RL_Project_code_19023 & 19024

May 13, 2022

0.1 Alli Khadga Jyoth - 19024 DSE

0.2 Akash Kumar Singh - 19023 DSE

[1]:
```
from gym import Env
import gym
import pygame
from gym.spaces import Discrete, Box,Dict
import numpy as np
import random
import matplotlib.pyplot as plt
from IPython.display import clear_output
np.set_printoptions(linewidth=500,threshold=np.inf)
```

0.3 ATC Environment

[2]:
```
class AirTraffic():
    def __init__(self,planes:int = 2, grid_size:list = [5,5],radius:int =␣
↪2,seed = None,destinations:int =None ):
        self.radius = radius
        self.planes = planes
        self.destinations = destinations
        self.GRID_DIM = grid_size
        self.seed = seed
        self._max_dist = np.sqrt(self.GRID_DIM[0]**2 + self.GRID_DIM[1]**2)
        self._action_to_direction = {
            0: np.array([-1, 0]),
            1: np.array([-1, 1]),
            2: np.array([0,  1]),
            3: np.array([-1 , -1]),
            4: np.array([0 , -1]),

        }

        self._ACTIONLOOKUP = {
            0: 'Up',
            1: 'front right',
            2: 'right',
```

```
            3: 'front left',
            4: 'left'
            }
        self.GRID_DIM = np.array(self.GRID_DIM)
        self.GRID = np.zeros(self.GRID_DIM)
        self.action_space = Discrete(len(self._ACTIONLOOKUP.keys()))
        self.state_space = Discrete(self.GRID_DIM[0]*self.GRID_DIM[1])
        self._ob_space = {}
        for p in range(self.planes):
            self._ob_space[f'plane{p}'] = Box(np.array([0,0]), np.array([self.
↪GRID_DIM[0]-1,self.GRID_DIM[1] - 1]), shape=(2,), dtype=int)
            self._ob_space[f'dest{p}'] = Box(np.array([0,0]), np.array([self.
↪GRID_DIM[0]-1,self.GRID_DIM[1] - 1]), shape=(2,), dtype=int)

        self.observation_space = Dict(self._ob_space)
        self._agent_location = []
        self._target_location = []
        # Randomly initialize the start points of planes and destinations
        if self.seed is not None:
            np.random.seed(seed = seed)
        for p in range(self.planes):
            while (pc:= [np.random.randint(self.GRID_DIM[0]),np.random.
↪randint(self.GRID_DIM[1])]) in self._agent_location:
                continue
            self._agent_location.append(pc)
            if self.destinations is None:
                while (dc:=[np.random.randint(self.GRID_DIM[0]),np.random.
↪randint(self.GRID_DIM[1])]) in self._agent_location + self._target_location:
                    continue
                self._target_location.append(dc)
        if self.destinations is not None:
            for d in range(self.destinations):
                while (dc:=[np.random.randint(self.GRID_DIM[0]),np.random.
↪randint(self.GRID_DIM[1])]) in self._agent_location + self._target_location:
                    continue
                self._target_location.append(dc)

            extra = np.random.choice(range(len(self._target_location)),self.
↪planes-self.destinations)
            for d in extra:
                self._target_location.append(self._target_location[d])
        self._agent_location = np.array(self._agent_location)
        self._target_location = np.array(self._target_location)
        self._count = [0]*self.planes # to check if its already landed/reached␣
↪destination
        self._done_prev = [False]*self.planes
```

```
        self._prev_agent_location = self._agent_location.copy()


    def step(self,action):
        self._prev_agent_location = self._agent_location.copy()##### Find the␣
↪vector of agent from destination to find ### reward of terminal state for a␣
↪plane is 0 and that plane cant move now
        moved_plane = [False]*self.planes
        moved_plane = self._move(action)
        done,reward =  self._is_over()
        observation = self._get_obs()
        info = self._get_info()
        self._done_prev = done.copy()
        return observation, reward, done, info
    def _move(self, action):
         ##### Find the vector of agent from destination to find
        ### reward of terminal state for a plane is _todestination and that␣
↪plane cant move now
        self._prev_agent_location = self._agent_location
        for plane,act in enumerate(action):
            if np.array_equal(self._agent_location[plane],self.
↪_target_location[plane]):
                self._count[plane]+=1
                continue
            elif np.sign(self._target_location[plane]-self.
↪_prev_agent_location[plane])[0]<=0:
                self._agent_location[plane] = np.clip(self.
↪_agent_location[plane] + self._action_to_direction[act],[0,0],[self.
↪GRID_DIM[0] - 1, self.GRID_DIM[1] - 1])
            else:
                self._agent_location[plane] = np.clip(self.
↪_agent_location[plane] + -1*self._action_to_direction[act],[0,0],[self.
↪GRID_DIM[0] - 1, self.GRID_DIM[1] - 1])

        return   self._prev_agent_location== self._agent_location
    def _is_over(self):
        done = (self._agent_location==self._target_location).all(axis = 1)
        reward = self._get_reward()
        return done, reward
    def _get_reward(self):
        reward = []
        # reward will be  intruder + todestination
        distance, closest_dist ,_1,_2= self._get_info()
        for index,plane in enumerate(range(self.planes)):
            if self._count[plane] >0:
                reward.append(self._max_dist)## terminating state reward
```

```
            else:
                self._intruder = 0
                if closest_dist[plane] <self.radius:

                    self._intruder = -(self.radius **2 -␣
↪closest_dist[plane]**2)/(self.radius**2/500)
                self._todestination = self._max_dist - distance[plane]
                reward.append(self._todestination + self._intruder)
        return reward

    def _get_obs(self):
        return {'planes': self._agent_location,'destinations':self.
↪_target_location}

    def _get_info(self):
            closest_dist = []
            distance = []
            plane_theta = []
            plane_rho = []
            dne= self._done_prev
            for index,plane in enumerate(range(self.planes)):
                dist = np.linalg.norm([self._agent_location[plane]]*self.
↪planes-self._agent_location,axis = 1)
                dist[plane] = np.inf
                dist[dne] = np.inf
                if dne[plane]:
                    dist += np.inf
                    plane_rho.append(np.inf)
                    plane_theta.append(np.inf)
                    closest_dist .append(np.min(dist))
                else:
                    closest_dist .append(np.min(dist))
                    arg_plane = np.argmin(dist)

                    if (self._prev_agent_location==None).all():
                        self._prev_agent_location = self._agent_location.copy()
                    ownship = self._agent_location[plane]-self.
↪_prev_agent_location[plane]
                    intruder = self._agent_location[arg_plane]-self.
↪_prev_agent_location[arg_plane]
                    closest_path = self._agent_location[arg_plane]-self.
↪_agent_location[plane]
                    y = np.array([intruder[1],ownship[1]])
                    x = np.array([intruder[0],ownship[0]])
                    degree = np.rad2deg(np.arctan2(y,x))
                    deg = degree[1]-degree[0]
```

```
                    alpha = np.rad2deg(np.arctan2(ownship[1],ownship[1]))
                    beta =  np.rad2deg(np.
↪arctan2(closest_path[1],closest_path[0]))
                    deg_rho =  (alpha-beta)
                    plane_rho.append(deg_rho)
                    plane_theta.append( deg)
                distance .append(np.linalg.norm(self.
↪_target_location[plane]-self._agent_location[plane]))
            return distance,closest_dist,plane_theta,plane_rho

    def _did_collide(self):
        distance,closest_dist,plane_theta,plane_rho = self._get_info()
        collide =  np.array(closest_dist).all()
        return collide

    def _state_to_index(self):
        _,closest_dist,plane_theta,plane_rho = self._get_info()
        dis_bin = np.arange(self.radius)
        angle_bins = np.arange(-180,180,10)
        dist_index = np.digitize(closest_dist,dis_bin) -1
        theta_index = np.digitize(plane_theta,angle_bins) -1
        rho_index = np.digitize(plane_rho,angle_bins) -1
        return dist_index,theta_index,rho_index

    def _state_in_seq(self):
        seq = []
        for plane in self._agent_location:
            m, n = plane
            seq.append(m * self.GRID.shape[1] + n)
        return seq

    def render(self):
        rend = self.GRID.copy().astype(dtype = 'U2')
        for plane in range(self.planes):
            rend[self._agent_location[plane][0],self._agent_location[plane][1]]␣
↪= f'p{plane}'
            rend[self._target_location[plane][0],self.
↪_target_location[plane][1]] = f'd{plane}'
        return print(rend)

    def reset(self):
        self._agent_location = []
        self._target_location = []
        if self.seed is not None:
            np.random.seed(seed = self.seed)
        for p in range(self.planes):
```

```
            while (pc:= [np.random.randint(self.GRID_DIM[0]),np.random.
↪randint(self.GRID_DIM[1])]) in self._agent_location:
                continue
            self._agent_location.append(pc)
            while (dc:=[np.random.randint(self.GRID_DIM[0]),np.random.
↪randint(self.GRID_DIM[1])]) in self._agent_location + self._target_location:
                continue
            self._target_location.append(dc)
        self._agent_location = np.array(self._agent_location)
        self._target_location = np.array(self._target_location)
        self._count = [0]*self.planes
```

0.4 Random Policy Agent

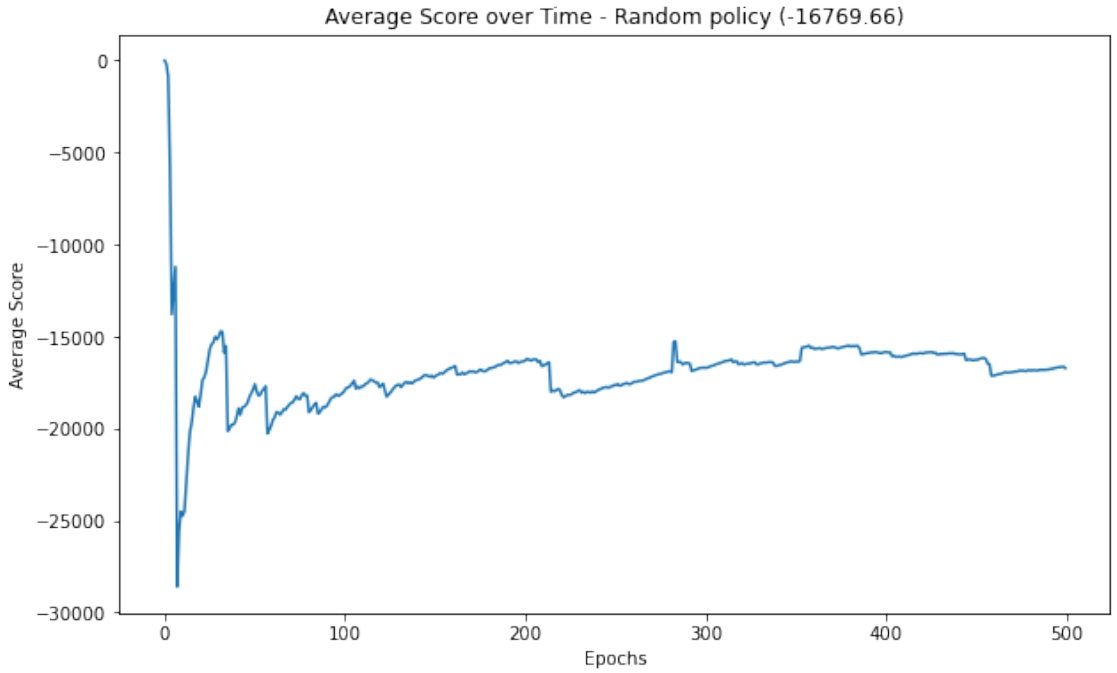
[14]:
```
def random_policy(env,episodes):
    action_space ={
        0: np.array([-1, 0]),
        1: np.array([-1, 1]),
        2: np.array([0,  1]),
        3: np.array([-1 , -1]),
        4: np.array([0 , -1]),
    }
    timestep_reward = []
    cumsum_reward  = []
    cumsum_ep= []
    for ep in range(episodes):
        env.reset()
        done =  [False]*env.planes
        total_reward = [0]*env.planes
        t=0
        while (not np.array(done).all()) and env._did_collide() and t<1e6:

            action = []
            distance,closest_dist,plane_theta,plane_rh = env._get_info()
            for plane in range(env.planes):
                if closest_dist[plane]< env.radius:
                    action.append(np.random.randint(env.action_space.n))
                else:
                    if np.array_equal(env._target_location[plane],env.
↪_agent_location[plane]):
                        action.append(0)
                    else:
                        x = np.sign(env._target_location[plane] - env.
↪_agent_location[plane])
                        if x[0] in [0,-1]:
                            for act,a in enumerate(action_space.values()):
                                _ = np.array_equal(x,a)
```

```
                                if _:
    #                                 print(_)
                                    action.append(act)
                                    break
                            else:
                                x = -x
                                for act,a in enumerate(action_space.values()):
                                    _ = np.array_equal(x,a)
                                    if _:
                                        action.append(act)
                                        break
                    t+=1
                    obs,reward,done,info  = env.step(action)
                    total_reward= np.add(total_reward,reward)
                    print(t,ep,end=' ')
                    clear_output(wait=True)
                cumsum_reward.append(total_reward/t)
                timestep_reward.append(np.average(total_reward))
                cumsum_ep.append(np.average(total_reward)/(ep+1))
            clear_output()
            return timestep_reward,cumsum_reward,cumsum_ep
```

[15]:
```
env = AirTraffic(planes=25,grid_size=[100,100],radius=50,destinations=1)
episodes = 500
timestep_reward_rand,cumsum_reward_rand,cumsum_ep_rand =␣
 ↪random_policy(env,episodes)
```

[16]:
```
rewd_rand = []
rew = 0
for i,j in enumerate(timestep_reward_rand):
    rew = sum(timestep_reward_rand[0:i])/(i+1)
    rewd_rand.append(rew)
plt.figure(figsize = (10,6))
plt.title(label=f'Average Score over Time - Random policy ({np.
 ↪average(rewd_rand[-50:]) :.2f})')
plt.plot(rewd_rand)
plt.xlabel('Epochs')
plt.ylabel(f'Average Score')
```

[16]:
```
Text(0, 0.5, 'Average Score')
```



0.5 Greedy Policy Agent

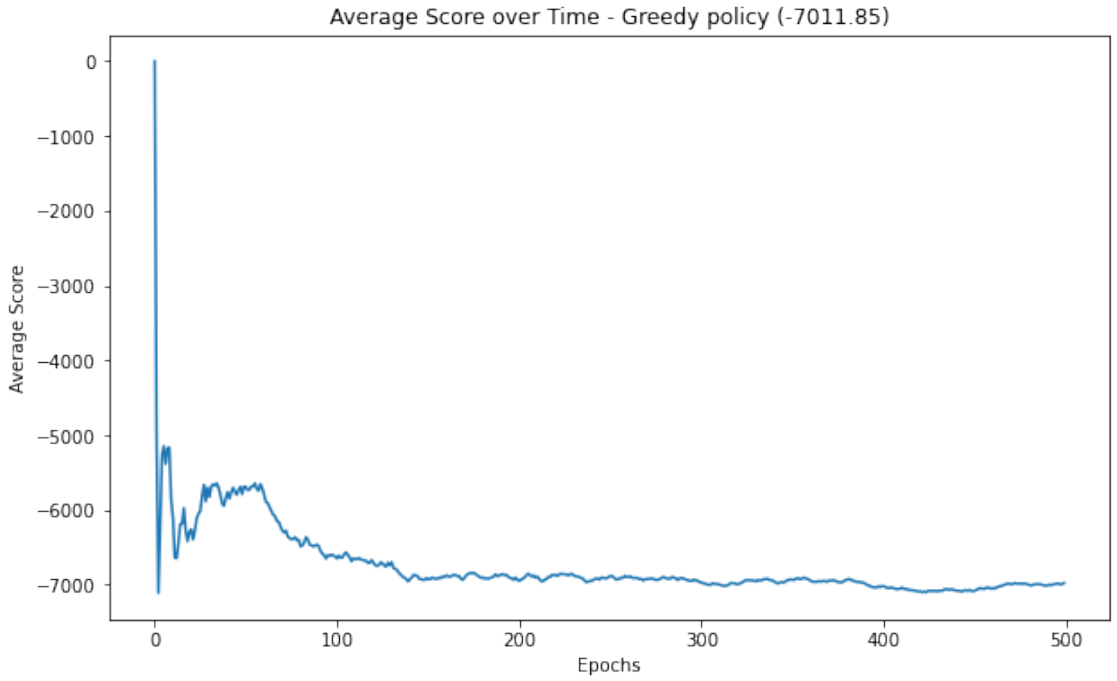
[18]:
```
def greedy_policy(env,episodes):
    action_space ={
        0: np.array([-1, 0]),
        1: np.array([-1, 1]),
        2: np.array([0,  1]),
        3: np.array([-1 , -1]),
        4: np.array([0 , -1]),
    }
    timestep_reward = []
    cumsum_reward  = []
    cumsum_ep= []
    max_step=1e6
    for ep in range(episodes):
        env.reset()
#         print(env._get_obs())
        done =  [False]*env.planes
        total_reward = [0]*env.planes
        t=0
        while (not np.array(done).all()) and env._did_collide() and t<max_step:

            action = []
            for plane in range(env.planes):
```

```
                x = np.sign(env._target_location[plane] - env.
↪_agent_location[plane])
                if x[0] in [0,-1]:
                    for act,a in enumerate(action_space.values()):
                        _ = np.array_equal(x,a)
                        if _:
   #                                        print(_)
                            action.append(act)
                            break
                    else:
                        action.append(0)
                else:
                    x = -x
                    for act,a in enumerate(action_space.values()):
                        _ = np.array_equal(x,a)
                        if _:
                            action.append(act)
                            break
            t+=1
            obs,reward,done,info  = env.step(action)
            total_reward= np.add(total_reward,reward)
            print(t,ep,end=' ')
            clear_output(wait=True)
        cumsum_reward.append(total_reward/t)
        timestep_reward.append(np.average(total_reward))
        cumsum_ep.append(np.average(total_reward)/(ep+1))
        clear_output()
    return timestep_reward,cumsum_reward,cumsum_ep
```

[19]:
```
env = AirTraffic(planes=25,grid_size=[100,100],radius=50,destinations=1)
episodes = 500
timestep_reward_greed,cumsum_reward_greedy,cumsum_ep_greedy =␣
↪greedy_policy(env,episodes)
```

[20]:
```
rewd_greed = []
rew = 0
for i,j in enumerate(timestep_reward_greed):
    rew = sum(timestep_reward_greed[0:i])/(i+1)
    rewd_greed.append(rew)
plt.figure(figsize = (10,6))
plt.title(label=f'Average Score over Time - Greedy policy ({np.
↪average(rewd_greed[-50:]) :.2f})')
plt.plot(rewd_greed)
plt.xlabel('Epochs')
plt.ylabel(f'Average Score')
```

[20]:
```
Text(0, 0.5, 'Average Score')
```



0.6 Greedy + SARSA

[3]:
```
def Sarsa_policy(env,Q,epsilon,count_state=None):
    action_space ={
        0: np.array([-1, 0]),
        1: np.array([-1, 1]),
        2: np.array([0,  1]),
        3: np.array([-1 , -1]),
        4: np.array([0 , -1]),
    }

    action = []
    Plane = []
    distance,closest_dist,plane_theta,plane_rh = env._get_info()
    dist_index,theta_index,rho_index = env._state_to_index()
#     print(closest_dist)
    for plane in range(env.planes):

        if closest_dist[plane]<= env.radius:
            if np.random.random()>epsilon:
                action.append(np.
 ↪argmax(Q[dist_index[plane],theta_index[plane],rho_index[plane],:]))
            else:
                action.append(np.random.randint(env.action_space.n))
```

```
            Plane.append(plane)
        else:
            if np.array_equal(env._target_location[plane],env.
 ↪_agent_location[plane]):
                action.append(0)
            else:
                x = np.sign(env._target_location[plane] - env.
 ↪_agent_location[plane])
                if x[0] in [0,-1]:
                    for act,a in enumerate(action_space.values()):
                        _ = np.array_equal(x,a)
                        if _:
    #                                       print(_)
                            action.append(act)
                            break
                else:
                    x = -x
                    for act,a in enumerate(action_space.values()):
                        _ = np.array_equal(x,a)
                        if _:
                            action.append(act)
                            break
#     print(Plane)
    return action, Plane

def greedy_Sarsa(env,Q,alpha, gamma, epsilon, episodes,temperature=None):

    cumsum_reward  = []
    count_state = np.zeros_like(Q)
    Q_temp = np.zeros_like(Q)

    for ep in range(episodes):
        env.reset()
        done = [False]*env.planes
        total_reward = [0]*env.planes
        curr_dist_index,curr_theta_index,curr_rho_index = env._state_to_index()
        curr_a,curr_Planes =␣
 ↪Sarsa_policy(env,Q,epsilon=epsilon,count_state=count_state)
        t = 0
        while (not np.array(done).all()) and env._did_collide() and t<1.2e4:
            obs, reward, done, info = env.step(curr_a)
            nxt_dist_id,nxt_theta_id,nxt_rho_id = env._state_to_index()
            total_reward= np.add(total_reward,reward)
            next_act,nxt_Planes = Sarsa_policy(env,Q,epsilon,count_state=None)
            t+=1
#             print(curr_a)
            if len(curr_Planes) != 0:
```

```
                for plane in curr_Planes:
                    ␣
 ↪Q[curr_dist_index[plane],curr_theta_index[plane],curr_rho_index[plane],curr_a[plane]]␣
 ↪+= alpha * ( reward[plane] + (gamma *␣
 ↪Q[nxt_dist_id[plane],nxt_theta_id[plane],nxt_rho_id[plane], next_act[plane]])
                                                                                ␣
 ↪                                          ¯␣
 ↪Q[curr_dist_index[plane],curr_theta_index[plane],curr_rho_index[plane],curr_a[plane]])
            curr_dist_index,curr_theta_index,curr_rho_index =␣
 ↪nxt_dist_id,nxt_theta_id,nxt_rho_id
            curr_a = next_act
            curr_Planes = nxt_Planes
            print(t,ep,end=' ')
            clear_output(wait=True)
#          env.render()
        cumsum_reward.append(total_reward/t)
        timestep_reward.append(np.average(total_reward))
        cumsum_ep.append(np.average(total_reward)/(ep+1))
        if ep%25 ==0:
            Q_temp = Q.copy()
        if temperature is not None and ep%temperature==0:
            epsilon = 0.9*epsilon
            print(epsilon)

    clear_output()
    return timestep_reward,cumsum_reward,cumsum_ep
```

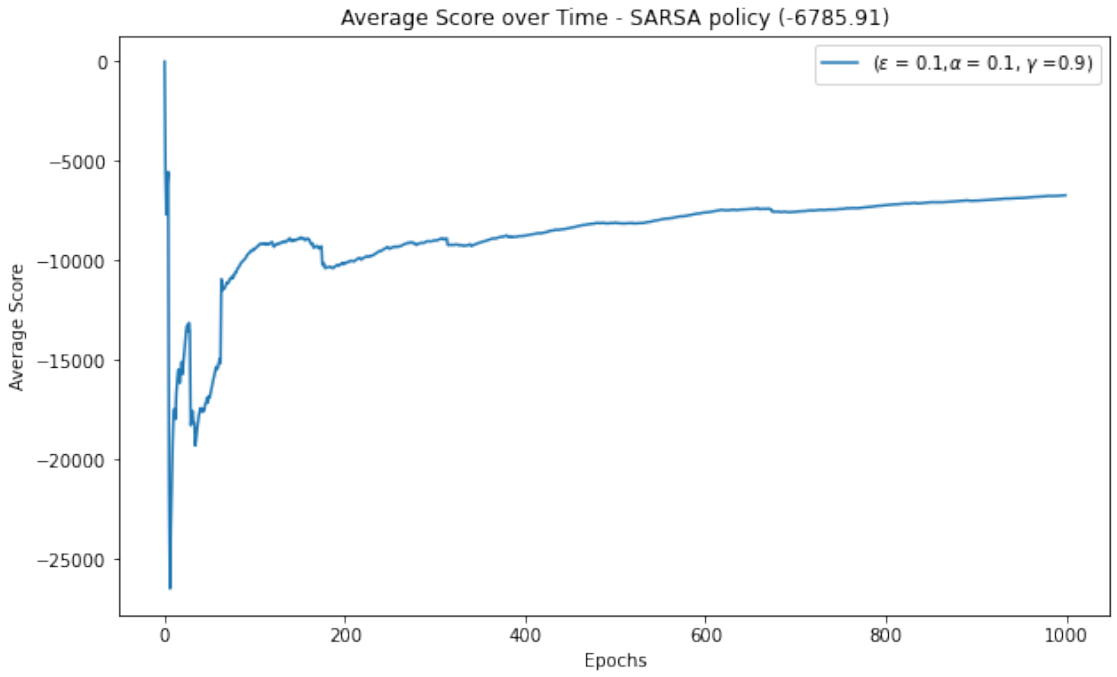
0.6.1 Constant Temperature SARSA

[22]:
```
env = AirTraffic(planes=25,grid_size=[100,100],radius=50,destinations=1)
n_radius,n_theta, n_rho,n_actions = env.radius, 36,36, env.action_space.n
Q_sarsa = np.zeros((n_radius,n_theta, n_rho,n_actions))
alpha = 0.1
gamma = 0.9
epsilon = 0.1
episodes = 1000
cumsum_ep= []
timestep_reward = []
timestep_reward_sarsa,cumsum_reward,cumsum_greedy_sarsa =␣
 ↪greedy_Sarsa(env,Q_sarsa,alpha, gamma, epsilon, episodes)
```

[23]:
```
rewd_sarsa = []
rew = 0
for i,j in enumerate(timestep_reward_sarsa):
    rew = sum(timestep_reward_sarsa[0:i])/(i+1)
    rewd_sarsa.append(rew)
```

```
plt.figure(figsize = (10,6))
plt.title(label=f'Average Score over Time - SARSA policy ({np.
 ↪average(rewd_sarsa[-50:]) :.2f})')
plt.plot(rewd_sarsa,label = r'($\epsilon$ = '+ f'{epsilon},'+r'$\alpha$ = '␣
 ↪+f'{alpha}, '+ r'$\gamma$ =' +f'{gamma})')
plt.xlabel('Epochs')
plt.ylabel(f'Average Score')
plt.legend()
```

[23]: <matplotlib.legend.Legend at 0x28127f60fd0>

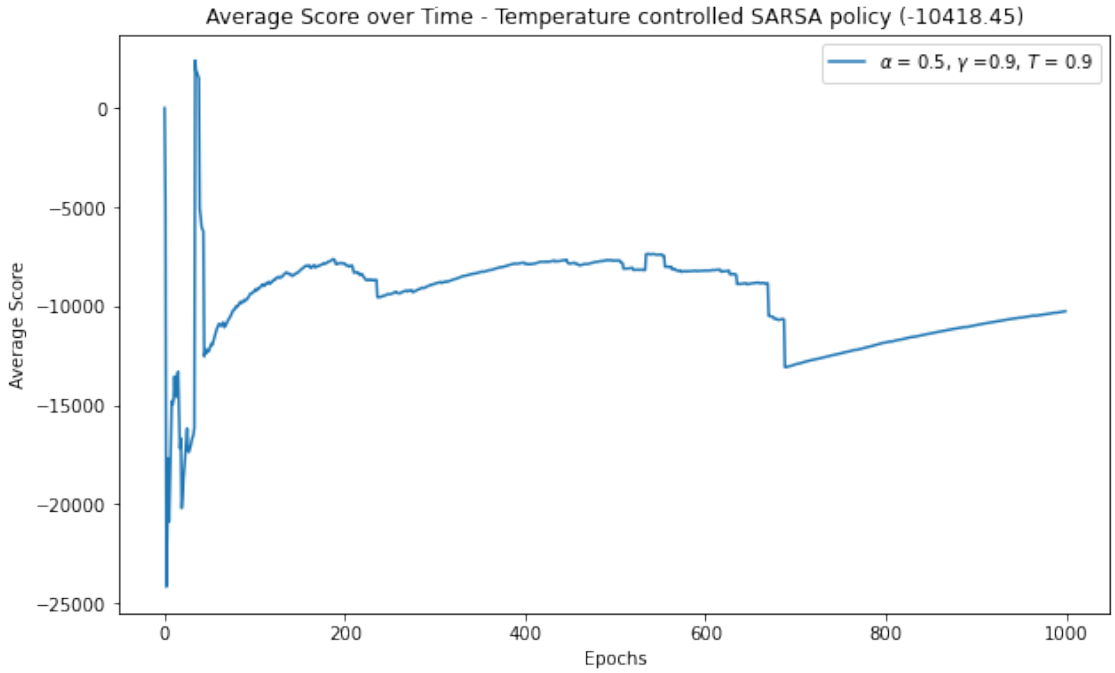


0.6.2 Temperature Controlled Exploration

[24]:
```
env = AirTraffic(planes=25,grid_size=[100,100],radius=50,destinations=1)
n_radius,n_theta, n_rho,n_actions = env.radius, 36,36, env.action_space.n
Q_temp = np.zeros((n_radius,n_theta, n_rho,n_actions))
alpha = 0.5
gamma = 0.9
epsilon = 0.5
episodes = 1000
cumsum_ep= []
timestep_reward = []
temperature = 10
timestep_reward_temp,cumsum_reward,cumsum_greedy_sarsa_temp =␣
↪greedy_Sarsa(env,Q_temp,alpha, gamma, epsilon, episodes,temperature)
```

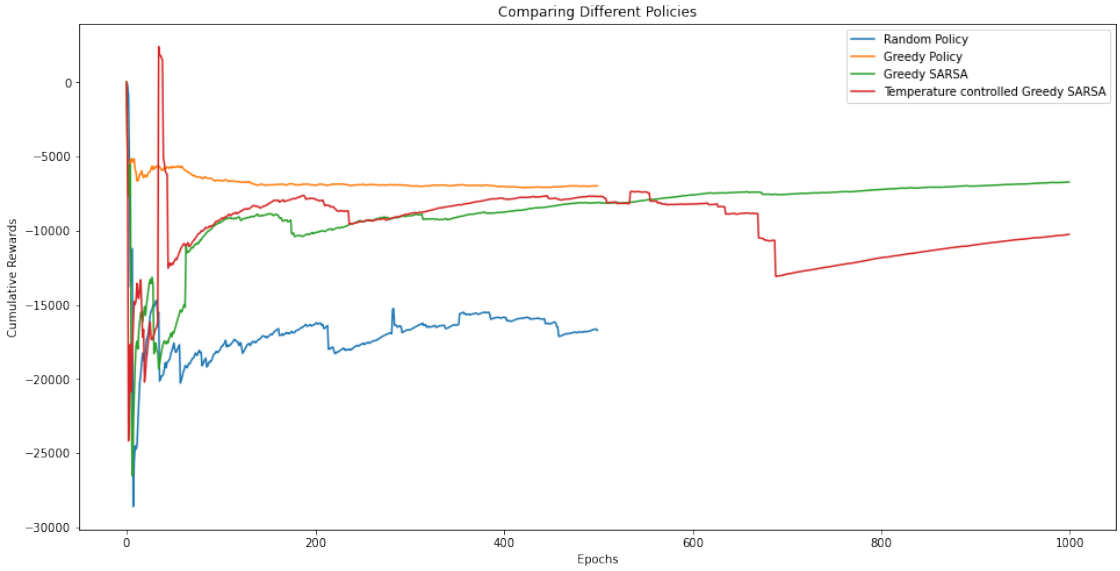
[25]:
```
rewd_temp = []
rew = 0
for i,j in enumerate(timestep_reward_temp):
    rew = sum(timestep_reward_temp[0:i])/(i+1)
    rewd_temp.append(rew)

plt.figure(figsize = (10,6))
plt.title(label=f'Average Score over Time - Temperature controlled SARSA policy␣
↪({np.average(rewd_temp[-50:]) :.2f})')
plt.plot(rewd_temp,label = r'$\alpha$ = ' +f'{alpha}, '+ r'$\gamma$ ='␣
↪+f'{gamma}, '+r'$T$ = '+f'{1-1/temperature}')
plt.xlabel('Epochs')
plt.ylabel(f'Average Score')
plt.legend()
```

[25]:
```
<matplotlib.legend.Legend at 0x28127fd6f10>
```



0.7 Comparative Plots

[26]:
```
plt.figure(figsize = (16,8))
plt.plot(rewd_rand, label = 'Random Policy')
plt.plot(rewd_greed,label = 'Greedy Policy')
plt.plot(rewd_sarsa,label = 'Greedy SARSA')
plt.plot(rewd_temp,label = 'Temperature controlled Greedy SARSA')
plt.legend()
plt.xlabel('Epochs')
plt.ylabel('Cumulative Rewards')
plt.title('Comparing Different Policies')
plt.show()
```

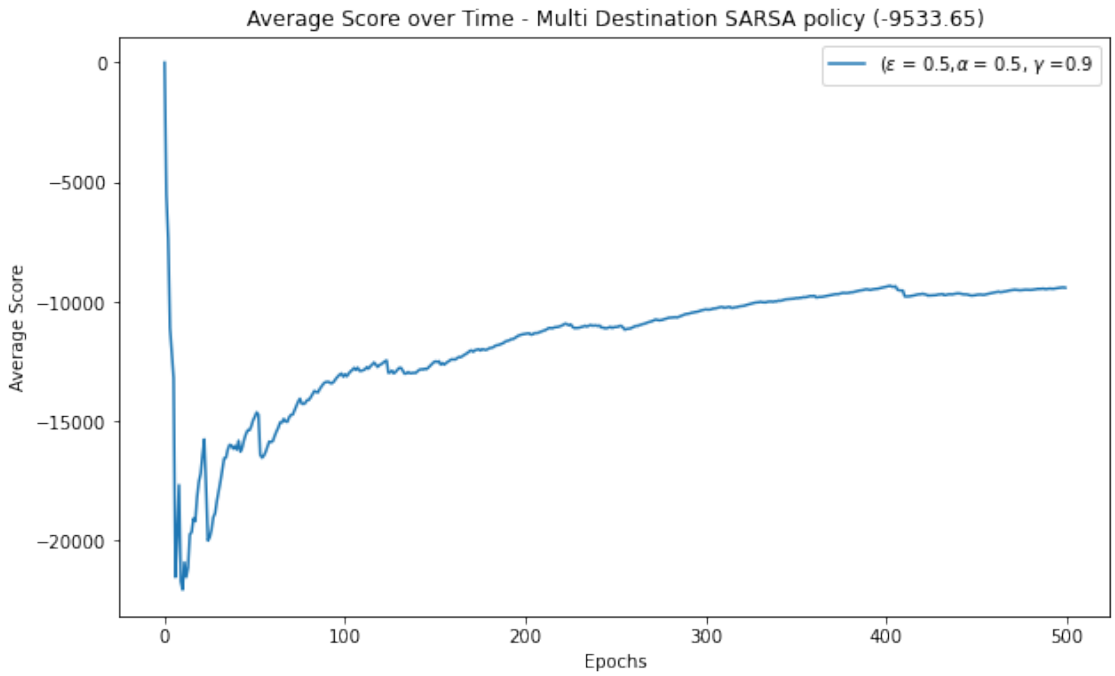


0.8 Multi Agent - Multi Destination Greedy + SARSA

[20]:
```
env = AirTraffic(planes=25,grid_size=[100,100],radius=50)
n_radius,n_theta, n_rho,n_actions = env.radius, 36,36, env.action_space.n
Q_multi = np.zeros((n_radius,n_theta, n_rho,n_actions))
alpha = 0.1
gamma = 0.9
epsilon = 0.1
episodes = 500
cumsum_ep= []
timestep_reward = []
timestep_reward_multi,cumsum_reward_multi,cumsum_sarsa_multi =␣
↪greedy_Sarsa(env,Q_multi,alpha, gamma, epsilon, episodes)
```

[31]:
```
rewd_multi = []
rew = 0
for i,j in enumerate(timestep_reward_multi):
    rew = sum(timestep_reward_multi[0:i])/(i+1)
    rewd_multi.append(rew)

plt.figure(figsize = (10,6))
plt.title(label=f'Average Score over Time - Multi Destination SARSA policy ({np.
↪average(rewd_multi[-50:]) :.2f})')
plt.plot(rewd_multi,label = r'($\epsilon$ = '+ f'{epsilon},'+r'$\alpha$ = '␣
↪+f'{alpha}, '+ r'$\gamma$ =' +f'{gamma}')
plt.xlabel('Epochs')
plt.ylabel(f'Average Score')
plt.legend()
```

[31]: <matplotlib.legend.Legend at 0x1fb88771a90>



0.8.1 Temperature Controlled

[23]:
```
env = AirTraffic(planes=25,grid_size=[100,100],radius=50)
n_radius,n_theta, n_rho,n_actions = env.radius, 36,36, env.action_space.n
Q_temp_multi = np.zeros((n_radius,n_theta, n_rho,n_actions))
alpha = 0.5
gamma = 0.9
epsilon = 0.5
episodes = 500
cumsum_ep= []
timestep_reward = []
temperature = 10
timestep_reward_multi_temp,cumsum_reward,cumsum_greedy_sarsa_temp_multi =␣
↪greedy_Sarsa(env,Q_temp_multi,alpha, gamma, epsilon, episodes,temperature)
```

[32]:
```
rewd_multi_temp = []
rew = 0
for i,j in enumerate(timestep_reward_multi_temp):
    rew = sum(timestep_reward_multi_temp[0:i])/(i+1)
    rewd_multi_temp.append(rew)

plt.figure(figsize = (10,6))
```

```
plt.title(label=f'Average Score over Time - Temperature controlled Multi␣
 ↪Destination SARSA policy ({np.average(rewd_multi_temp[-50:]) :.2f})')
plt.plot(rewd_multi_temp,label = r'$\alpha$ = ' +f'{alpha}, '+ r'$\gamma$ ='␣
 ↪+f'{gamma}, '+r'$T$ = '+f'{1-1/temperature}')
plt.xlabel('Epochs')
plt.ylabel(f'Average Score')
plt.legend()
```

[32]: <matplotlib.legend.Legend at 0x1fb87449880>