CMPUT 366 Reading-Writing Exercise 8

Name: Minghan Li
Student ID: 1561234
CCID: minghan4

In the previous chapters we have been talking about using tabluar method to store the state values in a table and update them individually. One advantage of this method is the update on each state are independent, meaning that the other states won't be affected if one state changes its value. However, the limitation of this method is also quite obvious that it becomes very ineffcient and impratical when the state and action space is large. It becomes even worse when we have continous states and actions because we simply just cannot directly apply the tabular method(unless we discretize it). So this naturally calls for using function approximation to represent the state value, which basically takes any kinds of state inputs and mapps them to real number. Function approximation is indeed powerful but the drawback of it is also obvious: We can only update state values by changing the weights of the funciton which results in the value estimate changing in other states. The main idea of this chapter is to apply function approxiamtion on policy evaluation, but we needs to modify our original reinforcement learning algorithms carefully before we combine them together.

When we try to optimize a function on some fixed dataset, one natural objective is the mean squared error between the target and the estimate, and then apply gradient descent method to find the best parameters of the function. In reinforcement learning the objective is similar which is called *Mean Squared Value Error*, and the target can be actual returns or TD target. However, it is also a little different because now we have a constant stream of data, and if we try to minimize the msve between TD target and the estimate, we can't directly apply standard gradient descent method since the target needs the estimate from the function approxiamtor. So this leads to using *Semi Gradient Descent* algorithm which assumes the target is independent of the fucntion parameters. Another difference in reinforcement learning with function approximation is that the states actually has certain stationary distribution under a policy, so a standard thing to do is weight the objective with the state-distribution, which means that we cares more about the states that we frequently run into.

After we determine our objective and optimization method, we will have to choose a functional class to represent our state value. One simplest choice will be choosing linear function. If we also use TD(0) target then we will have *Linear TD(0)*, which has fixed point solution and bounded by the the multiple of the minimum msve. An then we also have to decide how we would like to construct our features of the states. Common choics including but not limited to Fourier basis and Polynomial Basis. If the state is continous, we can also choose to discretize the states, or aggregate the states. Naive *state aggregation* method doesn't work well because the groups are totally disjoint with each other. Overlapping between groups will generally give us better generalization, just like the convolutional layer in neural network. This leads us to methods such as *corse coding* and *tile codings*, which has overlapping features and small computation costs. However, choosing the size of the tiles and the number of tilings are task-dependent and we might need to tune these parameters. But it is found that in practice methods such as tile-coding work well with corser tiles and more tilings.

At the end of this chapter the authors also discuss about non-linear function approximation methods such as *radial basis function* and *artificial neural network*. Non-linear methods allow us to have more expressive function but it is also more problematic since it tends to overfit and is hard to optimize on non-convex(sometimes) error surface.