"How Much Data is Enough?" Learning Curves for Machine Learning

Investigating alternatives to the Levenberg-Marquardt algorithm for learning curve extrapolation.

Lucian Negru[1]

Supervisors: Dr. Jesse Krijthe[1]**, Dr. Tom Viering**[1]

[1]EEMCS, Delft University of Technology, The Netherlands

A Thesis Submitted to EEMCS Faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering
June 25, 2023

Name of the student: Lucian Negru
Final project course: CSE3000 Research Project
Thesis committee: Dr. Jesse Krijthe, Dr. Tom Viering, Dr. Zhengjun Yue

Abstract

The conducted research explores fitting algorithms for learning curves. Learning curves describe how the performance of a machine learning model changes with the size of the training input. Therefore, fitting these learning curves and extrapolating them can help determine the required data set size for any desired performance.

The paper specifically explores the Learning Curve Database (LCDB) [1] and investigates alternative fitting algorithms to the employed Levenberg-Marquardt (LM). These algorithms are Gradient Descent and BFGS, and the paper aims to determine whether they are more suitable for fitting learning curves than LM.

The algorithms were implemented, both in their default and optimised states, and the results were compared to LM. The results measured mean-squared error (MSE), L1 Loss, individual parametric model performance, and computation time.

The findings showed that Gradient Descent is not a suitable alternative to LM; however, BFGS proved to be competitive, as it is practically identical in performance while being significantly faster than LM.

Further exploration of the BFGS algorithm and its application on learning curve fitting is recommended. Comparisons between the MSE distribution of LM and BFGS can be further explored, as well as comparisons on new parametric models, learners, and datasets.

Keywords: learning curve · Levenberg-Marquardt · Gradient Descent · Broyden-Fletcher-Goldfarb-Shanno · LCDB

1 Introduction

This research focuses on *learning curves*. Viering and Loog [2] state that learning curves display the performance of machine learning models relative to the number of training examples. Generally, the more data and training samples a model is fed, the better its performance. However, the rate of improvement is not always increasing with the size of the input.

Estimating how much improvement can be expected given a certain increase in training samples is important for applications that deal with complex data and expensive labelling. As applications of machine learning models are developed and applied at an increased rate, the training data they require will have to be larger, more complex, and more extensively labelled. OpenAI researcher Ben Cottier [3] estimates the training costs of large AI models to reach as high as $300 million by 2030. These include the cost of resources (such as hardware and electricity) as well as the cost of experts labelling data. Therefore, the performance gained from enlarging the training data must outweigh the increase in costs. For this reason, we are interested in learning curves; to observe the estimated increase in performance given an increase in data size.

Related work: The research conducted by Viering and Loog [2] goes in-depth on various models and their respective learning curve shapes, as well as provides the results of their rounds of training in the Learning Curve Database (LCDB) [1]. Their research aimed to study the different shapes learning curves can have and why they do so. The results of this study indicate that, for the majority of learning curves, error rates are non-increasing as training set sizes increase.

The study uses fitting algorithms to assign a parametric model to the error data and extrapolate. A limitation of the research is its sole use of the Levenberg-Marquardt (LM) [4, 5] fitting algorithm without exploring the performance of other promising optimisation methods such as the Gradient Descent[1] and the Broyden-Fletcher-Goldfarb-Shanno[2] (BFGS) algorithm, which are widely used in minimisation and optimisation applications.

This research, therefore, aims to answer the following question:

Are Gradient Descent and BFGS better suited for learning curve extrapolation than Levenberg-Marquardt?

In Section 2, the Methodology will be described, including the three algorithms and the general fitting procedure of the LCDB. Section 3 will explain the setup conditions of the experiments. The results will be shown in Section 4 and their implications will be discussed further in Section 5.

2 Methodology

This work considers two alternative methods of fitting learning curves, specifically Gradient Descent and BFGS. This section will explore the curve fitting method of the LCDB theoretically, and the two proposed alternative algorithms.

We will first explore the learning curve fitting process, followed by the currently implemented algorithm, LM. Then, we shall explore the two alternatives and how they function.

2.1 LCDB Fitting Procedure

The LCDB is a database consisting of 4,367 distinct learning curves. Every learning curve features different learners implemented in Python and has varying error rates given the training data size. Therefore, the LCDB provides an extensive analysis of how some of the most widely used learners perform.

The process of fitting the curves begins with generating the anchors. These are the average error rates of 125 testing set runs at any particular training set size. The anchors are calculated by running each learner on the specific training size of the data set 125 times, and their varying results are averaged into a single anchor.

[1]https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.fmin_cg.html

[2]https://docs.scipy.org/doc/scipy/reference/optimize.minimize-bfgs.html

Parametric models are chosen for each learning curve which best display the trend of the data. These models can have anywhere from 1 to 4 parameters and various functions. The best performing models from Viering and Loog's study [1] are *mmf4* and *wbl4* seen bellow:

$$\begin{aligned} mmf4 &= (\alpha\beta + \gamma x^{\delta})/(\beta + x^{\delta}) \\ wbl4 &= -\beta exp(-\alpha x^{\delta}) + \gamma \end{aligned}$$

Multiple parametric model fits are performed on the final anchor points, each with a different number of training and testing anchors. The training anchors are used in the fitting procedure, they are the points on which the fitting algorithm attempts to fit. The testing anchors are used to evaluate the fit concerning the extrapolation and prediction performances. An important distinction is that achieving a low MSE on the training anchors does not always guarantee a low MSE on the testing anchors. Therefore, when running the fitting algorithms, it is important to consider which type of anchor MSE should be minimised. This research will focus on minimising the MSE on testing anchors, as those best represent extrapolation.

Finally, the results of the fits can be analysed to determine which learning curve best describes the performance of the learner and data set.

2.2 Defining Performance

The research compares the performances of the fitting algorithms. This performance is composed of MSE and L1 Loss. Comparisons between computation times have also been addressed, but not regarded as 'performance'. The functions of these metrics can be observed below, where n represents the amount of data.

$$\begin{aligned} MSE &= \tfrac{1}{n}\textstyle\sum_{i=1}^{n}(Y_{actual} - Y_{predicted})^2 \\ L1\ Loss\ Function &= \textstyle\sum_{i=1}^{n}|Y_{actual} - Y_{predicted}| \end{aligned}$$

2.3 The Levenberg-Marquardt Algorithm

The Levenberg-Marquardt (LM) algorithm is an iterative method of finding the minimum of a function that represents the sum of squares of non-linear functions [4, 5]. It can be described as an interpolation between Gauss-Newton (Newton's) method and Gradient Descent [5], and benefits from higher robustness than the individual methods it is derived from.

$$(\mathbf{J}_k{}^T\, \mathbf{J}_k + \lambda_k\, \mathbf{I})\, p_k \;=\; -\mathbf{J}_k{}^T\, f_k$$

The above function describes the LM algorithm searching in the direction of the optimum solution p. $\mathbf{J}_k$ represents the Jacobian – a matrix of partial derivatives – of the objective function at point k. $\mathbf{I}$ is the identity matrix. λ is the dampening factor and f is the objective function.

One interesting property of the LM algorithm is that it behaves differently depending on the value of the damping factor λ – which relates to how close it is to the solution. When it is far from the solution it behaves similarly to Gradient Descent, and when it is close to the solution it behaves similarly to the Newton's method [6]. Therefore, by alternating between the two algorithms, LM does not suffer from their limitations.

The LM implementation in the LCDB makes use of the SciPy *least_squares*[3] library and is aimed at finding the optimum parameters for the model curve such that the MSE of the testing anchors is minimal. One limitation of the implementation is that the initial parameters of the model curves are selected from a normal distribution of values rather than a model-specific initialisation, resulting in higher chances of the algorithm reaching a local minimum rather than a global one [7].

2.4 Gradient Descent

Gradient Descent is an optimisation method widely used in mathematics and machine learning. It aims to minimise objective functions, or in the case of the LCDB, minimise the MSE of the generated fits. It does so by following the direction opposite to the gradient to find the minimum of the aforementioned function [8]. This leads to a solution that progresses very quickly at the start but shuffles back and forth the nearer it gets to the solution [9].

To implement the Gradient Descent algorithm on the fittings of the LCDB, the SciPy *optimize.fmin_cg* function was used on the LCDB's objective function that needed to be minimised. While the method does not allow for extensive hyper-parameter tuning, the tolerance for termination can be altered to influence the learning rate – and by extension the step size – by decreasing or increasing the acceptable offshoot towards the optimum solution. Offshoot refers to Gradient Descent's possibility to skip over the minimum of the function, causing it to iterate in the opposite direction.

2.5 The Broyden-Fletcher-Goldfarb-Shanno Algorithm

The Broyden-Fletcher-Goldfarb-Shanno (BFGS) algorithm is another promising non-linear optimisation algorithm widely used for unconstrained problems – problems in which the variables can take any value. Similarly to gradient descent, it makes use of the estimated gradient at a certain point. The gradient is estimated numerically using secants, which are lines that cut through the function curve at two points $x - \delta$ and $x + \delta$ to estimate the gradient at x [10]. The algorithm decreases the size of δ as it approaches the minimum of the function.

The advantage that BFGS has over Gradient Descent and Newton's method is that it does not need matrix inversions for calculating the gradient, resulting in a lower computational complexity of $O(n^2)$, compared to $O(kn^2)$ and $O(n^3)$ respectively. Therefore, it may prove to not only be an accurate optimisation algorithm but also faster than the current Levenberg-Marquardt implementation – composed of both Gradient Descent and Newton's method.

[3]https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.least_squares.html

3 Experimental Setup

Having explored the current fitting implementation of the LCDB, as well as the two alternative algorithms, the experiments were carried out to answer the research question: are Gradient Descent and BFGS better suited for learning curve extrapolation than Levenberg-Marquardt? In the experiments, performance was measured using the metrics from Section 2.2. The research question was split into the following two hypotheses:

- Gradient Descent offers a more performant fitting alternative for learning curves than Levenberg-Marquardt.
- BFGS offers a more performant fitting alternative for learning curves than Levenberg-Marquardt.

The following section will describe the environment and explore the reasoning and implementation behind the two experiments. The first experiment explored the performance of the alternative algorithms implemented with their default parameters – matching the implementation of LM. The second experiment featured optimised implementations of the alternative algorithms.

3.1 Experiment Environment

The experiments featured in this study have been written in Python[4] and compiled using Jupyter Notebooks[5]; these are all available on the author's GitHub[6].

The experiments use the LCDB library[7] for accessing the learners and datasets used in the initial study by Viering and Loog [2], as well as for comparing results. The learners are implemented using the Scikit Learn[8] library and the datasets are retrieved from OpenML [11].

3.2 Experiment 1: Default Algorithm Performance

The alternative algorithms were implemented and ran upon the same 10,000 randomly sampled learning curves from the LCDB with their maximum number of anchors. 10,000 samples were chosen as it is a large portion of all samples and can accurately represent the LCDB as a whole while keeping the duration of the experiment down.

Gradient Descent was implemented with the aforementioned *optimize.fmin_cg* library and a scalar objective function – the function that is minimised. The BFGS algorithm was implemented by passing the MSE function gradient as the Jacobian matrix, as recommended in the SciPy library. The MSE function gradient is calculated numerically using the following function:

```
def objective_der(beta):
    grad = np.zeros_like(beta)
    for i in range(len(beta)):
        beta_plus = beta.copy()
        beta_minus = beta.copy()
        beta_plus[i] += 1e-8
        beta_minus[i] -= 1e-8
        grad[i] = (objective(beta_plus)
                  - objective(beta_minus))
                  / (2 * 1e-8)
    return grad
```

It should be noted that the function uses estimations and in the instance of a highly varying learning curve, such as a very sudden and large increase, this may affect the accuracy of the estimation.

3.3 Experiment 2: Optimised Algorithm Performance

The second experiment featured more optimised versions of the Gradient Descent and BFGS algorithms. The experiment was run on various values for hyperparameters and the best-performing combinations were kept in the results section.

The Gradient Descent implementation discussed in section 2.4 does not allow for extensive hyperparameter tuning; however, it does allow for tuning of the tolerance for termination. This also affects the rate at which the solution is reached. The final value used for the tolerance was $1e-5$.

For BFGS, the main difference was the calculation of the Jacobian matrix. Previously, the gradient of the MSE function was calculated numerically. While that is fast for most learning curves, some exceptions occur where the algorithm reaches the maximum number of iterations – needed to avoid infinite loops – and settles on the current optimisation. This can be improved by allowing for more iterations per learning curve, increased from 5 to 15, requiring faster gradient calculations. Therefore, the Autograd[9] library is used to instead calculate the analytical gradient. Thus, the gradient of the actual parametric model function can be used directly without needing to estimate [12], increasing the number of possible iterations significantly.

4 Results

This section features the results of the aforementioned experiments and analysis of the various performance metrics, such as testing data MSE, L1 Loss, computation time, and parametric model performance. Firstly, the results of experiment 1 will be analysed, and then experiment 2.

4.1 Experiment 1 Results: Default Algorithm Performance

Given that the alternative algorithms have been implemented with their default parameters, to match the original LM implementation, large performance improvements were not expected. When running the algorithms on 10,000 samples, it became clear that the size was too much for Gradient Descent to compute the fitting; causing it to crash around the one-hour

[4]https://www.python.org/
[5]https://jupyter.org/
[6]https://github.com/MissingCurlyBracket/research-project
[7]https://github.com/fmohr/lcdb
[8]https://scikit-learn.org/stable/
[9]https://pytorch.org/docs/stable/notes/autograd.html

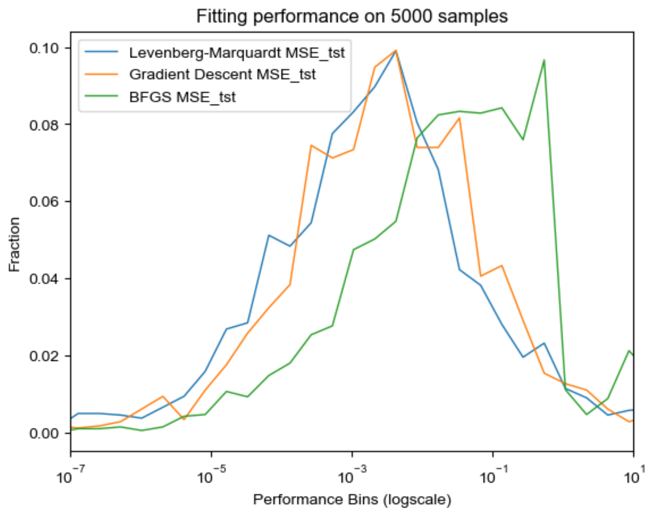

Figure 1: Plot showing test anchor MSE distributions of the three default algorithms, as tested on 5,000 random samples. The X-axis represents the value of MSE, and the Y-axis represents the portion of samples featuring the certain MSE, i.e., the largest green line peak would show that approximately 10% of the samples resulted in an MSE of approximately $10^{-0.5}$. In plots such as these, right-sided distributions mean higher MSE.

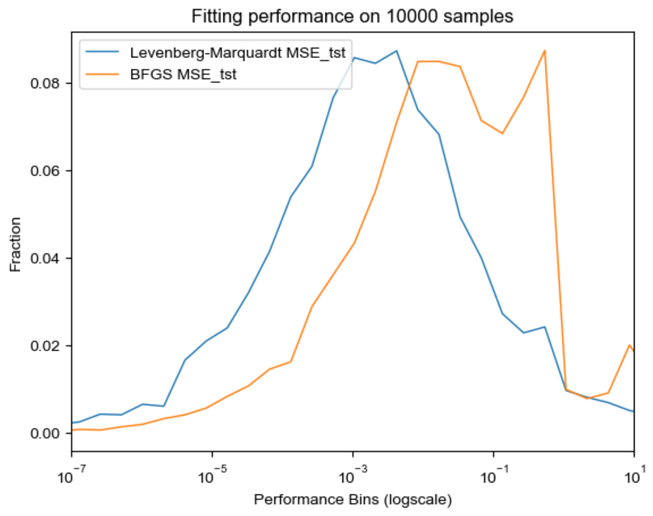

Figure 2: Plot showing test anchor MSE distributions of LM and BFGS, as tested on 10,000 samples. Graph is read as in Figure 1.

mark each time. This is most likely due to a Jupyter Notebook limitation such as maximum available memory. Therefore, results for 5,000 samples are also included to allow for equal comparisons. Figures 1 and 2 showcase the results.

Observing the results, it is clear that Gradient Descent is very similar in performance to the initial LM, nearly perfectly doing so on the 5,000 samples run. The default implementation of BFGS, however, seems to perform consistently worse by degrees of approximately $10 \sim 100$ times higher MSE.

A promising result for BFGS is the narrower distribution, although on average it performs worse, its performance does not vary as much as LM or Gradient Descent; this may be an indication that a more optimised version of this algorithm may prove more consistent than LM.

Table 1: Table showcasing the fitting times of the three tested default algorithms on 128, 5,000, and 10,000 sample sizes. The outlying Gradient Descent time appears as expected due to the algorithm's limitations when being near a solution. Furthermore, BFGS shows considerably lower fitting time than Levenberg-Marquardt.

Algorithm	**Time on sample size**		
	128 size	5,000 size	10,000 size
Levenberg-Marquardt	4.73 s	88.86 s	477.74 s
Gradient Descent	64.00 s	593.09 s	N/A
BFGS	3.88 s	66.98 s	406.69 s

One outstanding factor was the computation time of these learning curves, shown below in Table 1. As stated in Section 2.4, Gradient Descent slows down significantly when approaching the minimum objective, resulting in the increased fitting time. It can be observed that the differences in complexity, stated in Section 2.5, result in the BFGS algorithm being considerably faster than LM on all sample sizes.

4.2 Experiment 2 Results: Optimised Algorithm Performance

The results of the second experiment show that the optimisations improved the overall performance of the algorithms, particularly BFGS. As with the first experiment, 5,000 and 10,000 sample runs were fitted to test both the slower Gradient Descent and the BFGS algorithms.

Gradient Descent performance has shown a trade-off between MSE and computation time due to the increased tolerance of termination. The average MSE increased by a degree of 10, as seen in Figure 3, but the computation time decreased from 593.09 to 477.95. Although not a proportional trade-off, some learning curve fitting tasks may benefit from the increased speed while not being affected by the higher MSE.

BFGS has benefited greatly from the analytical gradient optimisation, increasing the number of iterations per second, as well as the larger number of repetitions, increasing the accuracy of the final optimisation. In the 5,000 sample run, Figure 3, it performed within $\sim 2\%$ average MSE and 1% L1 Loss off LM. In the 10,000 sample run, BFGS showed 3.6% higher MSE than LM and $< 1\%$ L1 Loss, represented in Figures 4 and 5 respectively. The faster analytical gradient calculation also resulted in a decreased computation time from 406.69 seconds to 366.53, making BFGS practically identical in performance to LM, while being $\sim 23\%$ faster.

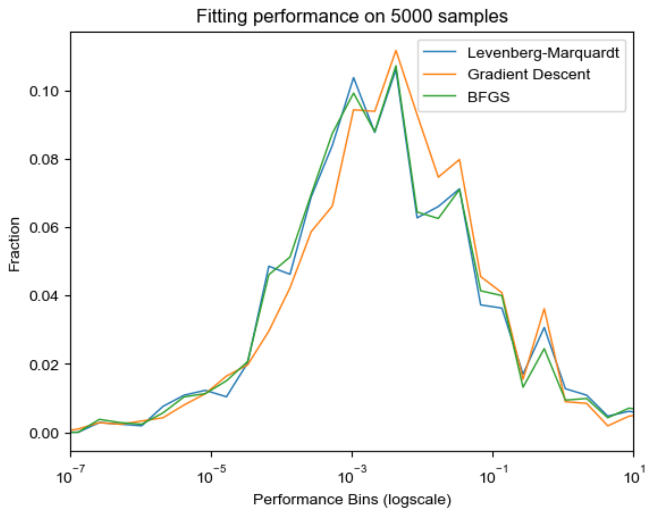

Figure 3: The testing anchor MSE distributions of all three algorithms after optimisations, tested on 5,000 samples. Graph is read as in Figure 1.

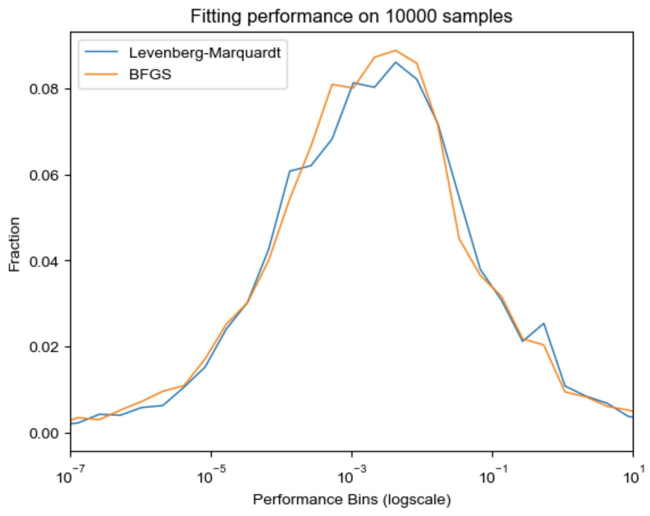

Figure 4: Plot showing the testing anchors MSE distribution of the optimised BFGS algorithm, with analytical gradient calculation, compared to the LM implementation. Tested on 10,000 samples. Graph is read as in Figure 1.

Interestingly, for the outer MSE performance bins $< 10^{-3}$ and $> 10^{-2}$ BFGS consistently shows lower distribution, while the inner range features higher distribution of MSE. These observations are carried out to the 10,000 sample run, Figure 4, showing that the distribution of BFGS has lower a variance than that of LM.

Table 2: The average testing anchor MSE of the top five performing parametric models for each of the tested algorithms. The data was generated from a 5,000 sample run and has been cleaned up before analysis to remove overflows, NaNs, infinities, and fits with over 100 MSE.

Levenberg-Marquardt		**Gradient Descent**		**BFGS**	
Curve Model	**MSE**	**Curve Model**	**MSE**	**Curve Model**	**MSE**
last1	0.005832	last1	0.005832	last1	0.005832
wbl4	0.092318	wbl4	0.586516	wbl4	0.120457
exp4	0.094922	exp4	0.693752	exp4	0.094035
mmf4	0.113522	mmf4	0.846390	mmf4	0.135952
pow4	0.130181	pow4	1.136965	logpower3	0.207976

Taking a look at the specific parametric model performance of the tested algorithms, Table 2, we see that the LM results indicate similar top-performing models as the initial study by Viering and Loog [2]. Gradient Descent shows the same top contenders as LM, albeit with an average MSE 10 times higher, unsurprising given the results in Figure 4 and the fact that LM implements Gradient Descent. Surprisingly, BFGS also features most of the same top-performing parametric models (logpow3 out-performs pow4). This result supports the initial study's findings that 4-parameter models are the most competitive for extrapolating learning curves.

4.3 Answering The Hypotheses

Gradient Descent showed very similar performance to the LM implementation on the LCDB, however, it did so at the cost of very high computation time. This indicates that there would be no purpose in considering Gradient Descent over LM, as the compromise between MSE performance and computation time would result in an implementation that is either not accurate or extremely slow.

BFGS, on the other hand, proved to be a very competitive algorithm. It matched the LM implementation in MSE of the testing anchors, often resulting in better performance for some parametric models. Furthermore, it is a considerably faster algorithm, allowing for more use cases in learning curve extrapolation. Therefore, BFGS has been found to perform better at fitting learning curves than Levenberg-Marquardt.

5 Discussion

With the obtained results, the algorithms can be analysed to determine where they excel and fall short. This section will offer analysis and theoretical explanations behind the results, as well as limitations to the research.

5.1 Insights

These experiments helped compile new insights about fitting algorithms and their applications on learning curves. In general, Gradient Descent was determined to not be a suitable alternative to the Levenberg-Marquardt algorithm due to its substantial computation time and no real benefits.

However, the fact that Gradient Descent follows so closely to LM may be an indication that LM relies more on the Gradient Descent functionality and not as much on the Gauss-Newton counterpart. This would entail that LM, and all tested algorithms in retrospect, manage to reach close to the optimum solution very fast.

The main finding of this study was the insight that BFGS is very competitive to LM, to the extent that it can be recommended as the alternative for learning curve fitting. It is as performant while being faster, and features slightly lower

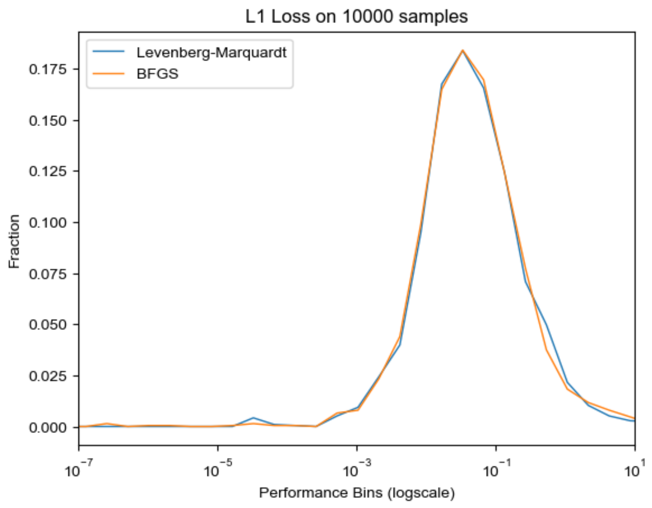

Figure 5: Plot showing the L1 Loss distribution of the optimised BFGS algorithms compared to LM. Graph is read as in Figure 1 but features L1 Los instead of MSE.

MSE variance on average (although this should be further tested).

The final insight gained from this study was that the best-case scenario runs for both LM and BFGS averaged around $10^{-3} \sim 10^{-2}$. Their performance differences on individual parametric models were also non-significant, suggesting that perhaps these results are as good as possible with the current parametric models. This outlines the possibility of exploring more parametric models in the future, as they are currently the determining factor in performance.

5.2 Limitations

One of the limitations encountered during the fitting process was the initialisation of the parameters. In the LCDB, before the fitting algorithm is run, parameters are initialised from a normal distribution. This method has been noted to have limitations by Kim [7], and alternative initialisation methods, such as K–Means clustering, have proved to be more performant. The performance of the alternative algorithms may have increased if initialisation was optimised as well.

Another limitation is the experiments being run on an upper bound of 10,000 samples, instead of the entire data set. This choice was made to save time, which could later be used for further optimisation, but will always be limited in not showing the 'full picture'.

The Gradient Descent algorithm could have also been more thoroughly optimised if it were implemented by hand instead of using SciPy. This could have been done with the TensorFlow[10] or PyTorch[11] libraries.

The final limitation would be that the experiments were run on datasets with the maximum number of training anchors. Alternative tests could be made on various 'ranks', different splits between training and testing anchors, to measure consistency or lack thereof. It could be that BFGS performs very similarly to LM when considering as much data is available as possible, but what if the data is minimal? Would it still perform as well?

[10]https://www.tensorflow.org/api_docs/python/tf/compat/v1/train/GradientDescentOptimizer

[11]https://pytorch.org/docs/stable/generated/torch.optim.SGD.html

6 Responsible Research

The following section will explore the reproducibility of the study following the Yale Law Roundtable recommendations [13] for upholding credibility.

The first recommendation is to publish the source code of the experiments and statistical analyses [13]. The source code for the implementations of the algorithms and the statistical scripts can be publicly viewed at https://github.com/MissingCurlyBracket/research-project. Furthermore, the used learners, datasets, and parametric models can be found at https://github.com/fmohr/lcdb.

The second recommendation is to assign a unique version ID to the pieces of software developed. This recommendation does not apply to the study, as it contains the sole version of the implementations and no future updates are yet considered.

The third recommendation is to describe the computing environments and software used in producing the experiments. The implementations and experiments were written in Jupyter Notebooks using Python version 3.8.3. The experiments were run on an 8-core M1 Pro processor on macOS 13.4. The required libraries and their versions can also be found at https://github.com/fmohr/lcdb.

The fourth recommendation is to use open-licensed code. As the code developed in the study is not licensed, this recommendation does not apply.

As the fifth recommendation, Yale Law recommends that the study be published using an open-access contract. This paper will be featured in the TU Delft Repository, open to the public.

The sixth and final recommendation is to consider readability and re-usability. The implemented algorithms have been named and written using standard Python conventions. The use of the code and the procedure for running it has been documented, both in https://github.com/MissingCurlyBracket/research-project as well as in https://github.com/fmohr/lcdb.

7 Conclusions and Future Work

To conclude the study, alternative algorithms were suggested for fitting learning curves in the LCDB – these being Gradient Descent and BFGS. Default and optimised versions of both alternative algorithms were implemented in the fitting procedure and were analysed based on the MSE of the testing anchors, L1 Loss of the testing anchors, average MSE on individual parametric models, and computation time.

The results favoured against the first hypothesis and Gradient Descent indicated to not be a suitable alternative to Levenberg-Marquardt for learning curve fitting, according to the tests. On the other hand, the first hypothesis was supported, showing that BFGS performs practically to the same degree of accuracy as LM while being significantly faster. Therefore, BFGS is a suitable alternative to LM for learning curve fitting.

Further research is recommended in the field, particularly regarding the promising results of BFGS. Firstly, the experiments can be run on the entirety of the LCDB, a test that proved infeasible due to time constraints. Second, more parametric models can be explored, perhaps with even more parameters. The most interesting finding that merits further investigation, I believe, is the smaller MSE variance of BFGS and its significance. If indeed BFGS shows significant signs of being less varied than LM, that could mean it can be applied in practice more consistently.

References

[1] Felix Mohr, Tom J Viering, Marco Loog, and Jan N van Rijn. Lcdb 1.0: An extensive learning curves database for classification tasks. In *Machine Learning and Knowledge Discovery in Databases. Research Track - European Conference, ECML PKDD 2022, Grenoble, France, September 19-24, 2022*, 2022.

[2] Tom Viering and Marco Loog. The shape of learning curves: A review. *IEEE Trans. Pattern Anal. Mach. Intell.*, PP:1–20, November 2022.

[3] Ben Cottier. Trends in the dollar training cost of machine learning systems, 2023. Accessed: 2023-4-25.

[4] Kenneth Levenberg. A method for the solution of certain non-linear problems in least squares. *Quarterly of applied mathematics*, 2(2):164–168, 1944.

[5] Donald W Marquardt. An algorithm for least-squares estimation of nonlinear parameters. *Journal of the society for Industrial and Applied Mathematics*, 11(2):431–441, 1963.

[6] Manolis IA Lourakis et al. A brief description of the levenberg-marquardt algorithm implemented by levmar. *Foundation of Research and Technology*, 4(1):1–6, 2005.

[7] Donghwi Kim. Different approaches to fitting and extrapolating the learning curve. *EEMCS Faculty Delft University of Technology*, 2022.

[8] Haskell B Curry. The method of steepest descent for non-linear minimization problems. *Quarterly of Applied Mathematics*, 2(3):258–261, 1944.

[9] Claude Lemaréchal. Cauchy and the gradient method. *Doc Math Extra*, 251(254):10, 2012.

[10] JE Dennis Jr and Robert B Schnabel. Secant methods for unconstrained minimization. *Numerical Methods for Unconstrained Optimization and Nonlinear Equations, Englewood Cliffs, NJ: Prentice-Hall*, pages 194–215, 1983.

[11] Matthias Feurer, Jan N Van Rijn, Arlind Kadra, Pieter Gijsbers, Neeratyoy Mallik, Sahithya Ravi, Andreas Müller, Joaquin Vanschoren, and Frank Hutter. Openml-python: an extensible python api for openml. *The Journal of Machine Learning Research*, 22(1):4573–4577, 2021.

[12] Fei-Fei Li, Yunzhu Li, and Ruohan Gao. Convolutional neural networks for visual recognition. *Stanford Vision and Learning Lab (SVL)*, 2023.

[13] Victoria C Stodden. Reproducible research: Addressing the need for data and code sharing in computational science. 2010.