CS550: Convergence

For this homework, we want to determine whether the RL converges and whether it converges to a good policy.

Question 1: Random Nature of Reinforcement Learning

There are a lot of random decisions that RL uses in during both learning and training.

Writeup 1.1 Describe at least 3 decisions that are randomly made during training.

Writeup 1.2 Describe at least 1 decision that is randomly made during testing. Hint: see the question titled Test Distribution.

Question 2: Learning Multiple Policies

In your previous homework, you used Reinforcement Learning to learn a single policy. Given that there are a lot of random decisions that are made, the amount of time needed for a RL to converge might different from one time to the next. So, when evaluating how well RL is learning a policy, you must run RL a number of times. Each time we run RL, we will call this ‘learning a policy’.

In this homework, we will be analyzing RL and the car-buying domain. We will determine how well RL is converging, how long it takes to get to a good policy, how the epsilon decay rate affects Monte Carlo learning, how alpha affects learning, and how Q-Learning differs from Monte Carlo. Hence, we will be learning multiple policies for each experiment.

To control your experiment, we will add two global variables to the ones we added in the previous homework. We will have a global variable for number of policies being learned, and the number of epochs of training. The full set of variables being used is given below:

set ::NumPolicies 3
set ::NumEpochs 25
set ::InitialEpsilon 0.40
set ::EpsilonDecay 0.97
set ::RunsPerTest 500

Add a procedure called MultiTrain that will run Train for NumPolicies times. Make sure that when you start learning a new policy, you reset your Q scores, your counts, and your policy to ensure that each policy is learned completely independent of the earlier policies. You can do that with the following code:

catch {unset ::RLQ}
catch {unset ::RLPolicy}
catch {unset ::RLCnt}

With these settings, my computer takes around 4 minutes to run the experiment (training 3 policies for 25 epochs each with 6 test sessions).

Writeup 2.3 Hand in a copy of your Train and MultiTrain code.
Question 3: Reporting Results

To better analyze the results from the multiple runs, it is useful to make a table of the average dialogue costs from each test session of each policy learned.

For each test session of each policy, keep track of the average dialogue cost ::DialogueCosts($policy,$epoch), where $policy ranges is an integer that ranges over the number of policies (0 to 2 for the current example).

For each test session, keep track of what epsilon is in ::EpsilonValues($epoch). Do not worry about rewriting this for each policy, as you will be rewriting it with the same value.

Create a tcl procedure that will print all of your results in a table with each policy in a different column, and each epoch in a different row. Include a final column that averages the results across all policies for each epoch. Include an initial column that has the epsilon value. Note that your code should work for any values of the parameters MaxEpochs, NumPolicies, TestEvery, and TestAlso.

To make the table of results, use the tcl format command. To print one result, use

puts -nonewline [format "%7.3f" ::DialogueCosts($policy,$epoch)]

Note the use of the -nonewline. This allows you to build up each output line in a for loop.

Also note the use of the format command. This has the same syntax as C++’s printf command. The format specifier of %7.3f will print a floating point number using 7 characters with exactly 3 decimal places. This will allow you to easily line up each row of results on the tcl console, and without relying on tab characters. The 7 refers to the number of characters in total, and since there are 3 decimal places and 1 period, dialogue costs up to 999.999 can be displayed without causing misalignment. Since the average dialogue cost for an epoch though will be rarely over 100, you can omit spaces between these costs so that you can include more dialogue policies in your experiment, and still have them nicely formatted on a single line.

You will be running a number of experiments in this section. To avoid mixing up your experiments, print the values of the parameters that we will be changing before printing the table. There is no need to include NumPolicies, NumEpochs, or InitialEpsilon as these can be seen in the table itself. As you make your way through the homework, you’ll be adding extra parameters.

Below is the code that will print this. Note that the set command, when given a variable name, and no other option, will simply return the value of the variable. This allows you build variable names, something you can only do in interpreted languages.

foreach var {EpsilonDecay RunsPerTest} {
  puts [format "%-12s %s" $var [set ::$var]]
}

In my code, after each policy is learned, I print all of the current results (rather than just wait until the end). This allows me to see how the experiment is progressing.

Below is what my table of results looks like. Your results should look similar. After one epoch of training, the average dialogue cost could be anywhere from around 15 to over 200. By 50 epochs, all three policies should have an average dialogue cost under 12. For each policy, the average cost might occasionally increase as training increases. However, for the last column, which averages over the 3 policies, you should still a steady decrease.

EpsilonDecay 0.99
RunsPerTest 500

<table>
<thead>
<tr>
<th>epsilon</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>all</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.2000</td>
<td>40.830113.582</td>
<td>19.680</td>
<td>58.031</td>
</tr>
<tr>
<td>3</td>
<td>0.1960</td>
<td>20.874</td>
<td>17.408</td>
<td>21.532</td>
</tr>
</tbody>
</table>
Writeup 3.4  Hand in your code for producing the table, and hand in your table of results. (If you include the output in your homework writeup using a fixed-size font, you should not have to do any extra formatting of your table.)

Question 4: First Experiment

Now that you have your code working, let’s do a larger evaluation:

```c
set ::NumPolicies 5
set ::MaxEpoch 150
```

This took about 26 minutes to run on my computer. You should notice two things. First, after 150 epochs of training, no two policies probably agree on the same average cost. Second, for each of the 5 policies, you should see that the average cost differs between the last epoch and the second last epoch. Third, for some policies, you might see in the last 3 test epochs that the dialogue cost going up and down, rather than staying the same or decreasing. Part of the reason for this is the focus of the next question.

Writeup 4.5  Hand in your table of results.

Question 5: Test Distribution

One of the difficulties in analyzing the results of the previous question is that there are many sources of noise in testing a policy. One of them is due to how the users are sampled during testing, which is what this question will address.

To re-iterate, we are assuming that are population of actual users have a distribution where they like each of the 2,000 cars in the database equally likely (uniform distribution).

In the previous questions, during testing, we randomly chose 500 users in order to evaluate how well the learned policy is performing so far.

Writeup 5.6  For testing: If processing time is of no concern, which of the following options do you think will give the best estimate of the policy’s performance: (a) randomly choose 100 users; (b) randomly chose 500 users; (c) randomly choose 2,000 users; (d) randomly chose 100,000 users; (e) go through each of the 2,000 different users exactly once. Order the above from best to worse, and justify your answer.

Writeup 5.7  For training: Why do you think it might be better to randomly sample, and not to go through each of the 2,000 users systematically.

Change your code so that for testing, you just cycle through the 2,000 users once. Hence, you will be choosing users differently for testing than for training. I actually use a global variable to control this, TestUniform. When this is 1, during training, I just set the users to the one after I previously tested with. So, to uniformly test through all 2000 users, I used the following parameters:

```c
set ::TestUniform 1
set ::RunsPerTest 2000
```

By controlling this functionality through global variables, you can easily rerun your earlier experiments if you find a bug while you are working on the remainder of the homework.
Writeup 5.8  Hand in the code that you changed.

Rerun the experiment of the previous question, learning 5 policies for 150 epochs. This took me 43 minutes to run on my computer. The increase is due to spending more time in the test sessions. Training is taking the same amount of time.

Writeup 5.9  Hand in your table of results.

Writeup 5.10  How do these results differ from your previous results?

Writeup 5.11  How do you know that RL is not converging for this experiment?

Writeup 5.12  Compare the dialogue cost of the 5 learned policies after 150 epochs of learning against the average dialogue cost of the best hand-crafted policy from two homeworks ago (where the system asked year, mileage and the rest of the attributes, and summarized with there was at most one matching car). Given your answer, even though RL might not have converged, are the resulting policies still useful?

Question 6: Simplifying the Problem a Bit

The previous experiment probably took my computer 43 minutes to run. But, rather than do the upcoming experiments on just 5 policies for 150 epochs, we need to do larger tests (more policies and more epochs) to understand how RL works.

set ::NumPolicies 8
set ::MaxEpoch 300
set ::TestEvery 25
set ::TestAlso {1 3 5 10 15}
set ::RunsPerTest 2000
set ::TestUniform 1

Writeup 6.13  How many dialogues are being simulated in learning each policy? How many are during training epochs and how many for testing the current policy? Comment on the relative number of training versus testing dialogue runs.

The amount of time needed to run an experiment is roughly linearly correlated to the number of dialogues being run (whether during training or testing). So, if the number of testing sessions can be reduced, while doing the same number of training, we can decrease how long this homework assignment will take!

To speed up testing, lets assume that rather than having 2000 unique users, we only have 200, which correspond to the first 200 cars in the inventory. So, during testing and training, we will just pick users that have one of the first 200 cars in mind. Based on the previous question, for testing we will just run through the 200 users exactly once.

Even though we are changing the users so that they just have one of the first 200 cars in mind, lets still have the system knowing about all 2000. This way, the system’s policy is roughly just as complicated as before, as it has to learn to distinguish between the 2,000 cars. Also, this way our new results will also roughly correspond to our previous results in this homework and the previous two homeworks (in terms of dialogue cost of the best policies).

Writeup 6.14  With this change, how many dialogues will be run for learning each policy, both for training and testing? How much should we have sped up learning a policy?

I used a global variable to control how many different users that I am training and testing with. The parameters to control this experiment are now as follows:

set ::NumPolicies 8
set ::MaxEpoch 300
set ::TestEvery 25
set ::TestAlso {1 3 5 10 15}
set ::NumUsers 200
set ::RunsPerTest 200
set ::TestUniform 1

Writeup 6.15 Hand in the code that you changed.

Now, run the experiment with the parameters above (8 policies for 300 epochs). This took my computer 74 minutes to run.

Writeup 6.16 Hand in the table of results.

Question 7: More Epsilon Decay with Monte Carlo

Epsilon controls how often RL considers an action other than the action prescribed by the current policy. In this experiment, we contrast the decay rate of 0.99 with 0.97.

Writeup 7.17 By 150 epochs, what is the value of epsilon for a decay rate of 0.97 versus a decay rate of 0.99?

Let’s estimate that each dialogue has about 10 system utterances in it. An epoch of training has 100 dialogues.

Writeup 7.18 How many non-policy actions will it probably make when it is at 150 epochs for both decay rates? Explain how you arrived at your answer (I’m just looking for a rough calculation).

Writeup 7.19 Based on your previous answer, why should we not see much change after 150 epochs of training at a decay rate of 0.97?

Writeup 7.20 How does this affect whether RL can converge on the optimal policy?

Writeup 7.21 Hand in the table of the average costs using a decay rate of 0.97. How do the results differ with the results that you had for 0.99? Here, focus on how the average cost of the 8 policies changes as training progresses? Is there a certain epoch where improvement in dialogue cost stops?

Question 8: No Epsilon Decay with Monte Carlo

Repeat the last experiment but set the epsilon decay to 1. This means your epsilon should stay at 0.2.

Writeup 8.22 Hand in your table of results. How are the dialogue costs changing as training increases, at both the level of individual policies, and overall. For the average of the eight policies, are the dialogue costs better than for an epsilon decay of 0.99? Are they continuing to improve, even at 300 epochs?

Writeup 8.23 Why do you think this version is not converging? Think in terms of how the updating is being done in Monte Carlo. Are the Q scores for a state action pair being updated with the properly if there is a non-policy action later in the dialogue?

Writeup 8.24 What if you restrict updating Q scores so that state-action pairs in a dialogue history that precede a non-policy action are not updated? With this fix the above problem? Will it introduce new problems?

Writeup 8.25 What is lowering epsilon really accomplishing?
Question 9: Another Attempt at Setting the Epsilon Decay

Try an epsilon decay rate of 0.998.

**Writeup 9.27** Hand in your table of results. Is this resulting in a better overall average than no decay? Is this resulting in better monotonic decreases in dialogue cost for each of the individual policies?

Question 10: Alpha

Add a global variable AlphaEight to control how alpha is set. If it has the value of 1, alpha should be set to 1 over the power of the number of visits to 0.8: \(\frac{1.0}{\text{pow}($\text{cnt}, 0.8$)}\).

**Writeup 10.28** Explain what this change is doing. Why should it speed up learning?

**Writeup 10.29** Hand in the code for the procedure that you changed.

**Writeup 10.30** Hand in the table of results. Discuss how your results differ from your previous results.

Question 11: Q-Learning

Implement Q-learning.

When you are updating a state-action pair \(s-a\), you need to make use of the Q score of the subsequent state \(s'\) with the policy action for that state. However, the policy might not yet specify an action for state \(s'\) as this state might not have been seen in the previous epochs. In this case, you should just update the Q scores using Monte Carlo.

Also, remember that Q-learning does not need epsilon to change. Use an epsilon of 0.20 with no decay, and learn 10 policies for 300 epochs.

Test your procedure with the following code

```plaintext
# lets assume that state a=0 b=0 c=0 had previously been set to askB
# and we now have the following Q values for the states
set ::RLPolicy(s1) a1
set ::RLPolicy(s2) a2
set ::RLHistory {{s1 a1 1} {s2 a1 1} {s3 a1 5}}
UpdateQ
```

**Writeup 11.31** Hand in the code for the procedure that you changed. Hand in your table of results. Discuss how your results differ from the previous quest, and why that might be.