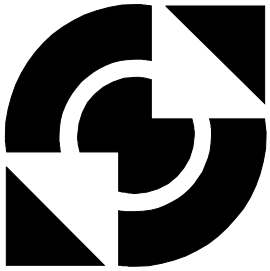
University of Twente

Faculty of Electrical Engineering,
Mathematics, and Computer Science



Comparison of Learning Methods in the Learning Feed-forward Control Setting

Zulkifli Hidayat

M.Sc. Thesis

Supervisors: prof. dr. ir. J. van Amerongen
dr. ir. T.J.A. de Vries
ir. B.J. de Kruif
ir. R. Radzim

January 2004
Report Number
003CE2004

Control Engineering
Department of Electrical Engineering

University of Twente
P.O. Box 217
NL 7500 AE Enschede
The Netherlands

Summary

This thesis details the comparison between several learning methods that are to be used in the Learning Feed-forward Control (LFFC) setting. These learning methods consist of the Multilayer Perceptron (MLP), the Radial-Basis Function Network (RBFN), the Support Vector Machine (SVM), the Least Square Support Vector Machine (LSSVM), and the Key Sample Machine (KSM).

The comparison is performed by simulations with respect to reproducibility - the ability to train various training sets from the same function, high dimensional inputs - the ability to train samples from a high dimensional function, and real-world data of a linear motor - the ability to train samples from the motion of a linear motor.

Simulation results show that there is no best method. The reproducibility comparison shows that the RBFN from visual design is the best as it gives the smallest variance. The mean square error of the high dimensional inputs comparison shows that the KSM is the best. From approximation of linear motor data, the KSM also gives the best result with respect to mean square error.

Acknoledgement

Alhamdulillaahi rabbil álaamiin, this thesis is finally finished. I would like to thank to people who have helped me directly or indirectly to finish this thesis. They are:

- prof. Amerongen, who supervised me and provided facilities to work in the Control Engineering Group and also some useful advices.
- Theo, for supervision and critical comments.
- Bas, from whom the idea of the this thesis came, for his supervision and comments to my results during my working for the thesis.
- Richie, who made this report much more readable linguistically and also gave a lot of suggestions during writing this report.
- The last but not the least, my parents, who always give support and courage during my life in this world. For them I dedicated this thesis.

I also want to thank to all my friends here, in PPI and especially in IMEA, with whom I share my life, my happiness and sadness. Those make life much better in Enschede. Thanks for all of you.

Enschede, end of January 2004

Contents

Summary i

Acknoledgement iii

1 Introduction 1
- 1.1 Motivation . . . 2
- 1.2 Problem Definition . . . 2
- 1.3 Report Structure . . . 3

2 Learning Methods 5
- 2.1 Introduction . . . 5
- 2.2 Artificial Neural Networks . . . 5
- 2.3 Function Approximation . . . 7
- 2.4 Multilayer Perceptron . . . 7
- 2.5 Radial-Basis Function Networks . . . 11
- 2.6 Support Vector Machine . . . 16
- 2.7 Least Square Support Vector Machine . . . 19
- 2.8 Key Sample Machine . . . 20
- 2.9 Theoretical Comparison of Learning Methods . . . 22
- 2.10 Summary . . . 24

3 Learning Method Comparison by Simulation 25
- 3.1 Introduction . . . 25
- 3.2 Reproducibility Simulation . . . 26
- 3.3 High Dimensional Inputs Simulation . . . 34
- 3.4 Off-line Approximation of Linear Motor . . . 40
- 3.5 Summary . . . 42

4 Conclusions and Recommendations 43
- 4.1 Conclusions . . . 43
- 4.2 Recommendations . . . 44

List of Figures

1.1 Block diagram of on-line FEL control 2

2.1 Artificial neural network 6
2.2 Model of a neuron with its inputs 6
2.3 Signal-flow graph highlighting the details of the output neuron 8
2.4 Signal-flow graph highlighting the details of the output neuron connected to the hidden neuron j 10
2.5 Plot of activation functions in the MLP 11
2.6 Plot of RBFs 12
2.7 SVM for Function Approximation 17
2.8 SVM Structure for Function Approximation 17

3.1 Approximated function in reproducibility simulation 26
3.2 Different initial weights reproducibility for the MLP 28
3.3 Reproducibility simulation result of the MLP 29
3.4 Reproducibility simulation result of the RBFN 29
3.5 Reproducibility simulation result of the SVM 31
3.6 Reproducibility simulation result of the LSSVM 31
3.7 Reproducibility simulation result of the KSM 33
3.8 High dimensional inputs simulation results of the MLP and the RBFN 35
3.9 High dimensional inputs simulation results of the SVM, the LSSVM, and the KSM 36
3.10 Working principle of the linear motor 41
3.11 Plot of the part of training samples, target vs position and speed reference 41

List of Tables

3.1 Number of support vectors of the reproducibility simulation of the SVM 30
3.2 Number of support vectors of the reproducibility simulation of the KSM 32
3.3 Summary of the MSE for the reproducibility simulations 33
3.4 Summary of the number for support vectors of the reproducibility simulations 33
3.5 High dimensional inputs simulation results of the MLP 36
3.6 High dimensional inputs simulation results of the RBFN 37
3.7 High dimensional inputs simulation results of the SVM 37
3.8 High dimensional inputs simulation results of the LSSVM 38
3.9 High dimensional inputs simulation results of the KSM 38
3.10 Approximation result of linear motor 41

Chapter 1

Introduction

Control problems normally involve designing a controller for a plant based on a model of the plant. This model is in the form of one or more mathematical equations that describe the dynamic behaviour of the plant. A requirement of this model is that it is competent in the problem context. Model competency enables achieving a good working control system according to desired specifications.

A model of a plant can be obtained by the process of system identification. One approach of system identification involves analysing all elements that compose the plant and then deriving a model based on the individual elements. Another approach is assuming the plant as a black-box and then estimating model parameters from input-output data. This approach can be used when there is knowledge of the plant dynamic structure and external disturbances on the plant are available, but the values of some parameters cannot be determined. When a competent model is difficult to obtain by conventional methods, for example, the model is so complex that parameters and/or structure cannot be determined, the use of learning control may be considered (Kawato et al., 1987, de Vries et al., 2000).

Learning control is a method of control that applies learning. In systems that implement learning control, there is an element that learns to compensate for system dynamics. Based on how this learning element is used, there exist two types of learning control (Ng, 1997):

- *Direct learning control*. The learning element is employed to generate the control signal.

- *Indirect learning control*. The learning element is used to learn the model of the plant. Based on the resulting model, a controller is designed.

One of the schemes of learning control is Learning Feed-forward Control (LFFC), which is based on Feedback Error Learning (FEL) (de Vries et al., 2000). FEL consists of two parts (Velthuis, 2000):

- *Feedback controller*. Determines the minimum tracking at the beginning of learning and compensates for random disturbances.

- *Feed-forward controller*. Maps the reference signal and its derivatives to generate part of the control signal. When it is equal to the inverse of the plant, the output of the plant will follow the reference.

A block diagram of on-line FEL is shown in Figure 1.1. r is the reference signal, $\dot{r}$ to $r^{(n)}$ are the reference signal derivatives, C is the controller, and P is the plant. The controller C can be designed using methods available in literature. The feed-forward controller in this scheme is designed as a function approximator that tries to imitate the inverse plant P. Signals that are used to train the approximator are the reference signal and its derivatives ($\dot{r}$ to $r^{(n)}$) as the inputs, and the output of feedback controller C as the target.

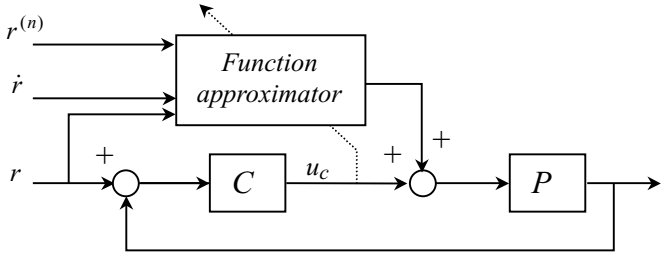


Figure 1.1: Block diagram of on-line FEL control

1.1 Motivation

Until recently there have been a number of function approximators developed. Some of them are discussed in this thesis, namely the Multilayer Perceptron (MLP), the Radial-Basis Function Network (RBFN), the Support Vector Machine (SVM), the Least Square Support Vector Machine (LSSVM), and the Key Sample Machine (KSM). Except the KSM, these methods are intended to solve broad spectrum problems (Haykin, 1999, Cristianini and Shawe-Taylor, 2000), for example classification, image processing, and regression/function approximation. Only that the KSM is firstly developed to be used in the learning feed-forward control setting (de Kruif, 2003).

In general, no method is superior for all applications. Each has its advantages and disadvantages when applied to a certain problem. This also applies to learning control. Velthuis (2000) mentions the characteristics of a function approximator that are suitable for use in control. These characteristics are:

- Small memory requirement in implementation
- Fast calculation for input-output adaptation and prediction
- Convergence of the learning mechanism should be fast
- The learning mechanism should not suffer from local minima
- The input-output relation must be locally adaptable
- Good generalising ability
- The smoothness of the approximation should be controllable

1.2 Problem Definition

Based on the fact that every learning method has advantages and disadvantages for a certain problem, the following question which forms the basis for this thesis arises: *What is the performance comparison of learning methods when used as a function approximator in LFFC?*

This question can now be divided into a number of questions, being:

1. How does a learning method work as a function approximator?
   This question should be answered by studying literature about learning methods.
2. How to compare all learning methods as function approximators?
   This question should be answered by finding aspects that can be compared.
3. How good are the performances of the learning methods compared to each other?
   This question should be answered by simulations of aspects that are found from question 2.

1.3 Report Structure

The structure of the report is as follows:

Chapter One details the background, motivation, and problem definition of this thesis.

Chapter Two presents an overview of all learning methods. Each method is presented followed by detail of the functioning of the method.

Chapter Three describes aspects used to show the performance comparison of function approximators, followed by simulation results of comparisons. Discussion of the results is presented.

Chapter Four presents the conclusions and recommendations of this thesis.

Chapter 2

Learning Methods

2.1 Introduction

This chapter gives an overview of the learning methods that are compared in this thesis. Those methods are backpropagation used in the Multilayer Perceptron (MLP), the Radial-Basis Function Network (RBFN), the Support Vector Machine (SVM), the Least Square Support Vector Machine (LSSVM), and the Key Sample Machine (KSM).

Firstly an introduction to the Artificial Neural Network (ANN) is given. Then each learning method will be briefly introduced followed by an insight into its working principle. After all methods have been presented, their working principle will be compared and discussed.

2.2 Artificial Neural Networks

The ANN is a method that is inspired by the works of a human brain. It uses weighted connections of processing elements called neurons into which signals are fed and processed to yield outputs. The connector is called a synaptic connector. For specified input signals, outputs of the ANN depend on the weight of the synaptic connectors, the structure of the network, and the function inside the neurons. In this report, the term 'network' will refer to the ANN as well.

Before going further into detail of the structure and elements of the ANN, the basic working principle will be explained. There are two processes occurring in an ANN: The first is signals from the input layer are transmitted to the output layer. In this process, there is no change in the network and the propagated signal is called function signal. This process is called prediction. The second is learning or training. In this process, the parameters of the network are adapted or trained based on the output and will be different for different types of ANNs. Both process are performed iteratively and ended when one or more stopping criterion are fulfilled.

In general, the structure of an ANN can be illustrated as shown as in Figure 2.1. As shown in the figure, the structure consists of an input layer, one or more hidden layers, and an output layer. In the input layer there is no neuron. Therefore signals are propagated directly to the right, to the hidden layer. This hidden layer contains one or more neurons and it is possible to have more than one hidden layer in a network. The last part is the output layer (most right of figure). The number of neurons in the output layer depends on the number of outputs of the network.

Consider a model of a neuron with its inputs as shown in Figure 2.2. The neuron (grey part) receives the weighted signals from the synaptic connectors as its input. These signals are then summed and processed using the so-called activation function. This activation function can be either linear or non-linear. The output is then propagated to the next layer until the output layer. A different activation function used in the neurons can possibly lead to different naming of the ANN.

If the neuron in Figure 2.2 is denoted as neuron k in a network, its output can then be described by a

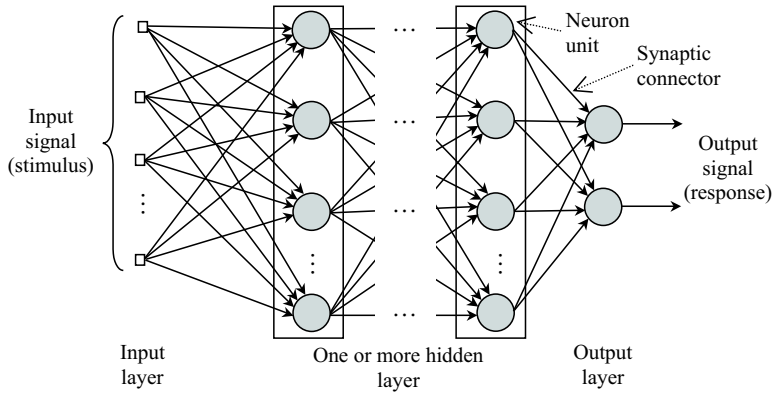


Figure 2.1: Artificial neural network (Haykin, 1999)

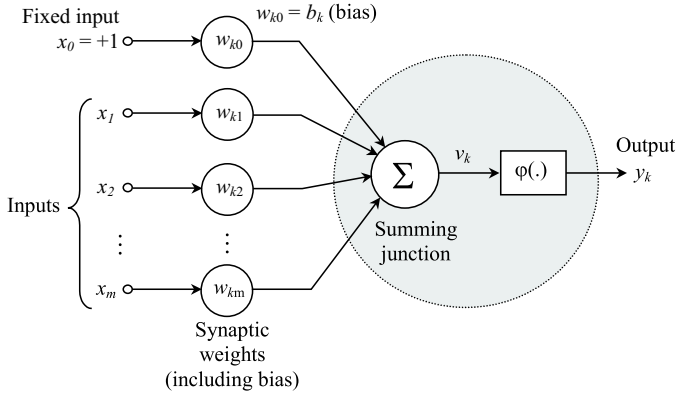


Figure 2.2: Model of a neuron with its inputs (Haykin, 1999)

pair of equations as follows

$$v_k = \sum_{j=1}^{m} w_{kj} x_j + w_0 b_k \tag{2.1}$$

and

$$y_k = \varphi(v_k) \tag{2.2}$$

where $x_1, x_2, \ldots, x_m$ are input signals, $w_{k1}, w_{k2}, \ldots, w_{km}$ are the synaptic weights of the neuron k, v_k is the output of the linear combination of input signals, b_k is the input bias, $\varphi(.)$ is the activation function, and y_k is the output signal. Input bias is a constant input signal which is set to a value of +1 and its amplitude to neuron input is determined by the weight of its synaptic weight. Activation functions that can be used inside a neuron are the threshold function, piecewise linear function, and sigmoid function.

There are two classes of networks that can be constructed from these neurons:

1. *Feed-forward networks*. In this structure, input signals are fed from the input layer and then forwarded to the output layer.

2. *Recurrent networks*. Different from previous, in this structure, some of the network outputs are fed back to the input layer.

Another classification is based on the learning method. There are two classes:

1. *Learning with a teacher*. The training of the network in this class is performed based on examples. The error between the network output and the examples are used for training. The objective is to minimize the error. An example of this class is the MLP.

2. *Learning without a teacher*. There is no example to follow in this method. An example of this class is the Kohonen self-organising.

In this report all of the networks discussed are in the class of *feed-forward networks* that *learn with teacher*.

2.3 Function Approximation

Function approximation involves approximating a relation between the input and output data from a set of training samples or data. Given a set of training samples $\{\mathbf{x}_k, d_k\}_{k=1,\ldots,N}$ where $\mathbf{x}_k$ is the input vector and d_k is the corresponding desired value for sample k, a function approximator will map $\mathbf{x} \rightarrow \hat{y}(\mathbf{x})$. This mapping can be evaluated for any value of $\mathbf{x}$ that is not part of a training set. The performance of a function approximator is measured from the error between the output of the approximator and the target of the training samples.

A problem that is related with the training data is noise. In most cases, real-world data is not noise-free. This noise may reduce the performance of the approximator. This means if the difference between the approximator output $\hat{y}_k$ and d_k, or the square of the training error, is zero, the approximator follows corrupted data which are not the real function value. This problem is known as overfitting.

Overfitting will cause low generalisation performance (Haykin, 1999). Generalisation is one of the performance measures of a function approximator that is about the ability to predict values from data that are not in the training data. In the case of overfitting, the approximator memorises the learning data and does not learn.

Another related problem is approximating a function with high dimensional inputs. This problem is known as *Curse of Dimensionality*. The problem is the higher the dimensionality of the approximated function, the more complex the computation needed and therefore more training data is required to keep the level of accuracy (Haykin, 1999).

In practice, training of a function approximator involves two steps (Prechelt, 1994):

1. Training

2. Validation

Each step uses a different data set. This method is called Cross-Validation. Therefore, to use this training method, the training set has to be divided into two subsets. The first subset is used as a training set for the approximator and then generalisation performance is checked using the second subset. Detail of this method and its variations can be found in (Prechelt, 1994, Kohavi, 1995). This cross-validation is applied in this thesis.

The next part of this report will discuss the MLP using back-propagation learning followed by the RBFN, the SVM and the LSSVM, respectively and the KSM.

2.4 Multilayer Perceptron

The MLP is a class of feed-forward neural networks. Structurally, it has one input layer, one output layer, and one or more hidden layers, where each of the layers consist of one or more neurons. In (Haykin, 1999), the MLP is covered thoroughly and so this section provides a summary.

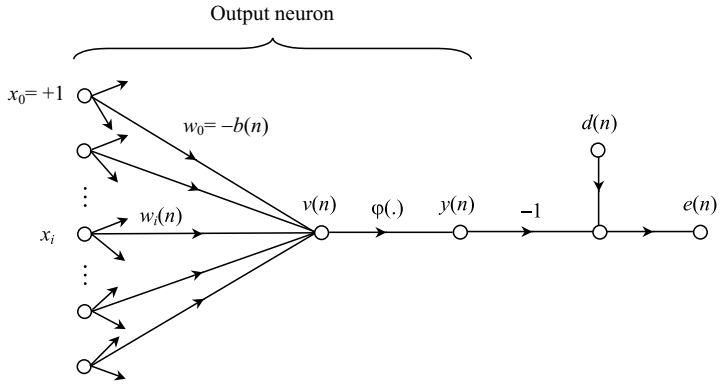


Figure 2.3: Signal-flow graph highlighting the details of the output neuron (Haykin, 1999)

2.4.1 Back-Propagation Algorithm

The most popular training method for the MLP is Back-Propagation Learning or Backprop for short. In this method there are two steps involved which are performed iteratively, the forward and the backward step. In the forward step or prediction, input signals are propagated through the network layer by layer until a set of output signals are produced while all synaptic weights in the network are kept fixed.

These output signals are then compared to the desired signals. The differences (error signals) are then propagated backward through the network from the output layer to input layer against the direction of synaptic connection. In the backward step or correction, the synaptic weights are adjusted according to the error correction rule to make the actual response of the network closer to the desired response. In this step, the learning process is performed. These two steps are repeated until a stopping criteria is reached.

The correction step begins from calculation of the output error of neurons in the output layer of a network. If we assume that only one neuron exists, the output error at time n can be calculated from

$$e(n) = d(n) - y(n) \tag{2.3}$$

The instantaneous value of error energy is defined as

$$\mathcal{E}(n) = \frac{1}{2}e^2(n) \tag{2.4}$$

If there are N number of training samples available, the average square of the error energy can be calculated as

$$\mathcal{E}_{av}(n) = \frac{1}{N}\sum_{n=1}^{N}\mathcal{E}(n) \tag{2.5}$$

It can be seen that both functions above, (2.4) and (2.5), are functions of free parameters, i.e. synaptic weights and bias levels, and (2.5) represents the cost function as a measure of learning performance for a given training set. The objective of the learning process is to adjust the free parameters to minimize $\mathcal{E}_{av}$.

Consider Figure 2.3 which highlights the detail of an output neuron. The sum of input signal x_i, $v(n)$, is written as

$$v(n) = \sum_{i=0}^{m} w_i(n)x_i(n) \tag{2.6}$$

where m is the total number of inputs (including bias).

Output signal $y(n)$ at iteration n can be written as

$$y(n) = \varphi(v(n)) \tag{2.7}$$

In the Backprop algorithm, the correction factor of the synaptic weight $w_i(n)$, $\Delta w_i(n)$, is proportional to the partial derivative $\partial \mathcal{E}/\partial w_i$. Using the chain rule from Calculus, $\partial \mathcal{E}/\partial w_i$ can be calculated from

$$\frac{\partial \mathcal{E}(n)}{\partial w_i(n)} = \frac{\partial \mathcal{E}(n)}{\partial e(n)} \frac{\partial e(n)}{\partial y(n)} \frac{\partial y(n)}{\partial v(n)} \frac{\partial v(n)}{\partial w_i(n)} \tag{2.8}$$

which results in

$$\frac{\partial \mathcal{E}(n)}{\partial w_i(n)} = -e(n)\varphi'(v(n))x_i(n) \tag{2.9}$$

The correction factor $\Delta w_i(n)$ employed to $w_i(n)$ is defined by the delta rule as follows

$$\Delta w_i(n) = -\eta \frac{\partial \mathcal{E}(n)}{\partial w_i(n)} \tag{2.10}$$

where η is the learning-rate parameter. The minus sign accounts for gradient descent in weight space. (2.10) can be rewritten as

$$\Delta w_i(n) = \eta \delta(n) x_i(n) \tag{2.11}$$

where the local gradient $\delta(n)$, which is points to required changes in synaptic weight, is defined by

$$\delta(n) = e(n)\varphi'(v(n)) \tag{2.12}$$

Calculation of the local gradient in (2.12) applies only if the neuron is in the output layer. For a neuron in the hidden layer, consider Figure 2.4. For neuron j as a hidden node, the local gradient is

$$\begin{aligned} \delta_j(n) &= -\frac{\partial \mathcal{E}(n)}{\partial x_j(n)} \frac{\partial x_j(n)}{\partial v_j(n)} \\ &= -\frac{\partial \mathcal{E}(n)}{\partial x_j(n)} \varphi_j'(v_j(n)) \end{aligned} \tag{2.13}$$

The first term of (2.13) can be calculated as

$$\frac{\partial \mathcal{E}(n)}{\partial x_j(n)} = -\delta(n) w_j(n) \tag{2.14}$$

Local gradient $\delta_j(n)$ can be written as

$$\delta_j(n) = \varphi_j'(v_j(n))\delta(n) w_j(n) \tag{2.15}$$

where δ is the local gradient calculated previously in (2.12).

The training time using the method derived above is long. One heuristic way to increase the learning speed is using a momentum term. The delta rule in (2.11) can be generalized by adding a momentum term that is defined as the previous synaptic weight multiplied by a constant, so (2.11) becomes

$$\Delta w_{ji}(n) = \alpha \Delta w_{ji}(n-1) + \eta\, \delta_j(n) y_i(n) \tag{2.16}$$

where α is the momentum constant which is usually positive.

The MLP has two modes of training, namely:

1. **Batch mode**. Synaptic weight adaptation is computed after all training samples have been presented to the network. One phase of feeding all training samples is called *epoch*.

2. **Sequential mode**. Synaptic weight adaptation is computed for each input-output data pair.

Inside a neuron, two activation functions which can be used are:

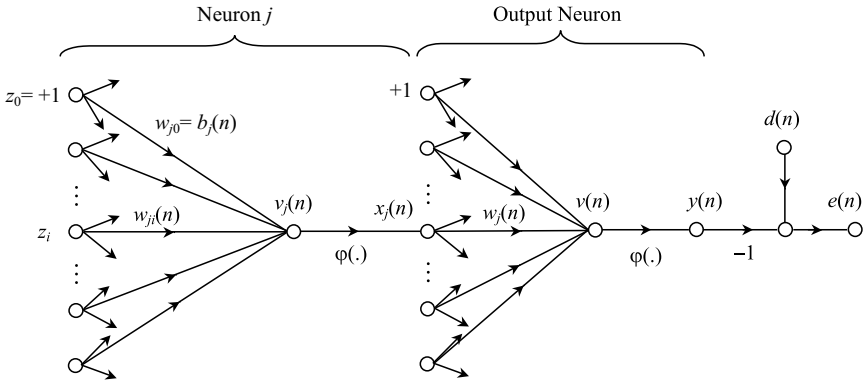


Figure 2.4: Signal-flow graph highlighting the details of the output neuron connected to the hidden neuron j (Haykin, 1999)

1. *Logistic function*. This function has the form as follows

$$\varphi_j(v_j(n)) = \frac{1}{1 + \exp(-a\, v_j(n))} \qquad a > 0, -\infty < v_j < \infty \tag{2.17}$$

Its derivative with respect to $v_j(n)$ is

$$\varphi'_j(v_j(n)) = \frac{a\, \exp(-a\, v_j(n))}{[1 + \exp(-a\, v_j(n))]^2} \tag{2.18}$$

Using $y_j(n) = \varphi_j(v_j(n))$

$$\varphi'_j(v_j(n)) = a\, y_j(n)[1 - y_j(n)]$$

$\varphi'_j(v_j(n))$ is maximum at $y_j(n) = 0.5$ and minimum at $y_j(n) = 0$ or $y_j(n) = 1$. The amount of change in synaptic weight of the network is proportional to the derivative $\varphi'_j(v_j(n))$ so that a large change in synaptic weight occurs for neurons into which the input signals are in their mid-range. This contributes to the functions stability as a learning algorithm. A plot of the logistic function is shown in Figure 2.5(a).

2. *Hyperbolic Tangent Function*. This function is defined by

$$\varphi_j(v_j(n)) = a\, \tanh(b\, v_j(n)) \qquad a, b > 0 \tag{2.19}$$

Its derivative with respect to $v_j(n)$ is

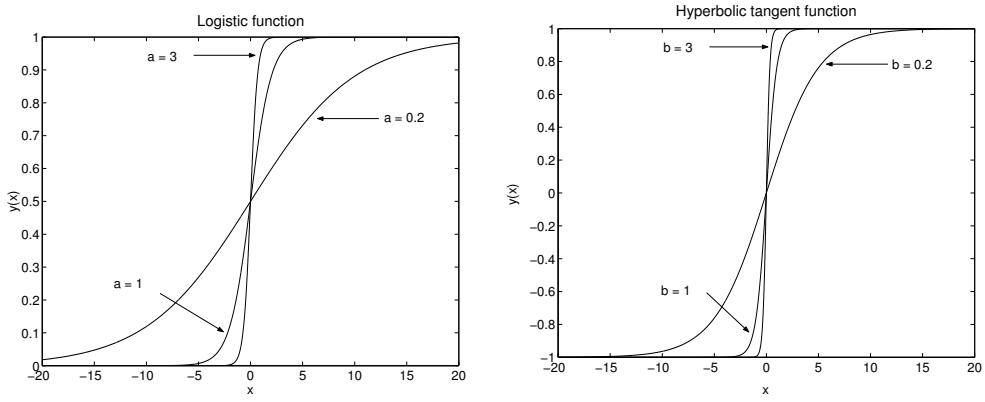
$$\varphi'_j(v_j(n)) = \frac{b}{a}[a - y_j(n)][a + y_j(n)] \tag{2.20}$$

A plot of the hyperbolic tangent function is shown in Figure 2.5(b).

2.4.2 Discussion

In one view the MLP can be seen as a mapping from input space to output space via one or more feature spaces in hidden layers. The dimension of this mapping in the feature spaces is determined by the number of neurons in each hidden layer and the number of feature spaces depend on the number of hidden layers.

In another view, the MLP works based on the allocation of output error of the network in each synaptic weight. After the error is calculated from the difference between the output of the network and target value, the weight of the synaptic connections are updated based on the error contribution of each neuron, while the error from a neuron itself is a function of the synaptic weights connected to it. So the bigger



(a) Logistic (b) Hyperbolic tangent, $a = 1$

Figure 2.5: Plot of activation functions for different parameters

the contribution of a synaptic weight to the error, the bigger the modification of its value in the adaptation process and vice versa.

The role of learning parameter η is to determine the quantity of the synaptic weight correction and its value is between 0 and +1. Setting η too large can cause instability to the training process, whereas setting η too small will lengthen the training time.

One drawback of the MLP learning is the slow learning time. One heuristic way to increase the learning time is by adding a momentum term, see Page 9. When the sign of a synaptic correction is always the same during the iteration, the momentum has the effect of accelerating the learning, however if the sign of the correction is changed during the iteration, the momentum gives a stabilizing effect (Haykin, 1999).

Another drawback is related with the initial value of the synaptic weights which are set small and randomly. For different initial weights the result of the approximation can be different. This can be seen if an MLP is trained and simulated with the same training data but different initial weights. There will be conditions where the approximation result of a network is worse than others. This problem is known as the local minima problem.

In dealing with curse of dimensionality, no further information can be found except from (Haykin, 1999) which states that the MLP suffers from curse of dimensionality since its complexity increases with an increase in input dimensionality.

2.5 Radial-Basis Function Networks

The RBFN is a feed-forward neural network that uses a radial-basis function in its processing element (neuron). In its structure it has an input layer, an output layer and one or more hidden layers, but one hidden layer is most common.

Before proceeding into the discussion about the RBFN, it is necessary to know the definition of a radial-basis function (RBF). A RBF is a function whose output is symmetric around an associated center $\boldsymbol{\mu}$ (Ghosh and Nag, 2001). Those that are of particular interest in the RBFN are (Haykin, 1999):

1. Multiquadrics
$$\phi(r) = \left(r^2 + c^2\right)^{\frac{1}{2}} \qquad \text{for some } c > 0$$
2. Inverse multiquadrics
$$\phi(r) = \frac{1}{\left(r^2 + c^2\right)^{\frac{1}{2}}} \qquad \text{for some } c > 0$$

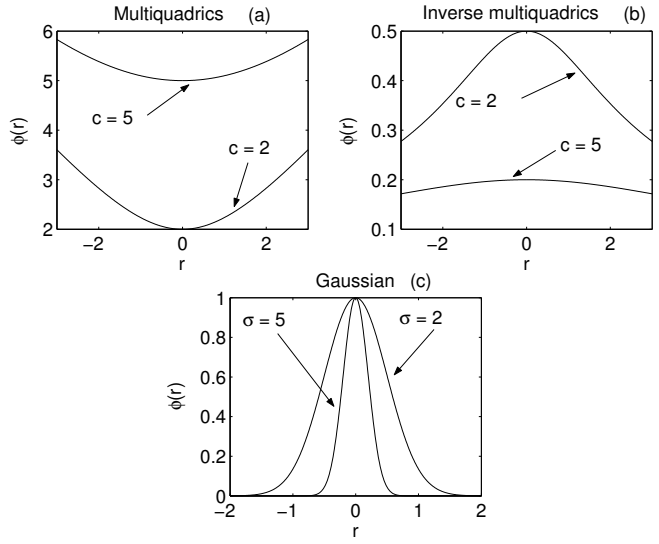


Figure 2.6: Plot of RBFs $\phi(r)$ with different parameters: (a)Multiquadrics (b)Inverse multiquadrics (c)Gaussian

3. Gaussian function

$$\phi(r) = \exp\left(-\frac{r^2}{2\sigma^2}\right) \qquad \text{for some } \sigma > 0$$

Among the above functions, the Gaussian is most commonly used. The shape of all functions above is illustrated in Figure 2.6.

The application of the RBFN as a function approximator can be derived from solving the strict interpolation problem as follows

Definition 1 (Interpolation Problem) *(Haykin, 1999) Given a set of N different points* $\{\mathbf{x}_i \in \mathbb{R}^{m_0} | i = 1, 2, \ldots, N\}$ *and a corresponding set of N real numbers* $\{d_i \in \mathbb{R}^1 | i = 1, 2, \ldots, N\}$ *find a function F that satisfies the interpolation condition:*

$$F(\mathbf{x}_i) = d_i, \qquad i = 1, 2, \ldots, N \tag{2.21}$$

A RBF technique to solve this problem is by choosing a function F that has the following form:

$$F(\mathbf{x}) = \sum_{i=1}^{N} w_i \, \phi(\|\mathbf{x} - \mathbf{x}_i\|) \tag{2.22}$$

where $\{\phi(\|\mathbf{x} - \mathbf{x}_i\|) | i = 1, 2, \ldots M\}$ is a set of RBFs and $M = N$, namely the number of RBFs is the same as the number of data, and notation in (2.22) $\|.\|$ is usually Euclidean.

From (2.21) and (2.22), a set of simultaneous linear equations for unknown coefficients is obtained:

$$\begin{pmatrix} \phi_{11} & \phi_{12} & \cdots & \phi_{1M} \\ \phi_{21} & \phi_{22} & \cdots & \phi_{2M} \\ \vdots & \vdots & \vdots & \vdots \\ \phi_{N1} & \phi_{N2} & \cdots & \phi_{NM} \end{pmatrix} \begin{pmatrix} w_1 \\ w_2 \\ \vdots \\ w_N \end{pmatrix} = \begin{pmatrix} d_1 \\ d_2 \\ \vdots \\ d_N \end{pmatrix} \tag{2.23}$$

where

$$\phi_{ji} = \phi(\|\mathbf{x}_j - \mathbf{x}_i\|), \qquad i = 1, 2, \ldots, N \quad j = 1, 2, \ldots, M \tag{2.24}$$

In matrix form, (2.23) can be written as

$$\mathbf{\Phi w} = \mathbf{x} \tag{2.25}$$

where $N-$by-1 vectors $\mathbf{d}$ and $\mathbf{w}$ are the desired response vector and linear weight vector respectively, N is the size of the training sample, M is the number of RBFs, and $\mathbf{\Phi}$ is an $N-$by$-M$ matrix with element ϕ_{ji} written as

$$\mathbf{\Phi} = \{\phi_{ji} | i = 1, 2, \ldots, N, j = 1, 2, \ldots, M\}$$

If $\mathbf{\Phi}$ is nonsingular, vector $\mathbf{w}$ can be calculated as

$$\mathbf{w} = \mathbf{\Phi}^{-1}\mathbf{x} \tag{2.26}$$

Singularity of $\mathbf{\Phi}$ can be avoided by a set of training data whose elements/points are different, so that all rows and columns are linearly independent.

Once the weights have been determined, they can be used for approximation by feeding the validation data set. If the training data follow an unknown function y and its estimate is denoted by $\hat{y}$, the approximation of new data $x_k, k = 1, \ldots, P$ can be written as

$$\begin{pmatrix} \phi_{11} & \phi_{12} & \cdots & \phi_{1M} \\ \phi_{21} & \phi_{22} & \cdots & \phi_{2M} \\ \vdots & \vdots & \vdots & \vdots \\ \phi_{P1} & \phi_{P2} & \cdots & \phi_{PM} \end{pmatrix} \begin{pmatrix} w_1 \\ w_2 \\ \vdots \\ w_M \end{pmatrix} = \begin{pmatrix} \hat{y}_1 \\ \hat{y}_2 \\ \vdots \\ \hat{y}_P \end{pmatrix} \tag{2.27}$$

where

$$\phi_{ki} = \phi(\|\mathbf{x}_k - \mathbf{x}_i\|), \quad i = 1, 2, \ldots, M \quad k = 1, 2, \ldots, P$$

In the strict interpolation above the number of RBFs are as many as the data points, which may end up with overfitting. To avoid such a problem, the number of RBFs used is less than the number of data, i.e. $M < N$. This scheme is known as the Generalised RBFN (Haykin, 1999).

2.5.1 Training Radial-Basis Function Networks

After defining the network, as it is in other ANNs, the RBFN needs training. The parameters that can be trained in this network are RBFs' parameters (centres and widths), and the synaptic weights. Different from the MLP, training of this network can be performed in three different ways:

1. Training with fixed randomly selected centres and fixed equal width. This way the training is only searching for the synaptic weights.
2. Self-organised selection of centres. The centres and width of the RBF are sought first and once they are determined the synaptic weight is calculated.
3. Supervised selection of centres. All of the parameters are trained simultaneously using gradient descent optimisation.

All these training methods are explained in detail in the subsequent sub-sections.

Fixed centres selected at random

In this training method, firstly, the centers of a RBF network are located randomly and are then kept fixed during training. Assume that Gaussian RBFs whose standard deviation is fixed according to the spread of the centers are used, namely

$$G\left(\|\mathbf{x} - \boldsymbol{\mu}_i\|^2\right) = \exp\left(-\frac{m_1}{d_{max}}\|\mathbf{x} - \boldsymbol{\mu}_i\|^2\right), \qquad i = 1, 2, \ldots, m_1$$

where m_1 is the number of centers and d_{max} is the distance between the chosen centers.

The parameters that need to be trained in this way are the weights in the output layer. This can be solved by using the pseudoinverse method

$$\mathbf{w} = \mathbf{G}^{+}\mathbf{d} \tag{2.28}$$

where $\mathbf{d}$ is the desired response vector in the training set, $\mathbf{G}^{+}$ is the pseudoinverse of $\mathbf{G}$. The pseudoinverse for a nonsquare matrix is functionally equal with the inverse of a square matrix.

Calculation of the pseudoinverse of any matrix $\mathbf{A}$ depends on three conditions as follows (Golub and Loan, 1996):

- Columns in $\mathbf{A}$ are linear independent, pseudo inverse of $\mathbf{A}$ is defined as

$$\mathbf{A}^{+} = \left(\mathbf{A}^{T}\mathbf{A}\right)^{-1}\mathbf{A}^{T} \tag{2.29}$$

- Rows in $\mathbf{A}$ are linear independent, pseudoinverse of $\mathbf{A}$ is defined as

$$\mathbf{A}^{+} = \mathbf{A}^{T}\left(\mathbf{A}^{T}\mathbf{A}\right)^{-1} \tag{2.30}$$

- Neither rows nor columns in A are linear independent, pseudoinverse of $\mathbf{A}$ is calculated using Singular Value Decomposition (SVD).

Self-organised selection of centers

This method consists of two steps:

1. Estimating the appropriate locations for the centers of RBFs in the hidden layer.
2. Estimating the linear weight of the output layer.

The center locations of the RBFs and their common widths are computed using a clustering algorithm, for example, the k-means clustering algorithm or self-organising feature map from Kohonen. Once the centers are identified the computation of the weights of the output layer can be performed using pseudoinverse as it is done in the previous subsection.

Supervised selection of centers

In this approach, the training of the RBFN takes the most general form, i.e. all of the free parameters of the network undergo a supervised learning process. This process is a kind of error learning that is implemented using the gradient descent procedure. The first step in the development of the learning procedure is to define the instantaneous value of the cost function

$$\mathcal{E}(n) = \frac{1}{2}\sum_{j=1}^{N} e_j^2 \tag{2.31}$$

where N is the size of the training sample used to do the training and e_j is the error signal defined by

$$\begin{aligned} e_j &= d_j - F^{*}(\mathbf{x}_j) \\ &= d_j - \sum_{i=1}^{M} w_i\, G(\|\mathbf{x}_j - \boldsymbol{\mu}_i\|_{C_i}) \end{aligned} \tag{2.32}$$

A Gaussian RBF $G(\|\mathbf{x}_j - \boldsymbol{\mu}_i\|_{C_i})$ centered at $\boldsymbol{\mu}_i$ and with norm weighting matrix $\mathbf{C}$ may be expressed as

$$\begin{aligned} G(\|\mathbf{x}_j - \boldsymbol{\mu}_i\|_{C_i}) &= \exp\left[-(\mathbf{x} - \boldsymbol{\mu}_i)^{T}\mathbf{C}^{T}\mathbf{C}(\mathbf{x} - \boldsymbol{\mu}_i)\right] \\ &= \exp\left[-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_i)^{T}\boldsymbol{\Sigma}^{-1}(\mathbf{x} - \boldsymbol{\mu}_i)\right] \end{aligned} \tag{2.33}$$

where

$$\frac{1}{2}\mathbf{\Sigma}^{-1} = \mathbf{C}^T\mathbf{C}$$

is the spread of the RBF.

The requirement is to find the three parameters $w_i, \boldsymbol{\mu}_i,$ and, $\mathbf{\Sigma}^{-1}$ to minimise $\mathcal{E}$. The result is summarised as follows:

1. Linear weights (output layer)

$$\begin{aligned}\frac{\partial \mathcal{E}(n)}{\partial w_i(n)} &= -\sum_{j=1}^{N} e_j(n) G(\|\mathbf{x}_j - \boldsymbol{\mu}_i\|_{C_i}) \\ w_i(n+1) &= w_i(n) - \eta_1 \frac{\partial \mathcal{E}(n)}{\partial w_i(n)} \qquad i = 1, 2, \ldots, m_1\end{aligned}$$

2. Position of centers (hidden layer)

$$\begin{aligned}\frac{\partial \mathcal{E}(n)}{\partial \boldsymbol{\mu}_i(n)} &= -2w_i(n)\sum_{j=1}^{N} e_j(n) G'(\|\mathbf{x}_j - \boldsymbol{\mu}_i\|_{C_i}) \mathbf{\Sigma}_i^{-1}[\boldsymbol{x}_j - \boldsymbol{\mu}_i] \\ \boldsymbol{\mu}_i(n+1) &= \boldsymbol{\mu}_i(n) - \eta_2 \frac{\partial \mathcal{E}(n)}{\partial \boldsymbol{\mu}_i(n)} \qquad i = 1, 2, \ldots, m_1\end{aligned}$$

3. Spreads of centers (hidden layer)

$$\begin{aligned}\frac{\partial \mathcal{E}(n)}{\partial \mathbf{\Sigma}_i^{-1}(n)} &= w_i(n)\sum_{j=1}^{N} e_j(n) G'(\|\mathbf{x}_j - \boldsymbol{\mu}_i\|_{C_i}) O_{ji}(n) \\ O_{ji}(n) &= [\mathbf{x}_j - \boldsymbol{\mu}_i][\mathbf{x}_j - \boldsymbol{\mu}_i]^T \\ \mathbf{\Sigma}_i^{-1}(n+1) &= \mathbf{\Sigma}_i^{-1}(n) - \eta_3 \frac{\partial \mathcal{E}(n)}{\partial \mathbf{\Sigma}_i^{-1}(n)}\end{aligned}$$

2.5.2 Discussion

The RBFN, from its name, uses RBFs to construct the input-output nonlinear mapping, instead of using a sigmoidal nonlinear function as in the MLP. The RBF is called a kernel function which maps the input into the feature space. It can be seen that the input space should be covered by the RBFs that are distributed inside it.

The RBFN works by calculating the distance between the input data and the center of the RBF. The nearer the input with the center, the closer the output of the RBF to +1. When one input is very close to one of the neuron centres, its output dominates the output of the network while the output of other neurons is much smaller. This shows that the RBF constructs a local approximation for each neuron. Furthermore, a neuron directly corresponds to the mapping of an interval input-output. Thus, it is enough to use only one hidden layer in a RBFN.

Input space coverage is determined by the width and the number of RBFs. If the RBFs' widths are large and they almost overlap each other, there are inputs that activate more than one RBF so that the columns represented by those RBFs are closer to being linear dependent. On the other hand if the distance is too far, there will be data that do not activate any RBF so that the network is not trained at that region. However, using more RBFs of small width will make the network behaviour closer to strict interpolation which means overfitting can occur.

From the previous paragraph it can also be concluded that the generalisation of a RBFN is dependent on how well the neurons cover the input space. So one problem in designing a RBFN is balancing the number of RBFs and their width such that optimal training and generalisation performance are obtained.

Training with random centre positions does not give optimal results since the results will be different for each run. Alternatively, visual design can be considered if the input dimension is less than three. The RBFs' positions can be arranged in such a way that they are located on positions that give less error. Nevertheless it is hard to apply such a design method since, in general, real-world data has a large dimension.

Self-organised selection of centres puts the centres of a certain width on the centres of a group of data. It is expected that the error contributions from data in those groups are small, since they are close to the RBFs centres. In this training method, blank spots that are not covered by the RBFs possibly occur.

In the case of supervised selection of centres and width, since this method uses gradient descent optimisation, it could probably be that the error minimisation with respect to both parameters is trapped into local minima. The other disadvantage is that the training time is long. But the result of this training, if the training is not trapped into local minima, is optimal in the sense that the RBF parameters and the synaptic weights are set to give minimal error.

In the case of high dimensional function approximation, as it is in the MLP, this method also suffers from curse of dimensionality (Haykin, 1999).

2.6 Support Vector Machine

The Support Vector Machine (SVM) is a relatively new method developed based on statistical learning theory proposed by Vapnik (Smola and Schölkopf, 1998).

In this section, the function to be approximated is written as

$$f(\mathbf{x}) = \mathbf{w}^T \varphi(\mathbf{x}) + b \tag{2.34}$$

In the SVM, the goal of the approximation is to find a function $f(\mathbf{x})$ that has at most ε deviation from the actual output of the approximator for all training data and, at the same time, be as flat as possible. Flat means that $\mathbf{w}$ should be small and one way to ensure flatness is by minimising $||w||^2$. The deviation has a relation with the ε -insensitive loss function which is defined as (Haykin, 1999)

$$L_\varepsilon = \begin{cases} |d - \hat{y}| - \varepsilon & \text{for } |d - \hat{y}| \geq \varepsilon \\ 0 & \text{otherwise} \end{cases} \tag{2.35}$$

where $\mathbf{d}$ is the target, $\hat{y}$ is the approximated value. (2.35) says that if the difference between the target and the approximated value is less than ε , it is ignored.

The approximation problem can be formulated as a convex minimisation problem as follows (Smola and Schölkopf, 1998):

$$\mathbf{\Phi}(\mathbf{w}) = \frac{1}{2}\mathbf{w}^T\mathbf{w}$$

subject to

$$\begin{aligned} d_i - \mathbf{w}^T\boldsymbol{\varphi}(\mathbf{x}_i) - b &\leq \varepsilon \qquad i = 1, 2, \ldots, N \\ \mathbf{w}^T\boldsymbol{\varphi}(\mathbf{x}_i) + b - d_i &\leq \varepsilon \qquad i = 1, 2, \ldots, N \end{aligned}$$

To allow some errors, slack variables ξ and ξ^* are introduced. The minimisation problem then becomes

$$\mathbf{\Phi}(\mathbf{w}, \xi, \xi') = \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\left(\sum_{i=1}^{N}(\xi_i + \xi_i')\right) \tag{2.36}$$

subject to

$$d_i - \mathbf{w}^T\boldsymbol{\varphi}(\mathbf{x}_i) - b \leq \varepsilon + \xi_i \qquad i = 1, \ldots, N \tag{2.37}$$

$$\mathbf{w}^T\boldsymbol{\varphi}(\mathbf{x}_i) + b - d_i \leq \varepsilon + \xi_i' \qquad i = 1, \ldots, N \tag{2.38}$$

$$\xi_i, \xi_i' \geq 0 \qquad i = 1, \ldots, N \tag{2.39}$$

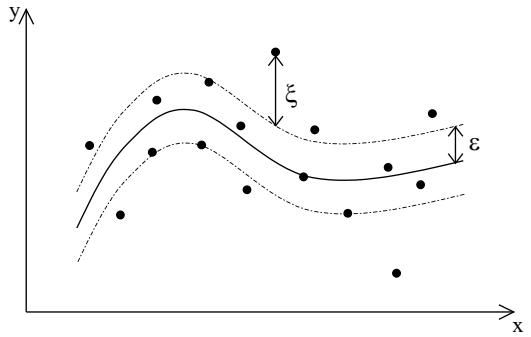


Figure 2.7: SVM for 1D Function Approximation

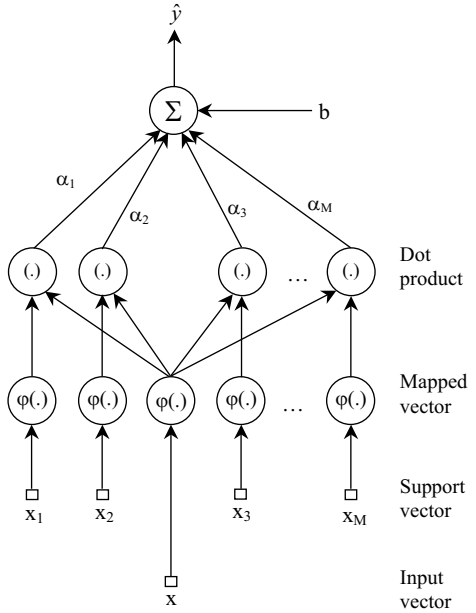


Figure 2.8: SVM Structure for Function Approximation

An illustration of the SVM for 1D function approximation can be seen in Figure 2.7.

The Lagrangian function can then be defined as (Haykin, 1999)

$$\begin{aligned} J(\mathbf{w}, b, \xi, \xi', \alpha, \alpha', \gamma, \gamma') &= C\sum_{i=1}^{N}(\xi_i + \xi_i') + \frac{1}{2}\mathbf{w}^T\mathbf{w} - \sum_{i=1}^{N}\alpha_i\left[\mathbf{w}^T\boldsymbol{\varphi}(\mathbf{x}_i) + b - d_i + \varepsilon + \xi_i\right] \\ &- \sum_{i=1}^{N}\alpha_i'\left[d_i - \mathbf{w}^T\boldsymbol{\varphi}(\mathbf{x}_i) - b + \varepsilon + \xi_i'\right] - \sum_{i=1}^{N}(\gamma_i\xi_i + \gamma_i'\xi_i') \end{aligned} \tag{2.40}$$

where α_i and α_i' are the Lagrange multipliers. The requirement is to minimise $J(\mathbf{w}, \xi, \xi',$
$\alpha, \alpha', \gamma, \gamma')$ with respect to weight vector **w**, b, and slack variables ξ and ξ', and maximise it with respect to α, α', γ, γ'.

The corresponding dual problem can be written as (Haykin, 1999)

$$\begin{aligned} Q(\alpha_i, \alpha_i') &= \sum_{i=1}^{N} d_i(\alpha_i - \alpha_i') - \varepsilon \sum_{i=1}^{N}(\alpha_i - \alpha_i') \\ &- \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}(\alpha_i - \alpha_i')(\alpha_j - \alpha_j')K(\mathbf{x}_i - \mathbf{x}_j) \end{aligned} \tag{2.41}$$

subject to

$$\sum_{i=1}^{N}(\alpha_i - \alpha_i') = 0 \tag{2.42}$$

$$0 \le \alpha_i \le C, \quad i = 1, 2, \ldots, N \tag{2.43}$$

$$0 \le \alpha_i' \le C, \quad i = 1, 2, \ldots, N \tag{2.44}$$

where $K(\mathbf{x}_i, \mathbf{x}_j)$ is the inner-product kernel defined to fulfill Mercer's theorem:

$$K(\mathbf{x}_i, \mathbf{x}_j) = \boldsymbol{\varphi}(\mathbf{x}_i)^T \boldsymbol{\varphi}(\mathbf{x}_j)$$

The approximation for the new value data $\mathbf{x}_n$ is written as follows (Haykin, 1999)

$$\begin{aligned} F(\mathbf{x}_n, \mathbf{w}) &= \mathbf{w}^T\mathbf{x}_n + b \\ &= \sum_{i=1}^{N}(\alpha_i - \alpha_i')K(\mathbf{x}_n, \mathbf{x}_i) + b \end{aligned} \tag{2.45}$$

The SVM structure for function approximation is shown in Figure 2.8. The constant b can be calculated from (Smola and Schölkopf, 1998)

$$\begin{aligned} b &= y_i - \mathbf{w}^T\boldsymbol{\varphi}(\mathbf{x}_i) - \varepsilon \quad \text{for } \alpha_i \in (0, C) \\ b &= y_i - \mathbf{w}^T\boldsymbol{\varphi}(\mathbf{x}_i) + \varepsilon \quad \text{for } \alpha_i' \in (0, C) \end{aligned} \tag{2.46}$$

2.6.1 Discussion

As it is illustrated in Figure 2.7, due to the ε-insensitive loss function, the SVM works by making a band/tube with diameter $2\,\varepsilon$ around the approximated function. The tube is shaped in the training error optimisation process and in this process, error from samples inside the tube are ignored. The error is optimised indirectly in the feature space using the kernel function in the input space, i.e. by minimising the innerproduct of the kernel's output. It is expected that the nonlinear function to approximate, can be mapped linearly in the feature space. At the end of training there are some samples outside the tube, known as *Support Vectors* and they are used for prediction.

It can be said that the number of support vectors depends on the tube's width which is determined by ε while the parameter C controls the trade-off between the training error and the magnitude of the weights. This parameter C determines the weight of error minimisation such that a larger C will force the optimisation process to minimise the slack variables. Therefore it can be seen that both parameters control the result of the approximation and also the number of support vectors. The difficulty in this method emerges as to how to determine both parameters since they are independent and should be tuned simultaneously.

With respect to computation, it is desired to have a small number of support vectors in the model such that the prediction can be performed with less calculation while keeping training error small, and good generalisation. A small number of support vectors is also desired to avoid overfitting.

While from (2.45), it can be concluded that good generalisation is possible with more data points in the resulting model. Using this consideration, tuning of parameters ε and C is directed to a trade-off between the number of support vectors and the computational efficiency. However, overfitting problem should be considered.

2.7 Least Square Support Vector Machine

The Least Square Support Vector Machine (LSSVM) is a variant of the SVM explained previously. This method is developed by Suykens et. al. (Suykens, 2000).

The difference of the LSSVM is in its cost function. It has a quadratic cost function J which is defined as

$$J(\mathbf{w}, e) = \frac{1}{2}\mathbf{w}^T\mathbf{w} + \gamma\frac{1}{2}\sum_{i=1}^{N} e_i^2 \tag{2.47}$$

subject to

$$d_i = \mathbf{w}^T\boldsymbol{\varphi}(\mathbf{x}_i) + b + e_i, \quad i = 1, 2, \ldots, N \tag{2.48}$$

where γ is a constant and e_i is the error between the estimated value $\hat{y}_i$ and the the desired value d_i.

The Lagrangian of the optimisation problem can be expressed as

$$J(\mathbf{w}, e, b, \alpha) = \frac{1}{2}\mathbf{w}^T\mathbf{w} + \gamma\frac{1}{2}\sum_{i=1}^{N} e_k^2 - \sum_{i=1}^{N}\alpha_i\left[\mathbf{w}^T\boldsymbol{\varphi}(\mathbf{x}_i) + b + e_i - d_i\right] \tag{2.49}$$

where α is the Lagrange multiplier.

As in the SVM, the error has to be optimised for which the condition for optimality is fulfilled, by setting the derivatives of $J(\mathbf{w}, e, b, \alpha)$ with respect to $\mathbf{w}, b, e_i,$ and α_i equal to zero.

The optimisation result can be written in matrix form as follows (Suykens, 2000)

$$\begin{pmatrix} 0 & 1 \\ 1 & \boldsymbol{\varphi}(\mathbf{x}_i)^T\boldsymbol{\varphi}(\mathbf{x}_j) + \gamma^{-1} \end{pmatrix}\begin{pmatrix} b \\ \alpha_i \end{pmatrix} = \begin{pmatrix} 0 \\ d_i \end{pmatrix} \qquad i, j = 1, 2, \ldots, N$$

and in more compact form as in (Suykens, 2000)

$$\begin{pmatrix} 0 & \bar{\mathbf{1}}^T \\ \bar{\mathbf{1}} & \boldsymbol{\Omega} + \gamma^{-1}\mathbf{I} \end{pmatrix}\begin{pmatrix} b \\ \boldsymbol{\alpha} \end{pmatrix} = \begin{pmatrix} 0 \\ \mathbf{d} \end{pmatrix} \tag{2.50}$$

where $\mathbf{d}, \bar{\mathbf{1}},$ and $\boldsymbol{\alpha}$ are an $N \times 1$ array, namely

$$\begin{aligned} \mathbf{d} &= [d_1 \quad d_2 \quad \cdots \quad d_N]^T \\ \bar{\mathbf{1}} &= [1 \quad \cdots \quad 1]^T \\ \boldsymbol{\alpha} &= [\alpha_1 \quad \alpha_2 \quad \cdots \quad \alpha_N]^T \end{aligned}$$

and

$$\begin{aligned} \Omega &= \boldsymbol{\varphi}(\mathbf{x}_i)^T\boldsymbol{\varphi}(\mathbf{x}_j), \qquad i, j = 1, 2, \ldots, N \\ &= K(\mathbf{x}_i\mathbf{x}_j) \end{aligned}$$

The values of $\boldsymbol{\alpha}$ and b can be calculated by solving (2.50). The approximation equation for the new value data $\mathbf{x}_n$ is written as

$$F(\mathbf{x}_n) = \sum_{i=1}^{N}\alpha_i K(\mathbf{x}_n, \mathbf{x}_i) + b \tag{2.51}$$

In (2.51), the new value data is calculated without using weight vector $\mathbf{w}$ since it vanished when the cost function J in (2.49) was differentiated with respect to $\mathbf{w}$.

2.7.1 Discussion

Different from its predecessor, the LSSVM works by optimising the position of a line (the approximated function) such that the cost function is a minimum. In its cost function, only one parameter needs to be set,

γ, that determines how low the error is minimised which then consequently determines the roughness of the estimation. The larger the value of γ the smaller the cost function but the coarser estimation is obtained. Coarse estimation means the model resulting from training includes a lot of samples or support vectors.

After fulfilling the condition for optimality, the optimisation problem is represented by a set of simultaneous linear equations. By solving these equations, the LSSVM looses sparseness in the solution. To force a sparse solution, the process of pruning is performed after training. This process removes some support vectors while the training performance is measured to track its decrease. Pruning is continued as long as the new performance is not less than specified.

2.8 Key Sample Machine

The Key Sample Machine (KSM) is developed by Kruif (de Kruif and de Vries, 2002) from Least Squares (LS) regression. The overview of this method begins with an explanation of LS, primal and dual LS and follows with how the KSM differs from LS regression.

Given a set of input-output data pairs as the samples and a set of indicator functions $f_k(\mathbf{x}), k = 1, \ldots, P$, the LS method searches for parameters $w_k, k = 1, \ldots, P$ to fulfill the linear relation in the equation

$$\hat{y}(\mathbf{x}) = w_1 f_1(\mathbf{x}) + w_2 f_2(\mathbf{x}) + \cdots + w_P f_P(\mathbf{x}) \tag{2.52}$$

where $\hat{y}$ is the approximation of y, by minimising the sum-of-square approximation error $(y - \hat{y})$ over all samples. The form of the indicator $f_k(\mathbf{x}), k = 1, \ldots, P$ can be constant, linear, or nonlinear, that maps the input space to a feature space. In matrix form, (2.52) can be formulated as follows

$$\hat{\mathbf{y}} = \mathbf{X}\mathbf{w} \tag{2.53}$$

Matrix $\mathbf{X}$ is defined as the indicator for all N samples and target values respectively, and can be formulated as

$$\mathbf{X} = \begin{pmatrix} f_1(\mathbf{x_1}) & f_2(\mathbf{x}_1) & \cdots & f_P(\mathbf{x}_1) \\ f_1(\mathbf{x}_2) & f_2(\mathbf{x}_2) & \cdots & f_P(\mathbf{x}_2) \\ \vdots & \vdots & & \vdots \\ f_1(\mathbf{x}_N) & f_2(\mathbf{x}_N) & \cdots & f_P(\mathbf{x}_N) \end{pmatrix}, \qquad \mathbf{w} = \begin{pmatrix} w_1 \\ w_2 \\ \vdots \\ w_P \end{pmatrix}$$

where the subscript of $\mathbf{x}$ denotes the sample number. The LS problem can be solved by primal and dual optimisation approaches.

2.8.1 Primal form

In the primal form, the minimisation problem is expressed as

$$\min_{\mathbf{w}} \ \frac{1}{2}\lambda\|\mathbf{w}\|_2^2 + \frac{1}{2}\|\mathbf{X}\mathbf{w} - \mathbf{d}\|_2^2 \tag{2.54}$$

where $\mathbf{d}$ is the target vector corresponding to the samples.

The term $\frac{1}{2}\|\mathbf{w}\|_2^2$ is included to decrease the variance of the $\mathbf{w}$'s at the cost of a small bias and λ is the regularisation parameter. The solution of the above minimisation can be found by setting the derivative of (2.54) equal to zero, resulting in

$$\left(\lambda\mathbf{I} + \mathbf{X}^T\mathbf{X}\right)\mathbf{w} = \mathbf{X}^T\mathbf{d} \tag{2.55}$$

In this form, (2.55) requires that the number of training data is equal to or more than the number of indicators. If this requirement is not fulfilled, the matrix $\mathbf{X}^T\mathbf{X}$ will be singular.

The estimation $F(\mathbf{x})$ for a new input x_n, denoted by $\hat{y}$ is

$$\hat{y} = \sum_{i=1}^{N} f_i(\mathbf{x}_n) w_i \tag{2.56}$$

2.8.2 Dual form

Another way to solve the minimisation problem in the primal form is by using its dual form. This is done by the minimisation

$$\min_{\mathbf{w}} \ \frac{1}{2}\lambda \mathbf{w}^T\mathbf{w} + \frac{1}{2}\mathbf{e}^T\mathbf{e} \tag{2.57}$$

with equality constraint

$$\mathbf{d} = \mathbf{X}\mathbf{w} + \mathbf{e}$$

where $\mathbf{e}$ is a column-vector containing the approximation error for all samples.

The Lagrangian of the problem is

$$J(\mathbf{w}, \mathbf{e}, \boldsymbol{\alpha}) = \frac{1}{2}\lambda \mathbf{w}^T\mathbf{w} + \frac{1}{2}\mathbf{e}^T\mathbf{e} - \boldsymbol{\alpha}^T(\mathbf{X}\mathbf{w} + \mathbf{e} - \mathbf{d})$$

The vector α contains the Lagrangian multiplier with the dimension N, the number of data samples. The problem can be solved by setting the derivatives of the Lagrangian with respect to $\mathbf{w}$, $\mathbf{e}$, and $\boldsymbol{\alpha}$ equal to zero.

The solution can be formulated in the primal variable $\mathbf{w}$ or in the dual variable $\boldsymbol{\alpha}$. The solution in primal form, which is the same as in the previous section, can be seen in (2.56).

In dual form, the solution is given by

$$\left(\lambda\mathbf{I} + \mathbf{X}\mathbf{X}^T\right)\boldsymbol{\alpha} = \mathbf{d} \tag{2.58}$$

$$\mathbf{w} = \mathbf{X}^T\boldsymbol{\alpha} \tag{2.59}$$

The estimate of a new input $\mathbf{x}_n$ can be calculated as

$$\hat{y} = f(\mathbf{x}_n)\mathbf{X}^T\left(\lambda\mathbf{I} + \mathbf{X}\mathbf{X}^T\right)^{-1}\mathbf{d} \tag{2.60}$$

$$= \sum_{i=1}^{N}\langle f(\mathbf{x}_n), f(\mathbf{x}_i)\rangle\alpha_i \tag{2.61}$$

This form requires that all of the rows of matrix $\mathbf{X}$ are linearly independent, which means data with the same value is not allowed (leads to singularity of $\mathbf{X}\mathbf{X}^T$). Different to the primal form, dual form does not have problems with smaller training data compared to indicator functions.

2.8.3 Primal and Dual Combination

From (2.56) it can be seen that the calculation of new data input in primal form uses the $f(\mathbf{x})$ explicitly. On the other hand, in dual form, the $f(\mathbf{x})$ is found implicitly in the inner product, as seen (2.61). The combination of primal and dual tries to minimise the difference between the training output in dual form and the target outputs for all data samples, but not all the samples are used for prediction. This is the point where this method goes further than the LS method.

The minimisation of the combination is formulated as follows

$$\min_{\alpha_j}\sum_{i=1}^{N}\left(\left(\sum_{j=1}^{M}\langle f(\mathbf{x}_j), f(\mathbf{x}_i)\rangle\alpha_j\right) - d_i\right)^2 \tag{2.62}$$

where M is the number of vector/indicator functions used in the approximation and N is the number of samples ($M \leq N$). In this combination, not all data is used to calculate the output prediction. This can be done by setting α_i equal to zero that makes $M < N$.

Looking at (2.62), the term in the inner bracket is similar to (2.61) by changing N in (2.61) to M. Based on that similarity, (2.62) can be rewritten as

$$\min_{\alpha_j} \left(\hat{y}_{i,\alpha_j} - d_i\right)^2 \tag{2.63}$$

where $\hat{y}_{i,\alpha_j}$ is the prediction of y_i using $M < N$ indicator functions, and subscript j, $j = 1, \ldots, M$ is the index of the indicator used. Using (2.53), (2.63) is similar with (2.54) if the regularisation term λ is zero.

2.8.4 Choosing Indicator Functions

In the previous subsection, (2.62) indicated that not all indicators are used for prediction. To choose the indicators, a method called *Forward Selection* is used.

This selection starts with no indicator and includes one indicator candidate at a time. This will add one column to $\mathbf{X}$ in (2.53). The criterion for the inclusion of a candidate is by taking an indicator such that the cosine square of the angle between the residual $\mathbf{e} = \mathbf{d} - \mathbf{X}\boldsymbol{\alpha}$ and the generated vector in $\mathbf{X}$ is maximum.

The inclusion of the candidate as an actual indicator depends on whether adding more indicators gives a significant increase to performance or that overfitting occurs. To check this, a statistical test is performed to see whether α_i with some given probability is zero. If the calculated probability is below some user-defined probability bound, the candidate should be included.

The test is performed as follows. Define S^2 as the difference between the sum-of-square approximation error over all samples before and after the inclusion of the candidate. For additive Gaussian noise on y, S^2 is chi-square distributed χ_1^2 if and only if the indicator weight is zero. We have

$$p = \frac{S^2}{\sigma^2} \sim \chi_1^2 \Leftrightarrow \alpha_i = 0 \tag{2.64}$$

where σ^2 is the variance of the noise. σ is a noise estimate which, in this method, can be made as a design parameter to control the coarseness of estimation (de Kruif, 2003). Larger σ makes more support vectors included in the model, therefore the estimation is coarser.

If the probability of the indicator weight is zero, S should be chi-square distributed. The probability of α_i being zero can be determined from the chi-square distribution table. For example, calculation gives $p = 4$. Using the chi-square distribution table gives

$$P(\alpha_i = 0 | p \leq 4) = 4.6\,\%$$

So, the probability that $\alpha_i \neq 0$ is 95.6%.

Dicussion of the KSM method in this section is very detailed so no other discussion will be given.

2.9 Theoretical Comparison of Learning Methods

This section tries to find the distinctions between the theoretical principle of each learning method. In order to make a systematic comparison, the order of comparison will be made similar with that of the method overview in the previous sections, comparison of the RBFN and its previous, the MLP, then comparison of the SVM with both previous methods, the MLP and the RBFN, and so on.

2.9.1 Radial-Basis Function Networks

Learning in RBFNs and MLPs is based on different theories. Backpropagation in the MLP is based on error-correction, namely gradient descent optimisation. Learning in the RBFN is developed based on a strict interpolation problem, but it can be extended to use gradient descent as well.

In the case where RBF parameters have been determined, training of the RBFN is much faster than that of the MLP, since training the RBFN is only involved with solving simultaneous linear equations.

Futhermore, its convergence is guaranteed. But if all parameters of the RBF are trained as well, as seen in Subsection 2.5.1, a long training time and local minima, as in the MLP, is likely to occur. This training mode, when not trapped in local minima, will result in an optimal solution and the RBFN will outperform the MLP (Haykin, 1999).

Another difference between the MLP and the RBFN is their activation function. The MLP uses a sigmoid-like activation function, for example logistic, which makes it a global approximator. This means, every training sample will influence all synaptic weights in the network. On the other hand, each neuron of the RBFN works as a local approximator (see Subsection 2.5.2 on page 15).

2.9.2 Support Vector Machine

The physical structure of the SVM for prediction is similar with that of the MLP and the RBFN. However, learning in the SVM is developed based on statistical learning theory. This theory gives a strong mathematical foundation in developing the SVM method (Haykin, 1999, Smola and Schölkopf, 1998, Cristianini and Shawe-Taylor, 2000).

One of the advantages of the SVM over the MLP and the RBFN lies in the training process. Minimisation of the cost function in the SVM is a convex function minimisation where global minima is guaranteed. Error minimisation in the MLP and the RBFN (in supervised centres training mode) has the possibility to get stuck in local minima. The RBFN in random centre selection and self-organised centre selection modes is not optimal due to the randomness of the centres.

The other advantage of the SVM is when dealing with high dimensional inputs or curse of dimensionality. This advantage comes from implicit mapping in the feature space, where only the dot product of the mapped inputs is optimised and not the actual value (Cristianini and Shawe-Taylor, 2000). Since the dot product of two points is always a scalar, regardless of how high the input dimension is, this method only optimises a scalar and the optimisation can be performed in the input space. This feature is not possessed by the MLP or the RBFN.

2.9.3 Least Square Support Vector Machine

The LSSVM has the majority of the features of the SVM. One difference between both methods is the cost function. The cost function in the SVM is derived from the ε -insensitive loss function while in the LSSVM it is derived from the square error cost function. This difference leads to different ways of optimising training error. The SVM uses quadratic programming to solve the optimisation problem and results in a sparse model. The LSSVM solves the optimisation problem by rearranging it to a set of simultaneous linear equations. This way makes the LSSVM lose sparseness in its model. Sparseness can be forced by the pruning process.

Compared to the MLP, the LSSVM is a very different method.

With the RBFN, a similarity is seen in the use of a set of simultaneous linear equations to calculate the weight parameters, $\mathbf{w}$ in the RBFN and $\boldsymbol{\alpha}$ in the LSSVM.

2.9.4 Key Sample Machine

This method is derived from regression theory from statistics and therefore has similarity with the SVM which is developed from a statistical point of view of learning (Haykin, 1999).

Compared to the MLP, this method is very different and consequently no similarities can be found.

When the RBF paramaters have been determined, RBFNs can be seen as a special case of the primal-form of LS. Neurons in the RBFN are functionally equal with the indicators in LS. Since the number of neurons in the RBFN is always smaller than the training samples, the pseudoinverse of the indicator matrix $\boldsymbol{\Phi}$ in (2.23) can be calculated using (2.29). It can be seen that (2.29) is equal with (2.55) if λ is made zero.

The other similarity is that the RBFN can be designed using forward selection as well. This can be achieved by adding one neuron at a time to reduce error training.

With the SVM it is not easy to reveal their similarities or differences except by deeper study. What can be seen easily is that both methods use kernel functions and perform optimisation in the feature space indirectly, i.e using the dot product of the kernel output and the quadratic cost function in the KSM.

Comparing the KSM with the LSSVM, both are the same except when handling the bias. While in the LSSVM the bias is treated as a separate element, in the KSM it can be included in the indicator function which set to +1.

The other difference is in the pruning method to get a simpler model. In the LSSVM, model simplification is performed by omitting some training samples and retraining the model. Omitting and retraining is repeated until a user-specified value of performance degradation is met. In the KSM, a simple model is obtained by forward selection of indicators. Addition of indicators is terminated if it statistically does not give a significant performance improvement.

2.10 Summary

In this chapter some learning methods that are compared in this thesis for function approximation have been introduced. The overview not only presents the methods from literature, but also gives a comparison between them from a theoretical point of view, such as basic working concepts, and the similarities and differences. Having become acquainted with the theoretical side of learning methods, the next chapter will empirically compare all methods discussed, in simulation.

Chapter 3

Learning Method Comparison by Simulation

3.1 Introduction

In this chapter an empirical comparison of the considered learning methods will be presented. This comparison is performed by simulations used to asses the performance of the learning methods for approximation of different functions. From the simulations, the ability of the learning methods to operate in certain conditions can be discovered.

There are three types of simulations performed in this thesis. They are:

1. Reproducibility
2. High dimensional input
3. Linear motor data

The goal of the first simulation is to see a similarity in the performance of the learning methods for a function that is implemented with different training sets and noise realisations. When a learning method is impelemented, it works with the same plant which means that the function to be approximated is always the same. But apart from the plant itself there are numerous input-output signals that can be measured and sampled for use in training. Furthermore, noise that corrupts the samples can vary even if the same input is applied due to the stochastic nature of noise. From this consideration, reproducibility is an important characteristic of a learning method in control systems and therefore needs to be investigated.

In order to formulate a model of a plant, in relation with a specific task, it is necessary to have sufficient information about the plant. This sufficient information not only concerns how much data is required to characterise the plant, but also what kind of data is needed so that a complete description of the plant for the task can be formulated. For example, consider a moving mass following a trajectory to reach a specified position. Its characteristics to achieve a certain position are not only determined by the position reference, but also the speed and the acceleration. Therefore, in order to control the motion of the mass, these three types of information are needed.

High dimensional input for a function approximator is related with the so-called *Curse of Dimensionality* as discussed previously. More simply this says that the higher the input dimension the more data is needed for an approximator to maintain its generalisation performance (Haykin, 1999). Good generalisation means the approximator is able to mimic the behaviour of the plant. For a more complex plant, more types of inputs are needed to learn the plant. From the curse of dimensionality point of view, this means that more training samples are needed as well. The ability of the learning methods to deal with high dimensional input will be observed in the second simulation.

The last simulation has the goal of comparing the learning methods when dealing with real-world data.

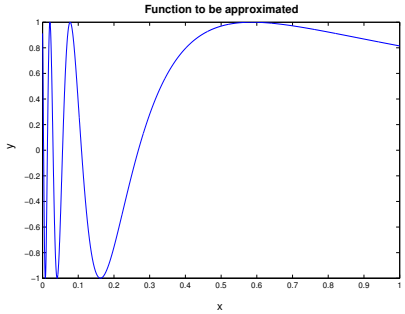


Figure 3.1: Approximated function in reproducibility simulation

From five learning methods, only three are tested. Two methods, the MLP and the RBFN, are omitted since their performance is not as good as that of the other three learning methods.

3.2 Reproducibility Simulation

This section describes the reproducibility simulation. The data used in this simulation is synthetic and the performance measure is mean square error (MSE). Details of the simulation are presented in subsequent subsections and a discussion that compares the result of each method is given following.

3.2.1 Simulation Setup

The simulation setup information can be summarised as follows:

- The function to be approximated is

$$y = f(x) = \sin\left(\frac{1}{x + 0.05}\right), \quad 0 \leq x \leq 1$$

  and can be seen in Figure 3.1

- Two levels of noise are added to the training samples. They are with a standard deviation $\sigma = 0.05$ and $\sigma = 0.15$ from normal distribution.

- The training data consists of two sets of 1650 uniformly random inputs, x_1 and x_2, and from which two sets of output, y_1 and y_2, are calculated. From y_1, ten sets of corrupted y_1 are generated for each noise level that reflect ten different noise realisations. From y_2, ten sets of corrupted y_2 are also generated using noise with standard deviation $\sigma = 0.15$. So there are 30 sets of training data for each approximation method. 150 samples are separated for validation. The use of validation is to ensure that the method has a comparable performance with the training. The validation result is not included for evaluation. The inputs x_1 and x_2 are denoted as *Input 1* and *Input 2* respectively.

- The performance indicators are presented in a histogram. The range of the histogram is between 0 and 0.16 and is divided into 32 bins, i.e. the range for each strip in the histogram is 0.005. The use of the histogram is to provide a fast evaluation at first sight of the results without going further into detail. This means a good or bad result can easily be seen at first sight, although for some results more calculations might be required. The mean and standard deviation of the result accompanies the histogram plot.

In this section, noise with standard deviation 0.05 is referred to as noise $\sigma = 0.05$ and sometimes called as low (level) noise. Also noise with standard deviation 0.15 is referred to as noise $\sigma = 0.15$ and sometimes called as high (level) noise.

In this simulation, there are some categories of reproducibility that will be observed:

1. Reproducibility with respect to different initial weights.
   This category applies only to the MLP, as this method has to be initiated with random initial synaptic weights, and therefore different networks will have distinct initial weights. Ten MLPs with different initial synaptic weights are trained using the same training set.

2. Reproducibility with respect to different noise realisations.
   The comparison is between the MSE of the learning methods that are trained by a training set corrupted with ten realisations of noise $\sigma = 0.15$.

3. Reproducibility with respect to different training sets.
   The learning methods are trained by two different training sets corrupted with equal levels of noise $\sigma = 0.15$.

4. Reproducibility with respect to different noise levels.
   The learning methods are trained by a training set but corrupted by two different noise levels, noise $\sigma = 0.05$ and noise $\sigma = 0.15$.

After defining the reproducibility categories, the evaluation for each category is determined as follows:

1. For reproducibility w.r.t. different initial weights, the evaluation will be applied only to the training set from *Input 1* and based on the width of the histogram's strip. One strip or the difference between the maximum and minimum MSE being less than or equal to 0.005 will be deemed as a *Perfect* result. Two strips or the difference between the maximum and minimum MSE being less than or equal to 0.01 will be deemed as a *Good* result. Other results are deemed as *Bad*.

2. For reproducibility w.r.t. different noise realisations, the evaluation will be applied only to the training set from *Input 1* and based on the width of the histogram's strip. One strip or the difference between the maximum and minimum MSE being less than or equal to 0.005 will be deemed as a *Perfect* result. Two strips or the difference between the maximum and minimum MSE being less than or equal to 0.01 will be deemed as a *Good* result. Other results are deemed as *Bad*.

3. For reproducibility w.r.t. different training sets, the evaluation will be based on the position and the number of the band of different input sets of noise $\sigma = 0.15$. If both results yield the same number of strips on same position, the reproducibility is deemed *Perfect*. If both results yield the same number of strips with one strip difference in the position, the reproducibility is deemed *Good*. Other results are deemed as *Bad*.

4. For reproducibility w.r.t. different noise levels, it will be evaluated whether the MSE of the approximation is an estimate of the noise variance of the target (Draper and Smith, 1998). Thus, the expected MSE for a training set with noise $\sigma = 0.05$ is 0.0025 and noise $\sigma = 0.15$ is 0.0225.

3.2.2 Multilayer Perceptron

The first simulation in the MLP is to test the influence of different initial weights to reproducibility. For this test, ten networks with different initial weights are generated and each is trained with the same input set. The network parameters are set to:

- Number of hidden neurons: 10
- Learning rate: 0.01
- Momentum coefficient: 0.5
- Number of epochs: 500

The result of the different initial weights simulation is shown in Figure 3.2. From the figure, it can be seen that different initial synaptic weights result in a different MSE. It is assumed that the most-left strip

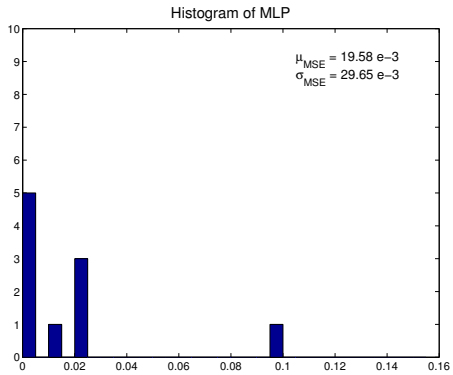


Figure 3.2: Histogram of MSE for different initial weights of the MLP

indicates a global minima result. It is observed that different initial weights of the MLP, will result in some local minima. In fact for parameters in this simulation, local minima are often found.

For the other three categories, a MLP which gives global minima from the previous simulation is used. In the figures, σ denotes standard deviation of noise.

With respect to different noise realisations, the result is shown in Figure 3.3(b). It can be seen from the histogram that the result is bad. The figure shows that 4 out of 10 are trapped in local minima. As it is in the different initial weights simulation, local minima occurs frequently.

With respect to different training sets, compare Figures 3.3(b) and 3.3(c). It can be seen that the results are bad as the positions of the bars in the histograms are not the same.

With respect to different noise levels, compare Figures 3.3(a) and 3.3(b). In Figure 3.3(a), for the low level noise data, one outlier appears. From the mean of the MSE, it can be seen that the MSE is larger than the expected. The same thing can also be seen at the high level noise result in Figure 3.3(b).

3.2.3 Radial-Basis Function Network

The RBFN used in the simulation is designed visually based on the plot of the training data. The number of hidden neurons used in the simulation is 8. The centres of the RBFs are 0.0005, 0.008, 0.015, 0.045, 0.075, 0.14, 0.3, 0.8 and the widths are 0.001, 0.004, 0.011, 0.01, 0.034, 0.1, 0.4, 0.45. The result is shown in Figure 3.4 and σ denotes standard deviation of noise.

With respect to different noise realisations, consider Figure 3.4(b). From the figure, the result is deemed good as the difference between the maximum and the minimum MSE is 3.6 e-4.

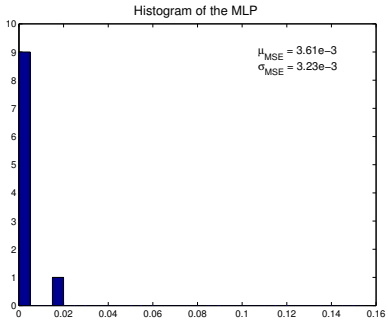
With respect to different training sets, compare Figures 3.4(b) and 3.4(c). From these results, the ability of this method to reproduce similar input is perfect. For both training sets the position of the strips is exactly the same.

With respect to different noise levels, compare Figures 3.4(a) and 3.4(b). For the low level noise, the mean of MSE is about twice as large as expected. For the high level noise, the mean of the MSE is only slightly larger.

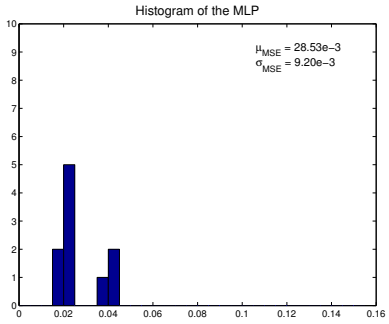
3.2.4 Support Vector Machine

Reproducibility simulations of the SVM are performed using a 1^{st} order spline kernel. Parameters are set as follows:

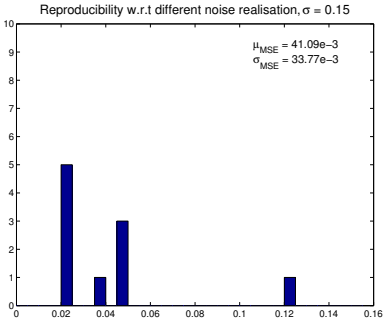
- For noise $\sigma = 0.05$: $C = 7$ and $\varepsilon = 0.05$
- For noise $\sigma = 0.15$: $C = 10$ and $\varepsilon = 0.1$



(a) Input 1, $\sigma = 0.05$

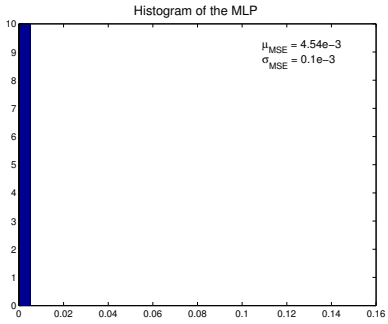


(b) Input 1, $\sigma = 0.15$

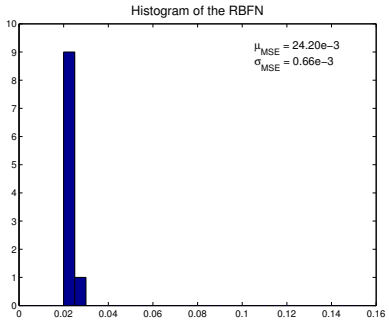


(c) Input 2, $\sigma = 0.15$

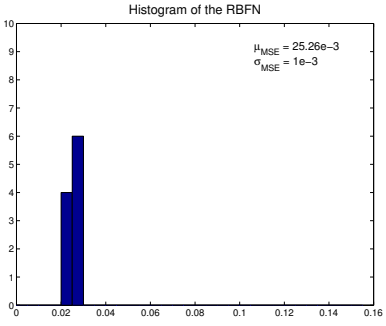
Figure 3.3: Histogram of MSE for the MLP simulation



(a) Input 1, $\sigma = 0.05$



(b) Input 1, $\sigma = 0.15$



(c) Input 2, $\sigma = 0.15$

Figure 3.4: Histogram of MSE for the RBFN simulation

Table 3.1: Number of support vectors for the SVM

Input 1		Input 2
Noise $\sigma = 0.05$	Noise $\sigma = 0.15$	Noise $\sigma = 0.15$
724	700	717
698	679	674
705	658	686
717	695	674
711	676	711
685	686	665
670	648	656
690	673	698
714	641	686
718	650	725

Cherkassky in (Cherkassky and Ma, 2002) proposed a method to determine the parameters for the SVM regression, but the results are not good, i.e. the parameters still need further tuning to get better results. The histograms are shown in Figure 3.5 and the number of support vectors in the model is presented in Table 3.1, where σ denotes the standard deviation of noise.

With respect to different noise realisations, consider Figure 3.5(b). The figure shows that all of the MSEs are in one band and therefore the result can be deemed perfect.

With respect to different training sets, compare Figures 3.5(b) and 3.5(c). It can be seen that all of the MSEs fall into one strip for both results and therefore the result can be deemed *perfect*.

With respect to different noise levels, compare Figures 3.5(a) and 3.5(b). Considering the mean of the MSEs, for both noise levels the values are larger than expected.

3.2.5 Least Square Support Vector Machine

Reproducibility simulations of the LSSVM are performed with a 1^{st} order spline kernel and $\gamma = 1000$, using the sparselssvm function in order to get sparse solution. In this function, data are removed until the performance of the pruned LSSVM is about 95% of the original. The histograms are shown in Figure 3.6. The point that should be noted for these results is that the number of support vectors is the same for all simulations, namely 1356.

With respect to different noise realisations, see Figure 3.6(b). The result can be deemed perfect for all results as all MSE are in one strip in the histograms.

With respect to different training sets, compare Figures 3.6(b) and 3.6(c). In Figure 3.6(c) there is one strip in which eight results are in the same position as in Figure 3.6(b). Therefore the result can be deemed good.

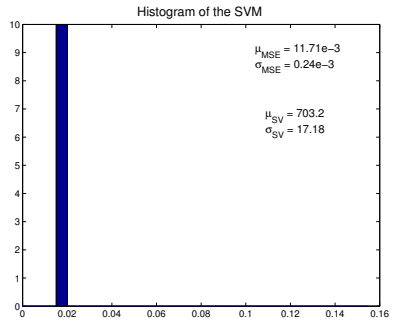
With respect to different noise levels, compare Figures 3.6(a) and 3.6(b). The mean of the MSE for both noise levels are larger than expected.

3.2.6 Key Sample Machine

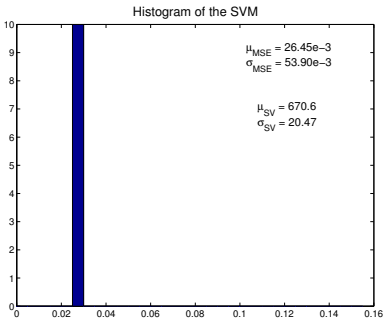
Reproducibility simulations of the KSM use a 1^{st} order spline kernel as the indicator function. The number of support vectors are shown in Table 3.2.

With respect to different noise realisations, see Figure 3.7(b). From this figure, it can be deemed that the result is bad as the MSEs are spread in three strips.

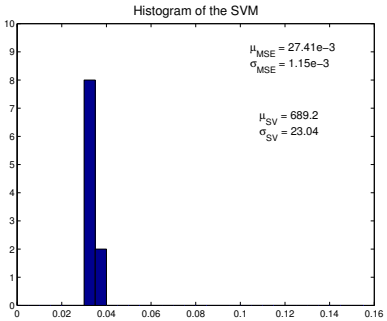
With respect to different training sets, compare Figures 3.7(b) and 3.7(c). It can be deemed that the result is bad as the width of the spread of the MSEs is not the same, the difference is two strips. It is larger



(a) Input 1, $\sigma = 0.05$

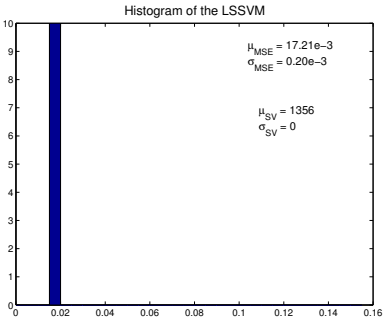


(b) Input 1, $\sigma = 0.15$

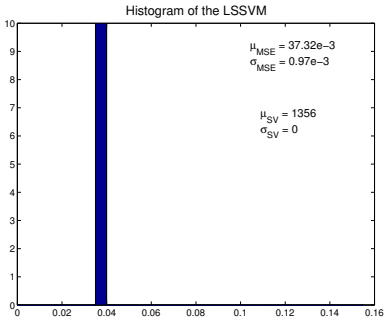


(c) Input 2, $\sigma = 0.15$

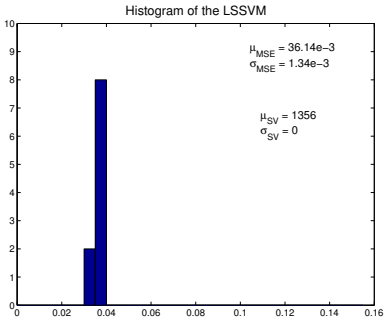
Figure 3.5: Histogram of MSE for the SVM simulation



(a) Input 1, $\sigma = 0.05$



(b) Input 1, $\sigma = 0.15$



(c) Input 2, $\sigma = 0.15$

Figure 3.6: Histogram of MSE for the LSSVM simulation

Table 3.2: Number of support vectors for the KSM

Input 1		Input 2
Noise $\sigma = 0.05$	Noise $\sigma = 0.15$	Noise $\sigma = 0.15$
20	13	13
20	14	14
21	18	13
23	14	13
21	16	8
20	15	20
22	13	13
24	15	14
18	13	9
18	14	13

for Input 2.

With respect to different noise levels, compare Figures 3.7(a) and 3.7(b). For both results the mean of the MSE for both noise levels are larger than expected.

3.2.7 Discussion

The summary of the statistical MSE is shown in Table 3.3 and the number of support vectors in Table 3.4. In the tables, μ is statistical mean and σ is the standard deviation.

In the MLP, different initial weights will result in a different value error correction in each epoch. Therefore, one network is possibly trapped into local minimal while the other is not.

Different error corrections also occur if different noise realisations are applied to a training set. For example, consider a point x_1. Different noise realisations will result in various values of target y_1. So local minima also can occur from different noise realisations.

For different training sets of the same function, the approximation should be the same. In this case, the MLP cannot adapt its free parameters to learn from the training set to yield a similar approximation.

With training of high noise data, the MSE spread is larger than that of the low noise and the spread is not similar for both. Therefore for high noise, the MLP cannot keep the MSE training within a band.

The relatively good result for low level noise in the MLP is related with neurons that act as global approximators. Free parameters can be adapted to handle small variation of the targets. As the variations are larger due to higher noise level, the adaptation of the free parameters are also large. Therefore, it is difficult to have a good prediction with respect to a target without worsening the prediction of the others.

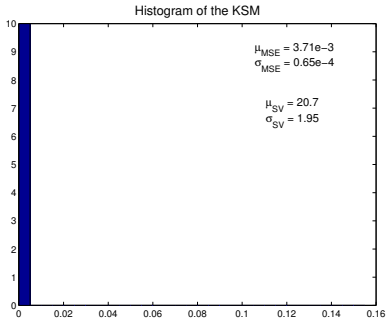
The RBFN in this section can be seen as a LS problem using eight indicator functions, namely the RBFs of fixed parameters. Using this view, it can be understood that the results are consistent for all categories.

Regarding the MSE, for low level noise data the MSEs are much larger than expected.

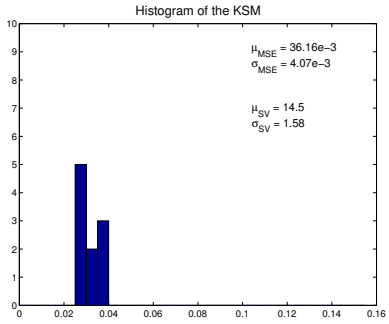
The perfect results from the RBFN come from the visual design method. The centres and the widths of RBFs can be set so that they give a small MSE, therefore the design being optimal. Results for this method are much better than that of the MLP, as it is said in (Haykin, 1999) that the RBFN when designed optimally will outperform the MLP.

Results of the SVM are dependent on how the SVM is tuned. Tuning of the SVM is related with setting the width and the flatness of the tube, i.e. setting the parameters ε and C, and the kernel parameters. However, the kernel used in the simulations, a 1^{st} order spline kernel, has no parameter to be set.

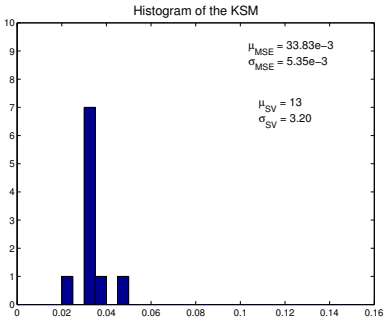
The width of the tube in the SVM, ε , is set in relation with the noise level in the training samples, it should be larger as the noise level increases. The number of support vectors will decrease as ε increases. By keeping ε fixed for all training sets with the same noise distribution, the optimisation process searches



(a) Input 1, $\sigma = 0.05$



(b) Input 1, $\sigma = 0.15$



(c) Input 2, $\sigma = 0.15$

Figure 3.7: Histogram of MSE for the KSM simulation

Table 3.3: Summary of the MSE for the reproducibility simulations

Learning Methods	Input 1				Input 2	
	Noise $\sigma = 0.05$		Noise $\sigma = 0.15$		Noise $\sigma = 0.15$	
	μ_{MSE}	σ_{MSE}	μ_{MSE}	σ_{MSE}	μ_{MSE}	σ_{MSE}
MLP	3.61 e-3	3.23 e-3	28.53 e-3	9.20 e-3	41.09 e-3	33.77 e-3
RBFN	4.54 e-3	0.1 e-3	24.2 e-3	0.66 e-3	25.26 e-3	1.00 e-3
SVM	11.71 e-3	0.24 e-3	26.45 e-3	53.90 e-3	27.41 e-3	1.15 e-3
LSSVM	17.21 e-3	0.2 e-3	37.32 e-3	0.97 e-3	36.14 e-3	1.34 e-3
KSM	3.71 e-3	0.65 e-3	36.16 e-3	4.07 e-3	33.83 e-3	5.35 e-3

Table 3.4: Summary of the number of support vectors for the reproducibility simulations

Learning Methods	Input 1				Input 2	
	Noise $\sigma = 0.05$		Noise $\sigma = 0.15$		Noise $\sigma = 0.15$	
	μ_{SV}	σ_{SV}	μ_{SV}	σ_{SV}	μ_{SV}	σ_{SV}
SVM	703.20	17.18	670.60	30.47	689.20	23.04
LSSVM	1356.00	0	1356.00	0	1356.00	0
KSM	20.70	1.95	14.50	1.58	13.00	3.20

for the minimum distance between the support vectors and the tube, so that the estimation result is bounded in the tube. This means the estimation error will not deviate a lot for different noise realisations.

The tube formed due to the ε-insensitive loss function will depend only on the function to be approximated, and not the input points. Therefore different training sets of the same function will not influence the approximation result.

In the LSSVM, solving the simultaneous linear equations results in a model that uses all training data. The resulting model is not sparse, namely every training data is connected to each other. This learning method gives a very good result for regression of noisy data and different training sets. The trade-off for this is a large number of the support vectors in the resulting model. With a large number of support vectors, overfitting possibly occurs.

The large number of support vectors is possibly obtained by setting the parameter γ large. This γ is related with how the error is to be optimised. The larger the γ, the smaller the MSE and the larger the number of support vectors in the model. This parameter can also be seen as a regularisation parameter. Small γ makes the first term in (2.47), which determines the flatness, more optimised.

The same number of support vectors in each model of the LSSVM comes from the pruning process. In the pruning process, the same number of samples omitted in all training sets always results in the same performance decrease.

The KSM uses a similar approach with the LSSVM, i.e. by solving a set of linear equations. The difference is in the way of simplification of the resulting model. This method is trained using all of the training data and to make a prediction, only data that gives significant error reduction is included. So the model prediction is simple, see Table 3.2, i.e. only using some of the training samples.

In the case of the LSSVM, some training samples are removed for some performance decrease. In the case of a desired small performance decrease, the number training samples that are removed are so small that the number of support vectors used for prediction is much larger compared to that of the SVM and the KSM.

Although the KSM is similar with the LSSVM, the simulation results are different. The differences come from the indicator selection method. Before adding an indicator, the KSM checks to see the significance of the indicator to the approximation. The MSE before the indicator addition is stopped and the number of support vectors are different for different training sets. These differences result in a spread histogram.

Comparing the number of support vectors for the SVM and the LSSVM, it is clear that the KSM has the least. The way it searches for support vectors is better which can be seen from the simulation results. With a small number of support vectors the model of the KSM is simpler and takes less computation time for prediction.

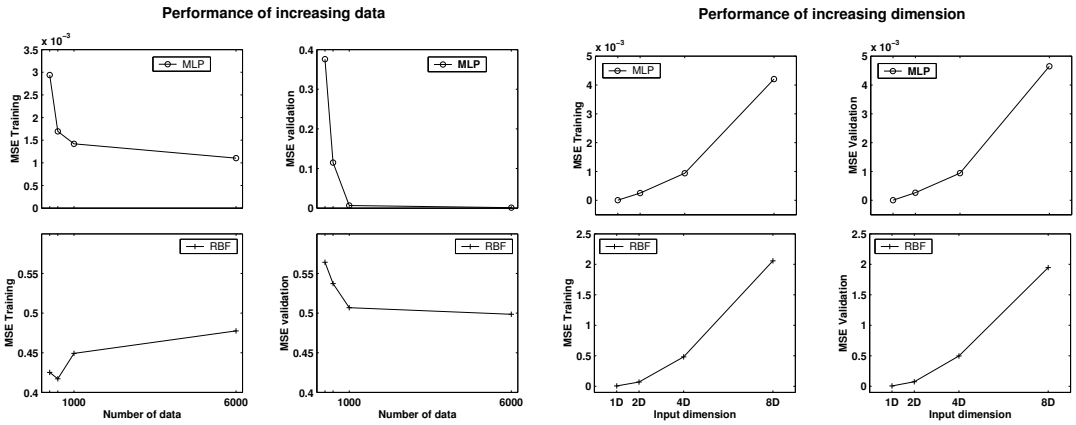
3.3 High Dimensional Inputs Simulation

This section describes the simulation of comparing the performance of the function approximators in dealing with high dimensional inputs. In function approximation theory it is known that to keep the generalisation performance the same while the input dimension increases, the number of data needed for training increases exponentially (Haykin, 1999).

While the simple statement above has a lot of mathematical explanations, the simulation here only tries to investigate the ability of the learning methods to deal with high dimensional inputs. This will be done by observing the influence of an increase in the number of training data and an increase in input dimension to the performance, namely the MSE.

These two simulations are basically the same i.e. changing data density in the input space for both, from low to high in the first simulation by increasing training samples and from high to low in the second by increasing input dimension.

The higher the data density in the input space, the more information is supplied in the learning process so that the approximators are trained better and give better performance during training and validation. For local-approximation-based learning, higher data density means each neuron is supplied with more data to



(a) Increasing samples (b) Increasing dimension

Figure 3.8: Simulation results of the MLP and the RBFN

adapt and generalise so that it is trained better around its coverage.

3.3.1 Simulation Setup

The training data used is synthetic and the function to be approximated is as follows

$$f(x_1, \ldots, x_n) = 2 \sum_{i=1}^{n} \max(\sin 3\pi x_i - \frac{1}{2}, 0), \quad 0 \leq x_i \leq 1 \tag{3.1}$$

where n is the input dimension. Using this function, the input dimension can be controlled and therefore the simulation of increasing input dimension can be performed.

The simulations use 1D, 2D, 4D, and 8D samples. For training, 7000 random samples of each dimension are generated and 1000 samples are separated for validation. The samples are not corrupted with noise, thus observation can be made only to investigate the ability of the learning methods to handle high dimensional inputs, and not noise.

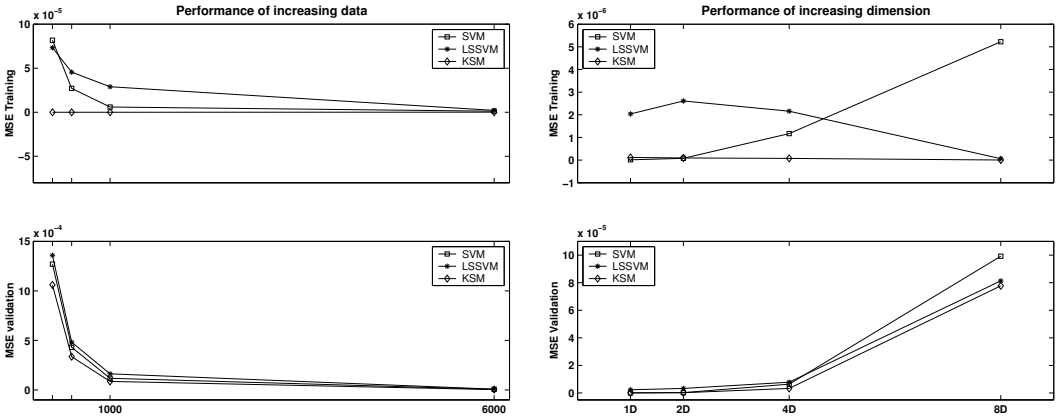
For the first simulation, only 4D samples are used, with a number of 250, 500, 1000, and 6000 samples respectively. The learning methods are trained using these samples and then the training and validation performances are recorded. In the second simulation, the learning methods are trained using all of the 6000 samples of 1D, 2D, 4D, and 8D training sets and their training and validation performances are recorded as well.

The training results are summarised in Figures 3.8(a) to 3.9(b). The separation of the figures is based on the difference of the MSE value since its difference is large among all learning methods. Therefore the shape of each graph can be clearly seen and the graph can be evaluated more easily. Numerical results of the simulations are shown in tables.

3.3.2 Multilayer Perceptron

The MLP used in this simulation has the following parameters:

- Number of hidden neuron: 20
- Learning coefficient: 0.05
- Momentum coefficient: 0.5
- Logistic activation function



(a) Increasing samples (b) Increasing dimension

Figure 3.9: Simulation results of the SVM, the LSSVM, and the KSM

Table 3.5: The MLP simulation result

(a) Increasing data

#Data	Mean Square Error	
	Training	Validation
250	0.0029	0.3758
500	0.0017	0.1150
1000	0.0014	0.0066
6000	0.0011	0.0011

(b) Increasing dimension

Input dimension	Mean Square Error	
	Training	Validation
1D	3.303 e-5	3.465 e-5
2D	2.513 e-4	2.463 e-4
4D	9.376 e-4	9.423 e-4
8D	4.204 e-3	4.649 e-3

- Maximum epoch: 2000

Numerical results are shown in Table 3.5.

In Table 3.5(a), the first simulation shows that as the number of training samples increase, training and validation error decrease. If we use the training error of 250 samples as a reference, it can be calculated that the training error of 500 samples is about 0.6 times of 250 samples, the error of 1000 samples is about 0.5 times, and the error of 6000 samples is about 0.14 times. For the validation error they are about 0.3 times, 0.02 times, and 0.003 times for 500, 1000, and 6000 samples respectively.

In Table 3.5(b), the second result shows that the training and validation error increase as the input dimension increases. If we use the training error of 1D input as a reference, it can be calculated that the training error of 2D input is about 6 times of 1D input, the error of 4D is 28 times, and the error of 8D is 127 times. For the validation error they are about 7.2 times for 2D, 27 times for 4D, and 133 times for 8D input.

3.3.3 Radial-Basis Function Network

The RBFN uses 30 Gaussian RBFs where the centres are randomly fixed and the widths are uniformly 0.35, except in the case of 1D input. For 1D input, the centres are set visually with 9 neurons and their widths are 0.35 and 0.45. In the simulation of 2D input and higher, ten networks are constructed to be trained. The final result is the mean of the MSE of all networks. Numerical results are shown in Table 3.6.

In Table 3.6(a), it can be seen that in general, adding more samples to the network only has a small influence to the decrease in error. However, it still can be seen that the training error decreases form 250 to 500 training data, but then increases as the number of samples increase. On the other hand, the validation

Table 3.6: The RBFN simulation result

(a) Increasing samples

#Data	Mean Square Error	
	Training	Validation
250	0.4252	0.5642
500	0.4171	0.5372
1000	0.4492	0.5069
6000	0.4776	0.4985

(b) Increasing dimension

Input dimension	Mean Square Error	
	Training	Validation
1D	5.7128 e-3	5.5577 e-3
2D	0.6944	0.7367
4D	0.4804	0.5065
8D	2.0580	1.9468

Table 3.7: The SVM simulation result

(a) Increasing samples

#Data	Mean Square Error		#SV
	Training	Validation	
250	8.1746 e-5	1.2694 e-3	239
500	2.7140 e-5	4.3062 e-4	468
1000	5.9445 e-6	1.1785 e-4	867
6000	1.1701 e-6	6.3866 e-6	2920

(b) Increasing dimension

Input Dimension	Mean Square Error		#SV
	Training	Validation	
1D	1.0934 e-8	1.7413 e-8	1235
2D	7.7343 e-8	2.2445 e-7	1752
4D	1.1701 e-6	6.3866 e-6	2920
8D	5.2262 e-6	9.9209 e-5	4523

error decreases as the number of samples increase.

In Table 3.6(b), it can be seen that both the training and validation error increase with an increase in input dimension. The increases are so large that it is larger than 1 for 8D input. If we use the training error of 1D input as a reference, the training error of 2D input is about 121 times of 1D input, the error of 4D is 84 times, and the error of 8D is 360 times. For the validation error they are about 130 times for 2D, 90 times for 4D, and 350 times for 8D input.

3.3.4 Support Vector Machine

In this simulation, the SVM uses a 1^{st} order spline kernel and its parameters are set by trial and error. The parameters are chosen to give a small error and the number of support vectors are as small as possible. By trial-and-error, it is possible that the parameters are not optimal.

For the increasing input dimension simulation, the C parameter is set to 2 and ε is set to 5 e-6, 5 e-4, and 5 e-3, for 1D, 2D, and 4D data respectively, and C is set to 10 and ε is set to 0.2 for 8D data. For the increasing number of data simulation, C is set to 2 and ε is set to 5e-3, and these parameters are made the same for all number of samples. Numerical results are shown in Table 3.7.

Table 3.7(a) shows that training and validation error decrease as the training number of samples increase. If we use the training error of 250 samples as a reference, it can be seen that the training error of 500 samples is about 0.3 times of 250 samples, the error of 1000 samples is about 0.07 times, and the error of 6000 samples is about 0.015 times. For the validation error they are about 0.35 times, 0.08 times, and 0.005 times for 500, 1000, and 6000 samples respectively. The percentage of training samples used as support vectors are, 95.6% for 250, 93.6% for 500, 86.7% for 1000, and 29.2% for 6000 training samples.

Table 3.7(b) shows that the error increases as the input dimension increases. If we use the training error of 1D input as a reference, the training error of 2D is about 7.5 times of 1D input, the error of 4D is 110 times, and the error of 8D is 522 times. For the validation error they are about 13 times for 2D, 375 times for 4D, and 5800 times for 8D input. The percentage of the training samples used as support vectors are 20.6%, 29.2%, 48.7%, and 75.4% for 1D, 2D, 4D, and 8D input respectively.

Table 3.8: The LSSVM simulation result

(a) Increasing samples

#Data	Means Square Error		#SV
	Training	Validation	
250	7.3274 e-5	1.3587 e-3	229
500	4.5489 e-5	4.8236 e-4	454
1000	2.9030 e-5	1.6207 e-4	905
6000	2.1604 e-6	7.8057 e-6	5417

(b) Increasing dimension

Input Dimension	Means Square Error		#SV
	Training	Validation	
1D	2.0381 e-6	2.3067 e-6	5417
2D	2.6093 e-6	3.2730 e-6	5417
4D	2.1604 e-6	7.8057 e-6	5417
8D	6.3118 e-8	8.1334 e-5	5701

Table 3.9: The KSM simulation result

(a) Increasing samples

#Data	Means Square Error		#SV
	Training	Validation	
250	1.4931 e-10	1.0597 e-3	247
500	8.9291 e-10	3.3580 e-4	490
1000	4.8132 e-9	8.6121 e-5	949
6000	7.5212 e-8	3.2616 e-6	3344

(b) Increasing dimension

Input Dimension	Means Square Error		#SV
	Training	Validation	
1D	1.1884 e-7	1.3981 e-7	87
2D	9.4890 e-8	1.7763 e-7	598
4D	7.5212 e-8	3.2616 e-6	3344
8D	4.2158 e-9	7.7724 e-5	5805

3.3.5 Least Square Support Vector Machine

In this simulation, the parameter γ of the LSSVM is set to 50. The kernel used for this simulation is also a 1^{st} order spline. Numerical results are shown in Table 3.8.

Table 3.8(a) shows that both training and validation error decrease as the training samples increase. If we use the training error of 250 samples as a reference, it can be seen that the training error of 500 samples is about 0.6 times of 250 samples, the error of 1000 samples is about 0.4 times, and the error of 6000 samples is about 0.3 times. For the validation error they are about 0.4 times, 0.1 times, and 0.006 times for 500, 1000, and 6000 samples respectively. The percentage of the training samples used as support vectors are 91.6%, 90.8%, 90.5% and 90.28%, so almost all training samples are used in the model.

Table 3.8(b) shows that the training error firstly increases and then decreases while the validation error always increases. If we use the training error of 1D input as a reference, the training error of 2D is about 1.3 times of 1D input, the error of 4D is 1.1 times, and the error of 8D is 0.03 times. For the validation error they are about 1.4 times for 2D, 3.4 times for 4D, and 35 times for 8D input. The percentage of the training samples used as support vectors are 90.28% for 1D, 2D, and 4D, and 95% for 8D input.

3.3.6 Key Sample Machine

By default, the simulation for this method uses a 1^{st} order spline and the noise prediction σ is set to 0.0001. The numerical results are shown in Table 3.9.

In Figure 3.9(a), the training error seems linear but this is because of the small error, but from Table 3.9(a) it can be seen that the training error increases as the number of training samples increase. If we use the training error of 250 samples as a reference, it can be seen that the training error of 500 samples is about 5.9 times of 250 samples, the error of 1000 samples is about 32 times, and the error of 6000 samples is about 500 times. For the validation error they are about 0.3 times, 0.08 times, and 0.003 times for 500, 1000, and 6000 samples respectively. The percentage of training samples used as support vectors are, 98.8% for 250, 98% for 500, 94.9% for 1000, and 55.7% for 6000 training samples.

Table 3.9(b) shows that the performance behaves the other way around, training error decreases while validation error increases. If we use the training error of 1D input as a reference, the training error of 2D is

about 0.8 times of 1D input, the error of 4D is 0.6 times, and the error of 8D is 0.04 times. For validation error they are about 1.2 times for 2D, 23 times for 4D, and 555 times for 8D input. The percentage of the training samples used as support vectors are 1.45%, 9.97%, 55.73%, and 96.75% for 1D, 2D, 4D, and 8D input respectively.

3.3.7 Discussion

The errors in the MLP simulation are the second worst and it is shown that the MLP is sensitive to data density in the input space. Both the increasing number of samples and the increasing input dimension simulations agree that the more dense data in the input space the better the approximation. This can be seen from the nonlinear change in error as the training samples or input dimension are increased.

In the MLP point of view, the increasing number of samples simulation, which from this point is called *the first simulation*, is to give an increasing number of training samples but the number of free parameters are kept fixed, so the free parameters are adapted by the increasing number of samples. On the other hand, in the increasing input dimension simulation, which from this point is called *the second simulation*, the free parameters are made larger while the supplied training samples are fixed, thus worse approximations are yielded. The worse results come from the decreasing number of samples used for adapting each of the free parameters.

Another observation from the first simulation of the MLP is that the number of hidden neurons is sufficient so that there is no overfitting occuring especially when using the small number of training samples compared to the free parameters. The overfitting can be seen from the decreasing training error while the validation error increases, and at some points both errors tend to be constant.

The behaviour of the training result from the first simulation of the RBFN is not predictable, but the validation result behaves just like the others. The result from the second simulation behaves similarly with that of the MLP, which means that approximation by the RBFN also needs more RBFs to cover the higher dimensional input space.

With respect to the magnitude of the MSE the RBFN is the worst. This comes from the fixed random RBF centres and uniform widths design which is not optimal. Using predictable data like used here, the centre position can be arranged as a grid, but this approach applies only when the input dimension is small. The higher the input dimension, the larger the number of RBFs in the grid. For example, with a 3-by-3 grid and 4D input, the number of neurons is $3^4 = 81$ but increasing it to a 4-by-4 grid the number of neurons will be $4^4 = 256$. This number is too many and leads to overfitting. Besides, the simulations performed here need a fixed number of neurons to show the effect of increasing input dimension and therefore the fixed random centre training is more suitable.

A remark can be made on the design of the RBFN. Designing a RBFN is more difficult especially when the input dimension is more than two. For one or two dimensional input, a RBFN can be designed visually by plotting the training data and then placing the RBFs on the critical points, namely the points where the plot turns. When the input dimension is more than two, other design methods, for example in (Chen et al., 1991) can be considered.

For the SVM, the resulting MSEs are much smaller than that of the MLP and the RBFN although the error tendency is similar. In the first simulation, the SVM gives a similar result behaviour with that of the MLP but the error scale is much smaller.

Now consider the synaptic weights of the MLP as the model parameters equal to the support vectors in the SVM. The SVM's model has more model parameters which can be seen from the number of support vectors after training. This is one difference of the SVM, or the support vector-based method in general, with the MLP and the RBF. In the SVM method, model parameters are not known until the SVM is trained. And those parameters are sensitive to the design parameters, C and ε. On the contrary, the model parameters of the MLP and the RBF are determined beforehand.

The error behaviour of the first simulation of the LSSVM is the same as all of the previous learning methods, but the magnitude is only comparable to the SVM. In the second simulation, the training error behaves differently, it decreases while the others increase. However it is still not experimentally known whether it still decreases if a higher input dimension is used.

The model of the LSSVM is relatively more complex than that of the SVM, since the number of support vectors of the LSSVM is relatively large and constant while in the SVM the number of support vectors increase as the input dimension increases. The constantly large support vectors yielded from the pruning method in the LSSVM, is explained in Section 2.7.1. If a small accuracy is desired, the training time is longer but the model is simpler.

The KSM simulation results are relatively better than that of the other learning methods. This can be seen from Figures 3.9(a) and 3.9(b) where the graph of the KSM lies lower. From Table 3.9(a) it can be seen that the training error of the KSM behave differently, it increase as the training samples increase while the error of the other learning methods decrease. The training error of the KSM in the second simulation has similarity with that of the LSSVM, it decreases as input dimension increases but it is not known empirically whether it will keep decreasing. This decrease can be explained from more samples included in the model.

The model of the KSM is the simplest for 1D and 2D while for a higher dimension it is comparable to the other learning methods. This simpler model comes from the indicator selection applied in the KSM. It can be concluded then that the higher the input dimension, the more data are significant for prediction and this explains why the support vectors increase for higher input dimension in the SVM and the LSSVM.

3.4 Off-line Approximation of Linear Motor

This section presents the last simulation performed to compare the learning methods. After assessment of the learning methods using synthetic data, their performance is tested using real-world data. This simulation is different from both the previous simulations in that no particular effect is intended to be observed, except to see the behaviour of the learning methods in a real-life setup.

The plant is a permanent magnet synchronous linear motor. This kind of motor is often designed to performed linear motions with sub-millimeter accuracy. The motor configuration consists of a base plate covered with permanent magnets and a sliding part, a translator that holds the electric coils and their iron cores. Applying a three phase current to three adjoining coils of the translator a sequence of attracting and repelling forces between the poles and the permanent magnets will be generated. This results in a relative motion between the base plate and the translator. The basic behaviour of the motor is that of a moving mass. An illustration of the motor is shown in Figure 3.10.

During its operation there are disturbances that work on the translator (Velthuis, 2000):

- Ripple forces that are caused by the following:
  - A strong magnetic interaction between the permanent magnets on the base plate and the iron cores, on which the translator coils are mounted, called *Cogging*. This force tries to align the magnets and iron cores to stable ('detent') positions on the translator. The cogging force depends only on the relative position of the translator with respect to the magnets.
  - Errors in the commutation, i.e. the way the current is applied to the coils. Calculation of this disturbance requires very detailed information about the plant.
- Friction force in the ball bearings between the translator and the guiding rails on which it rests.

In this section, not all the approximators are tested. From previous simulations, the performance of the MLP and the RBFN are not good enough compared to the other three.

3.4.1 Simulation Setup and Result

The simulation is performed off-line and the learning signals used for training are as follows. The inputs are the position reference, its time derivative (speed reference), and its second derivative (acceleration reference). The target is the feedback force which is the output of the feedback controller. The number of samples used is 32690 which is divided into two training set consisting of 18000 samples and the rest is used in the validation set. The division of samples does not follow any preferences, except considering the length of training time.

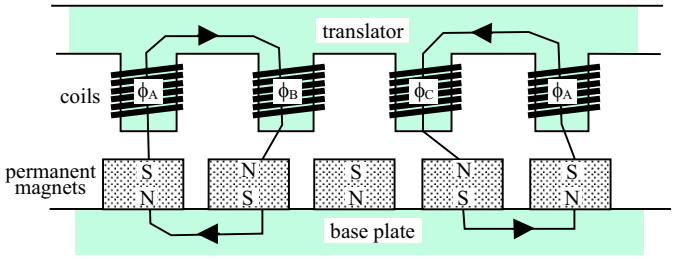


Figure 3.10: Working principle of the permanent magnet synchronous linear motor (Velthuis, 2000)

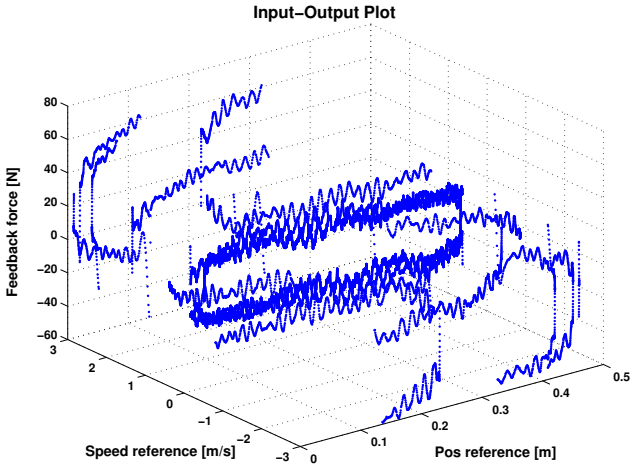


Figure 3.11: Plot of part of training samples, target vs position and speed reference

Although it is not a complete description, the target is plotted versus the position reference and the velocity reference in Figure 3.11 to get an idea about the nonlinearity of the data.

These samples are applied to the SVM, the LSSVM, and the KSM all with a 1^{st} order spline kernel. Parameter setting for each approximator is as follows:

1. SVM: $C = 10$ and $\varepsilon = 0.019$
2. LSSVM: $\gamma = 10$
3. KSM: $\sigma = 0.0005$

The parameters used here may not be optimal since there are time constraints for simulation. Parameter C for the SVM and γ for the LSSVM cannot be set larger and σ in the KSM can be set smaller to obtain a better approximation, however the result is comparable, the performances are almost the same.

Table 3.10: Approximation result of linear motor

Method	Mean Square Error		#SV
	Training	Validation	
SVM	9.9063 e-3	9.0368 e-3	5527
LSSVM	9.8414 e-3	9.1375 e-3	6811
KSM	8.5000 e-3	7.9000 e-3	2609

Table 3.10 shows that the error difference, both for training and validation, for all learning methods is not large. However, the KSM has the smallest errors.

While the error difference is not significant, the number of support vectors in the models differ considerably. In this case the KSM has the smallest number of support vectors which is less than half of those in the SVM and the LSSVM.

3.4.2 Discussion

In this simulation real-world data is used with an unestimated noise level. Nevertheless, the parameters of each method can be set to achieve relatively good results (the MSE less than 1%) without directly relying on the noise estimation. For the learning methods used here for example, the value of ε in the SVM is related with the noise level. In the KSM, the estimated noise level is used as a parameter in the stopping criteria of indicator addition.

Although the training plot is not complete, it shows 2D input instead of 3D, and it can be seen that the approximated function is very nonlinear. Despite the nonlinearity, the KSM is able to use a very small number of support vectors in its model which has a comparable validation performance. This shows that forward selection with statistical indicator inclusion stopping has an advantage over the SVM and the LSSVM in obtaining a simpler model.

3.5 Summary

In this chapter, the empirical comparison of learning methods by simulation is addressed. Each method is compared with respect to its ability to yield similar results with various training samples of the same function (reproducibility). Another simulation is about how well the methods can handle high dimensional inputs. This is important since the nature of its application in the LFFC setting will be high dimensional. After simulation with synthetic data, a simulation is performed using real-world data. This simulation is performed off-line using data taken from an experiment of a linear motor. Each simulation is followed by a discussion of the results.

Chapter 4

Conclusions and Recommendations

This chapter presents the conclusions that have been drawn from this thesis and gives recommendations for future work.

4.1 Conclusions

All of the learning methods presented in this thesis can be categorised into two classes. The first is a class that has fixed free parameters, namely the MLP and the RBFN. The second is a class that consists of support vector-based or kernel-based learning methods. Both classes have a basic difference when dealing with the feature space. The first class uses the feature space directly, while the second uses it implicitly by the dot product of the mapped samples which leads to a distinct optimisation problem.

Categorisation can also be made based on the parameters in the model, the first is a model in which the number of parameters is known beforehand and the second is a model in which the parameters are known afterwards. The MLP and the RBFN presented in this thesis belong to the first class since the hidden neurons must be determined before training, while the other learning methods, the SVM, the LSSVM, and the KSM, belong to the second class.

From the reproducibility simulation, it has been shown that the MLP with backpropagation method is sensitive to initial weights. Sensitivity is influenced by the optimisation method, where different results occur for different networks, and the random initial weights which possibly leads to local minima.

Another result from the reproducibility simulation is that the RBFN gives the best result in terms of a consistent MSE and it produces the smallest error spread of the training sets with different realisation and level of gaussian noise. The SVM, the LSSVM, the KSM, and the MLP follow in second, third, fourth, and fifth respectively. This result also shows that optimal visual design of the RBFN produces an RBFN that can surpass other methods.

For support vector-based methods, the KSM has advantages over the SVM and the LSSVM for the number of parameters. In all simulations of reproducibility the KSM gives the least number of support vectors with comparable error. This shows that the forward search along with the statistical test to stop indicator addition has an advantage over the SVM and the LSSVM.

In dealing with high dimensional inputs, it is shown that increasing input space dimension while the number of samples is kept fixed will increase the validation error. The behaviour of both classes in this case is different. In the support vector-based methods the error increase is not as large as in the fixed parameter methods. However, the error increase in the support vector-based methods is followed by an increase in the model parameters, while the fixed parameter methods cannot compensate for the input dimension increase.

Another aspect can be observed among the support vector-based methods. It can be seen that the KSM gives the best result for the high dimensional inputs simulation. Its error is always smaller than that of the other methods with a smaller or comparable number of parameters.

From the approximation of the linear motor data, the KSM gives a comparable MSE but its model uses

the smallest number of support vectors. In this simulation, the result is relatively good inspite of the fact that the noise level is not known and the parameter setting is done by trial-and-error.

With respect to the number of support vectors, the LSSVM and the KSM give very different results. Despite much similarity between both learning methods, the different pruning method between them leads to a big difference in the parameters in the final model. Pruning implemented in the LSSVM cannot give a significant reduction in support vectors while in the KSM it can.

4.2 Recommendations

From this thesis there are some recommendations for future work. These are:

1. All of the learning methods that are compared are off-line methods. It can be extended to on-line methods.
2. Support vector-based methods that are assessed all use a 1^{st} order spline kernel. It can be extended to see the influence of the kernel function to the approximation result.
3. Comparison can be extended to other aspects of learning method.

Bibliography

Chih-Chung Chang and Chih-Jen Lin. *LIBSVM: a library for support vector machines*, 2001. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

S. Chen, C. F. N. Cowan, and P. M. Grant. Orthogonal least squares learning algorithm for radial basis function networks. *IEEE Transactions on Neural Networks*, 2(2):302–309, March 1991.

Vladimir Cherkassky and Yunqian Ma. Selection of meta-parameters for support vector regression. In *Proceedings of the International Conference on Artificial Neural Networks (ICANN)*, pages 687–693, 2002.

N. Cristianini and J. Shawe-Taylor. *An Introduction to Support Vector Machines*. Cambridge University Press, Cambridge, UK, 2000. For further information go to http://www.support-vector.net.

Bas J. de Kruif. Function approximation for learning control, applied in learning feed forward, 2003. Ph. D Thesis preprint.

Bas J. de Kruif and T. J. A. de Vries. Support-vector-based least squares for learning non-linear dynamics. 41^{st} *IEEE Conference on Decision and Control, Las Vegas, USA*, pages 10–13, December 2002.

Theo J. A. de Vries, Wubbe J. R. Velthuis, and Job van Amerongen. Learning feed-forward control: A survey and historical note. 1^{st} *IFAC conference on Mechatronic Systems, Darmstadt, Germany*, pages 949–954, September 2000.

Norman R. Draper and H. Smith. *Applied Regression Analysis,* 3rd ed. John Wiley, New York, 1998.

Joydeep Ghosh and A. Nag. *An Overview of Radial Basis Function Networks*, chapter 1, pages 1–30. Physica-Verlag, New York, 2001.

Gene H. Golub and Charles F. Van Loan. *Matrix Computations*. Johns Hopkins Studies in the Mathematical Sciences. The Johns Hopkins University Press, Baltimore, MD, USA, third edition, 1996.

Simon Haykin. *Neural Networks A Comprehensive Foundation*. Prentice Hall, New Jersey, 2nd edition, 1999.

M. Kawato, K. Furukawa, and R. Suzuki. A hierarchical neural-network model for control and learning of voluntary movement. *Biological Cybernetics*, 57:169–185, 1987.

Ron Kohavi. A study for cross-validation and bootstrap for accuracy estimation and model selection. *International Joint Conference on Artificial Intelligence (IJCAI)*, 1995.

G. W. Ng. *Application of Neural Networks to Adaptive Control of Nonlinear Systems*. Research Study Press Ltd., Somesrset, England, 1997.

Lutz Prechelt. Proben1 – a set of neural network benchmark problems and benchmarking rules. Technical Report 21/94, Fakultät Für Informatik, Universität Karlsruhe, September 1994. From ftp://ftp.ira.uka.de/pub/uni-karlsruhe/papers/techreports/1994/1994-21.ps.gz.

Alex J. Smola and B. Schölkopf. A tutorial on support vector regression. NeuroCOLT Technical Report NC-TR-98-030, Royal Holloway College, University of London, UK, 1998. URL http://www.kernel-machines.org/papers/tr-30-1998.ps.gz.

Johan A. K. Suykens. Least square support vector machine. Lecture in NATO-ASI Learning Theory and Practice, July 2000. URL http://www.esat.kuleuven.ac.be/sista /natoasi/suykens.pdf.

Johan A. K. Suykens, T. Van Gestel, J. De Brabanter, B. De Moor, and J. Vandewalle. *Least Squares Support Vector Machines*. World Scientific Pub. Co., Singapore, 2002. ISBN 981-238-151-1. URL http://www.esat.kuleuven.ac.be/sista/lssvmlab/book.html.

Wubbe Jan Velthuis. *Learning Feed-Forward Control - Theory, Design and Applications*. Ph. d thesis, University of Twente, Enschede, 2000.