Assignment 2

Reinforcement Learning, WS22

Team Members		
Last name	First name	Matriculation Number
Amering	Richard	1331945
Michael	Mitterlindner	11824770

1 Task 1 - RL in a grid world

In this task, a grid world problem from the OpenAI gym-package is solved with different RL algorithms, namely temporal difference learning algorithms deterministic SARSA, expected SARSA and deterministic Q-learning. To investigate the influence of hyperparameters, parameter sweeps for α, ϵ and γ were performed.

1.1 Comparing the influence of hyperparameter ϵ

We evaluated ϵ values in the range 0..1 with a stepsize of 0.1, 11 different values in total. Additionally, α values of the same range and stepsize were investigated, giving a total of 121 combinations. γ was set to 0.9 for these runs. The training and test-performances for the parameter sweeps are shown in figure 1, 2 and 3.

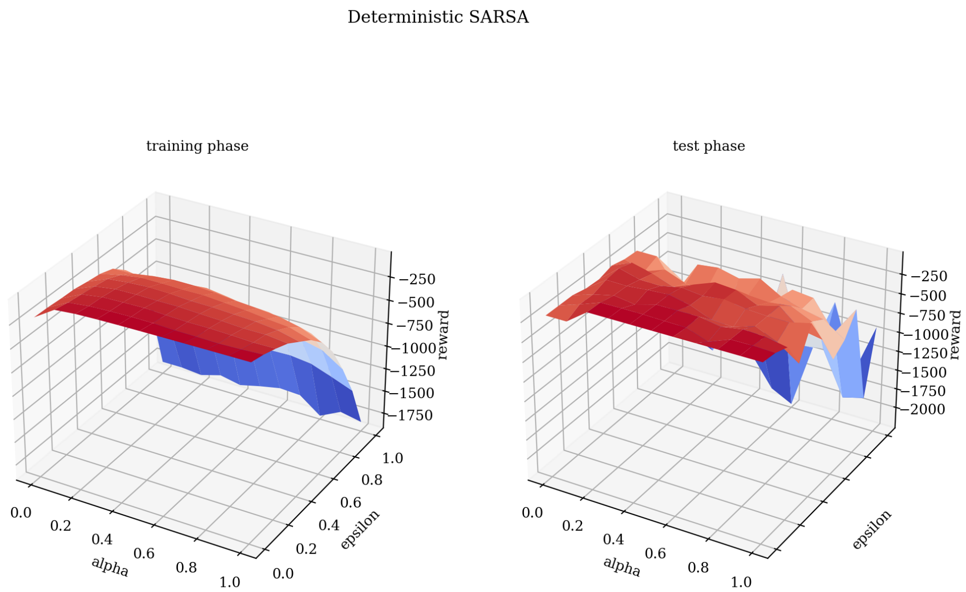

Figure 1: Training and test-reward for the deterministic SARSA model under ϵ and α sweeps

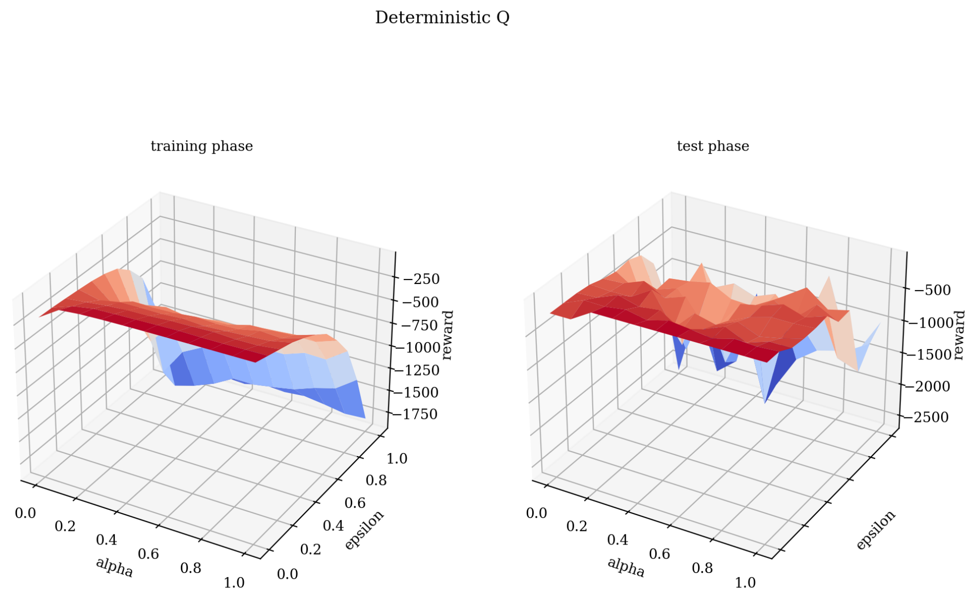

Figure 2: Training and test-reward for the deterministic Q-learning model under ϵ and α sweeps

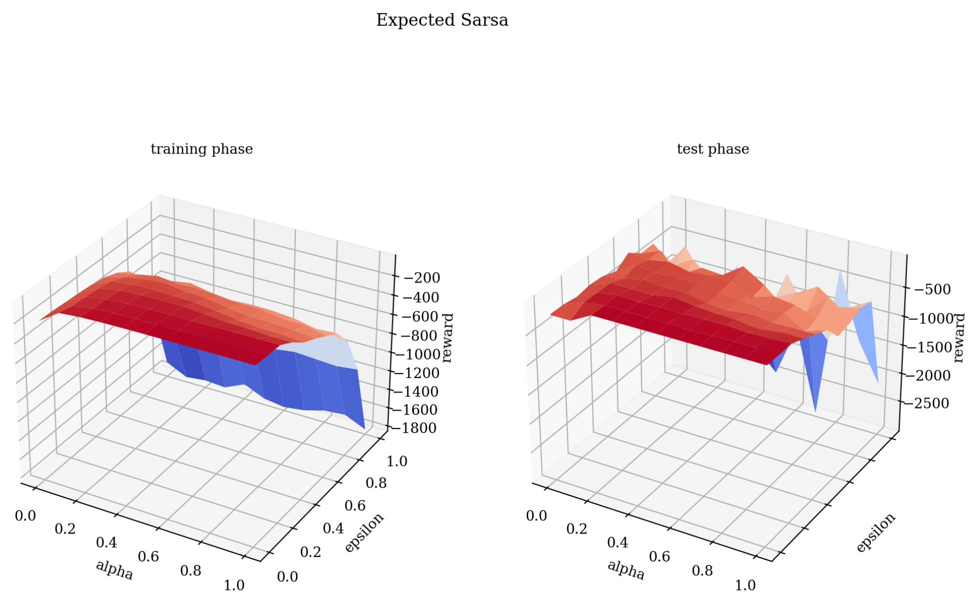

Figure 3: Training and test-reward for the expected SARSA model under ϵ and α sweeps

As we expect, we see the best test-performance for very low epsilon values, since the optimal policy is the greedy one, and not an epsilon-greedy policy that randomly chooses sub-optimal actions. The deterministic SARSA algorithm was able to find the optimal solutions of the grid world problem with a accumulated return of -13, and achieved this with multiple parameter combinations during the test runs, but always with $\epsilon = 0$. The exact combinations and rewards can be found in Table 1.

α	ϵ	γ	reward
0.6	0	0.9	-13
0.2	0	0.9	-13
0.8	0	0.9	-13

Table 1: Overview of hyperparameter combinations with corresponding return for deterministic SARSA

The deterministic Q-learning algorithm showed similar results. Againg, the algorithm found the optimal solution with multiple parameter sets, as shown in Table 2.

α	ϵ	γ	reward
0.2	0	0.9	-13
0.3	0	0.9	-13
1	0	0.9	-13

Table 2: Overview of hyperparameter combinations with corresponding return for deterministic Q-learning

Likewise, the expected SARSA algorithm achieved the same results under multiple hyperparameter combinations, as shown in Table 3.

α	ϵ	γ	reward
0.3	0	0.9	-13
0.6	0	0.9	-13
1	0	0.9	-13

Table 3: Overview of hyperparameter combinations with corresponding return for expected SARSA

1.2 Comparing the influence of hyperparameter γ

Similar to ϵ, we made parameter sweeps with hyperparameter γ in order to show its influence. It was again evaluated in the range 0..1 with a stepsize of 0.1, 11 different values in total, as was α, giving a total of 121 combinations. ϵ was set to 0.1 for these runs. The training and test-performances for the parameter sweeps are shown in Figures 4, 5 and 6. The training and test performances of deterministic Q-learning seem to be generally higher than for the other algorithms, and that over a wider range of parameter settings (Figure 5). The expected SARSA algorithm appears to achieve similar performance, but only on a narrow range of γ values (Figure 6).

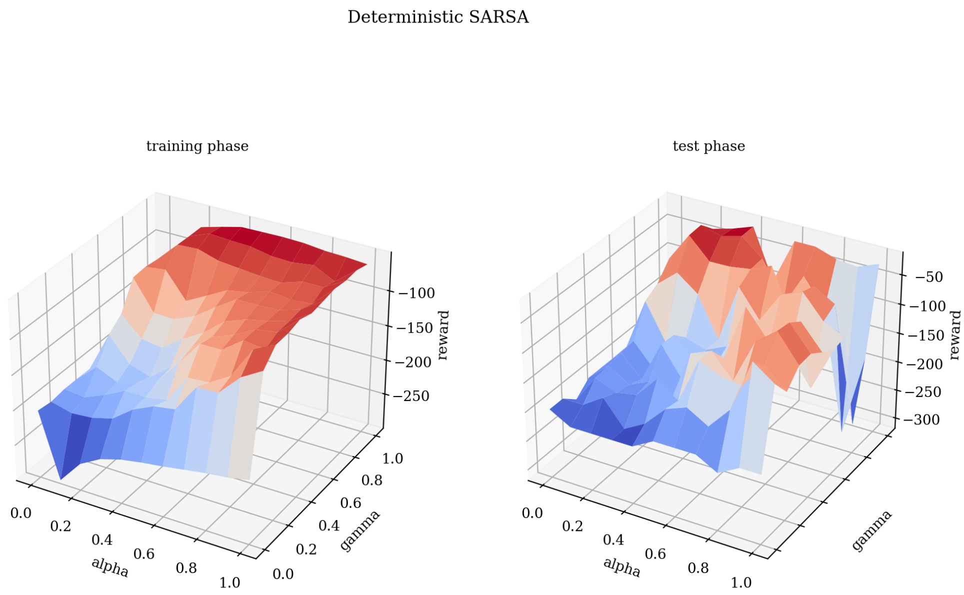

Figure 4: Training and test-reward for the deterministic SARSA model under γ and α sweeps

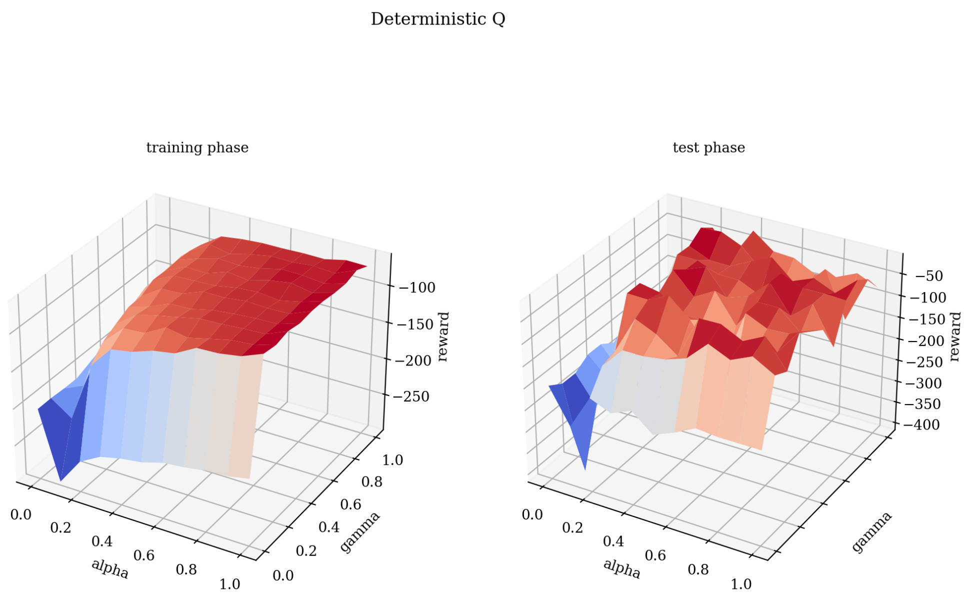

Figure 5: Training and test-reward for the deterministic Q-learning model under γ and α sweeps

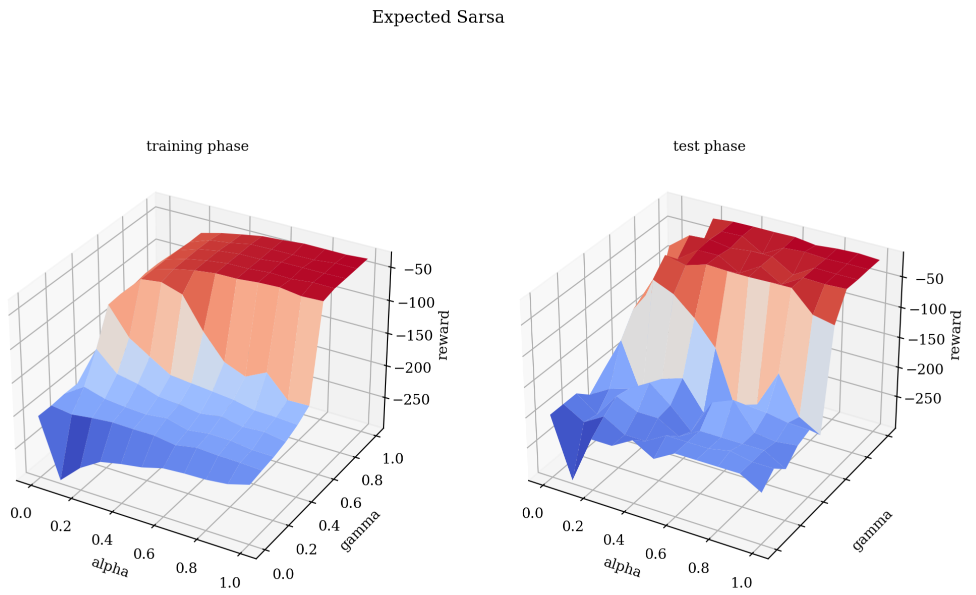

Figure 6: Training and test-reward for the expected SARSA model under γ and α sweeps

The parameter sweeps for α and γ made apparent that higher γ values in the range of 0.8..1 are beneficial to the test performance. While the $\alpha - \epsilon$ sweeps did not exhibit a strong influence of α, here we see a much stronger return for high values of α than for low values during the training runs. Over all algorithms, the best test performances were lower than during the $\alpha - \epsilon$ sweeps. This can be explained by the value of ϵ that was set to a fixed value of 0.1, hindering the algorithm from converging to the optimal policy. The best performing parameter sets for deterministic SARSA can be found in Table 4.

α	ϵ	γ	reward
0.2	0.1	0.9	-18.0
0.4	0.1	1.0	-18.8
0.4	0.1	0.9	-18.2

Table 4: Overview of hyperparameter combinations with corresponding return for deterministic SARSA under γ and α sweeps

The Q-learning algorithm performed better than the determistic SARSA algorithm in that it found optimal or almost optimal strategies with total rewards of -13 or -13.4 in three cases, as documented in Table 5.

α	ϵ	γ	reward
0.7	0.1	0.6	-13.4
0.6	0.1	0.8	-13
0.3	0.1	0.9	-13

Table 5: Overview of hyperparameter combinations with corresponding return for deterministic Q-learning under γ and α sweeps

Expected SARSA appeared to perform slightly better than deterministic SARSA, but not as good as Q-learning, and far from optimal. The best test results are shown in Table 6.

α	ϵ	γ	reward
0.3	0.1	0.8	-16.2
0.6	0.1	1.0	-16.2
0.9	0.1	1.0	-16.2

Table 6: Overview of hyperparameter combinations with corresponding return for expected SARSA under γ and α sweeps

1.3 Why does $\gamma = 1$ work in our MDP problems?

Usually, $0 < \gamma < 1$ is required to ensure convergence of the value function for infinite episodes. Since our MDP problem is capped at 200 steps max, and the fact that a step into the cliff terminates the episode, infinite episode lengths are not possible. Therefore, the value function is bounded and the algorithm also work with $\gamma = 1$.

2 Task 2: Non-tabular RL

2.1 Deriving the parameter update rules for linear models

Linear models for non-tabular reinforcement learning replace the Q-function with a linear model, as in the following:

$$Q(s,a) \rightarrow Q_\theta(s,a) = \theta_a^T \cdot s$$

Where s is not a vector of states, but instead a vector describing one state. The Q-function is now a linear function of the vector describing this state.

The update of the Q-function is then replaced with an update of the parameter matrix, according to gradient decent of a cost function E. α can be interpreted as a stepsize.

$$Q(s,a) \leftarrow Q(s,a) - \alpha(Q(s,a) - y)$$

$$\theta_a \leftarrow \theta_a - \alpha \nabla_{\theta_a} E$$

We therefore need a cost-function and its gradient with respect to the parameter matrix. A common choice is the quadratic error function of the following form:

$$E = \frac{1}{2}(Q(s,a) - y)^2$$

We consider two different cases for parameter updating, one with constant y and one with y being a function of the parameter-matrix θ_a. The gradients are found by deriving the cost function with respect to the parameter matrix.

2.1.1 Constant y:

$$E(\theta_a) = \frac{1}{2}(\theta_a^T s - y)^2$$

Derivation of E with respect to θ_a requires the chain rule and gives the following result:

$$\nabla_{\theta_a} E = (\theta_a^T s - y)s$$

Inserting the gradient of the error-function into above equation gives us the new update rule and results in the following form:

$$\theta_a \leftarrow \theta_a - \alpha(\theta_a^T s - y)s$$

Which is equivalent to

$$\theta_a \leftarrow \theta_a - \alpha(Q_\theta(s,a) - y)s$$

2.1.2 y as a function of θ_a:

Here, we will assume y to be of the form:

$$y = r + \gamma Q_\theta(s', a')$$

The cost function is now dependent on θ_a^T and $\theta_{a'}^T$:

$$E(\theta_a, \theta_{a'}) = \frac{1}{2}(\theta_a^T s - y(\theta_{a'}))^2 = \frac{1}{2}(\theta_a^T s - r - \gamma\theta_{a'}^T s')^2$$

We can now derive two gradients, one with respect to θ_a like before and one with respect to $\theta_{a'}$. This will again require the chain rule and yield the following results:

$$\nabla_{\theta_a} E = (\theta_a^T s - y)s$$

$$\nabla_{\theta_{a'}} E = (\theta_a^T s - y)(-\gamma s')$$

Inserting these gradients of the error-function into the gradient-descent function gives us two update rules of the following form:

$$\theta_a \leftarrow \theta_a - \alpha(Q_\theta(s, a) - y)s$$

$$\theta_{a'} \leftarrow \theta_{a'} + \alpha\gamma(Q_\theta(s, a) - y)s'$$