Evaluating Feed Forward Neural Networks on Classification and Regression Tasks
FYS-STK4155 - Project 2

Erik Berthelsen, Morten Taraldsten Brunes, Johanna Annie Kristina Tjernström
Department of Geoscience, University of Oslo

Feed Forward Neural Networks (FFNN's) are fundamental tools in modern day machine learning, significantly advancing the capabilities of data analysis. Such networks are capable of capturing complex, non-linear relationships in data that linear models cannot. There is a need to broaden the understanding and knowledge of neural networks, the use of different hyperparameters and it's performance on regression and classification tasks. We coded our own feed forward neural network code with back-propagation, different optimizers and norms. The code is adaptable to an arbitrary number of hidden layers, input features and predictors. We have analyzed regression on the Runge function and classification on the well known MNIST database of handwritten digits, both with our own code and an established python library for a comparative study. Our own code and established python library yields the same results. The analysis show that choice of hyperparameters are important and must be chosen dependent on dataset. This imply that several numerical analysis with different hyperparameters must be conducted to find the model that generalized the best to unseen data, independent if the task is regression or classification. **To be continued**

I. INTRODUCTION

Machine learning models provide the ability to learn and model non-linear patterns in complex datasets, where traditional approaches often fail. Nowadays large amount of data are available and machine learning has developed to algorithms that derive information from data. A subset of machine learning is supervised learning which use labeled data to train a predictive model, which is used to make predictions on future data. This method can be used for discrete classification and continuous regression tasks [5]. Feed forward neural networks uses a defined mapping $y = f(x;\theta)$ to capture these relationships by learning the best values for the parameters θ that results in the best approximation of the true function [3]. However, their dependence on hyperparameters and architecture strongly affects the performance of such models. In this report we consider regression on the Runge phenomenon [1] and classification on the well known MNIST database of handwritten digits [4]. We have coded our own implementation of FFNN with different optimizers and norms, while also being flexible to the number of hidden layers, input features and predictors. We also did comparative studies between own code and open source python library to ensure that numeric computation were correct. Analysis were done to evaluate which hyperparameters to use to train the best models. For the regression task we also compared with an analytical Ordinary Least Squares computation. The methods used in this report are described in detail in section II. The results of regression and classification with FFNN, as well as any insights are presented in II B and II C. **This must be revised and further elaborated when report is completed**

II. METHODS

A. Feed Forward Neural Network

FFNNs has a architectural design where input data flow from input layer, through hidden layers to output layer. In input layer the number of neurons (nodes) corresponds to the number of features in input dataset. Number of hidden layers and neurons within them are customizable and considered hyperparameters in the model architecture. The output layer contains a number of neurons equal to the number of target variables, or parameters to be predicted. Each connecting line between neurons in hidden layers represents a weight, and in each node biases and weights are updated by use of a activation function. Learning is achieved by adjusting weights and biases to minimize a cost function using gradient descent optimization methods like ADAM, RMSProp and Stochastic Gradient Descent. This section will describe steps and functions in a FFNN algorithm.

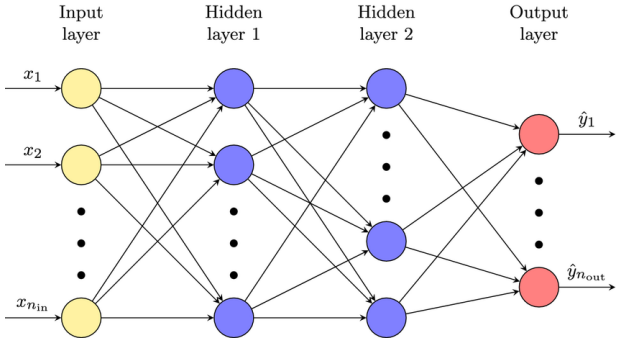

Figure 1: Schematic structure of FFNN [2].

1. Activation functions

Activation functions are used in FFNN to introduce non-linearity, without the FFNN would actually only be a linear transformations. Hence, activation functions are a vital part of FFNN to learn complex patterns. These functions map and control the output range from input values at each neuron. Activation functions are chosen by their capabilities for regression or classification tasks, and it is normal to have the same activation function in all hidden layers and another activation function in output layer.

The sigmoid function map input values to the range (0,1) and can be considered as probabilities.

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

In figure 2 the x-axis represents input values in range (-5, 5) and y-axis represents values from sigmoid function. The blue line is the sigmoid function. Red points exemplify input values to a neuron that is mapped onto the sigmoid function. The steepest part of the curve is around 0, smaller values of x give large change in activation function. For very small or very large input values this function can lead to vanishing gradient problem. With output from function very close 0 or 1 the update of weights become negligible and learning slows down, stops or never converge.

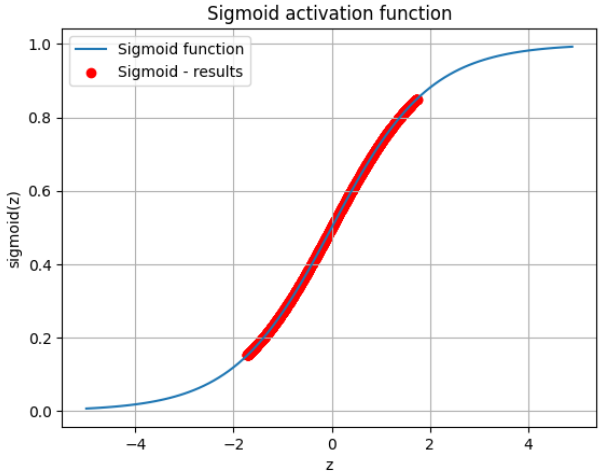

Figure 2: Sigmoid function and mapped values.

A simpler activation in mathematical terms is the ReLU function. Any negative input value is mapped to 0, while any positive input value is identical for output value. Hence, the range is $(0, \infty)$. This activation function is quite common in neural networks and avoid vanishing gradient problem for positive values. For negative values neurons can be stuck at 0 and neurons are inactive and stops learning (dying ReLU).

$$f(x) = \max(0, x)$$

Leaky ReLU can avoid dying ReLU with a bit more complex function where a small term α is added to negative input values. Value of α must be chosen with care, with to small value will behave like ReLU and to large can distort training.

$$f(x) = \begin{cases} x & \text{if } x > 0, \\ \alpha x & \text{if } x \leq 0 \end{cases} \tag{1}$$

2. Regularization terms

Regularization terms are used in machine learning to punish overly large weights, which can lead to overfitting. **L1** and **L2** are two commenly used regularization terms that add a penalty to the cost functions.

$$C_{\mathbf{total}} = C_{\mathbf{data}} + \lambda_1 \sum_1 |w_1| + \frac{\lambda_2}{2} \sum_i w_i^2 \tag{2}$$

Where $C_{\mathbf{data}}$ is the data loss and λ_1 and λ_2 are hyperparameters that controls the strength of the regularization.

L1 regularization, also known as the Lasso regularization, adds an absolute value for the weights, resulting in small and insignificant values turns to zero. **L2** regularization, or Ridge regularization, adds a squared value to the weights, hindering scaling problems where a single weight becomes very large.

In the gradient computation, **L1** and **L2** modify the weight update as follow:

$$\frac{\partial C_{\mathbf{total}}}{\partial w_i} = \frac{\partial C_{\mathbf{data}}}{\partial w_i} + \lambda_1 sign(w_i) + \lambda_2 w_i \tag{3}$$

Where the regularization terms pushes the weights towards zero, which reduces the model complexity and improves generalization on unseen data.

3. Cost functions

Cost functions measure how well a predicted value from the forward pass fits the target values. Common cost functions are Mean Squared Error for regression and cross-entropy for classification. The derivative of the cost function is used in back propagation. The Mean Square Error is given by

$$MSE(\boldsymbol{y}, \tilde{\boldsymbol{y}}) = \frac{1}{n} \sum_{i=0}^{n-1} (y_i - \tilde{y}_i)^2, \tag{4}$$

where y are the data and $\tilde{y}$ are the predictions. The MSE describes the mean of the squared differences between the model and the data. The smaller the MSE is, the closer the predicted values are to the real ones.

4. *Forward pass*

The forward pass receives input data where each feature corresponds to one neuron in the input layer. The data propagate forward through all hidden layers and output layer. Computed values for weights and biases in each layer and neuron are stored to be used in back-propagation.The output layer contains predicted values for one or more predictors. These values are input to the loss function.

5. *Backpropagation*

The gradient for each each weight and bias from the forward pass is computed from the loss function. The gradients in backward propagation then tells how much each bias and weight contributed to the error.

6. *Gradient descent methods*

Optimizers like ADAM, RMSProp and Stochastic Gradient Descent (SGD) are used to update each weight and bias by the gradients.

7. *Repeat*

This process in repeated until loss converges at a satisfactory level or stops at a given maximum iterations.

8. *Pseudocode for own implementation of FFNN*

backpropagation, one of them, choose late **like in project 1?**
class-diagram for object oriented?

9. *By use of python libraries*

they use some optimization we don't...?

B. Regression Results & Analysis

This figure is implemented by sklearn, in a potential heatmap at the end of the report, we would use the neural network code. Although, the heatmap is generated from the part c code in the project. Other than that, the figure includes x, y and z labels with a sensible output as well.

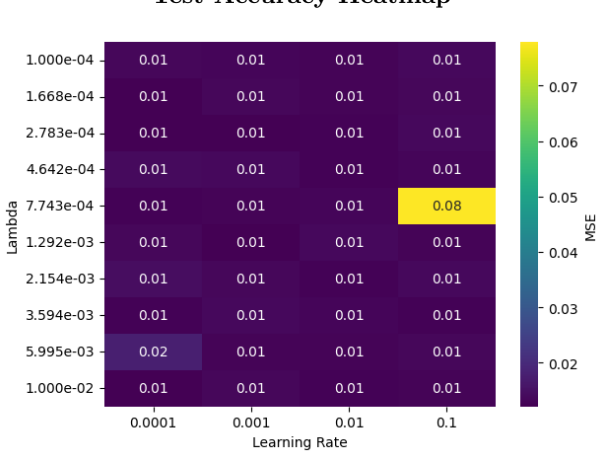

Figure 3: Test accuracy heatmap with [1, 2, 3, 4] hidden layers, [5, 10, 25, 50] nodes and with 1000 iterations

C. Classification results & Analysis

D. Hyperparameter & Architecture Exploration

E. Impact of Gradient Methods & Learning Rates

F. General Discussion

III. CONCLUSION

IV. CODE AVAILABILITY

The code for this project is available at: `https://github.com/johtj/data_analysis_ml_projects/tree/main/assignment2`

[1] W. contributors. Runge's phenomenon. `https://en.wikipedia.org/wiki/Runge%27s_phenomenon`. Accessed: 2025-10-29.

[2] D. Coscia, L. Meneghetti, N. Demo, G. Stabile, and G. Rozza. A continuous convolutional trainable filter for modelling unstructured data. *Computational Mechanics*, 72:1–13, 03 2023.

[3] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, https://www.deeplearningbook.org/, 2016.

[4] Y. LeCun, C. Cortes, and C. J. Burges. The mnist database of handwritten digits. `https://www.kaggle.com/datasets/hojjatk/mnist-dataset/data`.

[5] S. Raschka, Y. Liu, and V. Mirjalili. *Machine Learning with PyTorch and Scikit-Learn*. Packt Publishing, https://sebastianraschka.com/blog/2022/ml-pytorch-book.html, 2022.

APPENDIX