

7 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.

1.1 Main idea

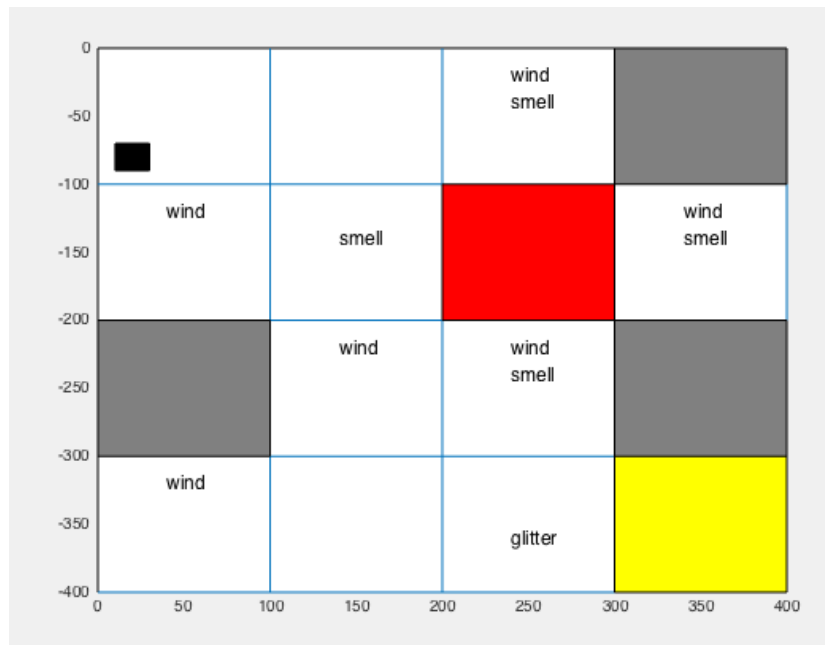


Fig. 1: User interface for the Wumpus game

1.1 Main idea

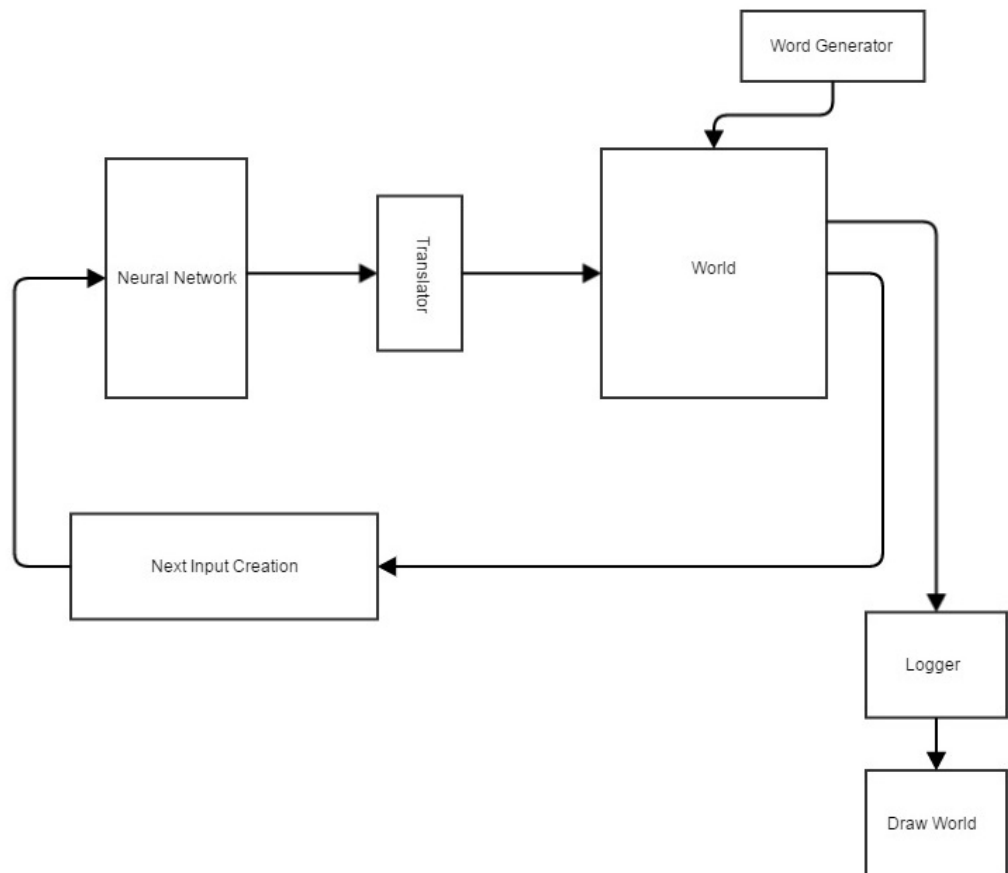


Fig. 2: Data flow of the program

2 First Approach

2.1 Neural Network

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

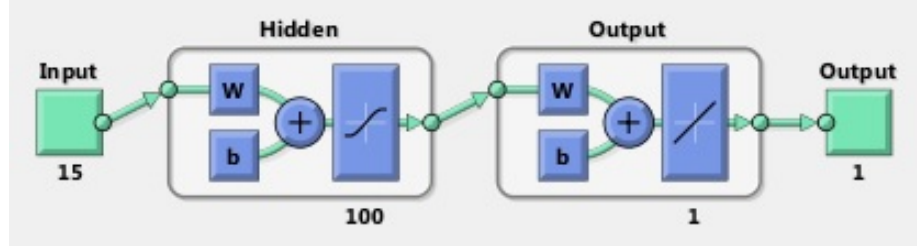


Fig. 3: One of the network architectures

The input vector is as follows:

$$[x_i, y_i, senses(i), x(i-1), y(i-1), senses(i-1), x(i-2), y(i-2), senses(i-2)]$$

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

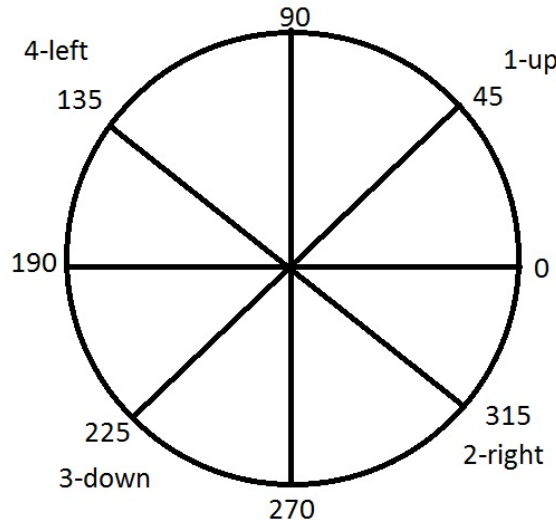


Fig. 4: The picture shows how angles are mapped into move

2.2 Implementation

- 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 priori 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 data prepared in the previous stage to train the network. The concept of the training is presented on Pic. 5

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.

2.2 Implementation

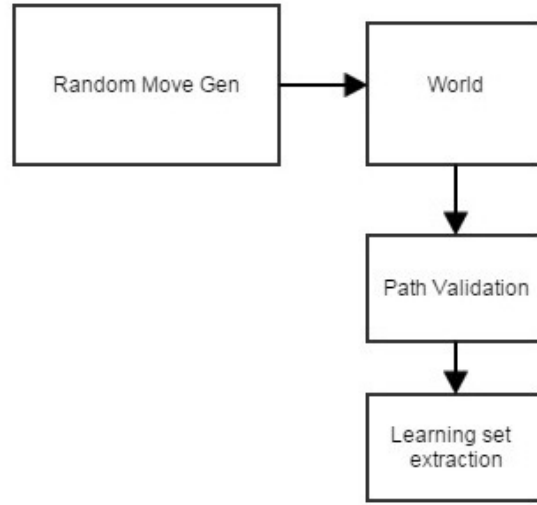


Fig. 5: The algorithm of the training

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.

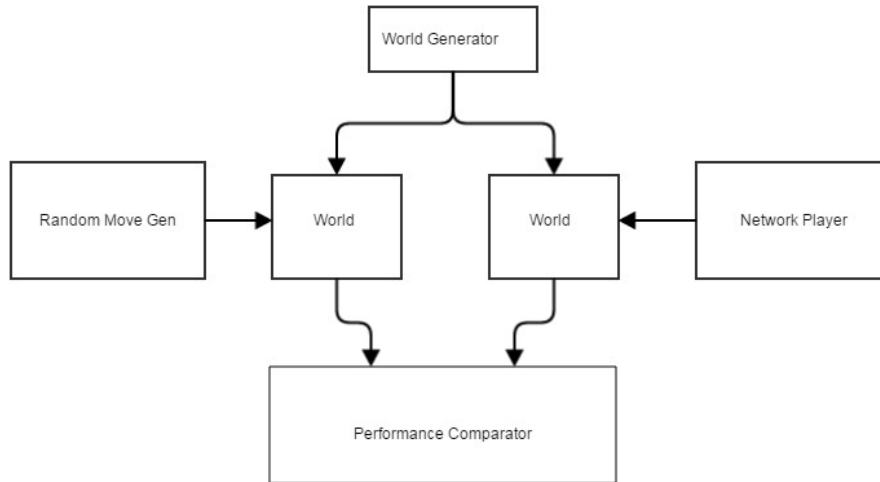


Fig. 6: Performance comparison

2.3 Results

2.3 Results

In this test we used Neural Networks that were learned using the same automatically generated learning set. Then 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 losses 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.

2.4 Conclusions

The problem stated that the agent has no a priori knowledge of the world. The knowledge was provided to him in the training of the neuron network. Unfortunately the level of abstraction was too high for the net configuration and architecture presented in the project. The input to our neuron network was constructed with 1 current and 2 previous position of the agent with the senses. It is very hard to match the input vector with the meaning - position and the senses. In the end we obtained results a little bit better than random movement in win/lose ratio. On the other hand our network gives much better result taking into consideration the average points result.

This project prepared a good framework for training, testing, and teaching the network how to play a Wumpus game. Unfortunately the network performance results are not as good as we wanted them to be. The next step should be to consider again the structure of the neural network, the number of hidden layers and the input itself.

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

3.1 Neural Network project

pic. 7 we can see the architecture of the network.

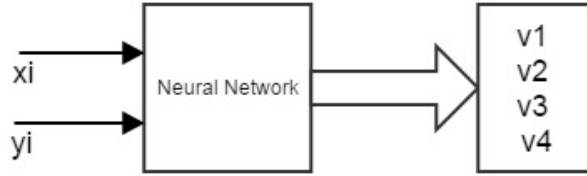


Fig. 7: Valid moves network

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 with 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).

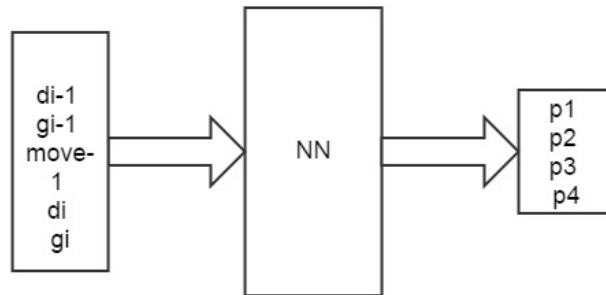


Fig. 8: The decision network

The output was a 4×1 vector containing probabilities of this move to be chosen by the agent.

Combination The outputs from the network were two 4×1 vectors. One was representing the validity of the move the second one probability of this move being a good one given current situation.

3.2 Results

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 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 32% 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 Human player

We also were curious how much games can a human win. Actually it came as an shock to us when it turned out that a trained man cannot win much more than 50% of games even when all the history is displayed on the screen pic.9

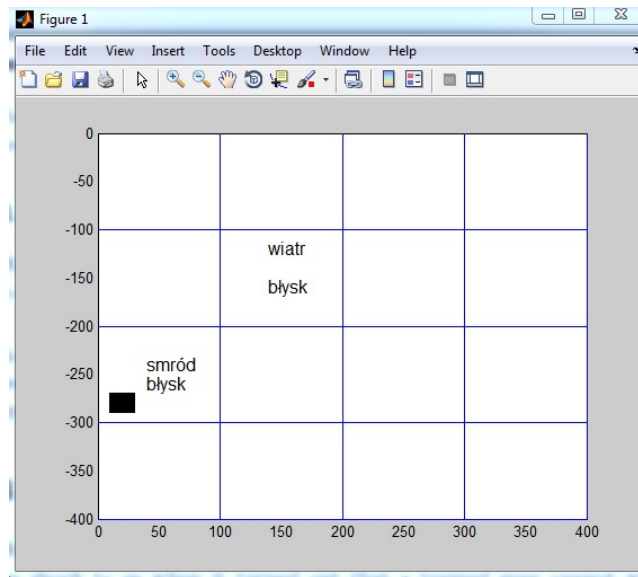


Fig. 9: Human game play

5 Summary