TASK QUARTERLY 7 No 1 (2003), 115–130

A NEURAL SYSTEM OF PHONEMATIC TRANSFORMATION

IGOR T. PODOLAK AND ANDRZEJ BIELECKI

Institute of Computer Science, Jagiellonian University, Nawojki 11, 31-072 Cracow, Poland
uipodola@theta.uoks.uj.edu.pl, bielecki@ii.uj.edu.pl

(Received 5 November 2001; revised manuscript received 8 September 2002)

Abstract: A common task in speech processing for which neural networks are widely employed is text-to-phoneme conversion. In this paper we propose a novel solution to this problem by combining a multilayer neural network and a modular hybrid system that uses basic rules to subdivide the original problem into easier tasks which are then solved by dedicated neural networks. A hybrid solution can be more rapidly constructed than a single net solution, and is easily extendable. Input data representation is also discussed. A voting committee concept is used to enhance generalization abilities of the system. Efficiency of the proposed systems is compared.

Keywords: neural networks, input/output data representation, phonematic transformation

1. Introduction

Artificial neural networks (ANN) are widely recognised as tools for solving tasks which can pose problems for rule-based systems. This is especially true when there is not enough knowledge available to design an algorithm but there are examples of correct performance. One of such tasks is phonematic transformation.

Phonematic transformation is a basic tool for any system of speech synthesis and can be described as translation of a written text into a string of phonematic characters defining the way in which it should be pronounced. One of the first systems for phonematic translation was DECtalk [1], which consisted of a vast number of translation rules including irregularities. Similarly, MITalk [2] remembered a large number of very common and irregular words and applied many rules to other words. Another system [3] was designed for translation of Polish texts. Basing on phonological knowledge, words were divided into several subgroups in a tree-like fashion, with each subgroup again divided according to pre-established patterns in the words. A number of rules were designed for the translation of each subgroup. The design of the system required a vast amount of phonological knowledge and could be applied to one language only.

Quite early in the development of neural networks, Sejnowski *et al.* [4] designed and implemented a neural network system for the translation of English texts using a multilayer feed-forward network in order to show that neural networks are able to perform tasks which are highly context-sensitive.

In this paper we present some approaches to Polish language phonematic translation with the use of ANNs. These include different feed-forward network architectures, with the stress on input data representation, and a hybrid network approach, where additional available knowledge about the problem is used in order to subdivide the problem into parts and use a specialised network for each of the parts. The results of the latter have been partially presented in [5, 6].

2. Phonematic translation

Phonematic transformation is a highly context sensitive task. The Polish language seems to be much simpler than English to specify all transformation rules. Bolc and Maksymienko have implemented a system basing on this approach and obtained good results [3].

A phoneme is the smallest element of a language which differentiates meaning but has no meaning itself. For instance, two Polish words:

tam [tam] (there) and
sam [sam] (alone)

differ only by the elements [t] and [s]. Therefore [t] and [s] are phonemes according to the definition given above. We have defined a phoneme by its functional role in a word, but another definition can be used as well: a phoneme is a set of features which distinguish the given phoneme from another one. Such features are called distinctive. In this way the phoneme is defined by its structure, *i.e.* the way it is pronounced. Let us present another Polish example:

dam [dam] (I will give),
tam [tam] (there).

The phoneme [d] is voiced, dental, stop, buccal, hard, whereas [t] is unvoiced, dental, stop, buccal, hard. It has been shown that voice is a distinctive feature. We used the Jakobson theory [7] of phoneme description, according to which all phonologic features are binary, therefore each phoneme is represented by a vector with binary ingredients. Fourteen distinctive features (binary ingredients) are needed to unambiguously describe all Polish phonemes (see Tables 1, 2 and 3).

Table 1. Distinctive features of vocal phonemes

feature	u	ɔ	i	ɛ	a	ɨ
velar	+	+	−	−	−	−
high-pitched	+	−	+	−	−	+
low	−	−	−	−	+	−
centralised	−	−	−	−	−	+

Automatic transformation from spelling into a phonematic transcription causes a number of problems.

First of all, spelling differs from a phonematic transcription — the transformation of a single letter into a phoneme is usually ambiguous. In most cases it depends on the context in which the letter appears — letters (or, generally, characters) both

Table 2. Distinctive features of consonantal phonemes

feature	ɲ	m	n	w	ŋ	j	l	r
velar	−	−	−	+	+	−	−	−
nasal	+	+	+	−	+	−	−	−
palatal	+	−	−	−	−	+	−	−
labial	−	+	−	−	−	−	−	−
lateral	−	−	−	−	−	−	+	−
dental	−	−	+	−	−	−	−	−

Table 3. Distinctive features of other consonantal phonemes

feature	β	ɸ	b	p	x	g	k	ʂ	ʑ	ç	ɟ	ʥʲ	c’	z	s	ʒ	ʃ	d	t	ʣ	ts	ʧ	ʨ
velar	−	−	−	−	+	+	+	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−
palatal	−	−	−	−	−	−	−	+	+	+	+	+	+	+	−	−	−	−	−	−	−	−	−
labial	+	+	+	+	−	−	−	−	−	−	−	−	+	+	+	+	−	−	−	−	−	−	−
fricative	+	+	−	−	+	−	−	+	+	+	−	−	−	−	+	+	+	+	−	−	−	−	−
dental	−	−	−	−	−	−	−	−	−	−	−	−	−	−	+	+	−	−	+	+	+	+	−
explosive	−	−	+	+	−	+	+	+	−	−	+	+	−	−	−	−	−	−	+	+	−	−	−
voiced	+	−	+	−	−	+	−	−	+	−	+	−	+	−	+	−	+	−	+	−	+	−	+

before and after the letter in question. For example, the letter sequence *dzi* gives in effect a single phoneme [ʥʲ] and using the rules of Polish language, letter *d* is transformed into [ʥʲ] and both *z* and *i* into empty phonemes. Thus, the transformation of *d* in this sequence depends upon the two letters following it, the transformation of *z* – upon both the letter before it and the one after, and that of *i* – upon the two preceding letters. Some letters are independent of the context (like *a*, *j*, *l*, *ł*, *ń*, *o*, and *ó*), some are dependent on only the neighbouring letters, and others – on two or three letters. Apart from this, inter-word and mid-word assimilations exist in Polish, for instance:

kosz truskawek [kɔʃ truskavɛk] (a basket of strawberries) and
kosz gruszek [kɔʒ gruʃɛk] (a basket of pears).

Another example of difficulties arising, are the possibilities of transformation of Polish nasals (*ą*, *ę*):

ręka [reŋka] (a hand): ę ↦ [eŋ]
dębu [dɛ͡mbu] (oak – a genitive form): ę ↦ [ɛ͡m]
Kęty [kɛ͡nty] (a town in Poland): ę ↦ [ɛ͡n]
kręcił [krɛɲʨiw] (he turned round): ę ↦ [ɛɲ]

Additional difficulties appear in rare and unorthodox events (for instance, in almost all cases, the digraph *rz* is pronounced in Polish as [ʒ], but in a word *zamarzać*, to freeze, it is pronounced as two phonemes [rz]). Transformations of which only few examples exist also pose many problems (for instance: *ź* ↦ empty phoneme, *d* ↦ [ts] in Polish). Furthermore, there are no rules for a few borrowings and specialist terms (see [8]). Moreover, a given text can be correctly pronounced in different ways. There is not only the classical and common Polish, but the Warsaw and Cracow pronunciations as well. We trained the implemented neural network according to the classical Cracow pronunciation basing on [9]. The possibilities of spelling-phonologic transformations used are presented in Table 4.

Table 4. Possibilities of phonematic transformations in Polish language

letter	possible phonemes							
a	a							
ą	ɔ	ɔ͡m	ɔ͡n	ɔ͡ɲ	ɔ͡ŋ			
b	b	p						
c	ʣ	ʧ	⌝	ʦ	c’	ʨ		
ć	ʣʲ	ć						
d	ʣ	ʣʲ	ʧ	ʦ	c’	ʨ	d	t
e	e							
ę	e	e͡m	e͡n	e͡ɲ	e͡ŋ			
f	ɸ	β						
g	g	ç	k					
h	x							
i	⌝	i	j					
j	j							
k	g	k	ɟ					
l	l							
ł	w							
m	m							
n	n	ɲ						
ń	ɲ							
o	ɔ							
ó	u							
p	b	p						
r	r	ʃ	ʒ					
s	s	ʑ	ʃ	z	ʒ			
ś	ʑ	ʂ						
t	d	t						
u	u	w						
w	ɸ	β						
y	j	ɨ						
z	⌝	s	z	ʂ				
ź	⌝	ʑ	ʂ					
ż	⌝	ʃ	ʒ					

A set of transformations given above allows us to correctly translate all Polish words save for a few borrowings, for instance *cyjanek* (cyanide) or foreign names that often appear in Polish, for instance the surname *Katz*.

The nature of the text-to-phoneme transformation problem calls for a solution involving a neural network system, as in a classic algorithmic approach one would need a set of all relevant transformation rules. Finding a proper set of rules for a given language is usually connected with many difficulties, and sometimes it is virtually impossible [3]. Even if a suitable set is available, it is always huge and therefore arduous to implement in an algorithmic way. At the same time, phonematic transformation of a given word is well known, if the type of pronunciation is given (*e.g.* literary or cockney for English). When using a neural network we only need to provide a set of examples, so it seems to be a convenient method to solve the problem.

There are some disadvantages though. No artificial neural network can assure 100% accuracy. There can be problems with finding a set of training examples that would be statistically suitable and small enough for the training part to end fast enough. Additionally, to find a proper network architecture is not an easy task and, usually, a number of trials are needed. A number of neural systems with architectures different from the one we have implemented are described in detail in Sections 4 and 5.

3. A feed-forward neural network

The problem of phonematic translation in this paper is dealt with a feed-forward ANN architecture. It consists of a number of neurons organised into layers. Neurons in subsequent layers (generally, though shortcut connections between neurons in non-subsequent layers are also possible) are connected with weighted synapses. Only neurons in different layers may be connected, with connections in one direction only, so that there can be no cycles. The input to each neuron is a weighted sum of previous layer neuron outputs, and its output is computed with an activation function.

The feed-forward network is usually trained with back-propagation which is a kind of a gradient algorithm. As the knowledge gathered by a network during learning is saved in the network's parameters, that is the synaptic weights, back-propagation repeatedly presents the network with successive examples consisting of tuples (x_i, d_i), where x_i is the network input vector and d_i – the desired answer. After the network computes its answer y, the instantaneous sum of squared errors at time step t

$$E(t) = \frac{1}{2}\sum_j (d_j - y_j)^2, \tag{1}$$

over all output neurons j is computed. In order to subsequently decrease the error signal, adjustments are applied to modify weights in the direction opposite to that of the gradient vector. The activation function $\phi(\cdot)$ has to be smooth, *i.e.* differentiable everywhere, in order to use a gradient descent learning algorithm, therefore a type of sigmoidal nonlinearity is used, most frequently a logistic function:

$$\phi(v) = \frac{1}{1+\exp(-v)}. \tag{2}$$

A number of other similar functions can be used [10].

To perform the appropriate weight modification the instantaneous error function gradient

$$\frac{\partial E}{\partial w_{kj}} = \frac{\partial E}{\partial e_k}\frac{\partial e_k}{\partial y_k}\frac{\partial y_k}{\partial v_k}\frac{\partial v_k}{\partial w_{kj}} \tag{3}$$

(with $e = E(t)$) needs to be computed for each weight. If the *local gradient* of a neuron indexed with i is defined as:

$$\delta_i = \frac{\partial E}{\partial e_k}\frac{\partial e_k}{\partial y_k}\frac{\partial y_k}{\partial v_k} \tag{4}$$

for neuron k, the weight upgrade may be given as:

$$\Delta w_{kj} = \eta \delta_j y_j, \tag{5}$$

where y_j is the activation of the neuron i connected with the neuron k with a synapse with the weight w_{kj}, and η is the learning parameter. Local gradients can be computed starting from the last (output) layer, where the error signal is available explicitly, and propagated back, hence the name of the whole algorithm.

The back-propagation algorithm has many modifications, designed to accelerate it. One of the simplest, yet very effective, is back-propagation with the *momentum* term:

$$\Delta w_{kj}(t) = \alpha \Delta w_{kj}(t-1) + \eta \delta_j y_j, \tag{6}$$

where the weight is partially modified in the direction computed at the previous step $t-1$. This has the beneficial effect of accelerating the weight modifications if weight updates, computed at each successive step, have the same sign, and stabilizing the upgrade if they have an opposite sign. Most other convergence acceleration methods pursue the same idea, but weight updates are computed in more detail.

4. A simple feed-forward ANN approach

Our first approach was to define a feed-forward ANN, where a letter to be translated, together with its left and right contexts (of three letters each) would be presented as input, and the network would be expected to produce the feature vector of the correct phoneme. Using a window sliding over the characters, the whole input could be translated.

The output representation was fixed for all the architectures tested: it consisted of the 14 important features plus 3 elements (neurons) needed for the representation of punctuation marks; a 17 element binary vector in all. At the same time, the representation of inputs varied.

4.1. Input data representation

Apart from the choice of a correct architecture for a given task, the input data should be appropriately presented. This problem is often overlooked, however, the actual representation can be of great influence on the way the network behaves: the quality of its answers, the training time, the amount of memory needed to simulate it.

Data can be generally divided into two types: real valued and symbolic valued. Real valued data is mainly represented using the following methods:

– **with 1 neuron**, where single real value is presented to the network „as is”, or scaled to a given interval, *e.g.*

$$sc : [a,b] \ni x \mapsto \frac{x-a}{b-a} \in [0,1] \tag{7}$$

if most of the values are known to be inside $[a,b]$; the input vector is very compact, but such representation usually requires a large number of hidden neurons and long training times;

– **binary**, where each real value is translated into a short n-element binary vector, *i.e.* the input scope $[a,b]$ is divided into n intervals and only one selected with

$$rb : [a,b] \ni x \mapsto \left\lfloor (n-1)\frac{x-a}{b-a} \right\rfloor + 1 \in [1,n] \tag{8}$$

is activated; the precision of value representation is therefore set prior to learning;

– **fuzzy**, which is a variant of binary representation: a real value is represented with an n-element vector with one neuron set to 1 as in binary, and the neurons on both sides of it set to intermediate vales, *e.g.* 0.5; this method makes the representation more foolproof;

– **the thermometer method**, where again an n-element vector is used and all neurons to the left and including the neuron selected with the function $rb(\cdot)$ are set to 1;
– **distributed**, where a real value from $[a,b]$ is represented with a distributed binary vector and a combination of different neuron activations; this type of representation is very foolproof and makes it possible to vary its size according to the variable's importance, but is the most difficult one to design.

The problem in phonematic transformation is of another nature: the input data are not numbers but symbols, which have to be represented with number vectors, *i.e.* we have to find a mapping:

$$rep: D \rightarrow \mathbb{R}^k, \tag{9}$$

where D is an n-element data set which has to be presented with k-element real valued vectors. With no relation between elements of the data set known, there is no order in the data set defined. However, order and relations among data set elements may be present, but are known only implicitly through the definition of the task a neural network is to solve.

Actually, the ANN is to find these relations and order. However, depending on the approach to the problem, a satisfactory solution may be found easily or not found at all. If some prior knowledge about the problem is available, it should be included not only in the design of network architecture, but in the creation of input data representation. Several methods are available to achieve this.

4.1.1. Unary representation

In NETtalk [4], each character was represented with a binary vector of 34 neurons, each fired for each letter or punctuation mark used. It is a straightforward representation, the main advantage of which is that all vectors are orthogonal. This usually leads to quicker learning and the representation does not need any preprocessing.

The design of a distributed representation of symbolic values is not so straightforward. Haphazard assignment of distributed codes to individual symbols can lead to unwanted effects. For example, some relations between different symbols, which have nothing to do with the problem at hand, can appear implicitly. Therefore, the process of distributed representation design should proceed in a systematic way.

On the other hand, the resulting nets have a large number of synapses and weights, which frequently hinders the generalization abilities. Vapnik and Chervonenkis [11] have shown that the probability of generalization error π_f, *i.e.* probability of correct classification of an example not included in the data set, does not differ by more than ε from the empirical (training) error ν_f is:

$$Pr[\sup|\nu_f - \pi_f| < \varepsilon] \leq 4\Phi(2N)\exp\left(-\frac{\varepsilon^2 N}{8}\right), \tag{10}$$

where N is the number of examples and $\Phi(2N)$ is the number of binary functions definable over $2N$ examples. $\Phi(2N)$ is bounded by N^d, where d is the Vapnik-Chervonenkis (VC) dimension of the net. For a class of functions to be learnable, the VC-dimension must be kept finite and low. It can be proven [12] that for feed-forward networks the VC-dimension is bounded by:

$$2\left\lfloor\frac{N_{hid}}{2}\right\rfloor N_{in} \leq VCdim \leq N_{weight}\log(eN_{all}), \tag{11}$$

where N_{in}, N_{hid}, N_{all} are the numbers of input, hidden and all neurons, and N_{weight} is the total number of weights. Unary representation introduces a high number of weights and, therefore, needs a high number of learning examples to achieve good generalization. This, however can be changed by introducing *distributed* representations generated through vector dissimilarity measures maximization.

4.1.2. Hamming distance maximization

There are some methods for decreasing the number of parameters, *i.e.* weights, in a neural network. On of the most popular is pruning, that is the removal of unneeded weights. Another approach could be through such preprocessing and design of a representation, that each symbol (a letter in our example) would be presented with a smaller number of neurons, resulting in a smaller number of weights. On the other hand, an orthogonal representation usually leads to faster learning and needs as many neurons as there are symbols used.

In a distributed representation we may try to maximize the measure of orthogonality, *e.g.* the Hamming distance between two n-dimensional vectors x and y:

$$H(x,y) = \sum_{i=1}^{n} |x_i - y_i|. \tag{12}$$

We have proposed the so called *cubic representation*

Definition 1 *A cubic representation of n symbols d_i is such a set of n k-dimensional binary vectors that the expected value $E[H(x,y)]$ of Hamming distances between all vector pairs is maximised with variance $var[H(x,y)]$ kept below a set ε.*

In such a representation, symbols are represented with maximally equidistant vectors, there are no relations among symbols that are not in the actual problem definition (or they would be minimised), and the network size is minimised.

To test this representation we have generated sets of vectors of a given dimensionality and selected equidistant subsets of vectors of a given size. Afterwards, the vectors were arbitrarily assigned to symbols used in the problem.

4.1.3. Correlation minimization

Another proposed approach is through minimization of correlation among vectors representing symbols.

Definition 2 *A representation minimizing the correlation of n symbols d_i is such a set of n k-dimensional vectors that the expected value of correlation between all vector pairs is minimised with variance $var[x^T y]$ smaller then a set ε.*

The virtues of this approach are similar to Hamming distance maximization. There are two possible procedures to find such representations: sequential, with generation of all possible vectors followed by selection of those with the lowest correlations, or using a purpose-designed neural network.

Such a network performs the task of auto-association: the input and output layers have the same number of neurons as the number of symbols, while the single hidden layer has the number of neurons equal to the size of the designed representation. The network is trained using a modified cost function (with example-by-example weight adaptation):

$$E(w(t)) = E_a(w(t)) + E_b(w(t)) + E_c(w(t)), \tag{13}$$

$$E_a(w(t)) = \frac{1}{2}\sum_{k=1}^{n}(d_k(t) - y_k^{out}(t))^2, \tag{14}$$

$$E_b(w(t)) = \frac{1}{2}\left(\sum_{j=1}^{l} y_j^{hid}(t) y_j^{hid}(t-1)\right)^2, \tag{15}$$

$$E_c(w(t)) = \frac{1}{2}\left(l - \sum_{j=1}^{l} y_j^{hid}(t) y_j^{hid}(t)\right)^2, \tag{16}$$

that is $E_a(w(t))$ is the standard sum of squared errors, $E_b(w(t))$ is the correlation of hidden layer activations generated for two successive input activations, and $E_c(w(t))$ is needed for the representations of all symbols have similar weights. A modification of the back-propagation learning algorithm, provided that learning rate is kept low, gives a simple method of training. The training set consists of n (same as the number of symbols) binary vectors, just as in unary representation, both as input and as output. After training the hidden layer, activations are used as the resulting representation. This method is naturally much faster than the sequential methods and makes it possible to find representations for greater numbers of symbols. One of the authors has presented the details in [13, 14].

Arbitrary assignment of vectors to symbols was employed in all the proposed methods. However, one should point out that some symbols used in the problem are interrelated, *e.g.* their occurrence leads to similar answers. Neural networks should uncover these relations, however this information can be found earlier and included in symbol representation as prior knowledge.

This can easily be achieved with a small modification of the dedicated neural network approach: instead of auto-association of n-element vectors, a small subset of the training set can be used in the following way (here described for the phonological transformation problem):

as input the middle letter with a small context (one letter on each side, widely proved to be satisfactory),

as output the expected phonological transformation.

This approach proved to yield satisfactory results. For symbols with different roles in the problem, different (*i.e.* uncorrelated) representations were found, whereas for symbols with similar roles, similar (*i.e.* more correlated) representations were assigned. Thus, training is done in two parts: first through representation design using a small subset of the original examples, then training with this representation the actual network, using the whole training set.

In Figure 1, a Kohonen map shows the clustering of vectors obtained with correlation minimization using a small subset of the actual data set. The obtained vectors were used as a training set for this Self Organizing Map (SOM), and similar vectors are known to activate SOMs of nearby neurons. In the figure, letters are

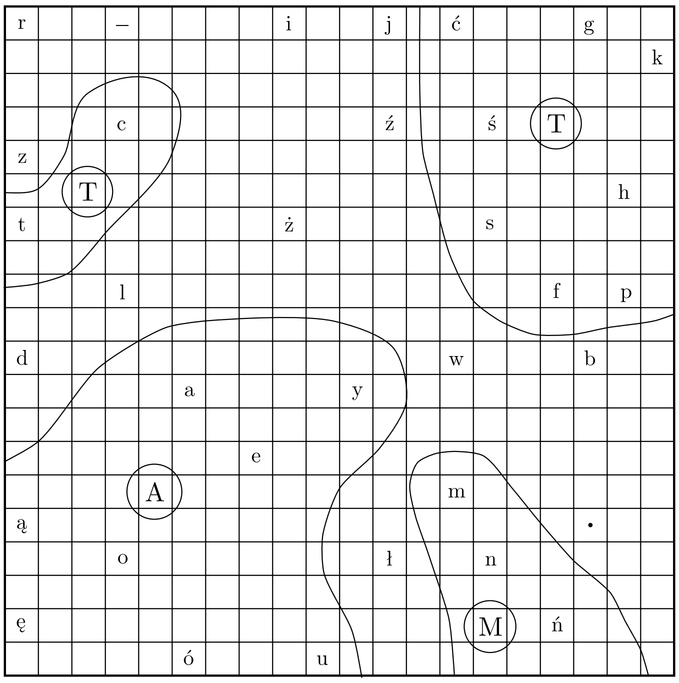

Figure 1. A Kohonen map representing a grouping of representation vectors obtained with data subset information. Letters in circles correspond to functional group names given in [8]. Two **T** groups visible constitute a single Kohonen map cluster

grouped in a way which corresponds to groups of letters defined in the rule-based system described in [8]. As the symbols are grouped just as in the rule system, it can clearly be inferred that the representation reflects the problem.

The search for symbol representation with the use of dedicated neural networks, as described, may yield activation vectors which are not binary, as in the previous methods. This extends the input space used, although in our experiments, for most problems (not only text-to-phoneme conversion), hidden neurons have usually became saturated and activation vectors have actually been (almost) binary. In some cases, when the hidden layer was large, some neurons were found to be unimportant and not needed to distinguish between different inputs, which could be inferred from their activation being constant. In such cases, these neurons were not used.

4.1.4. Average disorder maximization

In decision tree learning methods, examples are divided in such a way that resulting subsets are maximally homogeneous, which is achieved through minimizing the *average disorder AD*:

$$AD=\sum_b\left(\frac{n_b}{n}\right)\left(\sum_c-\left(\frac{n_{bc}}{n_b}\right)\log_2\left(\frac{n_{bc}}{n_c}\right)\right), \tag{17}$$

where n is the overall number of examples, n_b – the number of examples in branch b, n_{bc} – the number of these in branch b which belong to class c [15]. A neural network, on the other hand, uses all features (that is, all input neuron activations) at the same time, so all should be equally important and the test of each would divide the possible inputs into equal parts. This can be achieved by disorder maximization.

Definition 3 *A representation maximizing the average disorder of n symbols d_i is such a set of n k-dimensional binary vectors represented with a matrix M which maximizes disorder* $\mathrm{ent}(M)$*:*

$$\mathrm{ent}(M) = \mathrm{vent}(M) \times \mathrm{hent}(M), \tag{18}$$

$$\mathrm{vent}(M) = \frac{1}{k}\sum_{i=1}^{k}\left(-\frac{l_i^0}{n}\log_2\frac{l_i^0}{n} - \frac{l_i^1}{n}\log_2\frac{l_i^1}{n}\right), \tag{19}$$

$$\mathrm{hent}(M) = \frac{1}{n}\sum_{j=1}^{n}\left(-\frac{m_i^0}{k}\log_2\frac{m_i^0}{k} - \frac{m_i^1}{k}\log_2\frac{m_i^1}{k}\right), \tag{20}$$

where l_i^1 is the number of 1's in column i, l_i^0 is the number of 0's in column i, m_i^1 is the number of 1's in row i, and l_i^0 is the number of 0's in row i.

The representation is found with a probabilistic algorithm choosing a set of random vectors, computing their average disorder measure, and replacing it afterwards with a better one [13]. A genetic algorithm can also be used for this purpose.

4.1.5. "Mixed" representation

If some information about the problem, apart from the example input-output pairs, is known in the form of additional features, it should be used. This gives rise to a *mixed* representation.

Definition 4 *A mixed representation of n symbols is a set of n k-dimensional vectors constructed of p known features of each symbol and $(k-p)$-dimensional distributed vectors.*

In the phonological transformation problem, a set of features were known to be important, namely: whether a letter is nasal, whether soft, whether a sonorant, whether it can appear in a digraph (*e.g.* letters *c*, *d*, *r*, *s*, *z*, *ź*, *ż*), or it is softened in certain situations. Each of these features was represented with a single neuron.

As the distributed part of the representation, any of previously defined methods can be used.

4.2. Results

Several tests were performed using the representations described, with results given in Table 5. For the sake of comparison *binary* representation results are also included. This is the most condensed method where each symbol is represented as a binary number converted to a vector, therefore there is a need for 6 neurons to represent 34 characters used in the phonological transformation problem (letters plus punctuation marks). In so compressed a representation, the problem becomes very difficult and results are far from satisfactory.

The test set included 9 922 examples, all of which were not included in the train set. Each network was trained for 5 000 epochs.

The representations, basing on generalization results, can clearly be divided into groups: binary, unary, equal distance methods, where after finding representation vectors they were arbitrarily assigned to symbols (cubic, correlation minimization, disorder maximization), methods which, when designed, used some information about

Table 5. Generalization results for different representations and number of hidden neurons

representation type	representation length	hidden neurons	% wrong
binary	6	20	11.89
	6	30	11.66
	6	40	11.34
cubic	10	25	8.29
	15	25	5.48
	20	25	5.72
correlation minimization	10	25	9.82
	15	25	5.98
	20	25	5.53
	25	25	4.78
	34	25	4.43
average disorder maximization	10	25	9.68
	15	25	6.35
	20	25	5.06
	25	25	6.35
correlation minimization trained on data subset	10	25	6.27
	15	25	4.78
	20	25	5.08
	25	25	4.25
	34	25	3.73
mixed	15	25	4.63
	20	25	4.43
	25	25	3.79
	34	25	3.60
unary	34	25	3.94

the actual problem (correlation minimization, but trained on an actual data subset, and the mixed representation composed of a number of symbol features known from an expert, with a cubic distributed representation). Methods based on arbitrary vector assignment achieve good results, but are less efficient than methods which utilise some priorly available knowledge of the problem.

5. A hybrid system

It is possible to find better approximations through the modularization of problem solution. A modular architecture captures the problem at two levels: first at a global level, finding its global structure, then at a local level (the committee machine is an example of such approach [16]). Modules can also be defined using the prior knowledge available.

Neural network systems are mainly used for problems defined only through a number of examples. On the other hand, there is usually some prior knowledge which could, and should be included in the problem's solution. This proves to be complicated in most cases. Neural networks can be used in hybrid systems that use rules for introductory division of the problem into subtasks, which afterwards either give an outright answer or can be solved with other methods, for example other neural networks.

Neural hybrid systems can be divided into different categories: *unified*, with all processing elements implemented by neural networks, *transformational*, where symbolic and neural representations are transformed to and fro, and *modular*, which comprise of cooperating symbolic and neural modules [17].

In the text-to-speech transformation problem there are many rules which decide on the transformation. A rule-based system [8] for the Polish language consists of over 700 highly complicated rules which are hard to modify. On the other hand, a small subset of these rules can divide the whole problem into subtasks. Pronounciation of some of the letters is invariable and independent of the context, *e.g.* a or j, or invariable provided they occur in a given context, *e.g.* c followed by h is translated into a null phoneme, as is z preceded by s.

There are other situations where it is only needed to find a single feature of a letter, *e.g.* if b is voiced, then it is pronounced as [b], if unvoiced then as [p]. There are some variations when a single feature needs to be checked and the letter has to occur in a given context, *e.g.* if letter d is followed by z and d is voiced, then it is pronounced with phoneme [ʥ]. This predicate can be checked with a specially trained neural network. Since its input space is much smaller and the problem better defined, we can predict that its accuracy should be much higher than that of a conventional neural network.

There are other rules which recognize more complicated situations and utilize a neural network. This results in a highly coupled modular hybrid system with 83 rules organised in a tree, of which 23 give direct answers and 61 use one of the networks. 5 networks were trained to respond to the following situations:

ann1 whether a letter should be pronounced as voiced,
ann2 whether a letter pair rz should be pronounced as [rz], [ʒ], or [ʃ],
ann3 recognizes velarity, lowedness, nasality, palatality and labiality in the pronounciation of *ą* and *ę*,
ann4 finds the proper phoneme for u, i and y, which are pronounced similarly,
ann5 recognizes how an n should be pronounced.

Training data is divided into small subsets appropriate to the task learned. Since the nets solve easier tasks now, they have simpler architectures, with a smaller number of output neurons (not all features of the output phoneme are required) and some require smaller context, *i.e.* with less input neurons. Correct architectures with optimal number of hidden neurons are therefore simpler to find, even though there are more of them now. At the same time, with smaller networks there are less free parameters, what results in better generalization.

The modular architecture is also much easier to manage, as it allows relearning of individual small networks. This is in contrast to a single network, which requires much processing to retrain, while feedforward networks trained with a backpropagation type of algorithm cannot learn incrementally.

To enhance the system's precision, we have used voting committees of networks by training a pool of 3 differently initialised networks to solve each part of the problem and then choosing the answer returned by most. Using k classifiers, the probability that the test error is no more than ε greater than the validation error is at least $1-2ke^{-\frac{1}{2}\varepsilon D}$, where D is the minimum of the number of validation and test examples [18].

All the available 41 982 examples were first divided into subsets appropriate for each of the networks, then divided into training, validation, and test sets. The number of examples in each are given in Table 6. As the number of examples was much smaller, because of direct translation of some letters, as well as architectures, the time needed to train the whole system was much shorter than when a single network was used. The validation sets were used to prevent overtraining; training was stopped when the validation error began to rise. The test sets were used to select the committee members.

Table 6. Number of examples for all the module networks. Since some of the data was converted directly by rules, the numbers do not add up to the total number of available examples

	ann1	ann2	ann3	ann4	ann5
training	6047	146	487	2538	907
validation	2740	88	210	1281	488
test	2676	94	189	1284	517

The results, including the learning results of individual small problem nets, then the overall rate for the hybrid system without voting, are shown in Table 7. Results of each third best subnet were used to compute the overall error. Comparisons with the single net approach for different representation lengths are also included for each character. Results were computed over the same test set. For each small network, several architectures were tested and the three with best results over the test set selected. Therefore, the results are fine-tuned for the test set with an error decrasing to 0.35%. All characters were represented with the cubic method described in Section 4.1.2, but of a different length.

To give actual insight into the system's performance, it was tested on a separate test set of 6 322 examples, results of which are given in Table 8. The use of voting committees adds a small increase in performance, but even the simple hybrid system, *i.e.* one using only one small net for each of the subtasks, gives much better results than the single net approach. All single nets had 50 hidden neurons, therefore the generalization rates are better than those in Table 5, where all nets had only 25 hidden neurons.

The results clearly show that implementation of a hybrid system gives significantly better generalization rates. It is therefore worth building such a system if it is possible, *i.e.* when some prior knowledge is available, quite apart from the much better manageability of such systems.

6. Conclusions

The described experiments show that phonematic transformation of Polish texts is a task which can be solved effectively using ANNs. Classical multilayer nets performed the task with above 95% efficiency, the best one even over 98%. On the other hand, it proves to be difficult to select the optimal network architecture. Therefore *a priori* knowledge of the problem should be used whenever available. If the knowledge is incomplete, it can be used in a hybrid rule-network system, which

Table 7. Best subtask networks (ann1–ann5) and single net (big10–big34) performances

	length of representation	number of hidden neurons	mean square test error	badly recognised	wrong
ann1	34	18	0.00558	22	0.82%
	34	15	0.00666	24	0.90%
	34	21	0.00673	21	0.78%
ann2	25	4	0.00376	0	0.00%
	25	6	0.00539	0	0.00%
	25	10	0.00548	0	0.00%
ann3	10	10	0.01402	1	0.53%
	25	8	0.01576	1	0.53%
	25	10	0.01584	2	1.06%
ann4	34	18	0.00722	6	0.47%
	25	21	0.00782	6	0.47%
	34	12	0.00789	5	0.39%
ann5	34	15	0.01248	3	0.58%
	25	21	0.01276	3	0.58%
	10	12	0.01294	4	0.77%
hybrid system				36	0.35%
big10	10	50	0.04212	381	3.66%
big15	15	50	0.01396	132	1.27%
big20	20	50	0.01387	132	1.27%
big25	25	50	0.01640	159	1.53%
big34	34	50	0.03251	241	2.31%

Table 8. Generalization results for hybrid systems and for single nets with different representations

	badly recognised examples	wrong
voting committee hybrid	50	0.79%
simple hybrid system	57	0.90%
big10 (10 neurons per letter)	226	3.57%
big15 (15 neurons per letter)	84	1.33%
big20 (20 neurons per letter)	80	1.27%
big25 (25 neurons per letter)	97	1.53%
big34 (34 neurons per letter)	136	2.15%

is believed to be more effective than a single ANN. The experiments performed have confirmed this opinion (see Tables 7 and 8).

References

[1] Digital Equipment Corporation, Maynard Mass, *DECtalk DTC01 Owner's Manual*

[2] Allen J 1985 *From Text to Speech: the MITalk System*, Cambridge University Press

[3] Bolc L and Maksymienko M 1981 *Text Processing Computer System*, Warsaw University Press, Poland (in Polish)

[4] Sejnowski A and Rosenberg C R 1986 *Technical Report* 86/01, John Hopkins University, Department of Electrical Engineering and Computer Science, USA
[5] Bielecki A, Podolak I T, Wosiek J and Majkut E 1996 *Acta Physica Polonica* **B 27** (9) 2253
[6] Podolak I T, Lee S -W, Bielecki A and Majkut E 2000 *Proc. Int. Conf. on Pattern Recognition*, Barcelona, Spain, Vol. 1, pp. 720–724
[7] Jakobson R and Halle M 1969 *Language Foundations*, PWN, Wroclaw (in Polish)
[8] Steffen-Batogowa M 1975 *Automatisation of Polish Language Phonematic Transcription*, PWN, Warsaw (in Polish)
[9] Karaś M and Madejowa M 1977 *Polish Language Pronounciation Dictionary*, PWN, Warsaw (in Polish)
[10] Duch W and Jankowski N 1999 *Neural Computing Surveys* **2** 163, http://www.icsi.berkeley.edu/˜jagota/NCS
[11] Vapnik V 1998 *Statistical Learning Theory*, John Wiley and Sons
[12] Baum E B and Haussler D 1989 *Neural Computation* **1** (1) 3
[13] Podolak I T 1998 *Problems of Data Representation in Layered Neural Networks*, PhD Thesis, University of Mining and Metallurgy, Cracow, Poland (in Polish)
[14] Podolak I T 1998 *Comput. Phys. Commun.* **117** (1–2) 62
[15] Winston P H 1992 *Artificial Intelligence*, Addison Wesley
[16] Haykin S 1994 *Neural Networks: A Comprehensive Foundation*, Maxwell Macmillan
[17] McGarry K, Wermter K and McIntyre J 1999 *Neural Computing Surveys* **2** 62, http://www.icsi.berkeley.edu/˜jagota/NCS
[18] Bax E 1998 *Neural Computation* **10** (4) 975