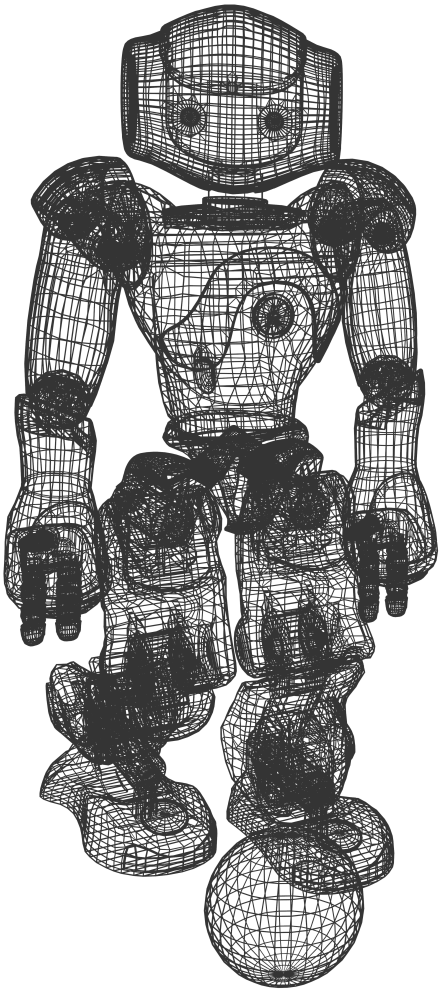
Learning a Robot to Score a Penalty

Minimal Reward Reinforcement Learning



Caitlin G. Lagrand

Supervisor
Dr T.E.J. Mensink

July 2nd, 2017

University of Amsterdam

Learning a Robot to Score a Penalty

Minimal Reward Reinforcement Learning

Caitlin G. Lagrand
10759972

Bachelor thesis
Credits: 18 EC

Bachelor Opleiding Kunstmatige Intelligentie

University of Amsterdam
Faculty of Science
Science Park 904
1098 XH Amsterdam

Supervisor
Dr. T. E. J. Mensink

Informatics Institute
Faculty of Science
University of Amsterdam
Science Park 904
1098 XH Amsterdam

July 2nd, 2017

Abstract

Reinforcement Learning is applied to learn a robot how to score a penalty in simulation. Minimal rewards are given to let the robot learn as much as possible by itself. Intermediate rewards are applied to reduce the amount of trials needed to learn a good action policy up to 75% without effecting the accuracy. Furthermore, different methods of combining object states (multiplying and additive) are examined to determine the method that uses less trials to learn. The additive method results in a decrease of 58% in the amount of trials. A robot is able to learn how to score a penalty against a standing goalkeeper within an average of 170 trials using all intermediate rewards, the additive state representation method and sixteen different states for both the ball and the goalkeeper, with an average accuracy of 69%.

Contents

Figure 1: A Nao robot that is used in the RoboCup Standard Platform League.

1 Introduction

Playing football with robots is more than programming how robots should walk and shoot, the (team)tactics of the robots play a major part during a match, which leads to the question what a robot should do in a specific situation. For example, while standing in front of the ball and the goal, and no keeper in between them, the robot should kick the ball and score. Driven by the popularity of the RoboCup Standard Platform League (SPL) competition[1], where all robots are identical in hardware, we focus on the Nao robot[2] shown in Figure 1.

Currently, most behaviours of the robots are based on rules that the teams invent themselves, such as shooting the ball if the robot is at a certain distance to the ball (Lagrand et al., 2016) (Röfer et al., 2016). However, these man-made rules require a large amount of precision and many small adjustments of parameters. For example, the parameter that defines the distance to the ball before kicking differs for various fields and needs to be manually adjusted until the correct value is obtained. Therefore the main hypothesis of this thesis is that tactic should be learnt from experience and not programmed by engineers.

Reinforcement Learning uses rewards obtained from the environment to learn actions that can be taken, resulting in an action policy that is learnt from own experiences. For example, Reinforcement Learning has been used for learning to play games like Pong (Mnih et al., 2015), where only the pixels of a game state were used as input environment and the obtained score as reward. Depending on the scenario, rewards are only given at the end of the game (e.g. chess or checkers), or already during the game (e.g. Pong, SuperMario). The action policy is learnt from its own experience and used to play these games successfully.

Since Reinforcement Learning learns from experiences, we experiment with it, using minimal reward set-ups to learn a robot to score a penalty. Contrary to preceding research, we focus on learning the actions that lead to a scoring behaviour and not the joint values (Hester et al., 2010) or higher level behaviours (Smart and Kaelbling, 2002). Different scenarios for learning behaviours will be validated to demonstrate that behaviours can be learnt using Reinforcement Learning.

[1] `http://spl.robocup.org/`

[2] `https://www.ald.softbankrobotics.com/en/cool-robots/nao`

The next section discusses preceding research about reinforcement learning. In section 3, Q-learning is explained. Section 4 describes the method used to learn a robot its own behaviour and section 5 presents the experiments that were performed together with their results. Finally, section 6 presents the conclusion and discussion, and future work is proposed.

2 Related Work

Presently, Reinforcement Learning is used for many different purposes. For example, a Reinforcement Learning algorithm to automatically learn a controller for helicopters is applied by Ng et al. (2006). In Mnih et al. (2015), Reinforcement Learning was applied to learn a computer to play classic Atari 2600 games. Using a deep neural network, a winning strategy for different games was learnt based on the most raw data available namely the pixels of the image rendered by the game and the game scores.

In robotics, Reinforcement Learning is used less often, primarily since the considerable amount of trials. By using a simulator for this research, we circumvent this problem and can run a large number of trials. Contrary to the behaviours learnt in this thesis, most Reinforcement Learning that is applied in robotics concerns control policies. For example, in Smart and Kaelbling (2002) Reinforcement Learning was applied to learn control policies for mobile robots. Q-learning (Watkins, 1989) was used in order to learn a robot two tasks, which are Corridor Following and Obstacle Avoidance. During both tasks, the rotation speed of the robot was successfully learnt in approximately two hours and the robot was able to complete the tasks. Q-learning requires discrete states and actions, which is often not the case with mobile robots. To overcome this problem, Q-learning was implemented with HEDGER (Smart and Kaelbling, 2000), an algorithm that safely approximates the value function for continuous state control tasks.

Furthermore, the use of Reinforcement Learning in order that a robot learns to kick a penalty is described in (Hester et al., 2010). A robot learnt how many degrees it had to shift its leg outwards, before kicking the ball. Reinforcement Learning with decision trees (Pyeatt, 2003) was applied in order to limit the number of trials needed for learning. This algorithm uses decision trees instead of a lookup table or a neural network to acquire the best policy for enormous problems. A control policy for shifting the leg outwards was learnt and the robot was able to score on 85% of its attempts. While Hester et al. are learning the joint positions for scoring a penalty, we focus on higher level actions that together form a scoring tactic.

When using robots in simulation, not only control policies, but also behaviour policies are learnt. Stone et al. (2005) addresses a three versus two *keepaway* football problem in simulation. In this situation, a team of three robots, the keepers, should maintain possession of the ball, while two takers try to capture the ball. A semi-Markov decision process (SMDP) (Baykal-Gürsoy and Gürsoy, 2007) is used to let the robots learn their actions, such as Receive, Pass or Hold. Stone et al. show that the policy learnt could hold the ball an average of 9.6 seconds while a hand-coded policy was able to hold the ball for an average of 8.2 seconds. In contrary to the high level behaviours learnt by them, this thesis focusses on learning the actions that together form a tactic, in order that no tactics, such as hold the ball, have to be predefined as they do.

$$\begin{bmatrix} Q(s_1, a_1) & Q(s_1, a_2) & \cdots & Q(s_1, a_{n_{actions}}) \\ Q(s_2, a_1) & Q(s_2, a_2) & \cdots & Q(s_2, a_{n_{actions}}) \\ \vdots & \vdots & \ddots & \vdots \\ Q(s_{n_{states}}, a_1) & Q(s_{n_{states}}, a_2) & \cdots & Q(s_{n_{states}}, a_{n_{actions}}) \end{bmatrix}$$

Figure 2: Q-learning with a lookup table. The Q-values of all state-action pairs are saved in a table that consists of n_{states} rows and $n_{actions}$ columns.

3 Q-learning

Q-learning (Watkins, 1989) is a Reinforcement Learning algorithm that learns an action-value function (Q) which represents the expected utility (Q-value) of taking a given action and state. The optimal policy that follows is constructed by selecting the action with the highest Q-value. An agent starts with randomly initialised Q-values and obtains the state (s) from the environment. Using action selection (section 3.3), an action (a) is selected that will be performed in this state. After performing the action, the reward (r) for taking the action in this state, and the next state (s') are obtained. Next, the Q-value is updated as follows:

$$Q(s, a) \leftarrow Q(s, a) + \alpha[r(s, a) + \gamma max_{a'} Q(s', a') - Q(s, a)] \qquad (1)$$

where α is the learning rate and γ is the discount factor which weights the importance of the Q-value of the next state.

3.1 Q-learning with a Lookup Table

The classic Q-learning algorithm makes use of a lookup table to save the Q-values of all state-action pairs. The table consists of n_{states} rows and $n_{actions}$ columns (Figure 2). Each time a Q-value is updated (1), the Q-values for this state and the next states are obtained from the table and used to update the Q-value. One of the advantages of a lookup table is that the exact Q-values are available for each state-action pair. Therefore, the action policy found will be highly accurate. Saving all Q-values is also a disadvantage, as it needs a large amount of memory, when the amount of states and actions increases.

3.2 Q-learning with a Neural Network

Another method, which is useful for handling a large amount of states and actions, is Q-learning combined with a neural network. Since neural networks can use multiple hidden layers that are smaller than the amount of states and actions, the memory required is lower than that for a lookup table in cases with a large amount of states and actions. However, when using a small amount of states and actions, a lookup table uses less memory. Instead of saving the Q-value of each state-action pair, a neural network, having a vector representing the state as input and the Q-values of each action as output, is used (Figure 3). Each time a Q-value is updated (1), the network is trained having the state vector as input and the updated Q-value as output. The disadvantage of a neural network is that it only estimates the Q-values, which results in noisy updates and thus an even more noisy neural network.

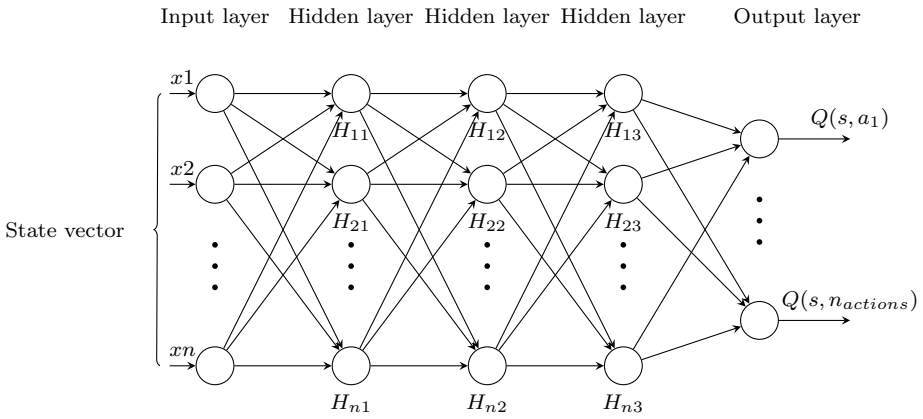


Figure 3: Q-learning using a Neural Network. The state vector is used as input vector and the Q-values for this state are the output.

3.3 Action Selection

Action selecting is used to determine which action to take based on the acquired Q-values. Several methods are introduced (Sutton and Barto, 1998, p. 30-31), such as the greedy method, ε-greedy and softmax action selection. The simplest action selection is to always select the action with the highest Q-value. This greedy method always exploits current knowledge, but never explores other (possibly better) actions.

To be able to explore other actions, ε-greedy is a good alternative. ε-greedy is mostly greedy, but with a probability of ε, an action is randomly selected amongst all actions. An advantage of ε-greedy is the certainty that every action will be sampled when the amount of trials increases. However, the disadvantage is that it selects equally among all actions when exploring, which results in choosing the worse action with the same likelihood as the second best action.

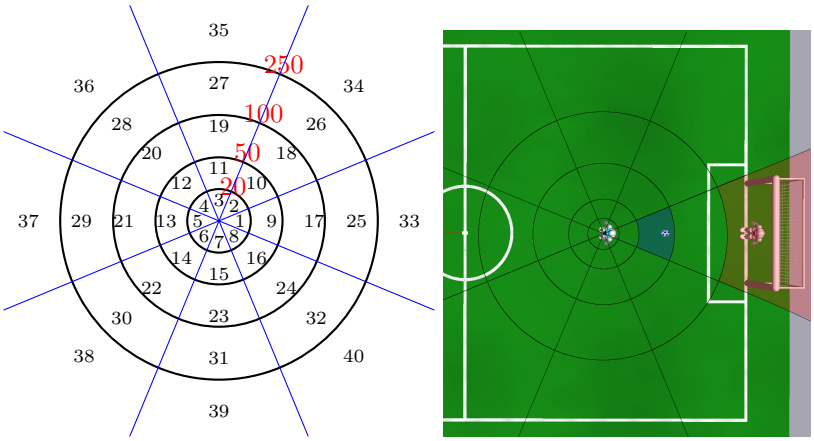
Softmax action selection avoids this equally distributed selection by computing a weighted distribution based on the Q-values. Boltzmann action selection is one of the most common softmax methods. This method uses a Boltzmann distribution and chooses action a with the following probability:

$$\frac{e^{Q(a)/\tau}}{\Sigma_{i=1}^{n} e^{Q(i)/\tau}} \tag{2}$$

where τ is the temperature parameter. Low temperatures cause a greater difference in the probabilities, while high temperatures cause the probabilities of the actions to be all (nearly) the same.

4 Method

This section describes the method used to apply Q-learning to learn a robot to score a penalty. Firstly, the states and their representations that are used are explained in section 4.1, followed by the actions in section 4.2. Section 4.3 describes the different rewards that can be obtained from the environment and in section 4.4, the network which is used to learn



(a) (b)

Figure 4: State bins relative to the robot. Left shows the different bins, with the robot as the centre point. Depending on the distance (red) and angle (blue), a position on the field is placed in a bin. On the right, the state bins are projected on the field. The ball is situated in the blue state and the goalkeeper and goal are both in the red state.

and predict the Q-values is explained. Furthermore, an overview is given in section 4.5 of how these components are combined to create a learning process. Lastly, the evaluation method is described in section 4.6.

4.1 States

The states describe the environment in which the robot is situated. The environment consists of various objects on different positions on the field, such as the ball, the goals, and other robots (both opponents and teammates).

4.1.1 Local versus Global Positions

The states can be based on either local or global positions of objects in the field. We chose to use local positions (robot frame), since the positions of the objects are chiefly based on the observations of the robot made with its cameras and thus already relative to the robot. Moreover, the global positions are computed from the relative positions, so if a small error would occur in the relative position, a bigger error would occur in the global position.

4.1.2 Discretised Positions

The position of the objects is discretised to reduce the state space. Each position is put into a bin based on its angle and its distance to the robot, see Figure 4. Eight bins for angles and five different distances can be used, which results in 40 bins per object. Depending on the experiment, all five distances or less are used. Bins that are closer to the robot are smaller than the bins farther away from the robot, as a higher precision is desired closer to the robot.

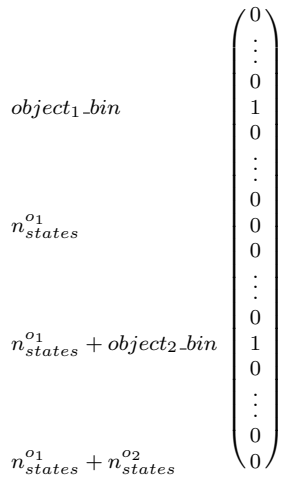


Figure 5: Example of a n-hot-vector with $n = 2$. The one-hot-vectors of both objects are appended to each other, resulting in a two-hot-vector with $n^{o_1}_{states} + n^{o_2}_{states}$ elements.

4.1.3 Combining States from Different Objects

The states of different objects can be combined in several ways. Firstly, all possible permutations can be computed by (3), resulting in $\prod_{o=0}^{n_{objects}} n^{(o)}_{states}$ states. This method can be used when Q-learning with a lookup table is applied, because it contains all possible states separately. However, it results in a large state space, and thus we expect that more trials are needed to explore it.

$$state\_bin = \sum_{o=0}^{n_{objects}} (bin^{(o)} - 1) \prod_{i=1}^{o} (n^i_{bins}) \tag{3}$$

Another method to combine different objects is to create a n-hot-vector, based on the amount of different objects in the field (n). Firstly, a one-hot-vector is created for each object separately, containing a one at the $object\_bin^{th}$ position and zeros elsewhere. These one-hot-vectors are then appended to each other, resulting in a vector of $\sum_{o=0}^{n_{objects}} n^{(o)}_{states}$ elements, see Figure 5. This method uses a smaller state space to represent all possible states and is very useful when using Q-learning with a neural network, since a neural network can handle a n-hot-vector.

4.2 Actions

In contrary to the actions in (Hester et al., 2010), that involve moving one joint, and (Smart and Kaelbling, 2002) which uses higher level behaviours, the actions that the robot can perform in this thesis are all single actions, such as walking in a specific direction or kicking. Figure 6 shows the thirteen actions the robot can perform.

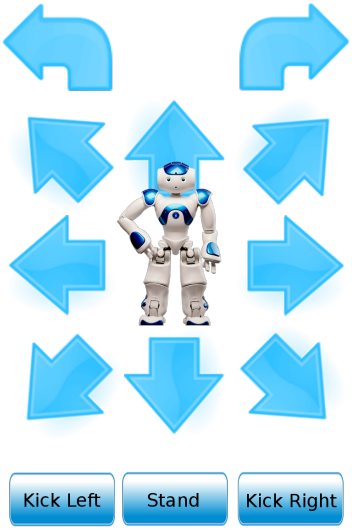


Figure 6: Possible actions that the robot can perform.

4.3 Rewards

The robot learns from the rewards obtained from the environment. To let the robot learn as much as possible by itself, minimal amount of rewards are desirable. However, as the state space increases, many trials are needed to explore the states and reducing this amount of trials would be preferred. Since some actions lead to a higher probability of scoring, minimal guidance that encourages the robot to take these actions, can decrease the search space rapidly. Therefore, we experiment with different rewards.

4.3.1 Minimal Reward

Initially, a reward is only given at the end of a trial. This minimal reward is comparable with a real match, as only the final score is taken into account. Scoring a goal results in a positive reward (2) and, when no goal is scored within a minute, a negative reward (-2) is given. However, a disadvantage of obtaining a reward only at the end of a match is that it results in sparse rewards, which can lead to many trails needed to learn a good policy, since many actions are needed to be taken before obtaining any reward.

4.3.2 Ball Distance Reward

Minimal guidance by giving intermediate rewards is expected to decrease the search space significantly. Since the ball is the most important object during a match, because it needs to end up in the goal of the opponent, minimal guidance uses the distance between the ball and the robot, and between the ball and the goal as intermediate rewards.

4.3.2.1 Distance between the Ball and the Robot

Firstly, the distance between the robot and the ball is used as intermediate reward, as movement to the ball is expected to be a good action, since the robot has to move the ball towards the goal. If an action causes the robot to be closer to the ball compared to the state before, a positive reward (0.5) is given, otherwise a negative reward (-0.5) is given.

4.3.2.2 Distance between the Ball and the Goal

Another intermediate reward is the distance between the ball and the goal, because an action that causes the ball to be closer to the goal is expected to be a good action. If an action causes the ball to be closer to the goal, a positive reward (1) is given, otherwise a negative reward (-1) is given.

To stimulate scoring using a kick, the reward for the distance between the ball and the goal is higher than the reward for the distance between the robot and the ball. If the robot scores by kicking, the distance to the ball increases, which results in a negative reward. Due to the bigger positive reward for the distance between the ball and the goal, the robot still receives a positive reward when scoring using a kick.

4.3.3 Time Rewards

Furthermore, the time left for the trial is given as additional reward when a goal is scored, to encourage the robot to score as fast as possible. The robot is expected to score more by kicking using this reward, since the time to score by kicking is lower than by walking the ball into the goal. The time is scaled to a number between 0 and 1 and added to the reward the robot received for scoring.

4.4 Network

Despite only the small amount of states used in this thesis, we use a neural network to learn and predict the Q-values instead of a lookup table, which would have used less memory as explained in 3.2. We chose to use a neural network to be able to easily expand this thesis to research that uses many states and actions.

A five-layer fully-connected network is used to learn and predict the Q-values (Figure 3). The size of the input layer and the three hidden layers equals the size of the state vector. The amount of nodes in the output layer equals the amount of possible actions, which is thirteen. A rectified linear unit (ReLU) is used as activation function for the input layer and the three hidden layers. The output layer uses the identity function, since Q-values can be both positive and negative.

4.5 Training using a Simulator

Q-learning is implemented within the framework of the Dutch Nao Team 2016 (Lagrand et al., 2016), which is based on B-Human 2015[3]. The only major differences are the ball detector and the behaviour engine[4]. This framework uses SimRobot (Laue and Röfer, 2008) as simulation tool.

[3] `https://github.com/bhuman/BHumanCodeRelease/tree/coderelease2015`
[4] `https://github.com/pkok/behavior-engine`

SimRobot uses external XML files that are loaded at runtime to model the environment. These files can be easily edited to create the desired scenario, such as the one versus one penalty scene shown in Figure 7. The simulation is adjusted in order to have trials of 1 minute and a goal for the opponent team if no goal is scored within this minute. A simulation is used to circumvent the limitation a real robot has with the amount of trials. Moreover, a fully observable field is used in simulation, to ensure that the focus can be on learning behaviours, without having errors in the position of the objects due to poor observations of the robot.

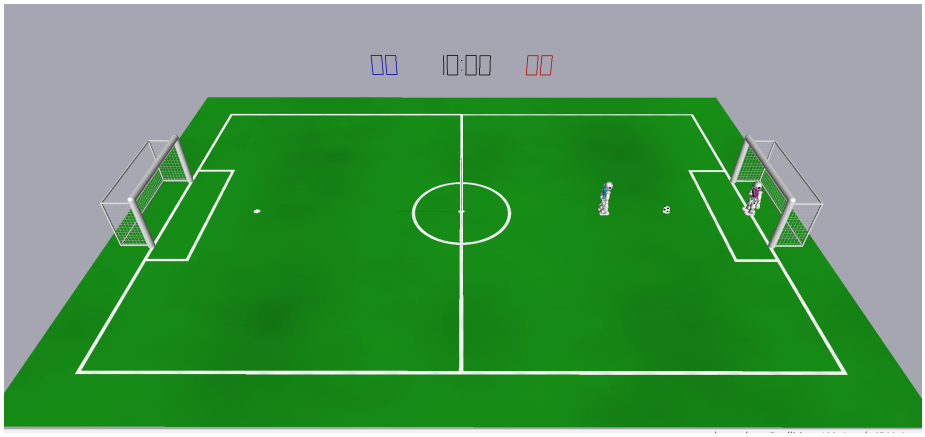


Figure 7: Penalty situation in SimRobot simulation.

A session is one training session in which a robot learns to score by repeatedly starting an attempt to score, a trial. Figure 8 shows an overview of the training process, the outer (red) circle represents a trial while the inner (blue) circle is executed every time the robot enters a different state. When an action would be chosen every simulation cycle, different actions would be executed while the robot is still in the same state, due to exploration. The previously chosen action would only have had one simulation step to execute itself, while they require more than one simulation cycle to have some effect on the environment and to get a relevant reward.

By starting a session the network is initialised (1) with random weights and the first trial is started. At the start of each trial the robots and the ball are placed on their starting positions, the time is set to one minute and a new attempt begins (2). The attacking team now tries to score within a minute by repeatedly obtaining the state it is in (3), selecting and performing an action (4), obtaining the reward (5) and updating the network (6). The state information is obtained from the environment by computing the state bin (section 4.1) of which a state vector is created. Subsequently, the state vector is used as input for the network to predict the Q-values of the actions. Boltzmann action selection (3.3) is now applied on the predicted Q-values to select the next action and by performing this action, the next state and the reward are obtained from the environment. Next, the Q-value is updated and the network is trained with the state vector as input and the updated Q-values as output. These steps are repeated until a trial ends, which is after one minute or after a goal is scored. If the robot has scored x times in a row, the session ends, otherwise, a new trial is started.

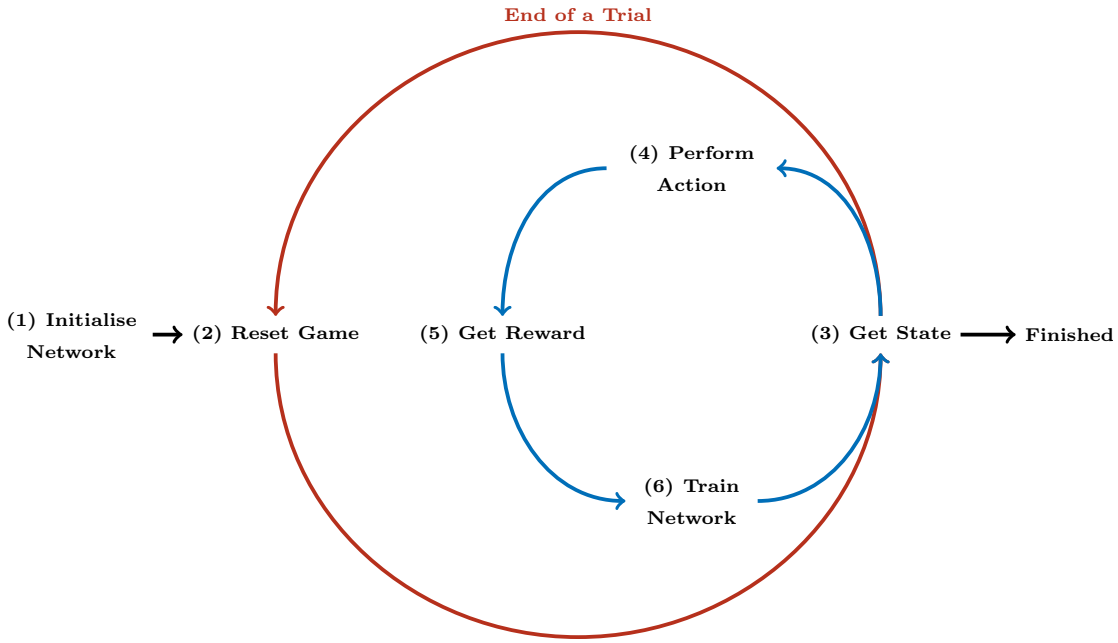


Figure 8: Learning process of the robot (session). The outer (red) circle represents a trial while the inner (blue) circle is executed every time the robot enters a different state. A trials ends after 1 minute or after a goal is scored. The session is finished when the robot has scored x times in a row, with x being different for different experiments.

4.6 Evaluation

The robot has learnt a good action policy and stops training when it has scored x times in a row. x differs for the different experiments, since scoring by kicking is less reliable than walking the ball into the goal. In the experiments in which a goal can be easily scored by walking, x is higher, to ensure the robot has explored enough other actions and states before it stops training. Lower values of x are used in situations where the robot has to score by kicking, because one missed attempt quickly leads to going to explore many other actions before trying the action that previously led to a goal again. Since the network is always randomly initialised and high variation in the amount of trials can occur between different sessions, each experiment is performed multiple times. To evaluate the speed at which a network learns to score, the average amount of trials needed to train the network is computed. The performance of the learnt network is evaluated by computing the accuracy over fifty attempts to score, by averaging over the different sessions, formally defined as follows:

$$\frac{1}{n_{sessions}} \sum_{s=0}^{n_{sessions}} \frac{n_{goals}^{(s)}}{n_{attempts}} \tag{4}$$

5 Experiments

This section will present the experiments that were performed together with their results. Firstly, section 5.1 will explain the experiment to test the effects of intermediate rewards. Secondly, the two methods to combine states are compared in section 5.2. Finally, a one versus one situation is tested in section 5.3. All experiments are performed with $\alpha = 1$, $\gamma = 0.2$ and $\tau = 0.5$ (as defined in Equation 1 and 2).

5.1 Effects of Intermediate Rewards

The first experiment involves only one robot and the ball. The robot has to score in an empty goal. The robot always starts one metre behind the ball and only uses the relative position from the ball to the robot as input from the environment. This experiment is used to test the effects of different intermediate rewards on the amount of trials needed to train and the accuracy of the trained network. The intermediate rewards are expected to decrease the search space of the robot and thus lower the amount of trials. Since the focus of this experiment is to demonstrate the effect of intermediate rewards and each experiment is performed twenty times, only sixteen states are used for the ball to speed up the learning process. After having scored ten times in a row ($x = 10$), the learning process is finished and the network is saved. Firstly, no intermediate rewards are used as baseline. Next, the distance between the robot and the ball or the distance between the ball and the goal are added separately as intermediate reward. Lastly, both intermediate rewards are added concurrently.

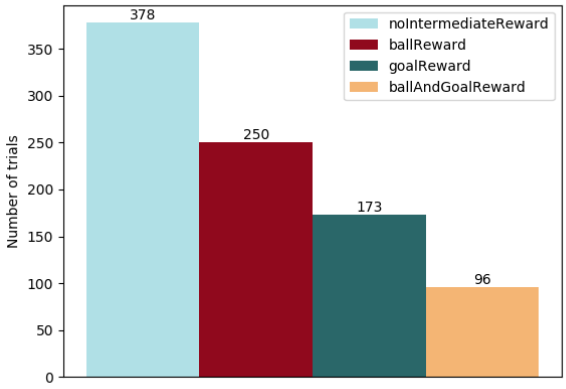
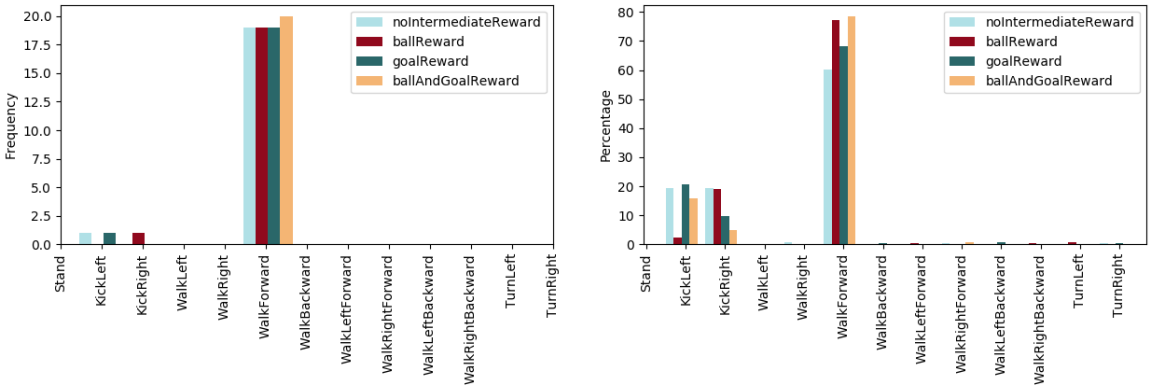


Figure 9: Average amount of trials of the sessions using different intermediate rewards.

5.1.1 Results

Figure 11 shows the positions the robot has visited during the sessions for the different rewards. The plots on the left hand side show the first 70% of the positions of the robot, which are spread out in all directions,



(a) (b)

Figure 10: The actions performed that led to a goal. (a) shows that last action performed in a session to score. (b) presents all actions during a session that led to a goal.

because the robot has to explore first. The plots on the right hand side show the last 30% that are more focused toward the goal (red line). As predicted, the last 30% of the sessions with no intermediate reward is still exploring much in all directions. Contrarily, the sessions that did receive intermediates rewards explore more in the direction of the goal. Using both the distance between the robot and the ball, and the distance between the ball and the goal as reward results in even more exploring in the direction of the goal.

Furthermore, the amount of trials decreases when using additional intermediate rewards as shown in Figure 9. Receiving the distance between the ball and the robot as intermediate reward results in a decrease of 34%, while the intermediate reward given for the distance between the ball and the goal causes a decrease of 54%. Using both intermediate rewards decreases the amount of trials by 75%.

When analysing the actions that led to the last goal in a session (Figure 10a, no significant difference can be found between the different intermediate rewards. Scoring by walking the ball into the goal is definitely preferred to kicking, but when looking at all actions that led to a goal 10b, there are goals scored by kicking. However, when analysing the accuracies shown in Figure 12, the accuracy for scoring by kicking is significantly lower than the accuracy for scoring by walking, which can explain the lower interest in scoring by kicking, since it results in more missed attempts and ten times scoring in a row is thus harder to achieve by kicking. The lower accuracy for scoring by kicking can be explained by some small offset in the positioning of the robot and the ball in the simulation, which leads to different ways of hitting the ball and thus more variation in the result of kicking the ball.

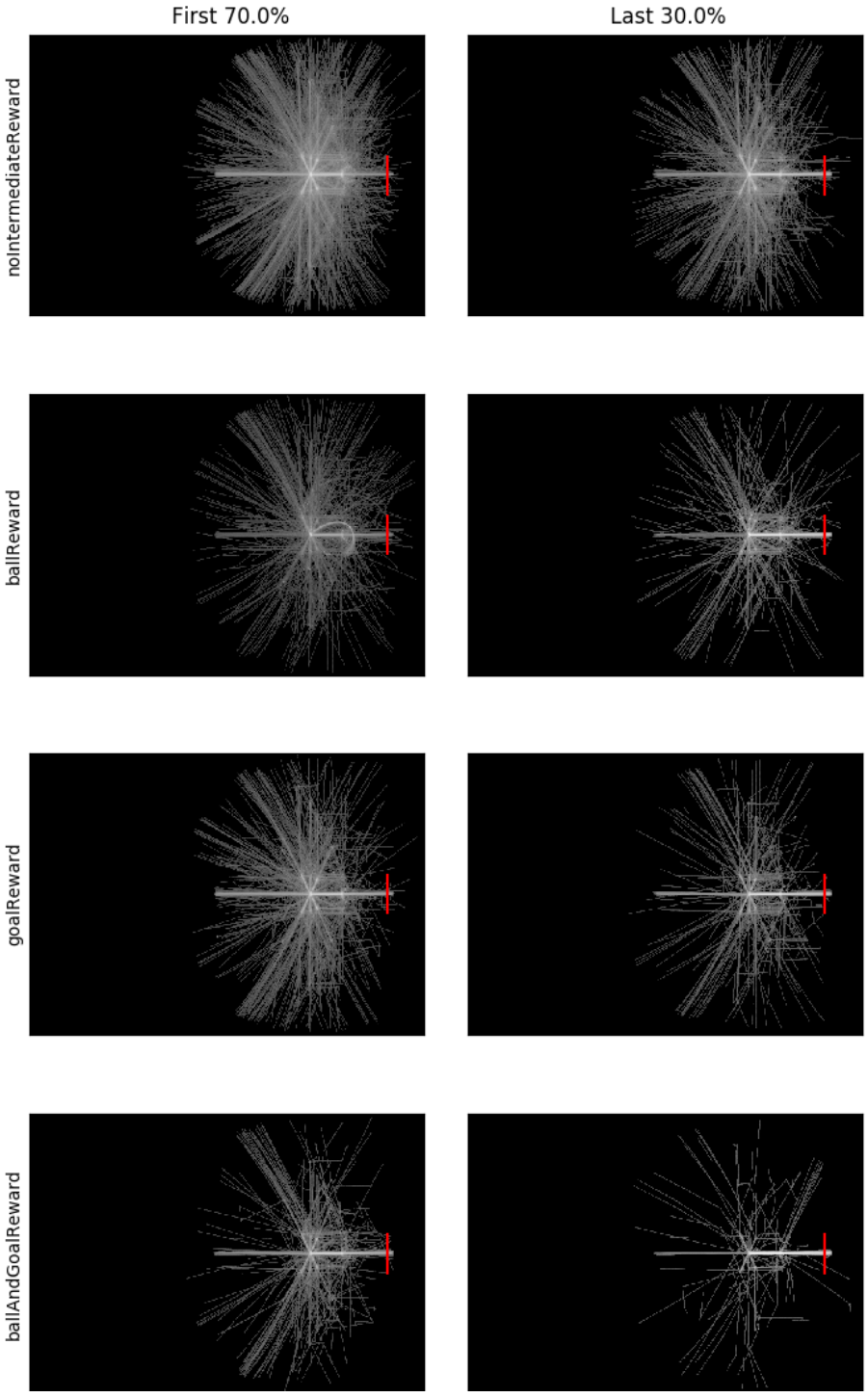


Figure 11: Positions the robot has visited during training. The red line represents the goal.

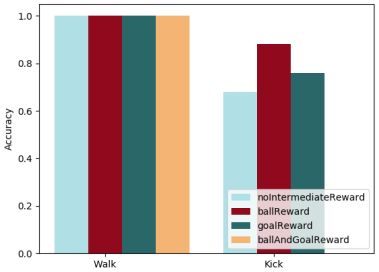


Figure 12: Accuracy of the different actions that led to a goal using different intermediate rewards.

5.2 Combining States by Multiplying or Addition

The next experiment compares the multiplying and additive methods of combining states (section 4.1.3). A one versus one situation, with the ball having sixteen states and eight states for the goalkeeper, is used to compare the different methods. Since the additive method results in a smaller state space, a decrease in the amount of trials is expected using this method. Each experiment is performed five times to average the results and the robot has to score five times in a row ($x = 5$), before the learning process is finished and the network is saved.

5.2.1 Results

Figure 13 shows that the amount of trials needed to train the network is indeed decreased by 58% when using the additive method. This can be explained by the smaller search space used by the additive method. Figure 14 shows that the additive method does not have a negative effect on the accuracy. In contrary, the accuracy of the additive method is higher than the accuracy of the multiplying method. However, this can be the result of the same offset in the simulation as mentioned in section 5.1.1, which leads to more variation in the result of kicking the ball.

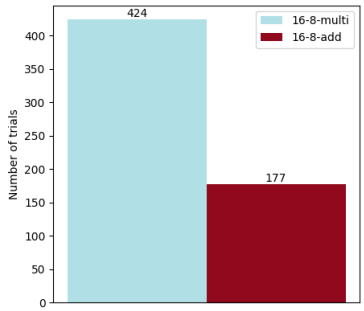


Figure 13: Average amount of trials of the sessions using the additive and multiplying method for combining states.

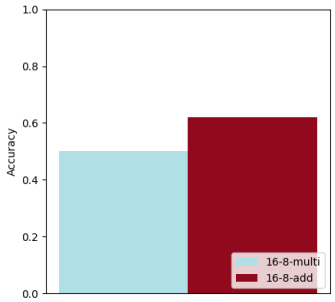


Figure 14: Accuracy of the trained network using the additive and multiplying method for combining states.

5.3 One versus One

In the last experiment, the goal is defended by a goalkeeper that does not move. Simply walking forward and scoring is no longer possible and a different tactic than the one used in 5.1 needs to be learnt. Different amount of states are tested for both the attacker and the goalkeeper, which are combined using the additive method. Furthermore, all intermediate rewards (distance between robot and ball, distance between ball and goal, and time reward) are used, to decrease the amount of trials needed to score. The training process is finished after three goals are scored in a row ($x = 3$), or after 7500 trials, which is about thirty hours on a Intel 6700K processor.

5.3.1 Results

Figure 15 shows the number of goals scored during the training process. Using sixteen states for the ball and either eight or sixteen for the goalkeeper results in a good action policy in less than 300 trials (Figure 15a. However, when using more states, the robot did not satisfy the condition of scoring five times in a row within 7500 trials as shown in Figure 15b. Nevertheless, the figure shows that the amount of trials needed to score increases when using more states, this is due to having more state-action pairs that are explored before scoring.

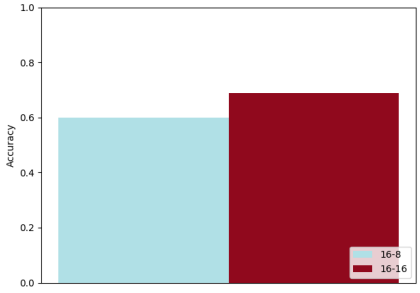
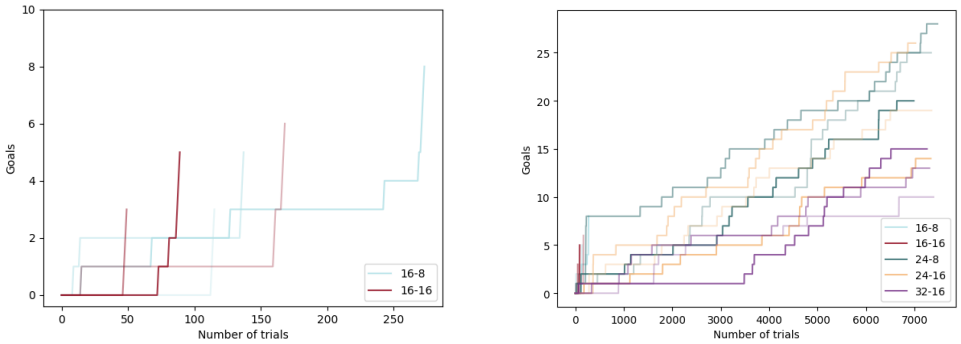


Figure 16: Accuracy of the trained network for a one-versus-one scenario.



(a) (b)

Figure 15: Number of goals scored by the robot using different amount of states for training a one-versus-one scenario. Scoring five times in a row stops the training process.

The accuracy of the trained networks using sixteen states for the ball and either eight or sixteen for the goalkeeper is shown in Figure 16. Using more states results in a slightly higher accuracy, since a more precise action policy can be learnt. For example, the robot can walk forward to the ball, turn a little bit, and then kick the ball into the goal.

6 Conclusion and Discussion

The experiments show that a robot is able to learn how to score a penalty using Reinforcement Learning. With no goalkeeper defending the goal, walking the ball into the goal is the most successful behaviour for scoring. This can be explained by the fact that kicking is less reliable, because the ball can be hit at different positions, resulting in the ball moving in a slightly different direction and missing the goal. Having a goalkeeper that defends the goal results in a behaviour that scores by kicking, since the path taken by walking straight ahead with the ball is now obstructed by the goalkeeper, while needing on average 84% more trials to optimize.

The search space and thus the amount of trials can be reduced in different manners, without effecting the accuracy of the learnt policy. Firstly, intermediate rewards used to guide the learning by taking the distance between the ball and the robot, and between the ball and the goal into account, can reduce the amount of trials up to 75%. Furthermore, combining states additively instead of by multiplying to decrease the size of the state vector reduces the amount of trials with 58%.

Currently, the reward for scoring a goal and missing a goal have the same value with the difference being that when scoring the reward is applied positively (2) and when missed negatively (-2). However, most attempts do not lead to a goal, thus a negative reward is given more often, causing the positive reward to have relatively small impact on the Q-values. Future research can experiment with different rewards to increase the impact of positive rewards on the Q-values.

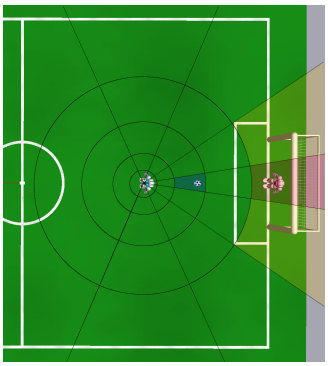


Figure 17: Non equally divided bins, more bins used in front of the robot and less bins behind the robot.

Furthermore, objects in states far away from the robot are often in the same bin. For example, the robot and the goal are in the same bin at the start of a trial (Figure 4b). However, having the goal and the goalkeeper in different bins seems to be more useful, since the goalkeeper is defending the goal. If the robot knows the difference between the positions of the goal and the goalkeeper, the robot can learn to kick past the goalkeeper into the goal. Future research can experiment with different state bins, for example, by using non equally divided bins, such as more (and smaller) bins in front of the robot and less (and bigger) bins behind the robot as shown in Figure 17.

Besides, the state space of the robot can be reduced to only that part the robot can observe (Figure 18), to ensure that it is more realistic to a real robot, and simpler to extend to a real robot. When extending to a real robot, a network that is trained with simulation can be used as initial network for training on a robot to optimise it further for use on the real robot, also known as transfer learning (Taylor and Stone, 2009).

This thesis presents a foundation of learning the actions that form a tactic. The next step is to extend this research to more complex scenarios, such as a moving goalkeeper, or a goalkeeper that learns its behaviour concurrently. Another complex scenario is a two-versus-one situation, in which a teammate is added to be able to learn team tactics.

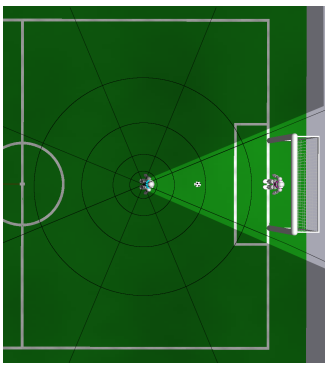


Figure 18: Non fully observable states, such as a real robot observes the world.

References

Baykal-Gürsoy, M. and Gürsoy, K. (2007). Semi-markov decision processes: Nonstandard criteria. *Probability in the Engineering and Informational Sciences*, 21(4):635–657.

Hester, T., Quinlan, M., and Stone, P. (2010). Generalized model learning for reinforcement learning on a humanoid robot. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 2369–2374. IEEE.

Lagrand, C., Negrijn, S., de Kok, P., van der Meer, M., van der Wal, D., Kronemeijer, P., and Visser, A. (2016). Team qualification document for RoboCup 2017, Nagoya, Japan. Technical report, University of Amsterdam, Science Park 904, Amsterdam, The Netherlands.

Laue, T. and Röfer, T. (2008). Simrobot-development and applications. In *International Conference on Simulation, Modeling and Programming for Autonomous Robots (SIMPAR)*.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533.

Ng, A., Coates, A., Diel, M., Ganapathi, V., Schulte, J., Tse, B., Berger, E., and Liang, E. (2006). Autonomous inverted helicopter flight via reinforcement learning. *Experimental Robotics IX: The 9th International Symposium on Experimental Robotics*, pages 363–372.

Pyeatt, L. D. (2003). Reinforcement learning with decision trees. In *Applied Informatics*, pages 26–31.

Röfer, T., Laue, T., Kuball, J., Lübken, A., Maaß, F., Müller, J., Post, L., Richter-Klug, J., Schulz, P., Stolpmann, A., Stöwing, A., and Thielke, F. (2016). B-Human team report and code release 2016.

Smart, W. D. and Kaelbling, L. P. (2000). Practical reinforcement learning in continuous spaces. In *ICML*, pages 903–910.

Smart, W. D. and Kaelbling, L. P. (2002). Effective reinforcement learning for mobile robots. In *Robotics and Automation, 2002. Proceedings. ICRA'02. IEEE International Conference on*, volume 4, pages 3404–3410. IEEE.

Stone, P., Sutton, R. S., and Kuhlmann, G. (2005). Reinforcement learning for robocup soccer keepaway. *Adaptive Behavior*, 13(3):165–188.

Sutton, R. S. and Barto, A. G. (1998). *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge.

Taylor, M. E. and Stone, P. (2009). Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, 10(Jul):1633–1685.

Watkins, C. J. C. H. (1989). *Learning from delayed rewards*. PhD thesis, King's College, Cambridge.