university of groningen / faculty of mathematics and natural sciences / artificial intelligence

# Playing Ms. Pac-Man using Advanced Neural Networks

Bachelor's Project Thesis

Sjaak ten Caat, s2477637, a.j.ten.caat@student.rug.nl
Supervisor: Dr. M.A. Wiering

**Abstract:** In this thesis, we used several types of neural networks, differing in the amount of hidden layers and if they use a sigmoid function or ReLU to calculate the activations. These networks, along with Q-learning, were used to play Ms. Pac-Man. Previous research already used Q-learning along with neural networks to play Ms. Pac-Man, but with one hidden layer and only sigmoid activation functions, to show the effectiveness of higher-order action-relative inputs. The same inputs were used here, along with an extra addition. The final results of the networks from this thesis were compared with those of the previous research, both when the environment remained the same during training and testing, and when the environment differs. The outcome was that the additions made to the original network did indeed lead to a significantly better performance.

## 1 Introduction

Reinforcement learning is a much studied subject within the field of Artificial Intelligence (Mnih et al., 2015). A popular way of finding and testing new approaches has been through the use of games (Galway et al., 2008; Szita et al., 2009). At first these were traditional board and card games, such as checkers (Schaeffer et al., 1992), backgammon (Tesauro, 2002) and poker (Bosilj et al., 2011), but in later years this has moved to digital games, like racing games (Cardamone et al., 2010) and first-person shooters (Makarov et al., 2016).

The game that this research studies is Ms. Pac-Man (Handa and Isozaki, 2008; Szita and Lõrincz, 2007). Developed in 1981, Ms. Pac-Man is an improved version of the original Pac-Man. The goal of the game is to gather all of the pills scattered throughout a maze, without being caught by the ghosts following you. In Pac-Man, these ghosts would follow set patterns, so that even though the state space is very large, it is still a deterministic environment. However, Ms. Pac-Man introduced a factor of randomness, which changed the environment from deterministic to stochastic and thus increased the complexity. As of yet, the high score obtained by humans is much higher than that of AI agents (Samothrakis et al., 2011), therefore leaving much room for improvement.

This paper primarily continues the work done in previous research (Bom et al., 2013), where reinforcement learning was used in combination with a multi-layer perceptron, or MLP. This research featured three mazes to simulate a game of Ms. Pac-Man, where two mazes were used for training the network, and a third for testing. This research project uses the same mazes under the same circumstances. The goal of the previous research was to reach a low-complexity solution in order to get good results in relatively little time, even in large state spaces, by using smart input feature extraction algorithms.

This thesis attempts to improve this agent by using a deep neural network as opposed to an MLP with only a single hidden layer. The reasoning behind this is that adding complexity to the neural network may lead to a higher variety of responses, so that the agent will be able to pick the optimal response in a wider variety of cases. The drawback of this approach is that it may no longer result in a low-complexity solution. We also replace the previous activation function, which was a sigmoid function, with rectifying linear units (ReLUs) where $f(x) = \max(0, x)$. Such a rectifier will reduce the amount of computations necessary by deactivating a subset of all hidden neurons for each input pattern (Glorot et al., 2011). We test all combinations of the two activation functions, sigmoid and ReLU,

and up to four hidden layers. The average score and the win rate of each network type are measured in both the same environment as they were trained in, as well as in a new environment. These are then compared among the different network types and with the results from the previous research. Ultimately, our research question is as follows: will the additions made to the original neural network significantly improve its final performance?

The next section will go over all the different aspects of the experimental setup, by first explaining the environment in which the networks were tested, and the neural network structures. This is followed by a brief explanation of reinforcement learning and how it is applied here, then a short summary of each input algorithm will be presented, and finally how the training and testing of the networks themselves were set up. The third section covers the results from the experiments, along with statistical tests. The fourth section shows the conclusion that can be made from the results, as well as suggestions for further research.

# 2 Methods

## 2.1 Environment

The mazes used in Ms. Pac-Man consist of the following objects: walls, ghosts, pills, power-pills and Ms. Pac-Man. Walls, represented by blue squares, are impassable by both Ms. Pac-Man and the ghosts. Ghosts, of which there are four, should be avoided since they cause the game to end and to be counted as a loss when Ms. Pac-Man collides with one of them. Their exact behaviour will be described further in the next paragraph. Collecting the pills, represented as small yellow dots, is Ms. Pac-Man's main objective, and the game is considered a win when Ms. Pac-Man manages to obtain all of them. Power-pills, represented as big white dots, allow Ms. Pac-Man to safely collide with the ghosts when collected for a limited time. When this happens, the ghost's position is reset to the center and the score is increased, which will be further explained in the reinforcement learning section. Finally, Ms. Pac-Man is controlled by the neural network which is the main focus here.

Ghosts have four states, namely scared, scatter, random and chase. They first spawn at the center of the maze, and use the first moves to exit their spawn point. They then cycle through some of the states every few moves, following this order: chase, random, chase, random, scatter. The chase state causes the ghosts to calculate the shortest path towards Ms. Pac-Man, and follow this path. The random state lets the ghosts move to one of the currently available directions at random. The scatter state causes all ghosts to move to all four corners of the maze by calculating the shortest path and following this. The scared state only activates when Ms. Pac-Man has recently collected a power-pill. In this state, the ghosts will calculate the shortest path towards Ms. Pac-Man and then choose the opposite direction.

The simulation uses the structures of the first two mazes from the original Ms. Pac-Man game to train the network by randomly selecting one of the two for each game. The network is then tested using both the structure of the third maze from Ms. Pac-Man to test the generalization of each network, and the first two mazes. Figures 5.1, 5.2 and 5.3 in the appendix show screenshots of these mazes.

## 2.2 Neural Network Structure

The neural network that was used in the previous research was a multilayer perceptron with 1 hidden layer, and a sigmoid activation function

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

and

$$\sigma'(x) = \sigma(x)(1 - \sigma(x))$$

used when adjusting the weights using backpropagation. Each direction that Ms. Pac-Man can go (up, down, left, right) has its own neural network. These neural networks all have 50 hidden nodes and 1 output node. The direction which ends up having the highest activation of the output node is performed, except when exploring, which is explained in section 2.3. The amount of input nodes depends on the input algorithms being employed, which are explained in section 2.4.

The first contribution we made is the addition of more hidden layers to the network. To train this network, the formula

$$\delta_k = \sigma'(a_k) \sum_{j \in I_k} \delta_j w_{jk}$$

is used, where $k$ is the node to be adjusted, $a_k$ is the activation, $I_k$ is the set of nodes connected to node $k$ and $w_{jk}$ is the weight between nodes $k$ and $j$. Each hidden layer again has 50 hidden neurons.

The second contribution we made is the replacement of the sigmoid function by a rectified linear unit

$$f(x) = \begin{cases} 0, & \text{if } x \leq 0 \\ x, & \text{otherwise} \end{cases}$$

and

$$f'(x) = \begin{cases} 0, & \text{if } f(x) \leq 0 \\ 1, & \text{otherwise} \end{cases}$$

so that the delta rule now becomes

$$\delta_k = f'(a_k) \sum_{j \in I_k} \delta_j w_{jk}$$

When adjusting the weights, $\delta_k$ is multiplied by the learning rate $\alpha$ as follows

$$\Delta w_{jk} = \alpha \delta_k x_j$$

with $x_j$ being the input. The optimal learning rate differs depending on the activation function and the amount of hidden layers.

## 2.3 Reinforcement Learning

The neural network described in the previous section requires some supposedly correct output so that the weights can be adjusted to more closely resemble this output in this research. This output is computed using reinforcement learning (Sutton and Barto, 1998). Reinforcement learning is a type of machine learning paradigm which uses a policy, value function, reward function, environment model and agent model. Each state will give some immediate reward, defined as the reward function. They also each give some potential future rewards depending on the states that follow it, defined by the value function. The reward function is fixed and is loosely based on how the score was calculated in the original Ms. Pac-Man. The rewards corresponding to each event are shown in Table 2.1. The policy determines which action should be taken in which state in order to maximize the value of each state.

The specific type of reinforcement learning used in this study is Q-learning (Watkins and Dayan, 1992). In Q-learning, an agent learns a Q-function

**Table 2.1: Events and their rewards**

| Event | Reward | Description |
|---|---|---|
| Pill | 12 | When Ms. Pac-Man eats a pill. |
| Powerpill | 0 | When Ms. Pac-Man eats a power pill. While it does not give a reward in itself, it allows the Ghost-reward to trigger and in this manner may lead to future rewards. |
| Ghost | 40 | When Ms. Pac-Man collides with a scared ghost. |
| Step | -5 | When Ms. Pac-Man takes a step. |
| Reverse | -6 | When Ms. Pac-Man reverses on her previous route. This will discourage Ms. Pac-Man from only walking back and forth. |
| Win | 50 | When Ms. Pac-Man ate all the pills in the maze. |
| Lose | -375 | When Ms. Pac-Man collides with a non-scared ghost. |

for state-action pairs, and the utility value of a state becomes

$$U(s) = max_a Q(s, a)$$

This Q-function is updated whenever action $a$ is used to reach state $s'$ from state $s$, and goes as follows

$$Q(s, a) \leftarrow Q(s, a) + \alpha(R(s) + \gamma max'_a Q(s', a') - Q(s, a))$$

where $R(s)$ is the reward for state $s$, $\alpha$ is the learning rate and $\gamma$ is the discount factor.

Trying to learn the best policy for playing Ms. Pac-Man like this may result in the agent getting stuck at a local maximum, and then only trying very minor variations. To avoid this, an exploration function was added in the previous research to the reinforcement learning algorithm which could be triggered at any step throughout the game. This function chooses a random direction whenever it is triggered, or the same direction as the previous step if the previous step also triggered the exploration function. The probability that this function is triggered is defined as the exploration rate.

## 2.4 Input Algorithms

A human player would be able to see the entire maze and make decisions based on all kinds of things, like the positions of the ghosts, positions of the pills, etc. The goal then is to analyse what information an agent would need about the environment to make good decisions, and convert this to values that can be used in a neural network, without creating so many representations that the network becomes too difficult to adjust. To this end several input algorithms were created. Each of these algorithms adds four input nodes. These either correspond to a direction, or are copies of the same value. As there are eight input algorithms, this results in 32 inputs in total.

### 2.4.1 Action

It may happen that two directions have almost equal output values, so that slight changes in the input values cause Ms. Pac-Man to repeatedly alternate between the two. Generally speaking, it is more desirable if Ms. Pac-Man chooses one of the two directions and stick with it, as the alternative is that she hardly moves from her position. This will then increase the time to complete the objective unnecessarily. This first algorithm aims to solve this problem, by returning a binary signal for each direction where 1 means that it is currently moving in this direction.

### 2.4.2 Pills

The primary goal of Ms. Pac-Man is finding all pills. The easiest way to do this would be to go to the nearest pill each time until the level is completed. To represent this strategy, this algorithm will attempt to find the nearest pill in each direction. Initially, breadth-first search will be used until the algorithm has reached a depth of 10 without finding any pills, at which point breadth-first search will be switched to A*. Once it has found a distance for each direction, these will be normalised to all be below 1. The algorithm can be summarized with the formula

$$Pills(a) = (b - \alpha(a))/b$$

where

$$a = \text{the direction to be calculated}$$
$$b = \text{maximum distance possible}$$
$$\alpha(a) = \text{shortest path to pills for direction } a$$

### 2.4.3 Ghost Direction

Besides finding all the pills in the maze, it is also important that Ms. Pac-Man avoids collisions with non-scared ghosts. This is represented by the Ghost Direction input, which increases when a situation becomes more dangerous. The danger of a situation is calculated for each direction, by calculating the distance to the nearest non-scared ghost in that direction, since any ghosts further away would reach Ms. Pac-Man after the first ghost when the game is already over, so they would be irrelevant. If there is an intersection between Ms. Pac-Man and the nearest ghost in a direction, it is possible that Ms. Pac-Man could avoid the ghost by turning into the other hallway, so the input also takes the distance to the nearest intersections in account. However, if the ghost has closed of the nearest intersection, then the input for that direction becomes the maximum value. The algorithm then becomes

$$GhostInput(a) = (b + \beta(a) * v - \gamma(a))/b$$

where

$$a = \text{the direction to be calculated}$$
$$b = \text{maximum distance possible}$$
$$v = \text{speed of the ghost relative to Ms. Pac-Man}$$
$$\beta(a) = \text{distance between Ms. Pac-Man and nearest intersection in direction } a$$
$$\gamma(a) = \text{distance between nearest non-scared ghost and nearest intersection in direction } a$$

### 2.4.4 Ghost Afraid

When Ms. Pac-Man eats a Power Pill, ghosts will become scared, which causes them to flee from Ms. Pac-Man at reduced speed, and colliding with them causes them to return to the center of the maze. After a few steps, they will respawn again as non-scared ghosts even if the Power Pill is still in effect.

This is the reason that a distinction was made between scared and non-scared ghosts in the previous algorithm, since both types would require different reactions. This algorithm aims to account for the scared ghosts in much the same way as the Pills-algorithm, so by finding the nearest scared ghost in each direction and normalizing this distance to be between 1 and 0. This can be summarized as

$$GhostAfraid(a) = (b - \delta(a))/b$$

where

$$a = \text{the direction to be calculated}$$
$$b = \text{maximum distance possible}$$
$$\delta(a) = \text{shortest path to scared ghost for direction } a$$

#### 2.4.5 Entrapment

While the Ghost Direction algorithm causes Ms. Pac-Man to flee from nearby ghosts unless the potential rewards are worth the risks, this may not always be enough. Imagine a situation where Ms. Pac-Man turns into a hallway when fleeing from ghost A, only to have ghost B close off the other end while ghost A has now reached the entrance.

To prevent such situations from happening Ms. Pac-Man will attempt to distinguish which actions could leave her trapped. This is done by calculating for each intersection on the map for how long it takes for the nearest ghost and for Ms. Pac-Man to reach it. Every intersection that Ms. Pac-Man can reach before any ghost is stored. If a series of intersections from this list can get Ms. Pac-Man at least three intersections away, it is deemed safe. If there are none that fit this criteria, the number of intersections is lowered to two, etc. This is calculated for each direction, and the portion of unsafe paths in each direction is used as the input, so that this algorithm reflects how dangerous a given direction is. This can be summarized as

$$Entrapment(a) = (c - \epsilon(a))/c$$

where

$$a = \text{the direction to be calculated}$$
$$c = \text{total number of intersections}$$
$$\epsilon(a) = \text{number of safe routes in direction } a$$

#### 2.4.6 Pills Eaten

Because the main goal is still collecting all the pills in the maze, Ms. Pac-Man might consider more dangerous situations if there are less pills left, since the game ends when all the pills are collected. This algorithm tries to give Ms. Pac-Man some sense of the level progression, using the amount of pills left and the total amount of pills at the start. This is the one of two algorithms that does not depend on the direction, and thus is calculated as follows, along with its normalization:

$$PillsEaten = \sqrt{1 - d/e}$$

where

$$d = \text{the amount of pills left}$$
$$e = \text{the total amount of pills}$$

#### 2.4.7 Power Pill Duration

While it is generally advantageous for Ms. Pac-Man to ignore scared ghosts or even pursue them for a short while, the advantage may be minor to non-existent if the Power Pill is just about to wear off. For instance, chasing a ghost that is 2 steps away, when the Power Pill wears of in the next step will just leave Ms. Pac-Man in danger without any possible payoff. So the remaining time of a Power Pill should also be used when deciding the next step. This algorithm therefore calculates the portion of time left, which becomes 0 if the last Power Pill has worn off or when there are no more scared ghosts. Like the Pills Eaten algorithm, this one does not use the directions. This is calculated as

$$PowerPillDuration = \begin{cases} 0, & \text{if f = 0 or g = 0} \\ f/h, & \text{otherwise} \end{cases}$$

where

$$f = \text{Power Pill duration left}$$
$$g = \text{number of scared ghosts}$$
$$h = \text{Power Pill total duration}$$

#### 2.4.8 Pills Quadrant

When looking at the visualisation of a trained network with the original inputs, one thing that be-

comes noticeable is how Ms. Pac-Man may sometimes get stuck in one corner, while the remaining pills are still in some other corner of the maze. A possible explanation of this behavior is that some of the Pills-input nodes have too similar activations so that the output constantly switches between directions.

This input algorithm, which was not present in the original model, aims to solve this problem. It divides the maze into a $5 \times 5$ grid and then calculates for each area what share of the remaining pills are still contained in that area. Each area then adds this share to the input corresponding to its direction as seen from Ms. Pac-Man, so that all areas above her current area add to the up-input, etc. Areas can also add their share to multiple directions.

At the start of a game, it is preferable that Ms. Pac-Man first starts with the pills close to her, although this would decrease the input of that area from the Pills Quadrant algorithm, while later on in the game it would be better if Ms. Pac-Man went to the largest concentrations of pills. To allow this, the algorithm also multiplies the input with the current step divided by 100, so that the input increases over time, until there have been 100 steps. At that point the input is always multiplied by 1. This algorithm can be summarized with the formulas

$$share(x) = \zeta(x)/d$$

$$PillsQuadrant(a) = i/100 * \sum_{x \in j} share(x)$$

where

$$a = \text{the direction to be calculated}$$
$$\zeta(x) = \text{the pills in area } x$$
$$d = \text{the amount of pills left}$$
$$i = \text{the current step}$$
$$j = \text{all areas in direction } a$$

where $d/100$ is changed to 1 after step 100.

## 2.5 Experimental Setup

The experiment consists of two phases, the training phase and the testing phase. At the end of every game, the total percentage of pills that had been collected were recorded. There were eight different types of network that were tested, namely every combination of either sigmoid or ReLU activation functions and either 1, 2, 3 or 4 hidden layers. Each type of network was trained and tested 10 times. The results collected during training were put in a graph, with the number of games along the x-axis, the score along the y-axis and a line showing the average score for each game. For the results collected during testing, the average and standard deviation of the scores as well as the amount of games won, where a win is defined as a score of 100%, were calculated. These were then compared between all different types and the original from the previous research, so a network with sigmoid function, one hidden layer and all input algorithms except Pills Quadrant. This was done with two two way ANOVAs, one for the average score and one for the win rate.

**Table 2.2: Optimal learning rates**

| Hidden layers | sigmoid | ReLU |
|---|---|---|
| 1 | 0.00015 | 0.00015 |
| 2 | 0.00015 | 0.00005 |
| 3 | 0.00015 | 0.00001 |
| 4 | 0.00015 | 0.000005 |

In the training phase, 10.000 games were played per network and at the start of the first game the weights of the network were initialized at random between -0.3 and 0.3. Table 2.2 shows the learning rates that were used in the simulations. These values were optimized through pilot experiments. The exploration rate was set to 40% and diminished equally with each step until 0 at game 4500, or $40/4500 = 0.0089\%$ per step. The mazes used were maze A and B from figures 5.1 and 5.2 respectively. The discount factor was 0.95.

The testing phase consisted of two parts, with one part using the original two mazes, and the other using a new third maze. In both of these parts, 5.000 games were played. The weights were set to their values at the end of the training phase. The maze used for the second part was maze C from figure 5.3 to evaluate the generalisation of the networks. For both testing phases the exploration rate was set to 0% and The learning rate was set to 0.

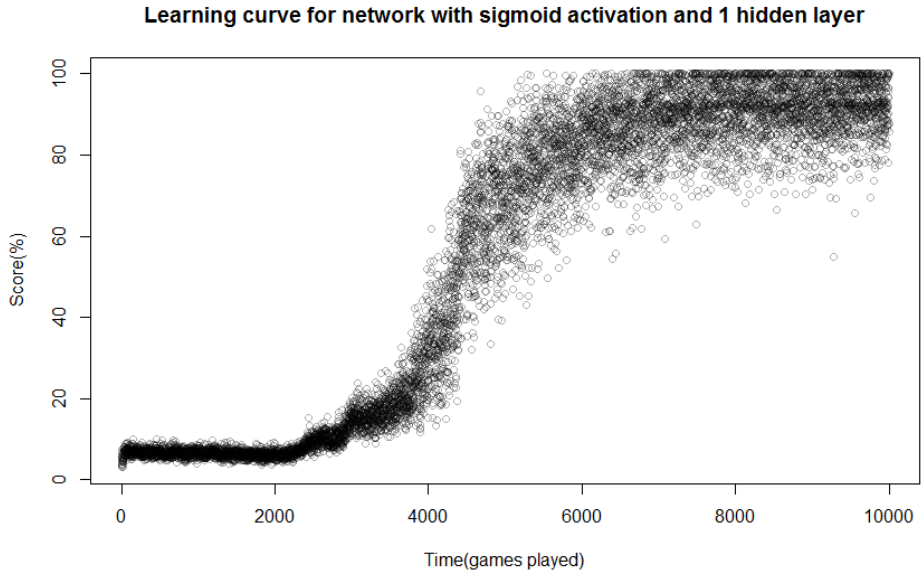


**Figure 3.1: Results during training of a sigmoid-network with one hidden layer**

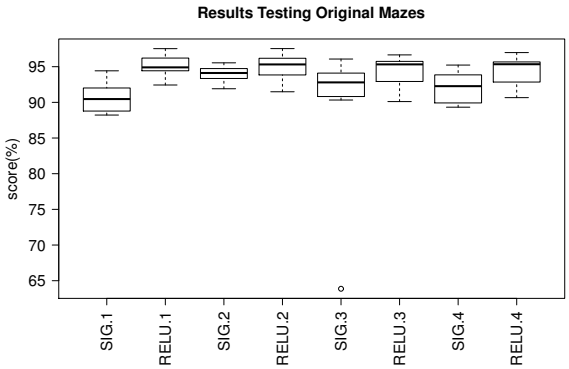


**Figure 3.2: Boxplots of the average scores from the testing phase**

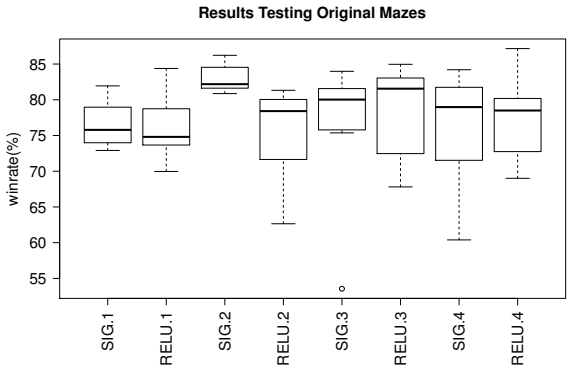


**Figure 3.3: Boxplots of the win rates from the testing phase**

# 3 Results

## 3.1 Training Results

Figure 3.1 shows an example of training results. In this case, a sigmoid activation was used, along with one hidden layer. The dots each indicate the average score of that particular game over all 10 networks of that type. Figures 5.4, 5.5, 5.6, 5.7, 5.8, 5.9 and 5.10 in the Appendix show the results for all other types of networks

## 3.2 Test Results

Table 3.1 shows the results from the testing phase, in which the trained neural networks were used on the same two mazes as in the training phase. Figure 3.2 shows box plots concerning the average scores for each neural network type. There are two independent categorical groups and one independent variable, so a two-way ANOVA was used to detect any noteworthy effects. A significant effect of the activation function on the average score was

**Table 3.1: Results testing original mazes**

| | sigmoid | | ReLU | |
|---|---|---|---|---|
| **Hidden layers** | **Average completion** | **Win rate** | **Average completion** | **Win rate** |
| 1 | 90.7% (SD 1.99) | 76.5% | 95.0% (SD 1.69) | 76.2% |
| 2 | 94.1% (SD 1.06) | 82.9% | 95.1% (SD 1.76) | 75.7% |
| 3 | 90.1% (SD 9.4) | 77.1% | 94.5% (SD 2.07) | 78.7% |
| 4 | 92.1% (SD 2.21) | 75.9% | 94.5% (SD 2.16) | 77.2% |

found ($F(1, 72) = 13.1, p < 0.001$). Taking figure 3.2 into consideration, this would mean that networks which use a ReLU activation function perform considerably better than those that use a sigmoid activation function. However, there was no obvious interaction effect between the number of layers and the activation function used ($F(3, 72) = 0.963, p = 0.41$), nor was there a significant effect of the number of layers ($F(3, 72) = 1.34, p = 0.27$).

Figure 3.3 shows boxplots of the win rates of each neural network type. Another two-way ANOVA was used to detect considerable effects on the win rates, but there was no significant effect of either the amount of hidden layers ($F(3, 72) = 1.17, p = 0.33$) or the activation function used ($F(1, 72) = 0.817, p = 0.37$), nor of the interaction between the two ($F(3, 72) = 2.54, p = 0.063$).

## 3.3 Policy Transfer

Table 3.2 shows the results from the policy transfer tests, in which the trained neural networks were used in a new unseen maze. Figure 3.4 shows box plots concerning the average scores for each neural network type. For the same reasons as described in section 3.2 a two-way ANOVA was used to check for effects. A noteworthy interaction effect was discovered between the number of hidden layers and the type of activation function used ($F(3, 72) = 5.08, p = 0.0030$). Separate one-way ANOVAs for

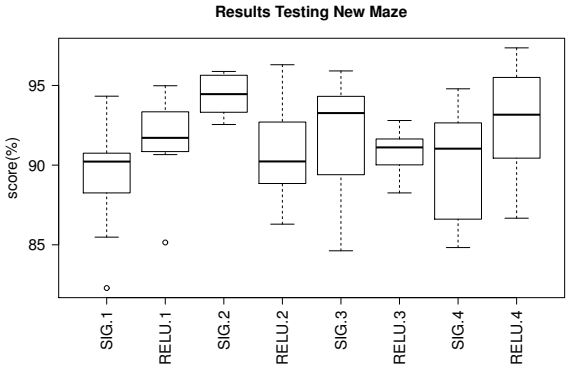


**Figure 3.4: Boxplots of the results from the policy transfer tests**

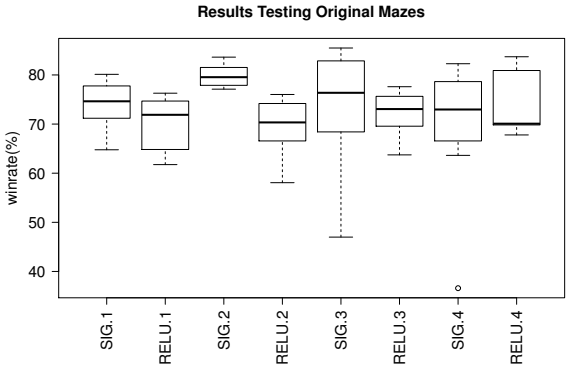


**Figure 3.5: Boxplots of the win rates from the policy transfer tests**

both types of activations showed a significant effect of the amount of layers for networks with a sigmoid activation function ($F(3, 36) = 5.16, p = 0.004$), but not for networks which use ReLUs ($F(3, 36) = 1.17, 0.34$). Pairwise one-tailed t-tests for the networks with sigmoid activation functions showed only considerable differences between one hidden layer and two hidden layers ($p = 0.0027$), and four and two hidden layers ($p = 0.011$).

Figure 3.5 shows boxplots of the win rates for each neural network type. Another two-way ANOVA indicates obvious effects of the interaction between both the amount of hidden layers and the activation function used ($F(3, 72) = 3.02, p = 0.035$). However, separate one-way ANOVAs of the win rates grouped by the activation functions did not show a significant effect of the amount of hidden layers for sigmoid activation networks ($F(3, 36) =$

**Table 3.2: Results policy transfer**

| | sigmoid | | ReLU | |
|---|---|---|---|---|
| **Hidden layers** | **Average completion** | **Win rate** | **Average completion** | **Win rate** |
| 1 | 89.4% (SD 3.35) | 73.8% | 91.5% (SD 2.62) | 70.1% |
| 2 | 94.5% (SD 1.19) | 79.9% | 90.8% (SD 2.86) | 69.6% |
| 3 | 91.6% (SD 4.05) | 73.5% | 90.8% (SD 1.44) | 72.0% |
| 4 | 90.1% (SD 3.33) | 69.7% | 92.8% (SD 3.44) | 73.5% |

**Table 3.3: Results original research**

| testing | | policy transfer | |
|---|---|---|---|
| **Average completion** | **Win rate** | **Average completion** | **Win rate** |
| 85.6% (SD 5.7) | 50.3% (SD 8.21) | 74.4% (SD 8.8) | 35.9% (SD 8.61) |

$2.14, p = 0.11$) or for ReLU activation networks ($F(3, 36) = 1.10, p = 0.36$).

## 3.4 Original Results

The scores from the neural network from the previous research are shown in table 3.3. Comparing these statistics to those of the other networks for the testing phase using a one-tailed t-test, the results of which are seen in table 3.4, shows that the original neural network does not score considerably less than a sigmoid network with three hidden layers. It does score less than all the other networks. Repeating this for the policy transfer shows that the original network had a significantly lower average score than all of the networks from this research. Furthermore, table 3.5 shows the results of all one-tailed t-tests between the average win rate of the original network and that of each network from this research. From the p-values it can be concluded that the win rate of every new network is significantly higher than that of the original network.

# 4 Conclusion

## 4.1 Consequences

From the results of the tests on the original mazes, it has become clear that ReLU activation networks will obtain a significantly higher score than sigmoid activation networks when tested in the same environment as they were trained on. There is no noteworthy difference between the average scores when varying the number of hidden layers. There is also no obvious difference between win rates, despite the higher average scores of the ReLU activation networks.

The results from the policy transfer tests are less clear. There is a significant interaction effect between the amount of hidden layers and the activation function, but not of either variable on its own, so certain combinations result in better scores than others but this depends on both the activation used and the number of hidden layers, and not just one of the two. Within the group of sigmoid activation networks, one that uses two hidden layers performs significantly better than one with one or four hidden layers. Unfortunately, no such distinction can be found within the group of ReLU activation networks. As such, it could be said that a sigmoid activation network with two hidden layers would be the safest bet when the test environment does not match the training environment, considering its higher mean and lower variability than other settings. In terms of win rate, there is no clear winner, although the sigmoid activation networks with two hidden layers again have the lowest variability.

As for the research question, there is a clear difference between almost all networks and the original networks from the previous research when testing takes place in the same environment as training. The only exception is the sigmoid activation network with three hidden layers, although like all other networks, it does obtain a significantly higher win rate. This shows the effectiveness of the added input algorithm. This becomes clearer when testing the networks in a new environment, where each type of network clearly outperforms the original networks. This is likely due to the environment being represented in more detail, so that the agent can

**Table 3.4: Results of all one-tailed t-tests versus the average scores of the original research**

| | testing | | policy transfer | |
|---|---|---|---|---|
| | **t** | **p** | **t** | **p** |
| Sigmoid, 1 hidden layer | 2.14 (df 5.75) | 0.039 | 3.98 (df 5.87) | 0.0038 |
| Sigmoid, 2 hidden layers | 3.67 (df 5.21) | 0.0067 | 5.42 (df 5.11) | 0.0012 |
| Sigmoid, 3 hidden layers | 1.21 (df 14.0) | 0.12 | 4.50 (df 6.29) | 0.0018 |
| Sigmoid, 4 hidden layers | 2.71 (df 5.94) | 0.018 | 4.17 (df 5.86) | 0.0031 |
| ReLU, 1 hidden layer | 4.00 (df 5.54) | 0.0042 | 4.63 (df 5.53) | 0.0022 |
| ReLU, 2 hidden layers | 4.01 (df 5.59) | 0.0041 | 4.41 (df 5.63) | 0.0026 |
| ReLU, 3 hidden layers | 3.73 (df 5.82) | 0.0051 | 4.53 (df 5.16) | 0.0029 |
| ReLU, 4 hidden layers | 3.73 (df 5.94) | 0.0050 | 4.88 (df 5.92) | 0.0014 |

**Table 3.5: Results of all one-tailed t-tests versus the win rates of the original research**

| | testing | | policy transfer | |
|---|---|---|---|---|
| | **t** | **p** | **t** | **p** |
| Sigmoid, 1 hidden layer | 8.10 (df 7.06) | $< 0.001$ | 6.77 (df 9.05) | $< 0.001$ |
| Sigmoid, 2 hidden layers | 10.3 (df 6.49) | $< 0.001$ | 12.3 (df 5.42) | $< 0.001$ |
| Sigmoid, 3 hidden layers | 6.47 (df 13.6) | $< 0.001$ | 7.51 (df 12.9) | $< 0.001$ |
| Sigmoid, 4 hidden layers | 6.59 (df 12.1) | $< 0.001$ | 6.20 (df 13.8) | $< 0.001$ |
| ReLU, 1 hidden layer | 7.68 (df 8.15) | $< 0.001$ | 8.74 (df 7.40) | $< 0.001$ |
| ReLU, 2 hidden layers | 7.06 (df 10.1) | $< 0.001$ | 8.51 (df 7.73) | $< 0.001$ |
| ReLU, 3 hidden layers | 7.66 (df 10.9) | $< 0.001$ | 9.48 (df 6.75) | $< 0.001$ |
| ReLU, 4 hidden layers | 7.60 (df 9.58) | $< 0.001$ | 9.38 (df 8.08) | $< 0.001$ |

use more information and becomes more adaptable.

In summary, every network outperforms the original networks, even the sigmoid activation networks with three hidden layers, given that the environment is different from training or when looking at the win rate rather than the average score. If the environment were kept unchanged, the ReLU activation networks would clearly have the upper hand, while in a new environment the best network would be somewhat more difficult to decide. In that case, the safest option would be a sigmoid activation network with two hidden layers.

## 4.2 Future Research

While this research only covers a small section of possible types of neural networks that can be used, many variations are still conceivable, like ELU activation functions (Clevert et al., 2015). The network can also be expanded upon, with more hidden layers or more nodes per layer, although this will slow simulations down considerably without actually improving anything, as can be seen from the performances of the networks with four hidden layers. The input algorithms could also be improved or expanded upon, although this goes against the original research, which set out to represent the environment of Ms. Pac-Man with only a small number of simple inputs while still getting a good performance. Finally, this research suffers somewhat from a ceiling effect, as the networks often achieve a perfect score in a game. In order to continue making meaningful comparisons between networks, the mazes used should become bigger or the game itself should become more difficult so that networks become less likely to get 100% on a maze.

# References

L. Bom, R. Henken, and M. Wiering. *Reinforcement learning to train Ms. Pac-Man using higher-order action-relative inputs*, pages 156–163. IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL). 2013.

P. Bosilj, P. Palasek, B. Popovic, and D. Stefic. *Simulation of a Texas Hold'Em poker player*,

pages 1628–1633. Proceedings of the 34th International Convention MIPRO. 2011.

L. Cardamone, D. Loiacono, and P. L. Lanzi. Learning to drive in the open racing car simulator using online neuroevolution. *IEEE Transactions on Computational Intelligence and AI in Games*, 2(3):176–190, 2010.

D.-A. Clevert, T. Unterthiner, and S. Hochreiter. Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs). *ArXiv e-prints*, Nov. 2015.

L. Galway, D. Charles, and M. Black. Machine learning in digital games: a survey. *Artificial Intelligence Review*, 29(2):123–161, 2008.

X. Glorot, A. Bordes, and Y. Bengio. Deep sparse rectifier neural networks. *Journal of Machine Learning Research*, 15:315–323, 2011.

H. Handa and M. Isozaki. Evolutionary fuzzy systems for generating better Ms. Pac-Man players. *IEEE International Conference on Fuzzy Systems*, pages 2182–2185, 2008.

I. Makarov, P. Zyuzin, P. Polyakov, M. Tokmakov, O. Gerasimova, I. Guschenko-Cheverda, and M. Uriev. Modelling human-like behavior through reward-based approach in a first-person shooter game. *CEUR Workshop Proceedings*, 1627:24–33, 2016.

V. Mnih, K. Kavukcuoglu, D. Silver, A. Rusu, J. Veness, M. Bellemare, A. Graves, M. Riedmiller, A. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–33, 2015.

S. Samothrakis, D. Robles, and S. Lucas. Fast approximate max-n Monte Carlo tree search for Ms Pac-Man. *IEEE Transactions on Computational Intelligence and AI in Games*, 3(2):142–154, 2011.

J. Schaeffer, J. Culberson, N. Treloar, B. Knight, P. Lu, and D. Szafron. A world championship caliber checkers program. *Artificial Intelligence*, 53(2):273–289, 1992.

R. S. Sutton and A. G. Barto. *Reinforcement learning : an introduction*. MIT Press, 1998.

I. Szita and A. Lõrincz. Learning to play using low-complexity rule-based policies: Illustrations through Ms. Pac-Man. *Journal of Artificial Intelligence Research*, 30:659–684, 2007.

I. Szita, M. Ponsen, and P. Spronck. Effective and diverse adaptive game AI. *IEEE Transactions on Computational Intelligence and AI in Games*, 1 (1):16–27, 2009.

G. Tesauro. Programming backgammon using self-teaching neural nets. *Artificial Intelligence*, 134 (1-2):181–199, 2002.

C. J. Watkins and P. Dayan. Technical note: Q-learning. *Machine Learning*, 8(3):279–292, 1992.

# 5 Appendix

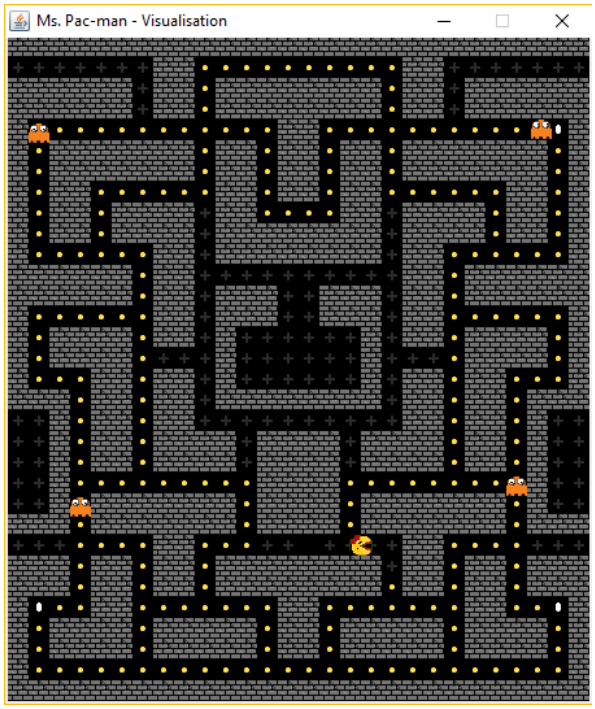


**Figure 5.1: Screenshot of maze A, which is used in the training and testing phase**

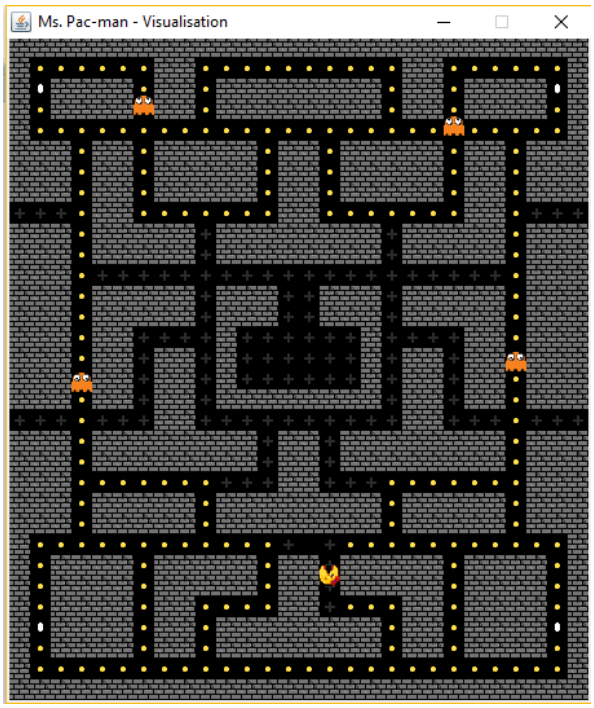


**Figure 5.2: Screenshot of maze B, which is used in the training and testing phase**

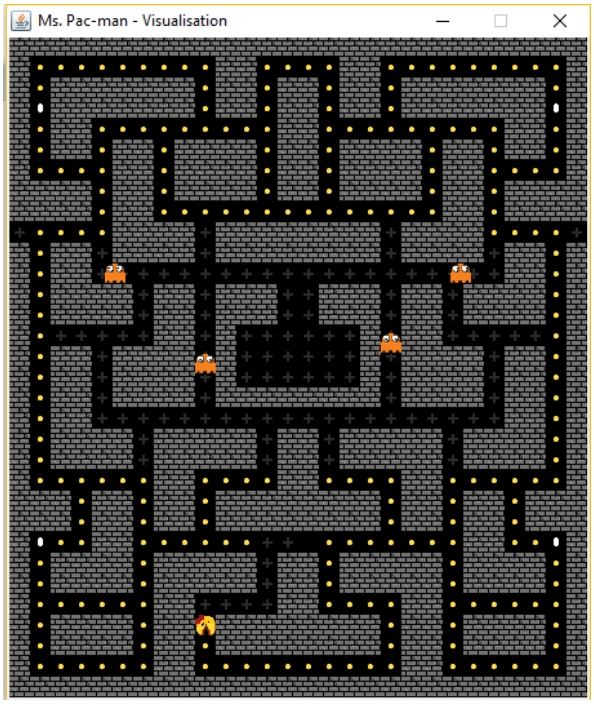


**Figure 5.3: Screenshot of maze C, which is used in the testing phase to test the policy transfer**

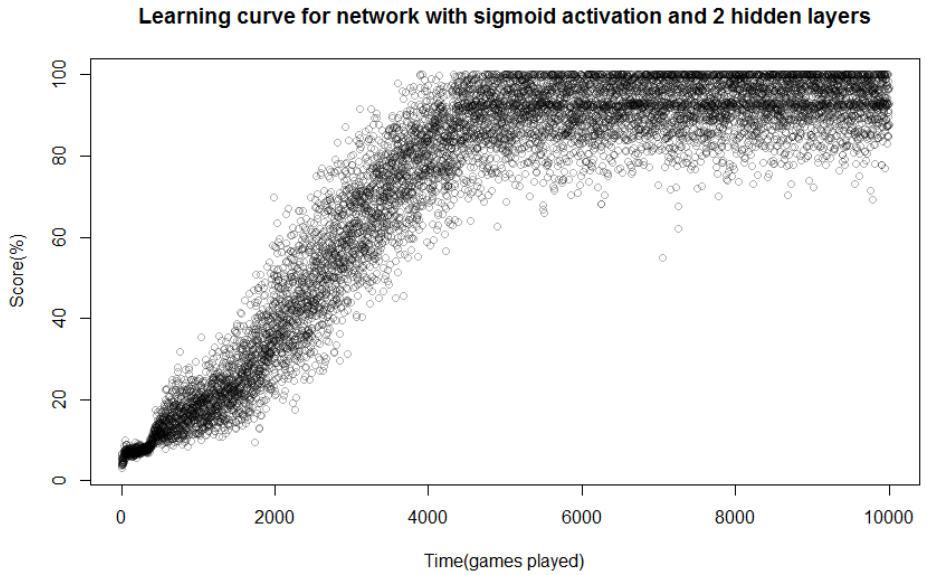


**Figure 5.4: Results during training of a sigmoid-network with two hidden layers**

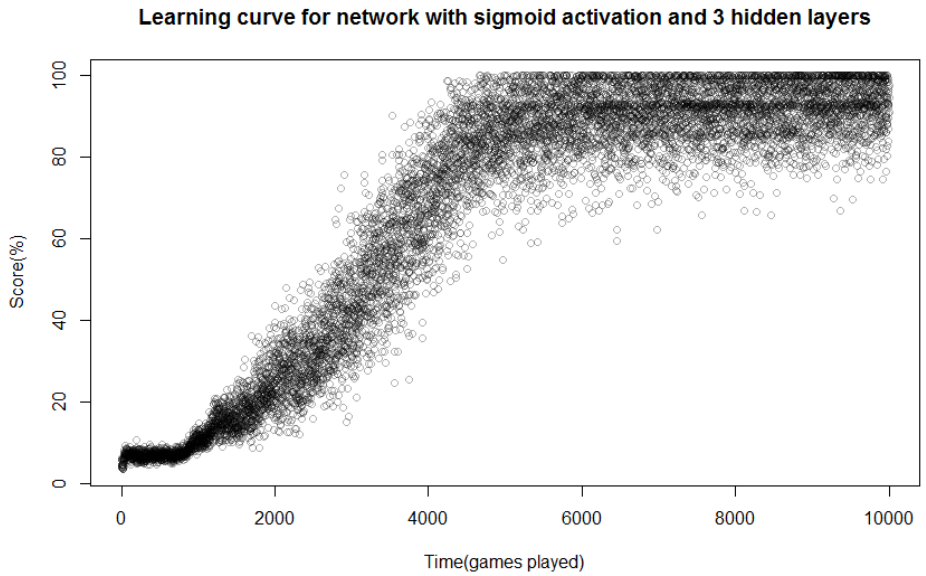


**Figure 5.5: Results during training of a sigmoid-network with three hidden layers**

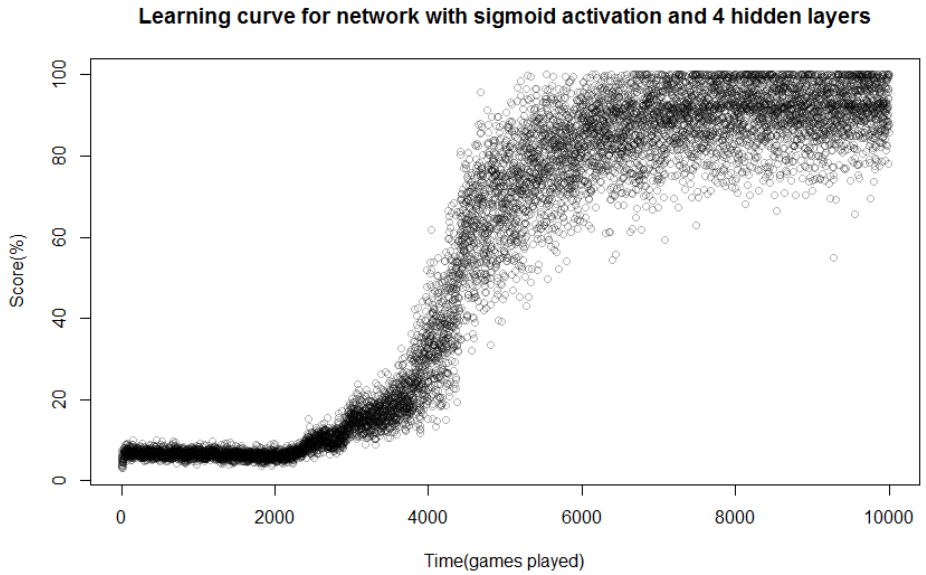


**Figure 5.6: Results during training of a sigmoid-network with four hidden layers**

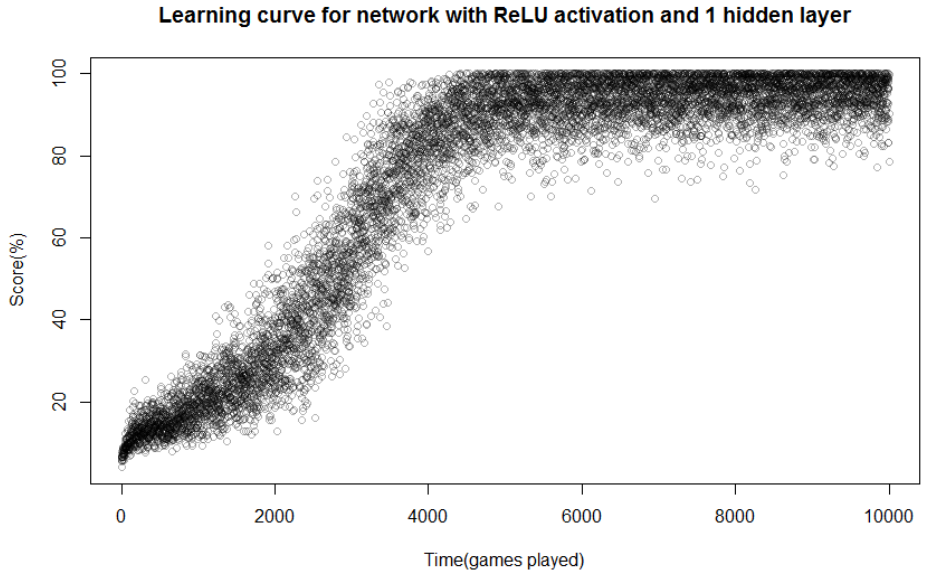


**Figure 5.7: Results during training of a ReLU-network with one hidden layer**

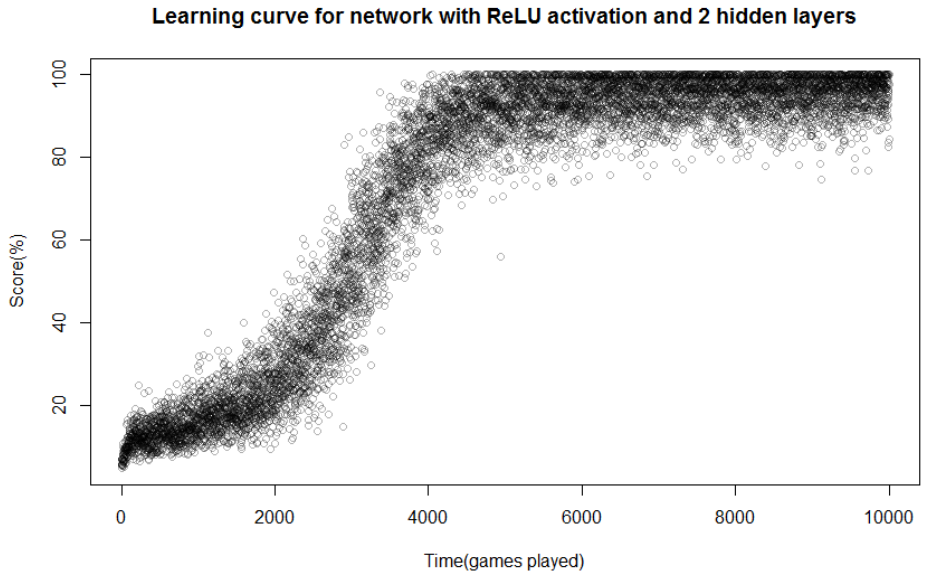


**Figure 5.8: Results during training of a ReLU-network with two hidden layers**

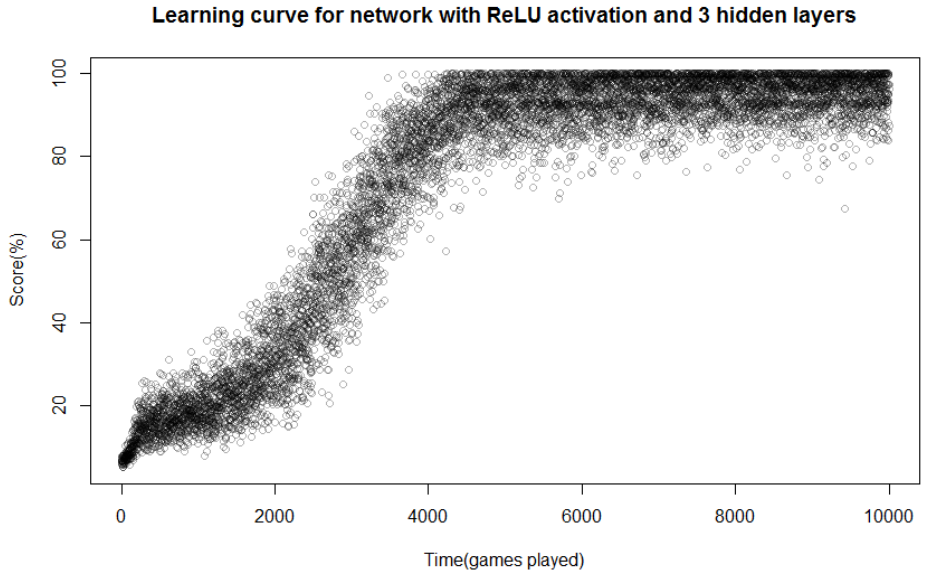


**Figure 5.9: Results during training of a ReLU-network with three hidden layers**

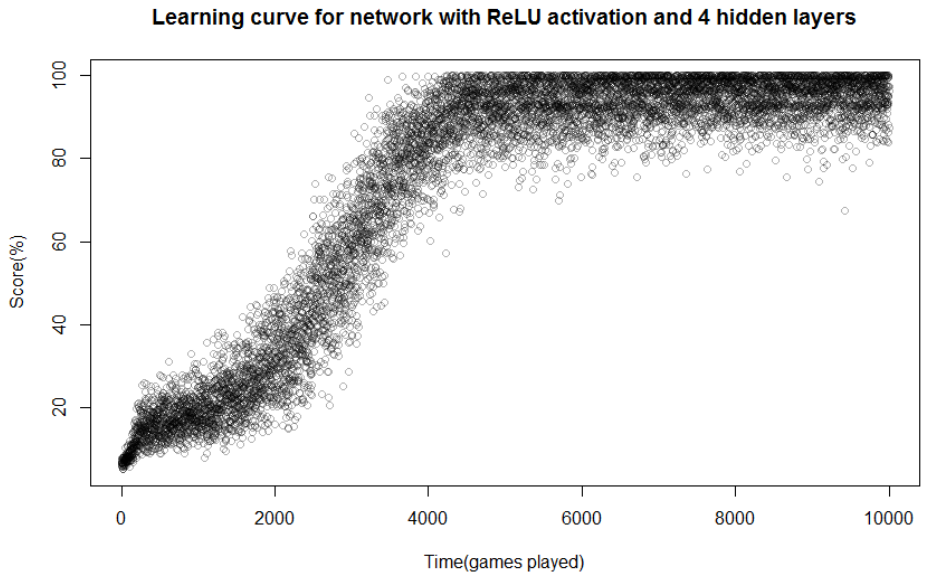


**Figure 5.10: Results during training of a ReLU-network with four hidden layers**