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 11, 2023

https://github.com/bragewiseth/MachineLearningProjects

Contents

Abstract

In this paper, we delve into the realm of machine learning model optimization and evaluation. Our study encompasses various regression techniques, including Ordinary Least Squares (OLS), Ridge, and Lasso regression, to analyze their effectiveness in handling simple and more complex datasets. Additionally, we employ bootstrap resampling and cross-validation methodologies to rigorously assess model performance and enhance generalization. A significant portion of our investigation is dedicated to understanding the delicate balance between bias and variance. We explore how regularization methods like Ridge and Lasso impact bias-variance trade-offs, offering insights into the stability and predictive power of these models. Furthermore, we provide empirical evidence of the benefits of cross-validation and bootstrap techniques in mitigating overfitting and improving model robustness. We found that { ..results.. }. Additionally we verify and compare our findings with well established theory and libraris such as SKLearn.

Keywords: Linear Regression, Scaling, Bias & Variance

1. Introduction

An overview

Gradient Decent:
Data:
Results:
Conclusion:

In project 1[6], 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 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?

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.

https://github.com/bragewiseth/MachineLearningProjects

code for generating all figures and data can be found at /MachineLearningProjects/project1/src

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.

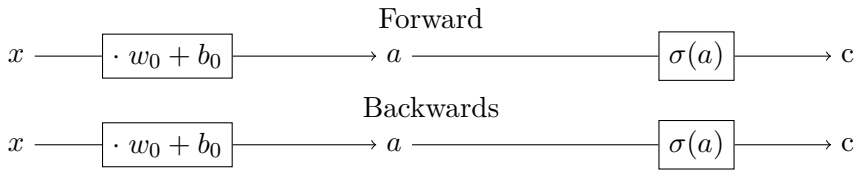
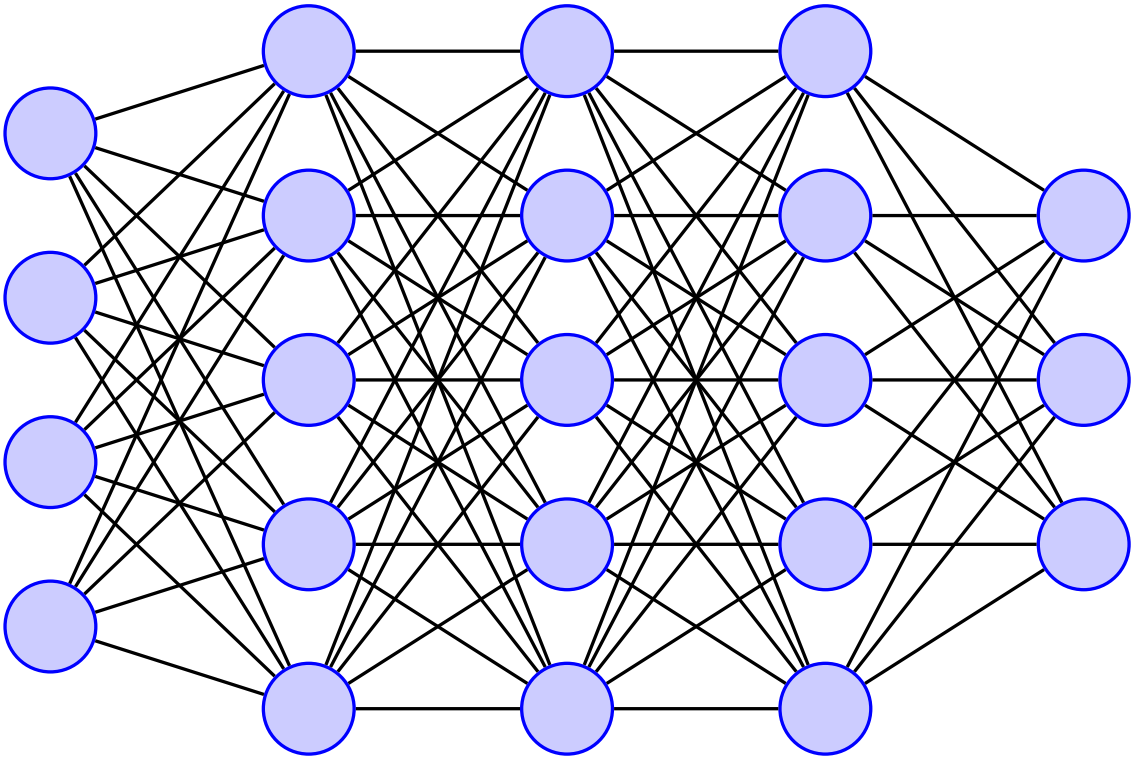


Figure 1: chain rule

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

5. Results and Discussion

6. Conclusion

Appendix

Appendix A. Universal Approximation Theorem

There is a famous theorem called the universal approximation theorem, which states that a neural network with one hidden layer can approximate any function. This is great, but it does not say anything about how many neurons we need in the hidden layer. It turns out that we need an infinite number of neurons to approximate any function. This is not very practical, so we need to find a way to approximate any function with a finite number of neurons.

Geofry Hinton showed that a simple perceptron is incapable of learning the XOR function. This is because the perceptron is a linear model, and the XOR function is not linearly separable. However, as we hinted at earlier, we can approximate any function with a polynomial or a neural network.

Lines glued together β

learn xor with perceptron

Bibliography

[1] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer, 1 edition, 2007.

[2] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. Book in preparation for MIT Press.

[3] 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.

[4] Michael A. Nielsen. Neural networks and deep learning, 2018.

[5] Wessel N. van Wieringen. Lecture notes on ridge regression, 2023.

[6] Brage Wiseth, Felix Cameren Heyerdahl, and Eirik Bjørnson Jahr. MachineLearning-Projects. `https://github.com/bragewiseth/MachineLearningProjects`, November 2023.