Optimizing DNN Model Architecture for Function Approximation

Hamza Akmal
24100232@lums.edu.pk
School of Science and Engineering
LUMS

December 14, 2023

Abstract

This paper presents an empirical study on the optimization of Deep Neural Network (DNN) architectures in TensorFlow for modeling complex functions, specifically aiming to determine the ideal combination of layers, neurons, and an extrinsic parameter 'a' for the function y = arcsin(ax) + sin(10x). The study explores the effects of varying the number of layers and neurons, along with three distinct values of the extrinsic parameter 'a', across twelve different datasets. Preliminary results are visualized using a DNN with two hidden layers and ten neurons with ReLU activation to fine-tune the number of epochs for subsequent analysis. The paper discusses the methodology for generating the datasets, the training process, and the implications of the findings for DNN architecture design in function modeling. A qualitative analysis of trends is presented followed by a rigorous quantitative analysis using covariance perturbation and random forest regressors to uncover the functional relationship between the parameters. The results prove the running hyptothesis that lowering the number of layers and increasing the number of neurons for a fully connected dense layer reduces the minimum convergent loss independent of the choice of function, activation or optimizer.

1 Introduction

Dense fully connected neural networks, also known as multilayer perceptrons (MLPs), are foundational constructs in the field of deep learning. Composed of successive layers of interconnected nodes or neurons, these networks are characterized by the full connectivity between the neurons of adjacent layers. Each connection, representing a synaptic weight, modulates the signal transmitted between neurons. The output of each neuron is computed as a weighted sum of its inputs followed by a non-linear activation function, formalized as:

$$h_i^{(l)} = \phi\left(\sum_j w_{ij}^{(l)} x_j + b_i^{(l)}\right) \tag{1}$$

where $h_i^{(l)}$ represents the output of the i-th neuron in the l-th layer, ϕ denotes the activation function, $w_{ij}^{(l)}$ are the weights connecting the j-th neuron of the $(l-1)$-th layer to the i-th neuron of the l-th layer, x_j are the inputs from the previous layer, and $b_i^{(l)}$ is the bias term for the i-th neuron.

The objective of the present study is to ascertain an optimal architecture for dense fully connected neural networks tasked with modeling the function $y = \arcsin(ax) + \sin(10x)$. The optimization process involves varying the number of layers and neurons within the network, as well as tuning an extrinsic parameter a. This parameter's influence on the network's ability to accurately represent the function is examined through a series of experiments. The datasets for these experiments are generated across two domain ranges with varying sample sizes, forming a comprehensive evaluation matrix to assess model performance against the complexity of the function.

2 Theoretical Background

The theoretical underpinnings of dense fully connected neural networks are deeply rooted in the approximation theory. The Universal Approximation Theorem posits that a feedforward network with a single hidden layer containing a finite number of neurons can approximate continuous functions on compact subsets of $\mathbb{R}^n$, under mild assumptions on the activation function (1; 2). Formally, given any $\epsilon > 0$ and any continuous function $f : \mathbb{R}^n \to \mathbb{R}$, there exists a neural network N with activation ϕ such that for all $x \in \mathbb{R}^n$,

$$|N(x) - f(x)| < \epsilon. \tag{2}$$

Extending this concept, recent studies have shown that deeper architectures can reduce the computational cost and the number of required neurons for achieving a desired level of approximation, thus highlighting the significance of depth in neural networks (3; 4).

In the context of the function $y = \arcsin(ax) + \sin(10x)$, the challenge lies in the network's ability to capture both the local variations induced by the $\sin(10x)$ term and the global behavior governed by the $\arcsin(ax)$ term. The former requires the network to have a high-frequency modeling capacity, while the latter demands a robust representation of inverse trigonometric functions. The choice of activation function ϕ and the network's architecture play a pivotal role in this dual objective.

The extrinsic parameter a introduces a scaling effect, influencing the amplitude and frequency of the arcsine component. This scaling can have a profound impact on the learning dynamics of the network and its eventual generalization performance. Theoretical exploration of such parameters is seen in the work of Goodfellow et al., which discusses the impact of data representation on deep learning (5).

The optimization of network architecture thus becomes a non-trivial task, requiring the balancing of model complexity against the risk of overfitting, a phenomenon well-documented by Bishop in (6). The trade-off between a model's capacity and its performance on unseen data is quantified by the notion of Vapnik–Chervonenkis (VC) dimension (7), which provides a theoretical bound on a model's generalization error.

In computational terms, the training of such networks is facilitated by backpropagation, an algorithm leveraging the chain rule to compute gradients efficiently (8). The advent of modern optimization techniques, such as Adam (9), further refines the learning process by adapting learning rates for individual parameters.

The present project, therefore, seeks to navigate this complex landscape by conducting a series of empirical experiments, the results of which are expected to contribute valuable insights into the design of neural network architectures for function approximation.

3 Experimental Procedure

3.1 Data Generation

The experimentation was initiated by generating synthetic datasets to evaluate the neural network's performance. Three distinct values for the extrinsic parameter a, namely 10, 2, and 1, were chosen to investigate their effects on the modeling capacity of the networks. For each value of a, a dataset consisting of 1000 samples ($m = 1000$) was generated within the domain $x \in [-1, 1]$. The synthetic datasets were produced by uniformly sampling points in the given domain and computing the corresponding output y using the function:

$$y(x) = \arcsin(\min(\max(ax, -1), 1)) + \sin(10x) \tag{3}$$

The function encompasses both an inverse trigonometric function arcsin and a periodic function sin, with the latter possessing a frequency tenfold that of the input signal. The arcsin function is bounded within $[-1, 1]$, thus requiring the use of a clipping operation to ensure the argument lies within this interval.

This procedure was encapsulated within a function `generate_data`, which accepts the parameter a, the domain of x, and the sample size m as inputs and returns a dataset comprising input-output pairs (x, y). This function was iteratively applied for each chosen value of a to compile a list of datasets, which were then utilized to train and evaluate the neural network models.

Here we present the visualization of the generated data for different values of the extrinsic parameter a within the domain $x \in [-1, 1]$ and with a sample size of $m = 1000$.

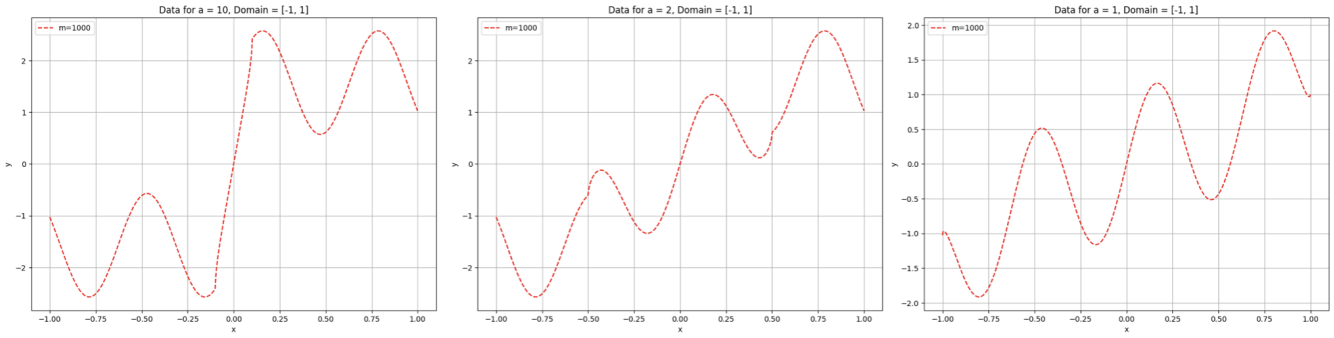


Figure 1: Generated data for $a = 10$, $a = 2$, and $a = 1$ with domain $[-1, 1]$.

The subsequent stages of the experimentation will involve configuring various neural network architectures, training them using the generated datasets, and analyzing their performance based on loss metrics and generalization capabilities. Each model's ability to learn and predict the underlying function will be rigorously assessed, providing insights into the interplay between neural network complexity and function approximation accuracy.

3.2 Standardized Training Procedure

Upon generating the datasets, we established a standardized procedure for training the neural network models. This procedure is delineated as follows:

1. The number of epochs was set to 200 for all runs during Step 2 of the experimentation phase and increased to 400 for runs in Step 3 to ensure model convergence. We determined the adequacy of these epochs by observing the loss stabilization point at approximately 400 epochs during preliminary trials.
2. The number of neurons n in each layer was varied across the set {1,2,4,8,16,32,64,128,256}, with the corresponding number of layers L fixed at 4 to analyze the effects of network width on learning capabilities.
3. Similarly, the number of layers L was varied across the set {1,2,4,8,16,32,64,128}, while maintaining the number of neurons per layer n at 4, to investigate the impact of network depth.
4. Converged loss values were plotted on the y-axis against $\log(n)$ or $\log(L)$ on the x-axis to visually assess the relationship between network architecture complexity and the minimization of loss.
5. Three datasets were visualized using pairs of plots: one plotted against $\log(n)$ and the other against $\log(L)$, combined with three different activation functions to compare performance across varying network configurations.

This training protocol was meticulously followed to ensure consistent and comparable results across all neural network architectures. The plots generated during this process serve as an empirical basis to deduce the optimal structure for the neural networks in question.

The findings from these standardized training runs are expected to provide a comprehensive understanding of the interplay between the number of layers and neurons, as well as the selection of activation functions, in the context of function approximation tasks.

4 Neural Network Model Performance

The following figures were generated using a Python script employing Keras, a high-level neural networks API, to create and train neural network models with various numbers of neurons and activation functions. The training dataset is represented by x_{train} and y_{train}, and was used to fit models with different architectures. Seaborn, a statistical data visualization library, was utilized for styling the plots. We will discuss the plots for a = 10 here while the plots for the other datasets can be found in Table 1 and Table 2.

4.1 Actual Data vs Model Prediction

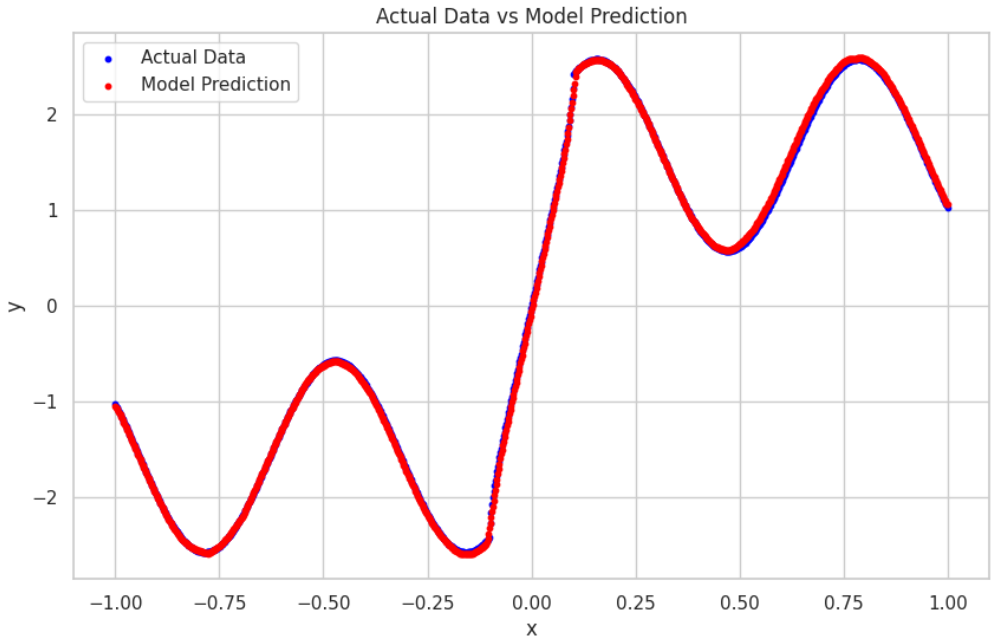


Figure 2: This plot shows the scatter of actual data against the predictions made by the best-performing model. The blue dots represent the actual data points, while the red dots represent the model's predictions. The close alignment between the two sets of points indicates the model's accuracy in predicting the underlying function of the dataset.

Minimum Loss vs. Number of Neurons

The model with the lowest loss across all configurations was saved, and its predictions were compared against the actual dataset to evaluate its performance. The process of training involved 200 epochs for each model configuration, with a callback function providing updates every 100 epochs. The loss function used was the mean squared error, which is a common choice for regression problems.

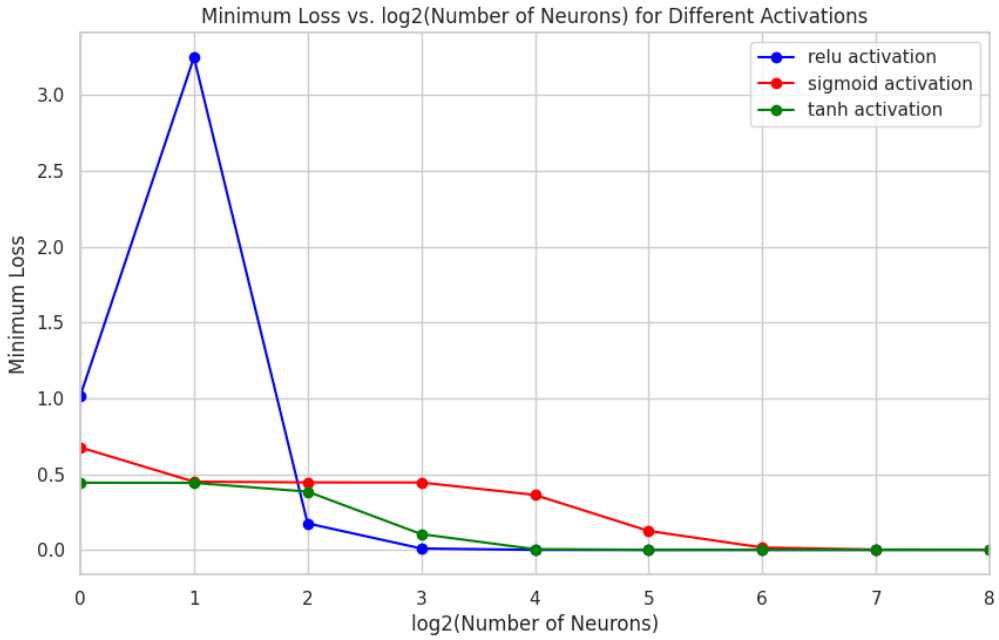


Figure 3: This plot illustrates the relationship between the minimum loss achieved and the number of neurons in the model, on a logarithmic scale. Three different activation functions were tested: ReLU (blue), Sigmoid (red), and Tanh (green). As the number of neurons increases, the minimum loss generally decreases, indicating improved model performance. However, the performance varies with different activation functions.

Minimum Loss vs. Number of Layers

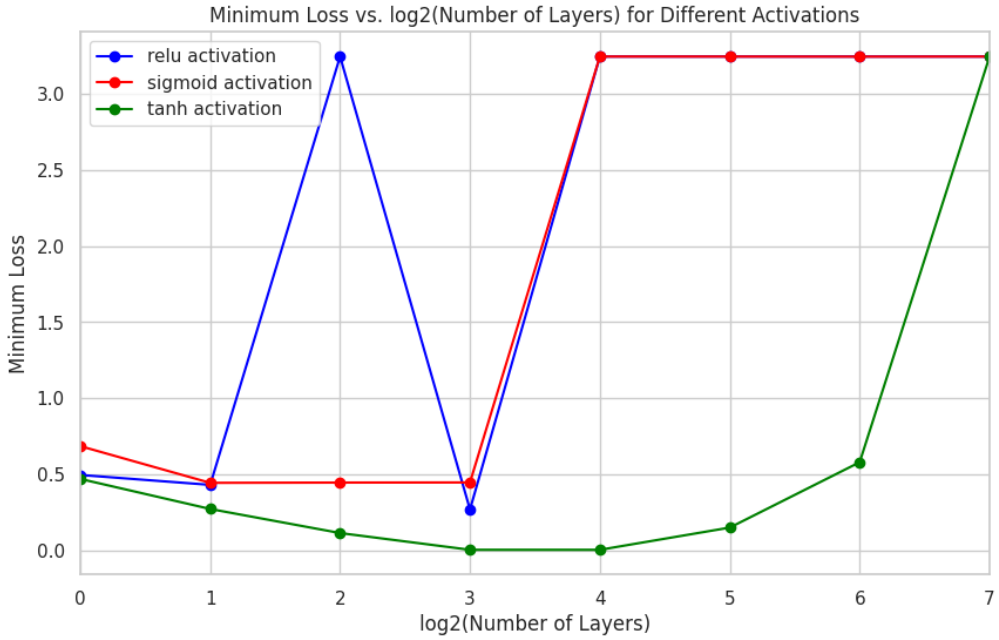
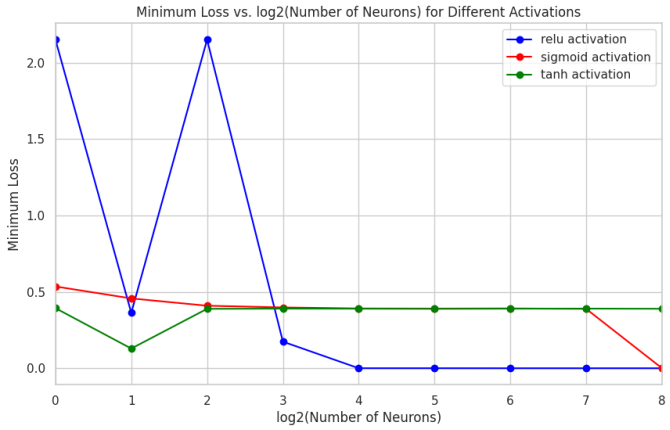
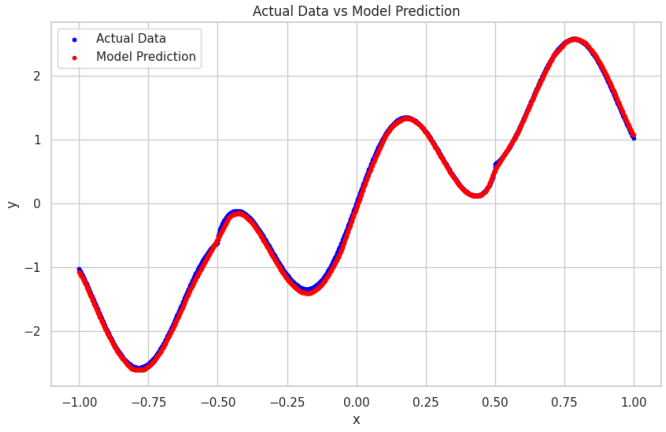


Figure 4: This plot demonstrates the minimum loss values obtained for neural network models with varying numbers of layers, using a logarithmic scale for the layer count. The models were evaluated using three different activation functions: ReLU (blue), Sigmoid (red), and Tanh (green). The graph shows how the complexity of the model affects its performance, with certain numbers of layers performing better for specific activation functions. This visualization helps in understanding the optimal architecture for the neural network depending on the activation function used.

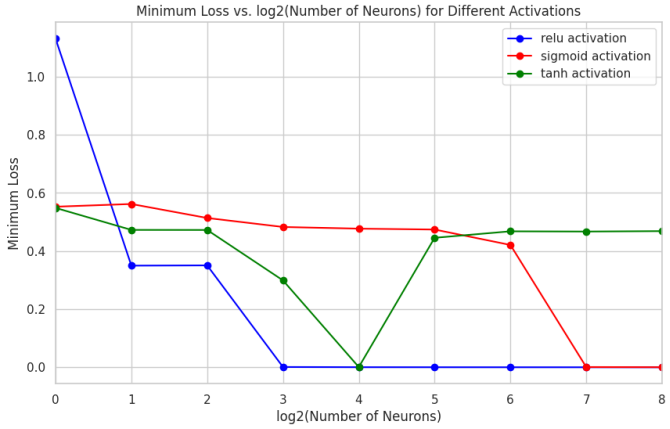
The graph provides insight into the depth of the neural network architecture, highlighting the impact of the number of layers on the model's loss performance. Notably, the models exhibit different behaviors as the number of layers changes, suggesting that the optimal number of layers for minimizing loss may depend on the chosen activation function. This could be due to overfitting or underfitting at various levels of model complexity.



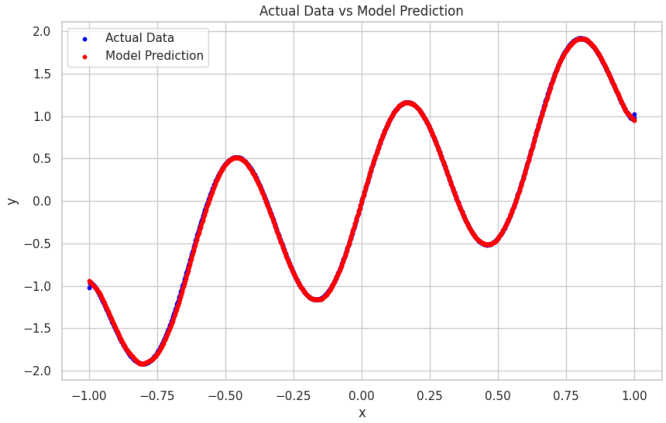
(a) Minimum Loss vs. neurons for a = 2



(b) Data vs Model Prediction for a = 2

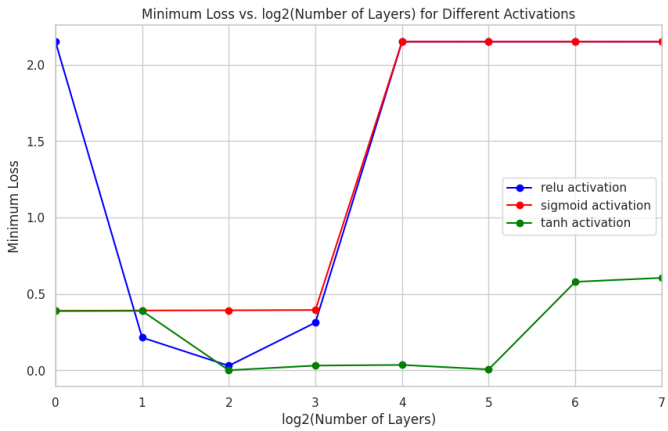


(c) Minimum Loss vs. neurons for a = 1

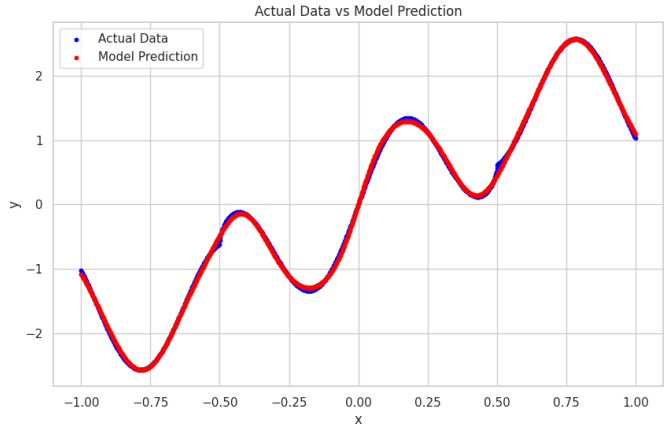


(d) Data vs Model Prediction for a = 1

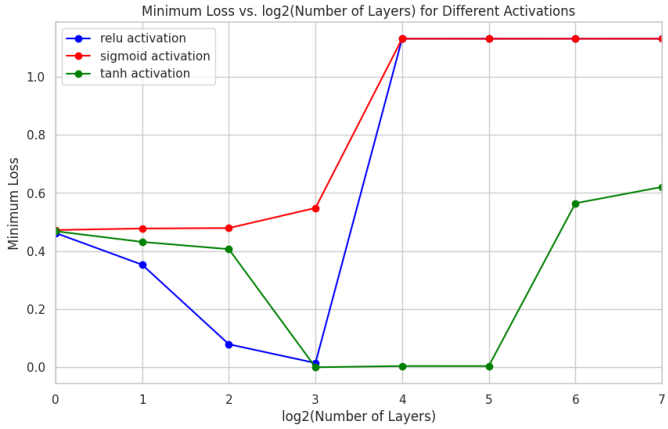
Table 1: Comparison of minimum loss and model predictions for different neuron counts and activation functions with parameter a set to 2 and 1.



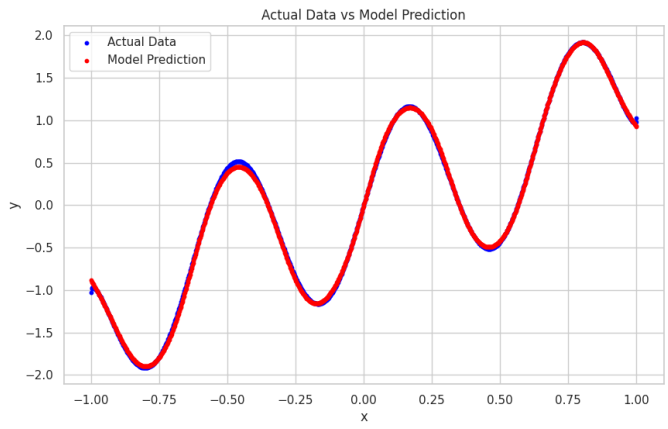
(a) Minimum Loss vs. layers for a = 2



(b) Data vs Model Prediction for a = 2



(c) Minimum Loss vs. layers for a = 1



(d) Data vs Model Prediction for a = 1

Table 2: Comparison of minimum loss and model predictions for different layer counts and activation functions with parameter a set to 2 and 1.

4.2 Qualitative Analysis of Neural Network Performance

The analysis of Table 1 and Table 2 suggests a nuanced relationship between the neural network's architecture and its performance in approximating a given function. The figures depicting loss as a function of the number of neurons, n, and layers, L, suggest that there exists an optimal complexity that minimizes loss. Specifically, the figures indicate that both underfitting and overfitting may occur: underfitting when the model is too simple (few neurons and layers) and overfitting when the model is excessively complex (too many neurons and layers).

The parameter a, which scales the input data, seems to affect the function's shape that the network aims to learn. The variation in a changes the difficulty level of the learning problem, which is reflected in the loss values observed. For $a = 2$ and $a = 1$, we can infer that the neural network's ability to generalize improves as the model complexity increases to a point, beyond which the model may start to overfit.

Mathematically, if $f(x)$ is the function to be approximated and $\hat{f}(x;\theta)$ is the neural network's

prediction with parameters θ, the mean squared error (MSE) loss function L is given by:

$$L(\theta) = \frac{1}{N}\sum_{i=1}^{N}(f(x_i) - \hat{f}(x_i;\theta))^2, \tag{4}$$

where N is the number of data points. The goal of training is to find θ that minimizes L.

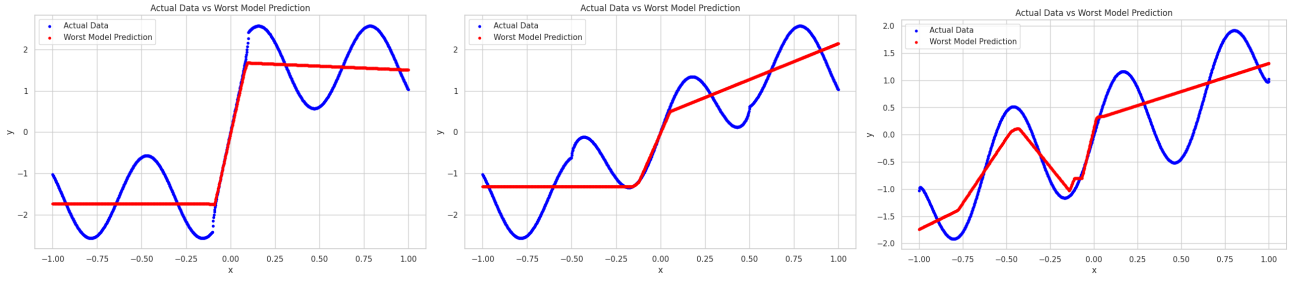
The model's predictions versus actual data plots serve as a visual representation of this loss. Ideally, the points representing the model's predictions should coincide with the actual data points. Discrepancies between these indicate prediction errors, which are captured quantitatively by the loss function.

4.3 Analysis of Divergent Model Predictions

The provided plots for n = 200 and L = 50 reveal the performance of a neural network model when the training process does not lead to convergence. Specifically, these plots compare the actual data with the worst model predictions starting after 200 epochs for different values of the parameter a. The first plot, corresponding to $a = 10$, exhibits significant deviations between the model predictions and the actual data, indicating a failure to converge. Similarly, the plots for $a = 2$ and $a = 1$ demonstrate substantial discrepancies, with the predicted values diverging from the expected results. Such divergence is typically a symptom of a model that is either overfitting or underfitting, or one that has not been adequately trained to capture the complexity of the underlying function.

The erratic behavior observed in the worst model predictions is characterized by sharp, unrealistic changes in the predicted values, which are not present in the actual data. This phenomenon can occur due to several factors, including but not limited to:

- An excessively high learning rate that overshoots the optimal points in the loss landscape.
- A large number of neurons and layers that give the model excessive capacity, leading to overfitting.
- Insufficient regularization, which might allow the model to focus too much on the noise within the training data.
- Inadequate training duration, meaning the model has not had enough iterations to learn the patterns effectively. Refer to the appendix for more examples of diverging plots.



(a) for $a = 10$ (b) for $a = 2$ (c) for $a = 1$

Figure 5: Actual Data vs Worst Model Prediction

4.4 Model Performance with Different Optimizers

We have evaluated the performance of four different optimizers: SGD, Adagrad, RMSprop, and Adam. Figures 6, 7, and 8 on the next page show the comparison of actual data versus model prediction, and the minimum loss achieved during training for different numbers of neurons.

4.4.1 Stochastic Gradient Descent (SGD)

SGD is a simple yet often effective approach to optimization. As shown in Figure 6a, the minimum loss decreases as the number of neurons increases, indicating that a larger network capacity can lead to better learning. However, Figure 6b reveals that the model's predictions are not always in close agreement with the actual data, suggesting that SGD may require careful tuning of learning rates and may benefit from momentum to achieve better convergence.

4.4.2 Adaptive Gradient Algorithm (Adagrad)

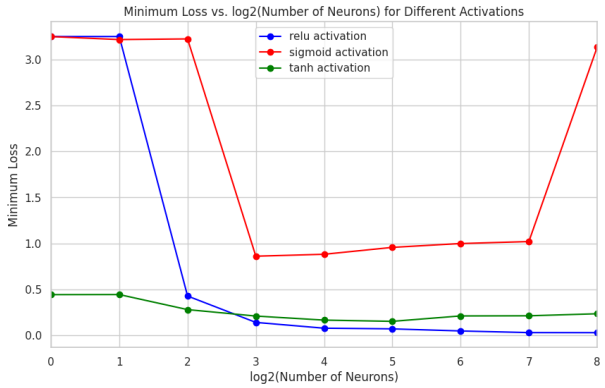
Adagrad adapts the learning rate to the parameters, performing smaller updates for frequent parameters and larger updates for infrequent parameters. From Figure 7a, Adagrad demonstrates a smoother convergence in loss compared to SGD, especially for a smaller number of neurons. The model predictions in Figure 7b are more aligned with the actual data, indicating better generalization.

4.4.3 Root Mean Square Propagation (RMSprop)

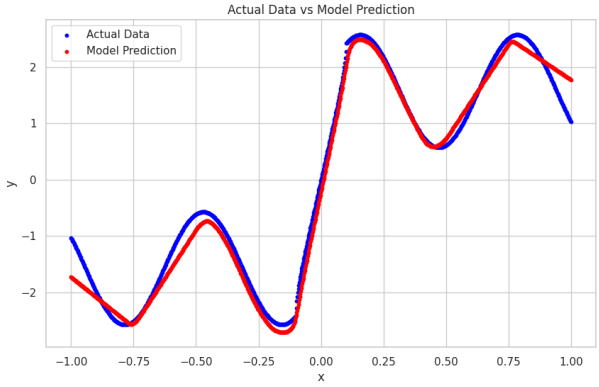
RMSprop builds upon Adagrad's approach by also considering the moving average of the squared gradients to adapt the learning rate. As illustrated in Figure 8a, RMSprop exhibits a stable decrease in loss with an increase in neurons, with no significant spikes in loss. The predictions in Figure 8b fit the actual data well, suggesting that RMSprop effectively captures the underlying function of the data.

4.4.4 Adaptive Moment Estimation (Adam)

Lastly, the Adam optimizer combines the advantages of Adagrad and RMSprop. As demonstrated in Figures 3 and 2, Adam consistently achieves lower minimum losses across different neuron counts and layers, with model predictions that closely match the actual data points. The adaptive learning rate mechanism of Adam, which adjusts rates based on the moving average of the gradients, proves to be particularly effective in handling the complexities of the model, leading to superior convergence and generalization capabilities.

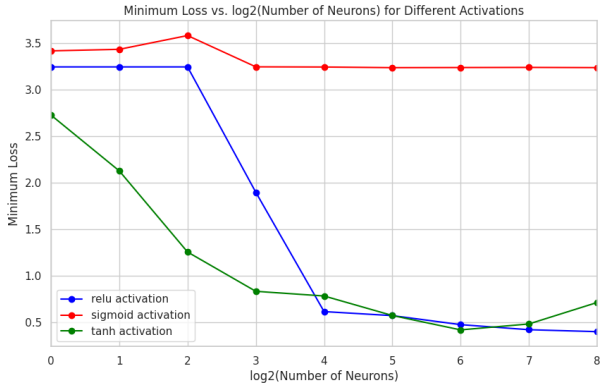


(a) Min loss vs. no. of neurons using SGD.

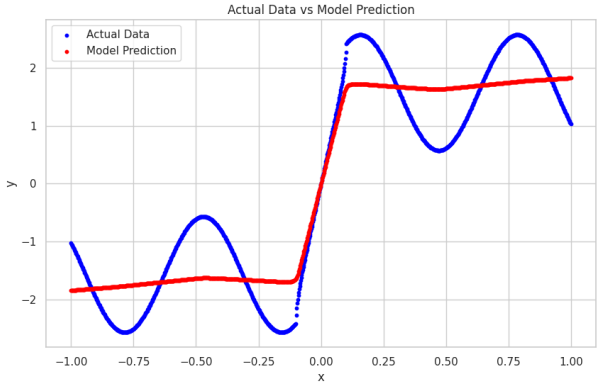


(b) Data vs. model prediction using SGD.

Figure 6: The performance of the model using Stochastic Gradient Descent (SGD) optimizer.

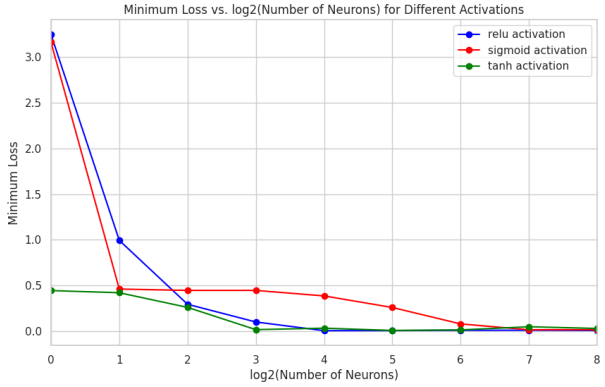


(a) Minimum loss vs. no. of neurons using Adagrad.

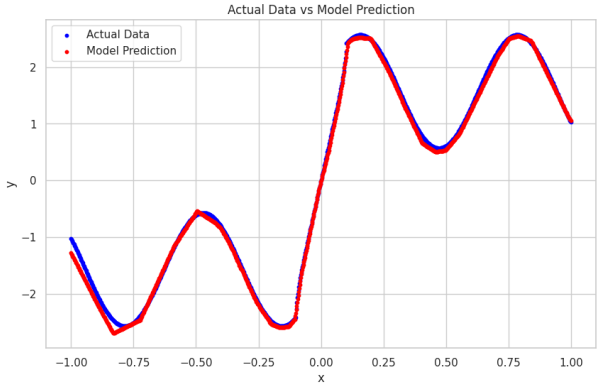


(b) Data vs. model prediction using Adagrad.

Figure 7: The performance of the model using the Adagrad optimizer.



(a) Minimum loss vs. no. of neurons using RMSprop.



(b) Data vs. model prediction using RMSprop.

Figure 8: The performance of the model using the RMSprop optimizer.

5 Data Analysis of Functional Relationship

In our study, we have compiled a comprehensive dataframe by augmenting data from our function with data from other arbitrary sinusoidal functions, such as $y = \sin(ax)\exp(-ax^2)$. This dataframe includes various parameters like activation function, number of layers, minimum loss, number of neurons, and the parameter a. The head of the dataframe is presented in Figure 9, which provides a snapshot of the data collected for preliminary analysis.

Our objective is to process this data to uncover the underlying functional relationship between the number of neurons, number of layers, the parameter a, and the minimum loss achieved during the training of our models. By examining these relationships, we aim to optimize our neural network architecture for better performance and efficiency.

	activation	layers	min_loss	neurons	a
0	relu	1.0	0.495541	4.0	10.0
1	relu	2.0	0.430382	4.0	2.0
2	relu	4.0	3.246183	4.0	1.0
3	relu	8.0	0.268542	4.0	10.0
4	relu	16.0	3.246211	4.0	2.0
5	relu	32.0	3.246192	4.0	1.0
6	relu	64.0	3.246133	4.0	10.0
7	relu	128.0	3.246196	4.0	2.0
8	sigmoid	1.0	0.684928	4.0	1.0
9	sigmoid	2.0	0.443691	4.0	10.0
10	sigmoid	4.0	0.445436	4.0	2.0
11	sigmoid	8.0	0.446376	4.0	1.0
12	sigmoid	16.0	3.246108	4.0	10.0
13	sigmoid	32.0	3.245961	4.0	2.0
14	sigmoid	64.0	3.245559	4.0	1.0
15	sigmoid	128.0	3.246044	4.0	10.0

Figure 9: Snapshot of the head of the dataframe containing the collected data.

5.1 Model Evaluation and Feature Importance

In our analysis, a random forest regressor was employed to assess the significance of different features in the model. The feature importances derived from the regressor provide insights into the influence of each feature on the model's predictive capability.

The model's performance was quantified using the mean squared error (MSE), which amounted to 11%. This metric indicates the average squared difference between the estimated values and the actual value, with a lower MSE representing a better fit of the model to the data.

The functional relationship between the features and the minimum loss was determined through covariance analysis. The relationship is encapsulated by the following equation:

$$\text{min\_loss} = -0.1159 - 0.1848 \times \text{no. of neurons} + 0.4741 \times \text{no. of layers} + 0.1787 \times a \tag{5}$$

This equation represents a linear model where the minimum loss is a function of the number of neurons, the number of layers, and the parameter a. The coefficients of the features reflect their respective contributions to the prediction of minimum loss, with the sign indicating the direction of the relationship.

5.2 Visualizing Feature Relationships

To further understand the relationship between neurons, layers, and minimum loss, a heatmap was constructed, as shown in Figure 10.

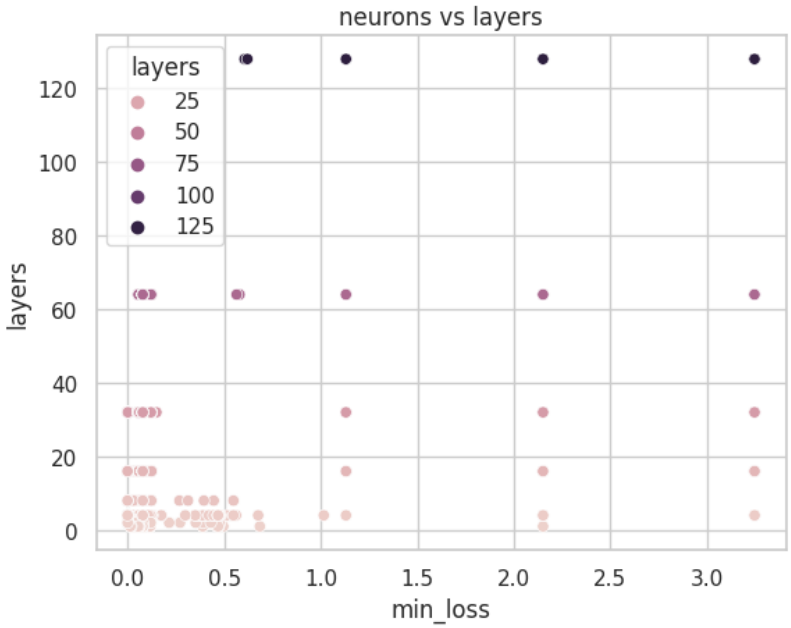


Figure 10: Heatmap of neurons versus layers with respect to minimum loss. The color intensity represents the number of layers, with darker shades indicating a higher number of layers. This visualization aids in discerning the combined effect of neurons and layers on the model's minimum loss.

The heatmap provides a graphical representation of data where the individual values contained in a matrix are represented as colors. In our case, the 'minimum loss' is plotted on the x-axis, and the 'layers' are represented on the y-axis. The color intensity corresponds to the number of neurons, allowing us to observe clusters of higher or lower losses associated with varying numbers of neurons and layers. The darker the color, the greater the number of layers, which suggests a complex relationship where both neurons and layers affect the loss outcome of the model.

5.3 Pairwise Relationships Analysis

In this section, we delve into the pairwise relationships between the number of neurons, layers, and the parameter a, with respect to the minimum loss observed during model training. Figure 11 showcases these relationships through a pairplot.

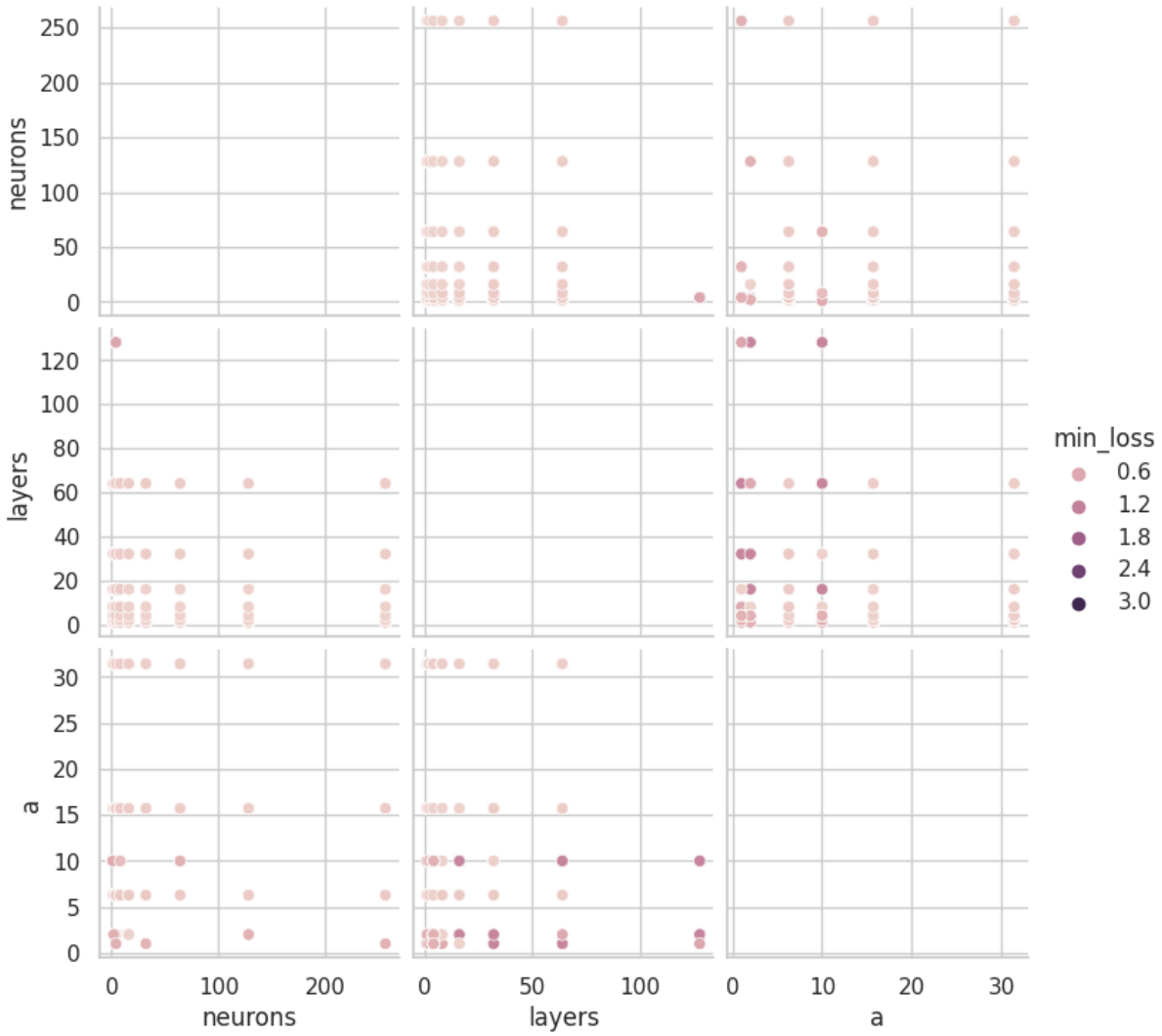


Figure 11: Pairplot showing the pairwise relationships between neurons, layers, and parameter a against the minimum loss. The size and color intensity of the dots correspond to the magnitude of the minimum loss, illustrating the multidimensional relationship between these model parameters.

The pairplot provides a comprehensive view of the possible correlations between the model's parameters. The distribution of points across the plots suggests varying degrees of sensitivity of the minimum loss to changes in neurons, layers, and the parameter a. Notably, we observe that:

- The relationship between the number of neurons and minimum loss appears to be nonlinear, with certain neuron counts correlating to both high and low losses.
- As the number of layers increases, there seems to be a threshold beyond which the minimum loss does not significantly improve, indicating potential overfitting or capacity limitations.
- The parameter a shows a complex interaction with the other parameters, where its effect on minimum loss is not immediately discernible without considering neurons and layers simultaneously.

5.4 Correlation Matrix Analysis

The correlation between the different features of our model is depicted in the correlation matrix shown in Figure 12. This matrix provides valuable insights into how each feature potentially influences the minimum loss achieved by the model.

Figure 12: Correlation matrix displaying the correlation coefficients between minimum loss, number of layers, number of neurons, and the parameter a. The color intensity and sign of each cell indicate the strength and direction of the correlation.

The values in the correlation matrix range from -1 to 1, where 1 indicates a perfect positive correlation, -1 indicates a perfect negative correlation, and 0 implies no correlation. Notable observations from the matrix include:

- A moderate positive correlation between the number of layers and the minimum loss, suggesting that as the complexity of the model increases, so does the loss, up to a certain point.
- A negative correlation between the number of neurons and minimum loss, as well as between the parameter a and minimum loss, hinting that these features inversely affect the loss.
- The correlation between neurons and layers is slightly negative, which may indicate that an increase in one does not necessarily compensate for the other in reducing loss.

These correlations can be mathematically expressed as part of a multiple regression model, which helps to predict the minimum loss based on the values of layers, neurons, and parameter a:

$$\text{min\_loss} = \alpha + \beta_1 \cdot \text{layers} + \beta_2 \cdot \text{neurons} + \beta_3 \cdot a$$

Here, α represents the intercept, and β_i terms represent the coefficients that are calculated based on the training data to minimize the prediction error. Each coefficient reflects the expected change in the minimum loss for a unit change in the corresponding feature, holding all other features constant.

Performing multiple linear regression on our augmented dataset now yields the following functional relationship:

$$\epsilon = -0.00071405 \cdot \text{neurons} + 0.00460649 \cdot \text{layers} - 0.01020218 \cdot a$$

6 Discussion

In the exploration of neural network optimization, the derived relationship

$$\epsilon = -0.00071405 \cdot \text{neurons} + 0.00460649 \cdot \text{layers} - 0.01020218 \cdot a$$

captures the influence of architectural parameters on the model's performance. Here, ϵ represents the minimum loss achieved during training, which is the metric of interest when attempting to minimize prediction error.

Firstly, the coefficient of the *neurons* term is negative, indicating that increasing the number of neurons tends to reduce the loss, possibly due to a more robust capacity to capture complex patterns. However, the small magnitude of this coefficient suggests a diminishing return on additional neurons, which is consistent with the understanding that excessively large networks are prone to overfitting and may not generalize well to unseen data.

Conversely, the positive coefficient for *layers* implies that additional layers may increase the loss, reflecting the complexity and potential over-parameterization of the network. This might lead to an increased risk of not converging or learning spurious patterns that do not generalize beyond the training set.

The negative coefficient for a, which may represent a complexity parameter of the function being approximated (e.g., frequency in a sinusoidal function), indicates that higher complexity in the target function can lead to lower loss, possibly due to the network leveraging additional neurons or layers to approximate complex functions more accurately.

To optimize the loss ϵ, one might consider a balance between the number of neurons and layers, aiming to increase the neuron count while carefully regulating the number of layers to prevent unnecessary model complexity and overfitting.

Given that this equation is derived from datasets encompassing a variety of functions, it suggests a degree of generalizability. The equation implies that while specific architectures may be optimal for certain function types, there exist common trends in how architectural changes impact model performance across different types of data (eg: neurons 16, layers 8).

In designing neural network architectures based on this equation, one should approach with caution, as the generalizability of these results may not hold in every domain or for every function type. It is often advisable to validate the architecture empirically with cross-validation techniques on the specific task at hand. Future research into investigating this relationship would entail sampling a diverse family of functions and achieving a generalizable architectural schema.

Appendix

A Code and Datasets

The complete code, datasets and modular functions and regression analysis is available under the MIT License at this link.

References

[1] Hornik, K., Stinchcombe, M., & White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5), 359-366.

[2] Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems*, 2(4), 303-314.

[3] Eldan, R., & Shamir, O. (2016). The power of depth for feedforward neural networks. In *Conference on Learning Theory*, 907-940.

[4] Telgarsky, M. (2016). Benefits of depth in neural networks. In *29th Annual Conference on Learning Theory*, 49, 1517-1539.

[5] Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep learning. *MIT press*.

[6] Bishop, C. M. (2006). Pattern recognition and machine learning. *springer*.

[7] Vapnik, V. (2013). The nature of statistical learning theory. *Springer science & business media*.

[8] Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, 323(6088), 533-536.

[9] Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.