Training an agent to play the game of Hex using reinforcement learning

Nicolas Flake
ai23m058

Georg Johannes Talasz
ai23m028

Marco Ross
ai23m022

***Abstract*—This paper investigates the application of reinforcement learning (RL) to train an agent for the game of Hex. It focuses on different RL algorithms, training strategies and neural network architectures. The main algorithms evaluated were Deep Q-learning (DeepQ), AlphaZero, and Soft Actor Critic (SAC). Initial experiments on smaller board sizes (3×3 and 5×5) helped in selecting the most promising algorithm, a fitting training strategy and the best architecture to utilize for training agents on a larger and more complex 7×7 board size. Results demonstrated that Deep Q-learning combined with convolutional neural networks achieved the highest overall performance and generalized across different board sizes.**

***Index Terms*—Reinforcement Learning, Hex, DeepQ Learning**

I. Introduction

The objective of this paper is to utilize reinforcement learning to train an agent playing the game of Hex, as part of the reinforcement learning course at FH Technikum Wien, led by Dr. Sharwin Rezagholi. Hex is a strategic board game where two players aim to connect opposite sides of a hexagonal grid. Although the grid size can vary, a fixed size of 7×7 was defined for this study. This task presents unique challenges for RL due to the game's complexity and the necessity for effective generalization across different board sizes.

Board games have long been of interest for advancements in the fields of reinforcement learning. Notable milestones include DeepMind's AlphaGo [1] for the game of Go and various RL algorithms applied to Chess and Shogi [2]. Before that, Deep Q-Learning was developed to train agents to play Atari computer games [3]. For the game of Hex, research has explored multiple approaches, including convolutional neural networks (CNNs) [4], Deep Q-learning strategies [5], and the application of AlphaZero [6]. These studies provide a foundation for exploring how different RL algorithms can be adapted and optimized for Hex and were the base for our different approaches in training a successful agent.

In our experiments, we tested different algorithms, including Proximal Policy Optimization (PPO) [7], Deep Q-learning [3], AlphaZero [2], and Soft Actor Critic for discrete action spaces [8] with different neural network architectures such as transformer, convolution and Feed-Forward networks. Initially, we started with smaller board sizes of 3×3 and 5×5 to evaluate the effectiveness of each algorithm and perform hyper-parameter tuning. This preliminary phase allowed us to identify the most promising algorithms and training strategies. We then proceeded to train these selected algorithms on the 7×7 board size. Finally, we set up an arena to pit the trained agents against each other. This allowed us to identify not only the agents with the best strategies and overall performance but also to conclude experiments involving competing agents trained on different field sizes.

II. Methods

During the exercise, we implemented and experimented with different training algorithms and model architectures to train various agents. Finally, we let the best agents from each training algorithm and model combination play against each other in a tournament to select the best agent for submission. Additionally, we conducted experiments regarding the capability of some models to play on board sizes different than the one they were trained on.

A. Training algorithms

We were provided Python code with an implementation of the game Hex as an environment to train agents using reinforcement learning techniques. We evaluated four different training algorithms:

1) Proximal Policy Optimization (PPO): In a first attempt, we implemented a wrapper for the given source code to make the code compatible to the Gymnasium framework[1]. This way we could experiment with readily implemented RL algorithms implemented in the Stable Baselines collection[2]. To establish a baseline, we trained an agent using PPO as provided by the framework. The algorithm converged and was quickly able to compete against a randomly acting opponent. These motivational results gave us a first impression of input encodings and hyper-parameters that might work. However, the task of this course was to implement the algorithms ourselves, therefore we abandoned this approach quickly and went on to implementing algorithms ourselves.

2) Alpha Zero: We implemented the AlphaZero algorithm [2] taking an universal alphaZero Github repository [9] as base and adjusted it to the hex game rules. As Network we used a shallow convolutional neural network with a 2-layer architecture. Training began on a 3×3 board, with each iteration consisting of 40 self-play games generating a Q-value learning buffer via Monte Carlo Tree Search (20 simulations per move), followed by 10 network training epochs. This process was repeated for 20 iterations.

[1]https://gymnasium.farama.org/
[2]https://stable-baselines.readthedocs.io/

3) Soft Actor Critic (SAC) for discrete action spaces: We implemented the Soft Actor Critic algorithm for discrete action spaces [8] and tested it with the Feed-Forward neural network architecture shown in Figure 1.

We cloned the source code from the Github repository[3] referenced in the published paper and experimented with hyper-parameters by varying the discount range between 0.9 and 0.99, the learning rate for the actor and the critic networks between 0.01 and 0.0001, as well as the target net update rate between 0.005 and 0.01. Furthermore, we tried replacing the mean squared error loss function used for optimizing the critic network with a binary cross-entropy loss. The agent was trained for 2000 episodes against a random opponent using numerous combinations of the aforementioned hyper-parameter settings on a 7×7 Hex board.

4) Deep Q-Learning: Deep Q-Learning is a reinforcement learning algorithm that extends Q-Learning by using deep neural networks to approximate the Q-value function. This approach allows the agent to handle high-dimensional state spaces, making it suitable for complex environments like board games. The training process for our Deep Q-Learning agent begins with the agent playing against an opponent making random moves. This initial phase allows the agent to explore the state space without any prior knowledge, facilitating the learning of basic strategies and responses.

We stopped the training of a model once it achieved a win rate of 100% in 50 consecutive training episodes or the maximum number of episodes for this generation was reached. After an agent had completed its training, we saved its model and used it to train subsequent agents. We repeated this process for 50 generations for each model architecture.

During training, the currently trained model competed against either a randomly selected previous agent or the random-move opponent. The model chose its actions using an ϵ-greedy strategy, where a random action was taken with a probability of ϵ instead of the action with the highest predicted Q-value (greedy action). We started with a high initial ϵ_0 value to facilitate exploration in the early training phase of each new agent. The schedule for reducing ϵ to its final value ϵ_f with a decay of β was defined as $\epsilon_i = \epsilon_f + (\epsilon_0 - \epsilon_f) \cdot \exp(-\frac{i}{\beta})$. Opponents are switched with every step (move) to maximize the variability in the replay buffer. For the selection process, recently trained agents are more likely to be chosen, based on the assumption that these agents play at a higher skill level and can thus provide a more challenging and informative learning experience for the training agent. Opponents were ranked by their generation and the probability of selecting an opponent was highest for the latest generation and decreased by a factor of 0.9 with each generation. The hyper-parameters used during training are summarized in Table I.

To simulate different starting conditions, the board was initialized either empty or with an opponent's stone placed randomly on the board. This variation ensures that the agent can adapt to different game scenarios, enhancing its overall robustness and strategic flexibility. To facilitate learning of more effective strategies (e.g. to reward winning within fewer moves) a relatively low discount factor of 0.9 was used. The combination of these strategies aims to progressively build more competent agents by leveraging past experiences and ensuring diverse training scenarios.

We experimented with three different kinds of neural networks for our agents:

Feed-forward A simple feed-forward network with a width of 128 and 2 hidden layers

Convolutional A convolutional network using three convolutional layers. Convolutional networks are well-suited for spatial data and can effectively capture the local dependencies on the game board.

Transformer A transformer network using four transformer encoder layers.

The architectures are described in more detail in section II-C.

TABLE I
HYPER-PARAMETERS SETTINGS APPLIED IN THE DEEP Q-LEARNING ALGORITHM

Parameter	Value
Replay buffer size	10000
Batch size	32
Maximum episodes for generation G	$2000 + 200G$
Discount factor (γ)	0.9
Learning rate (λ)	0.001
Target net update rate (τ)	0.005
Initial ϵ_0	0.9
Final ϵ_f (after maximum epsiodes)	0.05
ϵ-Decay (β)	1000

B. Data representation

1) State representation: We evaluated three different model architectures. To use a uniform training procedure for all model architectures, the state and action spaces were both encoded as one-dimensional vectors by simply reshaping the two-dimensional board of size $s \times s$ to a vector of length s^2. The vector could contain three values at each given position: **1** - white stone, **-1** - black stone, **0** - empty field. Empty fields are valid actions for an agent.

2) Cube Coordinates: On the hexagonal board, each field is connected to up to six neighbours along three axes. To provide a-priori knowledge about the geometry and neighbourhood connectivity on hex-boards and to reduce the number of learnable parameters, a three dimensional coordinate system with so-called cube coordinates was concatenated to the input vector for some of the models. As presented in [10], the x coordinate was given as a vector of evenly spaced numbers from $[-1, 1]$ along the axis of the white player. The y coordinate was given as a vector of evenly spaced numbers from $[-1, 1]$ along the axis of the black player. The third positional dimension is specified as $z_i = -x_i - y_i$.

[3]https://github.com/p-christ/Deep-Reinforcement-Learning-Algorithms-with-PyTorch

C. Model architectures

We tested one feed-forward neural network, two variations of a convolutional model, and two variations of a transformer based architecture.

1) Feed-Forward Model: The feed-forward neural network uses fully connected linear layers and rectified linear unit (ReLU) activation functions. Due to the connection of all inputs to all units, network parameters are unique for each input (no weights are shared) and no positional encoding is necessary. Therefore only the $s^2 \times 1$ shaped input vector representing the current state of the Hex board was used as input to the network. We used two hidden layers with a width of 256 units. Figure 1 illustrates the architecture. Outputs are multiplied with a validity mask that is 1 for empty fields and 0 otherwise.

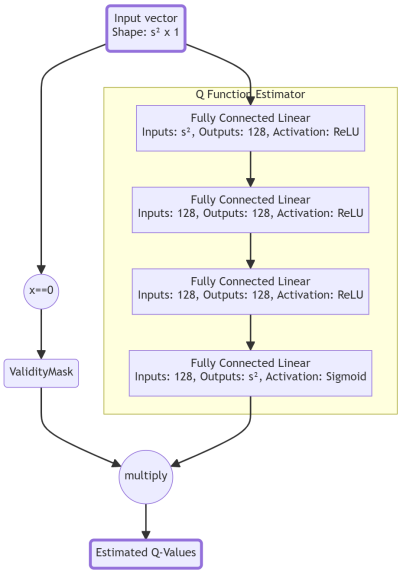

Fig. 1. Feed-Forward Neural Network Architecture

2) Convolutional Model: The convolutional model exploits shared weights by means of kernels that move across the input data to extract local patterns. In order to do that, the model first transforms the input vector of length s^2 back into its original two-dimensional shape $s \times s$. Estimated Q-values also have the same shape $s \times s$ and have to be converted back to the one-dimensional action space before returning the results to the training environment.

We used an input layer with a kernel size of 3×3 to feed 3 hidden layers with kernel shapes of 5×5 and 128 channels. Finally, another convolutional layer with a kernel size of 3×3 followed by a sigmoid activation function reduces the 128 feature maps to one estimated Q-value for each field. Skip connections were used to facilitate effective gradient flow and reduce training time. Outputs are multiplied with a validity mask that is 1 for empty fields and 0 otherwise.

We trained two variations of the model:

Convolutional Only the board was used as input. Therefore, convolutional layers could only learn local patterns independent of their absolute position on the board.

Convolutional+ Cube coordinates as described in section II-B2 were concatenated to the input vector to provide absolute position information of fields to the model. This results in an input shape of $s^2 \times 4$

Figure 2 illustrates the model architecture including the optional concatenation of cube coordinates to the input state vector.

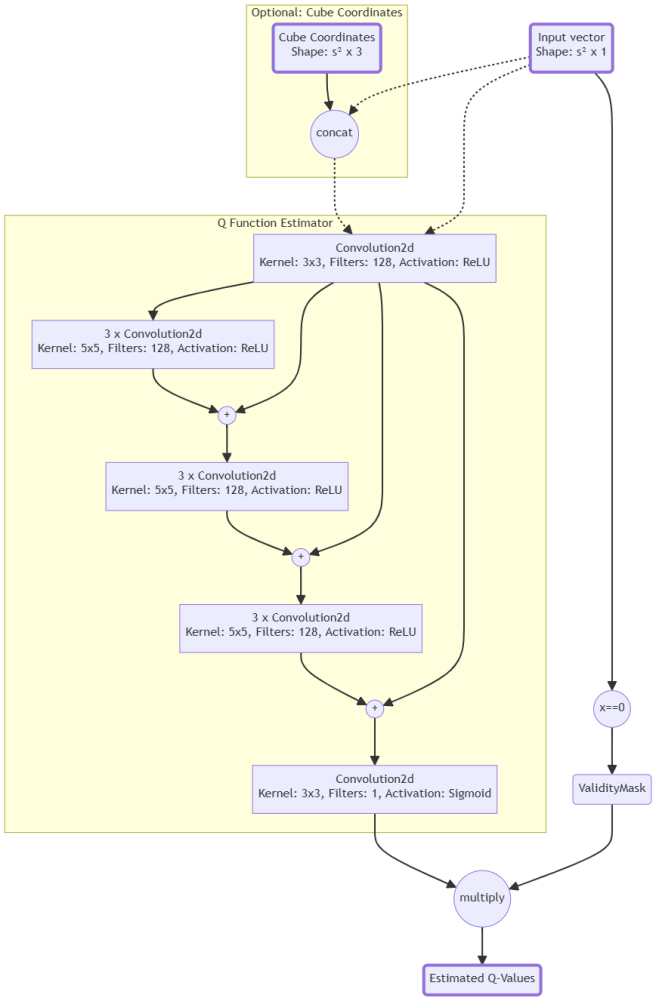

Fig. 2. Convolutional Neural Network Architecture. Dotted paths indicate alternatives regarding the positional encoding.

3) Transformer Model: The transformer model is using a standard transformer encoder architecture as used in the self-attention blocks in [11]. We used four multi-head attention layers with a feature dimensionality of 128 and 8 attention heads. For the feed-forward network hidden inside the transformer encoder layers, 512 features were used. We used learned embeddings to encode the state vector. Figure 3 presents an overview over the architecture of the transformer model.

For the model to be able to learn spatial patterns, a positional encoding was required. Two model variations were tested:

Transformer The model referred to as "Transformer" uses cube coordinates concatenated to the state vector as described in II-B2 to provide a-priori knowledge of the hexagonal geometry. Thus, the input to this model has shape $s^2 \times 4$.

Transformer2 This model uses learned positional embeddings. This approach is easier to implement but has the disadvantage that a-priori knowledge about the neighbourhood relationships between board fields is not provided and needs to be learned by the model, instead.

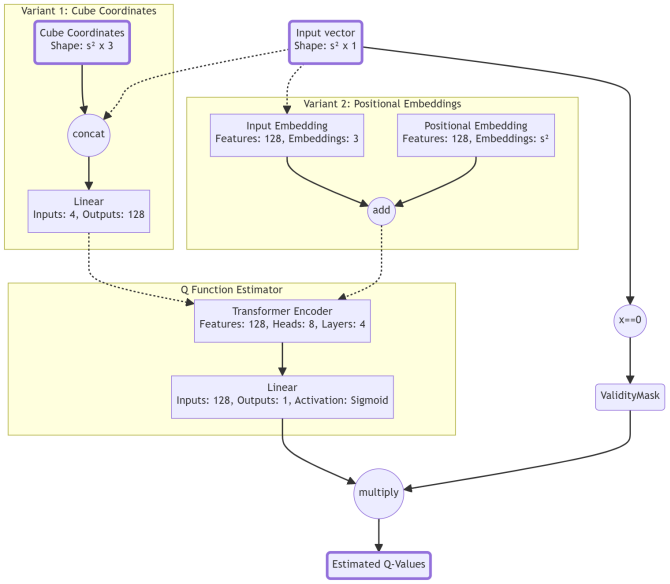

Fig. 3. Transformer Model Architecture. Dotted paths indicate alternative paths between the two model variations. Variant 1 illustrates the positional encoding using cube coordinates. Variant 2 shows the positional encoding using learned embeddings.

D. Agent Selection

To determine the best model, we decided to play a tournament between all trained agents. In Hex, especially on small boards, the beginner has an unfair advantage. Therefore, we decided that a match between two agents shall always consist of two games where each opponent may start once. Repeated matches between agents would be useless, since during the tournament all agents would always play greedy actions and, therefore, act deterministically. Due to the relatively low number of 250 trained agents, it was feasible to simply have every pair of agents play a match and sum up all acquired points for each agent. The agent with the highest number of points would win the tournament.

E. Board size generalization

The transformer model and the convolutional model are designed such that they can in principle work on different board sizes with the same model parameters. Since in previously described experiments, the convolutional model learned the best strategies, we conducted further experiments with that architecture.

The training strategy described in section II-A4 was applied to obtain 50 agents trained on a board size of 5×5 and 50 more agents trained on a board size of 7×7. The two groups of players were enrolled in a three-phase tournament, where one group consisted of agents trained on a 5×5 board, while the other group consisted of agents trained on a 7×7 board. For this tournament, a match between two agents was defined as playing two games of Hex where each opponent started with the first move once. One point was awarded for each victory. The three phases of the tournament are described below:

1) Phase I: Agent selection: The first phase served as a "qualification" phase for the actual tournament. From the 50 available agents within each group, only 10 were selected for competing in the subsequent phases of the tournament. Determine the best players within each group, each model was matched against each other model from the same group. Individual model scores served as the basis for qualification of the 10 best agents from each group to the next phase of the tournament.

2) Play against random actor: The goal of the second phase was to determine, whether agents trained on one board size were able to apply their strategies on a different board size at all. Therefore, each agent had to compete in 20 matches against a randomly acting opponent on both board sizes. The final win rates for one group of agents was determined by summing the scores obtained by all agents trained on the same board size.

3) Play against agent from different group: Finally, to determine if the strategies could also be competitive against a non-random acting opponent, every agent from one group had to compete in a match against every agent from the other group on both board sizes. The primary goal of this phase was to determine the advantage of playing on the same board size that the agent was trained on. The secondary objective was to determine if there was a difference in generalization capabilities, depending on whether the played board was smaller or larger than the board the agent was trained on. To determine the overall win rates for each group, scores obtained during matches were summed over all agents within each group, stratified by board size.

III. Results

A. Evaluation of training algorithms

We evaluated three different training algorithms for reinforcement learning. Given our resource constraints, we achieved the best results in training an agent to play the game of Hex using DeepQ learning.

1) Alpha Zero: The training procedure resulted in a continuously decreasing loss and ultimately achieved an 85% win rate against a random player, indicating learning progress. However, the training time for the 3×3 board was significant, yielding only moderate outcomes. Attempts to train on a 7×7 board revealed that the required code restructuring for better GPU parallelization and the extensive training time exceeded our project's timeframe. Therefore, we decided to

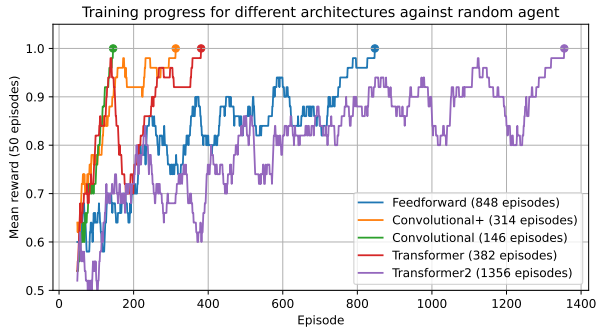

Fig. 4. Training progress of different model architectures against random actor

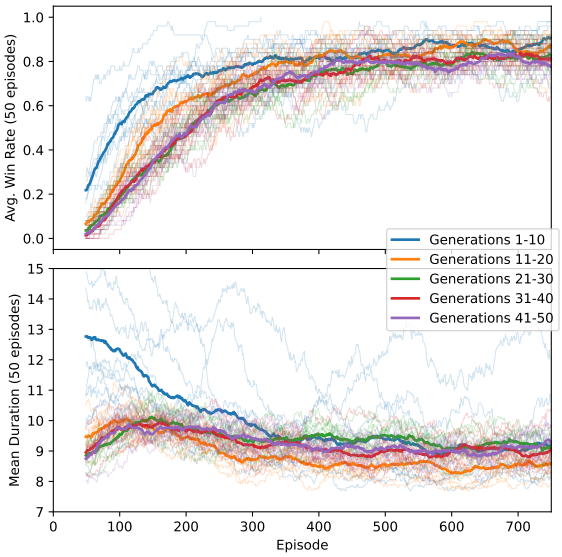

Fig. 5. Training progress of different model generations for the Convolutional+ architecture. The top panel shows average win rates over the previous 50 training episodes. The bottom panel shows the mean durations in steps the model had to play until a game was over. Thin lines represent the training of individual models, thick lines represent the average across a group of 10 generations.

focus on training our agents with DeepQ learning which delivered promising results with relatively moderate resource requirements.

2) DeepQ Learning: As explained in section II-A4, agents of different architectures were trained in an iterative process. Once a model reached a win rate of 100% against its opponents in the last 50 training episodes, its parameters were saved, and training continued including these saved parameters as a new opponent during training.

At the very beginning of the training process, agents had to compete against a randomly acting agent. Against a random opponent, winning should be easy, even by means of trivial strategies (e.g. fill a row from left to right). Figure 4 illustrates the speed of convergence for the five analyzed model architectures. The convolutional model converged the fastest and managed to beat the random actor at a win rate of 100% over 50 successive episodes after just 146 episodes while the transformer model using learned positional embeddings required 1356 episodes to acquire a winning strategy.

Figure 5 illustrates the training results for the convolutional model using positional encoding for different generations of the model. Models were grouped into bins of 10 generations. The first 10 generations were trained against relatively week opponents, while later groups had to train against much stronger opponents. The figure shows that win rates increased faster for the earlier generations of the model. It also shows that games took longer during training of the earlier model generations. For stronger models, game durations increased during the initial phase of the training and started to decrease again once the win rates of the model reached a certain point.

3) Soft Actor Critic for discrete action spaces: After performing a grid search with a large number of hyper-parameter settings, the best agents trained with the SAC algorithm did not exceed a 70% win rate against a randomly acting opponent. Therefore, we assumed that there must be either a mistake in the cloned source code or we just had not found a hyper-parameter setting suitable for training the agent to play a game of Hex. Ultimately, we decided to focus on training our agents with Deep Q-Learning which delivered promising results with relatively moderate resource requirements.

B. Agent selection

Given our limited success with other training algorithms, only models trained with Deep Q-Learning were considered for submission. All architectures described in section II-C were trained for 50 generations following the procedure described in section II-A4.

Figure 6 illustrates the results of the internal tournament where all trained models competed against each other. The top panel shows the number of wins for each model when only competing against models of the same architecture. The bottom panel shows overall tournament results with the number of wins when competing against all other models. The convolutional models clearly outperform the other architectures. In total, 5 architectures were commpared: A simple feed-forward neural network model, a convolutional architecture with (Convolutional+) and without (Convolutional) positional encoding, as well as a transformer based architecture with fixed positional encoding (Transformer) and learned positional embeddings (Transformer2).
All architectures show a similar and relatively monotonous increase of model performance with increasing model generation when compared to models of the same architecture. Overall, the two convolutional architectures achieve higher scores than the others. Generation 40 of the convolutional model with positional encoding achieved the highest score with 475 wins versus all other agents. This corresponds to a win rate of 93.9%.

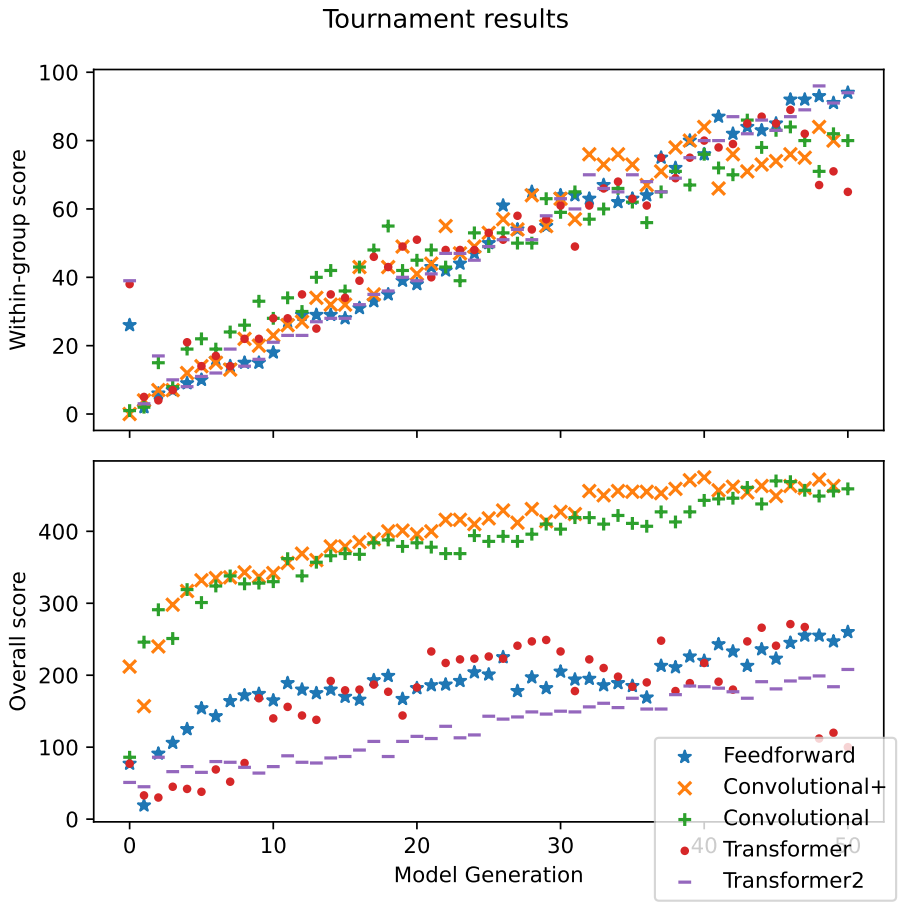

Fig. 6. Performance comparison of model generations and architectures.

C. Board size generalization

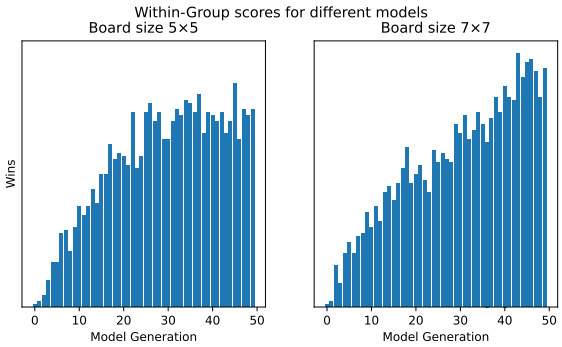

Fig. 7. Within-group results for agent selection

During the first phase of the board size generalization experiments, agents trained on the same board sizes competed against each other to select the 10 best models for the next stages of this experiment. Figure 7 illustrates the results of the qualification phase for the tournaments. While the scores for agents trained on a 5×5 seem to plateau aufter approximately 20 generations, the win rates for agents trained on 7×7 boards continue to grow until the last trained generation.

TABLE II
WIN RATES OF AGENTS PLAYING ON BOARD SIZES DIFFERENT THAN THEY WERE TRAINED ON. THE RANDOM ACTOR OPPONENT ALWAYS MAKES A RANDOM VALID MOVE. 7×7 AGENT AND 5×5 AGENT STANDS FOR HEX AGENTS TRAINED ON BOARD SIZES OF 7×7 AND 5×5, RESPECTIVELY.

Trained Board	Played Board	Opponent	Win Rate
5×5	5×5	Random Actor	100.0%
7×7	5×5	Random Actor	100.0%
5×5	7×7	Random Actor	89.3%
7×7	7×7	Random Actor	91.3%
5×5	7×7	7×7 Agent	2.5%
7×7	5×5	5×5 Agent	12.0%

Win rates obtained during phases 2 and 3 of the tournament are summarized in Table II. For each board size of 5×5 and 7×7, 10 trained agents using the convolutional neural network described in section II-C2 were selected. Each agent played a randomly acting opponent 20 times on each board size. On the 5×5 board, models trained on the 5×5 board and models trained on the 7×7 board always beat the randomly acting opponent. On the larger 7×7 board, both agent classes achieved high win rates around 90%.

In the final stage of the experiment, each of the 10 models from each class played against each of the 10 models from the other class on each board size. Models trained on a 5×5 board size were able to beat models trained on a 7×7 board in only 2.5% of the cases when playing on a board of size 7×7. Models trained on a 7×7 board size were able to beat their opponents on a 5×5 board in 23% of the cases.

IV. Discussion

We evaluated different training algorithms and model architectures. In our experiments, the Deep Q-Learning algorithm delivered good results with reasonable effort. Therefore we focused our experiments on agents trained with this algorithm. The almost monotonously increasing performance with each generation across all tested model architectures confirms the validity of the applied training scheme.

Convolutional models were able to learn patterns and strategies that were superior to the tested feed-forward and transformer architectures. While the allegedly best algorithm MOHEX 2.0 [12] is based on Monta Carlo Tree Search, some competitive algorithms use convolutional neural network as well [5], [13], [14]. We were disappointed with the low performance of the transformer model. Based on the quicker learning process right away and the much better final performance, we hypothesize that the focus on local neighbourhoods intrinsic to the convolutional neural network layers can provide an advantage when learning board games. Providing absolute positional information in the form of cube coordinates improved results for the convolutional neural network architecture as well as the transformer model.

In our experiments, convolutional models were in principle able to generalize from smaller to larger board sizes and vice versa, beating randomly acting opponents at very high rates in both cases. Convolutional models can in principle use the same parameters to process different input field sizes. However, to our knowledge, published models were always trained and evaluated on the same board sizes. For example 7×7, 9×9, and 11×11 in [4] or 13×13 in [5]. A tournament between agents trained on different board sizes resulted in higher win rates when models trained on a larger board were playing on smaller boards than the other way around. This suggests that generalization from larger to smaller board sizes might work better than applying the strategies learned from a smaller board to play on a larger board. Experiments with additional board sizes would be needed to substantiate this hypothesis. However, the results also show that the best performance is achieved by models playing on the same board size they were trained on.

Due to resource constraints, the evaluation of the Soft Actor Critic and the Alpha Zero training algorithms had to be stopped without achieving satisfying results. We hypothesize that these algorithms would have ultimately converged to viable solutions, given enough resources and further hyper-parameter tuning. Thus, all further analyses performed in this exercise were limited to agents trained with the DeepQ algorithm. Furthermore, the comparison of different model architectures was limited to one fixed model per architecture. Due to resource constraints, we did not experiment with model hyper-parameters such as the number of layers or the layer widths.

V. Conclusion

There are excellent frameworks available providing readily-implemented state-of-the-art RL algorithms. Given a suitable environment, these algorithms can be tested relatively easy. Hyper-parameter tuning for some RL algorithms can be difficult. We hypothesize that our experiments with Alpha Zero and SAC mainly failed because we could not find a working set of hyper-parameters given our resource and time constraints. Nevertheless, we were able to achieve good results with Deep Q-Learning and conducted secondary analyses regarding the capability of convolutional models to play on different board size without the need for re-training.

Future work could include experiments with larger boards and of course should include attempts to successfully train an agent using state of the art algorithms besides DeepQ learning. Additional experiments regarding the model architecture could reveal more insight into why the transformer model did not perform as expected. Regarding the generalization on different board sizes, experiments with more and larger board sizes might be interesting as well. Of course, an evaluation of the model comparing it to a well-established baseline like MOHEX 2.0 would be interesting as well.

References

[1] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, *et al.*, "Mastering the game of go with deep neural networks and tree search," *nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[2] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, *et al.*, "Mastering chess and shogi by self-play with a general reinforcement learning algorithm," *arXiv preprint arXiv:1712.01815*, 2017.

[3] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[4] K. Takada, H. Iizuka, and M. Yamamoto, "Reinforcement learning for creating evaluation function using convolutional neural network in hex," in *2017 Conference on Technologies and Applications of Artificial Intelligence (TAAI)*, pp. 196–201, 2017.

[5] K. Young, R. Hayward, and G. Vasan, "Neurohex: A deep q-learning hex agent," *CoRR*, vol. abs/1604.07097, 2016.

[6] C. Lovering, J. Z. Forde, G. Konidaris, E. Pavlick, and M. L. Littman, "Evaluation beyond task performance: Analyzing concepts in alphazero in hex," 2022.

[7] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[8] P. Christodoulou, "Soft actor-critic for discrete action settings," *arXiv preprint arXiv:1910.07207*, 2019.

[9] S. Nair *et al.*, "Alphazero general (any game, any framework)." https://github.com/suragnair/alpha-zero-general, 2019. Accessed: 2024-06-18.

[10] J. K. (https://math.stackexchange.com/users/441008/joshua kidd), "Hexagon grid coordinate system." Mathematics Stack Exchange. URL:https://math.stackexchange.com/q/2643016 (version: 2018-02-09).

[11] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[12] S.-C. Huang, B. Arneson, R. B. Hayward, M. Müller, and J. Pawlewicz, "Mohex 2.0: a pattern-based mcts hex player," in *International Conference on Computers and Games*, pp. 60–71, Springer, 2013.

[13] C. Gao, R. Hayward, and M. Müller, "Move prediction using deep convolutional neural networks in hex," *IEEE Transactions on Games*, vol. 10, no. 4, pp. 336–343, 2018.

[14] M. Lu and X. L. Li, "Deep reinforcement learning policy in hex game system," in *Chinese Control And Decision Conference (CCDC)*, IEEE, 2018.