Multi-layer Perceptron on Interval Data*

Fabrice Rossi[1,3] and Brieuc Conan-Guez[2]

[1] LISE/CEREMADE, UMR CNRS 7534, Université Paris-IX Dauphine,
Place du Maréchal de Lattre de Tassigny, 75016 Paris, France
[2] INRIA, Domaine de Voluceau, Rocquencourt, B.P. 105
78153 Le Chesnay Cedex, France
[3] Up to date contact informations for Fabrice Rossi are avaible at
http://apiacoa.org/

Abstract. We study in this paper several methods that allow one to use interval data as inputs for Multi-layer Perceptrons. We show that interesting results can be obtained by using together two methods: the extremal values method which is based on a complete description of intervals, and the simulation method which is based on a probabilistic understanding of intervals. Both methods can be easily implemented on top of existing neural network software.

1 Introduction

Interval-valued data are quite natural in many applications where they represent uncertainty on measurements (confidence intervals for instance), variability (minimum and maximum temperatures during a day), extremal behavior (maximal wind speed in a given area), etc. Many data analysis tools have been already extended to handle in a natural way interval data: Principal Component Analysis, K-means, etc. (see for instance Bock and Diday, (2000)).

In this paper we focus on nonlinear processing of interval-valued data thanks to Multi-layer Perceptrons (MLP). Several methods can be used to allow MLP to work with interval-valued data. In this paper, we present two kind of methods: the very simple extremal values approach and two probabilistic methods. Those methods can be implemented very easily on top of existing neural network software. We show that the naive center (or mean) based method should be replaced by the simulation-based approach which gives in general better results. We show on synthetic data that the simple extremal values method should be used together with the simulation-based method in order to provide meaningful results.

* Published in IFCS'2002 Proceedings.
Available at http://apiacoa.org/publications/2002/ifcs02.pdf

2 Interval processing methods for MLP

2.1 Framework

We consider in this paper that each studied individual is described by n intervals, i.e. $([\underline{x_1}, \overline{x_1}], \ldots, [\underline{x_n}, \overline{x_n}])$. The desired output can be an interval, a real output, or a class: our main concern is to be able to work as efficiently and simply as possible with interval-valued inputs.

Moreover, we consider that interval-valued inputs are kind of summary of underlying precise data. For instance, if we study the climate, we can describe a place by the minimum and maximum temperatures during the day. We consider that the interval gives a good description of temperature variations during the day. One requirement of our study is to be able to use the trained MLP both on new interval-valued inputs and on new real valued inputs. For instance, if we observe a temperature during the day, we want to be able to use it as an input to the MLP, even if it was trained with interval-valued inputs.

A very natural way to handle interval-valued inputs (and outputs) is to rely on interval arithmetic (Moore (1966)). The main idea of interval arithmetic is simply to define in a sound way interval product, sum, etc. It is easy to define an interval based MLP, which can be trained thanks to a modified back-propagation algorithm. Several authors have already worked on this kind of model (see for instance Beheshti et al., (1998), Šíma (1995) and Simoff (1996)). In this paper, we won't work with interval arithmetic for one main reason: it implies specific development, which means that this approach cannot be easily integrated in existing neural network software. Every thing (initialization, training, visualization, etc.) has to be modified and adapted to interval arithmetic and we consider this is not affordable for many practitioners.

2.2 Extremal values method

The simplest way to deal with interval-valued inputs is to transform each interval in a pair of real numbers, for instance the lower and upper bounds of the interval (or the middle and the length of the interval). We translate this way n interval inputs into $2n$ real value inputs (i.e., $([\underline{x_1}, \overline{x_1}], \ldots, [\underline{x_n}, \overline{x_n}])$ is simply replaced by $(\underline{x_1}, \overline{x_1}, \ldots, \underline{x_n}, \overline{x_n})$). The MLP is used exactly as a classical MLP with the augmented inputs. We call this approach the **extremal values method**.

In order to use a MLP trained with the extremal values method on real valued inputs, the simplest method is to replicate the data, i.e., input $(x_1, \ldots, x_n)$ becomes $(x_1, x_1, \ldots, x_n, x_n)$. It might be possible to use more elaborated methods, but this is outside the scope of this article.

2.3 Probabilistic methods

Another way to deal with interval-valued data is to consider them as simple probabilistic data. If a sample for the MLP is described by the interval $[a, b]$, a possible interpretation is to assume that in fact the sample can take any value between a and b, with uniform probability.

Considering intervals are only a way to express uncertainty, one can replace each interval by its middle (the mean) and train the network with the obtained values. We call this approach the **mean method**. When we want to use a trained MLP with new data, we replace again each interval by its middle value. We handle real valued input directly.

Another way to proceed is to replace each sample by a set of real valued samples. Those samples are obtained thanks to simulation, assuming that the interval $[a, b]$ corresponds to an uniform distribution in $[a, b]$. Moreover, if we work with multiple interval inputs, we assume that each variable (each input) is independent from the others. We call this approach the **simulation method**. For new real valued inputs, we use the trained MLP directly. For new interval-valued inputs, we generate simulated real valued inputs and we compute normally corresponding outputs. One simple way to define the output corresponding to the initial data is to use the interval of simulated outputs. The practical meaning of this interval is the variability on the output induced by the variability on the input. We can also define the output for interval-valued inputs as the mean output for simulated inputs. Note that even if the MLP as been trained with the mean approach, the simulation approach can still be used to compute the output corresponding to an interval-valued input.

3 Comparison of probabilistic methods

3.1 Theoretical discussion

The mean and simulation methods can use exactly the same neural architecture. Therefore, training a MLP with the simulation method takes longer time than training it with the mean method, simply because we have more examples for the first method. This is the main drawback of the simulation method compared to the mean method.

The main drawback of the mean approach is that the MLP is trained without any knowledge of the uncertainty on the samples and it will provide overconfident answers in difficult cases, as will be shown in section 3.2. In fact, the mean trained MLP will often have worse generalization results than the simulation trained one. Indeed, using simulated data rather than the mean is quite similar to noise injection techniques which were introduced by Sietsma and Dow, 1991. The main idea of such techniques is to add noise to input data during the training of a MLP. Simulation results in the pioneer article showed that generalization performances where much improved by this

technique. Several theoretical analysis of noise injection techniques have been done (see for instance Bishop, (1995b) and Grandvalet et al., (1997)). They tend to prove that noise injection is an efficient way to improve generalization.

We can therefore consider that the simulation method is a data driven noise injection technique applied to the mean method. The main difference with noise injection is that in our case, data give a precise description of the noise which depends on the individual (and on the variable). Indeed, each input variable is associated to an observed interval value, whereas for noise injection methods, variations are artificially generated around observed values.

3.2 Simulation results

In difficult cases, uncertainty should decrease the quality of the result provided by the MLP. Let us consider a simple example. We assume that we have two classes. Elements of the first class are described by a real variable chosen uniformly in $[-1, 0]$. For the second class, the variable is chosen uniformly in $[0, 1]$. In order to take into account measurement error, each real value is replaced by a interval centered on the value and with a length of 0.2. We have 20 interval-valued examples from each class and we simulate 10 real valued examples for each original one in the simulation approach.

With the mean approach, the measurement error is not taken into account, therefore we have perfectly separated classes. A simple MLP (one input, one hidden neuron, one output) can be used to learn[1] to separate samples with no error. This is obviously an overconfident behavior. Indeed, when an input is close to zero, the MLP should not give a sharp answer (0 or 1), but on the contrary give an answer close to 0.5, as the measurement error implies that the value can come from either class.

With the simulation approach, we obtain training samples from class one with strictly positive value and samples from class two with strictly negative values. When we train the MLP (similar to the one use with the mean approach), the prediction error does not reach zero (the mean square error stays around 0.02). As illustrated by figure 1, for input values close to zero, the MLP does not give a sharp output, but, on the contrary, outputs varying between 0 (for class one) and 1 (for class two). In fact, as it is well known, e.g. Bishop, (1995a), the MLP approximates the posterior probability of the input to belong to the second class (which is trained with one as desired output). This behavior is better suited to imprecise inputs because it's the only correct way to show that for some inputs, we cannot obtain a class, but only a probability to belong to the classes.

It is also interesting to see what happen if we calculate the output of trained MLPs for different input intervals:

[1] All simulations have been done with SNNS (Zell, (1995)), using Scaled Conjugate Gradient method.

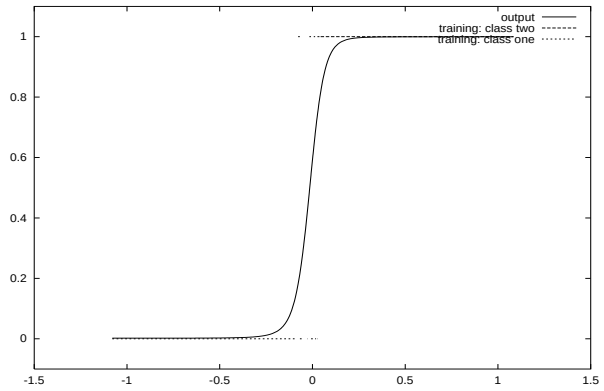

Fig. 1. Output of the MLP trained with the simulation approach

Input	mean output	interval output for mean	simulation output
$[-0.1, 0.1]$	1.00	$[0, 1]$	$[0.12, 0.95]$
$[-0.1, 0]$	0.00	$[0, 1]$	$[0.12, 0.59]$
$[-0.2, 0]$	0.00	$[0, 1]$	$[0.022, 0.59]$
$[-0.3, -0.1]$	0.00	$[0, 0]$	$[0.007, 0.12]$

It is quite clear that results provided by the simulation method are more interesting than results provided by the mean method. For the mean method, interval outputs are quite useless, even with a short interval as $[-0.1, 0]$ which cannot be classified. Moreover, the mean method classifies $[-0.1, 0.1]$ in the second class, which is not a good result. The simulation method give interesting intervals. For $[-0.1, 0.1]$, we obtain a quite broad interval which shows that it is quite difficult (and in fact meaningless) to try to classify this interval. For $[-0.1, 0]$ we obtain a wide interval, but closer to 0 than 1, therefore we can classify this interval in the first class, but the wide result shows that there is still a high probability that $[-0.1, 0]$ corresponds to a member of class two. Results obtained for other inputs show also that the simulation based approach gives more realistic results than the mean approach.

4 Comparison of simulation method and extremal values method

4.1 Theoretical comparison

The simulation method handles a n interval input with n input neurons, whereas the extremal values method needs $2n$ input neurons. Assume that we want to use a MLP with p hidden neurons and one output. Then, we have exactly $(n + 2)p + 1$ numerical parameters for the simulation based method. If we translate n intervals into $2n$ real values, we must use $(2n + 2)p + 1$ parameters. Therefore, if we have a fixed number of training patterns,

we must use less hidden neurons for the extremal values method than for the simulation method, in order to obtain the same estimation quality for parameters. Obviously, this reduces the computing power of the extremal values method as illustrated in section 4.2.

Another drawback of the extremal values method is that it cannot be extended easily to arbitrary probabilistic inputs. It can obviously be applied to parametric probabilistic inputs (for instance Gaussian distributed inputs), but not to histogram inputs (as long as the number of bin is not constant).

The extremal values method has two important advantages over the simulation method. First of all, if we use roughly the same number of parameters, the training time of the simulation method is in general longer than the one of extremal values method, simply because the former uses much more examples then the latter. Moreover, in some cases, the extremal values method can classify examples than cannot be separated by the simulation method, as will be demonstrated in section 4.3.

4.2 XOR problem

As explained in previous section, extremal values method uses more numerical parameters than probabilistic methods. Let us consider the case of two dimensional interval-valued inputs. With the probabilistic methods, a MLP with two hidden neurons and one output uses 9 parameters. With the extremal values method, it uses 13 parameters (and 7 with only one neuron).

Let us consider an interval version of the XOR problem. We have four training examples, centered on $(-1,-1)$ and $(1,1)$ for the first class, and on $(-1,1)$ and $(1,-1)$ for the second class. We assume that measurement errors replace the exact values by intervals of length 0.2. It is well known that we need at least 2 hidden neurons to solve the XOR problem. The interesting point is that with the probabilistic methods, this can be done with 9 parameters, whereas we need 13 parameters for the extremal values approach.

Of course, experiments confirm this discussion (for the simulation approach, we use 10 simulated examples for each original example):

hidden neurons	method	Mean Square Error
2	extremal values	0.0
2	simulation	0.0
1	extremal values	0.17
1	simulation	0.17

4.3 Overlapping individuals

We consider a very simple two classes problem. The unique example of class one is described by the interval $[-0.5, 0.5]$, and the unique example of class two is described by the interval $[-1, 1]$. With the extremal values approach,

we can use one hidden neuron to exactly classify both examples. With the probabilistic methods, the situation is far less satisfactory. Obviously, the mean method is not usable because both intervals have the same mean. For the simulation based method, we trained a MLP with two hidden neurons. Of course, we cannot correctly classify inputs that belongs to $[-0.5, 0.5]$. In fact, if we assume that simulated points are chosen uniformly in each interval and are in equal proportion, the probability that an input belongs to class one knowing that we observe a value in $[-0.5, 0.5]$ is $\frac{2}{3}$. The trained MLP agrees with this number and therefore misclassifies one third of the inputs.

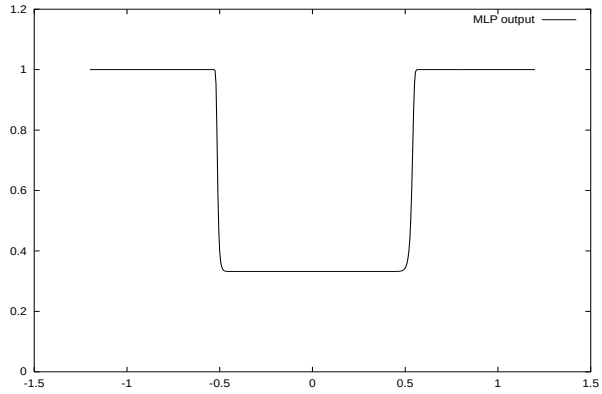

Fig. 2. Output of a MLP trained with the simulation method

Figure 2 gives that output of the trained MLP as a function of its input. As expected, the output is an approximation of the posterior probability of class two knowing the observed value. Therefore, when we submit a new input to the MLP, it gives a sound result. But if we input an interval with the simulation approach, we obtain less useful results. For instance, the output of the MLP for $[-1, 1]$ is the interval $[\frac{1}{3}, 1]$, whereas for $[-0.5, 0.5]$, we obtain approximately $[\frac{1}{3}, \frac{1}{3}]$. It is quite difficult to use this kind of result.

If we use now the extremal values MLP to classify new inputs, we also obtain unsatisfactory results which are summarized on figure 3. Any numerical input considered as a zero length interval is in fact classified into class one.

To summarize this simple numerical experiment, we have quite different results with the two proposed methods. Extremal values method gives good results on interval-valued inputs but cannot be use at all for numerical inputs (it performs exactly as a random classifier). On the contrary, simulation based method gives very useful results for numerical inputs, but cannot classify correctly interval-valued inputs.

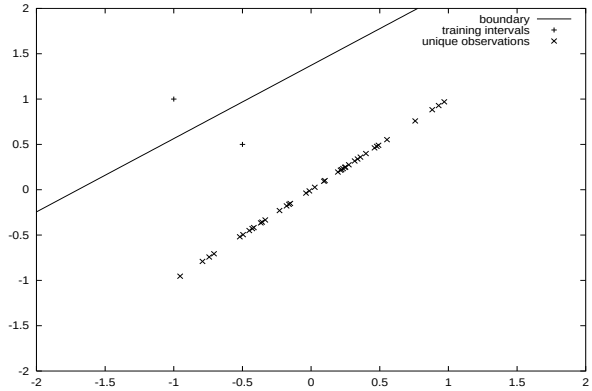

Fig. 3. Classification results for a MLP trained with extremal values

5 Conclusion and future work

We have presented in this paper three methods that can be used to process interval-valued inputs with multi-layer perceptrons. The mean method is obviously a limited method which should be avoided as the simulation method provides in general better results (with an increased training time). Comparing the extremal values method and the simulation method is more difficult. Whereas the extremal values method seems to perform better on interval-valued inputs, it cannot be generalized to arbitrary probabilistic inputs. Moreover, using a MLP trained with the extremal values method to classify new real valued inputs can give very incorrect results. Therefore, we recommend to use both methods together in order to add their respective qualities. We are currently implementing those methods for the ASSO (Analysis System of Symbolic Official data) European IST Project (see http://www.info.fundp.ac.be/asso/).

References

Beheshti, M., Berrached, A., de Korvin, A., Hu, C., and Sirisaengtaksin, O. (1998). On interval weighted three-layer neural networks. In *Proceedings of the 31 Annual Simulation Symposium*, pages 188–194. IEEE Computer Society Press.

Bishop, C. (1995a). *Neural Networks for Pattern Recognition*. Oxford Press.

Bishop, C. (1995b). Training with noise is equivalent to Tikhonov regularization. *Neural Computation*, 7(1):108–116.

Bock, H.-H. and Diday, E., editors (2000). *Analysis of Symbolic Data. Exploratory methods for extracting statistical information from complex data*. Springer Verlag.

Grandvalet, Y., Canu, S., and Boucheron, S. (1997). Noise injection: Theoretical prospects. *Neural Computation*, 9(5):1093–1108.

Moore, R. (1966). *Interval Analysis*. Englewood Cliffs, New Jersey.

Sietsma, J. and Dow, R. (1991). Creating articial neural networks that generalize. *Neural Networks*, 4(1):67–79.

Simoff, S. J. (1996). Handling uncertainty in neural networks: An interval approach. In *Int. Conf. on Neural Networks*, pages 606–610, Washington. IEEE.

Šíma, J. (1995). Neural Expert Systems. *Neural Networks*, 8(2):261–271.

Zell, A. et al. (1995). *SNNS 4.1 user manual*. University of Stuttgart.