In Proc. of the 14th BNAIC, Leuven, 2002.

Extracting multivariate power functions from complex data sets

E.M. Oost[ab] S.H.G. ten Hagen[a] F.H. Schulze[bc]

[a] University of Amsterdam, Kruislaan 403, 1098SJ Amsterdam
[b] Witteveen + Bos, P.O. Box 233 7400AE Deventer
[c] TU Delft CiTG, P.O. Box 5048 2600GA Delft

Abstract

In this paper, we present a modification of the RF5 hybrid neural equation discovery/rule extraction system. We show for synthetic and real data that the modification enables the system to handle more complex data sets.

1 Introduction

As renowned as multi-layer perceptrons (MLPs) are for being good predictors for a large variety of tasks, as notorious they are for the difficulty of extracting meaningful relationships between their inputs and outputs. Quoting Mozer and Smolensky [10],

> *One thing that connectionist networks have in common with brains is that if you open them up and peer inside, all you can see is a big pile of goo.*

In this paper we will look at the automated extraction of humanly understandable input/output relations from real data.

Over the years, numerous methods have been developed which aim to extract these relationships from networks, none of which have as yet established themselves as overall winner. Several surveys discussing these methods exist [1, 2, 11]. Most of these methods are intended for classification problems on binary or nominal data. Those that support regression problems usually first discretisize the output to form if-then-else rules, which we feel limits their explanatory power.

Equation discovery systems such as BACON [5] use a generate-and-test loop for formulas to directly fit them on the data. These formulas show us the relationships in the data set. Unfortunately the overhead of the formula generation process restricts the number of formulas that can be tried. Also they are usually much more sensitive to noise.

In this paper we will look into an alternative network representation which after training will provide insight in the relationships among features present in the data.

We introduce improvements to this framework in section 3, which enables its use on larger scaled problems. In section 4 we show that our improved approach is able to automatically extract understandable relationships from a real large-scale data set.

2 Background

The product unit was introduced in 1989 by Durbin and Rumelhart [3]. They provide a powerful addition to the toolbox of neural network researchers, as they allow higher orders of the unit inputs to be used. They are defined as

$$\prod_{j=1}^{n_x} x_j^{p_j} \qquad \text{instead of the regular summation unit} \qquad \sum_{j=1}^{n_x} w_j x_j$$

in which p is a power weight, w a multiplicative weight and n_x the number of inputs. A single product unit can therefore obtain relationships like $y = x_1^{3.14} / \sqrt[2.7]{x_2}$, which would require a network of summation units to approximate. On the other hand, they cannot create sums and as such can't replace summation units. Also, they seem more prune to creating local minima [6].

Networks should not fully consist of product units, as these would be no more powerful than a single unit. Instead, the most common approach is to have a hidden layer of product units and one summation unit as output, as proposed by Durbin and Rumelhart [3]. No biases are used in the hidden layer as these would fulfill the same multiplicative role the output weights have. This product unit network (PUN) structure is also the one we will use in this paper. Like regular MLPs [4], such product unit networks are universal approximators [7].

In 1997, Saito and Nakano published RF5 [14, 16], an equation discovery system using these PUNs. Besides these PUNs it included their regularization method [15] for the number of hidden units based on the MDL principle [13], and their quasi-Newton learning algorithm BPQ [17]. The function formed by a trained PUN is much more comprehensible than that from a trained MLP.

If we define n_h as the number of hidden units, n_x the number of inputs, w the weights between hidden and output (with w_0 being the bias weight), and p the input weights, this network structure can fit the following kind of functions:

$$y = \sum_{i=1}^{n_h} w_i \prod_{j=1}^{n_x} x_j^{p_{ij}} + w_0 \tag{1}$$

An example of such a function is:

$$y = 1.23 x_1^{-0.73} x_2^{2.34} x_3^{0} - 4.56 x_1^{0} x_2^{2} x_3^{-0.25} + 7.89$$

which can be simplified to

$$y = 1.23 x_1^{-0.73} x_2^{2.34} - 4.56 x_2^{2} x_3^{-0.25} + 7.89$$

To prevent output in the complex domain, the input vectors should be restricted to non-zero, positive values. After training the weights p and w can directly be interpreted as powers and coefficients respectively.

To train the network, RF5 uses BPQ [17], a quasi-Newton method they developed to use less memory and have more efficient step length calculations than other quasi-Newton methods. For determining the optimal size of the network, RF5 uses an MDL regularizer [15] on the number of hidden units, calculated using

$$\text{MDL} = 0.5 n_m \log(\text{MSE}) + 0.5 n_w \log(n_m)$$

with n_m being the number of samples, MSE the mean squared error and n_w the total number of weights (of all layers).

Saito and Nakano tested the PUN on artificial cases, real data and a time series. The most difficult artificial formula was

$$y = 2 + 3x_1^{-1}x_2^3 + 4x_3x_4^{1/2}x_5^{-1/3}$$

where the training set consisted of 9 inputs (thus, including 4 not used in the function) and 200 samples, with each value in the $\langle 0, 1]$ range[1]. Since no standard-deviation is mentioned we assume they have used a uniform random distribution. Gaussian noise with standard deviation 0.1 was added to the targets.

The real-data experiments by Saito and Nakano are rather small, requiring just one or two inputs and one hidden unit. As such, we will not describe them here. Finally, they present the result for a time series, which is not predicted very well.

3 RF5 improvements

Saito and Nakano demonstrated the use of RF5 in real data experiments with one or two inputs and one hidden unit. In these simple experiments RF5 is not very useful because understandable input and output relations can also be obtained by plotting the data. RF5 can be a very useful technique when applied to complex data sets that can not be understood by simply plotting the data.

Our first improvement is replacing BPQ training with the Levenberg-Marquardt [8, 9] algorithm. The Levenberg-Marquardt algorithm seems better suited for training PUNs as quasi-Newton algorithms, because it excels for relatively small networks with one output, trained on regression problems [19]. It does not perform well when residuals are large. This is not problem in the context of equation discovery, since large residuals indicate that any extracted formulas is not representative for the data.

Our second improvement is to minimize the risk of being trapped in local minima using a breadth-first search, in which we do many brief initialize-and-train rounds and continue the most promising network. For large data sets, we propose series of training rounds with an incremental number of hidden units. If after training the generalization error for a particular number of hidden units is better than for the previous rounds, one new unit is added to the network. All existing

[1] The text mentions $[0, 1]$, but all inputs should be positive.

	MSE			Iterations		Time (s)	
Units	Minimum	Average	Std. dev.	Average	Std. dev.	Average	Std. dev.
1	46.35	98.68	45.69	47.46	20.97	0.83	0.38
2	$3.87 \cdot 10^{-2}$	70.54	123.75	75.01	43.68	1.57	0.96
3	$4.80 \cdot 10^{-2}$	80.54	241.48	132.11	74.65	3.23	1.88

Table 1. Learning statistics for the modified Hochstein formula

weights p and w keep their value and the new unit is initialized with random power weights and a zero multiplicative weight. This way, well-generalizing networks are given extra degrees of freedom as long as they are doing well.

Our third improvement is the use of the test set error as indication of generalization performance. Saito and Nakano used MDL. The generalization performance in MDL focuses on the capabilities of the network representation, while for the test set error it is based on the data itself. So the test set error indicates the actual generalization performance given the current weights, instead of the potential generalization for other possible weights.

Note that for equation discovery there is an extra argument to reduce the number of hidden units. Too many hidden units enlarge the resulting equation, making it harder to understand. But regularizing the number of hidden units is not enough if the input space is high dimensional. Each hidden unit will contribute a term consisting of all inputs, which still result in complicated expressions. Inputs that hardly influence the output of a unit should be pruned away, to further simplify the equation [12]. The pruning of inputs is beyond the scope of this paper.

4 Experiments

4.1 Synthetic data

To demonstrate RF5 with our improvements on synthetic data sets, we chose to simulate a hydrological formula. We did this to show that natural laws can indeed be discovered when present in the data. The artificial data set was created to model the 'modified Hochstein' formula [18], defined as

$$y = r_1 U_s \left(\left(\frac{A_c}{A_w} \right)^{1.25} - 1 \right) b' + s_1$$

in which r_1 and s_1 are constants, and the other factors variables. For training, we used tiny training rounds of 10 epochs, resetting the weights when a round yielded less than 1% MSE decrease. The tests were performed on a 700Mhz Athlon machine.

The setup of the test was comparable to the synthetic test of Saito and Nakano, except that we have raised the bar for the network in a couple of ways. We used a data set with 15 (instead of 9) inputs, 4 of which are actually used in the formula. For the input value distribution, we opted for a normal distribution instead of the

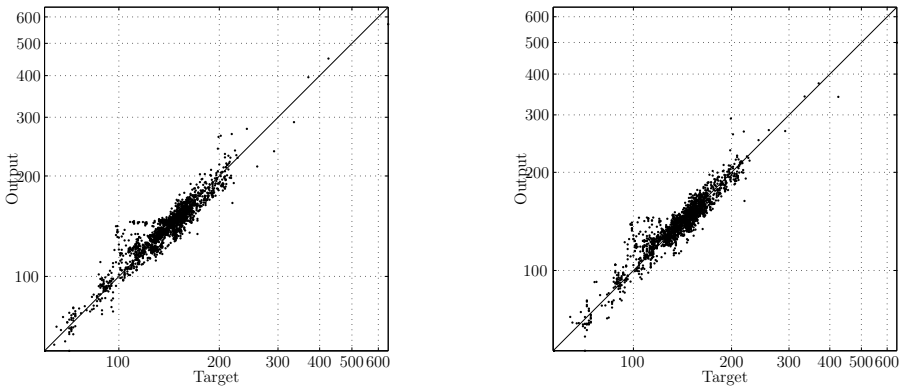

Figure 1. Log-log distribution of PUN (left) and MLP (right) test targets versus outputs

uniform distribution apparently used in Saito and Nakano's work, as we believe this to be more in line with real-world input distributions. Also, we added noise to the targets with standard deviation 0.2 instead of 0.1. The sample size remained 200. To select the number of hidden units, we have chosen to extract 20% of these for a test set, instead of training on all data and using the MDL criterion.

In the formula, we used constants $r_1 = 3$ and $s_1 = 2$. The results from 100 trials for 1, 2 and 3 hidden units are summarized in table 1. By rounding off after the second digit and removing weights with resulting value zero, we obtain the formula

$$2.06 + 3.06U_s^{1.00}A_c^{1.25}A_w^{-1.25}b'^{1.00} - 3.07U_s^{0.99}A_c^{0.01}A_w^{-0.01}b'^{1.00}$$

which is indeed practically identical to the original formula. The $R^2 \approx 1.00$ of the test set indicates an almost perfect fit[2].

4.2 Real data

Equation discovery only makes sense if the equation is not know and only some data set is available. We used the *chloride* data set to test our approach. The chloride data set is used to predict the amount of chloride in the canal Noord at a certain location. It consists of 8627 samples with 15 explanatory variables including chloride levels in other waterways at different time intervals, run-off of a nearby river, nearby water levels and wind speed.

Of specific interest were the large chloride peaks, which were quite rare (approximately 0.5 percent) but not attributable to measurement errors. To make sure both train set and test set still had a balanced distribution of peak samples we have used stratified sampling over the target space and used a relatively large test set (20%).

[2] R^2 is a measure of goodness of fit, being the proportion of variance in the target explained by the model, ranging from 0 for none to 1 for perfect fit.

	w	p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8	p_9	p_{10}	p_{11}	p_{12}	p_{13}	p_{14}	p_{15}
h_1	$4.5 \cdot 10^{138}$	6.5	0.9	-0.4	1.82	-5.5	4.1	-10.3	24.1	-7.4	-16.7	-47.2	-9.0	-1.1	-4.9	27.7
h_2	$4.0 \cdot 10^{-16}$	-0.0	0.1	0.1	0.2	-0.4	0.2	0.4	2.3	0.1	2.8	-2.6	4.6	-0.0	-0.0	0.3
h_3	$-3.1 \cdot 10^{-18}$	0.0	0.1	0.0	0.2	-0.4	0.2	0.7	2.7	0.1	2.6	-2.8	5.2	-0.0	-0.0	0.4
h_4	$1.0 \cdot 10^{-20}$	2.6	2.1	5.4	-7.4	13.0	-5.4	-1.5	15.8	-31.4	19.0	4.6	-12.4	0.6	-0.1	5.1
h_5	$4.0 \cdot 10^{-4}$	0.0	0.1	-0.0	0.3	-0.5	0.2	1.0	3.0	0.1	2.3	-2.7	5.6	0.0	-0.0	0.5
h_6	$-4.5 \cdot 10^{1}$	-0.0	0.0	0.0	0.1	-0.1	0.1	-0.2	-1.0	-0.3	0.9	0.3	-0.0	0.0	-0.0	0.1

Table 2. Explanatory variables w and p for chloride data set. $w_0 = 147.4$.

Figure 1 shows a scatter plot of the 20% test targets versus net output for respectively a PUN and a MLP. Best generalization performance was obtained using a PUN with 6 hidden units, and a MLP with 5 hidden units. For the PUN test set, $R^2 \approx 0.90$, and for the MLP $R^2 \approx 0.89$. Thus, the PUN slightly outperforms the MLP for this problem.

The weights of the network are shown in table 2. They can be directly interpreted as a function by starting off with bias w_0 and considering each row a separate summation term with coefficient w_i, and each input x_j raised to its respective power weight p_{ij}. Note that the coefficients of units do not tell us much about the importance of the units; a hidden unit with small coefficient may itself have a large activation value and vice versa.

To determine the behavior of the hidden units, we have created histograms of the weighed output of the hidden units (thus, of the hidden unit activations multiplied with their coefficients w), as shown in figure 2. These are single summation terms in the resulting equation. The X-axis has been translated to have mean 0 for ease of reference[3]. The hidden units have clearly distinct characteristics.

First, units 1 and 4 are clearly peak predictors. They are close to zero for almost the entire data set, except for rare cases in which they yield extreme values. Considering the resulting powers, inputs 8 and 15 are examples which both units appear to find very relevant for obtaining peaks[4].

Secondly, unit 2, 3 and 5 are highly related, with unit 3 damping the output of units 2 and 5. In table 2 this is clearly visible by the near-identical powers of the hidden units; only for some inputs the powers are different per unit. Thus, a composite term can be created of the form

$$y = (w_2 x_a^{p_{2a}} x_b^{p_{2b}} + w_3 x_a^{p_{3a}} x_b^{p_{3b}} + w_5 x_a^{p_{5a}} x_b^{p_{5b}})\, x_c^{p_c} x_d^{p_d} \ldots$$

where a and b are inputs with different powers p per hidden unit and c and d are inputs with identical powers for these three hidden units. As can be seen in the sum histogram of figure 2, the combined terms actually predict relatively small-scale chloride fluctuations.

[3]Hidden units may have output ranges far from the origin or target values; any range translation is compensated by bias w_0.

[4]Note that evaluation of importance strictly by power is mathematically unsound here as the input vector shape is also important, but a deeper analysis of these inputs is beyond the scope of this paper.

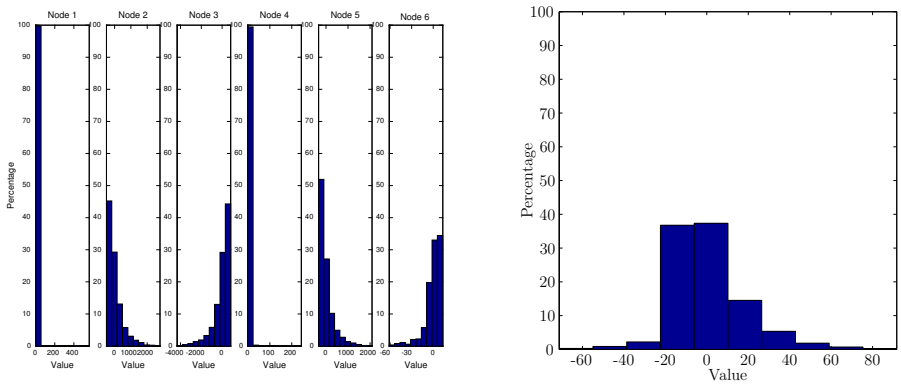

Figure 2. Histogram of weighed hidden unit output (l); sum for units 2, 3 and 5 (r)

Lastly, unit 6 has a similar small-scale behavior compared to the other units. It is unlikely to be of particular interest for peak prediction.

5 Conclusion

The use of product unit networks for discovering understandable relationships in data sets is not restricted to small problems. When using Levenberg-Marquardt and proper initialization they can be trained as easily as a regular MLP. They perform similar to MLPs, but have an additional advantage that the functions they form are comprehensible equations.

References

[1] R. Andrews, J. Diederich, and A. B. Tickle. A survey and critique of techniques for extracting rules from trained artificial neural networks. *Knowledge-Based Systems*, 8:373–389, 1995.

[2] O. Boz. *Knowledge integration and rule extraction in neural networks.* PhD thesis, EECS Department, Lehigh University, US, 1995.

[3] R. Durbin and D. E. Rumelhart. Product units: A computationally powerful and biologically plausible extension to backpropagation networks. *Neural Computation*, 1(1):133–142, 1989.

[4] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989.

[5] P. Langley. Bacon: A production system that discovers empirical laws. In *Proceedings of the 5th International Joint Conference on Artificial Intelligence (IJCAI-77)*, page 344. Morgan Kaufman, 1977.

[6] L. R. Leerink, C. L. Giles, B. G. Horne, and M. A. Jabri. Learning with product units. In *Advances in Neural Information Processing Systems*, volume 7, pages 537–544. The MIT Press, 1995.

[7] M. Leshno, V. Y. Lin, A. Pinkus, and S. Schocken. Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. *Neural Networks*, 6(6):861–867, 1993.

[8] D. W. Marquardt. An algorithm for least-squares estimation of non-linear parameters. *Journal for the Society of Industrial and Applied Mathematics*, 11(2):431–441, 1963.

[9] J. J. Moré. The Levenberg-Marquardt algorithm: Implementation and theory. In G. A. Watson, editor, *Numerical Analysis*, number 630 in Lecture Notes in Mathematics, pages 105–116. Springer Verlag, 1977.

[10] M. Mozer and P. Smolensky. Using relevance to reduce network size automatically. *Connection Science*, 1:3–16, 1989.

[11] J. Neumann. Classification and evaluation of algorithms for rule extraction from artificial neural networks, August 1998. PhD Summer Project, ICCS, Division of Informatics, University of Edinburgh, United Kingdom.

[12] E. M. Oost. Opening Pandora's box, bottom side up: Automated extraction of comprehensible multivariate power functions from real data. Master's thesis, University of Amsterdam, IAS Group, August 2002.

[13] J. Rissanen. A universal prior for integers and estimation by Minimum Description Length. *Annals of Statistics*, 11(2):416–431, 1983.

[14] K. Saito and R. Nakano. Law discovery using neural networks. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI-97)*, pages 1078–1083, San Francisco, August 23–29 1997.

[15] K. Saito and R. Nakano. MDL regularizer: a new regularizer based on MDL principle. In *Proceedings of the 1997 International Conference on Neural Networks (ICNN-97)*, pages 1833–1838, 1997.

[16] K. Saito and R. Nakano. Numeric law discovery using neural networks. In *Proceedings of the 4th International Conference on Neural Information Processing (ICONIP-97)*, pages 843–846, 1997.

[17] K. Saito and R. Nakano. Partial BFGS update and efficient step-length calculation for 3-layer neural networks. *Neural Computation*, 9(1):123–141, 1997.

[18] F. H. Schulze. *Rijkswaterstaat RIZA: Kunstmatige neurale netwerken toegepast op sedimenttransport in kribvakken langs de Waal, gevoeligheidsanalyse, fase 1.* Witteveen+Bos, P.O. Box 233, 7400AE Deventer, NL, 1999.

[19] A. J. Shepherd. *Second-order methods for neural networks: Fast and reliable training methods for multi-layer perceptrons.* Perspectives in Neural Computing. Springer Verlag, May 1997.