PROBEN1 — A Set of Neural Network Benchmark Problems and Benchmarking Rules

Lutz Prechelt (prechelt@ira.uka.de)
Fakultät für Informatik
Universität Karlsruhe
76128 Karlsruhe, Germany
++49/721/608-4068, Fax: ++49/721/694092

September 30, 1994

Technical Report 21/94

Abstract

PROBEN1 is a collection of problems for neural network learning in the realm of pattern classification and function approximation plus a set of rules and conventions for carrying out benchmark tests with these or similar problems. PROBEN1 contains 15 data sets from 12 different domains. All datasets represent realistic problems which could be called *diagnosis tasks* and all but one consist of real world data. The datasets are all presented in the same simple format, using an attribute representation that can directly be used for neural network training. Along with the datasets, PROBEN1 defines a set of rules for how to conduct and how to document neural network benchmarking.
The purpose of the problem and rule collection is to give researchers easy access to data for the evaluation of their algorithms and networks and to make direct comparison of the published results feasible. This report describes the datasets and the benchmarking rules. It also gives some basic performance measures indicating the difficulty of the various problems. These measures can be used as baselines for comparison.

Contents

List of Tables

1 Introduction

This section discusses why standardized datasets and benchmarking rules for neural network learning are necessary at all, what the scope of PROBEN1 is, and why real data should be used instead of or in addition to artificial problems as they are often used today.

1.1 Why a benchmark set?

A recent study of the evaluation performed in journal papers about neural network learning algorithms [15] showed that this aspect of neural network research is a rather poor one. Most papers present performance results for the new algorithm only for a very small number of problems — rarely more than three. In most cases, one or several of these problems are meaningless synthetic problems, for instance from the parity/symmetry/encoder family. Comparisons to algorithms suggested by other researchers are in many cases not done at all (exception: standard backpropagation).

Why is this so? Several explanations (read: excuses) are possible:

1. Since training a neural network usually takes quite long, a thorough evaluation takes a very large amount of CPU time.
2. The algorithms of other researchers are often not available as programs at all or their implementations are not stable or are based on some exotic environment.
3. It is difficult to get data for real problems.
4. It is much work to prepare data for neural network training.
5. Even results obtained for the same problem can often not be compared directly because of different problem representations or different experimental setups.

None of these arguments, however, is still a valid one today. I discuss them in order:

1. Not really a problem. Our machines are fast enough now to do a significant amount of training runs within a few days — at least for small or moderately large datasets. And, hey!, it's your computer that must do the work, not you!
2. Yes, often true. And probably nothing that we can easily avoid. However, it would not be a problem if we could just compare against the results of other researchers directly by making the corresponding experiment with a new algorithm.
3. Only partially true. Many researchers who have used real data in their research are willing to give it to others upon request. There are also publicly accessible collections of such data; most notably the UCI machine learning databases repository.
4. Correct. It really is. But not everybody needs to make that data preparation again. We as a research community can and should share the results of such work.
5. This is a real problem that comes in three variants: First, sometimes experimental setups are just plain wrong, giving invalid results. Second, often experimental setups are not documented properly in the papers published, making reproduction or exact comparison impossible. Third, often the documentation just *looks* obscure, because the same things are expressed in very different ways by different people. We need a standard set of conventions for our experiments and their documentation in order to fight this problem.

As we see from this discussion, there is a need for standard sets of problems and rules or conventions for applying them to be used in learning algorithm evaluations. PROBEN1 is meant as a first step towards a set of standard benchmarks for some areas of neural network training algorithm research.

Its availability lays ground for better algorithm evaluations (by enabling easy access to example data of real problems) and for better comparability of the results (if everybody uses the same problems and setup) — while at the same time reducing the workload for the individual researcher.

Aspects of learning algorithms that can be studied using PROBEN1 are for example learning speed, resulting generalization ability, ease of user parameter selection, and network resource consumption. What cannot be assessed well using a set of problems with fixed representation (such as PROBEN1) are all those aspects of learning that have to do with the selection or creation of a suitable problem representation.

Lack of standard problems is widespread in many areas of computer science. At least in some fields, though, standard benchmark problems exist and are used frequently. The most notable of these positive examples are performance evaluation for computing hardware and for compilers. For many other fields it is clear that defining a reasonable set of such standard problems is a very difficult task — but neural network training is not one of them.

1.2 Why benchmarking rules?

It is clear from the discussion above that having a standard set of benchmark problems is, although necessary, not sufficient to improve the de-facto scientific quality of our evaluations. A real improvement is made only if the results published for these benchmark problems are comparable and reproducible. This is not trivial, though, since every application of a neural network training algorithm to a particular problem involves a significant number of user selectable parameters of various kinds — often more than a dozen. If but one of these parameters is not published along with the result, the experiment becomes irreproducible and the comparability of the results is hampered. Even if all parameters are published, comparability might still be an issue due to the fact that many descriptions are ambiguous since we are lacking a standard terminology.

Thus, a set of benchmark problems should be complemented by a set of benchmarking rules (or benchmarking conventions, if you want) that describe and standardize ways of setting up experiments, documenting these setups, measuring results, and documenting these results. Such rules need not reduce the freedom of choosing among several possible experimental setups — they just suggest a core standard that should be used in order to maximize comparability of experimental results and show what should be documented in which way when one deviates from that standard.

As a side-effect, thoroughly documented benchmarking rules reduce the danger that a researcher makes a major fault in his or her experimental setup, thereby producing invalid results.

1.3 Scope of Proben1

Neural network learning algorithm research is a wide field trying to tackle many different classes of problems. Many subfields, such as machine vision, optical character recognition, or speech recognition, are quite specialized and hence also require specialized benchmarks. Other fields require or forbid certain properties to be present in any benchmark problem to be used. Thus, no single set of benchmark problems can be usable for the evaluation of research in the whole field.

The scope of the PROBEN1 problems can be characterized as follows. All problems are suited for supervised learning, since input and output values are separated. All problems are suited for use with networks that do not maintain an internal state, since all examples within a problem are independent of each other. Most of the problems can be tackled by pattern classification algorithms, while a few others need the capability of continuous multivariate function approximation. Most problems have both continuous and binary input values. All problems are presented as static problems in the sense that all data to learn from is present at once and does not change during learning. All problems except one (the mushroom problem) consist of real data from real problem domains.

The common properties of the learning tasks themselves are characterized by them all being what I call *diagnosis tasks*. Such tasks can be described as follows:

1. The input attributes used are similar to those that a human being would use in order to solve the same problem.
2. The outputs represent either a classification into a small number of understandable classes or the prediction of a small set of understandable quantities.
3. In practice, the same problems *are* in fact often solved by human beings.
4. Examples are expensive to get. This has the consequence that the training sets are not very large.
5. Often some of the attribute values are missing.

The scope of the PROBEN1 rules can be characterized as follows. The rules are meant to apply to all supervised training algorithms. Their presentation, however, is biased towards the training of feed forward networks with gradient descent or similar algorithms. Hence, some of the aspects mentioned in the rules do not apply to all algorithms and some of the aspects relevant to certain algorithms have been left out. The rules suggest certain choices for particular aspects of experimental setups as standard choices and say how to report such choices and the results of the experiments.

Both parts of PROBEN1, problems as well as rules, cover only a small part of neural network learning algorithm research. Additional collections of benchmark problems are needed to cover more domains of learning (e.g. application domains such as vision, speech recognition, character recognition, control, time series prediction; learning paradigms such as reinforcement learning, unsupervised learning; network types such as recurrent networks, analog continuous-time networks, pulse frequency networks. Sufficient benchmarks available today for only a few of these fields). Additions and changes to the rules will also be needed for most of these new domains, learning paradigms, and network types. This is why the digit 1 was included in the name of PROBEN1; maybe some day PROBEN100 will be published and the field will be mature.

1.4 Why no artificial benchmarks?

In the early days of the current era of neural network research (i.e., during the second half of the 1980s), most benchmark problems used were artificial. The most famous one of these is the XOR problem. Its popularity originates from the fact that being able to solve it was the great breakthrough (achieved by the error back-propagation algorithm), compared to the situation faced during the first era of neural network research in the 1960s when no learning algorithm was known to solve a not linearly separable classification task such as XOR.

Other training problems that were often used in the 1980s are the generalized XOR problem (n-bit parity), the n-bit encoder, the symmetry problem, the T-C problem, the 2-clumps problem, and others [3, 18]. Their deficiencies are known: all of these problems are purely synthetic and have strong a-priori regularities in their structure; for some of them it is unclear how to measure in a meaningful way the generalization capabilities of a network with respect to the problem; most of the problems can be solved 100% correct, which is untypical for realistic settings.

Later works used still other synthetic problems which can not be exactly solved so easily. Instances are the two spirals problem [4, 5, 10] or the three discs problem [19]. The problem with these problems is, similar to the ones mentioned above, that we know a-priori that a simple exact solution exists — at least when using the right framework to express it. It is unclear, how this property influences the observed capability of a learning algorithm or network to find a good solution: some algorithms may be biased towards the kind of regularity needed for a good solution of these problems and will do very good on these benchmarks, although other algorithms not having such bias would be better in more realistic domains.

Summing up, we can conclude that the main problem with the early artificial benchmarks is that we do not know what the results obtained for them tell us about the behavior of our systems on real world tasks.

One way to transcend this limitation is to make the data generation process for the artificial problems resemble realistic phenomena. The usual way to do that is to replace or complement the data generation based on a simple logic or arithmetic formula by stochastic noise processes and/or by realistic models of physical phenomena. Compared to the use of real world data this has the advantage that the properties of each dataset are known, making it easier to characterize for what kinds of problems (i.e., dataset characteristics) a particular algorithm works better or worse than another.

Two problems are left by this approach. First, there is still the danger to prefer algorithms that happen to be biased towards the particular kind of data generation process used. Imagine classification of datasets of point clouds generated by multidimensional gaussian noise using a gaussian-based radial basis function classifier. This can be expected to work very well, since the class of models used by the learning algorithm is exactly the same as the class of models employed in the data generation.[1]

Second, it is often unclear what parameters for the data generation process are representative of real problems in any particular domain. When overlaying a functional and a noise component, the questions to be answered are how strong the non-linear components of the function should be, how strong and of what type the non-linearities in that components should be, and what amount of noise of which type should be added. Choosing the wrong parameters may create a dataset that does not resemble *any* real problem domain.

Clearly artificial datasets based on realistic models and real data sets both have their place in algorithm development and evaluation. A reason for prefering real data over artificially generated data is that the former choice guarantees to get results that are relevant for at least a *few* real domains, namely the ones being tested. Multiple domains must be used in order to increase the confidence that the results obtained did not occur due to a particular domain selection only.

1.5 Related work

Despite the high importance of benchmarks, little is done on the field for neural networks. The only public benchmark collection available that is specifically meant for neural network research is the *Neural Bench* collection at Carnegie Mellon University maintained by Scott Fahlman and collaborators (anonymous ftp to `ftp.cs.cmu.edu`, directory `/afs/cs/project/connect/bench`). Although it was created years ago, it still contains only four sets of data from real world problems.

The only larger collection of benchmark learning problems is the UCI machine learning databases archive (anonymous ftp to `ics.uci.edu`, directory `/pub/machine-learning-databases`). This archive is maintained at the University of California, Irvine, by Patrick M. Murphy and David W. Aha. It contains several dozens of problems, some in multiple variants. The problems in this archive are

[1] My personal impression is that some researchers do this consciously: they make the data generation fit to the known bias of the algorithm they advocate in order to get better results.

meant for general machine learning programs; most of them cannot readily be learned by neural networks because an encoding of nominal attributes and missing attribute values has to be chosen first.

In both collections, the individual datasets themselves were donated by various researchers. With a few exceptions, no partitioning of the dataset into training and test data is defined in the archives. In no case a sub-partitioning of training data into training set and validation set is defined. The different variants that exist for some of the datasets in the UCI archive create a lot of confusion, because it is often not clear which one was used in an experiment. The PROBEN1 benchmark collection contains datasets that are taken from the UCI archive (with one exception). The data is, however, encoded for direct neural network use, is pre-partitioned into training, validation, and test examples, and is presented in a very exactly documented and reproducible form.

Zheng's benchmark [23], which I recommend everybody to read, does not include its own data, but defines a set of 13 problems, predominantly from the UCI archive, to be used as a benchmark collection for classifier learning algorithms. The selection of the problems is made for good coverage of a taxonomy of classification problems with 16 two- or three-valued features, namely type of attributes, number of attributes, number of different nominal attribute values, number of irrelevant attributes, dataset size, dataset density, level of attribute value noise, level of class value noise, frequency of missing values, number of classes, default accuracy, entropy, predictive accuracy, relative accuracy, average information score, relative information score. The PROBEN1 benchmark problems have not explicitly been selected for good coverage of all of these aspects. Nevertheless, for most of the aspects a good diversity of problems is present in the collection.

2 Benchmarking rules

This section describes

- how to conduct valid benchmark tests and
- how to publish them and their results.

The purpose of the rules is to ensure the validity of the results and reproducibility by other researchers. An additional benefit of standardized benchmark setups is that results will more often be directly comparable.

2.1 General principles

The following general principles guide the formulation of the benchmarking rules:

Validity: We need a minimum standard of experimentation that guarantees that the results obtained are valid in the sense that they are not artifacts created by random factors or by a faulty experimental setup. Invalid results are useless. The PROBEN1 benchmarking rules thus contain a number of DOs and DON'Ts to follow in order to avoid invalid results (although following the rules cannot *guarantee* validity of the results).

Reproducibility: The rules prescribe to specify all those aspects of the experimental setup that are needed for other researchers to repeat the experiments. Results that cannot be reproduced are no scientific results. The PROBEN1 benchmarking rules thus attempt to list the relevant aspects of a benchmarking setup that need to be published to attain reproducibility. For many of these aspects, standard formulations are suggested in order to simplify presentation and comprehension.

Comparability: It is very useful if one can compare results obtained by different researchers directly. This is possible if the same experimental setup is used. The rules hence suggest a number of so called *standard choices* for experimental setups that are recommended to be used unless specific reasons stand against it. The use of such standard choices reduces the variability of benchmarking setups and thus improves comparability of results across different publications.

In the rules below, phrases typeset in sans serif font like this indicate suggested formulations to be used in publications in order to reduce the ambiguity of setup descriptions. The following sections present the PROBEN1 benchmarking rules.

2.2 Benchmark problem used

For each benchmark problem X that you use, indicate exactly what X is. In the case of a PROBEN1 problem, just give its name, e.g. hearta. In other cases, specify how and where other researchers can get the problem dataset. Sometimes this can be done by giving a reference to a paper published earlier. Otherwise a file containing the dataset should be available for anonymous FTP somewhere and you should give the FTP address that must be used to get the dataset. If you prepare your own datasets, make them available publicly by FTP if possible. If you use problems from PROBEN1, just cite this report.

Often researchers use a problem that has been used several times before and refer to it by a natural language name, for instance "A test was made using Michalski's soybean data". Such kinds of references often result in confusion, because several different versions of the data exist. So please either refer to a named problem from a well-documented benchmark collection such as PROBEN1 or give the address of a data file available by FTP or reference a paper that does so.

2.3 Training set, validation set, test set

The data used for performing benchmarks on neural network learning algorithms must be split into at least two parts: one part on which the training is performed, called the training data, and another part on which the performance of the resulting network is measured, called the test set. The idea is that the performance of a network on the test set estimates its performance in real use. This means that absolutely no information about the test set examples or the test set performance of the network must be available during the training process; otherwise the benchmark is invalid.

In many cases the training data is further subdivided. Some examples are put into the actual training set, others into a so-called validation set. The latter is used as a pseudo test set in order to evaluate the quality of a network during training. Such an evaluation is called cross validation; it is necessary due to the overfitting (overtraining) phenomenon: For two networks trained on the same problem, the one with larger training set error may actually be *better*, since the other has concentrated on peculiarities of the training set at the cost of losing much of the regularities needed for good generalization [7]. This is a problem in particular when not very many training examples are available.

A popular and very powerful form to use cross validation in neural networks is early stopping: Training proceeds not until a minimum of the error on the training set is reached, but only until a minimum of the error on the validation set is reached during training. Training is stopped at this point and the current network state is the result of the training run. Note that the actual procedure is a bit more complicated since there may be many local minima in the validation set error curve and since in order to recognize a minimum one has to train until the error rises again, so that resetting the network to an earlier state is needed in order to actually stop at the minimum. See section 3.3 for a more concrete description. Other forms of cross validation besides early stopping are also possible. The data of the validation set could be used in any way during training since it is part of the training data. The actual name 'validation set', however, is only appropriate if the set is used to assess the generalization performance of the network. Note the differentiation: **training data** is the union of **training set** and **validation set**.

Be sure to specify exactly which examples of a dataset are used for the training, validation, and test set. It is insufficient to indicate the number of examples used for each set, because it might make a significant difference *which* ones are used where. As a drastic example think of a binary classification problem where only examples of one class happen to be in the training data.

For PROBEN1, a suggested partitioning into training, validation, and test set is given for each dataset. The size of the training, validation, and test set in all PROBEN1 data files is 50%, 25%, and 25% of all examples, respectively. Note that this percentage information is not sufficient for an exact determination of the sets unless the total number of examples is divisible by four. Hence, the header of each PROBEN1 data file lists explicitly the number of examples to be used for each set. Assume that these numbers are X, Y, and Z. Then the standard partitioning is to use the first X examples for the training set, the following Y examples for the validation set and the final Z examples for the test set. If no validation set is needed, the training set consists of the first $X + Y$ examples instead.

As said before, for problems with only a small number of examples, results may vary significantly for different partitionings (see also the results presented below in section 3.3). Hence it improves the significance of a benchmark result when different partitionings are used during the measurements and results are reported for each partitioning separately. PROBEN1 supports this approach. It contains three different permutations of each dataset. For instance the problem `glass` is available in three datasets `glass1, glass2`, and `glass3`, which differ only in the ordering of examples, thereby defining three different partitionings of the `glass` problem data. Additional partitionings (although not completely independent ones) are defined by the following rules for the order of examples in the dataset file:

`a` training set, validation set, test set.
`b` training set, test set, validation set.
`c` validation set, training set, test set.
`d` validation set, test set, training set.
`e` test set, validation set, training set.
`f` test set, training set, validation set.

This list is to be read as follows: From a partitioning, say `glass1`, six partitionings can be created by re-interpreting the data into a different order of training, validation, and test set. For instance `glass1d` means to take the data file of `glass1` and use the first 25% of the examples for the validation set, the next 25% for the test set, and the final 50% for the training set. Obviously, when no validation set is used, `a` is the same as `c` and `e` is the same as `f`, thus only `a, b, d`, and `e` are available. `glass1a` is identical to `glass1`. The latter is the preferred name when none of `b` to `f` are used in the same context.

Note that these partitionings are of lower quality than those created by the permutations 1 to 3, since the latter are independent of each other while the former are not. Therefore, the additional partitionings should be used only when necessary; in most cases, just using `xx1, xx2`, and `xx3` for each problem `xx` will suffice.

If you want to use a different partitioning than these standard ones for a PROBEN1 problem, specify exactly how many examples for each set you use. If you do not take them from the data file in the order training examples, validation examples, test examples, specify the rule used to determine which examples are in which set. Examples: glass1 with 107 examples used for the training set and 107 examples used for the test set for a standard order but nonstandard size of the sets or glass1 with even-numbered examples used for the training set and odd-numbered examples used for the test set, the first example being number 0 for a nonstandard size and order of sets. If you use the PROBEN1 conventions, just say glass1 and mention somewhere in your article that your benchmarks conform to the PROBEN1 conventions, e.g. All benchmark problems were taken from the Proben1 benchmark set; the standard Proben1 benchmarking rules were applied.

An imprecise specification of the partitioning of a known data set into training, validation and test set is probably the most frequent (and the worst) obstacle to reproducibility and comparability of published neural network learning results.

2.4 Input and output representation

How to represent the input and output attributes of a learning problem in a neural network implementation of the problem is one of the key decisions influencing the quality of the solutions one can obtain. Depending on the kind of problem, there may be several different kinds of attributes that must be represented. For all of these attribute kinds, multiple plausible methods of neural network representation exist. We will now discuss each attribute kind and some common methods to represent such an attribute.

Real-valued attributes are usually rescaled by some function that maps the value into the range $0 \dots 1$ or $-1 \dots 1$ in a way that makes a roughly even distribution within that range. They are represented either by a single network input or by a range of inputs using a topological encoding (e.g. overlapping gaussian receptive fields). PROBEN1 always uses a single input for a real-valued attribute, the rescaling function is always linear (with only one exception where the logarithm is used).

Integer-valued attributes are most often handled as if they were real-valued. If the number of different values is only small, one of the representations used for ordinal attributes may also be appropriate. Note that often attributes whose values are integer numbers are not really integer-valued but are ordinal or cardinal instead. PROBEN1 treats all integer-valued attributes as real-valued.

Ordinal attributes with m different values are either mapped onto an equidistant scale making them pseudo-real-valued or are represented by $m-1$ inputs of which the leftmost k have value 1 to represent the k-th attribute value while all others are 0. A binary code using only $\lceil \log_2 m \rceil$ inputs can also be used. There are only few ordinal attributes in the PROBEN1 problems. For these, either pseudo-real-valued or pseudo-nominal representation is used.

Nominal attributes with m different values are usually either represented using a 1-of-m code or a binary code. With the exception of gene, which uses a 2-bit binary code, PROBEN1 always employs 1-of-m representation for nominal attributes.

Missing attribute values can be replaced by a fixed value (e.g. the mean of the non-missing values of this attribute or a value found using an EM algorithm [8]) or can be represented explicitly by adding another input for the attribute that is 1 iff the attribute value is missing. PROBEN1 uses both methods; the fixed value method is used only when but a few of the values are missing. Other methods are possible if one extends the training regime away from static examples, e.g. by using a Boltzmann machine [18].

Most of the above discussion applies to outputs as well, except for the fact that there never are missing outputs. Most PROBEN1 problems are classification problems; all of these are encoded using a 1-of-m output representation for the m classes, even for $m = 2$.

The problem representation in PROBEN1 is fixed. This improves the comparability of results and reduces the work needed run benchmarks. The PROBEN1 datasets are meant to be used exactly as they are. The fixed neural network input and output representation is actually one of the major improvements of PROBEN1 over the previous situation. In the past, most benchmarks consisting of real data were publicly available only in a symbolic representation which can be encoded into a representation suitable for neural networks in many different ways. This fact made comparisons difficult.

When you perform benchmarks that do not use problems from a well-defined benchmark collection, be sure to specify exactly which input and output representation you use. Since such a description consumes a lot of space, the only feasible way will usually be to make the data file used for the actual benchmark runs available publicly.

Should you make small changes to the representation of PROBEN1 problems used in your benchmarks, specify these changes exactly. The most common cases of such changes will be concerned with the output representation. If you want to use only a single output for binary classification problems, say **card1, using only one output** or something similar. You may also want to ignore one of the outputs for problems having more than two classes, since one output is theoretically redundant since the outputs always sum to one. If you ignore an output, you should always ignore the *last* output from the given representation. If you want to use outputs in the range $-1 \ldots 1$ instead of $0 \ldots 1$ or in a somewhat reduced range in order to avoid saturation of the output nodes, say for example **with the target outputs rescaled to the range** $-0.9 \ldots 0.9$. It will be assumed that the rescaling was done using a linear transformation of the form $y' = ay + b$. Other possibilities include for instance **with the outputs rescaled to mean 0 and standard deviation 1**, which will also be assumed to be made using a linear transformation. Of course, all these rescaling modifications can be done for inputs as well, but tell us if you make such changes. I do not recommend to use PROBEN1 problems with representations that differ substantially from the standard ones unless finding good representations an important part of your work.

The input and output representations used in PROBEN1 are certainly not optimal, but they are meant to be good or at least reasonable ones. Differences in problem representation, though, can make for large differences in the performance obtained (see for instance [2]), so be sure to specify your representation precisely.

2.5 Training algorithm

Obviously, an exact specification of the training algorithm used is essential. When you use a known algorithm, specify it by giving a reference to a paper that describes it and then either use the algorithm exactly as specified in that paper or describe precisely all alterations that you make. If you introduce a new algorithm, give the algorithm a name to make it easier for other authors to refer to your algorithm. If there are several variants of your algorithm, give each variant its own name, perhaps by just appending a digit or letter to the primary name.

Whether new algorithm or not, clearly specify the values of all free parameters of the algorithm that you used. When introducing a new algorithm you should clearly indicate a prototype **parameter vector** (including parameter names) that must be specified to document each use of the algorithm. It is a common error that some of the parameter values used for an algorithm remain unspecified.

These parameters may include (depending on the algorithm) learning rate, momentum, weight decay, initialization, temperature, etc. For each such parameter there should be a clearly indicated unique name and perhaps also a symbol. For all of the parameters that are adaptive, the adaption rule and its parameters have to be specified as well. A particularly important aspect of a training algorithm is its stopping criterion, so do not forget to specify that as well (see section 3.3 for an example).

For all user-selectable parameters, specify how you found the values used and try to characterize how sensitive the algorithm is to their choice. Note that you must not in any way use the performance on the test set while searching for good parameter values; this would invalidate your results! In particular, choosing parameters based on test set results is an error.

2.6 Error measures

Many different error measures (also called error functions, objective functions, cost functions, or loss functions) can be used for network training. The most commonly used is the **squared error**: $E(o,t) = \sum_i (o_i - t_i)^2$ for actual output values o_i at the i-th output node and target output values t_i for one example. Note that some researchers multiply this by 1/2 in order to make the derivative simpler[2]; this is considered non-standard. The above measure gives one error value per example — obviously too much data to report. Thus one usually reports either the sum or the average of these values over the set of all examples. The average is called the **mean squared error**. It has the advantage of being independent of the size of the dataset and is thus preferred. Note that mean squared error still depends on the number of output coefficients in the problem representation and on the range of output values used. I thus suggest to normalize for these factors as well and report a **squared error percentage** as follows

$$E = 100 \cdot \frac{o_{max} - o_{min}}{N \cdot P} \sum_{p=1}^{P} \sum_{i=1}^{N} (o_{pi} - t_{pi})^2$$

where o_{min} and o_{max} are the minimum and maximum values of output coefficients in the problem representation (assuming these are the same for all output nodes), N is the number of output nodes of the network, and P is the number of patterns (examples) in the data set considered. Note that networks can (and in early training phases often will) produce more than 100% squared error if they use output nodes whose activation is not restricted to the range $o_{min} \ldots o_{max}$.

Other error measures include the softmax error, the cross entropy, the classification figure of merit, linear error, exponential error, minimum variance error, and others [20]. If you use any of these, state the error term explicitly. For some of them, the above idea of error percentages is applicable as well.

The actual target function for classification problems is usually not the continuous error measure used during training, but the classification performance. However, since neural networks with continuous outputs are able to approximate a-posteriori probabilities [16], which are often useful if the network outputs are to be used for further processing steps, the classification performance is not the only measure we are interested in. If space permits, you should thus report the actual error values in addition to the classification performance. Classification performance should be reported in percent of incorrectly classified examples, the classification *error*. This is better than reporting the percentage of correctly classified examples, the classification *accuracy*, because the latter makes important differences insufficiently clear: An accuracy of 98% is actually twice as good as one of 96%, which is easier to see if the errors are reported (2% compared to 4%). If classification accuracy was far below 50% instead of being far above 50%, the accuracy would better be report instead of the error, but this is an

[2]without the factor 1/2 in the error function, the correct derivative is twice as large as the one that is usually used in formulations of backpropagation. Using the common derivative thus amounts to using halved learning rates.

uncommon case. Avoid the term classification performance, use classification accuracy and classification error instead.

There are several possibilities to determine the classification a network has computed from the outputs of the network. We assume a 1-of-m encoding for m classes using output values 0 and 1. The simplest classification method is the winner-takes-all, i.e., the output with the highest activation designates the class. Other methods involve the possibility of rejection, too. For instance one could require that there is exactly one output that is larger than 0.5, which designates the class if it exists and leads to rejection otherwise. To put an even stronger requirement on the credibility of the network output one can set thresholds, e.g. accept an output as 0 if it is below 0.3 and as 1 if it is above 0.7, and reject unless there is exactly one output that is 1 while all others are 0 by this interpretation. There are several other possibilities. When no rejection capability is needed, the winner-takes-all method is considered standard. In all other cases, describe your classification decision function explicitly.

2.7 Network used

Specify exactly the topology of the neural network used in any benchmark test. The topology of a network is described by the graph of the nodes (units, vertices, neurons) and connections (weights, edges, synapses). Avoid the terms 'neuron' and 'synapse', because they are inappropriate for artificial neural networks. The term 'weight' should be used to refer to the parameter attached to a connection, but not to the connection itself.

To describe the topology, try to refer to common topology models. For instance for the common case of the so-called fully connected layered feed forward networks, the numbers of nodes in each layer from input to output can be given as a sequence: a 5-4-6 network refers to a network with 5 input, 4 hidden, and 6 output nodes. There is confusion how to count the number of layers in a network, so do not call a network like the one above a "three layer network" (counting all groups of nodes) nor a "two layer network" (counting only the groups of nodes with input connections). Instead, call it a network with one hidden layer. This generalizes to arbitrary numbers of layers. For instance, a 5-10-3-5-6 is a three hidden layer network.

Specifying the number of nodes is not sufficient even for the "fully connected" networks, because by this term, some people mean that all connections between adjacent layers are present, while others mean that all connections are present, even those that skip intermediate layers (shortcut connections). Thus, use formulations like with all feed forward connections between adjacent layers or with all feed forward connections, including all shortcut connections as a complement to the specification of the size of the layers. Examples: a 5-4-6 network with all feed forward connections, including all shortcut connections or networks with one hidden layer (having between 2 and 20 hidden nodes) and all feed forward connections between adjacent layers.

Most networks also have a bias (or threshold) for all hidden and output nodes. This bias can be implemented either as an incoming connection from a node with constant non-zero output (the bias node) or as an adaptable parameter of the node activation function. Since the style of implementation is usually irrelevant and networks without bias are the exception, bias need not be mentioned. If some nodes do *not* use bias, specify which (and why). Note that if you compute the number of free parameters in a network, the bias parameter of each hidden and output node has to be included. Since this may confuse the reader, you should mention the bias in this case.

For recurrent networks use standard names such as Jordan or Elman network where appropriate and back it up by a reference or further explanation. Non-standard network topologies or non-standard network models such as networks with shared weights [14] have to be described in detail.

Other properties of the network architecture also have to be specified: the range and resolution of the weight parameters (unless plain 32-bit floating point is used), the activation function of each node in the network (except for the input nodes which are assumed to use the identity function; see also section 2.10), and any other free parameters associated with the network.

2.8 Training results

Usually what one is interested in when training a neural network is its generalization performance. The value that is usually used to characterize generalization performance is the error on a test set. A test set is a set of examples that was not used in any way whatsoever during the training process (see section 2.3 above). This test set error is thus the primary result to be presented for any learning problem used. The corresponding errors on the training and validation set, if any, are of only marginal interest and need thus not be reported.

Since training a neural network usually involves some kind of random initialization, the results of several training runs of the same algorithm on the same dataset will differ. In order to make reliable statements about the performance of an algorithm it is thus necessary to make several runs and report statistics on the distribution of results obtained. If possible, use either 10 runs or 30 runs or some power of ten of these numbers, because if many researchers use the same numbers of runs direct comparisons are easier. If these numbers don't seem appropriate for some reason, try to use either 20 or 60 runs or some power of ten of these numbers. The commonly used statistics to report about the results of the runs should be primarily the mean and $n-1$ standard deviation[3] of test set error (and/or test set classification error) and the 'best' run (see below), secondarily the minimum, maximum, and median, and if still more data shall be presented, all five quartiles or even a fine-grained distribution histogram.

The meaning of the 'best' run result is to characterize what one could get using a method of model selection that trained several networks and then picked that one of them that "looked best". In contrast, all other statistics characterize the quality one can expect if one trains just one network. The selection of the 'best' run must thus not be based on the results of the test set, because that would mean to use the test set error during the model selection process whereas the test set error is conceptually the *result* of the model selection process. Instead, training set error or validation set error or some other quality measure computed exclusively from the network and the training data must be used. This means that the 'best' network will often not have the minimum test error! So for instance if your selection criterion is validation set error, you should report something like the network with lowest validation set error in 30 runs had a test set classification error of 2.34%.

If for some reason you want to exclude some of the runs from the results presented, for instance because these runs are considered to have *not converged* (whatever that may mean), always exclude exactly half of all runs. This allows for easier comparison with the results of other researchers. The runs to be excluded are the worst runs in the inverse sense of 'best' from above, i.e., you must *not* exclude those runs that have the worst test set error.

You may want to apply methods of statistical inference to your training results, for instance in order to test whether one algorithm is *significantly* better than another. In this case, it may be necessary to remove a small number of *outliers* from the samples to be compared in order to make the data satisfy some requirement of the statistical procedure. For instance in order to apply a t-test, the samples to be compared must have a normal distribution. If a sample (of, say, the test errors from 30 runs) is approximately normal except for, say, two outliers with very much larger (or smaller) errors than

[3] That is, standard deviation computed based on the degrees of freedom, which is one less than the number n of runs.

all the rest, you can remove these two outliers from the sample. Never remove more than 10% of the values from any one sample; usually one should remove much less. Never remove an outlier unless the resulting distribution satisfies the requirement well enough. Other data transformations than removing outliers may be more appropriate to satisfy the requirements of the statistical procedure; for instance test errors are often log-normally distributed, so one must use the logarithm of the test error instead of the test error itself in order to produce valid results. See also section 3.3.4.

2.9 Training times

Unless your algorithm does perform a lot of additional work besides propagating data through a neural network, the most sensible measure of training time is the number of connection traversals (connection crossings, sometimes misleadingly called connection updates) needed. This measure is useful because it is independent of a particular machine and implementation. Forward and backward propagation counts individually, for certain algorithms that require more than one quantity to be backwardly propagated through each connection such as [1, 13], each quantity counts as one traversal at each connection. Actual weight update steps also count as one traversal per updated connection. If possible report your training times using the connection traversal measure.

If your algorithm performs much work besides traversing the network, the actual CPU time spent is the best measure to give. The disadvantage of this measure is that it leaves two free parameters: the speed of the machine used and the efficiency of the software implementation. Thus, the measure is directly comparable only for the same software on the same machine. When reporting CPU times, give the precise brand and model number of the machine you used and its nominal performance in SPEC marks; give a hint as to whether the software used should be regarded efficient or not so efficient. However, CPU time is certainly always useful as a ballpark figure for the computational size of the tackled problem.

A less useful measure is the number of *epochs* used, i.e., the number of times each example was processed. This value can be misleading, because the computational cost of one epoch can differ significantly from one algorithm or network to another. It is nevertheless fine to present the epoch counts in addition to other measures.

Regarding non-converging runs [3], the values you report should reflect the actual amount of computation time that was spent. This means that your algorithm should define some stopping or restarting criterion and the sum of all computation actually performed before and after the restart(s) should be reported as the training time. It is important to report the precise stopping or restarting criterion that was used.

2.10 Important details

Finally, some important details are often forgotten; all of them were already shortly mentioned above.

Activation function. Exactly specify the activation function used in the nodes (units) of your network. You can say **standard sigmoid** to mean $1/(1+e^{-x})$ and you can say **tanh** to mean the tangens hyperbolicus, which is $2/(1+e^{-2x})-1$; these two are the standard choices. All other activation functions should be given explicitly. Specify whether the output nodes of the network also use this activation function or use the identity function instead. If the nodes of the input layer (fan-out nodes) perform any computation on the input values, specify this computation.

Network initialization. Specify the initialization conditions of the network. The most important point is the initialization of the network's weights, which must be done with random values for most algorithms in order to break the symmetry among the hidden nodes. Common choices for the initialization are for instance fixed methods such as random weights from range $-0.1 \ldots 0.1$, where the distribution is assumed to be even unless stated otherwise, or methods that adapt to the network topology used such as random weights from range $-1/\sqrt{N} \ldots 1/\sqrt{N}$ for connections into nodes with N input connections. Just like the termination criterion, the initialization can have significant impact on the results obtained, so it is important to specify it precisely. Specifying the exact sets of weights used is hopelessly difficult and should usually not be tried.

Termination and phase transition criteria. Specify exactly the criteria used to determine when training should stop, or when training should switch from one phase to the next, if any. For most algorithms, the results are very sensitive to these criteria. Nevertheless, in most publications the criteria are specified only roughly, if at all. This is one of the major weaknesses of many articles on neural network learning algorithms. See section 3.3 for an example of how to report stopping criteria; the GL_α family of stopping criteria, which is defined in that section, is recommended when using the early stopping method.

2.11 Author's quick reference

The following is a quick reference check list of all the points that should be mentioned in a publication reporting a benchmark test. Remember that peculiar points not listed here may apply additionally to the particular benchmarks you want to report.

1. Problem: name, address, version/variant.
2. Training set, validation set, test set.
3. Network: nodes, connections, activation functions.
4. Initialization.
5. Algorithm parameters and parameter adaption rules.
6. Termination, phase transition, and restarting criteria.
7. Error function and its normalization on the results reported.
8. Number of runs, rules for including or excluding runs in results reported.

3 Benchmarking problems

The following subsections each describe one of the problems of the PROBEN1 benchmark set. For each problem, a rough description of the semantics of the dataset is given, plus some information about the size of the dataset, its origin, and special properties, if any. For most of the problems, results have previously been published in the literature. Since these results never use exactly the same representation and training set/test set splitting as the PROBEN1 versions, the references are not given here; some of them can, however, be found in the documentation supplied with the original dataset, which is part of PROBEN1. The final section reports on the results of some learning runs with the PROBEN1 datasets.

3.1 Classification problems

3.1.1 Cancer

Diagnosis of breast cancer. Try to classify a tumor as either benign or malignant based on cell descriptions gathered by microscopic examination. Input attributes are for instance the clump thickness, the uniformity of cell size and cell shape, the amount of marginal adhesion, and the frequency of bare nuclei.

9 inputs, 2 outputs, 699 examples. All inputs are continuous; 65.5% of the examples are benign. This makes for an entropy of 0.93 bits per example[4].

This dataset was created based on the "breast cancer Wisconsin" problem dataset from the UCI repository of machine learning databases. Please mention in any publication presenting results for this data set that the data was originally obtained from the University of Wisconsin Hospitals, Madison, from Dr. William H. Wolberg. Also please cite one or more of the four publications mentioned in the detailed documentation of the original dataset in the `proben1/cancer` directory.

3.1.2 Card

Predict the approval or non-approval of a credit card to a customer. Each example represents a real credit card application and the output describes whether the bank (or similar institution) granted the credit card or not. The meaning of the individual attributes is unexplained for confidence reasons.

51 inputs, 2 outputs 690 examples. This dataset has a good mix of attributes: continuous, nominal with small numbers of values, and nominal with larger numbers of values. There are also a few missing values in 5% of the examples. 44% of the examples are positive; entropy 0.99 bits per example.

This dataset was created based on the "crx" data of the "Credit screening" problem dataset from the UCI repository of machine learning databases.

3.1.3 Diabetes

Diagnose diabetes of Pima indians. Based on personal data (age, number of times pregnant) and the results of medical examinations (e.g. blood pressure, body mass index, result of glucose tolerance test, etc.), try to decide whether a Pima indian individual is diabetes positive or not.

8 inputs, 2 outputs, 768 examples. All inputs are continuous. 65.1% of the examples are diabetes negative; entropy 0.93 bits per example. Although there are no missing values in this dataset according to its documentation, there are several senseless 0 values. These most probably indicate missing data. Nevertheless, we handle this data as if it was real, thereby introducing some errors (or noise, if you want) into the dataset.

This dataset was created based on the "Pima indians diabetes" problem dataset from the UCI repository of machine learning databases.

[4] Entropy $E = \sum_{\text{Classes } c} P(c) \log_2(P(c))$ for class probabilities $P(c)$

3.1.4 Gene

Detect intron/exon boundaries (splice junctions) in nucleotide sequences. From a window of 60 DNA sequence elements (nucleotides) decide whether the middle is either an intron/exon boundary (a donor), or an exon/intron boundary (an acceptor), or none of these.

120 inputs, 3 outputs, 3175 examples. Each nucleotide, which is a four-valued nominal attribute, is enoded binary by two binary inputs (The input values used are -1 and 1, therefore the inputs are *not* declared as boolean. This is the only dataset that has input values not restricted to the range $0 \ldots 1$). There are 25% donors and 25% acceptors in the dataset; entropy 1.5 bits per example.

This dataset was created based on the "splice junction" problem dataset from the UCI repository of machine learning databases.

3.1.5 Glass

Classify glass types. The results of a chemical analysis of glass splinters (percent content of 8 different elements) plus the refractive index are used to classify the sample to be either float processed or non float processed building windows, vehicle windows, containers, tableware, or head lamps. This task is motivated by forensic needs in criminal investigation.

9 inputs, 6 outputs, 214 examples. All inputs are continuous, two of them have hardly any correlation with the result. As the number of examples is quite small, the problem is sensitive to algorithms that waste information. The sizes of the 6 classes are 70, 76, 17, 13, 9, and 29 instances, respectively; entropy 2.18 bits per example.

This dataset was created based on the "glass" problem dataset from the UCI repository of machine learning databases.

3.1.6 Heart

Predict heart disease. Decide whether at least one of four major vessels is reduced in diameter by more than 50%. The binary decision is made based on personal data such as age, sex, smoking habits, subjective patient pain descriptions, and results of various medical examinations such as blood pressure and electro cardiogram results.

35 inputs, 2 outputs, 920 examples. Most of the attributes have missing values, some quite many: For attributes 10, 12, and 11, there are 309, 486, and 611 values missing, respectively. Most other attributes have around 60 missing values. Additional boolean inputs are used to represent the "missingness" of these values. The data is the union of four datasets: from Cleveland Clinic Foundation, Hungarian Institute of Cardiology, V.A. Medical Center Long Beach, and University Hospital Zurich. There is an alternate version of the dataset `heart`, called `heartc`, which contains only the Cleveland data (303 examples). This dataset represents the cleanest part of the heart data; it has only two missing attribute values overall, which makes the "value is missing" inputs of the neural network input representation almost redundant. Furthermore, there are still another two versions of the same data, `hearta` and `heartac`, corresponding to `heart` and `heartc`, respectively. The difference to the datasets described above is the representation of the output. Instead of using two binary outputs to represent the two-class decision "no vessel is reduced" against "at least one vessel is reduced", `hearta` and `heartac` use a single continuous output that represents by the magnitude of its activation the number of vessels that are reduced (zero to four). Thus, these versions of the heart problem are approximation tasks.

The `heart` and `hearta` datasets have 45% patients with "no vessel is reduced" (entropy 0.99 bits per example), for `heartc` and `heartac` the value is 54% (entropy 1.00 bit per example).

These datasets were created based on the "heart disease" problem datasets from the UCI repository of machine learning databases. Note that using these datasets requires to include in any publication of the results the name of the institutions and persons who have collected the data in the first place, namely (1) Hungarian Institute of Cardiology, Budapest; Andras Janosi, M.D., (2) University Hospital, Zurich, Switzerland; William Steinbrunn, M.D., (3) University Hospital, Basel, Switzerland; Matthias Pfisterer, M.D., (4) V.A. Medical Center, Long Beach and Cleveland Clinic Foundation; Robert Detrano, M.D., Ph.D. All four of these should be mentioned for the `heart` and `hearta` datasets, only the last one for the `heartc` and `heartac` datasets. See the detailed documentation of the original datasets in the `proben1/heart` directory.

3.1.7 Horse

Predict the fate of a horse that has a colic. The results of a veterinary examination of a horse having colic are used to predict whether the horse will survive, will die, or will be euthanized.

58 inputs, 3 outputs, 364 examples. In 62% of the examples the horse survived, in 24% it died, and in 14% it was euthanized; entropy 1.32 bits per example. This problem has *very* many missing values (about 30% overall of the original attribute values), which are all represented as missing explicitly using additional inputs.

This dataset was created based on the "horse colic" problem dataset from the UCI repository of machine learning databases.

3.1.8 Mushroom

Discriminate edible from poisonous mushrooms. The decision is made based on a description of the mushroom's shape, color, odor, and habitat.

125 inputs, 2 outputs, 8124 examples. Only one attribute has missing values (30% missing). This dataset is special within the benchmark set in several respects: it is the one with the most inputs, the one with the most examples, the easiest one[5], and it is the only one that is not *real* in the sense that its examples are not actual observations made in the real world, but instead are hypothetical observations based on descriptions of species in a book ("The Audubon Society Field Guide to North American Mushrooms"). The examples correspond to 23 species of gilled mushrooms in the Agaricus and Lepiota Family. In the book, each species is identified as definitely edible, definitely poisonous, or of unknown edibility and not recommended. This latter class was combined with the poisonous one. 52% of the examples are edible (ahem, I mean, have class attribute 'edible'); entropy 1.00 bit per example.

This dataset was created based on the "agaricus lepiota" dataset in the "mushroom" directory from the UCI repository of machine learning databases.

[5] The mushroom dataset is so simple that a net that performs only a linear combination of the inputs can learn it reliably to 0 classification error on the test set!

3.1.9 Soybean

Recognize 19 different diseases of soybeans. The discrimination is done based on a description of the bean (e.g. whether its size and color are normal) and the plant (e.g. the size of spots on the leafs, whether these spots have a halo, whether plant growth is normal, whether roots are rotted) plus information about the history of the plant's life (e.g. whether changes in crop occurred in the last year or last two years, whether seeds were treated, how the environment temperature is).

35 inputs, 19 outputs, 683 examples. This is the problem with the highest number of classes in the benchmark set. Most attributes have a significant number of missing values. The soybean problem has been used often in the machine learning literature, although with several different datasets, making comparisons difficult. Most of the past uses use only 15 of the 19 classes, because the other four have only few instances. In this dataset, these are 8, 14, 15, 16 instances versus 20 for most of the other classes; entropy 3.84 bits per example.

This dataset was created based on the "soybean large" problem dataset from the UCI repository of machine learning databases. Many results for this learning problem have been reported in the literature, but these were based on a large number of different versions of the data.

3.1.10 Thyroid

Diagnose thyroid hyper- or hypofunction. Based on patient query data and patient examination data, the task is to decide whether the patient's thyroid has overfunction, normal function, or underfunction.

21 inputs, 3 outputs, 7200 examples. For various attributes there are missing values which are always encoded using a separate input. Since some results for this dataset using the same encoding are reported in the literature, `thyroid1` is *not* a permutation of the original data, but retains the original order instead. The class probabilities are 5.1%, 92.6%, and 2.3%, respectively; entropy 0.45 bits per example.

This dataset was created based on the "ann" version of the "thyroid disease" problem dataset from the UCI repository of machine learning databases.

3.1.11 Summary

For a quick overview of the classification problems, have a look at table 1. The table summarizes the external aspects of the training problems that you have already seen in the individual descriptions above. It does also discriminate inputs that take on only two different values (binary inputs), inputs that have more than two ("continuous" inputs), and inputs that are present only to indicate that values at some other inputs are missing. In addition, the table indicates the number of attributes of the original problem formulation that were used in the input representation, discriminated to be either binary attributes, "continuous" attributes, or nominal attributes with more than two values.

3.2 Approximation problems

3.2.1 Building

Prediction of energy consumption in a building. Try to predict the hourly consumption of electrical energy, hot water, and cold water, based on the date, time of day, outside temperature, outside air humidity, solar radiation, and wind speed.

Problem	Problem attributes				Input values				Classes	Examples	E
	b	c	n	tot.	b	c	m	tot.	b		
cancer	0	9	0	9	0	9	0	9	2	699	0.93
card	4	6	5	15	40	6	5	51	2	690	0.99
diabetes	0	8	0	8	0	8	0	8	2	768	0.93
gene	0	0	60	60	120	0	0	120	3	3175	1.50
glass	0	9	0	9	0	9	0	9	6	214	2.18
heart	1	6	6	13	18	6	11	35	2	920	0.99
heartc	1	6	6	13	18	6	11	35	2	303	1.00
horse	2	13	5	20	25	14	19	58	3	364	1.32
mushroom	0	0	22	22	125	0	0	125	2	8124	1.00
soybean	16	6	13	35	46	9	27	82	19	683	3.84
thyroid	9	6	0	21	9	6	6	21	3	7200	0.45

Problems and the number of binary, continuous, and nominal attributes in the original dataset, number of binary and continuous network inputs, number of network inputs used to represent missing values, number of classes, number of examples, class entropy E in bits per example. (Continuous means more than two different ordered values).

Table 1: Attribute structure of classification problems

14 inputs, 3 outputs, 4208 examples. This problem is in its original formulation an extrapolation task. Complete hourly data for four consecutive months was given for training, and output data for the following two months should be predicted. The dataset `building1` reflects this formulation of the task: its examples are in chronological order. The other two versions, `building2` and `building3` are random permutations of the examples, simplifying the problem to be an interpolation problem.

The dataset was created based on problem A of "The Great Energy Predictor Shootout — the first building data analysis and prediction problem" contest, organized in 1993 for the ASHRAE meeting in Denver, Colorado.

3.2.2 Flare

Prediction of solar flares. Try to guess the number of solar flares of small, medium, and large size that will happen during the next 24-hour period in a fixed active region of the sun surface. Input values describe previous flare activity and the type and history of the active region.

24 inputs, 3 outputs, 1066 examples. 81% of the examples are zero in all three output values.

This dataset was created based on the "solar flare" problem dataset from the UCI repository of machine learning databases.

3.2.3 Hearta

The analog version of the heart disease diagnosis problem. See section 3.1.6 on page 19 for the description. For `hearta`, 44.7%, 28.8%, 11.8%, 11.6%, 3.0% of all examples have 0, 1, 2, 3, 4 vessels reduced, respectively. For `heartac` these values are 54.1%, 18.2%, 11.9%, 11.6%, and 4.3%.

3.2.4 Summary

For a quick overview of the approximation problems, have a look at table 2. The table summarizes

Problem	Problem attribs.				Input values				Outputs	Examples
	b	c	n	tot.	b	c	m	tot.	c	
building	0	6	0	6	8	6	0	14	3	4208
flare	5	2	3	10	22	2	0	24	3	1066
hearta	1	6	6	13	18	6	11	35	1	920
heartac	1	6	6	13	18	6	11	35	1	303

Problems and the number of binary, continuous, and nominal attributes of the original problem representation used, number of binary and continuous network inputs, number of network inputs used to represent missing values, number of outputs, number of examples. (Continuous means more than two different ordered values).

Table 2: Attribute structure of approximation problems

the external aspects of the training problems that you have already seen in the individual descriptions above. It does also discriminate inputs that take on only two different values (binary inputs), inputs that have more than two ("continuous" inputs), and inputs that are present only to indicate that values at some other inputs are missing. In addition, the table indicates the number of attributes of the original problem formulation that were used in the input representation, discriminated to be either binary attributes, "continuous" attributes, or nominal attributes with more than two values. The outputs have continuous values.

3.3 Some learning results

In this section we will see a few results of neural network learning runs on the datasets described above. The runs were made with linear networks, having only direct connections from the inputs to the outputs, and with various fully connected multi layer perceptrons with one or two layers of sigmoidal hidden nodes.

The method applied for training was the same in all cases and can be summarized as follows: Training was performed using the RPROP algorithm [17] with parameters as indicated below. RPROP is a fast backpropagation variant similar in spirit to Quickprop. It is about as fast as Quickprop but requires less adjustment of the parameters to be stable. The parameters used were not determined by a trial-and-error search, but are just educated guesses instead. RPROP requires epoch learning, i.e., the weights are updated only once per epoch. While epoch updates are is not desirable for very large training sets, it is a good method for small and medium training sets such as those of Proben1, because it allows the use of acceleration techniques as those used in RPROP. Conjugate gradient optimization methods would be another class of useful algorithms for this kind of training problems [11].

The squared error function was used. For each dataset, training used the training set and the error on the validation set was measured after every fifth epoch (this interval between two measurements of the validation set error is called the *strip length*, see below). Training was stopped as soon as the GL_5 stopping criterion was fulfilled (see below) or when training progress sank below 0.1 per thousand (see below) or when a maximum of 3000 epochs had been trained. The test set performance was then computed for that state of the network which had minimum validation set error during the training process.

This method, called *early stopping* [6, 9, 12], is a good way to avoid overfitting [7] of the network to the particular training examples used, which would reduce the generalization performance. For optimal performance, the examples of the validation set should be used for further training afterwards, in order not to waste valuable data. Since the optimal stopping point for this additional training is not clear, it was not performed in the experiments reported here.

The GL_5 stopping criterion is defined as follows. Let E be the squared error function. Let $E_{tr}(t)$ be the average error per example over the training set, measured during epoch t. $E_{va}(t)$ is the error on the validation set after epoch t and is used by the stopping criterion. $E_{te}(t)$ is the error on the test set; it is not known to the training algorithm but characterizes the quality of the network resulting from training.

The value $E_{opt}(t)$ is defined to be the lowest validation set error obtained in epochs up to t:

$$E_{opt}(t) = \min_{t' \leq t} E_{va}(t')$$

Now we define the *generalization loss* at epoch t to be the relative increase of the validation error over the minimum-so-far (in percent):

$$GL(t) = 100 \cdot \left(\frac{E_{va}(t)}{E_{opt}(t)} - 1 \right)$$

A high generalization loss is one candidate reason to stop training. This leads us to a class of stopping criteria: Stop as soon as the generalization loss exceeds a certain threshold α. We define the class GL_α as

$$GL_\alpha : \text{ stop after first epoch } t \text{ with } GL(t) > \alpha$$

To formalize the notion of training progress, we define a *training strip of length k* to be a sequence of k epochs numbered $n+1 \ldots n+k$ where n is divisible by k. The training *progress* (measured in parts per thousand) measured after such a training strip is then

$$P_k(t) = 1000 \cdot \left(\frac{\sum_{t' \in t-k+1 \ldots t} E_{tr}(t')}{k \cdot \min_{t' \in t-k+1 \ldots t} E_{tr}(t')} - 1 \right)$$

that is, "how much was the average training error during the strip larger than the minimum training error during the strip?" Note that this progress measure is high for instable phases of training, where the training set error goes up instead of down. The progress is, however, guaranteed to approach zero in the long run unless the training is globally unstable (e.g. oscillating). Just like the progress, GL is also evaluated only at the end of each training strip.

3.3.1 Linear networks

A first set of results is shown in the tables 3 (classification problems) and 4 (approximation problems). These tables contain the results of 10 runs training a linear neural network for each of the datasets. The network had no hidden nodes, just direct connections from each input to each output. The output units used the identity activation function, i.e., their output is just the summed input. The RPROP algorithm used the following parameters: $\eta^+ = 1.2$, $\eta^- = 0.5$, $\Delta_0 \in 0.005 \ldots 0.02$ randomly per weight, $\Delta_{max} = 50$, $\Delta_{min} = 0$, initial weights from $-0.01 \ldots 0.01$ randomly. Training was terminated according to the GL_5 stopping criterion using a strip length of 5 epochs.

The results of these training runs give a first impression of how difficult the problems are. There are some interesting observations to be made:

Problem	Training set		Validation set		Test set		Test set classification		Overfit		Total epochs		Relevant epochs	
	mean	stddev	mean	stddev	mean	stddev	mean	stddev	mean	stddev	mean	stddev	mean	stddev
cancer1	4.25	0.00	2.91	0.01	3.52	0.04	2.93	0.18	0.55	0.59	129	13	104	31
cancer2	3.95	0.52	3.77	0.47	4.77	0.39	5.00	0.61	5.36	10.21	87	51	79	51
cancer3	3.30	0.00	4.23	0.04	4.11	0.03	5.17	0.00	0.35	0.64	115	18	92	29
card1	9.82	0.01	8.89	0.11	10.61	0.11	13.37	0.67	4.57	1.05	62	9	26	3
card2	8.24	0.01	10.80	0.16	14.91	0.55	19.24	0.43	4.22	1.08	65	10	23	5
card3	9.47	0.00	8.39	0.07	12.67	0.17	14.42	0.46	1.52	0.69	102	9	44	12
diabetes1	15.39	0.01	16.30	0.04	17.22	0.06	25.83	0.56	0.05	0.07	209	50	203	47
diabetes2	14.93	0.01	17.47	0.02	17.69	0.04	24.69	0.61	0.02	0.02	209	32	204	34
diabetes3	14.78	0.02	18.21	0.04	16.50	0.05	22.92	0.35	0.12	0.17	214	22	185	46
gene1	8.42	0.00	9.58	0.01	9.92	0.01	13.64	0.10	0.03	0.07	47	6	43	10
gene2	8.39	0.00	9.90	0.00	9.51	0.00	12.30	0.14	0.02	0.03	46	4	40	6
gene3	8.21	0.00	9.36	0.01	10.61	0.01	15.41	0.13	0.03	0.06	42	4	39	6
glass1	8.83	0.01	9.70	0.04	9.98	0.10	46.04	2.21	3.81	0.42	129	13	23	5
glass2	8.71	0.09	10.28	0.19	10.34	0.15	55.28	1.27	5.74	0.67	34	6	14	2
glass3	8.71	0.02	9.37	0.06	11.07	0.15	60.57	3.82	1.76	0.57	135	30	27	11
heart1	11.19	0.01	13.28	0.06	14.29	0.05	20.65	0.31	1.14	0.45	134	15	41	5
heart2	11.66	0.01	12.22	0.02	13.52	0.06	16.43	0.40	0.13	0.09	184	14	146	48
heart3	11.11	0.01	10.77	0.02	16.39	0.18	22.65	0.69	0.14	0.23	142	15	113	53
heartc1	10.17	0.01	9.65	0.03	16.12	0.04	19.73	0.56	0.15	0.11	128	10	114	23
heartc2	11.23	0.03	16.51	0.08	6.34	0.25	3.20	1.56	3.98	0.56	136	22	25	10
heartc3	10.48	0.31	13.88	0.33	12.53	0.44	14.27	1.67	6.23	1.15	26	9	12	3
horse1	11.31	0.16	15.53	0.29	12.93	0.38	26.70	1.87	6.22	0.57	27	7	9	2
horse2	8.62	0.28	15.99	0.21	17.43	0.45	34.84	1.38	5.54	0.47	42	16	13	3
horse3	10.43	0.27	15.59	0.30	15.50	0.45	32.42	2.65	6.34	1.07	26	6	8	3
mushroom1	0.014	—	0.014	—	0.011	—	0.00	—	0.00	—	3000	—	3000	—
soybean1	0.65	0.00	0.98	0.00	1.16	0.00	9.47	0.51	0.28	0.18	553	11	418	41
soybean2	0.80	0.00	0.81	0.00	1.05	0.00	4.24	0.25	0.02	0.02	509	19	504	18
soybean3	0.78	0.00	0.96	0.00	1.03	0.00	7.00	0.19	0.03	0.04	533	27	522	28
thyroid1	3.76	0.00	3.78	0.01	3.84	0.01	6.56	0.00	0.01	0.03	104	16	99	22
thyroid2	3.93	0.00	3.55	0.01	3.71	0.01	6.56	0.00	0.01	0.02	98	16	96	16
thyroid3	3.85	0.00	3.39	0.00	4.02	0.00	7.23	0.02	0.02	0.02	114	22	109	21

Training set: mean and standard deviation (stddev) of minimum squared error percentage on training set reached at any time during training.
Validation set: ditto, on validation set.
Test set: mean and stddev of squared test set error percentage at point of minimum validation set error.
Test set classification: mean and stddev of corresponding test set classification error.
Overfit: mean and stddev of GL value at end of training.
Total epochs: mean and stddev of number of epochs trained.
Relevant epochs: mean and stddev of number of epochs until minimum validation error.

Table 3: Linear network results of classification problems

Problem	Training set		Validation set		Test set		Overfit		Total epochs		Relevant epochs	
	mean	stddev	mean	stddev	mean	stddev	mean	stddev	mean	stddev	mean	stddev
building1	0.21	0.01	0.92	0.06	0.78	0.02	2.15	4.64	407	138	401	142
building2	0.34	0.00	0.37	0.00	0.35	0.00	0.00	0.01	298	23	297	23
building3	0.37	0.04	0.38	0.07	0.38	0.08	1.99	4.45	229	107	217	102
flare1	0.37	0.00	0.34	0.01	0.52	0.01	2.17	1.61	41	5	12	4
flare2	0.42	0.00	0.46	0.00	0.31	0.02	0.72	0.90	37	3	16	10
flare3	0.39	0.00	0.46	0.00	0.35	0.00	0.57	0.73	35	6	18	12
hearta1	3.82	0.00	4.42	0.03	4.47	0.06	1.68	0.68	118	12	27	10
hearta2	4.17	0.00	4.28	0.02	4.19	0.01	0.06	0.13	112	10	107	15
hearta3	4.06	0.00	4.14	0.02	4.54	0.01	0.05	0.05	116	8	110	10
heartac1	4.05	0.00	4.70	0.02	2.69	0.02	0.01	0.02	98	10	96	11
heartac2	3.37	0.11	5.21	0.21	3.87	0.16	6.99	2.27	19	4	13	4
heartac3	2.85	0.09	5.66	0.16	5.43	0.23	6.06	0.99	29	9	14	3

(The explanation from table 3 applies, except that the test set classification error data is not present here.)

Table 4: Linear network results of approximation problems

1. Some of the problems seem to be very sensitive to overfitting. They overfit heavily even with only a linear network (e.g. card1, card2, glass1, glass2, heartac2, heartac3, heartc2, heartc3, horse1, horse2, horse3). This suggests that using a cross validation technique such as early stopping is very useful for the PROBEN1 problems.
2. For some problems, there are quite large differences of behavior between the three permutations of the dataset (e.g. test errors of card, heartc, heartac; training times of heartc; overfitting of glass). This illustrates how dangerous it is to compare results for which the splitting of the data into training and test data was not the same.
3. Some of the problems can be solved pretty well with a linear network. So one should be aware that sometimes a 'real' neural network might be an overkill.
4. The mushroom problem is boring. Therefore, only a single run was made. It reached zero test set classification error after only 80 epochs and zero validation set error after 1550 epochs. However, training stopped only because of the 3000 epoch limit; the errors themselves fell and fell and fell. Due to these results, the mushroom problem was excluded from the other experiments. Using the mushroom problem may be interesting, however, if one wants to explore the scaling behavior of an algorithm with respect to the number of available training examples.
5. Some problems exhibit an interesting "inverse" behavior of errors. Their validation error is lower than the minimum training error (e.g. cancer1, cancer2, card1, card3, heart3, heartc1, thyroid2, thyroid3). In a few cases, this even extends to the test error (cancer1, thyroid2).

3.3.2 Choosing multilayer architectures

As a baseline for further comparison, a number of runs was made using multilayer networks with sigmoidal hidden nodes. For each problem, 12 different network topologies were used: one-hidden-layer networks with 2, 4, 8, 16, 24, or 32 hidden nodes and two-hidden-layer networks with 2+2, 4+2, 4+4, 8+4, 8+8, and 16+8 hidden nodes on the first and second hidden layer, respectively. All of these networks had all possible feed forward connections, including all shortcut connections. The sigmoid activation function used was $y = x/(1 + |x|)$.

For each of these topologies, three runs were performed; two with linear output nodes and one with output nodes using the sigmoidal activation function. Note that in this case the sigmoid output nodes perform only a one-sided squashing of the outputs, because the sigmoid range is $-1 \ldots 1$ whereas the target output range is only $0 \ldots 1$. The parameters for the RPROP procedure used in all these runs were $\eta^+ = 1.1$, $\eta^- = 0.5$, $\Delta_0 \in 0.05 \ldots 0.2$ randomly per weight, $\Delta_{max} = 50$, $\Delta_{min} = 0$, initial weights -0.5...0.5 randomly. Exchanging this with the parameter set used for the linear networks would, however, not make much of a difference. Training was stopped when either $P_5(t) < 0.1$ or more than 3000 epochs trained or the following condition was satisfied: The GL_5 stopping criterion was fulfilled at least once *and* the validation error had increased for at least 8 successive strips at least once *and* the quotient $GL(t)/P_5(t)$ had been larger than 3 at least once[6].

The tables in tables 5 and 6 present the topology and results of the network that produced the lowest validation set error of all these runs for each dataset. The tables also contain some indication of the performance of other topologies by giving the number of other runs that were at most 5% (or 10%) worse than the best run, with respect to the validation set error. The range of test set errors obtained for these other topologies is also indicated.

The architectures presented in these tables are probably not the optimal ones, even among those considered in the set of runs presented. Due to the small number of runs per architecture for each problem, a suboptimal architecture has a decent probability of producing the lowest validation set error just by chance. Experience with the early stopping method suggests that using a network considerably larger than necessary often leads to the best results. As a consequence, the architectures presented in the table shown in table 7 were computed from the results of the runs as the suggested architectures for the various datasets to be used for training of fully connected multi layer perceptrons. These architectures are called the *pivot architectures* of the respective problems. The rule for computing which architecture is the pivot architecture uses all runs from the within-5%-of-best category as candidates. From these, the largest architecture is chosen. Should the same largest topology appear among the candidates with both linear and sigmoidal output units, the one with smaller validation set error is chosen, unless the linear architecture appears *twice*, in which case it is preferred regardless of its validation set error. The raw data used for this architecture selection is listed in appendix D.

It should be noted that these pivot architectures are still not necessarily very good. In particular for some of the problems it might be appropriate to train networks without shortcut connections in order to use networks with a much smaller number of parameters. For instance in the glass problems, the shortcut connections amount for as many as 60 weights, which is about the same number as are needed for a complete network using 4 hidden nodes but no shortcut connections. Since the problem has only 107 examples in the training set, it may be a good idea to start without shortcut connections. Similar argumentation applies for several other problems as well. Furthermore, since many of the pivot architectures are one of the two largest architectures available in the selection runs, namely 32+0 or 16+8, networks with still more hidden nodes may produce superior results for some of the problems.

The following section presents results for multiple runs using the pivot architectures, a subsequent section presents results for multiple runs with the same architectures except for the shortcut connections.

3.3.3 Multilayer networks

Tables 8 (classification problems) and 9 (approximation problems) show the results of training with the pivot architectures. For each variant of each problem, 60 runs were performed. The training

[6] The only reason for this complicated criterion is that the same set of runs was also used to investigate the behavior

Problem	Arch		Validation set		Test set		5%	10%	Epochs	Test range
			err	classif.err	err	classif.err				
cancer1	4+2	l	1.53	1.714	1.053	1.149	0	3	75-205	1.176-1.352
cancer2	8+4	l	1.284	1.143	4.013	5.747	0	0	95	—
cancer3	4+4	l	2.679	2.857	2.145	2.299	2	12	55-360	2.112-2.791
card1	4+4	l	8.251	9.827	10.35	13.95	15	23	20-65	10.02-11.02
card2	4+0	l	10.30	10.98	14.88	18.02	6	20	20-50	14.27-16.25
card3	16+8	l	7.236	8.092	13.00	18.02	0	1	50-55	14.52-14.52
diabetes1	2+2	l	15.07	19.79	16.47	25.00	11	23	65-525	16.3-17.52
diabetes2	16+8	l	16.22	21.35	17.46	23.44	4	23	85-335	17.17-18.4
diabetes3	4+4	l	17.26	25.00	15.55	21.35	8	33	65-400	15.65-18.15
gene1	2+0	l	9.708	12.72	10.11	13.37	7	13	30-1245	9.979-11.25
gene2	4+2	s	7.669	11.71	7.967	12.11	0	0	1680	—
gene3	4+2	s	8.371	10.96	9.413	13.62	0	1	1170-1645	9.702-9.702
glass1	8+0	l	8.604	31.48	9.184	32.08	4	21	40-510	8.539-10.32
glass2	32+0	l	9.766	38.89	10.17	52.83	12	31	15-40	9.913-10.66
glass3	16+8	l	8.622	33.33	8.987	33.96	9	35	20-1425	8.795-11.84
heart1	8+0	l	12.58	15.65	14.53	20.00	17	23	40-80	13.64-15.25
heart2	4+0	l	12.02	16.09	13.67	14.78	20	23	25-85	13.03-14.39
heart3	16+8	l	10.27	12.61	16.35	23.91	18	23	40-65	16.25-17.21
heartc1	4+2	l	8.057		16.82	21.33	2	7	35-75	15.60-17.87
heartc2	8+8	l	15.17		5.950	4.000	2	12	15-90	5.257-7.989
heartc3	24+0	l	13.09		12.71	16.00	3	10	10-105	12.95-16.55
horse1	4+0	l	15.02	28.57	13.38	26.37	8	29	15-45	12.7-14.97
horse2	4+4	l	15.92	30.77	17.96	38.46	21	34	15-45	16.55-19.85
horse3	8+4	l	15.52	29.67	15.81	29.67	14	31	15-35	15.63-17.88
soybean1	16+8	l	0.6715	4.094	0.9111	8.824	0	0	1045	—
soybean2	32+0	l	0.5512	2.924	0.7509	4.706	0	1	895-2185	0.8051-0.8051
soybean3	16+0	l	0.7147	4.678	0.9341	7.647	0	3	565-945	0.9539-0.9809
thyroid1	16+8	l	0.7933	1.167	1.152	2.000	1	1	480-1170	1.194-1.194
thyroid2	8+4	l	0.6174	1.000	0.7113	1.278	0	0	2280	—
thyroid3	16+8	l	0.7998	1.278	0.8712	1.500	2	3	590-2055	0.9349-1.1

Arch: nodes in first hidden layer + nodes in second hidden layer, `s`igmoidal or `l`inear output nodes for 'best' network, i.e., network used in the run with lowest validation set error.
Validation set: squared error percentage on validation set, classification error on validation set of 'best' run (missing values are due to technical-historical reasons).
Test set: squared error percentage on test set, classification error on test set of 'best' run.
5%: number of *other* runs with validation squared error at most 5 percent worse than that of best run (as shown in second column).
10%: ditto, at most 10% worse.
Epochs: Range of number of epochs trained for best run and within-10-percent-best runs.
Test range: Range of squared test set error percentages for within-10-percent-best runs excluding the 'best' run.

Table 5: Architecture finding results of classification problems

Problem	Arch		Validation set	Test set	5%	10%	Epochs	Test range
building1	2+2	l	0.7583	0.6450	4	13	625-2625	0.6371-0.6841
building2	16+8	s	0.2629	0.2509	5	20	1100-2960	0.246-0.2731
building3	8+8	s	0.2460	0.2475	5	16	600-2995	0.2526-0.2739
flare1	4+0	s	0.3349	0.5283	10	30	35-160	0.5232-0.5687
flare2	2+0	s	0.4587	0.3214	14	30	35-135	0.3167-0.3695
flare3	2+0	l	0.4541	0.3568	14	32	40-155	0.3493-0.3772
hearta1	32+0	s	4.199	4.428	19	33	35-180	4.249-4.733
hearta2	2+0	l	3.940	4.164	3	23	20-120	3.948-4.527
hearta3	4+0	s	3.928	4.961	15	31	20-105	4.337-5.089
heartac1	2+0	l	4.174	2.665	0	3	50-95	2.613-3.328
heartac2	8+4	l	4.589	4.514	0	2	15	4.346-4.741
heartac3	4+4	l	5.031	5.904	8	16	10-55	4.825-6.540

(The explanation from table 5 applies, except that the test set classification error data is not present here.)

Table 6: Architecture finding results of approximation problems

Problem	pivot arch		w	Problem	pivot arch		w	Problem	pivot arch		w
building1	16+0	l	333	building2	16+8	l	605	building3	16+8	s	605
cancer1	4+2	l	100	cancer2	8+4	l	196	cancer3	16+8	l	436
card1	32+0	l	1832	card2	24+0	l	1400	card3	16+8	l	1528
diabetes1	32+0	l	370	diabetes2	16+8	l	410	diabetes3	32+0	l	370
flare1	32+0	s	971	flare2	32+0	s	971	flare3	24+0	s	747
gene1	4+2	l	1115	gene2	4+2	s	1115	gene3	4+2	s	1115
glass1	16+8	l	572	glass2	16+8	l	572	glass3	16+8	l	572
heart1	32+0	l	1288	heart2	32+0	l	1288	heart3	32+0	l	1288
hearta1	32+0	l	1220	hearta2	16+0	l	628	hearta3	32+0	l	1220
heartac1	2+0	l	110	heartac2	8+4	l	512	heartac3	16+8	s	1052
heartc1	16+8	l	1112	heartc2	8+8	l	744	heartc3	32+0	l	1288
horse1	16+8	l	1793	horse2	16+8	l	1793	horse3	32+0	l	2161
soybean1	16+8	l	4153	soybean2	32+0	l	4841	soybean3	16+0	l	3209
thyroid1	16+8	l	794	thyroid2	8+4	l	398	thyroid3	16+8	l	794

Pivot architecture and the corresponding number w of connections for each data set.

Table 7: Pivot architectures for the datasets

parameters used are the same as for the linear networks as indicated in section 3.3.1. Several interesting observations can be made (please compare also with the discussion of the linear network results in section 3.3.1):

1. The results for some of the problems are *worse* than those obtained using linear networks. This is most notable for the gene problems and less severe for the horse problems and many of the heart disease problems.
2. Not surprisingly, the standard deviations of validation and test set errors and the the tendency to overfit are much higher than for linear networks in most of the cases.
3. The correlation of validation set errors with test set errors is quite small for some of the problems (less than 0.5 for cancer3, card3, flare3, glass1, heartac1, heartc1, horse1, horse2, soybean3). In

of different stopping criteria. Those results, however, are not reported here.

Problem	Training set		Validation set		Test set		ρ	Test set classification		Overfit		Total epochs		Relevant epochs	
	mean	stddev	mean	stddev	mean	stddev		mean	stddev	mean	stddev	mean	stddev	mean	stddev
cancer1	2.87	0.27	1.96	0.25	1.60	0.41	0.81	1.47	0.60	4.48	4.87	152	111	133	97
cancer2	2.08	0.35	1.77	0.32	3.40	0.33	0.51	4.52	0.70	5.76	6.70	93	75	81	72
cancer3	1.73	0.19	2.86	0.11	2.57	0.24	0.28	3.37	0.71	3.37	1.32	66	20	51	16
card1	8.92	0.54	8.89	0.59	10.53	0.57	0.92	13.64	0.85	3.77	4.47	33	7	25	5
card2	7.12	0.55	11.11	0.32	15.47	0.75	0.53	19.23	0.80	3.32	1.03	32	8	22	6
card3	7.58	0.87	8.42	0.37	13.03	0.50	-0.03	17.36	1.61	3.52	1.46	37	10	28	9
diabetes1	14.74	2.03	16.36	2.14	17.30	1.91	0.99	24.57	3.53	2.31	0.67	196	98	118	72
diabetes2	13.12	1.35	17.10	0.91	18.20	1.08	0.77	25.91	2.50	2.75	2.54	119	42	85	31
diabetes3	13.34	1.11	17.98	0.62	16.68	0.67	0.55	23.06	1.91	2.34	0.65	307	193	200	132
gene1	6.45	0.42	10.27	0.31	10.72	0.31	0.76	15.05	0.89	2.67	0.49	46	9	29	6
gene2	7.56	1.81	11.80	1.19	11.39	1.28	0.97	15.59	1.83	2.12	0.44	321	698	222	595
gene3	6.88	1.76	11.18	1.06	12.14	0.95	0.95	17.79	1.73	2.06	0.50	435	637	289	508
glass1	7.68	0.79	9.48	0.24	9.75	0.41	0.33	39.03	8.14	2.76	0.71	67	44	45	39
glass2	8.43	0.53	10.44	0.48	10.27	0.40	0.72	55.60	2.83	4.27	1.75	29	9	20	7
glass3	7.56	0.98	9.23	0.25	10.91	0.48	0.54	59.25	7.83	2.68	0.47	66	46	45	41
heart1	9.25	1.07	13.22	1.32	14.33	1.26	0.97	19.89	2.27	2.83	1.89	65	16	43	12
heart2	9.85	1.68	13.06	3.29	14.43	3.29	0.98	17.88	1.57	3.27	2.34	57	19	38	13
heart3	9.43	0.64	10.71	0.78	16.58	0.39	0.67	23.43	1.29	3.35	3.72	51	10	37	9
heartc1	6.82	1.20	8.75	0.71	17.18	0.79	0.10	21.13	1.49	4.04	2.98	45	12	36	11
heartc2	10.41	1.76	17.02	1.12	6.47	2.86	0.83	5.07	3.37	4.05	1.89	29	14	21	11
heartc3	10.30	1.79	15.17	1.83	14.57	2.82	0.85	15.93	2.93	8.22	18.67	24	13	17	11
horse1	9.91	1.06	16.52	0.67	13.95	0.60	0.30	26.65	2.52	4.66	2.28	28	5	20	4
horse2	7.32	1.52	16.76	0.64	18.99	1.21	0.30	36.89	2.12	3.87	1.49	31	8	22	8
horse3	9.25	2.36	17.25	2.41	17.79	2.45	0.92	34.60	2.84	3.48	1.26	30	10	21	7
soybean1	0.32	0.08	0.85	0.07	1.03	0.05	0.54	9.06	0.80	2.55	1.37	665	259	551	218
soybean2	0.42	0.06	0.67	0.06	0.90	0.08	0.77	5.84	0.87	2.17	0.16	792	281	675	243
soybean3	0.40	0.07	0.82	0.06	1.05	0.09	0.33	7.27	1.16	2.16	0.13	759	233	639	205
thyroid1	0.60	0.53	1.04	0.61	1.31	0.55	0.99	2.32	0.67	3.06	3.16	491	319	432	266
thyroid2	0.59	0.24	0.88	0.19	1.02	0.18	0.85	1.86	0.41	2.58	1.07	660	460	598	417
thyroid3	0.69	0.20	0.97	0.13	1.16	0.16	0.91	2.09	0.31	2.39	0.43	598	624	531	564

Training set: mean and standard deviation (stddev) of minimum squared error percentage on training set reached at any time during training.
Validation set: ditto, on validation set.
Test set: mean and stddev of squared test set error percentage at point of minimum validation set error.
ρ: Correlation between validation set error and test set error.
Test set classification: mean and stddev of corresponding test set classification error.
Overfit: mean and stddev of GL value at end of training.
Total epochs: mean and stddev of number of epochs trained.
Relevant epochs: mean and stddev of number of epochs until minimum validation error.

Table 8: Pivot architecture results of classification problems

Problem	Training set		Validation set		Test set		ρ	Overfit		Total epochs		Relevant epochs	
	mean	stddev	mean	stddev	mean	stddev		mean	stddev	mean	stddev	mean	stddev
building1	0.63	0.50	2.43	1.50	1.70	1.01	0.96	31.93	44.07	394	602	329	529
building2	0.23	0.02	0.28	0.02	0.26	0.02	0.98	0.11	0.70	1183	302	1175	303
building3	0.22	0.02	0.26	0.01	0.26	0.01	0.93	0.42	1.09	1540	466	1408	505
flare1	0.39	0.26	0.55	0.81	0.74	0.80	1.00	3.13	2.48	71	28	52	21
flare2	0.42	0.16	0.55	0.43	0.41	0.47	1.00	3.20	3.73	60	15	42	10
flare3	0.36	0.01	0.49	0.01	0.37	0.01	0.32	2.58	0.58	76	28	51	18
hearta1	3.75	0.76	4.58	0.81	4.76	1.14	0.95	4.98	7.85	46	16	34	13
hearta2	3.69	0.87	4.47	1.00	4.52	1.10	0.97	7.18	24.23	59	21	45	19
hearta3	3.84	0.66	4.29	0.73	4.81	0.87	0.97	5.34	14.19	45	13	35	13
heartac1	3.86	0.32	4.87	0.23	2.82	0.22	-0.06	3.98	2.25	44	23	34	21
heartac2	3.41	0.42	5.51	0.65	4.54	0.87	0.79	7.53	5.27	22	9	16	7
heartac3	2.23	0.57	5.38	0.37	5.37	0.56	0.80	4.64	2.96	38	10	30	10

(The explanation from table 8 applies, except that the test set classification error data is not present here.)

Table 9: Pivot architecture results of approximation problems

two cases it is even slightly negative (card3, heartac1).

4. The correlation value also differs dramatically between the three variants of some of the problems (card, flare, heartac, heartc, horse).
5. However, low correlation does not necessary imply bad overall test error results (see cancer, card, flare, heartac, horse).
6. The training times exhibit dramatic fluctuations in a few of the cases (building1, gene2, gene3, thyroid3, and less severely cancer1, cancer2, diabetes3, glass1, glass3, thyroid1, thyroid2).
7. The other numbers of training epochs tend to be of the same order as for linear networks, with a few exceptions that are much faster (most of the heart disease problems) or much slower (thyroid, building2, building3).
8. The "inverse" error behavior observed for some of the linear networks is no longer present for most of them (except cancer1, cancer2, card1).

As mentioned above, for some of the problems it might be more appropriate to work without shortcut connections. To quantify the effect of training without shortcut connections, another series of 60 runs per dataset was conducted using the same parameters as above. This time, however, the network architecture used was modified to include only connections between adjacent layers, i.e., no direct connections from the inputs to the outputs and for networks with two hidden layers also no connections from the inputs to the second hidden layer and from the first hidden layer to the outputs. I call these architectures the no-shortcut architectures.

The results of these runs are shown in tables 10 (classification problems) and 11 (approximation problems). Once again, a few interesting observations can be made (compare also with the above discussions of linear network and pivot architecture results):

1. Leaving out the shortcut connections seems to be appropriate more often than expected (see also section 3.3.4).
2. The test error results for the gene problems are better than for linear networks (for pivot architectures they were worse than the for linear networks). However, the classification errors are worse even than for the pivot architectures.

Problem	Training set		Validation set		Test set		ρ	Test set classification		Overfit		Total epochs		Relevant epochs	
	mean	stddev	mean	stddev	mean	stddev		mean	stddev	mean	stddev	mean	stddev	mean	stddev
cancer1	2.83	0.15	1.89	0.12	1.32	0.13	0.64	1.38	0.49	3.10	2.54	116	123	95	115
cancer2	2.14	0.23	1.76	0.14	3.47	0.28	0.14	4.77	0.94	3.82	1.90	54	31	44	28
cancer3	1.83	0.26	2.83	0.13	2.60	0.22	0.59	3.70	0.52	3.33	1.64	54	20	41	17
card1	8.86	0.41	8.69	0.26	10.35	0.29	0.25	14.05	1.03	3.54	1.25	30	7	22	5
card2	7.18	0.51	10.87	0.27	14.94	0.64	0.44	18.91	0.86	3.99	1.52	26	7	17	5
card3	7.13	0.62	8.62	0.46	13.47	0.51	0.41	18.84	1.19	4.81	3.24	29	7	22	6
diabetes1	14.36	1.14	15.93	1.04	16.99	0.91	0.95	24.10	1.91	2.23	0.53	201	119	117	83
diabetes2	13.04	1.27	16.94	0.91	18.43	1.00	0.76	26.42	2.26	2.50	0.50	102	46	70	26
diabetes3	13.52	1.46	17.89	0.90	16.48	1.16	0.91	22.59	2.23	2.32	0.59	251	132	164	85
gene1	2.70	1.52	8.19	1.33	8.66	1.28	0.91	16.67	3.75	2.46	0.53	124	58	101	53
gene2	4.55	2.60	9.46	1.95	9.54	1.91	0.97	18.41	6.93	2.29	0.28	321	284	250	255
gene3	4.99	2.79	9.45	2.17	10.84	1.93	0.97	21.82	7.53	2.33	0.39	262	183	199	163
glass1	7.16	0.65	9.15	0.21	9.24	0.32	0.13	32.70	5.34	2.69	0.64	71	31	52	27
glass2	8.42	0.66	10.03	0.27	10.09	0.28	0.37	55.57	3.70	4.00	1.80	30	9	22	8
glass3	7.54	1.06	9.14	0.24	10.74	0.52	0.73	58.40	7.82	2.97	1.17	60	30	46	26
heart1	9.24	0.82	13.10	0.65	14.19	0.64	0.89	19.72	0.96	3.16	2.38	57	15	38	12
heart2	9.73	1.24	12.32	1.09	13.61	0.89	0.88	17.52	1.14	3.56	3.47	51	15	36	12
heart3	9.46	0.88	10.85	1.39	16.79	0.77	0.93	24.08	1.12	3.91	4.42	46	13	32	10
heartc1	5.98	1.33	8.08	0.49	16.99	0.77	0.22	20.82	1.47	5.08	2.64	38	10	30	9
heartc2	9.85	1.16	16.86	0.70	5.05	1.36	0.40	5.13	1.63	4.83	2.34	25	10	18	9
heartc3	10.35	1.07	14.30	1.21	13.79	2.62	0.75	15.40	3.20	9.73	10.48	17	6	11	5
horse1	10.43	1.23	15.47	0.37	13.32	0.48	0.24	29.19	2.62	6.09	2.53	19	3	13	3
horse2	6.68	1.85	16.07	0.79	17.68	1.41	-0.19	35.86	2.46	4.28	1.67	25	7	18	6
horse3	10.54	1.68	15.91	1.19	15.86	1.17	0.88	34.16	2.32	5.51	3.89	20	5	14	5
soybean1	1.53	0.09	1.94	0.06	2.10	0.07	0.58	29.40	2.50	3.14	1.99	219	112	159	79
soybean2	0.46	0.19	0.59	0.13	0.79	0.22	0.96	5.14	1.05	5.06	6.49	417	222	362	202
soybean3	0.61	0.21	0.93	0.21	1.25	0.15	0.76	11.54	2.32	6.12	7.99	450	273	382	228
thyroid1	0.59	0.20	1.01	0.16	1.28	0.12	0.84	2.38	0.35	3.99	7.14	377	308	341	280
thyroid2	0.60	0.13	0.89	0.11	1.02	0.11	0.59	1.91	0.24	4.71	6.86	421	269	388	246
thyroid3	0.74	0.18	0.98	0.13	1.26	0.14	0.92	2.27	0.32	3.91	9.18	324	234	298	223

(The explanation from table 8 applies)

Table 10: No-shortcut architecture results of classification problems

3. The test error results for the horse problems have also improved, yet are still worse than for linear networks.

4. The correlations of validation and test error are sometimes very different than for the pivot architectures (see for example card, flare, glass, heartac).

5. For flare2 and flare3, although the correlation is much lower, the standard deviations of test errors are very much smaller, compared to pivot architectures.

Problem	Training set		Validation set		Test set		ρ	Overfit		Total epochs		Relevant epochs	
	mean	stddev	mean	stddev	mean	stddev		mean	stddev	mean	stddev	mean	stddev
building1	0.47	0.28	2.07	1.04	1.36	0.63	0.88	33.93	49.93	307	544	248	457
building2	0.24	0.15	0.30	0.19	0.28	0.20	1.00	0.14	0.78	1074	338	1044	330
building3	0.22	0.01	0.26	0.01	0.26	0.01	0.74	0.25	0.58	1380	350	1304	360
flare1	0.35	0.02	0.35	0.01	0.54	0.01	0.10	3.02	0.90	48	20	35	16
flare2	0.40	0.01	0.47	0.01	0.32	0.01	0.43	2.93	0.99	47	11	32	8
flare3	0.37	0.01	0.47	0.01	0.36	0.01	0.34	2.53	0.47	57	21	32	11
hearta1	3.55	0.53	4.48	0.35	4.55	0.41	0.93	4.17	7.53	47	18	35	16
hearta2	3.45	0.56	4.41	0.21	4.33	0.15	0.55	2.91	0.75	54	22	41	20
hearta3	3.74	0.72	4.46	1.01	4.89	0.91	0.99	5.35	9.90	46	17	34	15
heartac1	3.59	0.24	4.77	0.32	2.47	0.38	0.21	3.78	1.85	42	22	32	18
heartac2	2.58	0.42	5.16	0.32	4.41	0.56	-0.15	6.43	4.43	24	7	18	7
heartac3	2.45	0.46	5.74	0.36	5.55	0.52	0.84	5.52	4.02	31	12	23	10

(The explanation from table 8 applies, except that the test set classification error data is not present here.)

Table 11: No-shortcut architecture results of approximation problems

3.3.4 Comparison of multilayer results

Table 12 shows a comparison of the pivot architecture and no-shortcut architecture results presented above. The comparison was performed with the `ttest` procedure of the SAS statistical software

Problem	1	2	3
building	(—)	N 2.1	P 7.8
cancer	N 0.0	—	—
card	N 0.1	N 0.0	P 0.0
diabetes	—	P 2.9	N 2.1
flare	N 0.0	N 0.0	N 0.0
gene	N 0.0	(N 0.0)	(N 0.0)
glass	N 0.0	N 0.1	N 3.2
heart	N 1.6	N 0.6	—
hearta	—	—	P 0.0
heartac	N 0.0	—	(P 0.0)
heartc	—	N 0.0	N 2.6
horse	N 0.0	N 0.0	N 0.0
soybean	P 0.0	(N 0.0)	P 0.0
thyroid	P 6.9	—	P 0.1

Results of statistical significance test performed for differences of mean logarithmic test error between pivot architectures (P) and no-shortcut architectures (N). Entries show differences that are significant on a 90% confidence level plus the corresponding p-value (in percent); the letter indicates which architecture is better. Dashes indicate non-significant differences. Parentheses indicate unreliable test results due to non-normality of at least one of the two samples. The test employed was a t-test using the Cochran/Cox approximation for the unequal variance case. 2.6% of the data points were removed as outliers.

Table 12: t-test comparison of pivot and no-shortcut results

package. Since a t-test assumes that the samples to be compared have normal distributions, the logarithm of the test errors was compared instead of the test errors themselves, because test errors usually have an approximately log-normal distribution. This logarithmic transformation does not change the test result, since the logarithm is strictly monotone; log-normal distributions occur quite often and log-transformations are a very common statistical technique. Since a further assumption of the t-test is equal variance of the samples, the Cochran/Cox approximation for the unequal variance case had to be used, because at least for some of the sample pairs (cancer1, gene1, hearta1) the standard deviations differed by more than factor 2. Furthermore, a few outliers had to be removed in order to achieve an approximate normal distribution of the log-errors: In the 2520 runs for the pivot architectures, there were 4 outliers with too low errors and 61 with too high errors. For the no-shortcut architectures, there were no outliers with too low errors and 66 outliers with too high errors. Altogether this makes for 2.6% of outliers. At most 10% outliers, i.e., 6 of 60, were removed from any single sample, namely from heartac2 and heartc3 (pivot) and from horse3 (no-shortcut). A few of the samples deviated so significantly from a log-normal distribution that the results of the test are unreliable and thus must be interpreted with care. For the pivot architectures, these non-normal samples were those of building1, gene2, and gene3, for the no-shortcut architectures they were building1, gene3, heartac3, and soybean2. No outliers were removed from the non-normal samples. The respective test results are shown in parentheses in the table in order to indicate that they are unreliable. This discussion demonstrates how important it is to work very carefully when applying statistical methods to neural network training results. When applied carelessly, statistical methods can produce results that may look very impressive but in fact are just garbage.

For 10 of the sample pairs no significant difference of test set errors is found at the 90% confidence level (i.e., significance level 0.1). In 9 cases the pivot architecture was better while in 23 cases the no-shortcut architecture was better. This result suggests that further search for a good network architecture may be worthwhile for most of the problems, since the architectures used here were all found using candidate architectures with shortcut connections only and just removing the shortcut connections is probably not the best way to improve on them.

Summing up, the network architectures and performance figures presented above provide a starting point for exploration and comparison using the PROBEN1 benchmark collection datasets. It must be noted that none of the above results used the validation set for training. Surely, improvements of the results are possible by using the validation set for training in a suitable way. The properties of the benchmark problems seem to be diverse enough to make PROBEN1 a useful basis for improved experimental evaluation of neural network learning algorithms. Hopefully, many more similar collections will follow.

A Availability of Proben1, Acknowledgements

The PROBEN1 benchmark set (including this report) is available for anonymous FTP from the Neural Bench archive[7] at Carnegie Mellon University (machine `ftp.cs.cmu.edu`, directory `/afs/cs/project/connect/bench/contrib/prechelt`) and from machine `ftp.ira.uka.de` in directory `/pub/neuron`. The file name in both cases is `proben1.tar.gz`. This file contains the complete directory tree, including all data, documentation, and the techreport. The size of the file is about 2 MB[8]. When unpacked, the PROBEN1 benchmark set needs about 20 MB disk space. Of these, the actual data files consume about 15 MB.

The present report alone is available for anonymous FTP from machine `ftp.ira.uka.de` in directory `/pub/papers/techreports/1994` as file `1994-21.ps.Z`.

The original datasets on which the PROBEN1 datasets are based are included in the tar files. Their sources are the UCI machine learning databases repository and the energy predictor shootout

[7] Maintained by Scott Fahlman and collaborators. Many thanks to them for their service.

[8] The file is a GNU `gzip`'ed Unix tar format file. The GNU `gzip` compression utility is needed to uncompress it.

archive. The UCI machine learning databases repository is available by anonymous FTP on machine `ics.uci.edu` in directory `/pub/machine-learning-databases`. This archive is maintained at the University of California, Irvine, by Patrick M. Murphy and David W. Aha. Many thanks to them for their valuable service. The databases themselves were donated by various researchers; thanks to them as well. See the documentation files in the individual dataset directories for details. The building problem is from the energy predictor shootout archive at `ftp.cs.colorado.edu` in directory `/pub/distribs/energy-shootout`.

If you publish an article about work that used PROBEN1, it would be great if you dropped me a note with the reference to prechelt@ira.uka.de.

B Structure of the Proben1 directory tree

The PROBEN1 directory tree that results from unpacking the archive file has the following structure. The top directory is called `proben1`; it contains a `README` file for a quick overview, a `Doc` subdirectory, a `Scripts` subdirectory, and one subdirectory per problem, named like the problem itself.

The `Doc` directory contains this report as both a TEX `dvi` file and as a Postscript file.

The `Scripts` directory contains a number of small Perl scripts that I have used during the preparation of the datasets. I include them in the PROBEN1 distribution for all those people who want to generate additional datasets in the PROBEN1 file format or who want to change the representation used in one of the original PROBEN1 problems. These scripts are not needed for normal use of the PROBEN1 datasets.

Each problem subdirectory for a problem `xx` contains the following files: `README` gives an overview of the files in the directory plus a short description of the attribute encoding used in the PROBEN1 representation of the problem compared to the original representation. `xx1.dt, xx2.dt`, and `xx3.dt` are the actual data files. The only difference between them is that the examples are in a different order (which is always a random permutation, except for `building1` and `thyroid1`). `raw2cod` is the Perl script that was used to convert the original data file into the PROBEN1 data file. This script is the definitive documentation of the problem representation used (with respect to the original data). The problems `heart, hearta, heartac,` and `heartc` are all in the directory `heart`.

C Proben1 file format and data encoding

The following is what a data file looks like (example from glass1.dt):

```
bool_in=0
real_in=9
bool_out=6
real_out=0
training_examples=107
validation_examples=54
test_examples=53
0.281387 0.36391 0.804009 0.23676 0.643527 0.0917874 0.261152 0 0 1 0 0 0 0 0
0.260755 0.341353 0.772829 0.46729 0.545966 0.10628 0.255576 0 0 0 1 0 0 0 0
[further data lines deleted]
```

Each line after the header lines represents one example; first the examples of the training set, then those of the validation set, then those of the test set. The sizes of these sets are given in the last three header lines (the partitioning is always 50%/25%/25% of the total number of examples). The first four header lines describe the number of input coefficients and output coefficients per example. A boolean coefficient is always represented as either 0 (false) or 1 (true). A real coefficient is represented as a decimal number between 0 and 1. For all datasets, either bool_in or real_in is 0 and either bool_out or real_out is 0. Coefficients are separated by one or multiple spaces; examples (including the last) are terminated by a single newline character. First on each line are the input coefficients, then follow the output coefficients (i.e., each line contains `bool_in` + `real_in` + `bool_out` + `real_out` coefficients). Thus, lines can be quite long.
That's all.

The encoding used in the data files has all inputs and outputs scaled to the range $0 \ldots 1$. The scaling was chosen so that the range is at least almost (but not always completely) used by the examples occurring in the dataset. The `gene` datasets are an exception in that they use binary inputs encoded as -1 and 1.

D Architecture ordering

The following list gives for each problem the order of architectures according to increasing squared test set error. Of all architectures tried (as listed in section 3.3.2) only those are listed whose test set error was at most 5% larger than the smallest test set error found in any run. Architectures with linear output units can occur twice, because two runs were made for them and each run is considered separately.

building1: 2+2l, 4+2l, 4+4l, 4+0l, 16+0l.
building2: 16+8s, 16+8l, 8+4s, 16+8l, 32+0s, 32+0l.
building3: 8+8s, 32+0s, 4+4l, 32+0l, 16+8s, 8+4s.
cancer1: 4+2l.
cancer2: 8+4l.
cancer3: 4+4l, 16+8l, 16+0l.
card1: 4+4l, 8+8l, 8+8l, 16+0l, 8+4l, 8+0l, 16+8l, 4+2l, 4+4l, 32+0l, 4+2l, 8+4l, 16+0l, 2+0l, 4+0l, 24+0l.
card2: 4+0l, 16+0l, 16+8l, 24+0l, 2+2l, 8+4l, 4+4l.
card3: 16+8l.
diabetes1: 2+2l, 2+2l, 4+4l, 4+4l, 32+0l, 8+4l, 16+0l, 4+0l, 16+8l, 8+0l, 24+0l, 16+0l.
diabetes2: 16+8l, 24+0l, 8+0l, 8+4l, 4+4l.
diabetes3: 4+4l, 8+0l, 32+0l, 24+0l, 8+8l, 24+0s, 2+2l, 32+0l, 8+8l.
flare1: 4+0s, 2+2l, 4+0l, 2+2s, 2+0l, 32+0s, 4+2l, 2+2l, 2+0s, 4+0l, 16+0s.
flare2: 2+0s, 4+0s, 8+0s, 8+8s, 16+0s, 2+2s, 24+0s, 2+0l, 4+0l, 32+0s, 4+0l, 2+0l, 4+2s, 4+4l, 8+4s.
flare3: 2+0l, 2+0l, 2+0s, 4+0s, 2+2s, 2+2l, 2+2l, 4+4s, 16+0s, 16+0l, 4+0l, 4+4l, 8+0l, 24+0s, 8+8l.
gene1: 2+0l, 2+0s, 4+0l, 2+2l, 2+0l, 2+2l, 4+0l, 4+2l.
gene2: 4+2s.
gene3: 4+2s.
glass1: 8+0l, 16+8l, 4+0l, 32+0s, 8+4l.
glass2: 32+0l, 2+2s, 16+0s, 32+0s, 2+0l, 16+8l, 4+4s, 8+0s, 16+8s, 4+0s, 16+8l, 16+0l, 2+0s.
glass3: 16+8l, 2+0s, 16+0l, 16+0l, 8+4s, 16+8s, 8+8s, 8+4l, 2+0l, 16+8l.

heart1: 8+0l, 24+0l, 4+0l, 32+0l, 16+8l, 8+4l, 32+0l, 8+8l, 16+0l, 4+2l, 4+0l, 4+4l, 8+4l, 24+0l, 2+2l, 4+4l, 8+0l, 16+8l.
heart2: 4+0l, 16+0l, 32+0l, 4+0l, 8+0l, 32+0l, 4+4l, 8+8l, 2+2l, 8+4l, 16+8l, 2+0l, 2+2l, 24+0l, 2+0l, 4+2l, 4+2l, 16+0l, 24+0l, 8+8l, 4+4l.
heart3: 16+8l, 2+0l, 32+0l, 4+0l, 8+0l, 2+2l, 32+0l, 8+8l, 16+0l, 4+2l, 4+4l, 16+0l, 16+8l, 4+4l, 8+4l, 8+4l, 4+2l, 24+0l, 24+0l.
hearta1: 32+0s, 8+4l, 4+0l, 4+4s, 32+0l, 2+0l, 8+0l, 8+4l, 8+8l, 16+8l, 8+0l, 4+0l, 2+2l, 32+0l, 24+0l, 4+2l, 4+4l, 16+0l, 16+8s, 8+8s.
hearta2: 2+0l, 8+4l, 16+0l, 4+2s.
hearta3: 4+0s, 16+0l, 4+4l, 4+0l, 8+0l, 4+4l, 8+4l, 8+0l, 2+0l, 32+0l, 4+2s, 24+0l, 8+8l, 16+8l, 4+2l, 16+8l.
heartac1: 2+0l.
heartac2: 8+4l.
heartac3: 4+4l, 8+4s, 8+0l, 4+0l, 24+0l, 16+8s, 4+0s, 16+0s, 8+0s.
heartc1: 4+2l, 8+8l, 16+8l.
heartc2: 8+8l, 2+2l, 4+0l.
heartc3: 24+0l, 32+0l, 8+8l, 16+8l.
horse1: 4+0l, 4+4l, 4+4l, 4+2s, 2+2l, 8+0s, 16+8l, 4+0s, 16+0l.
horse2: 4+4l, 4+0l, 8+0s, 2+2s, 8+4l, 4+4s, 8+0l, 2+2l, 8+8l, 4+4l, 2+0l, 4+2l, 16+0l, 16+8l, 8+0l, 16+8l, 2+0s, 16+8s, 4+0l, 8+8s, 4+2s, 8+4s.
horse3: 8+4l, 8+0l, 8+4s, 4+0l, 4+4s, 2+0l, 8+8l, 16+0l, 4+4l, 4+4l, 32+0l, 24+0s, 4+2s, 8+0l, 16+8l.
soybean1: 16+8l.
soybean2: 32+0l.
soybean3: 16+0l.
thyroid1: 16+8l, 8+8l.
thyroid2: 8+4l.
thyroid3: 16+8l, 8+4l, 8+8l.

References

[1] Yann Le Cun, John S. Denker, and Sara A. Solla. Optimal brain damage. In *[22]*, pages 598–605, 1990.

[2] T.G. Dietterich and G. Bakiri. Error-correcting output codes: A general method for improving multiclass inductive learning programs. In *Proc. of the 9th National Conference of Artificial Intelligence (AAAI)*, pages 572–577, Anaheim, CA, 1991. AAAI Press.

[3] Scott E. Fahlman. An empirical study of learning speed in back-propagation networks. Technical Report CMU-CS-88-162, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213, September 1988.

[4] Scott E. Fahlman and Christian Lebiere. The cascade-correlation learning architecture. Technical Report CMU-CS-90-100, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213, February 1990.

[5] Scott E. Fahlman and Christian Lebiere. The Cascade-Correlation learning architecture. In *[22]*, pages 524–532, 1990.

[6] William Finnoff, Ferdinand Hergert, and Hans Georg Zimmermann. Improving model selection by nonconvergent methods. *Neural Networks*, 6:771–783, 1993.

[7] Stuart Geman, Elie Bienenstock, and René Doursat. Neural networks and the bias/variance dilemma. *Neural Computation*, 4:1–58, 1992.

[8] Michael I. Jordan and Robert A. Jacobs. Hierarchical mixtures of experts and the EM algorithm. *Neural Computation*, 6:181–214, 1994.

[9] K.J. Lang, A.H. Waibel, and G.E. Hinton. A time-delay neural network architecture for isolated word recognition. *Neural Networks*, 3(1):33–43, 1990.

[10] K.J. Lang and M.J. Witbrock. Learning to tell two spirals apart. In *Proc. of the 1988 Connectionist Summer School*. Morgan Kaufmann, 1988.

[11] Martin Møller. A scaled conjugate gradient algorithm for fast supervised learning. *Neural Networks*, 6(4):525–533, June 1993.

[12] N. Morgan and H. Bourlard. Generalization and parameter estimation in feedforward nets: Some experiments. In *[22]*, pages 630–637, 1990.

[13] Michael C. Mozer and Paul Smolensky. Skeletonization: A technique for trimming the fat from a network via relevance assessment. In *[21]*, pages 107–115, 1989.

[14] Steven J. Nowlan and Geoffry E. Hinton. Simplifying neural networks by soft weight-sharing. *Neural Computation*, 4(4):473–493, 1992.

[15] Lutz Prechelt. A study of experimental evaluations of neural network learning algorithms: Current research practice. Technical Report 19/94, Fakultät für Informatik, Universität Karlsruhe, D-76128 Karlsruhe, Germany, August 1994. Anonymous FTP: /pub/papers/techreports/1994/1994-19.ps.Z on ftp.ira.uka.de.

[16] Michael D. Richard and Richard P. Lippmann. Neural network classifiers estimate bayesian a-posteriori probabilities. *Neural Computation*, 3:461–483, 1991.

[17] Martin Riedmiller and Heinrich Braun. A direct adaptive method for faster backpropagation learning: The RPROP algorithm. In *Proceedings of the IEEE International Conference on Neural Networks*, San Francisco, CA, April 1993. IEEE.

[18] David Rumelhart and John McClelland, editors. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, volume Volume 1. MIT Press, Cambridge, MA, 1986.

[19] Steen Sjøgaard. *A Conceptual Approach to Generalisation in Dynamic Neural Networks.* PhD thesis, Aarhus University, Aarhus, Danmark, 1991.

[20] Brian A. Telfer and Harold H. Szu. Energy functions for minimizing misclassification error with minimum-complexity networks. *Neural Networks*, 7(5):809–818, 1994.

[21] David S. Touretzky, editor. *Advances in Neural Information Processing Systems 1*, San Mateo, California, 1989. Morgan Kaufman Publishers Inc.

[22] David S. Touretzky, editor. *Advances in Neural Information Processing Systems 2*, San Mateo, California, 1990. Morgan Kaufman Publishers Inc.

[23] Zijian Zheng. A benchmark for classifier learning. Technical Report TR474, Basser Department of Computer Science, University of Sydney, N.S.W Australia 2006, November 1993. anonymous ftp from ftp.cs.su.oz.au in /pub/tr.