Project 2

FYS-STK4155

Erik Joshua Røset & Oskar Idland
University of Oslo, Department of Physics
(Dated: November 7, 2024)

In this project, we extend our exploration of machine learning techniques to include advanced optimization algorithms and neural networks. We implement and analyze gradient descent methods, including Stochastic Gradient Descent (SGD), as well as feed-forward neural networks. Our analysis focuses on the mathematical foundations of backpropagation, activation functions, and regularization techniques. We apply these methods to both regression and classification tasks, using synthetic data generated through the Franke function and the Wisconsin Breast Cancer dataset. By comparing our implementations with established libraries like Scikit-learn and PyTorch, we gain insights into the practical considerations of neural network development and optimization. Our results demonstrate the importance of activation functions, initialization strategies, and regularization methods in training effective neural networks. We also highlight the impact of optimization techniques on model convergence and generalization, providing a comprehensive overview of the fundamental building blocks of modern machine learning systems.

https://github.com/Oskar-Idland/FYS-STK4155-Projects

I. INTRODUCTION

II. THEORY

III. METHODS & IMPLEMENTATION

IV. RESULTS & DISCUSSION

A. Regression Analysis

1. *Plain and Stochastic Gradient Descent*

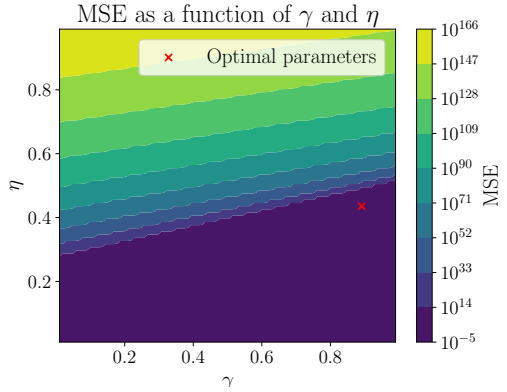

FIG. 1: Overview of the entire parameter space, plotting learning rate against momentum, using plain gradient descent. The optimal parameters are marked with a red cross, to use as a starting point for further analysis.

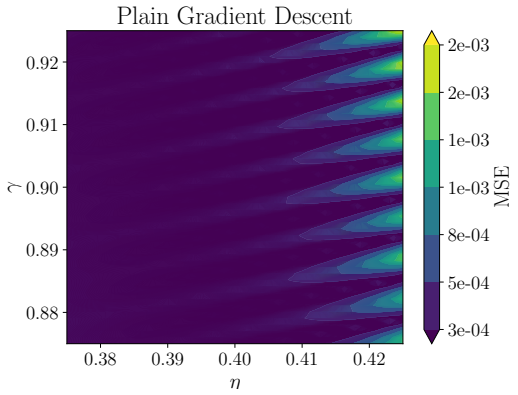

FIG. 2: Narrowed down parameter space from fig. 1.

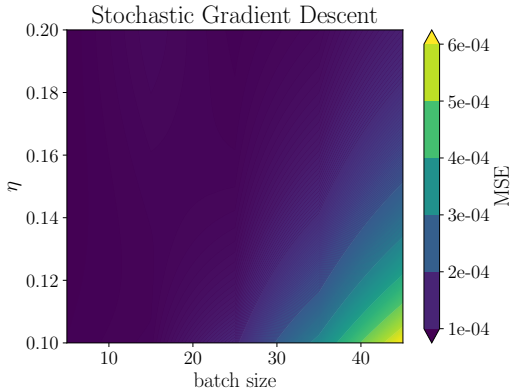

FIG. 3: Plotting batch size, against learning rate, using stochastic gradient descent.

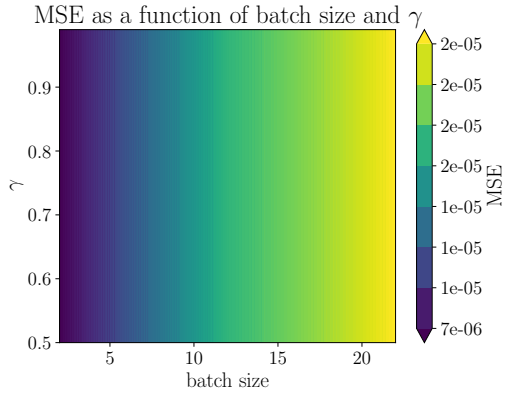

FIG. 4: Plotting the batch size against the momentum, using stochastic gradient descent. Using a learning rate of 0.2

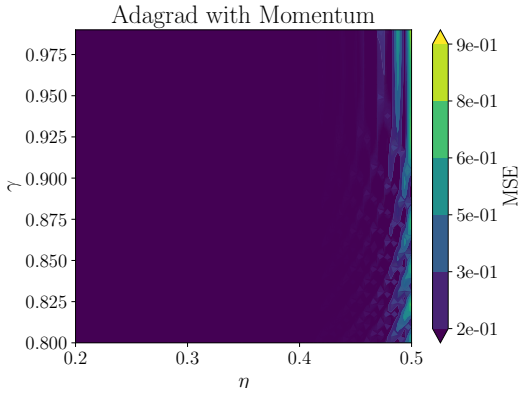

FIG. 5: Plotting the learning rate against the momentum, using regular AdaGrad

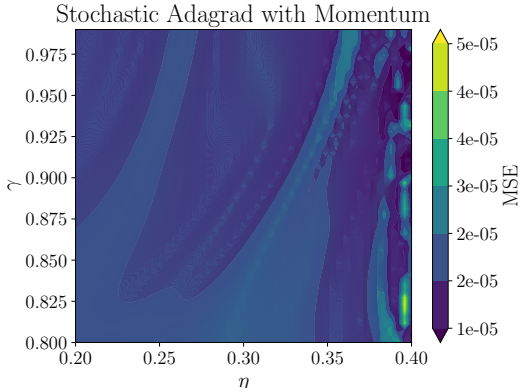

FIG. 6: Plotting the learning rate against the momentum, using stochastic AdaGrad. The batch size is set to 20.

2. *AdaGrad*

3. *RMSprop*

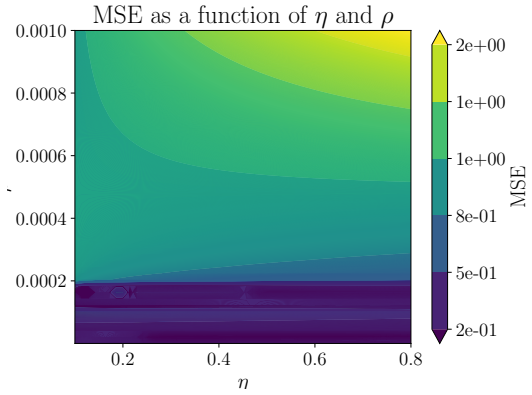

FIG. 7: Plotting the learning rate against the decay rate, using RMSprop

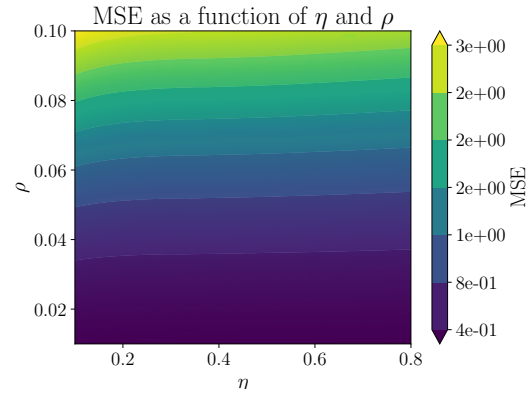

FIG. 8: Plotting the learning rate against the decay rate, using stochastic RMSprop. The batch size is set to 20.

4. *Adam*

Look at fig. 1 and fig. 2 we see a clear interval of parameters which gave low MSE values. Using the best parameters found as a starting point, we continued searching fig. 3 and fig. 4 to find an optimal batch, of between 5 and 20. Exploring further with fig. 5 and fig. 6 we found that the stochastic performed better then lpain gradient descent version with about 4 orders of magnitude. From fig. 7 and fig. 8 it seems that both versions underperformed with an MSE with around $10^{-}-1$. Looking closer at the MSE values of the stochastic variant, we found some spots with MSEs with a value around 10^{-5}. Lastly, fig. 9 and fig. 10 performed similar as RMSprop,

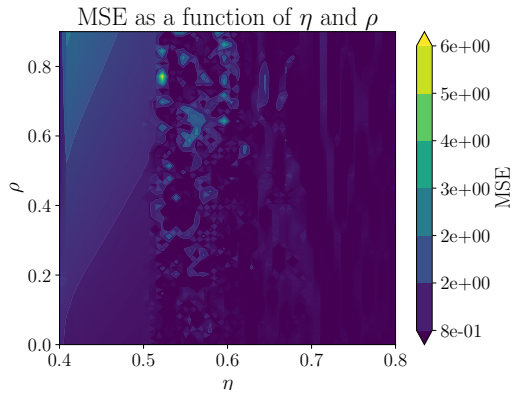

FIG. 9: Plotting the learning rate against the decay rate. We have set the same value for ρ_1 and ρ_2, using Adam

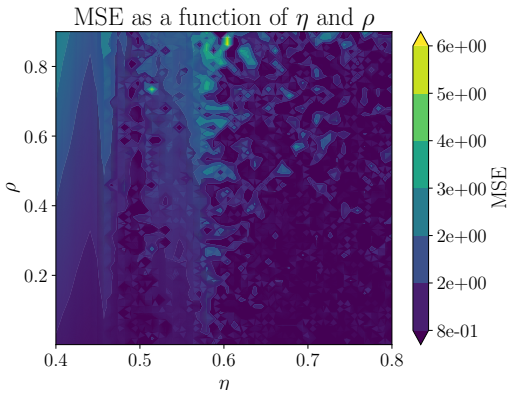

FIG. 10: Plotting the learning rate against the decay rate, using stochastic Adam. The batch size is set to 20, and the same value for ρ_1 and ρ_2 is used.

in general there where too high MSE values, but with spots in the same order of magnitude.

B. Neural Networks Regression

Our analysis of feed-forward neural networks began with the Franke function regression problem, allowing us to validate our implementation and explore the effects of various hyperparameters. The results demonstrate that our neural network implementation achieves robust performance across a wide range of configurations, with optimal models achieving MSE values around 10^{-3} and R^2 scores above 0.95.

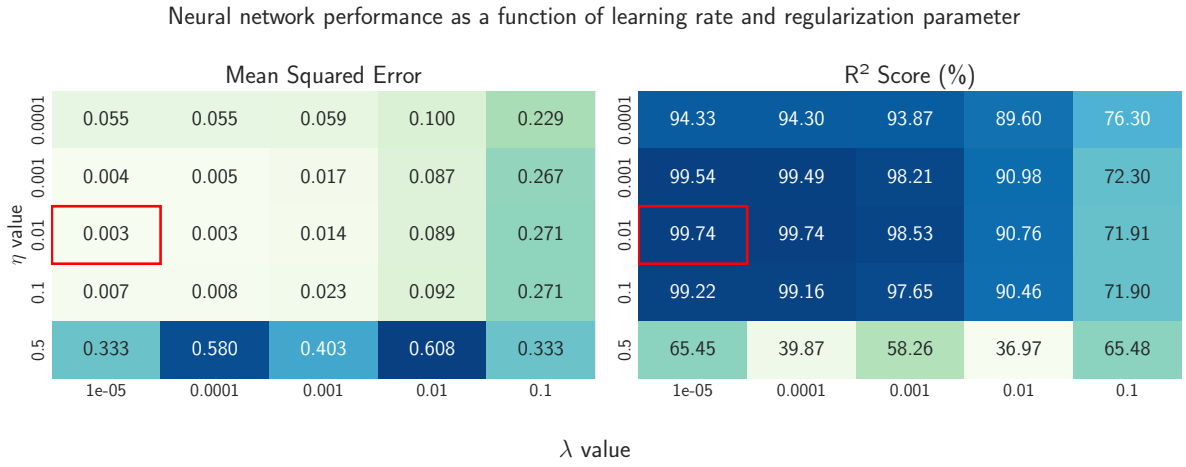

FIG. 11: The effect of learning rate and regularization strength on the Franke function regression problem. The plot shows the MSE and R^2 scores for different combinations of learning rate and regularization strength, with the optimal values highlighted in red.

Our feed-forward neural network demonstrated consistently strong performance on the Franke function regres-

sion task across a range of hyperparameters. As shown in fig. 11, the model achieves stable MSE scores around 10^{-3} and R^2 scores above 0.99 for combinations of learning rates η in the range 10^{-1} to 10^{-3} and regularization strengths λ in 10^{-3} to 10^{-5} . The model only shows significant performance degradation with the highest tested learning rate ($\eta = 0.5$), suggesting robust behavior across most of the hyperparameter space.

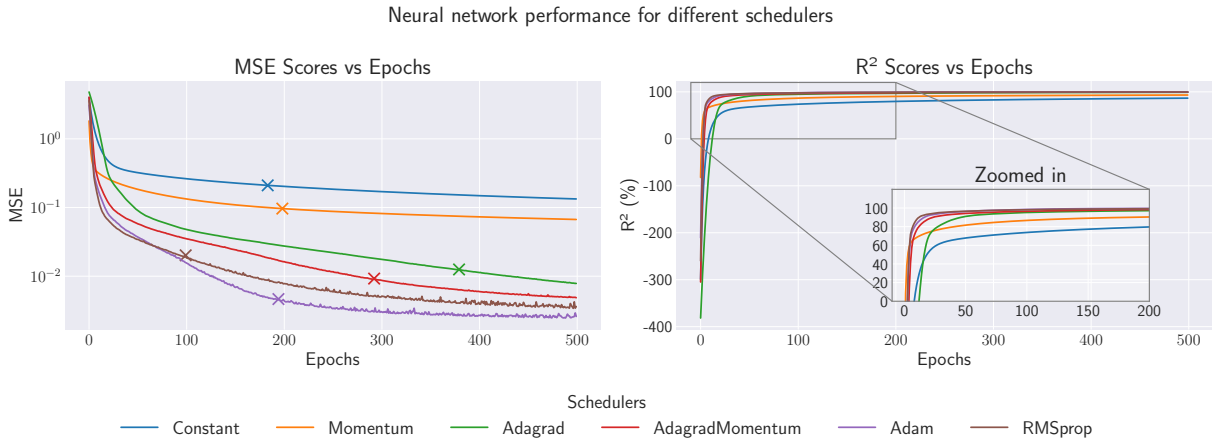

FIG. 12: The effect of different learning rate schedulers on the Franke function regression problem. The plot shows the training MSE and R^2 scores for different learning rate schedulers as a function of epochs. The schedulers MSEs are marked with a cross indicating at which epoch convergence was reached.

The comparison of different optimization schedulers in fig. 12 shows that all implemented methods have varying success with Adam and RMSprop performing the best. The marked convergence points indicate that both Adagrad methods are slower to converge than the other methods, while RMSprop reach its point around just 100 epochs. As the criteria for convergence is chosen somewhat arbitrarily, as describe in **??**, the marked crosses may not showcase where the schedulers converge, but gives insight into when the learning rates are slowing down relative to eachother as the networks are training. The figure also suggest that given enough epochs, both Adagrads, Adam and RMSprop will converge to an mse of just short of 10^{-3} and an R^2 score of 0.95. While constant learning rate and plain momentum will perform worse.

Testing different activation functions (fig. 13), reveals similar performance levels among non-linear functions. The sigmoid activation in the hidden layer performs slightly better with an MSE of 0.005 and R^2 of 0.995, but ReLU and Leaky ReLU follow closely with MSE around 0.011 and R^2 of 0.989. As expected, removing the non-linear capability of the network by using the identity function in the hidden layer results in significantly worse performance, with an MSE of ≈ 0.09 and R^2 of ≈ 0.91.

Comparing our results with the PyTorch implementation in fig. 14, we see that our model performed similarly to the PyTorch model, with an MSE around 10^{-3} and an R^2 score around 0.95. The main takeaway from this is that our models were faster to reach stable performance, wile the PyTorch model took more epochs to become stable at slightly better performance. This might suggest that the PyTorch model is more robust to overfitting, but at the cost of longer training times.

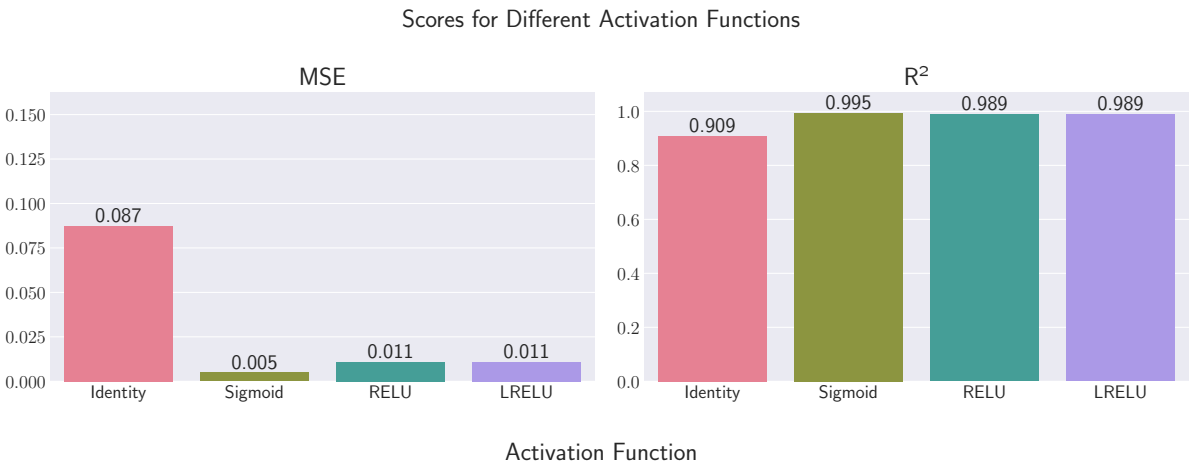

FIG. 13: The effect of different activation functions on the Franke function regression problem. The plot shows the test MSE and R^2 scores for different activation functions.

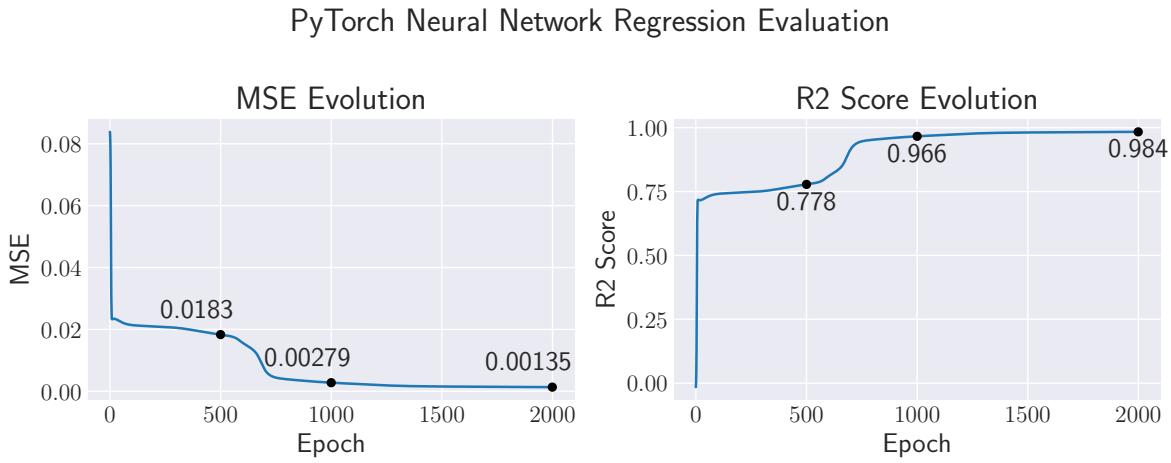

FIG. 14: PyTorch neural network regression on the Franke function. The plot shows the training and test MSE and R^2 as a function of epochs. The plots are annotated with the values at epochs 500, 1000 and 2000

C. Neural Networks Classification

Our feed-forward neural network achieved strong classification performance on the Wisconsin Breast Cancer dataset. As shown in fig. 15, the model maintains high accuracy ($< 95\%$) across a broad range of hyperparameter combinations. Performance is most dependent on the learning rate, having optimal performance for η between 10^{-1} and 10^{-2}, while the regularization strength is less critical with a slight trend for better performance with lower λ values. In the figure, the optimal values for training accuracy and test accuracy differ. Since the dataset is small, the model is prone to overfitting and this could be the reason for the discrepancy.

The exploration of network architectures demonstrates that model performance remains robust across different configurations. Networks with 2-3 hidden layers and 15-25 nodes per layer consistently achieve test accuracies above 96%. Notably, very small networks (5 nodes) show degraded performance, particularly with three layers where training accuracy drops to 61.5%, suggesting insufficient model capacity. Our initial assumptions that it would suffice to have a small network for this dataset are somewhat backed up by the results, but the faster runtimes do have a tradeoff in performance.

All tested activation functions perform similarly well, with accuracy variations within 2-3 percentage points. In single-layer configurations, sigmoid achieves slightly better performance, while ReLU and Leaky ReLU trends to better performance in deeper networks. This is only true for the training data, as all test accuracies degrade with more layers. This again suggest the danger of overfitting, and that the dataset is too small to support deep networks.

From fig. 18, we see that the PyTorch model is a more stable model, taking more time to reach high performance levels, but manages to hold a test accuracy of $< 98\%$ after 500 epochs. As our models are often scoring above 98% after only 20 epochs. A possible explanation for this is that the PyTorch model is a more advanced model, increasing the difficulty to train the network compared to our relatively simple model for a small data set as the Wisconsin Breast Cancer data.

D. Logistic Regression

Our logistic regression implementation achieves high performance on the breast cancer dataset, with test accuracies consistently above 96% across most hyperparameter combinations. fig. 19 shows that performance is robust across a wide range of learning rates and regularization strengths, with optimal test accuracy of 98.2% achieved at $\eta = 0.0001$ and a regularization strength of 0.1. The model demonstrates good generalization, with test accuracies closely matching training accuracies across the parameter space.

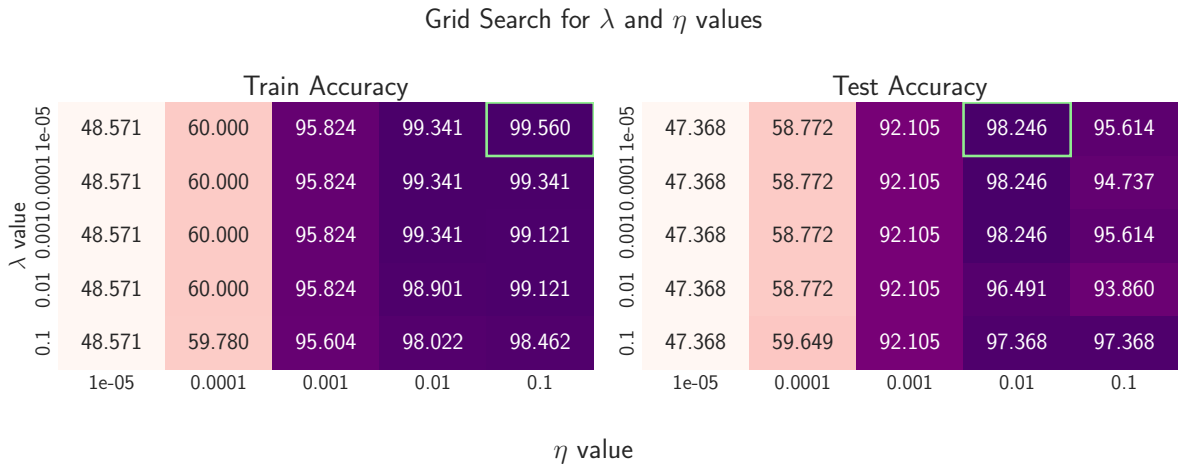

FIG. 15: Model performance for different learning rates and regularization strengths on the breast cancer classification problem. The plot shows the training and test accuracy scores for different combinations of learning rate and regularization strength. The optimal values are highlighted in green.

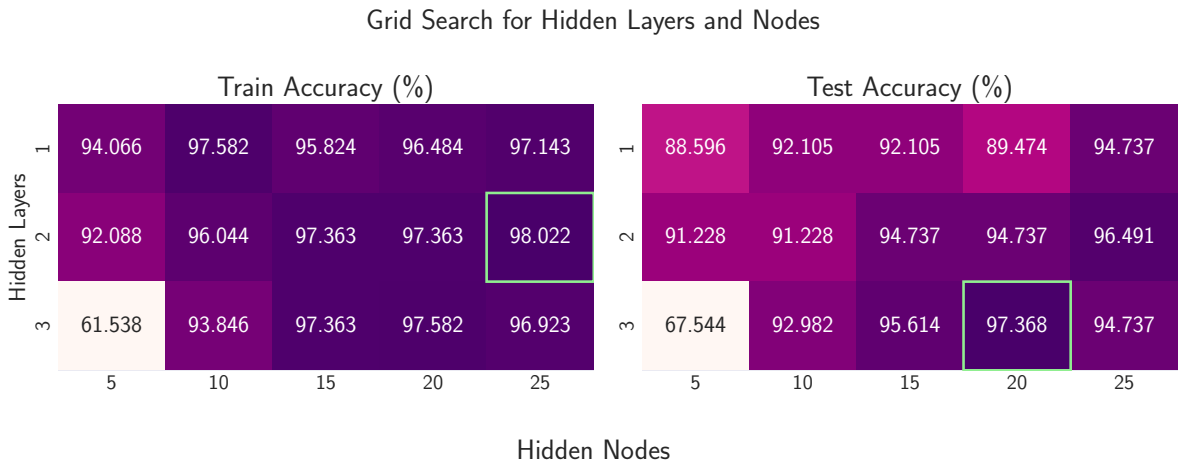

FIG. 16: Model performance for different network architectures on the breast cancer classification problem. The plot shows the training and test accuracy scores for different numbers of hidden layers and nodes. The optimal values are highlighted in green.

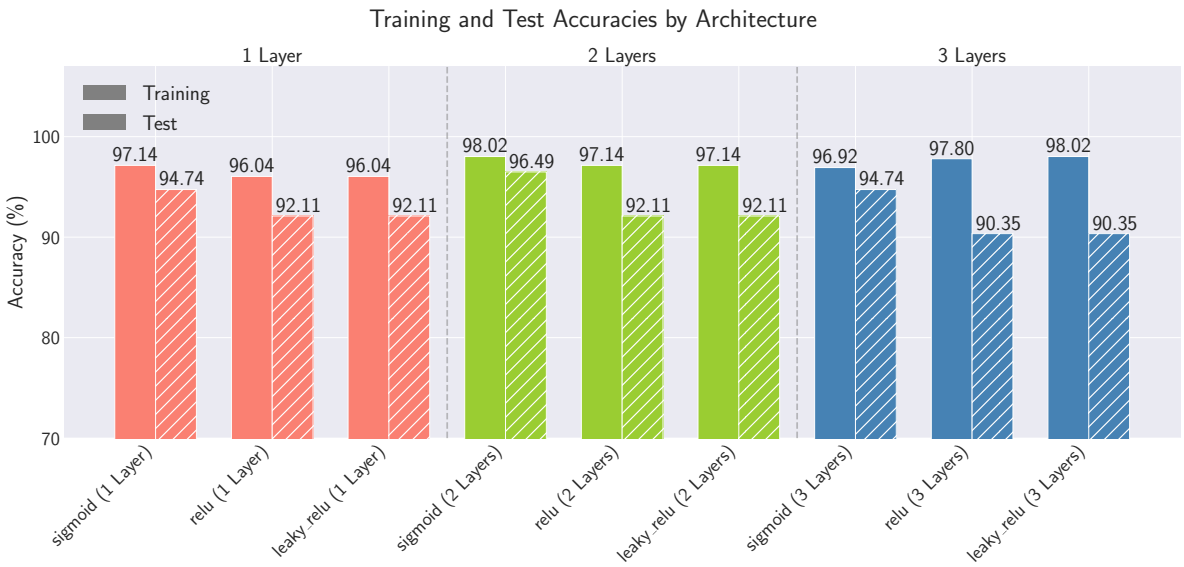

FIG. 17: Model performance for different activation functions on the breast cancer classification problem. The plot shows the training and test accuracy scores for different activation functions and numbers of hidden layers.

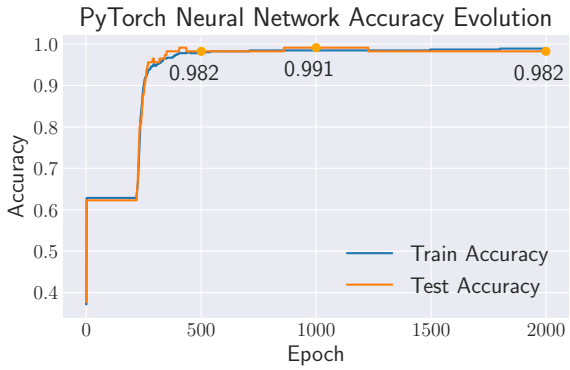

FIG. 18: PyTorch neural network classification on the breast cancer dataset. The plot shows the training and test accuracy as a function of epochs. The figure is annotated with the test accuracies at epochs 500, 1000 and 2000.

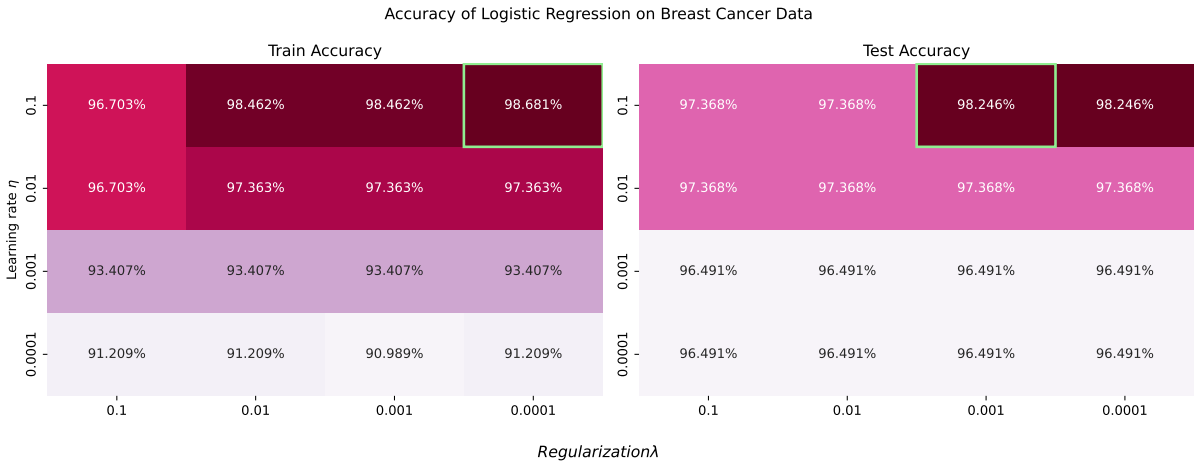

FIG. 19: Model performance for different learning rates and regularization strengths on the breast cancer classification problem using logistic regression. The plot shows the training and test accuracy scores for different combinations of learning rate and regularization strength. The optimal values are highlighted in green.

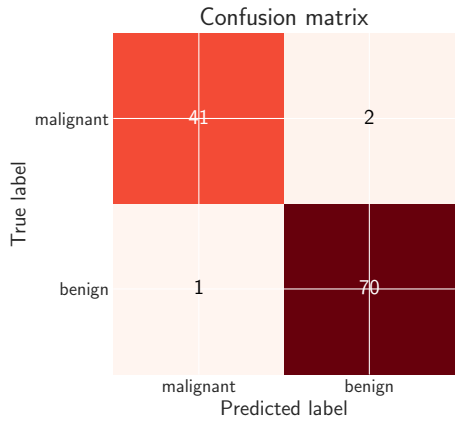

FIG. 20: Confusion matrix for the breast cancer classification problem with Skikit's Logistic Regression. The plot shows the confusion matrix for the test set, with the number of true positives, true negatives, false positives, and false negatives.

The confusion matrix from scikit-learn's implementation (**??**) shows excellent classification performance with only 3 misclassified samples out of 114 test cases. The model correctly identified 41 out of 43 malignant cases and 70 out of 71 benign cases, demonstrating balanced performance across both classes. The similar performance between our implementation and scikit-learn's validates our approach while suggesting that the classification task may be well-suited for linear decision boundaries.

V. CONCLUSION

In our study, our regression analysis proved mostly successful. With the exceptions of finding a good range of values for the Adam and RMSprop schedulers, although individual values were found. The neural network analysis proved to be more successful, performing well across the board. Our classification model were exceptional and scored better than the PyTorch implementation. Although on such a small dataset as the breast cancer dataset, one should be critical to the amount of training possible.

REFERENCES

VI. CODE

Link to our GitHub repository: https://github.com/Oskar-Idland/FYS-STK4155-Projects