Towards Fair Comparisons of Connectionist Algorithms through Automatically Optimized Parameter Sets

Frank E. Ritter

Department of Psychology
Carnegie Mellon University
Pittsburgh, PA 15213
Ritter@cs.cmu.edu

Abstract

The learning rate and convergence of connectionist learning algorithms are often dependent on their parameters. Most algorithms, if their parameters have been optimized at all, have been optimized by hand. This leads to absolute and relative performance problems. In absolute terms, researchers may not be getting optimal performance from their networks. In relative terms, comparisons of unoptimized or hand optimized algorithms may not be fair. (Sometimes even one is optimized and the other not.) This paper reports data suggesting that comparisons done in this manner are suspect. An example algorithm is presented that finds better parameter sets more quickly and fairly. Use of this algorithm (or similar techniques) would improve performance in absolute terms, provide fair comparisons between algorithms, and encourage the inclusion of parameter set behavior in algorithmic comparisons.

Keywords: Connectionist learning algorithms' parameters, Genetic algorithms, Comparative cognitive modeling, Learning algorithm optimization.

Connectionist networks have been used as a cognitive modeling and machine learning algorithm in numerous ways (for a plethora of examples see Rumelhart, McClelland and the PDP Research Group (1986) or any of the Proceedings of Cognitive Science, AAAI or IJCNN). Work with connectionist learning algorithms such as back-prop (Rumelhart, Hinton, & Williams 1986), Boltzmann learning (Ackley, Hinton, & Sejnowski 1985), the mean field theorem (Peterson & Hartman 1989), and even genetic algorithms (Montana & Davis 1987) have restricted themselves to optimizing connection weights (or additionally, the number of hidden nodes) within a given network, leaving certain crucial parameters of these algorithms, such as learning rate, to be set by the user. These parameters, unlike the weights, are not optimized automatically, yet optimal performance depends on them (Kolen & Pollack 1990, McClelland & Rumelhart 1988). Finding fast yet safe values for these parameters represents an important but often neglected research area. The parameters are also theoretically important because conclusions about the algorithms and the models using them are often based on how quickly the models learn and if they converge (e.g., Hinton & Shallice 1989, Small 1990).

Researchers currently attempt to understand and improve their networks by tuning these parameters by hand. Sometimes this work is reported in the literature (e.g., Fahlman 1988), and attempts are made to teach it (McClelland & Rumelhart 1988). While a certain understanding comes from observing networks as they work, using researchers to do this exclusively will pay an inherently large cost and suffer from the known pitfalls of human thinking, such as functional fixedness, limited memory, and slips.

Comparisons of different algorithms may optimize the parameter set by hand for each algorithm (Fahlman 1988) or may optimize one by hand and use the default values for the other (Montana & Davis 1989). At best, these comparisons are comparing how well each reporting researcher can optimize each algorithm. This is useful information about the algorithms' ease of use, but does not necessarily indicate which algorithm learns faster or better. There is little reason to believe that most algorithm researchers or users are finding or using the best parameters; the brief evidence reported below suggests that this is not the case in practice. (For an excellent counter-example, see Belew, McInerney & Schraudolph 1990. While they don't argue the same theoretical position, their results are consonant with the work reported here.)

Alternatively, researchers may just report the results for a variety of parameter settings and let the reader judge for themselves parameter sensitivity and optimality. Kruschke & Movellan (1991) do this, and the results are compelling because the algorithms are variants of each other and technically speaking, one dominates the other. In general however, this approach breaks down from the combinatorial explosion of possible parameter sets when additional parameters and values are considered.

It should not be initially surprising that connectionist networks are still optimized by hand. The algorithms that connectionist networks exploit and that their designers are

After randomly creating an initial generation of 30 members, Mendel-DP computed their fitness as the number of epochs needed to learn the XOR logic function or the artificial grammar to a stopping criterion (ecrit) of 0.1 TSS. The maximum number of epochs allowed was 400. Network runs that hadn't learned by 400 epochs were assigned 400 as a fitness measure. In this algorithm, lower numbers represented better fitness. After the best genotype was reported, the next generation was created from the members of the current generation. This cycle of creation and evaluation was carried out for 15 generations. The proposed best solutions from each of the subjects and from each run of Mendel-DP were then averaged over 150 trials for comparison.

The genetic operators used. The best 20% of the genotypes were directly copied (survived) into the next generation. The remaining 80% were created by breeding with 4 operators. Of the four operators used to create new genotypes, two of these, cross-over and mutation, can be considered traditional (Holland 1975), and two, creep-value and average, are relatively novel (Davis & Ritter 1987). *Cross-over* selected two parents (as described below) and for each parameter in the child, a value was randomly chosen from the two parents. *Mutation* randomly changed a parameter value in a genotype (choosing linearly in its range) and put the modified version into the next generation. *Creep-value* modified an interval parameter in a copy of the parent, creeping up or down 5%. *Average* averaged the interval parameter values from two parents to create a child, and copied the categorical values from one of them.

The probability of applying each operator was adjusted by a set amount each generation. The initial and final values (and thus also the change each generation) for each operator were copied with slight modification, from a similar genetic algorithm (Davis & Ritter 1987). The total probability of an operator applying was normalized to one by appropriately setting the probability of a *pass-through* operator that merely copied a selected parent into the next generation. Cross-over started at .3 and was adjusted to .1; mutate started as .1 and was adjusted down to .01; creep-value started at .01 and was increased to .1; average started at .3, and was adjusted down to .1; pass-through, which normalized the probability, increased from .29 to a final value of .69.

Parent selection. After an uniform random selection from the 30 member genotype pool, each potential parent was probabilistically selected as a parent based on their fitness compared to the best in their generation according to the following formula:

$$P(\text{genotype}_X)=\text{min-fitness}_{\text{Gen}_i}/\text{Fitness}(\text{genotype}_X)$$

For example, if selected, the best genotype in a generation would be automatically used if selected (P(best) = min/min = 1); poor genotypes with fitness of 400 might have a 80/400 or 1/5 chance of becoming a parent once selected.

Extensions to genetic algorithms to handle noise. Each time BP runs, it initializes the network with a different set of random weights. This causes the learning rate to vary wildly. Mendel-DP used two approaches to decrease the effect of this noise on its fitness evaluation: 1) Fitness was computed as the average across ten network runs. 2) Fitness was saved and averaged across generations for genotypes that survived, further smoothing the effect of an individual evaluation.

Comparison of the Results

Table 1 shows the mean fitness values for the various parameter sets. The top half of the table shows the parameter sets for XOR and the bottom half, the grammar learning network. The results for the default values comes first, then the subjects' results, and then the results from Mendel-DP, our example automatic parameter set optimizer. Interestingly, the results particularly surprised the grammar learning researcher in several ways: he believed that his parameters were optimal and that they gave him learning in approximately 300 epochs.

The XOR network did not converge with the default values. For this problem they are inappropriate, and this is even well known (McClelland & Rumelhart 1988). On the grammar learning network they did better than the researcher (but not reliably better, $t(149)=1.69$, $p> .05$).

The parameter sets found by Mendel-DP lead to faster learning than the default set or those produced by hand. For both problems, this improvement was reliable (in all cases $t(149) >= 3.3$, $p<.005$), and these differences were also large enough to be important, between 30% and an order of magnitude faster learning. The lower standard deviation for the grammar learning task also indicates that the learning was vastly more reliable.

Not only did Mendel-DP produce a better parameter set, it took less time. Even on the grammar learning network, Mendel-DP running overnight took less total time than the researcher did. The incremental cost of modifying Mendel-DP to optimize another parameter or to optimize another network is quite small: less than an hour was needed to extend it from XOR to the grammar task.

Robustness of results. Although Mendel-DP is itself a stochastic algorithms, the results reported here should be robust. Genetic algorithms are not sensitive to their own parameters (De Jong 1985) in the same way connectionist algorithms are. The selection of parameters for Mendel-DP should have little influence on its practical (finding good parameter sets) or theoretical results (finding fair parameter sets). On the other hand, a more complete analysis of this

most familiar with do not lend themselves to self optimization. These algorithms depend on the fitness function being local and inexpensive to compute. Evaluating parameter sets are neither. When the optimized function becomes expensive to compute, these algorithms do not optimize efficiently, and the user community rightfully drops them for other methods such as human implemented heuristic search. Such is the current place of the Boltzmann machine: it requires too much time to compute to be currently considered a viable architecture.

Automatic Computation of Parameter Sets

Let us consider a possible algorithm to optimize parameter sets, without comment on whether it is the best possible, just that it will serve as an example (for semi-automatic methods see Nowlan (1991)). Optimizing functions that are expensive to compute, such as the evaluation of network parameters, are the natural domain of genetic algorithms (Goldberg 1989, Holland 1975), a family of algorithms loosely based on Darwinian evolution. They optimize functions without assuming that the search space will be linear. They start with a population of templates for possible solutions (analogous to sets of chromosomes), and evaluate them to determine how well they perform (fitness). After the fitness values are computed, a new population is created. A variety of methods have been used to create the next generation, but in each case the underlying principle has been to including copies of the chromosomes proportional to their fitness, and at each generation to create new combinations by combining two parents' chromosomes. And then repeating the cycle of evaluation and creation.

In the genetic algorithm presented here, Mendel-DP, the chromosomes are sets of parameter values that initialize a network, setting learning rate, learning grain size, and so on. The fitness measure optimized will be how quickly the network learns. In general, this could be any measure of performance or fitness, including how safely the network performs (e.g., misclassification measures favored by Nowlan (1991)) or how many hidden nodes were used. These need not be combined into a single measure, but could be optimized as a vector (Schaffer 1985).

In order to find out how well people can tune parameter sets, several subjects optimized the parameter sets for two networks. Their parameter sets, along with the default set supplied with the algorithm and the parameter sets produced by Mendel-DP, were evaluated by averaging over multiple runs of the networks with each of them. In addition to the ubiquitous XOR, to balance its simplicity we also compared how well an artificial grammar learning network learned with its default parameters, those found and used by an active researcher, and those found by Mendel-DP.

The Networks and Subjects

The problems. There were two networks used. The first was the classic two hidden node XOR network provided as a training problem (McClelland & Rumelhart 1988), the second was an artificial grammar learning network with 9 input nodes, 6 hidden nodes, and 3 output nodes. Both networks learned through the back-propagation (BP) algorithm (Rumelhart, Hinton & Williams 1986) provided as part of McClelland & Rumelhart's (1988) book of examples.

The subjects and the parameters they modified. The subjects optimizing the XOR network (students in a graduate level PDP modeling course at CMU) were allowed to vary any parameter to any value. While they did not modify all parameters, they did modify both interval (e.g., learning rate) and categorical (e.g., training regime) parameters. The active researcher quite naturally manipulated a superset of these parameters. The complete lists are displayed as the column headings in Table 1.

In order to create a stronger comparison, Mendel-DP only manipulated the parameters that the corresponding subject(s) did.

The parameters varied. Intentionally broad limits were imposed on the automatic search performed by Mendel-DP for the parameters used by the subjects. *Learning rate*, which determines how much each training session influences the node weights, was allowed to vary between 0.05 and 12. *Learning grain size* represents how many patterns are presented in each BP training cycle of presentation of patterns, error accumulation, and then weight adjustment. This parameter specifies that either weight updates occur after every single *pattern* presentation or after every *epoch* (presentation of all input patterns). The *training regime* parameter specified the order of pattern presentation. Patterns could be presented in the same sequential order each epoch *(strain)* or in a permuted order *(ptrain)*. As indicated in Table 1, a slightly different set of parameters were varied for the grammar learning network. The remainder of the parameters were set to the defaults provided with the BP program (McClelland & Rumelhart 1988).

The optimizing algorithm.

The genetic algorithm, Mendel-DP, optimized networks through selective breeding of parameter sets. The detailed description provided below is for application and explanation purposes; genetic algorithms are not particularly sensitive to their own parameters (De Jong 1985), so this level of description should not be necessary for the replication of these results or to support any of the theoretical arguments.

The basic algorithm. Each genotype, or evaluated object, was a set of values for the parameters described above.

Table 1 Average epochs to learn (fitness) for generated parameter sets (N=150)

All other parameters left at default values.

A) XOR Task

Source	Epochs to learn	SD	Learning Rate	Learning Grain	Training Regime
BP Defaults	400	0	0.05	Pattern	Strain
Subject A	222	175	8.0	Epoch	Strain
Subject B	179	125	1.0	Pattern	Ptrain
Mendel-DP	130	131	4.6	Epoch	Strain

B) Grammar Learning Task

Source	Epochs to learn	SD	Learning Rate	Learning Grain	Training Regime	Wrange	Momentum
BP Defaults	147	19	0.05	Pattern	Strain	1.0	.90
Researcher	170	165	0.15	Pattern	Strain	2.0	.65
Mendel-DP	12	4	6.33	Pattern	Strain	1.0	.67

new technique on a practical level would require the ability to characterize how long Mendel-DP must be run to optimize a particular network. These differences may be small because finding network parameters may be less difficult than solving the network itself. This may remain an empirical rather than theoretical question because we are examining expected rather than worst case behavior. These results will also depend on the choice of how to penalize networks that get stuck in local minimums.

Non-normal distribution of learning times. There is one other regularity in the data worth noting. It is the large standard deviations in the learning rates, which are as large as their mean values and significantly skewed. For example, the distribution of learning times for the first entry in the table had a skewness measure of 1.75 (value/SE=8.75, $p<.001$), indicating a highly non-normal distribution. Highly skewed distributions decrease the sensitivity of significance tests (Mosteller & Tukey 1977) and presumably of genetic algorithms. Future work comparing parameters or network performance should use transformed data, which will improve the ability to discern reliable differences. The evaluation of Mendel-DP's results should be interpreted with this fact in mind. There may exist significant differences between parameter sets that standard statistical tests on untransformed data will not find, and the seemingly large sample sizes may not be large enough to find optimum values for minor parameters.

Conclusions

Having a computer do our work faster and better, isn't this what we all have dreamed of? We saw above that parameter sets derived by hand were sometimes better than the defaults provided by the algorithm's authors and sometimes worse. But in all cases they were inferior to those derived with the new method presented here. Researchers could use systems like Mendel-DP to find better parameter sets than they could on their own, in a fraction of the time. If Mendel-DP only encourages further explorations in automated algorithm evaluation, it should be considered a success. However, it goes beyond this.

In addition to promising to provide better parameter sets than those found by hand, Mendel-DP and the data it provides make several suggestions for connectionist research. The problem of evaluating learning algorithms because of different comparison criteria has been noted before (Fahlman 1988, Kruschke & Movellan 1991), but the problem of comparing best discovered performances should also be emphasized. Current comparisons often depend on the optimization abilities of the algorithms' designers instead of just the algorithms themselves. The use of Mendel-DP or similar techniques would put algorithms on a level playing field, and provide fair comparisons between algorithms (as one reviewer noted, there may even be biases inherent in this method of comparison, but none are known or theoretically expected, and this approach would still retain the advantage of making the comparison explicit). The large standard deviations and non-normal distributions reported here suggest that hand optimization is particularly difficult and that additional caution has to be taken in making such comparisons. Standard statistics are not valid, and sample sizes need to be large -- two problems that particularly affect human reasoning for the worse.

This approach allows, indeed encourages, new concepts in the evaluation of networks. With automatic processing available, work can go into evaluating the network parameters not only on learning rate, but also for safety and reliability. As noted above, because of the large variance in the data and our limited memory span, these concepts are not necessarily even computable without it. The ability to routinely find optimal or near optimal parameters should encourage these parameters to become part of theories rather than hearsay, and encourage comparisons of the optimum parameters between networks and what influences them. Important regularities of connectionist learning algorithms may not be apparent until finding sets of optimum parameters is routine.

Acknowledgements

This research was partially funded by a training grant from the Air Force Office of Scientific Research, Bolling AFB, DC. Partial computing support was provided by a grant (N00014-86-K-0678) from ONR. I thank Javier Movellan and two anonymous reviewers for their constructive comments, and an anonymous member of my department for his network and comments.

References

Ackley, D., Hinton, G. E., & Sejnowski, T. J. 1985. A Learning Algorithm for Boltzmann Machines. *Cognitive Science* 9(1): 147-169.

Davis, L. W., & Ritter, F. E. 1987. Schedule Optimization with Probabilistic Search. In *Proceedings of the Third Conference on Artificial Intelligence Applications*, 231-236. Hackensack, NJ: IEEE Computer Society.

De Jong, K. 1985. Genetic Algorithms: A 10 Year Perspective. In *Proceedings of an International Conference on Genetic Algorithms and their Applications,* 169-177. Grefenstette, J.J. ed. Pittsburgh: Texas Instruments & U.S. Navy Center for Applied Research.

Fahlman, S. E. 1988. An Empirical Study of Learning Speed in Back-propagation Networks (Tech. Rep. CMU-CS-88-162), Computer Science Department, Carnegie Mellon University.

Goldberg, D. E. 1989. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Reading, MA: Addison-Wesley.

Hinton, G. E., & Shallice, T. 1989. Lesioning a Connectionist Network: Investigations of Acquired Dyslexia (Tech.Rep. CRG-TR-89-3), Department of Computer Science, University of Toronto.

Holland, J. H. 1975. *Adaptation in Natural and Artificial Systems.* Ann Arbor, Michigan: University of Michigan Press.

Kolen, J. F., & Pollack, J. B. 1990. Scenes from Exclusive-Or: Back Propagation is Sensitive to Initial Conditions. In *Proceedings of the Annual Conference of the Cognitive Science Society*, 868-875. Hillsdale, NJ: Lawrence Earlbaum Associates.

Kruschke, J. K., & Movellan, J. R. 1991. Benefits of Gain: Speeded Learning and Minimal Hidden Layers in Back-propagation Networks. *IEEE Transactions on Systems, Man and Cybernetics.* 21(1): 273-280.

McClelland, J. L., & Rumelhart, D. E. 1988. *Explorations in Parallel Distributed Processing: A Handbook of Models, Programs, and Exercises.* Cambridge, Massachusetts: Massachusetts Institute of Technology.

Montana, D. J., & Davis, L. 1989. Training Feedforward Neural Networks Using Genetic Algorithms. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, 762-767. San Mateo, California: IJCAI.

Mosteller, F., & Tukey, J. W. 1977 *Data Analysis and Regression.* Reading, MA: Addison-Wesley.

Nowlan, S. J. 1991. Soft Competitive Adaptation: Neural Network Learning Algorithms Based on Fitting Statistical Mixtures. Ph.D. diss., School of Computer Science, Carnegie Mellon University.

Peterson, C., & Hartman, E. 1989. Explorations of the Mean Field Theory Learning Algorithm. *Neural Networks* 2:475-494.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. 1986. Learning Internal Representations by Error Propagation. In Rumelhart, McClelland, and the PDP Research Group.

Rumelhart, D. E., McClelland, J. L. & the PDP Research Group. 1986. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition Volume 1: Foundations.* McClelland, J. L., & Rumelhart, D. E. eds. Cambridge, Massachusetts: The MIT Press.

Schaffer, J. D. 1985. Learning Multiclass Pattern Discrimination, 74-79. In *Proceedings of an International Conference on Genetic Algorithms and their Applications,* 169-177. Grefenstette, J.J. ed. Pittsburgh: Texas Instruments & U.S. Navy Center for Applied Research.

Small, S. 1990. Learning Lexical Knowledge in Context: Experiments with Recurrent Feed Forward Networks, 479-486. In *Proceedings of the Annual Conference of the Cognitive Science Society.* Hillsdale, NJ: Lawrence Earlbaum Associates.