Programming Exercise 3

The aim of the exercise is to familiarize you with various learning control algorithms. The goal is to solve several variants of the puddle world problem, shown in Figure 1. This is a typical grid world, with 4 stochastic actions. The actions might result in movement in a direction other than the one intended with a probability of 0.1. For example, if the selected action is N, it will transition to the cell one above your current position with probability 0.9. It will transition to one of the other neighbouring cells with probability 0.1/3. Transitions that take you off the grid will not result in any change.

There is also a gentle Westerly blowing, that will push you one **additional** cell to the east, regardless of the effect of the action you took, with a probability of 0.5.[1]

The episodes start in one the start states in the first column, with equal probability. There are three variants of the problem, A, B, and C, in each of which the goal is in the square marked with the respective alphabet. There is a reward of +10 on reaching the goal. There is a puddle in the middle of the gridworld, which the agent likes to avoid. Every transition into a puddle cell, gives a negative reward, depending on the depth of the puddle at that point, as indicated in the figure.

RL-Glue is a standard software protocol for benchmarking and interconnecting RL agents and environments. This assignment must use the RL-Glue interface standards and benchmarking routines. The RL-Glue home page (http://rlai.cs.ualberta.ca/RLBB/top.html) is part of the RL-AI net, hosted at Univ. of Alberta. Please note that the puddle world implementation given there is a continuous version of the assignment problem, and hence cannot be used. :).

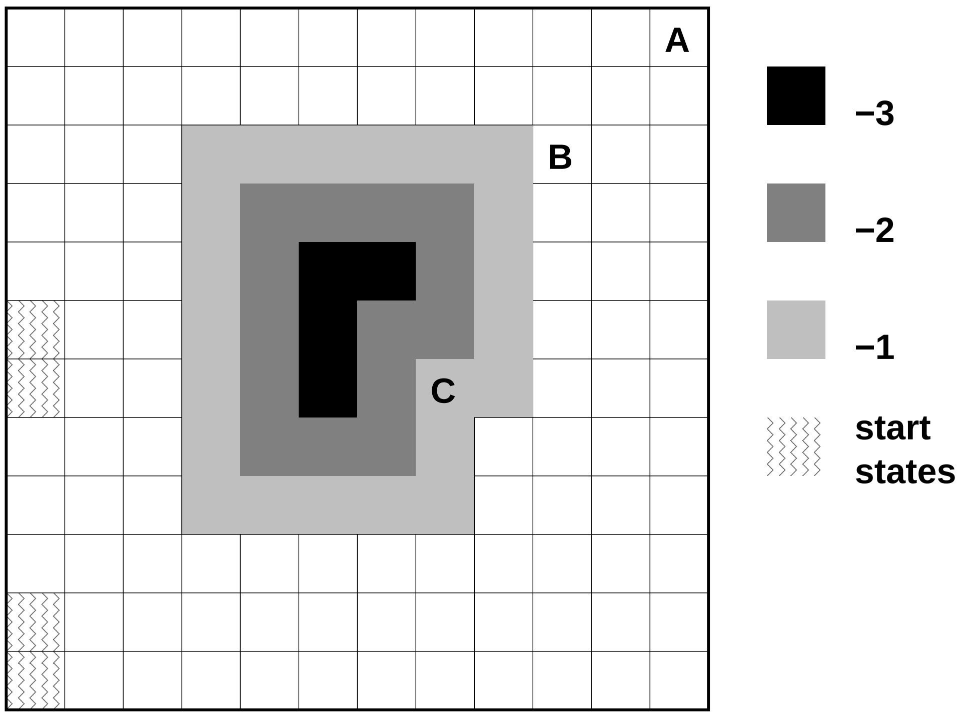

Figure 1: The Puddle World

[1]For task C you might want to turn off the wind.

1. Implement Q-learning to solve each of the three variants of the problem. For each variant run experiments with a gamma value of 0.9. Turn in two learning curves for each experiment, one that shows how the average number of steps to goal changes over learning trials and the other that shows how the average reward per episode changes. Compute the averages over 50 independent runs. Also indicate the optimal policies arrived at in each of the experiments.

2. Repeat part 1 with Sarsa. For both parts, pick a learning rate that seems to best suit the problem.

   (a) Repeat this part with Sarsa(λ), for *lambda* values of 0 (already done), 0.3 0.5, 0.9, 0.99, 1.0. In addition to the plots specified in part 1, also turn in a plot that shows the average performance of the algorithms, for various values of lambda after 25 learning trials. Again pick a learning rate that seems to best suit the problem.

3. Consider the following parameterization of a policy: There is a "preference" for each action, for each column and for each row. Thus the set of preferences can be denoted by $\{\theta_x(N,0), \theta_y(N,0), \theta_x(S,0), \theta_y(S,0), \theta_x(E,0), \theta_y(E,0), \ldots, \theta_x(W,9), \theta_y(W,9)\}$, for a total of 80 preference values.[2] The total preference for an action a in a state (i,j) is given by $\theta_x(a,i) + \theta_y(a,j)$. The action probabilities are generated by a soft-max function using these preferences.

   Implement a MC policy gradient algorithm, along the lines described in class. Choose appropriate learning rates, and turn in two curves for each variant as indicated in the first part as well as the optimal policies learnt.

   Explicitly derive the update equations. Also, comment on the suitability of this policy parameterization for the given task, and for gridworld problems in general.

Programming Exercise 4

Now consider a puddleworld exactly like the above, but blown up by 1000 times. While it is still possible to run your earlier code on this version, the aim of this exercise is to get you to play around with function approximation.

1. Repeat problem 1 from exercise 3, with a linear function approximator. You can try a simple CMAC if that is convenient. [Bonus: How do you think the parameterization given in part 3 for the policy will fare in case of value function approximation?]

2. Repeat problem 2 from exercise 3 with a linear function approximator. Report results for at least 2 values of λ.

3. (Bonus) Compare the performance of Sarsa(λ) with and without function approximation on this task. Specifically note the rates of convergence and the quality of the final policy.

Evaluation Criteria

The points will be given according to the following criteria:

[2]Compare this with the number of Q values needed: 400. Also, reflect on how this will scale compared to Q-learning or SARSA.

- Correct coding of the puddle world dynamics
- Correct coding of the learning algorithms
- Compliance with RL-Glue
- Correct performance of the learning algorithms
- Neatness of the graphs - correctly labeled etc. and well commented code
- Correct derivation of policy gradient updates and thoughtfulness of comments

Note: You can program in any language you desire, but the RL-Glue benchmarking routines can work only with in C/C++, Java, and Python (and other C-callable languages). For other languages, you might need to recode the benchmarking routines also.

You have to turn in a lot of learning curves. Organize them in a sensible fashion, across all parts, so that the number of graphs you turn in is minimal, yet each graph remains comprehensible. :-).

The due date for both the homeworks is **Friday, November 30th**.