Robot Soccer

Using Reinforcement Learning to train the Nao shoot the Ball

D. Claes

Maastricht University
Faculty of Humanities and Sciences

June 28, 2009

Abstract

In robot soccer, various techniques of artificial intelligence have to be applied to develop a decently performing autonomous player. This paper gives an introduction into Reinforcement Learning and describes the design and implementation of two techniques - SARSA and Q-Learning - used to train the Aldebaran NAO-robot to shoot the ball. As action selection policies ϵ-greedy, ϵ-soft and softmax are used and different learning parameters are tested. In particular, constant and decreasing learning rates are compared. The experiments show that Q-Learning in combination with softmax outperforms given good parameter settings and and a constant learning rate. Furthermore, SARSA yields inferior performance and in general, higher values for the learning rate and discount values gave better rewards.

1 Introduction

Most of us know that computer programs, also named agents, outperform humans in controlled environments like a chess game. However, developing fully autonomously acting robots in real life situation is a very challenging task. The robots have to adapt to the dynamically changing environment and select actions under uncertainty, since not all input can be determined and many unknown states will arise.

Therefore, a popular challenge for researchers is the annual RoboCup competition [8], where various different sized robots compete on a soccer field. Robot soccer is a very interesting field of computer science and artificial intelligence, since it is a real time game with lots of sensor input. On the one hand, the environment is already very complex, but on the other hand, there are clear rules which make the environment structured and controllable. Additionally, high-level behaviour – teamplay and strategy – are as important as low-level behaviour – walking, running and shooting – in order to determine a good soccer player.

A well known statement by the RoboCup is that humanoid robots will beat the human world champion by 2050 [8]. However, the robots and programs are still very limited up to today. In 2004, a new league in the RoboCup was introduced, the Standard Platform League, where each competitor works with the same robots – the Sony Aibo. Since 2008, it was succeeded by the humanoid Aldebaran NAO robot (Nao) [1] with 21 degrees of freedom in the RoboCup edition and 25 in the academics edition. Hence, the focus lies on the software development instead of the hardware and it is possible to compare algorithms for playing, since they are running on the same hardware.

The research of this paper focusses on the shoot movement of the Nao. Several Reinforcement Learning techniques are explained, and later on used to train the Nao to shoot the ball. As learning and testing environment, the Cyberbotics Webots robot simulation software is used, since it provides a controllable environment and saves lots of resources while testing.

The main research questions are:

- What is Reinforcement Learning?
- How can it be applied for creating movements on the Nao?
- Which algorithm works best for creating a decent shoot-movement?

The rest of this article is structured as follows: Section 2 introduces the Reinforcement Learning domain, Section 3 presents the learning algorithms and gives an overview about the action selection policies used. Section 4 discusses the implementation on the Nao and Section 5 and 6 present the conducted experiments and the discussion of the obtained results. Section 7 and 8 close with conclusions and an outlook on future work.

2 What is Reinforcement Learning?

Reinforcement Learning is a variant of machine learning, were a computer program (agent), is learning its behaviour only based on a system of reward and punishment. The concept is useful to model difficult tasks were it is easier to define the aim of the task instead of knowing how it is done exactly. The method of Reinforcement Learning is imitating a way how a human being would learn certain behaviours. We interact with our environment by performing actions and afterwards, the effects can be observed. This idea of "cause and effect" is used through our entire life for building up the knowledge of our world.

2.1 Markov Decision Process

The Reinforcement Learning problem for agents can be formulated as a Markov Decision process with discrete time, finite states and finite actions, that is defined as follows [7]:

- Finite set of possible actions $a \in A$ and states $s \in S$
- Initial state: $s_0 \in S$
- Transition function: $T : S \times A \rightarrow \mathbb{R}(S)$, where $\mathbb{R}(S)$ is a probability distribution over S
- Reward function $R : S \times A \rightarrow \mathbb{R}$

The task for the agent can now be translated into certain steps:

- Observe the current state s
- Determine next action a based on a certain action selection policy $\pi : S \rightarrow A$
- Perform the selected action
- Observe the reward gained from this action
- Save information about this state / action pair

To evaluate a state, the agent has to know the utility of the current state. This is explained in the next section.

2.2 Utility Functions and Utility Estimation

A utility function estimates the utility of a given state s under policy π. The utility of a state equals its own reward plus the expected utility of its successor states, when following policy π [10], as shown in Equation 1. In other words, it tells the agent how profitable it is, to be in this state.

$$U^{\pi}(s) = E[\sum_{t=0}^{\infty} \gamma^t R(s_t) | \pi, s_0 = s] \quad (1)$$

This includes a discount factor $\gamma \in [0, 1]$ that controls the agent's desire to needs to achieve the goal quickly. A low γ value leads to a setting where the goal is tried to be reached as fast as possible, since the reward gets less with every step taken.

The utility functions are commonly not known for all states. Otherwise direct utility estimation could be used, which will not be explained here [10]. Hence, the utility values have to be estimated, since they are crucial to be able to accurately choose an action that maximises the total reward. As one solution for the estimation, the so-called temporal difference learning can be used. It takes the already observed transitions to update the values of the observed states:

$$U^{\pi}_{t+1}(s) \leftarrow U^{\pi}_t(s) + \alpha(R(s) + \gamma U^{\pi}_t(s') - U^{\pi}_t(s)) \quad (2)$$

where $\alpha \in [0, 1]$ is the learning rate parameter and t the current time step. This learning rate should be a decreasing function in order to let $U^{\pi}(s)$ converge to the correct value. In this way, an estimate of the final reward is calculated at each state and the state-action value is updated on every step t on the way. This method is often called "bootstrapping", since it uses itself to update the utility values.

2.3 Policy Learning

In general, there are two ways to learn the estimated utility values, namely on-policy and off-policy methods. The on-policy methods use the action selection policy to make decisions. The utility functions are updated using the results from executing actions determined by an action selection policy [11].

Off-Policy methods use different policies for estimating the utility function and for the actual behaviour. The algorithms use hypothetical actions to update the utility functions. This is in contrast to on-policy methods which update value functions based strictly on experience. What this means is off-policy algorithms can separate exploration from control, which on-policy algorithms cannot. In other words, an agent trained using an off-policy method may end up learning behaviour that it did not necessarily exhibit during the learning phase [11].

The next section introduces one algorithm for each type, used for training the Nao.

3 How to use Q-values for learning?

A Q-value is the expected utility when starting at s and taking action a and following policy π afterwards. Therefore, the following equation holds:

$$U^{\pi}(s) = \max_a Q(s, a) \quad (3)$$

If the transition model is known, it is easily possible to correctly calculate all exact Q-values:

$$Q(s, a) = R(s) + \gamma \sum_{s'} T(s, a, s') \max_{a'} Q(s', a') \quad (4)$$

However, most of the time, the transition model is not known and therefore, these values have to be estimated.

3.1 Q-Learning

A famous method of Q-Learning was proposed by Watkins [12] in 1989. It iteratively approximates the Q values by an estimation function $\hat{Q}$:

$$\hat{Q}(s, a) = \hat{Q}(s, a) + \alpha(R(s) + \gamma \max_{a'} \hat{Q}(s', a') - \hat{Q}(s, a)) \quad (5)$$

where α is the learning rate, which can be a constant or a decreasing function depending on visits of each state and γ is the discount value. This method is an example of an off-policy learning algorithm. It only uses the current estimation $\hat{Q}$ and the observation of the current state's reward ($R(s)$) to update the Q-values. A big advantage is that the transition function and reward function do not need to be known in advance. For this reason, it is called a model-free method [10]. Algorithm 1 shows the implementation used for this paper.

Algorithm 1 Q-Learning

```
Initialize Q(s,a)
for all episodes do
    reset state s
    while s not terminal do
        Choose a from S using policy π
        take action a
        observe reward R(s) and following state s'
        Q(s,a) ← Q(s,a) + α(R(s) + γ max_a' Q(s',a') − Q(s,a))
    end while
end for
```

3.2 SARSA

Another algorithm that can be used to approximate the Q-values is called SARSA [11]. This names has its origin in the parameters used in the update function:

$$\hat{Q}(s,a) = \hat{Q}(s,a) + \alpha(R(s) + \gamma\hat{Q}(s',a') - \hat{Q}(s,a)) \quad (6)$$

where s, a, r, s' and a' are current state, current action determined by policy π, current reward, following state and following action using policy π respectively. As in Q-Learning the other parameters α and γ describe the learning rate and discount value respectively. As can be seen, due to this update function, it is an on-polcy algorithm. The Q-values are updated using the following action based on the used action selection policy instead of only on the Q-values as in the previous algorithm. Algorithm 2 shows the implementation used for this paper.

Algorithm 2 SARSA

```
Initialize Q(s,a)
for all episodes do
    reset state s
    Choose a from S using policy π
    while s not terminal do
        take action a
        observe reward R(s) and following state s'
        get action a' from s' using policy π
        Q(s,a) ← Q(s,a) + α(R(s) + γQ(s',a') − Q(s,a))
        s ← s' and a ← a'
    end while
end for
```

It completely depends on the task, which learning algorithms performs better. Furthermore, the parameters have to be selected carefully, since they have major influences on the performances of the algorithms.

3.3 Action selection policies

There exist several policies to decide which action should be taken next. Each one has to provide a good balance between exploration of the unknown and exploitation of the already learnt. Otherwise, if some decent action was found before, this will be chosen again and again, despite there may exist a better one. Therefore, these policies are usually "soft", meaning that every possible action has a non zero probability of being taken. Three common action selection policies are [11]:

- ε-greedy: The action with the highest estimated reward is chosen most of the time, therefore it is called greedy. With a small and decreasing probability ε, a random action will be performed, in order to provide the exploration in the beginning. These random actions are selected uniformly and independent of the utility estimations. Since ε is decreasing to zero in the limit, this policy becomes a static greedy policy without any exploration. Therefore, the decrease has to be set carefully.
- ε-soft: It is very similar to ε-greedy, however without the decrease in ε. The best action is selected with probability 1 - ε + $\frac{\varepsilon}{|A|}$ and the rest of the time a random action is chosen uniformly with probability $\frac{\varepsilon}{|A|}$. Hence, if it is running for infinite time, it ensures that the optimal actions are discovered, since each action will be tried out an infinite number of times.
- softmax: One possible drawback of ε-greedy and ε-soft is that they select the exploration actions uniformly. Therefore, the worst possible action will be selected with the same probability as the second best. The softmax policy uses a Gibbs function to overcome this problem. Each action gets weighted, according to their action-value estimate at the current state($Q(s,a)$):

$$P(s,a) = \frac{e^{Q(s,a)/\tau}}{\sum_b e^{Q(s,b)/\tau}} \quad (7)$$

were τ is the temperature. This function provides a probability density function, which can be used to select the actions based on their weight. A high temperature leads to higher probabilities of taking worse actions. This approach is favourable, when the worst actions should not be taken with a high probability.

Again, there is no general rule, which of these policies lead to the best result. Each task is different in how the policy influences the learning. Therefore, all possibilities should be tried out.

4 Reinforcement Learning on the Nao

The Nao is a humanoid robot produced by Aldebaran Robotics. As said in the introduction, it is the successor

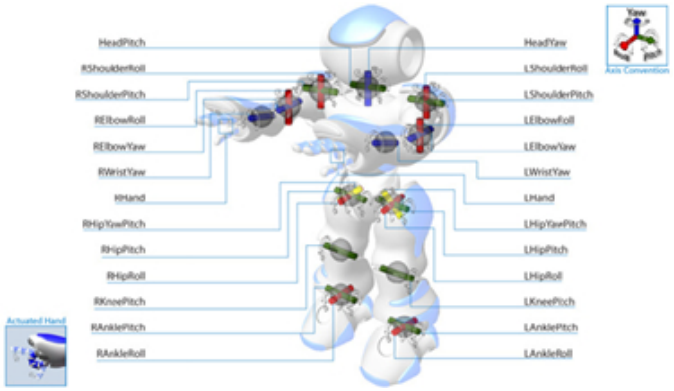

Figure 1: The Nao academic edition with its 25 degrees of freedom and the movement reference [1]. For a larger version see appendix Figure 11.

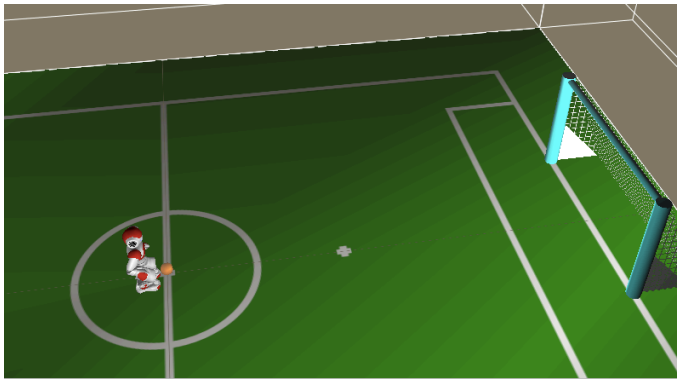

Figure 2: The setup in the simulation software Webots. Ball at position (0, 0), Nao at (0.153, -0.049).

of the Sony Aibo in the Standard Platform League of the RoboCup [9]. The academic edition has 25 degrees of freedom, whereas the RoboCup edition only has 21 [1], since the wrist- and hand-joints are fixed. For this paper it does not make any difference, since it focusses on the shoot movement and the hands do not provide any more stability during the shot. Figure 1 shows the Nao with its 25 joints and directions in which they can be turned. In the appendix Figure 12 shows an overview of all the joints and their movement range in degrees.

For using Reinforcement Learning on the Nao, it is necessary to represent the states and actions and the reward function in a way, such that each state and action can be uniquely identified. Likewise, the reward function needs to give a good feedback for the algorithm, if the current state is feasible or should rather be avoided.

4.1 How to represent the states and the actions?

Each joint is divided into a discrete number of positions, where each state is then represented by the current joint positions and the current time step starting from the point were the predefined movement stops. This is done by dividing the total range of each controlled joint into n discrete equal length buckets that are then labeled from 0 to $n-1$. The current state is then calculated by checking for each joint in which bucket the current position is in and the current time step.

Each actions is then defined as a change of one step of the current joint position in either direction. Where multiple joints can be manipulated at the same time. Specifically, this means that the length of one bucket of the joint to be moved is determined, and this value is then added to or subtracted from the current joint's position. Additionally, this facilitates the detection of an not supported move, since if the joint is currently in state 0, it cannot move further back, or in state $n-1$ it cannot move further up.

4.2 How to limit the states and actions?

The problems at hand are the many degrees of freedom and the continuous joint positions, since they lead to infinitely many states. Even when discretizing the joint values, the number of states rise exponentially for each additional joint. Therefore, it was chosen to focus only on the shooting leg, which has six joints. However, the RHipYawPitch joint is directly coupled with the LHipYawPitch, so it was chosen not to use this joint, too.

Furthermore, the actions were limited to manipulate two joints at most at the same time. Since each joint can be manipulated in two directions, there are $5*4*2*2 = 80$ possible actions already with five joints.

4.3 The reward function

At the beginning, a very sparse reward function was used, giving only punishment for falling over and rewards if shooting a goal. This turned out not to be a sufficient way, since even after a two hour training run of the algorithms, no decent movement could be accomplished. Therefore, more rewards were introduced.

A further reward was introduced for the speed and the direction of the shot. Naturally, a straight shot with a high speed got a higher reward than a diagonal shot. Furthermore, a better model for the stability of the robot was implemented. In the beginning, only by reading out the acceleration values of the Nao, which are provided in each direction, it was determined if the robot has fallen over or not. However, this measure said nothing about the stability of the previous states, since it only can distinguish between standing and lying on the ground. Afterwards, an approximation of the Zero Moment Point (ZMP) was introduced to provide a better model for the stability.

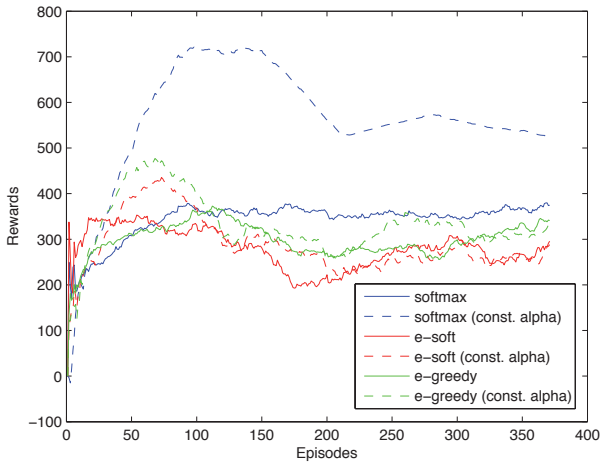

Figure 3: Results of the training with SARSA with parameters $\alpha = 0.9$ and $\gamma = 0.9$ for the different action selection policies

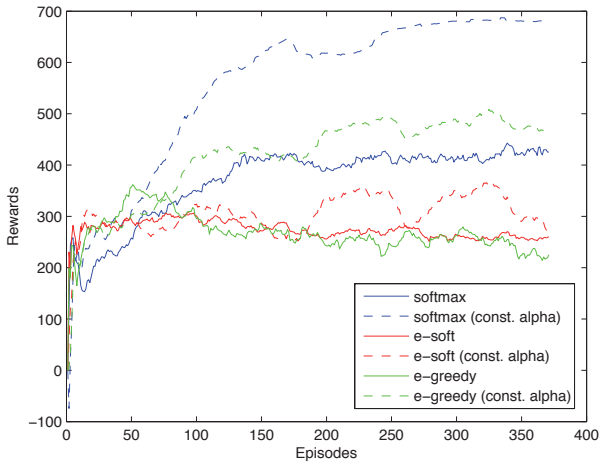

Figure 4: Results of the training with Q-Learning with parameters $\alpha = 0.9$ and $\gamma = 0.9$ for the different action selection policies

Zero Moment Point

The ZMP is the point were the at the ground were the total inertia force equals zero. This is the point on the ground below the center of masses (CoM). The concept of ZMPs is commonly used for controlling the walking of a humanoid robot, which can be compared to the well-known problem of balancing an inverted pendulum, where most of the mass is located at the top of the pendulum. This translates to the Nao, since the CoM is located at the torso and only the two legs are supposed to support and move the Nao, while keeping it from falling. The concept of the ZMP assumes that the CoM moves a long a plane at constant height (z_h), called the constraint plane [5]. An easy way to calculate the current position of the ZMP is [4]:

$$zmp_x = x_p - \frac{z_h}{g} * \ddot{x_p} \tag{8}$$

$$zmp_y = y_p - \frac{z_h}{g} * \ddot{y_p} \tag{9}$$

where x_p and y_p the projection on the floor of the position of the CoM are. $\ddot{x_p}$ and $\ddot{y_p}$ are the accelerations in x- and y-direction respectively, g is the gravitational constant and z_h is the height of the constraint plane. With the ZMP and the inverted pendulum model, a support polygon can be determined, which is the region of the ZMP where the robot does not fall over. If the ZMP is outside this support polygon, it is very likely that the pendulum or in this case the Nao cannot be saved from falling over. This region can be approximated by experiments. From the Equations 8 and 9 follows, that the height of the constraint plane only is a linear scaling factor. Hence, for the approximation of a balance region, it does not play a role. For simplicity, only a support rectangle was created by experiment and it was assumed that the projection of the CoM on the ground is at (0, 0) if the Nao is standing upright. Furthermore, the differences of the acceleration senor's values of the Nao at two consecutive time steps can directly be used as $\ddot{x_p}$ and $\ddot{y_p}$ in order to calculate an approximation of the position of the ZMP.

The reward for each state was then added with a positive value, depending on how far the current ZMP was inside this support rectangle or a high negative value, if it was outside it, since these states should be avoided to keep the Nao in a stable position.

4.4 Motivation for the use of the Webots Simulator

For Reinforcement Learning it is crucial to be able to reproduce certain states, for instance the starting state. Furthermore, many learning episodes have to be accomplished in order to get good results.

On the one hand, there is the usage of the real Nao, which would provide realistic experiments. Likewise, the understanding of the real mechanics would be supported. However, the Nao's hardware is very fragile and it is very time consuming to perform all the tests in realtime. The robot and the ball always have to be reset to the starting position. Furthermore, there is the possibility that the algorithm work in theory, but only hardware failures prevent it from performing correctly.

Therefore, it was chosen to use the Cyberbotics Webots robot simulation [3] software together with the robotstadium [13] package. This combination provides an accurate soccer simulation environment used by many

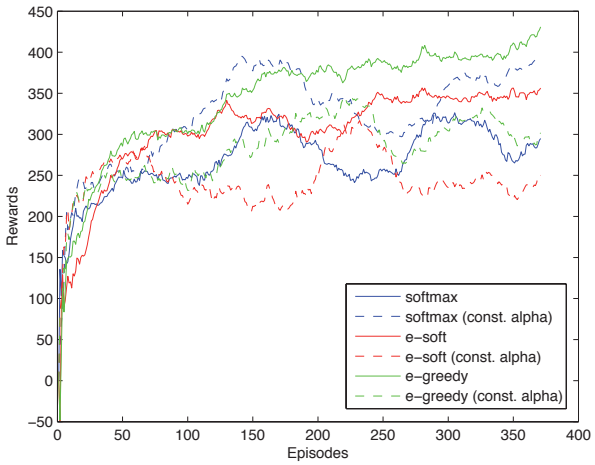

Figure 5: Results of the training with SARSA with parameters $\alpha = 0.9$ and $\gamma = 0.5$ for the different action selection policies

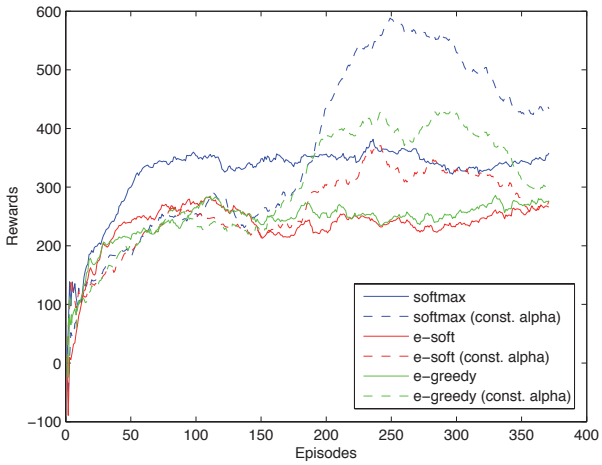

Figure 6: Results of the training with Q-Learning with parameters $\alpha = 0.9$ and $\gamma = 0.5$ for the different action selection policies

people, e.g. in [5], [4] and [2]. There it is possible to reset the robot and the ball to specific positions by a supervising program. Additionally, the correct ball position can be read out, which comes handy, since the image processing was not a part of this research. Another advantage is the possibility to run the simulation faster than realtime, which made it possible to use more training episodes for the algorithms. Some other important issues to consider are that the Nao will always work in the simulation, since it cannot break or have hardware issues.

The only drawbacks are that the simulation environment is probably to idealistic without too much noise and that the used software cannot be transferred directly to the real Nao.

5 Experiments

The Webots software package provides a predefined shoot movement. The first part of it was used as starting movement, since it nicely balances the Nao on the right foot and then performs the shoot movement with the left leg. Afterwards, values for the joints of the shooting leg were blanked out from the movement and controlled by the Reinforcement Learning algorithm.

For each episode of the experiments the ball was reset to the (0, 0) position, which is the center point of the field. For the positioning of the Nao, a hill climbing was set up to determine on which position the provided shoot movement got the best reward. As a result, the Nao was positioned at (0.153, -0.049) with respect to the ball. The setup in Webots is shown in Figure 2. The different settings used for the training were:

- Learning algorithm used (SARSA or Q-Learning)
- Action selection policy used:
  - softmax with $\tau = 1$
  - ε-greedy with $\varepsilon_0 = 0.1$ and decreasing by 0.5% after each episode
  - ε-soft with $\varepsilon = 0.1$
- Learning parameters (α and γ), all combinations of the values 0.5 and 0.9
- Constant α or a function of α decreasing over time with each visit of a state($\alpha(s, visits) = \alpha_0^{visits}$)

Each setting was run three times independently for 400 episodes. After each episode, the Nao's and the ball's position were reset to the values mentioned above. The resulting rewards for each setting were recorded and the mean across the three runs was plotted with a moving average window of 30. Furthermore, using Welch's procedure, a 90% confidence interval for the mean reward of each trial is calculated. The length of the warmup period was determined by inspection of the graphs [6].

6 Results

Table 1 gives a complete overview with the results of all the test runs, where the estimated mean intervals of the reward is calculated. As can be seen, when comparing Figure 3 and Figure 4, softmax outperforms all other action selection policies, when using $\alpha = 0.9$ and $\gamma = 0.9$, especially, when using a constant learning rate. Additionally, it can be noticed that ε-soft and ε-greedy perform very similarly. For instance, in Q-Learning, there is

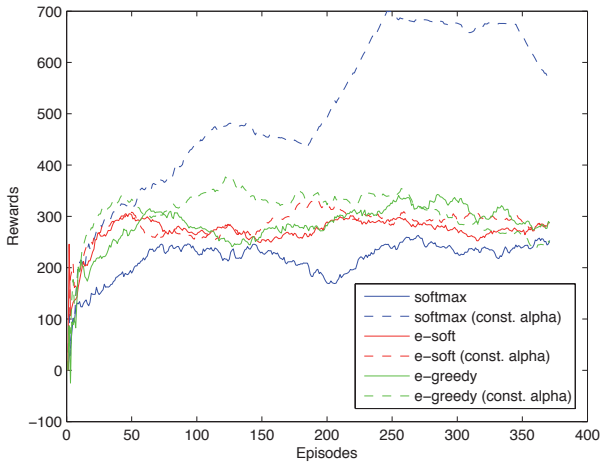

Figure 7: Results of the training with SARSA with parameters $\alpha = 0.5$ and $\gamma = 0.9$ for the different action selection policies

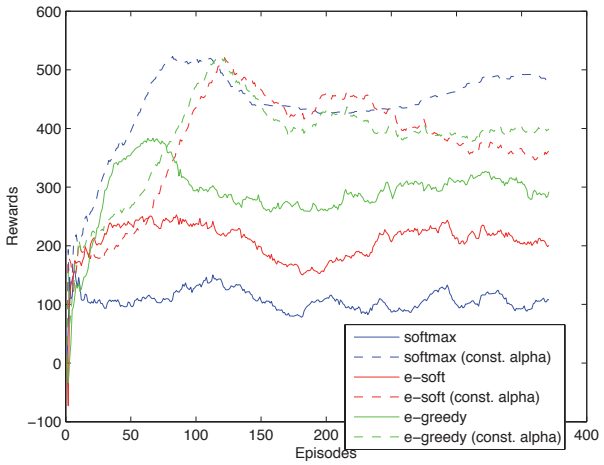

Figure 8: Results of the training with Q-Learning with parameters $\alpha = 0.5$ and $\gamma = 0.9$ for the different action selection policies

hardly any difference, when using a decreasing α. Furthermore, Q-Learning outperforms SARSA with these settings.

When decreasing the discount factor γ to 0.5, the overall performance decreases, as seen in Figure 5 and Figure 6. There again, softmax is performing best, when using constant α. However, with SARSA using ε-soft and ε-greedy with a decreasing α resulted in the best rewards. Again, the general trends of ε-soft and ε-greedy were similar for both learning techniques.

The next experiment conducted was with a decreased α value to 0.5. Figure 7 and Figure 8 present the results for this session with $\gamma = 0.9$ and Figure 9 and Figure 10 with $\gamma = 0.5$. Most remarkably is the fact that now softmax performed worst with decreasing learning rate, along with ε-soft and ε-greedy, which scored significantly more with a constant learning rate. With SARSA all possibilities perform quite similarly, except for the combination softmax with constant α, which outperformed for both settings. On the other hand, Q-Learning shows observable differences for a decreasing α. Here, with $\gamma = 0.9$, ε-greedy gives the best settings, whereas ε-soft resulted in the best rewards for $\gamma = 0.5$. When comparing the simulations with $\alpha = 0.9$ and $\alpha = 0.5$, it can be seen that in general, the higher value for the learning rate led to better results.

To sum up, Table 1 shows as first result that Q Learning outperformed SARSA in almost every case. This can be explained due to the fact that Q-Learning is a off-policy learning algorithm. Therefore, it is more exploration invariant. In this case, most explorations would most probably lead to the robot falling over and thus, to worse rewards.

A second result was that the higher learning rate provided better results. This is probably due to the high number of states and actions. The system was not close to converge after 400 episodes. Only a small percentage of possibilities was tried out, hence a higher learning rate provided the chance that when a decent move was found, it was repeated the next few times, since it had a major effect on the Q-values.

Additionally, it was found that a constant learning rate provided better results than a decreasing one. Furthermore, with $\alpha = 0.5$ the difference between the constant alpha and the decreasing was even more significant. This should be obvious when looking at the decreasing function that was α_0^{visits}. Therefore, the decrease with $\alpha = 0.5$ was a lot faster than with $\alpha = 0.9$. This fact, in combination with the fact that a higher α in general led to better results, explains this significant difference.

Likewise, softmax performed the worst when using the decreasing function of α. This is due to the fact that softmax really depends on the Q-values. It performs very good as soon as a few decent movements are found and changed the Q-values such that the best movements get selected with a very high probability. But with a low learning rate, the impact of a good trial is not that high. Hence, the probabilities do not change that much and the actions leading to the robot falling over or not hitting the ball properly are chosen almost as much as the better ones.

The last result was that a higher value for γ performed better. This is due to the representation of the rewards in the experiments. Mostly, the rewards were given in the end, for instance if the ball went into the goal could only be determined after the shot has been

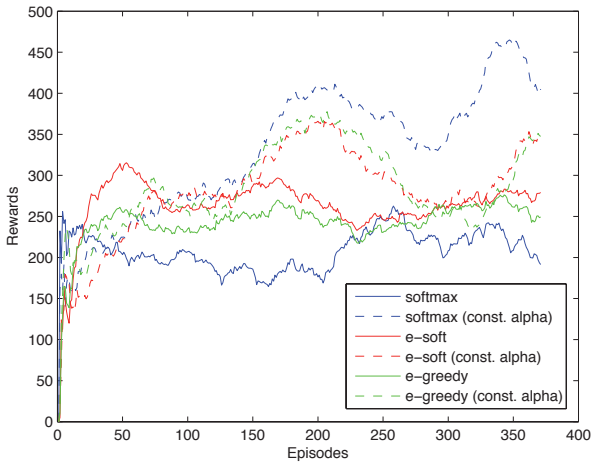

Figure 9: Results of the training with SARSA with parameters $\alpha = 0.5$ and $\gamma = 0.5$ for the different action selection policies

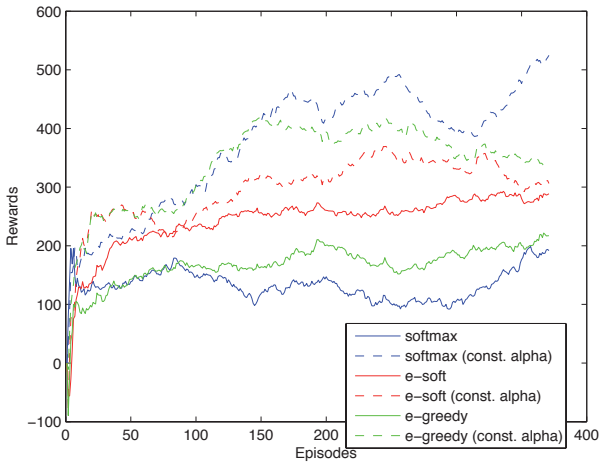

Figure 10: Results of the training with Q-Learning with parameters $\alpha = 0.5$ and $\gamma = 0.5$ for the different action selection policies

performed. Additionally, since the robot was already standing stable on one leg. Thus, the falling over happened most of the time at the very end of the movement, when the robot has not yet learned that it is feasible to put its leg back on the ground.

All in all, the best settings in these experiments were Q-Learning with softmax and $\alpha = 0.9$ (const.) and $\gamma = 0.9$. The obtained shoot movement was very similar to the predefined. However, the predefined shoot movement had less strength since it only came close to the penalty box, whereas the learned movement shot into the goal.

7 Conclusion

In this paper, two different Reinforcement Learning techniques – SARSA and Q-Learning – were introduced and used to train the Nao to shoot the ball. Three different action selection policies are compared with different settings.

To conclude, it can be said that it is feasible to use Reinforcement Learning on the Nao for training the shoot movement. However, the settings have to be chosen with great care and several limitation have to be taken into account. For instance, the high number of joints and the possibility to set them to continuous values are some problems at hand. Limiting those to some discretized buckets and only controlling a low number of joints provided some decent results. Nevertheless, in order to uses this approach a predefined movement has to be provided, and due to the exponential increase in states for each additional joint, the controlled joints have to be selected carefully.

Furthermore, each episode takes a lot of time, which means that only comparably few states can be explored. This experiment had roughly five million possible states, of which not all could be reached. However, already 400 learning episodes took around an hour in real time. Of course, with the simulator it was possible to speed the simulation up to four times real time speed, but nevertheless, for validation each training set had to be repeated several times.

Regarding the learning parameters, experiments showed that a high learning rate α and a discount value γ close to 1 led to the best results. Although in theory only a decreasing learning rate is proven to converge, it is not feasible for this setup. There are too many states, which have to be explored and too many actions that can be taken. Most rewards were allocated in the end of the movements, which explains that the higher discount value gave better results.

8 Further Outlook

This section will proposes some ideas that could be researched in the future. At first, the Zero Moment Point (ZMP) model can be used as a feedback loop in order to stabilize the Nao after the shot. For instance, from the time when the sensor in the shooting leg notices the ball hit, the movement gets adapted in order to bring the ZMP back to the center.

Additionally, it can be tested if a better shooting movement can be performed, when allowing the ZMP to be outside the support polygon. This is the way a human would shoot the ball, but it would need to control a lot more joints.

Another improvement to the current method could

be done by reducing the states to only those ones that can be achieved consecutively. Since it is only possible to move a joint gradually, not all states can be achieved by the actions. Hence, the state-space could be reduced immensely, when finding a suitable way to only represent the achievable states.

A different approach could be to use Multi Agent Learning methods where each joint is represented as one agent and only agents close to each other have an effect on their behaviour. This is promising, since it reduces the number of dependencies and in this way more joints could be controlled at the same time. The agents then try to collaborate to achieve the same goal, e.g. shoot the ball straight.

One last idea is to model the values of the joints over time by using Neural Networks. These networks can then be varied with Gaussian Noise and tested if the varied functions perform better than the old ones. These variations can be improved gradually by using Reinforcement Learning.

References

[1] Aldebaran Robotics (2008). Nao, the ideal partner for research and education in the field of robotics. `http://www.aldebaran-robotics.com/Files/NaoAcademicsEd_V3.3.pdf`.

[2] Cherubini, A., Giannone, F., Iocchi, L., Lombardo, M., and Oriolo, G. (2009). Policy gradient learning for a humanoid soccer robot. *Robotics and Autonomous Systems*, Vol. 57, pp. 808–818.

[3] Cyberbotics Ltd. (2009). Webots reference manual. `http://www.cyberbotics.com/cdrom/common/doc/webots/reference/reference.pdf`.

[4] Czarnetzki, S., Kerner, S., and Urbann, O. (2009). Observer-based dynamic walking control for biped robots. *Robotics and Autonomous Systems*, Vol. 57, pp. 839–845.

[5] Kajita, S., Kanehiro, F., Kaneko, K., Yokoi, K., and Hirukawa, H. (2003). Biped walking pattern generation by using preview control of zero moment point. *IEEE International Conference on Robotics and Automation*.

[6] Law, A.M. (2007). *Simulation Modeling and Analysis*. McGraw Hill, 4th edition.

[7] Mitchell, T. (1997). *Machine Learning*. McGraw Hill, 1st edition.

[8] RoboCup (2003). The robocup2003 presents: Humanoid robots playing soccer. `http://www.robocup.org/Press/pr/RoboCup2003_020603eng.pdf`.

[9] RoboCup (2009). Standard platform league history. `http://www.tzi.de/spl/bin/view/Website/History`.

[10] Russell, S. and Norvig, P. (2003). *Artificial Intelligence A Modern Approach*. Pearson Education, Inc., 2nd edition.

[11] Sutton, R.S. and A.G.Barto (1998). *Reinforcement Learning: An Introduction*. MIT Press, 1st edition.

[12] Watkins, C.J.C.H. and Dayan, P. (1992). Q-learning. *Machine Learning*, Vol. 8, pp. 279–292.

[13] www.robotstadium.org (2009). NaoV3R Model. `http://robotstadium.org/index.php?option=com_content&task=view&id=28&Itemid=29`.

A Estimated means

Algorithm	Action Selection Policy	α	γ	α decr.	α const.
Q-Learning	ϵ-soft	0.9	0.9	(255.04, 258.30)	(456.11, 461.87)
Q-Learning	ϵ-greedy	0.9	0.9	(268.70, 271.12)	(311.85, 317.87)
Q-Learning	softmax	0.9	0.9	(406.51, 410.01)	(635.95, 644.31))
SARSA	ϵ-soft	0.9	0.9	(297.94, 304.10)	(310.81, 316.13))
SARSA	ϵ-greedy	0.9	0.9	(260.10, 266.86)	(265.20, 270.61))
SARSA	softmax	0.9	0.9	(357.32, 358.96)	(589.07, 603.02))
Q-Learning	ϵ-soft	0.9	0.5	(256.53, 259.01)	(331.81, 346.00)
Q-Learning	ϵ-greedy	0.9	0.5	(242.23, 245.54)	(285.39, 293.98))
Q-Learning	softmax	0.9	0.5	(344.67, 346.99)	(414.62, 438.33)
SARSA	ϵ-soft	0.9	0.5	(373.90, 379.38)	(296.72, 301.77)
SARSA	ϵ-greedy	0.9	0.5	(327.79, 331.45)	(241.90, 247.39))
SARSA	softmax	0.9	0.5	(282.10, 287.316)	(347.96, 353.84)
Q-Learning	ϵ-soft	0.5	0.9	(288.01, 291.68)	(413.06, 420.35)
Q-Learning	ϵ-greedy	0.5	0.9	(202.09, 206.97)	(416.24, 425.35))
Q-Learning	softmax	0.5	0.9	(104.78, 108.01)	(455.23, 460.70)
SARSA	ϵ-soft	0.5	0.9	(291.42, 296.53)	(313.91, 320.75)
SARSA	ϵ-greedy	0.5	0.9	(274.17, 276.73)	(291.39, 294.74))
SARSA	softmax	0.5	0.9	(225.11, 229.33)	(567.90, 587.41))
Q-Learning	ϵ-soft	0.5	0.5	(178.91, 182.27)	(375.36, 380.66)
Q-Learning	ϵ-greedy	0.5	0.5	(262.11, 265.06)	(318.98, 324.26)
Q-Learning	softmax	0.5	0.5	(126.56, 131.49)	(422.81, 432.76)
SARSA	ϵ-soft	0.5	0.5	(247.90, 250.44)	(298.63, 307.42)
SARSA	ϵ-greedy	0.5	0.5	(264.75, 267.66)	(300.74, 307.30)
SARSA	softmax	0.5	0.5	(206.07, 210.91)	(362.22, 372.59)

Table 1: 90% confidence intervals of the estimated mean of the reward of the two algorithms in combination with all action selection policies and different α- and γ-values.

B The Nao

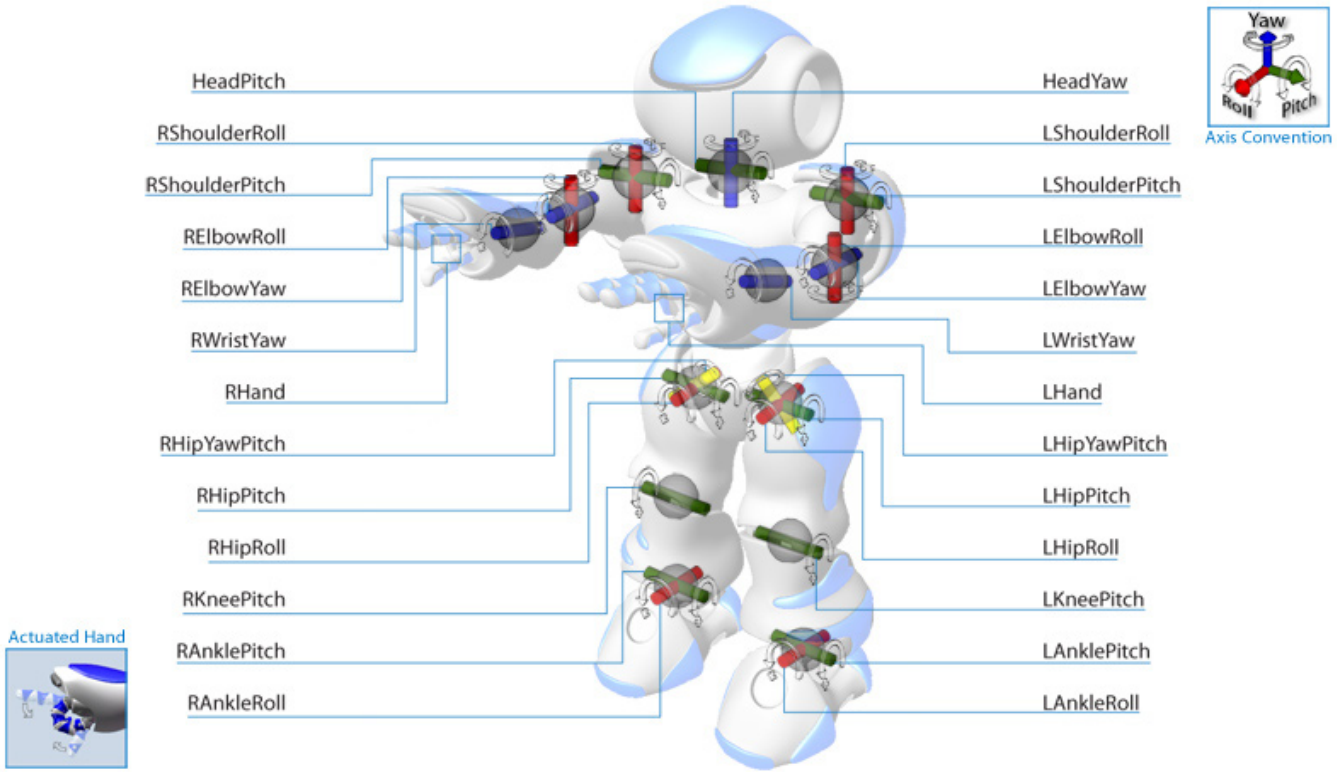

Figure 11: The Nao and its 25 degrees of freedom.

PART	JOINT NAME	MOTION	RANGE (degrees)
Head	HeadYaw	Head joint twist (Z)	-120 to 120
	HeadPitch	Head joint front & back (Y)	-45 to 45
Left arm	LShoulderPitch	Left shoulder joint front & back (Y)	-120 to 120
	LShoulderRoll	Left shoulder joint right & left (Z)	0 to 95
	LElbowRoll	Left shoulder joint twist (X)	-120 to 120
	LElbowYaw	Left elbow joint (Z)	-90 to 90
	LWristYaw	Left wrist joint twist (X)	-105 to 105
	LHand	Left hand	open & close
Left leg	LHipYawPitch	Left hip joint twist (Z45°)	-90 to 0
	LHipPitch	Left hip joint front and back (Y)	-100 to 25
	LHipRoll	Left hip joint right & left (X)	-25 to 45
	LKneePitch	Left knee joint (Y)	0 to 130
	LAnklePitch	Left ankle joint front & back (Y)	-75 to 45
	LAnkleRoll	Left ankle joint right & left (X)	-45 to 25
Right leg	RHipYawPitch	Right hip joint twist (Z45°)	-90 to 0
	RHipPitch	Right hip joint front and back (Y)	-100 to 25
	RHipRoll	Right hip joint right & left (X)	-45 to 25
	RKneePitch	Right knee joint (Y)	0 to 130
	RAnklePitch	Right ankle joint front & back (Y)	-75 to 45
	RAnkleRoll	Right ankle right & left (X)	-25 to 45
Right arm	RShoulderPitch	Right shoulder joint front & back (Y)	-120 to 120
	RShoulderRoll	Right shoulder joint right & left (Z)	-95 to 0
	RElbowRoll	Right shoulder joint twist (X)	-120 to 120
	RElbowYaw	Right elbow joint (Z)	-90 to 90
	RWrist Yaw	Right wrist joint twist (X)	-105 to 105
	RHand	Right hand	open & close

Figure 12: The motion range for each joint of the Nao.