Tandem-Chess AI

by group: Felix Japtok 977397 & Jan Kettler 979374

Cognitive Science, Universität Osnabrück

Deep Reinforcement Learning

Dr. Elia Bruni , Leon Schmid, Charlotte Lange

April 14, 2021

Task

As our project, we expanded our own implementation of a chess AI to be able to manage multiple agents with shared parameters who play tandem-chess and communicate with their teammates. We changed our approach from the previous implementation to let our network predict probability distributions for the actions additionally to the outcome of the game, closer resembling AlphaZero.

Tandem-chess is played in teams on two boards simultaneously. The player who plays the white pieces on the first board is teamed up with the one playing the black pieces on the other board. Victory is achieved as a team so once the game on one board ends the results for the teams are fixed. In tandem-chess it is possible to insert pieces that your teammate lost on your own board. There are multiple rule sets for these insertions and for tandem-chess in general. We decided not to restrict the possibilities for the insertions with the exception of occupied squares and not being able to place pawns in the last row of the board which would promote them immediately. Previously promoted pawns will still be inserted as pawns on the other board. We added the new rules for tandem-chess and reworked our network to obtain all the new outputs we need. Because of the new way our actions are chosen now we adapted the methods in the agent and search tree to fit the new requirements. Further we added another termination condition with a maximal number of moves played per game since the games tend to go much longer when pieces can be used multiple times between the boards.

Our new network consists of a value network outputting the predicted value for the outcome of the match, a policy network computing the probabilities for our moves, a message network to create our messages for the agent communication and a small start network which is applied to the input before it is passed to the other networks.

Model and training

Our base model[1] was already designed to learn by playing itself, but for tandem-chess we wanted to have multiple agents that communicate with each other. We decided to let only the agents in the same team communicate since only then their goal is shared. As our main source of inspiration for the design of our agent and search tree we used a github repository[2] and used the buffer from the re-ally framework[3] provided by Charlie for this course. The paper "Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm" provided the theoretical orientation for us (*Silver et al., 2017*).

Our method for the communication was Reinforced Inter-Agent Learning (RIAL), meaning that the agents do not get feedback for the sent message but share their parameters and learn to interpret the messages that way (*Foerster et al., 2016*). We used a buffer even though it is not recommended for RIAL to accelerate the training time with limited success. We experimented a bit with training the message head, the part of the network creating the message, but did not achieve noticeable improvements in performance. This could be due to us not finding a suitable target to optimize with. Even without optimizing the message head directly, since the message is in every input to our network, it influences the policy and value head's outputs. For that reason the agents learn how to interpret the message depending on the input instead of learning how to send better messages. The message is passed directly to the teammate who incorporates it in his network's input.

We adapted our network structure from our previous approach to our new requirements. So we divided the network into multiple parts to compute our different outputs: The value network, message network, policy network and start network. This division enables us to train different parts of the network with different loss functions and gradients. The value network is optimized with the mean squared error loss, `loss = square(y_true - y_pred)`, of it's prediction and the real outcome of the game. The policy network uses Kullback-Leibler divergence, `loss = y_true * log(y_true / y_pred),` with the targets being the actual search-probability returned by our tree-search. Both parts compute their own gradients respectively and apply them only to the corresponding weights. Optimization of the start network uses all gradients since it influences all outputs.

The reason we used an approach closer to AlphaZero than before is that we hoped to achieve an improvement in performance by letting the network learn how to predict legal moves. This was accomplished by finding all legal moves in the input state and comparing them to the network's predicted move distributions. Only the legal moves for the input state keep their predicted value while all illegal moves get assigned a probability of 0 enabling the network to learn to predict only legal moves. With this additional heuristic we think we can improve our performance.

---

[1] https://github.com/Bluhster/TensorFlow_final_project
[2] https://github.com/AppliedDataSciencePartners/DeepReinforcementLearning
[3] https://github.com/geronimocharlie/ReAllY/tree/master/really/really

Design

In the following we will be going more into detail about the choices we made concerning the structure of our agents and environment.

When we started to implement tandem-chess we made the choice to simulate play synchronously instead of asynchronously. This means that we fixed the move-order of the agents, not allowing the two boards to progress at different speeds. This has multiple advantages, but also some disadvantages. We did not have to include the possibility of an agent waiting for the other board-state to progress before moving. This also meant that we did not have to include a time limit and the possibility to lose on time, which could have sped up training, but would also have been more difficult to implement and handle the logic for. Additionally fixing the move order made the communication of the agents simpler to handle. This approach differs from the way tandem-chess is generally played but we chose it due to its simplicity.

Concerning the way the agent samples actions with Monte Carlo tree search we included the output of the policy network as a heuristic to guide the search. To choose which edge we expand we act greedily on a variable information gain, which in turn is based on the edge's value, the times we visited the edge already and the predicted move-probability. This should in theory be a great way of guiding the expansion of moves to states with high estimated values. This is the case because the policy network is trained on the search-probability of the tree, which is in turn partly based on it's estimated value. The search-probability is the number of times we visited a state in comparison to the number of times we visited other states. Additionally the hyperparameter tau of the agents controls the exploration at the start of the game for a fixed amount of moves.

The input to our network is of shape 8x8x20, in which we included, the one-hot encoded boardstate categorically representing the occupants of squares, whose turn it is (black/white), which pieces are insertable and the received message of the teammate.

To train the policy network we had to bring its output in a form that could predict all possible moves. For tandem-chess this is an 8x8x80 matrix, where the first two dimensions are coordinates from where a piece could be "picked up" and the last dimension of 80 encodes all possible actions that could be performed by any piece. The 80 moves consist of: 56 queen-moves (7 steps in any direction * 8 possible directions), 8 knight-moves, 9 pawn underpromotions, 5 insertable pieces, and lastly two castling moves. To make a comparison of legal moves and predicted moves possible we also had to change the way moves are represented and calculated in our board.

Results

Evaluating the behavior in chess is very complex let alone in tandem-chess and there is no distinct measure to compare chess AIs other than letting them compete against each other. This was not an option for us because playing even a single game would include hours of transforming the different board states during the game into correct inputs for our network. For this reason our evaluation will be based on the runtime, losses, observed behavior during the matches and the AI playing

against older, less trained versions of itself.

As for the needed time for playing and training, the playing was immensely more time consuming than the training. The parameter, num_steps, to limit the number of expansions per move in the search tree was most influential on the time consumption during playing.

Regarding the losses, we did not observe unusual behavior in the system. All losses decrease over time in reasonable speed, not indicating significant over- or underfitting. But since our targets are obtained through self-play we do not have fixed targets and the loss can show unstable behavior.

To evaluate the behavior we created a method to assign different networks to the different teams to test older less trained versions of our system against newer ones. On average the new versions won more games signaling an improvement in performance after training. From our point of view it is difficult to evaluate the behavior given our very limited experience with tandem-chess, since general strategies for chess differ heavily from tandem-chess. For example taking enemy pieces can increase the risk of your teammate getting checkmated. But subjectively we observed what seemed to be an improvement in strategies for longer trained versions of the system.

References

Foerster, J. N., Assael, Y. M., De Freitas, N., & Whiteson, S. (2016). Learning to communicate with deep multi-agent reinforcement learning. *arXiv preprint arXiv:1605.06676*.

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., ... & Hassabis, D. (2017). Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*.