Automatic Feature Selection in Neuroevolution

Shimon Whiteson, Kenneth O. Stanley, Risto Miikkulainen

Department of Computer Sciences
The University of Texas at Austin
1 University Station C0500
Austin, Texas 78712-0233
{shimon,kstanley,risto}@cs.utexas.edu
http://www.cs.utexas.edu/~{shimon,kstanley,risto}

Abstract. Feature selection is the process of finding the set of inputs to a machine learning algorithm that will yield the best performance. Developing a way to solve this problem automatically would make current machine learning methods much more useful. Previous efforts to automate feature selection rely on expensive meta-learning or are applicable only when labeled training data is available. This paper presents a novel method called FS-NEAT which extends the NEAT neuroevolution method to automatically determine the right set of inputs for the networks it evolves. By learning the network's inputs, topology, and weights simultaneously, FS-NEAT addresses the feature selection problem without relying on meta-learning or labeled data. Initial experiments in a line orientation task demonstrate that FS-NEAT can learn networks with fewer inputs and better performance than traditional NEAT. Furthermore, it outperforms traditional NEAT even when the feature set does not contain extraneous features because it searches for networks in a lower-dimensional space.

1 Introduction

To implement a successful machine learning (ML) system, human designers must make several preliminary decisions. For example, setup usually requires choosing a representation for the solution, selecting relevant inputs, and setting all the parameters associated with the given learning method. Performing these tasks manually is time-consuming and may yield sub-optimal results if not done correctly. Hence, one important goal of ML research is to develop techniques that shift some of the burden off the human designer and place it onto the learning algorithm itself. Doing so will reduce the time and expertise required to use ML techniques and greatly broaden the set of tasks to which they can be practically applied.

One of the decisions that is usually addressed manually is the *feature selection* problem. In many real world tasks, the set of potential inputs, or features, that can be fed to the learning algorithm is quite large. Feature selection is the process of determining which subset of these inputs should be included to generate the best performance. Doing so correctly can be critical to the success of an ML system. If any important features are excluded, it may be impossible to find an optimal policy. On the other hand, including superfluous inputs can also impede learning. Since the size of the search space increases exponentially with the input size, even a few extraneous features can be detrimental. However, the consequences of sub-optimal feature selection are not limited just to the learner's performance. If adding inputs costs money (e.g. putting more sensors on a robot), then pruning out unnecessary features can be vital.

Feature selection can often be performed by a human with the appropriate domain expertise. However, in some domains, no one has the requisite knowledge and even when experts do exist, employing them can be expensive and time-consuming. In such domains, automatic feature selection is necessary. Langley [6] divides feature selection techniques into two categories: *filters* and *wrappers*. Filters [1, 5] analyze the value of a feature set without regard to the learning algorithm that will use those features. Instead, they rely on labeled data. The data is analyzed to determine which features are most useful in distinguishing between the category labels. This approach has been successful but works only in supervised learning tasks. In reinforcement learning scenarios, when no labeled data is available, filtering techniques are not applicable.

By contrast, wrappers [7, 8] test a feature set by applying it to the given learning algorithm and observing its performance. Labeled examples are not necessary so this approach can be used in reinforcement tasks as well. However, it requires a meta-learner to search through the space of feature sets; evaluating any point in that space requires an entire ML run of its own. For most real-world problems, this approach is computationally infeasible.

Feature Selective NeuroEvolution of Augmenting Topologies (FS-NEAT) is a new learning method that avoids such limitations by incorporating the feature selection problem into the learning task. FS-NEAT searches for good feature sets at the same time it trains networks that receive those features as input. Hence, it does not depend on human expertise, labeled data sets, or meta-learning.

FS-NEAT is based on the NeuroEvolution of Augmenting Topologies (NEAT) algorithm [12], which evolves both the topology and weights of a neural network. NEAT already goes a long way towards reducing the burden on human designers. While most neuroevolution methods evolve only the weights of a network with fixed topology, NEAT learns an appropriate topology automatically. FS-NEAT goes one step further by learning the network's inputs too. Using evolution, it automatically and simultaneously determines the network's inputs, topology, and weights. Harvey et al. [4] also used neuroevolution to find useful subsets of available features though, unlike FS-NEAT, their system still requires a human to specify the size of that subset in advance.

A critical feature of NEAT is that it begins with networks of minimal topology and adds new nodes and links solely through mutation. Since only those additions that improve performance will be retained, it tends to learn small networks without superfluous structure. Starting minimally also helps NEAT learn more quickly. When networks in its population are small, it is optimizing over a lower-dimensional search space; it jumps to a larger space only when performance in the smaller one stagnates. FS-NEAT further exploits this same premise. It begins with a population of networks with a minimal number of inputs. New inputs are added through mutation and only the useful ones are likely to persist in the population.

Hence, we hypothesize that FS-NEAT, by automatically selecting appropriate inputs, can outperform traditional NEAT in tasks where no domain expert is available to prune a large input set with irrelevant and/or redundant features. Furthermore, because it searches in lower-dimensional spaces, FS-NEAT should outperform traditional NEAT even when the feature set does not contain extraneous features. The preliminary experiments presented in this paper confirm both these hypotheses.

The remainder of this paper is organized as follows. Section 2 explains the NEAT algorithm and the modifications made to it to yield FS-NEAT. Section 3 introduces the line orientation domain and presents the results of our preliminary experiments comparing NEAT and FS-NEAT. Section 4 discusses the implications of these results. Section 5 outlines some opportunities for future work.

2 Method

This section provides background on the NEAT algorithm. It also introduces FS-NEAT, our novel modification to the original NEAT method.

2.1 NeuroEvolution of Augmenting Topologies

In most domains, the optimal topology and complexity for a neural network is unknown. With most neuroevolution techniques, a human designer must try to guess it. Since the topology determines the size of the search space, the consequences of guessing wrong can be severe. Searching in too large a space is intractable while searching in too small a space limits solution quality. NeuroEvolution of Augmenting Topologies (NEAT) [12] is a neuroevolution technique that does not require a designer to choose a topology in advance. Instead, it automatically evolves the topology to fit the complexity of the problem. NEAT combines the usual search for appropriate network weights with complexification of the network structure.

This approach is highly effective: NEAT outperforms other neuroevolution (NE) methods, e.g. on the benchmark double pole balancing task [11, 12]. In addition, because NEAT starts with simple networks and expands the search space only when beneficial, it is able to find significantly more sophisticated controllers than fixed-topology evolution, as demonstrated in a robotic strategy-learning domain [10, 13]. These properties make NEAT an attractive method for evolving neural networks.

In this section, the NEAT method is briefly reviewed; a more comprehensive description of the NEAT method is given in [12].

Genetic Encoding with Historical Markings Evolving network structure requires a flexible genetic encoding. Each genome in NEAT includes a list of *connection genes*, each of which refers to two *node genes* being connected. Each connection gene specifies the in-node, the out-node, the weight of the connection, whether or not the connection gene is expressed (an enable bit), and an *innovation number*, which allows NEAT to find corresponding genes during crossover.

Mutations in NEAT can change both connection weights and network structures. Connection weights mutate as in any neuroevolution system; structural mutations, which allow complexity to increase, add either a new connection or node to the network. Through mutation, genomes of varying sizes are created, sometimes with completely different connections specified at the same positions.

In order to perform crossover, the system must be able to tell which genes match up between *any* individuals in the population. For this purpose, NEAT keeps track of the historical origin of every gene. Whenever a new gene appears (through structural mutation), a *global innovation number* is incremented and assigned to that gene. The innovation numbers thus represent a chronology of every gene in the system. Whenever these genomes crossover, innovation numbers on inherited genes are preserved. Thus, the historical origin of every gene in the system is known throughout evolution.

Through innovation numbers, the system knows exactly which genes match up with which. Genes that do not match are either *disjoint* or *excess*, depending on whether they occur within or outside the range of the other parent's innovation numbers. When crossing over, the genes in both genomes with the same innovation numbers are lined up. Genes that do not match are inherited from the more fit parent, or if they are equally fit, from both parents randomly.

Historical markings allow NEAT to perform crossover without expensive topological analysis. Genomes of different organizations and sizes stay compatible throughout evolution, and the problem of matching different topologies [9] is essentially avoided.

Speciation In most cases, adding new structure to a network initially reduces its fitness. However, NEAT speciates the population, so that individuals compete primarily within their own niches instead of with the population at large. This way, topological innovations are protected and have time to optimize their structure before they have to compete with other niches in the population.

Historical markings make it possible for the system to divide the population into species based on topological similarity. The distance δ between two network encodings is a simple linear combination of the number of excess (E) and disjoint (D) genes, as well as the average weight differences of matching genes ($\overline{W}$):

$$\delta = \frac{c_1 E}{N} + \frac{c_2 D}{N} + c_3 \cdot \overline{W}. \tag{1}$$

The coefficients c_1, c_2, and c_3 adjust the importance of the three factors, and the factor N, the number of genes in the larger genome, normalizes for genome size. Genomes are tested one at a time; if a genome's distance to a randomly chosen member of the species is less than δ_t, a compatibility threshold, it is placed into this species. Each genome is placed into the first species where this condition is satisfied, so that no genome is in more than one species.

The reproduction mechanism for NEAT is *explicit fitness sharing* [2], where organisms in the same species must share the fitness of their niche, preventing any one species from taking over the population.

Minimizing Dimensionality Unlike other systems that evolve network topologies and weights [3, 14] NEAT begins with a uniform population of simple networks with no hidden nodes and inputs connected directly to outputs. New structure is introduced incrementally as mutations occur, and only those structures survive that are found to be useful through fitness evaluations. This way, NEAT searches through a minimal number of weight dimensions and finds the appropriate complexity level for the problem. However, NEAT does not start minimally with respect to inputs: they are all assumed to be critical and immediately connected to all outputs. Hence, in domains with many irrelevant or redundant features, it is possible to begin even more minimally. FS-NEAT, explained next, is an effort to capitalize on that possibility.

2.2 Feature Selective NeuroEvolution of Augmenting Topologies

NEAT starts with networks of minimal structure and adds the nodes and links that are necessary to solve the task. Similarly, Feature Selective NeuroEvolution of Augmenting Topologies (FS-NEAT) starts with networks with very few inputs and adds more as necessary to solve the task. To implement FS-NEAT, two changes to NEAT are necessary. First, networks in the first generation must be initialized with minimal inputs. Second, a new mutation operator is needed to add new inputs to the network.

In FS-NEAT, the networks in the initial population are created via the following procedure. First, a bias input is connected to each output node. Second, given a set I of all possible inputs, each $i \in I$ is given a $\frac{1}{|I|}$ probability of being added to the network. Any input node that is added is connected to all the outputs. Since a network whose only input is a bias cannot be useful, a randomly selected input is added to any such network. This approach yields a diverse initial population from which evolution can select the strongest individuals.

In order for FS-NEAT to learn which inputs are important, it needs a way to add new inputs during evolution. A new mutation operator, illustrated in Figure 1, is added to NEAT's reproductive mechanism. When a new network is generated through crossover, there is a small probability that a new input will be added to the network; if so, it is connected to all the output nodes in the network, as during initial construction. The network may have evolved hidden nodes but the new input will not initially be connected to them. Such connections can be made later via NEAT's existing mutation operators.

These modifications to NEAT yield a new technique, FS-NEAT, which learns the network's inputs, structure, and weights simultaneously. By starting with a

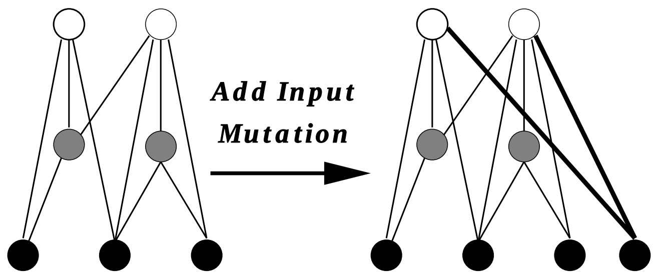


Fig. 1: An example of FS-NEAT's new mutation operator, which adds inputs to a network. Black circles indicate inputs, grey circles indicate hidden nodes, and white circles indicate outputs. New links, shown with thicker black lines, are added between the new input and all outputs, as in the initial construction of the network.

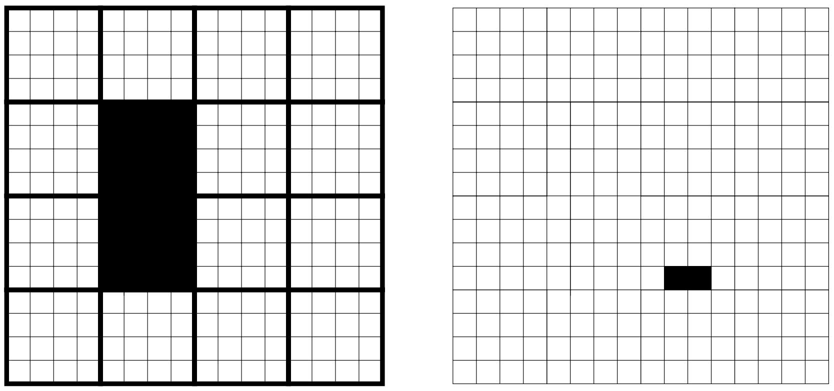
population of networks with very few inputs, adding new inputs through mutation, and retaining only the most useful inputs, FS-NEAT has the capacity to tackle the feature selection problem as part of the learning process.

These modifications to NEAT constitute the implementation of FS-NEAT used in the preliminary experiments reported here. However, they are by no means the only possible way to implement feature selection in NEAT. One alternative would be to initialize all networks without *any* connections. The networks would begin as pools of inputs and outputs. Instead of using a special mutation operator to add inputs, NEAT's existing operators could add hidden nodes and create connections between inputs, hidden nodes, and outputs. Feature selection would occur implicitly, as only connections emerging from useful inputs would tend to survive the selective pressure. How such an alternative approach compares to the implementation tested here is an important empirical question that we hope to address in future work.

3 Experiments

The experiments reported in this paper are conducted on the line orientation task. The network is given a grid of monochromatic pixels and must determine whether a line that appears in that grid has a horizontal or vertical orientation. The possible inputs for this task are simply the set of pixels, each of which has a value of one when the line crosses that part of the grid and zero otherwise. The single output is interpreted as "vertical" if its activation is greater than 0.5 and "horizontal" otherwise.

Figure 2a shows an example of the 16×16 grids used in our first experiment. The 4×8 lines can appear anywhere in the grid but must lie wholly inside two of the 4×4 regions indicated by the thicker lines. A line cannot occupy only part of a region. In this setup, there are 12 different places to put a horizontal line and 12 different places to put a vertical one. Hence, a fitness evaluation consists of asking a network to classify each of the 24 possible lines and counting how many it gets correct. In order to give the fitness function a more useful gradient, a network is assigned credit for how close it came to the perfect output. Hence, if A is the activation on the network's output, then the network is given a score of A for vertical examples and $1 - A$ for horizontal examples.



(a) The orientation task used in the first experiment.

(b) The orientation task used in the second experiment.

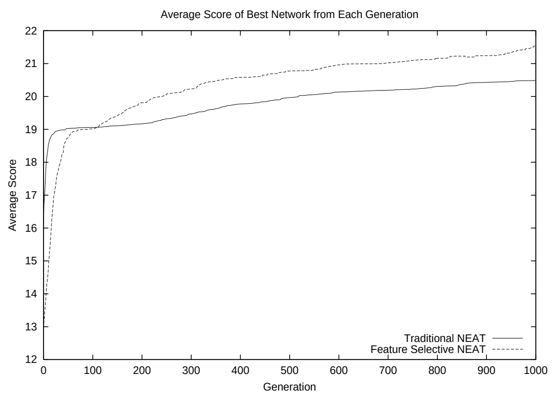
Fig. 2: The line orientation task used in both experiments. In (a), the grid is divided into 16 regions, denoted by thicker black lines. Lines always occupy two adjacent regions, each of which contains a 4×4 array of pixels, denoted with thin black lines, at most one of which is necessary to correctly classify the examples. In (b), a line consists of two adjacent pixels. Almost all of the available pixels are necessary to solve this version of the task.

Since the lines always line up with the regions, the 256 pixels in the grid offer more resolution than is necessary to solve the task. Pixels within a region always have the same value so at most one input is needed for each region. Solutions using even fewer than 16 inputs are possible, though they may be more difficult to learn since the status of some regions must be inferred from surrounding inputs. This setup simulates a situation in which no human experts are available to prune the input set in advance. With traditional NEAT, the only option is to train networks containing all the possible inputs. With FS-NEAT, evolution starts with minimal networks and places the entire burden of feature selection on the learning algorithm.

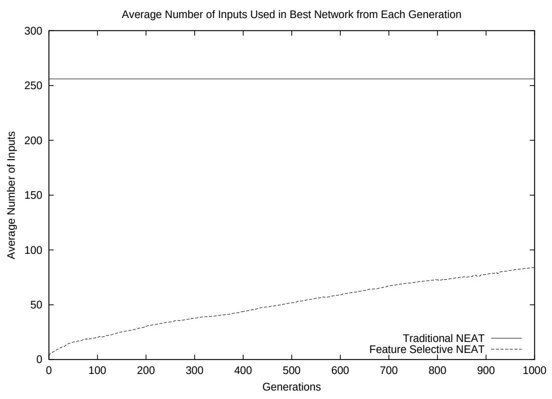
Figure 2b shows an example of the grids used in the second experiment. The grid remains 16×16 in size but now the lines are only 1×2. There are no restrictions on where the lines can be placed, yielding 480 training examples. Because the lines are so small and can appear anywhere, none of the pixels are redundant. While solutions that ignore a few features are possible, almost all of the available inputs are necessary. There is no feature selection challenge in this version of the task but FS-NEAT can still gain an advantage by starting minimally and searching in lower-dimensional spaces.

Clearly, the line orientation task has little to no practical application. It was selected as a preliminary testbed for FS-NEAT because it represents the simplest imaginable scenario in which FS-NEAT's enhancements could pay dividends. Hence, the experiments presented here are intended to demonstrate proof of concept and nothing more.

Figure 3a shows the results of the first experiment. Each line indicates the score received by the best network from each generation, averaged over 30 runs. Each run used a population of 200 networks. The appendix contains more details on the parameters used in both experiments. Early in the experiments, when the networks trained with FS-NEAT do not have sufficient inputs to correctly classify each example, traditional NEAT learns more quickly. By generation 100, however, FS-NEAT matches traditional NEAT's performance and continues to improve, finishing with a higher average score. A t-test confirms that this difference is statistically significant with 95% confidence after generation 149. Seven of the 30 FS-NEAT runs reached a perfect score of 24, whereas only one of the traditional NEAT runs did.



(a) Average score obtained in the first experiment.



(b) Average number of inputs used in the first experiment.

Fig. 3: Results of the first experiment. In (a), the performance of FS-NEAT, shown with a dashed line, is significantly higher than that of traditional NEAT, shown with a solid line. Seven of the 30 FS-NEAT runs reached a perfect score of 24 whereas only one of the traditional NEAT runs did. In (b), FS-NEAT, shown with a dashed line, achieves better performance with only 32.7% as many inputs as traditional NEAT, shown with a solid line.

Figure 3b shows the number of inputs used by the best network from each generation, averaged over all 30 runs. The networks learned with traditional NEAT always have 256 inputs. By contrast, the networks learned with FS-NEAT start with very few inputs and gradually increase in size over the course of evolution. At the end of the experiment, the best network had on average 83.6

inputs. Hence, FS-NEAT learned networks with better performance that require only 32.7% as many inputs.

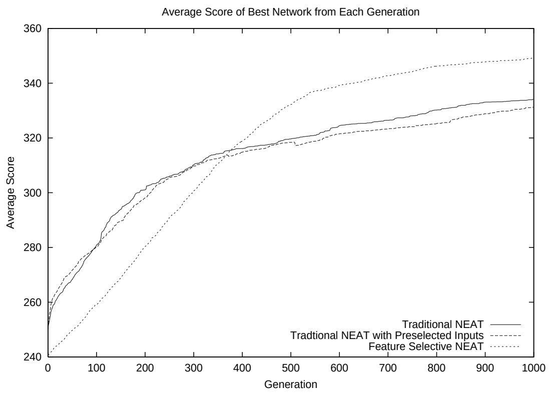
In the second experiment, the lines that appear in the grid are only 1×2 in size. Hence, none of the inputs are redundant. This experiment corresponds to a situation in which feature selection is not necessary but the number of inputs is large enough to make the task challenging. In this case, FS-NEAT starts at a disadvantage: traditional NEAT begins with a sufficient input set whereas FS-NEAT must learn it.

The results of the second experiment are shown in Figure 4a. These results are averaged over 10 runs, each of which uses a population of 200 networks. As in the first experiment, FS-NEAT begins with weaker performance. By generation 400, however, FS-NEAT exceeds traditional NEAT's performance and continues to improve, finishing with a higher average score. A t-test confirms that this difference is statistically significant with 95% confidence after generation 460. The networks evolved by FS-NEAT use on average only 235.7 inputs. Hence, the possibility remains that FS-NEAT's superior performance is due, not to searching in lower-dimensional spaces, but to discovering an advantageous subset of the 256 available inputs. To test this possibility, 10 more runs of NEAT were conducted in which the networks were given only the 238 inputs used by the best network FS-NEAT learned. When given only these preselected inputs, traditional NEAT does not perform any better than it did with all 256 inputs. Hence, the advantage of FS-NEAT in this scenario is due solely to searching in lower-dimensional spaces.

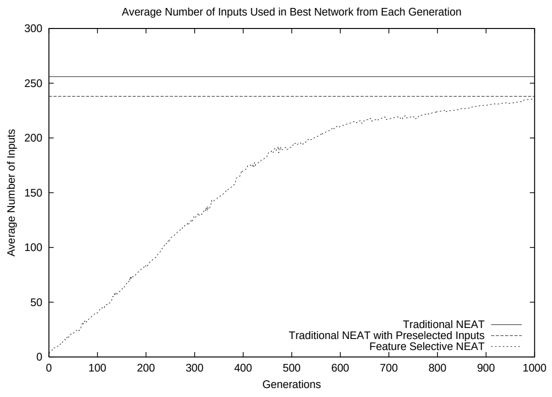
Figure 4b shows the number of inputs used by the best networks from each generation. Traditional NEAT uses whatever inputs it is offered: all 256 in one case and 238 in the other. As expected, FS-NEAT begins with minimal inputs climbs slowly upwards.

4 Discussion

The results reported here demonstrate that FS-NEAT can perform automatic feature selection in a simple domain. The first experiment shows that, given a large feature set but no expert to prune it, FS-NEAT can achieve high scores by using only a fraction of the possible inputs. By avoiding costly search in unnecessarily high-dimensional search spaces, FS-NEAT was also able to obtain better performance than traditional NEAT. While FS-NEAT obtains this advantage using only 32.7% as many inputs as traditional NEAT, it still uses many more than required for a truly minimal solution. However, upon reflection, this result is not surprising. FS-NEAT does not apply any explicit pressure to keep the number of inputs low. Instead, it simply breeds the networks in each species that receive the highest fitness evaluations. While having many superfluous inputs should be a disadvantage, a few may not be significant. After all, by learning to set to zero the weights emerging from useless inputs, even regular NEAT can perform a kind of implicit feature selection. It is only when the difficulty of doing



(a) Average score obtained in the second experiment.



(b) Average number of inputs used in the second experiment.

Fig. 4: Results of the second experiment. In (a), FS-NEAT, shown with a dotted line, performs significantly better than traditional NEAT, shown with a solid line. Traditional NEAT with preselected inputs, shown with a dashed line, does not perform better than traditional NEAT with all 256 inputs. In (b), traditional NEAT, shown with a solid line, always uses all 256 inputs. Traditional NEAT with preselected inputs, shown with a dashed line, uses the 238 inputs chosen by the best run of FS-NEAT. FS-NEAT, shown with a dotted line, starts minimally and gradually adds inputs.

so becomes significant that one should expect FS-NEAT to favor networks with smaller inputs.

The second experiment demonstrates that the value of FS-NEAT is not restricted to solving the feature selection problem. Even when none of the available inputs are redundant, FS-NEAT can still gain an advantage by starting with a minimal feature set. This advantage is not due to discovering a fortuitous subset of the 256 features because traditional NEAT does not match FS-NEAT's performance even when given the subset FS-NEAT uses in advance. In fact, this advantage does not improve its performance at all, probably because the difference between 238 and 256 features is not significant. Hence, the second experiment verifies that FS-NEAT can find higher quality solutions than traditional NEAT by conducting the early part of the search in lower-dimensional spaces.

5 Future Work

The experiments reported in this paper demonstrate that automatic feature selection is possible with FS-NEAT. Additional research is necessary to better determine the algorithm's robustness and range of applicability. The line orientation task has little practical application and is useful only for demonstrating proof of concept. A real test of FS-NEAT would involve a reinforcement learning task, where filtering methods of feature selection are not applicable. For example, robot control tasks with sparse reinforcement, such as predator-prey, are the type of domain NEAT was designed for and often pose difficult feature selection problems. Hence, they should reveal how well FS-NEAT can tackle real-world problems with large feature sets.

Furthermore, as mentioned in Section 2, there are many ways that feature selection could be implemented in NEAT and this paper offers a preliminary test of only one such implementation. Providing an empirical comparison of alternate implementations of this idea could lead to stronger results and more systematic methods.

6 Conclusion

FS-NEAT is a new neuroevolution technique that extends the NEAT method to perform feature selection, a task that human designers usually must complete manually. Unlike other neuroevolution methods, FS-NEAT learns a network's inputs, topology, and weights simultaneously. Initial experiments in a line orientation task demonstrate that FS-NEAT can learn networks with fewer inputs and better performance than the traditional version of NEAT. The results also show that, by exploiting the opportunity to search in lower-dimensional spaces, FS-NEAT can outperform traditional NEAT even when the feature set does not contain extraneous features. Hence, FS-NEAT is a promising new method for shifting some of the burden of optimizing machine learning methods off of human designers and onto the learning methods themselves.

Appendix: NEAT and FS-NEAT System Parameters

The population size was 200. The coefficients for measuring compatibility were $c_1 = 1.0$, $c_2 = 1.0$, and $c_3 = 2.0$. The initial compatibility distance was $\delta_t = 3.0$. However, because population dynamics can be unpredictable over hundreds of generations, a target of 13 species was assigned. If the number of species grew above 13, δ_t was increased by 0.3 to reduce the number of species. Conversely, if the number of species fell below 13, δ_t was decreased by 0.3 to increase the number of species. The champion of each species with more than five networks was copied into the next generation unchanged. The interspecies mating rate was 0.01. The probability of adding a new node was 0.02 and the probability of a new link mutation was 0.08. In FS-NEAT runs, the probability of adding a new input was set to 0.05 in the first experiment and 0.2 in the second experiment. These parameter values were verified experimentally.

References

1. B. V. Bonnlander and A. S. Weigend. Selecting input variables using mutual information and nonparametric density estimation. In *Proceedings of the 1994*

International Symposium on Artificial Neural Networks (ISANN"94), pages 42–50, Tainan, Taiwan, 1994.
2. D. E. Goldberg and J. Richardson. Genetic algorithms with sharing for multimodal function optimization. In *Proceedings of the Second International Conference on Genetic Algorithms*, pages 148–154, 1987.
3. F. Gruau, D. Whitley, and L. Pyeatt. A comparison between cellular encoding and direct encoding for genetic neural networks. In J. R. Koza, D. E. Goldberg, D. B. Fogel, and R. L. Riolo, editors, *Genetic Programming 1996: Proceedings of the First Annual Conference*, pages 81–89. MIT Press, 1996.
4. P. R. Harvey, D. M. Booth, and J. F. Boyce. Evolving the mapping between input neurons and multi-source imagery. In *Proceedings of the 2002 Congress on Evolutionary Computation*, pages 1878–1883, 2002.
5. K. Kira and L. Rendell. A practical approach to feature selection. In *Proceedings of the Tenth International Conference on Machine Learning*, Amherst, Massachusetts, 1992. Morgan Kaufmann.
6. P. Langley. Selection of relevant features in machine learning. In *Proceedings of AAAI Fall Symposium on Relevance*, 1994.
7. P. M. Narendra and K. Fukunaga. A branch and bound algorithm for feature subset selection. *IEEE Transactions on Computers*, 26:917–922, 1977.
8. J. Novovivova, P. Pudil, and J. Kittler. Floating search methods in feature selection. *Pattern Recognition Letters*, 15:1119–1125, 1994.
9. N. J. Radcliffe. Genetic set recombination and its application to neural network topology optimization. *Neural computing and applications*, 1(1):67–90, 1993.
10. K. O. Stanley and R. Miikkulainen. Continual coevolution through complexification. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2002)*, 2002.
11. K. O. Stanley and R. Miikkulainen. Efficient reinforcement learning through evolving neural network topologies. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2002)*, 2002.
12. K. O. Stanley and R. Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary Computation*, 10(2):99–127, 2002.
13. K. O. Stanley and R. Miikkulainen. Competitive coevolution through evolutionary complexification. *Journal of Artificial Intelligence Research*, 21, 2004. In press.
14. X. Yao. Evolving artificial neural networks. *Proceedings of the IEEE*, 87(9):1423–1447, 1999.