Methods for Supervised Learning of Tasks

Alper Denasi
DCT 2008.043

Traineeship report

Coaching:

Prof. dr. H. Nijmeijer
Technische Universiteit Eindhoven
Department Mechanical Engineering
Dynamics and Control Technology Group

Ir. B.T. Verhaar
Philips Applied Technologies

Eindhoven, March, 2008
Abstract

Robots equipped with artificial intelligence (AI) capabilities are entering our domestic environments. Manually designing and programming a controller and a trajectory generator for a robot performing a single task is doable for some tasks. For a robot that will perform many tasks, however, this will be too time-consuming. This makes imitating the demonstrations given by a supervisor, an appealing approach. The objective of this traineeship is to generate knowledge about imitation learning methods and to demonstrate how one of these methods, namely supervised learning, works on swinging up and balancing an inverted pendulum on a cart.
Acknowledgements

I would like to thank the following people. First of all I would like to thank my Philips supervisor Boudewijn Verhaar for his help for any given problem. I would also like to thank professor Robert Babuska for the discussions that we had about learning methods. I owe thanks to Rob Tousain and Thom Warmerdam for the discussions about imitation learning. Furthermore, I thank my professor Henk Nijmeijer for supervising me.
Table of Contents

ABSTRACT ....................................................................................................................................................... 1
ACKNOWLEDGEMENTS........................................................................................................................................ 2
TABLE OF CONTENTS......................................................................................................................................... 3
LIST OF FIGURES .............................................................................................................................................. 4
LIST OF TABLES ................................................................................................................................................ 5
1. INTRODUCTION ........................................................................................................................................... 6
   1.1  MOTIVATION ........................................................................................................................................... 6
   1.2  PROBLEM STATEMENT ......................................................................................................................... 6
   1.3  SPECIFICATIONS OF AN ImitATION LEARNING ALGORITHM ......................................................... 6
   1.4  OUTLINE .............................................................................................................................................. 7
2. LEARNING METHODS .................................................................................................................................... 8
   2.1  SUPERVISED LEARNING ......................................................................................................................... 8
   2.2  REINFORCEMENT LEARNING (RL) ...................................................................................................... 9
      2.2.1  Temporal-Difference (TD) Learning .................................................................................................. 10
      2.2.2  Actor-Critic Architecture .............................................................................................................. 10
   2.3  SUPERVISED ACTOR-CRITIC REINFORCEMENT LEARNING ......................................................... 11
   2.4  MOVEMENT PRIMITIVES .................................................................................................................... 11
3. FUNCTION APPROXIMATORS ................................................................................................................... 13
   3.1  ARTIFICIAL NEURAL NETWORKS ....................................................................................................... 13
      3.1.1  Multilayer Feedforward Neural Networks ...................................................................................... 14
      3.1.2  Radial Basis Function Networks .................................................................................................. 15
      3.1.3  Recurrent Neural Networks ......................................................................................................... 15
      3.1.4  Echo State Networks ................................................................................................................... 16
   3.2  MEMORY BASED LEARNING ............................................................................................................... 16
4. INVERTED PENDULUM CONTROL .......................................................................................................... 17
   4.1  CONTROL FOR INFINITE TRACK LENGTH ........................................................................................ 18
   4.2  CONTROL FOR FINITE TRACK LENGTH ............................................................................................ 20
5. NEURAL NETWORK CONTROLLER DESIGN .......................................................................................... 23
   5.1  NEURAL CONTROLLER FOR THE FIRST TYPE OF TEACHER .......................................................... 24
   5.2  NEURAL CONTROLLER FOR THE SECOND TYPE OF TEACHER .................................................... 34
6. CONCLUSIONS AND RECOMMENDATIONS .......................................................................................... 43
   6.1  CONCLUSIONS ..................................................................................................................................... 43
   6.2  RECOMMENDATIONS ......................................................................................................................... 43
A. APPENDIX ..................................................................................................................................................... 44
   A.1  EQUATIONS OF MOTION ...................................................................................................................... 44
   A.2  SIMULATION RESULTS ......................................................................................................................... 46
      A.2.1  Results with the First Type of Teacher ............................................................................................. 46
      A.2.2  Results with the Second Type of Teacher ...................................................................................... 50
   A.3  M-FILES .............................................................................................................................................. 56
   A.4  SIMULINK SCHEMES .......................................................................................................................... 67
B. REFERENCES .................................................................................................................................................. 73
List of Figures

Figure 2.1  Agent – environment interaction................................................................. 9
Figure 2.2  Actor-critic schematic .............................................................................. 11
Figure 3.1  Structure of a single neuron of an ANN ....................................................... 13
Figure 3.2  Different types of activation functions ....................................................... 14
Figure 3.3  The structure of a MLFF neural network ................................................... 14
Figure 3.4  Structure of an RBF network ................................................................. 15
Figure 3.5  Structure of a fully connected RNN with hidden neurons ....................... 16
Figure 4.1  Graphical representation of an inverted pendulum system ....................... 17
Figure 4.2  Simulation results with controller for infinite track length for $x_0 = [\pi 0 0 0]$... 19
Figure 4.4  Simulation results with controller for finite track length for $x_0 = [\pi 0 0 0]$ .... 22
Figure 5.1  Teaching phase for supervised learning..................................................... 23
Figure 5.2  Execution phase for supervised learning..................................................... 23
Figure 5.3  Pendulum angles for the demonstrations for the 1st experiment ............... 25
Figure 5.4  Training results of the neural network ...................................................... 26
Figure 5.5  Actuator forces from demonstrations and its NN approximation ............... 26
Figure 5.6  Actuator forces from 3rd demonstration and its NN approximation ............ 27
Figure 5.7  I/O map of 1st teacher for 1st experiment with demonstrated trajectories ... 27
Figure 5.8  I/O map of NN for the 1st experiment ....................................................... 28
Figure 5.9  Successful trajectories obtained with NN for the 1st experiment .......... 29
Figure 5.10 Several pendulum angle plots from execution phase ............................. 29
Figure 5.11 I/O map of 1st teacher for 2nd experiment with demonstrated trajectories ... 30
Figure 5.12 I/O map of NN for the 2nd experiment .................................................... 30
Figure 5.13 Several pendulum angle plots from execution phase ............................. 31
Figure 5.14 Training results of the neural network ..................................................... 32
Figure 5.15 I/O map of 1st teacher for 3rd experiment with demonstrated trajectories ... 32
Figure 5.16 I/O map of NN for the 3rd experiment .................................................... 33
Figure 5.17 Several pendulum angle plots from execution phase ............................. 33
Figure 5.18 Pendulum angles and cart positions for the demonstrations with 2nd teacher ... 35
Figure 5.19 Training results of the neural network ..................................................... 35
Figure 5.20 Actuator forces from demonstrations and its NN approximation ............... 36
Figure 5.21 I/O map of 2nd teacher with demonstrated trajectories ......................... 37
Figure 5.22 I/O map of NN controller ................................................................. 37
Figure 5.23 Pendulum angles and cart positions for the demonstrations with 2nd teacher ... 38
Figure 5.24 Training results of the neural network ..................................................... 39
Figure 5.25 Actuator forces from demonstrations and its NN approximation ............... 39
Figure 5.26 Successful trajectories obtained with NN ............................................. 40
Figure 5.27 Several pendulum angle and cart position plots from execution phase .... 41
Figure A.1.1  Free body diagrams of cart and pendulum ........................................ 44
Figure A.2.1  Pendulum angles for the demonstrations for the 2nd experiment .......... 46
Figure A.2.2  Training results of the neural network .................................................. 47
Figure A.2.3  Actuator forces from demonstrations and its NN approximation ............. 47
Figure A.2.4  Actuator forces from 3rd demonstration and its NN approximation ......... 48
Figure A.2.5  Pendulum angles for the demonstrations for the 3rd experiment .............. 48
Figure A.2.6  Actuator forces from demonstrations and its NN approximation ......................49
Figure A.2.7  Actuator forces from 2nd demonstration and its NN approximation ...............49
Figure A.2.8  Actuator forces from 1st demonstration and its NN approximation ...............50
Figure A.2.9  I/O map of 2nd teacher with demonstrated trajectories.................................51
Figure A.2.10 I/O map of NN controller .............................................................................51
Figure A.2.11 Pendulum angles and cart positions for the demonstrations with 2nd teacher ..52
Figure A.2.12 Training results of the neural network ............................................................53
Figure A.2.13 Actuator forces from demonstrations and its NN approximation ...............53
Figure A.2.14 Actuator forces from 1st demonstration and its NN approximation ...............54
Figure A.2.15 I/O map of 2nd teacher with demonstrated trajectories...............................54
Figure A.2.16 I/O map of NN controller ...............................................................................55
Figure A.4.1  Simulink model for 1st type of teacher ............................................................67
Figure A.4.2  Plant model for 1st type of teacher .................................................................67
Figure A.4.3  Simulink model for execution phase of 1st type of teacher .........................68
Figure A.4.4  Simulink model for 2nd type of teacher .........................................................69
Figure A.4.5  Plant model for 2nd type of teacher ...............................................................69
Figure A.4.6  Simulink model for execution phase of 2nd type of teacher .......................71
Figure A.4.7  Contents of Neural Network Controllers .......................................................71
Figure A.4.8  1st layer of Neural Network Controllers .........................................................72
Figure A.4.9  2nd layer of Neural Network Controllers .........................................................72

List of Tables

Table 4.1  Table of Parameters.............................................................................................18
Table 5.1  Table of experiments and NN specifications for 1st type of teacher ..................34
Table 5.3  Table of experiments and NN specifications for 2nd type of teacher ...............42
1. Introduction

1.1 Motivation

Robots equipped with artificial intelligence (AI) capabilities are entering our domestic environments. Examples of such robots are vacuum cleaning robots, and entertainment robots. This will continue more rapidly in the following years. Even though these tasks might seem a routine, they are performed successfully by humans in different and also non-stationary environments and there are a variety of them. Manually designing and programming a controller for a robot might seem reasonable only for a single task in reasonably well known conditions and environment. Nevertheless, using the same approach for a robot that would perform many tasks could be extremely time consuming. Furthermore, domestic robots with robot arms could be sufficiently generic for tasks defined by end-users. This makes imitating the demonstrations, often called teleoperations, given by a supervisor, an appealing approach.

1.2 Problem Statement

The problem can be defined as below:

The objective of the assignment is to generate knowledge about imitation learning. A motion task such as balancing an inverted pendulum on a cart is demonstrated by a teacher. The teacher can be a (nonlinear) controller, or a person doing the task. Of the examples demonstrated by the teacher, sensor values and actuator forces are recorded. From these examples, a learning algorithm should distil the required behavior to complete the task. Once the algorithm can imitate the task sufficiently well within a reasonable number of imitations, a possible next step could be to demonstrate more complex tasks.

In order to present how this method works, swinging up and balancing an inverted pendulum on a cart is selected as an example system for this internship.

1.3 Specifications of an Imitation Learning Algorithm

An imitation learning method to be developed to solve the problem defined before is subject to some specifications. At the time of writing, these specifications can be regarded as a “wish list”. They are not yet quantitative. It is the purpose of this study and following studies to gain quantitative information about the specifications.

- The imitation learning algorithm (ILA) should be able to generate a policy that will accomplish a task for a certain system (plant or process), given a certain number of examples. In the case of a robotic arm and hand, demonstrations (i.e. teaching) could be performed by the end user holding a handle that is rigidly connected to the robot’s wrist.
- The required number of examples should not be too large, depending on the task.
- The algorithm should be sufficiently generic. So it should be possible to teach a robot a variety of tasks that could be repetitive or non-repetitive, with or without a clear goal “state”.

The ILA should be able to generalize between examples, and distill the “invariant”, even if the demonstrations are not exactly the same as the situations encountered during imitation.

A control policy is learned by this ILA, so it is more than just replaying a certain sequence of forces or setpoints which is the case in some welding or painting robots that just re-play a trajectory presented to them.

Minimal task specific “keyboard input” is required for the teacher. It should be avoided for the end-user to define a task specific reward function, set point trajectory or state variable, etc.

The required input from a computer vision system (i.e. vision feedback) should be minimized (we aim at tasks that can be accomplished (mostly) blindly). In case vision input is required for the task, it should be specified what exactly is needed.

Another important issue is robustness to disturbances. For example, if during the demonstration of a task, certain disturbances were encountered and were acted upon by the teacher, they should also be dealt with during the execution of the task.

Time invariance: If during execution of a task (e.g. wiping a table), the end-effector is stopped for a certain amount of time, after releasing it should continue where it stopped, unless it has been demonstrated to act differently after a pause.

Robustness to “hidden states” such as parameter variations or having to memorize what has already been done is desired, for example, if demonstrations of a certain task have been given with different parameters (e.g. stirring in a pan, demonstrations with a big pan or a small pan, or demonstrations with a pan filled with a thin layer of water, or full with honey), the generated policy should be able to handle these different parameters. An example for a task with memory is, if a task consists of executing subtasks A, B and C, in the order ABABAC, then the ILA should know that after doing A the third time, C should be executed instead of B. This means that the ILA should memorize how many times A has been executed.

**1.4 Outline**

The second chapter of this master traineeship report discusses several learning methods that are used in the field of artificial intelligence and machine learning. The third chapter introduces some of the function approximators which are used in these learning methods. The supervising controllers that are used as the demonstrators are presented in fourth chapter. An example of supervised learning by imitation is presented in the fifth chapter. The conclusions and recommendations are given in the last chapter.
2. **Learning Methods**

2.1 **Supervised Learning**

Supervised learning is a technique, in which a learning agent is provided with a set of training examples to learn what output it should generate with what input. In our case, the learning agent is used to control a plant (the inverted pendulum in this case). The agent is provided with examples (demonstrations) on how to perform a task from a supervisor (teacher). The supervisor can either be a controller that is already in use to control a plant, or a person who is demonstrating a task. For our objective, the supervisor will be a person demonstrating a task. Nevertheless, since the demonstrations in this report are performed by means of simulations, for practical reasons controllers are designed for the task and demonstrations are generated by using these designed controllers. It should be kept in mind that a controller’s policy might be much more consistent than a person’s policy (which might vary both in time and between demonstrations). Furthermore smoothness of executed trajectories has been an issue with the human master. Since both cases can be considered as an input-output mapping, the learning agent can be provided with the inputs and outputs recorded from the examples and it can try to approximate this behavior from the given data. This method might more formally be explained as, for teacher supplying the desired (target) output $y$ for the inputs $x_i$, where $i \in \{1, 2, \ldots, n\}$, the learner forms an input-output mapping $\hat{y} = F(x)$ which has to minimize the error,

$$
E = \sum \| \hat{y} - F(x) \| \tag{2.1}
$$

There are different types of learning methods such as artificial neural networks or memory-based learning methods. The advantages of this approach can be considered as follows,

- Tasks do not need to be preprogrammed by “keyboard”, they can be taught.
- An evaluative feedback such as defining a reward function, which is very difficult for real-life tasks is not required.
- The method is generic since it is a function approximation problem and with this method it is possible to teach the robot a variety of tasks.
- The learning method can generalize between examples, and distill the "invariant", even if there are slight differences between demonstrations.
- The learning method has the potential to be robust against disturbances, i.e. disturbances encountered during the demonstrations and acted upon by the teacher. These are present in the training set and can implicitly be present in the resulting policy.

The drawbacks of this approach can be considered as below,

- The required number of examples might increase if the number of inputs to the learning agent increases. It is expected that the required number of examples will also depend on the level of difficulty of the task, or more specifically to the ‘sensitivity’ of the task to the exact policy.
Balancing an inverted pendulum, for instance, is a difficult task since a slightly deviating policy might result in a non-stabilizing pendulum.

- The way to perform demonstrations for the learning algorithm might be difficult. A rigidly mounted handle could be connected to the wrist of a 7 DOF robotic arm. To record 6DOF forces, the handle should be mounted to the wrist via a 6DOF force sensor. For example, a complicated haptic interface might be used for a dexterous hand, whereas a wearable motion capturing device might be used for a 7 DOF robotic arm.

- The type and parameters of the learning method is not known beforehand. Yet a generic learning method and parameters have to be chosen that work for a large set of tasks. For example, for a certain learning method, multilayer feedforward neural networks, the necessary approximation error tolerance, the number of hidden layers and the number of neurons for a satisfactory task performance have to be selected.

- The learning phase only takes place during demonstration, due to no learning of any value or reward function. If new circumstances are encountered after demonstrations, the robot behavior is unpredictable.

2.2 Reinforcement Learning (RL)

Reinforcement learning is learning what to do by trial and error in order to maximize an evaluation signal, namely the reward [1]. At the beginning, the learning agent does not have any knowledge of the environment that it operates in, but it builds up its own experience through interaction. A challenging issue which is often encountered in the area of RL is the trade-off between exploration and exploitation. Since the objective of the learning agent is maximizing the accumulated reward, it must exploit what it learned from its past trials that produced rewards. However it must also make explorations to find better actions in the future. The accumulated reward or the return of rewards that received after time step $t$, can formally be given as,

$$R_t = r_{t+1} + r_{t+2} + r_{t+3} + \ldots + r_T$$

(2.2)

where $T$ is a final time step and $r_i$ is the reward at time instant $i$. An RL system basically consists of four elements, a policy, a reward function, a value function and optionally a model of the environment. The policy is a mapping from the states of the environment to actions to be taken when in those states, or formally, $\pi(s) \rightarrow a$, where $s \in S$ and $a \in A(s)$. The reward function, whose argument is a state-action pair, is a scalar number defining the goal of the RL problem. A state’s value function, $V(s)$ is the learning agent’s expectation of the accumulated reward in the future, beginning from that state. An action-value function $Q(s,a)$ is the learning agent’s expectation of the accumulated reward in the future starting from state $s$ and taking action $a$. The agent interacts with the environment as in the Figure 2.1.

![Figure 2.1 Agent – environment interaction](image-url)
There are different methods like Monte Carlo methods, dynamic programming, actor-critic and temporal difference learning methods. The most important advantage of this approach can be stated as below,

- No demonstration is necessary as in the case of supervised learning, since the algorithm learns a control policy from scratch by trial-and-error.

The drawbacks of this approach can be stated as follows,

- The number of trials and errors can be very high. This is mostly the case when the number of states (or dimension of state) increases. This phenomenon is called curse of dimensionality in RL literature.
- System behavior during learning can lead to unacceptable risks.
- A reward function should be defined by the user in a way that it should match the goal of the task. So it is difficult, or even impossible to come up with a general reward function for a variety of tasks.

2.2.1 Temporal-Difference (TD) Learning

Temporal difference learning, whose name also reveals itself, is a method developed by Sutton [1] and is related to updating the value function defined before, at each time step. This difference at time \( t \), can more formally be defined as below,

\[
\delta_t = r_{t+1} + \gamma V(s_{t+1}) - V(s_t)
\]

where \( \gamma \) is the discount factor to distinguish between continuous (e.g. robot control) and episodic tasks (e.g. a chess game) and from here the update equation is as follows [1],

\[
V(s_t) \leftarrow V(s_t) + \alpha \delta_t
\]

Here \( \alpha \) is a constant step size parameter. Among the most used TD learning methods is Q-learning, where instead of learning the state value function \( V(s) \), an action-value function \( Q(s, a) \) is learned.

2.2.2 Actor-Critic Architecture

Actor-critic architecture contains two distinct structures to represent the policy and the value function independently [1]. The actor contains the policy structure which selects the actions, whereas the critic structure contains the value function which is used to criticize the actions of the actor. Both the actor and critic use the TD-error for updating. The actor-critic architecture can be presented as below in Figure 2.2.
2.3 **Supervised Actor-Critic Reinforcement Learning**

This method is a combination of supervised learning and actor-critic architecture from reinforcement learning designed by Rosenstein and Barto [3]. Their method contains a modified actor, called the “composite actor”, which combines the actor from RL with the supervisor by the help of a gain scheduler. This combined learning method provides shifting from full supervision to full autonomy. The composite action can formally be represented by,

\[
    a \leftarrow k a^E + (1-k)a^S
\]

Here \( a^E \) denotes the actor’s exploratory action, and \( a^S \) is the supervisory action, which are determined by policies \( \pi^E \) and \( \pi^S \), and \( k \in [0,1] \) which is dependent on the state determines the level of autonomy of the composite action. The exploratory action is the same as the actor’s greedy action \( a^\pi \), following policy \( \pi^\pi \) except that it is altered by an additive zero-mean random variable. The actions produced by the learning algorithm in their paper form desired trajectories, since they assume that proper low-level controllers exist. The advantages of this approach can be given as follows,

- Demonstrations can be used to help the learning agent speed up the learning process.

The drawbacks of this approach can be given as below,

- Necessary training time for real robots might be high, although it is expected to be smaller than for ‘regular’ reinforcement learning.
- Supervisory input can prevent the progress made by actor-critic architecture.
- A reward function should be defined similar to the case of simple RL.

2.4 **Movement Primitives**

Movement (motor, motion) primitives are basic building blocks of set point generators, and they can be combined to form a desired movement, which then can be used by motor command generators (i.e. feedback controllers, feedforward controllers, etc.) [11]. When the
literature is searched, it would be noticed that there are different approaches in how to teach movement primitives. A common approach developed by Schaal et al. [10], is used to teach discrete (i.e. point to point movements) and rhythmic (i.e. circular movements as in table wiping) movement primitives by using dynamical systems.

The advantages of this method are,

- Motion primitives being basic building blocks, provide achieving more complex movements when combined appropriately, which is applied successfully to a table wiping case [9].
- Reaction to disturbances (obstacles or human interaction) can be included in an appropriate manner. For example, when the teacher interferes and holds the robot arm, trajectory generation stops and when the teacher leaves the robot arm, it continues from where it was left.
- Working in higher dimensional spaces as in the case of humanoid robotics might be easier. This is claimed in [11] that it introduces some sort of a constraint decreasing the higher dimensionality of state space. This needs to be further investigated.
- The properties of the trajectories (e.g. amplitude and frequency of a rhythmic motion primitive) can be changed online after the primitive is taught.

The disadvantages and unclear parts of this method are,

- The dynamic systems should often be designed specifically for a task at hand [8].
- The way to combine these movement primitives in order to achieve a complete task such as wiping a table, etc. is not clear.
- The ability of the method to cope with variations between demonstrations that would occur when a primitive is demonstrated by a human is unclear.
3. Function Approximators

A function or mapping is a relation or behavior between inputs and outputs, or more formally \( y = f(x) \), where \( y \) is the output and \( x \) is the input and \( f \) is the behavior. A function approximator is necessary when there is a set of input output data at hand, whereas there is no structure to represent this data available. When using supervised learning for control, this case is encountered. A human demonstrates a task to a robotic system, and it is possible to record sensor and actuator signals from this demonstration. Even though sensor and actuator signals are at hand, a behavior representing this data set is not, so a function approximator should be used in order to deal with this. Function approximators are also used in RL in predicting the state value, \( V(s) \) or the action-value functions, \( Q(s,a) \) since they are not known explicitly. In this chapter, some of the function approximators that are often used in artificial intelligence are introduced.

3.1 Artificial Neural Networks

An artificial neural network (ANN or a neural network, NN) is a parallelly distributed computing system, which imitates the structure of the brain [6]. It has the ability to store experiential knowledge which is obtained from its surrounding by a learning process, and making it available for use. The basic building blocks of ANNs are neurons and they are connected to each other by synaptic weights, which are altered during the learning process. Alteration of the synaptic weights is used to save the obtained knowledge. The structure of a single neuron can be represented as in Figure 3.1,

![Figure 3.1 Structure of a single neuron of an ANN](image)

A neuron basically consists of three elements, synaptic inputs, summation point and an activation function, which can also be observed from Figure 3.1. The synaptic inputs are characterized by their related weights \( w_{kj} \), where \( k \) denotes the number of neuron, the input connected and \( j = 1,\ldots,m \) denotes the input signal. The summation point creates a linear combination of the weighted inputs. The activation function is used to obtain an output (either linear or nonlinear) from the weighted inputs. The processing of a single neuron can more formally be presented as below,
\[ u_k = \sum_{j=1}^{m} w_{kj} x_j \]  
(3.1)

\[ y_k = \varphi(v_k) = \varphi (u_k + b_k) \]  
(3.2)

There are many different types of activation functions, \( \varphi(v_k) \) such as threshold function, piecewise linear function and sigmoid function. The graphs of some of these functions are given in the Figure 3.2,

![Activation Functions](image)

**Figure 3.2** Different types of activation functions

### 3.1.1 Multilayer Feedforward Neural Networks

A multilayer feedforward neural network consists of an input layer, one or more hidden layers and an output layer [6]. The input layer receives the input signals and these propagate forward through the network until the last layer to form an output. The training of the network is performed by supervision using error back propagation, which is based on error-correction learning rule. The structure of a MLFF network is shown in Figure 3.3.

![MLFF Network Diagram](image)

**Figure 3.3** The structure of a MLFF neural network
3.1.2 Radial Basis Function Networks

These kind of networks use a distance function such as a Gaussian function, often used, or an exponential function etc. as their activation functions. They only contain a single hidden layer, which differs from the MLFF networks, since they can have more than one hidden layer [2]. Training in radial basis function networks (RBF) starts with determining the centre and width of the Gaussian functions, and then the weights of the output layer are adjusted to approximate the desired output by error minimization, while keeping the Gaussian functions fixed. The output of RBF can be given by the formula below,

$$y(x) = \sum_{k=1}^{N} w_k \exp\left(-\frac{\|x-c_k\|^2}{\sigma_k^2}\right)$$  \hspace{1cm} (3.3)

where $w_k$ stands for the weights, $c_k$ for the center vectors and $\sigma_k$ for the widths of the $k^{th}$ neuron, and $x = [x_1, x_2, \ldots, x_m]$ is the input vector. The network structure can graphically be presented as follows in Figure 3.4.

![Figure 3.4](image)

Figure 3.4 Structure of an RBF network

3.1.3 Recurrent Neural Networks

Recurrent neural networks (RNN) are similar to feedforward neural networks except the fact that they contain feedback loops around their neurons [6]. The feedback loops also contain unit delay operators ($z^{-1}$), which introduce some sort of memory structure to the network. By this way, RNNs can approximate dynamic input output mappings (either linear or nonlinear). A fully connected recurrent neural network with hidden neurons can be presented as below in Figure 3.5.
3.1.4 Echo State Networks

Echo state networks (ESN) are a type of recurrent neural networks, where a recurrent neural network with fixed (i.e. not trained) and randomly initiated weights serving as a dynamical reservoir (i.e. each neuron produces a nonlinear response signal) [7]. The neurons in the reservoir are not fully connected to the inputs, whereas their linear combination forms the desired output response through training. Training of the ESNs is quicker compared to other multilayer networks since only the output weights are trained.

3.2 Memory Based Learning

Memory based learning is a simple function approximation method, in which all the data that is obtained from demonstrations is stored in a memory of correctly classified input-output examples [2]. An explicit training step for approximating the given input output behavior is not necessary as in the case of neural networks. An output is produced only when a query is made, which can slow down the method if too many data points are stored in the memory. The most popular memory-based learning methods are distance weighted nearest neighbors and locally weighted learning.
4. Inverted Pendulum Control

There are many methods in the literature that are used to control an inverted pendulum on a cart, most of which are energy based methods. The system is underactuated since there are two degrees of freedom which are the pendulum angle and the cart position but only the cart is actuated by a horizontal force acting on it. The equilibrium points of the inverted pendulum are when the pendulum is at its pending position (stable equilibrium point, i.e. \( x_{eq} = [0 \ 0 \ x \ 0] \), where \( x \) is any point on the track where the cart moves) and when the pendulum is at its upright position (unstable equilibrium point, i.e. \( x_{eq} = [\pi \ 0 \ x \ 0] \)). The objective is to swing up the pendulum from its pending position or from another state to the upright position and balance it in that condition. Two cases can be discerned. In one case, the cart must be kept within a finite track length, in another case the track length is infinite. The objective would be achieved by means of supervised learning. In practice, the teacher would be a human performing the task. Since a human teacher is not available in simulation, however, we have chosen for a nonlinear controller as a teacher. From the teacher, sensor values (i.e. pendulum angle and angular velocity, cart position and velocity) and actuator values (i.e. input force acting on the cart) can be recorded. Although the assumption of infinite track length is not realistic, it has the advantage that the corresponding controllers have only two inputs, namely the angle and the angular velocity of the pendulum. This enables us to compare the input-output mappings of the supervisor and neural controller using 2D contour plots. In this chapter, the inverted pendulum control problem will be introduced. Furthermore, two types of teachers that will serve as the supervisors to the imitating controllers will be presented.

The inverted pendulum system on a cart is presented in Figure 4.1. The pendulum’s mass “m” is assumed to be equally distributed on its body, so its center of gravity is at its middle of its length at “\( l = L/2 \)”. No damping or other kind of friction has been considered when deriving the model. The parameters of the model that are used in the simulations are below in Table 4.1.
<table>
<thead>
<tr>
<th>Parameters</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mass of pendulum, m</td>
<td>0.1 [kg]</td>
</tr>
<tr>
<td>Mass of cart, M</td>
<td>1 [kg]</td>
</tr>
<tr>
<td>Length of bar, L</td>
<td>1 [m]</td>
</tr>
<tr>
<td>Gravity, g</td>
<td>9.81 [m/s²]</td>
</tr>
<tr>
<td>Moment of inertia of bar w.r.t its COG</td>
<td>( I = mL^2/12 \approx 0.0083 ) [kg*m²]</td>
</tr>
</tbody>
</table>

Table 4.1 Table of Parameters

The equations of motion can either be derived by the Lagrangian method or by the free-body diagram method. The free-body diagram method is selected in this report. The details of the derivation of equations of motion can be found in Appendix A.1. The inverted pendulum system can be written in state space form by selecting the states and the input as,

\[
x_1 = \dot{\theta}, \quad x_2 = \ddot{\theta}, \quad x_3 = x, \quad x_4 = \dot{x}, \quad u = F
\]

So the state space equations of the inverted pendulum can be given as below,

\[
\begin{align*}
\dot{x}_1 &= x_2 \\
\dot{x}_2 &= \frac{(M + m)g l \sin x_1 - m^2 l^2 x_2^2 \sin x_1 \cos x_1 - ml \cos x_1 \cdot u}{(I + ml^2)(M + m) - m^2 l^2 \cos^2 x_1} \\
\dot{x}_3 &= x_4 \\
\dot{x}_4 &= -\frac{m^2 l^2 g \sin x_1 \cos x_1 + (I + ml^2)mlx_2^2 \sin x_1 + (I + ml^2)u}{(I + ml^2)(M + m) - m^2 l^2 \cos^2 x_1}
\end{align*}
\]

(4.1)

The state space equations can be linearised around \([\theta, \dot{\theta}, x, \dot{x}, u] = [0, 0, 0, 0] \)

\[
\begin{bmatrix}
\dot{\tilde{x}}_1 \\
\dot{\tilde{x}}_2 \\
\dot{\tilde{x}}_3 \\
\dot{\tilde{x}}_4
\end{bmatrix} =
\begin{bmatrix}
0 & (M + m)gl & 1 & 0 & 0 \\
(I + ml^2)(M + m) - m^2 l^2 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 \\
(I + ml^2)(M + m) - m^2 l^2 & 0 & 0 & 0 & 0
\end{bmatrix}
\begin{bmatrix}
\tilde{x}_1 \\
\tilde{x}_2 \\
\tilde{x}_3 \\
\tilde{x}_4
\end{bmatrix} +
\begin{bmatrix}
0 \\
-ml \\
0 \\
(I + ml^2)(M + m) - m^2 l^2
\end{bmatrix} \ddot{u}
\]

(4.2)

The eigenvalues of this linearised system with the parameter values given in Table 4.1, are \(\lambda_1 \approx -3.9739, \lambda_2 \approx 3.9739, \lambda_3 = 0, \lambda_4 = 0\). Since one of the eigenvalues \(\lambda_2 \approx 3.9739\) is positive (i.e. in the right half plane), the upright equilibrium point is unstable.

4.1 Control for Infinite Track Length

A simple controller in order to swing up the pendulum from its pendant position can be developed by means of physical reasoning. A constant force can be applied to the cart according to the direction of the angular velocity of the pendulum so that the pendulum would swing till the vicinity of the upright position. The balancing controller is a stabilizing linear controller which
can either be a PD, PID or be designed by pole placement techniques (e.g. Ackermann) or by LQR. The switching from the swinging controller to the linear controller is done when the pendulum angle is between \( \theta = [0^\circ, 20^\circ] \), or when \( \theta = [340^\circ, 360^\circ] \). This controller can formally be presented as below,

\[
\begin{align*}
    u = u(\theta, \dot{\theta}) &= \begin{cases} 
        K_1(0 - \theta) + K_2(0 - \dot{\theta}), & \text{if } 0 \leq |\theta| \leq \frac{\pi}{9} \\
        K_1(2\pi - \theta) + K_2(0 - \dot{\theta}), & \text{if } \frac{17\pi}{9} \leq |\theta| \leq 2\pi \\
        F_0 \text{ sgn}(\dot{\theta}), & \text{else}
    \end{cases}
\end{align*}
\] (4.3)

where \( F_0 = 2 \) Newton, \( K_1 = -40 \), \( K_2 = -10 \). The closed loop poles obtained with these gains are -10.6026, -4.0315, 0, and 0. The simulations are performed with Matlab Simulink by using a fixed step size of 0.1 seconds and ode5 (Dormand-Prince) fixed step solver for sufficient accuracy. The results of the simulations performed with this controller with the initial conditions \( x_0 = [\pi \ 0 \ 0 \ 0] \), without any kind of sensor noise or actuator disturbance are as follows,

![Simulation results with controller for infinite track length for \( x_0 = [\pi \ 0 \ 0 \ 0] \)](image-url)
It can be observed from the Figure 4.2 that the pendulum has been stabilized in the upright position, whereas the cart continues its travel with a constant velocity.

4.2 Control for Finite Track Length

A controller which can stabilize the inverted pendulum at the upright position by using energy based methods, which takes into account finite track length of the cart, is proposed by K. Yoshida [4]. A successful implementation of it is performed by M. Bugeja [5]. The method will be introduced very briefly here, more information can be obtained from the references. The total energy of the pendulum and its derivative w.r.t. time are,

\[ E = \frac{1}{2}(I + ml^2)\dot{\theta}^2 + mgl\cos\theta \]  \hspace{1cm} (4.4)

\[ \dot{E} = (I + ml^2)\ddot{\theta}\dot{\theta} - mgl\dot{\theta}\sin\theta \]  \hspace{1cm} (4.5)

By combining (A.1.12) and (4.22), the following expression can be obtained

\[ \dot{E} = -ml\ddot{\theta}\cos\theta \cdot \ddot{x} \]  \hspace{1cm} (4.6)

The energy can be controlled by modifying the sign of \( \ddot{x} \) and \( \dot{\theta}\cos\theta \) accordingly, which can be observed from (4.6). If \( \text{sgn}(\ddot{x}) = \text{sgn}(\dot{\theta}\cos\theta) \) then \( \dot{E} < 0 \), whereas if \( \text{sgn}(\ddot{x}) = -\text{sgn}(\dot{\theta}\cos\theta) \) then \( \dot{E} > 0 \). This shows that the cart acceleration should be controlled keeping in mind that the track that it travels is finite. The cart dynamics, i.e. the second row of (A.1.17), can be used to construct a feedback linearised second order servo system with a sinusoidal input that would provide the necessary back and forth movements around the cart’s home position. This sinusoidal input which is obtained from \( \dot{\theta}, \dot{\theta} \) generates \( \ddot{x} \) so that the desired energy \( E_{ref} \) can be reached.

The linearised servo system with \( x_{ref} \) as its reference input can be obtained by the control law below,

\[ F = \left[ (I + ml^2)(M + m) - m^2l^2\cos^2\theta \right] + \frac{m^2l^2g\sin\theta\cos\theta}{I + ml^2} - ml\dot{\theta}^2\sin\theta \]  \hspace{1cm} (4.7)

where \( v \) is the new control input, and this control law linearises the cart dynamics;

\[ \ddot{x} = v \]  \hspace{1cm} (4.8)

Selecting \( v = f_1(x_{ref} - x) - f_2\dot{x} \), where \( f_1 = \Omega^2 \), \( f_2 = 2\beta\Omega \), \( \Omega = \frac{\omega_n}{c_0} \), and \( \omega_n = \sqrt{\frac{mg\ell}{l + ml^2}} \), are all constants and \( c_0 \) and \( \beta \) are design parameters. The transfer function from \( x \) to \( x_{ref} \) becomes

\[ G(s) = \frac{\Omega^2}{s^2 + 2\beta\Omega s + \Omega^2} \]  \hspace{1cm} (4.9)
where $s$ is the Laplace variable. The sinusoidal reference input $x_{ref}$ is given as,

$$x_{ref}(t) = \frac{a}{G(\omega_n)} \sin(\varphi(t) - \pi + \phi(\omega_n))$$  \hspace{1cm} (4.10)

This is not a perfect sinusoid since $\varphi(t)$ does not vary linearly with time. Here $G(\omega_n)$ and $\phi(\omega_n)$ are the gain and phase lag of $G(j\omega)$ at $\omega = \omega_n$ and $\varphi(t)$ is an angle defined by the trajectory of the pendulum which can be given as follows,

$$\varphi(t) = -\arctan(\dot{\theta}/\omega_n, \theta - \pi)$$  \hspace{1cm} (4.11)

The parameter $a$ in (4.10) is used to determine the amplitude and sign of $x_{ref}$, and it is defined by,

$$a = \begin{cases} a_0 \text{sgn}(E - E_{ref}) & \text{if } |E - E_{ref}| \geq b_0 \\ a_0(E - E_{ref})/b_0 & \text{if } |E - E_{ref}| < b_0 \end{cases}$$  \hspace{1cm} (4.11)

where $a_0$ and $b_0$ are positive constants determining the amplitude of $x$ and the time when the amplitude of $x$ decreases. The reference mechanical energy of the upright position is given by,

$$E_{ref} = mgl$$  \hspace{1cm} (4.12)

When the pendulum is close to the upright position, it should be switched from this controller to a stabilizing one, which in this case is designed as a linear quadratic regulator (LQR). The LQR is designed by using the Matlab command lqr. The weights in the cost function that are minimized in the LQR are selected as $Q = \text{diag}(0.1, 0.1, 100, 1)$ and $R = 0.1$. The LQR control law can be given as $u = -Kx$, where $K = [-107.909 \ -27.9547 \ -31.6228 \ -25.2215]$. The switching from the swing-up to the LQR controller is done when $|E - E_{ref}| < \varepsilon_1$ and $\cos \theta > \varepsilon_2$, where $\varepsilon_1 = 0.1$ and $\varepsilon_2 = \cos(\pi/18)$. The constants of the swing-up controller are selected as $a_0 = 0.3$, $b_0 = 0.4$, $c_0 = 0.9$ and $\varsigma = 1.5$. The results of the simulations performed with this controller with the initial conditions $x_0 = [\pi \ 0 \ 0 \ 0]$, without any kind of sensor noise or actuator disturbance, are as follows,
Figure 4.4  Simulation results with controller for finite track length for $x_0 = [\pi \ 0 \ 0 \ 0]$

It can be observed from Figure 4.4 that the pendulum is stabilized in the upright position and the cart has returned to its home position without too much oscillating.
Neural Network Controller Design

Supervised learning by using function approximators, is used to learn control policies instead of only replaying a certain sequence of forces or setpoints. This is true since the function approximation that would be used to learn the control policy, can generalize from what it has learned. The imitating controller is presented demonstrations which are recorded. The recorded samples are strung together. The experiments are based on simulations and conclude two phases; the demonstration and the execution phases. The simulations are done with Matlab Simulink with a fixed step size of 0.1 sec by using ode5 (Dormand-Prince) fixed step solver. The neural network type which is used for imitating the controller designed to stabilize the inverted pendulum is a multilayer feedforward neural network (MLFF). This network is selected in order to learn the demonstrated policy since both of the designed controllers are static input/output mappings, and MLFFs are good at learning static i/o mappings. The parameters of this network such as hidden layer number, number of hidden neurons, are selected by trial and error according to the success of the controller in the execution phase. There are some empirical formulas in the literature (i.e. rules of thumb), for how to select the number of hidden layers and number of hidden neurons.

The design can be separated into two phases; the teaching and the execution phases. The teaching phase for each demonstration can be presented graphically as follows in Figure 5.1.

For each demonstration, i

![Figure 5.1 Teaching phase for supervised learning](image)

There is sensor noise present in the simulations which is selected as normally distributed random numbers with different initial seeds for each demonstration. Also band-limited white noise is added at the output of the controller to represent the different choices that a human demonstrator can make in the teaching phase, in order to make the demonstrations less consistent, and therefore a more realistic representation of human behavior. When the teaching phase is finished, the neural network is generated in order to be added to the model using Matlab’s gensim command. After that the execution phase shown in Figure 5.2, starts with the nonlinear controller replaced with the neural network, and the actuator disturbance is removed.

![Figure 5.2 Execution phase for supervised learning](image)
The neural network approximates the input-output mapping $F(\theta, \dot{\theta})$, for the first type of teacher presented in chapter 4.1 and $F(\theta, \dot{\theta}, x, \dot{x})$ for the second type of teacher presented in chapter 4.2. It does not try to approximate $F(t)$, i.e. the actuator forces as a function of time. The supervised learning method works by demonstrating the task, from which the control policy is distilled and imitated. This makes the number of the demonstrations presented an important parameter for the learning problem. The number of demonstrations should not be too low so that the state space is searched sufficiently by the given examples. It should also not be too large since in a real implementation it would consume too much time to teach the task. In addition to this, the noise level present in the demonstrations is also important, since too much sensor noise would make it harder for NN to learn the demonstrated behavior. Both controllers presented in the previous chapter work for all initial pendulum angles between $\theta_0 = [0, 2\pi]$, so in order to check whether the neural controller also does the same thing, the neural controller can be tested for some initial pendulum angles between $\theta_0 = [0, 2\pi]$. The noise level in the execution phase is selected the same as the noise level in the training phase but different initial seeds are used for the random number generators.

5.1 Neural Controller for the First Type of Teacher

The first type of teacher that would be used to supervise the neural network is the controller that does not take into account the finite track length; it’s a mapping from only pendulum angles and angular velocities to actuator forces. When (4.3) is checked, it is observed that this is a relatively simple input/output mapping compared to the mapping of the second type of teacher, so higher noise levels can be used in order to make the learning more challenging for the NN. Three experiments are going to be performed, in order to compare the effect of noise level and the number of demonstrations. Each demonstration starts from the pending initial position, $x_0 = \begin{bmatrix} \pi & 0 & 0 & 0 \end{bmatrix}$, and lasts 15 seconds. The network which was selected in order to approximate the behavior of the first type of teacher (controller) has two inputs (pendulum angle and angular velocity) and a single hidden layer with 25 hidden neurons in it. The activation function that is used in the hidden layer is tan-sigmoid, whereas a linear activation function is used in the output layer. The network has been trained using a Levenberg-Marquardt backpropagation (i.e. trainlm training algorithm from Matlab) with a constant learning rate of 0.001. The weights are updated for 500 epochs. The desired m.s.e. levels are selected as the variance of the actuator disturbances, since the neural network would start fitting the noise after that value. The neural controller would be tested by initial pendulum angles incremented by $\pi/180$ (i.e. $1^\circ$) between $\theta_0 = [0, 2\pi]$ in the execution phase. The duration of the simulations in the execution phase is taken as 30 seconds. The results of these three experiments and the specifications of the neural networks can be found in Table 5.1. The results of the first experiments is discussed in detail in this chapter, the details of the last two experiments can be found in Appendix A.2.

The first experiment is performed with 5 examples, and the desired mean square of the approximation error (mse) level is selected as 0.1 N$^2$. The variance of the normally distributed random numbers for sensor noise in each demonstration is selected as $\sigma^2 = (\pi/180)^2$ rad$^2$, which corresponds to a standard deviation of $1^\circ$. The actuator disturbance, representing the different actions of the demonstrator has a noise power level of $\sigma^2 = 0.1$ N$^2$ and a sampling time of 1 sec,
which is close to the free swinging period of the pendulum. This means that the demonstrator can make an inconsistent action every second with a standard deviation of \( \sigma = \sqrt{0.1} \approx 0.316 N \), which is an order of magnitude less than the typical forces that are used for the task which can be observed from the upper part of Figure 5.5. The pendulum angles for the five demonstrations are as follows,

![Pendulum angle for the demonstrations](image)

Figure 5.3 Pendulum angles for the demonstrations for the 1st experiment

It can be observed from Figure 5.3 that in all the demonstrations the pendulum is stabilized in the upright position. The results of the training using these demonstrations are given below. It can be observed from Figure 5.4 that the desired m.s.e level has not been reached, yet the approximation error has converged at an m.s.e. (mean square error) level of 0.1644. So a good performance in the execution phase can be expected. The training of the NN took approximately 18.6 seconds (measured by Matlab’s tic, toc functions).
Figure 5.4 Training results of the neural network
The plant input data (i.e. the target for the neural network), and its approximation obtained by the neural network at the end of the training session which is related to these examples can be presented below in Figure 5.5. The green plot is obtained by using Matlab’s sim command in order to observe how well the neural network will fit the desired output with the pendulum angles and angular velocities from the demonstrations.

Figure 5.5 Actuator forces from demonstrations and its NN approximation
A closer look might be taken to the approximation of the 3rd demonstration in Figure 5.5 and it can be observed from Figure 5.6 that the neural network has made a good approximation.
The input output map of the first designed controller with the demonstration trajectories in the $\theta$, $\dot{\theta}$ portion of the state space on top of it, can be given by Figure 5.7 as follows.

The controller given by (4.3) is made of three regions, one swing-up control region, and two linear control regions, which can be observed from Figure 5.7. The input output map of the neural controller can be given as follows in Figure 5.8.
It can be observed from Figure 5.8 that the controller has learned most of the swinging region that is demonstrated to it, and it has made generalization errors in some regions marked with circles. The linear control region on the right and the angle to switch from swinging control to linear control has been learned better compared to the one on the left. This might be due to insufficiency of demonstrated data in those regions. When input output maps in Figures 5.7 and 5.8 are compared, it could be concluded that the neural network’s generalization is acceptable. The neural controller managed to swing up and stabilize the pendulum in 361 out of 361 initial angles which corresponds to a success rate of 100 %. The trajectories of some of these successful executions in the $\theta - \dot{\theta}$ portion of the phase space are below in Figure 5.9.
The pendulum angles for several initial pendulum angles from simulations in the execution phase are below in Figure 5.10.

The second experiment is done again with five demonstrations, but the noise levels are increased. The variance of the sensor noise is \( \sigma^2 = (3\pi/180)^2 \text{ rad}^2 \), which corresponds to a standard deviation of 3°. The actuator disturbance has a noise power level of \( \sigma^2 = 0.9 \text{ N}^2 \),
corresponding to a standard deviation of $\sigma = \sqrt{0.9} \approx 0.949\, N$. The desired mean square of the approximation error (mse) is selected as 0.9 $N^2$. The demonstration data, results of the training and the NN approximation obtained with Matlab’s sim command can be found in Appendix A.2.1. The input output map of the first designed controller with the demonstration trajectories in the phase plane on top of it, can be given by Figure 5.11 as follows.

Figure 5.11  I/O map of 1st teacher for 2nd experiment with demonstrated trajectories

The input output map of the neural controller for the second experiment is presented below in Figure 5.12.

Figure 5.12  I/O map of NN for the 2nd experiment
It can be observed from this figure that the controller has learned most of the swinging region and linear control regions that are demonstrated to it, and it has made generalization errors more obviously in some regions which are circled in Figure 5.16. This might be because the higher noise levels both in the inputs and the target of the neural network. The neural controller managed to swing up and stabilize the pendulum in 361 out of 361 initial angles which corresponds to a success rate of 100%. The pendulum angles for several initial pendulum angles from simulations in the execution phase are below in Figure 5.13.

![Figure 5.13 Several pendulum angle plots from execution phase](image)

The third experiment would be performed with fewer demonstrations but with the same type of neural network, same type of training algorithm and also same noise levels as in the first experiment. Two demonstrations are used in the third experiment. The results of the training using these demonstrations are given below. It can be observed from Figure 5.14 that the desired m.s.e. level has been reached at the 145th epoch. So a good performance in the execution phase can be expected. The training of the NN took approximately 3.8 seconds.
Figure 5.14  Training results of the neural network

The input output map of the first designed controller with the demonstration trajectories in the phase plane on top of it can be given by the Figure 5.15 as follows,

Figure 5.15  I/O map of 1st teacher for 3rd experiment with demonstrated trajectories

The input output map of the neural controller for the third experiment is presented below.
Figure 5.16  I/O map of NN for the 3rd experiment

It can be observed from this figure that the controller has learned most of the swinging region and linear control regions that are demonstrated to it, and it has made generalization errors in some regions which are circled in Figure 5.16. This might be because there is insufficient data to generalize well in those regions, since there are only two demonstrations. The neural controller managed to swing up and stabilize the pendulum in 361 out of 361 initial angles which corresponds to a success rate of 100 %. The pendulum angles for several initial pendulum angles from simulations in the execution phase are below in Figure 5.17.

Figure 5.17  Several pendulum angle plots from execution phase
The results of these three experiments and the specifications of the neural networks can be presented in the Table 5.1 as below.

<table>
<thead>
<tr>
<th>Experiment</th>
<th>Number of neurons</th>
<th>Number of demonstrations</th>
<th>Input noise level</th>
<th>Target noise level</th>
<th>m.s.e. at the end of training</th>
<th>Number of epochs</th>
<th>Success rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>1st experiment</td>
<td>25</td>
<td>5</td>
<td>1°</td>
<td>0.1 N²</td>
<td>0.1644</td>
<td>500</td>
<td>% 100</td>
</tr>
<tr>
<td>2nd experiment</td>
<td>25</td>
<td>5</td>
<td>3°</td>
<td>0.9 N²</td>
<td>1.203</td>
<td>500</td>
<td>% 100</td>
</tr>
<tr>
<td>3rd experiment</td>
<td>25</td>
<td>2</td>
<td>1°</td>
<td>0.1 N²</td>
<td>0.09929</td>
<td>500</td>
<td>% 100</td>
</tr>
</tbody>
</table>

Table 5.2 Table of experiments and NN specifications for 1st type of teacher

5.2 Neural Controller for the Second Type of Teacher

The second type of teacher which is explained in section 4.2, is a more complex mapping from pendulum angles and angular velocities, and cart positions and velocities to actuator forces, when compared to the first type of teacher. Therefore lower sensor noise and actuator disturbance levels compared to the first type of teacher will be used in the demonstrations. The variance of the sensor noise is $\sigma^2 = (0.2\pi/180)^2 \text{rad}^2$, which corresponds to a standard deviation of $0.2^\circ$.

The actuator disturbance has a noise power level of $\sigma^2 = 0.05 \text{N}^2$, corresponding to a standard deviation of $\sigma = \sqrt{0.05} \approx 0.224 \text{N}$. The desired mean square of the approximation error (mse) is selected as 0.05 N². The state-space in which the neural network would approximate the mapping is 4 dimensional, so a complete input-output map cannot be presented for this case, which makes it harder to interpret the results of learning of NN compared to 2D case, but some cross sections might be used to make comparisons. Such a cross section might be where $x = 0$ and $\dot{x} = 0$ or another cross section might be where $\theta = 0$ and $\dot{\theta} = 0$. A series of experiments have been done comparing the effect of initial conditions of demonstrations, the number of demonstrations, the number of epochs, and the number of neurons. The results of all the experiments are given at the end of chapter 5.2 in Table 5.3. The results of the 1st, 5th and 7th experiments will be investigated further in detail. In the first experiment, 5 demonstrations all starting from the initial condition $x_0 = [\pi \ 0 \ 0 \ 0]$, with duration of 15 seconds will be used. The NN that will be used to approximate the 4D mapping, consists of 25 hidden neurons with tansig activation functions. The learning algorithm is selected as Levenberg-Marquardt with a learning rate of 0.001 and 500 epochs are used to update the weights. These are the same settings as in the 1st type of teacher which is summarized in Table 5.1 and they are used here since the results obtained with the NN for the 1st type of teacher are quite well. The pendulum angles and the cart positions for these five demonstrations can be presented as follows in Figure 5.18.
Figure 5.18 Pendulum angles and cart positions for the demonstrations with 2nd teacher

It can be observed that the pendulum has been stabilized and the cart has returned to its home position in all demonstrations. The results of the training using this demonstrated data are below in Figure 5.19. The training took approximately 26.6 seconds. It could be observed that the training error has nearly converged but it is still high according to the m.s.e. level, so a good performance cannot be expected from the neural network in the execution phase.

Figure 5.19 Training results of the neural network

The plant input data (i.e. the target for the neural network), and its approximation obtained by the
neural network at the end of the training session which is related to these examples can be presented below. The green plot is obtained by using Matlab’s sim command in order to observe how well the neural network would fit the desired output with the pendulum angles, angular velocities, cart positions and velocities from the demonstrations.

![Plant input original and approximated](image1)

![Plant input approximation error](image2)

Figure 5.20 Actuator forces from demonstrations and its NN approximation

A closer look into Figure 5.20 is not necessary since from this figure it could already be observed that the approximation is not good. The $x = 0$ and $\dot{x} = 0$ cross section of the input output map of the 2nd type of teacher with the demonstration trajectories in the phase plane on top of it can be given by Figure 5.21 as follows.
Figure 5.21  I/O map of 2nd teacher with demonstrated trajectories

The $x = 0$ and $\dot{x} = 0$ cross section of the input output map of the neural controller can be given by Figure 5.22 as follows.

Figure 5.22  I/O map of NN controller

It may be concluded from Figure 5.22 that the neural controller’s approximation is not good. The neural controller will be tested by initial pendulum angles incremented by $\pi/180$ (i.e. $1^\circ$) between $\theta_0 = [0, 2\pi]$ in the execution phase. The duration of the simulations in the execution
phase is taken as 40 seconds. The neural controller could not stabilize the pendulum in any case, leading to a success rate of %0. The neural controller that is designed for the first type of teacher did not work for the 2nd type of teacher. This might be due to the more complex mapping of the 2nd type of teacher compared to the 1st type of teacher, and its larger input size (4 dimensions instead of 2). A better way to learn the second type of teacher is by starting the demonstrations from random initial conditions (i.e. not from \( x_0 = [\pi \ 0 \ 0 \ 0] \)) such as \( x_0 = [\pi + r_1 \ r_2 \ r_3 \ r_4] \), where \( r_1, r_2, r_3 \) and \( r_4 \) are normally distributed random numbers with zero mean and unit variance. This random variation of the initial conditions will facilitate the searching of the state space for the NN to learn the control policy of the 2nd type of teacher. The specifications of the NN and the learning algorithm has not been changed. In the 5th experiment, again five demonstrations are used in order to obtain the following results. The pendulum angles and the cart positions for these five demonstrations can be presented as follows.

![Pendulum angle for the demonstrations](image1)

![Cart position for the demonstrations](image2)

**Figure 5.23 Pendulum angles and cart positions for the demonstrations with 2nd teacher**

It could be observed that the pendulum has been stabilized and the cart has returned to its home position in all demonstrations. The results of the training using this demonstrated data are below. The training took approximately 26.6 seconds. It could be observed that the training error has nearly converged. It is lower than the case with the demonstrations starting from the initial conditions \( x_0 = [\pi \ 0 \ 0 \ 0] \), but it is still higher than the desired approximation error level.
Figure 5.24  Training results of the neural network

The plant input data (i.e. the target for the neural network), and its approximation obtained by the neural network at the end of the training session which is related to these examples can be presented below. The green plot is obtained by using Matlab’s sim command in order to observe how well the neural network would fit the desired output with the pendulum angles, angular velocities, cart positions and velocities from the demonstrations.

Figure 5.25  Actuator forces from demonstrations and its NN approximation

A closer look to the 1st demonstration of the 5th experiment, the input output map of the controller
and the NN controller can be found in Appendix A.2.2. The number of demonstrations used in the 5th experiment is not enough for a successful execution phase, so it is increased in the 7th experiment from five to ten. The demonstration data, the results of the training, the approximation of the NN, the input output map of the controller and the NN for the 7th experiment can be found in Appendix A.2.2. The neural controller managed to swing up and stabilize the pendulum and also returned to its home position in 361 out of 361 initial angles which corresponds to a success rate of 100 %. The trajectories of some of these successful executions in the $\theta - \dot{\theta}$ portion of the phase space are below in Figure 5.26.

![Figure 5.26 Successful trajectories obtained with NN](image)

The pendulum angles and cart positions for several initial pendulum angles from simulations in the execution phase are below in Figure 5.27.
All the experiments which are performed for the 4D case are given below in Table 5.3. Some final conclusions might be drawn from this table. When the first 4 experiments are checked, it can be observed that the success rate in completing the task in the execution phase is low. This is due to the fact that the initial conditions of the demonstrations are fixed at $x_0 = [\pi 0 0 0]$. This might be concluded since even though the number of demonstrations is increased from 5 to 10, the increase in the success rate is still very low. This can further be verified when the experiments from 9 to 12 are compared, since the number of neurons have been increased but the success rate
does not show a significant improvement at all. The number of demonstrations is also important which can be observed when experiments 5 and 7 are compared. This can also be seen from the 13th and 14th and 16th experiments, since with the 5 demonstrations the success rate is very low. When the 15th and 16th experiments are compared, the effect of the number of epochs can be observed better. The m.s.e. at the end of the training session has decreased by taking more epochs which led to a higher success rate in the execution phase. It might be observed from the last 8 experiments that the number of neurons in the hidden layer did not have that much effect on the success rate in the execution phase.

<table>
<thead>
<tr>
<th>No. of experiment</th>
<th>No. of neurons</th>
<th>No. of epochs</th>
<th>Initial conditions of demonstrations</th>
<th>No. of demonstrations</th>
<th>Training time</th>
<th>m.s.e. at the end of training</th>
<th>Success rate in executions</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>25</td>
<td>500</td>
<td>( x_0 = [\pi\ 0\ 0\ 0] )</td>
<td>5</td>
<td>26.6 sec</td>
<td>2.417 %</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>25</td>
<td>1000</td>
<td>( x_0 = [\pi\ 0\ 0\ 0] )</td>
<td>5</td>
<td>54.8 sec</td>
<td>2.365 %</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>25</td>
<td>500</td>
<td>( x_0 = [\pi\ 0\ 0\ 0] )</td>
<td>10</td>
<td>45.8 sec</td>
<td>1.751 %</td>
<td>11.4</td>
</tr>
<tr>
<td>4</td>
<td>25</td>
<td>1000</td>
<td>( x_0 = [\pi\ 0\ 0\ 0] )</td>
<td>10</td>
<td>102.7 sec</td>
<td>1.75 %</td>
<td>11.4</td>
</tr>
<tr>
<td>5</td>
<td>25</td>
<td>500</td>
<td>( x_0 = [\pi+\tau_1\ \tau_2\ \tau_3\ \tau_4] )</td>
<td>5</td>
<td>26.6 sec</td>
<td>0.3638 %</td>
<td>0.5</td>
</tr>
<tr>
<td>6</td>
<td>25</td>
<td>1000</td>
<td>( x_0 = [\pi+\tau_1\ \tau_2\ \tau_3\ \tau_4] )</td>
<td>5</td>
<td>56.4 sec</td>
<td>0.3584 %</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>25</td>
<td>500</td>
<td>( x_0 = [\pi+\tau_1\ \tau_2\ \tau_3\ \tau_4] )</td>
<td>10</td>
<td>46.1 sec</td>
<td>0.3706 %</td>
<td>100</td>
</tr>
<tr>
<td>8</td>
<td>25</td>
<td>1000</td>
<td>( x_0 = [\pi+\tau_1\ \tau_2\ \tau_3\ \tau_4] )</td>
<td>10</td>
<td>101 sec</td>
<td>0.357 %</td>
<td>99.2</td>
</tr>
<tr>
<td>9</td>
<td>50</td>
<td>500</td>
<td>( x_0 = [\pi\ 0\ 0\ 0] )</td>
<td>5</td>
<td>68.7 sec</td>
<td>1.5676 %</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>50</td>
<td>1000</td>
<td>( x_0 = [\pi\ 0\ 0\ 0] )</td>
<td>5</td>
<td>143 sec</td>
<td>1.5390 %</td>
<td>0</td>
</tr>
<tr>
<td>11</td>
<td>50</td>
<td>500</td>
<td>( x_0 = [\pi\ 0\ 0\ 0] )</td>
<td>10</td>
<td>119.5 sec</td>
<td>1.9754 %</td>
<td>6.9</td>
</tr>
<tr>
<td>12</td>
<td>50</td>
<td>1000</td>
<td>( x_0 = [\pi\ 0\ 0\ 0] )</td>
<td>10</td>
<td>254.1 sec</td>
<td>1.2620 %</td>
<td>9.1</td>
</tr>
<tr>
<td>13</td>
<td>50</td>
<td>500</td>
<td>( x_0 = [\pi+\tau_1\ \tau_2\ \tau_3\ \tau_4] )</td>
<td>5</td>
<td>70.8 sec</td>
<td>0.8113 %</td>
<td>0</td>
</tr>
<tr>
<td>14</td>
<td>50</td>
<td>1000</td>
<td>( x_0 = [\pi+\tau_1\ \tau_2\ \tau_3\ \tau_4] )</td>
<td>5</td>
<td>135 sec</td>
<td>0.0862 %</td>
<td>0</td>
</tr>
<tr>
<td>15</td>
<td>50</td>
<td>500</td>
<td>( x_0 = [\pi+\tau_1\ \tau_2\ \tau_3\ \tau_4] )</td>
<td>10</td>
<td>119.6 sec</td>
<td>0.9252 %</td>
<td>46.5</td>
</tr>
<tr>
<td>16</td>
<td>50</td>
<td>1000</td>
<td>( x_0 = [\pi+\tau_1\ \tau_2\ \tau_3\ \tau_4] )</td>
<td>10</td>
<td>256 sec</td>
<td>0.1076 %</td>
<td>98.3</td>
</tr>
</tbody>
</table>

Table 5.3 Table of experiments and NN specifications for 2nd type of teacher
6 Conclusions and Recommendations

6.1 Conclusions

Various machine learning methods for completing atomic tasks are investigated in this internship by introducing the advantages and drawbacks of each of them. Several function approximators that are frequently used in these machine learning methods are also introduced. Swinging up and balancing an inverted pendulum on a cart from initially pendant position to the upright position is taken as an example case to investigate one of these machine learning methods, namely supervised learning.

The supervised learning method consists of two phases, demonstration and execution. In the demonstration phase necessary data is collected from the supervisor in order to design a neural network. The objective of the neural network is to imitate the control law of the teacher. In the execution phase the neural controller is tested by replacing the teacher. The method is tested by using two kinds of supervisors (teachers), where the first one has a control policy that ignores the finite track length of the cart, and the second one does take into account the finite track length. Since the first type of teacher has a control law with only two inputs (pendulum angle and angular velocity), it was easier to draw conclusions about what the neural controller that is used to imitate the teacher does, by comparing the input/output (state-action) mappings using 2D color plots. The policy of a demonstrator (i.e. designed controller in this case) is not a function that can be directly measured by giving inputs and measuring outputs, so learning the complete i/o map and making a perfect generalization would not be possible. The complexity of the function that would be approximated is also important since the function may include both continuous and discontinuous components (such as switches between different control laws). The number of demonstrations using supervised learning are lower compared to the number of trials and errors in reinforcement learning. In reinforcement learning, these are at the level of hundreds or even thousands.

6.2 Recommendations

The input dimension of the policy is also important since for a domestic task that would be performed by a humanoid robot this might be quite high. A solution to this might be separating this high dimensional mapping into mappings with smaller input dimensions, preferably two, in order to check the i/o maps to draw conclusions. Then learning these smaller mappings by neural networks. But this solution would bring other problems such as how to separate the demonstrated data and also how to combine the learned mappings afterwards. The policy that would be learned by the neural network might contain hidden states, making the mapping dynamic rather than static. So it might be necessary to select a different type of neural network to cope with this. How to prevent the undesired actions of the imitation learning algorithm in unexplored areas of the control policy should also be further investigated.

For domestic tasks that would be taught to a home robot by imitation, the method that is presented in chapter 5, in this internship report might be used, but other methods which are presented in chapter 2 should also be investigated further.
A. Appendix

A.1 Equations of Motion

The equations of motion of the inverted pendulum on a cart are given in this section. The system can be divided into two separate free-body diagrams as shown in Figure A.1.1.

The coordinates of the center of gravity of the pendulum can be written as,

\[ x_G = x + l\sin \theta \]  
\[ y_G = l\cos \theta \]  

The velocities and accelerations in these directions can be calculated from here,

\[ \dot{x}_G = \dot{x} + \dot{\theta} \cos \theta \]  
\[ \ddot{x}_G = \ddot{x} + \dot{\theta} \cos \theta - \dot{\theta}^2 \sin \theta \]  
\[ \dot{y}_G = -\dot{\theta} \sin \theta \]  
\[ \ddot{y}_G = -\dot{\theta} \sin \theta - \dot{\theta}^2 \cos \theta \]

The force equilibrium for the pendulum in x-direction can be given as,

\[ -m\ddot{x}_G + H = 0 \]  

The force equilibrium for the pendulum in y-direction can be given as,

\[ V - mg - m\ddot{y}_G = 0 \]

The torque equilibrium for the pendulum is,

\[ -l\ddot{\theta} + Vl \sin \theta - Hl \cos \theta = 0 \]
The force equilibrium for the cart in x-direction is,

\[ F - M \ddot{x} - H = 0 \]  
\[ (A.1.10) \]

By combining (A.1.4), (A.1.6), (A.1.7), (A.1.8), and (A.1.9),

\[ -I \ddot{\theta} + \left( mg - ml \ddot{\theta} \sin \theta - ml \dot{\theta}^2 \cos \theta \right) l \sin \theta - \left( m \dddot{x} + ml \dot{\theta} \cos \theta - ml \dot{\theta}^2 \sin \theta \right) l \cos \theta = 0 \]  
\[ (A.1.11) \]

\[ (I + ml^2) \ddot{\theta} - mgl \sin \theta + ml \dot{x} \cos \theta = 0 \]  
\[ (A.1.12) \]

By combining (A.1.4), (A.1.7) and (A.1.10),

\[ F - M \ddot{x} - (m \dddot{x} + ml \dot{\theta} \cos \theta - ml \dot{\theta}^2 \sin \theta) = 0 \]  
\[ (A.1.13) \]

\[ (M + m) \ddot{x} + ml \dot{\theta} \cos \theta - ml \dot{\theta}^2 \sin \theta = F \]  
\[ (A.1.14) \]

By solving for \( \ddot{x} \), and \( \ddot{\theta} \), from (A.1.12) and (A.1.14),

\[
\begin{bmatrix}
I + ml^2 & ml \cos \theta \\
ml \cos \theta & M + m
\end{bmatrix}
\begin{bmatrix}
\ddot{\theta} \\
\ddot{x}
\end{bmatrix}
=
\begin{bmatrix}
mgl \sin \theta \\
F + ml \dot{\theta}^2 \sin \theta
\end{bmatrix}
\]  
\[ (A.1.15) \]

\[
\begin{bmatrix}
\ddot{\theta} \\
\ddot{x}
\end{bmatrix}
=
\frac{1}{(I + ml^2)(M + m) - ml^2 \cos^2 \theta}
\begin{bmatrix}
M + m & -ml \cos \theta \\
-ml \cos \theta & I + ml^2
\end{bmatrix}
\begin{bmatrix}
mgl \sin \theta \\
F + ml \dot{\theta}^2 \sin \theta
\end{bmatrix}
\]  
\[ (A.1.16) \]

\[
\begin{bmatrix}
\ddot{\theta} \\
\ddot{x}
\end{bmatrix}
=
\frac{1}{\Delta(\theta)}
\begin{bmatrix}
(M + m) mgl \sin \theta - ml^2 \dot{\theta}^2 \sin \theta \cos \theta - ml \cos \theta \cdot F \\
-ml^2 g \sin \theta \cos \theta + (I + ml^2) ml \dot{\theta}^2 \sin \theta + (I + ml^2) \cdot F
\end{bmatrix}
\]  
\[ (A.1.17) \]

where, \( \Delta(\theta) = (I + ml^2)(M + m) - ml^2 \cos^2 \theta \).
A.2 Simulation Results

A.2.1 Results with the First Type of Teacher

The results of the second and third experiments are presented in detail in this section. The pendulum angles for the five demonstrations in the 2\textsuperscript{nd} experiment can be given below.

![Figure A.2.1 Pendulum angles for the demonstrations for the 2\textsuperscript{nd} experiment](image)

It can be observed from the Figure A.2.1 that the pendulum is stabilized in the upright position in all demonstrations. The results of the training using these demonstrations are given below. It can be observed from Figure A.2.2 that the desired m.s.e level has not been reached, yet the approximation error has converged at an m.s.e. level of 1.203. So a good performance in the execution phase can be expected. The training of the NN took approximately 18.2 seconds.
The plant input data (i.e. the target for the neural network), and its approximation obtained by the neural network at the end of the training session which is related to these examples can be presented below. The green plot is obtained by using Matlab’s sim command in order to observe how well the neural network fits the desired output with the pendulum angles and angular velocities from the demonstrations.

A closer look might be taken to the approximation of the 3rd demonstration in Figure A.2.3 and it
can be observed from Figure A.2.4 that the neural network has made a good approximation.

![Plant input original and approximated for the 3rd demonstration](image1)

**Figure A.2.4  Actuator forces from 3rd demonstration and its NN approximation**

The pendulum angles for the two demonstrations in the 3rd experiment can be presented as follows.

![Pendulum angle for the demonstrations](image2)

**Figure A.2.5  Pendulum angles for the demonstrations for the 3rd experiment**

It can be observed from Figure A.2.5 that in all the demonstrations the pendulum is stabilized in the upright position. The plant input data (i.e. the target for the neural network), and its approximation obtained by the neural network at the end of the training session which is related
to these examples can be presented below. The green plot is obtained by using Matlab’s sim command in order to observe how well the neural network would fit the desired output with the pendulum angles and angular velocities from the demonstrations.

Figure A.2.6  Actuator forces from demonstrations and its NN approximation

A closer look might be taken to the approximation of the 2nd demonstration in Figure A.2.6 and it can be observed from Figure A.2.7 that the neural network has made a good approximation.

Figure A.2.7  Actuator forces from 2nd demonstration and its NN approximation
A.2.2 Results with the Second Type of Teacher

The results of the 5th and 7th experiments are given in detail in this chapter. For the 5th experiment, a closer look might be taken to the approximation of the 1st demonstration in Figure 5.25 and it can be observed from Figure A.2.8 that the neural network has made a better approximation.

![Plant input original and approximated for the 1st demonstration](image1)

![Plant input approximation error for the 1st demonstration](image2)

**Figure A.2.8  Actuator forces from 1st demonstration and its NN approximation**

The $x = 0$ and $\dot{x} = 0$ cross section of the input output map of the second designed controller with the demonstration trajectories in the phase plane on top of it can be given by the Figure A.2.9 as follows.
The $x = 0$ and $\dot{x} = 0$ cross section of the input output map of the neural controller can be given by Figure A.2.10 as follows. It might be observed from Figure A.2.10 that the NN started learning the areas that are encountered during the demonstrations, better compared to the last case.
The neural controller is tested by initial pendulum angles incremented by $\pi/180$ (i.e. $1^\circ$) between $\theta_0 = [0, 2\pi]$ in the execution phase. The duration of the simulations in the execution phase is taken as 40 seconds. The neural controller managed to swing up and stabilize the pendulum in 2 out of 361 initial angles which corresponds to a success rate of 0.5%. This low success rate might be due to the fact that the number of demonstrations are still low, so it will be increased from five to ten. The pendulum angles and the cart positions for these ten demonstrations for the $\gamma^{th}$ experiment can be presented as follows.

![Pendulum angles and cart positions for the demonstrations with 2nd teacher](image)

**Figure A.2.11** Pendulum angles and cart positions for the demonstrations with 2nd teacher

It could be observed that the pendulum has been stabilized and the cart has returned to its home position in all demonstrations. The results of the training using this demonstrated data are below in Figure A.2.12. The training took approximately 46.1 seconds. It could be observed that the training error has nearly converged. It is close to the case with the five demonstrations starting from the initial conditions $x_0 = [\pi + r_1 \ r_2 \ r_3 \ r_4]$, but it is still higher than the desired approximation error level.
The plant input data (i.e. the target for the neural network), and its approximation obtained by the neural network at the end of the training session which is related to these examples can be presented below. The green plot is obtained by using Matlab’s sim command in order to observe how well the neural network would fit the desired output with the pendulum angles, angular velocities, cart positions and velocities from the demonstrations.
A closer look might be taken to the approximation of the 1st demonstration in Figure 5.40 and it can be observed from Figure 5.41 that the neural network has made a good approximation.

![Plant input original and approximated for the 1st demonstration](image1)

![Plant input approximation error for the 1st demonstration](image2)

**Figure A.2.14 Actuator forces from 1st demonstration and its NN approximation**

The $x = 0$ and $\dot{x} = 0$ cross section of the input output map of the second designed controller with the demonstration trajectories in the phase plane on top of it can be given by the Figure 5.42 as follows.

![Input-output map of the 2nd classical controller](image3)

**Figure A.2.15 I/O map of 2nd teacher with demonstrated trajectories**

The $x = 0$ and $\dot{x} = 0$ cross section of the input output map of the neural controller can be given by Figure 5.43 as follows.
It could be observed from Figure 5.43, that the neural network has learned the swinging region better compared to the previous two experiments, and it has made some generalization errors which are circled on the figure.
A.3. M-files

This is the m-file which is used for the designing the LQR controller for the second type of teacher in chapter 4.2.

clear all, close all, clc
syms('x1','x2','x3','x4','u','real')
g = 9.81;       % gravity
m = 0.1;        % mass of bar
M = 1;          % mass of cart
L = 1;          % length of bar
I = m*L^2/12;   % moment of inertia of bar wrt COG
l = L/2;

% dx/dt = f(x) + g(x)*u
Deltax = (I + m*l^2)*(M + m) - m^2*l^2*cos(x1)^2;
fx = [x2;
    ((M + m)*m*g*l*sin(x1) - m^2*l^2*x2^2*sin(x1)*cos(x1))/Deltax;
   x4;
   (-m^2*l^2*g*sin(x1)*cos(x1) + (I + m*l^2)*m*l*x2^2*sin(x1))/Deltax];
gx = [0;
    -m*l*cos(x1)/Deltax;
    0;
    (I + m*l^2)/Deltax];
dxdt = fx + gx.*u;
x = [x1,x2,x3,x4];
A = jacobian(dxdt,x);
B = jacobian(dxdt,u);

% Substituting the equilibrium point
xeq = [0 0 0 0].';
Ae = double(subs(A,x,xeq));
Be = double(subs(B,x,xeq));
Ce = [1 0 0 0];
De = 0;
egv = eig(Ae);
sys = ss(Ae,Be,Ce,De);
[num,den] = ss2tf(Ae,Be,Ce,De);
for i = 1:length(num)
    if abs(num(i))<100*eps
        num(i) = 0;
    else
        num(i) = num(i);
    end
end
for j = 1:length(den)
    if abs(den(j))<100*eps
        den(j) = 0;
    else
        den(j) = den(j);
    end
end
Gs = tf(num,den);

% LQR design for the linearised plant
Q = diag([0.1 0.1 100 1]);
R = 0.1;
[K,S,E] = lqr(Ae,Be,Q,R);
This is the m-file which used in demonstration and execution phases in chapter 5.1.

clear all
close all
clc
% Number of demonstrations
n = 2;
% Initial data length size
len = 0;
% Teaching phase simulations
% Initial conditions of the simulations (i.e. pendant position)
x0 = [pi 0 0 0].';
% Demonstration time for each simulation
demo_time = 15;
% Initial seeds for random number generators
% Initial seed for noise in pendulum angle
rand('state',1);
rnd1 = rand(n,1);
% Initial seed for inconsistency of teacher
rand('state',2);
rnd2 = rand(n,1);
% Demonstration sensor noise level in pendulum angle
sgm = pi/180;
% Demonstration inconsistency of teacher
noise_power = 0.1;
input = [];
for i = 1:n
% Initial seed for measurement noise in theta
seed_theta = round(10*rnd1(i)*n);
% Initial seed for actuator disturbance
seed_input = round(10*rnd2(i)*n+1);
% Performing of simulations
sim('invpend_2D');
% Total data length
totlen = len + length(theta);
% Network input data
P(:,(len+1):totlen) = [theta thetadot].';
% Network target data
T(:,(len+1):totlen) = input.,'
% Theta for trajectories
traj(:,2*i-1) = theta;
% Thetadot for trajectories
traj(:,2*i) = thetadot;
% Total time vector
time(:,(len+1):totlen) = tout.,'
len = totlen;
end
figure(1);
set(gca,'FontSize',12)
plot(P(1,:))
xlabel('time index','FontSize',12)
ylabel('Pendulum angle, \theta [rad]','FontSize',12)
title('Pendulum angle for the demonstrations','FontSize',12)
xlim([0 totlen])
figure(2);
%% Training phase with Levenberg-Marquardt
% Type and initialization of neural network multilayer feedforward network
net1 = newff(minmax(P),[25,1],{'tansig','purelin'},'trainlm');
net1.trainParam.show = 100;
net1.trainParam.lr = 0.001;
net1.trainParam.epochs = 500;
net1.trainParam.goal = noise_power;
% Training of the neural network
[tic 
net1,tr] = train(net1,P,T);
tr_time_lm = toc;
% Simulation of the neural network
a = sim(net1,P);
figure(3);
set(gca,'FontSize',12)
plot(1:length(T),T,1:length(T),a)
xlabel('time index','FontSize',12)
ylabel('plant input [N]','FontSize',12)
title('Plant input original and approximated','FontSize',12)
legend('Plant input from demonstrations','Approximation at the end of training')
xlim([0 totlen])
figure(4);
set(gca,'FontSize',12)
plot(1:length(T),a-T)
xlabel('time index','FontSize',12)
ylabel('approximation error','FontSize',12)
title('Plant input approximation error','FontSize',12)
xlim([0 totlen])
figure(5);
set(gca,'FontSize',12)
semilogy(tr.epoch,tr.perf,'LineWidth',2);
xlabel('Epochs','FontSize',12);
ylabel('Performance','FontSize',12);
xlim([0 length(tr.epoch)])
ylim([0.05 100])
%% Generate simulink block
open('invpend_2D_NN');
% Take the block that is generated with this gensim command and then insert it in the place of the controller in the file called invpend_2D_NN.mdl
gensim(net1,-1)

%% Execution phase
count = 0;
leng = 0;
% Execution time for each simulation
exec_time = 30;
% Initial pendulum angles for execution phase
th0 = 0:pi/180:2*pi;
% Initial seeds for random number generators
rand('state',10);
rnd1 = rand(length(th0),1);
rand('state',20);
rnd2 = rand(length(th0),1);

% Demonstration sensor noise level
sgmnn = sgm;

% Execution phase simulations
for i = 1:length(th0)
    x0 = [th0(i) 0 0 0].';
    % Initial seed for measurement noise in theta
    seed_thetann = floor(1000*rnd1(i)*length(th0));
    % Initial seed for measurement noise in thetadot
    seed_thetadotnn = floor(1000*rnd2(i)*length(th0)+1);
    sim('invpend_2D_NN');
    if abs(mean(cos(thetann(200:301))-1))<=0.1 &&
    abs(mean(thetadotnn(200:301)))<=0.1
        % Update counter
        count = count + 1;
        % Theta for trajectories
        trajnn(:,2*count-1) = thetann;
        % Thetadot for trajectories
        trajnn(:,2*count) = thetadotnn;
        % Plot successful phase plane trajectories
        figure(6);
        plot(trajnn(:,2*count-1),trajnn(:,2*count),'Color',[0 0 0])
        xlabel('pendulum angle','FontSize',12)
        ylabel('pendulum angular velocity','FontSize',12)
        title('Successful trajectories obtained in the execution phase','FontSize',12)
        hold on
    else
        count = count;
        figure(7);
        set(gca,'FontSize',12)
        plot(thetann,thetadotnn,'Color',[0 0 0])
        xlabel('pendulum angle','FontSize',12)
        ylabel('pendulum angular velocity','FontSize',12)
        title('Unsuccessful trajectories obtained in the execution phase','FontSize',12)
        hold on
    end
    % Total data length
    total = leng + length(thetann);
    % Execution angles in a vector
    Fnn(:,(leng+1):total) = [thetann thetadotnn].';
    % Control inputs generated in the execution phase
    Tnn(:,(leng+1):total) = inputnn.';
    leng = total;
end

% Percentage of success rate in swinging up and balancing the pendulum
prcnt = count/length(th0)*100;
figure(8);
set(gca,'FontSize',12)
plot(Fnn(1,:))
figure(9);
set(gca,'FontSize',12)
\begin{verbatim}
plot(Tnn)
%% Input-output map
% Pendulum angles
th = -pi/9:pi/72:2*pi+pi/9;
% Pendulum angular velocities
thdot = -8:0.1:8;
for i = 1:length(thdot)
    for j = 1:length(th)
        % Original controller input output behavior
        u = pl_inp(th(j),thdot(i));
        % Approximated controller input output behavior
        unn = sim(net1,[th(j);thdot(i)]);
        func(i,j) = u;
        func_nn(i,j) = unn;
    end
end
figure(10);
set(gca,'FontSize',12)
surf(th,thdot,func)
colorbar;
caxis(5*[[-1,1]]);
view(0,90);
xlabel('theta','FontSize',12)
ylabel('thetadot','FontSize',12)
zlabel('controller output','FontSize',12)
title('Input-output map of the 1\textsuperscript{st} classical controller','FontSize',12)
% Trajectories recorded from demonstrations
hold on
for i = 1:n
    plot3(traj(:,2*i-1),traj(:,2*i),1000*ones(length(traj(:,2*i))),'Color',[0 0 0])
hold on
end
xlim(minmax(th))
ylim(minmax(thdot))
figure(11);
set(gca,'FontSize',12)
surf(th,thdot,func_nn)
colorbar;
caxis(5*[[-1,1]]);
view(0,90);
xlabel('theta','FontSize',12)
ylabel('thetadot','FontSize',12)
zlabel('controller output','FontSize',12)
title('Input-output map of the 1\textsuperscript{st} neural network controller','FontSize',12)
xlim(minmax(th))
ylim(minmax(thdot))
figure(12);
set(gca,'FontSize',12)
surf(th,thdot,func_nn)
colorbar;
caxis(5*[[-1,1]])
view(0,90);
xlabel('theta','FontSize',12)
ylabel('thetadot','FontSize',12)
zlabel('controller output','FontSize',12)
title('Input-output map of the 1\textsuperscript{st} neural network controller','FontSize',12)
\end{verbatim}
This is the m-file which used in demonstration and execution phases in chapter 5.2.

clear all
close all
clc

% Number of demonstrations
n = 10;
% Initial data length size
len = 0;
% Teaching phase simulations
% Initial conditions of the simulations
x0 = [pi 0 0 0].';
% Demonstration time for each simulation
demo_time = 15;
% Initial seeds for random number generators
rand('state',1);
rnd1 = rand(n,1);
rand('state',2);
rnd2 = rand(n,1);
rand('state',3);
rnd3 = rand(n,1);
% Demonstration sensor noise level
sgm = 0.2*pi/180;
% Demonstration actuator disturbance
noise_power = 0.05;
input = [];
for i = 1:n
    x0 = [pi+r1(i) r2(i) r3(i) r4(i)].';
    xinit(:,i) = x0;
    % Initial seed for measurement noise in theta
    seed_theta = round(10*rnd1(i)*n);
    % Initial seed for measurement noise in thetadot
    seed_x = round(10*rnd2(i)*n+1);
    % Initial seed for actuator disturbance
    seed_input = round(10*rnd3(i)*n+2);
% Performing of simulations
sim('invpend_4D');
% Total data length
totlen = len + length(theta);
% Network input data
P(:,(len+1):totlen) = [theta thetadot x xdot].';
% Network target data
T(:,(len+1):totlen) = input.';
% Theta for trajectories
traj(:,2*i-1) = theta;
% Thetadot for trajectories
traj(:,2*i) = thetadot;
% Theta for trajectories
postraj(:,2*i-1) = x;
% Thetadot for trajectories
postraj(:,2*i) = xdot;
% Total time vector
time(:,(len+1):totlen) = tout.';
len = totlen;
end
figure(1);
set(gca,'FontSize',12)
plot(P(1,:))
xlabel('time index','FontSize',12)
ylabel('pendulum angle, \theta [rad]','FontSize',12)
title('Pendulum angle for the demonstrations','FontSize',12)
xlim([0 totlen])
saveas(gcf,'demo_ang.fig')
figure(2);
set(gca,'FontSize',12)
plot(P(3,:))
xlabel('time index','FontSize',12)
ylabel('cart position, x [m]','FontSize',12)
title('Cart position for the demonstrations','FontSize',12)
xlim([0 totlen])
saveas(gcf,'demo_pos.fig')
figure(3);
set(gca,'FontSize',12)
xlim([0 totlen])
plot(T)
xlabel('time index','FontSize',12)
ylabel('plant input [N]','FontSize',12)
title('Plant input for the demonstrations','FontSize',12)
xlim([0 totlen])
saveas(gcf,'demo_inp.fig')

%% Training phase with Levenberg-Marquardt
% Type and initialization of neural network multilayer feedforward network
net1 = newff(minmax(P),[25,1],{'tansig','purelin'},'trainlm');
net1.trainParam.show = 100;
net1.trainParam.lr = 0.001;
net1.trainParam.epochs = 1000;
net1.trainParam.goal = noise_power;
% Training of the neural network
[tic,net1,tr] = train(net1,P,T);
tr_time_lm = toc;
% Simulation of the neural network
a = sim(net1,P);
figure(4);
set(gca,'FontSize',12)
plot(1:length(a),T,1:length(a),a)
xlabel('time index','FontSize',12)
ylabel('plant input [N]','FontSize',12)
title('Plant input original and approximated','FontSize',12)
legend('Plant input from demonstrations','Approximation at the end of training')
xlim([0 totlen])
saveas(gcf,'pl_inp_approx.fig')
figure(5);
set(gca,'FontSize',12)
plot(1:length(a),a-T)
xlabel('time index','FontSize',12)
ylabel('approximation error','FontSize',12)
title('Plant input approximation error','FontSize',12)
xlim([0 totlen])
saveas(gcf,'pl_inp_approx_err.fig')
figure(6);
set(gca,'FontSize',12)
semilogy(tr.epoch,tr.perf,'LineWidth',2);
xlabel('Epochs','FontSize',12);
ylabel('Performance','FontSize',12);
xlim([0 length(tr.epoch)])
saveas(gcf,'mse_epoch.fig')

%% Generate simulink block
open('invpend_4D_NN')
gensim(net1,-1)
% take the block that is generated with this gensim command and then
% insert it in the place of the controller in the file called
% invpend_4D_NN.mdl
%% Execution phase
count = 0;
uncount = 0;
leng = 0;
% Execution time for each simulation
exec_time = 40;
% Pendulum angles
th0 = 0:pi/180:2*pi;
% Cart positions
pos0 = -0.5:0.1:0.5;
% Initial seeds for random number generators
rand('state',10);
rnd1 = rand(length(th0),1);
rand('state',20);
rnd2 = rand(length(th0),1);
% Demonstration sensor noise level
sgmnn = sgm;
%% Execution phase simulations
for i = 1:length(th0)
x0 = [th0(i) 0 0 0].';
% Initial seed for measurement noise in theta
seed_thetann = floor(1000*rnd1(i)*length(th0));
% Initial seed for measurement noise in thetadot
seed_xnn = floor(1000*rnd2(i)*length(th0)+1);
sim('invpend_4D_NN');
if abs(mean(cos(thetann(200:401))-1))<=0.05 && abs(mean(thetadotnn(200:401)))<=0.05 && abs(mean(xnn(200:401)))<=0.05 && abs(mean(xdotnn(200:401)))<=0.05
    % Update counter
    count = count + 1;
    % Theta for trajectories
    trajnn(:,2*count-1) = thetann;
    % Thetadot for trajectories
    trajnn(:,2*count) = thetadotnn;
    % Theta for trajectories
    posnn(:,2*count-1) = xnn;
    % Thetadot for trajectories
    posnn(:,2*count) = xdotnn;
    figure(7);
    set(gca,'FontSize',12)
    plot(trajnn(:,2*count-1),trajnn(:,2*count),'Color',[0 0 0])
    xlabel('pendulum angle','FontSize',12)
    ylabel('pendulum angular velocity','FontSize',12)
    title('Successful trajectories obtained in the execution phase','FontSize',12)
    hold on
    saveas(gcf,'exec_success.fig')
    figure(8);
    set(gca,'FontSize',12)
    plot(trajnn(:,2*count-1),trajnn(:,2*count),'-b')
    xlabel('pendulum angle','FontSize',12)
    ylabel('pendulum angular velocity','FontSize',12)
    title('Trajectories obtained in the execution phase','FontSize',12)
    hold on
    saveas(gcf,'exec_traj.fig')
else
    count = count;
    uncount = uncount + 1;
    % Theta for trajectories
    untrajnn(:,2*uncount-1) = thetann;
    % Thetadot for trajectories
    untrajnn(:,2*uncount) = thetadotnn;
    % Theta for trajectories
    unposnn(:,2*uncount-1) = xnn;
    % Thetadot for trajectories
    unposnn(:,2*uncount) = xdotnn;
    figure(8);
    set(gca,'FontSize',12)
    plot(untrajnn(:,2*uncount-1),untrajnn(:,2*uncount),'-r')
    xlabel('pendulum angle','FontSize',12)
    ylabel('pendulum angular velocity','FontSize',12)
    title('Trajectories obtained in the execution phase','FontSize',12)
    hold on
    saveas(gcf,'exec_traj.fig')
    figure(9);
    set(gca,'FontSize',12)
    plot(thetann,thetadotnn,'Color',[0 0 0])
    xlabel('pendulum angle','FontSize',12)
    ylabel('pendulum angular velocity','FontSize',12)
    title('Unsuccessful trajectories obtained in the execution phase','FontSize',12)
    hold on
saveas(gcf,'exec_unsuccess.fig')
end
% Total data length
total = leng + length(thetann);
% Execution angles in a vector
Fnn(:,(leng+1):total) = [thetann xnn].';
% Control inputs generated in the execution phase
Tnn(:,(leng+1):total) = inputnn.';
leng = total;
end
% Percentage of success rate in swinging up and balancing the pendulum
prcnt = count/length(th0)*100;
figure(10);
set(gca,'FontSize',12)
plot(Fnn(1,:));
xlabel('time index','FontSize',12)
ylabel('Execution pendulum angles','FontSize',12)
saveas(gcf,'exec_ang_time.fig')
figure(11);
set(gca,'FontSize',12)
plot(Fnn(2,:));
xlabel('time index','FontSize',12)
ylabel('Execution cart positions','FontSize',12)
saveas(gcf,'exec_pos_time.fig')

%% Input-output map when x = 0 and xdot = 0
% Pendulum angles
th = -pi/9:pi/72:2*pi+pi/9;
% Pendulum angular velocities
thdot = -8:0.1:8;
for i = 1:length(thdot)
    for j = 1:length(th)
        % Original controller input output behavior
        u = pl_inp4(th(j),thdot(i),0,0);
        % Approximated controller input output behavior
        unn = sim(net1,[th(j);thdot(i);0;0]);
        func(i,j) = u;
        func_nn(i,j) = unn;
    end
end
figure(12);
set(gca,'FontSize',12)
surf(th,thdot,func)
caxis([min(min(func)) max(max(func))])
colorbar;
caxis(5*[-1,1]);
view(0,90);
xlabel('theta','FontSize',12)
ylabel('thetadot','FontSize',12)
zlabel('controller output','FontSize',12)
title('Input-output map of the 2^n.d classical controller','FontSize',12)

% Trajectories recorded from demonstrations
hold on
for i = 1:n
    plot3(traj(:,2*i-1),traj(:,2*i),1000*ones(length(traj(:,2*i))),'Color',[0 0 0])
hold on
end
xlim(minmax(th))
ylim(minmax(thdot))
saveas(gcf,'io_cc_th_thdot_traj.fig')
figure(13);
set(gca,'FontSize',12)
surf(th,thdot,func_nn)
  caxis([min(min(func)) max(max(func))])
  caxis(max(max(abs(func_nn)))*[-1,1]);colorbar;
  colorbar;
  caxis(5*[-1,1]);
view(0,90);
xlabel('theta','FontSize',12)
ylabel('thetadot','FontSize',12)
zlabel('controller output','FontSize',12)
title('Input-output map of the 2^n^d neural network controller','FontSize',12)
xlim(minmax(th))
ylim(minmax(thdot))
saveas(gcf,'io_nn_th_thdot.fig')
figure(14);
set(gca,'FontSize',12)
surf(th,thdot,func_nn)
  caxis([min(min(func)) max(max(func))])
  caxis(max(max(abs(func_nn)))*[-1,1]);colorbar;
  colorbar;
  caxis(5*[-1,1])
view(0,90);
xlabel('theta','FontSize',12)
ylabel('thetadot','FontSize',12)
zlabel('controller output','FontSize',12)
title('Input-output map of the 2^n^d neural network controller','FontSize',12)
% Trajectories recorded from executions
hold on
for i = 1:count
  plot3(trajnn(:,2*i-1),trajnn(:,2*i),1000*ones(length(trajnn(:,2*i)),1),'Color',[0 0 0])
hold on
end
xlim(minmax(th))
ylim(minmax(thdot))
saveas(gcf,'io_nn_th_thdot_traj.fig')
A.4. Simulink Schemes

These are the Simulink model, subsystems and Embedded Matlab Functions which are used in the demonstration phase of chapter 5.1.

The Embedded Matlab functions for the controller and the inverted pendulum model used in the above schemes are below.

```matlab
function u = fcn(x)
if abs(mod(x(1),2*pi)-10*pi/180)<=10*pi/180
    thdes = 0;
    dthdes = 0;
    u = -40*(thdes-mod(x(1),2*pi))-10*(dthdes-x(2));
elseif abs(mod(x(1),2*pi)-350*pi/180)<=10*pi/180
    thdes = 2*pi;
    dthdes = 0;
    u = -40*(thdes-mod(x(1),2*pi))-10*(dthdes-x(2));
end
end
```

Figure A.4.1: Simulink model for 1st type of teacher

Figure A.4.2: Plant model for 1st type of teacher
else
    u = 2*sign(x(2));
end

function dxdt = f(x,u)
% Parameters of the inverted pendulum
g = 9.81;        % gravity
m = 0.1;         % mass of bar
M = 1;           % mass of cart
L = 1;           % length of bar
I = m*L^2/12;    % moment of inertia of bar wrt COG
l = L/2;
% dx/dt = f(x) + g(x)*u
Deltax = (I + m*l^2)*(M + m) - m^2*l^2*cos(x(1))^2;
fx = [x(2);
    ((M + m)*m*g*l*sin(x(1)) - m^2*l^2*x(2)^2*sin(x(1))*cos(x(1)))/Deltax;
    x(4);
    (-m^2*l^2*g*sin(x(1))*cos(x(1)) + (I + m*l^2)*m*l*x(2)^2*sin(x(1)))/Deltax];
gx = [0;
    -m*l*cos(x(1))/Deltax;
    0;
    (I + m*l^2)/Deltax];
dxdt = fx + gx.*u;

This is the Simulink model which is used in the execution phase of chapter 5.1.

![Simulink model for execution phase of 1st type of teacher](image)
These are the Simulink model, subsystems and Embedded Matlab Functions which are used in the demonstration phase of chapter 5.2.

![Simulink model for 2nd type of teacher](image1)

Figure A.4.4  Simulink model for 2\textsuperscript{nd} type of teacher

![Plant model for 2nd type of teacher](image2)

Figure A.4.5 Plant model for 2\textsuperscript{nd} type of teacher

The Embedded Matlab functions for the controller and the inverted pendulum model used in the above schemes are below.

```matlab
function [u,E,phi_t] = inp(x)
x(1) = mod(x(1),2*pi);
```

69
% Parameters of the inverted pendulum
g = 9.81;       % gravity
m = 0.1;        % mass of bar
M = 1;          % mass of cart
L = 1;          % length of bar
I = m*L^2/12;   % moment of inertia of bar wrt COG
l = L/2;

Deltax = (I + m*l^2)*(M + m) - m^2*l^2*cos(x(1))^2;
f4x = (-m^2*l^2*g*sin(x(1))*cos(x(1)) + (I + m*l^2)*m*l*x(2)^2*sin(x(1)))/Deltax;
g4x = (I + m*l^2)/Deltax;

% Mechanical energy of pendulum
E = 0.5*(I + m*l^2)*x(2)^2 + m*g*l*cos(x(1));
Eref = m*g*l;

% New controller-v(t) parameters
wn = sqrt(m*g*l/(I+m*l^2));
a0 = 0.3;
b0 = 0.4;
c0 = 0.9;
zeta = 1.5;
g_jwn = 1/sqrt((1-c0^2)^2+(2*zeta*c0)^2);
phi_jwn = atan2((2*zeta*c0),(1-c0^2));

% Sinusoidal reference input to x
phi_t = -atan2(x(2)/wn,-pi+x(1));

if abs(E-Eref)>=b0;
a = a0*sign(E-Eref);
else
    a = a0*(E-Eref)/b0;
end
xref = (a/g_jwn)*sin(phi_t - pi + phi_jwn);

% The linear control input
v = -2*zeta*(wn/c0)*x(4) + (wn/c0)^2*(xref - x(3));

% The control input to the plant
% The input force
% Switching constants
eps1 = 0.1;
eps2 = cos(10*pi/180);
if abs(E-Eref)<eps1 && cos(x(1))>eps2 && x(1)<0.5*pi
    thdes = 0;
dthdes = 0;
xdes = 0;
dxdes = 0;
% LQR control coefficients
    u = [-107.909,-27.9547,-31.6228,-25.2215]*([thdes dthdes xdes dxdes]).'-x);
elseif abs(E-Eref)<eps1 && cos(x(1))>eps2 && x(1)>1.5*pi
    thdes = 2*pi;
dthdes = 0;
xdes = 0;
dxdes = 0;
% LQR control coefficients
    u = [-107.909,-27.9547,-31.6228,-25.2215]*([thdes dthdes xdes dxdes]).'-x);
else
    u = inv(g4x)*(v - f4x);
end

function dxdt = f(x,u)
% Parameters of the inverted pendulum
g = 9.81;       % gravity
\[ m = 0.1; \quad \% \text{mass of bar} \]
\[ M = 1; \quad \% \text{mass of cart} \]
\[ L = 1; \quad \% \text{length of bar} \]
\[ I = mL^2/12; \quad \% \text{moment of inertia of bar wrt COG} \]
\[ l = L/2; \]
\[ \% \text{dx/dt} = f(x) + g(x)u \]
\[ \text{Deltax} = (I + m^2l^2)(M + m) - m^2l^2\cos(x^2)^2; \]
\[ \text{fx} = [x(2); \]
\[ ((M + m)m^2g^2l^2\sin(x(1)) - m^2l^2x(2)^2\sin(x(1))\cos(x(1)))/\text{Deltax}; \]
\[ x(4); \]
\[ (-m^2l^2g^2\sin(x(1))\cos(x(1))) + (I + m^2l^2)m^2l^2x(2)^2\sin(x(1))) / \text{Deltax}; \]
\[ \text{gx} = [0; \]
\[ -m^2l^2\cos(x(1))/\text{Deltax}; \]
\[ 0; \]
\[ (I + m^2l^2)/\text{Deltax}; \]
\[ \text{dxdt} = \text{fx} + \text{gx}.^u; \]

These are the Simulink models and subsystems used in the execution phase of chapter 5.2.
Figure A.4.8 1\textsuperscript{st} layer of Neural Network Controllers

Figure A.4.9 2\textsuperscript{nd} layer of Neural Network Controllers
B. References