Exercises
Reinforcement Learning: Chapter 8

Exercise 8.1. Show that table-lookup TD(\(\lambda\)) is a special case of general TD(\(\lambda\)) as given by equations (8.5–8.7).

Exercise 8.2. State aggregation is a simple form of generalizing function approximation in which states are grouped together, with one table entry (value estimate) used for each group. Whenever a state in a group is encountered, the group’s entry is used to determine the state’s value, and when the state is updated, the group’s entry is updated. Show that this kind of state aggregation is a special case of a gradient method such as (8.4).

Exercise 8.3. The equations given in this section are for the on-line version of gradient-descent TD(\(\lambda\)). What are the equations for the off-line version? Give a complete description specifying the new approximate value function at the end of an episode, \(V'\), in terms of the approximate value function used during the episode, \(V\). Start by modifying a forward-view equation for TD(\(\lambda\)), such as (8.4).

Exercise 8.4. For off-line updating, show that equations (8.5–8.7) produce updates identical to (8.4).

Exercise 8.5. How could we reproduce the tabular case within the linear framework?

Exercise 8.6. How could we reproduce the state aggregation case (see Exercise 8.4) within the linear framework?

Exercise 8.7. Suppose we believe that one of two state dimensions is more likely to have an effect on the value function than is the other, that generalization should be primarily across this dimension rather than along it. What kind of tilings could be used to take advantage of this prior knowledge?
Exercise 8.8 (Programming Exercise; based on the paper (1)). The exponentiated gradient (EG) algorithm is a gradient algorithm with the following twist: instead of the weights it updates the logarithm of the weights by using the gradient.

Motivation: Imagine that the target function, $\theta_s^T \phi$, is such that many of the components of $\theta_s$ are zero. A little bit of terminology: When a vector has many zero elements, it is called a sparse vector, otherwise it is called dense. Now, in the supervised learning literature it is known that when $\theta_s$ is sparse and the features that are in the training set are dense then EG converges faster than the “additive” gradient method.

The following is the generalization of TD($\lambda$) to use the EG idea: Let the features be $\phi_0$. Since EG can only update positive weights, we need to duplicate the features and the weights. In particular, given the original features, $\phi_0$, we let $\phi(s) = [\phi_0(s)^T, -\phi_0(s)^T]^T$ (we concatenate $\phi_0(x)$ and $-\phi_0(x)$ as vectors). Let us call the dimension of the resulting vector $d$ (if the dimension of $\phi_0$ is $d_0$, $d = 2d_0$).

Now, the EG update with accumulating traces is

$$
\log(\theta_{t+1}) = \log(\theta_t) + \alpha_t \delta_t e_t,
$$

$$
\delta_t = r_{t+1} + \gamma \theta_t^T \phi(s_{t+1}) - \theta_t^T \phi(s_t),
$$

$$
e_t = \gamma \lambda e_{t-1} + \phi(s_t).
$$

If we want to use replacing traces, we may e.g. use Mohsen’s rule\(^1\):

$$
e_t = \Pi_\eta(\gamma \lambda e_{t-1} + \phi(s_t)),
$$

where $\eta > 0$, $\Pi_\eta(e) = (\Pi_{1,\eta}(e_1), \ldots, \Pi_{d,\eta}(e_d))^T$, with $\Pi_{i,\eta}$ projecting its argument to the interval $[-\eta \phi_{i,\max}, \eta \phi_{i,\max}]$, where $\phi_{i,\max} = \max_s |\phi_i(s)|$ ($\Pi_{i,\eta}(x) = (x \land \eta \phi_{i,\max}) \lor (-\eta \phi_{i,\max})$). This algorithm can be extended to SARSA($\lambda$) without any problems. Let us call the resulting algorithm EG-SARSA($\lambda$).

Your task is as follows:

- Implement the Mountain-Car problem as described in the book on pages 214–215.
- Implement SARSA($\lambda$) and EG-SARSA($\lambda$) and compare them in a number of different situations on the mountain car domain.

You can choose to implement tile-coding or to use radial-basis functions. Test the algorithms with $\lambda \in \Lambda \triangleq \{0, 0.25, 0.5, 0.75, 1.0\}$. You have to have results for various function approximators:

- Features with broad generalization (large tiles, or large standard deviation parameter for the RBFs)

\(^1\)In the experiments you can also try Gabor’s rule!
• Features with narrow generalization

• Mix features with broad and narrow generalization (multi-resolution case). You can simply merge the two features sets.\(^2\)

• Pick the best performing of the above three function approximator constructions. Let \(d_0\) be the dimension of the resulting feature. Then add \(d_0\) additional random features, so that the dimension of the feature-vector will be \(2d_0\).

The random features simulate the situation when you are trying to learn a controller for a plant (e.g. stock market!) and you add all the features that might be relevant for the task but some are not relevant. How to add random features? If \(\phi_i\) is a random feature then at time \(t\) its value is a random variable that you generate using your computer. This value is totally independent of \(s_t\), the state just visited at time \(t\).

Report your results and findings on pretty graphs, with explanation and the details of how you set the parameters (and why). To set the learning rate, you want to start with values that were reported to be successful previously in a relevant paper (or the book) in a similar situation and then use experiments to find out what a good value might be. Note that EG-SARSA(\(\lambda\)) will likely need learning rates quite different than SARSA(\(\lambda\)). Try to maximize the chances that the simulation leads to success!

Note: You have to run at least \(2 \times 5 \times 3 = 30\) experiments, so start early or you will no time to finish the experiments. A typical experiment, consisting of 100 runs, each with 10,000 episodes took ca. 10 minutes on my 1.86GHz Mobile Pentium machine, compiled under gcc. Pay attention to the variance of your measurements. What should you measure? Follow the book on pages 214–215. You may also want to plot the value function (in 3D!) to verify if your algorithm is working in a sensible way. You can use gnuplot (http://www.gnuplot.info/) for this.

If you want, you can use an existing simulation environment, such as RL-Glue, available at http://rlai.cs.ualberta.ca/RLBB/top.html. RL-Glue has the Mountain Car environment ready for use (http://rlai.cs.ualberta.ca/RLL/mcp.tar.gz), you just need to understand how to compile and use it. You can also use any other existing implementation of SARSA(\(\lambda\)) or Mountain Car (just this latter should be close to what is in the book, so that we can compare results!). However, you are still not supposed to copy each others’ implementations! (Pardon me for mentioning this.) Note that the above Mountain Car package in RL-Glue has everything: the environment and SARSA(\(\lambda\)), too. If someone wanted to contribute EG-SARSA(\(\lambda\)) to RL-Glue (meeting standards of quality, etc.), she/he will get bonus points for doing so!


\(^2\)For EG, don’t forget to use the negated features!