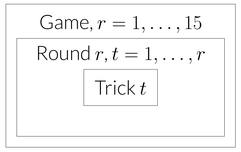
Applying Reinforcement Learning to the game of Wizard

Callum Waters Jonas Dippel Kai Jeggle Til Jasper Ullrich

About the game

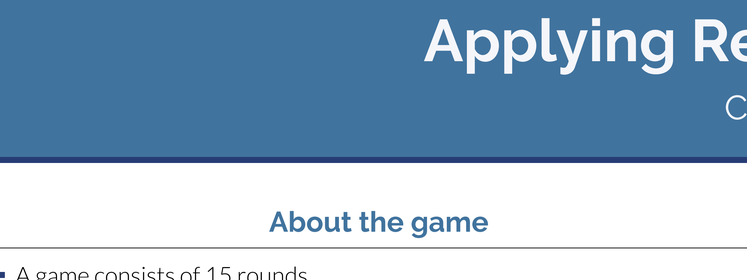
- A game consists of 15 rounds.
- The number of dealt cards and hence the number of tricks increases by one every round.
- In each trick, every player plays one card.
- The strongest card wins the trick.



- In each round, every player makes a prediction on how many tricks they will win.
- Points are gained for predicting correctly and lost for predicting wrongly.
- Fulfilling a higher prediction gains more points whilst a higher difference away from the prediction loses more points.

Architecture

- During the *Playing Phase*, the agent tries to reach the amount of tricks specified beforehand in the *Prediction Phase*.
- The amount of tricks that the agent actually achieved at the end of the round is then passed back to the prediction model as a feedback signal.



Prediction phase

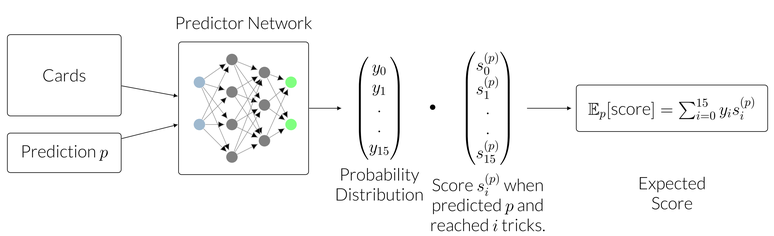


Figure 1. Prediction pipeline to calculate the expected score, if the agent predicts p tricks.

- The *Predictor* computes a round prediction based on the agent's initial cards.
- For every possible prediction p, the expected score $\mathbb{E}_p[score]$ is computed as shown in Figure 1. The prediction that yields the highest expected score $p_{max} = \arg\max_p \mathbb{E}_p[\text{score}]$ is then chosen.

Playing phase

- Our agent uses Proximal Policy Optimization (PPO) [2].
- Both actor and value network consist of 5 fully connected layers.
- We filter invalid actions at the output of the actor network.
- The reward is equal to the number points achieved in the round.

Features

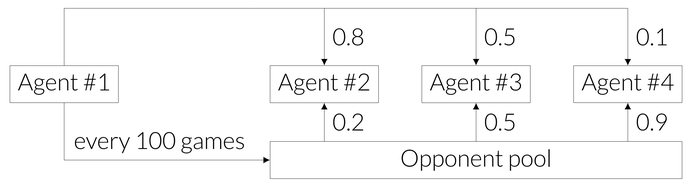
Features from [1]
180 dims
⇒ 175k params

- cards of agent
- #tricks achieved
- cards played during round
- current trick information
- ⋮

- most previous features
- cards played by player #1
- ⋮
- cards played by player #4
- ⋮

Our features
525 dims
⇒ 1.2M params

Training



Evaluation

Agent #1
- our trained RL agent
- fixed parameters

Agent #2
- plays cards at random
- predicts average amount

Agent #3
- rule based prediction
- rule based play

Agent #4
- prediction: NN, play: rule based
- still learning

Rule based agent

The rule based agent used in evaluation 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.

Results

Training

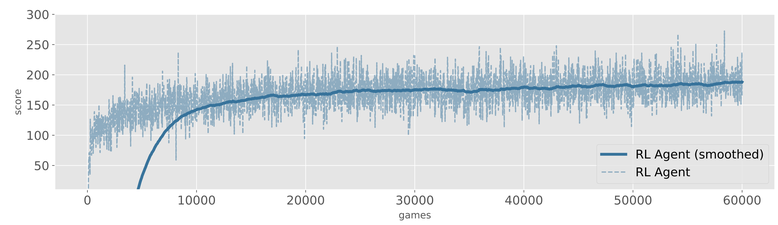


Figure 2. Performance of RL Agent during training in self play

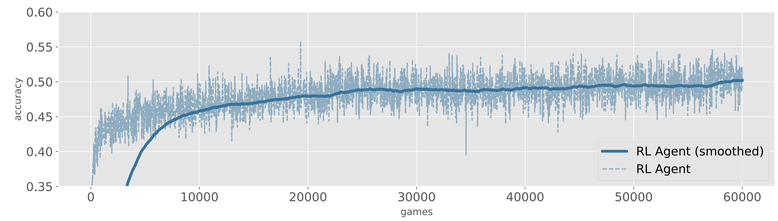


Figure 3. Accuracy of Predictor during training in self play

Evaluation

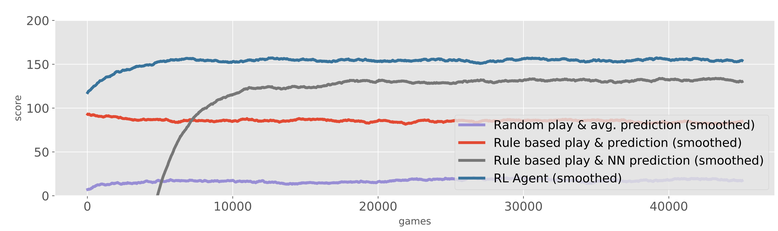


Figure 4. Evaluation of trained RL agent

Problems we encountered

- Invalid actions
- LSTMs not feasible
- Features didn't improve results
- Implementation issues

Future work

- Find Features that improve performance
- Try RNNs (LSTMs) again
- Model the hidden information and use it as features
- Try other algorithms (e.g. SAC)

References

[1] mvelax. *Wizard game simulator to use with reinforcement learning*. `https://github.com/mvelax/wizard-python`. 2017.

[2] John Schulman et al. "Proximal Policy Optimization Algorithms". In: *CoRR* abs/1707.06347 (2017). arXiv: `1707.06347`. URL: `http://arxiv.org/abs/1707.06347`.