Implementation of a Blackjack Playing Agent Using Reinforcement Learning

Janosch Jungo
jjungo@student.ethz.ch

Vasily Kopylov
vkopylov@student.ethz.ch

Abstract

Blackjack is a casino game with well-defined rules, reward structure, stochastic nature and existing closed-form solution for the optimal player's strategy, making it an attractive test setting for reinforcement learning (RL) algorithms. In this paper, we apply value iteration, Monte-Carlo ES, SARSA, Q-learning, Double Q-learning and DQN with various exploration strategies (random, greedy, ϵ-greedy, UCB and Boltzmann) to solve the game for a limited and full action space. In the former, double Q-learning with UCB converges to the near-optimal policy and beats state-of-the-art performance, whereas in the full action space, we find that our methods are unable to converge. We then present a novel dynamic betting algorithm, that enables RL methods to adapt their betting amount dynamically in each round. We show that our algorithm applied on trained RL methods improves the performance considerably in both action spaces.

1 Introduction

The game of Blackjack is one of the most popular casino games and the only one in which the winning strategy exists for a player. In Blackjack a player competes against a dealer, i.e. "house", which follows a fixed strategy during the game. Blackjack is most commonly played with 6 decks of ordinary cards. The objective of the game is to obtain cards the sum of whose values is as great as possible to 21 without exceeding this number. Cards from 2 to 10 are worth their face values, Jacks, Queens, and Kings are all worth 10 points, and Aces can have a value of either 1 or 11, whichever is the most beneficial to the player. A round of the game starts as soon as the player places the bet. At the beginning of each round, the player is dealt with two cards face up, and the dealer receives one card face up and another card face down. If the player's hand has the total of 21 points, i.e. "Blackjack", the player wins the round unless the dealer has "Blackjack" as well. Otherwise, the player has to select the most appropriate action based on all knowledge the game provides, i.e. player's hand, face up dealer's card, and the cards which left the shoe in previous rounds. Player's choice is limited to the following actions: 1) "hitting" - receiving an additional card; 2) "standing" - refusing from taking subsequent actions; 3) "splitting" - player's hand with two cards of identical denomination can be splitted into two hands which are then played independently; 4) "doubling-down" - doubling the bet, hitting, and then standing; 5) "insurance" - making a side bet against the event that dealer has a blackjack. A simplified version of the game, where allowed actions are limited to hitting and standing, is shown in A.2.

Arguably, the first attempt of deriving the optimal strategy for Blackjack was proposed by Baldwin et al. in 1956 [1]. The derived strategy was shown to lose 0.006 of the betting amount in expectation. It was assumed that the player's bet size is the same for all rounds and the player does not utilize the information about the cards dealt in preceding rounds.

In 1960s Edward Thorpe showed the existence of winning strategy for Blackjack [2]. It was showed that players can estimate the advantage they have from previously dealt cards and vary the bet sizes accordingly. This strategy was the first version of what is widely known today as "cards counting strategy". Subsequent works were mainly concerned about developing modified versions of this cards counting strategy with better advantage estimates [3].

In early 1970s, Blackjack was used to develop the idea of one of the first reinforcement learning algorithms called "selective bootstrap adaptation" [4]. The proposed method managed to converge to the closed-form policy for the reduced action space. However, one major limitation of their approach is that all initial Blackjack configurations were uniformly sampled to obtain sufficient exploration.

Since then different reinforcement learning methods have been applied to solve Blackjack. A.1 summarizes these attempts along with the achieved performance. In limited action space, the state-of-the-art performance of 42.5% win rate was reached by Q-learning and SARSA algorithms [5]. In full action space, the best performance of 0.012 loss of betting amount per round was reached with DQN algorithm [6].

In this work, we applied model-based and model-free value-based methods with different exploration strategies to solve Blackjack for the reduced and full action space. We show that we managed to beat state-of-the-art performance for limited action space. Furthermore, we derived a novel dynamic betting policy and evaluated it under both action spaces. To the best of our knowledge, dynamic betting version of the game has not yet been solved with RL methods.

This report is structured as follows. In Section 2, we present Blackjack modeling, value-based methods, exploration strategies, and dynamic betting policy. Section 3 presents the results of this work. The implications of the obtained results are elaborated on in Section 4.

Foundations of Reinforcement Learning HS2021, Department of Computer Science, ETH Zürich.

2 Methods

In this chapter we first present the applied modeling of Blackjack into a reinforcement learning setting (section 2.1).

Finding an optimal policy in this setting is very hard. Therefore we present an approach to make this problem more feasible with the partitioning of the policy into a static betting policy and a dynamic betting policy (section 2.2).

The first part of this project consists in finding an optimal static betting policy, presented in two separate action spaces with each having a unique state space (sections 2.3, 2.4). The second part of the project is then focused on finding an optimal dynamic betting policy (section 2.6) with the goal, that the combination of the two policies results in a near-optimal policy for the general Blackjack reinforcement learning setting.

2.1 Modeling Blackjack

2.1.1 Blackjack Rules

There is no standardized Blackjack game and each casino uses slightly different rules or variations of the game. In this project we settled on the following Blackjack rules which are thought to be a widely used variant:

- 6 decks in play, with a penetration of 80%
- Dealer hits soft 17
- Blackjack pays 3:2
- Insurance pays 2:1
- Double down allowed after splitting
- No surrender allowed

2.1.2 Markov Decision Process

A Markov decision process (MDP) can be fully described by the tuple $M = (\mathcal{S}, \mathcal{A}, P, r, \mu, \gamma)$, where $\mathcal{S}$ is the set of all possible states, $\mathcal{A}$ the set of all possible actions, $P(s'|s,a) \in P$ the state transition probability, $r(s,a)$ the reward function, μ the initial state distribution and γ the discount factor.

We model Blackjack as an episodic task for each round, ending in one of the three terminal states $S_{\text{terminal}} = \{\text{win}, \text{lose}, \text{draw}\}$. As each episode takes only a handful of state transitions, the discount factor γ is set to 1. The reward function is 0 except for the terminal states, where it is set irrespective of the taken action to $R(\text{win}) = +\text{bet}$, $R(\text{lose}) = -\text{bet}$, $R(\text{draw}) = 0$. The betting amount can be set dynamically by the agent or is set to 1 if it uses static betting. Blackjacks (naturals) are not included in the state space as no action is allowed and are rewarded with $+\frac{3}{2}\,\text{bet}$. With the exception of the model-based method (section 2.3.2), our model is represented as a dynamic environment, with the state transition probabilities $P(s'|s,a)$ changing with the remaining cards in the deck each round.

We use two different MDPs, one with the limited action space $\mathcal{A}' := \{\text{hit}, \text{stand}\}$ and the other with the full action space $\mathcal{A} := \{\text{hit}, \text{stand}, \text{double}, \text{split}, \text{insurance}\}$. This is done to simplify the problem, with the full action space building on top of the limited action space, and to make it possible to specify the value the additional actions (double, split, insurance) offer to the player when we later compare the performance of the agents in each action space. To simplify the (potentially very large) state space an approximation of the space is used, similar to the one used by de Grenville [7]. As this approximation is dependent on the action space, each action space defines its own approximated state space.

2.2 Policy Partitioning

Since the goal of Blackjack is the maximization of the cumulative reward over multiple rounds, the betting amount as an additional action is highly critical. To reduce the complexity of this difficult joint task, we propose the partitioning of the full policy $\hat{\pi}$ into two separate policies: The static betting policy π_s, that optimizes the action in one round and the dynamic betting policy π_d, that optimizes the betting amount before a round.

2.3 Limited Action Space

In the limited action space the action space is reduced to the binary set $\mathcal{A}' = \{\text{hit}, \text{stand}\}$ with the corresponding approximated state space $\mathcal{S}_{\mathcal{A}'}$.

2.3.1 State Space Approximation

The approximated state space in the limited action space $\mathcal{S}_{\mathcal{A}'}$ consists of 363 states, including the three terminal states $S_{\text{terminal}} = \{\text{win}, \text{lose}, \text{draw}\}$. A state is approximated by the sum of the agents hand, the dealers face up card and whether the agents hand is soft or hard. It should be noted though, that some states are unreachable (e.g. the hand $(Q, 2)$ can never be soft) but are intentionally kept as they neither interfere with the used algorithms nor cause any notable loss in computation performance, but allow for an easy analytic mapping to the approximated state space.

2.3.2 Model-based Method

The model-based method takes on a different approach from all other algorithms used in this project. Whereas the model-free methods model each Blackjack round as a new instance of a dynamic environment, the model-based method assigns each Blackjack round a separate MDP, by taking into account the correlation between different rounds (as the cards played in the previous round directly influence the state transition

probabilities of the next round). The state transition probabilities for each MDP are calculated by keeping track of the already played cards (i.e. using card counting).

The model-based method first calculates the state transition probabilities of the MDP of the current round and then performs value-iteration on that MDP to find the optimal action. Since the mapping $\mathcal{S}_{\text{true}} \to \mathcal{S}_{approx}$ is not injective (e.g. the hands (10,7) and (10, 2, 5) are mapped to the same state), the deck at s_{t+1} depends on the previous state s_t (using the previous example, the remaining deck of (10, 2, 5) has one card less in the deck than (10,7), leading to different transition probabilities), breaking the Markov property. This prevents us from easily calculating the state transition probabilities analytically for the approximated state space $\mathcal{S}_{\text{approx}}$. We therefore estimate the state transition probabilities $P(s'|s,a)$ by Monte Carlo estimation, following a similar approach as Geiser and Hasseler in [5]:

$$P(s'|s,a) \approx \frac{N(s',s,a)}{N(s,a)}$$

$$\text{with } N(\cdot): \text{ number of visitations of transition } (\cdot)$$

Due to the rather slow convergence of the described algorithm so far, the state space is further built dynamically at run-time, omitting unreachable states (e.g. the agent cannot travel back to a hand of sum 4 when he currently holds a hand of sum 19). The resulting $\mathcal{S}_{\text{reachable}}$ is much smaller than $\mathcal{S}_{\text{approx}}$, improving the run time of value iteration by a factor of more than 100 (from $\sim$ 5s to $\sim$ 27ms on our machine), with the Monte Carlo method (>1s) now dominating the computation time. Nevertheless, this is still much slower than the model-free methods we used, making it impossible for us to use this algorithm in conjunction with dynamic betting or full action space.

2.3.3 Model-free Methods

Monte Carlo ES
We implemented MC ES algorithm, as it is given in [8] specifically for the game of Blackjack with a small adjustment. In contrast to [8], we do not create artificial rounds starting from or leading to low-visited state action pairs. This avoiding of artificial simulations makes the training process of our Monte Carlo ES noticeable harder.

Q-learning, SARSA, Double Q-learning
Q-learning is a TD control method which is especially powerful for small state/action space. It is guaranteed to converge with probability 1 as long as each state-action pair is continually updated. For completeness, we also implemented two algorithms closely related to Q-learning: SARSA and Double Q-learning. Each of these algorithms has been tried with various exploration polices described in section 2.5.

DQN
We implemented Deep Q-network as a simple fully connected network with ReLU activations and batch normalizaton. A small hatch of 64 experiences was sampled from memory buffer every four iterations. Despite the use of the memory buffer the performance of DON method was unstable and has never overcome the performance of traditional Q-learning. Therefore, we omit the discussion of DQN results.

2.4 Full Action Space

In the full action space the action space is expanded to the set $\mathcal{A} := \{\text{hit, stand, double, split, insurance}\}$ with the corresponding approximated state space $\mathcal{S}_{\mathcal{A}}$.

2.4.1 State Space Approximation

The approximated state space in the full action space $\mathcal{S}_{\mathcal{A}}$ consists of 1443 states, including the three terminal states $S_{\text{terminal}} = \{\text{win, lose, draw}\}$. A state is again approximated by the sum of the agents hand, the dealers face up card and whether the agents hand is soft or hard, but also whether the hand is splittable (consists of same face value cards) and whether the hand consists of only two cards (as double, split and insurance are limited to this hand), as done by Wu [6].

2.4.2 Model-free Methods

SARSA and Q-learning have been used for full action space.

2.5 Exploration Policies

The following policies have been used during training of the aforementioned algorithms, with the goal of accelerated learning. Each policy uses a different exploration-exploitation trade-off strategy.

Random policy

$$\pi_{\text{random}}(a|s) = \text{Unif}(\mathcal{A})$$

Greedy policy

$$\pi_{\text{greedy}}(s) = \arg\max_a Q(s,a)$$

ϵ-greedy policy with exponential decay

$$\pi_{\epsilon-\text{greedy}}(s) = \begin{cases} \arg\max_a Q(s,a) & \text{with prob. } 1-\epsilon(t) \\ \text{Unif}(\mathcal{A}) & \text{with prob. } \epsilon(t) \end{cases}$$

$$\text{with } \epsilon(t) \text{ decaying exponentially}$$

UCB policy

$$\pi_{\text{UCB}}(s) = \arg\max_a Q(s,a) + C\sqrt{\frac{\log N(s)}{N(s,a)}}$$

$$\text{with } N(\cdot): \text{ number of visitations of transition } (\cdot)$$

Boltzmann policy

$$\pi_{\text{Boltzmann}}(a|s) = \frac{e^{\beta Q(s,a)}}{\sum_{a'} e^{\beta Q(s,a')}}$$

with β : 'greedy' parameter, with
$\lim_{\beta \to \infty} \pi_{\text{Boltzmann}}(a|s)$ converging to $\pi_{\text{greedy}}(s)$
and $\pi_{\text{Boltzmann}}(a|s; \beta = 0) = \pi_{\text{random}}(a|s)$

2.6 Dynamic Betting

We propose the following new algorithm for a dynamic betting policy:

Before each round, a new MDP $M_d = (\mathcal{S}_d, \mathcal{A}_d, P_d, r_d(\pi_s))$ is built, with $\mathcal{S}_d$ comprising all possible hands $(\text{hand} = (\text{card}_1, \text{card}_2, \text{card}_{\text{dealer}}))$ in the next round with s_0 serving as initial state, $\mathcal{A}_d = [\text{bet}_{\min}, \text{bet}_{\max}]$ being the set of all allowed bets, $P(s|s_0) \in P_d$ being the probability of drawing a specific hand in the next round and $r_d(\text{hand}, \text{bet}) := \text{bet} \cdot V^{\pi_s}(\text{hand})$ set to the expected return of a specific hand under a static betting policy π_s. To recap, the reward of reaching a specific hand is set to the expected return of that hand, which can be evaluated by the (value-based) static betting policy π_s. Since the static betting policy π_s plays with constant betting amount $\text{bet} = 1$, the expected return of a specific hand under a different betting amount is $\text{bet} \cdot V^{\pi_s}(\text{hand})$. It's important to note, that the dynamic betting policy π_d is highly dependent on the accuracy of the value functions of the static betting policy π_s, which are assumed to have converged to the true ones (under a specific policy). The state transition probabilities P_d can be accurately calculated analytically by keeping track of already played cards (i.e. card counting). With accurate value functions, the Q-function of the initial state s_0 represents a reasonable estimate of the expected return of the next round, played according to the static betting policy π_s:

$$\begin{aligned} Q^{\pi_d}(s_0, a = \text{bet}) &= \sum_{s \in \mathcal{S}_d \setminus \{s_0\}} P(s|s_0) \cdot r_d(s, a = bet) \\ &= \sum_{\text{hand}} P(\text{hand}) \cdot \text{bet} \cdot V^{\pi_s}(\text{hand}) \\ &= \text{bet} \cdot V^{\pi_s}(\text{round}) \end{aligned}$$

where $V^{\pi_s}(\text{round}) = \sum_{\text{hand}} P(\text{hand})\, V^{\pi_s}(\text{hand})$
(expected return of next round under π_s)
and $\text{hand} := s \in \mathcal{S}_d \setminus \{s_0\}$, $P(\text{hand}) := P(s|s_0)$

Applying greedy policy on the Q-functions leads to the following (very risky) policy:

$$\begin{aligned} \pi_{\text{greedy}}(\text{bet}) &= \arg\max_{\text{bet}} Q^{\pi_d}(\text{round}, \text{bet}) \\ &= \begin{cases} \text{bet}_{\max} & V^{\pi_s}(\text{round}) > 0 \\ \text{bet}_{\min} & V^{\pi_s}(\text{round}) \leq 0 \end{cases} \end{aligned}$$

For a more risk aware policy, we recommend the following policy, using the risk heuristic of placing the bet proportional to the amount it expects to win:

$$\pi_{\text{proportional}}(\text{bet}) = \text{bet}_{\max} \cdot V^{\pi_s}(\text{round})$$

3 Results

Each policy is evaluated by two metrics, the mean win rate and the mean loss per round. The mean win rate only evaluates the performance of the static betting policy π_s, whereas the mean loss per round assesses the benefit the dynamic betting policy π_s adds to the static betting policy π_s. Both metrics are evaluated 10 times for each policy in 100'000 testing rounds after each policy has been trained in 1 million rounds. During training, the model-free methods follow one of the exploration policies mentioned in section 2.5. During testing, they follow the greedy policy on the derived Q-values to prevent the algorithms from continuing to explore. Additionally, the following policies serve as baselines for comparison against our methods.

- **Low baseline**: Random agent
  A policy that picks its next action uniformly from the action space.

- **Medium baseline**: Dealer policy
  A policy that plays like the dealer, i.e. hits soft 17 and stands otherwise. We assume that most inexperienced human players play similar to this strategy.

- **Hard baseline**: Optimal policy ("basic strategy")
  A policy that acts according to one of the more popular lookup tables [9], derived mathematically by Thorpe [2], where each card combination is mapped to an action.

The hyperparameters of the model-free methods have been set in both action spaces to the values seen in the table below.

Algorithm	Variable	Value
Learning rate	α	0.05
ϵ-greedy policy	ϵ	0.5
ϵ-greedy policy	base (exp. decay)	0.99999
ϵ-greedy policy	time constant (exp. decay)	1
UCB policy	C	$\sqrt{2}$
Boltzmann policy	β	5

Table 1: Hyperparameters after optimization

3.1 Limited Action Space

3.1.1 Performance of Different Policies

The tables below show the individual performance scores for all policies in the limited action space $\mathcal{A}'$.

Model-based methods

Policy	Mean win rate	Mean loss/round
Hard baseline (optimal policy)	0.433 (±0.003)	−0.025 (±0.005)
Double Q-learning (UCB policy)	0.432 (±0.002)	−0.026 (±0.003)
Double Q-learning (ϵ-greedy policy)	0.432 (±0.001)	−0.027 (±0.001)
Double Q-learning (Boltzmann policy)	0.432 (±0.002)	−0.028 (±0.003)
SARSA (UCB policy)	0.430 (±0.002)	−0.029 (±0.003)
SARSA (Boltzmann policy)	0.431 (±0.002)	−0.029 (±0.003)
Double Q-learning (random policy)	0.432 (±0.002)	−0.029 (±0.004)
SARSA (ϵ-greedy policy)	0.430 (±0.002)	−0.030 (±0.004)
Q-learning (ϵ-greedy policy)	0.429 (±0.002)	−0.031 (±0.003)
Q-learning (UCB policy)	0.429 (±0.003)	−0.032 (±0.006)
Double Q-learning (greedy policy)	0.429 (±0.002)	−0.033 (±0.004)
SARSA (greedy policy)	0.428 (±0.003)	−0.033 (±0.004)
Q-learning (greedy policy)	0.426 (±0.002)	−0.036 (±0.003)
Q-learning (Boltzmann policy)	0.429 (±0.002)	−0.036 (±0.004)
Q-learning (random policy)	0.426 (±0.002)	−0.055 (±0.005)
Medium baseline (dealer policy)	0.410 (±0.002)	−0.060 (±0.004)
SARSA (random policy)	0.411 (±0.002)	−0.097 (±0.005)
Monte Carlo (greedy policy)	0.401 (±0.006)	−0.108 (±0.014)
Low baseline (random policy)	0.298 (±0.002)	−0.342 (±0.004)

Table 2: Performance of model-free polices (1M training rounds, 100k testing rounds), sorted by mean loss per round

Almost all policies were able to surpass both the low and the medium baseline with the best performing methods ranking very close to the hard baseline with the best policy losing on average only one thousandth of the betting amount more than the optimal policy. Performance-wise, reinforcement learning algorithms seem to be able to match the performance of the mathematically calculated optimal policy derived by Thorpe [2].

Among all applied policies, double Q-learning seems to be the most performant, with all double Q-learning algorithms surpassing the performance of the corresponding Q-learning algorithms, using the same exploration policy. Of all applied methods, UCB policy seems to be the best performing with the random policy - having no concept of exploitation - ranking worst. Surprisingly, the natural stochasticity of our (dynamic environment) Blackjack model is enough to balance exploration and exploitation, allowing strictly greedy policy to outperform several other methods.

Model-free methods

Policy	Mean win rate	Mean loss/round
Hard baseline (optimal policy)	0.433 (±0.003)	−0.025 (±0.005)
Value iteration	0.436 (±0.005)	−0.025 (±0.006)
Medium baseline (dealer policy)	0.410 (±0.002)	−0.060 (±0.004)
Low baseline (random policy)	0.298 (±0.002)	−0.342 (±0.004)

Table 3: Performance of model-based policy (10k testing rounds), sorted by mean loss per round

Value-iteration has been applied in the hopes of being able to beat the hard baseline since it can make use of the individual deck compositions in each round. Unfortunately, the high computational cost has not only prevented us from applying the method to the full action space $\mathcal{A}$ and dynamic betting, but also from reaching the aim of 100'000 testing rounds during evaluation. We therefore had to confine the testing rounds to 10'000 rounds and 10'000 Monte Carlo simulations per round. The results (see table 3) are inconclusive if the algorithm has surpassed the optimal baseline even if the higher win rate of value iteration looks promising. Nonetheless, we are optimistic that with increased simulations value iteration will surpass the optimal baseline, though further testing needs to be done to prove this claim.

3.1.2 State Action Table for Final Policy

Below the state action tables of our best performing policy is shown in comparison to the optimal policy found by Thorpe [2]. The actions are colored $\mathcal{A}' = \{\text{hit}, \text{stand}\}$, with the rows of the table showing the sum of the agent's hand and the columns showing the dealer's face up card.

	2	3	4	5	6	7	8	9	10	A
20	s	s	s	s	s	s	s	s	s	s
19	s	s	s	s	s	s	s	s	s	s
18	s	s	s	s	s	s	s	s	s	s
17	s	s	s	s	s	s	s	s	s	s
16	s	s	s	s	s	h	h	h	s	h
15	s	s	s	s	s	h	h	h	h	h
14	s	s	s	s	s	h	h	h	h	h
13	s	s	s	s	s	h	h	h	h	h
12	h	h	h	s	s	h	h	h	h	h
11	h	h	h	h	h	h	h	h	h	h
10	h	h	h	h	h	h	h	h	h	h
9	h	h	h	h	h	h	h	h	h	h
8	h	h	h	h	h	h	h	h	h	h
7	h	h	h	h	h	h	h	h	h	h
6	h	h	h	h	h	h	h	h	h	h
5	h	h	h	h	h	h	h	h	h	h
4	h	h	h	h	h	h	h	h	h	h

Table 4: Our policy for hard hands

	2	3	4	5	6	7	8	9	10	A
20	s	s	s	s	s	s	s	s	s	s
19	s	s	s	s	s	s	s	s	s	s
18	s	s	h	s	s	s	s	h	h	h
17	h	h	h	h	h	h	h	h	h	h
16	h	h	h	h	h	h	h	h	h	h
15	h	h	h	h	h	h	h	h	h	h
14	h	h	h	h	h	h	h	h	h	h
13	h	h	h	h	h	h	h	h	h	h

Table 5: Our policy for soft hands

	2	3	4	5	6	7	8	9	10	A
20	s	s	s	s	s	s	s	s	s	s
19	s	s	s	s	s	s	s	s	s	s
18	s	s	s	s	s	s	s	s	s	s
17	s	s	s	s	s	s	s	s	s	s
16	s	s	s	s	s	h	h	h	h	h
15	s	s	s	s	s	h	h	h	h	h
14	s	s	s	s	s	h	h	h	h	h
13	s	s	s	s	s	h	h	h	h	h
12	h	h	s	s	s	h	h	h	h	h
11	h	h	h	h	h	h	h	h	h	h
10	h	h	h	h	h	h	h	h	h	h
9	h	h	h	h	h	h	h	h	h	h
8	h	h	h	h	h	h	h	h	h	h
7	h	h	h	h	h	h	h	h	h	h
6	h	h	h	h	h	h	h	h	h	h
5	h	h	h	h	h	h	h	h	h	h
4	h	h	h	h	h	h	h	h	h	h

Table 6: Optimal actions for hard hands

	2	3	4	5	6	7	8	9	10	A
20	s	s	s	s	s	s	s	s	s	s
19	s	s	s	s	s	s	s	s	s	s
18	s	s	s	s	s	s	s	h	h	h
17	h	h	h	h	h	h	h	h	h	h
16	h	h	h	h	h	h	h	h	h	h
15	h	h	h	h	h	h	h	h	h	h
14	h	h	h	h	h	h	h	h	h	h
13	h	h	h	h	h	h	h	h	h	h

Table 7: Optimal actions for soft hands

Overall, our policy seems to have converged close to the optimal policy with there being only three cases where the two policies disagree, reaching a similar result as Geiser and Hasseler [5], who encountered three errors of their final policy in their hard table.

3.2 Full Action Space

3.2.1 Performance of Different Policies

The table below shows the individual performance scores for all policies in the full action space $\mathcal{A}$.

Policy	Mean win rate	Mean loss/round
Hard baseline (optimal policy)	0.434 (±0.001)	0.043 (±0.004)
SARSA (UCB policy)	0.434 (±0.002)	−0.017 (±0.007)
Q-learning (UCB policy)	0.434 (±0.001)	−0.017 (±0.011)
Q-learning (ϵ-greedy policy)	0.431 (±0.001)	−0.024 (±0.010)
SARSA (greedy policy)	0.43 (±0.002)	−0.025 (±0.008)
Q-learning (greedy policy)	0.431 (±0.002)	−0.028 (±0.010)
SARSA (ϵ-greedy policy)	0.430 (±0.002)	−0.029 (±0.009)
SARSA (Boltzmann policy)	0.433 (±0.002)	−0.034 (±0.010)
Q-learning (Boltzmann policy)	0.433 (±0.001)	−0.034 (±0.080)
Q-learning (random policy)	0.431 (±0.002)	−0.050 (±0.005)
Medium baseline (dealer policy)	0.410 (±0.002)	−0.060 (±0.004)
SARSA (random policy)	0.426 (±0.002)	−0.075 (±0.007)
Low baseline (random policy)	0.301 (±0.001)	−0.407 (±0.005)

Table 8: Performance of different polices (1M training rounds, 100k testing rounds), sorted by mean loss per round

We observe that all baselines have surpassed the low baseline by a margin and all but but one also the medium baseline. The exploration policies rank similarly as in the limited action space $\mathcal{A}'$, with UCB policy again performing best and random policy worst, with the exception of Boltzmann policy faring worse than in the limited action space $\mathcal{A}'$ compared to the other policies, which could be caused by suboptimal hyperparameter selection.

3.2.2 State Action Table for Final Policy

Below the state action tables of our best performing policy is shown in comparison to the optimal policy found by Thorpe [2]. The actions are colored $\mathcal{A} = \{$hit, stand, double, split$\}$, with the rows of the table showing the sum of the agent's hand and the columns showing the dealer's face up card.

	2	3	4	5	6	7	8	9	10	A
20	s	s	s	s	s	s	s	s	s	s
19	s	s	s	s	s	s	s	s	s	s
18	s	s	s	s	s	s	s	s	s	s
17	s	s	s	s	s	s	s	s	s	s
16	h	s	s	s	s	s	d	d	d	s
15	s	s	s	s	s	d	d	d	d	d
14	s	s	s	s	s	d	h	h	h	h
13	s	s	s	s	s	d	d	d	d	h
12	h	d	h	h	s	h	h	h	h	h
11	d	d	d	d	d	d	d	d	d	d
10	d	h	h	d	d	h	h	h	h	h
9	h	h	h	h	h	h	h	h	h	h
8	h	h	h	h	h	h	h	h	h	h
7	h	h	h	h	h	h	h	h	h	h
6	h	h	h	h	h	h	h	h	h	h
5	h	h	h	h	h	h	h	h	h	h
4	h	h	h	h	h	h	h	h	h	h

Table 9: Our policy for hard hands

	2	3	4	5	6	7	8	9	10	A
20	s	s	s	s	s	s	s	s	s	s
19	s	s	s	s	s	s	s	s	s	s
18	s	s	s	s	h	s	s	h	d	s
17	d	d	d	d	d	h	d	d	d	d
16	h	h	h	d	d	h	d	h	h	h
15	d	h	h	h	d	h	d	h	h	d
14	h	h	h	h	d	h	h	h	d	d
13	h	h	h	h	h	h	h	h	h	h

Table 10: Our policy for soft hands

	2	3	4	5	6	7	8	9	10	A
20	s	s	s	s	s	s	s	s	s	s
19	-	-	-	-	-	-	-	-	-	-
18	-	-	-	-	-	-	-	-	-	-
17	s	s	s	s	s	s	s	s	s	s
16	s	s	s	s	s	s	s	s	s	-
15	s	s	s	s	s	s	-	-	-	-
14	-	-	-	-	-	-	-	-	-	-
13	-	-	-	-	-	-	-	-	-	-
12	-	-	-	-	-	-	-	-	-	-
11	-	-	-	s	-	-	-	-	-	-

Table 11: Our policy for splittable hands

	2	3	4	5	6	7	8	9	10	A
20	s	s	s	s	s	s	s	s	s	s
19	s	s	s	s	s	s	s	s	s	s
18	s	s	s	s	s	s	s	s	s	s
17	s	s	s	s	s	s	s	s	s	s
16	s	s	s	s	s	h	h	h	h	h
15	s	s	s	s	s	h	h	h	h	h
14	s	s	s	s	s	h	h	h	h	h
13	s	s	s	s	s	h	h	h	h	h
12	h	h	s	s	s	h	h	h	h	h
11	d	d	d	d	d	d	d	d	d	d
10	d	d	d	d	d	d	d	d	h	h
9	h	d	d	d	d	h	h	h	h	h
8	h	h	h	h	h	h	h	h	h	h
7	h	h	h	h	h	h	h	h	h	h
6	h	h	h	h	h	h	h	h	h	h
5	h	h	h	h	h	h	h	h	h	h
4	h	h	h	h	h	h	h	h	h	h

Table 12: Optimal actions for hard hands

	2	3	4	5	6	7	8	9	10	A
20	s	s	s	s	s	s	s	s	s	s
19	s	s	s	s	d	s	s	s	s	s
18	d	d	d	d	d	s	s	h	h	h
17	h	d	d	d	d	h	h	h	h	h
16	h	h	d	d	d	h	h	h	h	h
15	h	h	d	d	d	h	h	h	h	h
14	h	h	h	d	d	h	h	h	h	h
13	h	h	h	d	d	h	h	h	h	h

Table 13: Optimal actions for soft hands

	2	3	4	5	6	7	8	9	10	A
20	s	s	s	s	s	s	s	s	s	s
19	-	-	-	-	-	-	-	-	-	-
18	s	s	s	s	s	-	s	s	-	-
17	s	s	s	s	s	s	s	s	s	s
16	s	s	s	s	s	s	-	-	-	-
15	s	s	s	s	s	-	-	-	-	-
14	-	-	-	-	-	-	-	-	-	-
13	-	-	-	s	s	-	-	-	-	-
12	s	s	s	s	s	s	-	-	-	-
11	s	s	s	s	s	s	-	-	-	-

Table 14: Optimal actions for splittable hands

Overall, it can be seen that our policy is similar to the optimal policy but there are a lot more circumstances (78 out of 350) where the policies disagree on the optimal action compared to the limited action space. This falls in line with the final policy of Wu [6] which differs from the optimal policy in 93 cases. In general though, our policy follows the same state-action patterns as the optimal policy, although not as refined as in the latter as it seems to be still in its learning phase.

3.3 Dynamic Betting

When evaluating our dynamic betting policy, a pretrained static betting policy is augmented with our method and two other policies for comparison:

- **Low baseline**
  Static betting (no augmented dynamic betting policy)
- **Hard baseline**
  "Hi-Lo" dynamic betting policy [10] - a card counting strategy used by many experienced Blackjack players, where the bet increases with the ratio of high value cards to low value cards left in the deck, as high value cards are beneficial to the player in general

Hi-Lo itself is dependent on the performance of the underlying static betting policy, as it simply raises the bet with the ratio of high-value cards left in the deck. In the limited action space $\mathcal{A}'$, no policy seems to be performant enough as Hi-Lo consistently decreases the performance of all policies and is therefore not listed in section 3.3.1.

SARSA with Boltzmann exploration was used in both action spaces as static betting policy with 1 million training rounds in the limited action space $\mathcal{A}'$ and 10 million training rounds in the full action space $\mathcal{A}$ with 100'000 testing rounds each.

3.3.1 Limited Action Space

In the limited action space $\mathcal{A}'$ we can differentiate two very different results in two different settings:

For the Blackjack setting described in section 2.1.1, with 6 decks and 80% penetration, the dynamic betting policy never bets more than the minimum betting amount and therefore behaves exactly like a static betting policy. In contrast, when equipped with a more advantageous deck, e.g. a single deck with 90% penetration, the dynamic betting policy increases its bet frequently and leads to a large positive net return, even surpassing the hard baseline by a margin.

The two figures below show this behaviour graphically, with the top plot showing the evolution of money over time and the bottom plot the betting amount over time under no dynamic betting policy and with dynamic betting policy.

Six decks

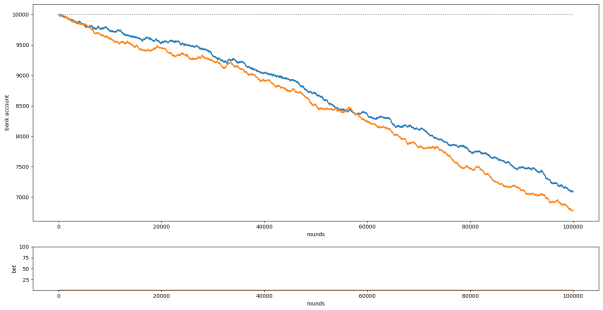

Figure 1: 6 decks, 80% penetration

Policy	Mean loss per round
Dynamic betting policy	$-0.027\ (\pm 0.004)$
Low baseline (static betting)	$-0.030\ (\pm 0.002)$
Hard baseline (Hi-Lo)	$-0.039\ (\pm 0.009)$

Table 15: Performance (6 decks, 80% penetration)

Single deck

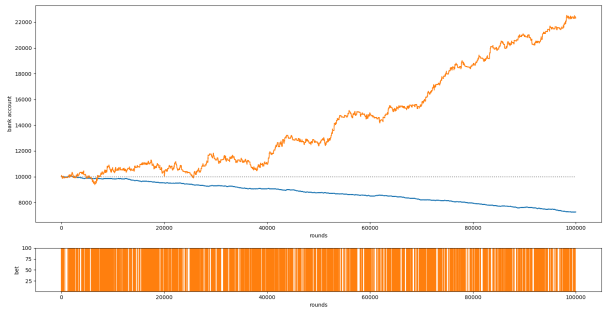

Figure 2: 1 deck, 90% penetration

Policy	Mean loss per round
Dynamic betting policy	$0.141\ (\pm 0.033)$
Low baseline (static betting)	$-0.026\ (\pm 0.003)$
Hard baseline (Hi-Lo)	$-0.039\ (\pm 0.006)$

Table 16: Performance (1 deck, 90% penetration)

For the limited action space, we derive the following two conclusions:

1. The dynamic betting policy works as intended and is able to provide large positive returns for smaller, more advantageous decks, even beating conventional card counting strategies such as Hi-Lo [10].

2. For larger decks, Blackjack as a game is too unfair when the agent only has access to the two actions $\{\text{hit}, \text{stand}\}$. The fact that also Hi-Lo [10] is unable to provide positive net return in that setting seems to affirm this statement.

3.3.2 Full Action Space

The additional training time for the full action space is needed to animate the dynamic betting policy to increase its bet more frequently. With only 1 million training rounds, the algorithm estimates on average an expected return of ~ -0.2 and consequently almost never increases its bet. With 10 million training rounds, the algorithm raises its average expected return to ~ -0.15 - with the Q-values seemingly being more accurate, even though the performance improves only marginally (see table 8) - and is betting more frequently as occasional advantageous decks are more likely to result in positive expected return. Notice that both values are far from the empirically calculated one (see table 8). We expect our estimate to become more accurate with more training time and thereby further improving the performance of the dynamic betting policy.

The figure below shows the evolution of money over time and the betting amount over time under no dynamic betting policy, dynamic betting policy and Hi-Lo.

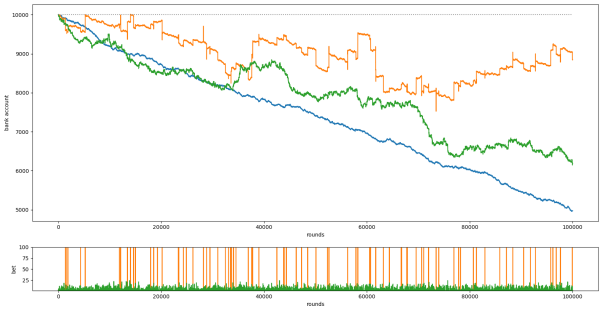

Figure 3: 6 decks, 80% penetration

Policy	Mean loss per round
Dynamic betting policy	$-0.016\,(\pm 0.035)$
Hard baseline (Hi-Lo)	$-0.027\,(\pm 0.026)$
Low baseline (static betting)	$-0.033\,(\pm 0.01)$

Table 17: Performance (6 decks, 80% penetration)

In conclusion, even though the dynamic betting policy is unable to give positive net return consistently, it still improves the performance of the static betting policy considerably.

4 Discussion

We have demonstrated near-optimal policies in the limited action space $\mathcal{A}'$, with the best policy losing only one thousandth of the betting amount more than the optimal policy derived by Thorpe [2]. Our final policy converges very close to the optimal policy (see section 3.2.2), confirming the results from Geiser and Hasseler [5] under new methods, showing that reinforcement learning is able to find the optimal policy in the limited action space $\mathcal{A}'$. Additionally, we have presented a model-based method (see section 2.3.2) that could be better than the optimal policy found by Thorpe [2], by taking the deck composition in each round into account. Unfortunately, we were neither able to prove nor disprove this claim, requiring further investigation into this method.

In the full action space $\mathcal{A}$ on the other hand, we encountered the problem that our final policy differs substantially from the optimal policy (see section 3.2.2). We believe that the full action space $\mathcal{A}$ represents a much more complex problem than the limited action space $\mathcal{A}'$ with a much larger state and action space and that our final algorithms haven't actually converged during our evaluation. Still, our results are comparable to the ones from Wu [6] with his final policy also showing substantial differences to the optimal policy. Unfortunately, we were not able to improve notably upon the existing literature, although we believe that more training time or more sample-efficient methods should improve the performance of the final policy considerably.

Furthermore, we have derived a novel dynamic betting policy - something that was missing in the literature before - and evaluated its performance under both action spaces. We have shown in both action spaces, that our method is able to improve the performance of the static betting policy considerably. While we were able to "solve" Blackjack for smaller decks, we were unable to consistently achieve positive net return for larger, more conventionally used decks. While we believe that inaccuracies of the value functions of the static betting policy, for which our method is sensitive, due to them having not yet converged, is culpable, further research is needed to proof if our method is able to "solve" Blackjack in the general case.

References

[1] Roger R Baldwin, Wilbert E Cantey, Herbert Maisel, and James P McDermott. The optimum strategy in blackjack. *Journal of the American Statistical Association*, 51(275):429–439, 1956.

[2] Edward Thorp. A favorable strategy for twenty-one. *Proceedings of the National Academy of Sciences of the United States of America*, 47(1):110, 1961.

[3] Graham Kendall and Craig Smith. The evolution of blackjack strategies. In *The 2003 Congress on Evolutionary Computation, 2003. CEC'03.*, volume 4, pages 2474–2481. IEEE, 2003.

[4] Bernard Widrow, Narendra K Gupta, and Sidhartha Maitra. Punish/reward: Learning with a critic in adaptive threshold systems. *IEEE Transactions on Systems, Man, and Cybernetics*, (5):455–465, 1973.

[5] Joshua Geiser and Tristan Hasseler. Beating blackjack - a reinforcement learning approach. *Stanford University*.

[6] Allen Wu. Playing blackjack with deep q-learning.

[7] Charles De Granville. Applying reinforcement learning to blackjack using q-learning. *University of Oklahoma*.

[8] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[9] Blackjack strategy charts - how to play perfect blackjack, Apr 2021. URL `https://www.blackjackapprenticeship.com/blackjack-strategy-charts/`.

[10] How to count cards in blackjack and bring down the house, May 2021. URL `https://www.blackjackapprenticeship.com/how-to-count-cards/`.

[11] Daniel Kenneth Olson. Learning to play games from experience: An application of artificial neural networks and temporal difference learning. 1993.

[12] Andrés Pérez-Uribe and Eduardo Sanchez. Blackjack as a test bed for learning strategies in neural networks. In *1998 IEEE International Joint Conference on Neural Networks Proceedings. IEEE World Congress on Computational Intelligence (Cat. No. 98CH36227)*, volume 3, pages 2022–2027. IEEE, 1998.

A Appendix

A.1 Approaches to Solve Blackjack with RL Algorithms

Year, Ref.	**Algorithm**	**Action Space**	**Num. of Decks**	**Win rate or loss per round**
1973 [4]	Selective bootstrap adaptation	hit, stand	1	43%
1993 [11]	Neural Networks with temporal difference	hit, stand	1	41.1%
1998 [12]	Neural Network / Q-Learning / SARSA	hit, stand	1	41.7 %
2003 [3]	Three Neural Networks	hit, stand, split, double	6	5% worse than the optimal
2005 [7]	Q-Learning	hit, stand, split, double	6	rougly 0.2 bets per round worse than the optimal
2018 [6]	Deep Q-Learning	hit, stand, split, double	infinite	0.01167 bets per round worse than the optimal
2020 [5]	Value Iteration, Sarsa, and Q-Learning	hit, stand	6	42.5%

Table 18: Attempts to apply RL algorithms to solve Blackjack

A.2 The Rules of the Game of Blackjack Expressed in Algorithmic Language

Algorithm 1 A round of Blackjack with limited action space $\mathcal{A} := \{hit,\ stand\}$

Initialize:
- Bet ▷ Player places a bet
- $PlayerHand$ ▷ Player is dealt with two cards
- $DealerHand$ ▷ Dealer is dealt with two cards

Play a Round:

if PlayerHand $\&$ $DealerHand$**have** $aBlackjack$ **then** Reward = 0 ▷ Draw - player receives his bet back =0
- **return** ▷ Finish the round
- PlayerHand **has** a Blackjack
- $Reward = 1.5 * Bet$ ▷ For Blackjack player gets 3/2 reward
- **return**
- DealerHand **has** a Blackjack
- $Reward = -Bet$ ▷ Player completely loses his bet
- **return**

while $PlayerAction$ **is** hit **do**
- $Add\ new\ card\ to\ PlayerHand$
- **if** PlayerHand **is** Busted **then**
  - $Reward = -Bet$ ▷ Player loses if he is busted
  - **return**
- **end if**

end while

$DealerHand \leftarrow DealerStrategy(DealerHand)$ ▷ Dealer acts according to the fixed strategy

if DealerHand **is** Busted **then**
- $Reward = Bet$ ▷ Player wins if dealer is busted
- **return**

end if

if DealerHand **is worth more** than PlayerHand **then**
- $Reward = -Bet$ ▷ Player loses
- **return**

else if DealerHand **is worth less** than PlayerHand **then**
- $Reward = Bet$ ▷ Player wins
- **return**

else
- $Reward = 0$ ▷ Draw
- **return**

end if

A.3 Learning Curves

A.3.1 Limited Action Space

Below, the learning curve during training of double Q-learning under the exploration strategies mentioned is shown. Double Q-learning was trained 10 times with 1 million episodes for each exploration strategy.

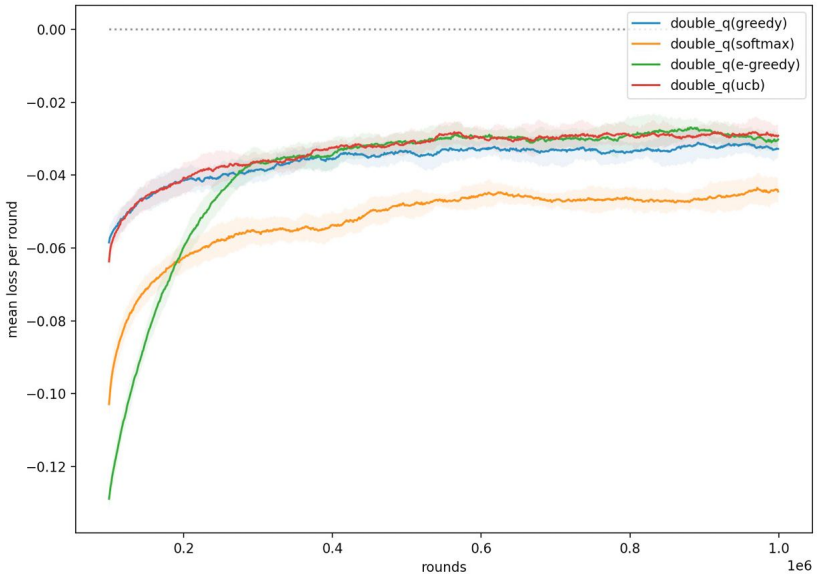

Figure 4: Learning curve of double Q-learning with different exploration policies

A.3.2 Full Action Space

Below, the learning curve during training of SARSA under the exploration strategies mentioned in section 2.5 is shown. SARSA was trained 10 times with 1 million episodes for each exploration strategy.

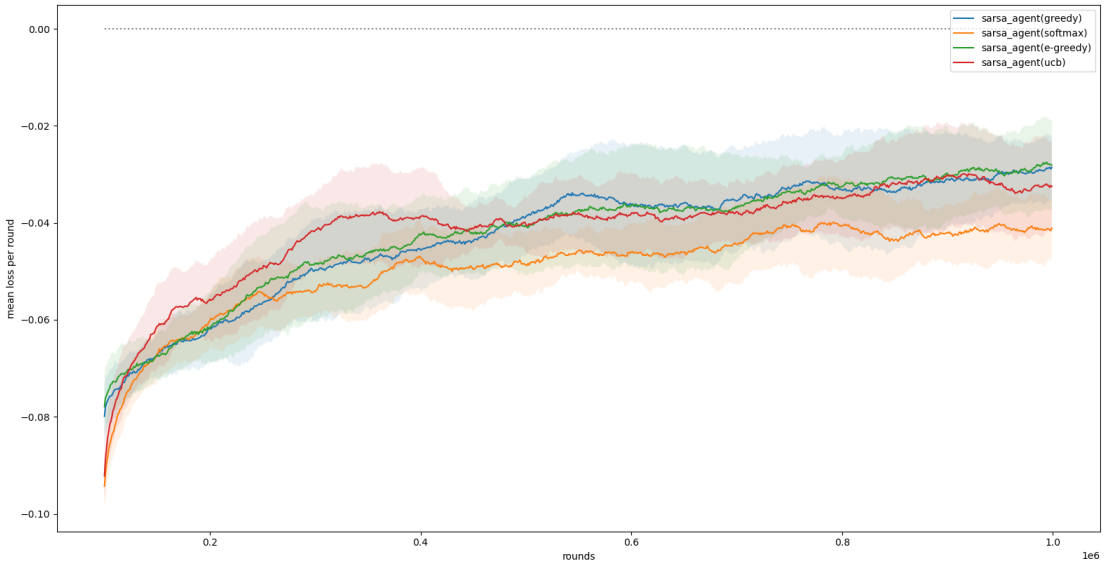

Figure 5: Learning curve of SARSA with different exploration policies