The Effects of Problem Representation and Network Representation on Training Results of Artificial Neural Networks

A. van den Berg,
supervised by D. Thierens
and G.A.W. Vreeswijk,
September 28, 2013,
7.5 ECTS.

Abstract

There are different ways to obtain a good Artificial Neural Network. When training, the choice of the data set is of importance to the quality of the resulting network. When evolving a network using Genetic Algorithms, it is important that the representation of the network does not interfere with the passing-on of information to next generations. I looked into the effects of data representation on the quality of the trained networks, and I investigated one solution proposed by Thierens (1996) to unheuristically remove redundancies in genotype. I could not verify the results found in the proposed solution.

1 Introduction

In Computer Science, Artificial Neural Networks (ANNs) are models inspired by the workings of animal neural networks. They are mostly used for two different purposes: either they are trained to perform specific tasks, such as pattern recognition tasks, or they are used to gain insight in natural neural capabilities.

ANNs are generally modelled using a simple layered approach with nodes being the processing units and weights between those nodes being the 'strength' between nodes. When training such networks to recognize for example handwritten digits, input data, represented for example as pixels with a specific shade of grey, are fed into the first layer of the network. In the last layer of the network the calculated result can then be found, which hopefully is the correct digit.

There are different ways to acquire a network that performs well on your data. The first method that pops to mind is training. Different training algorithms have been developed to adapt the network to the data depending on the availability of those data, such as Supervised learning, Unsupervised learning and Reinforcement learning. In order to train the network fast, many measures have to be taken into account.
First, the network topology needs to be large enough to be able to abstract over the input data, but it should be small enough to be trained fast, as large networks take longer to process.
Second, the representation of the data should fit the inner workings of ANNs. Moreover, the complexity of the data should match the proposed topology of the network: a smaller amount of (hidden) neurons is able to abstract less than more neurons can, so the relation between the given input and the desired output should be fit to the network.
Third, the training algorithm itself needs to be fast. Some of the fastest trainers are readily available.
One part of the remainder of this paper will go into some depths regarding the second option: data representation. Tests will show how the original data set and the adapted data set compare.

Another method to obtain a good network is through Evolutionary Algorithms. Those algorithms are used to optimize solutions to all kinds of problems, by means of evolution. Their use is based on nature itself, where species adapt to their environment through individuals passing their information (in the form of genes in nature's case) on to the next generation. In Genetic Algorithms (GA's),

'parent' solutions that solve a problem are recombined into 'child' solutions that hopefully perform even better than their parents, all the way until the children perform their task well enough.

The other majority of this paper will deal with GA's and the effect of the representation of an individual neural network on the speed of finding a good solution. Much of this work is based on the paper by [Thierens, 1996] who proposes a means to make the representations more suitable for GA's. There are others who have attempted to improve the performance of the representation of ANNs themselves. [8]

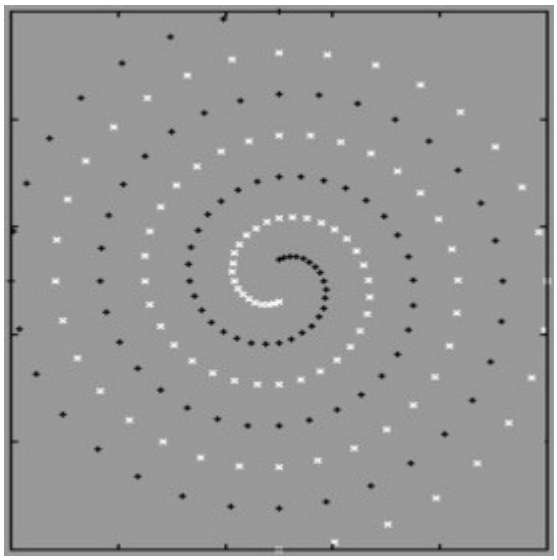
1.1 Structure of this Paper

First off, a specific data representation case will be covered. Background for this case will be given, followed by the experimental setup, and concluded with the results. Then there is a section covering findings on network representation. Again, first off there will be background information about this case, followed by the experimental setup and concluded with results.

2 Data Representation

When proposing input-output pairs of data to be learnt by an ANN, it is feasible to state those relations in terms which the network can comprehend. ANNs can find simple relations easily, but as they become more difficult, networks have increased trouble learning the data as a whole. Not only can the number of variables become more burdensome, which means that more and more weights need to be changed for each additional input neuron, but the increased difficulty can also lie in the number of abstractions that need to be made. Margolis [1] went into depths for the following problem space.

2.1 Background



A problem that is used often in ANN research is the Two Spirals Problem. In order to show to a human what the meaning of this problem is, we usually resort to an image. This is a picture of the Two Spirals Problem, and you can easily see what has to be learnt by the network: there are two spirals, a black one and a white one, which interlock. The question is what colour an arbitrary point on each of those spirals has. Or worse, because the input values for ANNs are real-valued: what is the colour of some random (x,y) point? Traditionally, the input-output pairs are given with three values, the first two being the x and y values, the third being either one or minus one, denoting either black or white. It should be noted that the invisible axes of this graph run through the centre of the picture, such that the inner dot of the black spiral is at (0,1), for example. (Coordinates can be multiplied by any factor while retaining the relation.) The white dot would then be at (0,-1).

The reason why this problem is so hard to learn for ANNs is because networks can't easily learn curves. They only learn linear relations between variables, which in this case means straight lines. The networks needs many lines to describe this relation, as can be clearly seen from this picture. So is this the best approach to state intrinsically difficult problems?

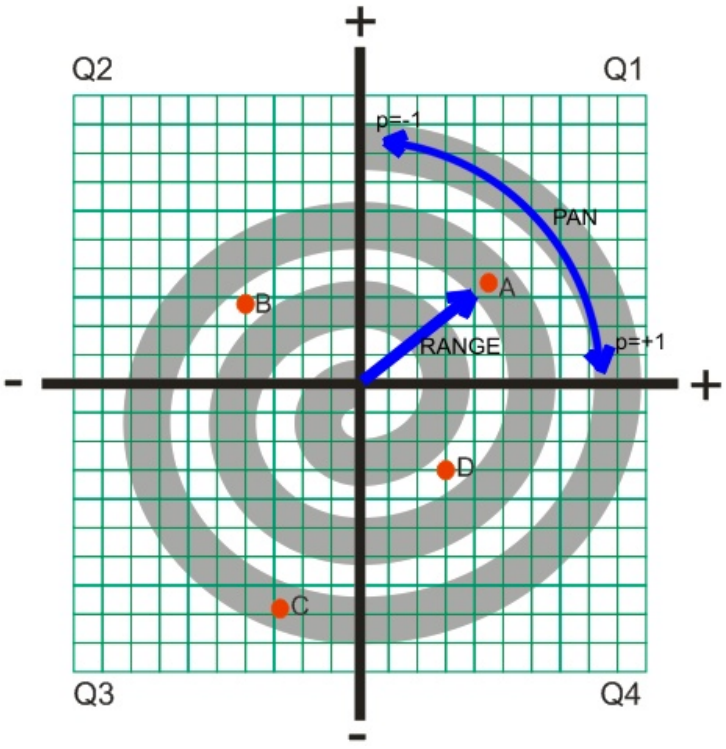
The traditional representation feels more like a game of Battleship, where one person calls two values and the other person calls either 'hit' or 'miss'. And as this picture is composed of 194

points, even a human being could not learn these 194 examples by heart and conclude by saying 'ahh, you mean two interlocking spirals!' without having seen the picture. It is not required to mirror machine intelligence to human intelligence, but there is something intrinsically difficult in this representation that should or should not be asked to be solved by a small network with at most 20 hidden neurons.

But there is more to this representation than philosophical questions. The ever-expanding series of coordinates never stays within the domain of [-1,1], whereas ANNs are mathematically tuned to processing values between those boundaries. Traditional representations cold-heartedly feed values of minus 6 into the input layer neurons. It is found customary to adjust all input values to the range [-1,1].

2.2 Methodology

There are some adjustments that can be made to this representation to fit it into a relation that a human can readily understand. And if a human can easily understand the relation without having to think, it is assumed that a network can learn the data too. So the outcome space is divided into four sectors, being [positive,positive], [negative,positive] and so on. This accounts for four input values, each having the value called Range in the picture for the sector that the value is in, and zero for the other three inputs. The Range value is of course the distance from the centre, essentially denoting which curve it's at.
As a trivial side-note, all input values were scaled down to fit into the [-1,1] interval.



The Step-Size Problem
The learning rate of the training algorithm determines how fast weights of the network change according to a correction value. If the learning rate is high, a correction will more easily lead to a correction in the neighbouring values as well, whereas if the learning rate is very low, it will take way too many training steps to slowly adjust the weights to their proper values.

Now if we leave the output values at their extremes, the learning rate has to be extremely low to learn the distinction between, for example, 0.00037 for white and 0.00039 for black. And if the learning rate would be larger, it would over-step the other value and change the neighbouring value for the wrong. So we really need to introduce some uncertainty into the output values. For this, the Pan value is used. Margolis claims that there is more uncertainty around the axes than around the 45 degree angles. That is where I disagree, as I think that there is uncertainty depending on the Range value, so the network should have to learn how wide each curve is and then modulate that over the Range value to obtain how certain a point is some colour. Nevertheless I followed the advice to take the Pan value into consideration. This value ranges from minus one to one depending on the distance to the axes. The absolute value of this Pan is added or subtracted from the output value to lean it towards zero, which means uncertainty in ANNs.

2.3 Experimental Setup

The Framework to compute and train ANNs is Encog [2]. For this experiment, it was only needed to choose a training algorithm and to load the correct dataset. What I wanted to find out was the difference in performance of each training set. The traditional set, containing values outside the [-1,1] range and giving no hint as of the structure of the data, was expected to be harder to learn both in time needed and in error rate itself. The improved dataset should however train very fast, require smaller networks and show less error after training than the original dataset.

Encog provides an rProp (Resilient Propagation) algorithm for training networks. Parameters for this training method are customizable but for standard runs, they are picked automatically. It was not in my interest to know the parameters, I was only interested in the training speed and training performance of the algorithm on each data set.

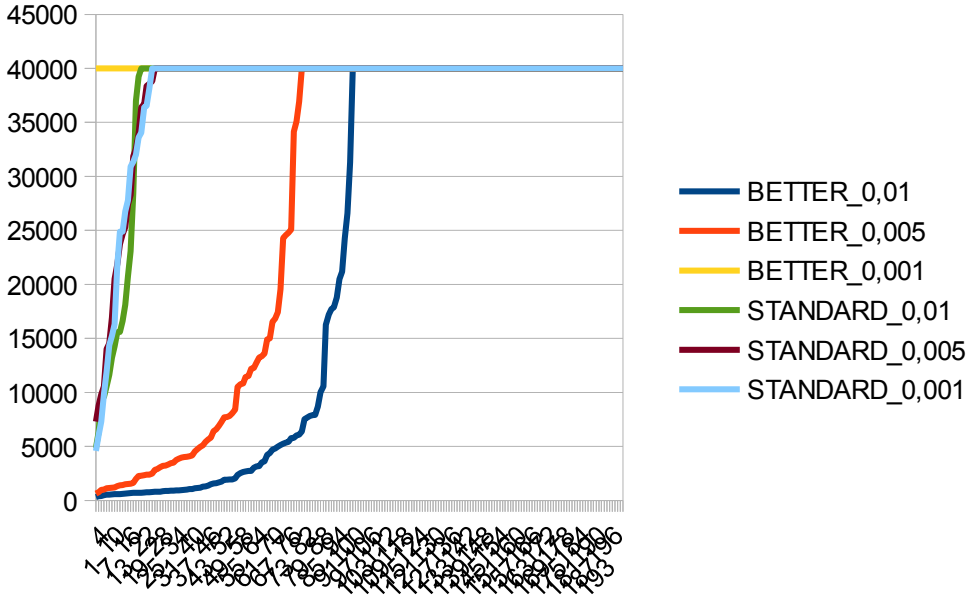
I ran several tests, of which two were of main interest. The first test ran the training for at most 40,000 training steps, after which the fitness, or error rate, of the data set was recorded. The fitness was computed using the MSE (Mean Squared Error) algorithm, which is the standard for the Encog Framework. A low fitness is preferable, as a low MSE means little error on average.

I used many networks of different size for training, preserving the four input neurons but varying the number of hidden neurons and number of hidden layers from zero to five neurons in the second layer, and five to fifteen neurons in the first layer. It was not my main interest to compare the topologies, but rather to give each topology it's own share of the testing benchmark and evaluate the error levels of each and every topology.

The second test would see how well a network could be trained in virtually infinite time. It was expected that the improved dataset would have a better time abstracting away over the problem and should thus converge to an error level of zero, in contrast to the original dataset which should have more trouble training for faultless fitness, especially for the smaller networks. Again, networks of varying topology were used to train and all runs were treated evenly, so that smaller networks were equally important than bigger networks. It was of only interest to see the difference in fitness convergence.

2.4 Results

The First Experiment



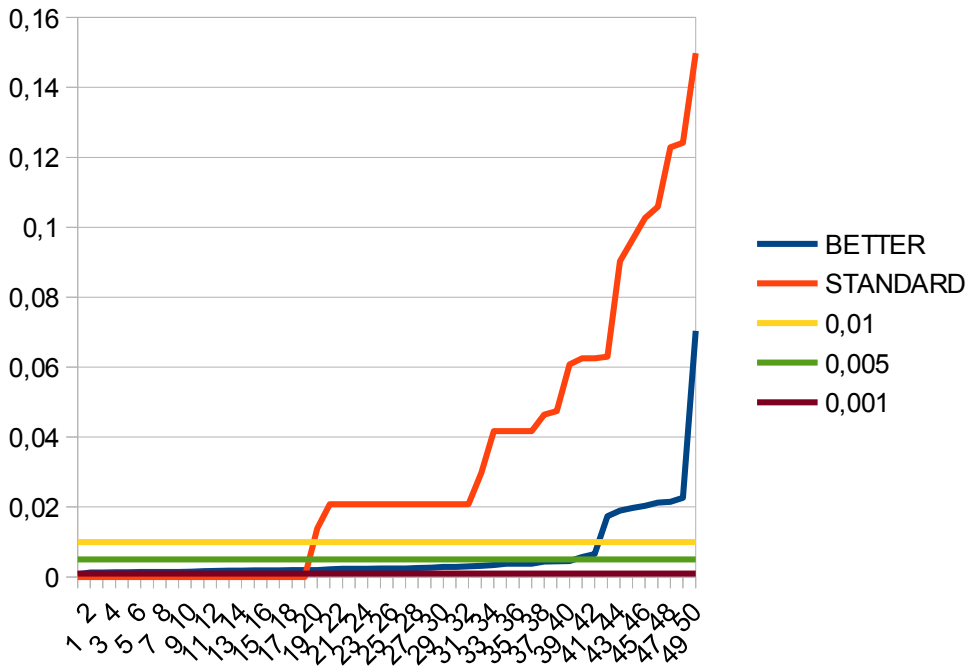
This graph shows the results of running multiple copies of networks of different topologies on the two different datasets until they reached the target fitness or until the 40,000 runs limit was reached. On the horizontal axis each networks is depicted. They are sorted by the number of runs needed, so it is possible for a small network to come before a large network if by chance its fitness was better than the other network. In practise, larger networks have better fitness because of their increased means of abstraction, but the difference between topology is uninteresting in this case.

For a first quick look, compare the dark blue line to the green line. The dark blue line is more to the right, meaning more networks reached the target fitness level of, in this case, 0.01, meaning one in hundred inputs evaluated wrongly. (Note that the Dutch way of denoting fractions is preserved in the picture.) The green line shows that only very few networks learnt until the target fitness level before the maximum number of runs was reached. Clearly, the improved dataset, namely BETTER or concretely the blue line, performs way better than the original dataset.

Now let's look into the other two pairs. An error level of 0.005 is also easily reached by the improved dataset, contrary to the original dataset which has much trouble training until the target fitness. Probably only the largest networks reach the target in the original dataset case, but this information is not preserved in the graph. More interesting is the yellow line. This line belongs to the 0.001 fitness target. No single network topology trains until that level when confronted with the improved dataset. On the contrary, the original dataset can be learnt up to that level in at least some cases.

The Second Experiment

For this case, virtually infinite training steps were allowed. Three indication lines are added, denoting the three error levels that could be reached. Again, the networks are sorted, this time by their fitness value, so the topology of each networks is not preserved in this graph.



Let's first focus on the orange line. This line depicts the fitness level of all networks trained on the original dataset. The first 19 networks could be trained until error level zero, while the next networks all perform worse than the most easily reachable target of 0.01.

As can clearly be seen, the improved dataset, denoted by the blue line named BETTER, only fails to train until the target of 0.01 for 7 or 8 networks. The other networks all perform good enough for that goal. The middle goal of 0.005 error-level is passed by all but only ten networks. So the improved dataset is clearly more easily learnable by all but some networks. But although this result looks very promising, what this graph cannot show is that the improved dataset can never be learnt until error-level zero. So that means that, however long any network is trained, it will never really understand the problem. But it should be noted that this is a dataset that involves uncertainty. My first intuition was that this inability to perform faultlessly is a flaw of the dataset. I initially thought that it was completely trivial to say which colour each point would have, given even that each point lies on a spiral per-se. On the other hand, the uncertainty that was put into the dataset might account for the inability to reach the lowest error level. And how bad is it to have less than one-in-thousand questions wrong?

2.5 Conclusions and Insights

The results clearly show superiority of the improved dataset over the original. It is thus important to present data in a manner that is easily understandable for an ANN. But does this defeat the purpose of the dataset, namely being difficult to learn? I think it does. I think this specific problem is formulated wrongly to be regarded as a difficult problem. The real difficult problems are way more abstract in nature. This though can lead to different conclusions. Firstly I thought that we should focus on neural constructions that can solve primitive tasks fast and efficiently. We should then combine the parts into more higher-level constructions that can deal with more difficult problems. I presume that is what happened in the evolutions of the natural brain, too. On the other hand, such problems as stated in the original formulation, more or less like the game of Battleship, are highly valuable for evolving networks that can pre-process data themselves, just like we humans just did while optimizing the dataset for use by more primitive networks. This, however, means that we should most probably evolve topologies that are not pre-set in size. Neither should they be smaller than a set size we would assume. In such cases, the impossibly stated formulation of this problem helps create an abstraction that might well be reused in other problems, which is one of the main goals of research in Artificial Intelligence.

So should we use the original or the advanced data representation when comparing network trainers or network recombination operators (a topic following this conclusion)? I strongly lean towards the improved dataset, because I feel that the original dataset is just not useful because of its too low-level formulation. Moreover, should we ever want to explore modular neural network implementations, then optimization will mainly be done in each module, where the problems is broken down into smaller parts, meaning it will have to deal with only little abstraction per module. Those are my interpretations concerning the choice of the dataset representation.

3 Network Representation

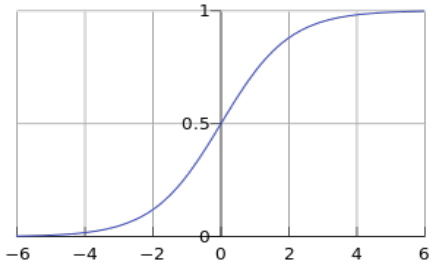
Genetic Algorithms (GA's) [6], [7] are used to optimize individual solutions in order to find a solution that explains a problem well enough. With this strategy, populations of individuals are kept, which are merged with each other while trying to find a combination that works best. In such algorithms, it is very important that recombining 'parent' solutions into 'child' solutions preserves the most information possible. It is desirable that children look like their parents, while otherwise children would be more 'random', which would result in a more randomized search.

3.1 Background

GA's rely on passing useful information from one generation to the next, just like what happens in real-life evolution: as the fittest individuals survive, their children will possess the same useful qualities their parents had with high probability. This makes the children equally fit to the environment as their parents. The same goes for GA's in Computer Science. However, the recombination operator that combines information from both parents into (both) children, will have to be coded by a programmer. The difficulty lies in the nature of ANNs: the complex relation between each weight between each neuron cannot be understood on a per-case basis. The network can only be evaluated as a whole. So the question how each part of each parent should be used in the children cannot be clearly understood.

ANNs themselves are naturally represented in one kind of structure. Normally they are thought of as special nodes connected with vertices storing the weights. This representation does not model the mathematical properties of ANNs all too well. There are two main (and easily reducible) redundancies in the representation. The rest of this section will deal with a solution proposed by (Thierens, 1996) [3].

Neural Networks have nodes connected to other nodes, usually presented in layers. In order to calculate the response of a neuron, it will calculate the sum of all incoming signals, transform that sum using the sigmoid function, and pass that value to each of its successor nodes. The first redundancy is found at the summation: when calculating the sum of all previous nodes, the order of those nodes is irrelevant. Mathematically there is no difference in the calculated result of the network if the order of the nodes in one layer is shuffled.



Another redundancy has to do with the logistic function used to squash the summed value between the interval [-1,1]. For the computed result, it is not important if the sum of the values is negated (by negating all incoming values, the sum is also negated), fed into the sigmoid function, and then negated again. There are now two configurations that compute the same result, so only one of those has to be preserved. Transforming the weights in such a way that only one fixed neuron has a positive weight, this redundancy is eliminated. In this case, the bias node is chosen to have positive weights. In order to get rid of the differences in order of neurons in one layer, again the bias node is taken as sorting token. So the weights of the bias node, that only feeds one weight to each node in the next layer, are sorted.

ANNs can be coded in different ways. The representation that is chosen here is one where all weights are jammed in one array. The length of the array is fixed, and the network has a fixed topology. Encog codes the weight array from the output layer to the input layer, listing all weights that arrive at one neuron, continuing to the next node until the layer is done and then skipping to the next layer, performing the same list-all-incoming-weights algorithm. Encog keeps certain indices in order to navigate through the weight array quickly.

3.2 Methodology

In order to find the effect of transforming one representation into another, it is sufficient to look at the correlation coefficient. This value expresses the amount of covariance between two datasets. In this case, to compute the correlation coefficient, you first calculate the covariance between the two sets, and divide that figure by the product of the individual variances. This value then gives an indication of how the second set relates to the first.

In this case we wanted to see how much information was passed onto the children, so to compute this in a figure, first a large number of parent-couples was generated. Each couple then created two offspring individuals. In order to calculate the coefficient, the mean of the fitness of each couple was taken. So there we had many mean-fitness values for both parents and children, and the coefficient would then say the correlation between the sets. A correlation of 1 would mean that good parents get good children, and bad parents always get bad children. That is the most favourable case, as the maximum amount of information is passed from the parents to the children. The other case, minus one, would mean that good parents always get bad children and visa versa. This is also very interesting, but unwanted in this setting. A value of zero means no correlation, so we effectively end up with random search.

Initially there are many possible parameters that we can vary. For example, Network Size plays a role in fitness evaluation. As we have seen in the first part of this paper, some networks will never learn the dataset, as other networks become lucky to learn it while large networks have a higher chance to eventually understand the problem. I started out testing two network topologies, one having 15 hidden neurons and one having an additional hidden layer containing 5 hidden neurons.

Another parameter is what I call the Initializer, which takes a network and trains it before it is recombined with another parent. I proposed several settings, ranging from zero pre-training to test purely the crossover operator (and pre-crossover operator coming next), to training for 200 steps before crossover.

Third, the Crossover itself can be varied. Thierens proposes to only crossover the input weights of a layer, which means that a child gets the input weights from one parent while retaining the output weights from the other parent. Moreover, after the crosspoint, all information of one parent is received. This is contrary to my own idea that a neuron as a whole is crossed, so inside the cross section, both input and output weights are passed on. The big distinction is of course the amount of information mixing. Eventually I tried both approaches.

Fourth, the way to change the representation can be varied. There are several ways to do this. First off, there are two operations to be done: rearrange the neurons in one layer (sorting) and flip the weights so the bias weights are all positive. I tried performing both operations and I tried performing only the sorting, as the flipping of the weights showed to be of great influence on the quality of the children.

The second way to change the representation is at the level of the sorting. Traditionally, the neurons are sorted according to the lonely weight of the bias neuron. As is obvious from a mathematical perspective, there need only be an arbitrary sorting of neurons to eliminate 'shuffled' configurations. Although this should be sufficient, I would like to propose two alternative ways to sort the neurons: in order to preserve information regarding the sum of all incoming weights, it might be worthwhile to sort the neurons according to this sum. Then, by crossing neurons, whether in total or only a part of their weights, neurons with few incoming weights are exchanged by neurons with also few incoming weights, and so on. This change may greatly increase the efficiency of the crossover operator, as similar information is exchanged.

It is also possible to base the sorting of the neurons on the spread of incoming weights. Then (simply) the variance of the weights has to be calculated in order to find the correct order in which to sort the nodes. As is obvious in ANNs, it is unclear whether this parameter has any effect on the efficiency of the crossover operator, but nevertheless, when crossing on end of the nodes with the same end of another parent, nodes with a similar spread of weights will be exchanged. It might well be that critical nodes which receive a wide range of incoming data, will be exchanged while nodes that simply pass on information will be exchanged with other nodes that perform a similar job.

This method can be expanded to the outgoing neurons, too. Another factor then comes into play, namely the bias towards input or output neurons. Do they weigh evenly, or is input variance more important than output variance? Or should we combine both notions to sort by some combination of strength of weights and spread between those weights? I have not incorporated these variables into my research, but this still is very interesting. This is definitely something for future research.

Then we come to the variable named Fitness Evaluation. MSE (Mean-Square-Error) or SSE (Sum-Square-Error) can be used, where Encog uses MSE and other (older) research [3] used SSE. But there is more to this case. What about networks that are not able to learn this problem very well, but are able to learn any problem fast? Then the fitness function would have to train a network for a fixed amount of steps, and obtain the MSE fitness of the network after the training. It is even possible to include the number of steps needed to learn until a fixed fitness threshold. Moreover, it is also possible to retain the trained network for future recombinations, thus effectively merging GA search with training. This is tightly couples to the Baldwin Effect [4] which states that learning a skill will slowly develop into genetic information. That means that it is important to allow networks to train next to evolving, in order to favour abstract 'learning' to just learning this very problem.

Important also is the choice for the data set. As seen in the previous part, the data needs to be suited to the network architecture. We do not want to present data to networks that they will never be able to learn. What then does the value of the fitness mean, if the children have the same clue about the problem as their parents. Not much.

We should also pay attention to training data vs. evaluation data. This is not needed if a network is only evaluated and not trained, because then the training data can function as the evaluation data, but if the network is trained and then evaluated, the researcher should pay close attention to use

the evaluation data in the fitness function as otherwise the network might over-fit and does not abstract over the data, instead only learning the training set by heart. This is equally the case for untraining networks, but then the researcher should split up the data and test on each part separately, which makes each data set smaller. So this is a difficult situation. It might also be a favour for training networks while performing GA.

Finally, the way the parent fitnesses are combined into one number can be changed. I have always used the mean of both values, but it is also possible to take the highest of the parents and compare that value to the best child. Or choose the worst of both parents and children. Choosing the best parent and child may even be better, as it is of lesser importance that both children be good, than it is that one child is very good.

Many different test have been run in order to prove the superiority of the changed representation over the original one. A large number of parents were generated randomly using Nguyen & Widrow's algorithm[5], coupled, and copied for use in both tests. All tests were two-fold, the first calculating the correlation for the original representation, the second for the changed representation. The way the representation was changed could vary, but the comparison was always between changed vs. not changed.

3.3 Results and Adaptations

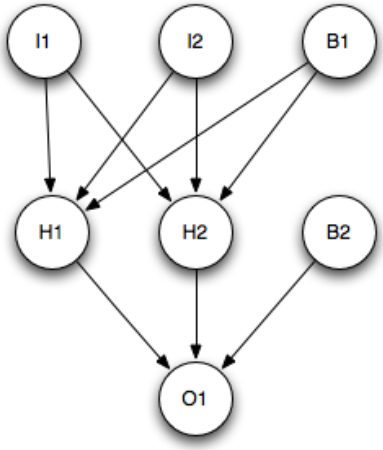
Early tests used only one network configuration of 15 hidden neurons, had a void initializer (so only random parents were crossed), used a pre-crossover that both sorted the neurons and flipped the weight signs, and calculated the fitness based on MSE. The original Two Spirals Dataset was used and the fitness was based on all 194 data entries in this set.

In this case, the correlation of the unchanged representation floated around 0.92, contrary to the expected value of around 0.5 found elsewhere. On the other hand, the correlation of the changed representation was very bad, 0.022, almost zero!

So there were two possibilities that could possibly coincide, but one was to lower the correlation of the unchanged representation, the second being to increase the changed one. Decreasing the high correlation was very difficult to almost impossible. The only way to lower this value proved to be to change the crossover point. Changing the amount of information that was actually crossed changed the coefficient from 0.89 to 0.94, so when little information was crossed then the coefficient was even higher, and lower when more information was crossed. That was more or less expected as merging parents always destroys information.

Changing the fitness evaluation from MSE to SSE had no noticeable impact. Neither did the network configuration change any value. Changing the initializer so it pre-trains the networks before crossing over, the values changed considerably, but so did the values for the changed representation, for the worse. After training, more information is lost when merging parents than when merging untrained parents. But this was still not the expected result. I did not change the dataset, so the solution had to be in the crossover operator.

3.4 Crossover Operator



The Crossover Operator takes two parents and splits their genetic information at one or several points in order to merge them into two children. As in Encog the network is stored as one large list of weights, and also in a way in which each neuron stores its incoming weights adjacently, a special function is needed to find the proper indices in this list storing the weights that need to be crossed. These indices are also dependent on the crossover point that is chosen. So let's say that we have a network like the network to the right. When we choose to split this network after hidden neuron H1, that means that one child will get weights from H1 to O1 and all weights from the input layer neurons to H1. The list of neurons is like this:
[H1->O1],[H2->O1],[B2->O1],[I1->H1],...,[B1->H1],[B1->H2].

One child will end up with one part of the list of one parent, and another part of the list of the other parent. When choosing a single crosspoint, for example after H1, we have two actual crosspoints: the weight from H1 to O1 is one part, while the other part of the crossover takes a consecutive part of all incoming weights for H1. When choosing two points to crossover, things get complicated fast. But considering only the input weights when crossing over, the situation becomes very easy, because then you need only copy the parent and replace one consecutive part of the other parent's incoming weights. This is more or less the setup I used, trying both cases where only the input weights were crossed and where both input and output weights were passed to the next generation.

3.5 Results Continued

As there seemed nothing more to do to downgrade the very good correlation of the unchanged representation, the only thing left to do was to upgrade the bad correlation of the changed representation. One main flaw of the pre-crossover, which actually changes one representation to the other, is the weight flip. After disabling the weight flipping, correlations reached the original correlations and surpassed those values very little. That was not surprising, as changing the order of the hidden neurons should have no noticeable effect on the network itself, contrary to actually changing weights and the more. The increase in correlation can be attributed to the reduction of equivalent representations, although by sorting the neurons on an arbitrary basis, I can hardly imagine the effects: if we assume that the chances are minimal that two equivalent representations show up as parents-to-be-coupled, then there should be minimal impact on the correlation if the neurons are only sorted on a basis that does not contribute or take into consideration any constructive aspect of the ANN. Sorting on the value of the bias neuron is not very constructive, as the value is completely random as each network is completely random.

I can therefore conclude that the introduction of the pre-crossover operator has no use in Genetic Algorithms, because the original representation passes most information on to the next generation and the pre-crossover only yields very minimal increase in suitability.

4 Future Research

It is very unexpected to find the correlation to be so high in the traditional representation. It is important to find out why this crossover operator performs so well, or why there is so much information passed on. Assuming that the crossover correlation coefficient is found to be actually lower, then it is interesting to compare the many options in choosing the hidden neuron sorting order.

More research could also be conducted in finding networks that are able to learn things fast, instead on relying on random networks that need to be trained from scratch. Using the initializer scheme to pre-train networks before they are being crossed over, and evaluating the fitness of an individual on the basis of the quality of its ability to learn data, abstract modules can be generated that can then be used to build more complex solver or recognition networks. If indeed the pre-crossover operator greatly increases the power of GA search, then these abstract modules and more high-level networks become an interesting research topic.

We could also try to get more details on the effect of data representation on the suitability of research using those data. If insensibly represented data indeed works worse in comparisons of network topologies or advanced algorithms, then we might indeed stick with the more easy problems and use those to climb to higher-level problems.

5 References

1. http://www.benmargolis.com/compsci/ai/two_spirals_problem.htm

– referencing as similar research: Jiancheng Jia; Hock-Chuan Chua, Solving two-spiral problem through input data representation. *Neural Networks, 1995. Proceedings., IEEE International Conference on (Volume:1)* (pp. 132-135).

2. http://www.heatonresearch.com/encog

3. Thierens D., Non-Redundant Genetic Coding of Neural Networks. *Proceedings of the 1996 IEEE International Conference on Evolutionary Computation-ICE'96* (pp. 571-575).

4. Depew, David J. (2003), "Baldwin Boosters, Baldwin Skeptics" in: Weber, Bruce H.; Depew, David J. (2003). *Evolution and learning: The Baldwin effect reconsidered.* Cambridge, MA: MIT Press. pp. 3–31. ISBN 0-262-23229-4.

5. Nguyen D. & Widrow B., Improving the learning speed of 2-layer neural networks by choosing initial values of the adaptive weights. *Proceedings of the International Joint Conference of Neural Networks, Vol3* 1990.

6. L. Davis (Ed.), *Genetic Algorithms and Simulated Annealing,* London: Pitman, 1987.

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

8. Nicolás García-Pedrajas, Domingo Ortiz-Boyer, César Hervás-Martínez, *An alternative approach for neural network evolution with a genetic algorithm: Crossover by combinatorial optimization,* Neural Networks, Volume 19, Issue 4, May 2006, Pages 514-528.