Report 2 - SF2957 Statistical Machine Learning

Simon Carlsson 970404-8033 August Regnell 970712-9491

December 16, 2021

Contents

1 Objectives

The aim of this project is to train an agent to play Blackjack using reinforcement learning. The state space of the game will be presented in two different ways, which will have an effect on the total number of states and thus the training of the agent. The agent will be trained using both *On-policy Monte-Carlo learning* and *Q-learning*. These training methods as well as the learning and exploration rates will be explored in order to find the (close to) optimal strategy, which will be presented at the end of the report. Finally we will also explore how the number of decks used affects the learning and the ordering of the best performing training algorithms.

2 Mathematical background

2.1 On-policy Monte-Carlo learning

Monte Carlo methods in reinforcement learning are on-policy methods where episodic tasks are considered. That is, the game is started and played until it terminates and

all states, actions and rewards are recorded. The key characteristic of the Monte Carlo method is that the transition probabilities $p(s', r \mid s, a)$ are unknown but that they can be sampled.

In order to calculate v_π and q_π one uses the generated episodes. After every episode the estimate of $q_\pi(\cdot, \cdot)$ with the update equations as defined in Algorithm 1. This is then repeated N times. Note here that an incremental update equation is used for the update of $\hat{q}_\pi(\cdot, \cdot)$, which is a form of a Robbins-Monro algorithm.

Since the number of state-action pairs $(s, a) \in (\mathcal{S}, \mathcal{A}(s))$ may be large, starting in the same state $S_0 = s_0$ every episode may lead to several un-visited state-action pairs. To combat this one uses *exploring starts*, that is one gives a positive probability for any combination (s, a) to be used as the initial value. Assuming exploring starts is used, the policy π_i is updated like follows after every episode

$$\pi_i(s) = \arg\max_{a \in \mathcal{A}(s)} \hat{q}_\pi^{(i-1)}(s, a) \tag{1}$$

However, we would like to remove the assumption of exploring starts. To make sure that every state-action pair still has a positive probability we can use an ϵ-soft policy, which is defined in the next section. Note that this policy cannot be optimal globally.

Algorithm 1: Monte Carlo

initialize Q and π arbitrarily

for *i=1* **to** N **do**

generate an independent episode $S_1^{(i)}, R_1^{(i)}, \dots S_{\tau^{(i)}}^{(i)}, R_{\tau^{(i)}}^{(i)}$ from π_i

for $(s,a) = (S_t^{(i)}, A_t^{(i)})$, $1 \le t \le \tau^{(i)}$ **do**

find the first visit to the state action pair (s,a),

$$\tau_{s,a}^{(i)} = \inf\{t \ge 0, S_t^{(i)} = s, A_t^{(i)} = a\}$$

calculate the discounted reward after state action pair (s,a),

$$G_{s,a}^{(i)} = \sum_{k=1}^{\tau^{(i)} - \tau_{s,a}^{(i)}} \gamma^k R_{\tau_{s,a}^{(i)}+k}^{(i)}$$

incrementally update Q,

$$\hat{q}_\pi^{(i)}(s,a) = \hat{q}_\pi^{(i-1)}(s,a) + \frac{1}{N_i(s,a)}(G_{s,a}^{(i)} - \hat{q}_\pi^{(i)}(s,a))$$

where $N_i(s,a)$ is the number of the first i episodes that visits (s,a)

end

update π_i according to ϵ-soft policy update equation

$$\pi_i(a|s) = \begin{cases} 1 - \epsilon + \epsilon/|A(s)|, & \text{if } a = a_*(s) \\ \epsilon/|A(s)|, & \text{if } a \ne a_*(s) \end{cases}$$

where

$$a_*(s) = \arg\sup_{a \in \mathcal{A}(s)} \hat{q}_\pi^{(i)}(s,a)$$

end

2.2 Q-learning

Q-learning is an off-policy temporal difference learning algorithm which is based on the Bellman optimality equation for the action value. Below, we give a brief summary of the algorithm as implemented in our code.

Algorithm 2: Q-learning

initialize $q(s,a)$ arbitrarily $\forall s \in \mathcal{S}, a \in \mathcal{A}(s)$
for *episode=1* **to** *n_sims* **do**
  initialize S
  while *state S not terminal* **do**
    choose A_{t-1} using ϵ-greedy
    take action A_{t-1}, observe R_t and S_t
    set $q_t(S_{t-1}, A_{t-1}) \leftarrow q_{t-1}(S_{t-1}, A_{t-1})$
    $$+\alpha_t \Big[R_t + \gamma \sup_{a \in \mathcal{A}(S_t)} q_{t-1}(S_t, a) - q_{t-1}(S_{t-1}, A_{t-1}) \Big]$$
  end
end

Let's quickly describe the ϵ-greedy policy used above. Denote

$$a_*(s) = \arg\sup_{a \in \mathcal{A}(s)} q_{t-1}(s,a). \tag{2}$$

Then as we sample A_{t-1} given S_{t-1} using ϵ-greedy, we sample from

$$\pi_{t-1}(a|s) = \begin{cases} 1 - \epsilon + \epsilon/|A(s)|, & \text{if } a = a_*(s) \\ \epsilon/|A(s)|, & \text{if } a \neq a_*(s) \end{cases} \tag{3}$$

2.3 Blackjack in Reinforcement Learning

For this problem a simplified version of blackjack is considered; there is only one player and the house and several options such as *splitting pairs*, *doubling down* and *insurance* are not considered.

The rewards are simple and are only paid out at the end of the episode:

- Obtain 21 points on the first two cards, knows as a blackjack, without a dealer blackjack.
  Reward 1.5
- Reach a final score higher than the dealer without exceeding 21.
  Reward 1
- Dealer gets points exceeding 21 and player does not.
  Reward 1
- The player's point equals the dealer's points.
  Reward 0
- All other scenarios.
  Reward -1

Two versions of the game's state space is considered, *the expanded state space* and *the sum state space*.

The Expanded State Space

In this representation of the state space the agent's hand is represented by a vector $(s_1^a, \ldots, s_{10}^a)$ where s_i^a is the number of cards of value i and the dealer's hand is represented by its card sum S_d. The state space can thus be reduced to

$$\mathcal{S}_1 = \{(s_1^a, \ldots, s_{10}^a, S_d) : \sum_{i=1}^{10} i s_i \leq 31, s_i \geq 0 \text{ for } i = 1, \ldots, 10, \text{ and } S_d \leq 26\}$$

The Sum State Space

One can further reduce the state space by only considering the agent's and the dealer's card sum (S_a and S_d) as well as keeping track if the player is holding a usable ace or not ($u \in \{0, 1\}$). The state space can thus be represented in the following way

$$\mathcal{S}_2 = \{(S_a, u, S_d) : 1 \leq S_a \leq 31, u \in \{0, 1\}, 1 \leq S_d \leq 26\}$$

This state space has some flaws, sometimes one may not know if the agent has Blackjack or not and it won't keep track of the remaining cards in the deck.

3 Assignments

3.1 (c) Comparing the algorithms

In figure 1, we display the expected reward plotted for the MC algorithm and the two Q-learning algorithms against the number of episodes with one, two, six, eight and infinitely many decks. They were all trained for 10^7 episodes.

We see that for one deck, the Q-learning on the $\mathcal{S}_1$ space performed the best. Second best was the on-policy Monte Carlo algorithm on $\mathcal{S}_1$. Worst performing was the Q-learning on the $\mathcal{S}_2$ space, however MC overtook the second place from Q-learning on $\mathcal{S}_2$ only after about $5 \cdot 10^6$ episodes. An additional remark is that the Q-learning on the $\mathcal{S}_2$ space performed the best up until about $1.5 \cdot 10^6$ episodes. Worth noting is that using the $\mathcal{S}_2$ space is especially bad for only one deck since keeping track of the remaining cards is far more valuable than for using more or even infinitely many decks.

For two, six, eight, and infinitely many decks, the Q-learning algorithm on $\mathcal{S}_2$ performed the best, followed by Q-learning on $\mathcal{S}_1$ and lastly the on-policy MC. While it was expected the advantage the $\mathcal{S}_1$ space has over the $\mathcal{S}_2$ space should diminish, we did not expect $\mathcal{S}_2$ to surpass it after only using two decks. The graphs in figure 1 gives a hint of why this may happen: the Q-learning algorithm on the sum state space seems to have converged while the average reward for the other two algorithms seem to still be increasing. This is because the $\mathcal{S}_1$ space is much larger and needs much more training before the best action for every state is found or maybe even every state is visited. E.g. around 42,000 states are visited during training of the Q-learning algorithm on the expanded state space while only 700 were visited during the training of the Q-learning

algorithm of the sum state space. We thus expect the $\mathcal{S}_1$ space to surpass the $\mathcal{S}_2$ if the model where trained on more episodes.

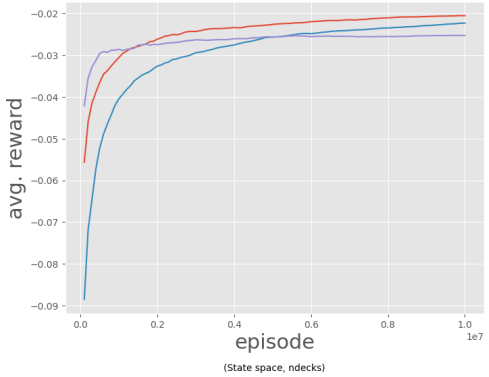
Next, we proceed to finding a suitable decay rate $\alpha_n = cn^{-\omega}$, $\omega \in [0, 1)$. We still use 10^7 episodes but for computational reasons we limit our analysis to one and infinitely many decks. We examine $\omega \in \{0.01, 0.33, 0.77, 0.99\}$. The results are presented in figure 2 for one deck. We see that a very small ω leads to significantly worse results. For $\omega = 0.33$, the performance is also somewhat lower than for 0.77. The performance for $\omega = 0.33$ is comparable to that of $\omega = 0.99$.

Additionally, we investigate the effect of a decaying exploration rate ϵ. Here, we use 10^7 episodes, $\omega = 0.77$ and let $\epsilon_n = \frac{\epsilon}{1+N(s)}$, where $\epsilon \in \{0.2, 0.1, 0.05\}$ and $N(s)$ the number of visits to a state s. Note that we also explore $\epsilon > 0.05$ since the exploration rate is decaying and we still want to have a reasonably high *average* exploration rate so it's comparable to when not using a decaying rate. We find consistent results that a decaying exploration rate, despite the initial exploration rate, is performing subpar to a constant exploration rate for all number of decks. The results can be seen in figures 3-7. It is possible that the chosen decay rate was not optimal, indeed, *towards data science* writes[1]

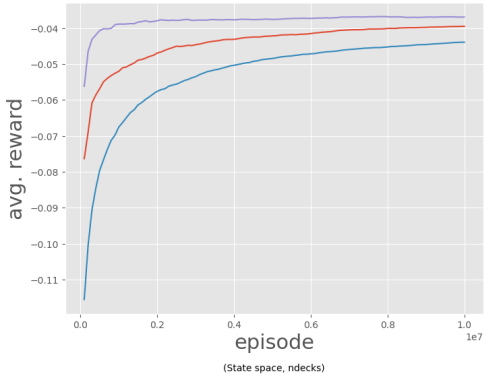
> One way to satisfy these conditions is to modify the value of ϵ, making it gradually decay, when specifying an ϵ-greedy policy. However, one has to be very careful with setting the decay rate for ϵ. Determining the best decay is not trivial, requiring a bit of alchemy, that is, experience...

Thus it is possible that we could have obtained better results with a decaying exploration rate defined somewhat differently, perhaps $\epsilon_n = \frac{\epsilon}{1+\lambda N(s)}$ or $\epsilon_n = \frac{\epsilon}{(1+N(s))^\lambda}$, where $\lambda \in (0, 1)$.

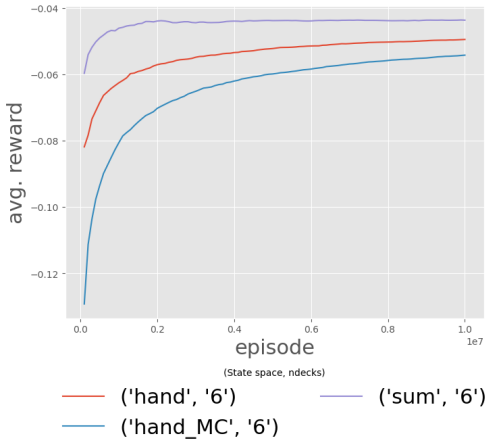
[1] `https://towardsdatascience.com/monte-carlo-methods-9b289f030c2e`



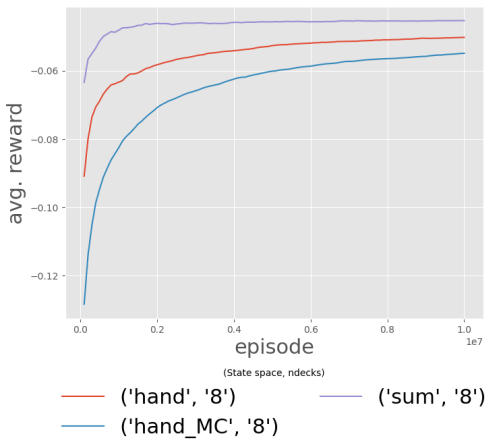
(a) One deck



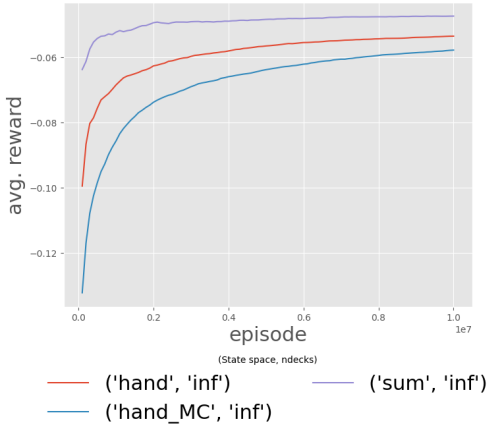
(b) Two decks



(c) Six decks

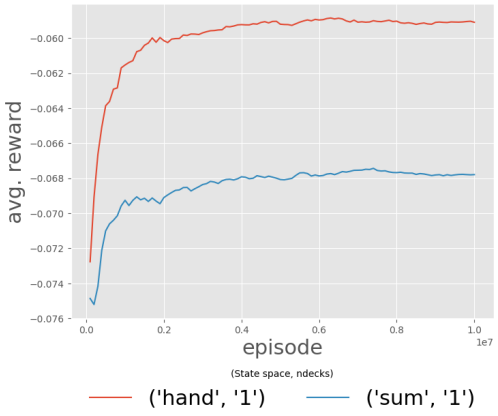


(d) Eight decks

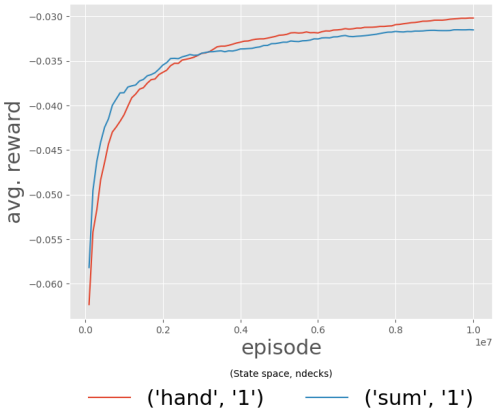


(e) Infinite decks

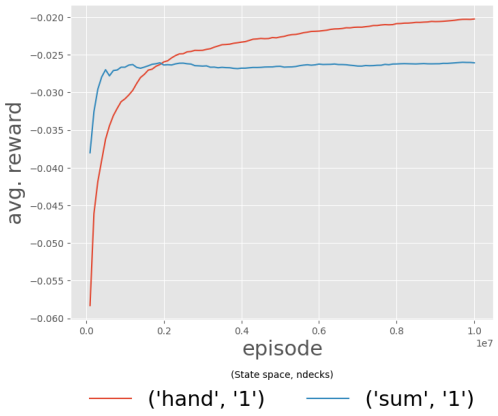
Figure 1: Expected return



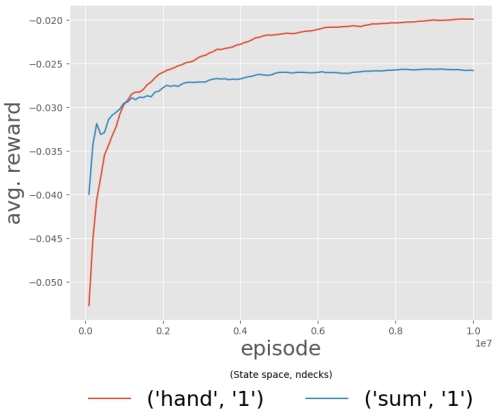
(a) $\omega = 0.01$



(b) $\omega = 0.33$

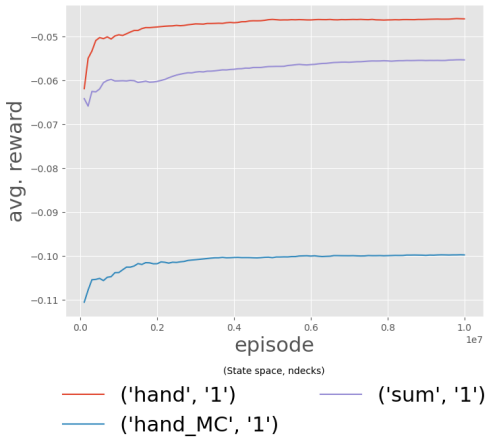


(c) $\omega = 0.77$

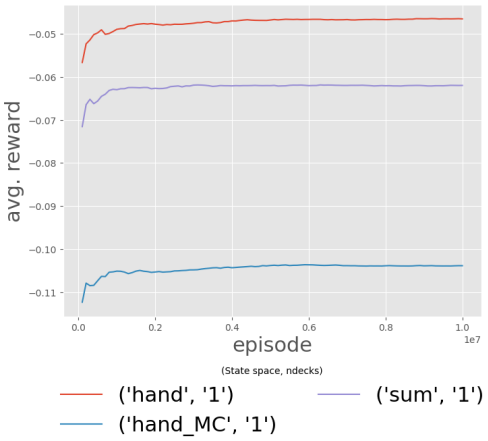


(d) $\omega = 0.99$

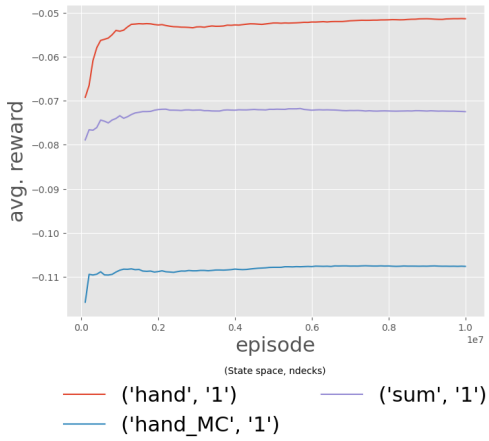
Figure 2: Expected return for different values on ω and for one deck



(a) $\epsilon = 0.2$

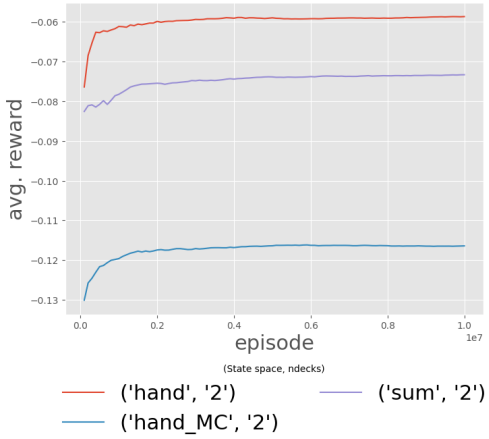


(b) $\epsilon = 0.1$

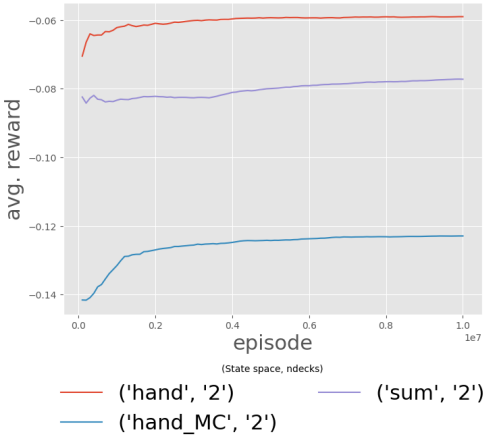


(c) $\epsilon = 0.05$

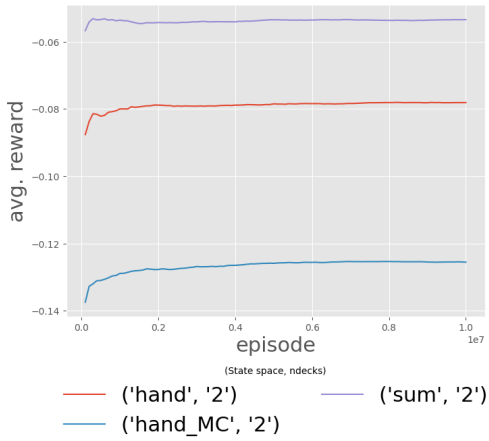
Figure 3: Expected return for different values of ϵ with decaying learning rate, for one deck.



(a) $\epsilon = 0.2$

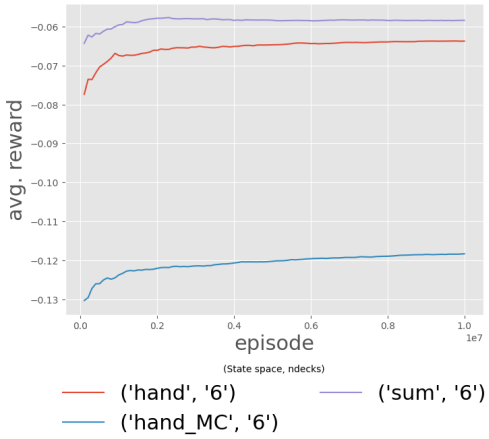


(b) $\epsilon = 0.1$

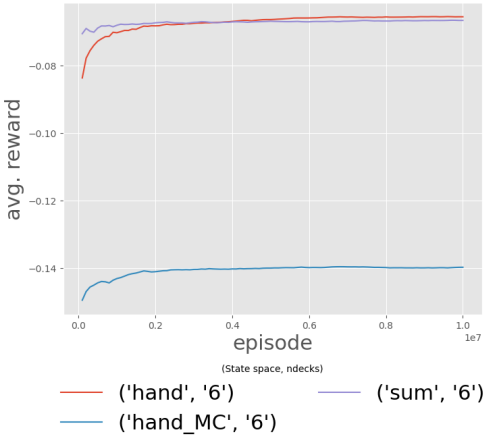


(c) $\epsilon = 0.05$

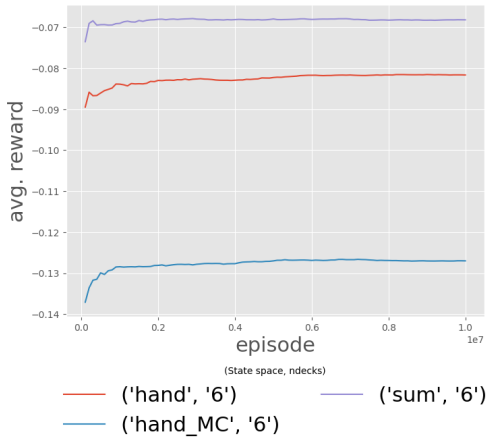
Figure 4: Expected return for different values of ϵ with decaying learning rate, for two decks.



(a) $\epsilon = 0.2$

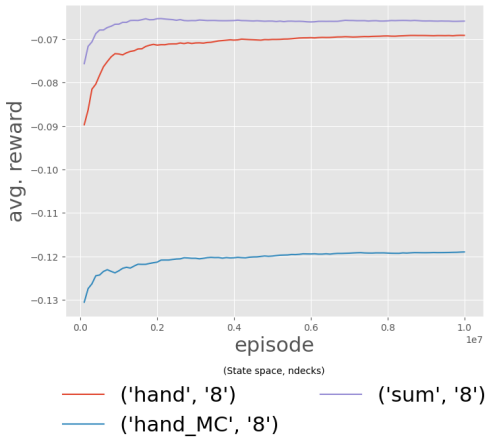


(b) $\epsilon = 0.1$

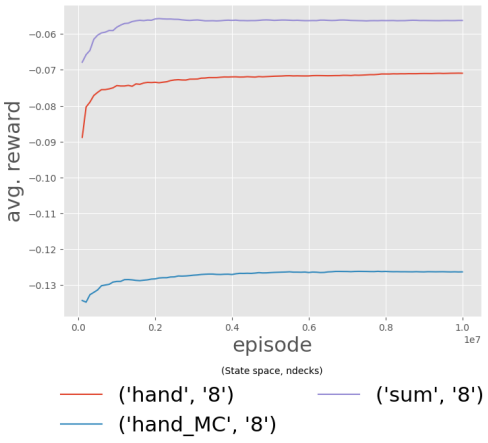


(c) $\epsilon = 0.05$

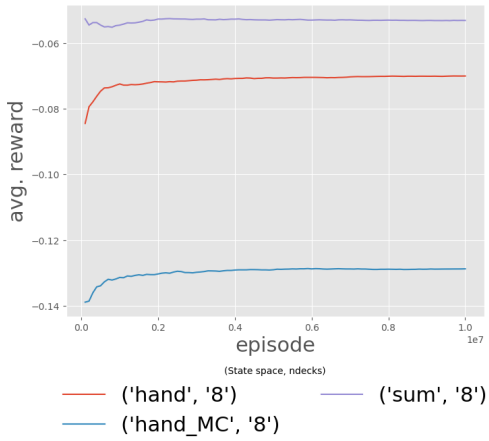
Figure 5: Expected return for different values of ϵ with decaying learning rate, for six decks.



(a) $\epsilon = 0.2$

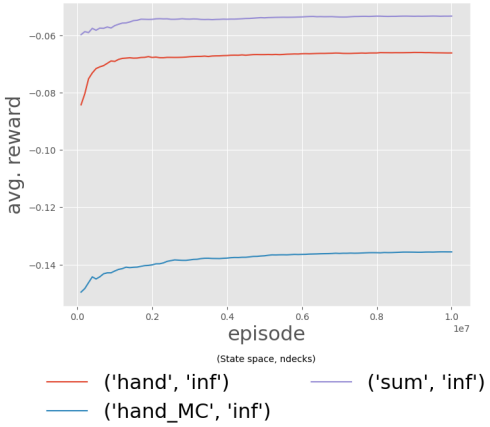


(b) $\epsilon = 0.1$

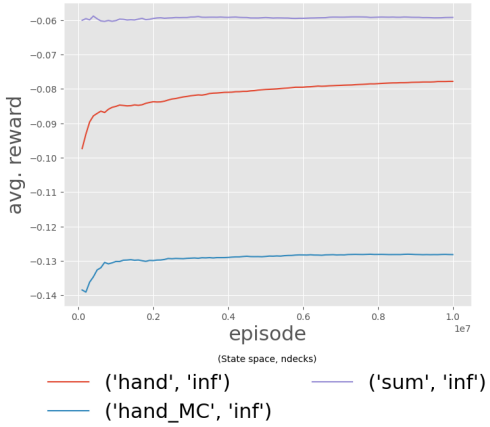


(c) $\epsilon = 0.05$

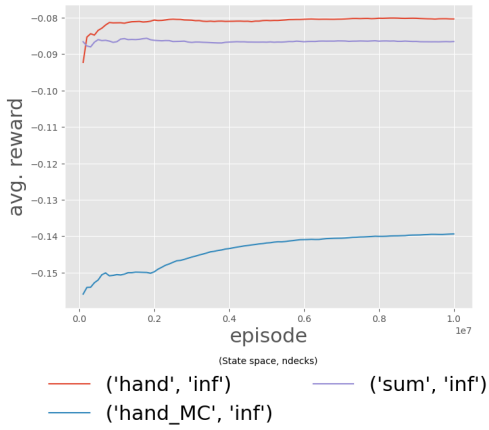
Figure 6: Expected return for different values of ϵ with decaying learning rate, for eight decks.



(a) $\epsilon = 0.2$



(b) $\epsilon = 0.1$

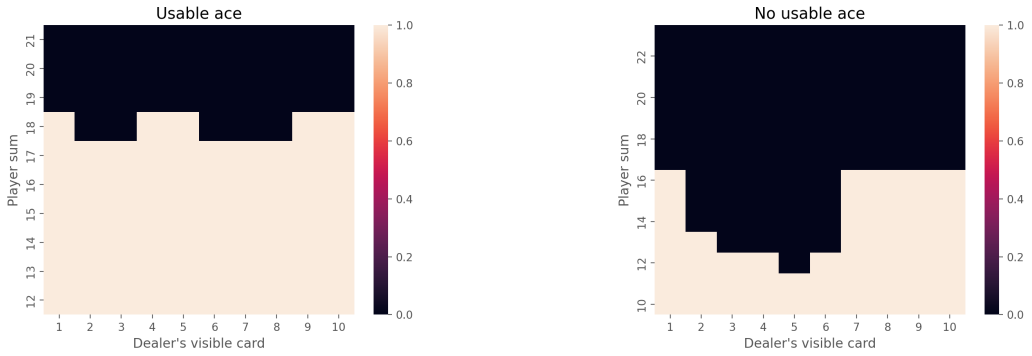


(c) $\epsilon = 0.05$

Figure 7: Expected return for different values of ϵ with decaying learning rate, for infinite decks.

3.2 (d) An optimal strategy

We have chosen to limit our analysis for an optimal strategy to the case where we have eight decks. The motivation for this is that eight decks corresponds to two full decks of cards, a reasonable setting for a casual round of Blackjack in real life. Our optimal strategy for the Q-learning with state space model $\mathcal{S}_2$ can be found in figure 8. This strategy is very similar to that presented in Sutton and Barton (2017), see figure 9. There are some discrepancies however, but not particularly severe. We expect that these would correct themselves if we increased the number of episodes even more and allowed the agent to use the options we removed (such as *splitting pairs*). As previously, 10^7 episodes were used here.



(a) Optimal strategy for eight decks, with usable ace

(b) Optimal strategy for eight decks, with no usable ace

Figure 8: Our optimal strategy for eight decks.

4 Conclusion

The on-policy Monte Carlo learning generally performed worse than the two Q-learning algorithms, the exception being for one deck and with a constant exploration rate. For a decaying exploration rate, the Monte-Carlo algorithm consistently performed the worst. Among the two Q-learning algorithms, the sum-based model performed the best when there were more than one deck and the exploration rate was constant and for different learning rates.

We found the optimal strategy when playing with eight decks for the sum-based model. The results were similar to the theoretically optimal strategy.

5 Appendix

Our code is included separately in the canvas upload.

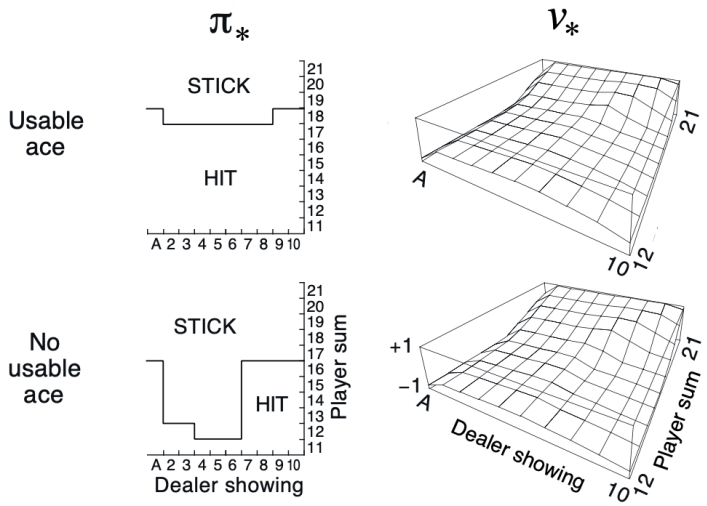


Figure 9: Theoretically optimal Blackjack strategy.