university of groningen / faculty of science and engineering

Reinforcement learning for the game of soccer in flexible environments

Bachelor's Project Thesis

Julian Hegeman
Yujin Kim
Supervisor: dr. Marco A. Wiering

Abstract: The soccer game is one of the most popular sport games. Even though the game is easy to play for humans, it requires a wide range of intelligent abilities. With this lens, the game of soccer is a very intriguing topic in Artificial Intelligence. In this research, we apply reinforcement learning techniques, which have been successful in various games, such as Go and Atari. Using these techniques, we focus on whether it is possible to design a system that performs in simulations with multiple team and field sizes at an equal level as a multilayer perceptron (MLP) based Q-learning agent that receives information from all points in a grid modeling the game of soccer and is trained specifically on one field size. This reference player is also trained on varying team sizes. The results illustrate that the vision grid player performs better than the reference player on all tested setups except the one with the smallest team and field size (9 by 9 cells and 1 player per team). The vision grid player is therefore flexible regarding the size of the field it plays in and the number of players it plays with, and able to outperform the reference player that is only trained using a varying number of players per team. The influence of the activation function in the hidden layer of the MLP is also researched. It is found that for this simulation, the sigmoid activation function works best.

1 Introduction

The game of soccer is recognized as a game that is easy to play for humans but the cognitive processes of the game player are not as simple as what we imagine. The players are required to have a wide range of intelligent abilities. It is not only important to have the ability to monitor the game consistently, but also to make the best decision in dynamic situations. Moreover, each player should consider speed, space, angles, nearby opponents, and available teammates.

This perspective shows that the game of soccer has a variety of different possible configurations and a dynamic environment, that is hard to predict. This makes the game an intriguing topic to the field of artificial intelligence, as is shown by numerous research papers such as those regarding the Robocup soccer competition (Macalpine, Torabi, Pavse, Sigmon, and Stone, 2019) and other soccer simulations (Littman, 1994). Laird also suggests that the creation of software agents, that obtain good results in dynamic environments, provides novel insights to the field of artificial intelligence (Laird, 2001).

In reality, the game of soccer has too many possible configurations and considerations to achieve its goal so we will cut these down in a discrete environment in this experiment. In this discrete environment, the action set will also be discrete.

To train the agents in the virtual game of soccer, we apply reinforcement learning techniques which have been successful in various games such as Go (Silver, Schrittwieser, Simonyan, Antonoglou, Huang, Guez, Hubert, Baker, Lai, Bolton, Chen, Lillicrap, Hui, Sifre, van den Driessche, Graepel, and Hassabis, 2017) and the Atari 2600 games (Mnih, Kavukcuoglu, Silver, Rusu, Veness, Bellemare, Graves, Riedmiller, Fidjeland, Ostrovski, Petersen, Beattie, Sadik, Antonoglou, King, Kumaran, Wierstra, Legg, and Hassabis, 2015). The purpose of this experiment is to observe whether we can design a system with a changing number

of players and size of the field which performs at a similar level as a multilayer perceptron (MLP)-based Q-learning agent that receives information from all cells in a grid modeling the game of soccer. For this goal, a vision grid agent was designed (Knegt, Drugan, and Wiering, 2018). The same vision grid agent will be able to play with a variety of team and field sizes as the size of the input to its MLP is independent of both. This approach will enable us to train and test the same agents in flexible environments which have multiple situations with different field and team sizes.

Besides, the performance of using the rectified linear unit (ReLU) function and of using the sigmoid function as the activation function for the hidden layer in the MLP.

In section 2, we provide a brief background of the learning approaches such as Q-learning, MLPs and vision grids. Then, we will explain the reason why we apply these methods. In section 3, we present a detailed state representation of our soccer game and its experimental setup. Section 4 demonstrates our results and the paper is concluded in section 5.

2 Reinforcement Learning

Reinforcement learning is a machine learning method that allows agents to learn to optimize their behavior through its interaction with an environment automatically (Sutton and Barto, 2015).

The agent initiates from an unknowing state in which it doesn't have any background knowledge about its environment. After that, it learns the way to choose and execute the optimal action by receiving rewards from the interaction with its environment over many time steps. The main purpose of the agent is to maximize the cumulative discounted reward sum which is also called the return.

$$R_t = \sum_{i=0}^{T-t} \gamma^i r_{t+i} = r_t + \gamma R_{t+1} \tag{2.1}$$

Where $\gamma \in [0,1]$ is the *discount factor* and T is the *Time*, at which the current episode ends.

At every time step t, the agent updates its beliefs using its observation of the reward and state that follows choosing a particular action in a particular state. It will also pick a new action based on the current state.

2.1 Q-learning

One approach for learning the optimal policy is to apply Q-learning (Watkins, 1989). Q-learning is based on the use of Q-values. A Q-value is the expected sum of rewards the player expects to receive after choosing an action a in a state s. Therefore, each combination of an action and a state available in the game has an associated Q-value for the player.

Q-learning agents use a function to obtain their estimated Q-value from the state and action under consideration. This function is referred to as the Q-function and often written as $Q(s, a)$, where s is a state, a is an action and $Q(s, a)$ is the estimated Q-value for that state-action combination.

Through exposure to the world and the results of its actions, the estimates of the future sum of rewards will move closer to the actual average sum of rewards an agent will receive for that state-action combination.

This is done by generating a new Q-value estimate at the next time the player was prompted to act again, using the reward it received directly after performing its action and the sum of future accumulated rewards the agent is expected to receive from being in the new state. This sum of rewards is most often taken to be the maximum of the Q-values for all actions available in that state.

Using the maximum Q-value makes this approach off-policy as the action is not only decided by finding the action with the maximum Q-value but often also uses an exploration technique to improve learning. There are other ways to obtain the new Q-value estimate, such as State-action-reward-state-action (SARSA) (Singh, Jaakkola, Littman, and Szepesvari, 2000). However, this was not elaborated upon in this paper.

This new estimate is found using the formula in equation 2.2, where e is the new estimate, r_{t-1} is the reward received after the previous action, s_t is the current state and a is used to iterate over all the actions.

$$e = r_{t-1} + \gamma \max_a Q(s_t, a) \tag{2.2}$$

The Q-function is then updated to move the estimate it generates, closer to this new estimate.

There exist multiple Q-functions, that have specific ways of generating and updating their Q-value

estimates.

One of these Q-functions is based on a lookup table in which for each state-action combination, the Q-value estimates are stored separately. To find a Q-value - the expected sum of future rewards - for a given state and action, the Q-function just looks up the value in the corresponding cell of the table. The update works similarly.

After updating, the updated Q-value is a weighted average of the new Q-value estimate that is generated based on the received reward and current state, and the previous Q-value, that was stored in the table.

The weights of the weighted average are based on the provided learning rate. This weighted average is shown in equation 2, where α is the learning rate, e is the generated Q-value estimate, the $Q(s_{t-1}, a_{t-1})$ on the right side is the previous Q-value and the $Q(s_{t-1}, a_{t-1})$ on the left side is the updated Q-value.

$$Q(s_{t-1}, a_{t-1}) \leftarrow (1-\alpha) * Q(s_{t-1}, a_{t-1}) + \alpha * e \quad (2.3)$$

Another type of Q-function is based on a neural network. It does not store the Q-values for each state-faction combination explicitly. Rather, it uses its neural network to generate a Q-value for each action by providing a number of values, that represent the state the agent is in, as input. To update its estimates, it finds the output neuron that corresponds to the action it chose and sets the target of that neuron to the new estimate. It then performs backward propagation to update the network, such that the output moves closer to this target. Since a new estimate can only be found for the action that was chosen, the Q-function does not have new targets for the other output neurons. Therefore, these targets are simply set to the value that neurons put out.

In this research, the latter approach is used as the number of states possible in the simulation is assumed to be too large for the former to learn in a reasonable amount of time. The neural network used is an MLP, a relatively simple fully connected neural network.

If the input size is big and the input values are spatially related, it has been shown that a Q-function based on a convolutional neural network might perform better (Mnih et al., 2015). However, this was not explored in this research.

2.2 Multilayer Perceptron (MLP)

As discussed earlier tabular Q-learning stores a Q-value for each state-action combination. The large state space requires this approach to need more memory then is feasible. Moreover, there is such a variety of states that all need to be visited multiple times to obtain correct Q-values that learning would take too much time. To solve this problem, we apply an MLP to Q-learning which means that state-action combinations do not have individually stored Q-values.

An MLP is a feed-forward neural network, so the Q-value estimates are approximated by the network when demanded through giving a state representation as input and yielding Q-values for all actions as output.

The detailed procedure is the following. Firstly, it runs the forward propagation using the representation of the state as input. Then, the targets are set for the backward propagation. Because all Q-values are approximated at the same time, targets have to be provided for each action. However, since we only want to update one estimate, the targets are set equal to estimates that were just produced for all actions except the action that has to be updated. This target is set to the newly founded Q-value estimate. After that, the backward propagation is used to move the estimate of the state-action combination closer to the newly found estimate.

2.3 Function Approximator

2.3.1 Activation Function

Each neuron in the neural network receives input from the neurons in the previous layer and outputs a value. This is output is used as input to the neurons in the next layer if the neuron is hidden, or as a final resultant value of the network if the neuron is part of the output layer.

The sum of the values coming in is not necessarily equal to the value the neuron puts out. Each neuron uses a function to derive the value to put out from the input value. This function is called the activation function.

In this research, three different activation functions are used. In the first, the output value is equal to the input value. This type of activation function is known as the linear unit activation function. It is used in the output layer so as not to limit the range

of the output and therefore the range of the estimates. This function is restricted in its complexity so that it has limited power to learn complicated functional mappings from the input.

The hidden neurons, on the other hand, used non-linear activation layers as it makes it possible for the network to learn and perform more complex tasks than linear regression.

Two different kinds of non-linear activation functions were tested: ReLU and sigmoid.

Firstly, ReLU, short for rectified linear unit function, is the most successful and widely-used activation function (Prajit, Barret, and Quoc, 2017). The range of the ReLU exists between zero and positive infinity. It is equal to the linear unit function for values greater than or equal to zero, while it outputs zero for all input values lower than zero. The ReLU activation function can increase the learning speed of the networks by avoiding the vanishing gradient problem which leads the updates by the stochastic gradient to be very small (Hidenori and Takio, 2017).

The other one is the sigmoid function. This function transforms the weighted sum of inputs for a hidden neuron to a value between 0 and 1.

2.3.2 Reward Function

Reinforcement learning (RL) agents alter their behavior based on the rewards they obtain. The reward function indicates the amount of reward the agent will receive for every possible state. While designing an RL system, a reward function can be chosen. Since the agents alter their behavior to optimize the rewards they will obtain, the choice can be made based on the desired behavior. In this research, the agents are desired, like in the real game, to win the matches they play. This means that they have to maximize the number of goals their team scores and minimize the amount of goals the opposing team scores.

The used rewards for the experiment are shown in table 2.1.

The value of the rewards has been decided to distinguish between positive and negative actions clearly. If the type of the goal is an own goal or the goal is made by the opposite team, the punishing reward -1 is given. Otherwise, if the player itself made the goal or the player's team scored the goal, the positive reward is provided to promote the same active behavior for maximizing the total reward intake.

2.4 Vision Grid

The input the vision grid receives is derived from an area of constant size around the player, which makes the input size independent of the number of players and the size of the field.

The soccer pitch considered in this research is built up using a grid. This means that all possible locations are discrete points on the pitch. These discrete points will be referred to as cells. The cells in the field can be occupied by four different types of units: agents, goals, walls, and the ball. For the MLP reinforcement learning player, the data about all the cells of the field are given to the player. This allows the agents to be able to learn for even complicated situations of the world and leads to finding the optimal action.

However, the size of the input to the player is dependent on the number of cells in the field. Therefore, we have to find a different approach if we want to design a player that performs well independent of the size of the field. One of the approaches we considered was the use of a vision grid. The vision grid is based on the idea that a player rarely needs to know the exact state of all possible locations in the field and that the region directly surrounding the agent includes most of the data that is important in making good decisions. Therefore, a vision grid only uses the data of a fixed number of cells surrounding the agent. This decreases the number of input neurons needed so the learning process will be faster. Importantly, the input size is also independent of both the size of and the number of agents on the field.

In this research, the vision grid used in Knegt's research (Knegt et al., 2018) was extended. Usually, each cell in the vision grid corresponds to one cell of the environment. This leads to the trade-off

Table 2.1: Reward function

Event	Reward
Goal from player itself	+1
Goal from same team player	+1
Own goal	-1
Goal from opposite team	-1

between having a small amount of input if you use a small vision grid, and being able to 'see' more of the environment if you use a large vision grid.

This approach was altered to make a system that can have a small input size, and see more of the environment. To do this, it was decided that the player does not have to know data for each cell that is further away, and that an aggregated summary of a group of cells would suffice. For example, it suffices to know that the ball is around 10 cells to left of the agent. Knowing whether it is actually 10 or maybe 11 or 12 cells is not expected to influence the behavior of the player in a significant way. Therefore, it is chosen to use a vision grid system that groups and aggregates data for the cells around the agent.

This was implemented by having a variable number of layers with variable sizes. The center cell of the vision grid always has a size of 1 by 1 and contains the current agent. The layers of the vision grid are present around this center cell.

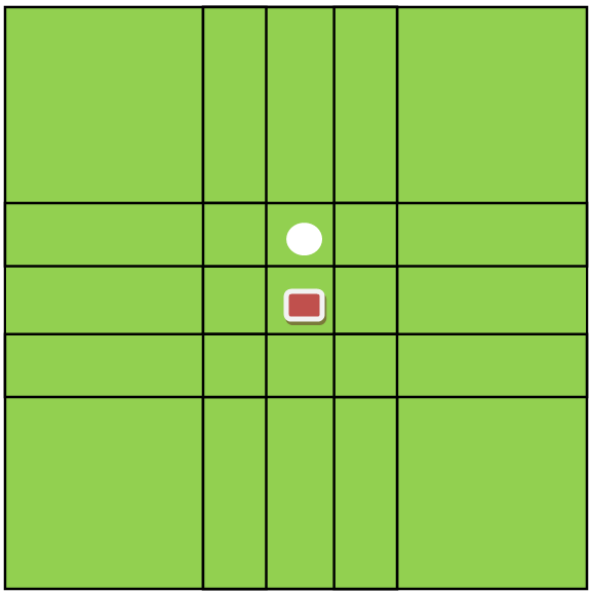

Figure 2.1: An example of a vision grid system with layers of different size. The inner has a size of 1, the outer a size of 3.

The layer sizes are illustrated in figure 2.1. In this figure, you can see the center cell containing the agent. The shown vision grid has two layers with two different layers: the inner layer has a size of 1 and the outer a size of 3.

The vision grids are always created with this kind of shape. To make sure that each cell has one cell adjacent to it on each side, the vision grid cell do not have to be square. For example, in the top row of the vision grid in the figure, the cells are of size 3 by 3, 1 by 3, 1 by 3, 1 by 3, and 3 by 3 respectively. This approach also makes changing the layer sizes easier, as it guarantees that the cells are properly aligned.

2.5 Exploration Methods

If the agent always chooses the action with the maximum Q-value from the state, it cannot explore the results of other possible actions in the state. This will cause the agent not to learn the optimal Q-function. Therefore, exploration actions are necessary to be able to learn how to choose the optimal Q-values and policy. However, it is also important to exploit its current knowledge stored in its learned Q-values to receive the maximized rewards. This results in the exploration-exploitation dilemma (Thrun, 1992).

Although there are many different exploration methods, two undirected strategies were considered for this research: ϵ-greedy and softmax.

2.5.1 ϵ-greedy

ϵ-greedy is one of the most used exploration methods. The parameter ϵ stands for the percentage of actions that are randomly chosen. It exists in the range between 0 and 1, where 0 means that only the exploitation method is used and 1 is that only the exploration method is used. The agent selects the action which has the highest Q-value with probability $1 - \epsilon$, and a random action is chosen otherwise.

2.5.2 Softmax

One limitation of ϵ-greedy is that it is equally possible to choose the worst action as the action for exploration, as it is to choose the second-best action. It is because all exploration actions are randomly chosen in the ϵ-greedy method. The softmax method can solve this problem by using the Boltzmann distribution to assign a probability to all actions based on their relative Q-values. Through assigning the probability to the actions, the softmax

exploration chooses actions with higher Q-values with higher probability.

3 Experiments

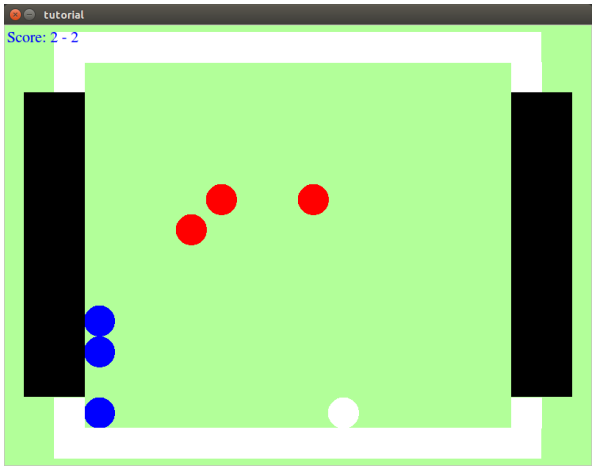

Figure 3.1: A visualized example of the simulation environment. The white circle indicates the ball, the blue circles indicate team 1 and the red circles indicate team 2. The white outline is the border of the field and the black bars are the goals

3.1 Simulation Environment

The model used for this paper was a two-dimensional representation of the game of soccer. This model has a grid structure. The use of this model was chosen to reduce the complexity of the game.

Each cell in this grid structure can generally contain one of the following elements: an agent, a ball, a part of a wall, or a part of a goal. Only at the locations of the goals, these elements can overlap. At the goals, the wall is still present as that made the creation of the program easier. It should not have a detrimental influence on the course of the game though as the program will first test whether the new location of the ball is a goal before it checks whether it is a wall. It does, however, mean that agents can not move into the goals.

Within this model, each player controls one agent. A player can choose from 18 actions each time it is prompted to take action. It can choose to move to any of the eight cells that surround it, or to kick the ball or do nothing at all. There are nine actions for kicking the ball. The player can choose to have its agent kick the ball at one of three different speeds, and in one of three different directions. To kick the ball, the agent has to be in a cell surrounding the ball. If the player chooses to kick the ball when the ball is not close enough, the ball will not move and the effect of the kicking action is the same as doing nothing at all.

To move the ball, the agents can push the ball, by moving into the cell of the ball. The ball will then be displaced by one cell in the same direction as the agent moved. The program will check whether the ball is allowed to move into the new cell. If the location is already occupied by a player or a wall and not by a goal, the ball will stay in its original cell, and the pushing agent will return to its original cell.

Another way to get the ball to move is to kick it. The player can choose three different directions for the kick and these depend on the location of the ball relative to the agent. The available directions are always the direction to the ball from the point of view of the agent and the directions to the 'left' and 'right' of it.

If we assume a grid that has compass directions, an agent can, for example, kick a ball that is to the northwest of it in three directions: the same direction as the ball is from the point of view of the ball, northwest; the direction to the left of that, west; and the direction to the right of that, north.

For each of these directions, the player has the choice between three speeds: 1, 2, and 3. This speed indicates the number of cells the ball is kicked. The ball will then try to move this distance before the next player is prompted to choose an action.

The movement will be fulfilled in steps of one cell. After each step, the program checks whether the ball has collided with another object similar to what happens after pushing the ball. The difference between this and pushing the ball is that here the ball can keep moving after hitting an object. When the ball hits a wall, this results in a bounce. The bounce is achieved by reversing the direction along the x or y-axis depending on the wall that it hit. If the ball hits a corner, both direction components are reversed.

When the ball enters one of the goals, the team

that has to score in that goal receives a point and the field is reset.

The simulation has two possible options for generating an initial state: one with the agents of each team lined up along the wall with their goal, and one where the agents of both teams are placed randomly in free cells in the field. In both cases, the ball is placed in the center of the field. The chosen method is used at the start of the match and when resetting the field.

3.2 State Representation

Based on the units present on the field and the fact that both the agents and goals can be either of the own team, or the opponent team, each player has six different relevant types of units. These are the agents of its own team, the agents of its opponent team, the ball, the walls, its own goal, and the goal of the opponent. Therefore, each player will generally be provided with information on each of these units.

In this paper, two types of reinforcement learning players are considered. They both use an MLP as their Q-function, but differ in the input they provide to the MLP as the state representation. The first gives information on the state of all cells in the field to its MLP. It provides binary data for the unit types to indicate whether a unit is present at that cell. Therefore, the agent would receive several binary values per cell in the field. As may be clear, this means that the size of the input depends on the number of cells in the field. Therefore, this approach is fundamentally unable to work on different field sizes as we are trying to research in this paper. It is believed that a player that has all cell data about the world available to it, should be able to perform well on the field size that it is trained on. Because of this, it was decided that this approach is a good reference to which we can compare our methods that work independent of the field and team sizes.

The second type is based on a vision grid. It recognizes that giving information about all the individual cells limits the generalizability of the player. It is based on the vision grid used in Shantia's research (Shantia, Begue, and Wiering, 2011) discussed earlier in the paper. It centers the perception of the player around its own agent on the field and only looks at a set amount of cells around it. This means that the number of cells it looks at is independent of the number of players and the size of the field. Therefore, a single one of these players can play on any size of the field.

The number of input values for the vision grid player will be lower than the number of cells in the field it works on with high probability. This means that, with a standard vision grid, in which the cells correspond one to one to the cells of the field, a player can only receive information about a very limited number of cells around it. This would limit the effectiveness of the system severely. When the ball is on the opposite side of the field, for example, the player has no clue where the ball might be. It is hypothesized that this is detrimental to the performance of the player. To counter this, we adapted the vision grid system such that the layers are of variable size. This allows us to make the final layer big enough to include the rest of the field. Currently, the size of the final layer is static and set to be big enough for the vision grid to cover the biggest field we use, from any location on that field. Therefore the player will always have an idea in which direction the ball, other players, walls and goals are, even though they do not exactly know in what location they are.

Assuming that the field size is constant, the number of possible states can be approximated. An example can be seen in table 3.1. Similar tables for the field sizes of 15 by 15 and 25 by 25 are shown in appendix A.

To limit the amount of data in the grid, it was assumed that the information about the walls was irrelevant, since they will always be in the same location. To reduce the number of inputs further, we made the size of the goals static, through which the goal cells also remain in the same location. This means that three of the six values per cell are irrelevant and can be removed. This leads to table 3.2.

Table 3.1: An example of the amount of available states for a field of 9 by 9 cells

Width	Height	No. players	No. states
9	9	1	479,808
9	9	2	996,664,704
9	9	5	5.1291835e+18
9	9	7	9.1832587e+24

The vision grid system can be set up to run with various layers. The number of input values depends on the number of layers the vision grid is set up with, and the number of values per cell in this grid. To find the number of input neurons based on these values, the following formula 3.1 can be used:

$$i = (2 * l + 1)^2 * v \tag{3.1}$$

where i is the number of input values to the neural network, l is the number of layers in the vision grid, and v is the number of values per cell in the vision grid.

Each player needs six values per cell to get a complete picture of the world around it. This leads to table 3.3.

The base player is helped by the fact that most of its input is zero at any moment. The number of ones in the input is always equal to the total number of agents on the field plus the number of balls on the field. In our trials, we found that this allows the input size to be bigger without compromising the performance of the MLP. The vision grid, on the other hand, represents the walls and goals as well, which can be spread over multiple different cells and therefore this increases the number of non-zero input values significantly.

As an extension to this research, we propose that the final layer of the vision grid can be made variable and be made to scale with the size of the field.

Table 3.2: The input size for the MLP players based on the size of the field

Width	Height	Values per cell	Input size
9	9	3	243
15	15	3	675
25	25	3	1575

Table 3.3: The relation between the number of layers and the input size for the vision grid players

Number of layers	input size
1	54
2	150
3	296
4	486

3.3 Experimental Setup

Each player will be trained from the ground up to play against random agents in ten separate runs. In each run, the player will play 15,000 training matches.

At the start of each training match, the size of the field and the teams are randomly generated. The field size is generated from an equal distribution over the inclusive range of 9 to 25. The team size is generated from an equal distribution over the inclusive range of 1 to 7. The team size is varied for all players during training, while the field size is only varied for the vision grid players.

The matches consist of 10,000 steps each. At each step, the program cycles through the players to request them to choose an action. The players are prompted in the same order at each step. The actions are executed directly after the player chooses it, which leads to an asynchronous update step. In the training matches, the agents are placed in random locations separate from each other and the ball, when the field is initialized or reset.

After the training matches, the player is tested through a number of test matches. The field sizes used in testing are 9 by 9, 15 by 15, and 25 by 25. The base players are tested on only one field size, while the vision grid players are tested on each of the available test sizes. This is because the base players are trained on the three field sizes separately, while the vision grid players learn to play with a varying field size.

For each of the tested field sizes, the player will be tested with four team sizes: 1, 2, 5, and 7. For each combination of a field and a team size, the player will be tested using 100 test matches per run. In the test matches, the agents are placed along the wall that has the goal they have to defend, when the field is initialized or reset.

3.3.1 Players

In this research, six types of players are tested. These are split into three groups each of which is tested using the ReLU activation function and the sigmoid activation function. The first of these is the base player that receives data from all cells in the field. The other two are vision grid players. These two differ in the layer sizes they use. One has the layer sizes (1, 1). The other has the layer sizes (1,

30). This means that the latter player is always able to receive information about all the cells in the field albeit in an aggregated way.

Because the base player has to be trained separately on each field size, the total number of different training runs is ten.

As a summary, the following five settings are trained and tested using the ReLU activation function and the sigmoid activation function.

1. Base player trained and tested on a field of 9 by 9 cells.
2. Base player trained and tested on a field of 15 by 15 cells.
3. Base player trained and tested on a field of 25 by 25 cells.
4. Vision grid trained on varying field sizes between 6 by 6 and 25 by 25 with layer sizes 1, 1.
5. Vision grid trained on varying field sizes between 6 by 6 and 25 by 25 with layer sizes 1, 30.

Since the vision grid players run more test matches after training than the base players, the total amount of test matches per combination of field size, team size, and type of player is always 1,000.

Apart from this input representation, the players are set up in the same way. They use a fully connected MLP as their Q-function. This MLP is set up with a learning rate of 0.01 and its weights are initiated randomly in the range of -0.01 and 0.01. It has one hidden layer consisting of 50 hidden neurons. The number of input neurons depends on the used input representation, while the number of output neurons is 18, the number of available actions. The hidden layer uses the activation function corresponding to the type being tested, ReLU or sigmoid, and the output layer always uses the linear unit function as its activation function.

Besides the MLP, the players use the same discount factor of 0.99. They also use the same exploration function. In this research, the ϵ-greedy exploration method is used. Softmax was also considered in our trials, but was not found to perform at the same level. During the training phase, the value of ϵ, was linearly decreased. The initial value of ϵ is 0.5. It will then decrease to a value of 0, which occurs first after 13 million training steps, which is two million training steps before the end. This means that ϵ is linearly decreased during the first 13,000 matches and remains at 0 for the last 2,000 matches. In these last matches, the player will not explore anymore and will only use the best actions it finds.

There is one reward and one punishment. If a player makes a goal, then each player in its team will receive a reward of +1, while each player in the opponent team receives a punishment of -1. There are no rewards or punishments attached to any other action, such as pushing or kicking the ball.

The events that happened in the simulation are stored in a queue. There are only events related to goals in the queue for this simulation. The events can be elaborated by adding options such as pushing or kicking the ball, but that is not explored in this research.

The players learn in episodes to ground their Q-values, such that the learned Q-values should not exceed the reward obtained from scoring. An episode ends either when the match ends, or when a goal is scored. After an episode, the locations of the ball and players are reset, and any events, such as goals that occurred are removed from their queue. To make sure that each player has updated their Q-function using all events in the episode, the program will cycle through the players and update their Q-functions when the episode ends. If this is not done, all players that did not score the goal themselves would not receive a reward or punishment after an episode ended because a goal had been scored. At the last update step of the episode, the next state is not considered. Normally, the estimated value of the next state is multiplied with the discount factor and added to the reward, to get a target Q-value estimate to train the MLP with. However, for this last update step, only the direct reward is used as the target, with which the MLP is trained. This direct reward is simply the reward the player received between the last action it chose and the end of the episode.

Each player in the team of reinforcement learning players uses the same Q-function and the same reward function. This means that the players learn quicker, as knowledge obtained by one player is directly shared with all other agents.

4 Results

The results of this research are split into two sections: the goal difference during training and the goal difference during the final test sets.

The performance improvement during this phase is shown in figures 4.1 and 4.2. To improve the legibility of the graphs, the data were grouped into blocks of 100 matches for each of the ten different training sessions per setting. Therefore, each data point shown in the graphs is the mean of 1,000 training matches: 100 from each of the ten training sessions.

To improve readability, these variations are condensed into two tables: tables 4.1 and 4.2.

Table 4.1 shows the summarized performance of players using the ReLU and sigmoid activation function. For this table, the data split on the number of players are grouped.

Table 4.2 shows the summarized performance of players in configurations with different team sizes. For this table, the data split on the type activation function are grouped.

The remarkable points of table 4.1 are the following. Firstly, the size of the field corresponds to the performance of the player. When the field size is smaller, the players can make more goals. Moreover, the vision grid (1, 1) demonstrates the best performance on each of the sizes of the fields.

Table 4.2 also shows some remarkable points. Firstly, the number of players per team influences the performance of the player. The reference players perform worse with bigger teams, while both types of vision grid player perform better with bigger team sizes. For these types, 7 players per team illustrates the best performance in most cases.

5 Conclusions

In this paper, two elements are researched with regards to the used soccer simulation. One is whether it is possible to make an RL agent that can learn to play with different field and player sizes and performs on an equal level or better than an MLP agent that receives input from all the cells of a field. The other is whether the use of the sigmoid function as the activation function for the hidden neurons in the MLP instead of ReLU, influences the performance of the player. The results illustrate that the vision grid approach with the sigmoid activation function shows the best performance. There are two benefits of using this approach. First of all, the use of the vision grid can overcome the problems which often occur by applying reinforcement learning in large state spaces. For example, it decreases memory usage and the time complexity so that it learns faster. Moreover, it not only increases the training speed, but also shows better performance. Since it reduces the number of inputs compared to using a full grid, the speed of learning becomes faster.

5.1 Discussion

As mentioned in the conclusion, after 150,000 games of training across all methods, the vision grid (1, 1) using the sigmoid activation function has been found to perform the best.

Several influential factors lead to a higher amount of scoring: the number of players, field size, and activation function. A smaller field size with more players which use the sigmoid activation function has shown better results in this experiment. However, there are several interesting possible future research topics.

The most surprising element of the results is that the vision grid with two layers of a size of 1 performed better than the vision grid player with an inner layer with size 1, and an outer layer with size 30. It was expected that the latter would perform better than the former, because it allowed the player to 'see' the entire field. Therefore, this player would be able to know it had to move in a certain direction to find the ball, whereas the former player would have no clue, if the ball is further than two cells away.

It might be that the input for the latter player is more complex, as the values are more often non-zero compared to the former player. A more complex input can mean that it takes longer to learn the desired behavior and that it might be harder even to get to a satisfactory result. Even though the current results show barely any improvements in the last stages, it is interesting to see whether training the players on more matches might provide an improvement that sees the latter vision grid perform better than the former vision grid.

Furthermore, the performance of the vision grid might be improved by choosing a different number of layers, with other layer sizes. Besides this, the

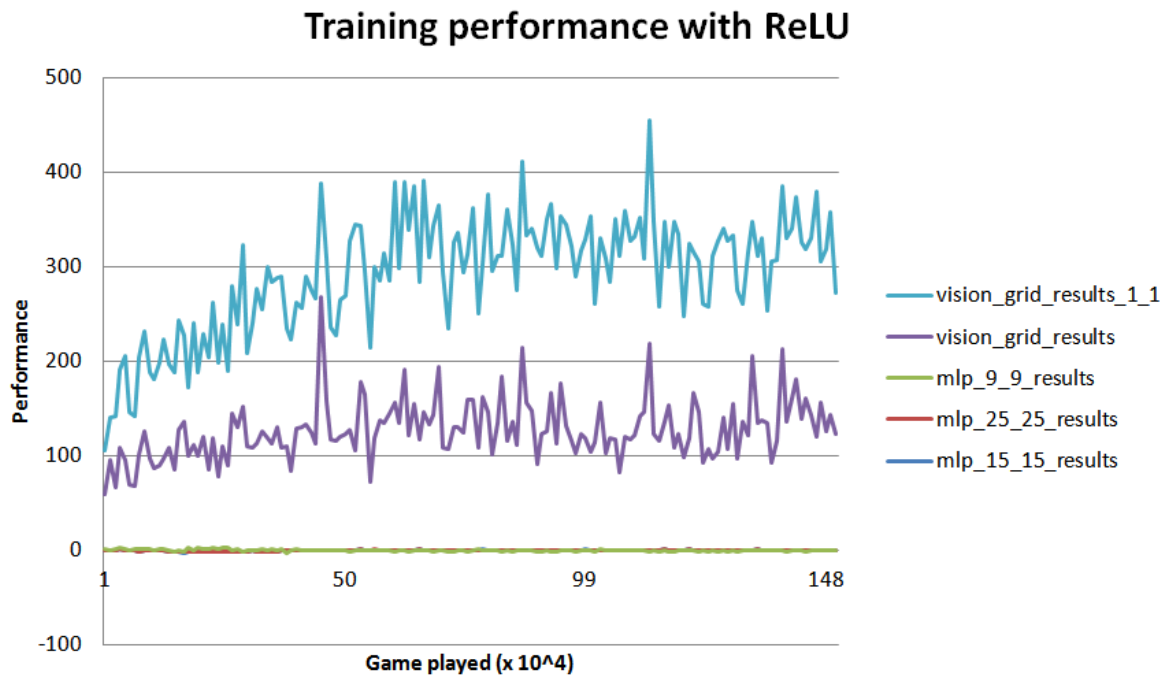

Figure 4.1: The performance score for training with ReLU over 1.5 million training games

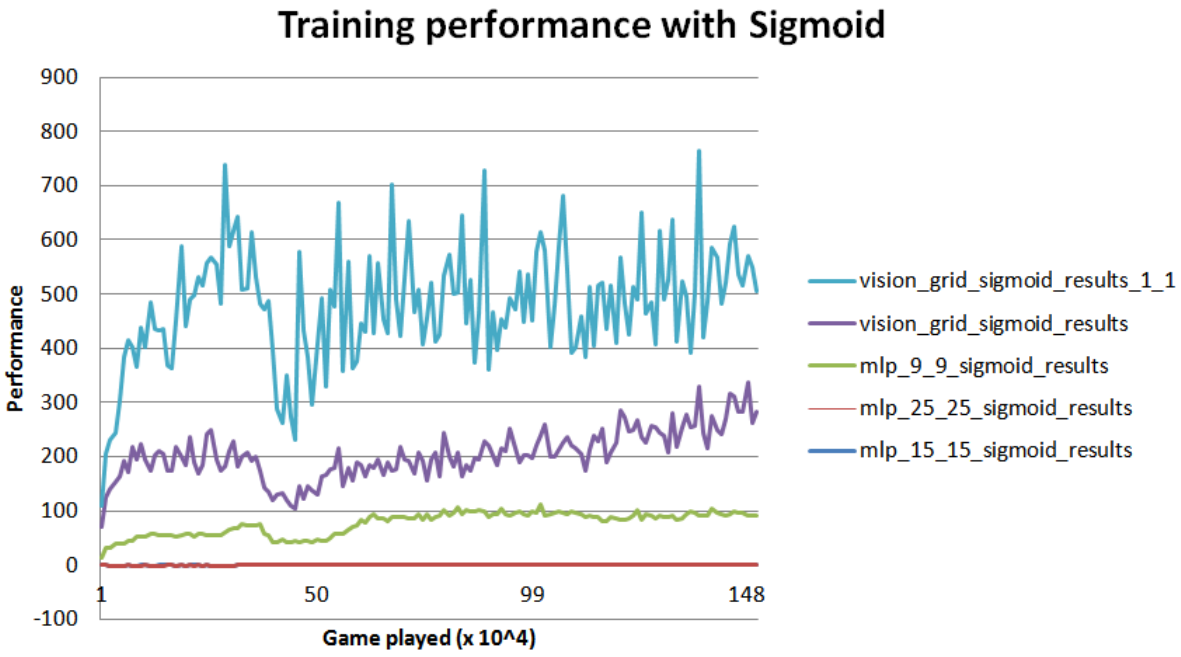

Figure 4.2: The performance score for training with Sigmoid over 1.5 million training games

Table 4.1: The table for the test result with standard error

		Activation function					
Type	Field	ReLU	(SE)	Sigmoid		Total	
MLP	9 x 9	0.24	0.24	151.5	6.8	75.9	3.5
VG 1-1	9 x 9	103.1	2.6	331.0	6.5	217.0	3.7
VG 1-30	9 x 9	53.1	2.0	128.8	2.4	90.9	1.6
MLP	15 x 15	-1.0	0.07	-0.9	0.07	-0.9	0.05
VG 1-1	15 x 15	4.2	0.2	60.6	2.04	32.4	1.1
VG 1-30	15 x 15	7.0	0.6	3.1	0.13	5.0	0.3
MLP	25 x 25	-0.07	0.02	0.12	0.02	0.03	0.01
VG 1-1	25 x 25	0.01	0.02	9.8	0.8	4.9	0.4
VG 1-30	25 x 25	-0.05	0.02	0.10	0.02	0.03	0.013
Total		18.50	0.41	76.01	1.24	47.26	1.48

Table 4.2: The table for the test result based on the team size with standard error

		Number of players per team									
Type	Field	1	(SE)	2		5		7		Total	
MLP	9 x 9	199.3	13.3	33.1	2.4	26.7	0.8	44.4	1.4	75.9	3.5
VG 1-1	9 x 9	76.6	6.9	159.3	8.1	297.1	7.6	335.2	5.4	217.0	3.7
VG 1-30	9 x 9	-0.6	0.2	36.3	1.5	101.1	2.6	226.8	4.2	90.9	1.6
MLP	15 x 15	-0.07	0.05	-0.5	0.07	-1.3	0.1	-1.9	0.14	-0.9	0.05
VG 1-1	15 x 15	3.4	0.2	13.3	0.9	53.8	3.0	59.3	2.7	32.4	1.1
VG 1-30	15 x 15	-0.1	0.05	-0.04	0.07	4.5	0.4	15.8	1.0	5.0	0.3
MLP	25 x 25	0.01	0.01	-0.01	0.02	0.08	0.03	0.02	0.03	0.03	0.01
VG 1-1	25 x 25	0.04	0.01	0.14	0.02	10.5	1.2	9.1	1.0	4.9	0.4
VG 1-30	25 x 25	-0.01	0.01	0.02	0.02	0.05	0.03	0.04	0.04	0.03	0.01
Total		30.95	1.73	26.84	1.03	54.71	1.18	76.52	1.20	47.26	1.48

outside layer can be made to be variable in size, such that the vision grid always includes all cells of the field. This is opposed to the hard limit of 30, which limited the field sizes that could be tried out.

Another interesting topic to consider is the use of different roles for the players in a team. The performance could be better, if certain players are tasked with defending and others with attacking, instead of all being trained to perform similar behavior in similar environments. We could see attacking players being more tended to move towards the opponent's goal, while the defending players tend to track back towards their own goal more. Real-life soccer players also get instructions on their role to get them to perform better as a team, which might also hold true in this case. This leads to the next point of interest: is it possible to add a coaching figure that allows instructions to be given to players? Would it be able to select approaches based on the beliefs it holds about the way the opponent will play? Is it possible to have certain players that can perform better in certain areas than others, like dedicated strikers are often better at shooting than their defending counterparts?

These elements of realism could be hard to realize. However, some aspects can be altered to make it closer to real life and allow for interesting results. The players can have the option, for example, to run at varying speeds, and there might be more randomness to kicking the ball, which simulates the inaccuracies that a normal soccer player faces. The ball can move for multiple time steps. If the information about the current speed is not included in the input representation of the player, it is no longer a Markovian process. This might be solved by using a Partially Observable Markov Decision Process(POMDP), or providing an input for the speed of the ball.

Another approach to make the simulation more realistic, is moving away from the current two-dimensional representation in favor of a three-dimensional one. The players, in this case, could choose a direction and speed of the ball by specifying the values of a three-dimensional movement vector, or two angles and speed. An example of a 3D soccer simulation can be found in (Macalpine et al., 2019). The simulation itself will already be a bit harder to make than the current two-dimensional simulation, and it will be harder for an RL player to learn to play it, but it would certainly lead to some interesting examples.

A further interesting research topic is to apply a convolutional neural network (CNN). Convolutional neural networks are known to save memory and decrease the complexity. Moreover, they are suited for the cases in which it is needed to extract the relevant information at a low computational cost, compared to a fully connected MLP. Although we apply the vision grid approach in our case, it would be great to compare this approach with vision grids.

The next possible future research is performing this research with more activation functions, and with various exploration approaches. According to the research (Knegt et al., 2018), there are some benefits to using the ELU activation function over the sigmoid function. Especially, the ELU function performs much better than the sigmoid function, when receiving less noisy updates due to having more deterministic opponents. Furthermore, another research argues that the Max-Boltzmann exploration approach performs better than the ϵ-greedy approach (Kormelink, Drugan, and Wiering, 2018).

The last interesting future research is to compare the MLP players which learn a generalization to a third class of players: the MLP players that are trained specifically on one field size and one team size. It is expected that this improves the performance in the world that the player is designed for. However, this is not included in this research. Therefore, it might be interesting to see if the MLP player, that performed rather poorly with a bigger field and team sizes, could perform better if it does not have to generalize for situations with a different number of players.

References

I. Hidenori and K Takio. Improvement of learning for CNN with ReLU activation by sparse regularization. *International Joint Conference on Neural Networks IEEE Anchorage*, pages 2684–2691, 2017.

S. Knegt, M. Drugan, and M. Wiering. Opponent modelling in the game of Tron using reinforcement learning. *In International Conference on Agents and Articial Intelligence (ICAART)*, pages 29–40, 2018.

J. Kormelink, M. Drugan, and M. Wiering. Exploration methods for connectionist Q-Learning in Bomberman. *In International Conference on Agents and Articial Intelligence (ICAART)*, 2018.

J. E. Laird. Using a computer game to develop advanced AI. *Computer*, 34 (7):70–75, 2001.

Michael L. Littman. Markov games as a framework for multi-agent reinforcement learning. *The Eleventh International Conference on International Conference on Machine Learning (ICML)*, 11:157–163, 1994.

Patrick Macalpine, Faraz Torabi, Brahma Pavse, John Sigmon, and Peter Stone. *UT Austin Villa: RoboCup 2018 3D Simulation League Champions*, pages 462–475. 08 2019. ISBN 978-3-030-27543-3. doi: 10.1007/978-3-030-27544-0$_3$8.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540): 529–533, February 2015. ISSN 00280836. URL `http://dx.doi.org/10.1038/nature14236`.

R. Prajit, Z. Barret, and V. L. Quoc. Searching for activation functions. *arXiv preprint*, 2017.

A. Shantia, E. Begue, and M. Wiering. Connectionist reinforcement learning for intelligent unit micromanagement in Starcraft. *The 2011 International*

Joint Conference on Neural Networks (IJCNN), IEEE, pages 1794–1801, 2011.

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche, Thore Graepel, and Demis Hassabis. Mastering the game of go without human knowledge. *Nature*, 550:354–, October 2017. URL `http://dx.doi.org/10.1038/nature24270`.

S. Singh, T. Jaakkola, M. Littman, and C. Szepesvari. Convergence results for single-step on-policy reinforcement-learning algorithms. *Machine Learning 38(3)*, pages 287–308, 2000.

R. S. Sutton and A. G. Barto. *Reinforcement Learning : An Introduction.* Bradford Books, 2015.

S. Thrun. Efficient exploration in reinforcement learning. *Technical Report*, 1992.

Christopher Watkins. Learning from delayed rewards. 01 1989.

A Appendix

Table A.1: The example of available states for a field of 15 by 15 pixels

Width	Height	No. players	No. states
15	15	2	34,581,456
15	15	4	9.4764796e+11
15	15	10	1.6834605e+25
15	15	14	1.0108538e+34

Table A.2: The example of available states for a field of 25 by 25 pixels

Width	Height	No. players	No. states
25	25	2	1,806,590,016
25	25	4	4.9893215e+14
25	25	10	1.0038425e+31
25	25	14	7.1452763e+41

B Division of work

We started off working together to find out what we should do and to make the simulation environment in which the players should run. Later, we started working on the different agents. We split this work into two parts. Firstly, we researched with vision grid and the base player, to see which performs best. We also researched the difference in performance between using the ReLU activation function and the sigmoid activation function. Yujin did the comparison between the activation functions, while Julian compared the vision grid player with the base player. Since Yujin had the password of the peregrine, she trained and tested the program. For the report, Yujin wrote the first draft of the paper, after that Julian elaborated on it. We went over the paper together afterwards to combine everything properly and iron out any mistakes that were made.