Reinforcement Learning in Games

GLines

Hélène Dintinger
ID: 260014115
CSID: hdinti
# Table of Contents

I - Introduction ........................................................................................................ 1

II - The Game: GLines ............................................................................................ 2
   A A BRIEF PRESENTATION .................................................................................. 2
   B WHY USE REINFORCEMENT LEARNING? ....................................................... 2

III - Other games studies review .......................................................................... 3
   A TD- GAMMON BY TESAURO .......................................................................... 3
   B OTHELLO ........................................................................................................... 4
   C CHESS ................................................................................................................ 6
   D GO ...................................................................................................................... 7
   E NEURAL NET & TICKTACKTOE ...................................................................... 8
   F NEURONAL EVALUATION FUNCTION ........................................................... 10

IV - Methodology .................................................................................................... 11
   A FIST IDEA ......................................................................................................... 11
   B GLINES COMPLEXITY PROBLEM .................................................................. 11
   C THEORETICAL STUDIES ................................................................................ 13

V - Conclusion ...................................................................................................... 16
The objective of this project was to create an agent available to compute high score in Glines game. I tough that, as it involves chance, previslon and strategy, this game fits well the reinforcement leaning process. But That not totally true as we will see in the paper.

After a brief presentation of the GLines game and it’s complexity, we will studied the use in reinforcement learning in games though some research paper on: Backgammon, Chess, Othello, Go and Ticktacktoe game. As we will see the use of reinforcement learning and game is very often coupled with neural network as a function evaluation.

At last we will discover GLines specific characteristics though some experiment and theoretical studies.
II - The Game: GLines

A A brief presentation

Glines is the Gnome port of the once popular Windows game called Color Lines. The game's objective is to align as often as possible five balls or more of the same color causing them to disappear, play as long as possible, and have the highest Score. At each ball move three new balls (color pre-announced) are drop on the board.

The board of the GLines games is 9x9.
The number of different color is 7.

B Why use reinforcement learning?

The interactive nature of reinforcement learning is particularly appealing for game learning. What is learned through interaction with the environment is an optimal policy mapping states to actions maximizing the total reward obtained. Compared to other learning paradigms, reinforcement learning has some nice properties for game learning: no expert knowledge is needed and it is incremental, that is continuous learning against a variety of opponents is possible.

The basic paradigm of reinforcement learning is as follows: The learning agent observes an input state or input pattern, it produces an output signal (most commonly thought of as an "action" or "control signal"), and then it receives a scalar "reward" or "reinforcement" feedback signal from the environment indicating how good or bad its output was. The goal of learning is to generate the optimal actions leading to maximal reward.

One problem has been that, in the case of reinforcement learning with delay, the temporal credit assignment aspect of the problem has remained extremely difficult. Another problem with many of the traditional approaches to reinforcement learning is that they have been limited to learning either lookup tables or linear evaluation functions, neither of which seems adequate for handling.

However, reinforcement learning application to deterministic games like chess, Go, and Othello, is considered to be more difficult than RL application to stochastic games like Backgammon or Glines.

In this study, the simplicity of the time taken for one game of Glines is advantageous, because the computer players are able to play many training games. More over the fact that the game is a unique player game is also an advantage as no opponent training method need to be trained.
III - Other games studies review

In this part I present some research paper on reinforcement learning applied to game in order have correct basis to apply to the GLines games.


[2] Taku Yoshioka, Shin Ishii and Minoru Ito. Strategy Acquisition for the Game Othello Based on Reinforcement Learning


As all these application use a neural network as an evaluation function, I also read two articles on Neural Network and Game.


A TD- Gammon by Tesauro

This is the main article as other article tends to comparer them selves with it.

This article presents a game-learning program called TD-Gammon. TD-Gammon is a neural network that trains itself to be an evaluation function for the game of backgammon by playing against itself and learning from the outcome. Comparing TD-Gammon with Neurogammon (a previous research of Tesauro), one can get a sense of the potential of TD learning relative to the more established approach of supervised learning.

Here the brute-force methodology of deep searches, which has worked so well in games such as chess, checkers and Othello, is not feasible due to the high branching ratio resulting from the probabilistic dice rolls.
The supervised training approach of Neurogammon is relies on human expertise. Here human expertise was build into an evaluation function, whether by knowledge engineering or by supervised training. Theses methods has been found to be difficult and with many potential pitfalls.

Some strategies (like the use of the doubling cube) were not included in TD-Gammon's training. Instead, a doubling algorithm was added after training that makes decisions by feeding TD-Gammon's expected reward estimates into a theoretical doubling formula.

TD-Gammon represents a radically different approach toward developing a program capable of sophisticated positional judgment. Rather than trying to imitate humans, TD-Gammon develops its own sense of positional judgment by learning from experience in playing against itself. While it may seem that forgoing the tutelage of human masters places TD-Gammon at a disadvantage, it is also liberating in the sense that the program is not hindered by human biases or prejudices that may be erroneous or unreliable. Indeed the result of TD-Gammon's self-training process is an incredibly sophisticated evaluation function, which, in at least some cases, appears to surpass the positional judgment of world-class human players.

At the heart of TD-Gammon is a neural network, which is organized in a standard multi-layer perception (MLP) architecture. The training procedure for TD-Gammon is as follows: the network observes a sequence of board positions starting at the opening position and ending in a terminal position characterized by one side having removed all its checkers. The board positions are fed as input vectors to the neural network. For each input pattern there is a neural network output vector indicating the neural network's estimate of expected outcome. At each time step, the TD (lambda) algorithm is applied to change the network's weights.

This paper is a reference for Reinforcement learning and introduces my idea of the need of a MLP function evaluation for the GLines problem. The comparison between the strength of backgammon game and Glines game in term of successfulness for such a method is given later.

### Othello

This article discusses automatic strategy acquisition for the game “Othello” based on a reinforcement-learning scheme. In each game, the computer player refines the evaluation function for the game state, which is achieved by min-max reinforcement learning (MMRL). MMRL is a simple learning scheme that uses the min-max strategy.
MMRL tends to minimize the difference between the present evaluation value and the prediction value based on the min-max search. It does not employ TD-error and is an off-policy learning scheme.

In their approach, the computer player does not conduct any deep search. Instead, it selects a move based on a 2-ply min-max search using its evaluation function. For this approach to work well, the evaluation function should be good. Here the optimal evaluation function predicts the number of discs finally won by the player, for a given board state.

The computer player then tends to learn evaluation values for limited states. In order to avoid this, in the learning phase, stochastic noise is added to the move selection process.

In the evaluation function each game state is represented by a vector \( s \), whose component is 01 (black disc), 0 (no disc), or 1 (white disc). Then, the number of possible states becomes roughly \( 3^{64} \). Since this number is huge, it is difficult to represent the evaluation function by a look-up table, that is why instead employ a normalized Gaussian network (NGnet) to approximate the evaluation function. Therefore board states that are apart from each other are often assigned a similar evaluation value: Considering the symmetry of the board, that four discs are initially placed on the board, and the states that are not reachable from the initial state, etc., the number of possible states becomes smaller than this value.

It seems difficult to represent the true evaluation function of Othello, because the game does not involve stochastic characteristics and the board state often changes significantly even after a single move. Moreover, a minor variation on the board can cause a significant difference of the evaluation function. In the conventional RL application to games, a linear function or the MLP was often used as a function approximator.

The input to the NGnet is the game state and the output is a prediction of the difference between the black and white discs on the final board expected to be reached from the input board state.

Evaluation In the experiment, the trained player is evaluated by playing with a player that takes a heuristic strategy. Learning scheme Learning is carried out in three different ways. Two learning players: The black and white sides employ their own NGnets. They are each trained by MMRL by playing a lot of games against each other.

The state space of Othello is huge, while the states do not uniformly appear. It is considered that by relocating the unit centers, the NGnet can approximate the evaluation function mainly for those regions where the states frequently occur. Consequently, the network can successfully approximate the evaluation function over the huge state space using only 100 units.
Therefore, it is considered that our approach can be applied to many other similar games. If such a game is more complex than Othello, more training games will be required to obtain a good evaluation function. In this case, it is important to employ a faster learning algorithm than the gradient descent algorithm, e.g., the EM algorithm.

As a result, the computer player becomes strong enough to beat a player employing a heuristic strategy.

This article experimentally shows that MMRL is better than TD (0) and also shows that the NGnet is better than a multi-layered perceptron, in our Othello task.

This article makes me understand the limit of the backgammon learning process as a minor variation on the board can cause a significant difference of the evaluation function also for the Glines problem.

### C Chess

This article describes a chess program that learns by combining TD (lambda) with game-tree search.

In this paper the authors present TDLeaf(lambda), a variation of the TD(lambda) algorithm that enables it to be used in conjunction with game tree search. They present some experiments in which a chess program used TDLeaf(lambda) to learn its evaluation function while playing on the Free Internet Chess Server.

Temporal Difference learning is a technique for approximating the expected long-term future cost of a stochastic dynamical system as a function of the current state. A parameterized function approximator such as a neural network implements the mapping from states to future cost. The parameters are updated online after each state transition, or possibly in batch updates after several state transitions. Finding such representation for Chess, Othello or Go, which allows a small neural network to order, moves at one-ply with near human performance is a difficult task.

It seems that for these games, reliable tactical evaluation is difficult to achieve without deep look ahead. As deep look ahead invariably involves some kind of minimax search, which in turn requires an exponential increase in the number of positions evaluated as the search depth increases, the computational cost of the evaluation function has to be low, ruling out the use of expensive evaluation functions such as neural networks.

Consequently most Chess and Othello programs use linear evaluation functions (the branching factor in Go makes minimax search to any significant depth nearly infeasible).

TDLeaf(lambda) algorithm is used to learn an evaluation function for use in deep minimax search. TDLeaf(lambda) is identical to TD(lambda) except that instead of operating on the positions that occur during the game, it operates on the leaf nodes
of the principal variation of a minimax search from each position (also known as the principal leaves).

The TD(lambda) algorithm was the strategy used in TD-Gammon. Unfortunately, for games like Othello and Chess it is very difficult to accurately evaluate a position by looking only one move or ply ahead. Most programs for these games employ some form of minimax search. In minimax search, one builds a tree from position $x$ by examining all possible moves for the computer in that position, then all possible moves for the opponent, and then all possible moves for the computer and so on to some predetermined depth $d$. The leaf nodes of the tree are then evaluated using a heuristic evaluation function, and the resulting scores are propagated back up the tree by choosing at each stage the move, which leads to the best position for the player on the move.

Then there is some experiments of chess evaluation function training using TDLeaf(lambda) by on-line play against a mixture of human and computer opponents. The experiments show both the importance of “on-line” sampling (as opposed to self play) for a deterministic game such as Chess, and the need to start near a good solution for faster convergence.

This TDLeaf algorithm is very interesting as the idea to combine 2 AI-game algorithms open a lot of ways of resolution.

DGO


This paper examine the Learning Vector Quantization (LVQ) function approximator for reinforcement learning methods, for a large decision search space, defined in terms of different classes of input patterns, like that found in the game of Go.

From a reinforcement learning point of view, Go is a finite-horizon sequential-task problem with discrete states. Because of the large state space, it is helpful to view the game with continuous states using the influence value propagation of the stones.

Self-Organization and LVQ Self-organization involves the ability to learn and organize sensory information without the benefit of a teacher. Learning is driven by measures of fitness, possible evolved over time. If the task to be learned is that of clustering, one example of such a fitness measure is that of similarity. The process of self-organization consists of iteratively modifying synaptic weights in response to sensory patterns until an optimal configuration, according to some closeness measure, eventually develops.

The LVQ algorithm is a supervised clustering method in which each output unit represents a particular class or category. The weight vector for an output unit is often referred to as a reference or codebook vector for the class that the unit represents. It is
also assumed that a set of training patterns with known class labels is provided, along with an initial distribution ("seed") of reference vectors. After training, the neural net classifies an input vector by assigning it to the same class as the output unit that has its weight vector closest to the input vector. After learning, the probability density function of the input is approximated by the modified set of discrete decoders or codebook vectors. The distributed representation of LVQ into codebook vectors as generalization of the input patterns significantly reduces the state space requirements and has a close correspondence to a tabular representation of state-action pairs.

SLVQ integrates Sarsa with LVQ, the estimation of the utility function Q is tied to the pattern recognition task of the situation.

Matching of a board configuration against a weight vector is done here by using the fuzzy contrast model. This distance is not a metric distance but takes into account the presence or absence of certain stones in a pattern. This characteristic makes it extremely well suited for the recognition of the vital stones in a game.

At the end of the paper there is some experimental data to show the feasibility of SLVQ Go and its comparative merits to other potential Go players.

The idea I retain here was the pattern recognition function, indeed in the GLines problem, the program has to be able to detect aligned structure.

E Neural net & Ticktacktoe


After having a look on the two neural network introduced in this paper I specifically studied the ticktacktoe part of this paper because, I thought it was a good way to begin the studies of neural networks.

E.1 NEURAL NETWORK

Designing an multiplayer perceptron (MLP) neural network requires the researcher to define a number of parameters such as the number of hidden layers, number of units in each layer, squashing function, learning rate, and momentum coefficient. In order to establish these, we have to experiment with different network parameter settings for learning three small tasks:

- exclusive-or This is the classical test of an MLP's ability to learn a nonlinear function.
- 2-D Euclidean distance from origin This task tests the network's ability to approximate an analog function accurately.
- Bounded random walk this problem, described in tests the ability of a network to solve a linear prediction problem by the method of temporal differences.
The back propagation algorithm updated the network weights after each training pattern, rather than only once per epoch. A useful suggestion is the addition of the constant 0.1 was added to the sigmoid derivative in order to keep the hidden units from getting prematurely “stuck” near 0 or 1.

**Design of Modular Architectures**

The author creates own modular architecture network. Called Designer Domain Decomposition (DDD) network. The DDD network consists of a collection of N monolithic sub networks and a hard-coded gating function, written by the designer, which partitions the domain space into N classes.

The DDD net allows the designer to use his or her domain knowledge in specifying a useful decomposition. The operation of the DDD net is extremely simple: in both forward and backward propagation, the gating program is called to select an appropriate sub network, which is then evaluated or trained as a monolithic net. Exactly one sub network is active at any time. Thus, assuming that the gating program can classify an input pattern in a negligible amount of time, the DDD net runs as quickly as a monolithic network the size of just one of its sub networks.

**E.2 TICKTACKTOE**

The combination of neural networks and temporal difference learning are applied to the game of ticktacktoe. Ticktacktoe is a conceptually simple game with a small state space (on the order of $10^4$ reachable positions), yet the optimal static position evaluator must be a complex non-linear function of the input position in order to recognize such features as the potential for forking plays.

Since ticktacktoe is deterministic, a program that is learning by self play may become stuck in a local minimum where it plays the same game repeatedly to a draw but has failed to explore large regions of the input space, the self play training procedure must involve some non-determinism.

An alternative approach is to train the network on games played against a high quality opponent algorithm. This algorithm would be able to "teach" the network by leading it into new regions of the game position space. This is the method the author used to train his ticktacktoe networks.

Two ticktacktoe evaluator networks were trained: a monolithic network and a modular network.

The monolithic network was a standard MLP with 18 binary input units, 20 hidden units, and a single output unit.

The modular network was a DDD net with 8 sub networks, each having the same 18-20-1 architecture as the monolithic network.

The monolithic network learned more quickly at first than its modular counterpart, no doubt because each modular sub network saw only 9%-14% as many training
patterns per epoch as the monolithic net did. However, the performance of the modular network soon surpassed that of the monolithic.

This article allows me to understand better how I could use neural network in order to achieve the function evaluation of the GLines Board.

F Neuronal Evaluation function

[6] Called Use of neural network as evaluation function of a game

This paper focuses on the problem of creating a good evaluation function, and principally a neural network function evaluation. After a little introduction on neural network, the game studied in this paper is the pushover game (king of Abalone game).

The great things with this paper is the different approach to the game problem: One based on database of game configuration were each combination is qualify as good or bad. The other taking into account the characteristic of the position (in a line or not...) The last kind of experiment having bad result the authors acknowledge the fact that the characteristics chosen weren't maybe the best one, but personally I tend to think of the problem of human biased method as explained in the backgammon paper.

The idea of simplifying the network entry seduces me but as the result didn’t follow I doubt of the efficacy of such methods.
IV - Methodology

A  Fist idea

I have first decided to follow an experimental strategy based on a Glines game that can be played by an agent.

The java GLines Game I created is inspired by the C GLines game by Robert Szokovacs and Szabolcs Ban. I use the same score grid, and their alignment detection algorithm as mine didn’t perform well.

When the game is over, the agent receives the reward, if as we see in the games description earlier it is typically ``1'' for a win, ``0'' for a draw and ``-1'' for a loss. I change that for the GLines reward. I didn’t want to bias the algorithm with my own judgment of what is a good strategy therefore I just give the score as a reward at the end of the game.

My Idea was then to use the TD algorithms we saw in class and that we have programmed for the assignment 2. But the problem arises when I tried to model the available state action pairs. As here are an enormous number of possible states, the complexity of the Glines game is very high:

B  GLines complexity problem

The GLines board is 9*9 it means 81 positions.
On each position it could be a ball or not. The ball could have 7 colors; that make 8 possibly per fields, that is $7^{81}$ combinations for the board (approximately $10^{68}$).
Furthermore, using the brute-force methodology of deep searches, which worked in games such as chess, checkers and Othello, is not feasible here due to the high branching ratio resulting from the random choose and placement of ball on the board at each time.

Apart from the board the player have the colors of the tree next balls that will appeared randomly on the board, this add $7^3$ to the state representation.
In order to reduce these things I remove the non-useful notion of order between the balls but that reduce it only to 259.

At last but not least there is the action to be taken.
In GLines game there is an only action: put on ball on an empty fields. Taken this way it make $81^{81}$ action...

I then attempt to reduce the state-action representation:

I first tried to reduce the board representation by a lot of means:
First using the idea of after state approach, for the color, in fact once we pick up a color the other colors value doesn't matter that reduce the board representation to a 3 power 81 possibilities but that's still too much.

I also try to apply the notion of non-color representation taking into account only the number of next color matching the active color, this was a great idea as the size is now 4 (no ball, one, two or three) instead of 259!

To simplify the problem I try to use a hierarchical action decomposition procedure. The first action being finding if it best to empty a Field or to fill one: in fact pickup a second action depending on the chosen field state (empty or full):

If the idea was to fill a fields, we know the color we need and the position to fill, the next action is to choose a ball to fill in the position, ideally the ball should be in annoying place (preventing from making a other line) of at least not in the alignment we are trying to make.

If the idea was to empty a field we have to look for the least annoying place, that is ideally in an same color alignment and at least not in an other color alignment or blocking the path to an alignment.

The Following thought far away from a Reinforcement learning problem that aims to avoid human expert methods in order to find it one. But I attempt tit in order to simplify the problem: I create weight board for filling a field or one for removing the ball in it. The weight were calculated in order improve the chose of the fields (with the rules that were describe sooner).

The basic random algorithm based on this simple idea isn't a success (maximum 10 point for the score i.e. just one 5-line) and has nothing to do with Reinforcement Learning.

I arrive to the conclusion that I need a function approximator for it like a neural network function evaluation. But as I didn’t knew how to program it, and as my tries, given the time I had, were not productive I haven’t generate any usable neural network yet.

In addition, a minor variation on the board can cause a significant difference of the evaluation function, as a ball placed in alignment empty the implicated fields. These features make the evaluation function in GLines more difficult to be approximated than those in Backgammon, chess, and Go

In order to continue the thinking of GLines problem I continue my project with theoretical studies of GLines
C.1 TRAINING IMPROVEMENT

A dynamic schedule could be used to vary the lambda parameter during training. Intuitively it makes sense to begin training with a large value of lambda and then decrease it as the learning progresses.

Moreover if the epsilon of the epsilon-greedy policy should be high in the training phase it has to be reduced after in order to see the correctness of the weight we obtain.

C.2 PERFORMANCE MEASUREMENT PREVISION

There are a number of methods available to assess the quality of play of the program; each of these methods has different strengths and weaknesses.

I present here the two methods I intended to use in order to analyze the strength of the algorithm.

The first method is playing and keeping trace of the score, if the method work well the score should grows until it reach the maximum limit.

The second method is to analyze individual move decisions. In other words, to check whether a player made the right move in a given situation. Comparing for instance with a expert human player choice.

C.3 TD-GAMMON VS. GLINES

As TD-Gammon is a great success in reinforcement learning game I try to compare the recognized appropriate properties of backgammon with the GLines ones.

Note the explanation of the backgammon property (in italic in the text is from Tesauro article)

- Absolute Accuracy vs. Relative Accuracy

Despite the large errors in TD-Gammon's evaluations, it is consistently able to make master-level move decisions.
In making a move decision, one has to choose between several candidate positions that are all very similar-looking. This is because they are all reached by small changes from a common starting position. Since the candidate positions have such a high degree of similarity, the neural network’s equity estimates will all be off by approximately the same amount of absolute error. Thus the potentially large absolute errors effectively cancel when comparing two candidate plays, leaving them ranked in the proper order.

In the Glines problem the choice of one movement can complete change the board configuration as the board could be separated in two parts, a wide range of ball could disappeared... the relative accuracy if very inferior of the absolute accuracy.

- **Stochastic Environment**
  A second key ingredient is the stochastic nature of the task coming from the random dice rolls. One important effect of the stochastic dice rolls is that they produce a degree of variability in the positions seen during training. As a result, the learner explores more of the state space than it would in the absence of such a stochastic noise source, and a possible consequence could be the discovery of new strategies and improved evaluations.

In this Case GLines have a great opportunities: the random color ball add to the random position of the ball allowed a wide stochastic environment. More over pathologies due to self-play training doesn't appeared here because of the "one man" nature of the game.

Another effect of random dice in Backgammon is that they contribute to the terminal states of the system being attractors (i.e., for all playing strategies including random strategies, the sequences of moves will eventually terminate in either a won or lost state). In backgammon this comes about partly due to the dice rolls, and partly due to the fact that one can only move one's pieces in the forward direction. The only way checkers ever move backwards is when the opponent hits them. These two factors imply that, even for random initial networks, games usually terminate in, at most, several thousand moves.

In GLines as three new ball are added to the board at each turn the terminal state (all board filled) is also very attractive. An interesting fact is the super fast arrival to a terminal state in case of bad game playing!

- **Learning Linear Concepts First**
  A third key ingredient has been found by a close examination of the early phases of the learning process. As stated previously, during the first few thousand training games, the network learns a number of elementary concepts, such as bearing off as many checkers as possible, hitting the opponent, playing safe (i.e., not leaving exposed blots that can be hit by the opponent) and building new points. It turns out that these early elementary concepts can all be expressed by an evaluation function that is linear in the raw input variables. Thus what appears to be happening in the TD learning process is that the neural network first extracts the linear component of the evaluation function, while nonlinear concepts emerge later in learning.
Such concepts are said to be context-insensitive, in that the evaluation function assigns a constant value to a particular feature, regardless of the context of the other features. On the other hand, an example of a context-sensitive concept that emerges later in learning is the notion that depends on where one’s other checkers are located.

It turns out that the linear function learned early on by the TD net gives a surprisingly strong strategy -- it is enormously better than the random initial strategy, and in fact is better than typical human beginner-level play. As such it may provide a useful starting point for subsequent further learning of nonlinear, context-sensitive concepts.

As in many applications, when training a multi-layer net on a complex task, the network first extracts the linearly separable part of the problem. This should also append to the GLines problem.

### C.4 ON OR OFF POLICY

There are two basic ways of using experience in reinforcement learning: off-policy and on-policy. Both differ only by the update rule used to arrive at an optimal policy. The off-policy (Q-learning algorithm) uses the estimate of the optimal policy for update of the existing policy and consequently separates exploration from control. The on-policy (Sarsa algorithm) uses the current estimate of an existing non-optimal policy for refinement towards a better existing policy. The only guarantee to arrive at an optimal policy with Sarsa is possible only if the existing policy progressively inches itself towards an optimal policy.

I found in on the article that: An on-policy approach for game learning has more opportunities for active learning in the exploration of moves as well as better on-line performance in teaching games.

Therefore the Sarsa algorithm should have given better result in the Glines game.
V - Conclusion

My first finding of this study is the real complexity of Reinforcement learning problem and more formulating it appropriately for probabilistic reasoning.

The second finding is that I understand better why Reinforcement Learning isn’t widely used in real world problem.

If the GLines problem was very motivating, it was also beyond my actual capacities, I’m very disappointed because I really wanted to find a method for having high score.

But seeking for information on RL-games and on neural network improves my knowledge and I didn’t think that my failure was a lack of time.

However I think I will make some research and programming on neural network during the break as the potential of such method really impress me. And why not longer research in function approximation to obtain a solution to the GLines problem!