Received October 22, 2019, accepted November 12, 2019, date of publication November 20, 2019,
date of current version December 23, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2954795

An Evolutionary Approach to Compact DAG Neural Network Optimization

CARTER CHIU[1] **AND JUSTIN ZHAN**[2]
[1]Department of Computer Science, University of Nevada Las Vegas, Las Vegas, NV 89154, USA
[2]Department of Computer Science and Computer Engineering, University of Arkansas, Fayetteville, AR 72701, USA

Corresponding author: Justin Zhan (jzhan@uark.edu)

This work was supported in part by the Department of Defense under Grant 911NF1910482, and in part by the Arkansas Research Alliances.

ABSTRACT Neural networks are the cutting edge of artificial intelligence, demonstrated to reliably outperform other techniques in machine learning. Within the domain of neural networks, many different classes of architectures have been developed for various tasks in specific subfields, as well as a multitude of diversity in the way of activation functions, loss functions, and other such hyperparameters. These networks are often large and computationally expensive to train and deploy, restricting their utility. Furthermore, the fundamental theory behind the effectiveness of particular network architectures and hyperparameters are often not well understood, and as such, practitioners frequently resort to trial-and-error techniques to optimize their model performance. To address these concerns, we propose the use of compact directed acyclic graph neural networks (DAG-NNs) and an evolutionary approach for automating the optimization of their structure and parameters. Our experimental results demonstrate that our approach consistently outperforms conventional neural networks, even while employing fewer nodes.

INDEX TERMS Evolutionary algorithms, DAG neural networks, compact neural networks, artificial intelligence.

I. INTRODUCTION

Artificial neural networks (ANNs) are a class of systems in machine learning modeled after the structure of the brain. In recent years, ANNs and particularly deep neural networks (DNNs) have risen to the forefront of the artificial intelligence community, attracting the lion's share of interest among academic and industry professionals owing to a variety of factors. Neural networks find applications across many disciplines within and outside computer science – computer vision and medical diagnosis are two such domains with profound and well-publicized advancements – and they consistently achieve the best performance among statistical models and machine learning techniques across various domains and applications. The existence of numerous well-supported software libraries for building and training DNNs make them a popular choice for data scientists. And perhaps most intriguingly, their underlying theoretical foundations are not well understood, lending promise to groundbreaking discoveries that may further improve their performance.

Research in neural networks has yielded a wide array of variations and configurations, suited to different applications and dataset conditions. One such design consideration is the network architecture, which determines the arrangement and manner in which nodes connect to and transmit information between each other. For instance, a shallow neural network (SNN) consists of an input layer, hidden layer, and output layer, where each layer consists of a number of nodes, each of which are connected to every node in the next layer. A deep neural network contains a greater number of hidden layers. Meanwhile, convolutional (CNN) and recurrent (RNN) neural networks have alternative architectures that make them particularly favorable for computer vision and natural language processing tasks, respectively. These frequently complex and massive network structures require expensive hardware to train and to run, limiting their use in applications such as embedded systems. Another issue afflicting the training of such networks is the great number of configurable hyperparameters which influence the way a neural network learns and adjusts to information. Hyperparameter tuning is frequently a process of trial and error, as current understanding of the relationship between hyperparameters

The associate editor coordinating the review of this manuscript and approving it for publication was Wei Liu.

and training data is limited. A practitioner may spend hours or days adjusting hyperparameters to obtain the best performance, an often frustrating task.

This paper proposes a solution to these difficulties by applying the principles of evolutionary algorithms to optimize the structure and parameters of a compact artificial neural network. Evolutionary algorithms (EAs) are a class of optimization algorithms inspired by another biological process – natural selection. Borrowing concepts from evolution such as crossover and mutation, EAs are a particularly clever way of exploring a search space for an optimal solution. We reimagine a neural network's structure as a directed acyclic graph, the weights as weighted edges, and biases as node attributes. The proposed evolutionary algorithm interprets networks as individuals in a population, eliminating poorly performing networks through selection and generating new ones through crossover and mutation. The result is a framework for training and optimizing artificial neural networks which exhibit high accuracy with far fewer nodes.

The remainder of our paper is presented as follows. Section 2 contains a survey of existing research in neural network architectures, neural architecture search, and compression. In Section 3 we present our proposed evolutionary framework, including the representation of the artificial neural network and the evolutionary algorithm for manipulating said representation. Then in Section 4, we apply our approach to real-world datasets representing a diverse array of applications and learning tasks, comparing performance against conventional neural networks. Finally, we conclude our paper in Section 5.

II. RELATED WORK

A. NETWORK ARCHITECTURES AND HYPERPARAMETER SELECTION

The history behind the development of novel architectures and hyperparameters is one characterized by experimentation just as much as theory. However, the development of novel architectures and hyperparameters is often conducted on the basis of experimentation rather than theory. And the process of selecting architectures and hyperparameters is frequently a process of trial and error on the part of the practitioner.

The discovery and adoption of activation functions typifies the largely empirical nature of deep learning research. Hahnloser *et al.* [1] introduced the celebrated rectified linear unit (ReLU) activation function, providing mathematical support, but it became popularized through [2] by primarily experimental means. Ramachandran *et al.* [3] took the empirical derivation of activation functions to an extreme by conducting a search through a search space of composed unary and binary functions. Through this method, they discovered an activation function which they termed Swish, finding that it consistently outperformed ReLU with all other parameters held constant. A similar trajectory describes the direction of research involving DNN architectures. LeCun *et al.* provide an overview of deep learning in [4], describing the development of architectures as an empirical process.

This is not to say that the theoretical foundations of DNNs have been neglected. With deep learning at the forefront of computing research, strides are indeed being made on the fundamental theory front. In an influential paper, Lin *et al.* [5], discuss the effectiveness of deep neural networks from a statistics and physics perspective. Similarly, Mhaskar *et al.* [6] describe the long-standing discussion over shallow versus deep layered architectures, providing theoretical insight into the particular conditions that allow deep networks to learn better representations than shallow ones. However, the bleeding edge of deep learning research builds so rapidly upon the repository of architectures and hyperparameters that pioneering researchers and practitioners are inevitably faced with experimentation and trial and error.

B. NEURAL ARCHITECTURE SEARCH

Architecture optimization of neural networks is a well-studied research area, with many different approaches having been adopted. Because the theory behind optimal structure is not sufficiently explored, most research attempts to experimentally determine an optimal network configuration through search algorithms, collectively known as neural architecture search (NAS). Elsken *et al.* [7] conduct a survey on such approaches, categorized along three dimensions: search space, search strategy, and performance estimation strategy.

Doering *et al.* [8] present an algorithm for architecture optimization utilizing the A* search algorithm, finding success when applied to the Pima Indians dataset. Meanwhile, Yang *et al.* [9] develop another such algorithm tailored to large dataset applications by introducing the concept of sparse representation pruning (SRP), and Ihme *et al.* [10] use a pattern search optimization algorithm, applying the approach to chemical systems approximation. In a departure from most approaches, Shirakawa *et al.* [11] utilize a probability distribution to generate neural network structures and optimize the parameters of the probability distribution rather than that of the structure, adopting an indirect strategy for structure optimization. The primary advantage of their approach is computational efficiency. A recent noteworthy implementation of neural architecture search is NASNet [12]. Its key contribution is the development of the "NASNet search space" which enables efficient architecture search over smaller datasets and transferability to larger ones. Employing this strategy, a NASNet found on CIFAR-10 and transferred to ImageNet achieved state-of-the-art accuracy on both datasets.

The concept of directed acyclic graph neural networks (DAG-NNs) have entered the nomenclature, representing a generalized network for architecture optimization. Yang and Ramanan [13] apply a DAG architecture to CNNs, obtaining exceptional accuracy with image classification tasks. In a similar fashion, Shuai *et al.* [14] propose DAG recurrent neural networks and apply their approach to scene labeling, achieving state-of-the-art results by outperforming well-known convolutional neural networks.

1) EVOLUTIONARY APPROACHES

DAG-NNs are particularly receptive to evolutionary approaches as a search heuristic for optimizing structure. Perhaps the most well-known contribution to this area is from Stanley and Miikkulainen in "Evolving Neural Networks through Augmenting Topologies" [15], who offer and demonstrate the application of genetic algorithms for evolving fluid network topologies. Shinozaki and Watanabe [16] borrow these principles by proposing evolutionary algorithms for NN topology optimization with a specific interest in speech processing applications. They utilize both a classic genetic algorithm [17] as well as a covariance matrix adaptation evolution strategy (CMA-ES) [18]. Husken *et al.* [19] adopt another evolutionary approach in the form of a covariance matrix adaptation strategy described in [20], empirically finding their structure optimization method yields models with better performance than that of a standard fully connected neural network.

The use of evolutionary algorithms is alive and well in cutting edge research. Miikkulainen *et al.* [21] describe a framework termed CoDeepNEAT for applying evolutionary methods to optimizing ANN architectures. They take the additional step of extending CoDeepNEAT to recurrent neural networks by evolving its component LSTM nodes. Experiments on an image captioning problem determine that the approach outperforms baseline methods in RNN applications. Real *et al.* [22] employ regularized evolution, a variant of evolutionary algorithms which modifies the tournament selection operator to favor newer individuals. Searching over the NASNet space from [12], they evolved AmoebaNet-A, which established a new state of the art on ImageNet.

C. NETWORK COMPRESSION

The unwieldy size and complexity of network architectures complicates the training process and limits the systems which may deploy such an ANN. For devices with lower computational power, the study of network compression to obtain smaller networks while minimizing performance loss is valuable. For instance, Ma *et al.* [23] considers a method for compression, achieving a speedup factor of 2 on VGG-16. Likewise, Cheng *et al.* [24] and Choi *et al.* [25] describe methods for compressing CNNs with quantization, obtaining considerable compression and acceleration factors while enabling practical usage on mobile devices. Other works include [26]–[44]

III. EVOLUTIONARY FRAMEWORK

Here, we develop a generalized concept of a neural network in the context of an evolutionary algorithm. Then, we define the rules and parameters with which we evolve a neural network for the purposes of learning.

A. DIRECTED ACYCLIC GRAPH NEURAL NETWORKS

The vast majority of ANN architectures utilize a fully-connected layered structure. This composition enables rapid GPU computation, which is a particular concern for larger networks and their applications, but it also imposes limitations that affect network learning capacity. Directed Acyclic Graph Neural Networks (DAG-NNs) relax these structural restrictions, providing the greatest flexibility for a feedforward neural network architecture. This enables a higher ceiling for representation learning when compared to traditional neural networks.

TABLE 1. Symbols describing a DAG-NN.

Symbol	Definition
$id(n)$	The index corresponding to a node n
I	The set of input nodes
H	The set of hidden nodes
O	The set of output nodes
$\mathbf{b}$	The vector of bias values with length $\lvert H\rvert + \lvert O\rvert$

A DAG-NN borrows the notion of a directed acyclic graph (DAG) consisting of nodes linked by directed edges and the absence of cycles. The cascading structure of a typical fully-connected network, in which nodes in layer n must have a directed edge to every node in layer $n + 1$, is discarded in favor of allowing connections between any node. The acyclic property is a necessary consequence of the feedforward neural network model, in which all information moves forward from the input to output nodes. In addition, two more properties apply to DAG-NNs:

1) Input nodes may not contain directed edges to other input nodes
2) Output nodes may not contain directed edges to other output nodes

Property (1) is trivial since input nodes are sourced directly from a dataset, while (2) enforces the intuition that output nodes are endpoints in the network. At this moment it would be worthwhile to assert the following lemma, referring to the notation described in Table 1:

Lemma 1: The number of possible edges in a DAG-NN is $(|I| + |H|)(|H| + |O|) - \binom{|H|+1}{2}$*, and the number of possible structural configurations is* $2^{(|I|+|H|)(|H|+|O|)-\binom{|H|+1}{2}}$.

Proof: We know that edges can only originate from input or hidden nodes according to Property (2), and similarly edges can only terminate from hidden or output nodes pursuant to Property (1), so at most the number of edges is $(|I| + |H|)(|H| + |O|)$. To prevent cycles, we must consider that the threat of a cycle can only arise from a node for which both inbound and outbound edges are possible, which in our case applies only to hidden nodes. To resolve, affix an order to the hidden nodes and disallow edges that self-loop or travel upstream – that is, to an node at or earlier in an ordering. It is known that any DAG can be ordered in this manner. There are $\binom{|H|}{2}$ upstream edges $+ |H|$ self-loops $= \binom{|H|+1}{2}$ total illegal edges, leaving $(|I|+|H|)(|H|+|O|) - \binom{|H|+1}{2}$ possible edges in a DAG-NN. Given that each edge may be present or absent

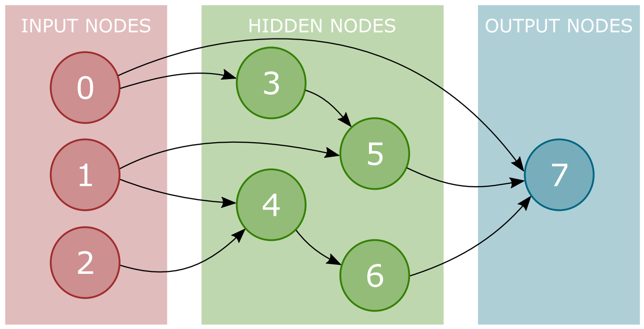
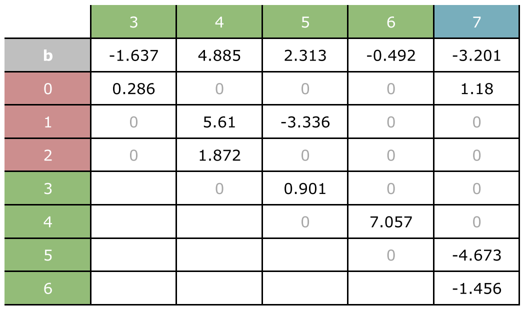


FIGURE 1. **An example phenotype for a Directed Acyclic Graph Neural Network (DAG-NN).**



	3	4	5	6	7
b	-1.637	4.885	2.313	-0.492	-3.201
0	0.286	0	0	0	1.18
1	0	5.61	-3.336	0	0
2	0	1.872	0	0	0
3		0	0.901	0	0
4			0	7.057	0
5				0	-4.673
6					-1.456

FIGURE 2. **A neural matrix for the DAG-NN depicted in Figure 1.**

in a configuration, we arrive at $2^{(|I|+|H|)(|H|+|O|)-\binom{|H|+1}{2}}$ possible structural configurations. $\square$

Figure 1 depicts a DAG-NN with 3 input nodes, 4 hidden nodes, and 1 output node. In evolutionary terminology we describe this as a phenotype – the conceptual manifestation of a DAG-NN. According to Lemma 1, this allows for 25 potential edges and 2^{25} possible configurations. A conventional shallow neural network with these hyperparameters would have 16 edges and a fixed configuration. We can observe that reduced restrictions on node connections allow for more diverse networks than typically permitted in traditional network architectures. The directed edge from input node `0` to output node `7` is a particular novelty introduced by a DAG architecture. It is also important to note that the figure presents only structural information and omits weights and biases for clarity, which would otherwise be a critical component of a DAG-NN and its phenotype.

B. NETWORK REPRESENTATION

For utilization in an evolutionary algorithm, a network must be encoded with a representation amenable to evolutionary processes such as crossover and mutation. In other words, we must transform the conceptual representation of a DAG-NN into a computational one.

It is known that any directed acyclic graph possesses at least one topological ordering where every node contains directed edges exclusively to nodes later in the ordering. Therefore, the nodes in a DAG can be indexed and represented by an ordered list, with input nodes occupying the lowest numbering and output nodes occupying the highest. For consistency with the literature, we index the first node in the ordering with value `0`. It is also essential to identify input, hidden, and output nodes due to the restrictions imposed on their connections as enumerated in the previous subsection. Additionally, weights and biases compose the parameters of the neural network and must also be encoded; weights are associated with edges while biases are associated with (non-input) nodes. This necessitates the formulation of a data structure which incorporates this information alongside the network structure composition. Toward this end, we propose a data structure which we refer to as a neural matrix.

Assume for notational simplicity that $id(\mathbf{b}) = -1$. Then the neural matrix contains real values for all mappings $f \to t$ where $f \in (I \cup H \cup \mathbf{b})$, $t \in (H \cup O)$, and $id(f) < id(t)$. A mapping from node to node indicates a weight value along the directed edge, while a mapping from special symbol $\mathbf{b}$ to a node indicates the bias value for the node. Lemma 1 dictates the total number of weight values in the neural matrix, with an additional $|\mathbf{b}|$ bias values. Consider Figure 2, which depicts the matrix for the phenotype illustrated in Figure 1. The neural matrix corresponds to the genotype of the DAG-NN, a representation manipulable by an evolutionary algorithm. With this representation, we are now able to define the algorithm in detail.

C. EVOLUTIONARY ALGORITHM

An evolutionary algorithm is an optimization heuristic which draws inspiration from the evolutionary processes found in nature and studied in biology. It borrows key concepts such as the doctrine of "survival of the fittest" to select well-performing solutions according to some evaluation of fitness, and enables solutions to transmute through the similarly-inspired concepts of crossover and mutation. In this subsection, we describe each component of an algorithm developed and tailored specifically for the neural matrix representation of a DAG-NN.

1) INITIALIZATION

An initial population of primitive DAG-NNs is necessary to begin the algorithm. The population size is configurable, with larger populations enabling higher network diversity at some cost in training time. For each network, certain characteristics are dictated by the dataset, namely the number of input and output nodes. In addition, the number of hidden nodes may be set by the practitioner to conform to the desired level of compactness and other computational requirements. With the amount of nodes in each class determined, the dimensions and number of values contained within the neural matrix genotype are established by Lemma 1. This leaves the values of the neural matrices to be initialized, and as the starting collection of networks represents a population with no evolutionary pressure applied, we can randomly generate values according to stochastic processes. The initialization process is outlined by Algorithm 1. With an unbiased initial edge probability $P(ie)$ of 0.5, each potential network configuration has an equal

Algorithm 1 initialize()

```
Require: population size $ps$, initial edge probability $P(ie)$,
    initial weight standard deviation $iwSD$, initial bias standard deviation $ibSD$
1:  $P \leftarrow \{\}$
2:  for $i = 1 \ldots ps$ do
3:      $n \leftarrow$ new network()
4:      for each potential edge $(f, t)$ do
5:          if $r$ sampled from $\mathcal{U}(0, 1) < P(ie)$ then
6:              $n[f, t] \leftarrow r$ sampled from $\mathcal{N}(0, iwSD)$
7:          else
8:              $n[f, t] \leftarrow 0$
9:          end if
10:     end for
11:     for each bias $b$ do
12:         $n[b, b] \leftarrow r$ sampled from $\mathcal{N}(0, ibSD)$
13:     end for
14:     $P \leftarrow P \cup n$
15: end for
16: return $P$
```

probability of being generated. Then, for each weight and bias in the network, a number is sampled from the Gaussian distributions $\mathcal{N}(0, iwSD)$ and $\mathcal{N}(0, ibSD)$ respectively, with $iwSD$ and $ibSD$ being constants supplied by the practitioner.

An alternative to hand-selecting a constant for $iwSD$ is to use an initialization scheme that determines a value for $iwSD$ as a function of network structure. A theoretically-supported and widely used initialization scheme for conventional ReLU-activated neural networks is He *et al.* initialization [45]. The scheme prescribes

$$iwSD = \sqrt{\frac{2}{in}},$$

where in is the number of input connections to a node. For a fully-connected network, in is equal to the number of nodes in the previous layer, and thus $iwSD$ is defined on a per-layer basis. We make two observations:

1) In a DAG-NN, the value of in for non-input nodes is bounded above by $min(id(n), |I| + |H|)$, which we denote by $in_{\max}$.
2) The expected value of in upon initialization is $P(ie) \cdot in_{\max}$.

From this, we can adapt He *et al.* initialization for a DAG-NN as follows:

$$iwSD = \sqrt{\frac{2}{P(ie) \cdot in_{\max}}}.$$

Once this step is completed, the population is considered initialized and the generation loop may begin.

2) SELECTION

Algorithm 2 details the steps for iterative generation computation. Each network n is scored based on a fitness function

Algorithm 2 nextGeneration()

```
Require: population $P$, population size $ps$
1: $P' \leftarrow \{\}$
2: sort $P$ by $\mathcal{F}$ descending
3: for $i = 1 \ldots ps$ do
4:     $a \leftarrow \lfloor (r \text{ sampled from } \mathcal{U}(0, 1))^2 \rfloor$
5:     $b \leftarrow \lfloor (r \text{ sampled from } \mathcal{U}(0, 1))^2 \rfloor$
6:     $n \leftarrow$ mutate(crossover($P[a]$, $P[b]$))
7:     $P' \leftarrow P' \cup n$
8: end for
9: return $P'$
```

$\mathcal{F}$ that compares network-predicted values with actual values from a training set. Predicted values are computed via forward propagation of training inputs through the DAG-NN. This fitness function is analogous to the loss function in a gradient-based ANN, and as such, the conventions for selecting a loss function apply here as well. For instance, consider a fitness function for regression tasks, for which we employ the negative mean squared error (MSE) of the m predicted values against actual values over a set of some l training examples:

$$\mathcal{F}(n) = -\sum_{i=0}^{l}\sum_{j=0}^{m} \frac{(\text{predicted}[i][j] - \text{actual}[i][j])^2}{l \times m}$$

The negation preserves the notion that a higher score indicates higher fitness.

Once the scores are calculated, selection is performed. In our algorithm, we utilize a efficient rank selection procedure. First, the population of DAG-NNs are sorted by fitness in descending order. Then, pairs of values are sampled from $\mathcal{U}(0, 1)$. These values are passed through a quadratic transformation and scaled to the population size ps. The resultant values are floored and correspond to the indices of the sorted DAG-NNs. In this manner, lower indices – and therefore fitter networks – are more likely to be selected, while less fit networks still retain a nontrivial chance for selection, stymieing premature convergence to local minima. This procedure is computationally efficient, with individual selection done in strictly $\mathcal{O}(1)$ time and requiring no comparison of fitness or any pre-processing beyond sorting.

3) CROSSOVER AND MUTATION

When two DAG-NNs are selected to reproduce, the resultant DAG-NN undergoes two processes to obtain its values. The first is crossover, which combines existing values from each contributing genotype. We opt for single-point crossover for its simplicity: a random node index i is selected, and the weights and biases from nodes $0 \ldots i$ in the first network are merged with the complement set of nodes from the second network. Once crossover is completed, we perform random mutation on the new network. Algorithm 3 describes the mutation process, which bears resemblance to

TABLE 2. Dataset characteristics.

Dataset	Attributes	Instances	Learning Task
Pima Indians Diabetes	8	768	Binary Classification
Air Quality	12	7674	Regression
Frog Calls	22	7195	Multiclass Classification
Abalone	10	4177	Regression
White Wine Quality	11	4898	Regression
Heart Disease	13	297	Binary Classification

Algorithm 3 Mutate()

```
Require: network $n$, weight mutation probability $P(wm)$, bias mutation probability $P(bm)$, weight mutation standard deviation $wmSD$, bias mutation standard deviation $bmSD$, edge addition probability $P(e+)$, edge removal probability $P(e-)$
1: $n' \leftarrow n$
2: for each edge $(f, t)$ do
3:     if $r$ sampled from $\mathcal{U}(0, 1) < P(wm)$ then
4:         $n'[f, t] \leftarrow n'[f, t]+r$ sampled from $\mathcal{N}(0, wmSD)$
5:     end if
6: end for
7: for each bias $b$ do
8:     if $r$ sampled from $\mathcal{U}(0, 1) < P(bm)$ then
9:         $n'[\mathbf{b}, b] \leftarrow n'[\mathbf{b}, b] + r$ sampled from $\mathcal{N}(0, bmSD)$
10:    end if
11: end for
12: for each potential edge $(f, t)$ do
13:     if $n'[f, t] = 0$ then
14:         if $r$ sampled from $\mathcal{U}(0, 1) < P(e+)$ then
15:             $n'[f, t] \leftarrow r$ sampled from $\mathcal{N}(0, wmSD)$
16:         end if
17:     else
18:         if $r$ sampled from $\mathcal{U}(0, 1) < P(e-)$ then
19:             $n'[f, t] \leftarrow 0$
20:         end if
21:     end if
22: end for
23: return $n'$
```

initialization. Weights and biases mutate with probabilities $P(wm)$ and $P(bm)$ and standard deviations $wmSD$ and $bmSD$, each respectively, while edges can be added or removed with probability $P(e+)$ and $P(e-)$ as well. These two processes enable solutions to move stochastically toward global minima and escape local minima.

4) TERMINATION

The algorithm terminates once a desired fitness score is achieved or the maximum number of generations is reached. At this time, the best-performing DAG-NN as measured by the fitness function is returned. Algorithm 4 illustrates the main algorithm, including the loop termination conditions and the selection of the fittest network.

Algorithm 4 Main()

```
Require: max generations $g$, target fitness $f$
1: $P \leftarrow$initialize()
2: while $f$ not achieved and $g$ not reached do
3:     $P \leftarrow$nextGeneration()
4: end while
5: $n* \leftarrow \underset{n \in P}{\operatorname{argmax}}(\mathcal{F}(n))$
6: return $n*$
```

IV. EXPERIMENTS

We conduct a number of experiments on datasets sourced from the real world in order to measure the performance of our approach. The results strongly demonstrate that compact DAG-NNs trained through our algorithm attain higher accuracies than that of the fully-connected networks.

A. DATASETS

To ensure the effectiveness of our evolutionary algorithm against the considerable diversity in real world applications, we sought datasets that represented a selection of different application spaces and learning tasks – binary or multiclass classification and regression. To this end, we selected six such datasets, the key characteristics of which are enumerated in Table 2. All datasets were feature-normalized to mean 0 and standard deviation 1.

1) PIMA INDIANS DIABETES

Pima Indians Diabetes is a dataset that challenges algorithms to predict the presence or absence of diabetes in 768 women based on eight medical characteristics – a binary classification task. The dataset is a well-known benchmark in the machine learning community and has been the subject of extensive research. Using a general regression neural network (GRNN), Kayaer and Yildirim [46] obtained 80.21% test accuracy on this dataset. For our experiments, the sigmoid activation function was used for the output node in all approaches, and binary cross-entropy was employed as the fitness/loss function.

2) AIR QUALITY

The Air Quality dataset [47] is a regression task for predicting carbon monoxide concentration in an Italian city utilizing readings from a gas sensor and other meteorological metrics. A number of missing values are present in the dataset, labeled with the value -200. We removed data instances with a missing output value as those have little use as training examples. In the experiments, output nodes had no activation function in either approach, and the fitness/loss function used was the mean squared error. When evaluating this dataset, we considered predictions within 0.5 mg/m^3 to be correct.

3) FROG CALLS

This dataset [48] contains processed audio data for frog calls belonging to 60 frogs encompassing multiple taxonomic families, genera, and species, presenting several potential multiclass classification tasks. For the purposes of our experiments, we sought to predict the family, a 4-class classification problem with unbalanced classes. The sigmoid activation function was used for output nodes in all networks and categorical cross-entropy was utilized as the fitness/loss function.

4) ABALONE

This dataset [49], [50] contains physical measurements for over four thousand abalone. Algorithms are challenged to use these measurements in lieu of an exact but time-consuming method for determining abalone age. The ordinal nature of the output variable led us to regard the learning task as a regression task. The dataset contains a categorical variable which we one-hot encoded, yielding in effect 10 attributes. In our experiments, output nodes had no activation function in either approach, and mean squared error was used as a fitness/loss function. Predictions within.5 years were considered to be accurate.

5) WHITE WINE QUALITY

White Wine Quality [51] is a regression task for assessing the quality of white wine samples on a 0-10 scale based on physicochemical tests, using ratings from wine experts as ground truth. We chose to treat the dataset as a regression task. In the experiments, output nodes did not have an activation function, and the fitness/loss function used was the mean squared error. When evaluating for accuracy against the ground truth, predictions within.5 were considered correct (i.e., predictions that would round to the correct score).

6) HEART DISEASE

The Heart Disease dataset [52] contains medical characteristics used to predict the presence of heart disease in individuals. Following the precedent set by previous studies, only the Cleveland subset was used, and the absence (output label 0) or presence (output labels 1-4) of heart disease was predicted, presenting a binary classification task. In addition, six instances containing incomplete data were removed. The sigmoid activation function was used for all output nodes, and binary cross-entropy was used as a fitness/loss function.

TABLE 3. Experimental parameters for our approach.

Parameter	Value
population size	500
initial edge probability	.5
initial weight SD	Modified He *et al.*
initial bias SD	.5
weight mutate probability	.2
weight mutate SD	.05
bias mutate probability	.2
bias mutate SD	.05
edge addition probability	.05
edge removal probability	0

B. SETTINGS

We implemented our approach in Python with CuPy, a NumPy-esque library for n-dimensional array processing with GPU acceleration. For comparison, we implemented traditional neural networks in Python utilizing Keras with a GPU-enabled TensorFlow backend. In our principal experiments, we evaluated three specific configurations for each dataset with k hidden nodes, $k \in \{2, 4, 6, 8, 10\}$:

1) DAG-NN with k hidden nodes
2) Fully-connected ANN with one k-node hidden layer
3) Fully-connected DNN with two $\frac{k}{2}$-node hidden layers

The intention of this experimental design was to determine the performance of our framework (configuration 1) against conventional architectures (configurations 2 and 3) at varying levels of compactness while also examining the effect of node counts on each architecture. In addition, we trained a larger deep neural network with two 10-node layers on each dataset in order to assess the competitiveness of our approach with more massive architectures.

The experiments were run on 16GB NVIDIA Tesla P100 GPUs. For impartial evaluation, it is desirable to set parameters in as equitable of a fashion as possible. The activation function used in hidden nodes for all models was the rectified linear unit (ReLU) [2]. Stochastic gradient descent was used for the fully connected networks with a mini-batch size of 32; no such equivalent parameters exist in our approach. Output activations and fitness/loss depended upon the learning task and were determined on a per-dataset basis as described in Section IV-A. Furthermore, the parameters specific to our algorithm as described in the previous section were set as shown in Table 3. A few values are worthy of note. The initial edge probability was set to.5 for the highest structural diversity in the initial population. Weight initialization was performed using the modified He et al. scheme described in Section III-C.1. The inequality in edge addition and removal probability predisposed the system to favor networks with more edges, which light experimentation indicated yielded improved results.

Test accuracy was the metric used to evaluate the performance of each configuration. The computation of accuracy is dependent upon the learning task. Consider a set of l examples containing m outputs each, evaluated on a network n: (1) For a classification task with binary outputs, each output is rounded to 0 or 1 and represents a single binary prediction, and thus the accuracy is calculated as

$$acc(n) = \sum_{i=0}^{l}\sum_{j=0}^{m}\frac{\delta\big(\lfloor \text{predicted}[i][j] + 0.5\rfloor\,,\,\text{actual}[i][j]\big)}{l \times m},$$

where δ is the Kronecker delta. (2) For a multiclass classification task with one-hot encoded outputs, accuracy is defined as

$$acc(n) = \sum_{i=0}^{l}\frac{\delta\big(\underset{j}{\text{argmax}}(\text{predicted}[i][j]),\, \underset{j}{\text{argmax}}(\text{actual}[i][j])\big)}{l}.$$

(3) For a regression task, we consider a prediction to be correct if it is within some value a from the actual value, and therefore accuracy is calculated as

$$acc(n) = \sum_{i=0}^{l}\sum_{j=0}^{m}\frac{\begin{cases}1 & \textbf{if} |\text{predicted}[i][j] - \text{actual}[i][j]| < a\\ 0 & \textbf{else}\end{cases}}{l \times m}.$$

For each model, we performed 5-fold cross-validation; that is, the dataset was partitioned into five contiguous subsets, with each held out in turn as a validation set while the remaining data was used to train the model. Each fold was trained for 1000 generations/epochs and the mean peak test accuracy for the folds was reported along with the standard error.

C. RESULTS

Table 4 displays the experimental results in tabular form. The best-performing model per row is denoted in bold. The numbers clearly indicate that the performance of our approach exceeds that of counterpart fully-connected networks, attaining the highest test accuracy in 26 of 30 contests. For five out of six datasets, the highest accuracy across all node counts belonged to a DAG-NN configuration, with the Air Quality dataset being the only exception. Figure 3 illustrates the effect of the number of hidden nodes on the resultant model accuracy. Remarkably, it can be observed that the DAG-NNs achieve extraordinary performance with comparatively fewer nodes than the fully-connected nets. For three datasets – Pima Indians Diabetes, White Wine Quality, and Heart Disease – our approach achieves greater performance with two nodes than the conventional networks do with ten. It is clear that DAG-NNs hold far greater representational capacity with compact node counts.

1) COMPARISON WITH DEEPER, WIDER NETWORKS

Seeing that DAG-NNs vastly outperformed their counterpart networks, we found it worthwhile to compare our approach with a larger fully-connected network. The results of a deep neural network with two hidden layers of 10 nodes each are depicted in Table 5, with which we stumble upon a curious result. The 20-node DNN achieved roughly equivalent performance to its more compact counterparts. This suggests that for these datasets the limit in representational capability for the basic fully connected architecture has been approached, and no amount of additional width and depth would result

TABLE 4. Experimental results: accuracy (%).

k = 2	Our Approach	ANN (2)	DNN (1/1)
Pima Indians Diabetes	**81.39 ± 1.78**	78.79 ± 1.80	65.12 ± 2.62
Air Quality	85.93 ± 0.46	**86.02 ± 0.51**	48.99 ± 13.81
Frog Calls	**97.33 ± 0.17**	92.69 ± 0.90	71.41 ± 6.70
Abalone	**29.47 ± 2.05**	28.12 ± 1.68	18.88 ± 1.33
White Wine Quality	**57.25 ± 1.85**	53.97 ± 1.49	47.36 ± 3.17
Heart Disease	**89.21 ± 1.29**	86.18 ± 2.26	71.84 ± 7.46

k = 4	Our Approach	ANN (4)	DNN (2/2)
Pima Indians Diabetes	**81.26 ± 1.44**	79.31 ± 1.81	79.70 ± 1.94
Air Quality	**86.72 ± 0.33**	86.34 ± 0.32	85.67 ± 0.42
Frog Calls	**98.33 ± 0.15**	96.73 ± 0.17	83.84 ± 5.83
Abalone	**30.05 ± 2.86**	29.17 ± 1.77	27.51 ± 2.99
White Wine Quality	**57.07 ± 1.94**	56.00 ± 1.36	49.71 ± 3.68
Heart Disease	**89.89 ± 1.79**	84.47 ± 2.34	84.82 ± 2.37

k = 6	Our Approach	ANN (6)	DNN (3/3)
Pima Indians Diabetes	**81.39 ± 1.58**	78.39 ± 1.60	77.35 ± 2.04
Air Quality	86.49 ± 0.25	**86.74 ± 0.40**	86.61 ± 0.33
Frog Calls	**98.74 ± 0.16**	97.93 ± 0.17	95.08 ± 0.39
Abalone	**30.86 ± 2.88**	29.14 ± 1.86	29.48 ± 1.68
White Wine Quality	**58.13 ± 1.74**	54.89 ± 1.28	55.77 ± 2.03
Heart Disease	**89.54 ± 1.83**	82.81 ± 1.76	81.49 ± 1.90

k = 8	Our Approach	ANN (8)	DNN (4/4)
Pima Indians Diabetes	**82.69 ± 1.55**	78.92 ± 2.49	79.04 ± 1.69
Air Quality	86.62 ± 0.31	**87.26 ± 0.49**	86.44 ± 0.46
Frog Calls	**98.65 ± 0.07**	98.33 ± 0.05	97.18 ± 0.19
Abalone	**30.48 ± 2.81**	29.51 ± 1.98	30.14 ± 2.02
White Wine Quality	**57.64 ± 1.77**	55.53 ± 1.22	55.86 ± 2.02
Heart Disease	**89.88 ± 1.54**	83.82 ± 2.33	84.48 ± 2.72

k = 10	Our Approach	ANN (10)	DNN (5/5)
Pima Indians Diabetes	**81.91 ± 1.25**	79.18 ± 2.00	78.53 ± 1.91
Air Quality	86.54 ± 0.35	**87.30 ± 0.45**	87.06 ± 0.25
Frog Calls	**98.75 ± 0.15**	98.68 ± 0.11	97.65 ± 0.23
Abalone	**30.45 ± 2.83**	29.63 ± 1.58	29.88 ± 2.10
White Wine Quality	**57.90 ± 1.77**	56.53 ± 1.85	55.79 ± 1.66
Heart Disease	**90.21 ± 1.75**	84.84 ± 1.88	83.82 ± 2.19

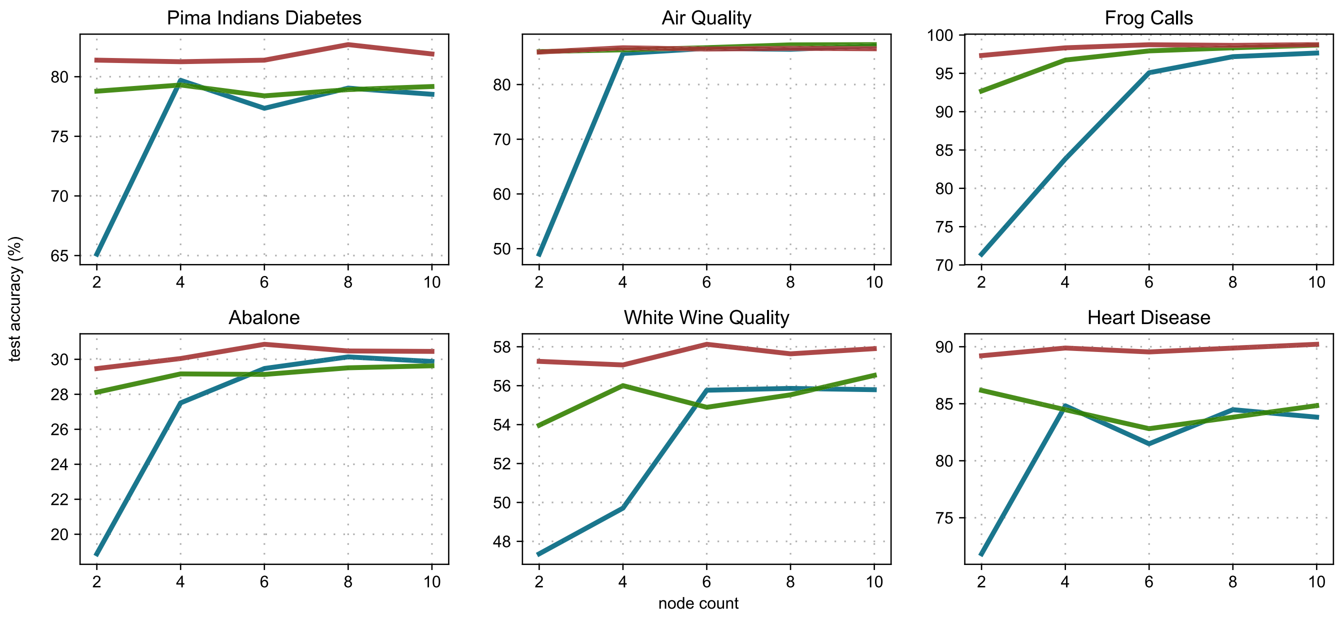


FIGURE 3. Comparison of test accuracies over increasing hidden node counts between our approach (red), a conventional ANN (green), and a conventional DNN (blue).

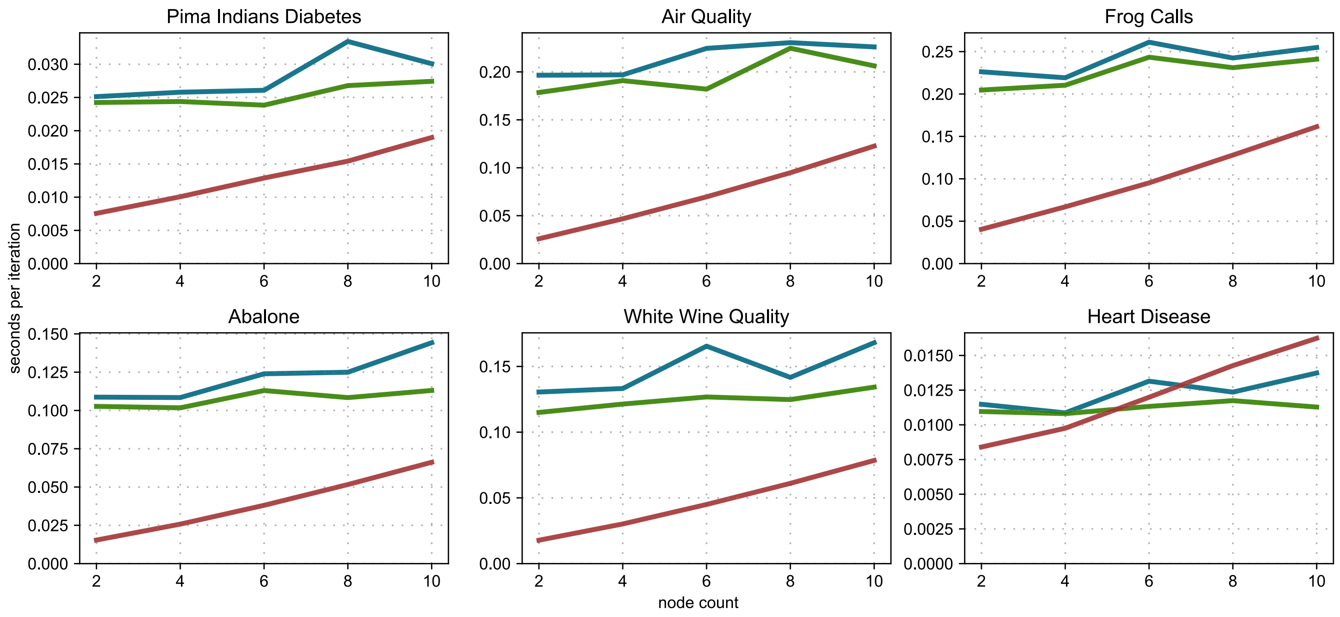


FIGURE 4. Comparison of seconds per iteration over increasing hidden node counts between our approach (red), a conventional ANN (green), and a conventional DNN (blue).

in further improvements. Yet, the DAG-NN attains favorable results to this larger network in 3 of 6 datasets while only using two hidden nodes. From this result, we can posit that our framework is capable of learning representations of the data that fully-connected architectures cannot, and in a way that computational power cannot overcome.

2) RUNTIME ANALYSIS

With computational efficiency being a core consideration for compact neural networks, a comparative analysis of the runtimes of the competing approaches would be a fruitful endeavor. In Figure 4, we present the average per-iteration runtime for each experimental configuration. It can be seen that our framework exhibits the highest efficiency in the vast majority of configurations, and often by a wide margin. The fully-connected networks appear to scale more efficiently with increasing node counts, but the results remind us that DAG-NNs are able to do more with less hidden nodes. It is important to note that the runtimes are highly dependent upon multiple factors; implementation, batch size

TABLE 5. Results on larger DNN: accuracy (%).

Pima Indians Diabetes	79.05 ± 1.95
Air Quality	87.90 ± 0.31
Frog Calls	98.49 ± 0.13
Abalone	29.68 ± 1.92
White Wine Quality	56.46 ± 1.79
Heart Disease	83.14 ± 2.54

for the gradient-based networks, and population size for our approach are particularly significant contributors. Nevertheless, our experiments highlight that increased accuracy can be attained with faster iteration times using our framework.

D. DISCUSSION

The use of evolved compact DAG-NNs for concept learning is encouraging, with marked improvement over conventional neural networks. The clear next step is to apply the algorithms described in this paper to larger networks and assess their effectiveness and competitiveness. We too are curious about what other parameters in DAG-NNs – ones which were set constant in our approach – could be evolved to obtain higher accuracies, such as node activation functions. Also, while DAG-NNs represent a generalized form of feedforward neural networks, it is interesting to consider non-feedforward architectures such as recurrent neural networks and the feasibility of applying our approach to generalized forms thereof. Furthermore, future work may entail other forms of evolutionary algorithms such as covariance matrix adaptation evolution strategies which also hold promise to effectively evolve DAG-NNs.

V. CONCLUSION

In this paper, we considered the challenges of training and deploying large neural networks with numerous hyperparameters. In response to these challenges, we proposed an evolutionary algorithm to train compact directed acyclic graph neural networks, enabling the automated selection of parameters and a concise network structure retaining high predictive accuracy. Our experiments indicate that DAG-NNs generated with our method are superior to traditional fully-connected architectures, even when evaluated against comparatively larger networks.

REFERENCES

[1] R. H. R. Hahnloser, R. Sarpeshkar, M. A. Mahowald, R. J. Douglas, and H. S. Seung, "Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit," *Nature*, vol. 405, no. 6789, pp. 947–951, Jun. 2000.

[2] V. Nair and G. E. Hinton, "Rectified linear units improve restricted Boltzmann machines," in *Proc. 27th Int. Conf. Int. Conf. Mach. Learn. (ICML)*, Madison, WI, USA: Omnipress, 2010, pp. 807–814. [Online]. Available: http://dl.acm.org/citation.cfm?id=3104322.3104425

[3] P. Ramachandran, B. Zoph, and Q. V. Le, "Searching for activation functions," *CoRR*, vol. abs/1710.05941, pp. 1–13, Oct. 2017. [Online]. Available: http://arxiv.org/abs/1710.05941

[4] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015, doi: 10.1038/nature14539.

[5] H. W. Lin, M. Tegmark, and D. Rolnick, "Why does deep and cheap learning work so well?" *J. Stat. Phys.*, vol. 168, no. 6, pp. 1223–1247, Sep. 2017, doi: 10.1007/s10955-017-1836-5.

[6] H. Mhaskar, Q. Liao, and T. Poggio, "When and why are deep networks better than shallow ones?" in *Proc. AAAI Conf. Artif. Intell.*, 2017, pp. 2343–2349. [Online]. Available: https://aaai.org/ocs/index.php/AAAI/AAAI17/paper/view/14849

[7] T. Elsken, J. H. Metzen, and F. Hutter, "Neural architecture search: A survey," *J. Mach. Learn. Res.*, vol. 20, no. 55, pp. 55-1–55-21, 2019. [Online]. Available: http://jmlr.org/papers/v20/18-598.html

[8] A. Doering, M. Galicki, and H. Witte, "Structure optimization of neural networks with the A*-algorithm," *IEEE Trans. Neural Netw.*, vol. 8, no. 6, pp. 1434–1445, Nov. 1997.

[9] J. Yang, J. Ma, M. Berryman, and P. Perez, "A structure optimization algorithm of neural networks for large-scale data sets," in *Proc. IEEE Int. Conf. Fuzzy Syst. (FUZZ-IEEE)*, Jul. 2014, pp. 956–961.

[10] M. Ihme, A. L. Marsden, and H. Pitsch, "Generation of optimal artificial neural networks using a pattern search algorithm: Application to approximation of chemical systems," *Neural Comput.*, vol. 20, no. 2, pp. 573–601, Feb. 2008, doi: 10.1162/neco.2007.08-06-316.

[11] S. Shirakawa, Y. Iwata, and Y. Akimoto, "Dynamic optimization of neural network structures using probabilistic modeling," *CoRR*, vol. abs/1801.07650, pp. 4074–4082, Jan. 2018. [Online]. Available: http://arxiv.org/abs/1801.07650

[12] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Salt Lake City, UT, USA, Jun. 2018, pp. 8697–8710. [Online]. Available: http://openaccess.thecvf.com/content_cvpr_2018/html/Zoph_Learning_Transferable_Architectures_CVPR_2018_paper.html

[13] S. Yang and D. Ramanan, "Multi-scale recognition with DAG-CNNs," *CoRR*, vol. abs/1505.05232, pp. 1215–1223, May 2015. [Online]. Available: http://arxiv.org/abs/1505.05232

[14] B. Shuai, Z. Zuo, G. Wang, and B. Wang, "DAG-recurrent neural networks for scene labeling," *CoRR*, vol. abs/1509.00552, pp. 1–10, May 2015. [Online]. Available: http://arxiv.org/abs/1509.00552

[15] K. O. Stanley and R. Miikkulainen, "Evolving neural networks through augmenting topologies," *Evol. Comput.*, vol. 10, no. 2, pp. 99–127, 2002.

[16] T. Shinozaki and S. Watanabe, "Structure discovery of deep neural network based on evolutionary algorithms," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, Apr. 2015, pp. 4979–4983.

[17] L. Davis, Ed., *Handbook of Genetic Algorithms*. New York, NY, USA: Van Nostrand, 1991.

[18] N. Hansen, S. D. Müller, and P. Koumoutsakos, "Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (CMA-ES)," *Evol. Comput.*, vol. 11, no. 1, pp. 1–18, Mar. 2003, doi: 10.1162/106365603321828970.

[19] M. Hüsken, Y. Jin, and B. Sendhoff, "Structure optimization of neural networks for evolutionary design optimization," *Soft Comput.*, vol. 9, no. 1, pp. 21–28, Jan. 2005, doi: 10.1007/s00500-003-0330-y.

[20] N. Hansen and A. Ostermeier, "Completely derandomized self-adaptation in evolution strategies," *Evol. Comput.*, vol. 9, no. 2, pp. 159–195, Jun. 2001, doi: 10.1162/106365601750190398.

[21] R. Miikkulainen, J. Z. Liang, E. Meyerson, A. Rawal, D. Fink, O. Francon, B. Raju, H. Shahrzad, A. Navruzyan, N. Duffy, and B. Hodjat, "Evolving deep neural networks," *CoRR*, vol. abs/1703.00548, pp. 1–8, Mar. 2017. [Online]. Available: http://arxiv.org/abs/1703.00548

[22] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le, "Regularized evolution for image classifier architecture search," in *Proc. 33rd AAAI Conf. Artif. Intell. (AAAI), 31st Innov. Appl. Artif. Intell. Conf. (IAAI), 9th AAAI Symp. Educ. Adv. Artif. Intell. (EAAI)*, Honolulu, HI, USA, Jan./Feb. 2019, pp. 4780–4789, doi: 10.1609/aaai.v33i01.33014780.

[23] Y. Ma, R. Chen, W. Li, F. Shang, W. Yu, M. Cho, and B. Yu, "A unified approximation framework for deep neural networks," Jul. 2018, *arXiv:1807.10119*. [Online]. Available: https://arxiv.org/abs/1807.10119

[24] J. Cheng, J. Wu, C. Leng, Y. Wang, and Q. Hu, "Quantized CNN: A unified approach to accelerate and compress convolutional networks," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 29, no. 10, pp. 4730–4743, Oct. 2018.

[25] Y. Choi, M. El-Khamy, and J. Lee, "Universal deep neural network compression," *CoRR*, vol. abs/1802.02271, pp. 1–5, Feb. 2018. [Online]. Available: http://arxiv.org/abs/1802.02271

[26] C. Chiu and J. Zhan, ''Deep learning for link prediction in dynamic networks using weak estimators,'' *IEEE Access*, vol. 6, pp. 35937–35945, 2018.

[27] M. Bhaduri and J. Zhan, ''Using empirical recurrences rates ratio for time series data similarity,'' *IEEE Access*, vol. 6, pp. 30855–30864, 2018.

[28] J. M.-T. Wu, J. Zhan, and S. Chobe, ''Mining association rules for low frequency itemsets,'' *PLoS ONE*, vol. 13, no. 7, 2018, Art. no. e0198066.

[29] P. Ezatpoor, J. Zhan, J. M.-T. Wu, and C. Chiu, ''Finding top-*k* dominance on incomplete big data using MapReduce framework,'' *IEEE Access*, vol. 6, pp. 7872–7887, 2018.

[30] P. Chopade and J. Zhan, ''A framework for community detection in large networks using game-theoretic modeling,'' *IEEE Trans. Big Data*, vol. 3, no. 3, pp. 276–288, Sep. 2017.

[31] M. Bhaduri, J. Zhan, and C. Chiu, ''A weak estimator for dynamic systems,'' *IEEE Access*, vol. 5, pp. 27354–27365, 2017.

[32] M. Bhaduri, J. Zhan, C. Chiu, and F. Zhan, ''A novel online and non-parametric approach for drift detection in big data,'' *IEEE Access*, vol. 5, pp. 15883–15892, 2017.

[33] C. Chiu, J. Zhan, and F. Zhan, ''Uncovering suspicious activity from partially paired and incomplete multimodal data,'' *IEEE Access*, vol. 5, pp. 13689–13698, 2017.

[34] J. M.-T. Wu, J. Zhan, and J. C.-W. Lin, ''Ant colony system sanitization approach to hiding sensitive itemsets,'' *IEEE Access*, vol. 5, pp. 10024–10039, 2017.

[35] J. Zhan and B. Dahal, ''Using deep learning for short text understanding,'' *J. Big Data*, vol. 4, no. 1, p. 34, 2017.

[36] J. Zhan, S. Gurung, and S. P. K. Parsa, ''Identification of top-*k* nodes in large networks using Katz centrality,'' *J. Big Data*, vol. 4, no. 1, p. 16, 2017.

[37] J. Zhan, T. Rafalski, G. Stashkevich, and E. Verenich, ''Vaccination allocation in large dynamic networks,'' *J. Big Data*, vol. 4, no. 1, p. 2, 2016.

[38] H. Selim and J. Zhan, ''Towards shortest path identification on large networks,'' *J. Big Data*, vol. 3, no. 1, 2016, p. 10.

[39] F. Zhan, A. Martinez, N. Rai, R. McConnell, M. Swan, M. Bhaduri, J. Zhan, L. Gewali, and P. Oh, ''Beyond cumulative sum charting in non-stationarity detection and estimation,'' *IEEE Access*, vol. 7, pp. 140860–140874, 2019.

[40] A. Hart, B. Smith, S. Smith, E. Sales, J. Hernandez-Camargo, Y. M. Garcia, F. Zhan, L. Griswold, B. Dunkelberger, M. R. Schwob, S. Chaudhry, J. Zhan, L. Gewali, and P. Oh, ''Resolving intravoxel white matter structures in the human brain using regularized regression and clustering,'' *J. Big Data*, vol. 6, no. 1, p. 61, 2019.

[41] F. Zhan, ''Hand gesture recognition with convolution neural networks,'' in *Proc. IEEE 20th Int. Conf. Inf. Reuse Integr. Data Sci.*, Los Angeles, CA, USA, Jul./Aug. 2019, pp. 295–298.

[42] F. Zhan, ''How to optimize social network influence,'' in *Proc. IEEE 2nd Int. Conf. Artif. Intell. Knowl. Eng. (AIKE)*, Sardinia, Italy, Jun. 2019, pp. 113–116.

[43] E. Aguilar, J. Dancel, D. Mamaud, D. Pirosch, F. Tavacoli, F. Zhan, R. Pearce, M. Novack, H. Keehu, B. Lowe, J. Zhan, L. Gewali, and P. Oh, ''Highly parallel seedless random number generation from arbitrary thread schedule reconstruction,'' in *Proc. IEEE Int. Conf. Big Knowl.*, Beijing, China, Nov. 2019, pp. 1–7.

[44] E. Hunt, R. Janamsetty, C. Kinares, C. Koh, A. Sanchez, F. Zhan, M. Ozdemir, S. Waseem, O. Yolcu, B. Dahal, J. Zhan, L. Gewali, and P. Oh, ''Machine learning models for paraphrase identification and its applications on plagiarism detection,'' in *Proc. IEEE Int. Conf. Big Knowl.*, Beijing, China, Nov. 2019, pp. 1–7.

[45] K. He, X. Zhang, S. Ren, and J. Sun, ''Delving deep into rectifiers: Surpassing human-level performance on imagenet classification,'' in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Santiago, Chile, Dec. 2015, pp. 1026–1034, doi: 10.1109/ICCV.2015.123.

[46] K. Kayaer and T. Yildirim, ''Medical diagnosis on Pima Indian diabetes using general regression neural networks,'' in *Proc. Int. Conf. Artif. Neural Netw. Neural Inf. Process.*, Jan. 2003, pp. 1–4.

[47] S. De Vito, E. Massera, M. Piga, L. Martinotto, and G. D. Francia, ''On field calibration of an electronic nose for benzene estimation in an urban pollution monitoring scenario,'' *Sens. Actuators B, Chem.*, vol. 129, no. 2, pp. 750–757, 2008. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0925400507007691

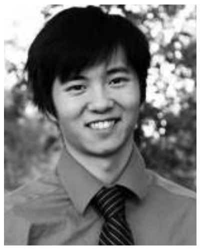
[48] J. G. Colonna, M. Cristo, M. Salvatierra, and E. F. Nakamura, ''An incremental technique for real-time bioacoustic signal segmentation,'' *Expert Syst. Appl.*, vol. 42, no. 21, pp. 7367–7374, Nov. 2015, doi: 10.1016/j.eswa.2015.05.030.

[49] W. J. Nash and T. M. R. Laboratories, ''The population biology of abalone (Haliotis species) in tasmania. 1, blacklip abalone (H. RUBRA) from the north coast and the islands of bass strait,'' in *Proc. CIP Conf.*, 1994, pp. 62–63.

[50] S. G. Waugh, ''Extending and benchmarking cascade-correlation: Extensions to the cascade-correlation architecture and benchmarking of feed-forward supervised artificial neural networks,'' Ph.D. dissertation, Univ. Tasmania, Hobart, TAS, Australia, 1997.

[51] P. Cortez, A. Cerdeira, F. Almeida, T. Matos, and J. Reis, ''Modeling wine preferences by data mining from physicochemical properties,'' *Decis. Support Syst.*, vol. 47, no. 4, pp. 547–553, 2009.

[52] R. Detrano, A. Janosi, W. Steinbrunn, M. Pfisterer, J.-J. Schmid, S. Sandhu, K. H. Guppy, S. Lee, and V. Froelicher, ''International application of a new probability algorithm for the diagnosis of coronary artery disease,'' *Amer. J. Cardiol.*, vol. 64, no. 5, pp. 304–310, 1989. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0002914989905249



CARTER CHIU is currently pursuing the Ph.D. degree with the Department of Computer Science, University of Nevada, Las Vegas. He is also the Lab Manager of the Big Data Hub with the University of Nevada, Las Vegas. His research interests include deep learning and big data analytics.

JUSTIN ZHAN is currently an ARA Scholar and a Professor of data science with the Department of Computer Science and Computer Engineering, University of Arkansas. He is also an Adjunct Professor with the Department of Biomedical Informatics, University of Arkansas for Medical Sciences. He has been the Director of the Big Data Hub and a Professor with the Department of Computer Science, College of Engineering, University of Nevada, Las Vegas. His research interests include data science, biomedical informatics, artificial intelligence, information assurance, and social computing. He was the Steering Chair of the IEEE International Conference on Social Computing (SocialCom), the IEEE International Conference on Privacy, Security, Risk and Trust (PASSAT), and the IEEE International Conference on Biomedical Computing (BioMedCom). He has been the Editor-in-Chief of the *International Journal of Privacy*, *Security and Integrity*, and the *International Journal of Social Computing and Cyber-Physical Systems*. He has served as a Conference General Chair, a Program Chair, a Publicity Chair, a Workshop Chair, and a Program Committee Member for 150 international conferences and the Editor-in-Chief, an Editor, an Associate Editor, a Guest Editor, an Editorial Advisory Board Member, and an Editorial Board Member for 30 journals. He has published 230 articles in peer-reviewed journals and conferences and delivered more than 30 keynote speeches and invited talks. He has been involved in more than 50 projects as a Principal Investigator (PI) or a Co-PI, which were funded by the National Science Foundation, Department of Defense, National Institute of Health and so on.