MULTILAYER PERCEPTRONS AND DATA COMPRESSION

Robert MANGER, Krunoslav PULJIĆ

Department of Mathematics
University of Zagreb
Bijenička cesta 30
10000 Zagreb, Croatia
e-mail: {`manger, nuno`}`@math.hr`

Abstract. This paper investigates the feasibility of using artificial neural networks as a tool for data compression. More precisely, the paper measures compression capabilities of the standard multilayer perceptrons. An outline of a possible "neural" data compression method is given. The method is based on training a perceptron to reproduce a given data file. Experiments are presented, where the outlined method has been simulated by using differently configured perceptrons and various data files. The best compression rates obtained in the experiments are listed, and compared with similar results produced in a previous paper by holographic neural networks.

Keywords: Artificial neural networks, data compression, multilayer perceptrons, holographic neural networks, experiments

1 INTRODUCTION

Artificial neural networks [5] are an interesting and flexible computing paradigm. They are usually applied to simulate systems that cannot otherwise be described by explicit mathematical modelling. The behaviour of a considered system is captured as a set of stimulus-response associations. A chosen network is trained on a training set consisting of some of the available stimulus-response pairs. During training, the parameters within the network are adjusted, so that for each stimulus from the

training set, taken as input, the network produces an output approximately equal to the associated response. After successful training, a network should not only be able to reproduce the learned stimulus-response pairs, but also to generalize the learned associations.

In this paper we are considering neural networks as a tool for *data compression* [12]. A file to be compressed is interpreted as a set of stimulus-response pairs. A chosen network is trained on that set. After training, the network should enable approximate reproduction of the original file. Consequently, the stored version of the trained network could be regarded as an encoded version of the file. If the stored network happens to be physically smaller than the original file, we can talk about data compression. Note that in the proposed application of neural networks we are not directly concerned with generalization of the learned associations. Instead, we are faced with a different problem of finding a suitable network, which is as small as possible, while still assuring a desired accuracy in reproducing the learned associations.

The idea of using artificial neural networks for data compression is not new, and it can be encountered for instance in [7, 8, 9, 11, 14]. However, most of the relevant papers found in literature deal with specific applications and special data, or employ neural networks only as enhancements to traditional compression methods. A more general approach has been introduced in a previous paper of ours [10], where a relatively exotic type of *holographic neural networks* [16] has been employed. The whole effort described in [10] has been motivated by the claim from [15] that holographic networks provide spectacular storage capacities. The primary aim of this paper is to check how successfully the same general approach can be realized with more conventional network types, such as *multilayer perceptrons* [3, 5]. Thus, in this paper we are concerned with measuring overall data compression capabilities of multilayer perceptrons, in order to see if those capabilities are better or worse than for holographic networks.

The paper is organized as follows. Section 2 gives an outline of a possible neural compression method. Section 3 describes properties of a suitable family of multilayer perceptrons. Section 4 presents experiments, where the outlined compression method has been simulated by using the described family of multilayer perceptrons and three groups of data files. Section 5 summarizes the results of experiments, and also compares the obtained compression rates with those produced in [10]. Finally, Section 6 offers a conclusion.

2 A NEURAL COMPRESSION METHOD

Suppose that we are given a sequential file consisting of p (let say) real values r_1, r_2, ..., r_p. We would like to design a method to store the file in a neural network; or, differently speaking, we would like to design a procedure to train the network with the values $r_1, r_2, \ldots, r_p$, so that after training each of those values is available (at least approximately) as a response to a suitable stimulus to the network. Let us now discuss how such a method could look like.

Since the values r_1, r_2, ..., r_p are generally independent, each of them must be used as the response in at least one training example. Obviously, one training example per value is enough. Also, for $j_1 \neq j_2$, the stimuli associated with r_{j_1} and r_{j_2}, respectively must be different (otherwise the network would be exposed to contradictory data). Thus, the stimulus associated with r_j must be equivalent to the index j. For simplicity, we can assume that our original training set consists of the pairs (j, r_j), $j = 1, 2, \ldots, p$. As we see, both the stimulus and the response are vectors of length 1, and the training set consists of p training examples. After training, r_j can hopefully be reproduced by stimulating the network with j.

For the success of our file storage method, it is crucial that the learned values r_j are reproduced with satisfactory precision. In order to assure this, we will have to use a sufficiently large network, i.e. a network consisting of an adequate number of neurons with a sufficient total number of free parameters. On the other hand, if we wish that the described storage method is also a compression method, we must require from the network to be smaller than the original file. Let n denote the physical size of the stored network, expressed as a multiple of the size needed to store one real number. Then on one hand n should be large enough to assure a satisfactory reproduction accuracy, and on the other hand it should still be smaller than p. In fact, we are interested in finding the smallest possible n meeting the described criteria.

Putting it all together, the basic version of our hypothetical compression method should look as shown in Figure 1. Note that Figure 1 gives only an outline, leaving many details unexplained. For instance, it is not quite clear how to generate efficiently the set of all suitable networks for a given file. Also, there are many possible algorithms for training a chosen network, and each of them has its own advantages and drawbacks.

Note that our proposed compression method treats all files in the same way, irrelevantly if they originate from one-dimensional or multi-dimensional applications. Indeed, if we deal for instance with a two-dimensional image, then the whole corresponding set of pixels would first be stored as a sequential file, and each pixel would obtain an implicit index (sequence number) showing its relative position within that file. During training, our method would use those implicit indices as stimuli, and the associated pixels as responses. Consequently, our two-dimensional image would be processed as being a one-dimensional data set.

Let us now consider the computational complexity of the proposed compression method. If the file size p becomes larger, the network size n should also increase in order to retain the same precision in response. It is well known [3, 5] that the available network training algorithms have large (surely nonlinear) complexities, i.e. for a bigger network and a bigger training set they require unproportionally more computing. Thus, it is obvious that our method would not be able to code directly very big files – it would be computationally too expensive.

The only way how to deal with a large file is to divide it into smaller blocks. A practical compression method should use a fixed block size. Each block should be compressed separately, according to the basic version of the method shown in
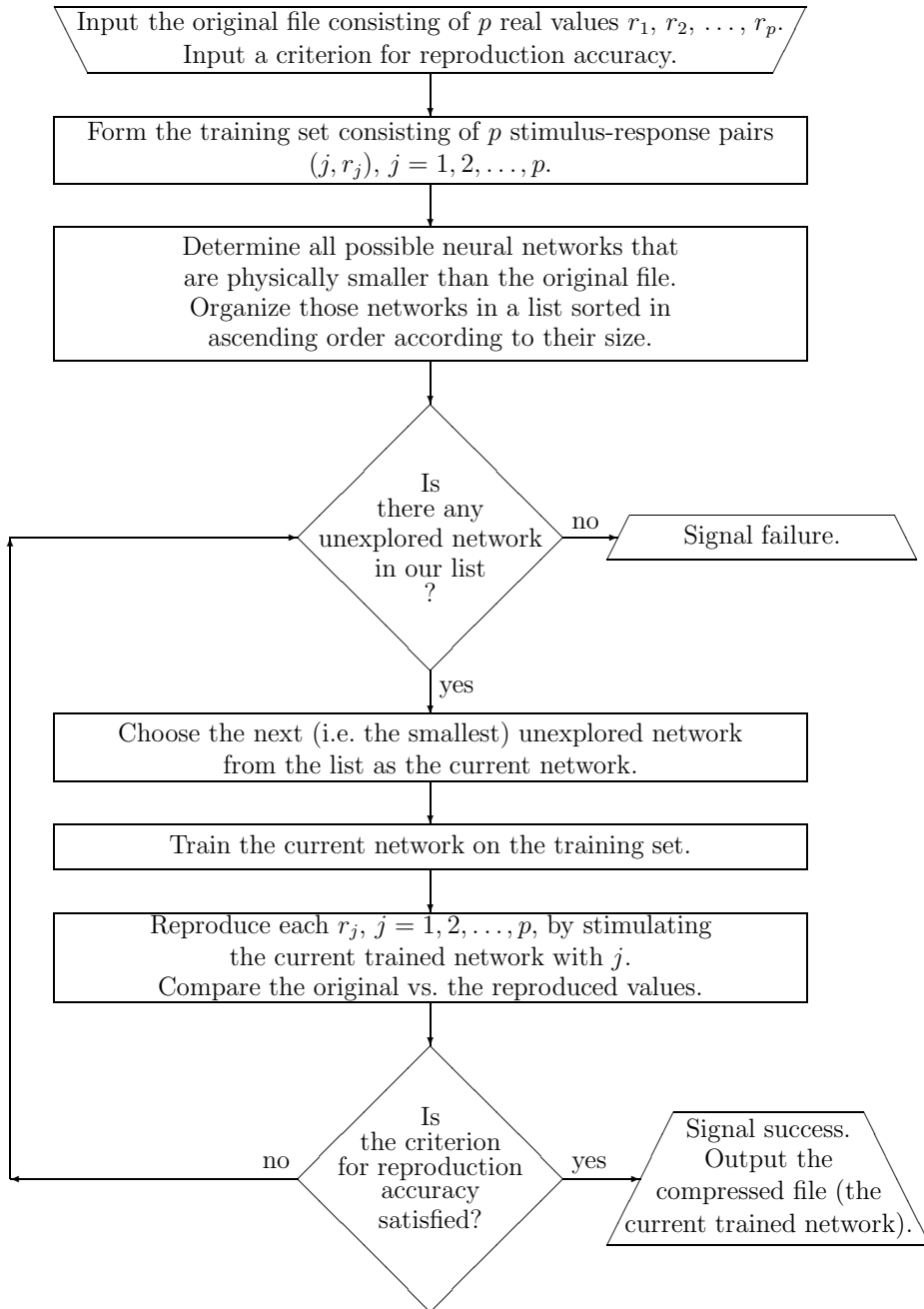
Input the original file consisting of p real values r_1, r_2, ..., r_p.
Input a criterion for reproduction accuracy.

Form the training set consisting of p stimulus-response pairs
(j, r_j), $j = 1, 2, \ldots, p$.

Determine all possible neural networks that
are physically smaller than the original file.
Organize those networks in a list sorted in
ascending order according to their size.

Is
there any
unexplored network
in our list
?

no → Signal failure.

yes

Choose the next (i.e. the smallest) unexplored network
from the list as the current network.

Train the current network on the training set.

Reproduce each r_j, $j = 1, 2, \ldots, p$, by stimulating
the current trained network with j.
Compare the original vs. the reproduced values.

Is
the criterion
for reproduction
accuracy
satisfied?

no

yes → Signal success.
Output the
compressed file (the
current trained network).

Fig. 1. Outline of a basic neural compression method

Figure 1. Or differently speaking, each block should be represented by a separate trained network. The final compressed file would be produced as the union of its compressed blocks, i.e. it would consist of the corresponding separate trained networks stored next to each other. The whole "block" version would then have linear computational complexity.

The question remains how to choose the block size for our block method. From the computational point of view, smaller blocks are better. But we must also take into account how the block size influences the overall compression capabilities. If our file exhibits some kind of regularity (redundancy), the network can "learn" that regularity and use it more efficiently to reproduce a larger block. For instance, if our file is a tabulated linear function, then the network should learn and store only one value (i.e. the slope), which represents 10 % of a 10-values block and only 1 % of a 100-values block. Thus, from the compression point of view it is safer to use larger blocks. Consequently, the method should use blocks that are small enough to assure reasonably fast computation, and still big enough to enable efficient compression.

Note that the proposed neural compression method bears some resemblance to the widely used JPEG compression standard [17]. First, our method is also *lossy*, i.e. it only approximately reproduces original data. Also, both procedures are *block* procedures, i.e. they divide a large file into smaller (equally sized) blocks that are compressed independently. Finally, our method is also *parameterizable* in the sense that accuracy can be traded for better compression.

3 USING MULTILAYER PERCEPTRONS

An artificial neural network is built of relatively simple computing elements called *neurons*. There are many types of neural networks found in literature. Particular types differ in the way how their neurons operate and how the neurons may be connected together.

In this paper we are dealing with the most popular type of neural networks: multilayer perceptrons [3, 5]. Neurons within a multilayer perceptron are normally arranged into *layers* (input, output, hidden). The input-layer neurons are considered to be degenerate: they actually have no parameters and they simply serve to distribute original stimulus values to the next-layer neurons. Any other neuron transforms data, by using its activation function f and its adjustable parameters (weights w_i, a threshold t). More precisely, for given inputs x_i, the single output z of the neuron is computed as

$$z = f(\sum_i w_i x_i - t).$$

The output-layer neurons have the "linear" activation function

$$f(y) = y,$$

while hidden neurons use the "logistic" function

$$f(y) = 1/(1 + e^{-y}).$$

All involved values are assumed to be real numbers.

To realize the neural compression method outlined in the previous section, we have used a slightly restricted family of standard multilayer perceptrons. In accordance with our particular application, we have fixed the number of neurons in both the input and output layers to 1. Since it is generally believed [13] that three or four layers can solve virtually all problems, we have limited the number of hidden layers to 2. Consequently, one particular configuration within our restricted family is determined by the number of hidden layers (0 or 1 or 2) and by the number of neurons in each of the hidden layers. Figure 2 presents all such configurations.
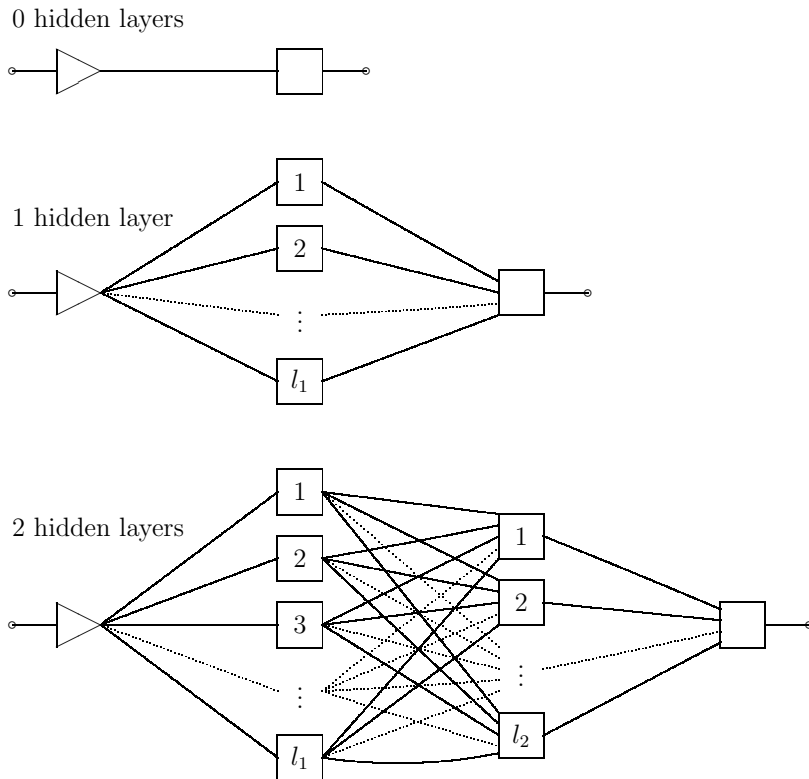
Fig. 2. A restricted family of multilayer perceptrons

For our purposes, it is very important to estimate the physical size (measured in real numbers) of any particular network. In this paper we assume that the size of a network is equal to the total number of weights and thresholds within all of its neurons. This assessment is slightly optimistic, since it does not take into account the overhead needed to record the network configuration. However, small networks from our restricted family are either uniquely determined by their number of weights and thresholds, or there are only few possible configurations. Thus, with appropriate auxiliary data, the mentioned overhead can be made negligible i.e. reduced to only few bits.

Using the above assessment, we can easily compute the size n of any considered network configuration. Namely, for the network with no hidden layers,

$$n = 2.$$

If the network has one hidden layer with l_1 neurons, then

$$n = 3l_1 + 1.$$

If the network has two hidden layers with, l_1 and l_2 neurons, respectively, then

$$n = 2l_1 + 2l_2 + l_1 l_2 + 1.$$

number of hidden layers	number of neurons in the 1st hidden layer	number of neurons in the 2nd hidden layer	size of the network
0	-	-	2
1	1	-	4
2	1	1	6
1	2	-	7
2	2	1	9
2	1	2	9
1	3	-	10
2	3	1	12
2	1	3	12
1	4	-	13
2	2	2	13
2	4	1	15
2	1	4	15
1	5	-	16
2	3	2	17
2	2	3	17
2	5	1	18
2	1	5	18
1	6	-	19

Table 1. Sorted list of all considered perceptrons whose size is less than 20

By varying network configurations and by using the above formulae, one can relatively quickly produce the sorted list of all networks whose size is below a certain limit, as required in our compression method. Table 1 shows the list of networks whose size is less than 20. The table can also be interpreted as the beginning of a similar list for any higher size limit.

It is interesting to note that our restricted family of perceptrons is still rich enough, so that it contains perceptrons which are able to reproduce any given file with any required precision. Moreover, a desired reproduction accuracy can always be accomplished by using configurations with only one hidden layer. In the following paragraph we give an informal proof of this fact.

So suppose again that we are given a file consisting of p real values $r_1, r_2, \ldots, r_p$. Let us choose the perceptron from our family, having one hidden layer and exactly p hidden neurons. With suitable weights and thresholds, logistic functions inside hidden neurons can be made "steep enough", so that they behave virtually as step functions. More precisely, it can be achieved that the ith hidden neuron produces the output ≈ 1 for an input $\geq i$, and the output ≈ 0 otherwise. With such settings, the resulting response from our perceptron to a stimulus j has the form

$$z_j \approx \sum_{i=1}^{j} w_i - t,$$

where w_i and t denote the weights and the threshold belonging to the single output-layer neuron. Let us now take $t = 0$, and set the values of w_i in turn:

$$
\begin{aligned}
w_1 &= r_1, \\
w_2 &= r_2 - w_1, \\
&\cdots \\
w_j &= r_j - \sum_{i=1}^{j-1} w_i, \\
&\cdots \\
w_p &= r_p - \sum_{i=1}^{p-1} w_i.
\end{aligned}
$$

Obviously, with these final adjustments, the response z_j to a stimulus j becomes $\approx r_j$. Thus, our perceptron correctly reproduces the given file.

The presented proof can of course be made more formal by using exact error estimates instead of approximate equalities. Rigorous proofs of similar or more general results are based on the well known Kolmogorov's theorem [3] and can be found for instance in [2, 6]. Note that although we have shown that a sufficiently large perceptron can always reproduce a given file to any desirable precision, we still cannot guarantee that such reproduction results can be realized in practice. Namely, the available training algorithms usually get stuck in local minima, thus never discovering the theoretically optimal adjustment of network weights and thresholds.

Another reason why the above described perceptron is not of much practical relevance is its size, which is several times greater than the original file size. Indeed, to achieve data compression, we actually have to use much smaller perceptrons with no guaranteed reproduction capabilities.

4 EXPERIMENTS

In order to test data compression capabilities of multilayer perceptrons, we have performed a series of experiments. We have followed the neural compression method outlined in Section 2. As the set of networks considered by the method, we have used the restricted family of multilayer perceptrons described in Section 3.

For one experiment, one particular combination of data file and perceptron was chosen. Then at least four training algorithms were tried in order to obtain the best possible results, including back propagation, conjugate gradient descent, quasi Newton, Levenberg-Marquardt, and quick propagation [3]. That gave altogether more than 4000 network training sessions. All computations were done by the *Statistica Neural Networks* software package [13]. The accumulated results of all experiments involving the same file depict the performance of our method for that file and for different accuracy criteria.

file name	description	graphical interpretation
linear	$r_j = \frac{j}{100}$, $j = 1, 2, \ldots, 100$	
step	$r_j = 0$ for $j = 1, 2, \ldots, 50$, $r_j = 1$ for $j = 51, 52, \ldots, 100$	
poly-nomial	$r_j = \frac{j(j-100)}{2500} + 1$, $j = 1, 2, \ldots, 100$	
sine	$r_j = \frac{1}{2} \sin \frac{2\pi j}{101} + \frac{1}{2}$, $j = 1, 2, \ldots, 100$	
sine+ cosine	$r_j = \frac{\sqrt{2}}{4}(\sin \frac{2\pi j}{101} + \cos \frac{2\pi j}{101}) + \frac{1}{2}$, $j = 1, 2, \ldots, 100$	

Table 2. Experimental data – regular files

We used altogether fifteen different data files from [10]. All of them had the same size $p = 100$ and consisted of real numbers r_j spanning the range between 0 and 1. Those data files should be regarded as blocks in the block version of the

file name	description	graphical interpretation
mul- tiple linear	$r_j = \frac{j}{100}$ for $1 \leq j \leq 25$, $r_j = \frac{2(j-25)}{100}$ for $26 \leq j \leq 50$, $r_j = \frac{3(j-50)}{100}$ for $51 \leq j \leq 75$, $r_j = \frac{4(j-75)}{100}$ for $76 \leq j \leq 100$	
mul- tiple step	$r_j = 0$ for $1 \leq j \leq 50$ or $76 \leq j \leq 88$ or $95 \leq j \leq 100$, $r_j = 1$ for $51 \leq j \leq 75$ or $89 \leq j \leq 94$	
amp- lified sine	$r_j = c_1 \cdot j \cdot sin\frac{4\pi j}{101} + c_2$, $c_1 = 0.006563739$, $c_2 = 0.5824075$, $j = 1, 2, \ldots, 100$	
accel- erated sine	$r_j = \frac{1}{2} \sin 4\pi (\frac{j}{101})^2 + \frac{1}{2}$, $j = 1, 2, \ldots, 100$	
modu- lated sine	$r_j = \left| \sin \frac{4\pi j}{101} \right|$, $j = 1, 2, \ldots, 100$	

Table 3. Experimental data - more demanding files

outlined compression method. Choosing the block size 100 can be regarded as an empirical rule targeted to maximize compression efficiency. Namely, the chosen size is almost too large from the computational point of view – one training session could last few minutes on a conventional PC. By working with such large blocks we tried to give more chance for successful compression.

Our fifteen data files have been chosen in order to form three different groups of five. Members of the first group are very "regular" files, obtained by tabulating elementary functions. Members of the second group are more demanding: they have been constructed by repeating or combining simple patterns used in the first group. The third group comprises extremely "irregular" files, i.e. sequences of random numbers uniformly distributed between 0 and 1. Exact file descriptions can be found in Tables 2, 3 and 4. Graphical interpretations within these tables clearly demonstrate to what extent a particular file is smooth or redundant or predictable. According to the shown graphs, one would expect that both regular and more demanding files should be compressible, and that regular files are easier to compress than more demanding ones. Also, truly random files should be incompressible by definition.

For a file of size $p = 100$, our method should in principle consider all network configurations with size $n < 100$. Actually, there exist 223 such perceptrons. How-
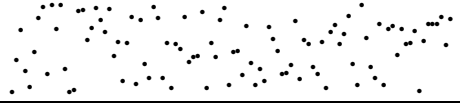
file name	description	graphical interpretation
random-1	first 100 values from a pseudo-random sequence	
random-2	101st to 200th value from the same sequence	
random-3	201st to 300th value from the same sequence	
random-4	301st to 400th value from the same sequence	
random-5	401st to 500th value from the same sequence	

Table 4. Experimental data – random files

ever, in our experiments involving regular or more demanding files (first two groups) we used only 113 networks with size up to 61. Namely, by employing these smaller networks it was already possible to reach even the sharpest reproduction accuracy criteria. Larger perceptrons were needed only for random files (third group), where we used all 223 configurations with size $n < 100$ and also few additional configurations with $n >> 100$.

To measure the current reproduction quality during training, we computed the so-called *RMS error* (root mean squared error). Thus, we collected squares of differences among original and reproduced file values, computed the mean value of those squares, and then square rooted the mean. Since our files consisted of values between 0 and 1, the chosen error measure can also be interpreted as "relative to full range". For instance, error 0.0100 simply means that the expected difference between an original value and its corresponding reproduced value is about 1 % of full range.

5 RESULTS

The more attractive part of experimental results has been summarized in Tables 5 and 6. Table 5 indicates how the outlined compression method with multilayer perceptrons behaves on regular data files (first group). Table 6 illustrates the behaviour on more demanding files (second group). Each row corresponds to combination of one particular file with one particular criterion for reproduction accuracy. The cho-

sen criteria simply demand that the RMS error should be below a certain limit. The results for one row are presented by quoting the corresponding compression rate, and by specifying the involved optimal network configuration.

file	RMS	com-	optimal network configuration:		
name	error $\leq$	pression rate	# of hidden layers	# of neurons in 1st hid. lay	# of neurons in 2nd hid. lay
linear	0.0100	2 %	0	–	–
	0.0010	2 %	0	–	–
	0.0001	2 %	0	–	–
step	0.0100	4 %	1	1	–
	0.0010	4 %	1	1	–
	0.0001	4 %	1	1	–
poly-	0.0100	7 %	1	2	–
nomial	0.0010	7 %	1	2	–
	0.0001	7 %	1	2	–
sine	0.0100	7 %	1	2	–
	0.0010	10 %	1	3	–
	0.0001	10 %	1	3	–
sine+	0.0100	7 %	1	2	–
cosine	0.0010	10 %	1	3	–
	0.0001	12 %	2	3	1

Table 5. Summary results – experiments with regular files

The *compression rate* is defined rather conventionally, as the ratio n/p expressed as a percentage, where n denotes the size of the involved optimal perceptron (i.e. the size of the obtained compressed file), and p is again the size of the original file. Both sizes are measured in the same units (i.e. in real numbers). Thus, our compression rate specifies the compressed file size relatively in terms of the original file size.

As we see from Table 5, our compression method with multilayer perceptrons can very successfully compress regular files. According to Table 6, the method is less successful in compressing more demanding files, although the results are still acceptable. It is very hard to determine which of the demanding files are "regular enough" to be efficiently compressed and which are not.

Tables 5 and 6 also reveal that optimal network configurations are sometimes based on one and sometimes on two hidden layers. As expected, the configuration with no hidden layers is useful only in one exceptional case. The presented results were obtained by combining different training algorithms. However, most accurate and efficient training was usually provided by Levenberg-Marquardt [3].

Some of the results from Table 6 are illustrated in more detail by Figures 3 and 4. Both figures correspond to the relatively demanding but still compressible "accelerated sine" file, defined by Table 3, row 4. Figure 3 shows precisely how the RMS error depends on the compression rate - note that the values of the RMS error

file	RMS	com-	optimal network configuration:		
name	error	pression	# of	# of	# of
	$\leq$	rate	hidden	neurons in	neurons in
			layers	1st hid. lay	2nd hid. lay
mul-	0.0100	16 %	1	5	–
tiple	0.0010	21 %	2	4	2
linear	0.0001	29 %	2	6	2
mul-	0.0100	12 %	2	3	1
tiple	0.0010	12 %	2	3	1
step	0.0001	12 %	2	3	1
amp-	0.0100	10 %	1	3	–
lified	0.0010	13 %	1	4	–
sine	0.0001	13 %	1	4	–
accel-	0.0100	10 %	1	3	–
erated	0.0010	13 %	2	2	2
sine	0.0001	19 %	1	6	–
modu-	0.0100	37 %	1	12	–
lated	0.0010	55 %	1	18	–
sine	0.0001	60 %	2	7	5

Table 6. Summary results – experiments with more demanding files

are plotted on the logarithmic scale. Figure 4 compares the original file values with the reproduced values obtained after compressing the file to 10 % of its original size (RMS error $\leq$ 0.01).

In the next part of this section we report on our attempts to compress random data files (third group). The results dealing with random files are less encouraging than those dealing with regular or more demanding files. Namely, our experiments have proved that the considered perceptrons of size $n < 100$ are not able to reproduce a random file of size $p = 100$ with any satisfactory accuracy. Or, differently speaking, our outlined compression method signals failure when applied to a random file.

It is certainly true that any compression procedure should fail on random files. Our method based on multilayer perceptrons makes no exception to this rule, thus its failure has been expected. Still, the question remains how far a certain method is from being successful. Or in our case, we could ask ourselves how large a perceptron should be in order to be able to reproduce a random file.

To answer the above question, we have done the previously mentioned additional series of experiments using perceptrons whose size is above 100. The results of those additional experiments are presented in Table 7, which is structured similarly as Tables 5 and 6. According to Table 7, a perceptron capable of reproducing a random file with low accuracy (RMS error $\leq$ 0.01) should be 50 % larger than the original file. For high accuracy (RMS error $\leq$ 0.0001) we would need a perceptron with size about 300 % of the original file size. Medium accuracy (RMS error $\leq$ 0.001) is achieved somewhere between the size limits of 150 % and 300 %.
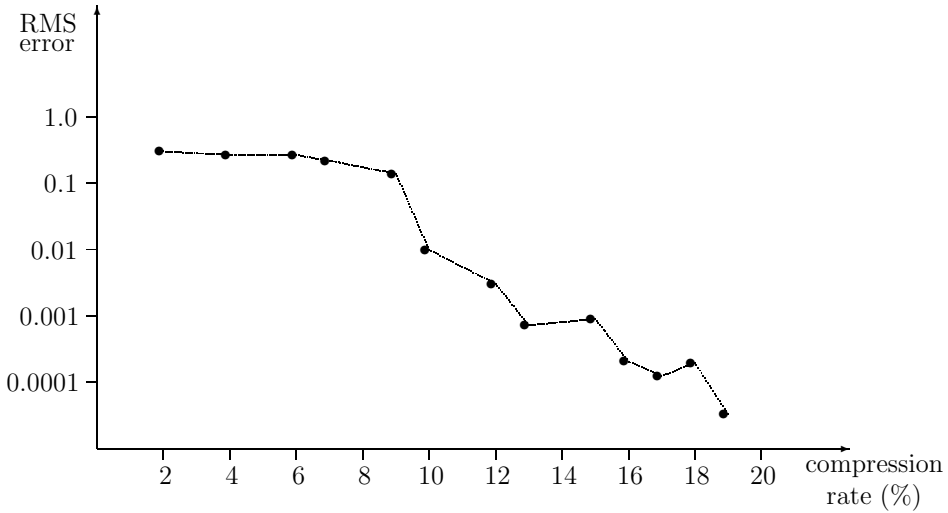
Fig. 3. Detailed results – experiments with the "accelerated sine" file

The results shown in Table 7 have been obtained again by using the Levenberg-Marquardt training algorithm. The recorded size estimates are only approximate – namely it was not possible to do an exhaustive test of all large perceptrons due to combinatorial explosion of configurations and prohibitive training costs. Note that the established size limit of 300 % corresponds exactly to the perceptron constructed within our informal reproduction-capability proof in Section 3.
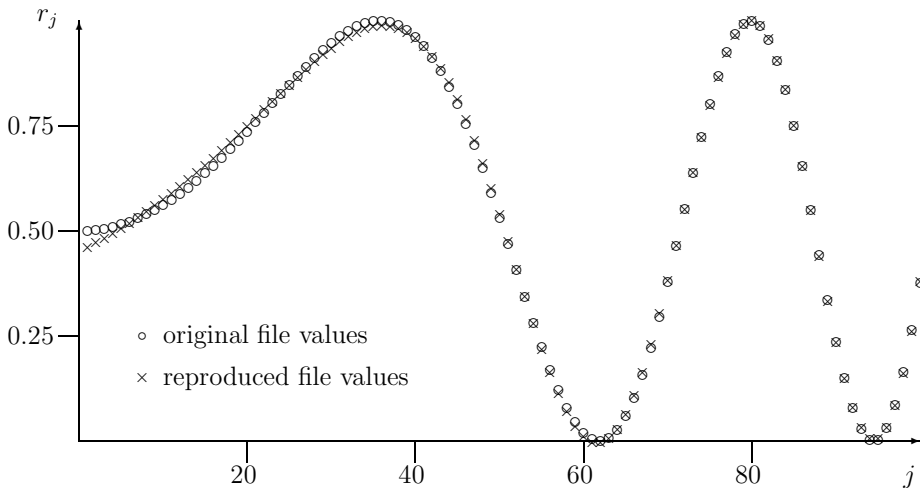

Fig. 4. More details – accelerated sine, compression rate 10 % (RMS error ≤ 0.01)

file name	RMS error $\leq$	com- pression rate	optimal network configuration:		
			# of hidden layers	# of neurons in 1st hid. lay	# of neurons in 2nd hid. lay
random-1	0.0100	151 %	1	50	–
	0.0010	211 %	1	70	–
	0.0001	301 %	1	100	–
random-2	0.0100	181 %	1	60	–
	0.0010	201 %	2	15	10
	0.0001	271 %	1	90	–
random-3	0.0100	151 %	1	50	–
	0.0010	201 %	2	15	10
	0.0001	301 %	1	100	–
random-4	0.0100	151 %	1	50	–
	0.0010	211 %	1	70	–
	0.0001	271 %	1	90	–
random-5	0.0100	121 %	1	40	–
	0.0010	261 %	2	20	10
	0.0001	286 %	2	15	15

Table 7. Summary results – experiments with random files

In the last part of the section we compare the results presented in this paper with similar results presented in the previous paper [10]. For each value in Tables 5 and 6, there is a corresponding value in [10]. By direct comparison of the corresponding values, we can see that the compression rates for regular and more demanding files obtained with multilayer perceptrons are in general better than those obtained with holographic neural networks. More precisely, the results from this paper are considerably better in 22 cases out of 30, and equal in 4 cases. Holographic networks perform better only on the "sine" file (Table 2, row 4), and partially on the "modulated sine" file (Table 3, row 5). A similar advantage of perceptrons vs. holographic networks can also be observed when dealing with random files. Namely, by comparing the values in Table 7 with the corresponding values in [10], we observe that a large perceptron capable of reproducing a random file with certain precision is in most cases smaller than the corresponding holographic network (typically 150 % vs. 180 % of the original file size, or 300 % vs. 400 %). Note that the two papers use the same data files but slightly different error measures. Still, the involved error measures have the same orders of magnitude, thus allowing fair comparison.

6 CONCLUSION

The primary aim of this paper has been to measure data compression capabilities of standard multilayer perceptrons, and to compare those capabilities with holographic networks. The results presented in the paper indicate that multilayer per-

ceptrons assure better compression rates than holographic networks, which is quite in contradiction with the claims given by some proponents of the holographic neural technology.

The secondary aim of this paper has been to outline a possible neural compression method. According to the presented results, such a method is possible, but it should be based on ordinary multilayer perceptrons rather than on holographic networks. Also, such a method can work well only if the file to be compressed is in some sense regular (smooth, redundant, predictable). Through the process of training, the involved perceptron discovers such regularity, and uses it for compression better than a holographic network would do. For random files, the method with multilayer perceptrons produces a substantial overhead in file size, but this overhead is still smaller than with holographic networks.

It is interesting to note that our approach to compression, based on discovering regularities, fits nicely into the well known Minimum Description Length (MDL) principle [4]. The fundamental idea behind MDL is that any regularity in a given set of data can be used to compress the data, i.e. to describe it using fewer symbols than needed to describe the data literally. Our neural compression method can be considered as a tool for automatic implementation of the MDL principle.

At this moment, we are still quite far from a robust and practically useful version of the outlined perceptron-based compression method. The development of such a version shall be the topic of our future research. The most important open problem to be solved is how to reduce computational complexity, which is at present prohibitive even if blocks are used. One approach to deal with computational complexity would be to choose a more restricted family of promising perceptrons in order to reduce searching for optimal configuration. Also, the method would surely be more efficient if dedicated and faster training algorithms could be designed.

Finally, let us note that the idea of using machine learning for the aim of data compression could as well be extended to other learning paradigms. It is quite conceivable that, for instance, regression trees [1] would deal better with irregular or random data than neural networks. Such possibilities should also be investigated in future papers.

REFERENCES

[1] BREIMAN, L. et al.: Classification and Regression Trees. Chapman & Hall, New York, NY, 1984.

[2] CYBENKO, G.: Approximation by Superpositions of a Sigmoidal Function. Mathematics of Control, Signals, and Systems, Vol. 2, 1989, pp. 303–314.

[3] FINE, T. L.: Feedforward Neural Network Methodology. Springer Series in Statistics, Springer Verlag, New York, NY, 1999.

[4] GRÜNWALD, P. D.: Introducing the Minimum Description Length Principle. In: P. D. Grünwald, I. J. Myung and M. A. Pitt (Eds.): Advances in Minimum Description Length: Theory and Applications, MIT Press, Cambridge, MA, 2005, pp. 3–22.

[5] HAYKIN, S.: Neural Networks – A Comprehensive Foundation. Second Edition. Prentice Hall, Englewood Cliffs, NJ, 1998.

[6] HORNIK, K.—STINCHCOMBE, M.—WHITE, H.: Universal Approximation of an Unknown Mapping and its Derivatives Using Multilayer Feedforward Networks. Neural Networks, Vol. 3, 1990, pp. 551–560.

[7] KOUDA, N.—MATSUI, N.—NISHIMURA, H.: Image Compression by Layered Quantum Neural Networks. Neural Processing Letters, Vol. 16, 2002, No. 1, pp. 67–80.

[8] LASKARIS, N. A.—FOTOPOULOS, S.: A Novel Training Scheme for Neural-Network-Based Vector Quantizers and its Application in Image Compression. Neurocomputing, Vol. 61, 2004, pp. 421–427.

[9] LOGESWARAN, R.: Fast Two-Stage Lempel-Ziv Lossless Numeric Telemetry Data Compression Using a Neural Network Predictor. The Journal of Universal Computer Science, Vol. 10, 2004, No. 9, pp. 1199–1211.

[10] MANGER, R.: Holographic Neural Networks and Data Compression. Informatica, Vol. 21, 1997, No. 4, pp. 665–673.

[11] MEYER-BÄSE, A. ET AL.: Medical Image Compression Using Topology-Preserving Neural Networks. Engineering Applications of Artificial Intelligence, Vol. 18, 2005, No. 4, pp. 383–392.

[12] SAYOOD, K.: Introduction to Data Compression. Second Edition, Morgan Kaufmann, San Francisco, CA, 2000.

[13] Statistica Neural Networks (user manual). StatSoft Inc, Tulsa, OK, 1998.

[14] SUN, K. T.—LEE, S. J.—WU, P. Y.: Neural Network Approaches to Fractal Image Compression and Decompression. Neurocomputing, Vol. 41, 2001, Nos. 1–4, pp. 91–107.

[15] SUTHERLAND, J. G.: Holographic Model of Memory, Learning and Expression. International Journal of Neural Systems, Vol. 1, 1990, No. 3, pp. 256–267.

[16] SUTHERLAND, J. G.: The Holographic Cell: a Quantum Perspective. In: V. L. Plantamura, B. Souček and G. Visaggio (Eds.): Frontier Decision Support Concepts. John Wiley and Sons, New York, NY, 1994, pp. 65–78.

[17] TAUBMAN, D. S.—MARCELLIN, M. W.: JPEG 2000: Image Compression Fundamentals, Standards and Practice. Kluver International Series in Engineering and Computer Science, 642. Kluwer Academic Publishers, Dordrecht, NL, 2001.

Robert MANGER received the B. Sc. (1979), M. Sc. (1982), and Ph. D. (1990) degrees in mathematics, all from the University of Zagreb. For more than ten years he worked in industry, where he gained experience in programming, computing, and designing information systems. He is presently an associate professor at the Department of Mathematics, University of Zagreb. His current research interests include: parallel and distributed algorithms, combinatorial optimization, and neural networks. He has published 15 papers in international scientific journals, over 20 scientific papers in conference proceedings, 10 professional papers, and 3 course materials. He is a member of the Croatian Mathematical Society, Croatian Society for Operations Research and IEEE Computer Society.

Krunoslav PULJIĆ received his B. Sc. and M. Sc. degrees in mathematics from the University of Zagreb in 1999 and 2005, respectively. He is now a research assistant at the Department of Mathematics, University of Zagreb, and is working on his Ph. D. thesis. His research interests include combinatorial optimization, evolutionary algorithms and object-oriented software engineering. He has published 2 scientific papers in conference proceedings. He is a member of the Croatian Mathematical Society.