LUDO Player: An Artificial Neural Network with evolved with Genetic Algorithms

Jorge Rodriguez

University of Southern Denmark, Campusvej 55, 5230 Odense M, Denmark
jorod14@student.sdu.dk

Abstract. To complete a LUDO tournament an artificial intelligence has been developed. A LUDO player with an Artificial Neural Network (ANN) that is evolved with a Genetic Algorithm (GA) has been desined and implemented for this task. To determine the skills of the player, a LUDO tournament is carried out with other players as a Q-learning with GA or a weighted input player.

1 Introduction

LUDO game belongs to a category of games with two main conditions: (1) is a game of chance and (2) the number of actions needs to be simplified. A LUDO player with an Artificial Neural Network (ANN) that is evolved with a Genetic Algorithm (GA) has been designed and implemented for this task. For each possible movement in the game, the ANN gives a number representing its the goodness. This output is always given even if the ANN has not been trained for it. The ANN is trained with a chromosome which defines how good is each situation. As the optimal chromosome is unknown, a GA is used to evolve the player. To determine the skill of the player, a LUDO tournament is carried out with other players as a Q-learning with GA or a simple weighted input player.

The paper is structured as follows: starts introducing the ANN used [2.1] and the GA [2.2], follows with the experimental results [3] and the analysis and discussion [4] to ends with the conclusions [5].

2 Methods

The LUDO player is consists of an artificial neural network that is evolved with a genetic algorithm (ANNGA Player).

2.1 Artificial Neural Network

A feedforward neural net with 9 inputs neurons, 3 hidden and 1 output is used [1]. Each input represents a situation of a brick of the player in the game. The output is the goodness of a movement based on the input and the internal ANN weights (Fig. 1). The net is trained with a sigmoid activation function (eq. 1) so

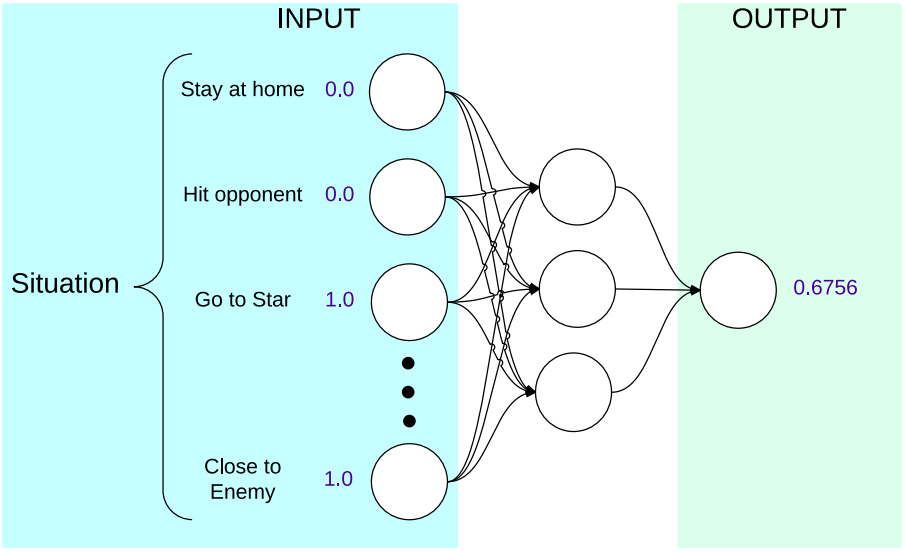


Fig. 1: ANN representation for a possible situation and its output

the output is a float number between 0 and 1. It is trained using backpropagation until a maximum defined error or a iteration limit is reached.

$$P(t) = \frac{1}{1 + e^{-t}} \tag{1}$$

Inputs of the ANN

There are 9 input neurons, each neuron is connected uniquely to an occurrence inside the game. Each of the inputs is expressed in the table 1.

Table 1: Inputs of the ANN

Input	Description
1	Move the brick to goal
2	Hit opponent and send it to its home
3	Hit myself and go to home
4	Go to star
5	Go to globe
6	Go out home
7	Close to an enemy
8	Go to safe area
9	Move into the safe area

Backpropagation

During the game, the 9 inputs lead to 17 different situations. A situation is defined as a combination of possible inputs. For example, a situation in which a brick is *close to an enemy* but also is *in a star*. However, there are also impossible situations as only *Go out home* because when a brick is taken out of home, immediately hits a globe.

When training the network, a desired output for an input is given. The training set is all the possible defined situations, in this case 17. And the desired output of each situation has to be chosen previously. As an example, the table 2 shows possible situations and a possible desired outputs. The ANN is trained

Table 2: Training example

	Inputs					**Desired output**
	Go to goal	*Hit opponent*	*Close to Enemy*	...	*Go to star*	
Situations	1.0	0.0	0.0	...	0.0	0.784
	0.0	1.0	0.0	...	0.0	0.784
	0.0	0.0	1.0	...	0.0	0.251
	...	...	...	...	...	...
	0.0	1.0	1.0	...	1.0	0.969

with backpropagation until an maximum error of 0.001 is found or 5000 iterations are made. The pseudo code of the algorithm used for training a ANN when a new player is created is:

1. Create ANN with its structure (9,3,1).
2. Define its ideal solution.
3. Train the ANN with the ideal solution and the training set until the error is less than 0.001 or 5000 iterations are made.

Selecting which brick move

The ANNGA player analyze the situation of each brick giving a numerical value of it. The situations are compared and the highest score is the choice to move. If two situations have the same value, both situations are equally good and a random brick is moved. The pseudo code for the algorithm of choice is expressed as:

1. For each brick
   (a) Determine the situation
   (b) Compute its value with the ANN
2. Move the brick with the best situation

2.2 Genetic Algorithm

For training the ANN, an ideal solution for the 17 situation is given. The ideal situations are unknown and due to is a game of chance, there is not an optimal. However, different ideal solutions lead to unequal results. Depending on the type of the opponent, a better strategy is searched. For example, if the ANNGA player is playing against three pacifist players, a good strategy is not to take into account the possibility of being hit by the others. The behavior of the ANNGA player is defined by the ideal solution for each situation and a GA is applied to find the most suitable strategy for each tournament.

Chromosome

Each gene of the chromosome is the desired output for each situation defined in the ANN. This means that if 17 situations have been defined, the length of the chromosome is 17. Each gene means the importance adopted for a situation. For example, a gene of 0.95 in *Hit the opponent* means that the ANN gives a lot of importance to the possibility of hitting. A gene of 0.01 in *Stay at home* and *Go to globe* means that the choice of being at home is not the priority.

Selection

A population of 8 players with different chromosomes is initiated randomly [3]. The size of the population is defined by the number of computer threads as will be explained in the implementation [3.1]. Each player is made to play a defined number of games against other players and for each game some points are given. At the end of all the games, the two players with more points will become the mother and the father. Two ways of giving points have been experimented: (1) one point given if the game is won and (2) a proportional number of points depending on you position, being the first 3, second 2, third 1 while the looser doesn't receive any points. Despite the selection of the chromosome is the same the two strategies can lead to different results.

Offspring

Once the mother and the father have been selected, the next generation can be made. The offspring is the population derived from the previous generation. An elitism of two-eighths is made. This means that, in the next generation, the father and the mother will remain the same. As there are 8 players, the other 6 are generated from them. As LUDO game is a game of chance, the players are made to play a defined number of games. This is known as a tournament. It has to be big enough to reduce the *luck component* up to an acceptable limit. If the tournament have achieved this, doesn't make sense to keep other chromosomes different from the two best ones. Otherwise, this would mean give more *opportunities* to a chromosome when it is supposed that the tournament should do that. Thus, only the father and the mother remain equal, the rest are changed.

Crossover

The chromosome of the other 6 players is created crossing the chromosomes of the parents. Two strategies have been implemented: (1) One point crossover and (2) uniform crossover. The uniform crossover have given better results and it is implemented making a random mask that chooses what genes from the father will be copied into the offspring. The rest is copied from the mother. The number of genes copied from one of the parents respect to the total is known as the *density* of the crossover. A density of 50% has been applied meaning that for one children half of the genes are from the father and the other half from the mother.

Mutation

After the offspring is generated a mutation is made. For all the children, a mask that defines which gen will mutate is made. The number of genes to mutate respect to the total is defined as *density* of the mutation. A random density has been applied meaning that a random number of genes is mutated. For each gene to mutate, a random number is generated between two limits: *max mutation* and *min mutation*. In the experiments, due to the high speeds of the generations, a mutation between (-0.2, 0.2) is chosen.

Pseudo code for the GA

As the implementation of the game make use of all the threads in the computer, 8 ANNGA players play each own tournament against 3 SemiSmart players. So the pseudo code is:

1. Initialize 8 ANNGA players with random chromosomes.
2. For each generation:
   (a) Each player, plays a defined number of games against 3 SemiSmarts.
   (b) When finished, the number of points of each player is compared and the two best ones are selected.
   (c) Create the offspring with the selected crossover method.
   (d) Mutate the offspring.
   (e) For the newborns, train their ANN with the new chromosome.
3. When last generation is reached, the player with more points is put into the last tournament where the overall results are obtained.

3 Results

3.1 Implementation

The ANNGA player consists of two parts: (1) the ANN player and (2) the genetic algorithm. The ANN player has been implemented using the *Encog* Libraries [2] that makes uses of multi threading and even GPU to compute some operations.

This library is used to create the ANN and train it based on the situations and the chromosome.

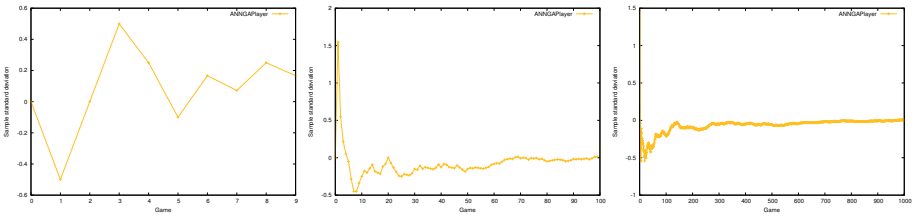
The GA has been implemented making use of the multi threading capabilities of Java. Due to the machine used has an 8 threaded processor, in each generation 8 tournaments are played in parallel. This makes the ANNGA improves 8 times faster compared with a single threaded GA. Despite the low number of threads makes the population really small, the high speed computation of the CPU makes the GA give good results.

Computation time

The machine has a processor Intel® Core™ i7-4702HQ CPU @ 2.20GHz 8x and 16 GB of RAM. When running, the computer uses the 100% of each thread and around 1.2 GB of RAM. In each thread, 1000 games last around 600 ms which means that, if 8 threads are used, 8000 games are done in the same time. The number of generations per second depends entirely on the number of games in a tournament, then this is determinant to find a good solution.

3.2 Luck factor analysis

Trying optimize the computation time, an study of the luck factor is made. The tournaments are made to reduce the luck component and determine the real skill of the player. Despite the by default 1000 games are played in a tournament, the luck factor have been analyzed. For this purpose, four ANN players with the same chromosome are made to play a defined number of games. When finished, the mean of points is obtained taking that value as what defines the skill of the player. Then, an sample of games is taken and its mean is calculated and compared with the global mean. This means are used create an sample standard deviation analysis and the result is that the pseudo random number generation is *patterned* (Fig. 2).



(a) Sample standard deviation for 10 games (b) Sample standard deviation for 100 games (c) Sample standard deviation for 1000 games

Fig. 2: Analysis of the luck component to determine how many games to play in a tournament

A stable result is considered when the sample standard deviation is in ± 0.2. When the games in a tournament are more than 10, for example 100 (Fig. 2b), a stable result is found around the 30% of the experiment, which would lead to think that 30 games is enough to eliminate the luck component. However, when the same experiment is carried out for 1000 games, the stabilization comes with 300, and no 30, again around 30%. This experiment has been repeated with different samples and the conclusion is than above 10 games, the uncertainty becomes acceptable always. This could be due to the way of generate random numbers that Java has, so at some point the random numbers stops being random and generates a pattern.

3.3 LUDO player comparison

To analyze the skill of the ANNGA player, is compared with other players already implemented in the code and others from other students. This section starts analyzing the fitness function when playing against SemiSmarts players. Then an analysis for the best chromosome of the evolution compared (named *the legend*) and the best one from the last generation is made. Finally, a group tournament is made with the players of some classmates. For this tournament one chromosome has to be chosen to play.

ANN evolution

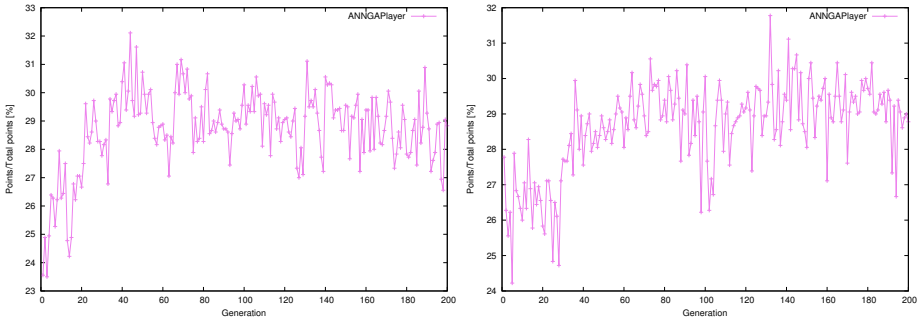
Based on the luck factor analysis [3.2], instead of the 1000 games given by default, the tournaments are made with 300. This suppose an improvement of 66% in time computation with the same results, which improves the GA. As the selection in the GA can be made based on *won games* or *accumulated points*, the total points also varies. However, the results are given in a percentage of the points given to make it independent of the number of games. The fitness function shows the evolution of the population in each generation.

Two studies have been carried out (Fig 3a and Fig 3b) for 500 generations, 300 games tournament, being the ANNGA player the first, with a uniform crossover and ± 0.2 in mutation with a random mask.

In both plots the improvement stabilizes around 29% of the possible points in the generation 40. However, the worst generation gets 24% of the points which means that wins almost all the games from an early stage. This with the fact that the highest success rate is not over 31% makes the fitness function quite small in improvement terms. No significant differences are found between selecting with the won games and the distributed points.

The legend chromosome

Due to the LUDO game is a game of chances, perhaps the last generation is not the one with most points. For this reason the best chromosome during the



(a) Fitness function when selecting only with won games

(b) Fitness function when selecting with distributed points

evolution is stored and compared with the best one in the last generation. This chromosome is called *the legend*. The comparison is only made playing a tournament being both ANNGA players the first ones and playing 10000 games against three SemiSmarts (Fig 4).

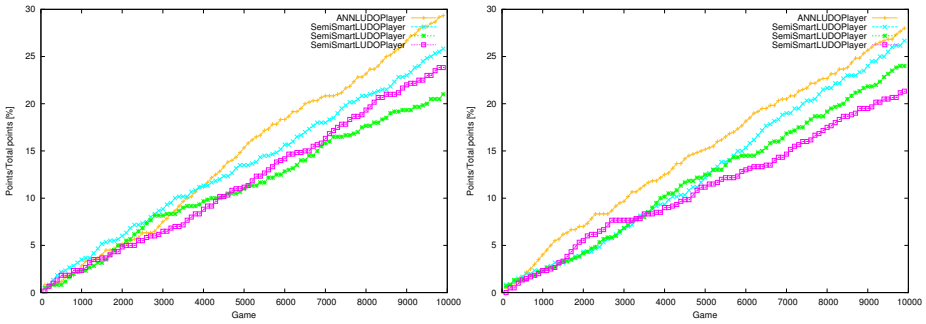


Fig. 4: Comparison of the results between *the legend* (Right) and the best chromosome of the last generation (Left)

The results shows how last generation perform slightly better, but enough to carry out the rest of the experiments with the last generation.

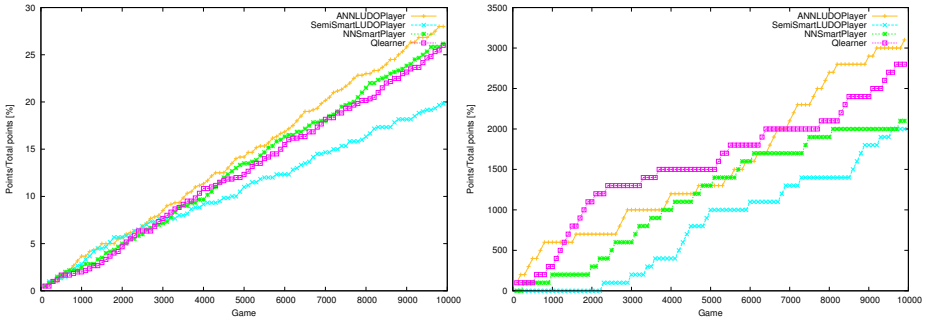
Group tournament

A tournament against the AIs from other students has been carried out. The tournament has been made with an ANNGA player, a SemiSmart, a Q-learner with GA from Carlos Moro [5] and a NN player from Ignacio Torroba [4].

– *The Reinforcement Learning with GA player*. This player implements a Q-learning algorithm with 9 actions and 7 states. The rewards are changed

with a GA, as in the ANNGA player, and then adjusted with a learning rate and a discount factor. This approach is potentially better than the ANNGA due to it contain more information than the ANNGA chromosome.
– *ANN Player*. This player is as the ANN player explained in this paper with two differences: (1) the internal structure is different having 25 neurons in the hidden layer and (2) the situation of the game is analyzed global, not for each brick. This leads to a global analysis that contains perhaps more information about the game compared with the ANNGA player. The ANN learns from a trainer agent that is based on a predefined actions.

The test was made during 10000 games and the chromosome chosen for the ANNGA was: (0.45, 1.00, 0.00, 0.63, 0.48, 0.10, 0.00, 0.94, 0.74, 0.66, 0.93, 0.15, 0.00, 0.95, 0.40, 0.11, 0.30). This chromosome was evolved during for 500 generation, 300 games tournament, being the ANNGA player the first, with a uniform crossover and ±0.2 in mutation with a random mask. Two tournaments were made, one counting the games won and other with the distributed points.



(a) Competition with distributed points (b) Competition only with won games

Fig. 5: Competition for 10000 games comparing won games and distributed points. ANNGA vs NN vs Q-learner vs SemiSmart

The results (Fig 5) shows how the ANNGA player performs better in both situations.

4 Analysis and Discussion

The results shows that the ANNGA player wins in all the tournaments against the AIs already implemented in the code and also against the AIs from other students. This have been achieved with 500 generations what implies around 3 minutes of processing time. This factor is important. Also, the player implemented has not a learning limit, meaning that the goal is to win not learning from another players. It tries to find the optimal solution for the specific players.

This is congruent with a luck based game because there is no an optimal solution.

Despite probably the Q-learning with GA was potentially a better player due to it stores more information about the game, the ANNGA has shown to be better and in less time. This could be because the Q-learner has not being trained enough and here is when the time factor becomes important. It doesn't matter all the possibilities that are considered if there is no time to test them. Probably, the best strategy is not try to study all the possibilities but only the best ones.

Now the ANNGA player has a fixed chromosome size, but a further work would be make this chromosome plastic. Depending on the game, maybe the situation in which the brick can hit an opponent, be in a star and be close from another enemy only appears the 5% of the times. When applying an specific gene to that situation maybe resources are being wasted. An improvement to the player would be to make it modify its chromosome online based on what it consider relevant to evolve. The main advantage of an ANN is that given an input it will always give an output, so despite the net has not been trained for that situation it can give a solution.

Other improvements can be made in the GA. The population now is really small (8), so increasing the size would increase the performance. This limitation is due to that now the CPU threads are used but perhaps the program could be implemented using GPU cores instead, where millions of parallel games can be played ad the same time.

5 Conclusion

A LUDO player based on an Artificial Neural Network and evolved with a Genetic Algorithm has been implemented. Due to its implementation in multithreading and some studies regarding the number of games to do in a tournament, the learning speed has been improved from the suggested implementation line. This has lead to a player that in few seconds is able to win not only against the already implemented players, but also the AIs of other students.

6 Acknowledgments

Thanks to Carlos Moro and Ignacio Torroba for being so inspirational when talking about this project and for its contribution to a deeper analysis comparing all the players. I would like to thank also my parents because somehow have funded this project.

References

1. Kasper R. Sæderup. *An evolving LUDO player based on neuronal networks, backpropagation and GA*. University of Southern Denmark.
2. Encog Machine Learning Framework, *http://www.heatonresearch.com/encog*
3. Poramate Manoonpong, *Tools for artificial intelligence (building brains for machines)*, Lecture slides, RMAI2-U1, SDU, (2015)
4. Ignacio Torroba, *LUDO game AI2*, Kunstigs intelligens værktøjer (SDU), (2015)
5. Carlos Moro, *Artificial intelligence for LUDO game*, Kunstigs intelligens værktøjer (SDU), (2015)