703642, Advanced Machine Learning course 21st January, 2019

**Assignment 3**

**Programming Project: Reinforcement Learning**

Kamal Zakieldin, Yusuf Ipek

1. **Introduction**

Reinforcement Learning is an aspect of Machine learning where an agent learns to behave in an environment, by performing certain actions and observing the feedback which it gets from those actions, to build a policy that can act appropriately according to these observations.

In this report we are describing our methods we have applied to the well-known **Mountain Car** problem [1] and our experiments we have made. First, we introduce our problem and our tuning showing the experiments we had. Afterwards, we discuss our chosen parameters and the results we got. Furthermore, we compare the results we got and finally give some recommendations for future tuning.



1. **Problem description**

The goal is to drive the car up the mountain on the right; however, the car's engine is not strong enough to scale the mountain in a single pass. Therefore, the only way to succeed is to drive back and forth to build up momentum.

The agent can perform three different actions; move left, move right or do not move.

Figure1: Mountain Car environment

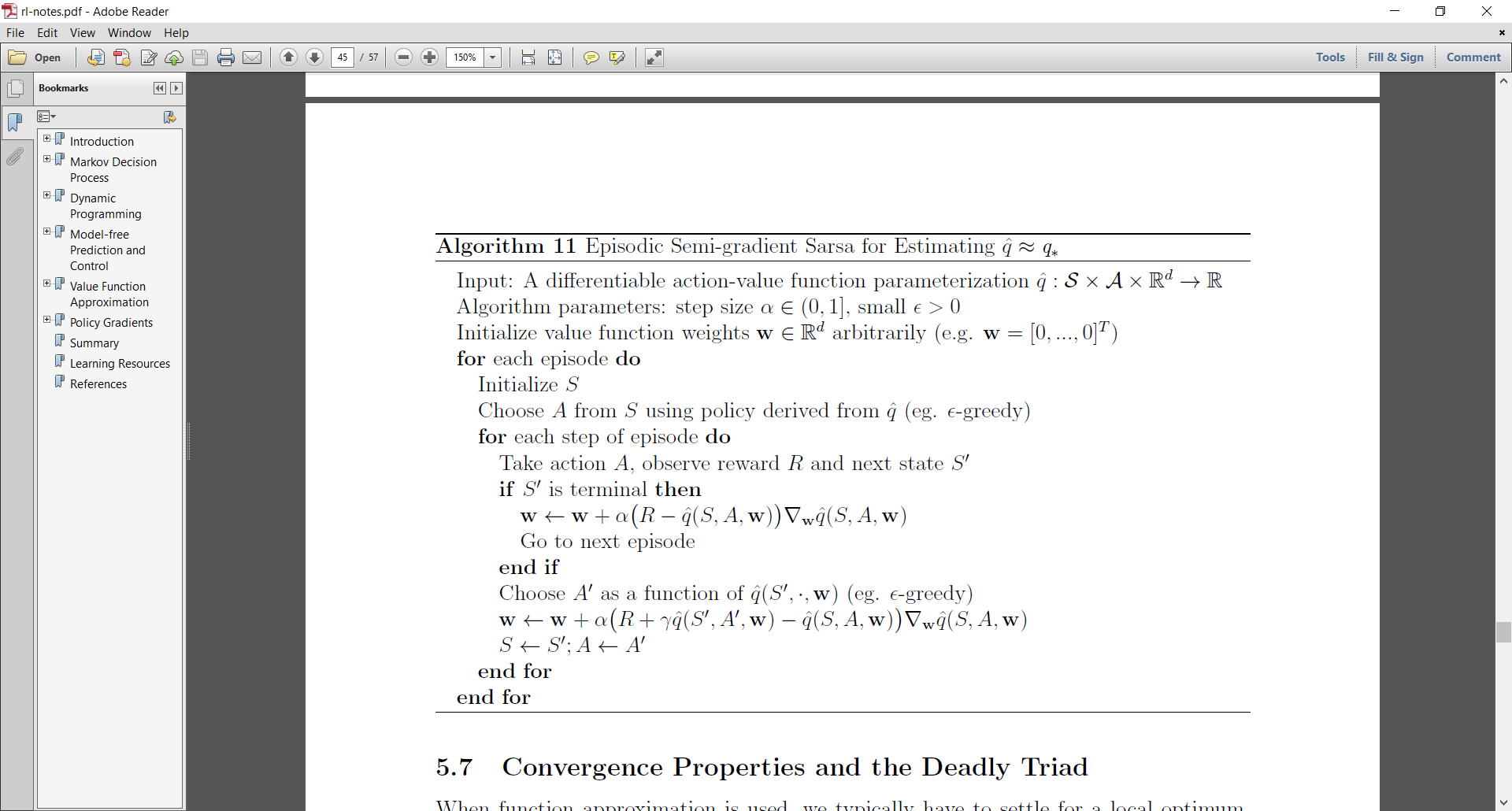
1. **Experiments**

At the beginning we started to try tabular algorithms for this problem, we have tried **tabular** **q-learning** algorithm and **tabular SARSA** algorithm, but after performing many tries with different parameters, such as changing the learning rates, exploration greedy (epsilon), but the agent could not learn the appropriate policy.

One of the main reasons for that was the rewarding system, giving a **-1** as a reward after every step was not a great indication, the more you spend iterations the more punishment you get. Theoretically this is a good approach, but because the agent never receives a positive reward and was not able to figure out the optimal policy till it explores the goal.

This raised the conflict of **the exploration versus the exploitation** between doing more exploration to search about the goal or be greedy and reduce the number of iterations to reduce the negative rewards. Without exploration, you will never know what is better ahead. But if it is overdone, we are wasting time. So, we have an exploration policy, like epsilon-greedy, to select the action taken in step 1. We pick the action with highest *Q* value but yet we allow a small chance of selecting other random actions. *Q* is initialized with zero. Hence, there is no specific action standing out in early training. As the training progress, more promising actions are selected and the training shift from **exploration to exploitation**.

It totally makes sense that **tabular methodologies** did not work, because tabular methodology works fine with small problems with small environment, although our environment does not look so complex, but it is not small and tabular methods doesn't achieve good results with it. thinking deeply about our problem space, we have only three actions to get observations about the state (the position and the velocity of the agent).

That's why combining the velocity and the position together generate a large environment that needs **function approximation methodologies** parameterized by some weight vector.

In these methodologies, the RL algorithms updates the parameters (weight vector) during the exploration phase. we are required to use a linear value-action function approximation.

First, we tried the **semi-episodic** algorithms to solve the problem without using any neural network architecture as a linear method.

Figure 2: Episodic SARSA algorithm

We have tried many different methods to construct the feature vector, first we tried the logistic function to build the features ( **ϕ(S) = Sj** , when S is the state and j = 0, ...., n ), however this feature representation did not give us good results, so we have tried the  **Fourier basis method**[2] which leads to an expansion in sinusoidal functions.

We have faced some problems during constructing the feature vector **ϕ**, that's why, we decided to implement another approach thinking about supervised machine learning by using a **neural network** with one linear layer to build the agent.

We use supervised learning to fit the *Q*-value function. In RL, we search better as we explore more. So, the input space and actions we searched are constantly changing. In addition, true labels are not available, hence we substitute it with Rt + ɣ Q(St+1,At+1,w) as the target value as well as the label for our neural network turning the problem into a supervised learning problem solvable using gradient descent.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Model | epsilon | Testing accuracy |
| a) | DQN with 200 units | 0.2 | 50% |
| b) | DQN with 200 units | 0.3 | 53% |
| c) | DQN with 300 units | 0.3 | 58% |
| d) | DQN with 500 units | 0.3 | 62% |
| e) | SARSA with 300 units | 0.3 | 60% |
| f) | SARSA with 500 units | 0.2 | 72% |
| g) | SARSA with 500 units | 0.3 | **75%** |

We have tried different models with deep Q-network algorithm but we could not get an optimal policy as q-learning doesn’t converge in linear methods, so we tried deep SARSA network algorithm, with different number of units in the hidden layer and different epsilon value, table 1 shows the results we got.

Table 1: different models for a one hidden layer NN using deep Q-network (DQN) models ( a – d) and deep SARSA-network algorithm models (e – g).

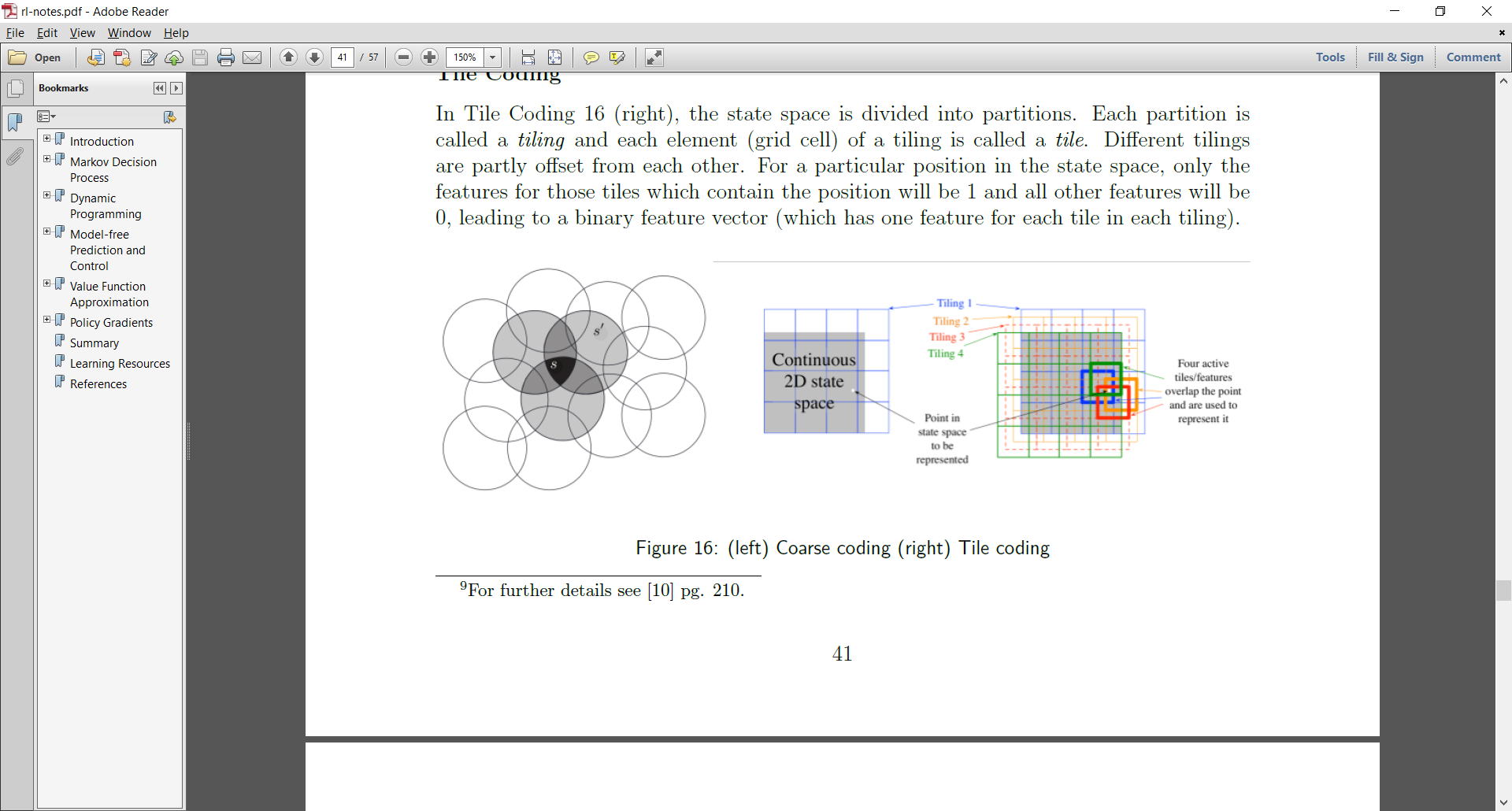
It is important to notice that increasing the number of hidden layer units have made a better performance increasing the number of total successful episodes, the more units in the hidden layer the more successful episodes we got.

During this experiment we have some parameters to tune, one of the most vital parameters that needed to be tuned is the exploration **epsilon** greedy parameter, choosing the epsilon high will make the agent act greedy with less exploration, in such a problem, exploration is critical and without making enough exploration the agent may not find the goal at all.

We also tried to adjust the epsilon after each successful step, by slightly reducing it after each successful episode (to act less greedy) and surprisingly this increased the number of successful episodes in the training phase but reduced slightly the number of successful episodes in the testing phase. on the other hand, increasing the epsilon after every successful episode made the agent more greedy and reduce the exploration, hence, the number of successful episodes in the training phase have decreased dramatically, but in the testing phase, it has increased much more.

It was clear that q-learning may not converge. although SARSA has achieved a higher accuracy, **SARSA did not also reach the optimal policy but chattered around it**.

Finally, we have tried to polynomial methods using the **tile coding** method. and surprisingly, it has shown a very good progress, although it may take longer time for training, but overall it does not need the same number of episodes to find the goal for the first time, and after few episodes it can converge.

Using semi-episodic algorithm has achieved a real convergence if we compared with earlier experiments, as we will see in section 4.

We have used a ready code doing tiling by Richard S. Sutton[3] , the idea behind tiling is just dividing the space into number of partitions, we have chosen 16 partitions as recommended from the lectures, **each partition is a tiling**, and element in the tiling is a tile, we are creating the features of each partition by combining the position and the velocity of the object to have a feature represents these tiles in its tiling partition.

Figure 3: Tile Coding

One important thing is that tiling is only a map from (state, action) to a series of indices, it doesn't matter whether the indices have meaning, only if this map satisfy some property. after trying some iterations, we found that our space is small to do a tiling for 16 partition, so we have tried 8 partitions and it has shown better results in finding a feature vector to represent it.

1. **Results and comparisons**
2. **References**

[1] <https://gym.openai.com/envs/MountainCar-v0/>

[2] <https://en.wikipedia.org/wiki/Basis_function#Fourier_basis>

[3] <http://incompleteideas.net/tiles/tiles3.html>

[4] <https://tomaxent.com/2017/07/05/On-policy-Prediction-with-Approximation/>

[5] <https://github.com/vmayoral/basic_reinforcement_learning>