Systems & Control Transactions

Research Article - Peer Reviewed Conference Proceeding
ESCAPE 35 - European Symposium on Computer Aided Process Engineering
Ghent, Belgium. 6-9 July 2025
Jan F.M. Van Impe, Grégoire Léonard, Satyajeet S. Bhonsale,
Monika E. Polańska, Filip Logist (Eds.)

PSE PRESS

An Efficient Convex Training Algorithm for Artificial Neural Networks by Utilizing Piecewise Linear Approximations and Semi-Continuous Formulations

Ece S. Köksal[a,b], Erdal Aydın[a], and Metin Türkay[c,d,*]

[a] Koc University, Department of Chemical and Biological Engineering, Istanbul, Sariyer, Turkey
[b] Turkish Petroleum Refineries Corporation, Kocaeli, Korfez, Turkey
[c] Koc University, Department of Industrial Engineering, Istanbul, Sariyer, Turkey
[d] SmartOpt, Istanbul, Turkey
* Corresponding Author: mturkay@ku.edu.tr.

ABSTRACT

Artificial neural networks are widely used as data-driven models for capturing complex, nonlinear systems. However, suboptimal training remains a significant challenge due to the nonlinearity of activation functions and the reliance on local solvers, which makes achieving global solutions difficult. One solution involves reformulating activation functions as piecewise linear approximations to convexify the problem, though this approach often requires substantial CPU time. This study demonstrates that a tailored branch-and-bound algorithm can effectively address these challenges by efficiently navigating the solution space using linear relaxations. The proposed method achieves minimal training error, offering a robust solution to the training bottleneck. Unlike traditional mixed-integer programming approaches, which often struggle to converge within reasonable CPU times, the SOSX algorithm shows superior scalability, with computational demand growing almost linearly rather than exponentially as problem size increases. Experiments on datasets with the same dimensionality confirm that the algorithm's efficiency is not specific to a particular dataset, as it demonstrates consistent CPU time trends across multiple datasets.

Keywords: artificial neural networks, piecewise linear functions, convex formulation, mixed-integer linear programming, computational complexity.

INTRODUCTION

A mechanistic process model relies on physical laws governing the system, but developing such models is in general time-consuming and challenging. In contrast, data-driven models offer a straightforward alternative, especially for representing nonlinear and complex systems. Artificial neural networks (ANNs), inspired by human brain neurons, are a popular type of data-driven models. Among them, feedforward neural networks (FNNs) are simpler, with data flowing in one direction from the input to the output layer, comprising an input layer, output layer, and user-defined hidden layers connected via activation functions.

Common activation functions include Rectified Linear Unit (ReLU), hyperbolic tangent ($tanh$), and sigmoid. ReLU is piecewise linear, while tanh and sigmoid are nonlinear and symmetric. These functions introduce nonlinearities, making ANN training a non-convex optimization problem classified under nonlinear programming (NLP). This can result in issues like underfitting, overfitting, or suboptimal solutions [1]. A major challenge is the reliance on local solvers, as global solvers often struggle to converge or require excessive CPU time due to nonlinearities in the activation function [2].

One way to address the non-convexity of neural network training is by using the piecewise linear (PWL) representation of nonlinear activation functions. Various methods for PWL formulations have been explored such as defining binary variables [3] or special ordered set variables [4]. This transforms the problem from an NLP formulation to a mixed-integer linear programming (MILP) formulation. In addition to these methods, Keha et al. proposed “SOSX variables,” which eliminate the need for

binary variables and enable PWL functions to be modelled with linear programming (LP) subproblems by modifying branching algorithms, yielding equivalent solutions to special ordered set type II (SOS2)-based MILP formulations [5].

Many studies have focused on optimizing ReLU-based neural networks, utilizing their naturally piecewise linear structure. Research on PWL ReLU-trained neural networks has primarily aimed at reducing computational time [6-8]. Beyond ReLU-based formulations, several studies have explored MILP formulations incorporating hyperbolic tangent activation functions in ANNs [9-10]. Similarly, Sildir and Aydin utilized the PWL representation of the hyperbolic tangent function during training with feature selection, effectively reducing the risk of suboptimal training and enhancing test accuracy [11]. However, a key limitation of this approach lies in the MILP formulation. As the number of features, neurons, and hidden layers in the NN increases, the number of binary variables grows exponentially, significantly increasing CPU time. This challenge underscores the need for more efficient MILP formulations to reduce computational demands while maintaining performance. Notably, studies addressing MILP-based neural network training remain relatively limited in the literature.

This work employs PWL approximations for the hyperbolic tangent activation function in a single-layer FNN [10-11] at the training phase. To address computational complexity, the approach by Keha et al. [5] is used to transform the MILP training problem into sequential LP problems by modifying the branching scheme, offering significant computational benefits. The novelty of this research lies in formulating the FNN training problem entirely as specially tailored LP subproblems.

METHODOLOGY

A feed-forward, fully connected artificial neural network with an identity function as the output layer can be expressed as:

$$\hat{y} = w_2 f(w_1 u + b_1) + b_2 \quad (1)$$

where $\hat{y}$ is the predicted output, u is the input, w_1, w_2, b_1 and b_2 are the weights and biases for the hidden and output layers, respectively. Typically, during training, mean squared error (MSE) is minimized. The objective function is given by:

$$\min_{w_1,w_2,b_1,b_2} MSE = \frac{1}{N}\sum_{i=1}^{N}(w_2 f(w_1 u_i + b_1) + b_2 - y_i)^2 \quad (2)$$

where N is the number of training data points and y_i is the actual i^{th} output value. To simplify the optimization and avoid the nonlinearity introduced by the square term, the mean absolute error (MAE) can be minimized instead. The objective function then becomes:

$$\min_{w_1,w_2,b_1,b_2} MAE = \frac{1}{N}\sum_{i=1}^{N}|w_2 f(w_1 u_i + b_1) + b_2 - y_i| \quad (3)$$

However, the absolute value introduces challenges for gradient-based optimization methods due to non-smoothness and non-differentiability at certain points. To address this, a linearized form of the absolute value can be used:

$$\min_{w_1,w_2,b_1,b_2} |\emptyset'| = \frac{1}{N}\sum_{i=1}^{N}|w_2 f(w_1 u_i + b_1) + b_2 - y_i| \quad (4a)$$

$$\emptyset' \le |\emptyset'| \quad (4b)$$

$$-|\emptyset'| \le \emptyset' \quad (4c)$$

Here, $|\emptyset'|$ is the modified objective function. For simplicity, the weight matrix of the output layer, w_2, can be extracted during regular training as w_2^* which is an extreme machine learning procedure [11-12].

Hyperbolic tangent (tanh) is a commonly used activation function in neural network training. To mitigate issues caused by its nonlinearity, several studies employ PWL approximations [9-11]. For 3-piece approximation, breakpoints can be defined as [-4,-1,+1,+4]. The 3-piece approximation of $tanh$ function can be defined as [3,10,11]:

$$tanh\,(x)_{3-piece} \approx \begin{cases} 0.08x - 0.68, & -4 \le x \le -1 \\ 0.76x, & -1 \le x \le +1 \\ 0.08x + 0.68, & +1 \le x \le +4 \end{cases} \quad (5)$$

Using a piecewise linear approximation of tanh, the training problem can be reformulated as a MILP problem:

$$\min_{w_1,w_2,b_1,b_2} |\emptyset'| = \frac{1}{N}\sum_{i=1}^{N}|w_2^* f(w_1 u_i + b_1) + b_2 - y_i| \quad (6a)$$

s.t.

$$x = w_1 u_i + b_1 \quad (6b)$$

$$p_0 \le w_1, b_1, b_2 \le p_P \quad (6c)$$

$$\Delta_k = p_k - p_{k-1} \quad (6d)$$

$$g_k = f'(p_k) - f'(p_{k-1}) \quad (6e)$$

$$\forall\, k \in \{1, \dots, P\} \quad (6f)$$

$$p_0 \le x \le p_P \quad (6g)$$

$$x = p_0 + \sum_{k=1}^{P} y_k \quad (6h)$$

$$f'(x) = f(p_0) + \sum_{k=1}^{P} \frac{g_k}{\Delta_k} y_k \quad (6i)$$

$$\lambda_0 \le z_0 \quad (6j)$$

$$\lambda_m \le z_{m-1} + z_m\,, \forall m \in \{1, \dots, P-1\} \quad (6k)$$

$$\lambda_m \le z_{m-1} \quad (6l)$$

$$\sum_{m=0}^{P-1} z_m = 1 \quad (6m)$$

$$z_m \in \{0,1\} \quad (6n)$$

where x is the input to the piecewise linear function, p_k denotes the k^{th} breakpoint, Δ_k represents the difference between consecutive breakpoints, and g_k is the difference in the function's value between these breakpoints. The parameter P specifies the number of pieces in the approximation. The variables y_k capture the differences used to define the input to the piecewise linear function, λ_m are nonnegative auxiliary variables, and z_m are binary variables that aid in formulating the problem as a MILP.

To simplify the problem and allow for formulation as a linear program (LP) before modifying the branching scheme, the following constraints can be used instead of (6j–6n) [13]:

$$\Delta_m s_{m+1} \le \Delta_{m+1} s_m \; \forall m \in \{1, \dots, P-1\} \quad (7a)$$

$$y_1 \le \Delta_1 \quad (7b)$$

$$y_P \geq 0 \quad (7c)$$

Finally, after obtaining the current solution $\tilde{y}$ with with $\tilde{y}_k < u_k$ and $\tilde{y}_{l+k} > 0$ for some $k \in \{1, \ldots, P-l\}$, the following branching options exist [13]:

$$Branch\ 1: y_k = \Delta_k, Branch\ 2: y_{k+1} = \cdots = y_P \quad (8)$$

Since formulations without binary variables provide the same LP bound as those that include them, maintain local ideality, and are more compact, they serve as superior models for PLFs compared to MIP models [5]. The algorithm can be summarized as follows where M denotes the number of neurons:

```
for i=1 to N do:
   for j=1 to M do:
      for k=1 to P do:
         Solve the model:
            Minimize Eq. 6a
            Subject to Eq. 6b-6i,Eq. 7a-7j
            If ỹ_k<u_k and ỹ_{l+k}>0 then
               Branch 1: y_k=Δ_k
               Branch 2: y_{k+1}=…=y_P
```

Implementing the SOSX algorithm in feedforward neural network training ensures reproducibility of results due to the convexity of the problem formulation. This property guarantees that the same training performance is achieved in every run, whereas regular training may converge to different solutions, potentially leading to suboptimal outcomes.

RESULTS AND DISCUSSION

We illustrate our proposed method on a case study that examines 253 snapshots of a distillation column, each described by 27 features, including variables such as temperatures, pressures, and flow rates, with the vapor pressure measured in a laboratory as the target variable [14]. The dataset is normalized between -1 and +1. Three training algorithms were applied to this dataset using a 70% training ratio. The first method, NLP training, employing the Adam optimizer [15] in TensorFlow's Keras [16] framework for 100 epochs. The second, MIP training, introducing binary variables to the problem structure, following previously established formulations in Eq.6. The third, SOSX training, using a convex branch-and-bound approach to train the neural network, recording CPU times. All methods used a single-hidden-layer FNN architecture with neurons ranging from 5 to 30, and for SOSX and MIP, piecewise linear approximation of the tanh activation function with 3 segments were implemented. Figure 1 shows the trend for SOSX training CPU times at different number of neurons. Table 1 summarizes the MSE results for both training and test performances. Figures 2 to 7 depict the corresponding trends for training and test performance across different piecewise approximations at varying number of neurons. In these visualizations, "n" denotes the number of neurons, while "p" represents the number of pieces used in the approximation. The training and test data are divided by a dashed black vertical line called the "train-test split."

The study initially sought to compare the CPU training times of the SOSX and MIP algorithms. However, under default solver settings, including cuts and optimality gap, MIP training failed to converge. As a result, MIP training was terminated once its CPU time equaled that of SOSX training. Consequently, the training and test results for the MIP algorithm are based on these aborted solutions.

A key finding pertains to CPU time. In MIP formulations, CPU time typically grows exponentially with an increase in the number of binary variables. In contrast, the SOSX algorithm, which replaces binary variables, demonstrates a non-exponential increase in CPU time, as represented in Figure 1.

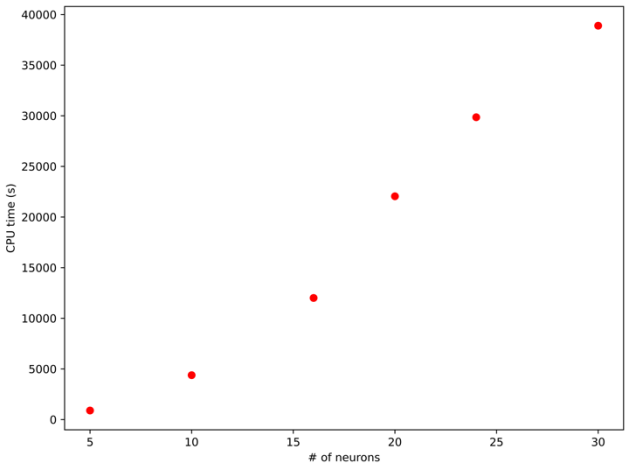

Figure 1. CPU times for SOSX training.

Another notable finding from the results is that the SOSX algorithm achieves nearly zero training error without signs of overfitting. The zero-training error is evident in the train split parts of Figures 2-7 as the actual and predicted values overlap. Additionally, in the test splits, SOSX predictions closely follow the actual test value trends, indicating no signs of overfitting. This success is attributed to its specially tailored branching mechanism, which strategically places PWL approximations of the nonlinear activation function. At each iteration, a linear relaxation is solved, and branching progresses based on the location within the PWL function. This approach not only ensures effective network training but also offers a substantial advantage in CPU efficiency.

Figures 2–7 depict the MIP training and test results using green dashed lines. In Figures 2 and 3, which correspond to cases with fewer neurons -specifically, 5-neuron-3-piece (*5n3p*) and 10-neuron-3-piece (*10n3p*)-, MIP training and test results closely follow the actual values compared to cases with a higher number of neurons, although SOSX still outperforms MIP. However, in Figure 4, which represents the 16-neuron-3-piece (*16n3p*) case, no MIP trend is observed, and Table 1 does not contain a corresponding value. This omission occurs because MIP training failed to converge to a feasible solution within the CPU time required by SOSX. In Figure 5, which illustrates the 20-neuron-3-piece (*20n3p*) case, a solution was found, but the MIP training and test results significantly deviate from the actual values. This divergence becomes even more pronounced in Figure 6 for the 24-neuron-3-piece (*24n3p*) case, where MIP training and test results fail to capture the trend altogether. Finally, in Figure 7, representing the 30-neuron-3-piece (*30n3p*) case, no MIP trend is present, mirroring the issue observed in the *16n3p* case. These findings highlight the robustness of the SOSX algorithm, which consistently finds a solution within a timeframe where MIP fails to yield a feasible result. Moreover, SOSX's effectiveness becomes even more evident in cases with a higher number of neurons, indicating its capability to handle more complex scenarios.

An important observation is the difference in test performance between the regular training (NLP) and SOSX algorithms. While SOSX may sometimes underperform due to its reliance on approximations, it still surpasses NLP in 4 out of 6 runs across different number of neurons. The lowest MSE test error was observed in the *24n3p* case, marking a particularly encouraging result. In the test splits of Figures 2–7, this pattern is clearly evident in the trends. Additionally, the actual data exhibit two peaks between data points 200 and 250. For the first peak, NLP predictions overshoot in the *5n3p*, *10n3p*, *16n3p*, and *30n3p* cases, as seen in Figures 2, 3, 4, and 7. Similarly, for the second peak, NLP predictions fall significantly below the actual value in the *24n3p* case, as shown in Figure 6, whereas SOSX predictions closely match the actual values. Overall, SOSX trends remain more aligned with the actual values compared to NLP test performance. These findings indicate that the SOSX algorithm effectively mitigates suboptimal training outcomes by leveraging its convex formulation and robust representation capabilities. Moreover, the benefits of the SOSX algorithm become even more evident in cases with a higher number of neurons.

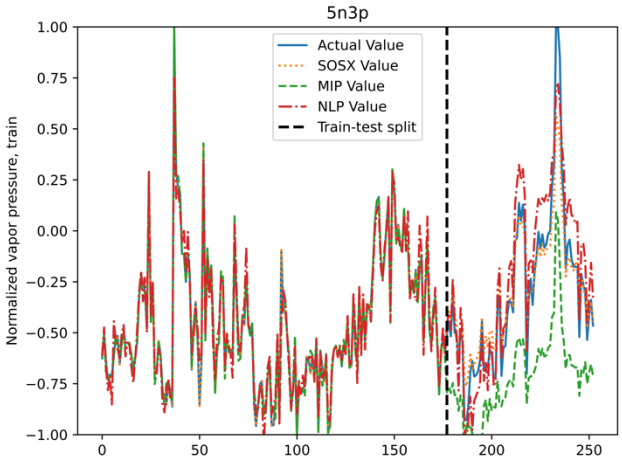

Figure 2. Training and test performances for 5 neurons.

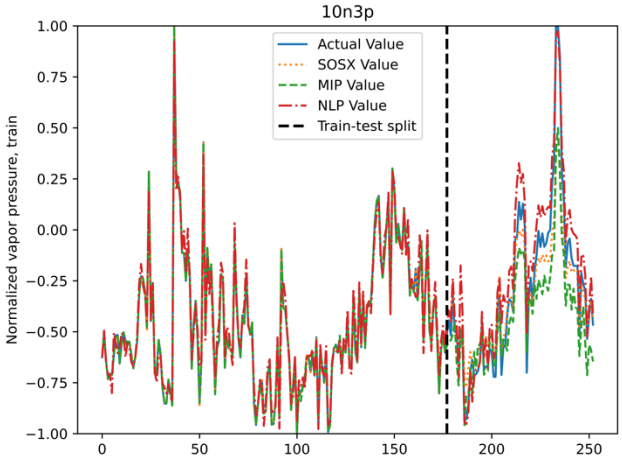

Figure 3. Training and test performances for 10 neurons.

#neuron	CPU time (s)	NLP training	MIP training	SOSX training	NLP testing	MIP testing	SOSX testing
5	890	0.0048	0.0022	0	0.0238	0.2237	0.0145
10	4388	0.0033	0.0002	0	0.0162	0.0393	0.0218
16	12011	0.0029	-	0	0.0291	-	0.0298
20	22052	0.0026	0.0823	0	0.0090	0.1981	0.0042
24	29856	0.0023	0.0853	0	0.0159	0.5641	0.0024
30	38897	0.0034	-	0	0.0233	-	0.0078

Table 1: Training CPU times, training and testing MSE value

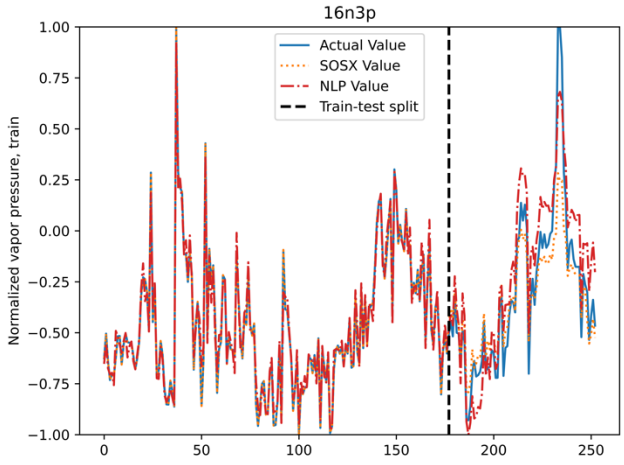

Figure 4. Training and test performances for 16 neurons.

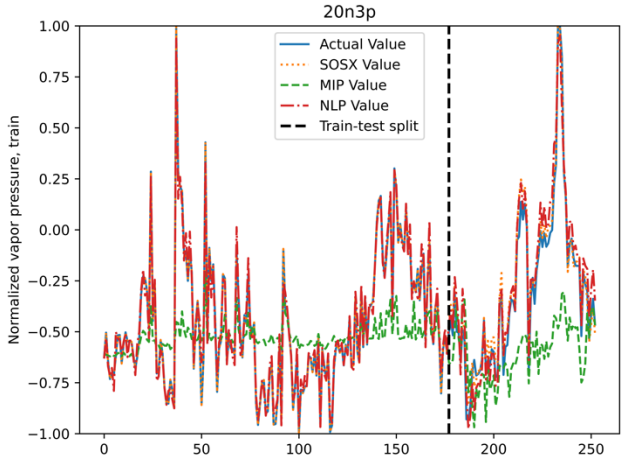

Figure 5. Training and test performances for 20 neurons.

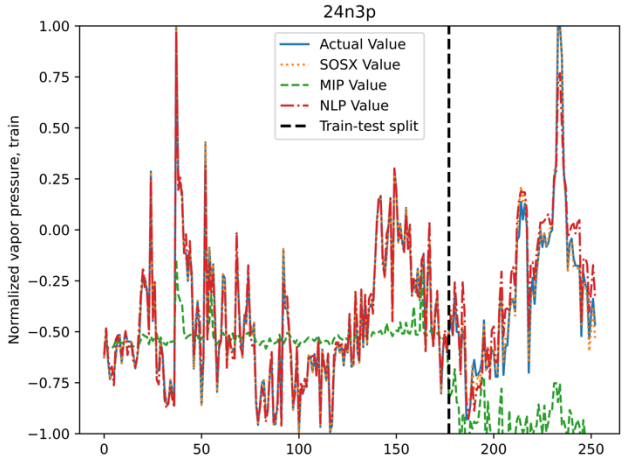

Figure 6. Training and test performances for 24 neurons.

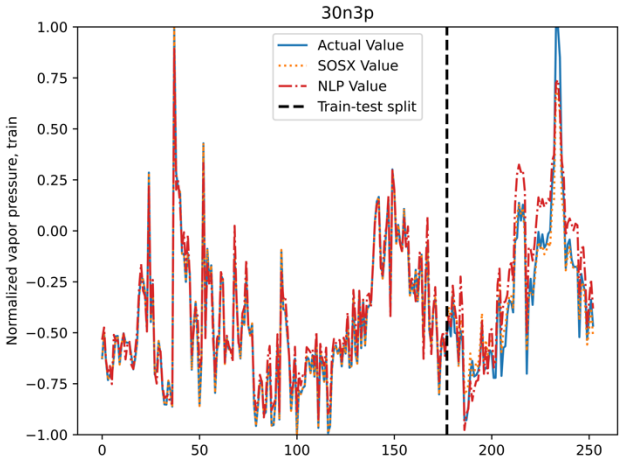

Figure 7. Training and test performances for 30 neurons.

The analysis of SOSX CPU times previously focused on how they vary with the number of neurons and piecewise approximations. However, an important question remains: Do these CPU times only apply to the distillation column dataset, or are they statistically significant for datasets with similar dimensionality? To explore this, four datasets are modified to match in dimensionality, ensuring consistency in both dataset size and number of features.

The first dataset is the previously examined distillation column data [14]. The second dataset consists of open simulation data from a wastewater treatment plant, where the target variable represents dissolved oxygen concentration in an activated sludge system [17]. The third dataset comes from a U.S.-based regression study predicting the average annual cancer diagnosis rate based on socioeconomic and demographic factors such as income, population size, age, and race [18]. The fourth dataset is the Boston housing prices dataset, which estimates property values using economic and real estate-related variables [19]. To ensure comparability, all datasets are reduced to eight features using F-value statistics, selecting the variables most strongly correlated with the output. Additionally, the training dataset sizes are standardized to 177 samples across all cases.

Figure 8 presents variations in the logarithm of CPU times across different number of neurons and segmentations in piecewise linear approximations for the four datasets. The plots include 95% confidence intervals, allowing for statistical comparisons between datasets. Since the dataset sizes are adjusted for consistency, their training and test performances should not be directly compared to original benchmarks. Instead, the focus is on determining whether the observed CPU times are statistically significant.

As shown in Figure 8, most data points fall within the 95% confidence interval. Any points outside this range are still close to the boundary and do not qualify as outliers. This suggests that for datasets with the same dimensionality, CPU times remain consistent. Therefore, the CPU times observed in the original analysis are not unique to the distillation column dataset.

CONCLUSION

This study highlights the efficacy and robustness of the "SOSX" algorithm for convex training of neural networks using PWL approximations. SOSX consistently achieves zero training error while balancing computational efficiency and model accuracy through modifications to the branch-and-bound algorithm. Unlike MIP training, which struggles with convergence for larger problems, SOSX offers superior performance with significantly reduced CPU times.

A comparative analysis reveals key findings. While regular (NLP) training provides reasonable test performance, SOSX outperforms NLP in many cases, such as the *24n3p* configuration, which achieved the lowest MSE test error. Additionally, MIP training struggles with convergence for larger networks, failing to provide feasible solutions within the time frame needed for SOSX.

CPU time analysis shows that SOSX demonstrates greater scalability. Unlike MIP, which exhibits exponential CPU time growth, SOSX displays a more manageable polynomial increase, making it a practical choice for larger-scale problems. In addition, experiments on four different datasets with the same dimensionality show that the CPU times are not dataset specific.

In summary, SOSX offers a robust, efficient framework for training neural networks with piecewise linear approximations. Its ability to achieve zero training error, coupled with its scalability, makes it a promising

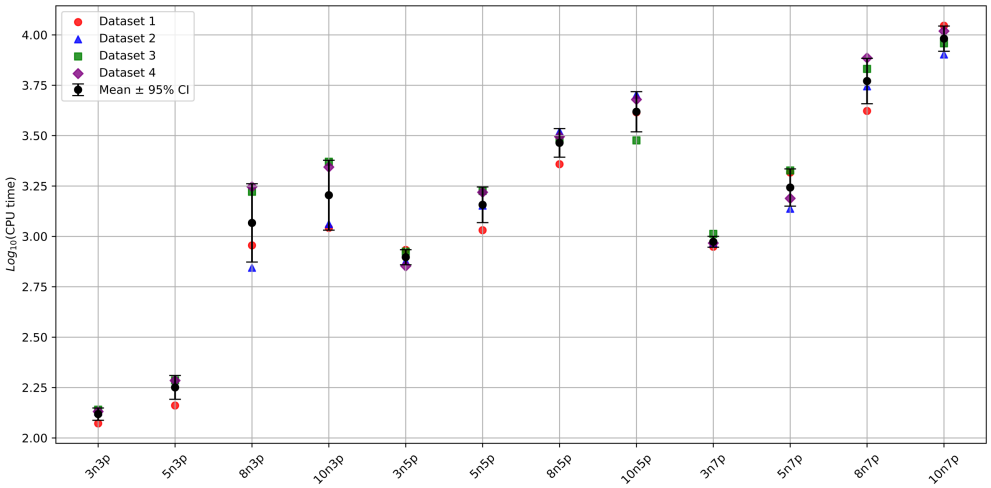

Figure 8. CPU times for different datasets.

alternative to methods like NLP and MIP, with strong potential for broader machine learning and optimization applications.

REFERENCES

1. A. Thebelt, J. Wiebe, J. Kronqvist, C. Tsay, and R. Misener, "Maximizing information from chemical engineering data sets: Applications to machine learning," *Chem Eng Sci*, vol. 252, Apr. 2022, doi: 10.1016/j.ces.2022.117469.
2. M. Bianchini and M. Gori, "Optimal learning in artificial neural networks: A review of theoretical results," 1996.
3. J. P. Vielma, "Mixed integer linear programming formulation techniques," *SIAM Review*, vol. 57, no. 1, pp. 3–57, 2015, doi: 10.1137/130915303.
4. E. M. L. Beale and J. A. Tomlin, "Special facilities in a general mathematical programming system for nonconvex problems using ordered sets of variables," *Operational Research*, vol. 69, pp. 447–454, 1970, [Online]. Available: https://www.researchgate.net/publication/313166553.
5. A. B. Keha, I. R. De Farias, and G. L. Nemhauser, "Models for representing piecewise linear cost functions," *Operations Research Letters*, vol. 32, no. 1, pp. 44–48, Jan. 2004, doi: 10.1016/S0167-6377(03)00059-2.
6. C. Tsay, J. Kronqvist, A. Thebelt, and R. Misener, "Partition-based formulations for mixed-integer optimization of trained ReLU neural networks," Feb. 2021, [Online]. Available: http://arxiv.org/abs/2102.04373
7. J. Katz, I. Pappas, S. Avraamidou, and E. N. Pistikopoulos, "Integrating deep learning models and multiparametric programming," *Comput Chem Eng*, vol. 136, May 2020, doi: 10.1016/j.compchemeng.2020.106801.
8. B. Grimstad and H. Andersson, "ReLU networks as surrogate models in mixed-integer linear programs," *Comput Chem Eng*, vol. 131, Dec. 2019, doi: 10.1016/j.compchemeng.2019.106580.
9. A. M. Schweidtmann and A. Mitsos, "Deterministic Global Optimization with Artificial Neural Networks Embedded," *J Optim Theory Appl*, vol. 180, no. 3, pp. 925–948, Mar. 2019, doi: 10.1007/s10957-018-1396-0.
10. E. S. Koksal and E. Aydin, "Physics Informed Piecewise Linear Neural Networks for Process Optimization," *Comput Chem Eng*, vol. 174, Jun. 2023, doi: 10.1016/j.compchemeng.2023.108244.
11. H. Sildir and E. Aydin, "A Mixed-Integer linear programming based training and feature selection method for artificial neural networks using piece-wise linear approximations," *Chem Eng Sci*, vol. 249, Feb. 2022, doi: 10.1016/j.ces.2021.117273.
12. S. Ding, H. Zhao, Y. Zhang, X. Xu, and R. Nie, "Extreme learning machine: algorithm, theory and applications," *Artif Intell Rev*, vol. 44, no. 1, pp. 103–115, Jun. 2015, doi: 10.1007/s10462-013-9405-z.
13. A. B. Keha, I. R. De Farias, and G. L. Nemhauser, "Models for representing piecewise linear cost functions," *Operations Research Letters*, vol. 32, no. 1, pp. 44–48, Jan. 2004, doi: 10.1016/S0167-6377(03)00059-2.
14. K. Dunn, "Distillation tower [Dataset]," OpenMV.
15. D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," Dec. 2014, [Online]. Available: http://arxiv.org/abs/1412.6980
16. F. Chollet and et al, "Keras." [Online]. Available: https://keras.io
17. Vanhooren, H., Nguyen, K., Vanrolleghem, P., & Spanjers, H. (1996). *Development of a simulation protocol for evaluation of respirometry-based control strategies*.
18. Otto, M., & Fong, A. (2013). Cancer data of United States of America [Dataset]. *Kaggle. Released under MIT Licence*. https://www.kaggle.com/datasets/varunraskar/cancer-regression/code
19. Harrison, D., & Rubinfeld, D. L. (1978). Hedonic Housing Prices and the Demand for Clean Air. *Journal of Environmental Economics and Management*, *5*, 81–102.