Evolving Recurrent Neural Networks for Time Series Data Prediction of Coal Plant Parameters

AbdElRahman ElSaid[1], Steven Benson[2], Shuchita Patwardhan[2], David Stadem[2], and Travis Desell[1]

[1] Rochester Institute of Technology, Rochester, NY 14623, USA
aae8800@rit.edu, tjdvse@rit.edu
[2] Microbeam Technologies Inc., Grand Forks, ND 58203
sbenson@microbeam.com,shuchita@microbeam.com,dstadem@microbeam.com

Abstract. This paper presents the Evolutionary eXploration of Augmenting LSTM Topologies (EXALT) algorithm and its use in evolving recurrent neural networks (RNNs) for time series data prediction. It introduces a new open data set from a coal-fired power plant, consisting of 10 days of per minute sensor recordings from 12 different burners at the plant. This large scale real world data set involves complex depedencies between sensor parameters and makes for challening data to predict. EXALT provides interesting new techniques for evolving neural networks, including *epigenetic weight initialization*, where child neural networks re-use parental weights as a starting point to backpropagation, as well as *node-level mutation operations* which can improve evolutionary progress. EXALT has been designed with parallel computation in mind to further improve performance. Preliminary results were gathered predicting the *Main Flame Intensity* data parameter, with EXALT strongly outperforming five traditional neural network architectures on the best, average and worst cases across 10 repeated training runs per test case; and was only slightly behind the best trained Elman recurrent neural networks while being significantly more reliable (*i.e.*, much better average and worst case results). Further, EXALT achived these results 2 to 10 times faster than the traditional methods, in part due to its scalability, showing strong potential to beat traditional architectures given additional runtime.

Keywords: Neuro-Evolution · Recurrent Neural Networks · Time Series Data Prediction.

1 Introduction

With the advent of deep learning, the use of neural networks has become widely popular across a variety of domains and problems. However, most of this success currently has been driven by human architected neural networks, which is time consuming, error prone and still leaves a major open question: *what is the optimal architecture for a neural network?* Further, optimality may have have multiple aspects and changes from problem to problem, as in one domain it may be better

to have a smaller yet less accurate neural network due to performance concerns, while in another accuracy may be more important than performance. This can become problematic as many applications of neural networks are evaluated using only a few select architectures from the literature, or may simply just pick an architecture that has shown prior success.

Another issue is that backpropagation is still the de-facto method for training a neural network. While significant performance benefits for certain types of neural networks (*e.g.*, Convolutional Neural Networks) can be gained by utilizing GPUs, other network types, such as recurrent neural netowrks (RNNs), typically cannot achieve such performance benfits without convolutional components. As backpropagation is an inherently sequential process, the time to train a single large neural network, let alone a variety of architectures, can quickly become prohibitive.

This work introduces a new algorithm, Evolutionary eXploration of Augmenting LSTM Topologies (EXALT), which borrows strategies from both NEAT (NeuroEvolution of Augmenting Topologies [1]) and its sister algorithm, EXACT (Evolutionary eXploration of Augmenting Convolutional Topologies [2, 3]) to evolve recurrent neural networks with long short-term memory (LSTM [4]) components. EXALT has been designed with concurrency in mind, and allows for multiple RNNs to be trained in a parallel manner using backpropagation while evolving their structures. EXALT expands on NEAT by having *node-level mutations* which can speed up the evolutionary process, and by utilizing backpropagation instead of an evolutionary strategy to more swiftly train the RNNs. Child RNNs re-use parental weights in an *epigenetic weight initialization* strategy, allowing them to continue training where parents left off, which further improves how quickly the algorithm evolves well performing RNNs.

This work evaluates the performance of EXALT as compared to six traditional neural network architectures (one layer and two layer feed forward neural networks; Jordan and Elman RNNs; and one layer and two layer LSTM RNNs) on a real world dataset collected from a coal-fired power plant. This data set consists of 10 days worth of per minute recordings across 12 sensors; from 12 different burners. The parameters are non-seasonal and potentially correlated, resulting in a highly complex set of data to perform predictions on. This dataset has been made open to encourage validation and reproducbility of these results, and as a valuable research to the time series data prediction research community. Having good predictors for these parameters will allow the development of tools that can be used to forecast and alert plant operators and engineers about poor boiler conditions which may occur as a result of incoming coal and/or current power plant parameters.

Preliminary results predicting the *Main Flame Intensity* parameter of this dataset with the EXALT algorithm are highly promising. K-fold cross validation was done, using each burner file as a test case; and 10 runs of each strategy were repeated for each fold. While the Elman networks were able to be trained to slightly better performance (within 0.0025 mean squared error), on average they were not nearly as reliable. EXALT outperformed all the other network

architectures in best, average and worst cases, and while finding more efficient (*i.e.*, smaller) RNNs than the traditional architectures, and was able to do so in significantly less time (between 2 to 10 times faster) operating in parallel across 20 processors. These preliminary results shows the strong potential of this algorithm in evolving RNNs for time series data prediction.

The remainder of this paper is as follows. Section 2 presents related work. Section 3 describes the EXALT algorithm in detail. Section 4 introduces the coal-fired power plant data set, and Section 5 provides initialization settings and results for the EXALT algorithm and fixed neural networks. The paper ends with a discussion of conclusions and future work in Section 6.

2 Related Work

2.1 Recurrent Neural Networks (RNNs)

RNNs have an advantage over standard feed forward (FF) neural networks (NNs), as they can deal with sequential input data, using their internal memory to process sequences of inputs and use previously stored information to aid in future predictions. This is done by allowing connections between neurons across timesteps, which aids them in predicting more complex data [5]. However, this leads to a more complicated training process as RNNs need to be "unrolled" over each time step of the data and trained using backpropagation through time (BPTT) [6].

In an effort to better train RNNs and capture time dependencies in data, long short-term memory (LSTM) RNNs were first introduced by S. Hochrieter & J. Schmidhuber [4]. LSTM neurons provide a solution for the exploding/vanishing gradients problem by utilizing input, forget and output gates in each LSTM cell, which can control and limit the backward flow of gradients in BPTT [7]. LSTM RNNs have been used with strong performance in image recognition [8], audio visual emotion recognition [9], music composition [10] and other areas. Regarding time series prediction, for example, LSTM RNNs have been used for stock market forecasting [11] and forex market forecasting [12]. Also forecasting wind speeds [7, 13] for wind energy mills, and even predicting diagnoses for patients based on health records [14].

2.2 Evolutionary Optimization Methods

The EXALT algorithm presented in this work is in part based its sister algorithm, Evolutionary eXploration of Augmenting Convolutional Topologies (EXACT), which has successfully been used to evolve convolutional neural networks (CNNs) for image prediction tasks [2, 3]. However, where EXACT evolves feature maps and filters to construct CNNs, EXALT utilizes LSTM and regular neurons along with feed forward and recurrent connections to evolve RNNs. EXALT also utilizes the *epigenetic weight initialization* strategy (see Section 3.2 that was shown by EXACT to improve training performance [3].

Other work by Desell and ElSaid [15–17] has utilized an ant colony optimization based approach to select which connections should be utilized in RNNs and LSTM RNNs for the prediction of flight parameters. In particular, this ACO approach was shown to reduce the number of trainable connections in half while providing a significant improvement in predictions of engine vibration [16]. However, this approach works within a fixed RNN architecture and cannot evolve an overall RNN structure.

Several other methods for evolving NN topologies along with weights have been researched and deployed. In [1], NeuroEvolution of Augmenting Topologies (NEAT) has been developed. It is a genetic algorithm that evolves increasingly complex neural network topologies, while at the same time evolving the connection weights. Genes are tracked using historical markings with innovation numbers to perform crossover among different structures and enable efficient recombination. Innovation is protected through speciation and the population initially starts small without hidden layers and gradually grows through generations [18–20]. Experiments have demonstrated that NEAT presents an efficient way for evolving neural networks for weights and topologies. Its power resides in its ability to combine all the four main aspects discussed above and expand to complex solutions. However NEAT still has some limitations when it comes evolving neural networks with weights or LSTM cells for time series prediction tasks as described in [15].

Other more recent work by Rawal and Miikkulainen has utilized tree based encoding [21] and information maximization objectives [22] to evolve RNNs. EXALT differs from this work in a few notable ways, first, the tree-based encoding strategy uses a genetic programming strategy to evolve connections within recurrent neurons, and only utilizes fixed architectures built of layers of evolved node types. On the other hand, the information maximization strategy utilizes NEAT with LSTM neurons instead of regular neurons. EXALT allows the evolution of RNNs with both regular and LSTM neurons, adds new node-level mutation operations and uses backpropagation to train the evolved RNNs (see Section 3). Furthermore, it has been developed with large scale concurrency in mind, and utilizes an asynchronous steady-state approach, which has been shown to allow scalability to potentially millions of compute nodes [23].

3 Evolutionary Exploration of Augmenting LSTM Topologies (EXALT)

EXALT has been developed with parallel/concurrent operation in mind. It utilizes a steady state population and generates new RNNs to be evaluated upon request by workers. When a worker completes training a RNN, it is inserted into the population if its fitness (mean squared error on the test data) is better than the worst in the population, and then the worst in the population is removed. This strategy is particularly important as the generated RNNs will have different architectures and will not take the same amount of time to train. By having a master process control the population, workers can complete the

training of the generated RNNs at whatever speed they can and the process is naturally load balanced. Further, this allows EXALT to scale to however many processors are available, while having the population size be independent of processor availability, unlike synchronous parallel evolutionary strategies. The EXALT codebase has a multithreaded implementation for multicore CPUs as well as an MPI (the message passing interface [24]) implementation for use on high performance computing resources.

3.1 Mutation and Recombination Operations

RNNs are evolved with edge-level operations, as done in NEAT, as well as with new high level node mutations. Whereas NEAT only requires innovation numbers for new edges, EXALT requires innovation numbers for both new nodes and new edges. The master process keeps track of all node, edge and recurrent edge innovations made, which are required to perform the crossover operation in linear time without a graph matching algorithm. Figure 1 displays a visual walkthrough of all the mutation operations used by EXALT. Nodes and edges selected to be modified are highlighted, and then new elements to the RNN are shown in green. Edge innovation numbers are not shown for clarity. Enabled edges are in black, disabled edges are in grey.

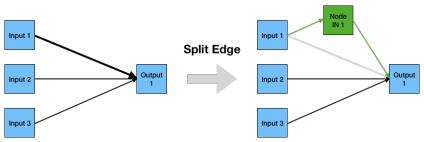
Edge Mutations:

Disable Edge This operation randomly selects an enabled edge or recurrent edge in a RNN genome and disables it so that it is not used. The edge remains in the genome. As the *disable edge* operation can potentially make an output node unreachable, after all mutation operations have been performed to generate a child RNN genome, if any output node is unreachable that RNN genome is discarded and a new child is generated by another attempt at mutation.

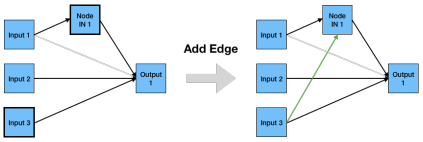
Enable Edge If there are any disabled edges or recurrent edges in the RNN genome, this operation selects one at random and enables it.

Split Edge This operation selects an enabled edge at random and disables it. It creates a new node (creating a new node innovation) and two new edges (creating two new edge innovations), and connects the input node of the split edge to the new node, and the new node to the output node of the split edge. The new node is either a regular neuron or LSTM neuron, selected randomly at 50% each.

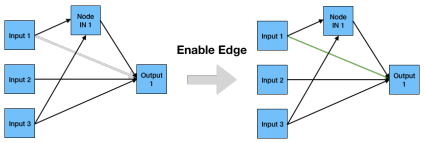
Add Edge This operation selects two nodes n_1 and n_2 within the RNN Genome at random, such that $depth_{n_1} < depth_{n_2}$ and such that there is not already an edge between those nodes in this RNN Genome, and then adds an edge from n_1 to n_2. If an edge between n_1 and n_2 exists within the master's innovation list, that edge innovation is used, otherwise this creates a new edge innovation.



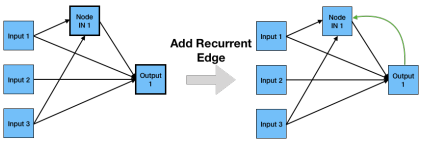
(a) The edge between Input 1 and Output 1 is selected to be split. A new node with innovation number (IN) 1 is created.



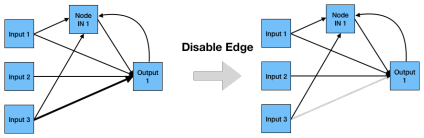
(b) Input 3 and Node IN 1 are selected to have an edge between them added.



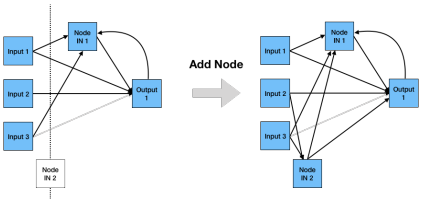
(c) The edge between Input 3 and Output 1 is enabled.



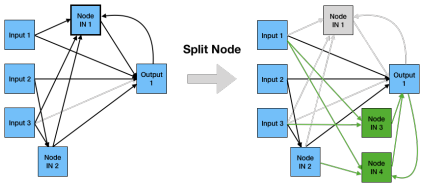
(d) A recurrent edge is added between Output 1 and Node IN 1



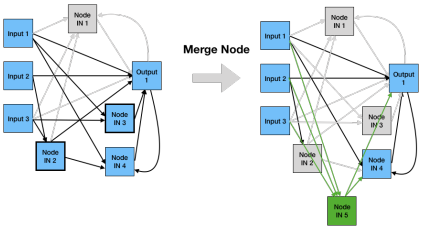
(e) The edge between Input 3 and Output 1 is disabled.



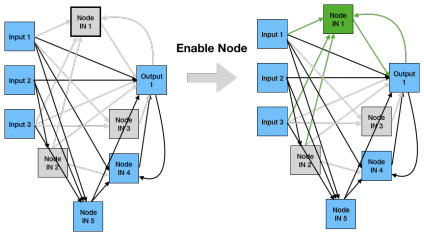
(f) A node with IN 2 is added at a depth between the inputs and Node IN 1. Edges are randomly added to Input 2 and 3, and Node IN 1 and Output 1.



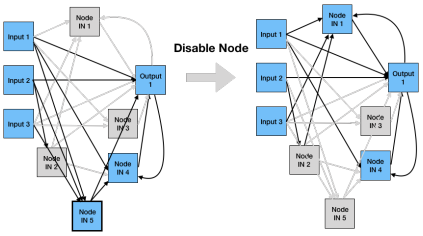
(g) Node IN 1 is split into Nodes IN 3 and 4, which get half the inputs. Both have an output edge to Output 1, because there was only one output from Node IN 1.



(h) Node IN 2 and 3 are selected to be merged. They are disabled along with their input/output edges. Node IN 5 is created with edges between all their inputs and outputs.



(i) Node IN 1 is selected to be enabled, along with all its input and output edges.



(j) Node IN 5 is selected to be disabled, along with all its input and output edges.

Fig. 1. Edge and node mutation operations.

Add Recurrent Edge This operation selects two nodes n_1 and n_2 within the RNN Genome at random and then adds a recurrent edge from n_1 to n_2. Recurrent edges can span multiple time steps, with the edge's *recurrent depth* selected uniformly at random between 1 and 10 time steps. If a recurrent edge between n_1 and n_2 exists within the master's innovation list with the same recurrent depth, that recurrent edge innovation is used, otherwise this creates a new recurrent edge innovation.

Node Mutations:

Disable Node This operation selects a random non-input and non-output node and disabled it along with all of its incoming and outgoing edges.

Enable Node This operation selects a random disabled node and enables it along with all of its incoming and outgoing edges.

Add Node This operation selects a random depth between 0 and 1, non-inclusive. Given that the input node is always depth 0 and the output nodes are always depth 1, this depth will split the RNN in two. A new node is created, at that depth, and 1-5 edges are randomly generated to nodes with a lesser depth, and 1-5 edges are randomly generated to nodes with a greater depth. The node size is set to the average of the maximum input node size and minimum output node size. The new node will be either a regular or LSTM neuron, selected randomly at 50% each. Newly created edges are 50% feed forward and 50% recurrent, selected randomly.

Split Node This operation takes one non-input, non-output node at random and splits it. This node is disabled (as in the disable node operation) and two new nodes are created at the same depth as their parent. One input and one output edge are assigned to each of the new nodes, with the others being assigned randomly, ensuring that the newly created nodes have both inputs and outputs. If there is only one input or one output edge to this node, then those edges are duplicated for the new nodes. The new nodes will be either a regular or LSTM neuron, selected randomly at 50% each. Newly created edges are 50% feed forward and 50% recurrent, selected randomly.

Merge Node This operation takes two non-input, non-output nodes at random and combines them. The selected nodes are disabled (as in the disable node operation) and a new node is created with a depth equal to average of its parents. This node is connected to the inputs and outputs of its parents, with input edges created to those with a lower depth, and output edges created to those with a deeper depth. The new node will be either a regular or LSTM neuron, selected randomly at 50% each. Newly created edges are 50% feed forward and 50% recurrent, selected randomly.

Other Operations:

Crossover utilizes two hyperparameters, the *more fit crossover rate* and the *less fit crossover rate*. Two parent RNN genomes are selected, and the child RNN genome is generated from every edge that appears in both parents. Edges that only appear in the more fit parent are added randomly at the *more fit crossover rate*, and edges that only appear in the less fit parent are added randomly at the *less fit crossover rate*. Edges not added by either parent are also carried over into the child RNN genome, however they are set to disabled. Nodes are then added for each input and output of an edge. If the more fit parent has a node with the same innovation number, it is added from the more fit parent.

Clone creates a copy of the parent genome, initialized to the same weights. This allows a particular genome to continue training in cases where further training may be more beneficial than performing a mutation or crossover.

3.2 Epigenetic Weight Initialization

For RNNs generated during population initialization, the weights are initialized uniformly at random between -0.5 and 0.5. Biases and weights for new nodes and edges are initialized randomly with a normal distribution based on the average, μ and variance, σ^2 of the parent's weights. However, RNNs generated through mutation or crossover re-use the weights of their parents, allowing the RNNs to train from where the parents are left off, *i.e.*, *"epigenetic" weight initialization* – these weights are a modification of how the genome is expressed as opposed to a modification of the genome itself.

Additionally, for crossover in the case of where an edge or node exists in both parents, the child weights are generated by recombining the parents weights. Given a random number $-0.5 <= r <= 1.5$, a child's weight w_c is set to $w_c = r(w_{p2} - w_{p1}) + w_{p1}$, where w_{p1} is the weight from the more fit parent, and w_{p2} is the weight from the less fit parent. This allows the child weights to be set along a gradient calculated from the weights of the two parents.

4 Open Data and Reproducibility

The dataset examined in this work is time series data gathered from a coal-fired power plant. The data consists of 10 days of per-minute data readings extracted from 12 of the plant's burners. The data has 12 parameters of time series data:

1. Conditioner Inlet Temp
2. Conditioner Outlet Temp
3. Coal Feeder Rate
4. Primary Air Flow
5. Primary Air Split
6. System Secondary Air Flow Total
7. Secondary Air Flow
8. Secondary Air Split
9. Tertiary Air Split
10. Total Combined Air Flow
11. Supplementary Oil Flow
12. Main Flame Intensity

In order to protect the confidentiality of the power plant which provided the data, along with any sensitive data elements, all identifying data has been

scrubbed from the data sets (such as dates, times, locations and facility names). Further, the data has been pre-normalized between 0 and 1 as a further precaution. So while the data cannot be reverse engineered to identify the originating power plant or actual parameter values – it still is an extremely valuable test data set for times series data prediction as it consists of real world data from a highly complex system with interdependent data streams.

In this work, one of the parameters was of key interest for time series data prediction, *Main Flame Intensity*, and was used as the parameter for prediction while gathering the results. In order to further reproducibility of these results and provide this important data set to the time series data prediction research community, it has been made available as part of the EXACT/EXALT GitHub repository, along with instructions on how to use the EXALT code base to recreate these results[3].

5 Results

Two sets of results were gathered predicting *Main Flame Intensity* from the coal plant data set. Six common fixed neural network architectures for time series data prediction were investigated: 1) a one layer feed forward (FF), neural network (NN) 2) a two layer FF NN, 3) an Jordan recurrent neural network (RNN), 4) an Elman RNN, 5) a one layer long short-term memory (LSTM) RNN and 6) a two layer LSTM RNN. K-fold cross validation was performed with 12 folds (*i.e.*, each of the 12 burner data sets was left out to be tested on after training using the other 11 burner data sets). Each NN was trained 10 times for each output data file, resulting in 120 NNs being trained for each NN type. Similarly, EXALT was run 10 times per fold, using each of the 12 burner data sets as testing data, for a total of 120 runs.

Results were gathered using university research computing systems. Compute nodes utilized ranged between 10 core 2.3 GHz Intel®Xeon®CPU E5-2650 v3, 32 core 2.6 GHz AMD Opteron^{TM}Processor 6282 SE and 48 core 2.5 GHz AMD Opteron^{TM}Processor 6180 SEs, which was unavoidable due to cluster scheduling policies. All compute nodes ran RedHat Enterprise Linux 6.10. This did result in some variation in performance, however discrepancies in timing were overcome by averaging over multiple runs in aggregate. The 720 fixed architecture runs were performed in parallel across 60 compute nodes and took approximately 1,500 compute hours in total. The 120 EXALT runs were performed with each run utilizing 20 processors in parallel, and required 50 compute hours in total.

All neural networks were trained with stochastic backpropagation using the same hyperparameters. Backpropagation was run with a learning rate $\eta = 0.001$, utilizing Nesterov momentum with $mu = 0.9$ and without dropout, as dropout has been shown in other work to reduce performance when training RNNs for time series prediction [16]. To prevent exploding gradients, gradient clipping (as described by Pascanu *et al.* [25]) was used when the norm of the gradient was

[3] URL removed due to the double blind review process.

above a threshold of 1.0. To improve performance for vanishing gradients, gradient boosting (the opposite of clipping) was used when the norm of the gradient was below a threshold of 0.05. Initial network weights were randomly initialized uniformly at random between -0.5 and 0.5, however the forget gate bias of the LSTM neurons had 1.0 added to it as this has shown significant improvements to training time by Jozefowicz *et al.* [26]. The fixed NN architectures were trained for 1000 epochs, and EXALT trained 2000 RNNs, with each trained for 10 epochs. As this was in total 20,000 epochs performed in parallel over 20 processors it was seen to be somewhat equivalent to training a single NN For 1000 epochs.

Each EXALT run was done with a population size of 20, and new RNNs were generated via crossover 25% of the time, and by mutation 75% of the time. Mutation operations were performed at the following rates:

1. *clone:* 1/17
2. *add edge:* 1/17
3. *add recurrent edge:* 3/17
4. *enable edge:* 1/17
5. *disable edge:* 3/17
6. *split edge:* 1/17
7. *add node:* 1/17
8. *enable node:* 1/17
9. *disable node:* 3/17
10. *split node:* 1/17
11. *merge node:* 1/17

Mutation rates were chosen in a manner to give mostly equal weighting to each mutation operation. *Add recurrent edge* was given some extra preference as it could be potentially adding recurrent edges with recurrent depths between 1 and 10, which provides a lot of potential options. *Disable edge* and *disable node* were also given extra preference to counteract the RNNs growing quickly, as the other options would put more weight on increasing the RNN size.

Figure 2 shows the minimum, maximum and average progress of the six fixed neural network architectures for each fold, along with the minimum, average and maximum progress of for each EXALT run on each fold. EXALT shows dramatic improvements in reliability and performance over training multiple fixed architecture neural networks. Table 1 presents the aggregrate results across each of the folds as well as in total. Two major observations can be made from this, first, the EXALT runs were shown to be much more reliable than training multiple fixed NN architectures, and second, the EXALT runs completed in significantly less time (which was unexpected).

While it was expected that having EXALT evaluate 2000 RNNs each for 10 epochs across 20 nodes in parallel would result in a relatively similar amount of time to training a fixed architectures for 1000 epochs; EXALT runs on average completed more than twice as fast as even the simplest architecture evaluated (a one layer FF NN). Table 2 shows the number of nodes, edges, recurrent edges and trainable connections (weights) for each neural network type, as well as the average counts of these across the best evolved RNNs by EXALT. Overall, EXALT found well performing RNNs that were much smaller than the fixed network sizes. Figure 3 presents some of the best evolved RNNs. These RNNs dropped out some inputs and were more sparsely connected. Interestingly, they were able to perform very well (better than most of the larger fixed architectures) with only a few hidden nodes and sparse connections.

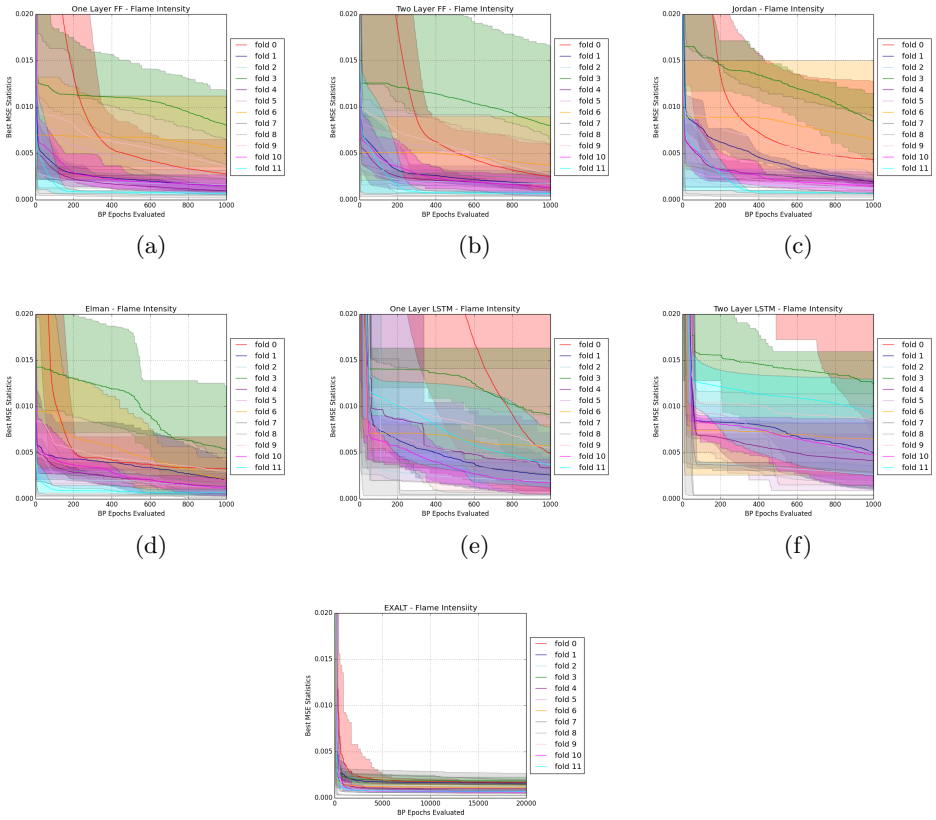


Fig. 2. These plots present the minimum, average, and maximum mean squared errors across each of the 12 folds used by K-fold cross validation by the one layer feed forward NN (2a), two layer feed forward NN (2b), Jordan RNN (2c), Elman RNN (2d), one layer LSTM RNN (2e), two layer LSTM RNN (2f), and by EXALT (2g).

	one layer ff			
	Min	**Avg**	**Max**	**Time**
Fold 0	0.031809	0.044369	0.072142	3658
Fold 1	0.024417	**0.031502**	0.040341	4040
Fold 2	0.020960	0.024908	0.033439	4033
Fold 3	0.033071	0.044107	0.056134	4027
Fold 4	0.030796	0.049311	0.085186	4079
Fold 5	0.033532	0.039205	0.047536	3967
Fold 6	0.010756	0.016743	0.023700	3633
Fold 7	0.030178	0.054017	0.075785	3943
Fold 8	0.019893	0.033458	0.047565	3938
Fold 9	0.016084	0.019077	0.023716	3958
Fold 10	0.023736	0.032435	0.040408	4029
Fold 11	0.041660	0.074404	0.100530	3781
Average	0.026408	0.038628	0.053874	3924

	two layer ff			
	Min	**Avg**	**Max**	**Time**
Fold 0	0.026313	0.042009	0.073753	6670
Fold 1	0.026775	0.033963	0.046181	7542
Fold 2	0.019418	0.028966	0.046257	7480
Fold 3	0.029042	0.051393	0.073627	7615
Fold 4	0.023416	0.037335	0.051478	7639
Fold 5	0.031064	0.039306	0.046585	7616
Fold 6	0.014612	0.016611	0.019820	6345
Fold 7	0.028875	0.045736	0.077376	7222
Fold 8	0.016406	0.031521	0.046914	7547
Fold 9	0.016174	**0.018498**	0.021877	7683
Fold 10	0.025587	0.033352	0.038321	7609
Fold 11	0.036185	0.065018	0.121369	7460
Average	0.024489	0.036976	0.055296	7369

	jordan			
	Min	**Avg**	**Max**	**Time**
Fold 0	0.035064	0.050483	0.097150	3793
Fold 1	0.033920	0.039394	0.043391	3663
Fold 2	0.029067	0.036748	0.046604	3696
Fold 3	0.022927	0.028984	0.034974	3821
Fold 4	0.038322	0.063602	0.098186	3715
Fold 5	0.034472	0.038310	0.043646	3735
Fold 6	0.013130	0.016467	0.020744	3895
Fold 7	0.038538	0.054139	0.090888	3684
Fold 8	0.020665	0.033360	0.043029	3395
Fold 9	0.016776	0.018601	**0.020237**	3439
Fold 10	0.025305	0.028733	**0.032498**	3423
Fold 11	0.055703	0.082065	0.097041	3507
Average	0.030324	0.040907	0.055699	3647

	exalt			
	Min	**Avg**	**Max**	**Time**
Fold 0	0.025360	**0.028749**	**0.030883**	**1675**
Fold 1	0.029976	0.031769	**0.033015**	**1864**
Fold 2	0.021359	**0.023095**	**0.024838**	**2137**
Fold 3	0.018214	**0.019229**	**0.020563**	**1911**
Fold 4	0.020932	**0.023170**	**0.025770**	**1701**
Fold 5	0.030464	0.036091	**0.042542**	**1812**
Fold 6	0.011974	0.012879	**0.013904**	**1763**
Fold 7	**0.016564**	**0.019358**	**0.020220**	**1847**
Fold 8	0.015867	**0.018151**	**0.020786**	**1885**
Fold 9	0.016922	0.019475	0.021441	**1751**
Fold 10	0.020945	0.030016	0.032662	**1741**
Fold 11	0.026530	**0.031207**	**0.035073**	**1573**
Average	0.021259	**0.024432**	**0.026808**	**1805**

	elman			
	Min	**Avg**	**Max**	**Time**
Fold 0	0.030173	0.047723	0.073134	6306
Fold 1	**0.014476**	0.035610	0.060415	6225
Fold 2	**0.017132**	0.027319	0.044997	5996
Fold 3	**0.016477**	0.027119	0.033858	5572
Fold 4	**0.017084**	0.029284	0.040682	5848
Fold 5	**0.022649**	**0.031657**	0.045849	5700
Fold 6	**0.008368**	**0.012861**	0.014999	5531
Fold 7	0.018732	0.045840	0.059511	5893
Fold 8	**0.012740**	0.027437	0.043608	6135
Fold 9	**0.013751**	0.018502	0.025968	5957
Fold 10	**0.017572**	**0.028322**	0.038500	6208
Fold 11	**0.024479**	0.053423	0.094839	5717
Average	**0.017803**	0.032092	0.048030	5924

	one layer lstm			
	Min	**Avg**	**Max**	**Time**
Fold 0	**0.017438**	0.052460	0.085376	8221
Fold 1	0.019471	0.037559	0.058379	8552
Fold 2	0.025880	0.039290	0.050615	8837
Fold 3	0.018254	0.025687	0.045868	8118
Fold 4	0.020927	102.586834	512.731000	7887
Fold 5	0.033102	0.043610	0.048107	7943
Fold 6	0.014528	37.755357	188.717000	7764
Fold 7	0.019844	0.034766	0.054353	7860
Fold 8	0.013022	0.098911	0.412826	8039
Fold 9	0.017950	0.035914	0.052342	8069
Fold 10	0.031792	0.035349	0.037908	8038
Fold 11	0.051159	0.076199	0.112534	7998
Average	0.023614	11.735161	58.533859	8110

	two layer lstm			
	Min	**Avg**	**Max**	**Time**
Fold 0	0.057165	0.135696	0.227263	16948
Fold 1	0.020384	0.049007	0.063610	19768
Fold 2	0.026154	0.037273	0.056096	19958
Fold 3	0.020337	0.059560	0.095907	20989
Fold 4	0.038711	0.044041	0.055016	22132
Fold 5	0.024799	0.043924	0.050945	21701
Fold 6	0.014154	0.014464	0.015441	16330
Fold 7	0.026489	0.085277	0.103150	17456
Fold 8	0.022628	0.050065	0.076219	20140
Fold 9	0.026221	0.042297	0.050324	21682
Fold 10	0.028380	0.035213	0.038571	22923
Fold 11	0.052778	0.069439	0.100021	22923
Average	0.029850	0.055521	0.077714	20336

Table 1. K-fold cross validation statistics for EXALT and the 6 fixed neural network architectures, presenting the mean squared error and runtime over the 10 repeated trainings. Best results for each fold are shown in bold.

	Nodes	Edges	Rec. Edges	Weights
One Layer FF	25	156	0	181
Two Layer FF	37	300	0	337
Jordan RNN	25	156	12	193
Elman RNN	25	156	144	325
One Layer LSTM	25	156	0	311
Two Layer LSTM	37	300	0	587
EXALT Best Avg.	14.7	26.2	14.6	81.5

Table 2. Number of nodes, edges, recurrent edges and trainable connections (weights) in each evaluated network type, and the average values for the best evolved RNNs by EXALT.

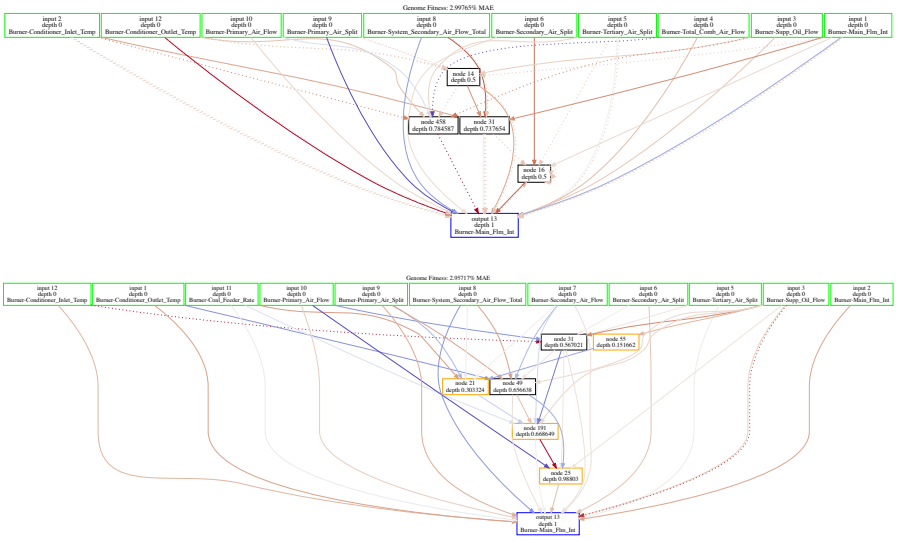


Fig. 3. Two examples of the best RNNs evolved by EXALT. Orange nodes are LSTM neurons, while black ndoes are regular neurons. Dotted lines represet recurrent connections, while solid lines represent feed forward connections. Colors of the lines represent the magnitude of the weights weights (-1.0 is the most blue to 1.0 being the most red).

So while the Elman network were sometimes able to find the best predictions for some folds, in aggregate these were much more unreliable than utilizing EXALT, which came quite close to these results in the best case. Further, the EXALT runs typically completed in under a third of the time. We expect that running EXALT for a similar length of time will be even closer or outperform these networks.

6 Discussion

Preliminary results for EXALT on this coal fired power plant dataset are very promising. EXALT is quickly able to evolve RNNs that are more efficient (*i.e.*, fewer nodes and trainable connections) than standard RNN architectures, with comparable results. EXALT's best found RNNs outperformed one and two layer feed forward and LSTM neural networks, and had much better average and worst case results than all tested architectures. While Elman networks did find some networks with better results (within a small margin of 0.0025 mean squared error), on average it performed quite a bit worse, and these networks also took over 3 times longer to train - future results providing more time for EXALT to evolve its networks should provide even better results.

This work also introduces a valuable time series dataset gathered from a coal fired power plant, presenting 10 days worth of per minute readings from 12 different burners across 12 different sensors. Having this open large scale real world time series data set of this nature will be very useful for researchers in the field of time series data prediction, and to the authors' knowledge there is not a similar data set available.

There is also potential for significant future work. While the *Main Flame Intensity* parameter was the focus of this work, as having a good predictor for this parameter can help improve plant performance; there are a number of other paramters which can be predicted as well. Further, predictions were only made one time step (*i.e.*, one minute) in the future. Investigating more parameters further in the future along with other data sets will help further demonstrate the effectiveness of the EXALT algorithm.

Using these trained RNNs, the project team aims to develop an advanced tool for coal-fired power plants to actively monitor and manage coal quality and overall boiler conditions that will provide a means to maximize availability and maintain generating capacity while reducing cost. The tool will be used to forecast and alert plant operators and engineers about poor boiler conditions which may occur as a result of incoming coal and/or current power plant parameters.

A more detailed look into how effective the various EXALT mutations are can further improve performance, as well as co-evolution of hyperparameters, which has shown to provide benefits when evolving convolutional neural networks with EXALT's sister algorithm EXACT [2, 3]. Additionally, as EXALT converged fairly quickly to a solution in this work, there is potential that methods for increasing speciation may help find better results. One approach would be to utilize mutliple islands evolving in parallel with occasional data transfer, which

has been shown by Alba *et al.* to provide significant performane benefits for parallel evolutionary algorithms [27]. Additionally, EXALT was run only utilizing 20 processors and an investigation of its scalability will be interesting.

Overall these preliminary results for EXALT are quite exciting as it provides a parallel algorithm to both train and evolve the structure of RNNs. It can perform parameter selection by dropping out input connections and for the data set tested it generated smaller more accurate RNNs in a shorter amount of time than traditional architectures and backpropagation alone. Further, as it in part utilizes backpropagation, it can be used in conjunction with and stands to benefit from other RNN training methodologies which the machine learning community may develop.

7 Acknowledgements

This material is based upon work supported by the U.S. Department of Energy, Office of Science, Office of Advanced Combustion Systems under Award Number #FE0031547.

References

1. Stanley, K., Miikkulainen, R.: Evolving neural networks through augmenting topologies. Evolutionary computation 10(2), 99–127 (2002)
2. Desell, T.: Developing a volunteer computing project to evolve convolutional neural networks and their hyperparameters. In: The 13th IEEE International Conference on eScience (eScience 2017). pp. 19–28 (Oct 2017)
3. Desell, T.: Large scale evolution of convolutional neural networks using volunteer computing. CoRR abs/1703.05422 (2017), http://arxiv.org/abs/1703.05422
4. S. Hochrieter & J. Schmidhuber: Long Short Term Memory. Neural Computation 9(8):1735-1780 (1997)
5. Gers, F.A., Schraudolph, N.N., Schmidhuber, J.: Learning precise timing with lstm recurrent networks. Journal of machine learning research 3(Aug), 115–143 (2002)
6. Werbos, P.J.: Backpropagation through time: what it does and how to do it. Proceedings of the IEEE 78(10), 1550–1560 (1990)
7. Felix A. Gers, Jürgen Schmidhuber, and Fred Cummins: Learning to Forget: Continual Prediction with LSTM. Neural Computation, Vol. 12, No. 10 , Pages 2451-2471 (October 2000)
8. Donahue, J., Anne Hendricks, L., Guadarrama, S., Rohrbach, M., Venugopalan, S., Saenko, K., Darrell, T.: Long-term recurrent convolutional networks for visual recognition and description. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 2625–2634 (2015)
9. Chao, L., Tao, J., Yang, M., Li, Y., Wen, Z.: Audio visual emotion recognition with temporal alignment and perception attention. arXiv preprint arXiv:1603.08321 (2016)
10. Eck, D., Schmidhuber, J.: A first look at music composition using lstm recurrent neural networks. Istituto Dalle Molle Di Studi Sull Intelligenza Artificiale 103 (2002)

11. Di Persio, L., Honchar, O.: Artificial neural networks approach to the forecast of stock market price movements. International Journal of Economics and Management Systems 1 (2016)
12. Maknickienė, N., Maknickas, A.: Application of neural network for forecasting of exchange rates and forex trading. In: The 7th international scientific conference" Business and Management. pp. 10–11 (2012)
13. Felder, M., Kaifel, A., Graves, A.: Wind power prediction using mixture density recurrent neural networks. In: Poster Presentation gehalten auf der European Wind Energy Conference (2010)
14. Choi, E., Bahadori, M.T., Sun, J.: Doctor ai: Predicting clinical events via recurrent neural networks. arXiv preprint arXiv:1511.05942 (2015)
15. Desell, T., Clachar, S., Higgins, J., Wild, B.: Evolving deep recurrent neural networks using ant colony optimization. In: European Conference on Evolutionary Computation in Combinatorial Optimization. pp. 86–98. Springer (2015)
16. ElSaid, A., El Jamiy, F., Higgins, J., Wild, B., Desell, T.: Optimizing long short-term memory recurrent neural networks using ant colony optimization to predict turbine engine vibration. Applied Soft Computing (2018)
17. ElSaid, A., Jamiy, F.E., Higgins, J., Wild, B., Desell, T.: Using ant colony optimization to optimize long short-term memory recurrent neural networks. In: Proceedings of the Genetic and Evolutionary Computation Conference. pp. 13–20. ACM (2018)
18. Annunziato, M., Lucchetti, M., Pizzuti, S.: Adaptive systems and evolutionary neural networks: a survey. Proc. EUNITE02, Albufeira, Portugal (2002)
19. Larochelle, H., Bengio, Y., Louradour, J., Lamblin, P.: Exploring strategies for training deep neural networks. Journal of Machine Learning Research 10(Jan), 1–40 (2009)
20. Kandel, E.R., Schwartz, J.H., Jessell, T.M., Siegelbaum, S.A., Hudspeth, A.J.: Principles of neural science, vol. 4. McGraw-hill New York (2000)
21. Rawal, A., Miikkulainen, R.: From nodes to networks: Evolving recurrent neural networks. CoRR abs/1803.04439 (2018), http://arxiv.org/abs/1803.04439
22. Rawal, A., Miikkulainen, R.: Evolving deep lstm-based memory networks using an information maximization objective. In: Proceedings of the Genetic and Evolutionary Computation Conference 2016. pp. 501–508. ACM (2016)
23. Desell, T.: Asynchronous Global Optimization for Massive Scale Computing. Ph.D. thesis, Rensselaer Polytechnic Institute (2009)
24. Message Passing Interface Forum: MPI: A message-passing interface standard. The International Journal of Supercomputer Applications and High Performance Computing 8(3/4), 159–416 (Fall/Winter 1994)
25. Pascanu, R., Mikolov, T., Bengio, Y.: On the difficulty of training recurrent neural networks. In: International Conference on Machine Learning. pp. 1310–1318 (2013)
26. Jozefowicz, R., Zaremba, W., Sutskever, I.: An empirical exploration of recurrent network architectures. In: International Conference on Machine Learning. pp. 2342–2350 (2015)
27. Alba, E., Tomassini, M.: Parallelism and evolutionary algorithms. Evolutionary Computation, IEEE Transactions on 6(5), 443–462 (2002)