Reinforcement and Online Learning Final Project

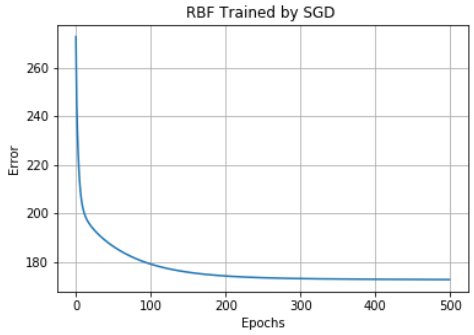
Zhou Jie 31384579 jz5n19@soton.ac.uk

1 Stochastic gradient descent

The learning rate was set to 0.001 and the weight was initialized as:

np.random.randn(J,1)

The graph below shows the convergence of the error over 500 epochs:



2 Implementation of learning controller

Both of the 2 ways to implement the learning controller were based on the pseudo code of *Episodic Semi-gradient Sarsa* in chapter 10 of the book *Reinforcement Learning* by *Richard S. Sutton and Andrew G. Barto* . The difference between 2 methods is reflected in the choice of target

Episodic Semi-gradient Sarsa for Estimating $\hat{q} \approx q_*$

Input: a differentiable action-value function parameterization $\hat{q} : \mathcal{S} \times \mathcal{A} \times \mathbb{R}^d \to \mathbb{R}$
Algorithm parameters: step size $\alpha > 0$, small $\varepsilon > 0$
Initialize value-function weights $\mathbf{w} \in \mathbb{R}^d$ arbitrarily (e.g., $\mathbf{w} = \mathbf{0}$)

Loop for each episode:
  $S, A \leftarrow$ initial state and action of episode (e.g., ε-greedy)
  Loop for each step of episode:
    Take action A, observe R, S'
    If S' is terminal:
      $\mathbf{w} \leftarrow \mathbf{w} + \alpha\big[R - \hat{q}(S, A, \mathbf{w})\big]\nabla\hat{q}(S, A, \mathbf{w})$
      Go to next episode
    Choose A' as a function of $\hat{q}(S', \cdot, \mathbf{w})$ (e.g., ε-greedy)
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha\big[R + \gamma\hat{q}(S', A', \mathbf{w}) - \hat{q}(S, A, \mathbf{w})\big]\nabla\hat{q}(S, A, \mathbf{w})$
    $S \leftarrow S'$
    $A \leftarrow A'$

The first method took the value in the q table as target, while the second method chose the recursive structure as the target in the red boxes of pseudo code.

The number of basis function includes 3 values: 6 ,10,20. But the car did not reach the top of hill when using any of them, i.e., I didn't get the reward 0.5 during the execution.

The following is a code snippet of *Episodic Semi-gradient Sarsa*(the second method) to implement the estimation process. And the first method is to replace the target with the corresponding values in the q table. The tabular method in the Appendix B was implemented according to the pseudo code of *Q-learning (off-policy TD control)* in the chapter 6. Different from my implementation, this implementation made the car reach the top of the hill during the policy iteration.

```python
for episode in range(episodes):
    print("Episode:",episode)

    obs=env.reset() # initialize state S
    pos, vel = discretization(env, obs) # discretize the current state
    if np.random.uniform(low=0, high=1) < epsilon:# E-greedy method: to initialize state A
        a = np.random.choice(env.action_space.n)
    else:
        a = np.argmax([q_hat(pos,vel,0,w)[0],q_hat(pos,vel,1,w)[0],q_hat(pos,vel,2,w)[0]])

    while True:
        env.render()
        pos, vel = discretization(env, obs) # discretize the current state
        obs, reward, terminate,_ = env.step(a)  #Take action A, observe R, S'
        print(w)
        print(obs, reward,a, terminate)
        delta_qhat=np.array(U[q_hat(pos,vel,a,w)[1],:]).T
        if terminate:
            w=w+alpha*(reward-q_hat(pos,vel,a,w)[0])*delta_qhat # Updtae w if the episode is terminated
            error.append((reward-q_hat(pos,vel,a,w)[0])**2)
            break #Go to the next episode
        #Choose A' as a function of q_hat(S', ·, w) (e.g., E-greedy)
        pos_, vel_ = discretization(env, obs) #discretize the next state
        if np.random.uniform(low=0, high=1) < epsilon:# E-greedy method: select a random action 'a' with probability E
            a_ = np.random.choice(env.action_space.n)
        else:
            a_ = np.argmax([q_hat(pos_,vel_,0,w)[0],q_hat(pos_,vel_,1,w)[0],q_hat(pos_,vel_,2,w)[0]])
        w=w+alpha*(reward+gamma*q_hat(pos_,vel_,a_,w)[0]-q_hat(pos,vel,a,w)[0])*delta_qhat #Update w if it's not terminated
        error.append((reward+gamma*q_hat(pos_,vel_,a_,w)[0]-q_hat(pos,vel,a,w)[0])**2)
        a=a_ #Reassign a
```

3 Implementation with incorporating RAN

The incorporated statements of RAN was marked blue in the pseudo code below, and the original implementation was also adjusted accordingly.

Loop for each episode:

S, A ←initial state and action of episode (e.g., ε-greedy)

$\delta=\delta_{max}$

Loop for each step of episode:

Take action *A*, observe *R, S'*

compute error

If S' is terminal:

error=R- q^(S, A, w)

Choose A' as a function of q^ (S', ·, w) (e.g., ε-greedy)

error=R+γq^(S', A', w)- q^(S, A, w)

find distance to nearest center d

If ||error|| >ε and d >δ:

allocate new unit

if this is the first unit to be allocated:

width of new unit = kδ

else

width of new unit = kd

else:

#perform gradient descent

If *S'* is terminal:

w ←**w** +α*error* ∇q^(*S, A*, **w**)

Go to next episode

w ←**w** +α*error* ∇q^(*S, A*, **w**)

if $\delta>\delta_{min}$:

$\delta=\delta*exp(-1/\Gamma)$

S, A ←*S', A'*

4 Scientific study summaries: Resource Management with Deep Reinforcement Learning

A succinct description of the problem domain and why reinforcement learning was thought to be the right approach.

The problem the paper tried to solve is the resource management problem in the computer. The paper tried to use a Deep reinforcement learning method (DeepRM) to schedule the workload.
The reasons for taking DeepRM are listed below:
First, decisions made by these systems are often highly repetitive, thus generating an abundance of training data for RL algorithms.
Second, RL can model complex systems and decision-making policies as deep neural networks analogous to the models used for game-playing agents. Different "raw" and noisy signals can be incorporated as input to these neural networks, and the resultant strategy can be used in an online stochastic environment.
Third, training for objectives that are hard to optimize directly because they lack precise models if there exist reward signals that correlate with the objective.
Finally, by continuing to learn, an RL agent can optimize for a specific workload and be graceful under varying conditions

Clear identification of states, rewards, state transition and policy models.

State of the system: the current allocation of cluster resources and the resource profiles of jobs waiting to be scheduled.

Rewards: $\sum_{j \in J} \frac{-1}{T_j}$ where J is the set of jobs currently in the system (either scheduled or waiting for service).

State transition: the scheduled job is moved to the appropriate position in the cluster image. Once the agent picks a $=\varnothing$ or an invalid action, time actually proceeds: the cluster images shift up by one timestep and any newly arriving jobs are revealed to the agent.

Policy models: the policy is represented as a neural network (called policy network) which takes as input the collection of images described above, and outputs a probability distribution over all possible actions.

A clear description of the learning paradigm.

The policy network was trained in an episodic setting. In each episode, a fixed number of jobs arrive and are scheduled based on the policy. The episode terminates when all jobs finish executing.
To train a policy that generalizes, the paper considered multiple examples of job arrival sequences during training, henceforth called jobsets. In each training iteration, simulate N episodes for each jobset to explore the probabilistic space of possible actions using the current policy, and use the resulting data to improve the policy for all jobsets. Specifically, the state, action, and reward information are recorded for all timesteps of each episode, and use these values to compute the (discounted) cumulative reward(return), v_t, at each timestep t of each episode.

The significance of the results obtained and where you think results in the paper might lead to

The state-of-the-art Deep RL provides an alternative method in many fields. This paper was inspired by these techniques and applied it to large-scale systems. The paper showed that the RL agent is comparable and sometimes better than ad-hoc heuristics for a multi-resource cluster scheduling problem in the early experiments. This method may be applied to other industries that have resource management issues, such as crew scheduling in the airlines or the parcels delivery dispatch in the mail express company.