FYS-STK3155 Project 2

Lasse Pladsen, Parham Qanbari, & Sander V. Vattøy

November 10, 2023

Abstract

...

I. INTRODUCTION

Neural networks have become a big deal in the world of machine learning and data predictions. They are a computer's way of mimicking how our brains work. In this project, we're digging into feed-forward neural networks, a popular type, and specifically, how we can make them work better using the gradient descent method and back-propagation algorithm.

Gradient descent, also called steepest descent, is a popular method for tweaking the inner parameters of neural networks. Back-propagation is a way to fine-tune weights in the network by minimizing errors between what the model predicts and what actually happens. Using these methods we allow the network to essentially learn to fit itself onto a data set.

Our main goal is to see how changing input parameters like learning rates, network structure, activation functions, affects how well feed-forward neural networks perform. By playing around with these, we hope to conclude the best methods and parameters for certain different problems.

This report is split into six major sections. The first section (A. Gradient descent regression) focuses on basic regression and comparing methods for gradient descent.

In the second section (B. Neural network regression) we write our neural network and perform linear regression.

The third section (C. Comparing activation functions) we focus on tweaking some of the parameters of our network's regression from the previous section, mostly on whats called the activation function, but this we will come back to in the theory part of the report.

The fourth section (D. Neural network classification) focuses on using our network for binary classification purposes like predicting breast cancer. Binary in the sense that we have two output categories, zero being a benign cancer and one being a malignant cancer.

In the fifth section (E. Logistic regression) we compare the performance from the fourth section (classification) to plain logistic regression.

The final section (F. Evaluation of the various algorithms) is about summarizing all techniques used in the report and performing an evaluation on them.

II. THEORY

A. Design matrix

B. Design matrix for linear regression

We create a so-called design matrix $\mathbf{X}$ for the regression methods from the input data vector $\mathbf{x} \in \mathbb{R}^{\mathbf{n}}$. Each row in $\mathbf{X}$ represents polynomial variables from one data sample. We choose a max polynomial degree p with n data samples such that $\mathbf{X} \in \mathbb{R}^{n\times(p+1)}$. The design matrix is given as the following (Hjorth-Jensen, 2023)

$$\mathbf{X} = \begin{bmatrix} 1 & x_1 & x_1^2 & \dots & x_1^p \\ 1 & x_2 & x_2^2 & \dots & x_2^p \\ \ddots & \ddots & \ddots & \ddots & \ddots \\ 1 & x_n & x_n^2 & \dots & x_n^p \end{bmatrix} \tag{1}$$

C. Measuring prediction performance

To calculate the performance of our predictions we use different methods depending on what type of problem we are dealing with.

For classification problems we use the accuracy score defined as

$$\begin{aligned} Accuracy &= \frac{\text{Number of correct predictions}}{\text{Total number of predictions}} \\ &= \frac{\sum_{i=1}^{n} I(t_i = y_i)}{n} \end{aligned} \tag{2}$$

where I is the indicator function that simply checks if they are equal:

$$I(t = y) = \begin{cases} 1, & t = y \\ 0, & t \neq y \end{cases} \tag{3}$$

For regression problems we use a so-called cost function and the coefficient of determination. A cost function is a function that defines how we measure error and it is what one wishes to minimize, because it directly relates to a more precise model.

The coefficient of determination, denoted R^2, is a measure that tells us how well our models can predict new data and is defined as

$$R^2(\mathbf{z}, \tilde{\mathbf{z}}_i) = 1 - \frac{\sum_{i=0}^{n-1}(z_i - \tilde{z}_i)^2}{\sum_{i=0}^{n-1}(z_i - \bar{z})^2} \tag{4}$$

C1. Ordinary least squares

Ordinary least squares (OLS) is a linear regression method that uses the mean squared error/difference (MSE) between a true value and a prediction as the cost function C. The MSE cost function is expressed as

$$MSE = C(\boldsymbol{\theta}) = \frac{1}{n}\sum_i \left(t_i - y_i(\boldsymbol{\theta})\right)^2 \tag{5}$$

where n is the number of data points, $\boldsymbol{t}$ is the target vector and $\mathbf{y}(\boldsymbol{\theta}) = \boldsymbol{X\theta}$ is the prediction vector. Here $\boldsymbol{X}$ is the design matrix and the parameter $\boldsymbol{\theta} \in \mathbb{R}^{p+1}$ represents the vector of coefficients and the intercept term that the model learns through training, defining the relationship between the input features and the predicted output.

C2. Ridge

Ridge is another linear regression method that adds a shrinkage term called an L2 regularization to the MSE from OLS' cost function:

$$C(\boldsymbol{\theta}) = \frac{1}{n}\sum_i \left(t_i - y_i(\boldsymbol{\theta})\right)^2 + \lambda \sum_i \theta_i^2 \tag{6}$$

where λ is the regularization/shrinkage parameter.

C3. Cross entropy/logistic regression

Cross entropy is a logistic regression method also used for classification problems. This method uses the Sigmoid function $\sigma(x)$ (eq. 23) to make a prediction by classifying some input as a probability (between 0 and 1). The cross entropy for binary classification is defined as

$$C(\boldsymbol{\theta}) = -\sum_i \left[t_i \log(y_i) + (1 - t_i)\log(1 - y_i)\right] \tag{7}$$

D. Gradient descent

Gradient descent is a way to minimize a function. By using the gradient one can move towards and potentially locate a minima of the function. All these coming expressions are also described in Goodfellow et al. (2016).

For a cost function $C(\boldsymbol{\theta})$ with $\boldsymbol{\theta} = \begin{pmatrix}\theta_1 & \theta_2 & \dots & \theta_n\end{pmatrix}$ we are interested in finding the optimal $\hat{\boldsymbol{\theta}}$:

$$\hat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}}{\operatorname{argmin}}\, C(\boldsymbol{\theta}) \tag{8}$$

The cost function's gradient is defined as

$$\boldsymbol{\nabla} C(\boldsymbol{\theta}) = \begin{pmatrix}\frac{\partial C(\boldsymbol{\theta})}{\partial \theta_1} & \frac{\partial C(\boldsymbol{\theta})}{\partial \theta_2} & \dots & \frac{\partial C(\boldsymbol{\theta})}{\partial \theta_n}\end{pmatrix} \tag{9}$$

To minimize the cost function the gradient descent technique begins an initial guess $\boldsymbol{\theta}_0$, then in steps we reduce this guess using the gradient $\boldsymbol{\nabla} C(\boldsymbol{\theta})$ by the following algorithm for a general iteration step i:

$$\boldsymbol{\theta}_{i+1} = \boldsymbol{\theta}_i - \eta_i \boldsymbol{\nabla} C(\boldsymbol{\theta_i}) \tag{10}$$

where the term η_i is called the *learning rate* which can either be some chosen constant or scaled over time (iterations) using a chosen scaling method. If the learning rate is sufficiently small the new guess will always be closer to the optimal $\hat{\boldsymbol{\theta}}$ such that $C(\boldsymbol{\theta}_{i+1}) \leq C(\boldsymbol{\theta}_i)$. With enough iterations we should approach the cost functions minima. The minima is not guaranteed to be a global minima as this method could potentially get stuck in a local minima.

D1. Gradient descent with momentum

This extended method of gradient decent uses a momentum/memory term from the previous iteration throughout the computation. This helps the convergence of $\boldsymbol{\theta}$ by remembering the direction that the previous iterations moved in. The algorithm can be described as

$$v_i = \gamma v_{i-1} + \eta_i \boldsymbol{\nabla} C(\boldsymbol{\theta_i}) \tag{11}$$

$$\boldsymbol{\theta}_{i+1} = \boldsymbol{\theta}_i - v_i \tag{12}$$

where $0 \leq \gamma \leq 1$ is a chosen momentum rate.

D2. Stochastic minibatch gradient descent

This method of gradient descent samples a batch with size M of the total dataset and instead calculates the gradient of the sampled batch. If the dataset has N datapoints then the total number of minibatches will be $m = N/M$. For an iteration over a sampled batch we use the phrase iteration, and the total iteration over all minibatches is called an epoch. This method has the potential to speed up the computational time, the convergence time of $\boldsymbol{\theta}$, and potentially provide better results by introducing randomness/stochasticity which lowers the chance of getting stuck at a local minima. In this report we will refer to this minibatch sampling as Stochastic gradient descent (SGD), and can be described as

In the same way as described for simple gradient decent, it is also possible to add a momentum term to SGD (SGDM), and will make the $\boldsymbol{\theta}$ convergence better.

E. Methods for scaling the learning rate

The choice of learning rate is very important to the results and because of this there are many different ways to adapt the learning rate over time/iterations to improve the results.

E1. Adagrad

The Adagrad algorithm scales the learning rate η_i with a moment term $\mathbf{r}$ with the following expressions

initialize $\mathbf{r}_0 = 0$

$$\mathbf{r}_{i+1} = \mathbf{r}_i + \boldsymbol{\nabla} C(\boldsymbol{\theta}_i) \odot \boldsymbol{\nabla} C(\boldsymbol{\theta}_i) \tag{13}$$

$$\eta_{i+1} = \frac{\eta_i}{\delta + \sqrt{\mathbf{r}_{i+1}}} \odot \boldsymbol{\nabla} C(\boldsymbol{\theta}_i) \tag{14}$$

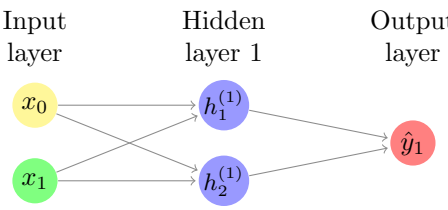

Figure 1: Neural network schematic.

where $\odot$ is the *Hadamard product* which means element-wise multiplication, and δ is a small constant to avoid dividing by zero.

E2. RMSProp

The RMSProp method is similar to Adagrad, but it adds a decay rate ρ to scale the learning rate as such:

initialize $\mathbf{r}_0 = 0$

$$\mathbf{r}_{i+1} = \rho \mathbf{r}_i + (1-\rho)\nabla C(\boldsymbol{\theta}_i) \odot \nabla C(\boldsymbol{\theta}_i) \tag{15}$$

$$\eta_{i+1} = \frac{\eta_i}{\delta + \sqrt{\mathbf{r}_{i+1}}} \odot \nabla C(\boldsymbol{\theta}_i) \tag{16}$$

E3. ADAM

The ADAM method is very similar to RMSProp but it uses an addition moment term $\mathbf{r}, s$ and decay rate ρ_1, ρ_2 and it also uses momentum. It is generally one of the more reliable methods for scaling and can be describes by the following:

initialize $\mathbf{s}_0 = 0$

initialize $\mathbf{r}_0 = 0$

$$\mathbf{s}_{i+1} = \rho_1 \mathbf{s}_i + (1-\rho_1)\nabla C(\boldsymbol{\theta}_i) \tag{17}$$

$$\mathbf{r}_{i+1} = \rho_2 \mathbf{r}_i + (1-\rho_2)\nabla C(\boldsymbol{\theta}_i) \odot \nabla C(\boldsymbol{\theta}_i) \tag{18}$$

$$\hat{\mathbf{s}}_{i+1} = \frac{\mathbf{s}_{i+1}}{1-\rho_1^i} \tag{19}$$

$$\hat{\mathbf{r}}_{i+1} = \frac{\mathbf{r}_{i+1}}{1-\rho_2^i} \tag{20}$$

$$\eta_{i+1} = \eta_i \frac{\hat{\mathbf{s}}_{i+1}}{\delta + \sqrt{\hat{\mathbf{r}}_{i+1}}} \tag{21}$$

F. Feed-Forward Neural Network

We design our Feed-Forward Neural Network (FFNN) algorithm using an input layer, at least one hidden layer and an output layer. In order to train the model we use backpropagation to calculate the parameters giving the most accurate prediction.

The basic design of out NN is demonstrated by the schematic (1). The input layer takes in $\mathbf{X}$ design matrix which is then multiplied by weights $\mathbf{W}_l$ in the general hidden layer l and the bias $\mathbf{b}_l$ in the hidden layer. This result in in the function $\mathbf{z}_l$ (22):

$$\mathbf{z}_l = \mathbf{X}\mathbf{W}_l + \mathbf{b}_l \tag{22}$$

Before moving to the next layer in the network, $\mathbf{z}_l$ needs to be fed into an activation function. It will consider whether nodes should be activated, with the resulting value $a = f(\mathbf{z}_l)$, where it is possible to test a NN for different activation functions.

For the output layer $l = L$, depending on the problem at hand, $\mathbf{z}_L$ could also be fed into an activation layer. For binary classifications problems it's useful to feed $\mathbf{z}_L$ into the an activation function (see F1). However, for regression problems we wish to use the actual calculated value for a prediction, not its activation value. Thus, for regression problems $\mathbf{z}_L$ is not fed into the Sigmoid function.

F1. Activation functions

An activation function is a function we use in a neural network under calculation of the weighted sum of output-weights and biases from the nodes in our layers, to be able to do back-propagation through the feed forward process. Inspired from biological networks (put in reference) the resulting value from the activation $a = f(\mathbf{z}_l)$ represents the activation level of a neuron, or in our case a node. It is a way to quantify the activity in the network in a normalized way. In this project we will consider three activation functions; Sigmoid-, ReLU- and Leaky ReLU function.

The Sigmoid function is defined as

$$f(x) = \frac{1}{1+e^x}, \tag{23}$$

and will take the inputs $\mathbf{z}_l$ in the feed forward process and give a value between 0 and 1:

$$a_l = f(\mathbf{z}_l) \rightarrow 0 \leq a \leq 1 \tag{24}$$

The ReLU (Rectified linear unit) function is defined as

$$f(x) = \begin{cases} x, & x > 0 \\ 0, & x \leq 0 \end{cases} \tag{25}$$

and the Leaky ReLU as

$$f(x) = \begin{cases} x, & x > 0 \\ x \cdot \delta, & x \leq 0 \end{cases} \tag{26}$$

where δ is a small constant - we use the constant 10^{-1}.

In the calculation of the hidden layer errors we also need the derivatives of the activation functions - which are the following equations (27)(28)(29):

The derivative of the Sigmoid function can be calculated to

$$f'(x) = f(x)(1 - f(x)) \tag{27}$$

The derivative of the ReLU:

$$f(x) = \begin{cases} 1, & x > 0 \\ 0, & x \leq 0 \end{cases} \tag{28}$$

and the derivative of Leaky ReLU as

$$f(x) = \begin{cases} 1, & x > 0 \\ \delta, & x \leq 0 \end{cases} \tag{29}$$

F2. Backpropagation

The parameters we want to optimize in the neural network is the weights $\mathbf{W}$ and biases $\mathbf{b}$ in the hidden layers and the output layer. To start the network we use random parameters, then we use a backpropagation algorithm to optimize the parameters. This means we start at the output layer L and propagate backwards through the hidden layers l while updating each of the parameters by using the proceeding layer $l + 1$.

For optimization of the parameters we update the weights and biases with a learning rate multiplied with the proceeding layer's error. The error of the output layer L can be calculated by differentiating the cost function with respect to the output z_L (eq. 30) and for the general hidden layer l we use the proceeding layer's error and weights as mentioned (eq. 31) (Hjorth-Jensen, 2023).

$$\begin{aligned} \boldsymbol{\delta}^L &= \frac{\partial C}{\partial \mathbf{z}} = \frac{\partial \mathbf{a}}{\partial \mathbf{z}} \frac{\partial \mathbf{C}}{\partial \mathbf{a}} \\ &= f'(\mathbf{z}) \frac{\partial \mathbf{C}}{\partial \mathbf{a}} \end{aligned} \tag{30}$$

$$\begin{aligned} \boldsymbol{\delta}^l &= \boldsymbol{\delta}^{l+1} (\mathbf{W}^{\mathbf{l+1}})^T \frac{\partial \mathbf{a}^\mathbf{l}}{\partial \mathbf{z}^\mathbf{l}} \\ &= \boldsymbol{\delta}^{l+1} (\mathbf{W}^{\mathbf{l+1}})^T f'(\mathbf{z}^\mathbf{l}) \end{aligned} \tag{31}$$

In order to find the optimal weights and biases we use the gradient descent method (see section D) to update them iteratively.

The expressions for the gradients for the gradient descent method are as following (Hjorth-Jensen, 2023)

$$\nabla \mathbf{W}^l = (\mathbf{a}^{l-1})^T \boldsymbol{\delta}^l \tag{32}$$

And for the first layer we will just have the design matrix as $\mathbf{a}$:

$$\nabla \mathbf{W}^{l=1} = (\mathbf{X})^T \boldsymbol{\delta}^{l=1} \tag{33}$$

The gradient for updating the bias for a the general layer l is

$$\nabla \mathbf{b}^l = \sum_{i=1}^{n_{inputs}} \boldsymbol{\delta}_i^l \tag{34}$$

We perform this algorithm iteratively until we have optimal parameters based on what we will consider a acceptable convergence threshold.

III. METHODS

A. Gradient descent regression

We begin with an analysis of gradient descent methods. In this analysis we will use both analytical gradients and automatic gradients (using the Python library `Autograd`) for four different gradient descent methods; gradient decent (GD), as well as with momentum (GDM), and stochastic gradient decent (SGD), as well as with momentum (SGDM). We do this analysis for both OLS, and Ridge regression for the chosen second order polynomial

$$f(x) = 4x^2 + 3x + 2 + \epsilon, \tag{35}$$

where ϵ represent the noise we simulate in our model. We also try out the three methods for scaling learning rate; Adagrad, RMSprop and ADAM. Different parameters will need to be tuned to give the best optimal results, for example the learning rate rate η and the shrinkage term λ for Ridge regression. Using the Python library `Seaborn` we can do what is called a grid search where we plot the performance as function of multiple parameters.

In the end we can compare the different regression methods by comparing the MSE with respect to iterations for the different models.

B. Neural network regression

We will now write our Neural Network implementation using the feed-forward and back propagation algorithms in Python. It should be flexible with regards to the network structure, activation and cost functions, and other parameters.

Afterwards we will use our network's regression capabilities on our second order polynomial (eq. 35). Here we use the MSE as cost function (eq. 5), and the sigmoid function (eq. 23) as activation function. We then use our FFNN and train our data, and thereafter test our model against Scikit-learn or Tensorflow. Before proceeding we make an analysis of the regularization parameters and the learning rate used to find the optimal MSE and R2 scores in the calculations.

C. Comparing activation functions

Instead of the sigmoid function, here we will test the two other activation functions for the hidden layers; RELU and the leaky RELU. We can then compare these to the results from the Sigmoid function used earlier.

D. Neural network classification

Next, we will perform a classification analysis using our neural network on the 'Wisconsin Breast Cancer' data set (Wolberg, 1992). The data set contains 30 features, each with $n = 569$ data points. The features represent various characteristics of a cancer sample and our goal is to predict whether the cancer sample based on the data is 'malignant' or 'benign'. The targets are binary classified where 1 is a malignant cancer sample, while 0 is benign.

We try to fit a network model with one hidden layer and compare it with using two hidden layers. For these two version we vary the number of nodes in each hidden layer, and we also compare the three different activation functions. We experiment to find the parameter combinations with the best accuracy score. We also wish to compare our network's performance to the performance of the library `Scikit-learn`. This library implements classification in the `MLPClassifier` class.

E. Logistic regression

For this section we compare the network's classification performance with a pure logistic regression (eq. 7) using the SGD algorithm. Here we also study the results as function of the learning rate η and regularization parameter λ. We also compare the results with `Scikit-learn`'s implementation `LogisticRegression`.

F. Evaluation of the various algorithms

Finally we wish to do a total critical evaluation between all different methods and algorithms summarizing their pros and cons and weighing them against each other.

IV. RESULTS

A. Gradient descent regression

First off we find the optimal parameters for SGD using ADAM and analytical gradient shown in figure 2. Then using these in all the related gradient decent methods we find the MSE with respect to iterations for all instances. We do this for both OLS and Ridge for our function (eq. 35) using analytical gradient shown in figure 3 and automatic gradient shown in figure 8 (appendix). We find that the combination of the optimal parameters is the following;

Learning rate $\eta = 0.3$, hyper parameter $\lambda = 0.001$, number of epochs = 50 and number of minibatches = 20. It is also possible to see that ADAM generally fall off to low MSE already around $100 - 200$ iterations.

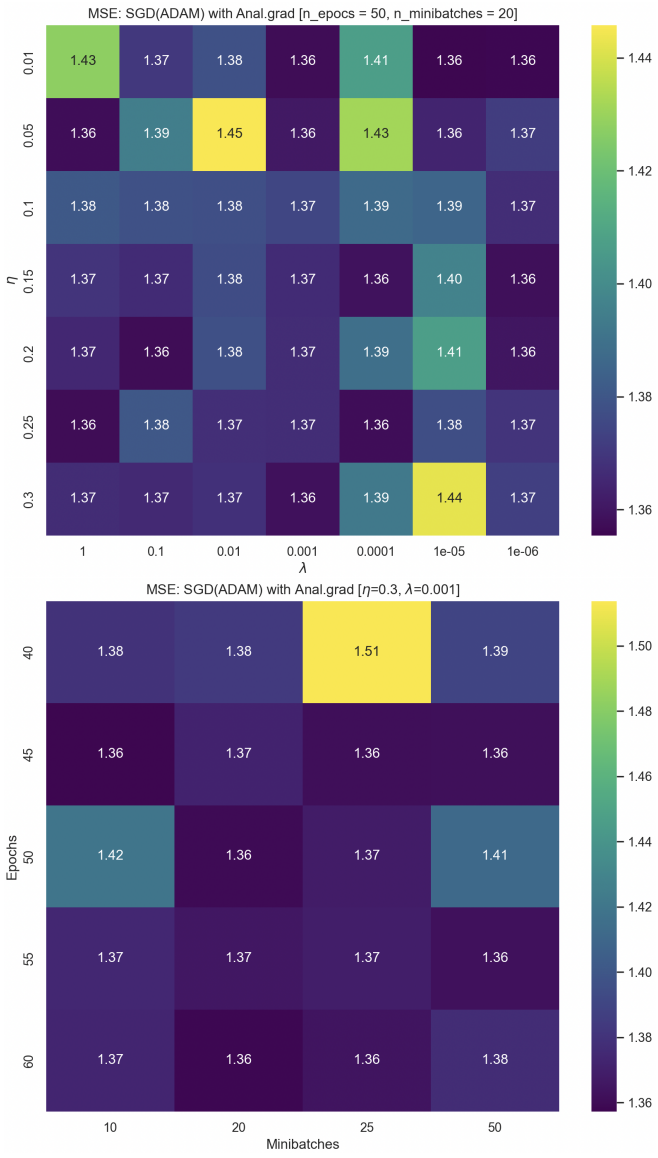

Figure 2: Grid search plot of different parameters using the SGD algorithm with the ADAM scaling method. By first finding optimal η and λ, we thereafter found optimal amount of epochs and minibatches.

B. Neural network regression

After writing our neural network we need to find the optimal parameters to minimize our cost function, and get the best fitting to our second order polynomial. We obtain the result for MSE and R2-score respectively shown

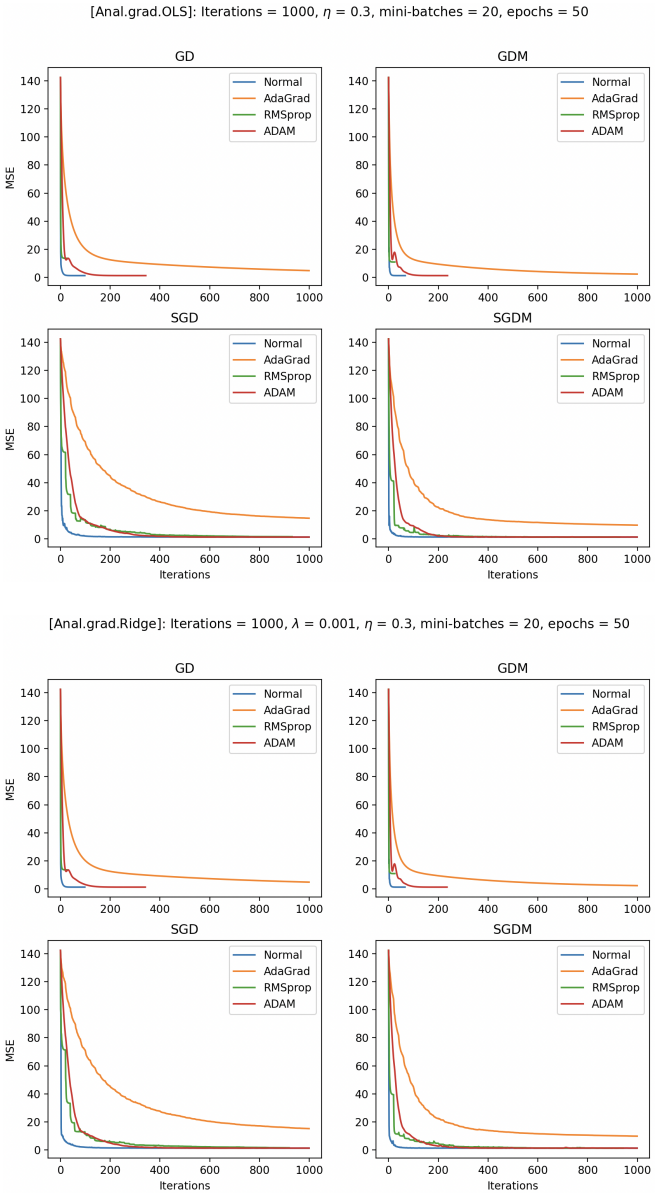

Figure 3: Plot of gradient decent and scaling methods using analytical gradient with optimal hyper parameter λ, learning rate η, and number of epochs and minibatches.

in figure 4 and 5. Here we used between 3 and 15 batches for the SGD but got similar results, so the figures focus on the 15 batches instance. We can see the optimal parameters can be chosen to be the same for the two, and end up obtaining the following optimal parameters: $\lambda = 0.1$ and $\eta = 0.01$.

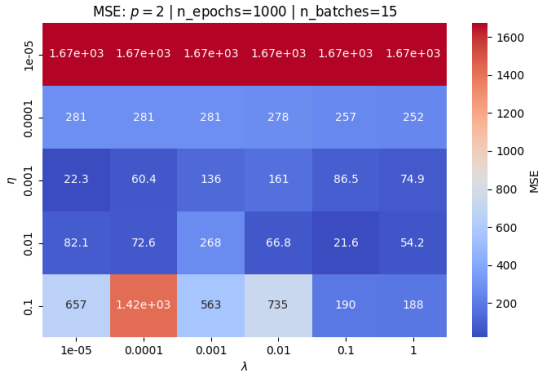

Figure 4: show ...

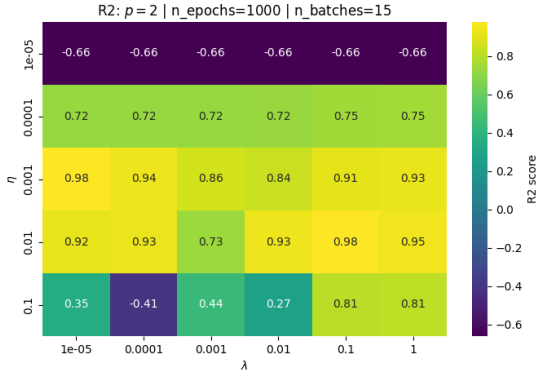

Figure 5: show ...

Using these optimal parameters in our neural network we obtain the polynomial-fitting to our second order polynomial shown in figure 6, while using scikit we obtain the fitting shown in figure 7.

C. Comparing activation functions

...

D. Neural network classification

A selection of prediction accuracies are shown in table (D).

E. Logistic regression

...

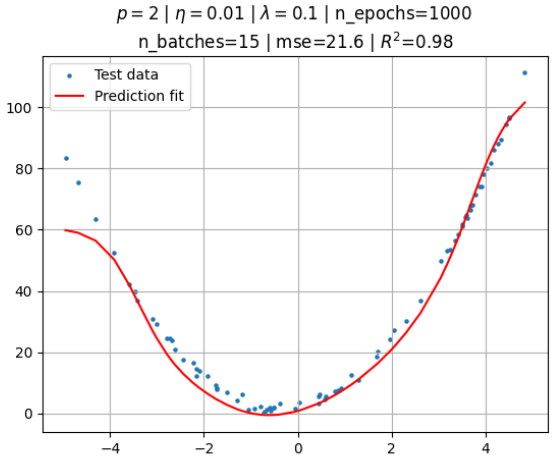

Figure 6: show ...

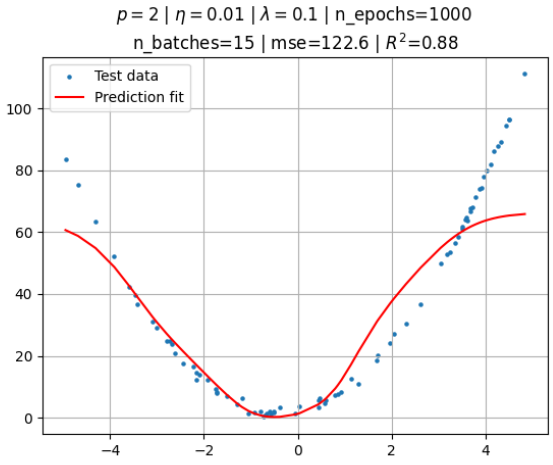

Figure 7: show ...

Model Configuration		Sigmoid	RELU	lRELU
1 Hidden Layer	50 Nodes	0.91	0.44	0.56
	150 Nodes	0.91	0.44	0.56
	200 Nodes	0.91	0.44	0.56
2 Hidden Layers	50 Nodes	0.91	0.39	0.61
	150 Nodes	0.91	0.39	0.61
	200 Nodes	0.93	0.39	0.61

Table 1: The table values show a selection of the prediction accuracies we found for our different model configurations.

F. Evaluation of the various algorithms

...

V. DISCUSSION

A. Gradient decent regression

It is possible to see that there are a lot of combinations of optimal parameters which is possible to choose from using ADAM and SGD. We therefor just choose the above mentioned set of parameters, which we can see resulted in the MSE decrease in figure 3. The plain gradient decent seem to have faster MSE drop-off in our instance, but ADAM is generally the second best, and end up with a very small MSE. We therefor choose to focus on using ADAM in the rest of the report – it is generally a more stable model (as mentioned in the theory section).

Focusing on ADAM, the MSE only converges for GD and GDM, but for all instances we generally end up with a very small MSE (1.36 for SGD as shown in figure 2).

B. Neural Network MSE & R2-score

...

C. Using different activation functions

...

D. Classification analysis of Wisconsin Breast Cancer Data

Our experimentation with the various model configurations seem to show that the sigmoid activation function gave the best prediction accuracies. We found no significant difference between one hidden layer or two hidden layers in prediction accuracies for all three activation functions. Nevertheless, sigmoid activation showed >0.90 prediction accuracy. It is important to note that these results where obtained using a plain gradient descent method (10). Our results are consistent `MLPClassifier` from `sklearn.neural_network`, given that the function argument `solver` is stochastic gradient descent. However, using the RELU activation function with the Adam `solver` in `MLPClassifier` shows a significant increase in prediction accuracy from approximately 0.50 to 0.98.

E. Logistic regression

...

F. Model comparison

...

VI. CONCLUSION

...

Appendix A. Github repository

https://github.com/LassePladsen/FYS-STK3155-projects/tree/main/project2

Appendix B. List of source code

Here is a list of the code we have developed in this project which can be found in the above Github repository:

- ...

Appendix C. Automatic gradient decent

In figure 8 we can see the MSE result we get in gradient decent when using automatic gradient. If you compare it to the analytical gradient results it is possible to see similarities, but the analytical instance more often converge faster with smaller MSE.

Appendix D. Analytical derivations

References

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. The MIT Press, Cambridge, Massachusetts. (http://www.deeplearningbook.org)

Hjorth-Jensen, M. (2023). *Applied data analysis and machine learning*. https://compphysics.github.io/MachineLearning/doc/LectureNotes/_build/html/intro.html. ([Online; accessed 30-September-2023])

Wolberg, W. (1992). *Breast cancer wisconsin*. https://archive.ics.uci.edu/dataset/15/breast+cancer+wisconsin+original. ([Online; accessed 07-November-2023])

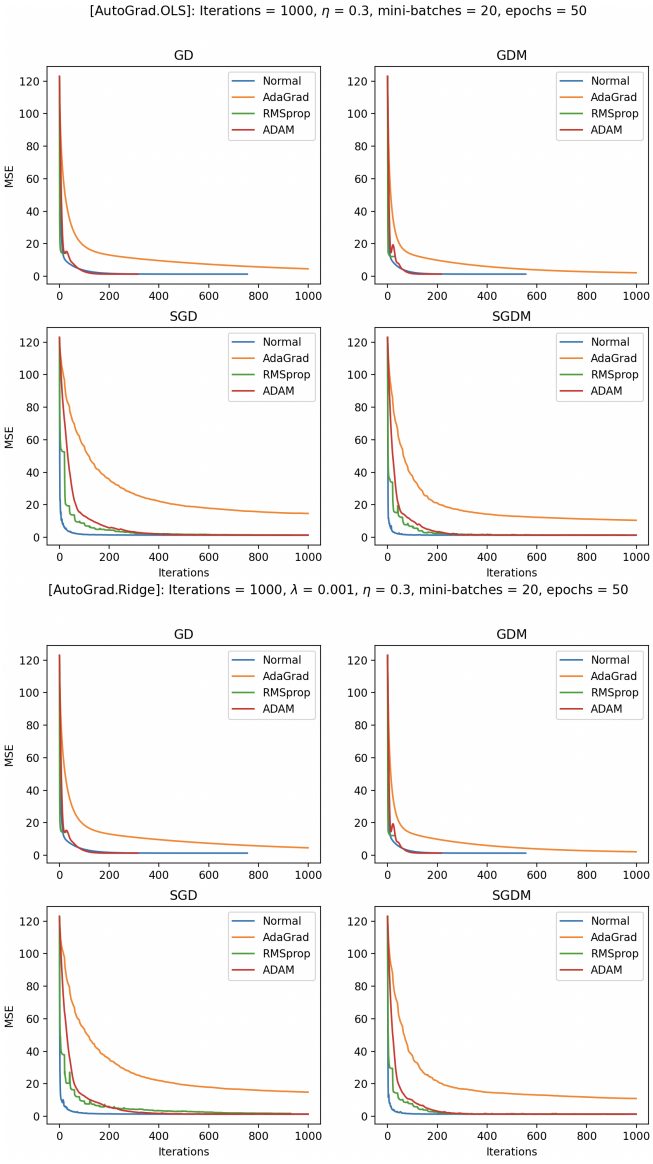

Figure 8: Plot of gradient decent and scaling methods using automatic gradient with optimal hyper parameter λ, learning rate η, and number of epochs and minibatches. We do this mostly to confirm our results for the analytical gradient.