Neuro-Evolutionary Transfer Learning through Structural Adaptation

AbdElRahman ElSaid, Joshua Karnas, Zimeng Lyu, Daniel Krutz, Alexander G. Ororbia, and Travis Desell

Rochester Institute of Technology, Rochester, NY 14623, USA
aae8800@rit.edu, josh@mail.rit.edu, zimenglyu@mail.rit.edu, dxkvse@rit.edu ago@cs.rit.edu, tjdvse@rit.edu

Abstract. Transfer learning involves taking an artificial neural network (ANN) trained on one dataset (the source) and adapting it to a new, second dataset (the target). While transfer learning has been shown to be quite powerful and is commonly used in most modern-day statistical learning setups, its use has generally been restricted by architecture, i.e., in order to facilitate the reuse of internal learned synaptic weights, the underlying topology of the ANN to be transferred across tasks must remain the same and a new output layer must be attached (entailing tossing out the old output layer's weights). This work removes this restriction by proposing a neuro-evolutionary approach that facilitates what we call *adaptive structure transfer learning*, which means that an ANN can be transferred across tasks that have different input and output dimensions while having the internal latent structure continuously optimized. We test the proposed optimizer on two challenging real-world time series prediction problems – our process adapts recurrent neural networks (RNNs) to 1) predict coal-fired power plant data before and after the addition of new sensors, and to 2) predict engine parameters where RNN estimators are trained on different airframes with different engines. Experiments show that not only does the proposed neuro-evolutionary transfer learning process result in RNNs that evolve and train faster on the target set than those trained from scratch but, in many cases, the RNNs generalize better even after a long training and evolution process. To our knowledge, this work represents the first use of neuro-evolution for transfer learning, especially for RNNs, and is the first methodological framework capable of adapting entire structures for arbitrary input/output spaces.

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

1 Introduction

Transfer learning has proven to be a powerful tool for training artificial neural networks (ANNs), allowing them to re-use knowledge gained after training on one task in order to better, more quickly generalize on a new target task. However, one of the key limitations of ANN-based transfer learning is that the

process typically requires neural architecture being transferred to remain fixed so that the previously its trained weights to be reused or tuned. Furthermore, transfer learning is generally only utilized for classification-based tasks, where the underlying neural structure is feedforward or convolutional in nature, with little to no attention paid to transferring knowledge across recurrent neural networks (RNNs) aside from the notable exception of partial knowledge transfer through the use of pre-trained embeddings of (sub)words and phrases [1, 2]. In addition, to our knowledge, no work exists on developing a transfer learning framework for RNNs for the far harder task of time series forecasting.

Why is time series forecasting so important? We argue that countless real-world systems would benefit from the ability to forecast or predict data. A cloud-based self-adaptive hosting system will benefit from being able to anticipate future resource needs. The ability to predict user loads and the time necessary to allocate the required resources to handle these loads would enable the system to proactively begin the adaptation process in advance. This would enable the system to account for the *tactic latency* [3] when invoking additional resources and appropriately support increased user levels. Conversely, being able to anticipate reduced user loads would enable such a system to proactively begin to deactivate resources; thus enabling the system to reduce costs [4, 5]. Mechanical systems, such as self-driving cars and UASs would also benefit from the ability to more accurately predict the need for preventative maintenance which is critically important for cost and safety reasons. RNNs have proven to be powerful predictors of these types of complicated and correlated time series data. Furthermore, transfer learning, the kind of which we described earlier, could potentially empower these systems even further, allowing them to readily internally-used ANNs/RNNs in the event that new sensors become available or are corrupted/destroyed or when mechanical structures are modified or upgraded.

A major use of transfer learning has been for specialization. Gupta *et al.* used RNNs to make predictions of phenotypes – an RNN is trained to predict 20 different phenotypes based on clinical time series data and then this trained network is retrained to predict previously unseen phenotypes for which there are varying amounts of labeled data [6]. Zhang *et al.* made use of this same principle when predicting the remaining useful life (RUL) of various systems when data was scarce [7]. They show that training a model with related source data and then "specializing" the model by tuning it to available target data leads to significantly better performance when compared to a model only trained on target data. There have been a number of variants and improvements built on this general idea [7–10], also known as "pre-training". However, none of these works alter the structure of the ANNs being transferred.

Transfer learning with minor structural changes has been investigated, where mid-level features are transferred from a source task to a target task. In these cases, new output layers are fine-tuned on target data. Mun *et al.* derived mid-level features (parameters) from the hidden layers of an ANN trained on a source task and then transferred those features to a target task. The ANN for the target task was constructed by removing the output layer from the source ANN

and then augmented with two new additional hidden layers, *i.e.*, "adaptation layers", as well as a new output layer [11]. Along similar lines, Taylor *et al.* utilize NEAT to evolve inter-task mappings to translate trained neural network policies from a source to a target network [12]. Yang *et al.* took this concept further to designing designing networks for cross-domain, cross-application, and cross-lingual transfer settings [13]. Vierbancsics *et al.* used a different approach where an indirect encoding is evolved which can be applied to neural network tasks different structures [14].

Hinton *et al.* proposed the concept of "knowledge distillation", where an ensemble of teacher models are "compressed" (or the knowledge of which is transferred) to a single pupil model [15]. In their experiments a distilled single model performed nearly as well as the ensemble itself (also outperforming a single baseline model). Tang *et al.* have also conducted the converse of this experiment – they train a complex pupil model using a simpler teacher model [16]. Their findings demonstrate that knowledge gathered by a simple teacher model can effectively be transferred to a more complex pupil model (which has greater generalization capability). Deo *et al.* also concatenated mid-level features from two datasets as input yo a target feed forward network [17].

Ultimately, none of the above transfer learning strategies have involved any significant architectural changes to the networks being transferred. This work overcomes this limitation by proposing a neuro-evolutionary approach. Previously trained neural networks can be adapted to new tasks with different inputs and outputs by allowing the system to alter the input and output layers and only generating new minimal connectivity to these new components. Following this, a neuro-evolutionary process can then be used to adapt the internal structure of the ANN/RNN, reusing all applicable internal architectural components and weights. We have expanded the Evolutionary eXploration of Augmenting Memory Models (EXAMM) algorithm [18] to facilitate this kind of "adaptive structure transfer learning" and apply it to time series data prediction problems in two real world domains: power systems and aviation. Our results indicate that this process yields good RNN estimators faster when new sensors are added to a coal-fired power plant, resulting in predictors with more accurate predictive ability than those trained from scratch. We show that in some cases it is even possible to the transfer knowledge of the RNNs trained to predict engine parameters between aircraft and airframes with different engines and structural designs.

2 Evolutionary eXploration of Augmenting Memory Models (EXAMM)

EXAMM progressively evolves larger RNNs through a series of mutation operations and crossover (reproduction), as shown in Figure 1. Mutations can occur at the edge level, *e.g.*, *split edge*, *add edge*, *enable edge*, *add recurrent edge*, and *disable edge* operations, or as higher-level node mutations, *e.g.*, *add node*, and *split node*. The type of an added node is selected uniformly at random from a

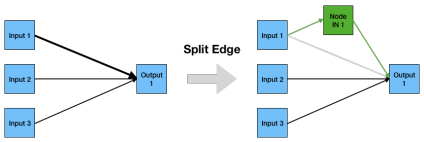
suite of simple neurons, Δ-RNN units [19], gated recurrent units (GRUs) [20], long short-term memory cells (LSTMs) [21], minimal gated units (MGUs) [22], and update gate RNN cells (UGRNNs) [23], which allows EXAMM to select for the best performing recurrent memory units. Recurrent edges are added with a time-skip selected uniformly at random, *i.e.*, $\sim U(1, 10)$. For more details on these operations we refer the reader to [18].

To speed up the neuro-evolution process, EXAMM utilizes an asynchronous, distributed computing strategy incorporating islands to promote speciation, which promotes both exploration and exploitation of the (massive) search space. A master process maintains a population for each island and generates new RNN candidate models from the islands in a round-robin manner. Candidate models are locally trained, via back-propagation of errors (backprop), upon request by workers. When a worker completes the training of an RNN, that RNN is inserted into the island it was generated from if and only if its fitness, *i.e.*, validation set mean squared error, is better than the worst fitness score in the island. The insertion of this RNN is then followed by removal of the worst RNN in the island. This asynchrony is particularly important as the generated RNNs will have different architectures, each taking a different amount of time to train. It allows the workers to complete the training of the generated RNNs at whatever speed they can, yielding an algorithm that is naturally load-balanced. Unlike synchronous parallel evolutionary strategies, EXAMM easily scales up to any number of available processors, allowing population sizes that are independent of processor availability. The EXAMM codebase has a multi-threaded implementation for multi-core CPUs as well as an MPI [24] implementation which allows EXAMM to operate using high performance computing resources.

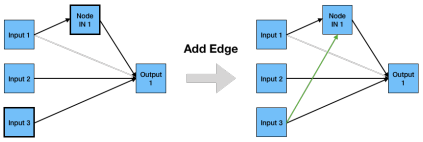
3 Adaptive Structure Transfer Learning

EXAMM typically initializes each island population with the minimal network configuration/topology possible for the given inputs and outputs, *i.e.*, each input has a single feedforward connection to each output (as shown in Figure 1a). Each island population starts with this minimal configuration, which is sent to the first worker requesting an RNN to be trained. Subsequent requests for work from that island create new RNN candidates from mutations of the minimal network until the population is full. After an island population is full, EXAMM will start generating new RNNs utilizing both intra-island crossover and mutation. When all islands are full, then EXAMM will generate new RNNs from inter-island crossover in addition to the aforementioned intra-island crossover and mutation.

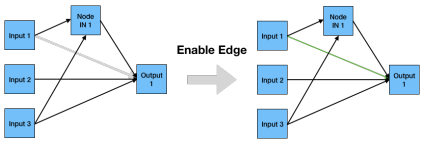
To enable transfer learning, this work extends EXAMM by allowing it to accept any ANN in place of the initial minimal network configuration. To fill the island populations, mutations of the provided initial ANN are made and intra-island and inter-island crossover continue as described earlier. In addition, we extend EXAMM to adjust the network in the face of different numbers of inputs and outputs, *a capability not yet demonstrated in any other transfer learning technique or application.* To accomplish this, EXAMM first adjusts the



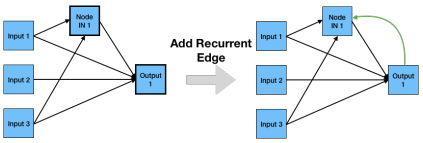
(a) The edge between Input 1 and Output 1 is split, adding two edges and a new node with innovation number (IN) 1.



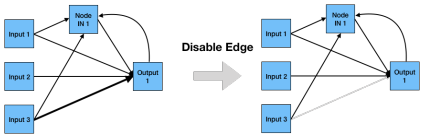
(b) Input 3 and Node IN 1 are selected to have an edge added between them.



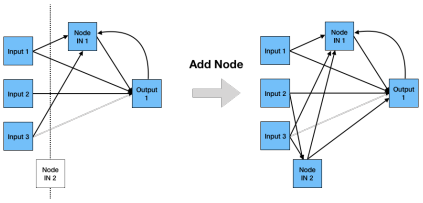
(c) The edge between Input 3 and Output 1 is enabled.



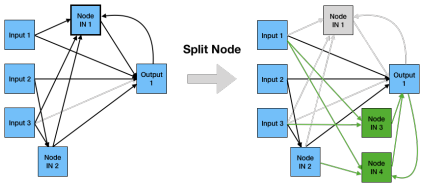
(d) A recurrent edge is added between Output 1 and Node IN 1



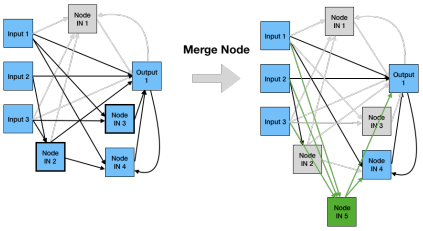
(e) The edge between Input 3 and Output 1 is disabled.



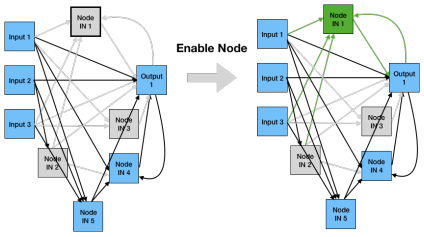
(f) A node with IN 2 is added at a random depth. Edges are added to randomly selected inputs/outputs.



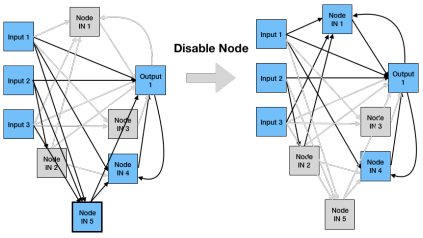
(g) Node IN 1 is split into Nodes IN 3 and 4, which get half, but at least 1, of the inputs and outputs.



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

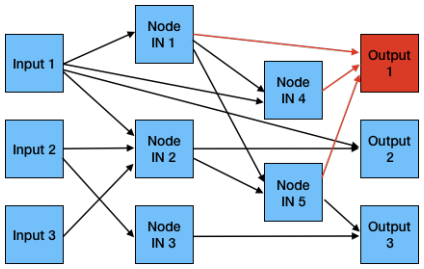


(i) Node IN 1 is selected to be enabled, along with all its input and output edges.

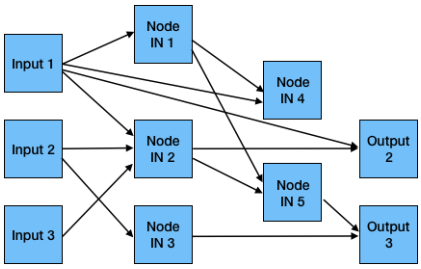


(j) Node IN 5 is selected to be disabled, along with all its input and output edges.

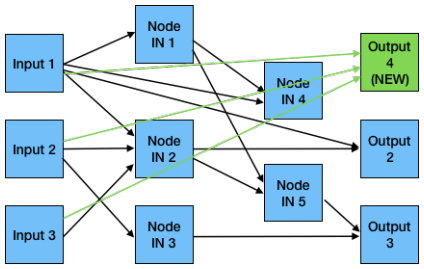
Fig. 1. Edge and node mutation operations.



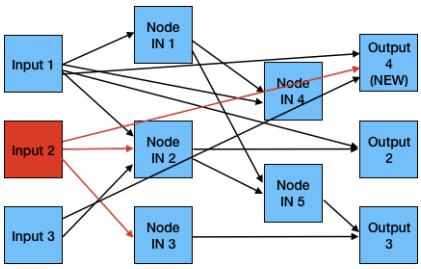
(a) Output 1 selected for removal.



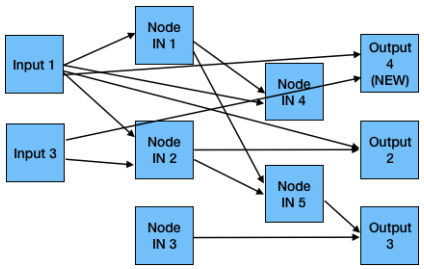
(b) Output 1 and connections are removed.



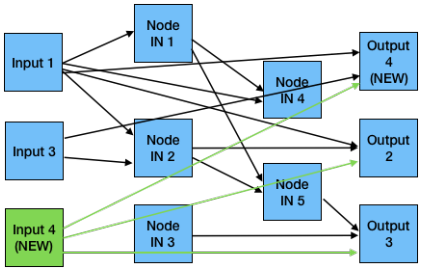
(c) Output 4 is added and connected to all inputs.



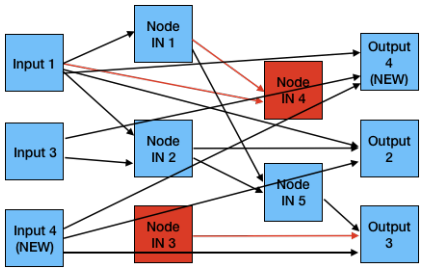
(d) Input 2 is selected for removal.



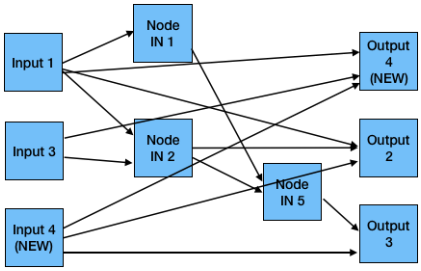
(e) Input 2 and connections are removed.



(f) Input 4 is added and connected to all outputs.



(g) Nodes 3 and 4 are not forward and backward reachable, they are selected for removal.



(h) Nodes 3 and 4 and their connections are removed.

Fig. 2. The adaptive structure transfer learning process.

network's output nodes and then its input nodes (see Figure 2 for an example of this process). If outputs need to be removed, their respective output nodes are removed along with any edges incoming to those particular output nodes. If outputs need to be added, they are connected to each existing input. Similarly, if inputs need to be removed, their respective input nodes are removed along with all of the respective outgoing edges. Finally, if input nodes need to be added, they are added and connected with a synapse connecting them to each output.

Removing inputs and outputs potentially disconnect parts of the RNN's graph such that they are either never reached from either a backprop pass or their output never effects the final output of the RNN. To safeguard against this, after a mutation or crossover operation is completed, EXAMM checks that all edges and nodes to ensure both forward reach-ability, *i.e.*, there is a path to the edge or node from an enabled input node, and their backward reach-ability, *i.e.*, there is a path from the node or to any output. Nodes and edges that are neither forward nor backward reachable (by any path of enabled nodes and edges) are disabled and no longer utilized in the backprop-adjustment process. They can however later be reconnected and enabled via EXAMM's mutation and crossover operations.

Following this, the EXAMM neuro-evolution process can continue as normal, adapting the transferred neural structures to the new inputs and outputs by adding new internal nodes and edges via EXAMM's mutation and crossover operations. This allows the learning process to be bootstrapped by reusing of the previously learned structure and potentially all of the (source) weights.

4 Transfer Learning Examples

This work examines two case studies of real-world transfer learning involving large-scale system sensor data. The first involves prediction of coal fired power plant parameters transferring RNNs trained before the addition of new sensor capabilities to new RNNs including the new sensor inputs. The second involves predicting engine parameters of three different aircraft with different airframes and engines. While the coal fired power plant data used in this study is proprietary and could not be made public, the aviation data has been made openly available to ensure reproducibility of our results and to encourage further study[1].

4.1 Coal-fired Power Plant Transfer Learning

The source dataset for training RNNs consisted of four data files, each containing approximately 24.5 hours of per-minute data readings from one of the coal plant's burners. This time series was characterized by 12 parameters: 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*

[1] https://github.com/travisdesell/exact/tree/master/datasets

Flow, 11) *Supplementary Fuel Flow*, and 12) *Main Flame Intensity*. From this dataset, the *Main Flame Intensity* parameter was the one selected for prediction due to its practical use in determining potential plant issues and performance optimization. It is mostly a product of internal burner conditions and parameters related to coal quality which makes it challenging to predict.

The target dataset consisted of another four data files, each containing 24.5 hours of per-minute data from the same burner, which also included fuel quality parameters from a newly-installed full stream elemental analyzer (FSEA) sensor. This data set added 8 new parameters: 1) *Base Acid Ratio*, 2) *Ash Content*, 3) *Na (Sodium) Content*, 4) *Fe (Iron) Content*, 5) *BTU*, 6) *Ash Flow*, 7) *Na (Sodium) Flow*, and 8) *Fe (Iron) Flow*. RNNs evolved by EXAMM on the source data were transferred to this dataset, making use of these additional 8 inputs.

4.2 Aviation Transfer Learning

The source data for the aviation transfer learning problem consisted of 36 flights gathered from the National General Aviation Flight Information Database[2]. This flight data comes from three different airframes, 12 of these flights are from Cessna 172 Skyhawks (C172s), 12 are from Piper PA-28 Cherokees (PA28s) and the last 12 from Piper PA-44 Seminoles (PA44s). Each of the 36 flights came from a different aircraft. The duration of each flight ranged from 1 to 3 hours, with data coming from 26 sensors for PA28s, 31 sensors for C172s, and 39 sensors for PA44s. These different airframes have significant design differences, as shown in Figure 3. C172s have a single engine and are "high wing", *i.e.*, wings are on the top, PA28s have a single engine and are "low wing", *i.e.*, wings are on the bottom, and PA44s have dual engines and are low wing.

(a) Cessna 172 Skyhawk (b) Piper PA-288 Cherokee (c) Piper PA-44 Seminole

Fig. 3. The three different airframes used for aviation transfer learning in this work (images under creative commons licenses).

These aircraft share 18 common sensor parameters: 1) *Altitude Above Ground Level (AltAGL)*, 2) *Barometric Altitude (AltB)*, 3) *GPS Altitude (AltGPS)*, 4) *Altitude Miles Above Sea Level (AltMSL)*, 5) *Fuel Quantity Left (FQtyL)*, 6)

[2] http://ngafid.org

Fuel Quantity Right (FQtyR), 7) *Ground Speed (GndSpd)*, 8) *Indicated Air Speed (IAS)*, 9) *Lateral Acceleration (LatAc)*, 10) *Normal Acceleration (NormAc)*, 11) *Outside Air Temperature (OAT)*, 12) *Pitch*, 13) *Roll*, 14) *True Airspeed (TAS)*, 15) *Vertical Speed (VSpd)*, 16) *Vertical Speed Gs (VSpdG)*, 17) *Wind Direction (WndDir)*, and 18) *Wind Speed (WndSpd).*

Since each of these airframes have different engines, they have different sets of engine sensor parameters. The C172 and PA44 have an absolute barometric pressure sensor which the PA28 does not. PA28s add the following 8 parameters: 1) *Engine 1 Exhaust Gas Temperature 1 (E1 EGT1)*, 2) *Engine 1 Exhaust Gas Temperature 2 (E1 EGT2)*, 3) *Engine 1 Exhaust Gas Temperature 3 (E1 EGT3)*, 4) *Engine 1 Exhaust Gas Temperature 4 (E1 EGT4)*, 5) *Engine 1 Fuel Flow (E1 FFlow)*, 6) *Engine 1 Oil Pressure (E1 OilP)*, 7), *Engine 1 Oil Temperature (E1 OilT)*, and 8) *Engine 1 Rotations Per minute (E1 RPM).*

C172s add the following 13 parameters: 1) *Absolute Barometric Pressure (BaroA)*, 2) *Engine 1 Cylinder Head Temperature 1 (E1 CHT1)*, 3) *Engine 1 Cylinder Head Temperature 2 (E1 CHT2)*, 4) *Engine 1 Cylinder Head Temperature 3 (E1 CHT3)*, 5) *Engine 1 Cylinder Head Temperature 4 (E1 CHT4)*, 6) *Engine 1 Exhaust Gas Temperature 1 (E1 EGT1)*, 7) *Engine 1 Exhaust Gas Temperature 2 (E1 EGT2)*, 8) *Engine 1 Exhaust Gas Temperature 3 (E1 EGT3)*, 9) *Engine 1 Exhaust Gas Temperature 4 (E1 EGT4)*, 10) *Engine 1 Fuel Flow (E1 FFlow)*, 11) *Engine 1 Oil Pressure (E1 OilP)*, 12) *Engine 1 Oil Temperature (E1 OilT)* and 13) *Engine 1 Rotations Per minute (E1 RPM).*

Finally, PA44s add the following 21 parameters: 1) *Absolute Barometric Pressure (BaroA)*, 2) *Engine 1 Cylinder Head Temperature 1 (E1 CHT1)*, 3) *Engine 1 Exhaust Gas Temperature 1 (E1 EGT1)*, 4) *Engine 1 Exhaust Gas Temperature 2 (E1 EGT2)*, 5) *Engine 1 Exhaust Gas Temperature 3 (E1 EGT3)*, 6) *Engine 1 Exhaust Gas Temperature 4 (E1 EGT4)*, 7) *Engine 1 Fuel Flow (E1 FFlow)*, 8) *Engine 1 Oil Pressure (E1 OilP)*, 9) *Engine 1 Oil Temperature (E1 OilT)*, 10) *Engine 1 Rotations Per minute (E1 RPM)*, 11) *Engine 1 Manifold Absolute Pressure (E1 MAP)*, 12) *Engine 2 Cylinder Head Temperature 1 (E1 CHT1)*, 13) *Engine 2 Exhaust Gas Temperature 1 (E1 EGT1)*, 14) *Engine 2 Exhaust Gas Temperature 2 (E1 EGT2)*, 15) *Engine 2 Exhaust Gas Temperature 3 (E1 EGT3)*, 16) *Engine 2 Exhaust Gas Temperature 4 (E1 EGT4)*, 17) *Engine 2 Fuel Flow (E1 FFlow)*, 18) *Engine 2 Oil Pressure (E1 OilP)*, 19) *Engine 2 Oil Temperature (E1 OilT)*, 20) *Engine 2 Rotations Per minute (E1 RPM)*, and 21) *Engine 2 Manifold Absolute Pressure (E1 MAP).*

The underlying task for these problems was to predict all the EGT parameters from the engines, so RNNs predicting on PA28 or C172 data would predict E1 EGT1-4, and RNNs predicting on PA44 data would predict both E1 EGT1-4 and E2 EGT1-4. We examine transferring RNNs trained and evolved by EXAMM on each of these 3 airframe sources to each of other airframes as a target. Inputs are added or removed by the evolutionary transfer process to make the most of available sensor inputs. Likewise, outputs are added or removed to predict the EGTs of either 1 or 2 engines depending on the airframe. This transfer problem is particularly interesting given that it requires evolved/trained RNNs

must learn to transfer useful knowledge between aircraft with different airframes and engines.

5 Results

5.1 EXAMM and Backpropagation Hyperparameters

Each EXAMM neuro-evolution run consisted of 4 islands, each with a population size of 10. New RNNs were generated via intra-island crossover (at rate of 20%), mutation at rate 70%, and inter-island crossover at 10% rate. All of EXAMM's mutation operations (except for *split edge*) were utilized, each chosen with a uniform 10% chance. EXAMM generated new nodes by selecting from simple neurons, Δ-RNN, GRU, LSTM, MGU and UGRNN memory cells (uniformly at random). Each EXAMM run on the plant data generated 2000 RNNs and each on the aviation data generated 4000 RNNs. Recurrent connections could span any time-skip between 1 and 10, chosen uniformly at random.

All RNNs were locally trained for 4 epochs with backpropagation through time (BPTT) [25] and stochastic gradient descent (SGD) using the same hyperparameters. SGD was run with a learning rate of $\eta = 0.001$, utilizing Nesterov momentum with $mu = 0.9$. No dropout regularization was used as it was shown in prior work to reduce performance when training RNNs for time series prediction [26]. For the LSTM cells that EXAMM could make use of, the forget gate bias had a value of 1.0 added to it (motivated by [27]). Otherwise, RNN weights were initialized by EXAMM's Lamarckian strategy. To prevent exploding gradients, gradient clipping (as described by Pascanu *et al.* [28]) was used when the norm of the gradient was above a threshold of 1.0. To improve performance on vanishing gradients, gradient boosting (the opposite of clipping) was used when the norm of the gradient went a threshold of 0.05.

5.2 Experiments

For each transfer experiment, EXAMM was repeated 10 times on the source data. For the coal plant data, data files 1-3 were used as training data and data file 4 was used as testing data. For the aviation data, the first 9 flights were used as training data, with the last 3 flights used as testing data. The RNNs with the best mean squared error (MSE) from each of the 10 EXAMM runs on the source data were used as the initial genomes for EXAMM when it was run on the target data.

Coal Plant Transfer Learning EXAMM was trained on the source coal plant data (without fuel quality parameters) 10 times as described above. EXAMM was then trained 10 times from scratch using the target data (the coal plant data including the fuel quality parameters), as well as 10 times using the best RNN generated from each of the 10 source data EXAMM runs. Figure 4 presents the convergence of the MSE of the best found RNNs of each EXAMM run started

from scratch on the target data, as well as the convergence of the 10 EXAMM runs starting from the RNNs transferred from the source runs.

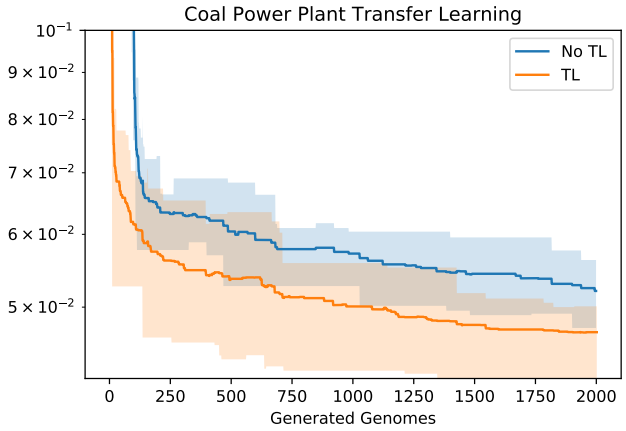
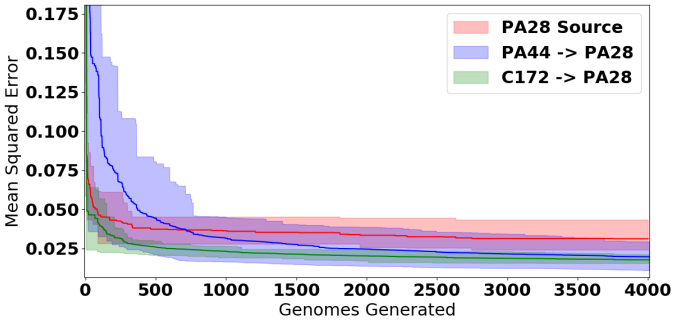


Fig. 4. Convergence rates (in terms of best MSE) for the EXAMM runs starting from scratch on the target coal plant data (No TL), and the EXAMM runs starting with RNNs transferred from the source data (TL).

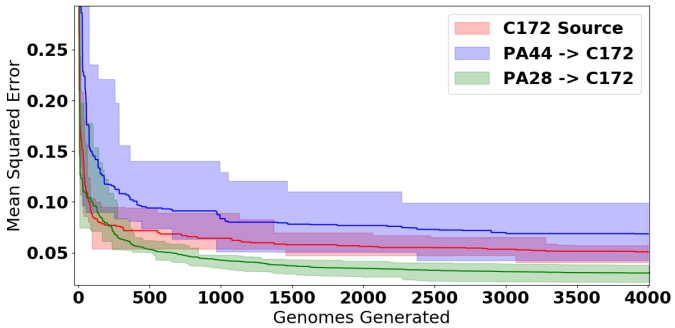
For the coal plant transfer learning scenario, the EXAMM runs using adaptive structure transfer learning showed significant improvements in performance. The MSE of predictions for the transfer learning runs started with lower MSEs and the non-transfer learning EXAMM runs never reached similar performance even after 2000 RNNs were evolved and trained.

Aviation Transfer Learning Figure 5 shows the performance of adaptive structure transfer learning using each of the airframes (C172, PA28 and PA44) as a source transferred to each other airframe as a target, *i.e.*, RNNs evolved and trained using EXAMM on C172 data were transferred to EXAMM runs with PA28 and PA44 data, PA28 source RNNs were transferred to C172 and PA44 targets, and PA44 source RNNs were transferred to C172 and PA28 RNNs. Since each airframe type had different input parameters, and the PA44 runs had additional outputs, this problem served as a useful case where adding and removing inputs and outputs from the RNN structures during transfer was necessary.

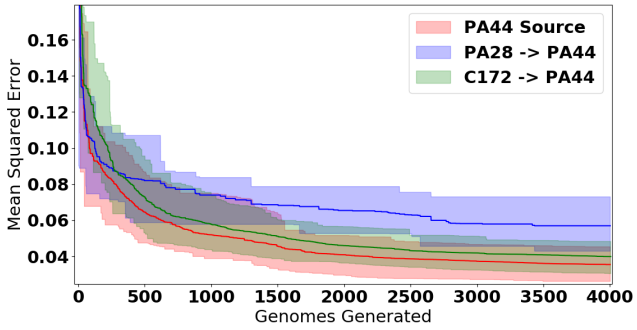
The aviation transfer learning problem proved to be more challenging than the coal plant problem. While the coal plant data all came from the same system, with the transfer target adding new sensor data; each of the flights used as training and testing data came from different aircraft, and the three different airframes transferred between had significant design and engine differences. Additionally, while the RNNs were predicting similar engine parameters they were transferred from different engine designs.



(a) Convergence rates (in terms of best MSE) for the EXAMM runs first using C172 and PA44 flights as source data and then transferred to PA28 data and EXAMM runs trained on PA28 flight data from scratch.



(b) Convergence rates (in terms of best MSE) for the EXAMM runs first using PA28 and PA44 flights as source data and then transferred to C172 data, and EXAMM runs trained on C172 flight data from scratch.



(c) Convergence rates (in terms of best MSE) for the EXAMM runs first using C172 and PA28 flights as source data and then transferred to PA44 data, and EXAMM runs trained on PA44 flight data from scratch.

Fig. 5. Aviation transfer learning experiments.

The simplest airframe with the least number of parameters (PA28) proved to be the easiest to transfer to. Using both C172 and PA44 source data, the transfer learning-based EXAMM runs swiftly outperformed the EXAMM runs trained on PA28 data from scratch while the runs from scratch never caught up to the transfer learning runs. Transferring from the most complicated PA44 source took longer to perform better than the runs from scratch, however it eventually reached similar performance to the C172 transfer learning runs. Interestingly, the runs from scratch never converged to the transfer learning runs, indicating that performing transfer learning may have improved the generalization ability of the evolved RNNs (in addition to faster learning).

For C172 predictions, using PA28 as a source for transfer had similar results as above with the transfer learning runs swiftly performing better than those trained from scratch on C172 data (with the from scratch runs never catching up). However, the transfer learning runs with PA44 data as a source never reached similar performance. This is potentially due to the difference between both the airframes, the number engines and a change in the number of outputs. PA44 transferred to PA28 may have performed better due to similarities in their engines and airframes, coming from the same manufacturer; whereas the C172 airframe and engine may have been too different for the transfer process to work.

Lastly, transfer learning struggled when PA28 and C172 data was used as a source with the most complicated PA44 data as a target. While the runs with C172 data came close to runs on the PA44 data from scratch they never quite reached the same performance. The PA28 source runs did not perform nearly as well, most likely due to the large difference in both inputs and outputs (26 inputs and 4 outputs vs. 39 inputs and 8 outputs). In general, this may indicate that adding outputs results in a more challenging knowledge transfer problem.

Group	Nodes			Edges			Recurrent Edges		
	min	max	avg	min	max	avg	min	max	avg
C172 Source	35	38.4	43	124	156.2	198	1	6.0	21
PA28 Source	30	32.2	38	103	124.1	179	0	4.1	21
PA44 Source	54	59.2	70	465	541.5	680	16	43.5	75
C172 to PA28	32	39.3	50	130	181.3	225	10	22.4	42
C172 to PA44	56	71.3	92	511	704.8	1014	28	83.6	171
PA28 to PA44	48	52.7	59	331	385.6	462	3	12.2	45
PA28 to C172	35	44.4	75	136	214.7	506	8	30.1	125
PA44 to PA28	46	75.7	128	351	716.9	1499	42	178.4	409
PA44 to C172	46	53.2	66	267	346.5	466	22	42.4	62

Table 1. Size of the evolved and transferred RNNs.

Table 1 provides a closer investigation of the evolved and transferred RNNs. The table demonstrates that the PA44 prediction problem is significantly more complicated than that of the PA28 and C172 airframes. The number of nodes, edges, and recurrent edges in the RNNs trained from scratch on the PA44 source

data (see the PA44 Source row) are an order of magnitude larger than those trained from scratch on the PA28 and C172 data. It appears that EXAMM runs with RNNs transferred from C172 and PA28 source data to PA44 target data were unable to reach the required complexity swiftly enough (causing issues).

6 Discussion

This work demonstrates initial findings in utilizing neuro-evolution as a strategy to speed up transfer learning of RNNs applied to time series data forecasting. This research is particularly novel in that it is not only the first work to utilize neuro-evoluton for transfer learning (to our knowledge) but also that it is the first transfer learning strategy that is capable of structural adaptation. Our approach modifies the input and output layers by adding and removing nodes as needed to transfer potential structures between different prediction tasks, continuing to evolve network structures beyond the initial network structure transfer.

The results are both promising and raise interesting directions for future work. With respect to data from a coal-fired power plant scenario where new sensor capabilities for determining fuel quality became available, RNNs trained on prior data without the new sensors serving as the transfer source were able to evolve and train on the target task faster in the short term and also continued to outperform RNNs only trained on the target dataset with the new sensor data; indicating that the transfer learning process improves generalization. Furthermore, using aviation data from three different airframes, it was shown that it is possible to successfully transfer knowledge from RNNs trained on aircraft with different airframes and engines when predicting engine parameters. This was particularly impressive given that we were transferring predictive ability from engine parameters from *different engine designs and different airframe designs*. When conducting transfer learning from the most complicated airframe type (PA44) to simpler airframe types (PA28 and C172) or between the simpler airframe types (PA28 and C172) we showed similar improvement in generalization – the RNNs generated via our transfer learning process performed better than those trained from scratch on target data.

Nonetheless, the aviation data did present some challenges. Transferring to the more complicated PA44 airframe with two engines proved to be challenging. The experiments transferring to PA44 as a target did not perform as well as those trained on the PA44 data from scratch. An analysis of the transferred and evolved RNNs showed that this could possibly be due to the additional target outputs from the additional engine compounded by the additional complexity of the prediction task (RNNs evolved for the PA44 data were an order of magnitude larger in size). Determining if it is possible to transfer to a more complicated system remains an area of future work and might potentially be enabled by utilizing different strategies for adding/removing the output/input nodes.

In conclusion, this work presents a novel and promising direction for neuro-evolution research. Compared to other transfer learning strategies that simply reuse weights and architectures on target data with minimal modification, using

a adaptive structure methodology driven by neuro-evolution allows modification of inputs in the case of sensor data and could even allow the modification of input layer size if applied to the adaptation of convolutional neural networks. In addition, this approach allows the internal structure of the transferred networks to continue to be adapted, providing even better performance. This work provides a new opportunity to allow ANNs to quickly learn when exposed to different data and more quickly transfer learned knowledge to across prediction tasks.

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. Mikolov, T., Sutskever, I., Chen, K., Corrado, G.S., Dean, J.: Distributed representations of words and phrases and their compositionality. In: Advances in neural information processing systems. pp. 3111–3119 (2013)
2. Devlin, J., Chang, M.W., Lee, K., Toutanova, K.: Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805 (2018)
3. Moreno, G.A., Cámara, J., Garlan, D., Schmerl, B.: Proactive self-adaptation under uncertainty: A probabilistic model checking approach. In: Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering. pp. 1–12. ESEC/FSE 2015, ACM, New York, NY, USA (2015), http://doi.acm.org/10.1145/2786805.2786853
4. Moreno, G.A.: Adaptation Timing in Self-Adaptive Systems. Ph.D. thesis, Carnegie Mellon University (2017)
5. Palmerino, J., Yu, Q., Desell, T., Krutz, D.: Accounting for tactic volatility in self-adaptive systems for improved decision-making. In: Proceedings of the 34th ACM/IEEE International Conference on Automated Software Engineering. ASE 2019, ACM, New York, NY, USA
6. Gupta, P., Malhotra, P., Vig, L., Shroff, G.: Transfer learning for clinical time series analysis using recurrent neural networks. arXiv preprint arXiv:1807.01705 (2018)
7. Zhang, A., Wang, H., Li, S., Cui, Y., Liu, Z., Yang, G., Hu, J.: Transfer learning with deep recurrent neural networks for remaining useful life estimation. Applied Sciences 8(12), 2416 (2018)
8. Yoon, S., Yun, H., Kim, Y., Park, G.t., Jung, K.: Efficient transfer learning schemes for personalized language modeling using recurrent neural network. In: Workshops at the Thirty-First AAAI Conference on Artificial Intelligence (2017)
9. Zarrella, G., Marsh, A.: Mitre at semeval-2016 task 6: Transfer learning for stance detection. arXiv preprint arXiv:1606.03784 (2016)
10. Mrkšić, N., Séaghdha, D.O., Thomson, B., Gašić, M., Su, P.H., Vandyke, D., Wen, T.H., Young, S.: Multi-domain dialog state tracking using recurrent neural networks. arXiv preprint arXiv:1506.07190 (2015)

11. Mun, S., Shon, S., Kim, W., Han, D.K., Ko, H.: Deep neural network based learning and transferring mid-level audio features for acoustic scene classification. In: 2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). pp. 796–800. IEEE (2017)
12. Taylor, M.E., Whiteson, S., Stone, P.: Transfer via inter-task mappings in policy search reinforcement learning. In: Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems. p. 37. ACM (2007)
13. Yang, Z., Salakhutdinov, R., Cohen, W.W.: Transfer learning for sequence tagging with hierarchical recurrent networks. arXiv preprint arXiv:1703.06345 (2017)
14. Verbancsics, P., Stanley, K.O.: Evolving static representations for task transfer. Journal of Machine Learning Research 11(May), 1737–1769 (2010)
15. Hinton, G., Vinyals, O., Dean, J.: Distilling the knowledge in a neural network. arXiv preprint arXiv:1503.02531 (2015)
16. Tang, Z., Wang, D., Zhang, Z.: Recurrent neural network training with dark knowledge transfer. In: 2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). pp. 5900–5904. IEEE (2016)
17. Deo, R.V., Chandra, R., Sharma, A.: Stacked transfer learning for tropical cyclone intensity prediction. arXiv preprint arXiv:1708.06539 (2017)
18. Ororbia, A., ElSaid, A., Desell, T.: Investigating recurrent neural network memory structures using neuro-evolution. In: Proceedings of the Genetic and Evolutionary Computation Conference. pp. 446–455. GECCO '19, ACM, New York, NY, USA (2019), http://doi.acm.org/10.1145/3321707.3321795
19. Ororbia II, A.G., Mikolov, T., Reitter, D.: Learning simpler language models with the differential state framework. Neural Computation 0(0), 1–26 (2017), https://doi.org/10.1162/neco_a_01017, pMID: 28957029
20. Chung, J., Gulcehre, C., Cho, K., Bengio, Y.: Empirical evaluation of gated recurrent neural networks on sequence modeling. arXiv preprint arXiv:1412.3555 (2014)
21. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural Computation 9(8), 1735–1780 (1997)
22. Zhou, G.B., Wu, J., Zhang, C.L., Zhou, Z.H.: Minimal gated unit for recurrent neural networks. International Journal of Automation and Computing 13(3), 226–234 (2016)
23. Collins, J., Sohl-Dickstein, J., Sussillo, D.: Capacity and trainability in recurrent neural networks. arXiv preprint arXiv:1611.09913 (2016)
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. Werbos, P.J.: Backpropagation through time: what it does and how to do it. Proceedings of the IEEE 78(10), 1550–1560 (1990)
26. 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)
27. Jozefowicz, R., Zaremba, W., Sutskever, I.: An empirical exploration of recurrent network architectures. In: International Conference on Machine Learning. pp. 2342–2350 (2015)
28. Pascanu, R., Mikolov, T., Bengio, Y.: On the difficulty of training recurrent neural networks. In: International Conference on Machine Learning. pp. 1310–1318 (2013)