



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

November 16, 2023

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

# Contents

Introduction . . . . .	3
1 Gradient Descent . . . . .	4
1.1 Backpropagation and Chain Rule . . . . .	4
2 Neural Networks . . . . .	5
3 Data . . . . .	6
4 Results and Analysis . . . . .	6
4.1 Hyperparameters . . . . .	6
4.2 Final Evaluation and Comparisons . . . . .	10
5 Conclusion . . . . .	10
<b>Appendix</b>	<b>11</b>
A Regression with Neural Networks . . . . .	11
B Universal Approximation . . . . .	14
Bibliography . . . . .	19

## Abstract

In this paper, we investigate the viability of using neural networks to solve classification problems. Specifically, we compare the performance of neural networks with that of traditional logistic regression models. There are many real world applications and problems that require us to correctly differentiate between some classes. For example, in the field of cancer research, it is often necessary to classify a tumor as either benign or malignant. This can be done by extracting a set of features from the tumor and differentiating between the two classes based on how different classes possess slight variations across the features. However, some features may be more or less equal for both classes, and the amount of features may be large. These two factors may make it difficult for a human to classify the tumor. By using a neural network, we achieved a classification accuracy of 98% on the Wisconsin breast cancer dataset. With logistic regression, we achieved an accuracy of 98%. Comparing that with SKLearn's model we achieved an accuracy of 96%. In addition to classification, we also investigated the use of neural networks for regression problems see Appendix A. We found that neural networks are able to approximate the Franke function with a mean squared error of 0.0001. This is comparable to the results we achieved with linear regression. However, we found that neural networks are much more sensitive to the choice of learning rate and number of epochs. We also found that neural networks are much more computationally expensive than linear regression.

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

## Introduction

In project 1[12], we found that we can fit lines or even polynomials that approximate the distribution of our data by solving an nice analytical expression for the optimal parameters  $\beta$ . We can in principle approximate any function with a polynomial, if we give ourselves infinite degrees of freedom. This is great but there are several limitations to this approach. First of all, we can not give ourselves infinite degrees of freedom, believe it or not. Secondly, what if we don't want to find a polynomial but rather classify our data into some classes?

To tackle classification we can use *logistic regression*, that is, first regression and then clamp the output to a binary value (for the binary case). We can do this with an activation function like the sigmoid function<sup>1</sup> or the heaviside function. However the first problem still remains, we want to approximate any function, but don't want to use an infinite Taylor series. We need a different approach. Instead of finding a single high degree polynomial, we can try to glue together a bunch of smaller line segments. For this we can use a *neural network*. As it turns out, the framework for neural networks is very similar to the framework for logistic regression. Neural networks can be interpreted as several logistic regression models glued together, which is exactly what we wanted! Another huge benefit of this is that we can use the same code for both logistic and linear regression as well as neural networks. This sounds great, but by introducing activation functions we lose the nice analytical expression, so we can't use the same matrix inversion approach as before.

So how do we learn?

---

1. The sigmoid function does not output a binary value, but a value between 0 and 1. We can then set a threshold, for example 0.5, and say that if the output is above the threshold, we classify it as 1, and if it is below, we classify it as 0. We can interpret the output as the probability of the data point being 1.

### Gradient Decent:

In this section we will discuss the gradient decent algorithm and how it can be used to optimize the parameters of a model.

### Data:

In this section we will discuss the data we will be using in this project.

### Results:

In this section we will discuss the results we achieved with our models.

### Conclusion:

In this section we will discuss the results we achieved with our models.

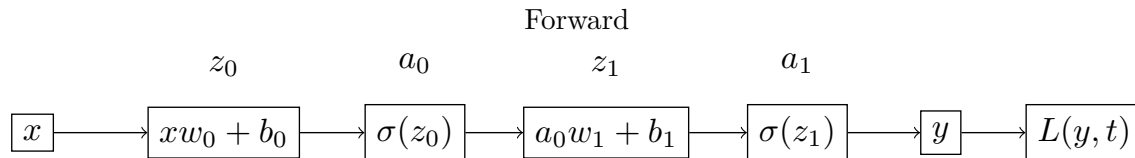
## 1. Gradient Descent

Gradient descent is an iterative optimization algorithm for finding the minimum of a function. The idea is to take steps in the direction of the negative gradient of the function. It is very similar to the Newton-Raphson method for finding roots of a polynomial, but instead of using the second derivative, we use the first derivative.

optimizers  
stochastic

### 1.1 Backpropagation and Chain Rule

Calculating derivatives is the bread and butter of machine learning. For the simplest models, like linear regression, the derivatives are relatively easy and straight forward to calculate. We simply take the derivative of the cost function with respect to our parameters. However, for more complex models, like neural networks, the derivatives are not so easy to calculate. When the loss function is a composition of several functions, we need to invoke the chain rule.



If we want to calculate the derivative of the loss function with respect to  $w_0$ , we must go through all the functions in the chain and calculate the derivative of each function with respect to the next function.

Backwards

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

Figure 1: chain rule

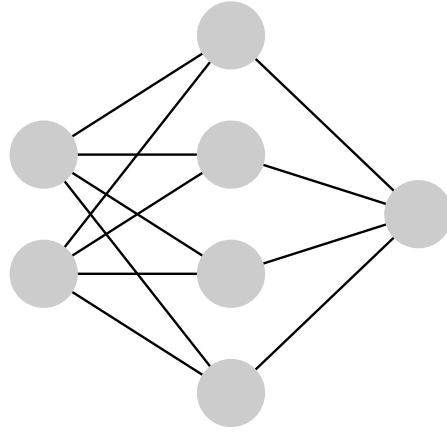


Figure 2: A model of a neural network with one hidden layer consisting of four nodes [7]

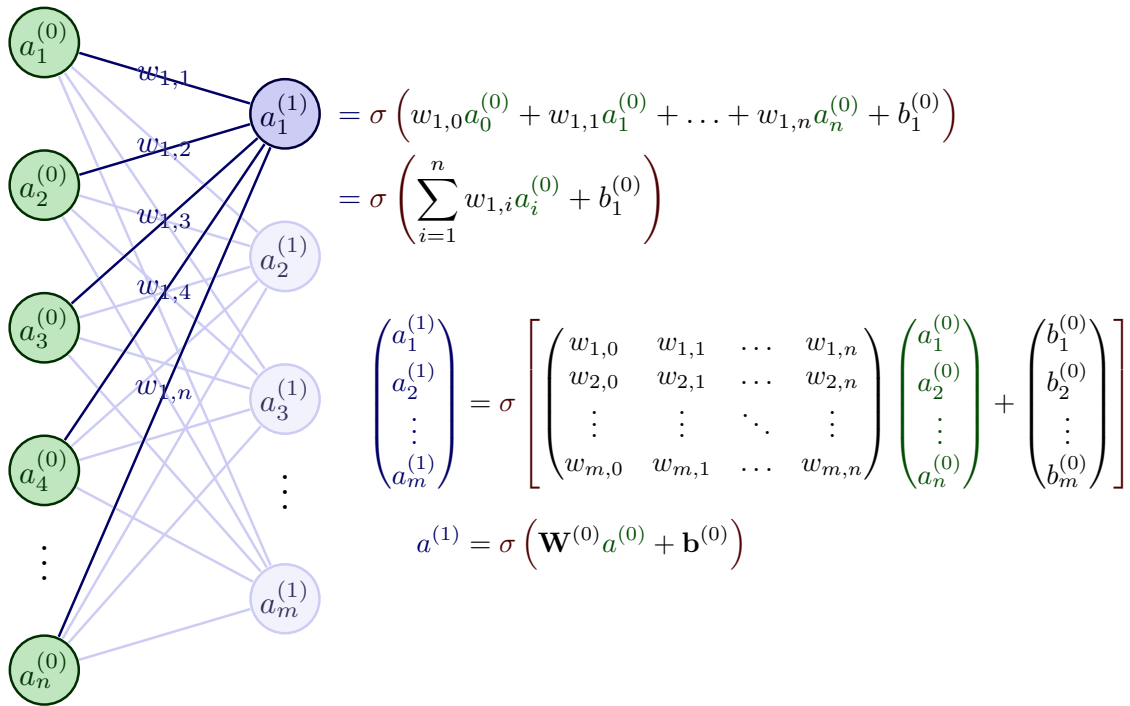


Figure 3

[7]

## 2. Neural Networks

With backpropagation we have a way to calculate the derivatives of the loss function with respect to the parameters. This allows us to train our neural network. with one layer the

network becomes a linear regression model.  $y = x_0w_0 + x_1w_1 + x_2w_2 + \dots + x_nw_n + b_0$  with a activation function at the end we can make it into a logistig regression model. This makes it very convenient to use the same code for both linear and logistic regression as well as for larger neural networks with hidden layers, by simply swapping out the different parts.

activation functions

### 3. Data

for classification we will be using the Wisconsin breast cancer dataset [cite here]. This dataset contains 569 data points with 30 features each. The features are computed from a digitized image of a fine needle aspirate (FNA) of a breast mass. They describe characteristics of the cell nuclei present in the image. The features are computed for each cell nucleus: radius, texture, perimeter, area, smoothness, compactness, concavity, concave points, symmetry, fractal dimension. The mean, standard error and "worst" or largest (mean of the three largest values) of these features were computed for each image, resulting in 30 features. For example, field 3 is Mean Radius, field 13 is Radius SE, field 23 is Worst Radius. All feature values are recoded with four significant digits. Missing attribute values: none. Class distribution: 357 benign, 212 malignant

```
malignant = cancer.data[cancer.target==0] benign = cancer.data[cancer.target==1]
```

the goal is to classify the tumors as either benign or malignant based on the features. a positive result means that the tumor is malignant, and a negative result means that the tumor is benign. in other words, we want to find the cases of cancer and classify them as positive

### 4. Results and Analysis

For classification we use the cross entropy loss function. We tune the hyperparameters in a grid searches with a maximum of two dimensions. As we find the optimal hyperparameters, we use the same optimal hyperparameters for the next grid search.

some of the hyperparameters are more dependent on each other than others. For example, the learning rate and the number of epochs the batch size and the learning rate. We can therefore not tune these hyperparameters independently. We must first find the optimal

due to our limited time and computational resources we have not been able to do a full grid search for all hyperparameters.

#### 4.1 Hyperparameters

epochs found to be 200

when training for batch size we had to decrease the optimal learning rate due to unstable results for smaller batches

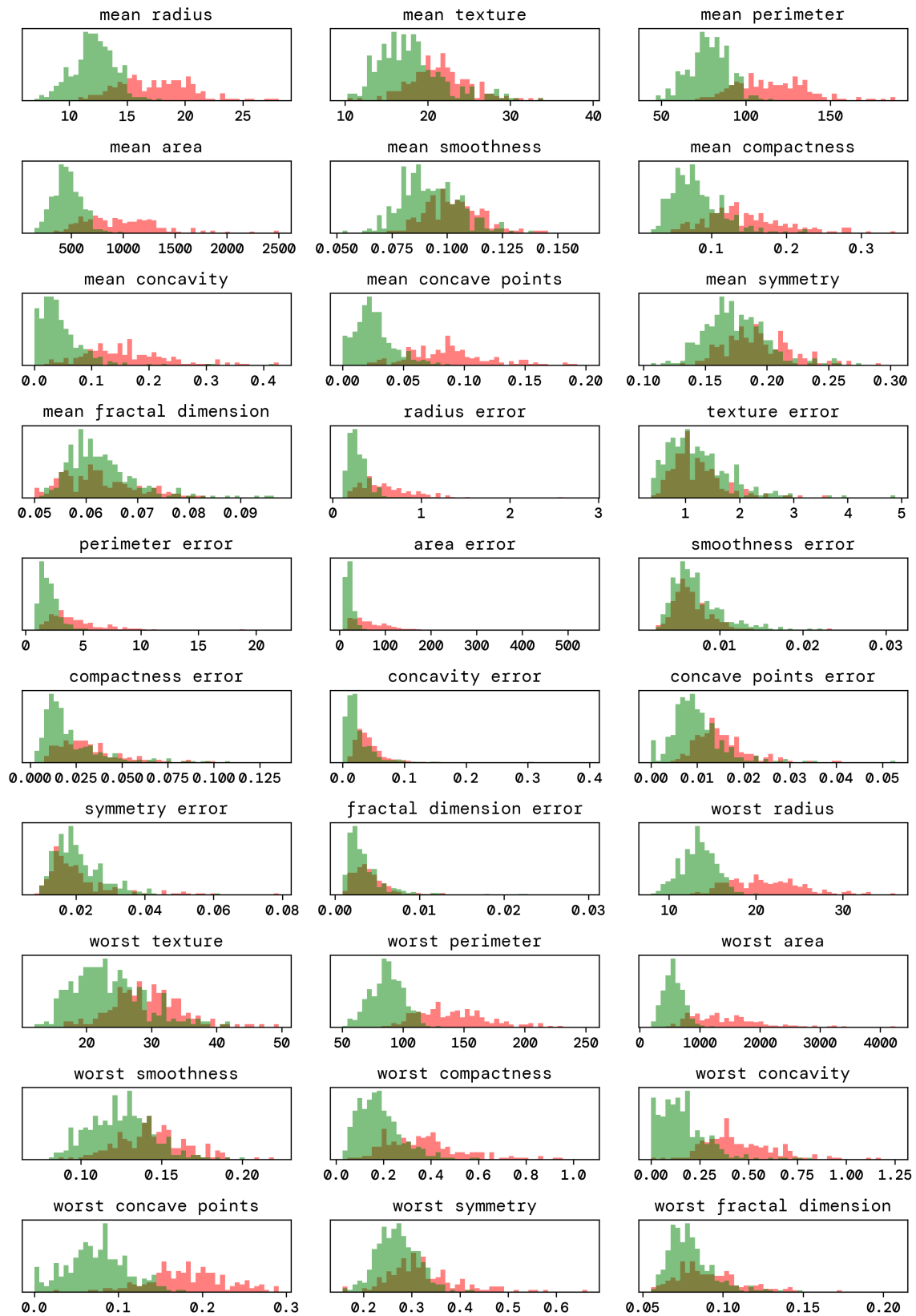


Figure 4: Feature histogram

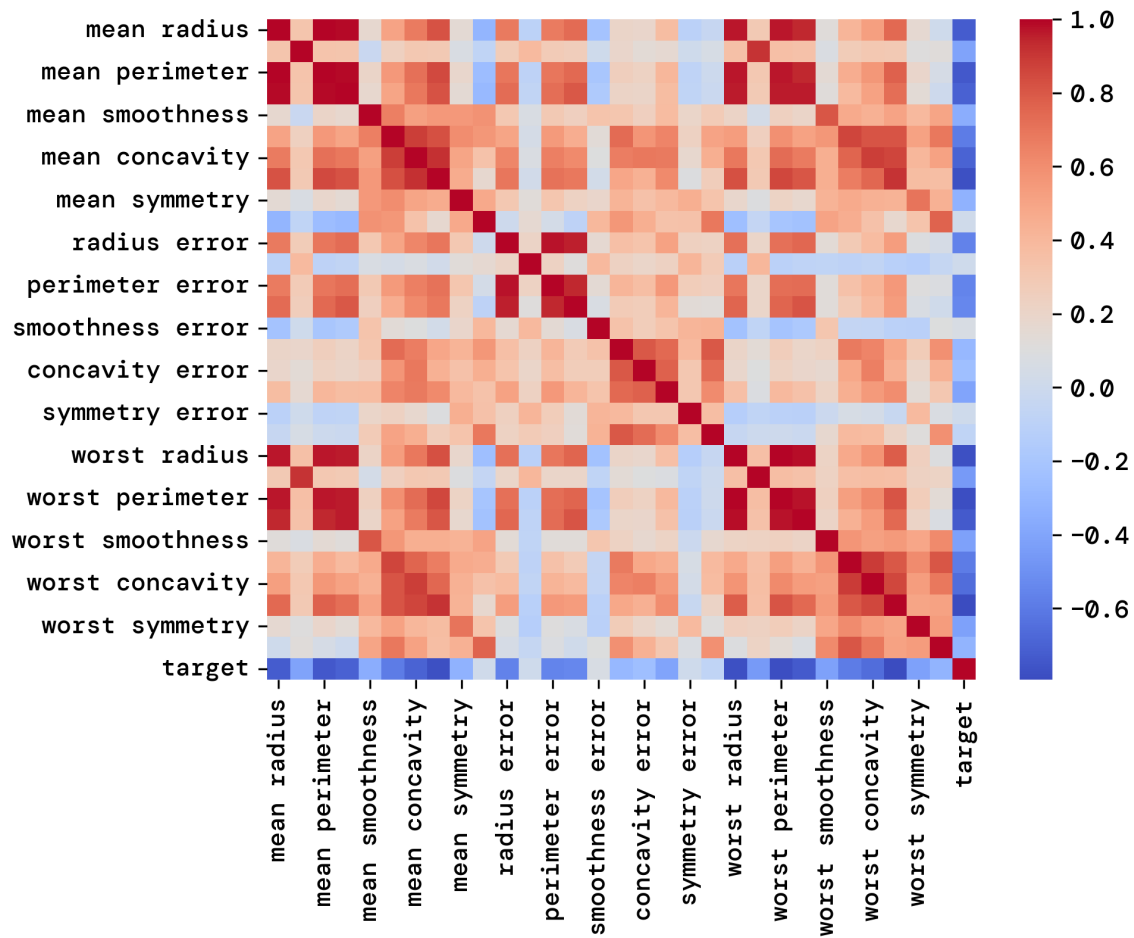


Figure 5: Feature correlation. Gives us an idea of the redundancy of the features. Only every other features is annotated but you can infer the missing labels from Figure 4



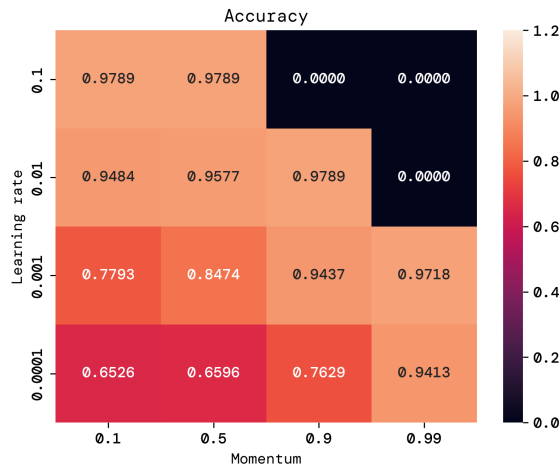


Figure 6: In this case having momentum seems to be beneficial. We maxed out our testing range and found that 0.9 was the best value for momentum. momentum allows a higher learning rate

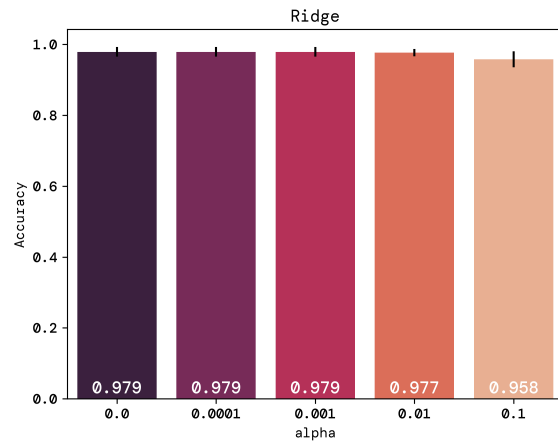


Figure 7: Introduction regularization does not seem to yield any benefits, in fact having too much regularization shunts performance

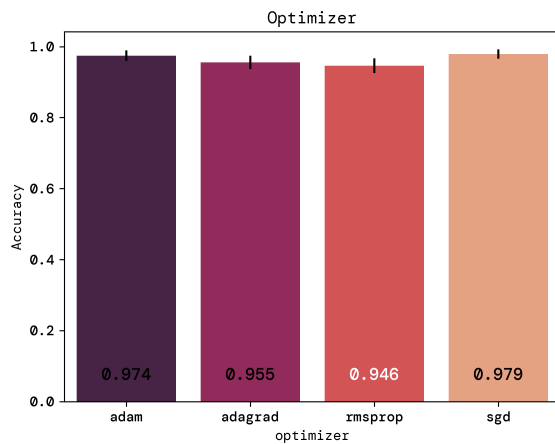


Figure 8: In this case having momentum seems to be beneficial. We maxed out our testing range and found that 0.9 was the best value for momentum. momentum allows a higher learning rate

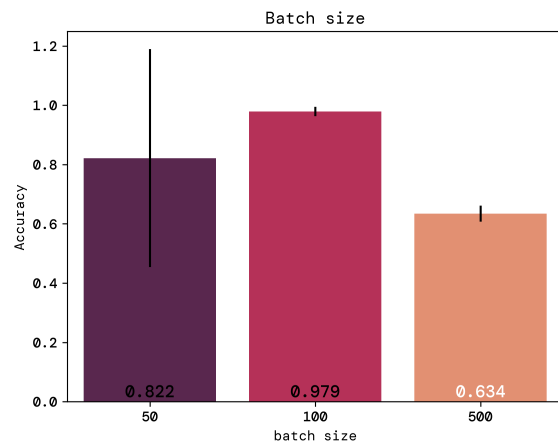


Figure 9: Introduction regularization does not seem to yield any benefits, in fact having too much regularization shunts performance

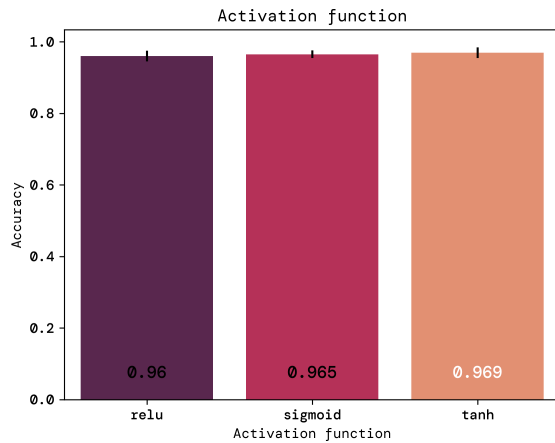


Figure 10: In this case havving momentum seems to be beneficial. We maxes out our testing range and found that 0.9 was the best value for momentum. momentum allows a higher learning rate

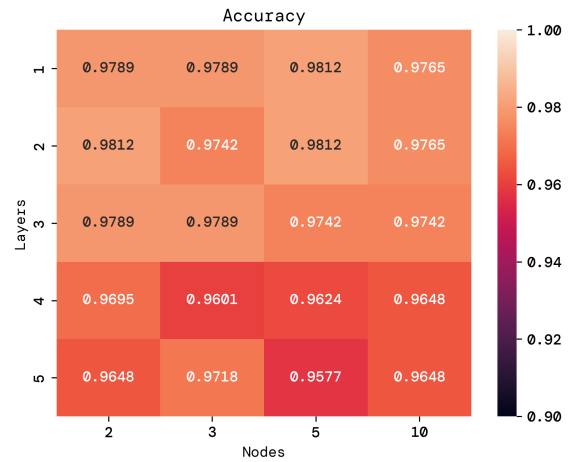


Figure 11: Introduction regularization does not seem to yield any benefits, in fact having too much regularization shunts performance

## 4.2 Final Evaluation and Comparisons

## 5. Conclusion

Lorem ipsum dolor sit amet, officia excepteur ex fugiat reprehenderit enim labore culpa sint ad nisi Lorem pariatur mollit ex esse exercitation amet. Nisi anim cupidatat excepteur officia. Reprehenderit nostrud nostrud ipsum Lorem est aliquip amet voluptate voluptate dolor minim nulla est proident. Nostrud officia pariatur ut officia. Sit irure elit esse ea nulla sunt ex occaecat reprehenderit commodo officia dolor Lorem duis laboris cupidatat officia voluptate. Culpa proident adipisicing id nulla nisi laboris ex in Lorem sunt duis officia eiusmod. Aliqua reprehenderit commodo ex non excepteur duis sunt velit enim. Voluptate laboris sint cupidatat ullamco ut ea consectetur et est culpa et culpa duis.

# Appendix A

## Appendix A. Regression with Neural Networks

### Data

For regression we will be using perlin noise to generate our dataset

Solving a regression problem with linear regression and neural networks using gradient optimization.

### Results and Analysis

### Hyperparameters

### Final Evaluation and Comparisons

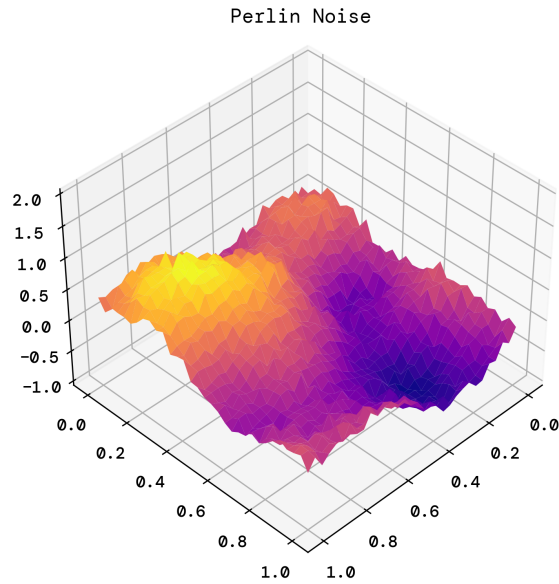


Figure 12

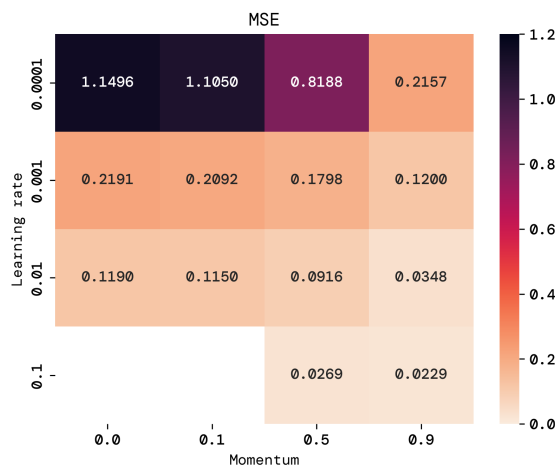


Figure 13: In this case having momentum seems to be beneficial. We maxes out our testing range and found that 0.9 was the best value for momentum. momentum allows a higher learning rate

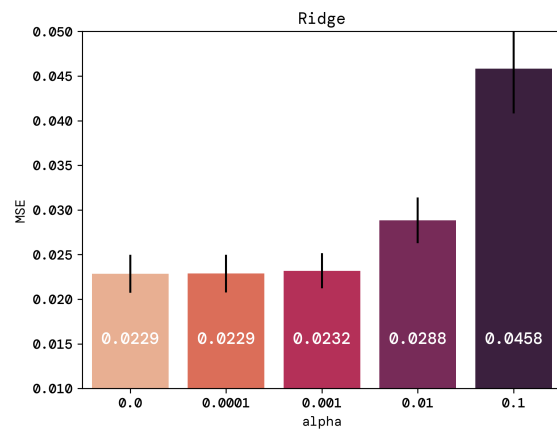


Figure 14: Introduction regularization does not seem to yield any benefits, in fact having too much regularization shunts performance

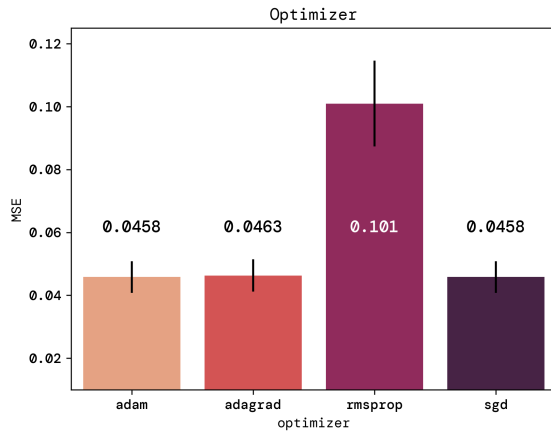


Figure 15: In this case having momentum seems to be beneficial. We maxes out our testing range and found that 0.9 was the best value for momentum. momentum allows a higher learning rate

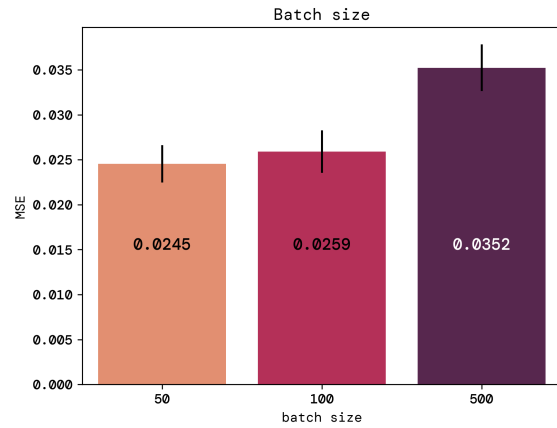


Figure 16: Introduction regularization does not seem to yield any benefits, in fact having too much regularization shunts performance

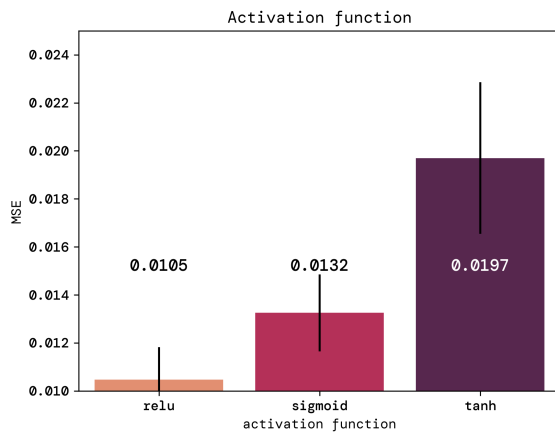


Figure 17: In this case having momentum seems to be beneficial. We maxes out our testing range and found that 0.9 was the best value for momentum. momentum allows a higher learning rate

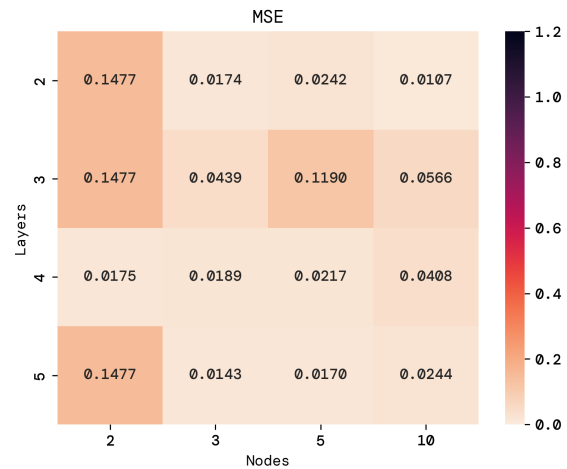


Figure 18: Introduction regularization does not seem to yield any benefits, in fact having too much regularization shunts performance

# Appendix B

## Appendix B. Universal Approximation

Where is the limit of what we can learn with a neural network? Can we approximate any function? There are several ways to approximate a function. For periodic functions it may be wise to use a Fourier series, another option is to use a Taylor series. If these methods don't float your boat, we can in fact use a neural network instead. A neural network with only one hidden layer can approximate any continuous function given enough neurons in the hidden layer. The more neurons, the higher the resolution of the Approximation [6].

### 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>2</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. However, this is not a scalable solution.

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. Manually adding polynomial features is not a scalable solution for high-dimensional data, whereas neural networks can learn arbitrarily complex functions. Adding polynomial features is also prone to bias problems.

### Combining Neurons

We want to approximate this sin function 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

---

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

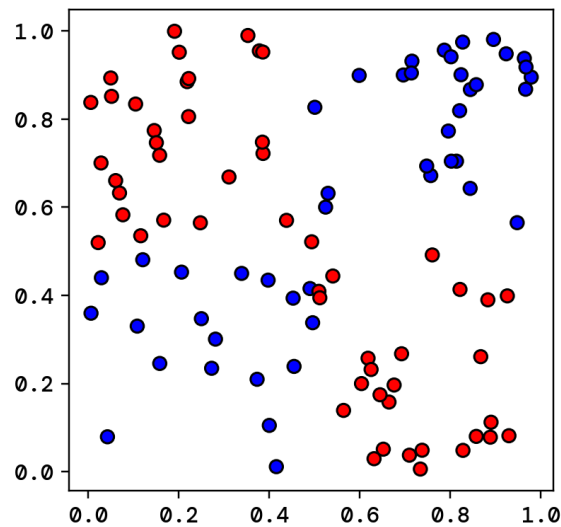


Figure 19: XOR-variant, it is impossible to separate the two classes with a straight line

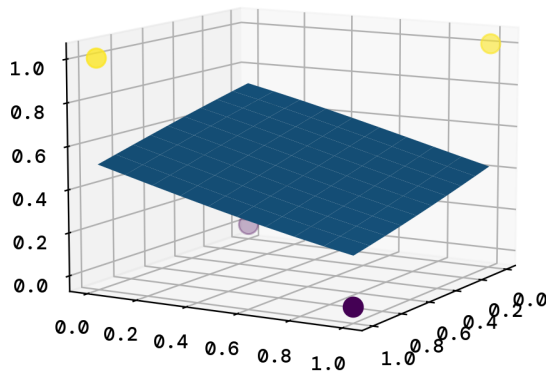


Figure 20

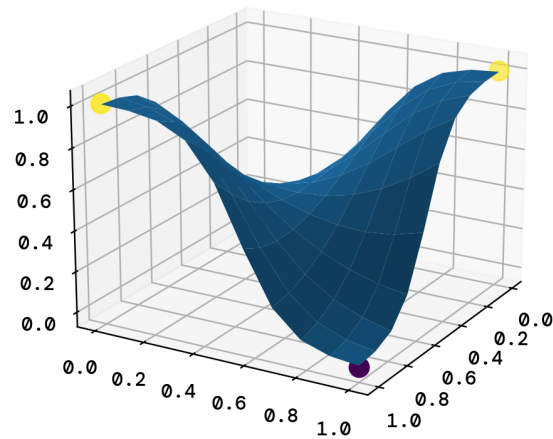


Figure 21

layer network is just a linear combination of these curves. We can add more nodes to get more curves, but we are still

We can choose whatever activation function we want as long as it is non-linear. Relu struggles to learn this sin function, but sigmoid works fine.

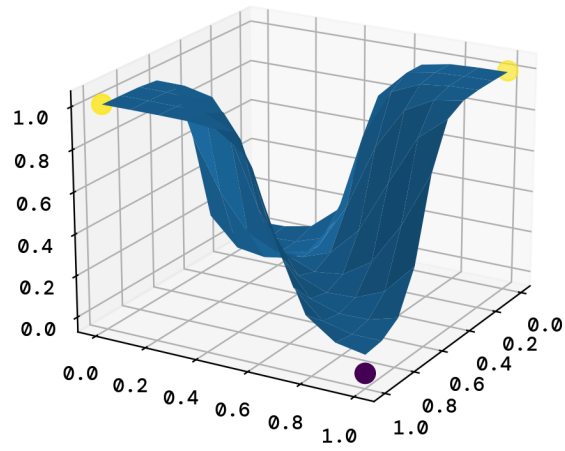


Figure 22

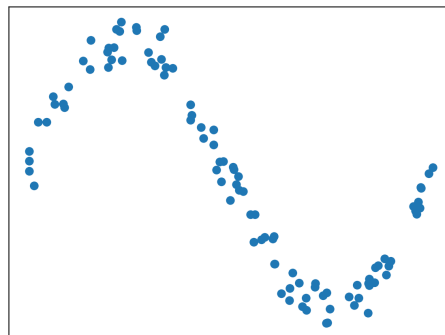


Figure 23: A sin function that we want to approximate



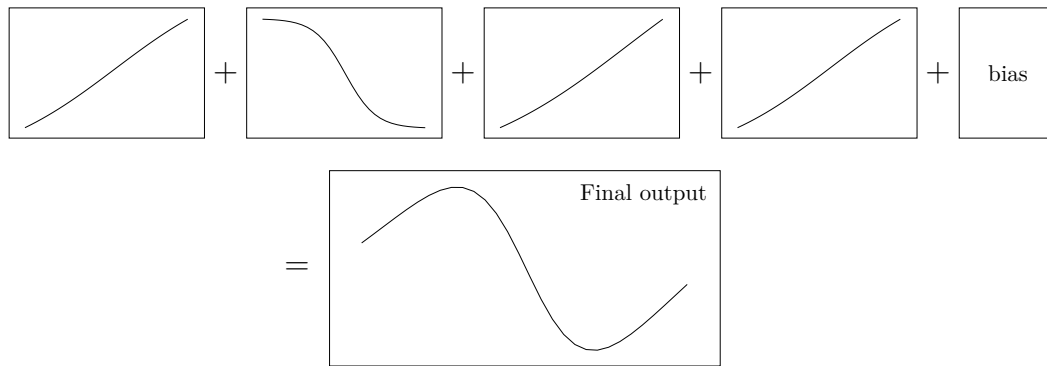


Figure 24: Four individual logistic regression outputs scaled and added together to produce a sin approximation

# Bibliography

- [1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.
- [2] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer, 1 edition, 2007.
- [3] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. Book in preparation for MIT Press.
- [4] 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.
- [5] 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.
- [6] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989.
- [7] 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.
- [8] Michael A. Nielsen. Neural networks and deep learning, 2018.
- [9] OpenAI. Chat-gpt. <https://openai.com/chatgpt>, 2023. Some of the code and formulations are developed with the help from Chat-GPT.
- [10] 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.

<https://github.com/bragewiseth/MachineLearningProjects>

- [11] Wessel N. van Wieringen. Lecture notes on ridge regression, 2023.
- [12] Brage Wiseth, Felix Cameren Heyerdahl, and Eirik Bjørnson Jahr. MachineLearning-Projects. <https://github.com/bragewiseth/MachineLearningProjects>, November 2023.