

Neural Network Feature Selection in Complex Trait Analysis

Lars Kaderali, ZAIK / Cologne University Bioinformatics Center (CUBIC)

Contact:

Lars Kaderali,

ZAIK/University of Cologne, Weyertal 80, 50931 Koeln, Germany

Tel ++49-221-470 6003

E-Mail kaderali@zpr.uni-koeln.de

Abstract

Neural networks are well-established tools in the pattern recognition community. They have previously been suggested for the analysis of complex genetic traits [Lucek et al., 1998], however with mixed results. While the method showed interesting results and even pointed to two genes previously not identified, also some doubts were raised as to the stability of results in a later analysis [Marinov and Weeks, 2001].

We give a brief overview of neural networks and their application to gene finding in affected-sibling-pair studies. We then show that the method is indeed unstable, identifying different genes and ranking them differently in multiple runs. Worse yet, we show that the method suffers from a high prediction error when the trained network is used to predict affection status from previously unseen marker data, giving an error rate that is higher than would be obtained from mere guessing. An analysis of the causes is given, identifying dataset sparsity and marker dimensionality as the two main concerns. We discuss pruning as a means to control these problems, and then show how the method can be combined with a marker subset selection step carried out by a genetic algorithm. Results are given on a simulated dataset.

Key Words: Pattern Classification, Complex Trait, Linear Classification, Neural Networks

Introduction

The identification of susceptibility genes in complex genetic diseases poses many challenging methodological questions. Complex traits are phenotypes that are not attributable to a single gene locus with classical mendelian inheritance, but instead are generally believed to be influenced by multiple interacting disease loci. In addition, environmental factors can interact with disease genes and should ideally be considered as well in analysis [Schork, 1997].

Traditionally, genetic linkage analysis is done by considering a single putative trait locus at a time, hoping to find the marginal individual effects. The affected-sibling-pair (ASP) method looks at identity-by-descent (IBD) sharing of alleles in affected sibling pairs, assessing one marker at a time and comparing expected and observed IBD sharing [Risch, 1990a,b,c].

Lucek and others [Lucek and Ott, 1997], [Lucek et al, 1998] have suggested to apply artificial neural networks (NNs) to address the complexity stemming from interacting gene loci. Neural networks have found wide use in statistical pattern recognition [Duda et al., 2001], with diverse applications ranging from character recognition tasks to strategy evaluation in chess robots. NNs are inspired by their biological counterparts in the human brain. They consist of several layers of interconnected “neurons”, where each neuron receives input signals from the previous layer, and passes a signal on to the next layer if the sum of the inputs received exceeds a threshold. A dedicated input layer receives its input signals directly from the pattern to classify, and a dedicated output layer represents the classification output. For example, a neural network to recognize handwritten characters of size 10x10 pixels would consist of 100 input neurons, each receiving the colour of one specific pixel as input, and 26 output neurons, with one or more hidden layers in between. Each output neuron corresponds to one “class”, respectively, a character from the English alphabet. If the handwritten character was an “A”, the output neuron corresponding to “A” would emit signal 1, and all other neurons would be zero.

Mathematically, each single neuron in a network calculates a function $f\left(\sum_i w_i x_i\right)$, where the sum is over all inputs x_i to the neuron and the weights w_i are parameters of the network. $f()$ is typically chosen as a nonlinear function, such as the sigmoid.

“Training” a neural network then refers to presenting the training patterns to the input neurons, and adjusting the weights such that the desired output is achieved at the output layer. Usually, this is done iteratively using a gradient descent procedure on the classification error for a given pattern, defined as the sum of the squared differences between the actual and the desired output for each output neuron. After several iterations of training, the network is normally used to predict classes of patterns where the output is not known.

In order to apply neural networks to affected sibling pairs, IBD sharing probabilities are calculated for each marker and serve as inputs for the network, each input neuron corresponding to one marker. The output neurons indicate whether the sibling pair presented is concordant affected, or not. In typical affected sibling pair studies, only families with (at least) two affected siblings are collected, thus concordant unaffected or discordant pairs to train the network are not available. [Lucek et al., 1998] circumnavigate this problem by generating random data as controls. They then train a three-layer-network on the data, where each neuron in one layer is connected to each neuron in the next layer. They use two output neurons, one that indicates whether the pattern is concordant affected or not (and that is assumed to depend on signal and noise in the data), the second one being set to 1 for all patterns and assumed to “attract the noise” only. Furthermore, for m input neurons, \sqrt{m} neurons are used in the hidden layer. In their setting, Lucek et al. calculate IBD sharing probabilities for each parent separately, thus each affected sibling pair generates two training patterns for the neural network, one showing paternal, the other maternal IBD sharing for each marker.

Even though it might be interesting to use neural networks as diagnostic tools to predict phenotypic traits given genotypic data, the key interest in ASP studies is to identify genes conferring to disease risk. Thus, one is interested in analysing the trained network to derive hypotheses on influences of individual genes on the trait under investigation. Lucek et al. use a somewhat ad-hoc procedure to calculate so-called “contribution-values”, which are calculated from the weights in the trained network and represent the influence of each individual gene on the disease. To this end, the contribution value from input neuron i to output neuron k is calculated by first computing

$$u_{ik} = \sum_j w_{ij} w_{jk}$$

where the sum is over all hidden units j , w_{ij} is the weight from input unit i to hidden unit j , and w_{jk} is the weight from hidden unit j to output unit k . From this, the contribution value $CV(i)$ of input neuron i is calculated as

$$CV(i) = |u_{i1} - u_{i2}|,$$

assuming that output neuron 1 representing the affection status attracts signal and noise, and output neuron 2 being constantly set to 1 attracts noise only, thus following the idea (signal + noise) – noise = signal in the analysis.

[Marinov and Weeks, 2001] have remarked later that contribution values and their ranks vary quite considerably from run to run of the neural network, raising some doubts as to the stability of the method. Typically in neural network training, the weights are initialised randomly at the beginning of training, thus results may depend on this random initialisation. Marinov and Weeks show that indeed over different runs on the same data, the contribution values calculated vary considerably, and argue that they were unable to replicate the results presented by Lucek et al. in their paper on the same dataset. They attribute this to the error function optimised in the neural

network having many local minima, in which the gradient descent performed during training gets stuck, results thus being very sensitive to starting conditions.

Generalization and Variability

There is indeed a second problem associated with the neural network, that has not been addressed before in this context. In order to assess the quality of a trained network, it is standard practice in the machine learning field to split the data set available in two independent sets, and use one half for training, the other for validation. During training, the error on the training data set is minimized. The validation set is used to assess the performance on unseen data. Ideally, a low training error would also yield a low error on the validation set, which is indication that the network has indeed picked up the relevant signal contained in the data during training. If, on the other hand, the training error is low but a high error is obtained on the validation set, this indicates that the network has been tuned to recognize the noise during training, it has memorized the training data, but not been able to extract meaningful signal. Only if both training and validation error are low should an analysis of the weights in the network be undertaken to generate hypotheses on trait genes.

As the number of available sibling pairs is small in most studies, it is tempting to use all the data available to train the neural network. We show in the following that this leads to severe overfitting of the network, with very poor generalization properties.

We simulated data with 100 markers on 500 sibling pairs in both training and control data. Data were simulated with 10 markers spaced 10 cM on each of 10 chromosomes, with four genes contributing to disease and 1 protective gene as specified in table I. No phenocopies were introduced in the simulation, and complete penetrance was assumed. Of the 500 sibling pairs simulated for each of training and control data, 250 pairs were concordant affected, and 250

discordant. Table 2 shows the distribution of disease causes for the simulated dataset in concordant affected pairs. The overall population frequency of the simulated trait amounts to about 0.14%. Figure 1 shows results for a classical χ^2 test, which identifies genes 3 and 4 at a significance level of 0.01, and the protective gene 5 at a significance level of 0.05. Genes 1 and 2 are not identified.

Figure 2 shows the mean squared error for 10 runs of a 3-layer-neural network on training and validation data, with 100 input neurons, 10 hidden neurons and 2 output neurons. Simulations were done using the Stuttgart Neural Network Simulator (SNNS), Version 4.2 [Zell et al., 2002]. Output neuron 1 was coded as +1 for concordant affected sibling pairs, and 0 for discordant pairs. Output neuron 2 was set to constantly +1, as described by Lucek et al. Training was done using the standard backpropagation function, with the default parameters. In contrast to the method described by Lucek et al., we use one vector for both paternal and maternal IBD sharing, containing for each marker the expected number of alleles shared IBD (between 0 and 2). We chose this method as it will identify interactions between alleles inherited from different parents, which are not found if two separate vectors are generated for paternal and maternal IBD sharing. It can be seen immediately that even though training is very successful (with a training error converging rapidly to zero), the validation error actually increases. This indicates that the network is learning “noise” that is (randomly) present in the training data, but has nothing to do with the trait and is thus not found in the validation dataset. An analysis of the contribution values as defined by Lucek et al. confirms this suspicion. Contribution values were calculated for the ten independent runs of the neural network. Consistent with the results presented by Marinov and Weeks, the actual CVs and their ranks vary considerably from run to run (fig. 3).

Complexity Control and Pruning

The difficulties arising as shown in the previous section stem from an overly complex decision function with many degrees of freedom, resulting in poor generalization of the results, but also in many local optima and bad repeatability of results. Two culprits complicate matters. One is Bellman's "curse of dimensionality" (too many features), the other is the "curse of dataset sparsity" (too few samples) [Somorjai, 1993]. Typically in genetic studies, each sample (individual or pedigree) is characterized by hundreds to thousands of markers, but only very few samples are available, as sample acquisition is associated with significant work and cost.

To control these culprits, first of all, the complexity of the classification function must be adapted to the sample size. It is known that three-layer neural networks can in principle express any continuous function from input to output, given sufficient hidden neurons, proper nonlinear activation functions and weights [Duda et al., 2001]. This expressive power comes at a price. For our network with 100 input, 10 hidden and 2 output layer neurons there are 1020 weights to be trained. Furthermore, assuming that each of the 100 markers in the simulated dataset either is or is not associated with the disease, there are 2^{100} or approximately 10^{30} possible combinations of disease causes. On the other hand, only 500 training points are available, making it very probable that the neural network picks up random noise present in the data and represents affection status as a function of this noise. Obviously, when the trained network is then validated on an independent set, performance will be very poor.

To complicate matters further, it should be noted that we should expect a high error for even the perfect classifier: IBD sharing of a marker is used as an indication for disease association in ASP studies, assuming that if a marker is associated with the disease, it will be shared IBD by affected sibling pairs. However, on average, we would expect 25% of healthy sibling pairs to share two alleles for any given marker identical by descent as well, and cannot distinguish these two cases.

As a consequence of the arguments given above, we opted to use a two-layer network instead of the three-layer network suggested before. Furthermore, we use only one output neuron, that is set to +1 if the sibling pair is concordant affected, and -1 otherwise. A two-layer network implements in principle a linear classifier (neglecting nonlinearities introduced by the networks activation function). Viewing the input space of d markers as a d -dimensional vector space, geometrically, such a linear classifier inserts a hyperplane into the vectorspace, such that the concordant affected and discordant sibling pairs are on different sides of the hyperplane. Such a hyperplane classifier can be described by a discriminant function

$$g(x) = \sum_{i=1}^d w_i x_i + w_0,$$

where x_i is the IBD score of the i -th marker of a given individual, w weighs each marker, and w_0 is a bias. The goal in training then is to choose the weights w such that $g(x) < 0$ for concordant affected and $g(x) > 0$ for discordant sibling pairs, and the points x with $g(x) = 0$ form the hyperplane. Restraining further, we want to choose the hyperplane such that $g(x) = -1$ for concordant affected sibling pairs, and $g(x) = +1$ otherwise (note that the restriction from the equality can be alleviated by using a sigmoid function around $g(x)$, which however complicates computations in the gradient descent and has not been done by the author).

By choosing $y = \begin{bmatrix} 1 \\ x \end{bmatrix}$ and $a = \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_d \end{bmatrix}$, $g(x)$ can be rewritten in vector form as $g(x) = y^t a$, where y^t

stands for the transpose of y . Furthermore, let Y be the matrix containing the vectors y^j of all pedigrees j in its rows, and let b be the $(-1, +1)$ column vector containing the affection status of the pedigrees. Then, in matrix form, our goal is to find a weight vector a satisfying $Ya = b$. If Y

were non-singular, we could write $a=Y^{-1}b$ and obtain a formal solution at once. However, normally, Y is rectangular. In that case, we can solve for a using the Pseudoinverse

$$Y^* = (Y^t Y)^{-1} Y^t$$

of Y , calculating $a=Y^* b$, where Y^t stands for the transpose of Y . If $Y^t Y$ is singular, we can generalize slightly and calculate

$$Y^* = \lim_{\epsilon \rightarrow 0} (Y^t Y + \epsilon I)^{-1} Y^t,$$

where I is the unit matrix with 1 on the diagonal and 0 everywhere else. It can be shown that using the pseudoinverse in the calculation is equivalent to defining an error $Err(a) = \|Ya - b\|^2$ and minimizing $Err(a)$, for example by a gradient descent procedure. Because the gradient of $Err(a)$ is

$$\nabla Err(a) = 2Y^t (Ya - b),$$

the obvious update rule is for the weights a is

$$a_{k+1} = a_k - \eta_k Y^t (Ya_k - b),$$

where a_0 can be chosen arbitrarily, $\eta_k = \eta_1 / k$ is the learning rate and η_0 is any positive constant.

Our results show that even such a linear classifier suffers from the curses of dimensionality and dataset sparsity (data not shown). It is tempting to use all the variables available for classification, but it is known that the addition of further variables that are not linked to the trait to the optimum set of markers will only increase variance of the prediction (Miller, 2002).

It can safely be assumed that for complex genetic traits, only a few of the markers identified will actually be linked to the disease. One simple approach frequently taken in neural network theory is to force many of the weights close to zero in training, thus restricting the number of input neurons that contribute to the output computed. In standard backpropagation training as

introduced by [Rumelhart and McClelland, 1986], the weight update Δw_{ij} for the weight connecting neurons i and j is calculated as

$$\Delta w_{ij} = \eta \delta_j o_i,$$

where η is the learning rate, o_i the output from neuron i , and δ_j is a function of the difference between target value and actual output of the neuron j . In addition to each update of a weight by standard backpropagation, in weight decay [Werbos, 1988], each weight is decreased by a part d of its old value:

$$\Delta w_{ij}(t) = \eta \delta_j o_i - d w_{ij}(t-1).$$

Thus, weights are driven to zero unless reinforced by backpropagation. Furthermore, connections between neurons are completely removed from the network if their weight falls below a predetermined threshold c .

We tried this with parameters $d = 5 \cdot 10^{-5}$ and $c = 1 \cdot 10^{-3}$, again using the Stuttgart Neural Network Simulator (SNNS) package. Both training and validation error decreased and converged to a value of around 230. The network is pruned considerably, with about 6 connections remaining in the network in each run. Figure 4 shows how often each marker remains in the network in the ten runs, indicating that with several repetitions the method is able to identify trait genes, but single runs still produce some variance. Furthermore, the method is very sensitive to initial parameter choice. In the following, we demonstrate an alternative strategy for marker subset selection using a genetic algorithm.

Marker Subset Selection

Genetic Algorithms are inspired by the process of biological evolution. A population of individuals, for example neural networks, each varying somewhat from another, is used to solve a given problem, and each is scored according to its performance on the problem, its fitness. Then,

the individuals are ranked according to their fitness, and the best are retained and can reproduce – either by accumulating some random mutations, or by mating with another individual and thus combining properties of both. Over many generations, the population will then adapt to the problem given, and hopefully give a good solution.

We suggest to combine a simple two-layer network with a genetic algorithm to select markers as input for the network. Thus, instead of training one single network on the full set of all markers, a whole population of networks is trained, each with just a few randomly chosen markers. Then, the “best” of those networks are combined, and form the next generation, whereas the “weak performers” die off and cannot reproduce. This is iterated over several generations, and the lowest-error network of the final generation is analysed to gather information about causative genes.

As the genetic algorithm requires repetitive training of many networks with different input neuron subsets, we used the network function

$$g(x) = \sum_{i=1}^d w_i x_i + w_o$$

as described above, and the matrix pseudoinverse technique to find a solution in one step. A population of n such linear classifiers is generated, where n is the number of markers in the dataset, and classifier i has only the i -th markers as input. The classifiers are then trained, and each classifier i is assigned a fitness value f_i by linear interpolation based on its error calculated by ten-fold cross-validation on the training data, and the errors of the best and the worst classifier, where the best classifier is assigned a fitness of 1, and the worst classifier a fitness of 0. Then, each classifier is chosen for reproduction with probability

$$P(i) = \frac{e^{f_i/T}}{E(e^{f_i/T})},$$

where the expectation $E(\cdot)$ is over the current generation and T is a control parameter loosely referred to as the temperature, similar to the temperature in simulated annealing. Early in the evolution, T is set high, giving roughly equal probability to all individuals and thus ensuring genetic variability. Later on, the temperature T is decreased, thus giving the fittest individuals best survival chances. We simply chose $T=1/g$, where g is the generation number.

If a classifier is chosen for reproduction, it is assigned to one of two groups with equal probability. In the first group, asexual reproduction is simulated by introducing random mutations to each classifier, i.e. markers are removed or added to the set of markers used by the classifier randomly with a user-defined probability. In the second group, for each marker m separately, it is determined which parent the offspring should match. If the parent chosen uses the marker as input for the classifier, then the new classifier does so as well, otherwise it does not. This procedure effectively “mixes” two classifiers by combining subsets of their inputs into a new classifier.

It is important in the genetic algorithm as well to control for classifier complexity. By combining classifiers and accumulating markers as inputs, the classification error on the training set will decrease (as with more input features it is easier to separate the datapoints), however leading to poor generalization performance. Different methods exist to control classifier complexity. One possibility is to “penalize” overly complex classifiers by reducing their fitness dependent on the number of input features they use. The disadvantage here is that fine-tuning is required to weigh the two goals one against the other in the fitness function, i.e. determine by how much the fitness should be decreased if an additional weight is used. On the other hand, this parameter can be used to tune the sensitivity of the method.

Alternatively, a pruning step as described above could be integrated, keeping weights close to zero by reducing their value by a fraction of their old value in the gradient descent procedure, and pruning weights if they fall below a threshold during training. The disadvantage here is that this method works only with gradient descent training, and not the pseudoinverse technique, which provides significant speed improvements. Furthermore, it is not immediately clear how to apply pruning if cross-validation is used and different weights are pruned in the repetitions.

The third alternative is to adjust the probabilities in the mutation module such that adding a marker to a classifier's input set is much less likely than removing one. We found that values of $p_{\text{insertion}}=0.001$ and $p_{\text{deletion}}=0.1$ seemed to work well for our examples (i.e., for each marker, if the marker is NOT in the input set of a given classifier it is added with probability $p_{\text{insertion}}$; whereas if it already is in the marker set, then it is removed with probability p_{deletion}). However, different probabilities for insertion and deletion alone will not guarantee quick convergence of the genetic algorithm. We have thus used a combination with a penalty for network complexity in the fitness function. For the simulated dataset, it was determined empirically that a penalty of 2 per input neuron resulted in quick convergence and networks with about 5 input neurons.

Results

Figure 5 shows errors for a sample representative run of the genetic algorithm over 150 generations. The chromosome 5, marker 8 locus is identified immediately in the first generation, however is replaced due to the random nature of the genetic algorithm by marker 5.9 in the consecutive iterations, which is reverted in generation 10 where marker 5.8 is reintroduced. Chromosome 7 marker 2 is added in generation 16 with a notable decrease in training and validation error. Marker 10.4 is added in generation 22, and marker 7.2 replaced by 7.1 in

generations 51-53. Finally, marker 2.3 is added in generation 99, after which no more changes occur until the end of the simulation.

To assess the stability of the results, the genetic algorithm was run 10 times. All of the ten runs identified markers 2.3, 5.8 and 7.1. Marker 10.4 was identified by 8 of the ten runs. Three runs also had markers 3.8 and one run marker 3.6, which are not in the proximity of a disease gene. Noteworthy, 7 of the runs produced completely identical results, all reporting markers 2.3, 5.8, 7.1 and 10.4. The remaining 3 runs each had marker 3.8, and one of them also marker 3.6 in the resulting network.

Discussion

Our results indicate two points. First of all, we demonstrate clearly the problems resulting from large dimensionality of the data together with small sample sizes, that pose a severe pitfall to pattern classification tools like neural networks if applied to the analysis of genetic studies.

Secondly, we show that these problems can be controlled by taking specific design precautions, such as limiting the complexity of the classifier and incorporating an additional marker subset selection step as is done with the genetic algorithm.

Neural Networks hold the promise to analyse all markers simultaneously in one analysis, which differs significantly from the current state-of-the-art methods that concentrate on the effects of single loci on the disease at a time. Even though we used only a two-layer neural network in our analysis, such a network will consider linear combinations of markers and should thus have higher power in detecting additive effects of multiple genes with only small individual contributions. As it is infeasible to search all possible marker subsets due to their exponential number, the genetic algorithm implements an efficient marker subset selection method, that converges quickly. Most importantly, results are stable and repeatable over different runs; the

marker subset selection performed by the genetic algorithm thus reduces problems stemming from the curses of dimensionality and dataset sparsity.

Several questions remain open though. Most importantly, still no method is known to assign bias-free significance levels to the markers selected. This could be done by estimates from computer simulation, but a more theoretically founded framework would be highly desirable. We are presently working on improvements in this domain.

Secondly, the method as presented makes use of only parts of the information available, and it seems that including the additional information will ameliorate results even further. The χ^2 hypothesis test makes use of knowledge about the distributions of IBD scores in sibling pairs. Such information could be used to improve ASP classification by the network, and may also serve as a means to assign confidence intervals to classification results. Similarly, knowledge about genomic distances between individual markers is available, and might be usable as a means to control noise. Further work is needed to clarify these ideas.

References

- Carroll L. 2002: Genes, virus implicated in multiple sclerosis. Reuters Health, Sep. 2.
- Duda RO, Hart PE, Stork DG. 2001: Pattern Classification, second edition. Wiley and Sons, New York.
- Lucek PR, Ott J. 1997: Neural Network Analysis of Complex Traits. *Genet Epidemiol* 14:1101-1106.
- Lucek PR, Hanke J, Reich J, Solla SA, Ott J. 1998: Multi-Locus Nonparametric Linkage Analysis of Complex Trait Loci with Neural Networks. *Hum Hered* 48:275-284.
- Marinov M, Weeks DE. 2001: The Complexity of Linkage Analysis with Neural Networks. *Hum Hered* 51:169-176.
- Miller A. 2002: Subset Selection in Regression. 2nd ed., Boca Raton.
- NIH, National Institute of Disorders and Stroke. 1996. Multiple Sclerosis: Hope through research. NIH Publication No. 96-75.
- Risch N. 1990a: Linkage strategies for genetically complex traits. I. Multilocus models. *Am J Hum Genet* 46:229-241.
- Risch N. 1990b: Linkage strategies for genetically complex traits. II. The power of affected relative pairs. *Am J Hum Genet* 46:229-241

- Risch N. 1990c: Linkage strategies for genetically complex traits. III. The effect of marker polymorphism on analysis of affected sib pairs. *Am J Hum Genet* 46:242-253.
- Robertson NP, Compston DAS. 1995: Surveying multiple sclerosis in the united kingdom. *J. Neurol Neurosurg Psych* 58, 2-6.
- Robertson NP, Fraser M, Deans J, Claiton D and Compston DAS, 1996: Age adjusted recurrence risks for relatives of patients with multiple sclerosis. *Brain* 119:449-455.
- Rumelhart DE, McClelland JL. 1986 : *Parallel Distributed Processing, Volume 1*. MIT Press.
- Sadovnick AD, Baird PA, Ward RH, 1988 : Multiple Sclerosis ; updated risks for relatives. *Am J Med Genet* 29:533-541.
- Sawcer, S. et al. 1996 : A genome screen in multiple sclerosis reveals susceptibility loci on chromosome 6p21 and 17q22. *Nat Genet* 13:464-468.
- Schork NJ. 1997: Genetics of complex disease: approaches, problems and solutions. *Am J Respir Crit Care Med* 156:S103-S109
- Somorjai RL, Nikulin A, 1993: The curse of small sample sizes in medical diagnosis via MR spectroscopy. *Proc Soc Magn Reson Med Twelfth Annual Scientific Meeting*. New York, pp 685.
- Werbos P. 1999: Backpropagation: Past and future. In *Proceedings of the IEEE International Conference on Neural Networks*, p. 343-353, IEEE Press.
- Zell A. et al. 2002 : *SNNS – Stuttgart Neural Network Simulator, User Manual, Version 4.2*, University of Stuttgart. Available from <ftp.informatik.uni-stuttgart.de>.

Tables and Figures

Gene	Chromosome	Marker	Distance	Probability	Modus	Contribution
1	1	2	5	0.01	Dominant	40%
2	2	5	1	0.1	Dominant	10%
3	5	8	2	0.05	Dominant	50%
4	7	1	0	0.005	Dominant	80%
5	10	5	5	0.3	Dominant	-100%

Table I. Parameters used for simulated dataset. 10 markers evenly spaced at 10 cM were simulated on each of 10 chromosomes. Shown are the locations of the four disease genes and one protective gene, together with the probability of carrying the diseased allele and the contribution of the allele to the disease phenotype.

Gene 1	Gene 2	Gene 3	Gene 4	Gene 5	Cases
		x	x		164
x	x	x			44
	x	x	x		15
x			x		13

Table II. Distribution of genetic causes in 250 concordant affected cases (training data). The remaining 14 affected pairs were distributed over the other cases, where in 4% of the data two siblings were both affected with different genetic causes.

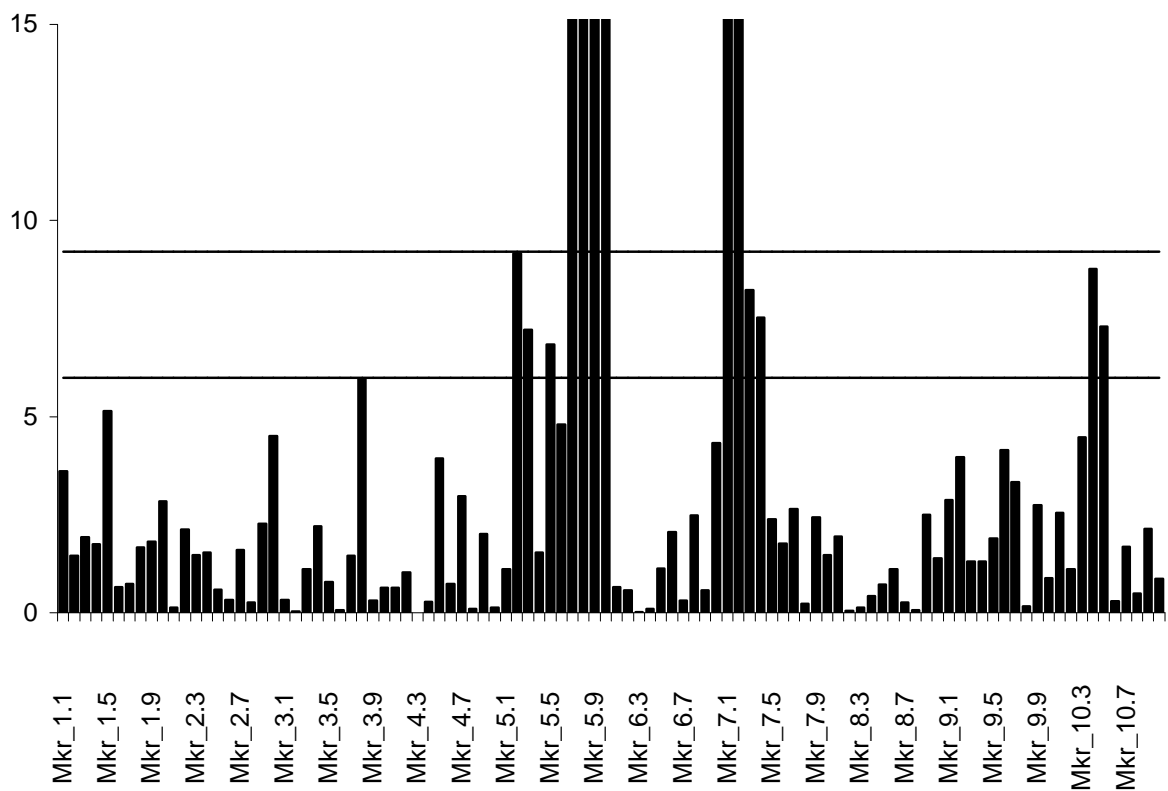


Figure 1. Chi-squared values for the simulated dataset. The two horizontal lines show the significance levels for $\alpha = 1\%$ and $\alpha = 5\%$, respectively. The loci on chromosomes 5 and 7 are detected at a significance level 0.01, the locus on chromosome 10 at a significance level of 0.05, whereas the loci on chromosome 1 and 2 are not detected.

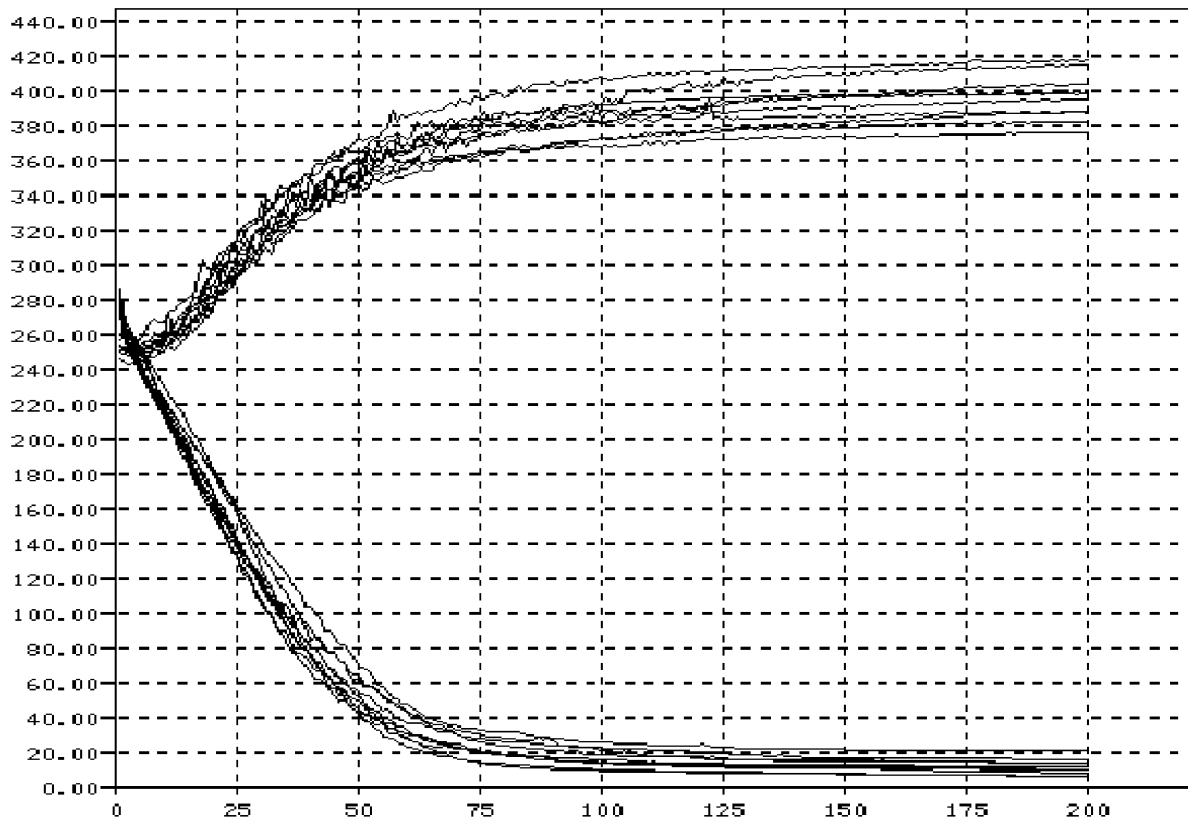


Figure 2. Sum-squared error of 10 runs of a 3-layer neural network on the simulated data. The x-axis shows the number of training steps performed, the y-axis represents the error on training and validation dataset. Decreasing curves correspond to training error, increasing curves to validation error.

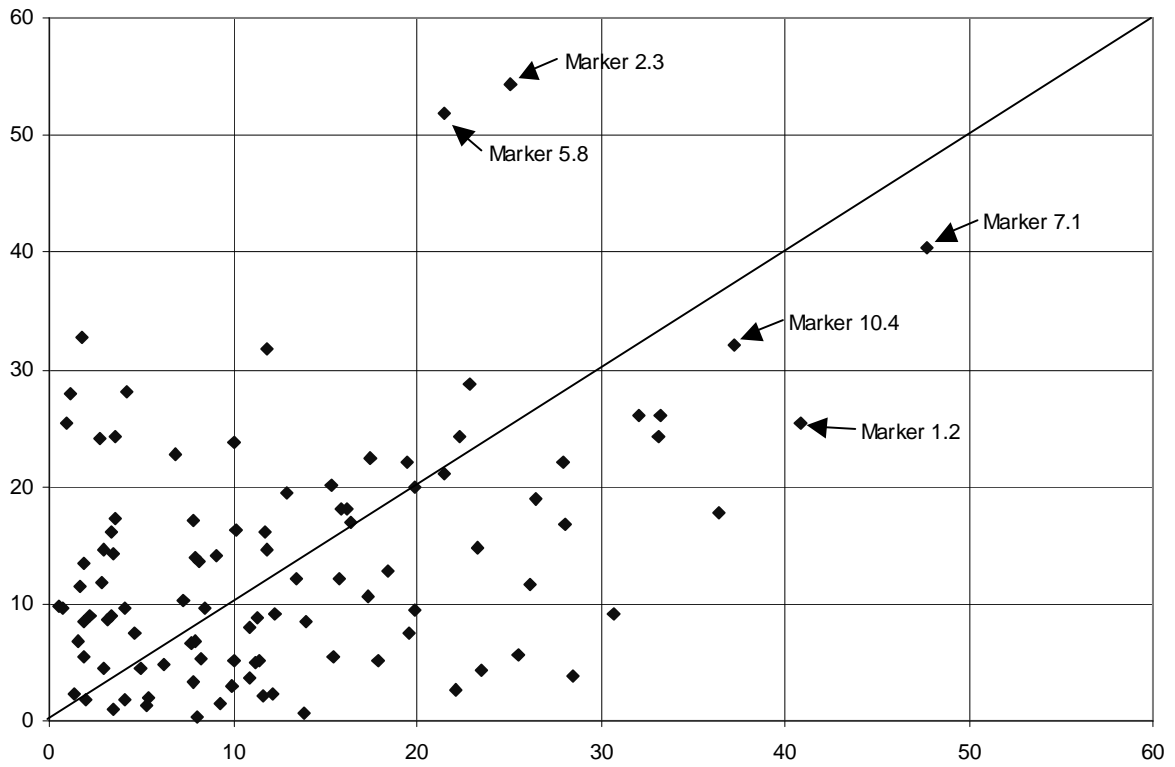


Figure 3. Contribution values for two sample runs on the simulated dataset. CVs for first run are shown on X axis, for second run on Y axis. Perfect correlation between two runs would put all points on the diagonal.

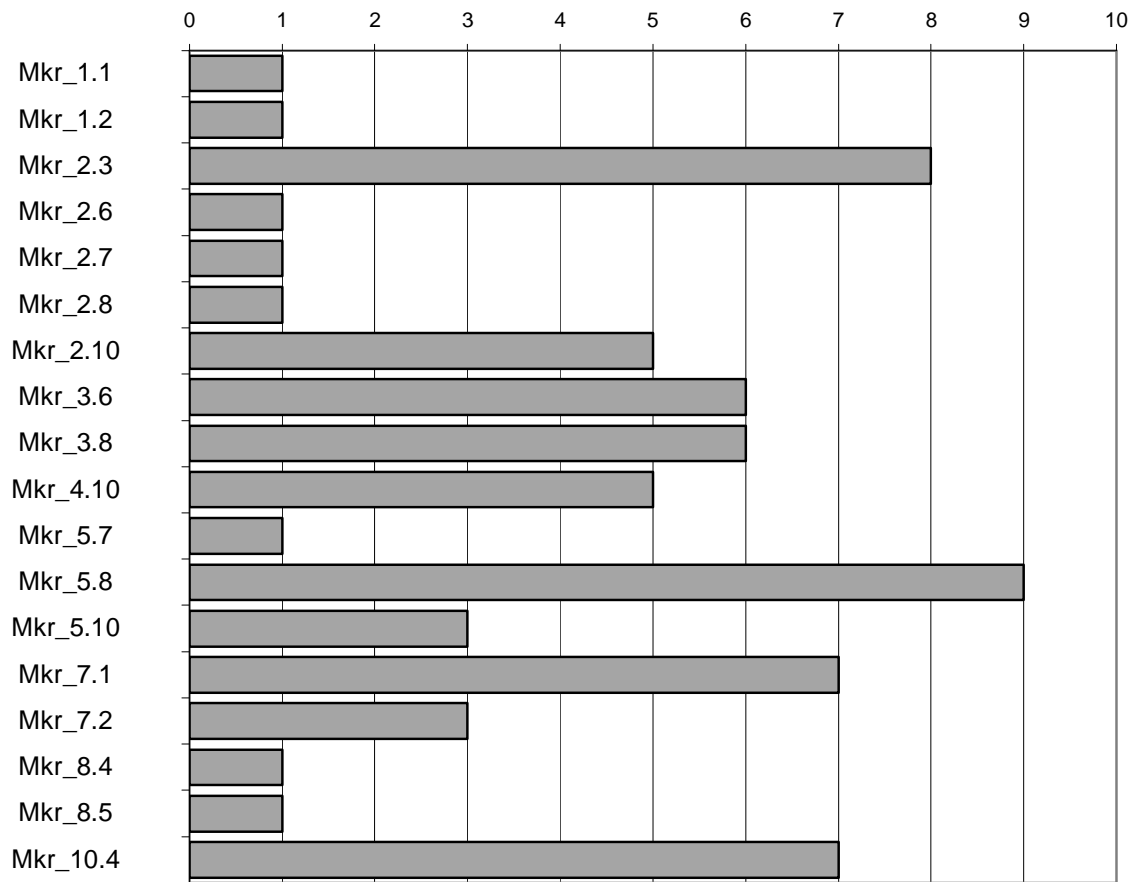


Figure 4. Frequency of markers remaining in pruned 2-layer neural network for 10 runs on the simulated dataset. The y-axis shows how often the respective marker was left in the network.

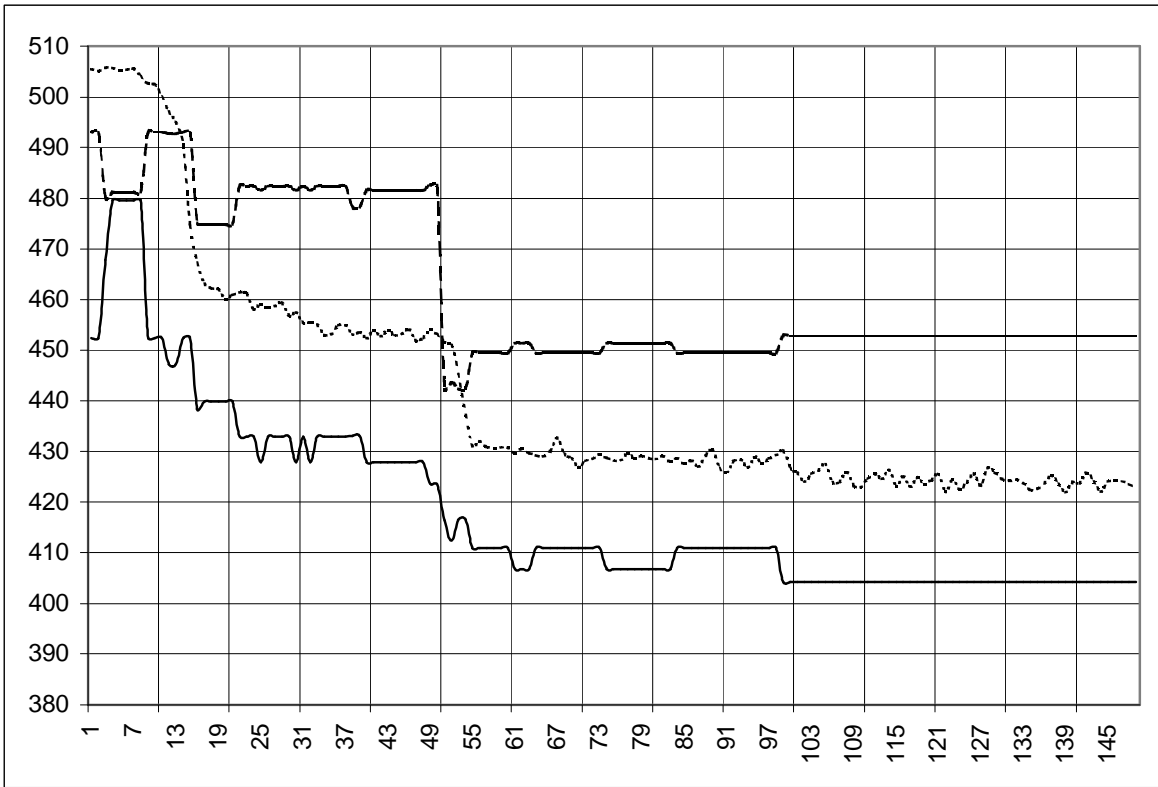


Figure 5. Training and validation error of fittest individual and average training error of entire population for genetic algorithm over 150 generations on simulated dataset. The upper dashed curve shows the validation error of the fittest individual in each generation, the lowest curve shows its training error. The decreasing dotted curve in the middle is the average training error of all 100 classifiers in each generation. Note that the expected error for “guessing” would be 500.