FYS-STK4155 - APPLIED DATA ANALYSIS AND MACHINE LEARNING

PROJECT 2

November 2021

Håkon Muggerud
Diederik van Duuren
Alf Sommer Landsend

UiO : University of Oslo

Abstract

Feed forward neural networks can be used for both regression and classification problems. In this report we tried to predict the surface of the Franke function using a neural network. Using the SGD algorithm we could compare it to the analytical solutions of OLS and ridge regression, and then also to our own neural network using the same SGD.

Contents

1 Introduction

Artificial neural networks have lately become popular partly because of increased computational capacity. Feed-forward neural networks can among other things be used for classification and regression [1]. In this project we will develop our own feed-forward neural network and study both classification and regression. We will compare the results with the results from our own ordinary least squares and logistic regression code and with results obtained with Scikit-Learn.

2 Methods

2.1 Stochastic gradient descent

In this part of the project we started by replacing in our standard ordinary least squares and Ridge regression codes from project 1 the matrix inversion algorithm with our own stochastic gradient descent (SGD) code.

```
def SGD(X, y, n_epochs, batch_size, eta, lmb=0):
    datapoints = X.shape[0]
    beta = np.random.randn(X.shape[1], 1)
    eta0 = eta
    for epoch in range(n_epochs):
        perm = np.random.permutation(datapoints)
        X_shuffled = X[perm, :]
        y_shuffled = y[perm, :]
        for i in range(0, datapoints, batch_size):
            x_i = X_shuffled[i:i+batch_size, :]
            y_i = y_shuffled[i:i+batch_size, :]
            gradient = (2 / batch_size) * (x_i.T @ ((x_i @ beta) - y_i) +
    lmb * beta)
            beta = beta - gradient * eta
    return beta
```

We tried to predict values that were calculated by means of the Franke function with noise added. For this we used polynomials of degree 10. For ordinary least squares regression we tried different values of the learning rate in order to find the learning rate that gave the lowest MSE value and the highest R2 value. For Ridge regression we made a grid search to find the hyper-parameter λ and the learning rate that gave the highest R2 value.

2.2 Neural network code

We wrote our own feed forward neural network code. As activation function for the hidden layer we used the sigmoid function in this part of the project. The activations of the final output layer were scaled. The weights and biases were initialized, using the standard normal distribution. In order to train the network we used stochastic gradient descent. We studied the Franke function. The results obtained with our neural network were compared to those obtained with ordinary least squares regression and Scikit-learn.

2.3 Testing different activation functions

We tried to replace the sigmoid activation function with the RELU activation function in the hidden layer of our own neural network. We also replaced the RELU activation function with the sigmoid activation function in the hidden layers of the Scikit-learn MLP-regressor.

3 Results

3.1 Stochastic gradient descent

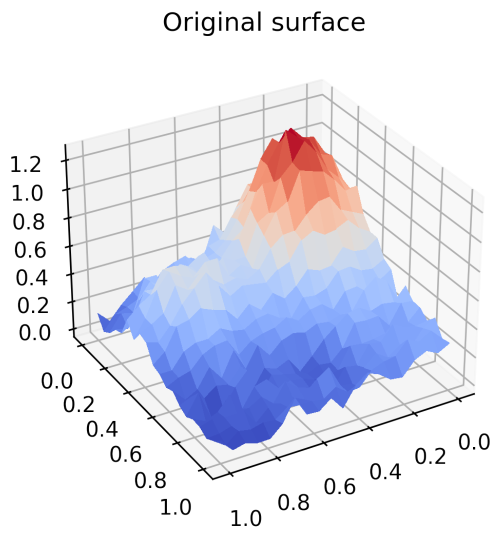

Figure 1: The original surface of the Franke function

Our models were trained to predict the Franke function showed in figure 1. Figure 2 shows the test mean squared error and test R2 respectively as a function of the logarithm of the learning rate. As one can see the learning rate must have a certain size.

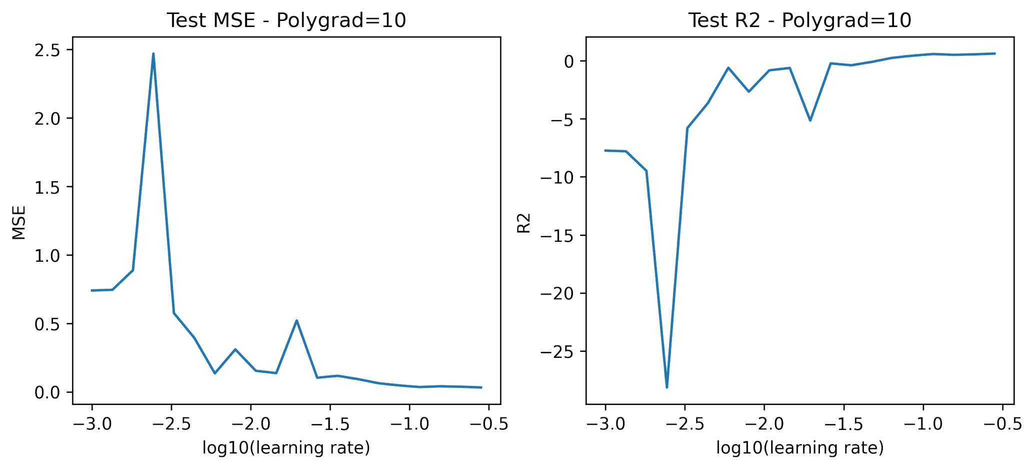

Figure 2: SGD for linear regression as a function of learning rate

The grid search in figure 3 shows that the best value of λ is 0.0001, which is as low as it can be in this grid search, and the best learning rate is 0.1. We wanted to look at values larger than 0.1 for the learning rate, but the function would diverge at a rate around 0.3.

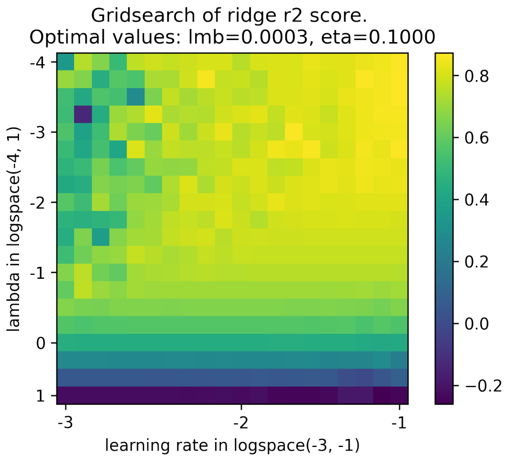

Figure 3: Gridsearch of ridge regression λ and the learning rate

For ordinary least squares regression we found that the SGD code gave higher MSE (0.010) and lower R2 value (0.884) than the matrix inversion algorithm (MSE 0.005 and R2 0.941). The learning rate for the SGD was 0.25.

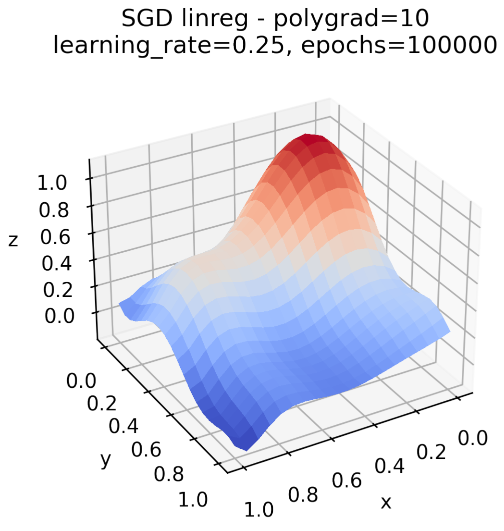

Figure 4: SGD applied to linear regression to predict the Franke surface

As for ordinary least squares regression we also found for Rigde regression that the SGD code gave higher MSE (0.009) and lower R2 value (0.890) than the matrix inversion algorithm (MSE 0.005 and R2 0.943). The results for Ridge regression are marginally better than those for ordinary least squares. The learning rate for the SGD was 0.1 and λ was 0.0001.

3.2 Franke surface predictions

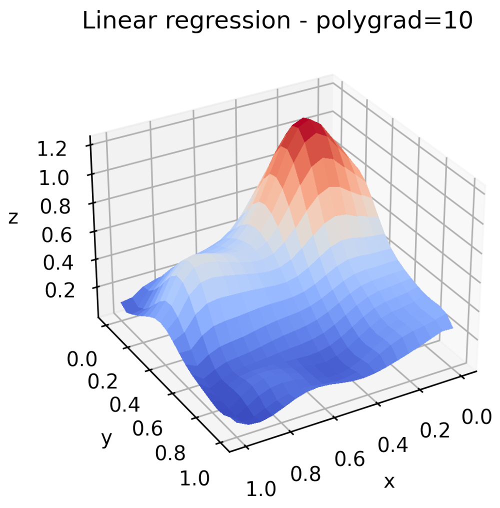

Figure 5: Prediction of the Franke surface with regular OLS. This gave a test R2-value of 0.955 and a test MSE of 0.003.

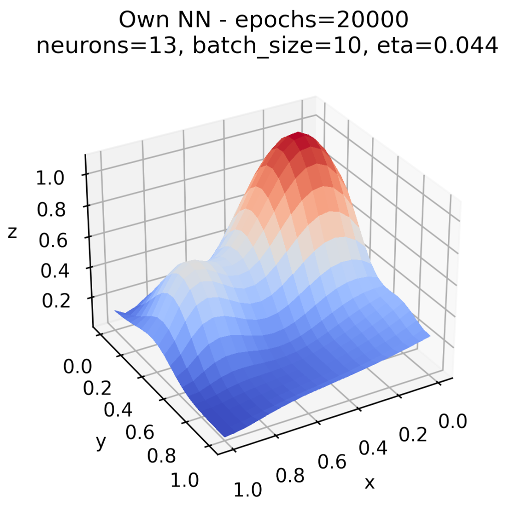

Figure 6: Prediction of the Franke surface with our own neural network. This gave a test R2-value of 0.940 and a test MSE of 0.004.

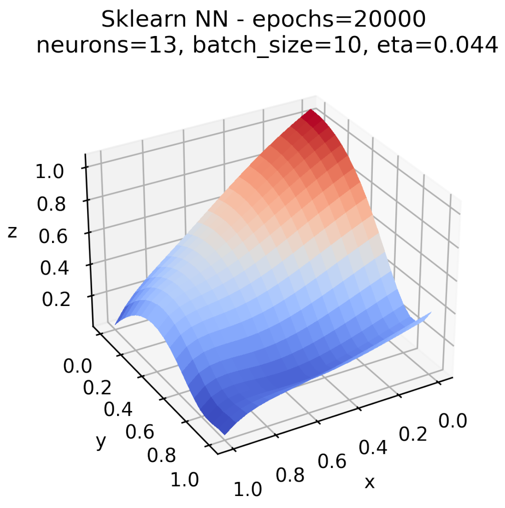

Figure 7: Prediction of the Franke surface with Scikit-Learn NN with the same hyperparameters as figure 6. This gave a test R2-value of 0.824 and a test MSE of 0.012.

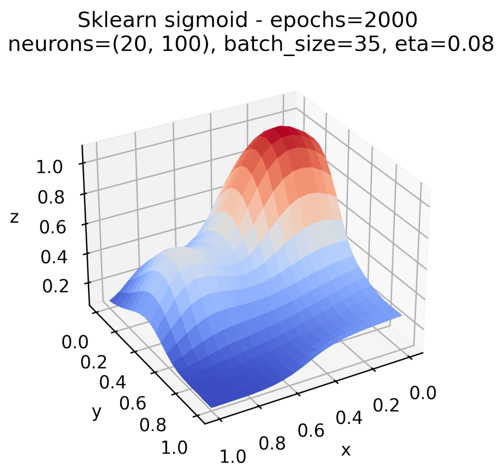

Figure 8: Prediction of the Franke surface with Scikit-Learn NN with the sigmoid activation function. This gave a test R2-value of 0.930 and a test MSE of 0.004.

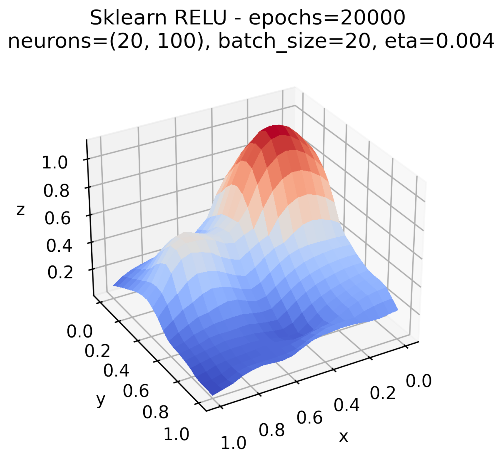

Figure 9: Prediction of the Franke surface with Scikit-Learn NN with the RELU activation function. This gave a test R2-value of 0.950 and a test MSE of 0.003.

	model	R2 score
0	Linreg	0.9554001146751 41
1	Own NN	0.9402793582990677
2	Sklearn NN	0.8243489012673634
3	Sklearn best	0.9504615270498996

Table 1: Comparison of the r2 scores of our different models

3.3 Testing different activation functions

When we replaced the sigmoid activation function with the RELU activation function in the hidden layer of our own neural network and used the network to predict the Franke function, we got an test R2-value of -1497 and an test MSE of 102.1. The graph looked very different form the Franke funktion graph. When the RELU activation function was replaced with the sigmoid activation function in the hidden layers of the Scikit-Learn MLP-regressor, the test R2-value was reduced from 0.950 to 0.931, and the test MSE was increased from 0.00338 to 0.00472.

4 Discussion

When we replaced the matrix inversion algorithm with an SGD code our findings for Ridge regression were consistent with those for ordinary least squares regression. In both cases the matrix inversion algorithm gave the best results. The reason for this could be that the SGD code only finds a local minimum. The fact that the R2-value decreased and the MSE increased when the RELU activation function was replaced with the sigmoid activation function in the hidden layers of the Scikit-Learn MLP-regressor, was as one would expect. In general the RELU function performs better than the sigmoid function.

There could also be some uncertainties regarding this result because we are comparing our own implementation to Scikit-Learn. We didn't use our own RELU fucntion because of issues where our weights diverged and provided results that were not useful. This issue persisted when we were adjusting the network to be used in classification. Logistic regression was not explored in this report.

5 Conclusion

When comparing the SGD algorithm to the analytical solutions for OLS and ridge we got our expected results. With such a high feature count the SGD couldn't quite find the global minimum, but it still did a pretty good job at the prediction. As the learning rate increased, the model also performed better. But after a learning rate of about 0.3 the model would diverge.

When predicting the surface of the Franke fucntion, the best result was achieved with regular OLS. We might have gotten a better result with more tweaking of the hyper parameters, but for this scenario the neural network did not perform better than the analytical solutions, although it only scored 0.005 points lower on the R2 score.

When we compared the activation functions, the RELU function outperformed the sigmoid functions for Scikit-Learn's implementation. The leaky RELU function could also be explored.

Our implementation did not work for classification as the weights in our network would diverge when trying to use different activation function for the hidden layers and the output layer. The difference between logistic regression and classification with neural networks could also be explored.

References

[1] A.K. Jain, Jianchang Mao, and K.M. Mohiuddin. Artificial neural networks: a tutorial. *Computer*, 29(3):31–44, 1996.