Evolving Neural Network Ensembles for Control Problems

David Pardoe
Department of Computer Sciences
The University of Texas at Austin
Austin, TX 78712
dpardoe@cs.utexas.edu

Michael Ryoo
Department of Electrical and Computer Engineering
The University of Texas at Austin
Austin, TX 78712
mryoo@mail.utexas.edu

Risto Miikkulainen
Department of Computer Sciences
The University of Texas at Austin
Austin, TX 78712
risto@cs.utexas.edu

ABSTRACT

In neuroevolution, a genetic algorithm is used to evolve a neural network to perform a particular task. The standard approach is to evolve a population over a number of generations, and then select the final generation's champion as the end result. However, it is possible that there is valuable information present in the population that is not captured by the champion. The standard approach ignores all such information. One possible solution to this problem is to combine multiple individuals from the final population into an ensemble. This approach has been successful in supervised classification tasks, and in this paper, it is extended to evolutionary reinforcement learning in control problems. The method is evaluated on a challenging extension of the classic pole balancing task, demonstrating that an ensemble can achieve significantly better performance than the champion alone.

Categories and Subject Descriptors

I.2.6 [**Artificial Intelligence**]: Learning

General Terms

Algorithms, Experimentation

Keywords

neural networks, genetic algorithms, ensembles, reinforcement learning

1. INTRODUCTION

In neuroevolution, a genetic algorithm (GA) is used to evolve a neural network to perform a particular task, such as pattern classification or robot control [13, 11]. The standard approach is to evolve a population over a number of generations, and then select the fittest individual from the final generation, known as the champion, as the end result. However, it is possible that there is valuable information present in the population that is not captured by the champion. For example, in a classification problem, some individuals may correctly classify an input that the champion does not. Such information is not utilized in the standard approach.

One possible solution to this problem is to combine multiple individuals from the final population into an ensemble [14, 9, 3]. This approach has primarily been used in supervised classification tasks, in which the correct output is provided for each input used in training. In this paper, this approach is extended to reinforcement learning tasks in which only a reward signal is provided. As a result of this difference, a method of creating ensembles designed specifically for control problems is needed.

In Section 2, the algorithms commonly used for evolving neural network ensembles are reviewed, and in Section 3, these methods are extended to reinforcement learning tasks. In Section 4, the experimental domain, an extension of the classic pole balancing problem to chasing a moving target, is introduced. The performance of the ensemble is shown to be superior to that of the champion alone in Section 5. These results are compared and contrasted with related work in Section 6, and directions for future work are outlined.

2. ENSEMBLES OF NEURAL NETWORKS

Neural network ensembles are composed of several networks and a method for combining their outputs [10, 7]. Each individual network receives the same input and outputs its own decision. Based on outputs from several different networks, the combination algorithm decides the final output. Ideally, the performance of the entire ensemble will be superior to the performance of any individual network it contains. Two main choices must be made when designing ensembles: a method of training the networks in a way that encourages diversity of behavior, and a mechanism to decide the final output based on outputs from individual networks.

2.1 Training the networks

When training networks to be used in ensembles, it is important that the outputs of the networks are diverse – otherwise, there could be no benefit from combining the networks. While a wide variety of algorithms exist for promoting diversity among ensemble members in supervised learning (e.g. bagging [2] and boosting [4]), this paper focuses on evolutionary methods, in particular neuroevolution. In the neuroevolution approach, the weights and topologies of neural networks are represented as genomes which are evolved over a number of generations through a GA using mutation and crossover operations.

Although standard GA methods are not designed with the goal of developing ensemble members, maintaining a diverse population is important for evolution in general. If no attempt is made to preserve diversity, genomes will converge until they are largely the same. After that point, crossover will have little effect and progress will depend mostly on mutation, increasing the likelihood that the population becomes trapped at a local minimum.

Several methods of maintaining population diversity, generally referred to as speciation (or niching), exist for GAs. One example is the island model , in which the population is divided into non-interacting subpopulations, and diversity is essentially promoted by chance. Another example is fitness sharing, in which fitness is divided among nearby neighbors. Neighbors can be determined through a comparison of network structure (explicit fitness sharing) or of behavior (implicit fitness sharing). The result is that networks are rewarded for uniqueness. The choice of speciation method is extremely important for neuroevolution of ensembles, as the final population must remain sufficiently diverse to make the creation of an ensemble beneficial.

2.2 Combination algorithm

Once a population of networks has been trained, an algorithm must be chosen for combining them into an ensemble. Four common combination algorithms are now described, and their applicability to control problems considered.

In the majority voting approach, each network in the ensemble casts a (possibly weighted) vote for the output. In cases where only a small number of discrete outputs are possible, this method might perform reasonably well. However, in many control problems the outputs represent real-valued quantities, such as a force to apply or an angle to turn, and no two networks will share exactly the same output.

Another possibility is to take a weighted average of the networks' outputs. While this method might seem appropriate for continuous actions, it can in fact lead to disaster. For example, consider designing a robotic car controller that chooses an angle in which to turn. If the controller receives inputs indicating that another car is heading directly towards it, it should either turn left or right. However, if a combination algorithm based on averaging network outputs is used, the average of several left and right turns might be no turn at all, which is the worst possible action. In the case of discrete actions, such an approach may simply be ill-defined. For instance, the available actions could be"sit" and "walk", which cannot be averaged in any meaningful way.

A third option is the winner-takes-all approach, in which the output of the network with the strongest activation is chosen (e.g., in the case of sigmoidal outputs, the output closest to 0 or 1). While this approach avoids the problems associated with the two previous methods, it is only suitable for problems in which a network's output can be interpreted as a measure of confidence in a binary decision. It cannot be used with real-valued actions.

A final choice is to use a gating network. In this approach, a network outside the ensemble is trained to form a proper combination of the outputs of the ensemble members (similar to experts in supervised learning [6]) as a function of the input signal. In essence, the gating network learns which experts to trust for various inputs. The output of the gating network can be interpreted either as the weights for a linear combination of the experts' outputs, or as a decision on which individual expert to follow. The latter method is used in the approach to evolving ensembles described in the next section.

3. NEUROEVOLUTION OF ENSEMBLES

In this section, an approach is presented for applying ensembles of neural networks to control problems. To evolve the population of networks, NeuroEvolution of Augmenting Topologies (NEAT) [12] is used. NEAT evolves both the topologies and weights of neural networks, starting with minimal networks and gradually adding complexity over time. NEAT's approach to speciation is depended on to ensure diversity in the population. In NEAT, speciation is performed using a form of explicit fitness sharing. Individuals are grouped into species based on a comparison of topological similarity, and then compete only within their species. This technique allows novel structures to evolve independently and realize their potential without having to compete with other successful solutions. The size of each species is limited by the shared fitness of its members. Although speciation based only on topological similarity does not guarantee that the behavior is diverse as well, for most problems the resulting diversity is in practice sufficient.

The population evolves until the fitness of its best individual remains the same for many generations. The individuals that will serve as experts are then selected by choosing the fittest individual from a number of different species. Finally, NEAT is used again to evolve a gating network that will make use of the experts. For the input to the gating algorithm the same input is used as for the experts, i.e. the current state of the control problem. The gating network has one output for each expert. For the reasons described in the previous section, the highest output is taken to represent the expert whose output should be chosen at the current step. One benefit to this method is that the gating network can be evaluated first, after which the output of only one of the experts needs to be computed, saving a significant amount of computation time when the number of experts is large.

This approach is similar to that of hierarchical reinforcement learning, in which control policies are learned on multiple levels (see [1] for an overview). In the simplest case of two levels, low-level policies are learned to choose among the primitive actions defined by a control problem, while a higher-level policy chooses among the abstract "actions" represented by the low-level policies. In general, once a low-level policy is chosen, it is followed until a termination condition is met, meaning that the high-level policy is actually choosing between temporally extended actions. Although the gating network chooses an expert at each time step, its effect can be viewed as a partitioning of the problem's state space into regions, with one expert assigned to each region. Thus, this approach fits into the hierarchical reinforcement learning framework if the gating network is viewed as the means of checking for the termination condition, where the condition for termination is that the current state input has left the region of the currently active expert.

One issue that must be considered when using this approach is that the behavior of the experts when used as part of an ensemble may be unpredictable if their networks involve recurrent connections. Recurrency is often useful in control problems where the state representation is non-Markovian, because it represents a way for a neural network to "remember" information from the past. For instance, in a control problem involving moving objects, if the input representing the current state provides only the positions of objects, recurrency might allow networks to deduce information such as the velocities of the objects that cannot be determined directly from the input at the current time step. The ability to utilize recurrency in this manner has been demonstrated by NEAT for various pole balancing problems [12]. However, if only the output of one expert is computed at each time step, the behavior of recurrent experts may change. While the output of all experts could be computed at each time step, problems could still arise if an expert used recurrency to remember past actions it had output, but these actions were not the ones taken by the ensemble. For these reasons, in the following experiments, recurrency was allowed in the gating network but not in the expert networks.

4. THE POLE CHASING PROBLEM

To test this method of evolving ensembles, an extension of the classic control problem of pole balancing is used. In the standard pole balancing problem, a pole is attached to the top of a cart with a hinge, forming an inverted pendulum, and the cart can move along a track of limited length. The goal is to learn how to apply force to the cart to keep the pole from falling over while keeping the cart within the boundaries of the track. A number of learning methods have been shown to be effective in learning to perform this task, including NEAT [12].

Because the goal in this paper is to show that the champion of an evolutionary process can be outperformed by an ensemble, a more difficult control problem than standard pole balancing is needed. While a number of extensions such as double pole balancing [12] and balancing a pole in two dimensions [5] have been explored in the literature, these problems can also be solved using existing methods. For evaluating the approach presented in this paper, a problem is needed for which current methods are unable to consistently learn an optimal solution, and for which the performance of solutions can easily be quantified.

To meet this need, a problem called the *pole chasing problem* is now introduced (see Figure 1; Appendix). In this problem, the task is not only to keep the pole from falling, but also to move the tip of the pole to a desired position. A randomly moving particle represents the target position, and the goal is to minimize the average distance over time between the tip of the pole and the particle, while keeping the pole up and the cart on the track. In addition, a telescoping pole is now used that can be slowly extended or retracted, within a limited range. This new ability makes it possible for the tip of the pole to be brought closer to the particle than before, but it complicates the system dynamics because the pole's behavior depends on its length. At the same time, the dimensionality of the action space is doubled. The particle's motion is described by a velocity and angle of movement, and these are updated at each step according to an acceleration and angular velocity that follow random walks. Collisions with a bounding box surrounding the track are elastic, and momentum is not modeled.[1]

[1]A video of the pole chasing problem, along with related information, is available at `http://nn.cs.utexas.edu/keyword?NEATEnsembles`

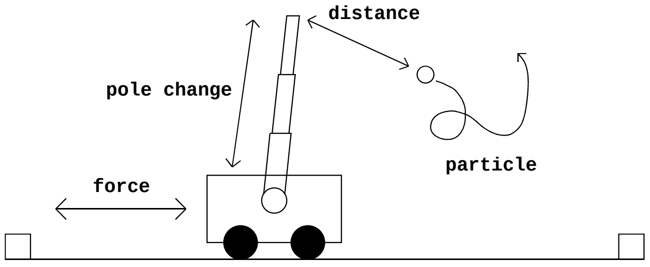

Figure 1: The pole chasing problem. As with the standard pole balancing problem, the pole must be kept upright and the cart must remain on the track. In addition, the tip of the pole is to be brought as close as possible to the randomly moving particle. The pole can be extended or retracted.

The state space is represented by the following nine variables:

1. the cart's x coordinate,
2. the cart's velocity,
3. the pole's angle,
4. the pole's angular velocity,
5. the length of the pole,
6. the x coordinate of the particle,
7. the y coordinate of the particle,
8. the change in the particle's x coordinate since the last step, and
9. the change in the particle's y coordinate since the last step.

The outputs to be determined at each step, representing the actions to take, are:

1. the sideways force to apply to the cart, and
2. the amount by which to change the pole length.

Both outputs are real values restricted to a specific interval.

An episode ends when the pole falls, the cart leaves the track boundary, or some maximum number of time steps is reached. While the goal is to minimize the average distance from the pole to the particle, the fitness function used also includes a reward for simply keeping the episode going, in order to speed up the initial stages of learning. The fitness function is defined as the total reward over an episode, where the reward at each time step is

$$reward = 1 + \left(1 - \frac{particle\ distance}{maximum\ particle\ distance}\right)$$

5. EXPERIMENTAL RESULTS

The method detailed in Section 3 was tested on the pole chasing problem with a maximum episode length of 10,000 steps and for ensemble sizes ranging from two to ten experts. The ANJI implementation of NEAT[2] was used, based on the default parameters provided (including a population size of one hundred), with the exception that recurrent connections were not allowed. In each of the 30 runs, the population was trained for 150 generations. For each ensemble size tested, the appropriate number of experts was chosen, and the gating network was trained for 50 generations. In each run, the number of species in the final population was at least ten, permitting the choice of up to ten experts from different species.

A representative run (using eight experts) is shown in Figure 2, which shows the decrease over time in the average distance between the pole and the particle. (Note that this is not equivalent to the fitness function: the reward for simply keeping the pole from falling is not represented in this plot, because for each point shown the maximum episode length was reached.) It turns out that the ensemble, controlling the pole after generation 150, immediately improves over the population's final champion. The ensemble further improves its performance over a few generations to reach its minimum result, and performance remains level afterward (i.e., training beyond 50 generations produces no improvement). This result suggests that it is not particularly difficult for the gating network to learn how to partition the state space, but that only a fixed amount of improvement is possible.

As can be observed in Figure 2, the fitness evaluation is somewhat noisy. To address this problem, the results of several episodes could be averaged when evaluating each individual's fitness. However, in addition to multiplying the already significant time required for computation, this method actually reduced the quality of the solutions evolved. Apparently, in the early generations, individuals that aggressively chase the particle are somewhat less able to keep the pole upright than more conservative individuals, and increasing the number of samples increases the likelihood that an aggressive individual's fitness will be hurt by an extremely poor episode, reducing that individual's chances of being chosen to reproduce. When only one sample is taken, such aggressive behaviors are given more time to develop, eventually leading to individuals that are both aggressive and stable. Fitness evaluations are therefore performed using only one episode during evolution. However, when evaluating the final populations of both the standard and gating networks, the average of 30 episodes is taken.

Figure 3 shows the average improvement over the single champion for each ensemble size tested, where the improvement is measured by the decrease in the average distance between the pole and the particle. A significant improvement can be observed even with only two experts. The best results are observed when eight experts are used, although the differences between the performances of six, eight, and ten experts are not statistically significant.

Figure 4 shows the results of the 30 runs when using eight experts. The x-axis represents the average distance between the pole and the particle for the champion after 150 generations, and the y-axis shows the amount by which this distance is decreased after the ensemble has evolved. In 28 of the 30 runs, the performance of the ensemble is superior to the performance of the single champion. In the two cases where the average distance increases, the amount of increase is well under 1%, possibly the result of noisy evaluations, while in 13 runs the average distance decreases by over 10%, including 34% in one case. The average improvement seen from the champion to the ensemble is statistically significant at the 99% confidence level according to a paired t-test.

[2]`http://anji.sourceforge.net`

6. DISCUSSION AND FUTURE WORK

The goal of this paper was to demonstrate that the performance of the final champion can be improved by forming an ensemble from the final population. As the experimental results show, this method produces consistent improvements across the wide range of runs, confirming this hypothesis.

One interesting aspect of the results is that the performance between the champions of the 150th generation varied significantly. Although in most runs fitness reached a plateau by the 150th generation, it is possible that given more time to evolve, this variance would decrease. In limited testing of longer runs, there were cases where populations that hit an early low-fitness plateau later improved. However, many hundred generations were needed for any improvement to appear, the level of fitness reached never exceeded the level reached by the more successful runs within 150 generations, and sometimes fitness actually decreased over time. Given these observations, it appears that the most efficient way to build a good controller in real-world applications is to simply perform several shorter runs and use the best.

Previous work on utilizing an entire evolved population of neural networks through the use of ensembles has focused on classification problems. While control problems present a very different set of challenges, it is possible that some of the ideas explored in the classification setting could be adapted to the control setting as well. For instance, Yao and Liu [14] used a GA to search for the optimal subset of the population to use in the ensemble, finding that the full benefit from using an ensemble could be obtained using much less than the full population. In further work [8], they developed a coevolutionary approach to encourage individual networks to specialize on part of the input space through implicit fitness sharing. In a similar coevolutionary approach, Garcia et. al. [9] proposed that a population of ensembles could be evolved along with the population of networks. Fitness evaluation can then take place using a form of multi-objective optimization; for instance, networks can be evaluated on how well the ensembles in which they participate perform, how accurate they are on the training data, and how strongly their output is correlated with that of other networks. Bruce and Miikkulainen [3] showed how networks could be evolved to output both a classification and a confidence factor. Classification accuracy was shown to be significantly higher when the output of the most confident network was used instead of the champion's output. All of these methods could be incorporated into the neuroevolution of ensembles for control problems, possibly improving the performance reported in this paper.

One particular possibility for future work suggested by these methods is explicitly encouraging specialization while evolving the networks. This paper has mainly considered

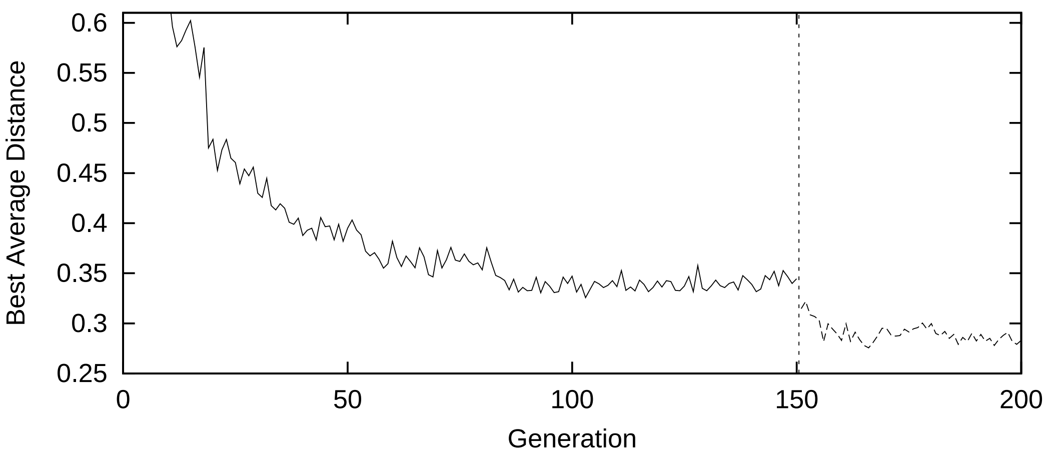

Figure 2: Results from one run using eight experts, showing the average distance between the tip of the pole and the particle over a number of generations. For the first 150 generations, the results represent the performance of the population's champion, and a plateau is reached. After generation 150, the results represent the performance of the evolving ensemble, and an improved level of performance is quickly reached.

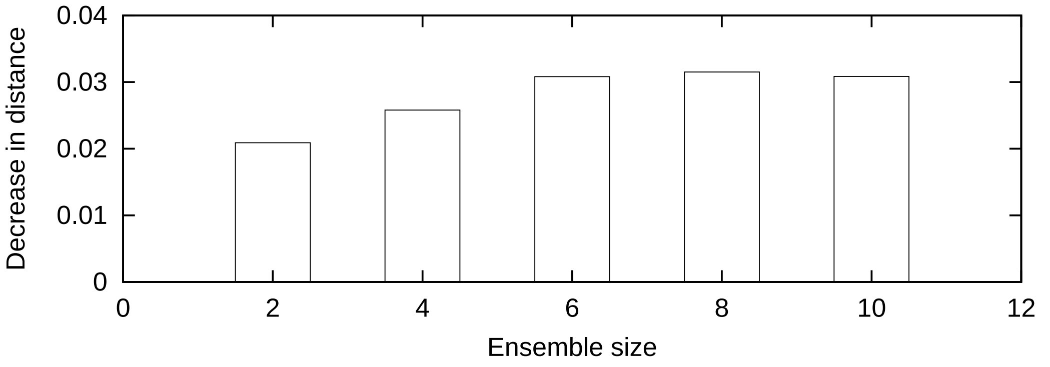

Figure 3: Average improvement over the champion for different ensemble sizes in 30 runs. The y-axis shows the reduction in the average distance between the tip of the pole and the particle that results when an ensemble of a given size is used. Performance improves significantly with only two experts, and is strongest with eight, although the differences between six, eight, and ten are not statistically significant.

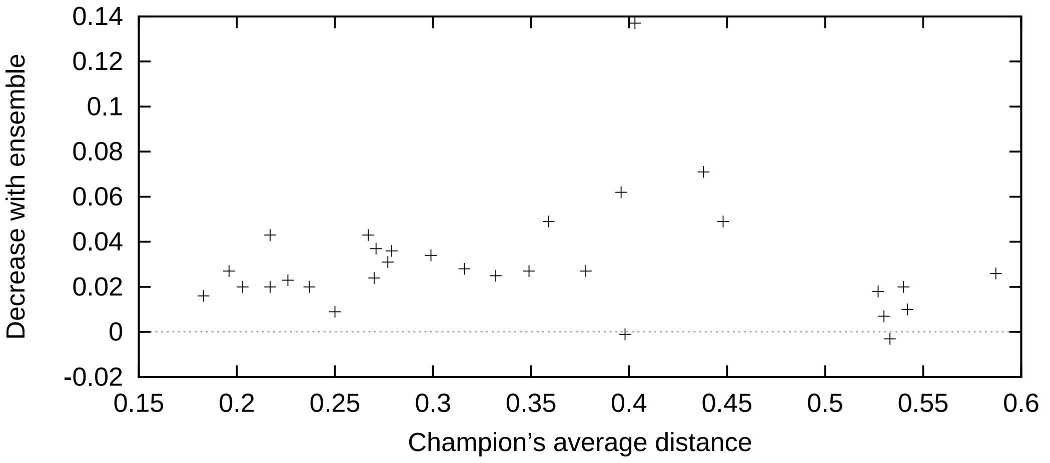

Figure 4: Improvement of the ensemble over the champion in 30 runs using eight experts. The x-axis represents the average distance between the tip of the pole and the particle for the champion of the 150th generation. The reduction in the average distance that results when an ensemble is evolved is shown on the y-axis. In 28 out of 30 runs, performance improved.

how to utilize the final population of networks, while assuming it would contain a useful amount of diversity due to speciation. Just as networks used for classification can be encouraged to specialize on a portion of the training data, networks used for control could be encouraged to specialize on a part of the problem's state space. Such specialization would complement the development of a gating network that chooses one expert for each input. However, evaluating the fitness of specialized individuals is difficult. When feedback takes the form of a reward signal, action choices cannot be evaluated in isolation, because the value of taking an action depends on future as well as immediate rewards. These future rewards, in turn, depend on future actions that may end up being chosen by another member of the ensemble. This problem may be solvable by a coevolutionary approach where both the experts and the gating network evolve simultaneously, with the fitness of an expert network depending on the success of the gating networks that use it.

The current approach could also be extended by continuing to evolve the experts once their roles are defined by the gating network, or even evolving them from scratch at that point. In preliminary experiments, it was found that performance could be slightly improved by re-evolving one expert while leaving the others fixed. How to systematically make use of such improvements is an interesting direction for future work.

7. CONCLUSION

In this paper, a method is is presented for evolving ensembles of neural networks to perform control tasks in a reinforcement learning setting. Experimental results show that given a population resulting from an evolutionary process, better performance can be obtained by combining individuals into an ensemble than by simply using the champion individual. The method described here is sufficiently general in principle that it can be used in conjunction with any approach to evolving a population of individuals representing policies for a control problem, making it widely applicable. As in classification tasks, ensembles can therefore be used to significantly increase the performance of neuroevolution in solving control problems.

Acknowledgments

This research was supported in part by the National Science Foundation under grant EIA-0303609.

8. REFERENCES

[1] A. Barto and S. Mahadevan. Recent advances in hierarchical reinforcement learning. *Discrete Event Dynamic Systems*, 13(4):341–379, 2003.

[2] L. Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.

[3] J. Bruce and R. Miikkulainen. Evolving populations of expert neural networks. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, pages 251–257, San Francisco, CA, 2001.

[4] Y. Freund and R. E. Schapire. Experiments with a new boosting algorithm. In *International Conference on Machine Learning*, pages 148–156, 1996.

[5] F. Gomez and R. Miikkulainen. 2-d pole balancing with recurrent evolutionary networks. In *Proceedings of the International Conference on Artificial Neural Networks (ICANN-98, Skovde, Sweden)*, pages 425–430, 1998.

[6] R. A. Jacobs, M. I. Jordan, S. Nowlan, and G. E. Hinton. Adaptive mixtures of local experts. *Neural Computation*, 3:79 – 87, 1991.

[7] A. Krogh and J. Vedelsby. Neural network ensembles, cross validation, and active learning. In *Advances in Neural Information Processing Systems*, volume 7, pages 231–238. The MIT Press, 1995.

[8] Y. Liu and X. Yao. Towards designing neural network ensembles by evolution. In *Parallel Problem Solving from Nature - PPSN V*, volume 1498, pages 623–632, Amsterdam, The Netherlands, 1998.

[9] D. O. N. Garca, C. Hervs. Cooperative coevolution of artificial neural network ensembles for pattern classification. *IEEE Transactions on Evolutionary Computation*, 2005. In press.

[10] D. W. Opitz and J. W. Shavlik. Generating accurate and diverse members of a neural-network ensemble. In *Advances in Neural Information Processing Systems*, volume 8, pages 535–541. The MIT Press, 1996.

[11] J. D. Schaffer, D. Whitley, and L. J. Eshelman. Combinations of genetic algorithms and neural networks: A survey of the state of the art. In *Proceedings of the International Workshop on Combinations of Genetic Algorithms and Neural Networks (COGANN-92)*, pages 1–37, Los Alamitos, CA, 1992. IEEE Press, Piscataway, New Jersey.

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

[13] X. Yao. Evolving artificial neural networks. *Proceedings of the IEEE*, 87(9):1423–1447, 1999.

[14] X. Yao and Y. Liu. Making use of population information in evolutionary artificial neural networks. *IEEE Transactions on Systems, Man, and Cybernetics*, 28B(2), April 1998.

Appendix

Table 1: Parameters for the pole chasing problem.

Parameter	Value or Range
cart and pole	
mass of cart	1.0 kg
mass of pole	0.1 kg
pole length	[0.5, 1.5] m
force	[-10, 10] N
position on track	[-2.4, 2.4] m
angle of pole from vertical	[-50, 50] degrees
time per step	0.05s
particle	
velocity	[-2, 2] m/s
angular velocity	[-360, 360] degrees/s
step size of random walk for vel.	[-0.1, 0.1] m/s
step size of r. walk for ang. vel.	[-.05, .05] m/s
x position	[-2.4, 2.4] m
y position (from base of pole)	[0.5, 1.5] m