A Neural Network Toolbox for Electricity Consumption Forecasting

Jarosław Protasiewicz
National Information Processing Institute
Warsaw, Poland
jaroslaw.protasiewicz@opi.org.pl

***Abstract*—The aim of this study is to propose a neural toolbox for electricity consumption prediction. The toolkit covers the implementation of three artificial neural networks, namely: (i) a multi-layered perceptron network; (ii) a radial basis function network; and (iii) a self-organising map. Moreover, the toolbox includes tools known as metamodels, which allow easy use of these networks for forecasting. There are two prediction systems, namely: (i) serially connected models; and (ii) a two-levelled structure containing a classifier and a set of parallel models. They have been validated experimentally in the task of electricity consumption prediction. The results and flexibility of use suggest that the neural toolbox should help users to develop prediction systems of electricity consumption more conveniently, as it is designed for that particular purpose. The toolbox has been developed as open source - no commercial software is required to use it.**

***Index Terms*—neural networks, open source, electric energy**

I. Introduction

The use of artificial neural networks remains a vital technique for data clustering and classification, forecasting, identification, and control. There is an extensive collection of studies demonstrating the application of various types of neural network to time series prediction, including in electricity consumption forecasting. Neural networks are commonly used due to their ability to model non-linear relations between input variables.

Although much software can be found which implements neural networks, most of it is dedicated to general purposes and needs to be adjusted for any particular task. The most popular open source packages presently are Tensorflow [1], [2], Keras, and Weka [3], [4]. Their universality is an advantage because anyone can adapt the software to his/her needs, though this may also act as a drawback, as implementing any task requires substantial effort from the user. It was thus necessary to propose a neural software toolbox for time series prediction.

This study introduces a neural toolbox designated especially for time series prediction. The toolbox covers the implementation of three artificial neural networks, namely: (i) a multi-layered perceptron network; (ii) a radial basis function network; and (iii) a self-organising map. Moreover, it includes the tools known as metamodels, which allow easy use of these networks for forecasting. I propose two prediction systems, namely: (i) serially connected models and (ii) a two-levelled structure containing a classifier and a set of parallel models. The software is implemented in Java, meaning that it can easily be integrated with other software.

The remainder of the paper is structured as follows: Firstly, Section II briefly discusses the most popular open source neural software. Next, Section III offers a short description of the neural networks that are implemented in the toolbox, and includes the technical details of the toolbox. Next, Section IV demonstrates two exemplary applications of the toolbox for time series prediction. Finally, conclusions and references are included in Section V.

II. Brief Overview of Open Source Neural Software

Much effort has been made to produce neural network software in recent years. Whilst there is a multitude of commercial software on the market currently, this study focuses exclusively on open source software. Thus, a brief discussion of open source neural software, which is freely accessible on the internet, is warranted.

One of the earliest implementations is the Stuttgart Neural Network Simulator (SNNS), developed at the University of Stuttgart[1] [5]. Like the Fast Artificial Neural Network[2] - originally established at the University of Copenhagen - the SNNS is written in C language [6]. Some neural software is also developed in Java, notably the Java Neural Network Simulator,[3] (JavaNNS) [7] developed at the University of Tubingen.

The latest implementations focus primarily on deep learning methods. Deeplearning4j[4] [8] is neural software, produced using Java, which uses the ND4J computing library in C++. The main advantage of this tool is the possibility to distribute computation tasks on multiple nodes by using Apache Spark or Hadoop. UC Berkeley researchers have produced yet another deep learning tool, Caffe[5] [9]. It was developed using C++ and Python, the latter of which is becoming an increasingly popular programming language for machine learning solutions. Some of the flagship software packages include: (i) the open-source software library for Machine Intelligence[6] (Tensor-

[1]http://www.ra.cs.uni-tuebingen.de/SNNS
[2]http://leenissen.dk/fann/wp
[3]http://www.ra.cs.uni-tuebingen.de/software/JavaNNS
[4]https://deeplearning4j.org
[5]http://caffe.berkeleyvision.org
[6]https://www.tensorflow.org

Flow) [1], [2]; (ii) the Python Deep Learning library[7] (Keras); (iii) the Python library, Theano[8] developed at the University of Montreal [10]; (iv) a scientific computing toolbox; Torch[9] developed in C and Lua [11], [12]; and Weka[10] [3], [4].

It would be highly challenging to enumerate all neural libraries - many of them are outdated because they are rarely maintained or updated following initial release.

III. Neural Toolbox

This section covers the implementation details of the toolbox, i.e. algorithms, relations between main objects, and architecture[11].

A. Algorithms

The toolbox implements three artificial neural networks, namely (i) a multi-layered perceptron (MLP) network; (ii) a radial basis function (RBF) network; and a self-organising map (SOM). Since these neural networks are widely known, only crucial information is included in this paper to clarify what exactly the toolbox implements.

The MLP network is equipped with a backpropagation algorithm in a basic form and with an additional momentum component [13], [14]. In both cases, the learning rate can be either fixed or adaptive. The fast backpropagation algorithm with heuristic modifications proposed by Fahlman [15] is also utilised in this model. The network may be composed of perceptrons or ADALINE neurons organised in several layers. A neuron may have a linear, sigmoidal, or Fahlman activation function.

The RBF network includes both gradient and non-gradient training methods [16]. The weight of an output layer can by calculated by using a matrix pseudo-inversion method, or a gradient descent algorithm which works stochastically. The gradient approach also applies to the parameters of a radial layer, i.e. the positions of the centres and degrees of the centres' fuzziness. The number of radial functions must be drawn randomly. They may then be fine-tuned by a k-means algorithm if necessary. A function may be Gaussian, power, or glued.

The SOM is consistent with the Kohonen proposal [17]. In the toolbox, a map may be one, two, or three-dimensional. A neighbourhood measure between neurons can be rectangular, hexagonal, or Gaussian. It is possible to measure distances by using the maximum, Euclidean, or Manhattan metrics. During self-organisation, the learning rate can be decreased exponentially or linearly.

[7]https://keras.io

[8]http://deeplearning.net/software/theano

[9]http://torch.ch

[10]https://www.cs.waikato.ac.nz/ml/weka

[11]Please note that terms in italics, e.g. *'Neuron'*, *'Layer'*, or *'Network'*, refer to classes or their instances, i.e. objects, in the context of an object programming language. Conversely, the same terms in standard fonts, e.g. 'Neuron', 'Layer', or 'Network', relate to concepts applied in the field of artificial intelligence.

B. Main classes

The toolbox is implemented in Java. Since it is a fully object-oriented language, Java offers advanced modelling techniques of concepts and data. There are six primary object types, namely: *Function*, *Neuron*, *Layer*, *Network*, *Algorithm*, and *DataSet*. Figure 1 shows the relationships between each of them.

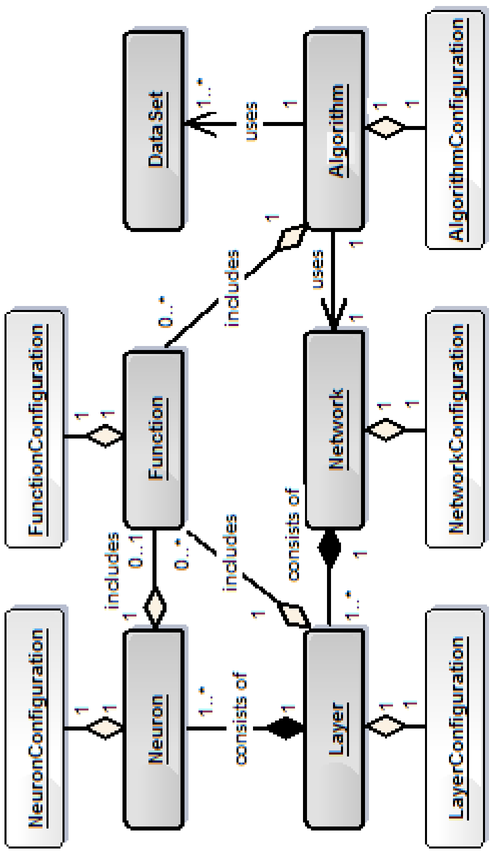

Fig. 1. A diagram showing the primary objects of the toolbox and the relationships between them. It complies with Unified Modeling Language 2.0 (UML), i.e. arrows indicate that one object uses another in some way; objects connected by an empty diamond indicate an inclusion; and objects connected by a filled diamond show a composition.

The most elementary component of the toolbox is *Function*, which can act as an activation function of a neuron; a method of centre selection; a distance metric; a neighbourhood measure; an error measure; or a learning rate. Functions are classes which provide uncomplicated methods for processing the signals of a neural network or measuring its parameters. They are therefore able to control training and simulation. All classes extend the basic abstract class, *Function* and inherit the ability to save or restore their state.

Additionally, *Neuron* is a fundamental processing element of the toolbox. It includes one *NeuronConfiguration* object, which defines neuron properties. It may include one *Function* object which implements its activation function. Neurons in an artificial neural network communicate with each other via their inputs and outputs, and they are unaware of anything more about themselves. The neuron classes in the toolbox satisfy these assumptions. Each neuron class extends the basic abstract class, *Neuron* and is concurrently able to calculate its outputs based on input signals.

Next, the *Layer* object consists of one or more *Neuron* objects. Practically, it is a layer or a matrix of artificial neurons. The layer is usually two-dimensional, whereas the matrix is able to be n-dimensional. Moreover, it includes exactly one *LayerConfiguration* object, and it might include one *Function* object. They determine a layer's topology and parameters. The *Layer* object organises the *Neuron* objects topologically. Hence, it is able to process signals concurrently. There are several layer types in the toolbox. Each of them extends the abstract basic class, *Layer*.

Then, the *Network* object is composed of one or more *Layers*. The classes of neural networks extend the abstract class, *Network*. The *Network* class is able to determine its outputs during simulation based on inputs. The *Network* class includes precisely one *NetworkConfiguration* object carrying its static properties, and one or more *Function* objects modelling its dynamic properties.

The *Network* uses the *Algorithm* object, which covers the implementation of a training algorithm. Its parameters are included in an *AlgorithmConfiguration* object. In addition, it may include one or more *Function* objects, which implement an algorithm's particular properties, e.g. changes in the learning rate or neighbourhood. Finally, the *Algorithm* object uses at least one *DataSet* object, which is responsible for serving data for training, validation, testing, and simulation. The training algorithms are separated from the elements of neural networks, which means that they are unaware of their own internal structure. However, they use objects such as *Function*, *DataSet*, and *Network* by executing their public methods. Each algorithm implemented in the toolbox extends the abstract class, *Algorithm*.

C. Architecture

The toolbox comprises a typical three-layered architecture: more specifically, it is composed of data, business, and interface layers (see Figure 2).

The data access layer contains a database, which stores trained artificial networks, data and predictions. They can be accessed as Value Objects through programming interfaces known as Data Access Objects (DAO). The DAOs are a group of classes suitable for data access and managing connections to the database. There are also API classes which provide more complex operations, e.g. storing forecasts or retrieving metamodels. Although the software currently works with the PostgreSQL[12] database, it can easily be adjusted to work with any database.

The business layer hosts all computational processes, including the training and simulation of neural networks; forecasting by using models and metamodels; and quality measurement (see the business layer in Figure 2). The model

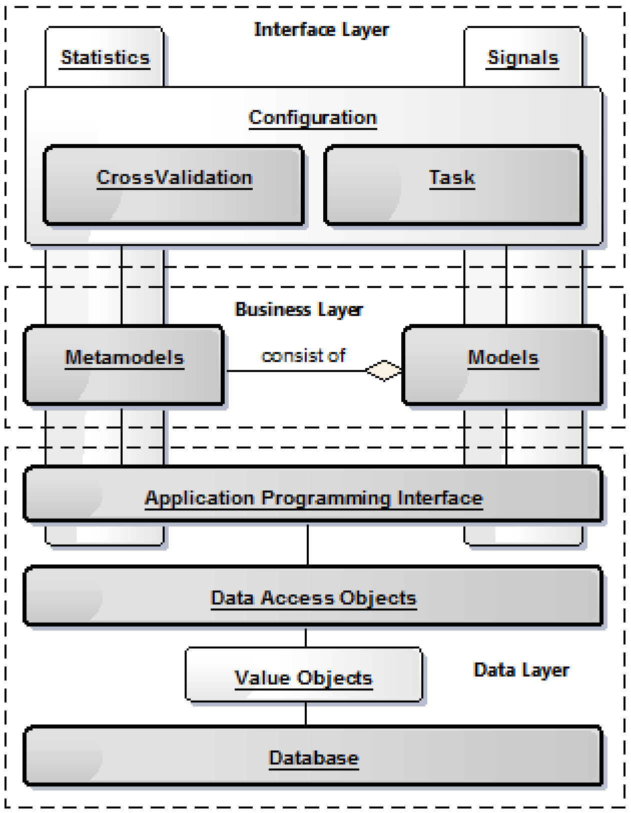

Fig. 2. An architecture of the toolbox.

corresponds to a neural network with its parameters. The model contains data for training and simulation of the neural network. It knows how to utilise the neural network and generate forecasts. Each model is based on the abstract class, *Model*. Moreover, a metamodel organises models into a forecasting system. It serves as a recipe, explaining how to use models to execute a required job. Such a system is able to generate better quality predictions than a single model, especially when considering longer-term predictions.

Both models and metamodels can be accessed from the interface layer. It covers two job types, namely: (i) a cross-validation suitable for training models; and (ii) a task, which primarily executes metamodels to obtain predictions. Both of them utilise configurations and produce either signals (forecasts) or statistics (performance indicators). Since the toolbox was designed to be incorporated easily into any software, it was unnecessary to equip it with a graphical interface. In its place, the toolbox contains dedicated classes. They constitute a programming interface, in which a user can define any task which is allowed by the software library. Naturally, its use requires some degree of programming skill from the user.

IV. Time Series Prediction

The section introduces two applications of the toolbox for time series prediction, namely: (i) a serial connection of MLP networks; and (ii) a two-levelled system composed of an SOM classifier and MLP or RBF networks [18].

[12]https://www.postgresql.org

A. Data analysis

The experiments were conducted on hourly electric energy consumption data from 2002 to 2004 in central Poland, excluding Warsaw (see data in Figure 3a).

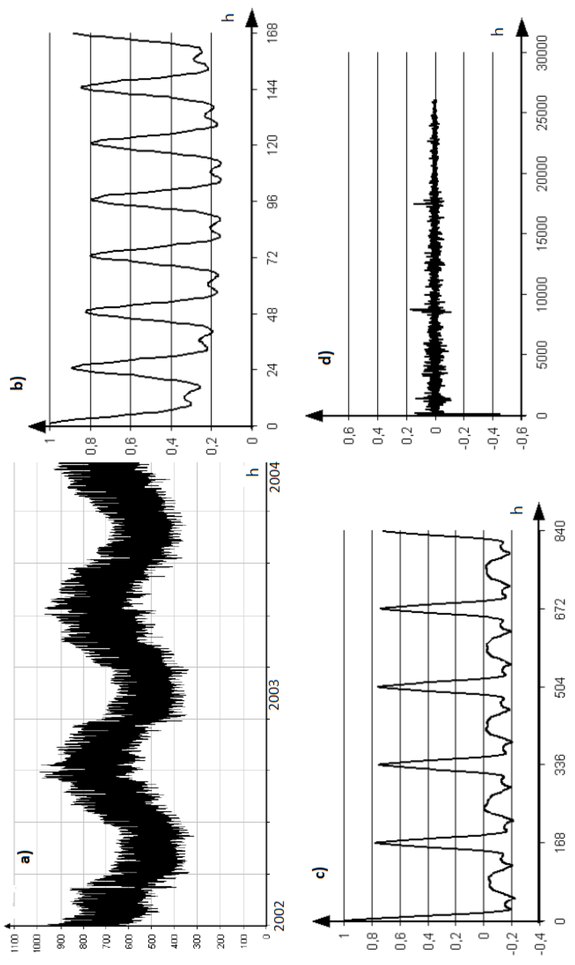

Fig. 3. Autocorrelation of time series of electric energy consumption: a) an original time series; b) daily autocorrelation; c) weekly autocorrelation; d) seasonal autocorrelation.

To propose an accurate prediction model, the time series under examination must be stationary. This can be checked by analysing its linear autocorrelation. Figures 3b and 3c show distinctive daily and weekly autocorrelation, whereas, Figure 3d depicts no seasonal autocorrelation. Thus, the electricity consumption is a stationary time series, and operates in daily and weekly cycles.

It is believed that meteorological factors may influence the electricity consumption of common consumers. To validate this assumption, correlations between the time series of electric energy and temperature, humidity, and insolation were calculated. The linear Pearson correlations are presented in Table I. Between the aforementioned weather factors and electricity consumption, a moderate correlation can be observed during summer, and a low correlation during spring and autumn.

year	r_{ET}			r_{EN}			r_{EW}		
month	2002	2003	2004	2002	2003	2004	2002	2003	2004
1	-0.27	-0.16	-0.01	0,10	0,08	0,12	0,03	-0,14	-0,23
2	0.02	**0.22**	-0.11	0,08	0,02	0,18	-0,17	-0,15	-0,07
3	0.09	-0.02	-0.09	0,04	0,05	0,12	-0,07	-0,12	**-0,21**
4	-0.01	**-0.20**	**0.20**	0,15	0,03	0,21	**-0,36**	0,15	**-0,25**
5	**0.45**	**0.40**	**0.34**	**0,37**	**0,40**	**0,37**	**-0,33**	**-0,37**	**-0,34**
6	**0.53**	**0.50**	**0.55**	**0,49**	**0,43**	**0,49**	**-0,48**	**-0,39**	**-0,54**
7	**0.46**	**0.55**	**0.44**	**0,42**	**0,51**	**0,48**	**-0,40**	**-0,54**	**-0,44**
8	**0.48**	**0.49**	**0.49**	**0,39**	**0,37**	**0,42**	**-0,45**	**-0,42**	**-0,38**
9	0.09	**0.34**	**0.25**	0,16	**0,20**	**0,21**	**-0,28**	**-0,34**	**-0,30**
10	-0.01	**-0.20**	0.07	0,11	0,11	0,13	-0,19	**-0,34**	**-0,33**
11	-0.06	0.08	-0.17	0,09	0,03	0,09	-0,16	-0,17	-0,18
12	-0.02	0.06	-0.06	0,10	0,05	0,09	-0,16	0,15	-0,11

TABLE I
LINEAR CORRELATION OF ELECTRIC ENERGY CONSUMPTION AND TEMPERATURE, HUMIDITY, AND INSOLATION IN MONTHS AND YEARS. RED INDICATES A MODEST CORRELATION; BLUE INDICATES A LOW CORRELATION; AND BLACK INDICATES NO CORRELATION.

Linear relationships may not explain the whole variability of electricity consumption. Additional tests, such as Spearman's rank correlation, may reveal previously hidden relationships. The results in Table II indicate that we cannot reject the hypothesis that rank correlations exist between weather factors and electricity consumption.

All of these tests suggest that a non-linear model, such as a neural network, is necessary to predict electric energy consumption. Each model may focus on particular days of the week, or on public holidays, such as Christmas Day or Easter Sunday, and may also be dependent on the data from a previous day.

B. Serial model

In this approach, several prediction models are connected serially. Each model represents a particular day type, and produces forecasts for the day ahead to which it corresponds. The models collectively form a metamodel, and their forecasts are logically joined to form long-term predictions. The first model in the series receives only historical data such as inputs; whereas each successive model utilises some of the data and forecasts produced by the previous model (Figure 4).

Figure 5 outlines the technical implementation of the algorithm. Initially, the models of all types are preliminarily trained using historical data, and are then stored in a database. Then,

year	ρET			ρEN			ρEW		
month	2002	2003	2004	2002	2003	2004	2002	2003	2004
1	**-0,26**	**-0,08**	-0,02	**0,19**	**0,18**	**0,18**	0,05	**-0,18**	**-0,23**
2	0,00	**0,22**	**-0,12**	**0,18**	**0,18**	**0,26**	**-0,17**	**-0,18**	-0,07
3	**0,16**	-0,01	**-0,12**	**0,19**	**0,19**	**0,24**	**-0,15**	**-0,14**	**-0,21**
4	-0,01	**-0,19**	**0,20**	**0,26**	**0,20**	**0,33**	**-0,37**	**0,10**	**-0,26**
5	**0,46**	**0,40**	**0,32**	**0,42**	**0,44**	**0,42**	**-0,30**	**-0,34**	**-0,33**
6	**0,56**	**0,50**	**0,53**	**0,52**	**0,47**	**0,51**	**-0,47**	**-0,37**	**-0,52**
7	**0,46**	**0,58**	**0,42**	**0,48**	**0,55**	**0,53**	**-0,38**	**-0,54**	**-0,43**
8	**0,49**	**0,48**	**0,51**	**0,45**	**0,44**	**0,50**	**-0,45**	**-0,41**	**-0,39**
9	0,06	**0,38**	**0,24**	**0,28**	**0,32**	**0,32**	**-0,28**	**-0,34**	**-0,32**
10	-0,04	**-0,22**	0,03	**0,21**	**0,20**	**0,23**	**-0,17**	**-0,34**	**-0,36**
11	-0,06	**0,09**	**-0,19**	**0,15**	**0,14**	**0,17**	**-0,18**	**-0,19**	**-0,13**
12	-0,05	0,04	**-0,08**	**0,14**	**0,15**	**0,17**	**-0,15**	**0,17**	**-0,13**

TABLE II
RANK CORRELATION OF ELECTRIC ENERGY CONSUMPTION AND TEMPERATURE, HUMIDITY, AND INSOLATION IN MONTHS AND YEARS. BOLD NUMBERS INDICATE THE PRESENCE OF SUCH A CORRELATION.

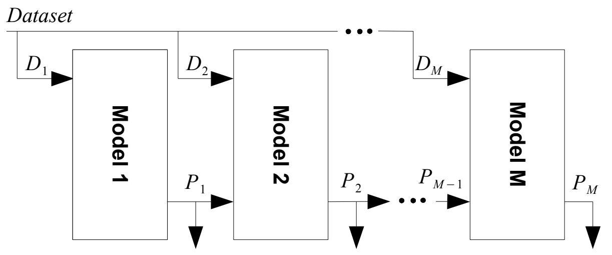

Fig. 4. A serial meta-model (D - a training dataset; P - forecasts).

the appropriate models are selected for the prediction period. They are fine-tuned making use of additional data. Finally, the models are ready to produce forecasts which are connected serially as outputs for each of the models.

An exemplary application of this approach is in the prediction of electric energy consumption based on autoregressive inputs, e.g. historical electric energy consumption, and regressive inputs such as temperature, humidity and insolation. Here, each model corresponds to a particular day type, and predicts for one day ahead. For instance, there may be separate models for each day of the week and for each public holiday. Thus, the position of the models in the series depends on a calendar. Figure 6 presents the distribution of percentage errors of electricity consumption forecasts for two days ahead in one

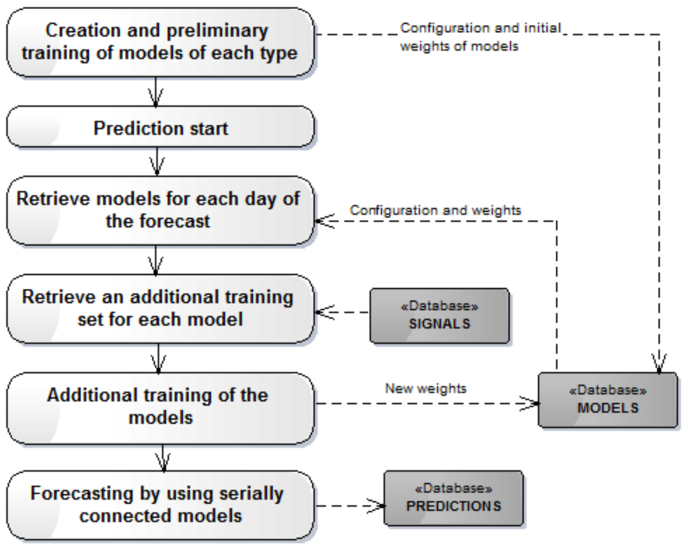
Creation and preliminary training of models of each type
Configuration and initial weights of models
Prediction start
Retrieve models for each day of the forecast
Configuration and weights
Retrieve an additional training set for each model
«Database» SIGNALS
Additional training of the models
New weights
«Database» MODELS
Forecasting by using serially connected models
«Database» PREDICTIONS

Fig. 5. Serial metamodel in practice.

year. The forecasts originate from eight serial metamodels, each dependent on different input parameters.

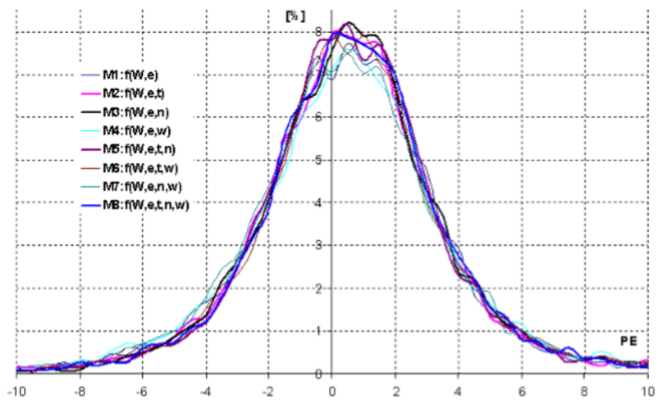

Fig. 6. The distribution of percentage errors of electricity consumption forecasts produced by exemplary serial meta-models (W - a neural network, e - electric energy consumption, t - temperature, n - insolation, w - humidity, PE - percentage error).

C. Parallel model

The second application of the toolbox is a parallel meta-model which contains a classifier at the first level, and a set of prediction models at the second. The classifier decides which model has to make predictions for a particular time in the future. In this way, a long-term forecast can be produced by well-adjusted prediction models (Figure 7).

In practice, the classifier is based on a self-organising network; whereas, the prediction models can be implemented either as MLP networks or RBF networks. The classifier clusters input data such as electricity consumption and weather factors. These clusters act as training sets for prediction models. Due to the relatively small cluster size, the models predict only a few hours ahead. That is less than in the serial

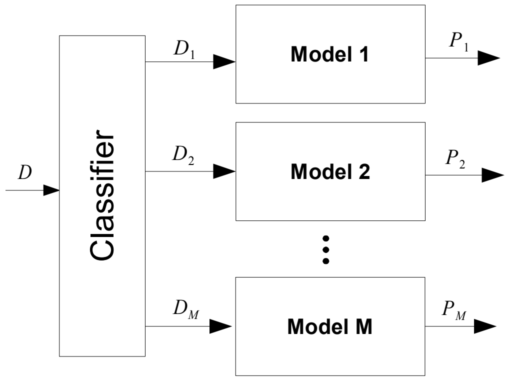

Fig. 7. A parallel metamodel (D - a training dataset; P - forecasts).

metamodel. Since the classifier organises data and models, the parallel metamodel is able to work without the knowledge of experts on the time series under investigation.

Figure 8 includes exemplary forecasts for one and two days ahead. We can easily notice that the predictions mirror real consumption. However, there are some imperfections in cases in which the data changes rapidly.

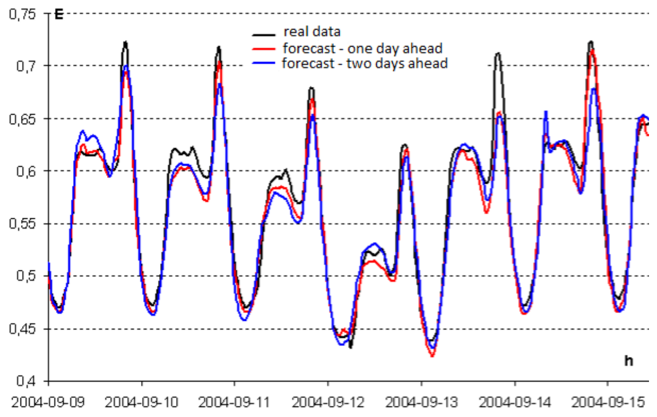

Fig. 8. The forecasts of electricity consumption for one and two days ahead.

V. Conclusion

This paper has presented the neural software that, rather than applying universally, is dedicated especially to time series prediction, in particular for the forecasting of electric energy consumption. Such an approach should help to develop prediction systems rapidly.

The toolbox was developed with the use of non-commercial tools. As open source software, it is available for everyone for free on a remote repository[13]. It can be executed on any operating system which contains a Java Virtual Machine. Further development should focus on introducing deep learning algorithms, as well as providing more forecasting metamodels.

The toolbox was developed in 2008 as a part of the writer's Ph.D. [18], and until now has never been released. Nowadays, it is evident that rapid development of open source neural software is occurring. With this in mind, the toolbox has now been made available publicly. The toolbox was an advanced piece of software in 2008: now, it is unlikely to compete with modern deep learning tools. However, this publication is the first step towards upgrading the toolbox to an automatic tool for time series prediction. This may involve automatic analysis of time series to construct forecasting models, and upgrading the toolbox to process modern neuronal structures.

[13]https://github.com/Jaroslaw-Protasiewicz/pythia

References

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin *et al.*, "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," *arXiv preprint arXiv:1603.04467*, 2016.

[2] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard *et al.*, "Tensorflow: A system for large-scale machine learning." in *OSDI*, vol. 16, 2016, pp. 265–283.

[3] E. Frank, M. A. Hall, and I. H. Witten, *The WEKA Workbench. Online Appendix for Ḋata Mining: Practical Machine Learning Tools and Techniques¨.* Morgan Kaufmann, Fourth Edition, 2016.

[4] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The weka data mining software: an update," *ACM SIGKDD explorations newsletter*, vol. 11, no. 1, pp. 10–18, 2009.

[5] A. Zell, N. Mache, R. Hübner, G. Mamier, M. Vogt, M. Schmalzl, and K.-U. Herrmann, *SNNS (Stuttgart Neural Network Simulator)*. Boston, MA: Springer US, 1994, pp. 165–186. [Online]. Available: http://dx.doi.org/10.1007/978-1-4615-2736-7

[6] S. Nissen, "Implementation of a fast artificial neural network library (fann)," *Report, Department of Computer Science University of Copenhagen (DIKU)*, vol. 31, p. 29, 2003.

[7] I. Fischer, F. Hennecke, C. Bannes, and A. Zell, "Javanns: Java neural network simulator," *User Manual, Version*, vol. 1, 2006.

[8] D. Team *et al.*, "Deeplearning4j: Open-source distributed deep learning for the jvm," *Apache Software Foundation License*, vol. 2, 2016.

[9] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," *arXiv preprint arXiv:1408.5093*, 2014.

[10] Theano Development Team, "Theano: A Python framework for fast computation of mathematical expressions," *arXiv e-prints*, vol. abs/1605.02688, May 2016. [Online]. Available: http://arxiv.org/abs/1605.02688

[11] R. Collobert, K. Kavukcuoglu, and C. Farabet, "Torch7: A matlab-like environment for machine learning," in *BigLearn, NIPS workshop*, no. EPFL-CONF-192376, 2011.

[12] R. Collobert, S. Bengio, and J. Mariéthoz, "Torch: a modular machine learning software library," Idiap, Tech. Rep., 2002.

[13] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Parallel distributed processing: Explorations in the microstructure of cognition, vol. 1," D. E. Rumelhart, J. L. McClelland, and C. PDP Research Group, Eds. Cambridge, MA, USA: MIT Press, 1986, ch. Learning Internal Representations by Error Propagation, pp. 318–362. [Online]. Available: http://dl.acm.org/citation.cfm?id=104279.104293

[14] ——, "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 6088, pp. 533–536, 1986. [Online]. Available: http://dx.doi.org/10.1038/323533a0

[15] S. E. Fahlman, "An empirical study of learning speed in back-propagation networks," Tech. Rep., 1988.

[16] D. S. Broomhead and D. Lowe, "Radial basis functions, multi-variable functional interpolation and adaptive networks," Royal Signals and Radar Establishment Malvern (United Kingdom), Tech. Rep., 1988.

[17] T. Kohonen, "Essentials of the self-organizing map," *Neural Netw.*, vol. 37, pp. 52–65, Jan. 2013. [Online]. Available: http://dx.doi.org/10.1016/j.neunet.2012.09.018

[18] J. Protasiewicz and P. S. Szczepaniak, "Neural models of demands for electricity-prediction and risk assessment," *Electrical Review*, vol. 88, no. 6, pp. 272–279, 2012.