Assignment 2 - Convolutional Neural Network

Course: *Advanced Machine Learning*

January 20, 2019

1 Teammembers and Contributions

- **Bernd Menia**, *01316129*: Report, Programming, Testing, Plots
- **Maximilian Samsinger**, *01115383*: Main Programming, Testing

The difference in our contributions is because Bernd just only started working with Machine Learning, while Maximilian has already prior knowledge in the subject. Thus we worked together, but the main programming work for this assignment was done by Maximilian.

In this month's assignment we had to program the Mountain Car task, as initially described by Moore's PhD thesis in 1990 [Moo90]. Though as for the actual implementation we used the github project from Senadhi as a basis [Sen18]. In contrast to the other assignments we don't have much code for this one, so we will show more of our code snippets in this report.

2 Task i

Ok now listen here dipshits, this is fucking shite.

In the first task we were asked to calculate the action-value function approximation by directly using the states and actions as inputs. For this we first transform the initial state and action into a vector, initialize the weight vector with all 0's and then multiply the two as seen as in (action_value()).

The main part of the code consists of 2 nested for-loops. The outer loop represents each episode of the algorithm while the inner loop represents the steps that are made in each generation. The outer loop chooses the action that will get used to start the episode. At the start this action is dependant on the initially set values like described in the previous paragraph.

From the second episode onwards the action that gets used is chosen by the algorithm, depending on which action is most suitable (choose_action). This is done creating a random number and comparing it to a threshold, epsilon, which we set to 0.02. In other words this means that in 2% of all cases the

algorithm performs a random action. This is done to ensure that even when the algorithm is shifting towards a certain policy and is therefore almost fixed in its actions, there is still the possibility that it performs random actions which might lead to better results. The percentage is purposely low so that the actions still get mainly chosen by the best action calculated. The action gets chosen anew in each step of each episode.

After an action has been chosen the algorithm performs a step. Hereby the actions can be 0, 1 or 2, corresponding to *go left*, *do nothing* and *go right* respectively. The step also calculates the new state and the respective reward of the previous state in combination with the action.

The reward gets calculated as follows: For each step that gets performed and after which the mountain car is not at the goal the reward is -1. On the contrary if after a step the goal got reached the reward is positive, though no exact numbers are given.

The only thing left to do is to update the weights so that the policy gets approximated to the best possible values. As long as the goal hasn't been reached the weights get updated by taking into account the action_value functions of the current- and previous states and also two not yet seen variables, learning_rate and discount. The learning rate is used to decrease the amount of calculations that we have to make. Theoretically in order to get the best possible policy we would have to perform all combinations of sets and actions that exist. However this amount of complexity is not practical and cannot be achieved in a senseful manner. So instead of trying out all combinations of states and actions we choose one as described in choose_action() and then dampen the value of the chosen action by multiplying the outcome with the learning_rate.

The other variable is the discount factor. The discount also acts as a kind of dampening factor, but it is more general than the learning_rate. The exact formula how weight update gets calculated can be seen in X.

3 Task ii

This task is quite similar to the first one. The difference is that

4 Task iii

Finally in this task we compare the results from Task i and Task ii.

5 Discussion

As we have seen we ran into some problems while working on this assignment. First of all using polynomials as the initial values to approximate the action-value function is not good because it is highly likely that these values differ vastly from the actual underlying function that we try to approximate. Of course

this is partly inevitable because by definition we don't know the underlying function yet and therefore have to approximate it. But there are far better methods to start the approximation, for example by using a Neural Network at first and then transferring the output to the Reinforcement Algorithms as the input, which is also the process that was used to train AlphaGo.
Another thing to mention is that our algorithm is unstable. Even after letting the algorithm run for 1000+ episodes the results from the episodes vary strongly as seen by the graph in X. Unfortunately we don't know why this is the case.

The major problem in our code is that our algorithm is prone to unlearning previously learned solutions. It may happen that it discovers a good way to drive upon the mountain, but the solution gets overwritten by the next episodes and therefore unlearns what brought it to the goal.

References

[Moo90] Andrew William Moore. Efficient memory-based learning for robot control. Technical report, 1990.

[Sen18] Vishma Senadhi. Mountaincar-v0. URL: (https://gym.openai.com/envs/MountainCar-v0/, August 2018.