



UNIVERSITY OF OSLO

FYS-STK3155

---

## Project 2

---

Brage Wiseth  
Felix Cameren Heyerdahl  
Eirik Bjørnson Jahr

BRAGEWI@IFI.UIO.NO  
FELIXCH@IFI.UIO.NO  
EIRIBJA@IFI.UIO.NO

December 9, 2023

[HTTPS://GITHUB.COM/BRAGEWISETH/MACHINELEARNINGPROJECTS](https://github.com/bragewiseth/machinelearningprojects)

# Contents

Introduction . . . . .	3
1 Gradient Descent . . . . .	4
1.1 Backpropagation and Chain Rule . . . . .	5
2 Neural Networks . . . . .	6
3 Data . . . . .	7
4 Results and Analysis . . . . .	10
4.1 Hyperparameters . . . . .	10
4.2 Final Evaluation and Comparisons . . . . .	12
5 Conclusion . . . . .	13
<b>Appendix</b>	<b>14</b>
A Regression with Neural Networks . . . . .	14
B Universal Approximation . . . . .	19
Bibliography . . . . .	22

## Abstract

In this study, we explore the effectiveness of neural networks in solving classification and regression problems, contrasting their performance with traditional logistic regression models. Utilizing the Wisconsin breast cancer dataset, we compared the accuracy of these methods in tumor classification. Our results show that neural networks achieved a classification accuracy of 95%, compared to logistic regression's 96%. Both are similar to the 97% achieved using SKLearn's models. Additionally, we examined the application of neural networks to regression problems, finding that they can approximate 2-dimensional perlin noise, with a mean squared error of 0.012, compared to linear regression's 0.028. This shows that both neural networks and logistic regression are powerful tools for classification tasks, but that neural networks seems to be more viable for complex regression tasks. Offering a more flexible framework for approximating functions

**Keywords:** Regression, Classification, Neural Networks

## Introduction

Real-world applications often require accurate classification between different classes. A notable example is in cancer research, where it's crucial to classify tumors as either benign or malignant. This classification typically involves analyzing a set of features extracted from the tumor. However, these features may not always clearly distinguish between classes, particularly when they are similar for both or when there is a large number of features. This complexity can pose a challenge for human analysis. To address this, *logistic regression* is often employed. This method involves using regression to produce a continuous output, which is then transformed into a binary value through an activation function, such as the sigmoid function<sup>1</sup> or the Heaviside function. The resulting binary value represents the data point's classification. However, logistic regression has its limitations, particularly when dealing with data that is not linearly separable. In such cases, logistic regression cannot effectively draw a boundary to separate the classes, leading to classification errors. This issue is examined in more detail in Appendix B. As an alternative, *neural networks* offer a more versatile solution. Capable of approximating any continuous function with a sufficient number of neurons in the hidden layers[4], neural networks provide a robust tool for both classification and regression problems. Their ability to learn complex, non-linear relationships makes them particularly powerful for tasks where traditional logistic regression falls short. The application of neural networks in regression is explored in Appendix A.

In our initial project[9], we utilized an analytical approach to derive optimal parameters for linear regression. However, the introduction of activation functions in more complex models, such as neural networks, breaks this linearity. Consequently, the same analytical methods cannot be applied to determine optimal parameters. Instead, we shift towards an iterative approach. For cost functions that are continuous and differentiable, gradient descent becomes a viable method to locate the function's minimum. The models explored in this project predominantly rely on gradient descent.

The structure of this report is as follows:

- In Section 2, we introduce the logistic regression model.

---

1. The sigmoid function outputs a value between 0 and 1. By setting a threshold, such as 0.5, values above this threshold can be classified as 1, and those below as 0. This output can be interpreted as the probability of the data point belonging to class 1.

- In Section 3, we discuss gradient descent and backpropagation, key techniques in the optimization of machine learning models.
- In Section 4, we delve into the fundamentals of neural networks.
- Section 5 covers the data used in our experiments and analyses.
- Section 6 presents our results and provides a discussion on their implications.
- Section 7 concludes the report, summarizing key findings and insights.
- Appendix A explores regression using neural networks.
- Appendix B investigates the universal approximation theorem and its applications.

## 1. Gradient Descent

Gradient descent is an iterative optimization algorithm used to find the minimum of a function. The fundamental principle behind gradient descent is to iteratively move towards the minimum of the function by taking steps proportional to the negative of the gradient (or approximate gradient) at the current point. Unlike the Newton-Raphson method, which is used for finding the roots of a function and involves the second derivative, gradient descent primarily utilizes the first derivative. While Newton-Raphson converges faster under certain conditions due to its use of second-order information, gradient descent is more widely applicable as it does not require the computation of the second derivative, making it simpler and more computationally efficient in many scenarios. In the context of machine learning, gradient descent is employed to minimize a cost function, which is a measure of how far a model's predictions are from the actual outcomes. The algorithm updates the parameters of the model in a way that the cost function is gradually reduced.

There are several variants of gradient descent, each with its unique characteristics:

- **Batch Gradient Descent:** This form computes the gradient of the cost function with respect to the parameters for the entire training dataset. While this can be computationally expensive<sup>2</sup>, it produces a stable gradient descent convergence.
- **Stochastic Gradient Descent (SGD):** In contrast to batch gradient descent, SGD updates the parameters for each training example one by one. It can be faster and can also help to escape local minima, but the path to convergence can be more erratic.
- **Mini-batch Gradient Descent:** This is a compromise between batch and stochastic gradient descent. It updates the parameters based on a small, randomly selected subset of the training data. This variant provides a balance between the efficiency of SGD and the stability of batch gradient descent.

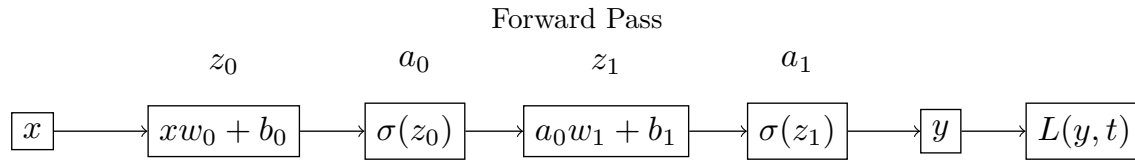
---

2. depending on the way computational resources are utilized, it is possible to parallelize the computation of the gradient for each training example, making it more efficient. For our case using batch gradient descent on a GPU is much faster than using stochastic gradient descent.

The choice of gradient descent variant and its parameters, like the learning rate, significantly impacts the performance and convergence of the algorithm. An appropriately chosen learning rate ensures that the algorithm converges to the minimum efficiently, while a poorly chosen rate can lead the algorithm to diverge or get stuck in a local minimum. Furthermore, advanced optimization algorithms based on gradient descent, such as Adam and RMSprop, have been developed to address some of the limitations of the traditional gradient descent method, offering adaptive learning rates and momentum features.

## 1.1 Backpropagation and Chain Rule

In machine learning, particularly in training neural networks, calculating derivatives is a fundamental process. For simpler models, such as linear regression, deriving these derivatives is relatively straightforward. We typically compute the gradient of the cost function with respect to the model parameters. However, when dealing with more complex models like neural networks, the process of calculating these derivatives becomes more intricate due to the architecture of the network. Neural networks consist of multiple layers of interconnected nodes, where each node represents a mathematical operation. The output of one layer serves as the input for the next, creating a chain of functions. When the loss function of the network is a composition of several such functions, applying the chain rule becomes essential for computing gradients. This process is known as backpropagation.



In backpropagation, to compute the derivative of the loss function with respect to a specific weight (e.g.,  $w_0$ ), it is necessary to trace the path of influence of that weight through all the functions in the network. This is done by applying the chain rule in a reverse manner, starting from the output layer back to the input layer.

Backward Pass

$$\boxed{\frac{\partial L}{\partial w_0}} = \boxed{\frac{\partial L}{\partial a_1}} \cdot \boxed{\frac{\partial a_1}{\partial z_1}} \cdot \boxed{\frac{\partial z_1}{\partial a_0}} \cdot \boxed{\frac{\partial a_0}{\partial z_0}} \cdot \boxed{\frac{\partial z_0}{\partial w_0}}$$

Figure 1: Illustration of the Chain Rule in Forward and Backward Passes

Key Concepts in Backpropagation:

- **Forward Pass:** In the forward pass, inputs are passed through the network, layer by layer, to compute the output. Each node's output is a function of its inputs, which are the outputs of nodes in the previous layer.
- **Backward Pass:** The backward pass is where backpropagation is applied. After computing the loss (the difference between the predicted output and the actual out-

put), we propagate this loss backward through the network. This involves applying the chain rule to compute partial derivatives of the loss with respect to each weight in the network.

- **Gradient Descent:** The gradients computed through backpropagation are used to update the weights of the network. This is typically done using gradient descent or one of its variants, where the weight update is proportional to the negative gradient, aiming to minimize the loss function.
- **Activation Functions:** The role of activation functions in each neuron is crucial. They introduce non-linearities into the model, allowing neural networks to learn complex patterns. Derivatives of these activation functions play a significant role in the backpropagation process.

By iteratively performing forward and backward passes and updating the model parameters using gradient descent, neural networks can effectively learn from data. Backpropagation is thus a cornerstone technique in neural network training, enabling these models to capture and learn from complex patterns in data.

## 2. Neural Networks

Neural networks extend the principles of logistic regression to more complex architectures, enabling the modeling of a wider range of nonlinear relationships. At their core, neural networks can be conceptualized as a series of logistic regression models interconnected in a network structure. This similarity allows for a degree of code reusability across logistic regression, linear regression, and neural network implementations.

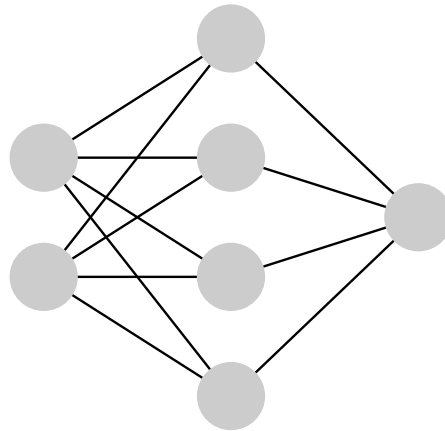


Figure 2: A model of a neural network with one hidden layer consisting of four nodes [5]

One of the defining features of neural networks is the use of activation functions. These functions introduce non-linearity into the model, allowing the network to learn complex patterns. Common examples of activation functions include the sigmoid, tanh, and ReLU

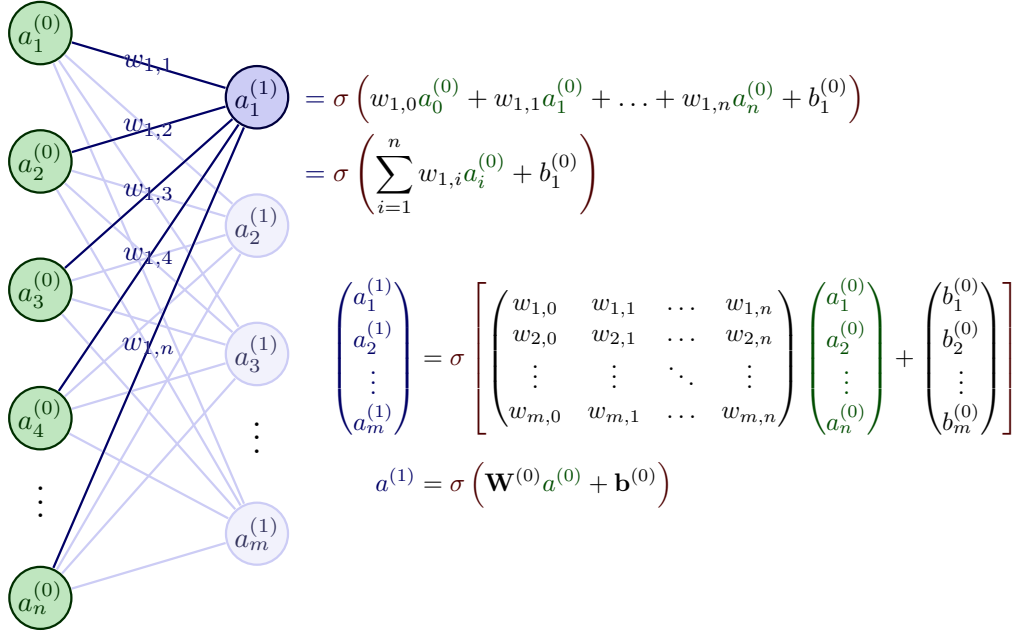


Figure 3: Illustration of how a single node gets its value during a forward pass. This shows that matrix operations can be used for more efficient calculations. When  $a$  is a batch of inputs, these calculations become matrix-matrix multiplications, significantly speeding up both training and inference.

[5]

functions. Each has unique characteristics that make them suitable for different types of neural network architectures. In a neural network with no hidden layers, the model essentially becomes a linear regression model, represented by the equation  $y = x_0w_0 + x_1w_1 + x_2w_2 + \dots + x_nw_n + b_0$ . By introducing an activation function at the output layer, the model transforms into a logistic regression model, suitable for binary classification tasks. The flexibility of neural networks arises from their ability to incorporate multiple hidden layers with a variety of activation functions, enabling them to capture complex relationships in data. Backpropagation is the mechanism through which neural networks learn. By calculating the derivatives of the loss function with respect to the network's parameters, backpropagation allows for the adjustment of these parameters in a way that minimizes the loss. This process is essential for training neural networks and is applicable across different network architectures, from simple single-layer networks to deep, multi-layered structures.

### 3. Data

For our classification task, we will utilize the widely recognized Wisconsin Breast Cancer Dataset[10]. This dataset comprises 569 data points, each with 30 distinctive features. These features are derived from digitized images of a fine needle aspirate (FNA) of breast masses, focusing on various characteristics of the cell nuclei depicted in the images. The

dataset quantifies several attributes for each cell nucleus, including radius, texture, perimeter, area, smoothness, compactness, concavity, concave points, symmetry, and fractal dimension. For each attribute, three types of measurements are provided: the mean, standard error, and the "worst" or largest (which represents the mean of the three largest values). This results in a total of 30 features for each data point. For instance, field 1 represents the Mean Radius, field 11 is Radius SE, and field 21 corresponds to the Worst Radius. All feature values in this dataset are recorded with four significant digits, and there are no missing attribute values. The class distribution within the dataset is as follows: 357 benign cases and 212 malignant cases. The primary objective is to classify these tumors as either benign or malignant based on the features provided. In this context, a positive result indicates a benign tumor, while a negative result signifies a malignant tumor. Essentially, our goal is to accurately identify cases of cancer and classify them as negative.

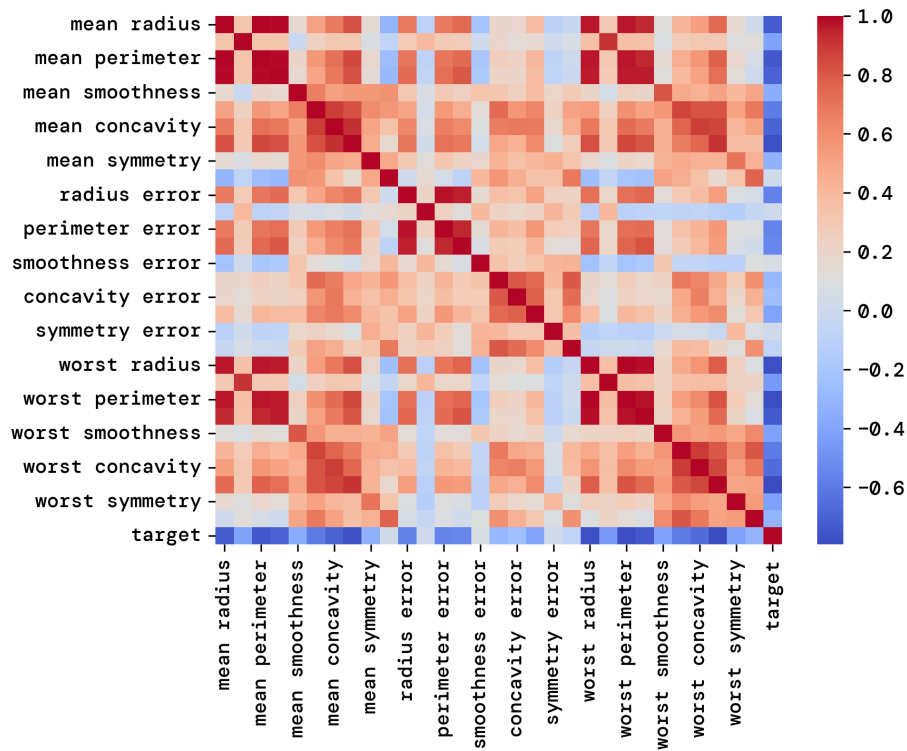


Figure 4: Feature correlation matrix. This provides insight into the redundancy among features. Only every other feature is annotated, but the missing labels can be inferred from Figure 5.



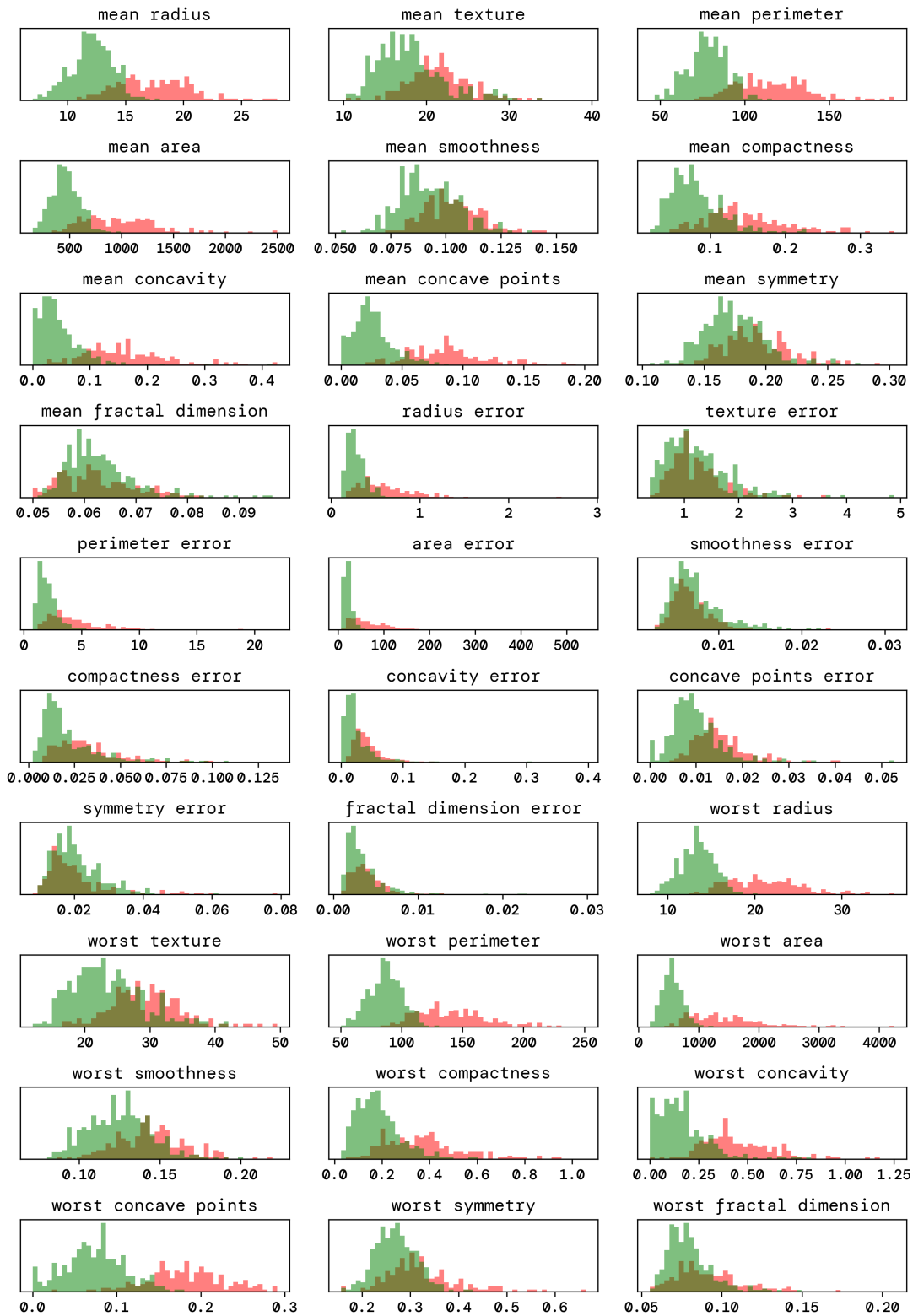


Figure 5: Feature histogram. The red class (0) represents malignant cases, while the green class (1) indicates benign cases.

## 4. Results and Analysis

In our experiments, we employed the cross-entropy loss function, The gradient of the loss function with respect to the model's parameters was computed using automatic differentiation. We used the SKLearn[8] library to perform cross-validation for a more robust evaluation of our models. We used k-fold cross-validation with  $k = 6$  folds. The results presented in this section are based on the average of the results from the 6 folds. The bar plots also show the standard deviation of the results from the 6 folds.

### 4.1 Hyperparameters

During our hyperparameter tuning phase, we observed interdependencies between certain hyperparameters. Notably, the learning rate and the number of epochs showed a significant correlation, as did the batch size and the learning rate. Such dependencies imply that these hyperparameters cannot be optimized independently for the most effective training process. Ideally, a comprehensive grid search across multiple dimensions of hyperparameters would be conducted. However, due to constraints in time and computational resources, our experiments were limited. We were able to conduct grid searches, but these were restricted to two dimensions at most. While this limitation prevented a thorough exploration of the hyperparameter space, the results from these partial grid searches provided valuable insights. They served as indicators, pointing us towards regions in the hyperparameter space where optimal settings might exist. The analysis of these results suggests that while we have identified promising hyperparameter settings, a more exhaustive search might yield further improvements. We use the same base hyperparameters when we tune the other hyperparameters.

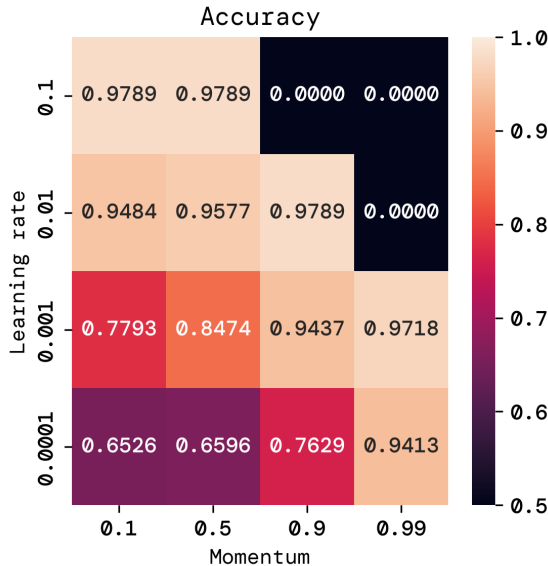


Figure 6: Accuracy versus learning rate and momentum.

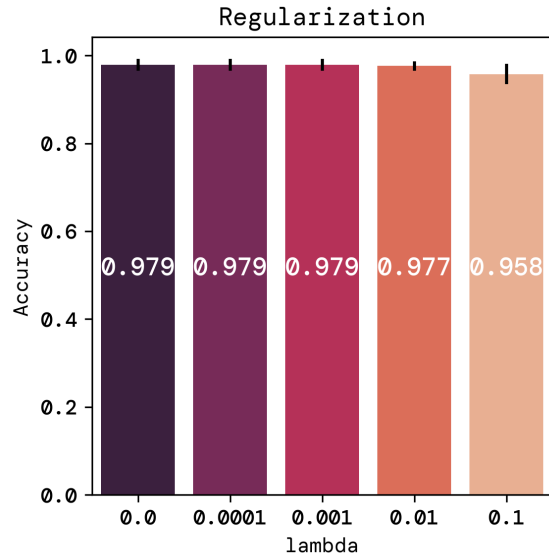


Figure 7: Accuracy versus regularization.

Our findings indicate that incorporating momentum is beneficial, particularly with a value of around 0.9, which emerged as the most effective in our tests. This momentum allows for a higher learning rate, enhancing the training process. In contrast, excessive regularization seems to adversely affect model performance.

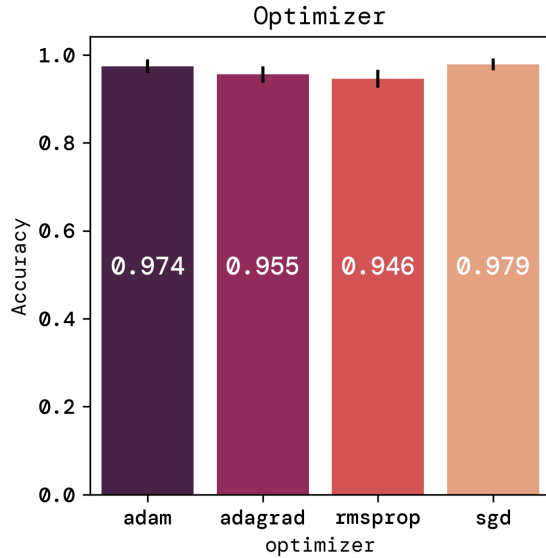


Figure 8: Model accuracy across different optimizers. here sgd is standard gradient descent.

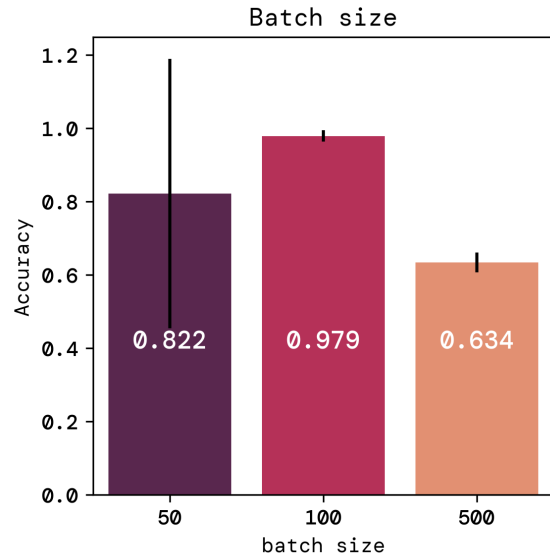


Figure 9: Accuracy based on batch size.

The choice of optimizer did not significantly impact the model's performance. However, Adam and standard gradient decent with momentum showed a slight advantage over others. The batch size also played a role, with certain sizes yielding better accuracy. With a batch size of 50, the model accuracy varied greatly between folds.

In order for our model to correctly classify the data, it is essential that the last layer's activation function is a sigmoid function. This is because the sigmoid function outputs a value between 0 and 1, which can be interpreted as the probability of the data point belonging to class 1. The choice of activation function for the hidden layers did not significantly impact the model's performance. We used one hidden layer with 2 nodes when testing activation functions.

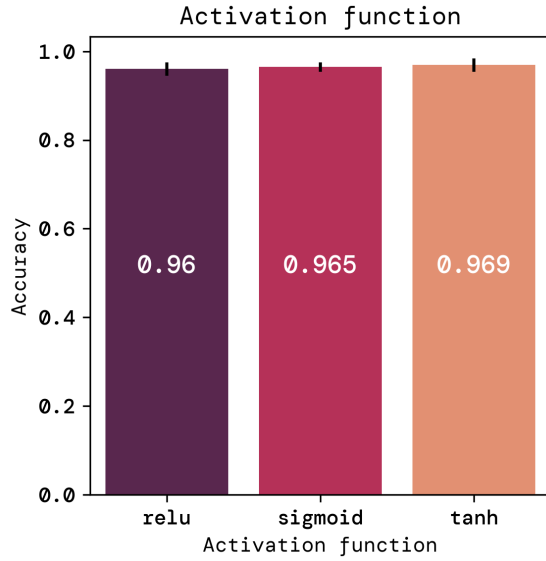


Figure 10: Model performance with different activation functions.

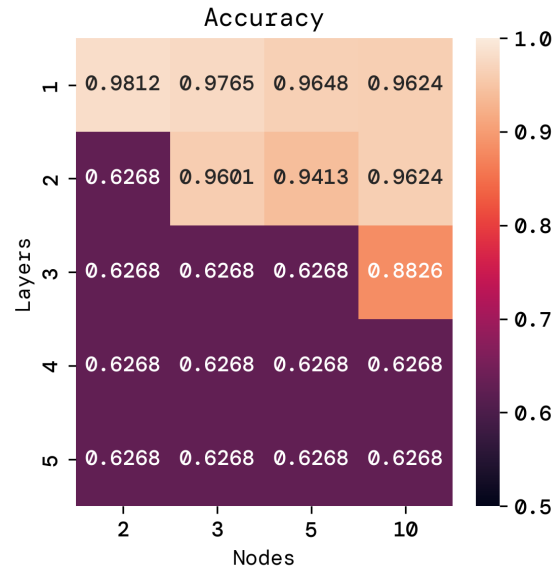


Figure 11: Accuracy in relation to the number of layers and nodes.

## 4.2 Final Evaluation and Comparisons

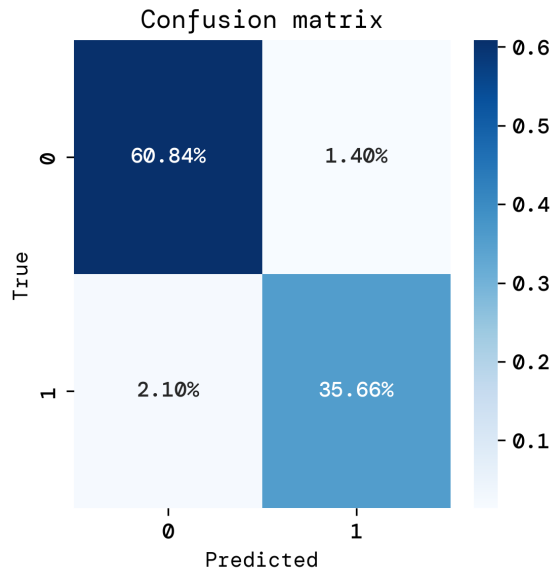


Figure 12: Confusion matrix for our model.

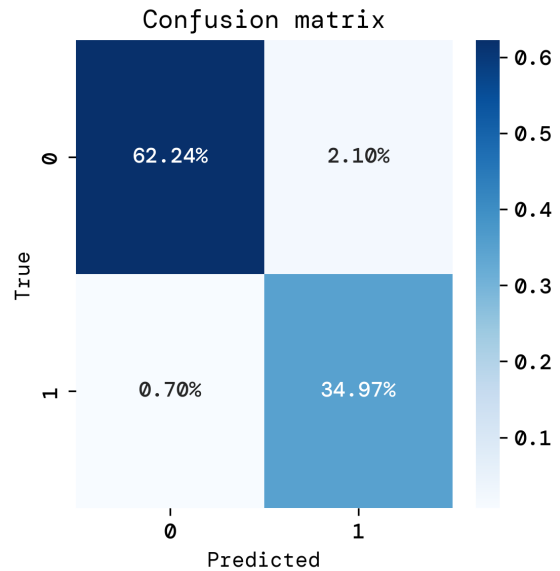


Figure 13: Confusion matrix for SKLearn's model.

In our classification task, logistic regression achieved an accuracy of 96% on the test set, while our neural network model achieved 95%. This performance is comparable to the 96% accuracy obtained with SKLearn's logistic regression model. It's important to consider that this comparison may not be entirely direct due to differences in hyperparameters between the models. Nevertheless, efforts were made to align the hyperparameters as closely as possible. An interesting observation is that increasing the number of layers and nodes in the neural network did not significantly enhance its performance. This suggests the possibility of overfitting to the training data, indicating that a simpler model could be more effective or that additional regularization techniques might be required.

## 5. Conclusion

This project has delved into the capabilities of neural networks in addressing classification challenges, comparing them with traditional logistic regression models. Using the Wisconsin breast cancer dataset, we found that neural networks, with a classification accuracy of 95%, perform nearly as well as logistic regression models, which achieved 96%. This result is in line with the 97% accuracy achieved using SKLearn's implementation. These findings highlight the competitive nature of neural networks in classification tasks, even when compared to more traditional models. However, they also underscore the importance of careful model selection and hyperparameter tuning in achieving optimal performance. While neural networks offer flexibility and power, they may not always outperform simpler models, especially in cases where the underlying data patterns are not exceedingly complex. Future work could explore further optimization of the neural network architecture and hyperparameters, as well as the application of these models to more diverse datasets to fully assess their generalizability and efficacy in various classification scenarios.

# Appendix A

## Appendix A. Regression with Neural Networks

In our previous project[9], we investigated linear models and polynomials for data fitting, deriving optimal parameters ( $\beta$ ) analytically. While this method theoretically allows polynomials to approximate any function providing the model with a high degree of freedom, it faces practical limitations such as computational constraints and overfitting risks. An alternative approach is using neural networks, which offer a flexible architecture to approximate a wide range of functions. Unlike polynomials, neural networks are composed of simpler, interconnected components, allowing them to model complex relationships within the data for both regression and classification tasks.

### Data

Our dataset for this study was generated using Perlin noise, a gradient noise function commonly used in computer graphics to produce natural-looking randomness. This approach allows us to create a challenging dataset for regression analysis, enabling us to evaluate the adaptability of our models to complex data patterns.

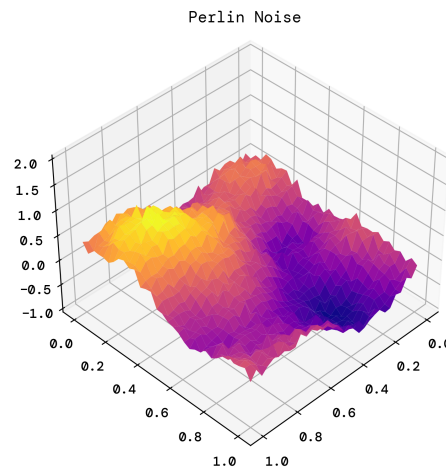


Figure 14: Sample visualization of Perlin noise. This noise function generates a variety of gradients, creating a natural, smooth variation in data.

## Results and Analysis

For regression, our approach mirrors that used for classification, with a key difference in the loss function. We employed the mean squared error (MSE) as our loss function and used the identity function as the final activation function, aiming to predict continuous values rather than probabilities.

### Hyperparameters

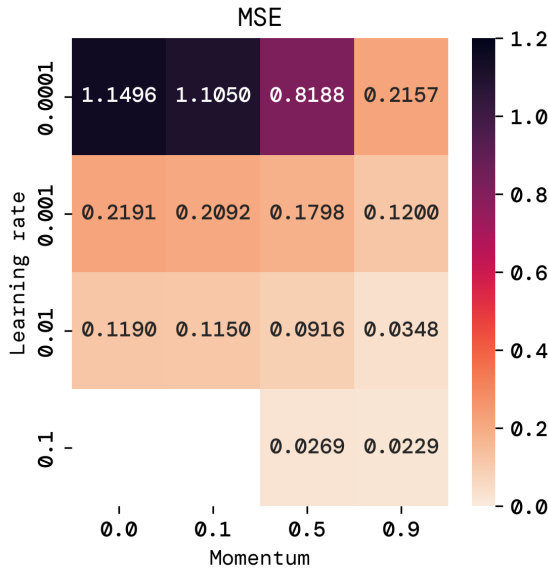


Figure 15: MSE versus learning rate and momentum

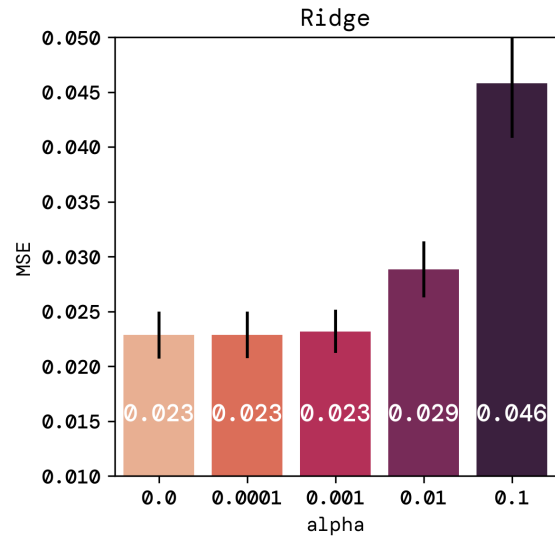


Figure 16: MSE versus regularization.

We observed that learning rate and momentum share a similar correlation to that in classification tasks. Regularization significantly impacted the model's performance, where excessive regularization hampered performance.

In figure 17, we can see that the optimizer we used had varying impact, with RMSprop underperforming compared to others. A smaller batch size generally led to better performance, see figure 18.

For the hidden layers, the ReLU activation function proved most effective. Our experiments found that a network with 5 layers, each containing 5 nodes, yielded the best results. It's crucial to use the identity function (or no activation function) in the final layer for regression tasks to predict continuous values accurately, as shown in figure 19 and figure 20.

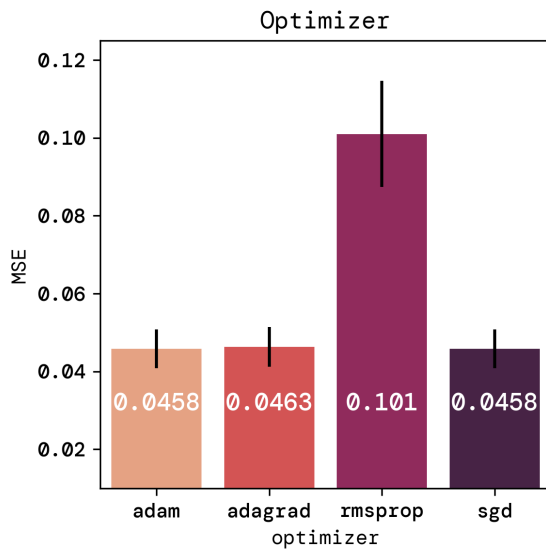


Figure 17: Model performance across different optimizers. here sg is standard gradient descent.

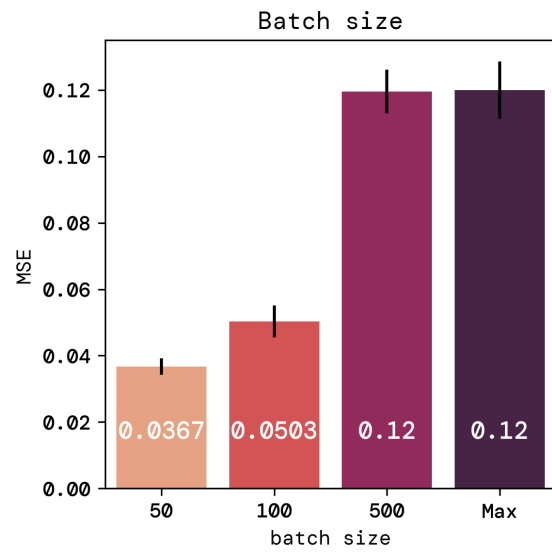


Figure 18: MSE based on batch size.

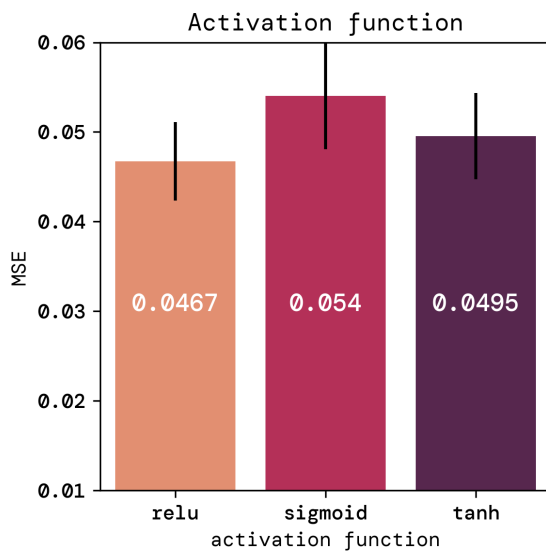


Figure 19: Model performance with different activation functions.

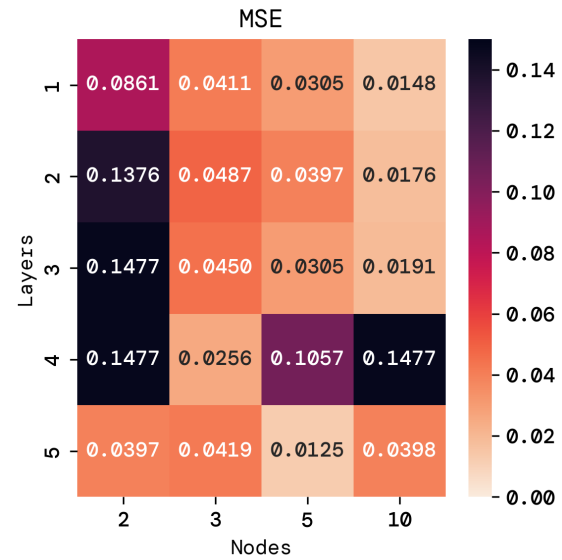


Figure 20: MSE in relation to the number of layers and nodes.

We also explored the impact of the degree of the polynomial on the model's performance. For the values tested, the model's performance improved with increasing polynomial degree.



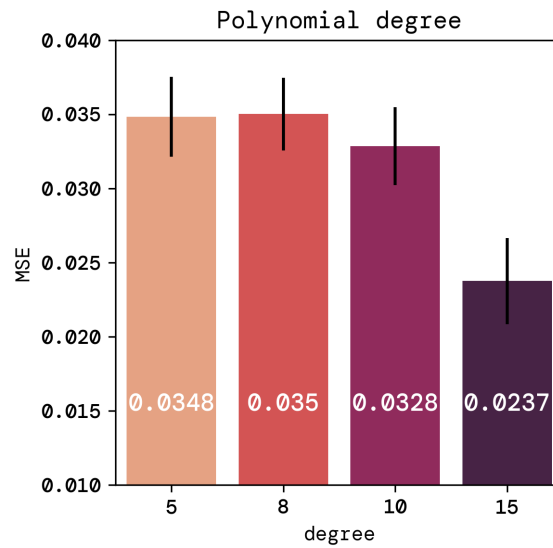


Figure 21: MSE in relation to the polynomial degree.

### Final Evaluation and Comparisons

Our final evaluation involved comparing the regression performance of neural networks with that of logistic regression on Perlin noise-generated data. The visualizations provide insights into each model's ability to capture the underlying patterns in the data.

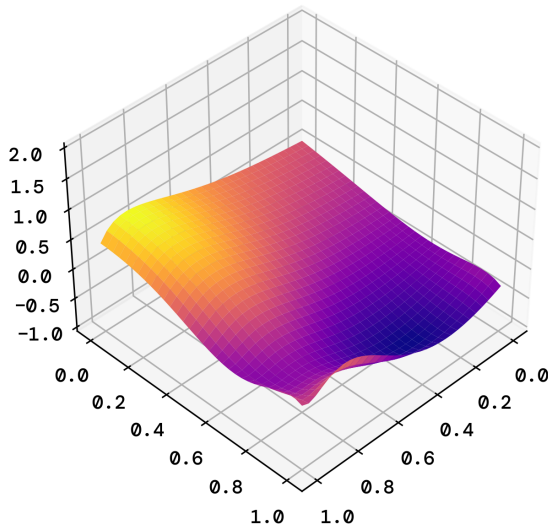


Figure 22: Logistic regression model predictions.

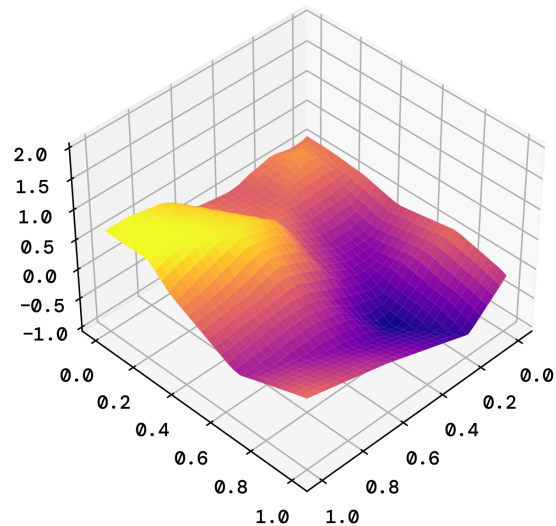


Figure 23: Neural network model predictions.

We managed to achieve a MSE of 0.0012 with our neural network model and 0.0028 with logistic regression. from figure 22 and figure 23 we can clearly see that the neural network is better at capturing the underlying pattern in the data. The logistic regression model is not able to capture the complexity of the data. It should be noted that if we allow the logistic regression model to have a higher degree of freedom, it will probably be able to compete with the neural network. However, this well not be a scalable solution, with a polynomial of degree 15 we have over 130 features.

# Appendix B

## Appendix B. Universal Approximation

A fundamental question in neural network theory is the extent of their learning capabilities. Specifically, can neural networks approximate any function? There are traditional methods for function approximation, such as Fourier series for periodic functions or Taylor series for local approximations. However, neural networks offer a more flexible approach. According to the Universal Approximation Theorem, a neural network with even a single hidden layer can approximate any continuous function, provided it has enough neurons [4]. This capacity for universal approximation underlines the power and versatility of neural networks in modeling complex relationships.

### XOR VS Perceptron

A perceptron is a simple model of a neuron. It takes in a set of inputs, multiplies them by a set of weights and sums them up to produce an output. The output is then passed through the heaviside<sup>3</sup> step function to produce a binary output. The perceptron can be used to solve simple classification problems. However, it is not able to solve the XOR problem. XOR is not linearly separable, meaning that it is not possible to draw a straight line that separates the two classes. This is a problem for the perceptron, as it can only draw straight lines. One way to solve this problem is to add some polynomial features to the data. This will allow us to draw a curved line that separates the two classes. NNs can also learn xor, without the need for a polynomial feature. This is important because it is not always obvious what feature engineering is required to solve a problem. Neural networks can learn the relevant features from the data.

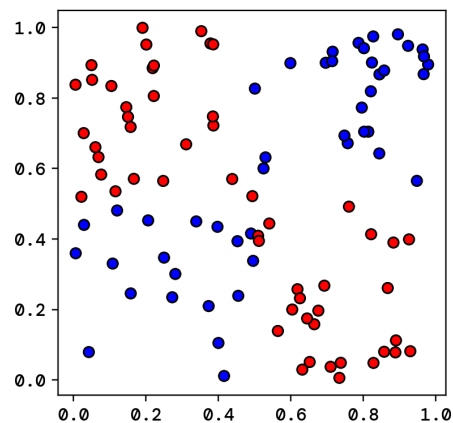


Figure 24: XOR-like dataset. The two classes are not linearly separable.

---

3. We use sigmoid instead of the heaviside, but the outcome of the classification is the same.

Consider figure 25, figure 26 and figure 27. The intersection of the plane and the height  $z = 0.5$  is the decision boundary. XOR gets separated by this boundry.

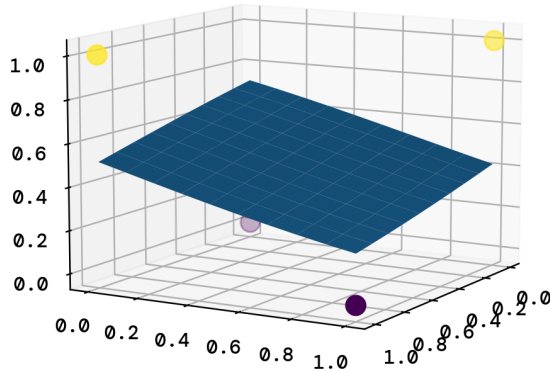


Figure 25: The output of a logistic regression model with no polynomial features added to the data. We get a classification accuracy of 0.5.

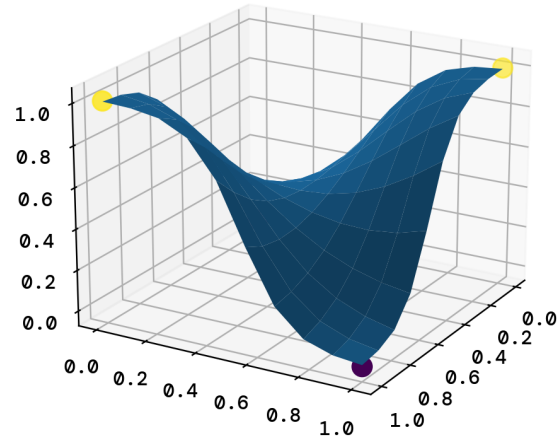


Figure 26: The output of a logistic regression model with polynomial features added to the data. We get a classification accuracy of 1.0.

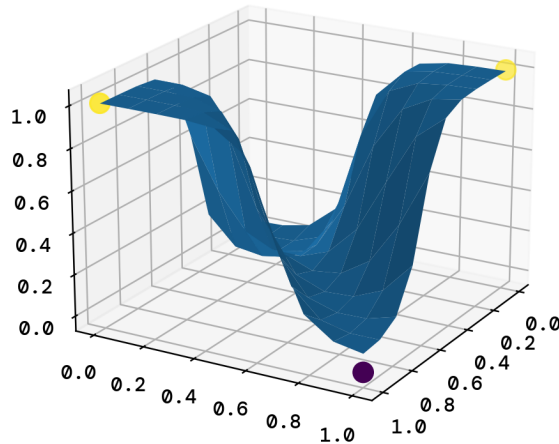


Figure 27: The output of a neural network with no polynomial features added to the data. We get a classification accuracy of 1.0.

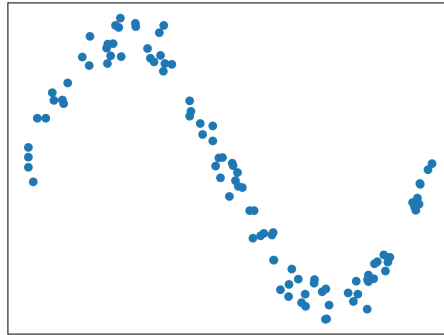


Figure 28: The sin function to be approximated.

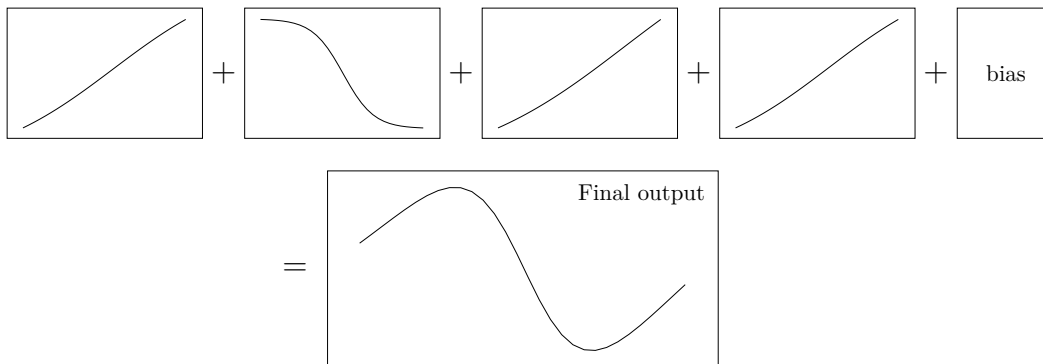


Figure 29: Combining outputs of individual neurons to approximate a sin function.

## Combining Neurons

We want to approximate a function like  $\sin(x)$  with a neural network. How might that look like? We can use a single neuron to approximate a line  $aw + b$ , putting it through a sigmoid function we get some non-linearity. Every node in the first hidden layer (and consequent layers) is then a line put through a sigmoid function. The final output of a single hidden layer network is just a linear combination of these curves. We can add more nodes to get more curves.

In this context, the choice of activation function is crucial. While ReLU might struggle with functions like  $\sin(x)$  due to its linear nature, sigmoid or similar non-linear functions can successfully approximate such patterns. This exemplifies how neural networks use simple components to build complex models, capturing a wide range of functional relationships. A great post on this topic can be found here[6].

# Bibliography

- [1] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. Book in preparation for MIT Press.
- [2] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. Springer Series in Statistics. Springer New York Inc., New York, NY, USA, 2001.
- [3] Morten Hjorth-Jensen. Fys-stk4155/3155 applied data analysis and machine learning. [https://compphysics.github.io/MachineLearning/doc/LectureNotes/\\_build/html/intro.html](https://compphysics.github.io/MachineLearning/doc/LectureNotes/_build/html/intro.html), 2021.
- [4] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989.
- [5] Izaak Neutelings. Tikz code for neural networks. [https://tikz.net/neural\\_networks/](https://tikz.net/neural_networks/), 2021. modified to fit the needs of this project.
- [6] Michael A. Nielsen. Neural networks and deep learning, 2018.
- [7] OpenAI. Chat-gpt. <https://openai.com/chatgpt>, 2023. Some of the code and formulations are developed with the help from Chat-GPT.
- [8] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.
- [9] Brage Wiseth, Felix Cameren Heyerdahl, and Eirik Bjørnson Jahr. MachineLearning-Projects. <https://github.com/bragewiseth/MachineLearningProjects>, November 2023.
- [10] W. Wolberg, O. Mangasarian, N. Street, and W. Street. Breast Cancer Wisconsin (Diagnostic). UCI Machine Learning Repository, 1995. DOI: <https://doi.org/10.24432/C5DW2B>.