SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY - INFORMATICS

Technische Universität München

Master's Thesis in Informatics

Comparison of training techniques in neural architecture search

Kushal Dhungana

SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY - INFORMATICS

Technische Universität München

Master's Thesis in Informatics

Comparison of training techniques in neural architecture search

Vergleich von Trainingsmethoden bei der Architektursuche von Neuronalen Netzwerken

Author:	Kushal Dhungana
Thesis Supervisor:	Dr. Felix Dietrich
Submission Date:	15.11.2023

I confirm that this master's thesis is my own work and I have documented all sources and material used.

15.11.2023 Kushal Dhungana

Acknowledgments

I would like to express my deepest gratitude and appreciation to my committee chair, my supervisor Dr. Felix Dietrich for his consistent support and guidance during this thesis. I am indebted for his unwavering support through scheduled meetings, prompt responses, and continuous encouragement. This accomplishment would not have been possible without him.
Thank you.

Abstract

Neural Architecture Search has been gaining popularity lately because it offers automation in designing neural networks and can deliver customized neural architectures for specific requirements. Although neural architecture search is expensive and takes a long time, it is able to search through a larger space of architectures, eventually delivering the best architecture for a given task. We compare the result of two training techniques, the iterative gradient-based method (Adam) and the sampling method (SWIM), in neural architecture search for approximating different functions and try to understand whether deep learning is relevant for multi-layer perceptrons. For most cases, we saw that a single hidden layer with more neurons is sufficient for approximation using SWIM; however, Adam requires more layers and decent neurons for a similar approximation. The results show that the SWIM method is comparable, if not better, to the iterative optimization method for approximation experiments, and the time to obtain a trained network is one to two orders of magnitude faster using SWIM than using Adam.

Contents

1 Introduction

Recently, there has been a lot of advancement and interest in the field of artificial intelligence and machine learning. Artificial intelligence means creating a system having intelligence similar to humans, and machine learning means the capability of machines to learn things and knowledge from data fed to it and apply that knowledge to other new things they observe. Machine learning is one of the subfields of artificial intelligence. Nowadays, machine learning has been applied in a wide range of fields, including Medical diagnosis, Automation, Natural Language Processing, Forecasting, Recommendation Systems, Speech Recognition, etc. There has been a lot of research going on in the machine learning topics to get meaning out of complex data, which introduces the need for deep learning, one of the subfields of machine learning. [13]

Deep learning is a field which deals with having more than one hidden layer. This field has gained a lot of attention from machine learning researchers as deep learning is able to interpret complex and large amounts of data, and it has evolved in a way that resembles our brain learning pattern. Some of it's use cases are image recognition, object detection, autonomous driving, etc. Although we know that deep learning is beneficial to learn complex patterns in data and to make machines do what humans can easily do, it has it's limitations. Mainly, the limitation is about the time deep network takes to train and the computational resources needed to perform deep learning. To address the computation time part, a sampling method introduced by [2] performs deep learning with comparable accuracy. It would be beneficial if the computational time is highly reduced in the deep learning scenario with good accuracy. However, it is still unclear whether deep learning is necessary for regression or classification tasks and whether it is relevant for multi-layer perceptrons. Some research [18] shows that shallow networks are sufficient to approximate certain functions when supplied with enough neurons, while other research ([16], [1]) claims that deep networks are needed to achieve better results. In this thesis, we are interested in knowing whether deep or shallow networks produce better results when approximating certain functions, which we find out using neural architecture search.

Neural Architecture Search (NAS) has gained a lot of popularity lately as it automates the design of neural networks, and some of the neural architecture generated by NAS ([20], [3]) has shown better performance than manually designed architecture [17]. When we talk about Neural Architecture Search, search space and search strategies are it's core components. In this thesis, we use NAS to find the best neural network architecture with number of layers and number of neurons as network parameters in the search space. Furthermore,

a grid search is used for neural architecture searches because we want to compare and plot the results of all search space configurations. The goal is to eventually find out if deep networks are essential for regression tasks and compare the result (in terms of error and computation time) of approximating different functions using two training methods (SWIM and Adam).

This thesis is divided into three sections. The first section is the state-of-the-art section where the theoretical background of the relevant topics such as Neural Architecture Search, Iterative gradient methods, Shallow and Deep neural networks, and Sampling weights of deep neural network [2] are discussed. Similarly, the second section is where the methodology used for the experiments, evaluation criteria, and results of the experiments are discussed. Finally, the third section is the conclusion section. It is divided into three sub-sections: Summary, Discussion, and Outlook. In the summary section, the summary of the findings is discussed; in the discussion section, the relevance of the thesis and findings is discussed; and in the outlook section, the challenges and the room for future work of the thesis are discussed.

2 State of the art

2.1 Neural Architecture Search

Neural Architecture Search is a technique to automatize the process of finding the best neural architecture suitable for a given classification/regression task. Lots of research is being conducted on NAS topics to design an efficient algorithm to automatize the neural network design process for a particular task. Lately, NAS has gained a lot of popularity after the paper [19] was released in 2017 [17]. To point out the numbers, almost 1000 papers have been released in the year 2020-2022 [17].

There are a lot of limitations when it comes to designing neural network architecture by humans. The overall neural network architecture design process is very time-consuming for humans. Unlike machines, humans cannot deliver high accuracy consistently. One more limitation is that humans are unlikely to explore all the search space, which, as a result, can result in missing better solutions. Furthermore, to design the architecture, one has to have a good grasp of the field. Without the proper knowledge about the problem at hand, hyperparameters, and in-depth neural network knowledge, one wouldn't be able to design the best architecture for the problem.

To some extent, NAS is able to solve the above problems. NAS is able to search through all the search spaces and come up with the best architecture. Additionally, it is much faster after the first setup as it automates the neural design process compared to hand-designing the network.

Neural Architecture Search has three main core components: Search Space, Search Method, and Performance Estimation [17]. A search space is a space that contains all the possible configurations that the NAS algorithm can explore to find the best configuration. It is essential to carefully design the search space because it has a huge influence on the result architecture you get. While choosing the search space, one should keep in mind about it's size. If one chooses a small handpicked search space, the influence of humans would be high on this compared to choosing a large search space, which can result in finding a new and better architecture, though it might take a longer time to search. [17]

A search method is an algorithm used to search over the search space. Choosing the search method is one of the crucial steps in neural architecture search. It determines how efficiently the algorithm is able to search in the search space and come up with the best

architecture design. [17]
There are different types of search methods used for NAS. Some of the search methods are:

- Evolutionary methods,
- Reinforcement learning,
- Grid search,
- Random search,
- Gradient-based methods.

The neural architecture search method used in our experiments was grid search (exhaustive search).

Grid Search
The grid search method is a method which is very popularly used for hyperparameter optimization [10]. Grid search searches sequentially through the predefined hyperparameter values to find the best parameter values. Moreover, grid search can be considered one of the baseline approaches for neural architecture search. One of the reasons to use this in our experiments was we wanted to find the result for each configuration and plot it in a 2D heatmap graph. Moreover, grid search also guarantees that one will find the best parameters. However, one of the challenges of grid search is it is computationally expensive when the search space is large. In our case, the search space is not that large, but if the search space is large and one wants to use grid search, then one can make the use of parallelization. [10]

Performance estimation is a strategy used to get an idea of the architecture's performance beforehand without fully training the architecture. This strategy saves us a lot of time as we can discard the bad architecture without fully training it. [17]
Some evaluation metrics used to judge the quality of the best architecture discovered by NAS methods are accuracy, mean squared error, F-1 score, etc. In our case, we are using mean squared error. Although there are a lot of benefits to using Neural Architecture Search, there are also challenges. Some of them are:

- Computationally expensive:
  Conducting a neural architecture search requires large computational resources as it has to train and evaluate many neural architectures. This takes a lot of time specially when the search space is large.
- Not up to the par as hand-designed architecture:
  NAS-generated neural network architecture is still not able to fully beat hand-designed architectures in all the tasks. There are only a few examples where NAS beats hand-designed architecture.

- Search Space:
  The search space of neural architecture search is generally large and complex, making it slower and computationally expensive.

2.2 Iterative gradient-based training method

Before understanding the iterative gradient-based training method, we must first understand what the gradient-based training method is. The gradient-based method is a method that is used to find the global optimum of a target function by going in the direction of a global optimum. When it is done iteratively, it is called the iterative gradient-based training method. In each iteration, parameters like weight, biases, and learning rates are changed based on the loss function value. There are different types of gradient-based optimizer:

- Gradient descent (GD)
  Gradient descent is an iterative optimization algorithm which is used to minimize a differentiable function by adjusting the parameters in the negative direction of the gradient of the function with respect to the parameters. The speed at which the descending is done to reach the minimum is determined by the step size (learning rate). [12]

- Stochastic gradient descent (SGD)
  As gradient descent takes many iterations to reach the minimum of the objective function, the idea of stochastic gradient descent was introduced. The idea of stochastic gradient descent is to randomly take one point from the dataset for each iteration to calculate the derivative instead of using all the data points. This makes computation much faster; however, the chances of convergence are a bit less compared to gradient descent, and there is a chance of overshooting as SGD updates the parameters frequently with a high step size. Additionally, the trade-off between speed and convergence rate can be managed by decreasing the learning rate as we approach the minimum of the objective function. [12]

- Adam
  Adam was introduced by this paper [9] and it is used for training deep neural networks. It combines the advantage of two important algorithms, AdaGrad [4] and RMSProp [14]. AdaGrad delivers good performance even with sparse gradients, and RMSProp is well-suited for online scenarios [9]. Adam utilizes the advantage of momentum and learning rate decay to converge faster. Moreover, Adam is faster than SGD as it converges faster. Although the faster convergence part of Adam was not focused on non-convex problems, the authors observe that Adam outperforms other methods in such scenarios. The results are better while using Adam to optimize multi-layer neural networks containing non-convex functions. [9]

In our setup, we are using Adam as an optimizer. We made this choice because it is one of the state-of-the-art optimizer methods. Additionally, it is faster and more robust compared to other methods.

2.3 Deep Neural Networks

2.3.1 Multi layer Perceptron (MLP)

Multi-layer Perceptron is one type of artificial neural network. It contains an input layer, one or more hidden layers, and an output layer. It contains neurons that are connected between layers. The neurons in one layer are fully connected to neurons in another layer, so it is also called a fully connected network. MLPs are used to learn the patterns in data. The data flows from the input layer to the output layer in a feed-forward network fashion. Each connection has a weight associated with it, which is adjusted to minimize the loss. The authors in [8] write about multi-layer perceptron being a universal approximator. Furthermore, MLP can approximate any Borel measurable function given enough hidden neurons [8]. Moreover, the structure of the multi-layer perceptron can be seen in figure 2.1. In the figure, a fully connected neural network with one hidden layer is shown. Each node in one layer is connected to nodes in the next layer and the previous layer. The input layer has three nodes, the hidden layer has three nodes, and the output layer has one node, respectively. In the figure, the blue colour represents input layer nodes, the orange colour represents hidden layer nodes, and the green colour represents output layer nodes.

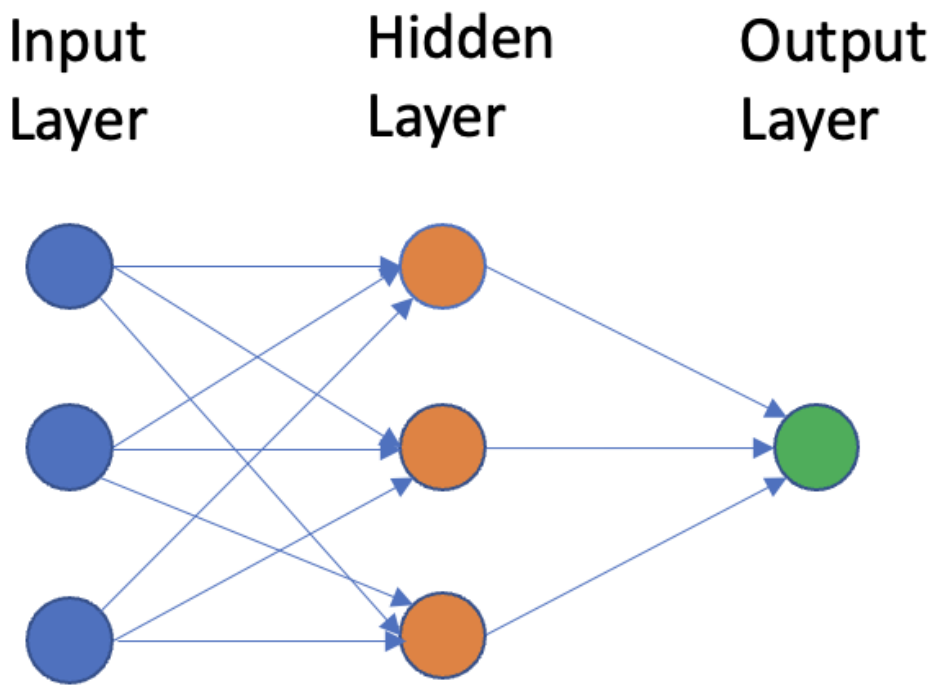

Figure 2.1: A fully connected network with one hidden layer. Each neuron in one layer is connected to neurons in the next layer and the previous layer.

2.3.2 Convolutional Neural Network (CNN)

MLP models suffer when the dimensionality of inputs increases; as a result, the convolutional neural network was introduced. For example, if we have a coloured image of input dimension 128*128, then the weights in the first neurons of the first layer would be 128*128*3, which equals 49,152 [11]. Thus, it is computationally costly to use MLPs to train such high-dimensionality inputs. The convolutional neural network basically consists of three types of layers, i.e. convolutional layers, pooling layers, and fully connected layers [11]. Feature extraction is performed in the convolutional layers, and dimensionality is reduced in the pooling layer, which helps to compute the result faster. A basic CNN architecture can be seen in the figure 2.2. [11]

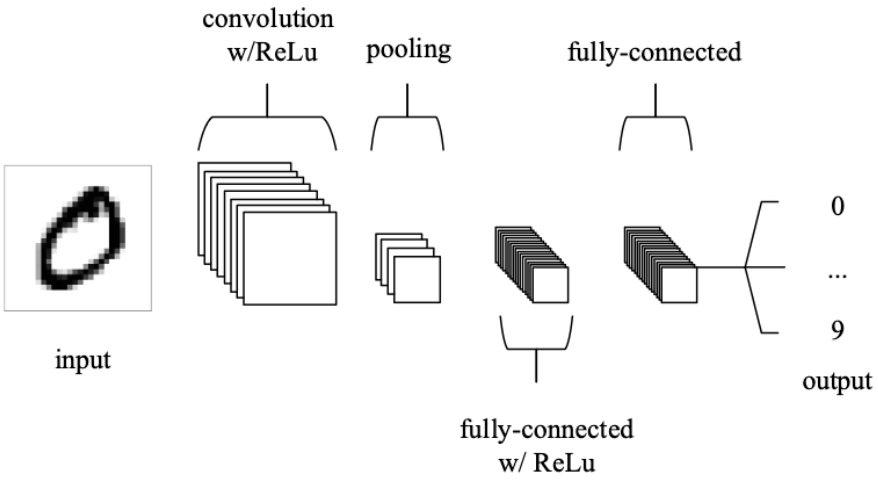

Figure 2.2: A basic CNN architecture. Figure taken from [11]

Residual Neural Network

Residual Neural Network was introduced in paper [7]. As networks are trained deeply, they suffer from vanishing/exploding gradients problem. A residual Network was introduced to address this problem. The architecture contains a concept called skip connections. As the name suggests, skip connections mean connecting the activation of a certain layer to other layers by skipping some layers in the middle. The residual block of a residual neural network can be seen in figure 2.3. The concept of this network is to fit the residual function instead of allowing the layer to learn the underlying mapping where H(x) is the underlying mapping that is equal to F(x)+x and H(x)-x is the residual function. [7]

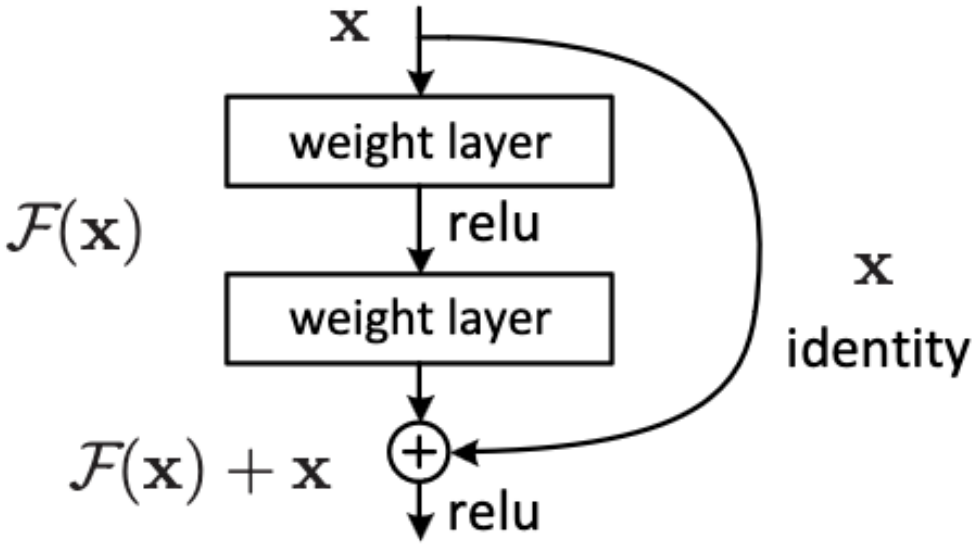

Figure 2.3: Basic blocks of Residual Neural Network;Residual Block. Figure taken from [7]

2.3.3 Transformer

Transformer was introduced by paper [15]. Transformer is heavily used in natural language processing. It makes use of self-attention mechanism to determine the importance of different sections of the input data and establish a relationship between them. As the transformer allows parallelization, it is easier and faster to train. Furthermore, it also solves the vanishing and exploding gradient problem, which allows for deep learning. [15]

2.3.4 Deep and shallow networks

Shallow networks are networks with fewer hidden layers; ideally, one layer (called single-layer perception), and deep networks are networks with more than one hidden layer. The main question of this thesis is whether deep learning is relevant for MLPs. Some papers point out that deep networks are better than shallow networks, and others claim the reverse.

One of the paper [18] conclude that shallow multi-layer perceptrons with 1 or 2 hidden layers with a large number of neurons outperform deeper multi-layer perceptrons. The authors compare five classical supervised classification methods (Linear Discriminant Analysis, Multinomial Logistic Regression, Naïve Bayes, Random Forest, Support Vector Machine), CNN and shallow and deep MLPS to detect diseased samples from normal sam-

ples as well as to determine the disease stages in 37 datasets [18]. The authors observed that MLPs (specially single hidden layers with a large number of neurons) outperform all the other methods [18].

Another paper [16], compares shallow MLPs with deep MLPs for identifying and categorising ransomware. The authors found that deeper MLPs with higher layers and higher neurons give the best detection and classification results [16]. One thing to consider here is the neuron number hasn't been kept fixed while comparing shallow and deep MLPs, so it can be said that it might have influenced the result.

One more paper [1] compares the accuracy of deep MLPs and shallow MLPs to predict the bearing capacity of shallow foundations. In this paper, the number of neurons is fixed while comparing the shallow and deep MLPs. The authors found that a network with 5-7 layers gives the best result while training on scarce datasets. It is observed that the number of layers has a high influence on accuracy compared to the number of neurons. [1]

The paper by Weinan E. [5] discusses the distance function 3.2. The author says that this function is not a Barron function in itself but is a composition of two Barron functions. As the function is a composition of two Barron functions, the author says that it can be represented by a neural network with exactly two hidden layers, which can easily be checked theoretically, but practically, the author does not say anything about the approximation by a finite neural network. So, in experiment one, we try to approximate the distance function with a finite neural network.

2.4 Sampling weights of deep neural networks

The paper [2] introduces a sampling algorithm for weights and biases of multi-layer perceptrons along with a probability distribution. All of the ideas in this section are from this paper. Instead of randomly sampling the weights and biases, the authors sample the weights and biases that are near the gradient of the target function. The visual representation of how the weights are chosen in random feature models and this sampling method can be seen in figure 2.4. One doesn't need to perform iterative parameter adjustments to train the network here. As a result, training a network is a magnitude faster using this method than training using an iterative gradient-based method. [2]

This sampling method also solves three problems that limit random feature models to compete with iterative optimization methods. One of them is that this sampling method allows for deep learning and constructing deep neural networks, unlike random feature models, which typically have only one hidden layer. The second one is that the sampling method provides accurate solutions with a decent number of neurons, unlike random fea-

ture models that need a higher number of neurons to give a comparable result. The third one is this sampling method samples weights and biases near the gradient of the target function by using the weight construction technique [6], which allows to get the solution faster as data are sampled where it matters the most. The sampling distribution introduced is invariant to scaled input data and orthogonal transformations, which in turn makes the pre-processing step superfluous. [2]

In this sampled network, two points in the input space define every combination of weight and bias in every hidden layer. The weight is defined by the disparity of two points in the input space, and the bias is defined by the inner product between the weight and one of the two points. Once the weight and bias are constructed, the next step is to optimize the coefficients of a linear layer, which maps the output from the last hidden layer to the final output. [2]

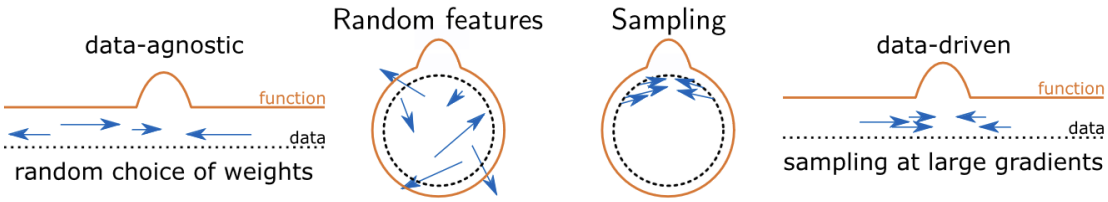

Figure 2.4: Illustration of how weights are chosen in random feature model vs how it is chosen using this sampling method. Figure taken from [2].

The result obtained with this method is comparable to that obtained using the iterative gradient method and is more accurate than random feature models [2]. As training even the deep network is magnitude faster than the iterative method, this method has a lot of potential. Furthermore, this paper [2] provides a detailed mathematical framework and swim algorithm explanation.

One of the limitations currently is that only fully connected network architecture can be sampled using this method. More advanced networks like convolution and transformers have not yet been tested using this method [2].

3 Comparison of training techniques in neural architecture search

Neural Architecture Search is a topic which has been gaining a lot of popularity in recent years. The fact that it automates the neural network design process and is able to produce tailored neural architecture for specific tasks is the main reason it is gaining such popularity. In this thesis, we worked with multi-layer perceptrons for neural architecture search and tried to understand if deep learning is relevant for the MLPS.

This section is divided into five sub-sections. In the first section, the motivation behind this thesis is discussed. In the second section, the evaluation metrics used are discussed. In the third section, the datasets used for different experiments are discussed. In the fourth section, the experimental setup of the experiment is discussed. Finally, in the fifth section, the result of the experiment is discussed.

3.1 Motivation

The main research question of this thesis is whether deep learning is relevant for MLPs. The thesis also compares and tries to find out if deeper networks are better than shallow networks or the other way around. In addition, this thesis compares the results of using Adam and SWIM methods for neural architecture search in MLPs for different function approximations. It also discusses whether the result obtained from the SWIM network is comparable to the result obtained from Adam and in what aspect is one better than the other.

3.2 Evaluation metrics

The experiments performed were all regression tasks. Therefore, the evaluation metric used was mean squared error (MSE).
Mean squared error is an evaluation metric which measures the average squared difference between predicted value and actual (true) value. The corresponding formula can be seen in 3.1.

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{3.1}$$

where n is the number of data points, y_i is the actual value of i-th data point and $\hat{y}_i$ is the predicted value of i-th data point.

3.3 Data Preparation / Datasets used

Four experiments were performed for this thesis paper. In the first experiment, we created 20000 two-dimensional random data points between -2 and 2 using a uniform distribution. This dataset was used as an input domain to approximate distance function 3.2. In the second experiment, we created 20000 one-dimensional random data points between 0.001 and 1 using a uniform distribution. This dataset was used as an input domain to approximate the sin function variation 3.3 In the third experiment, we have again created 20000 two-dimensional random data points between -2 and 2 using a uniform distribution. This dataset was again used as an input domain to approximate two decomposed distance functions 3.4 and 3.5 and to create a chained network. Finally, in the fourth experiment, we have created 40000 one-dimensional random data points between -1 and 1 using uniform distribution. This dataset was used as an input domain to approximate function 3.6.

There was no preprocessing involved in the datasets. All data were created using numpy random seed zero to ensure reproducibility. Data were created utilizing the python numpy random uniform method functionality that allowed the creation of a defined number of random data points between a defined number range that followed a uniform distribution.

3.4 Experimental Setup

3.4.1 Hardware and Software Specifications

All the experiments were performed on a MacBook Air M1,2020 laptop containing Apple M1 chip with 8GB RAM and only on CPU. Python 3.10.11 was used to execute the code. For the experiment using Adam, keras libraries, numpy, matplotlib pyplot and sklearn model selection libraries were used. Meanwhile for the experiment using SWIM, numpy, sklearn pipeline, swimnetworks, matplotlib pyplot and sklearn model selection libraries were used.

3.4.2 Methodology

We have performed four experiments. The first one was the approximation of the distance function:

$$f(x) = |1 - ||x||_{l2}|. \tag{3.2}$$

The visualization of function 3.2 can be seen in the figure 3.1

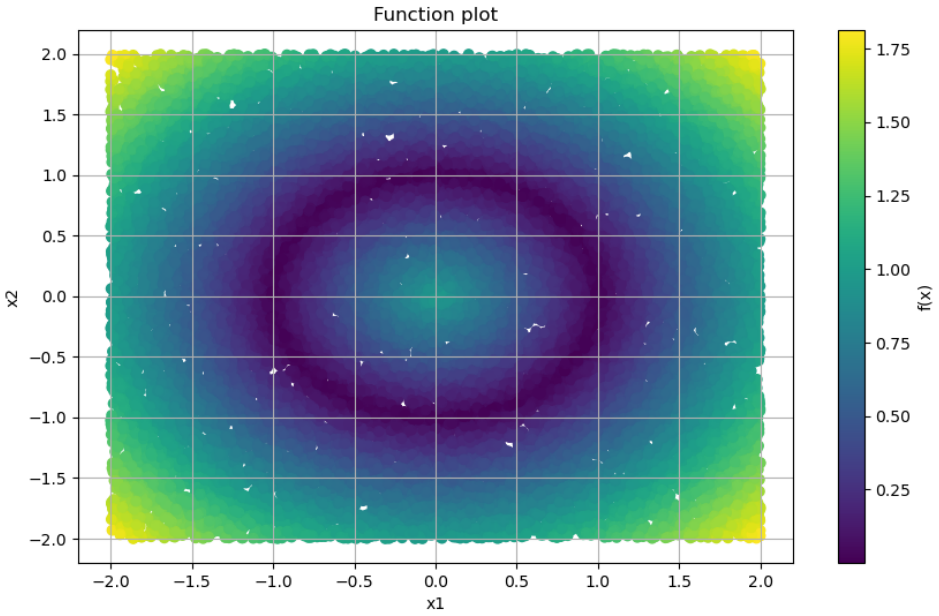

Figure 3.1: Plot of function 3.2

The second experiment was the approximation of the following sin function variation:

$$f(x) = \sin\left(\frac{1}{x}\right). \tag{3.3}$$

The visualization of the sin function variation 3.3 can be seen in the figure 3.2

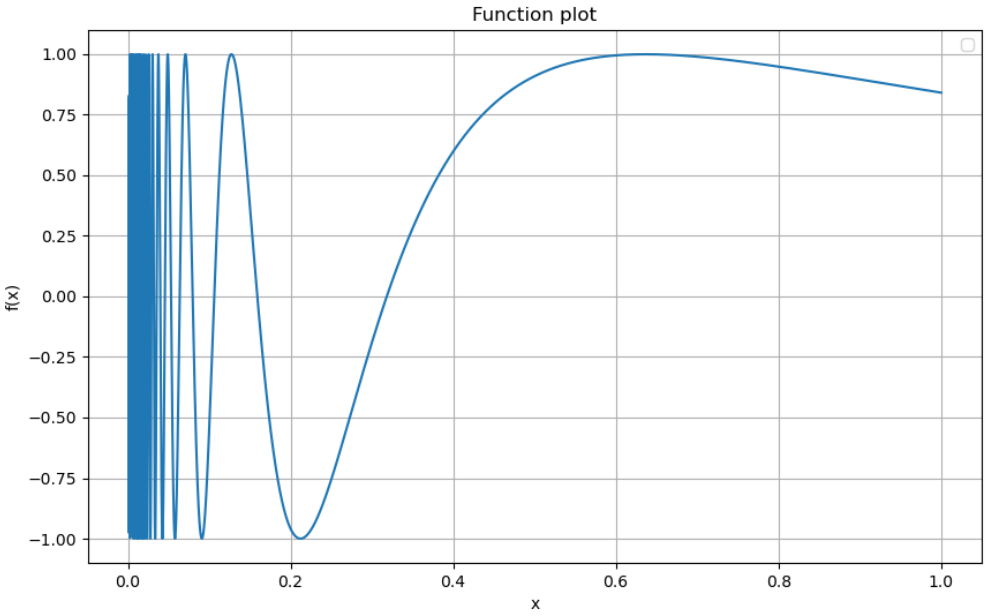

Figure 3.2: Plot of function 3.3

The third experiment was the distance function 3.2 was decomposed into two functions 3.4 and 3.5:

$$f(x) = ||x||, \tag{3.4}$$

$$g(x) = |1 - ||x||. \tag{3.5}$$

These functions were approximated separately using the SWIM method. The two separately trained networks were then chained and evaluated. Finally, the result of this chained network was compared with experiment one, i.e. an approximation of the function 3.2 using a single network. The visualization of function 3.4 can be seen in the figure 3.3.

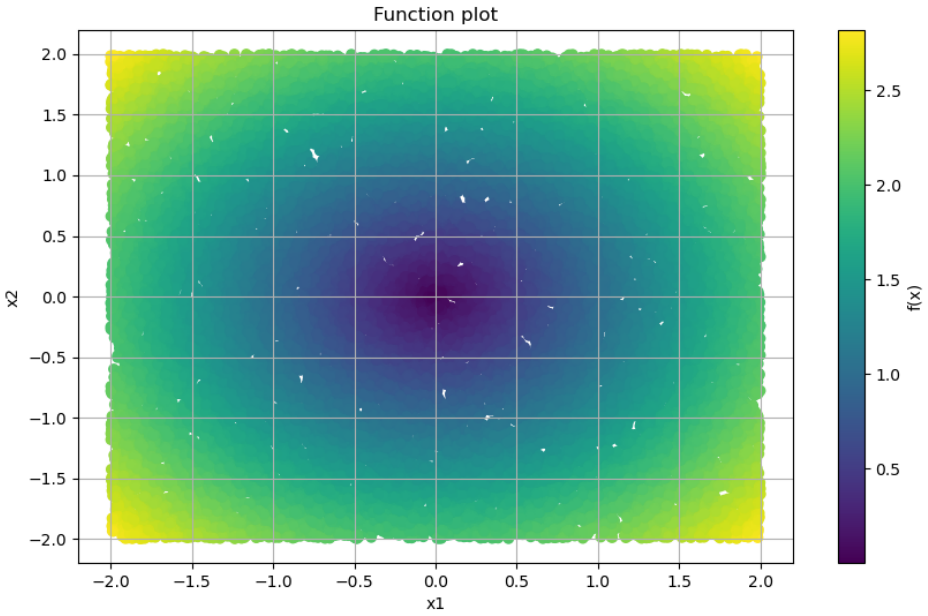

Figure 3.3: Plot of function 3.4

The visualization of function 3.5 can be seen in the figure 3.4.

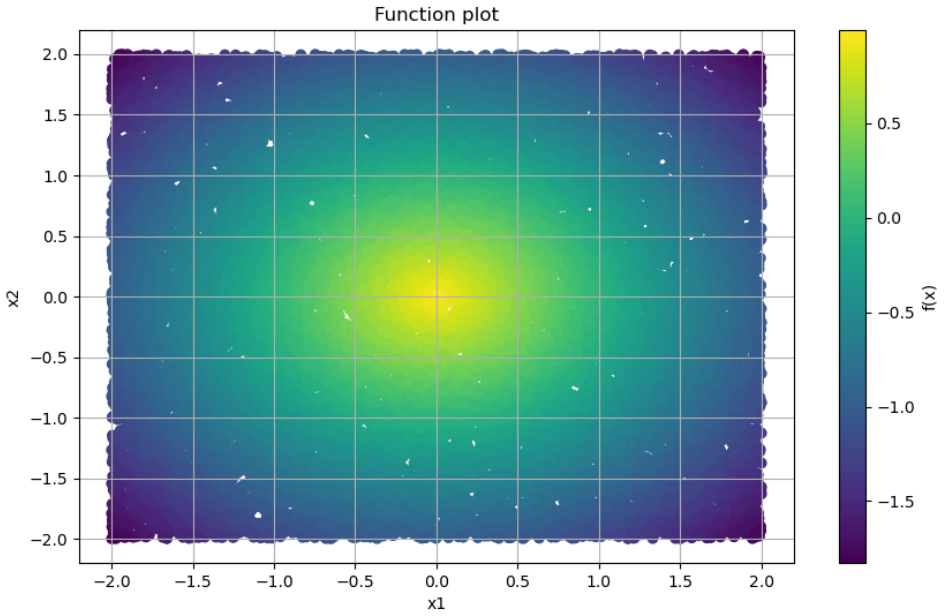

Figure 3.4: Plot of function 3.5

Finally, the fourth experiment was to approximate the function 3.6:

$$f(x) = \exp\left(-\left(\frac{x}{\sigma}\right)^2\right), \tag{3.6}$$

where, σ = np.power(10,linspace(-3,-1,10). The visualization of the function 3.6 can be seen in the figure 3.5

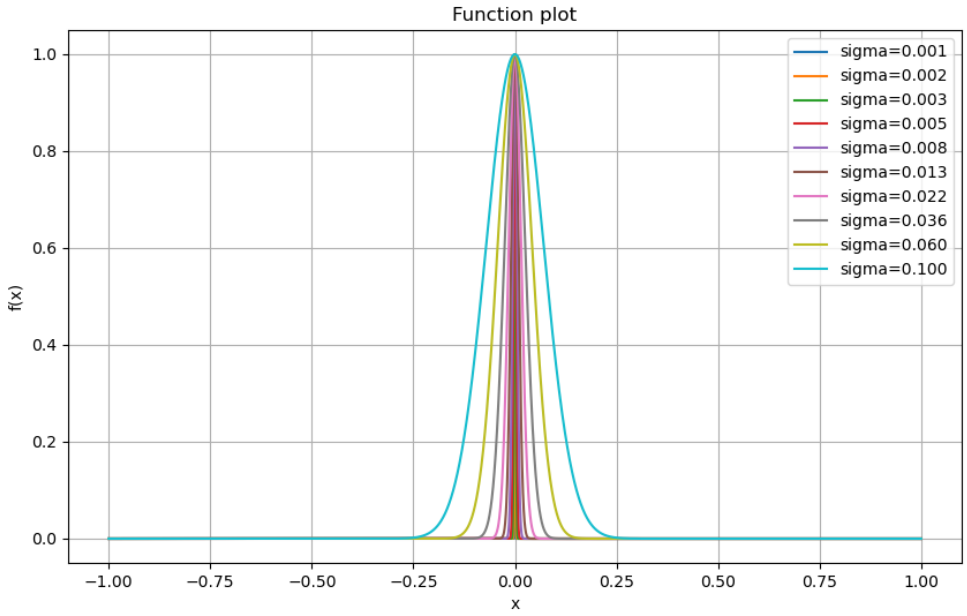

Figure 3.5: Plot of function 3.6

In all of the experiments, we have performed a grid search over the number of layers and number of neurons to find the best neural architecture. The best architecture would be the architecture with the configurations yielding the lowest mean squared error. The reason for choosing the above search space was to find out whether shallow or deeper MLPs yield the best results.

Experiments one, two and four were conducted using both Adam and SWIM methods, respectively, whereas experiment three was performed only using SWIM. Relu was used as an activation function for both Adam and SWIM experiments. 30 number of epochs was used for training in the Adam experiments. The number was chosen because, past that point, the loss function was not decreasing. In all the experiments except experiment three, cross-validation of five was used. The regularization in sampling was set to e^{-8}. The results of the experiments obtained using the two training techniques were compared.

3.5 Results

This result section is divided into four parts explaining four experiments conducted. The first two parts and the last part are subdivided into the results obtained from Adam and SWIM. In the third part, the result obtained from chained networks is compared with the

result obtained from a single network.

3.5.1 First Experiment

In the first experiment, distance function 3.2 is approximated using Adam and SWIM.

Table 3.1: Search space for number of layers and number of neurons in experiment one.

Method	Number of layers	Number of neurons
Adam and SWIM	1,2,3	200,400,600
Adam+	1	1000
SWIM+	1,2,3	100,200,300,400,500,600,700,800,900,1000

Adam

Firstly, we trained the model using Adam on the self-created dataset. Number of layers and number of neurons were used as search parameters in search space, and grid search was used to find the best architecture. The number of layers in the search space was set from one to three, and the number of neurons in the search space was set to 200, 400 and 600, which can be seen in table 3.1. The 2D heatmap plot of all the configurations where the number of layers is in the X-axis, the number of neurons is in the Y-axis, and the mean squared error (log-scale) is shown using a colour map as well as 2D heatmap of number of layers vs number of neurons vs fit time for all configurations can be seen in figure 3.6.

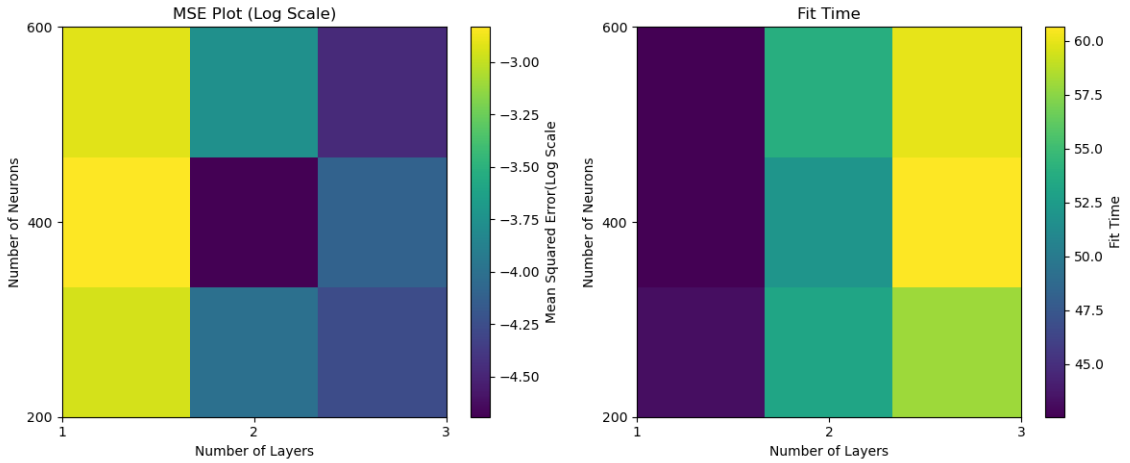

Figure 3.6: 2D heatmap plot of all the configurations where the number of layers is in the X-axis, the number of neurons is in the Y-axis and mean squared error (log-scale) is shown using colour map (left). 2D heatmap of the number of layers vs the number of neurons vs fit time (as colour map) for all configurations (right). This heatmap plot is created using Adam for experiment one.

In figure 3.6, we could see that two layers with 400 neurons give the lowest mean squared error result. The best architecture configuration with mean squared error value and total fit time for this experiment using Adam can be seen in the table 3.2.

SWIM

Secondly, we conducted the same experiment with SWIM. The number of layers in the search space was set from one to three, and the number of neurons in the search space was set to 200, 400 and 600, which can be seen in table 3.1. The corresponding 2D heatmap plot can be visualized in figure 3.7.

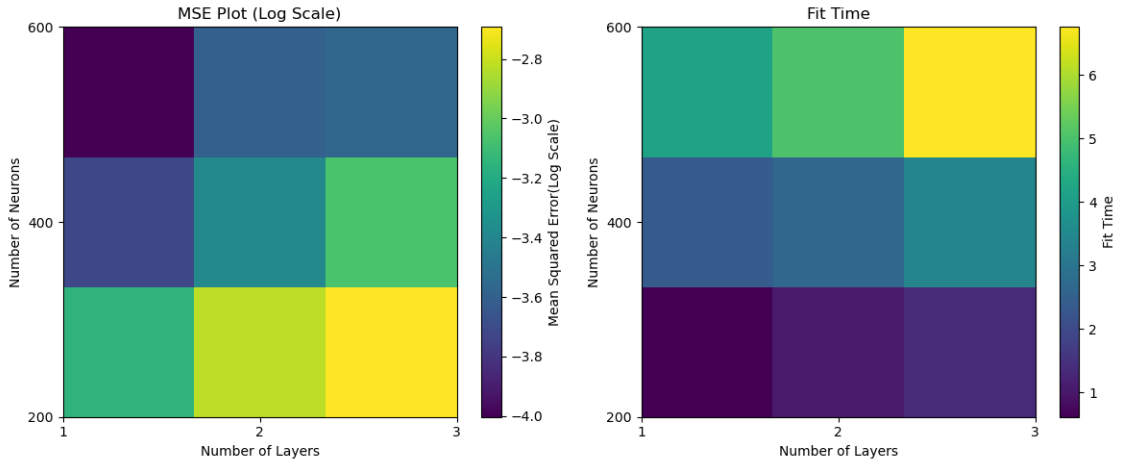

Figure 3.7: 2D heatmap plot of all the configurations where the number of layers is in the X-axis, the number of neurons is in the Y-axis and mean squared error (log-scale) is shown using colour map (left). 2D heatmap of the number of layers vs the number of neurons vs fit time (as colour map) for all configurations (right). This heatmap plot is created using SWIM for experiment one.

In the graph 3.7, we can see that one layer with 600 neurons gives the least mean squared error result. The best architecture configuration with mean squared error value and total fit time for this experiment can be seen in table 3.2. Again, we conducted the same experiment with SWIM but with a higher number of neurons represented by SWIM+. The number of layers in the search space was set from one to three, and the number of neurons in the search space was set from 100 to 1000 in 100 intervals, which can be seen in table 3.1. The corresponding 2D heatmap plot can be visualized in figure 3.8.

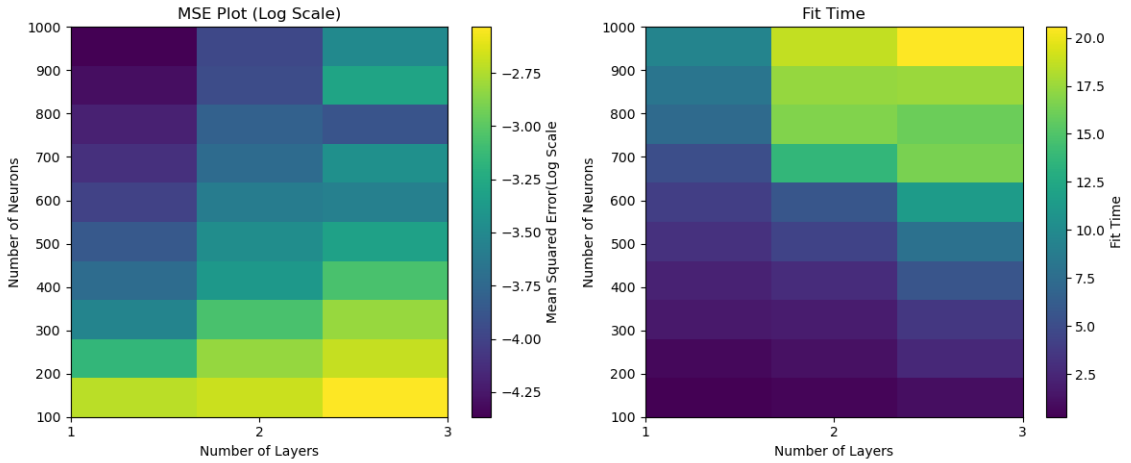

Figure 3.8: 2D heatmap plot of all the configurations where the number of layers is in the X-axis, the number of neurons is in the Y-axis and mean squared error (log-scale) is shown using colour map (left). 2D heatmap of the number of layers vs the number of neurons vs fit time (as a colour map) for all configurations (right). This heatmap plot is created using SWIM for experiment one with a higher number of neurons represented by SWIM+.

In the graph 3.8, we can see that one layer with 1000 neurons gives the least mean squared error result. Additionally, we tried to train using Adam with one layer and 1000 neurons, represented by Adam+, to compare it with the SWIM+ result. The best architecture configuration with mean squared error value and total fit time for this experiment can be seen in table 3.2.

Table 3.2: Best architecture configuration with mean squared error value and total fit time for experiment one. * Sign in the fit time for Adam+ indicates the fit time for the best architecture, not the total fit time. The lowest mean squared error and lowest total fit time are marked in bold.

Method	Number of layers	Number of neurons	Mean squared error	Fit time(s)
Adam	2	400	$\mathbf{2.03 * 10^{-05}}$	466.01
SWIM	1	600	$9.92 * 10^{-05}$	**27.38**
Adam+	1	1000	$5.77 * 10^{-04}$	105.86*
SWIM+	1	1000	$4.29 * 10^{-05}$	192.85

3.5.2 Second experiment

In the second experiment, we approximated the sin function variation 3.3. The number of layers and number of neurons were used as parameters in the search space, and grid

search was used to find the best architecture.

Table 3.3: Search space for the number of layers and the number of neurons in experiment two.

Method	Number of layers	Number of neurons
Adam and SWIM	1,2,3	100,500,1000
Adam+	1	2000
SWIM+	1,2,3	1000,2000,3000

Adam

Initially, we trained using Adam on the self-created dataset. The number of layers in the search space was set from one to three, and the number of neurons in the search space was set to 100, 500 and 1000, which can be seen in table 3.3. The 2D heatmap plot of all the configurations where the number of layers is in the X-axis, the number of neurons is in the Y-axis, and the mean squared error (log-scale) is shown using a colour map as well as a 2D heatmap of number of layers vs number of neurons vs fit time for all configurations can be seen in figure 3.9.

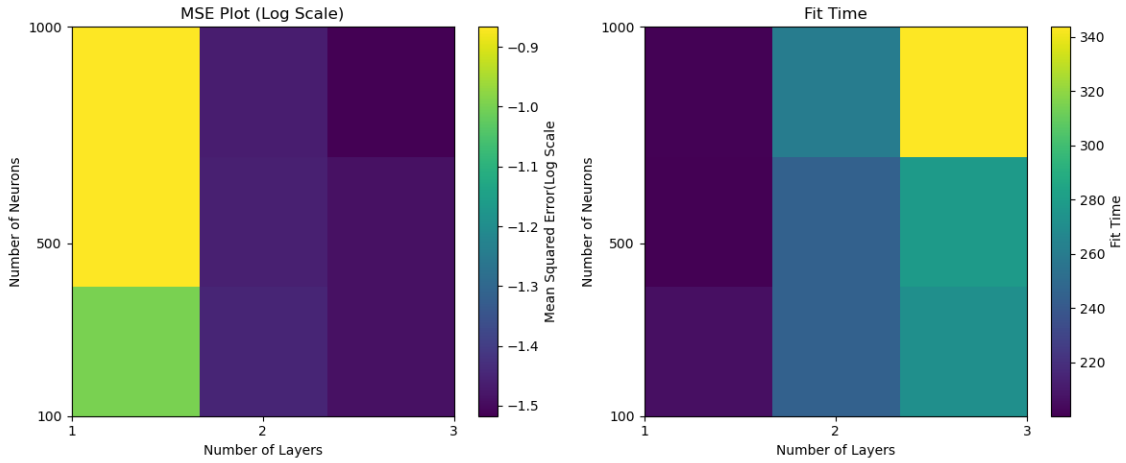

Figure 3.9: 2D heatmap plot of all the configurations where the number of layers is in the X-axis, the number of neurons is in the Y-axis and mean squared error (log-scale) is shown using colour map (left). 2D heatmap of the number of layers vs the number of neurons vs fit time (as colour map) for all configurations (right). This heatmap plot is created using Adam for experiment two.

In the figure 3.9, it can seen that three layers with 1000 neurons give the lowest mse re-

sult. The best architecture configuration with mean squared error value and total fit time for this experiment can be seen in table 3.4.

SWIM

After that, we trained using SWIM. The search space for the number of layers was set from one to three, and the search space for the number of neurons was set to 100, 500 and 1000, which can be seen in the table 3.3. The corresponding 2D heatmap can be visualized in figure 3.10.

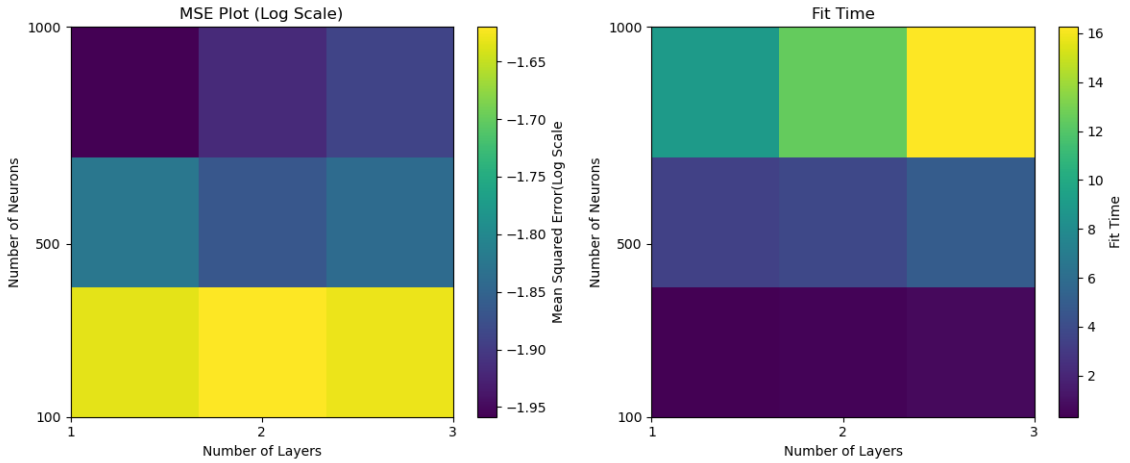

Figure 3.10: 2D heatmap plot of all the configurations where the number of layers is in the X-axis, the number of neurons is in the Y-axis and mean squared error (log-scale) is shown using colour map (left). 2D heatmap of the number of layers vs the number of neurons vs fit time (as a colour map) for all configurations (right). This heatmap plot is created using SWIM for experiment two.

In the figure 3.10, we saw that one layer with 1000 neurons gives the best result. The best architecture configuration with mean squared error value and total fit time for this experiment can be seen in table 3.4. We again trained using SWIM with a higher number of neurons, represented by SWIM+. The number of layers set from 1 to 3 and the number of neurons set to 1000,2000, and 3000, which can be seen in the table 3.3. The corresponding 2D heatmap plot can be visualized in figure 3.11.

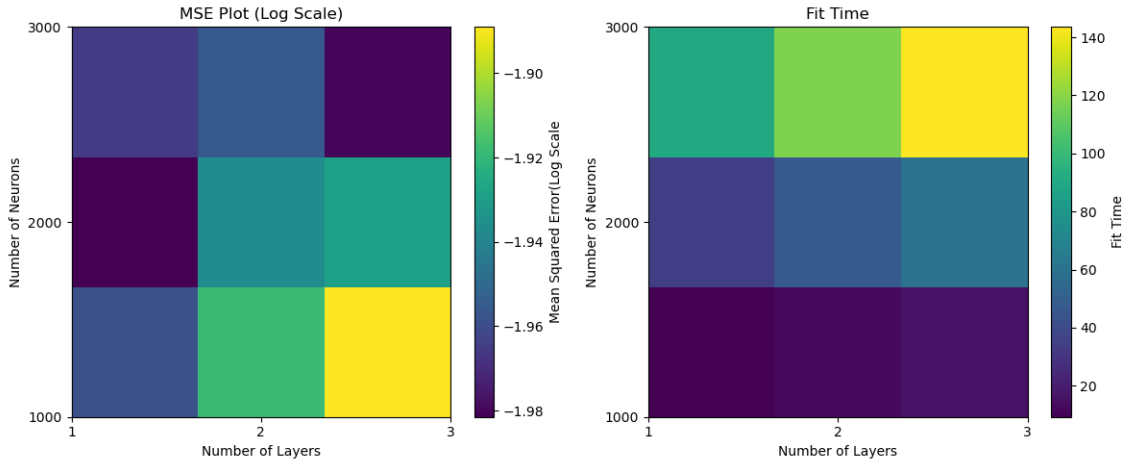

Figure 3.11: 2D heatmap plot of all the configurations where the number of layers is in the X-axis, the number of neurons is in the Y-axis and mean squared error (log-scale) is shown using colour map (left). 2D heatmap of the number of layers vs the number of neurons vs fit time (as a colour map) for all configurations (right). This heatmap plot is created using SWIM for experiment two with a higher number of neurons in the search space represented by SWIM+.

In the figure 3.11, it can be seen that one layer with 2000 neurons gives the best result. The thing to notice here was that it was not the highest value of neurons that gave the best result. Additionally, we tried to train the best architecture from SWIM+, using Adam (represented by Adam+) to compare the results. The best architecture configuration with mean squared error value and total fit time for this experiment can be seen in table 3.4.

Table 3.4: Best architecture configuration with mean squared error value and total fit time for experiment two. * Sign in the fit time for Adam+ indicates the fit time for the best architecture, not the total fit time. The lowest mean squared error and lowest total fit time are marked in bold.

Method	Number of layers	Number of neurons	Mean squared error	Fit time(s)
Adam	3	1000	0.04	585.31
SWIM	1	1000	0.011	**53.46**
Adam+	1	1000	0.135	42.89*
SWIM+	1	2000	**0.0104**	641.09

3.5.3 Third experiment

We also conducted another experiment with the distance function 3.2. We decomposed the function into two functions 3.4 and 3.5. We approximated the functions separately using SWIM. Number of layers and number of neurons were used as parameters in the search space, and grid search was used to find the best architecture.

In this experiment, for both functions, the number of layers in the search space was set from one to three, and the number of neurons in the search space was set from 100 to 1000 in 100 intervals, which can be seen in table 3.5.

Table 3.5: Search space for number of layers and number of neurons in experiment three to approximate function 3.4 and function 3.5 using SWIM.

	Number of layers	Number of neurons
Search Space	1,2,3	100,200,300,400,500,600,700,800,900,1000

Function 3.4

The 2D heatmap plot for function 3.4 can be visualized in the figure 3.12:

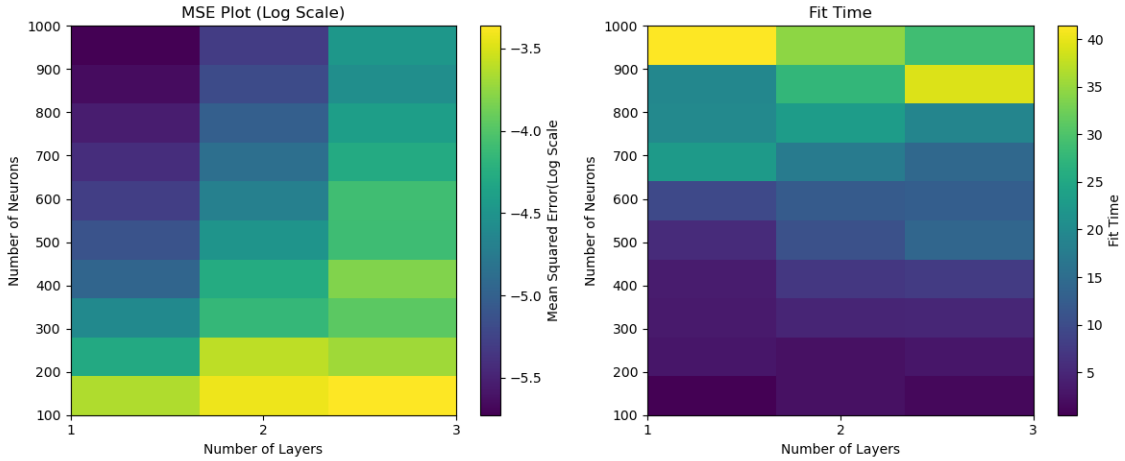

Figure 3.12: 2D heatmap plot of all the configurations where the number of layers is in the X-axis, the number of neurons is in the Y-axis and mean squared error (log-scale) is shown using colour map (left). 2D heatmap of the number of layers vs the number of neurons vs fit time (as colour map) for all configurations (right). This heatmap plot is created using SWIM for experiment three to approximate function 3.4.

In the figure 3.12, it can be seen that one layer with 1000 neurons gives the best result. The best architecture configuration with mean squared error value and total fit time for this experiment can be seen in table 3.6.

Table 3.6: Best architecture configuration with mean squared error value and total fit time for experiment three to approximate function 3.4 and 3.5 using SWIM.

	Number of layers	Number of neurons	Mean squared error	Fit time(s)
Function 3.4	1	1000	$1.87 * 10^{-6}$	197.15
Function 3.5	1	1000	$1.87 * 10^{-6}$	195.92

Function 3.5

Similarly, the 2D heatmap plot for the function 3.5 can be visualized in the figure 3.13.

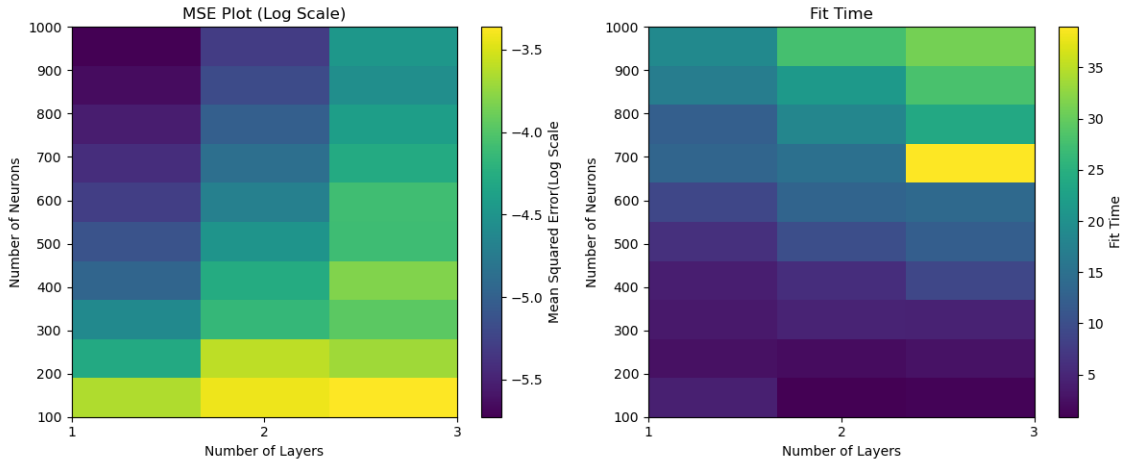

Figure 3.13: 2D heatmap plot of all the configurations where the number of layers is in the X-axis, the number of neurons is in the Y-axis and mean squared error (log-scale) is shown using colour map (left). 2D heatmap of the number of layers vs the number of neurons vs fit time (as colour map) for all configurations (right). This heatmap plot is created using SWIM for experiment three to approximate function 3.5.

In the figure 3.13, it can be seen that one layer with 1000 neurons gives the best result. The best architecture configuration with mean squared error value and total fit time for this experiment can be seen in table 3.6.

Chained Network

After that, we chained the two networks approximating function 3.4 and 3.5. Chaining the two networks meant passing the result of one network into the other network and finally calculating the mean squared error value. The first decomposed distance function 3.4 was approximated separately using network one. Similarly, the second decomposed distance function 3.5 was approximated separately using network two. The output of network one was passed into network two. The result of network two and the chained ground truth value were used to calculate the mean squared error. In this experiment, cross-validation wasn't used. Instead, we created a separate test set with 20000 data points between -2 and 2. The result obtained from chaining two networks can be seen in the figure 3.14.

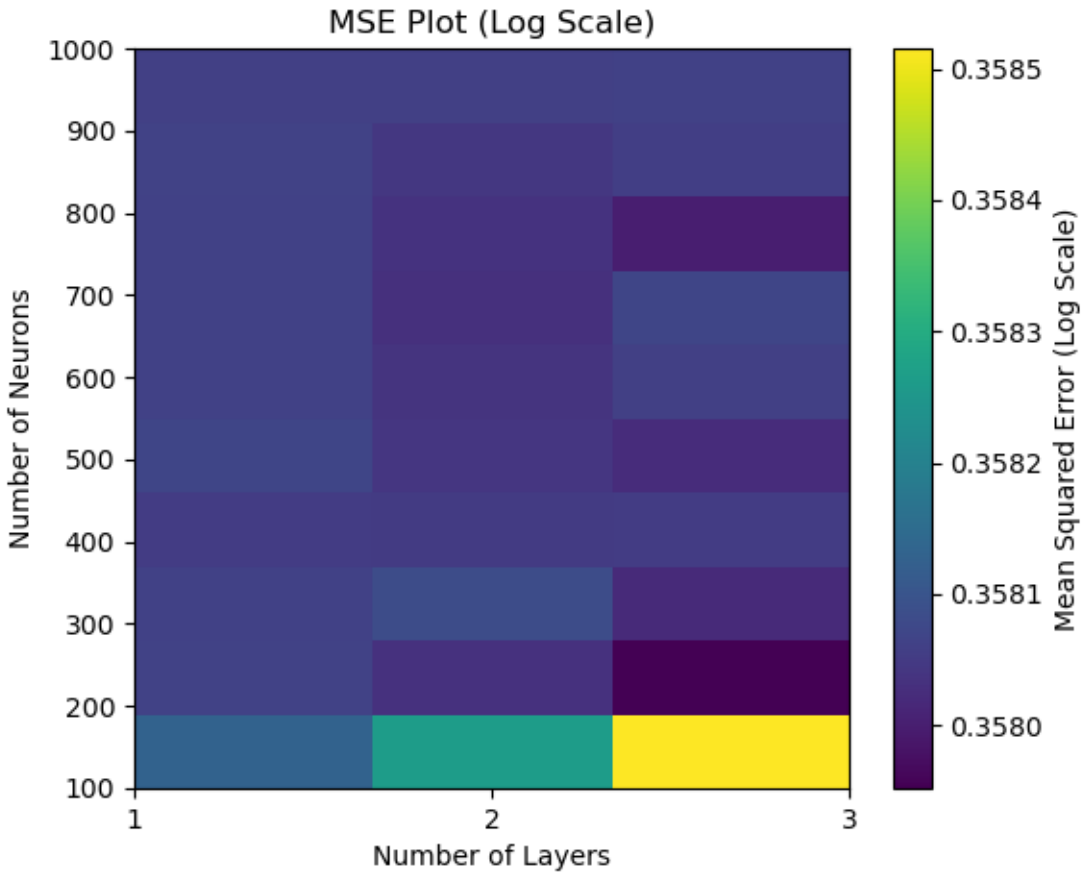

Figure 3.14: 2D heatmap plot of all the configurations where the number of layers is in the X-axis, the number of neurons is in the Y-axis and mean squared error (log-scale) is shown using a colour map. This heatmap plot is created using SWIM for experiment three, where the function is approximated using two chained networks.

In figure 3.14, we could see that the dark blue colour map representing the lower mse value was there when the number of layers was three. The configuration yielding the lowest mse value was when the number of layers was three, and the number of neurons was 200. One thing to note here was three layers with 100 neurons gave the worst result. The best architecture configuration with mean squared error value for this experiment can be seen in table 3.7.

Single Network

After that, we again separately approximated the distance function 3.2 without using cross-validation. We created a separate test set with 20000 data points between -2 and 2. The result obtained can be seen in the figure 3.15.

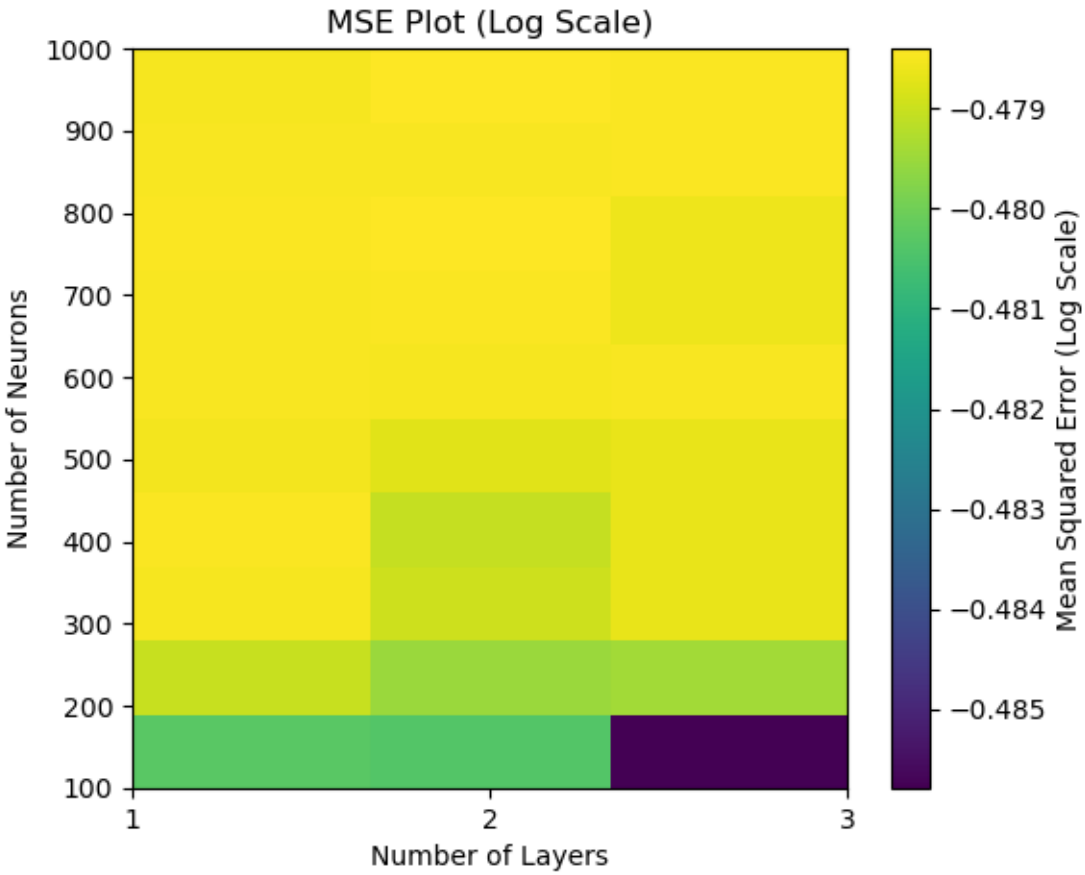

Figure 3.15: 2D heatmap plot of all the configurations where the number of layers is in the X-axis, the number of neurons is in the Y-axis and mean squared error (log-scale) is shown using a colour map. This heatmap plot is created using SWIM for experiment three to approximate function 3.2 using a single network.

In the figure 3.15, it can be seen that when the number of layers was three and the number of neurons was 100, it gave the best result. The best architecture configuration with mean squared error value for this experiment can be seen in table 3.7. The result obtained

from chaining two networks was compared with the result obtained using a single network. It could be seen that approximating the distance function using the single network gave a better result (lower mse) with less number of neurons than approximating it using the chained network. The mean squared error obtained using the single network was 0.326, whereas the mean squared error obtained using the chained network was 2.28.

Table 3.7: Best architecture configuration with mean squared error value and total fit time for experiment three to approximate function 3.2 with chained networks and single network using SWIM. The lowest mean squared error is marked in bold.

	Number of layers	Number of neurons	Mean squared error
Chained Network	3	200	2.28
Single Network	3	100	**0.326**

3.5.4 Fourth Experiment

Another experiment was conducted. We tried to approximate the function 3.6. The sigma values were the array of 10 to the power of 10 values between -3 and -1. This function was chosen because we wanted to fit a bump that was getting sharper. The idea behind this experiment was we wanted to know whether the number of layers required to approximate such functions would be higher (also while using the SWIM method). We approximated the function using both the Adam and SWIM methods.

Table 3.8: Search space for the number of layers and the number of neurons in experiment four.

Method	Number of layers	Number of neurons
Adam and SWIM	1,2,3	200,400,600
Adam+	2	2000
SWIM+	1,2,3	1000,2000,3000
SWIM++	1,2,3	1000, 1200, 1400, 1600, 1800, 2000, 2200, 2400, 2600, 2800

Adam

First, we trained the function using Adam with a self-created dataset discussed in the dataset used section. Grid search was conducted to determine the best neural architecture configuration for this task. The number of layers in the search space was set from one to three, and the number of neurons in the search space was set to 200, 400 and 600, which can be seen in table 3.8. We choose up to 600 number of neurons and up to three number of layers because it would take Adam a very long time to train the function if the parameters are very large. Additionally, the dataset of this experiment is also relatively larger than that of other experiments. The number of epochs was chosen to be 30 because, past that

point, we saw a marginal change in the loss value. The 2D heatmap plot of all the configurations where the number of layers is in the X-axis, the number of neurons is in the Y-axis, and mean squared error (log-scale) is shown using a colour map as well as a 2D heatmap of the number of layers vs the number of neurons vs fit time for all configurations can be seen in figure 3.16.

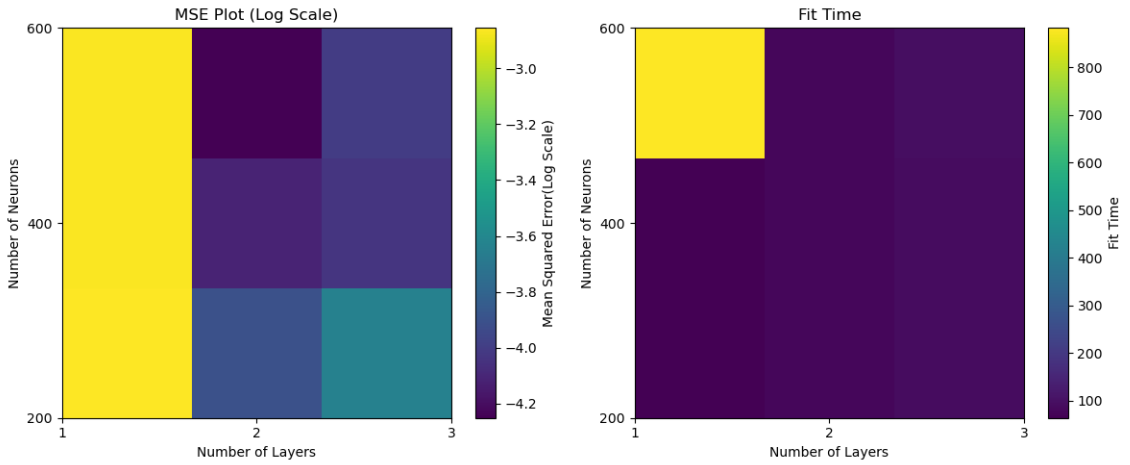

Figure 3.16: 2D heatmap plot of all the configurations where the number of layers is in the X-axis, the number of neurons is in the Y-axis and mean squared error (log-scale) is shown using colour map (left). 2D heatmap of the number of layers vs the number of neurons vs fit time (as colour map) for all configurations (right). This heatmap plot is created using Adam for experiment four.

The best architecture configuration with mean squared error value for this experiment can be seen in table 3.9.

SWIM

Secondly, we conducted the same experiment using the SWIM method. We have used the same dataset and search space that we used for Adam also for the SWIM method. Grid search was performed to get the best architecture configuration for the SWIM method. The number of layers in the search space was set from one to three, and the number of neurons in the search space was set to 200, 400 and 600; the same as Adam, which can be seen in table 3.8. The 2D heatmap plot of all the configurations where the number of layers is in the X-axis, the number of neurons is in the Y-axis, and mean squared error (log-scale) is shown using a colour map as well as a 2D heatmap of the number of layers vs number of neurons vs fit time for all configurations can be seen in figure 3.17.

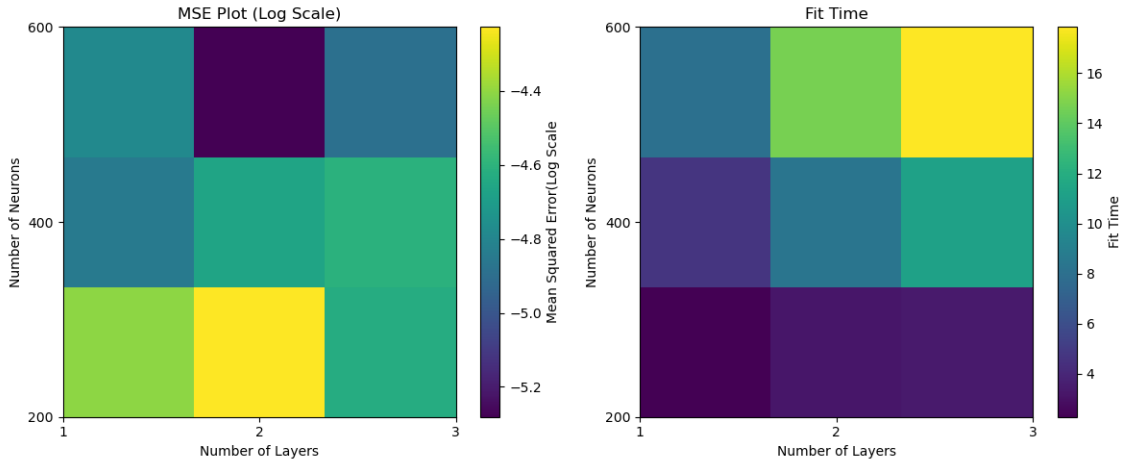

Figure 3.17: 2D heatmap plot of all the configurations where the number of layers is in the X-axis, the number of neurons is in the Y-axis and mean squared error (log-scale) is shown using colour map (left). 2D heatmap of the number of layers vs the number of neurons vs fit time (as a colour map) for all configurations (right). This heatmap plot is created using SWIM for experiment four.

In the figure 3.17, we could see that two layers with 600 neurons gave the best result. The other best configurations following this were three layers with 600 neurons and one layer with 400 neurons. We could see that fit time increased with the increasing number of layers and neurons. The lowest fit time we saw was around four seconds, and the highest fit time was around 16 seconds. The best architecture configuration with mean squared error value for this experiment can be seen in table 3.9. In the table 3.9, we can see that the best neural architecture configuration has two number of layers, 600 number of neurons, $5.23 * 10^{-6}$ mean squared error and 73.59 seconds fit time. The fit time calculated here is the sum of all the fit times of the configurations. If we compare the result of Adam with the SWIM method, we could see that SWIM gives better results with the same parameters. Also, the fit time using SWIM is extremely low, meaning it is much faster to approximate using SWIM than Adam.

For SWIM, we also trained using a larger number of neurons, represented by SWIM+, as it's computational time is much faster than Adam. So, we increased the number of neurons to 1000, 2000 and 3000, keeping the number of layers fixed. The corresponding search space can be seen in table 3.8. The 2D heatmap plot of all the configurations where the number of layers is in the X-axis, the number of neurons is in the Y-axis, and mean squared error (log-scale) is shown using a colour map as well as a 2D heatmap of the number of layers vs number of neurons vs fit time for all configurations can be seen in figure 3.18.

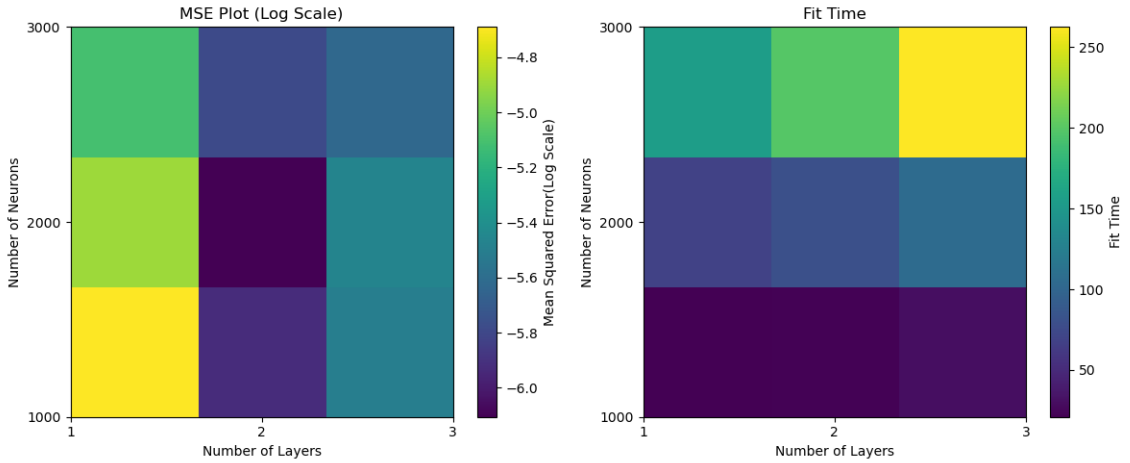

Figure 3.18: 2D heatmap plot of all the configurations where the number of layers is in the X-axis, the number of neurons is in the Y-axis and mean squared error (log-scale) is shown using colour map (left). 2D heatmap of the number of layers vs the number of neurons vs fit time (as a colour map) for all configurations (right). This heatmap plot is created for SWIM+ for experiment four.

In figure 3.18, we could see that two layers with 2000 neurons give the lowest mse value. The best configurations after this were two layers with 1000 neurons and two layers with 3000 neurons, respectively. We can see that fit time increased with the increasing number of layers and neurons. The lowest fit time we saw was around 50 seconds, and the highest fit time was around 250 seconds. The best architecture configuration with mean squared error value for this experiment can be seen in table 3.9 In the table 3.9, we can see that the best neural architecture configuration has 2 number of layers, 600 number of neurons, $5.23 * 10^{-6}$ mean squared error and 73.59 seconds fit time. The fit time calculated here is the sum of all the fit times of the configurations.

Again, we tested the SWIM method using ten neuron values between 1000 and 3000, represented by SWIM++, keeping the number of layers constant as in the previous experiments. The corresponding search space can be seen in table 3.8. The 2D heatmap plot of all the configurations where the number of layers is in the X-axis, the number of neurons is in the Y-axis, and mean squared error (log-scale) is shown using colour map as well as a 2D heatmap of the number of layers vs number of neurons vs fit time as colour map for all configurations can be seen in figure 3.19.

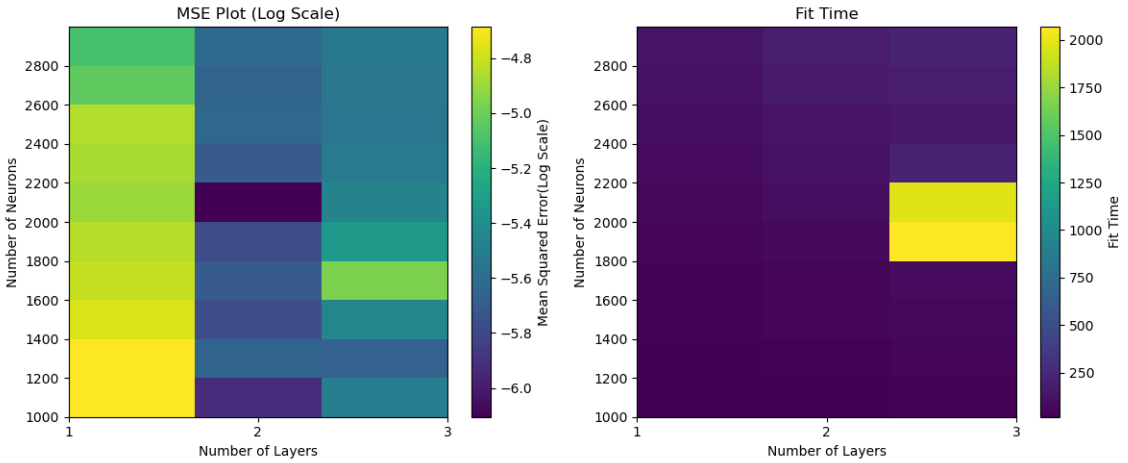

Figure 3.19: 2D heatmap plot of all the configurations where the number of layers is in the X-axis, the number of neurons is in the Y-axis and mean squared error (log-scale) is shown using colour map (left). 2D heatmap of the number of layers vs the number of neurons vs fit time (as colour map) for all configurations (right). This heatmap plot is created for SWIM++ for experiment four.

In figure 3.18, we could see that two layers with 2000 neurons give the lowest mse value. The following best configurations after this was two layers with 1000 neurons. We can see that fit time increased with the increasing number of layers and neurons. However, three layers with 2000 neurons and 1800 had the highest fit time. The lowest fit time we saw was around 250 seconds, and the highest fit time was around 2000 seconds. The best architecture configuration with mean squared error value and total fit time for this experiment can be seen in table 3.9. In the table 3.9, we can see that the best neural architecture configuration has two layers, 2000 neurons, $7.84 * 10^{-7}$ mean squared error and 6537.38 seconds fit time. The fit time calculated here is the sum of all the fit times of the configurations. Additionally, we trained the best architecture from SWIM+/SWIM++ using Adam (represented by Adam+) to compare the results. The result can be seen in table 3.9.

Table 3.9: Best architecture configuration with mean squared error value and total fit time for experiment four. * Sign in the fit time for Adam+ indicates that it is the fit time for the best architecture and not the total fit time. The lowest mean squared error and lowest total fit time are marked in bold.

Method	Number of layers	Number of neurons	Mean squared error	Fit time(s)
Adam	2	600	$5.60 * 10^{-5}$	1520.58
SWIM	2	600	$5.23 * 10^{-6}$	**73.59**
Adam+	2	2000	$1.44 * 10^{-5}$	138.37*
SWIM+	2	2000	$\mathbf{7.84 * 10^{-7}}$	944.79
SWIM++	2	2000	$\mathbf{7.84 * 10^{-7}}$	6537.38

In the first experiment, we approximated the distance function 3.2 using Adam and the SWIM method. Grid search was performed to get the best neural architecture configuration. Additionally, the same search space was used to compare the results from both methods. The number of layers was set to one, two and three and the number of neurons was set to 200, 400 and 600 for both methods. The best configuration obtained using Adam was two layers, 400 neurons, $2.03 * 10^{-05}$ mean squared error and 466.01 second total fit time (sum of fit time of all configurations). Moreover, the best configuration obtained using SWIM with the same parameters was one layer, 600 neurons, $9.92 * 10^{-05}$ mean squared error and 27.38 second total fit time. The best architecture configurations can be seen in table 3.10. From this experiment, we can see that Adam with a higher number of layers and a little lower number of neurons obtained a slightly lower mse value, while the total time to fit the whole configurations was 17 times lower in SWIM than Adam. In the first experiment, we again performed the grid search using more number of neurons using SWIM (represented by SWIM+) as it is magnitude faster to train. The search space for the number of neurons was set from 100 to 1000 in 100 intervals while the number of layers was unchanged. The best configuration obtained after the experiment was one layer, 1000 neurons, $4.29 * 10^{-05}$ mean squared error and 192.85 seconds total fit time. The mse value did not decrease much compared to having 600 neurons, and the overall fit time also increased from 27.38 seconds to 192.85 seconds. But the speed is still better than using Adam with comparable results. Additionally, we trained the best architecture from SWIM+ using Adam (represented by Adam+) to compare the results. It was seen that mse value was worse for Adam+ among all. The fit time for just the single configuration was found to be 105.86 seconds.

In the second experiment, we approximated the sin function variation 3.3 using Adam and the SWIM method. Again, we used grid search to find the best architecture configuration. The number of layers was set to one, two and three and the number of neurons was set to 100, 500 and 1000 for both methods. The best configuration obtained using Adam was three layers, 1000 neurons, 0.04 mean squared error, and 585.31 seconds total fit time.

Moreover, the best configuration obtained using SWIM with the same parameters was one layer, 1000 neurons, 0.011 mean squared error and 53.46 second total fit time. The best architecture configurations can be seen in table 3.10. Here, we could see that SWIM method had a slightly lower mse value than Adam, even with a lower number of layers and the same number of neurons, i.e. 1000. The total fit time using SWIM is almost eleven times faster than that using Adam. After that, we again experimented with a higher number of neurons using the SWIM method (called as SWIM+), keeping the search space for the number of layers the same. The search space for the number of neurons was set to 1000, 2000 and 3000. The best configuration obtained from the experiment was one layer, 2000 neurons, 0.0104 mse value and 641.09 seconds fit time. We observed that the mse value does not decrease much, even with a higher number of neurons. So, training with a higher number of neurons wasn't useful in this case; rather, it just increased the overall fit time. Additionally, we trained the best architecture from SWIM+ using Adam (called as Adam+) to compare the results. It was seen that mse value was worse for Adam+ among all. The fit time for just the single configuration was found to be 42.89 seconds.

In the third experiment, we decomposed the distance function 3.2 into two functions 3.4 and 3.5. The two decomposed functions were approximated separately using the SWIM method. Grid search was done to find the best architecture configurations. The number of layers in the search space was set to one, two and three, and the number of neurons was set from 100 to 1000 in 100 intervals. The best configuration obtained for 3.4 was one layer, 1000 neurons, $1.87 * 10^{-6}$ and fit time was 197.15 seconds. The best configuration can be seen in the table 3.10. Also, for the function 3.5, the search space for the number of neurons was set to one, two and three and the number of neurons was set from 100 to 1000 in 100 intervals. The best configuration obtained for 3.4 was one layer, 1000 neurons, $1.87 * 10^{-6}$ and fit time was 197.92 seconds. The best configuration can be seen in the table 3.10. Moreover, the networks approximating two functions 3.4 and 3.5 were chained together to see if the result is better than approximating using a single network. The search space was the same as before, i.e. the number of layers was set to one, two and three, and the number of neurons was set from 100 to 1000 in 100 intervals. The best architecture configuration obtained using the chained network was three layers, 200 neurons and a 2.28 mse value. However, the best architecture configuration obtained using the single network with the same parameters was three layers, 100 neurons and a 0.326 mse value. As we can see, the result was much better using the single network than the chained networks with the same number of layers and an even smaller number of neurons.

Finally, in the fourth experiment, we approximated the function 3.6 using Adam and the SWIM method. Again, we used grid search to find the best architecture configuration. The number of layers was set to one, two and three and the number of neurons was set to 200, 400 and 600 for both methods. The best configuration obtained using Adam was two layers, 600 neurons, $5.60 * 10^{-5}$ mean squared error and 1520.58 seconds total fit time. Moreover, the best configuration obtained using SWIM with the same parameters was two

layers, 600 neurons, $5.23 * 10^{-6}$ mean squared error and 73.59 seconds total fit time. The best architecture configurations can be seen in table 3.10. The result obtained from the SWIM method was better and with greater speed than the result obtained from the Adam method. Moreover, the result was obtained 20.6 times faster using SWIM than Adam. Furthermore, we experimented with a higher number of neurons (represented by SWIM+) using the SWIM method, keeping the search space for the number of layers the same. The number of neurons in the search space was set to 1000, 2000 and 3000. We observed that the best configuration for SWIM+ was two layers with 2000 neurons yielding $7.84 * 10^{-7}$ mse error and 944.79 seconds of total fit time. After that, we again experimented with a higher and more number of neurons in the search space (represented by SWIM++) using the SWIM method, keeping the search space for the number of layers the same. The number of neurons in the search space was set from 1000 to 3000 in 200 intervals. The best configuration obtained from the experiment was two layers, 2000 neurons, $7.84 * 10^{-7}$ mse value and 6537.38 seconds total fit time. We observed that the mse value increased with increasing neurons from 1000 up to 1800 neurons, and the mse value was the lowest when the number of layers was two and the number of neurons was 2000. The mse value was again higher for 2000 to 2800 neurons. So, training with more neurons than 2000 neurons wasn't very useful in this case. Furthermore, we can see that mse value for SWIM+ and SWIM++ was the same as the best architecture configuration was the same for both of them. Additionally, we trained the best architecture from SWIM+/SWIM++ using Adam (represented by Adam+) to compare the results. It was seen that mse value was worse compared to SWIM+, SWIM++ and SWIM, but it was a bit better than the Adam result. Moreover, the fit time for Adam+ to train just the single configuration was found to be 138.37 seconds.

Table 3.10: Best architecture configuration with mean squared error value and total fit time for experiments one, two, three and four. * Sign in the fit time for Adam+ indicates that it is the fit time for the best architecture and not the total fit time. The lowest mean squared error and lowest total fit time are marked in bold for each experiment.

	Method	Number of layers	Number of neurons	Mean squared error	Fit time(s)
Experiment 1	Adam	2	400	$\mathbf{2.03 * 10^{-05}}$	466.01
	SWIM	1	600	$9.92 * 10^{-05}$	**27.38**
	Adam+	1	1000	$5.77 * 10^{-04}$	105.86*
	SWIM+	1	1000	$4.29 * 10^{-05}$	192.85
Experiment 2	Adam	3	1000	0.04	585.31
	SWIM	1	1000	0.011	**53.46**
	Adam+	1	1000	0.135	42.89*
	SWIM+	1	2000	**0.0104**	641.09
Experiment 3	Chained Network	3	200	2.28	
	Single Network	3	100	**0.326**	
Experiment 4	Adam	2	600	$5.60 * 10^{-5}$	1520.58
	SWIM	2	600	$5.23 * 10^{-6}$	**73.59**
	Adam+	2	2000	$1.44 * 10^{-5}$	138.37*
	SWIM+	2	2000	$\mathbf{7.84 * 10^{-7}}$	944.79
	SWIM++	2	2000	$\mathbf{7.84 * 10^{-7}}$	6537.38

4 Conclusions

4.1 Summary

In this thesis project, we conducted three experiments to compare two training techniques (Adam and SWIM) in neural architecture search. In experiment one, we approximated functions 3.2 using Adam and SWIM and compared the mean squared value of the best neural architectures found using the two methods in neural architecture search and the total fit time. Again, in experiments two and four, we approximated functions 3.3 and 3.6, respectively and compared the results obtained using Adam and SWIM. Finally, in experiment three, we compared the result obtained from chaining the networks to the result obtained using a single network to approximate 3.2 function. It was seen that a single network approximated the function better than chaining two decomposed function networks. The SWIM result in all other experiments was comparable, if not better, to the result obtained using the Adam method. Additionally, the total fit time was one to two orders of magnitude lower for the SWIM method than the Adam method. From the result, we could see that Adam required more layers with decent neurons to get the best architecture. However, while using SWIM, we saw that increasing the layers did not affect the result at all and increasing the neurons only gave a slightly better result.

4.2 Discussion

From the results, we could see that using Adam for training required a higher layer with decent neurons to give the best result. However, while using SWIM for training, we could see that a single layer with mostly highest available neurons gave the best result except in experiment four, where it required two layers to obtain the best architecture. In experiment one, we could see that two layers with decent neurons could approximate the distance function 3.2 using Adam but one layer was enough to approximate it using SWIM with comparable results. So, we could agree with Weinan E. [5] in the finite setting too, about the approximation with two layers, but using Adam. From experiment four, we could say that a higher layer is required to approximate a steeper function, even with SWIM. The mse value using the SWIM method is better in two experiments (experiments two and four) and slightly lower in one experiment compared to using the Adam method. Although the results are in favour of the SWIM method here, we can't really decide whether it is better than the iterative optimization method in all cases as there are a lot of factors that might affect the result produced, but it is definitely comparable. Only in experiment four

could we really compare the result from Adam and SWIM as the number of layers and the number of neurons required to get the best architecture configuration are the same in both methods, and the mse value that we got was lower using SWIM method that too 20.6 times faster to get the trained network. One thing that really stood out was the time required to get the trained network using SWIM, which was much faster than Adam. To answer our main research question, as SWIM can produce comparable results with high speed in MLPs, deep learning can be relevant in MLPs, but currently, there seems to be way less researchers interested in using MLPs for deep learning.

4.3 Outlook

One thing to notice here is all of the experiments conducted here are self-created except experiment two, where we took the function from [5]. We could not find papers that used MLPs for neural architecture search and clearly distinguished whether shallow or deep neural networks are better for regression tasks. One future work could be experimenting more with SWIM and determining whether this method is competitive and robust enough. Another future work could be, in this thesis project, we wanted to have a larger search space in neural architecture search but due to time constraints and limitation of computational resources, we were not able to experiment with that.

List of Figures

List of Tables

Bibliography

[1] Marta Bagińska and Piotr E. Srokosz. The optimal ANN model for predicting bearing capacity of shallow foundations trained on scarce data. *KSCE Journal of Civil Engineering*, 23(1):130–137, dec 2018.

[2] Erik Lien Bolager, Iryna Burak, Chinmay Datar, Qing Sun, and Felix Dietrich. Sampling weights of deep neural networks, 2023.

[3] Liang-Chieh Chen, Maxwell D. Collins, Yukun Zhu, George Papandreou, Barret Zoph, Florian Schroff, Hartwig Adam, and Jonathon Shlens. Searching for efficient multi-scale architectures for dense image prediction, 2018.

[4] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(61):2121–2159, 2011.

[5] Weinan E, Chao Ma, Stephan Wojtowytsch, and Lei Wu. Towards a mathematical understanding of neural network-based machine learning: what we know and what we don't, 2020.

[6] Evangelos Galaris, Gianluca Fabiani, Ioannis Gallos, Ioannis Kevrekidis, and Constantinos Siettos. Numerical bifurcation analysis of pdes from lattice boltzmann model simulations: a parsimonious machine learning approach. *Journal of Scientific Computing*, 92(2):34, Jun 2022.

[7] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[8] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989.

[9] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[10] Petro Liashchynskyi and Pavlo Liashchynskyi. Grid search, random search, genetic algorithm: A big comparison for nas, 2019.

[11] Keiron O'Shea and Ryan Nash. An introduction to convolutional neural networks. *arXiv preprint arXiv:1511.08458*, 2015.

[12] Sebastian Ruder. An overview of gradient descent optimization algorithms, 2017.

[13] Pramila P. Shinde and Seema Shah. A review of machine learning and deep learning applications. In *2018 Fourth International Conference on Computing Communication Control and Automation (ICCUBEA)*, pages 1–6, 2018.

[14] Tijmen Tieleman and Geoffrey Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: neural networks for machine learning*, 4:26–31, 2012.

[15] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2023.

[16] R. Vinayakumar, K.P. Soman, K.K. Senthil Velan, and Shaunak Ganorkar. Evaluating shallow and deep networks for ransomware detection and classification. In *2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, pages 259–265, 2017.

[17] Colin White, Mahmoud Safari, Rhea Sukthanker, Binxin Ru, Thomas Elsken, Arber Zela, Debadeepta Dey, and Frank Hutter. Neural architecture search: Insights from 1000 papers, 2023.

[18] Hui Yu, David C. Samuels, Ying-yong Zhao, and Yan Guo. Architectures and accuracy of artificial neural network for disease classification from omics data. *BMC Genomics*, 20(1):167, Mar 2019.

[19] Barret Zoph and Quoc V. Le. Neural architecture search with reinforcement learning, 2017.

[20] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V. Le. Learning transferable architectures for scalable image recognition, 2018.