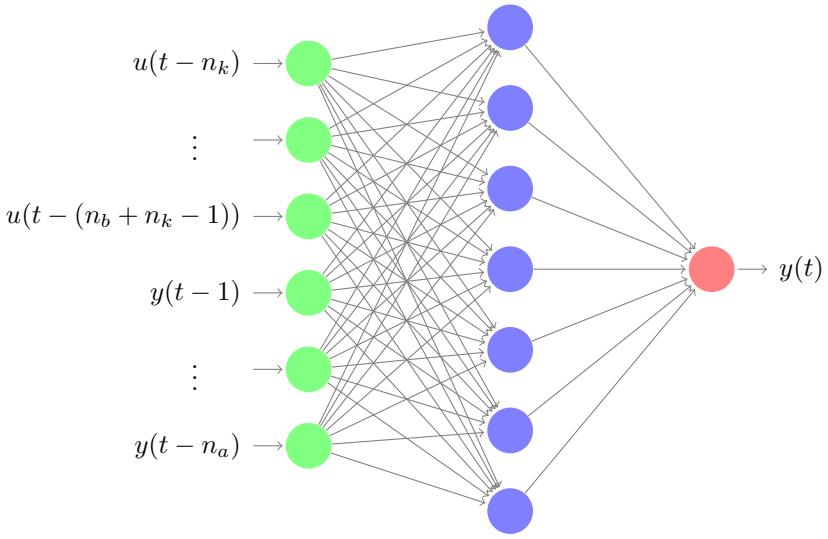
Nonlinear System Identification and Machine Learning

This version: 2023-10-12



REGLERTEKNIK
AUTOMATIC CONTROL
LINKÖPING

Name:	
P-number:	
Date:	
Passed:	

Chapter 1

Introduction

The purpose of this lab is to model the two-cart system shown in Figure 1.1. The carts are assumed to operate in a confined space with elastic walls. This elasticity is only relevant when the carts are in contact with the walls and as a consequence the system has switching characteristics depending on the carts' positions, $x_1(t)$ and $x_2(t)$. A greybox approach can readily be taken for modelling the system but in this lab the goal is primarily to explore nonlinear blackbox-modelling techniques. Recently there has been a wide interest in the research area of machine learning. Since that field also deals with estimating (or learning) models, there are connections to the techniques for modelling of dynamic systems which have been developed within the area of automatic control. For example, commonly used deep neural network structures such as feed-forward and cascade-forward networks can be viewed as nonlinear ARX (NLARX) models, where as what is called recurrent neural networks can be viewed as nonlinear OE (NLOE) models. In this lab, some of the connections between conventional system identification and machine learning are pointed out. The lab is carried out in simulation by use of MATLAB and performing the lab requires the System Identification Toolbox and the Deep Learning Toolbox.

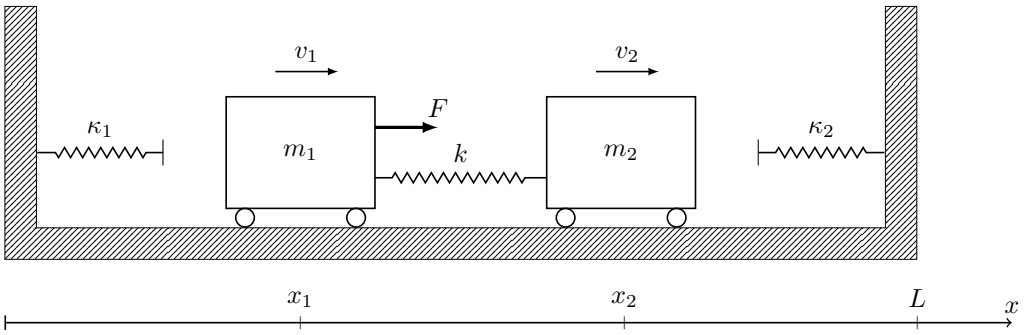


Figure 1.1: The studied two-cart system.

Chapter 2

Preparation

The questions below, and all questions in the document marked as **Preparation** must be done before attending the lab. Note that there are additional preparation exercises in Chapter 3.

Solutions to all questions should be available upon request from the lab assistant.

The time spent during the scheduled lab session is only a small part of the complete lab, as a major part is spent on the theoretical material during preparations. When the lab starts, it is assumed that you have done all preparations and have a clear idea of the tasks that will be performed during the lab.

Preparation 1 *Read Sections 14.2, 11.4, 12.6 and 15.2-3 in the course book by Ljung, Glad & Hansson.*

Preparation 2 *Revisit "Computer Exercises in System Identification - Part 4" and solve the first exercise.*

Preparation 3 *Read the complete lab-pm. There are some theoretical questions in the pm which are supposed to be completed as preparation.*

Chapter 3

The lab

In this lab we will try to find models for the two-cart system presented above. This will first be done by a greybox modelling approach and then by use of nonlinear blackbox methods.

Items labeled **Preparation** are questions you are supposed to solve and fill out before attending the lab.

Items labeled **Task** are questions you solve when attending the lab.

3.1 Modelling the system

We will first try to get a physical understanding for the two-cart system. As depicted in Figure 1.1, the carts are assumed to operate in a confined space with elastic walls. This elasticity is only relevant when the carts are in contact with the walls and we will assume that the walls' thicknesses are l_1 and l_2, respectively, such that

$$F_{\kappa_1}(x_1(t)) = \begin{cases} \kappa_1(l_1 - x_1(t)) & \text{if } x_1(t) < l_1, \\ 0 & \text{otherwise,} \end{cases} \tag{3.1}$$

$$F_{\kappa_2}(x_2(t)) = \begin{cases} \kappa_2(L - l_2 - x_2(t)) & \text{if } x_2(t) > L - l_2, \\ 0 & \text{otherwise,} \end{cases} \tag{3.2}$$

where κ_1 and κ_2 can be viewed as regular linear spring constants. Note that the signs of $F_{\kappa_1}(t)$ and $F_{\kappa_2}(t)$ depend on how the free-body diagram is drawn. In addition to the elastic walls the carts are connected by a regular linear spring, k, which we assume is at rest when $x_2(t) - x_1(t) = l_k$. The system is actuated with the external force, $F(t)$. The left and right carts are assumed to have masses m_1 and m_2 and to travel with velocities $v_1(t)$ and $v_2(t)$, respectively. Each cart is also subject to a rolling-resistance force proportional to its own speed. We denote the friction coefficients by μ_1 and μ_2.

Preparation 4 *Denote the state vector by* $\boldsymbol{x}(t) = \begin{bmatrix} x_1(t) & x_2(t) & v_1(t) & v_2(t) \end{bmatrix}^{\mathrm{T}}$, *the input by* $u(t) = \mathrm{F}(t)$ *and cast the two-cart system on state-space form*

$$\dot{\boldsymbol{x}}(t) = f\left(\boldsymbol{x}(t), u(t)\right). \tag{3.3}$$

Hint: *You can assume that* $0 < x_1 < x_2 < \mathrm{L}$, *i.e. that the springs are sufficiently stiff.*

For identifying this system it would be ideal to have independent measurements of all the 4 states. However, in practice it is often the case that only parts of the state vector can be measured directly. Here we will assume that the carts' positions are measured.

On the course webpage two datasets collected using the two-cart system can be found which are called `z_dist` and `z_undist`, respectively. The first of these were collected under presence of additive measurement noise, whereas the second dataset was collected in undisturbed conditions. Having access to undisturbed data is possible because these experiments were performed in simulation.

Task 1 (Create a greybox model) *Implement your derived state-space model structure as an* ***idnlgrey*** *object in* MATLAB *using the same syntax as you did in Preparation 2. You can use* $m_1 = 0.025$, $m_2 = 0.045$, $\kappa_1 = 65$, $\kappa_2 = 75$, $k = 55$, $\mu_1 = 4.5$, $\mu_2 = 4.5$, $l_1 = l_2 = l_k = 2.5$ *and* $\mathrm{L} = 10$ *as initial guesses for the parameters. It is often useful to specify the initial values of the states, this can be done using the syntax*
`m0 = idnlgrey('MODFILENAME',[ny nu nx],theta0,x0);`.

To validate you model, run the function `validate_model(m0)` *found on the course webpage.*

Note: *Make sure to use the initial state* `x0` *of the disturbed data. The cart velocities can be set to zero.*

Task 2 (Refine the estimates of the greybox model) *What values for the parameters do you get after refining the model using* `pem(.)` *with the disturbed data as estimation data? Are the estimates reasonable from a physical point of view? Validate the model using* `compare(.)` *with the undisturbed data as validation data.*

Note: *Since the initial values are not the same in the two datasets, the initial values of the model must be updated for the validation. This can be done with the command* `m = setinit(m, 'Value', {zv.y(1,1), zv.y(1,2), 0, 0});`.

Task 3 (Uncertain initial guesses for greybox estimation) *Since the state-space representation is nonlinear it is even more important to have good initial guesses for the parameters. Try corrupting your initial guesses and re-estimate the model. Are the results still good? See how much you can shift the initial guesses and still obtain a decent model.*

3.1.1 Discretization

The above derived system representations is in continuous time. For many applications this is a natural first step, because most known basic relationships are expressed in terms of differential equations. However, the only way to observe a system is by measurements which are generally obtained as a finite collection of values, *i.e.* in discrete time. There are many ways to transform a continuous-time system representation to a discrete-time approximation. The simplest is perhaps Euler's explicit method. This method is based on a finite-difference approximation of the derivative

$$\dot{\mathbf{x}}(k\mathrm{T}_s) \approx \frac{1}{\mathrm{T}_s}\left(\mathbf{x}(k\mathrm{T}_s + \mathrm{T}_s) - \mathbf{x}(k\mathrm{T}_s)\right),\ k = 1,\ 2,\ \ldots. \tag{3.4}$$

Here T_s is the time difference between two consecutive samples and $k\mathrm{T}_s$ denotes the measurement sampling instants.

Preparation 5 *Transform your derived continuous-time system representation to a discrete-time approximation*

$$\boldsymbol{x}(k\mathrm{T}_s + \mathrm{T}_s) = \tilde{f}\left(\boldsymbol{x}(k\mathrm{T}_s), u(k\mathrm{T}_s)\right)$$

using Euler's explicit method.

Hint: *You can assume that* $\mathrm{T}_s = 1$ *for simplified notation.*

Preparation 6 *From Preparation 5, substitute the nonmeasurable states to describe the system using only the measurable states and inputs.*

Preparation 7 *Based on Preparation 6, what values for* n_a, n_b *and* n_k *do you think would be suitable for a (nonlinear) ARX model of this system?*

Hint: *Since we have multiple outputs, the values* n_a, n_b *and* n_k *will be matrices.*

3.2 Artificial neural networks

As we have seen before, a model is a mapping from past data to the space of the output. In the nonlinear case, this mapping has the general structure

$$\hat{y}(t|\theta) = g(\varphi(t), \theta), \tag{3.5}$$

where $\varphi(t)$ as before denotes the regression vector. If we do not have particular insights into the system's properties, we should seek parameterizations of g that are flexible and cover most kinds of system behavior. This would give us a nonlinear counterpart to the linear blackbox structures that we have worked with before (ARX, OE, ARMAX, BJ). In the nonlinear case there is however an even larger number of possible parameterizations to consider. One class of model structures that has a quite long history but has gotten a new wave of attention recently, much thanks to the wide interest in machine learning, is the artificial neural network.

3.2.1 Network representation

Artificial neural networks have the capability of arbitrarily accurately approximating any reasonable function and have been successfully used in many applications. An artificial neural network can be described by

$$\hat{y}(t|\theta) = \sum_{\ell=1}^{d_1} \gamma_\ell \kappa\left(\alpha_\ell \varphi(t) - \beta_\ell\right), \tag{3.6}$$

where the model parameters, θ, to be identified is the set of $\gamma_\ell \in \mathbb{R}$, $\alpha_\ell \in \mathbb{R}^{1\times p}$, $\beta_\ell \in \mathbb{R}$, for $\ell = 1, ..., d_1$, where p is the length of the regression vector, $\varphi(t)$, and $\kappa(\cdot)$ is an activation function. Graphical illustrations of (3.6) look like a network where d_1 is the number of (blue) neurons in Figure 3.1.

As depicted in the figure, the neural network (3.6) consists of three layers. The left layer includes the model inputs and in the right layer the model output is formed. The layer in between the input and output layers is called the hidden layer. Since there is only one hidden layer, the mapping (3.6) is referred to as a single-hidden-layer feedforward neural network. The terms $\kappa\left(\alpha_\ell \varphi(t) - \beta_\ell\right)$ are the outputs of the hidden units (neurons). The middle layer is referred to as hidden because the outputs of the neurons are latent signals which a user of the network never sees, similar to the state vector in a state-space representation.

Note that the output (3.6) of a single hidden layer network can also be written $\hat{y}(t|\theta) = C\kappa(A_1\varphi(t) + b_1)$, where $C = \left[\gamma_1, ..., \gamma_{d_1}\right]$, and $\kappa(\cdot)$ is applied element-wise to the vector $A_1\varphi(t) + b_1$. The matrix $A_1 \in \mathbb{R}^{d_1\times p}$ is constructed by stacking the α_ℓ:s, and $b_1 \in \mathbb{R}^{d_1}$ is constructed by stacking the β_ℓ:s.

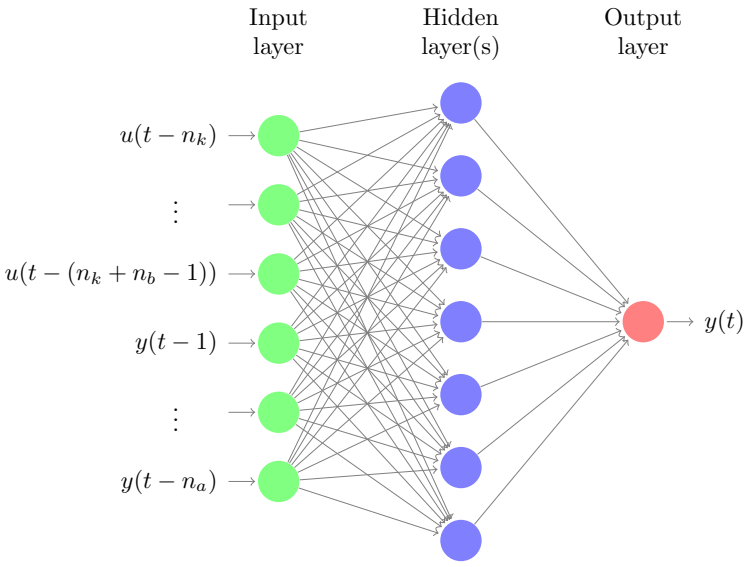


Figure 3.1: A graphical illustration of a feedforward neural network.

3.2.2 Deep learning

Networks with multiple hidden layers are referred to as deep networks and parameter estimation for deep networks is known as deep learning. In this case the neuron outputs of each layer are fed as input to the subsequent layer, such that

$$\mathbf{F}_1(t) = \kappa(\mathrm{A}_1\varphi(t) + b_1), \tag{3.7a}$$
$$\mathbf{F}_2(t) = \kappa(\mathrm{A}_2\mathbf{F}_1(t) + b_2), \tag{3.7b}$$
$$\vdots$$
$$\mathbf{F}_{\mathrm{N_{HL}}}(t) = \kappa(\mathrm{A}_{\mathrm{N_{HL}}}\mathbf{F}_{\mathrm{N_{HL}}-1}(t) + b_{\mathrm{N_{HL}}}), \tag{3.7c}$$
$$\hat{y}(t|\theta) = \mathrm{C}\mathbf{F}_{\mathrm{N_{HL}}}(t). \tag{3.7d}$$

where $\mathbf{F}_i(t)$, $i = 1, \ldots, \mathrm{N_{HL}}$ is the vector of neuron outputs from layer i. The A_i:s and b_i:s are constructed in the same way as A_1 and b_1. Expanding the network with additional hidden layers gives more parameters to estimate but at the same time it makes the model structure more flexible. In many applications, for example image processing and speech recognition, it has turned out to be very beneficial to consider deep networks.

The expansion (3.7) is called a feedforward neural network. A variant of the feedforward net is obtained when all previous hidden units are used as regressors in the next layer, not only the ones from the previous layer. This type of structure is called a cascade network. Both feedforward and cascade networks are examples of NLARX models.

The NLARX models allow a very flexible dependence of the predicted output on the regression vector $\varphi(t)$, but these mappings are static functions from the input $u(t)$ to the output $\hat{y}(t|\theta)$ and do not explicitly involve time t. As in the linear case, by including

multiple lags of the input and the output in $\varphi(t)$ it is possible to describe dynamic systems with NLARX models. However, to describe systems with slowly decaying impulse responses it is necessary to have a very high model order.

Recall from before that for linear output error (OE) models, lags of the predicted output $\hat{y}(t-1)\dots\hat{y}(t-n_b)$ are used in the predictor. The nonlinear OE (NLOE) versions of the above presented model structures are in machine learning nomenclature referred to as recurrent neural networks. Notably, for deep networks it is not only possible to include lags of the predicted output in the regression vector but also to include lags of the neuron outputs from the hidden layers. Recurrent networks thereby offer freedom to define very flexible model structures. At the same time, the parameters of a recurrent neural network are technically difficult to estimate. This is because the computations of gradients w.r.t. θ are challenging and the feedback around the nonlinear system may cause instability in the calculations.

3.2.3 Activation function

In artificial neural networks each neuron is associated with an activation function, here denoted by $\kappa(z)$. As seen in (3.6), the activation function of a neuron defines the output of that neuron given an input or set of inputs. There are many different activation functions that can be used and subsequently some of the most common ones are listed. The simplest activation function is the purely linear one

$$\kappa(z) = z. \tag{3.8}$$

This activation function is rarely used in the hidden layers but is frequently used for the rightmost layer where the predicted output is formed.

Two often used activation functions are the logistic (sigmoid) function

$$\kappa(z) = \frac{1}{1+e^{z}}, \tag{3.9}$$

and the hyperbolic tangent function

$$\kappa(z) = \frac{\sinh(z)}{\cosh(z)} = \frac{e^{z}-e^{-z}}{e^{z}+e^{-z}}. \tag{3.10}$$

Notably, the hyperbolic tangent function can be viewed as a rescaling of the logistic function, such that the output also ranges to negative numbers.

All the three previously mentioned activation functions are continuously differentiable. This is a very nice property to have when search methods are applied for finding the network parameters. However, the perhaps most common activation function is the rectified linear unit (ReLU)

$$\kappa(z) = \max(z, 0), \tag{3.11}$$

which is non-differentiable in the origin. This function picks the maximum between the input and 0 and is unlike the sigmoid and the hyperbolic tanget functions, able to give outputs in the whole interval $[0,\infty)$. The ReLU function makes the sum (3.6) piecewise linear with breakpoints in β_ℓ for $\ell = 1, \ldots, d_1$.

3.2.4 Deep learning in MATLAB

A major part of this lab can essentially be carried out by only using the Deep Learning Toolbox. The Deep Learning Toolbox is however more flexible than necessary and provides the user with quite many extra design choices. Also, it does not only support time-series modelling but also models of other types like image processing and speech recognition. Direct use of the Deep Learning Toolbox therefore requires some extra work. One way of interacting with it is to use the System Identification Toolbox as a wrapper. Doing so is convenient because it enables use of **iddata** objects, the conventional model-order representation with n_a, n_b and n_k, as well as functions such as `compare` and `resid` for model validation, which we have used previously throughout this course.

The System Identificaiton Toolbox in itself supports estimation of neural networks with a limited set of differentiable activation functions and a single hidden layer. For example, a sigmoid neural network with 10 neurons can be estimated as

```
opt = nlarxOptions('SearchMethod','lm');
m10 = nlarx(data, [na, nb, nk], idSigmoidNetwork(10),opt);
```

Here '**lm**' is the the Levenberg-Marquardt algorithm which is a common way to solve non-linear least squares problems. It is also known as damped least squares.

To estimate deep neural networks or networks with other activation functions, the System Identification Toolbox can be used as a wrapper to make calls to the Deep Learning Toolbox. A neural network model can in this way be estimated by first defining a **net estimator** and then applying the `nlarx` command

```
net_estimator = idFeedforwardNetwork(feedforwardnet(d));
Mnet = nlarx(data,[na nb nk],net_estimator);
```

Here the argument **d** which is passed to `feedforwardnet` is the number of neurons used in the network. If a network with multiple hidden layers is desired, the **d** argument can be passed as a vector containing the number of neurons in each layer.

Similarly, a cascade network can be estimated as

```
net_estimator = idFeedforwardNetwork(cascadeforwardnet(d));
```

```
Mnet = nlarx(data,[na nb nk],net_estimator);
```

The default activation function if networks are estimated in this way is the hyperbolic tangent function. If a network with rectified linear units is desired, the activation function of a specific layer, i, can be modified before the **net estimator** is defined as

```
net = feedforwardnet(d);
net.layers{i}.transferFcn = 'poslin';
net_estimator = idFeedforwardNetwork(net);
```

In the same way, the activation function can be set to *e.g.* '**purelin**', '**logsig**' or '**tansig**'. It is also possible to specify a custom-made activation function.

3.2.5 Bias/variance trade-off

The error in an estimated model has two components. The first component is the bias, which is the distance between the true system and the best model available in the set of candidate models that the search is carried out over. The second component is called the variance and is the distance between the best model available in the model set and the model that is actually obtained. Unlike the bias, the variance will depend on noise and other imperfections in the data. As the complexity/flexibility of the model set increases, the bias will decrease and the variance will increase. This means that in order to have a small total error, we want a model of sufficient flexibility with as few free parameters as possible.

Assume that we skipped the physical modelling in Section 3.1 and instead directly applied a neural network model structure. In this case it is necessary to consider a network with high complexity if we want to be certain that the true system is contained in the model set.

Task 4 (Estimation using flexible model structure) *Try estimating a couple of different (single-hidden-layer) sigmoid networks with 10-50 neurons, using the disturbed data for estimation. Compare the model fit with the one obtained using the greybox model structure studied before. Primarily consider the undisturbed data as validation data but also compare model fit with respect to the disturbed data.*

Both system identificaiton and machine learning algorithms estimate models based on finite sets of estimation data. During this estimation, the model is evaluated based on

how well it predicts the observations contained in the estimation datasets. However, the goal is usually to produce a model that generalizes, *i.e.* that predicts previously unseen observations. This is the reason that a separate dataset for validation is commonly used. Overfitting occurs when a model fits the data in the estimation set well, while showing a larger error on a set of previously unseen validation data.

Regularization, in the context of system identification and machine learning, refers to the modification of an estimation algorithm done in order to prevent overfitting. This often involves enforcing some sort of smoothness constraint on the model. The smoothness can be imposed explicitly, by fixing the number of parameters in the model, or by augmenting the cost function as in Tikhonov regularization like we saw in one of the teaching sessions. Recall that Tikhonov regularization can be applied using the System Identificaion Toolbox with the commands

```
opt = nlarxOptions('SearchMethod','lm');
opt.Regularization.Lambda = Lambda;
m = nlarx(data, orders, idSigmoidNetwork(d), opt);
```

Task 5 (Regularization for neural networks) *Pick a sigmoid network structure with a large number of neurons which results in poor fit to validation data and try including a Tikhonov regularization term during estimation. How does this affect the fit to estimation/validation data?*

When estimating the parameters of nonlinear model structure, it is common to apply a gradient descent method. Then, each iteration updates an approximate solution to the estimation problem by taking a step in the direction of the gradient of the objective function. By choosing the step-size appropriately, such a method can be made to converge to a local minimum of the objective function. For each iteration the fit to estimation data will increase. However, as we know a higher fit to estimation data does not necessarily imply a higher fit to validation data. Another common way to regularize the estimation procedure is therefore to apply what is called early stopping. This is done by putting a restriction on the maximum number of iterations and can be done in the System Identificaion Toolbox using the commands

```
opt = nlarxOptions('SearchMethod','lm');
opt.SearchOptions.MaxIterations = maxIt;
m = nlarx(data, orders, idSigmoidNetwork(d), opt);
```

Task 6 (Early stopping) *Try applying early stopping with different values of* ***maxIt*** *when*

estimating your sigmoid networks. How does this affect the fit to estimation/validation data?

Preparation 8 *If you consider the activation functions in Section 3.2.3, is there anyone in particular that you find suitable for describing the nonlinear spring forces? How many neurons with the chosen activation function would be required in a single-hidden-layer network to describe the nonlinear spring forces?*

Preparation 9 (An ideal model structure) *Assume that rectified linear units (ReLUs) are used as activation functions in a single-hidden-layer network, think about how many neurons that would be needed for obtaining a model structure which is able to exactly describe the true system .*

Task 7 *Estimate the parameters of the model structure based on your intuition from Preparation 9. Make sure to use the ReLU activation function in the network (see Section 3.2.4). What number of neurons gives best model fit? Compare the resulting model's fit to data with the ones obtained before. Do you need regularization in this case?*

3.2.6 Recurent Neural Networks (Optional)

As a final thing we will use recurrent neural networks for describing the two-cart system. As mentioned before, estimation of recurrent neural networks is a bit more challenging. Doing so therefore requires a direct call to the Deep Learning Toolbox. In order to do so the data must first be transformed to a cell-array format. From an **iddata**-representation, this can be done as

```
XTrain=cell(1,length(data.u));
YTrain=cell(1,length(data.y));
for j = 1:length(data.u)
XTrain{j} = data.u(j); % Inputs
YTrain{j} = data.y(j,:)'; % Targets
end
```

The network can then be estimated with the series of commands

```
net = layrecnet(layerDelays, hiddenLayerSize);
[inputs,inputStates,layerStates,targets] = preparets(net,XTrain,YTrain);
[model,tr] = train(net,inputs,targets,inputStates,layerStates);
```

Here layerDelays decides how many lags of the simulated output that are used in the predictor. In order to use $\hat{y}(t-1|\theta)$ and $\hat{y}(t-2|\theta)$ in the predictor we should specify

```
layerDelays = 1:2;
```

As before, we can choose rectified linear units as activation functions with

```
net.layers{1}.transferFcn = 'poslin'
```

In order to check the fit between the model response and the validation data, a simulation can be carried out as

```
[val_inputs,val_inputStates,val_layerStates,val_targets] = ...
preparets(model,XVal,YVal);
val_outputs = model(val_inputs,val_inputStates,val_layerStates);
tmp = cell2mat(val_outputs)';
yh = [tmp(1, 1:2)'.*ones(net.numLayerDelays,2); tmp];
```

The model fit is most easily computed by `fit = 100*(1-goodnessOfFit(yh,y0,'NRMSE'))`.

Task 8 (Optional) *Estimate a couple of recurrent neural networks. How well do these networks perform in comparison to the feedforward networks found before?*