Adapting AlphaZero to Imperfect Information

Kobe Knowles
The University of Auckland
kkno604@aucklanduni.ac.nz

Nathan Young
The University of Auckland
nyou956@aucklanduni.ac.nz

Ken Liu
The University of Auckland
kliu451@aucklanduni.ac.nz

Bhargava Gowda
The University of Auckland
bgow529@aucklanduni.ac.nz

William Holmes
The University of Auckland
whol106@aucklanduni.ac.nz

***Abstract*—Recent progress in applying reinforcement learning to game-playing has focused on games with perfect information (e.g. Go, chess). The application of these techniques to games with imperfect information has focused almost entirely upon poker, which involves relatively little hidden information (as important as that information may be). We apply progress made in learning to play information-perfect board games to information-imperfect variants of common games to see how well these techniques cope with uncertainty and indeterminacy.**

***Index Terms*—deep learning, reinforcement learning, convolutional neural networks**

I. Introduction

In recent years, much progress has been made in game-playing artificial intelligence. DeepMind's AlphaGo beat experts' predictions by 10 years to beat human champion Lee Sedol in Go [1], with this system later being generalised to play chess, shōgi and Atari games at similarly high levels [2] [3]. However, these techniques have not been generalised to other tasks, partially due to their focus on *information-perfect* games - that is, games in which all players are aware of the full game state. Such games provide what Hogarth, Lejarraga and Soyer call 'kind' learning environments, with immediate and unambiguous feedback. [4] Real-world learning tasks, by contrast, are commonly more 'wicked', rife with uncertainty and ambiguity. For these systems are to be made more generally intelligent, they must be adapted to cope with this wickedness.

For this reason, we attempt to generalise these systems to learn in similar environments, made slightly more wicked by being made *information-imperfect*, so that different players have different (but nonetheless accurate) information about the game state. To this end, we create information-imperfect variants of existing information-perfect games on which these systems have already been successfully trained.
(We assume that all games retain *complete information* - that is, information about the rules and their opponent's goals.)

In this paper, we aim to answer 4 questions about information-imperfect games:

1) How well does an agent trained on an information-imperfect version of a game perform on an information-perfect version?
2) How can agents trained on imperfect information be adapted for perfect information?
3) How can we improve performance in information-imperfect settings?
4) How well does learning and experience transfer from information-perfect to information-imperfect versions of games?

Our hypotheses are as follows.

1) Agents trained on an information-imperfect version of a game are able to adapt well to information-perfect environments.
2) Agents trained on imperfect information cope better with perfect information with use of a transformer.
3) We can improve performance in information-imperfect settings by processing the neural network inputs and outputs with a transformer.
4) There will be at least some useful transfer of learning from the perfect information version to the imperfect information version.

II. Related Work

Board game playing has long been used in the furthering of artificial intelligence techniques. Turing suggested chess as an example of one of the many tasks that a computer might one day learn to carry out at a human level [5] , and Samuel's original studies in machine learning made use of checkers. [6] This line of research continued with limited progress until Deep Blue defeated then-world champion Garry Kasparov at chess. [7]

Our work has its roots in AlphaZero [2] (developed by Silver et al. at DeepMind), a reinforcement learning system that outperformed both human and AI champions of chess, Go and shōgi within hours, learning entirely through self-play. It combined a deep neural network with Monte-Carlo tree search to value and explore possible future board states before selecting the one it deemed most likely to lead to victory. AlphaZero was later used to develop MuZero [3], which outperformed AlphaZero on all 3 games and also achieved state-of-the-art performance on Atari games. It uses MCTS internally and is able to make more consistent comparisons than AlphaZero. An open-source implementation of the methods used by these systems was developed by Thakoor, Nair and Jhunjhunwala.

[**?**] Their system was initially trained on Othello but their GitHub repository grew to include implementations of Tic-Tac-Toe, Connect Four, 3D Tic-Tac-Toe, Tafl, and others.

Past work at applying similar techniques to information-imperfect games include Brown et al.'s work on applying tree search to poker [8] (in a way that they deem unlikely to generalise easily to information-imperfect versions of combinatorial games like chess, Tic-Tac-Toe and Connect Four), Ciancarini and Favini's work applying Monte-Carlo tree search to Kriegspiel (an information-imperfect chess variant) [9], and AlphaStar, developed by Arulkumaran, Cully and Togelius (also from DeepMind) to play the real-time strategy game StarCraft II. [10] This game features inherent information imperfection and revolves in large part around exploration of the in-game map.

III. Methods

Our main code base was forked from the open-source implementation of AlphaZero developed by Thakoor, Nair and Jhunjhunwala [**?**], so our methods are largely derived from theirs (which is in turn derived from Silver et al. [2]). The high-level approach is to train a neural network to predict the value of any given board state along with a probability distribution over all available moves, use that information to evaluate possible future board states with Monte-Carlo tree search, and find the currently available move that is most likely to lead to victory. The network is trained entirely through self-play.

The following sections outline aspects specific to our information imperfection-focused approach.

A. Invisible Games

In applying reinforcement learning techniques to information-imperfect board games, we have found it helpful to be able to compare our results to those obtained on information-perfect games. We have therefore focused on imperfect variants of existing information-perfect games. We introduce the generalisable notion of *invisible games* - variants of existing games made through a general conversion process. This process allows information imperfection to be introduced to most combinatorial board games. We have based this notion on Kriegspiel, which is sometimes called Invisible Chess. In Kriegspiel, players cannot see each others' pieces, only their own, and attempt to make moves and put their opponents in checkmate despite their lack of information.

Note that the games concerned in this report are all two-player games. This need not be the case, as our methods are generalisable to games with more players, but for our purposes we assume two players in all games.

We follow these general principles in converting a game to its invisible variant:

- Invisible games include 3 separate boards: the *true board*, and a separate *visible board* for each player.
- Moves made by a player affect the true board and their visible board, but not their opponent's visible board.
- A move made by a player that is valid on their visible board but invalid on the true board results in a *collision*.
- Player must re-attempt moves that cause collisions.
- When successful moves and collisions allow a player to make a logically certain deduction about the true board, it is reflected on their visible board.
- Otherwise, all rules are as in the original game.

This method can also be thought of as giving the players a different 'interface' with which to play the original, information-perfect game.

There are other methods of creating information imperfection in games. For example, Newman et al. [11] describe Reconnaissance Chess, in which players may reveal a 3 by 3 section of a chessboard every turn before moving. We observe several advantages in our method of creating information imperfection in games:

- It is applicable to a wide range of combinatorial board games, including many not discussed in this paper.
- The format of the visible boards can be kept similar to the true board, which often allows agents designed to play the perfect version of the game to play the imperfect version, and vice versa.
- Giving no penalty for collisions (as opposed to forcing the player to forfeit their turn) encourages agents to balance probing the board for information with making optimal moves.

In this project, we focus on invisible versions of two games: Tic-Tac-Toe and Connect Four.

In **Invisible Tic-Tac-Toe**, players take turns placing pieces on an n by n board, attempting to place n in series in any cardinal direction. A collision at some position indicates that a piece of the other player's occupies that position. $n = 3$ by default.

In **Invisible Connect Four**, players take turns placing a piece in 1 of p columns of height q. The piece must occupy the lowest position in the chosen column. Players are shown where their pieces are placed (which allows them to deduce their opponent's pieces' positions in the same column). A collision in some column indicates that all unoccupied positions in the column are occupied by the other player's pieces. Players win by placing n pieces in series in any cardinal direction. $p, q, n = 7, 6, 4$ by default.

Note that these games are both *placement-based* games - that is, pieces are placed on the board and stay in the same state until the game concludes.[1] As will be explained in the following section, this property allows Monte-Carlo tree search to be adapted much more easily.

In Kriegspiel, announcements made to both players are a vital part of the game (for example, an announcement that a player has been put in check). These create common knowledge and allow players to guess how much knowledge

[1] While Tic-Tac-Toe is commonly played on paper, it can easily be played with physical pieces, just as Connect Four can be played on paper. We use the terms 'place' and 'piece' to refer to the method of making a move and the indicator of that move in the gamestate, respectively.

the other player has (as successful moves and collisions are announced separately). These have no equivalent in Invisible Tic-Tac-Toe or Invisible Connect Four, and so are left out of our model.

B. Monte-Carlo Tree Search

Monte-Carlo Tree Search (MCTS) forms a vital part of the original AlphaZero architecture, allowing the algorithm to explore possible future board states to better evaluate which moves are most likely to lead to victory. Possible states are valued according to their value (as evaluated by the neural network), weighted by their probability of occurring. The agent plays against itself, responding to its own moves in the ways it believes its opponent is likely to. Applying this method to info-imperfect games presents an obvious challenge: **how can an agent simulate its opponent's moves without knowing its opponent's position?** Allowing it to use the true game board would violate the game's information imperfection, allowing it to probe every option it has to find the most valuable option. Therefore any tree search used by an agent in info-imperfect games must operate using only the information available to the agent.

Suppose that Xavier is playing a game of Invisible Tic-Tac-Toe against Olivia. Xavier goes first and places Xs while Olivia places Os. The following will describe how Xavier might use our version of MCTS, with only his visible board state (which includes 3 Xs and 1 O) available to him. (Recall that, in our model, the game board includes three board states: a true state and a visible state for each of Xavier and Olivia.) Xavier constructs a new model game board based on his visible board state. In this new board, Xavier's visible state stays the same and also defines the true state. Olivia's visible board state (or rather, Xavier's model thereof) includes only what he knows to be visible to her - in this case, only the single O is visible on Olivia's board. As in regular MCTS, Xavier makes all moves for both himself and Olivia, with each new state reached creating a new node on the tree. As Olivia, Xavier first makes as many moves as are needed to 'catch up' - in this case, he moves as Olivia until two Os are placed before making his own moves. This will likely involve in collisions, adding Xs to Olivia's board and recreating the information asymmetry that exists in the original boards. Once Olivia has caught up, Xavier continues as normal.

As with regular MCTS, this results in a tree representing possible board states with values weighted by their assigned probabilities. The difference is that in our case, these probabilities represent possible *past* board states as well as future states. Further, it not only explores true board states, but visible board states for both players. These probabilities are weighted towards moves that Xavier believes Olivia is likely to have made, allowing him to explore different possible past moves and cope with the game's intrinsic ambiguity.

This approach works well with games that revolve around the permanent placement of pieces/marks within a known coordinate system (such as Tic-Tac-Toe and Connect Four) - Xavier can estimate Olivia's position, without knowledge of her previous moves, simply by placing extra pieces. It is for this reason that these games were chosen for analysis in this report. Games that involve moving pieces (such as chess) or changing pieces (such as Othello) are less well suited to this MCTS workaround, and require a different search algorithm (or for the neural network to be used alone).

Other search methods have been used successfully in information-imperfect contexts: Brown et al. [8] allowed for trees to be made and averaged over all possibilities based on common knowledge between players. As they acknowledged, this is intractable for combinatorial games such as those discussed in this paper. (There is also little useful common knowledge in our Invisible board game variants).
Another option was explored by Ciancarini and Favini [9] - exploring trees of possible announcements (such as moves being valid or invalid) rather than possible board states. This worked well for their Kriegspiel agent, showing potential for this variety of MCTS in imperfect contexts.

C. MuZero

As well as our adaption of AlphaZero, we use a modified version of MuZero for our learning transfer experiments. Our adapted MuZero codebase can be found here. We use the MuZero implementation developed by Duvaud et al. [12], which uses the Gym API [13] and contains implementations of Tic-Tac-Toe and Connect Four.

MuZero's MCTS uses learned internal representations rather than an external board, so we have left it as-is. This internal MCTS means that MuZero is able to cope with other varieties of information-imperfect games without much modification; however, to keep the results comparable with our AlphaZero implementation, we have continued to use the same games. We have implemented Gym-compatible versions of Invisible Tic-Tac-Toe and Invisible Connect Four using the existing information-perfect implementations as a base.

We have used MuZero rather than AlphaZero for our learning transfer experiments because its use of learned MCTS (in addition to its learned move and board valuations, which it shares with AlphaZero) means that more of the system is based on learned (rather than given) representations, making it easier to see the degree to which learning on one environment is able to help in a different environment.

D. AlphaZero

As previously mentioned, we use Thakoor, Nair and Jhunjhunwala's *Alpha Zero General* implementation [**?**] as our base. Our baseline model takes in as input the board state. This is passed through four 3x3 Conv2D layers with batch normalization and a relu activation function. This is flattened, then passed to a feed forward network to estimate the values v (probability it wins) and p_i (probability distribution of moves).

We update this architecture by adding in a transformer [14] layer before the input is passed to the convolutional layers. We hope that the transformer's attention mechanism allows the model to pay more attention to important parts of the board.

We also make the model deeper by doubling the number of convolutional layers from four to eight.

The training process for our models is as follows. All models are trained on imperfect information, while their performance is measured on both information-imperfect and information-perfect versions of the games. First we define a number of iterations to train the model. Each iteration we execute n episodes. An episode simulates playing the game against itself for a user defined number. This generates training examples which are added to a global pool of training instances. Given these training instances it trains the network for a used defined number of epochs e. The best model is stored. After the network is trained if it beats the best model a certain number of times (more than 60%) then this becomes the new best model. This process is repeated for every iteration.

IV. Experiments and Results

As previously stated, this paper aims to answer 4 questions about information-imperfect games:

- How well does an agent trained on an information-imperfect version of a game perform on an information-perfect version?
- How can agents trained on imperfect information be adapted for perfect information?
- How can we improve performance in information-imperfect settings?
- How well does learning and experience transfer from information-perfect to information-imperfect versions of games?

To evaluate the performance of our models in accordance to these questions we chose to play Tic-tac-toe and Connect 4 and adapted them to imperfect versions. Time constraints served a large barrier in implementing more advanced games such as checkers or chess. We used two models, AlphaZero and MuZero.

A. AlphaZero

We trained four Tic-Tac-Toe (TTT) and four Connect 4 (C4) models. All models are trained on the imperfect information variant of the games that we defined. We train using a NVIDIA Quadro RTX 8000 (48GB) and applied grid search to find all parameters defined below.

The four TTT models are trained as follows. The first was trained on the original architecture on a 3x3 board, the second was trained on the updated architecture on a 3x3 board. The last two were trained equivalently, except on a 4x4 TTT board instead. The models are referenced by TTT for the base architecture, a * afterwards for the updated architecture. The board dimension it was trained on follows the model (e.g. TTT* (3x3)). Each model is trained for 20 iterations, 20 epochs for the network, a learning rate of 0.001, batch size of 64, 200 pooling games every iteration and a dropout rate of 0.24.

TTT (3x3) trained in 3 hours, TTT* (3x3) trained in 6 hours, TTT (4x4) trained in 5 hours and TTT* (4x4) trained in 8 hours.

The four Connect 4 models are trained on the same default board size (6x7). The first model is trained with 25 MCTS simulation with the original architecture, the second similarly except on the updated architecture. The last two are trained similarly except 100 MCTS simulations were used instead. We hope that the 100 simulations will improve the models performance. Each model is referenced as follows. C4 represents the baseline model on the original architecture, a * represents the updated architecture. A + represents 100 MCTS simulations and no + represents 25 MCTS simulations. Each model is trained for 20 iterations, 20 epochs for the network, a learning rate of 0.001, batch size of 64, 200 pooling games every iteration and a dropout rate of 0.24.

C4 trained for 7 hours, C4* trained for 10 hours. The 100 MCTS simulation variants trained for an additional 1 hour over the 25 MCTS versions.

1) Tic-Tac-Toe: Table I shows the results of out models against a random opponent. We see that on the 3x3 board the updated architecture performs worse on the the perfect version, but better on the imperfect version compared to the baseline architecture/model. The same trend looks to hold for the 4x4 board, however on a perfect board the updated architecture gets more draws, while on the imperfect board it get significantly more wins than the baseline model, but with less draws and more losses.

In a head to head match up (TTT versus TTT*) on a 3x3 board, TTT wins 57.6 games on average, and loses 37.8 to the updated architecture when there is perfect information. For imperfect information this trend holds, but with slightly more draws and less wins for TTT II.

For a 4x4 board and perfect information, we get 99.8 draws on average and 0.2 wins for the updated architecture. This is good as this should occur during optimal play. However on the imperfect variant, TTT wins 50.4 games on average, compared to 36.2 for the updated model III.

The results here suggest that the updated architecture has a negative effect on out model and may not suit the static board (i.e. we know exactly where we place out piece. On games like Connect 4 where we don't know exactly where our piece drops, a transformer may be better suited.) Another hypothesis is that the transformer may push the model to a longer training time to converge to the same point, but it has higher potential they we haven't reached yet.

2) Connect 4: Table IV shows the results for our Connect 4 models against a random opponent. Surprisingly we see that our models with 100 MCTS simulations performs worse on average for our models with the updated architecture and similarly for the original model. We see not much of a change between perfect and imperfect version performance, with it either performing slightly better or worse.

Table V shows head to head performance of our models on imperfect information. We notice that a general trend is that the updated model with 100 MCTS simulations performs

Tic-Tac-Toe Versus Random Opponent on Perfect and Imperfect Settings				
(Win, Loss, Draw)	Perfect (3x3)	Imperfect (3x3)	Perfect (4x4)	Imperfect (4x4)
TTT(3x3)	(57.6, 37.8, 4.6)	(50.4, 36.2, 13.4)	-	-
TTT*(3x3)	(57, 38, 5)	(51.2, 36.8, 12.0)	-	-
TTT(4x4)	-	-	(45, 30.8, 24.2)	(30, 27.2, 42,6)
TTT*(4x4)	-	-	(40, 29.8, 30.2)	(45.6, 37.6, 16.8)

TABLE I: Measure of performance of the trained model on perfect and imperfect settings. TTT refers to the original architecture, while TTT refers to the updated architecture. (n x n) refers to what board size the model was trained on. The results are averaged over 10 sets of 100 games.*

Tic-Tac-Toe (3x3) Head-To-Head Perfect and Imperfect Settings		
Tic-Tac-Toe (3x3)	Perfect	Imperfect
(TTT win, TTT* win, Draw)	(57.6, 37.8, 4.6)	(50.4, 36.2, 13.4)

TABLE II: Measures the performance of the original and updated architectures in a head to head match up on both perfect and imperfect settings. The results are averaged over 10 sets of 100 games.

Tic-Tac-Toe (4x4) Head-To-Head Perfect and Imperfect Settings		
Tic-Tac-Toe (4x4)	Perfect	Imperfect
(TTT win, TTT* win, Draw)	(0, 0.2, 99.8)	(36.6, 28.6, 34.8)

TABLE III: Measures the performance of the original and updated architectures in a head to head match up on both perfect and imperfect settings. The results are averaged over 10 sets of 100 games.

Connect 4 Versus Random Opponent on Perfect and Imperfect Settings		
(Win, Loss, Draw)	Perfect (6x7)	Imperfect (6x7)
C4	(50.4, 49.6, 0)	(49.6, 50.4, 0)
C4*	(52.7, 47.3, 0)	(53.2, 46.8, 0)
C4+	(50, 50, 0)	(51.2, 48.8, 0)
C4+*	(49.8, 50.2, 0)	(48.8, 51.2, 0)

TABLE IV: Measure of performance of the trained model on perfect and imperfect settings. C4 refers to the original architecture, while C4 refers to the updated architecture. Both of these are trained with 25 MCTS simulations allowed. C4+ and C4+* are both the original and updated architecture respectively, but '+' refers to 100 MCTS simulation allowed. (n x n) refers to the board size. The results are averaged over 10 sets of 100 games.*

better against the baseline model, but otherwise equal with the other models. We see that there are situations where some models perform well against others, and other situations where they perform equal, or slightly worse. This suggests that the models play style is a good counter to its opponents. As this is imperfect information all models will have a strategy to counteract the imperfect informations, against some opponents it will work better, against others it will perform worse.

Table VI shows head to head results for the original architecture against the updated architecture on the perfect information variant of Connect 4. We see that for the 100 MCTS version, the updated model performs better, while for the 25 MCTS version a similar results holds.

B. Testing Learning Transfer

Perfect information games are often much easier to train on as there is more information for the network to learn from. But this principle can go beyond just board games and can be applied to any real world situation that can be modeled as a game. For most such models, training on a perfect information version can be much faster even though the real world is full of hidden information. So if there is significant learning transfer from perfect to imperfect information versions, such models can take advantage of this learning transfer for better performance. We will use our perfect and imperfect information versions of Tic-Tac-Toe and Connect Four to test this hypothesis.

In our transfer learning experiments (using MuZero), we trained the network on perfect Tic-Tac-Toe and Connect Four for 100,000 steps. The models were then evaluated against a random opponent for consistency between the game versions. A more sophisticated opponent would have to take into account the information provided which would be different between the perfect and imperfect information versions.

The performance of MuZero models on the different game versions is tabulated below. The scores are averaged over 100 games. The score given for winning is 20 for TTT and 10 for C4. For draws and losses, the score is 0.

Game	Perfect	Imperfect	Max
TTT	18.4	16.8	20
C4	8.8	7.3	10

Testing learning transfer with MuZero on different games and versions

V. Discussion

A. Adaptation to Information-Perfect Environments

I shows our models' performance on information-perfect and -imperfect variants of 3x3 and 4x4 Tic-Tac-Toe.

Connect 4 Head-To-Head Imperfect Information				
(Row win, Column Win, Draw)	C4	C4*	C4+	C4+*
C4	-	(52.4, 47.6, 0)	(48, 52, 0)	(47.6, 52.4, 0)
C4*	(47.6, 52.4, 0)	-	(51.8, 48.2, 0)	(50.6, 49.4, 0)
C4+	(52, 48, 0)	(48.2, 51.8, 0)	-	(49.8, 50.2, 0)
C4+*	(52.4, 47.6, 0)	(49.4, 50.6, 0)	(50.2, 49.8, 0)	-

*TABLE V: Measures the performance of the original and updated architectures in a head to head match on imperfect information. The results are averaged over 10 sets of 100 games. * represents the updated architecture and + represents the 100 MCTS version, otherwise it is the 25 MCTS variant.*

Connect 4 Head-To-Head Perfect Information		
(Row Win, Column Win, Draw)	C4+	C4*
C4	-	(48, 52, 0)
C4*+	(50.8, 49.2, 0)	-

*TABLE VI: Measures the performance of the original and updated architectures in a head to head match on perfect information. The results are averaged over 10 sets of 100 games. * represents the updated architecture and + represents the 100 MCTS version, otherwise it is the 25 MCTS variant.*

Imperfect vs Perfect 3x3 Tic-Tac-Toe			
Result	Perfect Rules (Standard Game)	Imperfect Rules (Blind)	Total
Perfect Agent Wins	0	825	825
Imperfect Agent Wins	0	1164	1164
Draw	2000	11	2011
Total	2000	2000	4000

TABLE VII: Result of pitting agents against each other on normal/imperfect rules over 4000 Games

Imperfect vs Perfect 4x4 Tic-Tac-Toe			
Result	Perfect Rules (Standard Game)	Imperfect Rules (Blind)	Total
Perfect Agent Wins	3	77	80
Imperfect Agent Wins	3	79	82
Draw	194	44	238
Total	200	200	400

TABLE VIII: Result of pitting agents against each other on normal/imperfect rules over 400 Games

In this experiment, the addition of the transformer helps the model perform better in information imperfect-settings, but actually makes it perform worse in information-imperfect settings compared to its performance without a transformer. These results support our 3rd hypothesis (that transformers help in information-imperfect settings), albeit somewhat weakly, as the differences between the two models is slight.
This table contradicts our 2nd hypothesis (that transformers help agents trained in information-imperfect settings adapt to information-perfect settings). This could be explained by the fact that a transformer changes the form of input to the neural network. The transformer might be changing common information-imperfect and information-perfect board states in different ways, causing the agent to react better in known (information-imperfect) environments but failing to recognise similarities between information-perfect and information-imperfect play.
This table also somewhat supports our 1st hypothesis (that agents trained with information imperfection can adapt well to perfect-information environments), although an adequate information-perfect 3 by 3 Tic-Tac-Toe agent should be able to prevent a loss against any opponent. The agents do outperform a random opponent, showing clear evidence of learning transfer, but do not seem to be able to block their opponent. It is possible that being trained in an environment where the opponent's pieces are invisible makes agents less likely to learn to block their opponents from getting 3 pieces in a row, and more likely to focus on their own pieces.

II shows our model's performance against itself, with and without a transformer, on perfect and imperfect 3x3 Tic-Tac-Toe.
This table further contradicts our hypothesis that transformers are likely to help in information-perfect environments, with our transformer-free model performing noticeably better with perfect information.
This table also provides evidence against our hypothesis that transformers help in information-imperfect environments, with the transformer-free model also outperforming the transformer model in information-imperfect Tic-Tac-Toe.

III shows our model's performance against itself, with and without a transformer, on perfect and imperfect 4x4 Tic-Tac-Toe.
Strangely, when playing information-imperfect 4x4 Tic-Tac-Toe, our agents almost always draw. This is strange because 3x3 Tic-Tac-Toe is more easily drawn than the 4x4 version. This may suggest that our 3x3 models have not properly converged. This may be due to the use of 25 MCTS simulations per turn, which does not always allow an agent to properly

Imperfect vs Perfect 5x5 Tic-Tac-Toe			
Result	Perfect Rules (Standard Game)	Imperfect Rules (Blind)	Total
Perfect Agent Wins	18	47	65
Imperfect Agent Wins	28	95	123
Draw	154	95	123
Total	200	200	400

TABLE IX: Result of pitting agents against each other on normal/imperfect rules over 400 Games

explore all its options (due to the large amount of possible past states that it must explore before exploring any future states).

The information-imperfect results show further evidence against our 2nd hypothesis, with the transformer layer harming the model's performance.

Table IV shows our model's performance against itself, with and without a transformer, on perfect and imperfect Connect Four. It also shows differences between use of 25 MCTS simulations and 100 MCTS simulations.

In this game, the transformer improves performance when using 25 MCTS simulations but not when using 100. (We suspect that the network may be training worse with 100 MCTS simulations. If this is true, then the transformer improves performance in both perfect and imperfect Connect Four.) While this may seem to contradict the results obtained on Tic-Tac-Toe, this is not necessarily the case. The transformer likely made the model perform worse on perfect Tic-Tac-Toe due to the large differences in observations between the information-imperfect and information-perfect versions. These differences are less drastic in Connect Four, because all of the opponent's pieces in a column are revealed at once (rather than having to be revealed cell-by-cell) and agents of similar skill levels are likely to want to place pieces in the same columns. This suggests that transformers may help in information-perfect games environments where information-imperfect inputs are similar to information-perfect versions.

Similarly to Table I, this table presents evidence somewhat contradicting our first hypothesis that agents trained on imperfect information should perform well on perfect information. These agents perform better on perfect than on imperfect information, but perform worse than they should against random opponents in an information-imperfect setting. The difference is less drastic than in Table I, suggesting that the lower difference in available information meant that information-imperfect Connect Four is easier than information-imperfect Tic-Tac-Toe.

Table V shows our Connect Four models' performances against each other in Invisible Connect Four. While there are differences between each agent's success rate vs. other agents, no agent loses more than 47% of their games against any agent. This suggests that the transformer and number of MCTS simulations both make little difference.

C4 loses against C4+ and C4+*, while they lose to C4*. This suggests that in the 25-simulation agents (i.e. the ones that have converged) the transformer makes a positive difference. However, the C4 vs. C4* results contradict this. Again, the variation is small enough to suggest that none make a difference, likely due to the low level of information imperfection in Invisible Connect Four.

Table VI shows a virtual tie between our transformer and transformer-free models in regular Connect Four. This may be due to the lower degree of ambiguity inherent to this game compared to Invisible Connect Four, which could reduce the advantage that the transformer conveys.

B. Perfect vs Imperfect Agent

(In this section, we use the term *perfect agent* to describe an agent trained on a information-perfect version of a game, and similarly for an textitimperfect agent.)

Table VII shows that when agents are pitted against each in 3x3 Tic-Tac-Toe, an imperfect agent will perform equally in standard game rules against a perfect agent, with most games all games being ended in a draw. This is despite the perfect agent having a "home advantage", from being trained on the standard game rule settings. Observe that in Invisible Tic-Tac-Toe, the imperfect agent won the majority of games (1164 of 2000) against 825 wins of the perfect agent, with very few draws. In Table VIII, using a 4x4 Tic-Tac-Toe board, a large majority of games will end in a draw between perfect and imperfect agents. In Invisible Tic-Tac-Toe, the imperfect agent wins about the same number of games. In table IX, we expand this to a 5x5 board. In standard Tic-Tac-Toe, both agents perform equally and tie a majority of games, while in tInvisible Tic-Tac-Toe the imperfect agent wins a majority of games. This supports our 1st hypothesis that an agent trained on perfect information will adapt well to perfect information, as it appears that an imperfect agent performs almost identically to a perfect agent in both regular and Invisible Tic-Tac-Toe.

The trained models appear to have converged to optimal play on perfect 3x3 Tic-Tac-Toe, even against human players. In Invisible Tic-Tac-Toe, the model appears to have a strong understanding of tactics involved in an imperfect setting, including against human players. However, the strength of its play decreases with the increased board size, due to the increased number of possible moves for both players increasing the number of possible board states, making the agent's constant amount of MCTS simulations less effective.

C. Learning Transfer

Our learning transfer experiments exhibit clear signs of transfer of learning. The models perform almost as well on the information-imperfect variants as on the information-perfect variants. However, there is a slight drop in performance on Invisible Tic-Tac-Toe. This is to be expected, since the models were not trained on these variants and they involve much

more uncertainty. It should be noted that these results were obtained against a random opponent, so the performance may be somewhat inflated.

Qualitatively, the MuZero Tic-Tac-Toe model performs quite well, even against human opponents. The model always blocks game-winning moves, showing that it has a clear understanding of the game. However, it makes sub-optimal starting moves, failing to place its piece in the center when starting out. Regardless, it is impressive that it still performs quite well on Invisible Tic-Tac-Toe. The Connect Four model, on the other hand, was somewhat more problematic. It seems to follow a set strategy regardless of its opponent's moves. This may be because Connect Four is a more complex game compared to Tic-Tac-Toe, and therefore requires more training. Still, there is noticeable learning transfer happening towards Invisible Connect Four.

VI. Conclusion

We have shown AlphaZero can reach effective performance on information-imperfect games. This performance improves on perfect information, and while this improvement is not as drastic as might be expected, agents trained on imperfect information seems to rival their imperfect counterparts. Adding a transformer layer to the architecture improves performance on imperfect information by helping the model focus on the little information it does have, but not as well on perfect information, performing worse when information-imperfect games differ substantially from their information-perfect originals. This illustrates the importance of similar information representations between games to transfer learning in AlphaZero.

Our learning-transfer experiments have shown that with MuZero, learning can be transferred effectively between different versions of games. This technique can be useful in various applications both within the domain and board games and outside of it.

Together, our results show that AlphaZero's game-playing techniques (as exhibited in both AlphaZero and MuZero) can, with some modifications, be adapted to deal effectively with imperfect information variants of the information-perfect games for which it is typically used. We hope that these results can be generalised further to allow it (or similarly effective systems) to find solutions to wicked real-world problems with similar effectiveness, and bring us closer to more general artificial intelligence.

VII. Future Work

Future research directions on this topic could investigate:

- Playing different kinds of information-imperfect games (such as with different tree search techniques, common knowledge integration, or constant knowledge representations for movement-based games)
- Quantitative analysis of information imperfection in these and similar games
- Optimal play of information-imperfect game variants
- How humans deal with information imperfection
- Training/testing on more variants of the same game (possibly in tandem)
- How MuZero adapts to other information-imperfect games
- Progressive learning (stepwise training) for MuZero on imperfect information games

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, S. Dieleman, D. Grewe, N. Kalchbrenner, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, *Mastering the game of Go with deep neural networks and tree search*, pp. 484–489. Nature, 2016.

[2] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, and et al., "A general reinforcement learning algorithm that masters chess, shogi, and go through self-play," *Science*, vol. 362, p. 1140–1144, Dec 2018.

[3] J. Schrittwieser, I. Antonoglou, T. Hubert, K. Simonyan, L. Sifre, S. Schmitt, A. Guez, E. Lockhart, D. Hassabis, T. Graepel, and et al., "Mastering atari, go, chess and shogi by planning with a learned model," p. 21, 2020.

[4]

[5] A. M. TURING, "I.—COMPUTING MACHINERY AND INTELLIGENCE," *Mind*, vol. LIX, pp. 433–460, 10 1950.

[6] A. L. Samuel, "Some studies in machine learning using the game of checkers," *IBM Journal*, p. 21, Jul 1959.

[7] M. Campbell, A. Hoane, and F. hsiung Hsu, "Deep blue," *Artificial Intelligence*, vol. 134, no. 1, pp. 57 – 83, 2002.

[8] N. Brown, A. Bakhtin, A. Lerer, and Q. Gong, "Combining deep reinforcement learning and search for imperfect-information games," *arXiv:2007.13544 [cs]*, Jul 2020. arXiv: 2007.13544.

[9] P. Ciancarini and G. P. Favini, "Monte carlo tree search in kriegspiel," *Artificial Intelligence*, vol. 174, p. 670–684, Jul 2010.

[10] K. Arulkumaran, A. Cully, and J. Togelius, "Alphastar: An evolutionary computation perspective," *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, p. 314–315, Jul 2019. arXiv: 1902.01724.

[11] A. J. Newman, C. L. Richardson, S. M. Kain, P. G. Stankiewicz, P. R. Guseman, B. A. Schreurs, and J. A. Dunne, "Reconnaissance blind multi-chess: an experimentation platform for isr sensor fusion and resource management," in *Proceedings of SPIE* (I. Kadar, ed.), p. 984209, May 2016.

[12] A. H. Werner Duvaud, "Muzero general: Open reimplementation of muzero." https://github.com/werner-duvaud/muzero-general, 2019.

[13] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," 2016.

[14] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," 2017.