FYS-STK4155 - Project2 TITLE SHOULD BE DESCRIPTIVE

Gard and Are*
Department of Geoscience, University of Oslo

David Andreas Bordvik
Department of Informatics, University of Oslo

(Dated: November 13, 2021)

This will be our wonderfully conclusive abstract

INTRODUCTION

Feed-Forward Neural Networks takes the idea of connecting artificial neurons in layers to create a fully connected network. Information flows in a single direction through the network, from the input layer to the evaluation of the input in the output layer. A Feed-Forward network can be used to approximate a function. The network does this by learning which parameters, given some input, results in the closest representation of the correct output [3].

The **Universal approximation theorem** states that a Neural Network can approximate any function with as few as a single hidden layer to any precision [2, 5]. Hence, any feedforward network with a single hidden layer is in theory ample to represent any function, due to their underlying universality [6]. For this project, we will use the **Universal approximation theorem** as motivation to develop our own Feed-Forward Neural Network code to study both classification and regression.

When studying regression using our developed Feed-Forward Neural Network code, we will return to the terrain data used in Project 1 when studying polynomial fitting with different variants of Least Squares Regression. Moreover, we will compare our findings in this project with the result of our previous project to further study the differences polynomial fitting and gradient based approximations.

Furthermore, we will study classification using the Wisconsin Breast Cancer Data provided by SciKit-learn [8]. When studying the accuracy of the Neural Net in the context of classification, we will also develop code for Logistic Regression for comparison.

The following sections will include background theory and proposals to algorithms which will be needed to construct our Feed-Forward Neural Network. All algorithms developed will be discussed in detail, however for the concrete implementation we refer to the GitHub repository linked in the Appendix under the section Source Code.

For clarity, all source code developed has been written using the Python programming language. Results obtained from our own developed code will also be compared to functions from the Machine Learning libraries SciKit-Learn [8], PyTorch [7] and TensorFlow [1].

In the following section, we will derive the theory and develop an algorithm for Stochastic Gradient Descent. Using that algorithm as a basis, we will further derive theory and develop code for our own Neural Network and Logistic Regression. With the developed algorithms, we will study their behavior and skill when used with both the Terrain data from Project 1 as well as the Breast Cancer data provided by SciKit-learn [8]. The results obtained from our own developed code, as well as the benchmarks with the previously stated Machine Learning software will be presented in the **Results** section. A discussion of our results will be confined to the **Discussion** section. Finally, the project will be concluded with our final thoughts for the results as well as this project as a whole in the **Conclusions** section.

THEORY AND METHODS

Moving along the gradient

Gradient Descent is an iterative algorithm that minimizes a given function by following its gradient down towards the global minimum. For a given cost-function $C(\beta)$ with predictors β and a hyperparameter η (which will be described in the following paragraph), the process of Gradient Descent can be expressed mathematically as follows

$$\beta_{k+1} = \beta_k - \eta \nabla_\beta \left(C(\beta_k) \right) \tag{1}$$

By inspecting Equation (1), it can be seen that given some random initialization of β_0, each step β_k will be closer to the optimal $\hat{\beta}$ than the previous step. This is done by repeatedly calculating the cost function and taking a step in the direction of its gradient. Moreover, the length of the step taken is decided by the newly introduced hyperparameter η. Where consecutively large values of η results in a large step along the gradient, and a small value of η a small step along the gradient. The learning rate is an essential parameter to ensure that the gradient descent converges, as small values might end up slow convergence and large values result in divergence [4].

Another pitfall for gradient based convergence methods is the existence of local minimums. As not all functions

* afkvanum@mail.uio.no

are created equal, some functions might feature some unevenness that the gradient descent will mistake as a global minimum. As such, gradient descent on the form as in Equation (1) will get stuck in any local minima that it encounters. However, given the case of regression, the MSE cost function is a convex function. As such, there is only one minimum of the function which is the global minimum [4]. Thus, when considering regression, there is no risk of the algorithm getting stuck in a local minima.

Introducing stochasticity to gradient descent

By utilizing the entire gradient of the cost function to compute the next step along the gradient, a lot of computational resources are utilized. Especially if one are to consider a large data-set consisting of several data-points and features. A method to alleviate some of the computational demand of the algorithm is to only compute the gradient for a smaller subset of the data. With Stochastic Gradient Descent, this idea is implemented with a stochastic element, that is, which subset of the data is used is selected at random. The resulting gradient algorithm converges towards the same global minimum as regular gradient descent over the entire data-set, though at a higher pace following an uneven gradient path [4].

Another benefit of Stochastic Gradient Descent (SGD) compared to its non-stochastic variant is its ability to exit local minima. As the SGD never reaches a true minimum, a local minima might not be able to contain the algorithm when recomputing the gradient based on a different subset of data. On the other hand, as a consequence of never achieving true minimum, SGD will never return optimal values when compared to non-stochastic gradient descent.

Implementing Stochastic Gradient Descent

By introducing the concept of mini-batches to Gradient Descent, we have to split our data randomly into mini-batches such that the resulting design matrix is

$$X_{\text{perm}} = \{X_0, X_1, \dots, X_M\}^T$$

where M represents the number of mini-batches. Moreover, X_i contains randomly drawn rows from X. The random draw can be both with and without replacement. In the case of the prior, some minibatches can be reused while others are skipped in their entirety. Whereas in the latter case, all minibatches have to be used to constitute one epoch. The difference between these implementations of Stochastic Gradient Descent is that the prior version with replacement tends to converge somewhat faster than the version without replacement [4]. Furthermore, the target vector t is shuffled in such a way that the rows in X_{perm} still adheres to the same target t_i.

With the construction of X_{perm}, the algorithm moves forward by computing the gradient and moving along its direction one mini-batch at a time. Traversing through all mini-batches or a set number of them constitutes to one epoch. The algorithm just described is then rerun for a specified number of epochs.

The gradient step can be described mathematically as follows in Equation (2)

$$\beta_{j+1} = \beta_j - \eta_j \sum_{i \in B_k}^{n} \nabla_\beta c_i(\boldsymbol{x}_i, \beta_j) \tag{2}$$

Stochastic Gradient Descent is implemented as pseudocode in Algorithm (1)

```
Data: Design Matrix (X), target array (t) and
      initial guess at predictors $\theta$
Result: Estimated value of the predictors $\theta$
for epoch in number of epochs do
    for batch in number of batches do
        $x_i \leftarrow$ X[batch];
        $t_i \leftarrow$ t[batch];
        Compute $\nabla_\theta C(\theta)$ with respect to $x_i$ and $t_i$;
        $\eta \leftarrow learning_schedule(\eta, \dots)$;
        $\theta \leftarrow \theta - \eta * \nabla_\theta C(\theta)$;
    end
end
return $\theta$
```

Algorithm 1: Stochastic Gradient Descent with minibatches and learning rate scheduler

Adding momentum to SGD

Equation (2) can be generalized with a momentum term as follows

$$\begin{aligned} \boldsymbol{v}_t &= \gamma \boldsymbol{v}_{t-1} + \eta_y \nabla_\theta E(\boldsymbol{\theta}_t) \\ \boldsymbol{\theta}_{t+1} &= \boldsymbol{\theta}_t - \boldsymbol{v}_t \end{aligned} \tag{3}$$

Where the momentum parameter $\gamma \in [0, 1]$. The intuition behind the introduction of the momentum parameter is the drag coefficient used in mechanics to describe friction. It's usage is similar when introduced in Stochastic Gradient Descent, as the momentum parameter enables the possibility for the gradient to attain momentum when traversing along a slope. Eventually if the gradient is ascending a gradient, due to the momentum, it might stop and return back down towards the minima. Thus Stochastic Gradient Descent with momentum enables the gradient to traverse through some local minima, while eventually stopping at the global minima. The algorithmic implementation of Momentum Stochastic Gradient Descent can be seen in Algorithm (2).

```
Data: Design Matrix (X), target array (t) and
      initial guess at predictors $\theta$
Result: Estimated value of the predictors $\theta$
for epoch in number of epochs do
    for batch in number of batches do
        $x_i \leftarrow$ X[batch];
        $t_i \leftarrow$ t[batch];
        Compute $\nabla_\theta C(\theta)$ with respect to $x_i$ and $t_i$;
        $\eta \leftarrow learning\_schedule(\eta, \ldots)$;
        $v \leftarrow \gamma v + \eta * \nabla_\theta C(\theta)$;
        $\theta = \theta - v$;
    end
end
return $\theta$
```

Algorithm 2: Stochastic Gradient Descent with momentum, minibatches and learning rate scheduler

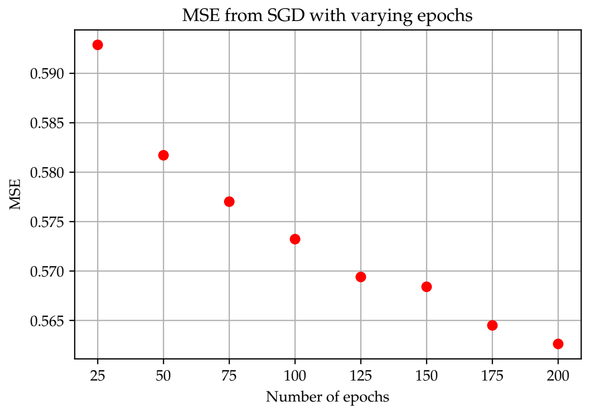

FIG. 1. MSE calculated from Stochastic Gradient Descent optimization of the predictors θ as function of the number of epochs with all other hyperparameters kept fixed

RESULTS

When creating the results, we note that all are generated from the Source Code in the appendix. For a further explanation on how to reproduce these results, we refer to the GitHub repository which contains a README explaining how to run the supplied source code.

Stochastic Gradient Descent

Our Stochastic Gradient Descent is implemented akin to Algorithm (1). Figure (1) plots the MSE computed from our SGD implementation against the number of epochs used.

The following Figure (3) plots the MSE as a function of batch_size using the SGD algorithm in 1.

Both Figures (3, 4) are created using a grid search algorithm over different values of η and λ, presented as a heatmap.

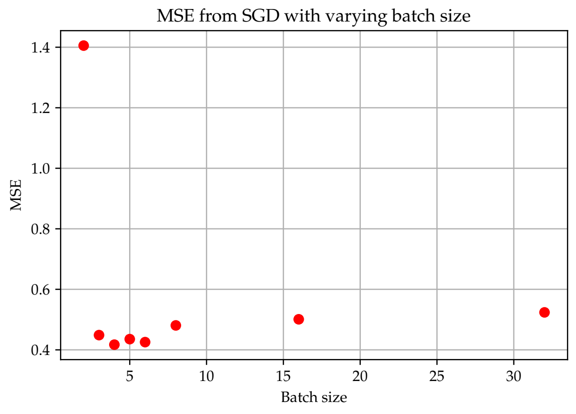

FIG. 2. MSE calculated from Stochastic Gradient Descent as function of the batch size with all other hyperparameters kept fixed

Table (I) displays MSE results computed from different configurations of the developed SGD algorithm as well as the MSE computed from the Momentum Stochastic Gradient Descent algorithm explained in Algorithm (2).

DISCUSSION

Analyzing our Stochastic Gradient Descent implementation

CONCLUSIONS

Source Code

Link to github repository containing all developed code for this project: `https://github.com/AndreasBordvik/FYS-STK4155-Prj2_report`

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] G. Cybenko. Approximation by superpositions of a sigmoidal function. 2(4):303–314, dec 1989.

[3] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

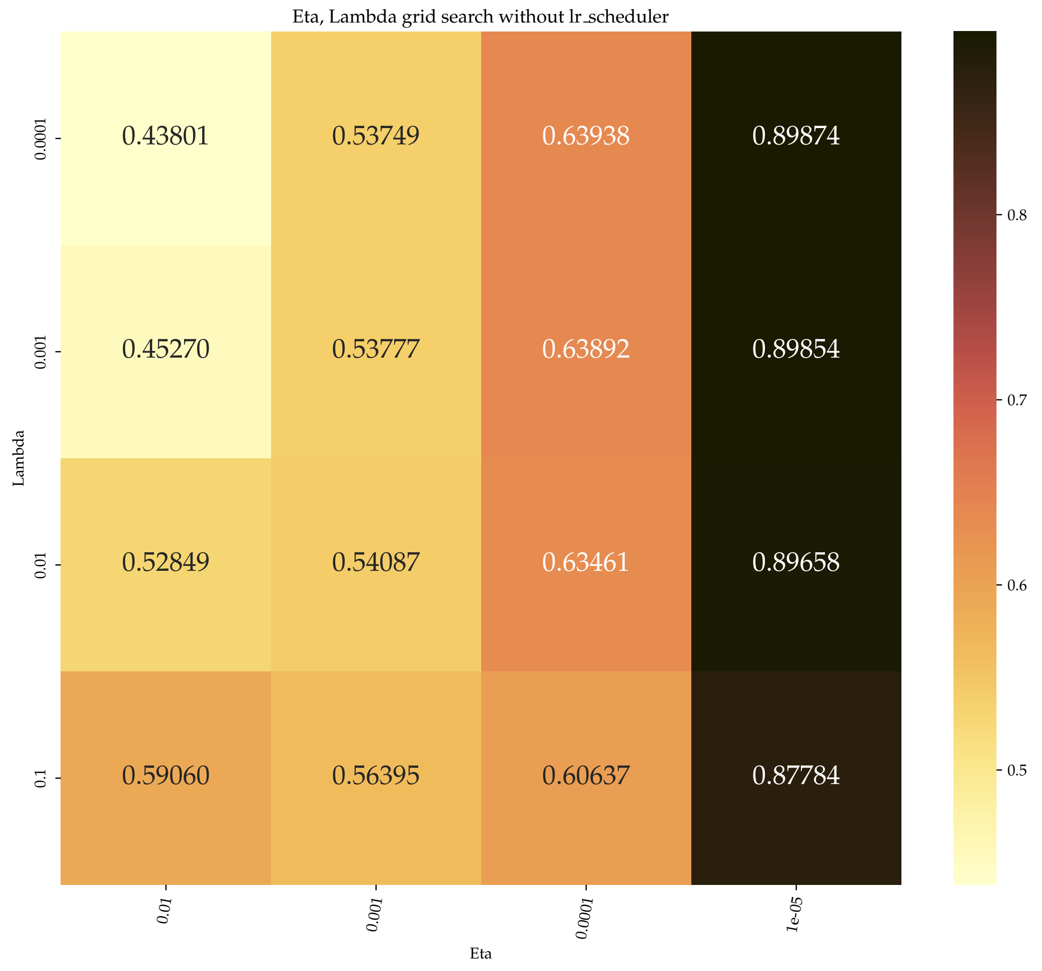

FIG. 3. MSE calculated for varying η and λ values in a grid search with learning rate scheduler turned off

TABLE I. MSE values from different SGD configurations based on the optimal hyperparameters as found in the grid search

SGD with lr scheduler	Momentum SGD with lr scheduler	SGD w.o. lr scheduler	SciKit-learn	Ridge
0.4783	0.4377	0.4203	1.0764	0.3256

[4] Aurélien Géron. *Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow*. O'Reilly UK Ltd., October 2019.

[5] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. 2(5):359–366, jan 1989.

[6] Michael A. Nielsen. *Neural Networks and Deep Learning*. Determination Press, 2015. Last update: Thu Dec 26 15:26:33 2019.

[7] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. December 2019.

[8] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

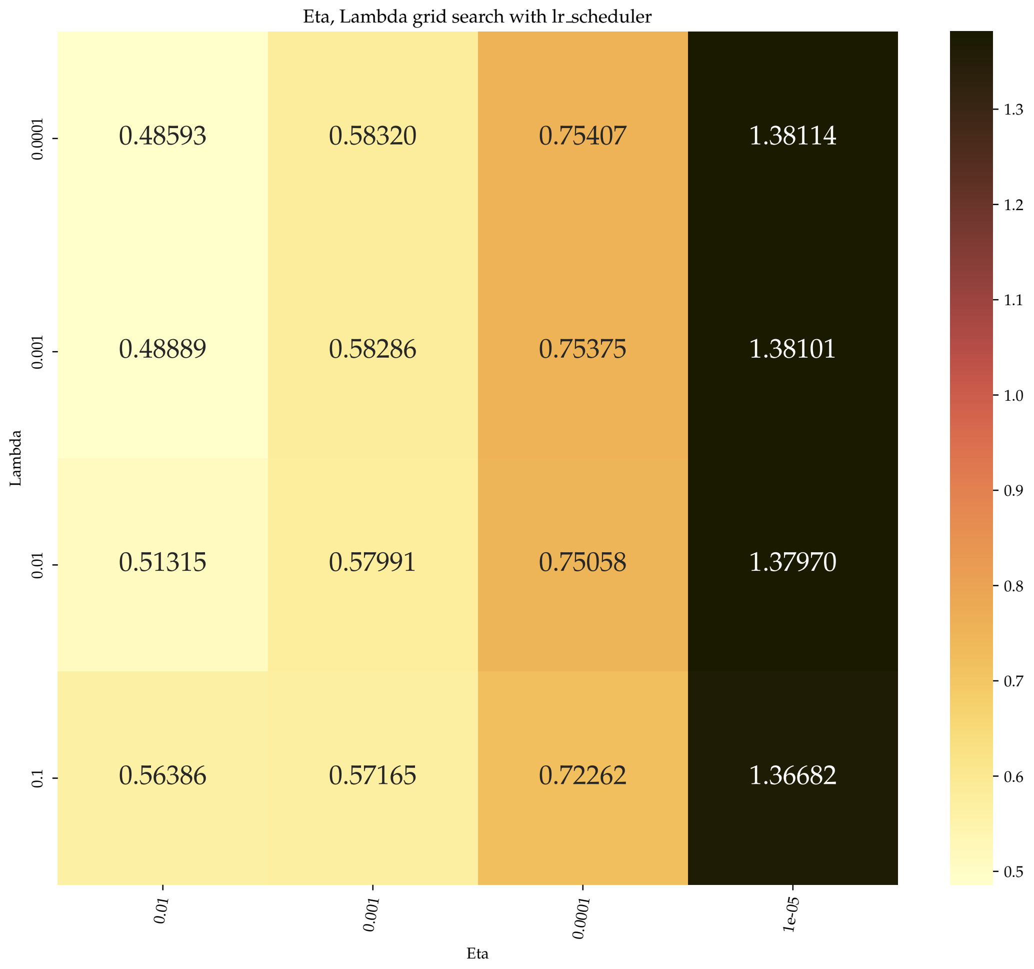

FIG. 4. MSE calculated for varying η and λ values in a grid search with learning rate scheduler turned on