Methods for Prediction Optimization of the Constrained State-Preserved Extreme Learning Machine

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

Quinn Hirt
Wright State University
Dayton, United States
hirt.14@wright.edu

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

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

Miltiadis Alamaniotis
University of Texas at San Antonio
San Antonio, United States
miltos.alamaniotis@utsa.edu

Nikolaos Bourbakis
CART Center, Wright State University
Dayton, United States
nikolaos.bourbakis@wright.edu

***Abstract*—Finding the maximum testing accuracy in Machine Learning has been the goal since its conception. From this goal, neural networks have been the primary source of continual improvements in prediction performance. Traditionally, backpropagation has been the primary way of training neural networks and the Levenberg-Marquardt (LM) backpropagation has become the fastest method. Recently, the Extreme Learning Machine was introduced which randomizes weights and biases of hidden layers and uses the Moore-Penrose generalized inverse of a matrix to calculate the output weights and biases, providing competitive results at significantly faster training times. In this study, we continue our work on the Constrained State-Preserved Extreme Learning Machine (CSPELM) with a Forest optimization (CSPELMF) and ε constraint Range-finder (CSPELMR). Furthermore, we provide hyper-parameter settings for the CSPELM to optimize accuracy over training time. Our results show that our methods outperformed the LM backpropagation in a majority of the 13 tested datasets and that the CSPELMF and CSPELMR matched or outperformed the CSPELM in all classification datasets.**

***Index Terms*—Neural Networks, Training, Machine Learning, Extreme Learning Machine, Constrained State-Preserved Extreme Learning Machine, Accuracy Optimization**

I. Introduction

The strive for perfecting training algorithms is ongoing with a multitude of avenues for improvements. For training neural networks, one avenue, that was introduced by Huang and colleagues, was the Extreme Learning Machine (ELM) [1]. They proposed the ELM which showed great success and that can be used for different network architectures such as Single Hidden Layer Feedforward Networks (SLFN) [1], Multilayer Feedforward Networks (MFN) [2], and an interesting inclusion to a Convolutional Neural Network (CNN) by using a Constrained ELM (CELM) to train the fully connected layers of the CNN [3]. In brief, the ELM trains the network by randomizing the hidden layer's weights and biases, then uses the Moore-Penrose generalized inverse of a matrix [4] to calculate the output weights and biases. This in turn trains the network faster and still produces competitive results as compared to traditional Levenberg-Marquardt (LM) backpropagation training.

This work was supported in part by the grant ONR-CART-N00014-18-1-2144.

*Corresponding author.

In our previous study, we introduced the Constrained State-Preserved ELM (CSPELM) as a new algorithm that included two novel aspects to the ELM that improved the results considerably while only sacrificing minimal additional training time [5]. That is, the CSPELM includes an ε constraint to the randomization portion of the algorithm and also a homing variable to allow for different constraint ε values during training. Furthermore, we discussed a set of default hyper-parameter values for the CSPELM that will work adequately for most datasets.

This study introduces two accuracy optimization additions to the CSPELM in the form of a forest optimization and a further improvement on the ε constraint in the form of a range-finder. For the forest optimization, we consider that the starting point provided to the CSPELM may not be the best possible starting point in the n-dimensional gradient space. That is, the State-Preserved Extreme Learning Machine (SPELM) [6], which saves the best performing ELM during n training iterations, is a sufficient starting point used by the CSPELM. Though, due to the general lack of explainability that neural network weights and biases are associated with, other SPELM networks of similar, though perhaps not the highest, accuracies may have a better solution in a completely different area of the n-dimensional gradient space. Therefore, we hypothesize that a forest optimization where each root is a different SPELM of varying training sizes could alleviate this potential issue and provide better prediction results. As for the ε constraint range-finder, the SPELM finds a local minimum in the gradient space or a location close to one. The hypothesis is that by gradually increasing the size of the ε constraint, it can act as a method for escaping the local minimum and thus improving the results. Finally, we also present the best experimentally obtained hyper-parameters for accuracy of the original CSPELM for comparison. These novel additions to the CSPELM are to act as tools for specific problems where one may be more relevant to the problem at hand than another.

The overall organization of the paper is as follows. Section II examines the recent related works of other ELM extensions and algorithms in the recent years in a representative instead of exhaustive manner. Next, Section III discusses the methods purposed in this study. That is, Section III-A presents the ELM in detail, Section III-B clearly describes the CSPELM, and Section III-C precisely shows the forest optimization and ε constraint range-finder algorithms. Following, Section IV details the numerical experiments performed to show the validity of these algorithms. Section V discusses our results and findings. Finally, Section VI concludes the paper and presents future work possibilities.

II. Related Works

We begin the examination of related works on ELM extensions with an article by Zhang et al. They developed and utilized a variant of the ELM, the Online Sequential Outlier Robust ELM (OSORELM), coupled with the Hybrid Mode Decomposition (HMD) method [7]. The purpose of their work was short-term wind speed prediction for use, in particular, with power grid management and wind power integration. Their input data was a time series of wind speeds. This data was pre-processed using the HMD, which is in turn comprised of Variable Mode Decomposition, Sample Entropy, and Wavelet Packet Decomposition. This pre-processing is important to reduce the impact of outlier data on the underlying ELM mechanisms. They also provide an optimized version of the OSORELM through a Criss-Cross Optimization (CSO) that further improves outcomes. Indeed, the OSORELM equipped with HMD and optimized through CSO significantly outperforms the benchmark.

Beyhan et al., utilized Artificial Neural Network Extreme Learning Machines (NELM) and Fuzzy Extreme Learning Machine (FELM) to model U-tube steam generators (UTSG) [8]. The model performances are analyzed by using different numbers of neurons for NELM and different numbers of rules for the FELM. Two comparison methodologies were used to quantify the models: Root Mean Squared Error (RMSE) and Minimum Descriptive Length Criteria (MDL). NELM and FELM models performed similarly for offline identification at each power level, where NELM models performed slightly better than the FELM models in terms of the RMSE measure. Online identification performance is the separating factor between NELM and FELM. RMSE performance of NELM is an order of magnitude better than the FELM in this case.

In Bui et al., they utilize an ELM to compute the susceptibility of a flash flood in a given area [9]. However, unlike in a traditional ELM, the initial weights are not random values, but are instead optimized using Particle Swarm Optimization (PSO). By iteratively refining the performance of the ELM via the PSO algorithm, they were able to achieve significantly better results over the same dataset, when compared to a Multilayer Perceptron-Artificial Neural Network (MLP-ANN), Support Vector Machine (SVM), and a C4.5 Decision Tree. However, they did not compare training time differences between the types of models.

Yaseen et al. propose an Enhanced ELM (EELM) [10]. A traditional ELM uses the Moore-Penrose generalized inverse, which is sometimes implemented via Singular Value Decomposition (SVD). In the EELM, instead, they use Complete Orthogonal Decomposition (COD). This provides them with increased stability, efficiency, and scalability. As a result, when they compared against a standard tool in the hydrology field, Support Vector Regression (SVR), they found that the EELM has a non-trivially better RMSE and Mean Absolute Error (MAE) for their use-case.

In Zhu et al., a novel Constrained Extreme Learning Machine (CELM) algorithm is developed based on the ELM [11]. The new SLFN draws randomly from a set of vectors of the connection weights between the layers (input and hidden neurons) that is comprised of between class samples, thus making it a constrained set. While CELM is a simple linear system, same as the ELM, the inclusion of a small set of weights showed the advantages over ELM while keeping the latter's advantages. The gain from comparing the two is the sample distribution that makes a better discriminative feature mapping allowing better performance on the CELM on both synthetic and real-world datasets. Having changed nothing else though, they keep the negatives of the ELM approach such as overfitting when there is a large number of hidden nodes.

Weng et al., developed a model for land use classification from high resolution remote sensing imagery [3]. They utilized a CNN and a CELM in a hybrid fashion. They removed the fully connected layers from the CNN and, after normalizing the deep convolutional features, they fed it to the CELM. By using the proposed methodology on high resolution data sets, they managed to improve the generalization ability and reduced training time by using the novel CNN-CELM model while avoiding overfitting of the original ELM. An extension is discussed as the current model can't be used for High Resolution Remote Sensing (HRRS) images.

Finally, the author developed a synergy for smart loading forecasting for a day ahead horizon [12]. This synergy is implemented by connecting a Deep Neural Network (DNN) and an ELM. There are two steps in the process of the synergy that start with the DNN making a day ahead forecast, followed by the ELM classifying the most observed values to tune then the DNN for the next half hour. Real world datasets were used and benchmarked against the learning Gaussian process resulting in better performance and accuracy over the latter. From these papers, there is lacking optimization techniques for the authors proposed methods and instead focused on creating problem specific techniques. Our study aims to provide tools that have the capability of being used for multiple problems.

III. Methods

In this section, we provide the details on the ELM in Section III-A, the algorithms and explanation of the CSPELM in Section III-B, and our accuracy optimization techniques in Section III-C.

A. Extreme Learning Machine

The ELM is a training methodology for SLFN [1], MFN [2], and hybrid CNN models [3]. In this description of the ELM, we present the SLFN version. The ELM functionality begins by randomizing the weights and biases of the hidden layer. Then, the network turns into a system of linear equations to calculate the output weights and biases.

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

So, from Equation 1, we have $\boldsymbol{H}$ which represents $\{h_{ij}\}$ where $i = 1, ..., N$ and $j = 1, ..., K$. Here, N is the size of the dataset and K is the number of hidden nodes. Further, $\{h_{ij}\}$ is the activation function represented by $g(w_j \cdot x_i + b_j)$. In this function, w_j is simply the weight vector, x_i is the current data point vector, and b_j is the bias vector. Following, $\boldsymbol{\beta}$ denotes the output weight matrix of $[\beta_1, ..., \beta_j]^T$. From this, every β_j is an individual weight vector. Finally, $\boldsymbol{T}$ represents the matrix of target outputs $[t_1, ..., t_N]^T$. Equation 1 is used in the calculation of the least-square solution. The least-square solution is given by Equation 2.

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

From Equation 2, $\hat{\boldsymbol{\beta}}$ is the result of the least-square calculation given by $\boldsymbol{H}^\dagger$ and $\boldsymbol{T}$. First, $\boldsymbol{H}^\dagger$ is the Moore-Penrose generalized inverse (represented by the †) of $\boldsymbol{H}$ which is then multiplied by the output targets $\boldsymbol{T}$. Note that the ELM is an iterative process where multiple iterations of randomization and least-square calculations are made to find a suitable result.

B. Constrained State-Preserved Extreme Learning Machine

Before discussing the CSPELM, we note that it is built upon the SPELM and not directly the ELM. The SPELM, as the name implies, saves the best state of weights and biases that provided the highest performance during the training iterations. This is a necessary component to the CSPELM as it takes in as input the weights and biases from an SPELM and continues to perturb these values. Specifically, the CSPELM introduces a user specified ε constraint to the randomization process ($\pm\varepsilon$).

Algorithm 1 shows the CSPELM training procedure with the ε constraint included in the randomization process. It takes in as input a trained SPELM to utilize the weights and biases of the hidden layer, the dataset in question, and the ε constraint. As we see from the beginning of the algorithm, the ε constraint shrinks the size of the randomization interval for both the weights and biases. The functions used within Algorithm 1 are as follows. Function "rand" generates a random float between the interval a and b. Function "zeros" creates an mxn matrix filled with zeros. Function "in" calculates the inner product between two matrices. Function "MP_Inverse" calculates the Moore-Penrose generalized inverse of the matrix. Finally, function "feedforward" tests the current model's performance. The CSPELM saves the current model for further perturbing if the current model outperforms the previous model.

Algorithm 1: The CSPELM algorithm which takes in as input a trained SPELM, dataset, and user specified ε.

```
input : A trained SPELM, dataset, and constraint $\varepsilon$
output: A trained CSPELM
Initialization:
$Best\_Result$ := int;
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($H$);
$\hat{\beta}$ = $H^\dagger T$;
$Result$ = feedforward($weight$, $bias$, $\hat{\beta}$);
if $Best\_Result$ < $Model.Result$ then
    $w$ = $weight$;
    $b$ = $bias$;
    $Best\_Result$ = $Result$;
end
```

Note that this is only one iteration of the CSPELM training process. Recall that the CSPELM is inherently an iterative process. Thus, Algorithm 2 is presented which calls the CSPELM training algorithm function and continuously iterates to find the best solution. Here, the ε_Mult variable is included as the previously discussed homing variable. This variable is used as the inner for-loop's size variable and shrinks to 1 over time. This produces an effect which causes the ε constraint to start larger and to shrink as the algorithm iterates, providing the homing functionality. Note that the ε, ε_Mult, and $CSPELM\_Iter$ variables are user defined. Though, it is important to realize the gravity of the ε and ε_Mult value selection as if they are set to large, the result is effectively retraining with no constraint and ignoring the benefits of the SPELM starting point.

Algorithm 2: The algorithm for using the CSPELM. The user specifies the ε constraint, the ε_Mult homing variable, and the $CSPELM\_Iter$ variable. As the inner loop iterates, the size of the ε constraint decreases to introduce the homing effect.

```
Initialization:
$\varepsilon$ := float;
$\varepsilon\_Mult$ := int;
$CSPELM\_Iter$ := int;
$Best\_Result$ := float;
$Best\_Model$ := list();
for $i \leftarrow 0$ to $CSPELM\_Iter$ do
    for $j \leftarrow \varepsilon\_Mult$ to $1$ do
        $Model$ = CSPELM(SPELM, data, $\varepsilon$ * $j$);
        if $Best\_Result$ < $Model.Result$ then
            $Best\_Model$ = $Model$;
            $Best\_Result$ = $Model.Result$;
        end
    end
end
```

C. Accuracy Optimization Techniques

As previously discussed, we introduce two novel accuracy optimization techniques to the CSPELM. First, a forest optimization is introduced which utilizes multiple SPELM, trained with a varying number of iterations, as roots to train the CSPELM under the pretense that different starting positions in the n-dimensional gradient space could lead to a better solution. Following, the ε range finder is utilized as a method of escaping local minimums.

Beginning with the forest optimization technique, we utilize the same theory of the CSPELM training but change the SPELM input into Algorithm 1. Previously, it was simply a single SPELM with the best performing weights and biases of a user defined number of training iterations. Now, we train multiple SPELM with varying numbers of iterations per model to act as the roots in the forest. That is, a user defined number of roots is chosen, say $n = 10$, then 10 SPELM are trained with the number of iterations for model one being 1, model two being 2, ..., model ten being 10. This allows us to achieve the benefits of an SPELM input while lowering the overall training time before the CSPELM training stage. Following, a CSPELM is trained per root while keeping track of the best performing model of the CSPELMs. This process can be seen in Algorithm 3. From the algorithm, the user defines the ε, ε_Mult, $\#Roots$, and $CSPELM\_Iter$ variables. The two functions used within this algorithm are the SPELM and CSPELM functions which simply train and return the corresponding models, respectively. We utilize the acronym of CSPELM Forest (CSPELMF) for this method.

Algorithm 3: The forest optimization technique for the CSPELM. The user specifies the ε, ε_Mult, $\#Roots$, and $CSPELM\_Iter$ variables. This trains an SPELM at varying number of training iterations for each root in the forest. Then, a new CSPELM is trained per SPELM root.

```
Initialization:
$\varepsilon$ := float;
$\varepsilon\_Mult$ := int;
$Best\_Result$ := float;
$Best\_Model$ := list();
$Roots$ := list();
$\#Roots$ := int;
$CSPELM\_Iter$ := int;
for $i \leftarrow 0$ to $\#Roots$ do
    $Train\_Iterations$ = $i + 1$;
    $Roots$.append(SPELM($Train\_Iterations$, data));
end
for $Root$ in $Roots$ do
    for $i \leftarrow 0$ to $CSPELM\_Iter$ do
        for $j \leftarrow \varepsilon\_Mult$ to $1$ do
            $Model$ = CSPELM($Root$, data, $\varepsilon$ * $j$);
            if $Best\_Result$ < $Model.Result$ then
                $Best\_Model$ = $Model$;
                $Best\_Result$ = $Model.Result$;
            end
        end
    end
end
```

Next is the ε range finder that acts as a method to escape local minimums that the CSPELM may encounter. This works by defining a list of ε and ε_Mult, respectively, to allow for multiple gradually increasing in size combinations of the two variables. For example, say the user defines the ε_List and ε_Mult_List variables to be $[5, 10]$ and $[0.01, 0.02]$, respectively. This results in the training from Algorithm 2 but with two additional loops to iterate combinations between these two lists. So, we would utilize the training from Algorithm 2 for ε_Mult and ε combinations of $[5, 0.01]$, $[5, 0.02]$, $[10, 0.01]$, and $[10, 0.02]$, respectively. This in theory mimics taking larger learning and network updating steps in traditional gradient descent as a way to escape local minimums. Shown in Algorithm 4, the user defines the ε_List, ε_Mult_List, and $CSPELM\_Iter$ variables. The only function used in Algorithm 4 is the CSPELM training function to return a trained network. We utilize the acronym of CSPELM Rangefinder (CSPELMR) for this method. Finally, we note that while it is technically possible to combine the CSPELMF and CSPELMR into a CSPELMFR, the complexity rises substantially and the cost greatly outweighs the benefits.

IV. Numerical Experiments

We now perform the numerical experiments utilizing the new accuracy optimization CSPELMF and CSPELMR techniques as shown in Section III-C and compare them to the CSPELM

Algorithm 4: The algorithm for the CSPELM ε range finder. The user specifies the ε_List, ε_Mult_List, and $CSPELM\_Iter$ variables. This allows for multiple ε and ε_Mult combinations to assist in escaping local minimums by gradually increasing the constraint size and mimicking larger steps in gradient descent.

```
Initialization:
$\varepsilon\_List$ := list();
$\varepsilon\_Mult\_List$ := list();
$CSPELM\_Iter$ := int;
$Best\_Result$ := float;
$Best\_Model$ := list();
for $\varepsilon\_Mult$ in $\varepsilon\_Mult\_List$ do
    for $\varepsilon$ in $\varepsilon\_List$ do
        for $i \leftarrow 0$ to $CSPELM\_Iter$ do
            for $j \leftarrow \varepsilon\_Mult$ to 1 do
                $Model$ = CSPELM(SPELM, data, $\varepsilon * j$);
                if $Best\_Result < Model.Result$ then
                    $Best\_Model = Model$;
                    $Best\_Result = Model.Result$;
                end
            end
        end
    end
end
```

and LM Backpropagation methods. We do not compare the original ELM and SPELM as our previous study showed that the CSPELM consistently outperforms them [5]. The datasets used for the experiments are the same as from our previous study [5] and consist of both classification and regression datasets. For classification, we have the Diabetes (A) [13], Ionosphere (B) [14], Heart Disease (C) [15], Breast Cancer (D) [16], Conditional Grammatical Facial Expressions (E) [17], Satellite (F) [18], Image Segmentation (G) [19], and Optical Recognition of Handwritten Digits (H) [20] datasets. Following for regression, we have the Weather in Szeged (I) [21], Concrete Compressive Strength (J) [22], Parkinson's Telemonitoring with the total_UPDRS target variable (K) [23], Combined Cycle Power Plant (L) [24], and Airfoil Self-Noise (M) [25] datasets. The number of classes, features, training data, and test data can be seen in Table I.

The experimental setup is as follows. The data split for training and testing was 80% and 20%, respectively, unless the dataset provided a pre-defined training and test split. Next, the number of fully connected neurons in the single hidden layer was determined using the rule of thumb method $\lfloor(2/3 * \#InNodes) + \#OutNodes\rfloor$ [26]. The only exception to this comes from dataset E which has 301 features and the system executing these experiments runs out of available memory during training. Thus, 40 hidden neurons were chosen for this network. Following, all data were normalized to $[-1, 1]$. The sigmoid activation function was used for all experiments. For the classification datasets, we present the accuracy as the primary performance metric. For the regression datasets, we present the RMSE as the primary performance metric. Finally, as we attempt to achieve the highest accuracy possible for each dataset, the hyper-parameters for each dataset differ and are shown in Table II. The adjustable hyper-parameters for the CSPELM are the ELM_Iter, $CSPELM\_Iter$, ε_Mult, and ε. For the CSPELMF, they are the $\#Roots$, $CSPELM\_Iter$, ε_Mult, and ε. For the CSPELMR, they are the ELM_Iter, ε_Mult_List, and ε_List. Lastly, for the LM Backpropagation, the only hyper-parameter is the $LMIterations$ variable.

TABLE I: The dataset attributes used in the numerical experiments. Note, "—" represents a regression dataset.

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

The hardware used to execute the experiments was an Intel Core i5-9600K CPU at 3.70 GHz, NVIDIA GeForce GTX 1070 Ti, and 16.00 GB of DDR4 RAM. The CSPELM and the accuracy optimization techniques were coded in Python 3.7.1 while using the random seed 2998322090 for reproducibility. The LM Backpropagation was coded using MATLAB R2018a with the pattern recognition toolbox.

We present the experimental results in Tables III and IV for classification and regression datasets, respectively. For both tables, the mean ± standard deviation of the results, best results, mean ± standard deviation training time in seconds, total training time in seconds, and number of hidden nodes used in the network are shown. The bold values represent the highest performance for each individual dataset.

V. Discussion

We begin by examining the classification and regression results shown in Tables III and IV, respectively. For the classification, we can see that our three methods greatly outperform the LM backpropagtion in a majority of datasets. The only exceptions being datasets D and G where the LM backpropagtion was the best performing algorithm. As for the CSPELMF and CSPELMR accuracy optimization techniques, one of the two, or both, matched or exceeded the CSPELM results in all cases. We also note that our methods found an astonishing result of 100% classification result for dataset H, the Optical Recognition of Handwritten Digits [20] dataset. As for the regression results, the optimization hyper-parameter settings

TABLE II: The hyper-parameter settings for each algorithm per dataset. The "—" means that the algorithm does not require that hyper-parameter.

Dataset	Algorithm	$\#Roots$	ε_Mult_List	ε_List	ELM_Iter	CSPELM_Iter	ε_Mult	ε	LM Iterations
A	CSPELM	—	—	—	400	30	10	0.05	—
	CSPELMF	19	—	—	—	5	10	0.01	—
	CSPELMR	—	2, 4, 6, 8	0.01, 0.02, 0.03, 0.04	50	5	—	—	—
	LM	—	—	—	—	—	—	—	35
B	CSPELM	—	—	—	250	30	10	0.04	—
	CSPELMF	25	—	—	—	5	10	0.01	—
	CSPELMR	—	1, 5, 10, 20	0.01, 0.02, 0.03, 0.04	50	5	—	—	—
	LM	—	—	—	—	—	—	—	30
C	CSPELM	—	—	—	175	10	10	0.02	—
	CSPELMF	25	—	—	—	5	10	0.01	—
	CSPELMR	—	1, 5, 10	0.01, 0.02, 0.03, 0.04, 0.05	50	5	—	—	—
	LM	—	—	—	—	—	—	—	50
D	CSPELM	—	—	—	50	5	10	0.01	—
	CSPELMF	12	—	—	—	5	10	0.01	—
	CSPELMR	—	1, 5	0.01, 0.02, 0.03, 0.04, 0.05	50	5	—	—	—
	LM	—	—	—	—	—	—	—	40
E	CSPELM	—	—	—	500	30	10	0.03	—
	CSPELMF	50	—	—	—	5	10	0.01	—
	CSPELMR	—	1, 5, 10	0.01, 0.02, 0.03, 0.04, 0.05	50	5	—	—	—
	LM	—	—	—	—	—	—	—	30
F	CSPELM	—	—	—	1000	50	10	0.05	—
	CSPELMF	20	—	—	—	5	10	0.01	—
	CSPELMR	—	10, 20	0.01, 0.02, 0.03, 0.04, 0.05	50	5	—	—	—
	LM	—	—	—	—	—	—	—	45
G	CSPELM	—	—	—	300	5	10	0.03	—
	CSPELMF	10	—	—	—	5	10	0.01	—
	CSPELMR	—	1, 5, 10	0.01, 0.02, 0.03, 0.04, 0.05	50	5	—	—	—
	LM	—	—	—	—	—	—	—	30
H	CSPELM	—	—	—	500	5	10	0.02	—
	CSPELMF	35	—	—	—	5	10	0.01	—
	CSPELMR	—	1, 5, 10	0.01, 0.02, 0.03, 0.04, 0.05	50	10	—	—	—
	LM	—	—	—	—	—	—	—	50
I	CSPELM	—	—	—	25	5	10	0.01	—
	CSPELMF	10	—	—	—	5	10	0.01	—
	CSPELMR	—	1, 5	0.01, 0.02	50	5	—	—	—
	LM	—	—	—	—	—	—	—	5
J	CSPELM	—	—	—	50	5	10	0.02	—
	CSPELMF	20	—	—	—	5	10	0.01	—
	CSPELMR	—	1, 5, 10	0.01, 0.02, 0.03, 0.04	50	5	—	—	—
	LM	—	—	—	—	—	—	—	25
K	CSPELM	—	—	—	100	5	10	0.02	—
	CSPELMF	15	—	—	—	5	10	0.01	—
	CSPELMR	—	1, 5, 10	0.01, 0.02, 0.03, 0.04	50	5	—	—	—
	LM	—	—	—	—	—	—	—	5
L	CSPELM	—	—	—	45	5	10	0.01	—
	CSPELMF	20	—	—	—	5	10	0.01	—
	CSPELMR	—	1, 5	0.01, 0.02, 0.03, 0.04, 0.05	50	5	—	—	—
	LM	—	—	—	—	—	—	—	20
M	CSPELM	—	—	—	45	5	10	0.01	—
	CSPELMF	10	—	—	—	5	10	0.01	—
	CSPELMR	—	1, 5, 10	0.01, 0.02, 0.03	50	5	—	—	—
	LM	—	—	—	—	—	—	—	15

for the CSPELM showed considerable improvements over our previous studies suggested default settings. Though, the CSPELMF and CSPELMR appear to have difficulties with regression datasets. We consider that the lack of features for the regression datasets could have an effect on the outcome of these optimization extensions. That is, from Table I, the regression datasets #Features column shows that the regression datasets have between 4-16 features while the classification datasets have between 8-301 features.

We reiterate the use-cases for our methodologies. That is, from our representative related works overview, we find a lack of non-problem specific methodologies while ours show impressive results for a wide variety of datasets. The CSPELM can be used from either a low training time hyper-parameter setting while producing competitive results [5] or, if tuned correctly, can sacrifice training time to produce some of the best results possible as shown here. For the CSPELMF, it can be applied to situations for when the user encounters multiple starting weights that train to the same local minimum. The forest optimization's large range of starting network weights can possibly allow for a better starting position in the n-dimensional gradient space. Finally, for the CSPELMR, the

TABLE III: The classification results for the CSPELM, CSPELMF, CSPELMR, and LM Backpropagation. The bold values represent the best results for each dataset.

Dataset	Algorithm	Accuracy %	Final/Best Accuracy %	Training Time (s)	Total Time (s)	#Hidden Nodes
A	CSPELM	75.66 ± 2.61	**80.65**	0.03 ± 1*10(-3)	20.39	7
	CSPELMF	74.34 ± 2.39	**80.65**	4.09 ± 2.83	155.61	7
	CSPELMR	**75.99 ± 1.79**	78.71	0.81 ± 0.38	13.73	7
	LM	70.34 ± 3.29	78.07	0.27 ± 0.35	9.42	7
B	CSPELM	**94.19 ± 2.53**	**98.61**	0.04 ± 2*10(-3)	22.76	24
	CSPELMF	92.00 ± 2.20	97.22	1.31 ± 0.79	65.41	24
	CSPELMR	91.38 ± 3.09	**98.61**	1.93 ± 1.49	32.73	24
	LM	85.23 ± 5.28	94.44	0.54 ± 0.42	16.22	24
C	CSPELM	**81.64 ± 3.03**	**87.3**	0.02 ± 1*10(-3)	4.36	10
	CSPELMF	80.03 ± 2.47	**87.3**	0.52 ± 0.33	25.82	10
	CSPELMR	78.58 ± 3.45	85.71	0.45 ± 0.29	7.23	10
	LM	73.97 ± 4.49	80.95	0.25 ± 0.29	12.23	10
D	CSPELM	**97.37 ± 1.06**	98.39	0.06 ± 4*10(-3)	5.81	22
	CSPELMF	96.59 ± 0.96	98.39	1.73 ± 1.36	41.48	22
	CSPELMR	96.15 ± 2.05	98.39	7.91 ± 2.97	86.98	22
	LM	95.63 ± 8.22	**99.19**	0.68 ± 0.51	27.35	22
E	CSPELM	89.13 ± 6.94	96.61	0.40 ± 4*10(-3)	305.85	40
	CSPELMF	92.63 ± 3.23	**97.65**	13.82 ± 6.14	1381.47	40
	CSPELMR	84.70 ± 13.74	97.13	11.15 ± 7.37	178.44	40
	LM	**93.52 ± 6.59**	97.39	707.68 ± 410.81	21230.32	40
F	CSPELM	90.96 ± 1.39	93.17	0.63 ± 0.01	942.39	30
	CSPELMF	**92.28 ± 0.94**	93.17	24.41 ± 16.31	976.26	30
	CSPELMR	91.17 ± 1.06	**93.51**	58.56 ± 20.16	644.17	30
	LM	82.01 ± 4.21	85.11	14.18 ± 2.59	638.09	30
G	CSPELM	94.83 ± 1.19	96.67	0.25 ± 4*10(-3)	85.45	19
	CSPELMF	94.11 ± 1.01	96.67	12.88 ± 10.41	257.66	19
	CSPELMR	94.70 ± 1.06	96.67	6.74 ± 4.49	107.82	19
	LM	**95.64 ± 2.99**	**98.57**	2.43 ± 0.60	73.01	19
H	CSPELM	**99.86 ± 0.09**	**100.00**	1.18 ± 0.01	641.59	52
	CSPELMF	99.67 ± 0.17	**100.00**	29.77 ± 15.37	2083.96	52
	CSPELMR	99.79 ± 0.12	**100.00**	62.82 ± 42.09	1005.14	52
	LM	95.64 ± 0.83	97.16	205.58 ± 29.65	10278.75	52

user can utilize this as a method for escaping local minimums if the traditional ways are ineffective.

Finally, we also note that from our representative related works search, the results produced by these papers always outperform the standard ELM. Though, they do not appear to achieve as impressive of results as our methodologies do. For example, in [9] and [11], while their methodologies are quite interesting and provide improved results, the total improvement is lower than ours. In a future study, we will compare and contrast our accuracy optimized methodologies with these related ones to examine if further improvements based on any combination of these methodologies is possible. Specifically, from [11], a combination between the ε constraint and their vectors of weights of between class samples could potentially provide a significant improvement.

VI. Conclusions & Future Work

In conclusion, this paper presents two accuracy optimization techniques to the Constrained State-Preserved Extreme Learning Machine (CSPELM) in the form of a Forest optimization (CSPELMF) and ε Range-finder (CSPELMR). Furthermore, we showed hyper-parameter settings to maximize the accuracy of the CSPELM as well while sacrificing training time. The CSPELMF was created under the hypothesis that multiple roots consisting of State-Preserved Extreme Learning Machine (SPELM) trained with a varying number of iterations could provide different starting positions in the n-dimensional gradient space and thus have an increased ability of finding better results. As for the CSPELMR, we hypothesized that by gradually increasing the size of the ε constraint that it could act as a method to escape local minimums. We compared the CSPELM, CSPELMF, CSPELMR, and Levenberg-Marquardt (LM) backpropagation with a total of 13 datasets. Our results showed that our methods outperformed the LM backpropagation in a majority of datasets and that the CSPELMF and CSPELMR matched or outperformed the CSPELM in all classification datasets.

The future work, as briefly mentioned in Section V, is twofold. First, we would like to first compare and contrast all new algorithms based off of the original Extreme Learning Machine (ELM) and examine their raw performance improvements as well as the ability to be used in a non-problem specific manner. Then, we believe that a combination of our ε constraint along with the work produced from [11] has the potential of producing even further improvements.

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.

TABLE IV: The regression results for the CSPELM, CSPELMF, CSPELMR, and LM Backpropagation. The bold values represent the best results for each dataset.

Dataset	Algorithm	RMSE	Final/Best RMSE	Training Time (s)	Total Time (s)	#Hidden Nodes
I	CSPELM	**0.14 - 4*10(-3)**	**0.13**	2.67 - 0.02	199.59	5
	CSPELMF	0.16 - 0.01	0.16	74.65 - 60.17	1492.91	5
	CSPELMR	0.15 - 4*10(-3)	0.15	59.19 - 44.40	295.95	5
	LM	0.14 - 0.02	**0.13**	2.79 - 0.71	13.94	5
J	CSPELM	**12.29 - 0.56**	**11.55**	0.03 - 2*10(-3)	3.36	6
	CSPELMF	18.05 - 2.84	17.27	1.03 - 0.69	41.14	6
	CSPELMR	18.05 - 4.45	13.16	0.96 - 0.63	12.53	6
	LM	17.25 - 2.70	12.33	0.19 - 0.31	4.91	6
K	CSPELM	10.68 - 0.29	10.33	0.32 - 0.01	47.52	11
	CSPELMF	11.99 - 0.49	11.72	9.34 - 6.86	280.28	11
	CSPELMR	12.20 - 0.69	10.96	9.18 - 6.06	119.35	11
	LM	**10.00 - 0.30**	**9.79**	0.65 - 0.66	3.24	11
L	CSPELM	**5.44 - 0.78**	**4.96**	0.18 - 0.01	17.37	3
	CSPELMF	17.49 - 8.89	11.39	5.62 - 3.78	224.96	3
	CSPELMR	14.96 - 8.91	6.28	3.38 - 2.54	37.16	3
	LM	15.65 - 1.76	14.09	0.27 - 0.35	5.29	3
M	CSPELM	**3.81 - 0.73**	**2.77**	0.04 - 2*10(-3)	3.41	4
	CSPELMF	10.40 - 3.64	11.12	1.01 - 0.81	20.15	4
	CSPELMR	7.74 - 2.16	4.78	1.03 - 0.66	10.27	4
	LM	6.48 - 1.05	5.62	0.28 - 0.37	4.18	4

[3] Q. Weng, Z. Mao, J. Lin, and X. Liao, "Land-use scene classification based on a cnn using a constrained extreme learning machine," *International journal of remote sensing*, vol. 39, no. 19, pp. 6281–6299, 2018.

[4] D. Serre, "Matrices theory and applications second edition," *GRADUATE TEXTS IN MATHEMATICS*, vol. 1, no. 216, pp. ALL–ALL, 2010.

[5] G. Goodman, C. Shimizu, and I. P. Ktistakis, "Constrained state-preserved extreme learning machine," in *2019 IEEE 31st International Conference on Tools with Artificial Intelligence (ICTAI)*. IEEE, 2019, pp. 752–759.

[6] 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.

[7] D. Zhang, X. Peng, K. Pan, and Y. Liu, "A novel wind speed forecasting based on hybrid decomposition and online sequential outlier robust extreme learning machine," *Energy conversion and management*, vol. 180, pp. 338–357, 2019.

[8] S. Beyhan and K. Kavaklioglu, "Comprehensive modeling of u-tube steam generators using extreme learning machines," *IEEE Transactions on Nuclear Science*, vol. 62, no. 5, pp. 2245–2254, 2015.

[9] D. T. Bui, P.-T. T. Ngo, T. D. Pham, A. Jaafari, N. Q. Minh, P. V. Hoa, and P. Samui, "A novel hybrid approach based on a swarm intelligence optimized extreme learning machine for flash flood susceptibility mapping," *Catena*, vol. 179, pp. 184–196, 2019.

[10] Z. M. Yaseen, S. O. Sulaiman, R. C. Deo, and K.-W. Chau, "An enhanced extreme learning machine model for river flow forecasting: State-of-the-art, practical applications in water resource engineering area and future research direction," *Journal of Hydrology*, vol. 569, pp. 387–408, 2019.

[11] 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.

[12] M. Alamaniotis, "Synergism of deep neural network and elm for smart very-short-term load forecasting," in *2019 IEEE PES Innovative Smart Grid Technologies Europe (ISGT-Europe)*. IEEE, pp. 1–5.

[13] 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

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

[15] 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

[16] W. H. Wolberg, W. N. 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)

[17] 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

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

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

[20] 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

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

[22] 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.

[23] 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.

[24] 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.

[25] 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

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