



# Machine Discovery of Comprehensible Strategies for Simple Games Using Meta-interpretive Learning

Stephen H. Muggleton<sup>1</sup> · Celine Hocquette<sup>1</sup>

Received: 28 February 2019 / Accepted: 19 March 2019 / Published online: 25 April 2019  
© The Author(s) 2019

## Abstract

Recently, world-class human players have been outperformed in a number of complex two-person games (Go, Chess, Checkers) by Deep Reinforcement Learning systems. However, the data efficiency of the learning systems is unclear given that they appear to require far more training games to achieve such performance than any human player might experience in a lifetime. In addition, the resulting learned strategies are not in a form which can be communicated to human players. This contrasts to earlier research in *Behavioural Cloning* in which single-agent skills were machine learned in a symbolic language, facilitating their being taught to human beings. In this paper, we consider Machine Discovery of human-comprehensible strategies for simple two-person games (Noughts-and-Crosses and Hexapawn). One advantage of considering simple games is that there is a tractable approach to calculating minimax regret. We use these games to compare Cumulative Minimax Regret for variants of both standard and deep reinforcement learning against two variants of a new Meta-interpretive Learning system called *MIGO*. In our experiments, tested variants of both normal and deep reinforcement learning have consistently worse performance (higher cumulative minimax regret) than both variants of *MIGO* on Noughts-and-Crosses and Hexapawn. In addition, *MIGO*'s learned rules are relatively easy to comprehend, and are demonstrated to achieve significant transfer learning in both directions between Noughts-and-Crosses and Hexapawn.

**Keywords** Inductive Logic Programming · Reinforcement learning · Games strategies

---

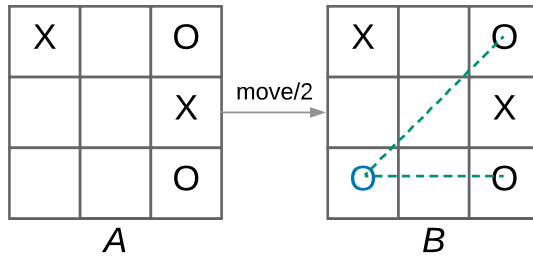
Stephen H. Muggleton acknowledges support from the EPSRC Human-Like Computing grant to the Department of Computing at Imperial College London. Celine Hocquette thanks the EPSRC for her Ph.D. studentship.

---

✉ Stephen H. Muggleton  
s.muggleton@imperial.ac.uk

Celine Hocquette  
celine.hocquette16@imperial.ac.uk

<sup>1</sup> Department of Computing, Imperial College London, 180 Queens Gate, London, UK



```

win_2 (A, B) :- win_2_1_1 (A, B), not (win_2_1_1 (B, C)) .
win_2_1_1 (A, B) :- move (A, B), not (win_1 (B, C)) .
win_1 (A, B) :- move (A, B), won (B) .

```

**Fig. 1** Noughts-and-Crosses: example of optimal move for O from board A to board B. For all moves of X from board B, O can win in one move. This statement can be expressed with the logic program presented: O makes a move, such that X cannot immediately win nor make a move that blocks O

## Introduction

Deep Reinforcement Learning systems have been demonstrated capable of mastering two-player games such as Go [25], outperforming the strongest human players. However, these systems (1) generally require a very large training set to converge toward a good strategy, (2) are not easily interpretable as they provide limited explanation about how decisions are made, and (3) do not provide transferability of the learned strategies to other games.

In this paper, we are interested in automated discovery of playing strategies for two-person games, with the aim of communicating the learned strategies to human players. In this way, our work follows in the tradition of Behavioural Cloning [3, 11, 15].

We demonstrate how machine learning strategies as logic programs not only produce comprehensible theories, but also learn accurate strategies using substantially smaller training sets than deep reinforcement learning.

As an example, consider the Noughts-and-Crosses positions shown in Fig. 1. In such positions, an applicable strategy is to create double attacks where possible. Player O executes a move from board A to board B which creates the two threats represented in green, and results in a forced win for O. The rules in Fig. 1 describe such a strategy. A, B, and C are variables representing state descriptions which encode the board together with the active player. The rules state that a move by the active player from A to B is a winning move if the opponent cannot immediately win and the opponent cannot make a move to prevent an immediate win by the active player. These rules provide an understandable strategy for winning in two moves. Moreover, these rules are transferable to more complex games as they are generally true for describing double attacks.

We introduce, in this article, a new logical system called *MIGO* (*Meta-Interpretive Game Ordinator*)<sup>1</sup> designed for learning two-player game optimal strategies of the form presented in Fig. 1. It benefits from a strong inductive bias which provides the capability to learn efficiently from a few examples of games played. Learned hypotheses are provided in a symbolic form, which allows their interpretation. Moreover, learned strategies are generally true for all two-player games, which provides straightforward transferability to more complex games.

*MIGO* uses Meta-interpretive Learning (MIL), a recently developed Inductive Logic Programming (ILP) framework that supports predicate invention, the learning of recursive programs [20, 21], and Abstraction [5]. *MIGO* additionally supports Dependent Learning [13]. The learning operates in a staged fashion: simple definitions are learned first and added to the background knowledge [13], allowing them to be reused during further learning tasks, and, thus, to build up more and more complex definitions. For instance, *MIGO* would first learn a simple definition of *win\_1/2* for winning in one move. Next, a predicate *win\_2/2* describing the action of winning in two moves can be built from *win\_1/2* as shown in Fig. 1.

To evaluate performance, we consider two evaluable games (Noughts-and-Crosses and Hexapawn). Our results demonstrate that substantially lower Cumulative Minimax Regret can be achieved by *MIGO* compared to variants of reinforcement learning.

Our contributions are the introduction of a system for learning optimal two-player-game strategies (Sect. 3) and the description of its implementation (Sect. 4). We demonstrate experimentally that it converges faster than reinforcement learning systems and that learned strategies are transferable to more complex games (Sect. 5).

## Related Work

### Learning Game Strategies

Various early approaches to game strategies [12, 24] used the decision tree learner ID3 to classify minimax depth-of-win for positions in chess end games. These approaches used a set of carefully selected board attributes as features. Conversely, *MIGO* is provided with a set of three relational primitives (*move/2*, *won/1*, *drawn/1*) representing the minimal information which a human would expect to know before playing a two-person game.

More recently, Transfer Learning of heuristic pattern concepts has been demonstrated for simple games such as Tic-tac-toe and Connect4 [23]. Unlike *MIGO*, this approach does not learn to play from scratch, but, instead, uses an alpha–beta player with a heuristic function based on set of specialised concepts learned using the ILP system ALEPH [17].

---

<sup>1</sup> From the children's game-playing phrase *My go!* and the literal translation into English of the French word *Ordinateur* which means computer.

## Reinforcement Learning

Reinforcement Learning considers the task of identifying an optimal agent policy to maximise the cumulative reward perceived by an agent. MENACE (Matchbox Educa-ble Noughts-And-Crosses Engine) [14] was the world's earliest reinforcement learning system and was specifically designed to learn to play Noughts-and-Crosses. An early manual version of MENACE used a stack of matchboxes, one for each accessible board position. Each box contained coloured beads representing possible moves. Moves were selected by randomly drawing a bead from the current box. After having completed a game, MENACE's punishment or reward consisted of subtracting or adding beads according to the outcome of the game. This modified the probability of the selected move being played in the position [4]. HER (Hexapawn Educational Robot) [9] is a similar system for the game of Hexapawn.

More generally,  $Q$ -learning [26] addresses the problem of learning an optimal policy from delayed rewards by trial and error. The learned policy takes the form of  $Q$  values for each actions available from a state. A guarantee of asymptotic convergence to optimal behaviour has been proved [27].

Deep  $Q$ -learning [16] is an extension that uses a deep convolutional neural network to approximate the different  $Q$  values for each actions given a particular state. It provides better scalability which has been demonstrated through a diverse range of tasks from the Atari 2600 games. However, this framework generally requires the execution of many games to converge. Moreover, the learned strategy is implicitly encoded into the  $Q$ -value parameters, which do not provide interpretability. In [10], a hybrid neural-symbolic system is described which address some of these drawbacks. A neural back-end transforms images into a symbolic representation and generates features. A symbolic front end performs action selection. Conversely, *MIGO* is based upon a purely symbolic approach and the number of primitives considered is reduced.

## Relational Reinforcement Learning

Relational reinforcement learning [8] is a reinforcement learning framework where states, actions, and policies are represented relationally. It benefits from background knowledge and declarative bias. It learns a  $Q$  function using a relational regression tree algorithm. Conversely, the learning framework *MIGO* is not based on the identification of  $Q$  values, but aims at deriving hypotheses describing an optimal strategy. Relational reinforcement learning also provides the ability to carry over the policies learned in simple domains to more complex situations. However, most systems aim at learning single-agent policy and, in contrast to *MIGO*, are not designed to learn to play two-person games.

## Theoretical Framework

### Credit Assignment

One can evaluate the success of a game by looking at its outcome. However, a problem arises for assigning the reward to the various moves performed. Reinforcement learning systems usually tackle this so-called Credit Assignment Problem by adjusting parameter values associated with the moves responsible for the reward observed. We introduce theorems for identifying moves that are necessarily positive examples for the task of winning and drawing.

We assume that the learner  $P_1$  plays against an opponent  $P_2$  that follows an optimal strategy and that the game starts from a randomly chosen initial board  $B$ . We consider the following ordering over the different outcomes for  $P_1$  and demonstrate the lemma below:

$$\text{won} > \text{drawn} > \text{loss}$$

**Lemma 1** *The expected outcome of  $P_1$  can only decrease during a game.*

**Proof**  $P_2$  plays optimally and, therefore, any move of  $P_2$  maintains or lowers the expected outcome. Therefore  $P_1$  cannot increase its outcome.  $\square$

We demonstrate the Theorems below given these assumptions and Lemma 1:

**Theorem 1** *If the outcome is won for  $P_1$ , then every move of  $P_1$  is a positive example for the task of winning.*

**Proof** Suppose that there exists a move of  $P_1$  from the board  $B_1$  to the board  $B_2$  within the game sequence that is a negative example for the task of winning. Then, the expected outcome of  $B_1$  is won and the expected outcome of  $B_2$  is strictly lower with respect to the order  $>$ . Then, following Lemma 1, the outcome of the game is strictly lower than won, which leads to contradiction with the outcome observed.  $\square$

**Theorem 2** *We additionally assume an accurate strategy  $S_W$  for winning has been learned by the learner  $P_1$ . If the outcome of the game is drawn and if the execution of  $S_W$  from  $B$  fails, then any move played by  $P_1$  or  $P_2$  is a positive example for the task of drawing.*

**Proof** The initial position does not have an expected outcome of won for  $P_2$ ; otherwise, the outcome would be won for  $P_2$ , since it plays optimally. The initial position is not an expected outcome of win for  $P_1$  by assumption. Therefore, the expected outcome of  $B$  is drawn. It follows from Lemma 1 that every position reached during the game has an expected outcome of drawn and that every move of both players is a positive example for the task of drawing.  $\square$

Theorems 1 and 2 demonstrate, that for an outcome of win for  $P_1$  or drawn and because the opponent plays optimally, the expected outcome is necessarily maintained as won or drawn respectively. This cannot be further generalised to  $P_2$ 's moves: an outcome of won for  $P_2$  might be the consequence of a mistake of  $P_1$  who does not play optimally.

One should also highlight the fact that Theorems 1 and 2 do not provide any negative examples for *win/2* or *draw/2*, as these theorems do not help to evaluate moves for which the expected outcome decreases. Practically, the learning system considered learns from positive examples only.

## Game Evaluation

Given Theorems 1 and 2, the opponent chosen is an optimal player following the minimax algorithm. Both for Noughts-and-Crosses and Hexapawn, and more generally for most fair two-player games, the opponent can always ensure a draw from the initial board, which leaves no opportunities for the learner to win. To ensure possibilities of winning, we start the game from a board randomly sampled from the set of one move-ahead accessible boards; this set provides different expected outcomes for the games considered. Then, the actual outcome relies on both the initial board and the sequence of moves performed. We define the minimax regret as follows:

**Definition 1** (*Minimax Regret*) The *minimax regret* of a game is the difference between the minimax expected outcome of the initial board and the actual outcome of the game.

Practically, the minimax expected outcome of a board can be evaluated from a minimax database computed beforehand. Definition 3.4 provides an absolute measure to evaluate the performances of a learning algorithm as it does not rely on the choice of initial board. Thereafter, we evaluate the cumulative minimax regret to compare different learning systems.

## Meta-Interpretive Learning (MIL)

The system *MIGO* introduced in this work is an MIL system. MIL is a form of ILP [19, 20]. The learner is given a set of examples  $E$  and background knowledge  $B$  composed of a set of Prolog definitions  $B_p$  and metarules  $M$ , such that  $B = B_p \cup M$ . The aim is to generate a hypothesis  $H$ , such that  $B, H \models E$ . The proof is based on an adapted Prolog meta-interpreter. It, first, attempts to prove the examples considered deductively. Failing this, it unifies the head of a metarule with the goal, and saves the resulting meta-substitution. The body and then the other examples are similarly proved. The meta-substitutions recovered for each successful proofs are saved and can be used in further proofs by substituting them into their corresponding metarules. Key features of MIL are that it supports predicate invention, the learning of recursive programs, and Abstraction [5]. In the following, we use the MIL system *Metagol* [6].

## MIGO Algorithm

We present within this section details of the MIGO algorithm.

*Learning from positive examples* Theorems 1 and 2 provide a way of assigning positive labels to moves. Therefore, the learning is based on positive examples only. This is possible because of Metagol's strong language bias and ability to generalise from a few examples only. However, one pitfall is the risk of over-generalisation due to the absence of negative examples.

*Dependent Learning* For successive values of  $k$ , a series of inter-related definitions are learned for predicates  $\text{win\_}k(A, B)$  and  $\text{draw\_}k(A, B)$ . These predicates define maintenance of minimax win and draw in  $k$ -ply when moving from position  $A$  to  $B$ . The learning algorithm is presented as Algorithm 1, each action 'learn' represents a call to Metagol. This approach is related to Dependent Learning [13]. The idea is to first learn low-level predicates. They are derived from single examples with limited complexity. The definitions are added into the background knowledge, such that they can be used in further definitions. The process iterates until no further predicates can be learned.

---

### Algorithm 1 MIGO Algorithm

---

**Input:** Positive examples for  $\text{win\_}k$  and  $\text{draw\_}k$

**Output:** Strategy for  $\text{win\_}k$  and  $\text{draw\_}k$

```

1: for  $k$  in  $[1, \text{Depth}]$  do
2:   for each example of  $\text{win\_}k/2$  do
3:     one shot learn a rule and add it to the BK
4:   end for
5:   Learn  $\text{win\_}k/2$  and add it to the BK
6: end for
7: for  $k$  in  $[1, \text{Depth}]$  do
8:   for each example of  $\text{draw\_}k/2$  do
9:     one shot learn a rule and add it to the BK
10:  end for
11:  Learn  $\text{draw\_}k/2$  and add it to the BK
12: end for

```

---

*Mixed Learning and Separated Learning* Theorem 2 assigns positive labels to  $\text{draw}/2$  examples assuming a winning strategy  $S_W$  has already been learned. In practice, we distinguish two variants of *MIGO*:

1. Separated Learning:  $\text{win}/2$  and  $\text{draw}/2$  are learned in two stages.  $\text{Win}/2$  is first learned. When a strategy for  $\text{win}/2$  is stable for a given number of iterations, the learner starts learning  $\text{draw}/2$ .
2. Mixed Learning:  $\text{win}/2$  and  $\text{draw}/2$  are learned simultaneously. Examples of  $\text{draw}/2$  are first evaluated with the current strategy for  $\text{win}/2$ . If this latter is updated, examples of  $\text{draw}/2$  are re-tested against the updated version of  $\text{win}/2$ .

**Table 1** Metarules considered: the letters P, Q, and R denote existentially quantified higher order variables. The letters A, B, and C denote universally quantified first-order variables

Name	Metarule
<i>postcond</i>	$P(A, B) \leftarrow Q(A, B), R(B).$
<i>negation</i>	$P(A, B) \leftarrow Q(A, B), \text{not}(R(B, C)).$

## Implementation

### Representation

A board  $B$  is encoded as a 9-vector of marks from the set  $\{O, X, \text{Empty}\}$ . States  $s(B, M)$  are atoms that represent the current board  $B$  and the active player  $M$ .

### Primitives and Metarules

The language belongs to the language class  $H_2^2$ , which is the subset of Datalog logic programs with predicates of arity at most 2 and at most 2 literals in the body of each clause. Learned programs are formed of dyadic predicates, representing actions, and monadic predicates, representing fluents. The background knowledge contains a general move generator *move/2*, which is an action that modifies a state  $s(B, M)$  by executing a move on board  $B$  and updating the active player  $M$ . *Move/2* only holds for valid moves; in other words, the learner already knows the rules of the game. The background knowledge also contains two fluents: a won classifier *won/1* and a drawn classifier *drawn/1*. They hold when a board is respectively won or drawn.

We consider the metarules *postcond* and *negation* described in Table 1. The metarule *negation* expresses the logical negation for primitive predicates, and is implemented as negation as failure. This form of Negation does not introduce invented predicates in Metagol.

### Execution of the Strategy

For each rule learned for *win<sub>i</sub>* and *draw<sub>i</sub>* a clause of the form below is added to the background knowledge:

```
win(A,B)  :- win_i(A,B).
draw(A,B) :- draw_i(A,B).
```

When executing a strategy described with a hypothesis  $H$ , the move performed is the first one consistent with  $H$ . Practically, it first attempts to prove *win<sub>i</sub>/2* for increasing values for  $i$ . Failing that, it attempts to prove *draw<sub>i</sub>/2* for increasing values for  $i$ . If these proofs fail, a move is selected at random among the possible moves.



The opponent plays a deterministic minimax strategy that yields the best outcome in the minimum number of moves.

## Learning a Strategy

At the end of a game, the outcome is observed and the sequence of visited boards is divided into moves. The depth of each board is measured as the number of full moves until the end of the game in the observed sequence. Moves are added to the set of positive examples for  $win_k/2$  or  $draw_k/2$  if they satisfy Theorems 1 or 2. Strategies are relearned from scratch after each game using the MIGO algorithm presented above. One additional constraint is added, such that  $draw/2$  cannot be learned before  $win/2$ , since this would cause the learner to always draw and never win.

## Experiments

### Experimental Hypothesis

This section describes experiments which evaluate the performance of *MIGO* for the task of learning optimal two-player game strategies.<sup>2</sup> We use the games of Noughts-and-Crosses and a variant of the game of Hexapawn [9]. *MIGO* is compared against the reinforcement learning systems MENACE/HER, Q-learning, and Deep Q-learning. Accordingly, we investigate the following null hypotheses:

**Null Hypothesis 1:** *MIGO cannot converge faster than MENACE/HER, Q-learning, and Deep Q-learning for learning optimal two-player game strategies.*

We additionally test the ability of *MIGO* to transfer learned strategies to more complex games, and, thus, verify the following null hypothesis:

**Null Hypothesis 2:** *MIGO cannot transfer the knowledge learned during a previous task to a more complex game.*

### Convergence

### Materials and Methods

*Common* We provide *MIGO*, Menace/HER, and Q-learning with the same set of initial boards randomly sampled from the set of one-full-move-ahead positions—positions that result from one move of each player. The systems studied play games starting from these initial boards, and they face the same deterministic minimax player. Therefore, the only variable in the experiments is the learning system. It is assumed that the learner always starts the game. The performance is evaluated in terms of cumulative minimax regret (Fig. 2) and CPU time (Table 2).

<sup>2</sup> Code for these experiments available at <https://github.com/migo19/migo.git>.

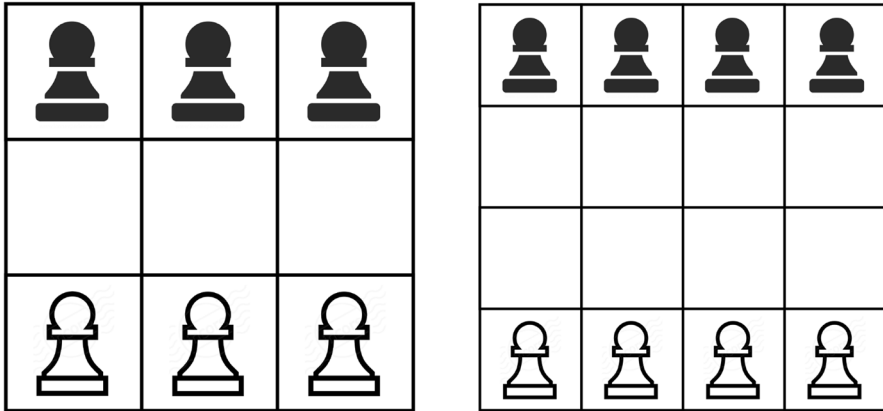


Fig. 2 Initial boards for Hexapawn<sub>3</sub> and Hexapawn<sub>4</sub>

Table 2 Average CPU time (s) of one iteration

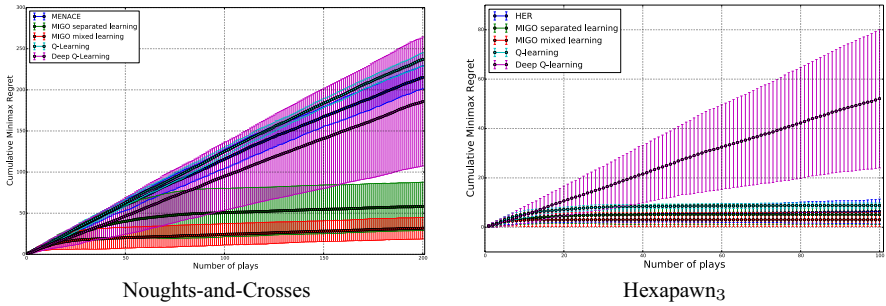
	OX	Hexapawn <sub>3</sub>	Hexapawn <sub>4</sub>
MIGO mixed learning	$1.5 \cdot 10^{-1}$	$3.0 \cdot 10^{-3}$	3.9
MIGO separated learning	$8.9 \cdot 10^{-2}$	$2.8 \cdot 10^{-3}$	3.8
MENACE / HER	$1.5 \cdot 10^{-3}$	$2.7 \cdot 10^{-4}$	/
Q-learning	$2.3 \cdot 10^{-1}$	$1.9 \cdot 10^{-3}$	$2.7 \cdot 10^{-1}$
Deep Q-learning	$2.4 \cdot 10^{-1}$	$1.7 \cdot 10^{-2}$	$2.1 \cdot 10^{-1}$

We follow an implementation of Tabular Q-learning available from [1] and used the parameter values which were provided for the Q-learning algorithm: the exploration rate is set to 0; the initial  $q$  values are 1; the discount factor is  $\gamma = 0.9$  and the learning rate  $\alpha = 0.3$ . Similarly, we follow an implementation of Deep Q-learning available from [2]. The provided parameters were used: the discount factor is set to 0.8; the regularization strength to 0.01 and the target network update rate to 0.01; the initial and final exploration rate are 0.6 and 0.1 respectively.

The results presented here have been averaged over 40 runs for Hexapawn<sub>3</sub> and 20 for Noughts and Crosses. Average running times are presented in Fig. 2.

*Noughts-and-Crosses* The set of initial boards comprises 12 boards taking into account rotations and symmetries of the board. Among them, 7 are expected win, and 5 are expected draw. Therefore, the worst case regret of a random player is 1.58. The counter for starting learning  $draw/2$  is set to 10.

*Hexapawn* Hexapawn’s initial board is represented in Fig. 2. The goal of each player is to advance one of their pawns to the opposite end. Pawns can move one square forward if the next square is empty or capture another pawn one square diagonally ahead of it [9]. Rules have been modified: the game is said to be drawn when the current player has no legal move. Thereafter, we refer to Hexapawn<sub>3</sub> and Hexapawn<sub>4</sub> for the game of Hexapawn in dimensions 3 by 3 and 4 by 4, respectively. The set of initial boards comprises 5 boards taking into account the vertical



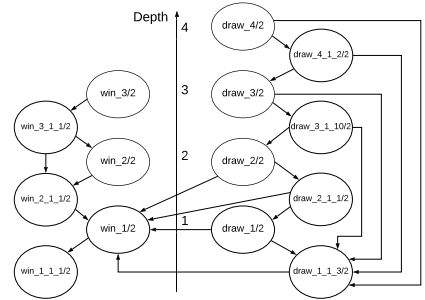
**Fig. 3** Cumulative Minimax Regret versus the number of iterations for Noughts-and-Crosses and Hexapawn<sub>3</sub>

**Fig. 4** Rules learned: **a** Noughts-and-Crosses (all) and Hexapawn<sub>3</sub> (above the double line). **b** Calling diagram of Learned Strategies

Depth	Rule
1	win.1(A, B) :- win.1.l.1(A, B), won(B).
	win.1.l.1(A, B) :- move(A, B), won(B).
2	draw.1(A, B) :- draw.1.l.3(A, B), not(win.1(B, C)).
	draw.1.l.3(A, B) :- move(A, B), not(win.1(B, C)).
3	win.2(A, B) :- win.2.l.1(A, B), not(win.2.l.1(B, C)).
	win.2.l.1(A, B) :- move(A, B), not(win.1(B, C)).
4	draw.2(A, B) :- draw.2.l.1(A, B), not(win.1(B, C)).
	draw.2.l.1(A, B) :- draw.1(A, B), not(win.1(B, C)).

(a)

(b)



symmetry. Among them, 3 are expected draw and 2 are expected win. Therefore, the average worst case regret is 1.4. As the dimensions are smaller for Hexapawn<sub>3</sub> than for Noughts and Crosses, the counter for starting learning *draw/2* is set to 5.

**Results**

Results are presented in Fig. 3 and show that *MIGO* converges faster than *MENCE/HER*, *Q-learning*, and *Deep Q-learning* for both games, refuting null hypothesis 1. As the maximum depth is larger for Noughts-and-Crosses than for Hexapawn<sub>3</sub>, all systems require more iterations to converge. *Deep Q-learning* performs worst for Hexapawn<sub>3</sub> as the parameters selected are the ones tuned for Noughts and Crosses and might not be adapted. For both games, mixed learning has lower cumulative regret than separated learning, because mixed learning does

not waste any examples of *draw/2* from the initial period in which *win/2* is being learned and it does not stop learning *win/2* after the initial period.

Rules learned by *MIGO* are presented in Fig. 4a. *MIGO* converges toward this full set of rules when playing Noughts-and-Crosses. Because the maximum depth of Hexapawn<sub>3</sub> is 2, *MIGO* learns up to the double line when playing Hexapawn<sub>3</sub>. If unfolding, the first rule can be translated into English as: *State A is won at depth 1 if there exists a move from A to B, such that B is won.* Similarly, winning at depth 2 can be described with the following statement: *State A is won at depth 2 if there exists a move of the current player from A to B, such that B is not immediately won for the opponent and such that the opponent cannot make a move from B to C to prevent the current player from immediately winning.* This statement is similar to the one presented in Sect. 1. Finally, winning at depth 3 can be explained as: *State A is won at depth 3 for the current player if there exists a move from A to B, such that B is not won for the opponent in 1 or 2 moves and such that the opponent cannot make a move from B to C to prevent the current player from winning in 1 or 2 moves.* None of the other systems studied can provide similar explanation about the moves chosen. Rules are built on top on each other; the calling diagram in Fig. 4b represents the dependencies between each learned predicates.

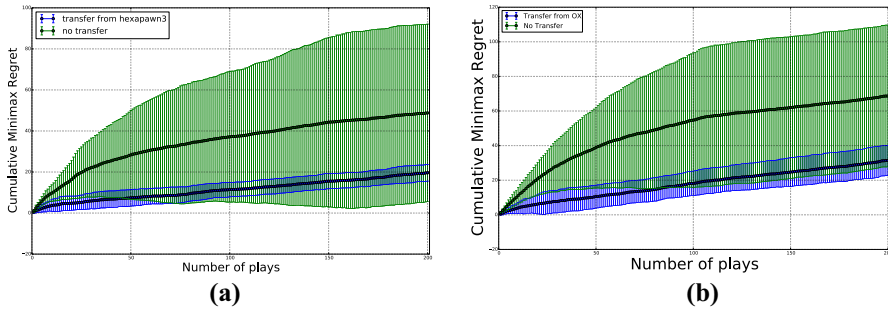
## Discussion

MENACE, HER, and Q-learning encode the knowledge into the parameters (number of beads or Q values). The states and their parameters are unique for each board. This results in a weaker generalisation ability: knowledge cannot be transferred from one state to another. Deep Q-learning can provide some generalisation ability; however, it is only visible after a large number of iterations. Conversely, *MIGO* generalises the boards characteristics and each rule learned describes a set of states, which considerably reduces the number of parameters to learn and, therefore, the number of examples required.

The reinforcement learning systems tested have an implicit representation of the problem. For instance, no geometrical concepts have been encoded. Conversely, *MIGO* benefits from a background knowledge which describes the notion of winning, and from which it can extract a notion of alignment. This allows a degree of explanation.

The running time increases rapidly with the state dimensions for *MIGO*. This reflects the increasing execution time of the learned strategy which is not efficient, since a deep evaluation requires extensive evaluation to decide whether a move leads to a win.

MENACE/HER is specifically tailored for these games. Conversely, Q-learning and Deep Q-learning are general approaches that can tackle a wide range of tasks, providing that parameters are tuned. *MIGO* benefits from underlying assumptions which reduce its range of applications. However, the primitives are abstract enough to allow playing a wide range of games and support transferring knowledge from one game to another as we will demonstrate in the next section.



**Fig. 5** Transfer learning: **a** Hexapawn<sub>3</sub> to Noughts and Crosses, **b** Noughts and Crosses to Hexapawn<sub>4</sub>. Similar results are obtained from Hexapawn<sub>3</sub> to Hexapawn<sub>4</sub>

## Transferability

*Materials and Methods* Strategies are first learned for Hexapawn<sub>3</sub> and Noughts-and-Crosses respectively. Strategies are learned with mixed learning and for 100 iterations for Hexapawn<sub>3</sub> and 200 iterations for Noughts and Crosses. The resulting learned program is transferred to the next learning task, which is learning a strategy for Hexapawn<sub>4</sub>. Results have been averaged over 20 runs.

*Results* The results presented in Fig. 5 show that transferring the knowledge learned in a previous task help to converge faster, thus refuting null hypothesis 2. Since the learner benefits from an initial knowledge, it is substantially improved compared to an initial random player.

## Conclusion and Future Work

This article introduces a novel logical system named *MIGO* for learning two-player game strategies. It is based on the MIL framework. This system distinguishes itself from classical reinforcement learning by the way that it addresses the Credit Assignment Problem. Our experiments have demonstrated that *MIGO* achieves lower Cumulative Minimax Regret compared to Deep and classical Q-Learning. Moreover, we have demonstrated that strategies learned with *MIGO* are transferable to more complex games. Strategies have also been shown to be relatively easy to comprehend.

*Future Work* One limitation of the system presented is the risk of over-generalisation, observable in the strategy learned. We will further extend the implementation to include a more-thorough context for learning from positive examples such as the one presented in [18].

The running time suggests that the execution time of the learned strategies increases with the dimensions of the states, which limits scalability. We will further extend *MIGO* to optimise the execution time for hypothesised programs. Selection of hypotheses could be performed following the idea described in [7].

Another limitation to scalability is the restriction imposed by the initial assumptions. The current version of *MIGO* requires an optimal opponent, which is intractable in large dimensions. We will further extend this system by relaxing Theorems 1 and 2 and weakening the optimal opponent assumption. A solution could be to learn from self-play.

Because *MIGO* benefits from a strong declarative bias, the sample complexity is much improved compared to the other approaches. However, most of the examples are wasted as no labels could be attributed. We plan to evaluate whether Active Learning could further help to reduce the sample complexity. The learner could choose an initial board to start the game, the choice being based on an information gain criterion.

Although learned strategies provide a certain form of explanation, we will further study how comprehensible learned strategies are. We will evaluate whether *MIGO* can fulfill Michie's Machine Learning Ultra Strong criterion, which requires the learner to be able to teach the learned hypothesis to a human [22].

Despite these limitations, we believe that the novel system introduced in this work opens exciting new avenues for machine learning game strategies.

**Open Access** This article is distributed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits unrestricted use, distribution, and reproduction in any medium, provided you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license, and indicate if changes were made.

## References

1. Q-learning tic-tac-toe. <https://gist.github.com/fheisler/430e70fa249ba30e707f> (2015)
2. Solving tic-tac-toe using deep reinforcement learning. <https://github.com/yanji84/tic-tac-toe-rl> (2016)
3. Bain, M., Sammut, C.: A framework for behavioural cloning. In: Furukawa, K., Michie, D., Muggleton, S. (eds.) Machine Intelligence 15: Intelligent Agents. Oxford University Press, Oxford (1999)
4. Brooks, R.: FoR & AI: Machine learning explained (2017). <https://rodnebrooks.com/forai-machine-learning-explained/>
5. Cropper, A., Muggleton, S.: Learning higher-order logic programs through abstraction and invention. In: IJCAI 2016, pp. 1418–1424 (2016). <http://www.ijcai.org/Abstract/16/204>
6. Cropper, A., Muggleton, S.: Metagol system. <https://github.com/metagol/metagol> (2016)
7. Cropper, A., Muggleton, S.: Learning efficient logic programs. Mach. Learn. (2018). <https://doi.org/10.1007/s10994-018-5712-6>
8. Džeroski, S., Raedt, L.D., Driessens, K.: Relational reinforcement learning. Mach. Learn. **43**(1), 7–52 (2001). <https://doi.org/10.1023/A:1007694015589>
9. Gardner, M.: Mathematical games. The Unexpected Hanging and Other Mathematical Diversions (1962)
10. Garnelo, M., Arulkumaran, K., Shanahan, M.: Towards deep symbolic reinforcement learning. CoRR abs/1609.05518 (2016). [arxiv:1609.05518](https://arxiv.org/abs/1609.05518)
11. Inoue, K., Furukawa, K., Kobayashi, I., Nabeshima, H.: Discovering rules by meta-level abduction. In: Proceedings of the 19th international conference on Inductive logic programming, ILP'09, pp. 49–64 (2010)
12. John Quinlan, J.R.: Learning Efficient Classification Procedures and Their Application to Chess End Games, pp. 463–482. Springer, Berlin (1983). [https://doi.org/10.1007/978-3-662-12405-5\\_15](https://doi.org/10.1007/978-3-662-12405-5_15)

13. Lin, D., Dechter, E., Ellis, K., Tenenbaum, J., Muggleton, S.: Bias reformulation for one-shot function induction. In: Proceedings of the 23rd European Conference on Artificial Intelligence (ECAI 2014), pp. 525–530. IOS Press (2014)
14. Michie, D.: Experiments on the mechanization of game-learning part I. Characterization of the model and its parameters. *Comput. J.* **6**(3), 232–236 (1963)
15. Michie, D., Sammut, C.: Behavioural clones and cognitive skill models. In: Furukawa, K., Michie, D., Muggleton, S. (eds.) *Machine Intelligence 14: Applied Machine Intelligence*. Oxford University Press, Oxford (1995)
16. Mnih, V., Kavukcuoglu, K., Silver, D., et al.: Human-level control through deep reinforcement learning. *Nature* **518**, 529–533 (2015)
17. Muggleton, S.: Inverse entailment and Progol. *New Gen. Comput.* **13**, 245–286 (1995). <http://www.doc.ic.ac.uk/~shm/Papers/InvEnt.pdf>
18. Muggleton, S.: Learning from positive data. In: Muggleton, S.H. editor, *Proceedings of the 6th International Workshop on Inductive Logic Programming (Workshop-96)*, LNAI 1314, Springer, New York, pp. 358–376 (1996)
19. Muggleton, S., Lin, D.: Meta-interpretive learning of higher-order dyadic datalog: predicate invention revisited. In: *Proceedings of the 23rd international joint conference artificial intelligence*, pp. 1551–1557 (2013)
20. Muggleton, S., Lin, D., Pahlavi, N., Tamaddoni-Nezhad, A.: Meta-interpretive learning: application to grammatical inference. *Mach. Learn.* **94**, 25–49 (2014)
21. Muggleton, S., Lin, D., Tamaddoni-Nezhad, A.: Meta-interpretive learning of higher-order dyadic datalog: predicate invention revisited. *Mach. Learn.* **100**(1), 49–73 (2015). <https://doi.org/10.1007/s10994-014-5471-y>
22. Muggleton, S., Schmid, U., Zeller, C., Tamaddoni-Nezhad, A., Besold, T.: Ultra-strong machine learning: comprehensibility of programs learned with ilp. *Mach. Learn.* **107**(7), 1119–1140 (2018). <https://doi.org/10.1007/s10994-018-5707-3>
23. Sato, Y., Iida, H., van den Herik, H.: Transfer learning by inductive logic programming. In: *14th International Conference on Advances in Computer Games*, LNCS 9525, pp. 223–234. Springer, Berlin (2015)
24. Shapiro, A., Niblett, T.: Automatic induction of classification rules for a chess endgame. In: Clarke, M. (ed.) *Advances in Computer Chess*, vol. 3, pp. 73–91. Pergamon, Oxford (1982)
25. Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., Lillicrap, T., Simonyan, K., Hassabis, D.: A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science* **362**, 1140–1144 (2018)
26. Watkins, C.: *Learning from Delayed Rewards*. PhD thesis (1989)
27. Watkins, C., Dayan, P.: Q-learning. *Mach. Learn.* **8**(3), 279–292 (1992)

**Publisher's Note** Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.