Computer Science Journal of Moldova, vol.**29**, no.3(87), 2021

To what extent is tuned neural network pruning beneficial in software effort estimation?

Muhammed Maruf Öztürk

Abstract

Software effort estimation (SEE) is of great importance for planning the budgets of future projects. The models of SEE are developed depending on the enhancements of hardware technology. However, developing such models based on neural networks remarkably increases the burden of computation. Neural network pruning may provide a suitable alternative to alleviate that burden. By detecting the elements making insignificant contributions to the output of a trained neural network, it is thus possible to obtain a reliable model. Otherwise, valuable information extracted from a trained neural network may be lost in pruning. In this work, the effects of pruning multi-layer perceptron (MLP) are investigated on SEE. To experimentally evaluate those effects, eight SEE data sets are employed. To find the optimal configuration of MLP, four optimization methods are utilized along with two pruning techniques. The results show that each optimization method has a distinctive threshold to suspend pruning. The model established to reach a low error of SEE, the number of features having low standard deviations should be greater than that of the features having high standard deviations. If a tuning process is applied to the hyperparameters of the pruning algorithm, the genetic algorithm is recommended to obtain high accuracy in the classification. This work provides a guideline for researchers to understand the effectiveness of neural network pruning in SEE.

Keywords: Effort estimation, hyperparameter optimization, neural network pruning

MSC: 62M45,68T05

1 Introduction

1.1 Motivation

Estimating an effort required to complete a product of a software system is called software effort estimation (SEE) which helps plan the budgets of future versions of the software. To that end, various methods developed so far have required intensive efforts. Conducting SEE results in a higher quality of software project, improved task distribution among team members, and preventing extreme optimal allocation [1].

Some mathematical models such as COCOMO [2] were predominant in SEE methods. However, conducting SEE with those methods is outmoded and insufficient. Analogy-based techniques were developed to increase the success of SEE [3]. The methods utilizing fuzzy logic were able to remove uncertainty and make knowledge integration [4]. Neuro-fuzzy models [5] remarkably increased the success of SEE but they take much time, as compared to the primitive models. Likewise, neural network models [6] showed a remarkable superiority over simple linear regression. Pioneering studies were developed based on mathematical models establishing classic regression analysis. However, software systems have a lot of versions that require more complex evaluations to predict future efforts. To address this problem, deep learning methods were proposed in recent years [7]. Deep learning methods could be a possible way to cope with vast data sets. But they dramatically increase the computational burden since ordinary computer stores and processes data at different places.

Predictive models can be leveraged by adopting deep transformation. Deep learning methods produced a remarkable success (80%) [8] in some phases of software including development [9], maintenance, testing [8], and management [10]. Deep models bring a big workload due to the computer architectures regardless of the computational strength. To solve that problem, some neurons of deep models can be pruned in which they make insignificant contributions to the output. Hence, the computational burden that originated from deep models might be alleviated via employing compressed deep models.

The two points constitute the main motivation of this paper as below:

1. Despite the fact that machine learning models increased the success of SEE, it remained insufficient compared to deep learning. Further, to exploit the optimal capacity of a machine learning model, some tuning processes are needed to achieve the best configuration.

2. Employing only the tuning process to the deep models can not alleviate the computational burden. Therefore, a tuning process should be performed by applying a certain rate of compression to the deep models.

Here we assert that using pruned neural networks helps alleviate the computational burden of SEE established with deep models. This paper explores whether pruned neural networks improve the success of effort estimation. Last, we examine the effects of some derivative-free optimization techniques including Bayes, Genetic algorithm, Nelder-mead, and Random search on finding optimal parameters of pruning.

1.2 Contribution

This work claims the following contributions below:
1. To our knowledge, this work is the first to investigate the effects of pruned neural networks on SEE.
2. Four derivative-free methods are employed to find the optimal configuration of pruning.
3. An in-depth analysis is conducted to determine what is the compression threshold for SEE.
4. The results reveal what sort of tuning strategy SEE tends to work together harmoniously.
The remainder of the paper is organized as follows: Section II gives a summary of related works. The proposed method is described in Section III. Experimental steps are given in Section IV. The results are presented in Section V and Section VI concludes the results.

2 Related work

2.1 Neural network pruning

One of the oldest and common methods for pruning a neural network is Optimal Brain Damage (OBD) [11]. That method computes the second

derivative of each Hessian matrix for each parameter to be optimized, thereby sorting the salience levels of the parameters. These processes are sustained until a reasonable result is obtained.

Optimal Brain Surgeon (OBS), another popular method, takes advantage of Lagrangian computation of the invert Hessian matrix to minimize the error of weights [12]. This operation is suspended once the optimal error is achieved. On the other hand, magnitude-based pruning compares the size of node weights to prune either related node or edge [13]. The main threat posed by it is the risk of disregarding the nodes creating remarkable effects on the output while pruning the nodes having small sizes. This is the major disadvantage of magnitude-based pruning.

The generic objective of OBD and OBS can be described as follows: 1. Improve the speed of the neural network, 2. Obtain a simple neural network that can be denoted with a smaller number of connections than that of the former neural network, 3. Provide various criteria to the practitioners in order to stop pruning.

Initially performed by utilizing one layer [14], neural network pruning has changed to multi-layer pruning over time [15]. However, doing that procedure manually leads to a heavy cost and spending much time. To address this problem, a method called MLPRUNE [16] was developed to set the compression rate automatically depending on the data set.

Zhao et al. [17] proposed a new pruning method based on the principles of channel salience of neural networks by using Bayes probability. Their method was tried on three convolutional networks specifically for image classification. Although high compression networks can not produce high accuracy, the burden on RAM is alleviated in that way, according to the results.

There is abundant scientific literature on how does pruning affect training loss. For instance, Laurent et al. [18] argued that the pruning process can lead to a high training loss that can be removed with the help of a fine-tuning process conducted with a few iterations. Likewise, Ding et al. [19] also detected that accuracy declines with a certain rate of pruning. However, employing a global compression rate yields

relatively high classification performance compared to the weighting-based pruning.

Molchanov et al. [20] predicted the importance of weights of the network by using Taylor expansion. Hence, they pruned less important neurons with models solving memory constraints. In [21], it was stressed that layer-wise pruning does not remarkably affect the prediction performance of a neural network. Further, a great number of training iterations, in such a way, are not required.

At the very least, magnitude-based pruning should be performed on one layer. On the contrary, Park et al. [14] asserted that magnitude-based pruning is much more compatible with multi-layers. In this context, the main advantage of their method namely lookahead pruning is that it requires hyperparameter optimization. Besides, lookahead pruning also considers the effects of edges nearby the pruned nodes.

2.2 Effort estimation

SEE was being conducted in the light of expert opinions while machine learning algorithms were not being thought. Subsequently, some mathematical models such as COCOMO [22] were applied to the SEE data sets. Genetic programming was utilized in the early works related to the SEE [23]. Even though genetic programming, which emerged as an alternative for linear models, yielded promising results on some data sets, it needs sophisticated experimental designs that are time-consuming and bring huge computation.

The methods combining fuzzy-logic and analogy outperformed linear computations [5]. However, the scale of data sets limits the comprehensiveness of the experiment. Fuzzy models should be employed by applying an optimization procedure according to their opinion. Fuzzy models were also combined with artificial neural networks. In such a study [24] neuro-fuzzy was found to be superior to the COCOMO.

Kocaguneli et al. [25] depicted that employing only one learning model on SEE data sets is not sufficient and thus they demonstrated the advantages of ensemble methods over single models. This is because the method chosen for SEE changes depending on the type of

software project [26]. Therefore, the following studies focused on data preprocessing [27] and investigating configurable parameters for SEE models [28].

3 Background

An MLP consists of three parts: the input layer, hidden layer, and output layer. MLP, which has one hidden layer, is a basic type of feed-forward neural networks. It is employed when there is little information about the experimental problem. Let R be the search space, where d is the dimension of inputs. H represents the hidden layer and $H = 1$ for an MLP. If n denotes the number of total nodes, n_i is the number of nodes in i^{th} layer, where $n > n_i$. For an input vector x, ϕ_j^i represents the output produced in the dense layers. Given k and j as the independent units of MLP, $w_{k,j}^i$ denotes the weight of edges between these units. Let b_j^i be the bias in which j is the unit of i^{th} layer. The output is calculated as in Equation 1, where σ denotes an activation function.

$$f(x) = \sigma(\sum_{k=1}^{n_H} w_{k,j}^H \phi_k^H + b_j^{H+1}). \tag{1}$$

In essence, a pruning function aims to remove some connections that strongly depend on the type of pruning method and the configuration of pruning function. If we have l layers and c connections, the distribution of p may change on each iteration of the pruning method. Concretely, pr shows the number of pruned connections. Subsequently, $c - pr$ is calculated to obtain the number of remaining connections in which n_p denotes the number of remaining nodes. Thereafter, the output of the neural network is calculated as:

$$f(x) = \sigma(\sum_{k=1}^{n_{pH}} w_{k,j}^H \phi_k^H + b_j^{H+1}). \tag{2}$$

A pruning method is able to change a neural network as shown in

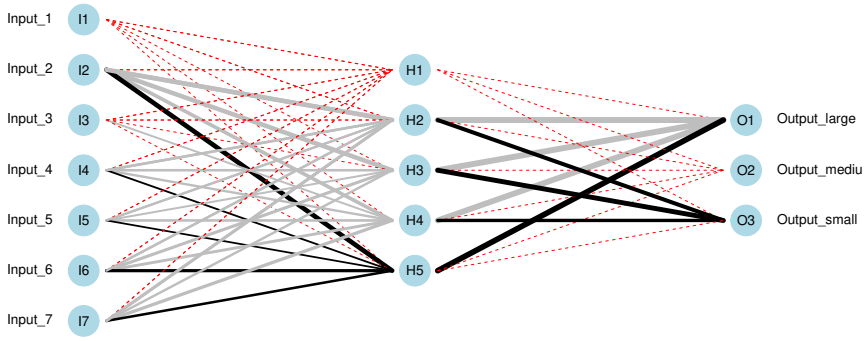


Figure 1. A pruned neural network. Positive and negative connections are represented with black and grey lines, respectively. Red dashed lines denote pruned connections

Figure 1. Here I denotes an input and H represents a hidden node. For Figure 1, l=3, c=50, pr=22, and n_p=12. There are seven inputs and three outputs. Pruning algorithms have many configurable hyper-parameters that change the structure of pruned network.

Let n_I be the number of inputs and n_O be the number of outputs. The number of black, gray, and red edges are represented with e_b, e_g, and e_r, respectively. The total weight related to the edges can be formulated with:

$$T_w = \sum_{i=1}^{n} w_i.e, \tag{3}$$

where T_w denotes the total weight for the number of edges. To explicitly define Equation 3, we have:

$$T_w = \sum_{i=1}^{n_1} w_i.e_b + \sum_{i=1}^{n_2} w_i.e_g + \sum_{i=1}^{n_3} w_i.e_r, \tag{4}$$

where n_1 is the number of edges associated with the nodes which greatly contribute to the output of the neural network; n_2 represents the number of gray edges having negative effects on deciding the outputs of neural network; n_3 is the number of pruned edges; and the number of edges from (3) is $n = n_3 + n_2 + n_1$.

Equation 4 is updated with $e_r \to 0$ due to the pruning. Hence, we define ultimate form of T_w:

$$T_w = \sum_{i=1}^{n_1} w_i.e_b + \sum_{i=1}^{n_2} w_i.e_g. \tag{5}$$

The cost of T_w depends on the number of layers which are involved in the pruning. For each layer, we obtain the output:

$$f_x^{'} = \prod_{i=1}^{z} T_{wt}, \tag{6}$$

where the calculation of cost is performed for the total layer z.

4 Experimental design

4.1 Hyperparameter optimization

Four optimization methods including Bayes, genetic algorithm, Nelder-mead, and random search are involved in this paper.

Genetic algorithm. The main objective of the genetic algorithm is to reach global optimum by evaluating various local optimums. It thus needs a lot of iterations. To diversify the outcomes, the genetic algorithm uses mutation that is utilized when there is a problem not compatible with differentiable or continuous objective functions. The use of a genetic algorithm, or its combination with other techniques, in SEE is not new [24], [29]. However, to our knowledge, there is not any comprehensive study evaluating the effects of tuning genetic algorithm for SEE.

Bayesian optimization. Bayes establishes a probabilistic model by utilizing a past event. There are abundant studies on how the Bayesian network can be used to perform SEE [30]. However, most of them did not consider tuning the hyperparameter of a Bayesian network. The major advantage of employing a Bayesian model over other competitors is its ability to cope with missing data.

Nelder-Mead optimization was presented in 1965. Nelder-Mead generates a triangular shape to conduct optimization. That shape is changed in every iteration to reach an optimal solution. Nelder-mead was applied to the problems associated with energy and structural engineering [31].

Random search seeks parameters in different places depending on the search space. Random search is useful to reduce optimization time because it is flexible with the configurable iteration threshold. Random search is compatible with a parallel tuning process. Because it does not need communication between threads. Random search-tuned SEE models yield results as successful as those of grid search, according to a recent study [32] to date shows.

4.2 Pruning methods

We utilized two pruning methods including OBD and OBS. These methods can be considered as the most known and oldest pruning methods in the literature. Although applying them to any classification problem dates back to the '90s, much of the research relevant to hyperparameter optimization has not focused on neural network pruning. Our work intends to fill this information gap.

OBD is constructed via a gradient G and the Hessian matrix H. They are calculated as follows:

$$G_i = \frac{\partial F}{\partial c_i}, h_{ij} = \frac{\partial^2 F}{\partial c_i \partial c_j}, \tag{7}$$

where c is the component of perturbation and G_i denotes a component of the gradient. h is an element of H and F represents the objective function. The objective function F is minimized as follows:

$$\phi F = \frac{1}{2} \sum_i h_{ii}.\phi.c_i^2, \tag{8}$$

where ϕ denotes the change in the objective function.

OBS aims to reach local minimum error of a trained neural network. H denotes the Hessian matrix of the inputs and it is calculated as

follows:

$$H = \partial_2 e / \partial w_2, \tag{9}$$

where the second-order derivatives are utilized. One arbitrary weight named w_q is set to zero in order to minimize the increase detected in the error. A set of nested functions are solved as in Equation 10:

$$F_q(F_{fw}(1/2.fw_T.H.fw)), \tag{10}$$

where f denotes a change either it is employed on the weight w or its transposed form w_T. F is the objective function that obeys the constraint:

$$u_q^T.fw + w_q = 0, \tag{11}$$

where u_q represents the unit vector that is associated with a w_q. However, the optimal change of the weight is:

$$fw = \frac{w_q}{[H^{-1}]_{qq}}.u_q. \tag{12}$$

4.3 Experimental data sets

In the experiment, eight data sets are chosen to expose the SEE process. Table 1 gives a summary of experimental data sets. The column namely "Instance" shows the number of instances. They are sampled and the final values are given in "Sampled". It is worth noting that the range of sampled instances is large. Hence, a large sample size contributes to greater generalizability of the results. SMOTE [33] is employed to oversample the experimental data sets.

SMOTE is a popular sampling algorithm proposed by Chawla et al. [33], and its accuracy was tested on the increasing number of training instances. Given that SMOTE has been combined with the right undersampling algorithm, it performs well. Blagus and Lusa [34] investigated SMOTE on large-scale data sets to prove its effectiveness. From the finding, they deduced that SMOTE is suitable for small-scale data and it requires specific preprocessing when the scale of the data sets is large.

Table 1. The details of experimental data sets

Name	Feature	Instance	Min	Max	Mean	Std.Dev.	Sampled
albrecht	7	26	0.5	105	20	27	416
china	18	499	26	54620	3921	6480	499
coc-81	6	74	128	41248	4375	8774	420
desharnais	11	81	546	23940	5046	4418	1641
kitchenham	9	145	219	113930	3113	9598	2111
maxwell	26	62	583	63694	8223	10499	1070
miyazaki94	7	48	6	1586	87	229	524
nasa93-dem	26	93	8	8211	624	1136	1225

Each effort data set has a column indicating effort as numeric. However, we herein intend to classify SEE data sets. To this end, effort values are labeled as "small", "medium", and "large" depending on the magnitude of a related effort. Algorithm 1 is devised to converting numeric effort values to factor values. Step 4 calculates the number of features for the given data set. $rowCount$ denotes the number of instances. In Steps 7-9, the sum of the effort values is calculated. $mean$ takes rounded mean effort value in Step 10. $mediumThreshold$ is defined to divide effort values into three parts. An empty list namely $Effort$ is created in Step 12. Factor assignments are done in Steps 13-21 by performing certain comparisons. Step 22 removes the effort feature including numeric values. Factor values are combined with the rest of the data set in Step 23. Lastly, the final data set is returned in Step 24.

4.4 Performance parameter

In the experiment, accuracy was evaluated by changing the compression rate for each data set. Equation 13 describes the details of accuracy that consists of confusion matrix elements:

$$accuracy = (TP + TN)/(TP + TN + FP + FN). \tag{13}$$

Compression ratio is computed via the following formula:

$$compressionRatio = PE/TE, \tag{14}$$

Algorithm 1 Algorithm for converting effort feature to factor values

Input:$D(SEE\_data\_set)$
Output:$D_p(processed\_SEE\_data\_set)$
$columnCount \leftarrow length(D)$
$rowCount \leftarrow length(D[,1])$
$sum \leftarrow 0$
for `i=0 to rowCount` **do**
  `sum←sum+D[i,columnCount]`
end for
$mean \leftarrow round(sum/rowCount)$
$mediumThreshold \leftarrow round(mean/2)$
$Effort \leftarrow emptyList()$
for `i=0 to rowCount` **do**
  if $D[i, columnCount] < mediumThreshold$ **then**
    $Effort[i] \leftarrow$ "$small$"
    elseif$D[i, columnCount] > mean$"
    $Effort[i] \leftarrow$ "$large$"
  else
    $Effort[i] \leftarrow$ "$medium$"
  end if
end for
$columCount \leftarrow columnCount - 1$
$D_p \leftarrow Data.Frame(D[,1 : columnCount], Effort)$
return D_p

where PE and TE denote the number of pruned and total edges, respectively. To change the compression ratio, a varying learning rate is applied to the multi-layer perceptron employed for the classification. Further, the weighted sum of square error (WSSE) is used to evaluate the iterative training and test error of the model as follows:

$$WSSE = \sum_{i=1}^{n} w_i.(y_i - \overline{y}_i), \tag{15}$$

where n is the number of observations and w_i is the wight of related observation. y_i represents the observed value and $\overline{y}_i$ denotes the predicted value. The smaller the $WSSE$, the better the result of predictions. Here, Equation 15 helps to perform a validation assessment.

4.5 Implementation details

The experiment is twofold. First, the classification is done with MLP so that its hyperparameters are subjected to the tuning process. Second, the hyperparameters of pruning methods are involved in such a process. The description of hyperparameters and the search range of them are presented in Table 2. The column namely "method" shows the type of hyperparameters.

Each data set is divided into three parts: 70% is for training, 15% is for validation, and 15% is for testing. First, the training part is exploited to fit the model. The validation part is utilized to tune hyperparameters. The testing part is used to reach the final model. These processes are repeated 200 times by randomly generating those parts. Last, average results are recorded. The main steps of the experiment are given in Figure 2. Red dashed lines represent the transmission of data parts. In the testing phase, validated configurations of the hyperparameters are employed. The results are produced in the last step.

For Bayesian optimization, the setting is as follows: the number of initialization points is 2, the total number of iterations – 10, the type of acquisition function (acq) – GP upper confidence bound, kappa of acq – 2.576. That setting has been successfully employed in various Bayesian optimization studies [35]. The genetic algorithm uses the

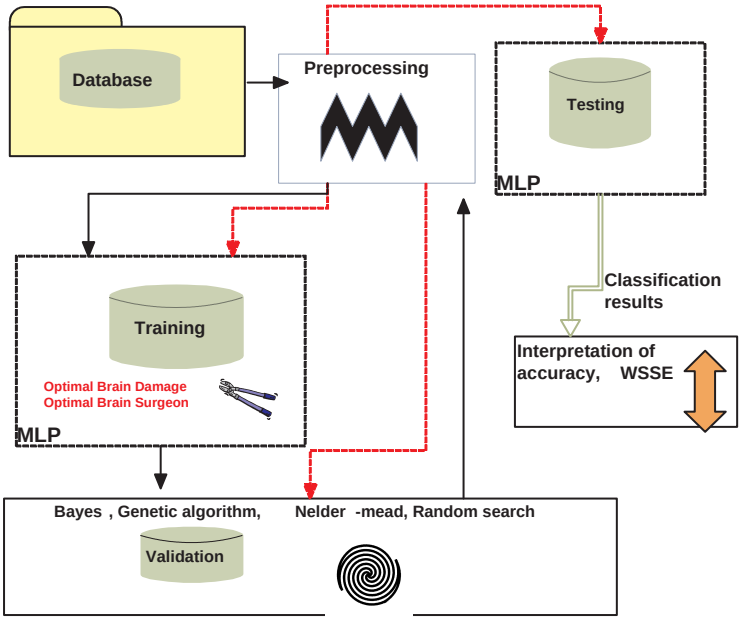


Figure 2. Overview of the experiment

Table 2. The details of hyperparameters and search space

Name	method	description	range	stepSize
learningrate	MLP	learning rate	0.1→0.9	0.01
maxit	MLP	maximum iteration	10→250	10
initFuncParams	MLP	the parameters for the initialization function	-0.3→0.8	0.1
max_pr_error_increase	pruning	maximum error increase for per iteration	1→10	1
pr_accepted_error	pruning	accepted error for per iteration	1→5	1
no_of_pr_retrain_cycles	pruning	number of retrain cyles	1→10	1
min_error_to_stop	pruning	stopping criteria	0.001→0.01	0.001

setting as follows: the probability of crossover used in chromosomes is 0.8, the mutation probability – 0.1, the ratio of elitism – 5%. The starting point of Nelder-mead is determined as follows: learningrate – 0.1, maxit – 40, initFuncParams – −0.3, max_pr_error_increase – 1, pr_accepted_error – 1, no_of_pr_retrain_cycles – 1, min_error_to_stop – 0.001.

5 Results

5.1 Do optimal configurations differ depending on the optimization method?

The values presented in Table 3 are generated by taking averages of OBD and OBS. Bayes and Nelder-mead need fewer iterations as compared to the others. On the other hand, the genetic algorithm requires the largest iterations as detected in the learning rate. The highest stopping error is of random search.

Table 4. Accuracy details of the pruning methods

optimization method	**pruning method**	**albrecht**	**china**	**coc-81**	**desharnais**	kitchenham	maxwell	miyazaki94	nasa93dem
Bayes	OBD	0.89	0.84	0.97	0.94	0.95	0.96	0.91	0.98
	OBS	0.90	0.84	0.97	0.94	0.93	0.97	0.88	0.97
Genetic algorithm	OBD	0.99	0.94	0.97	0.94	0.97	0.98	0.95	0.99
	OBS	0.98	0.95	0.96	0.92	0.94	0.99	0.95	0.99
Nelder-mead	OBD	0.95	0.96	0.97	0.89	0.97	0.98	0.92	0.99
	OBS	0.95	0.95	0.96	0.94	0.94	0.99	0.93	0.99
Random search	OBD	0.96	0.96	0.97	0.94	0.97	0.99	0.97	0.98
	OBS	0.98	0.95	0.97	0.91	0.97	0.98	0.97	0.99

5.2 Is hyperparameter tuning beneficial to reduce WSSE?

Figure 3 shows the WSSE results of the comparison algorithms. Iterative training and test errors are denoted with black line and red lines, respectively. It is worth noting that there is a reasonable model if those lines are very close and stable after a certain iteration. Otherwise, we need to change the model setting or to focus on data preprocessing. Random search and genetic algorithm yielded better curves than those of other methods. Further, we can conclude that optimization has a favorable effect on training vs. test lines.

The best observations of WSSE are related to the standard deviation of the features. If a set of instances includes features having

Table 3. Optimal configurations found by the optimization methods

Data set	Method	learningrate	maxit	initFuncParams	max_pr_error_increase	pr_accepted_error	no_of_pr_retrain_cycles	min_error_to_stop
albrecht	Bayes	0.1047	49	-0.2654	1.8486	1.8521	1.6604	0.0017
	Genetic algorithm	0.5099	74	-0.2086	1.5068	3.0949	2.2399	0.0019
	Nelder-mead	0.2932	50	-0.3	2.9927	1.8518	2.8407	0.0010
	Random search	0.7729	90	0.3488	5.6413	4.5727	8.0711	0.0050
china	Bayes	0.1035	47	-0.2965	1.4976	1.4359	1.4226	0.0016
	Genetic algorithm	0.4535	94	-0.2063	2.2675	1.9733	3.0627	0.0014
	Nelder-mead	0.48822	78	-0.2569	2.5175	1.6760	1.5370	0.0025
	Random search	0.3885	85	0.3825	9.2178	1.2036	9.3685	0.0044
coc-81	Bayes	0.1498	45	-0.2315	1.6952	1.5903	1.6565	0.0014
	Genetic algorithm	0.4789	66	0.3855	3.7164	2.8786	4.8613	0.0029
	Nelder-mead	0.2802	41	-0.2731	2.1764	1.0050	1.2196	0.0011
	Random search	0.4466	75	-0.1558	2.5110	1.0952	6.1939	0.0032
desharnais	Bayes	0.1190	46	-0.2183	1.7488	1.3093	1.2480	0.0015
	Genetic algorithm	0.70932	93	0.6871	6.5698	2.3201	8.5741	0.0041
	Nelder-mead	0.3734	53	-0.1837	1	1.0816	2.0626	0.0020
	Random search	0.2727	80	0.5097	6.7889	1.1378	1.3045	0.0024
kitchenham	Bayes	0.1209	45	-0.2137	1.8457	1.8014	1.9153	0.0013
	Genetic algorithm	0.4719	88	-0.1149	4.2376	3.2925	2.6307	0.0093
	Nelder-mead	0.3521	56	-0.1725	2	1.0562	2.0456	0.0021
	Random search	0.4178	90	0.2298	1.3869	4.1208	5.6294	0.0098
maxwell	Bayes	0.1483	48	-0.2898	1.9325	1.8115	1.1831	0.0015
	Genetic algorithm	0.5757	67	0.3382	7.4639	1.0806	3.5168	0.0038
	Nelder-mead	0.2444	58	-0.1611	1.4	1.0411	2.1122	0.0041
	Random search	0.4945	63	0.4408	5.2871	3.5066	7.5517	0.0046
miyazaki94	Bayes	0.1335	48	-0.2960	1.3546	1.6546	1.7836	0.0012
	Genetic algorithm	0.3161	79	0.5487	3.4386	4.8322	2.7963	0.0049
	Nelder-mead	0.3	40	-0.2999	1.4218	1	1.5625	0.0010
	Random search	0.7074	88	0.3005	5.8098	4.9289	9.6855	0.0026
nasa93-dem	Bayes	0.1255	43	-0.2136	1.9775	1.7683	1.0395	0.0017
	Genetic algorithm	0.4997	49	0.7323	7.1395	1.9984	1.8017	0.0059
	Nelder-mead	0.3079	41	-0.0899	2.9856	1.8318	2.7921	0.0010
	Random search	0.8385	81	0.2840	2.2025	4.8796	1.4265	0.0057

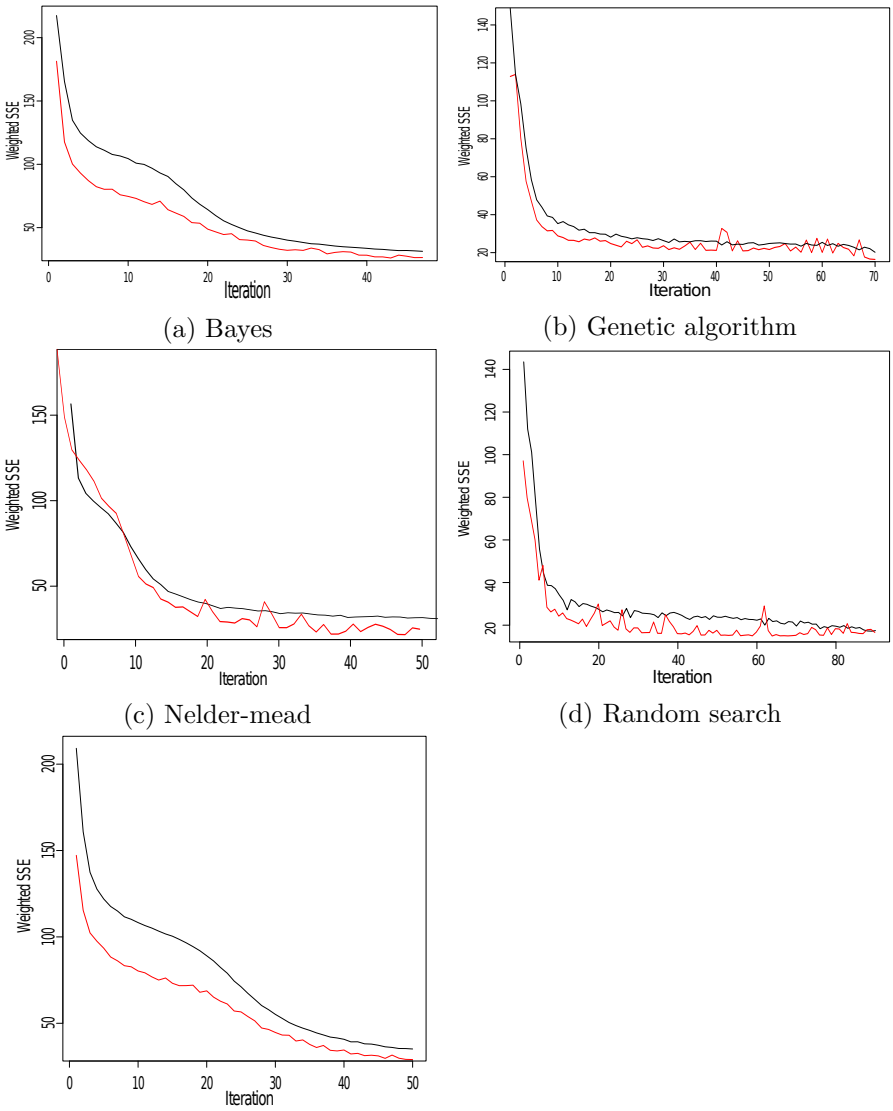
low standard deviations, the margin between training and test curves becomes narrow.

The accuracy changes across the data sets are given in Table 4. Note that the accuracy values are not dependent on the data set. Likewise, the type of pruning method employed on the MLP has little impact on accuracy. In summary, with respect to the compression rate, the genetic algorithm is more durable than its counterparts. Likewise, it was found to have the highest accuracy according to the overall performance presented in Figure 4f. Bayes showed a low level of resistance against compression that is not reasonable when optimizing pruned networks. Interestingly, MLP becomes more tolerant to compression if the pruning is performed without optimization when compared to Nelder-mead and Bayes as shown in Figure 4e.

5.3 Does the compression create the same effect on the optimization methods?

Figure 4 shows changing accuracies depending on the increasing compression rates. The results of Figure 4 were generated via the mean values of OBD and OBS. The best of Bayes given in Figure 4a is 93% accuracy with 3% compression rate. It does not respond to the compression rate above 18%. For that compression rate, the accuracy is 72%. Figure 4b is of the genetic algorithm that was able to achieve 90% accuracy with 1.7% of compression rate. The genetic algorithm does not calculate the accuracy of the compression rate above 58%. Nelder-mead can not perform optimization if the compression rate is under 39%. Its accuracy sharply reduces from 96% to 87% if the compression rate is augmented up to 20%. Such a decline is very high for the data sets that do not converge fast in practice. This case may have originated from a high number of features. Random search achieved the highest accuracy 92% with 2% compression. It does not respond to the compression rate above 32%.

In the experiment, the accuracy values were recorded with increasing iteration. Here we intend to observe how fast optimization methods reach a narrow and high accuracy range. Figure 5 presents accuracy values for 250 iterations. Bayes needs more iterations than the other



(a) Bayes

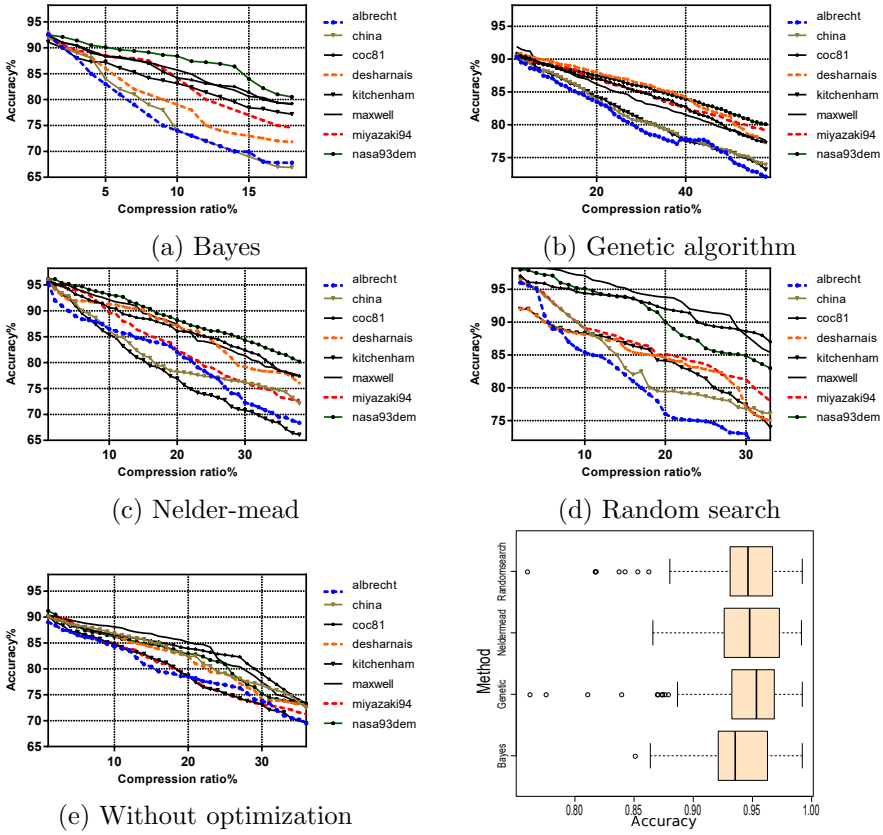
(b) Genetic algorithm

(c) Nelder-mead

(d) Random search

(e) Without optimization

Figure 3. The comparison of WSSE results including without optimization. The results are produced with 2000 randomly selected instances from the data sets. The pruning is performed via average setting of OBD and OBS



(a) Bayes

(b) Genetic algorithm

(c) Nelder-mead

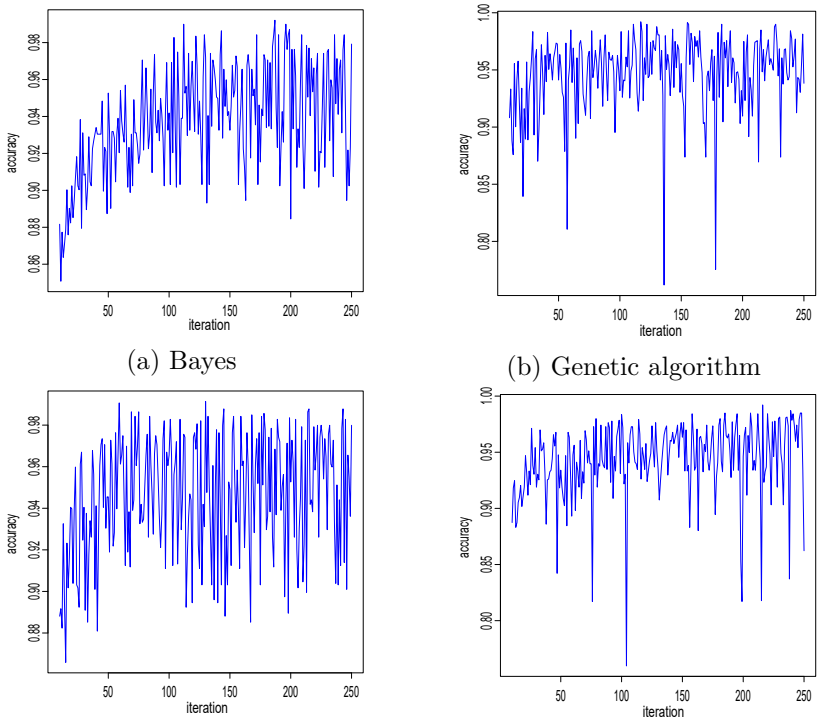
(d) Random search

(e) Without optimization

(f) Box-plot of accuracy values

Figure 4. Accuracy comparison of the algorithms with varying compression rates

methods to reach high and stable accuracy. The genetic algorithm was found to produce the best accuracy as given in Figure 4f. It has shown that it is no coincidence as demonstrated in Figure 5b.



(a) Bayes

(b) Genetic algorithm

(c) Nelder-mead

(d) Random search

Figure 5. Iteration and accuracy curves that show the details of accuracy changes about the comparison algorithms

6 Discussion

Some of the data sets have an insufficient number of instances in the experiment. To perform oversampling, SMOTE was used. Instead, an upgraded version of it such as LR-SMOTE [4] could change the tendency of results to reduce the fluctuation. To generalize the findings,

Table 5. Kruskal-Wallis test results of the optimization methods

P value	Significance	Groups	Kruskal-Wallis statistic	Number of values
0.0003	P<0.05	4	18.75	964

an MLP is utilized to create the model. Adding some deep learning techniques including stacked autoencoder and deep belief network may deepen our understanding of the effects of pruning on deep models. There is a major research gap we plan to investigate in future works that optimization methods also have various hyperparameters as machine learning algorithms have. For instance, Bayes has acq, init_points, n_iter, kappa, and eps. They need to be correctly tuned or initialized to obtain reliable outputs. Addressing that research gap could provide new insight into neural network pruning. Further, adding other techniques such as Magnitude-based to the pruning methods is expected to shift research direction in this field. It is worthwhile to note that conducting a multi-label classification on the SEE data sets limits the scope of the experiment. Performing a twofold experiment including both classification and regression may extend that scope. We further checked the difference in accuracy results among the comparison methods. To that end, a non-parametric test called Kruskal-Wallis is executed in no-pairing mode. According to the details of Table 5, optimization methods employed in the experiment are significantly different ($P<0.05$) with respect to the accuracy.

7 Conclusion

In the era of complex computing, SEE mainly relies on expert knowledge and basic machine learning methods employed with default configurations which remained insufficient especially for deep learning techniques. Because they create an enormous computational burden if there are large-scale data sets. In this work, we investigate in what ways that burden can be alleviated. To this end, an MLP is devised to perform SEE. Further, four optimization methods were employed to tune MLP and two pruning methods. The experiment realized with eight data sets shows that the genetic algorithm produces slightly better accuracy

than its counterparts when the compression rate is high. Without optimization, a pruned MLP produces higher performance as compared to Bayes and Random search. As a result, a pruned neural network may not be compatible with every optimization method. Further, the data sets having high standard deviations pose a crucial threat to the performance of SEE. Last, the type of pruning methods has a negligible effect on the accuracy of SEE.

References

[1] M. Usman, K. Petersen, J. Börstler, and P. Santos Neto, "Developing and using checklists to improve software effort estimation: A multi-case study," *Journal of Systems and Software*, vol. 146, pp. 286–309, dec 2018. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0164121218302073

[2] B. Boehm, B. Clark, E. Horowitz, C. Westland, R. Madachy, and R. Selby, "Cost models for future software life cycle processes: Cocomo 2.0," *Annals of software engineering*, vol. 1, no. 1, pp. 57–94, 1995.

[3] M. Shepperd, C. Schofield, and B. Kitchenham, "Effort estimation using analogy," in *Proceedings of IEEE 18th International Conference on Software Engineering*. IEEE Comput. Soc. Press, pp. 170–178. [Online]. Available: http://ieeexplore.ieee.org/document/493413/

[4] X. Liang, A. Jiang, T. Li, Y. Xue, and G. Wang, "Lr-smote–an improved unbalanced data set oversampling based on k-means and svm," *Knowledge-Based Systems*, p. 105845, 2020.

[5] C. L. Martin, J. L. Pasquier, C. M. Yanez, and A. Tornes, "Software development effort estimation using fuzzy logic: a case study," in *Sixth Mexican International Conference on Computer Science (ENC'05)*. IEEE, 2005, pp. 113–120.

[6] A. B. Nassif, M. Azzeh, L. F. Capretz, and D. Ho, "Neural network models for software development effort estimation:

a comparative study," *Neural Computing and Applications*, vol. 27, no. 8, pp. 2369–2381, nov 2016. [Online]. Available: http://link.springer.com/10.1007/s00521-015-2127-1

[7] P. Suresh Kumar, H. Behera, A. K. K, J. Nayak, and B. Naik, "Advancement from neural networks to deep learning in software effort estimation: Perspective of two decades," *Computer Science Review*, vol. 38, p. 100288, nov 2020. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S1574013720303889

[8] X. Li, H. Jiang, Z. Ren, G. Li, and J. Zhang, "Deep learning in software engineering," *arXiv preprint arXiv:1805.04825*, 2018.

[9] K. Madala, D. Gaither, R. Nielsen, and H. Do, "Automated identification of component state transition model elements from requirements," in *2017 IEEE 25th International Requirements Engineering Conference Workshops (REW)*. IEEE, 2017, pp. 386–392.

[10] V. J. Hellendoorn and P. Devanbu, "Are deep neural networks the best choice for modeling source code?" in *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, 2017, pp. 763–773.

[11] Y. LeCun, J. Denker, and S. Solla, "Optimal brain damage," *Advances in neural information processing systems*, vol. 2, pp. 598–605, 1989.

[12] B. Hassibi and D. Stork, "Second order derivatives for network pruning: Optimal brain surgeon," *Advances in neural information processing systems*, vol. 5, pp. 164–171, 1992.

[13] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," *Advances in neural information processing systems*, vol. 28, pp. 1135–1143, 2015.

[14] S. Park, J. Lee, S. Mo, and J. Shin, "Lookahead: A Far-Sighted Alternative of Magnitude-based Pruning," feb 2020. [Online]. Available: http://arxiv.org/abs/2002.04809

[15] P. Singh, R. Manikandan, N. Matiyali, and V. P. Namboodiri, "Multi-Layer Pruning Framework for Compressing Single Shot MultiBox Detector," in *2019 IEEE Winter Conference on Applications of Computer Vision (WACV)*. IEEE, jan 2019, pp. 1318–1327. [Online]. Available: https://ieeexplore.ieee.org/document/8658776/

[16] R. W. Zeng and Urtasun, "MLPrune: Multi-Layer Pruning for Automated Neural Network Compression," in *ICLR 2019*, 2019, pp. 1–12.

[17] C. Zhao, B. Ni, J. Zhang, Q. Zhao, W. Zhang, and Q. Tian, "Variational Convolutional Neural Network Pruning," in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, jun 2019, pp. 2775–2784. [Online]. Available: https://ieeexplore.ieee.org/document/8954234/

[18] C. Laurent, C. Ballas, T. George, N. Ballas, and P. Vincent, "Revisiting loss modelling for unstructured pruning," *arXiv preprint arXiv:2006.12279*, 2020.

[19] X. Ding, X. Zhou, Y. Guo, J. Han, J. Liu *et al.*, "Global sparse momentum sgd for pruning very deep neural networks," in *Advances in Neural Information Processing Systems*, 2019, pp. 6382–6394.

[20] P. Molchanov, A. Mallya, S. Tyree, I. Frosio, and J. Kautz, "Importance estimation for neural network pruning," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 11 264–11 272.

[21] X. Dong, S. Chen, and S. Pan, "Learning to prune deep neural networks via layer-wise optimal brain surgeon," in *Advances in Neural Information Processing Systems*, 2017, pp. 4857–4867.

[22] R. Smith, B. Dixon, A. Parrish, and J. Hale, "Enhancing the Cocomo estimation models," *IEEE Software*, vol. 17, no. 6, pp. 45–49, 2000. [Online]. Available: http://ieeexplore.ieee.org/document/895167/

[23] Y. Shan, R. McKay, C. Lokan, and D. Essam, "Software project effort estimation using genetic programming," in *IEEE 2002 International Conference on Communications, Circuits and Systems and West Sino Expositions*, vol. 2. IEEE, pp. 1108–1112. [Online]. Available: http://ieeexplore.ieee.org/document/1178979/

[24] X. Huang, D. Ho, J. Ren, and L. F. Capretz, "A soft computing framework for software effort estimation," *Soft Computing*, vol. 10, no. 2, pp. 170–177, 2006.

[25] E. Kocaguneli, T. Menzies, and J. W. Keung, "On the value of ensemble effort estimation," *IEEE Transactions on Software Engineering*, vol. 38, no. 6, pp. 1403–1416, 2011.

[26] S. K. Palaniswamy and R. Venkatesan, "Hyperparameters tuning of ensemble model for software effort estimation," *Journal of Ambient Intelligence and Humanized Computing*, sep 2020. [Online]. Available: http://link.springer.com/10.1007/s12652-020-02277-4

[27] J. Huang, Y.-F. Li, and M. Xie, "An empirical analysis of data preprocessing for machine learning-based software cost estimation," *Information and software Technology*, vol. 67, pp. 108–127, 2015.

[28] L. Song, L. L. Minku, and X. Yao, "The impact of parameter tuning on software effort estimation using learning machines," in *Proceedings of the 9th International Conference on Predictive Models in Software Engineering - PROMISE '13*. New York, New York, USA: ACM Press, 2013, pp. 1–10. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2499393.2499394

[29] S.-J. Huang, N.-H. Chiu, and L.-W. Chen, "Integration of the grey relational analysis with genetic algorithm for software effort estimation," *European Journal of Operational Research*, vol. 188, no. 3, pp. 898–909, 2008.

[30] F. Zare, H. Khademi Zare, and M. S. Fallahnezhad, "Software effort estimation based on the op-

timal Bayesian belief network," *Applied Soft Computing*, vol. 49, pp. 968–980, dec 2016. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S1568494616303969

[31] S. Xu, Y. Wang, and Z. Wang, "Parameter estimation of proton exchange membrane fuel cells using eagle strategy based on JAYA algorithm and Nelder-Mead simplex method," *Energy*, vol. 173, pp. 457–467, apr 2019. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0360544219303056

[32] L. Villalobos-Arias, C. Quesada-López, J. Guevara-Coto, A. Martínez, and M. Jenkins, "Evaluating hyper-parameter tuning using random search in support vector machines for software effort estimation," in *Proceedings of the 16th ACM International Conference on Predictive Models and Data Analytics in Software Engineering*. New York, NY, USA: ACM, nov 2020, pp. 31–40. [Online]. Available: https://dl.acm.org/doi/10.1145/3416508.3417121

[33] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "Smote: synthetic minority over-sampling technique," *Journal of artificial intelligence research*, vol. 16, pp. 321–357, 2002.

[34] R. Blagus and L. Lusa, "Evaluation of smote for high-dimensional class-imbalanced microarray data," in *Proceedings of the 2012 11th International Conference on Machine Learning and Applications-Volume 02*, 2012, pp. 89–94.

[35] R. Astudillo and P. I. Frazier, "Bayesian optimization of composite functions: Supplementary material," 2019.

M. Maruf Öztürk

Department of Computer Engineering
Faculty of Engineering
Isparta, TURKEY
Phone: +90 246 211 15 63
E–mail: muhammedozturk@sdu.edu.tr

Received January 17, 2021
Accepted September 23, 2021