Regression and classification with FFNN
FYS-STK3155 - Project 2

Herman Nissen-Sollie
University of Oslo

(Dated: October 31, 2025)

Feed-forward neural networks (FFNN) represent a major step from traditional machine learning to deep learning, enabling models of greater complexity through an architecture inspired by the human brain[1]. FFNNs can be applied to both regression and classification tasks, allowing them to model relationships between variables, predict unseen outcomes, and interpret or classify image data. In this study, we implement and analyze FFNNs for both regression and classification problems, focusing on how architectural choices, such as the number of layers and hyperparameters, affect model performance. For regression, we compare the FFNN results to traditional machine learning approaches using a synthetic dataset generated from the Runge function, known for its numerical complexity. For classification, we employ a subset of the "Quick, Draw!" dataset, an open collection of user-generated sketches representing everyday objects. The results, which will be evaluated and discussed in relation to theoretical expectations and previous studies, aim to provide deeper insight into how FFNNs perform across problem domains and how their design can be optimized for different learning tasks. Future work will consider further extensions and refinements of the network architecture.

I. INTRODUCTION

Neural networks, inspired by the structure of the human brain, represent the key advancement that transforms traditional machine learning into deep learning. This addition greatly increases model complexity and, consequently, its potential.

Traditional machine learning models are often constrained by explicit feature design and statistical algorithms, whereas deep learning introduces multiple hidden layers that automatically learn abstract representations of the data. Sometimes, these will appear almost random, yet capturing patterns that simpler models cannot.

Feed-forward neural networks (FFNN) can be applied to both classification and regression tasks. FFNNs can model non-linear relationships, handle a large number of high-dimensional input variables, and enable a much more automated learning process.

The study by Sharifi et al. (2020) demonstrated how an FFNN predicted chronic kidney disease with a 98% classification accuracy using 12 input variables, outperforming or matching the best feature-engineered SVM variants[2]. This underlines the paradigm shift brought about by feed-forward neural networks, where feature extraction and classification are learned jointly rather than being engineered manually.

FFNN-based regression has also shown its strength in the 2022 study by Qin et al., which investigated compound bioactivity as background research for anti-breast-cancer drugs[3]. Their FFNN produced a quantitative prediction model relating molecular descriptor variables to bioactivity, achieving a loss function value of 0.0146.

In this work, we develop our own FFNN implementation and apply it to both regression and classification problems. The regression task will use a synthetically generated dataset based on the Runge function, a function known for its challenging behavior in numerical integration and approximation.

For the classification task, we use a subset of the "Quick, Draw!" dataset, which contains over 50 million sketches across more than 300 classes collected from a public drawing game[4]. The dataset created from the game was made into a competition by Google in 2018, with thousands of applicants. For the sake of simplicity, we focus on the same 10 classes studied by a group of master's students: Apple, Banana, Book, Fork, Key, Ladder, Pizza, Stop Sign, Tennis Racquet, and Wheel[5].

The overall goal of this work is to gain a deeper understanding of FFNNs. That being their architecture, application, and optimization, and to analyze their performance on both regression and classification problems.

Section II provides an overview of how FFNNs perform regression and classification, explains our network architecture and hyperparameters, how to evaluate our models, and introduces the two datasets in detail. Section III presents and discusses our results, comparing them with previous studies. Finally, Section IV summarizes the main findings and outlines potential directions for future work on FFNNs.

II. METHOD

A. FFNN

1. *Network architecture*

A Feed-Forward Neural Network (FFNN) is composed of layers of interconnected nodes, often referred to as neurons[1]. Each connection between two nodes carries a weight, which determines the strength and direction of the signal being transmitted. Every node also has an associated bias, a parameter that allows the activation threshold of the neuron to shift, enabling the model to fit data more flexibly.

The network consists of three types of layers: the input layer, which receives the numerical input data (for example, pixel intensities or measured values); one or more hidden layers, which process and transform the data through weighted connections and activation functions; and the output layer, which produces the final prediction. Information flows in one direction, from input to output, without feedback between layers. Through this structure of weights, biases, and activations, the FFNN is able to model complex relationships within the data.

2. Activation Functions

Activation functions introduce non-linearity into neural networks, allowing them to model complex patterns and relationships in data. Without activation functions, each layer would perform only linear transformations, and the entire network would behave as a single linear model. To overcome this limitation, activation functions are placed between layers to transform how node outputs are passed forward through the network.

Activation functions are often divided into locally quadratic and piecewise linear types[6]. The sigmoid function is a traditional example of a locally quadratic activation, compressing values smoothly into the range [0,1]. In contrast, piecewise linear activations such as the Rectified Linear Unit (ReLU) set negative inputs to zero while keeping positive values unchanged, enhancing the model's ability to detect meaningful variations. The Leaky ReLU is a variant that retains a small gradient for negative inputs, preserving information that would otherwise be lost using normal ReLU.

Due to their computational efficiency and strong empirical performance, ReLU-based activation functions have become the standard choice in most modern neural network architectures.

3. Learning process

The learning process of an Artificial Neural Network (ANN) consists of two main phases: forward propagation and backpropagation[7]. In the forward phase, input data are passed through the network from the input layer, through the hidden layers, and finally to the output layer, where predictions are produced. Each node computes a weighted sum of its inputs, adds a bias term, and applies an activation function that determines its output. The outputs from one layer serve as the inputs to the next, allowing information to move forward through the network.

During backpropagation, the network's predictions are compared to the true values using a cost function, and the resulting error is propagated backward through the layers. The algorithm calculates how much each weight contributed to the error and updates the weights accordingly to minimize it. Through repeated forward and backward passes, the network gradually adjusts its parameters to capture complex patterns in the data and improve prediction accuracy.

B. Regression

C. Classification

D. Dataset

1. The Runge function

2. The "Quick, Draw!" dataset

E. Implementation

1. Regression

2. Classification

III. RESULTS AND DISCUSSION

When utilizing the sigmoid activation function, our best result was 0.230. This was surprisingly the outcome with only one hidden layer, with just five nodes. Figure 1 shows how the architecture of the neural network affects the result. When comparing with our results from using the closed-form solution, we see that we have actually outperformed this result, being approximately 0.236. The results are similarly presented in Figure 2, but here

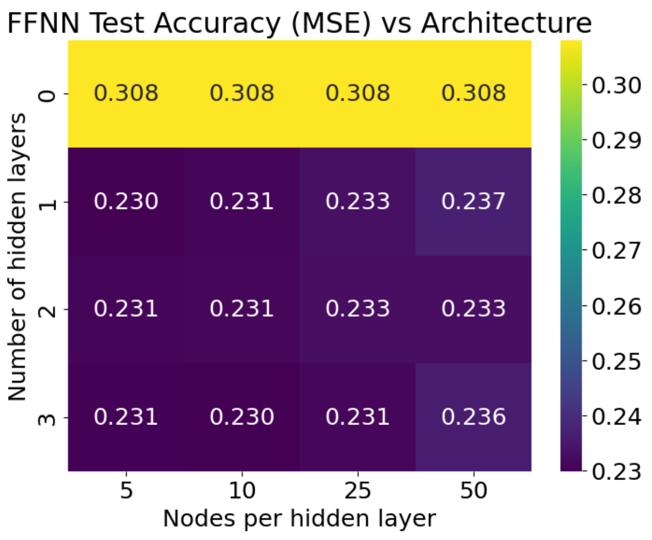

Figure 1: Accuracy from regression on the runge dataset with FFNN with the sigmoid activation function compared to the architecture of our neural network, testing [0, 1, 2, 3] hidden layers and [5, 10, 25, 50] nodes per hidden layer.

we have purposly tried to make the figure bad. We used a Qualatative colormap and a smaller font size, which makes the model hard to inturpret just by looking at the heatmap, and the numbers are offcourse hard to read themselves.

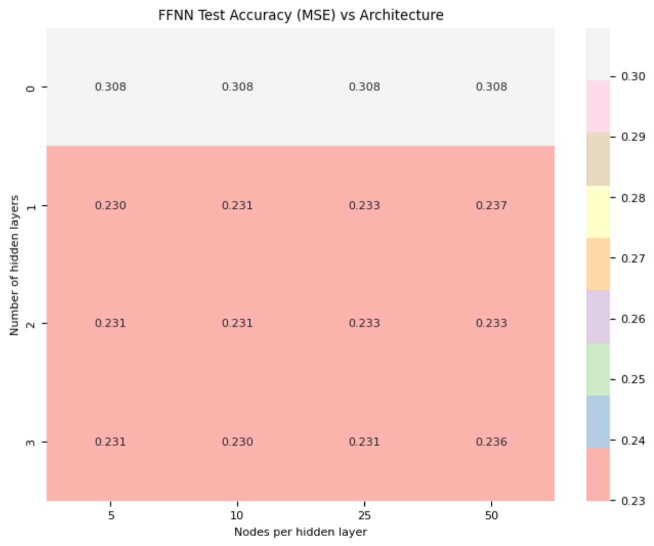

Figure 2: Accuracy from regression on the runge dataset with FFNN with the sigmoid activation function compared to the architecture of our neural network, testing [0, 1, 2, 3] hidden layers and [5, 10, 25, 50] nodes per hidden layer.

IV. CONCLUSION

[1] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning* (MIT Press, 2016), chapter 6: Deep Feedforward Networks, URL `https://www.deeplearningbook.org/`.

[2] A. Sharifi and K. Alizadeh, Annals of Military and Health Sciences Research **18**, e101585 (2020), accessed 2025-07-28, URL `https://doi.org/10.5812/amh.101585`.

[3] Y. Qin, C. Li, X. Shi, and W. Wang, Frontiers in Bioengineering and Biotechnology **10**, 946329 (2022), accessed 2025-07-28, URL `https://doi.org/10.3389/fbioe.2022.946329`.

[4] Kaggle, *Quick, draw! doodle recognition challenge*, Online competition and dataset (2018), https://www.kaggle.com/competitions/quickdraw-doodle-recognition (accessed 2025-10-28), URL `https://www.kaggle.com/competitions/quickdraw-doodle-recognition`.

[5] T. B. D. A. T. 10, *Deep learning for artificial intelligence project, group 10 (etsetb master students)*, Online report: “Quick, Draw! Doodle Recognition Challenge” (MLP, CNN, LSTM) (2018), available at https://telecombcn-dl.github.io/2018-dlai-team10/ (accessed 2025-10-31), URL `https://telecombcn-dl.github.io/2018-dlai-team10/`.

[6] Y. Wang, Y. Li, Y. Song, and X. Rong, Applied Sciences **10**, 1897 (2020), URL `https://doi.org/10.3390/app10051897`.

[7] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning* (MIT Press, 2016), chapters 6 and 8: Forward Propagation and Backpropagation, URL `https://www.deeplearningbook.org/`.