University of Oslo

FYS-STK3155

Project 2

Brage Wiseth bragewi@ifi.uio.no
Felix Cameren Heyerdahl felixch@ifi.uio.no
Eirik Bjørnson Jahr eiribja@ifi.uio.no

November 12, 2023

https://github.com/bragewiseth/MachineLearningProjects

Contents

Abstract

In this paper, we investigate the viability of using neural networks to solve both regression and classification problems. Specifically, we compare the performance of neural networks with that of traditional linear and logistic regression models. There are many real world applications and problems that require function approximation and classification. For example, in the field of cancer research, it is often necessary to classify a tumor as either benign or malignant. Often, this is done by extracting a set of features from the tumor and differentiating between the two classes based on these features. However, there may be a large degree of overlap between the two and the amount of features may be large. These two factors may make it difficult foe a human to classify the tumor. By using a neural network, we achieved a classification accuracy of 96% on the Wisconsin breast cancer dataset. With logistic regression, we achieved an accuracy of 94%. In addition to classification, we also investigated the use of neural networks for regression problems. 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

1. Introduction

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.

In project 1[12], we found that we can fit lines or even polynimals that approximate the distribution of our data by solving an nice analytical expression for the optimal parameters β. 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 polynimal 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 bianry case). We can do this with an activation function like the sigmoid function [1] 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 polynimial, 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

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.

regression. Neural networks can be interpeted 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?

2. 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 verry similar to the Newton-Raphson method for finding roots of a polynimal, but instead of using the second derivative, we use the first derivative.

2.1 Backpropagation and Chain Rule

Calculating derivatives is the bread and butter of machine learning. For the simpelest 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.

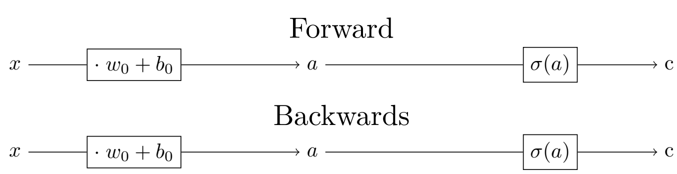


Figure 1: chain rule

3. Neural Networks

with one layer the network becomes a linear regression model. $y = x_0w_0 + x_1w_1 + x_2w_2 + ... + x_nw_n + b_0$ 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.

4. Data

Wisconsin breast cancer dataset & Boring regression data

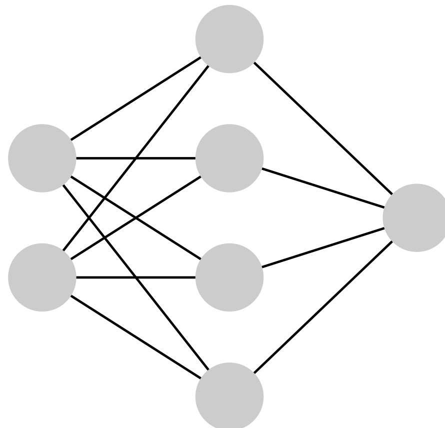


Figure 2:

[7]

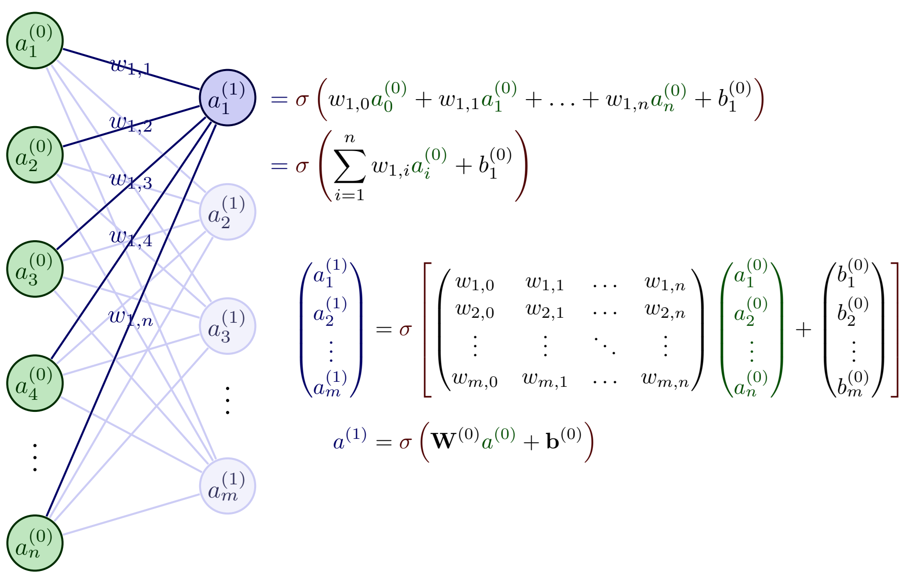


Figure 3:

[7]

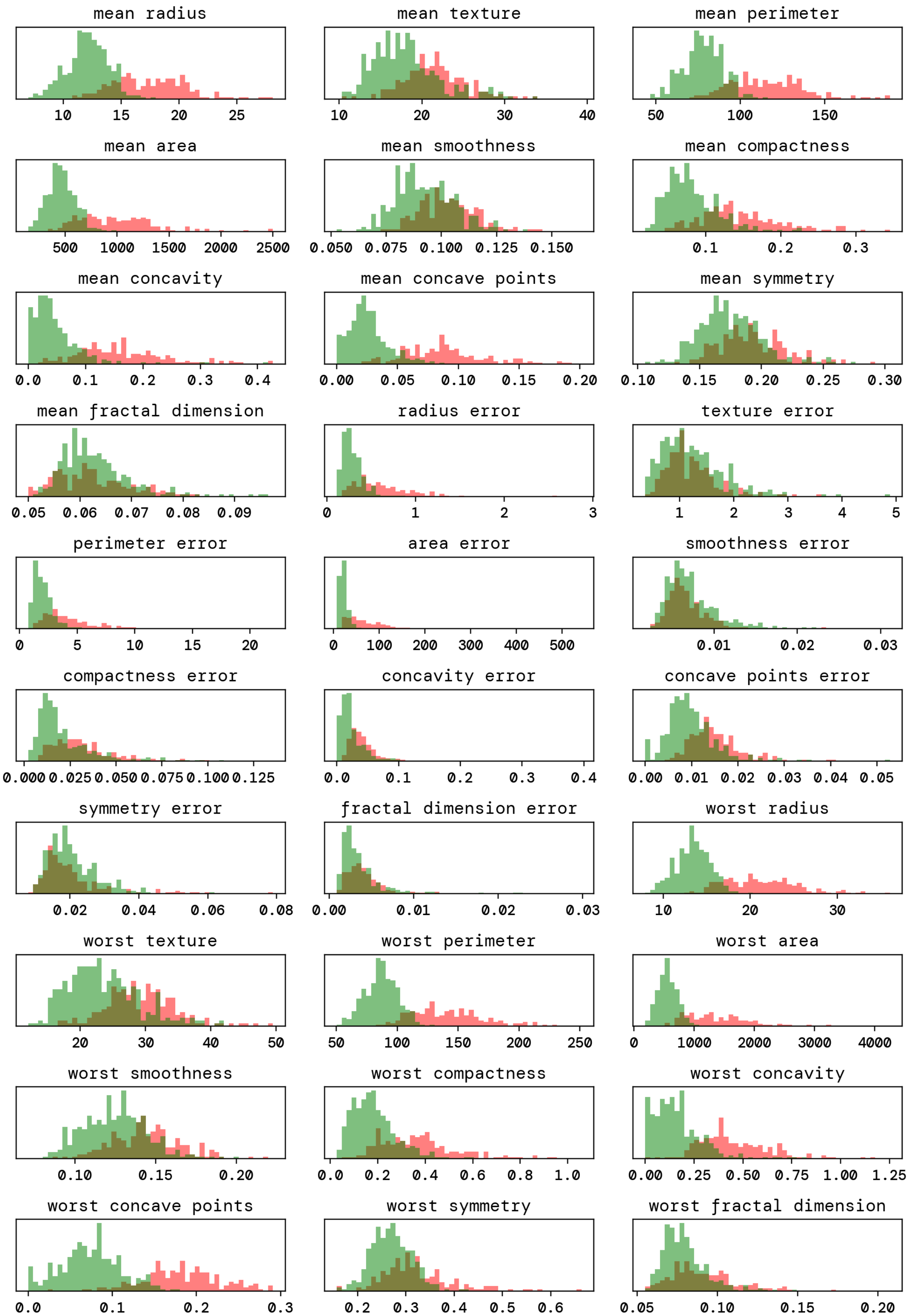


Figure 4: Feature histogram

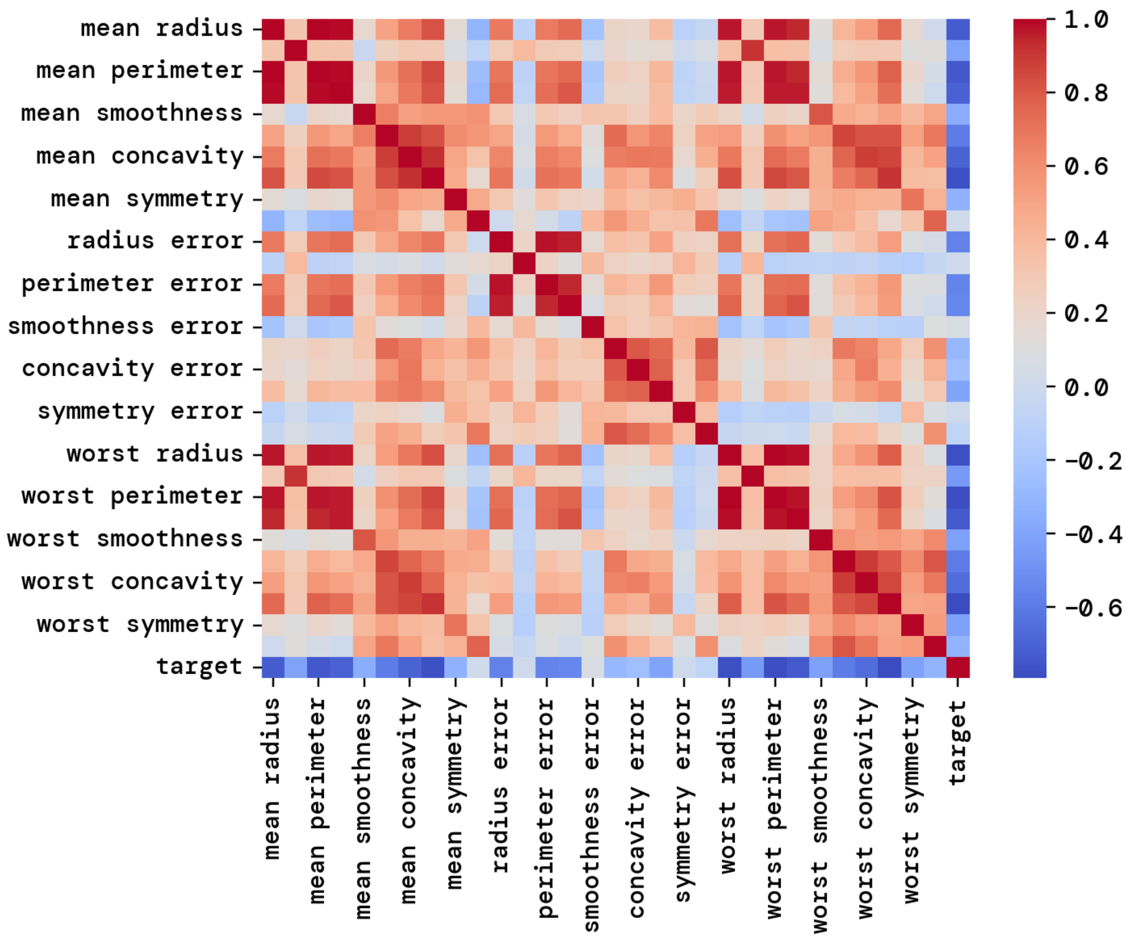


Figure 5: Feature correlation

https://github.com/bragewiseth/MachineLearningProjects

5. Results and Discussion

5.1 Classification

5.2 Regression

6. Conclusion

Appendix

Appendix A. 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 mehtods 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

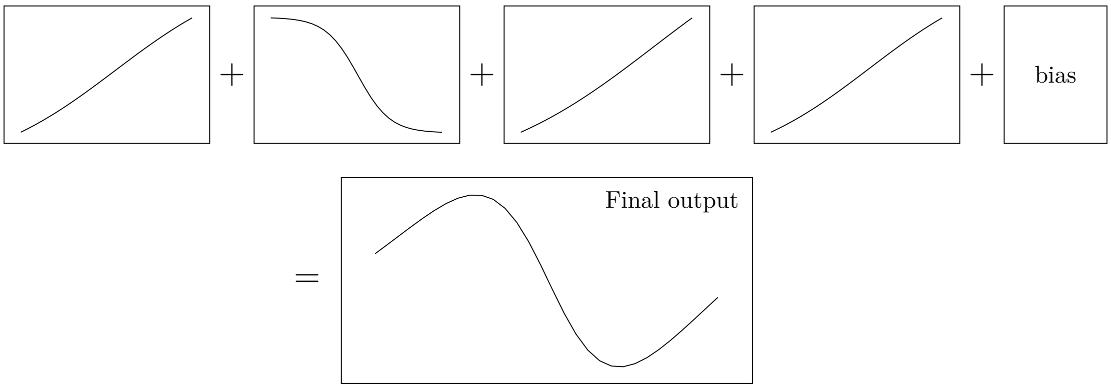
A perceptron is a simple model of a neuron. It takes in a set of inputs, multiplies them by a set of weights and sums them up to produce an output. The output is then passed through the heaviside [2] step function to produce a binary output. The perceptron can be used to solve simple classification problems. 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 featurer is also prone to bias problems.

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

Combining Neurons



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

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`, 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/`, 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.