Evolving Deep Recurrent Neural Networks Using Ant Colony Optimization

Travis Desell[1], Sophine Clachar[1], James Higgins[2], and Brandon Wild[2]

[1] Department of Computer Science, University of North Dakota
tdesell@cs.und.edu, sophine.clachar@my.und.edu
[2] Department of Aviation, University of North Dakota
jhiggins@aero.und.edu, bwild@aero.und.edu

Abstract. This paper presents a novel strategy for using ant colony optimization (ACO) to evolve the structure of deep recurrent neural networks. While versions of ACO for continuous parameter optimization have been previously used to train the weights of neural networks, to the authors' knowledge they have not been used to actually design neural networks. The strategy presented is used to evolve deep neural networks with up to 5 hidden and 5 recurrent layers for the challenging task of predicting general aviation flight data, and is shown to provide improvements of 63% for airspeed, a 97% for altitude and 120% for pitch over previously best published results, *while at the same time not requiring additional input neurons for residual values*. The strategy presented also has many benefits for neuro evolution, including the fact that it is easily parallizable and scalable, and can operate using any method for training neural networks. Further, the networks it evolves can typically be trained in fewer iterations than fully connected networks.

Keywords: Ant Colony Optimization, Time-Series Prediction, Neural Networks, Flight Prediction, Aviation Informatics

1 Introduction

Neural networks have been widely used for time series data prediction [11, 43]. Unfortunately, current popular techniques for designing and training neural networks such as convolutional and deep learning strategies, popular within computer vision, do not easily apply to time series prediction. This is in part because the number of input parameters is relatively small (compared to pixels within images), the fact they do not easily deal with recurrent memory neurons, and the goal is prediction, as opposed to classification. Even more problematic, these strategies do not help address the rather challenging problem of determining the best performing structure for those neural networks. Automated strategies for simultaneously evolving the structure and weights of neural networks have been examined through strategies such as NeuroEvolution of Augmenting Topologies (NEAT) [37] and Hyper-NEAT [38], and while these can evolve recurrent connections, they require non-trivial modification to evolve the recurrent memory nureons typically used for time series prediction.

Recent work in using neural networks for time series prediction has involved utilizing *residuals* or *lags* similar to the Auto-Regressive Integrated Moving Average

(ARIMA) model [42], as done by Khashei *et al.* [23] and Omer *et al.* [30]. Other work has investigated strategies for cooperative co-evolution for Elman recurrent neural networks [10, 9], however these strategies involve single parameter time series data such as the Mackey-Glass, Lorenz and Sunspot data sets.

Ant colony optimization (ACO) [6, 19, 17] is an optimization technique originally designed for use on discrete problems, with a common example being the Traveling Salesman Problem [18]. It has since been extended for use in continuous optimization problems [34, 36, 35, 5, 27, 20], including training artificial neural networks [24, 7, 31, 40, 3]. While ACO has been studied for training artificial neural networks (ANNs), to the authors' knowledge there is little work in using ACO to actually *design* neural networks, with the closest being Sivagaminathan *et al.* using ACO to select input features for neural networks [33].

This work presents a novel strategy based on ant colony optimization which evolves the structure of recurrent deep neural networks with multiple input data parameters. While ant colony optimization is used to evolve the network structure, any number of optimization techniques can be used to optimize the weights of those neural networks. Trained neural networks with good fitness will be used to update the pheromones, reinforcing connections between neurons that provide good solutions. The algorithm is easily parallelizable and scalable, using a steady state population of best performing neural networks to determine when pheromones are incremented, and any number of worker processes can asynchronously train neural networks generated by the ant colony optimization strategy.

This algorithm is evaluated using the real world problem of predicting general aviation flight data, and compared to previously best published results for a set of testing data. For three of the four parameters evaluated (airspeed, altitude, and pitch), this approach improves significantly on previously published results, while at the same time not requiring additional input nodes for ARIMA residuals. For the fourth parameter, roll, the strategy performs worse, however this may be due to the fact that the neural networks were not trained for long enough. The authors feel that the results provide a strong case for the use of ant colony optimization in the design of neural networks, given its ability to find novel and effective neural network topologies that can be easily trained (apart from the roll parameter which requires further study).

2 Predicting General Aviation Flight Data

General aviation comprises 63% of all civil aviation activity in the United States; covering operation of all non-scheduled and non-military aircraft [21, 32]. While general aviation is a valuable and lucrative industry, it has the highest accident rates within civil aviation [29]. For many years, the general aviation accident and fatality rates have hovered around 7 and 1.3 per 100,000 flight hours, respectively [1]. The general aviation community and its aircraft are very diverse, limiting the utility of the traditional flight data monitoring (FDM) approach used by commercial airlines.

The National General Aviation Flight Information Database (NGAFID) has been developed at the University of North Dakota as a central repository for general aviation flight data. It consists of per-second flight data recorder (FDR) data from three fleets of

aircraft. As of November 2014, the database stores FDR readings from over 200,000 flights, with more being added daily. It currently stores over 750 million per-second records of flight data. The NGAFID provides an invaluable source of information about general aviation flights, as most of these flights are from aviation students, where there is a wider variance in flight parameters than what may normally be expected within data from professionally piloted flights.

Having algorithms which can accurately predict FDR parameters would be able to not only warn pilots of problematic flight behavior, but also be used to predict impending failures of engines and other hardware. As such, investigating predictive strategies such as these has the potential to reduce costs for maintaining general aviation fleets, and more importantly save lives.

3 Previous Results

In previous work, the authors evaluated a suite of feed forward, Jordan and Elman recurrent neural networks to predict flight parameters [14]. This work was novel in that to our knowledge, neural networks have not been previously applied to predicting general aviation flight data. These results were encouraging in that some parameters such as altitude and airspeed can be predicted with high accuracy, at 0.22% - 0.62% for airspeed, 0.026-0.08% for airspeed, 0.88% - 1.49% for pitch and 0.5% to 2% for roll. These neural networks were trained using backpropagation via stochastic gradient descent, gradient descent from a baseline predictor (which mimicked how deep neural networks are currently trained by pre-training each layer to predict its input), and with asychronous differential evolution (ADE). ADE was shown to significantly outperform both types of backpropagation, provided solutions with up to 70% improvement. It was also shown that while ADE outperformed backpropagation, it still had trouble training the larger fully connected Jordan and Elman recurrent neural networks (which provided the best predictions), motivating further study.

4 Methodology

The ACO based strategy works as follows. Given a potentially fully connected recurrent neural network – where each node has a potential connection to every node in the subsequent layer and to a respective node in the recurrent layer – each connection between neurons can be seen as a potential path for an ant (see Figure 1). Every potential connection is initialized with a base amount of pheromone, and the master process stores the amount of pheromone on each connection. Worker processes receive neural network designs generated by taking a selected number of ants, and having them choose a path through the fully connected neural network biased by the amount of pheromone on each connection. Multiple ants can choose the connections between neurons. Those ant paths are be combined to construct a neural network design which is sent to worker processes and trained on the input flights using backpropagation, evolutionary algorithms or any other neural network training algorithm. The master process maintains a population of the best neural network designs, and when a worker reports the accuracy of a newly trained neural network, if it improves the population, the master process will increase

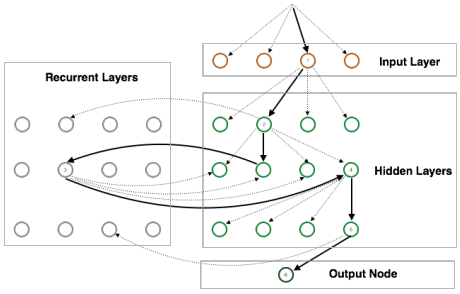

Fig. 1. Ants select a forward propagating path through neurons randomly based on the pheromone on each path, assuming a fully connected strategy between the input, hidden and output layers; and a potential connection from each hidden node to a respective node in the recurrent layer that is fully connected back to its hidden layer.

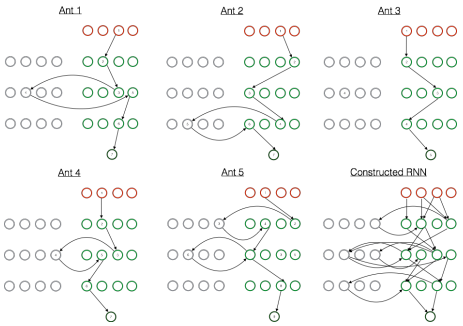

Fig. 2. The server creates neural networks for the workers to evaluate by combining the paths selected by a given number of ants. This generates various Elmann-like neural networks which have less training complexity than a fully connected Elman design.

the pheromone on every connection that was in that neural network. The master process periodically degrades pheromone levels, as is done in the standard ACO algorithm. This strategy allows the evolution of recurrent neural networks with potentially many hidden layers and hidden nodes, to determine what design can best predict flight parameters.

5 Results

5.1 Optimization Software, Data and Reproducibility

Given the complexity of examining complex neural networks over per-second flight data, a package requiring easy use of high performance computing resources was required. While there exist some standardized evolutionary algorithms packages [2, 8, 41, 25], as well as those found in the R programming language [28, 4] and MATLAB [26], they do not easily lend themselves towards use in high performance computing environments.

This work utilizes the Toolkit for Asynchronous Optimization (TAO), which is used by the MilkyWay@Home volunteer computing to perform massively distributed evolutionary algorithms on tens of thousands of volunteered hosts [15, 16, 12]. It is implemented in C++ and MPI, allowing easy use on clusters and supercomputers, and also provides support for systems with multiple graphical processing units. Further, TAO has shown that performing evolutionary algorithms asynchronously can provide significant improvements to performance and scalability over iterative approaches [39, 13]. TAO is open source and freely available on GitHub, allowing easy use and extensibility[3], and the presented ACO strategy has been included in that repository. The flight data used

[3] https://github.com/travisdesell/tao

in this work has also been made available online for reproducibility and use by other researchers[4].

5.2 Runtime Environment

All results were gathered using a Beowulf HPC cluster with 32 dual quad-core compute nodes (for a total of 256 processing cores). Each compute node has 64GBs of 1600MHz RAM, two mirrored RAID 146GB 15K RPM SAS drives, two quad-core E5-2643 Intel processors which operate at 3.3Ghz, and run the Red Hat Enterprise Linux (RHEL) 6.2 operating system. All 32 nodes within the cluster are linked by a private 56 gigabit (Gb) InfiniBand (IB) FDR 1-to-1 network. The code was compiled and run using MVAPICH2-x [22], to allow highly optimized use of this network infrastructure.

5.3 Data Cleansing

The flight data required some cleaning for use, as it is stored as raw data from the flight data recorders uploaded to the NGAFID server and entered in the database as per second data. When a FDR turns on, some of the sensors are still calibrating or not immediately online, so the first minute of flight data can have missing and erroneous values. These initial recordings were removed from the data the neural networks were trained on. Further, the parameters had wide ranges and different units, *e.g.*, pitch and roll were in degrees, altitude was in meters and airspeed was in knots. These were all normalized to values between 0 and 1 for altitude and airspeed, and -0.5 and 0.5 for pitch and roll.

5.4 Experiments

As backpropagation was shown to not be sufficient to train these recurrent neural networks, particle swarm optimization (PSO) was used to train the neural networks generated by ACO. Previous work has shown both particle swarm and differential evolution as being equally effective in training these networks. PSO used a population of 200, inertia weight of 0.75, and global and local best weights of 1.5 for all runs. PSO was allowed to train the neural networks for 250, 500 and 1000 iterations.

The ACO strategy was used to train networks with 3, 4, and 5 hidden layers (with a similar number of recurrent layers), using 4 and 8 nodes per layer and pheromone degradation rates of 10%, 5% and 1%. The number of ants used was equal to twice the number of nodes per layer (8 for 4 nodes per layer, and 16 for 8 nodes per layer). Each combination of settings was run 5 times for each of altitude, airspeed, pitch and roll, for a total of 1080 runs.

Each run was done allocating 64 processes across 8 nodes, and was allowed to train for 1000 evaluations of generated neural networks. Runs with 250 PSO iterations took around 30 minutes, 500 PSO iterations took around 1 hour, and 1000 PSO iterations took around 2 hours.

[4] http://people.cs.und.edu/∼tdesell/ngafid_releases.php

All runs were done on flight ID 13588 from the NGAFID data release. The neural networks were trained for individual output parameters, as tests have shown that trying to train for multiple output parameters simultaneously performs very poorly, with neither backpropagation or evolutionary strategies being able to find effective weights.

5.5 ACO Parameter Setting Analysis

Figure 3 presents the results of the parameter sweep. In general, there was a strong correlation between increased PSO iterations and the best fitnesses found. Across all runs, 4 nodes per layer performed the best, and apart from altitude, 5 hidden layers performed the best. There did not appear to be a strong trend for the pheromone degradation rate.

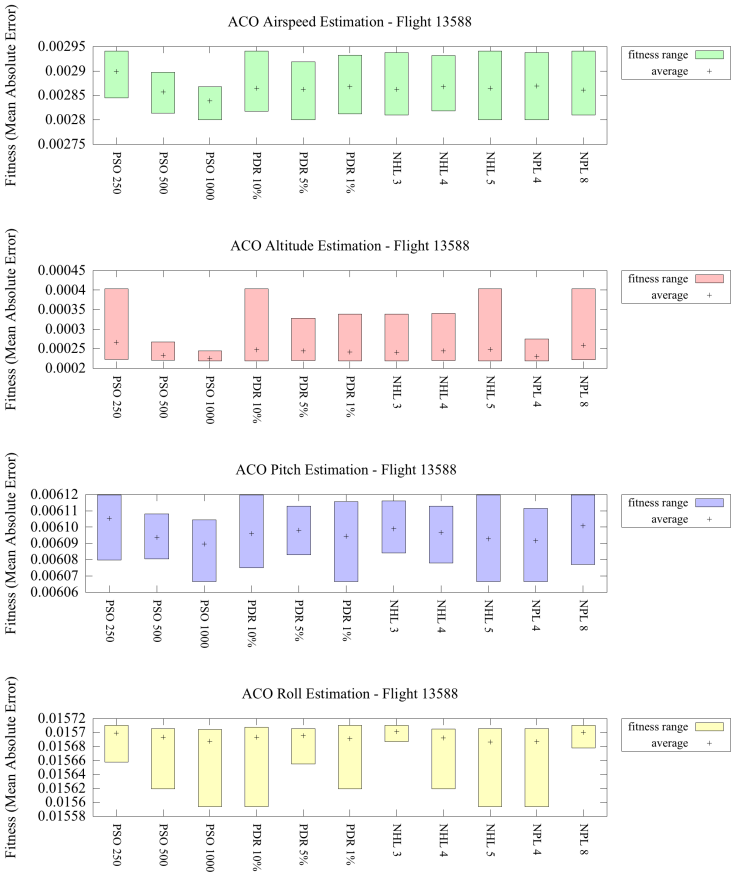

Fig. 3. Minimum, maximum and average fitness (mean average error) given the different ACO input parameters. Fitness values were averaged over each run with the parameter specified in the x-axis. Lower fitness is better. PSO is the number of PSO iterations, PDR is the pheromone degradation rate, NHL is the number of hidden layers, and NPL is the number of nodes per layer.

5.6 Best Found Neural Networks

Figure 4 displays the best recurrent neural networks evolved by the ACO strategy. For airspeed, pitch and roll, the best networks were the deepest – with 5 hidden and recurrent layers (although all nodes were not used). They also displayed interesting recurrent topologies, significantly different than the standard Jordan and Elman recurrent neural networks found in literature. The best evolved neural network for altitude was also interesting in that it completely ignored roll as an input parameter. The evolved networks also show some slight similarity to sparse autoencoders, with some of the middle layers being constrained to less nodes and connections.

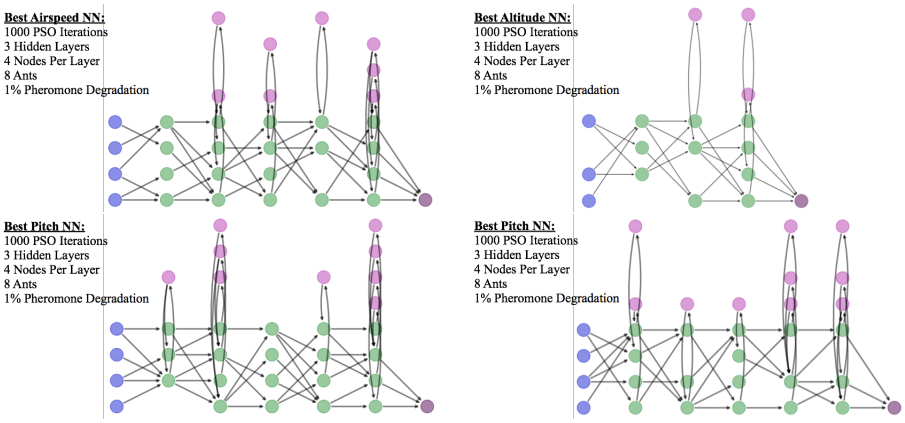

Fig. 4. The best found evolved neural networks across the 1080 runs performed. Input neurons are in blue, recurrent neurons are in pink, hidden neurons are in green, and the output neuron is in purple.

5.7 Comparison to Prior Results

The performance of the best evolved neural networks was compared to the previously best published results for flight ID 13588, which were an Elman network with 2 input lags and 1 hidden layer for airspeed; a Jordan recurrent neural network with 2 input lags and 0 hidden layers for altitude; an Elman network with 1 set put input lags and 1 hidden layer for pitch; and an Elman network with 2 input lags and 1 hidden layer for roll. In addition, results for a random noise estimator (RNE), which uses the previous value as the prediction for the next value, $prediction(t_{i+1}) = t_i$, were given as a baseline comparison, as it represents the best predictive power that can be achieved for random time series data. If the neural networks did not improve on this, then the results would have been meaningless and potentially indicate that the data is too noisy (given weather and other conditions) for prediction.

Additionally, the RNE provides a good baseline in that it is easy for neural networks to represent the RNE: all weights can be set to 0, except for a single path from the path

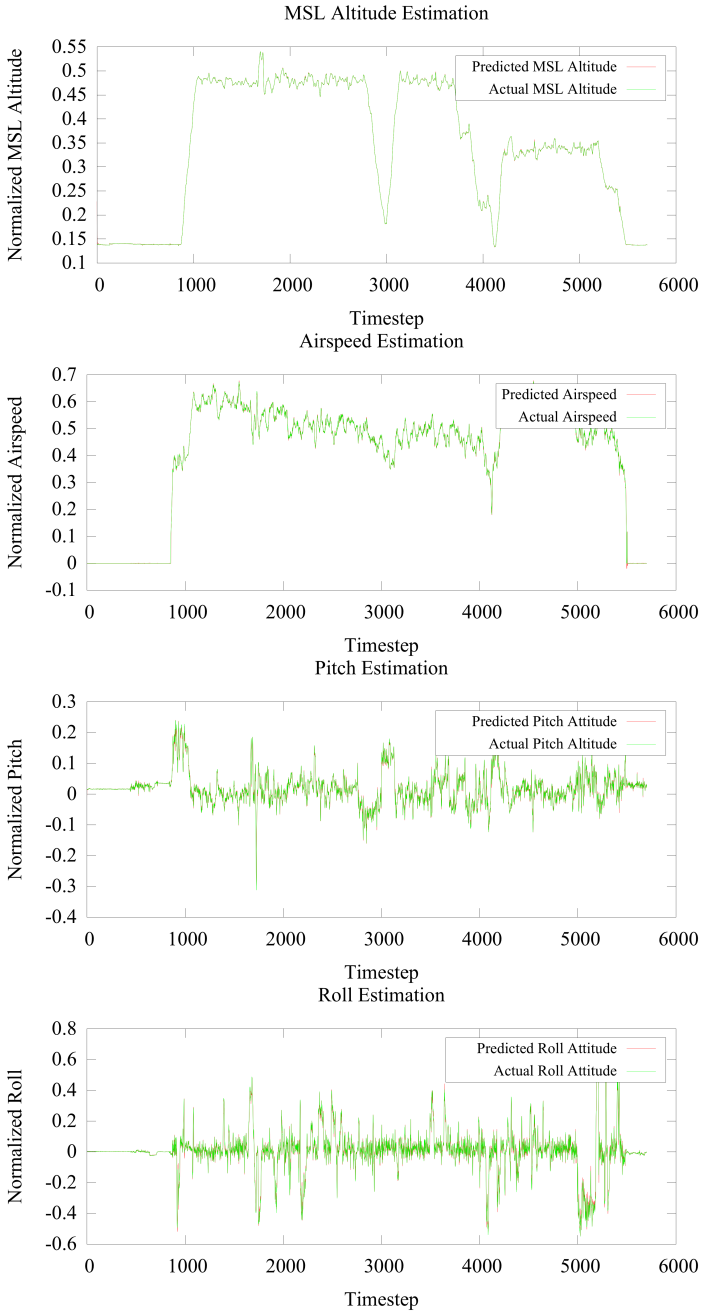

Fig. 5. The best neural networks trained on Flight #13588 were used to predict the parameters of Flight #17269. The actual values are in green and the predictions are in red. Altitude and airspeed were predicted with very high accuracy, however pitch and roll are more challenging. Time steps are in seconds, and parameters are normalized over a range of 1. Predicted and actual airspeed are indistinguishable at the scale of the figure and completely overlap.

from the corresponding input node to the output node having weights of 1. Because of this, it also provides a good test of the correctness of the global optimization techniques, at the very least they should be able to train a network as effective as a RNE; however local optimization techniques (such as backpropagation) may not reach this if the search area is non-convex and the initial starting point does not lead to a good minimum.

Airspeed

Method	13588	15438	17269	175755	24335
$t_{i+1} = t_i$	0.00512158	0.00316859	0.00675531	0.00508229	0.00575537
Prior Best	0.00472131	0.00250284	0.00656991	0.00465581	0.00495454
Best ACO	0.00279963	0.00145748	0.00433578	0.0028908	0.00305361

Altitude

Method	13588	15438	17269	175755	24335
$t_{i+1} = t_i$	0.00138854	0.00107117	0.00200011	0.00137109	0.00192345
Prior Best	0.000367535	0.000305193	0.000895711	0.000399587	0.000485329
Best ACO	0.0002183	0.000160932	0.000353502	0.000224827	0.000249197

Pitch

Method	13588	15438	17269	175755	24335
$t_{i+1} = t_i$	0.0153181	0.010955	0.0148046	0.0161251	0.0173269
Prior Best	0.014918	0.0100763	0.0147712	0.01514	0.0160249
Best ACO	0.00606664	0.00498241	0.00837594	0.005864	0.00733882

Roll

Method	13588	15438	17269	175755	24335
$t_{i+1} = t_i$	0.0158853	0.00604479	0.0204441	0.012877	0.0192648
Prior Best	0.0154541	0.00587058	0.0206536	0.0127999	0.0182611
Best ACO	0.0155934	0.00900393	0.0237235	0.0151416	0.0200261

Fig. 6. Comparison of the best found ACO evolved neural networks to the random noise estimator ($t_{i+1} = t_i$) and the previously published best found results. The mean average error for the neural networks trained on flight ID 13588 is given when they are tested on four other flights.

Figure 6 compares the best ACO results to the RNE and the previous best trained neural network for flight ID 13588. Results are the Mean Average Error (MAE) of the prediction to the actual value. As results were normalized over a range of 1, the MAE is also the percentage error. These neural networks and the RNE were also run on four other flights, IDs 15438, 17269, 175755 and 24335 from the NGAFID data release. On average compared to previous best results, the ACO evolved neural networks provided a 63% improvement over airspeed, a 97% improvement over altitude and a 120% improvement over pitch, *without requiring additional input neurons for lag values*. Given the fact that these neural networks also performed strongly on all test flights, these results are quite encouraging.

However, as in previous work, the roll parameter remains quite difficult to predict, and the ACO evolved neural networks actually resulted in a 14.5% decrease in prediction accuracy, performing worse than the RNE. Given the depth and complexity of the evolved neural networks, there is justifiable concern for over training, which may be

the case for this evolved network. Another reason for the poor performance of the ACO evolved neural networks may be due to the limited amount of training for each generated neural network. Previous results had the neural networks be trained for 15,000,000 objective function evaluations, while the best performing ACO evolved neural networks were trained with a maximum of 200,000 objective function evaluations (1000 iterations with population size 200). Given the strong correlation between increased PSO iterations and best fitness found for roll, it is also possible that the neural networks were not trained long enough for the roll parameter. Lastly, it could be that even though the input lag nodes were not required for the other parameters, they may be required for roll, or stand to provide even further prediction improvements. A further study of this stands for future work.

6 Conclusions and Future Work

This paper presents and analyzes a novel strategy for using ant colony optimization for evolving the structure of recurrent neural networks. The strategy presented is used to evolve deep neural networks with up to 5 hidden and 5 recurrent layers for the challenging task of predicting general aviation flight data, and is shown to provide improvements of 63% for airspeed, a 97% for altitude and 120% for pitch over previously best published results, *while at the same time not requiring additional input neurons for residual values*. Finding good predictions for the roll parameter still remains challenging and an area of future study.

Further, this work opens up interesting opportunites in applying ant colony optimization to neuro evolution. In particular, the authors feel that the approach could be extended to evolve neural networks for computer vision, by allowing ants to also select what type of activation function each neuron has (*e.g.*, ReLU, or max pooling). It may also be possible to utilize this strategy to further improve convolutional layers in neural networks. Additionally, this work only tested neural networks with a recurrent depth of one, where each recurrent node is immediately fed back into the neural network in the next iteration. It may be possible to use this strategy to generate neural networks with deeper memory, where recurrent nodes can potentially feed back into a deeper layer of recurrent nodes, and so on.

Finally, the National General Aviation Flight Database (NGAFID) provides an excellent data source for researching evolutionary algorithms, machine learning and data mining. Further analysis of these flights along with more advanced prediction methods will enable more advanced flight sensors, which could prevent accidents and save lives; which is especially important in the field of general aviation as it is has the highest accident rates within civil aviation [29]. As many of these flights also contain per-second data of various engine parameters, using similar predictive methods it may become possible to detect engine and other hardware failures, aiding in the maintenance process. This work presents a further step towards making general aviation safer through machine learning and evolutionary algorithms.

References

1. Aircraft Owners and Pilots Association (AOPA), January 2014.

2. M. Arenas, P. Collet, A. Eiben, M. Jelasity, J. Merelo, B. Paechter, M. Preuß, and M. Schoenauer. A framework for distributed evolutionary algorithms. *Parallel Problem Solving from Nature-PPSN VII*, pages 665–675, 2002.
3. R. Ashena and J. Moghadasi. Bottom hole pressure estimation using evolved neural networks by real coded ant colony optimization and genetic algorithm. *Journal of Petroleum Science and Engineering*, 77(3):375–385, 2011.
4. T. Bartz-Beielstein. SPOT: An R package for automatic and interactive tuning of optimization algorithms by sequential parameter optimization. *arXiv preprint arXiv:1006.4645*, 2010.
5. G. Bilchev and I. C. Parmee. The ant colony metaphor for searching continuous design spaces. In *Evolutionary Computing*, pages 25–39. Springer, 1995.
6. C. Blum and X. Li. *Swarm intelligence in optimization*. Springer, 2008.
7. C. Blum and K. Socha. Training feed-forward neural networks with ant colony optimization: An application to pattern classification. In *Hybrid Intelligent Systems, 2005. HIS'05. Fifth International Conference on*, pages 6–pp. IEEE, 2005.
8. S. Cahon, N. Melab, and E.-G. Talbi. Paradiseo: A framework for the reusable design of parallel and distributed metaheuristics. *Journal of Heuristics*, 10(3):357–380, 2004.
9. R. Chandra. Competitive two-island cooperative coevolution for training elman recurrent networks for time series prediction. In *Neural Networks (IJCNN), 2014 International Joint Conference on*, pages 565–572, July 2014.
10. R. Chandra and M. Zhang. Cooperative coevolution of elman recurrent neural networks for chaotic time series prediction. *Neurocomputing*, 86:116–123, 2012.
11. S. F. Crone, M. Hibon, and K. Nikolopoulos. Advances in forecasting with neural networks? Empirical evidence from the NN3 competition on time series prediction. *International Journal of Forecasting*, 27(3):635–660, 2011.
12. T. Desell. *Asynchronous Global Optimization for Massive Scale Computing*. PhD thesis, Rensselaer Polytechnic Institute, 2009.
13. T. Desell, D. Anderson, M. Magdon-Ismail, B. S. Heidi Newberg, and C. Varela. An analysis of massively distributed evolutionary algorithms. In *The 2010 IEEE congress on evolutionary computation (IEEE CEC 2010)*, Barcelona, Spain, July 2010.
14. T. Desell, S. Clachar, J. Higgins, and B. Wild. Evolving neural network weights for time-series prediction of general aviation flight data. In T. Bartz-Beielstein, J. Branke, B. Filipi, and J. Smith, editors, *Parallel Problem Solving from Nature PPSN XIII*, volume 8672 of *Lecture Notes in Computer Science*, pages 771–781. Springer International Publishing, 2014.
15. T. Desell, B. Szymanski, and C. Varela. Asynchronous genetic search for scientific modeling on large-scale heterogeneous environments. In *17th International Heterogeneity in Computing Workshop*, Miami, Florida, April 2008.
16. T. Desell, C. Varela, and B. Szymanski. An asynchronous hybrid genetic-simplex search for modeling the Milky Way galaxy using volunteer computing. In *Genetic and Evolutionary Computation Conference (GECCO)*, Atlanta, Georgia, July 2008.
17. M. Dorigo and M. Birattari. Ant colony optimization. In *Encyclopedia of Machine Learning*, pages 36–39. Springer, 2010.
18. M. Dorigo and L. M. Gambardella. Ant colonies for the travelling salesman problem. *BioSystems*, 43(2):73–81, 1997.
19. M. Dorigo and T. Stützle. Ant colony optimization: overview and recent advances. In *Handbook of metaheuristics*, pages 227–263. Springer, 2010.
20. J. Dréo and P. Siarry. A new ant colony algorithm using the heterarchical concept aimed at optimization of multiminima continuous functions. In *Ant algorithms*, pages 216–221. Springer, 2002.
21. B. Elias. *Securing general aviation*. DIANE Publishing, 2009.

22. W. Huang, G. Santhanaraman, H.-W. Jin, Q. Gao, and D. K. Panda. Design of high performance MVAPICH2: MPI2 over InfiniBand. In *Cluster Computing and the Grid, 2006. CCGRID 06. Sixth IEEE International Symposium on*, volume 1, pages 43–48. IEEE, 2006.
23. M. Khashei and M. Bijari. A novel hybridization of artificial neural networks and arima models for time series forecasting. *Applied Soft Computing*, 11(2):2664–2675, 2011.
24. J.-B. Li and Y.-K. Chung. A novel back-propagation neural network training algorithm designed by an ant colony optimization. In *Transmission and Distribution Conference and Exhibition: Asia and Pacific, 2005 IEEE/PES*, pages 1–5. IEEE, 2005.
25. M. Lukasiewycz, M. Glaß, F. Reimann, and J. Teich. Opt4j: a modular framework for meta-heuristic optimization. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, GECCO '11, pages 1723–1730, New York, NY, USA, 2011. ACM.
26. MathWorks. Global optimization toolbox. Accessed Online: March 2013.
27. N. Monmarché, G. Venturini, and M. Slimane. On how¡ i¿ pachycondyla apicalis¡/i¿ ants suggest a new search algorithm. *Future Generation Computer Systems*, 16(8):937–946, 2000.
28. K. Mullen, D. Ardia, D. Gil, D. Windover, and J. Cline. Deoptim: An r package for global optimization by differential evolution. *Journal of Statistical Software*, 40(6):1–26, 2011.
29. National Transportation Safety Board (NTSB), 2012.
30. D. Ömer Faruk. A hybrid neural network and arima model for water quality time series prediction. *Engineering Applications of Artificial Intelligence*, 23(4):586–594, 2010.
31. A. Pandian. *Training neural networks with ant colony optimization*. PhD thesis, California State University, Sacramento, 2013.
32. K. I. Shetty. *Current and historical trends in general aviation in the United States*. PhD thesis, Massachusetts Institute of Technology Cambridge, MA 02139 USA, 2012.
33. R. K. Sivagaminathan and S. Ramakrishnan. A hybrid approach for feature subset selection using neural networks and ant colony optimization. *Expert systems with applications*, 33(1):49–60, 2007.
34. K. Socha. Aco for continuous and mixed-variable optimization. In *Ant colony optimization and swarm intelligence*, pages 25–36. Springer, 2004.
35. K. Socha. *Ant Colony Optimisation for Continuous and Mixed-variable Domains*. VDM Publishing, 2009.
36. K. Socha and M. Dorigo. Ant colony optimization for continuous domains. *European journal of operational research*, 185(3):1155–1173, 2008.
37. K. Stanley and R. Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary computation*, 10(2):99–127, 2002.
38. K. O. Stanley, D. B. D'Ambrosio, and J. Gauci. A hypercube-based encoding for evolving large-scale neural networks. *Artificial life*, 15(2):185–212, 2009.
39. B. Szymanski, T. Desell, and C. Varela. The effect of heterogeneity on asynchronous panmictic genetic search. In *Proc. of the Seventh International Conference on Parallel Processing and Applied Mathematics (PPAM'2007)*, LNCS, Gdansk, Poland, September 2007.
40. M. Unal, M. Onat, and A. Bal. Cellular neural network training by ant colony optimization algorithm. In *Signal Processing and Communications Applications Conference (SIU), 2010 IEEE 18th*, pages 471–474. IEEE, 2010.
41. S. Ventura, C. Romero, A. Zafra, J. A. Delgado, and C. Hervás. Jclec: a java framework for evolutionary computation. *Soft Computing-A Fusion of Foundations, Methodologies and Applications*, 12(4):381–392, 2008.
42. W. W.-S. Wei. *Time series analysis*. Addison-Wesley Redwood City, California, 1994.
43. G. P. Zhang. Neural networks for time-series forecasting. In *Handbook of Natural Computing*, pages 461–477. Springer, 2012.