Investigating the Accuracy of Neural Networks Against Fourier Featured Networks at Predicting Survival Months of Patients with Breast Cancer

Research question: *To what extent is a Fourier Featured Network (FFN) more accurate than artificial neural networks (ANNs) at predicting survival months for persons diagnosed with breast cancer, depending on the size of the training data?*

Subject: *Computer Science*

Word Count: *3941*

Table of contents

Introduction:

Machine learning (ML) is a form of Artificial intelligence (AI) and is “the capability of machines to imitate intelligent human behavior” (Brown *et al., explained*). A machine learning model is a program that recognizes relationships between different variables and uses that to make predictions from unseen data (Databricks). ML is crucial as it plays an important role in decision-making by detecting complex relationships that would otherwise be challenging or impossible for humans to recognize (Kaushik), offering improved and resource-efficient assistance in places where humans cannot.

Neural networks (NN), a subset of ML which comes under supervised learning, where labeled data (for example images of cats and dogs with labels showing if it is a cat or a dog) is used to train a model to predict unlabeled data (IBM). NN’s are based on the human brain and work by using function approximation. Function approximation is finding an unknown hidden function that maps the data available (inputs) to the labels (outputs) (Brownlee).

Breast cancer is one of the most prevalent cancers among women (WHO) and the fourth leading cause of cancer-related deaths worldwide (Watson). Since 70% of cases arise without known risk factors, it is vital to accurately detect it as early as possible (Risk factors). Personally, breast cancer has impacted my family, with both my aunt currently in chemotherapy and my grandmother having passed from it. Therefore, it is essential to improve the prediction of survival rates post-diagnosis.

The breast cancer dataset from Kaggle, originally collected by the SEER program of the NCI can be used to train a NN to predict survival months post-diagnosis based on statistics such as age, race, tumor size, hormone levels, stage of cancer, and where it has spread and differentiability using function approximation.

The Fourier series is another way to approximate a function (MIT); a Fourier series is a mathematical series that consists of a sum of sines and cosines until a specified order N (Byju's), with coefficients following each sine and cosine term that change to approximate the target function. A Fourier-featured network FFN is a way in which the input value of the data is taken and transformed into a Fourier series of order N and then fed into a neural network, theoretically increasing the accuracy of the approximated function. (Tancik et al.) Hence, the research question: **To what extent is a Fourier Featured Network (FFN) more accurate than artificial neural networks (ANNs) at predicting survival months for persons diagnosed with breast cancer, depending on the size of the training data?**

Background Research:

Other methods:

On Kaggle, there are more than 86 different approaches for the dataset, including methods such as linear regression, random forests, gradient boosting, and ANNs. Currently, the best approach with the lowest RMSE (root mean square error) value was using an ANN, achieving an RMSE of 0.8470. However, no approaches use a Fourier Fourier-featured network, nor do they measure how effective it may be depending on the training data size, which this paper aims to expand upon.

Artificial Neural Network Architecture

Artificial neural networks are machine learning models based on studies of the human brain and nervous system (*Artificial Neural Network*). A neural network algorithm has two parts: the training process and the prediction process (Ryan).

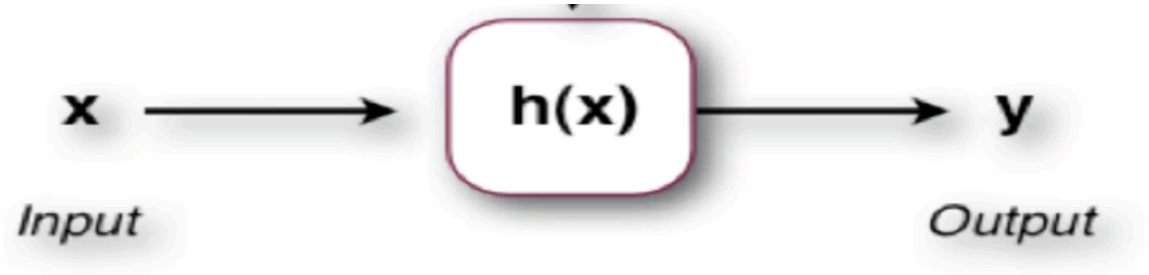


Fig 1.0 (Gillian)

As shown in Fig 1.0, Mathematical functions are input-output machines; they take in an input and give out a correlated output, and the function h(x) is the relationship between the input and the output. The problem NNs try to solve is finding the relationship between the input and output without knowing the function itself (Emergent Garden). This works on the universal approximation theorem, which states that neural networks are universal; no matter what the target function h(x) is, there will be a neural network that can approximate it, given enough data is provided (Sahay).

Neural networks work based on the architecture shown by fig 1.1

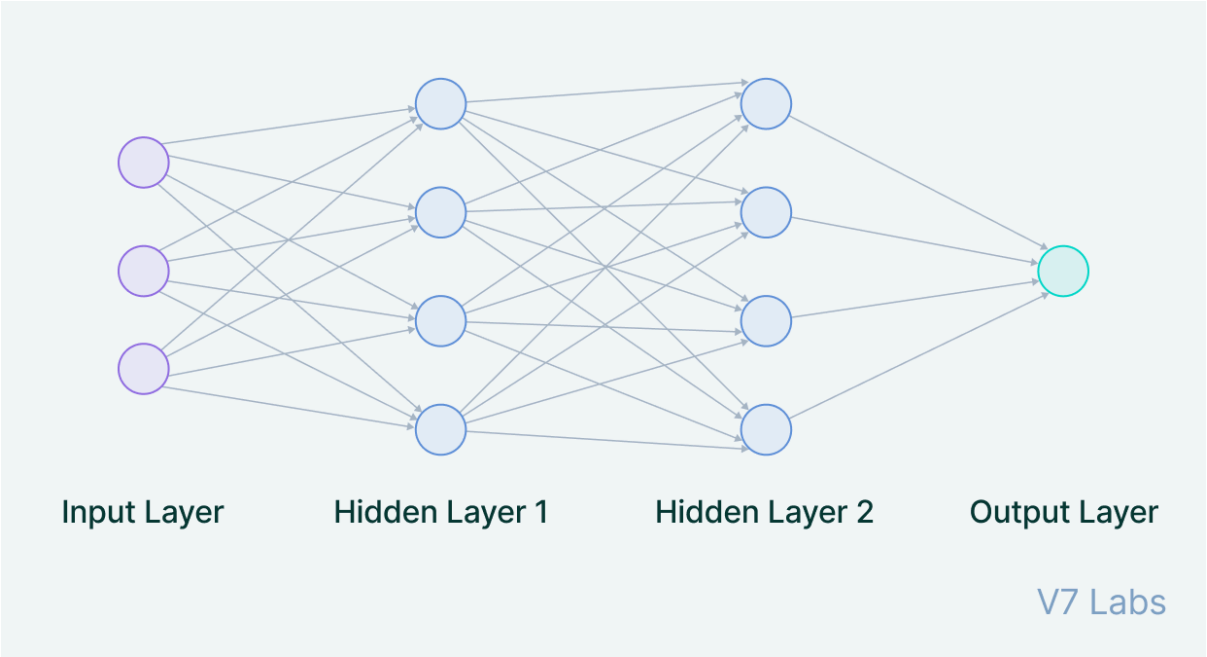


Fig1.1 (Baheti)

It consists of an input layer, hidden layers, and then an output layer. These layers contain neurons or nodes that make a neural network what it is.

Input layer

As shown in Fig. 1 .1, the input layer is the first layer in the neural network workflow. It stores the features of the initial data - an individual measurable value such as age, gender, pixel color value, etc. - in the neurons, which are then sent for further processing toward the hidden layers (Rouse).

Hidden layer

Hidden layers are layers of neurons where deep learning happens and where the network figures out complex patterns and data representation. Depending on how the Neural network is configured, there can be any number of hidden layers. Its function, through neurons, is to find the relationship between the features (input data) and the output and approximate the target function to link the input layer to the output layer (DeepAI). This is done through neurons.

Neurons

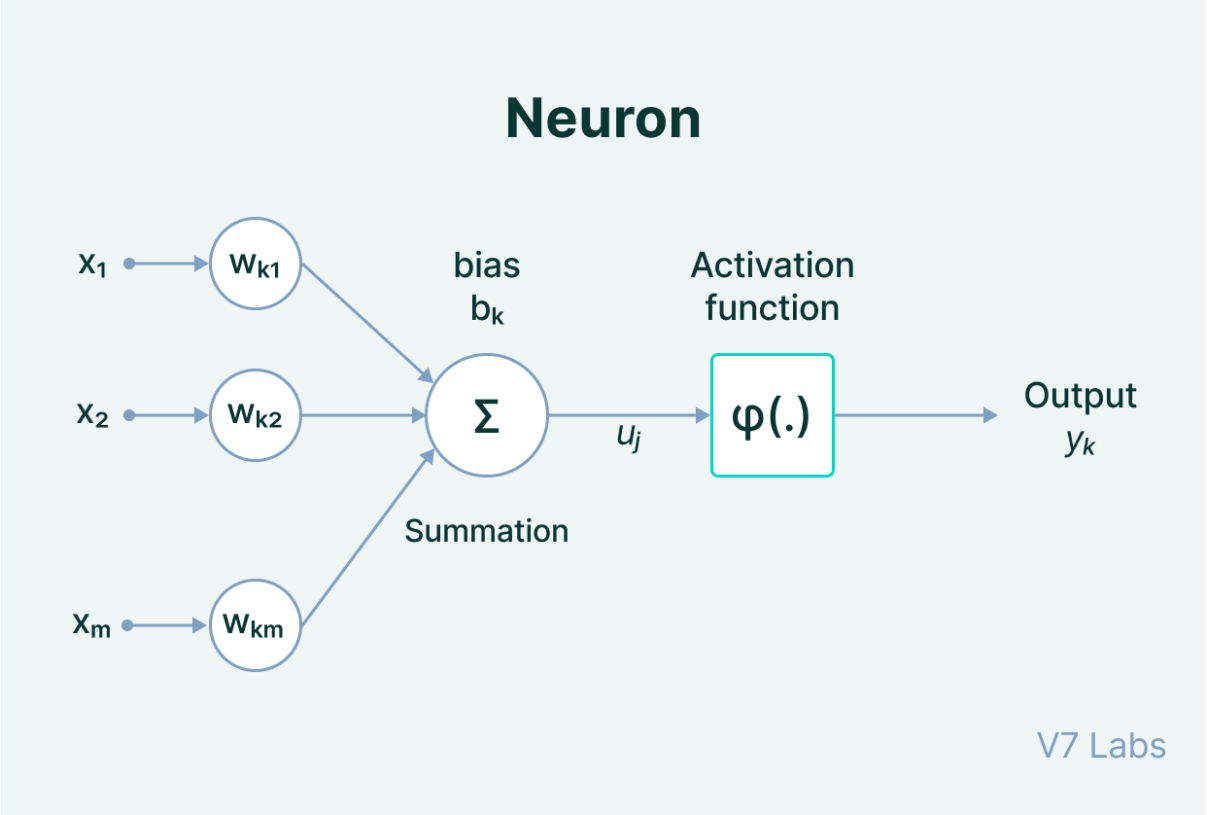


Fig 1.2(Baheti)

As seen in Fig 1.2, A neuron is a collection of inputs (X_1, X_2, X_3…), a set of weights (W_{k1}, W_{k2}, W_{km}…), bias (b_k), a summation, and an activation function. It takes the inputs and,

through the summation, weights, bias, and activation function, turns them into a single output, much like what the neural network does as a whole; this output becomes one of the inputs or features for a neuron in the next layer (Bhargav).

Inputs are the outputs of the previous layer or the values of the features from the data fed into the input layer. Each input has a weight attached to it. Weights are modifiers to the input values. They determine how important a certain feature is to the neuron. The stronger the weight is, the more important the value of that certain feature becomes, changing the value of the output. *A neural network learns or trains by changing the weights* (Gershenson). Bias are constant terms that shift the output value, which also changes during the learning process. They greatly optimize a neural network's accuracy (Vardi et al.).

The summation process is the addition of all the inputs multiplied by their weights plus the bias (Rabinovich), as shown in fig1.3

$$b_k + w_{k1} \cdot x_1 + w_{k2} \cdot x_2 + \ldots + w_{km} \cdot x_m = Z$$

Fig 1.3 weighted inputs plus the bias represented in linear form

After all the weighted inputs and biases are added up, they give a final value of Z; this value is then put into the activation function.

The activation function is a function that transforms the weighted sum value Z into a value between 0 and some number (depending on the activation function used); this value shows the *degree of confidence* that a neuron has that its particular feature is present in the data (McClelland and Rumelhart). This value is then sent as an input to the next layer. Here, each neuron in the next layer receives inputs from all the neurons in the previous layer, continuing the pattern through each layer of the network. This iterative process is essential and

progressively captures specific aspects of the data, starting from basic attributes in the first layers to more complex ones in the later layers. For example, in image recognition, learning the different colors, then learning how chains of those colors may become edges or textures, and then how those edges and textures come together to form shapes and eventually an object.

There are many different activation functions, such as linear, tanh, sigmoid, and ReLU (Sharma et al.), but the most used function and the one considered the best for the most use cases is the rectified linear unit ReLU and its different variations (Ramachandran et al.).

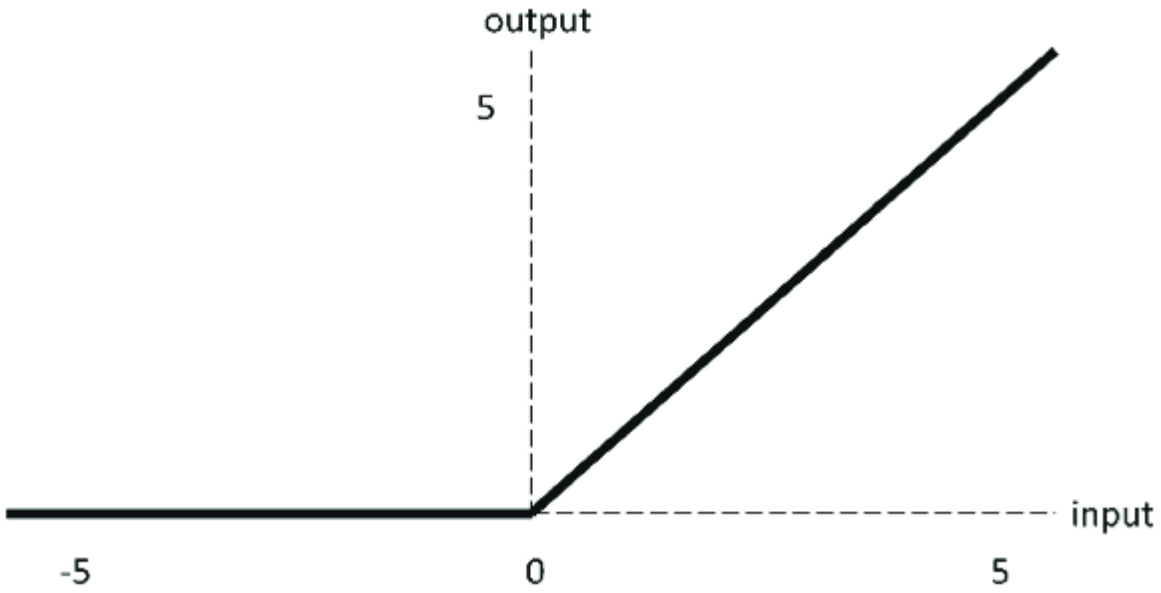


Fig 1.4 ReLU curve (Vivek)

The ReLU curve transforms the sum of the weighted inputs and bias f(Z) to max(0, Z); if the value of Z is negative, it becomes 0, but if it is not negative, it stays as itself. Activation functions introduce non-linearity because the weighted sum of the inputs and bias is linear; if the whole model were linear, it would not be able to approximate target functions that are not linear and are curves or exponential, etc. Therefore, activation functions such as tanh and sigmoid introduced non-linearity (Sweta). However, the problem with those activation

functions is that they are computationally more expensive than ReLU (Szandała) and have the problem of vanishing gradients.

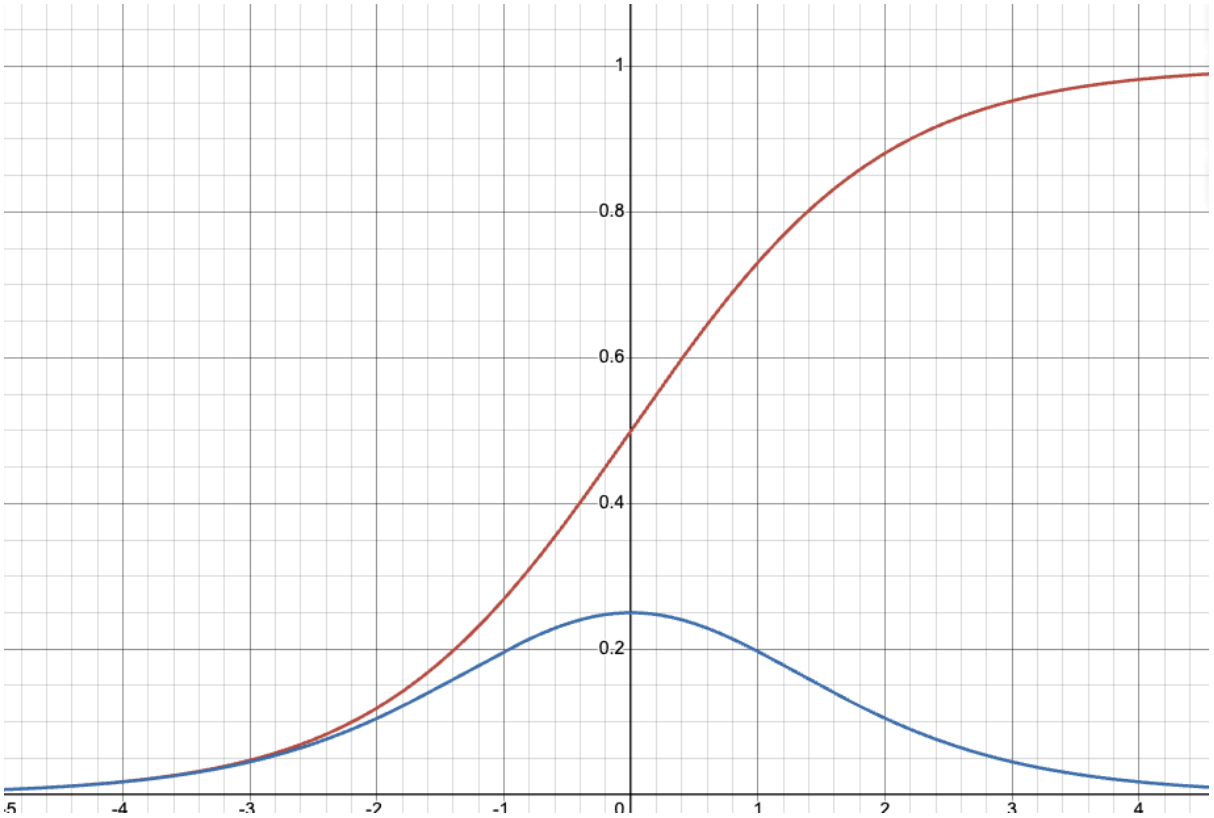


Fig 1.5 sigmoid curve (red) and its gradient (blue) created in desmos

Looking at Fig 1.5, as the output value goes near 1 or 0, the gradient of the sigmoid curve becomes really small; this causes problems for the backpropagation algorithm, which is used to change the value of the weights and biases while the ReLU curve has a constant gradient

(Ye). The ReLU curve is not technically a curve; it is a piecewise linear curve.

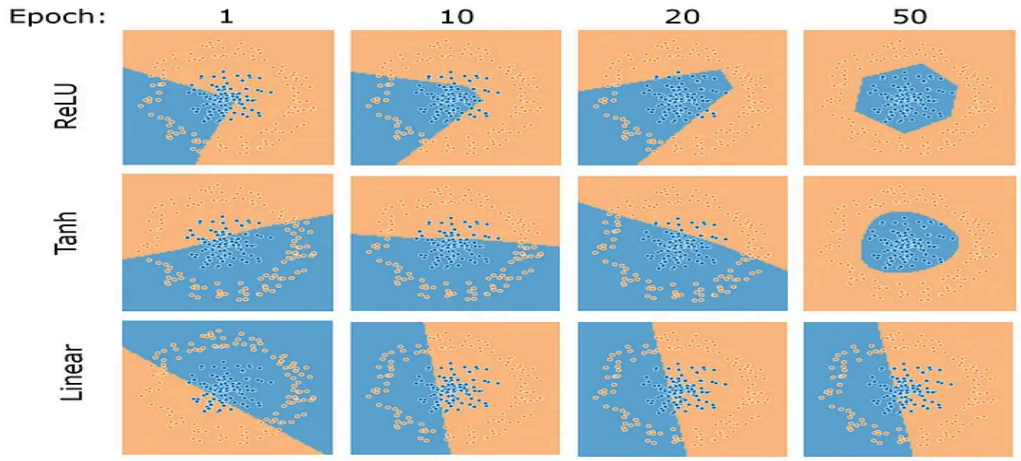


Fig 1.6(Ye)

In a non-linear problem, as shown in Fig 1.6, the linear activation function does not even come close compared to the tanh or ReLU functions. The tanh function is the closest, but the ReLU curve creates a hexagonal shape. Due to its piecewise nature, it can bend to approximate non-linear functions, and by using multiple ReLU activation functions in multiple layers, it can start to closely approximate a non-linear function while being less computationally expensive and avoiding the vanishing gradient problem.

Output layer

With the summation of weighted inputs and biases and the use of activation curves to introduce non-linearity happening across thousands of neurons across multiple layers, which feed into each other, a neural network is able to approximate the function describing the relationship between the inputs and the outputs. The output layer depends on the problem being solved; if it is a classification problem, the output layer will have multiple neurons depending on how many classes there are; however, in a prediction problem, the output layer has one neuron (Sarita). The value of this neuron corresponds to the prediction value, and it does not have an activation function; the output of the weighted sum is the prediction.

How do neural networks learn

Neural network learning occurs when the weights and biases change to approximate the relationship between the input and the output; this is done through a cost function and backpropagation.

Cost function:

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

Fig 1.7 MSE formula

A cost function, such as mean squared error (MSE), shown in Fig 1.7, measures how far off the NN's predictions are from the actual values. Mathematically, it's the difference between the actual output and the predicted output squared divided by the number of observations.

Essentially, it shows how far off the NN's predictions are from being correct on average. By minimizing this function, the NN becomes more accurate (Alooba).

Table 1: MSE formula variables (Frost)

y_i	Is the actual value for each instance
$\hat{y}_i$	Is the predicted value for each instance
n	Number of observations

Gradient descent

Gradient descent is the optimization method used to minimize this cost function. The goal is to find the point where the cost function is at its lowest value—the point where the neural network is most accurate.

To simplify, if the cost function is graphed in a 2d field

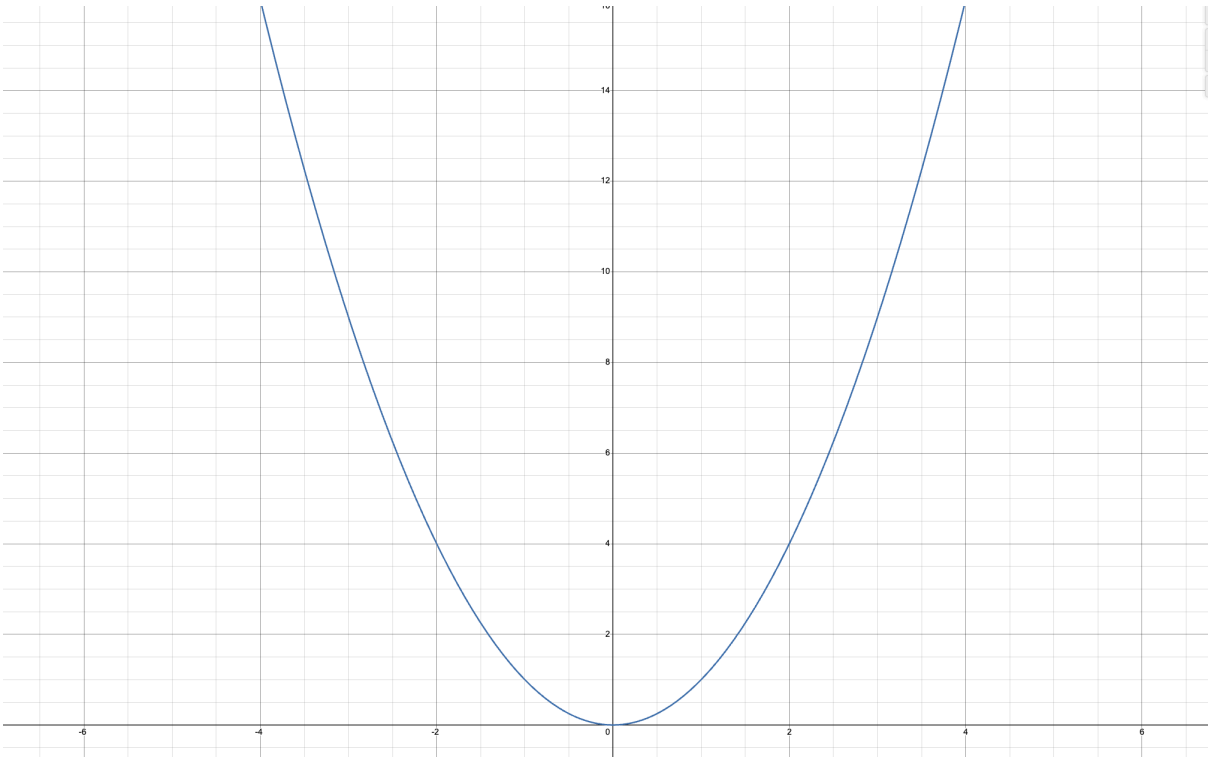


Fig 1.8 graph of x^2

Fig 1.8 is an x^2 graph with a derivative of $2x$. In calculus, the direction of the derivative at any point on a function tells us the direction of the steepest ascent (Marsden and Tromba); conversely, the opposite of that direction is the direction of the steepest descent going towards the minimum point of (0,0). For example, if the starting point is (+2,4) and the derivative at that point is +4, the direction of the steepest ascent (where the model gets less accurate) is towards the positive (going rightwards) direction; therefore, to minimize the cost, the point should move towards the opposite (negative or leftwards) direction where the model would get more accurate as the MSE value is decreased.

This rule is used to adjust the weights and biases to reach the minimum cost value. However, it should be noted that since the predicted value is a product of thousands of weights and biases, the cost function is not a simple 2D graph but a multi-dimensional

surface with more directions (Sanderson), even though the direction of the steepest ascent rule still applies.

Backpropagation

Backpropagation is the algorithm that takes the difference between the actual value and predicted value and calculates (using mathematics out of the scope of this paper) how much each weight and bias needs to change to make the difference zero; it does this for every output as they have different changes needed to the weights and biases. After that, it averages those changes by adding them up, dividing them by total outputs, and applying them to each weight and bias, theoretically reaching the minimum point of the cost function. (Sanderson)

Fourier Series

The Fourier series is a mathematical series composed of an infinite sum of sines and cosines (practically until a specified order N) and is another way to approximate functions. (Weisstein). A mathematical function f(t) can be represented through the Fourier series by changing the coefficients of the sine and cosine components.

$$f(x) = \frac{a_0}{2} + \sum_{n=1}^{N}\left[a_n \cdot \cos\left(\frac{2\pi nx}{T}\right) + b_n \cdot \sin\left(\frac{2\pi \mathrm{nx}}{\mathrm{T}}\right)\right]$$

Fig 1.9

Fig 1.9 shows the general form of Fourier series approximation where a0, an, and bn are coefficients that need to be changed, T is the period of the function, and N is the number of terms in the series (the specified order N). here is the Fourier series being used to predict different functions

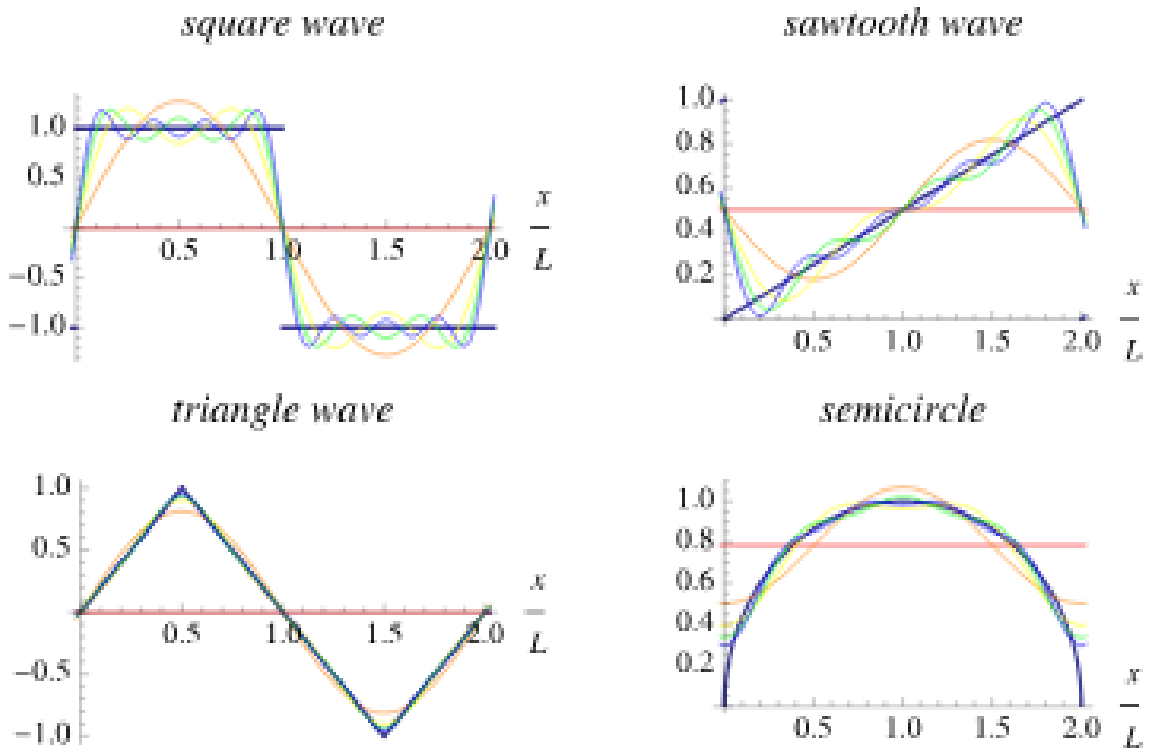


Fig 2.0(Weisstein)

The Fourier series is beneficial as it can turn complex functions representing the relationship between the inputs and outputs into a more straightforward set of terms, highlighting only the most noticeable relationships between the data, potentially filtering out other relationships that are unnecessary to the problem, improving the ability to generalize the patterns giving out more accurate predictions. Moreover, using a Fourier series also enhances feature extraction, finding important characteristics that are not easy to see in the original function and revealing hidden patterns. (Kodad)

Fourier Featured networks (FFNs)

A Fourier-featured network has the same architecture as an ANN but with one change - the input data points are taken and transformed into sinusoidal components of sines and cosines like the Fourier series in the first neural network layer (Tancik et al.)

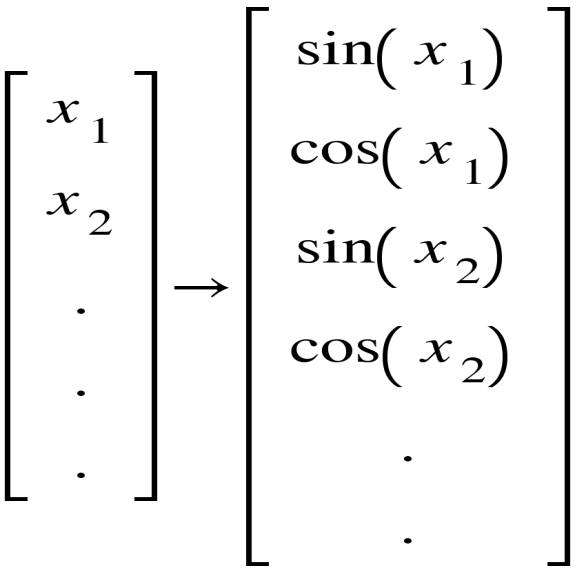


Fig 2.0: inputs are turned into sinusoidal components of order 1

Fig 2.0 shows how the inputs are turned into sinusoidal components. Notice how there are double the sinusoidal values for each x input when it is ordered 1 (one value for sin and one for cos). If the order is increased to 2, it would become four times the initial input, and so on. This increases the number of features extracted from one to two, potentially finding hidden relationships as described above.

To calculate the correct coefficients for the Fourier series, these sinusoidal components are fed into the hidden layers of the NN, and the weights and biases the NN calculates correspond to the coefficients of the Fourier series, so the weighted sum of the inputs becomes $b + W_1 \sin x + W_2 \sin x + W_3 \sin x \ldots$, forming a Fourier series (Sanderson).

There are some pitfalls with using an FFN, though, since it creates more features to extract if there is not enough data, the model may start to overfit, where it just learns how the exact training input maps to the exact training output instead of finding the general relationship as there is not enough data to generalize the increased number of features, leading to the accuracy decreasing for unseen values (Sanderson).

Methodology

The RQ measures the accuracy of a model with and without Fourier features based on training data sizes. Accordingly, the final accuracy of each method will be recorded, as will the increase in accuracy for both models as the training data increases. The accuracy will be measured in RMSE (root mean square error), which is the square root of MSE, showing on average how far the predictions are from the actual values.

The RQ requires the evaluation of these models to focus on predicting survival months for patients with breast cancer; therefore, the SEER breast cancer data set will be used to achieve this task.

Primary data from the training and evaluation of both models is used because it allows more control over the independent variables; as a consequence, the computational power available is limited.

Dataset and data format

The data set with 4024 different patients is downloaded from Kaggle into Python and processed. It is then split into training, test, and validation data (65%, 15%, and 25%). Then, the remaining training data is split into 20%, 40 %, 60%, 80%, and 100%

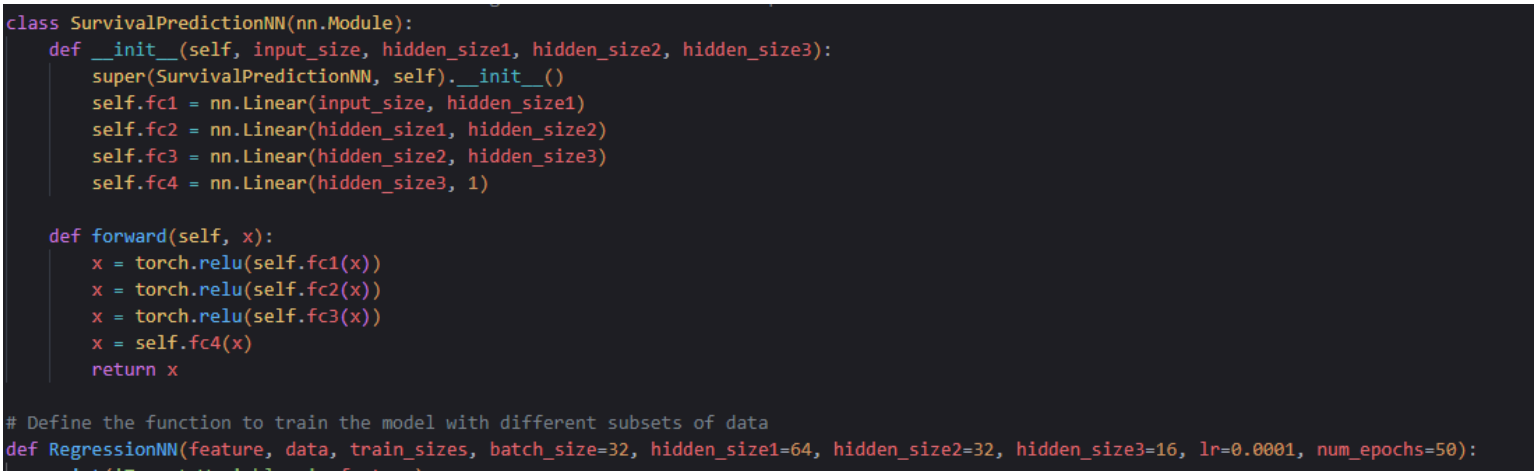
Data pre processing

The dataset consists of categorical variables such as race, marital status, A stage, T stage, N stage, differentiation of tumor, estrogen, and progesterone status, and numerical variables such as age, tumor size, and regional node. The data is split into numerical and categorical variables, and then the categorical variables are one-hot encoded, turning them into integer values that are processable by the NNs.

Variables

Independent variables:

Type of Neural network:
The ANN created by the submission to the dataset will be recreated using PyTorch



```
class SurvivalPredictionNN(nn.Module):
    def __init__(self, input_size, hidden_size1, hidden_size2, hidden_size3):
        super(SurvivalPredictionNN, self).__init__()
        self.fc1 = nn.Linear(input_size, hidden_size1)
        self.fc2 = nn.Linear(hidden_size1, hidden_size2)
        self.fc3 = nn.Linear(hidden_size2, hidden_size3)
        self.fc4 = nn.Linear(hidden_size3, 1)

    def forward(self, x):
        x = torch.relu(self.fc1(x))
        x = torch.relu(self.fc2(x))
        x = torch.relu(self.fc3(x))
        x = self.fc4(x)
        return x

# Define the function to train the model with different subsets of data
def RegressionNN(feature, data, train_sizes, batch_size=32, hidden_size1=64, hidden_size2=32, hidden_size3=16, lr=0.0001, num_epochs=50):
```

Fig 2.1: recreation of ANN using pytorch.

This code was adapted from the submitter. However, every NN approach to this problem uses a similar architecture, increasing its credibility.

This architecture will remain the same (number of layers and neurons per layer) for the FFN; however, the first layer will be modified to transform all the inputs going into it into sinusoidal components that are then passed to the hidden layers. The FFN will also be created using PyTorch and will have a Fourier series of order one due to the computational power available

```
class SurvivalPredictionNN(nn.Module):
    def __init__(self, input_size, hidden_size1, hidden_size2,hidden_size3, fourier_order=1):
        super(SurvivalPredictionNN, self).__init__()
        self.fourier_order = fourier_order
        self.orders = torch.arange(1, fourier_order + 1).float().to(device)
        self.fc1 = nn.Linear(input_size * (2 * fourier_order + 1), hidden_size1)# first layer takes calls a function that transforms input to the fourier series
        self.fc2 = nn.Linear(hidden_size1, hidden_size2)
        self.fc3 = nn.Linear(hidden_size2, hidden_size3)
        self.fc4 = nn.Linear(hidden_size3, 1)

    def forward(self, x):
        x = x.unsqueeze(-1)  # Add extra dimension for broadcasting
        fourier_features = torch.cat([torch.sin(self.orders * x), torch.cos(self.orders * x), x], dim=-1) # transforms input into fourier series
        fourier_features = fourier_features.view(x.shape[0], -1)
        x = torch.relu(self.fc1(fourier_features))
        x = torch.relu(self.fc2(x))
        x = torch.relu(self.fc3(x))
        x = self.fc4(x)
        return x

# Define the function to train the model with different subsets of data
def RegressionNN(feature, data, train_sizes, batch_size=32, hidden_size1=64, hidden_size2=32, hidden_size3=16, lr=0.0001, num_epochs=50):
```

Fig 2.2: creation of FFN using pytorch

Amount of training data:

The total training data will be taken, and the first 20% will be randomly selected and used to train both the models, then 40% will be randomly selected and used to train both the models, and so on, with 60%, 80%, and finally 100% of the training data available.

```
# Define train sizes and target feature
train_sizes = [0.2, 0.4, 0.6, 0.8, 1.0]  # Proportions of the total data
feature = 'Survival Months'  # Target column


data = data_preprocessed_df

# Call the function
results = RegressionNN(feature, data, train_sizes)
```

Fig 2.3: how training data is partitioned.

Dependent variables (DV's):

There are two different dependent variables - the final accuracy of the models and the change in accuracy as training data is increased calculated for the two different models. The final accuracy of the models will be checked by measuring the RMSE values of both models after training with 100% of the training data. These two values can then be compared.

The change in accuracy will be measured by taking a baseline RMSE value when both the models are trained using 20% of the data; as the training data increases, the increase in accuracy will be calculated by the formula in Fig 2.4, where N is the percentage of total training data.

$$\frac{\left(RMSE_{20\%} - RMSE_{N} \right)}{RMSE_{20\%}} \cdot 100$$

Fig 2.4

The RMSE values will be calculated using the code shown in fig 2.5

```
    # Evaluate on test data after training
    model.eval()
    test_loss = 0.0
    predictions = []
    with torch.no_grad():
        for inputs, labels in test_loader:
            inputs, labels = inputs.to(device), labels.to(device)
            outputs = model(inputs)
            test_loss += criterion(outputs, labels).item() * inputs.size(0)
            predictions.extend(outputs.squeeze(1).cpu().tolist())

    test_loss /= len(test_loader.dataset)
    print(f'Test Loss: {test_loss:.4f}')

    predictions_np = np.array(predictions) # creates a prediction array with the values it predicts
    y_test_np = y_test.cpu().numpy()
    mse = mean_squared_error(y_test_np, predictions_np) # checks against the array with the actual values (y_test_np) and calculates MSE
    rmse = np.sqrt(mse)
    mae = mean_absolute_error(y_test_np, predictions_np)

    print(f'MSE: {mse:.4f}, RMSE: {rmse:.4f}, MAE: {mae:.4f}')

    # Collect results
    results.append((train_size, mse, rmse, mae))

return results
```

Fig 2.5

Method:

Using the pre-processing steps, code shown, and the dependent and independent variables discussed, five trials for both models will be conducted, and the RMSE values will be put into a table. The average RMSE values will be calculated to reduce outliers and increase

precision, showing more accurate trends. The accuracy increase will be calculated on the average RMSE values. This table will be made for both different models.

Controlled variables:

Table 2 shows the variables that need to be kept constant so as not to affect the measurement of the DVs.

Table 2: Controlled variables

Variable	How its controlled	Function	What DV it affects
Architecture	The number of hidden layers will be 3 for both models, number of neurons will remain the same as well (64,32,16)	More neurons and layers add more parameters and can lead to differing accuracies	Both
Training, test and validation data	(65% training, 25% test and 15% validation (random seed = 42)	Models with more training data will have more accuracy	Both
Dataset	SEER breast cancer dataset will be used	Different datasets will have different features and quality of data provided	Both
Python and Pytorch version	Python 3.12 PyTorch 2.4.0	Different versions have differences in functionality	Both
Epochs	Both trained to 50 epochs	How many weight updates occur	Both
Loss function	Both use MSE	Optimization function	Both
Order of Fourier series	Code will only transform to order 1	More orders are computationally expensive and will change results	Both

Hypothesis:

Based on the above research It can be hypothesized that in terms of final accuracy, the FFN will have a lower RMSE value than the ANN, therefore being more accurate; this will happen because the FFN will be able to filter out the central relationship between the features and how they affect the survival months from the other noise in the data that would hinder the ANN. The FFN will also have improved feature extraction. However, during lower training data values, the ANN will be more accurate as the FFN may start to overfit. The ANN may have a more gradual accuracy increase as more training data is used, but the FFN will stop overfitting after a certain amount of data and increase its accuracy faster.

Results:

Table 3: RMSE values of ANN

Percentage of total training data (%)	RMSE values after 50 epochs						Average accuracy increase (%)
	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5	Average	
20	0.7931	0.8269	0.8134	0.8007	0.8319	0.8132	-
40	0.7797	0.7527	0.7413	0.7752	0.7631	0.7624	6.247
60	0.7539	0.7457	0.7389	0.7564	0.7331	0.7456	8.313
80	0.7461	0.7325	0.7337	0.7223	0.7259	0.7321	9.93
100	0.7119	0.7405	0.7237	0.7293	0.7401	0.7291	10.34

Table 4: RMSE values of FFN

Percentage of total training data (%)	RMSE values after 50 epochs						Average accuracy increase
	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5	Average	
20	0.9017	0.8921	0.9145	0.8713	0.8839	0.8927	-
40	0.8369	0.8429	0.8257	0.8031	0.8179	0.8253	7.550

60	0.7419	0.7665	0.7931	0.7787	0.7533	0.7667	14.11
80	0.7430	0.7341	0.7237	0.7019	0.7133	0.7232	18.99
100	0.7140	0.6863	0.6989	0.6731	0.6701	0.6862	23.13

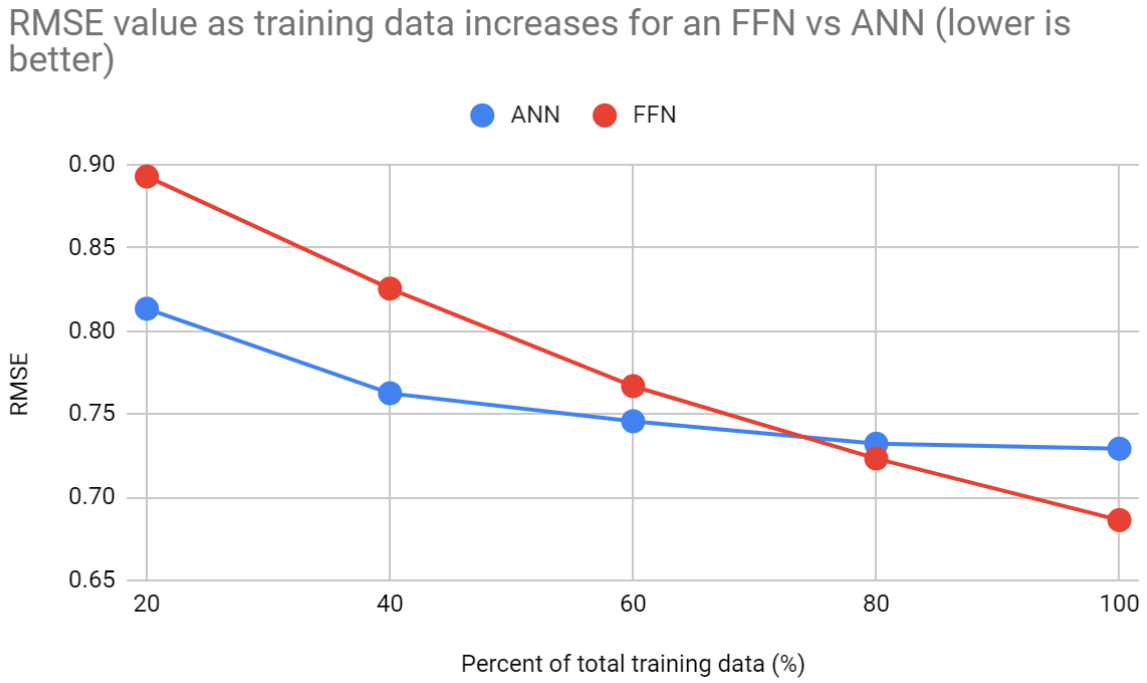


Fig 2.6

Analysis

Overall accuracy

In terms of overall accuracy, the FFN with 100% of the training data was the most accurate model, achieving an average RMSE value of 0.6862 and the lowest RMSE of 0.6701(table 4). The ANN achieved an average RMSE value of 0.7291 with 100% of the training data and the lowest RMSE of 0.7237 (table 3). Comparing the average RMSE values, the FFN was around 5.88% more accurate than the ANN. This is a non-negligible amount and can be attributed to the use of the FFN. looking at the shape of the graph (fig 2.6), the ANN seems to start plateauing at the end, signifying that if there were more data available, the accuracy

would not increase as drastically compared to the FFN, which shows signs of improvement if more data is available, which could make the overall accuracy even higher compared to the ANN. This increase in overall accuracy can be attributed to the FFN having better feature extraction, obtaining valuable trends in the data that the ANN could not recognize, and having a more precise relationship between the data itself without any redundant information affecting it.

As training data increases

Both models significantly improve their accuracy as training data increases from 20% of the total training data to 100%. The ANN improved by 10.34% overall, and the FFN improved by 23.13%. This was expected as training data should increase accuracy; however, the improvement of the FFN is quite surprising as it is drastically more significant than the improvement shown by the ANN. However, looking at the graph, the ANN is more accurate than the FFN until around 75% of the training data is used. At 20% of the total training data, the ANN is, on average, 8.9% more accurate than the FFN. This is due to the FFN overfitting, as when the training data is too low, the increased amount of features cannot be generalized to fit unseen data. Earlier, it was discovered that FFNs are prone to overfitting, and not having enough data exacerbated this issue. As the training data increased to 40% and 60%, the ANN was only 7.62% and 2.75%, respectively, signifying that the FFN was increasing its accuracy drastically as training data improved. In around 75% of the training data, the FFN seems to have stopped overfitting, leading to 80% of the training data, where the FFN was 1.22% more accurate. With the overall accuracy being 5.88 % more accurate than the ANN, the FFN's accuracy seems to be on an upward trend and could get even more accurate than the ANN.

.

Evaluation of the method and extensions:

The method answered the RQ to a satisfactory degree by considering both overall accuracy and the accuracy as training data increases. Finding that ANNs are better for less data while FFNs are better if there is more data available, the method had some potential areas for improvement.

Firstly, the most important limitation of ML is the lack of data. Data augmentation or synthetic data creation could have been used to increase the amount of data and make it more varied, improving model accuracies further and having a greater range of data to see changes as training data increases.

Secondly, the RQ could be answered in a more general way, using multiple datasets available involving different features to generalize the trend across m.

Some extensions could also be made, such as measuring the accuracy as the order of the Fourier featured network increases. The nature of the problem could also be changed to a classification task instead of a prediction one, and the accuracy could also be measured for image-related tasks, adding another layer of complexity.

Conclusion

Overall, the results mostly support the hypothesis, and it can be generalized that, on average, the FFN is more accurate given that enough training data is provided, while the ANN is more accurate if there is a shortage of training data and should be preferred in those scenarios. This paper helps improve the understanding of ANNs and FFNs and successfully investigates ways of improving a model's accuracy.

Bibliography

Admin. “Fourier Series - Definition, Formula, Applications and Examples.” *BYJUS*, BYJU’S, 22 Sept. 2022, byjus.com/maths/fourier-series/. Accessed 13 Aug. 2024.

alooba. “Cost Functions.” *Everything You Need to Know When Assessing Cost Functions Skills*, www.alooba.com/skills/concepts/machine-learning/cost-functions/#:~:text=During%20the%20training%20phase%2C%20the,minimizing%20errors%20and%20improving%20accuracy. Accessed 16 Aug. 2024.

“Artificial Neural Network.” *Artificial Neural Network - an Overview | ScienceDirect Topics*, www.sciencedirect.com/topics/earth-and-planetary-sciences/artificial-neural-network#:~:text=Artificial%20neural%20networks%20are%20a,concepts%20from%20biological%20neural%20systems. Accessed 14 Aug. 2024.

Baheti, Pragati. “The Essential Guide to Neural Network Architectures.” *V7*, www.v7labs.com/blog/neural-network-architectures-guide. Accessed 14 Aug. 2024.

Bhargav, Written by: Nikhil. “Neurons in Neural Networks.” *Baeldung on Computer Science*, 18 Mar. 2024, www.baeldung.com/cs/neural-networks-neurons. Accessed 14 Aug. 2024.

Boesch, Gaudenz. “A Complete Guide to Image Classification in 2024.” *Viso.Ai*, 19 June 2024, viso.ai/computer-vision/image-classification/#:~:text=Image%20Classification%20is%20the%20Basis%20of%20Computer%20Vision,-The%20field%20of&text=Image%20cla

ssification%20applications%20are%20used,%2C%20machine%20vision%2C%20and%20more. Accessed 13 Aug. 2024.

“Breast Cancer.” *World Health Organization*, www.who.int/news-room/fact-sheets/detail/breast-cancer. Accessed 14 Aug. 2024.

Brown, Sara. “Machine Learning, Explained.” *MIT Sloan*, 21 Apr. 2021, mitsloan.mit.edu/ideas-made-to-matter/machine-learning-explained. Accessed 13 Aug. 2024.

Brownlee, Jason. “Neural Networks Are Function Approximation Algorithms.” *MachineLearningMastery.Com*, 26 Aug. 2020, machinelearningmastery.com/neural-networks-are-function-approximators/#:~:text=Neural%20networks%20are%20an%20example%20of%20a%20supervised%20learning%20algorithm,error%20during%20the%20training%20process. Accessed 13 Aug. 2024.

Databricks. “What Are Machine Learning Models?” *Databricks*, www.databricks.com/glossary/machine-learning-models. Accessed 13 Aug. 2024.

DeepAI. “Hidden Layer.” *DeepAI*, 17 May 2019, deepai.org/machine-learning-glossary-and-terms/hidden-layer-machine-learning. Accessed 14 Aug. 2024.

Fourier Series Approximation, space.mit.edu/RADIO/CST_online/mergedProjects/DES/analysis/simulation/tasks/transienttask/fourier_series_approximation.htm. Accessed 13 Aug. 2024.

Frost, Jim. “Mean Squared Error (MSE).” *Statistics By Jim*, 28 May 2023, statisticsbyjim.com/regression/mean-squared-error-mse/. Accessed 16 Aug. 2024.

Garden , Emergent. “Watching Neural Networks Learn.” *YouTube*, 17 Aug. 2023, www.youtube.com/watch?v=TkwXa7Cvfr8. Accessed 14 Aug. 2024.

Gershenson, Carlos. “Artificial Neural Networks for Beginners.” *arXiv.Org*, 20 Aug. 2003, arxiv.org/abs/cs/0308031. Accessed 14 Aug. 2024.

Gillian , Nicholas. “(PDF) Gesture Recognition for Musician Computer Interaction.” *Researchgate*, Apr. 2011, www.researchgate.net/publication/230691317_Gesture_Recognition_for_Musician_Computer_Interaction. Accessed 14 Aug. 2024.

IBM. “What Is Gradient Descent?” *IBM*, 7 Oct. 2021, www.ibm.com/topics/gradient-descent#:~:text=Similar%20to%20finding%20the%20line,direction%20and%20a%20learning%20rate. Accessed 16 Aug. 2024.

Kaushik, Himanshu. “Machine Learning vs Human Learning: The Battle for Future Dominance.” *Medium*, 21 Sept. 2023, medium.com/@himanshubangalore/machine-learning-vs-human-learning-the-battle-for-future-dominance-fa7a1c99cd0c#:~:text=This%20has%20led%20to%20groundbreaking,and%20improved%20decision%2Dmaking%20processes. Accessed 13 Aug. 2024.

Kodad, Yassine. “Why Discrete Fourier Transformation over Continuous Fourier Transformation in Machine Learning.” *Medium*, 7 Oct. 2023, medium.com/@serene_mulberry_tiger_125/why-discrete-fourier-transformation-over-continuous-fourier-transformation-in-machine-learning-ab042e4a7294#:~:text=Machine%20Learning%20and%20Pattern%20Recognition,and%20grouping%20similar%20data%20points. Accessed 24 Aug. 2024.

“Machine Learning: What It Is and Why It Matters.” *SAS*, www.sas.com/en_sg/insights/analytics/machine-learning.html#:~:text=Machine%20lea

rning%20is%20a%20method,decisions%20with%20minimal%20human%20interventio n. Accessed 13 Aug. 2024.

Marsden, Jerrold E., and Anthony Tromba. “Vector Calculus.” *Https://Universitytime.Home.Blog*, W.H. Freeman, 2012, universitytime.home.blog/wp-content/uploads/2020/04/jerrold-e.-marsden-anthony-tro mba-vector-calculus.pdf. Accessed 16 Aug. 2024.

McClelland, James L., and David E. Rumelhart. “Chapter 10: The Appeal of Parallel Distributed Processing.” *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, MIT Press, 1986, web.stanford.edu/~jlmcc/papers/PDP/Volume%201/Chap10_PDP86.pdf. Accessed 14 Aug. 2024.

Rabinovich, Rafael V. “Understanding Neural Networks: How Neurons Capture and Process Information.” *Medium*, 6 Sept. 2023, medium.com/@rafvrab/understanding-neural-networks-how-neurons-capture-and-proc ess-information-ba8043569edf#:~:text=Summation%20Function%3A%20A%20mathe matical%20operation,into%20a%20sigmoid%2Dshaped%20curve. Accessed 14 Aug. 2024.

Ramachandran, Prajit, et al. “Searching for Activation Functions.” *Arxiv.Org*, 27 Oct. 2017, arxiv.org/pdf/1710.05941. Accessed 14 Aug. 2024.

“Risk Factors.” *National Breast Cancer Foundation*, 1 Aug. 2024, www.nationalbreastcancer.org/breast-cancer-risk-factors/#:~:text=Genetic%20Risk%2 0Factors,-Genetic%20risk%20factors&text=Race%3A%20Breast%20cancer%20is%2 0diagnosed,breast%20cancer%20in%20the%20future. Accessed 14 Aug. 2024.

Rouse, Margaret. “What Is an Input Layer? - Definition from Techopedia.” *Technopedia*, 31 May 2018,

www.techopedia.com/definition/33262/input-layer-neural-networks. Accessed 14 Aug. 2024.

Ryan, Muhammad. “How Neural Networks ‘Learn.’” *Medium*, Towards Data Science, 18 Oct. 2019, towardsdatascience.com/how-neural-network-learn-3b56c175b5ca. Accessed 14 Aug. 2024.

Sahay, Milind. “Neural Networks and the Universal Approximation Theorem.” *Medium*, Towards Data Science, 13 Mar. 2021, towardsdatascience.com/neural-networks-and-the-universal-approximation-theorem-8a389a33d30a. Accessed 14 Aug. 2024.

Sanad, Mohdsanadzakirizvi@gmail.com. “Image Classification Using CNN.” *Analytics Vidhya*, 22 July 2024, www.analyticsvidhya.com/blog/2020/02/learn-image-classification-cnn-convolutional-neural-networks-3-datasets/. Accessed 13 Aug. 2024.

Sanderson, Grant. “Gradient Descent, How Neural Networks Learn | Chapter 2, Deep Learning.” *YouTube*, 3Blue1Brown, 17 Oct. 2017, www.youtube.com/watch?v=IHZwWFHWa-w. Accessed 16 Aug. 2024.

Sarita, PhD. “Basic Understanding of Neural Network Structure.” *Medium*, 3 Oct. 2023, medium.com/@sarita_68521/basic-understanding-of-neural-network-structure-eecc8f149a23. Accessed 15 Aug. 2024.

Sharma, Siddharth Sharma, et al. “Activation Functions in Neural Networks.” *Ijeast.Com*, 2020, www.ijeast.com/papers/310-316,Tesma412,IJEAST.pdf. Accessed 14 Aug. 2024.

Sweta. “Why Do We Need Non Linear Activation Function?” *Medium*, 19 Aug. 2020, sweta-nit.medium.com/why-do-we-need-non-linear-activation-function-187e5eb88a19. Accessed 15 Aug. 2024.

Szandała, Tomasz. “Review and Comparison of Commonly Used Activation Functions for Deep Neural Networks.” *Studies in Computational Intelligence*, 22 July 2020, pp. 203–224, doi:10.1007/978-981-15-5495-7_11.

Tancik, Matthew, et al. “Fourier Features Let Networks Learn High Frequency Functions in Low Dimensional Domains.” *arXiv.Org*, 18 June 2020, arxiv.org/abs/2006.10739. Accessed 13 Aug. 2024.

Vardi, Gal, et al. “Learning a Single Neuron with Bias Using Gradient Descent.” *Advances in Neural Information Processing Systems*, 6 Dec. 2021, proceedings.neurips.cc/paper/2021/hash/f0f6cc51dacebe556699ccb45e2d43a8-Abstract.html. Accessed 14 Aug. 2024.

Vivek, Sandeep. “Introductory Guide on the Activation Functions.” *Analytics Vidhya*, 7 Mar. 2022, www.analyticsvidhya.com/blog/2022/03/introductory-guide-on-the-activation-functions/. Accessed 15 Aug. 2024.

Watson, Stephanie. “Which Cancers Take the Most Lives?” *WebMD*, www.webmd.com/cancer/leading-causes-cancer-deaths. Accessed 14 Aug. 2024.

Weisstein, Eric W. “Fourier Series.” *From Wolfram MathWorld*, mathworld.wolfram.com/FourierSeries.html. Accessed 24 Aug. 2024.

“What Is a Feature in Machine Learning and Data Science?: Domino.” *Domino Data Lab*, domino.ai/data-science-dictionary/feature. Accessed 16 Aug. 2024.

"What Is Supervised Learning?" *IBM*, 23 Sept. 2021, www.ibm.com/topics/supervised-learning. Accessed 13 Aug. 2024.

Ye, Andre. "If Rectified Linear Units Are Linear, How Do They Add Nonlinearity?" *Medium*, Towards Data Science, 2 Sept. 2020, towardsdatascience.com/if-rectified-linear-units-are-linear-how-do-they-add-nonlinearity-40247d3e4792. Accessed 15 Aug. 2024.