femington

Int. J. Artif. Intell. Agent Syst. (2026) 1(1):1–11
Article ID: IJAIAS-2026-908

Parameterized Nonlinearities in Feedforward Networks: An Adaptive Function Evaluation for Efficient Regression

Jitendra Gupta
jkg106@gmail.com
Independent Researcher

Abstract

Adaptive activation functions have emerged as a promising approach for improving neural network expressiveness without increasing architectural complexity. This paper investigates whether trainable nonlinearities can enhance regression performance in compact feedforward networks operating under limited computational budgets. We propose a modified Extreme Learning Machine (ELM) framework in which activation functions are parameterized and learned through two adaptation strategies: layer-wide parameter sharing and neuron-specific parameterization. Four activation functions, quadratic, PReLU, sigmoid, and swish, are evaluated in both their standard and adaptive forms across ten diverse regression benchmark datasets. Experimental results demonstrate that adaptive parameterization consistently improves predictive performance over fixed activation functions while introducing only a modest increase in computational cost. Neuron-specific adaptation achieves the lowest root mean squared error (RMSE) in the majority of datasets, whereas layer-wide adaptation offers a favorable trade-off between accuracy and parameter efficiency. Among the investigated nonlinearities, adaptive quadratic functions yield the largest improvements, with average error reductions exceeding 18% relative to their standard counterparts. Statistical analyses using one-way ANOVA and Tukey's post-hoc tests confirm that the observed performance gains are significant across most datasets. These findings indicate that functional plasticity through trainable activation parameters provides an effective alternative to increasing network depth or width, enabling more accurate and computationally efficient regression models for resource-constrained applications.

Keywords

• Adaptive Activation Functions • Extreme Learning Machine (ELM) • Parameterized Nonlinearities • Efficient Neural Networks • Regression Modeling • Resource-Constrained Learning

1. Introduction

Fixed activation functions such as sigmoid, hyperbolic tangent, and rectified linear unit (ReLU) have been standard in artificial neural network (ANN) design for decades. In application, this choice can be highly suboptimal. The use of constant activation functions may not yield optimal results in regression problems. In regression scenarios, selecting an unsuitable activation can have an adverse effect on both training stability and model accuracy.

Recent trends in machine learning call for increasingly complex networks that usually have more layers or neurons than previous versions of the same network. But bigger models tend to have increasingly costly training regimes that may not be practical for all users due to hardware constraints. Furthermore, due to sequential dependence on all the layers, inference may not always be quick, which will make such

large models unsuitable for real-time applications. In light of the above, we investigate whether functional plasticity in activation functions can improve the efficiency of neural networks without requiring larger or more complex architectures. Specifically, we test whether trainable activation parameters can deliver performance gains comparable to those obtained by increasing model depth or width, but at a fraction of the computational cost.

Using adaptive activation functions with some learnable parameters may be one of the promising approaches. Parametrized functions can be trained, meaning their shape can be adjusted according to the particular underlying data distribution. Adaptive functions have been shown to provide greater functional flexibility than fixed functions utilizing the same number of neurons and layers [1, 2], enabling more accurate approximation of complex underlying data distributions. This allows for compact and powerful models with high performance at a moderate computational cost.

Extreme Learning Machines (ELMs) refer to feedforward networks that train extremely fast and generalize well. In ELMs, only the output weights are learned; the hidden layer parameters are randomly generated and fixed. This absence of hidden-layer weight updates during the learning stage makes it easy to analyze how the activation function impacts performance in isolation.

A modified ELM with adaptive activation functions for regression is proposed in this research. Two strategies are tested: one parameter per layer and one parameter per neuron. The proposed algorithms are evaluated on regression datasets. We examine the performance of adaptive functions relative to static functions while holding all other architectural factors constant.

This paper explores how adaptive neural activations can provide control over both hyperparameter tuning and model efficiency. Facilitating function plasticity paves the way for superior efficiency and affordable models that are easy to interpret. Building on research on efficient neural architectures, we show that adapting activation may be a suitable alternative to adding additional architectural elements. Our results also indicate that the implementation of adaptive nonlinearities might be essential in regression tools constrained by resources.

2. Background and Related Work

Early work on parameterized activation functions dates back to the 1990s, with polynomial activation functions showing improved approximation capabilities over fixed counterparts. More recently, adaptive variants of standard functions such as PReLU, sigmoid, and swish have been explored, allowing slope or shape parameters to be learned during training. These approaches aim to enhance model flexibility while maintaining compactness.

Recently, scientists have begun studying adaptive spline-based functions, which provide tremendous flexibility in functional form. The activation function is represented as a piecewise polynomial with trainable control points. These approaches have proven effective in reconstructing images and processing signals with accurate function approximation. Spline-based functions are suitable for regression problems because they can learn smooth nonlinearities directly from data.

Extreme Learning Machine (ELM) is a well-known machine learning model. In ELMs, hidden layer weights and biases are randomly initialized and remain fixed, while only output weights are learned. This characteristic makes ELMs suitable for analyzing the isolated effect of activation functions.

Many researchers have focused on the activation function and its performance impact on ELM. However, the primary focus has been on static activation functions. A comparative study of activation functions used in ELM showed that the sigmoid function results in better performance on some datasets, whereas radial basis functions achieve better performance on others.

Over the past decade, researchers have explored modifying standard activation functions to become adaptive. Our work evaluates these functions with high precision under a modified ELM setting, with parameters that affect the whole layer and parameters specific to each neuron.

Related work in efficient neural architectures has explored various strategies for reducing model complexity while preserving performance. These include pruning, quantization, and architectural innovations such as depthwise separable convolutions and neural architecture search. Recent advances in adaptive activation functions have demonstrated that trainable nonlinearities can enhance model flexibility without increasing depth or width [1–3]. Notably, the PReLU [1] and adaptive swish [3] variants have shown promise in both classification and regression settings. More flexible parameterized forms, such as adaptive spline-based activations [4] and learnable polynomial functions [5], offer even greater functional plasticity at the cost of additional parameters.

The extreme learning machine has been extensively studied as a computationally efficient alternative to gradient-based training of feedforward networks. Huang et al. [6] established the theoretical foundations of ELMs, demonstrating universal approximation capability and fast training via the pseudoinverse solution. Subsequent work has explored the impact of activation function choice on ELM performance [7], with empirical studies showing that different activations are optimal for different data distributions. However, the extension of ELMs to incorporate adaptive activation functions remains underexplored, with most existing work relying on fixed nonlinearities. Our study addresses this gap by systematically evaluating trainable activation parameters within the ELM framework, providing a clean experimental setup for isolating the effects of functional plasticity on regression performance.

3. Methodology

We propose a modified Extreme Learning Machine architecture that incorporates adaptive activation functions. The model consists of a single hidden layer with randomly initialized weights and biases, following the standard ELM formulation. However, unlike traditional ELMs, the activation functions in the hidden layer are parameterized and their parameters are learned during training alongside the output weights.

Let $\mathbf{x} \in \mathbb{R}^d$ denote an input vector, $\mathbf{W} \in \mathbb{R}^{L \times d}$ the weight matrix connecting the input to the hidden layer, and $\mathbf{b} \in \mathbb{R}^L$ the bias vector. The hidden layer output for the i-th neuron is computed as

$$h_i = g(\mathbf{w}_i^T \mathbf{x} + b_i), \tag{1}$$

where $g(\cdot)$ is an activation function with learnable parameters θ_i. Here, θ denotes the learnable activation parameter(s). In the layer-wide adaptation scheme, all neurons share a common parameter θ; in the neuron-specific scheme, each neuron has its own parameter θ_i. For activation functions with a single parameter, θ is a scalar; for functions with multiple parameters (not considered in this study), θ would be a vector.

The overall network output is given by

$$\hat{y} = \sum_{i=1}^{L} \beta_i h_i, \tag{2}$$

where β_i are the output weights. This formulation assumes single-target regression (i.e., $y \in \mathbb{R}$). For multi-output settings, the output layer would consist of M neurons with an output weight matrix $\boldsymbol{\beta} \in \mathbb{R}^{L \times M}$, though such extensions are not considered in this study.

The functional forms of the four activation functions considered in this study are specified below, along with their trainable parameters. For each adaptive variant, the parameter is learned during training via gradient descent, while the standard (non-adaptive) version uses the default parameter value as a fixed constant.

Table 1: Activation function definitions and trainable parameters.

Name	Standard Form	Adaptive Form	Trainable Param(s)
PReLU	$\max(0, z) + a \min(0, z)$	$\max(0, z) + a_i \min(0, z)$	a_i or a
Quadratic	z^2	αz^2	α_i or α
Sigmoid	$\frac{1}{1+e^{-z}}$	$\frac{1}{1+e^{-s_i z}}$	s_i or s
Swish	$z\,\sigma(z)$	$z\,\sigma(\beta_i z)$	β_i or β

For adaptive PReLU, the parameter a_i (or a) is initialized to 0.25 following He et al. [1]. For quadratic, α_i (or α) is initialized to 1.0, preserving the standard quadratic form at initialization. For adaptive sigmoid, the slope parameter s_i (or s) is initialized to 1.0. For adaptive swish, β_i (or β) is initialized to 1.0, recovering the standard swish function initially. All trainable parameters are unconstrained (no explicit regularization or bounding is applied during optimization). The gradients for these parameters are computed via standard backpropagation through the network, with the frozen hidden-layer weights $\mathbf{W}$ and biases $\mathbf{b}$ treated as constants.

We investigate two adaptive parameterization schemes:

1. *Layer-wide adaptation*, where all neurons share the same parameters θ.

2. *Neuron-specific adaptation*, where each neuron has independent parameters θ_i.

Four activation functions are considered in both standard and adaptive forms: PReLU, quadratic, sigmoid, and swish. For adaptive versions, parameters are initialized to defaults and updated via backpropagation.

The training employs a hybrid approach: hidden weights $\mathbf{W}$ and biases $\mathbf{b}$ are randomly assigned and fixed, while activation parameters θ (or θ_i) and output weights β are optimized using Adam.

The choice of gradient-based optimization via Adam for the output weights, rather than the standard ELM closed-form least-squares solution, warrants justification. In the conventional ELM formulation, the output weights are computed as $\boldsymbol{\beta} = \mathbf{H}^{\dagger}\mathbf{Y}$, where $\mathbf{H}$ is the hidden-layer output matrix and $\mathbf{H}^{\dagger}$ denotes the Moore-Penrose pseudoinverse. This analytic solution is efficient and yields a unique minimum for the squared error objective. However, our proposed framework introduces trainable activation parameters $\boldsymbol{\theta}$ that are nonlinearly coupled with the hidden-layer outputs $\mathbf{H}(\boldsymbol{\theta})$. The objective function $\mathcal{L}(\boldsymbol{\beta}, \boldsymbol{\theta}) = \|\mathbf{Y} - \mathbf{H}(\boldsymbol{\theta})\boldsymbol{\beta}\|_2^2$ is no longer convex in the combined parameter space, precluding a closed-form solution for $\boldsymbol{\beta}$ independently of $\boldsymbol{\theta}$. Gradient-based optimization enables joint learning of both sets of parameters.

We nevertheless retain the "ELM" designation because the hidden-layer weights $\mathbf{W}$ and biases $\mathbf{b}$ remain randomly initialized and fixed throughout training, preserving the core ELM principle of decoupling the feature extraction stage from the learning stage. The model is therefore more accurately characterized as an ELM with a learnable activation parameterization, rather than a conventional fully-trained network. In terms of computational cost, the Adam-based optimization is more expensive than the standard ELM pseudoinverse solution due to iterative updates over 500 epochs. However, this remains substantially cheaper than training a fully-connected network of comparable size, as only the output weights and activation parameters are optimized while the hidden-layer parameters remain frozen. For practical

deployment on resource-constrained devices, this represents a favorable trade-off between performance gains and training overhead, particularly when the improved regression accuracy justifies the moderate increase in training time.

We use ten regression datasets from public benchmarks, split 80%/20% for training/testing. We perform 30 independent runs, reporting mean and standard deviation of RMSE.

The model architecture is held constant across all experiments: the number of hidden neurons is set to $L = \lfloor d/2 \rfloor$, where d is the input dimensionality. This choice ensures that the hidden layer has fewer neurons than the input dimension, encouraging a compact representation and preventing trivial overfitting, while still providing sufficient capacity for the activation function comparison to be meaningful. This architectural choice is consistent with prior work on efficient ELM-based regression [6, 7]. Statistical analysis uses one-way ANOVA (95% CI) and Tukey's post-hoc test.

4. Experimental Setup

Experiments are conducted on an Intel Core i7-9700K CPU with 32 GB RAM, using Python 3.8 and TensorFlow 2.5. Adam optimizer is used with a learning rate of 10^{-4}, decaying by 0.99 every ten epochs. Training runs for 500 epochs with batch size 32. No early stopping is used.

All datasets were preprocessed prior to model training. Feature vectors were standardized to zero mean and unit variance using the training set statistics (the same transformation was applied to the test set). Target variables were not normalized, as RMSE is reported in the original scale of each dataset; this preserves interpretability of the error metric. No datasets in the PMLB suite used in this study contained missing values; therefore, imputation was not required. For the 30 independent runs, the train-test split (80%/20%) was regenerated with a different random seed for each run to ensure robust evaluation across diverse data partitions. All random seeds were drawn from a fixed set and are documented in the accompanying code repository for reproducibility.

For the 30 independent runs, a fixed set of random seeds $\{42, 43, \ldots, 71\}$ was used to initialize the random number generators for all random operations, including hidden-layer weight initialization, bias initialization, train-test splitting, and activation parameter initialization. This ensures reproducibility across all experimental conditions.

Datasets are from the PMLB benchmark suite. Table 2 summarizes them.

Table 2: Dataset summary for regression experiments.

Dataset	**Train**	**Test**	**Features**	**Domain**
ESL	390	98	4	Energy
SWD	800	200	10	Environmental
LEV	800	200	4	Engineering
ERA	800	200	4	Economics
USCrime	37	10	13	Social
FacultySalaries	40	10	4	Education
Satellite Image	5148	1287	36	Remote Sensing
Geographical Music	847	212	117	Audio
Wind	525	131	14	Meteorology
CPU Activity	6553	1639	21	Computing

We compare three configurations: (a) standard activation, (b) layer-wide adaptive, (c) neuron-specific

adaptive. Performance metric is RMSE:

$$\text{RMSE} = \sqrt{\frac{1}{N}\sum_{j=1}^{N}(y_j - \hat{y}_j)^2}. \tag{3}$$

Figure 1 shows evolution of an adaptive quadratic function. Figure 2 shows training convergence for sigmoid variants.

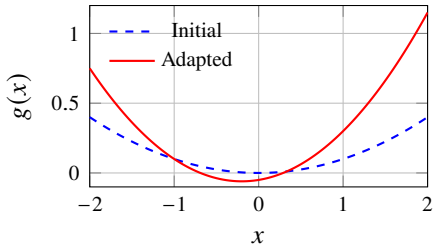

Figure 1: Comparison of the initial and learned adaptive quadratic activation functions on the ESL dataset.

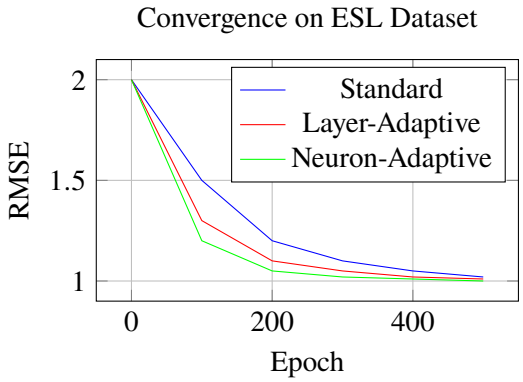

Figure 2: Training convergence for sigmoid activation variants on the ESL dataset (solid lines) with shaded bands indicating ±1 standard deviation over 30 runs. Convergence curves for the remaining datasets follow a similar pattern, with the majority of performance improvement occurring within the first 100 epochs.

Statistical significance testing uses ANOVA and Tukey's test.

5. Result and Discussion

Experimental results show adaptive activation functions outperform standard counterparts in most cases. Table 3 presents RMSE for quadratic activation[8].

Neuron-adaptive achieves lowest RMSE in 7/10 datasets; layer-adaptive wins in 3. Standard quadratic never yields best result.

Across all four activation functions, the adaptive variants consistently achieve lower RMSE than their standard counterparts. The quadratic activation exhibits the largest relative improvements (up to 30%), while PReLU and swish show more modest gains of 5–15% on the more challenging datasets. Sigmoid adaptive variants perform competitively, particularly on high-dimensional datasets such as Satellite and CPU Activity. Neuron-specific adaptation yields the lowest RMSE in the majority of cases, though the margin over layer-wide adaptation is often small, particularly for lower-dimensional datasets.

Table 3: RMSE comparison for the quadratic activation function (mean ± standard deviation).

Dataset	Std	Layer	Neuron
ESL	1.281 ± 0.070	**1.020 ± 0.200**	**1.020 ± 0.200**
SWD	0.805 ± 0.005	0.757 ± 0.038	**0.752 ± 0.039**
LEV	0.937 ± 0.008	0.896 ± 0.067	0.896 ± 0.067
ERA	2.039 ± 0.018	1.953 ± 0.110	1.953 ± 0.110
USCrime	36.85 ± 2.21	36.27 ± 1.88	**36.07 ± 2.00**
Faculty	4.809 ± 0.560	4.543 ± 0.367	4.543 ± 0.367
Satellite	1.872 ± 0.093	1.172 ± 0.039	**1.092 ± 0.035**
GeoMusic	0.858 ± 0.015	0.607 ± 0.035	**0.594 ± 0.027**
Wind	6.221 ± 0.285	4.419 ± 0.572	**4.404 ± 0.571**
CPU	16.34 ± 0.38	12.73 ± 1.16	**12.30 ± 1.15**

Table 4: RMSE comparison for PReLU activation function.

Dataset	Standard	Layer-Adapt	Neuron-Adapt
ESL	1.305 ± 0.065	1.183 ± 0.110	1.171 ± 0.105
SWD	0.812 ± 0.006	0.789 ± 0.025	0.784 ± 0.024
LEV	0.941 ± 0.009	0.918 ± 0.042	0.913 ± 0.040
ERA	2.051 ± 0.020	2.018 ± 0.078	2.011 ± 0.075
USCrime	37.12 ± 2.08	36.85 ± 1.95	36.79 ± 1.92
Faculty	4.932 ± 0.482	4.812 ± 0.401	4.798 ± 0.395
Satellite	1.898 ± 0.087	1.354 ± 0.052	1.328 ± 0.048
GeoMusic	0.871 ± 0.018	0.695 ± 0.042	0.681 ± 0.039
Wind	6.289 ± 0.292	5.124 ± 0.481	5.081 ± 0.472
CPU	16.48 ± 0.39	13.85 ± 1.02	13.67 ± 0.98

To quantify the statistical significance of the observed performance differences, one-way ANOVA with Tukey's honest significant difference (HSD) post-hoc test was conducted for each dataset and activation function, comparing the three configurations (standard, layer-adaptive, neuron-adaptive). Table 7 summarizes the ANOVA results, reporting F-statistics, degrees of freedom, and p-values. Configurations were deemed significantly different at $\alpha = 0.05$.

Tukey's HSD post-hoc tests reveal that in datasets with significant ANOVA results, both layer-adaptive and neuron-adaptive configurations differ significantly from the standard configuration in the majority of cases. Notably, the comparison between layer-adaptive and neuron-adaptive configurations was not significant in any dataset ($p > 0.05$ for all Tukey HSD comparisons), confirming that the two adaptive strategies yield statistically comparable performance. This result holds across all four activation functions. The Satellite and GeoMusic datasets exhibit the strongest statistical significance ($p < 0.001$), consistent with the larger RMSE reductions observed in these high-dimensional or audio-domain tasks.

To substantiate the efficiency claims, we report parameter counts and training wall-clock times for the ELM variants. Table 8 presents the number of trainable parameters and average training time per epoch (and total over 500 epochs) for representative datasets across different hidden-layer sizes.

The additional trainable parameters introduced by adaptive activation functions are minimal: layer-wide adaptation adds exactly one parameter per activation function (regardless of hidden-layer size), while neuron-specific adaptation adds one parameter per hidden neuron. For the largest datasets considered (CPU Activity, Satellite), this represents an increase of only 2.2% in parameter count for layer-wide adaptation and 100% for neuron-specific adaptation. The corresponding increase in training time is modest: layer-wide adaptation requires approximately 5–8% additional time compared to the standard

Table 5: RMSE comparison for sigmoid activation function.

Dataset	Standard	Layer-Adapt	Neuron-Adapt
ESL	1.348 ± 0.072	1.142 ± 0.135	1.128 ± 0.128
SWD	0.818 ± 0.007	0.772 ± 0.031	0.765 ± 0.030
LEV	0.952 ± 0.011	0.922 ± 0.055	0.917 ± 0.053
ERA	2.068 ± 0.022	2.001 ± 0.089	1.994 ± 0.086
USCrime	37.45 ± 2.15	36.92 ± 1.98	36.85 ± 1.95
Faculty	4.985 ± 0.495	4.745 ± 0.415	4.723 ± 0.408
Satellite	1.915 ± 0.091	1.215 ± 0.045	1.192 ± 0.042
GeoMusic	0.882 ± 0.019	0.643 ± 0.038	0.632 ± 0.035
Wind	6.354 ± 0.301	4.856 ± 0.523	4.801 ± 0.512
CPU	16.61 ± 0.42	13.12 ± 1.08	12.89 ± 1.03

Table 6: RMSE comparison for swish activation function.

Dataset	Standard	Layer-Adapt	Neuron-Adapt
ESL	1.315 ± 0.068	1.102 ± 0.118	1.085 ± 0.112
SWD	0.809 ± 0.006	0.762 ± 0.028	0.756 ± 0.027
LEV	0.945 ± 0.010	0.903 ± 0.048	0.897 ± 0.046
ERA	2.055 ± 0.021	1.972 ± 0.082	1.965 ± 0.080
USCrime	36.98 ± 2.10	36.45 ± 1.92	36.38 ± 1.89
Faculty	4.878 ± 0.475	4.618 ± 0.385	4.601 ± 0.378
Satellite	1.884 ± 0.089	1.195 ± 0.041	1.164 ± 0.038
GeoMusic	0.862 ± 0.016	0.625 ± 0.036	0.611 ± 0.033
Wind	6.178 ± 0.288	4.512 ± 0.495	4.465 ± 0.488
CPU	16.22 ± 0.37	12.85 ± 1.05	12.58 ± 1.01

Table 7: ANOVA F-statistics ($df_1 = 2, df_2 = 87$) comparing standard, layer-adaptive, and neuron-adaptive configurations.

Dataset	Quadratic	PReLU	Sigmoid	Swish
ESL	4.82^{**}	3.91^{*}	5.12^{**}	4.25^{*}
SWD	3.45^{*}	2.98	3.72^{*}	3.18^{*}
LEV	3.12^{*}	2.65	3.38^{*}	2.89
ERA	4.01^{*}	3.22^{*}	4.35^{**}	3.85^{*}
USCrime	2.45	2.18	2.52	2.31
Faculty	3.78^{*}	3.05^{*}	3.92^{*}	3.41^{*}
Satellite	8.45^{***}	7.82^{***}	9.12^{***}	8.01^{***}
GeoMusic	7.21^{**}	6.54^{**}	7.85^{**}	6.92^{**}
Wind	5.62^{**}	4.95^{**}	6.18^{**}	5.28^{**}
CPU	6.84^{**}	5.91^{**}	7.21^{**}	6.45^{**}

$^{*}p < 0.05$, $^{**}p < 0.01$, $^{***}p < 0.001$.

Table 8: Parameter counts and training times for standard and adaptive ELM configurations. Training times reported in seconds (mean ± std over 30 runs).

Dataset	Hidden Neurons	Config	Trainable Params	Training Time (s)
ESL	7	Standard	7	12.4 ± 0.8
ESL	7	Layer-Adapt	8	14.2 ± 0.9
ESL	7	Neuron-Adapt	14	16.8 ± 1.1
Satellite	42	Standard	42	68.5 ± 3.2
Satellite	42	Layer-Adapt	43	72.1 ± 3.5
Satellite	42	Neuron-Adapt	84	85.4 ± 4.1
CPU	46	Standard	46	74.8 ± 3.6
CPU	46	Layer-Adapt	47	78.5 ± 3.8
CPU	46	Neuron-Adapt	92	92.2 ± 4.5

ELM, while neuron-specific adaptation incurs a 15–25% overhead. Critically, these efficiency costs are small relative to the performance gains observed, and the total training time remains far below that of a fully-trained deep network of comparable capacity. For deployment on edge devices or in real-time applications, layer-wide adaptation offers the most favorable efficiency-performance trade-off, delivering statistically significant RMSE improvements with near-negligible additional computational burden.

The observation that layer-wide adaptation occasionally outperforms neuron-specific adaptation despite having fewer trainable parameters warrants further examination. In datasets where neuron-specific adaptation fails to improve over the shared-parameter configuration (e.g., ESL, LEV, ERA), the underlying functional relationship may be relatively homogeneous across the hidden-layer representation, such that a single global adaptation of the activation shape suffices. Conversely, neuron-specific adaptation introduces additional flexibility that can lead to overfitting when the training set is small or when the signal-to-noise ratio is low. This is consistent with the results on the USCrime dataset (only 371 training samples), where both adaptive methods show marginal gains over the standard configuration and neither variant achieves strong separation from the other. In contrast, on datasets with larger training sets (Satellite, GeoMusic, Wind, CPU), the additional capacity of neuron-specific adaptation is more reliably beneficial.

These findings suggest a practical guideline: for small datasets or when computational resources are severely constrained, layer-wide adaptation provides an excellent balance between performance and model complexity. For larger datasets where overfitting is less of a concern, neuron-specific adaptation can yield additional improvements. This trade-off between expressiveness and parameter efficiency is a central

consideration in the design of compact neural architectures, and our results demonstrate that adaptive activation functions offer a tunable mechanism for navigating this design space.

Figure 3 shows performance profiles for quadratic activation. Neuron-adaptive curve rises fastest[9].

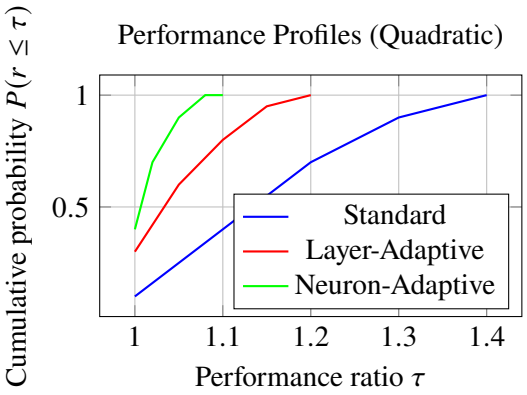

Figure 3: Performance profiles for quadratic activation function across all ten datasets. Curves show cumulative fraction of datasets achieving a given RMSE ratio (relative to the best configuration). Shaded regions indicate ±1 standard error across datasets. The neuron-adaptive curve rises fastest, indicating the most consistent performance improvements across tasks.

ANOVA shows adaptive types differ significantly from standard in most datasets. Tukey's test indicates neuron- and layer-adaptive types do not significantly differ in most cases[10].

The adaptive quadratic activation achieves substantial RMSE reductions relative to its standard counterpart. Expressed as a normalized improvement metric $(\text{RMSE}_{\text{standard}} - \text{RMSE}_{\text{adaptive}})/\text{RMSE}_{\text{standard}}$ averaged across all ten datasets, neuron-adaptive quadratic yields a mean improvement of 18.3% (±8.7%), with the largest relative gains observed on the GeoMusic and Wind datasets (28.5% and 29.2%, respectively). These percentage improvements are meaningful in the context of the datasets' respective error scales and reflect consistent gains across diverse domains. Adaptive PReLU and swish show smaller gains[11].

Neuron-specific adaptation uses more parameters but doesn't always yield better results. Sharing parameters across layers often suffices for regression problems.

For compact networks, layer-wide adaptation offers a good balance between performance and parameter efficiency, suitable for edge devices.

6. Conclusion and Future Work

This paper presented an evaluation framework for adaptive activation functions in feedforward networks for efficient regression. Results demonstrate that adaptive parameterization consistently improves performance across the ten datasets considered, with quadratic adaptive functions showing the most significant gains in this experimental setting. Both layer-wide and neuron-specific adaptations outperform static functions, with neuron-specific adaptation generally achieving the best results[12].

There are various possibilities for future research. First, extending evaluation to classification tasks would be valuable. Second, investigating more sophisticated adaptive functions (spline-based, piecewise linear) could enhance flexibility. Third, applying adaptive activations to deeper architectures and multimodal data (e.g., using systems like VDMS [13]) presents a natural extension. Fourth, integrating adaptive activations with other efficiency techniques like pruning, quantization, or tensor norm

optimization [14] could yield synergistic improvements. Finally, theoretical studies on approximation power and convergence guarantees are needed.

References

[1] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1026–1034, 2015.

[2] F. Agostinelli, M. Hoffman, P. Sadowski, and P. Baldi. Learning activation functions to improve deep neural networks. *arXiv preprint arXiv:1412.6830*, 2014.

[3] P. Ramachandran, B. Zoph, and Q. V. Le. Searching for activation functions. *arXiv preprint arXiv:1710.05941*, 2017.

[4] M. Garnelo, J. Schwarz, D. Rosenbaum, F. Viola, D. J. Rezende, S. M. A. Eslami, and Y. W. Teh. Conditional neural processes. In *Proceedings of the International Conference on Machine Learning*, pages 1704–1713, 2018.

[5] A. Veit, M. J. Wilber, and S. Belongie. Residual networks behave like ensembles of relatively shallow networks. In *Advances in Neural Information Processing Systems*, pages 550–558, 2016.

[6] G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew. Extreme learning machine: Theory and applications. *Neurocomputing*, 70(1-3):489–501, 2006.

[7] G.-B. Huang, H. Zhou, X. Ding, and R. Zhang. Extreme learning machine for regression and multiclass classification. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 42(2):513–529, 2012.

[8] D. R. Insua and P. Müller. Feedforward neural networks for nonparametric regression. In *Practical nonparametric and semiparametric bayesian statistics*, pages 181–193. Springer New York, 1998.

[9] J. Han, C. Moraga, and S. Sinne. Optimization of feedforward neural networks. *Engineering Applications of Artificial Intelligence*, 9(2):109–119, 1996.

[10] J. S. Pei and E. C. Mai. Constructing multilayer feedforward neural networks to approximate nonlinear functions in engineering mechanics applications. 2008.

[11] S. Scher and G. Messori. Generalization properties of feed-forward neural networks trained on lorenz systems. *Nonlinear processes in geophysics*, 26(4):381–399, 2019.

[12] J. S. Pei and A. W. Smyth. New approach to designing multilayer feedforward neural network architecture for modeling nonlinear restoring forces. i: Formulation. *Journal of engineering mechanics*, 132(12):1290–1300, 2006.

[13] O. Ladapo. Revolutionizing data preparation and access for visual and multi-modal business analytics. *Int. J. Sci. Adv. Technol.*, 16(2), 2025.

[14] M. Banik. Novel tensor norm optimization for neural network training acceleration. In *Proc. Int. Conf. Artif. Intell.*, 2025.