Applying Reinforcement Learning to the Game of Wizard

Callum Waters [1] **Jonas Dippel** [1] **Kai Jeggle** [1] **Til Jasper Ullrich** [1]

Abstract

In this paper, reinforcement learning is applied to the card game Wizard. We present *meRLin*, an agent with a two phase architecture that uses Proximal Policy Optimization for playing cards and a separate neural network for making trick predictions. Our proposed agent significantly outperforms rule based and other reinforcement learning approaches. It achieves an average score of xx% and an average win rate of xx% in a game against three other opponents. Evaluation against human players shows that *meRLin* is often able to win in the first few games, but then human players are able to adapt fast and exploit its weaknesses.

1. Introduction / Related Work — Callum

Games have for a long time been an ideal playground for advancing the field of artificial intelligence. From playing Atari games (**?**) to the development of Alphago (Silver et al., 2016) and Alphazero (Silver et al., 2017), games offer a complete environment with endless variety to continually challenge the frontiers of artificial intelligence.

Games like Chess or Go are complete information two-player zero-sum games meaning that the whole game state is observable for the players and the game is played optimally with the Nash equilibrium strategy. In many real world problems, more than two actors are involved and they are not able to observe everything that is related to the problem. Therefore, looking at multi-player incomplete information games is particularly interesting. One famous multi-player incomplete information game is *Poker*. It has been heavily studied by several groups and recent work achieves superhuman performance (**?**) in a multi-player setting. Like many others, they use a combination of Monte-Carlo Tree Search (Silver et al., 2016) and Counterfactual Regret Minimization (Brown et al., 2018).

Compared to this approach, we want to investigate the performance of vanilla reinforcement learning on multi-player incomplete information games. Especially, we investigate the game of *Wizard* which is a multi-player card game where the opponents' cards are hidden from the player and players increase their score only if they can correctly predict how many games they will win, requiring an understanding of card strategies and their effects. The game rules are further described in **??**.

Reinforcement learning has also been successfully applied to other card games (**?**)

The only academic work which applied reinforcement learning to *Wizard* that is known to us is (Backhus et al., 2013). They use Deep Q-learning. However they came across considerable difficulty in reaching a good level of performance. The field of reinforcement learning has massively advanced since then. A more recent work is a *GitHub* repository (mvelax, 2017) that implements the *Wizard* game rules and a Deep Q-learning approach with rule based predictions. We primarily base our work on the game engine of this repository and explore the performance of neural network based trick predictions and Proximal Policy Optimization (Schulman et al., 2017).

We evaluate our agent *meRLin* against rule based approaches, the DQN-agent of the repository (mvelax, 2017) and also against human players. The details of *meRLin* are described in **??**, the training and evaluation specifics are explained in section 3 and the results are presented in **??**.

2. Architecture — Jonas

In the game of *Wizard*, the agent has to solve two separate problems. First, it has to make a prediction and second has to play cards during the individual rounds in order to reach its prediction. Therefore, the agent's architecture is split into two phases. This is visualized in Figure 1. In the prediction phase (Section 2.1), the agent has to make a prediction about the amount of tricks that it will make during the round based on its initial hand cards. This prediction is then propagated to the playing phase where the individual tricks are played. During each trick, the agent has to decide which card he plays. For this decision, the agent has information about the current state of the game (hand cards, trick cards, etc.) and the prediction that was made during the prediction phase. Reinforcement learning is used to optimize the individual decisions in this phase. After the round is finished, the number of tricks that were actually achieved by the playing

[1]TU Berlin, Berlin, Germany. Correspondence to: Vaios <vaios.laschos@tu-berlin.de>.

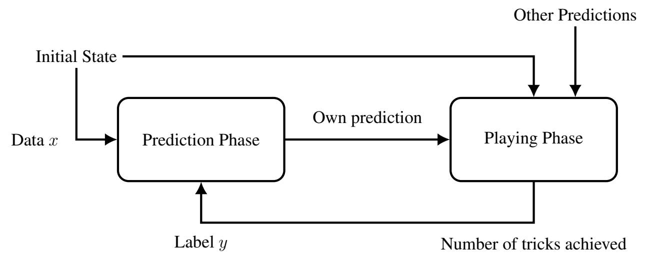

Figure 1. Architecture of the Agent

phase is propagated back to the prediction phase in order to improve future decisions. Therefore, the learning problem in the prediction phase is supervised where the features are the initial cards and the label the amount of tricks that was achieved by the playing phase. We call such a pair a sample of the prediction model.

2.1. Prediction Phase — Jonas

When humans play the game of Wizard, there is a lot of thought put into the prediction process. The exact combination of cards, the round number and the playing style of the opponents has to be taken into account. There is no known algorithm to find the optimal choice in a particular situation. Therefore, we decided to use a neural network for trick predictions. The simplest network would take the cards as an input and output a prediction. There are two major problems with this approach.

low predictions The model has no information about the fact that a higher prediction also leads to a greater score in case of success. In the game of Wizard, it is generally easier to reach lower predictions. Therefore, if the model observes that lower predictions are more often fulfilled by the playing model, there is no reason for the prediction model to make a higher prediction to eventually reach a greater score. To solve this problem, our network does not output a single prediction value. It instead outputs a probability distribution which represents the likelihood that a particular prediction is actually achieved by the playing phase. From this distribution, we then calculate an expected score and output the prediction which maximizes this expected score.

correlated samples The label samples that are shown to the predictor model during training are dependent on the prediction that was given to the playing phase for that sample. Therefore, the labels are correlated with the past predictions of the model. To solve this problem, our network does not only get the initial cards as an input but also receives the prediction that was given to the playing phase for that particular sample.

The final prediction model works in the following way. The neural network takes the initial cards and the prediction (one hot encoded) as an input and outputs a probability distribution over the 16 possible trick values. The initial cards are encoded in a 54 dimensional vector. Each dimension corresponds to one card type. The vector is mostly a one-hot encoding whether our agent has this card type but in the case of wizards and jesters numbers up to 4 can also occur. We calculate the output of the network for every possible prediction. Let $y^{(p)}$ be the output of the network for prediction p. Furthermore, let $s_i^{(p)}$ be the score that is received when p is predicted and i tricks are reached during a round. From these values, we can calculate the expected score for each possible p.

$$\mathbb{E}_p[\text{score}] = \sum_{i=0}^{15} y_i^{(p)} s_i^{(p)}$$

The prediction $p_{max} = arg\,max\,\mathbb{E}_p[\text{score}]$ which results in the highest expected score is then chosen. This computation pipeline is visualized in Figure 2.

The network has 3 layers with 128, 64, 32 neurons and uses batch normalization (Ioffe & Szegedy, 2015). During training, the prediction model uses a batch buffer of size 1000 that is filled during the game and used for training after observing 300 samples.

2.2. Playing Phase — Jasper

Compared to other perfect information games like Chess or Go, Wizard is a game of imperfect information. This makes it harder to apply methods like planning with Monte-Carlo Tree Search (Silver et al., 2016). In the past, games of imperfect information have been solved using Counterfactual Regret Minimization (Brown et al., 2018). Compared to these works, we apply Proximal Policy Optimization (PPO) (Schulman et al., 2017) which is regarded as one of the state of the art methods in model-free RL. We decided on model-free RL as compared to model-based methods because the main benefit of model-based RL is its sample efficiency, which is not of particular importance when considering that our problem can be fully simulated.

Our implementation of PPO uses the TF-Agents library (Guadarrama et al., 2018) which provides most building blocks. The reward is given to the agent every round and is equal to the number of game-points achieved. Compared to a binary reward (win or loss) at the end of the game, this reward is not as sparse and enables faster learning. It it also similar to rewarding the agent for fulfilling its prediction because the number of points achieved at the end of a round is positive if and only if the prediction was fulfilled.

The actor and critic network of the agent consist of 5 fully

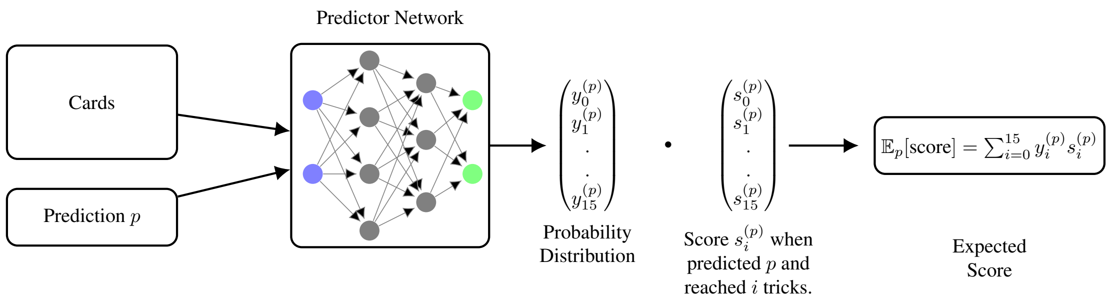
Predictor Network
Cards
Prediction p
$y_0^{(p)}$ $y_1^{(p)}$ . . $y_{15}^{(p)}$
Probability Distribution
$s_0^{(p)}$ $s_1^{(p)}$ . . $s_{15}^{(p)}$
Score $s_i^{(p)}$ when predicted p and reached i tricks.
$\mathbb{E}_p[\text{score}] = \sum_{i=0}^{15} y_i^{(p)} s_i^{(p)}$
Expected Score

Figure 2. Computation of the expected score for a particular trick prediction p during the *Prediction Phase*.

connected hidden layers. The input is a 180 dimensional feature vector which has been taken without modification from (mvelax, 2017). While we tried to modify these features, the results did not improve. The use of manual feature selection simplified the problem because simply using the raw history of played cards would result in more than 3000 dimensions. While this could be solved by incorporating information about the history of the game using an LSTM, experiments showed that this slowed down training significantly.

The following features from (mvelax, 2017) share similar encodings. A set of cards is encoded as a 54 dimensional vector as in section 2.1, i.e. each dimension describes a card type and the value describes the amount of cards of that type in the set. A color of a card is encoded as a 5 dimensional one-hot encoding of the 4 possible colors plus a special 5th value which is set if the card is a wizard or jester which do not have colors.

1. The set of cards the agent has on its hand.
2. The set of cards played in the current trick before the agents turn.
3. The set of cards played in the current round.
4. The trump color for the current round (if there is none, the 5th value is set).
5. The color of the first card in the current trick.
6. The number of won and predicted tricks in the round by all players (2 dimensions for each player meaning 8 dimensions). The values for meRLin are always at a fixed position in this vector independent of the player order.

The output of the actor network is a probability distribution over all 54 different cards. After filtering those cards which correspond to legal moves using an action mask, the cards are then played according to the distribution. We decided to use an action mask since using negative rewards to prevent the agent from playing invalid actions slowed down learning considerably.

3. Methodology

3.1. Training — Jasper

The PPO agent is trained in self-play using the Adam optimizer with a learning rate of 1e-5. We use undiscounted rewards ($\gamma = 1$) and instead reduce the variance using generalized advantage estimation (Schulman et al., 2015) with $\lambda = 0.85$. The agent is trained every 10 games.

Because self-play against the current version is known to lead to overfitting against itself, the agent plays some of its games against old versions. Specifically, 10% of the games are played against only current versions, 40% against two current versions and one old version, 30% against one current version and two old versions and 20% against only old versions. The current version of the agent is saved to a pool of old versions every 100 games from which the old opponents are then randomly selected. See fig. 3 for an illustration of this concept. The pool size is limited to 500 agents after which they are randomly replaced. Opponents are changed randomly every game and the order is shuffled. We decided to use this mix between old and new versions because playing only against old versions would increase the training time by a factor of 4 as the experience of the old opponents can not be used to train the on-policy PPO agent.

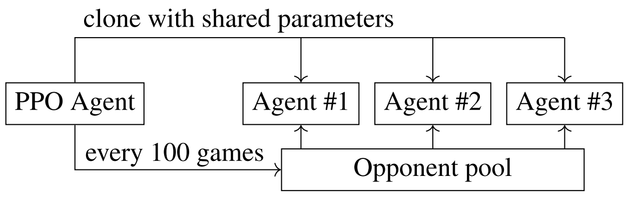

Figure 3. Each opponent is either a clone from the current version or drawn randomly from a pool of old versions.

3.2. Evaluation — Kai

3.2.1. Agents

We evaluate our agents against the following six agents:

1. meRLin with rule based predictor

2. Rule based agent with rule based predictor
3. Rule based agent with predictor from 2.1
4. DQN agent with rule based predictor
5. DQN agent with predictor from 2.1
6. Random agent with average predictor

The rule based agent used in *2.* and *3.* is designed around the following probabilistic based algorithm:

- For each card in the agents hand, calculate the probability of winning the trick against the pool of remaining cards that could be played
- Calculate the desirability of the agent to win the trick based on factors such as the agents prediction and the tricks left to be played.
- Play the card with the winning probability that most closely matches the desirability to win the trick.

The rule based predictor used in *1.* , *2.* and *4.* assigns a probability to win a trick to every card. Where a wizard has the highest probability and a jester hast the lowest probability. The amount of tricks are based on the sum of these probabilities.

The *Deep Q-Network (DQN)* agent used in *4.* and *5.* is an implementation from another project (mvelax, 2017).

Agent *6.* plays random cards and always predicts the average amount of tricks in a round.

3.2.2. Approach

To confront meRLin with several different scenarios, the evaluation is done against all possible combinations of three of the agents defined in 3.2.1. This is totaling to 20 evaluation runs. The models of meRLin are fixed during the evaluation phase. In contrast, the other agents get trained during the evaluation phase, which gives them an unfair advantage over our agent. We chose this approach to show the strength of meRLin, even if the opponents can still learn while playing against it.

Additionally, a graphical user interface (GUI) was developed to evaluate meRLin's performance against human players. Due to time constraints, is was not possible to play enough games to have empirical valid results for this evaluation, but a general trend can be seen. The four team members of this project played 10 full games each. All team members have experience playing Wizard and can be seen as average human players.

4. Results — Kai

4.1. Training

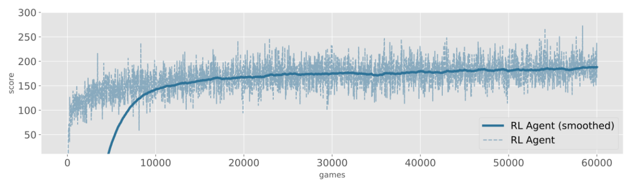

Figure 4. Performance of meRLin during training in self play

Here comes graph of meRLins performance (I am experienceing some issues including the graphics atm)

Here comes graph of meRLins accuracy

The performance of meRLin during training in selfplay can be seen in figure xy and yx. meRLin is converging to 190 points per game and an accuracy of 0.5 of the predictor after 60.000 games of training.

4.2. Evaluation

Here comes the table of results of the evaluation

Table xy shows the results of the evaluation according to 3.2. It can be seen that meRLin outperforms any other agent by at least xx points. The strength of meRLin is even more visible in the win percentage. (numbers about win percentage). (Write a bit more as soon as results are ready)

TBD - what we write about human evaluation

- plays well against amateur human players at the beginning
- human players can adapt to strategy after a few games
- wizards / jesters get played at the end of the round

5. Conclusion & Future Work

We presented a reinforcement learning based approach to the game of Wizard which is able to outperform existing RL and non-RL based approaches including our own rule based agent. The agent, meRLin, is also able to perform well against human players during the first few games. It is therefore shown that a two-phase architecture provides a valid approach for playing this game and that reinforcement learning can be applied to these types of non-markov imperfect information games.

Figure 5. Accuracy of meRLin's predictor during training in self play

	AverageRandomPlayer	OriginalRLAgent_NNPredictor	OriginalRLAgent_RuleBasedPredictor	RuleBasedAgent	RuleBasedAgentPredictor	TFAgentsPPOAgent_NNPredictor	TFAgentsPPOAgent_RuleBasedPredictor
1		120.14 / 0.27	61.61 / 0.12			140.65 / 0.38	97.74 / 0.23
2		131.15 / 0.21			114.61 / 0.18	154.44 / 0.34	131.59 / 0.27
3			13.38 / 0.06	74.14 / 0.17	135.55 / 0.37	136.34 / 0.39	
4			18.29 / 0.06		113.8 / 0.22	158.3 / 0.42	126.79 / 0.3
5	0.68 / 0.05	132.36 / 0.4	36.07 / 0.11			135.27 / 0.44	
6	18.68 / 0.05			90.19 / 0.18	137.76 / 0.33	159.06 / 0.44	
7		147.74 / 0.29		82.11 / 0.14		159.44 / 0.37	107.13 / 0.2
8				82.38 / 0.1	141.83 / 0.2	178.67 / 0.37	164.65 / 0.32
9	14.81 / 0.04				132.02 / 0.26	170.49 / 0.45	127.54 / 0.26
10	33.21 / 0.06			85.6 / 0.16		165.29 / 0.46	137.27 / 0.32
11	9.39 / 0.04	132.31 / 0.29			123.43 / 0.27	149.68 / 0.4	
12	12.99 / 0.03	145.25 / 0.31				162.42 / 0.43	113.45 / 0.23
13		137.01 / 0.35	34.08 / 0.08	83.46 / 0.19		135.44 / 0.38	
14	16.96 / 0.07		36.15 / 0.11	95.73 / 0.27		156.08 / 0.56	
15		127.3 / 0.35	23.1 / 0.08		115.68 / 0.31	98.78 / 0.26	
16	16.11 / 0.05	145.57 / 0.35		91.34 / 0.19		152.53 / 0.41	
17			85.59 / 0.18	86.13 / 0.19		153.48 / 0.43	85.43 / 0.19
18	8.88 / 0.05		41.26 / 0.12			154.99 / 0.54	103.69 / 0.3
19	2.54 / 0.06		20.26 / 0.09		119.54 / 0.37	139.47 / 0.49	
20		146.85 / 0.32		81.66 / 0.14	118.62 / 0.22	139.66 / 0.33	

While we used manual feature selection instead of LSTMs in order reduce the impact of the non-markov property of the environment, we still believe that recurrent architectures would outperform meRLin given a larger computational budget. Additional improvements could possibly be gained using different features, for example by modelling hidden information like opponents cards. Finally, while we only applied PPO in this paper, using other RL algorithms like Soft Actor-Critic (Haarnoja et al., 2018) or model based variants may also improve results.

References

Backhus, J. C., Nonaka, H., Yoshikawa, T., and Sugimoto, M. Application of reinforcement learning to the card game wizard. In *IEEE 2nd Global Conference on Consumer Electronics, GCCE 2013, Tokyo, Japan, 1-4 October 2013*, pp. 329–333. IEEE, 2013. doi: 10.1109/GCCE.2013.6664846. URL https://doi.org/10.1109/GCCE.2013.6664846.

Brown, N., Lerer, A., Gross, S., and Sandholm, T. Deep counterfactual regret minimization. *CoRR*, abs/1811.00164, 2018. URL http://arxiv.org/abs/1811.00164.

Guadarrama, S., Korattikara, A., Ramirez, O., Castro, P., Holly, E., Fishman, S., Wang, K., Gonina, E., Wu, N., Harris, C., Vanhoucke, V., and Brevdo, E. TF-Agents: A library for reinforcement learning in tensorflow. https://github.com/tensorflow/agents, 2018. URL https://github.com/tensorflow/agents. [Online; accessed 25-June-2019].

Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *CoRR*, abs/1801.01290, 2018. URL http://arxiv.org/abs/1801.01290.

Ioffe, S. and Szegedy, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

mvelax. Wizard game simulator to use with reinforcement learning. https://github.com/mvelax/wizard-python, 2017.

Schulman, J., Moritz, P., Levine, S., Jordan, M., and Abbeel, P. High-dimensional continuous control using generalized advantage estimation, 2015.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017. URL http://arxiv.org/abs/1707.06347.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., and Hassabis, D. Mastering the game of go with deep neural networks and tree search. *Nature*, 529:484–503, 2016. URL http://www.nature.com/nature/journal/v529/n7587/full/nature16961.html.

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354, 2017.