Do Not Disturb: Recommendations for Incremental Evolution

Nicholas Tomko and Inman Harvey
Centre for Computational Neuroscience and Robotics
Evolutionary and Adaptive Systems Group
University of Sussex, Brighton, UK
Emails: nt79@sussex.ac.uk and inmanh@sussex.ac.uk

Abstract

Incremental evolution can be used when a task is too complex to be solved in a single evolutionary step. For incremental evolution to function properly the end goal task must be broken down into easier sub-tasks that act as a set of intermediate steps en route to the final goal. When using incremental evolution to evolve neural controllers, such as artificial neural networks (ANNs), it may make sense to add additional neurons to the controller as the incremental tasks become more complicated. This paper will show that how you add these neurons makes a significant difference in the performance of evolution. More specifically we will show that adding neurons with large random weights significantly impairs the performance of evolution to the point that it is outperformed by non-incremental evolution. Therefore in most cases neurons added during evolution should be connected to the existing network with zero weights. This hypothesis was first proposed and used by Vaughan (2007) to evolve hexapod walkers but we have tried to show its validity on a more general test bed indicating that it could be used more widely.

Introduction

There are two ways evolution can be used to solve complex tasks. The first is non-incrementally where there is only a single evolutionary goal or task. A second method is incremental evolution where the final task is broken-up into more manageable sub-tasks. These sub-tasks are usually chosen in such a way so they act as evolutionary stepping stones, with the hope that the succession of easier tasks reduces the overall amount of time it takes evolution to solve the final task. When incrementally evolving neural controllers to do complex tasks it may be beneficial to add additional neural modules (brain power) to the controller at each evolutionary step. These neural modules can be connected in different ways and it will be shown in this paper that the method by which these modules are added significantly affects the performance of evolution.

When new neural modules are added during incremental evolution, the original connection weights will start the new phase of evolution with their former values. But we can consider three different ways for initialising the new connection weights (both between new nodes and old, and between the new nodes): with zero weights, with small random weights, or with large random weights. Vaughan (2007) uses the method of zero weight neural module addition to incrementally evolve walking hexapod rovers to navigate over rough terrain and avoid obstacles. Intially the rovers were evolved just to walk over flat ground. After successful flat ground walkers were evolved, the rovers were incrementally evolved to walk over rough terrain, navigate, and finally avoid obstacles. At each incremental stage new neurons were added to the rovers' neural controller with zero weights. The reason for adding the new nodes with zero weights, rather than with random weights, is that zero weight addition allowed evolution to gradually explore these new connections (Vaughan, 2007). Adding the new neurons with large random weights would have destructive influences on the overall network, negatively impacting the already evolved behaviours (Vaughan, 2007). Therefore our main hypothesis is that adding new neural modules with large random weights during incremental evolution will seriously hamper the performance of evolution. A related secondary hypothesis of this paper is that the best method of evolution is incremental evolution with the addition of new neural modules with zero weights. Intuitively both these hypotheses make sense but we know of no previous principled investigation of these methods of incremental evolution. We will therefore examine evolution using a test-bed in which the complexity of the task is easily scalable and one which generates reproducible results allowing this hypothesis to be generalized beyond the agent-environment domain in Vaughan (2007).

The rest of this paper is structured as follows: The background section is a literature survey of related incremental evolutionary research. The methods section describes the test-bed, the genetic algorithm, and how incremental evolution was structured. The results from our simulations are then presented and analyzed in the discussion section. The results confirm for this test-bed the hypothesis that adding neural modules with large random weights negatively impacts the performance of evolution. The results also show that for this specific test-bed incremental evolution with zero weights only outperforms non-incremental evolution under

certain conditions.

Background

Incremental Evolution in the Natural World

Evolution in the natural world has been incremental. It is only by building on previously evolved structures that the diversity and complexity of life has been able to increase over time. One of the most striking examples of incremental evolution in the natural world is that of the eye. Historically, many criticisms of evolutionary theory have tried to use the eye as an example of why evolutionary theory is incorrect. Many of these criticisms are based on an incomplete reading of the following quote from Darwin (1872).

> To suppose that the eye, with all its inimitable contrivances for adjusting the focus to different distances, for admitting different amounts of light, and for the correction of spherical and chromatic aberration, could have been formed by natural selection, seems, I freely confess, absurd in the highest possible degree...
>
> ...Yet reason tells me, that if numerous gradations from a perfect and complex eye to one very imperfect and simple, each grade being useful to its possessor, can be shown to exist; if further, the eye does vary ever so slightly, and the variations be inherited, which is certainly the case; and if any variation or modification in the organ be ever useful to an animal under changing conditions of life then the difficulty of believing that a perfect and complex eye could be formed by natural selection, though insuperable by our imagination, can hardly be considered real. (Darwin, 1872, pg. 143-144)

By reading only the first part of this passage, one could be led to believe that Darwin strongly doubted that something as complex as the eye could have been evolved. But based on the second part of this passage this obviously was not the case. Darwin did not believe that the eye could have been evolved in a single step but he did believe that there was a way for the eye to be evolved incrementally. Darwin understood the power of incremental evolution and how it could create something that was too complex to be evolved in a single step. Some critics have tried to claim that incrementally evolving an eye is impossible because half an eye is pretty much useless. This criticism has been proven wrong by the fact fossil records have revealed many primitive visual structures that could have been intermediate evolutionary steps en route to the modern eyes we see today. In the book *In the Blink of an Eye*, Parker (2004) describes a variety of basic eye structures such as light sensitive spots that are likely part of the incremental evolutionary history of the eye. Nilsson and Pelger (1994) present a mathematical estimate of the number of generations it would take a light-sensitive patch to gradually evolve into a focused lens eye through small design improvements. They show that even using very pessimistic assumptions the transition from light sensitive spot to lens-based eye may only have taken on the order of a few hundred thousand years, which is very short in terms of evolutionary timescales.

Artificial Incremental Evolution

Artificial incremental evolution is inspired by the power of natural evolution to produce diverse and complex biological forms. Artificial incremental evolution takes a complex task and breaks it up into evolutionary manageable steps so that finding the solution to the complex task becomes easier. Artificial incremental evolution has been implemented in many different ways. Mouret and Doncieux (2008) divide some of the more common incremental evolutionary techniques into the following four categories: **staged evolution, environmental complexification, behavioural decomposition, and fitness shaping.**

In **staged evolution** there are multiple fitness functions corresponding to multiple sub-tasks. Initially the population of individuals is evolved to complete the first sub-task. When a successful solution to the first sub-task has been evolved the fitness function is changed to the second sub-task's fitness function. **Environmental complexification** is similar to staged evolution but instead of having discrete changes in task complexity, the complexity of the task can be changed continuously by tuning certain parameters (Harvey et al. (1994) is an example). **Behavioural decomposition** or **modular evolution** sub-divides the neural controller into separate task-based sub-controllers. Each one of these sub-controllers are evolved separately to do a specific task and then a different evolutionary algorithm is used to combine these controllers into the master neurocontroller (see Larsen and Hansen (2005) for an example). **Fitness shaping** is similar to **behavioural decomposition** but uses a weighted sum of multiple evaluation criteria in order to create a fitness gradient that evolution tries to follow. Mouret and Doncieux (2008) believe the main shortcoming of staged evolution is that the programmer has to determine the sub-task fitness functions and also when to switch sub-tasks. In staged evolution there is also the issue of whether there is a fitness tradeoff between different sub-tasks. In other words, does being proficient at the second sub-task reduce the agent's proficiency at doing the first sub-task? To overcome these issues Mouret and Doncieux (2008) propose a method called multi-objective optimization where the agents are evaluated simultaneously against all the sub-task and main goal-task fitness functions.

The importance of how the sub-tasks are chosen is demonstrated in Tuci et al. (2002) who show how a high level, non-reactive learning behaviour can be incrementally evolved from low level, reactive behaviours. They evolved a simulated robot controlled using a continuous time recurrent neural network (CTRNN) with fixed weights to learn the relationship between a light and a target. This experiment was

based on a 2-D associative learning task in Yamauchi and Beer (1994). Unlike Yamauchi and Beer (1994), Tuci et al. (2002) were able to find an integrated solution to this task without using behavioural decomposition where the neuro-controller had to be modularized. It was found that only after the agents first learned to pay attention to the light were they able to learn the relationship between the target and light. In other words, the agents first needed to learn a very basic reactive behaviour before they were able to learn a higher-level, non-reactive behaviour. The important lesson from Tuci et al. (2002) related to this paper is that it demonstrates the importance of selecting the incremental sub-tasks with regards to evolutionary success.

Adding Neural Layers During Incremental Evolution

The four types of incremental evolution in the previous section describe how the fitness goal can be incrementalized but not how new neural modules are added during evolution. Togelius (2004) classifies incremental evolution not only based on how the fitness function is broken up but also on how new neural layers are added to the controller. Togelius (2004) defines **incremental evolution** as the evolution of a one layer network using multiple fitness functions, **modularised evolution** as the evolution of multiple layers or multiple networks with a single fitness function and **layered evolution** as the evolution of a multi-layered network using multiple fitness functions. In **layered evolution** Togelius (2004) adds the neural layers to mimic Brooks (1991) subsumption architecture. At the beginning of each evolutionary stage a new layer is added to the network and all previously added, lower layers have their weights frozen; ie. once a layer has been evolved its weights are fixed and are not changed during subsequent stages. Togelius (2004) tested these different types of incremental evolution on a simulated phototaxis and obstacle avoidance task and determined that layered evolution was faster and more reliable than the other methods of evolution.

Unanswered Questions

In the incremental evolution literature reviewed for this paper, details on how new neural modules were added during incremental evolution were lacking. Based on the work done in Vaughan (2007) on evolving walking robots, it seems that how neurons are added can significantly change the performance of incremental evolution and is therefore an important consideration when setting up artificial evolution. Other than Vaughan (2007), one study that did address the question of adding additional neurons during evolution was Stanley and Miikkulainen (2002) NEAT (Neuroevolution of Augmenting Topologies) algorithm. This evolutionary algorithm modifies both the network weights and its topology, so shares some similarities with the incremental evolution described in Togelius (2004). In NEAT there are two different mutation operators: add-connection and add-node. In add-connection a single new connection is added between two existing nodes and this new connection is initialized with a random weight. In add-node a new node is added in between two existing nodes. The input weight to this node is given a value of one while the output weight from the new node is given the same weights as the connection that was just broken. The reason for this is to ensure that the addition of the new node is a neutral mutation and will not initially affect the behaviour of the network. For indroductory material to the concept of neutral networks see: Huynen et al. (1996); Forst et al. (1995); Harvey and Thompson (1996); Barnett (2001). What is interesting about the NEAT mutation operators is that nodes are added neutrally but connections are not.

The importance of neutrality in the evolution of neural networks is also highlighted in work on center-crossing initialization of ANNs (Mathayomchan and Beer, 2002). Here they show that when applying evolutionary search to ANNs the frequency and speed of evolution can be improved by seeding the initial evolutionary population with center crossing networks rather than with random initial populations. This is most likely due to the fact that there is a wider range of network dynamics accessible from a population of center-crossing networks (Mathayomchan and Beer, 2002).

Methods

The goal of this project was to try to compare the performance of different methods of incremental evolution and non-incremental evolution in a more general context than the example in Vaughan (2007). In order to do this the test-bed used needed to be easily modifiable, simple to understand and not tied to a specific domain. Since the effects under investigation will depend strongly on the dimensionality of the fitness landscape and of any neutral networks, the framework used must allow for changes in dimensionality as new tasks are added. Because of these constraints a simple feed-forward autoencoding network was chosen. The autoencoder is easy to understand and analyze, and the complexity of autoencoding tasks can be easily increased by just changing the number of neurons in the network (see the next section for details). This means a given autoencoding task can either be solved using non-incremental evolution or using incremental evolution by adding simpler autoencoding tasks together thus allowing us to test the different methods of adding new neural modules.

In this section the autoencoding task, the genetic algorithm and the method of incremental evolution will be described. All simulations were programmed and run using Matlab.

The Autoencoder Network

An autoencoder is a feedforward ANN that reduces high dimensional input data into a lower dimensional code and then

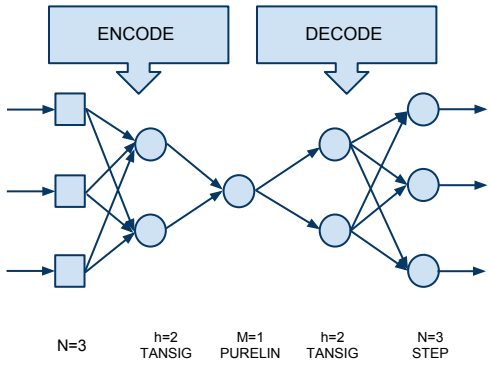

Figure 1: A 3-2-1-2-3 Autoencoder

recovers the original input from this code (see Hinton and Salakhutdinov (2006) for a good description). An autoencoder can be thought of having two halves; the encoding half which encodes the input data into a low dimensional code and the decoding half which decodes the low dimensional code back to the original inputs.

The autoencoders used in this study are of the form $N-h-M-h-N$. Where $N =$ the number of inputs = the number of outputs, $h =$ the size of the hidden layers, and $M =$ size of bottleneck layer (see figure 1). By changing N, h, and M the dimensionality of the autoencoder can be modified which corresponds to changing the difficulty of the autoencoding task. For a given value of N, the difficulty of the task is increased when either M, the size of the bottleneck, or h, the size of the hidden layer, is decreased.

All of the networks used were structured in the same way. The networks were fully connected and feedforward and all inputs to the networks were either 1 or -1. The hidden layer transfer function was hyperbolic tangential (Matlab tansig function) and the bottleneck layer transfer function was linear (Matlab purelin function). The output layer transfer function was a discrete step function. This means that any positive outputs were mapped to +1 and any negative outputs were mapped to -1. For simplicity, no biases were used in any of the networks.

Evolution

The Genetic Algorithm The genetic algorithm (GA) used in this simulation was the steady-state microbial GA with the recombination rate set to zero (Harvey, 2009). This is equivalent to a simple steady-state GA that uses tournament selection where the loser of each tournament is mutated and the winner remains unchanged. The number of generation equivalents in the microbial GA is related to the number of tournaments by: $generations = tournaments/popsize$. All results are presented as the number of tournaments it takes evolution to do a task rather than the number of generations.

The artificial genotype encoded the real valued autoencoder weights; these weights were limited to the range [+10,-10]. All of the experiments were done with a population size of 50. The population was initialized by generating a random, normally distributed string of numbers with mean = 0 and standard deviation set equal to the standard deviation of the mutation operator (see below).

Simulations were run using both single and multi-allele (multi-gene) mutation. In multi-allele mutation all alleles in the tournament loser's genotype were mutated a small amount. This small amount was a normally distributed random number with mean = 0 and standard deviation = 0.1. In single-allele mutation, only a single allele in the loser's genome was mutated each tournament. For single allele mutation the mutation rate was a normally distributed random number with mean = 0 and standard deviation = 0.5.

The Fitness Function The fitness of a given autoencoder was calculated as the total number of outputs that matched the inputs, for all possible 2^Ninput strings. Therefore for an autoencoder with N inputs the maximum fitness score it could get was $2^N * N$.

Incremental Evolution The method of incremental evolution used in this experiment does not fit perfectly into one of the types described in the background section, but is similar to layered evolution in Togelius (2004). The main difference is that after the new neurons are added the entire network is evolved with the new fitness function. In Togelius (2004) when a new neuronal layer was added only the new neurons were evolved, the weights of the existing neurons in the network remained fixed.

A given autoencoder can be incrementally evolved in n sub-tasks by splitting up the $N-h-M-h-N$ autoencoder into n smaller autoencoders. The difficulty of a given subtask depends on the size of Nn, hn, or Mn.

$$[N-h-M-h-N] = [N1-h1-M1-h1-N1]+ [N2-h2-M2-h2-N2]+\ldots+[Nn-hn-Mn-hn-Nn]$$

When incrementally evolving an autoencoder it must be noted that adding an additional module is the same as changing the task. This is because every time the number of inputs N is increased, there is a step change in the difficulty of the autoencoding task. So unlike the agent-environment incremental evolutionary simulations described in the background section where the addition of neural modules and choice of incremental task can be independent, in the autoencoder the addition of a neural module and the changing of a task are equivalent.

Each stage of incremental evolution is run until an autoencoder is found that can perfectly solve the current task. At this point evolution is paused and the perfect individual's genotype is replicated and replaces the entire current population as the seed population for the next stage of evolution. The additional neurons and corresponding weights of

Table 1: Incremental stages for tested autoencoder tasks

Overall Task	Stage 1	Stage 2	Stage 3
2-2-2-2-2	1-1-1-1-1	1-1-1-1-1	
3-3-3-3-3	1-1-1-1-1	1-1-1-1-1	1-1-1-1-1
3-36-2-36-3	2-2-1-2-2	1-34-1-34-1	
3-24-2-15-3	2-2-1-2-2	1-22-1-22-1	
3-12-2-12-3	2-2-1-2-2	1-10-1-10-1	
3-6-2-6-3	2-2-1-2-2	1-4-1-4-1	
4-68-3-68-4	2-2-1-2-2	1-22-1-22-1	1-44-1-44-1

the new autoencoder are then added to this cloned population in one of three ways so we could examine which one of these methods is most effective.

1. With zero weights
2. With small random weights (normally distributed random numbers with mean zero and standard deviation of 0.1)
3. With large random weights (random numbers in the entire weight range [+10,-10])

After the new neurons and weights are added evolution is restarted on the entire autoencoder network using the new fitness function.

Choice of Tasks

In order to generally test the hypothesis, the different methods of evolution needed to be tested on tasks of varying difficulty. Table 1 lists the different autoencoders tested and the incremental evolutionary sub-tasks for each different autoencoder. For example the simplest autoencoder tested was the 2-2-2-2-2 autoencoder which was incrementally evolved in two 1-1-1-1-1 stages. This means that during the first stage of evolution the solution to a 1-1-1-1-1 autoencoder was found. For the second stage of evolution another 1-1-1-1-1 autoencoder was added to the previously evolved 1-1-1-1-1 autoencoder using one of the three methods described in the previous section. Evolution was then restarted on this new 2-2-2-2-2 autoencoder.

Results

Figure 2 shows the results of non-incremental evolution, incremental evolution with zero weights and incremental evolution with large random weights on a 3-24-2-24-3 and a 3-6-2-6-3 autoencoder using multi-allele and single-allele mutation. We chose to present the results for only these tasks for two reasons. Firstly these tasks highlight the difference in performance between autoencoders with significantly different sized hidden layers. Secondly, the data from the other tasks do not change any of the conclusions discussed in the next section and therefore we felt it unnecessary to plot these results. We have also chosen not to show the results of incremental evolution with small random weights. This is because in the case of multi-allele mutation this method of evolution was not significantly different from either non-incremental or zero weight incremental methods and in the case of single-allele mutation the results of small random incremental evolution were similar to those of the large random incremental method.

Figure 2 shows the average number of tournaments over forty runs it took to find a solution to the given tasks for each type of evolution. If a perfect solution had not been found after 400k tournaments then the run was stopped and a new run was started. The choice of stopping simulations at 400k tournaments was somewhat arbitrary but was done to ensure that multiple runs of each simulation could be completed in a reasonable amount of time. The number of runs that were successful at finding a solution to the task before 400k tournaments is displayed on each bar. For example the non-incremental, single-allele simulation of the 3-24-2-24-3 autoencoder was able to find a solution in 37/40 runs. Because all runs are stopped at 400k tournaments, the averages for simulations with a high number of unsuccessful runs are lower then if evolution was allowed to continue indefinitely. This means that the real averages of the large random incremental evolution (the worst performing method of evolution) are understated. This does not change the conclusions of this paper so we chose not to scale the averages to take into account the number of successful runs.

The error bars in figure 2 show the standard deviation of each simulation. Again due to artificially stopping the runs at 400k tournaments, simulations with a large number of unsuccessful trials have unrealistically low standard deviations.

Discussion

The main hypothesis of this paper was that when evolving neural controllers incrementally, adding neural modules with large random weights would negatively impact the performance of evolution. The results from the previous section clearly support our main hypothesis. The secondary hypothesis of this paper was that incremental evolution with zero weights would be the best method of evolution. Based on our results this was not always the case, there were cases when non-incremental evolution performed as well incremental evolution with zero weights. In this section we will first discuss the reason why adding neural modules with large random weights is not a good method of evolving neural controllers. We will then discuss the relative performance of non-incremental versus zero weight incremental evolution and propose reasons why zero weight incremental evolution is not always better than non-incremental evolution for this experimental test-bed.

As shown in figure 2 the large random incremental method is the worst method of evolution by a significant margin. Not only is this method significantly worse than the incremental evolution with zero weights it is also sig-

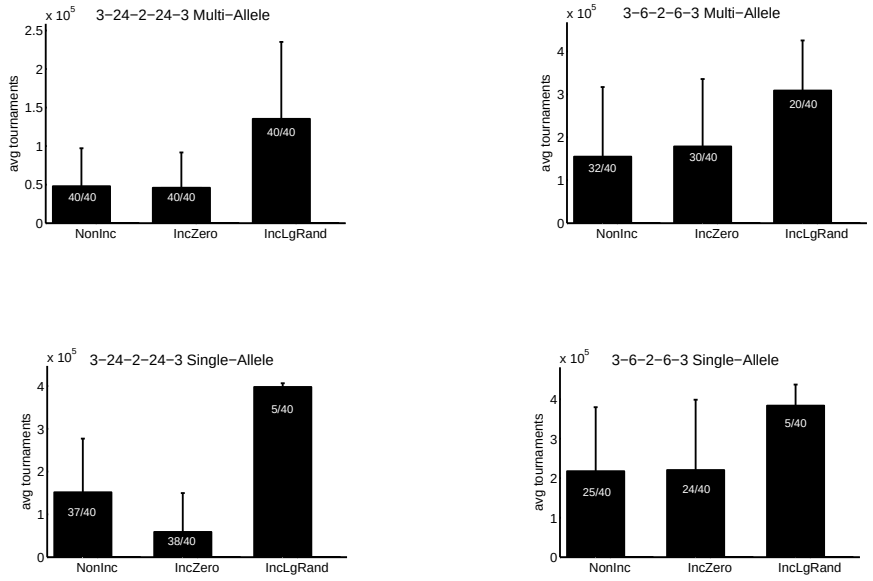

Figure 2: 3-6-2-6-3 and 3-24-2-24-3 autoencoder results for both the single (bottom) and multi-allele (top) mutation method. These results show the average number of tournaments it took to find a solution to the task over 40 runs. All runs were stopped at 400k tournaments if a solution had not been found. The number of successful runs out of 40 is shown on each bar. Error bars show standard deviation of each simulation.

nificantly worse than non-incremental evolution in all of the cases we tested. The type of mutation (single or multi-allele) used in evolution did not impact this result, large random incremental evolution was always the worst method of evolution. As discussed in Vaughan (2007) the reason for this is that adding new neurons with large random weights leads to destructive interference in the network destroying any previously evolved behaviours. When the new neurons are added with zero weights this destructive interference does not occur because the additional neurons are added neutrally. This allows evolution to slowly explore these new connections without negatively impacting what was learned in the previous stages of evolution. The results from this paper combined with the work done in Vaughan (2007) strongly suggest that anytime a neural network is evolved incrementally, any new nodes should be added with zero weights.

Based on our secondary hypothesis it was expected that zero weight incremental evolution would be the best method of evolution and always outperform non-incremental evolution; but based on our results this was not the case, sometimes non-incremental evolution outperformed incremental evolution with zero weights. We believe there are three factors that may contribute to this finding: the type of mutation used (single or multi-allele), the size of the hidden layer in the network, and the way the evolutionary sub-tasks are structured. What our results show is that when using multi-allele mutation there is no significant difference between non-incremental and incremental evolution with zero weights and when using single-allele mutation there is only a significant difference in performance when the hidden layer has many more neurons than the input and bottleneck layers.

We propose that one reason for the lack of a significant difference between zero weight incremental and non- incremental evolution when using multi-allele mutation is that multi-allele mutation causes interference in the entire network the very first time it is applied to an individual. The main benefit of adding new neurons with zero weights is that it allows evolution to gradually explore these new neural connections. When using multi-allele mutation this exploration becomes much less gradual because all of the zero weights are changed to non-zero weights the first time mutation is done. On the other hand when using single-allele mutation the new weights are changed from zero to non-zero one at a time ensuring that the effect of these new neurons on the overall network can be explored individually.

In the single-allele mutation case incremental evolution with zero weights outperformed non-incremental evolution when the autoencoder had a large number of hidden layer neurons (3-24-2-24-3, 3-36-2-36-3, 4-68-3-68-4). In fact all methods of evolution improved as the size of the hidden layer was increased; but it was incremental evolution with zero weights that showed the sharpest increase

in performance, to the point that it always outperformed non-incremental evolution when the hidden layer was big enough. This may have to do with how the size of the hidden layer corresponds to the size of the neutral network of the search space.

When there is a relatively small hidden layer this constrains the number of different ways the input information can be encoded in the bottleneck layer and then decoded in the output layer. This will cause difficulties for the zero weight incremental method and may help explain why it is not better than non-incremental evolution for these cases. However when the hidden layer is large, there is a multitude of encoding/decoding strategies - of nearly equivalent functionality - implying a large degree of neutrality in the search space. It is here the zero weight incremental with single allele mutation method can exploit this neutrality as task complexity increases and for this reason it outperforms non-incremental evolution.

Another factor that may help explain why non-incremental evolution was as good as incremental evolution with zero weights in certain cases has to do with how the incremental evolutionary stages are structured in the autoencoder tasks. As explained in the methods section the addition of a new input node corresponds to a new autoencoding task. While running the simulations it was discovered that an $N = 3$ task was significantly more difficult than an $N = 2$ task. It takes on the order of 1k tournaments to evolve a perfect solution to the $N = 2$ task and on the order of 100k tournaments to evolve a perfect solution to an $N = 3$ task. In other words, the first stage of incremental evolution is very easy compared to the subsequent evolutionary stages. This is opposite to almost all of the other incremental evolution experiments reviewed, where the difficulty of the first task is significantly higher then subsequent tasks.

For example in Vaughan (2007) he first evolved his hexapod walkers to walk on flat ground and then he evolved them to walk on rough ground, then to walk and navigate, and finally to avoid obstacles. In terms of task difficulty the evolution of the initial walking behaviour is by far the most difficult task. All subsequent tasks were just variations on the core walking behaviour. In the autoencoder simulation because the first task is so simple it has a large number of solutions. Some of these solutions may be an incremental short-cut to the solution for the subsequent task, but many solutions may not be an incremental short-cut and in this case incremental evolution would be no better then non-incremental evolution. To try to overcome this issue the first stage of all of the $N = 3$ tasks was 2-2-1-2-2 because this was the most difficult $N = 2$ task that was solvable in a reasonable amount of time. This again highlights how important task selection is with regards to the performance of incremental evolution.

One may be tempted to ask that if zero weight incremental evolution with single-allele mutation has all of the benefits mentioned above then why doesn't it perform better than the multi-allele method for equivalent tasks? Because neither of these evolutionary methods have been optimized, we believe that comparing the single-allele and multi-allele mutation results and trying to draw any conclusions from this comparison would be pointless. The mutation rate of the single-allele method was set higher than that of the multi-allele method to take into account the fact that there is less movement through the search space per mutation when using the single-allele method. But this higher mutation rate was not chosen with any rigorous testing to determine whether it was optimal. Understanding whether incremental evolution with single-allele or multi-allele mutation is better is an interesting question, but the results from our simulations cannot be used to make a claim one way or another. What our results show is that adding new nodes with large random weights during incremental evolution is detrimental to the performance of evolution and that the benefit of adding nodes with zero weights is seen when there is a large hidden layer and mutation is done one single weight at a time.

Conclusion

The main conclusion of this paper is that when incrementally evolving neural controllers, any new neural modules added during evolution should not be added to the existing network with large random weights. Adding neurons with large random weights reduces the peformance of evolution because it cause destructive interference in the network which causes previously evolved behaviours to be forgotten. Our results show that this destructive interference is so severe that incremental evolution with large random weights was by far worst method of evolution in all of our simulations (even compared to non-incremental evolution). A secondary finding was that using our test-bed, incremental evolution with the addition of new neurons with zero weights only outperformed non-incremental evolution when using the single-allele mutation method and when the hidden layer was big compared to the input and bottleneck layers. We believe this was because single-allele mutation allowed evolution to explore these new neural connections gradually one at a time. Therefore based on our results we believe that when incremental evoloving neural controllers any addition brain power added should be added neutrally with zero weights.

References

Barnett, L. (2001). Netcrawling-optimal evolutionary search with neutral networks. In *Proceedings of the 2001 Congress on Evol Comp*, volume 1. IEEE Press.

Brooks, R. (1991). Intelligence without representation. *Artificial intelligence*, 47(1-3):139–159.

Darwin, C. (1872). *The Origin of the Species*. Senate, London, 6th edition.

Forst, C. V., Reidys, C., and Weber, J. (1995). Evolutionary Dynamics and Optimization - Neutral Networks as Model-Landscapes for RNA Secondary-Structure Folding-Landscapes. In Moran, F., Moreno, A., Merelo, J., and Cachon, P., editors, *Advances in Artificial Life: Proceedings of the Third European Conference on Artificial Life (ECAL95)*, volume 929, pages 128–147. Springer Verlag.

Harvey, I. (2009). The Microbial Genetic Algorithm. In Kampis, G. E. A., editor, *Proceedings of the Tenth European Conference on Artificial Life*. Springer LNCS.

Harvey, I., Husbands, P., and Cliff, D. (1994). Seeing the light: Artificial evolution, real vision. In *From Animals to Animats 3. Proceedings of the Third International Conference on Simulation of Adaptive Behavior*, volume 1994, pages 392–401.

Harvey, I. and Thompson, A. (1996). Through the labyrinth evolution finds a way: A silicon ridge. In Higuchi, T., Iwata, M., and Weixin, L., editors, *Evolvable Systems: From Biology to Hardware, Proc. of The First International Conference on Evolvable Systems*, pages 406–422. Springer-Verlag.

Hinton, G. and Salakhutdinov, R. (2006). Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504.

Huynen, M. A., Stadler, P. F., and Fontana, W. (1996). Smoothness within ruggedness: the role of neutrality in adaptation. *Proceedings of the National Academy of Sciences of the United States of America*, 93(1):397–401.

Larsen, T. and Hansen, S. (2005). Evolving composite robot behaviour - a modular architecture. In *Proceedings of the Fifth International Workshop on Robot Motion and Control, 2005. RoMoCo '05.*, pages 271–276. Ieee.

Mathayomchan, B. and Beer, R. D. (2002). Center-crossing recurrent neural networks for the evolution of rhythmic behavior. *Neural computation*, 14(9):2043–2051.

Mouret, J. and Doncieux, S. (2008). Incremental evolution of animats' behaviors as a multi-objective optimization. *Lecture Notes in Computer Science*, 5040:210–219.

Nilsson, D. and Pelger, S. (1994). A pessimistic estimate of the time required for an eye to evolve. *Proceedings: Biological Sciences*, 256(1345):53–58.

Parker, A. (2004). *In the Blink of an Eye: How Vision Sparked the Big Bang of Evolution*. Basic Books, New York, 2nd edition.

Stanley, K. and Miikkulainen, R. (2002). Evolving neural networks through augmenting topologies. *Evolutionary Computation*, 10(2):99–127.

Togelius, J. (2004). Evolution of a subsumption architecture neurocontroller. *Journal of Intelligent and Fuzzy Systems*, 15(1):15–20.

Tuci, E., Quinn, M., and Harvey, I. (2002). An evolutionary ecological approach to the study of learning behavior using a robot-based model. *Adaptive Behavior*, 10(3-4):201–222.

Vaughan, E. D. (2007). *The Evolution of an Omni-Directional Bipedal Robot*. PhD thesis, University of Sussex.

Yamauchi, B. and Beer, R. (1994). Integrating reactive, sequential, and learning behavior using dynamical neural networks. In *From Animals to Animats 3: Proceedings of the Third International Conference on Simulation of Adaptive Behavior*, pages 382–391.