World Academy of Science, Engineering and Technology
International Journal of Computer and Information Engineering
Vol:1, No:12, 2007

Solving Partially Monotone Problems with Neural Networks

Marina Velikova, Hennie Daniels, and Ad Feelders

Abstract**—**In many applications, it is a priori known that the target function should satisfy certain constraints imposed by, for example, economic theory or a human-decision maker. Here we consider partially monotone problems, where the target variable depends monotonically on some of the predictor variables but not all. We propose an approach to build partially monotone models based on the convolution of monotone neural networks and kernel functions. The results from simulations and a real case study on house pricing show that our approach has significantly better performance than partially monotone linear models. Furthermore, the incorporation of partial monotonicity constraints not only leads to models that are in accordance with the decision maker's expertise, but also reduces considerably the model variance in comparison to standard neural networks with weight decay.

Keywords**—**Mixture models, monotone neural networks, partially monotone models, partially monotone problems.

I. Introduction

IN many applications, it is a priori known that the target function should satisfy certain constraints imposed by, for example, economic theory or a human-decision maker. In many cases, however, the final model obtained by data mining techniques alone does not meet these constraints. It is required that the algorithms have to be modified (enhanced) to obey the constraints in a strict fashion.

One type of constraint, which is common in many decision problems, is the monotonicity constraint stating that the greater an input is, the greater the output must be, all other inputs being equal. There is a wide range of applications where monotonicity properties hold. Well-known examples include credit loan approval, the dependence of labor wages as a function of age and education, investment decisions, hedonic price models, selection and evaluation tasks ([4],[6]).

Several data mining techniques have been developed, which incorporate monotonicity constraints such as neural networks ([3],[5],[11],[12]), rational cubic interpolation of one-dimensional functions ([10]), decision trees ([1],[8]), etc. However, the main assumption for the implementation of most of these methods is that the function (output) being estimated should be monotone in all inputs (so-called total monotonicity). This in practice, of course, is not always the case.

In this paper we consider partially monotone regression problems, where we assume that the dependent variable depends monotonically on some of the independent variables but not all. For example, common sense suggests that the house price has monotone increasing dependence on the number of rooms and the total house area, whereas for the number of floors this dependence does not necessarily hold. Such prior knowledge about monotone relationships can be incorporated as constraints in data mining algorithms in order to improve the accuracy and interpretability of the models derived as well as to reduce their variance on new data.

The paper is organized as follows. In the next section we introduce notations and definitions related to monotonicity, which are needed for the follow-up discussion. The main contribution of this paper is the approach for partial monotonicity presented in Sect.IVA. The approach is based on the convolution of a special type of monotone neural networks, introduced in Sect.III, and kernel functions. In Sect.IVB we present the design and the results from simulation studies carried out to test the performance of the proposed approach for partial monotonicity. Sect.IVC demonstrates the application of the approach on a real case study of house pricing. Concluding remarks are given in Sect.V.

II. Notations and Definitions

Let **x** denote the vector of independent variables, which takes values in a *k*-dimensional input space **X**, and *y* denotes the dependent variable that takes values in a one-dimensional space **Y**. We assume that a data set $D = (\mathbf{x}, y)$ of p points in **X***__Y__ is given.

For totally monotone problems, we assume that D is generated by a process with the following properties

$$y = f(\mathbf{x}) + \varepsilon, \tag{1}$$

where f is a monotone function, and ε is a random variable with zero mean and constant variance σ_ε^2. Monotonicity of f on **x** is defined on all independent variables by

Manuscript received 05.12.2005

M.Velikova is with the Center for Economic Research, Tilburg University, The Netherlands (phone: +31 13 466 8721, fax: +31 13 466 3069, e-mail: M.Velikova@uvt.nl)

H.Daniels is with the Center for Economic Research, Tilburg University, The Netherlands and ERIM Institute of Advanced Management Studies, Erasmus University Rotterdam, Rotterdam, The Netherlands (e-mail: daniels@uvt.nl)

A.Feelders is with the Department of Information and Computing Sciences, Utrecht University, The Netherlands (e-mail: ad@cs.uu.nl)

$$\mathbf{x}^1 \geq \mathbf{x}^2 \Rightarrow f(\mathbf{x}^1) \geq f(\mathbf{x}^2), \quad (2)$$

where $\mathbf{x}^1 \geq \mathbf{x}^2$ is a partial ordering on $\mathbf{X}$ defined by $x_i^1 \geq x_i^2$, for $i = 1, 2, \ldots, k$. The pair $(\mathbf{x}^1, \mathbf{x}^2)$ is called comparable and if the relationship defined in (2) holds, it is also a monotone pair.

Note that even though f is monotone, the data generated by (1) is not necessarily monotone due to the random effect of ε.

For partially monotone problems, we have $\mathbf{x}^m = \{x_i \in \mathbf{X} \mid i = 1,\ldots,k^m\}$, $\mathbf{x}^n = \{x_j \in \mathbf{X} \mid j = k^m + 1,\ldots,k\}$ for $1 \leq k^m < k$. Furthermore, a data set $D = (\mathbf{x}^m, \mathbf{x}^n, y)$ of p points is generated by

$$y = f(\mathbf{x}^m, \mathbf{x}^n) + \varepsilon, \quad (3)$$

where f is a monotone function in $\mathbf{x}^m$ and ε is a random variable with zero mean and constant variance σ_ε^2.

Our objective is to find a smooth approximation $\hat{f}$ of $f(\mathbf{x}^m,\mathbf{x}^n)$ such that $\hat{f}$ is monotone in $\mathbf{x}^m$, i.e., $\hat{f}$ is a *partially monotone estimator*.

A simple solution is to consider the class of partially monotone linear functions of the form

$$\hat{f}(\mathbf{x}^m, \mathbf{x}^n) = a_0 + \sum_{i=1}^{k^m} a_i x_i^m + \sum_{j=k^m+1}^{k} a_j x_j^n, \text{ subject to}$$
$$a_i \geq 0,\ i = 1,2, \ldots, k^m. \quad (4)$$

We expect that the estimate in (4) would produce good fit for linear functions; it would give, however, poor approximations for complex functions (see Sect.IVB). Therefore, it is necessary to consider more flexible models for estimating any continuous partially monotone function.

In this paper, we look at mixture models of the form

$$\hat{f}(\mathbf{x}^m, \mathbf{x}^n) = \sum_{c=1}^{C} \hat{f}_c(\mathbf{x}^m) \cdot \varphi_c(\mathbf{x}^n), \quad (5)$$

where C is a number of clusters (subsets of D), $\hat{f}(\mathbf{x}^m)$ is the output of a monotone (Sill) network, presented in the next section, built on $\mathbf{x}^m$, and $\varphi(\mathbf{x}^n)$ is a weight function (kernel) based on $\mathbf{x}^n$.

III. Monotone Networks

A. General Notation

A standard feedforward neural network with multi-layer architecture is represented by:

- Input layer containing k+1 units - one unit for each input variable x_k, and one bias unit set to a constant value of 1.
- Hidden layer(s) consisting of a set of H units, which are connected to the bias and input nodes.
- Output layer with one or more units that produce the output of the network. Here we consider neural networks with only one output node.

All the connections between the layers are weighted. Let w_{ij} denote the weight on the connection between input j and hidden unit i and v_i is the weight on the connection between hidden unit i and the output. Then, given $\mathbf{x}$ the functional form of the output $O_\mathbf{x}$ corresponding to a network with one hidden layer is represented by

$$O_\mathbf{x} = \sum_{i=1}^{H} v_i \lambda\left(\sum_{j=1}^{k} w_{ij} x_j + \theta_i\right) + \theta_0, \quad (6)$$

where θ_i, θ_0 are the bias terms to the hidden and output nodes, and λ is the activation function, which is usually taken to be the sigmoid function $\lambda(u) = 1/(1+e^{-u})$. The class of networks in (6) can approximate any continuous function of k variables on any compact subset of $\Re^k$ ([2]).

Since the focus of this paper is on monotone problems, we are interested in neural networks for which the output is guaranteed to be monotone. In [11] it is proved that a special class of neural networks has the capacity to approximate uniformly to an arbitrary degree of accuracy any continuous monotone function. This type of monotone networks is used in our study and we refer to it as to *Sill networks* in the remainder of the paper. For clarity, in the next section we briefly describe the architecture of Sill networks using similar definitions and notations as in [11].

B. Sill Networks

The network considered here is based on the three-layer (two hidden-layer) architecture introduced by Sill in [11]. The input layer is connected to the first hidden layer consisting of a set of linear units, which are combined into several groups, (the number of units in each group is not necessarily the same). Corresponding to each group is a second hidden-layer unit, which computes the maximum over all first-layer units within the group. The final output unit computes the minimum over all groups.

In formal notation, a Sill network can be represented as follows. Let R denote the number of nodes in the second hidden layer, that is the number of groups in the first hidden layer, with outputs $g_1, g_2, \ldots, g_R$, and h_r denotes the number of hyperplanes within group r, $r = 1, 2, \ldots, R$. The parameters (weights) of the hyperplanes in r are denoted by the vectors $\mathbf{w}_{(r,1)}, \mathbf{w}_{(r,2)}, \ldots, \mathbf{w}_{(r,h_r)}$. Then the output at group r is defined by

$$g_r(\mathbf{x}) = \max_j(\mathbf{w}_{(r,j)} \cdot \mathbf{x} + \theta_{(r,j)}), \quad 1 \leq j \leq h_r, \quad (7)$$

where θ is the bias term.

The final output $O_\mathbf{x}$ of the network for input $\mathbf{x}$ is given by

$$O_\mathbf{x} = \min_r g_r(\mathbf{x}). \quad (8)$$

From (8), it follows that one group and one hyperplane within this group determine the output of the network for each input vector. Such group and hyperplane are called *active*. In case of ties in the group or network outputs (though it is unlikely as the outputs are continuous), the choice of active hyperplane or group is made arbitrarily.

To guarantee that the network output is monotone, all weights from an input to the first hidden layer are constrained to be non-negative (non-positive), if increasing (decreasing) monotonicity is desired for that input. Here, the parameters in (7) are enforced to be non-negative by taking an appropriate transformation such as $w = z^2$, where z is a free parameter.

The Sill network thus described has several advantages. First, computation of the output is simple and fast due to the limited number of linear unit calculations and simple comparison operators performed. In addition, at each iteration of the training process the weights of a single linear unit (the active one) are only modified, which speeds up network's learning. Second, by constraining the coefficients of the linear units it is easy to incorporate domain knowledge in the network. Therefore, monotonicity can be easily imposed by restricting the coefficients to be positive or negative. Finally, for a particular input the output from Sill networks is easy to understand and interpret by the end user as the parameters of the linear units directly reflect the relationships in the data.

In [11] it is shown in a case study on bond rating that the three-layer monotone networks perform better than a linear model and standard neural networks for problems where monotonicity is present in the domain.

IV. Approach for Partial Monotonicity

A. Description

As stated earlier, our goal is to find a smooth estimation $\hat{f}$ of $f(\mathbf{x}^m, \mathbf{x}^n)$ given in (2) such that $\hat{f}$ is monotone in $\mathbf{x}^m$.

To find a smooth estimator of unknown regression function, a common approach is to use a non-parametric method such as kernel regression. An advantage of the method is its flexibility, i.e., it allows estimating functions of greater complexity. Besides being smooth, the estimator we look for should be also monotone in $\mathbf{x}^m$. However, the implementation of monotonicity constraints in a kernel estimator is not straightforward, especially for multidimensional functions.

Then an intuitive solution to guarantee that the estimator $\hat{f}$ is smooth and partially monotone is to construct monotone approximations of f with respect to $\mathbf{x}^m$ while $\mathbf{x}^n$ is fixed and then to smooth out the resulting estimates by using kernels based on $\mathbf{x}^n$.

This approach is used here for building a class of partially monotone functions in the form of (4). The idea is to convolute Sill networks built on $\mathbf{x}^m$ and suitable kernel functions based on $\mathbf{x}^n$.

In the first step of the proposed approach, the input space with respect to $\mathbf{x}^n$ is partitioned into a number of disjoint subsets (clusters) by using a hierarchical clustering algorithm. The appropriate number of subsets is determined automatically by cutting off the hierarchy obtained from the clustering procedure at several levels (from 2 to 10). Then for each of the partitioning outcomes we compute the silhouette value as a measure for the goodness of clustering (ranged from –1 for bad to +1 for good) ([9]). The outcome with the maximal silhouette value determines the final number of clusters. An additional improvement in the clustering procedure is adding weights to the variables in the standard Euclidean distance measure we use. In this way, we take into account the significance of each variable on the dissimilarities between the points and the formation of the clusters, respectively. The vector of weights $\boldsymbol{\alpha}$ are determined a priori by taking the absolute value of the respective coefficients for each variable obtained from the linear model fitted to the whole data set and normalizing them to sum up to 1.

As a result of this partitioning of the original data D, we obtain a number C of subsets D_1, D_2, …, D_C, where the number of points in the subsets is not necessarily the same. There is no restriction on the minimal number of points in a subset.

For each D_c, $c = 1, 2, \ldots, C$, which contains more than one point, the value of the non-monotone variable is fixed to the cluster mean $\mathbf{x}_c^n$. Furthermore, an estimate $\hat{f}_c(\mathbf{x}^m)$ of f is obtained based only on the values of the monotone variable $\mathbf{x}^m$ for the points belonging to D_c. This is done by using Sill networks, which guarantees that the function approximation is monotone within each subset.

If a cluster with only one point is created (i.e., an outlier in respect to the values of the non-monotone variables is detected) then the cluster mean takes the values of the non-monotone variable for that point and the function approximation is just the label of the point. The reasoning for not ignoring the one-point clusters is as follows. Suppose we want to predict the label $y_{\mathbf{z}}$ of a new point $\mathbf{z}$, which is closer to a one-point cluster than to the others (meaning that the values of the non-monotone variables are similar). Now if $\mathbf{z}$ has also values of the monotone variable that are similar to those of the point in the cluster, then the predicted label is expected to be also close to the label of the point. However, if the values of the monotone variable are dissimilar between the points then $\mathbf{z}$ can be considered as a point without analog in the data (i.e., outlier) but its label can be still predicted by using the function estimations from all the clusters as described below.

In the next step of the approach, for all D_c, $c = 1, 2, \ldots, C$, we define

$$\psi_c = \left\| \boldsymbol{\alpha}\left(\mathbf{x}^n - \mathbf{x}_c^n\right) \right\|^{-1},$$

where $\|\cdot\|$ is the Euclidean distance norm weighted by $\boldsymbol{\alpha}$, $\mathbf{x}^n \in D$ and $\mathbf{x}_c^n$ is the mean (centroid) value of cluster c. By definition $\psi \geq 0$ and it determines the distance of point $\mathbf{x}$ to the mean of cluster c. By normalizing ψ with

TABLE I
FACTORS AND THEIR VALUES USED IN THE SIMULATION EXPERIMENTS

Approach for partial monotonicity					Neural networks with weight decay				
Factors		Levels (values)			Factors		Levels (values)		
		1	2	3			1	2	3
1	# points in data	50	150	250	1	# points in data	50	150	250
2	Noise level (σ_ε^2)	0.01	0.5	2	2	Noise level (σ_ε^2)	0.01	0.5	2
3	# groups in Sill net	2	3	4	3	# hidden neurons	3	9	15
4	# planes in Sill net	2	3	4	4	Weight decay	0.000001	0.00001	0.0001

$$\varphi_c\left(\mathbf{x}^n\right)=\frac{\psi_c\left(\mathbf{x}^n\right)}{\sum_{c=1}^{C}\psi_c\left(\mathbf{x}^n\right)},$$

we obtain a function $\varphi \geq 0$, for which

$$\sum_{c=1}^{C}\varphi_c\left(\mathbf{x}^n\right)=1.$$

Hence, φ can be considered as a weighted function or a kernel in Nadaraya-Watson form ([7],[13]).

Finally, we convolute φ_c with the corresponding function approximations $\hat{f}_c\left(\mathbf{x}^m\right)$ for each cluster c by

$$\hat{f}\left(\mathbf{x}^m,\mathbf{x}^n\right)=\sum_{c=1}^{C}\hat{f}_c\left(\mathbf{x}^m\right)\cdot\varphi_c\left(\mathbf{x}^n\right),$$

to obtain the final estimate of *f*.

B. Simulation Studies

In this section, we present the results from the simulation studies designed to test the effectiveness of the approach for partial monotonicity. We generate artificial data based on two independent variables and a dependent variable computed by applying a function that is monotone in one of them and non-monotone in the other.

First, two vectors of p values, x^m and x^n, are generated independently from each other. The values of vector x^m are drawn from the uniform distribution on [0,1]. The vector x^n is composition of two sub-vectors each of size $p/2$ points, which are drawn from two normal (Gaussian) distributions: $N(1,0.5)$ and $N(5,0.5)$. Then, we compute the values of a third vector y by applying a monotone function on x^m and a non-monotone function on x^n plus a random perturbation $\varepsilon \sim N(0,\sigma_\varepsilon^2)$:

$$y = 3 + sin\left(\frac{\pi}{2}x^m\right)\cdot\left(2+sin\left(2\pi x^n\right)\right)+\varepsilon .$$

Hence, we can consider x^m and x^n as the independent variables and y as the dependent variable in a data set $D=(x^m, x^n, y)$ of p points.

Now based on D thus generated we want to build a model for predicting y. Given that the constructing function is known to be partially monotone, we apply the approach for partial monotonicity as an appropriate method for estimation. The clustering algorithm used in the approach finds two clusters in the data corresponding to the two Gaussians for x^n. Kernels and Sill networks' outputs are computed for each cluster and finally they are convoluted to obtain the final estimation of y.

To obtain sound assessment for the performance of our approach we use standard neural networks with weight decay and partially monotone linear models in the form of (3) as benchmark methods for comparison. The standard neural networks consist of an input layer, one hidden layer and one continuous output. In the hidden layer the activation function is sigmoid, whereas in the output it is linear. In addition, the weight decay is used as a regularization method to prevent the networks from overfitting. This is done by adding to the mean-squared error the term $\delta\sum_{ij}w_{ij}^2$ to penalize large weights, where δ is the weight decay parameter. The comparison between our approach and the benchmark methods is based on the mean-squared error (MSE) as a measure for the quality of estimation.

To obtain more complete performance analysis, the experiments with the approach for partial monotonicity and neural networks are conducted by using several factors with different values for comparison (Table I).

All possible combinations of the four three-value factors require the experiment with each method to run 81 times (3^4). In order to reduce the effort and experimental cost in the simulations, we use the so-called *fractional factorial design* ([14]), where only a certain number of combinations of factor values are taken to carry out the experiments. This is done in a systematic way by combining each value of each factor only once with each level of the other factors. In our case the fractional design requires 9 runs with each method.

For each run we generate a collection of 50 data samples following the data generating procedure described above. For computational convenience the values of the independent variables in each set are fixed, whereas the value of the dependent variable varies across different data samples. The approach for partial monotonicity, neural networks with weight decay and partially monotone linear models are applied on the same collections of data samples.

As a result from the experiments, for each method we obtain 9 estimations of MSE. These results are used then to compute the expected value $\mathbf{E}\{\mathrm{MSE}_{ijkl}\}$ for all possible combinations of factor values (i, j, k, l), where i, j, k and l range from 1 to 3. As described in [14], this is done by fitting the exponential model

$$\mathbf{E}\{\mathrm{MSE}_{ijkl}\}=exp\left(\mu+\left(\mu_i^1-\mu\right)+\left(\mu_j^2-\mu\right)+\left(\mu_k^3-\mu\right)+\left(\mu_l^4-\mu\right)\right), \quad (9)$$

where μ is the mean computed over all 9 estimations,

$\mu_i^1, \mu_j^2, \mu_k^3$ and μ_l^4 are the means for each factor value; the fit of an exponential function guarantees that the estimated $\mathbf{E}\{\text{MSE}_{ijkl}\}$ is positive. For example, for the combination of factor values (50 data points, σ_ε^2=0.5, 4 groups, 3 planes), i.e., (i=1,j=2,k=3,l=2) the approach for partial monotonicity has not been run. Then to compute $\mathbf{E}\{\text{MSE}_{1232}\}$, we use the respective means $\mu_1^1, \mu_2^2, \mu_3^3$, and μ_2^4 in (9).

Finally, as there are two factors (number of data points and noise level), which are the same in the experiments, we want to compare the performance of the methods for all combinations of values (i, j) (in total 9) of these two factors. For this purpose, within each (i, j) out of all value combinations (in total 9) we take the minimum estimation and variance of MSE over the other two factors. The results presented in Table VI in the Appendix show that the proposed approach for partial monotonicity is superior to the partially monotone linear models in providing a significantly better fit to the data, and superior to the standard neural networks in reducing considerably the model variance. The last finding is clear indication that the models obtained from our approach are more robust upon repeated sampling.

C. Real Case Study

In this section we present the results obtained from the application of the approach for partial monotonicity on a real case study of housing pricing. Furthermore, as in the simulation studies, the performance of the approach is compared to standard neural networks with weight decay and partially monotone linear models.

The data used in the study consist of 119 observations of houses in the Dutch city of Den Bosch. There are 11 independent variables describing the characteristics of a house (Table II). The dependent variable we want to predict is the house price and for further computational convenience it was transformed by taking its logarithm.

TABLE II
DEFINITION OF VARIABLES

Symbol	Variable
DISTR	Type of district, 4 categ. ranked from bad to good
AREA	Total house area including garden
RM	Number of bedrooms
TYPE	House type, 6 categ., ranked from flat to villa
VOL	Volume of the house
GARD	Type of garden, 4 categ. ranked from bad to good
GARG	1-no garage, 2-normal garage, 3-large garage
FLOORS	Number of floors
YEAR	Year of building
X-DIST	Horizontal map location
Y-DIST	Vertical map location

This data set has been used in previous studies ([3], [8]), which deal with incorporating monotonicity constraints in data mining algorithms but only for totally monotone problems. Therefore, in these studies, YEAR, X-DIST and Y-DIST were dropped out from the data as variables for which monotone relationship with the house price does not hold. Furthermore, we suspect that the monotonicity dependency on FLOORS is not expected (e.g., some expensive houses such as villas may have only one floor). Therefore, we conduct a test to check for which variables in the housing data the monotonicity assumption holds. This is done by using a measure for the degree of monotonicity (DgrMon) of data, namely the fraction of monotone pairs of all comparable pairs in the data. This measure is computed for the original data and for the data sets obtained after removing one or more of the four variables, for which we suspect lack of a monotone relationship with the house price. Based on the results in Table III, we can consider FLOORS, X-DIST, Y-DIST, and YEAR as the non-monotone variables in the data.

TABLE III
DEGREE OF MONOTONICITY FOR HOUSING DATA

Removed variable(s)	Comparable pairs	DgrMon
- (original data)	314	0.9140
FLOORS	331	0.9184
X-DIST	412	0.9199
Y-DIST	634	0.9495
YEAR	1073	0.9553
Y-DIST,YEAR	1534	0.9615
FLOORS,Y-DIST,YEAR	1620	0.9630
X-DIST,Y-DIST,YEAR	2217	0.9648
FLOORS,X-DIST,Y-DIST,YEAR	2345	0.9659

Using this knowledge about the (non)-monotone relationships in the housing data, we apply the approach for partial monotonicity (PartMon) in order to build a model for predicting the house price. To obtain a sound assessment of the generalization capabilities of the model obtained, we split randomly the original data into training data of 89 observations (75%) and test data of 30 observations (25%). The former is used to build a model whereas the latter is used to test the performance of the model. The random partition of the data is repeated 20 times.

Similarly to the simulation studies, we compare the performance of the approach with standard neural networks with weight decay (NNet) and partially monotone linear models (PMonLin), which are applied on the same data samples. We use again several combinations (in total 9) of parameters for the Sill networks (groups–2,3,4; planes–2,3,4) and neural networks (hidden nodes–5,13,20; weight decay–0.000001,0.00001,0.0001). The performance of the models is measured by computing the mean-squared error (MSE). The mean and variance of the minimum MSE over different parameter combinations are reported in Table IV.

To check the significance of the results we performed t-tests. Since the test set in the experiments with the three methods is the same, there is a natural pairing of the estimated errors. Therefore we use a paired t-test to test the null hypothesis that the models derived from the approach for partial monotonicity have the same error as the standard neural networks / partially monotone linear models against the one-sided alternatives. In addition, we perform F-tests for the significance in the variance difference of the models. The p-values obtained from all tests are given in Table V.

TABLE IV
MINIMUM MSES OBTAINED FROM THE EXPERIMENTS WITH HOUSING DATA

Method	Minimum MSE	
	Mean	Variance
PartMon	0.0158	0.0000
NNet	0.0201	0.0001
PMonLin	0.0217	0.0001

TABLE V
P-VALUES OBTAINED FROM THE STATISTICAL TESTS

Indicator	*p*-value	
	Mean	Var
Minimum MSE (PartMon–NNet)	0.005	0.010
Minimum MSE (PartMon–PMonLin)	0.000	0.033

The results show that the error obtained from the approach for partial monotonicity is significantly smaller than those obtained from the standard neural networks and partially monotone linear models. Furthermore, the significantly lower variances of the models derived from the approach for partial monotonicity show that they are more stable and robust upon repeated sampling.

V. CONCLUSION

In this paper we considered partially monotone regression problems where the response variable depends monotonically on some but not all predictor variables. An approach for building partially monotone models was presented, which is convolution of weight functions (kernels) based on the non-monotone variables and monotone (Sill) networks built only on the monotone variables. Simulation and real case studies showed that the overall performance of the approach is significantly better compared to the standard neural networks and partially monotone linear models. First the models derived from our approach are more accurate than the partially monotone linear models. Our method provided also a better fit than standard neural networks on real housing data. Further comparison with neural networks demonstrates an additional advantage of the proposed approach: faster training time due to smaller monotone networks built on subsets of data in contrast to training a single network with many parameters required to learn the whole large data. Also the incorporation of partial monotonicity constraints leads not only to models that are in accordance with the decision maker's expertise but also to significant reduction of the model variance, which results in more robust models.

REFERENCES

[1] A. Ben-David, "Monotonicity Maintenance in Information-Theoretic Machine Learning Algorithms", *Machine Learning*, 19, (1995), pp. 29-43
[2] C. Cybenko, "Approximation by Superpositions of a Sigmoidal Function", *Mathematics of Control, Signals, and Systems*, 2, (1989), pp. 303-314
[3] H.A.M. Daniels, and B. Kamp, "Application of MLP Networks to Bond Rating and House Pricing", *Neural Computation and Applications,* 8, (1999) pp. 226-234
[4] O. Harrison, and D. Rubinfeld, "Hedonic Prices and The Demand for Clean Air", *Journal of Environmental Economics and Management*, 53, (1978), pp.81-102
[5] H. Kay, and L.H. Ungar,, "Estimating Monotonic Functions and Their Bounds", *AIChE Journal,* 46, (2000), pp.2426-2434
[6] H. Mukarjee, and S. Stern, "Feasible Nonparametric Estimation of Multiargument Monotone Functions", *Journal of the American Statistical Association,* 89, (1994), pp. 77-80
[7] E.A. Nadaraya, "On Estimating Regression", *Theory Prob. Applic.*, 10, (1964), pp.186-90
[8] R. Potharst, and A. Feelders, "Classification trees for problems with monotonicity constraints", *SIGKDD Explorations Newsletter*, 4, (2002), Issue 1
[9] P.J. Rousseeuw, "Silhouettes: a graphical aid to the interpretation and validation of cluster analysis", *Journal of Computational and Applied Mathematics*, 20, (1987), pp.53-65
[10] M. Sarfraz, M. Al-Mulhem, and F. Ashraf, "Preserving Monotonic Shape of The Data by Using Piecewise Rational Cubic Functions", *Computers and Graphics*, 21, (1997), pp.5-14
[11] J. Sill, "Monotonic Networks", *Advances in Neural Information Processing Systems*, 10, (1998), pp.661-667
[12] S. Wang, "A Neural Network Method of Density Estimation for Univariate Unimodal Data", *Neural Computation* & *Applications*, 2, (1994), pp.160-167
[13] G. S. Watson, "Smooth Regression Analysis", *Sankhya*, Ser. A, 26, (1964), pp. 359-372
[14] Wu, C.F.J., and M. Hamada, *Experiments: Planning, Analysis, and Parameter Design Optimization*, Wiley Series in Probability and Statistics, John Wiley & Sons, Inc. New York, (2000).

APPENDIX

TABLE VI
RESULTS OBTAINED FROM THE SIMULATION STUDIES

Method	50 points			150 points			250 points		
	σ_ε^2=0.01	σ_ε^2=0.5	σ_ε^2=2	σ_ε^2=0.01	σ_ε^2=0.5	σ_ε^2=2	σ_ε^2=0.01	σ_ε^2=0.5	σ_ε^2=2
Minimum MSE/ Variance MSE									
PartMon*	**0.05/2e-05** (2, 3)	**0.11/ 2e-05** (2, 3)	**0.27/ 3e-05** (2, 3)	**0.02/ 5e-06** (2, 3)	**0.05/ 5e-06** (2, 3)	**0.11/ 7e-06** (2, 3)	**0.03/ 1e-05** (2, 3)	**0.05/ 1e-05** (2, 3)	**0.08/ 2e-05** (2, 3)
NNet*	**0.02/6e-03** (9,1e-006)	**0.10/ 1e-03** (9,1e-005)	**0.20/ 4e-02** (9,1e-004)	**0.01/ 9e-03** (9,1e-006)	**0.03/ 2e-03** (9,1e-006)	**0.09/ 1e-03** (9,1e-005)	**0.02/ 3e-02** (9,1e-006)	**0.03/ 6e-03** (9,1e-006)	**0.09/ 7e-03** (9,1e-006)
PMonLin	**0.15/ 2e-07**	**0.17/ 2e-06**	**0.25/ 3e-05**	**0.14/ 1e-07**	**0.15/ 5e-07**	**0.18/ 6e-06**	**0.14/ 1e-07**	**0.15/ 3e-07**	**0.17/ 2e-06**

***** The numbers in the brackets present the parameters of the Sill/ordinary network for which the minimum MSE is achieved.
PartMon - (# groups, # planes); NNet - (# hidden neurons, weight decay)