Course: Reinforcement Learning (SS2021)
Dr. Christoph Metzner & Honghu Xue, M.Sc.
Assignment Report Nr. II
Date: 26. Mai 2021

Team Nr. 007
Robin Luckey
Moritz Gerwin
Frederic Dlugi
Franek Stark
Contact: frederic.dlugi@student.uni-luebeck.de

1 Task: Theoretical understanding on Monte-Carlo Methods and Temporal Difference Learning

1.1 What is the key difference between Dynamic Programming (DP) approaches and Monte-Carlo (MC)/Temporal Difference Learning (TD) approaches, i.e. in which cases can DP approaches not be applied but MC and TD learning can still work?

The main difference is that in Dynamic Programming we assumed a complete knowledge of the environment where as in MC and TD this isn't necessary anymore.
DP can only be applied when the environment is fully known, therefore MC and TD can be applied for a wider range of problems, since no knowledge about the environment is required. As the environment isn't fully know, these methods are usually used to work with state-action pairs (and action-value functions). Otherwise we won't know how to get from one state into the next greedy state.

1.2 Explain what is on-policy learning, off-policy learning in detail.

In general MC and TD update a (state or action-)value function by running episodes with decision made from a policy π. The policy is derived from it's corresponding value function. So one can say decisions are made by that corresponding value function. There exist two ways of doing this:

on-policy learning improves or or rather updates the policy that we use to make decision with. One example for that is the Sarsa-Algorithm where for one step we choose an Action A from the current State S using a policy derived from Q and than observing the next State S' and the Reward R. Moreover we also need the next Action A' from the policy derived from Q. Than we update the value function (or respectively policy) via following update function:

$$Q\left(S,A\right) \leftarrow Q\left(S,A\right) + \alpha\left[R + \gamma Q\left(S',A'\right) - Q\left(S,A\right)\right]$$

As you can see we are constantly updating the action-value function which is used to derive the policy which is then used to choose an action in a certain state. One could say we are working "on" the current policy.

off-policy learning improves a different policy (target policy π) than we are working with (behaviour policy μ) by generating episodes from μ an use the experience to update π.

We take Q-Learning as an example: Given a State S we choose an Action A using a policy derived from Q and than observe the next State S' and the Reward R. Directly from that the value function is updated in the following way:

$$Q(S,A) \leftarrow Q(S,A) + \alpha\left[R + \gamma \max_a Q\left(S',a\right) - Q(S,A)\right]$$

The main difference to Sarsa (and on-policy-learning) is that in the update term we don't use the Action A' chosen by the derived policy but we use the action which would obtain the biggest value. Therefore the updated policy is different from the behaviour policy and it's called an off-policy algorithm.

This is getting interesting when we take a deeper look into what we've called "deriving a policy from a value function". If in our policy we would simply choose actions that result in the highest Return, Sarsa and Q-Learning would be the same. But to maintain exploration one can use the ϵ-greedy strategy during policy derivation.

In those policies also actions that don't maximize the value or return would be taken. During learning this is good to maintain exploration as explained above but for later uses the policy would be imperfect. In the Sarsa algorithm this "imperfect" policy is used to update the policy and therefore will be part of the final policy where as in Q-learning the final policy (π) is updated independently of the derived behaviour policy (μ).

1.3 Explain the stationary policy and non-stationary policy and its relation with the choice of step size α for updating the state value V or Q value.

In general a policy is called stationary if it doesn't change over time where a non-stationary policy changes for different time steps. A stationary policy therefore implies that the decision of the action only depends on the current state of the agent and a non stationary policy depends on the state of the agent and the current time step.
When searching for stationary and non-stationary in the context of reinforcement learning most of the time this distinction is about the environment. A non-stationary environment in this case means that the environment changes over time. A stationary environment therefore does not change over time. FrozenLake-v0 and MountainCar-v0 are stationary.
Generally a stationary policy is easier to train than a non-stationary policy. A stationary policy can be an optimal policy for a stationary environment.
Training in different environments
The MC action-value function update rule is:

$$Q(S_t, A_t) \longleftarrow Q(S_t, A_t) + \alpha[G_t - Q(S_t, A_t)]$$

And a simple TD action-value function update rule looks like that:

$$Q(S_t, A_t) \longleftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)]$$

Both make use of the step size α which encodes how much influence the past experience has on the respective updated function.
When training in a stationary environment a good idea can be to change α over time depending on the reward in the environment.
When training in a non-stationary environment the α value should not depend on the current time since the value function needs to be influenced for each time step equally.
The step size always needs to be $\alpha \in (0, 1]$ to never stop training or over emphasize steps taken.
It can be a good idea to decrease the step size with the number of episodes trained to converge on a optimal policy.

1.4 Apart from a proper choice of the step size α and the discount factor γ, what is another necessary condition to be fulfilled for convergence to an optimal policy in MC and TD approaches in a discrete state-action space?

Apart from a proper choice of α and γ it is also important to maintain exploration. Giving an arbitrary policy MC and TD sample its value function by simply following the policy. For each visited state (or state-action pair) they update the value function with it's reward. From this updated value function a better policy is derived which maximizes the Return. Therefore only for these, already known to be good, states enough samples exist to obtain a good approximation of the corresponding values.
In the worst case, states or state-actions which are assumed to be 'bad' won't be visited at all. But because the initial policy is (often) arbitrary, this assumption doesn't hold. That is why states or state-actions which maybe will lead to a better policy won't be discovered.
So it is important to force the RL-process to keep exploring also 'bad' decision instead of only exploit known good decisions. Summarizing, the probability for every state-action to be selected during training must be greater than zero.

1.5 Besides the common exploration strategy of (decaying) ϵ-greedy exploration, please explain another exploration strategy, you can choose either one of the following

1.5.1 Upper-Confidence-Bound Action Selection

In contrast to ϵ-greedy exploration which adds a probability of choosing actions A which aren't maximising $Q(A)$ (aren't the greedy ones), upper-confidence-bound action selection takes the uncertainty about individual actions into account. At a given timestep t it selects an action A_t as follows:

$$A_t = \arg\max_a \left[Q_t(a) + c\sqrt{\frac{\ln(t)}{N_t(a)}} \right]$$

Where $Q_t(a)$ denotes the action-value (giving the current policy) and $N_t(a)$ is representing the count of how often action a was already chosen. Each time an action a is selected, the value of $N_t(a)$ is incremented (starting from zero). For $N_t(a) = 0$ the action A_t is simply the one maximizing $Q(a)$. The term under the root becomes large for actions which were relatively few selected compared to all actions and becomes small for actions often selected actions. The confidence value $c > 0$ controls the degree of exploration as it weights the influence of the exploitative term into the actual value for this specific action from which the best action is than later selected. As time goes by, the logarithm of t in the right hand term increases more slowly, is however unbounded. This leads to the effect, that all actions will be chosen eventually, and the action-value function $Q(a)$ stays relevant to the selection.

1.5.2 Boltzmann exploration (also called softmax exploration). Please provide math expressions and explain the term.(please refer to the book Chapter 2)

2 Task: Programming part on MC and on-policy and off-policy one-step TD methods

2.1 Implement the algorithm On-policy first-visit MC control with ϵ-greedy exploration strategy on the OpenAI Gym environment "frozenlake-v0", where $\epsilon = 0.4$. Show the final learned policy in the same format as in Assignment I.

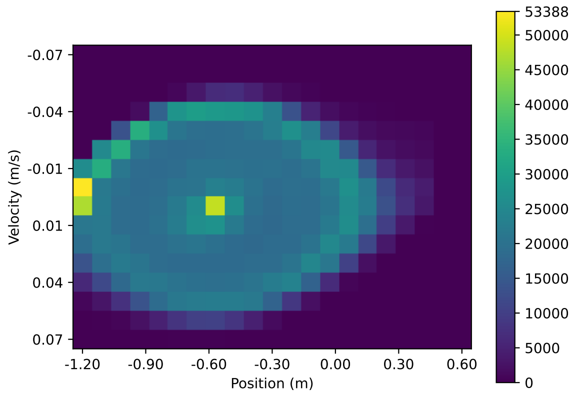
See Figure 4 to see our results from the implementation.

2.2 Implement the SARSA algorithm using linearly-decaying ϵ-greedy exploration strategy on the adapted "MountainCar-v0" environment with fixed step size ($\alpha = 0.05$). An example of linearly-decaying ϵ-greedy exploration strategy is $\epsilon = 1 - (E_c/E_t)$, where E_c and E_t refers to the index of the current episodes and the number of the total training episodes respectively. This means the agent starts exploring the environment with $\epsilon = 100\%$ (full exploration) and exploit more with the training episodes. Show the plot (episodic reward w.r.t number of training episodes) with multiple runs, one run per team member. Merge all the plots of your team in one plot using the template. Wisely choose your axis range for a clear presentation. Hint: you need save the statistics as .npy file and plot again. The episodic reward of a the finally learned policy is around -150. In the testing phase, please constrain the maximal episode length to 1000 to avoid infinite loop of interactions.

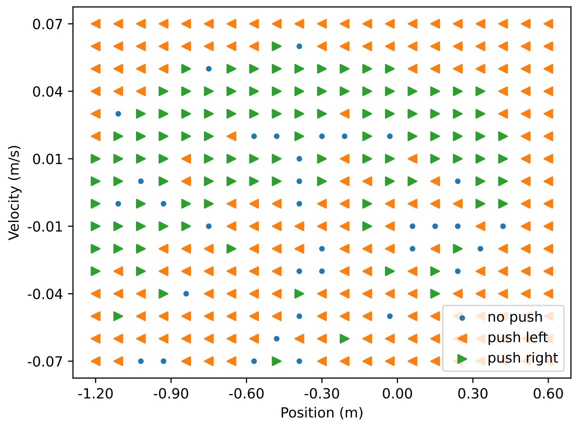
See Figure 1 to see our results from the implementation.

2.3 Implement the one step Q-learning algorithm with decaying ϵ-greedy exploration strategy on the adapted "MountainCar-v0" environment with fixed step size ($\alpha = 0.05$). Show the plot (episodic reward w.r.t number of training episodes) in the same format as the task of SARSA. The code of this task is very similar to that in SARSA. The episodic reward of the learned policy is around -150. In the testing phase, also constrain the maximal episode length to 1000 to avoid infinite loop of interactions.

See Figure 2 to see our results from the implementation.

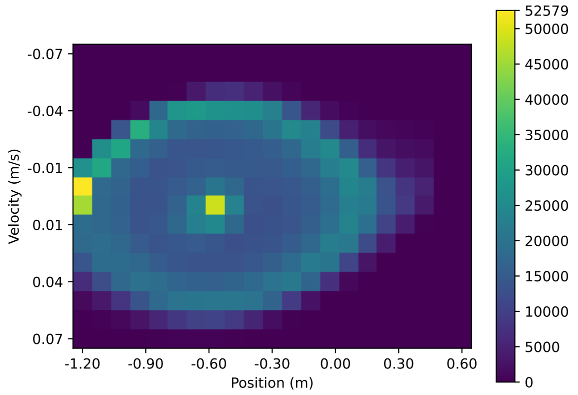


a) Number of visits per state while training

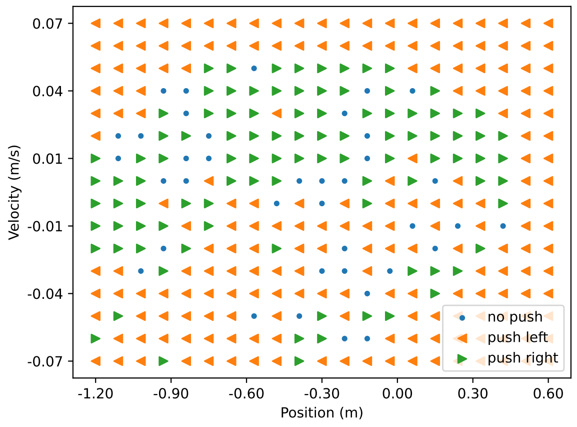


b) Learned Policy

Figure 1 Results for SARSA



a) Number of visits per state while training



b) Learned Policy

Figure 2 Results for Q-learning

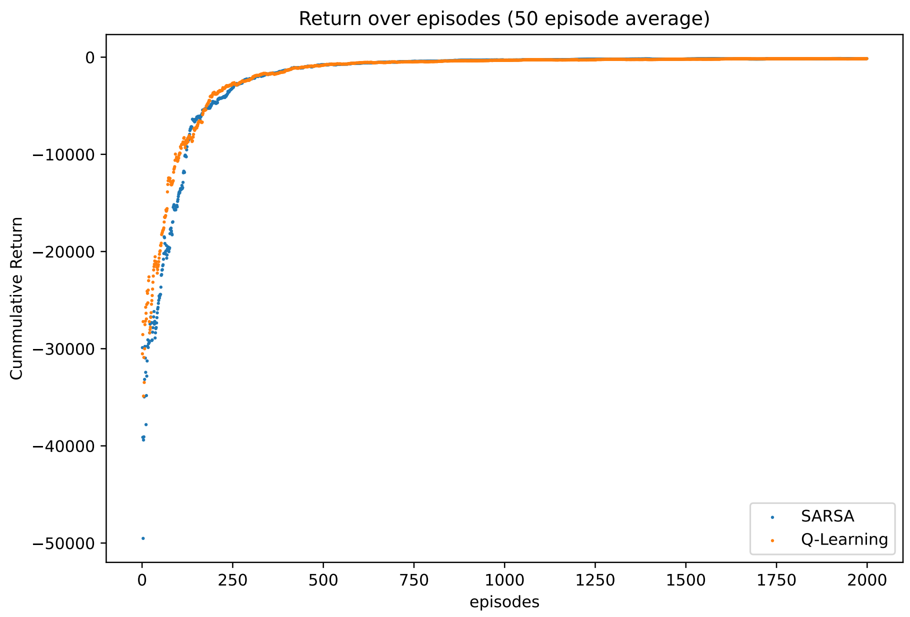


Figure 3 Return over Generations of SARSA and Q-Learning

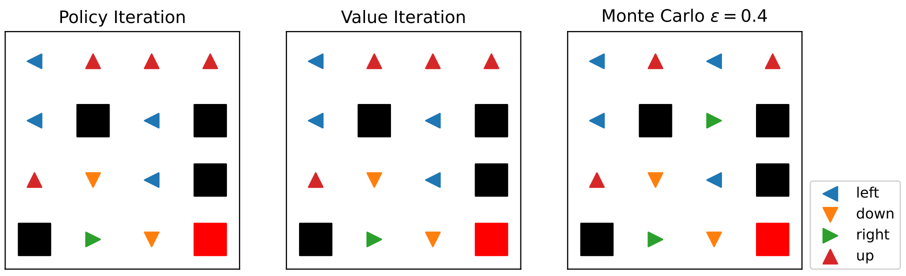


Figure 4 Policies for Frozenlake-v0 (red=Goal, black=Hole)

3 Task: Going Deeper

3.1 Compare the results of the learned policy of the MC approaches with the learned policy of PI/VI in Assignment I. Why can it happen that some states (e.g. state 2) show a different learned policy in MC compared to the one learned with PI/VI approaches (excluding the terminal state of holes)? Analyze the reason for that.

In figure 4 the results are shown. While The Policy and Value Iteration resulted in the same optimal policy, the Monte Carlo Method resulted in a different policy. There can be different causes involved: There might indeed be several optimal policies and since Monte Carlo is partly based on random events, the Monte Carlo approach does not necessarily always results in the same policy. Further more the Monte Carlo approach estimates $q(s,a)$ as the number of (first) visits approach infinity. However since we obviously only simulated a finite number of visits, the q-values we base our policy on are not perfect - at least in theory - but we do simulated 150.000 episodes and have only 64 different state/action combinations so we visit every state/action par a sufficient number of times. This can be seen in the left part of figure 6. The third reason is that the result depends on the exploration/exploitation trade off of namely how the ϵ-greedy function is implemented. As one can see in figure 5 the final policy is different for a different choice of ϵ.

3.2 For task II, question 1, if fixing $\epsilon = 5\%$ for all training episodes and keeping the total number of training episodes unchanged, how is the result of the learned policy compared to the ϵ-greedy exploration strategy given the same number of training episodes? Why is that?

The result (final policy) is shown in figure 5. The resulting policy differs from the policy obtained in task II, question 1. There could different explanations for this: A low ϵ can potentially lead to insufficient exploring. If the algorithm is too greedy it might not become aware of other suitable policies. However as shown in figure 6 the visits more evenly distributed with the smaller epsilon value. This is probably a result of the hole right next to the start. If the ϵ is very high, there is a high probability for the agent to end up in the hole. This can be seen in the left plot of the figure where the relative frequency of visits in the initial state is much higher then in the right plot. The lower exploration rate is not necessarily bad if the state/action set is small compared to the number of trials.

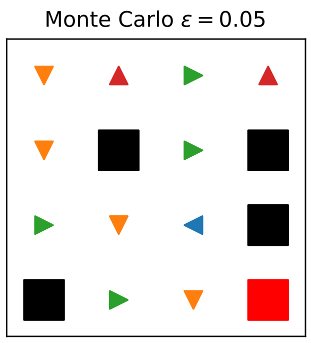


Figure 5 Policy MC with small ϵ for Frozenlake-v0

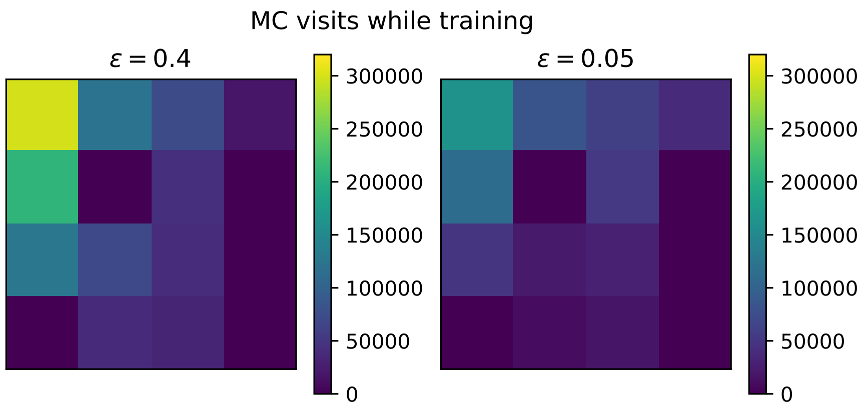


Figure 6 MC visits while training for Frozenlake-v0

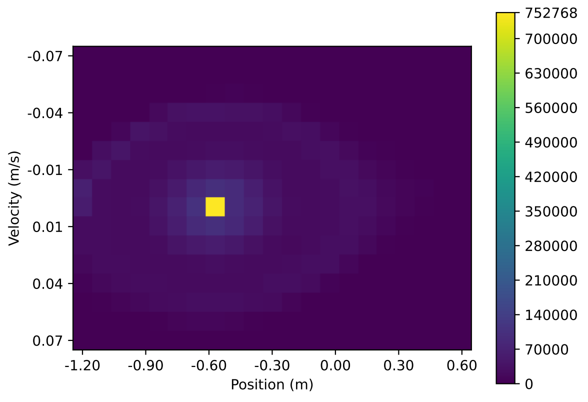
3.3 Repeat the task II, subtask 3 only with a different step size setting 'average sample' ($\alpha = 1/N_{visit}$), where the other settings remain the same. For this task, you don't need to show the plot. Nor is it necessary to wait until the program finishes. Explain what you observe for the learned policy with the same number of training episodes of 2000? Theoretically, can the setting of $\alpha = 1/N_{visit}$ reaches convergence to a near optimal policy and why? Does the time to converge to near-optimal action value $Q_\pi * (s, a)$ depends on the initial Q-value?

The learning rate defines the influence of the newly observed value on the previous q-value. If $\alpha = 1/N_{visit}$ the leaning rate is very high and the q-values change quickly.
In our experiment setting $\alpha = 1/N_{\text{visit}}$ does not converge to an optimal policy. See figure 9. This is largely because setting $\alpha = 1/N_{\text{visit}}$ leads to slow adjustments when a state gets visited often. It can be helpful to decrease α over the training process since large adjustments are often only necessary in the beginning of the training process and so they can be smaller when the agents policy needs to be fine tuned.

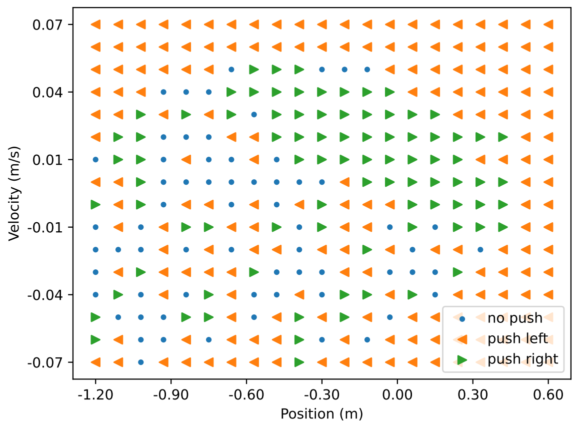
The initial q-values work like a bias and can help to converge to the optimal result more quickly if set correctly. However setting the initial q-values to values that are much lower then the expected real values e.g. to $-1E7$ leads to very slow convergence. The reason is that once one value has been corrected the greedy part of the algorithm will always choose this action since the other actions still have a much lower initial value. The agent only chooses new actions when the next action is chosen randomly by our epsilon greedy policy. In other words exploration is discouraged and the low values work exactly the opposite way of optimistic initial values.

3.4 What is the purpose of optimistic initial values? If one wants to perform optimistic initial values on the "MountainCar-v0" environment, which of the following initial values of each state-action pair are considered as valid? And also reason why. (Multiple answers could be correct) A. 0, B. 50 , C. -50 , D. -5, E. 5 ?

Optimistic values are a common strategy for balancing exploration and exploitation. They encourage exploitation early on but can have a disadvantage in non stationary problems, because when used without other means of exploration, changes in the environment might not be noticed. Using optimistic initial values causes the agent to behave as in a breadth first search: The initial values chosen for the q tables a chosen to be higher then they will turn out later (see 8). This way the initial value will be decreased after the first visit, because the real reward is lower then the chosen initial value. Since the agent is greedy, this will cause the agent to not choose the same action again but another action that is still set to the high initial value.
Since the reward for each state - except for the final state - is -1 each initial value greater then -1 can be considered to be a optimistic initial value. Thus A., B. and D. are correct.



a) Number of visits per state while training



b) Learned Policy

Figure 7 Results for Q-learning with $\alpha = 1/N_{\text{visit}}$

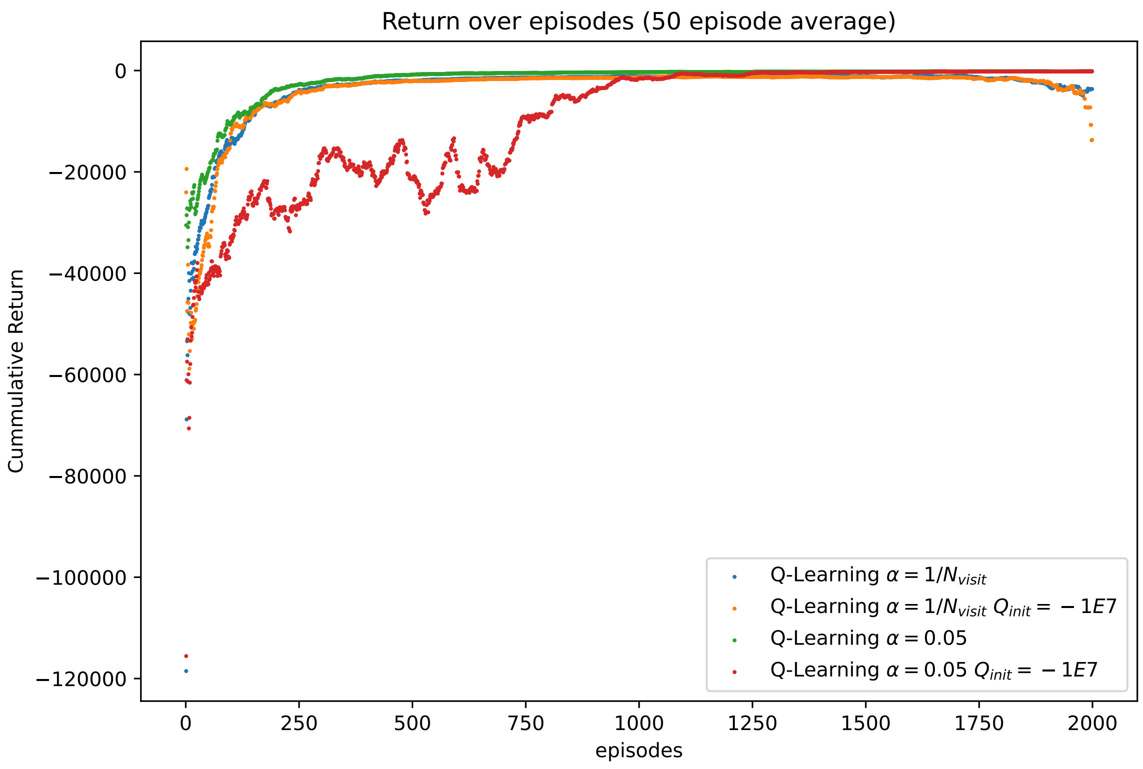


Figure 8 Return over Q-Learning Generations

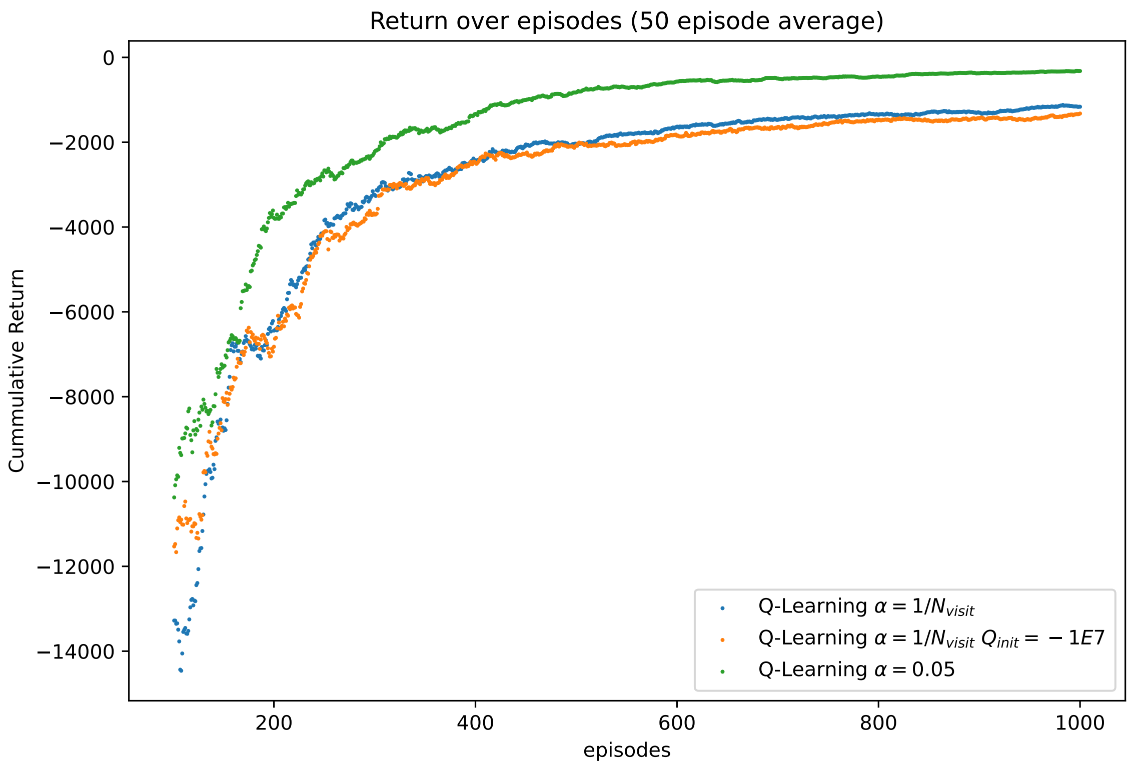


Figure 9 Return over Q-Learning Generations 100-1000