Nonlinear Regression with Logistic Product Basis Networks

Mehdi Sajjadi, Mojtaba Seyedhosseini, and Tolga Tasdizen, *Senior Member, IEEE*

***Abstract*—We introduce a novel general regression model that is based on a linear combination of a new set of non-local basis functions that forms an effective feature space. We propose a training algorithm that learns all the model parameters simultaneously and offer an initialization scheme for parameters of the basis functions. We show through several experiments that the proposed method offers better coverage for high-dimensional space compared to local Gaussian basis functions and provides competitive performance in comparison to other state-of-the-art regression methods.**

***Index Terms*—Basis functions, feature space, RBF, regression.**

I. Introduction

LINEAR combination of a set of basis functions is one of the most commonly used methods for regression. By choosing a proper set of basis functions, we can project the data into a feature space that provides effective representation of the data distribution [1]. Common basis functions include but are not limited to polynomials, natural splines, radial basis functions (RBF) and wavelet bases [2]. The choice of the basis functions determines the type of regression method. RBFs are important mathematical models and use radial functions such as Gaussians as basis functions [3]. Each radially symmetric Gaussian is tuned to respond to a local region of feature space. Therefore, RBFs require sufficiently large number of basis functions to cover the space.

In general, some regression methods originate from popular classification methods such as Random Forests [4], Support Vector Machines and Artificial Neural Networks. For example, random forest regression uses ensembles of regression trees instead of classification trees to make the final prediction [5]. In this paper, however, our attention is on a broad class of methods that work in *weight-space* according to [6]. The most commonly used example is linear regression. There have been many works trying to improve this linear model. For example, *Robust Regression*[7] was proposed to reduce the effect of outliers in training data. In this paper we introduce a new structure which is based on a new set of non-local basis functions and provides efficient coverage of high-dimensional spaces. We refer to the proposed structure as Logistic Product Basis Network (LPBN). We compare LPBNs with state-of-the-art regression methods such as Random Forests, Support Vector Regression [8] and RBFs.

Manuscript received October 16, 2014; revised November 23, 2014; accepted November 30, 2014. Date of publication December 18, 2014; date of current version December 23, 2014. This work was supported by NSF Grant IIS-1149299. The associate editor coordinating the review of this manuscript and approving it for publication was Prof. Xiaadong He.

The authors are with the Department of Electrical and Computer Engineering, University of Utah, Salt Lake City, UT 84112 USA (e-mail: mehdi@sci.utah.edu; mseyed@sci.utah.edu; tolga@sci.utah.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/LSP.2014.2380791

II. Related Work

Consider the problem of finding a function $f : \mathbf{R}^n \to \mathbf{R}$ for fitting a set of training data $\Gamma = \{\mathbf{x}_i \in \mathbf{R}^n, y_i \in \mathbf{R}\}_{i=1}^{p}$ such that $f(\mathbf{x}_i) = y_i$. Let's specify f as a linear combination of N basis functions $f(\mathbf{x}) = \sum_{i=1}^{N} \alpha_i \phi_i(\mathbf{x})$ with $\phi_i(\mathbf{x})$ the basis functions that map the data points to the feature space. This mapping is performed in order to obtain a better representation of the data and overcome the limited expressiveness of a linear model. The model is reduced to simple linear regression by setting $\phi_i(\mathbf{x}) = \mathbf{x}$. α_i are the weights of the linear combination. There is a broad range of regression models that are based on this form. Support Vector Regression (SVR) can be considered as an example of linear methods. In SVR, α_i are determined by minimizing the ℓ_2-norm of the weight vector α. The condition of this minimization is that the maximum deviation of the predicted value $f(\mathbf{x}_i)$ from y_i shouldn't exceed ϵ which is called the *margin of tolerance*[9]. The basis functions or mappings are implicitly determined by the type of kernel being used. A more recent approach is the *Extreme Learning Machine (ELM)*[10] for regression. They map the training data to ELM feature space. The feature mapping layer in ELM need not be tuned. It is also possible to use kernels if the mapping function is unknown. RBF regression is another example of the methods working in weight-space. The set of basis functions $\phi_i(\mathbf{x})$ are Gaussians and their parameters can be obtained by unsupervised clustering of the data or fitting a Gaussian mixture model to our data using the EM algorithm [11]. Then, we can obtain the linear weights α_i using linear regression. It is also possible to learn the RBF parameters in an unified manner by back-propagation of the error using chain rule [12]. RBFs are popular for modeling, regression and interpolation. Therefore, there are many works on improvement of the training and performance of these models. One approach is to use regularization in order to improve the generalization ability of the model and avoid overfitting (e.g. [13]). However, most regularization schemes are general techniques and can be applied to many other methods including our proposed structure. It is well known that an RBF network suffers from the curse of dimensionality [14]. In other words, as the number of dimensions grow, the number of radial basis functions required grows exponentially. Another work similar to our approach is Sum-Product Networks (SPN) [15]. SPNs are probabilistic models that provide tractable inferences. SPN is based on the notion of *network polynomial* and represents unnormalized probability distributions. This leads to a deep structure with interleaved layers of sums and products. SPN is similar to our proposed structure because it uses sum units to mix different submodels with mixing weights corresponding to α_i. It also uses products that combine features of submodels. However, our approach is different. Our proposed structure is based on a flexible set of basis functions. Unlike SPNs, we use logistic sigmoid functions before the product layer to approximate half-spaces. Then, we use products to form basis functions that are convex polytopes. Sigmoid functions are not present in SPNs but are crucial components of our approach. These basis functions are able to provide both local and non-local coverage of high-dimensional space which leads to more efficient representation of data space.

III. Methods

A. Network Architecture

In our proposed structure, we build the basis functions $\phi_i(\mathbf{x})$ by the intersection $\cap_{j=1}^{M_i}\mathcal{H}_{ij}$ of M_i half-spaces. $\mathcal{H}_{ij}$ is defined in terms of its indicator function

$$h_{ij}(\mathbf{x}) = \begin{cases} 1, & \sum_{k=1}^{n} w_{ijk}x_k + b_{ij} \geq 0 \\ 0, & \text{otherwise} \end{cases} \tag{1}$$

where w_{ijk} and b_{ij} are the weights and the bias term. Intersection of the half-spaces forms a convex polytope that covers desired parts of a multi-dimensional space. However, unlike RBFs, this is not necessarily a local coverage and we can cover the space more efficiently using flexible basis functions.

The indicator functions are binary variables. So, it is possible to rewrite the intersection of half-spaces as the conjunction of the indicator functions $\wedge_{j=1}^{M_i} h_{ij}(\mathbf{x})$. Our next step is to find a differentiable approximation for this general form. We replace the conjunctions by the product $\prod_{j=1}^{M_i} h_{ij}(\mathbf{x})$. This product can be considered as a soft AND gate that implements the conjunction. $h_{ij}(\mathbf{x})$ can be approximated with logistic sigmoid functions

$$\sigma_{ij}(\mathbf{x}) = \frac{1}{1+e^{-\sum_{k=1}^{n} w_{ijk}x_k + b_{ij}}}, \tag{2}$$

We can replace M_i with $M = \max_i M_i$. Based on our experiments, this replacement does not hurt the performance by overfitting. Hence, our basis functions will be in the form $\phi_i(\mathbf{x}) = \prod_{j=1}^{M} \sigma_{ij}(\mathbf{x})$ and we can construct the final fitting function as

$$f(\mathbf{x}) = \sum_{i=1}^{N} \alpha_i \left(\prod_{j=1}^{M} \sigma_{ij}(\mathbf{x}) \right) \tag{3}$$

This function can be visualized as a network shown in Fig. 1. We refer to the proposed network architecture as LPBN. In this structure, N is the number of the basis functions which is the same as the number of soft AND gates and M is the number of the logistic sigmoid functions per basis function. This structure is referred to as a $N \times M$ LPBN.

B. Model Initialization

The specific choice of the basis functions allows for a simple and intuitive initialization of the weights and biases of the logistic sigmoid functions. We can partition training data points $\{\mathbf{x}_i \in \mathbf{R}^n\}_{i=1}^{p}$ into N clusters. For the i'th cluster, we let $\mathbf{v}_{ij} = \mathbf{c}_i - \mathbf{c}_j$ where $\mathbf{c}$ are the cluster centroids and $j \neq i$. We initialize the weight vectors as $\mathbf{w}_{ij} = \mathbf{v}_{ij}/|\mathbf{v}_{ij}|$ for $i \neq j$. Finally, we initialize the bias terms b_{ij} such that the logistic sigmoid functions $\sigma_{ij}(\mathbf{x})$ take the value 0.5 at the midpoints of the lines connecting

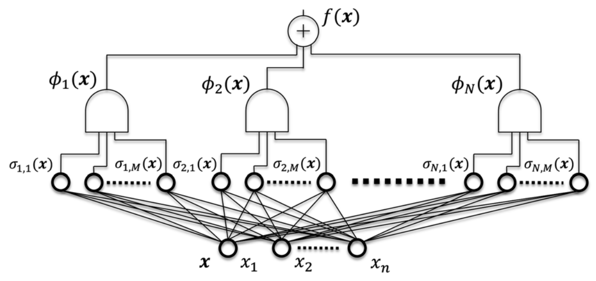
$f(\boldsymbol{x})$
$\phi_1(\boldsymbol{x})$
$\phi_2(\boldsymbol{x})$
$\phi_N(\boldsymbol{x})$
$\sigma_{1,1}(\boldsymbol{x})$
$\sigma_{1,M}(\boldsymbol{x})$
$\sigma_{2,1}(\boldsymbol{x})$
$\sigma_{2,M}(\boldsymbol{x})$
$\sigma_{N,1}(\boldsymbol{x})$
$\sigma_{N,M}(\boldsymbol{x})$
$\boldsymbol{x}$
x_1
x_2
x_n

Fig. 1. LPBN architecture. The first hidden layer is composed of $M \times N$ logistic sigmoid functions. The second hidden layer computes N conjunctions using soft AND gates. The output layer is a linear combination of the soft gates. The soft AND gates are implemented as continuous functions by product of their inputs.

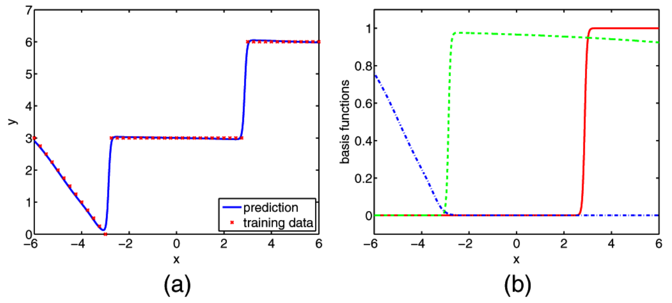

Fig. 2. LPBN performance on synthetic data using 3 basis functions: (a) training data and LPBN predictions, and (b) basis functions after training.

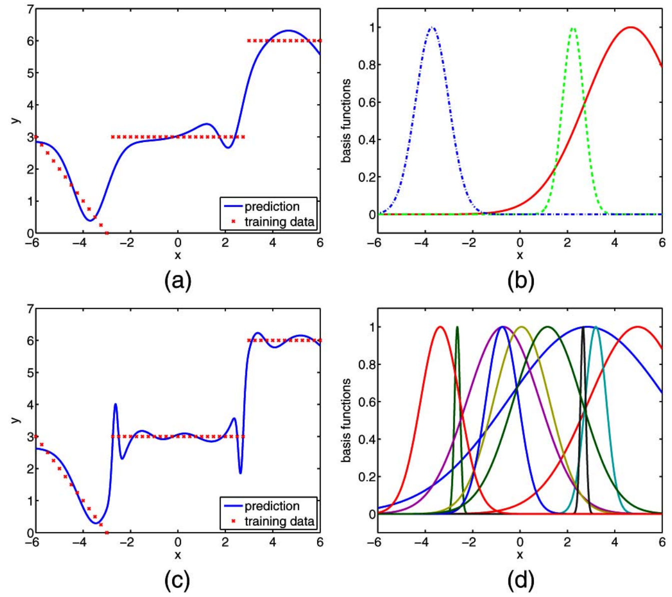

Fig. 3. RBF performance on synthetic data: training data and RBF predictions with (a) three kernels and (c) ten kernels, and the Gaussian kernels after training with (b) three kernels and (d) ten kernels.

$\mathbf{c}_i$ and $\mathbf{c}_j$. In other words, let $b_{ij} = \langle \mathbf{w}_{ij}, 0.5(\mathbf{c}_i + \mathbf{c}_j)\rangle$ where $\langle \mathbf{a}, \mathbf{b}\rangle$ denotes the inner product of the vectors $\mathbf{a}$ and $\mathbf{b}$. This procedure initilizes $\phi_i(\mathbf{x})$, the i'th basis function, to a convex polytope that separates the training instances in the i'th cluster from all other training data. This creates a $N \times N-1$ LPBN.

C. Model Optimization

The LPBN model can be trained by choosing the parameters $\mathcal{W} = \{\alpha_i, w_{ijk}, b_{ij}\}$ that minimize the quadratic error

$$E(\mathcal{W}, \Gamma) = (y(\mathbf{x}) - f(\mathbf{x}))^2, \ \mathbf{x} \in \Gamma \tag{4}$$

TABLE I

COLUMN 1: REGRESSION DATASETS, THEIR SOURCE, NUMBER OF TRAINING/TESTING EXAMPLES AND DATA DIMENSIONALITY. **COLUMN 2**: METHOD OF REGRESSION. **COLUMN 3–6**: AVERAGE TRAINING, AVERAGE TESTING OVER 50 ROUNDS, [MIN,MAX] TESTING R^2 AND COMPUTATION TIME (SECONDS). BEST AVERAGE TESTING RESULTS ARE SHOWN IN BOLD. **COLUMN 7**: MODEL AND CLASSIFIER TRAINING PARAMETERS USED. LPBN: N NUMBER OF BASIS FUNCTIONS AND ϵ STEP SIZE. RF: t NUMBER OF TREES, f NUMBER OF FEATURES CONSIDERED PER NODE AND s TRAINING INSTANCE SAMPLING RATE FOR EACH TREE. ϵ-SVR: C PENALTY FACTOR, γ RBF KERNEL WIDTH AND ϵ MARGIN OF TOLERANCE. NN: N NUMBER OF HIDDEN NODES AND ϵ STEP SIZE. ELM: C REGULARIZATION PARAMETER, γ KERNEL WIDTH AND N NUMBER OF BASIS FUNCTIONS.

Dataset, source and properties	Method	Av. train R^2	Av. test R^2	Test R^2 range	Time	Model Parameters
Concrete compressive strength	LPBN	0.9610	0.8750	[0.8598, 0.8901]	0.39	$N = 7$, $\epsilon = 0.001$, Epochs $= 300$
UCI	RBF	0.7931	0.7663	[0.7531, 0.7804]	0.16	No of RBF functions $= 30$
Train: 515 / Test: 515	RF	0.9556	0.8485	[0.8429, 0.8553]	0.08	$t = 100$, $f = 5$, $s = 2/3$
Dim = 8	ϵ-SVR	0.9306	0.8456	—	0.08	$C = 150$, $\gamma = 2.85$, $\epsilon = 0.1$
	NN	0.9685	**0.8840**	[0.8641, 0.9051]	0.40	$N = 42$, $\epsilon = 0.001$, Epochs $= 300$
	ELM	0.9431	0.8629	—	0.02	$C = 45$, $\gamma = 12$, kernel $=$ RBF
Energy efficiency - Heating Load	LPBN	0.9957	0.9886	[0.9784, 0.9932]	0.08	$N = 6$, $\epsilon = 0.01$, Epochs $= 100$
UCI	RBF	0.9951	0.9898	[0.9867, 0.9922]	0.16	No of RBF functions $= 45$
Train: 384 / Test: 384	RF	0.9982	0.9940	[0.9932, 0.9948]	0.04	$t = 100$, $f = 6$, $s = 2/3$
Dim = 8	ϵ-SVR	0.9998	0.9892	—	0.67	$C = 600$, $\gamma = 2.25$, $\epsilon = 0.1$
	NN	0.9981	0.9842	[0.9813, 0.9878]	0.07	$N = 30$, $\epsilon = 0.01$, Epochs $= 700$
	ELM	0.9994	**0.9963**	—	0.01	$C = 55000$, $\gamma = 24$, kernel $=$ RBF
Energy efficiency - Cooling Load	LPBN	0.9911	**0.9708**	[0.9399, 0.9839]	0.45	$N = 12$, $\epsilon = 0.02$, Epochs $= 150$
UCI	RBF	0.9741	0.9653	[0.9642, 0.9663]	0.16	No of RBF functions $= 45$
Train: 384 / Test: 384	RF	0.9829	0.9605	[0.9584, 0.9629]	0.04	$t = 100$, $f = 6$, $s = 2/3$
Dim = 8	ϵ-SVR	0.9961	0.9577	—	0.43	$C = 390$, $\gamma = 2.2$, $\epsilon = 0.2$
	NN	0.9862	0.9673	[0.9403, 0.9818]	0.47	$N = 132$, $\epsilon = 0.02$, Epochs $= 150$
	ELM	0.9927	0.9675	—	0.01	$C = 1500$, $\gamma = 12.5$, kernel $=$ RBF
Abalone	LPBN	0.6051	**0.5825**	[0.5734, 0.5904]	0.30	$N = 5$, $\epsilon = 0.001$, Epochs $= 100$
Libsvm	RBF	0.5387	0.5411	[0.5181, 0.5551]	0.23	No of RBF functions $= 12$
Train: 2088 / Test: 2088	RF	0.6750	0.5542	[0.5511, 0.5575]	0.51	$t = 400$, $f = 6$, $s = 1/5$
Dim = 8	ϵ-SVR	0.5721	0.5511	—	0.36	$C = 10$, $\gamma = 2.1$, $\epsilon = 0.1$
	NN	0.5920	0.5816	[0.5660, 0.5912]	0.26	$N = 20$, $\epsilon = 0.001$, Epochs $= 100$
	ELM	0.6120	0.5797	—	0.09	$C = 20$, $\gamma = 18$, kernel $=$ RBF
Cpusmall	LPBN	0.9785	0.9728	[0.9710, 0.9739]	9.12	$N = 15$, $\epsilon = 0.0005$, Epochs $= 150$
Libsvm	RBF	0.9638	0.9615	[0.9590, 0.9631]	0.94	No of RBF functions $= 30$
Train: 4096 / Test: 4096	RF	0.9927	**0.9740**	[0.9733, 0.9741]	4.75	$t = 200$, $f = 8$, $s = 2/3$
Dim = 12	ϵ-SVR	0.9792	0.9732	—	3.34	$C = 250$, $\gamma = 2.2$, $\epsilon = 0.1$
	NN	0.9816	0.9719	[0.9649, 0.9732]	9.75	$N = 210$, $\epsilon = 0005$, Epochs $= 150$
	ELM	0.9809	0.9700	—	0.32	$C = 1500$, $\gamma = 100$, kernel $=$ RBF
MG	LPBN	0.7402	0.7195	[0.7085, 0.7289]	1.53	$N = 10$, $\epsilon = 0.02$, Epochs $= 500$
Libsvm	RBF	0.7130	0.7032	[0.6934, 0.7120]	0.24	No of RBF functions $= 40$
Train: 692 / Test: 692	RF	0.8940	**0.7270**	[0.7197, 0.7309]	0.17	$t = 200$, $f = 3$, $s = 2/3$
Dim = 6	ϵ-SVR	0.7789	0.7033	—	0.07	$C = 17$, $\gamma = 3$, $\epsilon = 0.1$
	NN	0.6384	0.6315	[-0.0340, 0.7174]	1.62	$N = 90$, $\epsilon = 0.02$, Epochs $= 500$
	ELM	0.7671	0.7169	—	0.02	$C = 15$, $\gamma = 5$, kernel $=$ RBF
Space GA	LPBN	0.7541	**0.7341**	[0.7090, 0.7450]	1.29	$N = 8$, $\epsilon = 0.1$, Epochs $= 300$
Libsvm	RBF	0.6961	0.6949	[0.6769, 0.7065]	0.44	No of RBF functions $= 37$
Train: 1553 / Test: 1553	RF	0.9027	0.6137	[0.6102, 0.6165]	0.89	$t = 200$, $f = 5$, $s = 3/4$
Dim = 6	ϵ-SVR	0.7634	0.7105	—	0.52	$C = 21$, $\gamma = 3.2$, $\epsilon = 0.06$
	NN	0.6490	0.6470	[0.4607, 0.7150]	1.00	$N = 56$, $\epsilon = 0.1$, Epochs $= 300$
	ELM	0.7146	0.6986	—	0.07	$C = 370$, $\gamma = 41$, kernel $=$ RBF
Million Song Dataset (MSD)	LPBN	0.3356	**0.3138**	—	2270.51	$N = 4$, $\epsilon = 10^{-7}$, Epochs $= 1500$
UCI	RBF	0.2668	0.2554	—	242.00	No of RBF functions $= 145$
Train: 463715 / Test: 51630	RF	0.8398	0.2693	—	79805.00	$t = 100$, $f = 30$, $s = 4/5$
Dim = 90	ϵ-SVR	0.8390	0.2827	—	249060.00	$C = 20$, $\gamma = 60$, $\epsilon = 0.1$ (1/2 of training data)
	NN	0.3135	0.2888	—	3145.12	$N = 12$, $\epsilon = 10^{-7}$, Epochs $= 1500$
	ELM	0.3272	0.2972	—	4019.00	$C = 10^5$, $N = 5000$, basis functions $=$ Gaussian
Relative location of CT slices	LPBN	0.9969	0.9942	—	60024.00	$N = 17$, $\epsilon = 2 \times 10^{-6}$, Epochs $= 7000$
UCI	RBF	0.8654	0.8617	—	27.00	No of RBF functions $= 360$
Train: 26750 / Test: 26750	RF	0.9989	0.9955	—	768.67	$t = 200$, $f = 40$, $s = 4/5$
Dim = 385	ϵ-SVR	0.9994	0.9953	—	4022.30	$C = 10$, $\gamma = 0.1$, $\epsilon = 0.1$
	NN	0.9987	**0.9970**	—	67647.00	$N = 272$, $\epsilon = 2 \times 10^{-6}$, Epochs $= 7000$
	ELM	0.9692	0.9574	—	420.00	$C = 10^7$, $N = 15000$, basis functions $=$ Gaussian

Starting from initialization described in Section III-B for logistic functions and random initialization for α_i, we minimize (4) using gradient descent. In order to get the update equations, we need to find the partial derivatives of the error with respect to all the network weights and biases. It is done via chain rule

$$\begin{aligned}\frac{\partial E}{\partial w_{ijk}} &= \frac{\partial E}{\partial f}\frac{\partial f}{\partial \phi_i}\frac{\partial \phi_i}{\partial \sigma_{ij}}\frac{\partial \sigma_{ij}}{\partial w_{ijk}} \\ &= -2(y - f(\mathbf{x}))\alpha_i\left(\prod_{l\neq j}\sigma_{il}(\mathbf{x})\right)\sigma_{ij}(\mathbf{x})(1-\sigma_{ij}(\mathbf{x}))x_k \\ &= 2(f(\mathbf{x}) - y)\alpha_i\phi_i(\mathbf{x})\left(1-\sigma_{ij}(\mathbf{x})\right)x_k \qquad (5)\end{aligned}$$

Similarly, we obtain the derivative of the error function with respect to the network biases as

$$\frac{\partial E}{\partial b_{ij}} = 2(f(\mathbf{x}) - y)\alpha_i\phi_i(\mathbf{x})\left(1-\sigma_{ij}(\mathbf{x})\right) \qquad (6)$$

Partial derivatives with respect to α_i is $\frac{\partial E}{\partial \alpha_i} = \phi_i(\mathbf{x})$.

IV. EXPERIMENTS

First, we compare RBF and LPBN using a set of synthetic 1-dimensional data points shown in Figs. 2(a) and 3(a). We trained both networks with 3 basis functions. Network predictions are shown in the same Figures using solid lines. We can see that RBF failed to provide a good prediction for this data. On the other hand, LPBN fits the data efficiently. The basis functions learned by the networks are shown in Figs. 2(b) and 3(b). We observe that every basis function of the LPBN corresponds to a specific part of our data and are not necessarily local. We repeated the experiment for RBF using 10 basis functions. The network predictions and basis functions are shown in Figs. 3(c) and 3(d) respectively. We can see that RBF still doesn't fit the data very well. Next, we experimented with 8 regression datasets from *UCI Machine Learning Repository*[16] and the LIBSVM Tools webpage [17]. We tried to include datasets with different sizes and dimensions. However, datasets with too few or extremely high dimensions were excluded. We also avoided datasets with missing values. *Energy Efficiency* dataset has two different targets. We compared LPBN with Random Forests, ϵ-SVR, Neural Networks with one hidden layer, ELM and RBFs.

A. Dataset Normalization, Training/Testing Set Split

Datasets were normalized as follows: For LPBN, Neural Networks, ELM and RBF, all the datasets were normalized by cen-

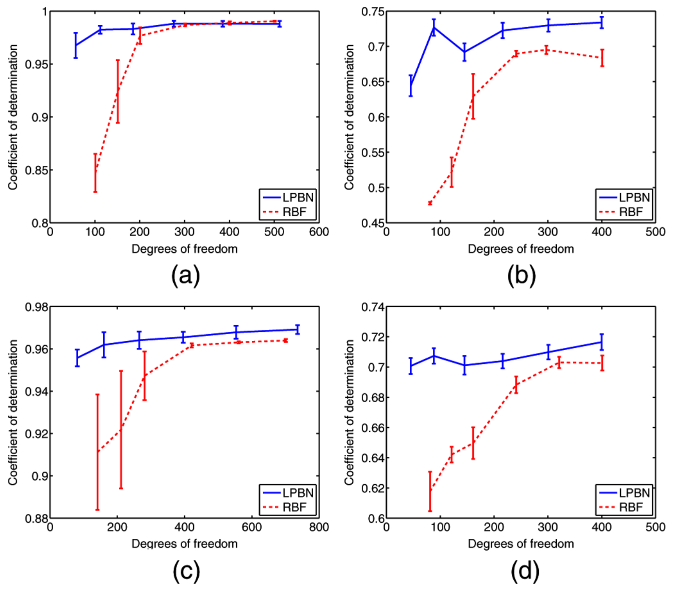

Fig. 4. Coefficients of determination vs degrees of freedom for the LPBN and RBF models: These graphs compare LPBN and RBF in terms of number of learning parameters for 4 datasets. Every experiments is repeated 50 times and the mean of R^2 is shown versus the number of free parameters. Variance of the experiments are shown by error bars. (a) Energy - Heating Load (b) Space GA (c) Cpusmall (d) MG.

tering each dimension of the feature vector at the origin by subtracting its mean and then scaling by dividing it with its standard deviation. For ϵ-SVR training, each dimension of the feature vector was linearly scaled to the range [0,1]. For all the datasets with the exception of *MSD*, we randomly picked half of the instances for training and the other half for testing. We used (10%) of the training data as cross-validation set to find the optimal training parameters.

Model and Classifier Training Parameter Selection: For LPBN we need to choose the number of basis functions (N), which is also the number of clusters for initialization process. For every dataset, we tried different numbers of basis functions in the range of 2 to 20 in order to find the selection that gives the best accuracy on the cross-validation set. Similarly, we found the gradient descent parameters (i.e., step size and number of training epochs) using the cross-validation set. For Neural Network, the number of hidden nodes and parameters of gradient descent were found using cross-validation set. We performed our experiments on Random Forest using the code available in [18]. For RF training, the first parameter is the number of trees. We choose a sufficiently large number of trees to ensure that the out of bag error rate converges. The second parameter is the number of features that will be considered in every node of the tree. We tried a range of numbers around the square root of the number of features [4]. The last parameter is the fraction of total samples that will be used in the construction of each tree. We tried 3/4, 2/3, 1/2, 1/3, 1/4 and 1/5 as possible values for this parameter. For ϵ-SVR training, a RBF kernel was used for all the datasets. The main parameters are γ of RBF kernel, penalty coefficient (C) and ϵ[19]. We performed a grid search to find these parameters using cross-validation set. For the first 6 datasets, we trained ELMs with Gaussian kernels. We found the kernel width (γ) and regularization parameter (C) using a grid search similar to ϵ-SVR. It was not possible to use kernel based ELM for the last two datasets because of their size. So, we used ELMs with Gaussian basis functions. The main parameters are the number of basis functions (N) and regularization parameter (C). We obtained these parameters by performing a grid search on cross-validation set. RBFs were trained using Netlab [20]. Netlab performs a few steps of k-means to initialize unsupervised learning of a Gaussian Mixture Model using Expectation-Maximization (EM) algorithm. The predicted value is calculated by linearly combining Gaussian kernels. Linear weights are obtained by least squares fitting. For every dataset, we need to find the number of basis functions. So, we tried up to 50 basis functions to pick the best using cross-validation set. We evaluate the regression methods using coefficient of determination (R^2 value). The training and model parameters selected for all models are listed in Table I.

Results: All of the regression methods we consider, with the exception of ϵ-SVR and ELM with kernels are stochastic. Therefore, each experiment on the first 6 datasets was repeated 50 times for stochastic methods in order to obtain mean, minimum and maximum of R^2 on test sets which are reported in Table I for all the methods. For the last two datasets, it was not possible to repeat experiments 50 times mainly because of training times. It was also infeasible to train ϵ-SVR on all training samples of *MSD* dataset. It can be seen that LPBNs performed better than other methods in 4 out of 9 regression problems and performed close to the best on the rest. LPBNs also outperform RBFs in 8 out of 9 problems.

We also compared LPBN with RBF in terms of number of learning parameters. We trained various LPBNs with $N = [3, 8]$ and RBFs with $N = [10, 50]$ for the number of basis functions. For every setting, we repeated the experiments 50 times and computed the average of the R^2 value for the test set for each model. The results are plotted against the degrees of freedom (total number of parameters) in Fig. 4 for different datasets. The variance of different experiments for every model is shown by error bars. We can see that LPBNs provide better performance with fewer degrees of freedom. This is more obvious when both methods have a small number of learning parameters. This is mainly because LPBNs provide non-local basis functions which provide better coverage of the space compared to local basis functions of RBFs. LPBNs also offer better coverage in higher dimensions. Consider the results on the last two datasets in Table I. These datasets are relatively large with high dimensions. We can see that LPBNs perform better than RBFs with significantly less basis functions. This is mainly because RBFs require exponentially more basis functions as the number of dimensions grow. On the other hand, the basis functions of LPBNs are non-local which can allow for a more efficient representation in higher dimensions. However, it must be noted that unlike RF, our method is not designed for very high dimensional data. Finally, we also tried using back-propagation to fine-tune the RBF networks further [12], but this resulted in only minor accuracy improvements.

V. Conclusion

We introduced LPBN as a general method of regression and proposed a training algorithm and an effective initialization scheme. The training algorithm learns all the weights simultaneously. We showed that LPBNs provide competitive results compared to popular regression methods. Especially, we showed that LPBNs provide more efficient space coverage compared to RBFs because of a more flexible set of non-local basis functions that leads to a feature space that represents the data distribution very well.

References

[1] K. Muller, S. Mika, G. Ratsch, K. Tsuda, and B. Scholkopf, "An introduction to kernel-based learning algorithms," *IEEE Trans. Neural Netw.*, vol. 12, no. 2, pp. 181–201, 2001.

[2] R. Kohn, M. Smith, and D. Chan, "Nonparametric regression using linear combinations of basis functions," *Statist. Comput.*, vol. 11, no. 4, pp. 313–322, 2001.

[3] D. S. Broomhead and D. Lowe, Radial basis functions, multi-variable functional interpolation and adaptive networks DTIC Document, Tech. Rep., 1988.

[4] L. Breiman, "Random forests," *Mach. Learn.*, vol. 45, no. 1, pp. 5–32, 2001.

[5] N. Meinshausen, "Quantile regression forests," *The J. Machine Learning Research*, vol. 7, pp. 983–999, 2006.

[6] C. E. Rasmussen*, Gaussian Processes for Machine Learning*. Cambridge, MA, USA: MIT Press, 2006.

[7] D. Huang, R. S. Cabral, and F. De la Torre, "Robust regression," in *Computer Vision–ECCV 2012*. Berlin, Germany: Springer, 2012, pp. 616–630.

[8] V. Vapnik, S. E. Golowich, and A. Smola, "Support vector method for function approximation, regression estimation, and signal processing," in *Advances in Neural Information Processing Systems 9*. Cambridge, MA, USA: MIT Press, 1996, pp. 281–287.

[9] A. J. Smola and B. Schölkopf, "A tutorial on support vector regression," *Statist. Comput.*, vol. 14, no. 3, pp. 199–222, 2004.

[10] G.-B. Huang, H. Zhou, X. Ding, and R. Zhang, "Extreme learning machine for regression and multiclass classification," *IEEE Trans. Syst, Man Cybern. B: Cybern.*, vol. 42, no. 2, pp. 513–529, 2012.

[11] M. J. Orr, "Recent advances in radial basis function networks," in *Relatório técnico, Centre for Cognitive Science*. Edinburgh, U.K.: Univ. Edinburgh Press, 1999.

[12] F. Schwenker, H. A. Kestler, and G. Palm, "Three learning phases for radial-basis-function networks," *Neural Netw.*, vol. 14, no. 4, pp. 439–458, 2001.

[13] C. Bishop, "Improving the generalization properties of radial basis function neural networks," *Neural Comput.*, vol. 3, no. 4, pp. 579–588, 1991.

[14] J.-N. Hwang, S.-R. Lay, and A. Lippman, "Nonparametric multivariate density estimation: A comparative study," *IEEE Trans. Signal Process.*, vol. 42, no. 10, pp. 2795–2810, 1994.

[15] H. Poon and P. Domingos, "Sum-product networks: A new deep architecture," in *2011 IEEE Int. Conf. Computer Vision Workshops (ICCV Workshops)*, 2011, pp. 689–690.

[16] "UCI machine learning repository," [Online]. Available: archive.ics.uci.edu/ml

[17] "Libsvm," [Online]. Available: www.csie.ntu.edu.tw/cjlin/libsvmtools

[18] "Random forest," [Online]. Available: code.google.com/p/random-forest-matlab/

[19] C.-W. Hsu, C.-C. Chang, and C.-J. Lin *et al.*, "A practical guide to support vector classification," 2003.

[20] "Netlab neural network software," [Online]. Available: www.aston.ac.uk/eas/research/groups/ncrg/resources/netlab