

Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

DIPLOMOVÁ PRÁCE



Petr Paščenko

Computational intelligence models for hydrological predictions

Katedra teoretické informatiky a matematické logiky

Vedoucí diplomové práce: *Mgr. Roman Neruda, CSc.*

Studijní program: *Informatika, Teoretická informatika*

2009

Rád bych poděkoval doktoru Romanu Nerudovi za všestrannou pomoc při výzkumu i zpracování diplomové práce až do úplného konce. Dále děkuji Českému hydrologickému ústavu, Ústí nad Labem za poskytnutí hydrologických dat. Konečně se cítím velmi zavázán pracovišti EPCC Department of Edinburgh University za možnost účastnit se stáže v rámci HPC Europa 2, grantového programu EK.

Prohlašuji, že jsem svou diplomovou práci napsal samostatně a výhradně s použitím citovaných pramenů. Souhlasím se zapůjčováním práce.

V Praze dne

Petr Paščenko

Název práce: Modely výpočetní inteligence pro hydrologické predikce

Autor: Petr Paščenko

Katedra: Katedra teoretické informatiky a matematické logiky

Vedoucí diplomové práce: Mgr. Roman Neruda, CSc.

e-mail vedoucího: roman.neruda@cs.cas.cz

Abstrakt: Tato práce se zabývá možnostmi využití metod výpočetní umělé inteligence v oblasti hydrologických předpovědí. Praktická studie problému krátkodobé predikce průtoku na základě dešťových srážek je provedena na skutečných fyzikálních datech popisujících relevantní časové řady naměřené v povodí řeky Ploučnice. Zevrubná statistická studie zahrnující korelační a regresní analýzu prokázala vysoký rozptyl naměřených hodnot. Pro konstrukci vstupního filtru pro neuronové modely byl proveden evoluční experiment.

V hlavní části práce bylo prozkoumáno několik modelů neuronových sítí založených na vícevrstevném perceptronu, sítích typu RBF a neuroevoluci společně se dvěma ansámblovými modely inspirovanými tzv. baggingem. Výsledné modely byly pečlivě testovány na datech pokrývajících letní období tří po sobě následujících let.

Bylo prokázáno, že modely založené na vícevrstevném perceptronu vykazují větší schopnost generalizace. Výsledné perceptronové modely jsou schopny snížit kvadratickou chybu předpovědi o zhruba 15% v porovnání s konzervativní predikcí současnou hodnotou.

Klíčová slova: hydrologické predikce, AI, ANN, evoluce

Title: Computational intelligence models for hydrological predictions

Author: Petr Paščenko

Department: Department of Theoretical Computer Science and Mathematical Logic

Supervisor: Mgr. Roman Neruda, CSc.

Supervisor's e-mail: roman.neruda@cs.cas.cz

Abstract: The thesis deals with the application of computational artificial intelligence models on hydrological predictions. The short term rainfall-runoff prediction problem is studied on the real data of physical time series measured in the watershed of river Plučnice. A brief statistical study including correlation and regression analyses is performed. The high level of variance and noise is concluded. The evolution of the proper input filter providing an input set for the neural network is performed.

In the main part of the thesis several neural network models based on multilayer perceptron, RBF units, and neuroevolution are constructed together with two neural ensembles inspired by the bagging method. The models are tested on the three subsequent years summer data.

The greater generalization ability of multilayer perceptron architectures is concluded. The resulting multilayer perceptron models are able to reduce the mean squared error of the prediction by 15% compared to the prediction by the previous value.

Keywords: hydrological predictions, AI, ANN, evolution

Contents

1	Introduction	6
2	Computational Intelligence in Hydrology	8
2.1	Artificial Neural Networks in Hydrology	8
2.1.1	Neural Networks Minimum	9
2.1.2	Introduction to Genetic Algorithms	11
2.1.3	Neural Ensembles	14
2.1.4	Related Work	15
3	Source Data	17
3.1	Time series	18
3.1.1	Runoff at the Mimoň station	18
3.1.2	Weather	19
3.1.3	UPS	19
3.2	Runoff prediction	21
3.3	Correlation analysis	21
3.3.1	Previous flow and watershed saturation	22
3.3.2	Rainfall	22
3.3.3	Temperatures	22
3.4	Regression analysis	23
3.4.1	Input data	23
3.4.2	Model composition	25
3.4.3	Results	25
4	Computational intelligence models	28
4.1	Input Filter Evolution	28
4.1.1	Problem definition	28
4.1.2	Filter Evolution	29
4.1.3	Experiment	32
4.1.4	Results	34
4.1.5	Defocusing Level of Detail Filter	34
4.2	Multilayer perceptron	35
4.2.1	Model description	35
4.2.2	Experiment	36
4.2.3	Results	36

4.3	Evolved Multilayer Perceptron	38
4.3.1	Model description	38
4.3.2	Experiment	39
4.3.3	Results	39
4.4	RBF Input Layer Network	40
4.4.1	Model description	41
4.4.2	Experiment	41
4.4.3	Results	42
4.5	Summary	43
5	Ensemble models	45
5.1	Bagging	45
5.1.1	Model description	45
5.1.2	Experiment	46
5.1.3	Results	47
5.2	Self-Organizing Ensemble	49
5.2.1	Model description	49
5.2.2	Experiment	50
5.2.3	Results	50
5.3	Summary	52
6	Final Model Test	53
6.1	Experiment	54
6.2	Results	55
7	Implementation notes	56
7.1	The Design Analysis	56
7.1.1	Compiled or Interpreted	56
7.1.2	Functionality	58
7.2	The Library Employment	58
8	Final Summary	60
	References	63

Chapter 1

Introduction

*For myself I am an optimist —
it does not seem to be much use being anything else.*

Winston Churchill

Hydrological modeling is very difficult branch of environmental physics. Importance of the attempt to understand the water in nature and forecast its behavior was demonstrated recently by the series of devastating floods that had occurred in various regions of the Czech Republic during last decade. The goal of this thesis is to participate with a very little contribution into this effort.

A part of hydrological modeling dealing with short term runoff predictions of little rivers and creeks, where the period between a torrential rain or a strong rainstorm and the flush of the floodwater wave can last only several hours is considered to be notoriously difficult. The problem of the rainfall-runoff modeling is so far examined using computational intense physical-based models.

This thesis persuades an aim to construct an alternative computational intelligence model based on artificial neural networks that performs the task of short term rainfall-runoff predictions focused on small watersheds. The model should forecast the values of future runoff according to a few well chosen input values. An important part of the task is to determine the model input set using the evolutionary techniques.

The computational artificial intelligence is a relatively new part of the artificial intelligence branch gathering the nature-inspired computational methods like the artificial neural networks, the evolutionary algorithms, the fuzzy logic, neuroevolution and the swarm intelligence.

The research is about to be performed using the real hydro-meteorological data obtained by The Czech Hydrological Institute. The models are constructed to predict the runoff of Ploučnice, a little river in the North Bohemia in the town Mimoň using the data from several hydrological and meteorological stations located to the watershed of this river.

The computational approach to artificial intelligence, according to its name, demands a large amount of computational capacity necessary to perform the

tasks as genetic optimization and neural model learning. That is why I highly appreciate the opportunity to join a six weeks internship within the scope of the project HPC Europa 2 at the EPCC Department of the Edinburgh University. The access to one of the top world supercomputers opened an experimental possibilities hardly able to be achieved otherwise.

Acknowledgment

This work was carried out under the HPC-EUROPA2 project (project number: 228398), with the support of the European Community - Research Infrastructure Action of the FP7.

Chapter 2

Computational Intelligence in Hydrology

Hydrology is the scientific study of water and its properties, distribution and effects on the earth's surface, soil, and atmosphere.

R. H. McCuen, 1997

Hydrology as a science is a very specific subbranch of physics interested in the study of water in nature. Like all the other fields of scientific research it was affected by the progress of computer technology. The automated information measurement, recording, and processing together with exponentially increasing computation power available to scientists lead to development of new more sophisticated techniques facing the old and already well known hydrological problems.

One of those problems is the rainfall-runoff modeling. In this chapter, previous attempts on this field will be briefly discussed with the focus on the computational intelligence approach.

2.1 Artificial Neural Networks in Hydrology

Currently used kinds of the hydrological models can be according to Govindaraju [3] classified into three groups: empirical, geomorphology based and physics based. Each sort of models uses different core methodology. While the empirical models miss out the physics at all and the geomorphology based models reduce the physical principles into a simple linear relationship among the model particles, the physical models try to represent nature with a maximal level of the physical accuracy.

The problem of rainfall-runoff modeling is widely studied using the physical numerical models based on building and solving differential equations systems. This state-of-the-art approach suffers from several limitations, however.

The researchers who compile the model have to know many physical values about the particular watershed; such as geological conditions, river bed slope, vegetation density at the neighborhood landscape etc. These information are usually difficult to gather. Also the final model is strongly specific and hardly proper to be generalized to another watershed.

The connectionist models based on artificial neural networks can be classified between the empirical models. They focus on the data itself and examine the relationship among the quantifiable input parameters such as previous flows, recent rainfall history, the temperature, the running river-basin saturation etc. and the output which approximates the current or future runoff.

The empirical models however suffer from several disadvantages. The structure of physical models input variables is appointed by the physics itself. In spite of that, the empirical models do not have such a simple criterion to confirm or reject the relevance of a particular input variable. Various statistical techniques could be used to determine a proper input set but the statistics cannot fully discover the true natural phenomena and their causal dependencies.

The neural networks bring another drawback: a trained neural network metaphorically represents a black box. Due to the advanced nonlinear recursive character of a neural network interior information flow it is virtually impossible to clearly identify the mechanism of decision making process running inside a nontrivially large neural network and derive it into a form which is understandable for the human.

In the rest of this section, a very brief initiation into the topics of artificial neural networks and genetic algorithms are involved. The retrieval of the application of neural networks in rainfall-runoff modeling follows in the second part of this section.

2.1.1 Neural Networks Minimum

The artificial neural network is a type of computational structure that consists of autonomous calculation units called neurons. A single neuron accepts an input information, performs a simple calculation and sends the output to another neurons through connections called synapse.

There exists plenty of various neural network architectures which differ in many conceptual or parametric aspects. A very brief description of those used in this thesis will be mentioned in this section. A more detailed introduction to artificial neural networks can be found for example in [17].

Multilayer Perceptron

The multilayer perceptron is a canonical type of neural network architecture. Its neurons are structured into layers: the first one is an input layer which accepts the input values, the last one is an output layer providing the product

of the neural network computation. One or more hidden layers can be placed between the input and the output layer.

The neurons in the neighbor layers are fully connected i. e. each neuron in a layer is connected to all neurons in a neighbor layer, while neurons in the same layer are not connected at all and there is no connection among the neurons in distant layers as well. All the connections between a neurons are unidirectional: from the input layer in a direction of the output layer.

The computation performed by a single neuron is very simple. It calculates the weighted sum of the input values, then adds its bias value and finally transforms the output using a non-linear transition function. In the case of the multilayer perceptron the transition function is the *logistic sigmoid*:

$$y = \frac{1}{1 + e^{-\sum w_i x_i + b}}$$

This function is common to all the neurons in the network, while the particular vector of weights and the bias varies from one to the others.

The feed forward neural network can be considered as a strongly recursive nonlinear function. It has been proved that the multilayer perceptron with two hidden layers can work as a universal function approximator and that networks with only one hidden layer have the same power when dealing with continuous functions.

The success of this kind of neural network architecture is caused by the existence of relatively fast and reliable learning algorithm called back-propagation[12]. The invention of this supervised learning algorithm based on the gradient descent in eighties started the modern period of the artificial neural networks science.

RBF Networks

The RBF – *radial basis function* – network[9] is a neural architecture derived from the multilayer perceptron. The structure of the network is similar to the original architecture. The only difference is in the neurons' non-linear transition function.

Instead of the logistic sigmoid having a shape of an endless wave above the neuron input space, a local kernel function is used. The value of the kernel function depends on the distance of the input from a specified point in the input space usually called centroid. The mostly used kernel function of the RBF neurons is the *symmetrical multidimensional gaussian*.

Each gaussian neuron can be characterized by its centroid c , the input weights vector w and its radius β which performs a similar task as the bias present in the sigmoidal neuron. The output y of the RBF neuron for a specified input vector x is computed as:

$$y = e^{-\frac{\sum w_i(x_i - c_i)^2}{\beta}}$$

The shape of this function is a multidimensional spherical object located to a particular space point. This makes it possible for the output neurons with a linear or sigmoidal transition function to combine bordered regions in a more efficient way compared to using the sigmoidal units.

Kohonen self-organizing map

Kohonen neural network also known as the *Kohonen self organizing map* [8] is a kind of a very special neural architecture designed to perform the vector quantization. In brief, it spreads a neural grid in the input patterns space in a way it respects the input set density with the maximal closeness. The neurons thus form an optimal set of representatives of the original input set.

At the beginning, all neurons are randomly positioned as points into the input space. The neurons are connected to each other forming either a string, two or more dimensional grid or any other space structure e. g. the hexagonal grid. Then the neurons are slowly moved towards iteratively presented input patterns by moving the closest neuron together with a few of its neighbors in a direction of the particular pattern.

As the training continues, the step length decreases as well as the neighborhood size. Finally, the neurons' positions are accepted as a set of representatives of the original input set.

2.1.2 Introduction to Genetic Algorithms

The genetic algorithm is a stochastic search method inspired by the process of the natural selection, the key idea of Darwin's Evolution. The genetic algorithm is not an algorithm in the classical sense of the word i. e. a rigorous sequence of steps performed above the data, but it is rather an algorithmic scheme which has to be adapted for the particular problem.

Before a very brief description of the the algorithm itself, a few definitions of basic evolutionary concepts involved in the algorithm should be mentioned.

Individual and its Genome

The individual is the term used for a single solution of a problem being optimized by the genetic algorithm. The encoding of the individual is called the genome. The genome consists of units called genes representing particular properties of the individual.

Since the birth of genetic algorithms in 1975 an acrid dispute has broken out about using the binary genome for the genetic individual encoding. Some consider the binary genome to be the optimal unit for genetic breeding because

Algorithm 1 Generic Scheme of Genetic Algorithm

Input: Population size p , Max. generation number g_{\max} , Desired fitness f_{stop} , Mutation probability p_{mut} , Crossover probability p_{cross} .

Output: Final best individual i^*

```
P ← randomly initialized population of size p
g ← 0
repeat
  for all  $i \in P$  do
    if  $\text{flip}(p_{\text{mut}})$  then
       $i.\text{mutate}()$ 
    end if
  end for
   $P_n \leftarrow$  empty population
  while  $|P_{\text{new}}| < p$  do
     $i_1 \leftarrow \text{select\_individual}(P)$ 
     $i_2 \leftarrow \text{select\_individual}(P)$ 
    if  $\text{flip}(p_{\text{cross}})$  then
       $\text{crossover}(i_1, i_2)$ 
    end if
     $P_{\text{new}} \leftarrow P_{\text{new}} \cup \{i_1, i_2\}$ 
  end while
  for all  $i \in P$  do
     $f_i \leftarrow i.\text{evaluate\_fitness}()$ 
  end for
   $P \leftarrow P_{\text{new}}$ 
   $i^* \leftarrow i \in P$  with maximal  $f_i$ 
   $g \leftarrow g + 1$ 
until  $f_{i^*} \geq f_{\text{stop}}$  or  $g > g_{\max}$ 
return  $i^*$ 
```

of its suitability for simple bit mutation and crossover as well as its similarity to the natural evolution chromosome, which is a quaternary string. The others prefer to keep the individual more in a form of phenotype that avoids the encoding issues and apply genetic operators adjusted to the particular individual class.

Fitness Function

The fitness evaluation is a key point of every particular genetic algorithm. A problem specific fitness function must be defined above the space of individ-

uals that assigns to every individual a (positive) real number. This number determine the quality of the solution i. e. its closeness to the desired optimum.

Population

One of the essential concepts of the evolutionary computation is the idea of population. The population is a kind of genetic pool which supplies the genetic algorithm with a source of genes for building new individuals. Using the words of optimization processes it provides some level of searching parallelism, which makes the optimization process more robust thus it is less likely that it will get stuck in a local optimum.

On the other hand, it is also the main source of computation complexity, since every operation is performed on all individuals in the population.

Selection

The selection is the engine of the genetic algorithm which moves the population in a direction of higher fitness. The selection operator is used by the algorithm to choose those individual of the population which survives into the new generation. There exist plenty of various selection operators. Most of them realize a kind of random sampling with a preference of individuals with higher fitness value.

Let us mention the selection operators used in this thesis. The *proportional selection* chooses the individuals with the probability proportional to its fitness value. The *tournament selection* picks repetitively two random individuals up from the old population, selects the individual with the higher fitness and places it into the new population.

Crossover

The crossover operator randomly recombines the genome of two (or rarely more) individuals by swapping their genes. The purpose of this operator is to put together valuable particles of the problem solution that emerged independently inside different individuals. The particular form of the recombination depends on the encoding of the genome and it is usually problem specific.

Mutation

The mutation is an instrument intended to preserve the natural genetic diversity of the population reduced by the selection press. The particular implementation of the mutation operator is also problem specific and it usually realizes minor localized changes of the genome with a very low probability of occurrence.

The basic skeleton of the genetic algorithm described in the pseudocode is denoted as the algorithm 1. At the beginning, the population P is initialized with random solutions. The algorithm consists of so called generations.

During every generation, the individuals in the population are randomly transformed using the previously mentioned genetic operators and specified probabilities. After this phase, the fitness of all individuals is computed. Consequently, the new population is composed of the individuals chosen by the selection operator according to the fitness values.

The algorithm finish its run when it either finds a solution with desired fitness value or reaches the specified maximal generation.

2.1.3 Neural Ensembles

The architectures of neural ensembles belongs to wider class of artificial intelligence methodology called combination of experts. The application of this approach on the field of neural networks is considered to be next step of the progress of neural networks [15]. For terminological clarity there should be mentioned, that the term *ensemble* is commonly used for a set of uniform experts, while for a set of experts that differ from each other in a way which is treated to be fundamental, rather the term *modular architecture* should be used. Both architectures employed in this thesis balance between these categories, however both are inspired by the *bagging* architecture which belongs to among the ensembles so this term will be in generally used. The purpose of this section is to briefly introduce the bagging method.

Bagging

The word *bagging* is an abbreviation of the term *bootstrap aggregation*. The method of *bootstrapping* is used in statistics to obtain multiple sub-samples from an original sample using *random selection with replacement*.

The *bagging*[16] is a methodology to create an ensemble containing neural networks characterized with some level of internal diversity. The ensemble output is then obtained as a combination of the individual networks outputs, usually an average or possibly a weighted average.

The key idea of the bagging method is to sample the training set using the bootstrapping method. Each network in the ensemble is randomly initialized and trained using a sample formed in this way. While the particular sample of the original training set does not contain all the training patterns and on the other hand some patterns appear in the set more than once, each network specializes slightly, which impose the desired diversity into the ensemble.

2.1.4 Related Work

In spite of the development of artificial neural networks in the eighties, first attempts of their applications in hydrology did not appear until nineties [4]. The physical numerical models still form the main stream of the hydrological modeling.

The main goal of rainfall-runoff modeling is to determine the dependency of river runoff volume on a set of relevant physical values: namely the rain drops levels together with other temporary and permanent conditions. The problem is considered to be very difficult because of the known nonlinearity and complexity of that dependency.

Two sets of model input values can be distinguished: the true functional inputs containing the rainfall, temperature or snow-melt directly involving the output runoff and background physical parameters of watershed such as local geological conditions, river bed slope, soil quality and vegetation.

Both of the input sets have to be available for a researcher composing a physical numerical model together with reliable theoretical knowledge of their role in this complex relationship. The neural networks however make it possible to extract the relationship between inputs and outputs of hydrological process without the physics being explicitly provided to them [4]. The robustness, when dealing with noisy and disrupted data sources, is considered to be another advantage of the artificial neural networks.

The first preliminary study using feed forward neural networks for a runoff prediction was made by Halff et al. [5]. A three layer perceptron with five hidden neurons was used to predict the hydrograph – a sequence of measured runoffs – depending on the hyetographs – sequences of measured rain drops – during five storm events in the northwest of US. Despite of a small scope, this study inspired another researches to explore applications of neural network on this field.

Zhu et al.[21] designed two neural models, both to predict minimum and maximum of the flood hydrograph in Butter Creek, New York. The first of them accepted both, the rainfall and the previous runoff inputs. The second designed for the cases when previous runoff values are not available accepted only the rainfall inputs. An attempt with the online learning of the model was made but the performance gradually deteriorated. The conclusion of the research was a strong dependency of the model prediction quality on the training data set diversity. The results of the model were considerably better in cases of interpolation than for extrapolation. This is one of the typical properties of the neural network models.

A similar point was made also by Minns and Hall [14] experimenting with a model learning from the outputs of the Monte Carlo simulations using the existing scalable nonlinear model for flood estimation. Despite the level of existing model nonlinearity having no significant impact on the performance of the neural network model, the predictions of the neural network model were strongly affected by standardization of predicted values. The prediction qual-

ity was thus distinctively better for interpolation than for extrapolation.

Jayawardena and Fernando [7] experimented with a flood forecasting neural model using the radial basis function instead of classical sigmoidal neurons. The network was trained on a very small 3.12km² watershed. The correlation analysis suggested very short period between the rain and corresponding runoff that was not longer than 3 hours. Although the performance of the RBF model was comparable to the standard multilayer perceptron, the training time for the network with RBF units was much shorter.

An interesting observation was published by Tokar and Johnson [20] referring about their attempt to train a neural network to forecast the daily runoff of Little Patuxent River in Maryland. The network learned on three sets containing wet, dry, and average periods of a year. The best predictions were obtained from the nets trained on the wet and dry periods compared with the average one. The effect of the period type was distinctively stronger than the effect of the period length.

Chapter 3

Source Data

In this chapter, the hydrological source data will be introduced together with their brief statistical analysis. The analysis is mainly focused to those aspects of the source data that are relevant to the desired prediction model construction.

All data discussed in this chapter were acquired from The Czech Hydrometeorological Institute, branch office in Ústí nad Labem. The data originates from long term periodical measurement of runoff, rain drops and air temperatures in the selected areas of the Ploučnice upper stream watershed.

Ploučnice, the river in the North Bohemia, flows from its springs at the foot of the hill Ještěd in the western direction where, after 106km long path, it finally joins to Labe (Elbe) in Děčín.

On the 73rd river kilometer (measured from the mouth i.e. 33rd from the spring) there is an automatic hydrological station in the town of Mimoň which measures the volume runoff hourly and sends the values to the CHMI as a part of flood forecast warning system.

The watershed area of the river above this point is approx. 270km². Just a few hundred meters up the river from the Mimoň station, a major tributary, the creek called Panenský potok, joins to Ploučnice. Panenský potok flows from the town Jablonné v Podještědí where it origins as a junction of several minor mountain streams.

There is a meteorological station in Jablonné v Podještědí, which measures among other things rain drops and air temperatures. Similar station is located also near the town Stráž pod Ralskem which is about 12km up the river from Mimoň.

The data that were available for the purposes of this thesis covers the measurement performed at previously listed stations during the period of years 2006 and 2007.

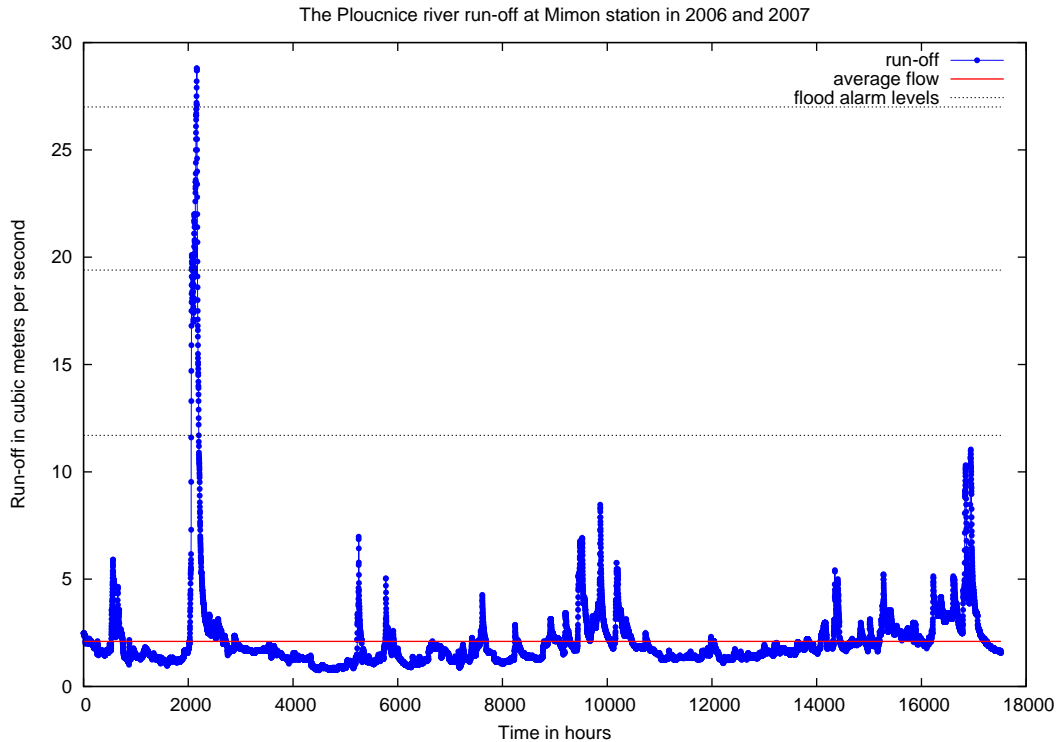


Figure 3.1: The chart of hourly measured volume flow of Ploučnice runoff at hydrological station in Mimoň. The red line shows the long term average runoff. The three dashed lines corresponds to the three levels of flood alarm.

3.1 Time series

3.1.1 Runoff at the Mimoň station

The source data consist of several time series. The key time series is the river flow runoff measurements of the Ploučnice river at the hydrological station in Mimoň.

The figure 3.1 shows this series. The chart clearly shows high level of variance present in the series. Although the series contains several long and calm periods of relatively low runoff volumes fluctuating around the average value, there are also a few peeks, in the middle of rather more varying segments, one of them higher than the 3rd level of flood alarm.

Descriptive statistics

The long term average runoff at this station is $2.3 \text{ m}^3/\text{s}$ but the cent-year's flood maximum runoff rises to $103 \text{ m}^3/\text{s}$. The average runoff of the examined two year segment is $2.104 \text{ m}^3/\text{s}$ and the standard deviation is $1.969 \text{ m}^3/\text{s}$. The standard deviation almost as high as the average in the case of positive value set suggests a hardly predictable time series with very irregular behavior.

The result of the statistical test of homoskedasticity i. e. the consistency of the variance during the time is also interesting. The Fligner-Kelleen median test is feasible for statistical sets whose normal distribution can not be expected. An application of this test to the couple of sets corresponding to the year 2006 and 2007 proved that these two sets have a different level of variance with statistical confidence level 4.10^{-7} . Similar result was also obtained by an application of this test to relatively calmly looking subsequence that covers the period of Autumn 2006.

3.1.2 Weather

Two time series of rainfall hourly measured as well as two time series of air temperatures are available from two meteorological stations in Jablonné and Stráž. While the series of air temperatures from meteorological station in Stráž are measured hourly, corresponding values from station in Jablonné are available only daily at 7am, 2pm and 9pm.

Descriptive statistics

The average rainfall computed from both stations and both years are 0.076 mm/hour with standard deviation 0.396. The correlation between the two stations rain drops is 0.516, which is the maximal value of correlation in regard to the mutual time shift of the series.

3.1.3 UPS

According to the advice of hydrologists from the CHMI, another complement value called UPS was involved into input data. The UPS (API) [18] value is a compound summary of recent rainfall in the area of the watershed. It describes an estimated saturation of the watershed. The initial value is computed as:

$$UPS = \sum_{t=1}^{t=n} \bar{S}_t C^t$$

where the t is the number of days to the past, S_t is the mean summary of daily rain drops at different places in the watershed area and the C is the evaporation coefficient lower than 1. The value of the evaporation coefficient is usually estimated by so called evaporation constant. Empirical value of this constant for Central Europe is 0.93. The UPS values for subsequent days are computed as:

$$UPS_{t+1} = UPS_t \cdot C + \bar{S}_{t+1}$$

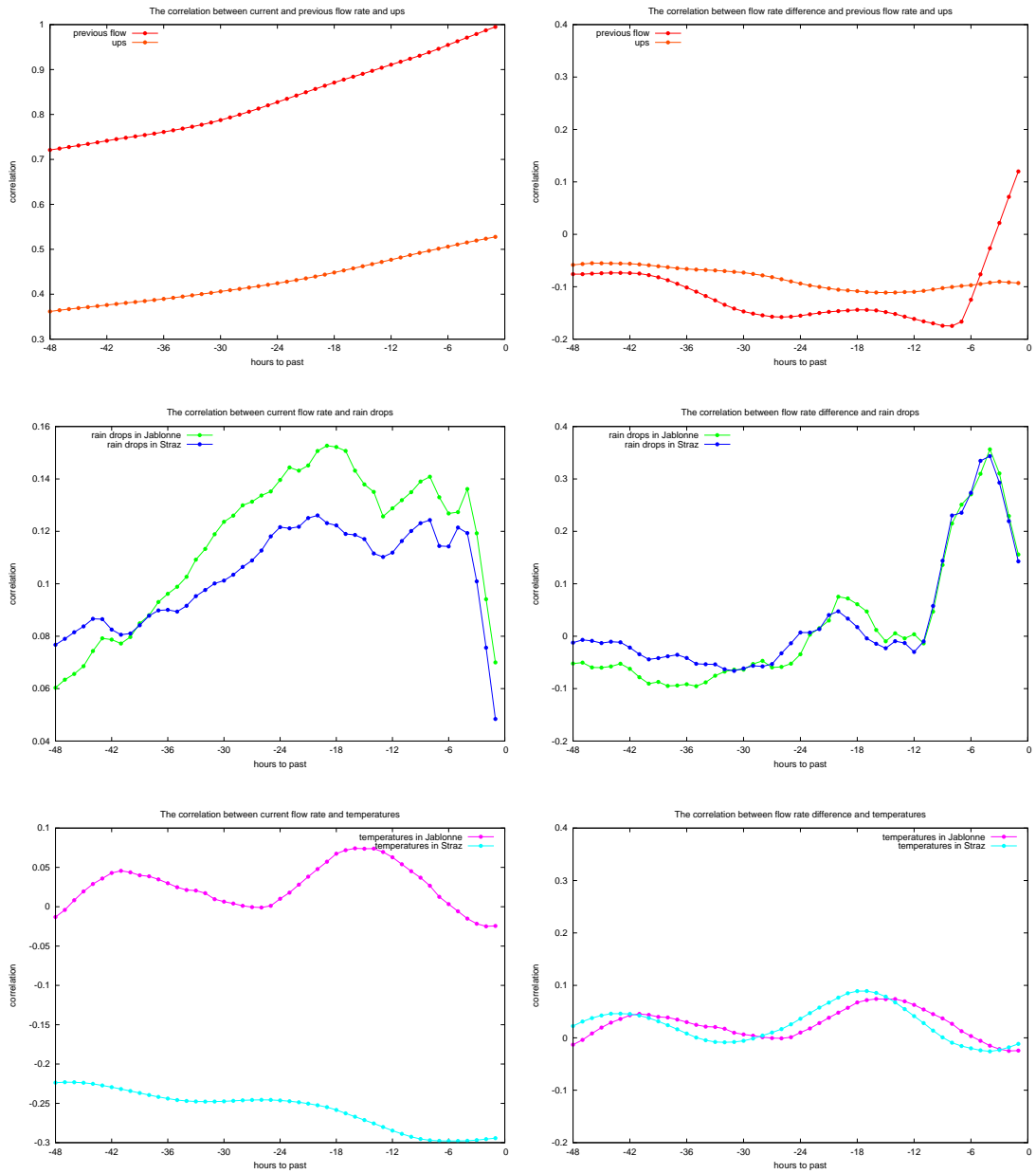


Figure 3.2: The left column shows the correlation between the current runoff and the previous values of runoff, rain drops and air temperatures (the y-axes are not normalized). The right column contains similar correlations with the series of differences between the recent runoff and the runoff values six hours to the past.

3.2 Runoff prediction

Let us discuss a rainfall-runoff time series prediction problem together with the quality measurement criteria. After the consultation with hydrological experts from The Czech Hydrometeorological Institute, it became clear that for the purpose of the flood early warning, the key problem is a precise short term runoff prediction. The most required forecast is a six hours ahead estimation of runoff volume. The interval of six hours seems to be short enough to have reasonable chance to good prediction results and long enough to have a time to utilize them. This objective will be followed in the whole thesis.

The natural metric to determine the quality of the solution is the mean squared error (MSE):

$$\text{MSE} = \frac{1}{n} \sum (\hat{x}_i - x_i)^2$$

where \hat{x} is the vector of predicted values while x is the actual values vector. Various critic, is known about this metric mostly focused on the problem of overestimating the dependence of outliers and also on its low descriptive power given by the quadratic character. For those reasons the mean absolute error (MAE) is used instead in the field of time series prediction:

$$\text{MAE} = \frac{1}{n} \sum |\hat{x}_i - x_i|$$

However, there is another metric widely used for measuring the prediction power of hydrological models. The metric is called Nash–Sutcliffe[11] model efficiency coefficient (EC), and it is defined by subsequent prescription:

$$\text{EC} = 1 - \frac{\sum (\hat{x}_i - x_i)^2}{\sum (\bar{x} - x_i)^2}$$

the \bar{x} is the long term mean of the predicted value. The EC can take a value between $-\infty$ and 1. The value of EC equal to zero means the prediction is as good – or rather as bad – as the estimation by the long term mean value. The positive value characterizes better predictions, while the negative value of EC stand for predictions even worse than long term mean estimation. The EC equal to one denotes the prediction exactly fitting the reality.

3.3 Correlation analysis

Each previously introduced time series provides infinitely many potential inputs whose relevance decreases as the time goes backwards. In regard to reasonable selection of proper input data, the standard statistical method of correlation analysis [19] can be employed.

The figure 3.2 contains a set of charts, each of them representing a time behavior of correlation coefficient between an input value and one of two output values.

The charts on the left are related to the output value of current runoff, while those in right column reflects the correlation of input series and the series of 6 hours differences which is the true desired information, we are expecting to get from the predictor.

3.3.1 Previous flow and watershed saturation

The high level of autocorrelation of the series of runoffs present on top left chart is not unexpected. It seems to be that the correlations between previous values of UPS and the current runoff behave similarly to the correlation with the series of previous runoffs, only having significantly lower correlation level.

Negative value of the correlation between the two input series and the series of differences plotted in the upper right chart conforms to the simple fact that maximal values of limited series are mostly followed by values lower and vice versa. The steep increase located at the final part of previous runoffs correlation series has similar reason. If the series is reaching its maximum, it must have risen for a few last hours so the difference is mostly positive.

Both of these effects are caused just by the general character of series of differences. None of them has any potential to be exploited for prediction and they are mentioned here only to avoid confusion.

3.3.2 Rainfall

The two charts in the middle provide much more promising information. The level of correlation coefficients between both series of rains and the current runoff as well as the series of runoff differences acknowledge the expected dependence of these series. The influence of the rain to the current runoff seems to be more spread into time of approximately last 40 hours. The time series of runoff differences are on the other hand apparently influenced by the rain drops fallen in only last 12 hours.

The last part of both series is also very descriptive. According to previous expectations the correlation between rainfall and output series decreases but the steepness of that decrease clearly shows that the last relevant input is the one 3 hours old (from the perspective of current flow or 3 hour ahead from the perspective of 6 hours prediction).

3.3.3 Temperatures

The correlations between the output series and the two series of air temperatures bring no valuable information. The sine shape of the curves is given

by the periodical nature of air temperatures during day. This autocorrelation behavior is then projected into the correlation with both output time series.

The blue curve in the first chart which represents correlations between previous hourly measured temperatures at meteorological station in Stráž pod Ralskem and in addition the current runoffs provide the observation that cold days are those with higher river flows. The rest three curves oscillate around zero correlation level.

3.4 Regression analysis

In this section, a linear regression analysis will be performed. The main goal of the analysis is to examine the suitability of linear regression model to be employed for the purposes of short term runoff prediction based on previously introduced input time series. The performance of the simple regression model will then serve as a comparison scale for more sophisticated artificial intelligence models.

The secondary goal is to get a notion about the size of a virtual time window which can be applied on the input time series in order to obtain all necessary information from the series on one hand and not to over-complicate the model by introducing too many unnecessary variance on the other hand. Therefore a simple algorithm to identify the statistically relevant regressors will be used.

Since the statistics is not a main topic of this thesis, only very simple regression analysis just fitting the two mentioned goals will be performed. No regressor interaction neither non linear regressors will be involved. This task is much more suitable for neural models studied in the main part of this thesis. Also, the statistical analysis of the model residuals will be omitted.

3.4.1 Input data

Time Period

The data of this regression analysis are chosen as a subset of previously analyzed time series data. For purposes of this analysis, only a sub-sequence of the data that covers the time period from May 2007 to October 2007 is explored.

This constraint is applied to screen out the impact of the river runoff increase caused by winter and spring snow melt. The knowledge about snow quantities is not available and the omission of this effect can cause considerable degradation of the quality of the regression model.

Sampling the time series

Time points from the period (May to October 2007) were chosen. The mean distance between time points was appointed to 24 ± 3 hours. Randomly chosen 75% time point is placed into the training set, the remaining fraction is placed

into the validation set. The minimal distance between points from different sets is 21 hours. This can make suitable basis for obtaining credible results.

Time Series

Suppose we have 283 input values. First 43 are the values of previous flows measured between 6 and 48 hours backwards. The two sequences of current rainfalls follows. Both of them contains 48 values, 42 measured, last six are obtained from weather prediction. The 48 values of aggregated rains function, the UPS are next in the line. Finally, the two series of temperatures are involved each of which contains 48 values, last six again comes from weather prediction.

Output

The output value of the regression is the 6 hours ahead prediction of the volume runoff. Alternatively the 6 hours forward difference of current runoff can be predicted. Both models were created to examine the difference.

Algorithm 2 Selection of The Linear Regressors Set

Input: Sets S_i of regressors each containing all input values of particular time series. The maximal acceptable zero element probability α .

Output: Set R containing final set of regressors

```

R ← ∅
for all  $S_i$  do
  R ←  $S \cup ReduceSet(S_i)$ 
end for
return  $ReduceSet(R)$ 

```

Procedure ReduceSet (Set S)

```

repeat
  Build model  $M$  using  $S$ 
  for all regressors  $b_i \in S$  do
    Compute partial t-statistic  $t_i$  of  $b_i$  related to  $M$ 
     $p_i \leftarrow$  probability:  $b_i = 0$ .
  end for
   $p_{i^*} \leftarrow$  minimal  $p_i$ 
  if  $p_{i^*} > \alpha$  then
     $S \leftarrow S \setminus b_{i^*}$ 
  end if
until  $p_{i^*} < \alpha$ 
return  $S$ 

```

3.4.2 Model composition

The composition of linear regression model basically means the selection of input values (regressors) which are relevant to predict the dependent value. Not all of the previously listed input values are proper to be involved into the model.

If a model has too many degrees of freedom i. e. it contains too many input values whose true dependency to the output value is weaker than their internal variance, the model will fit the training data pretty well, but the real prediction ability of the model will be poor.

The criterion designed to estimate, whether particular regressor is a relevant part of the model or if it should be removed are the statistical tests of partial regression coefficients. The test null hypothesis is that the partial regression coefficient is zero. The hypothesis must be rejected with the desired level of confidence α to consider the regressor as a true part of the model.

The algorithm 2 was used to gradually build the model. Because of computational problems with so many regressors, algorithm runs in two stages. Firstly, for partial models then for the final model:

3.4.3 Results

The algorithm has been run three times with three different threshold confidence levels α being equal to 0.1, 0.01 and 0.001. The resulting sets of relevant regressors are illustrated on two diagrams in the figure 3.3.

The upper diagram belongs to the model which predicts the runoff volume while dependent value of the lower one is the 6 hours runoff difference. Each line represents one input time series and each cell represents one hourly measured value of this series.

The colors of particular cells vary for different statistical levels of confidence that the particular variable has non zero impact to the output. The red cells stands for $\alpha \leq 0.001$, violet for $\alpha \leq 0.01$, blue for $\alpha \leq 0.1$, and white for less significant levels of confidence.

The prediction quality of those six models is summarized in the table 3.1. Each row corresponds to one confidence level α . The first row contains the mean absolute error values and the efficiency coefficients for prediction based on current value i. e. the forecast that always says that 6 hours ahead runoff will be exactly the same as the current runoff. These values are included into the table for comparison between the prediction quality of linear regression and the simple prediction method.

For both prediction models it seems to be clear that low confidence level implies better results on the training set together with worse results obtained on the validating set. This is caused by the fact that a higher number of regressors means also larger amount of natural variance of inputs to be wrongly explained as a causal dependency between an input and output.

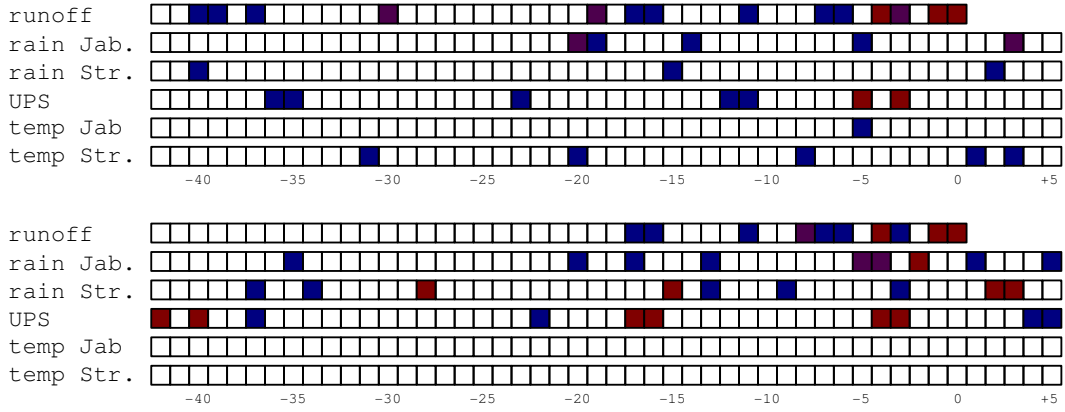


Figure 3.3: The two diagrams show the statistically relevant input variables for the two linear regression models. The red cells variables have $\alpha \leq 0.001$, the violet have $\alpha \leq 0.01$, the blue $\alpha \leq 0.1$ and the white cells stand for less significant inputs.

Superiority of the results of models having $\alpha = 0.01$ to those with $\alpha = 0.001$ suggests that this confidence level is already too tight. The quality of regression models predicting the future flow on one hand and models predicting the difference of future flow on the other hand is comparable, which agrees with the linear character of differentiation and linear regression.

The best efficiency coefficient on the validation set $EC = 0.750$ was obtained for $\alpha = 0.01$ by the model predicting future runoffs. It is just slightly better than the $EC = 0.713$ obtained by predicting future runoff by the current one. The absolute error obtained from the regression model is even marginally worse than the error of that simple forecast method.

The figure 3.4 contains a chart that shows a comparison between the real series of 6 hours differences and the series of differences predicted by the best difference focused model. The series corresponding to the simple forecast which estimates the future to be equal to the present is the constant line equal to zero. It is obvious from the chart that the prediction power of the linear regression model is not very impressive. Although there are a few suggestions of predicting behavior, from the general point of view the two time series do not seem to be fitting one another.

The conclusion of the regression analysis performed in this section is that

TABLE 3.1: THE RESULTS OF LINEAR REGRESSION								
α	runoff volume				runoff difference			
	MAE_t	EC_t	MAE_v	EC_v	MAE_t	EC_t	MAE_v	EC_v
—	0.0765	0.932	0.1084	0.713	0.0765	0.932	0.1084	0.713
0.1	0.0394	0.992	0.1684	0.619	0.0387	0.993	0.1458	0.599
0.01	0.0630	0.978	0.1089	0.750	0.0652	0.976	0.1095	0.745
0.001	0.0752	0.970	0.1167	0.709	0.0717	0.973	0.1260	0.724

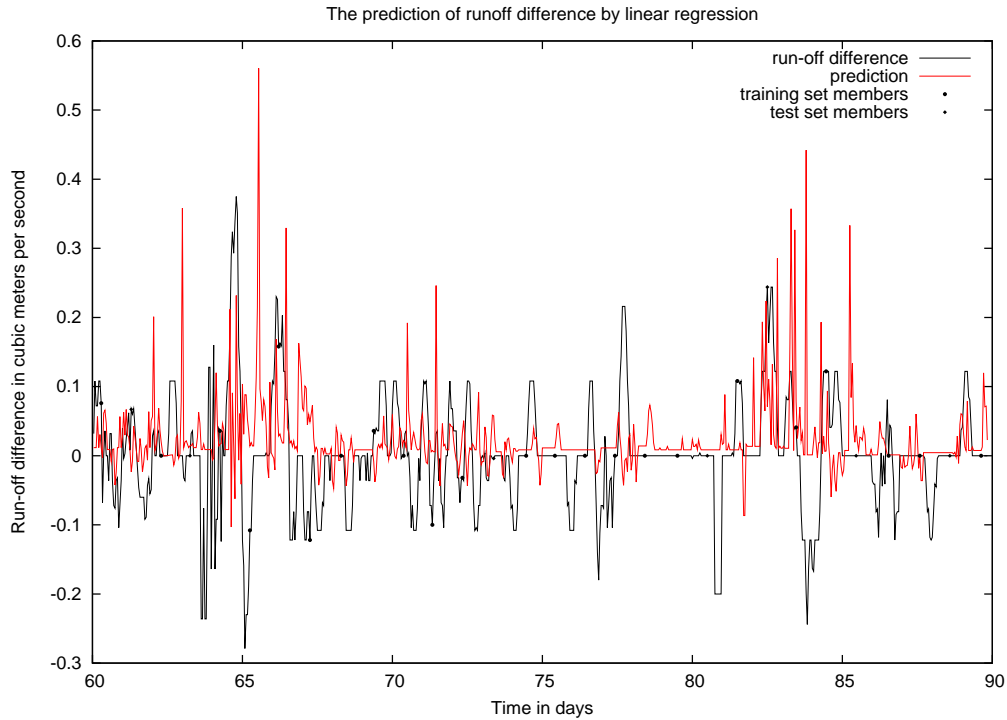


Figure 3.4: The best regression model applied to the part of the whole history of runoff differences. Because the chart containing all the points of the summer 2007 is too dense, this one covers only the period of July 2007

linear regression model can not be used for short-term runoff prediction problem. However, the regression analysis provided several valuable results on the field of relevant input set composition. All of the four hydrological time series showed themselves to be useful inputs to the model: the falls series and the previous flows mainly in the second half of the examined time window, the aggregated rainfalls series – UPS – evenly during all the period.

The air temperatures were not employed by the model in a satisfactory manner. A speculation can be expressed that this may be caused by the character of temperatures influence which could be more incorporated into an interaction with other time series. These possible nonlinear interactions were not a subject of an interest of this simple linear regression analysis.

Chapter 4

Computational intelligence models

The real question is not whether machines think but whether men do.

B. F. Skinner, 1969

In this chapter the neural nonlinear analysis is about to be performed on the source data examined in the previous chapter. The goal of this analysis is to construct the artificial intelligence model with maximal predicting power. The strategy is quite similar to the previously used statistical approach. Firstly the relevant regressors will be identified according to the intended neural methods. Then the predicting ability of various neural architectures will be examined.

4.1 Input Filter Evolution

This section partially elaborate on the effort to determine relevant input variables of the runoff forecast models. This was examined in the previous chapter as a part of the regression analysis (see section 3.4). Statistical tests were employed as an instrument for that purpose. In this section a different approach will be used for the same goal.

4.1.1 Problem definition

First, let us exactly define the input filter evolution problem. As it was discussed before in the section 3.1, the inputs consist of recent historical values of the six time series. Let us define the term *input filter* above them.

Suppose we have a set S of integer numbers from the interval $[1, 283]$ or better $[1, 43 + 5 \times 48]$. Let each number represent an input variable coming from a time series of those introduced in the section 3.1.

First 43 numbers in the set represent the past values of the runoff volume measured hourly til this time. Following 48 inputs stand for previous rainfalls measured at the first meteorological station, last five being a forecast of

those rainfalls which are about to come in the next five hours. Next 48 numbers represent the same rainfall inputs measured at the second meteorological station. The following 48 numbers stand for the input variables containing the value of hydrological cumulative rainfall coefficient UPS. Last two sub-intervals both containing 48 numbers represent past and future values of air temperature measured hourly at both stations.

Let us have a non empty set F , subset of S , and call it an *input filter*. This filter then represents a subset of all input values (i.e. regressors in the terms of regression analysis in section 3.4). The total number of filters is equal to the number of different nonzero binary vectors of size 283 which is $2^{283}-1$. The objective of this section is to methodically select the proper filter from the set of all possible filters, in accordance with the prediction model composition.

First, let us discuss the contribution of such selection. Each variable in the input set has a relevance to the output and thus the simplest way how to deliver maximum information to the model is involving all the 283 variables. However, as the tests of regression models have shown, the high degree of freedom tends to yield worse results of the model obtained with the validation set. This phenomenon is known as *overfitting* (also *overlearning* or *overtraining*). It appears whenever there is a discordance between training set and validation set.

Given by the fact that the usable data set covers just the period of summer 2007, the training and validation sets are relatively small. Independently on the method of choosing data patterns into the sets, significant discordance between those sets is inevitable. That is why the overfitting problem is the most vital obstacle on the path to well predicting model.

When dealing with the linear regression, standard statistical techniques such as previously used t-tests or ANOVA methodology of regression model composition could be used to solve the filter selection problem. Involving non-linearity inherently present and most desired at the neural network models, however, disable the usage of these simple fast and reliable statistical techniques. That is why evolution replaced statistics in this chapter.

4.1.2 Filter Evolution

At this point, let us discuss the application of the genetic algorithm as an instrument to solve the filter selection problem.

The genetic algorithm is an evolutionary technique which was briefly introduced in 2.1.2. As it was already mentioned, the genetic algorithm is not a final algorithm, but an algorithm scheme. To create the true algorithm which solve a particular problem, this scheme needs to be instantiated with the genetic individual definition and the algorithm skeleton has to be filled in with the particular crossover and mutation operators. Finally, implementation issues should be solved. In this section the main focus is put on the operators and individual definition, while the implementation will be briefly discussed

later.

Genetic Individual and Genome

The genetic individual shaped by the evolution is in this case obviously the input filter. When dealing with input filter evolution, the question of genome encoding fortunately need not to be solved, since the genotype and phenotype are identical. Thus a simple bit string of the length of 283 was chosen, having ones on positions of selected outputs and zeroes on the others.

Individual Initialization and Fitness Evaluation

According to the theory mentioned in 2.1.2, we are supposed to create a function from the space of filters into the set of positive real numbers which defines partial order consistent to the solution quality i.e. the maximal value of the fitness function assigned to the best possible solution.

Unfortunately, this essential condition cannot be always granted in real word applications and this is one of those cases. The best solution of this problem is the filter which provides a neural network model maximal potential to obtain a good result on the validation set.

The only way to explore that potential for particular filter is to let a neural network learn on the data set produced by the filter and measure its performance. Lower validation error means then higher fitness value. Several complications arise when trying to design the fitness function this way.

Firstly, the suitable neural network model should be chosen together with the learning algorithm. Reasonable requirement about the architecture and the learning algorithm is to be as general as possible providing reliable results for most of different architectures. Ideal would be a sort of a nonlinear version of the linear regression. That is why a canonical neural nonlinear model the *feed-forward* neural network based on classical back-propagation learning (also known as *multilayer perceptron*) briefly mentioned in section 2.1.1 was chosen. The variant using the momentum was chosen from the set of possible back-propagation based algorithms for its stability and speed. The algorithm 3 describes the computation of the fitness function for specified filter.

The second problem is more serious. The backpropagation learning of neural network is a gradient method and thus completely deterministic. On the other hand the initial position of weight vector in the weight space is chosen randomly. Not all initial positions are naturally equally good. When better initial position is set to the network learning on a worse filter it can achieve a better result than a network learning by using a better filter initialized to a worse position. This violates the previously defined partial order condition.

Straightforward solution could be proposed – to fix the initial position. This, however, will not truly solve the problem. Various initial positions could be differently favorable to various filters. Choosing one would destroy the requirement of generality mentioned before.

Algorithm 3 Input Filter Fitness Evaluation

Input: Input filter F , Training set T , Validation set V . Number of training epochs e .

Output: Fitness f .

```
TF ← T filtered using F
VF ← V filtered using F
N ← randomly initiated neural network
repeat
  N.train(TF, e)
  Compute error  $e_t$  of N above TF
  Compute error  $e_v$  of N above VF
until  $e_v$  increases
return  $1/e_v$ 
```

While the multiple initialization during one fitness evaluation would make the algorithm impossibly slow, the only solution left is to set up suitably large population size and generation number. Enough number of fitness evaluation can decrease the stochastic initialization effect thanks to the statistical character of obtained results.

The initialization of genetic individual chromosomes is the next question to be discussed. Common way of bitstring based individual initialization is randomly set each bit to 1 or 0 with the uniform probability. Experiments showed however, that the learning ability of networks with as many inputs was quite poor compared with networks having lower input number. That is why the fraction of involved inputs of initial individuals was set to 0.1 which showed to be suitable choice.

Genetic Operators

Using the bitstring as a chromosome brings the advantage that the standard genetic operators can be used in most cases.

The two point crossover operator was used as a recombination operator. This operator perfectly corresponds to the requirements expected from the recombination operator by the Holland's scheme theorem [6]. The locality of genes could be expected because the neighbor bits in the chromosome encode analogous information. Also another knowledge such as trend could be obtained from two or more closely placed bits.

The standard bit flipping mutation was rather less suitable. As the initial chromosome contains ten times more zeroes than ones, the simple uniform bit flipping mutation would strongly prefer flipping the zero to one than vice versa. This would tend to artificial growth of the filter not because of evolution process but simply because of the stochastic character of the mutation itself. This complication was simply solved by involving biased mutation which

statistically holds the ratio between zeroes and ones in the chromosome.

Finally, the selection was chosen. To avoid the problems with fitness scaling as well as to ensure strong enough selection pressure for the whole run of genetic algorithm, the one round deterministic tournament selection briefly introduced in section 2.1.2 seems to be the good choice.

4.1.3 Experiment

The set of experiments was performed using the parallel implementation of genetic algorithm designed for the purpose of this thesis. The extent of necessary computations exceeds the computational capacity of a single personal computer. The most time consuming operation performed by the genetic algorithm is the process of filter quality evaluation.

The parallelisation was an inevitable way to master such an enormous computation task. Although the task of efficient parallelisation of scientific computation problems is in general very difficult, the reasonable degree of parallelisation of this type of genetic algorithm is fortunately quite straightforward.

Due to the fact that evaluation of the fitness of a particular individual is independent on the fitness evaluation of the others, several individuals can be evaluated separately using different cores of a parallel computer. The concrete implementation details exceed the topic of this chapter and they are discussed later in chapter 7.

Three main experiments were performed on three different networks: with 2, 8, and 32 hidden neurons. Because of the stochastic character of an evolutionary algorithm run, five independent runs were executed to obtain reliable results. Each run consists of 1000 generations over the population of 160 individuals.

The population size equal to 160 was chosen due to the previously mentioned initialization mechanism. After simple calculation we get that for 10% of initially chosen bits, the population size should be at least 100 for suitable chance that each bit is set to one at least in one individual of the initial population. The second reason which supports such a large population is the already mentioned parallelism. Most of the machines employed by this algorithm run has the number of cores which is a multiple of 16. To obtain maximal efficiency the population size which is a multiple of 16 is necessary. Thus 160 was chosen as a population size which satisfies both requirements.

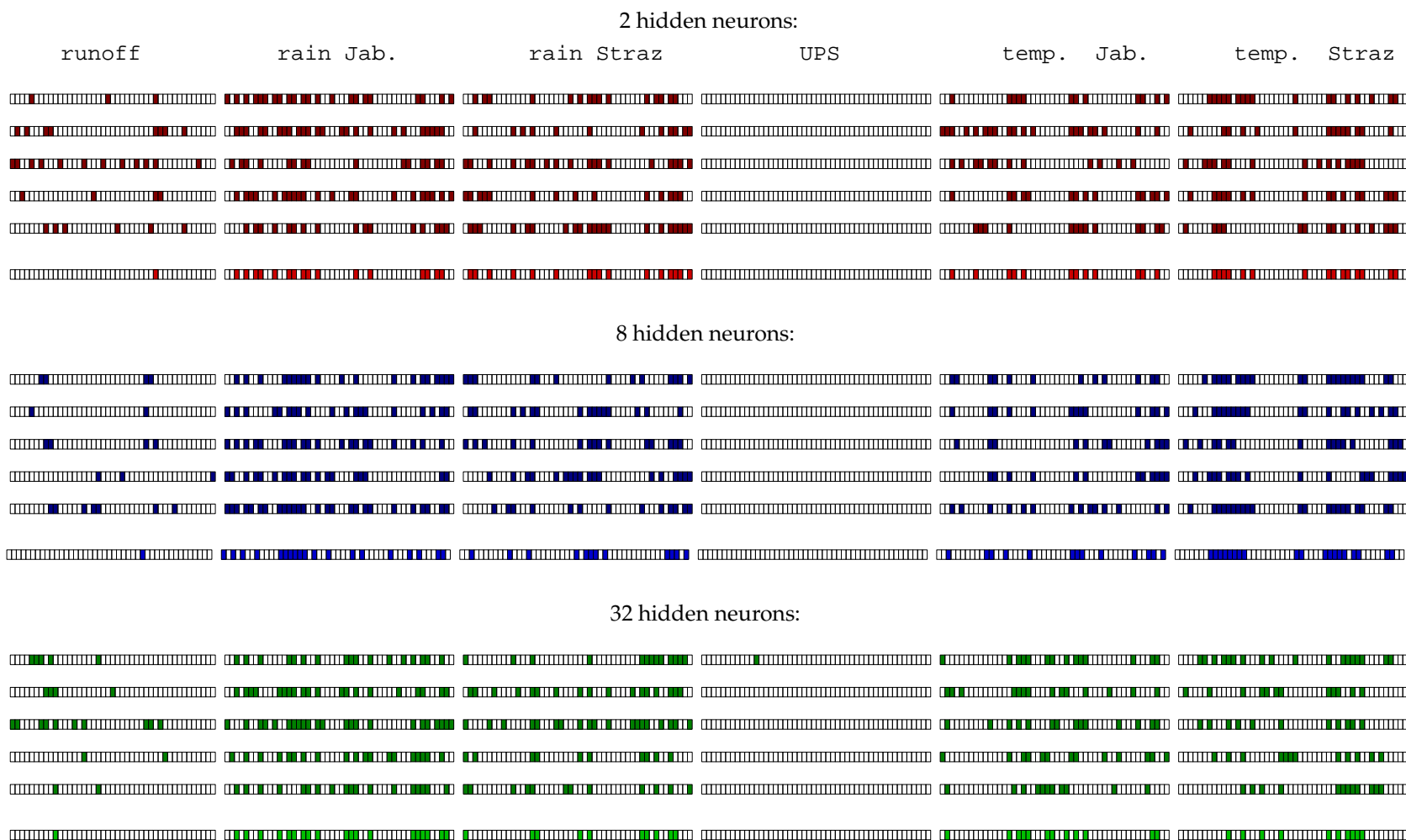


Figure 4.1: The results of the filter evolution experiment. The colored cells correspond to the chosen inputs. Every block represents the best filters obtained from 5 independent experiments performed using various number of hidden neurons. The lighter rows contains the most frequently chosen inputs for every group.

4.1.4 Results

The results of the three experiments are showed in the diagrams in the figure 4.1. Each color belongs to one network: red to the network with 2 hidden neurons, blue to 8 hidden neurons and green to 32 hidden neurons. Every row represents the final best individual of one particular run, divided into segments denoted to particular time series. The colored cells corresponds to the selected inputs while the white cells are the those omitted.

The experiment discovered several unexpected facts. It is obvious that the UPS coefficient showed to be dismissible or at least hardly exploitable by the neural models. From the fifteen runs an input was chosen from this segment only once apparently as a result of statistic discrepancy. Also the inputs from the time series of previous runoffs was not picked up very frequently. Both of these facts disagree with the results obtained from linear methods in 3.3 and 3.4.

The most frequently chosen inputs come under the segments of rain falls. The temperatures were selected quite often as well, again despite the expectation obtained from the linear models. The possible explanation of this discrepancy is either that the nonlinear character of neural models discovers dependencies hidden for the linear methods, or that higher degree of freedom of neural models makes them more susceptible to high level of noise which is present in the rain falls and temperature series compared with the UPS and runoff series. These hypothesis deserve closer examination that will be included in the next section.

Although the resulting filters emerged rather similar structures, they contain indispensable level of variance. Under each five rows representing the result of one experiment there is an extra row with the cells colored with a lighter tone of the same color. These rows were made as a summary of the five various resulting rows of every experiment.

The purpose of this summary is to extract the relevant information of the experiments and leave out the stochastic noise inevitably present in a particular solution. A cell is selected in the summarized solution if it is present at least in three of five runs. These three summarized filters are considered to be final output of this experiment.

4.1.5 Defocusing Level of Detail Filter

In the previous section a large-scale experiment of filter evolution was closely described. It is always useful, when so much effort is invested into one direction, to compare the resulting solution quality with a simple and straightforward method.

There is a frequently used expectation in the theory of time series which is used in simple statistical predicting methods like weighted moving average [2]. They assume that the relevance of values of a time series for the prediction purposes decreases (usually exponentially) to the past. This simple idea is

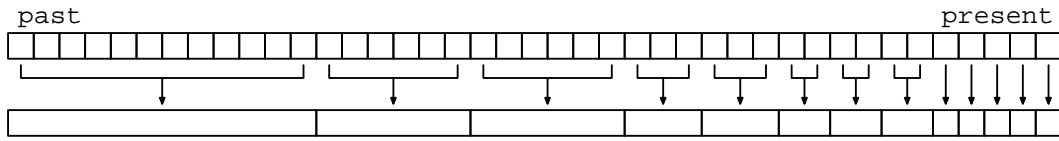


Figure 4.2: The scheme of defocusing detail filter construction.

used to construct summation filters which give the more recent values higher weight than those which are older.

Let us transform this idea slightly by introducing a level of detail as a measured value. When more recent inputs are more relevant than previous, than we can expect, that also more detailed timing is important for recent inputs than older ones. When information should be removed from the model in order to simplify it, let us sacrifice a time precision of older values first. Due to this assumption it is far more important if the rain stopped two or three hours before now, than if it had started 17 or 18 hours before.

The filter constructed according to this idea consists of fields computed as an average of segments containing original time series values. Those segments length grows approximately exponentially to the past. This mechanism is clearly illustrated in the scheme 4.2.

This simple filter, transforming the original time series into the series of its subsequences average values, will be used as a part of the following experiments together with the evolved filter to discover the contribution of the proper filter construction to the quality of the neural model.

4.2 Multilayer perceptron

The multilayer perceptron is the first in the line of computational intelligence models employed to learn to predict the 6 hours ahead runoff difference. It was chosen for two main reasons. Firstly, it is the most widely used neural architecture and thus good starting point for more complicated or specialized models construction. Secondly, as it was used as the fitness criterion of the filter evolution experiments performed in the previous section, it can quantify the benefit obtained from the filter evolution approach.

4.2.1 Model description

The model consists of one feed forward neural network. The network has one hidden layer containing various numbers of hidden neurons and one output neuron in the output layer. The hidden neurons are all of a same type realizing the sigmoidal transition function with normalized output which falls into the interval $[0,1]$.

The input weights of the neurons are randomly initialized by small real

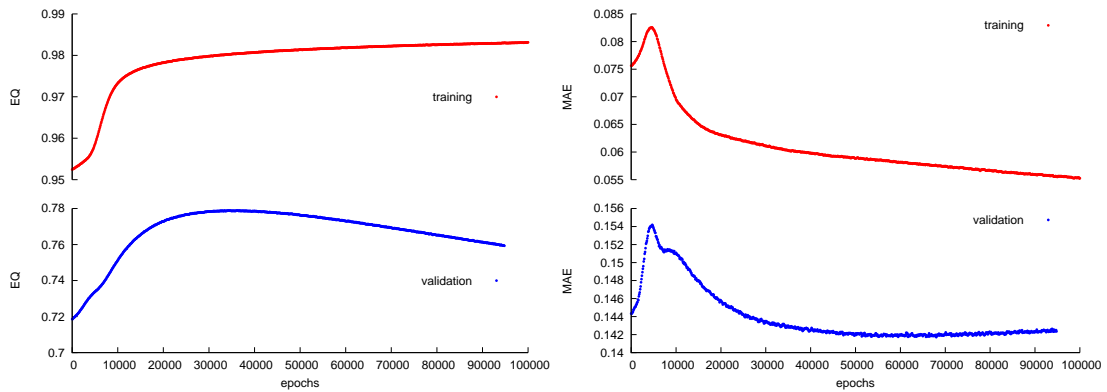


Figure 4.3: The course of mean absolute error and efficiency coefficient during the learning process computed for training and validation set.

numbers. A supervised learning method using the gradient optimization is employed to train the model. The standard back-propagation algorithm including the momentum improvement was selected for its numerical stability and implementation clarity. The online version of back-propagation algorithm was chosen i. e. the weights are updated separately after the gradient is computed for a single pattern.

4.2.2 Experiment

Four groups of experiments were performed for each of the three final filters obtained from the filter evolution process and for the detail defocusing filter introduced in section 4.1.5.

Every group contains five experiments each for a network with different number of neurons in the hidden layer. The smallest network has only 2 hidden neurons, while the largest has 50 neurons in its hidden layer.

The learning of a neural network which is a process of gradient optimization suffers, as every gradient method, from the occurrence of local optima. The gradient algorithm can not leave a minimum of error function. Due to the deterministic character of back-propagation algorithm, the final positions of the network in the weight space are determined by its initial position.

To avoid the effect of improper choice of the initial point, each experiment is repeated 10 times with randomly chosen initial points, and the result of the best run is considered as the result of the experiment.

4.2.3 Results

The four charts in the figure 4.3 show the typical learning progress. Four quality criteria are plotted in the charts: the mean absolute error and the efficiency coefficient, or in another words the coefficient of determination (see section 3.2), for both, training and validation set.

The point where the overtraining begins can be clearly identified at the maximum of the validation set efficiency coefficient approximately around the epoch number 30 000. To obtain the model with a minimal level of overtraining, the state of the model at this point should be taken as the resulting one.

The mean absolute error has also a similar minimum but it is located at the epoch number 60 000. The process of neural network learning using the back-propagation algorithm is a gradient descent on the mean squared error landscape. The efficiency coefficient is just a linear transform of the mean squared error and thus back-propagation directly minimizes it. The mean absolute error is a different metric which is similar but not identical to MSE hence its decrease is a side effect of learning and its extremes are located differently.

Table 4.1 summarizes the numerical results of the previously mentioned experiments. For all experiments the maximal validation efficiency coefficient was chosen and corresponding values of the other three quality criteria are listed in the table. For comparison, the results of simple prediction method which assumes zero progress of runoff are also included as the first row of the table.

The results of the experiments are very close to each other. However, all evolved filters achieved better validation efficiency coefficient than the simple defocusing filter for all neural network sizes. This supports the results of the filter evolution method and dissipates the doubts mentioned in section 4.1.4. In addition to that, all neural networks performed better than simple prediction method and the linear regression models (see 3.4.3) as well.

On the other hand, the level of mean absolute error is higher for neural models than for the others. Due to the previously mentioned reasons, back-propagation neural networks prefer MSE to MAE and so do hydrologists. MSE

TABLE 4.1: THE RESULTS OF MULTILAYER PERCEPTRON

n	defocusing detail filter				evolved filter – 2 hidden			
	MAE_t	EC_t	MAE_v	EC_v	MAE_t	EC_t	MAE_v	EC_v
–	0.0765	0.932	0.1084	0.713	0.0765	0.932	0.1084	0.713
2	0.0589	0.979	0.1403	0.766	0.0548	0.981	0.1403	0.776
5	0.0584	0.979	0.1407	0.765	0.0548	0.981	0.1401	0.776
10	0.0588	0.979	0.1409	0.766	0.0555	0.981	0.1392	0.776
20	0.0594	0.978	0.1409	0.769	0.0557	0.981	0.1409	0.773
50	0.0616	0.978	0.1450	0.771	0.0590	0.980	0.1426	0.778
n	evolved filter – 8 hidden				evolved filter – 32 hidden			
	MAE_t	EC_t	MAE_v	EC_v	MAE_t	EC_t	MAE_v	EC_v
2	0.0577	0.981	0.1394	0.775	0.0549	0.981	0.1341	0.773
5	0.0576	0.981	0.1401	0.776	0.0546	0.981	0.1346	0.771
10	0.0578	0.981	0.1406	0.774	0.0547	0.981	0.1342	0.771
20	0.0583	0.981	0.1405	0.776	0.0557	0.981	0.1357	0.772
50	0.0600	0.981	0.1427	0.779	0.0569	0.980	0.1371	0.775

is more affected by extreme values than MAE. For flood prediction purposes, it is more useful to have a model with better results on flood periods with higher variance and thus also higher errors than perfect results on long calm periods with low variance and also low hydrological importance.

4.3 Evolved Multilayer Perceptron

Promising results of previous experiment inspire to explore the potential of multilayer perceptron in more detail. The most serious problem of the back-propagation algorithms is the local optima entrapment as there was mentioned before. In order to avoid this problem, various stochastic methods were proposed including random weights perturbation or simulated annealing [10].

A more general approach was chosen in this experiment. The neural network is trained by the evolutionary algorithm. This method has a potential to reach the border of achievable solution quality for the particular model.

4.3.1 Model description

The model is formed by identical feed forward neural networks as in the case of previous section. The difference lies in the learning algorithm. A genetic population of neural networks is trained and evolved in the direction of the minimal mean squared error obtained on the training set.

The genetic individual is the neural network represented by the set of its neuron weights. The encoding is formed by the the vector of doubles that is closer to phenotype representation. The reason is a high complexity of neural network.

Two different types of mutation are involved. First is the *gaussian mutation* which modifies a single weight by adding random value from normal distribution with the center in zero and the standard deviation set to 0.1. However, this parameter is adaptive and it is a part of genome. This little example of meta-evolution makes a small adjustments of mutation disruption possible during the genetic algorithm run. The standard deviation of the mutation is mutated itself by gaussian mutation with fixed standard deviation.

Second one is a sort of *positive mutation*, something which does not happen in nature but could be prepared in the world of artificial evolution. From time to time, several epochs of back-propagation algorithm are performed on a randomly chosen individual. The recombination operator is on the contrary omitted at all as a problematic feature of most neural network evolution attempts.

Let us discuss the selection operator now. The modified version of *roulette selection* (see section 2.1.2) applied in this experiment assigns every individual the number of slots in the new population according to the proportion of its fitness with a very low level of variance. The weakness of this version of

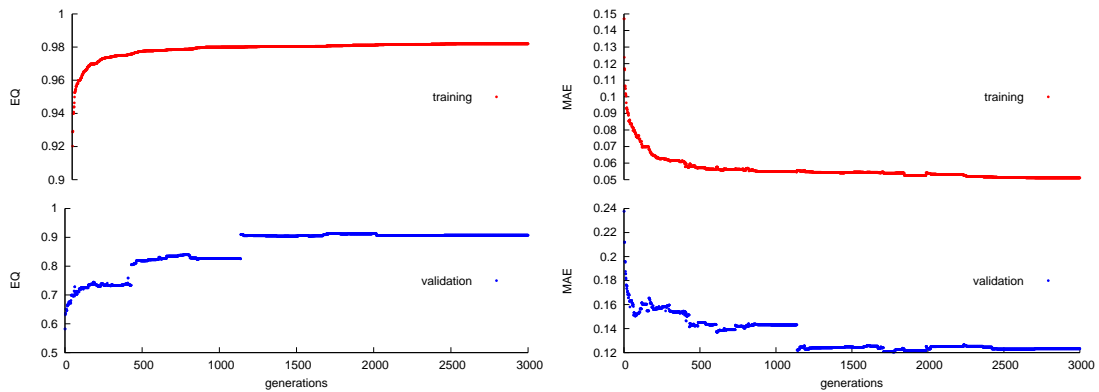


Figure 4.4: The course of mean absolute error and efficiency coefficient during the learning process of the best obtained solution computed for training and validation set.

proportional selection is balanced by the higher level of diversity preserved in a smaller population. To prevent the loss of a good solution when the population size is small and the selection is weak elitism preserving a single best individual is involved.

4.3.2 Experiment

Two sets of experiments were performed using the best evolved filter corresponding to the network with 8 hidden neurons and for comparison, the detail defocusing filter was used. All sets contain five experiments each learning different neural network. The types of neural networks are identical to the types used in the previous experiments in section 4.2.

In order to obtain valuable results, each experiment is performed 10 times. Due to a significantly higher level of stochasticity inherent with the genetic learning of neural networks, not the best but the second best result was chosen. If two of ten runs generate a solution of some quality or better, this result can be considered as reliable.

The number of generations performed by the genetic algorithm was set to 3 000 according to previous experience. For the reasons mentioned in the previous paragraph, the solution is not accepted until reaching generation 500, to ensure that the results are not a consequence of a random fluctuation.

4.3.3 Results

The results of the experiments show rather high level of entropy typical to evolutionary optimization. The charts in the figure 4.4 that belong to the best obtained solution ($EC_v = 0.913$) fall within those more ordered learning process courses.

Approximately half of the runs did not find a way to an acceptable solution. This clearly shows that multiple runs are inevitable while using evolutionary method for creating the final prediction model. On the other hand, this disadvantage is more than compensated with the best solutions quality.

The results of all experiments are listed in the table 4.2. Three interesting conclusions can be figured by comparing these results. Firstly, it is clear that the implemented version of neural networks evolutionary learning dominates the gradient learning in the matter of quality of obtained solutions, at least for this particular problem. All values of obtained efficient quality are significantly better than corresponding values in the table 4.1.

The second conclusion faces the relationship between the neural network size and the result quality. Due to the very high variance of the numerical results no reliable dependency can be discovered. The same holds also for the results of both used filters.

Finally, the relationship between the efficiency coefficient and the mean absolute error should be discussed. Both quality metrics begin to correlate one to another, in contrast with results of previous experiment listed in the table 4.1. This hints gradual approaching the final point, which is identical for both metrics.

4.4 RBF Input Layer Network

The motivation to employ the RBF network model is to explore the potential of a frequently used alternative neural network architecture based on local neural units. The radial basis function networks are known for their ability to properly transform the input and reduce its dimension. This feature is applied for example in the construction of classification models such as support vector machines.

The concept of RBF networks is used in the same way in this experiment: to reduce the dimension of the input space in order to reduce the noise, to speed up the training and to make it easier for the sigmoidal neural network to learn the proper input patterns.

TABLE 4.2: THE RESULTS OF EVOLVED PERCEPTRON

n	defocusing detail filter				evolved filter – 8 hidden			
	MAE _t	EC _t	MAE _v	EC _v	MAE _t	EC _t	MAE _v	EC _v
–	0.0765	0.932	0.1084	0.713	0.0765	0.932	0.1084	0.713
2	0.0533	0.978	0.1130	0.859	0.0612	0.978	0.1483	0.816
5	0.0670	0.974	0.1447	0.819	0.0584	0.979	0.1253	0.832
10	0.0570	0.980	0.1219	0.848	0.0638	0.977	0.1505	0.804
20	0.0576	0.981	0.1306	0.813	0.0616	0.977	0.1340	0.814
50	0.0641	0.977	0.1493	0.790	0.0653	0.976	0.1408	0.793

4.4.1 Model description

The model is a hybrid neural network combining the RBF and sigmoidal hidden neurons. The network contains two hidden layers. First of them is the RBF layer while the second one, together with the single neuron output layer, consists of sigmoidal neurons. Together they form a four layer neural network.

The *symmetrical multidimensional gaussian* described in section 2.1.1 is used as the kernel function of the RBF neurons. The weights are fixed to 1, which reduces the degree of freedom of the model as well as its complexity. The radius value is also fixed for the whole network. The centroids learning is performed using the Kohonen self organizing map briefly introduced in section 2.1.1.

If the number of neurons in the Kohonen grid is lower than the space dimension, the space is reduced while preserving maximum of the information contained in the input set. On the contrary, higher number of grid's neurons can discover patterns hidden in the inputs.

In case of this experiment the first transformation is used. The Kohonen network is trained on the input set and the positions of the grid neurons are then used as the centroids of radial basis units in the input layer of constructed hybrid neural network.

The hybrid neural model was introduced in a few previous paragraphs from the conceptual point of view as the four layers neural network with two different hidden layers. Due to the fact, that the lower (RBF) layer is not modified during the process of learning when being initiated with the neuron centroids from the Kohonen grid and having the weights and radii fixed, the training adapts only the second hidden and the single neuron output layer. That is why the RBF layer output can be computed only once and the training of the sigmoidal hidden layer can be performed using this output as its fixed input. The learning complexity of this network is then reduced to the level of a standard three layer perceptron learning.

4.4.2 Experiment

As in case of previous experiments, two input filters were applied to the input set of these experiments: the 8 hidden neurons evolved filter and the detail defocusing filter (see section 4.1).

The experiments were performed for various numbers of neurons in both hidden layers: 8, 16, 24, 36, and 48 RBF neurons in the first hidden layer and 5, 10, 20, and 30 sigmoidal neurons in the second hidden layer. Due to the capacity reasons, not all possible combinations were tried. The empirical *pyramid rule* [10] suggesting the number of neurons in the second hidden layer to be approximately equal to the half of the number of the neurons in the first hidden layer was roughly followed. In spite of this reduction still 13 experiments were performed. Each experiment consists of five runs. The best result was accepted.

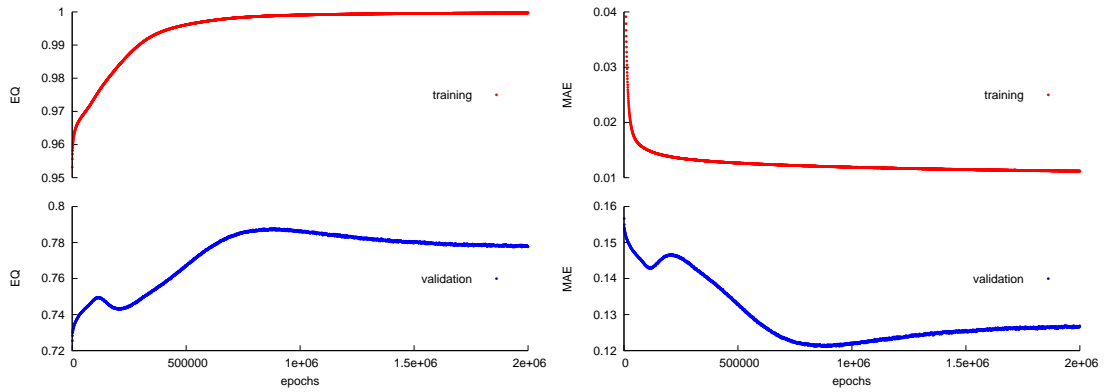


Figure 4.5: The course of mean absolute error and efficiency coefficient during the learning process of hybrid RBF network computed for training and validation set.

4.4.3 Results

There was not a typical run for this experiment, in which most of the error function courses differ one run from another. Due to the used gradient method, the curves are smooth but most of the validation error curves have an irregular course with a few local extremes such as those plotted on the figure 4.5.

The numerical results are listed in the table 4.3. The value of the efficiency coefficient falls within the interval 0.72 – 0.80 with a single extreme being of the value 0.85. This isolated extreme should be classified as a randomly obtained value and thus neglected from the consideration.

The complexity of the models grows from the top to the bottom. In general the better results were obtained from the more complex models, listed in the

neurons		defocusing detail filter				evolved filter – 8 hidden			
r	n	MAE _t	EC _t	MAE _v	EC _v	MAE _t	EC _t	MAE _v	EC _v
—	—	0.0765	0.932	0.1084	0.713	0.0765	0.932	0.1084	0.713
8	5	0.0137	0.964	0.1466	0.743	0.0545	0.953	0.1568	0.725
8	10	0.0164	0.963	0.1441	0.747	0.0434	0.949	0.1542	0.729
16	5	0.0154	0.972	0.1219	0.786	0.0193	0.965	0.1566	0.725
16	10	0.0138	0.972	0.1218	0.786	0.0348	0.950	0.1488	0.739
16	20	0.0134	0.973	0.1177	0.793	0.0194	0.959	0.1549	0.728
24	10	0.0154	0.967	0.1277	0.776	0.0507	0.944	0.1496	0.738
24	20	0.0207	0.963	0.1348	0.764	0.0516	0.951	0.1582	0.723
24	30	0.0124	0.993	0.1306	0.770	0.0142	0.968	0.1458	0.745
36	10	0.0121	0.997	0.0864	0.849	0.0192	0.968	0.1513	0.735
36	20	0.0179	0.969	0.1464	0.743	0.0155	0.967	0.1490	0.739
36	30	0.0254	0.968	0.1371	0.759	0.0109	0.999	0.1131	0.802
48	20	0.0194	0.975	0.1394	0.756	0.0118	1.000	0.1129	0.802
48	30	0.0210	0.966	0.1411	0.753	0.0120	0.999	0.1211	0.788

lower part of the table, compared to the ones gained from the smaller sized models in the upper part.

An interesting conclusion can be made about the comparison between the results obtained by both used filters. While the defocusing filter scored better for the first 10 rows of the table, the evolved filter dominated for the last three rows. This can be caused by the better resistance of the evolved filter to the overtraining. This hypothesis is supported by the positive correlation between the training and validation efficiency coefficient values of this model in contrast with the multilayer perceptron model (see section 4.2).

Finally, the comparison of the results of RBF Input Filter networks and the canonical multilayer perceptron is concluded: the results of larger RBF networks are slightly better than the results of the multilayer perceptron, on the other hand, the variance of the results is higher as well.

4.5 Summary

In this chapter, a large set of computational experiments was performed in order to explore the possibilities to create a connectionist model with the desired prediction ability. The conclusions of the correlation and the regression analyses described in the statistical part of the thesis (see chapter 3) were taken into the account.

The most computationally demanding experiment – the input filter evolution – was intended to determine the the set of relevant input values. Final results, however noisy, showed strong impact of the dynamic time series of the rainfalls and the air temperatures in contrast with the aggregated series such as the runoff and the UPS (see 4.1). As a comparative concept to the input filter evolution, the simple detail defocusing filter construction inspired by the statistical methods was introduced and then used together with the evolved filter by the neural models.

The first neural model trained to predict the 6 hours ahead runoff volume was the canonical multilayer perceptron trained by the back-propagation algorithm with momentum. The quality of the results was measured using the efficiency coefficient (see section 3.2). The experiments for various number of hidden neurons were performed, each containing 10 runs to screen out the effect of random initiation of the network. The best results of each run were accepted.

During the training the phenomenon of overlearning appeared, which became the main limitation of the model performance. The best obtained value of the efficiency coefficient on the validation set was 0.78 while using the evolved filter and 0.76 for the detail defocusing filter. This quality of results dominates both, the prediction by the current runoff having the EC equal to 0.71 and the linear regression with $EC = 0.75$.

Another learning algorithm was employed in section 4.3 to train the multilayer perceptron in the second set of experiments. The neural networks were

evolved by the genetic algorithm. The training is slower compared to the gradient descent method but it produces models of a higher quality.

The efficiency coefficient varies from 0.79 to 0.86. The results are very noisy and the part of the success of this method is caused by the stochastic variance however. When taking the best solution from a very noisy sequence, it is possible its quality is more caused by the noise itself than the training.

A different neural model were introduced in section 4.4. The hybrid neural network with two hidden layers. The first layer consists of RBF neurons trained by Kohonen vector quantization, the second layer is formed by standard sigmoidal neurons trained by the back-propagation. The results of this model learning suffer from a higher level of variance than the results of multi-layer perceptron and also the quality of the best obtained models is better. The efficiency coefficient varies from 0.72 to 0.80.

A positive conclusion from these three experiments can be made that the neural models are in general able to deal with the short term runoff prediction problem better than the linear regression or the referential simple predictor.

Another conclusion is not as positive: a high performance of a neural model can be achieved only at the expense of involving a high level of stochastic variance into the learning process. This however decrease the validity of the acquired solution quality measurement. To overcome this difficulty another sort of models based on composite modular neural networks will be explored in the next chapter of the thesis.

Chapter 5

Ensemble models

The whole is more than the sum of its parts.

Aristotle, Metaphysics

In the previous chapter a set of neural network models were introduced and employed in order to perform the nonlinear regression task of short term rainfall – runoff prediction. The models showed to be promising but they suffer from the discrepancy between the quality of the model and its creation reliability.

Joining individual predictors into a modular architecture is supposed to improve the generalization ability as well as reduce the variance of the predictions compared to the performance of any of its members [16].

The main purpose of this chapter is to explore the potential of the neural network ensembles to balance this trade-off, i. e. to provide models having high level quality and also low level of resulting model variance, supporting the belief, that the model of this quality can be recreated whenever after.

5.1 Bagging

The first method of training neural ensembles is a technique called *bagging*[16] which was briefly introduced in section 2.1.3. In this experiment a slightly modified version of bagging algorithm will be used. This modification is motivated by the effort to reduce the overtraining inherently present in the learning process of the individual networks.

5.1.1 Model description

The neural regression ensemble based on the modified bagging method is trained using the algorithm 4. In another words, the networks are trained on randomly chosen half of the original training set. The second half of the training set is used as the validation set. During the training, the error over

Algorithm 4 Modified Bagging Ensemble Construction

Input: Training set T , Ensemble size n , Oversize coefficient c , Iterations counts it_1, it_2 .

Output: Ensemble E .

```
E ← empty ensemble
for i ← 1 to c × n do
  Tt, Tv ← randomly split T
  N ← randomly initiated neural network
  for j ← 1 to it1 do
    N.train(Tt, it2)
    Compute error et of N w. r. t. Tt
    Compute error ev of N w. r. t. Tv
    if ev is minimal yet obtained then
      Nbest ← N
    end if
  end for
  E ← E ∪ {Nbest}
  if |E| > n then
    Nworst ← N ∈ E: error of N above T is maximal
    E ← E \ Nworst
  end if
end for
return E
```

the validation set is periodically computed. The state of the network with the minimal value of the validation error is added into the ensemble.

In order to avoid the poorly learned networks to be added into the ensemble, which could damage its performance, the number of trained networks exceeds the final ensemble size. Only networks with the lowest error computed over the whole training set are involved into the model. The parameter c denoted as the *oversize coefficient* determines the multiple of the trained networks compared to the ensemble size e .

The ensemble consists of identical individual networks, all of them are multilayer perceptrons with equal number of hidden networks. The final ensemble output is computed as arithmetic mean of the outputs of individual networks.

5.1.2 Experiment

A set of experiments were performed using various ensemble sizes and numbers of the individual network hidden layer neurons. Each experiment was performed 10 times and the result of best run was recorded.

An original intention was to make a similar two experimental groups as those contained in the chapter 4, but after a first few experiments it became

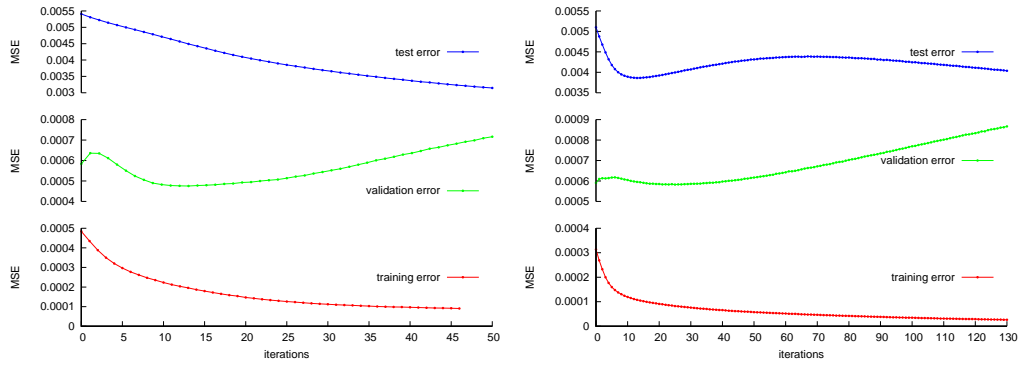


Figure 5.1: The course of mean squared error during the learning process of two arbitrary chosen networks of the ensemble with 36 networks and 10 hidden neurons per network. The red curve represents the error on the training half of the training set, while the course of error computed for validation half of the training set is plotted in green. The blue curve shows the sequence of errors computed for the original validation set.

clear that the performance of both filters is similar with a very little superiority ($\delta EC \doteq 0.05$) of the defocusing filter. Thus the rest of the experiments on the evolved filter were not completed.

5.1.3 Results

The numerical results of the experiments are listed in the table 5.1. Together with the training and the validation values of the efficiency coefficient another couple of these values is computed in order to discover the potential of the ensemble.

Let us assume we have an oracle suggesting the choice of the best network from the ensemble for every single input pattern in the training and validation set, i. e. the network with the lowest error for a particular pattern. The values of efficiency coefficient obtained when the optimal network is used for every input are listed in the column denoted as the *optimal choice*. The standard ensemble outputs are listed in the column called *over-all average*. The superior quality of the optimal results suggest that there is a potential in the ensemble which is not utilized completely by the averaging method.

The variance of the resulting EC is significantly lower compared to the single network models. The same holds for the variance among results inside a single run. The bagging ensemble clearly reduces the variance of the validation error.

The performance of the model is on the other hand lower than the performance achieved with a majority of the single network models. The EC_v values are practically identical to those of multilayer perceptron when using the defocusing filter.

The only difference is, that although the results were obtained on the same

set, in the case of the section 4.2, the value of the efficiency coefficient has slightly different meaning. While the same set was used as the validation set previously, it was used as the test set in the case of this experiment. A higher relevance of this experiment result thus could have been concluded unless a new rather discouraging fact appeared.

Using only a half of the training set to teach the individual neural networks made it possible to use the second half of the training set as the validation set to reduce the overtraining. That released the original validation set which could have been used as a test set. Up to now, the stability of validation set results were silently expected. This belief is however undermined by the results obtained from this experiment.

The two charts in the figure 5.1 present arbitrary chosen courses of the mean squared error on the three previously mentioned sets obtained from the experiment with ensemble of size 36. While the discrepancy between the training and the validation error course is well known from the previous experiments in chapter 4, the validation — testing discrepancy is rather surprising. The charts of the remaining 34 networks are similar in the sense that hardly any of them has a parallel-like course of those two curves.

The idea of using validation set to avoid the overtraining is based on a common expectation that stopping the training process of a neural network when the validation error starts rising while the training error decreases gradually provides a good chance to obtain a network with suitable level of generalization. The experiment did not confirm that expectation in this case, when the independent validation sets have its minimal values in completely different points of the training process.

Three points can be concluded from this corollary. Firstly, it is another proof of previously mentioned enormous inner variance of this sort of time series (see 3.1). Secondly, a larger training set if available could be a solution of this problem. Finally, if the measured model performance differs when two independent validation sets are used, the performance level on real data can not be guaranteed either.

TABLE 5.1: THE RESULTS OF BAGGING ENSEMBLE

params.		optimal choice		over-all average			
e	n	EC _t	EC _v	MAE _t	EC _t	MAE _v	EC _v
—		0.932	0.713	0.0765	0.932	0.1084	0.713
8	5	0.994	0.806	0.0633	0.977	0.1352	0.768
8	10	0.994	0.822	0.0648	0.976	0.1406	0.760
16	10	0.995	0.816	0.0799	0.975	0.1475	0.764
24	10	0.995	0.831	0.0694	0.977	0.1418	0.766
36	10	0.997	0.849	0.0610	0.978	0.1361	0.767
36	20	0.996	0.841	0.0630	0.976	0.1410	0.761

5.2 Self-Organizing Ensemble

The idea of bootstrap aggregation introduced in the previous section consists of two main concepts: the diversification of the ensemble and the combination of the results of the networks. Let us make another step forward in both directions. In the first phase of the learning process, fundamentally different networks will be created in order to be in the second phase combined in more sophisticated way compared to the simple averaging.

5.2.1 Model description

In the case of bagging, the diversity of the ensemble is derived from the diversity of the training set. The same approach is used in this model, only in its stronger form. The original training set is partitioned using the Kohonen self organizing map, briefly introduced in the section 2.1.1.

The Kohonen grid is deployed into the input space to represent the training set. The set is split into a group of disjunctive subsets afterward, each containing the training patterns represented by the same Kohonen neuron.

Each such subset is used to train one partial network of the ensemble. This learning method creates an ensemble containing very different networks, each specialized to a part of the input space. Two alternative methods are implemented to combine the outputs of the individual experts, constructing the final answer of the whole ensemble of any particular input pattern.

The first method is quite straightforward and it utilizes the way how the ensemble was composed. An input pattern is compared to the neurons in the Kohonen grid and the closest neuron is identified. The output of the network trained on the subset represented by this neuron is chosen to be the final ensemble output. The performance of this method depends on an accord between the Kohonen partitioning and the natural structure of the problem space. An advantage of this method is an absence of the validation set during the training process.

The second method of combining the outputs of particular experts is known as the *Mixture of Experts*[13]. The current model is extended by another combination neural network called *gating network*. This network is trained to determine the optimal selection of the ensemble member to be used for a particular input.

The input of the gating network consists of the same input vector as for the individual experts and its output is a vector of weights. The final output of the ensemble is computed as the weighted sum of the outputs of the individual networks according to this vector.

In this particular implementation, the gating network is trained to the vectors containing the normalized reciprocals of the mean squared errors of the individual networks computed for a particular input from the training set. During the training process of the gating network, the validation error of the

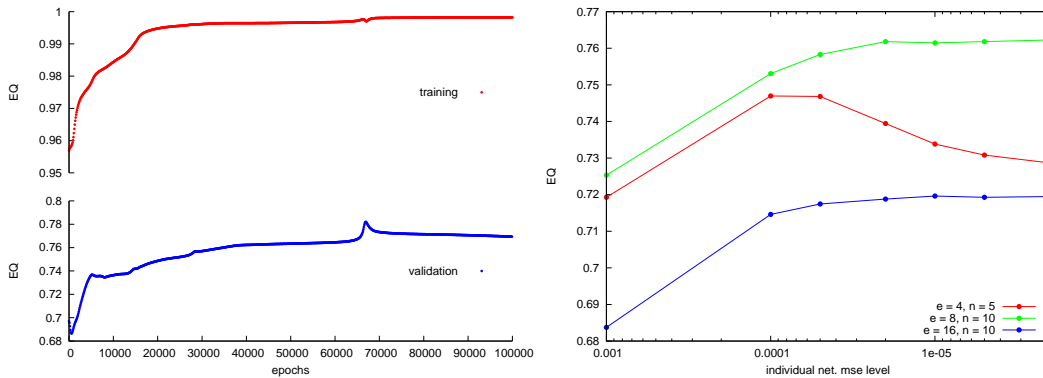


Figure 5.2: Left: the course of training and validation efficiency coefficient during the learning process of the network mixture. Right: the efficiency coefficient of the self-org. outputs for various levels of individual networks precision.

ensemble is computed and the best gained value is recorded as the model performance.

5.2.2 Experiment

This experiment contains the highest number of degrees of freedom in a form of model parameters as well as the training algorithm setting. The ensembles consist of multilayer perceptrons trained by the back-propagation algorithm with the momentum.

Each experiment is characterized by the ensemble size (will be denoted as e), the number of neurons in the hidden layer of individual networks (n) and the number of neurons in the gating network's hidden layer (g). Various combinations of these three parameters were used.

Every experiment consists of several phases. During each phase the ensemble individual networks were trained using their piece of training set to achieve higher level of training mean squared error. The desired error level was gradually decreasing according to the exponential sequence: $1 \cdot 10^{-3}$, $1 \cdot 10^{-4}$, $5 \cdot 10^{-5}$, $2 \cdot 10^{-5}$, $1 \cdot 10^{-5}$, $5 \cdot 10^{-6}$, and $2 \cdot 10^{-6}$. For every error level achieved, the gating network were randomly initialized and retrained. The best obtained validation result was recorded. This multi-phase training algorithm was involved in order to avoid the overlearning on the level of individual networks.

As well as in the previous cases, all the experiments were performed in ten runs recording the best run performance on the validation set to obtain as reliable results as possible.

5.2.3 Results

The numerical results of this experiments are listed in the table 5.2 together for both methods dealing with the combination of individual networks.

The Kohonen combination output reached the best value of efficiency coefficient equal to 0.76, which is worse than the results of a single network models examined in chapter 4. The right chart in the figure 5.2 suggests an existence of a kind of limitation of the model performance reached for quite low level of individual networks precision rather than the overlearning of the individual models. The deeper analysis of the model performance showed that the Kohonen combination method was not able to recognize the optimal network for most of the input patterns of the validation set. For example during the best run only a 30% of validation input patterns were processed using the optimal set compared to 85% for the training set patterns. It seems to be clear, that if there is a good partitioning of the input space which reduces the problem complexity, it was not found using the Kohonen quantization. The examination of this field, however tempting, over-exceeds the possibilities of this thesis.

The results of the combination using the gating network showed to have better performance compared to Kohonen combination and also the results of the bagging ensemble (see table 5.1). On the other hand the best value of EC equal to 0.78 together with the second best of 0.77 does not make this experiment results better than those of single neural models.

The smooth-less course of the error curves such as those two plotted in the left chart in the figure 5.2 shows that the gating network was operating above a more complex error landscape compared to previous experiments. This could be interpreted as that the training set decomposition was not performed optimally. On the other hand, adding an extra hidden neurons into the gating network did not bring explicit improvement.

TABLE 5.2: THE RESULTS OF TRAINED KOHONEN ENSEMBLE

params			optimal	Kohonen	weighted output			
e	n	g	EC _v	EC _v	MAE _t	EC _t	MAE _v	EC _v
—			0.932	0.713	0.0765	0.932	0.1084	0.713
4	5	10	0.766	0.747	0.0049	1.000	0.1427	0.763
4	5	20	0.766	0.747	0.0310	0.991	0.1502	0.772
4	10	10	0.757	0.743	0.0136	0.999	0.1527	0.772
8	5	10	0.792	0.737	0.0199	0.996	0.1339	0.770
8	10	10	0.833	0.762	0.0174	0.998	0.1474	0.774
16	5	10	0.843	0.750	0.0167	0.997	0.1607	0.782
16	5	20	0.843	0.750	0.0320	0.981	0.1378	0.768
16	10	10	0.831	0.735	0.0175	0.997	0.1529	0.767
24	5	10	0.845	0.742	0.0282	0.992	0.1436	0.754
24	10	10	0.862	0.748	0.0227	0.992	0.1580	0.753
36	10	10	0.879	0.718	0.0252	0.990	0.1491	0.750
36	10	20	0.879	0.718	0.0200	0.989	0.1405	0.769

5.3 Summary

The goals defined in the introduction of this section were satisfied only partially. The ensemble based on the modified version of bagging showed an extraordinary stability of the solution quality. The resulting efficiency coefficient equal to 0.76 however stable is on the other hand lower, than the results obtained from the single network models in the chapter 4.

The more sophisticated modular self-organizing mixture of neural networks which uses the Kohonen quantization to partition the training set was only slightly better. The resulting best efficiency coefficient for this model varies around 0.77 which does not exceed the quality of a single multilayer perceptron model (see 4.2).

Both experiments suffer from the small sizes of the training and validation sets which was clearly demonstrated by the discrepancy between the results obtained on two independent validation sets used during the bagging ensemble training. Unless a larger training set is available, the examined ensemble methods do not bring any additional value to the rainfall runoff modeling problem.

Chapter 6

Final Model Test

A multitude of rulers is not a good thing. Let there be one ruler, one king.

Herodotus

In the two main chapters 4 and 5, various computational intelligence models together with the regression analysis in chapter 3, were described and a large set of experiments was performed. The experiments determined the models ability to understand the internal dependencies hidden in the rainfall-runoff time series and predict the future values of the runoff. By comparing the performance of the models a broad estimate arised about the relative quality of the models.

The experiments however did not provide the answer to the essential question about the generalization ability of the models i. e. the ability to generate a valuable results on the data, that were not used during the learning process. A commonly used technique to examine the expected quality of the model results when used to predict real data is the *training – testing set* experiment. It is slightly different method than the usage of a validation set in the way it has been done in the previous chapters.

The validation set serves mainly to avoid the overlearning phenomenon described in section 4.1. When data are homogeneous enough, the validation results correspond more than less to the final expected performance of the model. The particular rainfall-runoff series examined in this thesis however do not belong among this sort as there was demonstrated in the section 5.1.

Ideally, the three set (train – validate – test) experiments would have been performed in the experimental chapters. Unfortunately, the very decent size of the input set covering just a summer period of 2007 was hardly even large enough for the training itself.

This is not the only generalization issue to be faced. Above the previously described rigorous methodology another level can be indicated. Every computational intelligence model introduced before demands a kind of tuning which consists of setting various minor parameters such as learning iteration numbers, a gradient step size or genetic operators probabilities. Majority of them

were mentioned only marginally or not at all because an effect of any such parameter is not very significant in most cases. Although, when tuning the models using a particular dataset, there is always a little danger of making them too fragile and unprepared for the outer world.

Shortly before the finish of the thesis a new set of data was finally yielded from the CHMI. Those data covers the period of years 2008 and 2009. Together with the already used series from 2007, the total size of training set tripled. This makes it possible to perform the final set of experiments in order to compare the model performance and to get insight into the real performance of the examined models.

6.1 Experiment

The time series were sampled in the way described in the section 3.4. This set of time points was randomly split into three disjunctive sets: training set, validation set and test set containing 50% respectively 25% and 25% of the points.

For every proposed model one configuration of parameters was chosen, usually the one having the best validation efficiency coefficient during the previous experiments. No additional model tuning were performed, the models were chosen exactly as they were used in chapter 4 or 5 respectively.

In order to yield the statistically relevant results, a hundred experiment runs were performed. Ten of them having the best validation results were chosen. The corresponding performance on the test set was measured and the average of the results was computed. This result is considered to be the final performance of the model. All models were trained using the defocusing filter as there were not a significant difference in performance for most of them.

model	configuration	EC	MAE	MSE _{pd}	time**
last value estimation		0.938	0.1036	0%	0:00
linear regression	$\alpha = 0.01$	0.937	0.1240	-1%	0:00
multilayer perceptron	$n = 10$	0.944	0.1252	10%	0:01
evolved perceptron	$n = 10$	0.947	0.1200	16%	0:18
rbf hybrid network	$r = 16, n = 10$	0.909	0.1432	-44%	0:19
bagging ensemble	$e = 16, n = 10$	0.947	0.1138	15%	0:24
self-org. ens. (koh.)*	$e = 8, n = 10, g = 10$	0.931	0.1392	-10%	1:44
self-org. ens. (gat.)*	$e = 8, n = 10, g = 10$	0.930	0.1392	-12%	1:32

* for the enormous computational time only 10 runs were performed and the best run was chosen

** in hours and minutes

6.2 Results

The numerical results are listed in the table 6.1. For every model, the obtained efficiency coefficient and mean absolute error are listed. In addition, the percentage decrease of the mean squared error compared to the last value estimation i. e. the estimation of zero change of the runoff is computed. This information clearly explains the benefit from the usage of a particular model compared to the simple prediction method.

The models were obviously divided by the test into two separated groups: those that succeeded and those that failed. All models based on the multilayer perceptron architecture belong to the first group while all the other models in spite of their superior results during the training fall into the second group.

The conclusion can be figured that the classical multilayer perceptron known for its robustness is the only neural architecture – of those examined in this thesis – suitable to deal with such a heterogeneous behavior of the examined data. The models based on the Kohonen quantization and RBF units proved not to be able to generalize the trained knowledge. In section 4.3 a doubt was expressed about the validity of the superior performance provided by the evolved perceptron. The final test, however, proved the quality of this model.

Together with the experimental results, the approximate run times of the model training process is included in the table. The run times were measured on a single core of these days average personal computer. The method of neuroevolution parallelisation was not used, thus the evolved perceptron training can be speed up as well as the ensemble methods.

Chapter 7

Implementation notes

A designer knows he has achieved perfection not when there is nothing left to add, but when there is nothing left to take away.

Antoine de Saint-Exupery

An important part of the work on this thesis was the design and the implementation of an *artificial intelligence library* providing the necessary computational framework. All the computational experiments listed in the chapters 4 and 5 were performed using this library. This thesis is focused mainly to the prediction performance of the computation models and thus mostly operates on the higher level of abstraction corresponding to the theoretical computer science. In spite of that, a very brief view into the design of the underlying computation infrastructure can provide the reader a valuable piece of knowledge.

7.1 The Design Analysis

The serious software engineering analysis fundamentally over-exceeds the possibilities of this thesis. In this section, only a few most important concepts will be introduced.

7.1.1 Compiled or Interpreted

The crucial question, when designing a new piece of software is the choice of the programming language and run-time environment. Several demands were taken into the consideration.

1. The chosen language should support a fast development. According to the expectation – which has been later fully confirmed – that most of the time will be spend by designing the experiments. It was vital that the ring: write, test, improve, will be as fast as possible.
2. The language should be reasonably computationally fast.

3. The parallel programming have to be supported by the language and the environment on a suitable level of performance as well as simplicity.
4. The platform independent language is vital while various architectures including supercomputers are expected to be used.
5. The high level of error checking is strongly welcomed. In spite of the conventional programming, where the wrong code is simply identified by the wrong behavior of the program, the artificial intelligence experiments often just suffer from low performance. Plenty of experience show that every automatic error check provided by the language will be repayed many times in the form of saved testing time.

After considering these requirements, the Java was chosen as the language for the library implementation. Thanks to the fully objective model, Java provides a very high level of the code reusability, which makes the development very fast. The only faster development is possible using a specialized language such as MatLab with all its libraries. MatLab, however, does not support parallel computation on a suitable level and thus is not proper for the case of this thesis.

The computational speed is not the strongest side of Java language or rather most of its implementations. On the other hand, according to several studies performed recently, its speed increases gradually and for several numerical problems its speed is comparable to c or c++ on many architectures.

On the field of parallel computing, Java is definitely the best choice. Firstly, the multithread programming is supported on the level of Java Virtual Machine which makes writing the multithread computational applications simple, or at least possible without usual months of debugging and deadlock resolving. Secondly, the Java supports simple, efficient, and reliable framework for the network communication via TCP/IP protocol, that creates the opportunity of writing applications for distributed computation. Both of these features are naturally present in most of the modern programming languages, the Java however excels in the field of design clarity, simplicity and proper level of abstraction provided.

Java is platform independent language operating literally almost on every washing machine. However, more important for this thesis is that the same code is suitable for running a parallel evolution task on a dual core notebook as well as a multicore supercomputer with no demand of additional libraries.

Finally, the error checking as a kind of bonus should be mentioned. The truly modern and objective languages such as Java (or C#, as its distant clone) have an advantage of the design clarity which makes it possible to automatically catch plenty of programming mistakes or just highlight problematic pieces of code. This kind of additional control reduces the debugging time and increases the programming performance. The Java also contains valuable framework of JUnit tests, which delivers a simple way of defining the partial unit tests to improve the resulting code quality and reliability.

7.1.2 Functionality

The library consists of several parts, each compiled (in the sense of bytecode compilation) into a particular jar library. Let us briefly introduce the functionality:

Neural Networks

This library contains the implementation of various neural models. The feed-forward neural networks are represented by Multilayer perceptron (see section 2.1.1) in a form of back-propagation learning as well as conjugate gradient learning implemented according to [10], the RBF layer network and the cascade correlation architecture. The library contains also the implementation of ensembles used in this thesis as well as Kohonen neural network (see section 2.1.1).

Evolution

The main part of this package is formed by the variants of the genetic algorithm (see section 2.1.2) together with various genetic operators. A three variants of genetic algorithm are implemented, first for the usage on a serial machine, second for the thread parallelisation suitable for multicore machines, the third is intended for the distributed evolution to be performed on a cluster of computers communicating over a network.

Client-Server Distributed Framework

Two jar libraries – the Sever and the Client – were written in order to perform distributed computations based on the barrier synchronization primitive[1] (a two phases computation-communication method synchronizing the machines after each communication phase). These libraries form a framework to be used by particular computational libraries such as the distributed evolution mentioned before.

7.2 The Library Employment

The library was intensively used during the work on the thesis mainly during the internship within the scope of the HPC Europa 2 program at the EPCC department of the Edinburgh University, UK. Thanks to the internship program, the access to HPCx supercomputer (recently the world 16th according to Top 500 statistics) was possible.

The supercomputer consists of 160 nodes, each containing 16 cores. The condition of being allowed to use such a powerful and busy machine were to justify the efficient usage of the allocated cores. That is why a combined distributed genetic algorithm had to be designed.

The problem of input filter evolution closely described in section 4.1 was solved using this method. As the fitness evaluation was the main source of the computation cost, this part of the algorithm run was distributed between the cores.

A method of bulk processing was chosen. The multithread server ran on one node communicating via TCP/IP with the clients, each one running on a single node. The genetic algorithm distributed the packages containing 16 filters among the nodes. Then the clients performed parallel processing of the fitness evaluation on these filters during a multithread run on the 16 cores that belong to every single node. Finally, the results were gathered in the same way.

The genetic algorithm is thanks to the population concept described in the section 2.1.2 easily able to be parallelized. This parallelisation is however limited by the population size. That is why no more than 176 cores were used a during single run. On the other hand, the queuing policy of the HPCx supercomputer batch system is configured in a way, it prefers smaller and shorter jobs compared to those which are larger and more time consuming. The medium size of such genetic jobs showed to be a good choice.

For the neuro-evolution task described in section 4.3 was not necessary to employ such a massive computational force. The networks were evolved by the multithread variant of the genetic algorithm using either a single node of the supercomputer or a smaller 16 cores shared memory server provided by the EPCC department.

Chapter 8

Final Summary

Prediction is very difficult, especially about the future.

Niels Bohr

In this thesis a computational intelligence approach was employed to deal with the rainfall-runoff modeling. The problem of short term (6 hrs.) runoff prediction as a form of time series forecasting was closely introduced in chapter 3 together with the relevant performance criteria.

The quality of results was measured mainly using the quadratic *efficiency coefficient* (EC – see section 3.2) and the linear mean absolute error (MAE). The quadratic error metric was chosen to be the prior as it provides more expressing comparisons from the hydrological point of view. While it puts a stronger weight on higher error values corresponding to the wilder periods of the runoff course, it accords the main focus of the flood predictor.

Six hourly measured input time series including the rainfall, cumulative rainfall, air temperatures and the series of the previous runoffs were introduced in the section 3.1. The statistical analysis of the series showed very high level of internal variance and heterogeneity in the rainfalls and runoff series.

The correlation analysis, performed in section 3.3 in order to explore the relationship between the input series and the series of runoffs, showed crucial dependency on the rainfall series together with complementary impact of the previous runoffs and the cumulative rainfall. The following regression analysis (see sec. 3.4) confirmed the role of rainfall series and it employed also the aggregated series. Both linear methods ignored an information possibly hidden in the air temperatures.

In chapter 4 the experiment of the input filter evolution was designed to determine the relevant input values in order to reduce the input dimension and thus also the model complexity (see section 4.1). The extremely computation-intensive experiment clearly showed dismissible role of the cumulative rainfall series and it confirmed the relevance of the rainfalls. In contrast to linear methods, the air temperatures were involved into the input filter as well.

An alternative filter based on statistical approach called defocusing filter was designed in section 4.1.5 for a comparison of performance with the filter yielded from the evolution. Although the evolved filter showed higher performance when used by the multilayer perceptron (see section 2.1.1) for which it was designed, when employed by the other models the results were almost comparable to the defocusing filter.

Three neural nonlinear models were constructed to predict the runoffs. The prediction power of all three models was better compared to linear regression. They reduced the validation quadratic error approximately of 20% – 30%. The best but most noisy results were obtained by the evolved multilayer perceptron. The results of this model trained by the back-propagation algorithm were not so good but more stable. The hybrid architecture combining the sigmoidal and the RBF neurons placed in between them in both aspects.

In the chapter 5 two neural ensemble architectures inspired by the bootstrap aggregation method were examined. First of them combines standard bagging with the overtraining prevention. The second uses the self-organizing input space quantization with the Kohonen neural network.

The bagging ensemble brings desired variance reduction of the results. On the other hand the performance of the model is located to the lower border of the interval occupied by the single network models. The bagging is based on assumption that model errors distribution is unbiased and thus averaging of individual network outputs will reduce both, the mean error and its variance. According to the results it seems to be that the error bias for particular pattern is fixed for majority of networks in the ensemble.

The self-organizing ensemble performed better. The partitioning of the input space using the Kohonen grid was however concluded as not being in accord with natural phase transition lines of the problem, since the optimal network for the particular input was chosen from the ensemble by the Kohonen network for 30% of input patterns at most. The concurrent method of output combination using the supervised gating network had better results but it did not excel the single network models either.

Finally the rigorous statistical comparison of all models generalization ability was made on the extended training set (see chapter 6). The results of this test differ from the validation results presented before. The architectures that use RBF neurons and Kohonen quantization failed to predict the runoff better than the simple estimation with last value.

The models based on multilayer perceptron architecture proved however, to be working independently on changing conditions. The superiority of neuroevolution suggested in the chapter 4 was confirmed.

The final performance of the constructed predictors i. e. the reduction of the mean squared error compared to the last value estimation varies around 15%. With respect to all the pitfalls and difficulties of the processed data, the reliable positive result, even decent, is more than encouraging.

According to my belief, the examined models exploited the potential of the

general computational intelligence approach. The most promising way to continue the work might be in the direction of intensive specialization of a model particles as well as an asymmetrical approach to the input time series. The construction of a method for proper partitioning of the input space with respect to the underlying physics is in my opinion the most promising challenge of a future research on this field.

Bibliography

- [1] R. H. Bisseling,
*Parallel Scientific Computation
A Structured Approach using BSP and MPI,*
Oxford University Press, USA, 2004
- [2] T. Cipra,
Analýza časových řad s aplikacemi v ekonomii,
Praha, 1986
- [3] R. S. Govindaraju,
Artificial Neural Networks in Hydrology I: Preliminary Concepts,
Journal of Hydrologic Engineering, April 2000
- [4] R. S. Govindaraju,
Artificial Neural Networks in Hydrology II: Hydrologic Applications,
Journal of Hydrologic Engineering, April 2000
- [5] A. H. Halff, M. H. Halff, and M. Azmodech,
Predicting runoff from rainfall using neural networks,
Proc. Engrg. Hydrol., ASCE, New York, 1993
- [6] J. H. Holland,
Adaptation in Natural and Artificial Systems,
University of Michigan Press, 1975
- [7] A. W. Jayawardena and D. A. K. Fernando,
*Comparision of multi-layer perceptron and radial basis function network as tools
for flood forecasting,*
Proc. North Am. Water and Envir. Conf. ASCE, New York, 457-458
- [8] T. Kohonen,
The self organizing map,
Proc. IEEE, 78, 1464-1480
- [9] J. A. Leonard, M. A. Kramer, and L. H. Ungar,
*Using radial basis functions to approximate a function
and its error bounds,*
IEEE Trans. Neural Netw., vol.3, no.4, pp.624–627, July 1992

- [10] T. Masters,
Practical Neural Network Recipes in C++,
Morgan Kaufmann Publishers, San Francisco, USA
- [11] J. E. Nash and J. V. Sutcliffe,
*River flow forecasting through conceptual models,
part I – A discussion of principles*,
Journal of Hydrology 1970, 10 (3), 282–290
- [12] D. E. Rumelhart, G. E. Hinton, R. J. Williams,
Learning internal representations by the error propagation,
Parallel distributed processing, vol. 1 – foundations,
MIT Press, 1986
- [13] P. Moerland,
*Mixture models for unsupervised and supervised learning,
chapter 3 – Supervised Mixture Models*,
EPFL, Lausanne 2000
- [14] A. W. Minns and M. J. Hall,
Artificial neural networks as rainfall-runoff models,
Hydrologic Science, 41, 1996
- [15] A. J. C. Sharkey (ed.),
Combining Artificial Neural Nets,
chapter 1 – A. J. C. Sharkey: Multi-Net Systems,
Springer US, 1999
- [16] A. J. C. Sharkey (ed.),
Combining Artificial Neural Nets,
chapter 2 – L. Breiman: Combining Predictors,
Springer US, 1999
- [17] J. Taheri and A. Y. Zomaya,
Handbook of Nature-Inspired and Innovative Computing,
part I – chapter 5,
Springer US, 2006
- [18] E. Shaw,
Hydrology in Practice,
335-336, Chapman & Hall London, 2004
- [19] K. Zvára a J. Štěpán,
Pravděpodobnost a matematická statistika,
Matfyzpress, Praha 2003. ISBN 80-85863-93-6

- [20] A. S. Tokar and M. Markus,
Artificial neural networks and conceptual models in water management of small basins in the central United States,
Proc. 3rd Int. Conf. on FRIEND, International Association of Hydrological Sciences, Wallingford U.K.
- [21] M. Zhu, M. Fujita, N. Hashimoto,
Application of neural networks to runoff prediction,
Stochastic and statistical method in hydrology and environmental engineering – vol. 3, K. W. Hipel et al. (ed.)
Kluwert, Dordrecht, The Netherlands 1994