A Rough-Neural Approach to Classifier Networks

Marcin Szczuka[1] and Jakub Wróblewski[2]

[1] Institute of Mathematics, Warsaw University
Banacha 2, 02-097 Warsaw, Poland
[2] Polish-Japanese Institute of Information Technology
Koszykowa 86, 02-008 Warsaw, Poland
szczuka@mimuw.edu.pl, jakubw@pjwstk.edu.pl

Abstract. We discuss the notion of hierarchical concept (classifier) schemes. Processes of construction, tuning and learning of hierarchical structures of concepts (granules of knowledge) are presented. The proposed solution consists of a generalised structure of feedforward neural-like network approximating the intermediate concepts similarly to traditional neurocomputing approaches. Fundamental works are also supported by a more practical part where implementation and experimental verification of presented ideas is discussed. We provide the examples of compound concepts corresponding to the Bayesian and rule based classifiers, and show some intuition concerning their processing through the network.

1 Introduction

The question: "What is neurocomputing?" may be answered in many ways, depending on context and the interests of the answering person. In this paper we would like to bring to the table yet another way of viewing the idea of neurocomputing. We attempt to define a framework for construction of classification support and concept approximation systems that are displaying features commonly attributed to the realm of neural computing. These are:

- Construction of a system capable of performing complex tasks using (relatively) simple computing units (neurons).
- Hierarchical structure that represents gradual formation of more complex entities represented by network's output from simpler building blocks (inputs and hidden units).
- Flexibility and robustness originating in highly adjustable structure of units and connections.
- Ability to learn from examples the desired setting of network parameters (weights).

The ability to adaptively learn the correct set of weights (network parameters) is in the centre of our focus. The hierarchical neural-like structure for concept approximation and/or classification is soulless if there is no efficient and robust way of adjusting it so that it does what it is supposed to.

In this paper we attempt to show our view on the process of construction and tuning of hierarchical structures of concepts (which can be also referred to as granules of knowledge [14, 16, 17]). We address these structures as *feedforward concept networks*,

which can be regarded as a special case of hierarchical structures developed within the *rough-neural computing* (RNC) methodology [9, 11, 12]. In particular, we consider *classifier networks*, where the input concepts correspond to the classified objects' behaviour with respect to the standard classifiers and the output (target) concept reflects the final classification. The basic idea is that the relationship between such input and output concepts is not direct but based on the internal layers of intermediate elements, which help in more reliable transition from the basic information to possibly compound classification goal. In this sense we create a neurocomputing scheme that uses concepts as neurons.

The proposed scheme is formalised with use of analogies rooted in areas, such as *artificial neural networks* [4, 5], *ensembles of classifiers* [2, 15, 27], and *layered learning* [24]. We show how the presented ideas can be exploited within wider frameworks of rough-neural and granular computing. We also make an effort to provide examples of actual models outlined in our earlier, application-oriented papers. This paper is – to large extent – a summarization and discussion of ideas already presented in our previous publications [20, 22, 23].

The paper starts with short section that presents motivation (section 2) behind undertaking this kind of investigations. Then, in section 3 we introduce general ideas that drive the construction of compound, multilayer concept schemes. These ideas are further extended and formalised in sections 4 and 5. The proposed general framework is then illustrated with an example of real scheme called Normalising Neural Network that implements a multilayer structure for Bayesian classifiers (section 7). After presenting this example we return to more general considerations (section 8) staring with discussion of possible use of proposed scheme in case of decision rule based classifiers and developing this into some general ideas regarding construction and learning in more general feedforward concept networks (subsections 8.1 and 8.2).

2 Motivation

If we take a look at the standard approach to classification and decision support with use of learning systems, we quickly realize that it does not always fit the purpose. Equipped with the hypothesis formation (learning) algorithm, we attempt to find a possibly direct mapping from the input values to decisions. Such an approach does not always result in success, because of various reasons.

First and foremost, the mapping from input to decision may not be learnable. The amount of data we possess is insufficient to form any viable decision-making solution that works in a single step. The target concept is by nature more complex and have to be decomposed into smaller blocks, sub-concepts, that are possible to learn.

We try to address the situation when the desired solution should be more fine-grained, namely, it should have an internal structure. Although possibly hard to find and learn, such architecture repays us by providing significant extensions in terms of flexibility, generality and expressiveness of the yielded model.

Another value-added effect we want to get from proposed hierarchical solution is the ability to extend and adapt the classification system once new data arrives, without the need for complete reconstruction of existing model.

3 Hierarchical learning and classification

In this paper we will be using the notion of concept and classifier exchangeably. In our understanding the classifier is a tool for describing the concept (of a decision class). Reversely, for a concept we may want to construct a classifier, i.e. an algorithm that helps us in deciding whether a given observation (data point) is a positive or negative example for this concept.

Let us start by explaining how we intend to treat the notion of a *concept*. In general, a concept is an element drawn from a *parameterised concept space*. By a proper setting of these parameters we choose the right concept. Note, that we do not initially demand that all concepts come from the same space. Essentially, a concept is a (sub)set but, due to the properties of space it comes from, it may display some special features, possess internal structure and so on.

Such an informal definition of a concept space can be referred to the notion of an *information granule system* $S = (G, R, Sem)$, where G is a set of parameterised formulas called *information granules*, R is a (parameterised) relation structure, and Sem is the semantics of G in R (cf. [17]). In our approach, we focus especially on the concept parameterisation and ability of parameterised construction of new concepts from the others. In this sense, our understanding of a concept space can be regarded as equivalent to information granule system and the terms *concept* and *granule* can be treated as complementary.

One should remember that that the whole scheme we want to bring here is aimed at construction of knowledge representation and classification system from experimental data. Therefore, we have to recall some of the basics regarding the task of learning how to classify (describe) concepts.

In the typical task of classification (cf. [7]) we are given a set of examples (training sample) T drawn from some universe X. We assume that every example $u \in X$ is represented by a vector of attribute (feature, measurement) values $a_1(u), ..., a_n(u)$, where $a_i : X \to A_i$, $i = 1, ..., n$. The set A_i is referred to as the *attribute value space*. The examples are also labeled with the value of decision d, treated as an additional attribute. We denote by $C_k \subset X$ the k-th decision class, i.e., the subset of examples labelled with decision value $k \in \{1, ..., r\}$.

Our goal in the classification problem is to find with some algorithm a hypothesis $h : X \to \{1, ..., r\}$, i.e. a mapping from X onto the set of decision values. Using the convention of this paper we may say that in classification the decision classes $C_1, \ldots, C_r$ are the concepts we attempt to describe by providing h. Mapping h is often assumed to be highly consistent with the training sample T. In other words, one expects that $d(u)$ should be similar to $h(u)$ for $u \in T$. Mapping h should be also – what is far more important – inductively correct, which means that it should be properly applicable for new, not labeled examples. Consistency with the training data hardly provides the inductive correctness. It is often better to base on less accurate, but less complex models (cf. [8]).

Now, let a concept represent an element acting on the basis of information originating from other concepts or directly from the data source. To better depict the whole structure, it is convenient to exploit the analogy with artificial neural networks. In this case, a concept corresponds to a signal transmitted through a neuron – the basic com-

puting unit. Dependencies between concepts, their precedence and importance, are represented by weighted connections between nodes. Similarly to the feedforward neural network, operations can be performed from bottom to top. They can correspond to the following goals:

Construction of compound concepts from the elementary ones. It can be observed in the case-based reasoning (cf. [6]), layered learning (cf. [24]), as well as rough mereology [13] and rough neural-computing [9, 11, 12], where we want to approximate target concepts step by step, using the simpler concepts that are easier to learn directly from data.

Construction of simple concepts from the advanced ones. It can be considered for the synthesis of classifiers, where we start from compound concepts (granules) reflecting the behaviour of a given object with respect to particular, often compound classification systems, and we tend to obtain a very simple concept of a decision class where that object should belong to [12, 14].

The first goal corresponds to generalisation of simple concepts while the second – to instantiation of general concept in a simpler, more specialized concept (cf. [20]). Obviously, we do not assume that the above are the only possible types of constructions. For instance, in a classification problem, decision classes can have a compound semantics requiring gradual specification corresponding to the first type of construction. Then, once we reach an appropriate level of expressiveness, we follow the second scenario to synthesize those compound specifications towards obtaining the final response of the classifier network.

4 General network architecture

When considering hierarchical structures for compound concept formation, several issues pop-up. At the very general level of hierarchy construction/learning, one has to make choices with respect to homogeneity and synchronization. We mention below how these factors determine the complexity of construction task.

Homogeneous vs. heterogeneous. At each level of hierarchy we make choice of the type of concepts to be used. In the simplest case each node implements the same type of mapping. We have studied such a fully homogeneous system in [22, 23] to express probabilistic classifiers based on the rough set reducts [15] and Naïve Bayes approach. This approach, dubbed Normalising Neural Networks (NNNs), is described later in the paper. First step towards heterogeneity is by permitting different types of concepts to be used at various levels in hierarchy, but retaining uniformity across a single layer. This creates typical layered learning model [24]. Finally, we may remove all restrictions on the uniformity of models in the neighbouring nodes. In this way we produce a structure which is more general but harder to control.

Synchronous vs. asynchronous. This issue is concerned with the layout of connections between nodes. If it has easily recognizable layered structure we regard it to be synchronized. In other words, we can analyze the hierarchical structure in a level-by-level manner and, consequently, have an ability to clearly indicate the level of abstraction for composite concepts. If we permit the connections to be established on less restrictive basis, the synchronization is lost. Then, the nodes from non-consecutive levels may interact and the whole idea of simple-to-compound precedence of concepts becomes less usable.

The layouts of classifier networks for various levels of homogeneity and synchronization are illustrated in Figure 1. The simplest case of homogeneous and synchronized network corresponds to Figure 1a. The partly homogeneous, synchronized architecture that we are attempting to formalize in this paper is shown in Figure 1b.

Figures 1c and 1d represent the harder cases. For a moment we do not attempt to address those eventualities. One can see that there are also other cases possible. For instance, we can consider asynchronous but homogeneous network described in [1], where the nodes correspond semantically to the complex concepts we want to approximate although syntactically the operations within the nodes remain of the same type, regardless of whether those nodes represent the advanced of very initial concepts.

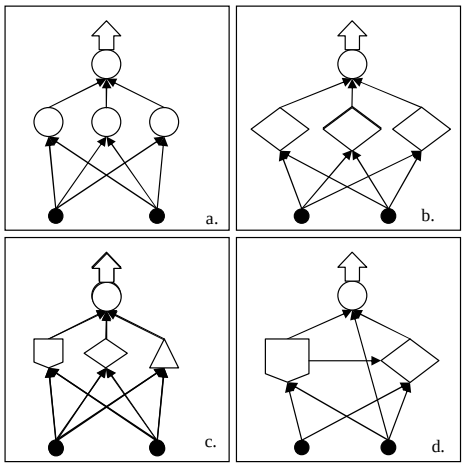

Fig. 1. Examples of network layout: a. both synchronized and homogeneous; b. synchronized and partly heterogeneous; c. synchronized and heterogeneous; d. neither synchronized nor homogeneous.

5 Hierarchical concept schemes

In this section we present a general notation for feedforward networks transmitting the concepts. Since we restrict ourselves to the two easier architecture cases illustrated by Figures 1a and 1b, we can consider the following notion:

Definition 1. *By a* hierarchical concept scheme *we mean a tuple* $(\mathcal{C}, \mathcal{MAP})$. $\mathcal{C} = \{C_1, \ldots, C_n, C\}$ *is a collection of the concept spaces (information granule systems), where* C *is called the target concept space. The* concept mappings

$$\mathcal{MAP} = \{map_i : C_i \to C_{i+1} : i = 1, \ldots, n, C_{n+1} = C\} \tag{1}$$

are the functions linking consecutive concept spaces.

We assume that any *feedforward concept network* corresponds to $(\mathcal{C}, \mathcal{MAP})$, i.e. each i-th layer provides us with the elements of C_i. In case of total homogeneity, we have equalities $C_1 = \cdots = C_n = C$ and $map_1 = \cdots = map_n = identity$. For partly homogeneous architecture, some of the mappings can remain identities but we should also expect non-trivial mappings between the concepts of entirely different nature, where $C_i \neq C_{i+1}$.

Following the structure of feedforward neural network, we calculate the inputs to each next layer as combinations of the concepts from the previous one. In general, we cannot expect the traditional definition of a linear combination to be applied. Still, the intuition says that the labels of connections should somehow express the level of concepts' importance in formation of the new ones. We refer to this intuition in terms of so called generalised linear combinations:

Definition 2. Feedforward concept scheme *is a triple* $(\mathcal{C}, \mathcal{MAP}, \mathcal{LIN})$*, where*

$$\mathcal{LIN} = \left\{lin_i : 2^{C_i \times W_i} \to C_i : i = 1, \ldots, n\right\} \tag{2}$$

defines generalised linear combinations *over the concept spaces* C_i*. For any* $i = 1, \ldots, n$*,* W_i *denotes the space of the* combination parameters*. If* W_i *is a partial or total ordering, then we interpret its elements as* weights *reflecting the relative importance of particular concepts in construction of the resulting concept.*

Let us denote by $m(i) \in \mathbb{N}$ the number of nodes in the i-th network layer. For any $i = 1, \ldots, n$, the nodes from the i-th and $(i+1)$-th layers are connected by the links labeled with parameters $w^{j(i)}_{j(i+1)} \in W_i$, for $j(i) = 1, \ldots, m(i)$ and $j(i+1) = 1, \ldots, m(i+1)$. For any collection of the concepts $c^1_i, \ldots, c^{m(i)}_i \in C_i$ occurring as the outputs of the i-th network's layer in a given situation, the input to the $j(i+1)$-th node in the $(i+1)$-th layer takes the following form:

$$c^{j(i+1)}_{i+1} = map_i\left(lin_i\left(\left\{\left(c^{j(i)}_i, w^{j(i)}_{j(i+1)}\right) : j(i) = 1, \ldots, m(i)\right\}\right)\right) \tag{3}$$

The way of composing functions within the formula (3) requires, obviously, further discussion. In this paper, we restrict ourselves to the case of Figure 2a, where map_i and

lin_i are stated separately. However, parameters $w_{j(i+1)}^{j(i)}$ could be also used directly in a *generalised concept mapping*

$$genmap_i : 2^{C_i \times W_i} \to C_{i+1} \tag{4}$$

as shown in Figure 2b. These two possibilities reflect construction tendencies described in Section 3. Function (4) can be applied to construction of more compound concepts parameterised by the elements of W_i, while the usage of Definitions 1 and 2 results rather in potential syntactical simplification of the new concepts (which can, however, still become more compound semantically).

One can see that function $genmap$ and the corresponding illustration 2b refer directly to the ideas of synthesizing concepts (granules, standards, approximations) known from rough-neural computing, rough mereology, and the theory of approximation spaces (cf. [9, 14, 17]). On the other hand, splitting $genmap$'s functionality, as proposed by formula (3) and illustrated in 2a, provides us with a framework more comparable to the original artificial neural networks and their supervised learning capabilities (cf. [23, 22]).

6 Weighted compound concepts

Beginning with the input layer of the network, we expect it to provide the concepts-signals $c_1^1, \ldots, c_1^{m(1)} \in C_1$, which will be then transmitted towards the target layer using (3). If we learn the network related directly to real-valued training sample, then we get $C_1 = \mathbb{R}$, lin_i can be defined as classical linear combination (with $W_i = \mathbb{R}$), and map_i as identity. An example of a more compound concept space originates from our previous studies [22, 23]:

Example 1. Let us assume that the input layer nodes correspond to various classifiers and the task is to combine them within a general system, which synthesizes the input classifications in an optimal way. For any object, each input classifier induces possibly incomplete vector of beliefs in the object's membership to particular decision classes. Let DEC denote the set of decision classes specified for a given classification problem. By the *weighted decision space* $WDEC$ we mean the family of subsets of DEC with elements labelled by their beliefs, i.e.:

$$WDEC = \bigcup_{X \subseteq DEC} \{(k, \mu_k) : k \in X, \mu_k \in \mathbb{R}\} \tag{5}$$

Any *weighted decision* $\widetilde{\mu} = \{(k, \mu_k) : k \in X_{\widetilde{\mu}}, \mu_k \in \mathbb{R}\}$ corresponds to a subset $X_{\widetilde{\mu}} \subseteq DEC$ of decision classes for which the beliefs $\mu_k \in \mathbb{R}$ are known.

7 Normalising Neural Networks

In this section we present detailed description of a classifier network for a particular type of probabilistic classifier. This work is more application-oriented and has been previously reported in [23]. The model that employs Bayesian classifiers as building blocks is dubbed Normalising Neural Network (NNN for short), as the transition functions used have the normalising ability (see subsection 7.3).

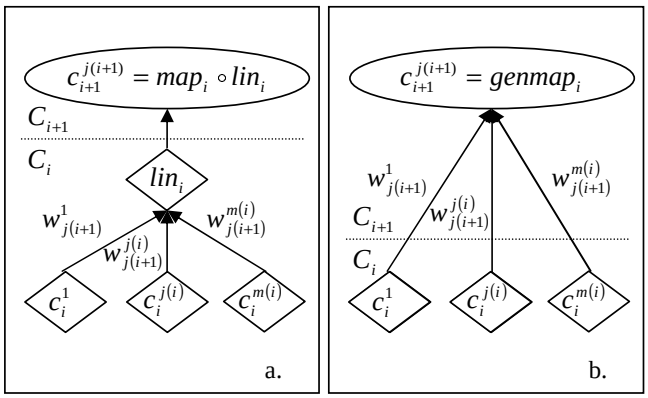

Fig. 2. Production of new concepts in consecutive layers: a. the concepts are first weighted and combined within the original space C_i using function lin_i and then mapped to a new concept in C_{i+1}; b. the concepts are transformed directly to the new space C_{i+1} by using the generalised concept mapping (4).

7.1 The starting point – the Naïve Bayes classifier

An example of the model, which is approximately consistent with the training cases, is the Naïve Bayes classifier. Although it ignores the attribute dependencies derivable from the data, it is proven to behave in a way very close to optimal in many classification problems [3]. It establishes h on the basis of probabilities $Pr(\cdot)$ estimated from sample T. The estimates are very simple, based on counting the occurrence of the attribute-value patterns in data (cf. [7, 8]). We use them as follows:

$$h(u) = \arg\max_{k\in\{1,\ldots,r\}} \Pr(d=k) \prod_{i=1}^{n} \Pr(a_i = a_i(u)|d=k) \qquad (6)$$

In the next subsections, we are also going to refer to an extended version of Naïve Bayes classifier, which is more flexible and less dependent on the "Naïve" assumptions about data independence. Let us present it using (natural) logarithms of probabilities, which makes it possible to replace the product in the above formula with the sum, as well as to change the outcome range from [0,1] to the entire real domain. The extended classifier uses the weighted sum of logarithms of the attribute probabilities:

$$h(u) = \arg\max_k v_0 \log \Pr(d=k) + \sum_{i=1}^{n} v_i \log(\Pr(a_i = a_i(u)|d=k)) \qquad (7)$$

Weights v_i determine the importance of attributes a_i in classification process. One of main strengths of NNNs is the existence of a method for learning these weights using the backpropagation-like technique for this generalised neural network model.

7.2 The structure of Normalising Neural Network

Given the list of the classifier components (like e.g. single attributes in the Naïve Bayes method), we put to the input layer neurons responsible for processing their classification preferences. We assume that each component provides the vector of r real values

expressing how it is likely to classify each given example to particular decision class. The difference with respect to the standard artificial network is that now we are going to combine the vectors instead of single real values.

The input to the neuron is a collection of vectors and the output is going to be a vector of r real values too. Abbreviation NNN comes from the fact that the weighted sums of vectors undergo normalization by means of the neuron transition functions $\phi : \mathbb{R}^r \to \triangle_{r-1}$ into the $(r-1)$-dimensional simplex of probabilistic distributions.

The structure of NNN with one hidden layer is presented in Figure 3. Vectors $x_i \in \mathbb{R}^r$ correspond to the classifier components for $i = 0, ..., n$. [1] Each j-th neuron in the hidden layer, for $j = 1, ..., m$, takes as an input the vector $s_j \in \mathbb{R}^r$ and provides as output the vector $y_j = \phi(s_j)$, where $y_j \in \triangle_{r-1}$ and $\phi : \mathbb{R}^r \to \triangle_{r-1}$. The input to the output neuron is denoted by $t \in \mathbb{R}^r$ and its output takes the form of $h = \phi(t)$, which is the result of the NNN calculations. Vectors $s_1, ..., s_m, t \in \mathbb{R}^r$ are the weighted sums of the outcomes of previous layers, i.e.:

$$t = \sum_{j=1}^{m} w_j y_j \quad \text{and} \quad s_j = \sum_{i=0}^{n} v_{ij} x_i \quad \text{for } j = 1, ..., m \tag{8}$$

Note, that this setting follows the general scheme for concept composition introduced in previous section (section 1).

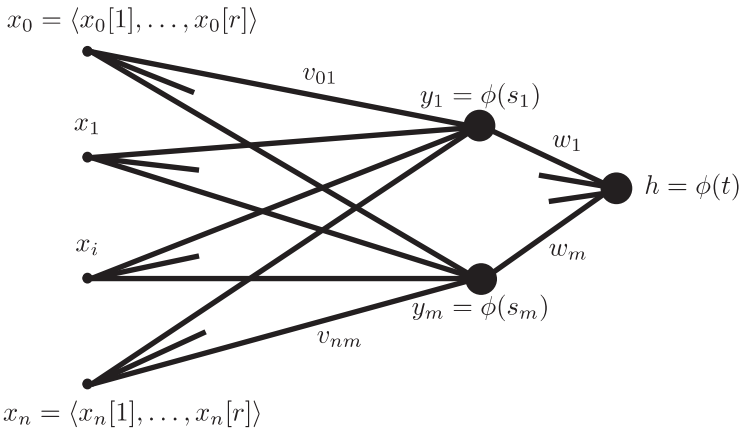

Fig. 3. The architecture of NNN with one hidden layer.

7.3 The NNN transition functions

Transition functions in NNN should be defined in a way that assures both the proper behaviour of calculation procedures and direct interpretation in extended neural network model. They should comply to some conditions, which generalise those formulated for

[1] Iteration $i = 0, ..., n$ is consistent with the application described in the next sections.

classical transition functions (cf. [4, 5, 7]). In NN we use (mostly sigmoidal) monotone functions. In case of NNN, one can say that transition function $\phi : \mathbb{R}^r \to \triangle_{r-1}$ is monotone, if it satisfies the following:

1. Inequality $s[k] > s[l]$ between the input vector coordinates results in inequality $\phi(s)[k] > \phi(s)[l]$ between the output vector coordinates, for $k, l = 1, ..., r$.
2. The increase in the input vector coordinate $s[k]$ results in increase of the corresponding output vector coordinate $\phi(s)[k]$, as well as decrease of the other coordinates $\phi(s)[l]$, for $l \neq k$.

We will use the following monotone function $\phi_\alpha : \mathbb{R}^r \to \triangle_{r-1}$, where parameter $\alpha > 0$ determines the steepness of transition:

$$\phi_\alpha(s) = \left\langle \frac{e^{\alpha s[1]}}{\sum_{l=1}^{r} e^{\alpha s[l]}}, \ldots, \frac{e^{\alpha s[k]}}{\sum_{l=1}^{r} e^{\alpha s[l]}}, \ldots, \frac{e^{\alpha s[r]}}{\sum_{l=1}^{r} e^{\alpha s[l]}} \right\rangle \tag{9}$$

Behavior of ϕ_α is illustrated in Figure 4, for two decision classes, i.e. $r = 2$.

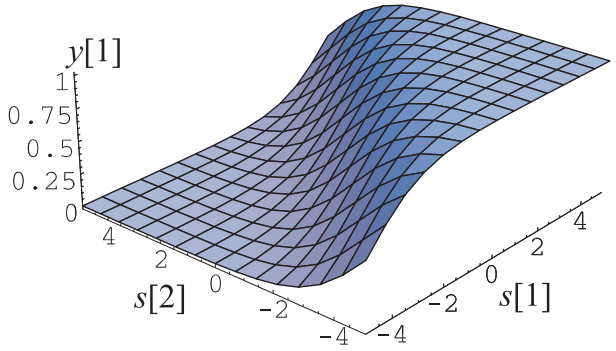

Fig. 4. Coordinate $y[1]$ of function $y = \phi_\alpha(s)$ for $\alpha = 1$ and $s \in \mathbb{R}^2$.

In the next subsection we generalise the backpropagation algorithm [4, 5, 8] in purpose of tuning the NNN weights. The advantage of using sigmoidal functions in this method is the way of calculating their derivatives. Figure 5 shows that ϕ_α generalises the behaviour of classical sigmoidal functions also at this level.

7.4 Backpropagation in NNN

The key issue is to equip the NNNs with an analogue of backpropagation procedure (cf. [4, 5]). In a nutshell, in the classical neural network model there exists effective method for calculating the error (gradient) ratios used in the weight updates. The error values for the output layer can be easily derived from the differences between the network outcomes and true answers for the training cases. For the hidden layers, the errors are calculated on the basis of linear combination of the error components propagated from the next layer.

$$\alpha \cdot \begin{bmatrix} \phi_\alpha(s)[1](1-\phi_\alpha(s)[1]) & \dots & -\phi_\alpha(s)[1]\phi_\alpha(s)[k] & \dots & -\phi_\alpha(s)[1]\phi_\alpha(s)[r] \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ -\phi_\alpha(s)[k]\phi_\alpha(s)[1] & \dots & \phi_\alpha(s)[k](1-\phi_\alpha(s)[k]) & \dots & -\phi_\alpha(s)[k]\phi_\alpha(s)[r] \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ -\phi_\alpha(s)[r]\phi_\alpha(s)[1] & \dots & -\phi_\alpha(s)[r]\phi_\alpha(s)[k] & \dots & \phi_\alpha(s)[r](1-\phi_\alpha(s)[r]) \end{bmatrix}$$

Fig. 5. Derivative matrix $D\phi_\alpha(s)$ for the function $\phi_\alpha : \mathbb{R}^r \to \triangle_{r-1}$ defined by (9).

The above method can also be applied in case of NNNs. Let us denote by $d = \langle d[1], \dots, d[r] \rangle$ the distribution, which we would like to obtain for a given training example. Let us consider the error function

$$E = \tfrac{1}{2} \textstyle\sum_{k=1}^{r} (h[k] - d[k])^2 \tag{10}$$

where $h = \langle h[1], \dots, h[r] \rangle$ is the output of NNN, as shown in Figure 3. Formula 10 is the normalized Euclidean distance between probabilistic distributions [19]. Its upper bound equals 1 and it is reached only if the two distributions have ones at different positions (e.g., $\langle 0, 0, 1, 0, 0 \rangle$, $\langle 0, 1, 0, 0, 0 \rangle$), i.e. if the output h is totally incorrect in comparison to d for a given training example.

Just like in the classical approach, we use negative gradient of E to tune the network weights. We treat gradient of (10) as the function of the weight vectors:

$$\frac{\partial E}{\partial w_j} = \left\langle h - d \,\middle|\, \frac{\partial h}{\partial w_j} \right\rangle \qquad \text{where} \qquad \frac{\partial h}{\partial w_j} = \left\langle \frac{\partial h[1]}{\partial w_j}, \dots, \frac{\partial h[r]}{\partial w_j} \right\rangle \tag{11}$$

Let us recall that $h = \phi(t)$ for $\phi : \mathbb{R}^r \to \triangle_{r-1}$ and $t = \sum_{j=1}^{m} w_j y_j$. We obtain

$$\left[\frac{\partial h}{\partial w_j} \right]^T = D\phi(t) \left[y_j \right]^T \tag{12}$$

where $D\phi$ denotes the derivative matrix of ϕ. Further, let us consider

$$\frac{\partial E}{\partial v_{ij}} = \left\langle h - d \,\middle|\, \frac{\partial h}{\partial v_{ij}} \right\rangle \tag{13}$$

Let us recall that $y_j = \phi(s_j)$ for $s_j = \sum_{i=0}^{n} v_{ij} x_i$, $j = 1, ..., m$. We obtain

$$\left[\frac{\partial h}{\partial v_{ij}} \right]^T = D\phi(t) w_j D\phi(s_j) \left[x_i \right]^T \tag{14}$$

Formula (14) provides an interpretation similar to that concerning backpropagation, described at the beginning of this subsection. For the consecutive layers, the error vectors are calculated on the basis of the error components propagated from the next layer. The way of calculations of $D\phi(t)$ and $D\phi(s_j)$ depends on the choice of ϕ. In our research we apply function ϕ_α introduced in Section 7.3.

7.5 NNNs for Bayesian classification

Now we show how to implement the concept of NNN in connection with the Naïve Bayes scheme. We consider the NNN architecture described in Figure 3. Given an example $u \in X$ to be classified, for each $i = 1, ..., n$, we put

$$x_i = \langle \log \Pr(a_i = a_i(u)|d = 1), \ldots, \log \Pr(a_i = a_i(u)|d = r) \rangle \tag{15}$$

We also add a special input that corresponds to the bias connection in a classical multilayer, feedforward neural network:

$$x_0 = \langle \log \Pr(d = 1), \ldots, \log \Pr(d = k), \ldots, \log \Pr(d = r) \rangle \tag{16}$$

For each $j = 1, ..., m$, we get the following formula for the coordinates of the input $s_j \in \mathbb{R}^r$ to the j-th neuron in the hidden layer:

$$s_j[k] = v_{0j} \log \Pr(d = k) + \textstyle\sum_{i=1}^{n} v_{ij} \log \Pr(a_i = a_i(u)|d = k) \tag{17}$$

It corresponds to the extended Naïve Bayes classifier (7). Since the NNN transition function ϕ is assumed to be monotone also in the sense of the properties presented in subsection 7.3, we obtain that $\arg\max_k s_j[k] = \arg\max_k y_j[k]$. Hence, if we classify cases using a single neuron with output y_j calculated from inputs (15,16), then the most probable decision class coincides with that given by (7).

Construction of the hidden layer with m neurons enables to learn automatically, using the generalised backpropagation introduced in Section 7.4, the coefficients of the ensemble of the weighted Naïve Bayes classifiers (7) and then – to synthesize them at the level of the output neuron $h = \phi(t)$. Such an approach closely follows the idea of classifier ensemble as introduced in [2].

In the next subsection we show the experiments with the application of function ϕ_α. Used in the structure from Figure 3, ϕ_α results with the vector coordinates

$$y_j[k] = \frac{\Pr(d = k)^{\alpha v_{0j}} \prod_{i=1}^{n} \Pr(a_i = a_i(u)|d = k)^{\alpha v_{ij}}}{\sum_{l=1}^{r} \Pr(d = l)^{\alpha v_{0j}} \prod_{i=1}^{n} \Pr(a_i = a_i(u)|d = l)^{\alpha v_{ij}}} \tag{18}$$

at the level of the hidden layer, and with the final output coordinates

$$h[k] = \frac{\prod_{j=1}^{m} e^{\alpha w_j y_j[k]}}{\sum_{l=1}^{r} \prod_{j=1}^{m} e^{\alpha w_j y_j[l]}} \tag{19}$$

Vectors y_j can take the form of arbitrary elements of $\triangle_{r-1}$ except its vertices. h can approach a vertex of $\triangle_{r-1}$ only up to the vector of the form $\langle \frac{1}{e^\alpha + r - 1}, ..., \frac{e^\alpha}{e^\alpha + r - 1}, ..., \frac{1}{e^\alpha + r - 1} \rangle$. This is why we decided to learn the NNNs using the reference vectors taking the following form for the training case $u \in T$:

$$d[k] = \begin{cases} \frac{e^\alpha}{e^\alpha + r - 1} & \text{iff } d(u) = k \\ \frac{1}{e^\alpha + r - 1} & \text{iff } d(u) \neq k \end{cases} \tag{20}$$

Using (20) decreases the risk of overfitting, what was confirmed by experiments.

Such change is quite natural if we take into account the fact that decision distributions estimated from the training sample T may only partially reflect the actual placement of decision classes in the universe of examples. By attaching non-zero probabilities to the values of decision other than the one actually observed in data we allow further variations in decision distributions that may happen when new, unseen objects arrive. The specific assignment of $t[i]$ is devised in a way that assures compatibility with network output and error formula (10).

7.6 Experiments with NNN

We implemented the generalised backpropagation algorithm described in subsection 7.4 and applied it to learning the NNN model described in subsection 7.5. Several data sets of different size and layout have been selected for this purpose (see Table 1). In most cases the split into training and test samples is inherited with the data set. The main observed value in all experiments is the ratio (percentage) of correctly classified objects in the training and testing sets. The presented results are averaged over several algorithm runs (usually 20 or more). They are compared with the results obtained with use of classical Naïve Bayes classifier. Experiments were also repeated with different choice of the NNN parameters. The results shown in Table 1 are repeated from publication [23].

The used data sets are taken from [25]. These are standard, well described benchmark data tables for which it is possible to find good reference results (see e.g. [8]). `DNA_small` is derived from the original table by taking only 20 attributes (out of 60), which are known to provide the largest amount of information (cf. [25]); `DNA_large` is the binary version of original data. `Soybean` and `primary_tumor` contain missing values.

The experiments were performed using the NNN network with one hidden layer composed 30-50 neurons in the hidden layer and the neuron transition function ϕ_α for $\alpha = 2$. In case of all data sets it was possible to obtain good classification results on training samples after reasonably small number of iterations of backpropagation algorithm (running about 2000 iterations by default). It confirms the idea of backpropagation presented in Section 7.4. Figure 6 illustrates the beginning of the learning process for the `DNA_small` data set.

One thing that remains to be investigated is the possible dependence between the number of units in hidden layer and the size and number of input (training) vectors. These dependencies are in general unknown even for the classical neural network models. However, even some partial, heuristic method for establishing the optimal size of network may help in making the process of NNN construction, learning and testing much more effective.

The results are summarized in Table 1 together with these obtained using the Naïve Bayes (NB) classifier. They are generally close to the best known results obtained for the data sets in discourse (cf. [8, 25]) and are significantly higher than some other classifier synthesis methods [22, 26]. In all cases the NNNs are noticeably better than NB classifiers on the training sets. In three out of four train/test cases (except `DNA_small`) the same may be told about the testing set.

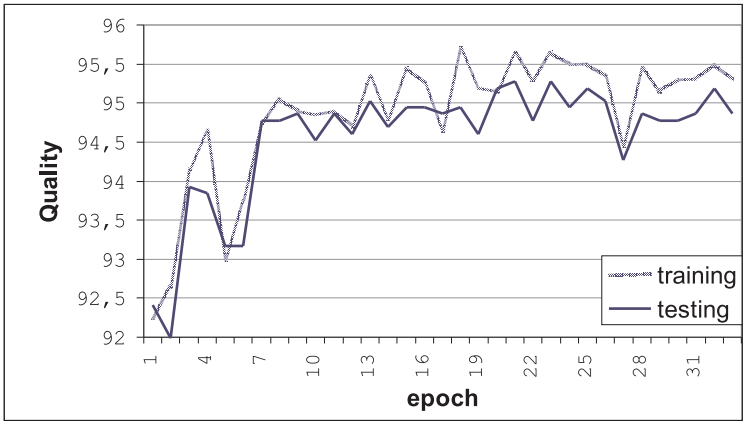

Fig. 6. Classification quality on the DNA_small data set – the consecutive epochs of backpropagation.

Relatively high classification rate (comparing to the best results known so far) on the primary_tumor data is encouraging. It was obtained as an average of several 10-fold cross-validation runs.

Table 1. Description of data sets (number of objects, attributes and decision classes) and summary of experimental results ($\alpha = 2, m = 30$).

Name	train/test obj.	attr./dec.	NNN–test	NB–test
DNA_small	2000/1187	20/3	95.34 %	95.62%
DNA_large	2000/1187	180/3	95.85%	93.68%
Soybean	307/376	35/19	91.14%	88.56%
SAT	4435/2000	36/6	82.96%	82.35%
primary_tumor	339/CV-10	17/21	45.55%	45.86%

The results of the NNN model are parameterised by the number m of neurons in the hidden layer and the value of parameter α. Figure 7 illustrates how the choice of these two parameters influences learning process and classification results. In case of the DNA_small data set, there is noticeable tendency suggesting that the larger number of neurons in the hidden layer contributes to the reduction of overfitting effect. These proves to be especially true for data sets with large number of decision values (small representation for each decision class) like primary_tumor and Soybean, for which the increase (from 30 to 40 and more) in the number of hidden neurons resulted in classification quality improvement. Optimal α value seems to be close to $\alpha = 2$, which can be deduced from Figure 7 and results (not presented) of the wider experiment, comparing the α values between 1.5 and 5.0. More massive experiments are obviously needed in purpose of finding optimal configurations of the NNN parameters.

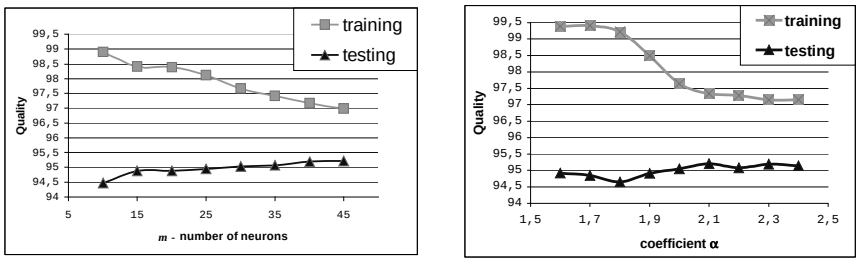

Fig. 7. Classification quality on `DNA_small` data with changing m and α.

8 Feedforward Concept Networks

In the previous sections we have already presented some examples of neurocomputing schemes for compound concept (classifier) construction. To go beyond and present general ideas regarding widely understood concept networks employing rough-neural paradigms, we present yet another example. This particular corresponds to the specific classifiers – the sets of decision rules obtained using the methodology of rough sets [15, 27]. The way of parameterisation is comparable to the proceedings with classification granules in [14, 17]. This example is for us a vehicle to introduce general concepts of our approach and to show the synergy with classical rough set methodologies. Please take note that throughout this section we will use the name of *concept network* and *classifier network* interchangeably.

Example 2. Let $DESC$ denote the family of logical descriptions, which can be used to define decision rules for a given classification problem. Every *rule* is labeled with its *description* $\alpha_{rule} \in DESC$ and *decision information*, which takes – in the most general framework – the form of $\widetilde{\mu}_{rule} \in WDEC$. For a new object, we measure its degree of satisfaction of the rule's description (usually zero-one), combine it with the number of training objects satisfying α_{rule}, and come out with the number $app_{rule} \in \mathbb{R}$ expressing the level of rule's *applicability* to this object. As a result, by the *decision rule set space* $RULS$ we mean the family of all sets of elements of $DESC$ labeled by weighted decision sets and the degrees of applicability, i.e.:

$$RULS = \bigcup_{X \subseteq DESC} \{(\alpha, \widetilde{\mu}, app) : \alpha \in X, \widetilde{\mu} \in WDEC, app \in \mathbb{R}\} \tag{21}$$

Definition 3. *By a* weighted compound concept space C *we mean a space of collections of* sub-concepts *from some* sub-concept space S *(possibly from several spaces), labeled with the* concept parameters *from a given space* V*, i.e.:*

$$C = \bigcup_{X \subseteq S} \{(s, v_s) : s \in X, v_s \in V\} \tag{22}$$

For a given $c = \{(s, v_s) : s \in X_c, v_s \in V\}$*, where* $X_c \subseteq S$ *is the* range *of* c*, parameters* $v_s \in V$ *reflect relative importance of sub-concepts* $s \in X_c$ *within* c_i*.*

Just like in case of combination parameters W_i in Definition 2, we can assume a partial or total ordering over the concept parameters. A perfect situation would be then to be able to combine these two kinds of parameters while calculating the generalised linear combinations and observe how the sub-concepts from various outputs of the previous layer fight for their importance in the next one.

For the sake of simplicity, we further restrict ourselves to the case of real numbers, as stated by Definition 4. However, in general W_i does not need to be in $\mathbb{R}$. Let us consider a classifier network, similar to Example 2, where decision rules are described by parameters of accuracy and importance (initially equal to their support). A concept transmitted by network refers to rules matched by an input object. The generalised linear combination of such concepts may be parameterised by vectors $(w, \theta) \in W_i$ and defined as a union of rules, where importance is expressed by w and θ states a threshold for the rules' accuracy.

Definition 4. *Let the i-th network layer correspond to the weighted compound concept space C_i based on sub-concept space S_i and parameters $V_i = \mathbb{R}$. Consider the $j(i+1)$-th node in the next layer. We define its input as follows:*

$$\begin{aligned} lin_i\left(\left\{\left(c_i^{j(i)}, w_{j(i+1)}^{j(i)}\right) : j(i) = 1, \ldots, m(i)\right\}\right) = \\ = \left\{\left(s, \sum_{j(i):s\in X_{j(i)}} w_{j(i+1)}^{j(i)} v_s^{j(i)}\right) : s \in \bigcup_{j(i)=1}^{m(i)} X_{j(i)}\right\} \end{aligned} \tag{23}$$

where $X_{j(i)} \subseteq S_i$ is simplified notation for the range of the weighted compound concept $c_i^{j(i)}$ and $v_s^{j(i)} \in \mathbb{R}$ denotes the importance of sub-concept $s \in S_i$ in $c_i^{j(i)}$.

Formula (23) can be applied both to $WDEC$ and $RULS$. In case of $WDEC$, the sub-concept space equals to DEC. The sum $\sum_{j(i):s\in X_{j(i)}} w_{j(i+1)}^{j(i)} v_s^{j(i)}$ gathers the weighted beliefs of the previous layer's nodes in the given decision class $s \in DEC$. In the case of $RULS$ we do the same with the weighted applicability degrees for the elements-rules belonging to the sub-concept space $DESC \times WDEC$.

It is interesting to compare our method of the parameterised concept transformation with the way of proceeding with classification granules and decision rules in the other rough set based approaches [14, 15, 17, 27]. Actually, at this level, we do not provide anything novel but rewriting well known examples within a more unified framework. A more visible difference can be observed in the next sections, where we complete our methodology.

8.1 Activation functions

The possible layout combining the concept spaces DEC, $WDEC$, and $RULS$ with the partly homogeneous classifier network is illustrated by Figure 8. Given a new object, we initiate the input layer with the degrees of applicability of the rules in particular rule-sets to this object. After processing with this type of concept along (possibly) several layers, we use the concept mapping function

$$map(ruls) = \left\{\left(k, \sum_{(\alpha,\widetilde{\mu},app)\in ruls:k\in X_{\widetilde{\mu}}} app \cdot \mu_k\right) : k \in \bigcup_{(\alpha,\widetilde{\mu},app)\in ruls} X_{\widetilde{\mu}}\right\} \tag{24}$$

that is we simply summarize the beliefs (weighted by the rules' applicability) in particular decision classes. Similarly, we finally map the weighted decision to the decision class, which is assigned with the highest resulting belief. The intermediate layers in

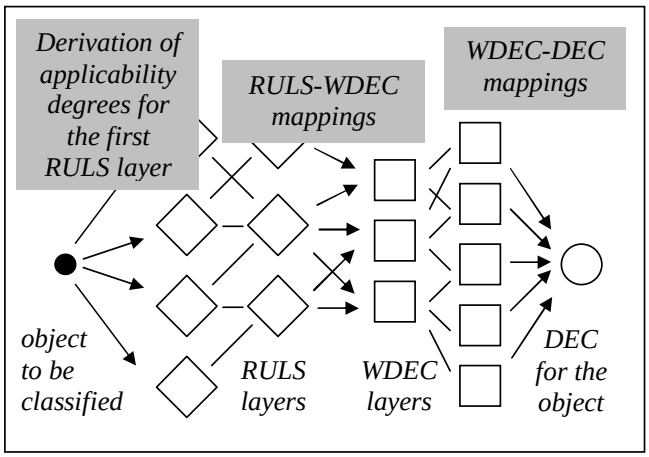

Fig. 8. The network-based object classification: the previously trained decision rule sets are activated by an object by means of their applicability to its classification; then the rule set concepts are processed and mapped to the weighted decisions using function (24); finally the most appropriate decision for the given object is produced.

Figure 8 are designed to help in voting among the classification results obtained from particular rule sets. Traditional rough set approach (cf. [15]) assumes specification of a fixed voting function, which, in our terminology, would correspond to the direct concept mapping from the first $RULS$ layer into DEC, with no hidden layers and without possibility of tuning the weights of connections. An improved adaptive approach (cf. [27]) enables us to adjust the rule sets, although the voting scheme still remains fixed. In the same time, the proposed method provides us with a framework for tuning the weights and, in this way, learning adaptively the voting formula (cf. [9, 14, 17]).

Still, the scheme based only on generalised linear combinations and concept mappings is not adjustable enough. The reader may check that composition of functions (23) for elements of $RULS$ and $WDEC$ with (24) results in the collapsed single-layer structure corresponding to the most basic weighted voting among decision rules. This is exactly what happens with classical feedforward neural network models with no non-linear activation functions translating the signals within particular neurons. Therefore, we should consider such functions as well.

Definition 5. Neural concept scheme *is a quadruple* $(\mathcal{C}, \mathcal{MAP}, \mathcal{LIN}, \mathcal{ACT})$*, where the first three entities are provided by Definitions 1, 2, and*

$$\mathcal{ACT} = \{act_i : C_i \to C_i : i = 2, \ldots, n+1\} \tag{25}$$

is the set of activation functions, which can be used to relate the inputs to the outputs within each i-th layer of a network.

It is reasonable to assume some properties of $\mathcal{ACT}$, which would work for the proposed generalised scheme analogously to the classical case. Given a compound concept consisting of some interacting parts, we would like, for instance, to guarantee that a relative importance of those parts remains roughly unchanged. Such a requirement, corresponding to monotonicity and continuity of real functions, is well expressible for weighted compound concepts introduced in Definition 3. Given a concept $c_i \in C_i$ represented as the weighted collection of sub-concepts, we claim that its more important (better weighted) sub-concepts should keep more influence on the concept $act_i(c_i) \in C_i$ than the others.

In section 7 we introduced sigmoidal activation function taken from [22, 23], working on probability vectors comparable to the structure of $WDEC$ in Example 1. That function, originating from the studies on monotonic decision measures in [18], can be actually generalised onto any space of compound concepts weighted with real values:

Definition 6. *By* α-sigmoidal activation function *for weighted compound concept space C with the real concept parameters, we mean function $act^{\alpha}_{C} : C \to C$ parameterised by $\alpha > 0$ which modifies these parameters in the following way:*

$$act^{\alpha}_{C}(c) = \left\{ \left(s, \frac{e^{\alpha \cdot v_s}}{\sum_{(t,v_t)\in c} e^{\alpha \cdot v_t}} \right) : (t, v_t) \in c \right\} \tag{26}$$

By composition of lin_i and map_i, which specify the concepts $c_i^{j(i+1)} \in C_{i+1}$ as inputs to the nodes in the $(i+1)$-th layer, with functions act^{α}_{i+1} modifying the concepts within the entire nodes, we obtain a classification model with a satisfactory expressive and adaptive power. If we apply this kind of function to the rule sets, we modify the rules' applicability degrees by their internal comparison. Such performance cannot be obtained using the classical neural networks with the nodes assigned to every single rule. Appropriate tuning of $\alpha > 0$ results in activation/deactivation of the rules with a relative higher/lower applicability. Similar characteristics can be observed within $WDEC$, where the decision beliefs compete with each other in the voting process (cf. [18]).

The presented framework also allows for modelling of other interesting behaviours. For instance, the decision rules which inhibit influence of other rules (so called *exceptions*) can be easily achieved by negative weights and proper activation functions, what would be hard to emulate by plain, negation-free conjunctive decision rules. Further research is needed to compare the capabilities of the proposed construction with other hierarchical approaches [9, 12, 13, 24].

8.2 Learning in classifier networks

A cautious reader have probably already noticed the arising question about the proper choice of connection weights in the network. The weights are ultimately the component that decides about the performance of entire scheme. As we will try to advocate, it is

– at least to some extent – possible to learn them in a manner similar to the case of standard neural networks. One such learning procedure we have already outlined in case of Normalising Neural Networks (subsection 7.4).

Backpropagation, the way we want to use it here, is a method for reducing the global error of a network by performing local changes in weights' values. The key issue is to have a method for dispatching the value of the network's global error functional among the nodes (cf. [5]). This method, when shaped in the form of an algorithm, should provide the direction of the weight update vector, which is then applied according to the learning coefficient. For the standard neural network model (cf. [4]) this algorithm selects the direction of weight update using the gradient of error functional and the current input. Obviously, numerous versions and modifications of gradient-based algorithm exist.

In the more complicated models which we are dealing with, the idea of backpropagation transfers into the demand for a general method of establishing weight updates. This method should comply to the general principles postulated for the rough-neural models (cf. [11, 27]). Namely, the algorithm for the weight updates should provide a certain form of *mutual monotonicity* i.e. small and local changes in weights should not rapidly divert the behaviour of the whole scheme and, at the same time, a small overall network error should result in merely cosmetic changes in the weight vectors. These principles are well exemplified by the procedure used in case of NNNs. The need of introducing automatic backpropagation-like algorithms to rough-neural computing were addressed recently in [9]. It can be referred to some already specified solutions like, e.g., the one proposed for rough-fuzzy neural networks in [10]. Still, general framework for RNC is missing, where a special attention must be paid to the issue of interpreting and calculating partial error derivatives with respect to the complex structures' parameters.

We do not claim to have discovered the general principle for constructing backpropagation-like algorithms for the concept (granule) networks. Still, in [22, 23] we have been able to construct generalisation of gradient-based method for the homogeneous neural concept schemes based on the space $WDEC$. The step to partly homogeneous schemes is natural for the class of weighted compound concepts, which can be processed using the same type of activation function. For instance, in case of the scheme illustrated by Figure 8, the conservative choice of mappings, which turn to be differentiable and regular, permits direct translation from the previous case. Hence, by small adjustment of the algorithm developed previously, we get a *recipé* for learning the weight vectors.

An example of two-dimensional weights $(w, \theta) \in W_i$ proposed in Section 5 is much harder to translate into backpropagation language. One of the most important features of classical backpropagation algorithm is that we can achieve the local minimum of an error function (on a set of examples) by local, easy to compute, change of the weight value. It does not remain easy for two real-valued parameters instead of one. Moreover, parameter θ is a rule threshold (fuzzified by a kind of sigmoidal characteristics to achieve differentiable model) and, therefore, by adjusting its value we are switching on and off (almost, up to the proposed sigmoidal function) entire rules, causing dramatic error changes. This is an illustration of the problems arising when we are dealing with

more complicated parameter spaces. In many cases we have to use dedicated, time-consuming local optimization algorithms.

Yet another issue is concerned with the second „tooth" of backpropagation: transmitting the error value backward throughout the network. The question is how to modify the error value due to connection weight, assuming that the weight is generalised (e.g. the vector as above). The error value should be translated into value compatible with the previous layer of classifiers, and should be useful for an algorithm of parameters modification. It means that information about error transmitted to the previous layer can be not only a real-valued signal, but e.g. a complete description of each rule's positive or negative contribution to the classifier performance in the next layer.

9 Conclusions

We have discussed construction of hierarchical concept (classifier) schemes aiming at layered learning of mappings between the inputs and desired outputs of classifiers. We proposed a generalised structure of feedforward neural-like network approximating the intermediate concepts in a way similar to traditional neurocomputing approaches. We provided the examples of compound concepts corresponding to the Bayesian and rule based classifiers, and showed some intuition concerning their processing through the network.

Although we have some experience with neural networks transmitting non-trivial concepts [22, 23], this is definitely the very beginning of more general theoretical studies. The most emerging issue is the extension of proposed framework onto more advanced structures than the introduced weighted compound concepts, without loosing a general interpretation of monotonic activation functions, as well as relaxation of quite limiting mathematical requirements corresponding to the general idea of learning based on the error backpropagation. We are going to challenge these problems by developing theoretical and practical foundations, as well as by referring to other approaches, especially those related to rough-neural computing [9, 11, 12].

Acknowledgements

The authors wish to thank Dominik Ślęzak who's ideas included in papers [18–23] are a cornerstone of this article.

This work is partly supported by grant 3T11C00226 from the Polish Ministry of Education and Science.

References

1. Bazan, J., Nguyen, S.H., Nguyen, H.S., Skowron, A.: Rough Set Methods in Approximation of Hierarchical Concepts. In: Proc. of RSCTC'2004. LNAI **3066**, Springer Verlag (2004) pp. 346–355
2. Dietterich, T.: Machine learning research: four current directions. AI Magazine **18/4** (1997) pp. 97–136.

3. Domingos, P., Pazzani, M.: On the Optimality of the Simple Bayesian Classifier under Zero-One Loss. Machine Learning **29** (1997) pp. 103–130.
4. Hecht-Nielsen, R.: Neurocomputing. Addison-Wesley (1990).
5. le Cun, Y.: A theoretical framework for backpropagation. In: Neural Networks – concepts and theory. IEEE Computer Society Press (1992).
6. Lenz, M., Bartsch-Spoerl, B., Burkhard, H.-D., Wess, S. (eds.): Case-Based Reasoning Technology: ¿From Foundations to Applications. LNAI **1400**, Springer (1998).
7. Mitchell T.M.: Machine Learning, McGraw Hill, Boston (1997)
8. Michie D., Spiegelhalter D.J., Taylor C.C.(eds): Machine Learning, Neural and Statistical Classification, Ellis Horwood, London (1994)
`http://www.amsta.leeds.ac.uk/~charles/statlog/`
9. Pal, S.K., Peters, J.F., Polkowski, L., Skowron, A.: Rough-Neural Computing: An Introduction. In: S.K. Pal, L. Polkowski, A. Skowron (eds.), Rough-Neural Computing. Cognitive Technologies Series, Springer (2004) pp. 15–41.
10. Pedrycz, W., Peters, J.F.: Learning in fuzzy Petri nets. In: J. Cardoso, H. Scarpelli (eds.), Fuzziness in Petri Nets. Physica (1998) pp. 858–886.
11. Peters, J.F., Szczuka, M.: Rough neurocomputing: a survey of basic models of neurocomputation. In: Proc. of RSCTC'2002. LNAI **2475**, Springer (2002) pp. 309–315.
12. Polkowski, L., Skowron, A.: Rough-neuro computing. In: W. Ziarko, Y.Y. Yao (eds.), Proc. of RSCTC'2000. LNAI **2005**, Springer (2001) pp. 57–64.
13. Polkowski, L., Skowron, A.: Rough mereological calculi of granules: A rough set approach to computation. Computational Intelligence, **17/3** (2001) pp. 472–492.
14. Skowron, A.: Approximate Reasoning by Agents in Distributed Environments. Invited speech at IAT'2001. Maebashi, Japan (2001).
15. Skowron, A., Pawlak, Z., Komorowski, J., Polkowski, L.: A rough set perspective on data and knowledge. In: W. Kloesgen, J. Żytkow (eds.), Handbook of KDD. Oxford University Press (2002) pp. 134–149.
16. Skowron, A., Stepaniuk, J.: Information granules: Towards foundations of granular computing. International Journal of Intelligent Systems **16/1** (2001) pp. 57–86.
17. Skowron, A., Stepaniuk, J.: Information Granules and Rough-Neural Computing. In: S.K. Pal, L. Polkowski, A. Skowron (eds.), Rough-Neural Computing. Cognitive Technologies Series, Springer (2004) pp. 43–84.
18. Ślęzak, D.: Normalized decision functions and measures for inconsistent decision tables analysis. Fundamenta Informaticae **44/3** (2000) pp. 291–319.
19. Ślęzak, D.: Various approaches to reasoning with frequency-based decision reducts: a survey. In: Polkowski, L., Tsumoto, S., Lin, T.Y. (eds): Rough Set Methods and Applications: New Developments in Knowledge Discovery in Information Systems. Physica Verlag, Heidelberg (2000) pp. 235–285.
20. Ślęzak, D., Szczuka, M., Wróblewski, J.: Harnessing classifier networks – towards hierarchical concept construction. In: Proc. of RSCTC'2004, LNAI **3066** Springer (2004) pp. 554–560
21. Ślęzak, D., Wróblewski, J.: Application of Normalized Decision Measures to the New Case Classification. In: W. Ziarko, Y. Yao (eds.), Proc. of RSCTC'2000. LNAI **2005**, Springer (2001) pp. 553–560.
22. Ślęzak, D., Wróblewski, J., Szczuka, M.: Neural Network Architecture for Synthesis of the Probabilistic Rule Based Classifiers. ENTCS **82/4**, Elsevier (2003).
23. Ślęzak, D., Wróblewski, J., Szczuka, M.: Constructing Extensions of Bayesian Classifiers with use of Normalizing Neural Networks. In: N. Zhong, Z. Raś, S. Tsumoto, E. Suzuki (eds.), Proc. of ISMIS'2003. LNAI **2871**, Springer (2002) pp. 408–416.
24. Stone, P.: Layered Learning in Multiagent Systems: A Winning Approach to Robotic Soccer. MIT Press, Cambridge MA (2000).

25. UCI Repository of ML databases, University of California, Irvine, (1998) http://www.ics.uci.edu/~mlearn/MLRepository.html.
26. Wróblewski J.: Ensembles of classifiers based on approximate reducts. Fundamenta Informaticae **47** (3,4), IOS Press (2001) pp. 351–360.
27. Wróblewski, J.: Adaptive aspects of combining approximation spaces. In: S.K. Pal, L. Polkowski, A. Skowron (eds.), Rough-Neural Computing. Cognitive Technologies Series, Springer (2004) pp. 139–156.