

FAST SIMULATION AND OPTIMIZATION WITH NEURAL NETWORKS

Friedrich Biegler-König
Fachbereich Mathematik und Technik
Fachhochschule Bielefeld
33604 Bielefeld, Germany
E-mail: friedrich.biegler-koenig@fh-bielefeld.de

Peter Deeskow
STEAG KETEK IT GmbH
Centroallee 261
46047 Oberhausen, Germany
E-mail: peter.deeskow@steag-ketek.de

KEYWORDS

Neural Networks, Real-Time Simulation, Power Plant Optimisation, Simulated Annealing

ABSTRACT

Neural Networks can be powerful tools for the approximation of complex nonlinear behaviour. After a learning phase neural networks are able to predict results with high accuracy. While the learning phase might be time consuming, prediction is very fast. After substituting complex simulation programmes by Neural Networks, these can be used in real-time operation, e.g. for optimisation processes. We demonstrate this approach by modelling the operation of a multi-block power plant. Afterwards operation can be optimised using any mixed integer optimisation algorithm, in our case Simulated Annealing.

INTRODUCTION

Essentially no complex technical device is built today without previous software-simulation. In many cases even ab initio models which are directly based on the law of physics can be employed.

However, these models which are used in the design process of industrial plants are often less usable in everyday business.

In everyday business the configuration of a plant is rarely changed but the personnel has to react quickly to changes of external parameters in order to get an optimal production output. Simulation with models which were used when designing the plant are usually too slow to meet the requirements of real time production.

In many cases simplified models are constructed to allow real time operation. Linear, piecewise linear, or polynomial fits to complex functions are often employed as well as characteristic curves for technical components.

These approaches work satisfactorily if there are only very few influencing variables (one or two) and if the

behaviour of the fitted functions is similar to the used fits (linear, quadratic, ...).

In this paper we present neural networks as a universal method to replace complex nonlinear models of several variables. The type of neural network we are describing can be used for any continuous differentiable function. They produce a very high accuracy of prediction and are not bound to a special behaviour of the original model.

In the next chapter we will give a short introduction into neural networks. Then we will apply this technique to a complex technical plant, and illustrate the possibilities opened by the neural network approximation.

NEURAL NETWORKS

Neurons in Computer Science are simple computational units. They receive inputs from outside or from other neurons, calculate a response, and send it on to other neurons or the outside. The connections between Neurons are also called synapses (with reference to the biological model). The strength of a connection is determined by its "synaptic weight".

In Computer Science a great number of possible neurons and connection types have been examined. For practical applications the so called feed-forward model is the one most widely used. Here the neurons are arranged in layers. Each neuron gets inputs from all neurons of the previous layer or (if it is the first one) from the outside and sends its output to all neurons of the following layer or (if it is the last one) the outside. Hence these networks always have an input and an output layer and a number of "hidden layers".

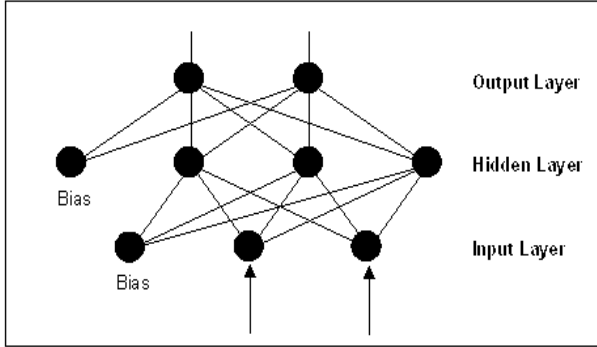
The neurons of this network type usually take the weighted sum of their inputs and apply an "activation function" to the result. Let i_1, \dots, i_n be the input values of a neuron, and w_1, \dots, w_n be the synaptic weights on the incoming connections. Then the output value of the neuron is calculated as:

$$a = \sigma(\text{net}) = \sigma\left(\sum_{k=1}^n w_k \cdot i_k\right) \quad (1)$$

with a so-called Sigmoid function

$$\sigma(x) = \frac{1}{1 + e^{-x}}. \quad (2)$$

In most cases only one hidden layer is used. Hecht-Nielsen proved the theoretical result that any analytical function can be approximated by a network with only three layers (Hecht-Nielsen 1987) (see figure 1).



Figures 1: Neural Network with 3 Layers

Learning

A neural network learns, if its synaptic weights are determined in a way that for given input values also given output values are generated. This is called “supervised learning“.

The synaptic weights are determined using a “learning algorithm“. The set of given input- and output-values is called “learning set“.

The best known learning algorithm is called Backpropagation (Rumelhart et. al. 1988). It minimizes the so-called energy function using a gradient descent scheme:

$$E = \frac{1}{2} \sum_{k=1}^N \sum_{j=1}^m (o_j^k - t_j^k)^2, \quad (3)$$

where N is the number of examples, m the number of neurons in the output layer, and o_j^k the output of the network in neuron j for example k . t_j^k is the wanted output value.

For neural networks with only one hidden layer there are other, sometimes more efficient learning algorithms (Bärmann and Biegler-König, 1992).

Prediction

It is the purpose of the learning process is to enable the Neural Network to predict output values for such examples which are not in the learning set.

In order to verify the quality of the learning process one keeps another set of examples, called the validation set.

This set is disjoint from the learning set, but also contains the correct output values which are compared to the predictions of the network.

Coding

Output values of Neural Networks with Sigmoid functions are always contained in the interval $(0, 1)$. In order to get a normalization, this is also desirable for the input values.

Before learning a learning set, its examples have to be transformed into values between 0 and 1. This coding has to be invertible to transform the net predictions into uncoded form.

In many cases a linear coding of the example sets is sufficient, but sometimes prediction quality can be enhanced by choosing a suitable more complex coding.

Measuring Errors

To quantify the prediction capacity of a network, we have to define a learning error (for the learning set) and a prediction error (for the validation set). Usually the Mean Squared Error is used for this purpose:

$$MSE = \frac{1}{N} \sum_{k=1}^N \sum_{j=1}^m (o_j^k - t_j^k)^2 \quad (4)$$

Another error measurement is provided by the empirical correlation coefficient:

$$EKK = \frac{\sum_{k,j} (o_j^k - o_j) \cdot (t_j^k - t_j)}{\sqrt{\sum_{k,j} (o_j^k - o_j)^2 \cdot (t_j^k - t_j)^2}}, \quad (5)$$

where o_j denotes the mean value of the outputs in neuron j and t_j the mean value of the corresponding exact values. If all examples in an example set are predicted correctly, the MSE will vanish and the EKK will be 1.

FAST SIMULATION OF POWER PLANTS

An example of the need of fast simulation can be found in the optimisation of power plants. We are considering a power plant with four blocks which are identical in construction (a simple case). Each block can generate power in the range of 108 MW to 360 MW. A block cannot produce less than 108 MW. If a block is switched off, there are considerable costs to start it up again.

A detailed model of a power plant block has been built using Epsilon, a programme developed by the company Sofbid (<http://www.sofbid.com/epsilon/>, see Brinkmann and Pawellek 2003 and 2004, and Brinkmann 2003) in Zwingenberg, Germany. Epsilon is a simulator specialized in power generating facilities. Epsilon

models can be very complex (see Figure 2 for a section of a model of a power station block). After the model has been constructed, Ebsilon needs about 5 to 20 seconds on a PC to simulate a given situation for one block. This is by far not fast enough e.g. for a complex real-time optimisation which usually needs many thousand evaluations.

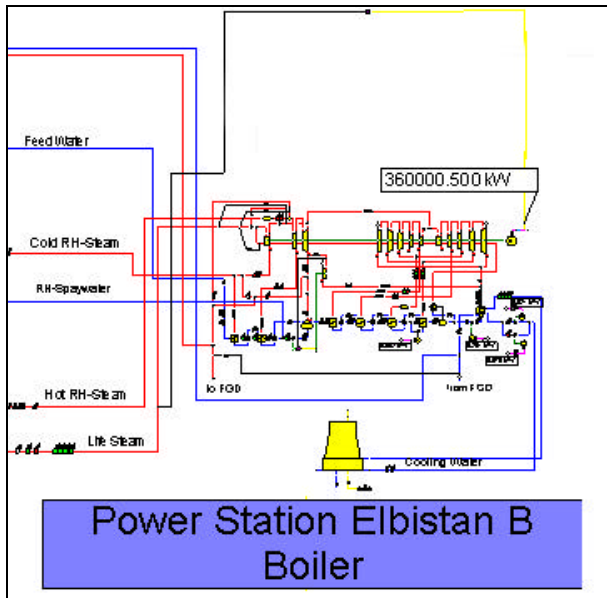


Figure 2: Section of Ebsilon Power Station Model

A specific situation of the power plant block is determined by only a few input parameter. The most important parameters is the amount of energy which the block is to produce per hour.

The other parameters describe the external conditions and settings by the operational staff:

- Temperature of cooling water (between 0 and 30 °C).
- Air ratio in combustion chamber (between 1.1 and 1.4).
- Live steam temperature (between 510 and 540 °C)
- Hot reheat temperature (between 510 and 540 °C)
- Flue gas recirculation (between 0 and 50 kg/s).
- 8 more parameters specifying the degree of heat surface fouling.

The main result Ebsilon is supplying is the amount of coal needed. This in turn determines the main part of the total costs.

In a first experiment 15000 random input data sets were produced, each set containing 14 randomly determined values for the input parameter. A data set describes a specific situation of a power plant block. Employing Ebsilon, for each data set the required amount of coal is calculated. This may take a long time.

We now have a set of 15000 data sets with 14 input and one output value. We divided this set into a learning set (10000 data sets) and a validation set (5000 data sets).

After coding the examples (standard linear coding) and deciding on a number of hidden layer neurons (60), we can try to teach the learning set to a neural network. The learning algorithm converges quickly and yields a MSE of 0.00009 and an EKK of 0.99901 after a few hundred iterations.

After decoding the examples, our network is able to predict the required amount of coal for a given situation with an average error of 0.6%. This error has the same order of magnitude as the simulation error of an Ebsilon calculation. The following is a typical example from the validation set.

Input Values:

| | |
|----------------------------------|---|
| Power requirement: | 317801 kW |
| Temperature of cooling water: | 5.583 °C |
| Air ratio: | 1.347 |
| Live steam temperature: | 522.517 °C |
| Hot reheat temperature: | 513.577 °C |
| Flue gas recirculation: | 38.055 kg/s |
| Heat surface fouling parameters: | 1.390, 1.09, 1.332, 1.217, 1.108, 0.801, 0.909, 1.269 |

Output Values:

| | |
|--|-------------|
| Amount of coal needed (Ebsilon calculation): | 633833 kg/h |
| Amount of coal needed (Network prediction): | 636844 kg/h |
| Absolute error: | 3011 |
| Relative error: | 0.475% |

The main advantage of our neural network is the response time. It is more than 5000 times faster than Ebsilon (response time: less than 0.001 seconds).

A SIMPLE CASE OF POWER PLANT OPTIMISATION

We will now consider a first and comparatively simple case of optimisation:

Usually a request for a certain amount of power is communicated to the power plant. In order to minimize the costs, we wish to find the least expensive configuration of blocks meeting this requirement.

This is a mixed integer optimisation problem. It contains an integer part and a continuous part. The integer part determines which blocks are switched off, and which are running. The continuous part determines the distribution of the requirement among the running power station blocks.

This type of problem has been solved by fitting a linear curve to the coal-power function and employing techniques from linear programming (Schultz 2003). This is no longer possible if we take all 14 input values into account.

We used a simulated annealing method (Gerdes et al. 2004; Nolle et al. 2001) to compute the global optimum

of our problem. This heuristic optimisation method turned out to be very reliable, but always needed between 1000 and 3000 evaluations to find the global optimum (clearly not feasible directly with Epsilon in a real-time environment).

We also used a standard optimisation algorithm, an SQP scheme ("Successive Quadratic Programming", see e.g. Bazaraa 1993). These algorithms have to be applied to each possible block-configuration, since they only optimise the continuous part of the problem. Furthermore, there are several minima for each block-configuration. In order to find the global optimum, we have to apply the SQP-scheme with many sets of starting values.

In the end, the SQP-scheme needs about 3000 evaluations to find the same optimum as the simulated annealing process. In a more complex situation, simulated annealing would clearly be superior.

SUMMARY AND FUTURE PROSPECTS

Approximating the behaviour of complex plants with neural networks turned out to be simple and easy-to-use method. These neural networks can replace elaborate simulation programmes in cases where several thousand evaluations are needed and real time results are required. This is, for instance, true in power plant optimisation for which a simple case has been described here.

In future we plan to apply optimisation algorithms to time series of power plant operations with a 24 hour advance planning. In these models, among others, start up costs and standstill periods have to be incorporated.

REFERENCES

- Hecht-Nielsen, R. 1987. "Kornolgorov's Mapping Neural Network Existence Theorem". *IEEE 1987*. San Diego, Vol. II.
- Rumelhart, D. et al. 1988. *Parallel distributed processing*. MIT Press.
- Bärnann, F. and Biegler-König, F. 1992. "On a Class of Efficient Learning Algorithms for Neural Networks". *Neural Networks* 5, 1992.
- Brinkmann, K. and Pawellek, R. 2003. "Epsilon - Examples for the easier design and better operation of power plants". Published at the conference "Energy Forum 2003" at Sv. Konstantin Varna.
- Brinkmann, K. and Pawellek, R. 2004. "Optimierte Prozessführung von Kraftwerksblöcken mit Online-Werkzeugen: Betriebserfahrungen". VDI Tagung "Wissensbasiertes Betriebsmanagement senkt Kosten", Frimmersdorf.
- Brinkmann, K. 2003. "Epsilon-EPOS - Beispiele für eine einfachere Auslegung und einen besseren Betrieb von Kraftwerken", 3. VDI Symposium Sao Paulo.
- Schultz. 2003. "Integer Programming Applied to Power Systems' Generation Operating Planning", Course Material.
- Gerdes, I.; Klawonn, F.; Kruse, R. 2004. *Evolutionäre Algorithmen*. Vieweg Verlag.
- Nolle, L.; Goodyear, A.; Hopgood, A.; Picton, P.; Braithwaite, N. 2001. "On Step Width Adaptation in Simulated Annealing for Continuous Parameter Optimisation", *Fuzzy Days 2001*, 589-598.
- Bazaraa, M. S. 1993, *Nonlinear Programming*, Wiley.