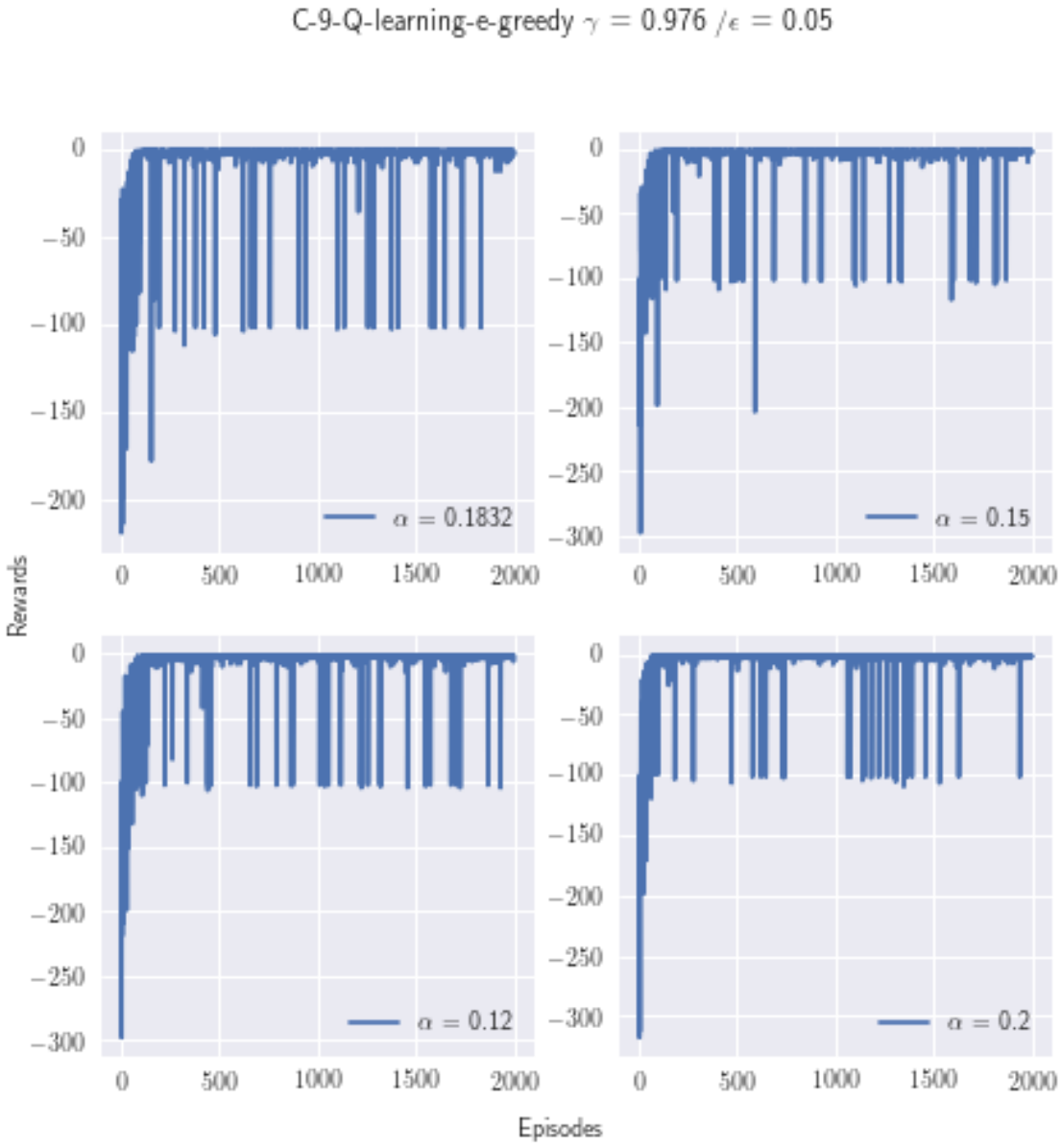
[24]:
```
plt.style.use('seaborn')
```

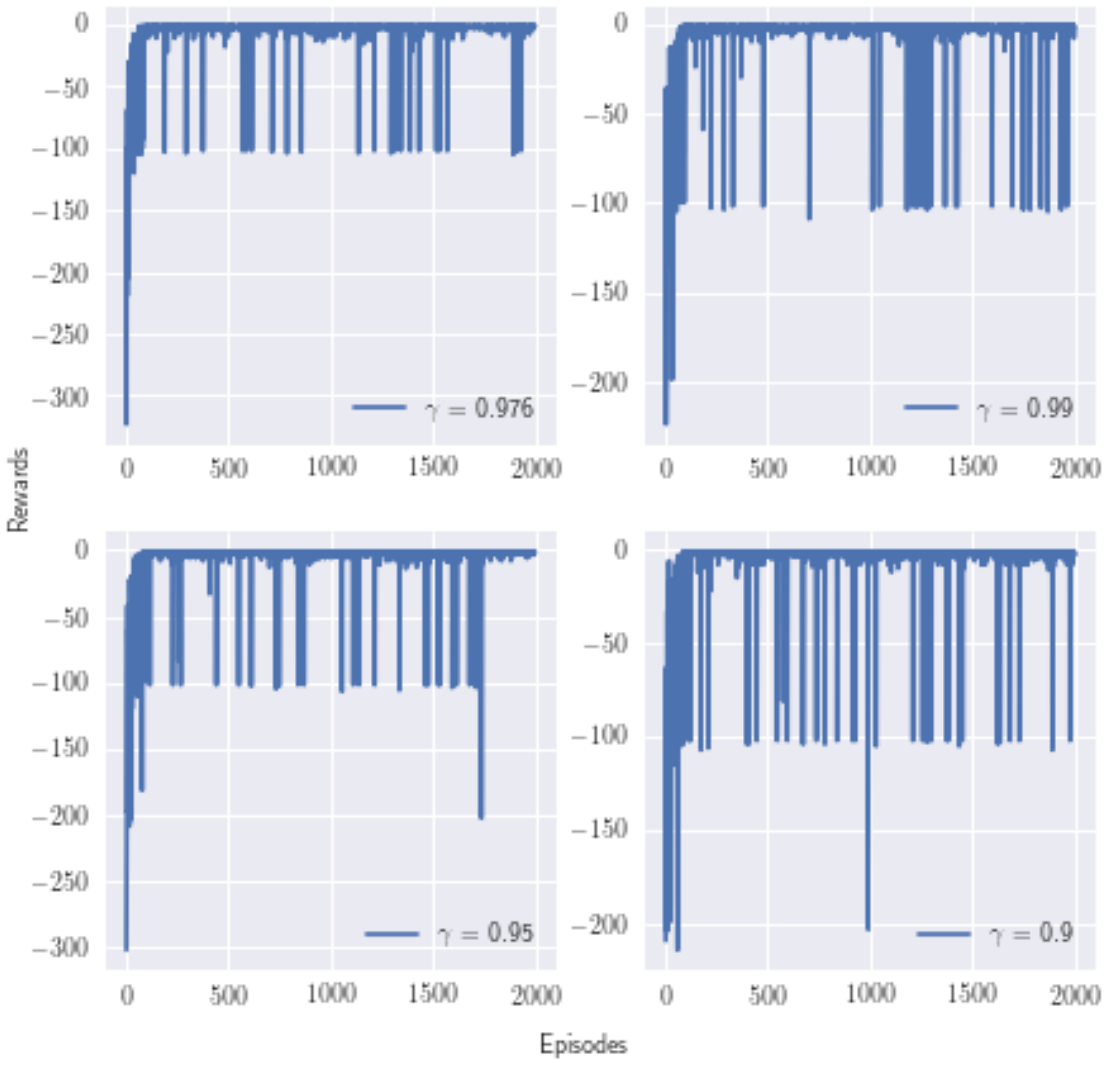
[25]:
```
plot_all(alpha, l_alpha, beta, l_beta, gamma, l_gamma, epsilon, l_epsilon)
```

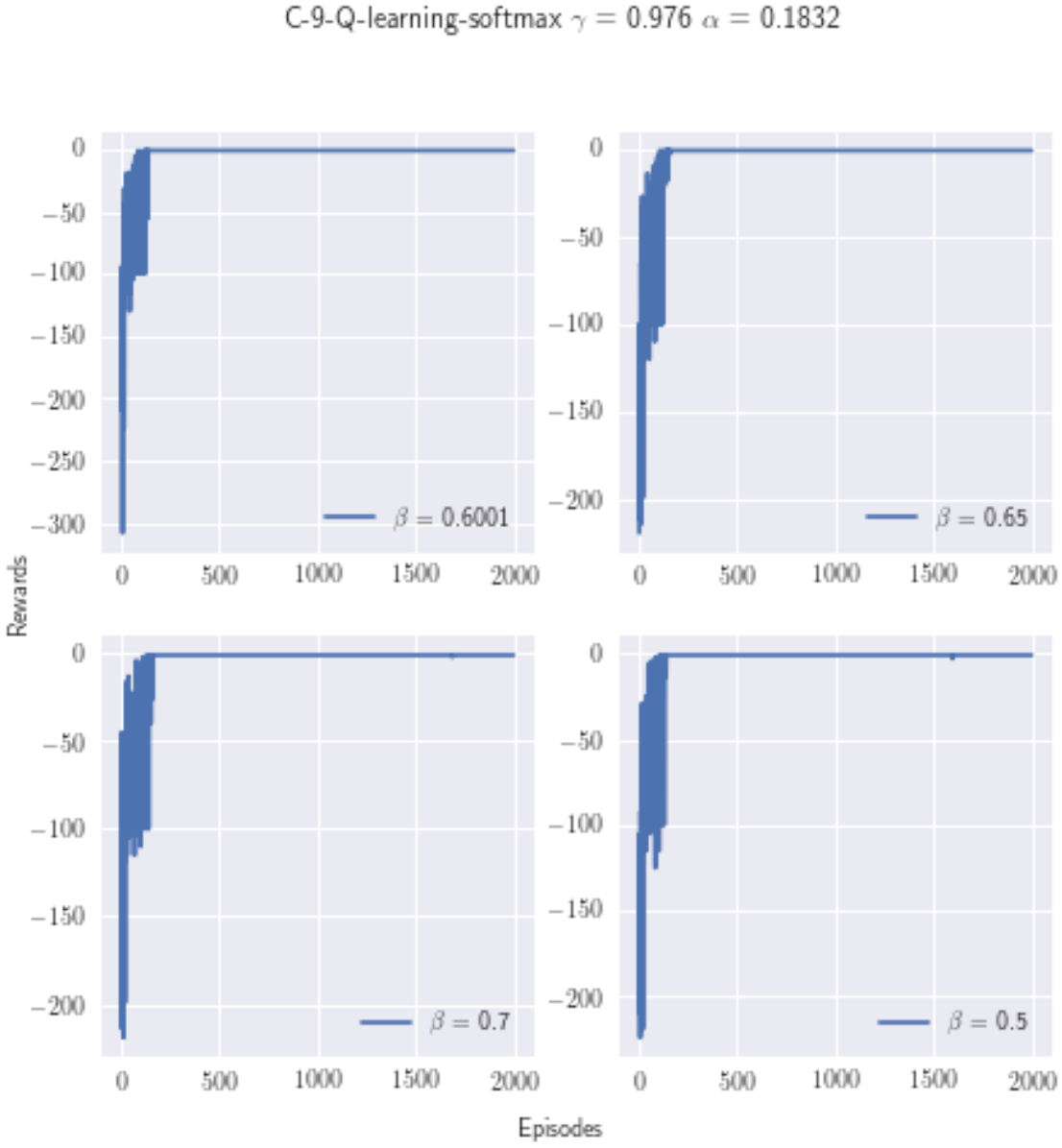
```
100%|| 2000/2000 [00:12<00:00, 165.98it/s]
100%|| 2000/2000 [00:15<00:00, 131.73it/s]
100%|| 2000/2000 [00:11<00:00, 176.01it/s]
100%|| 2000/2000 [00:12<00:00, 160.99it/s]
100%|| 2000/2000 [00:10<00:00, 191.66it/s]
100%|| 2000/2000 [00:10<00:00, 193.22it/s]
100%|| 2000/2000 [00:11<00:00, 176.46it/s]
100%|| 2000/2000 [00:10<00:00, 198.83it/s]
100%|| 2000/2000 [00:10<00:00, 192.41it/s]
100%|| 2000/2000 [00:10<00:00, 190.33it/s]
100%|| 2000/2000 [00:10<00:00, 185.61it/s]
100%|| 2000/2000 [00:10<00:00, 191.35it/s]
100%|| 2000/2000 [00:16<00:00, 120.17it/s]
100%|| 2000/2000 [00:16<00:00, 118.03it/s]
100%|| 2000/2000 [00:16<00:00, 119.17it/s]
100%|| 2000/2000 [00:16<00:00, 123.09it/s]
100%|| 2000/2000 [00:16<00:00, 122.42it/s]
100%|| 2000/2000 [00:17<00:00, 116.64it/s]
100%|| 2000/2000 [00:17<00:00, 115.07it/s]
100%|| 2000/2000 [00:16<00:00, 124.24it/s]
100%|| 2000/2000 [00:13<00:00, 150.20it/s]
100%|| 2000/2000 [00:16<00:00, 117.83it/s]
100%|| 2000/2000 [00:17<00:00, 112.12it/s]
100%|| 2000/2000 [00:18<00:00, 109.81it/s]
100%|| 2000/2000 [00:10<00:00, 191.32it/s]
100%|| 2000/2000 [00:11<00:00, 174.08it/s]
100%|| 2000/2000 [00:11<00:00, 169.78it/s]
100%|| 2000/2000 [00:12<00:00, 156.27it/s]
100%|| 2000/2000 [00:10<00:00, 197.72it/s]
100%|| 2000/2000 [00:10<00:00, 189.98it/s]
100%|| 2000/2000 [00:10<00:00, 187.23it/s]
100%|| 2000/2000 [00:10<00:00, 198.16it/s]
100%|| 2000/2000 [00:10<00:00, 195.67it/s]
100%|| 2000/2000 [00:10<00:00, 194.74it/s]
100%|| 2000/2000 [00:10<00:00, 195.22it/s]
100%|| 2000/2000 [00:10<00:00, 195.36it/s]
100%|| 2000/2000 [00:16<00:00, 123.00it/s]
100%|| 2000/2000 [00:16<00:00, 122.75it/s]
100%|| 2000/2000 [00:16<00:00, 123.33it/s]
100%|| 2000/2000 [00:15<00:00, 127.40it/s]
100%|| 2000/2000 [00:16<00:00, 123.80it/s]
100%|| 2000/2000 [00:16<00:00, 119.41it/s]
100%|| 2000/2000 [00:17<00:00, 115.51it/s]
100%|| 2000/2000 [00:16<00:00, 124.28it/s]
100%|| 2000/2000 [00:16<00:00, 124.30it/s]
```

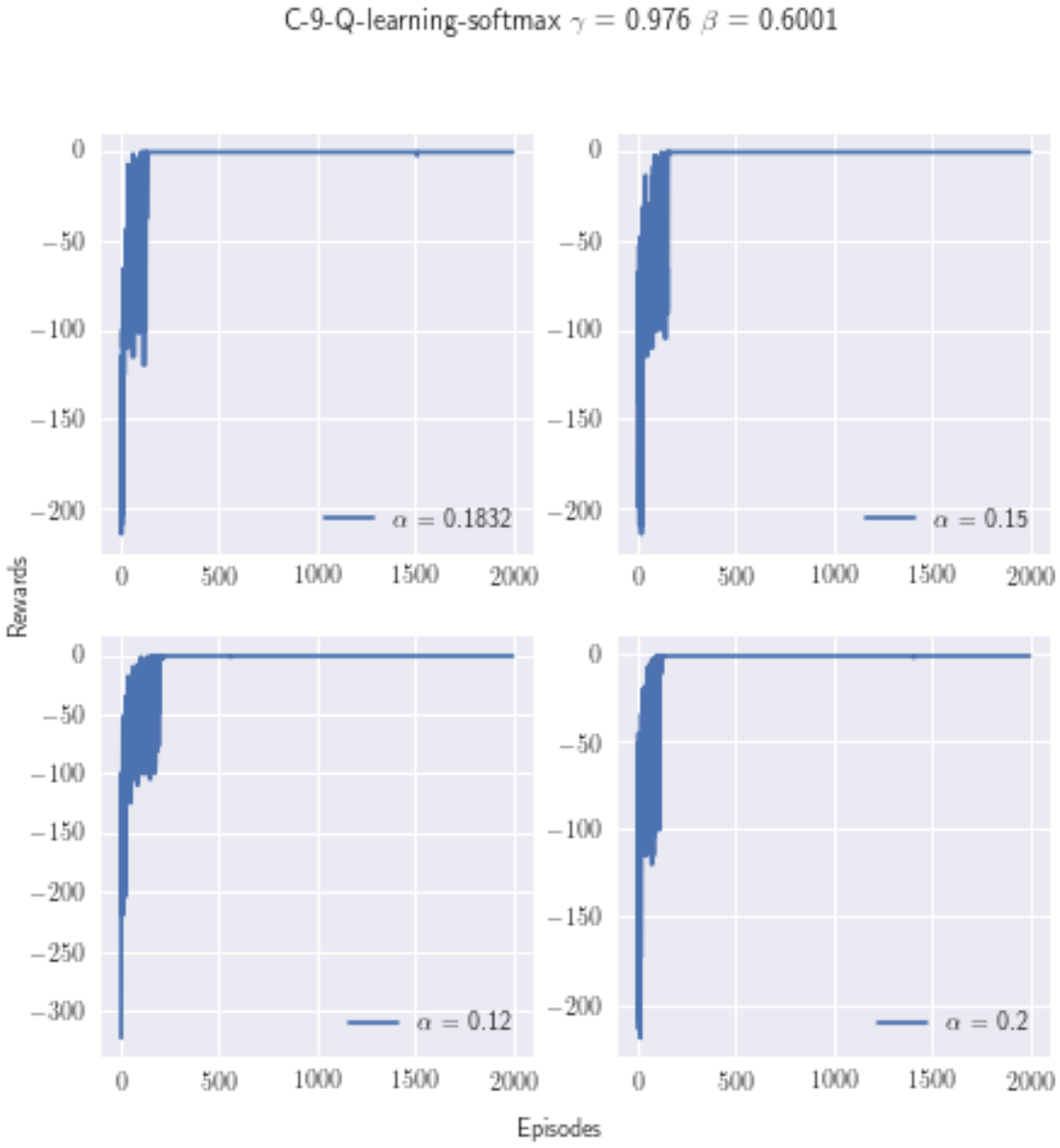
```
100%|| 2000/2000 [00:15<00:00, 127.89it/s]
100%|| 2000/2000 [00:13<00:00, 147.38it/s]
100%|| 2000/2000 [00:17<00:00, 115.12it/s]
```

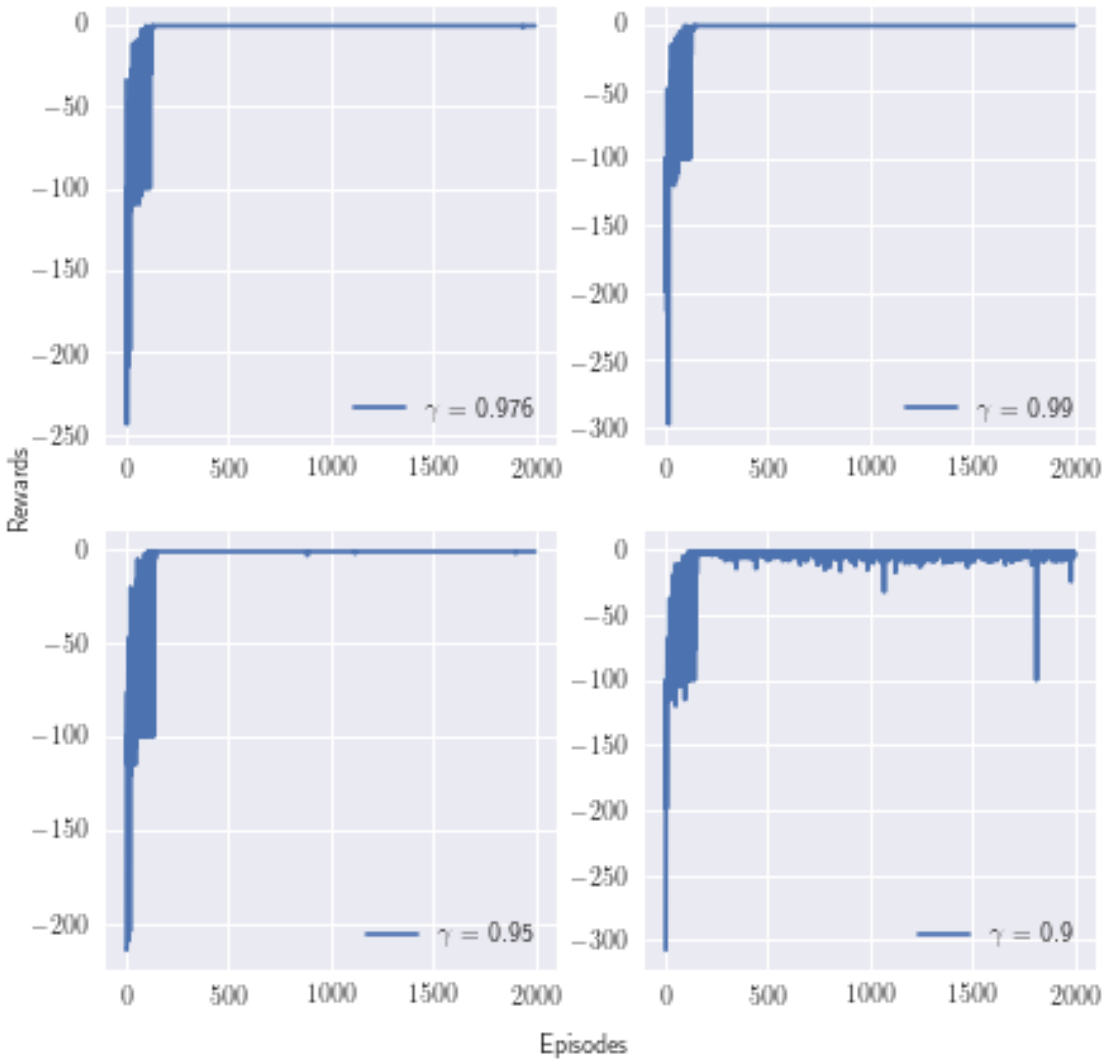


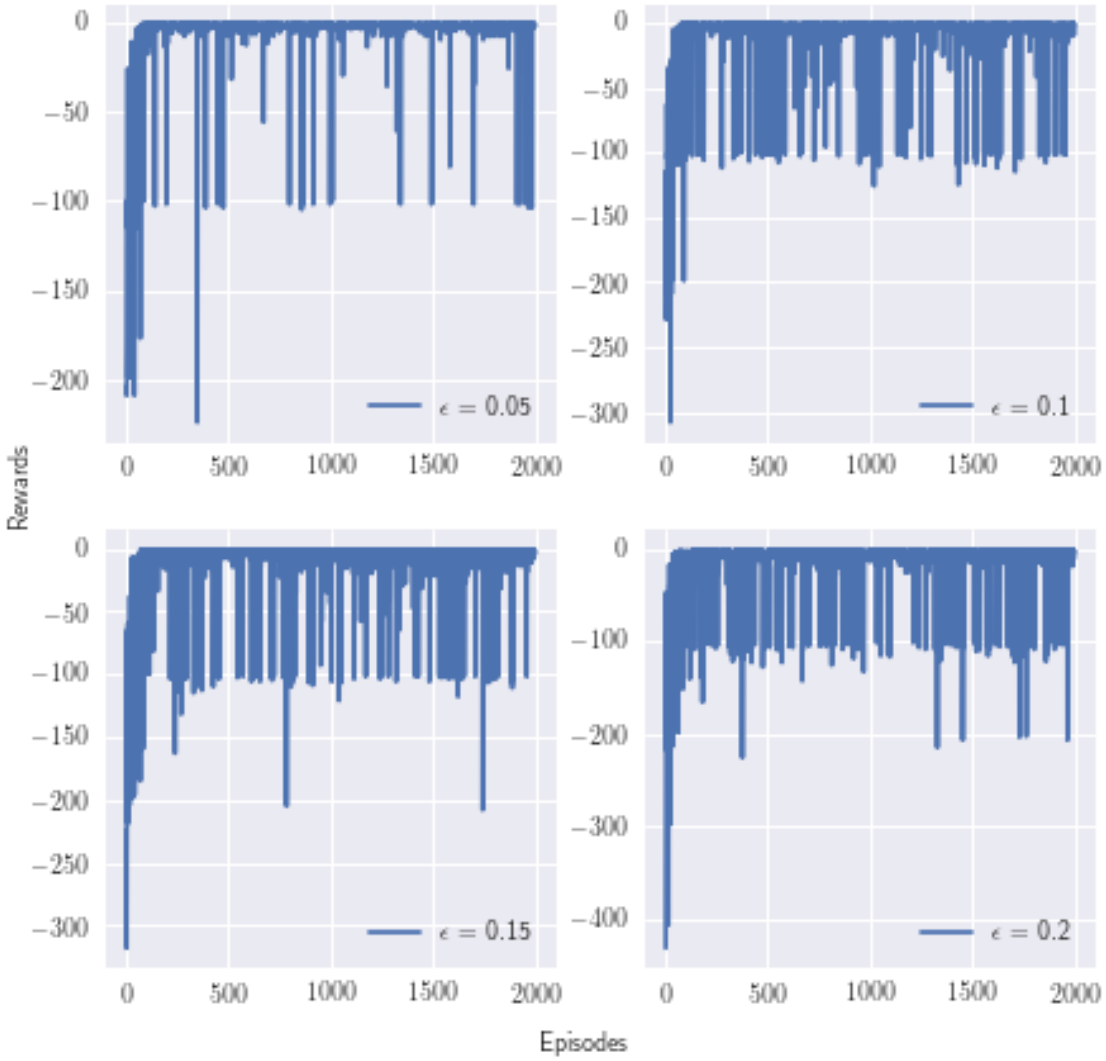

C-9-Q-learning-e-greedy $\gamma = 0.976$ /$\epsilon = 0.05$
Rewards
Episodes
$\alpha = 0.1832$
$\alpha = 0.15$
$\alpha = 0.12$
$\alpha = 0.2$
0
−50
−100
−150
−200
−250
−300
0
500
1000
1500
2000

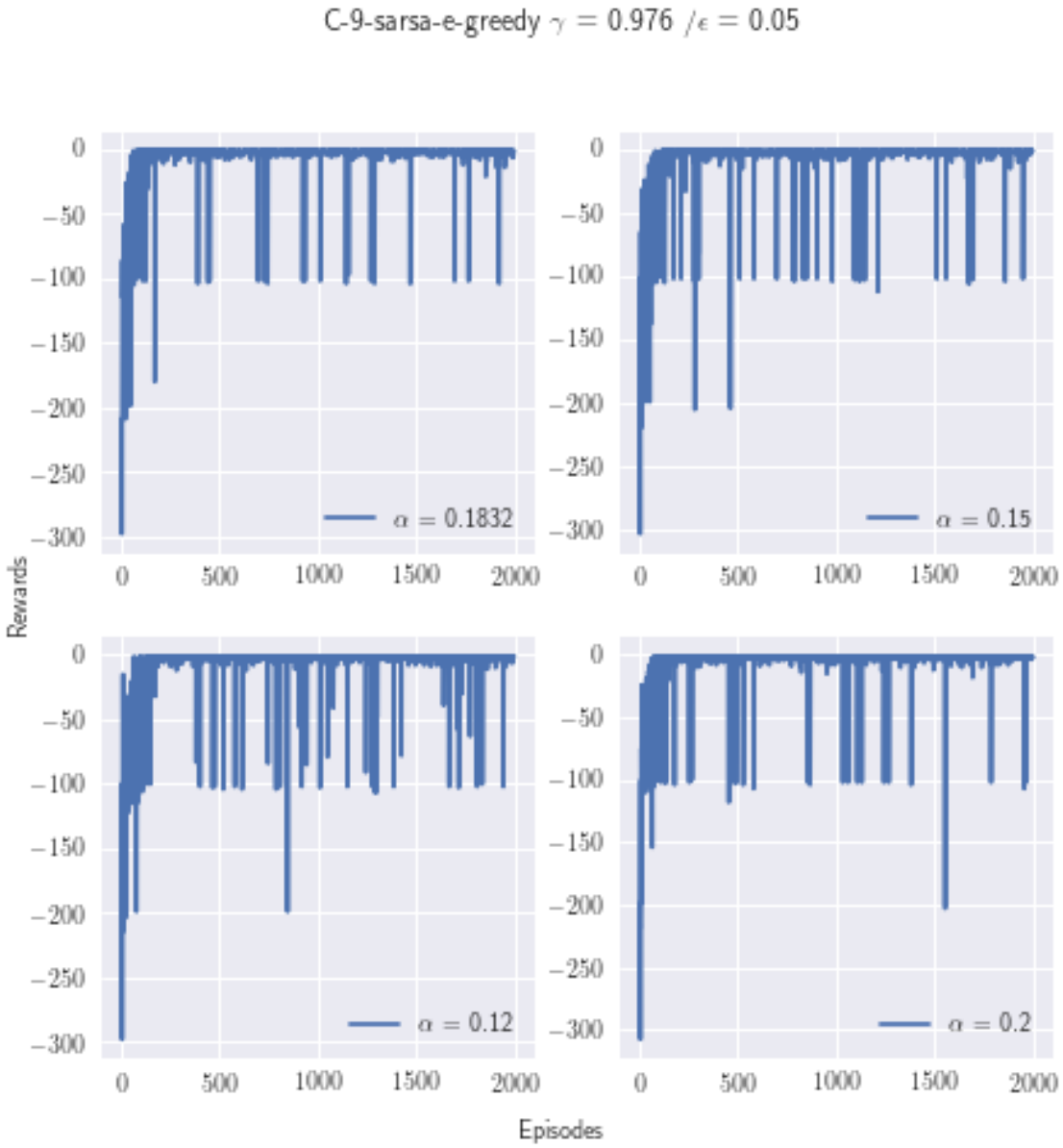

C-9-Q-learning-e-greedy $\alpha = 0.1832$ $\epsilon = 0.05$
Rewards
Episodes
$\gamma = 0.976$
$\gamma = 0.99$
$\gamma = 0.95$
$\gamma = 0.9$
0
−50
−100
−150
−200
−250
−300
0
500
1000
1500
2000

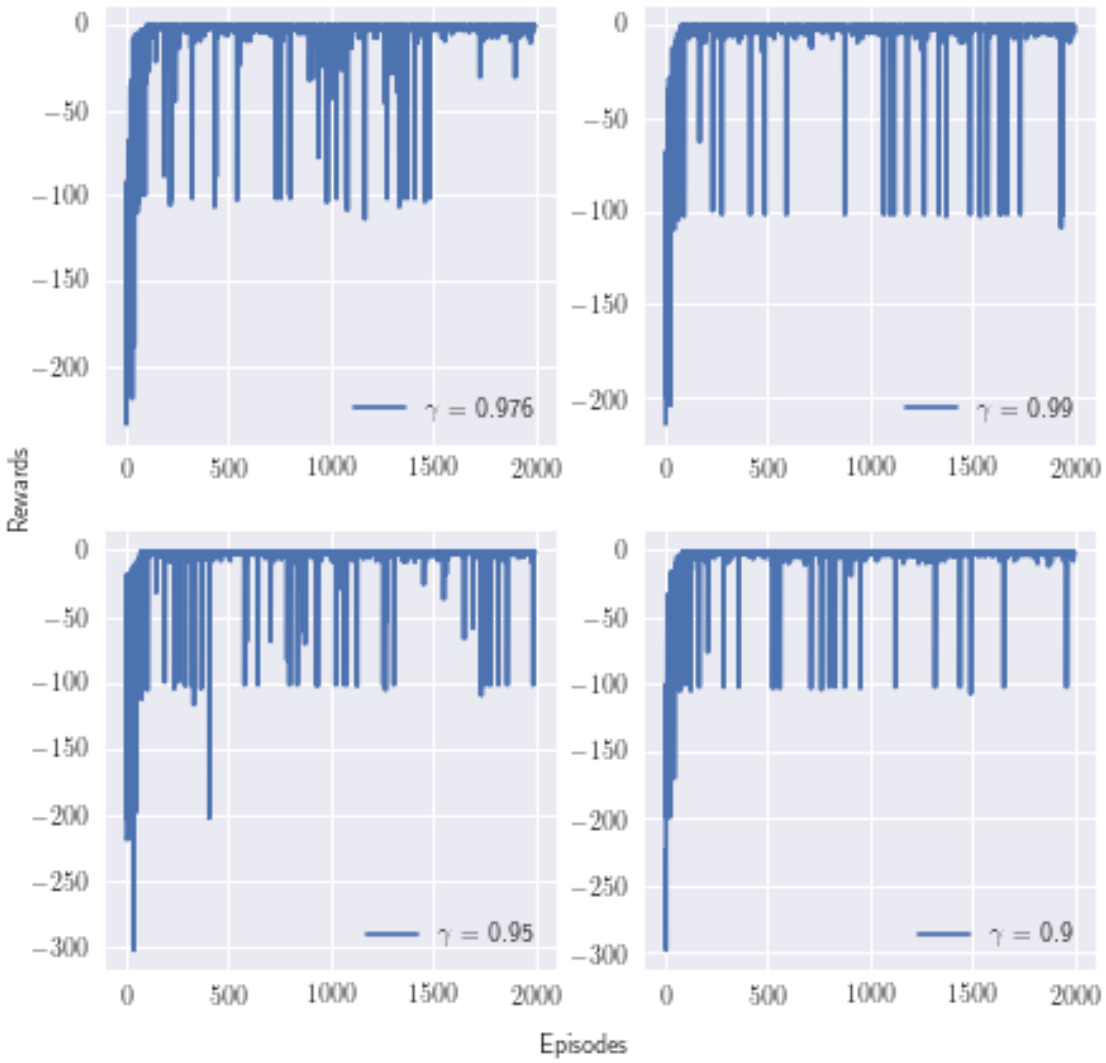

C-9-Q-learning-softmax $\gamma = 0.976$ $\alpha = 0.1832$
Rewards
Episodes
$\beta = 0.6001$
$\beta = 0.65$
$\beta = 0.7$
$\beta = 0.5$
0
−50
−100
−150
−200
−250
−300
500
1000
1500
2000

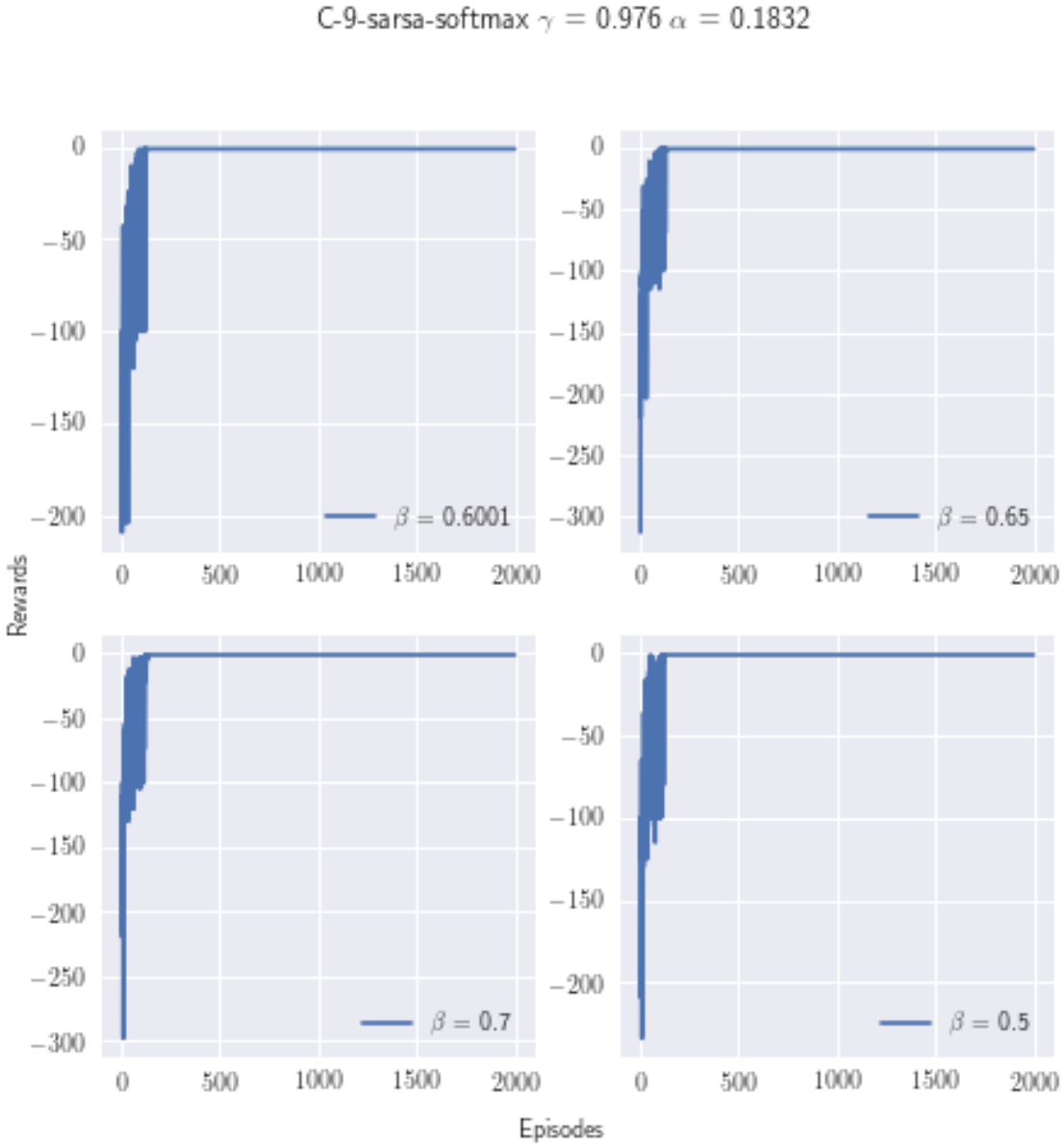

C-9-Q-learning-softmax $\gamma = 0.976$ $\beta = 0.6001$
Rewards
Episodes
$\alpha = 0.1832$
$\alpha = 0.15$
$\alpha = 0.12$
$\alpha = 0.2$

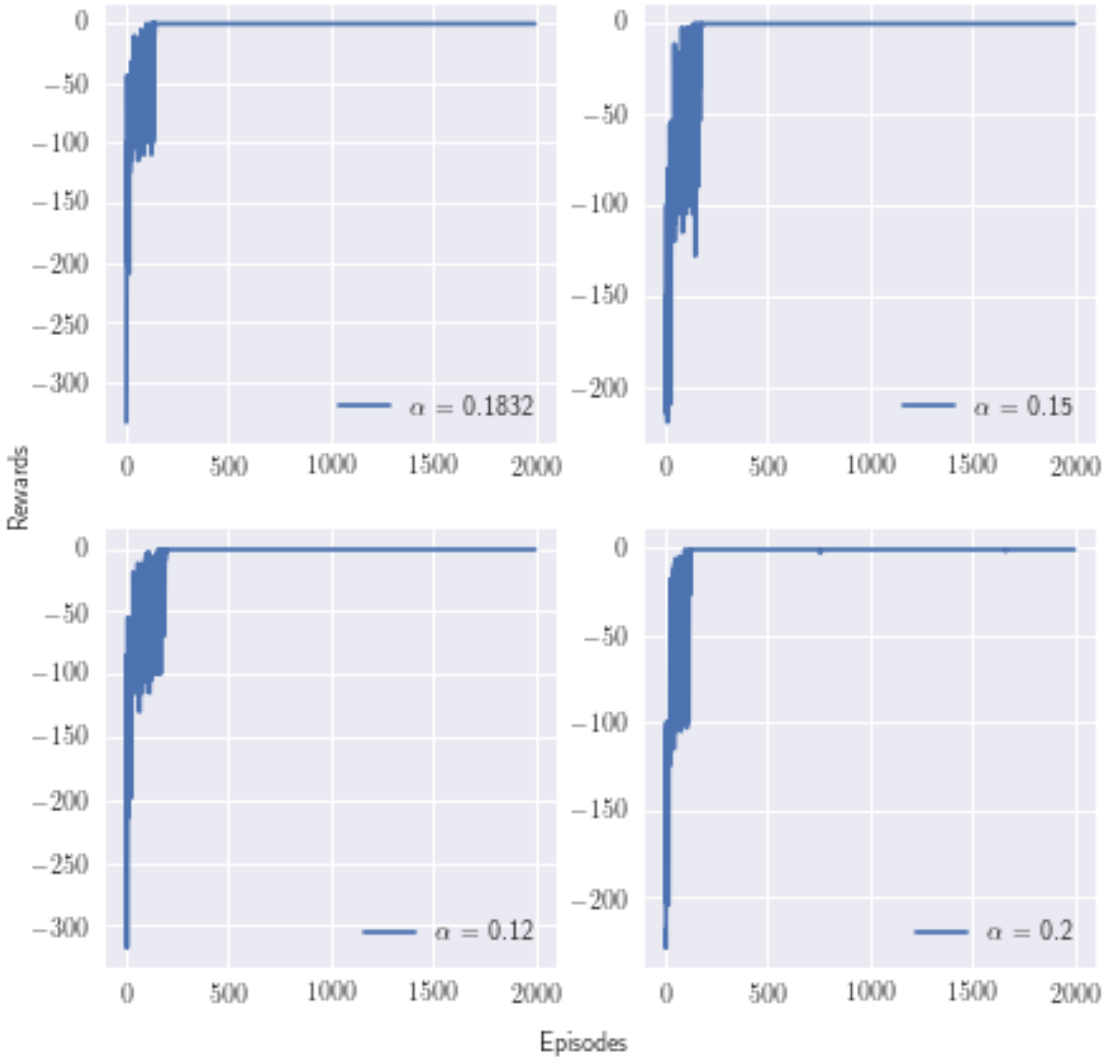

C-9-Q-learning-softmax $\alpha = 0.1832$ $\beta = 0.6001$
Rewards
Episodes
$\gamma = 0.976$
$\gamma = 0.99$
$\gamma = 0.95$
$\gamma = 0.9$

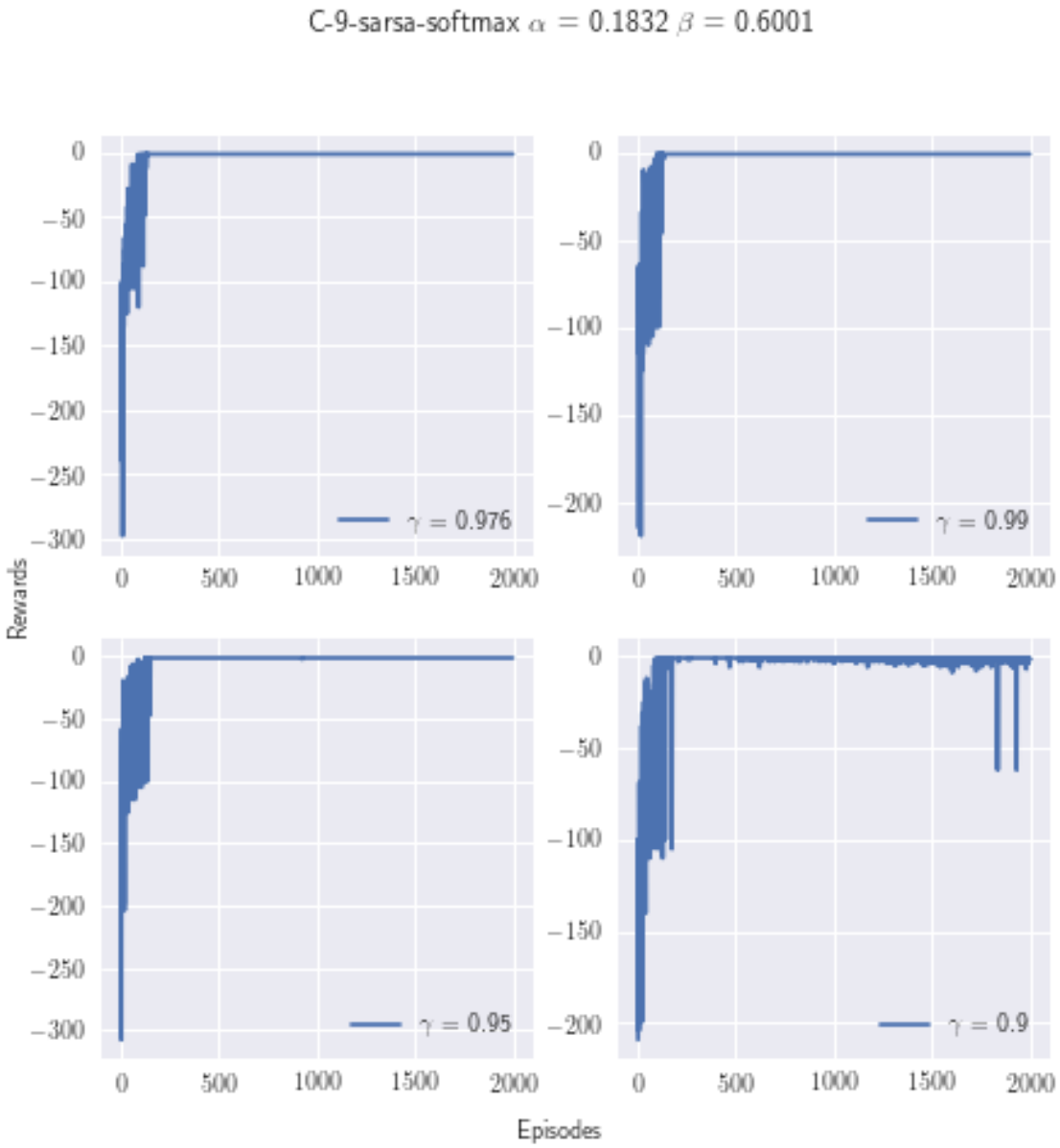

C-9-sarsa-e-greedy $\gamma = 0.976$ $\alpha = 0.1832$
Rewards
Episodes
$\epsilon = 0.05$
$\epsilon = 0.1$
$\epsilon = 0.15$
$\epsilon = 0.2$


C-9-sarsa-e-greedy $\gamma = 0.976$ /$\epsilon = 0.05$
Rewards
Episodes
$\alpha = 0.1832$
$\alpha = 0.15$
$\alpha = 0.12$
$\alpha = 0.2$
0
−50
−100
−150
−200
−250
−300
0
500
1000
1500
2000


C-9-sarsa-e-greedy $\alpha = 0.1832$ $\epsilon = 0.05$
Rewards
Episodes
$\gamma = 0.976$
$\gamma = 0.99$
$\gamma = 0.95$
$\gamma = 0.9$
0
−50
−100
−150
−200
−250
−300
0
500
1000
1500
2000

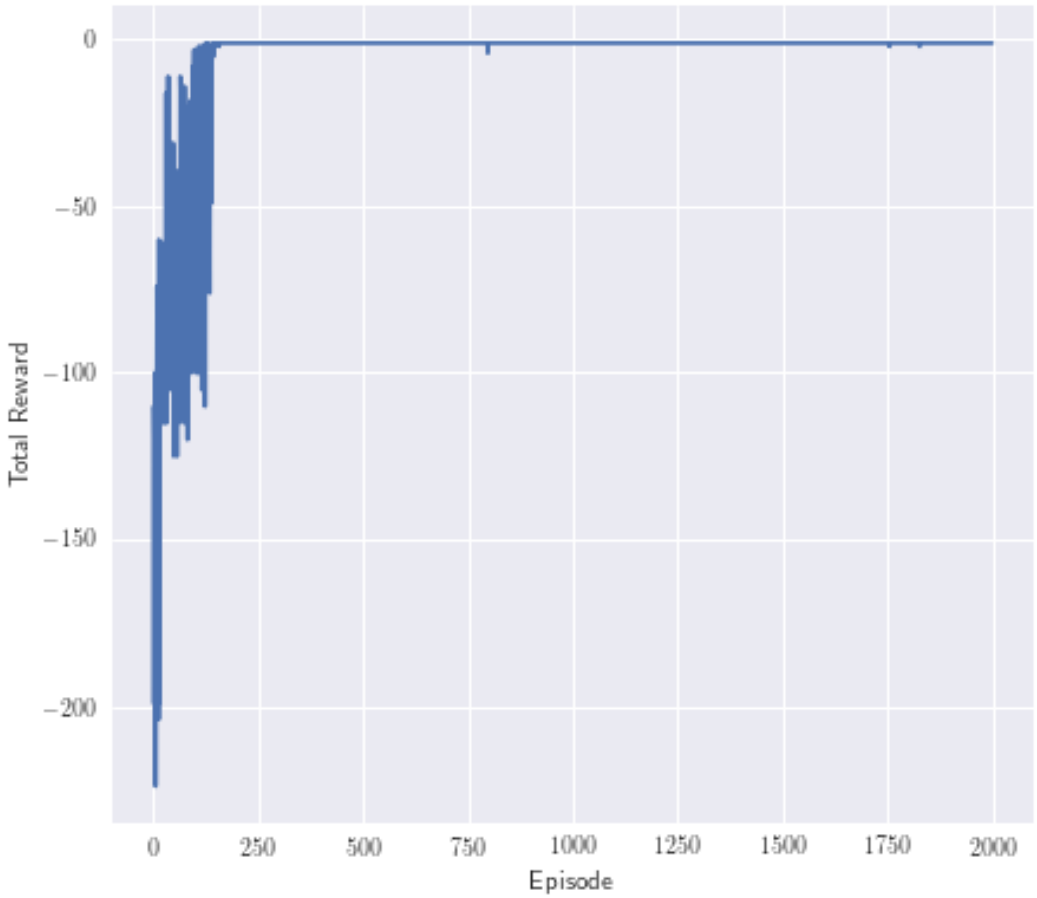

C-9-sarsa-softmax $\gamma = 0.976$ $\alpha = 0.1832$
Rewards
Episodes
$\beta = 0.6001$
$\beta = 0.65$
$\beta = 0.7$
$\beta = 0.5$

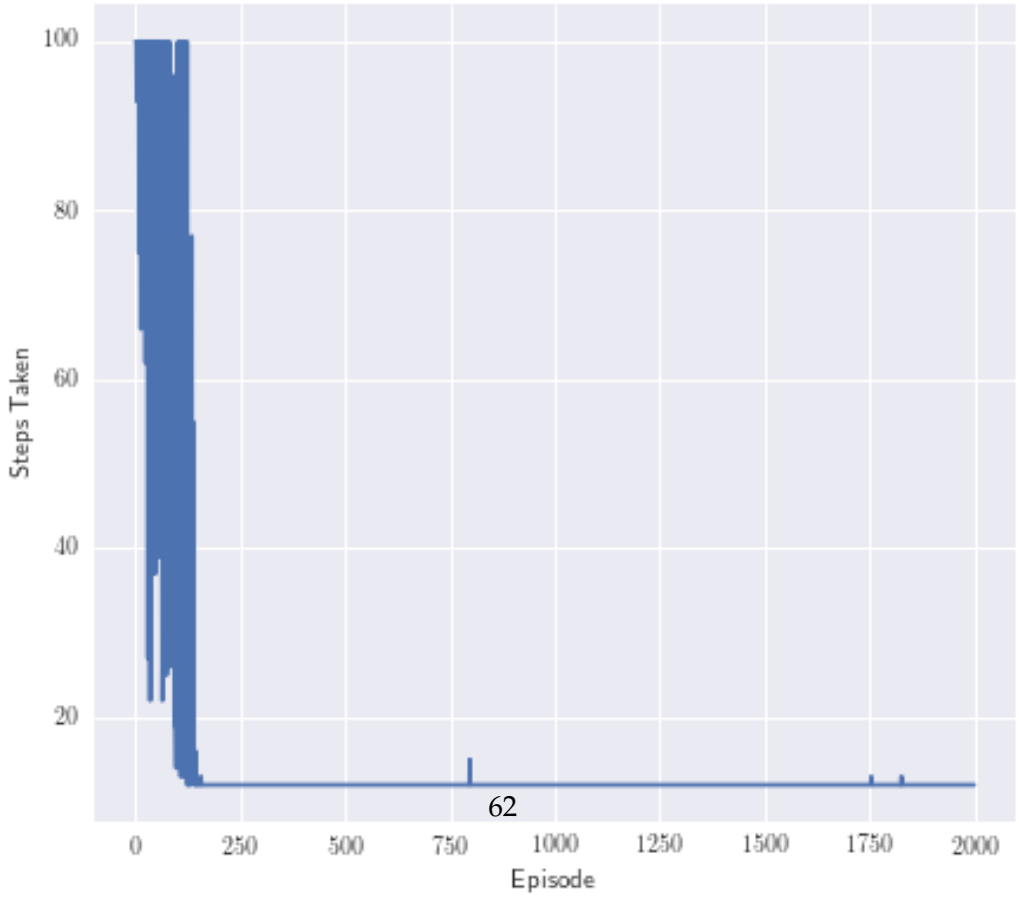

C-9-sarsa-softmax $\gamma = 0.976$ $\beta = 0.6001$
Rewards
Episodes
$\alpha = 0.1832$
$\alpha = 0.15$
$\alpha = 0.12$
$\alpha = 0.2$
0
−50
−100
−150
−200
−250
−300
0
500
1000
1500
2000

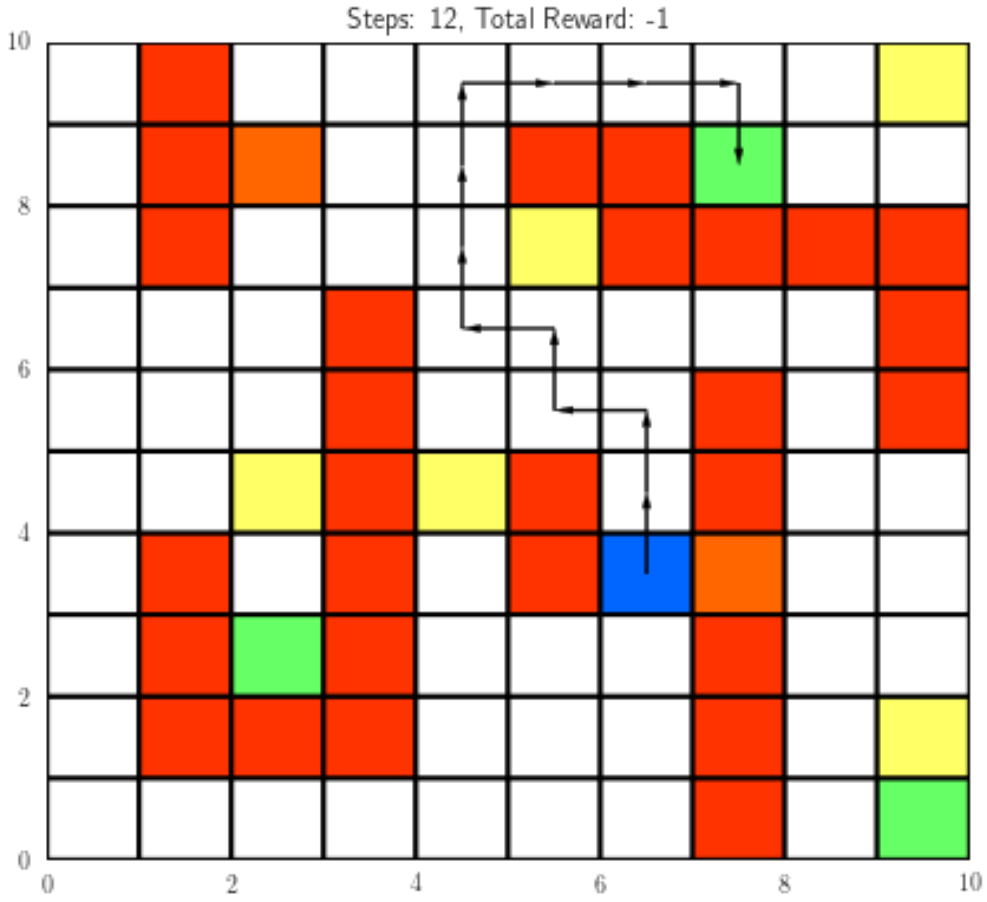


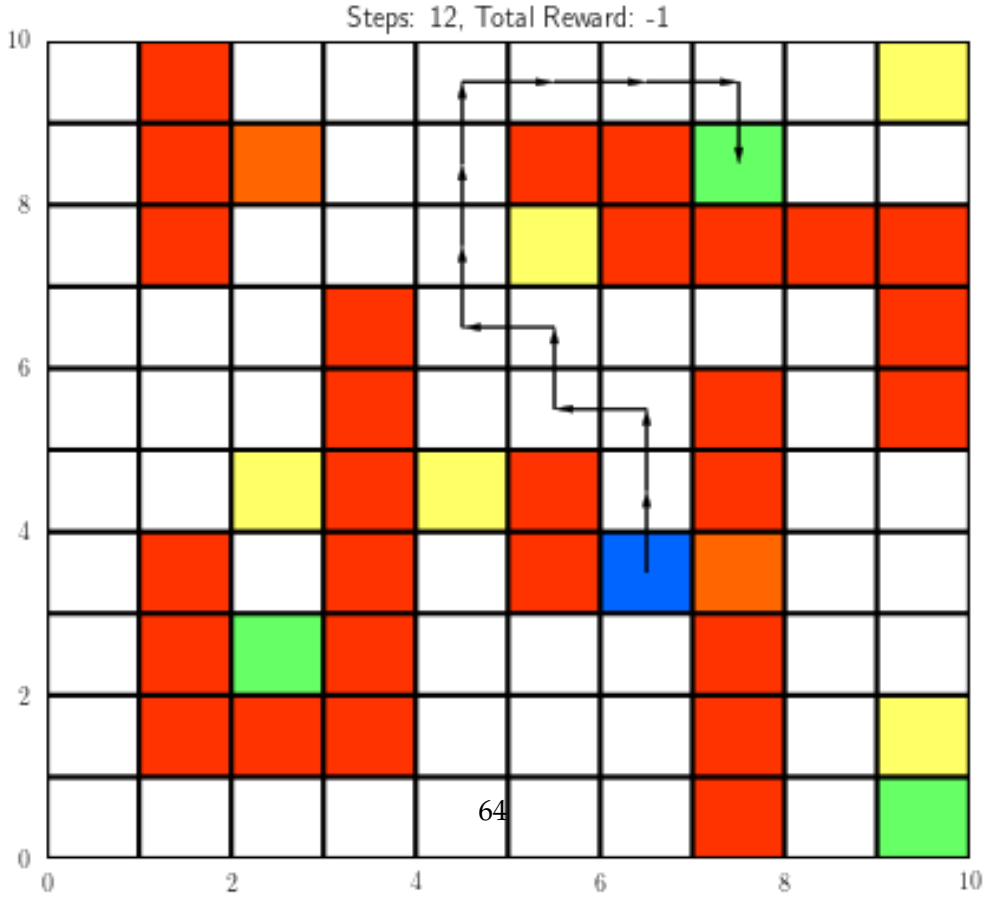
[ ]:
```
Q = best_plots(algorithm, policy, alpha,  epsilon, beta , gamma )
```

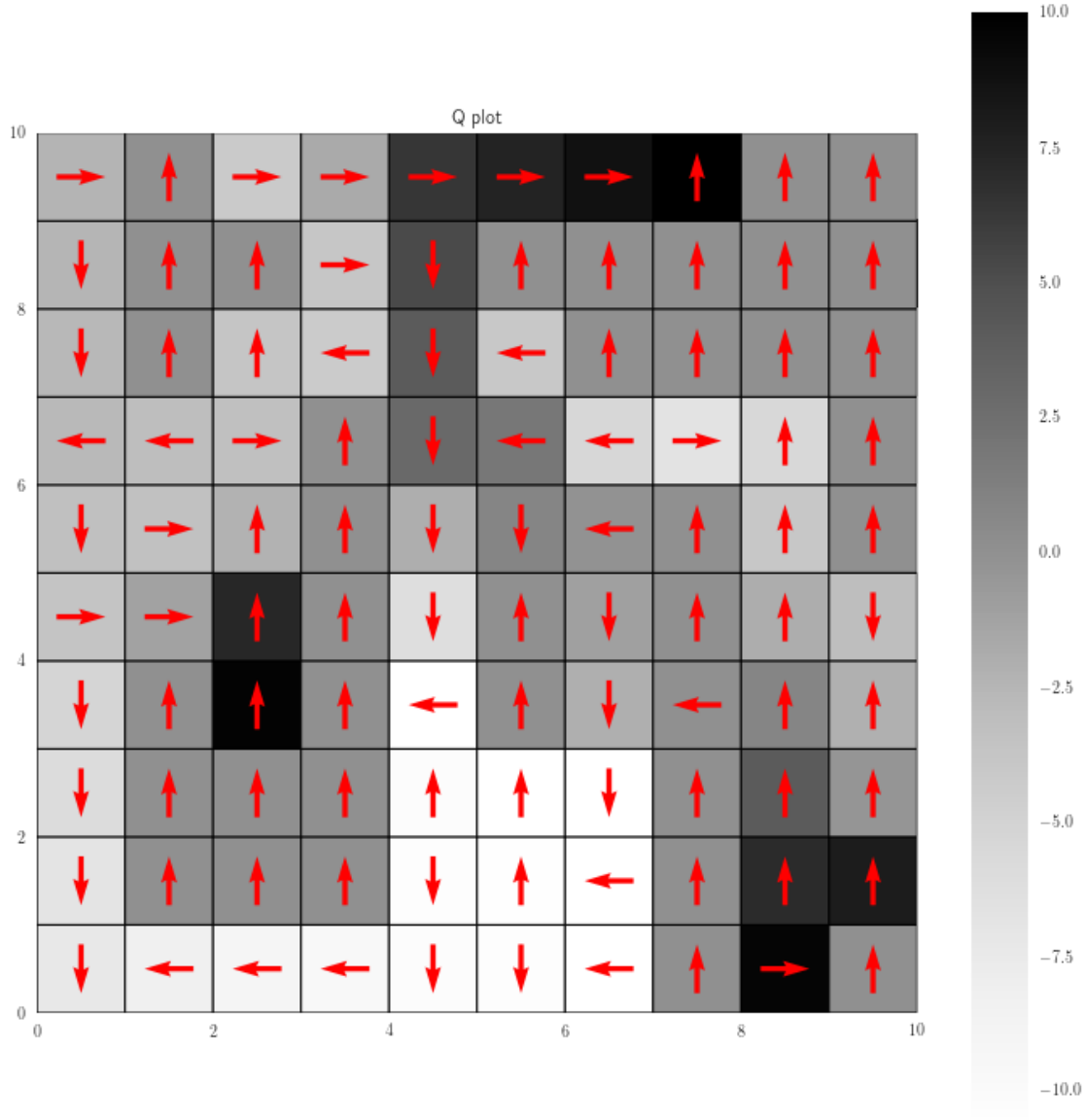
```
100%|| 2000/2000 [00:16<00:00, 118.41it/s]
```


0
−50
−100
−150
−200
Total Reward
0
250
500
750
1000
1250
1500
1750
2000
Episode


100
80
60
40
20
Steps Taken
0
250
500
750
1000
1250
1500
1750
2000
Episode


Steps: 12, Total Reward: -1
10
8
6
4
2
0
0
2
4
6
8
10


Steps: 12, Total Reward: -1
10
8
6
4
2
0
0
2
4
6
8
10



6.8 config 10

[26]:
```
NUM_CONFIG = 10

config_settings = configurations_l[NUM_CONFIG]

# create environment
env = create_env(**config_settings._asdict())
```

```
sweep_config = {
    "name" : f"{NUM_CONFIG}-config-sweep",
    "method": "random",
    "parameters": {
        "algorithm": {
            "values": ['sarsa', 'q_learning'],
        },
        "policy": {
            "values": ['softmax', 'epsilon_greedy'],
        },
        "epsilon": {
            "min": 0.0,
            "max": 1.0,
        },
        "alpha": {
            "min": 0.01,
            "max": 0.2,
        },
        "gamma": {
            "min": 0.5,
            "max": 1.0,
        },
        "beta": {
            "min": 0.5,
            "max": 1.5,
        }
    }
}
```

[ ]:
```
sweep_id = wandb.sweep(sweep_config, project='RLPA1')
```

[ ]:
```
wandb.agent(sweep_id, run, count=20)
```

[ ]:
```

```

6.8.1 Plotting

[27]:
```
alpha = 0.09535
l_alpha = [0.1, 0.15, 0.05]
beta = 1.254
l_beta = [1.1, 1.0, 1.4]
gamma = 0.9549
l_gamma = [0.99, 0.90, 1.0]
epsilon = 0.05
l_epsilon = [0.1, 0.15, 0.2]
policy = 'e-greedy'
algorithm = 'q-learning'
```

[ ]:
```
matplotlib.style.available
```

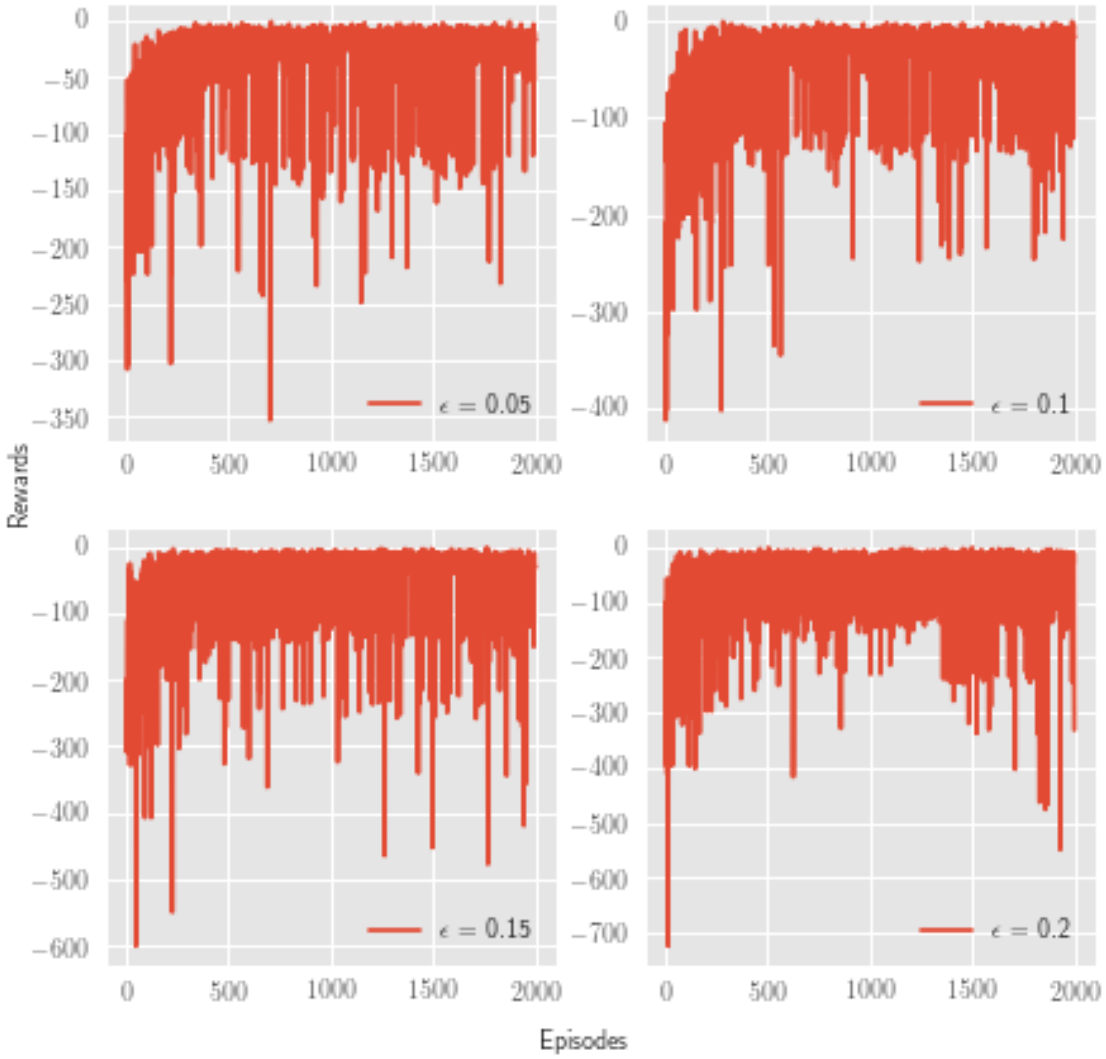
[ ]:
```
['Solarize_Light2',
 '_classic_test_patch',
 'bmh',
 'classic',
 'dark_background',
 'fast',
 'fivethirtyeight',
 'ggplot',
 'grayscale',
 'seaborn',
 'seaborn-bright',
 'seaborn-colorblind',
 'seaborn-dark',
 'seaborn-dark-palette',
 'seaborn-darkgrid',
 'seaborn-deep',
 'seaborn-muted',
 'seaborn-notebook',
 'seaborn-paper',
 'seaborn-pastel',
 'seaborn-poster',
 'seaborn-talk',
 'seaborn-ticks',
 'seaborn-white',
 'seaborn-whitegrid',
 'tableau-colorblind10']
```

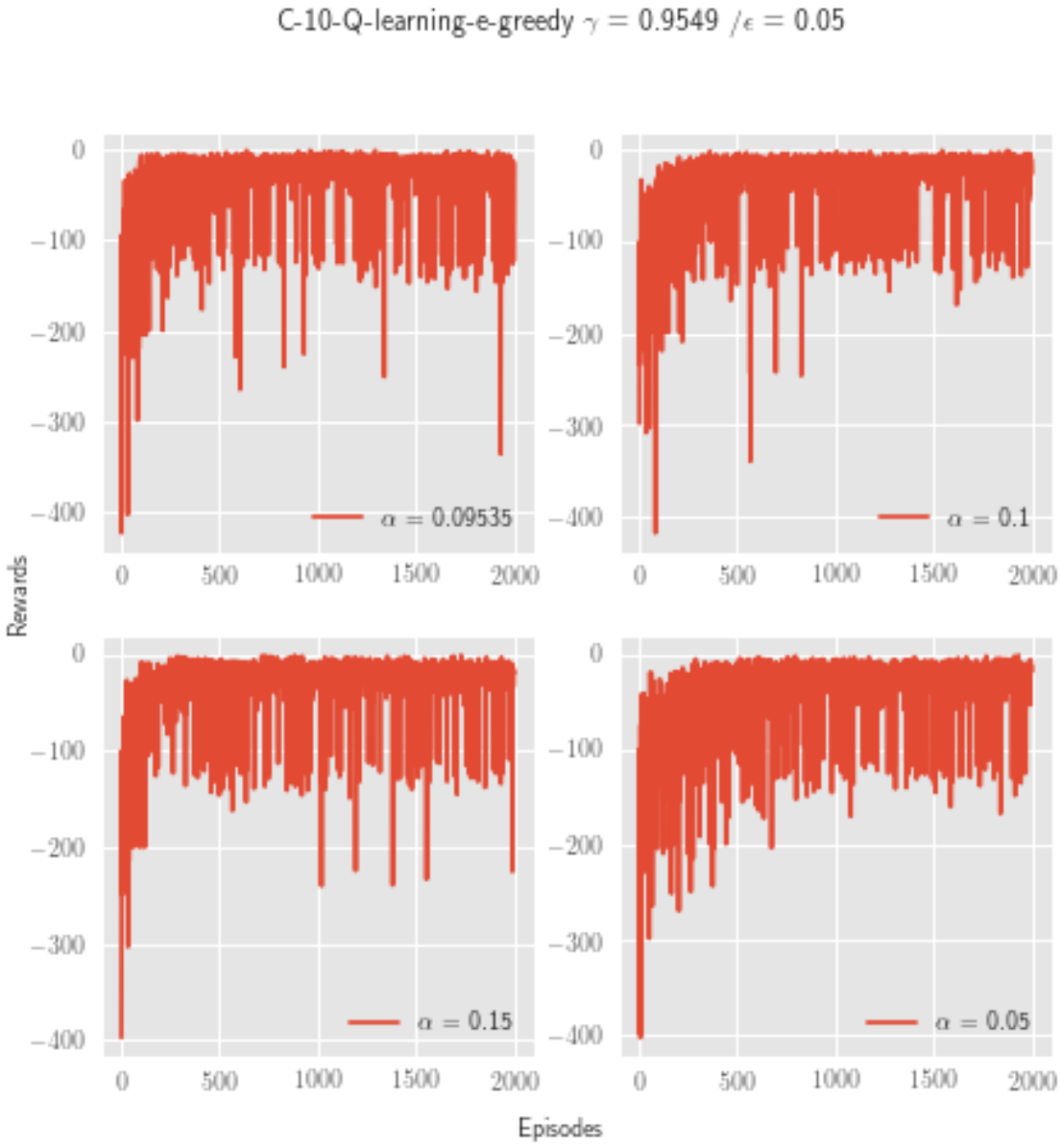
[28]:
```
plt.style.use('ggplot')
```

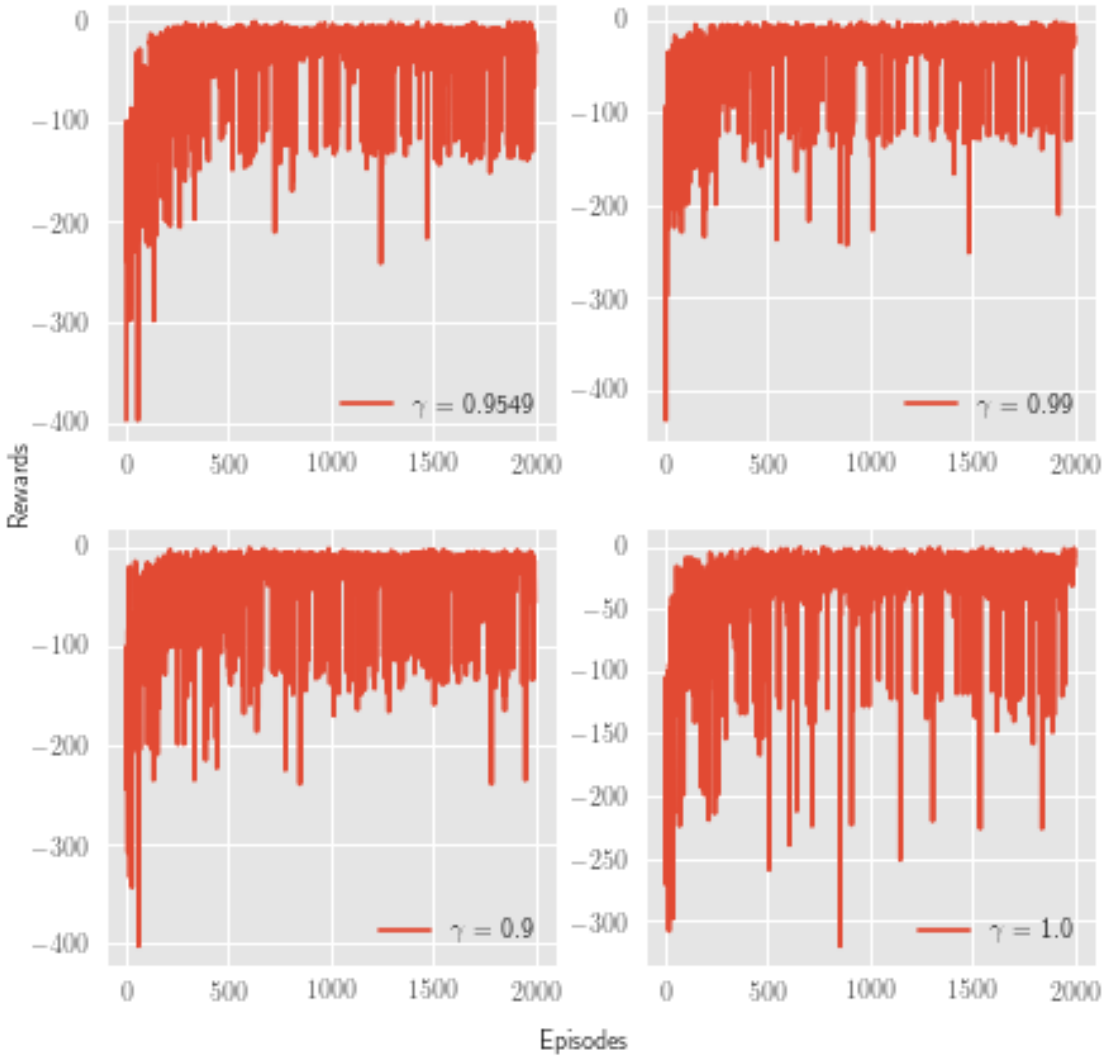
[29]:
```
plot_all(alpha, l_alpha, beta, l_beta, gamma, l_gamma, epsilon, l_epsilon)
```

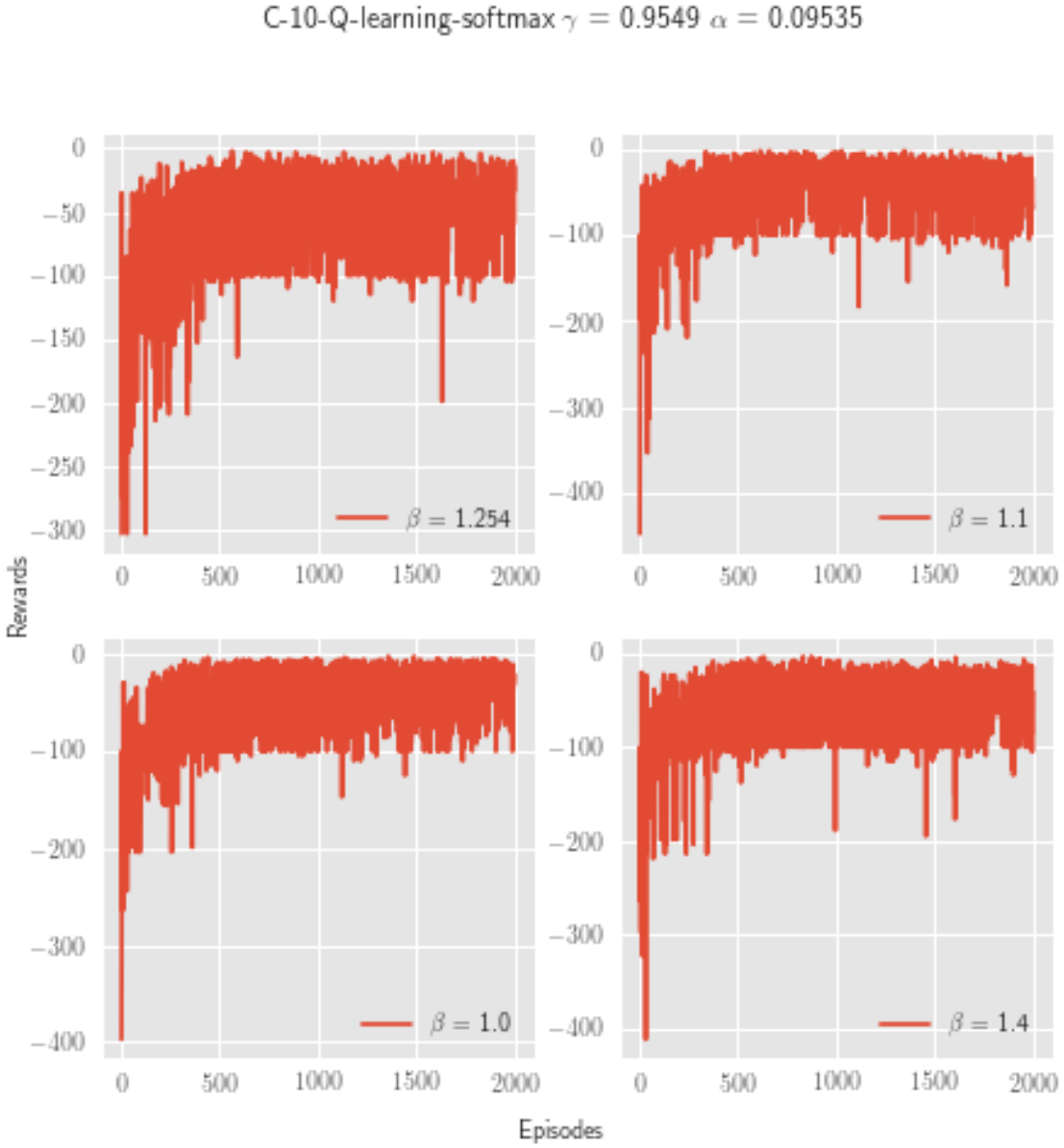
```
100%|| 2000/2000 [00:15<00:00, 126.16it/s]
100%|| 2000/2000 [00:16<00:00, 118.11it/s]
100%|| 2000/2000 [00:17<00:00, 113.77it/s]
100%|| 2000/2000 [00:18<00:00, 105.63it/s]
100%|| 2000/2000 [00:17<00:00, 114.68it/s]
100%|| 2000/2000 [00:17<00:00, 116.42it/s]
100%|| 2000/2000 [00:15<00:00, 133.17it/s]
100%|| 2000/2000 [00:17<00:00, 113.58it/s]
100%|| 2000/2000 [00:14<00:00, 134.37it/s]
100%|| 2000/2000 [00:14<00:00, 142.18it/s]
100%|| 2000/2000 [00:16<00:00, 118.67it/s]
100%|| 2000/2000 [00:17<00:00, 113.43it/s]
100%|| 2000/2000 [00:48<00:00, 41.07it/s]
100%|| 2000/2000 [00:45<00:00, 43.79it/s]
100%|| 2000/2000 [00:43<00:00, 46.00it/s]
100%|| 2000/2000 [00:52<00:00, 38.19it/s]
```

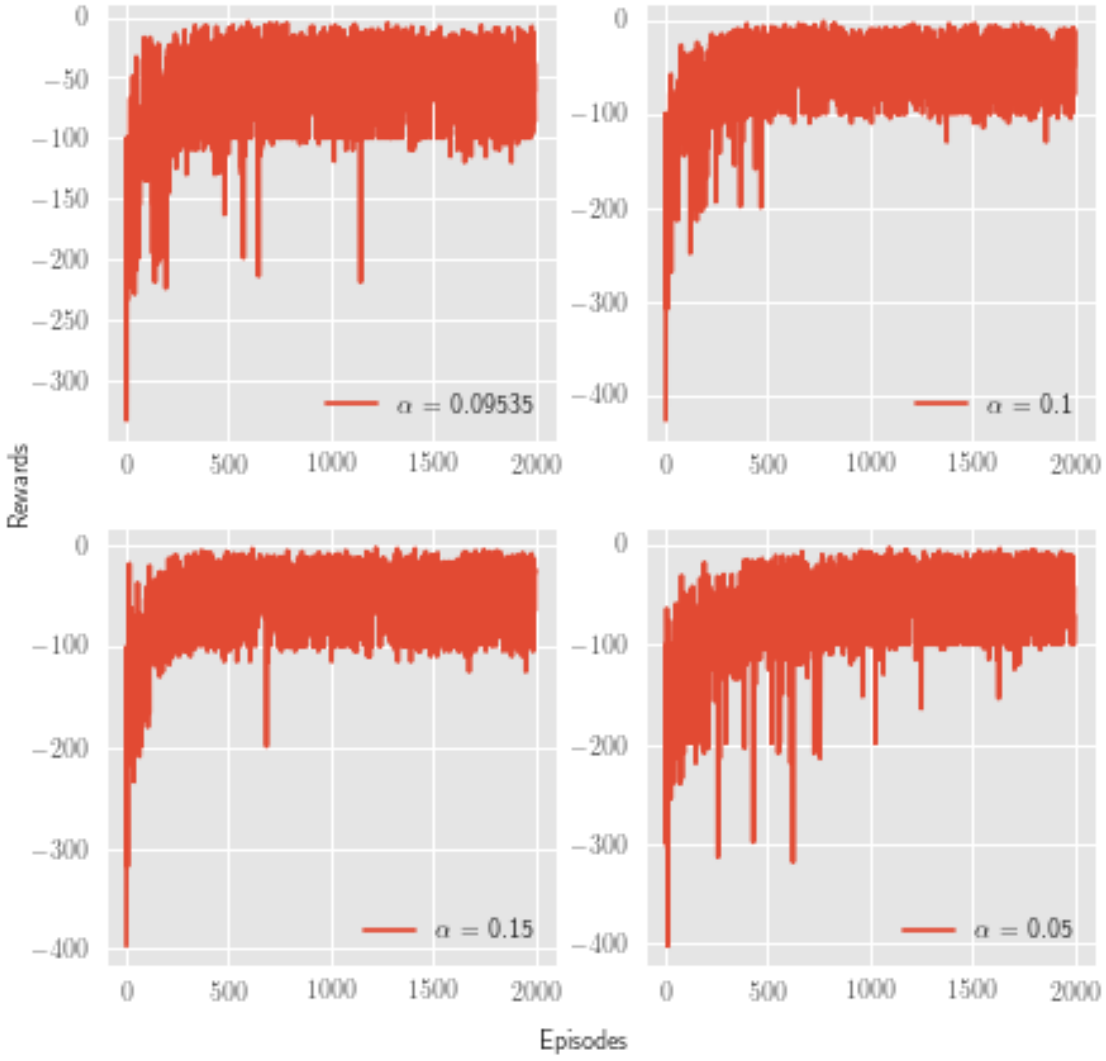
```
100%|| 2000/2000 [00:50<00:00, 39.63it/s]
100%|| 2000/2000 [00:48<00:00, 41.26it/s]
100%|| 2000/2000 [00:48<00:00, 41.40it/s]
100%|| 2000/2000 [00:53<00:00, 37.63it/s]
100%|| 2000/2000 [00:48<00:00, 40.98it/s]
100%|| 2000/2000 [00:40<00:00, 49.00it/s]
100%|| 2000/2000 [00:59<00:00, 33.35it/s]
100%|| 2000/2000 [00:42<00:00, 47.13it/s]
100%|| 2000/2000 [00:13<00:00, 145.75it/s]
100%|| 2000/2000 [00:15<00:00, 130.82it/s]
100%|| 2000/2000 [00:17<00:00, 113.00it/s]
100%|| 2000/2000 [00:17<00:00, 113.69it/s]
100%|| 2000/2000 [00:14<00:00, 136.15it/s]
100%|| 2000/2000 [00:15<00:00, 130.19it/s]
100%|| 2000/2000 [00:15<00:00, 130.75it/s]
100%|| 2000/2000 [00:17<00:00, 117.28it/s]
100%|| 2000/2000 [00:15<00:00, 126.66it/s]
100%|| 2000/2000 [00:14<00:00, 137.14it/s]
100%|| 2000/2000 [00:14<00:00, 137.79it/s]
100%|| 2000/2000 [00:14<00:00, 139.19it/s]
100%|| 2000/2000 [00:44<00:00, 44.75it/s]
100%|| 2000/2000 [00:41<00:00, 48.51it/s]
100%|| 2000/2000 [00:39<00:00, 50.83it/s]
100%|| 2000/2000 [00:46<00:00, 42.89it/s]
100%|| 2000/2000 [00:44<00:00, 45.36it/s]
100%|| 2000/2000 [00:43<00:00, 45.97it/s]
100%|| 2000/2000 [00:41<00:00, 47.66it/s]
100%|| 2000/2000 [00:49<00:00, 40.20it/s]
100%|| 2000/2000 [00:48<00:00, 41.15it/s]
100%|| 2000/2000 [00:37<00:00, 52.94it/s]
100%|| 2000/2000 [00:58<00:00, 34.45it/s]
100%|| 2000/2000 [00:36<00:00, 54.52it/s]
```

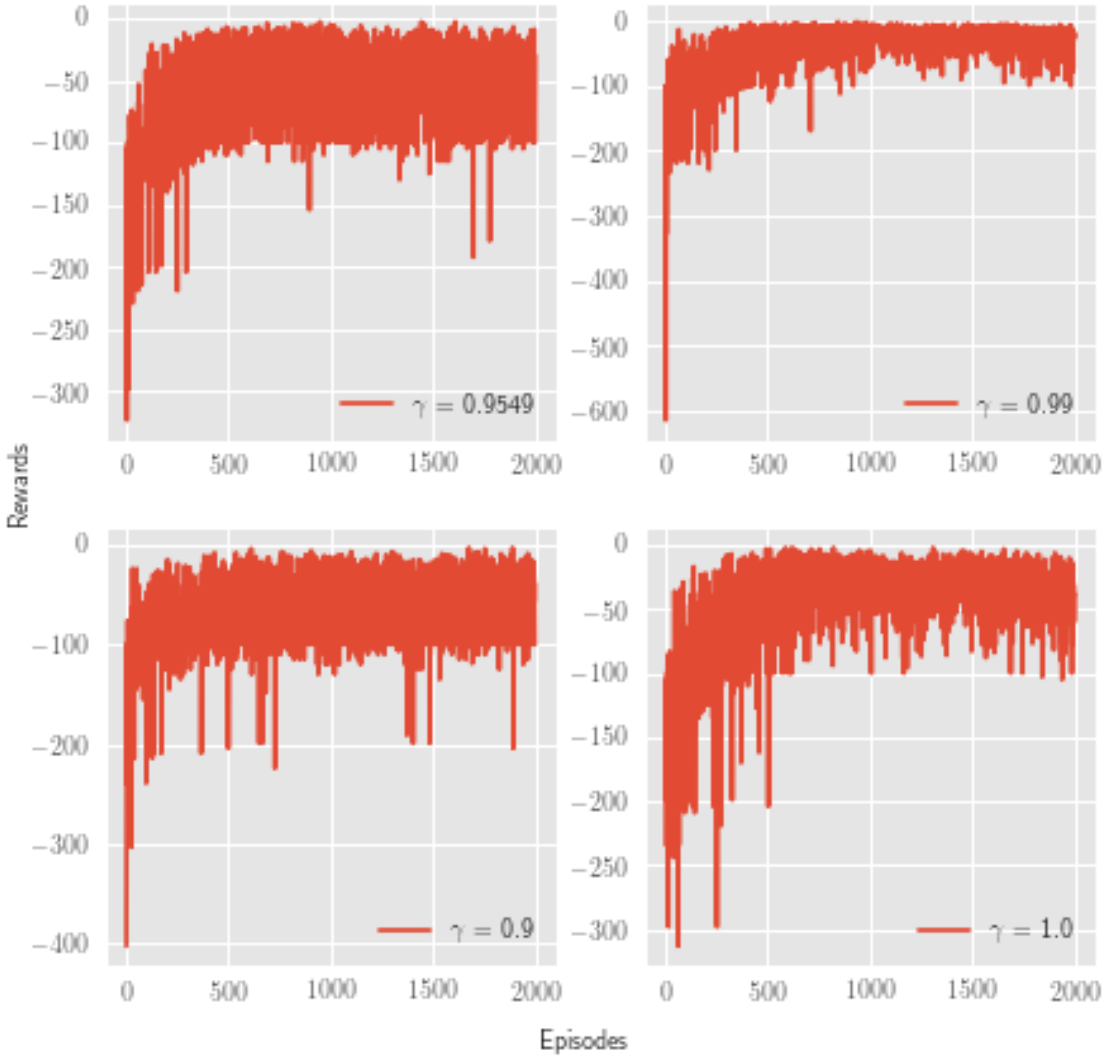

C-10-Q-learning-e-greedy $\gamma = 0.9549$ $\alpha = 0.09535$
Rewards
Episodes
$\epsilon = 0.05$
$\epsilon = 0.1$
$\epsilon = 0.15$
$\epsilon = 0.2$

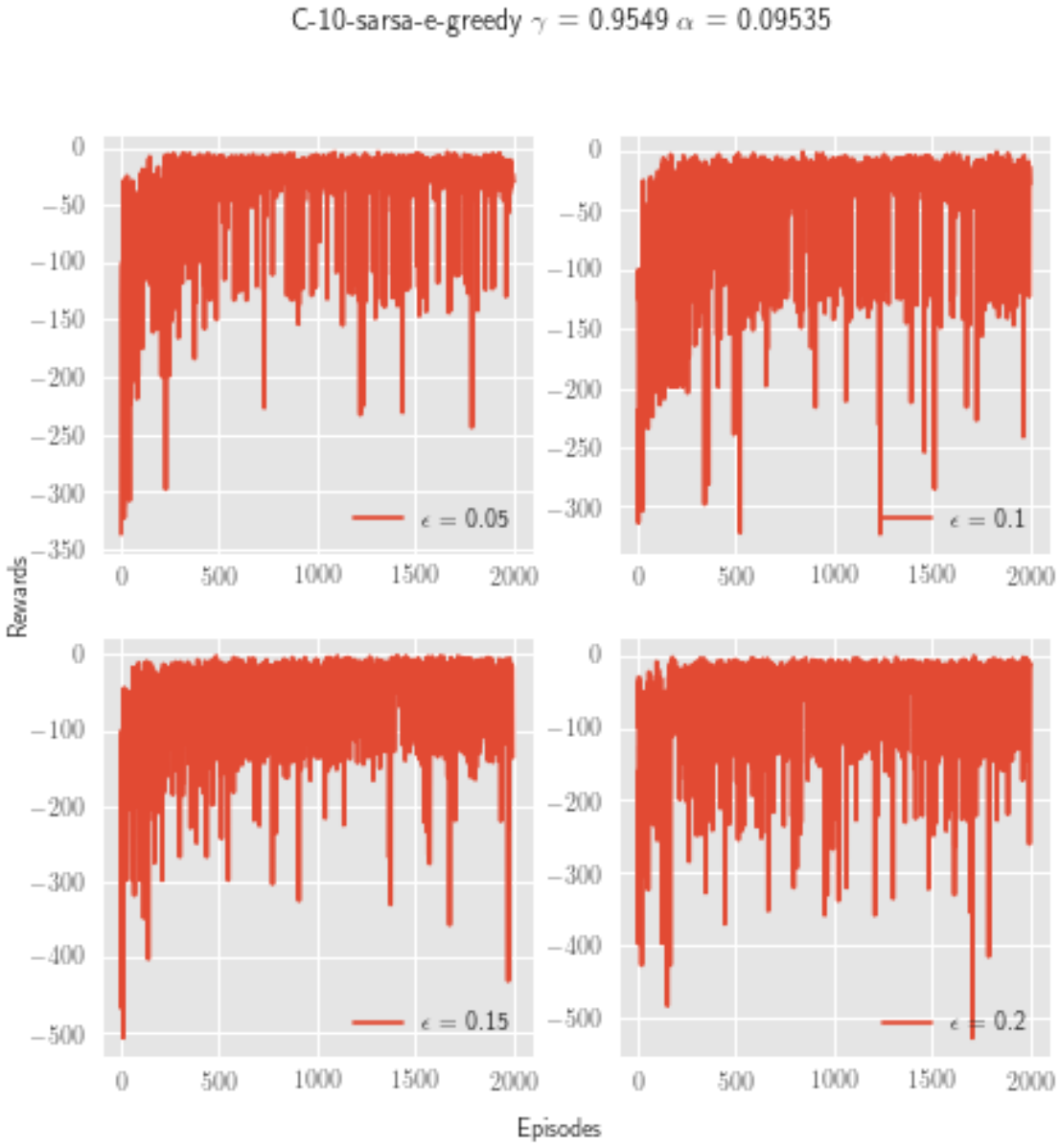

C-10-Q-learning-e-greedy $\gamma = 0.9549$ $/\epsilon = 0.05$
Rewards
Episodes
$\alpha = 0.09535$
$\alpha = 0.1$
$\alpha = 0.15$
$\alpha = 0.05$
0
−100
−200
−300
−400
0
500
1000
1500
2000

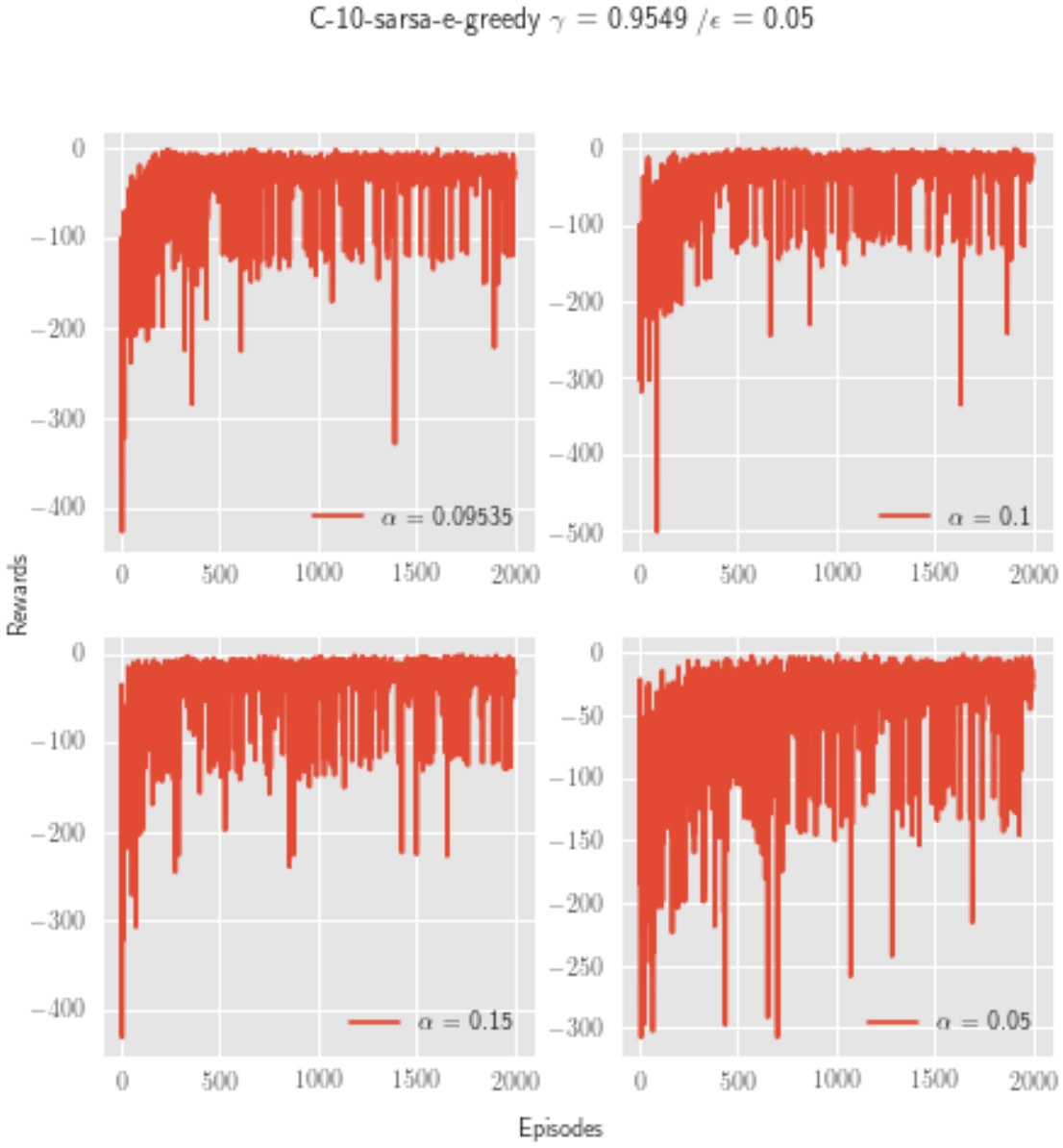

C-10-Q-learning-e-greedy $\alpha = 0.09535$ $\epsilon = 0.05$
Rewards
Episodes
$\gamma = 0.9549$
$\gamma = 0.99$
$\gamma = 0.9$
$\gamma = 1.0$

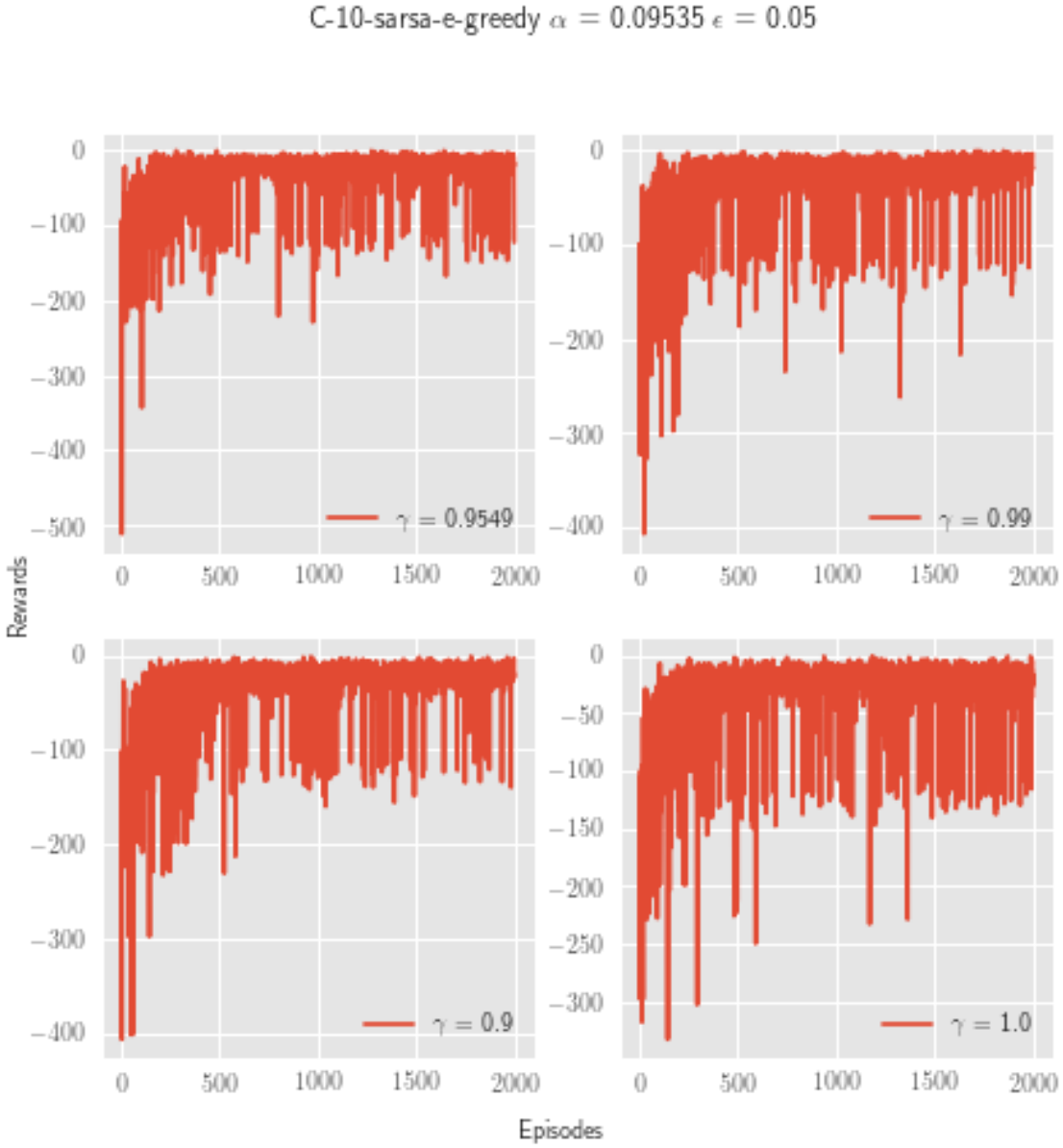

C-10-Q-learning-softmax $\gamma = 0.9549$ $\alpha = 0.09535$
Rewards
Episodes
$\beta = 1.254$
$\beta = 1.1$
$\beta = 1.0$
$\beta = 1.4$

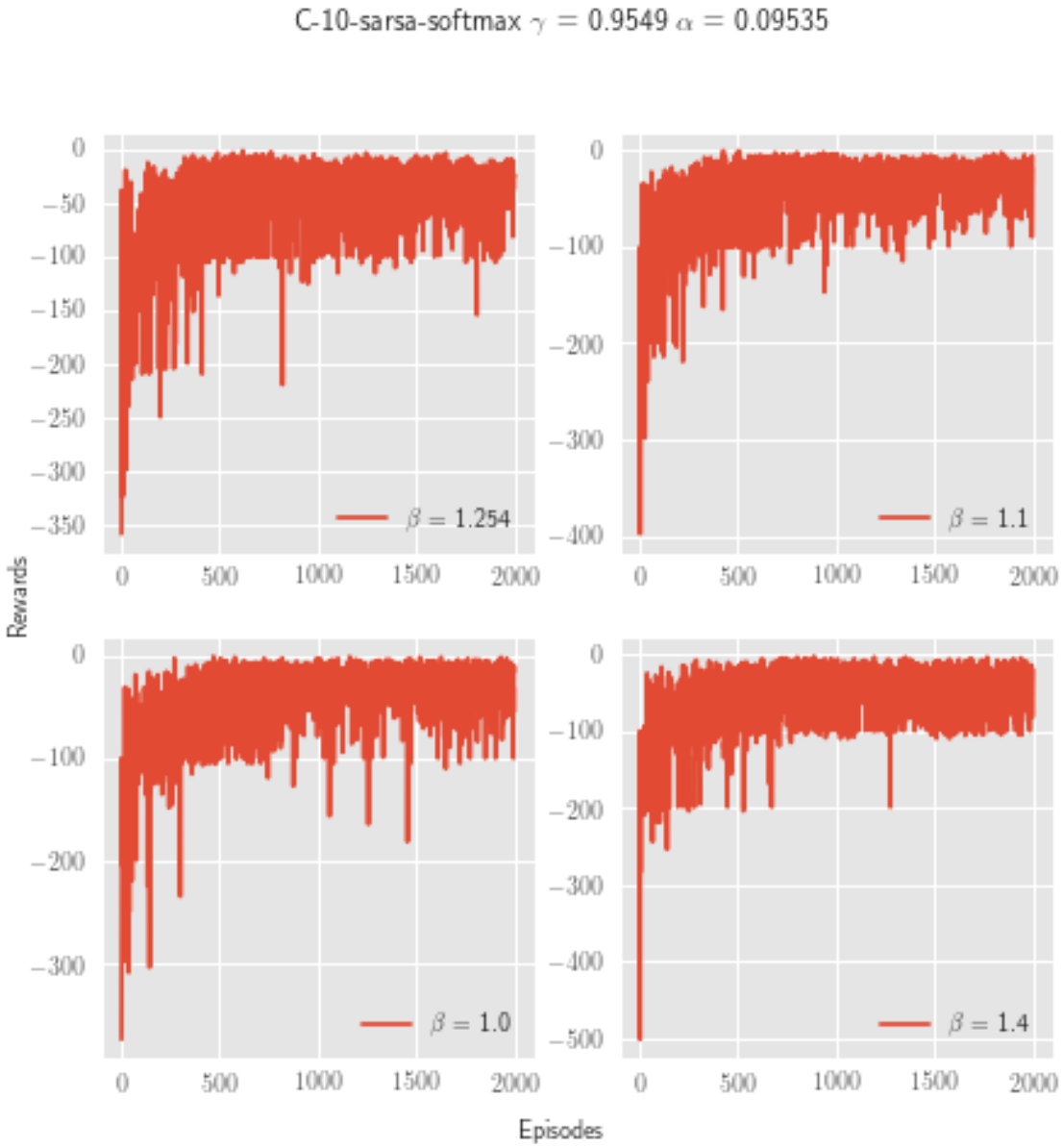

C-10-Q-learning-softmax $\gamma = 0.9549$ $\beta = 1.254$
Rewards
Episodes
$\alpha = 0.09535$
$\alpha = 0.1$
$\alpha = 0.15$
$\alpha = 0.05$

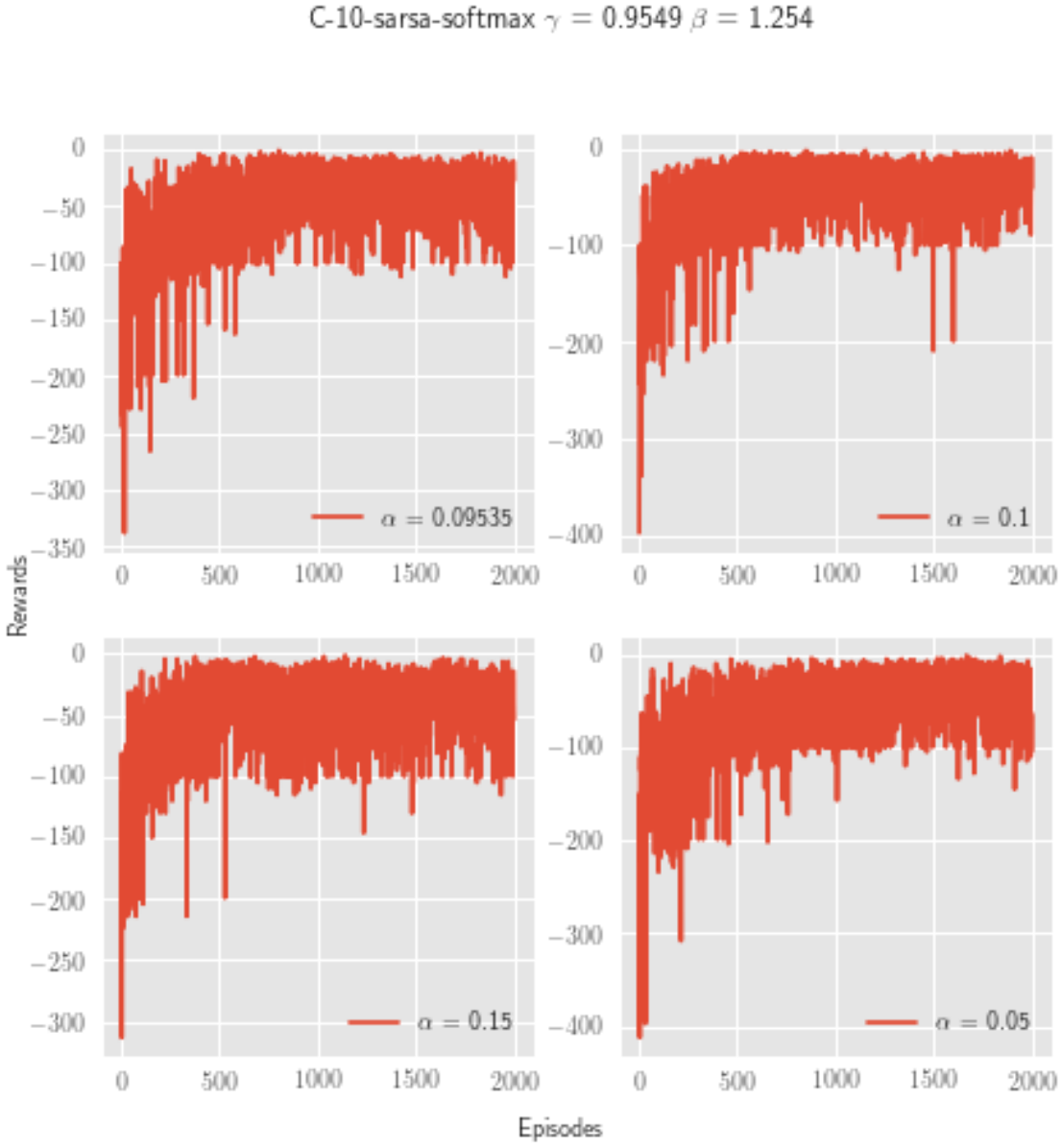

C-10-Q-learning-softmax $\alpha = 0.09535$ $\beta = 1.254$
Rewards
Episodes
$\gamma = 0.9549$
$\gamma = 0.99$
$\gamma = 0.9$
$\gamma = 1.0$

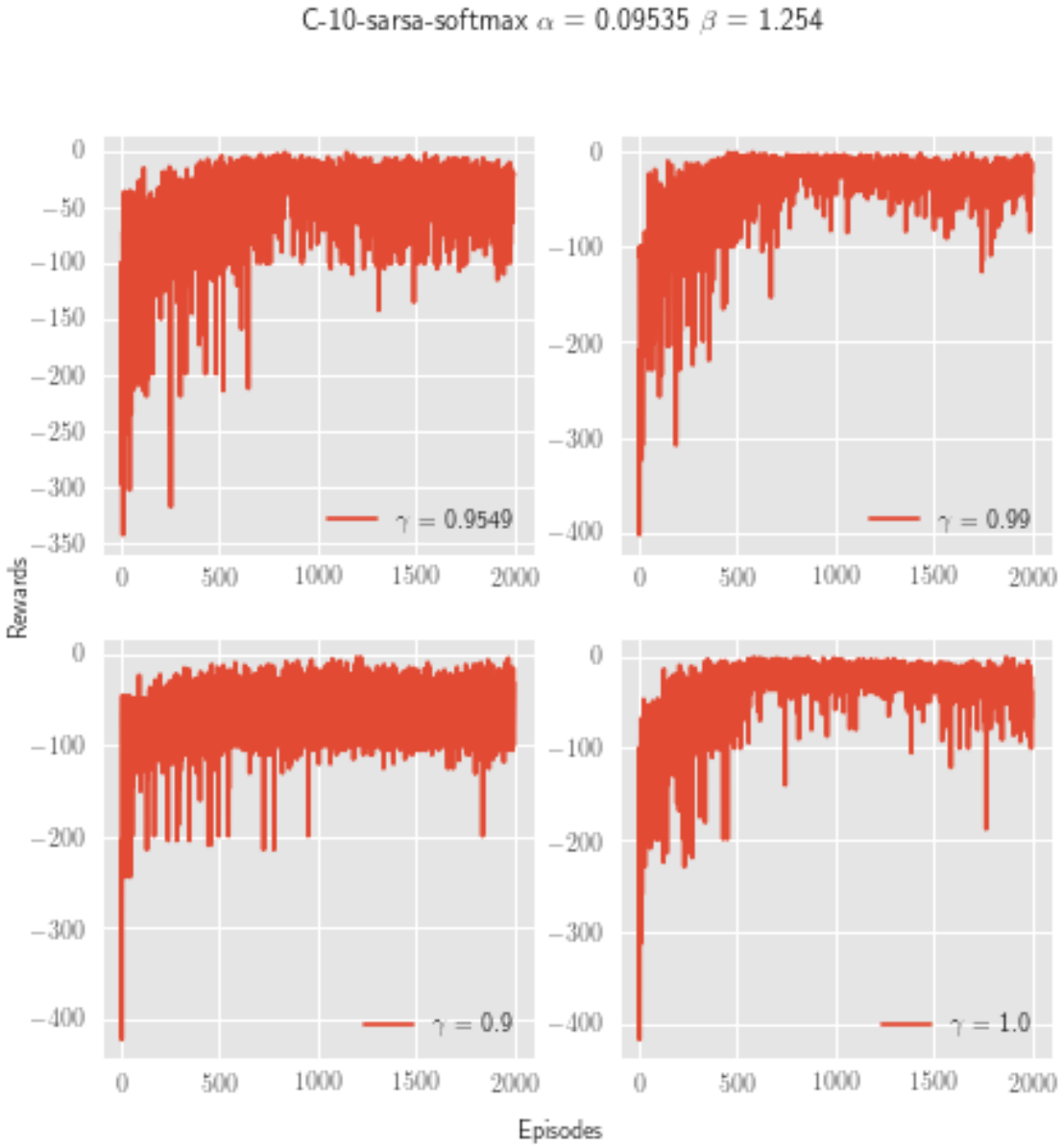

C-10-sarsa-e-greedy $\gamma = 0.9549$ $\alpha = 0.09535$
Rewards
Episodes
$\epsilon = 0.05$
$\epsilon = 0.1$
$\epsilon = 0.15$
$\epsilon = 0.2$


C-10-sarsa-e-greedy $\gamma = 0.9549$ /$\epsilon = 0.05$
Rewards
Episodes
$\alpha = 0.09535$
$\alpha = 0.1$
$\alpha = 0.15$
$\alpha = 0.05$


C-10-sarsa-e-greedy $\alpha = 0.09535$ $\epsilon = 0.05$
Rewards
Episodes
$\gamma = 0.9549$
$\gamma = 0.99$
$\gamma = 0.9$
$\gamma = 1.0$


C-10-sarsa-softmax $\gamma = 0.9549$ $\alpha = 0.09535$
Rewards
Episodes
$\beta = 1.254$
$\beta = 1.1$
$\beta = 1.0$
$\beta = 1.4$

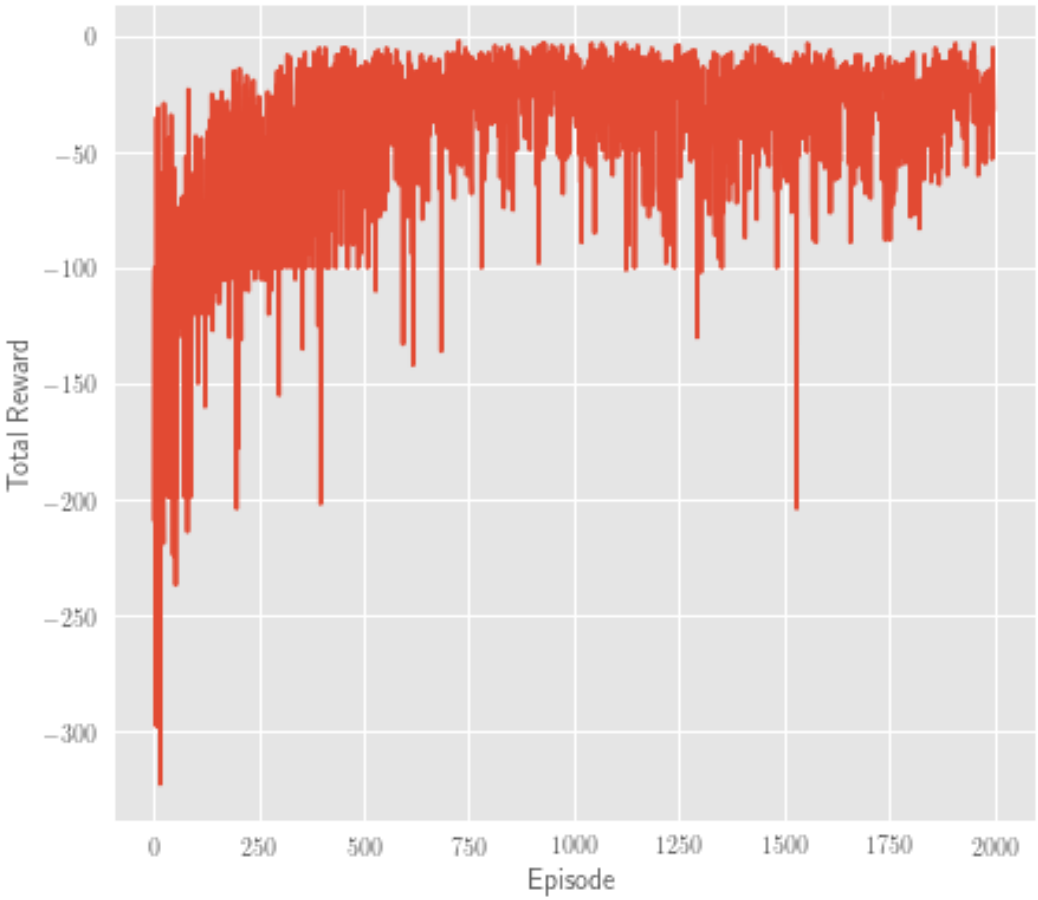

C-10-sarsa-softmax $\gamma = 0.9549$ $\beta = 1.254$
Rewards
Episodes
$\alpha = 0.09535$
$\alpha = 0.1$
$\alpha = 0.15$
$\alpha = 0.05$

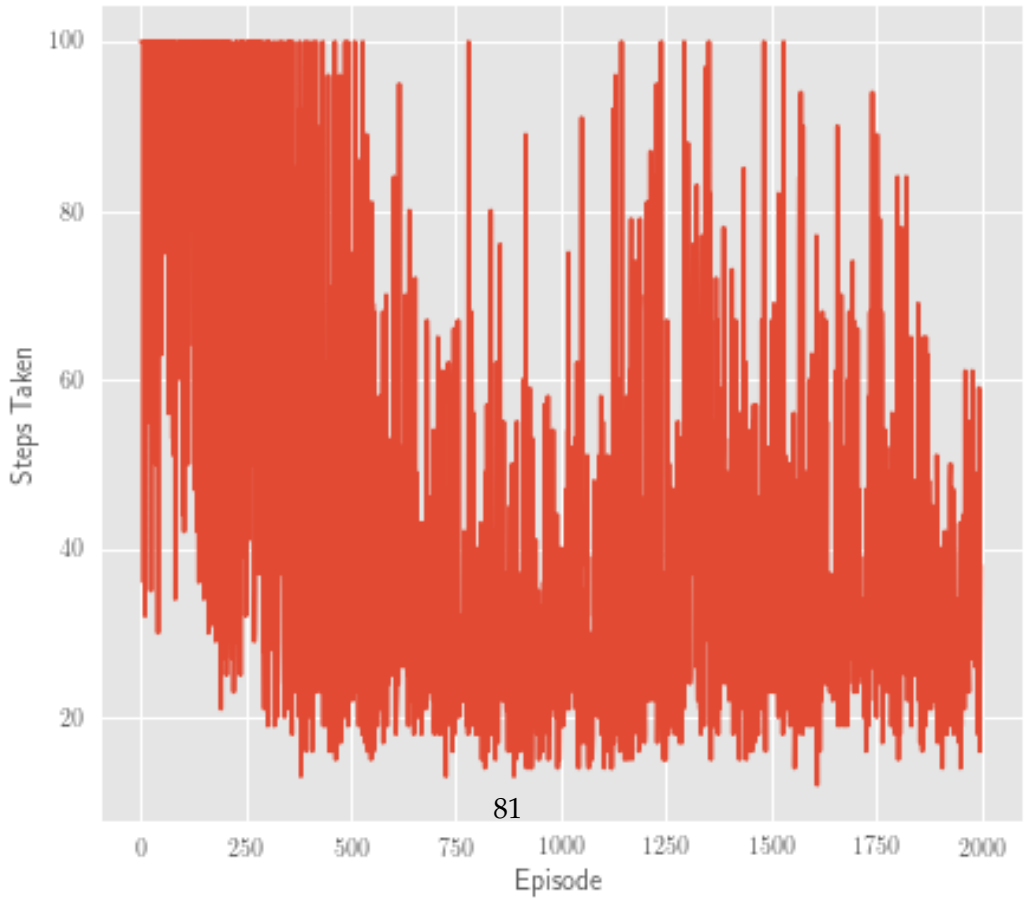


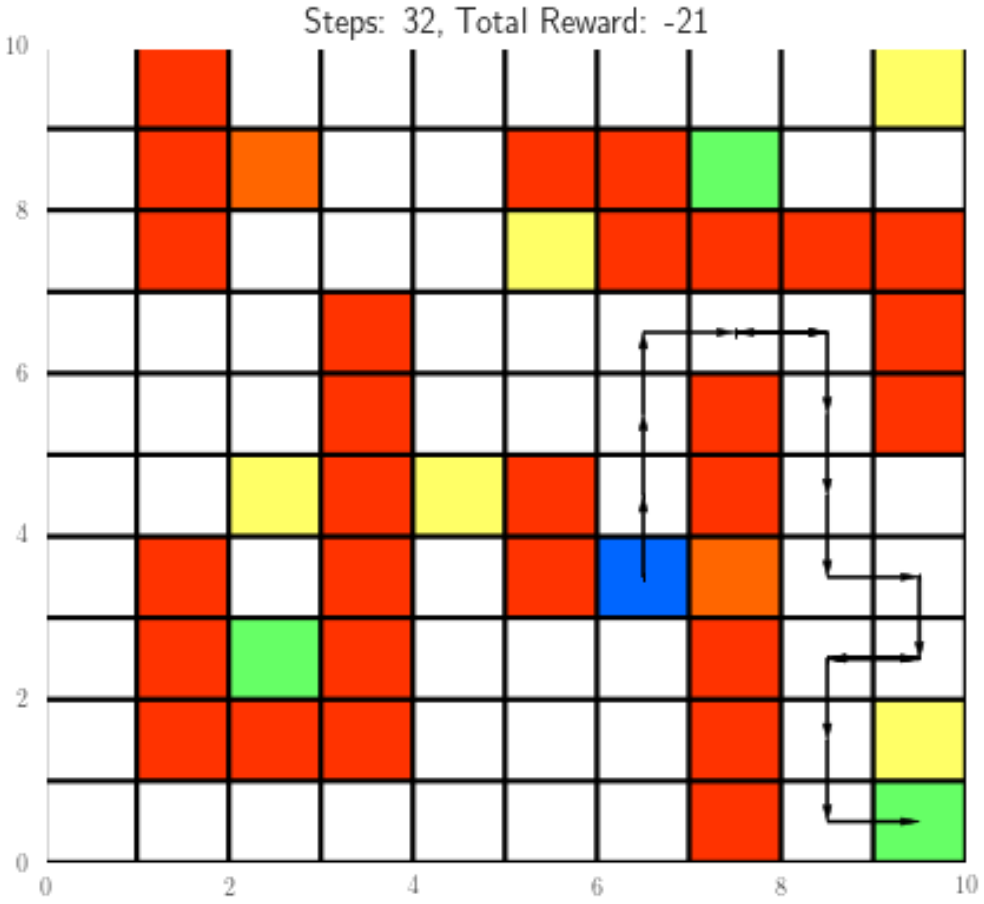
[ ]:
```
policy = 'softmax'
gamma = 0.99
```

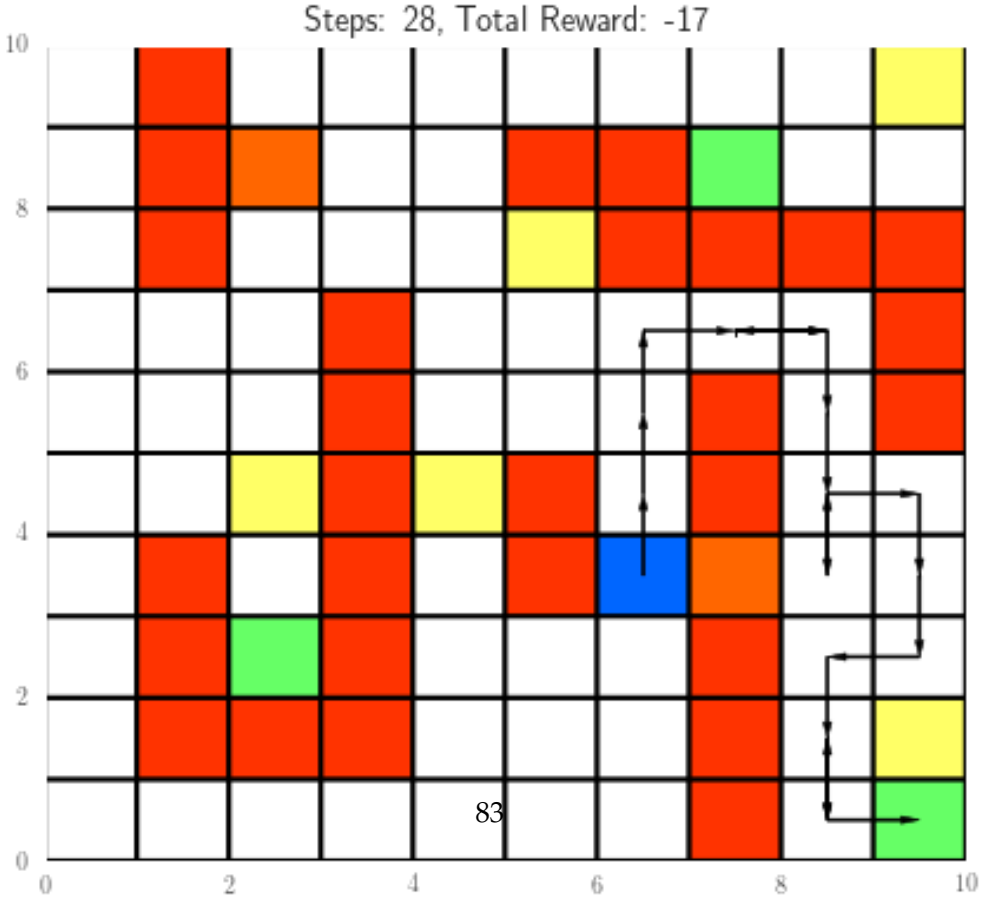
[ ]:
```
Q = best_plots(algorithm, policy, alpha,  epsilon, beta , gamma )
```

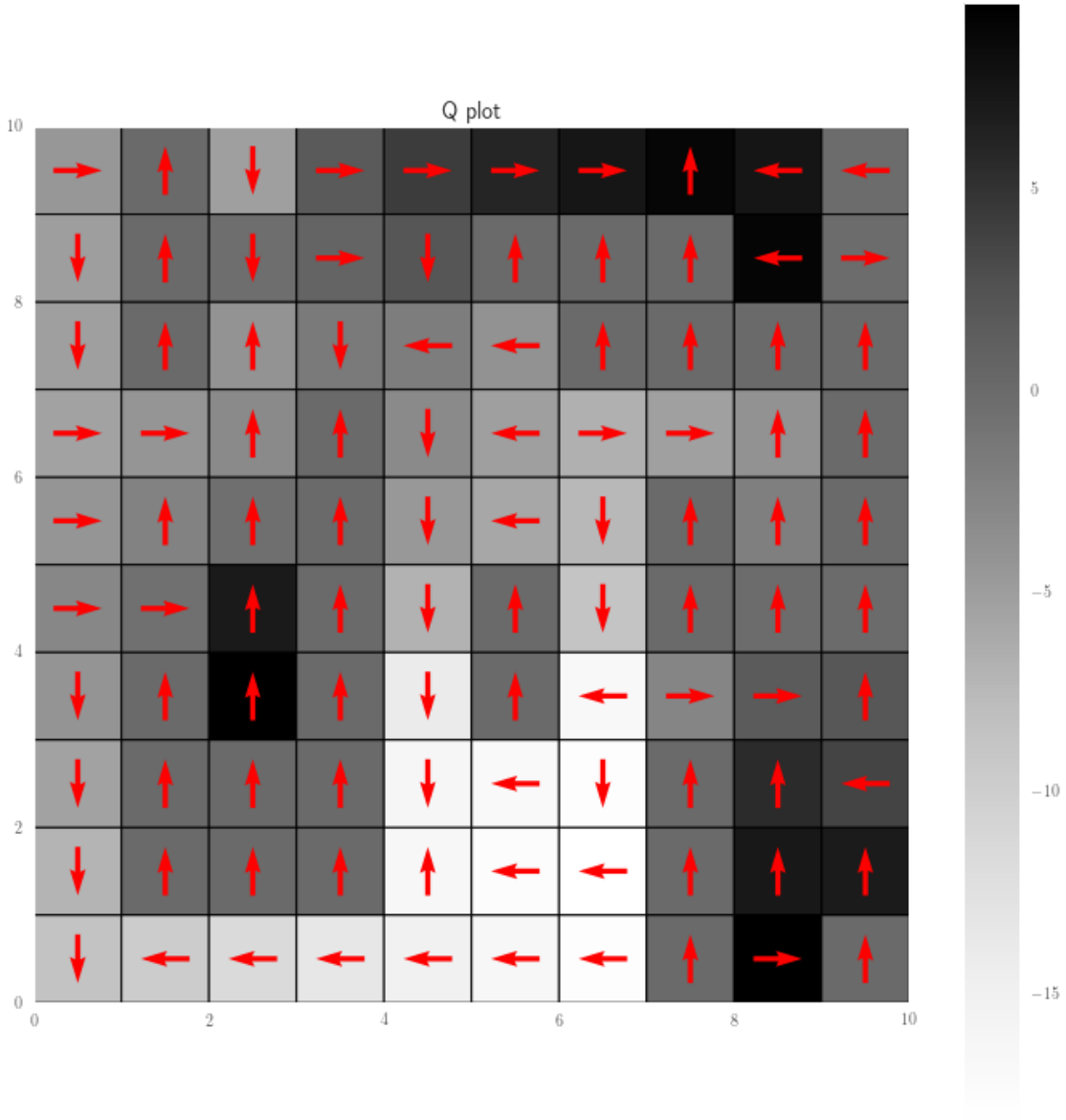
```
100%|| 2000/2000 [00:38<00:00, 51.46it/s]
```


Total Reward
0
−50
−100
−150
−200
−250
−300
0
250
500
750
1000
1250
1500
1750
2000
Episode


Steps Taken
100
80
60
40
20
0
250
500
750
1000
1250
1500
1750
2000
Episode


Steps: 32, Total Reward: -21
10
8
6
4
2
0
0
2
4
6
8
10


Steps: 28, Total Reward: -17
10
8
6
4
2
0
0
2
4
6
8
10



[ ]:

6.9 Config 4

[ ]:
```
NUM_CONFIG = 4

config_settings = configurations_1[NUM_CONFIG]

# create environment
```

```
env = create_env(**config_settings._asdict())

sweep_config = {
    "name" : f"{NUM_CONFIG}-config-sweep",
    "method": "random",
    "parameters": {
        "algorithm": {
            "values": ['sarsa', 'q_learning'],
          },
        "policy": {
            "values": ['softmax', 'epsilon_greedy'],
        },
        "epsilon": {
            "min": 0.0,
            "max": 1.0,
        },
        "alpha": {
            "min": 0.01,
            "max": 0.2,
        },
        "gamma": {
            "min": 0.5,
            "max": 1.0,
        },
        "beta": {
            "min": 0.5,
            "max": 1.5,
        }
    }
}
```

[ ]:
```
sweep_id = wandb.sweep(sweep_config, project='RLPA1')
```

```
Create sweep with ID: h84lgbi5
Sweep URL: https://wandb.ai/sathvikjoel/RLPA1/sweeps/h84lgbi5
```

[ ]:
```
wandb.agent(sweep_id, run, count=20)
```

6.9.1 Plotting

[ ]:
```
# swwep 19

alpha = 0.1732
l_alpha = [0.18, 0.15, 0.2]
beta = 0.8155
l_beta = [1, 0.7, 0.9]
gamma = 0.958
```

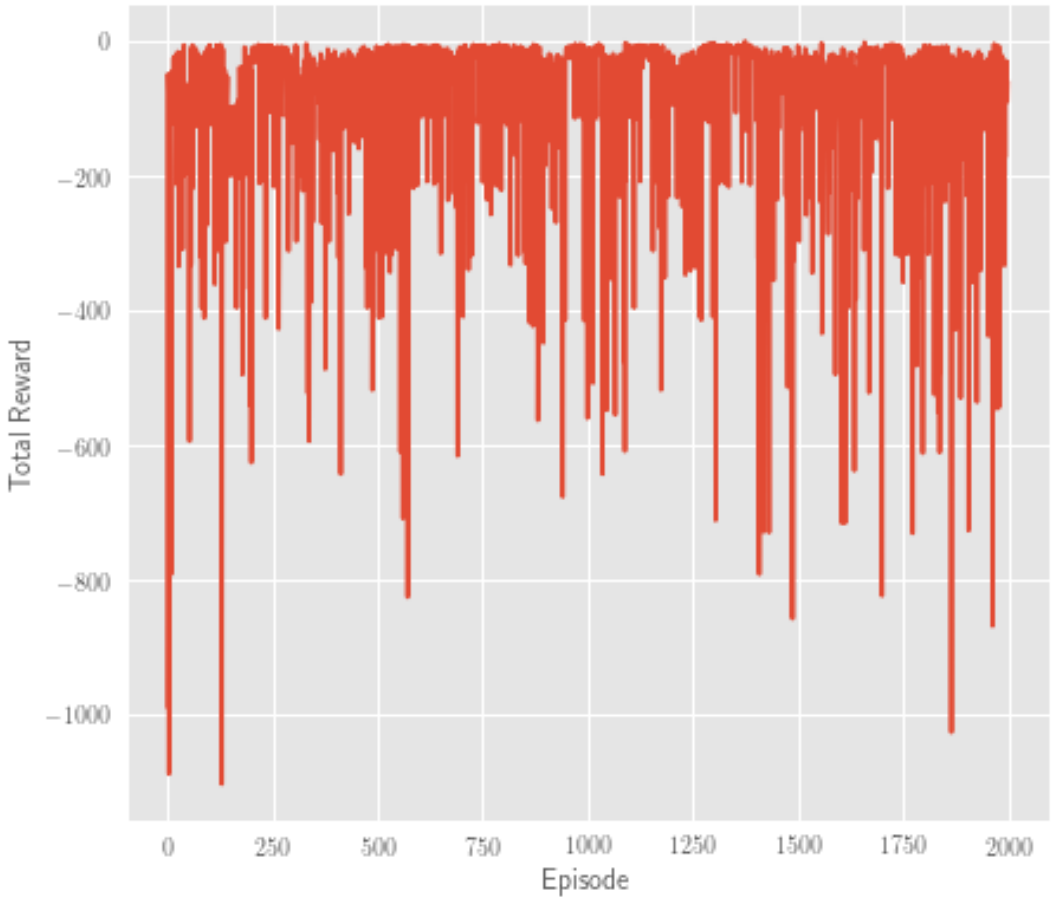
```
l_gamma = [0.85, 0.9, 1.0]
epsilon = 0.09116
l_epsilon = [0.1, 0.05, 0.2]
policy = 'e-greedy'
algorithm = 'sarsa'
```

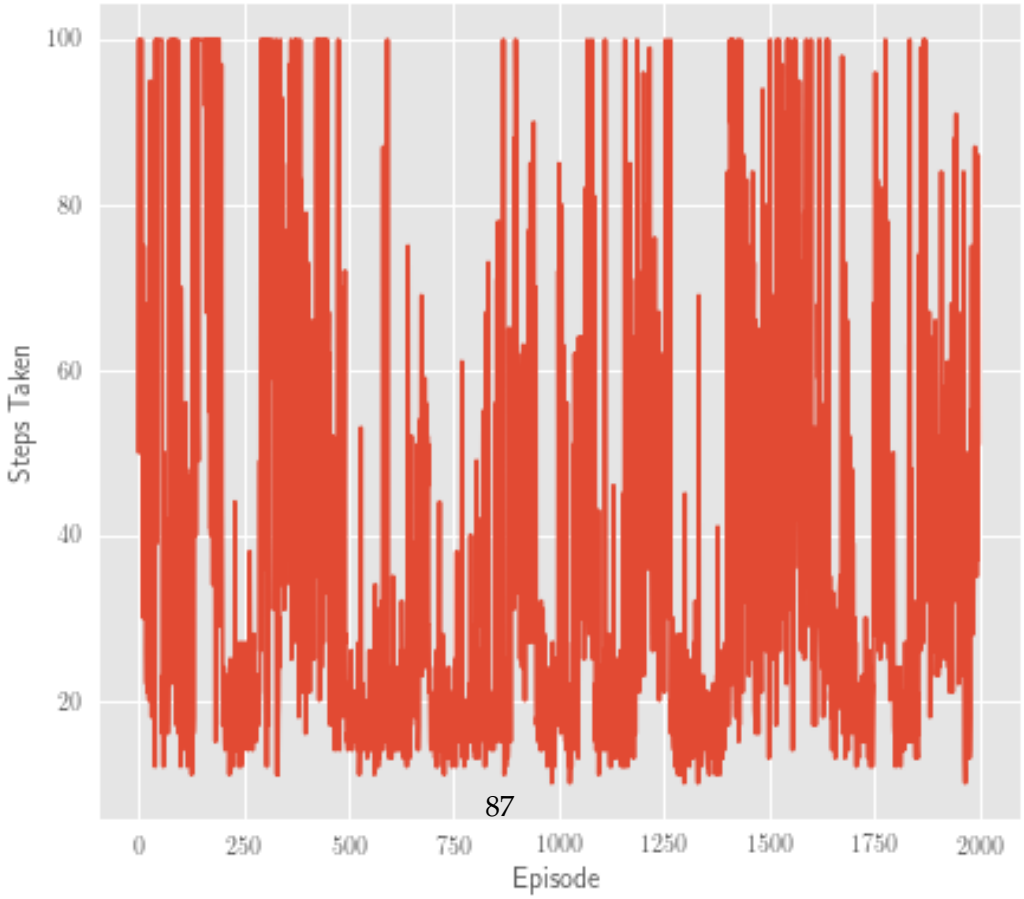
[ ]:
```
plt.style.use('ggplot')
```

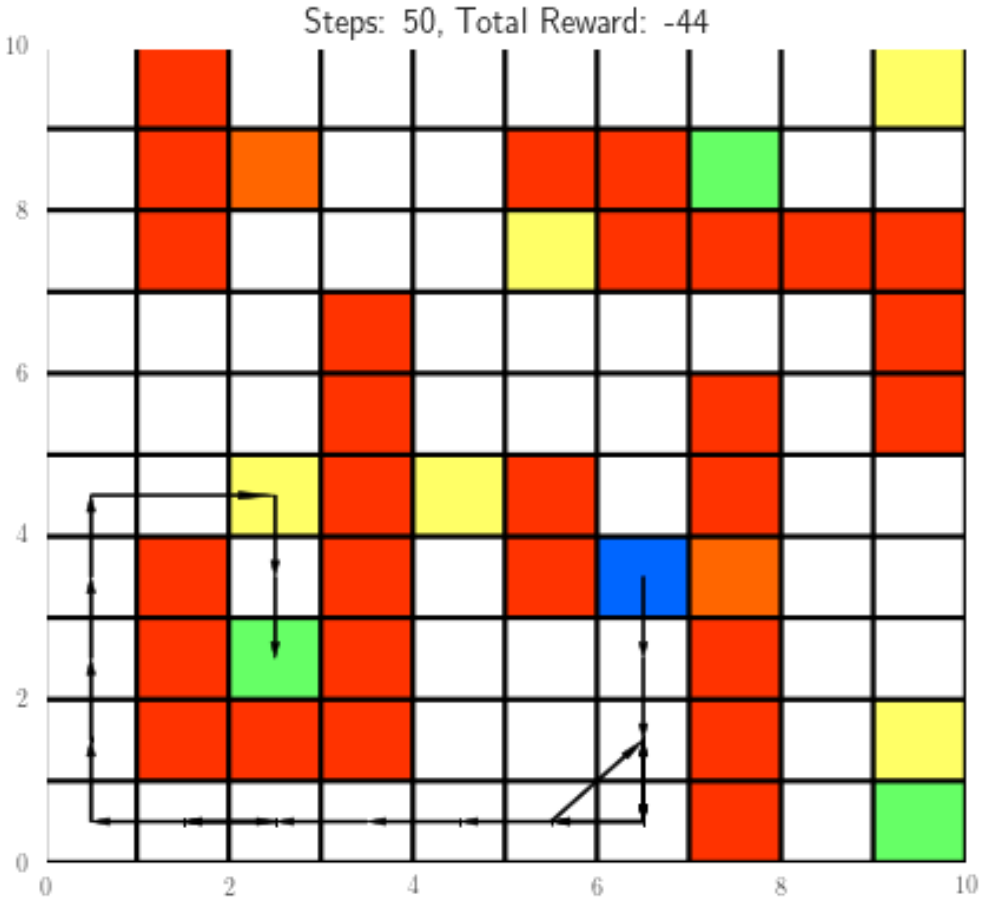
[ ]:
```
plot_all(alpha, l_alpha, beta, l_beta, gamma, l_gamma, epsilon, l_epsilon)
```

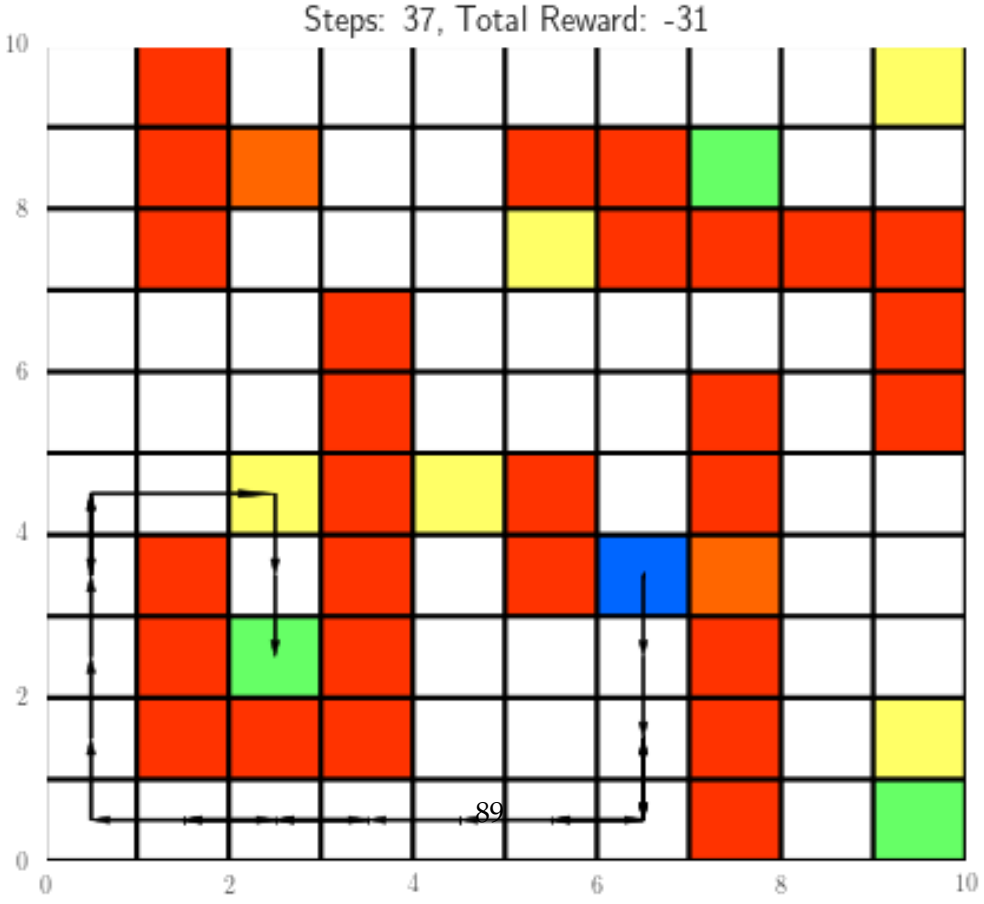
[ ]:
```
Q = best_plots(algorithm, policy, alpha,  epsilon, beta , gamma )
```

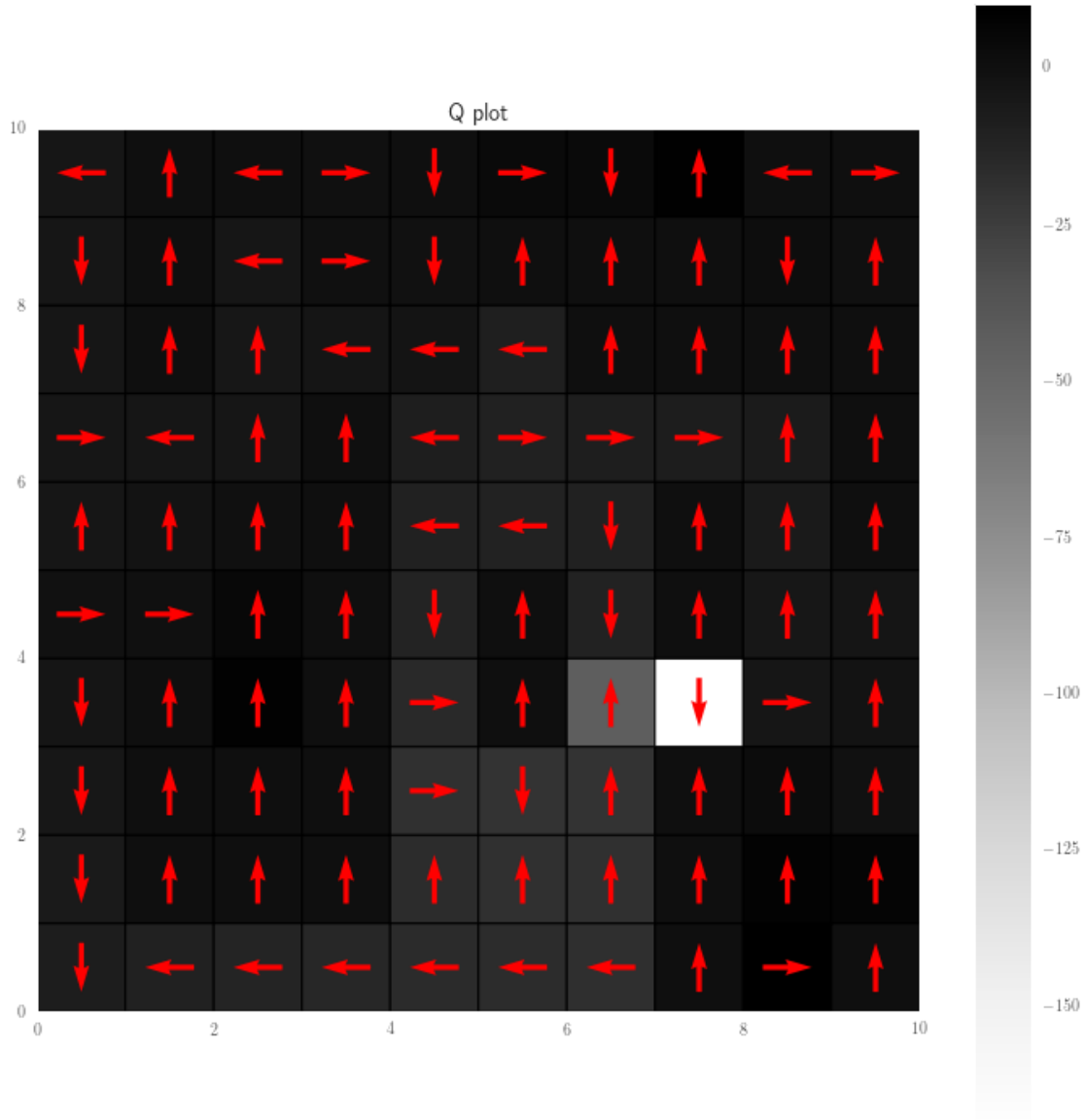
```
100%|| 2000/2000 [00:13<00:00, 149.32it/s]
```


0
−200
−400
−600
−800
−1000
Total Reward
0
250
500
750
1000
1250
1500
1750
2000
Episode


100
80
60
40
20
Steps Taken
0
250
500
750
1000
1250
1500
1750
2000
Episode


Steps: 50, Total Reward: -44
10
8
6
4
2
0
0
2
4
6
8
10


Steps: 37, Total Reward: -31
10
8
6
4
2
0
0
2
4
6
8
10



6.10 Config 5

[ ]:
```
NUM_CONFIG = 5

config_settings = configurations_l[NUM_CONFIG]

# create environment
env = create_env(**config_settings._asdict())
```

```
sweep_config = {
    "name" : f"{NUM_CONFIG}-config-sweep",
    "method": "random",
    "parameters": {
        "algorithm": {
            "values": ['sarsa', 'q_learning'],
        },
        "policy": {
            "values": ['softmax', 'epsilon_greedy'],
        },
        "epsilon": {
            "min": 0.0,
            "max": 1.0,
        },
        "alpha": {
            "min": 0.01,
            "max": 0.2,
        },
        "gamma": {
            "min": 0.5,
            "max": 1.0,
        },
        "beta": {
            "min": 0.5,
            "max": 1.5,
        }
    }
}
```

[ ]:
```
sweep_id = wandb.sweep(sweep_config, project='RLPA1')
```

[ ]:
```
wandb.agent(sweep_id, run, count=20)
```

[ ]:
```
```

6.10.1 Plotting

[ ]:
```
# swwep 19

alpha = 0.02983
l_alpha = [0.1, 0.07, 0.01]
beta = 1.478
l_beta = [1, 1.2, 1.3]
gamma = 0.9685
l_gamma = [0.85, 0.9, 1.0]
epsilon = 0.7722
l_epsilon = [0.5, 0.3, 0.8]
```

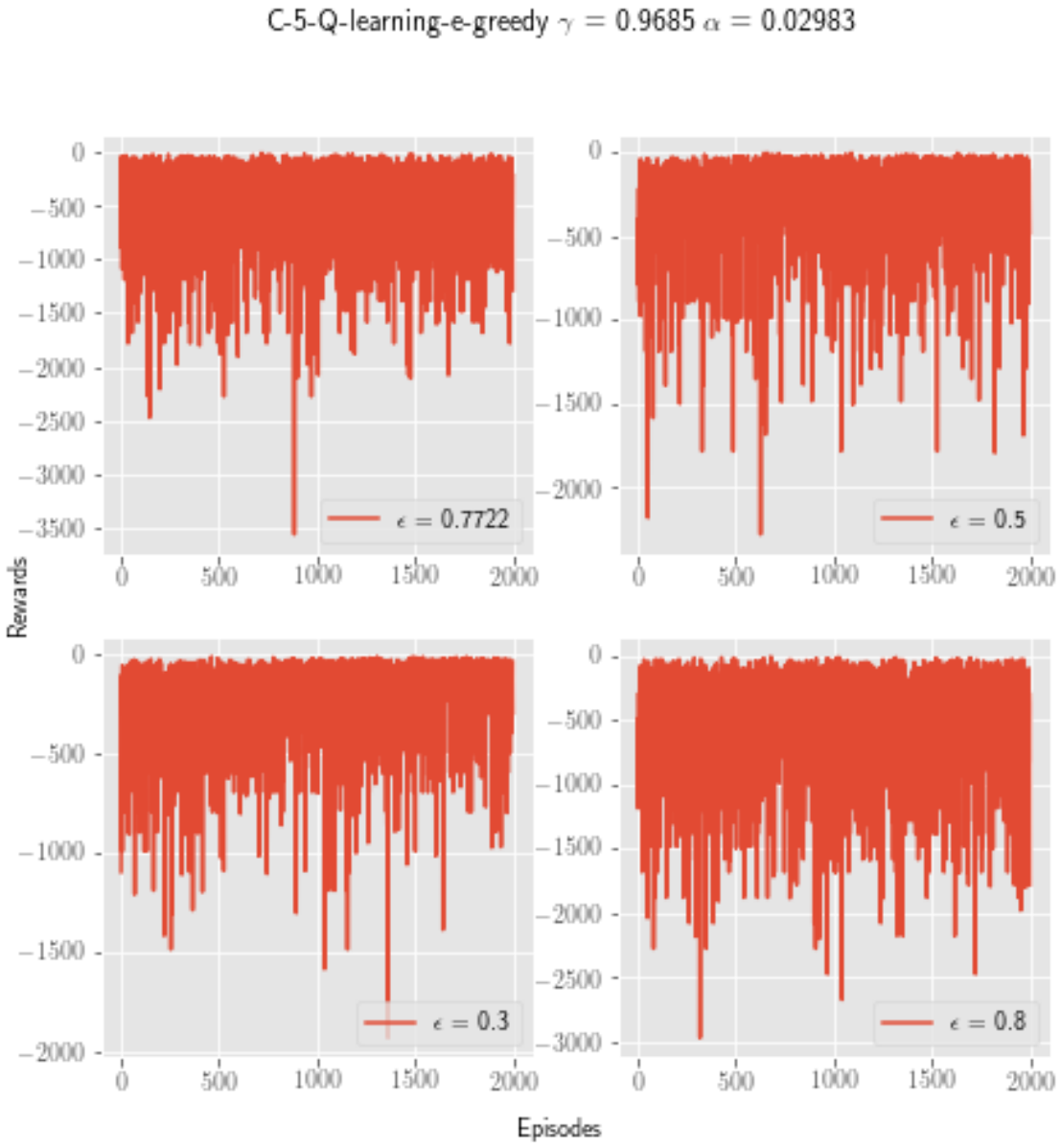
```
policy = 'softmax'
algorithm = 'q-learning'
```

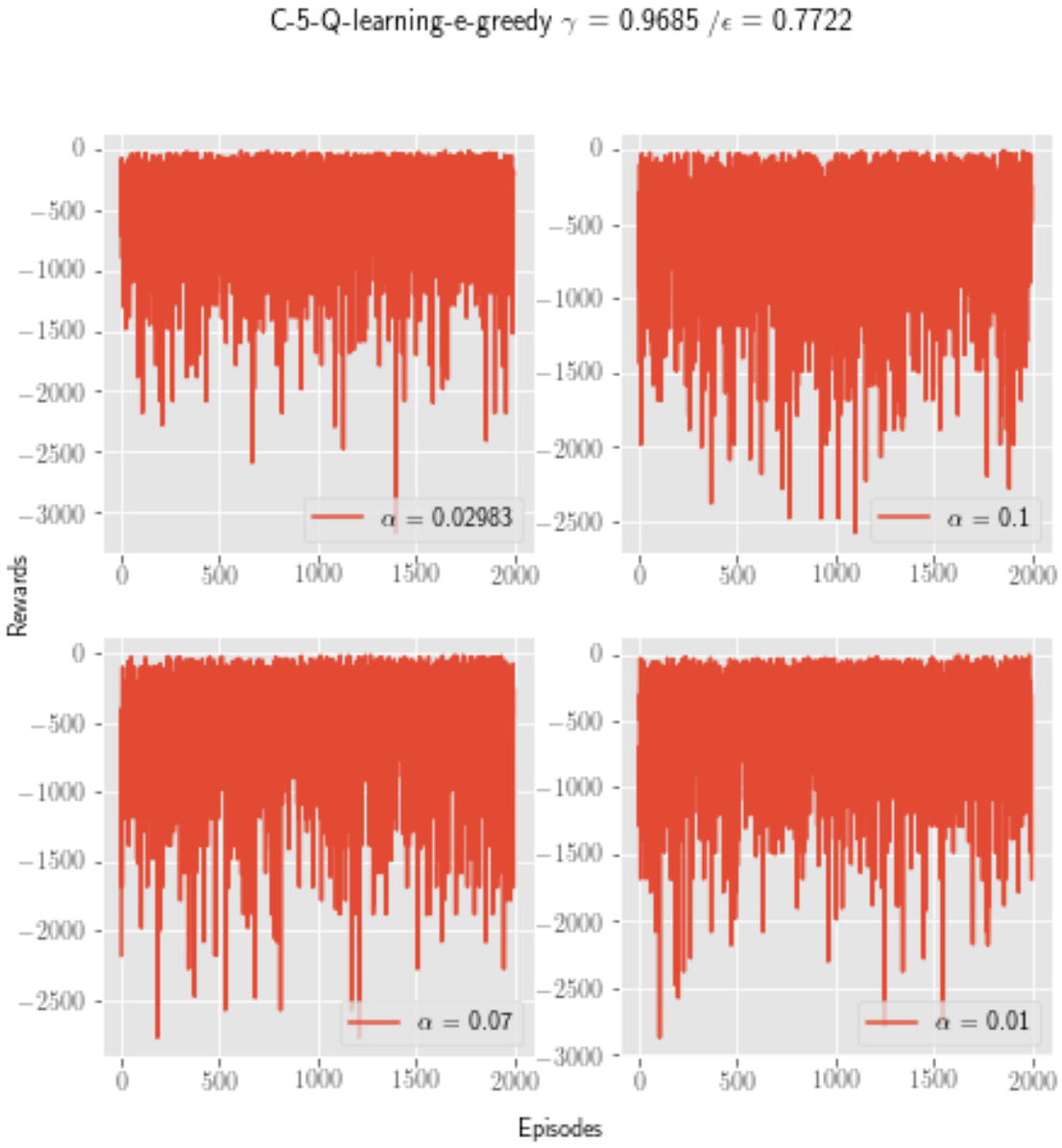
[ ]:
```
plt.style.use('ggplot')
```

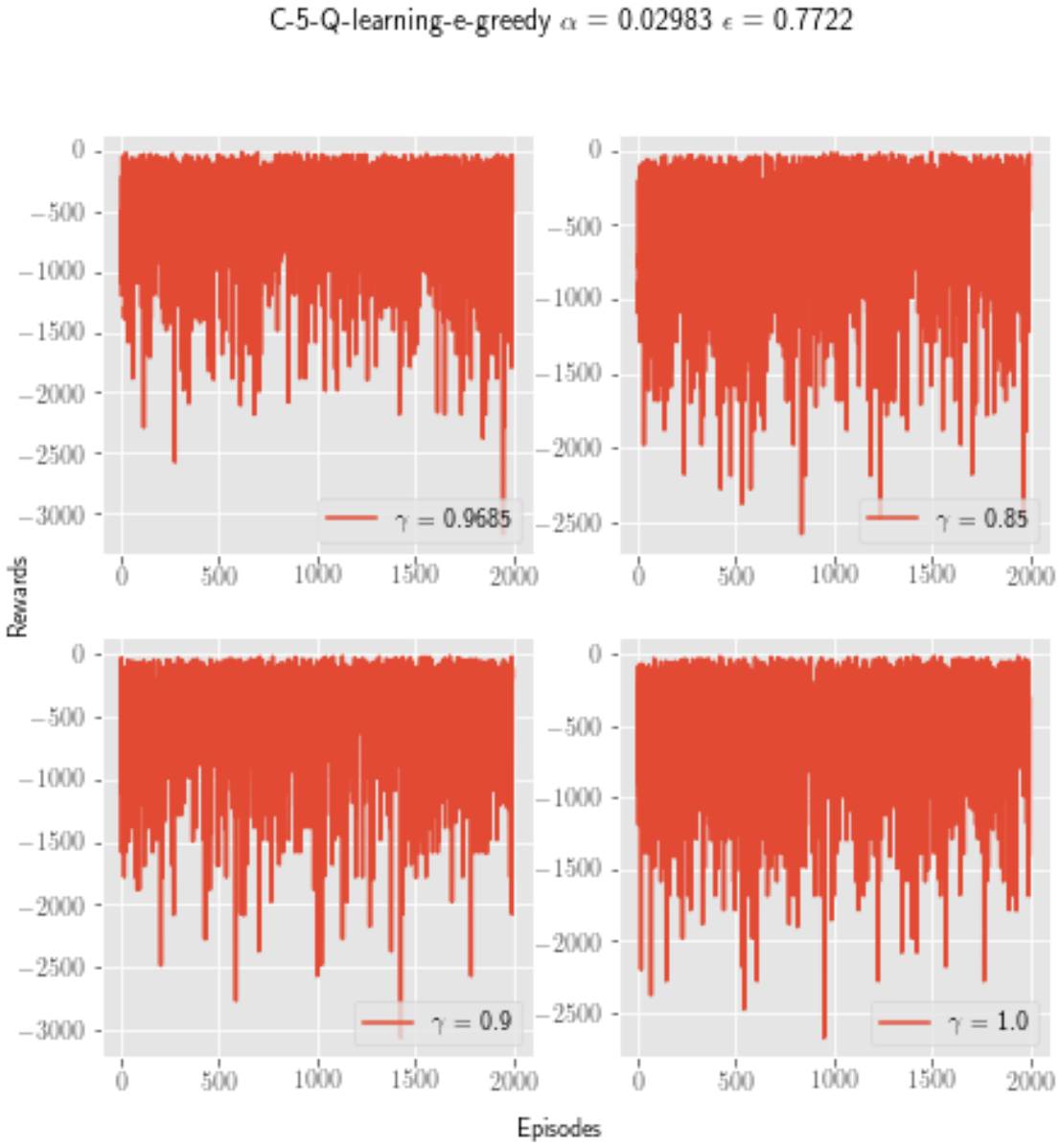
[ ]:
```
plot_all(alpha, l_alpha, beta, l_beta, gamma, l_gamma, epsilon, l_epsilon)
```

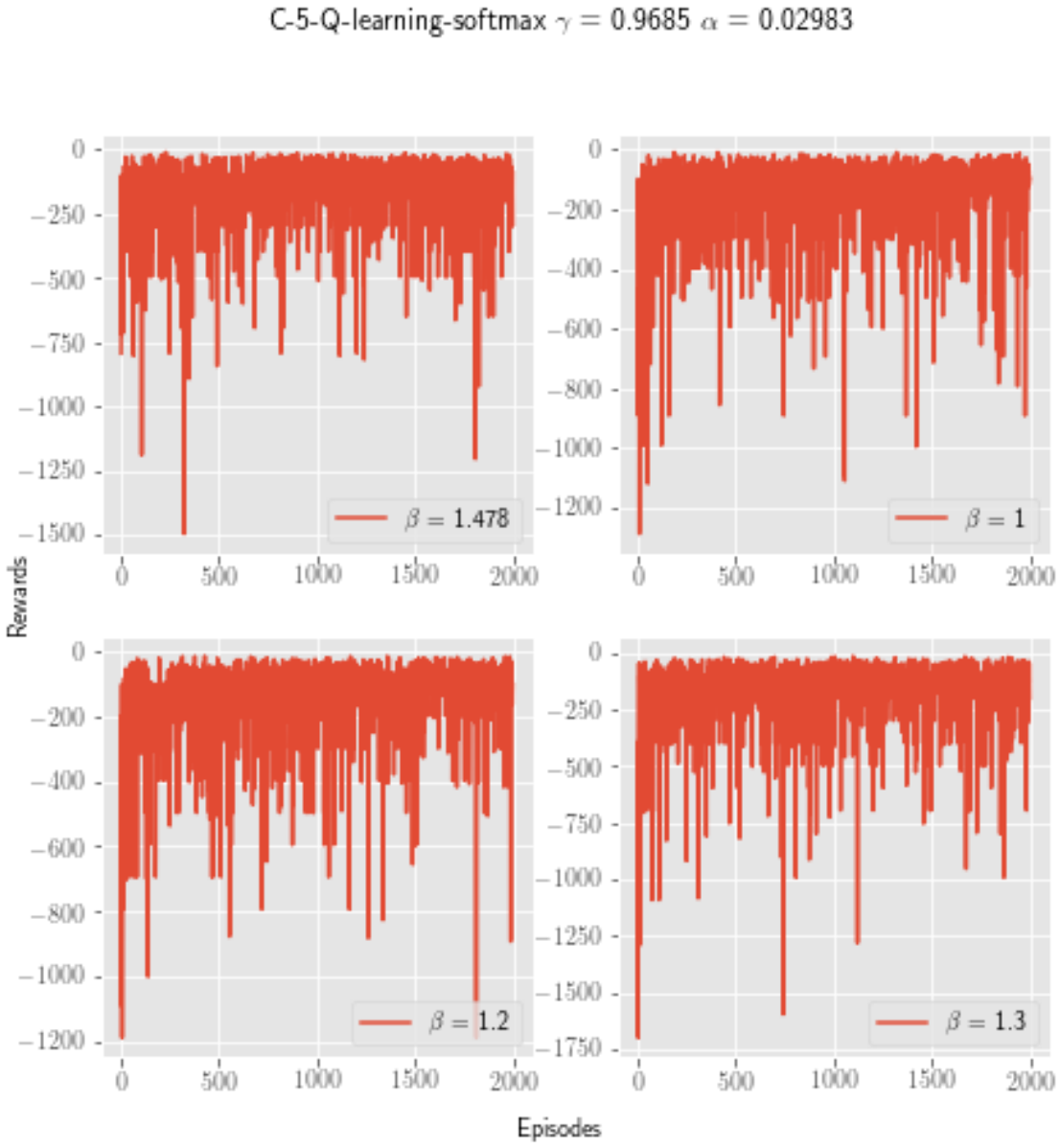
```
100%|| 2000/2000 [00:44<00:00, 44.51it/s]
100%|| 2000/2000 [00:37<00:00, 53.06it/s]
100%|| 2000/2000 [00:34<00:00, 57.84it/s]
100%|| 2000/2000 [00:44<00:00, 44.90it/s]
100%|| 2000/2000 [00:43<00:00, 45.52it/s]
100%|| 2000/2000 [00:43<00:00, 46.39it/s]
100%|| 2000/2000 [00:43<00:00, 46.41it/s]
100%|| 2000/2000 [00:45<00:00, 44.01it/s]
100%|| 2000/2000 [00:44<00:00, 44.85it/s]
100%|| 2000/2000 [00:45<00:00, 43.81it/s]
100%|| 2000/2000 [00:45<00:00, 43.76it/s]
100%|| 2000/2000 [00:45<00:00, 44.17it/s]
100%|| 2000/2000 [01:14<00:00, 27.00it/s]
100%|| 2000/2000 [01:13<00:00, 27.06it/s]
100%|| 2000/2000 [01:15<00:00, 26.57it/s]
100%|| 2000/2000 [01:14<00:00, 26.72it/s]
100%|| 2000/2000 [01:13<00:00, 27.35it/s]
100%|| 2000/2000 [01:11<00:00, 28.05it/s]
100%|| 2000/2000 [01:16<00:00, 26.26it/s]
100%|| 2000/2000 [01:28<00:00, 22.70it/s]
100%|| 2000/2000 [01:21<00:00, 24.47it/s]
100%|| 2000/2000 [01:18<00:00, 25.43it/s]
100%|| 2000/2000 [01:18<00:00, 25.54it/s]
100%|| 2000/2000 [01:16<00:00, 26.03it/s]
100%|| 2000/2000 [00:47<00:00, 42.15it/s]
100%|| 2000/2000 [00:38<00:00, 52.53it/s]
100%|| 2000/2000 [00:34<00:00, 58.79it/s]
100%|| 2000/2000 [00:45<00:00, 44.15it/s]
100%|| 2000/2000 [00:44<00:00, 44.55it/s]
100%|| 2000/2000 [00:45<00:00, 43.61it/s]
100%|| 2000/2000 [00:45<00:00, 44.21it/s]
100%|| 2000/2000 [00:44<00:00, 45.19it/s]
100%|| 2000/2000 [00:44<00:00, 44.86it/s]
100%|| 2000/2000 [00:46<00:00, 43.15it/s]
100%|| 2000/2000 [00:46<00:00, 43.44it/s]
100%|| 2000/2000 [00:44<00:00, 45.24it/s]
100%|| 2000/2000 [01:09<00:00, 28.71it/s]
100%|| 2000/2000 [01:07<00:00, 29.42it/s]
100%|| 2000/2000 [01:07<00:00, 29.44it/s]
100%|| 2000/2000 [01:09<00:00, 28.59it/s]
100%|| 2000/2000 [01:08<00:00, 29.19it/s]
```

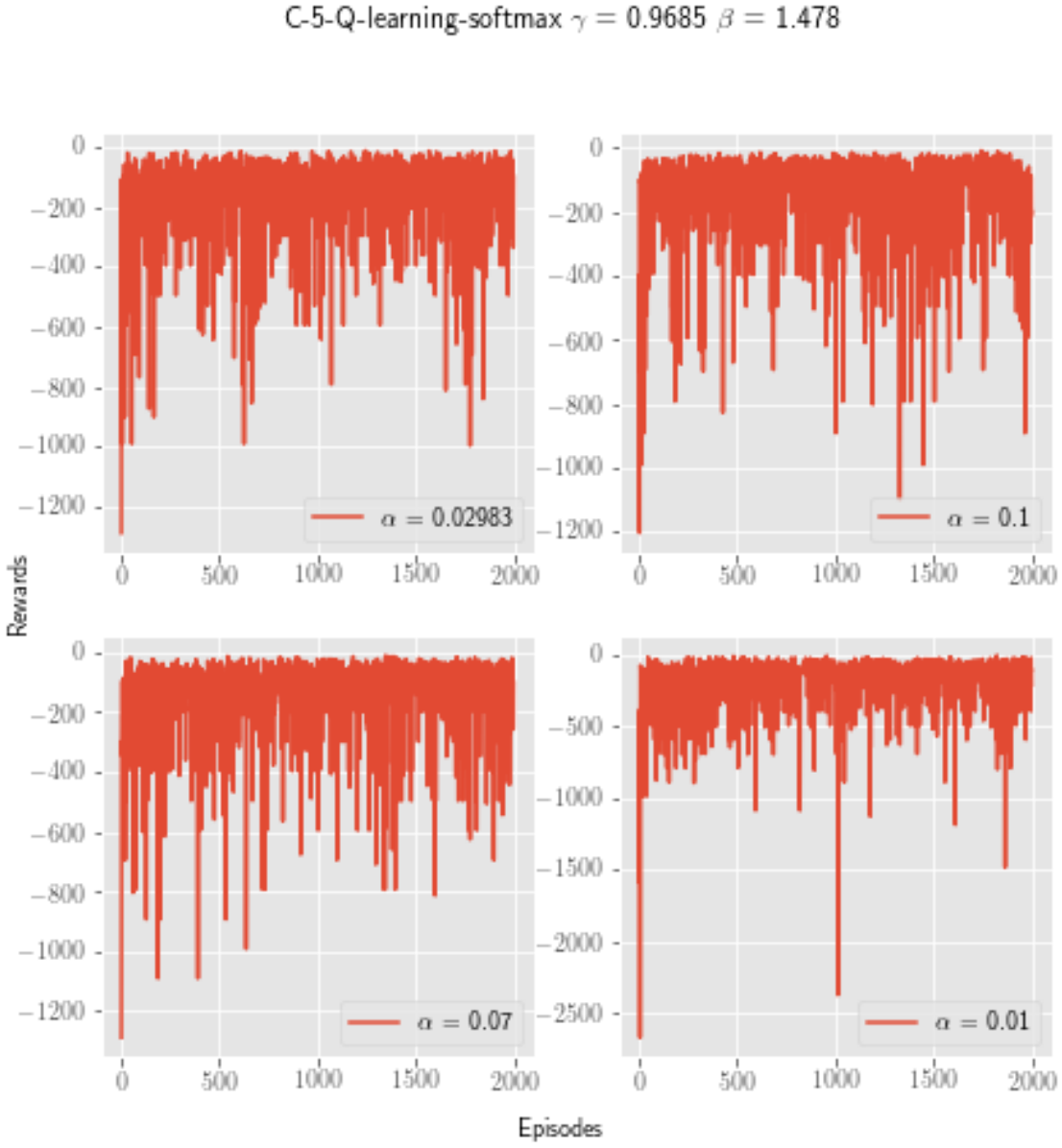
```
100%|| 2000/2000 [01:04<00:00, 31.01it/s]
100%|| 2000/2000 [01:06<00:00, 30.22it/s]
100%|| 2000/2000 [01:13<00:00, 27.04it/s]
100%|| 2000/2000 [01:10<00:00, 28.54it/s]
100%|| 2000/2000 [01:13<00:00, 27.39it/s]
100%|| 2000/2000 [01:12<00:00, 27.64it/s]
100%|| 2000/2000 [01:08<00:00, 29.39it/s]
```

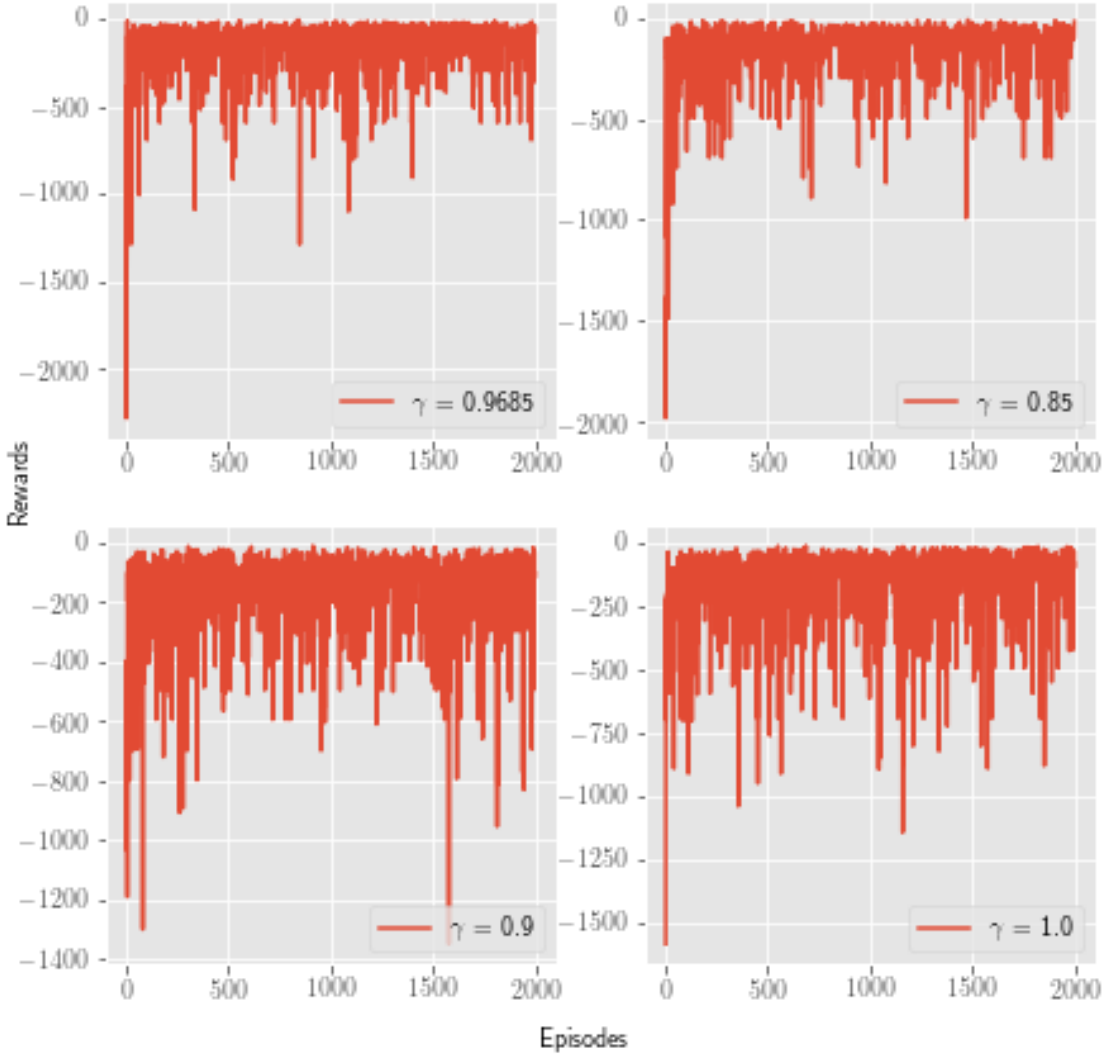


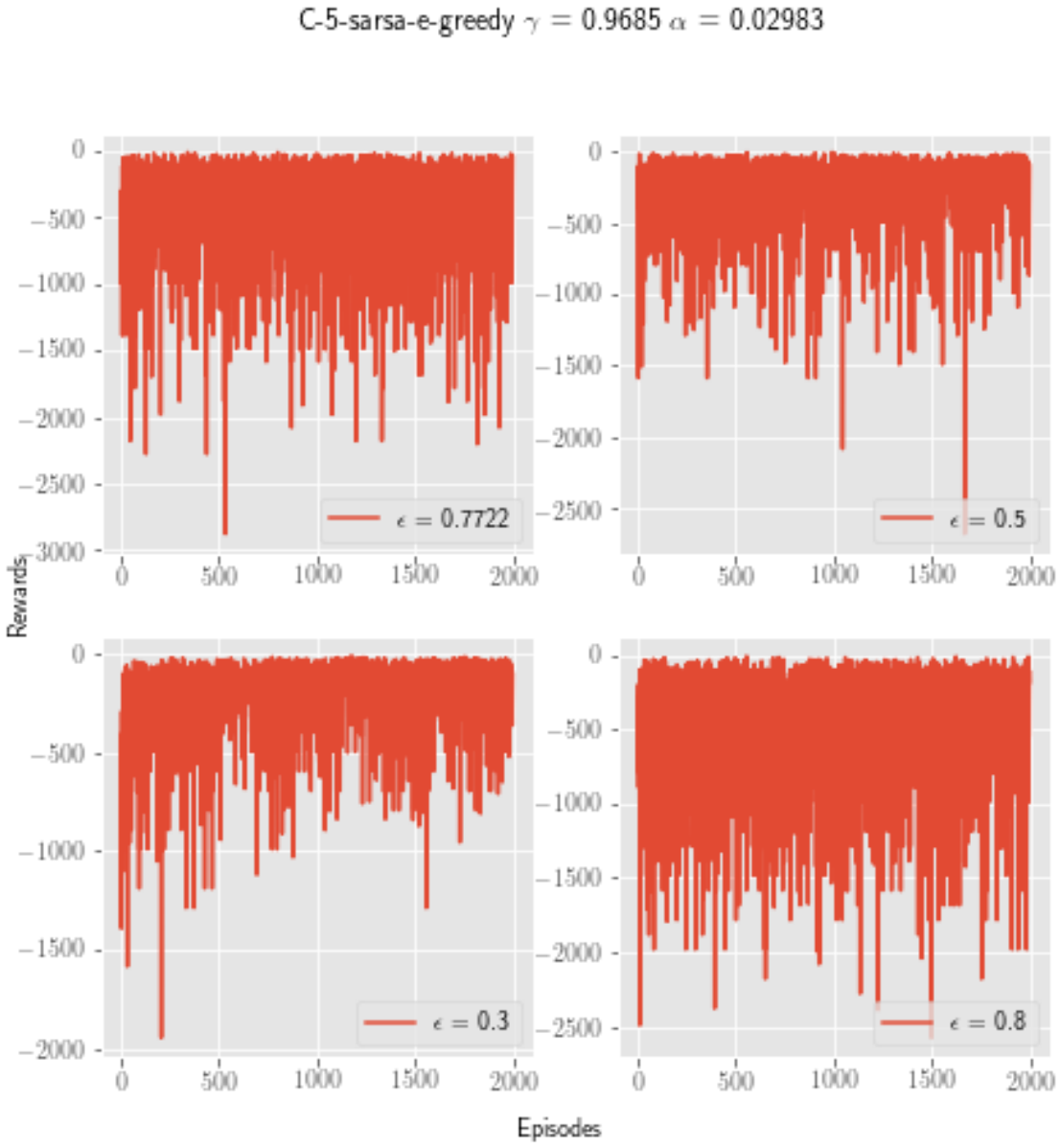

C-5-Q-learning-e-greedy $\gamma = 0.9685$ $/\epsilon = 0.7722$
Rewards
Episodes
$\alpha = 0.02983$
$\alpha = 0.1$
$\alpha = 0.07$
$\alpha = 0.01$

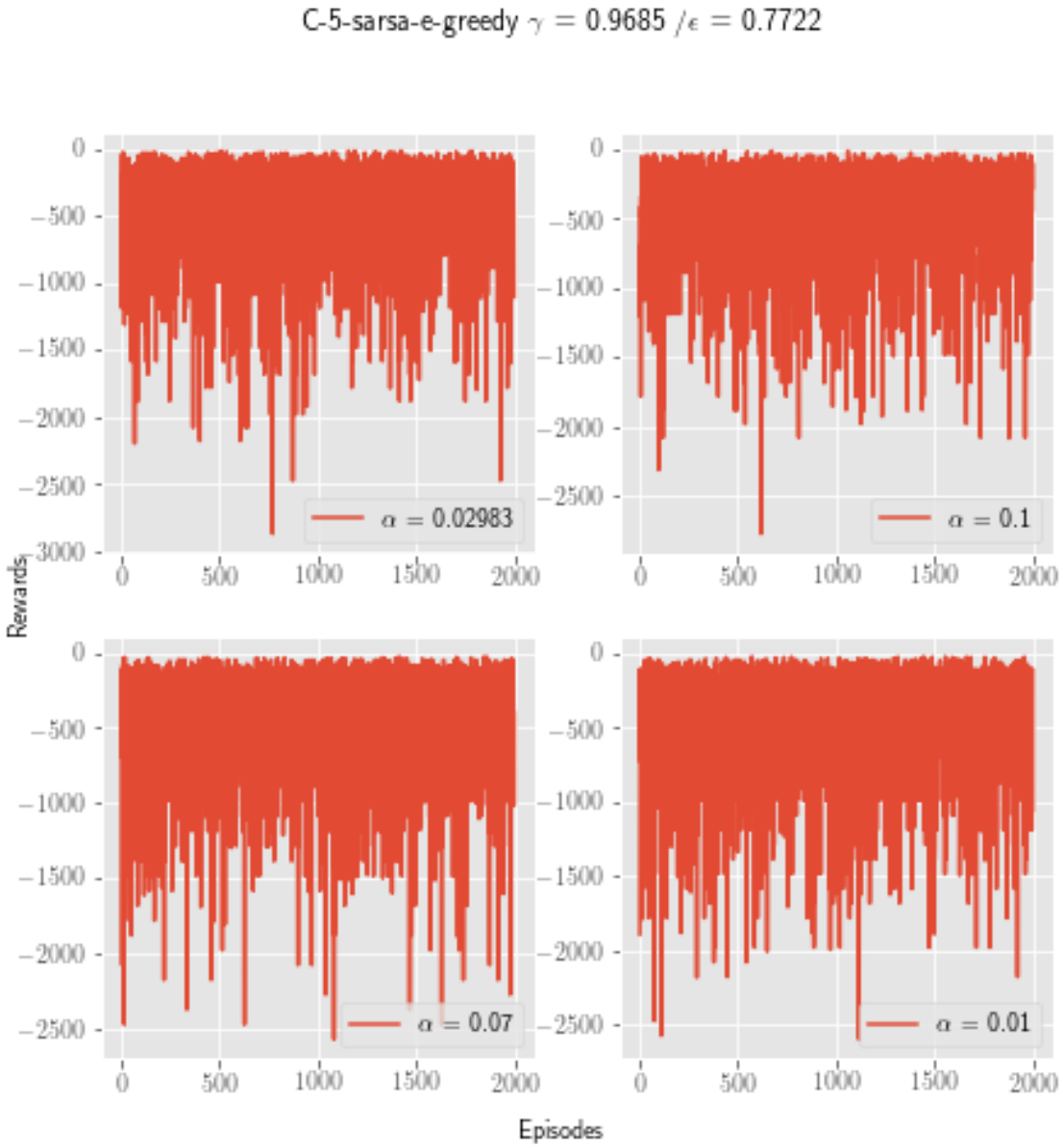

C-5-Q-learning-e-greedy $\alpha = 0.02983$ $\epsilon = 0.7722$
Rewards
Episodes
$\gamma = 0.9685$
$\gamma = 0.85$
$\gamma = 0.9$
$\gamma = 1.0$

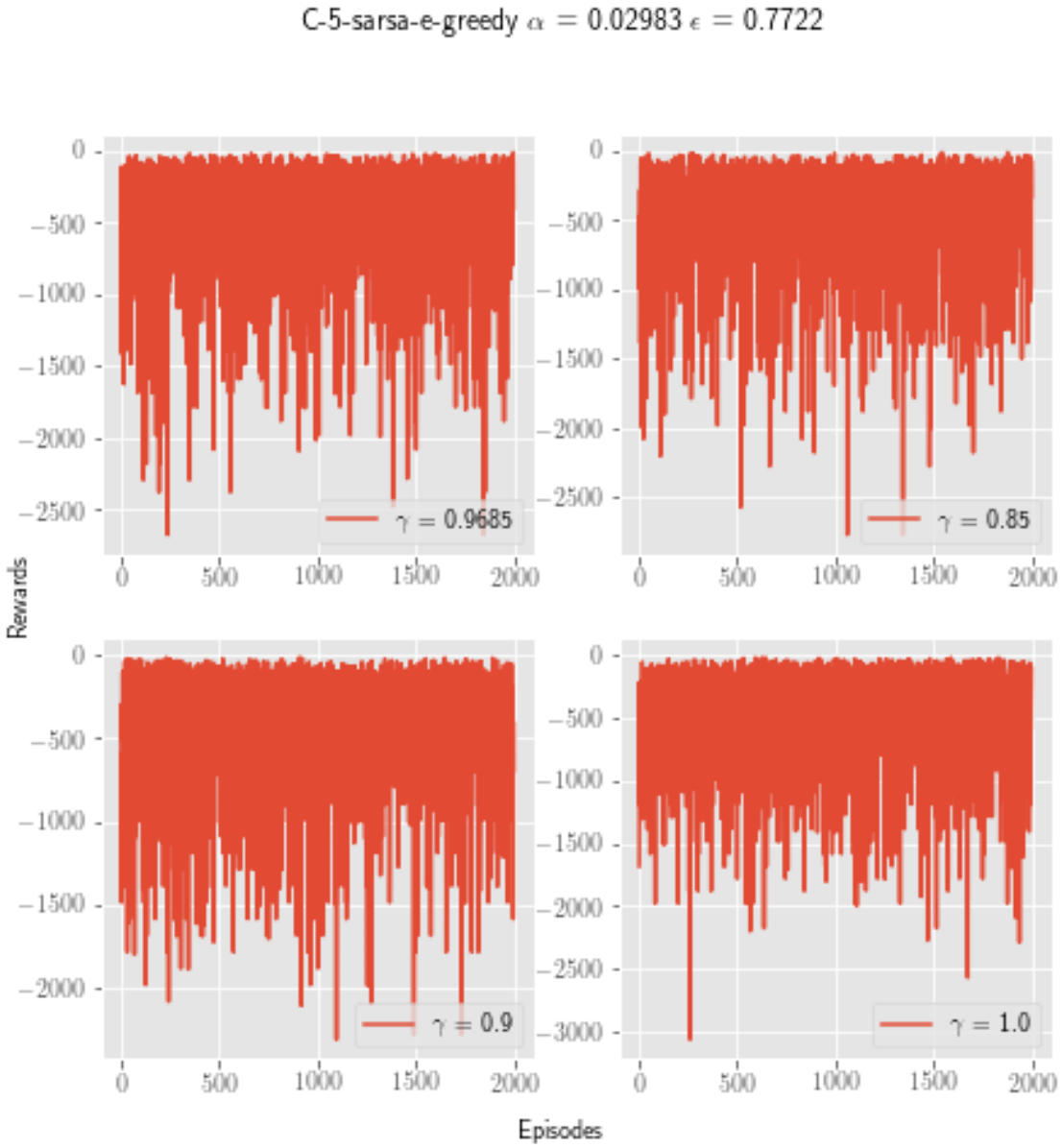

C-5-Q-learning-softmax $\gamma = 0.9685$ $\alpha = 0.02983$
Rewards
Episodes
$\beta = 1.478$
$\beta = 1$
$\beta = 1.2$
$\beta = 1.3$

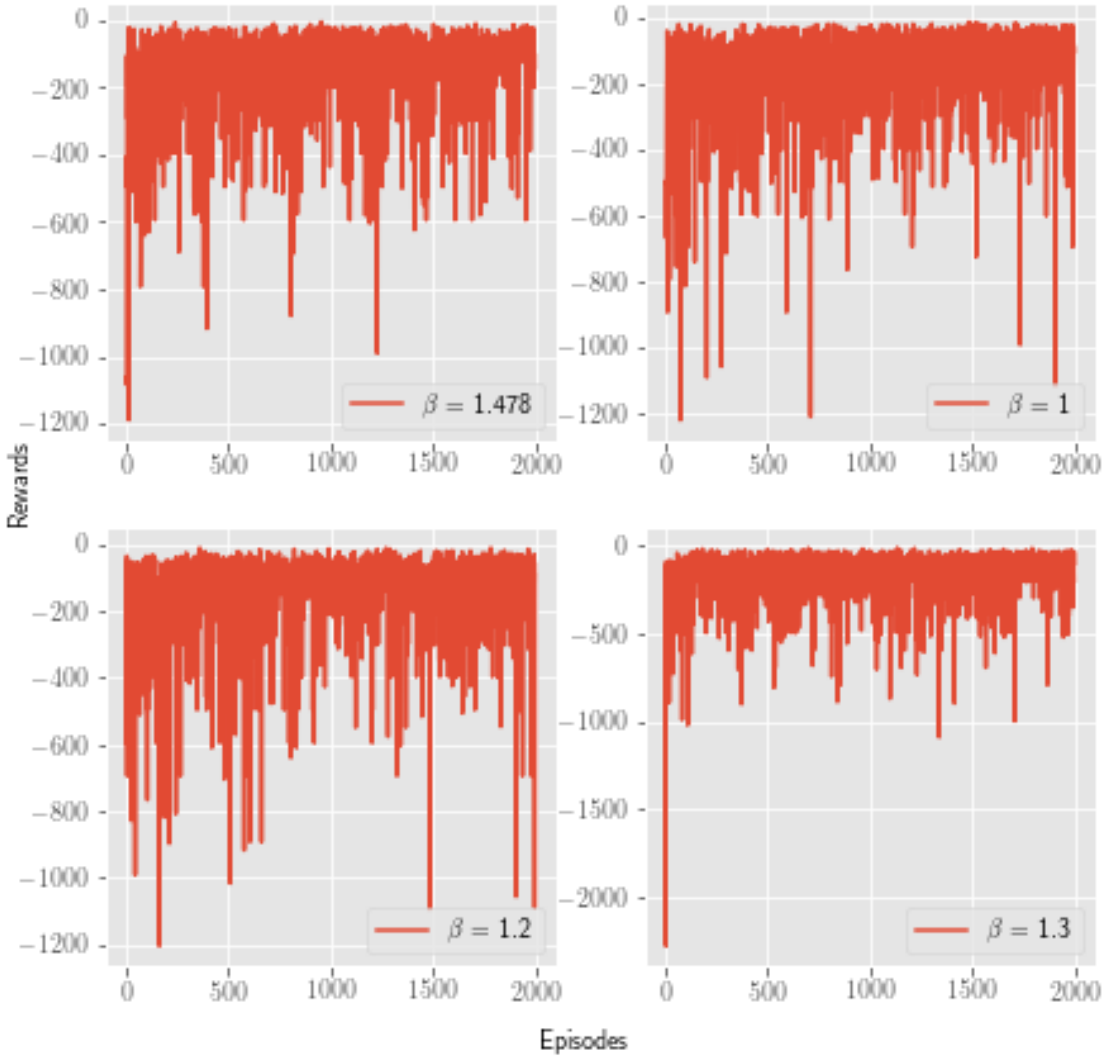

C-5-Q-learning-softmax $\gamma = 0.9685$ $\beta = 1.478$
Rewards
Episodes
$\alpha = 0.02983$
$\alpha = 0.1$
$\alpha = 0.07$
$\alpha = 0.01$

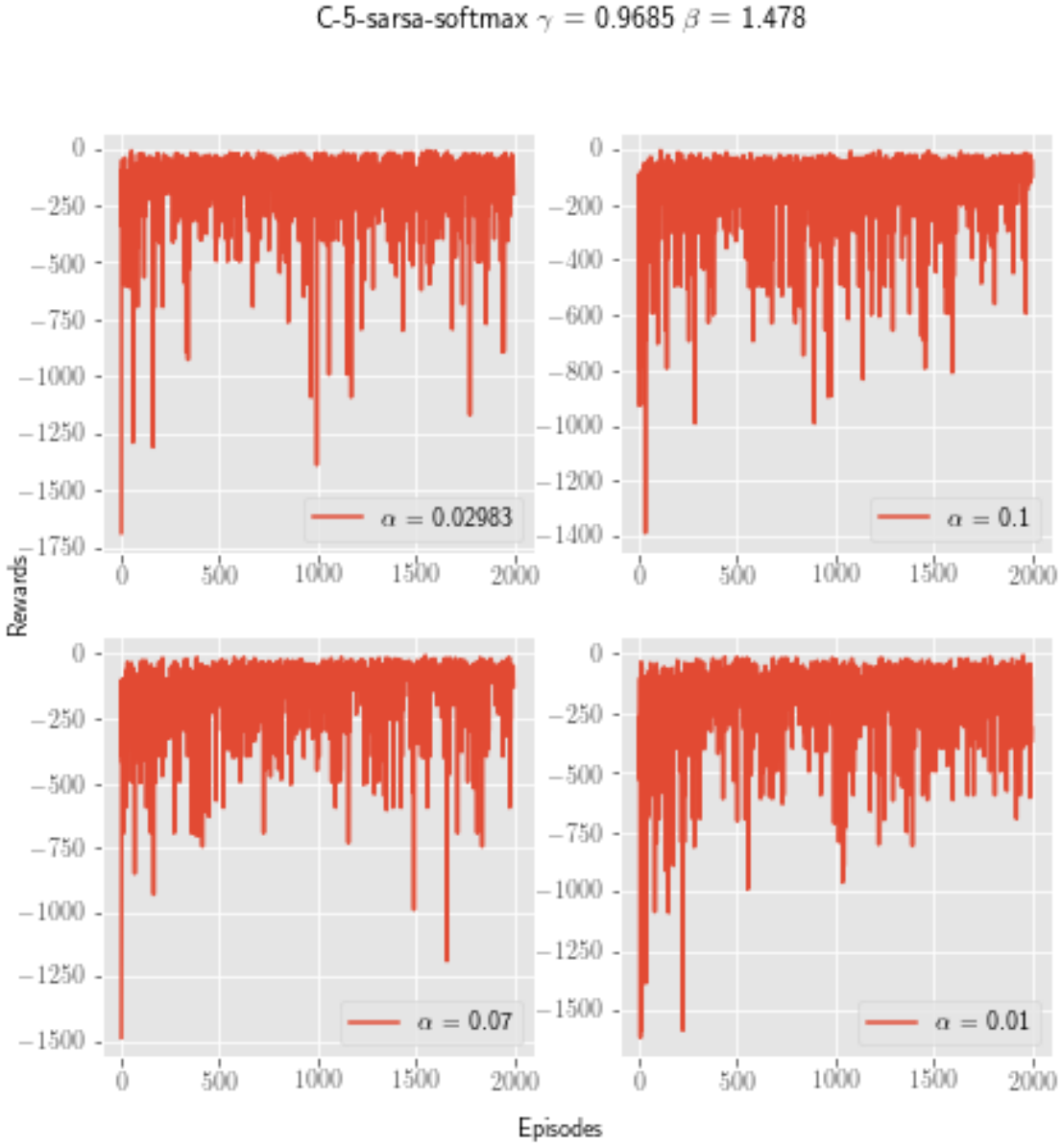

C-5-Q-learning-softmax $\alpha = 0.02983$ $\beta = 1.478$
Rewards
Episodes
$\gamma = 0.9685$
$\gamma = 0.85$
$\gamma = 0.9$
$\gamma = 1.0$

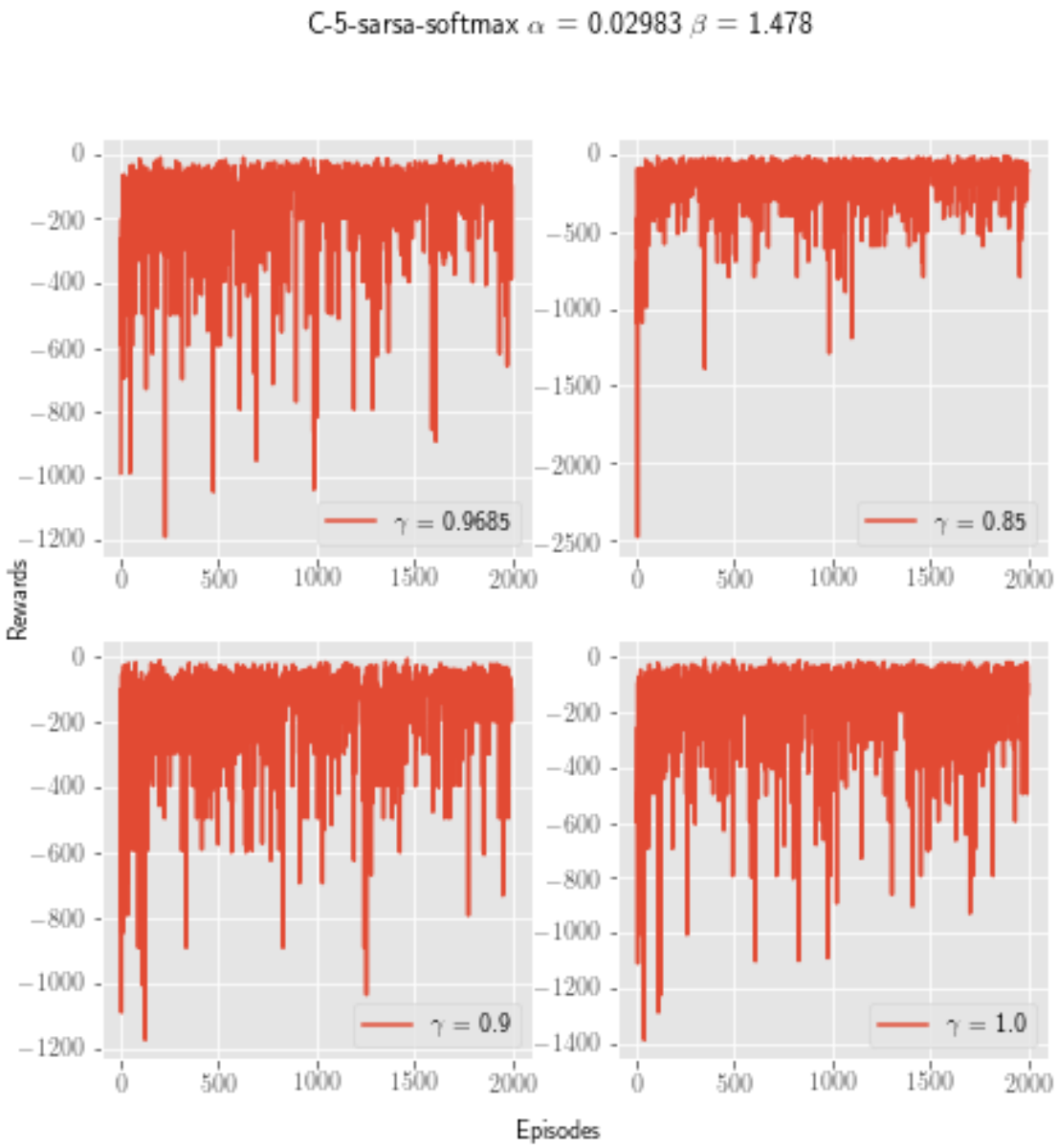

C-5-sarsa-e-greedy $\gamma = 0.9685$ $\alpha = 0.02983$
$\epsilon = 0.7722$
$\epsilon = 0.5$
$\epsilon = 0.3$
$\epsilon = 0.8$
Rewards
Episodes


C-5-sarsa-e-greedy $\gamma = 0.9685$ /$\epsilon = 0.7722$
Rewards
Episodes
$\alpha = 0.02983$
$\alpha = 0.1$
$\alpha = 0.07$
$\alpha = 0.01$
0
−500
−1000
−1500
−2000
−2500
−3000
0
500
1000
1500
2000


C-5-sarsa-e-greedy $\alpha = 0.02983$ $\epsilon = 0.7722$
Rewards
Episodes
$\gamma = 0.9685$
$\gamma = 0.85$
$\gamma = 0.9$
$\gamma = 1.0$


C-5-sarsa-softmax $\gamma = 0.9685$ $\alpha = 0.02983$
Rewards
Episodes
$\beta = 1.478$
$\beta = 1$
$\beta = 1.2$
$\beta = 1.3$


C-5-sarsa-softmax $\gamma = 0.9685$ $\beta = 1.478$
Rewards
Episodes
$\alpha = 0.02983$
$\alpha = 0.1$
$\alpha = 0.07$
$\alpha = 0.01$

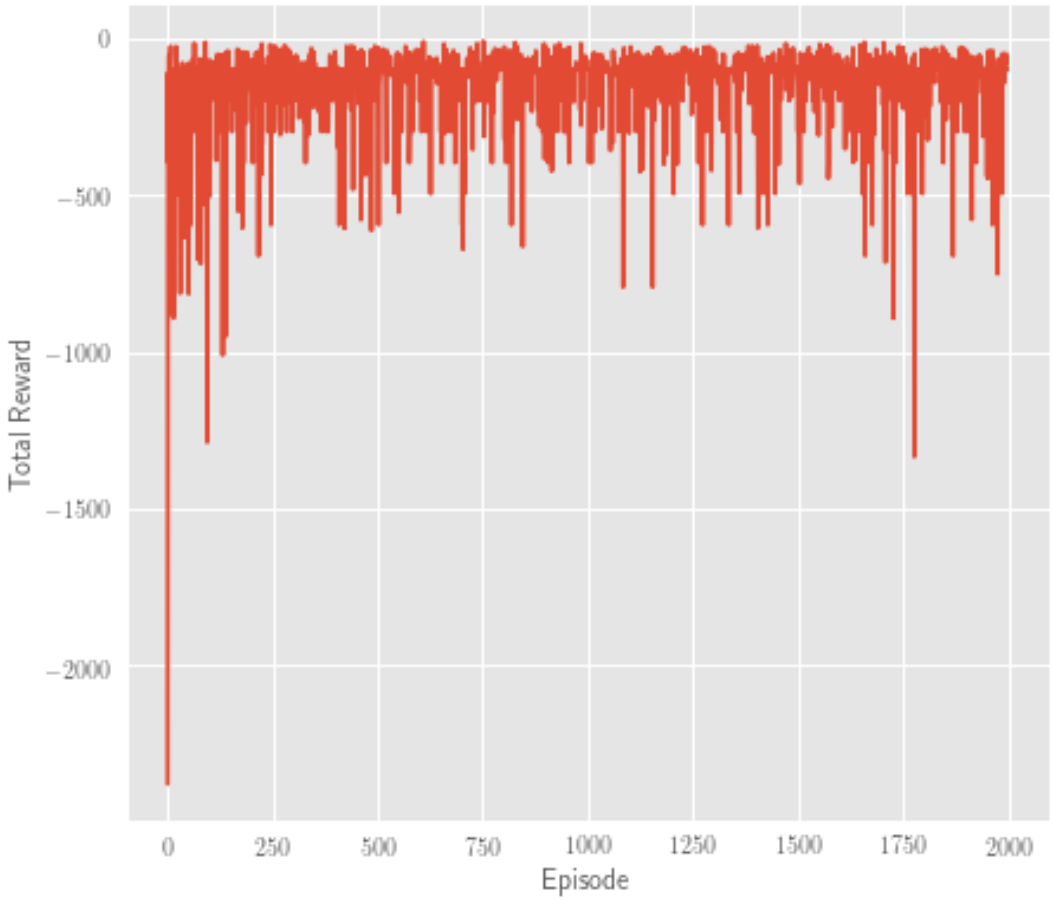


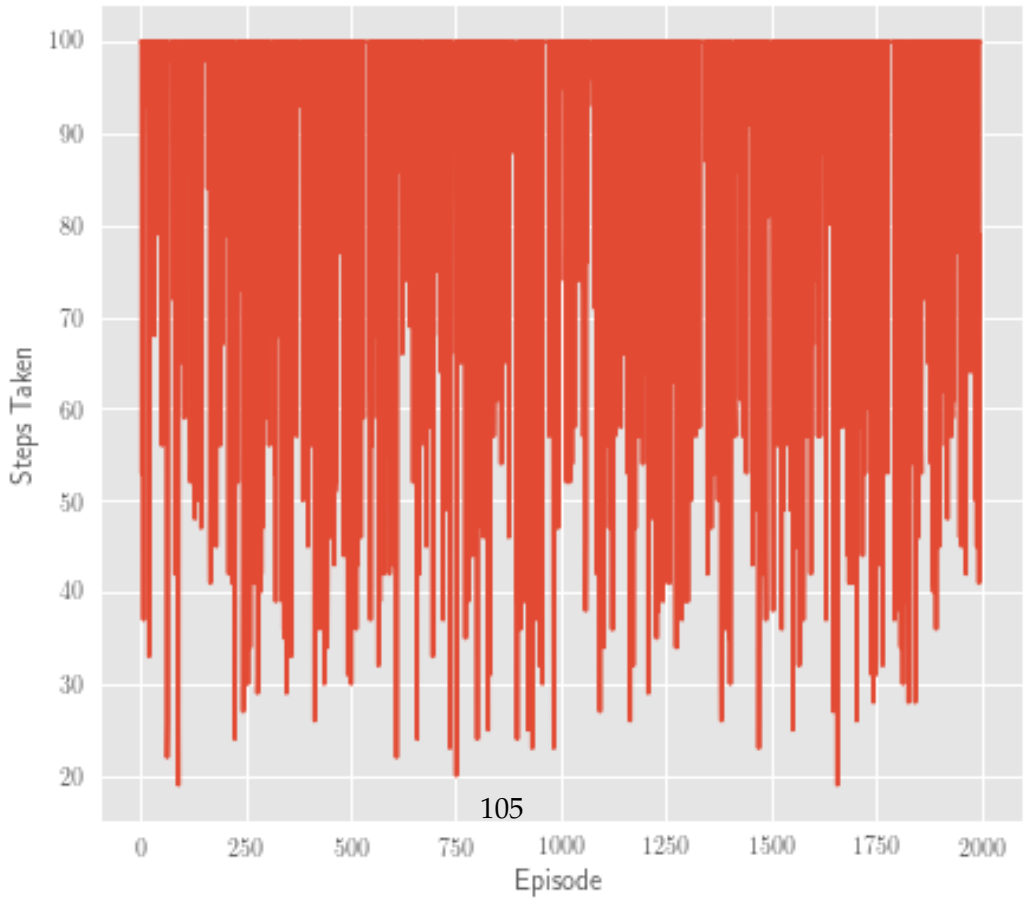
6.10.2 best plots

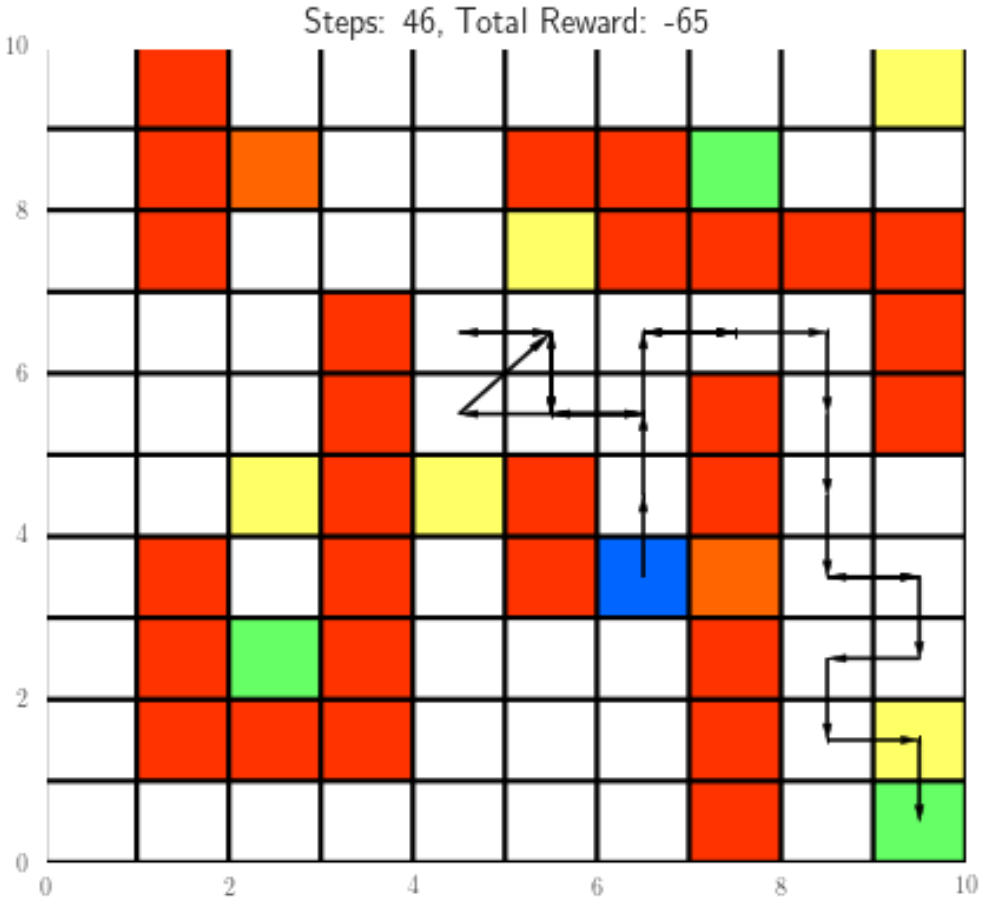
[ ]:
```
gamma = 0.85
```

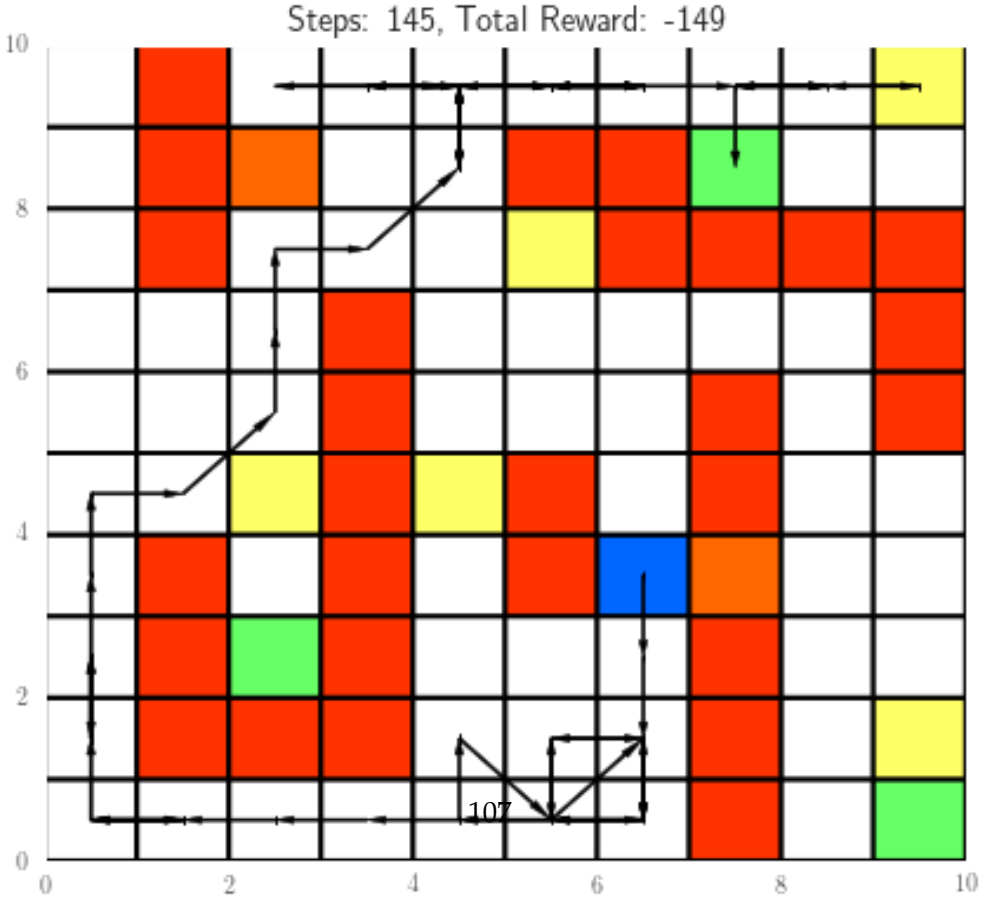
[ ]:
```
Q = best_plots(algorithm, policy, alpha,  epsilon, beta , gamma )
```

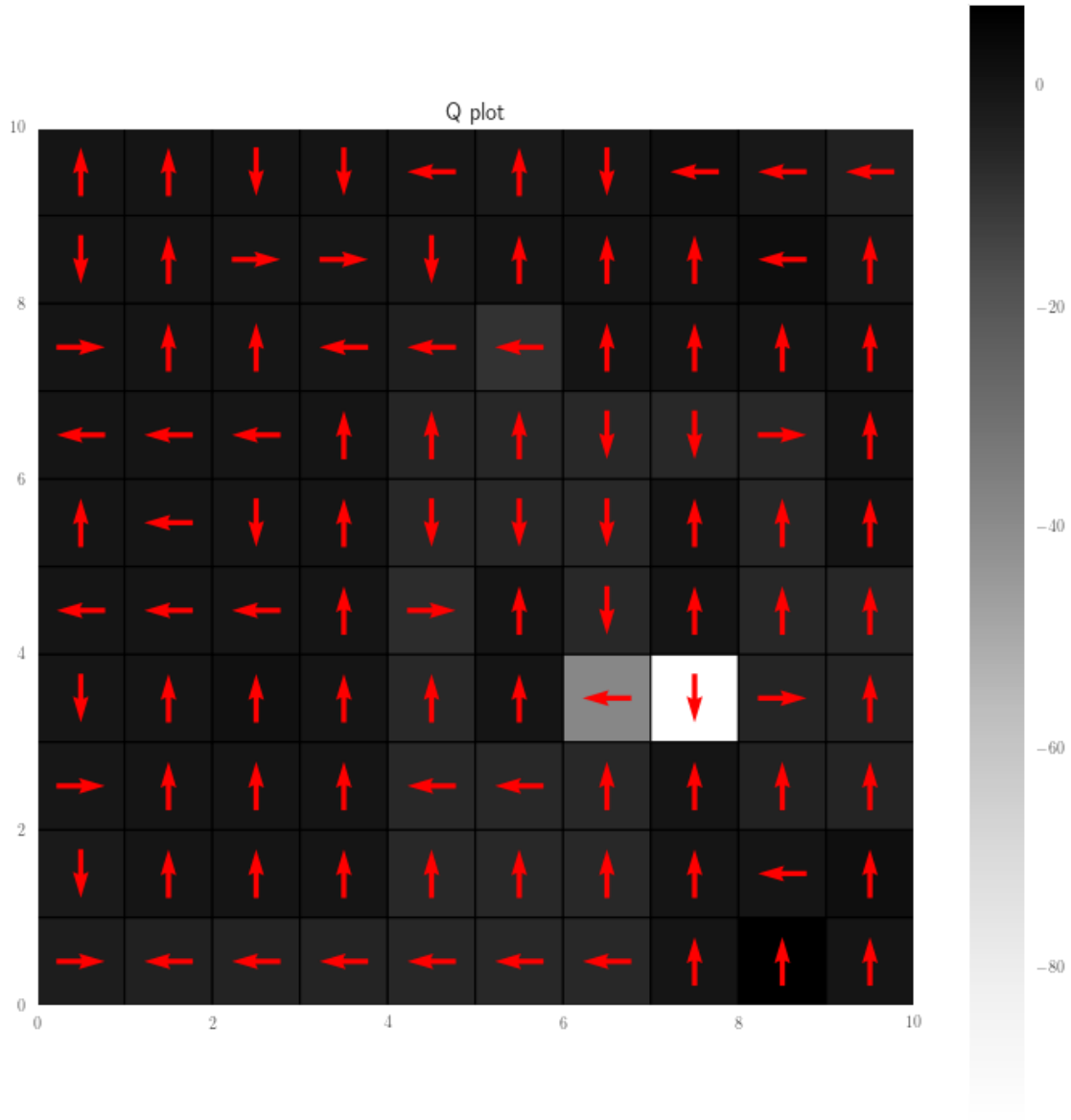
```
100%|| 2000/2000 [01:15<00:00, 26.61it/s]
```


0
−500
−1000
−1500
−2000
Total Reward
0
250
500
750
1000
1250
1500
1750
2000
Episode


100
90
80
70
60
50
40
30
20
Steps Taken
0
250
500
750
1000
1250
1500
1750
2000
Episode


Steps: 46, Total Reward: -65
10
8
6
4
2
0
0
2
4
6
8
10


Steps: 145, Total Reward: -149
10
8
6
4
2
0
0
2
4
6
8
10



6.11 Config 6

[ ]:
```
NUM_CONFIG = 6

config_settings = configurations_l[NUM_CONFIG]

# create environment
env = create_env(**config_settings._asdict())
```

```
sweep_config = {
    "name" : f"{NUM_CONFIG}-config-sweep",
    "method": "random",
    "parameters": {
        "algorithm": {
            "values": ['sarsa', 'q_learning'],
        },
        "policy": {
            "values": ['softmax', 'epsilon_greedy'],
        },
        "epsilon": {
            "min": 0.0,
            "max": 1.0,
        },
        "alpha": {
            "min": 0.01,
            "max": 0.2,
        },
        "gamma": {
            "min": 0.5,
            "max": 1.0,
        },
        "beta": {
            "min": 0.5,
            "max": 1.5,
        }
    }
}
```

[ ]:
```
sweep_id = wandb.sweep(sweep_config, project='RLPA1')
```

[ ]:
```
wandb.agent(sweep_id, run, count=20)
```

6.11.1 Plotting

[ ]:
```
# swwep 19

alpha = 0.1170
l_alpha = [0.1, 0.12, 0.09]
beta = 0.883
l_beta = [1, 0.8, 0.9]
gamma = 0.9851
l_gamma = [0.85, 0.9, 1.0]
epsilon = 0.4062
l_epsilon = [0.5, 0.2, 0.3]
policy = 'softmax'
```

```
algorithm = 'q-learning'
```

[ ]:
```
plt.style.use('seaborn')
```

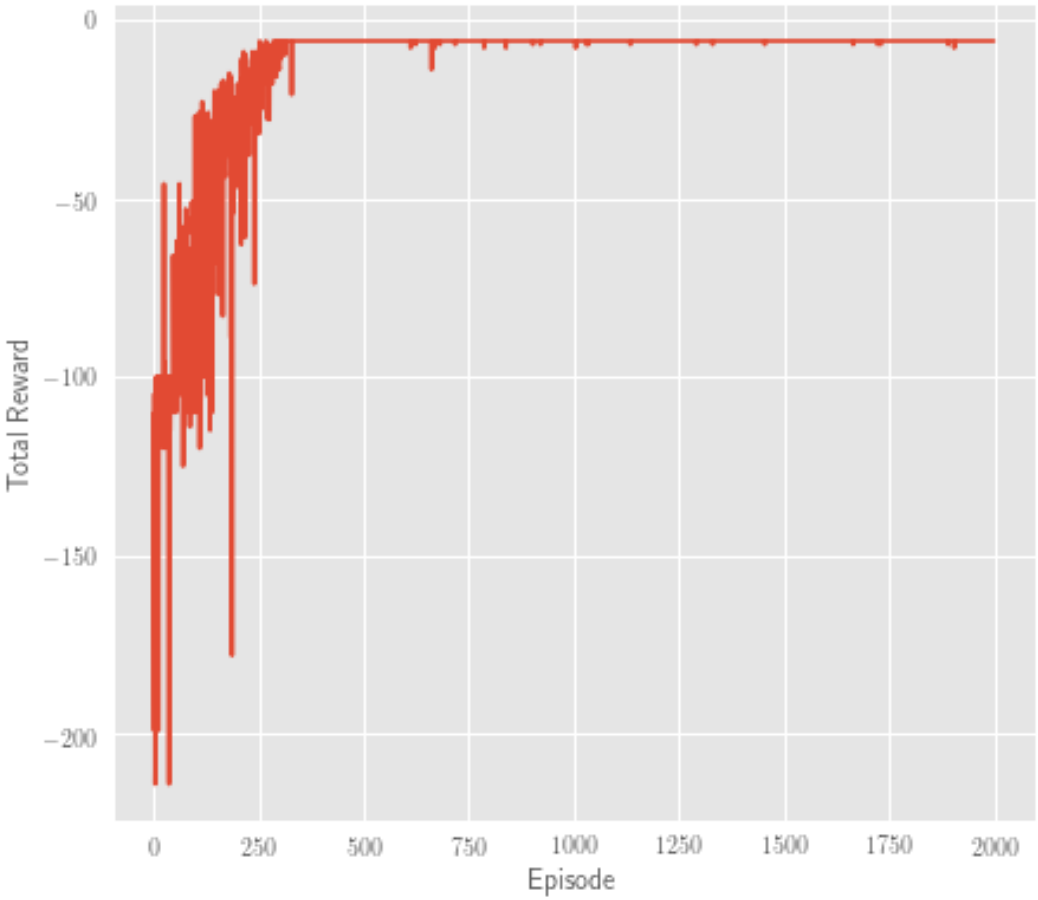
[ ]:
```
plot_all(alpha, l_alpha, beta, l_beta, gamma, l_gamma, epsilon, l_epsilon)
```

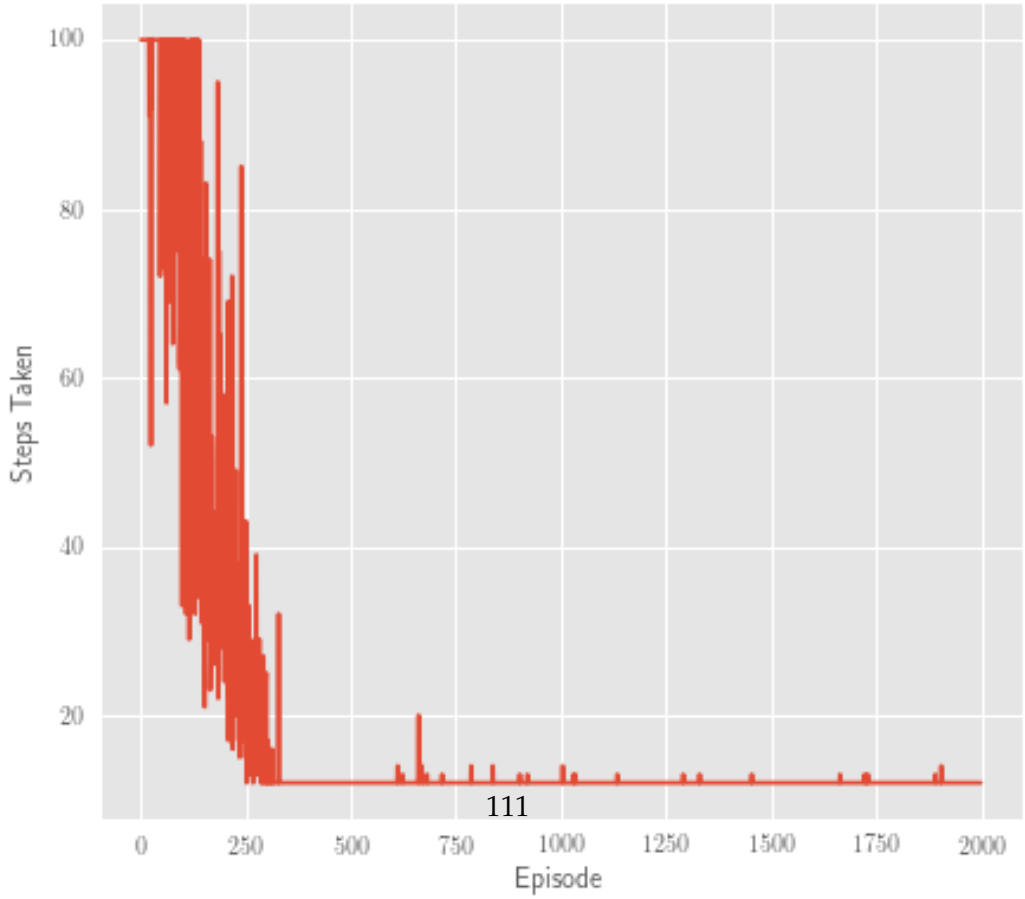
6.11.2 best plots

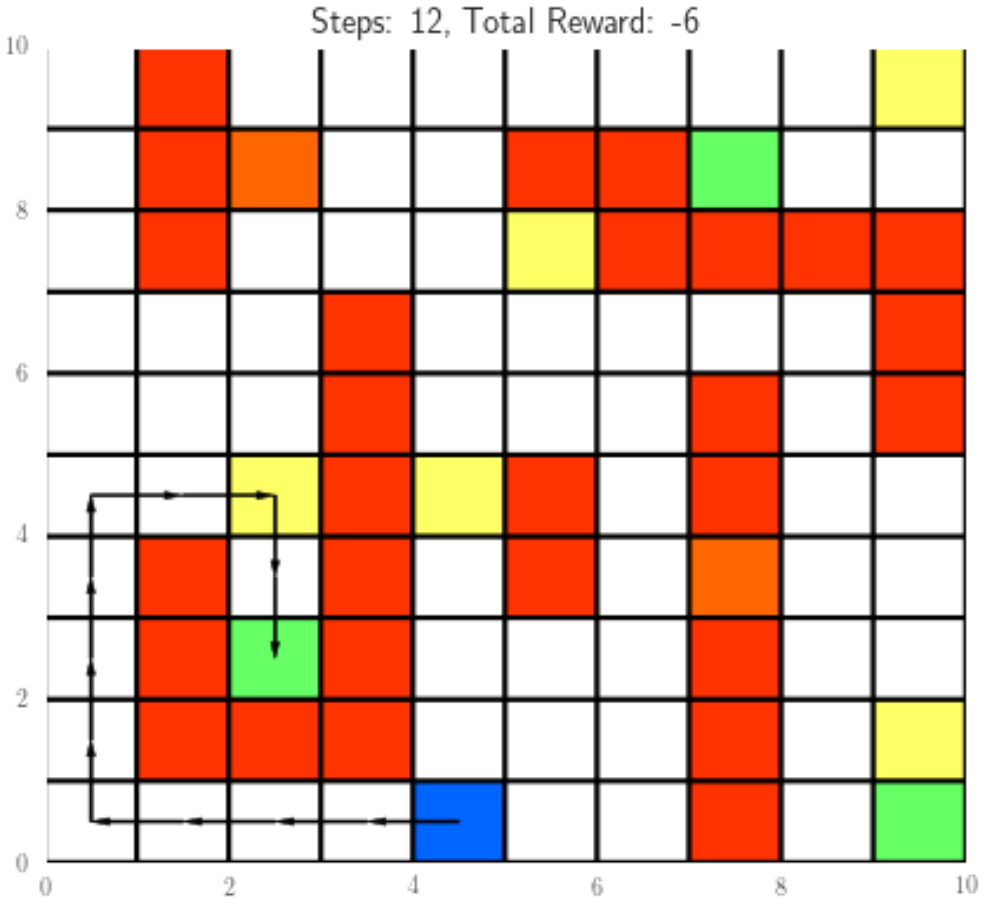
[ ]:
```
algorithm = 'sarsa'
```

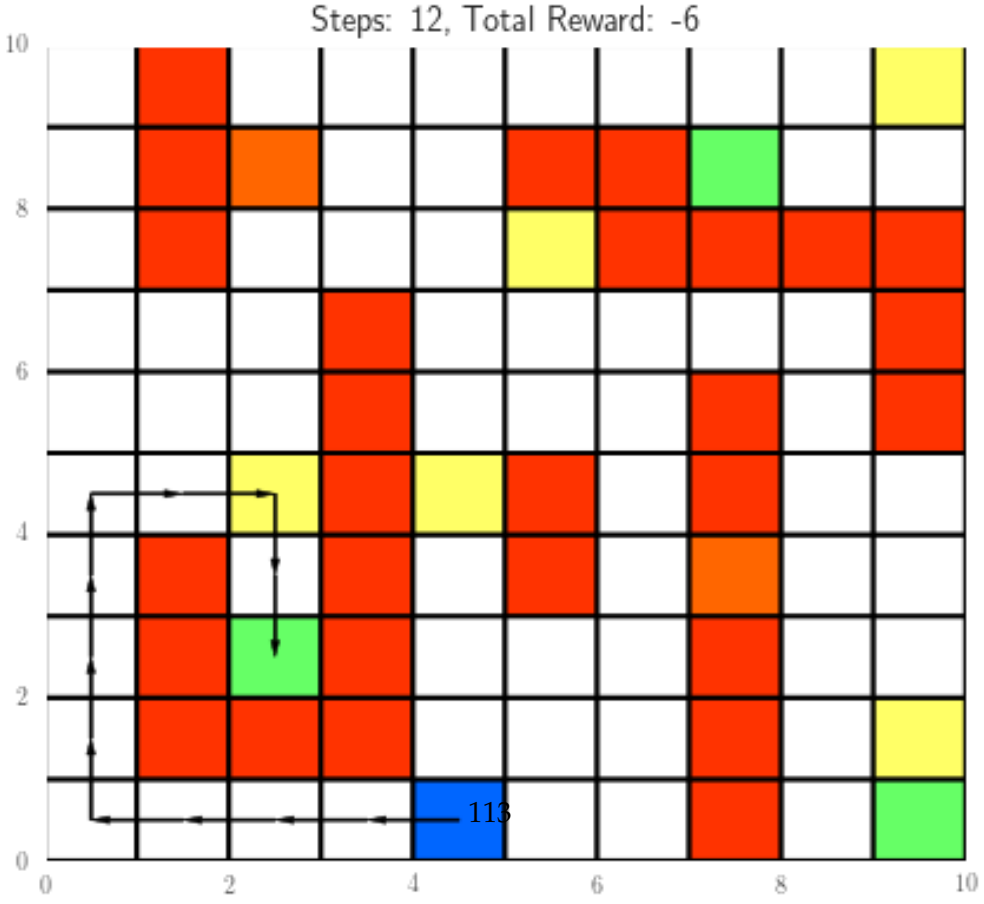
[ ]:
```
Q = best_plots(algorithm, policy, alpha,  epsilon, beta , gamma )
```

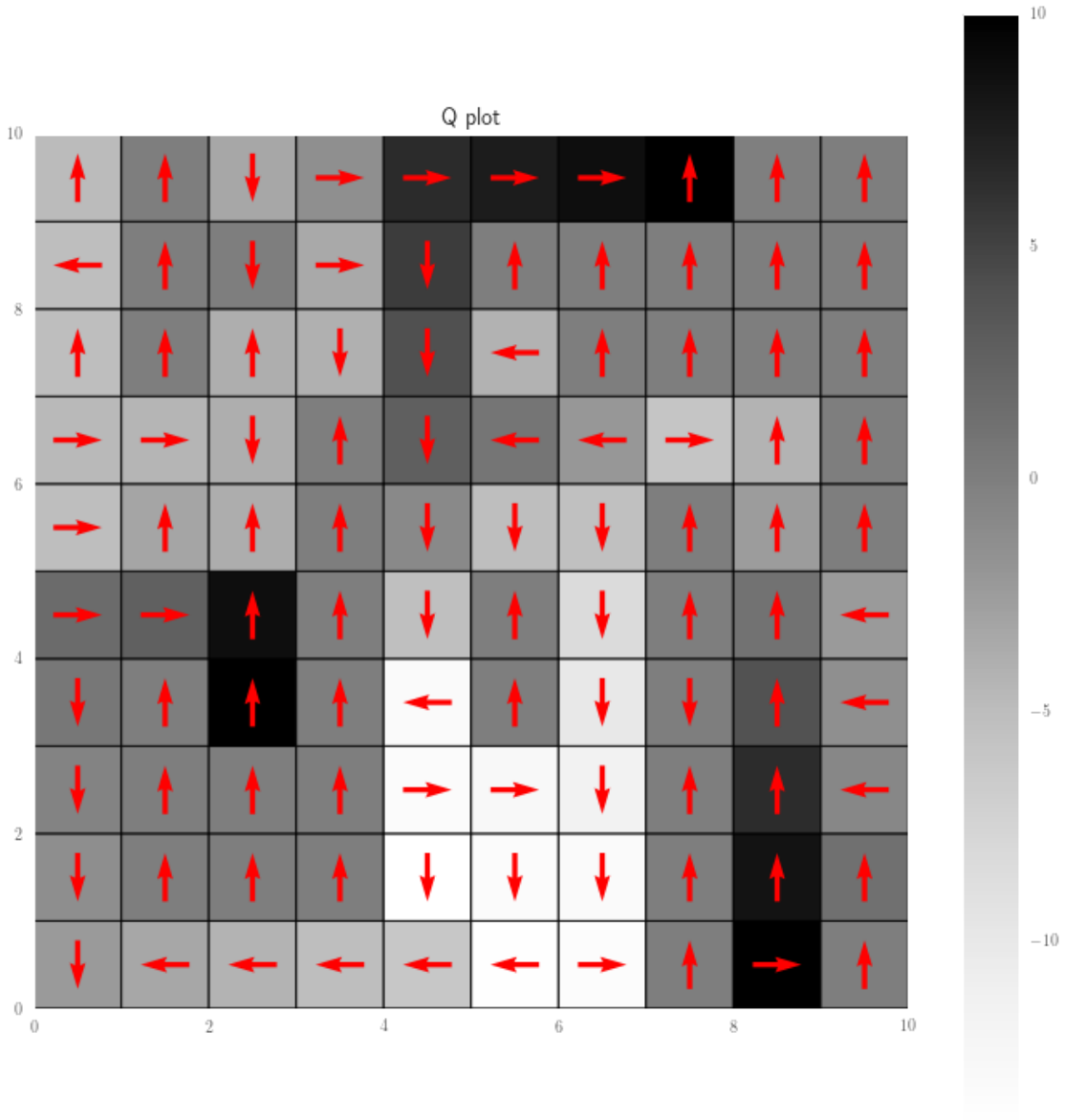
```
100%|| 2000/2000 [00:15<00:00, 132.92it/s]
```


Total Reward
0
−50
−100
−150
−200
0
250
500
750
1000
1250
1500
1750
2000
Episode


Steps Taken
100
80
60
40
20
0
250
500
750
1000
1250
1500
1750
2000
Episode


Steps: 12, Total Reward: -6
10
8
6
4
2
0
0
2
4
6
8
10


Steps: 12, Total Reward: -6
10
8
6
4
2
0
0
2
4
6
8
10



6.12 Config 7

[ ]:
```
NUM_CONFIG = 7

config_settings = configurations_l[NUM_CONFIG]

# create environment
env = create_env(**config_settings._asdict())
```

```
sweep_config = {
    "name" : f"{NUM_CONFIG}-config-sweep",
    "method": "random",
    "parameters": {
        "algorithm": {
            "values": ['sarsa', 'q_learning'],
        },
        "policy": {
            "values": ['softmax', 'epsilon_greedy'],
        },
        "epsilon": {
            "min": 0.0,
            "max": 1.0,
        },
        "alpha": {
            "min": 0.01,
            "max": 0.2,
        },
        "gamma": {
            "min": 0.5,
            "max": 1.0,
        },
        "beta": {
            "min": 0.5,
            "max": 1.5,
        }
    }
}
```

[ ]:
```
sweep_id = wandb.sweep(sweep_config, project='RLPA1')
```

```
Create sweep with ID: u6xuj3v1
Sweep URL: https://wandb.ai/sathvikjoel/RLPA1/sweeps/u6xuj3v1
```

[ ]:
```
wandb.agent(sweep_id, run, count=20)
```

6.12.1 Plotting

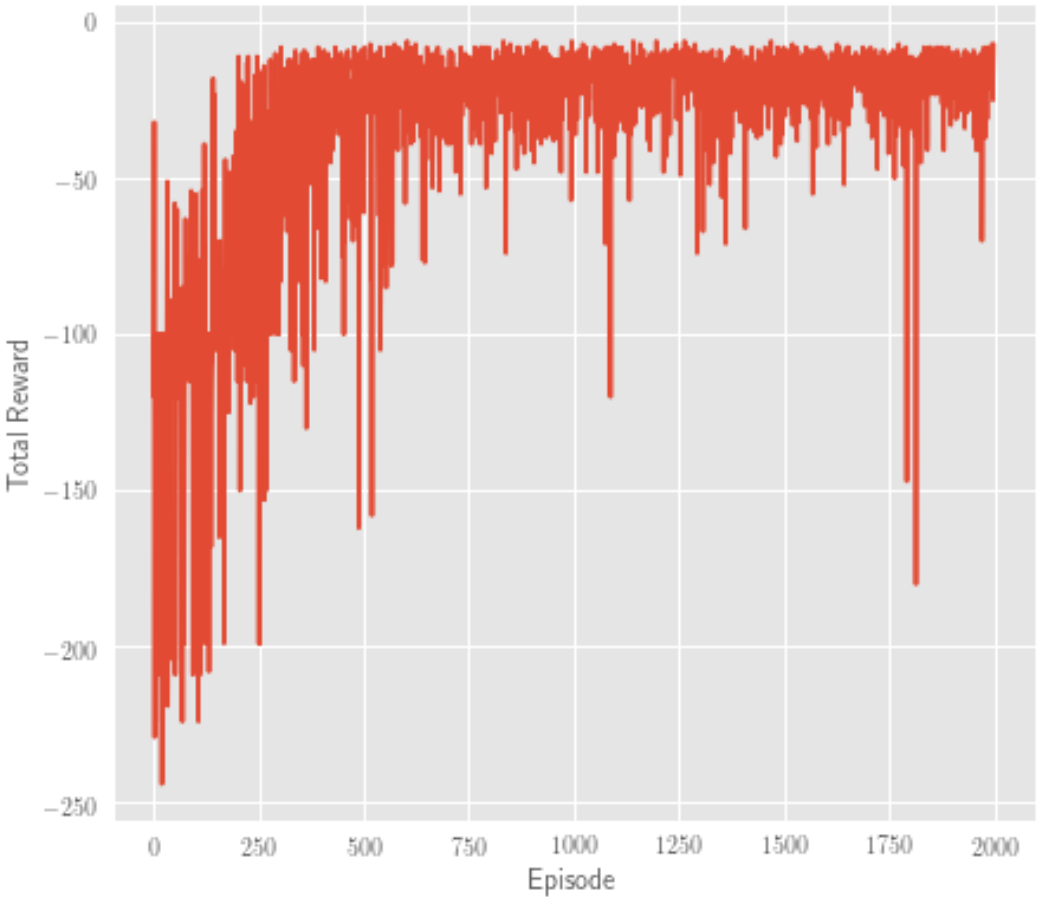
[ ]:
```
alpha = 0.06673
l_alpha = [0.05, 0.1, 0.09]
beta = 1.306
l_beta = [1, 1.2, 1.1]
gamma = 0.9298
l_gamma = [0.90, 0.91, 1.0]
epsilon = 0.0993
l_epsilon = [0.1, 0.15, 0.05]
```

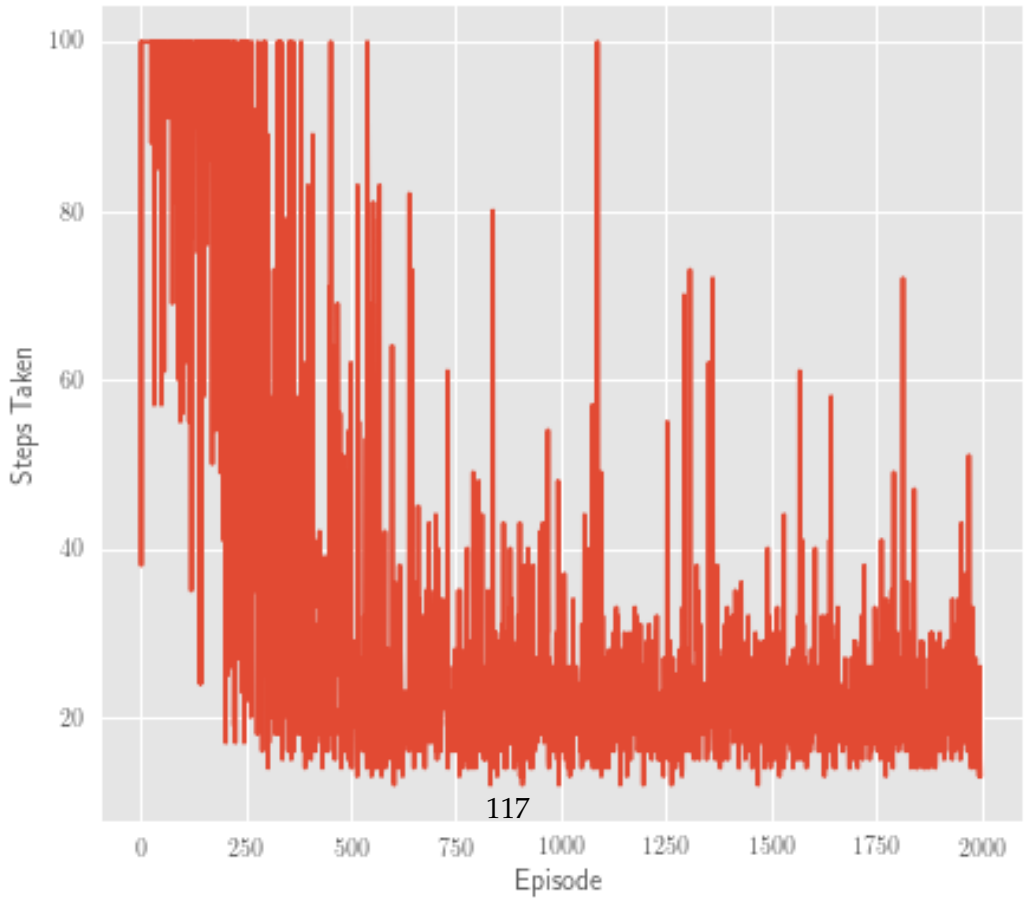
```
policy = 'e-greedy'
algorithm = 'sarsa'
```

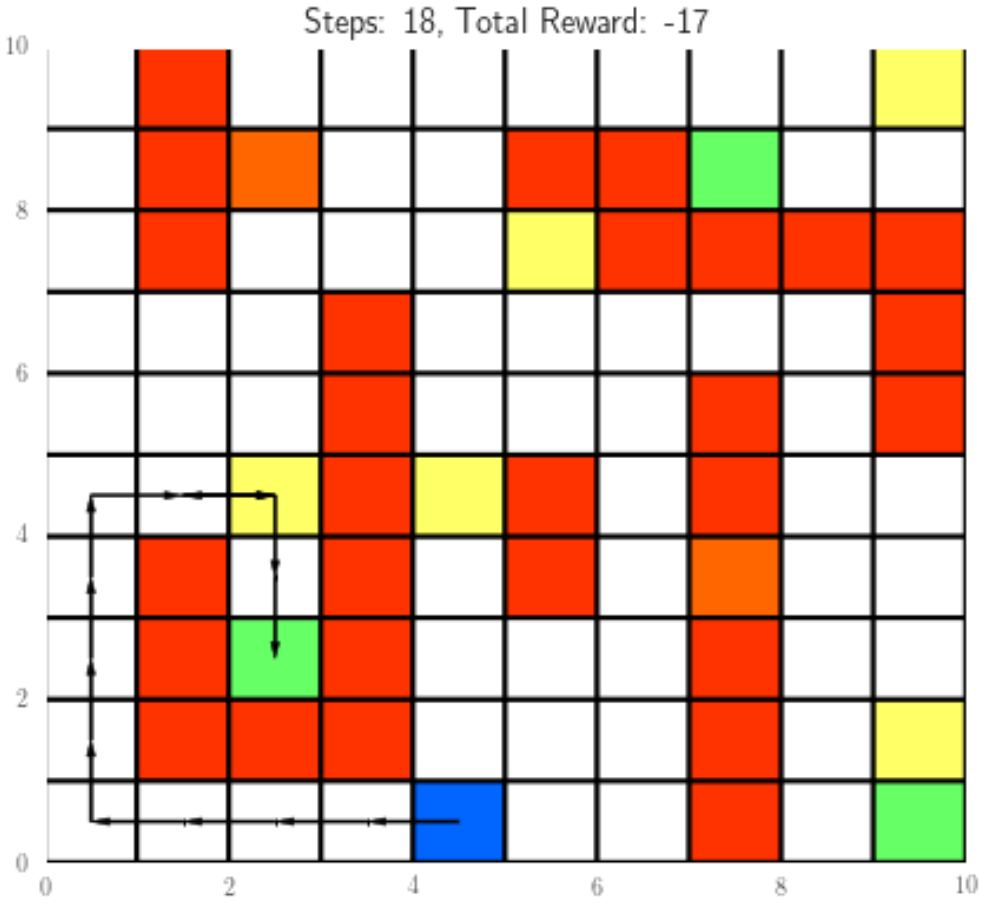
[ ]:
```
plot_all(alpha, l_alpha, beta, l_beta, gamma, l_gamma, epsilon, l_epsilon)
```

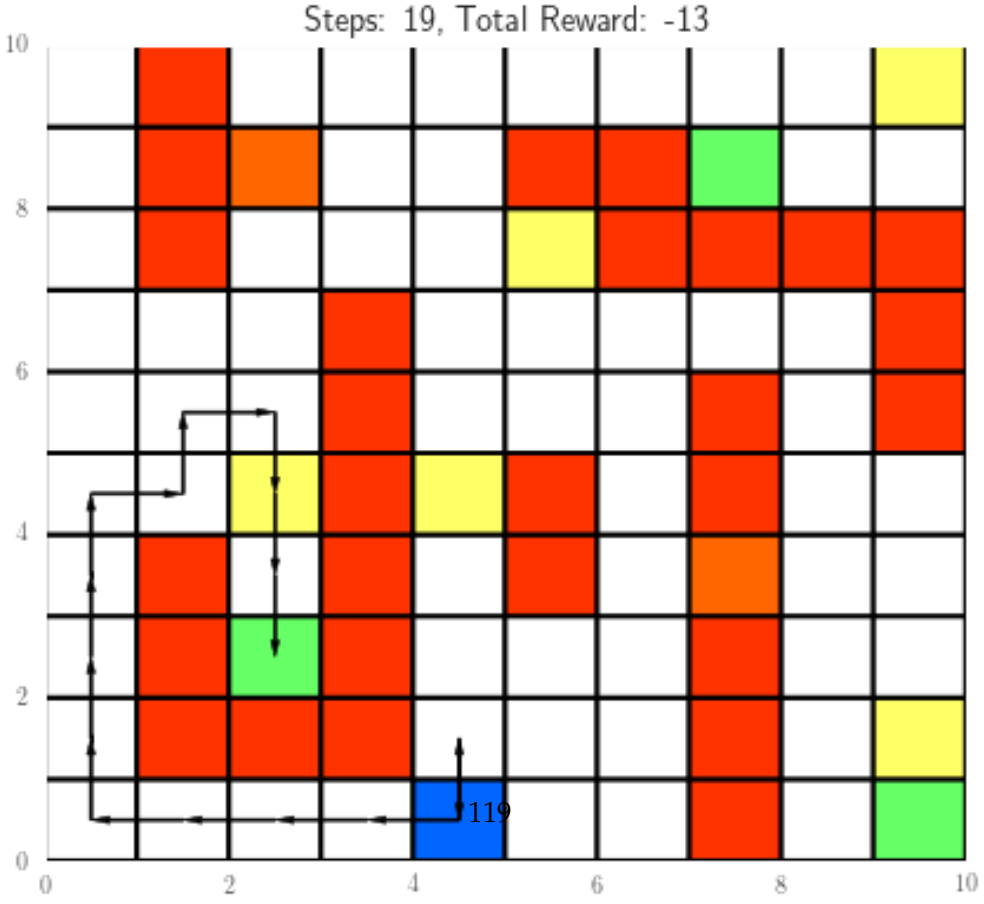
[ ]:
```
Q = best_plots(algorithm, policy, alpha,  epsilon, beta , gamma )
```

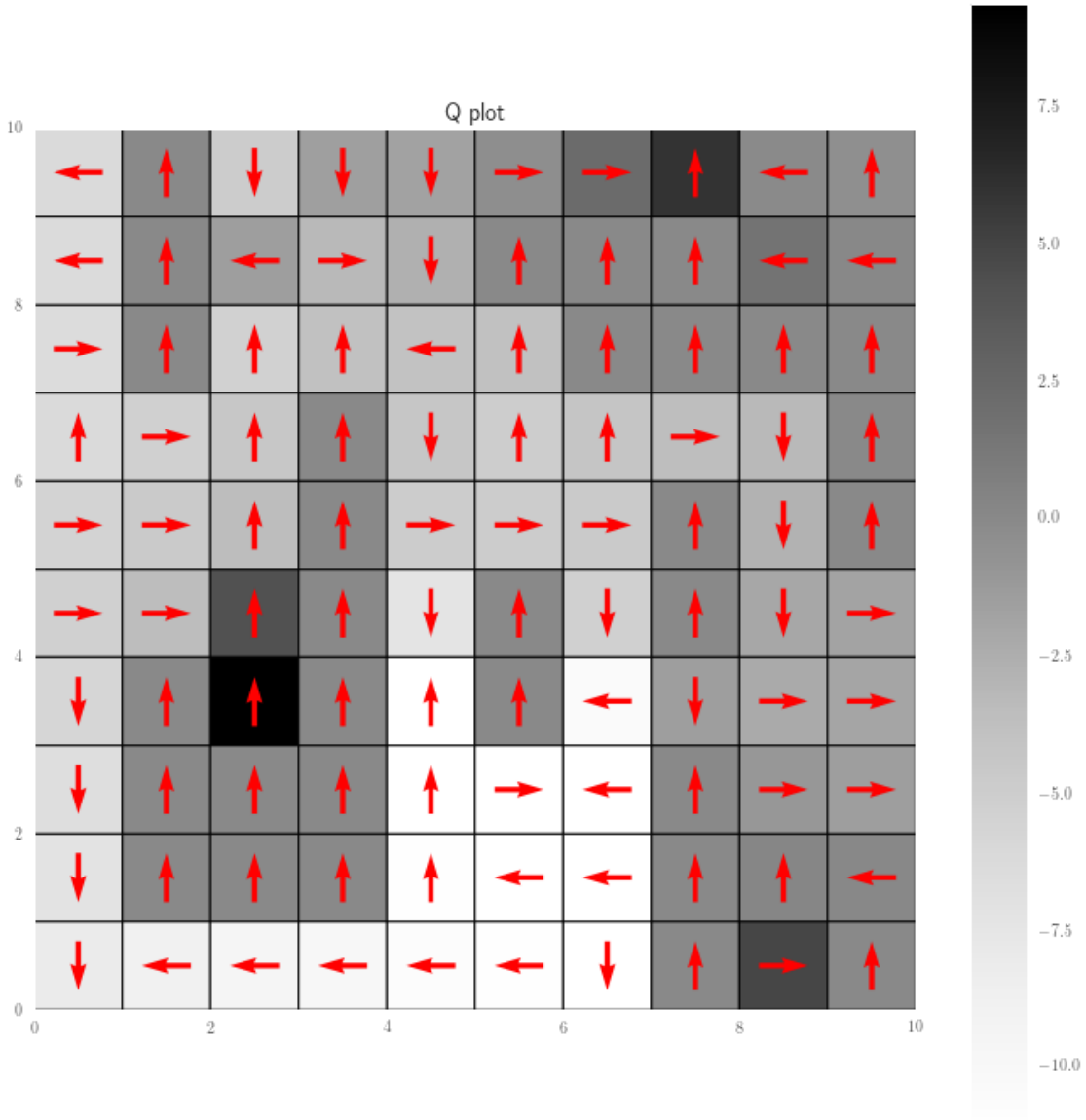
```
100%|| 2000/2000 [00:11<00:00, 168.62it/s]
```


Total Reward
0
−50
−100
−150
−200
−250
0
250
500
750
1000
1250
1500
1750
2000
Episode


Steps Taken
100
80
60
40
20
0
250
500
750
1000
1250
1500
1750
2000
Episode


Steps: 18, Total Reward: -17
10
8
6
4
2
0
0
2
4
6
8
10


Steps: 19, Total Reward: -13
10
8
6
4
2
0
0
2
4
6
8
10



6.13 config 8

[ ]:
```
NUM_CONFIG = 8

config_settings = configurations_1[NUM_CONFIG]

# create environment
env = create_env(**config_settings._asdict())
```

```
sweep_config = {
    "name" : f"{NUM_CONFIG}-config-sweep",
    "method": "random",
    "parameters": {
        "algorithm": {
            "values": ['sarsa', 'q_learning'],
        },
        "policy": {
            "values": ['softmax', 'epsilon_greedy'],
        },
        "epsilon": {
            "min": 0.0,
            "max": 1.0,
        },
        "alpha": {
            "min": 0.01,
            "max": 0.2,
        },
        "gamma": {
            "min": 0.5,
            "max": 1.0,
        },
        "beta": {
            "min": 0.5,
            "max": 1.5,
        }
    }
}
```

[ ]:
```
sweep_id = wandb.sweep(sweep_config, project='RLPA1')
```

```
Create sweep with ID: api7o9ik
Sweep URL: https://wandb.ai/sathvikjoel/RLPA1/sweeps/api7o9ik
```

[ ]:
```
wandb.agent(sweep_id, run, count=20)
```

6.13.1 Plotting

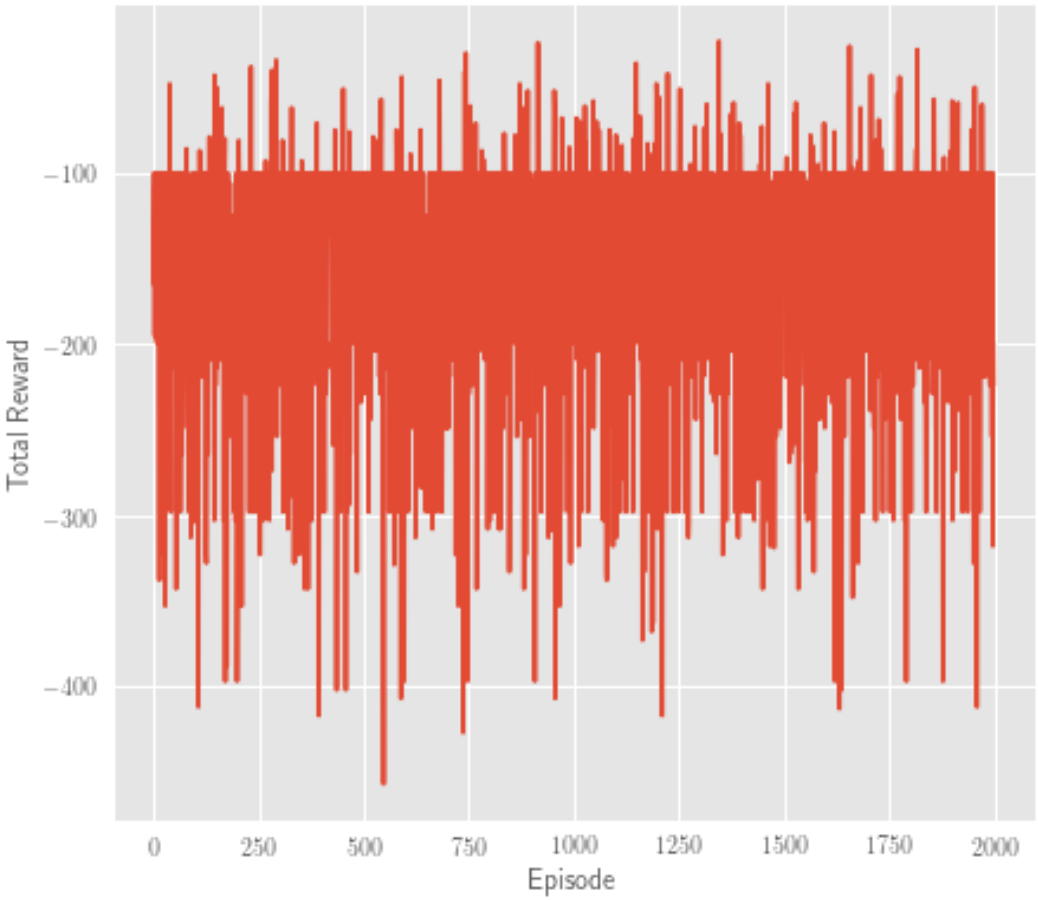
[ ]:
```
alpha = 0.09762
l_alpha = [0.05, 0.1, 0.09]
beta = 0.9012
l_beta = [1, 1.2, 1.1]
gamma = 0.9874
l_gamma = [0.90, 0.95, 1.0]
epsilon = 0.8899
l_epsilon = [0.80, 0.90, 0.85]
```

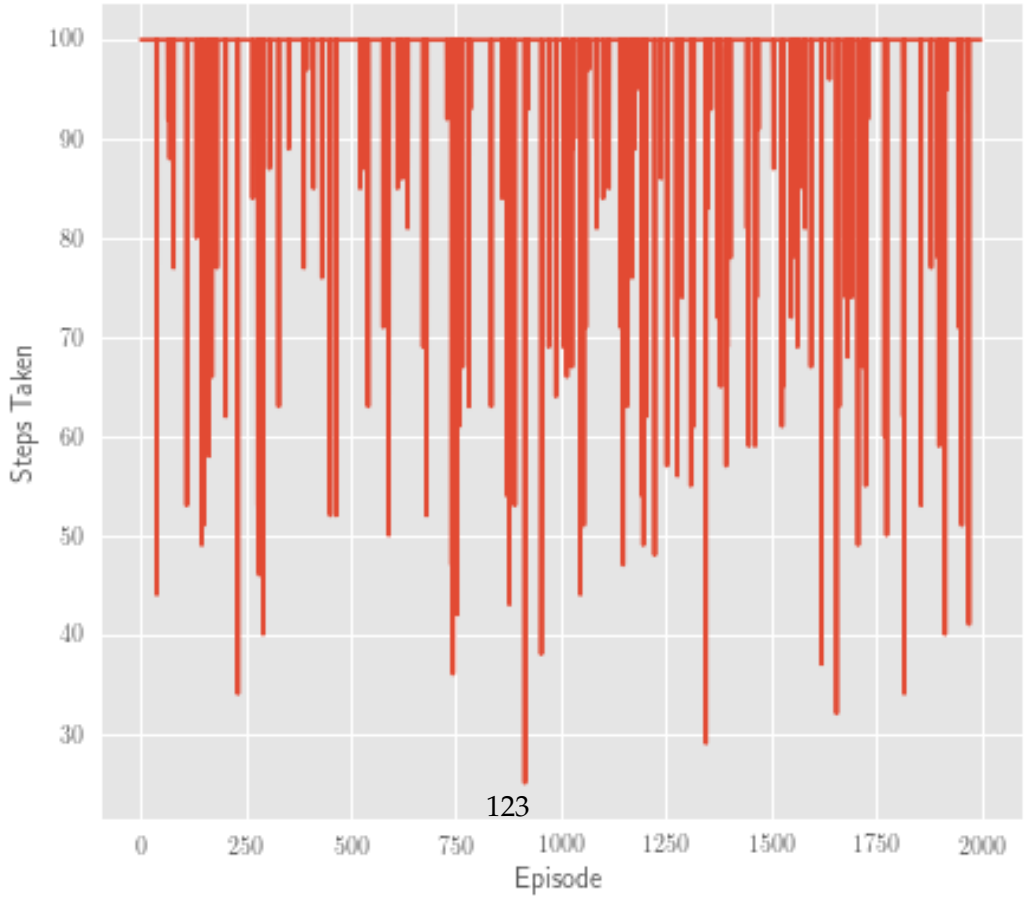
```
policy = 'e-greedy'
algorithm = 'q_learning'
```

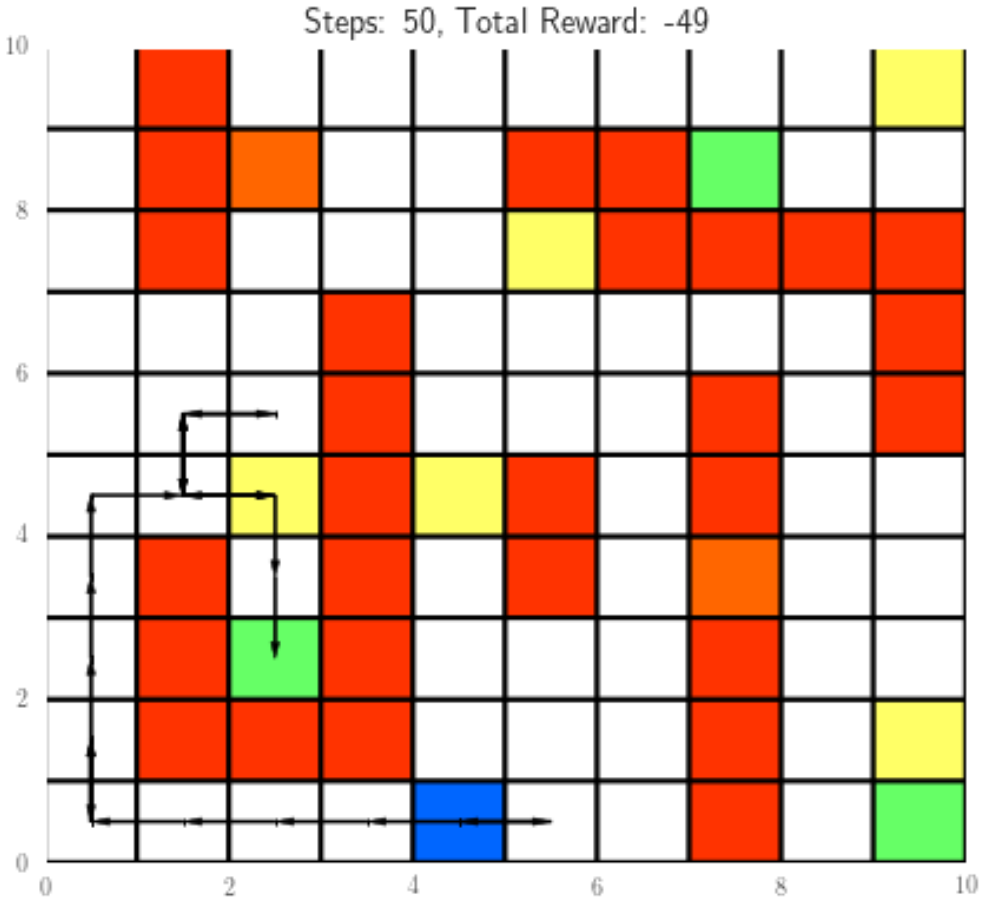
[ ]:
```
plot_all(alpha, l_alpha, beta, l_beta, gamma, l_gamma, epsilon, l_epsilon)
```

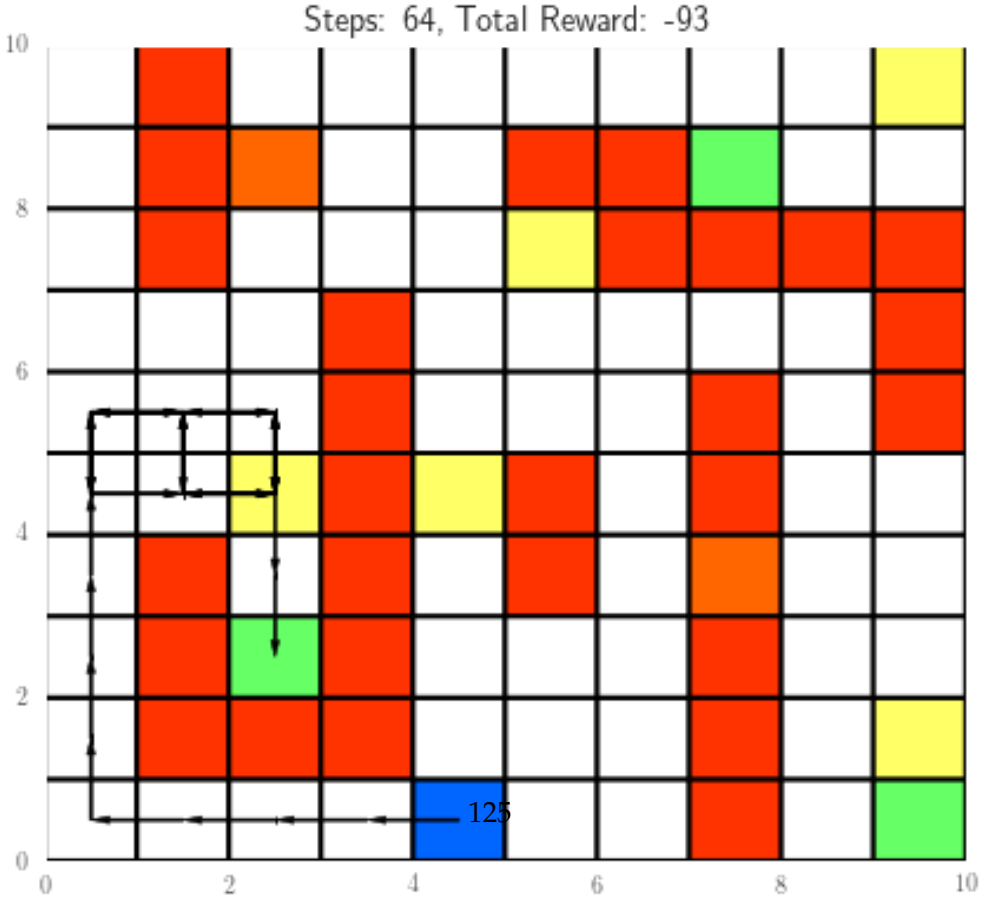
[ ]:
```
Q = best_plots(algorithm, policy, alpha,  epsilon, beta , gamma )
```

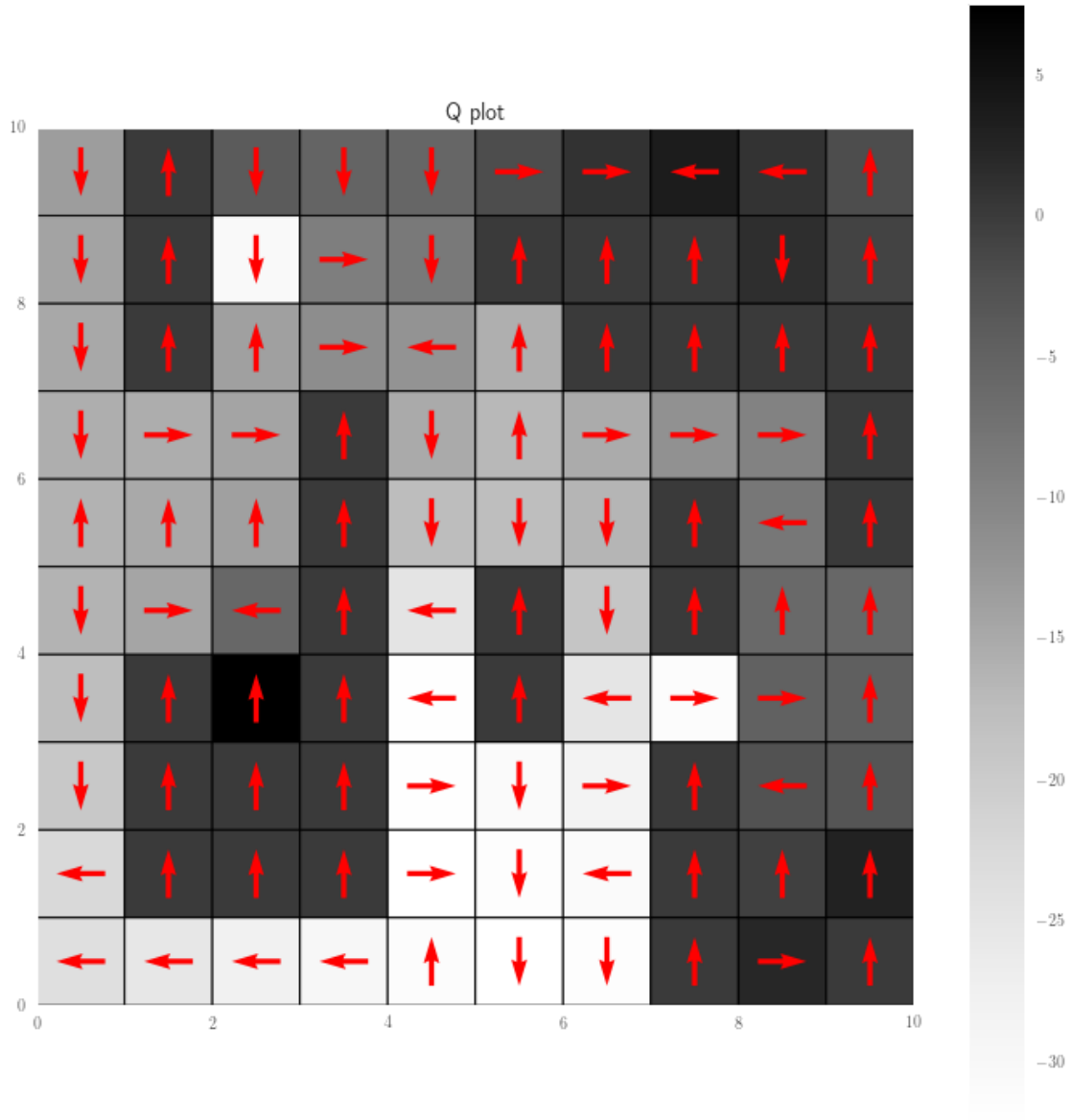
```
100%|| 2000/2000 [00:37<00:00, 53.98it/s]
```


Total Reward
−100
−200
−300
−400
0
250
500
750
1000
1250
1500
1750
2000
Episode


Steps Taken
100
90
80
70
60
50
40
30
0
250
500
750
1000
1250
1500
1750
2000
Episode


Steps: 50, Total Reward: -49
10
8
6
4
2
0
0
2
4
6
8
10


Steps: 64, Total Reward: -93
10
8
6
4
2
0
0
2
4
6
8
10



6.14 config 11

[ ]:
```
NUM_CONFIG = 11

config_settings = configurations_l[NUM_CONFIG]

# create environment
env = create_env(**config_settings._asdict())
```

```
sweep_config = {
    "name" : f"{NUM_CONFIG}-config-sweep",
    "method": "random",
    "parameters": {
        "algorithm": {
            "values": ['sarsa', 'q_learning'],
        },
        "policy": {
            "values": ['softmax', 'epsilon_greedy'],
        },
        "epsilon": {
            "min": 0.0,
            "max": 1.0,
        },
        "alpha": {
            "min": 0.01,
            "max": 0.2,
        },
        "gamma": {
            "min": 0.5,
            "max": 1.0,
        },
        "beta": {
            "min": 0.5,
            "max": 1.5,
        }
    }
}
```

[ ]:
```
sweep_id = wandb.sweep(sweep_config, project='RLPA1')
```

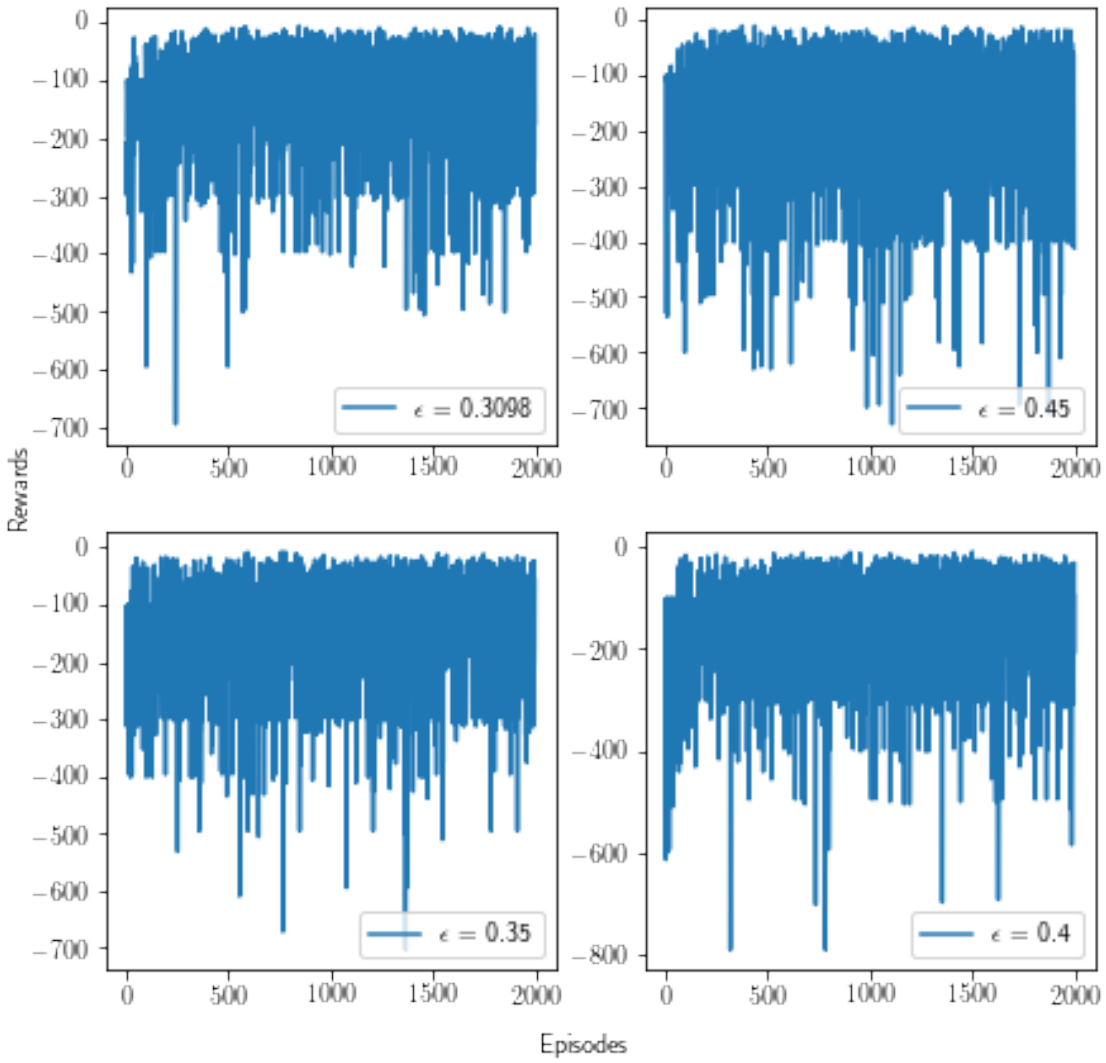
[ ]:
```
wandb.agent(sweep_id, run, count=20)
```

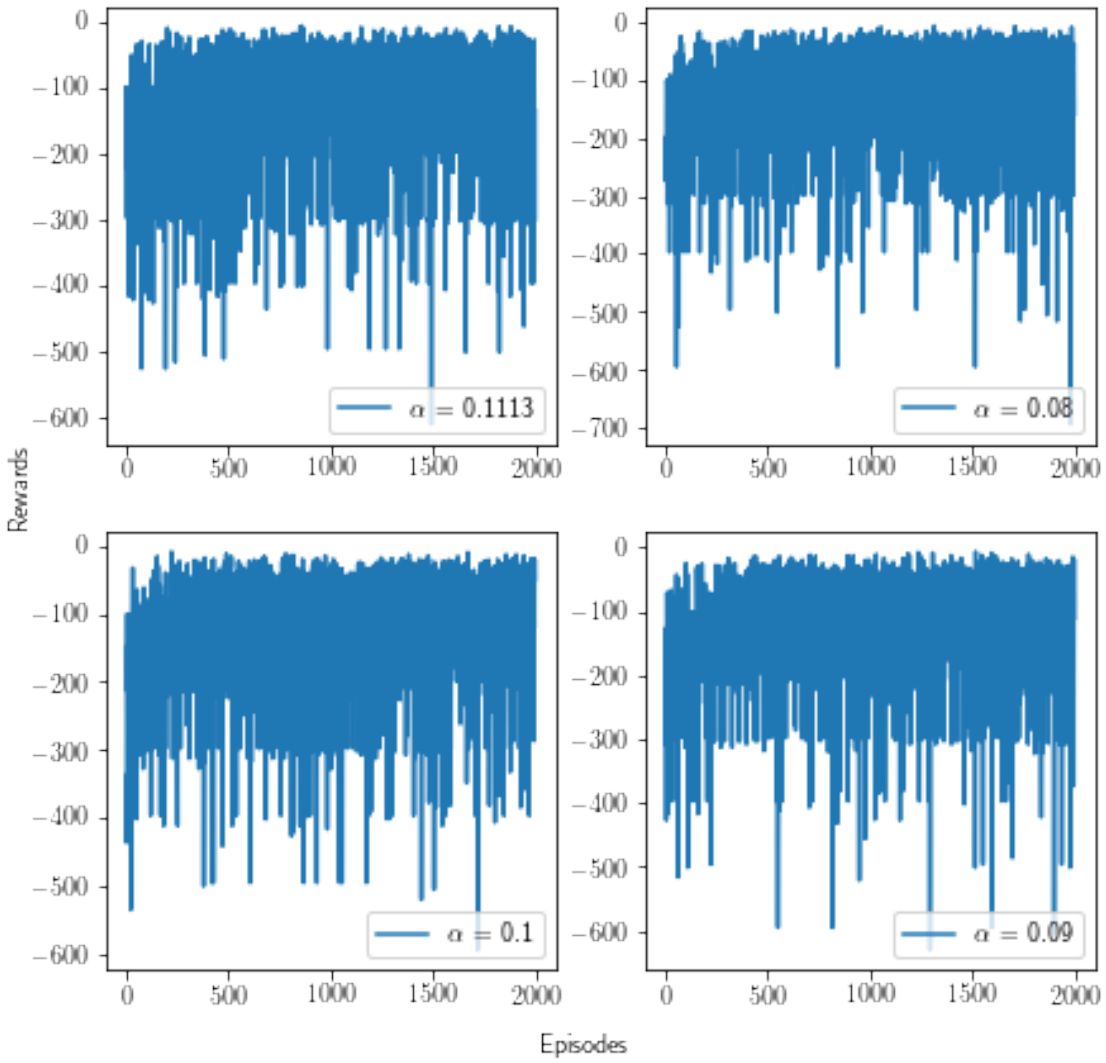
6.14.1 Plotting

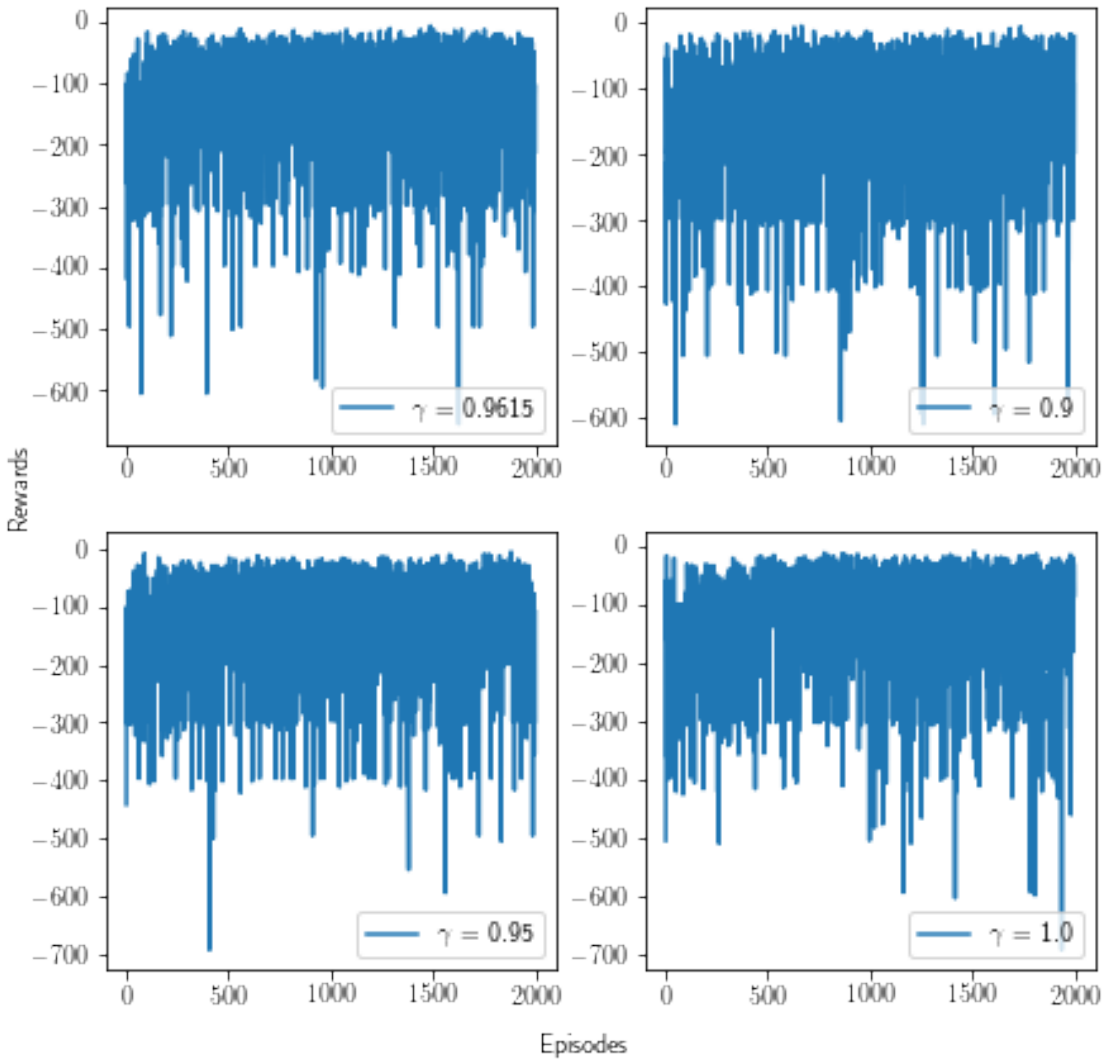
[ ]:
```
alpha = 0.1113
l_alpha = [0.08, 0.1, 0.09]
beta = 1.481
l_beta = [.55, 1.3, 1.1]
gamma = 0.9615
l_gamma = [0.90, 0.95, 1.0]
epsilon = 0.3098
l_epsilon = [0.45, 0.35, 0.40]
policy = 'e-greedy'
algorithm = 'q_learning'
```

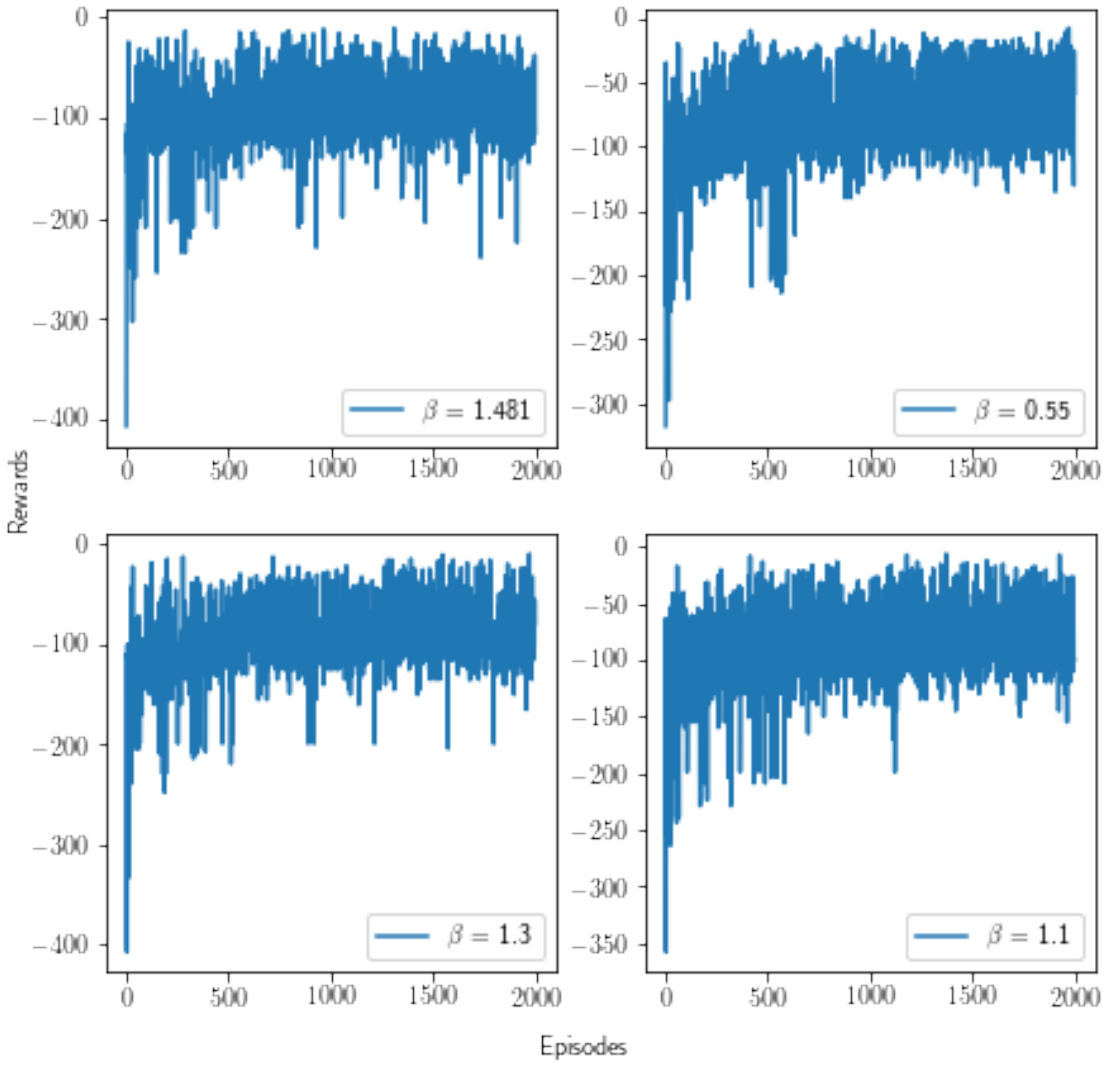
[ ]:
```
plot_all(alpha, l_alpha, beta, l_beta, gamma, l_gamma, epsilon, l_epsilon)
```

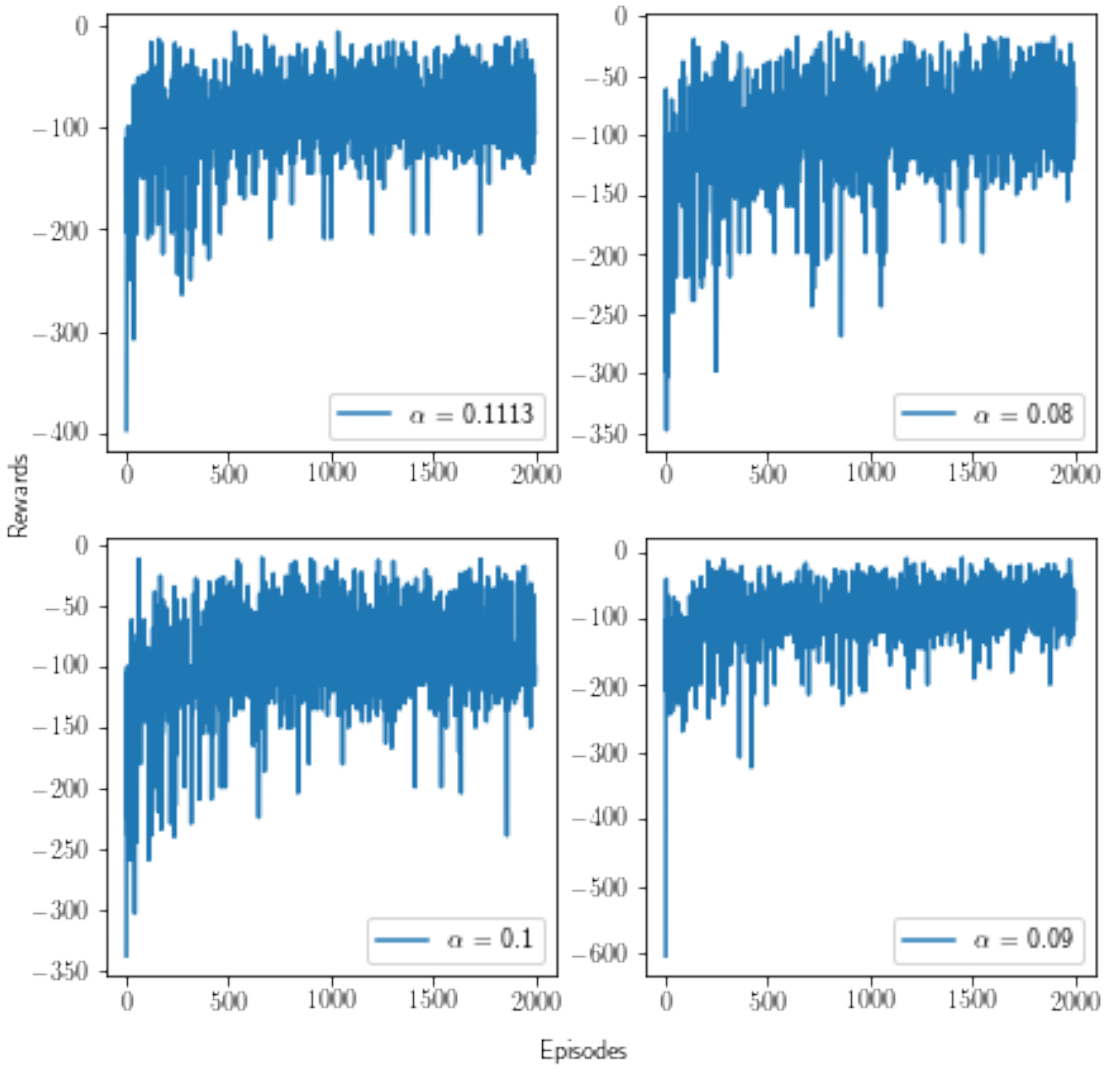
```
100%|| 2000/2000 [00:37<00:00, 52.88it/s]
100%|| 2000/2000 [00:40<00:00, 49.08it/s]
100%|| 2000/2000 [00:38<00:00, 51.36it/s]
100%|| 2000/2000 [00:40<00:00, 49.79it/s]
100%|| 2000/2000 [00:38<00:00, 52.33it/s]
100%|| 2000/2000 [00:37<00:00, 52.64it/s]
100%|| 2000/2000 [00:38<00:00, 52.50it/s]
100%|| 2000/2000 [00:37<00:00, 52.83it/s]
100%|| 2000/2000 [00:38<00:00, 51.89it/s]
100%|| 2000/2000 [00:40<00:00, 49.31it/s]
100%|| 2000/2000 [00:39<00:00, 51.12it/s]
100%|| 2000/2000 [00:37<00:00, 53.54it/s]
100%|| 2000/2000 [01:12<00:00, 27.74it/s]
100%|| 2000/2000 [01:04<00:00, 31.01it/s]
100%|| 2000/2000 [01:12<00:00, 27.51it/s]
100%|| 2000/2000 [01:10<00:00, 28.42it/s]
100%|| 2000/2000 [01:13<00:00, 27.07it/s]
100%|| 2000/2000 [01:14<00:00, 27.01it/s]
100%|| 2000/2000 [01:12<00:00, 27.46it/s]
100%|| 2000/2000 [01:12<00:00, 27.63it/s]
100%|| 2000/2000 [01:12<00:00, 27.58it/s]
100%|| 2000/2000 [01:14<00:00, 26.69it/s]
100%|| 2000/2000 [01:12<00:00, 27.41it/s]
100%|| 2000/2000 [01:08<00:00, 29.16it/s]
100%|| 2000/2000 [00:36<00:00, 54.74it/s]
100%|| 2000/2000 [00:38<00:00, 52.24it/s]
100%|| 2000/2000 [00:36<00:00, 54.53it/s]
100%|| 2000/2000 [00:38<00:00, 52.25it/s]
100%|| 2000/2000 [00:36<00:00, 54.14it/s]
100%|| 2000/2000 [00:37<00:00, 53.82it/s]
100%|| 2000/2000 [00:35<00:00, 55.91it/s]
100%|| 2000/2000 [00:36<00:00, 55.22it/s]
100%|| 2000/2000 [00:36<00:00, 54.11it/s]
100%|| 2000/2000 [00:40<00:00, 49.16it/s]
100%|| 2000/2000 [00:39<00:00, 50.82it/s]
100%|| 2000/2000 [00:36<00:00, 54.47it/s]
100%|| 2000/2000 [01:09<00:00, 28.92it/s]
100%|| 2000/2000 [01:00<00:00, 32.90it/s]
100%|| 2000/2000 [01:08<00:00, 29.25it/s]
100%|| 2000/2000 [01:08<00:00, 29.17it/s]
100%|| 2000/2000 [01:09<00:00, 28.67it/s]
100%|| 2000/2000 [01:10<00:00, 28.39it/s]
100%|| 2000/2000 [01:08<00:00, 28.99it/s]
100%|| 2000/2000 [01:08<00:00, 29.00it/s]
100%|| 2000/2000 [01:08<00:00, 29.01it/s]
100%|| 2000/2000 [01:11<00:00, 28.06it/s]
100%|| 2000/2000 [01:10<00:00, 28.37it/s]
100%|| 2000/2000 [01:01<00:00, 32.40it/s]
```

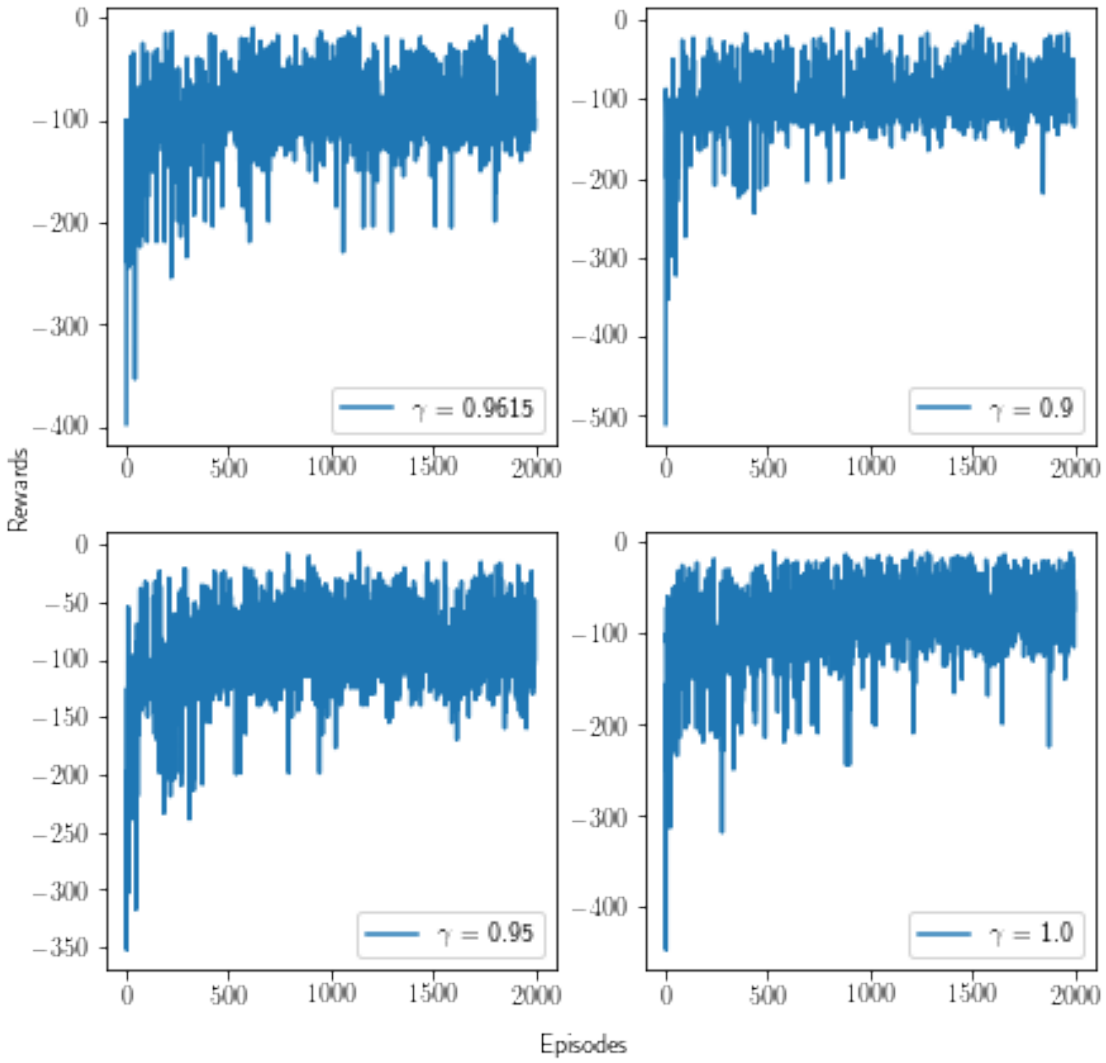

C-11-Q-learning-e-greedy $\gamma = 0.9615$ $\alpha = 0.1113$
Rewards
Episodes
$\epsilon = 0.3098$
$\epsilon = 0.45$
$\epsilon = 0.35$
$\epsilon = 0.4$

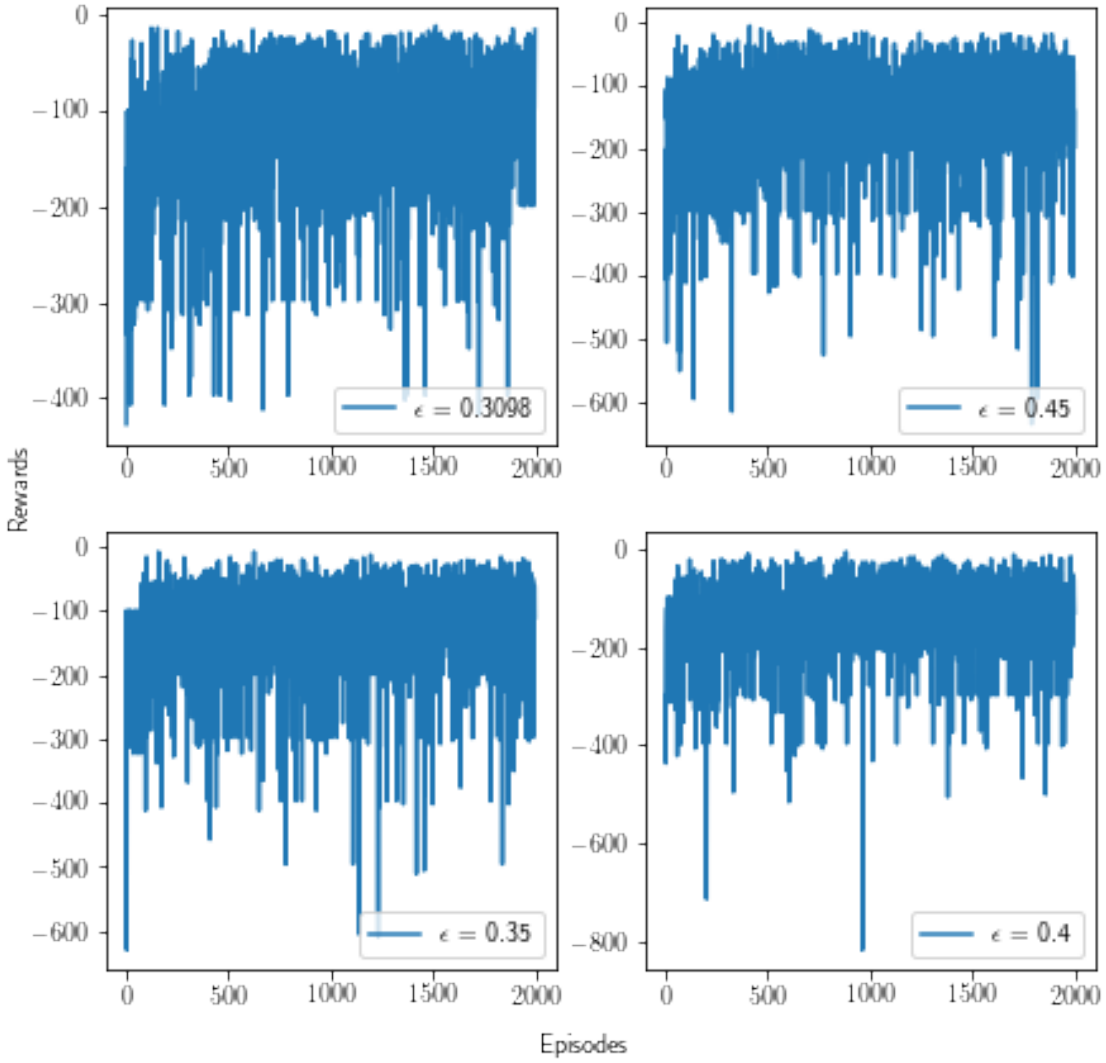

C-11-Q-learning-e-greedy $\gamma = 0.9615$ /$\epsilon = 0.3098$
Rewards
Episodes
$\alpha = 0.1113$
$\alpha = 0.08$
$\alpha = 0.1$
$\alpha = 0.09$

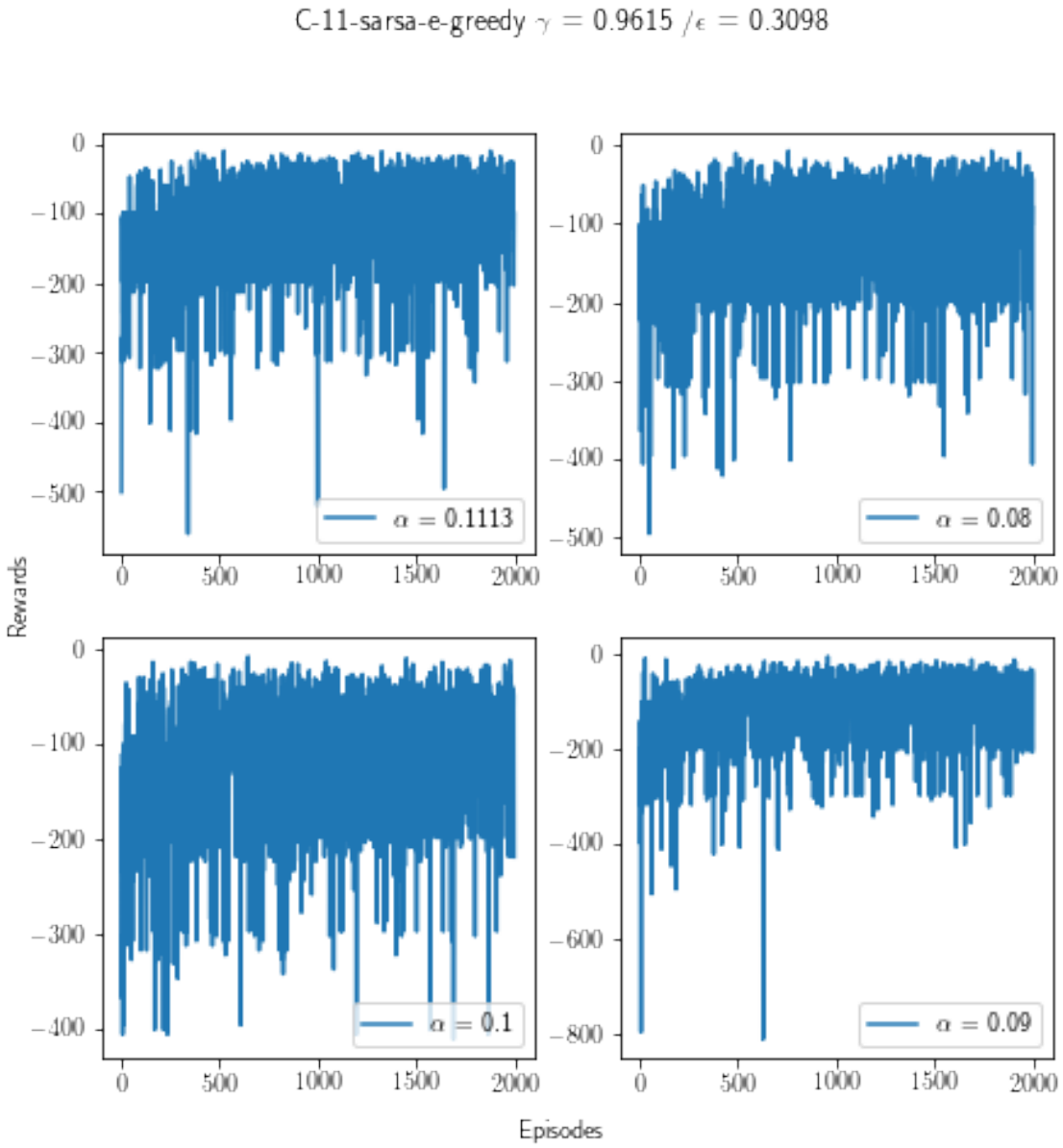

C-11-Q-learning-e-greedy $\alpha = 0.1113$ $\epsilon = 0.3098$
Rewards
Episodes
$\gamma = 0.9615$
$\gamma = 0.9$
$\gamma = 0.95$
$\gamma = 1.0$

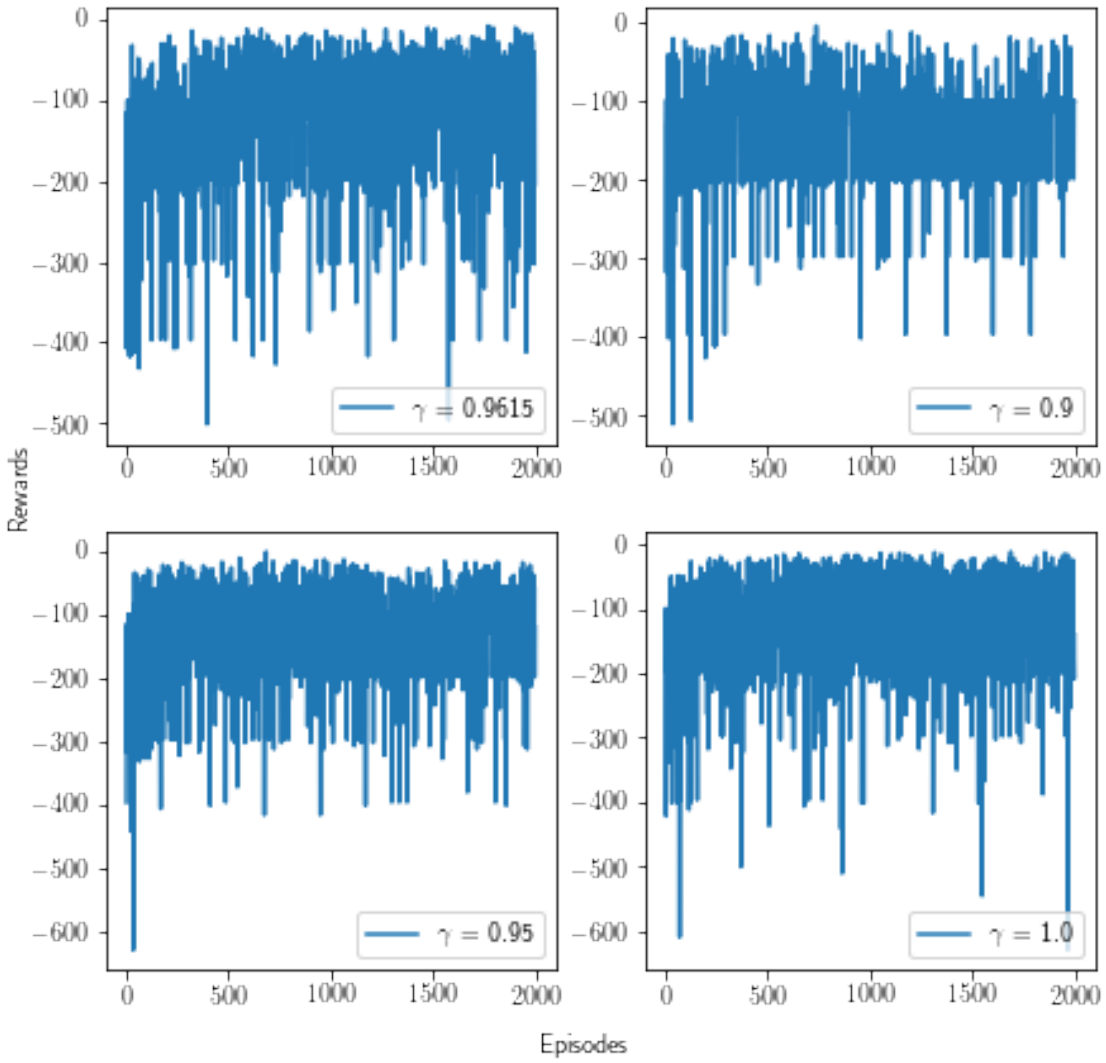

C-11-Q-learning-softmax $\gamma = 0.9615$ $\alpha = 0.1113$
Rewards
Episodes
$\beta = 1.481$
$\beta = 0.55$
$\beta = 1.3$
$\beta = 1.1$

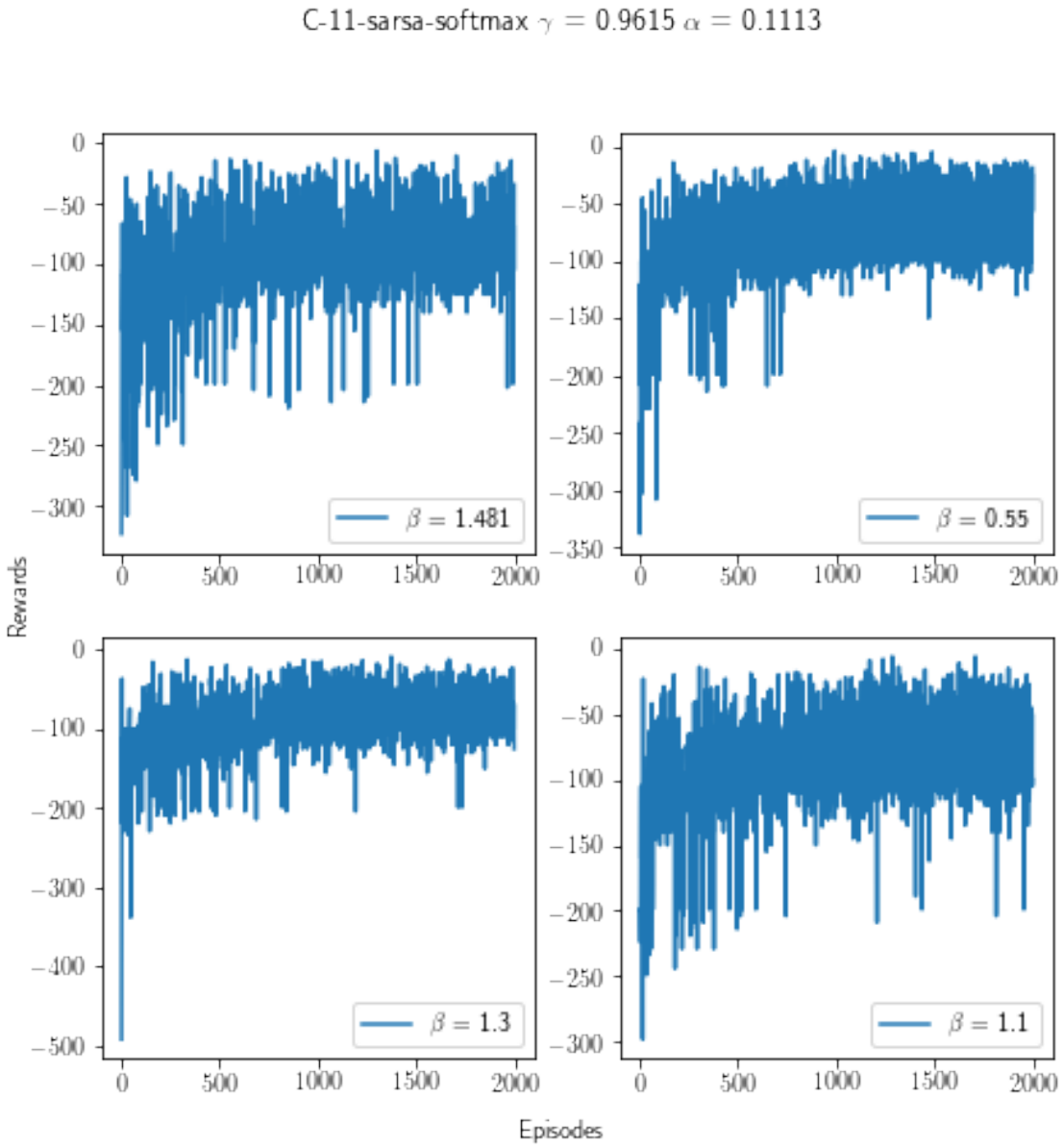

C-11-Q-learning-softmax $\gamma = 0.9615$ $\beta = 1.481$
Rewards
Episodes
$\alpha = 0.1113$
$\alpha = 0.08$
$\alpha = 0.1$
$\alpha = 0.09$

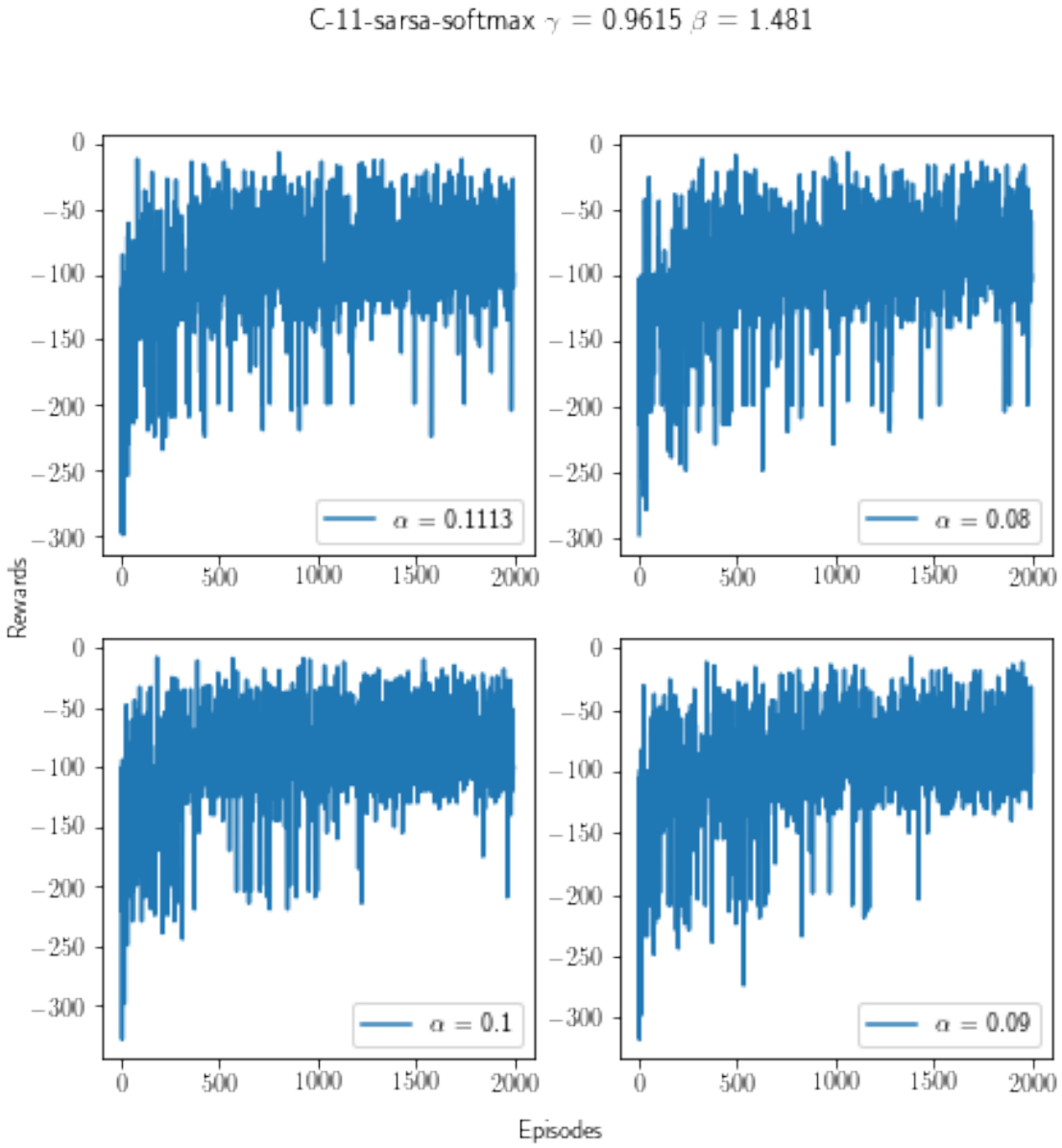

C-11-Q-learning-softmax $\alpha = 0.1113$ $\beta = 1.481$
Rewards
Episodes
$\gamma = 0.9615$
$\gamma = 0.9$
$\gamma = 0.95$
$\gamma = 1.0$

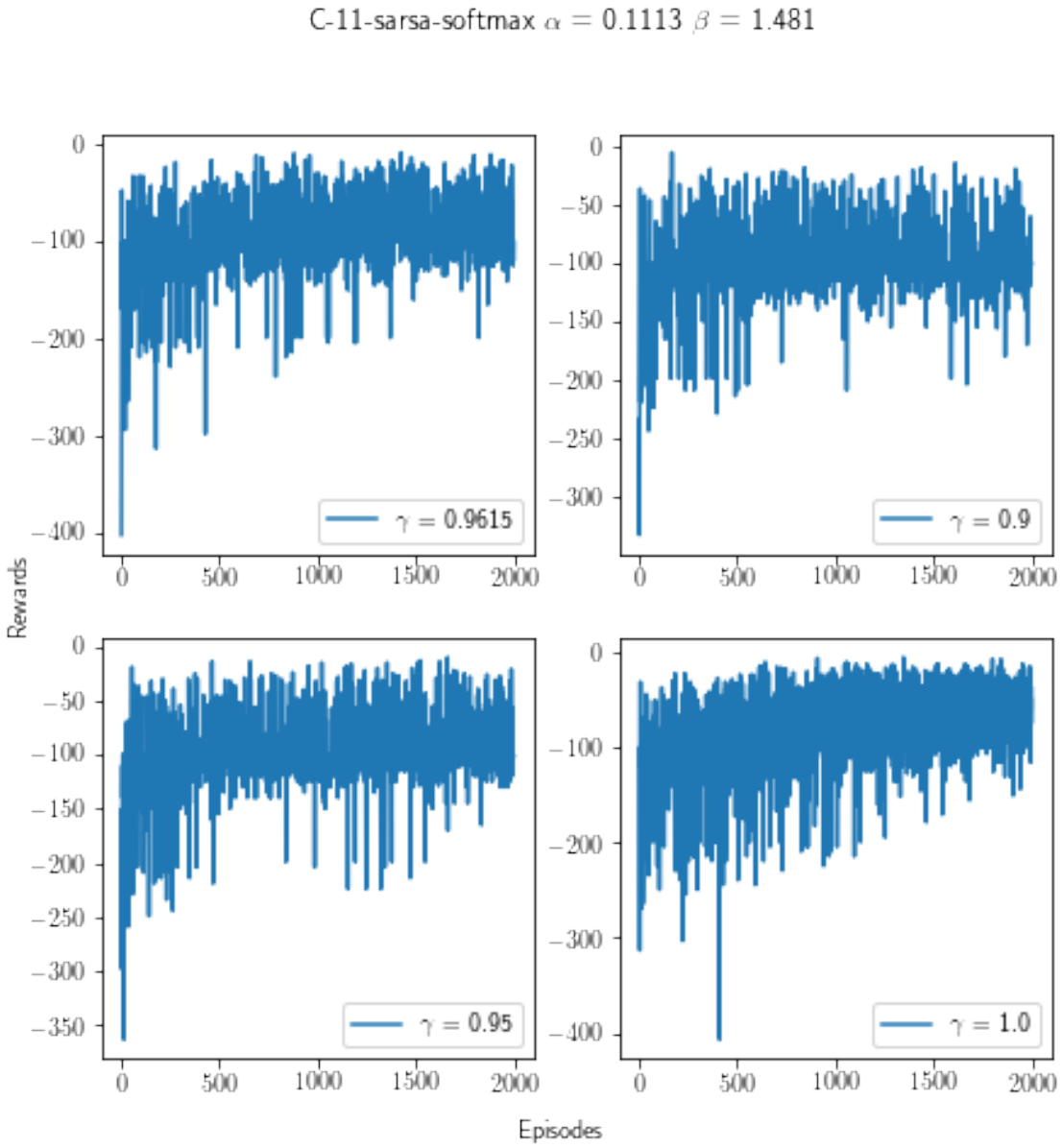

C-11-sarsa-e-greedy $\gamma = 0.9615$ $\alpha = 0.1113$
Rewards
Episodes
$\epsilon = 0.3098$
$\epsilon = 0.45$
$\epsilon = 0.35$
$\epsilon = 0.4$


C-11-sarsa-e-greedy $\gamma = 0.9615$ /$\epsilon = 0.3098$
Rewards
Episodes
$\alpha = 0.1113$
$\alpha = 0.08$
$\alpha = 0.1$
$\alpha = 0.09$


C-11-sarsa-e-greedy $\alpha = 0.1113$ $\epsilon = 0.3098$
Rewards
Episodes
$\gamma = 0.9615$
$\gamma = 0.9$
$\gamma = 0.95$
$\gamma = 1.0$


C-11-sarsa-softmax $\gamma = 0.9615$ $\alpha = 0.1113$
Rewards
Episodes
$\beta = 1.481$
$\beta = 0.55$
$\beta = 1.3$
$\beta = 1.1$

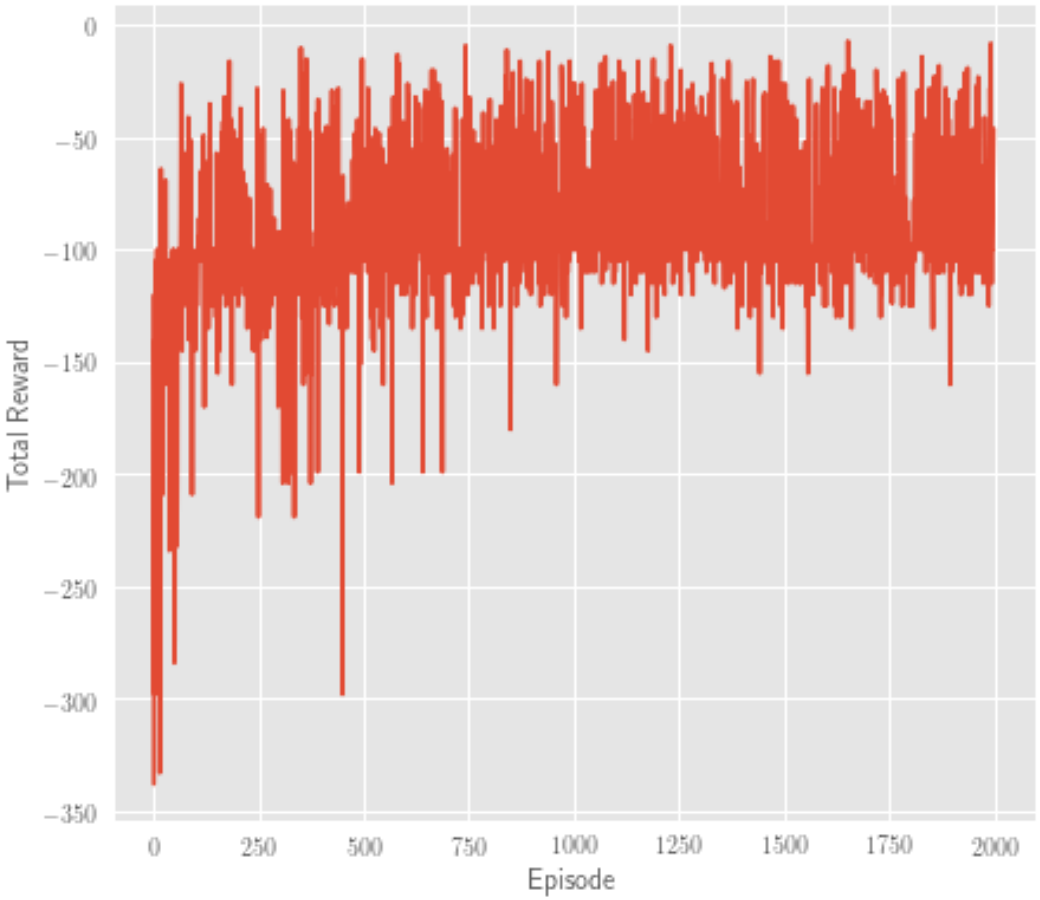

C-11-sarsa-softmax $\gamma = 0.9615$ $\beta = 1.481$
Rewards
Episodes
$\alpha = 0.1113$
$\alpha = 0.08$
$\alpha = 0.1$
$\alpha = 0.09$
0
−50
−100
−150
−200
−250
−300
0
500
1000
1500
2000

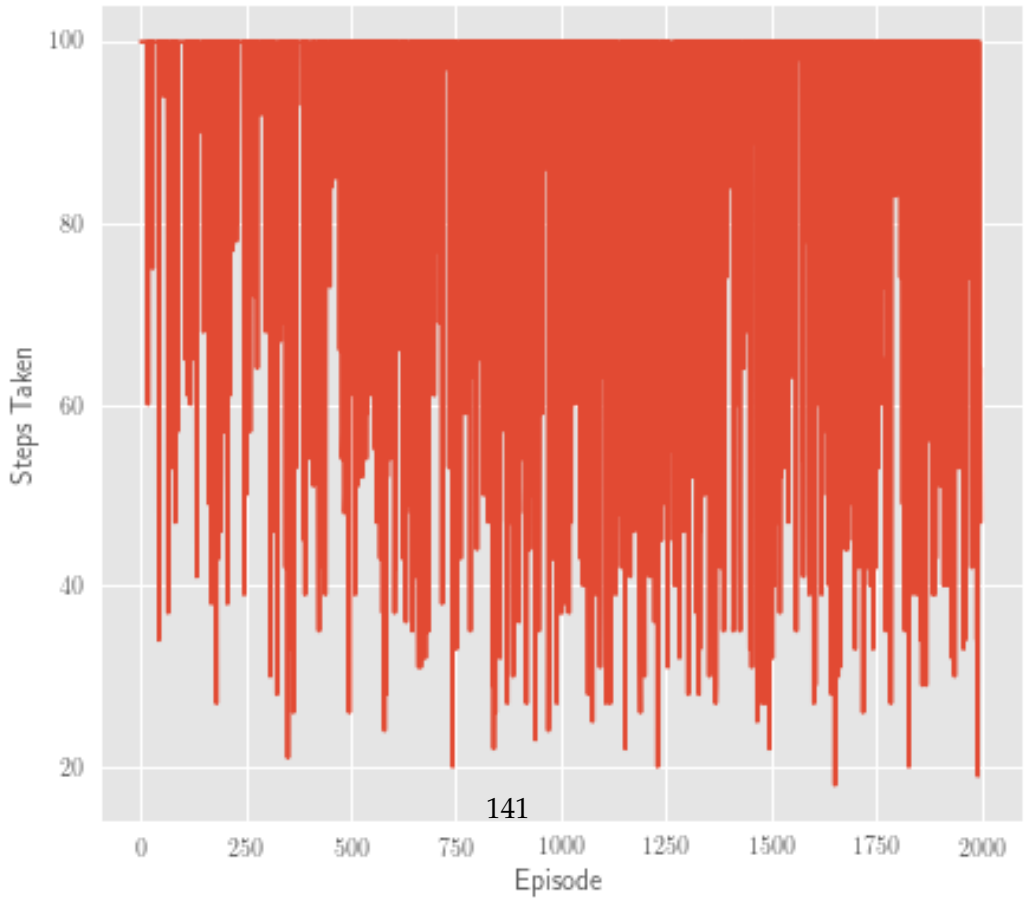


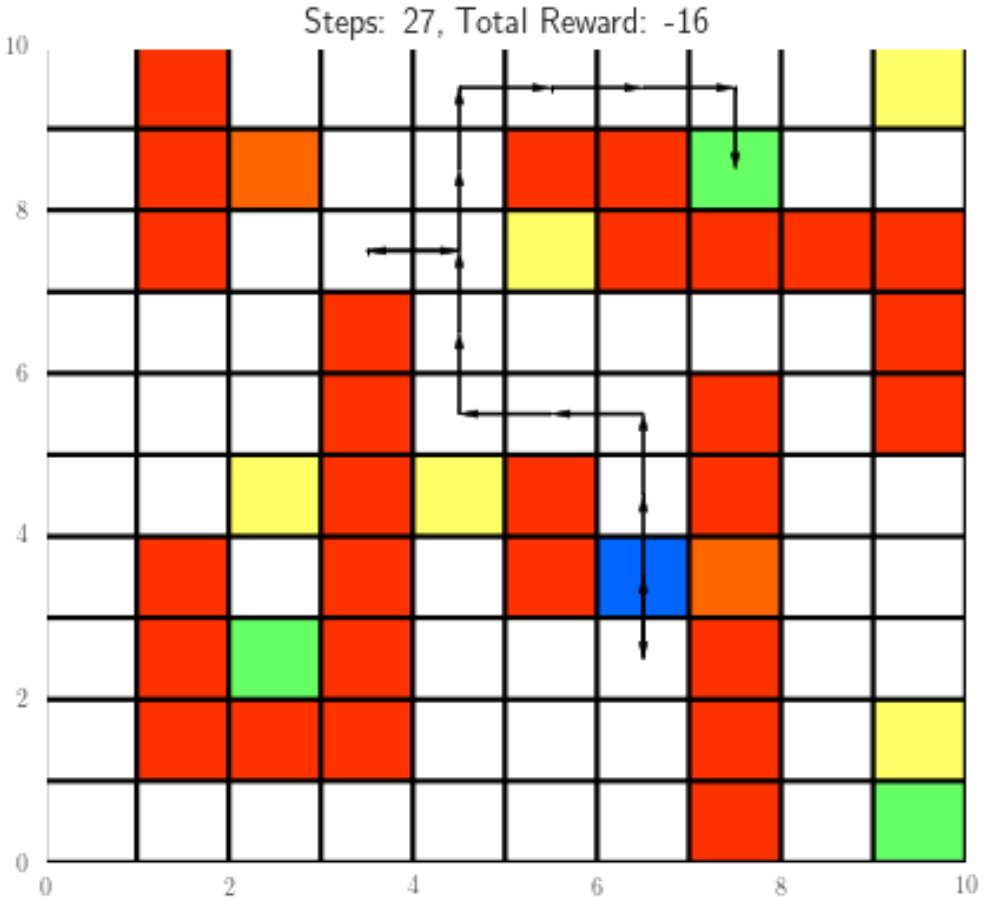
[ ]:
```
policy = 'softmax'
```

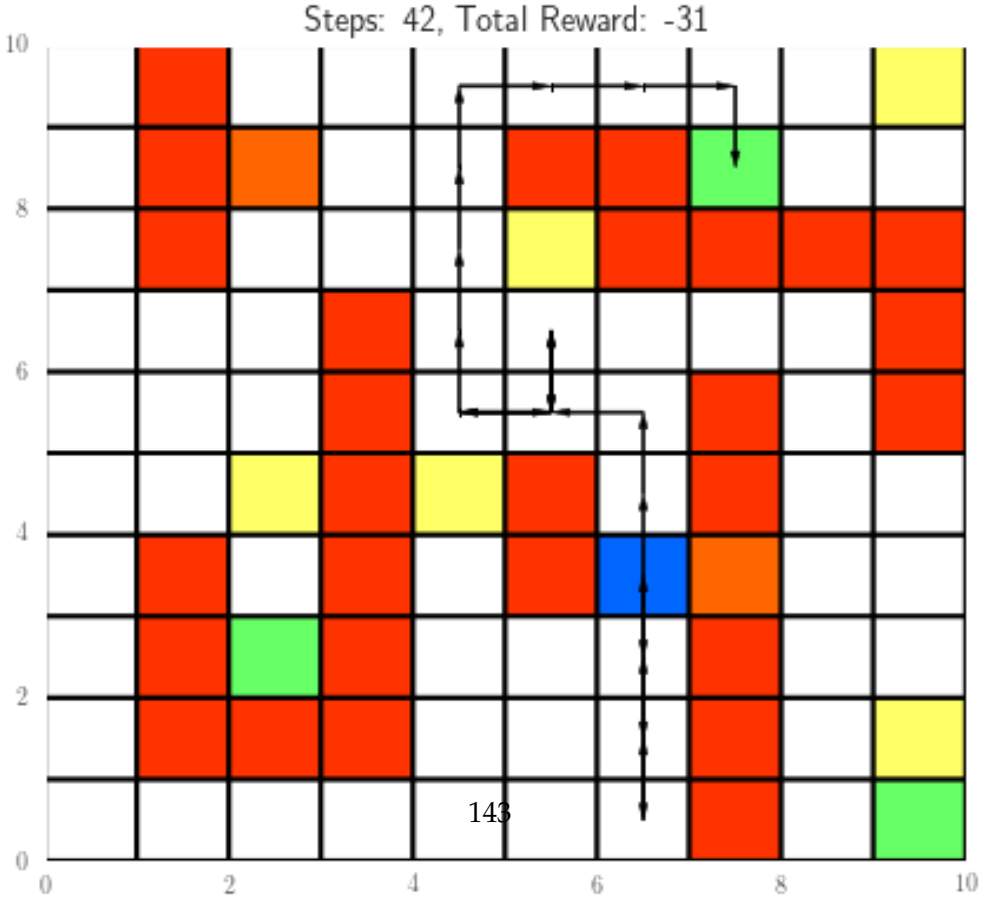
[ ]:
```
Q = best_plots(algorithm, policy, alpha,  epsilon, beta , gamma )
```

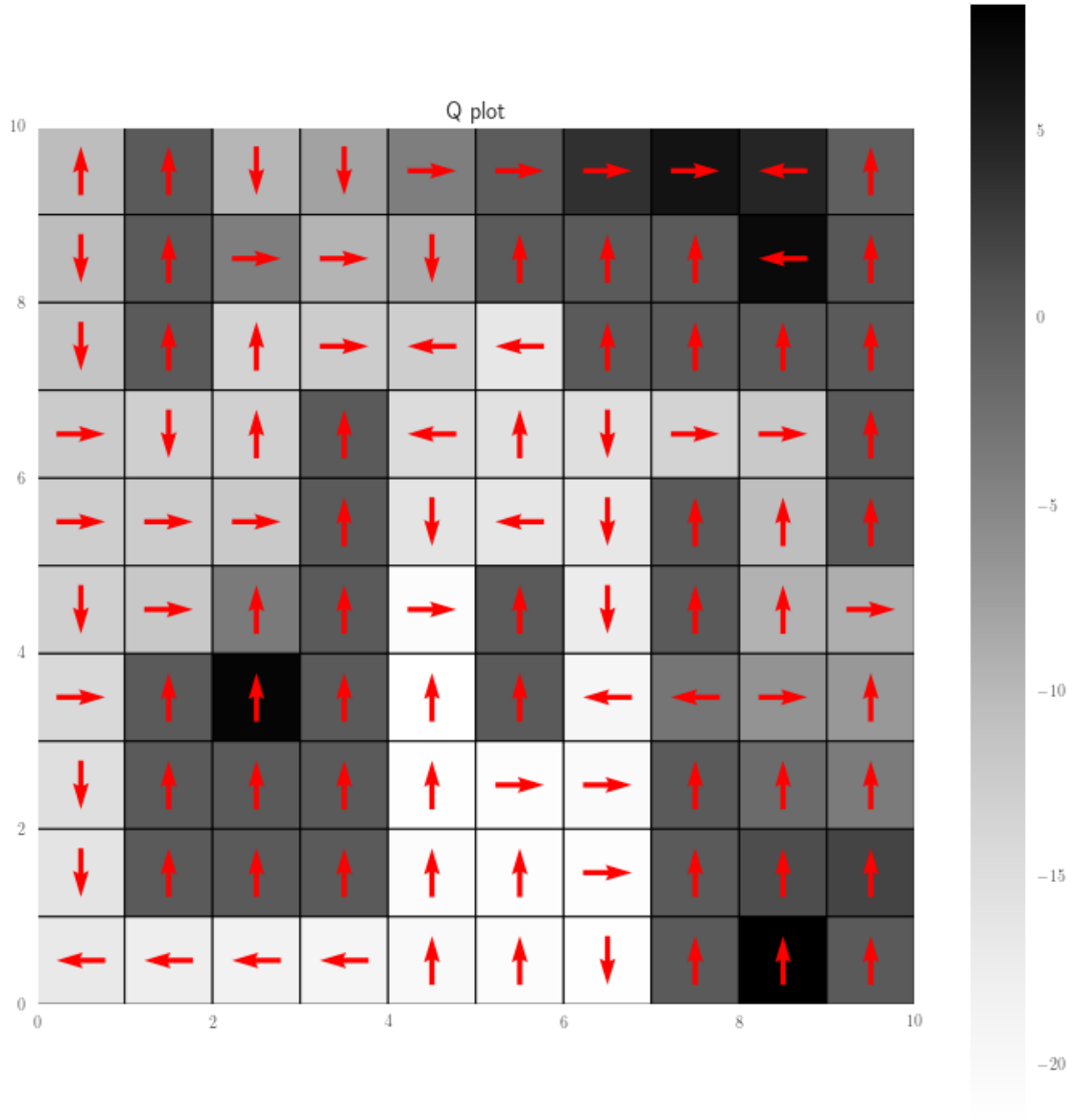
```
100%|| 2000/2000 [01:15<00:00, 26.65it/s]
```


0
−50
−100
−150
−200
−250
−300
−350
Total Reward
0
250
500
750
1000
1250
1500
1750
2000
Episode


100
80
60
40
20
Steps Taken
0
250
500
750
1000
1250
1500
1750
2000
Episode


Steps: 27, Total Reward: -16
10
8
6
4
2
0
0
2
4
6
8
10


Steps: 42, Total Reward: -31
10
8
6
4
2
0
0
2
4
6
8
10



7 Code for downloading notebook (IGNORE)

[28]:
```
!sudo apt-get install texlive-xetex texlive-fonts-recommended␣
↪texlive-plain-generic
```

```
Reading package lists... Done
Building dependency tree
```

```
Reading state information... Done
texlive-fonts-recommended is already the newest version (2017.20180305-1).
texlive-plain-generic is already the newest version (2017.20180305-2).
texlive-plain-generic set to manually installed.
The following NEW packages will be installed:
  texlive-xetex
0 upgraded, 1 newly installed, 0 to remove and 39 not upgraded.
Need to get 10.7 MB of archives.
After this operation, 21.4 MB of additional disk space will be used.
Get:1 http://archive.ubuntu.com/ubuntu bionic/universe amd64 texlive-xetex all
2017.20180305-1 [10.7 MB]
Fetched 10.7 MB in 15s (731 kB/s)
debconf: unable to initialize frontend: Dialog
debconf: (No usable dialog-like program is installed, so the dialog based
frontend cannot be used. at /usr/share/perl5/Debconf/FrontEnd/Dialog.pm line 76,
<> line 1.)
debconf: falling back to frontend: Readline
debconf: unable to initialize frontend: Readline
debconf: (This frontend requires a controlling tty.)
debconf: falling back to frontend: Teletype
dpkg-preconfigure: unable to re-open stdin:
Selecting previously unselected package texlive-xetex.
(Reading database ... 181671 files and directories currently installed.)
Preparing to unpack .../texlive-xetex_2017.20180305-1_all.deb ...
Unpacking texlive-xetex (2017.20180305-1) ...
Setting up texlive-xetex (2017.20180305-1) ...
Processing triggers for tex-common (6.09) ...
debconf: unable to initialize frontend: Dialog
debconf: (No usable dialog-like program is installed, so the dialog based
frontend cannot be used. at /usr/share/perl5/Debconf/FrontEnd/Dialog.pm line
76.)
debconf: falling back to frontend: Readline
Running mktexlsr. This may take some time... done.
Building format(s) --all.
        This may take some time... done.
```

[ ]:
```
!jupyter nbconvert --to pdf /content/drive/MyDrive/Documents/Sem6-drive/RL/
↪Assignments/1Assignment/PA1_working_copy-1.ipynb
```

```
[NbConvertApp] Converting notebook /content/drive/MyDrive/Documents/Sem6-drive/R
L/Assignments/1Assignment/PA1_working_copy-1.ipynb to pdf
[NbConvertApp] Support files will be in PA1_working_copy-1_files/
[NbConvertApp] Making directory ./PA1_working_copy-1_files
[NbConvertApp] Making directory ./PA1_working_copy-1_files
[NbConvertApp] Making directory ./PA1_working_copy-1_files
[NbConvertApp] Making directory ./PA1_working_copy-1_files
[NbConvertApp] Making directory ./PA1_working_copy-1_files
```

```
[NbConvertApp] Making directory ./PA1_working_copy-1_files
[NbConvertApp] Making directory ./PA1_working_copy-1_files
[NbConvertApp] Making directory ./PA1_working_copy-1_files
[NbConvertApp] Making directory ./PA1_working_copy-1_files
[NbConvertApp] Making directory ./PA1_working_copy-1_files
[NbConvertApp] Making directory ./PA1_working_copy-1_files
[NbConvertApp] Making directory ./PA1_working_copy-1_files
[NbConvertApp] Making directory ./PA1_working_copy-1_files
[NbConvertApp] Making directory ./PA1_working_copy-1_files
[NbConvertApp] Making directory ./PA1_working_copy-1_files
[NbConvertApp] Making directory ./PA1_working_copy-1_files
[NbConvertApp] Making directory ./PA1_working_copy-1_files
[NbConvertApp] Making directory ./PA1_working_copy-1_files
[NbConvertApp] Making directory ./PA1_working_copy-1_files
[NbConvertApp] Making directory ./PA1_working_copy-1_files
[NbConvertApp] Making directory ./PA1_working_copy-1_files
[NbConvertApp] Making directory ./PA1_working_copy-1_files
[NbConvertApp] Making directory ./PA1_working_copy-1_files
[NbConvertApp] Making directory ./PA1_working_copy-1_files
[NbConvertApp] Making directory ./PA1_working_copy-1_files
[NbConvertApp] Making directory ./PA1_working_copy-1_files
[NbConvertApp] Making directory ./PA1_working_copy-1_files
[NbConvertApp] Making directory ./PA1_working_copy-1_files
[NbConvertApp] Making directory ./PA1_working_copy-1_files
[NbConvertApp] Making directory ./PA1_working_copy-1_files
[NbConvertApp] Making directory ./PA1_working_copy-1_files
[NbConvertApp] Making directory ./PA1_working_copy-1_files
[NbConvertApp] Making directory ./PA1_working_copy-1_files
[NbConvertApp] Making directory ./PA1_working_copy-1_files
[NbConvertApp] Making directory ./PA1_working_copy-1_files
[NbConvertApp] Making directory ./PA1_working_copy-1_files
[NbConvertApp] Making directory ./PA1_working_copy-1_files
[NbConvertApp] Making directory ./PA1_working_copy-1_files
[NbConvertApp] Making directory ./PA1_working_copy-1_files
[NbConvertApp] Making directory ./PA1_working_copy-1_files
[NbConvertApp] Making directory ./PA1_working_copy-1_files
[NbConvertApp] Making directory ./PA1_working_copy-1_files
[NbConvertApp] Making directory ./PA1_working_copy-1_files
[NbConvertApp] Making directory ./PA1_working_copy-1_files
[NbConvertApp] Making directory ./PA1_working_copy-1_files
[NbConvertApp] Making directory ./PA1_working_copy-1_files
[NbConvertApp] Making directory ./PA1_working_copy-1_files
[NbConvertApp] Making directory ./PA1_working_copy-1_files
[NbConvertApp] Making directory ./PA1_working_copy-1_files
[NbConvertApp] Making directory ./PA1_working_copy-1_files
[NbConvertApp] Making directory ./PA1_working_copy-1_files
[NbConvertApp] Making directory ./PA1_working_copy-1_files
[NbConvertApp] Making directory ./PA1_working_copy-1_files
```

```
[NbConvertApp] Making directory ./PA1_working_copy-1_files
[NbConvertApp] Making directory ./PA1_working_copy-1_files
[NbConvertApp] Making directory ./PA1_working_copy-1_files
[NbConvertApp] Making directory ./PA1_working_copy-1_files
[NbConvertApp] Making directory ./PA1_working_copy-1_files
[NbConvertApp] Making directory ./PA1_working_copy-1_files
[NbConvertApp] Making directory ./PA1_working_copy-1_files
[NbConvertApp] Making directory ./PA1_working_copy-1_files
[NbConvertApp] Making directory ./PA1_working_copy-1_files
[NbConvertApp] Making directory ./PA1_working_copy-1_files
[NbConvertApp] Making directory ./PA1_working_copy-1_files
[NbConvertApp] Making directory ./PA1_working_copy-1_files
[NbConvertApp] Making directory ./PA1_working_copy-1_files
[NbConvertApp] Making directory ./PA1_working_copy-1_files
[NbConvertApp] Making directory ./PA1_working_copy-1_files
[NbConvertApp] Making directory ./PA1_working_copy-1_files
[NbConvertApp] Making directory ./PA1_working_copy-1_files
[NbConvertApp] Making directory ./PA1_working_copy-1_files
[NbConvertApp] Making directory ./PA1_working_copy-1_files
[NbConvertApp] Writing 243299 bytes to ./notebook.tex
[NbConvertApp] Building PDF
[NbConvertApp] Running xelatex 3 times: [u'xelatex', u'./notebook.tex',
'-quiet']
[NbConvertApp] Running bibtex 1 time: [u'bibtex', u'./notebook']
[NbConvertApp] WARNING | bibtex had problems, most likely because there were no
citations
[NbConvertApp] PDF successfully created
[NbConvertApp] Writing 2299907 bytes to /content/drive/MyDrive/Documents/Sem6-dr
ive/RL/Assignments/1Assignment/PA1_working_copy-1.pdf
```

[ ]:

```
```