MASTERARBEIT

Titel der Masterarbeit

Vienna Neural Network Specification Language 2.0

verfasst von

Thomas Kopica MSc BSc

angestrebter akademischer Grad

Diplom-Ingenieur (Dipl.-Ing.)

Wien, 2015

Studienkennzahl lt. Studienblatt:	A066926
Studienrichtung lt. Studienblatt:	Masterstudium Wirtschaftsinformatik
Betreuerin/Betreuer:	Univ.-Prof. Dipl.-Ing. Dr. Erich Schikuta

Abstract

English

This thesis presents the development of the Vienna Neural Network Specification Language (ViNNSL). At the beginning, neural networks are classified and mapped to ontologies. In addition, neural network applications based on publications are presented. The ontologies are used as search tree in order to find neural networks based on several parameters to solve a specific problem. Afterwards, ViNNSL 1.0 is outlined and extended to ViNNSL 2.0. Therefore, already proposed neural network specification languages are analysed. Based on the analysis ViNNSL 1.0 is extended and every schema described in detail. ViNNSL 2.0 provides the possibility to describe neural networks in order to enable IT systems to execute them automatically and present the outcome. For a better understanding use cases providing an application and evaluation example to ViNNSL 2.0 are given. Finally the future use of ViNNSL 2.0 is outlined.

German

Diese Masterarbeit beschreibt die Entwicklung der Vienna Neural Network Specification Language (ViNNSL). Zu Beginn werden neurale Netze klassifiziert und auf Ontologien abgebildet. Darüber hinaus werden Anwendungen für neurale Netze, basierend auf Publikationen, vorgestellt. Die Ontologien dienen als Suchbaum um neurale Netze zu identifizieren, die zum Lösen einer bestimmten Problemstellung geeignet sind. Anschließend wird ViNNSL 1.0 erläutert und auf ViNNSL 2.0 erweitert. Zu diesem Zweck werden existierende Beschreibungssprachen für neurale Netze analysiert. Basierend auf den Ergebnissen wird ViNNSL 1.0 ergänzt und jedes Schema im Detail beschrieben. ViNNSL 2.0 bietet die Möglichkeit neurale Netze in einer Form zu beschreiben, sodass IT-Systeme in der Lage sind, diese automatisiert ausführen und das Ergebnis präsentieren zu lassen. Für ein besseres Verständnis, werden Fallbeispiele zur Anwendung und Bewertung von ViNNSL 2.0 präsentiert. Am Ende wird die zukünftige Verwendung von ViNNSL 2.0 dargestellt.

Acknowledgements

I thank my parents Mr. Mag. Franz Kopica, MSc MBA and Mrs. Michaela Hawla-Kopica, MBA and grandmother Katharina Kopica for supporting my pretensions during my studies and pushing me smoothly forward.

Finally I would like to express the deepest appreciation to my supervisor and lecturer Mr. Univ.-Prof. Dipl.-Ing. Dr. Erich Schikuta for the excellence support and food for thoughts. Furthermore, he helped me to set the right focus for the thesis.

Contents

1. Introduction

The thesis describes a concept for an implementation of a specification language for neural networks. In connection with the specification language a classification ontology for neural networks by their type, problem domain and other categories based on published literature is presented. In addition the state of the art on neural network markup langugaes is used to develop ViNNSL 2.0.

Objectives: The first objective is developing a classification schema for neural networks and mapping of different neural network types to different fields of application based on literature. The following step is to find out how ViNNSL has to be implemented in order to be used within N2Grid and later on in N2Sky and maybe in other neural network cluster too.

Non-objectives: This master thesis will not follow the approach of implementing a running executable environment for ViNNSL. Only the structure of the first executable version of the language will be outlined and described.

1.1. Motivation

MANN (2013) proposed N2Sky, an artificial neural network simulation environment providing basic functions like creating, training and evaluating neural networks. The system is Cloud-based in order to allow for a growing user community. For enabling the communication between components of N2Sky and other resources within the cloud a neural network specification language is required.

In concern of a growing community, and available data as well as networks, a possibility to brows through the available information is necessary. At this point ViNNSL 2.0 comes to action. Beside the possibility to contain all relevant information to set up and work with a neural network, ViNNSL 2.0 shall also support the system by providing information on the neural network paradigm. This means an easy way to provide meta information on neural networks like problem domain, application domain and type is necessary.

1.2. Structure

At the beginning a neural network ontology, containing a neural network classification ontology and a problem domain ontology, will be introduced. The basis for the ontologies

is technical literature on neural networks. Within the ontology introduction different neural network types are described shortly.

The following chapter provides an overview of state of the art neural network markup languages. Sections within this chapter explain the current state of ViNNSL which is based on the published paper from the University of Vienna science group. They also give an overview of N2Grid as well as other languages.

Based on the results of previous chapters ViNNSL 2.0 will be explained. The sections describe influences of other languages, the extensions to ViNNSL 2.0 and a detailed description of the proposed structure provided in XSD-form.

The ViNNSL 2.0 explanation is followed by use cases, which are used as application and for evaluation of the language. Therefore, a special software tool for XML development is used.

In the end a conclusion on the results of this thesis and the future use of ViNNSL 2.0 in the N2Sky system is provided.

1.3. Methods

The ontology development is based on research on technical literature. Whereas the state of the art of neural network specification languages is determined by research on publications. Based on these findings ViNNSL 2.0 schemas are developed and use cases for their application provided. The use cases are evaluated using an XML tool.

2. Neural Networks

The scientific field distinguishes between two kinds of neural networks, natural and artificial networks. In this thesis the term *neural network* describes an artificial neural network. However, the difference between these types will be outlined.

Natural neural networks use computing elements, called neurons, which are located in the brain. A human brain for example consists of approximately 10^{11} neurons. Neurons communicate through a connection network of axons and synapses. (ZURADA 1992) They are self-organising systems and each neuron is a complex arrangement, which deals with incoming signals in many different ways. Although, the production and transport of signals is well-understood, the cooperation to form complex and massively parallel systems capable of incredible information processing is still unclear. Their advantage compared to conventional computer systems is the massive parallelism and redundancy, which they exploit in order to deal with the unreliability of the individual computing units. (ROJAS 1996)

Artificial neural networks are an attempt at modeling the information processing capabilities of nervous systems. (ROJAS 1996) They allow solving complex, mathematically ill-defined problems, nonlinear problems or stochastic problems. Those networks are from a computational and algorithmical point of view very simple and provide a self-organising feature to hold for a wide range of problems. Like brains they use high parallelity. If a *neuron* fails to work it won't affect the whole network. Natural neurons have switching times of milliseconds, which is slow compared to electronic logic gates achieving nanoseconds. (GRAUPE 1997) Therefore, neural networks have great potential to solve complex problems much faster than humans or animals.

2.1. Classification of Neural Networks

Regarding the increasing number of different neural networks an attempt for classification seems more and more reasonable. Therefore, this thesis provides a possibility based on the approach of HAUN (1998). Figure 2.1 provides a short overview on the classification levels developed by Haun. In this approach neural networks are classified into three levels. The levels are based on the connection type, neuron behaviour and learning methods.

- Level 1: Feedback and feedforward networks
- Level 2: Nonlinear and linear networks

- Level 3: Supervised and unsupervised networks

Haun doesn't provide many kinds of neural networks in his classification. Therefore, several types given by ZELL (1994) who doesn't give any classification as well and cellular neural networks (CNN) are added.

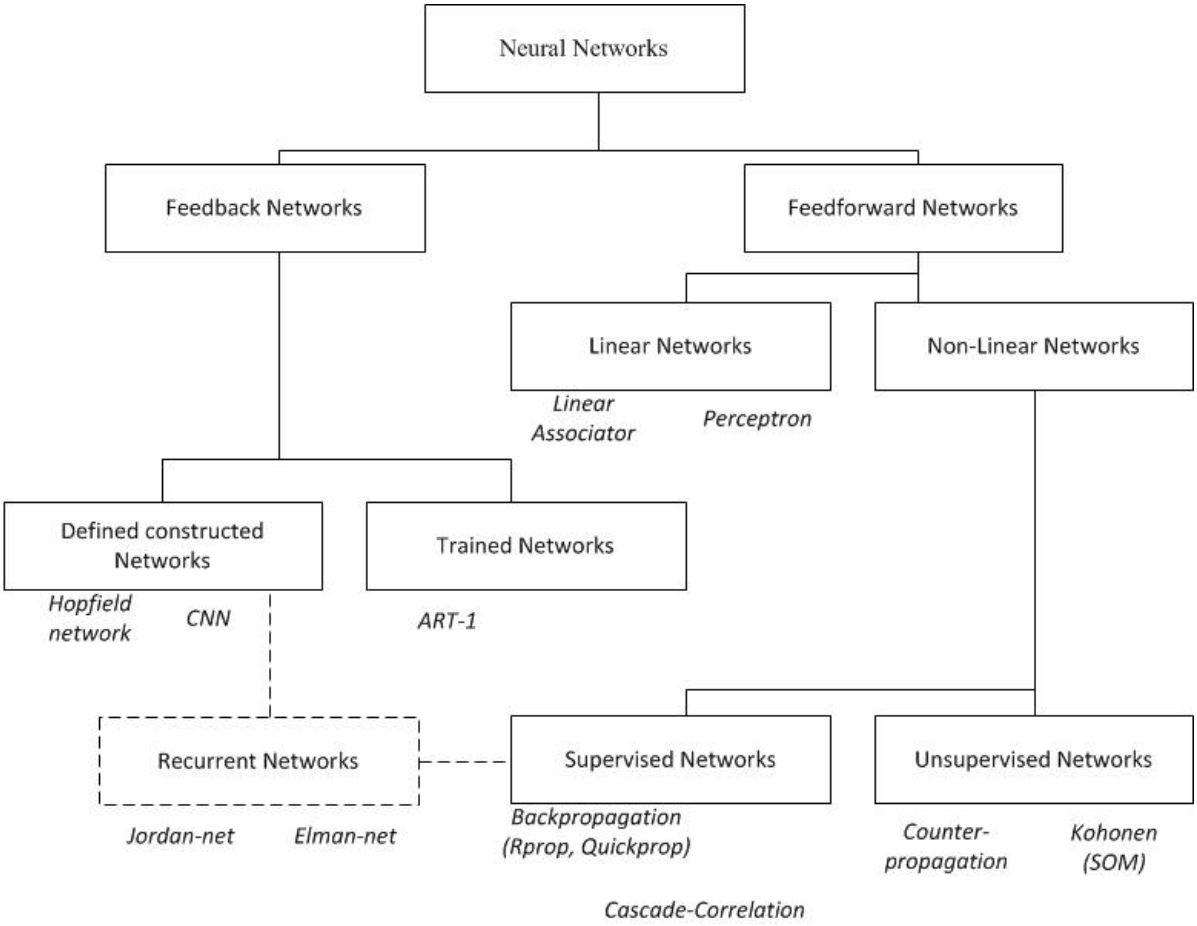

Figure 2.1.: Classification of neural networks modified from HAUN (1998)

The following chapters provide a more detailed explanation of the levels and their appropriate networks.

2.1.1. Feedback Networks

Feedback neural network signals travel in different directions. Therefore, the network's "state" continuously changes. According to Badiru & Cheung input values initialises the network only. Once initialised, the network output will continuously change. Depending on the network parameters a dynamic network could continue to change or stabilise at an equilibrium point. (BADIRU & CHEUNG 2002) These nets are able to deal with incorrect or missing input data. However, feedback neural networks do not always find the same or exact solution. (HAUN 1998)

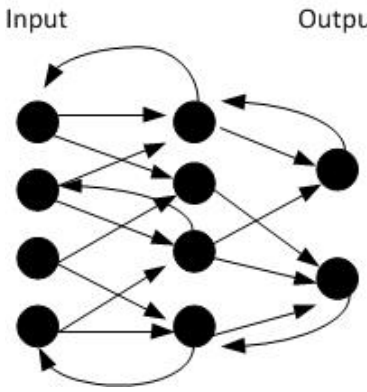

Figure 2.2.: Feedback network modified from HAUN (1998)

2.1.1.1. Defined Constructed Networks

These network's structure is already defined, when the data is presented. An example is the Travelling-Salesman-Problem, where a number of locations has to be visited once and the tour ends at the start point. Every location is represented by one neuron.

2.1.1.1.1. Hopfield Network

An example of a feedback network is the Hopfield network. It was a milestone in the field of neural networks, introduced at the beginning of the 1980s. Hopfield nets are asynchronous, which means each unit computes its excitation at random times and changes its state independently to 1 or -1. They keep their individual states till they are selected for an update. The selection is made randomly. A Hopfield net has n neurons, which are connected with all other neurons in the net except themselves. These nets are symmetric, because the weight of the connection from neuron i to neuron j has the same value as the connection from j to i.

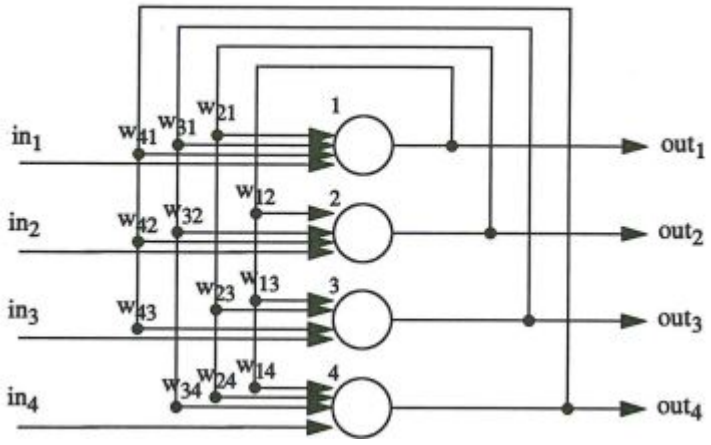

Figure 2.3.: Hopfield network with four neurons (ZELL 1994)

2.1.1.1.2. Cellular Neural Networks (CNN)

Cellular neural networks were the first time proposed by Chua & Lin in 1988. According to the researchers these networks can be viewed as a particular case of continuous Hopfield networks. (SLAVOVA & MLADENOV 2004) The main difference compared to the Hopfield network is, that neurons are only connected with neurons in their neighbourhood. (CHUA & Lin 1988) While neurons in Hopfield networks, as stated in 2.1.1.1.1, are connected with all other neurons.

CNN enables parallel processing in the true sense. It has advantageous characteristics compared to other neural networks. This net can be extended easily without readjusting all weights. Although its cellular structure does still keep the complex dynamic behaviour as seen with other neural networks. (SLAVOVA & MLADENOV 2004)

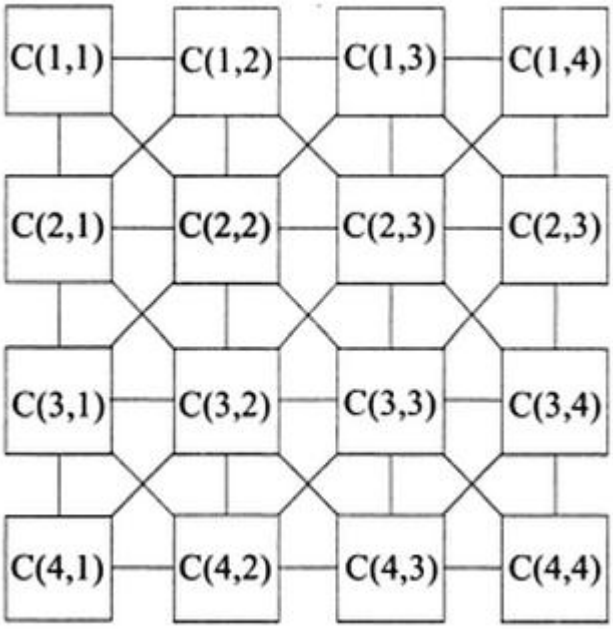

Figure 2.4.: 4x4 CNN (SLAVOVA & MLADENOV 2004)

2.1.1.2. Trained Networks

This group contains feedback networks which can be trained. These networks provide supervised and unsupervised training.

2.1.1.2.1. Adaptive Resonance Theory (ART) 1

ART usually describes a family of neural networks. They were developed in order to solve the stability-plasticity-dilemma. The issue was to determine how neural networks

can learn new associations without forgetting old ones. Plasticity is the term describing the modifiability of neural networks. Stability describes the ability to remember learned knowledge. (ZELL 1994)

In this thesis ART-1 is explained in more detail only in fact, that all members of this family are able to deal with the stability-plasticity-dilemma. However, well known representatives are given below.

- ART-1: original version, only able to deal with binary inputs
- ART-2: enhancement of ART-1 for continuous inputs
- ART-2A: simplification of ART-2 for faster convergence
- ART-3: enhancement of ART-2 to model chronological or chemical processes
- ARTMAP: combination of two ART-nets (1 or 2) having supervised learning
- FUZZY ART: combination of fuzzy logic and ART

(ZELL 1994)

ART-1 consists of a comparison, a recognition layer and a reset component. Each vector has an intensifying neuron called gain. They are acting as a switch for network synchronisation. The reset component has the value 1, when the tolerance for the difference of the input pattern and the result of the comparison layer excels a predefined level. In this case the neurons of the recognition layer don't fire. (ZELL 1994)

Every neuron in the recognition layer stores one pattern. Therefore, the comparison and the recognition layers are connected with two weighted matrices. One matrix is in the direction from the comparison to the recognition layer and vice versa. But the matrixes are not directly connected with each other. (ZELL 1994)

The network is initialised using a null vector. The weights of the bottom-up-matrix get the same low value and the weights of the top-down-matrix are set to 1. Afterwards, the input vector is presented to the network. Then the most similar neuron in the recognition layer fires. The result is that each neuron in the comparison layer gets a state (0 or 1). If the difference between the input vector and the state of comparison layer is too high, a reset will be set. The reset deactivates the previously firing neuron in the recognition layer. (ZELL 1994)

This process continues till a stored vector is found which is similar enough to the input vector or none of them is similar enough. If a similar vector was found, the network modifies the weights of the matrixes. In the other case an unused neuron of the recognition layer is used to store the input vector. (ZELL 1994)

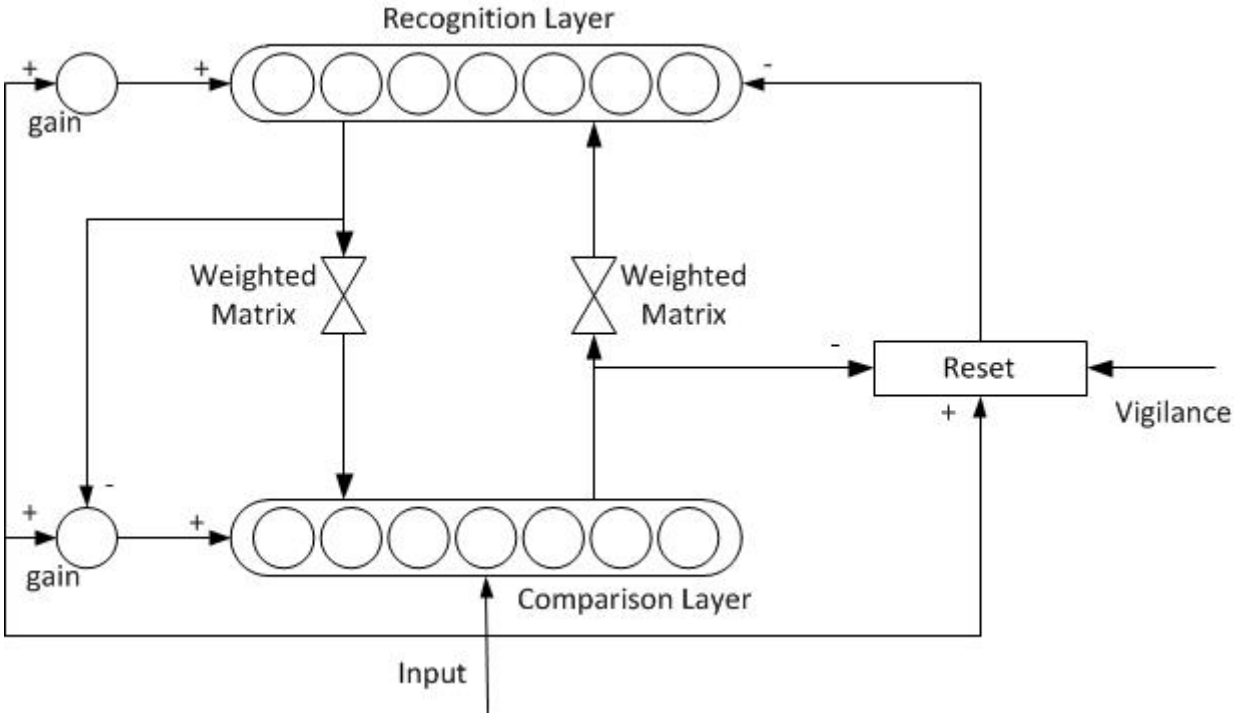

Figure 2.5.: ART-1 Architecture modified from ZELL (1994)

2.1.2. Feedforward Networks

Connections in feedforward networks are going in one direction only. Neurons will be connected between different layers. However, layers can be skipped too. (HAUN 1998) The output can be calculated directly from the input without knowing initial states. Unlike feedback networks it does not contain loops and time delays. (MEDSKER & JAIN 2000) That means an output of a neuron can't be an input of a neuron from a previous layer. Feedforward networks can be divided into linear and nonlinear networks (see 2.1), which will be explained in the following sections.

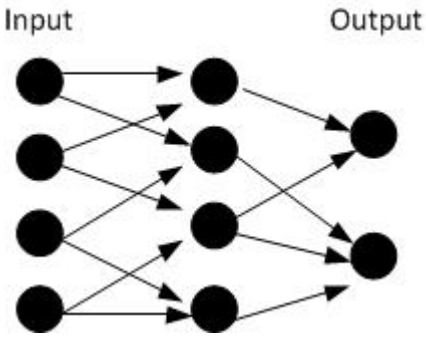

Figure 2.6.: Feedforward network modified from HAUN (1998)

2.1.2.1. Nonlinear Networks

The output value of neurons will be calculated with a nonlinear function based on the input values. The easiest implementation of these neurons has an activation value of 1 if the neuron's weighted sum is greater than a threshold otherwise it's 0. (HAUN 1998)

These networks can be split into supervised and unsupervised networks. The distinction applies to the learning behaviour. A more detailed explanation will be given in the following sections based on ROJAS (1996).

2.1.2.1.1. Supervised Networks

Input vectors are collected and presented to the network. The network computes the output, and the deviation from the expected results are measured. Afterwards, the weights, based on the magnitude of error, and the learning algorithm are corrected. The network uses reinforcement or error correction to improve its weights. Reinforcement learning is used when the result of the network is desired or not. In learning with error correction, the magnitude of the error, together with the input value, is used to update the weights. This type is also called *learning with a teacher*. (ROJAS 1996)

The process will be repeated till the error rate reaches an acceptable level. Usually nets, which use this learning type, are provided with a training set to correct their weights and a validation set to evaluate their settings.

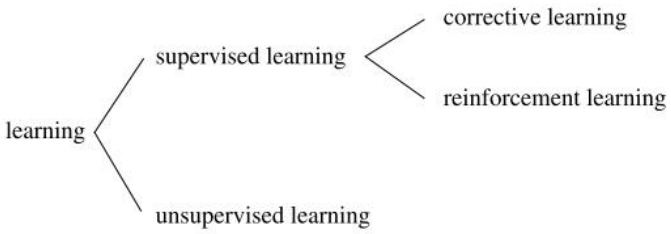

Figure 2.7.: Classes of learning algorithms (ROJAS 1996)

2.1.2.1.1.1. Backpropagation (Rprop / Quickprop)

Unlike defined constructed networks as discussed in 2.1.1.1, backpropagation is not a network design but the description of the learning algorithm, which repeatedly passes the training data set through a network to determine the corresponding weights for each input variable in the output and hidden layers.

The network calculates the predicted value and compares it with the actual value from the training set. The derivatives are evaluated based on the error function and the network weights. Therefore, the backpropagation algorithm goes backwards through the

net and adjusts the weights. This process is repeated till all error vectors are zero, i.e. a perfect fit of the data or convergence criterion values are met. (MATIGNON 2005) Rprop and Quickprop are improvements of the backpropagation algorithm.

Quickprop: This approach is used to determine the minimum of a feedforward net's error function. It assumes that the error function is locally powered by two. Two assumptions have to be given in order to work with Quickprop:

1. The error function can be locally approximated with a parabola.
2. The change of a weight wij is independent from changes of other weights.

If the assumptions are given Quickprop is faster by factor five to ten compared to backpropagation. (ZELL, 1994, pp. 120-124)

Rprop: It combines Manhattan-Trainings, SuperSAB and Quickprop. Like in Manhattan-Training weights are changed according to the arithmetic sign of the slope of the error function. In addition only the error function's slope of the current and previous time point like in SuperSAB and Quickprop are used. Moreover, every weight has its own parameter for value changes. Compared to other algorithms Rprop has the easiest propagation rule. (ZELL 1994)

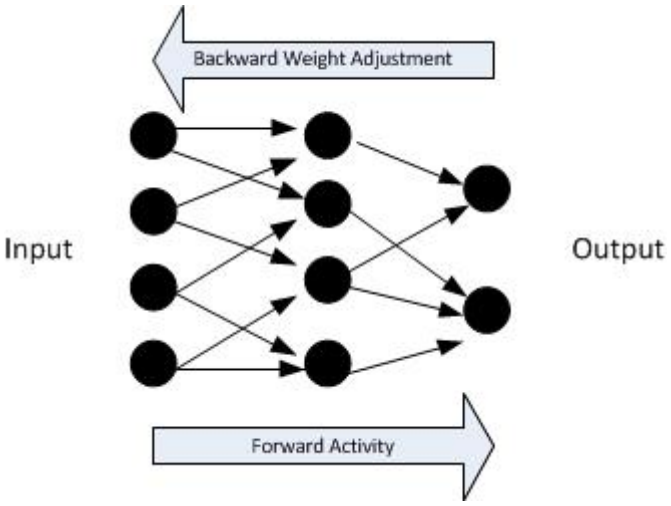

Figure 2.8.: Backpropagation network modified from HAUN (1998)

2.1.2.1.1.2. Cascade-Correlation Networks

The Cascade-Correlation learning architecture defines the weights between neurons and the topology of a network. It starts with the smallest possible network and adds hidden layer during training. However, each hidden layer contains only one neuron. If a new hidden neuron is added the input weights are frozen and only the output weights will

be changed. (ZELL 1994)

An advantage of Cascade-Correlation is the possibility to solve specific problems every time. For example the solution to a problem can be split into sub problems A and B. Once a sub problem is solved the net is always able to solve it and future neurons just focus on solving the other problem. Usually hidden neurons take a long time till they decide on which sub problem they focus. A solution to this loosed time is changing only a few weights. Cascade-Correlation uses an extreme interpretation of this strategy. (ZELL 1994)

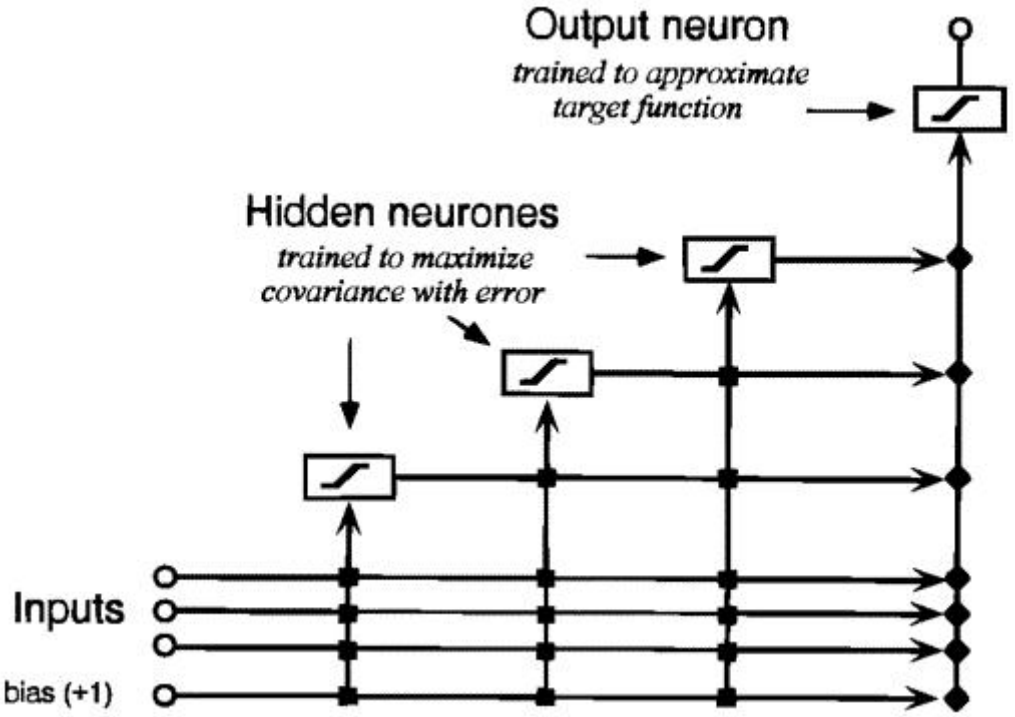

Figure 2.9.: Cascade-Correlation Network (KOVALISHYN et al. 1998)

2.1.2.1.2. Unsupervised Networks

The numerical output for a given input which shall be produced by the network is unknown. Therefore, the network must organise itself in order to produce appropriate results. (ROJAS 1996) The goal for unsupervised networks is describing groups of data that is similar to principal components, cluster analysis or Kohonen maps. (MATIGNON 2005)

2.1.2.1.2.1. Kohonen Network (Self-Organising Maps - SOM)

Kohonen networks, also called self-organising maps, are self-organising networks. The

output is not predefined and the mapping of weight vectors to clusters is an automatic process. At each step one input vector at the same time is presented. Together, they constitute the "environment" of the network. Each new input produces an adaptation of the parameters. Assuming these modifications are correctly controlled, the network can build a kind of internal representation of the environment. (ROJAS 1996)

HAUN (1998) and ROJAS (1996) independently state that Kohonen networks use a one dimensional chain of units. Each unit reacts on its neighbours. The goal is that neighbour units learn to react to closely related signals. At the beginning neurons react randomly to inputs but during the training they will start to group together. (HAUN 1998)

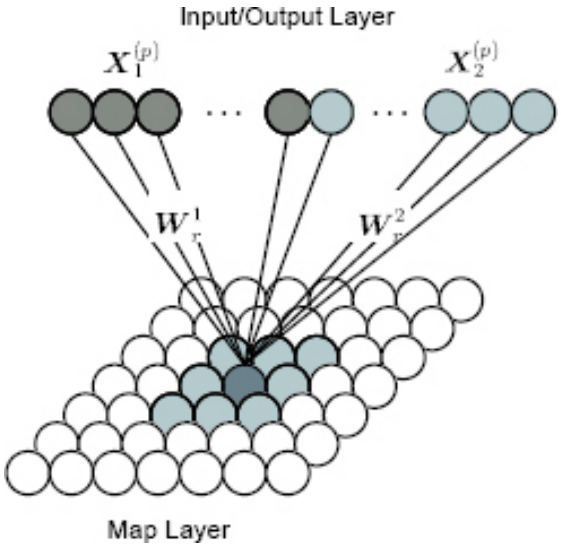

Figure 2.10.: Kohonen network (NOGUCHI & YOUKO 2010)

2.1.2.1.2.2. Counterpropagation Network

First introduced by Hecht-Nielsen, the Counterpropagation network is an extension to the Kohonen network. It contains a hidden layer of Kohonen neurons. They are for example connected with a linear associator (ROJAS 1996) or a Grossberg net (GRAUPE 1997).

The Kohonen layer is trained to converge to the average inputs. This layer is a preclassifier to account for imperfect inputs using unsupervised training whereas the Grossberg layer uses supervised training in order to converge to the desired output. (GRAUPE 1997)

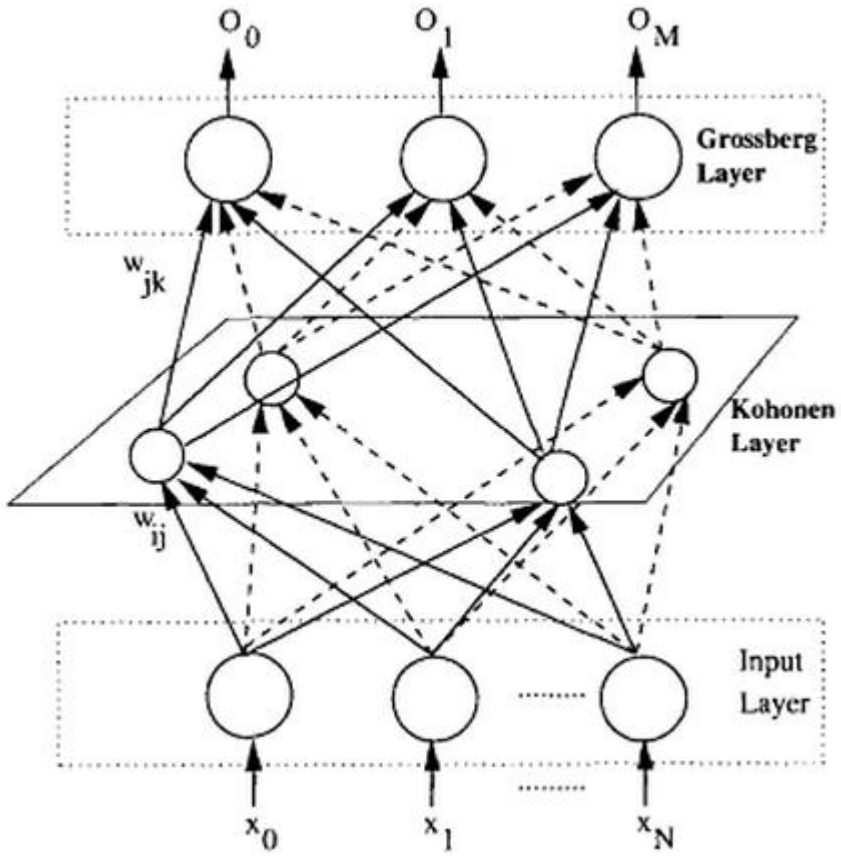

Figure 2.11.: Counterpropagation network (ACHARYA & RAY 2005)

2.1.2.2. Linear Networks

Linear networks use linear activation functions for their nodes. This means the output, activation value, of any node is linearly proportional to the sum of the inputs to the node. For example a neural network consists of an input, a hidden and an output layer having a linear activation function only. The hidden nodes output is a linear combination of the input values. Thus apply to the output nodes too. This creates linear combinations of the original inputs. The network can do no more than generate outputs that were linear functions of the input. (RZEMPOLUCK 1998)

2.1.2.2.1. Perceptron

ZELL (1994) distinguishes between three types of perceptrons: Single Layer, Double Layer and Triple Layer perceptron.

Perceptron describes a family of related neural networks. They are used for visual pattern identification. The structure of a general perceptron contains an input layer with fixed weighted connections to the working layer. The connections from the working

layer to the output layer are variable and can be trained. (ZELL 1994)

The number of layers in a perceptron is identical with the number of stages having variable weights. The working layers are named "Level 0" to "Level n" whereas "Level n" represents the output layer. According to Zell do more working layer increase the mightiness of perceptrons. However, after a third layer perceptrons don't gain additional abilities. (ZELL 1994)

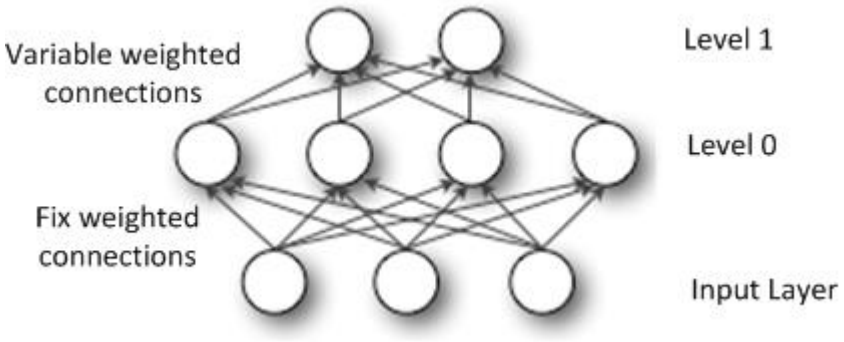

Figure 2.12.: Perceptron schema modified from LISA (2010) according to ZELL (1994)

2.1.2.2.2. Linear Associator

A linear associator is a computing unit which adds its weighted inputs. It shall reproduce the output of the input vectors of a training set. (ROJAS 1996) Linear associators are based on the Hebbian learning rule which allows initialising the network directly without explicit training. This network can be used as a method to store and recover patterns from memory. The memory can be retrieved in different forms. (LUGER 2005)

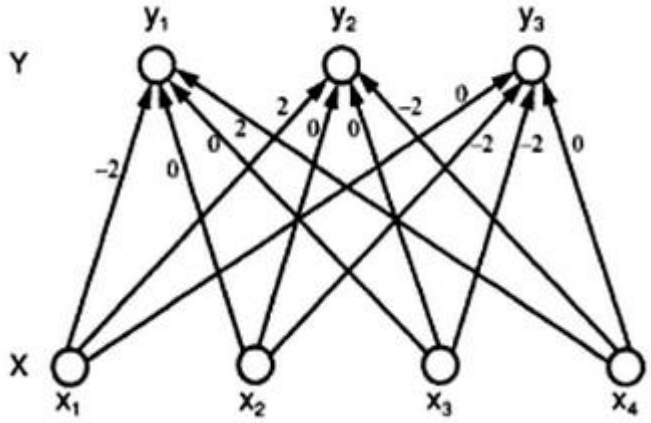

Figure 2.13.: Linear associator network (LUGER 2005)

2.1.3. Partial Recurrent Networks

Partial recurrent networks cannot be clearly grouped to feedforward or feedback networks. They are usually based on feedforward networks having characteristics of feedback networks. Their purpose is considering the chronological order of input data. These nets have context cells which implement a memory mechanism. Context cells receive input from hidden or output cells and transfer them as input to the net. Partial recurrent networks can be trained with slightly modified training algorithms of feedforward nets. (ZELL 1994)

2.1.3.1. Jordan-Net

Jordan-nets use a simple feedforward net architecture enhanced by context cells. The input cells and context cells deliver their values to the hidden cells. The result of the output cells are used as net results and input for the context cells. In addition context cells have a direct feedback from themselves. These weights are static values and cannot be changed by training. The feedback weight is a value between [0,1]. If the value tends to 0 older inputs have a decreasing influence. This means the net is forgetting but is flexible towards changes. If the value is close to 1 older inputs have a higher influence on the result. (ZELL 1994)

The number of output cells has to match the number of context cells. All trainable connections head to the output cells. The net's result is always influenced by external input and the state stored in the context cells. Disadvantage of this model is that many problems require a small value of the feedback weight and at the same time a high value is wanted in order to consider older inputs. In addition the state of the hidden layer cannot be stored. (ZELL 1994)

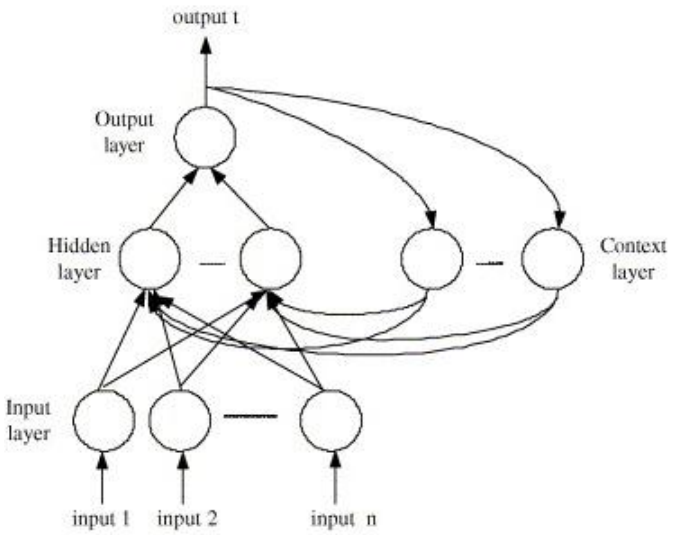

Figure 2.14.: Jordan-Net (JORDAN 1990)

2.1.3.2. Elman-Net

Elman-nets are a modification of Jordan-nets. The feedback loop goes from the hidden cells to the context cells and the direct feedback loop of context cells to themselves is skipped. In this case the number of context cells has to match the number of hidden cells. The connection weight to context cells is 1.0. Context cells store the values of the hidden cells from the previous sequence. Therefore, a chronological connection to previous sequences can be reached. Simple Elman-nets contain only one hidden layer, but for complex problems, nets with several hidden layer produce better results. In this case hierarchical Elman-nets are useful. (ZELL 1994)

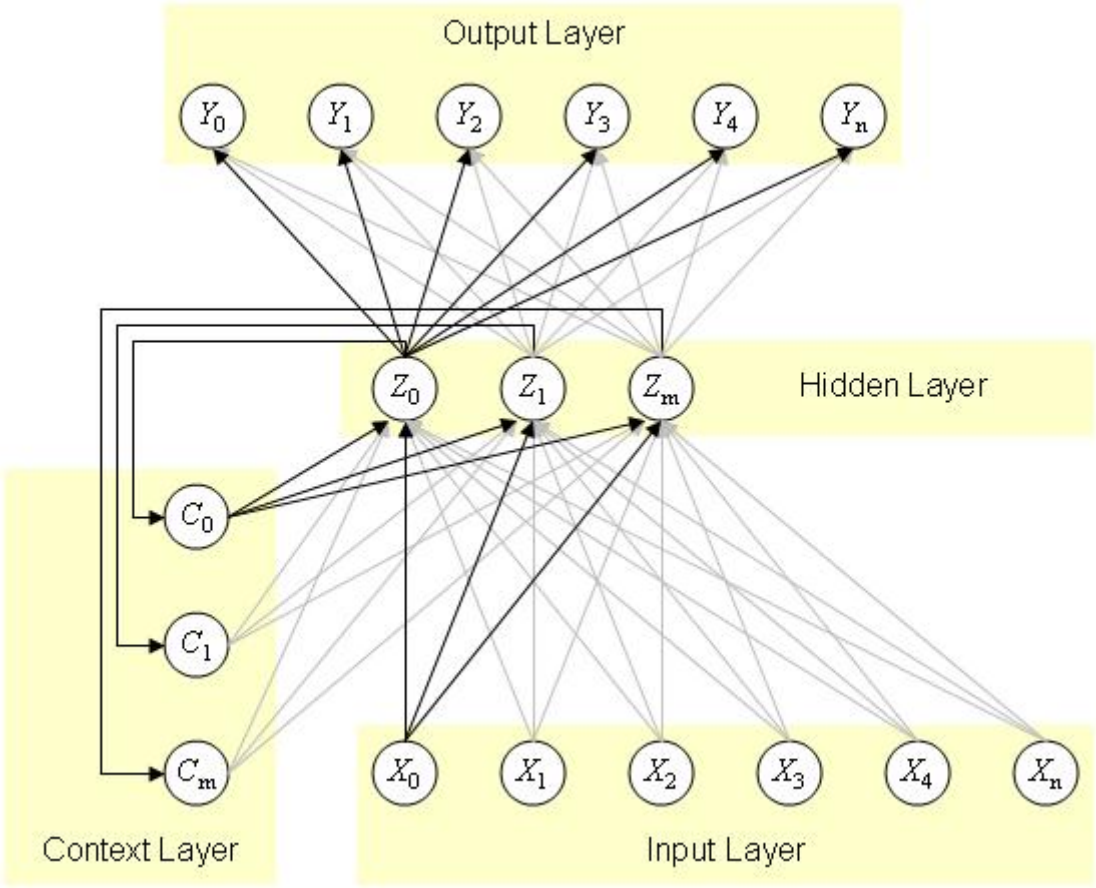

Figure 2.15.: Elman-Net (McCULLOCK 2012)

2.1.3.2.1. Hierarchical Elman-Net

Hierarchical Elman-nets allow the usage of several hidden layers. Each hidden layer and the output layer have its own layer of context cells. Additionally, every context cell has a feedback loop to itself. The weight of these loops can be defined on layer basis. Therefore, the memory behaviour for each hidden layer can be influenced. (ZELL 1994)

2.2. Neural Networks Application Domains

This section shall give an overview of the possibilities of specific neural network types to solve specific problems. Therefore, applications for neural networks will be generalised and described. In the second step, for each neural network stated in 2.1, applications based on literature will be given. The last step is the presentation of networks and their applications in table form.

2.2.1. Problem Domains of Neural Networks

ZURADA (1992) proposed examples of neural network paradigms. These are real world problems which are solved using neural networks. The problems are explained in detail within this section. All research papers presented in section 2.2.3 can be assigned to one of the following domains. Figure 2.16 shows the neural network paradigms. In order to keep the figure readable the dashed lines give a single example of an application and the corresponding network type. For example, a classifier problem occurs in Retail, Operations and other application domains. In the next step e.g. a Retail classification problem can be implemented using Backpropagation, Hopfield and other network types.

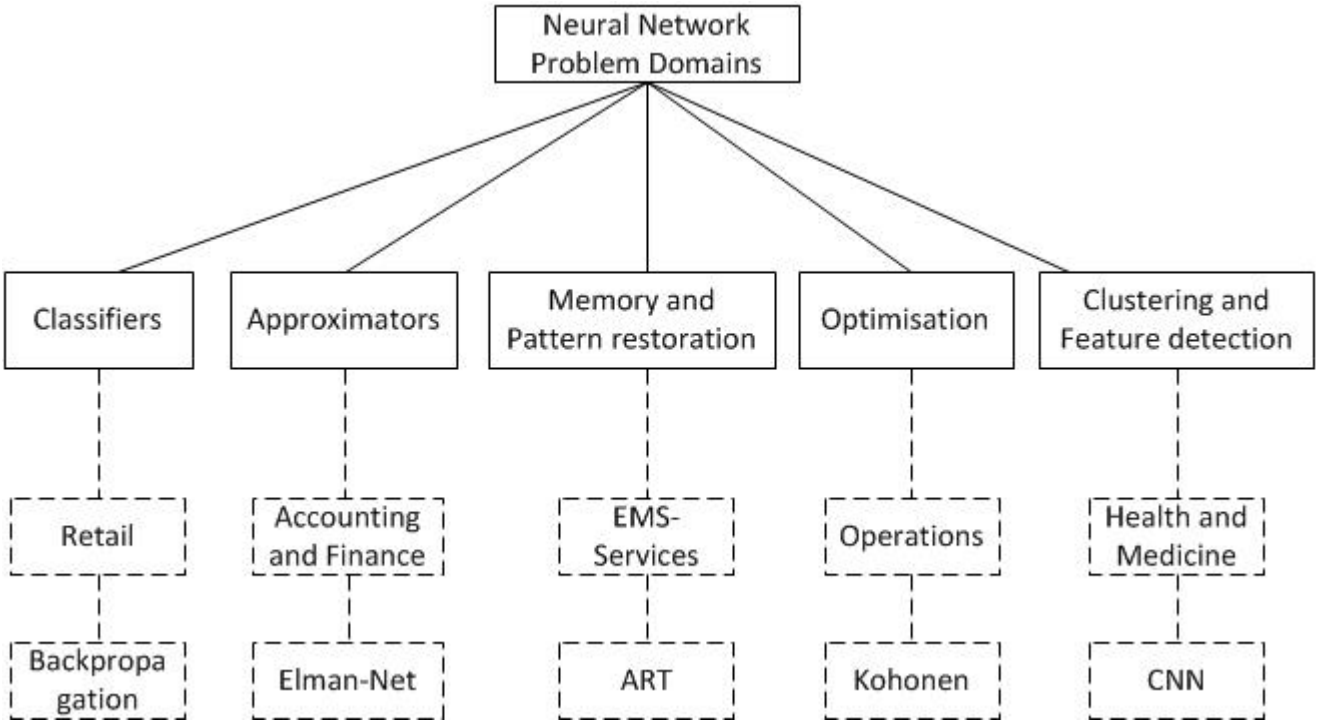

Figure 2.16.: Neural Network Problem Domains

2.2.1.1. Classifiers

Classifiers respond instantaneously to input. They classify input based on a decision function. The neural network design's importance increases with the complexity of the group requirements. An application example is the evaluation of electroencephalogram signals of patients. It is used to detect imminent epileptic seizures. (ZURADA 1992)

2.2.1.2. Approximators

Approximators are based on classifiers. Examples are autonomous driving systems. Based on road images and range finders the neural network trains and is able to define steering actions in order to keep a car in the middle of a road. The same principle is used to approximate to mathematic functions. (ZURADA 1992)

2.2.1.3. Simple Memory and Pattern Restoration

Memories respond, in time, to presented patterns. These networks process a gradual reconstruction of stored patterns in order to restore input patterns. For every stored pattern the memory provides a stable output. If a presented pattern does not agree with the stored stable output the network responds with the corrected entry. The network shall then terminate at one of its originally stored patterns. (ZURADA 1992)

2.2.1.4. Optimising Networks

Optimisation is one of the most important objectives of engineering. The goal is minimising certain cost functions, which are usually defined by the user. A number of optimisation problems can be translated directly into the minimisation of a neural network's energy function. When the translation is accomplished the optimisation task is presented for the actual solution. An example is the job-shop scheduling problem. It determines in which allocation jobs need to be done in order to minimise costs, time, etc. (ZURADA 1992)

2.2.1.5. Clustering and Feature detection

These networks are tuned to certain similarity aspects which are of interest in data. An example is grouping measurement results together to suppress any systematic errors that may have occurred. Another possibility is detecting frequently appearing signals as inputs. These may indicate the true input signals among noise, which occurs randomly. Feature detection is normally related to dimension reduction in data, also from fairly complex structures. For example mapping speech features. (ZURADA 1992)

2.2.2. Neural Network Applications

Based on the publications of SMITH & GUPTA (2000) and PALIWAL & KUMAR (2009) neural networks will be used in the following fields of application:

- Accounting and finance
- Health and medicine
- Marketing
- Retail
- Insurance
- Telecommunications
- Operations management
- Emergency and Social Services

Possibilities of neural network applications are constantly increasing and therefore, an overview is provided only.

2.2.2.1. Accounting and Finance

One of the main areas of banking and finance that heavily rely on neural networks is trading and financial forecasting. Neural networks have been applied successfully to problems like derivative securities pricing and hedging, futures price forecasting, exchange rate forecasting, stock performance and selection prediction. For many years banks have used credit scoring techniques based on statistical techniques to determine which loan applicants they should lend money to. However, neural networks became the underlying technique driving the decision making. Hecht-Nielson and Co. have developed a credit scoring system, which increased profitability by 27% by learning to correctly identify good and poor credit risks. (SMITH & GUPTA 2000)

Neural networks have also been successful in learning to predict corporate bankruptcy and have been used to model the relationships between corporate strategy, short-run financial health, and the performance of a company. This appears to be a promising new area of application. Financial fraud detection is another important area of neural networks in business. For example Visa International has an operational fraud detection system which is based upon a neural network. The network has been trained to detect fraudulent activity by comparing legitimate card use with known cases of fraud and saved Visa International approximately US$40 million within its first six months of operation. Neural networks have also been used in the validation of bank signatures, identifying forgeries significantly better than human experts. (SMITH & GUPTA 2000)

2.2.2.2. Health and Medicine

Neural networks are used instead of classical statistical approaches in medical informatics. They aid in the detection and classification of coronary artery diseases, breast cancer and many other. Neural networks provide better classification results, not only in the

training sample, but also in the test samples and it is demonstrated, that for medical diagnosis problems, when the data is often highly unbalanced, neural networks can be a promising classification method for practical use. (PALIWAL & KUMAR 2009)

2.2.2.3. Marketing

The application of neural networks in marketing is relatively new, but is becoming popular because of their ability of capturing nonlinear relationship between variables. Numerous applications of neural network models are available. Examples are Market Segmentation, Market Response Prediction, New Product Launch, Sales Forecasting, Consumer Choice Prediction, etc. (PALIWAL & KUMAR 2009)

Market modelling is an extremely important issue in marketing. At the aggregate level, market share models are commonly used in marketing for a number of different purposes. These include the estimation of price and advertising elasticity as well as more generally, predicting the effects of changes in marketing variables. (PALIWAL & KUMAR 2009)

2.2.2.4. Retail

Companies often need to forecast sales to make decisions about inventory, staffing levels, and pricing. Neural networks have had great success at sales forecasting, due to their ability to simultaneously consider multiple variables such as market demand for a product, consumers' disposable income, the size of the population, the price of the product, and the price of complementary products. (SMITH & GUPTA 2000)

The second major area where retail businesses can benefit from neural networks is the area of market basket analysis. Hidden amongst the daily transaction data of customers is information relating to which products are often purchased together, or the expected time delay between the sales of two products. Retailers can use this information to make decisions, for example define the store layout. If market basket analysis reveals a strong association between products A and B then they can entice consumers to buy product B by placing it near product A on the shelves. (SMITH & GUPTA 2000)

In case between two products exists a relationship over time, for example within six months after buying a printer the customer returns to buy a new cartridge, retailers can use this information to contact the customer, decreasing the chance that the customer will purchase the product from a competitor. Understanding competitive market structures between different brands has also been attempted with neural network techniques. (SMITH & GUPTA 2000)

2.2.2.5. Insurance

Policy holders can be classified into groups based on their behaviours, which can help to determine effective premium pricing. Prediction of claim frequency and claim cost can also help to set premiums, as well as find an acceptable mix or portfolio of policy

holders' characteristics. The insurance industry, like the banking and finance sectors, is constantly in need to detect fraud, and neural networks can be trained to learn detecting fraudulent claims or unusual circumstances. (SMITH & GUPTA 2000)

The final area where neural networks can be of benefit is in customer retention. Insurance is a competitive industry, and when a policy holder leaves, information why they have left can be determined from their history. Offering incentives to certain customers like reducing their premiums or providing no-claims bonuses can help to keep them. (SMITH & GUPTA 2000)

2.2.2.6. Telecommunication

Like other competitive retail industries, the telecommunications industry is concerned with customers joining a competitor and win-back. Therefore, series of neural networks are used to analyse customer and call data, predict if, when and why a customer is likely to leave. As well as predict the effects of forthcoming promotional strategies and interrogate data to find the most profitable customers. (SMITH & GUPTA 2000)

Telecommunication companies are also concerned with product sales, since the more reliant customers become on certain products the more likely they stay. Market basket analysis is significant here, since if a customer has bought one product from a common market basket, then enticement to purchase the others can help to reduce the likelihood that they will churn, and increases profitability through sales. (SMITH & GUPTA 2000)

Beside business applications, engineering applications of neural networks are interesting to the operations researcher because it involves optimisation. This includes the use of neural networks to assign channels to telephone calls for optimal network design, for efficient routing and control of traffic. (SMITH & GUPTA 2000)

2.2.2.7. Operations management

Neural networks have been used successfully in many areas of operations management. For example scheduling of machinery, assembly lines and cellular manufacturing as well as other scheduling problems like timetabling, project scheduling and multiprocessor task scheduling. All of these approaches are based upon the Hopfield network which is designed to solve complex optimisation problems. The use of neural networks in various operation planning and control activities cover a broad spectrum of application from demand forecasting to shop floor scheduling and control. (SMITH & GUPTA 2000)

Neural networks have also been used in conjunction with other techniques, for example simulation modelling to learn better manufacturing system design, traditional statistical control techniques to enhance their performance. They can also be used as a diagnostic tool, to detect faults in electrical equipment and satellite communication networks. (SMITH & GUPTA 2000)

In addition the term operations management within this thesis does not distinguish between industrial, agricultural or power production. Furthermore, it also contains all organisational tasks related with operation.

2.2.2.8. Emergency and Social Services

In addition neural networks can be used by emergency services in order to maintain public security. In this thesis emergency services include for example police, ambulance, fire brigade, public infrastructure like gas and power supplies, border control and many others. Additionally this field of application does also contain operations which support human beings in their daily life.

2.2.3. Application of specific Neural Networks

This section provides specific examples for possible applications for the described neural networks in section 2.1 A general overview of the fields of application is given in section 2.2.2 Thus a short view which type of network is applicable to which field of application is given in figure 2.17. The main purpose is giving examples for applications, most of them published within the past five years. However, mentioning all of them would go beyond the scope of this thesis.

Network Type	Acc./ Fin.[1]	Health/ Med.[2]	Marketing	Retail	Insur.[3]	Telecom.[4]	Operations	EMS-Services[5]
Hopfield		•					•	•
CNN		•					•	•
ART			•			•	•	•
Backpropagation	•	•	•	•	•	•	•	•
Cascade-Correlation	•						•	•
Kohonen	•	•	•	•	•	•	•	•
Counterpropagation	•	•			•		•	
Perceptron		•	•	•			•	•
Linear Associator		•						•
Jordan-Net					•		•	•
Elman-Net	•	•		•		•		•

Figure 2.17.: Fields of application for neural networks

[1] Accounting/Finance
[2] Health/Medicine
[3] Insurance
[4] Telecommunication
[5] Emergency and Social services

2.2.3.1. Applications for Hopfield Networks

The ELD (economic load dispatch) problem is one of the important optimisation problems in a power system. PARK et al. (1993) provide an example how a Hopfield network can be used to calculate the costs per generator. Their approach considers fuel, number of generators and others and is compared with numerical methods.

MAETSCHKE & RAGAN (2014) used a Hopfield network in which attractors characterise cell states and used the model to identify cancer subtypes in gene-expression data. Its advantages are unification of clustering, feature selection, network inference and it can be used as modelling framework for epigenetic landscapes.

SAMMOUDA et al. (2014) developed a Hopfield network for pixel clustering of agricultural satellite images. Their network clusters the image into non-overlapping, homogenous regions. In their case study they use the functionality to identify forage areas for bees.

WANG (2013) used a discrete Hopfield network to evaluate water quality. The research paper classifies water into the groups Oligotrophic, Nutrition, Eutrophication. Input values are for example Chlorophyll, Phosphorus and Nitrogen. The comparison with a backpropagation network showed that the Hopfield network is approximately 33% faster.

Concerning the different fields of application the published scientific papers show that Hopfield networks are applicable to be used in Operations, Emergency and Social Services, and Health and Medicine.

2.2.3.2. Applications for Cellular Neural Networks (CNN)

SUBUDHI et al. (2014) proposed an application for detecting moving objects from videos captured by a static camera. Their approach also includes a Gibbs-Markov random field, which is used to create a difference image from the image taken by the camera. The Hopfield network used in this paper is a cellular neural network. The CNN is used to detect changed and unchanged pixels in different frames. The result is, that this application provides images containing moving objects only.

HADAD & PIROOZMAND (2007) developed a CNN which solves the nuclear reactor dynamic equations. They used their model to simulate space-time response of different reactivity excursions in a nuclear reactor. As result, their CNN can be used as assistant for reactor operations and reactor training simulations.

SAHIN et al. (2011) used a CNN to predict missing air pollution data. They try to predict the daily mean of particulate matter and sulphur dioxide (SO_2). The paper shows that using a CNN provides a higher accuracy concerning predictions than multivariate linear regressions. They used datasets having 20% of missing data for their

predictions. In addition, predictions are more reliable in winter than in summer.

SHITONG et al. (2007) developed a prototype of an advanced fuzzy CNN in order to separate the liver out of computer tomography (CT) liver images. Their network is based on the fuzzy CNN proposed by Wang S. and Wang M. to detect white blood cells. The network's task is reducing all unnecessary information from the CT image in order to visualise liver diseases.

NAMBA & ZHANG (2006) introduced a CNN used for pattern recognition based on images showing braille letters. Furthermore, the images are taken by camera phones. A data sample consisting of images with different quality and a total number of 50 was used for the network validation. In addition, the CNN was compared with a multi layered perceptron (MLP). The result shows that the CNN is able to identify approximately 90% of all images correctly compared to 62% by the MLP.

CNNs can be used in different fields of application. Their ability for pattern recognition is often taken for image processing. This task is requested by Health and Medicine, and Emergency and Social Services. The proposed applications for missing air pollution data and braille image pattern recognition are examples from the Emergency and Social Services field of application.

2.2.3.3. Applications for Adaptive Resonance Theory (ART)

The proposed applications for ART-networks are not limited to ART-1, which was described in section 2.1.1.2.1, but also to further developed networks.

GIRI & MOULICK (2014) presented an approach for Group Technology (GT) improvement using an ART-1 network. The network determines the cells of GT. A cell contains machines which produce similar parts. Their proposal reduces the waste of manpower and idle times of machines by reducing the distances for components between different machines. The network's task is grouping the machines and components together, based on the components' route sheet.

DASH et al. (2013) developed an ART-1 network for offline signature verification. Their main purpose was verifying similar looking but forged signatures. Especially for legal issues signature verification is very important. The authors compared serial and parallel processing for their network. Finally, both approaches are almost 100% accurate.

CHANDRALEKHA & PRAFULLA (2009) defined parameters for vertical handovers (for example 3G to 4G) between heterogeneous telecommunication networks using an ART network. The network selects the best available wireless network during handoffs based on a set of predefined user preferences on a mobile device. The ART network was used in order to overcome the problem of learning stability. The proposed vertical handover approach was compared with several others by GONDARA & KADAM

(2011). The other approaches considered more parameters. Therefore, the introduced network does not consider the factors "Power Consumption", "Received Signal Strength" and "Velocity".

CHEN et al. (2002) used an ART-2 network to classify customers into several groups and determine which features of a product, in their case golf clubs, are mandatory for each of them. The classification of the customers was based on age, gender and skill, whereas the features were for example price, usability and design.

The referenced papers show that ART networks are applicable for Operations, Emergency and Social Services, Marketing as well as Telecommunications.

2.2.3.4. Applications for Backpropagation Networks

WANG et al. (2011) constructed a backpropagation network for predicting stock indices. They chose a backpropagation network due to its popular use in the short-term forecasting situations.

FENG et al. (2011) combined the backpropagation network with other techniques in order to forecast the ozone concentration in cities. A support vector machine (SVM) was used to classify the data into corresponding categories. After the data classification with SVM the backpropagation network, having a genetic algorithm for weight optimisation, was used for the prediction.

CHE (2010) delivered a study for a cost estimation approach for plastic injection products and molds. With it, designers and R&D specialists can consider competitiveness of product costs in the early stage. Therefore, the approach combines factor analysis, particle swarm optimisation and two backpropagation networks.

NAWI et al. (2010) improved a backpropagation network which is used to predict patients with heart diseases. The proposed algorithm modifies the gradient based search direction by introducing the value of gain parameter in the activation function. The result is an enhancement of the computational effectiveness of the training process.

HANAFIZADEH et al. (2010) proposed an expert system based on a backpropagation neural network to help customers on perfume selection. They used demographic, product specific and customer's behavioural data to suggest appropriate perfumes.

KAEFER et al. (2005) compared a multinomial logit model (MNL) with a backpropagation neural network to predict the best timing for direct marketing activities. Their results show that the MNL is useful to determine chosen input variables, but the neural network achieves a higher accuracy rate.

LIN (2009) developed a backpropagation neural network as a tool to support fire in-

surance underwriters to estimate in-between risks. This gives them more discretion in pricing in order to compete on the market.

ALTIPARMAK et al. (2009) developed a general backpropagation neural network to estimate the reliability of telecommunication networks with identical link reliabilities. They demonstrate the precision of the neural network estimate of reliability and its ability to generalise to a variety of network sizes, including large scale communication networks.

As the proposed fields of application show backpropagation networks have a wide range of possible applications. Thus are for example Accounting and Finance, Health and Medicine, Marketing, Operation, Retail, and Emergency and Social Services.

2.2.3.5. Applications for Cascade-Correlation Networks

NASSIF et al. (2012) presented a Cascade-Correlation network for estimating the software costs in an early life cycle stage. They use Use-Case diagrams for prediction. The model was evaluated by using MMER[1] and PRED[2] as criteria.

CHANDRA & VARGHESE (2007) discussed the possibility of employing neural networks for the identification of cipher systems from cipher texts. In their paper they compared the Cascade-Correlation with a backpropagation network. The main goal is reducing the effort in developing new cryptographic algorithms. Therefore, the neural network checks if developed algorithms already exist. The Cascade-Correlation network outperforms the backpropagation network with approximately 91%-93% to 73%-85% accuracy.

DIAMANTOPOULOU (2006) developed a neural network which is able to compute the volume of pine trees having the trunk diameter at 0.3 and 1.3 meters height and the overall height of the tree as input variables. The network predicts the diameter on every additional meter height between 1.3 meters and the total height for every tree. These values are used to get the trunk volume over all trees.

HODNETT & HSIEH (2012) introduced a Cascade-Correlation network for financial fore-casting as an active portfolio management tool. In their study, the Cascade-Correlation network requires less input variables as a backpropagation network and had greater strength in prediction future top performers on global equities.

The given examples show that Cascade-Correlation networks are able to be used in the fields Finance and Accounting, Operations, and Emergency and Social Services. However, Operations includes the calculation of trunk volumes of trees and software development.

[1]MMER: The Magnitude of Error Relative to the Estimate (NASSIF et al. 2012)
[2]PRED: Prediction Level (NASSIF et al. 2012)

2.2.3.6. Applications for Kohonen Networks

DRAGOMIR et al. (2014) developed a Matlab application of a Kohonen network to classify consumers' daily load profiles in a smart grid with power generation from renewable energy sources. The network also determines the load behaviour of prosumers', who are connected with a smart grid that integrates renewable energy sources.
The proposed approach of DRAGOMIR et al. (2014) would be applicable to 'people's power stations'. Private households install renewable energy sources on their property, for example the house. The produced energy will be fed into the power network. Such power stations for example exist in Vienna.

OLSZEWSKI (2014) introduced a Kohonen network for fraud detection. Therefore, the user accounts are visualised using the network. Furthermore, a method for the detection threshold setting on the basis of the network's U-matrix is proposed. The approach was confirmed with experiments in different research fields. Those were Telecommunications, Computer network intrusion and Credit cards. The telecommunication experiment was based on fraud phone calls. The computer network intrusion experiment distinguished between DOS, probing, unauthorised access to a normal user and unauthorised access to a super user. The credit card experiment had the goal to detect illegal transactions.

CHON (2011) provides a research paper for Kohonen network applications on ecological sciences. These are for example molecules and genes, organisms, communities and populations as well as ecosystems. The presented applications affect animals, plants and humans. For example climate change, water resources, ecological management, social-economic behaviour of humans and response to toxic substances.

KIM et al. (2003) used a Kohonen network to cluster senior tourists of a specific region into several groups. The goal was to empower tourism marketers to justify selective marketing actions. Additionally, the information is useful for travel planners including agents and government offices which promote leisure activities.

CREPUT & KOUKAM (2009) combined an evolutionary algorithm with a Kohonen network in order to improve its application to the Euclidian Travelling Salesman Problem (TSP). They compared their solution with 91 publicly available Kohonen network based TSP solutions on standard test problems having between 29 and 85.900 cities. This network performs better regarding the solution quality and/or computation time.

KONECNY et al. (2011) used a Kohonen network to evaluate a survey on insurance companies' clients. They were grouped into satisfied, less satisfied and critical clients. Afterwards, the authors analysed the coordinates of the centre of the groups in order to determine the key attributes for each class. For example the key attributes for critical clients are possible improvements, information on the news about a company, use of accident insurance and probability of changing the insurance company.

The presented papers show the wide range of possible applications for Kohonen networks. These include the fields Retail, Operation, Insurance, Telecommunication, Finance and Accounting, Emergency and Social Services, Marketing, and Health and Medicine.

2.2.3.7. Applications for Counter-Propagation Networks

CHANG et al. (2010) introduced a method to apply Counter-Propagation networks to audio copyright protection. Therefore, a synchronisation code was added to the low-frequency components of candidate frames. This code works as watermark.

JUANG et al. (1998) compared a backpropagation and a Counter-Propagation network in the modelling of the TIG welding process. This process is used for welding aluminium, stainless steel, magnesium, titanium, etc. Factors like welding speed, wire speed and cleaning percentage have an influence on the weld pool. Thus defines the reliability, cleanliness and strength of the weld. This method is used for examples in the aircraft industries. The Counter-Propagation network learns faster, whereas the backpropagation network has better generalisation ability.

LIU (2010) developed a network to predict the customer maturity level for the financial industry. Based on customers' demographic data, such as age, gender, education and income the network calculates the maturity level. For example during the bachelor stage a customer wants to save money, in the early marriage state the focus is on buying a home and insurance needs followed by saving for retirement and children's education.

STOJKOVIC et al. (2010) use a Counter-Propagation network as a tool for development of interpretable quantitative structure-property relationship models for prediction of pKBH+ [3] values of a series of amides. Additionally, they identified the LUMO energies and the number of halogen atoms as most influencing inputs.

The given examples for practical applications of Counter-Propagation networks show, that these can be used in the field of Health and Medicine, Finance and Accounting, Operations as well as Insurance.

2.2.3.8. Applications for Perceptrons

MEMON et al. (2013) presented an artificial neural network based automatic volt regulator controller for an excitation voltage system of synchronous machines in order to investigate the applicability and to improve the transient response. A Multi-Layer perceptron network was used due to its proven applicability in power system control and stability. Simulations prove applicability by removing oscillations very quickly which

[3] pKBH+ describes the strength of the base's conjugated acid. In chemistry the value pKB describes the strength of a base. Contrary pKA describes the strength of an acid. Both values have an influence on the pH-value. pKBH+ states that a base molecule got a proton from an acid molecule. In this case the acid molecule is described with a pKA- value.

improves the transient stability of power systems.

RASHIDI & RASHIDI (2004) developed a Multi-Layer perceptron network using emotional temporal difference learning as training method to predict solar activity. A set of time series of sunspots was used for result comparison. The error signal has been employed as emotional signal in the networks. The results show that temporal difference learning based Multi-Layer perceptron neural networks are capable of improving the prediction accuracy than neuro-fuzzy models.

ORHAN et al. (2011) introduced a Multi-Layer perceptron based classification model as a diagnostic decision support mechanism in the epilepsy treatment. EEG signals were decomposed into frequency sub-bands using discrete wavelet transform. They performed five different experiments to obtain the performance of the model. The results show that the model provides an accuracy of more than 95%.

LEE & CHOEH (2014) created a prediction model based on a Multi-Layer perceptron network to predict the level of review helpfulness using the determinants of product data, the review characteristics, and the textual characteristics of reviews. The results of this study can be used to develop guidelines for creating more valuable online reviews. The study sought to explore the characteristics of online user reviews and how they influence the number of helpfulness votes.

MARQUES et al. (2014) implemented a Multi-Layer perceptron to determine how marketing decisions influence the delivery performance. The marketing decisions with the highest impacts are those related to the distribution channels.

The papers show that Multi-Layer perceptron can be used in Health and Medicine, Retail, Marketing, Operations and Emergency and Social Services. The Emergency and Social Services field fits because of the solar activity prediction, due to their influence on power systems.

2.2.3.9. Applications for Linear Associators

AMIN et al. (2011) developed an approach to separate fetal electrocardiography (ECG) from maternal ECG using the abdominal ECG. Therefore, a linear neural network is used. Input is the maternal signal and target is the composite signal. The network emulate maternal signal as closely as possible to abdominal signal, thus only predict the maternal ECG in the abdominal ECG. The network error equals abdominal ECG minus maternal ECG, which is the fetal ECG. The advantage for the patient is increasing convenience by reducing the monitoring time.

BEKRANI et al. (2011) proposed an adaptive filtering algorithm for stereophonic acoustic echo cancellation (SAEC) for a linear neural network. SAEC is a crucial part of stereophonic audio communication systems. These systems are advantageous over mono-

channel systems since they possess an inherent ability of transmitting spatial information in addition to the voice information.
Linear Associator are therefore applicable to health and medicine and operations.

2.2.3.10. Applications for Partial Recurrent Networks

The applications for Jordan-Nets and Elman-Nets are often proposed together as recurrent networks in literature, this section contains published scientific papers which contain both networks, whereas sub sections provide applications for either of them.

De MULDER et al. (2014) published a survey on the application of recurrent networks to statistical language modelling. Statistical language modelling contains applications like speech recognition, spelling correction, machine translation, distribution of words various linguistic units such as words, sentences, words and whole documents. They proposed extensions to recurrent networks in order to increase the applicability.

The paper, including those from the subsections, show, that recurrent networks are useful to Emergency and Social Services, Insurance, Operations, Telecommunication, Retail, Accounting and Finance, and Health and Medicine.

2.2.3.10.1. Applications for Jordan-Nets

MALLESWARAN et al. (2014) analysed the integration of GPS and Inertial Navigation System (INS) based neural networks with the weight optimisation techniques genetic algorithm and particle swarm optimisation. The analysed Jordan-Net provides superior performance in error efficiency and positional accuracy. In addition, it provides better performance in non-linear manoeuvring trajectories than the other compared networks.

CAO et al. (2012) compared the autoregressive integrated moving average (ARIMA) model with a Jordan-net for forecasting wind speed. Wind speed has an effect on the energy industry, aerospace operations and the insurance market. Their results show, that Jordan-nets are more accurate in predicting wind speed than ARIMA models. However, they stated that higher accuracy can be achieved by taking the altitude of wind speed measurement into account.
Insurance companies use risk assessment models to assess the financial risk of their insurance exposure due to windstorms. (KHANDURI & MORROW 2003)

2.2.3.10.2. Applications for Elman-Nets

CHENG et al. (2012) proposed an Elman-net for DNA segmentation. In their study they applied their network on the SARS and H1N1 genome. It presents a technology to

study genome sequences without prior biological knowledge and it processes the ATCG[4] sequences only. In addition the results are strikingly consistent with findings from biologists. Therefore, this approach can be used to rank parts of genomes.

LIN et al. (2006) developed an Elman-net for a dynamic portfolio selection model in the financial industry. Their Elman-net is compared with the vector autoregression model which is outperformed by the neural network.

DAS & CHAUDHURY (2007) researched on a backpropagation and an Elman-net to forecast sales of a footwear company. They concluded that a hierarchical Elman-net with two hidden layers provides the best performance to forecast the sales up to six weeks. Their approach can be applied on several stores and products in order to optimise short-term or middle-term stock planning.

SANTOS et al. (2014) used an Elman-net to improve the quality of experience of video transmission over IP networks. Their proposed method shall be seen as an alternative to server marked video frames. The network analyses the data flow and classifies the packets on using their size and time intervals as information. It shall preserve I-Frames[5] from being discarded because they are used for decoding B- and P-Frames[6].

[4] ATCG stands for the four nucleotides adenine, thymine, cytosine and guanine, found in DNA. (Nature Education 2009)

[5] I-Frame: Intra frame. It can be decoded without any other frame.(SANTOS et al. 2014)

[6] B- and P-Frames: Bidirectional and Predictive frames. The P-frames depend on information from the nearest previous I- or P-frames and the B-frames use past and future I- or P-frames as references for image representation. (SANTOS et al. 2014)

3. State of the Art of Neural Network Markup languages

When an author writes something he or she "marks it up". For example, spaces indicate word boundaries, commas indicate phrase boundaries and periods indicate sentence boundaries. The markup is not part of the text or content, but tells us something about it. If we read something we do not read but interpret it. For example, the voice gets higher, when we read a question mark. (COOMBS et al. 1987)

The authors distinguish between four types of markups: Punctuational, Presentational, Procedural and Descriptive.

- Punctuational markup consists of the use of a closed set of marks to provide primarily syntactic information.
- Presentational markup includes horizontal and vertical spacing, folios, pages, breaks, etc. For example, marking the beginning of a paragraph with spaces. Another example is marking the paragraph with numbers.
- Procedural markup often replaces presentational markup. It consists of commands indication how a text should be formatted.
- Descriptive markup identifies the element types of text tokens. For example "`<lq>`" for long quotations.

(COOMBS et al. 1987)

3.1. XML

Extensible Markup Language (XML) is a simple, very flexible text format derived from SGML (ISO 8879). Originally designed to meet the challenges of large-scale electronic publishing, XML is also playing an increasingly important role in the exchange of a wide variety of data on the Web and elsewhere. (W3C 1996-2003)

XML documents form a tree structure which starts from the root and branches to the leaves. The first line is the XML declaration. It defines the version and encoding of the XML document. The elements in an XML form a document tree. Elements can have sub elements which also may have sub elements. (Refsnes Data 1999-2014)

In addition elements can be extended with attributes, which provide further information about them. (Refsnes Data 1999-2014)

The following figure gives an example of an XML document.

```
<bookstore>
  <book category="COOKING">
    <title lang="en">Everyday Italian</title>
    <author>Giada De Laurentiis</author>
    <year>2005</year>
    <price>30.00</price>
  </book>
</bookstore>
```

3.2. Vienna Neural Network Specification Language (ViNNSL)

This section provides an overview of the current state of ViNNSL. It is mainly provided by the scientific paper "ViNNSL - The Vienna Neural Network Specification Language" by BERAN, et al.(BERAN et al. 2008)

By using ViNNSL to describe the semantics and behaviour of a neural network paradigm service, it is possible to create "dynamic" services that react on each input in a different way. The ViNNSL approach, seen as a semantic language standard, aims to serve specifically the domain of neural networks by providing five schemata to manage the interaction between a client and a neural network service provider. Using these schemata users have the possibility to describe attributes like service capabilities, semantics, functions and parameters in a client interpretable way. (BERAN et al. 2008)

Because of the following reasons, the ViNNSL supported dynamism is much more powerful, than the usage of ordinary service data and interfaces only:

- By decoupling the language parts into five different schemata only a smaller part of the system has to be changed or extended in case of a schema change.
- The client can implement and interpret different semantic schemata (descriptions) and map them to one common service interface.

(BERAN et al. 2008)

ViNNSL consists of five parts: description schema, definition schema, data schema, instance schema and result schema.

Figure 3.1 shows the description schema. It builds the first part of the schematas that

are required to support the dynamism of a ViNNSL based neural network creation, training, retraining and evaluation process. (BERAN et al. 2008)

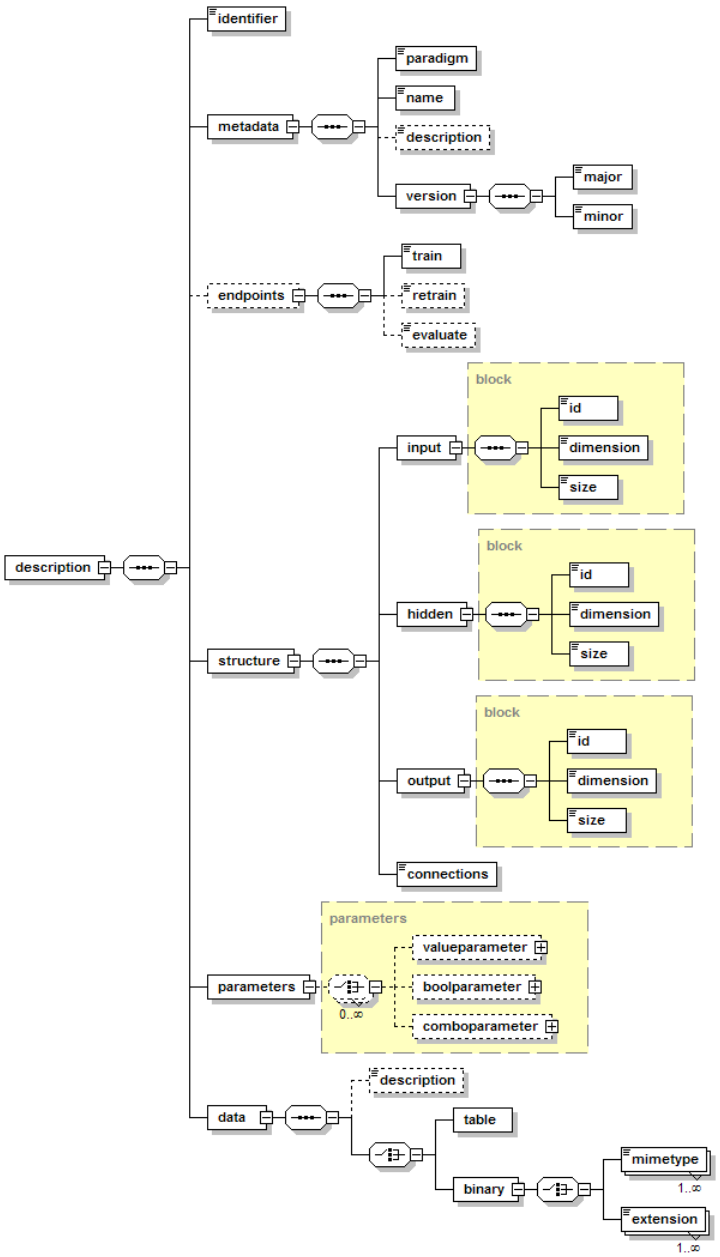

Figure 3.1.: ViNNSL Description Schema (BERAN et al. 2008)

The *identifier* field identifies a neural network implementation by a unique MD5 hash. The *metadata* element provides other fields for describing the service implementation and its mode of operation. The *paradigm* field is used as hint about the realised network type while *name* extends this information and specifies the service in a more precise way. However, with the usage of the *description* field users are able to describe the area of application as well as the domain of the neural network solution. Finally the *version*

tag contains a *major* and *minor* element to distinguish between different versions of a neural network implementation. (BERAN et al. 2008)

The Web Service endpoints to *train*, *retrain* and the *evaluation* method can be provided here too. The *structure* section, describes the topology of the neural network. It contains an *input*, a *hidden* and an *output* block each having an *id*, a *dimension* and a *size*. Within a service description these fields are blank. Only the *dimension* and *size* element contain an additional *min* and *max* attribute that have to be set in order to define the borders of the dimension (number of neurons inside a layer) and the size (number of layers of the used type). (BERAN et al. 2008)

A zero as value for the size-min attribute means, that this layer is not mandatory. However, a size-max attribute's value of zero means, that the layer is not supported in this particular neural network. In addition to the structure description the connectivity can also be pointed out by means of the *connection* element. Possible values are "full connected" (network supports full connections only), "shortcuts" (user has to specify each neuron- to-neuron connection) or "mixed".

The *parameter* section groups elements of type *valueparameter*, *boolparameter* or *comboparameter* together. *Valueparameter* allows specifying floating point numbers, *boolparameter* requires the selection of a logical value and *comboparameter* represents a list of selectable items. Examples are learning rate for *valueparameter*, bias for *boolparameter* and functions for *comboparameters*. The *data* section specifies the data which is required for different neural network operations (train, retrain, evaluate). The *description* is optional. However, the data format can be specified by using the table (e.g. csv-like data) or binary (e.g. base64-coded data) element. For the latter type the nature of accepted files can be specified using allowed mime types or possible file extensions. (BERAN et al. 2008)

The *definition* schema shown in 3.2 is user driven and represents an XML based formal specification of a newly created neural network that has to be trained. Like in the description schema the identifier element tags a neural network implementation. (BERAN et al. 2008)

The *structure* section in the definition schema is similar to the description schema's section. It contains a set of layers of type *input*, *hidden* or *output*. Each layer needs an *id* and a *dimension* (number of neurons in the layer). The size element is usually set to one, nevertheless a higher setting implies that the neurons within this layer are arranged as a multi-dimensional matrix instead of a uni-dimensional list. The neuron connections between different layers are defined in the *connections* section. The *fullconnection* element allows the user to specify neurons of a layer that are connected to neurons of another layer. Specific neuron-to-neuron connections are defined by using the *shortcut* element. (BERAN et al. 2008)

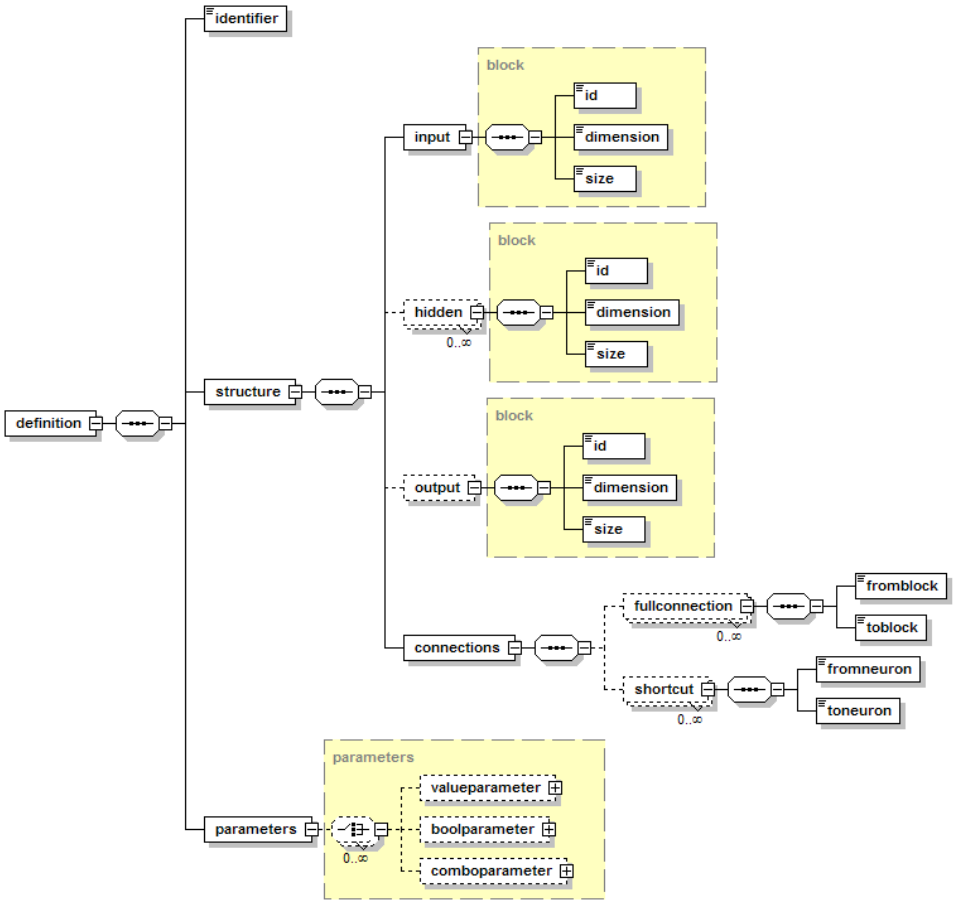

Figure 3.2.: ViNNSL Definition Schema (BERAN et al. 2008)

At the end of the document various *parameters* can be set defining the *value* element of each parameter.

However, the definition itself is not sufficient enough to get a trained neural network. In addition a training *data* document is required. This can be defined according to the data schema shown in 3.3. The data approach distinguishes between two fundamental cases. In the first case a neural network is used for which training data is based on *samples* consisting of a list of input values and possibly a list of *output* values. Besides providing csv-like data it is also possible to submit data files by using *binarysamples*. Each of these samples consist of a number of *input* and *output* file elements containing a *name* (filename), a *mimetype* and a *content* encoded as *xs:base64Binary*. (BERAN et al. 2008)

The *instance* schema, which is shown in 3.4 gives the user all possibilities to store a neural network in any kind of XML structured document. The content of this element is defined as *xs:anyType* and can be furthermore integrated in any *result* document. (BERAN et al. 2008)

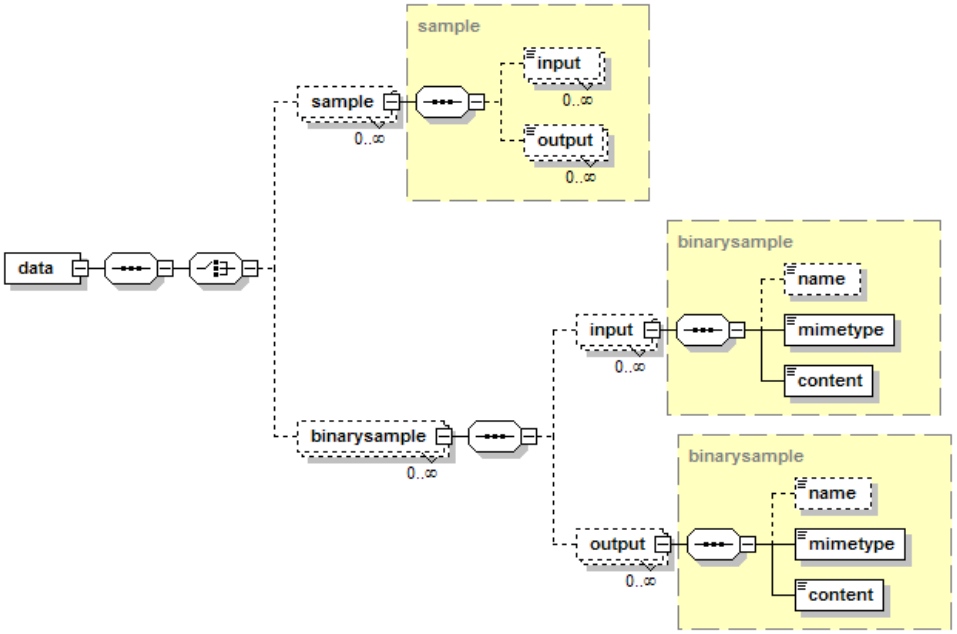

Figure 3.3.: ViNNSL Data Schema (BERAN et al. 2008)

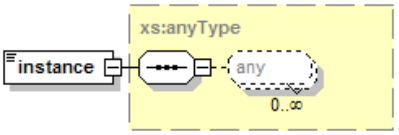

Figure 3.4.: ViNNSL Instance Schema (BERAN et al. 2008)

The last schema is the *result* schema which contains an *instance* document as a subpart and is shown in 3.5. The *identifier* is used to address the producer of a result. The *date* section contains a timestamp which stores temporal information about the execution time of the neural network activity that generates the result. The content of the *instance* element varies for different neural network implementations. However, it does always contain some serialised trained, retrained or evaluated neural network.

The *diagram2d* section is an XML based approach to provide information for drawing a two-dimensional diagram that may contain an error curve or another vector based drawing. The diagram has a *title* (name), a *description*, a *type* (lines or points), an *xaxis*, a *yaxis* and some values. Each axis has a *title*, a *description*, a *min*-value (lower bound) and a *max*-value (upper bound). Every *value* entry inside the *values* element is rendered as a line or a sum of points using a different colour.

The *table* section contains a list of *rows* which provide *input* and *output* floating point data samples. Using a *file* with its *binaryrows* is useful for neural network services with file based returns. *Error* and *status* can be provided inside the *messages* section.

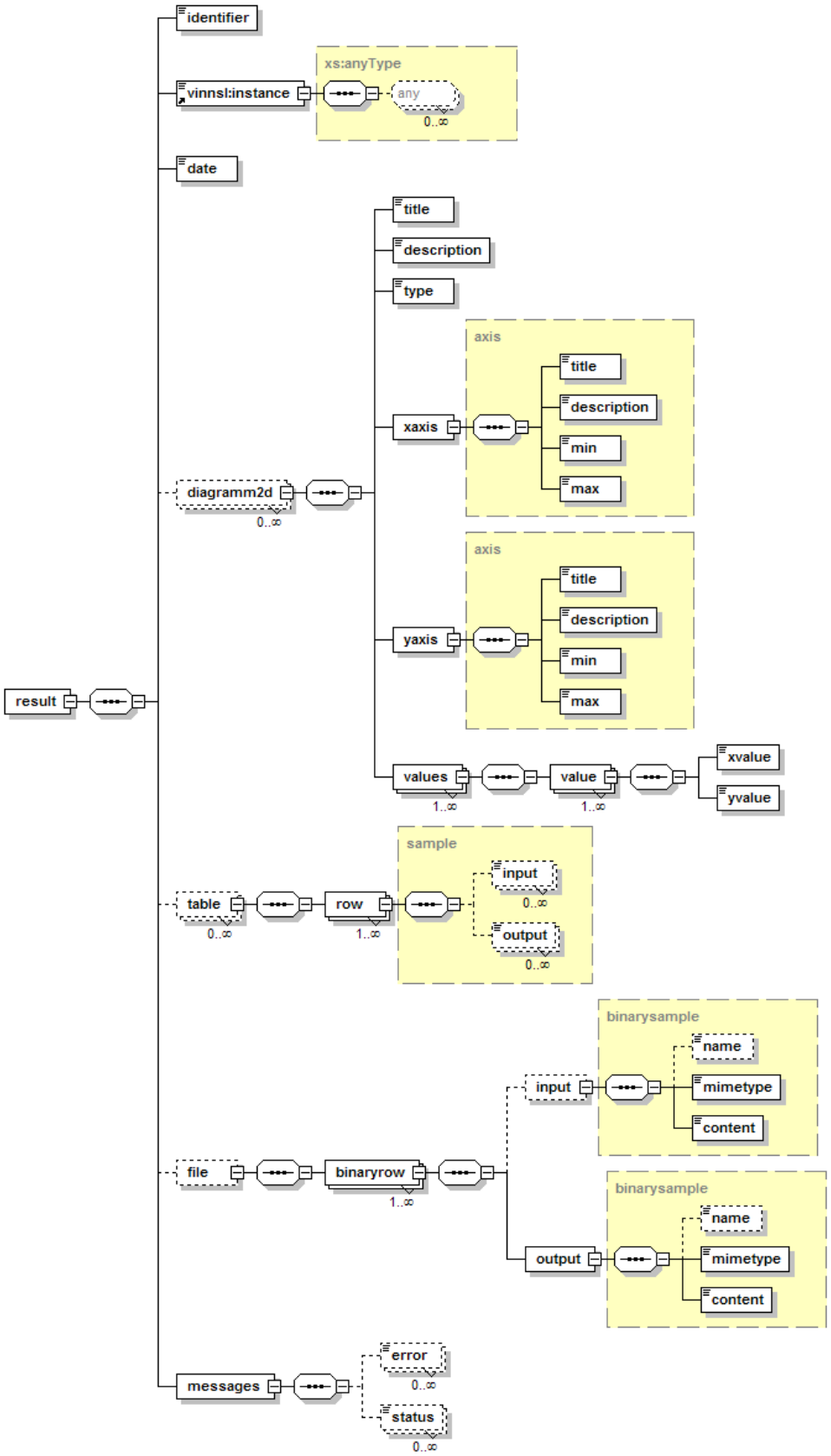

Figure 3.5.: ViNNSL Result Schema (BERAN et al. 2008)

3.3. Neural Network Cube (N2Grid)

The N2Grid System (Schikuta & Weishäupl 2004) is a further development of the NeuroWeb, which is an internet based neural network simulator used by students and researchers of the University of Vienna. (Schikuta 2002) N2Grid is a Java based environment for a distributed neural network simulation. Apache Axis library and Apache Tomcat Web container are used as hosting environment for the Web Services. They distribute the components of N2Grid. For the Web frontend Java Servlets/JSPs have been employed. This leads to the advantage of keeping the overall installation requirements quite low and to tie up a handy to deploy installation package. The whole system architecture and its components are depicted in figure 3.6. (BERAN et al. 2008)

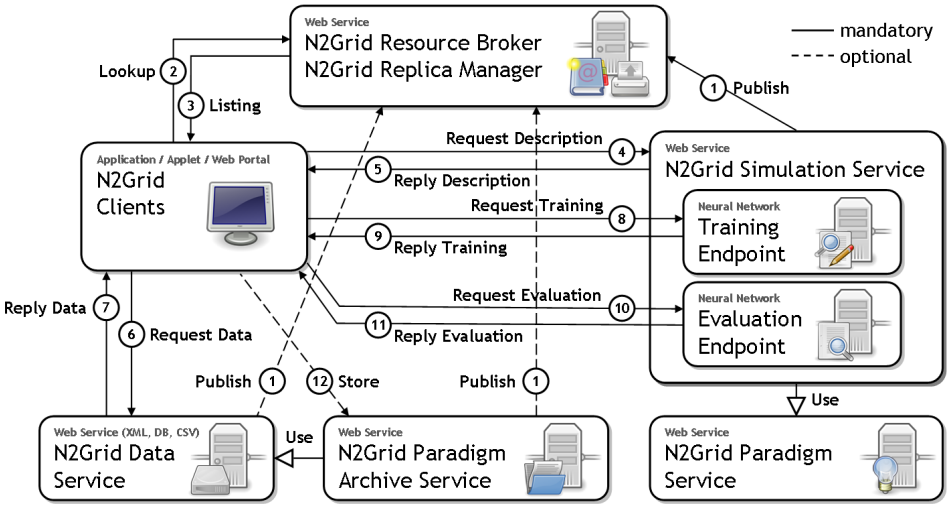

Figure 3.6.: N2Grid Architecture and Components (BERAN et al. 2008)

3.3.1. N2Grid Architecture

The N2Grid system is based on service oriented architecture. The original idea behind this system was seeing every part of an artificial neural network as data object, which can be serialised and stored at some data site. Since the new WSRF standard every data site is called "resource" and is usually implemented as a Web Service. Following Gundry's notion of "information" a layered architecture is suggested to depict the dimensionality of the different layers:

- Data Grid (zero-dimensional): The Data Grid builds the basis layer and stores data that represents just facts.

- Information Grid (one-dimensional): The Information Grid collects data of the Data Grid in a structured manner and attributes it with semantic content.

- Knowledge Grid (two-dimensional): The Knowledge Grid provides problem solution mechanisms on the administered information allowing a human for acting, deciding or planning.

In most layered architecture, layers relay their functionality to the next upper layer in form of services and data. Within the current version of N2Grid, these layers are realised and mapped to specific services in the system. (BERAN et al. 2008)

3.3.2. N2Grid Components

The components in 3.6 are starting at the bottom layer.

N2Grid Simulation Service: This service uses a paradigm implementation respectively a Paradigm Service for executing one of the following three actions:

- Train: Training of an untrained neural network.
- Retrain: Training of a previously trained network again in order to increase the training accuracy.
- Evaluate: Evaluating a trained network.

(BERAN et al. 2008)

N2Grid Data Service: This service provides access to distributed data sources by using protocols like GridFTP, OGSA-DAI or just HTTP. These data sources can provide either training or evaluation data as well as trained neural network instances. The service offers the following methods:

- put: Inserts data into a data source.
- get: Retrieves data from a data source.

(BERAN et al. 2008)

N2Grid Paradigm Archive Service: Sometimes users want to find already trained networks in order to use them as generic problem solvers. On the basis of Data Services this service archives implementations of neural network paradigms and provides them to users on a persistent basis. (BERAN et al. 2008)

N2Grid Resource Broker: This component keeps track of available services and acts as a single point of contact. Jobs can be submitted here instead of submitting them to a Simulation Service. The broker is furthermore tightly coupled with at least one Replica Manager and enables the user to search for different paradigms, e.g. Back Propagation, Quick Propagation, Jordan, ART-x, etc. (BERAN et al. 2008)

N2Grid Replica Manager: The Replica Manager replicates existing services to increase the overall system performance. The reason is avoiding time-consuming network data transfers following the "Owner-Computes" rule. In case of Paradigm Services, which offer some parallel processing capabilities, this manager controls the distribution of workload and data. (BERAN et al. 2008)

N2Grid Paradigm Service: It contains the paradigm implementation that can be seen as the business logic of a neural network. (BERAN et al. 2008)

N2Grid Java Application/Applet: The Application Client is intended to support experienced users to run their simulations by providing data stored in local databases. The Applet Client is very similar to the Application Client but has some limited functionality. Due to sandbox restrictions accessing local data sources and files is not allowed. (BERAN et al. 2008)

N2Grid Web Portal: For the purpose of thin clients a simple Web browser - preferably Internet Explorer or Firefox - can be used to access the front end of N2Grid, a Web Portal Client. It provides control over running simulation jobs and presents the calculated results to the user. This approach minimises the workload on the local machine. (BERAN et al. 2008)

The Web portal uses ViNNSL as a dynamic GUI interface language which renders each user control by using the Paradigm Service's description. Amongst others this allows advanced result presentation by using two-dimensional vector graphic based diagrams. By means of providing an appropriate document wrapper for the ViNNSL language it can also be converted to other languages like Mozilla's XUL. (BERAN et al. 2008)

3.4. Other Languages

This chapter gives an overview of other neural network specification languages based on XML.

3.4.1. iXCSL

iXSCL stands for Extensible Soft Computing Language and is an XML vocabulary for the specification of common objects in the Soft Computing area. The first version only considered Fuzzy systems. Later on iXCSL was enhanced in order to describe neural networks. (de SOTO et al. 2003)

The model describes the architecture of a neural network as a set of interconnected layers, and each layer as a set of interconnected processing nodes. Processing nodes defined so far include artificial neurons and bias nodes. This model does not address operations like training or testing the network, it only includes the necessary information

to compute its outputs given a set of inputs. (de SOTO et al. 2003)

In order to describe a neural network a set of layers is defined and connections between them declared using a signalFlow element. This element is similar to the one inside each layer, but here the *name:oN* represents the n-th output of the name layer. The description is completed with a stop criterion which can be either a fixed number of iterations or stability. (de SOTO et al. 2003)

The iXSCL schema defines two processing node types: the classic artificial neuron and bias nodes. The *nonLinearNeuronNode* carries an activation function, weights and optional named tags. Tags can be used to associate attributes to the neuron. Bias nodes provide a constant output of a given real value. Nodes are defined inside a layer, which also describes the set of connections between them. A *signalFlow* element declares the inputs' source of every node and which nodes provide the outputs of the layer. A simple syntax is used where *iN* represents the n-th input of the layer, *oN* is the n-th output, and the id of a node. The whole layer is given a synchronous or asynchronous model with a model attribute. (de SOTO et al. 2003)

```
<neuralNetwork name="Hopfield">
    <parameters>
        <input name="input-vector" type="/sensors/NoiseSample"/>
        <output name="output-vector" type="/levels/NoiseLevel"/>
    </parameters>
    <networkLayers>
        <layer id="lattice" nInputs="3" nOutputs="3" model="asynchronous">
            <processingNodes>
                <node id="n1" xsi:type="nonLinearNeuronNode" nInputs="5">
                    <activationFunction xsi:type="thresholdActivationFunction">
                        <min>0.0</min>
                        <max>1.0</max>
                    </activationFunction>
                    <weights>
                        0.123124124 0.259528285 0.87721231
                        0.1928123 0.12938241
                    </weights>
                    <tags>
                        <tag name="label">Class A1</tag>
                    </tags>
                </node>
                <!-- other nodes: n2, n3 -->
            </processingNodes>
            <signalFlow>
                <inputs node="n1">i1 i2 i3 n2 n3</inputs>
                <inputs node="n2">i1 i2 i3 n1 n3</inputs>
                <inputs node="n3">i1 i2 i3 n1 n2</inputs>
                <outputs>n1 n2 n3</outputs>
            </signalFlow>
        </layer>
    </networkLayers>
    <signalFlow>
        <inputs node="lattice">i1 i2 i3</inputs>
        <outputs>lattice:o1 lattice:o2 lattice:o3</outputs>
    </signalFlow>
    <stopCriterion xsi:type="stability"/>
</neuralNetwork>
```

3.4.2. Neural XML (NXML)

Neural XML is an XML based language to create, train and run neural networks. It allows generating, saving and loading neural networks to/from XML files. (MADHUSUDANAN 2006)

The method uses a description file, which describes the neural network. A network

consists of named layers and neurons. Each neuron has a bias, output and delta value. The hidden and output layer neurons additionally have a connection tag providing the input's source in form of layer and neuron as well as a weight. The user has the possibility to let the NXML generate a neural network description by giving the number of neurons per layer or create an own description file. (MADHUSUDANAN 2006)

A separate file contains all information, which is required in order to train or run a network. For example, the network, which shall be used, the operations, which have to be performed and in which file the results have to be saved. It is possible to define the number of training cycles, the input and output datatype as well as their values. The training results are stored in newly created files. (MADHUSUDANAN 2006)

NXML supports *PatternData* and *ImageData*. *PatternData* consists of Patterns, Numbers, Arrays and Characters as input and output types. *ImageData* requires an image file as input. In addition, its width and height, output type and value have to be given. (MADHUSUDANAN 2006)

3.4.3. NNDef

MAKHFI (2001-2011) states that the goal of NNDef is to "facilitate exchange and execution of neural networks in a standard way". It is a Java toolkit composed of a DTD[1], a Runtime Engine, a Runtime Library and a Matlab Exporter.

The Runtime Engine uses the defined neural network to process input data and generate output. The Runtime Library can be embedded in other applications to use NNDef defined neural networks. The Matlab Exporter is used to empower Matlab to import trained networks. (MAKHFI 2001-2011)

An NNDef XML document is thus structured in exactly one network and one to several layers having one to several neurons. The network contains a description, a name, a creation date and an author. A network has inputs and outputs. Each neuron in a layer is described with input weights and a bias value. Furthermore, a neuron has a transfer function and a combination function. (MAKHFI 2001-2011)

3.4.4. Artificial Neural Network Specification Language (ANNSL)

BARTZ (2008) developed an XML based language to describe neural networks for documenting and exchanging them between scientific and production tools. The paper determined the information classes, which are required for the usage of neural networks.

[1]DTD: The DTD (Document Type Definition) defines the legal building blocks of an XML document. A DTD defines the document structure with a list of legal elements and attributes. (Refsnes Data 1999-2015)

ANN Input Information contains administrative data plus one to many net input data sets $x^{(i)}$, N values each. (BARTZ 2008)

ANN Target Output Information contains administrative data plus one to many target output data sets $Y^{(i)}$, M values each. (BARTZ 2008)

ANN Output Information contains administrative data plus one to many net output data sets $y^{(i)}$, M values each. (BARTZ 2008)

Training Result Information contains administrative data, information on the initial ANN configuration, and information on a number of epoch results. The information on an epoch result may contain information on the errors observed during the epoch, on the ANN configuration at the end of the epoch, and on results of the individual training sets. Results of an individual training set may contain information on the resulting net output data set ${}^{(e)}y^{(i)}$, on the errors generated by this training set, and on the resulting ANN configuration after this training set has been used for training. Most of these information fragments are optional as there is not always the need to document all details of a training process. (BARTZ 2008)

ANN Configuration Information contains the complete configuration of the ANN.

The language consists of two specifications: Data Store Specification and Configuration Storage. The Data Store Specification contains in sequential order *NetInputData*, *TargetOutputData*, *TrainingResultData* and *NetOutputData*. However, *NetInputData* is mandatory. (BARTZ 2008)

NetInputData (figure 3.7) contains the following sections *InputCount* (Number of values per data set), *Name*, *Description*, *CreateDate*, *Creator* (has sub elements *Name* and *Contact*), *DataType*, *Precision* and *InputDataSet* (at least one is mandatory and has an index number as attribute).

As numeric precision is an important criterion for some applications, such information may be provided in the Precision element within each information class. It provides the smallest positive number that may be distinguished from zero. If required, the precision may be specified individually for each net input respectively output. Such information may be used for rounding, for deciding on training continuation, for textual or graphical presentation, etc. Currently, the data store specification supports simple numeric data types. Their kind is given by the DataType element. These may be extended in the future towards strings, composite types, and others. (BARTZ 2008)

TargetOutputData differs in having the equivalent elements to *InputCount* and *InputDataset*. In addition the value *setIndex* connects *InputData* with *TargetOutputData*. *NetOutputData* has additionally the possibility to store error information collected dur-

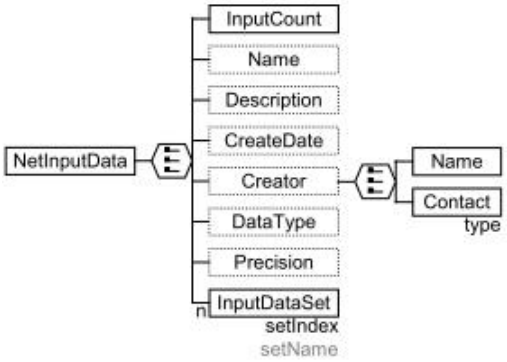

Figure 3.7.: NetInputData (BARTZ 2008)

ing training in case *TargetOutputData* was given. (BARTZ 2008)

TrainingResultData must be able to incorporate results of the whole training process. Therefore, it contains the element *EpochResult* (figure 3.8). *EpochIndex* gives the number of the epoch and *setIndex* the number of the corresponding *TrainingResultSet*. *SetIndex* is the link to the *InputDataSet* and its *TargetDataSet*. In this case all or predefined numbers of epochs and *trainingresults* can be chosen. (BARTZ 2008)

Figure 3.8.: EpochResult (BARTZ 2008)

As numeric precision is an important criterion for some applications, such information may be provided in the Precision element within each information class. It contains the smallest positive number that may be distinguished from zero. If required, the precision may be specified individually for each net input respectively output. Such information may be used for rounding, for deciding on training continuation, for textual or graphical presentation, etc. (BARTZ 2008)

The configuration schema is shown in figure 3.9. According to BARTZ the part *General* and one Layer are necessary.

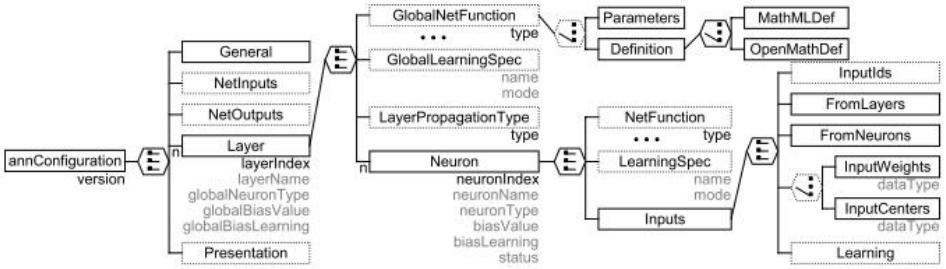

Figure 3.9.: Overview of the ANNSL configuration schema (BARTZ 2008)

The data stored in *General* and one layer is the minimum information required to specify a neural network with ANNSL. Net function, activation function and output function can be declared at layer or on neuron level. *NetInputs* are not explicitly modelled. They are formally assigned to layer zero and are referenced by a virtual neuron index starting with one. If the index needs to be modified *NetInputs* can be used. They represent the input values for the net and are used to route these values to neurons within the neural network. Net outputs are identical to the output of the highest layer index. This default behaviour can be modified with the element *NetOutputs*. (BARTZ 2008)

All connections to one neuron are stored within one XML element. Individual inputs are separated by XML whitespace. This reduces redundancy, avoids possible connection conflicts and provides a compact notation. ANNSL also provides the possibility to exclude neurons from the learning process using the Learning attribute. Special attention has been paid to the representation of functions within neurons. Pre-known functions like the weighted net function or sigmoid function can be defined by their name, whereas customised functions can be defined by using a specification language for mathematical expressions based on XML. (BARTZ 2008)

3.5. Comparison of ViNNSL with other languages

The iXCS-Language describes an already trained neural network, which can be used to process data. It is comparable to the definition schema of ViNNSL. However, unlike ViNNSL iXCSL doesn't provide the possibility to train or evaluate networks. The connections, layers, neurons and other information have to be given in order to use the network.

ViNNSL instead expects the definition on layer level. These are the layer ID, number of neurons within a layer and number of a specific layer type. The weights between neurons are calculated by the network. However, it is possible to define specific connections between neurons and layers, but these have to be defined separately. Moreover, ViNNSL doesn't provide the possibility to maintain weights and bias values of neurons.

	ViNNSL 1.0	iXCSL	NXML	ANNSL	ViNNSL 2.0
Technology	XML Schema	XML Schema, XSL Transformation	XML Schema	XML Schema	XML Schema 1.1
Neuron types	Artificial neuron, optionally bias neuron	Artificial (non linear) neuron, separately bias neuron	Artificial neuron, mandatory bias neuron	Artificial neuron, optional bias neuron	Artificial neuron, optionally bias neuron
Creation	Yes	Yes	Yes	Yes	Yes
Training	Yes	No	Yes	Yes	Yes
Evaluation	Yes	No	Yes	Yes	Yes
Inputs	Numerical data, binary data	Not specified	Pattern, Array, Number, Character, Image	Numerical data	Numerical Data, Alpha-numerical Data
Outputs	Tables, 2D-diagrams, files, messages	Not specified	Pattern, Array, Number, Character	Numerical data	Tables, 2D-diagrams, files
Training analysis	No	No	No	Yes	Yes
Select functions	Yes	Yes	No	Yes	Yes
Customised functions	No	No	No	Yes	No
Configuration					
Provide signal flow	Obligatory	Mandatory	Obligatory	Obligatory	Obligatory
Provide weights	No	Mandatory	Obligatory	Obligatory	No
Provide number of neurons	Range is mandatory	Yes	Yes	Yes	Yes
Provide number of layers	Range is mandatory	Yes	Yes	Yes	Yes

Figure 3.10.: Comparison of neural network specification languages modified from BERAN et al. (2008)

NXML offers the possibility to generate a neural network by giving the number of neurons per layer. Every neuron gets input from the neurons from the previous layer. ViNNSL for example offers the opportunity to create shortcuts between single neurons

across layers. In addition, ViNNSL enables the user to define a range of neurons for each layer. The NXML description file is comparable to the ViNNSL definition file.

Training results are stored in new files by NXML. Each file can be used as network, which can be executed. These files have the same state like the ViNNSL instance schema files. The files containing training and execution data are the equivalent to the data files of ViNNSL. However, the execution file also provides the results, whereas ViNNSL provides a separate result file.

The ViNNSL Data schema is the equivalent to ANNSL data store and the Definition schema to the ANN configuration. The ANNSL data store provides sections for different types of input and output, whereas ViNNSL expects a separate file for training or data processing. ANNSL saves the networks result in a data store section, while ViNNSL provides a separate result schema. Unlike ViNNSL, ANNSL delivers detailed epoch results during the neural network training.

In addition, ANNSL provides additional information for the net input data, whereas ViNNSL delivers a detailed result file. Furthermore, ANNSL allows defining neural networks on a very detailed level.

ANNSL empowers the user to define a neural network down to the specification of input parameters like weight, function, source layer and source neuron as well as excluding neurons from the learning process. On layer level, it allows defining customised functions using MathML and OpenMath definitions. These are XML based notations for the representation of mathematical functions.

ViNNSL instead let users set a learning value, a list of functions and the use of biases on network level. In addition it is possible to define a full connection of the network or shortcuts. Latter requires specific neuron-to-neuron connections or combine those.

The attributes of ViNNSL 2.0 will be explained in detail in chapter 4 ViNNSL 2.0.

4. ViNNSL 2.0

This chapter explains the ViNNSL extensions, which are based on the findings from the previous chapters. Models showing the structure and elements of every schema as well as associated XSDs will be presented. The first part of this chapter is the determination of influences of the other neural network specification languages summarised in this thesis. The extensions and adaptions to ViNNSL, shown in the structure and element models, are highlighted red. As fact that already existing elements are described in section 3.2, only the newly-formed elements are explained.

4.1. Influences of other Neural Network languages

The description schema is influenced by ANNSL. The possibility to add a creator and contact data to the network was inspired by the proposal of BARTZ (2008). This proposal also influenced ViNNSL by adding a training result schema. Moreover, providing a creation date and an author was also proposed by MAKHFI (2001-2011).

The data representation in table form is inspired by ANNSL. In order to keep the readability XML whitespaces are used to separate values within an element. Another influence lead to the training result schema. Users shall have the possibility to analyse the training process of networks.

The iXSCL schema proposes the possibility to include bias nodes into the network. This option was not covered by the first proposal of ViNNSL.

4.2. Extensions to ViNNSL

This section provides explanations on every extension to ViNNSL ordered by each schema. However, extensions are explained only once.

creator: The creator element contains the neural network creator's name and contact data, which are stored in the *name* and *contact* elements.

problemDomain: This element represents the neural network classification described in chapter 2. Assertions are used to ensure that *propagationType*, *learningType* and *networkType* fit together.
propagationType: It defines the propagation type of the used neural network.

learningType: It gives the learning type of the used neural network.
applicationField: It specifies the fields of application for the neural network.
networkType: It defines the type of the described neural network.
problemType: The problem type is based on the description in section 2.2.1

The elements *propagationType*, *learningType* and *networkType* are used to implement the neural network classification ontology described in section 2.1 and shown in figure 2.1, in XSD language. Their task is providing a unique identification of neural network types.

ApplicationField does currently provide examples for possible applications. This part of the neural network categorisation can be extended by neural network creators. They can either use one of the proposed values or create their own.

ProblemType contains the problem, which is solved by the neural network. The used ontology is based on the definitions of Jacek Zurada, which are provided in section 2.2.1.

The neural network classification ontology and the problem domain ontology shall provide all necessary attributes for a neural network characterisation. Within ViNNSL 2.0 these ontologies are asserted in order to ensure the validity of characterisations.

executionEnvironment: This element covers the execution paradigm taxonomy shown in figure 4.1.

The concrete values of the parallel execution are stored as values within the leaf nodes of this taxonomy. These elements also contain an attribute specifying the version of the used technology.

Examples for parallel executed networks are given in chapter 5. However, parallel execution can be implemented for other networks than backpropagation nets too, as proposed by Schikuta & Weidmann (1997) and Weishäupl & Schikuta (2003). Schikuta & Weidmann (1997) introduced a data parallel simulation on a hypercube system similar to the example shown in section 5.2, but for self-organising maps. Weishäupl & Schikuta (2003) introduced parallelisation for CNN, which are designed for image processing, comparable to UC1 (see section 5.1). Moreover, they also showed the possibilites of the parallel implementation of neural networks on the TSP problem. (Schabauer et al. 2005) Their implemented Kohonen network was executed with 100000 cities, unlike mentioned implementations from CREPUT & KOUKAM (2009).

sequential: The neural network is executed sequentially. The default value is true.
parallel: This element contains all settings for a parallel execution of a neural network
software: The selected software technique to implement parallelism.

control: Control-parallel architectures perform processing in a decentralised manner,

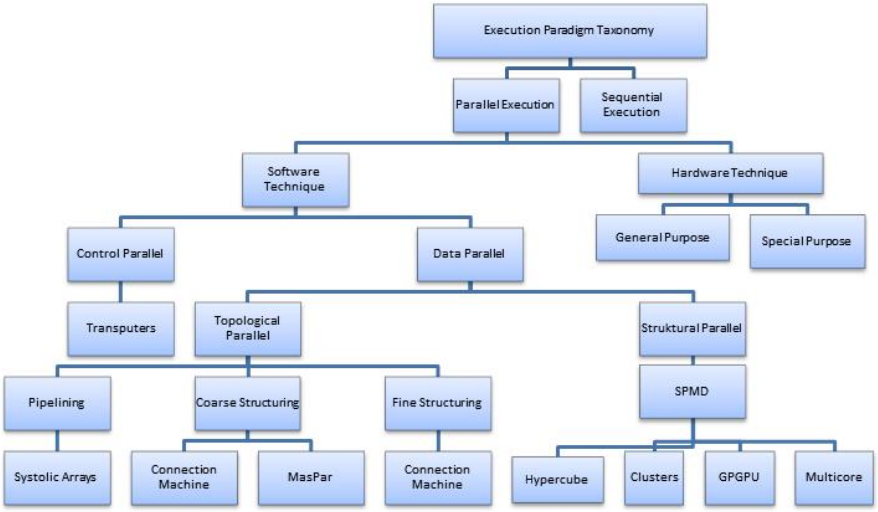

Figure 4.1.: Execution Paradigm Taxonomy (SCHIKUTA et al. 2015)

allowing different programs to be executed on different processors (typically: multiple instruction, multiple data). (SERBEDZIJA 1996)
transputer: The transputer architecture defines a family of programmable VLSI components. The definition of the architecture falls naturally into the logical aspects, which define how a system of interconnected transputers is designed and programmed, and the physical aspects, which define how transputers, as VLSI components, are interconnected and controlled. (Inmos 1987)

dataparallel: Data-parallel architectures simultaneously process large data sets using centralised (typically: single instruction, multiple data) or regular (for example: pipelined) control flow. (SERBEDZIJA 1996)
topological: Topological parallel means mapping of neural network elements (like neurons, links, etc.) to processor elements, like node-per-layer, single-node, or systolic arrays. (SCHIKUTA 1997)

pipelining: It is an implementation technique where multiple instructions are overlapped in execution. The computer pipeline is divided in stages. Each stage completes

a part of an instruction in parallel. The stages are connected one to the next to form a pipe - instructions enter at one end, progress through the stages, and exit at the other end. (PRABHU no date)
systolicarr: Systolic arrays are numerous simple processors arranged in one- or multidimensional arrays, performing simple operations in a pipelined manner. Circular communication ensures that data arrives at regular time intervals from (possibly) different directions. (SERBEDZIJA 1996)

coarsestruct: Coarse structuring means each processor is used to store a node from each of the layers, so that a "slice" of nodes lies on a single processor. The number of processors needed to store a network is equal to the number of nodes in the largest layer of the network. (SERBEDZIJA 1996)
connmachine: The Connection Machine is like an active memory made up of thousands, potentially millions, of small, simple, processors working simultaneously, each with its own tiny memory. (BLACK 1986)
maspar: MasPar stands for Massively Parallel Machine. The concept of Massively Parallel is a machine, which incorporates massive amounts of processing elements. By using a *Distributed Memory* architecture (one which combines local memory and processor at each level of the interconnection network) machines with almost an infinite number of processors without compromising design can be created. The only thing that limits these types of computers is the cost of the processing elements. (Corporation no date)

finestruct: For *Fine Structuring* processors were organised in a one-dimensional array with one processor being allocated to a node and two processors to each connection. One for the output and one for input side of a connection. (SERBEDZIJA 1996)

structural: Structural parallelism maps data structures representing neural network information containers (as weight matrices, error value structures, input vectors, etc.) onto processing elements according to a data-parallel scheme. (SCHIKUTA 1997)
spmd: Single-Program-Multiple-Data is the dominant style of parallel programming, where all processors use the same program, though each has its own data. (PIETERSE & BLACK 2004)
hypercube: It refers to a parallel computer with a common regular interconnect topology that specifies the layout of processing elements and the wiring in between them. (STRICKER 2011)
cluster: A commodity cluster is a distributed computer system consisting of an integrated set of fully and independently operational and marketed computer subsystems (node) used together to perform a single application program or workload. (PADUA 2011)
gpgpu: The gpgpu application systems use the gpu as a group of fast multiple coprocessors, that execute data-parallelised kernel code. (JUNG 2008)
multicore: The multicore system is a system with two or more cores on a single chip. In the homogeneous multicore system, it is used to program on the SPMD model. (LEE et al. 2010)

hardware: This element specifies the used hardware for the neural network implementation.
general: The hardware is not designed for a specific purpose.
special: The hardware was designed for parallel neural network execution.

endpoints: The endpoint element defines if a neural network is going to be trained or retrained.
resultSchema: It states which schema is a result of the execution.
dataSchemaID: This element is the link to the data for the neural network execution.
instanceSchemaID: If a neural network is retrained, it requires an existing instance.

The definition schema asserts for retraining, if a neural network gets an existing neural network instance. Schema IDs which are elements within a schema, connect schemas for the neural network execution

identifier: Unique identification of a schema instance.
table: Data is represented in table form within a data schema instance.
file: Link to a file containing the data for a neural network execution.

The data element is asserted to check if data is provided. In ViNNSL 2.0 the data schema is extended with an identifier.

The training result schema is a new schema developed for ViNNSL 2.0. It gives users the possibility to analyse neural network trainings. The *identifier*, *data*, *executionEnvironment*, *propagationType*, *learningType* and *networkType* elements are already explained in this section.

creationDate: Date of the training result.
netinput: Input data for the neural network training.
netoutput: Expected output data of the neural network.
weightmatrix: Link to a file containing the connection weights.
epochs: Number of training epochs.
activationfunction: It defines the used activation function for the neural network
totalexecutiontime: The total execution time of the training in seconds.
epocherrorvalue: Link to a file containing the error value per epoch.
learningrate: The specified learning rate for neural network training.
momentum: The specified momentum for neural network training.
threshold: The specified threshold for neural network training.
bias: The bias value of the neural network. Every bias value defined in the *definition* schema is a single element in the *training result* schema.

The *instance* schema is another schema, which is introduced with ViNNSL 2.0. Beside the structure of the neural network, all elements are extensions to ViNNSL or to

this schema. The elements within the *instance* schema were already described before. Together with a *data* schema, it represents a neural network execution. The *instance* schema provides all necessary information to empower the system to build a neural network. The *data* schema contains the data, which will be processed by the network.

The *result* schema is slightly modified in ViNNSL 2.0. The *data* and *file* element from ViNNSL are grouped together to the *data* element. The *vinnslinstance* element was renamed to *instanceSchemaID* and contains the ID of the used *instance* schema. The *messages* element was skipped because ViNNSL 2.0. provides a *training result* schema.

4.2.1. File definitions

This subsection provides a short explanation on the syntax of files used in ViNNSL 2.0.

4.2.1.1. Weightmatrix file

The weightmatrix file uses the following syntax to represent values:

- source layer ID ";" source neuron row ";" source neuron column ";" target layer ID ";" target neuron row ";" target neuron column ";" weight value

Example: Input1;2;1;Hidden1;1;1;0.321

The single values have a semicolon as delimiter. It is used in order to respect comma and dot as decimal point.

4.2.1.2. Data file

The data file provides an equivalent presentation like the table element in the ViNNSL schemas. The used syntax is:

- Input values of a dataset in decimal form using whitespace as delimiter
- Output values of a dataset in decimal form using whitespace as delimiter

This means for training purposes the odd row numbers provide the input values and the following even row number the target output values. In case of execution only input values are provided by the data file. If the data file is part of the result schema, odd row numbers are the input values and even row numbers are the calculated output values.

4.3. Schema models

This section provides all schema models (figures 4.2 - 4.7) in order to give an overview of the content of every schema which is provided below. Newly added elements and models are shown with red lines. Elements with black lines were already proposed in the previous version of ViNNSL.

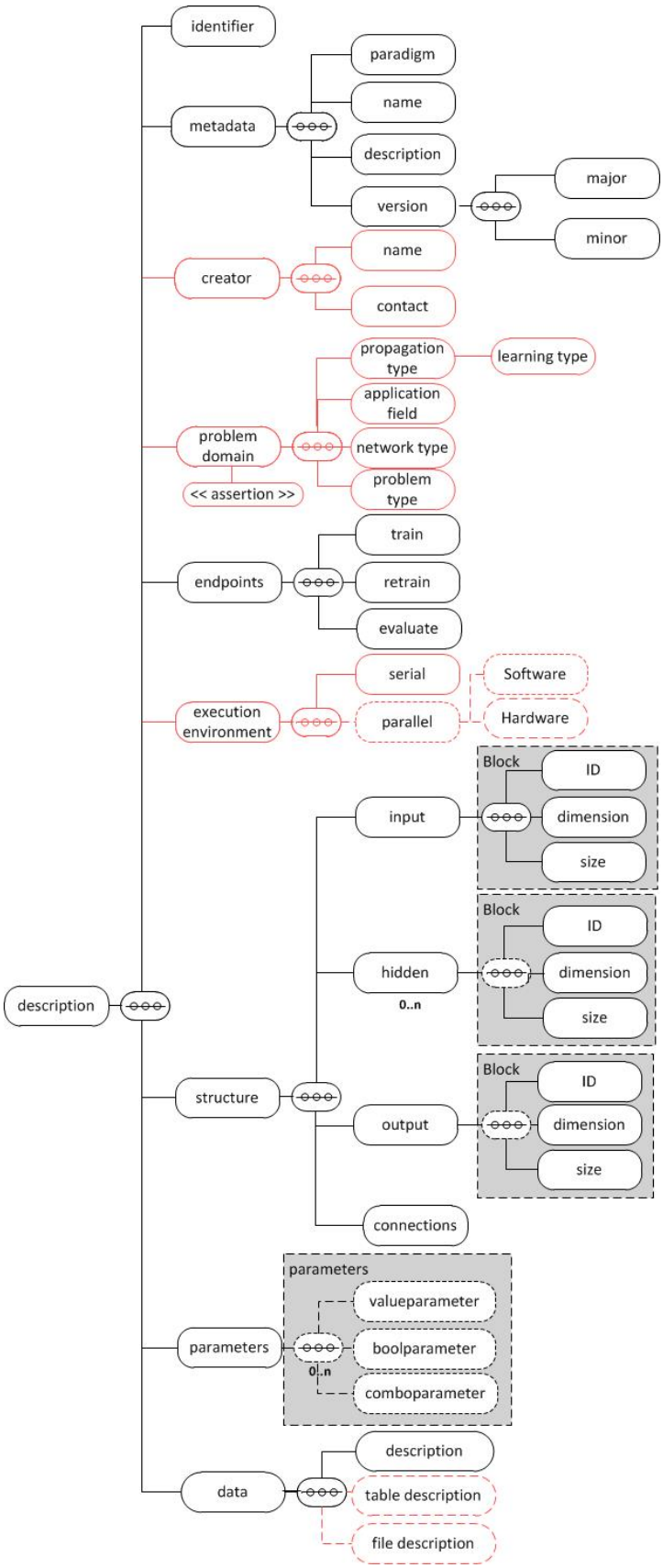

Figure 4.2.: Structure and elements of the description schema

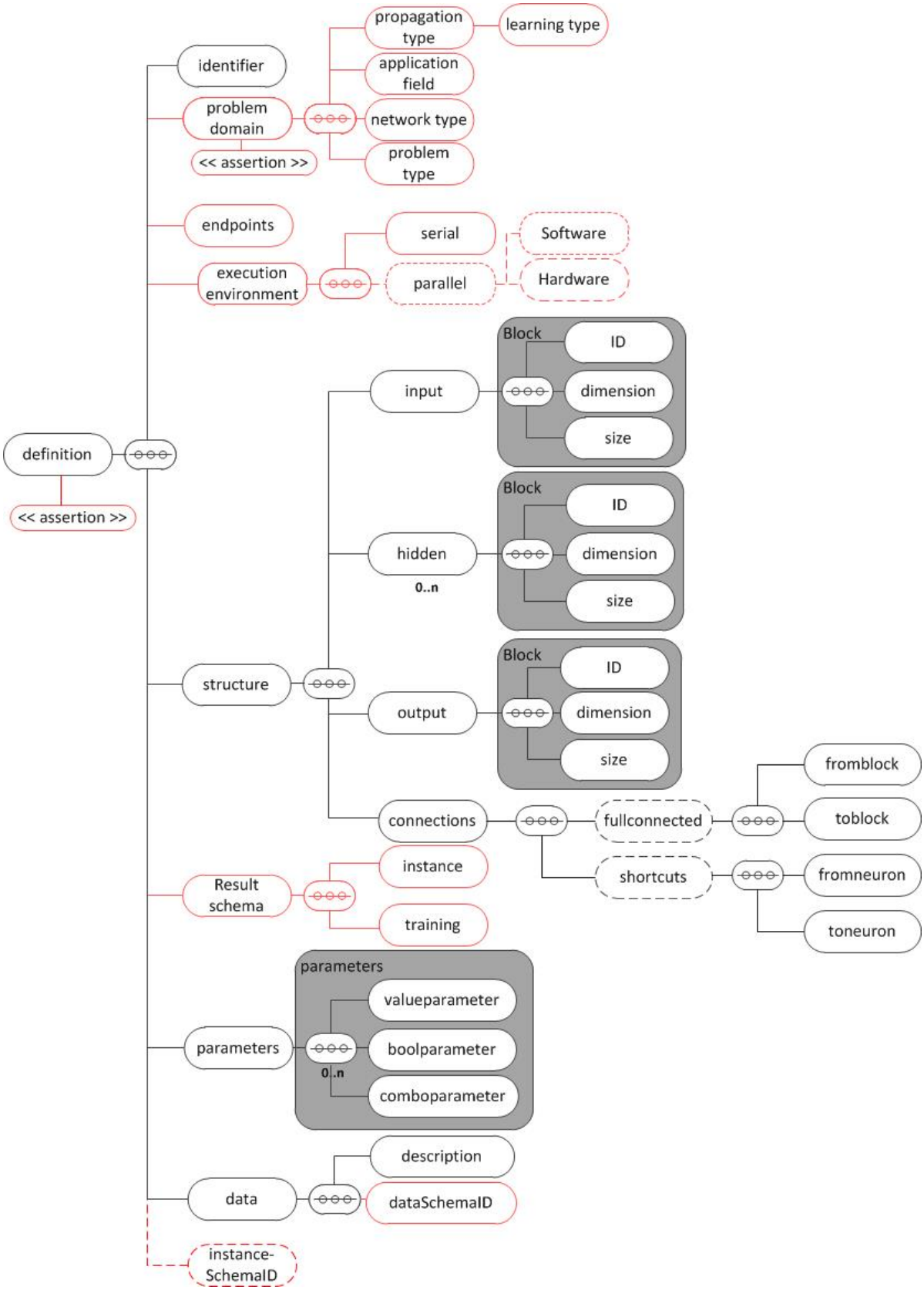

Figure 4.3.: Structure and elements of the definition schema

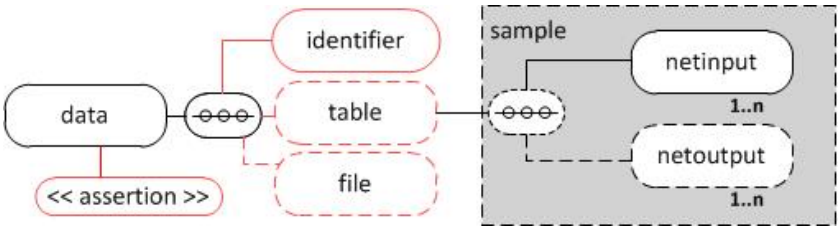

Figure 4.4.: Structure and elements of the data schema

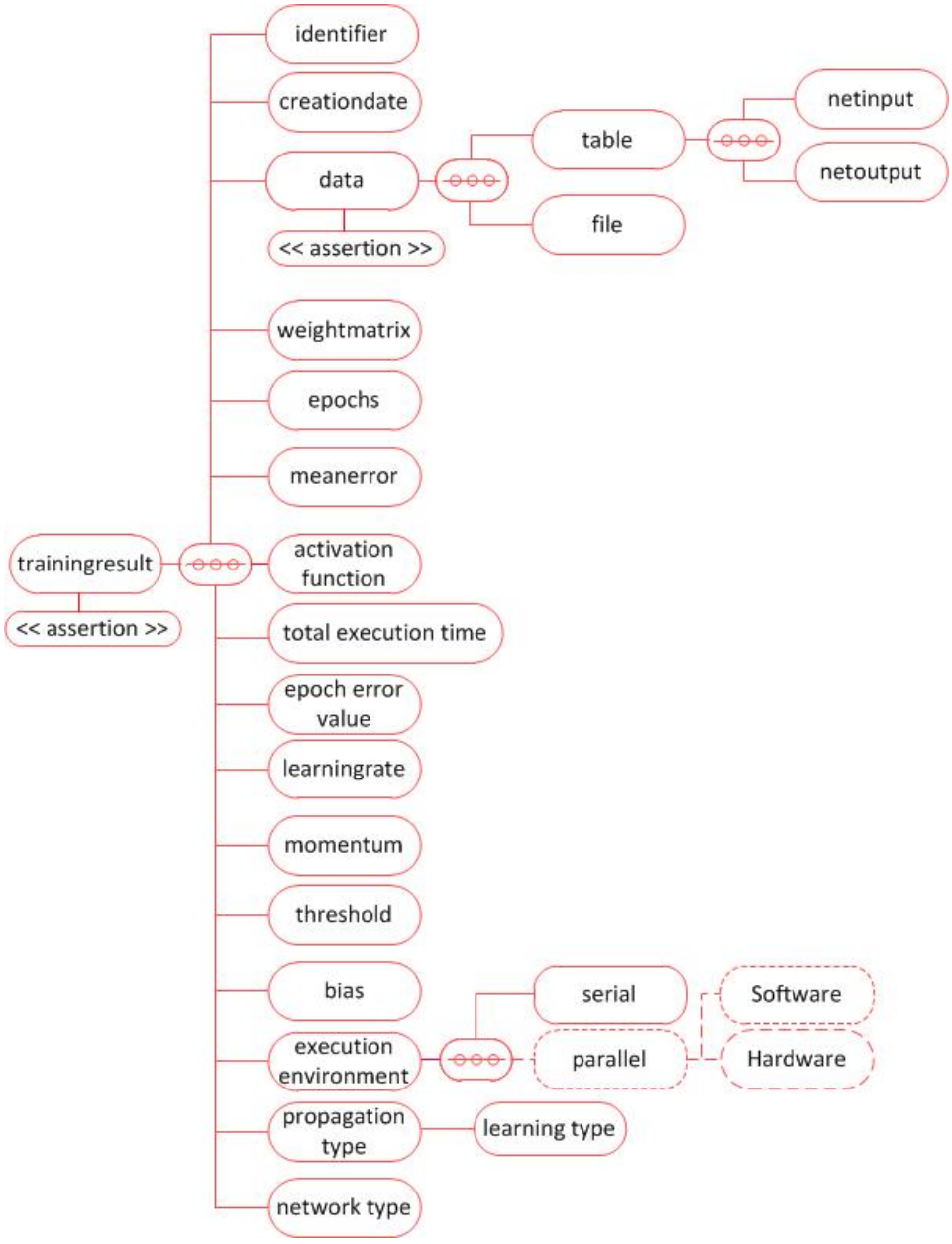

Figure 4.5.: Structure and elements of the training result schema

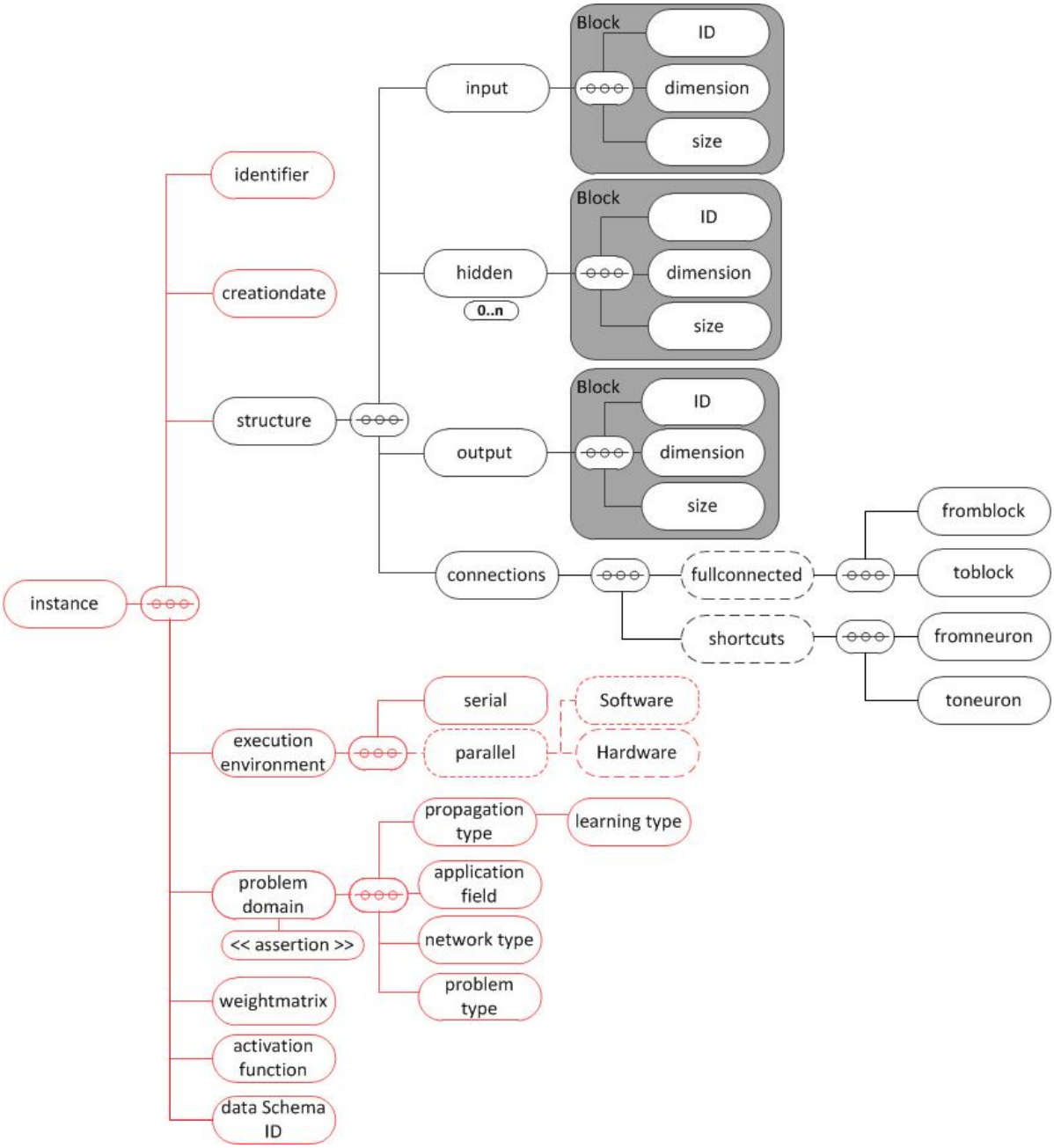

Figure 4.6.: Structure and elements of the instance schema

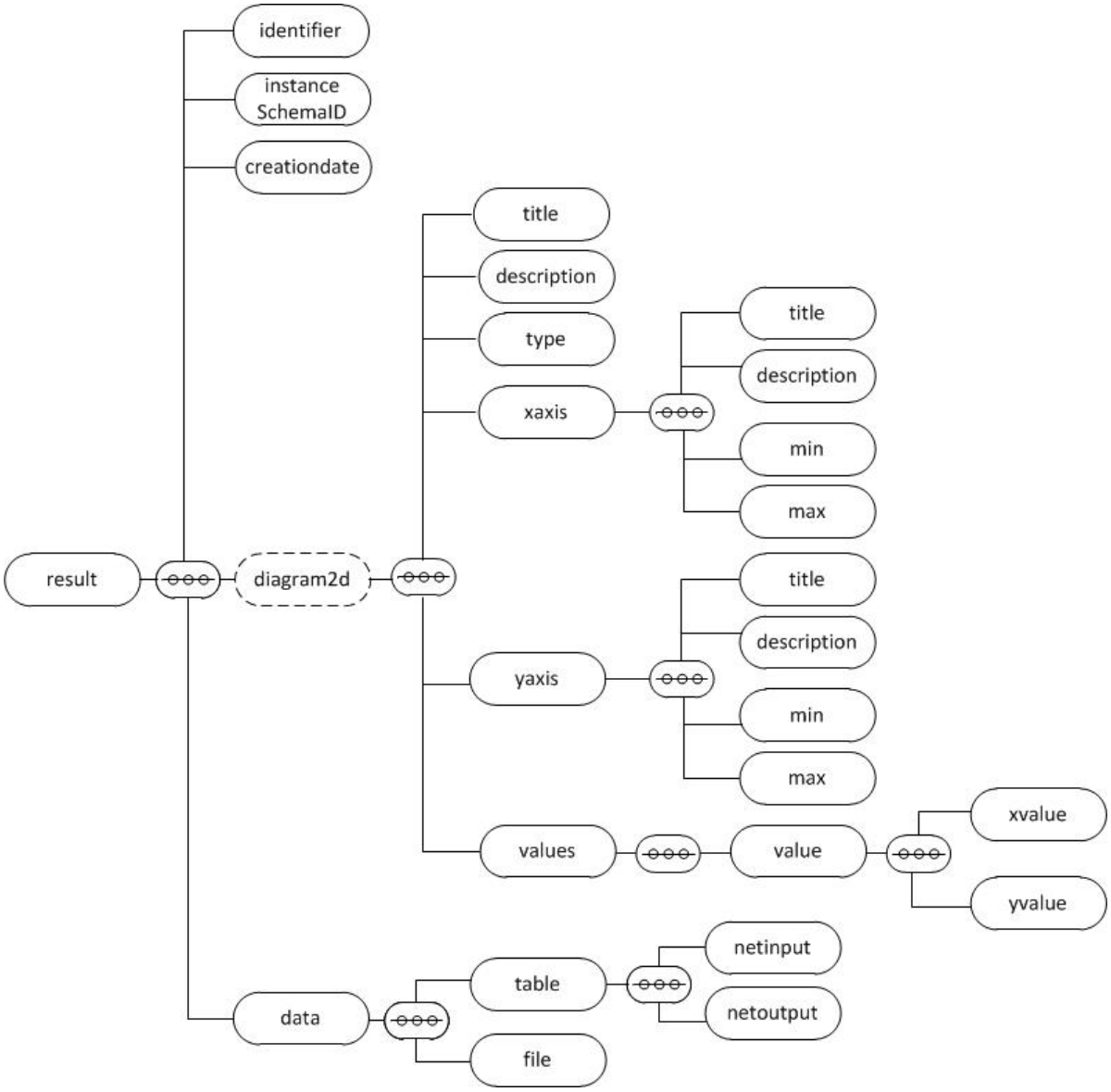

Figure 4.7.: Structure and elements of the result schema

4.4. Description Schema

The description schema is used in order to describe a neural network and its possibilities. Figure 4.2 shows the structure and the elements of the schema.

The root element specifies the required XSD version. In addition, it specifies a simple type, which provides the propagation method of neural networks, another one for the minimum value 1 and a complex type with different parameter values for specifying the hardware for parallel execution. The *description* element sequence is described within the subsections.

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
  vc:minVersion="1.1" xmlns:vc="http://www.w3.org/2007/XMLSchema-versioning">

  <xs:simpleType name="propa">
    <xs:restriction base="xs:string">
      <xs:enumeration value="feedforward"/>
      <xs:enumeration value="feedback"/>
      <xs:enumeration value="recurrent"/>
    </xs:restriction>
  </xs:simpleType>

  <xs:simpleType name="minno1">
    <xs:restriction base="xs:integer">
      <xs:minExclusive value="0"/>
    </xs:restriction>
  </xs:simpleType>

  <xs:complexType name="parametervalue">
    <xs:choice minOccurs="0" maxOccurs="unbounded">
      <xs:element name="valueparameter">
        <xs:complexType>
          <xs:simpleContent>
            <xs:extension base="xs:decimal">
              <xs:attribute name="name" type="xs:string"/>
            </xs:extension>
          </xs:simpleContent>
        </xs:complexType>
      </xs:element>
      <xs:element name="boolparameter">
        <xs:complexType>
          <xs:simpleContent>
            <xs:extension base="xs:boolean">
              <xs:attribute name="name" type="xs:string"/>
            </xs:extension>
          </xs:simpleContent>
        </xs:complexType>
      </xs:element>
      <xs:element name="comboparameter">
        <xs:complexType>
          <xs:simpleContent>
```

```
            <xs:extension base="xs:string">
              <xs:attribute name="name" type="xs:string"/>
            </xs:extension>
          </xs:simpleContent>
        </xs:complexType>
      </xs:element>
    </xs:choice>
  </xs:complexType>

  <xs:element name="description">
    <xs:complexType>
      <xs:sequence>
      ...
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

4.4.1. Identifier and Metadata

The *identifier* is an alphanumeric value, which is used as unique identifier for the *description* schema. The *metadata* element provides keywords for the neural network paradigm, the network name given by the creator and a version. The version may have a major and a minor number.

```
<xs:element name="identifier" type="xs:string"/>
<xs:element name="metadata">
  <xs:complexType>
    <xs:sequence>
      <!--paradigm keywords-->
      <xs:element name="paradigm" type="xs:string"/>
      <!--network name given by the creator-->
      <xs:element name="name" type="xs:string"/>
      <xs:element name="description" type="xs:string"/>
      <xs:element name="version">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="major" type="xs:integer"/>
            <xs:element name="minor" type="xs:integer"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

4.4.2. Creator

The *creator* element contains the name of the person developing the neural network and his contact information.

```
<xs:element name="creator">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="name" type="xs:string"/>
      <xs:element name="contact" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

4.4.3. Problem domain

The problem domain is based on the information gathered in chapter 2. It provides the elements to define the propagation type, learning type, application field, network type and problem type.

The possible value for *propagationType* is ”feedforward”, ”feedback” or ”recurrent”. In addition, *learningType* must be defined. Possible values are ”definedconstructed”, ”trained”, ”linear”, ”supervised” and ”unsupervised”. The *applicationField* element states, for which fields of application the neural network is usable. It provides a predefined enumeration of values but also accepts a custom field of application. Whereas the *networkType* element specifies, which types of neural network can be used. For example, Jordan-Net and Elman-Net are similar to each other. Therefore, the *description* schema fits to both networks. However, all proposed network types must be in the same class. The *problemType* element specifies the kind of problem the network shall solve e.g. Classification or Optimisation.

The *problemDomain* element provides assertions, which are used to check the consistency of the subelements’ values. For example, a neural network description cannot fit to a Hopfield and a Backpropagation network.

```
<xs:element name="problemDomain">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="propagationType">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="learningType">
              <xs:simpleType>
                <xs:restriction base="xs:string">
                  <xs:enumeration value="definedconstructed"/>
                  <xs:enumeration value="trained"/>
                  <xs:enumeration value="supervised"/>
                  <xs:enumeration value="unsupervised"/>
                  <xs:enumeration value="linear"/>
                </xs:restriction>
              </xs:simpleType>
            </xs:element>
          </xs:sequence>
          <xs:attribute name="type" type="propa"/>
        </xs:complexType>
      </xs:element>
      <xs:element name="applicationField" maxOccurs="unbounded">
        <xs:simpleType>
          <xs:union>
            <xs:simpleType>
              <xs:restriction base="xs:string">
                <xs:enumeration value="AccFin"/>
                <xs:enumeration value="HealthMed"/>
                <xs:enumeration value="Marketing"/>
                <xs:enumeration value="Retail"/>
                <xs:enumeration value="Insur"/>
                <xs:enumeration value="Telecom"/>
                <xs:enumeration value="Operations"/>
                <xs:enumeration value="EMS"/>
              </xs:restriction>
            </xs:simpleType>
            <xs:simpleType>
              <xs:restriction base="xs:string">
                <xs:pattern value="[A-Za-z]*"></xs:pattern>
              </xs:restriction>
            </xs:simpleType>
          </xs:union>
        </xs:simpleType>
      </xs:element>
      <xs:element name="networkType">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:enumeration value="Hopfield"/>
            <xs:enumeration value="CNN"/>
            <xs:enumeration value="ART"/>
            <xs:enumeration value="Backpropagation"/>
```

```
                    <xs:enumeration value="Cascade-Correlation"/>
                    <xs:enumeration value="Kohonen"/>
                    <xs:enumeration value="Counterpropagation"/>
                    <xs:enumeration value="Perceptron"/>
                    <xs:enumeration value="Linear-Associator"/>
                    <xs:enumeration value="Jordan-Net"/>
                    <xs:enumeration value="Elman-Net"/>
                  </xs:restriction>
                </xs:simpleType>
              </xs:element>
              <xs:element name="problemType">
                <xs:simpleType>
                  <xs:restriction base="xs:string">
                    <xs:enumeration value="Classifiers"/>
                    <xs:enumeration value="Approximators"/>
                    <xs:enumeration value="Memory"/>
                    <xs:enumeration value="Optimisation"/>
                    <xs:enumeration value="Clustering"/>
                  </xs:restriction>
                </xs:simpleType>
              </xs:element>
            </xs:sequence>
            <xs:assert
              test="((propagationType/@type = 'feedback' and propagationType/learningType = 'definedconstructed' and
(networkType = 'Hopfield' or networkType = 'CNN')) or (propagationType/@type = 'feedback' and propagationType/
learningType = 'trained' and networkType = 'ART') or (propagationType/@type = 'feedforward' and propagationType/
learningType = 'supervised' and (networkType = 'Backpropagation' or networkType = 'Cascade-Correlation')) or
(propagationType/@type = 'feedforward' and propagationType/learningType = 'unsupervised' and (networkType =
'Counterpropagation' or networkType = 'Kohonen')) or (propagationType/@type = 'feedforward' and propagationType/
learningType = 'linear' and (networkType = 'Linear-Associator' or networkType = 'Perceptron')) or (propagationType/
@type = 'recurrent' and propagationType/learningType = 'supervised' and (networkType = 'Jordan-Net' or networkType =
'Elman-Net')) and count(networkType) = 1 and count(propagationType/learningType) = 1 and count(propagationType/
@type) = 1)"/>
          </xs:complexType>
        </xs:element>
```

4.4.4. Endpoints

Within the *description* schema the *endpoints* specify if the neural network can be trained, retrained or evaluated.

```
<xs:element name="endpoints">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="train" type="xs:boolean"/>
      <xs:element name="retrain" type="xs:boolean"/>
      <xs:element name="evaluate" type="xs:boolean"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

4.4.5. Execution Environment

The *executionEnvironment* element defines if the neural network execution is serial and/or parallel. The serial execution is always provided. In case of parallel execution the whole parallelism taxonomy has to be provided. In addition, ViNNSL 2.0 provides the possibility to specify multiple execution environments. Every environment has to be defined by its own *executionEnvironment* element. The parallel execution is divided into software and hardware information. The *software* and the *hardware* element contain all information, which is necessary to set up and train the neural network automatically. The hardware is defined using *valueparameter*, *boolparameter* and *comboparameter*. The software specification has to be mapped to the *software* element's structure. An example is provided in chapter 5.

```
<xs:element name="executionEnvironment" maxOccurs="unbounded">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="serial" type="xs:boolean" fixed="true"/>
      <xs:element name="parallel" minOccurs="0">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="software">
              <xs:complexType>
                <xs:choice>
                  <xs:element name="control">
                    <xs:complexType>
                      <xs:sequence>
                        <xs:element name="transputer">
                          <xs:complexType>
                            <xs:simpleContent>
                              <xs:extension base="xs:string">
                                <xs:attribute name="version" type="xs:string" use="required"/>
                              </xs:extension>
                            </xs:simpleContent>
                          </xs:complexType>
                        </xs:element>
```

```
        </xs:sequence>
      </xs:complexType>
    </xs:element>
    <xs:element name="data">
      <xs:complexType>
        <xs:choice>
          <xs:element name="topological">
            <xs:complexType>
              <xs:choice>
                <xs:element name="pipelining">
                  <xs:complexType>
                    <xs:sequence>
                      <xs:element name="systolicarr">
                        <xs:complexType>
                          <xs:simpleContent>
                            <xs:extension base="xs:string">
                              <xs:attribute name="version" type="xs:string" use="required"/>
                            </xs:extension>
                          </xs:simpleContent>
                        </xs:complexType>
                      </xs:element>
                    </xs:sequence>
                  </xs:complexType>
                </xs:element>
                <xs:element name="coarsestruct">
                  <xs:complexType>
                    <xs:choice>
                      <xs:element name="connmachine">
                        <xs:complexType>
                          <xs:simpleContent>
                            <xs:extension base="xs:string">
                              <xs:attribute name="version" type="xs:string" use="required"/>
                            </xs:extension>
                          </xs:simpleContent>
                        </xs:complexType>
                      </xs:element>
                      <xs:element name="maspar">
                        <xs:complexType>
                          <xs:simpleContent>
                            <xs:extension base="xs:string">
                              <xs:attribute name="version" type="xs:string" use="required"/>
                            </xs:extension>
                          </xs:simpleContent>
                        </xs:complexType>
                      </xs:element>
                    </xs:choice>
                  </xs:complexType>
                </xs:element>
```

```
        <xs:element name="finestruct">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="connmachine">
                <xs:complexType>
                  <xs:simpleContent>
                    <xs:extension base="xs:string">
                      <xs:attribute name="version" type="xs:string" use="required"/>
                    </xs:extension>
                  </xs:simpleContent>
                </xs:complexType>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:choice>
    </xs:complexType>
  </xs:element>
  <xs:element name="structural">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="spmd">
          <xs:complexType>
            <xs:choice>
              <xs:element name="hypercube">
                <xs:complexType>
                  <xs:simpleContent>
                    <xs:extension base="xs:string">
                      <xs:attribute name="version" type="xs:string" use="required"/>
                    </xs:extension>
                  </xs:simpleContent>
                </xs:complexType>
              </xs:element>
              <xs:element name="cluster">
                <xs:complexType>
                  <xs:simpleContent>
                    <xs:extension base="xs:string">
                      <xs:attribute name="version" type="xs:string" use="required"/>
                    </xs:extension>
                  </xs:simpleContent>
                </xs:complexType>
              </xs:element>
              <xs:element name="gpgpu">
                <xs:complexType>
                  <xs:simpleContent>
                    <xs:extension base="xs:string">
                      <xs:attribute name="version" type="xs:string" use="required"/>
```

```
                                        </xs:extension>
                                    </xs:simpleContent>
                                </xs:complexType>
                            </xs:element>
                            <xs:element name="multicore">
                                <xs:complexType>
                                    <xs:simpleContent>
                                        <xs:extension base="xs:string">
                                            <xs:attribute name="version" type="xs:string" use="required"/>
                                        </xs:extension>
                                    </xs:simpleContent>
                                </xs:complexType>
                            </xs:element>
                        </xs:choice>
                    </xs:complexType>
                </xs:element>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
</xs:choice>
</xs:complexType>
</xs:element>
</xs:choice>
</xs:complexType>
</xs:element>
<xs:element name="hardware">
    <xs:complexType>
        <xs:choice>
            <xs:element name="general" type="parametervalue"/>
            <xs:element name="special" type="parametervalue"/>
        </xs:choice>
    </xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>
```

4.4.6. Structure

The *description* schema has to contain a *structure* element. It provides the information for input, hidden and output layers.

```
<xs:element name="structure">
  <xs:complexType>
    <xs:sequence>
              ...
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

4.4.6.1. Input (Layer)

The *input* element has to occur once. It contains an *ID* of type string, which is used to identify the input layer. In addition, it provides a *dimension* and a *size* field with a *min* and a *max* value as elements. *Dimension* sets the number of neurons inside the layer (rows) and *size* sets the number of layers of the specific type (columns). These values can be specified in order to consider the available resource for future network executions.

```
<xs:element name="input">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="ID" type="xs:string"/>
      <xs:element name="dimension">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="min" type="minno1"/>
            <xs:element name="max" type="minno1"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="size">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="min" type="minno1"/>
            <xs:element name="max" type="minno1"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

4.4.6.2. Hidden (Layer)

The *hidden* element is optional in the *description* schema. It contains an *ID* of type string, which is used to identify the hidden layer(s). It also provides a *dimension* and a *size* field with a *min* and a *max* value as elements. *Dimension* sets the number of neurons inside the layer (rows) and *size* sets the number of layers of the specific type (columns). These values can be specified in order to consider the available resource for future network executions.

```
<xs:element name="hidden" minOccurs="0" maxOccurs="unbounded">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="ID" type="xs:string"/>
      <xs:element name="dimension">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="min" type="minno1"/>
            <xs:element name="max" type="minno1"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="size">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="min" type="minno1"/>
            <xs:element name="max" type="minno1"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

4.4.6.3. Output (Layer)

The *output* element is optional too, but may occur only once. It contains an *ID* of type string as identifier. It provides a *dimension* and a *size* field. *Dimension* sets the number of neurons inside the layer (rows) and *size* sets the number of layers of the specific type (columns). These values can be specified in order to consider the available resource for future network executions.

```
<xs:element name="output" minOccurs="0">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="ID" type="xs:string"/>
      <xs:element name="dimension">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="min" type="minno1"/>
            <xs:element name="max" type="minno1"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="size">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="min" type="minno1"/>
            <xs:element name="max" type="minno1"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

4.4.6.4. Connections

Connections specifies the possible connection types between the neurons of the neural network.

```
<xs:element name="output" minOccurs="0">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="ID" type="xs:string"/>
      <xs:element name="dimension">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="min" type="minno1"/>
            <xs:element name="max" type="minno1"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="size">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="min" type="minno1"/>
            <xs:element name="max" type="minno1"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

4.4.7. Parameters

The *parameter* element contains several parameters. These can be used to specify, which values e.g. learning rate, bias value, momentum, threshold, activation function and others, can be provided for the neural network instantiation. The concrete values have to be given in the *definition* schema.

```
<xs:element name="parameters">
  <xs:complexType>
    <xs:choice minOccurs="0" maxOccurs="unbounded">
      <xs:element name="valueparameter" type="xs:string" maxOccurs="unbounded"/>
      <xs:element name="boolparameter" type="xs:string" maxOccurs="unbounded"/>
      <xs:element name="comboparameter" type="xs:string" maxOccurs="unbounded"/>
    </xs:choice>
  </xs:complexType>
</xs:element>
```

4.4.8. Data

The *data* element contains three elements, which are used to describe how data has to be presented to the network. The *description* element is mandatory. The *tabledescription* and *filedescription* elements can be used to give a more detailed explanation for data presentation.

As fact that ViNNSL provides a separate *data* schema the data itself need not be specified in the *description* schema. Especially, as every *definition* schema has to provide a corresponding *data* schema. Therefore, the *data* element provides a description how the data has to be presented to the network only.

```
<xs:element name="data">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="description" type="xs:string"/>
      <xs:element name="tabledescription" type="xs:string" minOccurs="0"/>
      <xs:element name="filedescription" type="xs:string" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

4.5. Definition Schema

The *definition* schema is used to describe a neural network, which shall be trained or executed. The root element specifies the required XSD version. In addition, it specifies simple types, which provide the propagation method of neural networks, another for the minimum value 1 and a complex type with different parameter values for specifying the hardware for parallel execution. If the *endpoint* value in the *definition* schema is set to "retrain" the user has to provide an *instance* schema ID. The *definition* element sequence is described within the subsections.

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
  vc:minVersion="1.1" xmlns:vc="http://www.w3.org/2007/XMLSchema-versioning">

  <xs:simpleType name="propa">
    <xs:restriction base="xs:string">
      <xs:enumeration value="feedforward"/>
      <xs:enumeration value="feedback"/>
      <xs:enumeration value="recurrent"/>
    </xs:restriction>
  </xs:simpleType>

  <xs:simpleType name="minno1">
    <xs:restriction base="xs:integer">
      <xs:minExclusive value="0"/>
    </xs:restriction>
  </xs:simpleType>
```

```
<xs:complexType name="parametervalue">
  <xs:choice minOccurs="0" maxOccurs="unbounded">
    <xs:element name="valueparameter">
      <xs:complexType>
        <xs:simpleContent>
          <xs:extension base="xs:decimal">
            <xs:attribute name="name" type="xs:string"/>
          </xs:extension>
        </xs:simpleContent>
      </xs:complexType>
    </xs:element>
    <xs:element name="boolparameter">
      <xs:complexType>
        <xs:simpleContent>
          <xs:extension base="xs:boolean">
            <xs:attribute name="name" type="xs:string"/>
          </xs:extension>
        </xs:simpleContent>
      </xs:complexType>
    </xs:element>
    <xs:element name="comboparameter">
      <xs:complexType>
        <xs:simpleContent>
          <xs:extension base="xs:string">
            <xs:attribute name="name" type="xs:string"/>
          </xs:extension>
        </xs:simpleContent>
      </xs:complexType>
    </xs:element>
  </xs:choice>
</xs:complexType>

<xs:element name="definition">
  <xs:complexType>
    <xs:sequence>
    ...
    </xs:sequence>
    <xs:assert
      test="if(endpoints = 'retrain') then
              if(count(instanceSchemaID) > 0) then
                true()
              else
                false()
            else
              if(count(instanceSchemaID) > 0) then
                false()
              else true()"/>
  </xs:complexType>
</xs:element>
</xs:schema>
```

4.5.1. Identifier

The *identifier* is a unique value to identify the *definition* instance. In order to assign the *definition* schema to a *description* schema instance the *identifier* will have the *description* schema identifier as prefix.

```
<xs:element name="identifier" type="xs:string"/>
```

4.5.2. Problem domain

The *problemDomain* element in the *definition* schema is similar to the equivalent element in the *description* schema. However, within the *definition* schema the elements *propagationType*, *learningType*, *networkType* and *problemtype* are allowed to occur once only.

The possible value for *propagationType* is "feedforward", "feedback" or "recurrent". In addition, *learningType* must be defined. Possible values are "definedconstructed", "trained", "linear", "supervised" and "unsupervised". The *applicationField* element states, for which fields of application the neural network is usable. It provides a predefined enumeration of values but also accepts a custom field of application. Whereas the *networkType* element specifies which types of neural network can be used. For example, Jordan-Net and Elman-Net are similar to each other. Therefore, the *description* schema fits to both networks. However, all proposed network types must be in the same class. The *problemType* element specifies the kind of problem the network shall solve e.g. Classification or Optimisation.

The *problemDomain* element provides assertions, which are used to check the consistency of the subelements' values. For example, a neural network description cannot fit to a Hopfield and a Backpropagation network.

```
<xs:element name="problemDomain">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="propagationType">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="learningType">
              <xs:simpleType>
                <xs:restriction base="xs:string">
                  <xs:enumeration value="definedconstructed"/>
                  <xs:enumeration value="trained"/>
                  <xs:enumeration value="supervised"/>
                  <xs:enumeration value="unsupervised"/>
                  <xs:enumeration value="linear"/>
                </xs:restriction>
              </xs:simpleType>
            </xs:element>
          </xs:sequence>
          <xs:attribute name="type" type="propa"/>
        </xs:complexType>
      </xs:element>
      <xs:element name="applicationField">
        <xs:simpleType>
          <xs:union>
            <xs:simpleType>
              <xs:restriction base="xs:string">
                <xs:enumeration value="AccFin"/>
                <xs:enumeration value="HealthMed"/>
                <xs:enumeration value="Marketing"/>
                <xs:enumeration value="Retail"/>
```

```
                    <xs:enumeration value="Insur"/>
                    <xs:enumeration value="Telecom"/>
                    <xs:enumeration value="Operations"/>
                    <xs:enumeration value="EMS"/>
                  </xs:restriction>
                </xs:simpleType>
                <xs:simpleType>
                  <xs:restriction base="xs:string">
                    <xs:pattern value="[A-Za-z]*"></xs:pattern>
                  </xs:restriction>
                </xs:simpleType>
              </xs:union>
            </xs:simpleType>
          </xs:element>
          <xs:element name="networkType">
            <xs:simpleType>
              <xs:restriction base="xs:string">
                <xs:enumeration value="Hopfield"/>
                <xs:enumeration value="CNN"/>
                <xs:enumeration value="ART"/>
                <xs:enumeration value="Backpropagation"/>
                <xs:enumeration value="Cascade-Correlation"/>
                <xs:enumeration value="Kohonen"/>
                <xs:enumeration value="Counterpropagation"/>
                <xs:enumeration value="Perceptron"/>
                <xs:enumeration value="Linear-Associator"/>
                <xs:enumeration value="Jordan-Net"/>
                <xs:enumeration value="Elman-Net"/>
              </xs:restriction>
            </xs:simpleType>
          </xs:element>
          <xs:element name="problemType">
            <xs:simpleType>
              <xs:restriction base="xs:string">
                <xs:enumeration value="Classifiers"/>
                <xs:enumeration value="Approximators"/>
                <xs:enumeration value="Memory"/>
                <xs:enumeration value="Optimisation"/>
                <xs:enumeration value="Clustering"/>
              </xs:restriction>
            </xs:simpleType>
          </xs:element>
        </xs:sequence>
        <xs:assert
          test="((propagationType/@type = 'feedback' and propagationType/learningType = 'definedconstructed' and
(networkType = 'Hopfield' or networkType = 'CNN')) or (propagationType/@type = 'feedback' and propagationType/
learningType = 'trained' and networkType = 'ART') or (propagationType/@type = 'feedforward' and propagationType/
learningType = 'supervised' and (networkType = 'Backpropagation' or networkType = 'Cascade-Correlation')) or
(propagationType/@type = 'feedforward' and propagationType/learningType = 'unsupervised' and (networkType =
'Counterpropagation' or networkType = 'Kohonen')) or (propagationType/@type = 'feedforward' and propagationType/
learningType = 'linear' and (networkType = 'Linear-Associator' or networkType = 'Perceptron')) or (propagationType/
@type = 'recurrent' and propagationType/learningType = 'supervised' and (networkType = 'Jordan-Net' or networkType =
'Elman-Net')) and count(networkType) = 1 and count(propagationType/learningType) = 1 and count(propagationType/
@type) = 1)"/>
      </xs:complexType>
    </xs:element>
```

4.5.3. Endpoints

In the *definition* schema *endpoints* has to specify if the defined neural network will be trained or retrained. For retraining, the *instance* schema has to be provided, and the specification set in the *definition* and the *instance* schema have to match. Therefore, the usage of the identifier as prefix is recommended.

```
<xs:element name="endpoints">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:enumeration value="train"/>
      <xs:enumeration value="retrain"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

4.5.4. Execution environment

The *executionEnvironment* element defines if the neural network execution is serial and/or parallel. Serial execution is always provided. In case of parallel execution the whole parallelisation taxonomy has to be provided. Parallel execution is divided into software and hardware information. The *software* and the *hardware* element contain all information, which is necessary to set up and train the neural network automatically. The hardware is defined using *valueparameter*, *boolparameter* and *comboparameter*. The software specification has to be mapped to the *software* element's structure. An example is provided in chapter 5.

```
<xs:element name="executionEnvironment">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="serial" type="xs:boolean" fixed="true"/>
      <xs:element name="parallel" minOccurs="0">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="software">
              <xs:complexType>
                <xs:choice>
                  <xs:element name="control">
                    <xs:complexType>
                      <xs:sequence>
                        <xs:element name="transputer">
                          <xs:complexType>
                            <xs:simpleContent>
                              <xs:extension base="xs:string">
                                <xs:attribute name="version" type="xs:string" use="required"/>
                              </xs:extension>
                            </xs:simpleContent>
                          </xs:complexType>
                        </xs:element>
                      </xs:sequence>
                    </xs:complexType>
                  </xs:element>
```

```
<xs:element name="data">
  <xs:complexType>
    <xs:choice>
      <xs:element name="topological">
        <xs:complexType>
          <xs:choice>
            <xs:element name="pipelining">
              <xs:complexType>
                <xs:sequence>
                  <xs:element name="systolicarr">
                    <xs:complexType>
                      <xs:simpleContent>
                        <xs:extension base="xs:string">
                          <xs:attribute name="version" type="xs:string" use="required"/>
                        </xs:extension>
                      </xs:simpleContent>
                    </xs:complexType>
                  </xs:element>
                </xs:sequence>
              </xs:complexType>
            </xs:element>
            <xs:element name="coarsestruct">
              <xs:complexType>
                <xs:choice>
                  <xs:element name="connmachine">
                    <xs:complexType>
                      <xs:simpleContent>
                        <xs:extension base="xs:string">
                          <xs:attribute name="version" type="xs:string" use="required"/>
                        </xs:extension>
                      </xs:simpleContent>
                    </xs:complexType>
                  </xs:element>
                  <xs:element name="maspar">
                    <xs:complexType>
                      <xs:simpleContent>
                        <xs:extension base="xs:string">
                          <xs:attribute name="version" type="xs:string" use="required"/>
                        </xs:extension>
                      </xs:simpleContent>
                    </xs:complexType>
                  </xs:element>
                </xs:choice>
              </xs:complexType>
            </xs:element>
            <xs:element name="finestruct">
              <xs:complexType>
                <xs:sequence>
                  <xs:element name="connmachine">
                    <xs:complexType>
                      <xs:simpleContent>
                        <xs:extension base="xs:string">
                          <xs:attribute name="version" type="xs:string" use="required"/>
                        </xs:extension>
                      </xs:simpleContent>
                    </xs:complexType>
                  </xs:element>
                </xs:sequence>
              </xs:complexType>
            </xs:element>
          </xs:choice>
        </xs:complexType>
      </xs:element>
      <xs:element name="structural">
        <xs:complexType>
          <xs:sequence>
```

```
                                        <xs:element name="spmd">
                                          <xs:complexType>
                                            <xs:choice>
                                              <xs:element name="hypercube">
                                                <xs:complexType>
                                                  <xs:simpleContent>
                                                    <xs:extension base="xs:string">
                                                      <xs:attribute name="version" type="xs:string" use="required"/>
                                                    </xs:extension>
                                                  </xs:simpleContent>
                                                </xs:complexType>
                                              </xs:element>
                                              <xs:element name="cluster">
                                                <xs:complexType>
                                                  <xs:simpleContent>
                                                    <xs:extension base="xs:string">
                                                      <xs:attribute name="version" type="xs:string" use="required"/>
                                                    </xs:extension>
                                                  </xs:simpleContent>
                                                </xs:complexType>
                                              </xs:element>
                                              <xs:element name="gpgpu">
                                                <xs:complexType>
                                                  <xs:simpleContent>
                                                    <xs:extension base="xs:string">
                                                      <xs:attribute name="version" type="xs:string" use="required"/>
                                                    </xs:extension>
                                                  </xs:simpleContent>
                                                </xs:complexType>
                                              </xs:element>
                                              <xs:element name="multicore">
                                                <xs:complexType>
                                                  <xs:simpleContent>
                                                    <xs:extension base="xs:string">
                                                      <xs:attribute name="version" type="xs:string" use="required"/>
                                                    </xs:extension>
                                                  </xs:simpleContent>
                                                </xs:complexType>
                                              </xs:element>
                                            </xs:choice>
                                          </xs:complexType>
                                        </xs:element>
                                      </xs:sequence>
                                    </xs:complexType>
                                  </xs:element>
                                </xs:choice>
                              </xs:complexType>
                            </xs:element>
                          </xs:choice>
                        </xs:complexType>
                      </xs:element>
                      <xs:element name="hardware">
                        <xs:complexType>
                          <xs:choice>
                            <xs:element name="general" type="parametervalue"/>
                            <xs:element name="special" type="parametervalue"/>
                          </xs:choice>
                        </xs:complexType>
                      </xs:element>
                    </xs:sequence>
                  </xs:complexType>
                </xs:element>
              </xs:sequence>
            </xs:complexType>
          </xs:element>
```

4.5.5. Structure

The *definition* schema has to contain a *structure* element. It provides the information for input, hidden and output layers.

```
<xs:element name="structure">
  <xs:complexType>
    <xs:sequence>
      ...
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

4.5.5.1. Input (Layer)

The *input* element has to occur once. It contains an *ID* of type string, which is used to identify the input layer. In addition, it provides a *dimension* and a *size* field. *Dimension* sets the number of neurons inside the layer (rows) and *size* sets the number of layers of the specific type (columns).

```
<xs:element name="input">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="ID" type="xs:string"/>
      <xs:element name="dimension" type="minno1"/>
      <xs:element name="size" type=" minno1"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

4.5.5.2. Hidden (Layer)

The *hidden* element is optional in the *definition* schema. It contains an *ID* of type string, which is used to identify the hidden layers. It also provides a *dimension* and a *size* field. *Dimension* sets the number of neurons inside the layer (rows) and *size* sets the number of layers of the specific type (columns).

```
<xs:element name="hidden" minOccurs="0" maxOccurs="unbounded">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="ID" type="xs:string"/>
      <xs:element name="dimension" type="minno1"/>
      <xs:element name="size" type="minno1"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

4.5.5.3. Output (Layer)

The *output* element is optional too, but may occur only once. It contains an *ID* of type string as identifier. It provides a *dimension* and a *size* field. *Dimension* sets the number of neurons inside the layer (rows) and *size* sets the number of layers of the specific type (columns).

```
<xs:element name="output" minOccurs="0">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="ID" type="xs:string"/>
      <xs:element name="dimension" type="minno1"/>
      <xs:element name="size" type="minno1"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

4.5.5.4. Connections

Depending on the *description* schema the *definition* schema can specify connections between neurons. The *fullconnected* element has to specify pairs of *fromblock* and *toblock* elements. The values of these elements are the layer IDs. Concerning better understandability the layer IDs are defined as string data type. For example IDs "Input1", "Hidden1", etc. are possible. Every element can only contain one ID. In case shortcuts are used, connections between neurons across layers can be specified.

The connection between single neurons has to be specified using the following order: layer ID, neuron row position and neuron column position. The delimiter has to be ";".

```
<xs:element name="connections">
  <xs:complexType>
    <xs:choice maxOccurs="unbounded">
      <xs:element name="fullconnected" minOccurs="0">
        <xs:complexType>
          <xs:sequence maxOccurs="unbounded">
            <xs:element name="fromblock" type="xs:string"/>
            <xs:element name="toblock" type="xs:string"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="shortcuts">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="fromneuron" type="xs:string"/>
            <xs:element name="toneuron" type="xs:string"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:choice>
  </xs:complexType>
</xs:element>
```

4.5.6. Result schema

The *resultSchema* element contains two elements: *instance* and *training*. These values define the outcome after processing the neural network. If *instance* is set to true the user receives an *instance* schema, whereas *training* delivers a *training result* schema. However, these values can be set only if the *endpoint* value is "train" or "retrain".

```
<xs:element name="resultSchema">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="instance" type="xs:boolean"/>
      <xs:element name="training" type="xs:boolean"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

4.5.7. Parameters

The *parameter* element contains several parameters of type decimal (*valueparameter*), boolean (*boolparameter*) and string (*comboparameter*). These can be used to specify values like learning rate, bias value, momentum, threshold, activation function and others. Every parameter contains an attribute having its name, which was specified in the *description* schema.

```
<xs:element name="parameters" type="parametervalue"/>
```

4.5.8. Data

The *data* element contains a description of the data and the assigned data schema ID.

```
<xs:element name="data">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="description" type="xs:string"/>
      <xs:element name="dataSchemaID" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

4.5.9. Instance Schema ID

If an existing neural network shall be retrained the *definition* schema has to provide the *instance* schema ID. The schemas soundness' has to be checked by the application. *Definition* and *instance* schema have to have an identical structure element as well as propagation type, learning type and network type. The other elements, which both schemas have in common, will have the *definition* schema's values.

```
<xs:element name="instanceSchemaID" type="xs:string" minOccurs="0"/>
```

4.6. Data Schema

The *data* schema is used to provide the values for a neural network, which shall be trained or executed. The root element specifies the required XSD version and the name of the schema.

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
  vc:minVersion="1.1" xmlns:vc="http://www.w3.org/2007/XMLSchema-versioning">
  <xs:element name="dataschema">
    <xs:complexType>
      <xs:sequence>
                ...
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

4.6.1. Identifier and Creationdate

The *data* schema contains an *identifier* element, which is a unique string containing numeric and alphanumeric values. *Creationdate* specifies the date on which the instance was created.

```
<xs:element name="identifier" type="xs:string"/>
<xs:element name="creationdate" type="xs:date"/>
```

4.6.2. Data element

The *data* element consists of a table element and a file element. This element has to contain at least a table or a file element.

```
<xs:element name="data">
  <xs:complexType>
    <xs:sequence>
      ...
    </xs:sequence>
    <xs:assert test="(count(table) + count(file)) > 0"/>
  </xs:complexType>
</xs:element>
```

4.6.2.1. Table element

The *table* element stores all values in table form. It contains a *netinput* and a *netoutput* element, which are used for training purposes. Each pair of these elements represents one dataset. In case the neural network is evaluated the *netoutput* will be skipped and input values are provided only. Within the elements single values are separated using XML whitespace. All values are allowed to have decimal numbers only.

```
<xs:element name="table" minOccurs="0">
  <xs:complexType  >
    <xs:choice>
      <xs:sequence maxOccurs="unbounded">
        <xs:element name="netinput">
          <xs:simpleType>
            <xs:restriction base="xs:string">
            <xs:pattern value="(([0-9])*([.])?([0-9])*(\s)?)+"/>
            <xs:whiteSpace value="preserve"/>
            </xs:restriction>
          </xs:simpleType>
        </xs:element>
        <xs:element name="netoutput">
          <xs:simpleType>
            <xs:restriction base="xs:string">
            <xs:pattern value="(([0-9])*([.])?([0-9])*(\s)?)+"/>
            <xs:whiteSpace value="preserve"/>
            </xs:restriction>
          </xs:simpleType>
        </xs:element>
      </xs:sequence>
      <xs:sequence>
        <xs:element name="input" maxOccurs="unbounded">
          <xs:simpleType>
            <xs:restriction base="xs:string">
            <xs:pattern value="(([0-9])*([.])?([0-9])*(\s)?)+"/>
            <xs:whiteSpace value="preserve"/>
            </xs:restriction>
          </xs:simpleType>
        </xs:element>
      </xs:sequence>
    </xs:choice>
  </xs:complexType>
</xs:element>
```

4.6.2.2. File element

The *file* element stores the link to the file, which contains the data for the neural network.

```
<xs:element name="file" type="xs:string" minOccurs="0"/>
```

4.7. Training Result Schema

The *training result* schema contains all values, which are explored within the training phase of a neural network. It shall provide information on the training results and determine if the trained network is satisfactory for the user.

The root element specifies the required XSD version and the name of the schema. In addition, it provides a simple type to specify the propagation type and complex types for parameter values and bias values.

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
  vc:minVersion="1.1" xmlns:vc="http://www.w3.org/2007/XMLSchema-versioning">

  <xs:simpleType name="propa">
    <xs:restriction base="xs:string">
      <xs:enumeration value="feedforward"/>
      <xs:enumeration value="feedback"/>
      <xs:enumeration value="recurrent"/>
    </xs:restriction>
  </xs:simpleType>

  <xs:complexType name="parametervalue">
    <xs:choice minOccurs="0" maxOccurs="unbounded">
      <xs:element name="valueparameter">
        <xs:complexType>
          <xs:simpleContent>
            <xs:extension base="xs:decimal">
              <xs:attribute name="name" type="xs:string"/>
            </xs:extension>
          </xs:simpleContent>
        </xs:complexType>
      </xs:element>
      <xs:element name="boolparameter">
        <xs:complexType>
          <xs:simpleContent>
            <xs:extension base="xs:boolean">
              <xs:attribute name="name" type="xs:string"/>
            </xs:extension>
          </xs:simpleContent>
        </xs:complexType>
      </xs:element>
      <xs:element name="comboparameter">
        <xs:complexType>
          <xs:simpleContent>
            <xs:extension base="xs:string">
              <xs:attribute name="name" type="xs:string"/>
            </xs:extension>
          </xs:simpleContent>
        </xs:complexType>
      </xs:element>
    </xs:choice>
  </xs:complexType>

  <xs:complexType name="parametervalueBias">
    <xs:choice minOccurs="0" maxOccurs="unbounded">
      <xs:element name="valueparameter">
        <xs:complexType>
          <xs:simpleContent>
            <xs:extension base="xs:decimal">
              <xs:attribute name="name" type="xs:string"/>
            </xs:extension>
          </xs:simpleContent>
        </xs:complexType>
      </xs:element>
    </xs:choice>
  </xs:complexType>
```

```
<xs:element name="trainingresultschema">
  <xs:complexType>
    <xs:sequence>
    …
    </xs:sequence>
    <xs:assert
     test="((propagationType/@type = 'feedback' and propagationType/learningType = 'definedconstructed' and
(networkType = 'Hopfield' or networkType = 'CNN')) or (propagationType/@type = 'feedback' and propagationType/
learningType = 'trained' and networkType = 'ART') or (propagationType/@type = 'feedforward' and propagationType/
learningType = 'supervised' and (networkType = 'Backpropagation' or networkType = 'Cascade-Correlation')) or
(propagationType/@type = 'feedforward' and propagationType/learningType = 'unsupervised' and (networkType =
'Counterpropagation' or networkType = 'Kohonen')) or (propagationType/@type = 'feedforward' and propagationType/
learningType = 'linear' and (networkType = 'Linear-Associator' or networkType = 'Perceptron')) or (propagationType/
@type = 'recurrent' and propagationType/learningType = 'supervised' and (networkType = 'Jordan-Net' or networkType =
'Elman-Net')) and count(networkType) = 1 and count(propagationType/learningType) = 1 and count(propagationType/
@type) = 1)"/>
  </xs:complexType>
</xs:element>
</xs:schema>
```

4.7.1. Identifier and Creationdate

The *training result* schema contains an *identifier*, which is a unique string containing numeric and alphanumeric values. *CreationDate* specifies the date on which the instance was created.

```
<xs:element name="identifier" type="xs:string"/>
<xs:element name="creationdate" type="xs:date"/>
```

4.7.2. Data element

The *data* element consists of a *table* and a *file* element. It has to contain at least one of the subelements.

```
<xs:element name="data">
  <xs:complexType>
    <xs:sequence>
      ...
    </xs:sequence>
    <xs:assert test="(count(table) + count(file)) > 0"/>
  </xs:complexType>
</xs:element>
```

4.7.2.1. Table element

The *table* element stores all values in table form. It contains a *netinput* and a *netoutput* element, which are used for training purposes. Each pair of these elements represents one dataset. Within the elements single values are separated using XML whitespace. All values are allowed to have decimal numbers only.

```
<xs:element name="table" minOccurs="0">
  <xs:complexType>
    <xs:sequence  maxOccurs="unbounded">
      <xs:element name="netinput" >
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:pattern value="(([0-9])*([.])?([0-9])*(\s)?)+"/>
            <xs:whiteSpace value="preserve"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:element>
      <xs:element name="netoutput">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:pattern value="(([0-9])*([.])?([0-9])*(\s)?)+"/>
            <xs:whiteSpace value="preserve"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

4.7.2.2. File element

The *file* element stores the link to the file which contains the data for the neural network.

```
<xs:element name="file" type="xs:string" minOccurs="0"/>
```

4.7.3. Result parameter

The *weightmatrix* element specifies the link to a file containing all weights between the neurons. *Epochs* gives the number of epochs during training. *Meanerror* is given as decimal number. *Activationfunction* names the activation function only. *Totalexecutiontime* is given in seconds. The *epocherrorvalue* element is similar to the *weightmatrix* element. It provides the link to a file containing the error value for every epoch. *Learningrate*, *momentum* and *threshold* are decimal numbers. The *bias* element is optional. Depending on the definition schema the number of *bias* elements can vary.

```
<xs:element name="weightmatrix" type="xs:string"/>
<xs:element name="epochs" type="xs:integer"/>
<xs:element name="meanerror" type="xs:decimal"/>
<xs:element name="activationfunction" type="xs:string"/>
<xs:element name="totalexecutiontime" type="xs:decimal"/> <!--always in seconds-->
<xs:element name="epocherrorvalue" type="xs:string"/>
<xs:element name="learningrate" type="xs:decimal"/>
<xs:element name="momentum" type="xs:decimal"/>
<xs:element name="threshold" type="xs:decimal"/>
<xs:element name="bias" type="xs:decimal" minOccurs="0" maxOccurs="unbounded"/>
```

4.7.4. Execution environment

This element provides the execution environment used for training. It defines if the neural network execution is serial and/or parallel. The serial execution is always provided. In case of parallel execution the whole parallelism taxonomy has to be provided. The parallel execution is divided into software and hardware information. The *software* and the *hardware* element contain all information, which is necessary to set up and train the neural network automatically. The hardware is defined using *valueparameter*, *boolparameter* and *comboparameter*. The software specification has to be mapped to the *software* element's structure. An example is provided in chapter 5.

```
<xs:element name="executionEnvironment">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="serial" type="xs:boolean" fixed="true"/>
      <xs:element name="parallel" minOccurs="0">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="software">
              <xs:complexType>
                <xs:choice>
                  <xs:element name="control">
                    <xs:complexType>
                      <xs:sequence>
                        <xs:element name="transputer">
                          <xs:complexType>
                            <xs:simpleContent>
                              <xs:extension base="xs:string">
                                <xs:attribute name="version" type="xs:string" use="required"/>
                              </xs:extension>
                            </xs:simpleContent>
                          </xs:complexType>
                        </xs:element>
                      </xs:sequence>
                    </xs:complexType>
                  </xs:element>
                  <xs:element name="data">
                    <xs:complexType>
                      <xs:choice>
                        <xs:element name="topological">
                          <xs:complexType>
                            <xs:choice>
                              <xs:element name="pipelining">
                                <xs:complexType>
                                  <xs:sequence>
                                    <xs:element name="systolicarr">
                                      <xs:complexType>
                                        <xs:simpleContent>
                                          <xs:extension base="xs:string">
                                            <xs:attribute name="version" type="xs:string" use="required"/>
                                          </xs:extension>
                                        </xs:simpleContent>
                                      </xs:complexType>
                                    </xs:element>
                                  </xs:sequence>
                                </xs:complexType>
                              </xs:element>
                              <xs:element name="coarsestruct">
                                <xs:complexType>
                                  <xs:choice>
                                    <xs:element name="connmachine">
                                      <xs:complexType>
                                        <xs:simpleContent>
                                          <xs:extension base="xs:string">
                                            <xs:attribute name="version" type="xs:string" use="required"/>
                                          </xs:extension>
                                        </xs:simpleContent>
                                      </xs:complexType>
                                    </xs:element>
```

```
                <xs:element name="maspar">
                  <xs:complexType>
                    <xs:simpleContent>
                      <xs:extension base="xs:string">
                        <xs:attribute name="version" type="xs:string" use="required"/>
                      </xs:extension>
                    </xs:simpleContent>
                  </xs:complexType>
                </xs:element>
              </xs:choice>
            </xs:complexType>
          </xs:element>
          <xs:element name="finestruct">
            <xs:complexType>
              <xs:sequence>
                <xs:element name="connmachine">
                  <xs:complexType>
                    <xs:simpleContent>
                      <xs:extension base="xs:string">
                        <xs:attribute name="version" type="xs:string" use="required"/>
                      </xs:extension>
                    </xs:simpleContent>
                  </xs:complexType>
                </xs:element>
              </xs:sequence>
            </xs:complexType>
          </xs:element>
        </xs:choice>
      </xs:complexType>
    </xs:element>
    <xs:element name="structural">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="spmd">
            <xs:complexType>
              <xs:choice>
                <xs:element name="hypercube">
                  <xs:complexType>
                    <xs:simpleContent>
                      <xs:extension base="xs:string">
                        <xs:attribute name="version" type="xs:string" use="required"/>
                      </xs:extension>
                    </xs:simpleContent>
                  </xs:complexType>
                </xs:element>
                <xs:element name="cluster">
                  <xs:complexType>
                    <xs:simpleContent>
                      <xs:extension base="xs:string">
                        <xs:attribute name="version" type="xs:string" use="required"/>
                      </xs:extension>
                    </xs:simpleContent>
                  </xs:complexType>
                </xs:element>
```

```
                                        <xs:element name="gpgpu">
                                            <xs:complexType>
                                                <xs:simpleContent>
                                                    <xs:extension base="xs:string">
                                                        <xs:attribute name="version" type="xs:string" use="required"/>
                                                    </xs:extension>
                                                </xs:simpleContent>
                                            </xs:complexType>
                                        </xs:element>
                                        <xs:element name="multicore">
                                            <xs:complexType>
                                                <xs:simpleContent>
                                                    <xs:extension base="xs:string">
                                                        <xs:attribute name="version" type="xs:string" use="required"/>
                                                    </xs:extension>
                                                </xs:simpleContent>
                                            </xs:complexType>
                                        </xs:element>
                                    </xs:choice>
                                </xs:complexType>
                            </xs:element>
                        </xs:sequence>
                    </xs:complexType>
                </xs:element>
            </xs:choice>
        </xs:complexType>
    </xs:element>
    <xs:element name="hardware">
        <xs:complexType>
            <xs:choice>
                <xs:element name="general" type="parametervalue"/>
                <xs:element name="special" type="parametervalue"/>
            </xs:choice>
        </xs:complexType>
    </xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>
```

4.7.5. Propagation and Learning type

This element contains the specification of the trained neural network. The possible value for the *type* attribute is "feedforward", "feedback" or "recurrent". In addition, the *learningtype* has to be given. Possible values are "definedconstructed", "trained", "linear", "supervised" and "unsupervised".

```
<xs:element name="propagationType">
  <xs:complexType>
    <xs:sequence>
    <xs:element name="learningType">
      <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:enumeration value="definedconstructed"/>
        <xs:enumeration value="trained"/>
        <xs:enumeration value="supervised"/>
        <xs:enumeration value="unsupervised"/>
        <xs:enumeration value="linear"/>
      </xs:restriction>
      </xs:simpleType>
    </xs:element>
    </xs:sequence>
    <xs:attribute name="type" type="propa"/>
  </xs:complexType>
</xs:element>
```

4.7.6. Network type

The *networkType* element defines which neural network was trained.

```
<xs:element name="networkType">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:enumeration value="Hopfield"/>
      <xs:enumeration value="CNN"/>
      <xs:enumeration value="ART"/>
      <xs:enumeration value="Backpropagation"/>
      <xs:enumeration value="Cascade-Correlation"/>
      <xs:enumeration value="Kohonen"/>
      <xs:enumeration value="Counterpropagation"/>
      <xs:enumeration value="Perceptron"/>
      <xs:enumeration value="Linear-Associator"/>
      <xs:enumeration value="Jordan-Net"/>
      <xs:enumeration value="Elman-Net"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

4.8. Instance Schema

The *instance* schema provides all parameter for a trained and executable neural network. Furthermore, it can be also used in order to retrain it. In this case the process results in

a new *instance* schema and if requested a *training result* schema. For a neural network execution users have to provide the *instance* schema and a *data* schema, which contains the input values.

The root element specifies the required XSD version and the name of the schema. In addition it provides a simple type to specify the propagation type as well as a complex type for parameter values.

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
  vc:minVersion="1.1" xmlns:vc="http://www.w3.org/2007/XMLSchema-versioning">

  <xs:simpleType name="propa">
    <xs:restriction base="xs:string">
      <xs:enumeration value="feedforward"/>
      <xs:enumeration value="feedback"/>
      <xs:enumeration value="recurrent"/>
    </xs:restriction>
  </xs:simpleType>

  <xs:complexType name="parametervalue">
    <xs:choice minOccurs="0" maxOccurs="unbounded">
      <xs:element name="valueparameter">
        <xs:complexType>
          <xs:simpleContent>
            <xs:extension base="xs:decimal">
              <xs:attribute name="name" type="xs:string"/>
            </xs:extension>
          </xs:simpleContent>
        </xs:complexType>
      </xs:element>
      <xs:element name="boolparameter">
        <xs:complexType>
          <xs:simpleContent>
            <xs:extension base="xs:boolean">
              <xs:attribute name="name" type="xs:string"/>
            </xs:extension>
          </xs:simpleContent>
        </xs:complexType>
      </xs:element>
      <xs:element name="comboparameter">
        <xs:complexType>
          <xs:simpleContent>
            <xs:extension base="xs:string">
              <xs:attribute name="name" type="xs:string"/>
            </xs:extension>
          </xs:simpleContent>
        </xs:complexType>
      </xs:element>
    </xs:choice>
  </xs:complexType>

  <xs:element name="instanceschema">
    <xs:complexType>
      <xs:sequence>
                  ...
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

4.8.1. Identifier and Creationdate

The *instance* schema contains an *identifier*, which is a unique string containing numeric and alphanumeric values. *Creationdate* specifies the date on which the instance was created.

```
<xs:element name="identifier" type="xs:string"/>
<xs:element name="creationdate" type="xs:date"/>
```

4.8.2. Structure

The *instance* schema has to contain a *structure* element. It provides the information for input, hidden and output layers.

```
<xs:element name="structure">
  <xs:complexType>
    <xs:sequence>
      ...
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

4.8.2.1. Input (Layer)

The *input* element has to occur once. It contains an *ID* of type string, which is used to identify the input layer. In addition, it provides a *dimension* and a *size* field. *Dimension* sets the number of neurons inside the layer (rows) and *size* sets the number of layers of the specific type (columns).

```
<xs:element name="input">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="ID" type="xs:string"/>
      <xs:element name="dimension" type="xs:integer"/>
      <xs:element name="size" type="xs:integer"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

4.8.2.2. Hidden (Layer)

The *hidden* element is optional in the *instance* schema. It contains an *ID* of type string, which is used to identify the hidden layers. It also provides a *dimension* and a *size* field. *Dimension* sets the number of neurons inside the layer (rows) and *size* sets the number of layers of the specific type (columns).

```
<xs:element name="hidden" minOccurs="0" maxOccurs="unbounded">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="ID" type="xs:string"/>
      <xs:element name="dimension" type="minno1"/>
      <xs:element name="size" type="minno1"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

4.8.2.3. Output (Layer)

The *output* element is optional too, but may occur only once. It contains an *ID* of type string as identifier. It provides a *dimension* and a *size* field. *Dimension* sets the number of neurons inside the layer (rows) and *size* sets the number of layers of the specific type (columns).

```
<xs:element name="output" minOccurs="0">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="ID" type="xs:string"/>
      <xs:element name="dimension" type="minno1"/>
      <xs:element name="size" type="minno1"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

4.8.2.4. Connections

The connections set in the *instance* schema are the same as given in the *definition* schema. The *fullconnected* element has to specify pairs of *fromblock* and *toblock* elements. The values of these elements are the layer IDs. Concerning better understandability the layer IDs are defined as string data type. For example, IDs ”Input1”, ”Hidden1”, etc. are possible. Every element can only contain one ID. In case *shortcuts* are used, connections between neurons across layers can be specified.

The connection between single neurons has to be specified using the following order: layer ID, neuron row position and neuron column position. The delimiter has to be ”;”.

```
<xs:element name="connections">
  <xs:complexType>
    <xs:choice maxOccurs="unbounded">
      <xs:element name="fullconnected" minOccurs="0">
        <xs:complexType>
          <xs:sequence maxOccurs="unbounded">
            <xs:element name="fromblock" type="xs:string"/>
            <xs:element name="toblock" type="xs:string"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="shortcuts" minOccurs="0">
        <xs:complexType>
          <xs:sequence maxOccurs="unbounded">
            <xs:element name="fromneuron" type="xs:string"/>
            <xs:element name="toneuron" type="xs:string"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:choice>
  </xs:complexType>
</xs:element>
```

4.8.3. Execution environment

The *executionEnvironment* element defines if the neural network execution is serial and/or parallel. The serial execution is always provided. In case of parallel execution the whole parallelisation taxonomy has to be provided. The parallel execution is divided into software and hardware information. The *software* and the *hardware* element contain all information, which is necessary to set up and train the neural network automatically. The hardware is defined using *valueparameter*, *boolparameter* and *comboparameter*. The software specification has to be mapped to the *software* element's structure. An example is provided in chapter 5.

```
<xs:element name="executionEnvironment">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="serial" type="xs:boolean" fixed="true"/>
      <xs:element name="parallel" minOccurs="0">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="software">
              <xs:complexType>
                <xs:choice>
                  <xs:element name="control">
                    <xs:complexType>
                      <xs:sequence>
```

```
            <xs:element name="transputer">
              <xs:complexType>
                <xs:simpleContent>
                  <xs:extension base="xs:string">
                    <xs:attribute name="version" type="xs:string" use="required"/>
                  </xs:extension>
                </xs:simpleContent>
              </xs:complexType>
            </xs:element>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="data">
        <xs:complexType>
          <xs:choice>
            <xs:element name="topological">
              <xs:complexType>
                <xs:choice>
                  <xs:element name="pipelining">
                    <xs:complexType>
                      <xs:sequence>
                        <xs:element name="systolicarr">
                          <xs:complexType>
                            <xs:simpleContent>
                              <xs:extension base="xs:string">
                                <xs:attribute name="version" type="xs:string" use="required"/>
                              </xs:extension>
                            </xs:simpleContent>
                          </xs:complexType>
                        </xs:element>
                      </xs:sequence>
                    </xs:complexType>
                  </xs:element>
                  <xs:element name="coarsestruct">
                    <xs:complexType>
                      <xs:choice>
                        <xs:element name="connmachine">
                          <xs:complexType>
                            <xs:simpleContent>
                              <xs:extension base="xs:string">
                                <xs:attribute name="version" type="xs:string" use="required"/>
                              </xs:extension>
                            </xs:simpleContent>
                          </xs:complexType>
                        </xs:element>
                        <xs:element name="maspar">
                          <xs:complexType>
                            <xs:simpleContent>
                              <xs:extension base="xs:string">
                                <xs:attribute name="version" type="xs:string" use="required"/>
                              </xs:extension>
                            </xs:simpleContent>
                          </xs:complexType>
                        </xs:element>
                      </xs:choice>
                    </xs:complexType>
                  </xs:element>
```

```
        <xs:element name="finestruct">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="connmachine">
                <xs:complexType>
                  <xs:simpleContent>
                    <xs:extension base="xs:string">
                      <xs:attribute name="version" type="xs:string" use="required"/>
                    </xs:extension>
                  </xs:simpleContent>
                </xs:complexType>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:choice>
    </xs:complexType>
  </xs:element>
  <xs:element name="structural">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="spmd">
          <xs:complexType>
            <xs:choice>
              <xs:element name="hypercube">
                <xs:complexType>
                  <xs:simpleContent>
                    <xs:extension base="xs:string">
                      <xs:attribute name="version" type="xs:string" use="required"/>
                    </xs:extension>
                  </xs:simpleContent>
                </xs:complexType>
              </xs:element>
              <xs:element name="cluster">
                <xs:complexType>
                  <xs:simpleContent>
                    <xs:extension base="xs:string">
                      <xs:attribute name="version" type="xs:string" use="required"/>
                    </xs:extension>
                  </xs:simpleContent>
                </xs:complexType>
              </xs:element>
              <xs:element name="gpgpu">
                <xs:complexType>
                  <xs:simpleContent>
                    <xs:extension base="xs:string">
                      <xs:attribute name="version" type="xs:string" use="required"/>
                    </xs:extension>
                  </xs:simpleContent>
                </xs:complexType>
              </xs:element>
```

```
                                        <xs:element name="multicore">
                                          <xs:complexType>
                                            <xs:simpleContent>
                                              <xs:extension base="xs:string">
                                                <xs:attribute name="version" type="xs:string" use="required"/>
                                              </xs:extension>
                                            </xs:simpleContent>
                                          </xs:complexType>
                                        </xs:element>
                                      </xs:choice>
                                    </xs:complexType>
                                  </xs:element>
                                </xs:sequence>
                              </xs:complexType>
                            </xs:element>
                          </xs:choice>
                        </xs:complexType>
                      </xs:element>
                    </xs:choice>
                  </xs:complexType>
                </xs:element>
                <xs:element name="hardware">
                  <xs:complexType>
                    <xs:choice>
                      <xs:element name="general" type="parametervalue"/>
                      <xs:element name="special" type="parametervalue"/>
                    </xs:choice>
                  </xs:complexType>
                </xs:element>
              </xs:sequence>
            </xs:complexType>
          </xs:element>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
```

4.8.4. Problem domain

The *problemDomain* element in the *instance* schema is similar to the equivalent element in the *description* schema. However, within the *instance* schema the elements *propagationType*, *learningType*, *networkType* and *problemtype* are allowed to occur once only.

The possible value for *propagationType* is "feedforward", "feedback" or "recurrent". In addition, *learningType* must be defined. Possible values are "definedconstructed", "trained", "linear", "supervised" and "unsupervised". The *applicationField* element states, for which fields of application the neural network is usable. It provides a predefined enumeration of values but also accepts a custom field of application. Whereas the *networkType* element specifies which types of neural network can be used. For example, Jordan-Net and Elman-Net are similar to each other. Therefore, the *instance* schema has to contain only one of them. The *problemType* element specifies the kind of problem the network shall solve e.g. Classification or Optimisation.

The *problemDomain* element provides assertions, which are used to check the consistency of the subelements' values. For example, a neural network instance cannot fit to a Hopfield and a Backpropagation network.

```
<xs:element name="problemDomain">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="networkType">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:enumeration value="Hopfield"/>
            <xs:enumeration value="CNN"/>
            <xs:enumeration value="ART"/>
            <xs:enumeration value="Backpropagation"/>
            <xs:enumeration value="Cascade-Correlation"/>
            <xs:enumeration value="Kohonen"/>
            <xs:enumeration value="Counterpropagation"/>
            <xs:enumeration value="Perceptron"/>
            <xs:enumeration value="Linear-Associator"/>
            <xs:enumeration value="Jordan-Net"/>
            <xs:enumeration value="Elman-Net"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:element>
```

```
      <xs:element name="propagationType">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="learningType">
              <xs:simpleType>
                <xs:restriction base="xs:string">
                  <xs:enumeration value="definedconstructed"/>
                  <xs:enumeration value="trained"/>
                  <xs:enumeration value="supervised"/>
                  <xs:enumeration value="unsupervised"/>
                  <xs:enumeration value="linear"/>
                </xs:restriction>
              </xs:simpleType>
            </xs:element>
          </xs:sequence>
          <xs:attribute name="type" type="propa"/>
        </xs:complexType>
      </xs:element>
      <xs:element name="applicationField">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:enumeration value="AccFin"/>
            <xs:enumeration value="HealthMed"/>
            <xs:enumeration value="Marketing"/>
            <xs:enumeration value="Retail"/>
            <xs:enumeration value="Insur"/>
            <xs:enumeration value="Telecom"/>
            <xs:enumeration value="Operations"/>
            <xs:enumeration value="EMS"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:element>
      <xs:element name="problemType">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:enumeration value="Classifiers"/>
            <xs:enumeration value="Approximators"/>
            <xs:enumeration value="Memory"/>
            <xs:enumeration value="Optimisation"/>
            <xs:enumeration value="Clustering"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

4.8.5. Activation function, weightmatrix and Data schema

The *weightmatrix* element specifies the link to a file containing all weights between the neurons. The *activationfunction* element gives the name of the used activation function. The *dataSchemaID* element contains the ID of the used *data* schema for the neural network evaluation.

```
<xs:element name="weightmatrix" type="xs:string"/>
<xs:element name="activationfunction" type="xs:string"/>
<xs:element name="dataSchemaID" type="xs:string" minOccurs="0"/>
```

4.9. Result Schema

The *result* schema contains the result of an executed neural network. The root element contains the schema type of ViNNSL as attribute and specifies the minimum XSD version. As fact that the diagram is totally optional the assertion is used to check that the result data is at least delivered in table form or as file.

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
  vc:minVersion="1.1" xmlns:vc="http://www.w3.org/2007/XMLSchema-versioning">
  <xs:element name="resultschema">
    <xs:complexType>
      <xs:sequence>
      ...
      </xs:sequence>
        <xs:assert test="(count(table) + count(file)) > 0"/>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

4.9.1. Identifier, Instance and Creationdate

The *result* schema contains an identifier element, which is a unique string containing numeric and alphanumeric values. *Creationdate* specifies the date on which the instance was created. The *instanceSchemaID* element contains the unique identification of the used instance.

```
<xs:element name="identifier" type="xs:string"/>
<xs:element name="instanceSchemaID" type="xs:string"/>
<xs:element name="creationdate" type="xs:date"/>
```

4.9.2. 2D Diagram

The *result* schema provides the possibility to store a 2D diagram, which presents the results.

```
<xs:element name="diagram2d" minOccurs="0">
  <xs:complexType>
    <xs:sequence>
      ...
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

4.9.2.1. Diagram parameter

The diagram contains a *title*, a *description* and a *type*, which can be defined by the neural network creators.

```
<xs:element name="title" type="xs:string"/>
<xs:element name="description" type="xs:string"/>
<xs:element name="type" type="xs:string"/>
```

4.9.2.2. X-Axis

The *xaxis* element provides the parameters for the x-axis of the diagram. These are the *tile*, a *description*, the *minimum* and the *maximum* value of the axis.

```
<xs:element name="xaxis">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="title" type="xs:string"/>
      <xs:element name="description" type="xs:string"/>
      <xs:element name="min" type="xs:integer"/>
      <xs:element name="max" type="xs:integer"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

4.9.2.3. Y-Axis

The *yaxis* element provides the parameters for the y-axis of the diagram. These are the *tile*, a *description*, the *minimum* and the *maximum* value of the axis.

```
<xs:element name="yaxis">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="title" type="xs:string"/>
      <xs:element name="description" type="xs:string"/>
      <xs:element name="min" type="xs:integer"/>
      <xs:element name="max" type="xs:integer"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

4.9.2.4. Values

The *values* element contains the values for the diagram. Every value is defined by the *value* element, which contains the value for the x-axis and the y-axis.

```
<xs:element name="values">
  <xs:complexType>
    <xs:sequence maxOccurs="unbounded">
      <xs:element name="value">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="xvalue" type="xs:decimal"/>
            <xs:element name="yvalue" type="xs:decimal"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

4.9.3. Table element

The *table* element stores all values in table form. It contains an *input* and an *output* element. The input values were provided by the *data* schema and *output* contains the calculated output values. Within the elements single values are separated using XML whitespace. All values are allowed to have decimal numbers only.

```
<xs:element name="table" minOccurs="0">
  <xs:complexType>
    <xs:sequence minOccurs="0" maxOccurs="unbounded">
      <xs:element name="input">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:pattern value="(([0-9])*([.])?([0-9])*(\s)?)+"/>
            <xs:whiteSpace value="preserve"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:element>
      <xs:element name="output">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:pattern value="(([0-9])*([.])?([0-9])*(\s)?)+"/>
            <xs:whiteSpace value="preserve"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

4.9.4. File element

The *file* element stores the link to the file, which contains the result data of the executed neural network. It contains the provided input values as well as the calculated output values.

```
<xs:element name="file" type="xs:string" minOccurs="0"/>
```

5. Application and Evaluation of ViNNSL 2.0

This chapter contains two use cases to show the application of ViNNSL 2.0. Furthermore, it shows how the XML instances were evaluated against the schemas.

5.1. Use Case 1: Face recognition using a backpropagation network

This example is based on the developed backpropagation network for face recoginition by HUQQANI et al. (2013). Data sources for this neural network are images of 32x32 pixels. Examples are shown in Figure 5.1. Every image pictures a face looking to the left or right side, up or down, with closed or open eyes, etc. Figure 5.2 gives an overview of the network structure and parallelisation. The neural network shall be executed on a GPU based system. It can be trained or executed. The input layer requires a single dimension with 960 input neurons. The network has to contain a hiddenlayer with one dimension and 1 to 1024 neurons. The output layer has only one neuron. The schemas are linked to each other by the identifiers' numerical prefix. The multithreaded GPU program was compiled by CUDA NVCC 3.0 and runs on a Tesla C1060 graphics card (240x 1296 MHz streaming cores, 4GB memory at 800MHz).

Figure 5.1.: Image example for the face recognition network (HUQQANI et al. 2013)

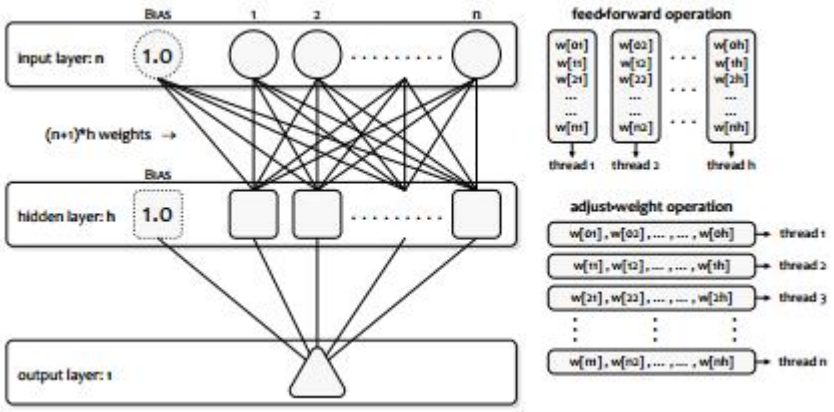

Figure 5.2.: Neural network structure and parallelisation (HUQQANI et al. 2013)

Hardware specification of the neural network:

- CPU program: dual processors Xoen 5570 2.93GHz quad core with hyperthreading (16 logic cores)
- GPU program: tesla C1060 GPU with following features:
- GPUmemory: 4 GB
- GPU memory frequency: 800 MHz
- Max Block size: 512
- Max Blocks: 512*127*512*127
- Threads: 512*512*127*512*127
- Max work blocks: 30
- Max clock frequency: 1296 MHz
- Global memory size: 4294770688 bytes
- Max constant buffer size: 65536 bytes
- Local memory size: 16384 bytes

5.1.1. Creator provides the description schema

The neural network creator has to create the *description* schema of his application. It is the basis for users to define their neural network instances.

The *identifier* is a system e.g. N2Sky generated ID. First of all, the creator fills in

the *metadata*, which can be used as overview for the whole *description* schema. *Creator* is necessary for communication between the creator and users. *ProblemDomain* contains the neural network attributes and fields of application. The creator can use predefined fields or specify his own. Afterwards, the endpoints have to be provided. They define what users can do with the network. For automatisation, the creator has to provide the execution environment of the neural network. The *serial* element is always set to true. In case parallelism is not available, serial execution will be used. The hardware can be either *general* or *special*. The creator can use decimal (*valueparameter*), boolean (*boolparameter*) and string (*comboparameter*) values to define the hardware specification.

Afterwards, the structure of the neural network has to be defined. The creator sets the *ID* for the input layer and its *dimension* and *size*. For these values has to be a range given. The same procedure follows for the hidden and output layer. Then the *connections* parameter has to be set. ViNNSL empowers creators to define custom parameter for their neural network. In this example are values for learning rate, bias, momentum, threshold and activation function. Finally the expected data for the neural network has to be described. Therefore, a general data explanation as well as a description for each representation possibility can be given.

```
<?xml version="1.1" encoding="UTF-8"?>
<description xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="file:/C:/…/desc_schema.xsd">
  <identifier>8765445678123desc</identifier>
  <metadata>
    <paradigm>classification</paradigm>
    <name>backpropagation classification</name>
    <description>face recognition using gpu</description>
    <version>
      <major>1</major>
      <minor>5</minor>
    </version>
  </metadata>
  <creator>
    <name>Author1</name>
    <contact>author1@institution.com</contact>
  </creator>
  <problemDomain>
    <propagationType type="feedforward">
      <learningType>supervised</learningType>
    </propagationType>
    <applicationField>EMS</applicationField>
    <applicationField>Operations</applicationField>
    <applicationField>FaceRecognition</applicationField>
    <networkType>Backpropagation</networkType>
    <problemType>Classifiers</problemType>
  </problemDomain>
```

```
<endpoints>
  <train>true</train>
  <retrain>true</retrain>
  <evaluate>true</evaluate>
</endpoints>
<executionEnvironment>
  <serial>true</serial>
  <parallel>
    <software>
      <data>
        <structural>
          <spmd>
            <gpgpu version="3.0">CUDA NVCC</gpgpu>
          </spmd>
        </structural>
      </data>
    </software>
    <hardware>
      <general>
        <comboparameter name="CPU"> dual processors Xoen 5570 2.93GHz quad core</comboparameter>
        <valueparameter name="CPULogicCores">16</valueparameter>
        <comboparameter name="GPU"> tesla C1060 GPU </comboparameter>
        <comboparameter name="GPUDescription">The following values are GPU features</comboparameter>
        <comboparameter name="GPUmemory">4 GB</comboparameter>
        <comboparameter name="GPUmemoryFrequency">800 MHz</comboparameter>
        <valueparameter name="maxBlockSize">512</valueparameter>
        <valueparameter name="maxBlocks">4228120576</valueparameter>
        <valueparameter name="maxThreads">2164797734912</valueparameter>
        <valueparameter name="maxWorkBlocks">30</valueparameter>
        <valueparameter name="maxClockFrequMhz">1296</valueparameter>
        <valueparameter name="globalMemorySizeBytes">4294770688</valueparameter>
        <valueparameter name="maxConstantBufferBytes">65536</valueparameter>
        <valueparameter name="localMemorySizeBytes">16384</valueparameter>
      </general>
    </hardware>
  </parallel>
</executionEnvironment>
<structure>
  <input>
    <ID>Input1</ID>
    <dimension>
      <min>1</min>
      <max>1</max>
    </dimension>
    <size>
      <min>960</min>
      <max>960</max>
    </size>
  </input>
```

```
    <hidden>
      <ID>Hidden1</ID>
      <dimension>
        <min>1</min>
        <max>1</max>
      </dimension>
      <size>
        <min>1</min>
        <max>1024</max>
      </size>
    </hidden>
    <output>
      <ID>Output1</ID>
      <dimension>
        <min>1</min>
        <max>1</max>
      </dimension>
      <size>
        <min>1</min>
        <max>1</max>
      </size>
    </output>
    <connections>fullconnected</connections>
  </structure>
  <parameters>
    <valueparameter>learningrate</valueparameter>
    <valueparameter>biasInput</valueparameter>
    <valueparameter>biasHidden</valueparameter>
    <valueparameter>momentum</valueparameter>
    <comboparameter>activationfunction</comboparameter>
    <valueparameter>threshold</valueparameter>
  </parameters>
  <data>
    <description>Input are face images with 32x30 pixels</description>
    <tabledescription>no input as table possible</tabledescription>
    <filedescription>Prepare the input as file by reading the image files</filedescription>
  </data>
</description>
```

5.1.2. User provides Definition schema

The user has to provide a *definition* schema, which serves as basis for the system to create the neural network instance. The schema ID is set by the system, but the user has to provide the *problemDomain*, which is specified in the *description* schema. The neural network instance applicability has to be set using the *applicationField* element. However, only values given in the *description* schema are valid. The task for the neural network is given in the *endpoints* element.

The user has to select one of the execution environments, which are specified in the *description* schema. The *definition* schema accepts only one environment. Unlike the *description* schema, which provides ranges, the *definition* schema must contain a concrete structure. The layers are identified using the *ID* element. Their *dimension* and *size* require a value within the range given in the *description* schema. The connections are fullconnected. This means this element has to contain the connections between the layers of the neural network.

The *resultSchema* element defines, which results, based on this neural network definition, shall be delivered. In this example, it's an *instance* and a *training* result schema. The *parameters* element has to contain the values for the parameters specified in the *description* schema. The *parameter* values from the *description* schema are the values of the *name* attribute in the *definition* schema. The values are the values for the neural network, e.g. 0.4 as learning rate and 0.1 as momentum. The training data is given in the *data* element.

```
<?xml version="1.1" encoding="UTF-8"?>
<definition xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="file:/C:/…/def_schema.xsd">
  <identifier>8765445678123def</identifier>
  <problemDomain>
    <propagationType type="feedforward">
      <learningType>supervised</learningType>
    </propagationType>
    <applicationField>EMS</applicationField>
    <applicationField>FaceRecognition</applicationField>
    <networkType>Backpropagation</networkType>
    <problemType>Classifiers</problemType>
  </problemDomain>
  <endpoints>train</endpoints>
  <executionEnvironment>
    <serial>true</serial>
    <parallel>
      <software>
        <data>
          <structural>
            <spmd>
              <gpgpu version="3.0">CUDA NVCC</gpgpu>
            </spmd>
          </structural>
        </data>
      </software>
```

```
      <hardware>
        <general>
          <comboparameter name="CPU"> dual processors Xoen 5570 2.93GHz quad core</comboparameter>
          <valueparameter name="CPULogicCores">16</valueparameter>
          <comboparameter name="GPU"> tesla C1060 GPU </comboparameter>
          <comboparameter name="GPUDescription">The following values are GPU features</comboparameter>
          <comboparameter name="GPUmemory">4 GB</comboparameter>
          <comboparameter name="GPUmemoryFrequency">800 MHz</comboparameter>
          <valueparameter name="maxBlockSize">512</valueparameter>
          <valueparameter name="maxBlocks">4228120576</valueparameter>
          <valueparameter name="maxThreads">2164797734912</valueparameter>
          <valueparameter name="maxWorkBlocks">30</valueparameter>
          <valueparameter name="maxClockFrequMhz">1296</valueparameter>
          <valueparameter name="globalMemorySizeBytes">4294770688</valueparameter>
          <valueparameter name="maxConstantBufferBytes">65536</valueparameter>
          <valueparameter name="localMemorySizeBytes">16384</valueparameter>
        </general>
      </hardware>
    </parallel>
  </executionEnvironment>
  <structure>
    <input>
      <ID>Input1</ID>
      <dimension>1</dimension>
      <size>960</size>
    </input>
    <hidden>
      <ID>Hidden1</ID>
      <dimension>1</dimension>
      <size>1024</size>
    </hidden>
    <output>
      <ID>Output1</ID>
      <dimension>1</dimension>
      <size>1</size>
    </output>
    <connections>
      <fullconnected>
        <fromblock>Input1</fromblock>
        <toblock>Hidden1</toblock>
        <fromblock>Hidden1</fromblock>
        <toblock>Output1</toblock>
      </fullconnected>
    </connections>
  </structure>
  <resultSchema>
    <instance>true</instance>
    <training>true</training>
  </resultSchema>
  <parameters>
    <valueparameter name="learningrate">0.4</valueparameter>
    <valueparameter name="biasInput">1</valueparameter>
    <valueparameter name="biasHidden">1</valueparameter>
    <valueparameter name="momentum">0.1</valueparameter>
    <comboparameter name="activationfunction">sigmoid</comboparameter>
    <valueparameter name="threshold">0.00001</valueparameter>
  </parameters>
  <data>
    <description>Input are face images with 32x30 pixels prepared like it's stated in the description schema</
description>
    <dataSchemaID> 8765445678123data</dataSchemaID>
  </data>
</definition>
```

Beside the *definition* schema, the user also has to provide a *data* schema. For network training it provides the input and output values. In this case the values are stored within a file and the path to its location is provided.

```
<dataschema xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="file:/C:/…/data_schema.xsd">
   <identifier>8765445678123data1</identifier>
   <creationdate>2014-01-27</creationdate>
   <data>
      <file>D:\…\training\trainingFile.txt</file>
   </data>
</dataschema>
```

5.1.3. System delivers Training Result schema and Instance schema

Based on the neural network definition provided with the *definition* schema the system returns a *training result* schema providing information on the neural network training process. The *executionEnvironment* value may differ from the *definition* schema. For example, if the parallel processing cannot be fulfilled the system has to switch to serial processing automatically.

Identifier and *creationdate* are automatically generated values. The *data* element contains the path to the file with the training data and *weightmatrix* to the file containing the weightmatrix values from the neural network training. *Epochs* is the number of training epochs and *meanerror* their mean error value. *Totalexecutiontime* is the total time of the neural network execution in seconds. *Epocherrorvalue* contains the path to a file providing information on every training epoch. *Activationfunction*, *learningrate*, *momentum* and *threshold* are the values specified in the *definition* schema. The same case applies to the *executionEnvironment*, *propagationType*, *learningType* and *networkType*. However, the neural network has more than one bias value. Therefore, the *bias* element contains several elements of type *valueparameter*. Each presents one bias value given in the *definition* schema.

```
<trainingresultschema xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="file:/C:/…/trainres_schema.xsd">
  <identifier>8765445678123defdata1trainres</identifier>
  <creationdate>2014-01-31</creationdate>
  <data>
    <file>D:\...\training\trainingFile.txt</file>
  </data>
  <weightmatrix>D:\...\training\trainingFileWeightmatrix.txt</weightmatrix>
  <epochs>100</epochs>
  <meanerror>0.001473</meanerror>
  <activationfunction>sigmoid</activationfunction>
  <totalexecutiontime>54</totalexecutiontime> <!--always in seconds-->
  <epocherrorvalue> D:\...\training\epocherrorvalue.txt </epocherrorvalue>
  <learningrate>0.4</learningrate>
  <momentum>0.1</momentum>
  <threshold>0.00001</threshold>
  <bias>
  <bias>
    <valueparameter name="biasInput">1</valueparameter>
    <valueparameter name="biasHidden">1</valueparameter>
  </bias>
  <executionEnvironment>
    <serial>true</serial>
    <parallel>
      <software>
        <data>
          <structural>
            <spmd>
              <gpgpu version="3.0">CUDA NVCC</gpgpu>
            </spmd>
          </structural>
        </data>
      </software>
      <hardware>
        <general>
          <comboparameter name="CPU"> dual processors Xoen 5570 2.93GHz quad core</comboparameter>
          <valueparameter name="CPULogicCores">16</valueparameter>
          <comboparameter name="GPU"> tesla C1060 GPU </comboparameter>
          <comboparameter name="GPUDescription">The following values are GPU features</comboparameter>
          <comboparameter name="GPUmemory">4 GB</comboparameter>
          <comboparameter name="GPUmemoryFrequency">800 MHz</comboparameter>
          <valueparameter name="maxBlockSize">512</valueparameter>
          <valueparameter name="maxBlocks">4228120576</valueparameter>
          <valueparameter name="maxThreads">2164797734912</valueparameter>
          <valueparameter name="maxWorkBlocks">30</valueparameter>
          <valueparameter name="maxClockFrequMhz">1296</valueparameter>
          <valueparameter name="globalMemorySizeBytes">4294770688</valueparameter>
          <valueparameter name="maxConstantBufferBytes">65536</valueparameter>
          <valueparameter name="localMemorySizeBytes">16384</valueparameter>
        </general>
      </hardware>
    </parallel>
  </executionEnvironment>
  <propagationType type="feedforward">
    <learningType>supervised</learningType>
  </propagationType>
  <networkType>Backpropagation</networkType>
</trainingresultschema>
```

The result of the neural network training is an *instance* schema, which can be used with actual data. *Identifier* and *creationdate* are system generated values. The *struc-*

ture, *executionEnvironment*, *problemDomain* and *activationfunction* are identical to the elements in the *definition* schema. The *weightmatrix* contains the path to the file with the values. In case the specified endpoint is "train", the values of the weightmatrix are the same as those of the weightmatrix file in the *trainingresult* schema.

```
<?xml version="1.1" encoding="UTF-8"?>
<instanceschema xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="file:/C:/…/inst_schema.xsd">
  <identifier>8765445678123defdata1inst</identifier>
  <creationdate>2014-02-11</creationdate>
  <structure>
    <input>
      <ID>Input1</ID>
      <dimension>1</dimension>
      <size>960</size>
    </input>
    <hidden>
      <ID>Hidden1</ID>
      <dimension>1</dimension>
      <size>1024</size>
    </hidden>
    <output>
      <ID>Output1</ID>
      <dimension>1</dimension>
      <size>1</size>
    </output>
    <connections>
      <fullconnected>
        <fromblock>Input1</fromblock>
        <toblock>Hidden1</toblock>
        <fromblock>Hidden1</fromblock>
        <toblock>Output1</toblock>
      </fullconnected>
    </connections>
  </structure>
  <executionEnvironment>
    <serial>true</serial>
    <parallel>
      <software>
        <data>
          <structural>
            <spmd>
              <gpgpu version="3.0">CUDA NVCC</gpgpu>
            </spmd>
          </structural>
        </data>
      </software>
      <hardware>
        <general>
          <comboparameter name="CPU"> dual processors Xoen 5570 2.93GHz quad core</comboparameter>
          <valueparameter name="CPULogicCores">16</valueparameter>
          <comboparameter name="GPU"> tesla C1060 GPU </comboparameter>
          <comboparameter name="GPUDescription">The following values are GPU features</comboparameter>
          <comboparameter name="GPUmemory">4 GB</comboparameter>
          <comboparameter name="GPUmemoryFrequency">800 MHz</comboparameter>
```

```
            <valueparameter name="maxBlockSize">512</valueparameter>
            <valueparameter name="maxBlocks">4228120576</valueparameter>
            <valueparameter name="maxThreads">2164797734912</valueparameter>
            <valueparameter name="maxWorkBlocks">30</valueparameter>
            <valueparameter name="maxClockFrequMhz">1296</valueparameter>
            <valueparameter name="globalMemorySizeBytes">4294770688</valueparameter>
            <valueparameter name="maxConstantBufferBytes">65536</valueparameter>
            <valueparameter name="localMemorySizeBytes">16384</valueparameter>
          </general>
        </hardware>
      </parallel>
    </executionEnvironment>
    <problemDomain>
      <networkType>Backpropagation</networkType>
      <propagationType type="feedforward">
        <learningType>unsupervised</learningType>
      </propagationType>
      <applicationField>EMS</applicationField>
      <applicationField>FaceRecognition</applicationField>
      <problemType>Classifiers</problemType>
    </problemDomain>
    <weightmatrix> D:\...\instance\instanceWeightmatrix.txt </weightmatrix>
    <activationfunction>sigmoid</activationfunction>
</instanceschema>
```

5.1.4. User provides Instance schema and Data schema

The user is able to provide the neural network application with an *instance* schema and a *data* schema. The *data* schema contains the input values only. In this example, the *instance* schema is the same, which was created as result of the neural network training based on the *definition* schema in section 5.1.2

```
<?xml version="1.1" encoding="UTF-8"?>
<instanceschema xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="file:/C:/…/inst_schema.xsd">
    <identifier>8765445678123defdata1inst</identifier>
    <creationdate>2014-02-11</creationdate>
    <structure>
      <input>
        <ID>Input1</ID>
        <dimension>1</dimension>
        <size>960</size>
      </input>
      <hidden>
        <ID>Hidden1</ID>
        <dimension>1</dimension>
        <size>1024</size>
      </hidden>
      <output>
        <ID>Output1</ID>
        <dimension>1</dimension>
        <size>1</size>
      </output>
```

```
      <connections>
        <fullconnected>
          <fromblock>Input1</fromblock>
          <toblock>Hidden1</toblock>
          <fromblock>Hidden1</fromblock>
          <toblock>Output1</toblock>
        </fullconnected>
      </connections>
    </structure>
    <executionEnvironment>
      <serial>true</serial>
      <parallel>
        <software>
          <data>
            <structural>
              <spmd>
                <gpgpu version="3.0">CUDA NVCC</gpgpu>
              </spmd>
            </structural>
          </data>
        </software>
        <hardware>
          <general>
            <comboparameter name="CPU"> dual processors Xoen 5570 2.93GHz quad core</comboparameter>
            <valueparameter name="CPULogicCores">16</valueparameter>
            <comboparameter name="GPU"> tesla C1060 GPU </comboparameter>
            <comboparameter name="GPUDescription">The following values are GPU features</comboparameter>
            <comboparameter name="GPUmemory">4 GB</comboparameter>
            <comboparameter name="GPUmemoryFrequency">800 MHz</comboparameter>
            <valueparameter name="maxBlockSize">512</valueparameter>
            <valueparameter name="maxBlocks">4228120576</valueparameter>
            <valueparameter name="maxThreads">2164797734912</valueparameter>
            <valueparameter name="maxWorkBlocks">30</valueparameter>
            <valueparameter name="maxClockFrequMhz">1296</valueparameter>
            <valueparameter name="globalMemorySizeBytes">4294770688</valueparameter>
            <valueparameter name="maxConstantBufferBytes">65536</valueparameter>
            <valueparameter name="localMemorySizeBytes">16384</valueparameter>
          </general>
        </hardware>
      </parallel>
    </executionEnvironment>
    <problemDomain>
      <networkType>Backpropagation</networkType>
      <propagationType type="feedforward">
        <learningType>supervised</learningType>
      </propagationType>
      <applicationField>EMS</applicationField>
      <applicationField>FaceRecognition</applicationField>
      <problemType>Classifiers</problemType>
    </problemDomain>
    <weightmatrix> D:\...\instance\instanceWeightmatrix.txt </weightmatrix>
    <activationfunction>sigmoid</activationfunction>
    <dataSchemaID>8765445678123data2</dataSchemaID >
</instanceschema>

<dataschema xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="file:/C:/…/data_schema.xsd">
    <identifier>8765445678123data2</identifier>
    <creationdate>2014-01-27</creationdate>
    <data>
      <file>D:\…\execution\executionFile.txt</file>
    </data>
</dataschema>
```

5.1.5. System delivers Result schema

After the execution, the user gets the *result* schema from the neural network application. Depending on the neural network implementation the *result* schema may contain a 2D diagram, a table and a file with the result values. *Identifier* and *creationdate* are system generated values. *Vinnslinstance* contains the *identifier* value of the used *instance* schema in order to assign the XML files to each other.

```
<resultschema xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="file:/C:/.../result_schema.xsd">
  <identifier>8765445678123data2res</identifier>
  <instanceSchemaID>8765445678123defdata1inst</ instanceSchemaID>
  <creationdate>2014-02-13</creationdate>
  <file>D:\...\execution\executionFileResult.txt</file>
</resultschema>
```

5.1.6. User retrains the neural network

For retraining, the user has to provide the original *definition* schema and the *instance* schema, together with a new *data* schema providing the training values. The user can specify if a *training result* schema and/or *instance* schema shall be delivered. In this example, a new instance schema is requested only and the *definition* schema is mostly identical to the schema in section 5.1.2. However, the endpoints element's value changed to "retrain".

```
<?xml version="1.1" encoding="UTF-8"?>
<definition xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="file:/C:/.../def_schema.xsd">
  <identifier>8765445678123def</identifier>
  <problemDomain>
    <propagationType type="feedforward">
      <learningType>supervised</learningType>
    </propagationType>
    <applicationField>EMS</applicationField>
    <applicationField>FaceRecognition</applicationField>
    <networkType>Backpropagation</networkType>
    <problemType>Classifiers</problemType>
  </problemDomain>
  <endpoints>retrain</endpoints>
  <executionEnvironment>
    <serial>true</serial>
    <parallel>
      <software>
        <data>
          <structural>
            <spmd>
              <gpgpu version="3.0">CUDA NVCC</gpgpu>
            </spmd>
          </structural>
        </data>
      </software>
      <hardware>
        <general>
          <comboparameter name="CPU"> dual processors Xoen 5570 2.93GHz quad core</comboparameter>
```

```
            <valueparameter name="CPULogicCores">16</valueparameter>
            <comboparameter name="GPU"> tesla C1060 GPU </comboparameter>
            <comboparameter name="GPUDescription">The following values are GPU features</comboparameter>
            <comboparameter name="GPUmemory">4 GB</comboparameter>
            <comboparameter name="GPUmemoryFrequency">800 MHz</comboparameter>
            <valueparameter name="maxBlockSize">512</valueparameter>
            <valueparameter name="maxBlocks">4228120576</valueparameter>
            <valueparameter name="maxThreads">2164797734912</valueparameter>
            <valueparameter name="maxWorkBlocks">30</valueparameter>
            <valueparameter name="maxClockFrequMhz">1296</valueparameter>
            <valueparameter name="globalMemorySizeBytes">4294770688</valueparameter>
            <valueparameter name="maxConstantBufferBytes">65536</valueparameter>
            <valueparameter name="localMemorySizeBytes">16384</valueparameter>
          </general>
        </hardware>
      </parallel>
    </executionEnvironment>
    <structure>
      <input>
        <ID>Input1</ID>
        <dimension>1</dimension>
        <size>960</size>
      </input>
      <hidden>
        <ID>Hidden1</ID>
        <dimension>1</dimension>
        <size>1024</size>
      </hidden>
      <output>
        <ID>Output1</ID>
        <dimension>1</dimension>
        <size>1</size>
      </output>
      <connections>
        <fullconnected>
          <fromblock>Input1</fromblock>
          <toblock>Hidden1</toblock>
          <fromblock>Hidden1</fromblock>
          <toblock>Output1</toblock>
        </fullconnected>
      </connections>
    </structure>
    <resultSchema>
      <instance>true</instance>
      <training>false</training>
    </resultSchema>
    <parameters>
      <valueparameter name="learningrate">0.4</valueparameter>
      <valueparameter name="biasInput">1</valueparameter>
      <valueparameter name="biasHidden">1</valueparameter>
      <valueparameter name="momentum">0.1</valueparameter>
      <comboparameter name="activationfunction">sigmoid</comboparameter>
      <valueparameter name="threshold">0.00001</valueparameter>
    </parameters>
    <data>
      <description>Input are face images with 32x30 pixels prepared like stated in the description schema</
description>
      <dataSchemaID> 8765445678123data3</dataSchemaID >
    </data>
    <instanceSchemaID>8765445678123defdata1inst </instanceSchemaID>
</definition>
```

5.1.7. System delivers a new Instance schema

Based on the retraining results, a new *instance* schema is provided by the neural network application. Concretely spoken the weightmatrix will change only.

```
<?xml version="1.1" encoding="UTF-8"?>
<instanceschema xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="file:/C:/…/inst_schema.xsd">
  <identifier>8765445678123defdata3inst2</identifier>
  <creationdate>2014-02-13</creationdate>
  <structure>
    <input>
      <ID>Input1</ID>
      <dimension>1</dimension>
      <size>960</size>
    </input>
    <hidden>
      <ID>Hidden1</ID>
      <dimension>1</dimension>
      <size>1024</size>
    </hidden>
    <output>
      <ID>Output1</ID>
      <dimension>1</dimension>
      <size>1</size>
    </output>
    <connections>
      <fullconnected>
        <fromblock>Input1</fromblock>
        <toblock>Hidden1</toblock>
        <fromblock>Hidden1</fromblock>
        <toblock>Output1</toblock>
      </fullconnected>
    </connections>
  </structure>
  <executionEnvironment>
    <serial>true</serial>
    <parallel>
      <software>
        <data>
          <structural>
            <spmd>
              <gpgpu version="3.0">CUDA NVCC</gpgpu>
            </spmd>
          </structural>
        </data>
      </software>
```

```
        <hardware>
          <general>
            <comboparameter name="CPU"> dual processors Xoen 5570 2.93GHz quad core</comboparameter>
            <valueparameter name="CPULogicCores">16</valueparameter>
            <comboparameter name="GPU"> tesla C1060 GPU </comboparameter>
            <comboparameter name="GPUDescription">The following values are GPU features</comboparameter>
            <comboparameter name="GPUmemory">4 GB</comboparameter>
            <comboparameter name="GPUmemoryFrequency">800 MHz</comboparameter>
            <valueparameter name="maxBlockSize">512</valueparameter>
            <valueparameter name="maxBlocks">4228120576</valueparameter>
            <valueparameter name="maxThreads">2164797734912</valueparameter>
            <valueparameter name="maxWorkBlocks">30</valueparameter>
            <valueparameter name="maxClockFrequMhz">1296</valueparameter>
            <valueparameter name="globalMemorySizeBytes">4294770688</valueparameter>
            <valueparameter name="maxConstantBufferBytes">65536</valueparameter>
            <valueparameter name="localMemorySizeBytes">16384</valueparameter>
          </general>
        </hardware>
      </parallel>
    </executionEnvironment>
    <problemDomain>
      <networkType>Backpropagation</networkType>
      <propagationType type="feedforward">
        <learningType>unsupervised</learningType>
      </propagationType>
      <applicationField>EMS</applicationField>
      <problemType>Classifiers</problemType>
    </problemDomain>
    <weightmatrix> D:\...\instance\instanceWeightmatrix2.txt </weightmatrix>
    <activationfunction>sigmoid</activationfunction>
</instanceschema>
```

5.2. Use Case 2: Parallelised backpropagation network on a hypercube system

For the evaluation of the Structural Data Parallel (SDP) method a parallel neural network simulation of the backpropagation neural network paradigm was developed. The important part of the SDP approach is the identification of data structure and their mapping onto the neural network structure. Data arrays transformed into subarrays, which are afterwards distributed among the processor. Examples are shown shown in Figure 5.3. The neural network was executed on an Intel iPSC860 hypercube system with four processors. (SCHIKUTA 1997)

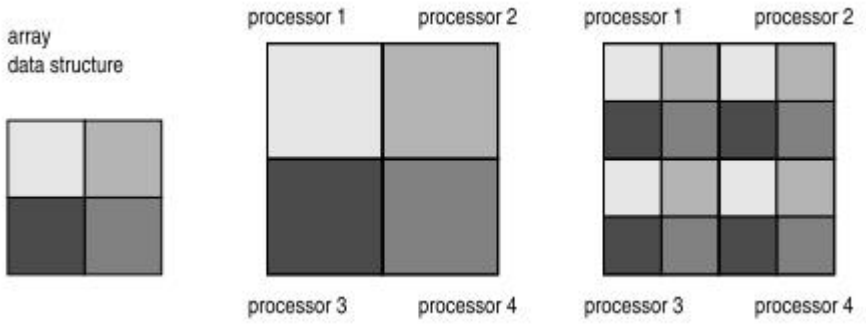

Figure 5.3.: Data distribution schemes (SCHIKUTA 1997)

5.2.1. Creator provides a Description schema

The neural network creator has to create the *description* schema of his application. It is the basis for users to define their neural network instances.

```
<?xml version="1.1" encoding="UTF-8"?>
<description xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:noNamespaceSchemaLocation="file:/C:/…/desc_schema.xsd">
   <identifier>8765445678666desc</identifier>
   <metadata>
      <paradigm>Approximation</paradigm>
      <name>Backpropagation SDP</name>
      <description>SPD performance test</description>
      <version>
         <major>1</major>
         <minor>0</minor>
      </version>
   </metadata>
   <creator>
      <name>Author2</name>
      <contact>author2@institution.com</contact>
   </creator>
   <problemDomain>
      <propagationType type="feedforward">
         <learningType>supervised</learningType>
      </propagationType>
      <applicationField>SDPtest</applicationField>
      <networkType>Backpropagation</networkType>
      <problemType>Approximators</problemType>
   </problemDomain>
```

```
<endpoints>
   <train>true</train>
   <retrain>false</retrain>
   <evaluate>false</evaluate>
</endpoints>
<executionEnvironment>
   <serial>true</serial>
   <parallel>
      <software>
         <data>
            <structural>
               <spmd>
                  <hypercube version="860">iPSC</hypercube>
               </spmd>
            </structural>
         </data>
      </software>
      <hardware>
         <special>
            <comboparameter name="HWproducer">Intel</comboparameter>
            <comboparameter name="HWdescription">Intel iPSC/860</comboparameter>
            <valueparameter name="processors">8</valueparameter>
         </special>
      </hardware>
   </parallel>
</executionEnvironment>
<structure>
   <input>
      <ID>Input1</ID>
      <dimension>
         <min>1</min>
         <max>1</max>
      </dimension>
      <size>
         <min>5</min>
         <max>10</max>
      </size>
   </input>
   <hidden>
      <ID>Hidden1</ID>
      <dimension>
         <min>1</min>
         <max>1</max>
      </dimension>
      <size>
         <min>1</min>
         <max>20</max>
      </size>
   </hidden>
```

```
    <hidden>
      <ID>Hidden2</ID>
      <dimension>
        <min>1</min>
        <max>1</max>
      </dimension>
      <size>
        <min>1</min>
        <max>16</max>
      </size>
    </hidden>
    <hidden>
      <ID>Hidden3</ID>
      <dimension>
        <min>1</min>
        <max>1</max>
      </dimension>
      <size>
        <min>1</min>
        <max>8</max>
      </size>
    </hidden>
    <output>
      <ID>Output1</ID>
      <dimension>
        <min>1</min>
        <max>1</max>
      </dimension>
      <size>
        <min>1</min>
        <max>1</max>
      </size>
    </output>
    <connections>fullconnected</connections>
  </structure>
  <parameters>
    <valueparameter>learningrate</valueparameter>
    <valueparameter>momentum</valueparameter>
    <comboparameter>activationfunction </comboparameter>
    <valueparameter>threshold</valueparameter>
  </parameters>
  <data>
    <description>High-performance languages are used dor data distribution </description>
    <tabledescription>input as table is not possible</tabledescription>
    <filedescription>Prepare the input as file by providing the values only</filedescription>
  </data>
</description>
```

5.2.2. User provides a Definition schema

The user has to provide a *definition* schema, which serves as basis for the system to create the neural network instance. The parameters are distinguished by their specified name as attribute and value as value. The network shall be trained and the neural network application shall return a *training result* schema of this particular network. The training data is given in the *data* element.

```
<?xml version="1.1" encoding="UTF-8"?>
<definition xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:noNamespaceSchemaLocation="file:/C:/…/def_schema.xsd">
   <identifier>8765445678666def</identifier>
   <problemDomain>
      <propagationType type="feedforward">
         <learningType>supervised</learningType>
      </propagationType>
      <applicationField>SDPtest</applicationField>
      <networkType>Backpropagation</networkType>
      <problemType>Approximators</problemType>
   </problemDomain>
   <endpoints>train</endpoints>
   <executionEnvironment>
      <serial>true</serial>
      <parallel>
         <software>
            <data>
               <structural>
                  <spmd>
                     <hypercube version="860">iPSC</hypercube>
                  </spmd>
               </structural>
            </data>
         </software>
         <hardware>
            <special>
               <comboparameter name="HWproducer">Intel</comboparameter>
               <comboparameter name="HWdescription">Intel iPSC/860</comboparameter>
               <valueparameter name="processors">8</valueparameter>
            </special>
         </hardware>
      </parallel>
   </executionEnvironment>
   <structure>
      <input>
         <ID>Input1</ID>
         <dimension>1</dimension>
         <size>5</size>
      </input>
```

```
    <hidden>
      <ID>Hidden1</ID>
      <dimension>1</dimension>
      <size>20</size>
    </hidden>
    <hidden>
      <ID>Hidden2</ID>
      <dimension>1</dimension>
      <size>12</size>
    </hidden>
    <hidden>
      <ID>Hidden3</ID>
      <dimension>1</dimension>
      <size>4</size>
    </hidden>
    <output>
      <ID>Output1</ID>
      <dimension>1</dimension>
      <size>1</size>
    </output>
    <connections>
      <fullconnected>
        <fromblock>Input1</fromblock>
        <toblock>Hidden1</toblock>
        <fromblock>Hidden1</fromblock>
        <toblock>Hidden2</toblock>
        <fromblock>Hidden2</fromblock>
        <toblock>Hidden3</toblock>
        <fromblock>Hidden3</fromblock>
        <toblock>Output1</toblock>
      </fullconnected>
    </connections>
  </structure>
  <resultSchema>
    <instance>false</instance>
    <training>true</training>
  </resultSchema>
  <parameters>
    <valueparameter name="learningrate">0.4</valueparameter>
    <valueparameter name="momentum">0.1</valueparameter>
    <comboparameter name="activationfunction">sigmoid</comboparameter>
    <valueparameter name="threshold">0.00001</valueparameter>
  </parameters>
  <data>
    <description>High-performance languages are used dor data distribution </description>
    <dataSchemaID> 8765445678666data</dataSchemaID >
  </data>
</definition>
```

Beside the *definition* schema, the user also has to provide a *data* schema. For network training it provides the input and output values. In this case the values are stored within a table.

```
<dataschema xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="file:/C:/…/data_schema.xsd">
  <identifier>8765445678666data</identifier>
   <creationdate>2015-01-27</creationdate>
   <data>
     <table>
       <netinput>2 0.3 5 6 10</netinput>
       <netoutput>3.46</netoutput>
       <netinput>1.4 0.3 55 6.09 2050</netinput>
       <netoutput>2.40</netoutput>
       <netinput>86.5 0.3 512 6.4 101</netinput>
       <netoutput>89.35</netoutput>
       <netinput>206 0.3 5.2905 16 10.10</netinput>
       <netoutput>300</netoutput>
       <netinput>646.45 0.3 500 6.105 16</netinput>
       <netoutput>75</netoutput>
       <netinput>2.153 0.69 5.66 61 7</netinput>
       <netoutput>8.16</netoutput>
       …
     </table>
   </data>
</dataschema>
```

5.2.3. System delivers a training result schema

Based on the neural network definition provided with the *definition* schema the system returns a *training result* schema providing information on the neural network training process.

```
<trainingresultschema xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="file:/C:/…/trainres_schema.xsd">
   <identifier>8765445678666defdata1trainres</identifier>
   <creationdate>2015-01-31</creationdate>
   <data>
     <table>
       <netinput>2 0.3 5 6 10</netinput>
       <netoutput>3.46</netoutput>
       <netinput>1.4 0.3 55 6.09 2050</netinput>
       <netoutput>2.44</netoutput>
       <netinput>86.5 0.3 512 6.4 101</netinput>
       <netoutput>89.35</netoutput>
       <netinput>206 0.3 5.2905 16 10.10</netinput>
       <netoutput>350</netoutput>
       <netinput>646.45 0.3 500 6.105 16</netinput>
       <netoutput>77</netoutput>
       <netinput>2.153 0.69 5.66 61 7</netinput>
       <netoutput>8.16</netoutput>
       …
     </table>
   </data>
   <weightmatrix>D:\...\training\trainingFileWeightmatrix.txt</weightmatrix>
   <epochs>90</epochs>
   <meanerror>0.00123</meanerror>
   <activationfunction>sigmoid</activationfunction>
   <totalexecutiontime>75</totalexecutiontime> <!--always in seconds-->
   <epocherrorvalue> D:\...\training\epocherrorvalue.txt </epocherrorvalue>
   <learningrate>0.4</learningrate>
   <momentum>0.1</momentum>
   <threshold>0.00001</threshold>
```

```
<executionEnvironment>
  <serial>true</serial>
  <parallel>
    <software>
      <data>
        <structural>
          <spmd>
            <hypercube version="860">iPSC</hypercube>
          </spmd>
        </structural>
      </data>
    </software>
    <hardware>
      <special>
        <comboparameter name="HWproducer">Intel</comboparameter>
        <comboparameter name="HWdescription">Intel iPSC/860</comboparameter>
        <valueparameter name="processors">8</valueparameter>
      </special>
    </hardware>
  </parallel>
</executionEnvironment>
<propagationType type="feedforward">
  <learningType>supervised</learningType>
</propagationType>
<networkType>Backpropagation</networkType>
</trainingresultschema>
```

5.3. XML Schema Evaluations

The proposed XML examples for ViNNSL 2.0 were evaluated using Oxygen XML Editor. It allows the validation of XML documents with XML Schema, Relax NG, DTD, NVDL and Schematron. According to its publisher, it supports all XML schema languages. The Oxygen XML Editor's advantage is the full implementation of the XSD 1.1 specification. The tool provides a visual marking of error sources and also links to the exact location in the specification for XML schema errors. (Syncro Soft SRLE. 2002-2015) On the internet exist free XML to XSD evaluators e.g. W3C XML Schema (XSD) Validation online, available under `http://www.utilities-online.info/xsdvalidation`. However, these applications didn't support XSD 1.1.

For the evaluation of the proposed schema files to the XML examples an XSD file and an XML file were created. The first step was the validation of the XSD files shown in figure 5.4. Afterwards the XML examples were inserted into the XML files and validated on well-formedness. An example is provided in figure 5.5. As both files are structured properly they were evaluated against each other. In case of errors Oxygen XML Editor provides a detailed error explanation and also links to the W3C definition. When the XML example matches the XSD schema, the tool shows the message "Document is valid" (figure 5.6).

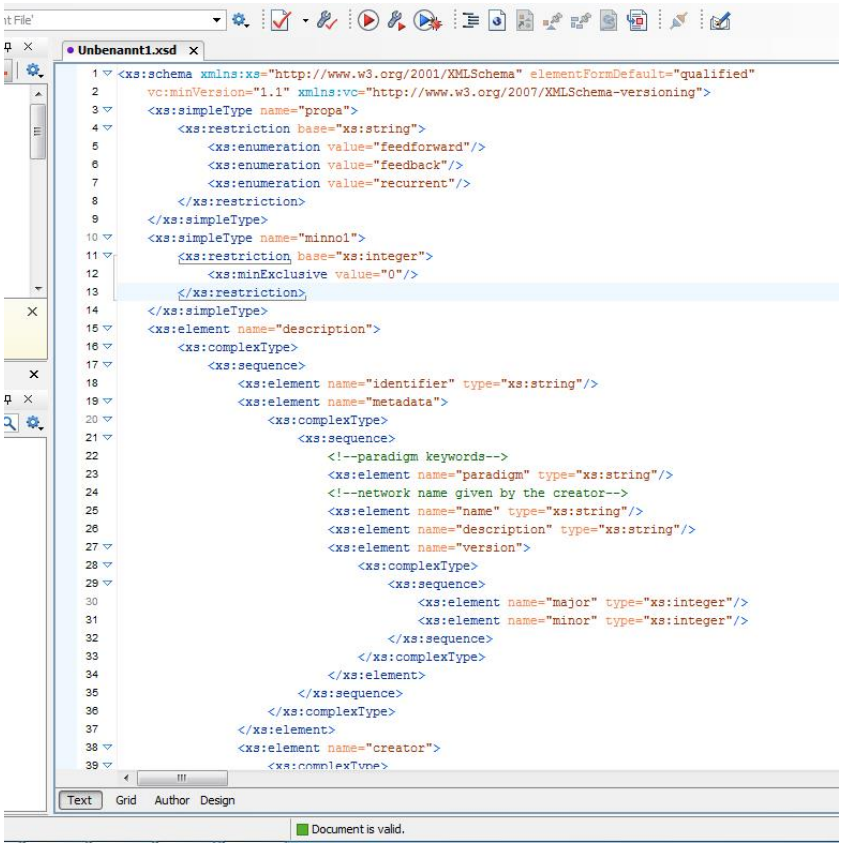

Figure 5.4.: XSD validation Oxygen XML Editor

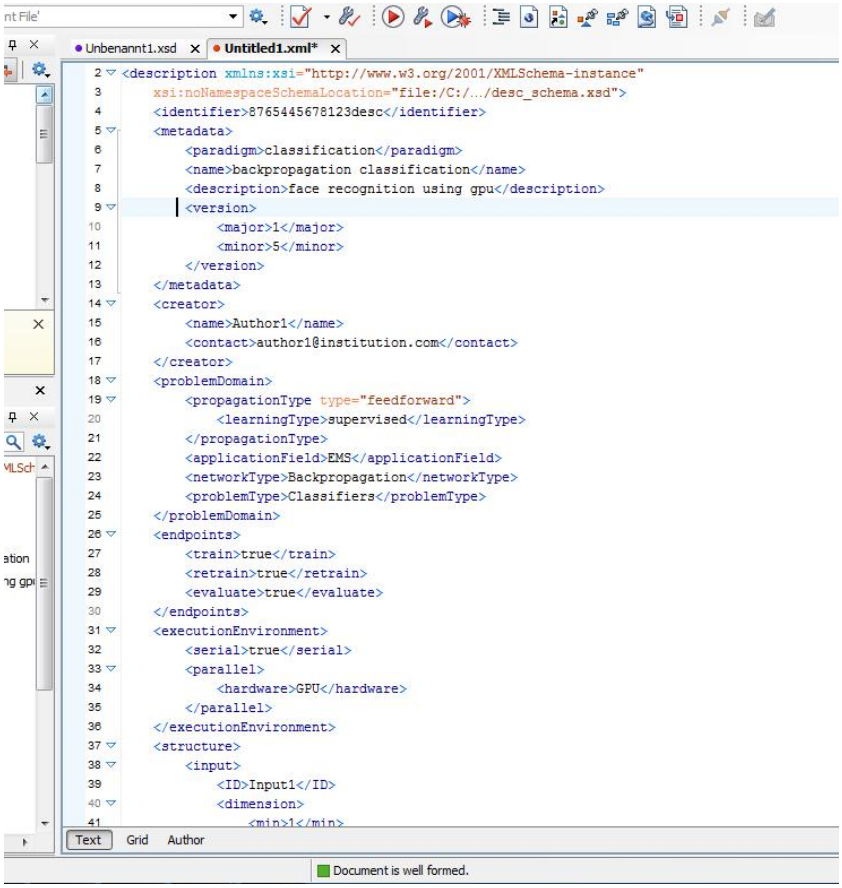

Figure 5.5.: Check well-formedness in Oxygen XML-Editor

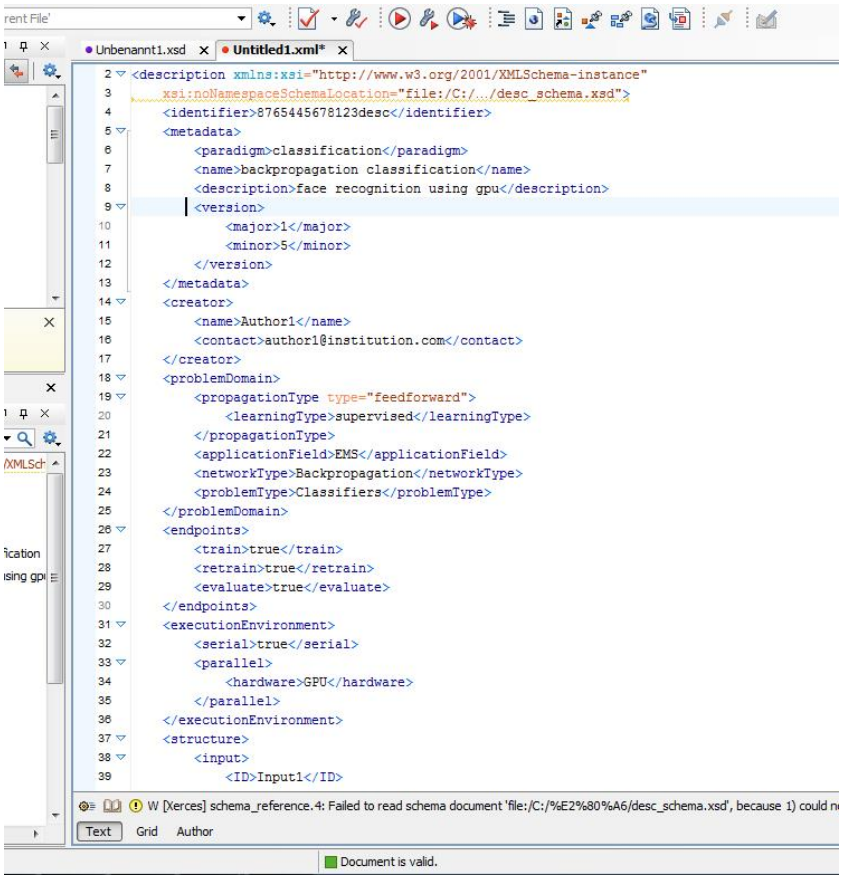

Figure 5.6.: XML matches XSD in Oxygen XML Editor

6. Conclusion and Future Work

The taxonomies developed in chapter 2 provide the possibility to classify neural networks using the neural network classification taxonomy or the neural network paradigm taxonomy. In addition, neural network application domains are introduced. The taxonomies were completed with examples of concrete neural network implementations for application domains. The main focus is on proposed neural networks published within the last five year.

The first proposal of ViNNSL was developed in order to be used in the Neural Network Cube (N2Grid) (section 3.3). However, MANN (2013) introduced N2Sky, a further development of N2Grid, which uses RAVO and the virtual organisation paradigm. [1]

Therefore, ViNNSL is extended to ViNNSL 2.0. Based on the ViNNSL specification and analysis of other neural network specification langauges ViNNSL 2.0 is designed to cover all needs to specify neural networks. The extensions contain new schemas, new elements and XSD assertions to map the ontology into the schemas.

Compared to other neural network specification languages mentioned in this thesis, ViNNSL 2.0 empowers users to link several schemas with each other. For example, in case of retraining the definition schema can be linked to an instance schema and a data schema. The instance schema describes the existing neural network and the data schema provides the new training data.

N2Sky uses the taxonomies in order to propose neural networks to a specific problem. The user is able to search for neural networks by network attributes, for example propagation type, learning type, etc. Other possibilites are searching via problem domain and/or application domain. Depending on the input N2Sky will browse through the taxonomies and presents those networks, which apply to the search results. A graphical representation of the connection between those ontologies is shown in Figure 6.1.They match on the table providing examples of which neural network is applicable for which application domain. ViNNSL 2.0 and its extensions build the communication channel of the components of N2Sky. (SCHIKUTA et al. 2015)

[1] Reference Architecture for Virtual Organizations was developed by Wajeeha Khalil, a PhD student at the University of Vienna. It is presented as a standard for building Virtual Organizations (VOs).

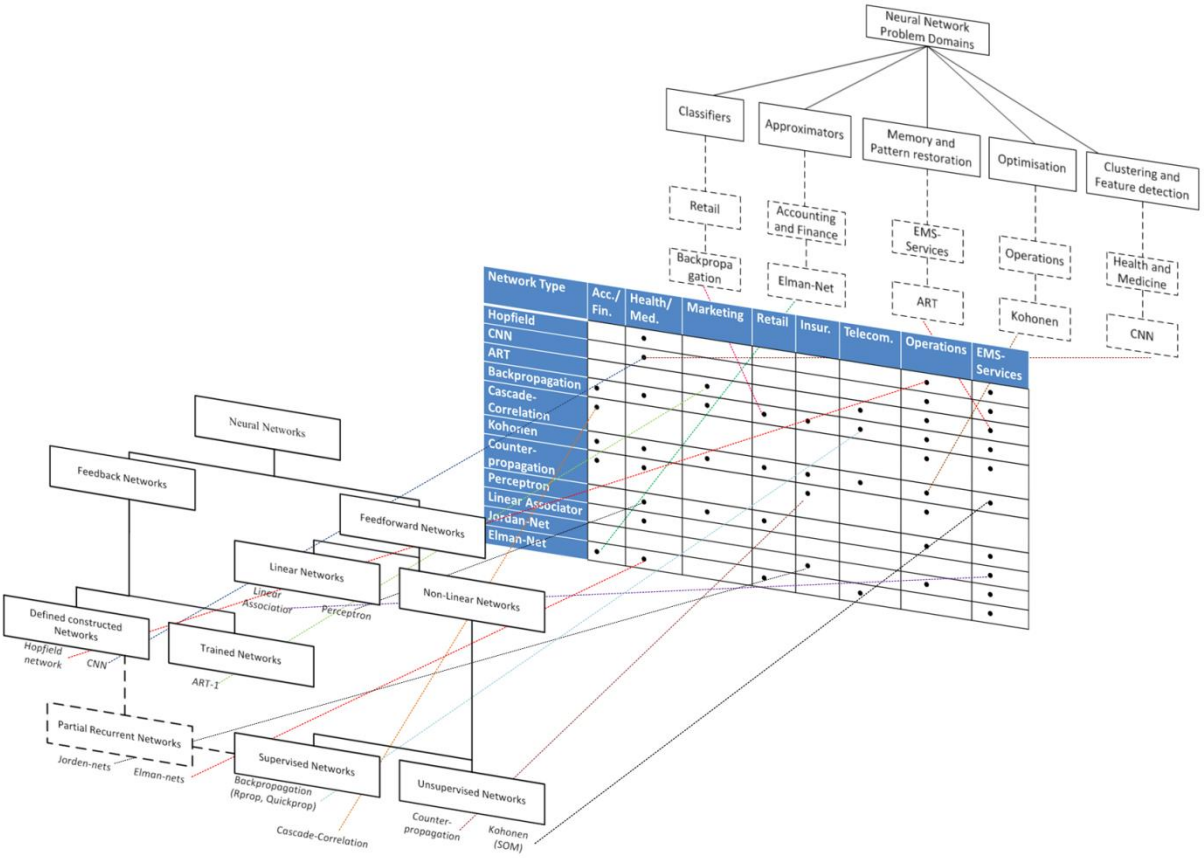

Figure 6.1.: Problem domain ontology for N2Sky modified from SCHIKUTA et al. (2015)

6.1. N2Sky

N2Sky is an environment for creating, training and evaluating neural networks. The system is Cloud-based in order to allow for a growing user community. The simulator interacts with Cloud data resources (i.e. databases) to store and retrieve all relevant data about static and dynamic components of neural network objects. Cloud computing resources provide elastic processing cycles for "power-hungry" neural network simulations. Within N2Sky ViNNSL is used as a standardised description language for describing neural net paradigms and objects called VINNSL. In addition it provides a business model for researchers and students but also for any interested customer. (MANN 2013)

The N2Sky architecture is shown in figure 6.2.

Infrastructure as a Service (IaaS): The IaaS layer is all about managing resources, IaaS "basically provides enhanced virtualisation capabilities. Accordingly, different re-

sources may be provided via a service interface". In the N2Sky architecture the IaaS layer consists of two sublayers: Factory Layer and Infrastructure Enabler Layer. Users need administrative rights for accessing the resources in Layer 0 (contains physical and logical resources) over the resource management services in Layer 1 (allows access to resources).

Platform as a Service (PaaS): PaaS is all about application or service hosting on an abstract or more domain-specific basis. PaaS provides "computational resources via a platform upon which applications and services can be developed and hosted. PaaS typically makes use of dedicated APIs to control the behaviour of a server hosting engine, which executes and replicates the execution according to user requests". It provides transparent access to resources offered by the IaaS layer and transparent access for applications offered by the SaaS layer. Common examples are the Google App Engine, Force.com and Windows Azure. In the N2Sky architecture it is divided into two sublayers. Layer 2 contains domain-independent tools, that are designed not only for use in connection with neural networks. Layer 3 is composed of domain-specific (i.e. neural network-specific) applications.

Software as a Service (SaaS): Finally, the SaaS layer on top of the SPI stack consists of Cloud-enabled ready-to-use applications or services, Saas offers "implementations of specific business functions and business processes, that are provided with specific Cloud capabilities, i.e. they provide applications / services using a Cloud infrastructure or platform, rather than providing Cloud features themselves". Common examples are Google Docs, Microsoft Office 365, SAP Business by Design or Salesforce CRM. In context of N2Sky, SaaS is composed of one layer, namely the Service Layer.

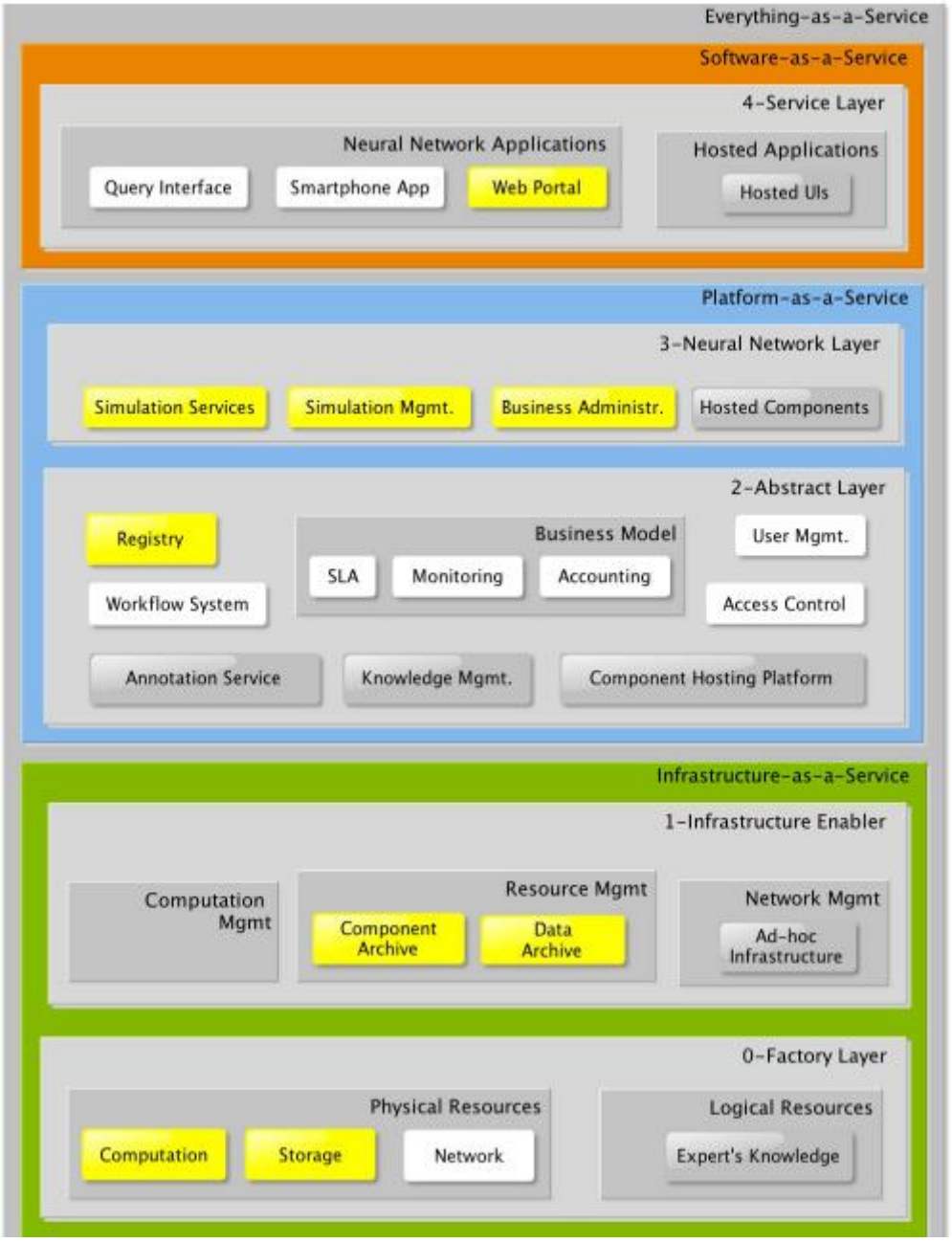

Figure 6.2.: N2Sky Architecture (MANN 2013)

6.1.1. Sample Workflow

The following sample workflow provided by MANN (2013) shall give an explanation on how N2Sky is planned to work. The whole workflow is shown in figure 6.3 and described below.

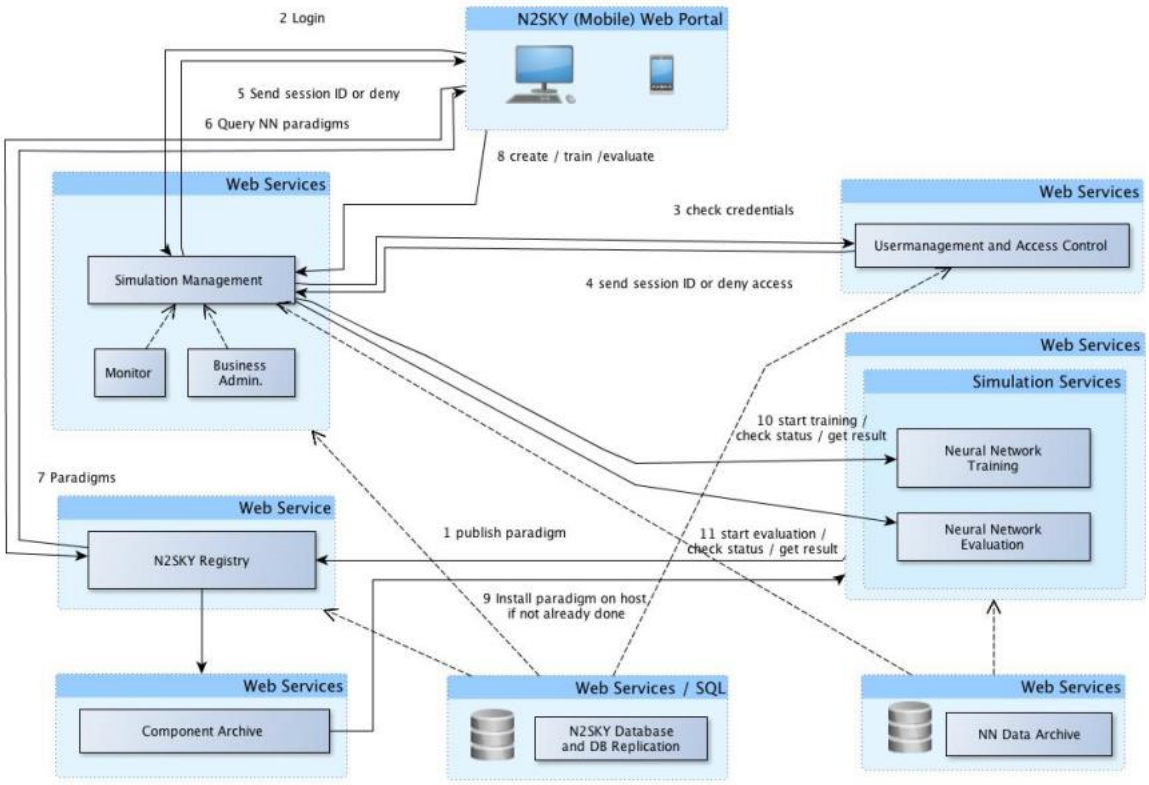

Figure 6.3.: N2Sky Sample workflow (MANN 2013)

1. The developer publishes a paradigm service to N2Sky.
2. Stakeholder login via (mobile) web browser (AJAX request, RESTful Web Service).
3. Simulation management service dispatches login request to User management and access control component per RESTful Web Service.
4. Callback to Simulation management service either sending a new session id or deny access.
5. Callback to (mobile) web browser, redirecting session id or deny access.
6. Query Registry for neural network paradigms for problem solving.
7. Callback to (mobile) web browser by sending paradigm metadata.
8. Create new neural object by using selected paradigm for free, start new Eucalyptus node instance if needed, start training and after them start a new evaluation by using the training result.
9. Before a training task is able to start properly, it is checked if the desired paradigm is provided at this host. If not, a Java EE web archive is deployed to this host by retrieving it from the component archive service.

10. Start a new training thread - Simulation management checks training status periodically until status = 100, then gets result and stores it over the data archive in the database.

11. Start a new evaluation thread - Simulation management checks evaluation status periodically until status = 100, then gets result and stores it over the data archive in the database.

List of Figures

Bibliography

ACHARYA, T. & RAY, A. K. (2005), Image Processing: Principles and Applications, John Wiley & Sons, Inc., Hoboken.

ALTIPARMAK, F., DENGIZ, B. & SMITH, A. E. (2009), 'A general neural network model for estimating telecommunications network reliability', IEEE TRANSACTIONS ON RELIABILITY pp. 2–9.

AMIN, M. S., MAMUN, M., HASHIM, F. H. & HUSAIN, H. (2011), 'Separation of fetal electrocardiography (ecg) from composite ecg using adaptive linear neural network for fetal monitoring', International Journal of the Physical Sciences pp. 5871–5876.

BADIRU, A. B. & CHEUNG, J. Y. (2002), Fuzzy engineering expert systems with nerual network applications, John Wiley & Sons, New York.

BARTZ, R. (2008), Contribution to an xml-based representation of information related to artificial neural networks, in 'Industrial Electronics, 2008. IECON 2008. 34th Annual Conference of IEEE', IEEE, Orlando, pp. 1863–1868.

BEKRANI, M., KHONG, A. W. H. & LOTFIZAD, M. (2011), 'A linear neural network-based approach to stereophonic acoustic echo cancellation', Audio, Speech, and Language Processing pp. 1743 – 1753.

BERAN, P. P., VINEK, E., SCHIKUTA, E. & WEISHAUPL, T. (2008), Vinnsl-the vienna neural network specification language, in 'Neural Networks, 2008. IJCNN 2008.(IEEE World Congress on Computational Intelligence). IEEE International Joint Conference on', IEEE, pp. 1872–1879.

BLACK, M. J. (1986), 'The connection machine', AI Magazine **7**, 169.

CAO, Q., EWING, B. T. & THOMPSON, M. A. (2012), 'Forecasting wind speed with recurrent neural networks', European Journal of Operational Research pp. 148–154.

CHANDRA, B. & VARGHESE, P. P. (2007), 'Applications of cascade correlation neural networks for cipher system identification', International Journal of Computer, Information, Systems and Control Engineering pp. 350–353.

CHANDRALEKHA, B. & PRAFULLA, K. (2009), 'Use of adaptive resonance theory for vertical handoff decision in heterogeneous wireless environment', International Journal of Recent Trends in Engineering pp. 56–60.

CHANG, C.-Y., WANG, H.-J. & SHEN, W.-C. (2010), 'Copyright-proving scheme for audio with counter-propagation neural networks', Digital Signal Processing pp. 1087–1101.

CHE, Z. H. (2010), 'Pso-based back-propagation artificial neural network for product and mold cost estimation of plastic injection molding', Computers & Industrial Engineering pp. 625–637.

CHEN, C.-H., KHOO, L. P. & YAN, W. (2002), 'A strategy for acquiring customer requirement patterns using laddering technique and art2 neural network', Advanced Engineering Informatics pp. 229–240.

CHENG, W.-C., HUANG, J.-C. & LIOU, C.-Y. (2012), 'Segmentation of dna using simple recurrent neural network', Knowledge-Based Systems pp. 271–280.

CHON, T.-S. (2011), 'Self-organizing maps applied to ecological sciences', Ecological Informatics pp. 50–61.

CHUA, L. O. & Lin, Y. (1988), 'Cellular neural networks: Theory', IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS pp. 1257–1272.

COOMBS, J. H., RENEAR, A. H. & DeROSE, S. J. (1987), 'Markup systems and the future of scholarly text processing', Communications fo the ACM pp. 933–947.

Corporation, M. C. (no date), 'Maspar machines'.
URL: *http://home.wlu.edu/~whaleyt/classes/parallel/topics/maspar/maspar.html*

CREPUT, J.-C. & KOUKAM, A. (2009), 'A memetic neural network for the euclidean traveling salesman problem', Neurocomputing pp. 1250–1264.

DAS, P. & CHAUDHURY, S. (2007), 'Prediction of retail sales of footwear using feed-forward and recurrent neural networks', Neural Computing and Applications pp. 491–502.

DASH, T., NAYAK, T. & CHATTOPADHYAY, S. (2013), 'Offline verification of hand written signature using adaptive resonance theory net (type-1)', International Journal of Signal Processing Systems pp. 17–22.

De MULDER, W., BETHARD, S. & MOENS, M.-F. (2014), 'A survey on the application of recurrent neural networks to statistical language modeling', Computer Speech and Language .

de SOTO, A. R., CAPDEVILA, C. A. & FERNANDEZ, E. C. (2003), 'Fuzzy systems and neural networks xml schemas for soft computing', Mathware and Softcomputing pp. 43–56.

DIAMANTOPOULOU, M. J. (2006), 'Tree-bole volume estimation on standing pine trees using cascade correlation artificial neural network models', CIGR E-Journal pp. 1–6.

DRAGOMIR, O. E., DRAGOMIR, F. & RADULECU, M. (2014), 'Matlab application of kohonen self- organizing map to classify consumers' load profiles', Procedia Computer Science pp. 474–479.

FENG, Y., ZHANG, W., SUN, D. & ZHANG, L. (2011), 'Ozone concentration forecast method based on genetic algorithm optimized back propagation neural networks and support vector machine data classification', Atmospheric Environment pp. 1979–1985.

GIRI, P. K. & MOULICK, S. K. (2014), 'Application of neural network for cell formation in group technology', International Journal of Modern Engineering Research **7**, 1–5.

GONDARA, M. K. & KADAM, S. (2011), 'Requirements of vertical handoff mechanism in 4g wireless networks', International Journal of Wireless & Mobile Networks pp. 18–27.

GRAUPE, D. (1997), Principles of artificial neural networks, World Scientific Publishing Co. Pte. Ltd., Singapore.

HADAD, K. & PIROOZMAND, A. (2007), 'Application of cellular neural network (cnn) method to the nuclear reactor dynamics equations', Annals of nuclear energy pp. 406–416.

HANAFIZADEH, P., RAVASAN, A. Z. & KHAKI, H. R. (2010), 'An expert system for perfume selection using artificial neural network', Expert Systems with Applications pp. 8879–8887.

HAUN, M. (1998), Simulation Neuronaler Netze, Expert Verlag, Renningen-Malmsheim.

HODNETT, K. & HSIEH, H.-H. (2012), 'Application of cascade-correlation neural networks in developing stock selection models for global equities', International Business & Economics Research Journal pp. 375–396.

HUQQANI, A. A., SCHIKUTA, E., YE, S. & CHEN, P. (2013), 'Multicore and gpu parallelization of neural networks for face recognition', Procedia Computer Science **18**, 349–358.

Inmos (1987), 'Transputer architecture'.
URL: *http://www.transputer.net/fbooks/tarch/tarch.html*

JORDAN, M. I. (1990), Artificial neural networks, IEEE Press, Piscataway, chapter Attractor dynamics and parallelism in a connectionist sequential machine, pp. 112–127.

JUANG, S., TARNG, Y. & LII, H. (1998), 'A comparison between the back-propagation and counter-propagation networks in the modeling of the tig welding process', Journal of Materials Processing Technology pp. 54–62.

JUNG, S. (2008), GPU Data-parallel Computing of Sequence Alignment Using CUDA, ProQuest.

KAEFER, F., HEILMANN, C. M. & RAMENOFSKY, S. D. (2005), 'A neural network application to consumer classification to improve the timing of direct marketing activities', Computers & Operations Research pp. 2595–2615.

KHANDURI, A. & MORROW, G. (2003), 'Vulnerability of buildings to windstorms and insurance loss estimation', Journal of Wind Engineering and Industrial Aerodynamics pp. 455–467.

KIM, J., WEI, S. & RUYS, H. (2003), 'Segmenting the market of west australian senior tourists using an artificial neural network', Tourism Management pp. 25–34.

KONECNY, V., TRENZ, O. & DVORAKOVA, D. (2011), 'Evolution of insurance company service quality survey, using self-learning neural network', Acta univ. agric. et silvic. Mendel. Brun. pp. 149–154.

KOVALISHYN, V. V., TETKO, I. V., LUIK, A. I., KHOLODOVYCH, V. V., VILLA, A. E. P. & LIVINGSTONE, D. J. (1998), 'Neural network studies. 3. variable selection in the cascade-correlation learning architecture', Journal of Chemical Information and Computer Sciences pp. 651–659.

LEE, L.-T., LEE, S.-T. & CHEN, C.-W. (2010), Parallel programming on a soft-core based multi-core system, in 'Algorithms and Architectures for Parallel Processing', Springer Science & Business Media, Busan, pp. 22–31.

LEE, S. & CHOEH, J. Y. (2014), 'Predicting the helpfulness of online reviews using multilayer perceptron neural networks', Expert Systems with Applications pp. 3041–3046.

LIN, C. (2009), 'Using neural networks as a support tool in the decision making for insurance industry', Expert Systems with Applications pp. 6914–6917.

LIN, C.-M., HUANG, J.-J., GEN, M. & TZENG, G.-H. (2006), 'Recurrent neural network for dynamic portfolio selection', Applied Mathematics and Computation pp. 1139–1146.

LISA, l. (2010), 'Multilayer perceptron'.
URL: *http://deeplearning.net/tutorial/mlp.html*

LIU, C. (2010), Customer maturity model based on counter propagation network, in '2010 International Conference on Management and Service Science', IEEE, Wuhan, pp. 1–4.

LUGER, G. F. (2005), Artificial Intelligence: Structures and Strategies for Complex Problem Solving, 5. edn, Pearson Educated Limited, Essex.

MADHUSUDANAN, A. (2006), 'Nxml - introducing an xml based language to perform neural network processing, image analysis, pattern detection etc'.
URL: *http://www.codeproject.com/Articles/14387/NXML-Introducing-an-XML-Based-Language-To-Perform*

MAETSCHKE, S. R. & RAGAN, M. A. (2014), 'Characterizing cancer subtypes as attractors of hopfield networks', Bioinformatics pp. 1273–1279.

MAKHFI, P. (2001-2011), 'Nndef toolkit'.
URL: *http://www.makhfi.com/nndef.htm*

MALLESWARAN, M., VAIDEHI, V. & SIVASANKARI, N. (2014), 'A novel approach to the integration of gps and ins using recurrent neural networks with evolutionary optimization techniques', Aerospace Science and Technology pp. 169–179.

MANN, E. (2013), N2sky - a cloud-based neural network simulation environment, Master's thesis, University of Vienna.

MARQUES, A., LACERDA, D. P., CAMARGO, L. F. R. & TEIXEIRA, R. (2014), 'Exploring the relationship between marketing and operations: Neural network analysis of marketing decision impacts on delivery performance', Int. J. Production Economics pp. 178–190.

MATIGNON, R. (2005), Neural Network Modeling Using Sas Enterprise Miner, Authorhouse, s.l.

McCULLOCK, J. (2012), 'Elman networks'.
URL: *http://mnemstudio.org/neural-networks-elman.htm*

MEDSKER, L. & JAIN, L. (2000), Recurrent Neural Networks Design and Applications, CRC Press LLC, Boca Raton.

MEMON, A. P., MEMON, A. S., AKHUND, A. A. & MEMON, R. H. (2013), 'Multilayer perceptrons neural network automatic voltage regulator with applicability and improvement in power system transient stability', International Journal of Emerging Trends in Electrical and Electronics pp. 30–38.

NAMBA, M. & ZHANG, Z. (2006), Cellular neural network for associative memory and its application to braille image recognition, in 'Neural Networks, 2006. IJCNN'06. International Joint Conference on', IEEE, pp. 2409–2414.

NASSIF, A. B., CAPRETZ, L. F. & HO, D. (2012), Software effort estimation in the early stages of the software life cycle using a cascade correlation neural network model, in 'Software Engineering, Artificial Intelligence, Networking and Parallel & Distributed Computing (SNPD), 2012 13th ACIS International Conference on', IEEE, Kyoto, pp. 589–594.

Nature Education (2009), 'Scitable'.
URL: *http://www.nature.com/scitable/content/the-four-bases-atcg-6491969*

NAWI, N. M., GHAZALI, R. & SALLEH, M. N. M. (2010), The Development of Improved Back-Propagation Neural Networks Algorithm for Predicting Patients with Heart Disease, 6377 edn, Springer Berlin Heidelberg, Berlin, pp. 317–327.

NOGUCHI, S. & YOUKO, O. (2010), Improved kohonen feature map probabilistic associative memory based on weights distribution, in 'Neural Networks (IJCNN), The 2010 International Joint Conference on', IEEE, Barcelona, pp. 1–8.

OLSZEWSKI, D. (2014), 'Fraud detection using self-organizing map visualizing the user profiles', Knowledge-Based Systems pp. 324–334.

ORHAN, U., HEKIM, M. & OZER, M. (2011), 'Eeg signals classification using the k-means clustering and a multilayer perceptron neural network model', Expert Systems with Applications pp. 13475–13481.

PADUA, D. (2011), Encyclopedia of Parallel Computing, Vol. 4, Springer Science & Business Media.

PALIWAL, M. & KUMAR, U. A. (2009), 'Neural networks and statistical techniques: A review of applications', Expert Systems with Applications pp. 2–17.

PARK, J. H., KIM, J. H., EOM, I. K. & LEE, K. Y. (1993), 'Economic load dispatch for piecewise quadratic cost function using hopfield neural network', IEEE Transactions on Power Systems pp. 1030–1038.

PIETERSE, V. & BLACK, P. E. (2004), 'single program multiple data'.
URL: *http://xlinux.nist.gov/dads/HTML/singleprogrm.html*

PRABHU, G. (no date), 'Pipelining'.
URL: *http://www.cs.iastate.edu/~prabhu/Tutorial/PIPELINE/pipe_title.html*

RASHIDI, F. & RASHIDI, M. (2004), Emotional temporal difference learning based multi-layer perceptron neural network application to a prediction of solar activity, in 'Rough Sets and Current Trends in Computing', Springer, Uppsala, pp. 685–690.

Refsnes Data (1999-2014), 'Xml tree'.
URL: *http://www.w3.org/XML/*

Refsnes Data (1999-2015), 'Dtd tutorials'.
URL: *http://www.w3schools.com/dtd/*

ROJAS, R. (1996), Neural Networks, Springer-Verlag, Berlin.

RZEMPOLUCK, E. J. (1998), Neural Network Analysis Using Simulnet, Springer Verlag New York, Inc., New York.

SAHIN, Ü. A., BAYAT, C. & UCAN, O. N. (2011), 'Application of cellular neural network (cnn) to the prediction of missing air pollution data', Atmospheric Research pp. 314–326.

SAMMOUDA, R., ADGABA, N., TOUIR, A. & AL-GHAMDI, A. (2014), 'Agriculture satellite image segmentation using a modified artificial hopfield neural network', Computers in human behavior pp. 436–441.

SANTOS, C. E. M., RIBEIRO, E. P. & PEDROSO, C. M. (2014), 'The application of neural networks to improve the quality of experience of video transmission over ip networks', Engineering Applications of Artificial Intelligence pp. 137–147.

Schabauer, H., Schikuta, E. & Weishäupl, T. (2005), Solving very large traveling salesman problems by som parallelization on cluster architectures., in 'PDCAT', pp. 954–958.

SCHIKUTA, E. (1997), Structural data parallel neural network simulation, in 'Proceedings of 11th Annual International Symposium on High Performance Computing Systems (HPCS'97), Winnipeg, Canada'.

Schikuta, E. (2002), Neuroweb: an internet-based neural network simulator, in '2012 IEEE 24th International Conference on Tools with Artificial Intelligence', IEEE Computer Society, pp. 407–407.

SCHIKUTA, E., HUQQANI, A. & KOPICA, T. (2015), Semantic extensions to the vienna neural network specification language, in 'Neural Networks (IJCNN), The 2015 International Joint Conference on', IJCNN, Killarney.

Schikuta, E. & Weidmann, C. (1997), 'Data parallel simulation of self-organizing maps on hypercube architectures', Proceedings of WSOM **97**, 4–6.

Schikuta, E. & Weishäupl, T. (2004), N2grid: neural networks in the grid, in 'Neural Networks, 2004. Proceedings. 2004 IEEE International Joint Conference on', Vol. 2, IEEE, pp. 1409–1414.

SERBEDZIJA, N. (1996), 'Simulating artificial neural networks on parallel architectures', Computer **29**, 56–63.

SHITONG, W., DUAN, F., MIN, X. & DEWEN, H. (2007), 'Advanced fuzzy cellular neural network: Application to ct liver images', Artificial Intelligence in Medicine pp. 65–77.

SLAVOVA, A. & MLADENOV, V. (2004), Cellular Neural Networks: Theory and Applications, Nova Science Publishers Inc., New York.

SMITH, K. A. & GUPTA, J. N. (2000), 'Neural networks in business: techniques and applications for the operations researcher', Computers & Operations Research pp. 1023–1044.

STOJKOVIC, G., NOVIC, M. & KUZMANOVSKI, I. (2010), 'Counter-propagation artificial neural networks as a tool for prediction of pkbh+ for series of amides', Chemometrics and Intelligent Laboratory Systems pp. 123–129.

STRICKER, T. M. (2011), Hypercubes and Meshes, Vol. 4, Springer Science & Business Media, p. 2175.

SUBUDHI, B. N., GOSH, S. & GOSH, A. (2014), 'Application of gibbs-markov random field and hopfield-type neural networks for detecting moving objects from video sequences captured by static camera', Soft Computing .

Syncro Soft SRLE. (2002-2015), 'Oxygen xml editor'.
URL: *http://www.oxygenxml.com/*

W3C (1996-2003), 'Extensible markup language (xml)'.
URL: *http://www.w3.org/XML/*

WANG, J.-Z., WANG, J.-J., ZHANG, Z.-G. & GUO, S.-P. (2011), 'Forecasting stock indices with back propagation neural network', Expert Systems with Applications pp. 14346–14355.

WANG, X. (2013), 'The application research on discrete hopfield neural network in water quality evaluation', Applied Mechanics and Materials pp. 1338–1341.

Weishäupl, T. & Schikuta, E. (2003), Parallelization of cellular neural networks for image processing on cluster architectures, in 'Parallel Processing Workshops, 2003. Proceedings. 2003 International Conference on', IEEE, pp. 191–196.

ZELL, A. (1994), Simulation neuronaler Netze, 1. edn, Addison-Wesley GmbH, Bonn.

ZURADA, J. M. (1992), Introduction to Artificial Neural Systems, West Publishing Company, St. Paul.

A. Description Schema

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
    vc:minVersion="1.1" xmlns:vc="http://www.w3.org/2007/XMLSchema-versioning">
 <xs:simpleType name="propa">
  <xs:restriction base="xs:string">
   <xs:enumeration value="feedforward"/>
   <xs:enumeration value="feedback"/>
   <xs:enumeration value="recurrent"/>
  </xs:restriction>
 </xs:simpleType>
 <xs:simpleType name="minno1">
  <xs:restriction base="xs:integer">
   <xs:minExclusive value="0"/>
  </xs:restriction>
 </xs:simpleType>
 <xs:complexType name="parametervalue">
  <xs:choice minOccurs="0" maxOccurs="unbounded">
   <xs:element name="valueparameter">
    <xs:complexType>
     <xs:simpleContent>
      <xs:extension base="xs:decimal">
       <xs:attribute name="name" type="xs:string"/>
      </xs:extension>
     </xs:simpleContent>
    </xs:complexType>
   </xs:element>
   <xs:element name="boolparameter">
    <xs:complexType>
     <xs:simpleContent>
      <xs:extension base="xs:boolean">
       <xs:attribute name="name" type="xs:string"/>
      </xs:extension>
     </xs:simpleContent>
    </xs:complexType>
   </xs:element>
   <xs:element name="comboparameter">
    <xs:complexType>
     <xs:simpleContent>
      <xs:extension base="xs:string">
       <xs:attribute name="name" type="xs:string"/>
      </xs:extension>
     </xs:simpleContent>
    </xs:complexType>
   </xs:element>
  </xs:choice>
 </xs:complexType>

 <xs:element name="description">
  <xs:complexType>
   <xs:sequence>
    <xs:element name="identifier" type="xs:string"/>
    <xs:element name="metadata">
     <xs:complexType>
      <xs:sequence>
       <!--paradigm keywords-->
       <xs:element name="paradigm" type="xs:string"/>
       <!--network name given by the creator-->
       <xs:element name="name" type="xs:string"/>
       <xs:element name="description" type="xs:string"/>
       <xs:element name="version">
        <xs:complexType>
         <xs:sequence>
          <xs:element name="major" type="xs:integer"/>
          <xs:element name="minor" type="xs:integer"/>
         </xs:sequence>
        </xs:complexType>
       </xs:element>
      </xs:sequence>
     </xs:complexType>
    </xs:element>
    <xs:element name="creator">
     <xs:complexType>
      <xs:sequence>
       <xs:element name="name" type="xs:string"/>
       <xs:element name="contact" type="xs:string"/>
      </xs:sequence>
```

```
 </xs:complexType>
</xs:element>
<xs:element name="problemDomain">
 <xs:complexType>
  <xs:sequence>
   <xs:element name="propagationType">
    <xs:complexType>
     <xs:sequence>
      <xs:element name="learningType">
       <xs:simpleType>
        <xs:restriction base="xs:string">
         <xs:enumeration value="definedconstructed"/>
         <xs:enumeration value="trained"/>
         <xs:enumeration value="supervised"/>
         <xs:enumeration value="unsupervised"/>
         <xs:enumeration value="linear"/>
        </xs:restriction>
       </xs:simpleType>
      </xs:element>
     </xs:sequence>
     <xs:attribute name="type" type="propa"/>
    </xs:complexType>
   </xs:element>
   <xs:element name="applicationField" maxOccurs="unbounded">
    <xs:simpleType>
     <xs:union>
      <xs:simpleType>
       <xs:restriction base="xs:string">
        <xs:enumeration value="AccFin"/>
        <xs:enumeration value="HealthMed"/>
        <xs:enumeration value="Marketing"/>
        <xs:enumeration value="Retail"/>
        <xs:enumeration value="Insur"/>
        <xs:enumeration value="Telecom"/>
        <xs:enumeration value="Operations"/>
        <xs:enumeration value="EMS"/>
       </xs:restriction>
      </xs:simpleType>
      <xs:simpleType>
       <xs:restriction base="xs:string">
        <xs:pattern value="[A-Za-z]*"></xs:pattern>
       </xs:restriction>
      </xs:simpleType>
     </xs:union>
    </xs:simpleType>
   </xs:element>
   <xs:element name="networkType">
    <xs:simpleType>
     <xs:restriction base="xs:string">
      <xs:enumeration value="Hopfield"/>
      <xs:enumeration value="CNN"/>
      <xs:enumeration value="ART"/>
      <xs:enumeration value="Backpropagation"/>
      <xs:enumeration value="Cascade-Correlation"/>
      <xs:enumeration value="Kohonen"/>
      <xs:enumeration value="Counterpropagation"/>
      <xs:enumeration value="Perceptron"/>
      <xs:enumeration value="Linear-Associator"/>
      <xs:enumeration value="Jordan-Net"/>
      <xs:enumeration value="Elman-Net"/>
     </xs:restriction>
    </xs:simpleType>
   </xs:element>
   <xs:element name="problemType">
    <xs:simpleType>
     <xs:restriction base="xs:string">
      <xs:enumeration value="Classifiers"/>
      <xs:enumeration value="Approximators"/>
      <xs:enumeration value="Memory"/>
      <xs:enumeration value="Optimisation"/>
      <xs:enumeration value="Clustering"/>
     </xs:restriction>
    </xs:simpleType>
   </xs:element>
  </xs:sequence>
  <xs:assert
   test="((propagationType/@type = 'feedback' and propagationType/learningType = 'definedconstructed' and (ne
  />
 </xs:complexType>
</xs:element>
<xs:element name="endpoints">
 <xs:complexType>
  <xs:sequence>
   <xs:element name="train" type="xs:boolean"/>
   <xs:element name="retrain" type="xs:boolean"/>
   <xs:element name="evaluate" type="xs:boolean"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
<xs:element name="executionEnvironment" maxOccurs="unbounded">
```

```
<xs:complexType>
 <xs:sequence>
  <xs:element name="serial" type="xs:boolean" fixed="true"/>
  <xs:element name="parallel" minOccurs="0">
   <xs:complexType>
    <xs:sequence>
     <xs:element name="software">
      <xs:complexType>
       <xs:choice>
        <xs:element name="control">
         <xs:complexType>
          <xs:sequence>
           <xs:element name="transputer">
            <xs:complexType>
             <xs:simpleContent>
              <xs:extension base="xs:string">
               <xs:attribute name="version" type="xs:string" use="required"/>
              </xs:extension>
             </xs:simpleContent>
            </xs:complexType>
           </xs:element>
          </xs:sequence>
         </xs:complexType>
        </xs:element>
        <xs:element name="data">
         <xs:complexType>
          <xs:choice>
           <xs:element name="topological">
            <xs:complexType>
             <xs:choice>
              <xs:element name="pipelining">
               <xs:complexType>
                <xs:sequence>
                 <xs:element name="systolicarr">
                  <xs:complexType>
                   <xs:simpleContent>
                    <xs:extension base="xs:string">
                     <xs:attribute name="version" type="xs:string" use="required"/>
                    </xs:extension>
                   </xs:simpleContent>
                  </xs:complexType>
                 </xs:element>
                </xs:sequence>
               </xs:complexType>
              </xs:element>
              <xs:element name="coarsestruct">
               <xs:complexType>
                <xs:choice>
                 <xs:element name="connmachine">
                  <xs:complexType>
                   <xs:simpleContent>
                    <xs:extension base="xs:string">
                     <xs:attribute name="version" type="xs:string" use="required"/>
                    </xs:extension>
                   </xs:simpleContent>
                  </xs:complexType>
                 </xs:element>
                 <xs:element name="maspar">
                  <xs:complexType>
                   <xs:simpleContent>
                    <xs:extension base="xs:string">
                     <xs:attribute name="version" type="xs:string" use="required"/>
                    </xs:extension>
                   </xs:simpleContent>
                  </xs:complexType>
                 </xs:element>
                </xs:choice>
               </xs:complexType>
              </xs:element>
              <xs:element name="finestruct">
               <xs:complexType>
                <xs:sequence>
                 <xs:element name="connmachine">
                  <xs:complexType>
                   <xs:simpleContent>
                    <xs:extension base="xs:string">
                     <xs:attribute name="version" type="xs:string" use="required"/>
                    </xs:extension>
                   </xs:simpleContent>
                  </xs:complexType>
                 </xs:element>
                </xs:sequence>
               </xs:complexType>
              </xs:element>
             </xs:choice>
            </xs:complexType>
           </xs:element>
           <xs:element name="structural">
            <xs:complexType>
             <xs:sequence>
```

```
                    <xs:element name="spmd">
                     <xs:complexType>
                      <xs:choice>
                       <xs:element name="hypercube">
                        <xs:complexType>
                         <xs:simpleContent>
                          <xs:extension base="xs:string">
                           <xs:attribute name="version" type="xs:string" use="required"/>
                          </xs:extension>
                         </xs:simpleContent>
                        </xs:complexType>
                       </xs:element>
                       <xs:element name="cluster">
                        <xs:complexType>
                         <xs:simpleContent>
                          <xs:extension base="xs:string">
                           <xs:attribute name="version" type="xs:string" use="required"/>
                          </xs:extension>
                         </xs:simpleContent>
                        </xs:complexType>
                       </xs:element>
                       <xs:element name="gpgpu">
                        <xs:complexType>
                         <xs:simpleContent>
                          <xs:extension base="xs:string">
                           <xs:attribute name="version" type="xs:string" use="required"/>
                          </xs:extension>
                         </xs:simpleContent>
                        </xs:complexType>
                       </xs:element>
                       <xs:element name="multicore">
                        <xs:complexType>
                         <xs:simpleContent>
                          <xs:extension base="xs:string">
                           <xs:attribute name="version" type="xs:string" use="required"/>
                          </xs:extension>
                         </xs:simpleContent>
                        </xs:complexType>
                       </xs:element>
                      </xs:choice>
                     </xs:complexType>
                    </xs:element>
                   </xs:sequence>
                  </xs:complexType>
                 </xs:element>
                </xs:choice>
               </xs:complexType>
              </xs:element>
             </xs:choice>
            </xs:complexType>
           </xs:element>
           <xs:element name="hardware">
            <xs:complexType>
             <xs:choice>
              <xs:element name="general" type="parametervalue"/>
              <xs:element name="special" type="parametervalue"/>
             </xs:choice>
            </xs:complexType>
           </xs:element>
          </xs:sequence>
         </xs:complexType>
        </xs:element>
       </xs:sequence>
      </xs:complexType>
     </xs:element>
     <xs:element name="structure">
      <xs:complexType>
       <xs:sequence>
        <xs:element name="input">
         <xs:complexType>
          <xs:sequence>
           <xs:element name="ID" type="xs:string"/>
           <xs:element name="dimension">
            <xs:complexType>
             <xs:sequence>
              <xs:element name="min" type="minno1"/>
              <xs:element name="max" type="minno1"/>
             </xs:sequence>
            </xs:complexType>
           </xs:element>
           <xs:element name="size">
            <xs:complexType>
             <xs:sequence>
              <xs:element name="min" type="minno1"/>
              <xs:element name="max" type="minno1"/>
             </xs:sequence>
            </xs:complexType>
           </xs:element>
          </xs:sequence>
         </xs:complexType>
```

```
      </xs:element>
      <xs:element name="hidden" minOccurs="0" maxOccurs="unbounded">
       <xs:complexType>
        <xs:sequence>
         <xs:element name="ID" type="xs:string"/>
         <xs:element name="dimension">
          <xs:complexType>
           <xs:sequence>
            <xs:element name="min" type="minno1"/>
            <xs:element name="max" type="minno1"/>
           </xs:sequence>
          </xs:complexType>
         </xs:element>
         <xs:element name="size">
          <xs:complexType>
           <xs:sequence>
            <xs:element name="min" type="minno1"/>
            <xs:element name="max" type="minno1"/>
           </xs:sequence>
          </xs:complexType>
         </xs:element>
        </xs:sequence>
       </xs:complexType>
      </xs:element>
      <xs:element name="output" minOccurs="0">
       <xs:complexType>
        <xs:sequence>
         <xs:element name="ID" type="xs:string"/>
         <xs:element name="dimension">
          <xs:complexType>
           <xs:sequence>
            <xs:element name="min" type="minno1"/>
            <xs:element name="max" type="minno1"/>
           </xs:sequence>
          </xs:complexType>
         </xs:element>
         <xs:element name="size">
          <xs:complexType>
           <xs:sequence>
            <xs:element name="min" type="minno1"/>
            <xs:element name="max" type="minno1"/>
           </xs:sequence>
          </xs:complexType>
         </xs:element>
        </xs:sequence>
       </xs:complexType>
      </xs:element>
      <xs:element name="connections">
       <xs:simpleType>
        <xs:restriction base="xs:string">
         <xs:enumeration value="fullconnected"/>
         <xs:enumeration value="shortcuts"/>
         <xs:enumeration value="mixed"/>
        </xs:restriction>
       </xs:simpleType>
      </xs:element>
     </xs:sequence>
    </xs:complexType>
   </xs:element>
   <xs:element name="parameters">
    <xs:complexType>
     <xs:choice minOccurs="0" maxOccurs="unbounded">
      <xs:element name="valueparameter" type="xs:string" maxOccurs="unbounded"/>
      <xs:element name="boolparameter" type="xs:string" maxOccurs="unbounded"/>
      <xs:element name="comboparameter" type="xs:string" maxOccurs="unbounded"/>
     </xs:choice>
    </xs:complexType>
   </xs:element>
   <xs:element name="data">
    <xs:complexType>
     <xs:sequence>
      <xs:element name="description" type="xs:string"/>
      <xs:element name="tabledescription" type="xs:string" minOccurs="0"/>
      <xs:element name="filedescription" type="xs:string" minOccurs="0"/>
     </xs:sequence>
    </xs:complexType>
   </xs:element>
  </xs:sequence>
 </xs:complexType>
</xs:element>
</xs:schema>
```

B. Definition Schema

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
     vc:minVersion="1.1" xmlns:vc="http://www.w3.org/2007/XMLSchema-versioning">
 <xs:simpleType name="propa">
  <xs:restriction base="xs:string">
   <xs:enumeration value="feedforward"/>
   <xs:enumeration value="feedback"/>
   <xs:enumeration value="recurrent"/>
  </xs:restriction>
 </xs:simpleType>

 <xs:simpleType name="minno1">
  <xs:restriction base="xs:integer">
   <xs:minExclusive value="0"/>
  </xs:restriction>
 </xs:simpleType>

 <xs:complexType name="parametervalue">
  <xs:choice minOccurs="0" maxOccurs="unbounded">
   <xs:element name="valueparameter">
    <xs:complexType>
     <xs:simpleContent>
      <xs:extension base="xs:decimal">
       <xs:attribute name="name" type="xs:string"/>
      </xs:extension>
     </xs:simpleContent>
    </xs:complexType>
   </xs:element>
   <xs:element name="boolparameter">
    <xs:complexType>
     <xs:simpleContent>
      <xs:extension base="xs:boolean">
       <xs:attribute name="name" type="xs:string"/>
      </xs:extension>
     </xs:simpleContent>
    </xs:complexType>
   </xs:element>
   <xs:element name="comboparameter">
    <xs:complexType>
     <xs:simpleContent>
      <xs:extension base="xs:string">
       <xs:attribute name="name" type="xs:string"/>
      </xs:extension>
     </xs:simpleContent>
    </xs:complexType>
   </xs:element>
  </xs:choice>
 </xs:complexType>

 <xs:element name="definition">
  <xs:complexType>
   <xs:sequence>
    <xs:element name="identifier" type="xs:string"/>
    <xs:element name="problemDomain">
     <xs:complexType>
      <xs:sequence>
       <xs:element name="propagationType">
        <xs:complexType>
         <xs:sequence>
          <xs:element name="learningType">
           <xs:simpleType>
            <xs:restriction base="xs:string">
             <xs:enumeration value="definedconstructed"/>
             <xs:enumeration value="trained"/>
             <xs:enumeration value="supervised"/>
             <xs:enumeration value="unsupervised"/>
             <xs:enumeration value="linear"/>
            </xs:restriction>
           </xs:simpleType>
          </xs:element>
         </xs:sequence>
         <xs:attribute name="type" type="propa"/>
        </xs:complexType>
       </xs:element>
       <xs:element name="applicationField" maxOccurs="unbounded">
        <xs:simpleType>
         <xs:union>
```

```
          <xs:simpleType>
            <xs:restriction base="xs:string">
            <xs:enumeration value="AccFin"/>
            <xs:enumeration value="HealthMed"/>
            <xs:enumeration value="Marketing"/>
            <xs:enumeration value="Retail"/>
            <xs:enumeration value="Insur"/>
            <xs:enumeration value="Telecom"/>
            <xs:enumeration value="Operations"/>
            <xs:enumeration value="EMS"/>
           </xs:restriction>
          </xs:simpleType>
          <xs:simpleType>
           <xs:restriction base="xs:string">
            <xs:pattern value="[A-Za-z]*"></xs:pattern>
           </xs:restriction>
          </xs:simpleType>
         </xs:union>
        </xs:simpleType>
       </xs:element>
       <xs:element name="networkType">
        <xs:simpleType>
         <xs:restriction base="xs:string">
          <xs:enumeration value="Hopfield"/>
          <xs:enumeration value="CNN"/>
          <xs:enumeration value="ART"/>
          <xs:enumeration value="Backpropagation"/>
          <xs:enumeration value="Cascade-Correlation"/>
          <xs:enumeration value="Kohonen"/>
          <xs:enumeration value="Counterpropagation"/>
          <xs:enumeration value="Perceptron"/>
          <xs:enumeration value="Linear-Associator"/>
          <xs:enumeration value="Jordan-Net"/>
          <xs:enumeration value="Elman-Net"/>
         </xs:restriction>
        </xs:simpleType>
       </xs:element>
       <xs:element name="problemType">
        <xs:simpleType>
         <xs:restriction base="xs:string">
          <xs:enumeration value="Classifiers"/>
          <xs:enumeration value="Approximators"/>
          <xs:enumeration value="Memory"/>
          <xs:enumeration value="Optimisation"/>
          <xs:enumeration value="Clustering"/>
         </xs:restriction>
        </xs:simpleType>
       </xs:element>
      </xs:sequence>
      <xs:assert
       test="((propagationType/@type = 'feedback' and propagationType/learningType = 'definedconstructed ' and (ne
/>
     </xs:complexType>
    </xs:element>
    <xs:element name="endpoints">
     <xs:simpleType>
      <xs:restriction base="xs:string">
       <xs:enumeration value="train"/>
       <xs:enumeration value="retrain"/>
      </xs:restriction>
     </xs:simpleType>
    </xs:element>
    <xs:element name="executionEnvironment">
     <xs:complexType>
      <xs:sequence>
       <xs:element name="serial" type="xs:boolean" fixed="true"/>
       <xs:element name="parallel" minOccurs="0">
        <xs:complexType>
         <xs:sequence>
          <xs:element name="software">
           <xs:complexType>
            <xs:choice>
             <xs:element name="control">
              <xs:complexType>
               <xs:sequence>
                <xs:element name="transputer">
                 <xs:complexType>
                  <xs:simpleContent>
                   <xs:extension base="xs:string">
                    <xs:attribute name="version" type="xs:string" use="required"/>
                   </xs:extension>
                  </xs:simpleContent>
                 </xs:complexType>
                </xs:element>
               </xs:sequence>
              </xs:complexType>
             </xs:element>
             <xs:element name="data">
              <xs:complexType>
               <xs:choice>
```

```
<xs:element name="topological">
 <xs:complexType>
  <xs:choice>
   <xs:element name="pipelining">
    <xs:complexType>
     <xs:sequence>
      <xs:element name="systolicarr">
       <xs:complexType>
        <xs:simpleContent>
         <xs:extension base="xs:string">
          <xs:attribute name="version" type="xs:string" use="required"/>
         </xs:extension>
        </xs:simpleContent>
       </xs:complexType>
      </xs:element>
     </xs:sequence>
    </xs:complexType>
   </xs:element>
   <xs:element name="coarsestruct">
    <xs:complexType>
     <xs:choice>
      <xs:element name="connmachine">
       <xs:complexType>
        <xs:simpleContent>
         <xs:extension base="xs:string">
          <xs:attribute name="version" type="xs:string" use="required"/>
         </xs:extension>
        </xs:simpleContent>
       </xs:complexType>
      </xs:element>
      <xs:element name="maspar">
       <xs:complexType>
        <xs:simpleContent>
         <xs:extension base="xs:string">
          <xs:attribute name="version" type="xs:string" use="required"/>
         </xs:extension>
        </xs:simpleContent>
       </xs:complexType>
      </xs:element>
     </xs:choice>
    </xs:complexType>
   </xs:element>
   <xs:element name="finestruct">
    <xs:complexType>
     <xs:sequence>
      <xs:element name="connmachine">
       <xs:complexType>
        <xs:simpleContent>
         <xs:extension base="xs:string">
          <xs:attribute name="version" type="xs:string" use="required"/>
         </xs:extension>
        </xs:simpleContent>
       </xs:complexType>
      </xs:element>
     </xs:sequence>
    </xs:complexType>
   </xs:element>
  </xs:choice>
 </xs:complexType>
</xs:element>
<xs:element name="structural">
 <xs:complexType>
  <xs:sequence>
   <xs:element name="spmd">
    <xs:complexType>
     <xs:choice>
      <xs:element name="hypercube">
       <xs:complexType>
        <xs:simpleContent>
         <xs:extension base="xs:string">
          <xs:attribute name="version" type="xs:string" use="required"/>
         </xs:extension>
        </xs:simpleContent>
       </xs:complexType>
      </xs:element>
      <xs:element name="cluster">
       <xs:complexType>
        <xs:simpleContent>
         <xs:extension base="xs:string">
          <xs:attribute name="version" type="xs:string" use="required"/>
         </xs:extension>
        </xs:simpleContent>
       </xs:complexType>
      </xs:element>
      <xs:element name="gpgpu">
       <xs:complexType>
        <xs:simpleContent>
         <xs:extension base="xs:string">
          <xs:attribute name="version" type="xs:string" use="required"/>
         </xs:extension>
```

```
                    </xs:simpleContent>
                   </xs:complexType>
                  </xs:element>
                  <xs:element name="multicore">
                   <xs:complexType>
                    <xs:simpleContent>
                     <xs:extension base="xs:string">
                      <xs:attribute name="version" type="xs:string" use="required"/>
                     </xs:extension>
                    </xs:simpleContent>
                   </xs:complexType>
                  </xs:element>
                 </xs:choice>
                </xs:complexType>
               </xs:element>
              </xs:sequence>
             </xs:complexType>
            </xs:element>
           </xs:choice>
          </xs:complexType>
         </xs:element>
        </xs:choice>
       </xs:complexType>
      </xs:element>
      <xs:element name="hardware">
       <xs:complexType>
        <xs:choice>
         <xs:element name="general" type="parametervalue"/>
         <xs:element name="special" type="parametervalue"/>
        </xs:choice>
       </xs:complexType>
      </xs:element>
     </xs:sequence>
    </xs:complexType>
   </xs:element>
  </xs:sequence>
 </xs:complexType>
</xs:element>
<xs:element name="structure">
 <xs:complexType>
  <xs:sequence>
   <xs:element name="input">
    <xs:complexType>
     <xs:sequence>
      <xs:element name="ID" type="xs:string"/>
      <xs:element name="dimension" type="minno1"/>
      <xs:element name="size" type="␣minno1"/>
     </xs:sequence>
    </xs:complexType>
   </xs:element>
   <xs:element name="hidden" minOccurs="0" maxOccurs="unbounded">
    <xs:complexType>
     <xs:sequence>
      <xs:element name="ID" type="xs:string"/>
      <xs:element name="dimension" type="minno1"/>
      <xs:element name="size" type="minno1"/>
     </xs:sequence>
    </xs:complexType>
   </xs:element>
   <xs:element name="output" minOccurs="0">
    <xs:complexType>
     <xs:sequence>
      <xs:element name="ID" type="xs:string"/>
      <xs:element name="dimension" type="minno1"/>
      <xs:element name="size" type="minno1"/>
     </xs:sequence>
    </xs:complexType>
   </xs:element>
   <xs:element name="connections">
    <xs:complexType>
     <xs:choice maxOccurs="unbounded">
      <xs:element name="fullconnected" minOccurs="0">
       <xs:complexType>
        <xs:sequence maxOccurs="unbounded">
         <xs:element name="fromblock" type="xs:string"/>
         <xs:element name="toblock" type="xs:string"/>
        </xs:sequence>
       </xs:complexType>
      </xs:element>
      <xs:element name="shortcuts">
       <xs:complexType>
        <xs:sequence>
         <xs:element name="fromneuron" type="xs:string"/>
         <xs:element name="toneuron" type="xs:string"/>
        </xs:sequence>
       </xs:complexType>
      </xs:element>
     </xs:choice>
    </xs:complexType>
   </xs:element>
```

```
      </xs:sequence>
     </xs:complexType>
    </xs:element>
    <xs:element name="resultSchema">
     <xs:complexType>
      <xs:sequence>
       <xs:element name="instance" type="xs:boolean"/>
       <xs:element name="training" type="xs:boolean"/>
      </xs:sequence>
     </xs:complexType>
    </xs:element>
    <xs:element name="parameters" type="parametervalue"/>
    <xs:element name="data">
     <xs:complexType>
      <xs:sequence>
       <xs:element name="description" type="xs:string"/>
       <xs:element name="dataSchemaID" type="xs:string"/>
      </xs:sequence>
     </xs:complexType>
    </xs:element>
    <xs:element name="instanceSchemaID" type="xs:string" minOccurs="0"/>
   </xs:sequence>
   <xs:assert
    test="if(endpoints = 'retrain') then
    if(count(instanceSchemaID) > 0) then
    true()
    else
    false()
    else
    if(count(instanceSchemaID) > 0) then
    false()
    else true()"/>
  </xs:complexType>
 </xs:element>
</xs:schema>
```

C. Data Schema

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
    vc:minVersion="1.1" xmlns:vc="http://www.w3.org/2007/XMLSchema-versioning">
 <xs:element name="dataschema">
  <xs:complexType>
   <xs:sequence>
    <xs:element name="identifier" type="xs:string"/>
    <xs:element name="creationdate" type="xs:date"/>
    <xs:element name="data">
     <xs:complexType>
      <xs:sequence>
       <xs:element name="table" minOccurs="0">
        <xs:complexType>
         <xs:choice>
          <xs:sequence maxOccurs="unbounded">
           <xs:element name="netinput">
            <xs:simpleType>
             <xs:restriction base="xs:string">
              <xs:pattern value="(([0-9])*([.])?([0-9])*(\s)?)+"/>
              <xs:whiteSpace value="preserve"/>
             </xs:restriction>
            </xs:simpleType>
           </xs:element>
           <xs:element name="netoutput">
            <xs:simpleType>
             <xs:restriction base="xs:string">
              <xs:pattern value="(([0-9])*([.])?([0-9])*(\s)?)+"/>
              <xs:whiteSpace value="preserve"/>
             </xs:restriction>
            </xs:simpleType>
           </xs:element>
          </xs:sequence>
          <xs:sequence>
           <xs:element name="input" maxOccurs="unbounded">
            <xs:simpleType>
             <xs:restriction base="xs:string">
              <xs:pattern value="(([0-9])*([.])?([0-9])*(\s)?)+"/>
              <xs:whiteSpace value="preserve"/>
             </xs:restriction>
            </xs:simpleType>
           </xs:element>
          </xs:sequence>
         </xs:choice>
        </xs:complexType>
       </xs:element>
       <xs:element name="file" type="xs:string" minOccurs="0"/>
      </xs:sequence>
      <xs:assert test="(count(table) + count(file)) > 0"/>
     </xs:complexType>
    </xs:element>
   </xs:sequence>
  </xs:complexType>
 </xs:element>
</xs:schema>
```

D. Training Result Schema

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
    vc:minVersion="1.1" xmlns:vc="http://www.w3.org/2007/XMLSchema-versioning">
 <xs:simpleType name="propa">
  <xs:restriction base="xs:string">
   <xs:enumeration value="feedforward"/>
   <xs:enumeration value="feedback"/>
   <xs:enumeration value="recurrent"/>
  </xs:restriction>
 </xs:simpleType>

 <xs:complexType name="parametervalue">
  <xs:choice minOccurs="0" maxOccurs="unbounded">
   <xs:element name="valueparameter">
    <xs:complexType>
     <xs:simpleContent>
      <xs:extension base="xs:decimal">
       <xs:attribute name="name" type="xs:string"/>
      </xs:extension>
     </xs:simpleContent>
    </xs:complexType>
   </xs:element>
   <xs:element name="boolparameter">
    <xs:complexType>
     <xs:simpleContent>
      <xs:extension base="xs:boolean">
       <xs:attribute name="name" type="xs:string"/>
      </xs:extension>
     </xs:simpleContent>
    </xs:complexType>
   </xs:element>
   <xs:element name="comboparameter">
    <xs:complexType>
     <xs:simpleContent>
      <xs:extension base="xs:string">
       <xs:attribute name="name" type="xs:string"/>
      </xs:extension>
     </xs:simpleContent>
    </xs:complexType>
   </xs:element>
  </xs:choice>
 </xs:complexType>

 <xs:complexType name="parametervalueBias">
  <xs:choice minOccurs="0" maxOccurs="unbounded">
   <xs:element name="valueparameter">
    <xs:complexType>
     <xs:simpleContent>
      <xs:extension base="xs:decimal">
       <xs:attribute name="name" type="xs:string"/>
      </xs:extension>
     </xs:simpleContent>
    </xs:complexType>
   </xs:element>
  </xs:choice>
 </xs:complexType>

 <xs:element name="trainingresultschema">
  <xs:complexType>
   <xs:sequence>
    <xs:element name="identifier" type="xs:string"/>
    <xs:element name="creationdate" type="xs:date"/>
    <xs:element name="data">
     <xs:complexType>
      <xs:sequence>
       <xs:element name="table" minOccurs="0">
        <xs:complexType>
         <xs:sequence maxOccurs="unbounded">
          <xs:element name="netinput">
           <xs:simpleType>
            <xs:restriction base="xs:string">
             <xs:pattern value="(([0-9])*([.])?([0-9])*(\s)?)+"/>
             <xs:whiteSpace value="preserve"/>
            </xs:restriction>
           </xs:simpleType>
          </xs:element>
```

```
        <xs:element name="netoutput">
         <xs:simpleType>
          <xs:restriction base="xs:string">
           <xs:pattern value="(([0-9])*([.])?([0-9])*(\s)?)+"/>
           <xs:whiteSpace value="preserve"/>
          </xs:restriction>
         </xs:simpleType>
        </xs:element>
       </xs:sequence>
      </xs:complexType>
     </xs:element>
     <xs:element name="file" type="xs:string" minOccurs="0"/>
    </xs:sequence>
    <xs:assert test="(count(table) + count(file)) > 0"/>
   </xs:complexType>
  </xs:element>
  <xs:element name="weightmatrix" type="xs:string"/>
  <xs:element name="epochs" type="xs:integer"/>
  <xs:element name="meanerror" type="xs:decimal"/>
  <xs:element name="activationfunction" type="xs:string"/>
  <xs:element name="totalexecutiontime" type="xs:string"/>
  <xs:element name="epocherrorvalue" type="xs:string"/>
  <xs:element name="learningrate" type="xs:decimal"/>
  <xs:element name="momentum" type="xs:decimal"/>
  <xs:element name="threshold" type="xs:decimal"/>
  <xs:element name="bias" type="parametervalueBias" minOccurs="0" maxOccurs="unbounded"/>
  <xs:element name="executionEnvironment">
   <xs:complexType>
    <xs:sequence>
     <xs:element name="serial" type="xs:boolean" fixed="true"/>
     <xs:element name="parallel" minOccurs="0">
      <xs:complexType>
       <xs:sequence>
        <xs:element name="software">
         <xs:complexType>
          <xs:choice>
           <xs:element name="control">
            <xs:complexType>
             <xs:sequence>
              <xs:element name="transputer">
               <xs:complexType>
                <xs:simpleContent>
                 <xs:extension base="xs:string">
                  <xs:attribute name="version" type="xs:string" use="required"/>
                 </xs:extension>
                </xs:simpleContent>
               </xs:complexType>
              </xs:element>
             </xs:sequence>
            </xs:complexType>
           </xs:element>
           <xs:element name="data">
            <xs:complexType>
             <xs:choice>
              <xs:element name="topological">
               <xs:complexType>
                <xs:choice>
                 <xs:element name="pipelining">
                  <xs:complexType>
                   <xs:sequence>
                    <xs:element name="systolicarr">
                     <xs:complexType>
                      <xs:simpleContent>
                       <xs:extension base="xs:string">
                        <xs:attribute name="version" type="xs:string" use="required"/>
                       </xs:extension>
                      </xs:simpleContent>
                     </xs:complexType>
                    </xs:element>
                   </xs:sequence>
                  </xs:complexType>
                 </xs:element>
                 <xs:element name="coarsestruct">
                  <xs:complexType>
                   <xs:choice>
                    <xs:element name="connmachine">
                     <xs:complexType>
                      <xs:simpleContent>
                       <xs:extension base="xs:string">
                        <xs:attribute name="version" type="xs:string" use="required"/>
                       </xs:extension>
                      </xs:simpleContent>
                     </xs:complexType>
                    </xs:element>
                    <xs:element name="maspar">
                     <xs:complexType>
                      <xs:simpleContent>
                       <xs:extension base="xs:string">
                        <xs:attribute name="version" type="xs:string" use="required"/>
                       </xs:extension>
```

```
                </xs:simpleContent>
               </xs:complexType>
              </xs:element>
             </xs:choice>
            </xs:complexType>
           </xs:element>
           <xs:element name="finestruct">
            <xs:complexType>
             <xs:sequence>
              <xs:element name="connmachine">
               <xs:complexType>
                <xs:simpleContent>
                 <xs:extension base="xs:string">
                  <xs:attribute name="version" type="xs:string" use="required"/>
                 </xs:extension>
                </xs:simpleContent>
               </xs:complexType>
              </xs:element>
             </xs:sequence>
            </xs:complexType>
           </xs:element>
          </xs:choice>
         </xs:complexType>
        </xs:element>
        <xs:element name="structural">
         <xs:complexType>
          <xs:sequence>
           <xs:element name="spmd">
            <xs:complexType>
             <xs:choice>
              <xs:element name="hypercube">
               <xs:complexType>
                <xs:simpleContent>
                 <xs:extension base="xs:string">
                  <xs:attribute name="version" type="xs:string" use="required"/>
                 </xs:extension>
                </xs:simpleContent>
               </xs:complexType>
              </xs:element>
              <xs:element name="cluster">
               <xs:complexType>
                <xs:simpleContent>
                 <xs:extension base="xs:string">
                  <xs:attribute name="version" type="xs:string" use="required"/>
                 </xs:extension>
                </xs:simpleContent>
               </xs:complexType>
              </xs:element>
              <xs:element name="gpgpu">
               <xs:complexType>
                <xs:simpleContent>
                 <xs:extension base="xs:string">
                  <xs:attribute name="version" type="xs:string" use="required"/>
                 </xs:extension>
                </xs:simpleContent>
               </xs:complexType>
              </xs:element>
              <xs:element name="multicore">
               <xs:complexType>
                <xs:simpleContent>
                 <xs:extension base="xs:string">
                  <xs:attribute name="version" type="xs:string" use="required"/>
                 </xs:extension>
                </xs:simpleContent>
               </xs:complexType>
              </xs:element>
             </xs:choice>
            </xs:complexType>
           </xs:element>
          </xs:sequence>
         </xs:complexType>
        </xs:element>
       </xs:choice>
      </xs:complexType>

     </xs:element>
    </xs:choice>
   </xs:complexType>
  </xs:element>
  <xs:element name="hardware">
   <xs:complexType>
    <xs:choice>
     <xs:element name="general" type="parametervalue"/>
     <xs:element name="special" type="parametervalue"/>
    </xs:choice>
   </xs:complexType>
  </xs:element>
 </xs:sequence>
</xs:complexType>
</xs:element>
```

```
      </xs:sequence>
     </xs:complexType>
    </xs:element>
    <xs:element name="propagationType">
     <xs:complexType>
      <xs:sequence>
       <xs:element name="learningType">
        <xs:simpleType>
         <xs:restriction base="xs:string">
          <xs:enumeration value="definedconstructed"/>
          <xs:enumeration value="trained"/>
          <xs:enumeration value="supervised"/>
          <xs:enumeration value="unsupervised"/>
          <xs:enumeration value="linear"/>
         </xs:restriction>
        </xs:simpleType>
       </xs:element>
      </xs:sequence>
      <xs:attribute name="type" type="propa"/>
     </xs:complexType>
    </xs:element>
    <xs:element name="networkType">
     <xs:simpleType>
      <xs:restriction base="xs:string">
       <xs:enumeration value="Hopfield"/>
       <xs:enumeration value="CNN"/>
       <xs:enumeration value="ART"/>
       <xs:enumeration value="Backpropagation"/>
       <xs:enumeration value="Cascade-Correlation"/>
       <xs:enumeration value="Kohonen"/>
       <xs:enumeration value="Counterpropagation"/>
       <xs:enumeration value="Perceptron"/>
       <xs:enumeration value="Linear-Associator"/>
       <xs:enumeration value="Jordan-Net"/>
       <xs:enumeration value="Elman-Net"/>
      </xs:restriction>
     </xs:simpleType>
    </xs:element>
   </xs:sequence>
   <xs:assert
    test="((propagationType/@type = 'feedback' and propagationType/learningType = 'definedconstructed' and (netwo
  </xs:complexType>
 </xs:element>
</xs:schema>
```

E. Instance Schema

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
    vc:minVersion="1.1" xmlns:vc="http://www.w3.org/2007/XMLSchema-versioning">
 <xs:simpleType name="propa">
  <xs:restriction base="xs:string">
   <xs:enumeration value="feedforward"/>
   <xs:enumeration value="feedback"/>
   <xs:enumeration value="recurrent"/>
  </xs:restriction>
 </xs:simpleType>

 <xs:complexType name="parametervalue">
  <xs:choice minOccurs="0" maxOccurs="unbounded">
   <xs:element name="valueparameter">
    <xs:complexType>
     <xs:simpleContent>
      <xs:extension base="xs:decimal">
       <xs:attribute name="name" type="xs:string"/>
      </xs:extension>
     </xs:simpleContent>
    </xs:complexType>
   </xs:element>
   <xs:element name="boolparameter">
    <xs:complexType>
     <xs:simpleContent>
      <xs:extension base="xs:boolean">
       <xs:attribute name="name" type="xs:string"/>
      </xs:extension>
     </xs:simpleContent>
    </xs:complexType>
   </xs:element>
   <xs:element name="comboparameter">
    <xs:complexType>
     <xs:simpleContent>
      <xs:extension base="xs:string">
       <xs:attribute name="name" type="xs:string"/>
      </xs:extension>
     </xs:simpleContent>
    </xs:complexType>
   </xs:element>
  </xs:choice>
 </xs:complexType>

 <xs:element name="instanceschema">
  <xs:complexType>
   <xs:sequence>
    <xs:element name="identifier" type="xs:string"/>
    <xs:element name="creationdate" type="xs:date"/>
    <xs:element name="structure">
     <xs:complexType>
      <xs:sequence>
       <xs:element name="input">
        <xs:complexType>
         <xs:sequence>
          <xs:element name="ID" type="xs:string"/>
          <xs:element name="dimension" type="xs:integer"/>
          <xs:element name="size" type="xs:integer"/>
         </xs:sequence>
        </xs:complexType>
       </xs:element>
       <xs:element name="hidden" minOccurs="0">
        <xs:complexType>
         <xs:sequence>
          <xs:element name="ID" type="xs:string"/>
          <xs:element name="dimension" type="xs:integer"/>
          <xs:element name="size" type="xs:integer"/>
         </xs:sequence>
        </xs:complexType>
       </xs:element>
       <xs:element name="output" minOccurs="0">
        <xs:complexType>
         <xs:sequence>
          <xs:element name="ID" type="xs:string"/>
          <xs:element name="dimension" type="xs:integer"/>
          <xs:element name="size" type="xs:integer"/>
         </xs:sequence>
        </xs:complexType>
```

```
    </xs:element>
    <xs:element name="connections">
     <xs:complexType>
      <xs:choice maxOccurs="unbounded">
       <xs:element name="fullconnected" minOccurs="0">
        <xs:complexType>
         <xs:sequence maxOccurs="unbounded">
          <xs:element name="fromblock" type="xs:string"/>
          <xs:element name="toblock" type="xs:string"/>
         </xs:sequence>
        </xs:complexType>
       </xs:element>
       <xs:element name="shortcuts" minOccurs="0">
        <xs:complexType>
         <xs:sequence maxOccurs="unbounded">
          <xs:element name="fromneuron" type="xs:string"/>
          <xs:element name="toneuron" type="xs:string"/>
         </xs:sequence>
        </xs:complexType>
       </xs:element>
      </xs:choice>
     </xs:complexType>
    </xs:element>
   </xs:sequence>
  </xs:complexType>
 </xs:element>
 <xs:element name="executionEnvironment">
  <xs:complexType>
   <xs:sequence>
    <xs:element name="serial" type="xs:boolean" fixed="true"/>
    <xs:element name="parallel" minOccurs="0">
     <xs:complexType>
      <xs:sequence>
       <xs:element name="software">
        <xs:complexType>
         <xs:choice>
          <xs:element name="control">
           <xs:complexType>
            <xs:sequence>
             <xs:element name="transputer">
              <xs:complexType>
               <xs:simpleContent>
                <xs:extension base="xs:string">
                 <xs:attribute name="version" type="xs:string" use="required"/>
                </xs:extension>
               </xs:simpleContent>
              </xs:complexType>
             </xs:element>
            </xs:sequence>
           </xs:complexType>
          </xs:element>
          <xs:element name="data">
           <xs:complexType>
            <xs:choice>
             <xs:element name="topological">
              <xs:complexType>
               <xs:choice>
                <xs:element name="pipelining">
                 <xs:complexType>
                  <xs:sequence>
                   <xs:element name="systolicarr">
                    <xs:complexType>
                     <xs:simpleContent>
                      <xs:extension base="xs:string">
                       <xs:attribute name="version" type="xs:string" use="required"/>
                      </xs:extension>
                     </xs:simpleContent>
                    </xs:complexType>
                   </xs:element>
                  </xs:sequence>
                 </xs:complexType>
                </xs:element>
                <xs:element name="coarsestruct">
                 <xs:complexType>
                  <xs:choice>
                   <xs:element name="connmachine">
                    <xs:complexType>
                     <xs:simpleContent>
                      <xs:extension base="xs:string">
                       <xs:attribute name="version" type="xs:string" use="required"/>
                      </xs:extension>
                     </xs:simpleContent>
                    </xs:complexType>
                   </xs:element>
                   <xs:element name="maspar">
                    <xs:complexType>
                     <xs:simpleContent>
                      <xs:extension base="xs:string">
                       <xs:attribute name="version" type="xs:string" use="required"/>
                      </xs:extension>
```

```
                </xs:simpleContent>
               </xs:complexType>
              </xs:element>
             </xs:choice>
            </xs:complexType>
           </xs:element>
           <xs:element name="finestruct">
            <xs:complexType>
             <xs:sequence>
              <xs:element name="connmachine">
               <xs:complexType>
                <xs:simpleContent>
                 <xs:extension base="xs:string">
                  <xs:attribute name="version" type="xs:string" use="required"/>
                 </xs:extension>
                </xs:simpleContent>
               </xs:complexType>
              </xs:element>
             </xs:sequence>
            </xs:complexType>
           </xs:element>
          </xs:choice>
         </xs:complexType>
        </xs:element>
        <xs:element name="structural">
         <xs:complexType>
          <xs:sequence>
           <xs:element name="spmd">
            <xs:complexType>
             <xs:choice>
              <xs:element name="hypercube">
               <xs:complexType>
                <xs:simpleContent>
                 <xs:extension base="xs:string">
                  <xs:attribute name="version" type="xs:string" use="required"/>
                 </xs:extension>
                </xs:simpleContent>
               </xs:complexType>
              </xs:element>
              <xs:element name="cluster">
               <xs:complexType>
                <xs:simpleContent>
                 <xs:extension base="xs:string">
                  <xs:attribute name="version" type="xs:string" use="required"/>
                 </xs:extension>
                </xs:simpleContent>
               </xs:complexType>
              </xs:element>
              <xs:element name="gpgpu">
               <xs:complexType>
                <xs:simpleContent>
                 <xs:extension base="xs:string">
                  <xs:attribute name="version" type="xs:string" use="required"/>
                 </xs:extension>
                </xs:simpleContent>
               </xs:complexType>
              </xs:element>
              <xs:element name="multicore">
               <xs:complexType>
                <xs:simpleContent>
                 <xs:extension base="xs:string">
                  <xs:attribute name="version" type="xs:string" use="required"/>
                 </xs:extension>
                </xs:simpleContent>
               </xs:complexType>
              </xs:element>
             </xs:choice>
            </xs:complexType>
           </xs:element>
          </xs:sequence>
         </xs:complexType>
        </xs:element>
       </xs:choice>
      </xs:complexType>

     </xs:element>
    </xs:choice>
   </xs:complexType>
  </xs:element>
  <xs:element name="hardware">
   <xs:complexType>
    <xs:choice>
     <xs:element name="general" type="parametervalue"/>
     <xs:element name="special" type="parametervalue"/>
    </xs:choice>
   </xs:complexType>
  </xs:element>
 </xs:sequence>
</xs:complexType>
</xs:element>
```

```
      </xs:sequence>
     </xs:complexType>
    </xs:element>
    <xs:element name="problemDomain">
     <xs:complexType>
      <xs:sequence>
       <xs:element name="networkType">
        <xs:simpleType>
         <xs:restriction base="xs:string">
          <xs:enumeration value="Hopfield"/>
          <xs:enumeration value="CNN"/>
          <xs:enumeration value="ART"/>
          <xs:enumeration value="Backpropagation"/>
          <xs:enumeration value="Cascade-Correlation"/>
          <xs:enumeration value="Kohonen"/>
          <xs:enumeration value="Counterpropagation"/>
          <xs:enumeration value="Perceptron"/>
          <xs:enumeration value="Linear-Associator"/>
          <xs:enumeration value="Jordan-Net"/>
          <xs:enumeration value="Elman-Net"/>
         </xs:restriction>
        </xs:simpleType>
       </xs:element>
       <xs:element name="propagationType">
        <xs:complexType>
         <xs:sequence>
          <xs:element name="learningType">
           <xs:simpleType>
            <xs:restriction base="xs:string">
             <xs:enumeration value="definedconstructed"/>
             <xs:enumeration value="trained"/>
             <xs:enumeration value="supervised"/>
             <xs:enumeration value="unsupervised"/>
             <xs:enumeration value="linear"/>
            </xs:restriction>
           </xs:simpleType>
          </xs:element>
         </xs:sequence>
         <xs:attribute name="type" type="propa"/>
        </xs:complexType>
       </xs:element>
       <xs:element name="applicationField" maxOccurs="unbounded">
        <xs:simpleType>
         <xs:union>
          <xs:simpleType>
           <xs:restriction base="xs:string">
            <xs:enumeration value="AccFin"/>
            <xs:enumeration value="HealthMed"/>
            <xs:enumeration value="Marketing"/>
            <xs:enumeration value="Retail"/>
            <xs:enumeration value="Insur"/>
            <xs:enumeration value="Telecom"/>
            <xs:enumeration value="Operations"/>
            <xs:enumeration value="EMS"/>
           </xs:restriction>
          </xs:simpleType>
          <xs:simpleType>
           <xs:restriction base="xs:string">
            <xs:pattern value="[A-Za-z]*"></xs:pattern>
           </xs:restriction>
          </xs:simpleType>
         </xs:union>
        </xs:simpleType>
       </xs:element>
       <xs:element name="problemType">
        <xs:simpleType>
         <xs:restriction base="xs:string">
          <xs:enumeration value="Classifiers"/>
          <xs:enumeration value="Approximators"/>
          <xs:enumeration value="Memory"/>
          <xs:enumeration value="Optimisation"/>
          <xs:enumeration value="Clustering"/>
         </xs:restriction>
        </xs:simpleType>
       </xs:element>
      </xs:sequence>
      <xs:assert
       test="((propagationType/@type = 'feedback' and propagationType/learningType = 'definedconstructed' and (ne
       />                    </xs:complexType>
    </xs:element>
    <xs:element name="weightmatrix" type="xs:string"/>
    <xs:element name="activationfunction" type="xs:string"/>
    <xs:element name="dataSchemaID" type="xs:string" minOccurs="0"/>
   </xs:sequence>
  </xs:complexType>
 </xs:element>
</xs:schema>
```

F. Result Schema

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
    vc:minVersion="1.1" xmlns:vc="http://www.w3.org/2007/XMLSchema-versioning">
 <xs:element name="resultschema">
  <xs:complexType>
   <xs:sequence>
    <xs:element name="identifier" type="xs:string"/>
    <xs:element name="instanceSchemaID" type="xs:string"/>
    <xs:element name="creationdate" type="xs:date"/>
    <xs:element name="diagram2d" minOccurs="0">
     <xs:complexType>
      <xs:sequence>
       <xs:element name="title" type="xs:string"/>
       <xs:element name="description" type="xs:string"/>
       <xs:element name="type" type="xs:string"/>
       <xs:element name="xaxis">
        <xs:complexType>
         <xs:sequence>
          <xs:element name="title" type="xs:string"/>
          <xs:element name="description" type="xs:string"/>
          <xs:element name="min" type="xs:integer"/>
          <xs:element name="max" type="xs:integer"/>
         </xs:sequence>
        </xs:complexType>
       </xs:element>
       <xs:element name="yaxis">
        <xs:complexType>
         <xs:sequence>
          <xs:element name="title" type="xs:string"/>
          <xs:element name="description" type="xs:string"/>
          <xs:element name="min" type="xs:integer"/>
          <xs:element name="max" type="xs:integer"/>
         </xs:sequence>
        </xs:complexType>
       </xs:element>
       <xs:element name="values">
        <xs:complexType>
         <xs:sequence>
          <xs:element name="value" maxOccurs="unbounded">
           <xs:complexType>
            <xs:sequence>
             <xs:element name="xvalue" type="xs:decimal"/>
             <xs:element name="yvalue" type="xs:decimal"/>
            </xs:sequence>
           </xs:complexType>
          </xs:element>
         </xs:sequence>
        </xs:complexType>
       </xs:element>
      </xs:sequence>
     </xs:complexType>
    </xs:element>
    <xs:element name="table" minOccurs="0">
     <xs:complexType>
      <xs:sequence minOccurs="0" maxOccurs="unbounded">
       <xs:element name="input">
        <xs:simpleType>
         <xs:restriction base="xs:string">
          <xs:pattern value="(([0-9])*([.])?([0-9])*(\s)?)+"/>
          <xs:whiteSpace value="preserve"/>
         </xs:restriction>
        </xs:simpleType>
       </xs:element>
       <xs:element name="output">
        <xs:simpleType>
         <xs:restriction base="xs:string">
          <xs:pattern value="(([0-9])*([.])?([0-9])*(\s)?)+"/>
          <xs:whiteSpace value="preserve"/>
         </xs:restriction>
        </xs:simpleType>
       </xs:element>
      </xs:sequence>
     </xs:complexType>
    </xs:element>
    <xs:element name="file" type="xs:string" minOccurs="0"/>
   </xs:sequence>
   <xs:assert test="(count(table) + count(file)) > 0"/>
```

```
    </xs:complexType>
  </xs:element>
</xs:schema>
```

G. Summary - German

Das Ziel dieser Arbeit war die Weiterentwicklung der Strukturspache ViNNSL, um neurale Netze beschreiben und ausführen zu können.

Zu Beginn wird eine Einführung in das Themengebiet der neuralen Netze, sowie einer Klassifizierung anhand der Eigenschaften vorgenommen. Im Anschluss werden die Problemdomänen neuraler Netze vorgestellt und praktische Beispiele für Anwendungen dargeboten. Nachfolgend wird der aktuelle Status von ViNNSL erläutert und mit anderen Strukturspachen, zur Beschreibung neuraler Netze, verglichen um Unterschiede herauszufiltern.

Auf Basis dieser Analyse wird ViNNSL auf ViNNSL 2.0 erweitert. Diese Erweiterungen führen zur Adaptierung bestehender und die Vorstellung neuer Schemas. Die Änderungen umfassen die Möglichkeit, mithilfe von ViNNSL 2.0, neurale Netze im Hinblick auf ihre Klassifizierung, Problemdomäne, fachliches Anwendungsgebiet einzuteilen und die technische Umsetzung zu definieren. Darüber hinaus werden zwei neue Schemas eingeführt. Diese ermöglichen die Analyse des Netzwerktrainings, sowie ein bereits trainiertes Netz neu instanzieren zu lassen. Mit dem zweiten Schema kann ein Netz, ohne Training, ausgeführt werden. ViNNSL 2.0 ist, wie der Vorgänger, eine XML basierte Sprache. Aus diesem Grund sind die einzelnen Schemas nach XSD 1.1 definiert. Dieser Standard ermöglicht die Definition von Regeln, sodass sichergestellt wird, dass die Schemas gültige und vollständige Informationen enthalten. Diese Arbeit enthält daher eine Visualisierung und genaue Beschreibung der einzelnen Komponenten jedes Schemas. Zusätzlich wird die praktische Verwendung mit zwei Fallstudien verdeutlicht.

ViNNSL 2.0 dient in N2Sky als standardisiertes Werkzeug zur Beschreibung der neuralen Netze.

H. Curriculum Vitae

Personal Data

Name:	Thomas Kopica

Education and Training

since March 2012	Master Wirtschaftsinformatik University of Vienna
September 2011 - February 2013	Master Information Systems Management FH Technikum Wien / Kharkiv National University of Economics
August 2012 - September 2012	Summer School China Know-How Peking University / Fudan University
September 2008 - June 2011	Bachelor Wirtschaftsinformatik FH Technikum Wien