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

Zimeng Lyu  
Rochester Institute of Technology  
Rochester, New York, USA

Travis Desell  
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 pretrained models being able to learn and adapt to unpredictable patterns as previously unseen data arrives over longer time scales. To address this, models must be periodically retained or redesigned, which takes 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 as well as a univariate Dow Jones Industrial Average (DJIA) dataset, and is shown to outperform traditional statistical time series forecasting, including naive, moving average, and exponential smoothing methods, as well as state of the art online ARIMA strategies.

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

1 INTRODUCTION

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 [10, 42], transportation [17, 44], health care [6, 47], and power systems [3, 34]. A major problem in TSF is that models often do not perform well over long time periods due to changing distributions between their original training data and new inference data, as the properties or important features of the data change over time [18]. Due to this, these models can only be used for predictions for a reasonable period of time into the future, and then need to be retrained or redesigned as more recent data arrives so that the model can learn previously unseen patterns. This repeated process is time consuming especially as the architectures of artificial neural networks (ANNs) become more and more complicated to handle multi-variate large-scale time series data.

While there has been significant research into TSF where models are first trained offline [23, 28, 41], there is significantly less research into online TSF, where predictions must be made in real time and models are pretrained on prior data, making the problem significantly more challenging. Recent strategies for online TSF include variations on online ARIMA [4, 24] for univariate time series forecasting, or for online time series forecasting with machine learning [18]. Online ARIMA and ARMA models are mostly limited to univariate datasets, which have limitations to most real-world TSF applications which are multi-variate. For multivariate TSF, recurrent neural networks (RNNs) can be used, however finding the right depth or capacity of the RNN is challenging, requiring significant effort to avoid vanishing gradients (when models are too complex) and not being able to learn complex patterns (when models are too simple) [36]. Additionally, online learning algorithms also need to deal with the problem of catastrophic forgetting [1, 30], where important historical information can be forgotten over time.

Online time series forecasting strategies which incorporate neuroevolution, on the other hand, can tackle those challenges. Many neuroevolution algorithms, such as those related to the popular NeuroEvolution of Augmenting Topologies (NEAT) algorithm [38], 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.

Therefore, we propose 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. It is able to more quickly be trained in response to incoming data streams, as opposed to alternative methods that require significant offline time training using previously gathered training data sets. As historical time series data can have important temporal information, we train

1Offline strategies suffer the reverse problem, as described above. They fail to learn from new data, and eventually the distributions and features that the models were trained on do not represent the data they are performing inference on.
the child genomes with randomly selected historical data, so different historical temporal information is saved in different children. Eventually the genomes with useful weights and structures survive and others gradually die off.

Our results evaluate ONE-NAS against traditional statistical online TSF methods, on highly challenging, noisy, real world multivariate time series data from wind turbine sensors, as well as against modern online ARIMA based methods on univariate Dow Jones Industrial Average (DJIA) data, and show significant improvements in accuracy over these methods. Further, ONE-NAS utilizes a distributed, scalable algorithm and is shown to be able to operate efficiently in real time over short time scales. We also show that by utilizing multiple islands which are periodically repopulated to prevent stagnation in ONE-NAS, we significantly improve performance of the algorithm.

2 RELATED WORK

Most online time series forecasting models use fixed mathematical models or fixed NN topologies. These models adapt to different datasets by training and updating the weights as new data arrives. For statistical approaches, variations on online Autoregressive Integrated Moving Average (ARIMA) models have been proposed for online time series forecasting by Liu et al. [24], as well as for anomaly detection by Kozitsin et al. [22]. Online ARIMA has also been used for unsupervised anomaly detection [37, 37]. The Autoregressive moving average (ARMA) model and Seasonal Autoregressive Integrated Moving Average (SARIMA) proposed by Anava et al. and He et al. have also been used for time series forecasting [4, 19], with Anava et al. additionally proposing an autoregressive (AR) model for time series forecasting with missing data [5].

For neural network based approaches, Guo et al. proposed an adaptive gradient learning method for training recurrent neural networks (RNNs) for time series forecasting in the presence of anomalies and change points [18]. Their proposed model dynamically slides over the input data and updates the weights of an RNN network to reduce the weights of suspicious anomalies. Yang et al. use RoAdam (Robust Adam) to train long short-term memory (LSTM) RNNs for online time series prediction in the presence of outliers [45]. To reduce the adverse effect of outliers on the LSTM networks, they use an adaptive learning rate which reduces when the relative prediction error is reduced, and is increased when the relative error improves [45]. Wang et al. design an online sequential extreme learning machine with kernels (OS-ELMK) for nonstationary time series forecasting. A fixed memory online time series prediction scheme is used to adapt the weights in response to incoming data [43].

Neuroevolution has been widely used for time series prediction and neural architecture search in offline scenarios [25, 26, 32]. However, online neuroevolution has only been rarely investigated, with a few algorithms designed for games or simulators that involves real-time interactions, such as an online car racing simulator [11], online video games [2, 39], and robot controllers [16]. These online NE NAS algorithms are based on the venerable NeuroEvolution of Augmenting Topologies (NEAT) algorithm [38] and start with minimal networks and evolve topology and weights during the evolutionary process. Agogino et al. developed online NE algorithm that evolve feed-forward NNs to play against humans in real-time [2]. After a short period of training and evaluation offline, the NN agents continually evolve online and potentially improve performance significantly, even adapting to novel situations brought about by changing strategies in the opponent and the game layout. Stanley et al. designed an online real-time online NE video game, Neuroevolving Robotic Operatives (NERO), based on real-time Neuroevolution of Augmenting Topologies (rtNEAT) method [39]. Players train an NE agent team real-time to play with the team trained by another player [39]. Cardamone et al. developed online car racing simulator based on NEAT [38] and rtNEAT [39], combined with four evaluation strategies (ɛ-greedy [40], ɛ-greedy-improved, softmax, and interval-based). Their algorithm evolves car drivers from scratch, which can be transferred to other race car tracks. The online NE simulator was also shown to be able to outperform offline models [11].

To our knowledge, none of these online neuroevolution algorithms were capable of evolving recurrent neural networks, nor have any been developed to perform time series forecasting, making ONE-NAS the first of its kind.

3 METHODOLOGY

The proposed work utilizes components from the Evolutionary Exploration of Augmenting Memory Models (EXAMM) algorithm [31] as its core and utilizes its mutation, crossover and training operations in online scenarios. EXAMM is a distributed NE algorithm that evolves progressively larger RNNs for large-scale, multivariate, real-world TSF [14, 15]. EXAMM evolves RNN architectures consisting of varying recurrent connections and memory cells through a series of mutation and crossover (reproduction) operations. Memory cells are selected from a library of L-RNN units [33], gated recurrent units (GRUs) [12], long short-term memory cells (LSTMs) [20], minimal gated units (MGUs) [46], and update-gate RNN cells (UGRNNs) [13]. ONE-NAS utilizes EXAMM’s parallel asynchronous strategy which naturally load balances itself and allows for decoupling population size from the number of workers during each generation [31]. Generated offspring inherit their weights from their parents, which can significantly reduce the time needed for their training and evaluation [25]. It has been shown that EXAMM can swiftly adapt RNNs in transfer learning scenarios, even when the input and output data streams are changed [15] [14], which served as a preliminary motivation and justification for being able to adapt and evolve RNNs for TSF in online scenarios.

3.1 ONE-NAS

Figure 1 presents a high level view of the asynchronous, distributed, online ONE-NAS algorithm, and Algorithms 1, 2, and 3 present pseudocode for the 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. ONE-NAS does not require pre-training on any data before the online NE process.

ONE-NAS’s online NE process starts with a minimal seed genome (an RNN which consists of online input nodes fully connected to
output nodes, with no hidden layers), which serves as an initial genome. It progressively evolves generations of genomes by:

- Selecting a set of $n$ best genomes as an elite population, defined as $E_t$ for generation $t$.
- Using the elite population to generate an additional $m$ genomes for the next generation through mutation and crossover from only elite parents, defined as $O_{t+1}$.
- Performing selection on the elite population and adding them to the next generation.
- Training all the non-elite genomes in the new generation for a specified number of backpropagation epochs.
- Evaluating all the generation’s genomes on recent validation data to calculate their fitnesses.

ONE-NAS is asynchronous, with generated genomes for each generation being trained by worker processes using a work stealing strategy [9] 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.

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, after which the entire population (including elite $E_t$) will be validated on the most recent $B_{validation}$ subsequences. Each genomes fitness, calculated as the mean squared error loss over $B_{validation}$, 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 perform better than those in $O_t$, “obsolete” RNNs will naturally die off over time, however RNNs with strong performance will remain. Also note that one of EXAMM’s mutation operations utilized by ONE-NAS is a clone operation, which allows for a duplicate of an elite parent to be
with each island having their own set of elite genomes. Each island evolves similarly as in ONE-NAS, except as an extension to ONE-NAS, we add a version which includes genomes. Each island evolves similarly as in ONE-NAS, except for extinction and repopulation. 

During each ONE-NAS generation, each newly generated genome is retained and trained (re-using its parent’s weights due to EXAMM’s Lamarckian weight inheritance strategy) in the next generation. 

3.2 ONE-NAS Island Repopulation Strategy

As an extension to ONE-NAS, we add a version which includes islands which are periodically repopulated after extinction events. In the ONE-NAS Island Repopulation strategy, there are $m$ islands, with each island having 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. This strategy helps islands grow into their own niches and helps better explore the search space. It also provides robustness to variations in the online data as it arrives.

However, it is possible for islands to get stuck in local optima and stagnate. We utilize the island extinction and repopulation strategy introduced by Lyu et al. [26], 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. This strategy has been shown to even further improve the performance of island based neuroevolution strategies. For ONE-NAS’s island population strategy, after the repopulating island is erased, only the elite population $E_t$ of the island is filled with mutations of the best global 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 $E_t$.

4 DATASETS

This work utilized two real-world data sets for predicting time series data with RNNs. The first was wind turbine engine data collected and made available by ENGIE’s La Haute Borne open data windfarm[^5] 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 noiseiness and complexity of the output parameter, as well as an example of ONE-NAS’s accurate predictions on this data. The second dataset is the daily index of Dow Jones Industrial Average (DJIA) from the years 1885-1962. This dataset is univariate with 35701 samples. Both datasets contain raw and abnormal data points, and have not been cleaned, so spikes and outliers have not been removed or smoothed in the dataset.

5 EXPERIMENTAL DESIGN

Research has shown that utilizing shorter subsequences of time series data during training can improve an RNN’s convergence rate and overall performance [27]. For these experiments, 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

[^5]: https://opendata-renewables.engie.com
subsequences of historical data. There is no overlap between training and validation data (the most recent 100 subsequences are added into the historical pool after being used for validation). All the experiments for wind dataset were run for 2000 generations, which represents one singular pass over the entire wind data time series.

In the ONE-NAS experiments, 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 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 [7, 8]. 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.

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 were by selecting from EXAMM’s library of simple neurons, L-NN, 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 [21]). To prevent exploding gradients, gradient scaling [35] was used when the norm of the gradient exceeded a threshold of 1.0. To combat vanishing gradients, gradient boosting (the opposite of scaling) was used when the gradient norm was below 0.05. These parameters have been selected as suggested in prior published work on EXAMM.

6 RESULTS
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.
We are aware that the choice of window size and $\alpha$ significantly affect the prediction performance of the moving average and exponential smoothing methods. Figure 3 shows the MA and EXP prediction MSE using different window sizes ($n$) and $\alpha$ values on the wind dataset. The plot shows that the wind dataset is highly complex, where naive almost entirely predicts better than MA or EXP (apart from a negligible improvement with EXP for $alpha$ values of 0.8 and 0.9). Due to this, we can only use the naive prediction MSE to represent classic prediction MSE in the following sections.

Figure 4 shows a box and whiskers plots for 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.

While the ONE-NAS online prediction performs worse than all three classical methods, the ONE-NAS Repopulation method not only significantly outperforms the base model, but has better online prediction performance than the classical methods across all repeats except for one outlier.

### 6.2 One-NAS Repopulation

Figure 5 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 allows more variety of species which allows the algorithm to have more chance to get out of the local optima, and also potentially provides more robustness to noise and overfitting of the data.

With the same number of islands, 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. So while this brings more variety to the population, the repopulated island needs time to evolve and eventually adapt to the environment. So the number of islands and total number of generations needs to be taken into consideration when choosing extinction and repopulation frequencies.

### 6.3 Online Predictions over Time

The previous plots show the overall performance of ONE-NAS and the classical methods for the entire wind time series, which in some sense provides an additional advantage to the classical methods in that they do not require any training, and as such have a significant advantage for earlier time steps when ONE-NAS has not had much opportunity to train and evolve RNNs.

In order to investigate how much the RNNs evolved and trained by ONE-NAS were improving as they saw more data in an online fashion, we measured the for each generation how many time steps had a better prediction between the naive method and ONE-NAS. Figure 6 shows the percentage of predictions that each method provided a more accurate prediction as the search progresses, with the red line representing 50%.

From this plot we can observe that while ONE-NAS does not out perform the naive strategy within the first 500 generations, the performance of ONE-NAS improves as it sees more data, which means ONE-NAS does not only train and predict values online, it also gets better throughout the evolutionary process which is what we would expect from an online algorithm. Also,
it would be trivial to combine the two strategies, using a naive predictor until ONE-NAS has had enough evolution time to be more accurate.

However, we also notice that during generations 1500-2000, there are still more than 30% of the generations that naive does better. In practice the sensor readings are not very accurate and sometimes some sensor readings do not change or have readings of 0 for a while (i.e., they are noisy or sometimes turn off). Those situations are inevitable in practice and that data would be fed to an online predictor. When it happens, naive predictor can outperform ONE-NAS for periods of time while ONE-NAS readjusts. We did not remove invalid datapoints from the wind dataset for these experiments and we are presenting the raw prediction counts between naive and ONE-NAS, which further highlights its robustness on real-world data.

### 6.4 Online Prediction Time Efficiency

Another key concern for evaluation of online algorithms apart from prediction accuracy is time efficiency. If an online algorithm cannot provide predictions at a rate less than the arrival rate of new data to be predicted than it is not usable. ONE-NAS somewhat alleviates this problem in that it utilizes the best genome from the previous generation to provide predictions while concurrently training the next generation.

| Num Islands | Avg Time (s) | Longest Time (s) |
|-------------|-------------|------------------|
| 20          | 35.67       | 109.20           |
| 30          | 41.72       | 127.44           |
| 40          | 69.96       | 189.72           |

Table 1: The average and longest time needed to train and evolve each generation for the wind dataset.

Even so, for this work, each generation was generated and trained during a single subsequence of 25 time steps. For the wind dataset, each time step was gathered at a 10 minute interval, so this provides a significant buffer, however for many time series datasets, time step frequency can be per minute, per second, or even faster - making time efficiency a serious concern. Table 1 presents the average and longest time required to evolve and train one generation of the ONE-NAS Repopulation experiments for the varying numbers of islands. Note that population size was tied to island size, with 5 elite genomes and 10 other genomes per island, which is why the 40 island genomes took approximately twice the time. In the worst case for 40 islands, the longest time per generation was a bit above 3 minutes, which is far below the 250 minute generations time for the wind data. Variability for the generation evolution and training times comes both from the stochastic nature of how the RNN architectures are evolved, but also from the fact that the strategy will progressively grow larger RNNs, until a size is reached where adding additional components doesn’t improve performance.

It should also be noted that the workers training the genomes for each generation were distributed across 16 processors, and that performance will linearly scale upwards until the number of available processors was equal to the population size (i.e., all generated genomes can be independently trained in parallel without reduction in performance apart from fixed communication and genome generation overhead). This scalability of ONE-NAS makes it well suited to online learning. For example, if we scaled up to 200 processors from the 16 used for the 20 island experiments, we can estimate approximately 2.85 seconds per generation (i.e., training the 200 non-elite genomes at once, instead of 16 at a time), plus some additional communication and generation overhead. With a generation time of 25 time steps, this would allow for incoming data to be processed at almost 10 readings each second. Given some flexibility in determining subsequence/generation time, ONE-NAS shows the potential to be able to operate for very high frequency time series given enough computing power.

### 6.5 ONE-NAS vs Online ARIMA

While there is a significant lack of methods for online multivariate TSF, recent work by Liu et al. [24] has developed an online ARIMA method for univariate TSF. To compare ONE-NAS with a state of the art method as opposed to the classical methods investigated prior, we also compare ONE-NAS to this online ARIMA.

To reproduce results from Liu et al., we first performed experiments using the Dow Jones Industrial Dataset (DJIA), which was used for evaluation in their work. Figure 8 presents results for
their ARIMA-ONS (Arima Online Newton Step) and ARIMA-OGD (ARIMA Online Gradient descent) variants, which were the best performing in their paper. These are compared to the ONE-NAS Repopulation method with 10 islands and an extinction frequency of 200, and a similar generation and subsequence length of 25. For both ONE-NAS and online ARMIA, the plots show the online root mean squared error (RMSE) over time, from the average over 10 repeated experiments. The online RMSE over time is calculated as the average RSME of all previous predictions. For this DJIA data, we show that while the online ARIMA mirror results from their work, ONE-NAS outperforms this method by multiple orders of magnitude.

We then further compared ONE-NAS with online ARIMA on the wind datasets, which is shown in Figure 7. Similarly, the results show the average performance over 10 repeated experiments, however in this case ONE-NAS used the best hyperparameters from previous results (40 islands and an extinction frequency of 100). We performed a hyperparameter sweep for the online ARIMA methods, and selected the best hyperparameters. For ARIMA-ONS, the learning rate was set to $e^{-3}$ and $\epsilon = 3.16e^{-6}$. For ARIMA-OGD, learning rate was set to $e^{3}$, and $\epsilon = e^{-5.5}$. Similarly, ONE-NAS also outperforms the online ARIMA methods on the wind dataset.

7 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 after an initial burn-in learning period, ONE-NAS outperforms traditional online statistical time series forecasting methods such as naive, moving average and exponential smoothing, as well as modern online ARMIA methods. 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. The method is distributed and highly scalable to compute resources, allowing for generated RNNs to be trained in parallel, and performance results show that the algorithm is capable of operating on high frequency (e.g., per second) data streams given enough compute resources.

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, albeit 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. It also opens the door to significant future work, as it shows that the development of online neuroevolution algorithms is possible, as they can easily take advantage of distributed and parallel computing strategies.

8 FUTURE WORK
The ability to dynamically evolve and train populations of recurrent neural networks in online scenarios, which can outperform traditional methods, is very exciting and opens up significant avenues for future work. ONE-NAS as presented has a number of opportunities for further increasing performance. Currently, ONE-NAS continually increases its historical training data, however being selective about what historical data to retain and use for training could stand to benefit performance. Additionally, ONE-NAS currently uses a fixed generation time, it may be possible to dynamically adjust this to find more optimal generations that better align with computational requirements for real time predictions. Additionally, while we provided a brief discussion of ONE-NAS’s performance, doing a serious performance and scalability study could help inform and find areas for further performance optimizations to improve its how quickly it can generate, train and perform predictions. We further plan to investigate ONE-NAS on other datasets, as well as compare it to fixed architecture RNN strategies [18, 43, 45].

Finally, and perhaps most interesting, ONE-NAS provides populations of RNNs which can potentially provide online predictions at the end of each generation, and current results are based on utilizing only the RNN which performed best on the selected validation set. It may be possible to utilize selected RNNs as ensembles to further improve predictive ability, or even provide confidence or error bounds as to the potential accuracy of the best RNNs predictions. It is also possible to investigate retaining RNNs that perform well on historical data to further reduce or prevent catastrophic forgetting.

REFERENCES
[1] Wickliffe C Abraham and Anthony Robins. 2005. Memory retention—the synaptic stability versus plasticity dilemma. Trends in neurosciences 28, 2 (2005), 73–78.
[2] Adrian Agogino, Kenneth Stanley, and Risto Miikkulainen. 2000. Online interactive neuro-evolution. Neural Processing Letters 11, 1 (2000), 29–38.
[3] Mohanad S Al-Musaylh, Ravinesh C Deo, and Yan Li. 2020. Electrical energy demand forecasting model development and evaluation with maximum overlap discrete wavelet transform-online sequential extreme learning machines algorithms. Energies 13, 9 (2020), 2307.
[4] Oren Anava, Elad Hazan, Shie Mannor, and Ohad Shamir. 2013. Online learning for time series prediction. In Conference on learning theory. PMLR, 172–184.
