

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 containing strategies on how to win at Blackjack [1]. This made it possible for players to win against casinos in the game of Blackjack. Since then the casinos changed their rules which makes winning for the player impossible now, but the game remains 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]. 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. 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 [4]. The different Reinforcement Learning methods like Monte Carlo Control, SARSA and Q-Learning only take a different approach of updating this Q-Table [4]. Although they are powerful methods, due to their nature of learning, 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 these Reinforcement Learning methods (Monte Carlo Control, SARSA and Q-Learning) perform as we increase the state-space by changing the card counting method. We make the card counting method more complex by keeping track of the values of every observed card compared to a single counter in the case of the Complete Point Count System [1].

C. Deep Reinforcement Learning

In 2013 DeepMind combined Deep Learning with Reinforcement Learning in the paper *Playing Atari with Deep*

Reinforcement Learning [2]. The idea of Deep Reinforcement Learning is to replace the Q-Table with a deep neural network. This enables estimating Q-Tables for complex state-spaces that traditional Reinforcement Learning methods were unable to.

In this paper we first investigate whether Deep Reinforcement Learning is also able to achieve the baseline of the basic strategy of Edward Thorb [1]. Secondly we compare how the Deep Reinforcement Learning method performs compared to the more traditional RL-methods like Monte Carlo Control, 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 receiving two cards - with the player only seeing one of the dealer's cards and his own hand. The player then 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. It was decided to focus on this reduced action-space since the main points of interest are the Reinforcement Learning methods and not the accurate representation of the game.

The player can then hit as many times as he wants with the goal of coming as close to a hand value of 21 as possible. After the player finished his turn the dealer hits as long as his hand value is below or equal 16. At the end of one game the player's return is two times his bet, his bet or nothing depending on whether it was a win, loss or draw. For Reinforcement Learning the rewards is chosen as +1, -1 and 0.

To model the basic strategy with Reinforcement learning, one game is modeled as one episode. It is possible to model only one game as one episode since the basic strategy works only with the information of the dealer's card and the player's hand value while information about the rest of the deck is uninteresting.

When introducing card counting on the other hand, one episode has to be extended to playing multiple games within one episode. This is necessary because card counting works with the information of cards already played in addition to the cards of the current game. The relation between high cards to low cards left in the deck influence the player's chances of winning. When increasing one episode to playing through a

whole deck, the chances of an imbalanced card deck increase which therefore makes card counting more interesting.

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 counting score is high [1]. And it is favourable for the dealer when the counting score is low. The main reason for that is that the dealer's chances of busting increases when there are more high cards than 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 **counting score**. The counting score in one episode 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 it is important to note that a counting 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. Although the actual state-space is a little less considering that the player can only see one card of the dealer's cards. 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 [4].

Characteristics that the mainly define different RL-methods are the following [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 updates 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 visited state until the end of the episode
- R: reward
- S: state
- A: action
- t: timestep
- γ : discount rate
- α : 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:

$$V(S_t) \leftarrow V(S_t) + \alpha[G_t - V(S_t)] \quad (1)$$

MC does not use Bootstrapping which means that the updates are calculated based only on 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 is part of the Temporal Difference Learning methods family. SARSA is also an on-policy learning method which means that the same policy is used for exploration and exploitation steps which is also done with epsilon-greedy. For the update rule, SARSA takes the reward of the next state in consideration:

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

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 also counts as a Temporal Difference Learning method. Q-learning is an off-policy learning method that uses bootstrapping [4]:

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

D. Deep Q-Learning

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

Pseudo-code for the DQN can be found in the *Playing Atari with Deep Reinforcement Learning* paper [2]. For implementation details of the DQN architecture, *Implementing the Deep Q-Network* [6] is a helpful resource.

1) *Experience Replay Buffer*: At each step of the training loop the Experience Replay Buffer is filled with the Experience of the current 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 introduces the advantage of reusing Experiences for training the neural network which increases the data efficiency [2]. Additionally, sampling random mini-batches uniformly introduces the advantage of breaking the dependency between a state and its preceeding state which helps training 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*: The traditional Q-Learning method [4], 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 network [6].

The target 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 advantage of using the target neural network is 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., Belle-mare, 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.