ESANN'1996 proceedings - European Symposium on Artificial Neural Networks
Bruges (Belgium), 24-25-26 April 1996, D-Facto public., ISBN 2-9600049-6-5, pp. 279-284

An algorithm for training multilayer networks on non-numeric data

Wojciech Kowalczyk

Vrije Universiteit Amsterdam
Department of Mathematics and Computer Science
De Boelelaan 1081a
1081 HV Amsterdam
The Netherlands
wojtek@cs.vu.nl

Abstract. *Artificial neural networks work with numbers. To apply them to non-numeric data one has to represent this data by numbers. Finding an appropriate numeric representation is a difficult task which usually involves some general heuristics, common sense and some intuitions about the behaviour of neural nets and the problem itself. In this paper we present an approach to automatic discovery of optimal representations of data for feed-forward multilayer networks. The resulting algorithm extends in a natural way the standard backpropagation scheme.*

1. Introduction

Many practical problems that are solved by neural networks involve non-numeric data. For example, in the field of medical diagnosis one deals with symptoms, observations, types of diseases, etc. The first successful application of neural networks, NETtalk [9], dealt with characters. On the other hand, neural networks, almost by definition, can process only numbers. Therefore, before applying a neural network to non-numeric data, one has to represent this data by numbers. Sometimes also numeric data has to be transformed into other form which is more suitable in the given context. Surprisingly, although it is obvious that the final performance of the network strongly depends on the chosen representation, there is no systematic methodology for constructing such representations. In the available literature only some specific aspects of data representation are investigated: missing values, discretization of domains, scaling, [6], [3]. Moreover, in several textbooks on neural networks, e.g., [1] or [10], one can find some general hints dealing with this issue.

Let us consider the following example. Suppose that we want to apply a neural network in the context of car insurance. One of the variables that will certainly play an important role is the color of a car. For simplicity, let us restrict the set of possible colors to [white, green, red, black]. How should we represent these colors by numbers? Should they be mapped into [1, 2, 3, 4]? And maybe into [1, 5, 4, 9]? Units of the network are monotonic with respect to their inputs so it is reasonable to search for such representations that reflect somehow a "structure of importance" of these colors. In other words, any representation imposes some (implicit) topology on colors. For example, the first representation may reflect our intuition about visibility of cars: white cars are more visible than blue, blue are more visible than red, etc. The second representation reflects another intuition: white cars are much better visible than red and

ESANN'1996 proceedings - European Symposium on Artificial Neural Networks
Bruges (Belgium), 24-25-26 April 1996, D-Facto public., ISBN 2-9600049-6-5, pp. 279-284

blue ones, black cars are very bad with that respect. Clearly, it is difficult to make a choice without deeper knowledge about the importance of colors in this particular context. Usually we take a representation which, in our opinion, reflects the "importance structure" of colors in the most adequate way; the network should, hopefully, adjust itself to possible mistakes that we have made, producing good results. One might also try to use a 'neutral' vector representation: white is represented by [1, 0, 0, 0], blue by [0, 1, 0, 0], etc. But it implies adding extra input nodes to the network and usually leads to networks that have very bad generalization performance (especially, when the number of colors is relatively big).

In this paper we will focus on the issue of automatic discovery of optimal data representations. Obviously, to talk about optimal representations we should have a specific network in mind: different networks may require different representations. In our approach we focus on conventional multilayer feed-forward networks. For such networks we present a generalized backpropagation algorithm that operates directly on arbitrary data (not necessarily numeric) producing an optimal representation and simultaneously training the network. The algorithm can also be used for problems that involve both numeric and non-numeric data.

2. Definitions and notation

Let us consider a collection of training pairs $<p_1, t_{p1}>$, $<p_2, t_{p2}>$, ..., $<p_s, t_{ps}>$, where $\boldsymbol{P} = \{p_i : i = 1, ..., S\} \subset D_1 \times ... \times D_I$ is a collection of input patterns and $t_{p1}, t_{p2}, ..., t_{p2} \in \boldsymbol{R}^K$ are corresponding output patterns. Without loss of generality we may assume that every domain D_i has exactly n elements, i.e., $D_i = \{d_{i1}, d_{i2}, ..., d_{in}\}$, for $i = 1,..., I$. To develop a neural network with I input units that could be trained on our training set we have to represent elements of the domains D_i by numbers. To this end let us introduce, for every element d_{ij}, a corresponding variable x_{ij} (a 'code' of d_{ij}) that will be instantiated in $\boldsymbol{R}$. In this way we can associate with any pattern $p = <d_{1i1}, d_{2i2}, ..., d_{IiI}>$ a vector of variables $x_p = < x_{1p1}, x_{2p2}, ..., x_{IpI}>$ and a vector of their values. Finding these values is the key problem of this paper.

Let us consider an arbitrary feed-forward network N with I input units and K output units and let w_{ji} denote the weight of a connection going from unit u_i to u_j. The behaviour of this network on input pattern p (or, more precisely: on input vector x_p) can be described as follows:

the output of any input unit i , o_{pi}, is just the value of the variable x_{ipi}

the output of any non-input unit j, o_{pj}, is $f(net_{pj})$, where f is a differentiable activation function (e.g. sigmoid) and $net_{pj} = \Sigma w_{ji} o_{pi}$.

Now we can define the error made by our network on the training set as a function of connection weights $\mathbf{w}$ and variables $\mathbf{x} = (x_{ij})$:

$$E(\mathbf{w},\mathbf{x}) = \sum_{p \in P} E_p(\mathbf{w},\mathbf{x}), \text{ where } E_p(\mathbf{w},\mathbf{x}) = 0.5\sum_k (t_{pk} - o_{pk})^2$$

The problem of finding an optimal representation is defined as follows:

Definition

Any assignment of variables $\mathbf{x} = (x_{ij})$ that minimizes the function

$$E(\mathbf{x}) = \inf_{\mathbf{w}} E(\mathbf{w}, \mathbf{x})$$

is called an optimal representation of P with respect to N.

Obviously, the problem of finding an optimal representation is strongly related to the problem of finding a global minimum of $E(\mathbf{w}, \mathbf{x})$: the function E is continues, so

$$\inf_{\mathbf{x}}(E(\mathbf{x})) = \inf_{\mathbf{w},\,\mathbf{x}}(E(\mathbf{w}, \mathbf{x}))$$

thus the global minimum of E determines an optimal representation and an optimal setting of weights. From now on we will focus on the problem of finding a minimum of $E(\mathbf{w}, \mathbf{x})$.

3. The Algorithm

To find a minimum of $E(\mathbf{w}, \mathbf{x})$ we will modify the standard backpropagation algorithm which is based on the gradient method [8]: starting with some arbitrary values of $\mathbf{w}$ and $\mathbf{x}$ we walk in the direction of $-\nabla E(\mathbf{w}, \mathbf{x})$ making steps of fixed size. To apply this method we have to know partial derivatives of $E(\mathbf{w}, \mathbf{x})$, or, equivalently, of $E_p(\mathbf{w}, \mathbf{x})$.

Theorem

Partial derivatives of $Ep(\mathbf{w}, \mathbf{x})$ are given by the following formulas:

$$\frac{\partial E_p}{\partial w_{ij}} = -\delta_{pj} o_{pi}, \text{ and}$$

$$\frac{\partial E_p}{\partial x_{pi}} = -\sum_j \delta_{pj} w_{ij}, \text{ where}$$

$$\delta_{pj} = (t_{pj} - o_{pj}) f'(net_{pj}) \text{ for output units, and}$$

$$\delta_{pj} = f'(net_{pj}) \sum_k \delta_{pk} w_{jk} \text{ for hidden units.}$$

Here the index j is varying over hidden units and k over output units.

Proof. For simplicity we will assume that the network has only one hidden layer; otherwise we should apply induction on the number of hidden layers. Let us note that variables w_{ij} do not depend on x_{pi} thus the derivation of the formula for $\partial E_p/\partial w_{ij}$ leads directly to the generalized delta rule, [8]. To derive $\partial E_p/\partial x_{pi}$ we have to apply the chaining rule several times:

$$\frac{\partial E_p}{\partial x_{pi}} = 0.5 \frac{\partial}{\partial x_{pi}} \sum_k (t_{pk} - o_{pk})^2 = -\sum_k (t_{pk} - o_{pk}) \frac{\partial o_{pk}}{\partial x_{pi}} =$$

$$-\sum_k (t_{pk} - o_{pk}) f'(net_{pk}) \frac{\partial net_{pk}}{\partial x_{pi}} = -\sum_k \delta_{pk} \frac{\partial}{\partial x_{pi}} \sum_j w_{jk} o_{pj} =$$

$$-\sum_k \delta_{pk} \sum_j wjk \frac{\partial o_{pj}}{\partial x_{pi}} = -\sum_k \delta_{pk} \sum_j w_{jk} f'(net_{pj}) \frac{\partial net_{pj}}{\partial x_{pi}}.$$

But

$$\frac{\partial net_{pj}}{\partial x_{pi}} = \frac{\partial}{\partial x_{pi}} \sum_h w_{hj} x_{ph} = w_{ij},$$

thus

$$-\sum_k \delta_{pk} \sum_j w_{jk} f'(net_{pj}) \frac{\partial net_{pj}}{\partial x_{pi}} = -\sum_k \delta_{pk} \sum_j w_{jk} f'(net_{pj}) w_{ij} =$$
$$-\sum_j w_{ij} f'(net_{pj}) \sum_k \delta_{pk} w_{jk} = -\sum_j \delta_{pj} w_{ij},$$

what completes the proof.

We can incorporate our formulas in an extended backpropagation algorithm that takes as input non-numeric data and produces optimal representations and connection weights:

```
0. For every element of every domain introduce a new variable $x_{ij}$
1. Initialize them (e.g. using available knowledge or randomly)
2. Initialize weights (e.g. randomly)
3. Select an input pattern $p$ and present the corresponding vector of values $x_p$ to the network
4. Calculate the actual output
5. For every unit compute its $\delta$
        - for output units:
                $\delta_{pj} := f'(net_{pj})(t_{pj} - o_j)$
        - for hidden units:
                $\delta_{pj} := f'(net_{pj})(\Sigma\delta_{pk}w_{kj})$
6. For every variable $x_{pi}$ compute its $\delta$
                $\delta_{pi} := \Sigma\delta_{pj}w_{ij}$
7. Update weights:
                $w_{ji} := w_{ji} + \eta\delta_{pj}o_{pi}$
8. Update 'codes':
                $x_{pi} := x_{pi} + \eta\delta_{pi}$
9. Goto 3
```

Let us note that our algorithm combines two processes: training the network (updating weights) and learning representations (updating *x*'s). Both processes can be controlled independently, for example by choice of different η's for steps 7 and 8 or by allocating different amounts of time (iterations) to these two steps. In this way the overall performance of the algorithm could be improved.

The formula for $\partial E_p/\partial x_{pi}$ could be obtained in a simpler way. Namely, one can notice that finding optimal values for *x*'s and *w*'s is equivalent to training a network that can be constructed from the original one. The construction is straightforward: every input unit of the original network is replaced by an ordinary neuron with *n* inputs and a linear activation function (i.e. $f(net) = net$). Each d_{ij} should be represented then by a vector of *n* numbers, with 1 on the *j*-th place and 0 on the remaining places (see Figure 1). The resulting network has a nice property: weights of connections between inputs and linear units correspond to *x*'s. Therefore, finding values of *w*'s and *x*'s is equivalent to training this modified network, which is just an ordinary feedforward network with an extra hidden layer of linear units. Clearly, this extra layer is not fully connected to the input layer. Now, by applying the standard delta-rule to this extra layer (and keeping in mind that for linear units we have $f'(net) = 1$) we immediately get the formula for $\partial E_p/\partial x_{pi}$.

The fact that the problem of finding optimal representations can be reduced to the problem of training a modified network has some advantages: one can directly use much faster algorithms (e.g., based on conjugate gradient methods) or more sophisticated error functions (e.g., based on Kullback's relative entropy measure) [2]. On the other hand, the algorithm presented above has some appealing simplicity and is more efficient than a straightforward application of backpropagation to the modified network: in the latter case a lot of effort would be wasted on processing numerous 0's present in the input vectors.

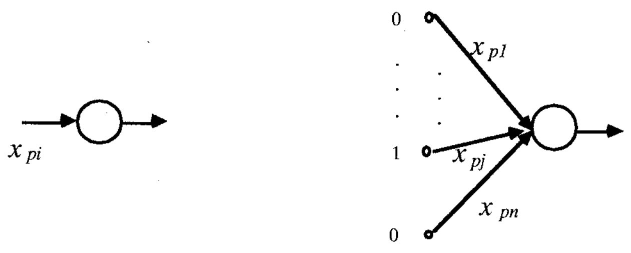

Figure 1. An input node that corresponds to *i*th domain and its counterpart in the modified network.

Finally let us note that our algorithm can be easily extended to deal with multi-dimensional representations. For instance, if we think that the color of a car provides two independent pieces of information (e.g., it says something about car's visibility and the character of the driver) it is reasonable to associate with every color two variables and then apply our algorithm for finding optimal values for them.

4. Experiments

The algorithm has been implemented and tested on *Auto Import Dataset* (available from Machine Learning Repository of University of California, ftp://ics.uci.edu/pub/machine-learning-databases/autos/. The data set contains 205 records that describe cars imported to the USA in 1985. Every record has 26 attributes: 10 nominal (including car make, body style, engine type, etc), 1 integer and 15 continuous. Some records contain missing attribute values. The set, restricted to numeric attributes, was used in the past (for predicting car prices) to illustrate the superiority of Instance-based learning (IBL) over linear regression, [4]. For our purposes we removed some nominal and all numeric attributes (except the target attribute, price) and consequently, all records that contained missing values (four). The remaining attributes were: car maker (22 values), fuel type and aspiration (both 2 values), body style (5 values), drive wheels (3 values) engine type, number of cylinders and fuel system (with 7, 7 and 8 values, respectively). The whole set was split into training set (101 cases) and test set (100 cases). Because we were interested in a comparison between our algorithm and the standard back-propagation, two series of experiments have been performed: one with ordinary networks (input vectors were represented by sequences of 0's and 1's of length 56) and another one with networks trained by our algorithm (input vectors were represented by sequences of 8 nominal values). As a measure of performance we used the root of mean squared error made by the network on the test set, RMS; the training was interrupted when this error started to increase, [6]. Numerous network architectures have been tried (about 20) and in all cases the difference in performance was not significant-about 1% with no clear winner. For example, for networks with 3 hidden units, the backpropagation algorithms reached minimal RMS= 0.061 after 550 epoch, whereas our algorithm reached 0.048 after 510 epochs. Therefore, on this particular problem, both algorithms perform equally well. On the other hand, our algorithm gives a better insight into values of attributes. For example, 'codes' discovered for different makes corresponded to our intuitions about make-price relation. In another experiment we trained a network (with 3 hidden units) using 2-dimensional representation of car make. The resulting set of points resembled a straight line, demonstrating that, with respect to price, car make should be considered as a one dimensional attribute.

5. Conclusions

Our approach to finding optimal representations has many advantages. A difficult problem of finding representations is now solved automatically by the algorithm. The problem of representing missing values can be solved in a similar way: they might be represented by the string `unknown` and the algorithm should find the best value representing it. The same applies to outliers–extreme values in the training set that have negative influence on the learning process. Moreover, because derived representation (x's) contains a lot of relevant information about the data, it might be expected that much smaller networks would be strong enough to learn the given mapping. Clearly, simpler networks are more suitable for getting an insight into the problem: the task of network interpretation (analyzing relations between inputs and outputs) becomes much easier. The representation itself should also provide a better insight into data. Results of experiments mentioned earlier strongly support this claim.

There are some open questions left. First of all, it is not clear how the generalization performance of the network would by affected by introducing extra variables. On the one hand, encoding extra information about the problem should have a negative impact on network's performance (due to the minimal description length principle, [7], [3]), but on the other hand, it may lead to simplifying the network and improving its performance. Another question deals with the overall behaviour of the algorithm: its speed, avoiding local minima, accuracy, robustness, etc. These questions can be answered only by performing more experiments on different problems.

References

[1] Gallant (1993). *Neural Networks: Learning and Expert Systems.* The MIT Press.

[2] Hertz, A. Krogh, and R. Palmer (1990). *Introduction to the Theory of Neural Computation.* Addison-Wesley: Redwood City, California.

[3] Kamimura, T. Takagi and S. Nakanishi (1994). "Improving generalization performance by information minimization." In *Proceedings of IEEE International Conference on Neural Networks*, Orlando 1994, 143-148.

[4] Kibler, D., Aha, D. W., and M. Albert (1989). "Instance-based prediction of real-valued attributes." *Computational Intelligence*, **5**, 51-57.

[5] Lee and D.G. Shin (1994). "A context-sensitive discretization of numeric attributes for classification learning." In *Proceedings of 11th European Conference on Artificial Intelligence*, Amsterdam 1994, 428-432.

[6] Prechelt, L. (1994). "Proben1–A Set of Neural Network Benchmarking Rules," University of Karlsruhe, Technical Report 21/94. Also available by ftp from `ftp.ira.uka.de` as `/pub/papers/techreports/1994/1994-21.ps.Z`.

[7] Rissanen (1978). "Modelling by shortest data description." *Automatica* **14**, 1978, 465-471.

[8] Rumelhart, G. Hinton, and R. Williams (1986). "Learning internal representations by error backpropagation." in Rumelhart, McClelland and the PDP research group, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition.* The MIT Press, 1986.

[9] Sejnowski and C.R. Rosenberg (1987). "Parallel networks that learn to pronounce English text." *Complex Systems* **1**, 145-168.

[10] Skapura (1996). *Building Neural Networks.* Addison-Wesley.