

# Training Neural Network for Ludo

Frederik Hagelskjær

University of Southern Denmark, Campusvej 55, 5230 Odense M, Denmark  
frhag10@student.sdu.dk

**Abstract.** This article is the hand in for the course AI2 at the University of Southern Denmark. It concerns the training of a neural network for playing ludo. The approach is a black box approach where the algorithm is to be trained without using any prior analysis of the ludo game. Thus the goal is to analyse the abilities of using simple training with backpropagation using moves from good LUDO players. Thus the state of the board is used without any preprocessing. It is shown that normal learning algorithms are not able to cope with the complexity of LUDO. Backpropagation trying to learn from players using GOFAI cannot simply learn. It is concluded that preprocessing of data is an important aspect in learning algorithms. An interesting aspect found is that most neural networks initiated with random parameters will outperform a random player significantly.

## Introduction

The LUDO game.

Using AI to develop a Ludo playing agent.

There are no real requirements to the agents performance, but the results of the developed AI should be extensively documented.

The goal of AI is an introduction to artificial neural networks, and therefore the agents developed in this article will use such an approach.

The complexity of the LUDO game, makes it impossible to make a analytical solution by simple determining every solution for every state, thus making decision making a lookup table, as shown in [?]. This article tries to determine an appropriate neural network for an agent playing on level with other solutions.

The performance of the ludo player will be measured by the proverb: "If you ain't first you're last"<sup>1</sup>, thus only the amount of victories will be measured. That is test will be run for the ludo player against proportional opponents until a linearity is found, i.e. more plays wont change the relationship between wins.

Tweaking of input could also be called embodied AI, move when pushed or something.

The goal will be to avoid any "tweaking" of parameters towards the creation of the neural network. Examples of such tweaking will be shown, both to show it's strengths and why it is desirable to avoid. Neural networks are often created with preprocessed input [?].

---

<sup>1</sup> [http://en.wikiquote.org/wiki/Talladega\\_Nights:\\_The\\_Ballad\\_of\\_Ricky\\_Bobby](http://en.wikiquote.org/wiki/Talladega_Nights:_The_Ballad_of_Ricky_Bobby)

An example of such preprocessing can be seen by looking at the ludo game and its dynamics. Every turn a piece is dictated to move forward, and except from stars and globes overall progress is the same. Thus one could speculate that the most important factor is hitting home. One could thus give the relationship of positions between pieces instead of all positions, as seen in figure. The indicating the gain from using stars and globes.

The problem is that this isn't necessarily the best solution and could thus steer the network towards a suboptimal solution.

Thus the idea is to create neural networks trained only by the knowledge that winning is positive. Seeing ludo as a black box problem.

Training the neural network.

## Methods

The neural networks were implemented using PyBrain described in [?]. This was done because of a desire to obtain the flexibility and ease of Python while still retaining the speed of C.

The implementation of the ludo game was done using Leon Larsens ludosimulator.

Before training the neural networks the dimension of the neural network must be decided. There exist no formal definition for how to choose the size of a network with unknown complexity. Though to avoid over fitting the system when training, the dimension should not be too large. A standard element of neural networks is the perceptron taking weighted input and returns 0 or 1. Thus a combination of these could decide which brick to move. The problem is that a single layer perceptron is a linear separator and to avoid this a hidden layer is introduced [?].

The ludo player developed uses a neural network to decide which brick to move given state and roll values. The values are scaled between 0 and 1. In ludo one cannot decide not to move any of the bricks and thus it is necessary to guarantee that the move is legal. This is done by taking the best value that gives a change in states and using that as the move. The start position is a random start value, so if all ranks were equal it would be a random output.

The number of and size of the hidden layers will be described in each test, though input and output size stays the same.

To train the network towards the optimal multiple procedures were explored. These methods are described in the following section.

## Imitating Human Player

Originally neural networks were trained using different error minimization procedures. Back-propagating was used.

Thus it is necessary to provide a training signal for this approach. By looking at different ludo games and measuring the procedure of the winner one could create train a neuron to make winner decisions. The system is still seen as a

black box as no evaluation is done on the actual moves, they are simply used by the winner.

As no data set of ludo winner moves were available an imitation were performed. Test data were collected by agents taking random moves playing against each other. By a chance of 1/200 the board state and dice roll were added to a dataset.

A test person then evaluates the data, and chose the "perceived" optimal move. Thus a learning set were created for the back propagation algorithm.

The first test is to see whether it is possible to train a neural network using datasets with an answers provided by a human. 240 situations were evaluated before before the training began. Using pack-propagation the neural network were trained to fit the data set, and made to play against opponent.

### **Random Wins**

Another test were performed to test the pack propagations possibility of learning simply from random games. Numerous games were played with random agents and for every game that resulted in a win, states and answer were collected for every round. Thus a dataset to win against random agents were created.

Not really but the idea is there.

To see whether winning patters could be seen from whole games. could be seen in the

### **Random Neural Networks**

A test were performed to check the performance of neural networks with randomly generated parameters. The test was done to check the effectiveness of the training.

### **Imitating Another Agent**

### **Results**

By using the trained data from the human choices the following results were found.

Fitting the data using automatically generated data. Trying to decide the optimal size to fit the data, i.e. the test data.

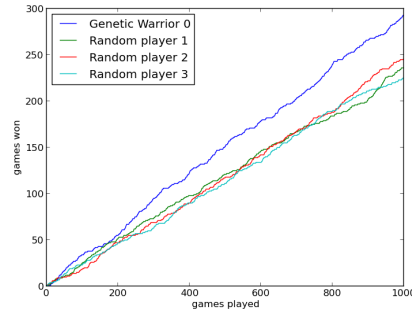
For the genetic algorithm multiple randomly generated parameters were created.

### **Random Data**

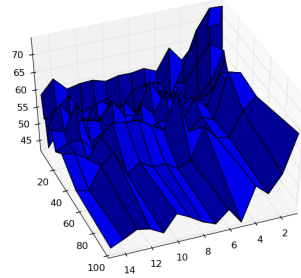
Random data results. All neural networks outperform the random players. Data set, example. The results from such a test involving 50 different networks can be seen in figure 3.

Analysis of moves actually being used.

No immediate connection were seen between the sound and good moves.



**Fig. 1.** Result of 1000 ludo games. Genetic warrior being the trained neural network.



**Fig. 2.** Plot showing the error of the training set using different epochs and number of hidden neurons. Respectively on the right and left side.

## Analysis and Discussion

From the initial generation of datasets with random parameters it can be seen that almost all of them outperform the random approach. This indicates that using somewhat of a plan will outperform a completely random approach, expect of course if the plan is

Look into grouping of whether certain moves belong to a "winning" or losing set.

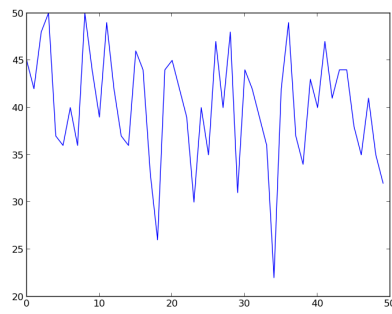
## Conclusion

Conclusive works better than random, almost

Very complicated when using complete board as input.

Training towards other players - what are the results.

Instead one should look into the problem. Properly solve the small problems first.



**Fig. 3.** Number of wins for 100 ludo games for 50 different randomly generated neural networks.