

**Department of Business Administration**

**Chair of Quantitative Business Administration**

**Seminar project Deep Reinforcement Learning**

**Programming the game 2048 with methods of Deep Reinforcement Learning**

**Submitted by Group ShaLuMa:**

Lutharsanen Kunam, 15-700-875

Matteo Brändli,

Shabarna Chandrabala, 15-734-650

**Supervisor:** Dr. Robert Leonard Earle

**place, submission date:** Zurich, 12.12.2019

Table of content

[0. Contribution of the group members to the project 1](#_Toc26969630)

[1. Project- programming 2048 with deep reinforcement learning 1](#_Toc26969631)

[1.1 Our reinforcement problem 1](#_Toc26969632)

[1.2 Q-Learning 2](#_Toc26969633)

[1.2.1 Pre-test-runs 2](#_Toc26969634)

[1.2.2 Q-learning test series: experimental design & results 4](#_Toc26969635)

[1.2.3 Q-learning test series: comparison 5](#_Toc26969636)

[1.2.3 Q-learning test series: conclusion 5](#_Toc26969637)

[1.3 SARSA 6](#_Toc26969638)

[1.3.1 First pre-test-run 6](#_Toc26969639)

[1.3.2 Second pre-test-run 7](#_Toc26969640)

[1.3.3 Third pre-test-run 7](#_Toc26969641)

[1.3.4 Fourth pre-test-run 8](#_Toc26969642)

[1.3.5 Fifth pre-test-run 8](#_Toc26969643)

[1.3.6 Full SARSA test series 8](#_Toc26969644)

[1.3.7 Comparing Q-learning and SARSA 9](#_Toc26969645)

[1.4 Deep Q-Learning 10](#_Toc26969646)

[1.4.1 First test-run 10](#_Toc26969647)

[1.4.2 Second test-run 11](#_Toc26969648)

[1.4.3 Third test-run 11](#_Toc26969649)

[1.4.4 Fourth test-run 12](#_Toc26969650)

[1.4.5 Fifth test-run 12](#_Toc26969651)

[1.4.6 Full DQN test series 13](#_Toc26969652)

[1.4.6.1 Varying alpha 14](#_Toc26969653)

[1.5 Conclusion 15](#_Toc26969654)

[2. Question 1 16](#_Toc26969655)

[3. Question 2 17](#_Toc26969656)

[4. Question 3 18](#_Toc26969657)

[6. Bibliography 19](#_Toc26969658)

# 0. Contribution of the group members to the project

|  |  |
| --- | --- |
| Lutharsanen Kunam |  |
| Matteo Brändli |  |
| Shabarna Chandrabala |  |

# 1. Project- programming 2048 with deep reinforcement learning

## Our reinforcement problem

Our objective was to program an algorithm, which solves the game 2048 with Reinforcement Learning methods learned in class. We used python to program. The goal of the game 2048 is to combine two same-value numbered tiles, so that they merge and result in their sum, to finally attain the tile 2048. However, in the game it is possible to attain higher numbers. Nonetheless, after attaining 2048 the player has won the game.

For our project, we have used three methods, namely Q-Learning, SARSA and Deep Q-learning to implement the learning. We used an open AI environment from gym and installed the version gym-2048. This environment has an output of a state space with a 4x4 matrix. We have mostly transformed this 4 x 4 matrix into a list with 16 elements to make it easier to use them in the model. The discrete action space consists of four elements for all the states, even if a particular action would be unfeasible in a specific state. We had to control for this in our tabular models, but the Deep-Q learning appeared to avoid this problem by itself. The four different actions possible were: swipe up, swipe right, swipe down and swipe left. The reward was given by the environment as the sum of merged cells in a step. The goal of our models is to maximize the sum of the rewards to solve the 2048 task. We ran multiple versions of code with very mixed results, even in terms of getting software and code to work.

To interpret the learning process, we are using three different graphs and plots. We will divide all our episodes into batches corresponding to percentiles (always 100 batches). We expect to see an improvement from one batch to the next batch. With our plots we will analyse the win statistics. In each test-run we will define a winning rate - we are going to set a tile number as a goal and count how often the model has achieved this goal. The goal was set at 256 due to the computational needs of the tabular models. Other plotted statistic will present the maximum value tile reached in a batch, the average maximum tile reached in a batch and the average rewards obtained by the interaction with the environment. In all plots, we expect an improvement of the calculated values, especially for the “average reward” statistic (which is actually being optimized by the models). As a comparison for performance we had an agent act randomly at any point. If they fail to outperform such an agent our models my not have learned much.

## Q-Learning

The first model we used to solve the game 2048 was Q-learning. In our opinion, it was theoretically a well-suited model for the 2048 problem. The reasons are that both reward and transitions can be observed and that there is a natural limit to the reward due to the rules of the game. As a result, a Q-Table can be computed and one should not have to worry too much about discounting. We thus chose discounting at gamma=0.99 to explore the foresight without completely neglecting possible loops.

We started to implement the Q-Learning model by adding a json file to the model, where it should store the Q-table. Due to type issues we stopped this version. At the end, we implemented the Q-table similarly to SARSA. In fact, both models are almost identical but for the updating function. Our goal is to compare both approaches and see whether one is more suitable to our 2048 environment.

Many were originally taken from the(add hyperlink later…) Much of the code has been modified to cope with our environment and to visualize the data. In the process of developing the Q- model our agent got continuously stuck in the same state where no step is activated (due to the supposed “optimality” of that move by Q considerations) and could not overcome this problem with the computational power at hand. This has been overcome by the choseandcheck function that tracks if the state changed after an action and reduces the corresponding Q value directly in case of an inactive move. Like this we avoid infinite loops and we reused this code snippet for the SARSA model in the next chapter. The function checks, if the state changes and if it doesn’t, it reduces the Q-value. With this function we want to make sure, that our model will avoid these steps in future and thus avoid dead-ends.

### Pre-test-runs

In our first test-run we run 5’000 episodes. This means that each batch contains 50 episodes. We have used alpha as the learning rate with a value of 0.2, as gamma we have used 0.99 and epsilon, which is the exploration rate, decreases from 1 to 0.0002 over time. Our goal in this test-run was 256. Due to a bug, we couldn’t produce the average score in this test-run.

|  |  |
| --- | --- |
| Figure 1:maximal values statistic | Figure 2:winning-statistic |

Following our first test-run we can see, that there is a positive correlation between the batches and the winning statistic. This shows as that our model improves itself from game to game. In our next try, we will raise the goal from 256 to 512, to see, if our model can attain higher tiles. The y-axis of Figure 2 stands for the games won.

In our second test-run we decreased alpha to 0.1 to see what the effect of a lower alpha is on our model. Intuitively, we would say that a lower learning rate should have a negative impact on the model. But we know from lecture, that there is an issue of overfitting and therefore we want to avoid to overfit our model. We also increased the goal to 512 to see, if our model can achieve this goal too.

|  |  |
| --- | --- |
|  |  |

Unfortunately, we didn’t have much success by increasing the goal to 512. The model never reached this value and therefore the winning statistic didn’t make any sense in this test-run. The maximal value from this test-run was 256. In our opinion, the *graph with the average of rewards* is a suited graph to measure how well the model is able to learn, especially the slope of the graph shows that the learning effect is increasing. The regression line of the figure *average of maximal values* shows a constant line. However, it can be clearly seen, that the points are randomly distributed, and the line doesn’t represent them too well.

In our third test-run, we increased our alpha to 0.3. Therefore, we wanted to explore the trade-off of a higher or lower learning rate. We have decreased the goal back to 256 to have a better view on how t how often the model reaches the goal. Due to runtime time restriction, we weren’t able to increase the episodes to an amount, which reaches constantly 512.

|  |  |
| --- | --- |
|  |  |

An increase of the learning rate to 0.3 had an enormous impact on the slope of the average rewards graph. On average the reward increases per batch by over 3 units. This is an increase of over 100% compared to the graph with a learning rate of 0.1. We also see an increase of the percentual win rate over the batches. The maximal value, which was achieved by the model was 256.

In the fourth test-run we decreased again the learning rate to 0.01 to verify our hypothesis, that the scope of the reward graph will decrease, if we decrease our alpha. There was an additional exception, as we were running only 4000 episodes instead of 5000. All the other parameters stayed the same.

|  |  |
| --- | --- |
|  |  |

As we have expected, the slope of the reward graph decreased in comparison to the third test-run due to a lower learning rate and a lower range of episodes. Also, the slope of the winning statistic is 3 times smaller than in the previous test-run. The maximal value, which was achieved in this test-run was as expected 256, because, it performs worse than in the previous test-run.

In the fifth test-run, we once again increased the learning rate alpha to 0.5. The maximal value of the test-run was still 256.We expect to have a higher slope in the rewards graph and therefore to have the best possible result so far.

|  |  |
| --- | --- |
|  |  |

As expected, this test-run outperformed all the previous test-runs. The slope of the reward graph is quite steep, and you could even see a positive correlation between the batches and the reward without a regression line. The reward increases per batch on average by 5 units.

### 1.2.2 Q-learning test series: experimental design & results

In the github folder Q learning you will find 6 specifications, varying along two dimensions.

There are two epsilon schemes; whereas one is a monotonously linearly decreasing epsilon, floored at 0.01 the other scheme is a discontinuous epsilon development with multiple resets of epsilon to a higher level. This should reduce the risk of ending up at a non-optimal solution. The other varying dimension are the alphas, chosen at 0.1, 0.5 and 0.8. Thus there are 2x3=6 specifications for testing and comparison. After 5000 training episodes, there are performance tests with 500 epsiodes. Results are always expressed in percentiles, thus the average result over the percentiles (100 datapoints, in test mode 5 episodes are pooled to one datapoint)

Q learning summary table of parametrizations and main performance measures after 5000 training episodes[[1]](#footnote-1)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| alpha | epsilon | Win percentage | Avg max tile | Avg reward |
| Alpha = 0.1 | Monotonous | 0.044 | 86.99 | 1017.93 |
| Alpha = 0.1 | Nonmonotonous | 0.067 | 93.18 | 1037.97 |
| Alpha = 0.5 | Monotonous | 0.094 | 108.61 | 1256.05 |
| Alpha = 0.5 | Nonmonotonous | 0.115 | 107.11 | 1240.44 |
| Alpha = 0.8 | Monotonous | 0.13\*, \*\* | 109.64\* | 1273.13\* |
| Alpha = 0.8 | Nonmonotonous | 0.10\*, \*\* | 109.42\* | 1260.52\*, \*\* |

\*The automatic average performance calculations have been added in time for the models with alpha = 0.8. Those numbers do not contain compounded rounding errors.   
\*\*These estimates differ from the manual calculations, by -0.02 (monotonous win %) and +0.01 / -0.23 (non-monotonous win % / avg reard). The effect on the main variable avg reward is negligible, whereas the win % may have lost some comparative power as statistic.

### 1.2.3 Q-learning test series: comparison

Non-mono, varying alpha 🡪 alpha =

Mono, varying alpha 🡪

Overall 🡪 alpha = XX seems to outperform

Alpha = 0.1, varying epsilon scheme 🡪 XXX epsilon scheme seems superior

Alpha = 0.5, varying epsilon scheme 🡪 XXX epsilon scheme seems superior

Alpha = 0.8, varying epsilon scheme 🡪 XXX epsilon scheme seems superior

Overall 🡪

### 1.2.3 Q-learning test series: conclusion

## SARSA

We used SARSA, because our environment fits to the requirements of it. The reason is, that our environment has a limited space of states. Therefore, it is possible for the model to know each state at one point and calculate the perfect action for the given state.

Our initial goal was to program a model, where it was possible to attain the tiles up to . However, due to missing disk memory and computing power of our machines it wasn’t possible. Therefore, we lowered the highest attainable tile to , which is equal to 256. We wanted to store the whole space of states in a text file, but the file, where we saved our data, was bigger than 40 GB! As mentioned before, in the original version of 2048, it is possible to attain higher number than 2048. As a template we have used a tutorial from the youtuber Machine Learning with Phil (Machine Learning with Phil (2019b)). He has created a SARSA model for the game carpole-v0, which is also set in an open AI gym environment. We adjusted his code to our environment, which was in our view more complex than his environment. We have also added a get\_max () function and some plots, so the analysis of the results can be more profound.

Compared to the version of Phil, we dynamically updated our q-table. We started to generate the whole Q-table at the beginning too, but we never managed to fully create it. Therefore, we started to create a list of all possible states and a Q-table dynamically. The SARSA model uses the choseandcheck function too, which checks if the model gets stuck somewhere. In order to make the models SARSA and Q-Learning more comparable, we used the identical hyperparameters and the same size of batches, goals and episode to run the tests.

### 1.3.1 First pre-test-run

Similar to the Q learning model, we first started with an alpha of 0.2, a gamma of 0.99 and an epsilon, from 1 to 0.0002. The goal was still to attain 256. Comparing with the Q-Learning model, it can be concluded that, there is a positive correlation between the batches and winning statistics too, like in the Q learning model. Therefore, the model was improving from game to game. It never reached values above 256, however the next step of our model testing would consist of increasing the goal to 512.

|  |  |
| --- | --- |
| Ein Bild, das Screenshot enthält.  Automatisch generierte Beschreibung  Figure 3: Max-statistics | Ein Bild, das Screenshot enthält.  Automatisch generierte Beschreibung  Figure 4: Win statistics |

If we compare the win statistics model from Q-Learning and SARSA, we see that Q-Learning performed slightly better than the one from SARSA. Regarding the max value statistic, both models performed equally well.

### 1.3.2 Second pre-test-run

In our second test-run we also decreased the learning rate alpha to 0.1. Similarly, to the Q-learning model, we increased the goal to 512 too.

|  |  |
| --- | --- |
|  |  |

Like in the Q-learning model there is a positive correlation between the number of the batches and the average reward per batch. With these hyperparameters and graphs, we see that the SARSA model performs better than the Q-learning model. One indicator is the higher slope of the reward graph and the other indicator is the positive slope of the average of maximal values graph.

### 1.3.3 Third pre-test-run

In the third test-run, we set the learning rate alpha to 0.3. The goal was set back to 256.

|  |  |
| --- | --- |
|  |  |

Looking at the figures, it becomes clear, that the reward and the win statistic have increased. The average in rewards have increased almost twice as much as in the second test-run. The Win-statistics doubled in comparison to the first test-run. In comparison to the Q-Learnings, SARSA slightly better in the Win Statistics and Average of Rewards.

### 1.3.4 Fourth pre-test-run

We lowered the alpha to 0.01 and did a test-run with 4000 episodes. Clearly the rewards were almost three times lower than in the third test-run. The same applies to the win statistics. Comparing with the Q-Model, the average of Rewards was slightly better and the Win statistics slightly worsened in the SARSA model.

|  |  |
| --- | --- |
|  |  |

### 1.3.5 Fifth pre-test-run

As lowering the alphas did not improve the performance of the model, we raised the alpha again to 0.5. We did a test-run with 5000 episodes. Again, the rewards were increasing quicker with the number of episodes than in fourth test-run. The same applies to the win statistics. The reward increased per batch on average by 5 units. In comparison to the Q-Learnings, SARSA identically concerning the average of rewards, however had superior win statistics.

|  |  |
| --- | --- |
|  |  |

### 1.3.6 Full SARSA test series

In the github folder SARSA you will find 6 specifications, varying along two dimensions. There are two epsilon schemes; whereas one is a monotonously linearly decreasing epsilon, floored at 0.01, whereas the other scheme is a discontinuous epsilon development with multiple resets of epsilon to a higher level. This should reduce the risk of ending up at a non-optimal solution. The other varying dimension are the alphas, chosen at 0.1, 0.5 and 0.8. Thus, there are 2x3=6 specifications for testing and comparison. After 5000 training episodes, there are performance tests with 500 epsiodes. Results are always expressed in percentiles, thus the average result over the percentiles (100 datapoints, in test mode 5 episodes are pooled to one datapoint).

SARSA summary table of parametrizations and main performance measures after 5000 training episodes (we forgot to specify those measures in the code and thus extrapolated the average of the test performance as intercept + 50\*slope of each statistic in the performance test. Rounding errors will occur but are omitted from consideration)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| alpha | epsilon | Win percentage | Avg max tile | Avg reward |
| Alpha = 0.1 | Monotonous | 0.034 | 87.73 | 996.02 |
| Alpha = 0.1 | Nonmonotonous | 0.046 | 95.07 | 1030.42 |
| Alpha = 0.5 | Monotonous | 0.105 | 111.10 | 1299.20 |
| Alpha = 0.5 | Nonmonotonous | 0.151 | 115.94 | 1332.19 |
| Alpha = 0.8 | Monotonous | 0.076 | 110.00 | 1266.85 |
| Alpha = 0.8 | Nonmonotonous | 0.090 | 105.29 | 1233 |

Considering the alpha, which is the learning rate, the model with the hyperparameter 0.5 outperformed the other ones. So, we can assume that the global maximum of the SARSA model could be at the alpha level of 0.5.

Comparing the results of the nonmonotonous epsilon decay and the monotonous epsilon decay, nonmonotonous epsilon decay outperforms monotonous epsilon decay, where learning state is increasing and, in a state, where the learning state is decreasing the monotonous epsilon decays the nonmonotonous epsilon. It is difficult to make a conclusion about the alpha level 0.5. In our experiment it is the global maximum, but if there was more time, there is a possibility that there is a global maximum at a 0.65 alpha level and there, there is no difference between a nonmonotonous and a monotonous epsilon decay. Due to time restrictions we unfortunately couldn’t test this hyptothesis.

Thus, we can conclude that the learning rate alpha has a much bigger influence on the result of the model than the decay of epsilon.

### 1.3.7 Comparing Q-learning and SARSA

## Deep Q-Learning

Deep Q-Learning uses neural networks to approximate the Q-value. In contrast to Q-Learning, only the state is given as an input. Nonetheless, the Q-value of all possible actions is the outcome (Choudhary, A. (2019)). This is the reason, why we chose Deep Q-Learning as one of the methods for our project.

We programmed the Deep-Q learning model with Keras including an Adam optimizer. Once again, we used the tutorial of the youtuber Machine Learning with Phil (2019a) as a template. He created a Deep Q learning model for the open gym ai environment called LunarLander v2. This environment was less complex than ours and can be solved in less than 500 episodes. The neural network, which Phil created suited really well to our environment. We just had to transform the 4 x 4 matrix into a list to be able to use the deep q network of Phil. Later, we even realised that we forgot to apply a function, which was similar to choseandcheck form the Q-learning/SARSA model. But during our test-runs the model got never stuck in a certain loop. We assume that the neural network learned it by itself.

### 1.4.1 First test-run

In our first test-run with 8000 episodes, our model achieved an impressive value of 512. The learning-rate alpha was 0.0005 and epsilon decay was 0.096. The epsilon started at 1 and decayed till 0.01. To plot this test-run, we used the graph from Phil. While this plot worked for his model, it gets less clear if our DQN really learnt during these 8’000 episodes. We can see a slow positive scope however there is too much variation in the data. Therefore, we are going to use our plots again for the next test and once again divide the episode to 100 batches. Therefore, the learning becomes more evident,

|  |  |  |
| --- | --- | --- |
| Ein Bild, das Screenshot enthält.  Automatisch generierte Beschreibung  ep. 500 | ep. 4000 | :ep. 8000 |

### 1.4.2 Second test-run

Remarkable is that the updated DQN version was much quicker than the SARSA model. Almost as twice as quick. With a goal of 2048, an alpha of 0.025 and 5000 episodes the model didn’t do that well either. There was a slightly negative correlation between the batch and the reward of the batches. The highest value the model hit was 256, however only once. Mostly 32 and 64 were hit. The performance was therefore quite poor.

|  |  |
| --- | --- |
| Ein Bild, das Screenshot enthält.  Automatisch generierte Beschreibung | Ein Bild, das Screenshot enthält.  Automatisch generierte Beschreibung |

As the results were not satisfying, we still made another test-run with the same parameters. However, this time, the model performed slightly better. The model mostly achieved tiles of 128 and 256.Nonetheless, the regression line shows that the batch and reward per batch were still negatively correlated.

### 1.4.3 Third test-run

As we weren’t satisfied with the results, we searched for projects, which implemented DQN for the game 2048. We found the paper of *Kaundinya, V., Jain, S., Saligram, S., Vanamala, C.K, Avinash, B. (2018),* who did a similar analysis as we did. In this fourth test-run we used the parameter of this paper, to see if their parameter would enhance our model. Therefore, we defined alphas as 0.009 and gamma as 0.9. The number of episodes remained 5000. Similar to the second part of the third test-run, there is negative correlation and maximal value where either 128 or 256. However, comparing the reward statistic, with the second test-run, it was only half as high.

|  |  |
| --- | --- |
| Ein Bild, das Screenshot enthält.  Automatisch generierte Beschreibung | Ein Bild, das Screenshot enthält.  Automatisch generierte Beschreibung |

### 1.4.4 Fourth test-run

Comparing the last few test-runs we made an assumption, that a lower alpha resulted in a better performance. Therefore, we defined an alpha of 0.005 for the fourth test-run. Again running 5000 episodes. The maximum goal was set at the tile 256. Interestingly, the model run really slowly, however the reward static was higher than the previous two test-runs.

|  |  |
| --- | --- |
|  |  |

### 1.4.5 Fifth test-run

To confirm our assumption that a lower alpha did better the model, we did a fifth test-run with an alpha of 0.001, 6000 episodes and a maximum goal of 512 tiles. Our assumption was validated again. The reward statistic shows a steeper line, than in the fourth test-run. Therefore, the positive correlation between the batch and reward per batch is higher. A possible explanation for the better performance with a lower alpha is overfitting.

|  |  |
| --- | --- |
|  |  |

### 1.4.6 Full DQN test series

Given that we do not have access to a personal GPU, we did the calculations for our Deep-Q learning on Google Colab. In the workspace folder shared below you will find 9 specifications of jupyter notebooks, varying along three dimensions: epsilon decay, alpha and the dimensions of the neural network. Enter COLAB LINK

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Model | Training | Neural dims | Epsilon decay | Alpha | Win % | Avg max tile | Avg reward |
| Best SARSA |  |  |  |  |  |  |  |
| Best Q | Comparison… |  |  |  |  |  |  |
| DQN1 | 5000 | 256,256 | 0.096 | 0.001 | 0.08 | 103.14 | 1127.57 |
| DQN2 | 5000 | 256,256 | 1/5000 | 0.0001 |  |  |  |
| DQN3 | 5000 | 256,256 | 0.096 | 0.0001 |  |  |  |
| DQN4 | 5000 | 256,256 | 1/5000 | 0.001 |  |  |  |
| DQN5 | 5000 | 64,32 | 1/5000 | 0.0001 | 0.33 | 156.41 | 1694.18 |
| DQN6 | 5000 | 64,32 | 0.096 | 0.0001 |  | 156.74 | 1703.26 |
| DQN7 | 5000 | 64,32 | 0.096 | 0.001 | 0.07 | 93.33 | 1039.31 |
| DQN8 | 5000 | 64,32 | 1/5000 | 0.001 | 0.10 | 102.66 | 1107.65 |
| DQN9 | 5000 | 8,4 | 0.096 | 0.0001 |  |  |  |
| DQN10 | 5000 | 8,4 | 1/5000 | 0.0001 |  |  |  |
| DQN11 | 5000 | 8,4 | 0.096 | 0.001 | 0.02 | 75.02 | 841.13 |
| DQN12 | 5000 | 8,4 | 1/5000 | 0.001 | 0.06 | 91.36 | 1041.59 |

Given our finding above that alpha= 0.001 tends to do better, we incorporated a smaller alpha equal to 0.0001 which even tended to do better. After 5000 training episodes, the further reduced alpha seems to be superior in terms of our performance measures. Maybe this decrease by a factor of 10 could slow down the learning too much and result in underfitting, as presented in class. Nevertheless, the strong performance of this parametrizations does not support this idea yet. Maybe this could become more evident after more training.

### 1.4.6.1 Varying alpha

After 5000 training episodes, the further reduced alpha seems to be superior in terms of our performance measures. Maybe this decrease by a factor of 10 could slow down the learning too much and result in underfitting, as presented in class. Nevertheless, the strong performance of this parametrizations does not support this idea yet. Maybe this could become more evident after more training.

Note: DQN does not immediately get out of “unfeasible move loop” itself! Had to randomize multiple times

Non-mono, varying alpha 🡪 alpha =0.5 seems to outperform the others

Mono, varying alpha 🡪

Overall 🡪 alpha = XX seems to outperform

Alpha = 0.1, varying epsilon scheme 🡪 nonmonotonous epsilon scheme seems superior

Alpha = 0.5, varying epsilon scheme 🡪 nonmonotonous epsilon scheme seems superior

Alpha = 0.8, varying epsilon scheme 🡪 nonmonotonous epsilon scheme seems superior

Overall 🡪

Compare SARSA and Q learning results. Tendency for superiority?

## 1.5 Conclusion

To conclude, we have used three model for our Reinforcement problem. By mostly updating the alphas in every test-run, we saw significant difference in the performance of the model. As the alpha is equal to the learning rate of the model, we however saw contrary results in the single models. The highest attained tile was throughout 256. Except in the first test-run of the model DQN, we attained the tile 512.

Comparing the two models Q-Learning and SARSA, we have made the following conclusions: In the Q-Learning model a higher alpha resulted in a better performance of the model. The same

deduction can be applied to the SARSA model as well. However, comparing these two models, SARSA outperformed the Q-Learning model most of the times. A reason could be, that SARSA incorporates the effect of exploration on the Q value estimates. This adequately fits our reinforcement problem. Another one is that, SARSA includes effects of exploration, when updating Q-Values. Therefore, disasters can be decreased considerably than with the Q-Learning method. In contrast to these two models, the DQN performed much better with a lower alpha. It could be a consequence of overfitting. Put in other words, as there are too many “irrelevant” variables, the training is too big, and it overshoots. Therefore, a lower alpha improves the performance.

Unfortunately, our computers didn’t have the power to compute with a higher number of episodes. As an outlook, a continuation of this project could consist of increasing the number of episodes to see the consequences.

# 2. Question 1

*Larry, Moe, and Curly are fighting as usual, but this time about reinforcement learning. Larry claims that whether exploring or exploiting, learning can always occur as long as the agent encounters something unexpected. Moe shouts, “Larry, you are wrong as usual! Learning can occur whether expectations are met or not! But only during exploration!” Curly yells, “No! You are both wrong! Learning can occur under any circumstances, but you have to expand the state space!” At this point, the police arrive, arrest them for disturbing the peace, and take them to the judge. Please tell the judge who (of Larry, Moe, and Curly) is right about what and why.*

A problem, which the agent has to face in Reinforcement Learning is the trade-off between trial and error. The agent learns through trial and error, put in other words with the interaction with the environment. In order to maximize the rewards, the agent has to prefer actions, which he has tried in the past and which were effective in attaining reward. However, to discover such actions, the agent has to explore other strategies, which he hasn’t tried before. In other words, on the one side the agent has to exploit his knowledge from past experience to maximize rewards. On the other side he has to explore the environment to find better actions, which can maximize the rewards. Therefore, the agent can’t learn only through exploration or only through exploitation without failing. During exploration, where he tries various actions, he needs to think forward and be able to favour the one which appear to be the best. To conclude, Moes’s claim is wrong. As the agent needs to find the balance between exploitation and exploration in order to achieve the best learning (Sutton, R.S. and Barto, A.G. (2017)).

Curly’s claim is wrong too. Expanding the state space can be even problematic. The larger the space, the longer the time needed understand the environment and learn. It is almost impossible find the “perfect strategy” with Q-Learning method, when the state space is large. Traying all state spaces will cost a lot of time (Hu, J. (2016)).

Therefore, Larry’s claim was right. It fits the above-described explanation of Reinforcement Learning adequately.

# **3. Question 2**

*Consider a dynamic problem with states, actions, and rewards. After a random number of episodes (say 5,000 to 10,000), the parameters controlling the problem go through a regime shift. So, for example, in the box world this might mean the rewards would change, and/or the transition probabilities would change. There are a finite number of regimes that can occur, but which regimes occur is dependent on a probability distribution. Is it possible to come up with a modified Q-Learning algorithm that could develop a good policy for the dynamic problem? If yes, please describe such an algorithm. If no, describe why it is not possible.*

Q learning predicts the best action to maximize the cumulative rewards. Q learning is already an iteration, which evaluates his strategy step-by-step after each action. However its values iteration is limited. At least the states are known to compute the Q -table.

For Q-learning and are not needed for the computations. The learning rate is updated based on the transition we have seen. Therefore, a modified Q-Learning isn’t needed, to evaluate the change in rewards or transition probability.

# 4. Question 3

*Please explain in terms your grandmother/9-year-old nephew/CEO can understand the differences and similarities between supervised learning and reinforcement learning.*

Supervised learning and reinforcement learning are both paradigms of machine learning. There are three paradigms in total: supervised learning, unsupervised learning and reinforcement learning. In machine learning you use computer algorithms, which is a piece of logic written in computer code that the computer can compile(understand) it. The algorithms of machine learning are based on statistical or mathematical model, which maximizes a certain value or search for patterns or similarities. All algorithms of machine learning have a high computing intensity and regularly you run the code much longer than a usual computer script to get valuable results. Both methods use neural networks, if it is required.

The difference between supervised learning and reinforcement learning is that supervised learning receives two data sets, which has two types of data. The first type is a x-value and the second type is a y-value. Each x-value is connected with a y-value. Then there are two types of sets: a training set and a test set. The data from the training set you use to train the model. By giving the model a big set of training data, it can easily figure out the relationship between x and y. Then you use the test data to test your model, if it learned appropriately. You always predict a y-value using a x-value from the test set. Then you compare the predicted y with the real y from your test set and calculate an accuracy. The accuracy should rate your model and describe how precise it recognizes the pattern resp. relationship between x and y.

Reinforcement learning on the other side doesn’t need data set. There, you have agents, environments, actions, reward and states. The agent, which is usually a model as well tries to explore his environment by trial and error. An environment is usually a game, but it can be also something else. Then he memorizes his states and the actions, which he chose. For each action he received e certain reward. Depending the environment, the reward could even be negative.

So, the main difference is for sure, that the learning in the supervised learning is limited. The model can only learn as good as his training data are. In reinforcement learning a model can adapt to changes in the environment, if the model is optimized.

# 6. Bibliography

Choudhary, A. (2019) *A Hands-On Introduction to Deep Q-Learning using OpenAI Gym in Python.* Accessed on November 22, 2019, from https://www.analyticsvidhya.com/blog/2019/04/introduction-deep-Q-Learning-python/.

Hu, J. (2016). *Reinforcement learning explained.* Accessed on November 22, 2019,

https://www.oreilly.com/radar/reinforcement-learning-explained/.

Kaundinya, V., Jain, S., Saligram, S., Vanamala, C.K, Avinash, B. (2018) Game Playing Agent for 2048 using Deep Reinforcement Learning*, NCICCNDA*, 363-370. doi: https://doi.org/10.21467/proceedings.1.57

Machine Learning with Phil (2019a) *Deep Q Learning is Simple with Keras | Tutorial*. Accessed on November 22, 2019, from https://www.youtube.com/watch?v=5fHngyN8Qhw.

Machine Learning with Phil (2019b) *Reinforcement Learning in the OpenAI Gym (Tutorial) - SARSA*. Accessed on November 30, 2019, from https://www.youtube.com/watch?v=P9XezMuPfLE&list=PL-9x0\_FO\_lglnlYextpvu39E7vWzHhtNO&index=4.

Sentdex (2019) *Q Learning Intro/Table - Reinforcement Learning p.1.* Accessed on November 22, 2019, https://www.youtube.com/watch?v=yMk\_XtIEzH8&list=PLQVvvaa0QuDezJFIOU5wDdfy4e9vdnx-7.

Szepesvári,C. (2009) *Algorithms for Reinforcement Learning.* Accessed on November 13, 2019,https://sites.ualberta.ca/~szepesva/papers/RLAlgsInMDPs-lecture.pdf.

1. We initially forgot to specify those measures in the code and thus extrapolated the average of the test performance as intercept + 50\*slope of each statistic in the performance test. Rounding errors may have occured [↑](#footnote-ref-1)