# Deep Reinforcement Learning for Blackjack

Marius Oechslein
*Faculty of Computer Science and Business Information Systems*
*University of Applied Sciences Würzburg-Schweinfurt*
Würzburg, Germany
marius.oechslein@study.thws.de

***Abstract*—**
***Index Terms*—Blackjack, Reinforcement Learning, Monte Carlo**

## I. Introduction

### A. Blackjack and Reinforcement Learning

Blackjack is a popular card game played in casinos worldwide. In 1962 the mathematician Edward Thorb published a Book on how to beat the house in Blackjack [1]. This made is possible for players to win money in the game of Blackjack in casinos. Although the casinos changed their rules which makes winning for the player impossible now, the game is still an interesting subject to mathematical modeling due to its probabilistic nature.

Edward Thorb introduced the idea of card counting with the Complete Point Count System [1] which makes it possible for players to keep track of a score while playing that indicates their chances of winning. The Complete Point Count System keeps track of a counter that adds +1 or -1 based on each card that are seen by the player. This easy counting method makes it possible for players to count cards while playing the game.

### B. Reinforcement Learning for Blackjack

The game can be modeled as a Markov Decision Process which makes it possible to apply Reinforcement Learning. With todays computation power it is now possible to let the computer play over 100.000.000 games, which wasn't possible in the 1960s. The baseline goal is therefore to at least achieve the basic strategy of Edward Thorb [1] with Reinforcement Learning methods.

All of the Reinforcement Learning methods play a large number of games while keeping track of a Q-Table containing the expected returns to each of the possible game states [**?**]. The different Reinforcement Learning methods like Monte Carlo Control, SARSA and Q-Learning only take a different approach of updating this Q-Table [**?**]. Although these Reinforcement Learning methods are very powerful, they require a high amount of games to be able to model the Q-Table as the state-space of the game increases. In this paper we therefore investigate how well the Reinforcement Learning methods Monte Carlo Control, SARSA and Q-Learning perform as we increase the state-space by changing the card counting method. We change the card counting method by keeping track of the values of every seen card and not just keeping track of a counter.

### C. Deep Reinforcement Learning

In 2013 DeepMind published the paper *Playing Atari with Deep Reinforcement Learning* [2], Deep Learning and Reinforcement Learning were combined. The idea of Deep Reinforcement Learning is to replace the Q-Table with a deep neural network. This makes it possible to estimate the expected returns even for complex state-spaces.

In this paper we first investigate whether Deep RL is also able to achieve the baseline of the basic strategy of Edward Thorb [1]. Secondly we compare how the Deep RL performs compared to the more traditional RL-methods like Monte Carlo, SARSA and Q-Learning [4] as the state-space increases.

## II. Methods

### A. The Blackjack Environment

The game of Blackjack starts with the dealer and the player recieving two cards where the player is only allowed to see one of the dealer's cards and his own cards. The player only has the two actions to choose from: taking another card (Hitting) and not taking another card (Standing). Although the real rules of Blackjack extend this action-space by options like Doubling Down and Splitting, in this paper we only focus on Standing and Hitting.

The player can then hit as many times as he wants with the goal of coming as close to the hand value of 21 as possible. After the player finished his turn the dealer hits as long as his hand value are below or equal 16. At the end of one game the player's return is +1, -1 or 0 based on whether it was a win, loss or draw.

To model the basic strategy with Reinforcement learning, one game is modeled as one episode. This is possible since the basic strategy only focuses on whether the player should hit or stand based on the dealer's card value and the player's hand value. Therefore only the observed card of the dealer and the hand value of the player is of interest for deciding on hitting and standing which makes the rest of the deck uninteresting.

When introducing card counting one episode has to be extended by playing multiple games in one episode. This is necessary since card counting is dependent on seeing a high number played cards since the player's chances change depending on the ratio of low cards/ high cards played. In the case of one game the player on average see 3-5 cards which is not enough for an imbalanced ratio between high and low cards. Therefore one episode is extended to playing through one whole deck.

The Complete Point Count System [1] keeps a running score and updating it every time the player sees a card:

- +1 for: 2, 3, 4, 5, 6, 7
- -1 for: 10, A
- 0 for: 8, 9

It is favourable for the player when the running score is high [1]. And it is favourable for the dealer when the running score is low. This is because the dealer's chances of busting increases when there are only few low cards left in the deck.

### *B. State-action space*

For learning the basic strategy the state consists of the **player's hand value**, the **dealer's card value** and if the player has an **usable ace**. This makes a total number of **250 states** that need to be considered:

- 10 possible dealer states: 2,3,4,5,6,7,8,9,10,A
- 25 possible player states:
  No Ace: 4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20
  With Ace: 13,14,15,16,17,18,19,20

For learning the Complete Point Count System the state has to be extended by a **running score**. The running score in one game can at most be +24 (= 6 cards * 4 suits) and at least -20 (= 5 cards * 4 suits). This would increase the state-space to **11.000 states** (= 250 states * (24 - 20)) for the Complete Point Count System. Although a running score around 0 is far more likely than of +24 or -20.

When making the card counting method more complex by counting every value of the seen cards, the state space increases to approximately **907.000.000 states** (= 250 states * ((52 cards / 4 suits) - 3 face cards)! ) for playing through one card deck. Altough the actual state-space is a little less considering that the player can only see one card of the dealer. The most important thing to note here is that the state-space explodes when using this more complex card counting method.

With this huge state space it is interesting to observe how Deep Reinforcement Learning performs and how other Reinforcement Learning methods perform compared to that.

### *C. Reinforcement Learning methods*

All of the Reinforcement Learning methods share the basic approach of taking actions in an environment, observing the reward and keeping track of the expected reward to each state-action [4]. One more thing that all of the RL methods have in common is that they have to play a large number of episodes to be able to estimate the expected rewards well.

The implementation of RL methods Monte Carlo Control, SARSA and Q-Learning mainly differ on the following characteristics [4]:

- **On-policy vs. Off-policy Learning**: If the same policy is used for exploration and exploitation steps.
- **Update Rule**: With what information the updates are done.
- **Bootstrapping**: If estimations are done based on prediction of future values.

These are the terminologies used for the rest of this paper:

- Q: action-value function
- V: state-value function
- G: cumulative reward after state is visited until the end of the episode
- R: reward
- S: state
- A: action
- t: timestep
- $\gamma$: discount rate
- $\alpha$: learning rate

*1) Monte Carlo Control:* Monte Carlo Control (MC) was introduced in the book *Reinforcement learning: An introduction* [4]. MC is an on-policy learning method which uses the epsilon-greedy method for deciding on whether to take a exploration or an exploitation step. This is the update rule of Monte Carlo Control:

$$V(S_t) \leftarrow V(S_t) + \alpha[G_t - V(S_t)] \tag{1}$$

MC does not use Bootstrapping which means that the updates are calculated based on only real observations. This property makes Monte Carlo Control converge slower compared to methods that use Bootstrapping [4].

*2) SARSA:* The State-Action-Reward-State-Action (SARSA) method was also introduced in the book *Reinforcement learning: An introduction* [4] and counts as a Temporal Difference Learning method. SARSA is also an on-policy learning method which means that the same policy is used for exploration and exploitation steps. For the update rule SARSA takes the reward of the next state in consideration:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)] \tag{2}$$

This means that SARSA uses bootstrapping since $Q(S_{t+1})$ is used for updating $Q(S_t, A_t)$.

*3) Q-Learning:* Q-learning first introduced by Watkins [5] and taken up in [4] and it also counts as a Temporal Difference Learning method. Q-learning is an off-policy learning method that uses bootstrapping [4]:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)] \tag{3}$$

Although on-policy learning has its advantages, for the case of Blackjack the off-policy learning is expected to perform better since the target policy does not have to do exploration steps which lower the expected reward.

### *D. Deep Q-Learning*

Deep Q-Learning (DQN) originated in 2013 in the DeepMind paper *Playing Atari with Deep Reinforcement Learning* [2]. The main idea of DQN to replace the Q-Table with a deep neural network [2]. Further, two important implementation details are the usage of an **Experience Replay Buffer** and a **Target Neural Network** which are presented in the following section.

The pseudo-code for the DQN can be inspected in the *Playing Atari with Deep Reinforcement Learning* paper [2]. For a

deeper explanation of the DQN architecture is *Implementing the Deep Q-Network* [6] a helpful resource.

*1) Experience Replay Buffer:* At each step of the training loop the Experience Replay Buffer is filled with the Experience of this training step. An Experience consists of:

- state
- action
- reward
- next state

At every training step a random mini-batch of the Buffer is then sampled and used for the training of the neural network.

The Experience Buffer has the advantage of reusing Experiences for training the neural network which increases the data effiency [2]. Additionally, sampling random mini-batches uniformly introduces the advantage of breaking the dependency of a state with its preceeding state which achieves a more general neural network [2].

The maximum size of the Buffer is set to 10.000. When the maximum size is reached, the oldest Experiences are removed and for new Experiences to be added.

*2) Target Neural Network:* In traditional Q-Learning the current Q-value is updated by the estimated Q-value of the next state. To facilitate the same update rule in Deep Q-Learning a Target Neural Network, called $Q^*$ in the DeepMind paper [2], is used for predicting the returns of the next state [6]. These predicted returns of the next state are then used for the loss function and ultimately for the gradient descent of the learning neural network [6].

The Target Neural Network is initialized with the same architecture as the learning neural network. The weights of the Target Neural Network are fixed and only get updated by the weights of the training network. In the paper [6] this update of the weights is done only every few training steps with the advantage of faster training time. In the implementation for this paper, the updates are done every training step but with a update rate of only 0.005 to prevent overfitting.

The advantages of using the target neural network are more stable training and that the errors in estimation are better controlled [6].

## III. Results

### A. Basic Strategy

### B. Increased state space - counting all cards

## IV. Discussion

## V. Conclusion

## References


[1] Thorp, E. O. (1966). Beat the dealer: A winning strategy for the game of twenty-one (Vol. 310). Vintage.

[2] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. (2013). Playing atari with deep reinforcement learning. arXiv preprint arXiv:1312.5602.

[3] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., ... & Hassabis, D. (2015). Human-level control through deep reinforcement learning. nature, 518(7540), 529-533.

[4] Sutton, R. S., & Barto, A. G. (2018). Reinforcement learning: An introduction. MIT press.

[5] Watkins, C. J., & Dayan, P. (1992). Q-learning. Machine learning, 8, 279-292.

[6] Roderick, M., MacGlashan, J., & Tellex, S. (2017). Implementing the deep q-network. arXiv preprint arXiv:1711.07478.