Towards a General Transfer Approach for Policy-Value Networks

Anonymous authors
Paper under double-blind review

Abstract

Transferring trained policies and value functions from one task to another, such as one game to another with a different board size, board shape, or more substantial rule changes, is a challenging problem. Popular benchmarks for reinforcement learning (RL), such as Atari games and ProcGen, have limited variety especially in terms of action spaces. Due to a focus on such benchmarks, the development of transfer methods that can also handle changes in action spaces has received relatively little attention. Furthermore, we argue that progress towards more general methods should include benchmarks where new problem instances can be described by domain experts, rather than machine learning experts, using convenient, high-level domain specific languages (DSLs). In addition to enabling end users to more easily describe their problems, user-friendly DSLs also contain relevant task information which can be leveraged to make effective zero-shot transfer plausibly achievable. As an example, we use the Ludii general game system, which includes a highly varied set of over 1000 distinct games described in such a language. We propose a simple baseline approach for transferring fully convolutional policy-value networks between any pair of games modelled in this system, and present extensive results—including various cases of highly successful zero-shot transfer.

1 Introduction

While deep reinforcement learning (RL) (Sutton & Barto, 2018) models have produced many strong and noteworthy results (Mnih et al., 2015; Silver et al., 2016; Brown & Sandholm, 2019; Schrittwieser et al., 2020), they are also frequently associated with limitations in terms of generalisation and transfer (Rusu et al., 2016; Marcus, 2018; Zhang et al., 2018; Justesen et al., 2018; Cobbe et al., 2019; Zhang et al., 2020; Zhu et al., 2020). For example, models trained using AlphaGo (Silver et al., 2016) or its successors (Silver et al., 2017; 2018) can perform well in the exact games they are trained on (such as Go on a 19×19 board), but cannot at all play even slight variants (such as Go on a 17×17 board).

Standardised benchmarks and frameworks with collections of environments play an important role in evaluating and advancing deep RL methods. Examples of such frameworks include the Arcade Learning Environment (ALE) (Bellemare et al., 2013; Machado et al., 2018), OpenAI Gym (Brockman et al., 2016), ProcGen (Cobbe et al., 2020), and MiniHack (Samvelyan et al., 2021). Most of these are restricted in terms of extensibility or variety of environments. ALE contains a substantial number of games, but they are all arcade games, and all restricted to a discrete action space of up to 18 elements—most of which are movement directions that are semantically similar between different games. ProcGen was specifically created to benchmark generalisation in RL, but it is restricted to environments with exactly 15-dimensional discrete action spaces, and 64×64×3 RGB state observations. Hence, while it can benchmark generalisation across levels, it cannot benchmark generalisation across state or action representations. OpenAI Gym can in principle be extended to include any environment, but adding new environments consists of programming them from scratch (e.g., in Python), and determining their state and action representations by hand. Except for MiniHack, none of the frameworks listed above include any explicit form of task descriptions (arguably except for their raw source). This means that effective zero-shot transfer can never be reasonably expected to be possible when

tasks are made sufficiently varied in these frameworks, in the same way that humans also cannot be expected to continue playing well if a game's rules are changed without informing the human.

GVGAI (Schaul, 2013; Perez-Liebana et al., 2019), Ludii (Browne et al., 2020; Piette et al., 2020), Stratega (Perez-Liebana et al., 2020), Griddly (Bamford et al., 2021), and MiniHack (Samvelyan et al., 2021) are examples of frameworks that use relatively high-level domain-specific languages (DSLs) or game description languages (GDLs) to describe the rules of games and/or their levels. Such high-level languages are substantially easier to use for defining new environments than a general-purpose programming language. Over 1000 games have already been written in Ludii's GDL in a few years' time, of which approximately 10% were contributed by third-party users, including users with backgrounds in (board) game design rather than programming. For research towards more generally applicable artificial intelligence, we consider that models should support highly extensible and varied problem sets, where problem instances are defined by end users (such as domain experts) using such convenient languages, rather than machine learning experts or software engineers. While the frameworks listed as examples above are all for games, similar DSLs can also be used to describe different families of problems, such as staff scheduling problems (Schafhauser, 2010) or algorithm design (Sironi & Winands, 2021). High-level DSLs have also been shown to facilitate evolutionary generation of interesting new problems (such as new games) (Browne, 2009), which is another interesting property that may benefit the training of generalisable agents (Justesen et al., 2018; Cobbe et al., 2020; Jiang et al., 2021; Parker-Holder et al., 2022).

This paper focuses on transfer of policy-value networks—as trained by AlphaZero-based (Silver et al., 2018) approaches—between games in Ludii. We focus on AlphaZero-based approaches because it is a standard technique that is known to work well in 2-player zero-sum games (Silver et al., 2018; Cazenave et al., 2020), and also provides an interesting challenge for transfer in that it involves both policy and value heads. All of the frameworks with DSLs listed above have a primary focus on a certain category of games. For example, GVGAI focuses on avatar-centric arcade games with action spaces of up to a size of 5, MiniHack focuses on NetHack-like games (Küttler et al., 2020), and Ludii primarily includes board games (but also some deduction puzzles and other abstract games). Of these, Ludii arguably has the greatest variety in domains, with many different board geometries (Browne et al., 2022) and action spaces. This makes it particularly interesting and challenging for transfer in deep RL, and allows for many different types of targeted experiments; it is easy to change board sizes, board shapes, flip win conditions, and so on, and evaluate how such different types of changes affect transfer.

We propose a relatively simple baseline approach for transfer between Ludii games, and extensively evaluate zero-shot transfer as well as transfer with fine-tuning in a wide variety of pairs of source and target games. One of two key ingredients of the transfer approach is the use of fully convolutional networks with global pooling for their ability to adjust to changes in the spatial dimensions of a game (i.e., changes in board sizes) (Shelhamer et al., 2017; Wu, 2019; Cazenave et al., 2020). In board games this is also typically associated with substantial changes in the size of the action space. This has been neglected in related work on transfer for deep RL, which almost exclusively tends to consider transfer between state spaces or representations. We use the AlphaZero-like training algorithms as implemented in Polygames (Cazenave et al., 2020) for training initial models and fine-tuning. The second key ingredient is that we leverage the shared, uniform state and action representations (Piette et al., 2021) that result from describing problems in a DSL or GDL like Ludii's to identify shared semantics in the non-spatial aspects of source and target domains.

Source and target domains are sometimes minor variants of the same game (e.g., with a single change in board size, geometry, or win condition), and sometimes different games altogether with a single shared theme (e.g., many different line-completion games that all have some form of line completion as win condition). In total, we report results for 227 different pairings of source and target domains. We identify particularly successful transfer results—including highly successful zero-shot transfer—when transferring from smaller games (often smaller boards) to larger ones, but also several other cases of successful transfer, and some with negative transfer (Zhang et al., 2020).

2 Background

This section provides relevant background information on self-play training, tensor representations, and fully convolutional architectures for transfer, in Polygames and Ludii.

2.1 Learning to Play Games in Polygames

Similar to AlphaZero, game-playing agents in Polygames use a combination of Monte-Carlo tree search (MCTS) (Kocsis & Szepesvári, 2006; Coulom, 2007) and deep neural networks (DNNs). Experience for training is generated from self-play games between MCTS agents that are guided by the DNN. Given a tensor representation of an input state s, the DNN outputs a value estimate $V(s)$ of the value of that state, as well as a discrete probability distribution $\mathbf{P}(s) = [P(s, a_1), P(s, a_2), \ldots, P(s, a_n)]$ over an action space of n distinct actions. Both of these outputs are used to guide the MCTS-based tree search. The outcomes (typically losses, draws, or wins) of self-play games are used as training targets for the value head (which produces $V(s)$ outputs), and the distribution of visit counts to children of the root node by the tree search process is used as a training target for the policy head (which produces $\mathbf{P}(s)$ outputs).

For board games, input states s are customarily represented as three-dimensional tensors of shape (C, H, W), where C denotes a number of channels, H denotes the height of a 2D playable area (e.g., a game board), and W denotes the width. The latter two are interpreted as spatial dimensions by convolutional neural networks. It is typically assumed that the complete action space can be feasibly enumerated in advance, which means that the shape of $\mathbf{P}(s)$ output tensors can be constructed such that every possibly distinct action a has a unique, matching scalar $P(s, a)$ for any possible state s in the policy head. A DNN first produces *logits* $L(s, a)$ for all actions a, which are transformed into probabilities using a softmax after masking out any actions that are illegal in s.

In some general game systems, it can be impossible to guarantee that multiple different actions will never share a single output in the policy head (Soemers et al., 2022). We say that distinct actions are *aliased* if they are represented by a single, shared position in the policy head's output tensor. In Polygames, the MCTS visit counts of aliased actions are summed up to produce a single shared training target for the corresponding position in the policy head. In the denominator of the softmax, we only sum over the distinct logits that correspond to legal actions (i.e., logits for aliased actions are counted only once). All aliased actions a receive the same prior probability $P(s, a)$ to bias the tree search—because the DNN cannot distinguish between them—but the tree search itself can still distinguish between them.

2.2 The Ludii General Game System

Ludii (Browne et al., 2020; Piette et al., 2020) is a general game system with over 1000 built-in games, many of which support multiple variants with different board sizes, board shapes, rulesets, etc. It provides suitable object-oriented state and action representations for any game described in its game description language, and these can be converted into tensor representations in a consistent manner with no need for additional game-specific engineering effort (Soemers et al., 2022). All games in Ludii are modelled as having one or more "containers," which can be viewed as areas with spatial semantics (such as boards) that contain relevant elements of game states and positions that are affected by actions. This means that all games in Ludii are compatible with fully convolutional architectures from an engineering point of view, although it is possible to describe games where spatial semantics have little meaning and the inductive biases encoded in convolutional layers are ineffective.

2.3 Transfer of Policy-Value Networks: the Case for Fully Convolutional Architectures

AlphaZero-like training approaches (Silver et al., 2018) typically require training from scratch before any new individual (variant of a) game can be played. Transfer learning (Taylor & Stone, 2009; Lazaric, 2012; Zhu et al., 2020) may allow for significant savings in computation costs by transferring trained parameters from a *source domain* (i.e., a game that we train on first) to one or more *target domains* (i.e., variants of the source game or different games altogether). We focus on transferring complete policy-value networks—with

policy as well as value heads—between a large variety of source and target domains as provided by Ludii. Crucially, the transfer of trained parameters for a network with a policy head requires the ability to construct a mapping between action spaces of source and target domains. This is in contrast to approaches that only transfer value functions, which only require the ability to create mappings between state spaces.

Following Subsection 2.1, we assume that game states s are represented as tensors of shape (C_{state}, H, W), and that the action space can be represented by a shape (C_{action}, H, W)—we may think of actions as being represented by an action channel, and coordinates in the 2D space of H rows and W columns. A network's policy head therefore also has a shape of (C_{action}, H, W). We focus on 2-player zero-sum games, which means that a single scalar suffices as output for the value head. Note that different games may have different numbers of channels and different values for H and W. It is common to use architectures that first process input states s using convolutional layers, but at the end use one or more non-convolutional layers (e.g., fully-connected layers) preceding the outputs. This leads to networks that cannot handle changes in the spatial dimensions (i.e., changes in H or W), and have no inductive biases that leverage any spatial structure that may be present in the action representation. Fully convolutional architectures with global pooling, such as those implemented in Polygames, can address both of those concerns. In particular the ability to handle changes in spatial dimensions is crucial for transfer between distinct games or game variants, which is why we restrict our attention to these architectures in the majority of this paper. An example of such an architecture is depicted in Appendix A.

Formally, the goal of transfer is as follows. Let $\mathcal{S}$ denote a source task (i.e., a game we train in and transfer from), with a state space $S^{\mathcal{S}}$ and an action space $A^{\mathcal{S}}$. A trained policy-value network can produce value outputs $V(s)$, and policy outputs $\pi(s, a)$, for (tensor representations of) input states $s \in S^{\mathcal{S}}$ and actions $a \in A^{\mathcal{S}}$. Let $\mathcal{T}$ denote a target task (i.e., a game we transfer to and evaluate performance of agents in), with (potentially) different state and action spaces $S^{\mathcal{T}}$ and $A^{\mathcal{T}}$. Then, the goal of transfer is to use a network trained in $\mathcal{S}$ to initialise the weights of a new neural network, such that the new network can directly perform well and/or speed up learning in $\mathcal{T}$.

3 Transferring Parameters Between Games

The fully convolutional architectures as described in the previous section allow for DNNs trained in a source task $\mathcal{S}$ to be directly used in any target task $\mathcal{T}$ if it only has different values for one or both of the spatial dimensions H and W. However, it is also possible that different tasks have different numbers of channels C_{state} or C_{action} for state or action representations, or significantly different semantics for some of those channels. We propose a simple baseline approach for transferring the weights of fully convolutional policy-value networks in this setting.

The key insight that we use is that, when a wide variety of problems are all described in a single, high-level DSL—as is the case with over 1000 distinct games in Ludii—the framework can generally represent their states and actions in a uniform, consistent manner, where the different variables used internally closely correspond to the semantically meaningful high-level keyword that are present in the DSL (Piette et al., 2021; Soemers et al., 2022). Based on these consistent representations across the wide variety of problems that can be modelled in the same language, we can automatically identify variables of state or action representations that are semantically similar. In the case of states, these variables correspond to channels of the input tensors. In the case of actions, these variables correspond to channels of the policy output tensors. For example, an input plane that encodes all the empty positions in a source game is said to be semantically "equivalent" to another input plane that encodes all the empty positions for a target game. Note that this idea of equivalence is based on the data that is encoded, but not necessarily on how it affects strategies or optimal policies in a game. For example, a plane that encodes the presence of friendly pieces is semantically equivalent regardless of the game in which it encodes that data in terms of the raw data that it encodes, but may have very different implications in terms of the optimal strategy to play depending on whether the goal in a game is to connect pieces or to avoid connecting pieces. It may be possible to automatically learn how to account for such differences (Kuhlmann & Stone, 2007; Bou Ammar, 2013; Bou Ammar et al., 2014), but we consider this outside the scope of this paper.

Given equivalence relations between state and action channels for $\mathcal{S}$ and $\mathcal{T}$, we can copy parameters from a network trained in $\mathcal{S}$ to play $\mathcal{T}$. Without prior knowledge of the specific games under consideration, there is no explicit guidance on how to transfer other than the equivalence relations. Therefore, to build a rigorous notion of what would constitute "correct" transfer, we make the assumption that $\mathcal{S}$ and $\mathcal{T}$ *are exactly the same game, but with different state or action tensor representations*; some channels may have been added, removed, or shuffled. Under this assumption, we can think of correct transfer as any approach that ensures that, as much as possible, equivalent input states lead to equivalent outputs. This can be done by copying, reordering, duplicating, or removing channels of the first and final convolutional layers according to the equivalence relations; other parameters can be directly copied. This may sometimes only be approximately possible, because differences in representation can mean that $\mathcal{T}$ contains less information than $\mathcal{S}$; either representation may have only partial information in state tensors, or some degree of action aliasing. In practice $\mathcal{S}$ and $\mathcal{T}$ need not represent identical games, but if they are sufficiently similar the same transfer approach can still be beneficial. Appendix B provides more extensive technical details on this approach.

4 Experiments

This section discusses experiments[1] used to evaluate the performance of fully convolutional architectures, as well as transfer learning between variants of games and between distinct games. We used the training code from Polygames. For transfer learning experiments, we used games as implemented in Ludii v1.1.6. Every game used in these experiments is a zero-sum game for two players. All games may be considered sparse-reward problems in the sense that they only have potential non-zero rewards when terminal game states are reached, and no reward shaping is used. Some of the games, such as *Breakthrough* and *Hex*, naturally progress towards terminal game states with non-zero rewards regardless of player strength (even under random play), whereas others also have terminal game states with outcomes equal to 0 for both players (e.g., *Diagonal Hex*). Appendix D provides details on hyperparameter values used for training throughout all experiments.

4.1 Fully Convolutional Architectures

Our approach for transfer requires the use of fully convolutional architectures with global pooling to facilitate transfer between domains with differences in spatial aspects (e.g., different board sizes). The goal of our first experiment is to ensure that the choice for this type of architecture—as opposed to the more common architectures with fully-connected layers prior to the outputs (Silver et al., 2018)—does not come at a substantial cost in baseline performance. We selected a variety of board games as implemented in Polygames, and trained networks of various sizes and architectures, using 24 hours on 8 GPUs and 80 CPU cores per model. Models of various sizes—measured by the number of trainable parameters—have been constructed by randomly drawing choices for hyperparameters such as the number of layers, blocks, and channels for hidden layers. After training, we evaluated the performance of every model by recording the win percentage of an MCTS agent using 40 iterations per move with the model, versus a standard untrained UCT (Browne et al., 2012) agent with 800 iterations per move. These win percentages are depicted in Figure 1. In the majority of cases, `ResConvConvLogitPoolModel`—a fully convolutional model with global pooling—is among the strongest architectures. Fully convolutional models generally outperform ones with dense layers, and models with global pooling generally outperform those without global pooling. This suggests that using such architectures can be beneficial in and of itself, and their use to facilitate transfer learning does not lead to a sacrifice in baseline performance. Therefore, all transfer learning experiments discussed below used the `ResConvConvLogitPoolModel` architecture from Polygames. All models were trained for 20 hours on 8 GPUs and 80 CPU cores, using 1 server for training and 7 clients for the generation of self-play games.

4.2 Transfer Between Game Variants

We selected a set of nine different board games, as implemented in Ludii, and for each of them consider a few different variants. The smallest number of variants for a single game is 2, and the largest number of

[1]Source code URL to be added in non-anonymous version.

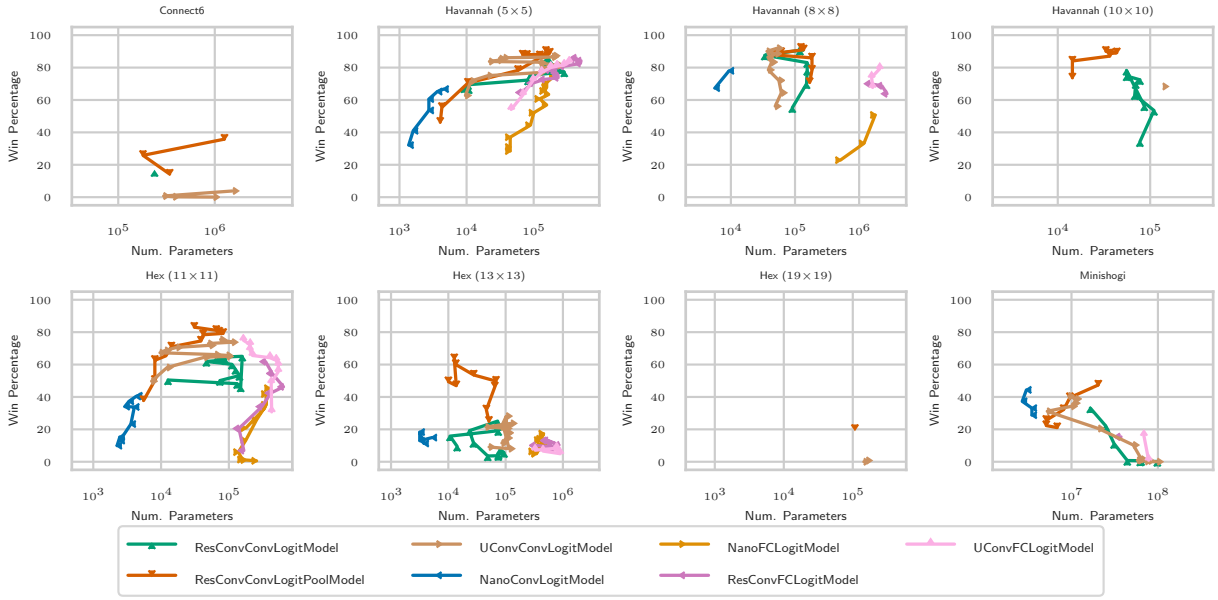
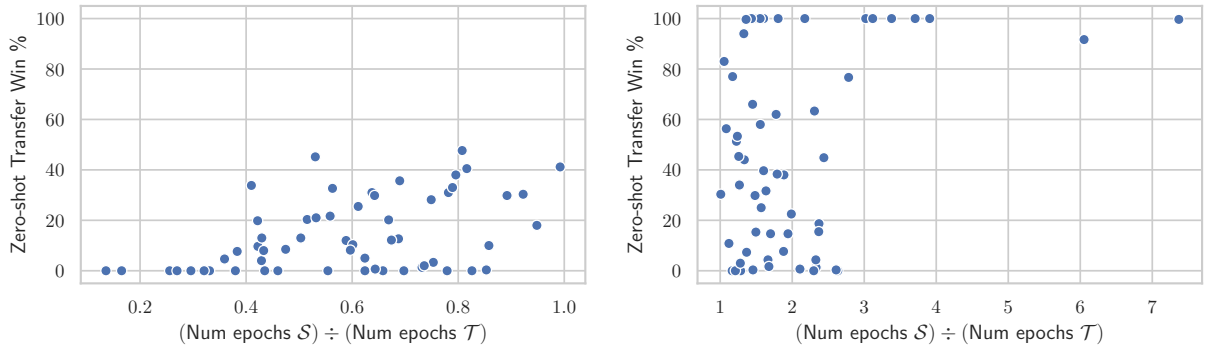


Figure 1: Win percentages of trained MCTS with 40 iterations/move vs. UCT with 800 iterations/move. `Nano`: shallow architectures. `ConvConv`: deep fully convolutional. `ConvFC`: fully connected layers after convolutional ones. `Pool`: adding global pooling; `U`: adding U-connections (Ronneberger et al., 2015). Deep is better than shallow, U-nets are slightly better than their classical counterparts, and deep nets are improved by using (i) fully convolutional policy heads (ii) global pooling.

variants for a single game is 6. In most cases, the different variants are simply different board sizes. For example, we consider *Gomoku* played on 9×9, 13×13, 15×15, and 19×19 boards as four different variants of *Gomoku*. We also include some cases where board shapes change (e.g., *Breakthrough* played on square boards as well as hexagonal boards), "small" changes in rules (e.g., *Broken Line* played with a goal line length of 3, 4, 5, or 6), and "large" changes in rules (i.e., *Hex* with the standard win condition and *Misère Hex* with an inverted win condition). The set of games is highly varied in aspects such as sizes of state and action spaces, board geometries (square boards and three different board shapes with hexagonal cells), types of win and loss conditions (based on completing lines of pieces, connecting chains of pieces, reaching specific areas of the board, or blocking the opponent from having any legal moves), move mechanisms (some games involve placing pieces anywhere on the board, others involve small steps or longer-range slides of pieces, or hopping over opposing pieces), mechanisms for removing pieces (in some games pieces are never removed, others involve capturing pieces by landing on them, hopping over them, or surrounding them), and so on. Further details on all the games and game variants used are provided in Appendix E.

We trained a separate model for every variant of each of these games, and transferred models from all variants to all other variants within each game. We evaluate zero-shot transfer performance for a source domain $\mathcal{S}$ and target domain $\mathcal{T}$ by reporting the win percentage of the model trained in $\mathcal{S}$ against the model that was trained in $\mathcal{T}$, over 300 evaluation games per $(\mathcal{S}, \mathcal{T})$ pair running in $\mathcal{T}$. In each set of 300 evaluation games, each agent plays as the first player in 150 games, and as the second player in the other 150 games.

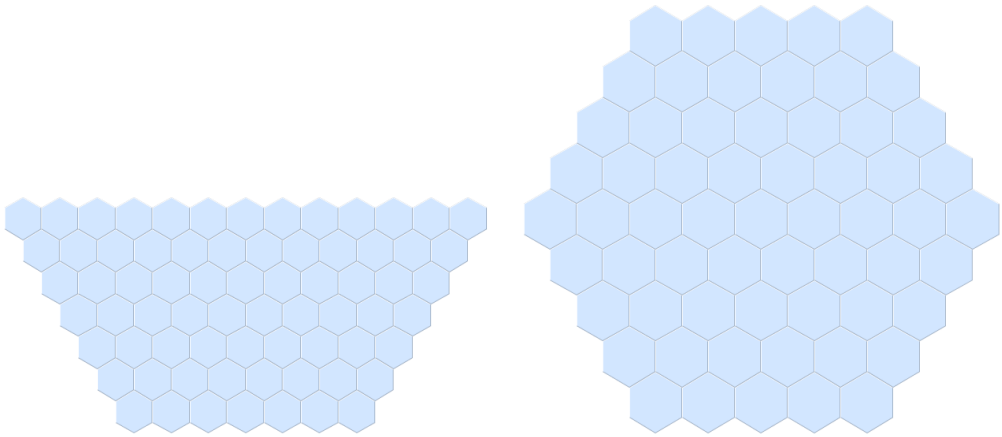
Figures 2(a) and 2(b) depict scatterplots of these zero-shot transfer evaluations for cases where $\mathcal{S}$ has a larger board size than $\mathcal{T}$, and where $\mathcal{S}$ has a smaller board size than $\mathcal{T}$, respectively. Win percentages of the transferred model against the baseline model are on the y-axis, and the ratio of the number of training epochs of the source model to the number of epochs of the target model is on the x-axis. Models trained on larger board sizes tend to have lower numbers of epochs for three reasons; (i) the neural network passes are more expensive, (ii) the game logic in Ludii is more expensive, and (iii) episodes often tend to last for a higher number of turns when played on larger boards. Hence, points in Figure 2(a) have a ratio ≤ 1.0, and



(a) $\mathcal{S}$ with larger board sizes than $\mathcal{T}$.

(b) $\mathcal{S}$ with smaller board sizes than $\mathcal{T}$.

Figure 2: Zero-shot win percentages of models trained on $\mathcal{S}$, against models trained on $\mathcal{T}$—evaluated on $\mathcal{T}$. For zero-shot transfer, any win percentage greater than 0% indicates some level of successful transfer, and results greater than 50% suggest training on $\mathcal{S}$ is better than training on $\mathcal{T}$—even for a subsequent evaluation on $\mathcal{T}$. "Num epochs $\mathcal{S}$" is the number of training epochs that fit within the training time budget for the model trained on $\mathcal{S}$ (the model being evaluated). "Num epochs $\mathcal{T}$" is the number of training epochs for the model trained on $\mathcal{T}$ (the opponent that we evaluate against).



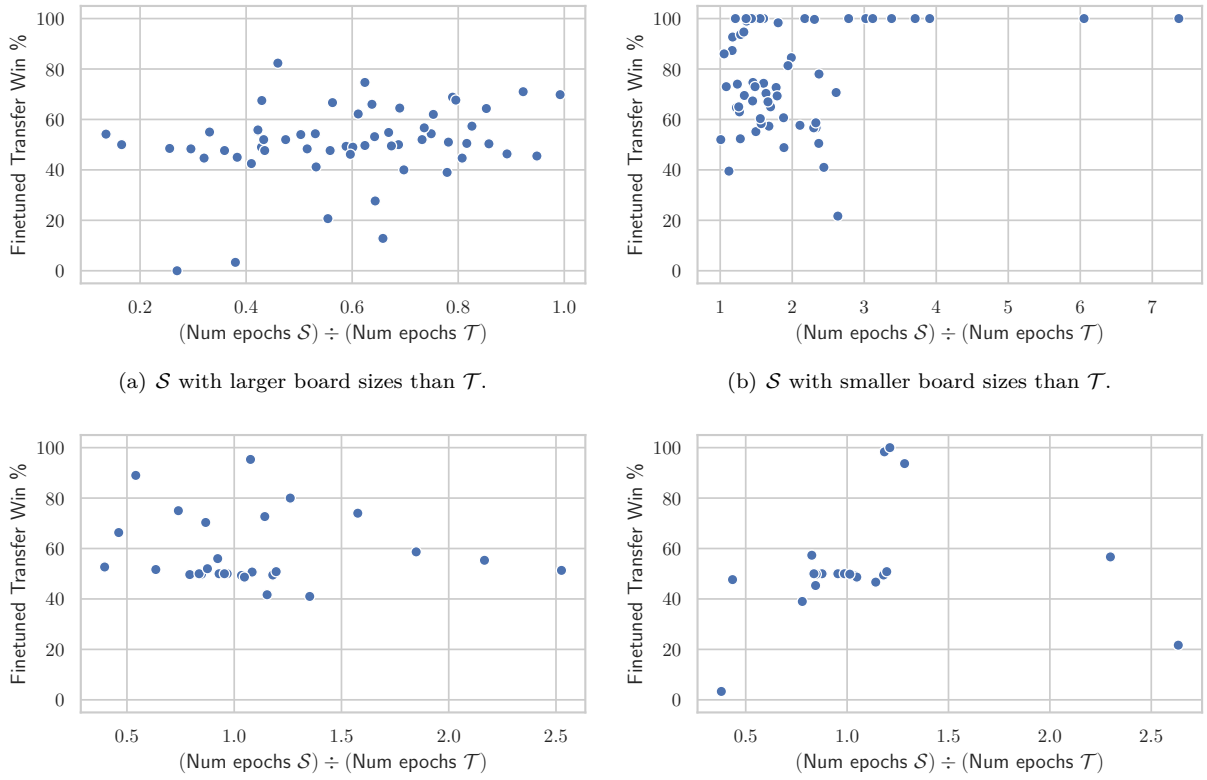
(a) The HalfHexHex board.

(b) The HexHex board.

Figure 3: Two different boards used for *Pentalath*.

points in Figure 2(b) have a ratio ≥ 1.0. Detailed tables with individual results for every $(\mathcal{S}, \mathcal{T})$ pair are listed in Appendix F.

For zero-shot transfer of a model trained on a large board to a small board (Figure 2(a)), win percentages tend to be below 50%, but frequently still above 0%; training on a larger board than the one we intend to play on does not outperform simply training on the correct board directly, but often still produces a capable model that can win a non-trivial number of games. There is a meaningful level of transfer in this direction, but (as would be expected) it is still better to use the same amount of computation time directly on the target domain. Transferring a model from a small board to a large board (Figure 2(b)) frequently leads to win percentages above 50%, even reaching up to 100%, against models trained directly in $\mathcal{T}$. Results above 50% are particularly interesting, since they suggest that computation time is spent more wisely by training on different (smaller) boards than the target domain, even in zero-shot settings.



(a) $\mathcal{S}$ with larger board sizes than $\mathcal{T}$.

(b) $\mathcal{S}$ with smaller board sizes than $\mathcal{T}$.

(c) Different board shapes for $\mathcal{S}$ and $\mathcal{T}$.

(d) Different win conditions for $\mathcal{S}$ and $\mathcal{T}$.

Figure 4: Win percentages of models trained on $\mathcal{S}$ and subsequently fine-tuned on $\mathcal{T}$, against models trained only on $\mathcal{T}$—evaluated on $\mathcal{T}$. Results above 50% may be interpreted as beneficial transfer—assuming that pre-training on $\mathcal{S}$ is "for free", without depleting any budget—whereas results below 50% indicate negative transfer.

Zero-shot transfer between variants with larger differences, such as changes in board shapes or win conditions, only leads to win percentages significantly above 0% in a few cases. For example, there is some meaningful zero-shot transfer with win percentages of 26.67% and 18.00% for transfer between two different board shapes in *Pentalath*. The different boards are depicted in Figure 3.

For every model transferred from a domain $\mathcal{S}$ to a domain $\mathcal{T}$ as described above, we train it under identical conditions as the initial training runs—for an additional 20 hours—to evaluate the effect of using transfer for initialisation of the network. Figure 4 depicts scatterplots of win percentages for four distinct cases; transfer to variants with smaller board sizes, with larger board sizes, with different board shapes, and with different win conditions. Detailed tables are provided in Appendix G. There are many cases of win percentages close to 50%, which can be interpreted as cases where transfer neither helped nor hurt final performance, and many cases with higher win percentages—which can be interpreted as cases where transfer increased final performance in comparison to training from scratch. We observe a small number of cases, especially when $\mathcal{S}$ has a larger board than $\mathcal{T}$ or has different win conditions, in which there is clear negative transfer (Zhang et al., 2020) and the final performance is still closer to 0% even after fine-tuning on $\mathcal{T}$. When there are substantial differences between source and target domains, a likely possible cause of any negative transfer (even after fine-tuning) is that a poor initial policy may be detrimental to exploration and the quality of collected experience, which in turn may prevent the training process from escaping the poor initialisation. The most drastic cases of negative transfer are a 0.00% winrate after transfer with fine-tuning from *Diagonal Hex* (19×19) to (11×11), and a 3.33% winrate for transfer from *Hex* (19×19) to *Misère Hex* (11×11). Interestingly, these are both cases of transfer from connection games on 19×19 hexagonal boards to 11×11

hexagonal boards, but transfer from the same source domains to larger as well as smaller target domains than 11×11 worked better. It is not yet clear why this particular scenario is more challenging, but exactly the same pattern in results emerged again in the repeated runs for ablations described in 4.4. This observation makes it unlikely that these results are simply statistical anomalies.

The experiments described above include two core settings; (i) zero-shot transfer, evaluating training for 20 hours on $\mathcal{S}$ and 0 on $\mathcal{T}$, against 20 hours on $\mathcal{T}$, and (ii) fine-tuning, evaluating training for 20 hours on $\mathcal{S}$ and 20 on $\mathcal{T}$, against just 20 hours on $\mathcal{T}$. A third setting that could be considered would be a variant of the fine-tuning setting, against a baseline with 40 hours of training on $\mathcal{T}$. However, in such a setting, there is no clear threshold that objectively separates successful transfer from unsuccessful transfer. Therefore, we prioritise dedicating our efforts to the two settings discussed above, which more readily allow for objective conclusions.

4.3 Transfer Between Different Games

Finally, we selected four sets of games that each share a single overarching theme, and within each set carried out similar experiments as described above—this time transferring models between distinct games, rather than minor variants of the same game. The first set consists of six different **Line Completion Games**; in each of these, the goal is to create a line of n pieces, but games differ in the value of n, board sizes and shapes, move rules, loss conditions, etc. We evaluate transfer from each of those games, to each of these games. The second set consists of four **Shogi Games**: *Hasami Shogi*, *Kyoto Shogi*, *Minishogi*, and *Shogi*, and evaluate transfer from and to each of them. The third set evaluates transfer from each of four variants of *Broken Line*, to each of the six line completion games. *Broken Line* is a custom-made line completion game where only diagonal lines count towards the win condition, whereas the standard line completion games allow for orthogonal lines. The fourth set evaluates transfer from each of five variants of *Diagonal Hex*, to each of six variants of *Hex*. *Diagonal Hex* only considers diagonal connections for its win condition, whereas the *Hex* variants only consider orthogonal connections. Appendix E provides more details on all the games and variants.

In most cases, zero-shot win percentages for models transferred between distinct games are close to 0%. We observe some success with win percentages over 30% for transfer from several line completion games to *Connect 6*, win percentages between 20% and 50% for transfer from three different Shogi games to *Hasami Shogi*, and a win percentage of 97% for transfer from *Minishogi* to *Shogi*. The last result is particularly significant, since it suggests that, when the ultimate goal is to train a *Shogi* agent, computation time is spent more efficiently training purely on the distinct game of *Minishogi* (at least for the first 20 hours). Appendix H contains more detailed results.

Figure 5 depicts win percentages for transferred models after they received 20 hours of fine-tuning time on $\mathcal{T}$. We observe many cases of succesful transfer between various line-completion games (including *Broken Line*), as well as several Shogi games. Most notably for various cases of transfer from *Diagonal Hex* to *Hex*, there is a high degree of negative transfer, with win percentages far below 50% even after fine-tuning. It may be that the differences in connectivity rules are too big to allow for consistently successful transfer. It may also be due to large differences in the distributions of outcomes, resulting in mismatches for the value head; ties are common in some variants of *Diagonal Hex*, but impossible in *Hex*.

4.4 Variations on Fine-tuning Experiments

For all fine-tuning experiments described throughout 4.2 and 4.3, we considered two additional variations on the experiments. The first is one where, after transfer but before fine-tuning, all the weights in the final layers are reinitialised. This can reduce the risk of remaining stuck in a poor local minimum in cases where transferred models perform poorly in the target domain. The second variation is one where, after transfer, randomly selected channels from hidden layers are removed, or new ones are added, to ensure that the transferred model has exactly as many parameters as the original model trained on $\mathcal{T}$ did. Without this additional step, models trained on distinct domains $\mathcal{S}$ and $\mathcal{T}$ sometimes have different numbers of channels in hidden layers give the default settings of Polygames. By default, it uses twice the number of channels in

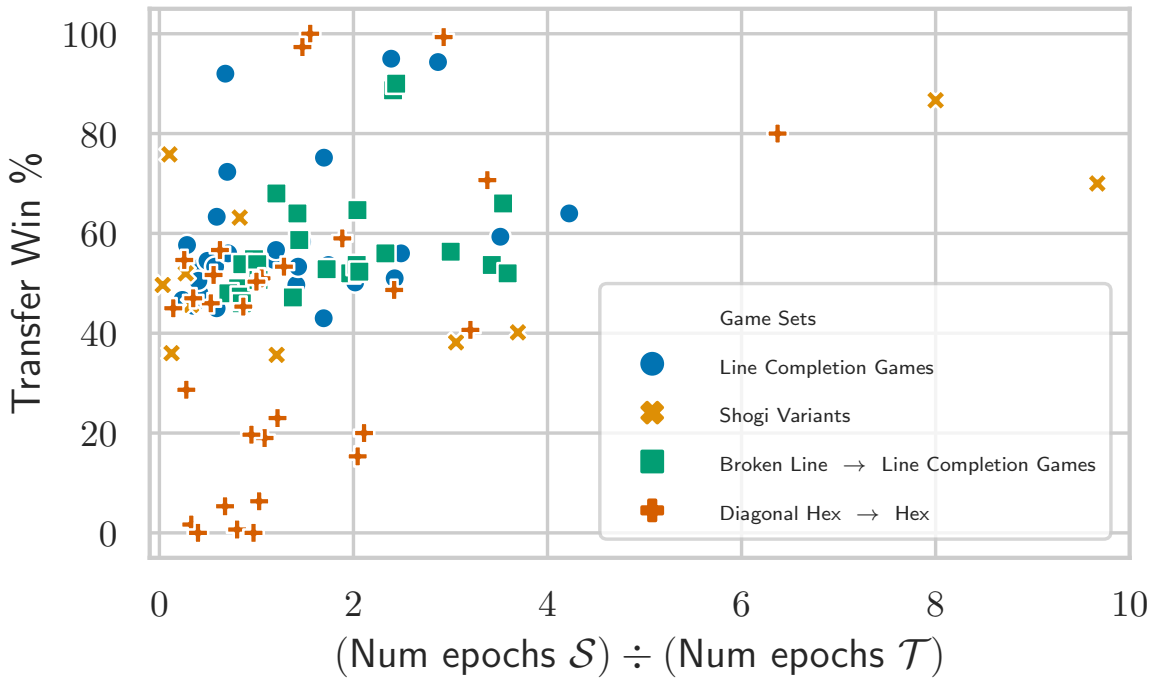


Figure 5: Win percentages of models that were trained in $\mathcal{S}$, transferred to $\mathcal{T}$, and fine-tuned in $\mathcal{T}$, evaluated in $\mathcal{T}$ against models trained directly in $\mathcal{T}$. $\mathcal{S}$ and $\mathcal{T}$ are different games.

the game's state tensor representation as the number of channels for hidden layers. Neither of these ablations resulted in any significant changes in results. Full results are included in Appendices G and I.

5 Related Work

ProcGen (Cobbe et al., 2020) is a framework in which levels are procedurally generated, which was proposed to benchmark generalisation in RL. However, its fixed action space across all supported domains means that it is unsuitable for evaluating transfer with changes in action spaces. Many other popular frameworks, such as ALE (Bellemare et al., 2013; Machado et al., 2018) and GVGAI (Perez-Liebana et al., 2019) are restricted to similarly small action spaces. MiniHack (Samvelyan et al., 2021) was proposed as a benchmark for evaluating several aspects of RL, including transfer in RL. It can be used to model domains with substantially larger action spaces than the other frameworks mentioned above, but is still limited to action spaces of at most 98 actions. This is significantly smaller than many board games supported by Ludii, which can easily have action spaces of many hundreds of elements. The action space of MiniHack is also relatively unstructured, with only a small subset of it being structured in the sense that they correspond to compass directions.

Like the benchmarks themselves, related work on algorithms and experiments for transfer in deep RL tends to be limited in terms of generality and variety of domains. Some work focuses only on transfer based on visual elements of states (Glatt et al., 2016; Mittel et al., 2018; Sobol et al., 2018; Gamrian & Goldberg, 2019; Zhu et al., 2020), which does not generalise beyond environments with pixel-based state representations such as Atari games. A substantial amount of work is furthermore limited to avatar-centric, single-agent domains with little variety in (often small) action spaces (Rusu et al., 2016; Tessler et al., 2017; Oh et al., 2017; Nichol et al., 2018; Glatt et al., 2020), transfer between domains with identical dynamics (Yang, 2021), or evaluations with only a small number of different games (Asawa et al., 2017). There is some work on learning action embeddings that can aid transfer between domains with different (but related) action spaces (Chen et al., 2021; You et al., 2022), but these approaches rely on experience in the target domain and cannot ever be expected to produce zero-shot results because they do not leverage descriptions of tasks. REvolveR is

highly specific robotic control settings, and assumes that it is possible to design continuous interpolations between source and target domains (Liu et al., 2022). Prior to the popularisation of deep learning in the last decade, some work was published on transfer between general games (Banerjee & Stone, 2007; Kuhlmann & Stone, 2007) described in the logic-based GDL from the Stanford Logic Group (Love et al., 2008). However, this work is not directly applicable to deep learning, and not as convenient and extensible as the significantly higher-level more modern GDLs. Gato (Reed et al., 2022) can perform a wide variety of tasks with a single network (i.e., a single set of trained weights). It likely performs some degree of implicit transfer between the various tasks it is trained for. However, it cannot perform zero-shot transfer to new tasks, as considered in this paper. Furthermore, it requires tasks to be described to it in the form of expert trajectories, which is a less general, convenient and practical manner of describing tasks than DSLs.

6 Conclusions

In this paper, we argue that research towards more generally applicable artificial intelligence should include models that are applicable to, and can generalise and transfer across, sets of problems that are expressed in high-level domain-specific languages (DSLs). Such sets of problems are easily extensible, and can be created and used by domain experts as end users, rather than programmers or machine learning experts. At the same time, DSLs also provide explicit knowledge of tasks which may facilitate zero-shot transfer. As an example benchmark, we consider Ludii, which contains a highly varied set of over 1000 distinct games expressed in a convenient, high-level game description language. We proposed a relatively simple baseline approach for transferring fully convolutional policy-value networks between pairs of games described in Ludii, and present empirical results for a wide variety of pairings of source and target domains. This includes transferring models to game variants with modified board sizes, board shapes, inverted win conditions, and different games altogether with more numerous and substantial differences in rules. We provided results for transfer with fine-tuning, but also zero-shot transfer. We identified some cases of negative transfer, but also many with successful transfer. One of the most noteworthy results is that training on a smaller version (e.g., smaller board) of a target game and transferring to a larger version in a zero-shot manner frequently outperforms training for the same amount of time on the actual target domain.

In the implementation of our proposed transfer approach, the identification of related channels between source and target domains is Ludii-specific, but the general design and philosophy behind the approach is generally applicable to any domain where large sets of problems under a common theme are described using a DSL. One avenue for future research would be to incorporate automated, data-driven learning techniques to guide the identification of shared semantics between source and target domains (Kuhlmann & Stone, 2007; Bou Ammar, 2013; Bou Ammar et al., 2014). As a more ambitious and distant future goal, we may consider working towards models that support tasks being described in natural languages, rather than DSLs.

Acknowledgments

Omitted for anonymity.

References

Chaitanya Asawa, Christopher Elamri, and David Pan. Using transfer learning between games to improve deep reinforcement learning performance. 2017.

C. Bamford, S. Huang, and S. Lucas. Griddly: A platform for AI research in games. In *AAAI-21 Workshop on Reinforcement Learning in Games*, 2021.

B. Banerjee and P. Stone. General game learning using knowledge transfer. In *The 20th International Joint Conference on Artificial Intelligence*, pp. 672–677, 2007.

Marc G. Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47(1):253–279, 2013.

H. Bou Ammar. *Automated transfer in reinforcement learning*. PhD thesis, Maastricht University, Maastricht, the Netherlands, 2013.

H. Bou Ammar, E. Eaton, M. E. Taylor, D. C. Mocanu, K. Driessens, G. Weiss, and K. Tuyls. An automated measure of MDP similarity for transfer in reinforcement learning. In *Proceedings of the Interactive Systems Workshop at the American Association of Artificial Intelligence (AAAI)*, pp. 31–37, 2014.

G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. OpenAI gym. https://arxiv.org/abs/1606.01540, 2016.

N. Brown and T. Sandholm. Superhuman AI for multiplayer poker. *Science*, 365(6456):885–890, 2019.

C. Browne, E. Powley, D. Whitehouse, S. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton. A survey of Monte Carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(1):1–49, 2012.

C. Browne, M. Stephenson, É. Piette, and D. J. N. J. Soemers. A practical introduction to the Ludii general game system. In T. Cazenave, J. van den Herik, A. Saffidine, and I.-C. Wu (eds.), *Advances in Computer Games. ACG 2019*, volume 12516 of *Lecture Notes in Computer Science (LNCS)*. Springer, Cham, 2020.

C. Browne, É. Piette, M. Stephenson, and D. J. N. J. Soemers. General board geometry. In C. Browne, A. Kishimoto, and J. Schaeffer (eds.), *Advances in Computer Games (ACG 2021)*, volume 13262 of *Lecture Notes in Computer Science*, pp. 235–246. Springer, Cham, 2022.

C. B. Browne. *Automatic Generation and Evaluation of Recombination Games*. Phd thesis, Faculty of Information Technology, Queensland University of Technology, Queensland, Australia, 2009.

T. Cazenave, Y.-C. Chen, G.W. Chen, S.-Y. Chen, X.-D. Chiu, J. Dehos, M. Elsa, Q. Gong, H. Hu, V. Khalidov, C.-L. Li, H.-I Lin, Yu-J. Lin, X. Martinet, V. Mella, J. Rapin, B. Roziere, G. Synnaeve, F. Teytaud, O. Teytaud, S.-C. Ye, Y.-J. Ye, S.-J. Yen, and S. Zagoruyko. Polygames: Improved zero learning. *ICGA Journal*, 42(4):244–256, 2020.

Y. Chen, Y. Chen, Z. Hu, T. Yang, C. Fan, Y. Yu, and J. Hao. Transfer with action embeddings for deep reinforcement learning. https://arxiv.org/abs/1909.02291v3, 2021.

K. Cobbe, O. Klimov, C. Hesse, T. Kim, and J. Schulman. Quantifying generalization in reinforcement learning. In K. Chaudhuri and R. Salakhutdinov (eds.), *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pp. 1282–1289. PMLR, 2019.

K. Cobbe, C. Hesse, J. Hilton, and J. Schulman. Leveraging procedural generation to benchmark reinforcement learning. In H. Daumé III and A. Singh (eds.), *Proceedings of the 37 International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 2048–2056, 2020.

R. Coulom. Efficient selectivity and backup operators in Monte-Carlo tree search. In H. J. van den Herik, P. Ciancarini, and H. H. L. M. Donkers (eds.), *Computers and Games*, volume 4630 of *Lecture Notes in Computer Science*, pp. 72–83. Springer Berlin Heidelberg, 2007.

Shani Gamrian and Yoav Goldberg. Transfer learning for related reinforcement learning tasks via image-to-image translation. In *Proceedings of the 36th International Conference on Machine Learning*, pp. 2063–2072, 2019.

R. Glatt, F. L. da Silva, and A. H. R. Costa. Towards knowledge transfer in deep reinforcement learning. In *Brazilian Conference on Intelligent Systems (BRACIS)*, pp. 91–96. IEEE, 2016.

R. Glatt, F. L. Da Silva, R. A. da Costa Bianchi, and A. H. R. Costa. Decaf: Deep case-based policy inference for knowledge transfer in reinforcement learning. *Expert Systems with Applications*, 156, 2020.

K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778. IEEE, 2016.

S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In F. Bach and D. Blei (eds.), *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pp. 448–456, 2015.

M. Jiang, E. Grefenstette, and T. Rocktäschel. Prioritized level replay. In M. Meila and T. Zhang (eds.), *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pp. 4940–4950. PMLR, 2021.

N. Justesen, R. R. Torrado, P. Bontrager, A. Khalifa, J. Togelius, and S. Risi. Illuminating generalization in deep reinforcement learning through procedural level generation. 2018.

L. Kocsis and C. Szepesvári. Bandit based Monte-Carlo planning. In J. Fürnkranz, T. Scheffer, and M. Spiliopoulou (eds.), *Machine Learning: ECML 2006*, volume 4212 of *Lecture Notes in Computer Science*, pp. 282–293. Springer, Berlin, Heidelberg, 2006.

G. Kuhlmann and P. Stone. Graph-based domain mapping for transfer learning in general games. In J.N. Kok, J. Koronacki, R.L. Mantaras, S. Matwin, D. Mladenič, and A. Skowron (eds.), *Machine Learning: ECML 2007*, volume 4071 of *Lecture Notes in Computer Science (LNCS)*, pp. 188–200. Springer, Berlin, Heidelberg, 2007.

H. Küttler, N. Nardelli, A. Miller, R. Raileanu, M. Selvatici, E. Grefenstette, and T. Rocktäschel. The nethack learning environment. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 7671–7684. Curran Associates, Inc., 2020.

A. Lazaric. Transfer in reinforcement learning: a framework and a survey. In M. Wiering and M. van Otterlo (eds.), *Reinforcement Learning*, volume 12 of *Adaptation, Learning, and Optimization*, pp. 143–173. Springer, Berlin, Heidelberg, 2012.

X. Liu, D. Pathak, and K. M. Kitani. REvolveR: Continuous evolutionary models for robot-to-robot policy transfer. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *PMLR*, pp. 13995–14007, 2022.

N. Love, T. Hinrichs, D. Haley, E. Schkufza, and M. Genesereth. General game playing: Game description language specification, 2008.

M. C. Machado, M. G. Bellemare, E. Talvitie, J. Veness, M. Hausknecht, and M. Bowling. Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents. *Journal of Artificial Intelligence Research*, 61:523–562, 2018.

G. Marcus. Innateness, alphazero, and artificial intelligence. *CoRR*, abs/1801.05667, 2018. URL https://arxiv.org/abs/1801.05667.

Akshita Mittel, Sowmya Munukutla, and Himanshi Yadav. Visual transfer between atari games using competitive reinforcement learning, 2018.

V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518:529–533, 2015.

V. Nair and G. Hinton. Rectified linear units improve restricted boltzmann machines. In J. Fürnkranz and T. Joachims (eds.), *Proceedings of the 27th International Conference on Machine Learning*, pp. 807–814. Omnipress, 2010.

Alex Nichol, Vicki Pfau, Christopher Hesse, Oleg Klimov, and John Schulman. Gotta learn fast: A new benchmark for generalization in rl. *CoRR*, 2018. URL https://arxiv.org/abs/1804.03720.

J. Oh, S. Singh, H. Lee, and P. Kohli. Zero-shot task generalization with multi-task deep reinforcement learning. In *Proceedings of the International Conference on Machine Learning (ICML)*, pp. 2661–2670. PMLR, 2017.

J. Parker-Holder, M. Jiang, M. Dennis, M. Samvelyan, J. Foerster, E. Grefenstette, and T. Rocktäschel. Evolving curricula with regret-based environment design. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *PMLR*, pp. 17473–17498, 2022.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (eds.), *Advances in Neural Information Processing Systems 32*, pp. 8024–8035. Curran Associates, Inc., 2019.

D. Perez-Liebana, A. Dockhorn, J. H. Grueso, and D. Jeurissen. The design of "Stratega": A general strategy games framework. In J. C. Osborn (ed.), *Joint Proceedings of the AIIDE 2020 Workshops co-located with 16th AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE 2020)*. CEUR Workshop Proceedings, 2020.

Diego Perez-Liebana, Jialin Liu, Ahmed Khalifa, Raluca D. Gaina, Julian Togelius, and Simon M. Lucas. General video game AI: A multitrack framework for evaluating agents, games, and content generation algorithms. *IEEE Transactions on Games*, 11(3):195–214, 2019.

É. Piette, D. J. N. J. Soemers, M. Stephenson, C. F. Sironi, M. H. M. Winands, and C. Browne. Ludii – the ludemic general game system. In G. De Giacomo, A. Catala, B. Dilkina, M. Milano, S. Barro, A. Bugarín, and J. Lang (eds.), *Proceedings of the 24th European Conference on Artificial Intelligence (ECAI 2020)*, volume 325 of *Frontiers in Artificial Intelligence and Applications*, pp. 411–418. IOS Press, 2020.

É. Piette, C. Browne, and D. J. N. J. Soemers. Ludii game logic guide. https://arxiv.org/abs/2101.02120, 2021.

S. Reed, K. Żołna, E. Parisotto, S. G. Colmenarejo, A. Novikov, G. Barth-Maron, M. Giménez, Y. Sulsky, J. Kay, J. T. Springenberg, T. Eccles, J. Bruce, A. Razavi, A. Edwards, N. Heess, Y. Chen, R. Hadsell, O. Vinyals, M. Bordbar, and N. de Freitas. A generalist agent. https://arxiv.org/abs/2205.06175, 2022.

Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In Nassir Navab, Joachim Hornegger, William M. Wells, and Alejandro F. Frangi (eds.), *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, pp. 234–241, Cham, 2015.

A. A. Rusu, N. C. Rabinowitz, G. Desjardins, H. Soyer, J. Kirkpatrick, K. Kavukcuoglu, R. Pascanu, and R. Hadsell. Progressive neural networks. https://arxiv.org/abs/1606.04671, 2016.

M. Samvelyan, R. Kirk, V. Kurin, J. Parker-Holder, M. Jiang, E. Hambro, F. Petroni, H. Küttler, E. Grefenstette, and T. Rocktäschel. Minihack the planet: A sandbox for open-ended reinforcement learning research. In *Advances in Neural Information Processing Systems*, 2021.

W. Schafhauser. *TEMPLE - A Domain Specific Language for Modeling and Solving Real-Life Staff Scheduling Problems*. Phd thesis, Fakultät für Informatik, Technischen Universität Wien, Vienna, Austria, 2010.

T. Schaul. A video game description language for model-based or interactive learning. In *Proceedings of the IEEE Conference on Computational Intelligence in Games*, pp. 193–200. IEEE, 2013.

J. Schrittwieser, I. Antonoglou, T. Hubert, K. Simonyan, L. Sifre, S. Schmitt, A. Guez, E. Lockhart, D. Hassabis, T. Graepel, T. Lillicrap, and D. Silver. Mastering Atari, Go, Chess and Shogi by planning with a learned model. *Nature*, 588:604–609, 2020.

E. Shelhamer, J. Long, and T. Darrell. Fully convolutional networks for semantic segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(4):640–651, 2017.

D. Silver, A. Huang, C. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. van den Driessche, T. Graepel, and D. Hassabis. Mastering the game of Go without human knowledge. *Nature*, 550:354–359, 2017.

D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan, and D. Hassabis. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science*, 362(6419):1140–1144, 2018.

C. F. Sironi and M. H. M. Winands. Adaptive general search framework for games and beyond. In *2021 IEEE Conference on Games*, pp. 1051–1058. IEEE, 2021.

Doron Sobol, Lior Wolf, and Yaniv Taigman. Visual analogies between atari games for studying transfer learning in rl, 2018.

D. J. N. J. Soemers, V. Mella, C. Browne, and O. Teytaud. Deep learning for general game playing with Ludii and Polygames. *ICGA Journal*, 43(3):146–161, 2022.

R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 2 edition, 2018.

M. E. Taylor and P. Stone. Transfer learning for reinforcement learning domains: A survey. In S. Mahadevan (ed.), *Journal of Machine Learning Research*, volume 10, pp. 1633–1685, 2009.

C. Tessler, S. Givony, T. Zahavy, D. Mankowitz, and S. Mannor. A deep hierarchical approach to lifelong learning in minecraft. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 1553–1561. AAAI, 2017.

David J. Wu. Accelerating self-play learning in Go. https://arxiv.org/abs/1902.10565v3, 2019.

Kaige Yang. Learn dynamic-aware state embedding for transfer learning, 2021.

H. You, T. Yang, Y. Zheng, J. Hao, and M. E. Taylor. Cross-domain adaptive transfer reinforcement learning based on state-action correspondence. In *Proceedings of the Thirty-Eighth Conference on Uncertainty in Artificial Intelligence (UAI 2022)*, volume 180 of *PMLR*, pp. 2299–2309, 2022.

C. Zhang, O. Vinyals, R. Munos, and S. Bengio. A study on overfitting in deep reinforcement learning. https://arxiv.org/abs/1804.06893, 2018.

K. Zhang and D. Shasha. Simple fast algorithms for the editing distance between trees and related problems. In *SIAM Journal on Computing*, volume 18, pp. 1245–1262, 1989.

W. Zhang, L. Deng, L. Zhang, and D. Wu. Overcoming negative transfer: A survey. *CoRR*, abs/2009.00909, 2020. URL https://arxiv.org/abs/2009.00909.

Z. Zhu, K. Lin, and J. Zhou. Transfer learning in deep reinforcement learning: A survey. *CoRR*, abs/2009.07888, 2020. URL https://arxiv.org/abs/2009.07888.