Switched Neural Networks for Simultaneous Learning of Multiple Functions

Mehmet Önder Efe, Burak Kürkçü, Coşku Kasnakoğlu, *Member, IEEE*, Zaharuddin Mohamed, and Zhijie Liu, *Member, IEEE*

***Abstract*—This paper introduces the notion of switched neural networks for learning multiple functions under different switching configurations. The neural network structure has adjustable parameters and for each function the state of the parameter vector is determined by a mask vector, 1/0 for active/inactive or +1/-1 for plain/inverted. The optimization problem is to schedule the switching strategy (mask vector) required for each function together with the best parameter vector (weights/biases) minimizing the loss function. This requires a procedure that optimizes a vector containing real and binary values simultaneously to discover commonalities among various functions. Our studies show that a small sized neural network structure with an appropriate switching regime is able to learn multiple functions successfully. During the tests focusing on classification, we considered 2-variable binary functions and all 16 combinations have been chosen as the functions. The regression tests consider four functions of two variables. Our studies showed that simple NN structures are capable of storing multiple information via appropriate switching.**

***Index Terms*—Neural networks, parameter switching, learning multiple functions, genetic algorithms.**

NOMENCLATURE

α_b	$N \times 1$ random vector containing zeros and ones.
γ_c	A small positive value for mutation operator in continuous subroutine.
$\mathbb{B}$	A real set containing only 0 and 1
$\mathcal{D}_l$	l^{th} dataset.
$\mathcal{R}$	The set of real numbers.
$\mathcal{S}^b\{\cdot\}$	Random initializer of binary vector sets.
$\mathcal{S}^c\{\cdot\}$	Random initializer of continuous vector sets.
μ_b	A threshold value for mutation operator in binary subroutine.
μ_c	A threshold value for mutation operator in continuous subroutine.
Φ_L, Φ_R	Nonlinear activation functions.
σ_c	A constant for mutation operator in continuous subroutine.
θ_m	Concatenated (row) mask vector.
θ_m^*	Best value of the concatenated mask vector observed so far.
$\{\theta_m\}_1^{n_b}$	Mask vector population.
$\{w\}_1^{n_c}$	Parameter vector population.
B_L	Bias vector for the hidden neurons.
B_R	Bias value for the output neuron.
C	Number of correctly learned functions in 100 experiments.
d_p^l	Target value in l^{th} function's p^{th} row.
f	Neural network response.
f_{m_l}	Neural network response with the l^{th} mask.
g_{max}	Maximum iteration (generation) number.
J	Total loss function.
L	Number of different functions.
m	Mask vector from $\mathbb{B}^{1\times LN}$
m_l	l^{th} mask vector from $\mathbb{B}^{1\times N}$
M_{B_L}	Mask for the hidden neurons' bias vector.
M_{B_R}	Mask for the output neuron's bias value.
M_{W_L}	Mask for the left weight matrix.
M_{W_R}	Mask for the right weight matrix.
N	Total number of adjustable continuous parameters (weights/biases) in a NN.
n_b	Population size for binary search.
n_c	Population size for continuous search.
n_{Cb}	Number of binary offsprings generated at each generation.
n_{Cc}	Number of continuous offsprings generated at each generation.
P_l	Number of rows in l^{th} function.
R	The number of hidden neurons.
r_c	A normal distributed random variable ($\mathcal{N}(0,1)$)
r_m	A uniformly distributed random variable from (0,1)
u	Input vector.
w	Parameter vector.
w^*	Best value of the parameter vector observed so far.
W_L	Left weight matrix.
W_R	Right weight matrix.

Manuscript received 28 June 2023; revised 30 October 2023; accepted 11 February 2024. Date of publication 11 March 2024; date of current version 24 July 2024. *(Corresponding author: Mehmet Önder Efe.)*

Mehmet Önder Efe is with the Department of Computer Engineering, Hacettepe University, 06800 Ankara, Türkiye (e-mail: onderefe@hacettepe.edu.tr).

Burak Kürkçü is with the Department of Mechanical Engineering, University of California Berkeley, Berkeley, CA 94720 USA, on leave from the Department of Computer Engineering, Hacettepe University, 06800 Ankara, Türkiye (e-mail: bkurkcu@cs.hacettepe.edu.tr).

Coşku Kasnakoğlu is with the Electrical and Electronics Engineering Department, TOBB University of Economics and Technology, 06510 Ankara, Türkiye (e-mail: kasnakoglu@etu.edu.tr).

Zaharuddin Mohamed is with the School of Electrical Engineering, Universiti Teknologi Malaysia, Johor Bahru 81310, Malaysia (e-mail: zahar@fke.utm.my).

Zhijie Liu is with the Institute of Artificial Intelligence, University of Science and Technology Beijing, Beijing 100083, China (e-mail: liuzhijie2012@gmail.com).

Recommended for acceptance by A. Gupta.

Digital Object Identifier 10.1109/TETCI.2024.3369981

I. Introduction

BRAIN research often considers finding sub volumes in the brain that are peculiar to particular cognitive or motor functions. This viewpoint assumes that the neural structures develop specific control functions and neuron groups are responsible for accomplishing single tasks. The question asking whether a neuron could be multifunctional is an interesting question and in the current paper, we address this problem for artificial neural network (NN) structures.

In large sized NNs, as in deep NNs, the number of adjustable parameters is very large and there is an excess capacity that could to either be pruned or exploited appropriately to get a small sized structure. NN pruning has been considered many times in the literature and the goal was to maintain an acceptable performance after some parts of the NN were deleted, [1], [2], [3]. The use of the excess capacity, on the other hand, requires adjusting the internal structure together with the network parameters in such a way that superposition of many models into one is made possible, [6], [7]. The structures having many layers and neurons may display such an excess capacity, however, as the NN size gets smaller, the problem becomes optimizing the structure and the adjustable parameter vector simultaneously. Then, the solution subspace is much narrower than that in large sized neural structures, where there is a remarkably large parametric redundancy to store information within mutually exclusive functional subnetworks. This paper addresses the problem of optimizing the structure and parameter set for simultaneous learning of multiple functions. The approach adopted here employs a set of mask vectors as in [6], [7] and the entire process is governed by a genetic optimization approach.

In the current paper, we deliberately work on small sized NN structures and seek for smallest possible architectures, where there is no redundancy and the optimum use of the structure can be claimed. The NN structure is forced to discover the maximal similarities, i.e. commonalities, among the given functions and the output layer aims to aggregate the extracted information. Though implementing multiple hidden layers is possible, we will confine ourselves to simple structures to maintain our claim of best use of limited resources.

The problem of exploiting a NN structure for multiple purposes have been studied few times in the literature. In [4], a neural structure is trained for a given dataset, the network is then pruned and the process for next dataset considers the previously fixed weights together with the adjustable weight subset, which was the pruned subset of the previous stage. Here, the process considers the datasets one by one and tries to teach the new dataset to a gradually contracting neural space (free neurons or weights) aided also by the previously fixed weight values. The results focus on deep neural structures with several well known datasets. This is a good example of aforementioned case storing different datasets within different subnetworks of a large NN structure, which was studied also in [5].

The work in [6] considers two separate online classification settings. The proposed approach is based on the availability of a number of orthogonal vectors that makes it possible to store the parameter vector as a sum. When the appropriate context key (mask) multiplies the weight vector, the weight set associated to the context key is obtained. The method requires finding orthogonal terms to store the weight set. The main difficulty here is to find the orthogonal terms when there are many different datasets/functions. Further, since the considered neural structure is a deep NN containing thousands of neurons and many layers, one cannot make sure that the structure is the smallest possible one that meets the desired learning goals.

In [7], [8], a large NN structure is initialized and without any modification, for a given task, the best weight subset is sought. As long as the NN is large in structure, the diversity that spans the functional space of the NN can be ensured and a best matching mask vector could be inferred for a given arbitrary dataset. The work in [7] defines four continual learning scenarios and describes how a best matching mask could be extracted if a dataset is given. The issue of randomly initialized and untrained NNs that do not undergo an active weight update phase aside from supermask optimization has been studied in [9] with empirical evidence.

In [10], a deep NN structure is initialized and every weight is multiplied by 1 to keep, 0 to hide and -1 to invert. The optimization problem is to find the best multiplier vector to obtain a good performance. The authors reported that the approach resulted in up to 99% of pruning rate. In the sequel, we will adopt such an approach yet we will confine ourselves to ± 1 as multipliers and optimize the continuous weights simultaneously.

In [11], the notion of masked NN has been studied within the context of autoencoders. The work shows how to mask the weighted connections of a standard autoencoder to convert it into a distribution estimator. A cross-entropy loss is chosen and the model is updated using the stochastic gradient descent approach.

The cited body of literature focuses largely on the deep NN structures, where the redundancy is excessive and necessity to each one of the adjustable parameters is questionable. Further, the works cited assume the lottery ticket hypothesis proposed by Frankle and Carbin, [12], which states that there exists a subnetwork that performs the desired mapping with an acceptable performance and the problem is to locate its elements within the large NN structure. Obviously, the approach is based on determining and deleting unnecessary elements, which corresponds to pruning or a 1/0 masking.

Current paper dwells on small sized neural structures, where the structure is exploited maximally for given a number of different functions. The distribution of the functional capabilities and collection of them over a given function is discussed under the structural simplicity requirement. We believe that the current paper lets us understand the neuronal contribution and its interaction with the mask vector towards the goal of building much simpler neural structures that display versatility for complicated problems. Further, no pruning is done during optimization process. The approach presented here can also be considered as a remedy to the problems arising due to catastrophic forgetting, a method to alleviate which is studied in [13] within the context of elastic weight consolidation. In our approach, as the learning progresses for all functions simultaneously, the critical information stored within the neural network is refined gradually without losing any previously gained capability. This is the result of the

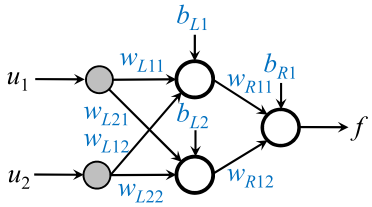

Fig. 1. A 2-2-1 Neural Network (NN) structure containing $N = 9$ adjustable parameters.

overall cost function defined over all functions and minimization of it for all tasks at the same time. Further, the approach in the current paper does not utilize gradient information.

This paper is organized as follows: In the second section, we give the problem definition. The adopted genetic algorithm is summarized in the third section. Next, we explain how the vector composed of binary (mask) and continuous (parameter) values is optimized simultaneously. In the fifth section, we introduce a modified mask, which considers -1 instead of 0 in the mask vector. Following sections describe the links that can be established in between federated learning, dynamical systems and control, explainable artificial intelligence and so on. Finally, we give the concluding remarks at the end of the paper.

II. Problem Definition

Consider the neural network structure shown in Fig. 1. The NN has two inputs, one hidden layer containing $R = 2$ neurons and one output neuron. The continuous parameters, i.e. weights and biases (parameters), of the NN have the following matrix representation.

$$W_L := \begin{bmatrix} w_{L11} & w_{L12} \\ w_{L21} & w_{L22} \end{bmatrix}, \; B_L := \begin{bmatrix} b_{L1} \\ b_{L2} \end{bmatrix} \tag{1}$$

$$W_R := \begin{bmatrix} w_{R11} & w_{R12} \end{bmatrix}, \; B_R := b_{R1} \tag{2}$$

With these parameters, defining the input vector as $u := [u_1 \, u_2]^T$, the output function f can be written as

$$f = \Phi_R(W_R \Phi_L(W_L u + B_L) + B_R) \tag{3}$$

where, $\Phi_L(.)$ is the nonlinear activation function of the hidden neurons and $\Phi_R(.)$ is the activation function of the output neuron. The NN input/output relation above has the following parameter vector, which is a 1×9 vector for the NN shown in Fig. 1.

$$w := [w_{L11} \; w_{L21} \; w_{L12} \; w_{L22} \; b_{L1} \; b_{L2} \; w_{R11} \; w_{R12} \; b_{R1}] \tag{4}$$

Now define the following mask matrices, the entries of which can either be zero or one, i.e. mask matrices contain binary variables.

$$M_{W_L} := \begin{bmatrix} m_{wL11} & m_{wL12} \\ m_{wL21} & m_{wL22} \end{bmatrix}, \; M_{B_L} := \begin{bmatrix} m_{bL1} \\ m_{bL2} \end{bmatrix} \tag{5}$$

$$M_{W_R} := \begin{bmatrix} M_{wR11} & M_{wR12} \end{bmatrix}, \; M_{B_R} := m_{bR1} \tag{6}$$

The binary mask variables seen above are vectorized as a row given as below.

$$m := [m_{wL11} \, m_{wL21} \, m_{wL12} \, m_{wL22} \, m_{bL1} \, m_{bL2} \, m_{wR11} \, m_{wR12} \, m_{bR1}] \tag{7}$$

where $m \in \mathbb{B}^{1\times N}$ and $N = 9$ for 2–2–1 NN structure.

Definition 1. Elementwise Multiplication: The operator $\odot$ performs elementwise multiplication of the operands. Let W_L and M_{W_L} be two matrices having the same dimensions, elementwise multiplication of these two matrices produce $W_L \odot M_{W_L} = M_{W_L} \odot W_L := H$ where $H_{ij} := W_{L_{ij}} M_{W_{L_{ij}}}$.

Definition 2. Masked NN: Let the continuous parameters of a NN be as given in (1)–(2), let the corresponding binary mask parameters be as given in (5)–(6). A two inputs single output masked NN structure is then defined as

$$f(u_1, u_2, m, w) = \Phi_R \{(W_R \odot M_{W_R}) \Phi_L((W_L \odot M_{W_L}) u + (B_L \odot M_{B_L})\} + (B_R \odot M_{B_R}) \tag{8}$$

For the NN in (3), all of the variables given by (4) are active, whereas, for the masked NN in (8), some of the parameters are deactivated via $m \odot w$ and only a subnetwork determined by m is active.

Consider there is a set of N dimensional row mask vectors, say $m_1, m_2, \ldots, m_L$ that control the structure of a NN for L different functions; then for $l = 1, 2, \ldots, L$, we denote $f_{m_l}(u_1, u_2, m_l, w)$ as the function realized by NN with the masked parameter vector $m_l \odot w$.

Definition 3. Functions: Let $\delta(x) = \begin{cases} 1 & x = 0 \\ 0 & x \neq 0 \end{cases}$. Let l denote the label of functions to be used in simultaneous training and let there be L such functions, which are defined as $\mathcal{D}_l := \sum_{p=1}^{P_l} d_p^l \prod_{k=1}^{2} \delta(u_k - u_{pk}^l), l = 1, 2, \ldots, L$. Since $w \in \mathcal{R}^{1\times N}$, there are at most 2^N different switching configurations and $L \leq 2^N$. A function here is a table (or a dataset) that has two columns for inputs u_1 and u_2 and one column for the target output. The l^{th} table has P_l rows and p^{th} row is $[u_{p1}^l \, u_{p2}^l \, d_p^l]$.

Having the three definitions in mind, the optimization problem can be stated as follows: Let a NN having the structure $2 - R - 1$ be the model to be optimized. Let w be the parameter vector of the NN and let there be L different functions to be learned. Let $m_l \in \mathbb{B}^{1\times N}$ is the mask vector associated to each function. The optimization problem can be given as

$$\underset{m_1, m_2, \ldots, m_L, w}{\text{minimize}} \; J \tag{9}$$

where the loss over all functions is

$$J := \sum_{l=1}^{L} \mathcal{L}(\mathcal{D}_l, f_{m_l}) \tag{10}$$

and the loss over l^{th} function is

$$\mathcal{L}(\mathcal{D}_l, f_{m_l}) := \sum_{p=1}^{P_l} \left(d_p^l - f_{m_l}(u_{1p}, u_{2p}, m_l, w)\right)^2 \tag{11}$$

The concatenated parameter vector (θ) to be optimized over the given NN structure and the given functions is composed of the binary mask vectors $(m_1, m_2, \ldots, m_L)$ and the continuous

parameter vector (w) containing weights and biases, i.e.

$$\theta := [\underbrace{[m_1\, m_2\, \ldots\, m_L]}_{\theta_m}\, w] \in \mathbb{E} \tag{12}$$

where, the extended search space $\mathbb{E} := \mathbb{B}^{1\times(NL)} \cup \mathcal{R}^{1\times N}$. One has to note that the optimization algorithm must find the optimal value of the parameter vector (θ^*) by performing consecutive searches in the extended space $\mathbb{E}$ for the NN having structure $2-R-1$, i.e. 2 inputs, R hidden neurons and one output.

Remark 1: For the problem defined above, use of a gradient based optimization algorithm (e.g. Levenberg-Marquardt algorithm) is computationally infeasible as the NN is subject to structural changes during the optimization process. Further, since the goal is to teach multiple functions to a single NN structure, matrices involved in gradient based algorithms are very likely to produce excessively large matrices, and their possible inversions are factors forcing us to perform a cost-efficient search algorithm, [14]. These facts recommend gradient-free search algorithms and we adopt genetic algorithms for the parameter optimization in binary $(\mathbb{B}^{1\times(NL)})$ and continuous $(\mathcal{R}^{1\times N})$ search subspaces.

Remark 2: After the optimization process is terminated and an optimum or satisfactorily good near optimum is reached, the mask vector may i) keep some entries of w always on for every $\mathcal{D}_l$, ii) keep some entries always off for every $\mathcal{D}_l$, and iii) switch some entries between on and off depending on the relevant function's data $\mathcal{D}_l$.

Algorithm 1: Evolutionary Search for θ_m and w.

1: Initial mask generation $\mathcal{S}^b\{\{\theta_m\}_1^{n_b}\}$
2: Initial parameter vector $\mathcal{S}^c\{w\}$ (w^* initially)
3: **for** $g = 1$ to g_{max} **do**
4: `%Binary part`
5: Parent selection from $\{\theta_m\}_1^{n_b}$
6: Crossover among selected (binary) parents
7: Mutation in $\{\theta_m\}_1^{n_b}$
8: Cost evaluation with $\{\theta_m\}_1^{n_b}$ and w^*
9: Sort and select
10: Choose current best $\theta_m^* \leftarrow \arg\min J(\{\theta_m\}_1^{n_b}, w^*)$
11: `%Continuous part`
12: **if** $g == 1$ **then**
13: Initial parameter generation $\mathcal{S}^c\{\{w\}_1^{n_c-1}\}$
14: Append w^* (used initially) as n_c^{th} candidate
15: **end if**
16: Parent selection from $\{w\}_1^{n_c}$
17: Crossover among selected (continuous) parents
18: Mutation in $\{w\}_1^{n_c}$
19: Cost evaluation with $\{w\}_1^{n_c}$ and θ_m^*
20: Sort and select
21: Choose current best $w^* \leftarrow \arg\min J(\theta_m^*, \{w\}_1^{n_b})$
22: **if** $J == 0$ **then**
23: Exit
24: **end if**
25: **end for**

III. Genetic Algorithms

Genetic Algorithms (GAs) has been successfully applied to wide range of optimization problems. The fact that they do not require derivatives of the loss function make them preferable and under certain circumstances this property is very useful, [20], [21]. Training of NNs using GAs has been reported many times in the literature, [22]. The work of Kotyrba et. al. [15], considers the learning problem as an optimization of NN topology for a given training dataset. Chung et. al., [16], reports GA based tuning of a convolutional NN parameters for a prediction application, Hamdia et. al., [19], considers the GA based tuning of a machine learning model in a broader context. In [17], GAs are used for deep NN training for reinforcement learning and in [18], GA based evolutionary ensemble method utilizing NN based classifiers for solving highly imbalanced classification problems is presented. Cited volume of the literature and the references therein stipulate that GAs constitute a powerful alternative for NN parameter adjustment.

In this section, we will specialize our discussion to the binary search and continuous search algorithms for the loss function in (10) and the parameter vector in (12). The difficulty here is to define the hierarchy of binary and continuous search subroutines. The initial set of mask vectors is formed randomly from $\mathbb{B}^{n_b\times(NL)}$, where n_b is the number of candidate solutions i.e. the number of elements in the population. In order to determine best performing mask vector in this set, we need an initial w vector, which is a randomly selected vector from $\mathcal{R}^{1\times N}$. The best mask vector can now be determined using the expression in (10).

At this stage, a loop over generations (g) is implemented. The first part in the loop handles the binary subroutine and selects parents, performs crossover, mutates and cost evaluation for the next θ_m generation is performed. The best performing mask vector for the g^{th} generation is now known and w has not changed so far.

For the continuous variable optimization, first generation is specially formed. The used w above is included in the first generation, which is from $\mathcal{R}^{n_c\times N}$, and n_c is the population size for GA responsible for optimization variables, i.e. weights and biases. The remaining entries other than the initial w vector are randomly set for the first generation. Continuous variable optimization uses the best mask vector from the above subroutine and produces the next generation utilizing parent selection, crossover, mutation operations, and a cost evaluation using (10) is performed again for the continuous variable optimization subroutine. Now we have the new set of w vectors and loop continues with the binary optimization stage using the best w vector found. This is summarized in Algorithm 1.

The crux of the optimization process is the fact that binary subroutine uses the local best parameter vector (w) and continuous optimization process uses the local best mask vector (θ_m) of the consecutive generations. This process gradually minimizes the cost denoted by J and the optimal or a near optimal solution is found.

In Algorithm 1, the crossover (line 6) is a two input two output operator that receives two vectors based on roulette wheel selection strategy (parents) and performs several exchanges

produce two offsprings. In our study, a random number decides on which strategy to choose for the current parents. Among the equiprobable three outcomes, for the first outcome, we choose single point crossover, for the second, we choose double point crossover and for the last, a random vector $\alpha_b \in \mathbb{B}^{1\times N}$ containing zeros and ones is generated and the two offsprings are obtained via the uniform crossover approach defined as

$$\text{offspring}_1 = \alpha_b \odot \text{parent}_1 + (\mathbf{1} - \alpha_b) \odot \text{parent}_2$$
$$\text{offspring}_2 = (\mathbf{1} - \alpha_b) \odot \text{parent}_1 + \alpha_b \odot \text{parent}_2$$

where $\mathbf{1}$ is a $1 \times N$ vector composed of all ones. The strategy defined above eliminates the drawbacks of using single type crossover and we generate n_{Cb} offsprings, where n_{Cb} is an even number as we generate two offsprings at each trial and it is equal to n_b in this paper.

Mutation operator in line 7 of Algorithm 1 is a single input single output operator that enhances the exploratory capability of the search mechanism. At the g^{th} generation, the i^{th} entry of $\theta_m(g)$, denoted by $\theta_{mi}(g)$, is modified using a uniformly distributed random number, r_m, which varies in between 0 and 1. For a given threshold, say μ_b, if $r_m < \mu_b$, the value of $\theta_{mi}(g)$ is flipped, i.e. $\theta_{mi}(g) \leftarrow 1 - \theta_{mi}(g)$.

At every generation, the population together with the offsprings are evaluated using the loss function, the individuals are sorted according to their performances. The poorest ones are then deleted to maintain the population size constant, i.e. n_b. This ensures that the $(g+1)^{\text{th}}$ population will not contain poorer individuals than the poorest one of the g^{th} generation.

For the continuous optimization subroutine of Algorithm 1, roulette wheel selection is used to determine parents (line 16) and uniform crossover approach defined below is adopted (line 17).

$$\text{offspring}_1 = \alpha_c \odot \text{parent}_1 + (\mathbf{1} - \alpha_c) \odot \text{parent}_2$$
$$\text{offspring}_2 = (\mathbf{1} - \alpha_c) \odot \text{parent}_1 + \alpha_c \odot \text{parent}_2$$

where α_c is a uniform random number that varies in between $(-\gamma_c, 1+\gamma_c)$ with $0 < \gamma_c \ll 1$ being a positive value improving the exploration capability yielding offsprings that are slightly different than ordinary mixture (i.e. $\gamma_c = 0$) of their parents. For the i^{th} entry of the w vector, mutation operator taking place in line 19 of Algorithm 1 implements $w_i(g) \leftarrow w_i(g) + \sigma_c r_{ci}$ if $r_{ci} > \mu_c$ with μ_c being a small positive threshold, $0 < \sigma_c < 1$ being a constant and r_{ci} being a normal distributed random variable. The evaluation, sorting and selection procedure is the same as we discussed for binary case, and the result in g^{th} generation is $w^\star$, which is to be used by the binary subroutine in $(g+1)^{\text{th}}$ generation.

The algorithm defined above generates well performing binary mask population for the current best parameter vector $(w^\star(g))$ and a well performing parameter vector population for the best mask vector $(\theta_m^\star(g))$ until either a predefined number of generations is achieved or $J = 0$ is reached.

As seen from the given discussion, the optimization process considering only binary or only continuous case adopts a standard genetic optimization process. Our paper integrates them in a way that a suitable mask vector accompanied by a parameter vector can be reached to realize multiple functions under different switchings, the outcomes of which will be discussed next.

TABLE I
BINARY FUNCTIONS (TRAINING DATA) FOR THE CLASSIFICATION PROBLEM

u_1	u_2	d_1	d_2	d_3	...	d_{15}	d_{16}
-1	-1	-1	-1	-1	...	1	1
-1	1	-1	-1	-1	...	1	1
1	-1	-1	-1	1	...	1	1
1	1	-1	1	-1	...	-1	1

IV. NN TRAINING VIA OPTIMIZATION OF BINARY AND CONTINUOUS VARIABLES USING GAS

Definition 4. Solvable Problem: Let a general NN have the structure $q - R - 1$. Initialize the NN parameters (w) randomly and choose $2 \le L \ll 2^N$ ($N = 9$ if $q = R = 2$) and $\forall l \in \{1, 2, \ldots, L\}$ generate randomly set mask vector $m_l \in \mathbb{B}^{1\times N}$. Now the NN is ready to respond to any input appearing at its inputs. Under these structural settings, for the l^{th} function, generate a randomly selected input set containing P_l rows, and for each row, obtain the response of the NN, $f_{m_l}(u_{1p}, \ldots, u_{qp}, m_l, w)$. Repeating this for L different cases $(l = 1, 2, \ldots, L)$ would yield L set of datasets, $\mathcal{D}_l$, generated directly by the chosen NN structure.

Starting from an arbitrarily selected binary mask vector $(m_l, l := 1, 2, \ldots, L)$ and parameter vector (w) and optimizing the NN structure to reach the parameters that have generated the training datasets above is called the solvable problem as long as the chosen NN structure is not smaller than $q - R - 1$ structure.

Apparently, a larger sized NN structure, e.g. $q - \tilde{R} - 1$ ($\tilde{R} > R$) instead of $q - R - 1$, can be made equivalent to the original $q - R - 1$ structure by appropriately masking the excess neurons in the hidden layer.

Solvable problem setting ensures the existence of the optimal solution as the training datasets are in the functional and structural span of the $q - R - 1$ NN. This explanation links the lottery ticket hypothesis to our discussion. Initializing a NN having a $q - R - 1$ structure and choosing R very large makes it sure that a subnetwork satisfies the desired input/output mapping to an acceptable accuracy level. Therefore, in a practical case, datasets will be generated by independent sources of information and they will not be in the functional and structural span of the chosen NN, and this will require choosing large R values, parameter/mask optimization and an acceptable precision level.

In this study, we considered two different problems. In the first problem, we considered the realization of the entire set of two variable binary functions, where $u_1 \in \{-1, 1\}$ and $u_2 \in \{-1, 1\}$. We have four lines for each function indicating that $P_l = 4$ for all l. Obviously, the four lines may produce 16 different functions (or maps) and our goal is to obtain a NN structure that realizes all of the mappings given in Table I simultaneously. The optimization trials reported in the sequel assume the values tabulated in Table II and we adopt $\Phi_L(.) = \Phi_R(.) = \text{sgn}(.)$ as the neuronal activation functions.

We conducted 100 experiments for the each $R \in \{2, 3, \ldots, 25\}$ and in each experiment, the genetic optimization

TABLE II
PARAMETERS OF THE GENETIC OPTIMIZATION PROCESS

Variable	Value
g_{max}	1000 and 10000 for classification 20000 for regression
R	$2, 3, \ldots, 25$ for classification 20 for regression
n_c	50
n_b	50
n_{Cb}	50
n_{Cc}	50
μ_b	0.02
μ_c	0.02
σ_c	0.1
γ_c	0.1

TABLE III
NUMBER OF OCCURRENCES AFTER 1000 GENERATIONS FOR $R = 2, 3, \ldots, 25$

R/C	3	4	5	6	7	8	9	10	11	12	13	14	15	16
2	1	8	13	34	27	8	7	2						
3			1	1	12	29	12	15	12	8	1			
4					6	7	16	21	16	23	6	4	1	
5						4	1	1	15	23	19	12	7	
6							3	1	1	19	31	22	4	1
7							3	2	6	1	27	38	13	1
8									3	1	28	39	19	1
9										6	29	4	19	6
10									3	6	19	43	28	1
11										4	15	4	32	9
12									1	3	14	34	42	6
13									1	2	8	57	31	1
14										4	7	46	35	8
15										1	7	52	33	7
16										2	14	42	39	3
17										2	14	4	36	8
18										1	8	44	4	7
19											5	5	35	10
20										3	9	44	38	6
21										2	4	43	43	8
22											6	53	36	5
23										1	9	46	32	12
24										1	3	52	31	13
25										1	8	45	40	6

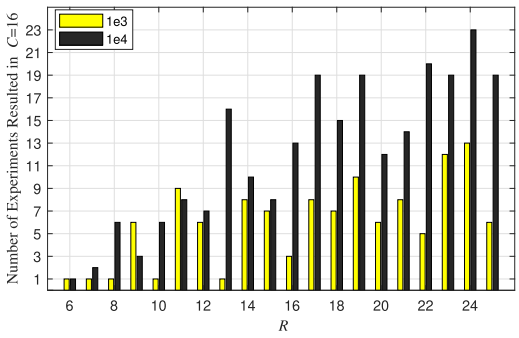

Fig. 2. Number of experiments resulted in correct learning of all 16 functions.

process has been continued for 1000 generations. After an experiment is completed, we counted the number of correctly learned functions denoted by C. The results are given in Table III, where the values in each row sum up to 100, which is the total number number of experiments for each R. The columns are the values of C, and for better understanding, we only entered nonzero values to the table.

In Table III, each experiment completed after 1000 generations takes nearly 2 minutes on an average PC (This duration for 10000 generations is approximately 10 minutes). The findings shown in Tables III and Fig. 2 emphasize the fact that the simultaneous optimization problem has useful and reachable solutions and it becomes more likely to obtain a solution that yields $C = 16$ as R increases. Fig. 2 shows the results after 1000 generations and after 10000 generations together. Interestingly, $R = 6$ describes the smallest NN structure for both cases and experiments lasted for $g_{max} = 10000$ generations result in larger

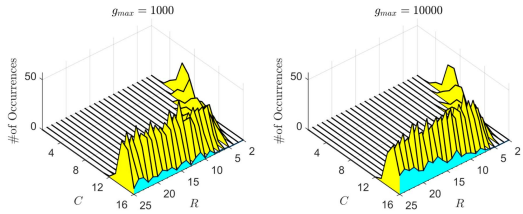

Fig. 3. Distributions of the C values along constant R values indicate that the distributions approach $C = 16$ border (colored face) as R increases.

number of experiments resulting in $C = 16$ for most R values in the horizontal axis compared to $g_{max} = 1000$ experiment set.

According to the results tabulated in Table III, we see that $\max(C)$ when $R = 2$ is 10. This means that in 2 experiments, NN was able to learn 10 columns of Table I correctly. As R increases, we see from the table that NN learns larger number of functions and when $R = 6$, one experiment, bold in Table III, results in correct learning of all 16 functions. $R = 6$ describes the smallest NN structure that can correctly reproduce the map in Table I entirely. If we continue increasing R, the NN has larger degrees of freedom to learn the desired data and the left side of the Table is full of zeros and we leave those entries empty.

Thus, the information available in Table III and Fig. 2 clarify which R values are most likely to result in the correct learning of all 16 functions. Further to this, the results unfold the smallest R value that result in correctly learning of all 16 functions. According to the presented values, we understand that $R = 6$ produces the smallest NN structure that solves the desired learning problem, whose solution is achievable even within 1000 generations.

In Table III, the rightmost column depicting the results for $C = 16$ is of special importance. For the considered R values, the general trend in $C = 16$ column has an increasing nature. Similarly, majority of the considered R values were able to learn 15 columns and slices obtained for each case are shown in Fig. 3, where 1000 and 10000 generations are shown separately to illustrate $C = 16$ trends on the front face.

Remark 3: Vapnik-Chervonenkis (VC) dimension is defined as the maximum number of points that can be placed arbitrarily and labeled arbitrarily yet shattered by a classifier with zero training error. The proposed approach makes the VC dimension of proposed NN classifier equal to 4 (i.e. $C = 16$) over all 16 functions that satisfy arbitrary labeling and zero training error at the same time. This viewpoint assumes the designed mask vector as an integral part of the NN classifier.

Now we will consider the proposed scheme for a regression problem. We considered $L = 4$ functions and used $\Phi_L(x) = \tanh(x)$ as the nonlinear activation function for the hidden layer neurons. We adopted a linear output neuron, i.e. $\Phi_R(x) = x$ for the sole output of the NN structure. In the learning trials, we set $R = 20$ and obtained the results seen in Fig. 4, where the desired and reconstructed data values are shown together. The difference between these surfaces are shown in Fig. 5 and the decrease of the cost along the generations is shown in Fig. 6.

The results seen in these figures stipulate that the learning of multiple functions can be achieved by utilizing the proposed approach and a switching mechanism can be defined to make

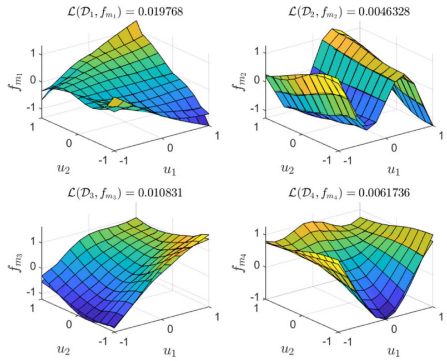

Fig. 4. Desired and the realized functions for $L = 4$ and $R = 20$.

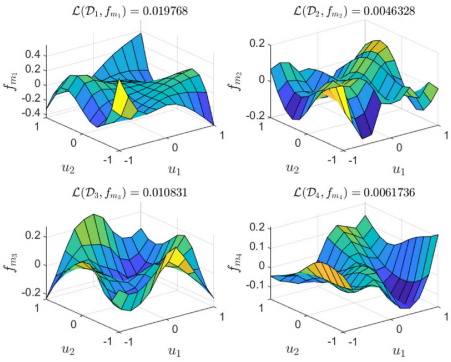

Fig. 5. Error surfaces for the regression problem.

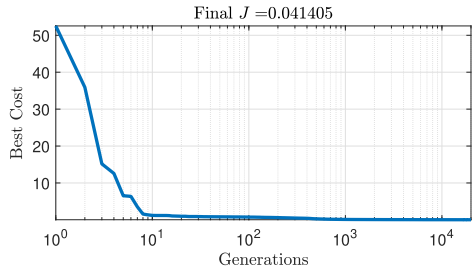

Fig. 6. Time evolution of the cost over 20 000 generations.

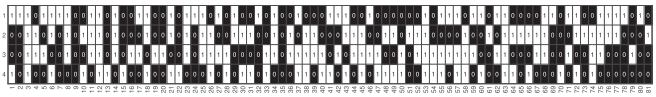

Fig. 7. Mask plot when $R = 20$ for regression among four functions shows that there is no repeating pattern or clustered groups and the algorithm successfully distributes the task and extracts the functional commonalities leading to simultaneous learning.

the NN specific to a chosen function. The mask vectors for this case are illustrated in Fig. 7.

One can increase the number of hidden neurons or describe a multilayer structure and schedule the parameters and mask values in such a way that smaller loss values are observed. The goal in the current work is to obtain simplest structures and discover their unexploited capacities to store new information via appropriate switchings.

Remark 4. Scalability: The proposed approach is applicable to all NN structures yet the effects of the proposed approach

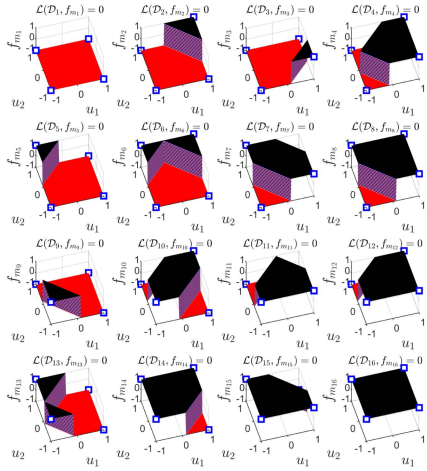

Fig. 8. Results with modified mask when $R = 2$.

are visible better in small sized NN structures than complicated ones. This is primarily because of the use of available limited capacity for multiple purposes. For large NN structures, one has to consider the increased cardinality of the search space, which in turn entails more computational power.

Remark 5. Interpretability: After a proper masking of the neural network that is peculiar to a chosen dataset, the resulting active neural network structure may become a sum of few of the activation functions that simplifies the devised model and that enables the designer to approach the problem analytically. This remedies the difficulties arising from the black box nature of the neural network.

V. A Bipolar Mask for the Binary Classification Problem

In this section, we reconsider the optimization problem by utilizing a modified mask, which adopts $\{-1, +1\}$ instead of $\{0, +1\}$. The motivation behind this operation is the available training data, where the values are ± 1 in Table I and flipping the sign of a variable is supposed to further reduce the smallest R value obtained in the previous section. Such a masking strategy does not kill the functionality of a weight, instead, the weight is determined in such a way that its positive and negative values contribute the learning of different functions, [10]. In Fig. 8, the surfaces generated by the algorithm adopting ± 1 in the mask is shown. In the figure, the desired values of in each subplot is marked by a square.

VI. Modifications for Continuous Parameters

Physical implementation of a trained NN that realizes multiple functions accompanied by a mask ($\theta_m^\star$) and a parameter ($w^\star$) vectors would prefer simple representations. This may force us to consider the following two issues regarding the continuous variables, $w^\star$:

1) The variance of $w^\star$ is minimum, i.e. the numbers in weight and bias matrices are clustered and repetitions are encouraged. The solution to this issue requires modifying

the cost as

$$J := \lambda \mathrm{Var}(w) + \sum_{l=1}^{L} \mathcal{L}\left(\mathcal{D}_l, f_{m_l}\right) \tag{13}$$

where Var(.) stands for the variance operator and $\lambda > 0$ is a weight determining the relative importance. In the current paper, we choose $\lambda = 1$.

2) The entries of $w^\star$ are forced to be integers for easy representation in low cost hardware. This could be handled by rounding all candidate solutions as $\{w\}_1^{n_c} \leftarrow \lfloor \{w\}_1^{n_c} \rceil$.

The results seen in Fig. 8, where $R = 2$ and the NN is exactly as depicted in Fig. 1, can be verified using the following weight, bias and mask values. If only variance compensation is taken without integer representation, all weights and biases converge to 2.1515 and the following mask matrix is obtained. The columns of the below matrix are ordered as given in (7).

$$m = \begin{bmatrix} 1 & 1 & -1 & -1 & 1 & 1 & -1 & 1 & -1 \\ 1 & -1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\ 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 & -1 \\ -1 & -1 & -1 & 1 & -1 & -1 & -1 & -1 & -1 \\ -1 & 1 & -1 & -1 & 1 & 1 & 1 & -1 & -1 \\ -1 & -1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 & -1 & 1 & 1 & -1 & 1 \\ 1 & -1 & 1 & 1 & 1 & 1 & 1 & -1 & 1 \\ 1 & -1 & 1 & -1 & 1 & -1 & -1 & 1 & -1 \\ 1 & -1 & -1 & 1 & 1 & 1 & 1 & 1 & -1 \\ -1 & -1 & 1 & -1 & 1 & -1 & -1 & 1 & 1 \\ 1 & 1 & -1 & -1 & 1 & 1 & 1 & 1 & 1 \\ -1 & 1 & -1 & -1 & 1 & -1 & 1 & -1 & -1 \\ -1 & -1 & 1 & 1 & -1 & 1 & 1 & 1 & 1 \\ -1 & -1 & -1 & -1 & -1 & 1 & 1 & 1 & 1 \\ -1 & 1 & -1 & 1 & -1 & -1 & -1 & -1 & 1 \end{bmatrix} \tag{14}$$

If integer weight values are forced further to the minimum variance solution, we get the following weight, bias and mask values and Fig. 8 depicts the results under these final settings. The order of the parameters in (15) is described in (4) and the column order for m in (16) is given in (7).

$$w^\star = [8\ 8\ 8\ 8\ 7\ 9\ 8\ 8\ 8] \tag{15}$$

$$m = \begin{bmatrix} -1 & -1 & 1 & -1 & 1 & 1 & -1 & -1 & -1 \\ -1 & -1 & -1 & 1 & 1 & -1 & -1 & -1 & -1 \\ -1 & 1 & 1 & -1 & 1 & -1 & -1 & 1 & -1 \\ 1 & 1 & -1 & 1 & 1 & 1 & 1 & 1 & -1 \\ -1 & 1 & 1 & -1 & 1 & 1 & 1 & -1 & -1 \\ -1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & -1 & 1 & -1 & -1 \\ -1 & -1 & -1 & 1 & -1 & -1 & -1 & 1 & 1 \\ -1 & 1 & -1 & -1 & -1 & -1 & 1 & -1 & -1 \\ -1 & -1 & 1 & 1 & 1 & -1 & 1 & -1 & -1 \\ -1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 & -1 \\ -1 & 1 & -1 & -1 & 1 & 1 & -1 & 1 & 1 \\ -1 & -1 & 1 & -1 & -1 & -1 & 1 & 1 & 1 \\ 1 & -1 & 1 & 1 & 1 & 1 & -1 & 1 & 1 \\ -1 & -1 & -1 & 1 & 1 & -1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 & -1 & 1 & -1 & 1 & 1 \end{bmatrix} \tag{16}$$

VII. Comparison of the Proposed Algorithm With the Available Techniques

To position the algorithm proposed in this study, a series of experiments was conducted to compare it with methodologies reported in references [4], [5], [6], [7]. The approach introduced herein is deemed unsuitable for training deep models due to its reliance on a genetic algorithm-based methodology, demanding excessive computational resources. Contrarily, the methodologies detailed in references [4], [5], [6], [7] are more compatible with deep models. However, when considering applicability to shallow models akin to those examined in this study, only the approach presented in [6] is found to be applicable, albeit exhibiting inferior performance compared to the proposed methodology. Uniform initial conditions were established for 1000 experiments conducted, yet none achieved the performance level denoted by $J = 0.0414054$, as demonstrated in the regression problem highlighted in Fig. 4. The approach outlined in [4] involves active pruning, where 50%–75% of the weights are pruned, a method not conducive for shallow models, especially those structured as 2–2–1. Similarly, in [7], certain neurons are identified as redundant, while our proposed approach maintains an overly simplistic structure that does not permit neuron deletion.

Another significant structural consideration involves the utilization of discontinuous activation functions throughout the neural structure. Our proposed approach can accommodate such functions, unlike the methodologies detailed in references [4], [5], [6], [7], as they necessitate continuous output activation functions for generating class probabilities. Furthermore, suitability for regression is an attribute shared by our approach and the one outlined in [6], differing from the algorithms in [4], [5], [7], which lack suitability for regression applications.

While all approaches detailed in references [4], [5], [6], [7], including the proposed one, are deemed suitable for classification applications, the effectiveness of the proposed approach diminishes when applied to deep models. A performance comparison reveals that our proposed technique achieves $J = 0$ for a 2-2-1 neural network structure, an achievement not realized by the alternative methodologies. Notably, the time required to attain a well-trained model is considerably longer for the deep models explored in references [4], [5], [6], [7] compared to the swift training enabled by the proposed approach for shallow models.

A distinguishing factor lies in the utilization of gradient information, prevalent in methodologies detailed in references [4], [5], [6], [7], but notably absent in the proposed technique. The gradient-independent feature of our approach allows for the update of mask vectors, a capability lacking in [4], [6]. In [5], the mask is represented as a real number, thresholded for binary conversion during operation, while in [7], the mask is stored in a Hopfield network. However, these approaches necessitate changes in the mask without directly delineating an update mechanism. Finally, the methodologies detailed in references [4], [5], [6], [7], alongside the current study, have effectively addressed the issue of catastrophic forgetting.

VIII. Extensions of the Proposed Approach

A. Links to Federated Learning

In this section, we discuss how the aforementioned switched learning scheme could enhance the well known Federated Learning (FL), where a central computer receives the weight update information from every client and returns a new weight set. In the FL problem setting, NN structure is fixed among the clients, and clients have their own datasets. The clients do not want to share their data with each other. At each iteration, the central computer distributes the new weight set that generalizes the NN model fitting each user's data locally.

The scheme described above can be enhanced by from the central computer's side such that the central computer distributes the new weight set together with a private key (mask) vector (m_i) that describes the switching vector of each client. Such an approach improves the privacy of the used data during training and makes it difficult to figure out which data belongs to which user while a generalizing model is provided to each client.

B. Links to Dynamical Systems and Control

The proposed scheme is able to store multiple functions. A natural implication of this is to develop data driven models within a single NN structure scheduling different masks for different systems. Let $x \in \mathbb{R}^{n_x}$, $u \in \mathbb{R}^{m_u}$ and a system dynamics be described as in (17).

$$x(k+1) = \begin{cases} F_1(x(k),u(k)) & \text{if condition 1 holds true} \\ F_2(x(k),u(k)) & \text{if condition 2 holds true} \\ \vdots & \vdots \\ F_L(x(k),u(k)) & \text{if condition } L \text{ holds true} \end{cases} \tag{17}$$

Referring to (8), and denoting the NN response as $\hat{x}$, the proposed scheme will yield a compact representation given as

$$\hat{x}(k+1) = f(x(k),u(k),m_i^\star,w^\star), i = 1,2,\ldots,L \tag{18}$$

where m_i is selected if condition i of (17) holds true.

A natural application emerges in Sliding Mode Control (SMC), where the control signal switches in between two different functions and the process is managed by the argument of a signum function. Since signum function has three possible outcomes, a practicing engineer needs to choose $L = 3$ and obtain $m_1^\star$, $m_2^\star$ and $m_3^\star$ as well as $w^\star$ that govern the SMC regimes.

C. Links to Explainable Artificial Intelligence (XAI)

XAI emphasizes the accountability of the building blocks, i.e. their roles, limitations, contributions, pros and cons of their presence/absence and so on. In the proposed approach, the neurons acting in the network are forced to discover and maximize commonalities among the functions and this is observed better as R decreases. In other words, the complementary role of the neurons increases in an optimized NN for small R and this reduces the redundancy thereby making use of the unexploited capacity. As the structure is simplified, the role of each neuron becomes explainable comparably among different switching configurations.

D. Links to Hardware Implementation

Contemporary hardware realizations utilize FPGAs or similar DSP tools frequently and this is very much subject to management of resources as well as simplicity of models defined via software. The approach presented here removes the necessity of storing many models within a resource limited environment, and facilitates the execution of multiple mathematical functions while keeping the structure simple and storing several mask vectors in registers. Further, the solution of the presented approach can be sought in the domain of integer variables or variables having a predefined floating point precision. We demonstrated that a NN structure can be obtained and reachability of it can be maximized by increasing the network complexity, i.e. R in this paper.

IX. Contributions of the Paper

This paper describes a simple yet effective approach to store multiple functions within a single and simple NN structure. The problem is cast into an optimization problem that has binary and continuous variables and an effective solution is proposed. The proposed optimization strategy remedies the catastrophic learning and it can find out the best performing subnetwork even without continuous parameter (weight/bias) modification. If the problem is a type of aforementioned solvable problem, the algorithm finds out the NN that generates the training data. The VC dimension of the designed classifier is equal to 4 for the two variable binary classification example. The algorithm is shown to be effective in multiple function regression. The current study advances the subject area toward multifunctional NNs that may contain dissimilar information in an optimized neuronal structure accompanied by an optimized mask.

X. Conclusion

This paper shows that a NN structure can store information contained within multiple functions via appropriate switching of the NN weights and biases. Switching action either employs a weight/bias value by multiplying it by 1 or deactivates/inverts the parameter by multiplying it by $0/-1$. Considering there are N weight/bias parameters of the NN, 2^N different functions can be stored in a NN structure. Imposing to learn arbitrary functions, the NN structure can be optimized both in continuous weight/bias values and in binary mask parameters using genetic algorithms to minimize the a loss function based on a squared errors defined over L different datasets containing different number of patterns. Our experiments show that an evolutionary approach is able to solve the optimization problem and a NN can be operated by switching its parameters to represent different functions successfully. In the future, by using the the method presented here, given appropriate computational facilities, large sized neural network structures will either be smaller in structure or be multipurpose in function.

REFERENCES

[1] T. L. Liang, J. Glossner, L. Wang, S. B. Shi, and X. T. Zhang, "Pruning and quantization for deep neural network acceleration: A survey," *Neurocomputing*, vol. 461, pp. 370–403, 2021.

[2] Q. Guo, X.-J. Wu, J. Kittler, and Z. Feng, "Weak sub-network pruning for strong and efficient neural networks," *Neural Netw.*, vol. 144, pp. 614–626, 2021.

[3] Y. J. Zheng, S. B. Chen, C. H. Q. Ding, and B. Luo, "Model compression based on differentiable network channel pruning," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 34, no. 12, pp. 10203–10212, Dec. 2023.

[4] A. Mallya and S. Lazebnik, "PackNet: Adding multiple tasks to a single network by iterative pruning," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 7765–7773.

[5] A. Mallya, D. Davis, and S. Lazebnik, "Piggyback: Adapting a single network to multiple tasks by learning to mask weights," in *Proc. Eur. Conf. Comput. Vis.*, 2018, pp. 67–82.

[6] B. Cheung, A. Terekhov, Y. Chen, and P. Agrawal, and B. Olshausen, "Superposition of many models into one," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 32, H. Wallach, H. Larochelle, A. Beygelzimer, Florence d'Alché-Buc, E. Fox, and R. Garnett, Eds., 2019. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2019/file/4c7a167bb329bd92580a99ce422d6fa6-Paper.pdf

[7] M. Wortsman et al., "Supermasks in superposition," in *Proc. 34th Conf. Neural Inf. Proc. Syst.*, 2020, pp. 15173–15184.

[8] H. Zhou, J. Lan, R. Liu, and J. Yosinski, "Deconstructing lottery tickets: Zeros, signs, and the supermask," in *Proc. Adv. Neural Inf. Process. Syst.*, 2019, pp. 3592–3602.

[9] V. Ramanujan, M. Wortsman, A. Kembhavi, A. Farhadi, and M. Rastegari, "What's hidden in a randomly weighted neural network?," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2020, pp. 11893–11902.

[10] N. Koster, O. Grothe, and A. Rettinger, "Signing the supermask: Keep, hide, invert," 2022, *arXiv:2201.13361*.

[11] M. Germain, K. Gregor, I. Murray, and H. Larochelle, "MADE: Masked autoencoder for distribution estimation," in *Proc. the 32nd Int. Conf. Mach. Learn.*, 2015, pp. 881–889.

[12] J. Frankle and M. Carbin, "The lottery ticket hypothesis: Finding sparse, trainable neural networks," 2018, *arXiv:1803.03635*.

[13] J. Kirkpatrick et al., "Overcoming catastrophic forgetting in neural networks," in *Proc. the Nat. Acad. Sci.*, vol. 114, no. 13, pp. 3521–3526, 2017.

[14] J. N. D. Gupta and R. S. Sexton, "Comparing backpropagation with a genetic algorithm for neural network training," *Omega-Int. J. Manage. Sci.*, vol. 27, no. 6, pp. 679–684, 1999.

[15] M. Kotyrba, E. Volna, H. Habiballa, and J. Czyz, "The influence of genetic algorithms on learning possibilities of artificial neural networks," *Computers*, vol. 11, no. 5, May 2022, Art. no. 70.

[16] H. Chung and K.-S Shin., "Genetic algorithm-optimized multi-channel convolutional neural network for stock market prediction," *Neural Comput. Appl.*, vol. 32, pp. 7897–7914, 2019.

[17] F. P. Such, V. Madhavan, E. Conti, J. Lehman, K. O. Stanley, and J. Clune, "Deep neuroevolution: Genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning," 2017, *arXiv:1712.06567*.

[18] P. Z. H. Sun, T. Y. Zuo, R. Law, E. Q. Wu, and A. Song, "Neural network ensemble with evolutionary algorithm for highly imbalanced classification," *IEEE Trans. Emerg. Topics Comput. Intell.*, vol. 7, no. 5, pp. 1394–1404, Oct. 2023.

[19] K. M. Hamdia, X. Zhuang, and T. Rabczuk, "An efficient optimization approach for designing machine learning models based on genetic algorithm," *Neural Comput. Appl.*, vol. 33, pp. 1923–1933, 2020.

[20] S. N. Sivanandam and S. N. Deepa, *Introduction to Genetic Algorithms*. Berlin, Germany: Springer, 2008.

[21] A. Konak, D. W. Coit, and A. E. Smith, "Multi-objective optimization using genetic algorithms: A tutorial," *Rel. Eng. Syst. Saf.*, vol. 91, no. 9, pp. 992–1007, 2006.

[22] X. Yao, "A review of evolutionary artificial neural networks," *Int. J. Intell. Syst.*, vol. 8, no. 4, pp. 539–567, 1993.

Mehmet Önder Efe received the Ph.D. degree from EEE, Bogazici University, Istanbul, Turkey. Having spent a year with Carnegie Mellon and The Ohio State University, Columbus, OH, USA. He was with several private universities in Turkey during 2003–2013. Since 2013, he has been with the Department of Computer Engineering, Hacettepe University, Ankara, Türkiye. His research interests include control systems, neural networks, and autonomous vehicles.

Burak Kürkçü received the B.Sc. degree from Istanbul Technical University, Istanbul, Türkiye, in 2010, and the M.Sc. and Ph.D. degrees from the EEE Department, TOBB ETÜ, Ankara, Türkiye, in 2015 and 2019, respectively. He is currently a Research Scholar with the Department of Mechanical Engineering, University of California, Berkeley, Berkeley, CA, USA. He is also an Assistant Professor with the Department of Computer Engineering, Hacettepe University, Ankara. He was the recipient of IEEE Turkey Ph.D. Thesis Award in 2020.

Coşku Kasnakoğlu (Member, IEEE) received the B.S. degree from the Department of EEE and Department of Computer Engineering, Middle East Technical University, Ankara, Turkey, in 2000, and the M.S. and Ph.D. degrees from the Department of ECE, The Ohio State University, Columbus, OH, USA, in 2003 and 2007, respectively. He is currently a Professor with the Department of EEE, TOBB University of Economics and Technology, Ankara.

Zaharuddin Mohamed received the B.Eng. degree with the Department of EEE, University Kebangsaan Malaysia, Bangi, Malaysia, in 1993, and the M.Sc. and Ph.D. degrees in control systems Engineering from the University of Sheffield, Sheffield, U.K., in 1995 and 2003, respectively. He is currently with the Faculty of Electrical Engineering, University Teknologi Malaysia, Iskandar Puteri, Malaysia. His research interests include control of underactuated systems, robotics, and mechatronics.

Zhijie Liu (Member, IEEE) received the B.Sc. degree in automation from the China University of Mining and Technology Beijing, Beijing, China, in 2014, and the Ph.D. degree in control science and engineering from Beihang University, Beijing, in 2019. In 2017, he was a RA with the Department of EE, University of Notre Dame, Notre Dame, IN, USA, for 12 months. He is currently a Professor with the School of Intelligence Science and Technology, University of Science and Technology Beijing, Beijing. His research interests include adaptive control, modeling and vibration control for flexible structures, and distributed parameter system.