Continuous State Space Q-Learning for control of Nonlinear Systems

ten Hagen, S.H.G.

Citation for published version (APA):

General rights
It is not permitted to download or to forward/distribute the text or part of it without the consent of the author(s) and/or copyright holder(s), other than for strictly personal, individual use, unless the work is under an open content license (like Creative Commons).

Disclaimer/Complaints regulations
If you believe that digital publication of certain material infringes any of your rights or (privacy) interests, please let the Library know, stating your reasons. In case of a legitimate complaint, the Library will make the material inaccessible and/or remove it from the website. Please Ask the Library: http://uba.uva.nl/en/contact, or a letter to: Library of the University of Amsterdam, Secretariat, Singel 425, 1012 WP Amsterdam, The Netherlands. You will be contacted as soon as possible.
Chapter 5

Neural Q-Learning using LQRQL

5.1 Introduction

The LQRQL approach was derived for linear systems with quadratic costs. Although this approach can also be applied to nonlinear systems or other cost functions, the resulting feedback will always be linear. This is a consequence of the restriction that the Q-function is quadratic. The use of a quadratic Q-function was motivated by the possibility to use a linear least squares estimation to obtain the parameters of the Q-function. Then the parameters of the linear feedback follow directly from the parameters of the Q-function.

If the system is nonlinear, it is very likely that the Q-function is not quadratic. In such situations it is possible to use a general approximator, like a feed forward network, to approximate the Q-function. The feedback can be obtained by finding for all states the control action that minimizes the feedback function. We will show that this can have the effect that the feedback function is not a continuous function and we will explain why it is desirable to have a continuous feedback function. In order to have a continuous feedback function, another network can be used to represent the feedback function. This is the actor/critic configuration as described in section 2.3. This has the drawback that two networks have to be trained, where the training of one network depends on the other. This makes the training very hard.

The choice is either to have a discontinuous feedback or to use two networks. In this chapter we propose a method in which we only use one network and still have a continuous feedback function. The idea is to combine LQRQL with a carefully chosen network and we called this Neural Q-Learning. The feedback can be derived in the same way as in LQRQL.

The simulation and real experiments were based on the same nonlinear system as in chapter 4. We applied the Neural Q-Learning method and compared the result with Gain Scheduling based on the extended LQRQL approach. The resulting nonlinear feedback functions have to be globally valid. So instead of focusing on a local part of the state space, we looked at larger parts of the state space.
5.2 Neural Nonlinear Q-functions

Instead of a quadratic Q-function, a general approximator like a feed forward neural network can be used. It describes the function

$$Q(\xi, w) = \Gamma_o(w_o^T \Gamma_h(W \xi + b_h)^T + b_o),$$

(5.1)

with $\Gamma_o$ and $\Gamma_h$ the activation functions of the units in the output and hidden layer. The rows of matrix $W$ contain the weight vectors $w_{h,i}$ of the hidden units and the vector $b_h$ contains the corresponding biases. The weights of the output unit are given by $w_o$ and the bias by $b_o$. The network has only one output $Q(\xi, w)$, where the vector $w$ indicates all weights of the network. The $\xi$ represents the input of the Q-function.

If a network is used to represent a Q-function we might obtain a typical function as in figure 5.2(a). This is drawn for a scalar state and control vector, and the weights of the network are picked randomly. It is clear that this Q-function is a smooth continuous function. Given this Q-function the feedback has to be determined.

According to (2.15) and (3.15) the greedy feedback is computed by taking the minimum value of the Q-function for each state. In figure 5.2(b) the top view of the Q-function is shown with lines indicating similar Q-values. In this plot also the greedy feedback function is shown that is computed by taking for each $x$ the value of $u$ for which the Q-value is minimal. Using the greedy feedback from the network has two drawbacks:

- The computation of the greedy action involves finding the extremum of a nonlinear function. In general this is not a trivial task.

- Even when the Q-function is smooth as in figure 5.2(a), it is still possible that the greedy feedback function (shown in figure 5.2(b)) is not a continuous function. This

![Diagram](image)

Figure 5.1. The network with two hidden units
Figure 5.2. The Q-function is formed by a feed forward network. In (b) we see the top view of the Q-function with lines indicating the height. The bold line indicates the greedy feedback.

...can cause problems for real applications when there is noise in the system. Near the discontinuities a little noise can have a high influence on the control action, so the behavior of the system becomes very unpredictable.

The second drawback can be overcome by introducing a Dynamic Output Element [60]. The static feedback is followed by a low pass filter, removing the effects of noise at discontinuities of the feedback function. In this way the control action is not only based on the current state value, but also on the previous control actions. This approach is not a solution to the first drawback.

One approach that removes both drawbacks is the Actor/Critic approach described in chapter 2. A second function approximator is introduced to represent the feedback function. If a continuous differential function is used as actor, the second drawback is overcome. By training the actor based on the critic also the first drawback is removed.

The main difference between the actor/critic configuration and Q-Learning is shown in figure 5.3 and is the way in which the feedback is derived from the Q-function. In the actor/critic configuration the actor is trained based on the critic. In the Q-Learning configuration the Q-function implicates the feedback, so that the Q-function and feedback are represented by one and the same network.

The two networks in the actor/critic configuration have to be trained, and this is the major problem of this configuration. Training two networks means that training parameters and initial settings have to be selected for two networks. It is very hard to determine beforehand the appropriate setting for the training of the networks. Since the settings influence the result, the interpretation of the results is very hard.
In the LQRQL approach the greedy feedback is computed by setting the derivative of the Q-function to the control action to zero. So for LQRQL the quadratic Q-function implicates the linear feedback function, and the parameters of the feedback function can be expressed as functions of the parameters of the Q-function. This is a property we want to keep for the neural Q-function.

We will propose a method that uses LQRQL to keep this property for a neural Q-function. In this method the feedback function can be expressed using the weights of the neural Q-function. No second network is required. The method will also guarantee that there are no discontinuities in the feedback function.

5.3 Neural LQRQL

Neural LQRQL\(^1\) is based on the standard LQRQL described in chapter 3, but then with a feed forward network to represent the Q-function. The standard LQRQL approach is based on three steps. First the least squares estimation is applied to the train set, resulting in an estimation \( \theta \). Then the parameters of the quadratic Q-function \( \hat{H} \) are derived from the estimated \( \theta \). Finally the value of linear feedback \( L \) is computed based on \( \hat{H} \) using (3.16). This indicates that if it is possible to derive \( \hat{\theta} \) from the neural Q-function, the feedback can be computed from \( \hat{\theta} \) analogue to the LQRQL approach.

In LQRQL the quadratic Q-function was represented as a linear multiplication of the quadratic combinations of the state and action with the estimated parameters. This was introduced to make it possible to use a linear least squares estimation for estimating \( \theta \)

---

\(^1\)An almost similar approach in [33] is called Pseudo-Parametric Q-Learning.
based on the measurements. In section 3.2.4 the Q-function is written as

$$Q^L(x_k, u_k) = \text{vec}^o(\phi_k \phi_k^T)^T \text{vec}^o(H^L) = \Omega_k^T \theta.$$  \hspace{1cm} (5.2)

Here $H^L$ represents the parameters of the quadratic Q-function and $\phi_k$ the vector with the state and control action. This can be regarded as just a multiplication of two vectors $\Omega_k$ and $\theta$. We see that writing the quadratic Q-function as a linear function requires that the input vector is "quadratic". Instead of having a quadratic function with input $\phi$, we have a linear function with input $\Omega$. Input $\Omega$ contains all quadratic combinations of elements of $\phi$ and vector $\theta$ contains the corresponding values from $H^L$.

The representation of the Q-function in (5.2) can also be viewed as a one layer feed-forward network with a linear transfer function and input $\Omega_k$. We can extend this representation by writing this similar to (5.1):\(^2\)

$$Q(x_k, u_k) = \Gamma_o(w_o^T \Omega_h^T + b_h) + b_o.$$ \hspace{1cm} (5.3)

If we take $\Gamma_o$ and $\Gamma_h$ as linear functions and the biases $b_h$ and $b_o$ all zero, then there exist values for $W$ and $w_o$ such that (5.3) equals (5.2). Note that these weights are not unique, because the number of weights is higher than the number of parameters of (5.3). Since we want to stay as close as possible to the original LQRQL approach, we will only consider activation functions that resemble linear functions.

The neural representation is introduced to deal with non quadratic Q-functions. The network in (5.3) with linear transfer functions can not represent these functions, therefore we have to use nonlinear transfer functions. Let the output of hidden unit $i$ is given by:

$$\Gamma_{h,i}(w_{h,i}^T \Omega_k^T + b_{h,i}) = \tanh(w_{h,i}^T \Omega_k^T + b_{h,i}).$$ \hspace{1cm} (5.4)

The hyperbolic tangent is nonlinear, but in the origin it is zero and its derivative is one. So for small values of $w_{h,i}$ (5.4) still resembles a unit with a linear transfer function. Only when the weights of the hidden units become large, the Q-function will no longer be a quadratic function.

### 5.3.1 Deriving the feedback function

Given the Q-function and its input, the values of $\hat{\theta}$ should be obtained. In the case of the standard LQRQL these are just the parameters that are estimated, so they are immediately available. For the neural Q-function this is different. The weights are the parameters that are estimated based on the train set, so the value of $\hat{\theta}$ should be derived from the weights.

Given $Q^L(\Omega_k)$ from (5.2), the parameters $\theta$ can also be obtained by computing the derivative of this function to the input $\Omega$. This means that parameter $\theta_i$ can be computed according to:

$$\theta_i = \frac{\partial Q^L(\Omega_k)}{\partial \Omega_i}.$$ \hspace{1cm} (5.5)

\(^2\)This Q function still depends on the feedback function used to generate the data, which can also be a nonlinear function. Therefore we will omit the superscript $L$.\]
This shows that the $\theta_i$ does not depend on $\Omega_k$, so the values of $\theta_i$ do not depend on the state and control action. When this is computed for all parameters of $\theta$, then $H^L$ can be derived and therefore $\hat{L}$ can be computed.

In the same way $\hat{\theta}_i(\Omega)^3$ can be computed for the neural Q-function in (5.3):

$$\hat{\theta}_i(\Omega_k) = \frac{\partial Q(\Omega_k)}{\partial \Omega_i} = \sum_{j=1}^{n_h} w_{o,j} w_{h,i,j}(1 - \tanh^2(w_{h,i}^T \Omega_k + b_{h,i}))$$  \hspace{1cm} (5.6)

where $w_{h,i,j}$ represents the weight from input $\Omega_i$ to unit $j$ and $n_h$ indicates the number of hidden units.

When $\hat{\theta}(\Omega)_k$ is available, it can be rearranged into $\hat{H}(\Omega)$ so that $L(\Omega)$ can be computed. The control action can be computed according to $u_k = L(\Omega_k)x_k$. This is still a linear multiplication of the state with $L(\Omega_k)$, but it is a nonlinear feedback function because $\Omega_k$ contains $x_k$. The problem is that it also contains $u_k$, which is the control action that has to be computed.

In order to solve this problem (5.6) can be split into two parts:

$$\hat{\theta}_i = \sum_{j=1}^{n_h} w_{o,j} w_{h,i,j} - \sum_{j=1}^{n_h} w_{o,j} w_{h,i,j} \tanh^2(w_{h,i}^T \Omega_k + b_{h,i}).$$  \hspace{1cm} (5.7)

The first part is indicated with “linear”, because this corresponds to the $\hat{\theta}_i$ that would be computed when all hidden units are linear. The resulting feedback function derived from this network would also be linear. The second part is indicated with “nonlinear” because this is the effect of the nonlinearities introduced by the hyperbolic tangent transfer functions.

A linear feedback can be obtained from the network by just ignoring the hyperbolic tangent functions. Define $\tilde{\theta}$ by:

$$\tilde{\theta}_i = \sum_{j=1}^{n_h} w_{o,j} w_{h,i,j}.$$  \hspace{1cm} (5.8)

This leads to a vector $\tilde{\theta}$. Analog to $\tilde{\theta}$ we can rearrange $\hat{\theta}$ to form a quadratic function with parameters $\hat{H}$. From $\hat{H}$ we can derive a linear feedback $\hat{L}$.

The feedback $\hat{L}$ can be used to compute the control action $\hat{u} = \hat{L}x$. This control action can be used to control the system, but it can also be used to obtain the vector $\hat{\Omega}$. The vector $\hat{\Omega}$ represents the quadratic combination of the state $x$ with the control action $\hat{u}$. This can be used to obtain the nonlinear feedback

$$u_k = (H_{uu}(\hat{\Omega}_k))^{-1} H_{ux}(\hat{\Omega}_k)x_k = L(\hat{\Omega}_k)x_k.$$  \hspace{1cm} (5.9)

Here the linear feedback $L(\hat{\Omega}_k)$ is a function of $x_k$, so that the resulting feedback function is nonlinear. If we compute this feedback function for the Q-function in figure 5.2(a) then this results in the feedback function shown in figure 5.4.

---

3Strictly speaking, the $\hat{\theta}_i(\Omega)$ is not an estimation. For consistency with the previous chapters we will keep indicating it with a hat.
5.3. NEURAL LQRQL

5.3.2 Discussion

If we want to interpret the resulting feedback function, we have to look at (5.6). In case $w_{h,i}^T\Omega_k + b_{h,i}$ is large for a hidden unit $i$, the $(1 - \tanh^2(w_{h,i}^T\Omega_k + b_{h,i}))$ will be very small. The contribution of this hidden unit to $\hat{\theta}_i$ will be very small. In case $w_{h,i}^T\Omega_k + b_{h,i}$ is zero for a hidden unit $i$, the $\tanh(w_{h,i}^T\Omega_k + b_{h,i})$ will be very small. The hyperbolic tangent can be ignored and the contribution of this hidden unit is the same as for the computation of the linear feedback $\hat{L}$. We therefore can interpret the feedback function as a locally weighted combination of linear feedback functions.

Each hidden unit will lead to a linear feedback function and the value of the state $x$ determines how much the feedback function of each state contributes to the total feedback. When we compute $\hat{L}$ all hidden units are weighted equally. This results in a linear feedback function that is globally valid, because it no longer depends on the state value. When the neural approach is applied to the LQR task, then the resulting feedback should not depend on the state value. The weights of the hidden units should be very small, so that the feedback function should become linear.

The linear feedback $\hat{L}$ does not have to be the same as the result $\hat{L}_{QL}$ of LQRQL, when applied to the same data set. In standard LQRQL the parameters of a quadratic Q-function are estimated. When the Q-function to approximate is not quadratic, then a small part of the train set can have a large influence on the estimated parameters. This is the case for training samples obtained in parts of the state space where the Q-function deviates a lot from a quadratic function. If (5.3) is used to approximate the Q-function, the influence of these points is much less. One hidden unit can be used to get a better approximation of the true Q-function for that part of the state space. Since $\hat{L}$ is based on

Figure 5.4. Nonlinear feedback example. The feed forward network forms the same Q-function as in figure 5.2(a). This figure shows the resulting linear feedback $\hat{L}$ and the nonlinear feedback function according to (5.9). For comparison also the greedy feedback from figure 5.2(b) is plotted.
the average linear feedback formed by the hidden units, the contribution of the training samples on the resulting linear feedback \( \hat{L} \) is much smaller. This means that if the true Q-function is not quadratic, the linear feedback derived from the network is more reliable than the result of the standard LQRQL approach.

The linear feedback \( \hat{L} \) can always be derived from the network. This does not hold for the nonlinear \( L(\bar{\Omega}_k) \), because it depends on the weighting of the linear feedbacks based on the state value. It might be possible that for some state values, the contribution of all hidden units to \( \bar{\Theta} \) are zero. Then no feedback can be computed. This can only occur when all weights of the hidden units are large. This means that it is important to prevent the weights from becoming too large. One way of doing this is by incorporating a form of regularization in the learning rule. Another way is to scale down the reinforcements \( r \), so that the Q-function to approximate becomes smoother. The easiest way is to make sure that the number of hidden units is not too large, so that it becomes very unlikely that the weights become too large.

5.4 Training the Network

The weights of the network in (5.3) can be found by training the network according to one of the methods described in section 2.3. If the method that minimizes the quadratic temporal difference error (2.24) using (2.25) and (2.26) with a discount factor \( \gamma = 1 \) is used, then the training is based on minimizing the same criterion as (3.25). The only difference is that the network starts with some random initial weights that are incrementally updated at each iteration step. The least squares estimation gives the global minimum of the error at once. A consequence of this difference is that the weights of the network always have a value, while the least squares estimation gives no solution in case of a singularity. On the other hand, the training of the network can fail to find the global minimum of the error.

Because the network approximates a Q-function (2.24) is written using \( Q(\Omega, w) \):\(^4\)

\[
E = \frac{1}{2} \sum_{k=0}^{N-1} (r_k + Q(\Omega_{k+1}, w) - Q(\Omega_k, w))^2. \tag{5.10}
\]

and (2.26) becomes

\[
\Delta w_k = (r_k + Q(\Omega_{k+1}, w) - Q(\Omega_k, w)) (\nabla_w Q(\Omega_{k+1}, w) - \nabla_w Q(\Omega_k, w)). \tag{5.11}
\]

Before the network can be trained, the initial weights have to be selected. If the weights of the hidden units are chosen very small, the feedback function derived from this initial network is linear. If the true Q-function is quadratic, the weights change such that the feedback remains linear. Only when it is necessary, when the true Q-function is not quadratic, the weights of the hidden units become larger. Then the resulting feedback will be linear. In this way it is prevented that the resulting feedback becomes very nonlinear, while the simpler linear feedback is also possible.

\(^4\)When the discount factor \( \gamma = 1 \).
There are some differences between the estimation of the parameters of a quadratic function and the training of the network. The important differences are:

- **No singularity**
  For too low exploration no feedback can be computed for the standard approach. The neural approach already starts with an initialized network and the network only changes its parameters. This means that there is no singularity for too low exploration. Singularity can only happen when the weights of the hidden units are too large so that $\hat{\theta}$ is almost zero.

- **Different results for low exploration**
  The training of the network is such that the temporal difference is minimized. Even if the global minimum is found this does not necessary result in an improved feedback function. If insufficient exploration is used then LQRQL results in the feedback used to generate the train set. It is very unlikely that the feedback derived from the network will be the same as the feedback used to generate the train set. Even when the same train set is used, the resulting feedbacks will not be the same when the networks are initialized differently.

- **Improvement for high exploration**
  Sufficient exploration is achieved, if the exploration contributes to the temporal difference error. Only then, the minimization of the temporal difference error will lead to an improvement of the feedback function. The exploration will make the temporal difference error much higher. This means that a network, trained on a set with insufficient exploration, will have a lower error than one trained on a set with sufficient exploration. For a train set with sufficient data, different networks initialized with small weights will eventually result in similar feedback functions.

### 5.5 Simulation Experiments with a Nonlinear System

In chapter 4 we used a mobile robot to experiment with the different approaches. We used the same system to test the Neural Q-Learning approach. The main difference with chapter 4 is that the resulting feedback function should be globally valid. This means that we test the feedback function in a larger part of the state space. In order to compare the results with a different nonlinear feedback, we also used Gain Scheduling based on the Extended Q-Learning approach.

#### 5.5.1 Introduction

The system was identical to the one in chapter 4, we only have to specify the settings for the two approaches. To be able to compare the results both approaches used the same train sets. This means that the train set was generated with one linear feedback $L = [10^{-3} \ 10^{-3}]$. 
Gain Scheduling

For the Gain Scheduling approach we had to partition the state space in separate partitions. For each partition one local linear feedback was obtained by using extended LQRQL. For each estimation a train set was selected consisting of all training samples in that particular partition. This means that we had to make sure that for each partition sufficient training samples were available to obtain a local feedback. We did this by generating more train sets by starting the system from different initial states.

In chapter 4 we observed that different feedbacks were found when train sets were generated with a different initial $\delta$. This implies that the feedback should be different for different values of $\delta$. We therefore divide the state space into three partitions according to:

- Negative partition: $-\infty < \delta < -1$
- Middle partition: $-1 < \delta < 1$
- Positive partition: $1 < \delta < \infty$

The resulting feedback function consists of three local linear functions. At the border of the partitions these functions can be different, so that the feedback function is not continuous. It is possible to make this a continuous differential function by smoothing it near the borders of the partitions. We did not do this because we wanted to see the local linear feedbacks as clearly as possible for the comparison with the Neural Q-Learning approach.

Neural Q-Learning

For the Neural Q-Learning approach, all train sets used for Gain Scheduling were combined into one train set. This train set was used to train the network, by minimizing the quadratic temporal difference error using (5.10). Because the result also depends on the initial weights, we trained the network for ten different initial weights. Then we used the network with the lowest quadratic temporal difference error. To prevent over fitting it is possible to split the train set in two and use only one set for training and the other for testing. We did not do this because we wanted the resulting feedback to be based on the same train set as the Gain Scheduling approach. Instead we made sure that we did not have too many hidden units.

We chose the following settings for the network:

- Number of hidden units: 3.
  This was to prevent over fitting and to keep the number of weights close to the number of parameters of the Gain Scheduling approach.

- The value of the initial output weights: 0.

- Values of initial weights and biases of hidden units: Random between $-10^{-4}$ and $10^{-4}$.
5.5. SIMULATION EXPERIMENTS WITH A NONLINEAR SYSTEM

These weights are taken small so that the initial resulting feedbacks are linear and perhaps become nonlinear during training.

5.5.2 The experimental procedure

The purpose of this experiment is to point out the major differences with the experiments in chapter 4. Different train sets are generated by starting the system for different initial $\delta$. For sequence of exploration the train sets are shown in figure 5.5.

All train sets are combined to form one train set that is used to obtain the two global

![Figure 5.5. The trajectories while generating the train set.](image1)

![Figure 5.6. The control action as function of the state](image2)

(a) Gain Scheduling (b) Neural Q-Learning
nonlinear feedback functions. The resulting feedback functions of the train sets in figure 5.5 are shown in figure 5.6. In figure 5.6(a) the Gain Scheduling result is shown. The feedback function consists of three local linear feedbacks, but in this case two linear feedbacks are quite similar. The third feedback is different and the "jump" between the boundary of the two partitions is very clear. In figure 5.6(b) the resulting feedback based on Neural Q-Learning is shown. This looks like one linear feedback with a smooth nonlinear correction.

The feedback functions were tested by starting the robot for different initial δ. In figure 5.7(a) four trajectories are shown for the Gain Scheduling result. The trajectory that starts in δ = 1.5 clearly shows a jump when it crosses the boundary between the partitions of the state space. After the jump the trajectory moves similar to the other trajectories in that partition towards the line.

In figure 5.7(b) the resulting trajectories for Neural Q-Learning are shown. It is clear that there are no jumps in the trajectories. Also it can be noticed that for large initial distances to the line, δ = -1.5 and δ = 1.5, the robot initially moves faster towards the line. This is a consequence of the nonlinear correction.

The main difference with the experiments in chapter 4 is that now two global nonlinear feedbacks are used. This means that the tests from different initial δ are performed with the same feedback function. In chapter 4, each initial δ was tested with the corresponding local linear feedback.

![Figure 5.7. The trajectories](image-url)
5.5.3 The performance of the global feedback functions

We did 30 simulation experiments to compare the performance of the two approaches. Each experiment was performed as described in the preliminary experiments. As initial $\delta$ we used: $-1.75$, $-0.75$, $0.75$ and $1.75$. For the Neural Q-Learning approach, the network was trained with 5 different initial weights. The network for which the lowest temporal difference error was reached was tested.

The tests were performed by running the robot for 302 time steps. This is equivalent to driving for approximately 2 minutes. The reason for using a long test period is that we wanted the robot to visit a large part of the state space.

In figure 5.8(a) we plotted the total costs as a function of the initial $\delta$ for the best feedbacks of the two approaches. We see that the values of the resulting total costs of the Neural Q-Learning approach are symmetric around $\delta = 0$. This is because of the quadratic input of the network. Also we see that it is lower than the feedback for the Gain Scheduling approach. The result of the Gain Scheduling approach is not symmetric, which indicates that the robot will not approach the line exactly. Instead it will drive at a small distance parallel to the line, which also explains why the total costs are a little higher than for the Neural Q-Learning approach.

In figure 5.8(b) we see the average total costs of both approaches for all 30 experiments. Note that we plotted the the average log$(J)$, because the value of $J$ varies between 10 and $10^5$. We see that the Gain Scheduling approach performs very badly on average. This is because the results are either good as shown in figure 5.8(a) or very bad. These very

Figure 5.8. The performances of the global feedback functions.
bad results are consequences of bad local feedbacks at the middle partition. If the local feedback around the line makes the robot move towards the outer partitions, the total costs will always be very high. If in this case the linear feedback in that outer partition is good, it will make the robot move towards the middle partition. As a consequence the robot gets stuck at the boundary of these two partitions.

The reason for the bad performance of Gain Scheduling is that the resulting feedback function of the extended LQRQL approach is completely determined by the train set. In chapter 4 we already showed that not all resulting feedback functions will be good. In Gain Scheduling the state space is partitioned and for each partition the train set should lead to a good function. The partitioning makes that it becomes less likely that all partitions will have a good performance. The Neural Q-Learning approach uses the complete train set. Therefore it does not have this problem, and performs much better on average.

5.5.4 Training with a larger train set

The previous experiments showed that Gain Scheduling did not perform so good. The only way to improve this result is to make sure that for each partition the train set is good. We did an experiment to see whether the result of Gain Scheduling can be improved by using a larger train set for each partition. We used all the train sets from the previous experiment and combined them into one large train set. Then we applied the Gain Scheduling and Neural Q-Learning approach to this larger train set.

In figure 5.9(a) we see the resulting nonlinear feedback for Gain Scheduling. For the middle partition $I = -0.0032$, which shows that it is getting closer to the correct value of $\frac{33}{2} \alpha$. The reason for the bad performance of Gain Scheduling is that the resulting feedback function of the extended LQRQL approach is completely determined by the train set. In chapter 4 we already showed that not all resulting feedback functions will be good. In Gain Scheduling the state space is partitioned and for each partition the train set should lead to a good function. The partitioning makes that it becomes less likely that all partitions will have a good performance. The Neural Q-Learning approach uses the complete train set. Therefore it does not have this problem, and performs much better on average.

**Figure 5.9.** The control action as function of the state when a larger train set was used.
5.5. SIMULATION EXPERIMENTS WITH A NONLINEAR SYSTEM

\[
\begin{array}{l|ccccc}
\delta & -1.75 & -0.75 & 0.75 & 1.75 \\
\hline
J_{GS} & 219.3274 & 105.8559 & 91.3872 & 201.2774 \\
J_{NQ} & 121.8163 & 16.8742 & 16.8742 & 121.8163 \\
\end{array}
\]

Table 5.1. The total costs for the feedback function based on the large train set.

\( l = 0 \). The value for \( \hat{l} \) is 0.8085 for the negative and -1.0222 for the positive partition. This agrees with the discussion in section 4.5.1 where we indicated that \( L > 0 \) if \( \delta < 0 \) and \( L < 0 \) if \( \delta > 0 \). In figure 5.9(b) we see the resulting nonlinear feedback for Neural Q-Learning. This feedback is steeper when \( \delta \approx 0 \), compared to the feedback in figure 5.6(b).

We tested the resulting feedback functions by starting the robot in the same initial states. Table 5.1 shows the resulting total costs for both approaches for different \( \delta_0 \). The total costs \( J_{GS} \) are much lower than the average value in figure 5.8(b). This shows that the increase of the train set has a huge impact on the performance of Gain Scheduling. The total costs of the neural Q-Learning approach is again lower that that of Gain Scheduling. The performance in figure 5.8(b) was already good based on smaller train sets, therefore the increase in train set does not have such a huge impact.

The trajectories of the test runs are shown in figure 5.10 and figure 5.11. In figure 5.10 we see that for the outer partitions the robot rotates \( \frac{1}{2} \pi \) in the direction of the line within one step. Then it arrives at a state where the control action is almost zero, so the robot moves straight ahead until it enters the middle partition. In the middle partition the robot will slowly approach the line. This is because the linear function of the middle partition in figure 5.9(a) is not steep enough. The reason is that for the middle partition, the feedback \( L \) hardly contributes to the control action, so that it is very difficult to estimate the quadratic Q-function. This implies that the total costs are higher than those of the neural Q-Learning approach, because of the feedback in the middle partition. The trajectories of the Neural Q-Learning approach in figure 5.11 shows that the robot moves faster to the line than the trajectories in figure 5.7(b). This is because the feedback in figure 5.9(b) is steeper near \( \delta \approx 0 \).

\[ \text{Figure 5.10. The trajectories for the Gain Scheduling feedback based on the large train set.} \]
Figure 5.11. The trajectories for the Neural Q-Learning feedback based on the large train set.

5.6 Experiment on a Real Nonlinear System

We did an experiment with the real robot, where we used the exploration noise sequence that gave the best results for both LQRQL approaches in chapter 4. The resulting feedback functions are shown in figure 5.12. In figure 5.12(a) it is clear that the local linear feedback for the partition, with the negative initial $\delta$, is wrong. This indicates that the exploration noise sequence used, only results in a useful training set for positive initial $\delta$. Since the resulting feedback in this partition is completely determined by the train set, the only way to solve this is by generating a new training set for this partition.

In figure 5.12(b) we see that the result of the Neural Q-Learning approach is again linear with a smooth nonlinear correction. Also we see that the control actions are large (we observed the same in chapter 4 for the SI approach). In fact, the linear feedback derived from the network is too large. We see that in some parts of the state space the control action is reduced by the nonlinear correction. At these states the control actions are between the action bounds of the real robot, so these actions can be applied.

Because of the space limitation of the room, we tested only for one minute. The resulting total costs are shown in Table 5.2. Again we see that the resulting total costs for the Gain Scheduling approach are not symmetric. Also we see that the costs for $\delta_0 = -1.75$ is very high. This corresponds to the partition for which the local linear feedback is wrong. The results of the Neural Q-Learning approach show that it performs very well for all initial $\delta$. In order to understand the results in Table 5.2, we plotted the trajectories. In (5.13(a))

$$
\begin{array}{|c|c|c|c|c|}
\hline
\delta & -1.75 & -0.75 & 0.75 & 1.75 \\
\hline
J_{GS} & 634.007 & 89.008 & 29.993 & 103.707 \\
J_{NQ} & 82.362 & 11.600 & 11.471 & 82.189 \\
\hline
\end{array}
$$

Table 5.2. The total costs for the real nonlinear system
5.6. EXPERIMENT ON A REAL NONLINEAR SYSTEM

we see that for the negative partition the robot moves in the wrong direction. For this partition the local feedback is wrong. For the other partition the robot moves toward a line parallel to the line to follow. This indicates that it will never follow the line exactly. This is a consequence of the local feedback at the middle partition.

In (5.13(b)) we see that the two trajectories that start close to the line will approach the line. We also see that the trajectories far from the line first start to rotate very fast towards the line. The first few time steps the action is higher than the safety limit, so we only let the robot rotate with maximum speed. After that all actions are within the safety limit and the robot starts following the line. The high linear feedback from the network makes the robot move to the line very efficiently, when it is close to the line. The linear feedback will result in control actions that are too large far from the line. Due to the nonlinear correction the size of these actions are reduced. In figure 5.12(b) we see that this correction is for state values that were part of the train set. For other parts of the state space the control actions are still too large.

These experiments have shown that the Gain Scheduling approach depends very much on the training sets for each of the local linear feedbacks. It is possible that for some parts of the state space the feedback function found is not appropriate. The Neural Q-Learning approach results in a feedback function that is linear in most parts of the state space. In parts of the state space where the training set indicates that a correction is required, the control action deviates from the linear feedback. The resulting feedback function of Neural Q-Learning is a smooth function, unlike the feedback function of the Gain Scheduling approach where there can still be discontinuities.
5.7 Discussion

In this chapter we proposed a method to apply Q-Learning for general Q-functions. We described how a feed-forward network can be used to approximate the Q-function. The result is that the resulting feedbacks can also be nonlinear. We showed how the LQRQL approach can be used to obtain the feedback from the approximated Q-function. Then only one function has to be approximated, the feedback follows directly from the Q-function.

The advantage of using only one network is that only one network has to be trained. This makes this approach more convenient to use. More important is that the feedback function directly follows from the approximated Q-function. This is similar to Q-learning where the greedy policy is obtained by selection the greedy action for each state.
A second advantage of our approach is that the resulting feedback function is smooth. This means that there are no sudden jumps as in Gain Scheduling. These jumps can also appear when the greedy action is directly computed from the function approximator. The problem with these jumps is that the behavior of the system becomes more unpredictable in the presence of noise. The effect of the jumps can be overcome, but for Neural Q-Learning this is not necessary.

Both the Gain Scheduling approach and the Neural Q-Learning approach derive a global nonlinear feedback function from a set of linear feedback functions. In Gain Scheduling only one local linear feedback determines the control action, so it is essential that for every partition the feedback function is good. This is not always the case, resulting sometimes in very poor performances. The only way to solve this is to generate more data.

The feedback function of the Neural Q-Learning approach can be seen as a locally weighted combination of linear feedback functions. Therefore the result is less dependent on the correctness of each linear feedback function. This means that a poor performance as that of Gain Scheduling becomes very unlikely. Because the training is based on the complete train set, it will result in a good feedback function for a smaller train set than the Gain Scheduling approach.

5.8 Conclusions

There are different ways to get a feedback when using a feed forward network as a Q-function. Our method is based on the idea that there should be a direct relation between the Q-function and the feedback. The method uses LQRQL to obtain a linear feedback from the Q-function. Then the linear feedback is used to compute the nonlinear feedback function. The result is a global linear function with local nonlinear corrections.