ONE-NAS: An Online NeuroEvolution based Neural Architecture Search for Time Series Forecasting

Zimeng Lyu
zimenglyu@mail.rit.edu
Rochester Institute of Technology
Rochester, New York, USA

Travis Desell
tjdvse@rit.edu
Rochester Institute of Technology
Rochester, New York, USA

ABSTRACT

Time series forecasting (TSF) is one of the most important tasks in data science, as accurate time series (TS) predictions can drive and advance a wide variety of domains including finance, transportation, health care, and power systems. However, real-world utilization of machine learning (ML) models for TSF suffers due to data drift. To address this, models must be periodically retained or redesigned, which requires significant human and computational resources. This work presents the Online NeuroEvolution based Neural Architecture Search (ONE-NAS) algorithm, which to the authors' knowledge is the first neural architecture search algorithm capable of automatically designing and training new recurrent neural networks (RNNs) in an online setting. Without any pretraining, ONE-NAS utilizes populations of RNNs which are continuously updated with new network structures and weights in response to new multivariate input data. ONE-NAS is tested on real-world large-scale multivariate wind turbine data and is shown to outperform traditional statistical time series forecasting, including naive, moving average, and exponential smoothing methods.

CCS CONCEPTS

• **Theory of computation → Online algorithms**; • **Computing methodologies → Neural networks**; • **Applied computing → Forecasting**.

KEYWORDS

NeuroEvolution, Online Algorithms, Time Series Forecasting, Recurrent Neural Networks, Neural Architecture Search

ACM Reference Format:
Zimeng Lyu and Travis Desell. 2022. ONE-NAS: An Online NeuroEvolution based Neural Architecture Search for Time Series Forecasting. In *Genetic and Evolutionary Computation Conference Companion (GECCO '22 Companion), July 9–13, 2022, Boston, MA, USA.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3520304.3528962

1 INTRODUCTION

Time series forecasting (TSF) [6] is one of the most important tasks in data science, as accurate time series (TS) predictions can drive and advance a wide variety of domains including finance [5], transportation [9], health care [1], and power systems [17]. A major problem with TSF is that models always suffer from data drift, due to the fact that TSF models trained on historical data suffer as the data distribution changes over time [10] [8]. Due to this, the models trained offline needs retrained on newer representative data, and this repeated process is time consuming especially when handling multi-variate large-scale time series data.

Online learning algorithms mitigate the problem of data drift by continually training models as new data arrives [20] [12], however this results in two major challenges: *i)* online learning scenarios typically assume no historical data is available, so these models are not pre-trained for multiple epochs and must start from scratch – making it an extremely challenging learning problem; and *ii)* while continual updating of models as new data arrives can address data drift, models still must overcome catastrophic forgetting, where new sequences of data can cause the model to forget common but not recent patterns, significantly reducing performance.

Online time series forecasting strategies which incorporate neuroevolution have the potential to overcome these challenges. Many neuroevolution algorithms, such as those related to the popular NeuroEvolution of Augmenting Topologies (NEAT) algorithm [19], start with seed genomes with a minimal structure. During the evolutionary process, networks grow to gradually adapt to the complexity of the dataset, which reduces the effort required to hand tune and design the networks. Architectures can be updated and trained as new data arrives, while populations can retain older networks which can contain valuable historical information to reduce or prevent catastrophic forgetting.

This work presents a novel *online* NeuroEvolution (NE) Neural Architecture Search (NAS) method that evolves Recurrent Neural Networks (RNNs) for time series data prediction (ONE-NAS). To authors' knowledge this is the first algorithm capable of the online evolution of RNNs for time series data predictions. It provides numerous benefits over traditional fixed RNN architectures and even other modern NE strategies, allowing the RNN architectures to be continually updated in response to new input data, as well as gracefully handling the effect of unpredictable events.

2 METHODOLOGY

2.1 ONE-NAS

Figure 1 presents a high level view of the asynchronous, distributed, online ONE-NAS algorithm. This paper evaluates two versions, one which utilizes a single island, and the second which utilizes multiple islands and a repopulation strategy. ONE-NAS concurrently evolves and trains new RNNs while performing online time series data prediction. Unlike offline methods, *ONE-NAS does not require pre-training on any data before the online NE process.*

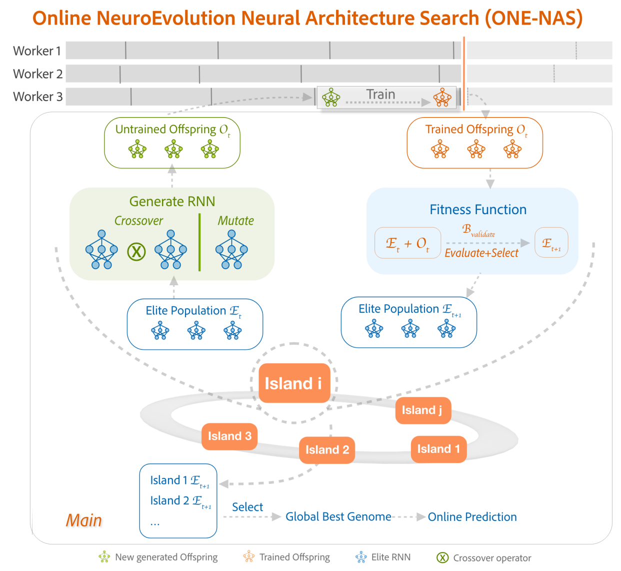

Figure 1: A representation of one generation of the ONE-NAS online neuroevolution process.

ONE-NAS is built utilizing components of the Evolutionary eXploration of Augmenting Memory Models (EXAMM) offline neuroevolution framework [16], such as its mutation and crossover operations, however it expands on EXAMM by: *i)*, allowing for online prediction using a genome while concurrently evolving and training a new population of genomes, *ii)*, collecting incoming streaming time series data and saving it to incorporate it in future genome evolution and training, *iii)*, generating and evaluating new genomes for every generations to quickly prevent the data drift problem, and *iv)*, utilizing elite populations for retention across batches of new streaming data.

ONE-NAS's online NE process is initialized with a minimal seed genome (an RNN which consists of online input nodes fully connected to output nodes, with no hidden layers). It progressively evolves genomes of generation t by:

- Selecting a set of n best genomes as an elite population E_t.
- Generating offspring O_t from E_t through mutation and crossover.
- Selecting on E_t and adding them to the next generation.
- Training all the offspring of E_t with backpropagation.
- Evaluating all genomes in generation t on most recent data.

ONE-NAS is asynchronous, with generated genomes for each generation being trained by worker processes using a work stealing strategy [4] where each worker independently requests more genomes for training when it finishes training its previous requested genome. This strategy is naturally load balanced as workers do not block on any other worker for training, and scales up to a number of workers equal to the population size [7]. While the evolution process is being performed, the best genome from the previous generation is utilized to perform online prediction of the time series.

Each generation lasts for a specified period of time steps, p (in this work, $p = 25$), which provides a subsequence of time series data. The best genome from the previous generation performs online predictions of the new subsequence (B_{next}) as it arrives, while concurrently the new generation of genomes is generated and trained. At the end of a generation, this new subsequence of data is added to ONE-NAS's historical training data.

During a generation, the generated genomes O_t are trained on a randomly selected set of B_{train} subsequences of historical training data. Then the entire population (including elite E_t) has their fitness calculated as the mean squared error loss over $B_{validation}$, which is then used to select the next elite population E_{t+1}. The best genome in E_{t+1} is used for online prediction for current generation. Because of this, while O_t are trained using backpropagation on batches drawn from the historical data, the RNNs in E_t do not continue to be trained. As not all RNNs in E_t will preform better than those in O_t, "obsolete" RNNs will naturally die off over time, however RNNs with strong performance will remain.

2.2 ONE-NAS Island Repopulation Strategy

We extended ONE-NAS to utilize islands which are periodically repopulated after extinction events, due to success in prior work [13]. In this version, ONE-NAS utilizes m islands, each of which have their own set of elite E and generated O genomes. Each island evolves similarly as in ONE-NAS, except crossover now has two forms, *intra-island crossover*, where both parents are selected from the elite population in the same island, and *inter-island crossover*, where when generating a new child for an island, one parent is selected from that island's elite population, and the other is the best genome from a randomly selected different island. To prevent islands getting stuck in local optima, we utilize island extinction and repopulation, which periodically selects the worst performing island and removes all its genomes, replacing them with mutations of the global best genome from the search at that time. ONE-NAS's makes a modification on this island repopulation strategy, where after the repopulating island is erased, only the elite population E_t of the island is filled with mutations of the global best genome, while the generated population O_t is empty. At next generation $t+1$, O_{t+1} is filled with child genomes generated by the repopulated E_t. This strategy was shown to provide significant performance improvements to ONE-NAS.

3 EXPERIMENTAL DESIGN

Prior work has shown that utilizing shorter subsequences of time series data during training can improving an RNN's convergence rate and overall performance [14]. Due to this, for the experiments in this work, the original datasets were divided into subsequences of 25 timesteps each. During each ONE-NAS generation, each newly generated genome was trained on 600 randomly selected subsequences from the historical data pool and then validated using the most recent 100 subsequences of historical data. There was no overlap between training and validation data (the most recent 100 subsequences are added into the historical pool after being used for validation). All experiments were run for 2000 generations, which represents a single pass over the entire wind data time series.

During each generation, 50 elite genomes from the previous generation were retained, and the elite genomes were used to generate 100 new genomes using a mutation rate of 0.4 and crossover rate of 0.6. Each of the 100 non-elite genomes in the new generation were

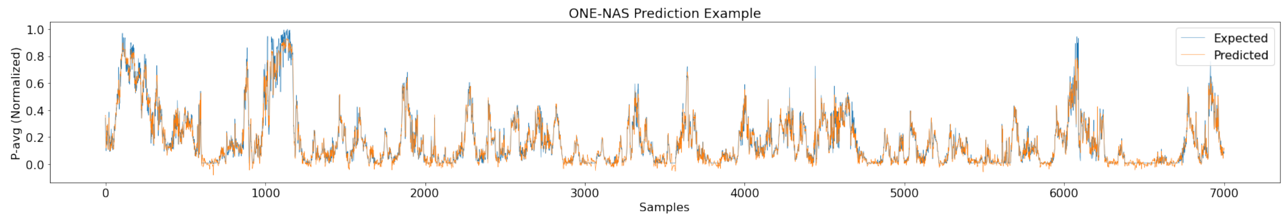

Figure 2: The *average active power* parameter from the wind dataset used in this work, as well as example of ONE-NAS's predictions on this dataset.

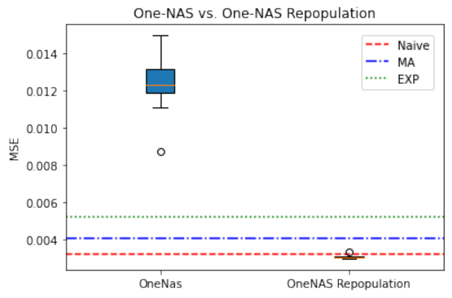

Figure 3: Online prediction MSE of ONE-NAS and One-NAS island repopulation

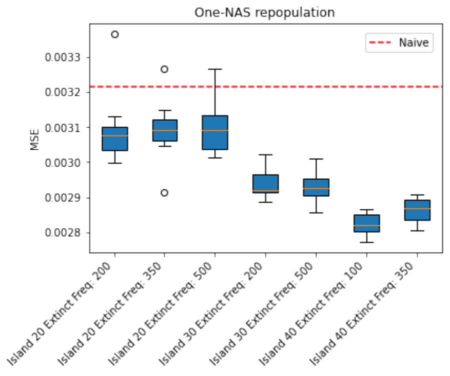

Figure 4: ONE-NAS evolution process

trained in a worker process for 10 backpropagation epochs, with the first 5 epoch trained on the original subsequence data, and then in each of the last 5 epochs, 10% Gaussian noise was added with mean and standard deviation of the sliced data as an augmentation technique to help prevent over fitting [2, 3]. ONE-NAS with Island Repopulation utilized 10, 20, 30 or 40 islands, with each having its own elite population of 5 genomes which generated an additional 10 genomes per generation. New genomes were generated with a mutation rate of 0.3, inter-island crossover rate of 0.4, and intra-island crossover rate of 0.3. These parameters were hand-tuned and selected for this work.

For genome generation, 10 out of EXAMM's 11 mutation operations were utilized (all except for *split edge*), and each was chosen with a uniform 10% chance. ONE-NAS generated new nodes by selecting from EXAMM's library of simple neurons, Δ-RNN, GRU, LSTM, MGU, and UGRNN memory cells uniformly at random. Recurrent connections could span any time-skip generated randomly between $\mathcal{U}(1, 10)$. Backpropagation (BP) through time was run with a learning rate of $\eta = 0.001$ and used Nesterov momentum with $\mu = 0.9$. For the memory cells with forget gates, the forget gate bias had a value of 1.0 added to it (motivated by [11]). Gradient scaling [18] and gradient boosting was used to prevent gradient exploding and vanishing.

4 RESULTS

This work utilized wind turbine engine data collected and made available by ENGIE's La Haute Borne open data windfarm[1] which was gathered between 2017 and 2020. This wind dataset is very long, multivariate (with 22 parameters), non-seasonal, and the parameter recordings are not independent. The wind turbine data consists of readings every 10 minutes from 2013 to 2020. *Average Active Power* was selected as output parameter to forecast for the wind turbine data set. Figure 2 provides an example of the noisiness and complexity of the output parameter, as well as an example of ONE-NAS's accurate predictions on this data.

Each experiment was repeated 10 times using Rochester Institute of Technology's research computing systems. This system consists of 2304 Intel® Xeon® Gold 6150 CPU 2.70GHz cores and 24 TB RAM, with compute nodes running the RedHat Enterprise Linux 7 system. Each experiment utilized 16 cores.

[1]https://opendata-renewables.engie.com

4.1 Comparison to Classical TSF Methods

To test the performance of ONE-NAS, it was first compared to classical TSF methods: naive prediction, moving average prediction, and simple exponential smoothing. These methods were selected because they are easily capable of online prediction, whereas other methods require the entire dataset *a priori*. Additionally, for online TSF, research has shown simple classical methods, such as linear methods and exponential smoothing, can outperform complex and sophisticated methods [15].

Naive prediction simply uses the parameter's previous value, x_{t-1} as the predicted value, $\hat{y_t}$, for the next time step: $\hat{y_t} = x_{t-1}$. Moving average prediction uses the average of the last n time steps as the predicted for the next time step, where n is the moving average data smoothing window (a hyperparameter): $\hat{y_t} = 1/n * \sum_{n=1}^{n} x_{t-n}$. Simple Exponential smoothing (Holt linear), computes a running average of the previously seen parameters, where α is the smoothing factor, and $0 < \alpha < 1$: $\hat{y}_t = \alpha * x_{t-1} + (1 - \alpha) * \hat{y}_{t-1}$.

Figure 3 shows a box and whiskers plots for the the online prediction mean squared error (MSE) of ONE-NAS and ONE-NAS Repopulation algorithms over 10 repeated runs, alongside lines for each of the three classical time series forecasting methods (these methods are not randomized so their performance is always the same). Moving average (MA) prediction was done with a window size $n = 3$ and exponential smoothing was done with an $\alpha = 0.2$. The ONE-NAS Repopulation strategy shown in this plot used 20 islands (each with 5 elite genomes and 10 others per generation), with an extinction and repopulation frequency of 200 generations.

4.2 One-NAS Repopulation

With these results as encouragement, that it is possible to effectively evolve and train RNNs in an online setting, we found that two hyperparameters significantly affect online prediction performance: island size and the extinction/repopulation frequency.

Figure 4 shows a box plot of the online prediction MSE using island sizes of 20, 30, and 40 over 10 repeated experiments with varying repopulation frequencies. As the number of islands increase, the prediction performance improves. Additionally, more frequent island repopulation also shows improvements in prediction performance, although not as significant. More islands allow more variety of species which gives the algorithm more chances to get out of the local optima, and also potentially provides more robustness to noise and overfitting of the data.

More frequent extinction and repopulation events has better performance on average. Given that the number of generations is fixed at 2000 due to the length of the wind time series, an extinction frequency of 200 means at every 200 generations, the worst performing island is erased and repopulated, which results in 10 extinction and repopulation events. Once an island is erased, it is then repopulated with mutations of the current global best genome to add variety to the island. So the number of islands and total number of generations needs to be taken into consideration when choosing extinction and repopulation frequencies.

5 CONCLUSION

This work presents our novel Online NeuroEvolution based Neural Architecture Search (ONE-NAS) algorithm and applies it to time series forecasting (TSF) on a challenging real world wind turbine dataset. To the authors knowledge, ONE-NAS is the first neural architecture search algorithm capable of designing and training recurrent neural networks in real time as data arrives in an online scenario. We show that ONE-NAS outperforms traditional online statistical time series forecasting methods such as naive, moving average and exponential smoothing. ONE-NAS uses a novel strategy where generations of RNNs are evolved for TSF while concurrently the best RNN from the previous generation is used for generating predictions.

One major advancement which allowed ONE-NAS to be capable of accurate online predictions was the use of multiple islands of populations, where at specific frequencies the worst performing islands are deleted and repopulated with mutations of the global best genome (RNN). While the single population ONE-NAS had poor performance, we found that increasing the number of islands provided significant increases in performance, and also that increasing the extinction frequency also improved performance, albiet less dramatically. This strategy helps increase the diversity of the RNNs across islands, allowing for a more robust population which can also break out of local optima. ONE-NAS eliminates the need to perform offline retraining or redesigning of RNNs for new time series data, and shows good results on a real world, noisy dataset which includes periods of time where various sensors are offline.

REFERENCES

[1] Sunil Bhatnagar, Vivek Lal, Shiv D Gupta, Om P Gupta, et al. 2012. Forecasting incidence of dengue in Rajasthan, using time series analyses. *Indian journal of public health* 56, 4 (2012), 281.

[2] Chris M Bishop. 1995. Training with noise is equivalent to Tikhonov regularization. *Neural computation* 7, 1 (1995), 108–116.

[3] Christopher M Bishop et al. 1995. *Neural networks for pattern recognition.* Oxford university press.

[4] R. D. Blumofe and C. E. Leiserson. 1994. Scheduling Multithreaded Computations by Work Stealing. In *Proceedings of the 35th Annual Symposium on Foundations of Computer Science (FOCS '94).* Santa Fe, New Mexico, 356–368.

[5] Jian Cao, Zhi Li, and Jian Li. 2019. Financial time series forecasting model based on CEEMDAN and LSTM. *Physica A: Statistical Mechanics and its Applications* 519 (2019), 127–139.

[6] Chris Chatfield. 2000. *Time-series forecasting.* Chapman and Hall/CRC.

[7] Travis Desell, Dave Anderson, Malik Magdon-Ismail, Boleslaw Szymanski Heidi Newberg, and Carlos Varela. 2010. An Analysis of Massively Distributed Evolutionary Algorithms. In *The 2010 IEEE Congress on Evolutionary Computation (IEEE CEC 2010).* Barcelona, Spain.

[8] Tonya Fields, George Hsieh, and Jules Chenou. 2019. Mitigating drift in time series data with noise augmentation. In *2019 International Conference on Computational Science and Computational Intelligence (CSCI).* IEEE, 227–230.

[9] Bidisha Ghosh, Biswajit Basu, and Margaret O'Mahony. 2005. Time-series modelling for forecasting vehicular traffic flow in Dublin. In *84th Annual Meeting of the Transportation Research Board, Washington, DC.*

[10] Tian Guo, Zhao Xu, Xin Yao, Haifeng Chen, Karl Aberer, and Koichi Funaya. 2016. Robust online time series prediction with recurrent neural networks. In *2016 IEEE International Conference on Data Science and Advanced Analytics (DSAA).* Ieee, 816–825.

[11] Rafal Jozefowicz, Wojciech Zaremba, and Ilya Sutskever. 2015. An empirical exploration of recurrent network architectures. In *International Conference on Machine Learning.* 2342–2350.

[12] Jesus L Lobo, Ibai Laña, Javier Del Ser, Miren Nekane Bilbao, and Nikola Kasabov. 2018. Evolving spiking neural networks for online learning over drifting data streams. *Neural Networks* 108 (2018), 1–19.

[13] Zimeng Lyu, Joshua Karnas, AbdElRahman ElSaid, Mohamed Mkaouer, and Travis Desell. 2021. Improving Distributed Neuroevolution Using Island Extinction and Repopulation. *The 24th International Conference on the Applications of Evolutionary Computation (EvoStar: EvoApps)* (2021).

[14] Zimeng Lyu, Shuchita Patwardhan, David Stadem, James Langfeld, Steve Benson, Seth Thoelke, and Travis Desell. 2021. Neuroevolution of recurrent neural networks for time series forecasting of coal-fired power plant operating parameters. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion.* 1735–1743.

[15] Spyros Makridakis, Evangelos Spiliotis, and Vassilios Assimakopoulos. 2018. Statistical and Machine Learning forecasting methods: Concerns and ways forward. *PloS one* 13, 3 (2018), e0194889.

[16] Alexander Ororbia, AbdElRahman ElSaid, and Travis Desell. 2019. Investigating Recurrent Neural Network Memory Structures Using Neuro-evolution. In *Proceedings of the Genetic and Evolutionary Computation Conference* (Prague, Czech Republic) *(GECCO '19).* ACM, New York, NY, USA, 446–455. https://doi.org/10.1145/3321707.3321795

[17] EN Osegi. 2020. Using the hierarchical temporal memory spatial pooler for short-term forecasting of electrical load time series. *Applied Computing and Informatics* (2020).

[18] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. 2013. On the difficulty of training recurrent neural networks. In *International Conference on Machine Learning.* 1310–1318.

[19] Kenneth Stanley and Risto Miikkulainen. 2002. Evolving neural networks through augmenting topologies. *Evolutionary computation* 10, 2 (2002), 99–127.

[20] Shuo Wang, Leandro L Minku, and Xin Yao. 2018. A systematic study of online class imbalance learning with concept drift. *IEEE transactions on neural networks and learning systems* 29, 10 (2018), 4802–4821.