FYS-STK3155 - Project 2

Dag Arne Lydvo
(Dated: November 13, 2020)

In this project I will be writing a neural network class to employ at both regression and classification tasks. The network will be compared with both linear regression with and without regularization including a Scikit-learn neural net. Neural networks as proves to be stable on regression tasks and avoids over fitting, this is also true for the Ridge regression. On the classification task the neural networks performs very well, but are dependent on the learning rate and not so much the regularization parameter, this seems to change when adding more layers.

I. INTRODUCTION

This project seeks to utilize neural networks to perform both regression and classification tasks. For the regression section I will be revisiting the Franke function from project 1 and employ a neural network as well as linear regression methods and do a comparison.

In the classification section the neural network will be employed to classify handwritten numbers from the MNIST dataset. I will be looking at different learning rates, regularization parameters and different number of hidden layers in the network. A neural network from Scikit-learn will also be implemented for comparison.

II. METHOD

A. Stochastic gradient descent

The stochastic gradient descent is implemented to improve training speed versus the normal gradient descent which uses the whole training set and can be quite computational expensive. Gradient descent method seeks to minimize a cost function by finding the fastest way to a minimum, preferably a global minimum. The method seeks to reach the point where the gradient of the cost function is zero by iteration tweaking parameters of a model until a minimum is found. In the equation below theta represents the various parameters of a model and gamma is a learning rate. [1]

$$\theta^{nextstep} = \theta - \gamma \boldsymbol{\nabla}_\theta MSE(\theta)$$

The stochastic gradient descent picks random instances of the training sets and uses the sample to compute the gradient. The SGD is much more irregular than ordinary gradient descent. This can be an advantage when trying to avoid getting stuck in local minima.

B. Neural networks

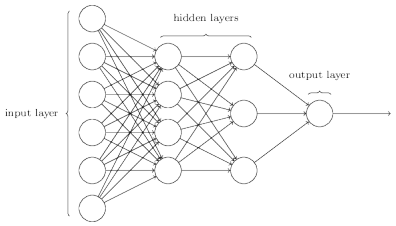
Neural networks are ordered in layers with various number of nodes. A network got an input layer corresponding to the number of features in the design matrix. Then a number of hidden layers before the output layer with number of nodes corresponding to the number of output values. The nodes in the hidden layers are activated by a activation function which is a sum of the input values from the previous layer multiplied by weights corresponding to each connection then a bias term is added, the bias belongs to a a specific node while the biases to every connection to a node.

$$z = \omega^l a^{l-1} + b^l$$

Node input

$$a^l(z) = \sigma(z)$$

Activation function



C. Accuracy score

For classification an accuracy score is used

$$Accuracy = \frac{\sum_{i=1}^{n} I(t_i = y_i)}{n}$$

The accuracy scores measure the number of successful classifications over the number of data points being classified. The function I here produces 1 if classification is successful otherwise 0.

D. Regression methods

For regression in this project in addition to a neural network I will be using the Linear regression and Ridge regression specified in project 1.

E. Datasets

1. *The Franke function*

For the regression part of this project I will be using the Franke function used and described in project 1.

2. *MNIST digits dataset*

For classification the MNSIT dataset of handwritten numbers will be used.

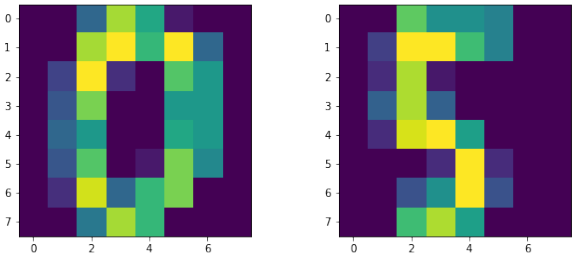


Figure 1. Examples of 8x8 pixels pictures of handwritten figures of the MNIST dataset.

The dataset is imported using the Scikit-learn library and module datasets. There are a total of 1797 numbers, each with 8x8 pixels giving a total of 64 features.

III. RESULTS

Running an experiment with linear regression with and without regularization using stochastic gradient descent over the Franke function, using the Scikit-learn SGD module for comparison.

Learning rate	0.00001	0.00004	0.00013	0.00046	0.00163	0.00584	0.02089	0.07470	0.26710	0.95499
Lambdas										
0.0	29.4	27.52	27.15	27.15	27.15	27.15	27.15	27.15	27.15	27.15

Figure 2. Mean squared error score for Linear regression using SGD

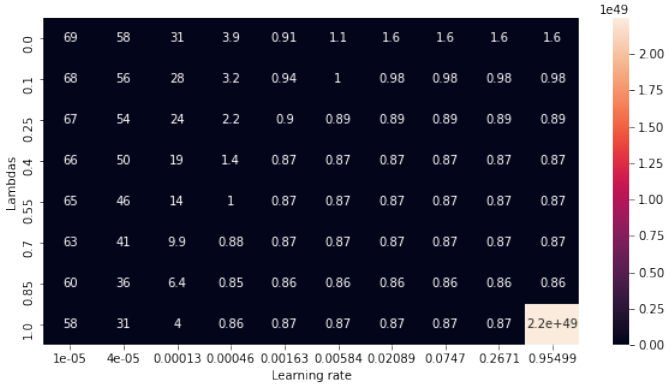


Figure 3. Mean squared error score for Ridge regression using SGD

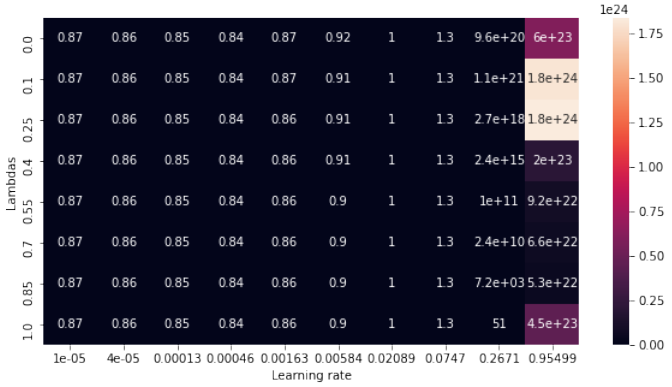


Figure 4. Mean squared error score for SGD Regression using Sklearn with regularization parameter lambda, λ.

Results for regression experiments on the Franke function using a neural network.

Tests using learning rate of 0.1

Testing of different activation functions.

PolyDegree	0			5			10			15			20		
Model	OLS	FFNN	Sklearn NN	OLS	FFNN	Sklearn NN	OLS	FFNN	Sklearn NN	OLS	FFNN	Sklearn NN	OLS	FFNN	Sklearn NN
Learning rate															
0.000010	1.949	25.598	2.097	1.327	26.845	0.742	3.050681e+06	19.716	1.115	1.685535e+08	28.122	1.552	2.508132e+09	22.390	1.158
0.022231	1.960	20.877	1.974	1.493	22.754	0.499	5.525115e+06	25.955	1.052	6.344885e+09	29.073	1.510	4.330268e+09	25.165	1.202
0.044452	1.948	29.798	1.958	1.370	19.070	0.503	4.882060e+07	21.112	1.143	1.837031e+09	25.368	1.708	9.248600e+10	24.951	1.265
0.066673	1.954	27.661	2.023	1.264	23.188	0.577	1.112662e+07	24.166	1.116	3.288897e+10	25.468	1.623	1.408236e+12	24.811	1.227
0.088894	1.950	21.169	2.200	1.518	24.534	0.605	1.175800e+07	21.410	1.154	6.725222e+09	21.773	1.484	1.383304e+10	26.179	1.243
0.111116	1.958	16.878	2.435	1.460	18.617	0.631	8.350315e+07	23.450	1.310	2.403175e+09	24.863	1.504	1.248911e+10	26.891	1.305
0.133337	1.951	14.307	2.040	1.959	26.660	0.739	3.634190e+07	21.137	1.295	1.928201e+08	24.363	1.606	4.725656e+11	28.351	2.003
0.155558	1.960	26.727	2.125	1.496	21.565	0.946	1.632347e+09	19.322	1.264	6.370862e+08	29.455	1.918	4.398385e+10	26.620	1.685
0.177779	1.955	24.469	2.131	1.129	21.754	1.044	7.834854e+05	24.389	1.281	1.849033e+10	25.238	2.015	1.425143e+09	28.562	2.878
0.200000	1.955	24.961	2.161	2.254	23.460	0.930	2.953103e+06	27.866	1.449	3.592577e+09	22.958	2.263	1.037413e+09	24.233	2.972

Figure 5. Mean squared error score for 3 models, OLS, a neural net class and a Scikit-learn neural net over selected model complexities (Polynomial degree) and different learning rates.

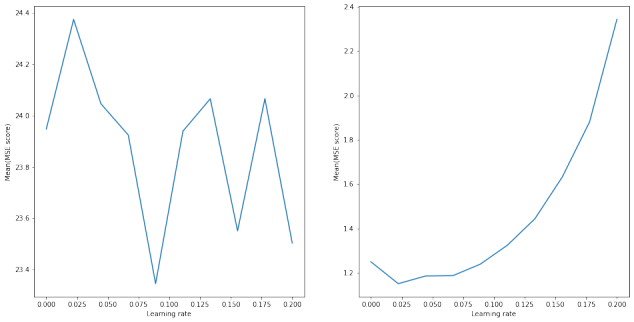


Figure 6. The Mean of the MSE score for all polynomial degrees for the NN class and the Scikit-learn NN, ploted against the range of learning rates.

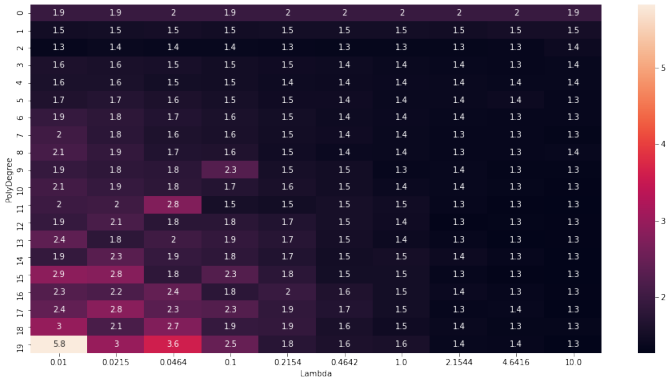


Figure 7. Mean squared error scores for Ridge regression over values of Lambda and Polynomial degrees.

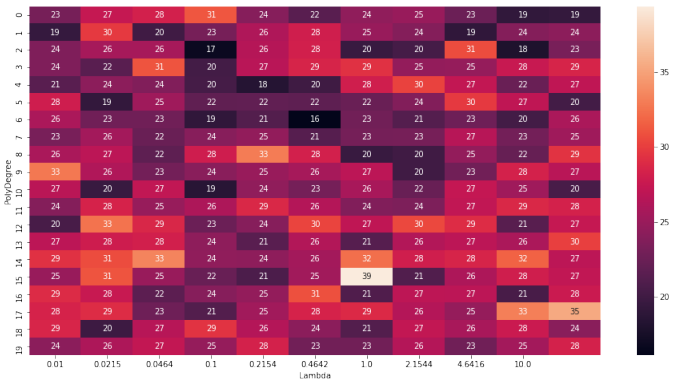


Figure 8. Mean squared error scores for Neural network regression over values of Lambda and Polynomial degrees.

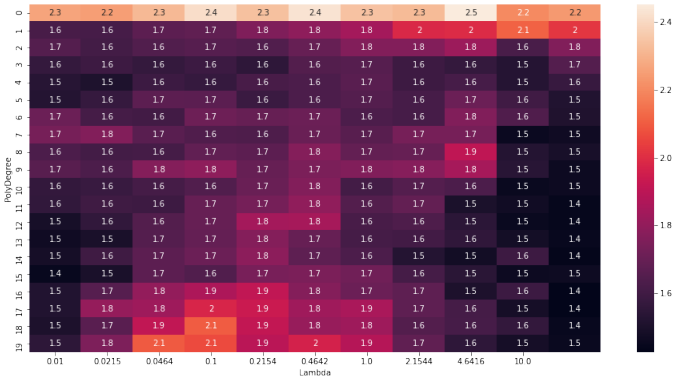


Figure 9. Mean squared error scores for the Scikit-learn Neural network regression over values of Lambda and Polynomial degrees

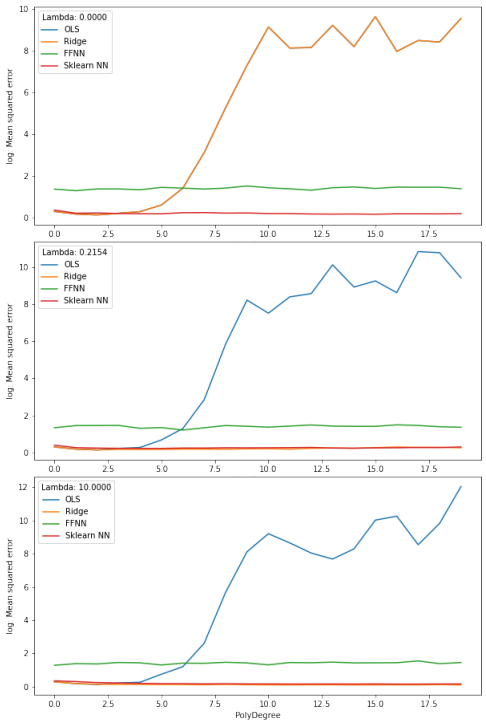


Figure 10. Mean squared error scores for all four models over Polynomial degree, plottet at selected values of lambda.

Model	FFNN-Relu	FNN-LRelu	Sklearn NN-Relu
PolyDegree			
0	86.973	50.491	1.958
1	105.300	123.283	1.562
2	239.196	227.120	1.618
3	508.507	405.424	1.563
4	524.595	624.455	1.624
5	710.719	753.529	1.699
6	987.200	721.467	2.358
7	1286.738	1311.251	3.454
8	1569.780	1471.502	5.523
9	1569.101	1989.291	5.448
10	2013.906	2369.837	10.977
11	2559.794	2337.551	16.124
12	3056.879	2863.528	24.560
13	3894.792	3139.900	23.080
14	3651.509	4248.525	45.460
15	5013.095	3917.416	37.946
16	3867.168	4030.998	68.301
17	7267.082	4506.751	136.924
18	5280.109	5751.855	109.218
19	5430.529	6912.658	148.334

Figure 12. MSE score for RELU and leaky RELU implementation of my neural network at $lambda = 0$. RELU activation for the Scikit-learn implementation.

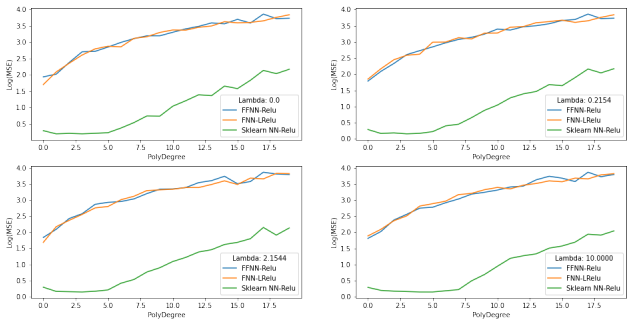


Figure 11. Log(MSE) for neural network implementation using RELU and leaky RELU activation functions for the hidden layer over polynomial degrees.

IV. CLASSIFICATION RESULTS

The following are results from classification experiments on the MNIST dataset...

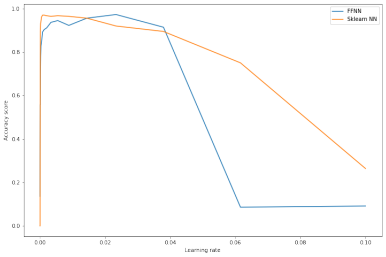


Figure 13. Accuracy score over a range of 20 learning rates from log(-5) to log(-1)

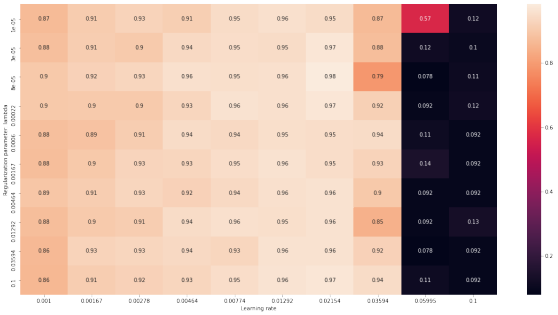


Figure 14. Accuracy score for the neural network over learning rates and regularization parameter lambda.

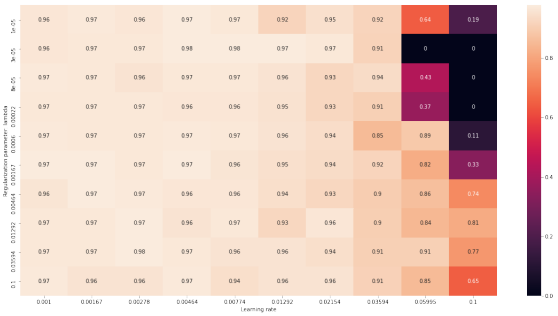


Figure 15. Accuracy score for the Scikit-learn neural network over learning rates and regularization parameter lambda.

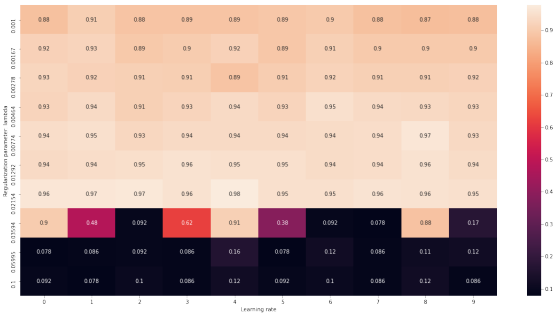


Figure 16. Accuracy score for the neural network with two hidden layers with 100 nodes each over learning rates and regularization parameter lambda.

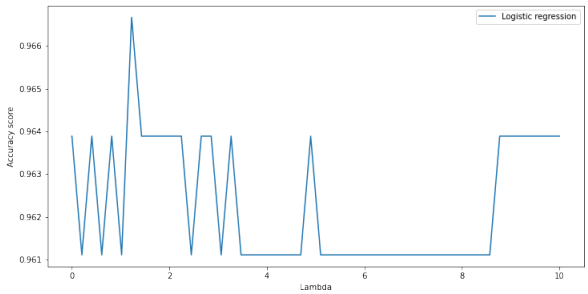


Figure 17. Accuracy score of logistic regression on the MNIST dataset plotted against a range of regularization parameters.

V. DISCUSSION

The stochastic gradient descent method was used to train models on the Franke function for linear regression and Ridge regression with a regularization parameter lambda.The experiment was done over a range of learning rates for all three models. The linear regression seem not to have a very strong dependence on the learning rate, atleast not the range of rates chosen here. This is quite strange as the Ridge regression and Scikit-learns SGD Regression seems to be reacting to a changing learning rate, leading me to suspect some flaw in the code.

The ridge regression have been trained over a range of lambdas from 0.1 to 1 and learning rates ranging from log(-5) to log(-0.02). The ridge model seems to be sensitive to the learning rate and seems to be converging on a MSE score of 0.87 at the higher end of the learning rate range. This is quite close to the MSE score produced by the Scikit-learn model. But the low error score is at the lower range of the learning rates for Scikit-learn whiles my ridge model has it low scores at the high end. This is quite unexpected and could be the result of some flaw in the code or misunderstanding in implementing the SGDRegressor.

Next I implemented a Forward-feed neural network for the Franke function with 100 data points. The network is tested together with OLS regression, ridge regression and an Scikit-learn MLPRegressor implementation. The model was tested using bootstrap with 50 iterations and analysed with a range of 0 to 21 polynomial degrees. The two neural networks was implemented using one hidden layer of 50 nodes and the sigmoid activation function. For regression no activation function is used on the output layer. Number of epochs was 50.

Figure 5 show the MSE score for a selection of the polynomial degrees and learning rates, here with no regularization. At the lower degrees of complexity the OLS method seems to be performing fairly well while my implementation of the neural network is performing quite worse and does not match the Scikit-learn implementation which also performs well. But as complexity increases the OLS MSE seems to blow up while the neural networks remain quite stable. Looking at Figure 6 one sees the mean of the MSE over all polynomial degrees plotted against the learning rate. There is an distinct difference in behavior for the two implementations which I had expected to perform quite similar. The learning rate of 0.1 seems to be close to the optimal for both models and will be used in the analysis of regularization of the models.

Regularization parameter is implemented for a range of log(-2) to log(1) for both ridge regression and the neural networks. The architecture of the networks are the same as before. The ridge regression is performing quite well over the complexity of the model, see Figure 7, suggesting success in avoiding over-fitting of the model. Ridge seems to be performing better towards the higher end of lambda values, but overall remaining quite stable.

The implementation of my neural network seems to be performing quite worse looking at Figure 8. A general higher MSE score overall. The models seems to be quite stable and does not blow up at higher polynomial degrees as did the OLS regression, suggesting it does not over-fit. But it does perform quite worse than the MLPRegression from Scikit-learn which is more in line with the performance of the ridge regression. Figure 10 shows three different plots for different values of lambda, plotting the log(MSE) over the polynomial degrees. It clearly shows the OLS regression blowing up while the ridge and neural networks remaining stable across lambda values.

Looking to other activation functions for the neural networks hidden layer I implemented the RELU and leaky RELU functions for the hidden layer. Looking at Figure 12 we see quite high MSE scores for the neural network implementation and it seems to be blowing up with the polynomial degree. The relative difference between the RELU and leaky RELU seems to be small. The scikit-learn implementation is performing alot better, but it also seems to be blowing up at higher polynomial degrees, as can be seen in figure 11.

A. Classification

The neural network will be implemented for classification together with an MLPClassifier from Scikit-learn. Training on the MNIST dataset, with a training set of 1437 pictures, each with a total of 64 pixels as features. The network is first trained with one hidden layer of 100 nodes at 100 epochs and batch size of 50. Using the sigmoid function for activation in the hidden layer and softmax for the output layer. Network is trainer over 20 different learning rates from log(-5) to log(-1). Figure 13 shows the accuracy score of my network and the Scikit-learn implementation. Both models reaches a max score of about 94% at a learning rate from about 0.01 to 0.025. Before both models suffers in performance as the learning rate increases. The Sklearn models performs a bit better at the higher learning rates.

Using regularization improves the models somewhat, but not a great deal. My neural network reaches a max accuracy score of 96.67 % while the Scikit-learn model reaches 97.2 %. Both models seems to be more dependent on learning rate than regularization. The Scikit-learn model holds up better at higher learning rates and seems to favor higher lambdas at the high end of learning rates.

Trying out adding an additional layer to the network, this one also with 100 layers we see the heatmap flip. The models becomes sensitive to regularization and performs worse at higher lambdas. This was quite unexpected and I do not know how to explain this.

Training a logistic regression on different values for regularization produces scores that are quite similar to the neural networks. The model seems quite stable across regularization parameters.

VI. CONCLUSION

In the regression experiments the neural networks seems to comparable to the Ridge regression, being stable across an increase in the complexity of the model. Even without regularization the neural networks have a stable profile. My own network seems to perform a bit worse than the Scikit-learn implementation, a bit disappointing, although being stable.

On the classification part my neural network seem to perform quit well, able to reach accuracy score of 97 % and here my network seems to be able to keep up with the Scikit-learn implementation. But having some faster decline in performance when learning rate increases. I conclude that regularization did not have a big inpact on the classification models. Learning rate was a far bigger influence with one later. Adding a second layer flipped that and made regularization matter more, this was an interesting discovery.

REFERENCES

[1] Aurelien Geron - Hands-on Machine Learning with Scikit-Learn, Keras & TensorFlow: Page 122, equation 4-7 : Gradient descent step