Optimal Allocation of Information Granularity to the Inputs of Granular Neural Networks

by

Elaheh Akhoundi

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science

in

Software Engineering and Intelligent Systems

Department of Electrical and Computer Engineering

University of Alberta

© Elaheh Akhoundi, 2014

Abstract

In this thesis, we propose a design process to construct granular neural networks with granular inputs and numeric network parameters. The proposed granular network is formed on the basis of a numeric neural network whose inputs are augmented using probabilistic information granules. The design problem is formulated as an optimization problem which aims to allocate a given level of information granularity to the inputs of the network such that the specificity of the network outputs gets maximized. The resulting optimization problem is solved analytically and the derived solution determines the optimal granularity levels corresponding to the input features of the granular neural network. The proposed design process is then used to construct granular neural networks for several synthetic and real datasets.

Dedicated to My Beloved Husband, Ali

Acknowledgements

I use this opportunity to express my gratitude to everyone who supported me to bring my work to a successful completion.

First of all, I owe many thanks to my supervisors, Professor Witold Pedrycz and Professor Marek Reformat for supervising my research in two years of my M.Sc. program and for all their invaluable assistance and guidance.

I would also like to thank the members of my evaluation committee: Professor Petr Musilek from the Electrical and Computer Engineering Department and Professor Josef Szymanski from the Civil and Environmental Engineering Department of the University of Alberta. In addition, I thank Pinder Bains for organizing the thesis defence session.

Finally, I wish to express my love and gratitude to my parents for their endless love and my most heartfelt thanks go to my husband Ali for his endless love and support. Without his support, trust and encouragement, I would not have been able to complete my thesis successfully.

Table of Contents

List of Tables

List of Figures

List of Symbols

Symbol	Definition	First Use
n	Number of input features	12
y	Numeric output	12
g	Input-output relationship of a NN	12
$\mathbf{x}$	Numeric input vector	12
$E(\cdot)$	Expectation operator	13
$\mathbf{X}$	Granular input vector	13
$\boldsymbol{\sigma^2}$	Variance vector of input random variables	13
Y	Output random variable	14
σ_y	Variance of output random variable	14
M	Number of input-output data points	14
I	Average of output variances	15
ε	Total granularity level	16
P_X	Probability mass function	17
H	Entropy of a discrete random variable	17
h	Entropy of a continuous random variable	17
p_X	Probability density function	17
$\mathcal{H}$	Hessian matrix	20

List of Abbreviations

Chapter 1

Introduction

Neural Networks (NNs) form a significant class of nonlinear and adaptable systems and are widely used in numerous areas of application. A NN acquires knowledge from its environment in a learning process and adapts its interneuron connection weights (synaptic weights) to the changes in the environment [1]. When the NN is properly trained, it generalizes the knowledge obtained to the inputs it has not encountered before and produces reasonable corresponding outputs [1]. The fact that NNs "learn by example" makes them appealing candidates to be used to solve complex problems where we have incomplete understanding of the underlying systems but training data is readily available [2]. As a result, NNs have been widely used in different application areas including time series prediction, function approximation, pattern recognition, speech synthesis and clustering [1, 2].

1.1 From Numeric to Granular Neural Networks

Traditionally, NNs were considered as numeric constructs which realize a certain nonlinear mapping [3, 4]. In this setting, all the networks parameters, inputs and the resulting outputs were considered to be numeric. In other words, one could essentially view a NN as a numerical mapping from a numeric input space to a numeric output space. In practice, however, it is very useful to generalize the application of NNs to *granular mappings*. By constructing granular mappings, one takes into account the fact that the precise numeric outputs resulting from numeric models are not realistic, since numeric models cannot ideally represent the phenomena they are dealing with [5]. So, instead of numeric values, one can use *information granules* in building models, in order to quantify the lack of numeric precision and the limited knowledge about the underlying phenomena [6].

Information granules can be viewed as linked collections of objects (like data points) that are drawn together because of their similarity, indistinguishability or functionality [4].

One can characterize an information granule by its specificity. Specificity is a measure to quantify how detailed (specific) an information granule is and is related to the number of elements in the granule [5]. A higher specificity implies a smaller information granule which contains a smaller number of elements in it.

There are several methods to construct granular mappings and to quantify information granularity. Interval analysis, fuzzy sets and probability theory are among the most commonly-used approaches for information granulation [4]. In these approaches, one replaces the numeric parameters to be granulated with intervals, fuzzy sets and random variables, respectively.

To realize a granular mapping, one needs to determine how to measure the goodness of the mapping. There are two criteria to determine how good a granular mapping is, the coverage criterion and the specificity criterion. Based on the coverage criterion, one uses a certain inclusion measure to quantify the extent to which target values are included in the corresponding output information granules. In case when the specificity criterion is used, we measure the specificity of the output information granules to determine the goodness of the granular mapping. The more specific the output information granules are, the better the mapping is [5, 6]. Once we selected the criterion to measure the goodness of a granular mapping, we can formulate the problem of allocation of a certain level of information granularity to the parameters or to the input dimensions as an optimization problem. The optimization problem aims to either maximize the overall inclusion level or the overall specificity level of the mapping based on the criterion selected [5, 6].

1.2 Motivation

Analysis of granular mappings is useful in several applications. This thesis in particular, focuses on the application of granular mappings for sensitivity analysis of NNs. Generally speaking, in sensitivity analysis, we are interested in the behavior of the outputs of a system when either its parameters or inputs deviate from their expected numeric values. In other words, sensitivity analysis determines which inputs or parameters of a model are the main drivers of the model, and it can be used to design systems with more reliable outputs. In case of NNs, we aim to know that variations in which input features or which interneuron connection weights lead to more significant changes in the NN outputs. The answer to this question gives a very useful insight on how precisely the inputs or parameters of a NN must be determined or estimated.

The case in which the specificity criterion is used to optimize the granularity allocation to the mapping parameters can be viewed as a version of sensitivity analysis, where our goal is to determine how the different parameters of a mapping can be modified (varied) with as limited as possible impact on the specificity of its outputs. By designing a granular mapping based on the specificity criterion, we obtain the optimal granularity allocation to the parameters or the inputs. The parameters which are given lower granularity levels (lower specificity) in the optimization process are those who exhibit lower sensitivity. Therefore, a precise estimation of their values is not critical. Similarly, using the specificity criterion to determine the granularity levels allocated to the inputs makes it possible to rank the input dimensions in order of the output sensitivity to their variations. This provides us with an insight on how precisely each one of the inputs of the mapping must be specified to have accurate outputs. This is particularly significant in decision making models, where the input values need to be estimated based on a limited amount of information and resources, and it is extremely helpful to identify how precisely an input value must be estimated [5, 6].

1.3 Objectives and Summary of Contributions

In this thesis, we construct granular mappings for NNs by allocating a certain level of information granularity to the input dimensions of the network. In other words, the parameters of the network (weights, biases, activation function parameters) remain numeric while its inputs become granular. We call the resulting NN a Granular Neural Network (GNN) . In our analysis, we use the specificity criterion to derive the optimal allocation of a predetermined granularity level to the input dimensions. As was mentioned in Sec. 1.2, the solutions to this problem are particularly useful in sensitivity analysis of NNs to variations in their inputs. Our approach in this thesis is to first train a numeric NN using the training data and then granulate its numeric inputs using independent random variables from the same family of distributions (like normal distributions or uniform distributions). In this work, we allow different input random variables to be assigned different granularity levels, but we assume that their distributions are symmetric around their mean values.

In this thesis, we formulate the problem of allocation of information granularity to the inputs of a GNN as a constraint optimization problem using the specificity criterion. We justify the use of entropy to quantify the granularity level associated with the input random variables and we use it to formulate the constraint of the optimization problem. We make the resulting optimization problem more mathematically tractable by providing an approximation to its objective function. The solution to the resulting optimization problem is derived which determines the standard deviations of the input random variables. Optimal standard deviations are given in terms of the parameters of three different NN architectures, namely Multi-Layer Perceptron (MLP) with one and two hidden layers and Radial Basis Function Network (RBFN) . In the experimental part of this thesis, we use our theoretical results to design GNNs using synthetic and real data sets. For those data sets, we investigate the relationship between the granularity level allocated to each input dimension and the variability of the output along that dimension. Based on the experimental evidences we obtain, we conclude that lower granularity level (higher standard deviation) is allocated to input dimensions that we there are lower output variations along them. We also compare the optimal standard deviations when training and testing data are used to derive the optimal allocations and we show that the results in both cases are almost identical. We investigate the effect of the parameters of the GNN architectures (like the number of neurons in the network) on the resulting optimal values of standard deviations, and we show that as far as the GNNs are trained properly, their internal parameters have no significant effect on the optimal standard deviation values. Finally, we investigate the effect of perturbing the standard deviations from their optimal values on the specificity level of the GNN output.

1.4 Thesis Organization

This thesis is organized in five chapters. In Chapter 2, we provide the background knowledge and review some of the existing works in the GNN literature. In Chapter 3, we mathematically formulate our optimization problem and derive the solutions to that problem for different GNN architectures. In Chapter 4, we present and analyze experimental results obtained by applying our theoretical findings to several synthetic and real data sets. Chapter 5 reviews our contributions and introduces potential future works.

Chapter 2

Background and Literature Review

2.1 Neural Networks

Artificial neural networks are significant adaptable systems that have the ability to learn from data examples [1]. Once they are trained, they can use the knowledge obtained through the learning phase to predict the outputs for unseen data. NNs are parallel distributed computational models made up of highly interconnected simple processing units [1]. The processing units are called neurons and they represent the nodes of the network. The interconnections between these nodes determine the information flow in the network and its global behavior. Although a NN is formed from simple units, it can characterize the behavior of complex systems [1]. Moreover, the parallel distributed architecture of NNs makes it possible to implement them on parallel digital processors, which allows for solving problems fast and efficiently [2]. Because of these features, NNs are commonly used in different application areas including complex boundary decision problems, pattern recognition, prediction, industrial process control. In this thesis, we consider two most popular neural networks architectures, namely MLPs and RBFNs.

2.1.1 Multi-Layer Perceptrons

Multi-layer perceptrons are one of the most popular neural networks architectures used in many applications today. MLPs belong to the feed-forward neural networks and are commonly used in function approximation problems. According to the universal approximation theorem [7], there always exists a MLP with one hidden layer that can approximate any nonlinear, continuous, and multi-dimensional function with any accuracy. The architectural graph of a MLP with two hidden layers and one output layer is shown in Fig. 2.1. In this figure, each neuron (except input neurons) receives the stimuli (input) from the neurons in the previous layer. Then, a weighted summation of these inputs is passed through the neuron activation function to produce the output of the neuron [1].

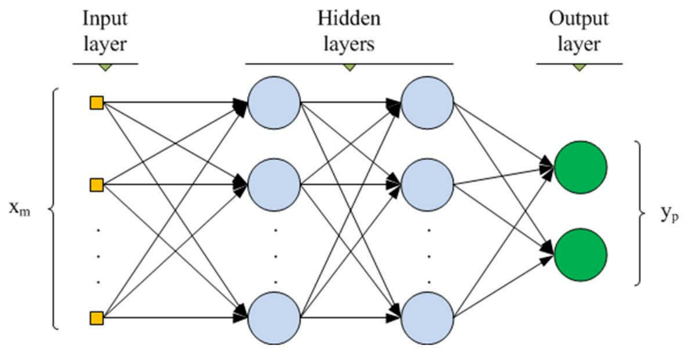


Figure 2.1. The architecture of MLP with two hidden layers [8].

2.1.2 Radial Basis Function Networks

Radial basis function networks are feed-forward neural networks with a single hidden layer and radial basis activation functions such as Gaussian function. RBFNs have been used in different approximation problems and the universal approximation theorem is also proved for these networks [9]. The three layered structure of a RBFN is shown in Fig. 2.2. Each hidden neuron of RBFN has a bell shaped radial-basis activation function centred on a vector in the feature space. Here, first the Euclidean distance between an input vector and the center of each hidden neuron is calculated. Then, the radial basis function is applied to this distance, and its resulting output is a measure of closeness of the input vector to the center of the hidden neuron. The final outputs of an RBFN are weighted sums of the outputs of the hidden layer [1].

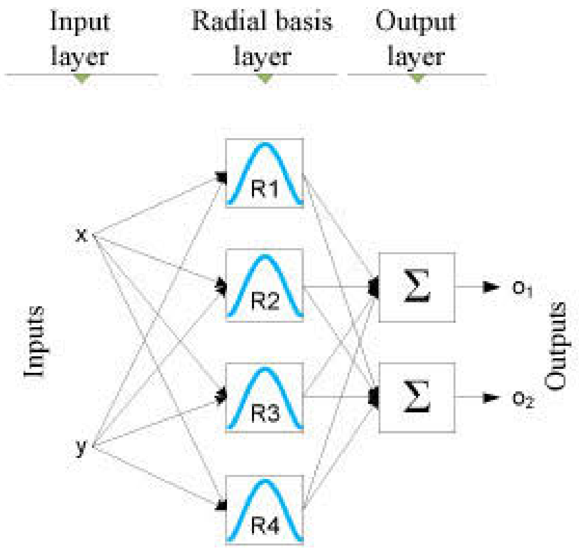


Figure 2.2. The architecture of RBFN [10].

2.1.3 Comparison Between MLPs and RBFNs

Both MLP networks and RBF networks belong to the feed-forward neural networks as the input signals just flow in one direction. However, the activation functions of these two networks work in different ways. In MLPs, the activation function of each neuron is applied to the inner product of the input vector and the weight vector corresponding to that neuron [11]. On the other hand, in RBFNs, the activation function is applied to the distance between the input vector and the center of that hidden neuron [11]. As a result of this difference, RBFNs construct local approximations to the a nonlinear function while the approximations obtained by MLP networks are global [11]. Therefore, MLP networks have good generalization ability even in the areas of the input space where there is not enough training data [1, 11]. As the last point, RBFNs usually need more hidden neurons to achieve the similar accuracy to MLPs that is because of local nature of RBFN activation functions [11].

2.2 Granular Models

As was mentioned before, conventional NNs were numeric models as their inputs, parameters and the resulting outputs were all numeric. Unfortunately, the outputs given by numeric NNs are not completely realistic, because of the modeling errors and the shortage of information about the underlying phenomena [4]. As a result of the need for having more realistic models, the concept of granular modeling of NNs is proposed. In characterization of granular neural networks or GNNs, we give up on numeric models and admit granular parameters to quantify the lack of accuracy in the model [6]. By admitting a specific amount of information granularity to the parameters or the inputs of numeric models, the resulted model can be a better representation of the underlying phenomena. To construct GNNs, the numeric inputs or the numeric parameters of the network must be replaced by information granules through a granularity allocation process. In the following, we will review the concept of information granularity and its application for modeling purposes.

2.2.1 Information Granule and its Quantification

The term information granularity first time was used by Zadeh [12] in the framework of fuzzy sets. Information granule is a collection of objects that are similar or indistinguishable according to some criteria [13].

One can characterize an information granule by its specificity. Specificity shows how detailed an information granules is [5]. As the term implies, a higher specificity corresponds to a case when smaller number of elements exist in the information granule (more specific elements).

Information granules can be represented using several methods namely, intervals [14], fuzzy sets [15], random sets and probability theory. In interval analysis, one replaces the numeric parameter to be granulated with an interval. The broader the interval is, the less specific is the information granule. Similarly, one can use fuzzy sets to form information granules. Fuzzy cardinality can be used as a measure of granularity [4]. When probability theory is used for information granulation, we replace the numeric data by random variables. In this case, the standard deviations (variances) of the random variables are directly connected to the sizes of the corresponding probabilistic information granules. So, a higher standard deviation corresponds to admitting more objects in the information granule and therefore a lower specificity. Note that once a parameter is granulated, the values in the corresponding (interval, fuzzy or probabilistic) set lose their identity and they are assumed to be indistinguishable from each other [4].

2.2.2 Applications of Granular Models

Granular models are useful in various applications and analysis. Two examples of such applications are [4, 6]:

- *Knowledge transfer*: They can arise as a manifestation of knowledge transfer in applications where very limited experimental evidence or data is available. In such a case, the small data available is not sufficient to construct a reliable and realistic model. However, one can rely on an existing model which deals with a close problem and is developed based on a large body of experimental evidences, then, adjust it to the current situation. In such a scenario, we can quantify the effect of the partial relevance between the existing model and the current situation by making the parameters of the model granular to make the model more general.

- *Non-stationary phenomena*: Another application of granular models is in the modeling of non-stationary phenomena where the system exhibits temporal changes. Updating the model continuously to accommodate these variations can lead to considerable development overhead. However, one can alternatively account for these temporal changes by using granular parameters for the system.

2.3 Design of Granular Neural Networks

In general, four different kinds of GNNs can be envisioned depending on the size of data and network information granules [4, 6]:

- *high granularity of both data and the network parameters*

  This is the case for conventional NNs where the input data to the network and the connections of the network are numeric. The training methods for this networks can be easily found in the literature [1].

- *low granularity of data and high granularity of network parameters*

  Here, the input data to the network are nonnumeric while the network itself has numeric parameters. As a result, the output of this network will be granular.

- *high granularity of data and low granularity of the network parameters*

  In this case, the input data are numeric, however, the network itself has granular connections. The network output will be granular as well.

- *low granularity of both data and network parameters*

  Here, we have nonnumeric inputs as well as nonnumeric parameters. The outputs of these networks have also low granularity.

In this thesis, we are concerned with the second type of GNNs which have low granularity of input data and high granularity of network parameters. As discussed in Sec. 1.2, this case can be used when we are interested in the sensitivity analysis of the network output to variations in its inputs.

After the type of the GNN is determined, one can follow two different approaches for its design: design from scratch and design based on a numeric NN. In the design from scratch, the GNN is constructed on the basis of some numeric or granular data [4, 14, 15].

In [4], a design process for GNNs is outlined in two fundamental phases. First, the input data to the network are made condensed in the form of information granules. Second, the network is trained using the resulting granular data instead of the original data. In [14], the architecture of NNs with interval weights is discussed where both the outputs of the network and the targets are in the form of intervals. Here, the cost function is defined based on these interval outputs and interval targets. Then, a new learning algorithm, which is similar to BP (Back-Propagation), is derived to minimize the cost function during learning phase of the NN. Therefore, in this work, the network itself is basically granular by having interval weights. What is done in [15] is similar to the studies presented in [14]. However, in [15], the weights of the network are made granular using fuzzy sets. In addition, the network can also handle the granular inputs in terms of fuzzy sets.

The second approach to design GNNs is on the basis of a numeric NN [3]. In this approach, one first constructs a numeric NN to fit a given numeric input-output relationship. Then, either the network connections or the network inputs are augmented using information granules. In fact, a given level of information granularity is admitted to the parameters and then the optimal allocation of this information granularity is found using a proper optimization method [5].

In this thesis, we follow the second approach for designing GNNs. The work done in [3] is the most closely related work to this thesis. In [3], a GNN is designed on the basis of a numeric NN by replacing the network weights with information granules in the form of intervals. The lengths of these intervals are obtained such that they maximize the coverage of the resulting GNN (coverage criterion). The optimization is performed using single-objective version of particle swarm optimization. Our work in this thesis is different from the work in [3] in four ways. First, we granulate the inputs of the network to construct GNNs and we keep the network parameters numeric, while in [3], the weights of the network are granulated. Second, in this thesis we use probability sets to granulate numeric values while intervals are used in [3] for constructing information granules. Third, our design process is based on the specificity criterion while [3] used coverage criterion for optimizing the granularity allocation. Fourth, the optimization process in [3] is accomplished by evolutionary optimization algorithms, while in this thesis, we derive analytic solutions to the optimization problem.

Chapter 3

An Analytical Approach to Optimal Granularity Allocation in GNNs

In this chapter, we mathematically investigate the design process of GNNs with granular inputs and numeric parameters. Our approach to design the GNN is to first build a numeric NN and then augment its inputs by making them granular using random variables. These random variables can have any symmetric distribution. As an essential design asset, the optimal allocation of a given level of information granularity is found such that the specificity of the network output maximizes.

In the following, first, we explain the way that the inputs of the network are made granular using random variables. Then, we formulate the problem of allocation of information granularity to the inputs as an optimization problem. In particular, we discuss mathematical formulation of both the cost function of the optimization problem and its constraint. Finally, we analytically derive the solution to the optimization problem and we discuss the evaluation of the optimal solution for different NN architectures.

The theoretical results obtained in this chapter are used as a basis for the experimental explorations in Chapter 4.

3.1 Granulating Inputs Using Random Variables

In this section, we discuss the procedure of granulating the inputs of a function through a probabilistic approach by using random variables. Suppose that we have an n-dimensional function

$$y = g(\mathbf{x}) \tag{3.1}$$

where $\mathbf{x} = (x_1, x_2, ..., x_n)$ represents the n-dimensional input vector. In particular, $g(\cdot)$ can be the input-output relationship of a NN. To granulate the numeric vector $\mathbf{x}$, we replace each one of its entries with a random variable and we set the mean of the random variable equal to the numeric value of that entry. In other words, we replace x_i with a random variable X_i such that $E(X_i) = x_i$, for $i = 1, ..., n$, where $E(\cdot)$ is the expectation operator. We choose the n random variables to be independent, so the resulting granular input vector $\mathbf{X} = (X_1, X_2, ..., X_n)$ is formed by n independent random variables with the mean vector $\mathbf{x} = (x_1, x_2, ..., x_n)$ and the variance vector $\boldsymbol{\sigma}^{\mathbf{2}} = (\sigma_1^2, \sigma_2^2, ..., \sigma_n^2)$. Note that $\sigma_i = 0$ corresponds to the case that the i-th input is a numeric value. Increasing σ_i decreases the specificity or the granularity level of the i-th input.

When random variables are used to granulate data, our approach in choosing the distributions of the random variables and their corresponding parameters determines the allocation protocol. Generally, one can follow different protocols in granularity allocation process, depending on the diversity considered in setting the allocation parameters. The protocols discussed here are generalizations of the granulation protocols discussed in [6] when interval analysis is used to make information granules.

- *Random Allocation*: A random allocation protocol randomly assigns a given total granularity level to different inputs. This allocation protocol is obviously sub-optimal but it can be used as a reference to comment on how superior other allocation protocols perform compared to a purely random approach.

- *Uniform vs. Non-uniform Allocation*: In uniform allocation, we select the random variables to have identical distributions around their means. This means that except for their mean values (which are set equal to the corresponding numeric data), all other parameters of the distributions used are similar. For example, we only allow for equal variances $\sigma_1^2 = \sigma_2^2 = ... = \sigma_n^2$ in the allocation process. It is needless to say that no parameter optimization is involved in this protocol and one simply gives equal variances to the distributions such that the total granularity level becomes equal to the predetermined given value. In a non-uniform allocation protocol, however, we allow the distributions to have different parameters. Clearly, parameter optimization is required when a non-uniform protocol is used, in order to maximize an objective function (like output coverage or specificity level). One also notes that the term "uniform" here does not refer to uniform distribution for the random variables, but refers to uniform allocation of granularity to different random variables.

- *Symmetric vs. Asymmetric Allocation*: In a symmetric allocation protocol, the distribution of each random variable is chosen to be symmetric around its expected value (mean). Examples of symmetric distributions are normal, uniform or Laplace distributions. On the other hand, a non-symmetric allocation protocol allows for using distributions that are not necessarily symmetric around their mean values. Gamma random variables are examples of random variables with asymmetric distributions. An asymmetric protocol provides higher degrees of freedom in the allocation process, but it requires to enter measures like *skewness* to quantify the asymmetry level of distributions around their means values [16] in the allocation problem.

In this thesis, we follow a non-uniform but symmetric granularity allocation protocol. In other words, we choose the random variables from a symmetric family of distributions, but we allow them to have different distribution parameters.

As the last remark, one notices that by making the input vector **x** granular, the corresponding output also becomes granular. In other words, the output will also be a random variable Y= g(**X**) instead of a deterministic numeric value $y = g(\mathbf{x})$. We denote the standard deviation of the output distribution by σ_y and it reflects the specificity level of the output information granule, the smaller σ_y is, the more specific is the output information granule.

3.2 Formulation of Granularity Allocation as An Optimization Problem

Assume that $\mathbf{X}^{(1)}, ..., \mathbf{X}^{(M)}$ are M different n-dimensional granular input vectors with corresponding n-dimensional mean vectors $\mathbf{x}^{(1)}, ..., \mathbf{x}^{(M)}$ and variance vectors $\boldsymbol{\sigma^2}^{(1)}, ..., \boldsymbol{\sigma^2}^{(M)}$. Since we allocate granularity along input dimensions, different input vectors have the same variance along a specific dimension, therefore, $\boldsymbol{\sigma^2} = \boldsymbol{\sigma^2}^{(1)} = ... = \boldsymbol{\sigma^2}^{(M)}$. This means that the i-th entries of all the input vectors have the same variance. One notes that this does not mean a uniform granularity allocation protocol, because here different entries of a fixed variance vector are different. In fact, the variances are different along input dimensions but not along time. For $m = 1, ..., M$, $Y^{(m)}$ denotes the output random variable corresponding to the m-th granular input vector $X^{(m)}$, and we use $\sigma^2_{y^{(m)}}$ to represent its variance.

Now, given the M input vectors and a *given* level of total information granularity ε, we aim to find the optimal allocation of ε among the n input dimensions such that the output becomes as specific as possible (specificity criterion). Formally speaking, our goal is to find the optimal variance vector $\boldsymbol{\sigma^2} = (\sigma_1^2, \sigma_2^2, ..., \sigma_n^2)$ that minimizes the average of the M output variances, given the constraint that $\boldsymbol{\sigma^2}$ introduces a certain amount of information granularity to the inputs. To formulate this problem as an optimization problem with parameters $\sigma_1, ..., \sigma_n$, we determine the mathematical forms of the objective function and the constraint in Secs. 3.2.1 and 3.2.2, respectively.

3.2.1 The Objective Function of the Optimization Problem

As mentioned before, we use the specificity criterion to design the GNN with granular inputs, therefore, our goal is to choose $\sigma_1, ..., \sigma_n$ to allocate a given level of information granularity to the input dimensions such that the average output variance minimizes. The average output variance can be written as

$$I = \frac{1}{M} \sum_{m=1}^{M} \sigma^2_{y^{(m)}} \tag{3.2}$$

where M is the number of available input-output pairs. Note that derivation of the relationship between the objective function I and the optimization parameters $\sigma_1^2, ..., \sigma_n^2$ is not always straightforward. This is particularly the case when we are dealing with NNs where the output of the network is a complicated non-linear mathematical function of the inputs. As a result, it is very useful to approximate the objective function in (3.2) with a more mathematically tractable function. To do this, we approximate the variance of a function of random variables in terms of the variances of its inputs using [17, eq. 6.115], which leads to

$$\sigma^2_{y^{(m)}} \approx \sum_{i=1}^{n} \left(\frac{\partial g}{\partial x_i} \, |_{\mathbf{x}=\mathbf{x}^{(m)}} \right)^2 \sigma_i^2, \quad m = 1, ..., M \tag{3.3}$$

where $g(\cdot)$ is the NN output function in terms of the inputs, and $\frac{\partial g}{\partial x_i} \, |_{\mathbf{x}=\mathbf{x}^{(m)}}$ is $\frac{\partial g}{\partial x_i}$ calculated at the m-th input vector. Here, we have used the fact that the input random variables are independent. This approximation becomes more precise as values of σ_i^2 get smaller. Using (3.3) in the objective function (3.2), we approximate the objective function by

$$I(\boldsymbol{\sigma^2}) \approx \frac{1}{M} \sum_{m=1}^{M} \sigma^2_{y^{(m)}} = \frac{1}{M} \sum_{m=1}^{M} \sum_{i=1}^{n} \left(\frac{\partial g}{\partial x_i} \, |_{\mathbf{x}=\mathbf{x}^{(m)}} \right)^2 \sigma_i^2. \tag{3.4}$$

By changing the order of summations in (3.4), we have

$$I(\boldsymbol{\sigma^2}) \approx \frac{1}{M} \sum_{i=1}^{n} \left(\sum_{m=1}^{M} \left(\frac{\partial g}{\partial x_i} \, |_{\mathbf{x}=\mathbf{x}^{(m)}} \right)^2 \right) \sigma_i^2. \tag{3.5}$$

In (3.5), the expression inside the outer parenthesis is a constant value (independent of variances) for each i and we denote it by

$$a_i = \frac{1}{M} \sum_{m=1}^{M} \left(\frac{\partial f}{\partial x_i} \, |_{\mathbf{x}=\mathbf{x}^{(m)}} \right)^2. \tag{3.6}$$

Therefore, the objective function can be approximated by

$$I(\boldsymbol{\sigma^2}) \approx \sum_{i=1}^{n} a_i \sigma_i^2. \tag{3.7}$$

So, we use (3.7) as the objective function of our optimization problem.

3.2.2 The Constraint of The Optimization Problem

Another significant step towards the formulation of our optimization problem is to quantify the total granularity level allocated to the input dimensions and set it equal to a given value ε. So, by changing ε one controls the total granularity level allocated to the input information granules.

Using Variance as Granularity Measure

The first candidate to quantify the granularity level associated with a random variable is its variance. In this case, we can write the total granularity level allocated to the inputs as the sum of the variances and therefore the constraint becomes

$$\sum_{i=1}^{n} \sigma_i^2 = \varepsilon. \tag{3.8}$$

This constraint together with the objective function (3.7) leads to the optimization problem

$$\begin{aligned} \underset{\boldsymbol{\sigma^2}=(\sigma_1^2,\ldots,\sigma_n^2)}{\text{minimize}} \quad & I(\boldsymbol{\sigma^2}) = \sum_{i=1}^{n} a_i \sigma_i^2 \\ \text{subject to} \quad & \sum_{i=1}^{n} \sigma_i^2 = \varepsilon,\ \sigma_i^2 \geq 0\ \forall i. \end{aligned} \tag{3.9}$$

(3.9) is a linear programming optimization problem with one linear equality constraint (see [18, Ch. 4]). One notes that the optimization problem (3.9) can be interpreted as a investment problem, where $\boldsymbol{\sigma^2}$ represents the allocation of the total budget ε to different assets i with the corresponding return of $-a_i$. We are interested in maximizing the total return which is equivalent to minimizing $\sum_{i=1}^{n} a_i \sigma_i^2$. Since the return values are known, it is obvious that the optimal solution to this problem is to invest all the budget ε on the asset with the highest return (i.e., smallest a_i). In other words, the solution to (3.9) is to allocate all ε to the input which has the smallest corresponding value of a_i. A formal proof of this result is given in Appendix A.

The use of the optimization problem (3.9) is very restricted in practical applications. In particular, one important application of granular models was in sensitivity analysis. In sensitivity analysis using GNNs, we are interested to comment on the precision required in estimating each input value based on the optimal granularity level allocated to it (see Sec. 1.2). In this case, a higher granularity level allocated to an input dimension implies that higher precision is required in determination or estimation of its value to keep the outputs precise. However, the solutions to (3.9) do not give us much insight for sensitivity analysis, because (3.9) allocates all the granularity budget to one input dimension that has the smallest coefficient a_i and zero granularity to the other dimensions. For example, an optimal variance vector looks like $\boldsymbol{\sigma^2} = (0, \varepsilon, 0, 0, ..., 0)$ which does not provide us with any information about the relative sensitivity of the output to the inputs that are given granularity level 0.

Using Entropy as Granularity Measure

We observed that the use of the sum of variances as the total granularity level of the inputs led to a solution to granularity allocation problem whose application was very limited in practice. Therefore, in this part, we introduce another measure for the granularity level of the inputs.

In probability and information theory, the uncertainty associated with a discrete random variable is characterized by its entropy. For *discrete* random variables entropy is defined as

$$H(X) = -\sum_i P_X(x_i) \ln P_X(x_i) \tag{3.10}$$

where $P_X(\cdot)$ represents the probability mass function of the discrete random variable X. Note that for discrete random variables, $H(X) \geq 0$ where $H(X) = 0$ corresponds to the case that X takes just one value and therefore we have zero uncertainty about the value of the random variable X. In an analogous way, the entropy of a *continuous* random variable is defined as

$$h(X) = -\int_{-\infty}^{\infty} p_X(x) \ln p_X(x)\, dx \tag{3.11}$$

where $p_X(\cdot)$ is the probability density function (PDF) of the continuous random variable X. For a discrete random variable X, the entropy $H(X)$ is positive and it is used as the measure of uncertainty about X. For continuous random variables, however, entropy can take any value from $-\infty$ to $+\infty$ and it can be used to measure the changes in uncertainty. So, when dealing with a continuous random variable, we measure its uncertainty level with respect to a reference random variable [16, Ch. 14, Note 1]. For instance, consider two zero-mean normal random variables with variances σ_1^2 and $\sigma_2^2 > \sigma_1^2$. The entropy of a normal random variable with mean μ and variance σ^2 is [16, eq. 14.84]

$$h(X) = \frac{1}{2} \ln \left(2\pi e \sigma^2\right). \tag{3.12}$$

So, our uncertainty about X_2 compared to X_1 is $h(X_2) - h(X_1) = \ln \frac{\sigma_2}{\sigma_1} > 0$. This means that more uncertainty is associated with the normal random variable with the higher variance.

The entropy of a random variable is invariant to translation, i.e., [16, 19]

$$h(X + \mu) = h(X). \tag{3.13}$$

So, changing the mean of a random variable does not change its entropy. However, entropy varies by changing the variance of a random variable. In general, a higher spread of a random variable around its mean associates with a higher entropy (higher uncertainty level). This fact relates the entropy of a random variable with its variance. Mathematically speaking, assume that X_{ref} is a random variable with variance 1. Then, $X = \sigma X_{\text{ref}}$ has the same distribution form but its variance is σ^2. Following the scaling property of entropy [16, eq. 14.115], the entropies of X and X_{ref} are related as

$$h(X) = h(X_{\text{ref}}) + \ln \sigma. \tag{3.14}$$

In other words, the uncertainty about X compared to the reference X_{ref} with variance 1 is

$$h(X) - h(X_{\text{ref}}) = \ln \sigma. \tag{3.15}$$

Finally, the total uncertainty of a number of independent random variables is equal to the sum of the entropies of individual random variables [16, Sec. 14.3]. This property is very useful for formulation of our problem, as we have used independent random variables to granulate the inputs and we are interested in the total uncertainty level associated with them.

Now, we return to the original optimization problem and we use the entropy to measure the granularity level allocated to each input random variable. Since the input random variables are independent, the total uncertainty level given to input random variables is the sum of their individual uncertainties. Using the reference distribution X_{ref} with variance 1 from the same distribution family as X_is, we formulate the constraint of the optimization problem as the total uncertainty given to the inputs

$$\sum_{i=1}^{n} \left[h(X_i) - h(X_{\text{ref}})\right] = \sum_{i=1}^{n} \ln \sigma_i = \varepsilon. \tag{3.16}$$

where we used (3.15) to write the last equality.

Finally, using (3.16) and (3.7) we write the resulting optimization problem as

$$\begin{aligned} &\underset{\boldsymbol{\sigma^2}=(\sigma_1^2,\ldots,\sigma_n^2)}{\text{minimize}} \quad I(\boldsymbol{\sigma^2}) = \sum_{i=1}^{n} a_i \sigma_i^2 \\ &\text{subject to} \quad \sum_{i=1}^{n} \ln \sigma_i = \varepsilon \end{aligned} \tag{3.17}$$

where a_is are calculated using (3.6).

In the next section, we derive the optimal solution of the optimization problem (3.17).

3.3 Derivation of The Optimal Standard Deviations For The Inputs

To solve the optimization problem in (3.17), we first use $\ln a + \ln b = \ln(ab)$ to rewrite the constraint (3.16) as

$$\prod_{i=1}^{n} \sigma_i = e^{\varepsilon} \triangleq \alpha. \tag{3.18}$$

By using (3.18) to write σ_n in terms of other values of σ_i, the n-dimensional constrained optimization problem (3.17) can be converted to a $(n-1)$-dimensional unconstrained optimization problem

$$\underset{\sigma_1^2,\ldots,\sigma_{n-1}^2}{\text{minimize}} \quad I'(\sigma_1^2, \ldots \sigma_{n-1}^2) = a_1\sigma_1^2 + a_2\sigma_2^2 + \ldots + a_{n-1}\sigma_{n-1}^2 + a_n \frac{\alpha^2}{\sigma_1^2\sigma_2^2\ldots\sigma_{n-1}^2}. \tag{3.19}$$

To solve (3.19), our first step is to find the critical points of I', i.e., the points that make all the function's partial derivatives zero. The second step is to use the second derivative test to determine whether each critical point of the function is a maximum, a minimum or a saddle point. For a function of more than one variable, this test is based on evaluation of the Hessian matrix at the critical points. In particular, assuming that all the second order partial derivatives of the function are continuous on a neighbourhood of a critical point, if the Hessian matrix at the critical point is positive definite, the point is a local minimum [20]. In linear algebra, a symmetric $n \times n$ real matrix A is said to be positive definite if $\mathbf{x}^T A\mathbf{x}$ is positive for every non-zero column vector x of n real numbers [21]. All the eigenvalues of a positive definite matrix are positive [21]. In the following two subsections, we accomplish these two steps to solve the optimization problem (3.19).

1. Finding the Critical Points of the Objective Function

To find the critical points of I', we find $\sigma_1, \sigma_2, \ldots, \sigma_{n-1}$ that simultaneously make all the partial derivatives zero. The derivatives of I' with respect to its variables are

$$\begin{aligned}
\frac{\partial I'}{\partial \sigma_1} &= 2a_1\sigma_1 - \frac{2a_n\alpha^2}{\sigma_1^3 \prod_{\substack{k=1\\k\neq 1}}^{n-1} \sigma_k^2} \\
\frac{\partial I'}{\partial \sigma_2} &= 2a_2\sigma_2 - \frac{2a_n\alpha^2}{\sigma_2^3 \prod_{\substack{k=1\\k\neq 2}}^{n-1} \sigma_k^2} \\
\vdots \quad & \quad \vdots \\
\frac{\partial I'}{\partial \sigma_{n-1}} &= 2a_{n-1}\sigma_{n-1} - \frac{2a_n\alpha^2}{\sigma_n^3 \prod_{\substack{k=1\\k\neq n-1}}^{n-1} \sigma_k^2}.
\end{aligned}$$

Setting all the derivatives equal to zero, we have

$$\sigma_i^4 \prod_{\substack{k=1\\k\neq i}}^{n-1} \sigma_k^2 = \left(\frac{a_n}{a_i}\right)\alpha^2, \quad i = 1, ..., n-1. \tag{3.20}$$

By solving the system of $(n-1)$ equations and by considering the fact that standard deviation is non-negative, we calculate the unknowns as

$$\sigma_i = \sqrt[2n]{\alpha^2 \prod_{k=1}^{n} \frac{a_k}{a_i}}, \quad i = 1, ..., n-1. \tag{3.21}$$

(3.21) gives the critical points of function I' in its domain $\mathbb{R}_+^{n-1}$. Now that we have the critical points of I', we can proceed to the second step, which is the second derivative test.

2. Second Derivative Test

To do the second derivative test, one first needs to calculate the Hessian matrix of the function I' at its critical points to determine if they are maximums, minimums or saddle points [20]. Hessian matrix is a square matrix and its computation requires the evaluation of the second-order partial derivatives of the function. Given the real-valued function I', if all second-order partial derivatives of I' exist and are continuous over the function domain, then the Hessian matrix of I' is

$$\mathcal{H}(I') = \begin{bmatrix} \frac{\partial^2 I'}{\partial \sigma_1^2} & \frac{\partial^2 I'}{\partial \sigma_1 \partial \sigma_2} & \cdots & \frac{\partial^2 I'}{\partial \sigma_1 \partial \sigma_{n-1}} \\ \frac{\partial^2 I'}{\partial \sigma_2 \partial \sigma_1} & \frac{\partial^2 I'}{\partial \sigma_2^2} & \cdots & \frac{\partial^2 I'}{\partial \sigma_2 \partial \sigma_{n-1}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 I'}{\partial \sigma_{n-1} \partial \sigma_1} & \frac{\partial^2 I'}{\partial \sigma_{n-1} \partial \sigma_2} & \cdots & \frac{\partial^2 I'}{\partial \sigma_{n-1}^2} \end{bmatrix}. \tag{3.22}$$

Using the expression for I' given in (3.19), we evaluate the partial derivatives of I' as

$$\frac{\partial I'}{\partial \sigma_i} = 2a_i\sigma_i - \frac{2a_n\alpha^2}{\sigma_i^3 \prod_{\substack{k=1\\k\neq i}}^{n-1} \sigma_k^2} \tag{3.23}$$

and therefore

$$\frac{\partial^2 I'}{\partial \sigma_i^2} = 2a_i + \frac{6a_n\alpha^2}{\sigma_i^4 \prod_{\substack{k=1\\k\neq i}}^{n-1} \sigma_k^2}. \tag{3.24}$$

Evaluation of (3.24) in the critical points yields

$$\frac{\partial^2 I'}{\partial \sigma_i^2} = 8a_i \tag{3.25}$$

where we have used (3.20) to write (3.25).

In a similar way, we use (3.23) to evaluate the rest of the partial derivatives as

$$\frac{\partial^2 I'}{\partial \sigma_i \partial \sigma_j} = \frac{\partial^2 I'}{\partial \sigma_j \partial \sigma_i} = \frac{4a_n\alpha^2}{\sigma_i^3\sigma_j^3 \prod_{\substack{k=1\\k\neq i,k\neq j}}^{n-1} \sigma_k^2}. \tag{3.26}$$

Again, we evaluate (3.26) at the critical points which gives

$$\frac{\partial^2 I'}{\partial \sigma_i \partial \sigma_j} = \frac{\partial^2 I'}{\partial \sigma_j \partial \sigma_i} = 4a_i\frac{\sigma_i}{\sigma_j} = 4\sqrt{a_i a_j} \tag{3.27}$$

where we have used (3.20) and (3.21), respectively. Substitution of (3.25) and (3.27) in (3.22) gives the Hessian matrix of I' at the critical point as

$$\mathcal{H}(I') = 4\begin{bmatrix} 2a_1 & \sqrt{a_1a_2} & \dots & \sqrt{a_1a_{n-1}} \\ \sqrt{a_2a_1} & 2a_2 & \dots & \sqrt{a_2a_{n-1}} \\ \vdots & \vdots & \ddots & \vdots \\ \sqrt{a_{n-1}a_1} & \sqrt{a_{n-1}a_2} & \dots & 2a_{n-1} \end{bmatrix} \tag{3.28}$$

Now we complete the second derivative test by determining if the Hessian matrix in (3.28) is positive definite. To do this, we need to show that $\mathbf{x}^T\mathcal{H}(I')\mathbf{x}$ is positive for every non-zero column vector $\mathbf{x}$ of $n-1$ real numbers. We prove $\mathcal{H}(I')$ is positive definite as

$$\begin{aligned}\mathbf{x}^T\mathcal{H}(I')\mathbf{x} &= \begin{bmatrix} x_1 & x_2 & \dots & x_{n-1}\end{bmatrix} \times 4\begin{bmatrix} 2a_1 & \sqrt{a_1a_2} & \dots & \sqrt{a_1a_{n-1}} \\ \sqrt{a_2a_1} & 2a_2 & \dots & \sqrt{a_2a_{n-1}} \\ \vdots & \vdots & \ddots & \vdots \\ \sqrt{a_{n-1}a_1} & \sqrt{a_{n-1}a_2} & \dots & 2a_{n-1} \end{bmatrix} \times \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{n-1}\end{bmatrix} \\ &= 4\left(\left(\sum_{i=1}^{n-1} 2a_ix_i^2\right) + \left(\sum_{i=1}^{n-2}\sum_{j=i+1}^{n-1} 2\sqrt{a_ia_j}x_ix_j\right)\right) \\ &= 4\left(\left(\sum_{i=1}^{n-1} a_ix_i^2\right) + \left(\sum_{i=1}^{n-1}\sqrt{a_i}x_i\right)^2\right) > 0 \end{aligned} \tag{3.29}$$

Therefore, we have shown that the Hessian matrix at the critical point is positive definite. In other words, the critical point of the objective function I' is a minima of the function.

In summary, we derived the optimal amounts of standard deviations that minimize the objective functions I' as

$$\sigma_i = \sqrt[2n]{\alpha^2 \prod_{k=1}^{n} \frac{a_k}{a_i}}, \quad i = 1, ..., n-1. \tag{3.30a}$$

The same values of σ_i, $i = 1, ..., n-1$, minimize I in the original optimization problem (3.17). The optimal value of σ_n can be evaluated by substitution of (3.30a) in (3.18) as

$$\sigma_n = \frac{\alpha}{\sigma_1 \sigma_2 ... \sigma_{n-1}} = \sqrt[2n]{\alpha^2 \prod_{k=1}^{n} \frac{a_k}{a_n}}. \tag{3.30b}$$

The solution for σ_n has the same form as the solutions in (3.30a) which is expected because of the symmetry of the original optimization problem to the input variances. Using (3.30a) and (3.30b), we summarize the optimal solution of (3.17) as

$$\sigma_i^* = \sqrt[2n]{\alpha^2 \prod_{k=1}^{n} \frac{a_k}{a_i}}, \quad i = 1, ..., n \tag{3.31}$$

Eq. (3.31) gives the optimal standard deviations and it is the solution to the optimization problem (3.17).

***Discussion*:**

Eq. (3.31) gives us a good insight about the process of allocation of information granularity to different inputs. According to (3.6), the coefficient a_i characterizes the average variability of the network output to the changes in the input x_i. Therefore, the optimal value σ_i, which is inversely proportional to a_i, increases as the average variability of output in the direction of x_i decreases. In other words, the more the function changes along one input, the less standard deviation is allocated to it.

The following algorithm shows the way for calculating the optimal standard deviations.

Given the mean values of the input random variables which are equal to the corresponding numeric inputs and the standard deviations (3.31), we can derive the parameters of the input random variables based on the distribution family they belong to. For example, let's consider three distributions for the input random variables:

```
Require: $\varepsilon$ , $M$, $(\mathbf{x}^{(m)}, y^{(m)})$   $m = 1, ..., M$,
1: $\alpha \leftarrow e^{\varepsilon}$
2: $g \leftarrow$ train a numeric NN to fit the given data set
3: for $i = 1$ to $n$ do
4:     $a_i \leftarrow \frac{1}{M} \sum_{m=1}^{M} \left( \frac{\partial g}{\partial x_i} |_{\mathbf{x}=\mathbf{x}^{(m)}} \right)^2$
5: for $i = 1$ to $n$ do
6:     $\sigma_i \leftarrow \sqrt[2n]{\alpha^2 \prod_{k=1}^{n} \frac{a_k}{a_i}}$,
   return $\sigma_i$s
```

- *Normal Distributions:* Assume that the input random variables have normal distributions denoted by $\mathcal{N}(\mu_i, \sigma_i^2)$ with the PDF

$$p_{X_i}(x) = \frac{1}{\sqrt{2\pi\sigma_i^2}} \exp\left(-\frac{(x-\mu_i)^2}{2\sigma_i^2} \right) \tag{3.32}$$

  So, we have $\mu_i^* = x_i$, and the standard deviations σ_i^* are readily given by (3.31).

- *Uniform Distributions:* As the second example, we consider uniform distributions denoted by $\mathcal{U}(c_i, d_i)$ where

$$p_{X_i}(x) = \begin{cases} \frac{1}{d-c} & c_i \leqslant x \leqslant d_i \\ 0 & otherwise \end{cases} \tag{3.33}$$

  and its mean and standard deviation are given by

$$E(X_i) = \frac{c_i + d_i}{2} \tag{3.34a}$$

$$Var(X_i) = \frac{(d_i - c_i)^2}{12} \tag{3.34b}$$

  Setting $E(X_i) = x_i$ and substituting (3.31) in (3.34b), we obtain the optimal parameters of the uniform distributions as

$$c_i^* = x_i - \sqrt{3}\sigma_i^* \tag{3.35a}$$

$$d_i^* = x_i + \sqrt{3}\sigma_i^*. \tag{3.35b}$$

- *Laplace Distributions:* A laplace distribution $\mathcal{L}(\alpha, \beta)$ is characterized by two parameters, location parameter γ and scale parameter β. The Laplace PDF is

$$p_{X_i}(x) = \frac{1}{2\beta_i} \exp\left(-\frac{x-u_i}{\beta_i} \right) \tag{3.36}$$

  and its mean and variance are given as

$$E(X_i) = \gamma_i \tag{3.37a}$$

$$Var(X_i) = 2\beta_i^2. \tag{3.37b}$$

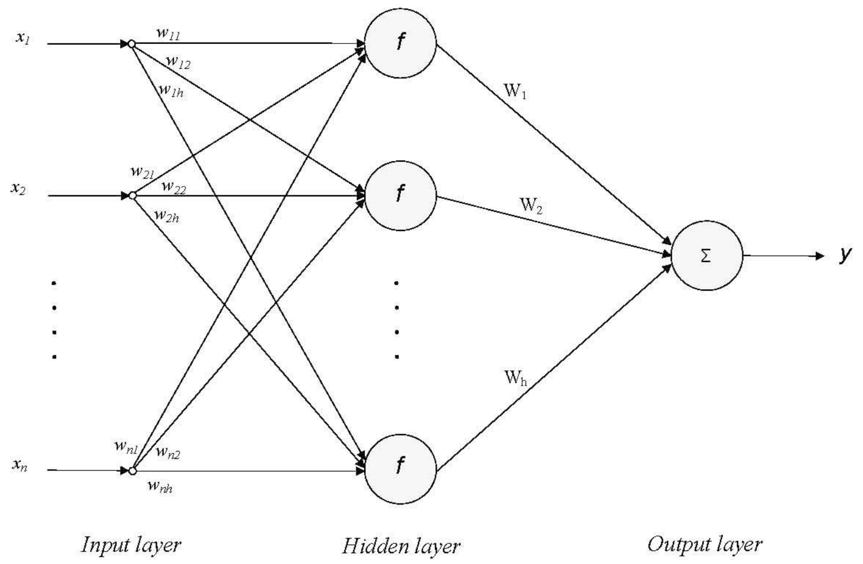


Figure 3.1. The structure of a MLP network with one hidden layer.

So, the optimal parameters of a Laplace distribution are

$$\gamma_i^* = x_i \tag{3.38a}$$

$$\beta_i^* = \frac{\sigma_i *}{\sqrt{2}}. \tag{3.38b}$$

3.4 Evaluation of coefficients a_i for Different Neural Network Architectures

In Sec. 3.2, we formulated an optimization problem to distribute a given amount of information granularity among the inputs of a function $g(\cdot)$, so that the output becomes as specific as possible. In the optimization problem (3.17), the coefficients $\{a_i\}_{i=1}^{n}$, which are defined in (3.6), are determined by function $g(\cdot)$. When dealing with neural networks, the input-output function depends on the architecture of the neural network chosen. In this section, we derive the coefficients a_i for three different neural network architectures, namely MLP networks with one and two hidden layers and RBFN.

3.4.1 MLP Networks with One Hidden Layer

The structure of a MLP network with one hidden layer is shown in Fig. 3.1. Suppose that all the hidden neurons have hyperbolic tangent sigmoid transfer function

$$f(t) = \frac{2}{1+e^{-2t}} - 1 \tag{3.39}$$

and also assume that the single output neuron has a linear transfer function. In this case, the relationship between the output and the inputs of the neural network is

$$y = g(\mathbf{x}) = \sum_{j=1}^{h+1} W_j \left(\frac{2}{1+\exp(-2(\sum_{i=1}^{n+1} x_i w_{ij}))} - 1 \right). \tag{3.40}$$

In equation (3.40), $\{W_j\}_{j=1}^{h}$ are the weights between the hidden layer and the output layer, and w_{ij} represents the weight between the i-th input and the j-th neuron of the hidden layer. The bias of output neuron and the biases of hidden neurons are considered in the vector W and matrix w, respectively.

According to equation (3.21), for finding the optimal standard deviations, we need to calculate the derivative of g with respect to each one of its inputs. This yields

$$\frac{\partial g}{\partial x_i} = 4 \sum_{j=1}^{h} W_j w_{ij} \frac{A_j}{(1+A_j)^2} \tag{3.41}$$

where $A_j \triangleq \exp\left(-2\sum_{i=1}^{n+1} x_i w_{ij}\right)$. Substitution of (3.41) in (3.6) gives the coefficients a_i which are required to compute the optimal solution in (3.21) for a MLP network with one hidden layer.

3.4.2 MLP Networks with Two Hidden Layers

In this section, we consider a MLP network with two hidden layers. Similar to the previous section, we assume the activation functions of all the hidden neurons are hyperbolic tangent sigmoid and the activation function of the single output neuron is a linear function. The relationship between the output and the inputs of the considered neural network is

$$g(\mathbf{x}) = \sum_{j=1}^{h_2+1} W_j \left(\frac{2}{1+\exp\left(-2\sum_{k=1}^{h_1+1} v_{kj}\left(\frac{2}{1+\exp(-2\sum_{i=1}^{n+1} x_i w_{ik})} - 1\right)\right)} - 1 \right) \tag{3.42}$$

where h_1 and h_2 are the number of hidden neurons in the first and the second hidden layers, respectively. w_{ik} represents the weight between the i-th input node and the k-th neuron in the first hidden layer, and v_{kj} shows is the weight between the k-th neuron in the first hidden layer and the j-th neuron in the second hidden layer. Finally, vector W denotes the weights between the second hidden layer and the output layer. The derivative of g with respect to x_i is

$$\frac{\partial g}{\partial x_i} = 16 \sum_{j=1}^{h_2} \left(W_j \frac{B_j}{(1+B_j)^2} \sum_{k=1}^{h_1} \left(w_{ik} v_{kj} \frac{A_k}{(1+A_k)^2} \right) \right) \tag{3.43a}$$

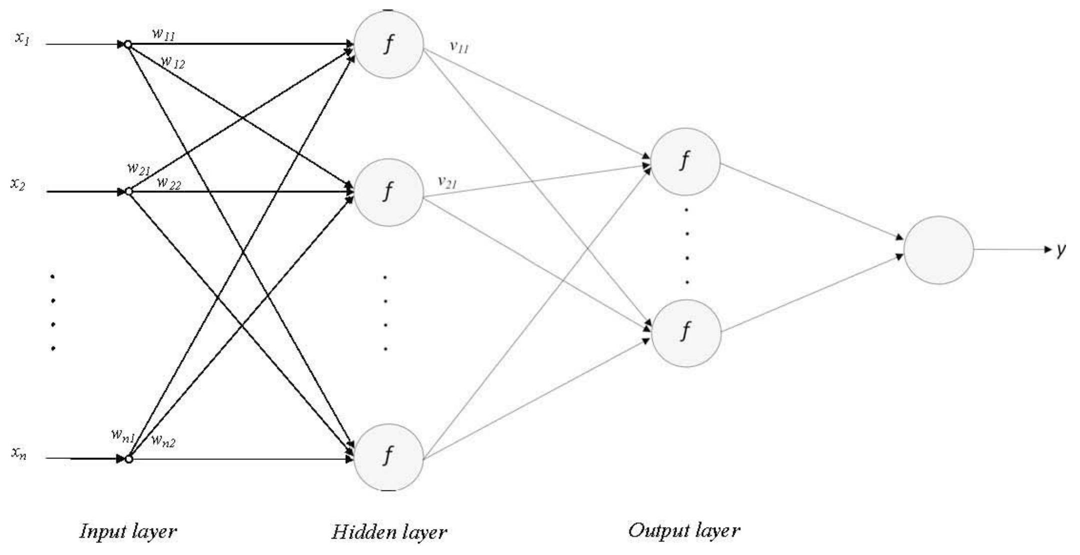


Figure 3.2. The structure of a MLP network with two hidden layer.

where

$$A_k = \exp\left(-2\sum_{i=1}^{n+1} x_i w_{ik}\right) \tag{3.43b}$$

$$B_j = \exp\left(-2\sum_{k=1}^{h_1+1} v_{kj}\left(\frac{2}{1+A_k}-1\right)\right). \tag{3.43c}$$

Eq. (3.43) can be used in (3.6) to compute the coefficients a_i for a MLP network with two hidden layers.

3.4.3 RBFNs

The structure of a RBFN is shown in Fig. 3.3. Each neuron in the hidden layer of the RBFN has a Gaussian activating function whose output is inversely proportional to the distance between an input and the center of the neuron. So, for the j-th hidden neuron we have

$$f_j(\mathbf{x}) = \exp(-b_j^2(\|\mathbf{x}-\mathbf{c}_j\|)^2) \tag{3.44}$$

where **x** is the input vector, $\mathbf{c}_j$ is the n-dimensional center vector of the j-th hidden neuron and b_j characterizes the spread of the j-th Gaussian function ($j = 1, ..., h$). In (3.44), $\|\mathbf{x}-\mathbf{c}_j\| = \sqrt{\sum_{i=1}^{n}(x_i - w_{ij})^2}$ is the Euclidean distance between the n-dimensional input vector and the center of the j-th hidden neuron. Note that the values of parameters $\mathbf{c}_j$ and b_j are determined in the training phase of the network.

The relationship between the output and the inputs of the RBFN can be written as

$$g(\mathbf{x}) = \sum_{j=1}^{h+1} W_j \exp\left(-b_j^2\|\mathbf{x}-\mathbf{c}_j\|^2\right) \tag{3.45}$$

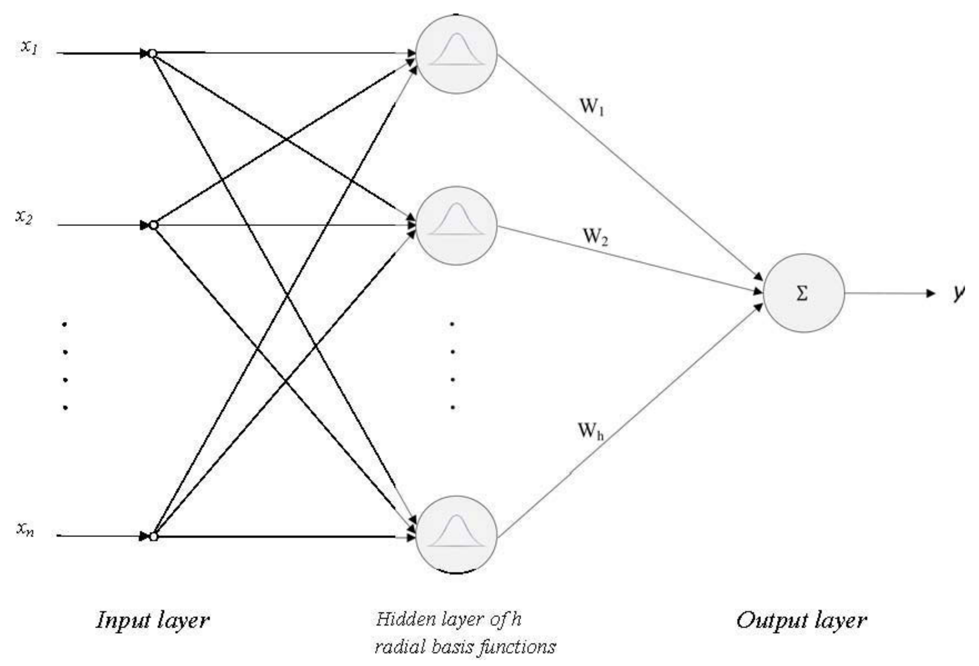


Figure 3.3. The structure of a RBFN.

In equation (3.45), W is the vector of weights between hidden layer and output layer. The bias of the single output neuron is also included in W .

The derivative of $g(\mathbf{x})$ along each of its input dimensions x_i is

$$\frac{\partial g}{\partial x_i} = -2 \sum_{j=1}^{h} W_j (x_i - c_{ij}) b_j^2 \exp\left(-b_j^2 \|\mathbf{x} - \mathbf{c}_j\|^2\right) \tag{3.46}$$

and c_{ij} is the i-th entry of vector $\mathbf{c}_j$. Again, one can use (3.46) in (3.6) to compute the coefficients a_i when RBFN architecture is used to construct the GNN.

3.5 Concluding Remarks

In this chapter, we formulated the problem of finding optimal allocation of information granularity to the inputs of a GNN as an optimization problem. We summarized the optimization problem in (3.17), where the objective function was expressed as a linear combination of the variances of the input distributions, and the constraint of the optimization problem was derived by using the entropy of input random variables as a measure of the granularity level associated with them. The optimal values of the standard deviations of the input random variables were derived analytically in (3.31) as the solution to the optimization problem. Finally, we discussed evaluation of the optimal standard deviations when RBFN or MLP with one or two hidden layers were used to construct the GNN. In the next chapter, we use the theoretical results of this chapter for the design of GNNs on the basis of NNs constructed to fit several synthetic and real data sets.

Chapter 4

Experimental Results

In Chapter 3, we investigated the problem of optimal allocation of information granularity to the inputs of GNNs. In this chapter, we apply the theoretical results derived in Chapter 3 to design GNNs based on NNs constructed to fit several synthetic and real data sets. In particular, we use a total of five synthetic and six real data sets in our experiments. For each data set, three NN architectures are used to fit the data, namely MLP networks with one and two hidden layers and RBFN. Then, the NNs are made granular using independent *normal* random variables with the optimal standard deviations given by (3.31). The resulting GNNs with different architectures are compared to each other, and the relationship between the granularity levels allocated to input dimensions and the output variations along those dimensions is investigated. In addition, for each data set, we perform the following three tasks:

1. The optimal granularity allocation for each GNN is slightly perturbed. The effect of this perturbations on the objective function is investigated.
2. The optimal vector of standard deviations is obtained for different input data, namely training data, testing data, and a combined set of both training and testing data.
3. The effect of NN parameters (such as the number of neurons in the hidden layers) in the resulting optimal granularity levels is investigated.

For all the following data sets, 75% of the entire data is used as training data and the rest as testing data. In addition, the training function used for all the NNs updates the weights and the biases according to Levenberg-Marquardt optimization.

The experimental results for synthetic and real data sets are presented in Secs. 4.1 and 4.2, respectively.

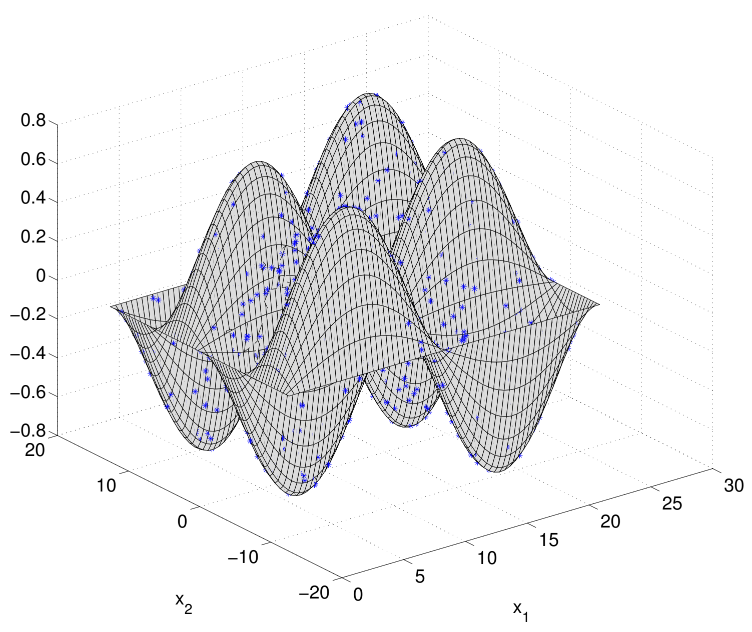


Figure 4.1. The 3D representation of the 2D function g_1. Blue points show the superimposed training and testing data.

4.1 Experimental Results on Synthetic Data Sets

This section contains the experimental results on synthetic data sets. To perform our analysis for synthetic data, we generate five data sets using five different functions. The synthetic data sets are generated based on a uniform distribution over the input spaces of the functions. For each data set, we have generated 600 input-output pairs for our experiments, where 75% of the input-output pairs (450 data points) are used as training data and the rest (150 data points) are used for testing. The value of ε for the first four data sets is set to -5 and it is equal to -6 for the last data set.

A. Synthetic Data #1

The function used to generate the first synthetic data is

$$g_1(x_1, x_2) = \sin\left(\frac{x_1}{2}\right)\sin\left(\frac{x_2}{4}\right), \quad 0 \leq x_1 \leq 8\pi, -4\pi \leq x_2 \leq 4\pi. \tag{4.1}$$

Fig. 4.1 shows the 3D representations of $g_1(x_1, x_2)$ for the specified ranges of parameters. The blue points on Fig. 4.1 show the superimposed training and testing data.

- *Construction of Numeric NNs*

  After splitting the data into training and testing, we have trained different neural networks to fit the data. The correlation coefficient R^2 is used as the performance measure of the constructed networks. R^2 ranges from 0 to 1, the closer R^2 is to 1 the better the model fits the data. Table 4.1 shows the parameters of different NN architectures used to fit the data as well as their corresponding amounts of R^2 for training and testing data sets.

	1 hidden layer MLP	**2 hidden layers MLP**	**RBFN**
No. of input neurons	2	2	2
No. of first hidden neurons	15	15	54
No. of second hidden neurons	-	20	-
Spread	-	-	10
R^2 on training data set	0.999997	0.999934	0.999848
R^2 on testing data set	0.999988	0.999795	0.999609

Table 4.1. Parameters and performances of the constructed NNs for synthetic data #1

- *Comparison of Optimal Standard Deviations For Different NNs Architectures*

  Now, we calculate the optimal allocation of information granularity to different input features of data set #1. As mentioned before, to make the constructed NNs granular, we use independent normal random variables with the standard deviations given by (3.31). To calculate the standard deviations, we first compute the coefficients a_i using (3.41), (3.43) or (3.46) depending on the network input-output relationship of the NN architecture used. We also derive a_is using the original function form (4.1) and use the resulting optimal standard deviations as a reference to comment on the accuracy of optimal standard deviations when NNs are used to fit data.

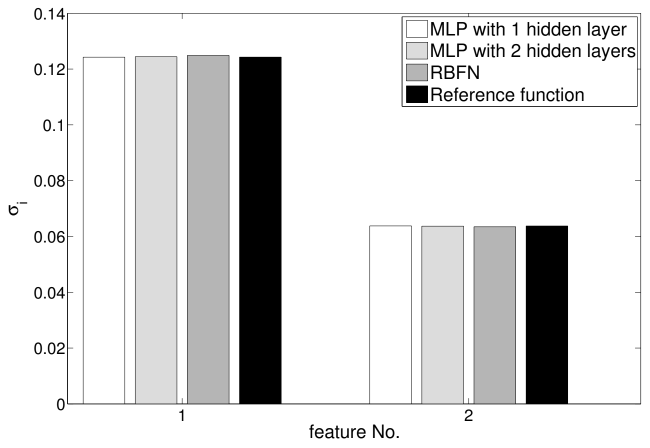


Figure 4.2. Comparison of the optimal allocations of standard deviations to the input features for different network architectures.

Fig. 4.2 shows the resulting optimal values of standard deviations allocated to different features when different NNs architectures are used. As we observe, all different network architectures led to almost the same optimal values for the standard deviations. The resulting standard deviations are very close to the standard deviations obtained by the reference function. This result is very interesting, because the mathematical forms of the input-output relationships of these NNs and the original function used to generate the data set are fundamentally different (see (3.40), (3.42) and (3.45)). Nonetheless, since the NNs are precisely trained to the data sets as the values of R^2 are very close to 1, they behave similar to the reference function which was used to generate the data set. This same behavior leads to similar results for optimal allocation of information granularity when various network architectures are used.

According to Fig. 4.1 and (4.1), the variations of the output is two times slower along the first input feature compared to the second input feature. In Fig. 4.2, we observe that the standard deviation given to the first feature is considerably more than the standard deviation given to the second feature. So, as a very significant point we observe that more uncertainty is allocated to the feature that the function shows smoother variations along its dimension.

- Task1: *Results for Perturbation Analysis*

  As the next step, we disturb the optimal vector of standard deviations and see how much the objective function (3.7) increases. Fig. 4.3 shows the effect of standard deviation perturbation on the objective function. In this figure, the horizontal axis is the amount of standard deviation disturbance. For each value of d, we calculate the average of objective function for 1000 randomly chosen vectors in the interval $[(1-d)\boldsymbol{\sigma}, (1+d)\boldsymbol{\sigma}]$. As expected, we see that the objective function is a monotonically increasing function of the disturbance level.

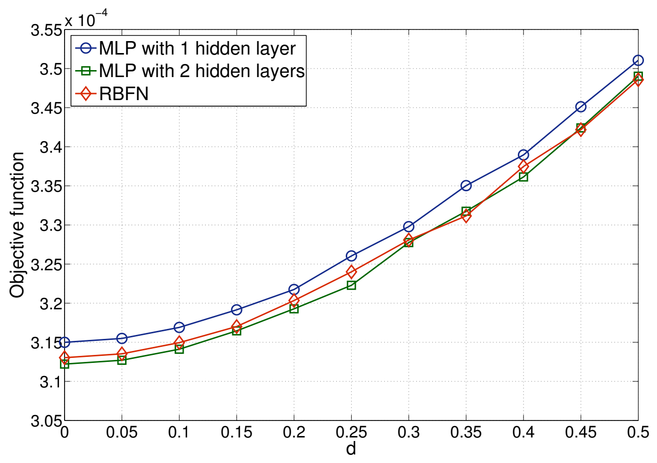
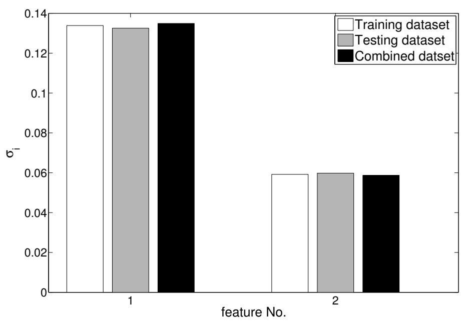


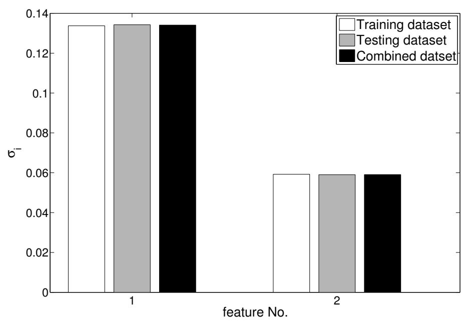
Figure 4.3. Effect of standard deviation perturbation on objective function.

- Task2: *Results For Training, Testing and Combined Data*

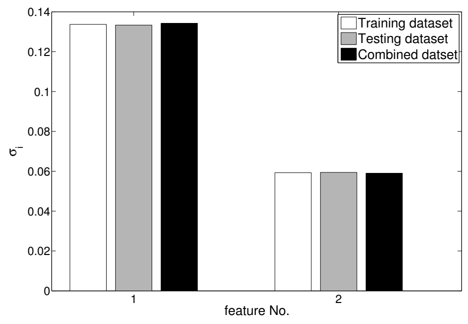
  From (3.31), one recalls that the optimal values of standard deviations depend on the numeric values of the input data. In Fig. 4.4, we compare the optimal standard deviation obtained when training, testing or a random combination of training and testing data are used to determine the optimal standard deviations. This study is important because the trained networks have not seen the testing data and one needs to know how similar the optimal standard deviations are in these two cases. For all GNN architectures, we observe that the amount of the resulting standard deviations for the training, testing, and combined data are close to each other. So, the standard deviations obtained by using training data can be used for testing data as well, without any considerable impact on the performance of the GNN.



(a) MLP with one hidden layer



(b) MLP with two hidden layers

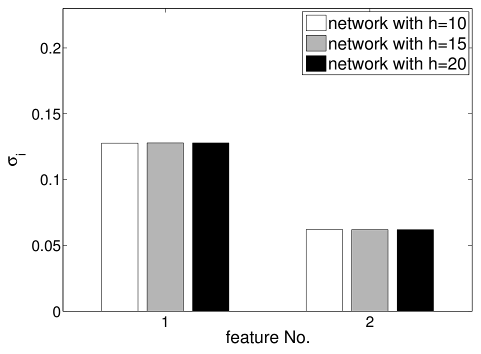


(c) RBFN

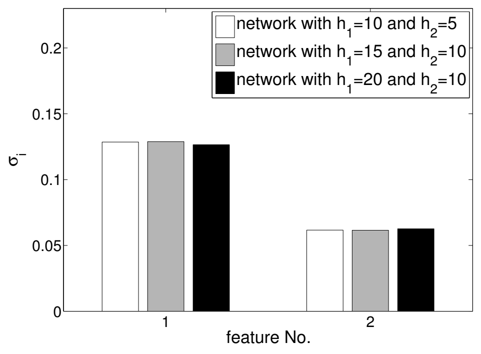
Figure 4.4. Optimal standard deviations when training, testing, or the combined data set are used.

- Task3: *Results For Different Network Paramaters*

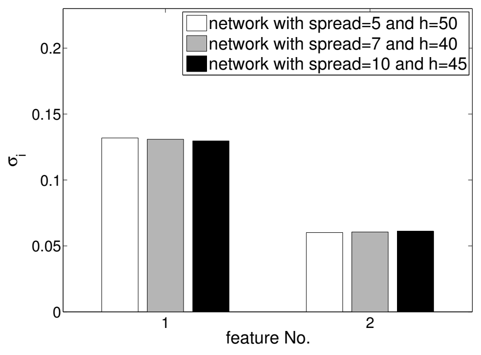
  In previous parts, we discussed the sensitivity of the optimal standard deviation values, first to the GNN architecture used, and second, to the type of data used for derivation of the optimal standard deviatins (whether it is the training or testing data). In both cases, we observed that different scenarios led to almost the same allocations for standard deviations. To further investigate the robustness of the optimal values, we consider their variations to different values of NN parameters as well. In Fig. 4.5, we compare the optimal allocations for different number of neurons in the hidden layers of the MLP networks, and different spread values for RBFN. Again, we observe that the optimal values allocated to each feature remains almost the same when different NN parameters are used. These results show the robustness of the optimal results to the changes of parameters.



(a) MLP with one hidden layer



(b) MLP with two hidden layers



(c) RBFN

Figure 4.5. Comparison of the optimal allocations of standard deviations to the input features for different network parameters.

For the other synthetic data sets, we follow the same procedure as what we did for synthetic data #1 and present the results. We will discuss our observations for all synthetic data sets at the end of this section.

B. Function #2

For generation of synthetic data #2 we use function

$$g_2(x_1, x_2) = x_1^2 + x_2^2, \quad -10 \le x_1, x_2 \le 10 \tag{4.2}$$

with the 3D representation in Fig. 4.6.

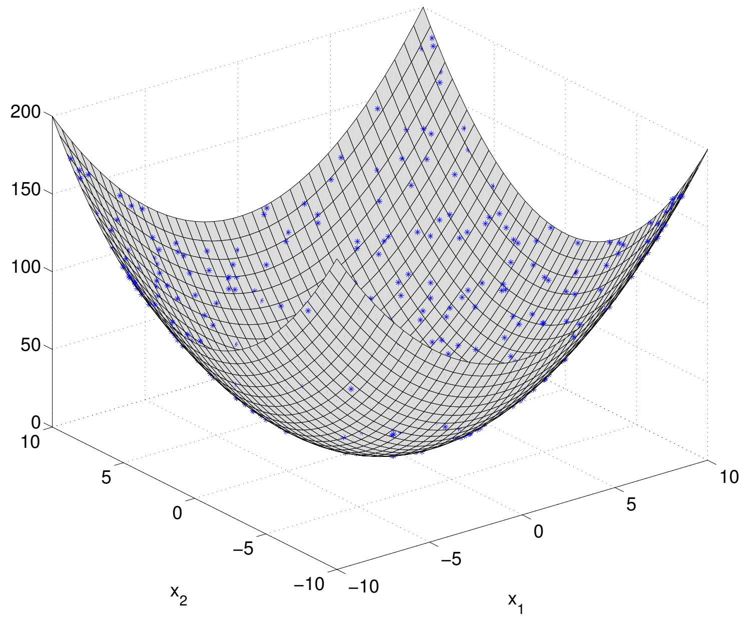


Figure 4.6. The 3D representation of 2D function g_2. Blue points show the superimposed training and testing data.

- *Construction of Numeric NNs*

	1 hidden layer MLP	**2 hidden layers MLP**	**RBFN**
No. of input neurons	2	2	2
No. of first hidden neurons	5	5	10
No. of second hidden neurons	-	5	-
Spread	-	-	50
R^2 on training data set	1	1	1
R^2 on testing data set	1	1	1

Table 4.2. Parameters and performances of the constructed NNs for the synthetic data #2

- *Comparison of Optimal Variances For Different Architectures*

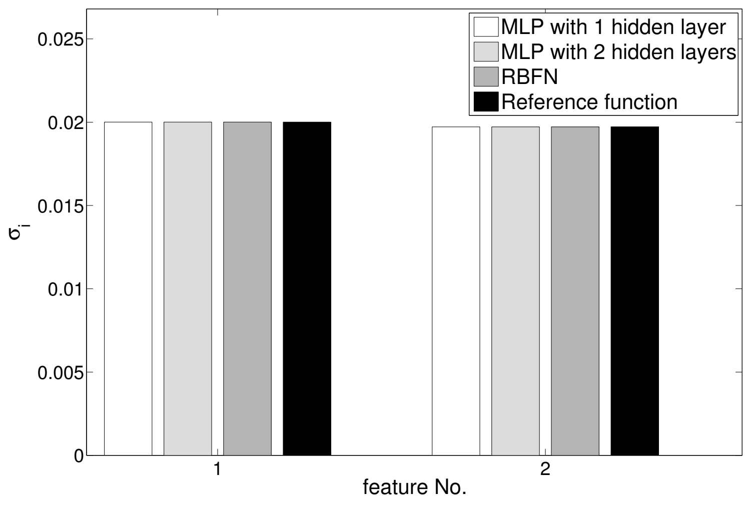


Figure 4.7. Comparison of the optimal standard deviations for different network architectures.

This data set has the same level of output variations along its input dimensions. In Fig. 4.7, we observe that the standard deviations allocated to the input features are almost the same.

- Task1: *Results for Perturbation Analysis*

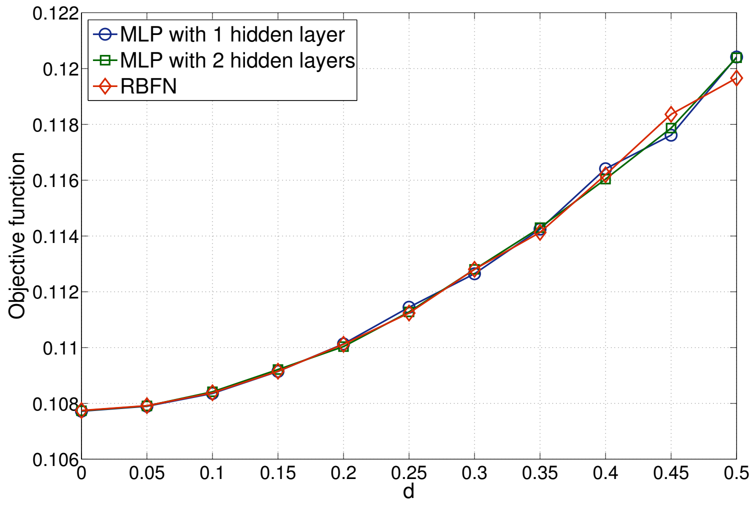
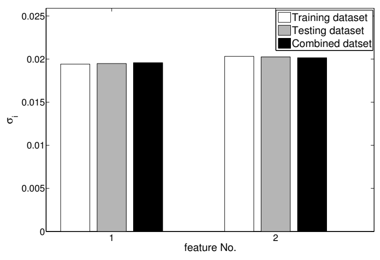
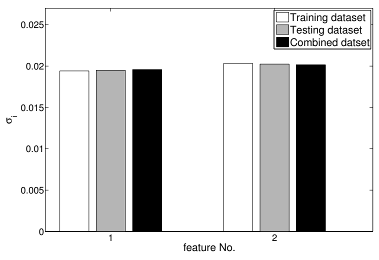


Figure 4.8. Effect of standard deviation perturbations on objective function.

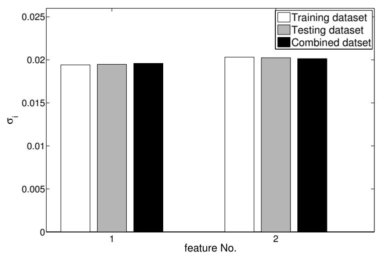
- Task2: *Results For Training, Testing and Combined Data*



(a) MLP with one hidden layer



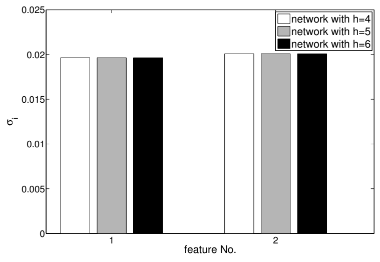
(b) MLP with two hidden layers



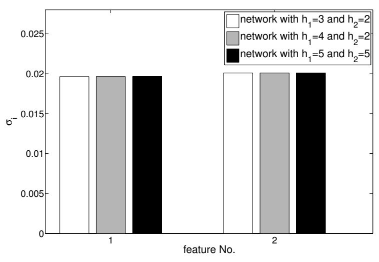
(c) RBFN

Figure 4.9. Optimal standard deviations for training, testing and combined data sets.

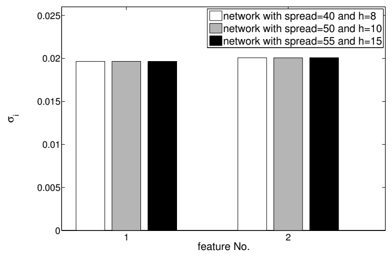
- Task3: *Results For Different Network Paramaters*



(a) MLP with one hidden layer



(b) MLP with two hidden layers



(c) RBFN

Figure 4.10. Comparison of optimal standard deviations for different network parameters.

C. Function #3

For generation of synthetic data #3 we use function

$$g_3(x_1, x_2) = 10(x_2^2 - x_1)^2 + (1 - x_1)^2, \quad -10 \leq x_1, x_2 \leq 10 \tag{4.3}$$

with the 3D representation in Fig. 4.11.

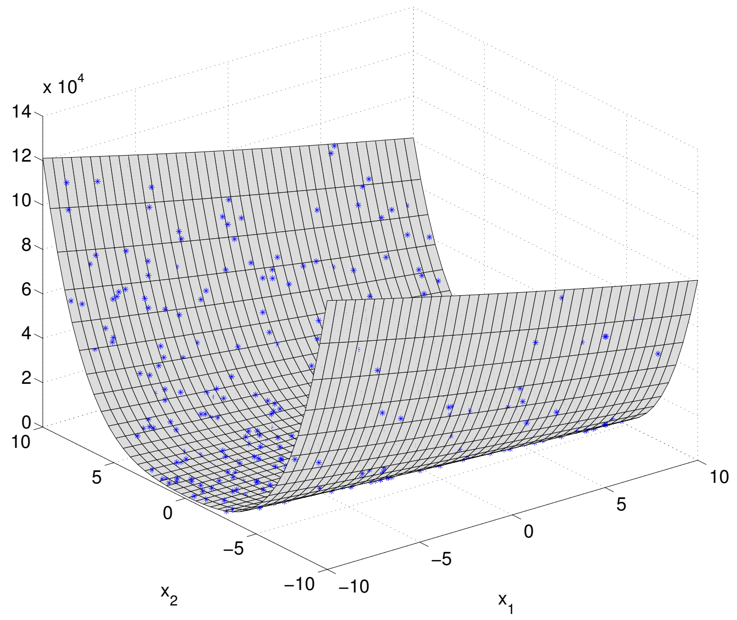


Figure 4.11. The 3D representation of 2D function g_3. Blue points show the superimposed training and testing data.

- *Construction of Numeric NNs*

	1 hidden layer MLP	**2 hidden layers MLP**	**RBFN**
No. of input neurons	2	2	2
No. of first hidden neurons	10	5	38
No. of second hidden neurons	-	10	-
Spread	-	-	20
R^2 on training data set	1	1	1
R^2 on testing data set	1	1	1

Table 4.3. Parameters and performances of the constructed NNs for the synthetic data #3

- *Comparison of Optimal Variances For Different Architectures*

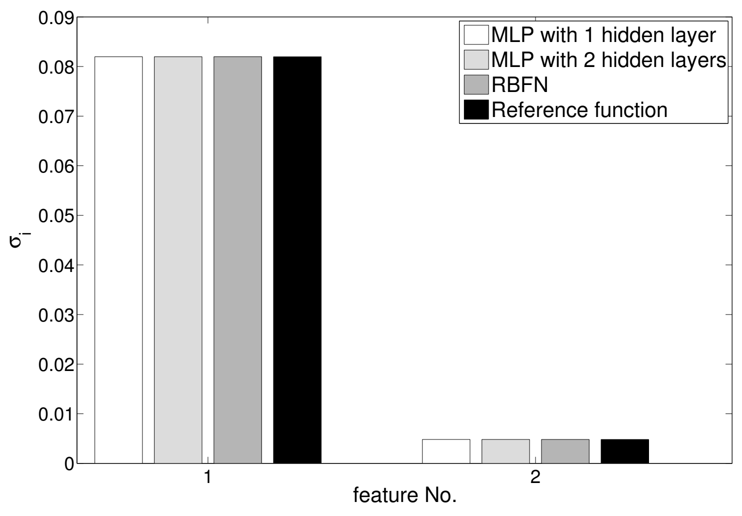


Figure 4.12. Comparison of the optimal standard deviations for different network architectures.

Fig. 4.11 shows that g_3 varies almost linearly along x_1 and it has quadratic changes along x_2. Therefore, this function changes faster with x_2 compared to x_1. We observe in Fig. 4.12 that more standard deviation is allocated to x_1.

- Task1: *Results for Perturbation Analysis*

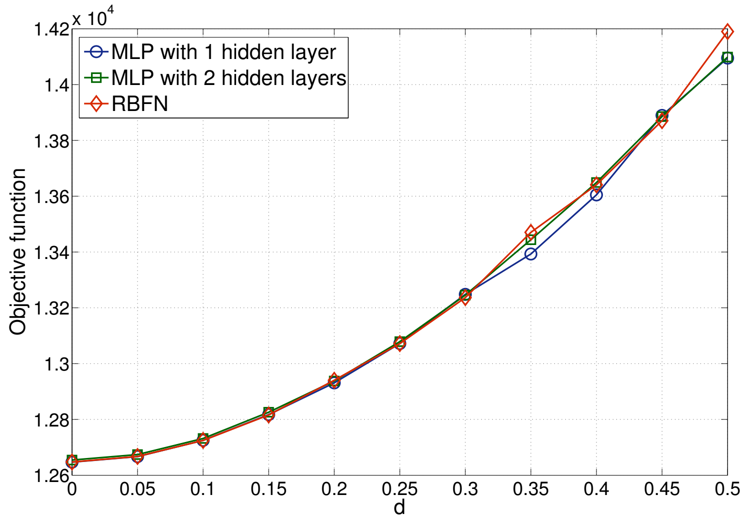
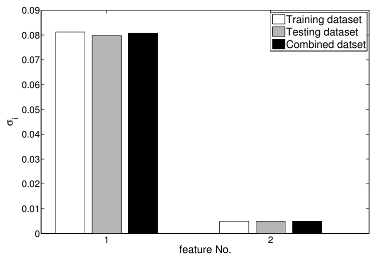
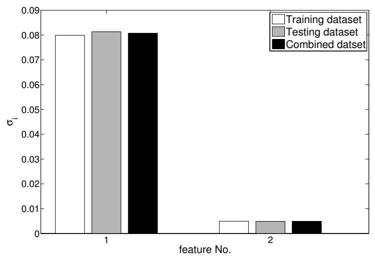


Figure 4.13. Effect of standard deviation perturbations on objective function.

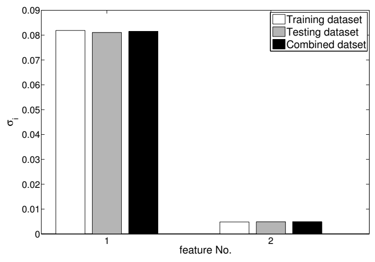
- Task2: *Results For Training, Testing and Combined Data*



(a) MLP with one hidden layer



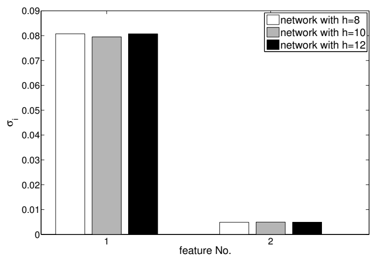
(b) MLP with two hidden layers



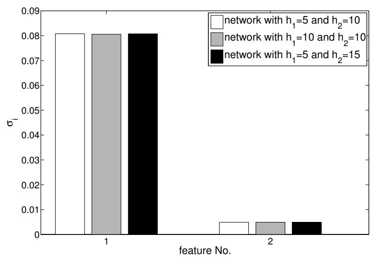
(c) RBFN

Figure 4.14. Optimal standard deviations for training, testing and combined data sets.

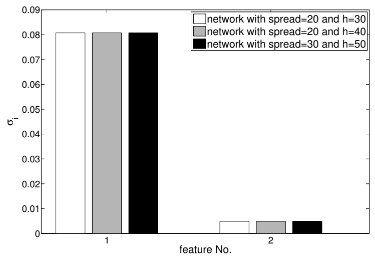
- Task3: *Results For Different Network Paramaters*



(a) MLP with one hidden layer



(b) MLP with two hidden layers



(c) RBFN

Figure 4.15. Comparison of optimal standard deviations for different network parameters.

D. Function #4

For generation of synthetic data #4 we use function

$$g_4(x_1, x_2) = 4\frac{\sin(x_1)}{x_1}\frac{\sin(\frac{x_2}{4})}{x_2}, \quad -7.5 \le x_1 \le 7.5, 0 \le x_2 \le 15 \tag{4.4}$$

with the 3D representation in Fig. 4.16.

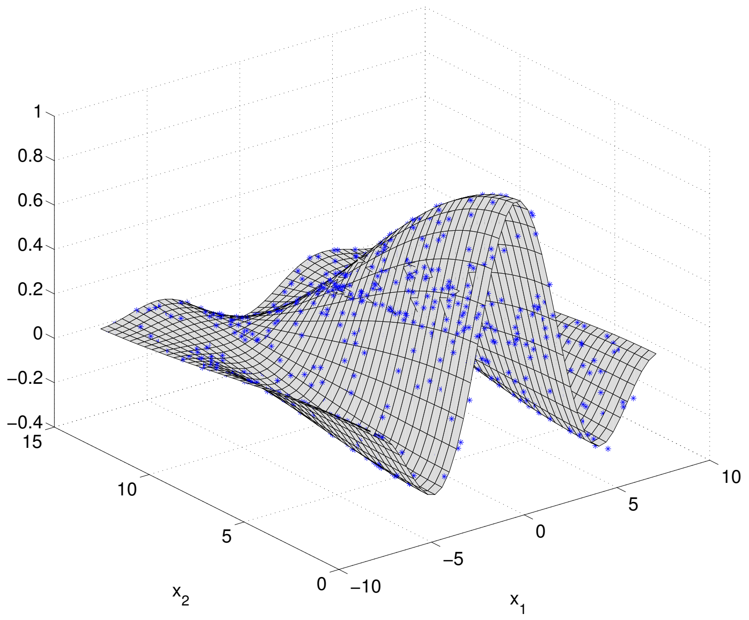


Figure 4.16. The 3D representation of 2D function g_4. Blue points show the superimposed training and testing data.

- *Construction of Numeric NNs*

	1 hidden layer MLP	**2 hidden layers MLP**	**RBFN**
No. of input neurons	2	2	2
No. of first hidden neurons	30	20	50
No. of second hidden neurons	-	30	-
Spread	-	-	3
R^2 on training data set	0.999944	0.999981	0.99981
R^2 on testing data set	0.999924	0.999913	0.99978

Table 4.4. Parameters and performances of the constructed NNs for the synthetic data #4

- *Comparison of Optimal Variances For Different Architectures*

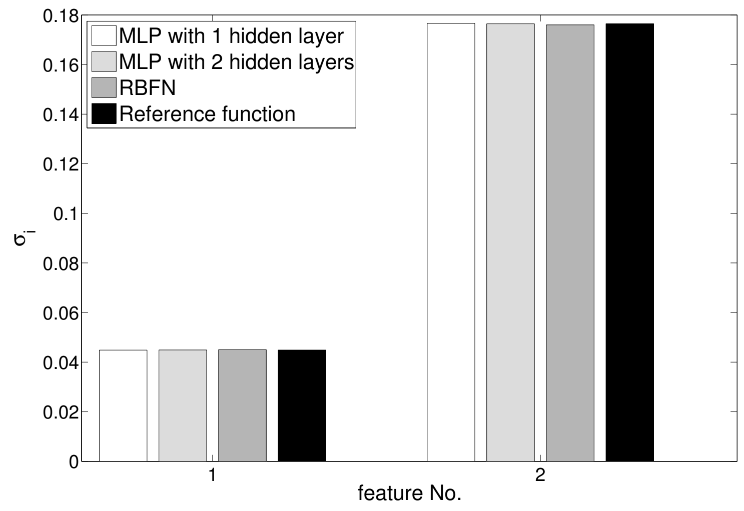


Figure 4.17. Comparison of the optimal standard deviations for different network architectures.

As we observe in Fig. 4.16, in the specified ranges of parameters, the changes of function $g_4(x_1, x_2)$ along input x_1 is more than the changes of function in the direction of x_2. Fig. 4.17 shows the optimal value of σ_1 is smaller than the optimal value of σ_2.

- Task1: *Results for Perturbation Analysis*

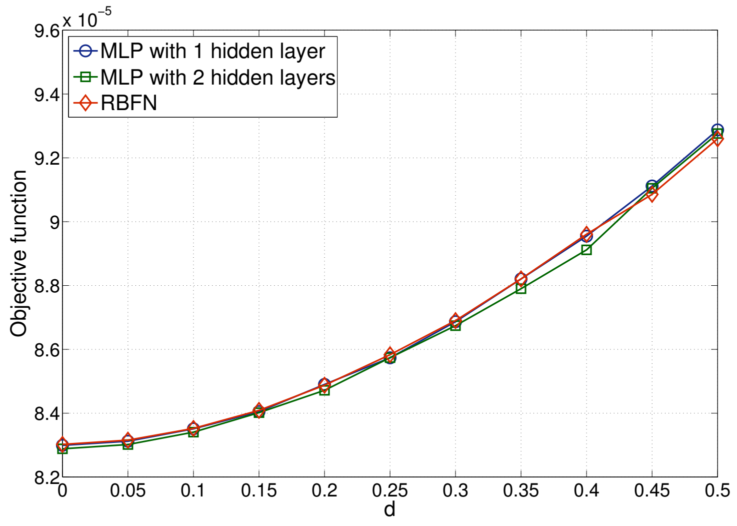
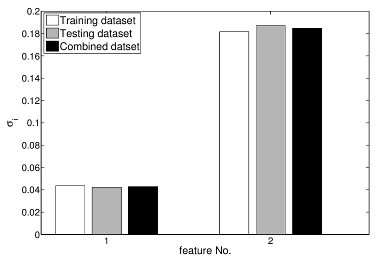
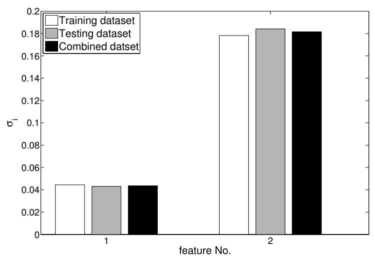


Figure 4.18. Effect of standard deviation perturbations on objective function.

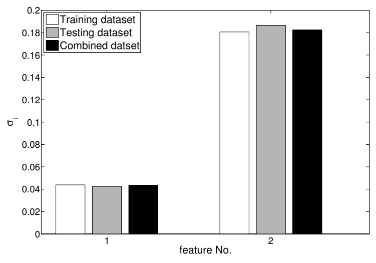
• Task2: *Results For Training, Testing and Combined Data*



(a) MLP with one hidden layer



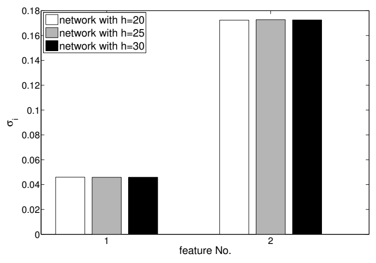
(b) MLP with two hidden layers



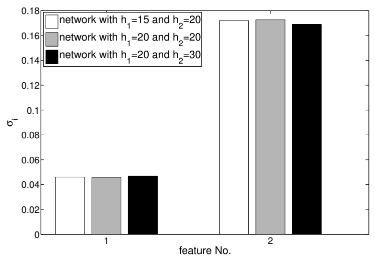
(c) RBFN

Figure 4.19. Comparison of optimal standard deviations for training, testing and combined data sets.

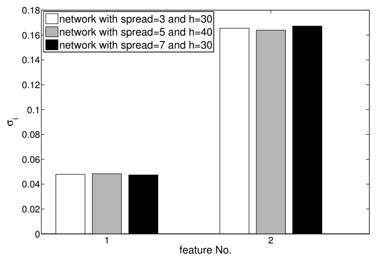
• Task3: *Results For Different Network Parameters*



(a) MLP with one hidden layer



(b) MLP with two hidden layers



(c) RBFN

Figure 4.20. Comparison of optimal standard deviations for different network parameters.

E. Function #5

For generation of synthetic data #5 we use function

$$g_5(x_1, x_1, x_3) = 0.8 \sin\left(\frac{x_1}{8}\right) \sin\left(\frac{x_2}{4}\right) \sin\left(\frac{x_3}{2}\right), \quad 0 \le x_1 \le 10, -5 \le x_2, x_3 \le 5. \tag{4.5}$$

- *Construction of Numeric NNs*

	1 hidden layer MLP	**2 hidden layers MLP**	**RBFN**
No. of input neurons	3	3	3
No. of first hidden neurons	20	15	70
No. of second hidden neurons	-	20	-
Spread	-	-	10
R^2 on training data set	0.999726	0.998826	0.999457
R^2 on testing data set	0.999779	0.995244	0.998884

Table 4.5. Parameters and performances of the constructed NNs for the synthetic data #5

- *Comparison of Optimal Variances For Different Architectures*

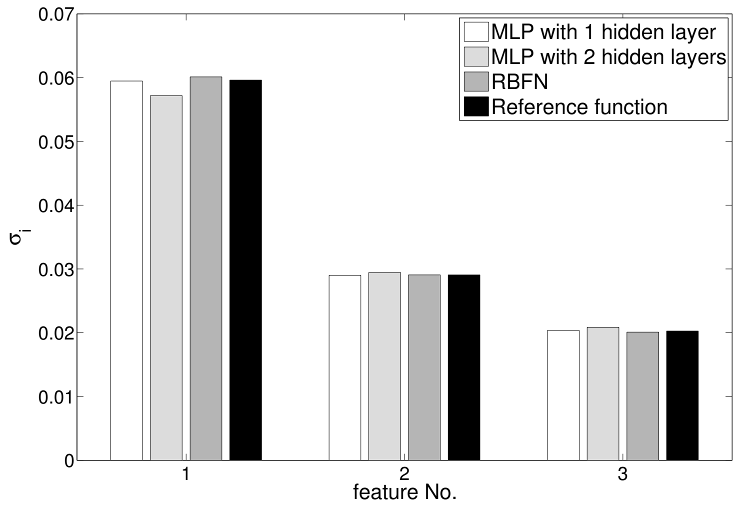


Figure 4.21. Comparison of optimal standard deviations for different network architectures.

From the mathematical form of $g_5(\cdot)$ in (4.5), one notes that $g_5(\cdot)$ varies along x_3 two times faster than x_2 and it varies along x_2 two times faster than x_1. Interestingly, if we rank the input features by the amount of standard deviations allocated to them (Fig. 4.21), we observe that feature 3 is given the smallest standard deviation while feature 1 is allocated the highest standard deviation. In other words, the faster the output variations are along an input dimension, the smaller standard deviation is given to that dimension.

- Task1: *Results for Perturbation Analysis*

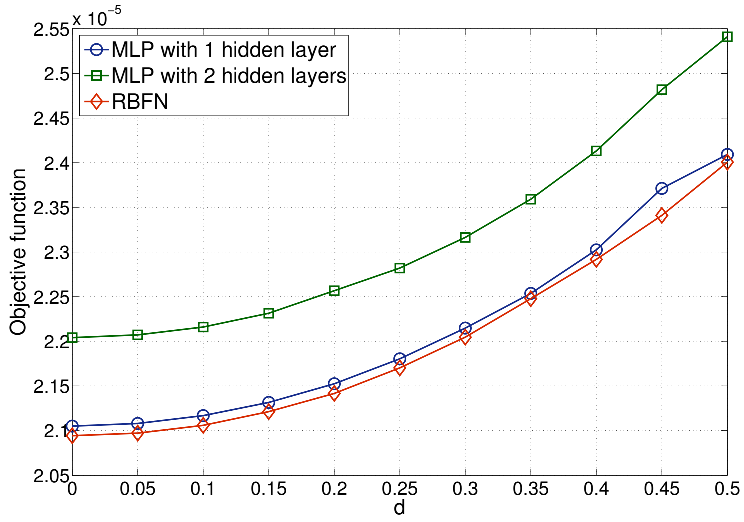
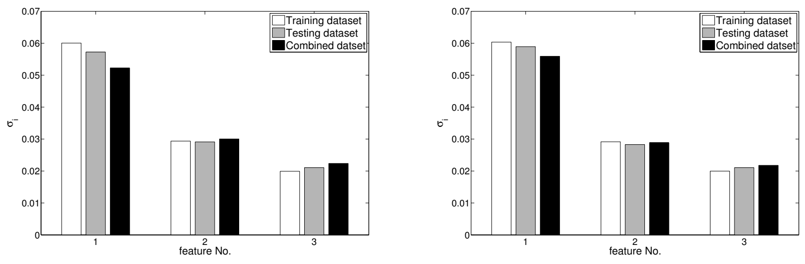


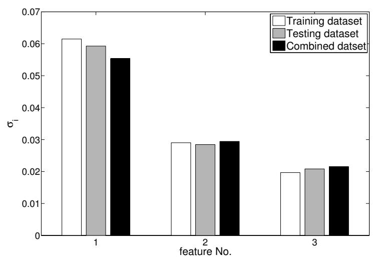
Figure 4.22. Effect of standard deviation perturbations on objective function.

- Task2: *Results For Training, Testing and Combined Data*



(a) MLP with one hidden layer

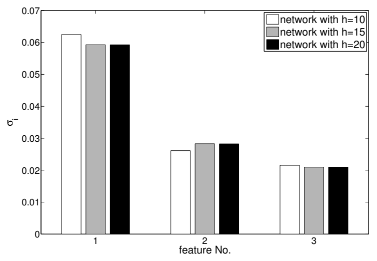
(b) MLP with two hidden layers



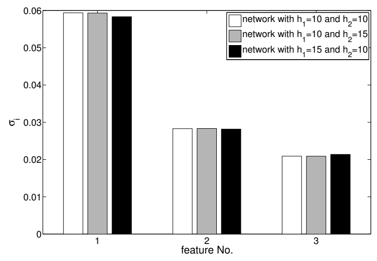
(c) RBFN

Figure 4.23. Comparison of optimal standard deviations for training, testing and combined data sets.

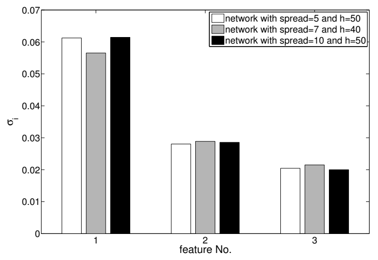
- Task3: *Results For Different Network Paramaters*



(a) MLP with one hidden layer



(b) MLP with two hidden layers



(c) RBFN

Figure 4.24. Comparison of optimal standard deviations for different network parameters.

Discussion of Experimental Results on Synthetic Data Sets

Based on the experimental results obtained for synthetic data sets, we observe that when NNs are properly trained (R^2 is close to one), the following points are true:

- The faster the output data changes along an input dimension, the smaller the standard deviation (higher granularity) is allocated to that input dimension.
- No matter which GNN architectures is used, the optimal standard deviations (granularity levels) allocated to the input random variables remain almost the same.
- Using training data, testing data or a mixture of the training and testing data in the allocation process leads to almost the same optimal standard deviations.
- As far as NNs are properly trained, variation of their network parameters (like the number of neurons in hidden layers) does not affect the optimal standard deviations for the input random variables.

4.2 Experimental Results on Real Data Sets

This subsection contains the experimental results on GNNs constructed for real data sets. For the experiments in this section, we use Auto MPG, Boston, Bodyfat, Servo, Engine, and Mackey-Glass data sets [22]. A short description about each data set is provided in its corresponding subsection. Similar to our experiments on synthetic data sets, for each real data set, we first construct numeric NNs with different architectures. The parameters and performance metrics (R^2) of the resulting numeric NNs are provided and the scatter plots corresponding to each NN are presented. After constructing numeric NNs, we augment their inputs by granulating them using normal random variables. We compare the optimal standard deviations when different NN structures are used. Note that unlike the synthetic data, for real data sets we do not have any reference function to compare the results with. Since different data sets have different number of input neurons, the value of ϵ is chosen differently for each one. The values of epsilon for different data sets are reported in Table 4.6.

Auto MPG	Boston	Bodyfat	Servo	Engine	Mackey-Glass
-25	-40	-25	-5	-5	-10

Table 4.6. The levels of total information granularity ϵ allocated to the inputs of the GNNs for different data sets

We also perform the three tasks described in Sec. 4.1 on each GNNs constructed for the real data sets. In the remainder of this section, we first present all the experimental results we obtained, and then, we summarize and discuss our observations.

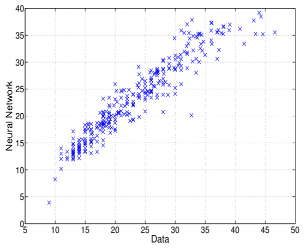
A. Automobile Miles-Per-Gallon (MPG) Data

This data set has seven input features and its output is the amount of fuel consumption of a car in miles per gallon. After removing incomplete data points, we have 392 input-output pairs where 294 pairs are chosen for training and 98 pairs for testing.

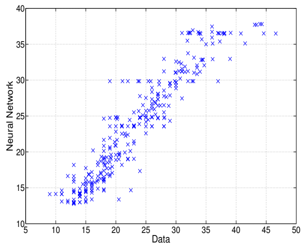
- *Construction of Numeric NNs*

	1 hidden layer MLP	2 hidden layers MLP	RBFN
Input neurons No.	7	7	7
First hidden neurons No.	15	10	70
Second hidden neurons No.	-	5	-
Spread	-	-	185
R^2 on training data	0.895252	0.881189	0.921261
R^2 on testing data	0.838109	0.765013	0.746824

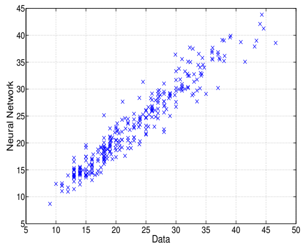
Table 4.7. Performance and the parameters of constructed networks for Auto MPG data



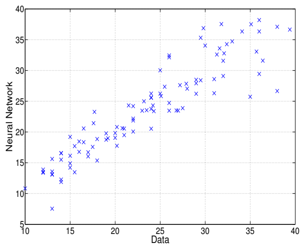
(a) One hidden layer MLP



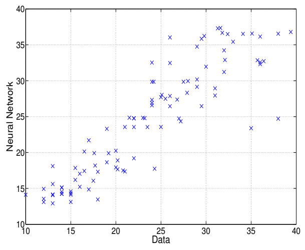
(b) Two hidden layer MLP



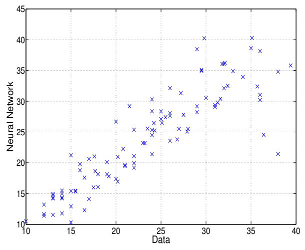
(c) RBFN



(d) One hidden layer MLP



(e) Two hidden layer MLP



(f) RBFN

Figure 4.25. Scatter plots of the output of the network versus data: (a), (b), (c) training data (d), (e), (f) testing data

- *Comparison of Optimal Variances For Different Architectures*

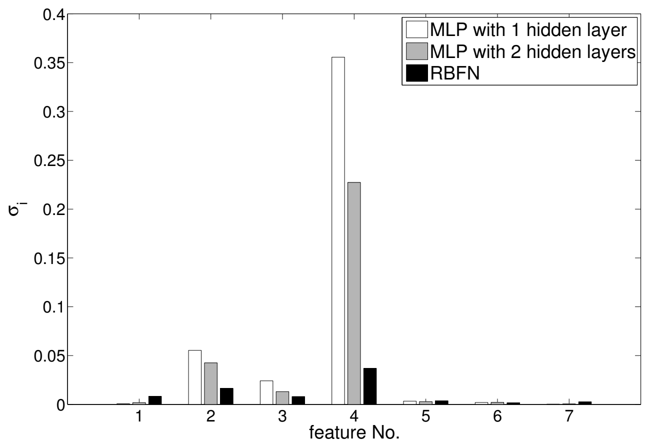


Figure 4.26. Comparison of the optimal standard deviations for different network architectures and for training data.

If we rank the input features in (decreasing) order of the variance levels allocated to them, we observe that all three architectures have led to almost the same rankings. If we look at the amount of standard deviations allocated to each input feature, we see that the results for MLP with one and two hidden layers are almost the same. Specifically speaking, according to Fig. 4.28, we see that these two kind of networks allocate most of the standard deviations to the features 4, 2, and 3, respectively, and other features' portions are almost nothing. However, RBFN considers other features as well and distributes uncertainty more uniformly among input features.

- Task1: *Results for Perturbation Analysis*

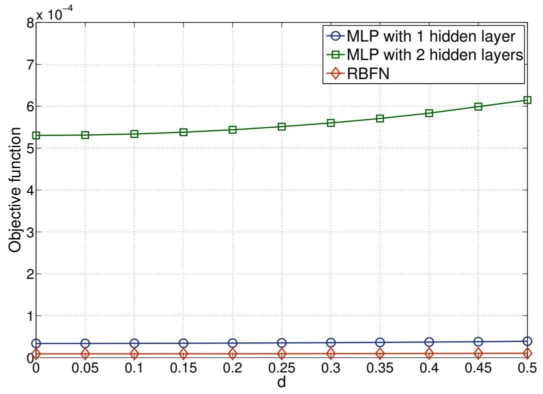
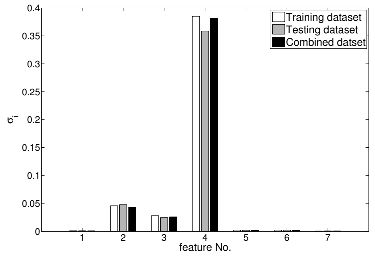
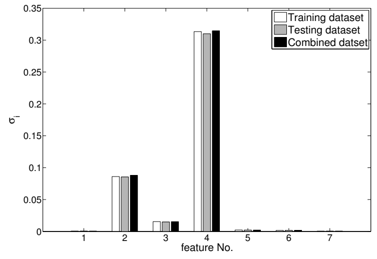


Figure 4.27. Effect of standard deviation perturbations on objective function.

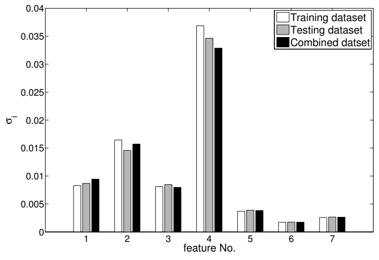
- Task2: *Results For Training, Testing and Combined Data*



(a) One hidden layer MLP



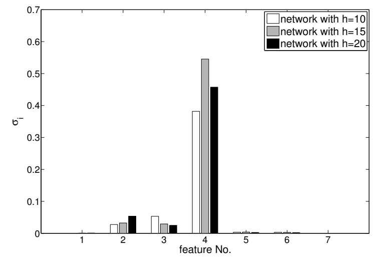
(b) Two hidden layer MLP



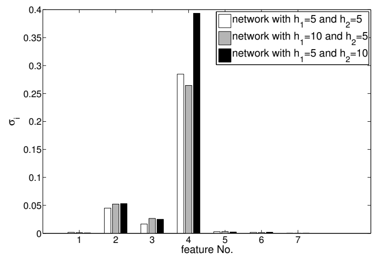
(c) RBFN

Figure 4.28. Comparison of optimal standard deviations for training, testing and combined data sets.

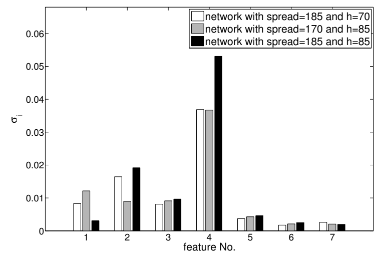
- Task3: *Results For Different Network Paramaters*



(a) One hidden layer MLP



(b) Two hidden layer MLP



(c) RBFN

Figure 4.29. Comparison of optimal standard deviations for different network parameters.

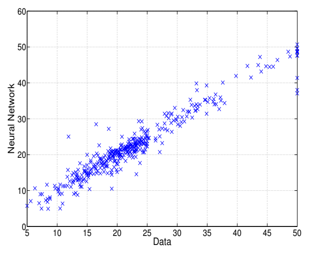
B. Boston Housing Data

This data set concerns the median value of owner-occupied home in Boston area. The number of input-output pairs for this data is 506 where each input is a vector of 13 variable.

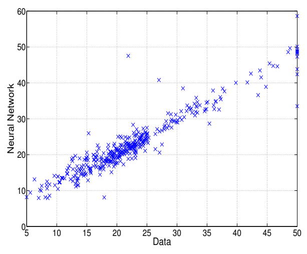
- *Construction of Numeric NNs*

	1 hidden layer MLP	2 hidden layers MLP	RBFN
Input neurons No.	13	13	13
First hidden neurons No.	15	30	85
Second hidden neurons No.	-	30	-
Spread	-	-	120
R^2 on training data	0.926253	0.895901	0.920047
R^2 on testing data	0.800941	0.833183	0.851525

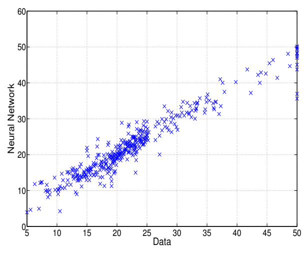
Table 4.8. Performance and the parameters of constructed networks for Boston housing data



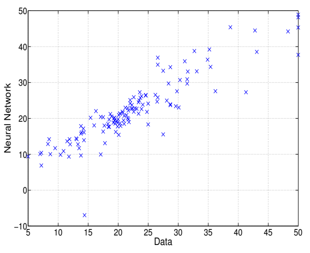
(a) One hidden layer MLP



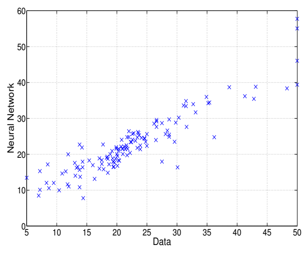
(b) Two hidden layer MLP



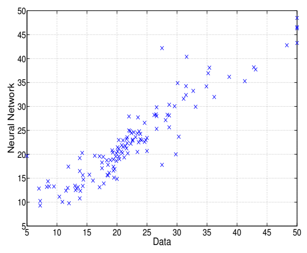
(c) RBFN



(d) One hidden layer MLP



(e) Two hidden layer MLP



(f) RBFN

Figure 4.30. Scatter plots of the output of the network versus data: (a), (b), (c) training data (d), (e), (f) testing data

- *Comparison of Optimal Variances For Different Architectures*

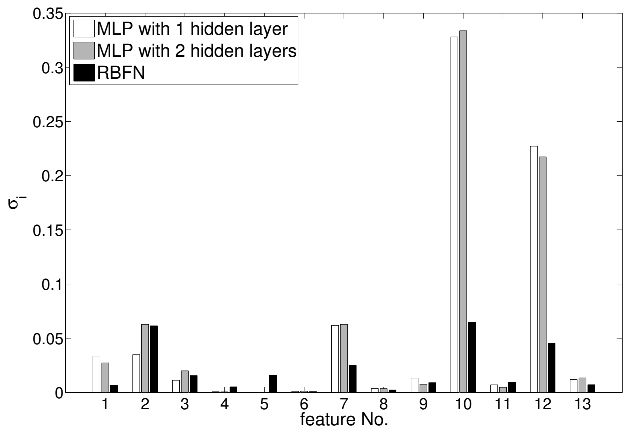


Figure 4.31. Comparison of the optimal standard deviations for different network architectures and for training data.

Similar to our observation for Auto MPG data, we observe that both MLP architectures lead to very similar rankings of the features in order of the variance levels allocated. The feature ranking for RBFN is also similar to MLP networks. The ranking is better preserved for features that are given the highest and the lowest variances and gets more irregular for features with medium variance levels. Again, here, we see that there exists some features that MLPs do not give them almost anything (such as features 4 and 5) although RBFN allocate them some value. In summary, the three networks allocate the most amount of standard deviations to the features 12, 10, 2, and 7, and almost nothing is given to feature 6.

- Task1: *Results for Perturbation Analysis*

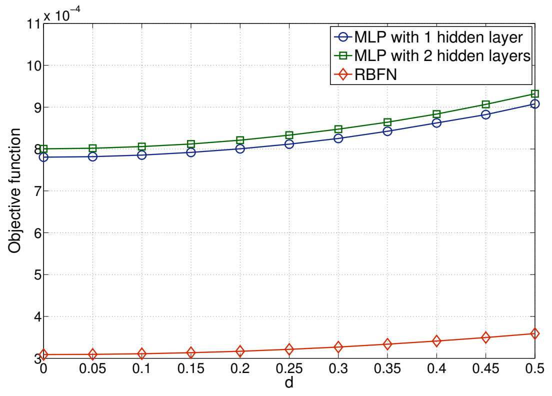
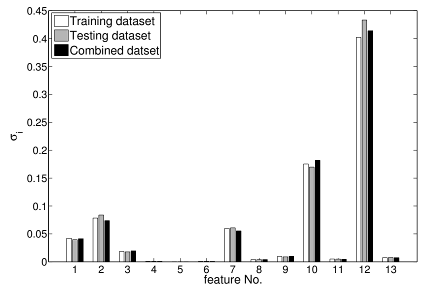
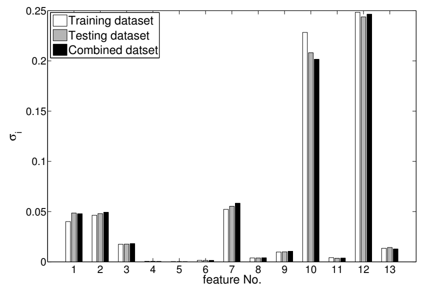


Figure 4.32. Effect of standard deviation perturbations on objective function.

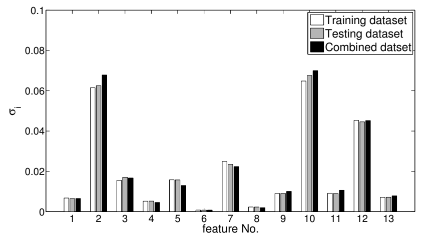
- Task2: *Results For Training, Testing and Combined Data*



(a) One hidden layer MLP



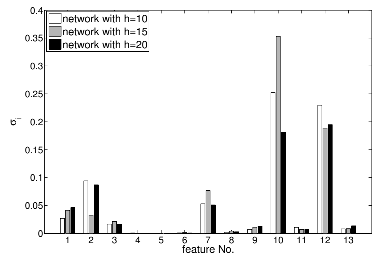
(b) Two hidden layer MLP



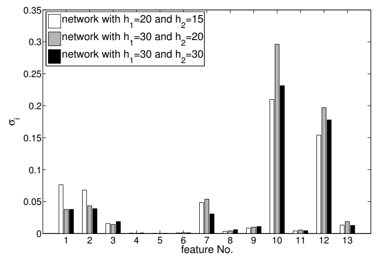
(c) RBFN

Figure 4.33. Comparison of optimal standard deviations for training, testing and combined data sets.

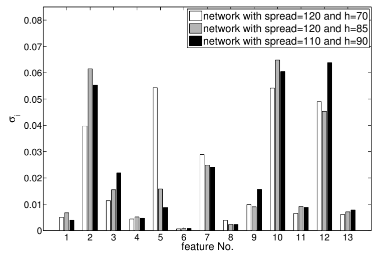
- Task3: *Results For Different Network Paramaters*



(a) One hidden layer MLP



(b) Two hidden layer MLP



(c) RBFN

Figure 4.34. Comparison of optimal standard deviations for different network parameters and for training data.

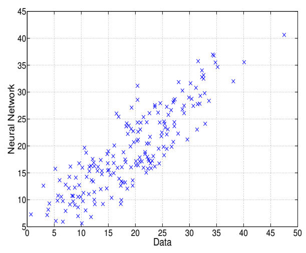
C. Bodyfat Percentage Data

This data set contains the percentage of body fat, age, height, weight and circumference measurements for ten various body parts of 252 men. The goal is estimating the body fat percentage using the provided body measurements.

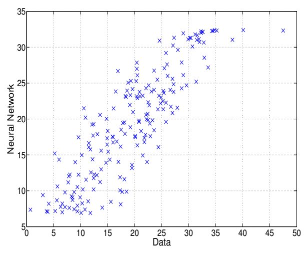
- *Construction of Numeric NNs*

	1 hidden layer MLP	2 hidden layers MLP	RBFN
Input neurons No.	13	13	13
First hidden neurons No.	2	3	24
Second hidden neurons No.	-	3	-
Spread	-	-	150
R^2 on training data	0.736916	0.7483	0.776243
R^2 on testing data	0.727712	0.738665	0.705003

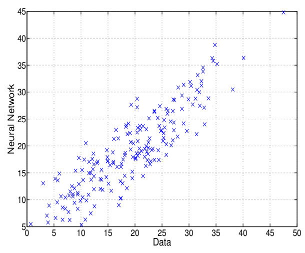
Table 4.9. Performance and the parameters of constructed networks for Bodyfat data



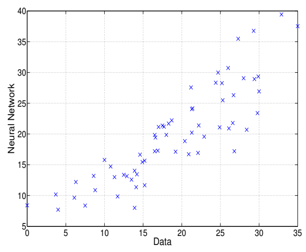
(a) One hidden layer MLP



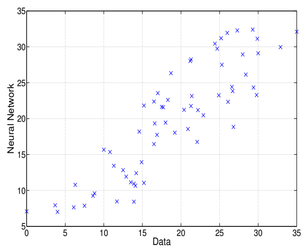
(b) Two hidden layer MLP



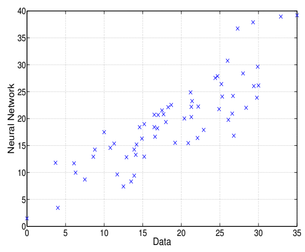
(c) RBFN



(d) One hidden layer MLP



(e) Two hidden layer MLP



(f) RBFN

Figure 4.35. Scatter plots of the output of the network versus data: (a), (b), (c) training data (d), (e), (f) testing data

- *Comparison of Optimal Variances For Different Architectures*

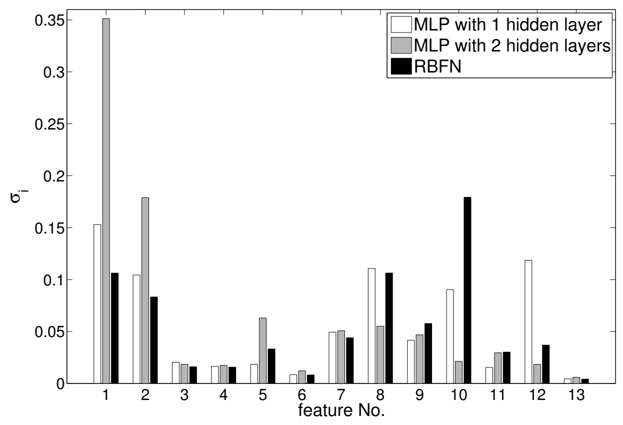


Figure 4.36. Comparison of the optimal standard deviations for different network architectures and for training data.

In Table 4.9, we observe that R^2 on training and testing data is between 0.7 and 0.8, which is lower than the R^2 values for the previous two data sets. In Figs. 4.36 and 4.39, we observe how this lower performance has affected the optimal standard deviations for different architectures and different network parameters. Unlike the previous data sets, we can observe some differences in the rankings of the features in order of the granularity levels allocated to them when we use different architectures or when we change the parameters of the networks. Despite the differences in the rankings, we still observe that some features are given large standard deviations (like features 1 and 2) and some features are given small standard deviations (like features 3, 4, 6, 11, 13) regardless of the architecture used. This implies a consistent trend in granularity allocation using the proposed granulation process.

- Task1: *Results for Perturbation Analysis*

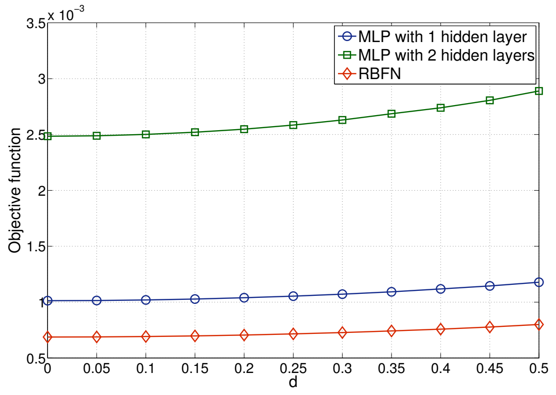
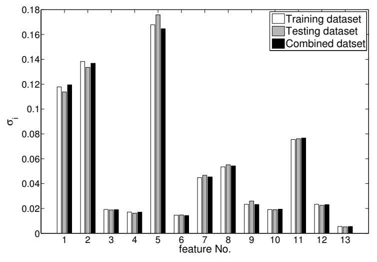
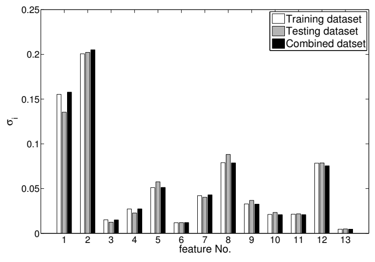


Figure 4.37. Effect of standard deviation perturbations on objective function.

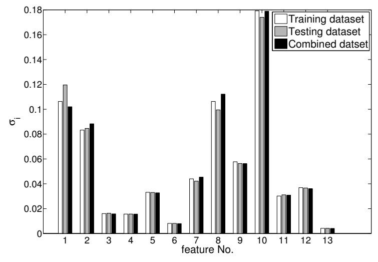
- Task2: *Results For Training, Testing and Combined Data*



(a) One hidden layer MLP



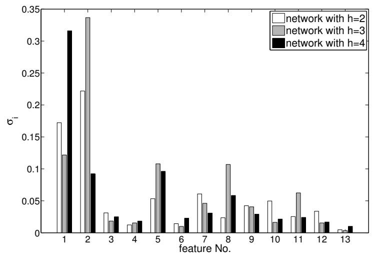
(b) Two hidden layer MLP



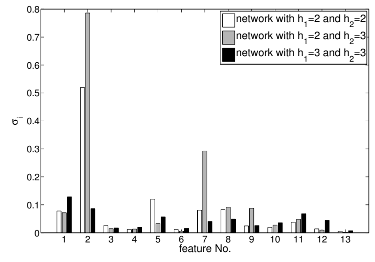
(c) RBFN

Figure 4.38. Comparison of optimal standard deviations for training, testing and combined data sets.

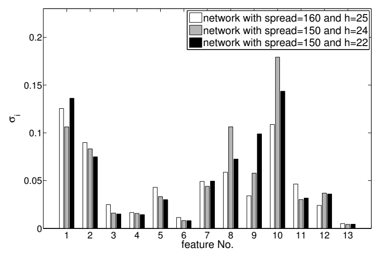
- Task3: *Results For Different Network Paramaters*



(a) One hidden layer MLP



(b) Two hidden layer MLP



(c) RBFN

Figure 4.39. Comparison of optimal standard deviations for different network parameters and for training data.

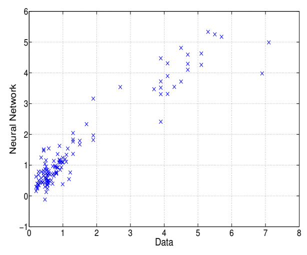
D. Servo Data

This data set deals with the simulation of a servo system. The output which is the the rise time of the system can be treated as a multi-variable function of four input variable. This data contains 167 instances.

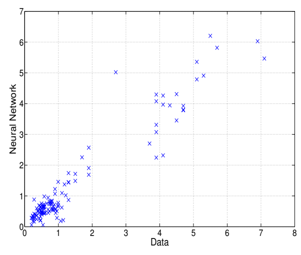
- *Construction of Numeric NNs*

	1 hidden layer MLP	2 hidden layers MLP	RBFN
Input neurons No.	4	4	4
First hidden neurons No.	10	5	13
Second hidden neurons No.	-	10	-
Spread	-	-	5
R^2 on training data	0.894815	0.912314	0.835501
R^2 on testing data	0.886201	0.854826	0.767038

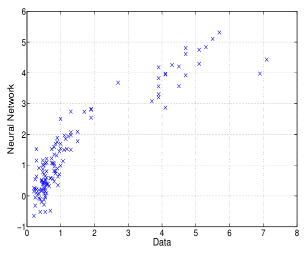
Table 4.10. Performance and the parameters of constructed networks for Servo data



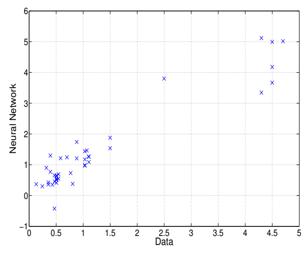
(a) One hidden layer MLP



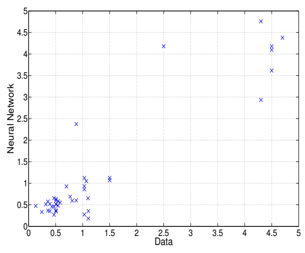
(b) Two hidden layer MLP



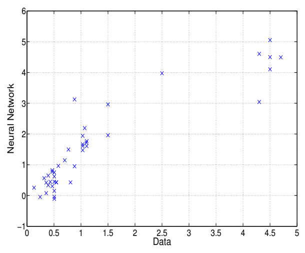
(c) RBFN



(d) One hidden layer MLP



(e) Two hidden layer MLP



(f) RBFN

Figure 4.40. Scatter plots of the output of the network versus data: (a), (b), (c) training data (d), (e), (f) testing data

- *Comparison of Optimal Variances For Different Architectures*

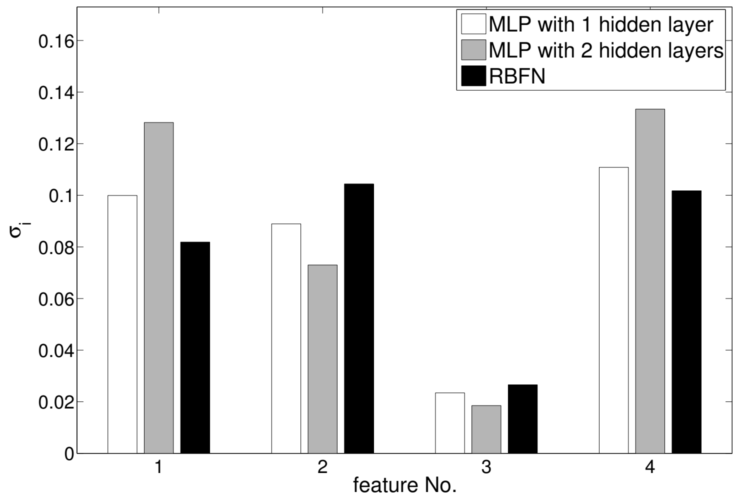


Figure 4.41. Comparison of the optimal standard deviations for different network architectures and for training data.

According to Fig. 4.41, all three different networks allocate most of standard deviations to the features 1, 2, and 4, and feature 3 receives the minimum amount of standard deviation.

- Task1: *Results for Perturbation Analysis*

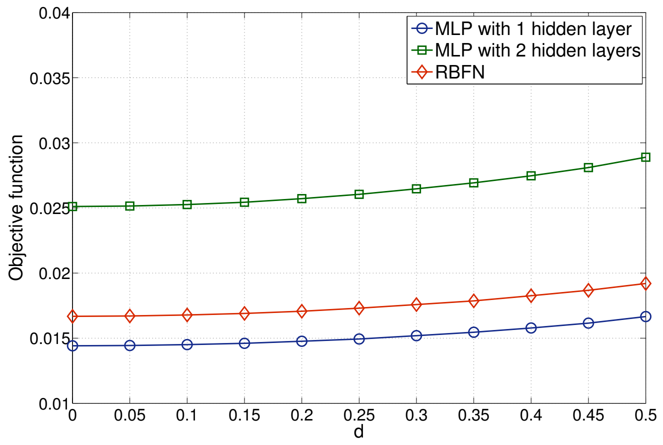
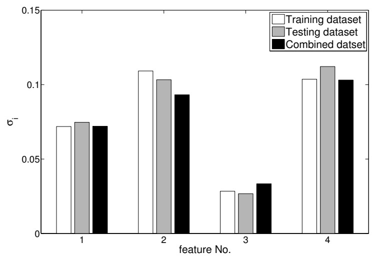
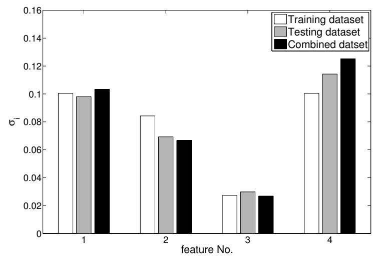


Figure 4.42. Effect of standard deviation perturbations on objective function.

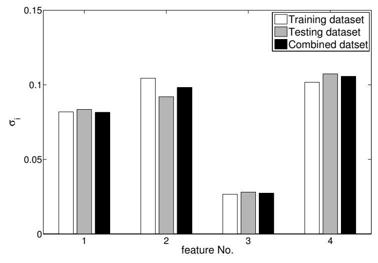
- Task2: *Results For Training, Testing and Combined Data*



(a) One hidden layer MLP



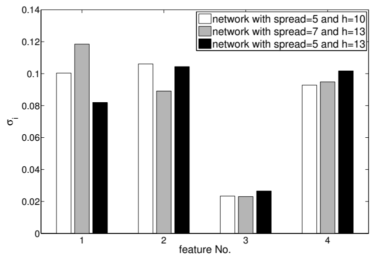
(b) Two hidden layer MLP



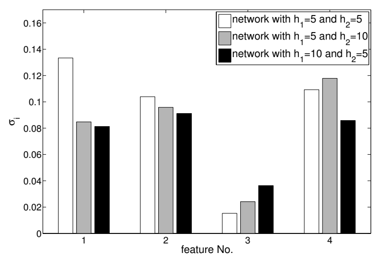
(c) RBFN

Figure 4.43. Comparison of optimal standard deviations for training, testing and combined data sets.

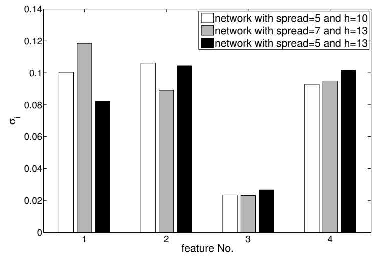
- Task3: *Results For Different Network Paramaters*



(a) One hidden layer MLP



(b) Two hidden layer MLP



(c) RBFN

Figure 4.44. Comparison of optimal standard deviations for different network parameters and for training data.

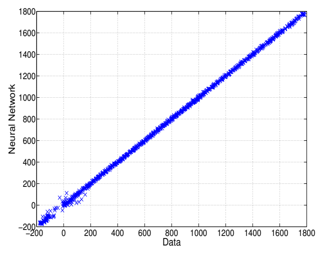
E. Engine Data

Engine data set is obtained from the operation of an engine. There are two outputs for this data, namely torque and emission levels although we consider just the first output. The inputs are engine speed and fueling levels.

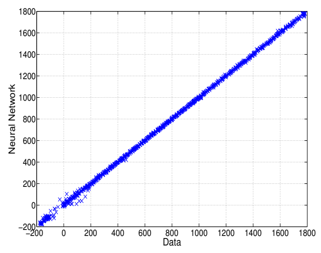
- *Construction of Numeric NNs*

	1 hidden layer MLP	2 hidden layers MLP	RBFN
Input neurons No.	2	2	2
First hidden neurons No.	15	15	75
Second hidden neurons No.	-	5	-
Spread	-	-	0.3
R^2 on training data	0.999432	0.999563	0.999621
R^2 on testing data	0.999462	0.999531	0.999583

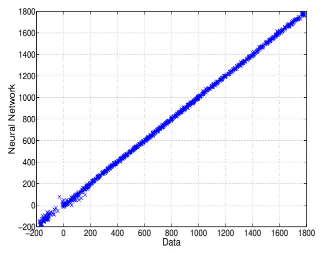
Table 4.11. Performance and the parameters of constructed networks for Engine data



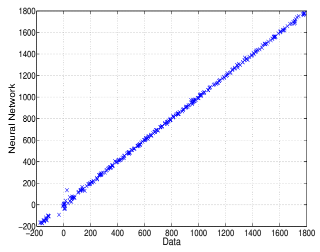
(a) One hidden layer MLP



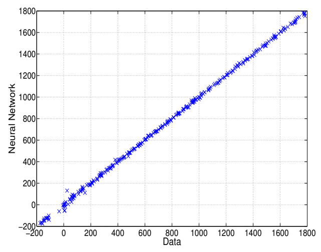
(b) Two hidden layer MLP



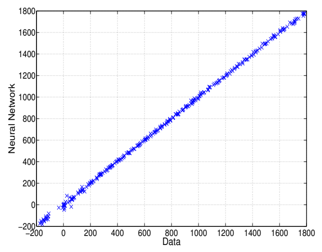
(c) RBFN



(d) One hidden layer MLP



(e) Two hidden layer MLP



(f) RBFN

Figure 4.45. Scatter plots of the output of the network versus data: (a), (b), (c) training data (d), (e), (f) testing data

- *Comparison of Optimal Variances For Different Architectures*

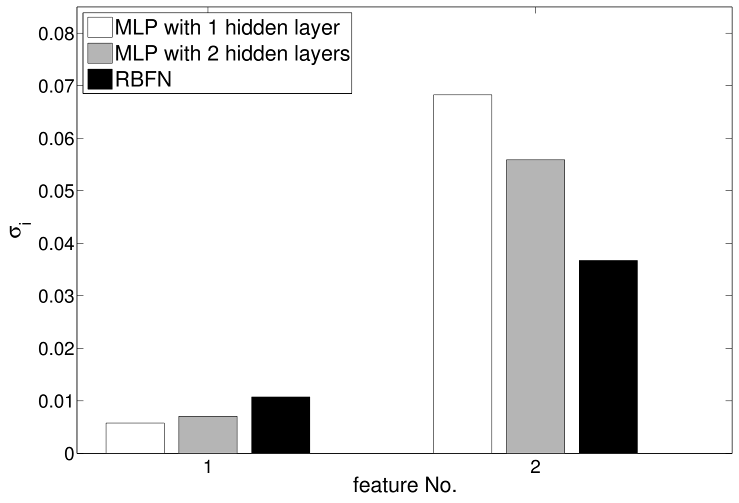


Figure 4.46. Comparison of the optimal standard deviations for different network architectures and for training data.

Table 4.11 shows that the networks are trained very well as their R^2 values on both training and testing data are very close to 1. Therefore, in Fig. 4.46, we observe that three networks lead to almost same result which is larger standard deviations is given to feature 2 compared to feature 1.

- Task1: *Results for Perturbation Analysis*

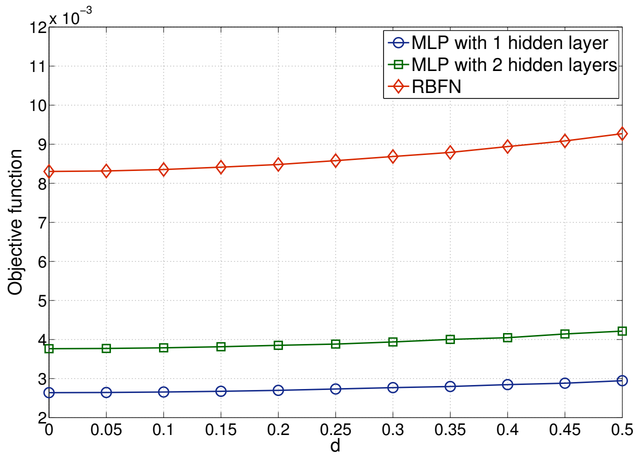
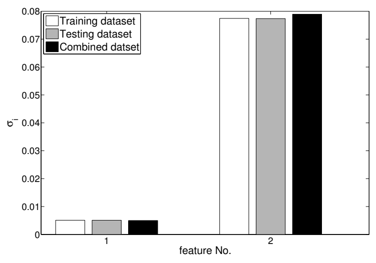
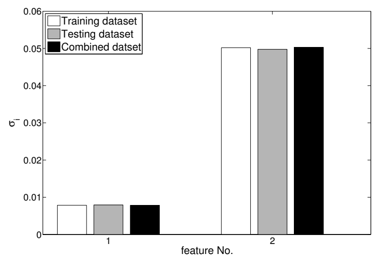


Figure 4.47. Effect of standard deviation perturbations on objective function.

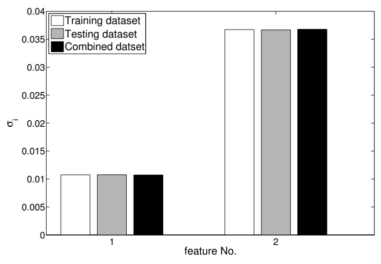
- Task2: *Results For Training, Testing and Combined Data*



(a) One hidden layer MLP



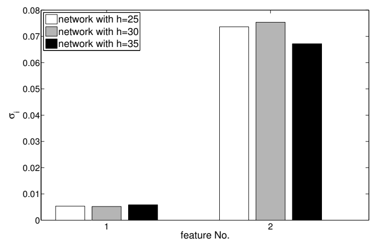
(b) Two hidden layer MLP



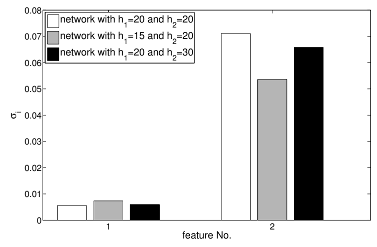
(c) RBFN

Figure 4.48. Comparison of optimal standard deviations for training, testing and combined data sets.

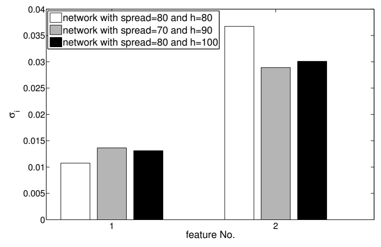
- Task3: *Results For Different Network Paramaters*



(a) One hidden layer MLP



(b) Two hidden layer MLP



(c) RBFN

Figure 4.49. Comparison of optimal standard deviations for different network parameters and for training data.

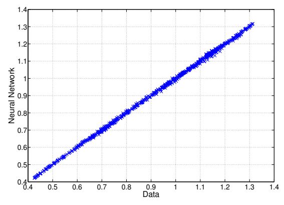
F. Mackey-Glass Time Series

Mackey-Glass time series is a well known time series prediction problem. The time series forecasting is defined as the process of predicting one or more future observations of time series given the subset of past observations. This process is done to predict the entire of time series. So , a data can be input in one time and it can be output in another time. In other words, there is no fixed discrimination between input and output. On the other hand, the machine learning algorithms use fixed-length feature vectors corresponds to a specific output variable. Therefore, the most essential and important preprocessing step in time series forecasting is converting the time series into a suitable form for machine learning algorithms. Here, we have used *time delay embedding* to convert time series to input-output pairs [23]. The amount of delay and dimension for this time series is calculated 13 and 3, respectively. So, the number of input neurons for this network is equal to 3.

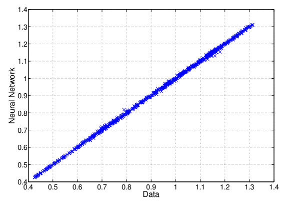
- *Construction of Numeric NNs*

	1 hidden layer MLP	2 hidden layers MLP	RBFN
Input neurons No.	3	3	3
First hidden neurons No.	30	20	100
Second hidden neurons No.	-	30	-
Spread	-	-	80
R^2 on training data	0.999553	0.999581	0.999586
R^2 on testing data	0.999528	0.999467	0.999577

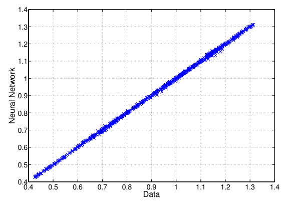
Table 4.12. Performance and parameters of constructed networks for Mackey-Glass data



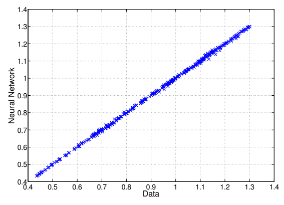
(a) One hidden layer MLP



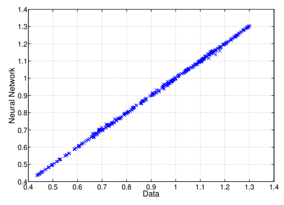
(b) Two hidden layer MLP



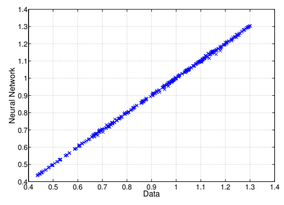
(c) RBFN



(d) One hidden layer MLP



(e) Two hidden layer MLP



(f) RBFN

Figure 4.50. Scatter plots of the output of the network versus data: (a), (b), (c) training data (d), (e), (f) testing data

- *Comparison of Optimal Variances For Different Architectures*

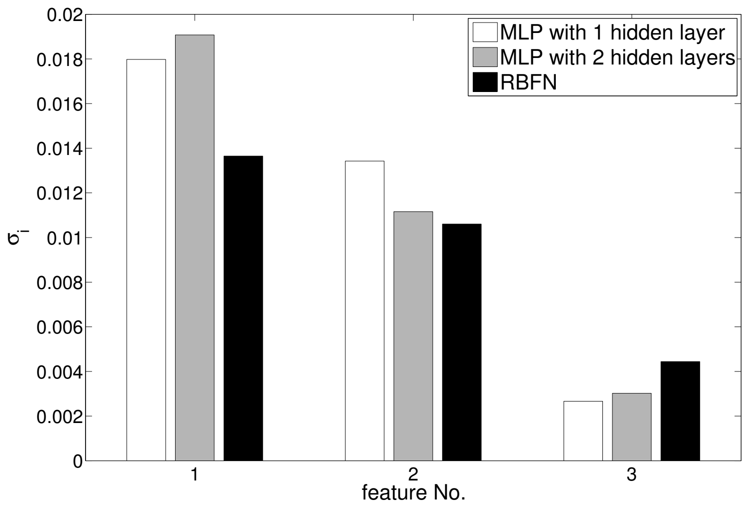


Figure 4.51. Comparison of the optimal standard deviations for different network architectures and for training data.

Here, again, we see that the networks are trained very well and therefore, the resulting standard deviations for different GNNs are very similar.

- Task1: *Results for Perturbation Analysis*

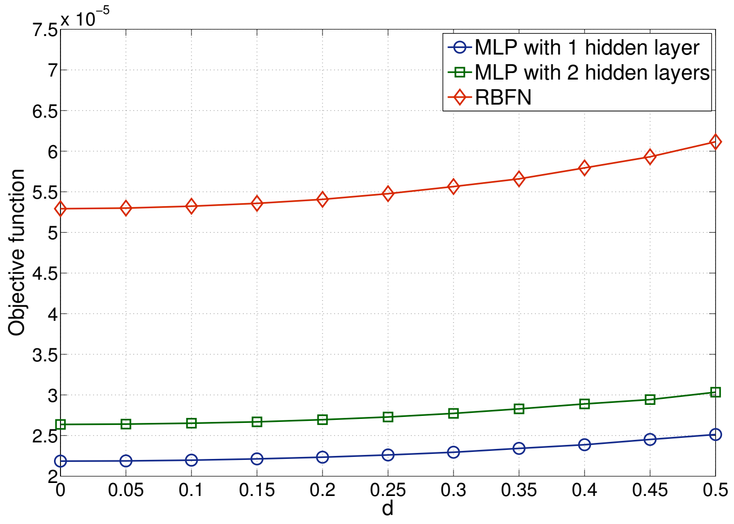
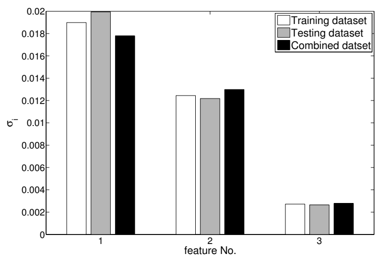
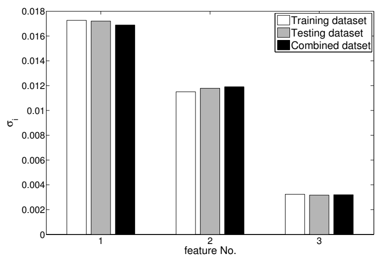


Figure 4.52. Effect of standard deviation perturbations on objective function.

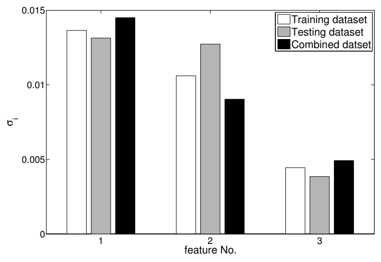
- Task2: *Results For Training, Testing and Combined Data*



(a) One hidden layer MLP



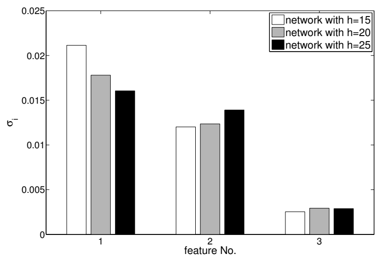
(b) Two hidden layer MLP



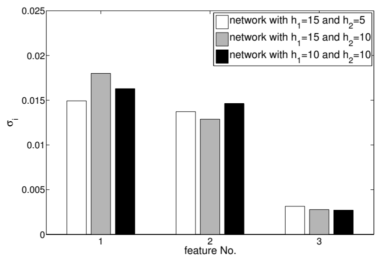
(c) RBFN

Figure 4.53. Comparison of optimal standard deviations for training, testing and combined data sets.

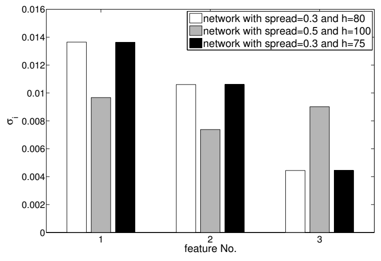
- Task3: *Results For Different Network Paramaters*



(a) One hidden layer MLP



(b) Two hidden layer MLP



(c) RBFN

Figure 4.54. Comparison of the optimal standard deviations for different network parameters and for training data.

Discussion of Experimental Results on Real Data Sets

Based on the experimental results obtained for real data sets, we observe that:

- As the values of R^2 get closer to 1 for different NN architectures, their optimal standard deviation allocations become more similar to each other. Moreover, the standard deviations of GNNs with higher R^2 values are less sensitive to both the changes of the network parameters (like the number of neurons in hidden layers) and the input data (whether training, testing or a mixture of them) used. The NNs constructed for the Engine data and Mackey-glass time series are examples with R^2 values close to 1.

- Despite the differences in the standard deviations allocated when different architectures are used, for all data sets some features receive larger standard deviations regardless of the network architecture. Similarly, for all network structures some features are allocated small amounts of standard deviation. This implies that in general, the optimal granularity allocation procedure we proposed leads to consistent results.

- The standard deviations resulting from MLP networks with one and two hidden layers are closer to each other compared to the standard deviations obtained for RBFN. In general, when RBFN structure is used, the granulation process leads to a smoother (more uniform) allocation of standard deviations over the input features. This point is more clear for Auto MPG and Boston data sets.

Chapter 5

Conclusion

5.1 Summary of Contributions

In this thesis, we investigated the process of designing GNN on the basis of a numeric NN whose inputs were augmented using random variables. The design process we proposed aimed to optimally allocate a given level of information granularity to the input features of a GNN such that the specificity of its output maximizes. In our work, we followed a nonidentical but symmetric granularity allocation paradigm which allowed different input random variables to be assigned different granularity levels while their distributions were symmetric around their expected values. We used the entropy of the input random variables as a measure to characterize the granularity level associated with input features, and we formulated our design problem as an optimization problem with the standard deviations of the input distributions as the parameters to be optimized.

Based on the solution we derived for the optimization problem, we can summarize our proposed design process in three steps as follows. In the first step, we train a numeric NN with a predetermined architecture to fit a given data set. In the second step, we compute the variability coefficients $\{a_i\}_{i=1}^{n}$ given in (3.6) corresponding to the constructed NN. Each of these coefficients basically characterizes the average variability level of the network output to the changes in an input feature. As the last step, we determine the optimal standard deviation of each input distribution by substitution of the variability coefficients obtained in the second step in the analytical formula (3.31).

The second part of this thesis was devoted to experiments on constructing GNNs for synthetic and real data sets using our proposed design process. In particular, we applied our proposed method to five synthetic data sets and six real data sets. Our experimental explorations provided us with a number of interesting observations. First, we observed that smaller standard deviation levels were allocated to the random variables corresponding to the input dimensions that the function exhibits faster changes along them. This observation is consistent with our intuition to the problem, because to minimize the changes to the GNN output, one has to limit the variations of those input features that cause fast changes of the output. As the second point, we observed that the optimal standard deviations for different NNs architectures were similar to each other when the performances of the trained networks were very high. Third, we observed that when the networks were trained very well to fit the data, our proposed design was very robust to changes of the network architectures, networks parameters, and to the input data (training or testing) used to derive the standard deviations.

5.2 Future Works

The research project we accomplished in this thesis can be followed by two closely related projects as will be discussed below:

1. In this thesis, we used the specificity criterion as the measure of goodness of a GNN and using this measure we formulated our design problem. In this work, we did not comment on the performance of the proposed design based on the coverage criterion.

2. In this work, we constructed GNN by granulating the inputs of a numeric NN using random variables, however, we kept the network parameters numeric. Another way to construct GNNs is to granulate the weights of the numeric NNs while keeping the inputs numeric. Investigation of GNNs with granular network parameters, where granulation is maid using random variables, is a interesting potential research direction. Such an analysis can be very useful to understand the effect of deviation of the network parameters from their expected values as a result of noise or disturbances.

Bibliography

[1] S. Haykin, *Neural Networks and Learning Machines*, 3rd ed. Upper Saddle River, New Jersey: Pearson Education, 2009.

[2] M. H. Hassoun, *Fundamentals of Artificial Neural Networks*. MIT Press, 2003.

[3] M. Song and W. Pedrycz, "Granular neural networks:concepts and development schemes," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 24, no. 4, pp. 542 –553, April 2013.

[4] W. Pedrycz and G. Vukovich, "Granular neural networks," vol. 36, no. 1-4, pp. 205 –224, February 2001.

[5] W. Pedrycz, "Allocation of information granularity in optimization and decision-making models: Towards building the foundations of granular computing," *European Journal of Operational Research*, vol. 232, no. 1, pp. 137–145, January 2014.

[6] ——, *Granular Computing: Analysis and Design of Intelligent Systems*. CRC Press, 2013.

[7] G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Math. Control Signals Systems*, vol. 2, February 1989.

[8] S. Kenji, *Artificial Neural Networks - Architectures and Applications*. InTech, 2013.

[9] J. Park and I. W. Sandberg, "Universal approximation using radial-basis-function networks," *Neural Computation*, 1991.

[10] Z. Karakehayov, *Data Acquisition Applications*. InTech, 2012.

[11] Q. J. Zhang and K. C. Gupta, *Neural Networks for RF and Microwave Design*. Artech House, 2007.

[12] L. A. Zadeh, "Fuzzy sets and information granularity," *Advances in Fuzzy Set Theory and Applications*, 1979.

[13] A. Skowron and J. Stepaniuk, "Towards discovery of information granules," *Proceedings of Third European Conference on Principles and Practice of Knowledge Discovery in Databases*, vol. 1704, no. 1, pp. 542–547, September 1999.

[14] H. Ishibuchi and H. Tanaka, "An architecture of neural networks with interval weights and its application to fuzzy regression analysis," *Fuzzy Sets Syst.*, vol. 57, no. 1, pp. 27–39, July 1993.

[15] K. K. Ishibuchi, H. and H. Tanaka, "A learning algorithm of fuzzy neural networks with triangular fuzzy weights," *Fuzzy Sets Syst.*, vol. 71, no. 3, pp. 277–293, May 1995.

[16] A. Papoulis and S. U. Pillai, *Probability, Random Variables and Stochastic Processes*, 4th ed. McGraw-Hill, 2002.

[17] B. M. Ayyub and R. H. McCuen, *Probability, Statistics, and Reliability for Engineers and Scientists*, 3rd ed. CRC Press, 2011.

[18] S. Boyd and L. Vandenberghe, *Convex Optimization*, 7th ed. Cambridge University Press, 2009.

[19] J. A. Thomas and T. Cover, *Elements of Information Theory*, 2nd ed. Wiley, 2012.

[20] M. Grasselli and D. Pelinovsky, *Numerical Mathematics*. Jones and Bartlett Publishers, 2008.

[21] K. M. Abadir and R. M. Jan, *Matrix Algebra*. Cambridge University Press, 2005.

[22] "Uci machine learning repository," http://archive.ics.uci.edu/ml/.

[23] H. K. R. Hegger1 and T. Schreiber, *Practical Implementation of Nonlinear Time Series Methods: The TISEAN package*. American Institute of Physics, 1999.

Appendix A: Derivation of the Solution of Optimization Problem (3.9)

In this appendix, we derive the solution to the optimization problem (3.9); i.e.,

$$\begin{aligned} &\underset{\boldsymbol{\sigma^2}=(\sigma_1^2,\ldots,\sigma_n^2)}{\text{minimize}} \quad && I(\boldsymbol{\sigma^2}) = \sum_{i=1}^{n} a_i \sigma_i^2 \\ &\text{subject to} && \sum_{i=1}^{n} \sigma_i^2 = \varepsilon,\ \sigma_i^2 \geq 0\ \forall i. \end{aligned} \tag{A.1}$$

To begin, we sort a_is in increasing order

$$a^{(1)} = a^{(2)} = a^{(k)} < a^{(k+1)} \leq a^{(k+2)} \leq \ldots \leq a^{(n)} \tag{A.2}$$

where the superscripts represent the rank of the coefficients a_i. Therefore,

$$\sum_{i=1}^{n} a_i \sigma_i^2 \geq a^{(1)} \sum_{i=1}^{n} \sigma_i^2 = a_{\min}\varepsilon \tag{A.3}$$

where $a_{\min} = a^{(1)}$. Denoting the variance corresponding to the coefficient $a^{(i)}$ with $\sigma^{2^{(i)}}$, we have (A.3) with equability if and only if

$$\sigma^{2^{(1)}} + \sigma^{2^{(2)}} \ldots + \sigma^{2^{(k)}} = \varepsilon \tag{A.4a}$$

$$\sigma^{2^{(k+1)}} = \sigma^{2^{(k+2)}} = \ldots = \sigma^{2^{(n)}} = 0. \tag{A.4b}$$

So, the minimum value of the cost function is $a_{\min}\varepsilon$. Also, the optimal values of σ_i^2 are 0 if $a_i > a_{\min}$ and for the rest of σ_is, in which $a_i = a_{\min}$, all positive combinations that add up to ε are optimal.