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.

II. THEORY

A. Cost function

A cost function is a function we define to provide an error (cost) estimate.

A1. Mean squared error

[...]

$$MSE = \frac{1}{N}\Sigma_i^n \left(\mathbf{t} - \mathbf{a}\right)^2 \tag{1}$$

[ols, ridge, logreg...]

B. Gradient descent

We will be using the technique known as gradient descent to minimize a chosen cost function. All these algorithms are 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 $\boldsymbol{\theta}$:

$$\hat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}}{\operatorname{argmin}}\, C(\boldsymbol{\theta}) \tag{2}$$

The cost function's gradient is defined as

$$\boldsymbol{\nabla} C(\boldsymbol{\theta}) = \begin{pmatrix}\frac{\partial C(\boldsymbol{\theta})}{\partial \theta_1} & \frac{\partial C(\boldsymbol{\theta})}{\partial \theta_2} & \dots & \frac{\partial C(\boldsymbol{\theta})}{\partial \theta_n}\end{pmatrix} \tag{3}$$

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:

$$\boldsymbol{\theta}_{i+1} = \boldsymbol{\theta}_i - \eta_i \boldsymbol{\nabla} C(\boldsymbol{\theta_i}) \tag{4}$$

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.

B1. 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

$$v_i = \gamma v_{i-1} + \eta_i \boldsymbol{\nabla} C(\boldsymbol{\theta_i}) \tag{5}$$

$$\boldsymbol{\theta}_{i+1} = \boldsymbol{\theta}_i - v_i \tag{6}$$

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

B2. 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.

C. 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.

C1. Adagrad

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

$$\text{initialize } \mathbf{r}_0 = 0$$
$$\mathbf{r}_{i+1} = \mathbf{r}_i + \boldsymbol{\nabla} C(\boldsymbol{\theta}_i) \odot \boldsymbol{\nabla} C(\boldsymbol{\theta}_i) \tag{7}$$
$$\eta_{i+1} = \frac{\eta_i}{\delta + \sqrt{\mathbf{r}_{i+1}}} \odot \boldsymbol{\nabla} C(\boldsymbol{\theta}_i) \tag{8}$$

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

C2. RMSProp

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

$$\text{initialize } \mathbf{r}_0 = 0$$
$$\mathbf{r}_{i+1} = \rho \mathbf{r}_i + (1-\rho) \boldsymbol{\nabla} C(\boldsymbol{\theta}_i) \odot \boldsymbol{\nabla} C(\boldsymbol{\theta}_i) \tag{9}$$
$$\eta_{i+1} = \frac{\eta_i}{\delta + \sqrt{\mathbf{r}_{i+1}}} \odot \boldsymbol{\nabla} C(\boldsymbol{\theta}_i) \tag{10}$$

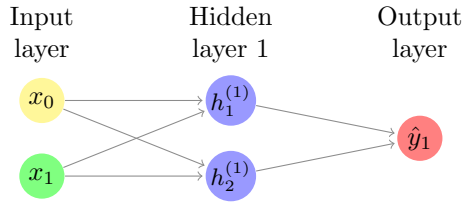
C3. ADAM

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:

$$\text{initialize } \mathbf{s}_0 = 0$$
$$\text{initialize } \mathbf{r}_0 = 0$$
$$\mathbf{s}_{i+1} = \rho_1 \mathbf{s}_i + (1-\rho_1) \boldsymbol{\nabla} C(\boldsymbol{\theta}_i) \tag{11}$$
$$\mathbf{r}_{i+1} = \rho_2 \mathbf{r}_i + (1-\rho_2) \boldsymbol{\nabla} C(\boldsymbol{\theta}_i) \odot \boldsymbol{\nabla} C(\boldsymbol{\theta}_i) \tag{12}$$
$$\hat{\mathbf{s}}_{i+1} = \frac{\mathbf{s}_{i+1}}{1-\rho_1^i} \tag{13}$$
$$\hat{\mathbf{r}}_{i+1} = \frac{\mathbf{r}_{i+1}}{1-\rho_2^i} \tag{14}$$
$$\eta_{i+1} = \eta_i \frac{\hat{\mathbf{s}}_{i+1}}{\delta + \sqrt{\hat{\mathbf{r}}_{i+1}}} \tag{15}$$

D. 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 (D). The input layer takes in $\mathbf{X}$ design matrix which is then multiplied by weights $\mathbf{W}_h$ in the hidden layer and the bias $\mathbf{b}_h$ in the hidden layer. This result in in the function $\mathbf{z}_h$ (17):

$$\mathbf{z}_h = \mathbf{X}\mathbf{W}_h + \mathbf{b}_h \tag{16}$$

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

Going from the hidden layer to the output layer (given one hidden layer as shown in D) the resulting activation value from the previous hidden layer is multiplied by the output weights $\mathbf{W}_o$ and the output bias is added $\mathbf{b}_h$, resulting in $\mathbf{z}_o$ (the weighted sum):

$$\mathbf{z}_o = \mathbf{a}\mathbf{W}_o + \mathbf{b}_o \tag{17}$$

Depending on the problem at hand, $\mathbf{z}_o$ could also be fed into an activation layer. For binary classifications problems it's useful to feed $\mathbf{z}_o$ into the Sigmoid activation function (see D1). However, for regression problems we wish to use the actual calculated value for a prediction, not its activation value. Thus, for regression problems $\mathbf{z}_o$ is not fed into the Sigmoid function.

D1. 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 $a = f(\mathbf{z}_h)$ 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

$$f(x) = \frac{1}{1+e^x}, \tag{18}$$

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

$$a = f(\mathbf{z}_h) \rightarrow 0 \leq a \leq 1 \tag{19}$$

The ReLU (Rectified linear unit) function is defined as

$$f(x) = \begin{cases} x, & x > 0 \\ 0, & x \leq 0 \end{cases} \tag{20}$$

and the Leaky ReLU as

$$f(x) = \begin{cases} x, & x > 0 \\ x \cdot \delta, & x \leq 0 \end{cases} \tag{21}$$

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 (22)(23)(24):

The derivative of the Sigmoid function is defined as

$$f'(x) = f(x)(1 - f(x)) \tag{22}$$

The derivative of the ReLU:

$$f(x) = \begin{cases} 1, & x > 0 \\ 0, & x \leq 0 \end{cases} \tag{23}$$

and the derivative of Leaky ReLU as

$$f(x) = \begin{cases} 1, & x > 0 \\ \delta, & x \leq 0 \end{cases} \tag{24}$$

D2. Backpropagation

Our main parameters 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 parhameters; however, we use a backpropagation algorithm to optimize the parameters. For optimazation of the parameters we update the weights and biases with a learning rate multiplied with a error. The error of the output layer L is defined in general as (25) and for a general hidden layer l we use (26) (Hjorth-Jensen, 2023).

$$\begin{aligned} \boldsymbol{\delta}^L &= \frac{\partial \mathbf{a}}{\partial \mathbf{z}} \frac{\partial \mathbf{C}}{\partial \mathbf{a}} \\ &= f'(\mathbf{z}) \frac{\partial \mathbf{C}}{\partial \mathbf{a}} \end{aligned} \tag{25}$$

$$\begin{aligned} \boldsymbol{\delta}^l &= \boldsymbol{\delta}^{l+1} (\mathbf{W}^{\mathbf{l+1}})^T \frac{\partial \mathbf{a}^{\mathbf{l}}}{\partial \mathbf{z}^{\mathbf{l}}} \\ &= \boldsymbol{\delta}^{l+1} (\mathbf{W}^{\mathbf{l+1}})^T f'(\mathbf{z}^{\mathbf{l}}) \end{aligned} \tag{26}$$

Where $\frac{\partial \mathbf{a}}{\partial \mathbf{z}}$ is the gradient of the activation in the output layer with respect to the $\mathbf{z}$ function, and $f'(\mathbf{z})$ is the derivative of the Sigmoid function. In the hidden layer, the error of the proceeding layer $\boldsymbol{\delta}^{l+1}$ is multiplied with its proceeding layers weight and the the derivative of the Sigmoid function in the hidden layer $f'(\mathbf{z}^{\mathbf{h}})$.

In order to find the optimal weights and biases we use the gradient descent method to iteratively update them as expressed in the general equation (4) and explained in B, where we calculate the gradients from the following expressions (Hjorth-Jensen, 2023):

$$\nabla \mathbf{W}^l = (\mathbf{a}^{l-1})^T \boldsymbol{\delta}^l \tag{27}$$

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

$$\nabla \mathbf{W}^{l=1} = (\mathbf{X})^T \boldsymbol{\delta}^{l=1} \tag{28}$$

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

$$\nabla \mathbf{b}^l = \sum_{i=1}^{n_{inputs}} \boldsymbol{\delta}_i^l \tag{29}$$

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

E. Classification of neural network

This is a type of neural network often used to recognice sounds and photos (as face recognition). It uses a different type of cost function then refered to earlier defined as

[equation] (Parham?)

To measure the preformance we now use the accuracy-score function from `sklearn.metrics`, defined as

$$a = \frac{\sum_{n=1}^{i} I(t_i = y_i)}{n} \tag{30}$$

where I is the indicator function (1 if $t_i = y_i$ and 0 otherwise), and n is the number of t_i. In our case

t_i represent the target and y_i the output from our FFNN.

In this project we will perform a classfication analysis on the 'Wisconsin Breast Cancer' data set (Wolberg, 1992). The data set contains 30 features, each with 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 will feed in the data into our neural network and adjust the parameters of the network based on the accuracy score (30) to attempt to predict whether the cancer sample is malignant or benign based on the features.

III. METHODS

A. Gradient decent MSE

We begin with an analysis of gradient descent methods. In this analysis we will use both analytical gradients and automatic gradients (using the Python `Autograd` package) for four different gradient descent methods; gradient decent (GD), as well as with momentum (GDM), and stocastic gradient decent (SGD), as well as with momentum (SGDM). We do this analyzis for both Ordinary least squares (OLS), and Ridge regression for the chosen function

$$f(x) = 4x^2 + 3x + 2 + \epsilon, \tag{31}$$

where ϵ represent the noise we simulate in our model. Not only do we use plain gradient decent, but also the three methods for scaling learning rate; Adagrad, RMSprop and ADAM. Different parameters will need to be tuned to give the best optimal results, f.eks. λ and η for Ridge regression. Using the Python `Seaborn` package we can easily find these optimal parameters. We choose to focus on using the optimal parameters for SGD using Ridge with analytical gradient since we this method will be focused on later in the report, and is likely to give the best results. In the end we can compare the different regression methods by looking at decrease of mean square error (MSE) with respect to iterations for the different models.

B. Neural Network MSE & R2-score

We will now write a Neural Network (NN) code, using Feed Forward and implementing Back propagation on our second order polynomial (eq. 31). Here we use the MSE as cost function (eq. 1), and the sigmoid function (eq. 18) 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. Using different activation functions

Instead of the sigmoid function, we can now also test the following different 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. Classification analysis using Neural Network

Next, we will perform a classification analysis using our neural network on the 'Wisconsin Breast Cancer' data set (Wolberg, 1992). In this analysis we use the accuracy score (30) to adjust the η and λ parameters and the number of nodes in the network. Furthermore, we try a network with one hidden layer and two hidden layers. For these two version we vary the number of nodes in each hidden layer, and we also try using three different activation functions. More specifically, we experiment with the Sigmoid (18), rectifier linear unit (RELU) (20) and the leaky RELU (lRELU) (21)functions. We experiment with the various inputs and function and try to find the combinations that results in the best accuracy score. We then compare the results obtained from our code with `Scikit-learn`.

E. Logistic regression

Finally we make a Logistic regression function (using our SGD algorithm) and compare it's results to the FFNN code from earlier. We also study the results as function of the learning rate η and regularization parameter λ in the SGD-method. At last we can once again compare this to Scikit-learn (using ...).

F. Model comparison

In the end we can compare our models and look at which ones makes the best predictions with respect to MSE and accuracy score.

IV. RESULTS

A. Gradient decent MSE

First of we find the optimal parameters for SGD using ADAM and analytical gradient shown in figure 1. 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. 31) using analytical gradient shown in figure 2 and automatic gradient shown in figure 3 in the 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.

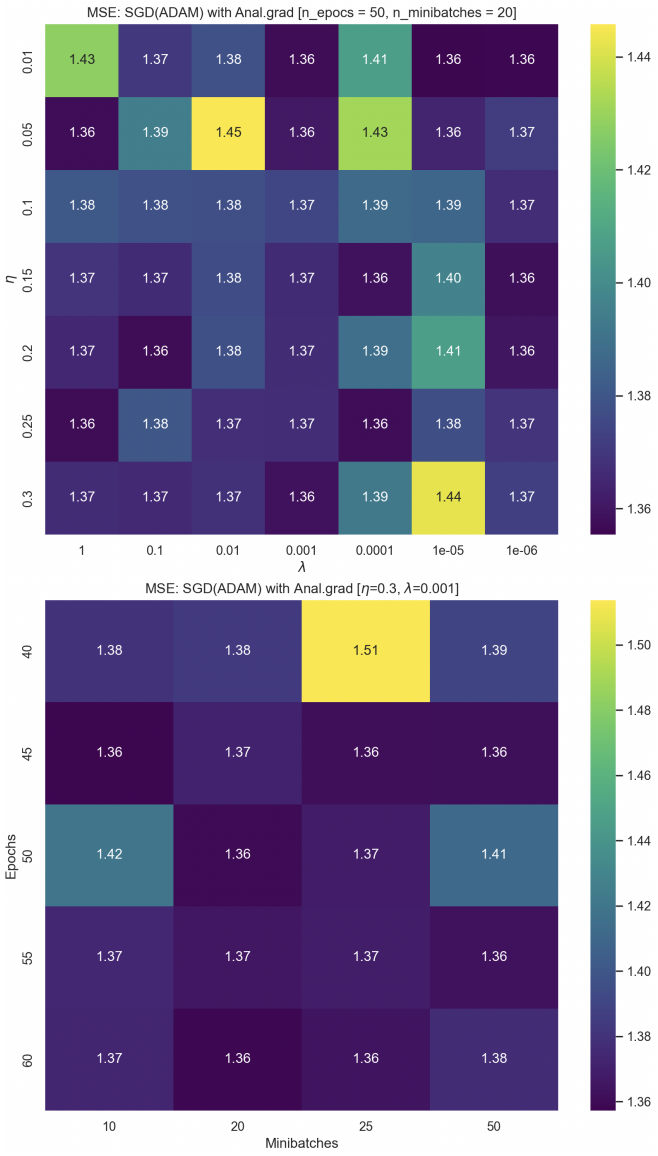


Figure 1: Gridmap of optimal parameters for lowest MSE for SGD using the ADAM scaling method. By first finding optimal η and λ, we thereafter found optimal amount of epochs and minibatches.

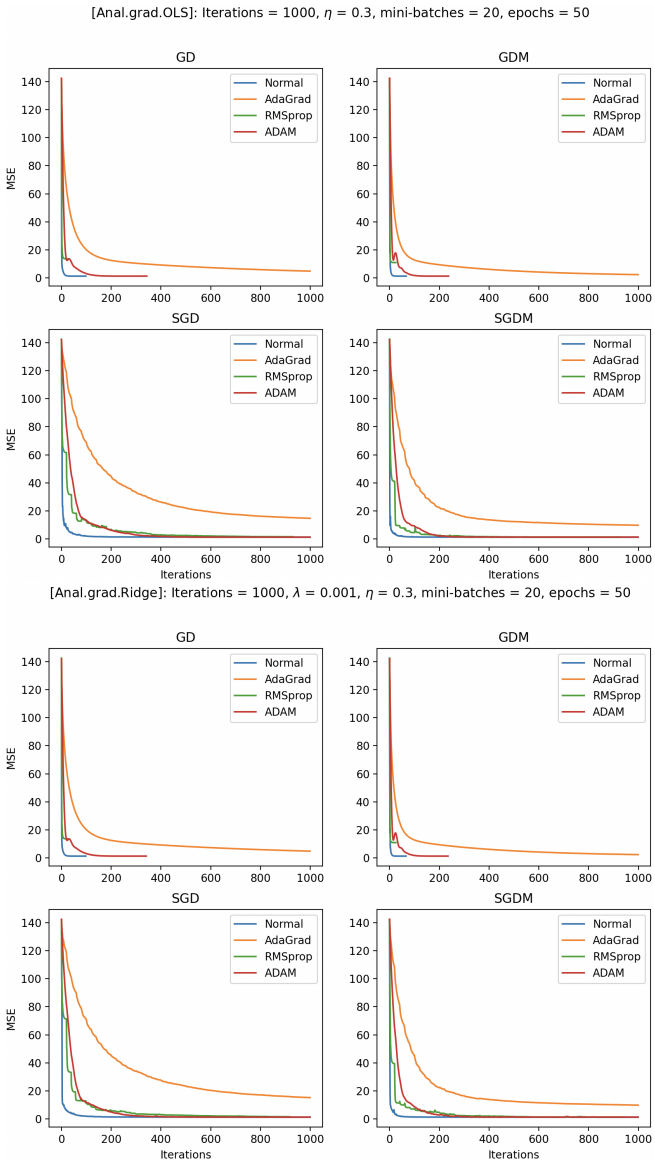


Figure 2: Plot of gradient decent and scaling methods using analytical gradient with optimal hyper parameter λ, learning rate η, and number of epochs and minibatches.

B. Neural Network MSE & R2-score

...

C. Using different activation functions

...

D. Classification analysis using Neural Network

Model Configuration		Sigmoid	RELU	lRELU
1 Hidden Layer	50 Nodes			
	150 Nodes			
	200 Nodes			
2 Hidden Layers	50 Nodes			
	150 Nodes			
	200 Nodes			

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

E. Logistic regression

...

F. Model comparison

...

V. DISCUSSION

A. Gradient decent MSE

...

B. Neural Network MSE & R2-score

...

C. Using different activation functions

...

D. Classification analysis using Neural Network

...

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 3 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.

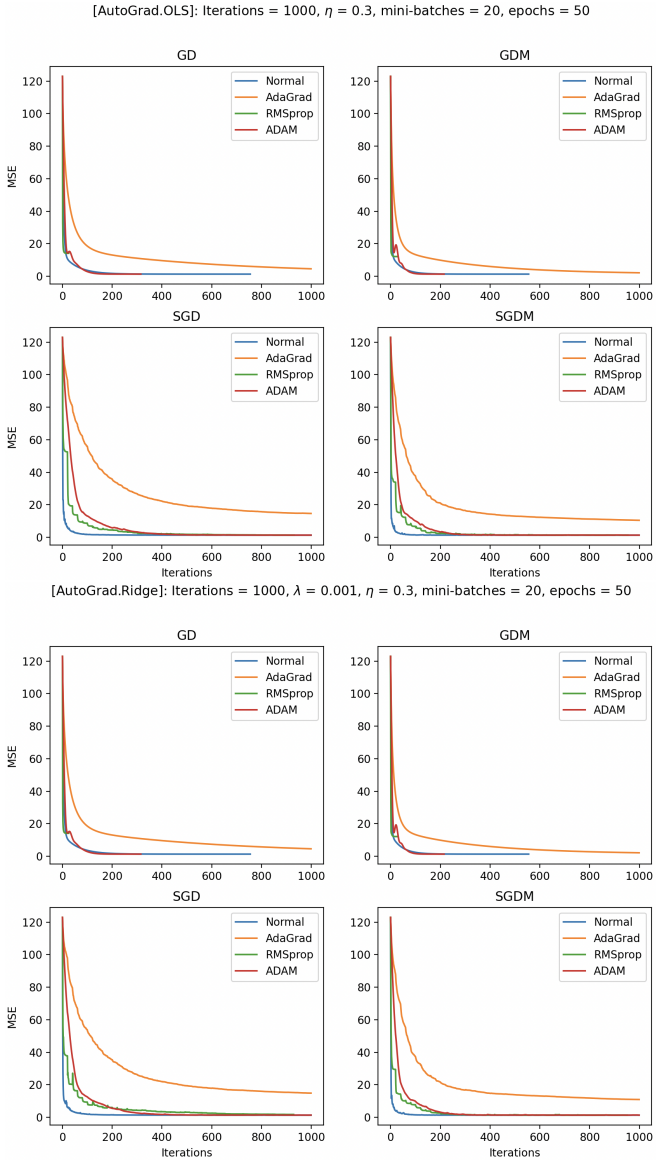


Figure 3: Plot of gradient decent and scaling methods using automatic gradient with optimal hyper parameter λ, learning rate η, and number of epochs and minibatches. We do this mostly to confirm our results for the analytical gradient.

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])