8 lutego 2016

Intelligent Information Retrieval

Project

Agent with cognitive structure

Authors:
inż. Mateusz Bodziak 131101 IDS, KSDiR
inż. Szymon Grocholski 131126 IDS, KSDiR

1 Introduction

The project is about to implement an agent which should learn the rules of the Wumpus World game. The solution will be based on neuron networks.

1.1 Main idea

Our task was to create an agent without a priori knowledge about the game, the environment and rules then train him and check whether he has learnt something.Main idea is to prepare a training set which will be used by a neural network to learn the rules of the world and test is comparing with different network architectures. Main data flow is shown in the Pic. 2.

Everything starts from worldGenerator where the world is being set. The GUI for the game is presented in Pic. 1. The legend is as follows:

- red field - Wumpus!
- gray field - hole
- yellow field - gold
- black rectangle - player
- senses are wrote explicite

Then we can play the game by our own or let the network take the decisions.

For every move the agent gets -1 pt. For finding gold agent gets 1000 pts and for encountering the Wumpus or a hole gets -1000 pts. After the game we can check the status of points and the game result.

The world is generated in the beginning of every game and is randomly set. Condition is that in the field $(1, 1)$ where the player starts there is no gold, Wumpus nor hole. Unfortunately around 10% of the maps are unsolvable because of unfortunate placement of Wumpus and holes. The other 90% are the

1.1 Main idea

maps which are solvable with certain probability. Not all of them are solvable with probability of 100%. In many cases the player has 50% chance of a good move regardless of previous moves and previous senses.

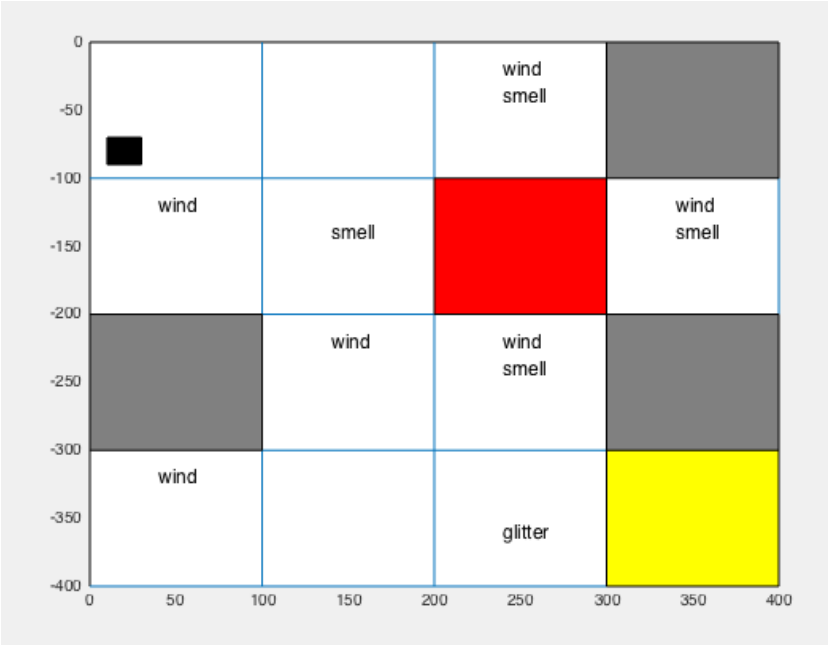

Fig. 1: User interface for the Wumpus game

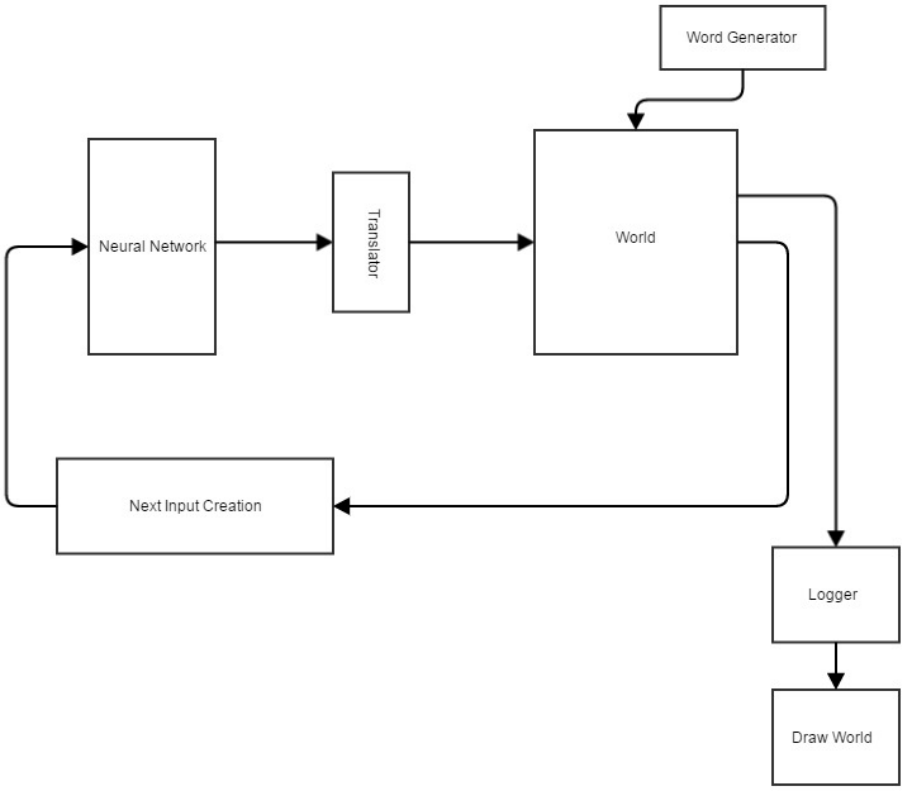

Fig. 2: Data flow of the program

2 First Approach

2.1 Neural Network

Our neural network contains parametirized number of hidden layers in testing phase we used 1 and 2 layers. The version with one layer is presented on the Pic 3

The input vector is as follows:

$$[x_i, y_i, senses(i), x_(i-1), y_(i-1), senses(i-1), x_(i-2), y_(i-2), senses(i-2)]$$

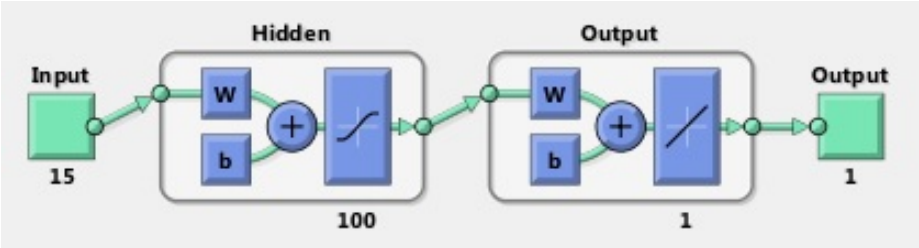

Fig. 3: One of the network architectures

where x_i - is the current position the row coordinate y_i - is the current position the column coordinate $senses(i)$ - is a vector of senses from field with coordinates (x_i, y_i) rest of the coefficients are analogical from previous steps

The output of the network gives us a number from $0-360$ which is the angle. This value is translated to a move according to the Pic. 4

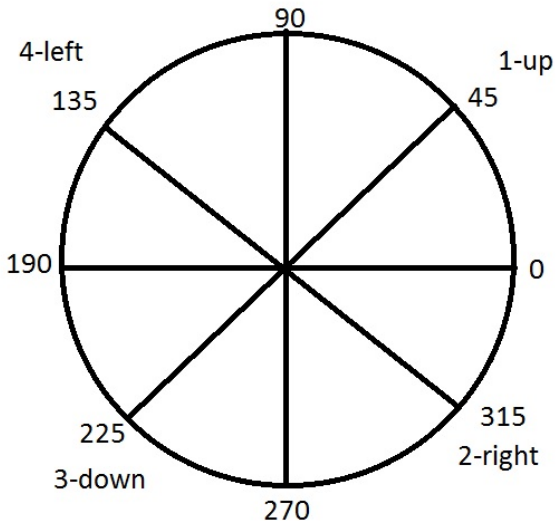

Fig. 4: The picture shows how angles are mapped into move

- 1 - UP
- 2 - RIGHT
- 3 - DOWN
- 4 - LEFT

2.2 Implementation

1. Training data preparation

- Manual data generator
- Automated data generator

2. Network training
3. Execution
    - Animation of one game
    - Performance comparison

Training data We construct two ways of preparing the training data. First one was automated, prepared based on random walkthroughs from which we select and store only good ones. This method generated walkthroughs with success - gold was found. Unfortunately this method has stored many moves which had no sense, like bumping often into walls and ignoring the meaning of the senses.

Improvement of this was to input some a prori knowledge by human, so we created the manual training data generator which stored the walkthroughs which were committed by a real human player. This enabled us to prepare more suitable training set but it was tiresome and time consuming method.

After every good walkthrough of the given game all the world are stored on the worldList. This list is needed to display the world for debugging purposes and for preparation of the input vector for the network.

Network training In this stage we use our training date prepared in the previous stage to train the network. The concept of the training is presented on Pic. 5

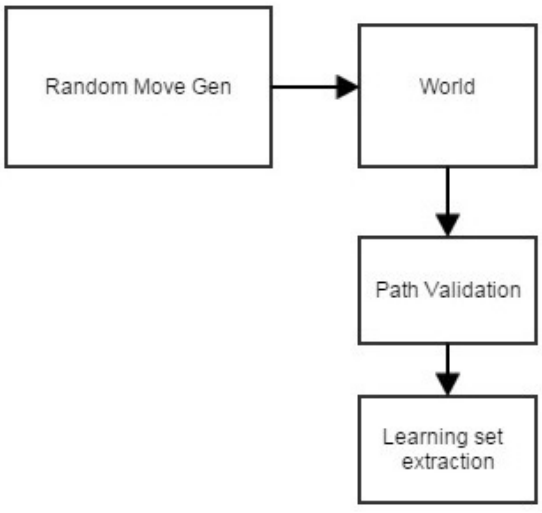

Fig. 5: The algorithm of the training

Animation of one game There is an option of running one game using the network of choice. It is easy to change the time intervals and all the other animation details. This option allows to see how the agent is behaving in different situations. To gather some statistical data it is better to use Performance comparison.

Performance comparison In order to compare the performance of the networks we constructed an automated tool that was playing the game using trained network. At the same time a random player was playing the same worlds. After selected number of games we calculated the average score and the success rate. Look Pic. 6.

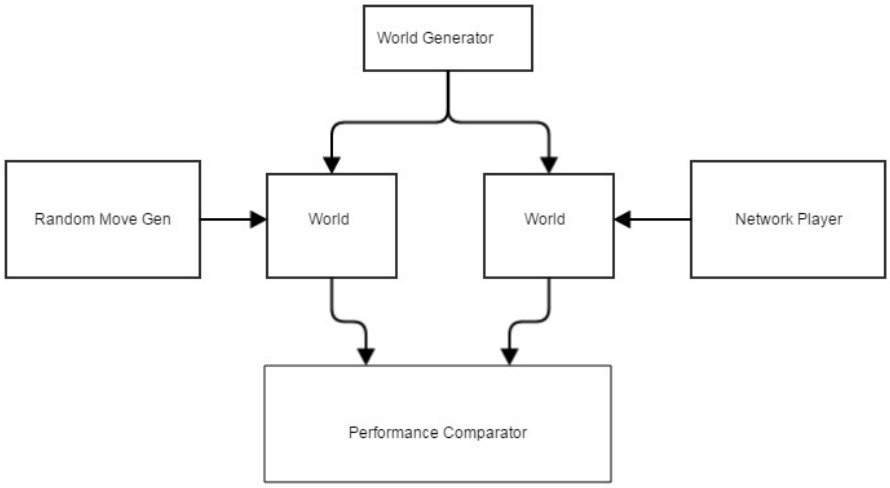

Fig. 6: Performance comparison

2.3 Results

In this test we used Neural Networks that were learned using the same automatically generated learning set. The we check the performance by playing 1000 games and comparing the results to random player scores.

Tablica 1: Networks based on 'randomized' training data

Number of Neurons	Avg Score	Success Rate	Random Avg Points	Random Success Rate
80	-259.72	17%	-588.03	18%
100	-467.25	21%	-580.02	17%

The points are in the range of $< -1000, 1000 >$. It can be observed that our agent is a little bit more aware than the random one. He has committed fewer loses and got more points. Unfortunately the overall result is not satisfactory at all. The winning ration is at level of 20% and a little bit better for 100 neurons.

3 Second approach

It is clearly visible that the first approach was not one that was promising any improvement. So we decided to rearrange the network. The main issue was that it was hard to provide correct training data. Since we had a framework that was flexible and robust enough we could easily test different configurations of the network.

3.1 Neural Network project

Valid moves We found that our network often chooses move that hit the wall so we decided that this is the part of solution that we should focus on. On the pic. 7 we can see the architecture of the network.

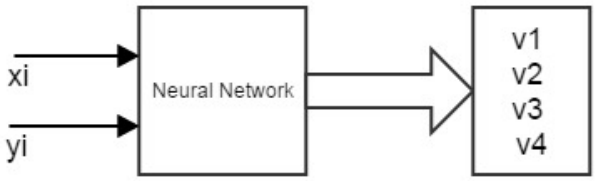

Fig. 7: Valid moves network

The network returns back a vector of 'validity' of the moves. So each output is a vector $[v1, v2, v3, v4]$ where $vi \in < 0, 1 >$. When the value is 1 is surely valid when 0 it is completely not valid.

Decision Network Only avoiding walls was not our goal, so we then thought about a decision making network. Since our last tries consisting three last points were unsuccessful we decided to go witch more local but easier to train controller.

What we noticed is that a network once detects glitter should not diverge more than one field from the point in which the sense was picked.

To make it easier for the network we stated that it is not important if we feel wind or smell since both are equally deadly. So we created a combined sense called 'danger'. Glitter was left intact.

As an input we provided last two senses, last move and current senses (look 8).

The output was a $4x1$ vector containing probabilities of this move to be chosen by the agent.

Combination The outputs from the network were two $4x1$ vectors. One was representing the validity of the move the second one probability of this move being a good one given current situation.

To combine the two vectors we multiply the value by value. Then normalize the outcome and create 'proportional roulette'. The move that is picked is the

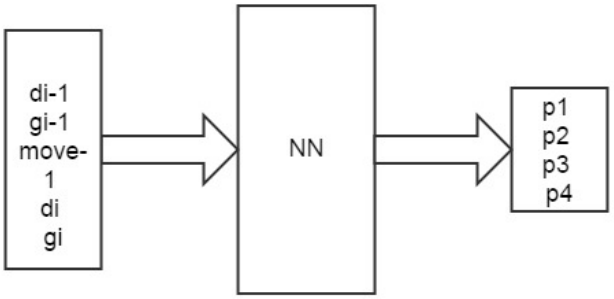

Fig. 8: The decision network

move that the agent will take.

3.2 Results

We encountered a big increase of success ration. Newly created agent was able to win around 31% of games.

There are still some issues that needs to be addressed. When an agent doesn't have any memory it may fall into loops that may cause death cause by two many moves. It doesn't know that he has to risk it when there is no other options.

4 Performance testing

Finally we wanted to test our network and compare the results with different kind of players. One of the player was a random player which just choose its move with the probability of 25% for each direction. The second one was our first approach with one network. Third one was the combained network where we splited the hard task into two simpler taks. Other players are human players with different knowing of the game. The performance was measured by ratio

$$performance = \frac{wonGames}{allGames}$$

and is presented in Table.2.

From the results we can assume that the number of 10K games estimates fairly the performance of a random player. That was our minimum goal - to be better than random player. From the results we can easily deduct that splitting the difficult task into two easier tasks was a good idea. The performance is higher than in the first approach. After that we wanted to compare our result to humans. What is really suprising our agent was more effective than most of human players! On the other hand the number of games was different so the results of human players are not accurate enough, although humans during the game are more and more tired and loose their attention so that we assume that the performance in 10000 games could be even worse. Mr Paweł who is a Master of Psychology got 27%. Nevertheless our agent is the leader except of

Tablica 2: Results of the performance of tested players

Player	Won games	All games	Performance
Random	1925	10000	19%
1^{st}network	2132	10000	21%
Mr Paweł from the train	21	78	27%
Szymon	35	120	29%
2^{nd}combained networks	3117	10000	31%
Szymon's Mum	20	36	56%

Szymon's Mum result. In this case we assume that small number of tries could result with bad estimation.

The human game is presented in Pic.9 - the previous senses are visible but the types of fields are hidden.

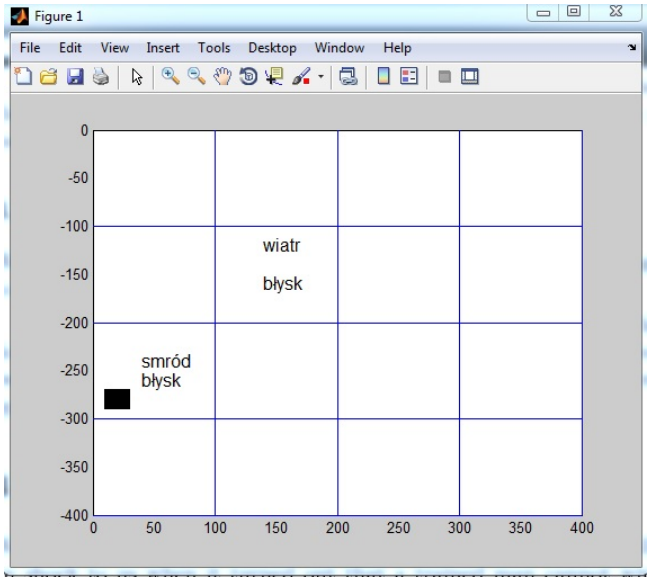

Fig. 9: Human game play

5 Summary

The problem stated to write an agent with cognitive structure. The knowledge was provided to him in the training of the neuron network. The first approach was not accurate because of the the trainig set, high level of abstraction and too many inputs. The conclusion from second approach is that its better to split the task into two easier tasks. The train data was prepared in better way too which resulted with a good performance. This project prepared a good framework for training, testing, and teaching the network how to play a Wumpus game.