CSCE 580: Artificial Intelligence
Graduate Project
Due: 04/22/2021 at 11:59pm

Installation

We will be using the same `conda` environment as in Homework 0.

The entire GitHub repository should be downloaded again as changes were made to other files. You can download it with the green "Code" button and click "Download ZIP".

1 Value Iteration (100 pts)

Key building blocks:

- `env.sample_transition(state, action)`: returns, in this order, the next state and reward
- `env.get_actions()` function that returns a list of all possible actions
- `env.is_terminal(state)` returns True if the state is terminal
- `env.states_to_nnet_input(states)` takes a list of states and returns their representation for a neural network as a numpy.ndarray.

For this project you will be implementing approximate value iteration with a deep neural network. We will be approximating the optimal value function (the negative cost-to-go) for the 8-puzzle. In the previous coding project, the answer was given and a neural network needed to be trained. However, in this coding project you will be using approximate value iteration to approximate this from scratch, without any prior knowledge related to the optimal value function.

Value iteration has the following form:

$$V(s) = \max_a (r(s,a) + \gamma \sum_{s'} p(s'|s,a)V(s')) \tag{1}$$

Since the 8-puzzle is deterministic, we can say that the next state is given by a deterministic function $A(s,a)$. Additionally, we set γ to 1. Therefore, we can simplify the equation:

$$V(s) = \max_a (r(s,a) + V(A(s,a))) \tag{2}$$

Since this is a deterministic environment, you can use `env.sample_transition(state, action)` to obtain $A(s,a)$ and $r(s,a)$. The functions that you have to implement are in `proj_code/proj_grad.py`.

The full approximate value iteration is shown in Algorithm 1. You will implement the construction of the neural network model in `get_nnet_model` and the training in `train_nnet`. The structure of your model and training scheme may have to be different for this coding project than in the supervised learning case.

You will then implement `value_iteration` (the for loop starting at line 4 in Algorithm 1). For speed concerns, your implementation should batch the computation of $v_\theta(A(s,a))$ instead of performing them individually in the for loop. You can use `flatten` and `unflatten` provided in `proj_code/proj_grad.py` to help you accomplish this.

After each iteration of approximate value iteration, your value function will be compared to the ground truth. **To get full credit, you must have a final MSE of 5.0 or less**. If you are having difficulty achieving this, see the lecture on PyTorch to look up ways to improve your neural network. Your neural network should run on a laptop CPU in 5 minutes or less. Excessively long run-times will be penalized. You can compare your results to the sample output provided on the GitHub under:
`sample_outputs/grad_proj_output.txt`.

Running the code:

```
python run_approximate_value_iteration.py
```

Algorithm 1 Deep Approximate Value Iteration

```
1: $\theta \leftarrow initialize\_parameters()$
2: for $m = 1$ to 50 do
3:     $S \leftarrow get\_training\_states()$
4:     for $s \in S$ do
5:         if s is terminal then
6:             $y_i \leftarrow 0$
7:         else
8:             $y_i \leftarrow \max_a(r(s,a) + v_\theta(A(s,a)))$
9:         end if
10:    end for
11:    $train(v_\theta, X, \mathbf{y})$
12: end for
13: Return $\theta$
```

What to Turn In

Turn in your implementation of to `proj_code/proj_grad.py` to Coding Projects/Graduate Project on Blackboard.