Constrained State-Preserved Extreme Learning Machine

Garrett Goodman
Wright State University
Dayton, United States
garrett.goodman@wright.edu

Cogan Shimizu
Kansas State University
Dayton, United States
coganmshimizu@ksu.edu

Iosif Papadakis Ktistakis
ASML
Wilton, United States
sktistakisp@gmail.com

***Abstract*—Reducing the training time for neural networks is a primary focus of research in the field of machine learning. Currently, the Levenberg-Marquardt (LM) method is one of the fastest backpropagation methods. A recently popular alternative to LM backpropagation is the Extreme Learning Machine (ELM), which produces a closed form optimization of a Single Layer Feed Forward Network for an initially randomized input weight matrix. In this study, we further extend the performance of an ELM by incrementally building on the state of the art, the State-Preserved ELM (SPELM), to produce a Constrained SPELM (CSPELM). To do so, we introduce a constraint, ε, which randomly perturbs the input weight matrix after each training cycle, providing a honing mechanism during the search for a better local optimum. We evaluated CSPELM against 13 benchmark datasets, both categorical and continuous. For 8 of the 13 benchmark datasets, CSPELM outperformed, with respect to average accuracy and RMSE, the ELM, SPELM, and LM methods. Further, the results show that in 8 of the 13 benchmark datasets used, CSPELM was the best performing model and only reached a maximum of 195.10 seconds total training time in one example. The results show a more consistent and higher accuracy than the ELM and SPELM and competitive or better results with LM with training time being only approximately 10% of traditional LM backpropagation training time.**

***Index Terms*—Neural Networks (NN), Training, Machine Learning (ML), Extreme Learning Machine (ELM), Single Layer Feedforward Network (SLFN)**

I. Introduction

Extreme Learning Machines (ELM) have been introduced as another approach to training neural networks in both the Single Layer Feedforward Network (SLFN) [1] case and the Multilayer Feedforward Network (MFN) [2] case. The primary benefit to the ELM technique is the speed in training, as the input weight matrices and hidden biases are randomized rather than trained with the traditional gradient learning methodologies [3]. Then, the output weight matrix is analytically calculated using the Moore-Penrose [4] generalized inverse of a matrix [1].

In this paper, we propose an algorithm based on the most recent version of the ELM, the State Preserving Extreme Learning Machine (SPELM) [5], an ELM which saves the randomized weight matrices to determine the best candidate at the end of the iteration cycle. Our algorithm will create a Constrained State-Preserved Extreme Learning Machine (CSPELM) by constraining the randomization of the input plus hidden bias weight matrices of a SLFN to produce a higher overall accuracy as opposed to complete uniform randomness.

The standard ELM will continuously apply randomly generated weight plus bias matrices to the neural network architecture without taking into account the previous iterations accuracy. Our logical addition to the ELM would introduce an ε which would limit the possible randomized values ($\pm \varepsilon$) for the weight and bias matrices in the pursuit to further increase accuracy with a minor increase to training time. The initial randomization of the ELM input weights and threshold are an approximation of the local optimum; by perturbing this approximation, we hypothesize that it is possible to find a measurably better performing model.

The remainder of this paper is organized as follows. Section II presents the related work in the field of ELM. Then, Section III describes the original ELM and our CSPELM methodology in detail. Following, Section IV provides numerical experiments to show the improvements provided by the CSPELM. Next, Section V provides discussion about the advantages and use cases of the CSPELM. Finally, Section VI concludes the study and poses potential future work.

II. Related Work

The related work covers multiple aspects of the extreme learning technique. For the sake of brevity and concision, the literature discussed herein is representative, not exhaustive.

Huang et al. first introduced the term ELM in [1] in December of 2005 and stated that a new algorithm is needed to replace the conventional implementations of the learning speed of feedforward neural networks. The conventional methods of using gradient learning which is slow and all the parameters are iteratively tuned are the reasons for why they proposed for SLFNs to randomly choose the hidden weights and biases and analytically determine the output weights. In their paper they explain the reasons for this proposal and they show experimentally that ELM in comparison with the conventional algorithms is faster for real life and experimental classification. For example, ELM is 170 times faster than the conventional

Back Propagation (BP) algorithm resulting in both faster speeds and better generalization [1].

The authors then proposed a hybrid learning algorithm based on ELM [6]. They stated that even though the ELM needs a larger number of hidden neurons due to the randomization of the weights of the inputs, this hybrid algorithm will use the Differential Evolutionary (DE) algorithm to select the hidden weights and the Moore-Penrose (MP) inverse to determine the output weights. The hybrid approach called Evolutionary ELM (E-ELM) is shown to achieve greater generalization performance and is claimed to be easier to use with nonlinear hidden units. Though it does not reach the high training speeds that a simple ELM has achieved but is still faster than the conventional methodologies [6].

Huang et al. introduced in [7] that SLFNs work as universal approximators in an incremental constructive method by proving that when randomly selecting the hidden nodes, then one only needs to adjust the output weights by linking the hidden layer and the output layer. Their experimental results showed that their proposed Incremental Extreme Learning Machine (I-ELM) method is not only efficient for the use of continuous activation functions, but also for threshold activation functions. Theoretically, the authors state that the learning algorithms from their constructive method could be applied to both sigmoidal and non-sigmoidal activation functions. They also proposed a special Two hidden Layer Feedforward Network (TLFN) architecture that consists of many SLFNs with additive nodes, locally trained [7].

Huang and Lei Chen, after proposing the I-ELM, state in [8] that they can improve the current algorithm even further. They show that by recalculating the output weights of the existing nodes by applying a convex optimization method, a new algorithm is created that can perform even better. The Convex Incremental Extreme Learning Machine (CI-ELM) randomly generates and adds nodes to the hidden layer and calculates the output weights analytically. They conclude by proving the universal approximation capability of this new algorithm for generalized feedforward networks [8].

Huang and Lei Chen continued their work improving the versions of their algorithm [9]. After I-ELM and CI-ELM, they found that some of the hidden nodes play a minor role in the output and may also increase complexity. To avoid this and obtain a better structure of the network, they proposed an enhanced I-ELM that they refer to as the Enhanced Incremental Extreme Learning Machine (EI-ELM) where at each learning step hidden nodes with the largest residual error decreasing will be added to the network. They proved that by using the new proposed methodology, it works for widespread threshold hidden nodes and it could also be applied to the CI-ELM [9].

Feng et al. proposed in [10] an efficient approach to automatically determine the number of hidden nodes in SLFNs that are not neural alike. They called this new approach the Error Minimized Extreme Learning Machine (EM-ELM) and it could add hidden nodes that are randomly selected either one by one or more at the same time. They proved that the convergence of this approach is faster than other sequential or incremental algorithms with respect to generalization. Moreover, they compared other growing algorithms that add nodes one by one and the new approach is faster from the previous ones [10].

Huang et al. further studied ELM for classification for non-neuron alike networks and extended it to the Support Vector Machine (SVM) algorithm [11]. They showed that by using ELM, the SVM maximal margin property as well as the minimal norm of weights are actually consistent. They also showed that from the standard optimization method, ELM and SVM are equivalent but ELM has less constraints for optimization. Lastly, they analyzed in theory and by simulation results that ELM have better generalization performance [11].

Furthering the exploration of the ELM, Tang et al. state that due to the shallow architecture of the algorithm, feature learning using an ELM might not be as effective for natural signals or with a greater amount of hidden nodes [2]. They propose a new Hierarchical Extreme Learning Machine (H-ELM) which is originally modeled after a multilayer perceptron. Initially, the self-taught feature extraction takes place and then is followed by a supervised classification. The hidden weights are then randomly initialized. An unsupervised multilayer encoding is used for feature extraction for the first time and an autoencoder based on a constraint is developed. Experimental results showed that the new methodology achieves faster convergence and it shows potential application in the computer vision area. Lastly, the new H-ELM methodology outperforms the original ELM algorithm but also other state of the art methods [2].

Another group published a modification of the original ELM algorithm in [5]. The authors of the paper analyze the random generation of weights and state that they could potentially be a part of unstable outputs in the training and testing phases. Using that reasoning they proposed a way to compute the weight matrices in an ELM and save the best performing model which would create an SPELM. The authors evaluated training and testing phases and found that their approach increased the accuracy as well as the performance after they incorporated it with popular feature extraction techniques [5].

Next, Huang et al. showed in [12] that the Least Square Support Vector Machine (LS-SVM) and Proximal Support Vector Machine (PSVM) could be made simpler by creating a unified framework with respect to ELMs. The authors proposed that an ELM could be applied in regression and from an optimization point of view they showed that ELM has milder constraints compared to the previous SVMs. Moreover, their results showed that LS-SVM and PSVM could reach a better solution with higher complexity and that ELM have better scalability and generalization performance [12].

The authors of [13] proposed the Optimally Pruned Extreme

Learning Machine (OP-ELM) that takes the original algorithm and attempts to make it more robust. Their experimental results show that their proposed algorithm performs faster, with the original ELM being the exception, despite its simplicity. The accuracy is also comparable to an SVM [13].

Similar to our proposed methodology, Wentao et al. proposed a Constrained Extreme Learning Machine (CELM) to do discriminative random feedforward neural networks where they showed that the weights connecting the input layer and hidden neurons could be randomly selected from a set instead of a uniform distribution [14]. They showed that their CELM is more stable for discriminative tasks while still having many ELM advantages, thus showing efficiency of CELM in comparison with ELM and other learning techniques such as SVMs and Reduced Support Vector Machines (RSVM) [14].

With the exception of [14], these ELM extensions continue to follow the premise of continual randomization of hidden weights and biases, then analytically calculating the output weights. We believe that continuously randomizing the weights and biases over a uniform distribution may lead to a local optimum and can be further improved upon with a constraint, ε on the randomization after finding said local optimum.

III. Methods

In this section, we present a brief overview of an ELM, then proceed to our CSPELM algorithm.

A. Extreme Learning Machine

As discussed in the introduction, an ELM is a method for training a neural network, both with a single hidden layer or multiple hidden layers. In this paper, we utilize the SLFN version as it is proven to produce a universal approximate of a function [7]. Given any network architecture, the input weights and biases are randomized. Then, the network turns into a linear system of equations.

$$\mathbf{H}\boldsymbol{\beta} = \mathbf{T} \tag{1}$$

From Equation 1, $\mathbf{H} = \{h_{ij}\}$ where $i = 1, ..., N$ and $j = 1, ..., K$ in which N is the number of data points and K is the number of hidden neurons. The h_{ij} is the activation function in which case is generally represented by $\mathrm{g}(\mathrm{w_j} \cdot \mathrm{x_i} + \mathrm{b_j})$ where $\mathrm{w_j}$ is the weight vector $[\mathrm{w_{j1}}, ..., \mathrm{w_{jn}}]^{\mathrm{T}}$ connecting the jth input neuron and hidden neuron. Following, $\mathrm{w_j} \cdot \mathrm{x_i}$ is the inner product of the hidden weight vector and a single input data point vector $\mathrm{x_i}$. Then, the bias vector $\mathrm{b_j}$ is finally added. The variable $\boldsymbol{\beta}$ denotes $[\boldsymbol{\beta}_1, ..., \boldsymbol{\beta}_1]^{\mathrm{T}}$ which is the matrix of output weights. An individual $\boldsymbol{\beta}_{\mathrm{j}}$ is the weight vector $[\boldsymbol{\beta}_{\mathrm{j1}}, ..., \boldsymbol{\beta}_{\mathrm{jm}}]^{\mathrm{T}}$ connecting the jth hidden neuron and output neuron. Finally, $\mathbf{T}$ denotes the matrix of target outputs $[\mathrm{t_1}, ..., \mathrm{t_N}]^{\mathrm{T}}$. The ELM then uses Equation 1 to find the least-square solution. This solution is given by Equation 2.

$$\hat{\boldsymbol{\beta}} = \mathbf{H}^{\dagger}\mathbf{T} \tag{2}$$

Where $\hat{\boldsymbol{\beta}}$ is the calculated output weight matrix given by $\mathbf{H}^{\dagger}$, the Moore-Penrose generalized inverse [4] (otherwise known as the pseudo-inverse) of the matrix $\mathbf{H}$, multiplied by the targets $\mathbf{T}$. Now, we have a network where the input weights and biases are randomly generated and the output weights are calculated from the Moore-Penrose generalized inverse. We note that the ELM is still inherently an iterative process. While it is still possible to obtain a good initial randomization, it is unlikely and will take multiple low time complexity iterations to find satisfactory results.

Now, we can use this SLFN for classification or regression purposes by using linear algebra matrix multiplications to calculate the feedforward of the network given Equation 3 [15].

$$\mathbf{x_n} \overset{\mathbf{W}}{\rightarrow} \mathbf{a_n} \overset{\mathbf{g}}{\rightarrow} \mathbf{z_n} \overset{\hat{\boldsymbol{\beta}}}{\rightarrow} \mathbf{b_n} \overset{\mathbf{h}}{\rightarrow} \hat{\mathbf{y}}_{\mathbf{n}} \tag{3}$$

Equation 3 shows the data point $\mathbf{x_n}$ multiplied by the randomized matrix $\mathbf{W}$ which produces $\mathbf{a_n}$. The result is passed through the activation function $\mathbf{g}$ to produce $\mathbf{z_n}$. That result is multiplied by the least-square solution matrix $\hat{\boldsymbol{\beta}}$ resulting in $\mathbf{b_n}$. This final result is then passed through the activation function $\mathbf{h}$ for the final result $\hat{\mathbf{y}}_{\mathbf{n}}$.

B. Constrained State Preserved Extreme Learning Machine

Having described and discussed the ELM and the feedforward using the trained network, we continue by describing the next steps of improving the state-preserved version of the ELM by adding the ε to the randomization aspect. The SPELM is chosen to build upon instead of a traditional ELM because in order for our algorithm to function, we need a saved state to use as a starting point. As discussed in Section IV, the improvements are impressive with respect to our algorithm. As the SPELM is an iterative process still, the CSPELM builds off of this iteration by constraining the random search ($\pm\ \varepsilon$) from a local optimum produced from the SPELM. This process can be seen in Algorithm 1.

As shown in Algorithm 1, the CSPELM takes in as input a trained SPELM to utilize its weights and biases, the corresponding dataset used to train said SPELM, and the ε randomization constraint. The constraint, ε is seen at the beginning of the algorithm and is used to restrict the interval in which a random value can be generated. We note that the $Num\_Iterations$ variable is user defined. The algorithm also follows a honed search by saving the weights and biases of the best performing model to be used in the next iteration. The weights w and biases b seen in Algorithm 1 are extracted from the SPELM passed via input. There are also five functions used in the algorithm. The function "rand" denotes a function which randomly generates a value between an upper and lower threshold. The function "zeros" generates an m x n sized matrix where each element is zero. The function "in" calculates the inner product of two matrices. The function "MP_Inverse" performs the Moore-Penrose generalized inverse on a matrix.

```
input  : A trained SPELM, dataset, and constraint $\varepsilon$
output: A trained CSPELM
Initialization:
$Num\_Iterations$ := int;
$Best\_Accuracy$ := int;
for $k \leftarrow 0$ to $Num\_Iterations$ do
    for $i \leftarrow 0$ to $Num\_Nodes$ do
        $bias[i]$ = rand($b[i]$ - $\varepsilon$, $b[i]$ + $\varepsilon$);
        for $j \leftarrow 0$ to $Num\_Features$ do
            $weight[i][j]$ = rand($w[i][j]$ - $\varepsilon$, $w[i][j]$ + $\varepsilon$);
        end
    end
    $H$ = zeros($Num\_Samples$, $Num\_Nodes$);
    for $i \leftarrow 0$ to $Num\_Nodes$ do
        for $j \leftarrow 0$ to $Num\_Samples$ do
            $Inner\_Product$ = in($weight[i,:]$, $x[j,:]$);
            $H[j,i]$ = g($Inner\_Product$ + $bias[i]$);
        end
    end
    $H^\dagger$ = MP_Inverse($H$);
    $\hat{\beta}$ = $H^\dagger T$;
    $Results$ = feedforward($weight$, $bias$, $\hat{\beta}$);
    if $Best\_Accuracy$ < $Results$ then
        $w$ = $weight$;
        $b$ = $bias$;
        $Best\_Accuracy$ = $Results$;
    end
end
```

Algorithm 1: The CSPELM algorithm which utilizes a trained SPELM then performs a constrained search with respect the SPELM local optimum weight and bias matrices.

Finally, the function "feedforward" uses the best model, test data, and Equation 3 to perform feedforward to obtain results.

We further expand upon Algorithm 1 by including an iterative shrinking variable to the constraint. So, suppose the user defined constraint is assigned a value of 0.01. We include another variable called ε_Mult which is multiplied by the constraint, ε for a number of iterations to have differing constrained random searches. Algorithm 2 shows this process. The ε, ε_Mult, and $Num\_Iterations$ variables are user defined. This addition allows us to perform a constrained random search at varying intervals. Given the original assigned ε of 0.01 and then choosing a value of 10 for ε_Mult, this allows for multiple iterations of randomization at different constrained intervals starting at 0.10 to 0.01 shrinking by 0.01 per the inner for loop iterating. The process follows a honed search over time, constantly saving the best performing model with respect to accuracy as the next beginning point for weights and biases.

IV. Numerical Experiments

To show the improvement of the CSPELM algorithm, we examine its performance over multiple datasets. Specifically, we

```
Initialization:
$\varepsilon$ := float; // The constraint
$\varepsilon\_Mult$ := int;
$Num\_Iterations$ := int;
for $i \leftarrow 0$ to $Num\_Iterations$ do
    for $j \leftarrow \varepsilon\_Mult$ to 1 do
        $Result$ = CSPELM($ELM$, $data$, $\varepsilon$ * $\varepsilon\_Mult$);
    end
end
```

Algorithm 2: Including the ε_Mult variable to the CSPELM to vary the constraint sizes for the random search.

use the Diabetes (A) [16], Ionosphere (B) [17], Heart Disease (C) [18], Breast Cancer (D) [19], Conditional Grammatical Facial Expressions (E) [20], Satellite (F) [21], Image Segmentation (G) [22], and Optical Recognition of Handwritten Digits (H) [23] datasets for classification. Then, we use the Weather in Szeged (I) [24], Concrete Compressive Strength (J) [25], Parkinson's Telemonitoring (K) with the total_UPDRS outcome variable [26], Combined Cycle Power Plant (L) [27], and Airfoil Self-Noise (M) [28] datasets for regression.

For each dataset, we implemented the same testing philosophies. That is, we split the data into 80% training and 20% test sets unless the dataset provided pre-split training and test sets. Following, the data were normalized using min-max normalization from [-1, 1]. The activation function used in the networks was the sigmoid function of $1/(1+e^{-x})$. Then, to decide the number of hidden nodes, we followed the rule of thumb of $\lfloor (2/3 * \#InNodes) + \#OutNodes \rfloor$ provided in [29]. The exception to this rule of thumb is dataset E, which ran out of memory during training. So, we used 40 hidden nodes for this dataset. For the regression datasets, we utilize the Root Mean Square Error (RMSE) for examining the performance of the network. Every network used for testing was fully connected. Finally, all three algorithms are trained 50 times. For the user defined variables of the CSPELM, we set the $Num\_Iterations$ to 5, ε_Mult to 10, and ε to 0.01. This will allow for 5 iterations at each ε interval totaling to 50 iterations.

The hardware in which these tests were executed on consists of an Intel Core I7-4790K CPU at 4.00 GHz, an AMD Radeon R9 380 Series GPU, and 16.00 GB of RAM. The ELM, SPELM, and CSPELM were coded in Python 3.7.1 with the random seed of 101 for reproducibility and the LM backpropagation was coded in MATLAB R2018a using the pattern recognition neural network toolbox.

We provide the attributes for each dataset shown in Table III, which shows the number of classes, number of features, size of the training set, and size of the test set. The classification test results, shown in Table I, show the mean and standard deviation accuracy percentage, the final/best accuracy, mean and standard deviation training time in seconds, total time

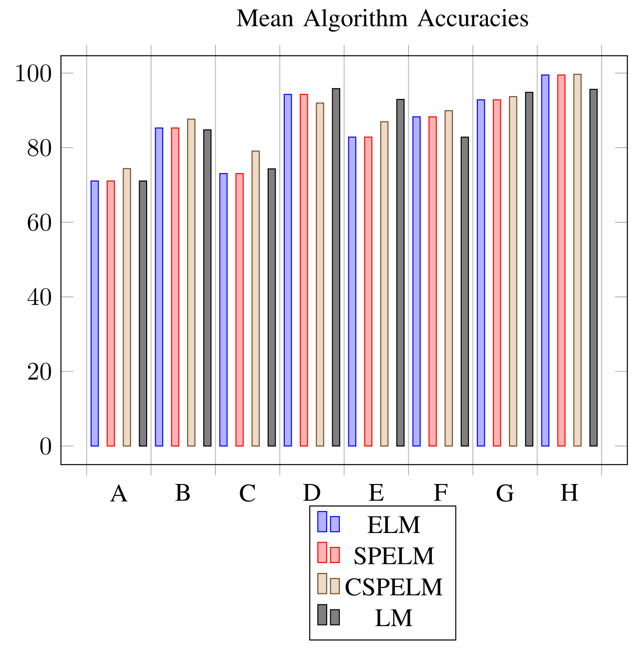

Fig. 1: Histogram of the mean accuracies of all three algorithms with respect to the eight classification datasets. Recall that with accuracy, the higher the value is, the better. From the figure, our CSPELM algorithm performs better in datasets A, B, C, F, and H.

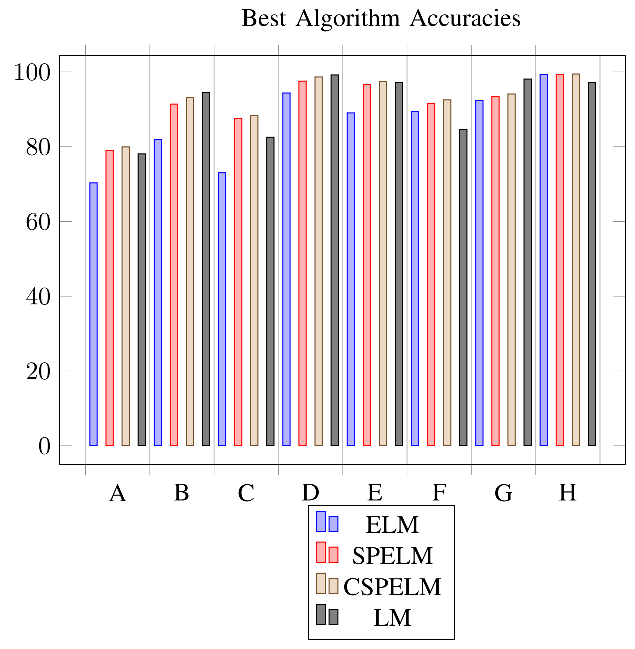

Fig. 2: Histogram of the best accuracies of all three algorithms with respect to the eight classification datasets. Recall that with accuracy, the higher the value is, the better. From the figure, our CSPELM algorithm performs better in datasets A, C, E, F, and H.

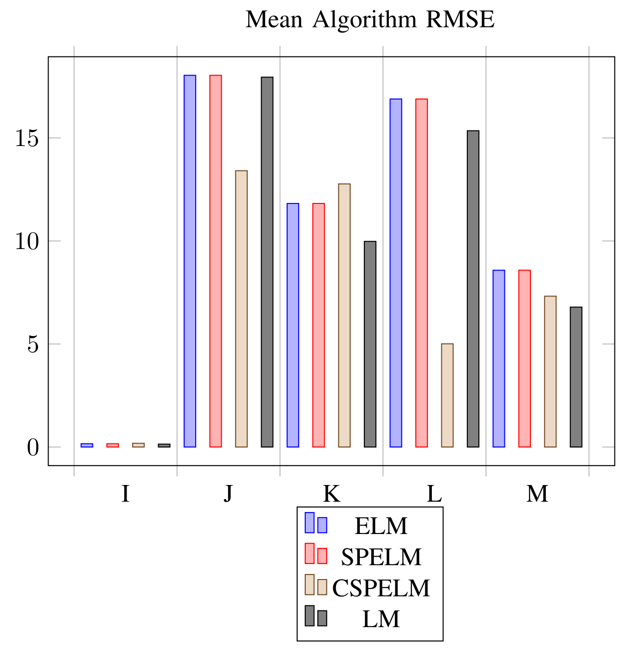

Fig. 3: Histogram of the mean RMSE of all three algorithms with respect to the five regression datasets. Recall that with RMSE, the lower the value is, the better. From the figure, our CSPELM algorithm performs better in datasets J and L.

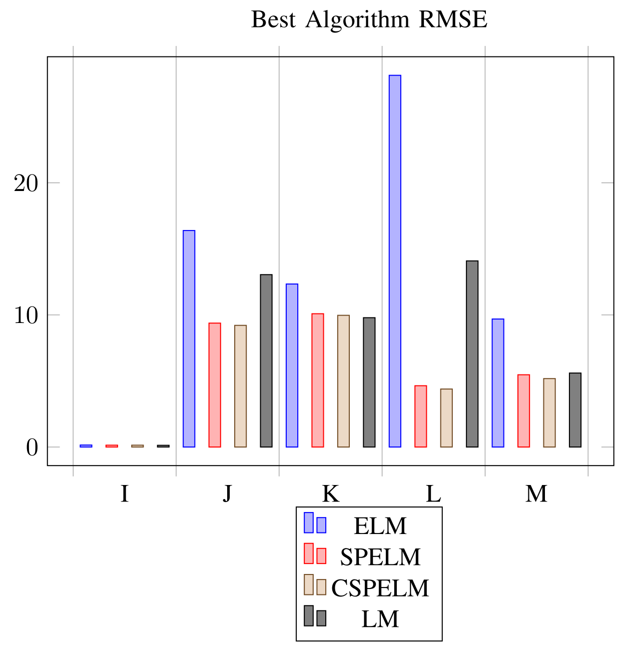

Fig. 4: Histogram of the best RMSE of all three algorithms with respect to the five regression datasets. Recall that with RMSE, the lower the value is, the better. From the figure, our CSPELM algorithm performs better in datasets J, L, and M.

Dataset	Algorithm	Accuracy %	Final/Best Accuracy %	Training Time (s)	Total Time (s)	Hidden Nodes
A	ELM	71.07 ± 3.15	70.32	0.02 ± 2.8x10(-3)	1.19	7
	SPELM	71.07 ± 3.15	78.96	0.02 ± 0.01	1.22	
	CSPELM	**74.39 ± 2.22**	**79.94**	0.03 ± 0.01	2.51	
	LM	71.06 ± 2.22	78.07	0.24 ± 0.29	12.2	
B	ELM	85.28 ± 4.05	81.94	0.03 ± 3.1x10(-3)	1.54	24
	SPELM	85.28 ± 4.05	91.39	0.03 ± 0.01	1.57	
	CSPELM	**87.64 ± 1.12**	93.19	0.03 ± 0.01	3.2	
	LM	84.78 ± 5.16	**94.44**	0.59 ± 0.31	29.49	
C	ELM	73.05 ± 4.58	73.02	0.01 ± 9.4x10(-4)	1.43	10
	SPELM	73.05 ± 4.58	87.50	0.01 ± 2.1x10(-3)	0.59	
	CSPELM	**79.08 ± 1.48**	**88.33**	0.01 ± 2.8x10(-3)	1.25	
	LM	74.32 ± 4.52	82.54	0.24 ± 0.29	12.32	
D	ELM	94.29 ± 2.16	94.36	0.05 ± 0.01	2.34	22
	SPELM	94.29 ± 2.16	97.53	0.05 ± 0.01	2.42	
	CSPELM	91.97 ± 1.56	98.65	0.05 ± 0.01	4.7	
	LM	**95.84 ± 7.10**	**99.19**	0.76 ± 0.43	37.81	
E	ELM	82.83 ± 9.20	89.03	0.27 ± 0.01	13.87	40
	SPELM	82.83 ± 9.20	96.65	0.27 ± 0.02	13.56	
	CSPELM	86.96 ± 1.17	**97.38**	0.29 ± 0.02	28.37	
	LM	**92.94 ± 7.75**	97.13	1141.23 ± 606.48	57061.44	
F	ELM	88.28 ± 2.06	89.36	0.47 ± 0.03	23.57	30
	SPELM	88.28 ± 2.06	91.59	0.45 ± 0.02	22.39	
	CSPELM	**89.91 ± 0.28**	**92.52**	0.45 ± 0.01	44.77	
	LM	82.82 ± 0.74	84.55	18.02 ± 2.84	900.98	
G	ELM	92.84 ± 1.05	92.38	0.19 ± 0.01	9.24	19
	SPELM	92.84 ± 1.05	93.38	0.18 ± 0.01	9.11	
	CSPELM	93.69 ± 0.87	94.09	0.19 ± 0.01	18.37	
	LM	**94.84 ± 3.65**	**98.09**	4.15 ± 1.53	207.64	
H	ELM	99.49 ± 0.23	99.33	0.83 ± 0.02	41.28	52
	SPELM	99.49 ± 0.23	99.37	0.79 ± 0.01	39.83	
	CSPELM	**99.68 ± 0.05**	**99.43**	0.82 ± 0.02	80.74	
	LM	95.64 ± 0.79	97.16	256.56 ± 27.91	12828.08	

TABLE I: Results of the ELM, CSPELM, and LM algorithms with multiple publicly available classification datasets. The bold values represent the best averages and best overall accuracies for each dataset. The CSPELM algorithm had a better overall accuracy for every dataset over the ELM algorithm and was the best algorithm in 5 of the 8 datasets. Note, the CSPELM total training time is its own training time plus the ELM training time (effectively ELM training time doubled).

in seconds, and number of hidden nodes in the network. Similarly, the regression test results in Table II show the mean and standard deviation RMSE, the final/best RMSE, mean and standard deviation training time in seconds, total time in seconds, and number of hidden nodes in the network. We note that the column of final/best accuracy and RMSE in Tables I and II, respectively, is listed as "final/best" as a traditional ELM does not have a saving feature. Therefore, the last iteration is what would be used for the final network. The bold values in both tables show the best performing algorithm for a specific dataset with respect to both the average and best models. In some cases, CSPELM performed better than the ELM, SPELM, and LM backpropagation. Note that the total time for the CSPELM is the SPELM training time plus the CSPELM training time (effectively SPELM training time doubled) as the CSPELM uses a trained SPELM as a local optimum starting point. Finally, we provide histograms of the mean (Figure 1) and best (Figure 2) classification accuracy and mean (Figure 3) and best (Figure 4) RMSE for all thirteen datasets. The histograms help visualize the consistent improvement of the CSPELM over the previous algorithms. To clarify, in Figures 1 and 2, the larger the accuracy the better the algorithm is performing. Then, in Figures 3 and 4, the lower the RMSE the better the algorithm is performing.

V. Discussion

Tables I and II show consistent improvement of the CSPELM best accuracy over the original ELM and SPELM. The training time of a particular model is very fast and the difference in training time between SPELM and CSPELM is negligible. However, between SPELM and CSPELM, we see that CSPELM has approximately 100% increase in total time, but still, in general, 10% or less of LM backpropagation training time, thus remaining competitive. The corresponding increase in training time is negligible in the sense that only datasets H and I reached over one minute total training time. If the user lowers the number of training iterations, it is possible to find an improvement while simultaneously lowering the training time if that is critical to the user. Also, the CSPELM was the best model for dataset H with just 80.74 seconds of training time to reach 99.43% accuracy. Where as the LM backpropagation had a best accuracy of 97.16% with a significant training time increase equaling 12828.08 seconds.

Potential use cases for using the CSPELM algorithm are when using datasets with a considerable amount of features. While dimensionality reduction is a widely used tool for determining the most appropriate features for the task while losing minimal amounts of information [30], this may not always be advantageous. For example, when working in the

Dataset	Algorithm	RMSE	Final/Best RMSE	Training Time (s)	Total Time (s)	Hidden Nodes
I	ELM	0.16 ± 0.01	0.15	1.98 ± 0.03	99.11	5
	SPELM	0.16 ± 0.01	0.14	1.95 ± 0.06	97.58	
	CSPELM	0.18 ± 0.01	0.14	1.95 ± 0.06	195.10	
	LM	**0.14 ± 0.02**	**0.13**	4.62 ± 3.63	230.74	
J	ELM	18.04 ± 2.87	16.39	0.03 ± 3.8x10(-3)	1.35	6
	SPELM	18.04 ± 2.87	9.38	0.03 ± 0.01	1.29	
	CSPELM	**13.41 ± 0.38**	**9.21**	0.03 ± 0.01	2.81	
	LM	17.95 ± 1.99	13.05	0.17 ± 0.23	8.57	
K	ELM	11.82 ± 0.32	12.34	0.25 ± 0.01	12.29	11
	SPELM	11.82 ± 0.32	10.09	0.25 ± 0.02	12.48	
	CSPELM	12.77 ± 0.21	9.97	0.25 ± 0.02	24.74	
	LM	**9.98 ± 0.38**	**9.79**	0.35 ± 0.31	17.34	
L	ELM	16.89 ± 10.98	28.14	0.15 ± 0.01	7.28	3
	SPELM	16.89 ± 10.98	4.64	0.14 ± 0.01	7.02	
	CSPELM	**5.01 ± 0.23**	**4.39**	0.15 ± 0.02	11.41	
	LM	15.35 ± 1.66	14.09	0.25 ± 0.26	12.38	
M	ELM	8.58 ± 2.51	9.69	0.03 ± 4.1x10(-3)	1.43	4
	SPELM	8.58 ± 2.51	5.47	0.01 ± 2.6x10(-3)	1.32	
	CSPELM	7.32 ± 0.55	**5.18**	0.03 ± 4.1x10(-3)	2.70	
	LM	**6.79 ± 1.08**	5.60	0.21 ± 0.23	10.71	

TABLE II: Results of the ELM, CSPELM, and LM algorithms with multiple publicly available regression datasets. The bold values represent the best averages and best overall accuracies for each dataset. The CSPELM algorithm had equal or better overall accuracy for every dataset over the ELM algorithm and was the best algorithm in 3 of the 5 datasets. Note, the CSPELM total training time is its own training time plus the ELM training time (effectively ELM training time doubled).

Dataset	#Classes	#Features	#Training Data	#Test Data
A	2	8	613	155
B	2	34	279	72
C	2	13	240	63
D	2	30	445	124
E	2	301	1524	383
F	6	36	3542	893
G	7	19	2100	210
H	10	64	3823	1797
I	–	6	77162	19291
J	–	8	825	205
K	–	16	4700	1175
L	–	4	7654	1914
M	–	5	1202	301

TABLE III: The datasets used for the comparison between the ELM, CSPELM, and LM algorithms. The "–" represents a regression dataset.

medical field, every piece of information can be important. So, if a backpropagation model takes 57061.44 seconds to train, such as with the model created for dataset E, then utilizing the CSPELM should be a choice to consider. As we can see from dataset E, the CSPELM only took 28.37 seconds of training and outperformed the other algorithms with the best accuracy of 97.38% where the next best algorithm was LM backpropagation of 97.13% at 57061.44 seconds of training.

Next, as the ε and ε_Mult variables are user defined, it is important to be weary of the values assigned to these variables. In order to prevent effectively retraining the SPELM model, we recommend a small constraint value. As the SPELM has the weights and biases randomized uniformly from the interval [-1, 1], having a large constraint such as 0.5 can cause the CSPELM training to essentially ignore the previous best starting weights and biases.

VI. Conclusions

We presented a new improved algorithm based from the State-Preserved Extreme Learning Machine (SPELM) called the Constrained State Preserved Extreme Learning Machine (CSPELM). The CSPELM uses the baseline randomization for the weights and biases determined by a traditional Extreme Learning Machine (ELM) as described in [1] and the state preserving feature of the SPELM described in [5] and adds a constraint on the randomization. That is, we add a constraint, ε to the randomization step ($\pm\ \varepsilon$) as well as a honing mechanism which allows the search to begin with a larger constraint interval and shrinks over multiple iterations. Our results show a more consistent and higher accuracy over the original ELM and SPELM with negligible time increase while still being competitive with traditional backpropagation. Specifically, the CSPELM outperforms the comparison algorithms ELM, SPELM, and Levenberg-Marquardt (LM) backpropagation in 8 of the 13 benchmark datasets. Furthermore, the longest training time taken by CSPELM in any of the datasets was 195.10 seconds. Also, CSPELM was impressively the best algorithm for the Optical Recognition of Handwritten Digits (H) dataset where the training time was just 80.74 seconds reaching an accuracy of 99.43%. The comparison algorithms for dataset H of ELM, SPELM, and LW had accuracies of 99.33%, 99.37%, and 97.16%, respectively and LW took 12828.08 seconds to train.

Potential future work for the CSPELM is twofold. First, examining this methodology on a Multilayer Feedforward Network (MFN) to determine the performance for classification, similar to [2]. Second, examine the possibility of a backtracking system for escaping local minimum during the training.

References

[1] G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew, "Extreme learning machine: theory and applications," *Neurocomputing*, vol. 70, no. 1-3, pp. 489–501, 2006.

[2] J. Tang, C. Deng, and G.-B. Huang, "Extreme learning machine for multilayer perceptron," *IEEE transactions on neural networks and learning systems*, vol. 27, no. 4, pp. 809–821, 2015.

[3] S. S. Haykin, S. S. Haykin, S. S. Haykin, K. Elektroingenieur, and S. S. Haykin, *Neural networks and learning machines*. Pearson education Upper Saddle River, 2009, vol. 3.

[4] D. Serre, *Matrices Theory and Applications*. Springer, 2010, vol. 2.

[5] M. Z. Alom, P. Sidike, V. K. Asari, and T. M. Taha, "State preserving extreme learning machine for face recognition," in *2015 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2015, pp. 1–7.

[6] Q.-Y. Zhu, A. K. Qin, P. N. Suganthan, and G.-B. Huang, "Evolutionary extreme learning machine," *Pattern recognition*, vol. 38, no. 10, pp. 1759–1763, 2005.

[7] G.-B. Huang, L. Chen, C. K. Siew *et al.*, "Universal approximation using incremental constructive feedforward networks with random hidden nodes," *IEEE Trans. Neural Networks*, vol. 17, no. 4, pp. 879–892, 2006.

[8] G.-B. Huang and L. Chen, "Convex incremental extreme learning machine," *Neurocomputing*, vol. 70, no. 16-18, pp. 3056–3062, 2007.

[9] G.-B. Huang and L. Chen, "Enhanced random search based incremental extreme learning machine," *Neurocomputing*, vol. 71, no. 16-18, pp. 3460–3468, 2008.

[10] G. Feng, G.-B. Huang, Q. Lin, and R. Gay, "Error minimized extreme learning machine with growth of hidden nodes and incremental learning," *IEEE Transactions on Neural Networks*, vol. 20, no. 8, pp. 1352–1357, 2009.

[11] G.-B. Huang, X. Ding, and H. Zhou, "Optimization method based extreme learning machine for classification," *Neurocomputing*, vol. 74, no. 1-3, pp. 155–163, 2010.

[12] G.-B. Huang, H. Zhou, X. Ding, and R. Zhang, "Extreme learning machine for regression and multiclass classification," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 42, no. 2, pp. 513–529, 2011.

[13] Y. Miche, A. Sorjamaa, P. Bas, O. Simula, C. Jutten, and A. Lendasse, "Op-elm: optimally pruned extreme learning machine," *IEEE transactions on neural networks*, vol. 21, no. 1, pp. 158–162, 2009.

[14] W. Zhu, J. Miao, and L. Qing, "Constrained extreme learning machine: a novel highly discriminative random feedforward neural network," in *2014 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2014, pp. 800–807.

[15] K. P. Murphy, *Machine learning: a probabilistic perspective*. MIT press, 2012.

[16] J. W. Smith, J. Everhart, W. Dickson, W. Knowler, and R. Johannes, "Pima indians diabetes database," 1988. [Online]. Available: https://www.kaggle.com/uciml/pima-indians-diabetes-database/version/1

[17] V. Sigillito, "Ionosphere data set," 1989. [Online]. Available: https://archive.ics.uci.edu/ml/datasets/Ionosphere

[18] A. Janosi, W. Steinbrunn, M. Pfisterer, and R. Detrano, "Heart disease data set," 1989. [Online]. Available: https://archive.ics.uci.edu/ml/datasets/Heart+Disease

[19] W. H. Wolberg, N. W. Street, and O. L. Mangasarian, "Breast cancer wisconsin (diagnostic) data set," 1993. [Online]. Available: https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+(Diagnostic)

[20] F. A. Freitas, F. V. Barbosa, and S. M. Peres, "Grammatical facial expressions data set," 2014. [Online]. Available: https://archive.ics.uci.edu/ml/datasets/Grammatical+Facial+Expressions

[21] A. Srinivasan, "Statlog (landsat satellite) data set." [Online]. Available: https://archive.ics.uci.edu/ml/datasets/Statlog+(Landsat+Satellite)

[22] C. Brodley, "Image segmentation data set." [Online]. Available: http://archive.ics.uci.edu/ml/datasets/image+segmentation

[23] E. Alpaydin and K. Cenk, "Optical recognition of handwritten digits data set," 1995. [Online]. Available: https://archive.ics.uci.edu/ml/datasets/Optical+Recognition+of+ Handwritten+Digits

[24] N. Budincsevity, "Weather in szeged 2006-2016," 2016. [Online]. Available: https://www.kaggle.com/budincsevity/szeged-weather

[25] I.-C. Yeh, "Modeling of strength of high-performance concrete using artificial neural networks," *Cement and Concrete research*, vol. 28, no. 12, pp. 1797–1808, 1998.

[26] A. Tsanas, M. A. Little, P. E. McSharry, and L. O. Ramig, "Accurate telemonitoring of parkinson's disease progression by noninvasive speech tests," *IEEE transactions on Biomedical Engineering*, vol. 57, no. 4, pp. 884–893, 2009.

[27] P. Tüfekci, "Prediction of full load electrical power output of a base load operated combined cycle power plant using machine learning methods," *International Journal of Electrical Power & Energy Systems*, vol. 60, pp. 126–140, 2014.

[28] R. Lopez, T. Brooks, S. Pope, and M. Marcolini, "Airfoil self-noise data set," 1989. [Online]. Available: https://archive.ics.uci.edu/ml/datasets/Airfoil+Self-Noise

[29] H. B. Demuth, M. H. Beale, O. De Jess, and M. T. Hagan, *Neural network design*. Martin Hagan, 2014.

[30] H. Liu and H. Motoda, *Feature extraction, construction and selection: A data mining perspective*. Springer Science & Business Media, 1998, vol. 453.