Neural network models applied to Runge's function and MNIST datasets
FYS-STK4155 Project 2

Helene Lane and Manuela Leal Nader
University of Oslo
(Dated: November 10, 2025)

Versatility is one of the key features of neural networks. Not only it can be applied to regression tasks, but it can also be used for classification. In previous work, we found that Ordinary Least Squares (OLS) gave the best approximation of Runge's function out of the tested models. In this report, we have approximated Runge's function with a feed forward neural network. A neural network with only one hidden layer with 50 nodes already performed better than Ordinary Least Squares, with an R^2 of 0.8835 compared to 0.8472. To showcase its versatility, we applied the network on the MNIST data set, which gave a 93,5% accuracy in recognizing the digits and a 81,4% accuracy on recognizing the clothing items from Fashion MNIST. Even though the feed forward neural network is more complicated to implement, it gave better results for approximating Runge's function and it could also be applied to a broader selection of tasks like classification, as shown with MNIST and fashion-MNIST.

I. INTRODUCTION

Neural networks have gone from a niche field in machine learning to something everyone is talking about. However, most people are talking about large language models (LLMs) or image and video generation. As researchers, this is not the use cases we are interested in. We are interested in finding good models to categorize our research data.

Models can be divided into two categories, regression and classification [1]. Regression models are the ones that deals with continuous functions, while classification models aim to predict discrete ones.

In this report, we will compare two different methods to find a good fit for Runge's function, ordinary least squares (OLS) or a neural network with hidden layers.

Runge's function can be difficult to approximate with higher order polynomials [2]. Therefore, we aim to compare a polynomial approximation, in the case of OLS, with a non-linear fit, exploiting the power of neural networks.

As a starting point we used a neural network which performed well on Runge's function to categorize the MNIST data set [3]. Here, we used plain gradient descent with RMSProp. The neural network had one hidden layer with 200 nodes.

In section II, we briefly cover the linear regression methods used in this project. This section also covers neural networks and the choices made in the implementation for this project. We cover cost and activation functions commonly used in neural networks, before describing how our feed forward neural network works. This section is concluded with a description of our implementation and a description of our use of AI tools.

In section III, we will present our results comparing the performance of our neural network on approximating Runge's function to the best regression model from our previous work. Then, we will present the results from the categorization task using our neural network. Finally, section IV concludes our report with a summary.

II. METHODS

A. Data

1. Runge's function

Runge's function behaves in a peculiar way, approximating it with higher order polynomials does not imply a better fit [2]. The one dimensional function is:

$$f(x) = \frac{1}{1+25x^2}. \tag{1}$$

2. MNIST

MNIST is a freely available data set consisting of images of hand drawn digits [3]. This data set is commonly used to train and benchmark models in machine learning. A sample of its 70000 elements (56000 for training and 14000 for test purposes) is depicted in figure 1.

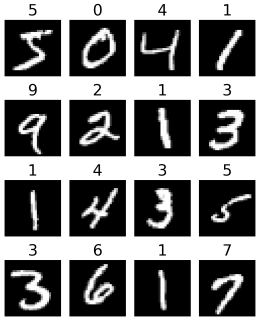

Figure 1. Sample from the MNIST data set.

3. Fashion MNIST

Instead of handwritten numbers, the Fashion MNIST data set consists of images of clothing items from the website of the online retailer Zalando [4]. Each image is labeled with a number that indicates which type of item it is. For example, ankle boots are labeled with the number 9 and dresses with the number 3. Figure 2 shows some examples.

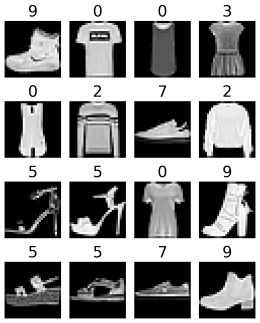

Figure 2. Sample from the Fashion MNIST data set.

B. Regression and classification methods

1. OLS regression

OLS regression is a linear regression method for which the cost function is given by:

$$C\left(X,\theta\right)=\frac{1}{n}\left\{\left(\boldsymbol{y}-X\theta\right)^{T}\left(\boldsymbol{y}-X\theta\right)\right\}. \tag{2}$$

Here, X is the design matrix based on the input values from the data set, θ is a vector containing the optimization parameters, and $\boldsymbol{y}$ is a vector of the outputs from the data set. The analytical expression for the optimal parameters θ is:

$$\hat{\theta}_{OLS}=\left(X^{T}X\right)^{-1}X^{T}\boldsymbol{y}. \tag{3}$$

2. Logistic regression

Logistic regression is commonly used to classify data with two possible categories. Here we assume that we can approximate a function which gives the likelihood of an event. A cost function from [5] for this is

$$C\left(\theta\right)=-\sum_{i=1}^{n}\left(y_{i}\left(\theta_{0}+\theta_{1}x_{i}\right)-\log\left(1+\exp\left(\theta_{0}+\theta_{1}x_{i}\right)\right)\right). \tag{4}$$

There is no analytical solution for the best θ-parameters which is why we have to solve this numerically.

C. Gradient Descent

Gradient descent (GD) methods are often used in optimization problems, using the negative gradient of a function as a compass to find its minimum [6]. In the context of machine learning, it plays a central role in the iterative search for the optimal parameters that minimize the cost function. The simplest GD procedure goes as follows:

$$w_{i+1}=w_{i}-\eta\Delta C(w_{i}), \tag{5}$$

with w representing the parameter of interest, η the learning rate, and $C(w)$ the cost function. In this case, the learning rate is taken to be a constant value and represents the step taken towards the minimum at each iteration.

Nevertheless, the convergence and running time of the optimization are very dependent on the choice of learning rate. Therefore, some modifications to the GD method have been proposed to mitigate this problem.

1. RMSProp method

The RMSProp method uses the exponential decaying average of previous gradient evaluations to update the learning rate [6]. This average and the current gradient evaluation are further modified by a hyperparameter ρ. The procedure can be represented by:

$$r_{i}=\rho r_{i-1}+(1-\rho)[g(w_{i})\circ g(w_{i})], \tag{6}$$

with $g(w_i)\circ g(w_i)$ corresponding to the element-wise square of the gradient vector. The parameters are then modified as follows:

$$w_{i+1}=w_{i}-\frac{\eta}{\sqrt{r_{i}}+\epsilon}g(w_{i}). \tag{7}$$

A small constant ϵ is used to ensure a non-zero denominator.

2. ADAM method

Another popular method for updating the learning rate is ADAM [6]. It modifies the learning rate based on two exponential moving averages of the gradients, one corresponding to the first-order moment of the gradient:

$$m_{i}=\beta_{1}m_{i-1}+(1-\beta_{1})g(w_{i}), \tag{8}$$

and another for the second-order:

$$v_{i}=\beta_{2}v_{i-1}+(1-\beta_{2})[g(w_{i})\circ g(w_{i})]. \tag{9}$$

In addition, these averages are bias-corrected in order to prevent initialization bias:

$$\hat{m}_{i}=\frac{m_{i}}{1-\beta_{1}^{i}}, \tag{10}$$

$$\hat{v}_i = \frac{v_i}{1-\beta_2^i}. \tag{11}$$

Finally, the parameters are updated as follows:

$$\theta_{i+1} = \theta_i - \frac{\eta}{\hat{v}_i + \epsilon}\hat{m}_i \tag{12}$$

3. Stochastic Gradient Descent

Another possible bottleneck during optimization procedures is the calculation of the gradients at every step. Instead, Stochastic Gradient descent only evaluates the gradient for random samples of the original data set [6]. The size of the samples, often referred to as mini-batches, becomes a hyperparameter of the model, and so does the number of iterations, or epochs, performed over them.

D. Feed forward neural networks

The overarching architecture of our feed forward Neural Network (FFNN) consists of input layer, at least one hidden layer and an output layer. Each layer applies an activation function to its input and the output is fed as input into the next layer. Before the activation function is applied, each node modifies its input as:

$$z_j^l = \sum_{i=1}^{D} w_{ji}^l a_i^{l-1} + b_j^l\,, \tag{13}$$

where a_i^l is the output from the previous layer $(l-1)$, w_{ji} are the weights of layer l and b_j^l are the biases of layer l. Then, z_j^l is fed into the non-linear activation as follows:

$$a_j = f(z_j)\,. \tag{14}$$

The activation function of the last layer depends on the type of problem. For regression analysis, we use the identity function. For classification problems, one solution is for the last activation function to return the probability according to the model of which category each data point belongs to. The training of this network consists of obtaining the parameters (weights and biases) that minimize a chosen cost function.

1. Activation functions

Activation functions are how we introduce nonlinearity into neural networks to fulfill the universal approximation theorem [5]. Each hidden layer uses an activation function and the results from each node are combined when fed into the next layer in the network.

a. Sigmoid The sigmoid function

$$f(x) = \sigma(z) = \frac{1}{1+\exp(-z)}, \tag{15}$$

is a continuous function with output between 0 and 1. This function behaves similarly to a step function, but one can take the derivative of this function for all values of z which is

$$\frac{\partial\sigma(z)}{\partial z} = \sigma\left(z\right)\left(1-\sigma\left(z\right)\right). \tag{16}$$

b. RELU The rectified linear unit (RELU) function

$$\mathrm{RELU}\left(x\right) = \begin{cases} x & \text{if } x > 0 \\ 0 & \text{if } x \leq 0\,, \end{cases} \tag{17}$$

is a different type of activation function. For any input value below zero it returns zero, otherwise it returns the input value. The derivative of this activation function is

$$\frac{\partial\,\mathrm{RELU}}{\partial x}\left(x\right) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{if } x < 0\,. \end{cases} \tag{18}$$

c. Leaky RELU The leaky RELU function

$$\mathrm{RELU}_{\mathrm{leaky}}\left(x\right) = \begin{cases} x & \text{if } x > 0 \\ \alpha x & \text{if } x \leq 0\,, \end{cases} \tag{19}$$

is a variant of the RELU function. Here, a parameter α is added. It is normally set to $0.01-0.3$. This avoids the derivative being zero for inactive neurons. The derivative of the leaky RELU is

$$\frac{\partial\,\mathrm{RELU}_{\mathrm{leaky}}}{\partial x}\left(x\right) = \begin{cases} 1 & \text{if } x > 0 \\ \alpha & \text{if } x < 0\,. \end{cases} \tag{20}$$

d. Identity The identity function returns the value which it gets as an argument. This is one of the simplest activation functions.

e. Softmax Softmax is a generalization of a logistic function. Softmax is given by

$$f\left(z_i\right) = \frac{\exp\left(z_i\right)}{\sum_{m=1}^{K}\exp\left(z_m\right)}. \tag{21}$$

The softmax function applies the exponential function to every value of the output layer and by dividing by the sum of the exponentials of the values the output of softmax is normalized. The derivative of the softmax is given by

$$\frac{\partial f\left(z_i\right)}{\partial z_i} = f\left(z_i\right)\left(\delta_i j - f\left(z_j\right)\right). \tag{22}$$

2. Backpropagation

The backpropagation algorithm consists of a fast method used to compute the gradients of the cost function with respect to the parameters of the network (weights and biases) [7]. Through the chain rule, we have that the derivative of the cost function with respect to the weights has a general form of:

$$\frac{\partial \mathcal{C}}{\partial w_{jk}^l} = \frac{\partial \mathcal{C}}{\partial a_j^l}\frac{\partial a_j^l}{\partial z_j^l}\frac{\partial z_j^l}{\partial w_{jk}^l} = \delta_j^l a_k^{l-1}, \tag{23}$$

and the derivative with respect to the biases is:

$$\frac{\partial \mathcal{C}}{\partial b_j^l} = \frac{\partial \mathcal{C}}{\partial a_j^l}\frac{\partial a_j^l}{\partial z_j^l}\frac{\partial z_j^l}{\partial b_j^l} = \delta_j^l, \tag{24}$$

with l representing the layers and j the neurons in that layer. For the output layer ($l = L$), δ_j is given by:

$$\delta_j^L = \frac{\partial \mathcal{C}}{\partial a_j^L}\sigma'(z_j^L), \tag{25}$$

and, for the hidden layers:

$$\delta_j^l = w^{l+1}\delta^{l+1}\sigma'(z_j^l). \tag{26}$$

3. Cost functions

a. MSE Mean squared error (MSE) is given by

$$MSE = \frac{1}{n}\sum_{i=0}^{n-1}\left(y_i - \tilde{y}_i\right)^2. \tag{27}$$

MSE calculates the squared difference between two sets of data points ($\boldsymbol{y}$ and $\tilde{\boldsymbol{y}}$), sums these squares and divides the sum by the number of elements. This gives the mean of the squared difference between two sets.

b. Cross entropy Cross entropy is a generalisation of the cost function for logistic regression (Equation 4).

4. Regularization

Regularization is often used to prevent overfitting the model, by penalizing high values for the weights [7]. Here, we will describe two of the most common regularization methods.

a. L1 The L1 regularization consists of the addition of a penalty term to the cost function, which is the norm-1 vector of the weights. The new cost function becomes:

$$C_{L1} = C_{original} + \frac{\lambda}{2n}\sum_w |w|. \tag{28}$$

Here, the regularized cost function (C_{L2}) is a function of the unregularized one ($C_{original}$), a hyperparameter (λ), the size of training set (n) and the weights (w).

b. L2 In the case of the L2 regularization, the penalty term is the sum of the squared weights. We can then rewrite the cost function as:

$$C_{L2} = C_{original} + \frac{\lambda}{2n}\sum_w w^2. \tag{29}$$

E. Evaluation of the models

For regression cases, it is common to use both the MSE, previously described as a cost function, and the R^2 metrics to evaluate the quality of the fit. Instead, for classification tasks, the accuracy is a good measure of performance.

1. R^2 metrics

The R^2 metrics is defined as:

$$R^2 = 1 - \frac{\sum_{i=1}^n (y_i - \tilde{y}_i)^2}{\sum_{i=1}^n (y_i - \bar{y})^2}, \tag{30}$$

with y representing the observed values, $\tilde{y}$ the predicted values and $\bar{y}$ the mean of the observed values. The closest it is to 1, the better the fit.

2. Accuracy

The accuracy is calculated by:

$$\text{Accuracy} = \frac{\sum_{i=0}^{n-1} I(y_i = \tilde{y}_i)}{n}. \tag{31}$$

A value of 100% indicates that all samples were properly classified.

F. Implementation

For the regression case, the data set consisted of 1000 points uniformly distributed between -1 and 1. Random noise was introduced by adding a term to the Runge's function exact value taken randomly from the normal distribution. This term was scaled by 0.1 so that meaningful results were obtained. For the classification case, the MNIST and Fashion MNIST data sets were obtained from the `scikit-learn`'s `datasets.fetch_openml` function.

Then, the data was split into training and test sets (80:20) with scikitlearn's `train_test_split` and scaled based on the mean value of the input training set with scikitlearn's `StandardScaler`. Unlike for the OLS model, the design matrix for NN does not have to reflect a polynomial approximation. Instead, by providing the x values as input, the network is able to represent more complex functions.

Before training, the weights of the model were initialized with a normal distribution and, in order to avoid inactive neurons, the biases were taken to be equal to 0.01. We tested different activation functions for the hidden layers (Sigmoid, ReLU and Leaky ReLU), as well as the number of layers and the number of nodes. For the output layer, the activation functions used were the following: Identity function for the regression case and the softmax for the classification ones. We also tested the inclusion of L1 or L2 regularization.

The initialization of the neural network class developed was done as in the example bellow:

```
NN = NeuralNetwork(
        x_train_s,              # Data points
        targets,                # Target values
        layer_output_sizes,     # List of number of nodes for
            hidden + output layers
        activation_funcs,       # List of activation functions
            for hidden + output layers
        activation_ders,        # List of derivatives of
            activation functions for hidden + output layers
        MSE,                    # Cost
        mse_der,                # Derivative of the Cost
            function
        L1=True,                # If set to true, L1
            normalization is used
        L2=False,               # If set to true, L2
            normalization is used
        lmbda = 0.01,           # Hyperparameter from
            regularization
    )
```

The training of the network was performed by calling the functions `train_network_plain_gd` or `train_network_stochastic_gd`. After that, the predictions with the final model was done by the function `predict`. An example of it is presented below:

```
NN.train_network_plain_gd(
        learning_rate=0.01,
        max_iter=100000,
        stopping_criteria=1e-10,
        lr_method='ADAM',
        delta=1e-8,
        rho=0.9,
        beta1=0.9,
        beta2=0.999,
    )
predictions = NN.predict(x_test_s)
```

G. Use of AI tools

GPT UiO was used to obtain information on how to use some functionalities of Latex. The exported conversation can be acessed in the Github repository of the project (m-nader/FYS-STK4155-group14), inside a folder called LLM. In addition, Copilot and GPT UiO were used to debug code.

III. RESULTS AND DISCUSSION

The neural network code developed was applied to a regression and two classification problems. The regression case was the approximation of Runge's function. While the classification cases relied on the MNIST and the Fashion MNIST datasets.

A. Neural network applied to Runge's Function

To fit Runge's function, we employed a neural network composed of 1 or 2 hidden layers, made of 50 or 100 nodes each. First, we used the Sigmoid activation function for the hidden layers and the identity for the output layer. MSE was used as a cost function, without any regularization terms. The optimization algorithm employed was plain gradient descent, with a constant learning rate equal to 0.01. It is valid to note that a smaller learning rate led to convergence problems. The comparison of the MSE and the R^2 between our NN model and the OLS model (with a polynomial of degree 10) is shown in figure 3.

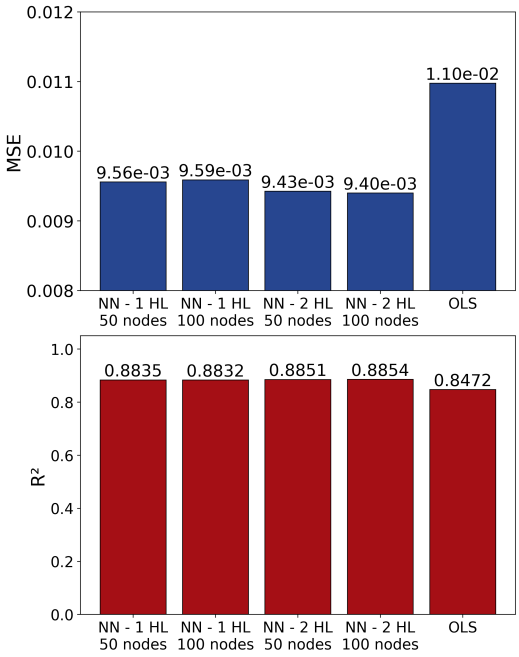

Figure 3. Error analysis performed on the test set of NN models and a OLS model applied to Runge's equation. The four NN models correspond to 1 or 2 hidden layers (HL), with either 50 or 100 nodes per layer. Upper panel shows the MSE and lower panel the R^2 metric.

As expected, because of the ability of a neural network to introduce nonlinearity, the NN model performed better than the simple OLS model. Among the different NN architectures, there was no significant improvement by increasing the number of hidden layers or nodes. Therefore, all the models analyzed from now on have been constructed with only one hidden layer of 50 nodes, also reducing the computational costs. The influence of regularization terms, L1 or L2, and of algorithms for updating the learning rate were investigated and the results are presented in figure 4. The hyperparameter λ for the regularization methods was set to be equal to 0.001.

ADAM resulted in a significant better fit compared to RMSProp, and quite similar to the results obtained using a fixed learning rate (Figure 3). It can also be seen that

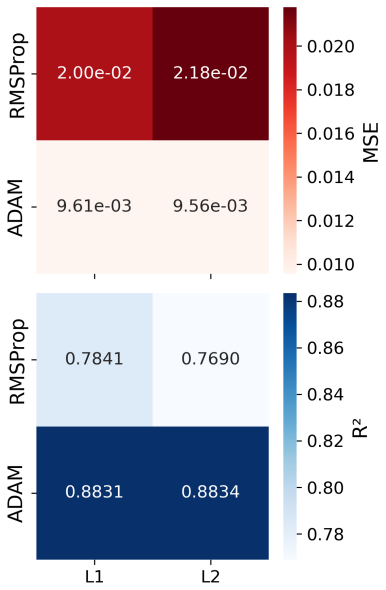

Figure 4. Error analysis performed on the test set of NN models applied to Runge's equation. The four NN models differed by the type of regularization used, either L1 or L2, and the method for updating the learning rate. Upper panel shows the MSE and lower panel the R^2 metric.

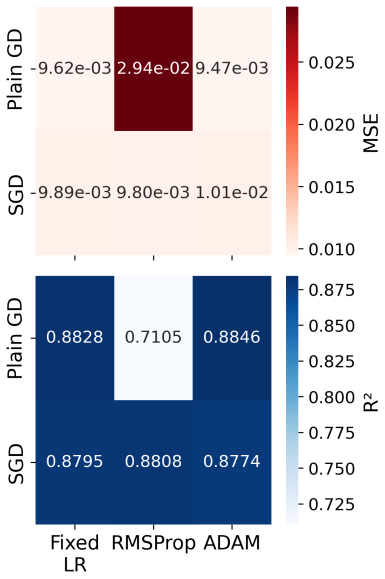

Figure 5. Error analysis performed on the test set of NN models applied to Runge's equation. The six NN models varied the optimization method used, either plain GD or SGD, and the method for updating the learning rate (LR), either a fixed value, RMSProp or ADAM. Upper panel shows the MSE and lower panel the R^2 metric.

there was no great difference between the regularization methods.

Using the L2 regularization, we compared the use of stochastic and plain gradient descent, with a constant learning rate (0.01), and updating it with ADAM and RMSProp. The error analysis is displayed in figure 5. It is important to point out that the use of 0.001 as the hyperparameter λ led to an unfeasible computational cost for the SGD method. Therefore, we opted for a value equal to 0.01 for all the trials evaluated in figure 5.

Apart from plain GD combined with RMSProp, there is no significant improvement between the models. They also perform just as good as the previous trials (Figures 3 and 4). Lastly, we investigated the use of different activation functions for the hidden layers (Figure 6).

The sigmoid activation function led to better results when compared to ReLU and Leaky ReLU, with the last being the best of the two.

B. Neural network applied to MNIST

The first neural network we applied on the MNIST data set we based on what gave good results on approximating Runge's function. Since there are almost infinite different choices to make when designing a feed FFNN starting with a structure that works gave a good starting point to improve upon.

From initial exploration, a wider neural network performed better on classifying the MNIST data (Figure 7). With 50, 100 and 200 nodes per hidden layer, the use of two hidden layers performed worse than one hidden layer. With 300 nodes, both one and two hidden layers behaved similarly, with one hidden layer still outperforming two. This was with the leaky RELU as the activation function for the hidden layer(s) and softmax as the final activation function.

Testing out different λ values with L1 and L2 regularization shows L2 with $\lambda = 1 \cdot 10^{-4}$ performed best of the tested regularization parameters. We used the neural network consisting of one hidden layer with 200 nodes. Even though 300 nodes performed better, we used 200 nodes to save on computation time. This gave an accuracy of $92,6$ %, which was better than the same model with no regularization which gave an accuracy of $91,6$ %.

Using the best regularization, we compared the performance of different gradient descent methods, plain gradient descent and stochastic gradient descent with and without the learning rate methods ADAM and RMSProp. As Figure 8 shows, plain GD with RMSProp performed best. Stochastic GD outperformed plain GD with no learning update of the learning rate and ADAM. In general, stochastic GD was faster and was therefore the first choice when testing out new configurations to see that the model still ran.

The best performing neural network on the MNIST data set was a FFNN with one hidden layer consisting of 200 neurons. This FFNN used plain gradient descent with RMSProp and L2 regularization. The accuracy of this network was 93,4%. The confusion matrix in Figure 9 shows the network's performance per category. It is

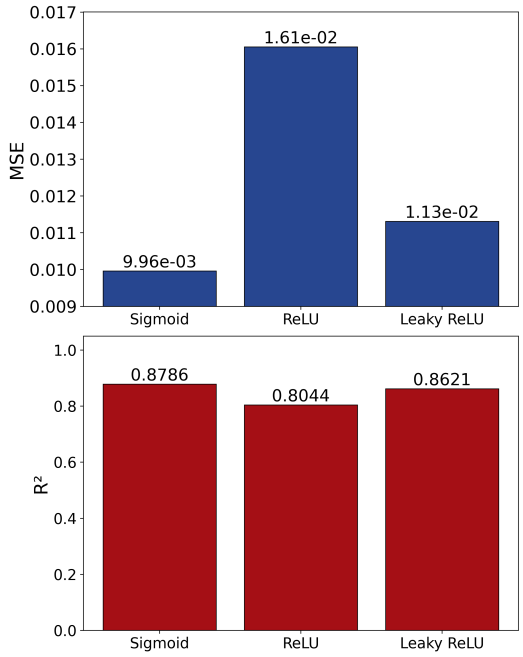

Figure 6. Error analysis performed on the test set of NN models applied to Runge's equation. The four NN models varied the type of activation function used for the hidden layers. Upper panel shows the MSE and lower panel the R^2 metric.

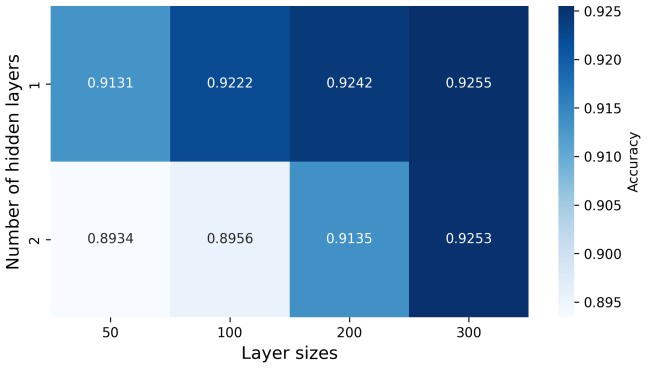

Figure 7. Heatmap of the accuracies of a classification FFNN with stochastic gradient descent using ADAM with a varying number of hidden layers and nodes per layer.

best at categorizing ones and performs worst on identifying eights.

C. Neural network applied to Fashion-MNIST

We applied the best performing model from MNIST on the Fashion MNIST data set. This gave an accuracy of 81,4 %. Not as good as the best accuracy on MNIST. The confusion matrix in figure 10 shows how the correctly and incorrectly labels the images.

An interesting result here is that the model mostly

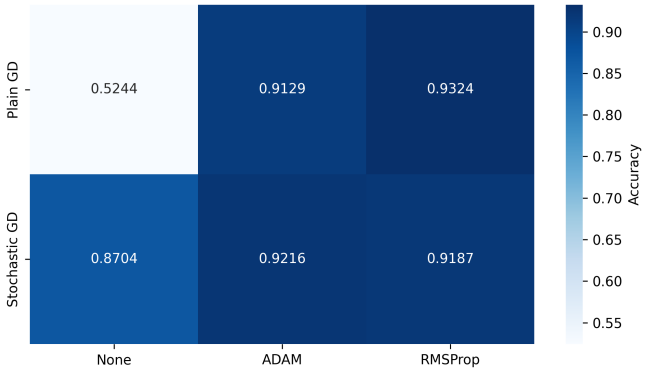

Figure 8. Heatmap showing the performance of the FFNN with different learning methods.

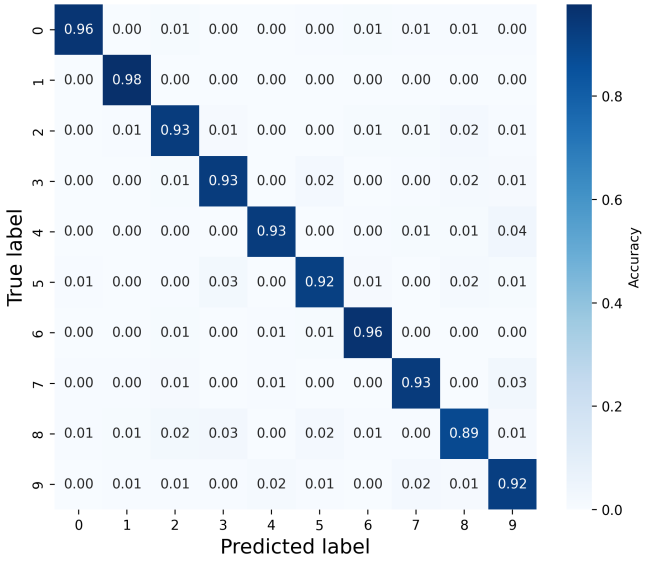

Figure 9. Confusion matrix for the MNIST data. A FFNN with 200 nodes in one hidden layer with plain gradient descent and RMSProp.

mixes up clothing on the upper body with other types of clothing on the upper body. It also mislabels footwear as different footwear. While the miscellaneous like trousers and bags do not seem to have patterns in how they are mislabeled.

IV. CONCLUSION

To showcase the versatility of a feed forward neural network, we tested its application to both regression and classification problems. The regression case was approximating Runge's function. For that, we found that the neural network models performed better than a simple OLS fit which our previous work had shown to be a good regression model.

From the different architectures tested, a neural net-

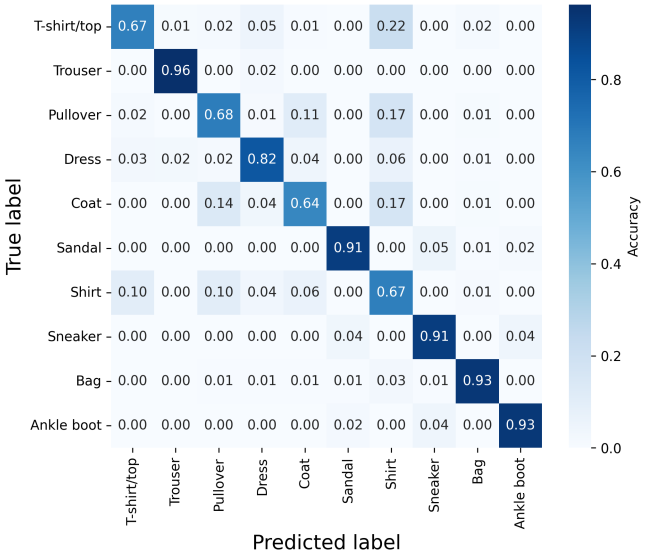

Figure 10. Confusion matrix of Fashion MNIST data.

work composed of 2 hidden layers of 100 nodes each, optimized with plain gradient descent and a constant learning rate of 0.01 performed the best obtaining a R^2 equal to 0.8854. However, most of the models analyzed performed similarly well.

We also applied our implementation of a neural network to two classification problems, using the MNIST and the Fashion MNIST datasets. Although plain GD with RMSProp performed best on the MNIST data, stochastic gradient descent was more efficient based on time spent training the neural network in the classification case. Luckily the time difference with the computation power we had access to was a difference in minutes.

Based on our results, we recommend using stochastic gradient descent, especially on larger data sets, to find a general architecture which performs well before fine tuning based on gradient descent with a learning method like RMSProp. Even with the leaps in computational power these last years, most people do not have access to super computers.

[1] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction, Second Edition. Springer Series in Statistics* (Springer, New York, 2009).

[2] Runge's phenomenon, Runge's phenomenon — Wikipedia, the free encyclopedia (2025), [Online; accessed 06-November-2025].

[3] L. Deng, The mnist database of handwritten digit images for machine learning research, IEEE Signal Processing Magazine **29**, 141 (2012).

[4] H. Xiao, K. Rasul, and R. Vollgraf, Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, arXiv preprint arXiv:1708.07747 (2017).

[5] M. Hjorth-Jensen, Applied data analysis and machine learning (2023).

[6] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning* (MIT Press, 2016) `http://www.deeplearningbook.org`.

[7] M. A. Nielsen, *Neural networks and deep learning*, Vol. 25 (Determination press San Francisco, CA, USA, 2015).