Mini-Batching, Gradient-Clipping, First-versus Second-Order
What Works in Gradient-Based Coefficient Optimisation for Symbolic Regression'

Harrison, Joe; Virgolin, Marco; Alderliesten, Tanja; Bosman, Peter

DOI
10.1145/3583131.3590368

Publication date
2023

Document Version
Final published version

Published in
GECCO 2023 - Proceedings of the 2023 Genetic and Evolutionary Computation Conference

Citation (APA)
Harrison, J., Virgolin, M., Alderliesten, T., & Bosman, P. (2023). Mini-Batching, Gradient-Clipping, First-versus Second-Order: What Works in Gradient-Based Coefficient Optimisation for Symbolic Regression'. In *GECCO 2023 - Proceedings of the 2023 Genetic and Evolutionary Computation Conference* (pp. 1127-1136). (GECCO 2023 - Proceedings of the 2023 Genetic and Evolutionary Computation Conference). Association for Computing Machinery (ACM). https://doi.org/10.1145/3583131.3590368

Important note
To cite this publication, please use the final published version (if applicable).
Please check the document version above.

Mini-Batching, Gradient-Clipping, First- versus Second-Order: What Works in Gradient-Based Coefficient Optimisation for Symbolic Regression?

Joe Harrison
Joe.Harrison@cwi.nl
Centrum Wiskunde & Informatica
Amsterdam, The Netherlands

Marco Virgolin
Marco.Virgolin@cwi.nl
Centrum Wiskunde & Informatica
Amsterdam, The Netherlands

Tanja Alderliesten
T.Alderliesten@lumc.nl
Leiden University Medical Center
Leiden, The Netherlands

Peter A.N. Bosman
Peter.Bosman@cwi.nl
Centrum Wiskunde & Informatica
Amsterdam, The Netherlands
Delft University of Technology
Deflt, The Netherlands

ABSTRACT

The aim of Symbolic Regression (SR) is to discover interpretable expressions that accurately describe data. The accuracy of an expression depends on both its structure and coefficients. To keep the structure simple enough to be interpretable, effective coefficient optimisation becomes key. Gradient-based optimisation is clearly effective at training neural networks in Deep Learning (DL), which can essentially be viewed as large, over-parameterised expressions: in this paper, we study how gradient-based optimisation techniques as often used in DL transfer to SR. In particular, we first assess what techniques work well across random SR expressions, independent of any specific SR algorithm. We find that mini-batching and gradient-clipping can be helpful (similar to DL), while second-order optimisers outperform first-order ones (different from DL). Next, we consider whether including gradient-based optimisation in Genetic Programming (GP), a classic SR algorithm, is beneficial. On five real-world datasets, in a generation-based comparison, we find that second-order optimisation outperforms coefficient mutation (or no optimisation). However, in time-based comparisons, performance gaps shrink substantially because the computational expensiveness of second-order optimisation causes GP to perform fewer generations. The interplay of computational costs between the optimisation of structure and coefficients is thus a critical aspect to consider.

CCS CONCEPTS

• **Computing methodologies** → **Genetic programming**; **Supervised learning by regression**; **Hybrid symbolic-numeric methods**.

GECCO '23, July 15–19, 2023, Lisbon, Portugal

ACM ISBN 979-8-4007-0119-1/23/07.
https://doi.org/10.1145/3583131.3590368

KEYWORDS

Gradient Descent, Symbolic Regression, Genetic Programming, Explainable AI, Coefficient Optimisation

ACM Reference Format:
Joe Harrison, Marco Virgolin, Tanja Alderliesten, and Peter A.N. Bosman. 2023. Mini-Batching, Gradient-Clipping, First- versus Second-Order: What Works in Gradient-Based Coefficient Optimisation for Symbolic Regression?. In *Genetic and Evolutionary Computation Conference (GECCO '23), July 15–19, 2023, Lisbon, Portugal.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3583131.3590368

1 INTRODUCTION

Symbolic Regression (SR) is the problem of finding both the structure (or parametric function) ϕ and coefficients (or parameters) θ that form a symbolic expression that fits a given set of data well [26, 51]. Symbolic expressions are appealing to use as machine learning models because they have the potential of being interpretable. Interpretability is an important property for using Artificial Intelligence (AI) in a responsible manner (see, e.g., the EU or the US Act on AI [1, 12]), as well as for scientific discovery. Indeed, SR has been used to discover interpretable models in, e.g., space exploration [18], medicine [47], and physics [35].

Different techniques have been proposed to search for the structure ϕ. These include exhaustive search [4, 21], greedy search [9], Genetic Programming (GP) [28], Deep Learning (DL) [20, 42], and hybrids [8, 23, 32, 46]. However, the accuracy of the entire model ϕ_θ clearly also depends on optimising the coefficients θ. If the optimisation of θ is poor, then the model ϕ_θ might need a large and complex structure ϕ in order to be sufficiently accurate. For example, GP is known to *bloat*, i.e., to evolve large structures with limited gains in accuracy [28]. For symbolic expressions to stand a chance at being interpretable, ϕ needs to remain sufficiently small. Therefore, it is critical that the coefficients θ are optimised effectively.

Gradient-based optimisation is popular in machine learning in general, and much work has been done on improving its efficacy in DL in particular. Advancements in activation functions [37], network architecture [15], normalisation schemes [17], and weight initialisation [14] have shown to substantially help train accurate

and well-generalising DL models with gradient-based optimisation. The number of contributions in the context of SR is far more modest (see Sec. 2). We cannot assume that findings that apply to DL transfer to SR, because SR expressions are substantially different from DL networks: SR expressions are normally represented with sparse, relatively shallow graphs, use operators that can induce very large or very small gradients, and are subject to structural change, while DL architectures are normally fixed. Moreover, DL models are often overparameterised, a property that does not apply to SR, where the goal is to find interpretable and thus relatively small models (i.e. expressions).

Overall, the literature is unclear on what configurations of gradient-based optimisation might work well in SR. In this paper, we address this issue by bringing together concepts and techniques, scattered in the literature on SR or prominent in the literature on DL, into an SR-focused comparison. Specifically, we design experiments with different configurations of gradient-based optimisers, gradient-clipping, learning rates, and options for mini-batching. These configurations are first applied to SR expressions using initialisation values for coefficients θ with static structure ϕ, to gain insights that are agnostic of the algorithm chosen to search for ϕ, as well as also in conjunction with GP, i.e., when ϕ evolves alongside the optimisation of θ, since GP is a classic method for SR.

2 RELATED WORK

Historically, SR has been tackled mostly with GP [30, 51]. One of the first works on gradient-based optimisation in GP for SR is [45]. This work considers a single specific configuration of gradient-based optimisation, i.e., 3 steps of full batch gradient descent (i.e., all available training data is used to calculate the gradient) with a learning rate (`lr`) of 0.5, reverting worsening steps. Similar settings are considered in [7][1], which investigates different options for which expressions should undergo coefficient optimisation. In [10], the authors assess whether feature standardisation, which is commonly used when training neural networks, also increases accuracy when applied to gradient-based optimisation in GP. Other works consider more involved optimisers, such as Adam (in combination with semantic GP) [43], Levenberg-Marquardt (LM) [6, 21, 26, 27], and Broyden–Fletcher–Goldfarb–Shanno (BFGS) [4, 5, 20, 35]. However, only a few of these works include some comparisons between different gradient-based optimisers/configurations [35, 43].

Also in works where the structure of symbolic expressions is sought with other methods than GP, an analysis of different coefficient optimisation options remains absent or limited. For example, [5, 20, 42] use DL models (e.g., recurrent neural nets or transformers) to generate symbolic expressions, whose coefficients are optimised, in all cases, with BFGS. Motivated by this, our work includes experiments where the structure is fixed, in addition to experiments with GP.

To the best of our knowledge, our work is the first that attempts to provide a more comprehensive comparison on gradient-based optimisation in SR (and GP). Lastly, it is important to mention that other approaches to coefficient optimisation exist besides gradient-based ones, such as mutation-based coefficient optimisation [3, 13, 50]. We compare gradient-based to classic mutation-based optimisation in our experiments with GP.

[1]This work states that gradient descent is used, however, the pseudocode describes coordinate descent, where coefficients are optimised one at a time.

3 METHOD

The use of gradient-based optimisation within a certain SR algorithm (e.g., GP) leads to complex dynamics that make it hard to isolate the contribution of gradient-based optimisation. Also, insights obtained using one SR algorithm may not hold for different SR algorithms (e.g., [20] outputs an expression in one shot, without any search iteration). Therefore, we split our contribution into (1) *static experiments*, where ϕ remains fixed and only θ is optimised, and (2) *dynamic experiments*, where ϕ is also optimised, via GP. This way, we can assess to what extent findings that apply to the static case transfer to GP.

3.1 Setup static experiments

The basic principle for these experiments is as follows: we generate random symbolic expressions and consider them as our target expression. We then perturb the coefficients within them and observe the capability of gradient-based optimisers to improve the coefficients. To test the sensitivity of the optimisation methods to the proximity to the optimum, we perturb the coefficients with three levels of Gaussian noise $\mathcal{N}(\mu, \sigma)$ with $(\mu, \sigma) \in \{(0.1, 0.01), (1, 0.1), (10, 1)\}$. Each perturbed symbolic expression is then optimised using the options for gradient-based optimisation that are described below.

In order to gain insight into the effect of gradient-based optimisation on typical structures ϕ that appear in SR, we need a wide variety of symbolic expressions. We sample 10,000 random symbolic expressions encoded by binary-unary trees (i.e., trees that contain operator nodes with an arity of 1 or 2), with operators from $O = \{+, -, \times, \div, Pow, Max, Min, Sin, Cos, Ln, Exp, \sqrt{\cdot}, Abs\}$. The sampling process is taken from [31], which is uniform in terms of trees that can occur up to a tree height of 4, with coefficients and input features sampled with an equal probability. This means that the largest tree that can be generated has 31 nodes, corresponding to a symbolic expression with 31 terms. We make this choice because arguably, symbolic expressions with 31 terms can already be very hard to interpret [49]. Coefficients and values for the input features are uniformly sampled between -10 and 10; the latter are used to generate training and validation sets with 1,000 observations. The process of sampling input features is repeated 10 times for each level of Gaussian noise per tree. During our input feature sampling process, we filter out: input features that lead to a Mean Squared Error (MSE) lower than 10^{-6} on the training data, expressions without coefficients, and expressions that result in invalid outputs on the training or validation data (e.g., $\sqrt{x}$ when x can be negative).

While in GP it is common to use *protected* operators to avoid invalid computations at all times, different implementations exist, and they may harm interpretability. We, therefore, do not use protected operators. Note that this means that the optimisation of coefficients can fail (e.g., if an optimisation step for $\sqrt{\theta_i x}$ leads to $\theta_i x < 0$). Regarding the operators in O that are not differentiable everywhere, we make use of Pytorch's [41] approximations (e.g., $\frac{\delta \max(x,0)}{\delta x}$ for $x = 0$ is 0).

We experiment with a limit of 10 and 100 evaluations of the Jacobian, i.e., the gradient vector of all elements of the (mini-)batch. A small evaluation limit is chosen because, typically, only a small evaluation budget per individual can be afforded when gradient-based optimisation is combined with GP (e.g., [7] and [45] use 3 evaluations). To evaluate the performance, we consider two aspects. First, we consider the accuracy gain in terms of MSE reduction between the ground truth and perturbed symbolic expression, normalised on the starting MSE:

$$Accuracy\ gain = \frac{MSE_{\text{start}} - \min(MSE_{\text{end}}, MSE_{\text{start}})}{MSE_{\text{start}}}. \quad (1)$$

The normalisation of the MSE allows us to compare different symbolic expressions on the same footing. Gradient-based optimisation can fail due to divergence ($MSE_{end} > MSE_{start}$), or stepping outside of the symbolic expressions' domain (e.g., by producing a `nan` or `inf` output). Symbolic expressions resulting in `nan` or `inf` are considered to have an accuracy gain of zero and to have *failed* to optimise. Since certain configurations can lead to higher accuracy gains but also higher failure rates (and vice versa), we analyse performance in terms of the trade-off between these two objectives. A selection of configurations that lie on the non-dominated front will then be used in experiments in a dynamic setting. All experiments are executed on AMD EPYC ROME 7282 32-core 3.2GHz processors.

Optimisers. We experiment with various optimisers. Optimisers that use momentum may be more robust to the choice of `lr`: we experiment with SGD+momentum with the momentum parameter set to 0.9, Adam [25], which is popular in DL [44], LM [33, 36], and BFGS [55]. The implementations of LM and BFGS in the SciPy library [52] do not require setting a learning rate. For the others, we consider an `lr` of 0.5, 0.01, and 0.001. LM and BFGS are "pseudo"-second-order optimisers in that they iteratively approximate the Hessian. This type of optimiser is not usually used for deep neural networks because the number of parameters is typically too large and results in infeasible runtimes. However, for the relatively small number of coefficients in SR expressions, they are viable.

Note that whereas the first-order methods can calculate one Jacobian matrix per batch, in order for the pseudo-second-order methods to work, multiple Jacobian calculations are needed per batch. This makes it hard to compare all methods on an equal footing when using a Jacobian evaluation budget. For example, BFGS may use all available budget for one single batch. To make the comparison fair, a larger Jacobian evaluation budget of 100 evaluations is also experimented with, so that BFGS and LM can process more batches.

Batch size. Typically the full training batch is used in SR when using gradient-based optimisation [7, 45]. In DL, mini-batches are used for several reasons. One is due to memory constraints. In our experiments, the difference in memory consumption and computation speed is negligible for the first-order optimisation methods, but for LM; the batch size does influence the execution speed, see Figure 2. Another reason why mini-batches are used in DL is that their use may lead to better generalisability, due to optimisation using mini-batches finding wide minima in the optimisation landscape more often [24, 53, 56]. It is unknown whether this might also happen for small symbolic expressions. We experiment with batch sizes of 32, 256, and 1000 (full batch).

Clipping. Operators such as *Pow* can induce extremely large gradients. Clipping the value of the gradient can help to prevent numerical instability and divergence [40]. Here, we consider the following options: *Clip Value*—we clip gradients between −1 and +1; *Clip Norm*—we divide gradients by their norm, as proposed in [40]; *Clip Operator*—we implement clipping at each intermediate computation that takes place in the backpropagation chain.

3.2 Results static experiments

A selection of the results of the static experiments can be observed in Figure 1, which shows where configurations lie compared to the best-obtained results, i.e., the non-dominated configurations with respect to the aforementioned objectives. Results for perturbations with $\mu = 1, \sigma = 1$, and with 100 evaluations, can be found in Appendix A.

Optimisers. We observe that in general pseudo-second-order methods perform well on all perturbations in both the training and validation set, as both LM and BFGS are located on every non-dominated front. Configurations of SGD+momentum and Adam is located on the non-dominated front due to their low failure rates, however, their training accuracy gains are limited. The accuracy gain of Adam is close to that of BFGS or LM only when the perturbance is small (i.e., $\mu = 0.1$ and $\sigma = 0.01$), but comes with higher failure rates. A similar pattern holds for 100 evaluations (see Appendix A).

Batch size. We find that the smallest batch size (32) leads to limited gain in training accuracy. With 10 evaluations this can be expected, as with a mini-batch of 32 and 1,000 training observations, *at most* 32% of the data is observed (BFGS and LM use several evaluations per batch). With 100 evaluations, BFGS, with a batch size of 256, is located on all non-dominated fronts. Interestingly, also in the validation plots, BFGS with a batch size of 256 is located on every non-dominated front (both 10 and 100 evaluations, all noise levels). This may be due to a batch size of 256 striking the right balance between observing enough data, and finding wider basins of attraction around the minima (compared to full batch), which are believed to improve generalisation in DL.

Although full batch LM is located on each training non-dominated front for both 10 and 100 evaluations, we observe that full batch evaluations are expensive for LM. Figure 2, inspired from [26], shows how runtime increases with respect to the batch size. As expected, the pseudo-second-order optimisers, i.e., BFGS and LM, take more time compared to first-order ones. Furthermore, LM becomes particularly expensive with the largest batch size we consider.

Clipping. In general, clipping reduces failure rates (especially for LM and Adam), but may lower accuracy gains (e.g., for BFGS): the variant of BFGS that uses no clipping appears in 4 out of 6 training fronts (including the 100 evaluation cases). Within the BFGS method, gradients of subsequent timesteps are subtracted from each other. This term could become zero, e.g., when both gradients are clipped to 1, or much smaller when using *Clip Norm*, causing BFGS's estimate of the inverse Hessian to worsen.

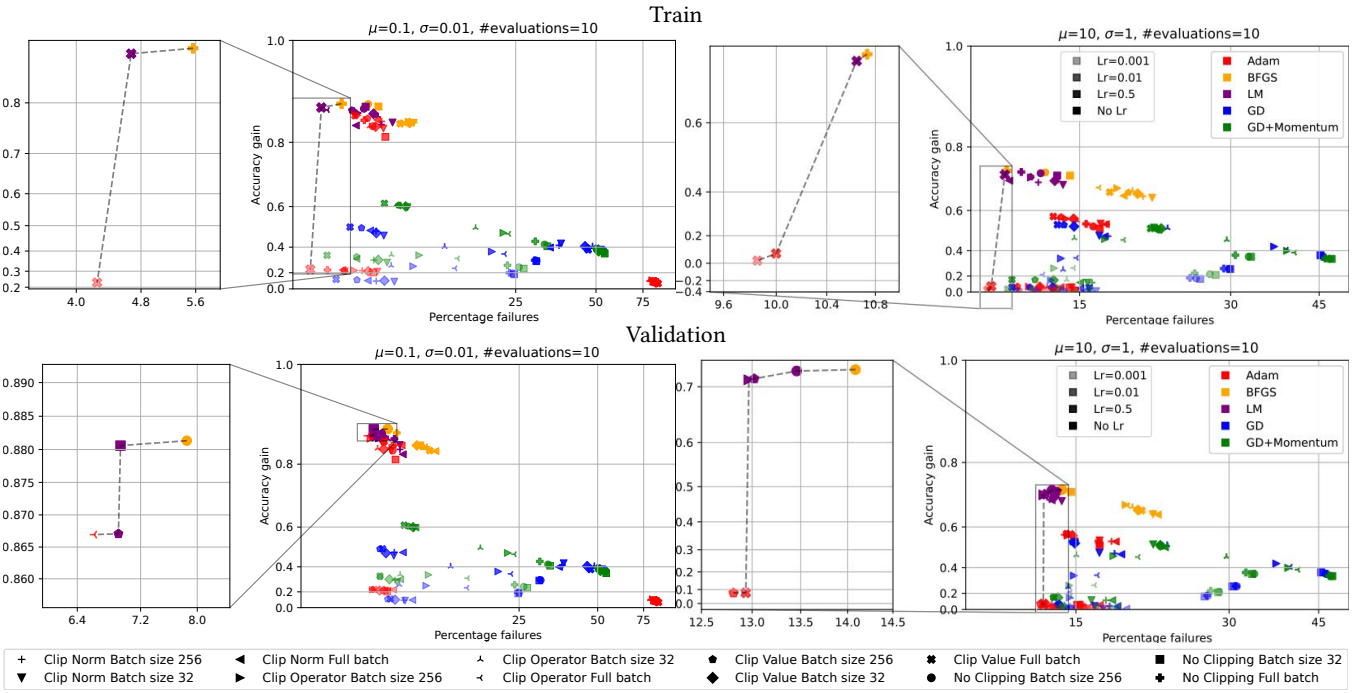

Figure 1: Average accuracy gain vs. percentage of failures on a logarithmic scale. The non-dominated front is indicated by a dashed line. Zoomed-in plots of the configurations on the non-dominated front are found to the left of the main plots with a grey rectangle indicating the zoomed-in area. The magnitude of `lr` corresponds with the opacity of the colour.

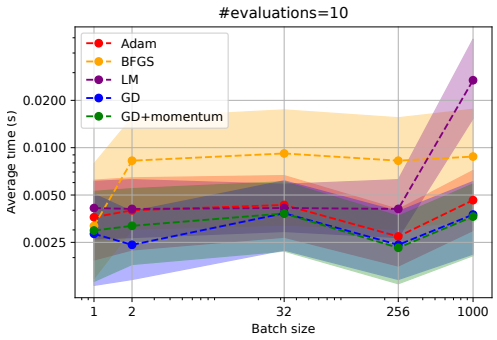

Figure 2: Evaluation speed per batch size, with $\mu = 10$, `lr` $= 0.001$, and no clipping. The scale of both axes is logarithmic. Lines are median values over the 10,000 trees of the experiment described in Sec. 3.1, and shaded areas indicate interquartile ranges.

3.3 Setup dynamic experiments

We now bring promising settings for gradient-based optimisation to GP, to assess whether the previous findings apply to one of the well-known methods for SR. In particular, we consider the following selection of options for gradient-based optimisation, which is located on the non-dominated validation fronts (see Figure 1 and Appendix A):

(1) Adam with *Clip Value*, using full batch and `lr` $= 0.01$,
(2) BFGS without clipping, using a batch size of 256,
(3) LM with *Clip Value*, using a batch size of 256.

From hereon, for brevity, we refer to these simply by Adam, BFGS, and LM, respectively. As baselines, we consider GP without any form of coefficient optimisation (from now on, *pure GP*), and GP with a simple coefficient mutation step (*GP with CM*) from [16, 50]:

$$\theta_i \leftarrow \theta_i + \tau|\theta_i| \cdot \mathcal{N}(0,1). \tag{2}$$

We choose $\tau = 0.25$, the default value from the library used for our experiments (omitted for double-blind review).

We use a typical implementation of GP, see Algorithm 1, with traditional settings [39] (see Table 1). We remark that in pure GP new (random) coefficient values can be sampled through structural

Table 1: General settings for the dynamic set of experiments.

Parameter	Setting
Population size	1000
Crossover rate	0.9 subtree swap
Mutation rate	0.1 subtree mutation, 0.1 node mutation
Tree initialisation	Ramped half & half (depth from 2 to 6)
Max tree size	31 nodes
Terminal set	Equal chance of features and constants
Constant initialisation	$\sim U(-10, 10)$
Selection	Parents ∪ offspring, tournaments of 7
Fitness function	MSE with linear scaling

mutations (from $U(-10, +10)$, see Table 1). We experiment with 5 UCI datasets [2] (see Table 2 for more details), with 75% train–25% validation random splits. Runs are repeated 30 times and are terminated at six hours. In preliminary experiments, we tested the use of linear scaling [22] (explained below), z-scaling (i.e., data normalisation by subtraction of the mean and division by the standard deviation), and the combination of linear and z-scaling, as advised in [10, 38]. We experimentally found that the combination leads to the best performance, confirming prior work.

Algorithm 1 Implementation of one GP generation including optimisation.

```
offspring ← ∅
for individual ∈ population do
    o ← clone(individual)
    o ← undergoVariation(o)  # crossover or mutation
    if optimiseCoefficients then
        o ← gradientBasedOptimisation(o) or coefficientMutation(o)
    end if
    offspring ← offspring ∪ {o}
end for
population ← tournamentSelection(offspring ∪ population)
```

Linear scaling simplifies the search for ϕ by analytically calculating an intercept a and slope b that make the expression invariant to translation and scaling. This is realised by:

$$\begin{aligned} \phi_{\theta LS}(x) &= a + b \cdot \phi_\theta(x), \\ b &= \frac{\sum_{i=1}^{n} \left(\phi_\theta(x_i) - \overline{\phi_\theta(x)}\right)(y_i - \overline{y})}{\sum_{i=1}^{n} \left(\phi_\theta(x_i) - \overline{\phi_\theta(x)}\right)^2}, \\ a &= \overline{y} - b \cdot \overline{\phi_\theta(x)}. \end{aligned} \tag{3}$$

Linear scaling is normally applied (including here) at fitness evaluation time and is relatively inexpensive ($O(n)$) [22]. In our work, the linear scaling slope and intercept are kept fixed during gradient-based optimisation (instead, e.g., [26] uses gradient-based optimisation also upon a and b).

Table 2: General dataset information. Datasets are sorted in terms of the number of samples, since these influence evaluation time. UCI datasets can be downloaded on the UCI repository website. The SciPy dataset can be found in the SciPy dataset loading utility. Mean and variance of y are reported as context for Eq. (4).

Dataset (source)	#Samples	#Features	Mean y	Variance y
Diabetes (SciPy)	442	10	152.1	77.0
Boston (UCI)	506	13	22.5	9.2
Concrete (UCI)	1030	8	35.8	16.7
Airfoil (UCI)	1503	5	124.8	6.9
Tower (UCI)	4999	25	342.1	87.8

3.4 Results dynamic experiments

We present the results in two ways: results obtained (1) with an equal number of generations and (2) with an equal time budget. For the former, we consider 100 generations, which are reached by all configurations. For the latter, we consider six hours. Pros and cons for both choices are discussed in Sec. 4.

Generation-based comparison. Figure 3 (two left-most columns) shows the training and validation performance per generation for the best-found expression in terms of coefficient of determination

$$R^2 = 1 - \frac{MSE(y, \phi_\theta(x))}{Var(y)}, \tag{4}$$

equivalent to minimising the MSE but maximisation is sought and 1.0 represents the perfect fit. We find that LM and BFGS have a comparatively fast rate of convergence, both reaching a high R^2 at the 100^{th} generation cut-off. Table 3 shows that LM significantly outperforms the GP and coefficient mutation baselines in 4/5 training sets. Furthermore, BFGS and in particular LM also generalise well to the validation set in 3/5 sets. We also observe that the maximum generation that is actually reached by LM and BFGS within six hours of runtime is approximately five times smaller than the other methods. This suggests that the success of the second-order methods is due to superior coefficient optimisation and not due to structural optimisation.

To gain a deeper understanding of the effect of gradient-based optimisation, Figure 4 (left) shows the percentage of offspring that are better in terms of training accuracy than their respective parent per generation, for one dataset. We choose Concrete as it has the median number of observations; we find similar patterns for the other datasets, see Figure 7. It can be seen that the percentage of offspring better than their respective parents is higher for gradient-based methods compared to the baseline for the majority of generations. This emphasises the importance of good coefficient optimisation in the initial generations. From Figure 4 (middle) it can be seen that the number of unique expressions (in terms of structure excluding differences in coefficients) in the population drops off roughly at the same rate for all methods. This indicates that the higher success rate of the gradient-based methods does not impact the rate of convergence.

Time-based comparison. From Figure 3 (two right-most columns) it can be seen that under a time-based comparison, coefficient mutation is a strong competitor to gradient-based methods. Coefficient mutation converges to good expressions quicker than second-order optimisers (BFGS and LM) and, at validation time, these expressions tend to generalise well. In fact, even baseline GP (i.e., no coefficient optimisation) generally performs rather well. Hence, these time-based results indicate that, given a same-time budget, having more focus on structure optimisation and less powerful but much faster coefficient optimisation can lead to decent results.

When considering statistical significance (Table 3), we find that second-order methods (LM in particular) perform well on the Tower and Airfoil datasets and significantly outperform the baseline but not coefficient mutation. Yet, overall, pseudo-second-order methods remain a suitable choice as they are not outperformed.

Figure 3: Median best-found training R^2 and respective validation R^2 during GP with different options for parameter optimisation. In the generation-based plots, the dotted vertical line indicates 100 generations, and coloured dots represent the maximum generation reached under the six hours time budget.

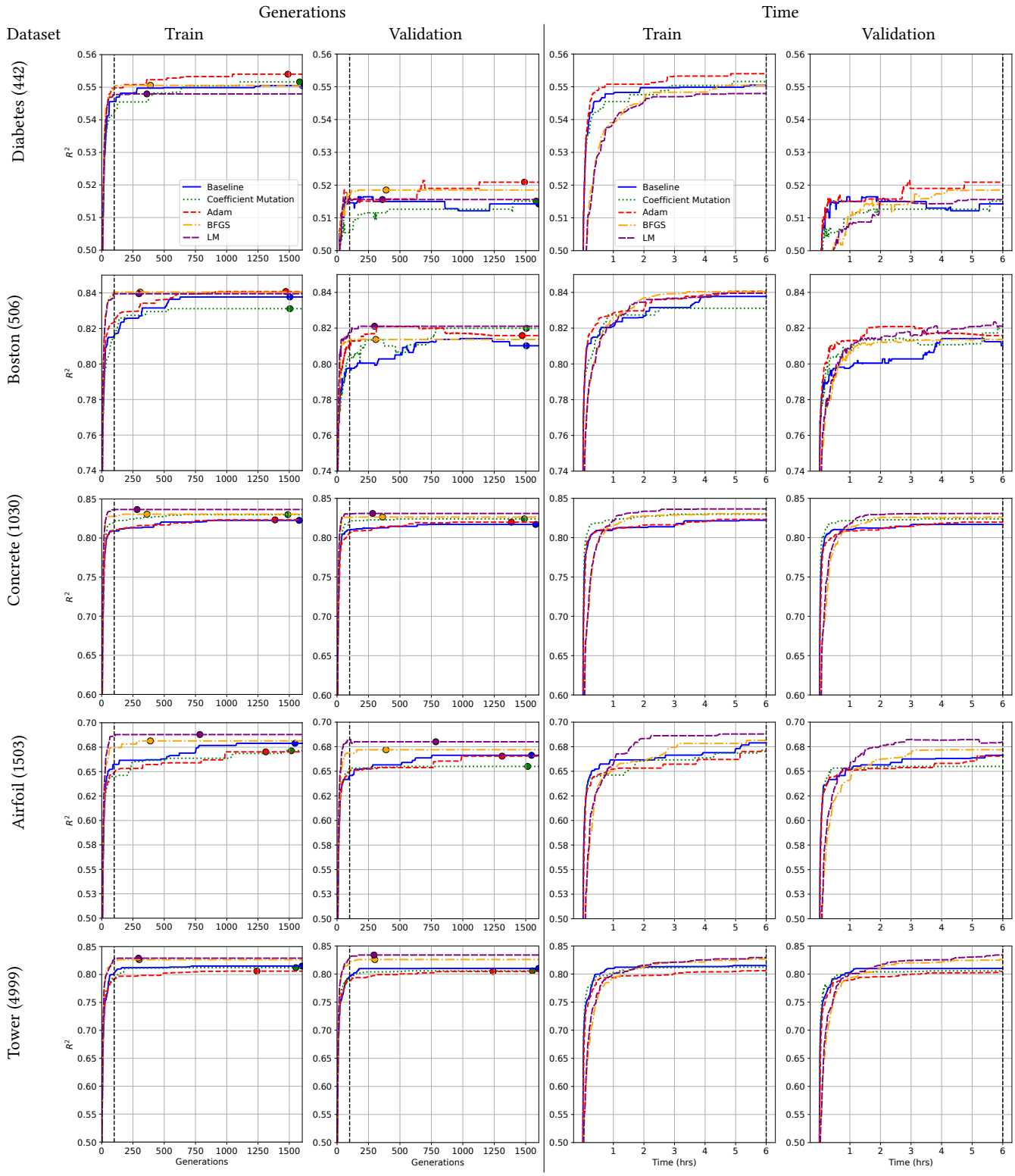

Table 3: Significance of validation results at 100 generations or 6hr of runtime. Kruskal-Wallis is used as omnibus test (Omn) and Wilcoxon signed-rank is used as post-hoc test for pairwise comparisons, both with $p < 0.05$ and Holm-Bonferroni correction. B=Baseline, CM=Coefficient Mutation.

	Train				Validation			
	100 generations		6hrs		100 generations		6hrs	
Dataset	Omn	Post-hoc	Omn	Posthoc	Omn	Post-hoc	Omn	Post-hoc
Diabetes	×	-	×	-	×	-	×	-
Boston	✓	LM, BFGS ≻ B, CM LM ≻ Adam	×	-	✓	BFGS ≻ CM	×	-
Concrete	✓	LM ≻ B, CM, Adam BFGS ≻ B, Adam	✓	LM ≻ B, CM	✓	LM, BFGS, CM ≻ B	✓	LM ≻ B
Airfoil	✓	LM ≻ B, CM, Adam	×	-	✓	LM ≻ B, CM, Adam	✓	LM ≻ B
Tower	✓	LM, BFGS ≻ B, CM, Adam	✓	LM, BFGS ≻ Adam	✓	LM, BFGS ≻ B, CM	✓	LM, BFGS ≻ Adam

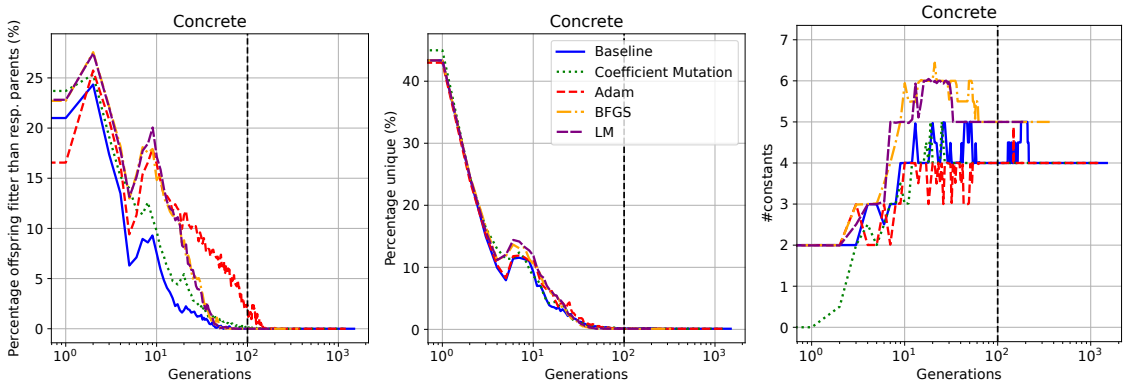

Figure 4: Left—Coefficient optimisation leads to a higher percentage of successful offspring. Middle—When comparing uniqueness (exact tree match), no substantial differences are found. Right—Second-order methods use more constants than other methods. Lines are median values over 10 runs, shaded areas indicate interquartile ranges.

Differences in discovered expressions. In [29, 45] it was found that using optimisation during GP influences the types of expressions that are discovered. Similarly, we find that the choice of optimiser can influence the number of coefficients that are used. BFGS and LM find end-of-run (six hours time limit) symbolic expressions with more coefficients than other techniques for all datasets as can be observed in Table 4. A similar pattern arises when the number of generations is capped at 100. From Figure 4 (right) it can be seen that the pseudo-second-order methods use more coefficients early on in the evolutionary process. This comes at the cost of using fewer input features or operators as all methods result in expressions that are close to the maximum number of allowed terms (31).

4 DISCUSSION

We find a reasonable level of transfer of the results from the static experiments (Sec. 3.1) to the dynamic ones (Sec. 3.3). The pseudo-second-order methods tend to outperform the other approaches when considering a generation-based comparison (in the static experiments, we also used generations and not runtime).

It is important to note that while we selected promising configurations on the basis of accuracy gain and the percentage of failures, there are also other trade-offs that could be made, e.g., taking the number of optimisation steps into account. Observing that the dynamic configuration using Adam did reasonably well in the dynamic experiments while not reaching a high accuracy gain in the static experiments indicates that other qualities besides accuracy gain may work well in a dynamic setting.

Comparing results with an equal number of generations is particularly meaningful when coefficient optimisation time is negligible with respect to structure optimisation time. For example, this may be the case in approaches that use probabilistic models or neural networks to capture and sample salient structure patterns (see, e.g., [19, 34, 54]). Moreover, generation-based comparisons are common in literature on gradient-based optimisation in GP [10, 45]. For classic GP, however, crossover, mutation, and selection operations take a negligible amount of time compared to fitness or gradient evaluations, thus a time-based comparison may be more sensible. Still, a limitation of a time-based comparison is that it depends on the specific hardware and code implementation. We used Python, and specifically SciPy for LM and BFGS, and Pytorch for the other optimisers. More performant implementations may exist, e.g., the implementation of GP with LM from [6] reaches excellent results in the recent SR benchmark SRBench [30].

Table 4: Median ± interquartile range of expression term use in end-of-run elite expressions resulting from the dynamic experiment.

Dataset	Optimiser	#total	#coefficients	#features
Diabetes	Baseline	31.0 ± 0.0	2.0 ± 2.0	11.0 ± 2.2
	Coeff. Mutation	31.0 ± 0.0	2.0 ± 2.0	12.0 ± 2.0
	Adam	31.0 ± 1.0	2.0 ± 1.0	12.0 ± 1.0
	BFGS	31.0 ± 0.0	4.0 ± 2.0	10.0 ± 2.0
	LM	31.0 ± 1.0	4.0 ± 1.0	10.0 ± 1.0
Boston	Baseline	31.0 ± 0.0	4.0 ± 2.2	10.0 ± 3.0
	Coeff. Mutation	31.0 ± 1.0	5.0 ± 1.0	8.0 ± 3.0
	Adam	31.0 ± 1.0	4.5 ± 2.2	10.0 ± 2.2
	BFGS	31.0 ± 0.2	6.0 ± 2.2	8.0 ± 2.0
	LM	31.0 ± 1.0	6.0 ± 2.0	8.0 ± 1.2
Concrete	Baseline	31.0 ± 0.0	4.0 ± 2.0	9.0 ± 1.2
	Coeff. Mutation	31.0 ± 0.0	4.0 ± 2.0	10.0 ± 1.2
	Adam	31.0 ± 0.0	3.5 ± 3.0	9.0 ± 2.0
	BFGS	31.0 ± 1.0	5.0 ± 3.0	8.0 ± 2.0
	LM	31.0 ± 1.0	5.0 ± 2.0	8.5 ± 1.0
Airfoil	Baseline	31.0 ± 1.0	2.5 ± 2.0	11.0 ± 2.0
	Coeff. Mutation	31.0 ± 0.0	4.0 ± 2.0	11.0 ± 2.0
	Adam	31.0 ± 0.2	3.0 ± 2.2	11.0 ± 2.2
	BFGS	31.0 ± 1.0	5.0 ± 2.0	9.0 ± 3.0
	LM	31.0 ± 1.0	5.0 ± 2.0	9.0 ± 2.0
Tower	Baseline	31.0 ± 0.0	3.0 ± 2.0	9.0 ± 3.0
	Coeff. Mutation	31.0 ± 0.0	4.0 ± 2.2	9.5 ± 3.0
	Adam	31.0 ± 1.0	4.0 ± 3.2	9.0 ± 2.0
	BFGS	31.0 ± 1.0	6.0 ± 2.0	8.0 ± 2.0
	LM	31.0 ± 1.0	6.0 ± 2.2	9.0 ± 3.0

The rates for GP's variation operators (Table 1) were inspired by work in [10]. For completeness, we performed additional experiments (see Appendix B) with equal variation rates for subtree crossover, subtree mutation, and operator mutation (one-third each). Results from this experiment show less pronounced differences among the configurations, with LM only significantly outperforming coefficient mutation in 1/5 datasets (see Figure 6 and Table 5 in the appendix). This strongly indicates that the settings pertaining to GP have a large influence on the effectiveness of coefficient optimisation. Nevertheless, we do not find cases where second-order optimisers are outperformed by other methods.

In several works only arithmetic operators are chosen while certain expressions may require non-linear operators [7, 10, 43, 45]. Non-linear operators may be approximated with linear operators, e.g., using Taylor's method, but this typically requires many operators and are hard to interpret. We choose to use a more extensive operator set, both to increase the expressivity of expressions confined to a small number of nodes while remaining interpretable, and to experiment with optimising non-linear operators. Additionally, [29] finds that using many non-linear operators with a small expression size reduces the prevalence of ill-conditioned Jacobians, which benefits optimisation using LM.

In this work, we focused on small symbolic expressions (at most totalling 31 terms) because we wish to obtain expressions that may be interpretable. The combination of linear scaling and z-scaling works better for some configurations. However, this may come at the cost of interpretability. The combination of linear and z-scaling adds operators to the expression and furthermore changes the feature scale that practitioners may be used to.

While we chose to optimise the coefficients of all offspring, there exists some literature on optimising only part of the population, to be efficient [7, 27]. This is another dimension that is worth exploring in future work. Indeed, from the graphs in Figure 3, we can see that the pseudo-second-order methods improve quickly, but also run for fewer generations. The GP baseline keeps improving after the pseudo-second-order methods have reached their maximum number of generations, this could mean that the structure evolved by the GP (combined with coefficient optimisation) may be suboptimal and that it may be beneficial, e.g., to optimise the coefficients at a later stage when a more appropriate structure is found.

Another dimension worth exploring further is the option to stop coefficient optimisation early. Here, we always optimised up to 10 evaluations in the dynamic experiments, accepting bad steps, i.e., steps that worsen the fitness (as common in DL). Instead, [7, 45] stop and revert the coefficients at the first occurrence of a bad step. A window of consecutive bad steps could be considered as an additional hyper-parameter. Similarly, more research could be done on re-initialising coefficient values, which is not done in GP literature but is common when optimising the architecture of deep neural networks [11].

Here, we considered classic GP for simplicity, but state-of-the-art SR algorithms such as GP-GOMEA [48, 49] could potentially also benefit from integrating gradient-based coefficient optimisation, and Operon [6], which uses LM, could benefit from mini-batching and gradient-clipping. Similarly, it would be interesting to assess the transferability of our results from the static experiment to other algorithms than GP, such as large recurrent neural networks [42] and pre-trained transformers [5, 20].

5 CONCLUSION

We studied the application of gradient-based optimisation with techniques used in DL, both in isolation on random symbolic expressions with a static structure, and in combination with GP. We found that mini-batching in combination with second-order optimisation, namely, BFGS or LM with gradient-clipping, results in effective optimisation of SR expressions. When used within GP, these methods delivered solid performance in a generation-based comparison, typically outperforming pure GP and GP with coefficient mutation. However, when the comparison is framed in terms of time, the performance gap decreased substantially because pure GP and GP with coefficient mutation are far less expensive to execute. In fact, coefficient mutation performed on par with second-order optimisation. In conclusion, gradient-based optimisation can work well for SR, particularly when adopting second-order optimisers. However, as this form of coefficient optimisation is relatively expensive, there is only a clear advantage in choosing it over simpler methods (e.g., coefficient mutation) if the cost of searching for structure strongly dominates the cost for optimising coefficients.

ACKNOWLEDGMENTS

This research is part of the research programme Open Competition Domain Science-KLEIN with project number OCENW.KLEIN.111, which is financed by the Dutch Research Council (NWO). We further thank the Maurits en Anna de Kock Foundation for financing a high-performance computing system.

REFERENCES

[1] 117th US Congress. 2022. Algorithmic accountability act. https://www.congress.gov/bill/117th-congress/house-bill/6580/

[2] Arthur Asuncion and David Newman. 2007. UCI machine learning repository.

[3] Vladan Babovic and Maarten Keijzer. 2000. Genetic programming as a model induction engine. *Journal of Hydroinformatics* 2, 1 (2000), 35–60.

[4] Deaglan J Bartlett, Harry Desmond, and Pedro G Ferreira. 2022. Exhaustive Symbolic Regression. *arXiv preprint arXiv:2211.11461* (2022).

[5] Luca Biggio, Tommaso Bendinelli, Alexander Neitz, Aurelien Lucchi, and Giambattista Parascandolo. 2021. Neural symbolic regression that scales. In *International Conference on Machine Learning*. PMLR, 936–945.

[6] Bogdan Burlacu, Gabriel Kronberger, and Michael Kommenda. 2020. Operon C++ an efficient genetic programming framework for symbolic regression. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion*. 1562–1570.

[7] Qi Chen, Bing Xue, and Mengjie Zhang. 2015. Generalisation and domain adaptation in GP with gradient descent for symbolic regression. In *2015 IEEE Congress on Evolutionary Computation (CEC)*. IEEE, 1137–1144.

[8] Miles Cranmer, Alvaro Sanchez Gonzalez, Peter Battaglia, Rui Xu, Kyle Cranmer, David Spergel, and Shirley Ho. 2020. Discovering symbolic models from deep learning with inductive biases. *Advances in Neural Information Processing Systems* 33 (2020), 17429–17442.

[9] Fabrício Olivetti de França. 2018. A greedy search tree heuristic for symbolic regression. *Information Sciences* 442 (2018), 18–32.

[10] Grant Dick, Caitlin A Owen, and Peter A Whigham. 2020. Feature standardisation and coefficient optimisation for effective symbolic regression. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*. 306–314.

[11] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. 2019. Neural architecture search: A survey. *The Journal of Machine Learning Research* 20, 1 (2019), 1997–2017.

[12] European Commission. 2021. Artificial intelligence act. https://artificialintelligenceact.eu/

[13] Matthew Evett and Thomas Fernandez. 1998. Numeric Mutation Improves the Discovery of Numeric Constants in Genetic Programming, Genetic Programming 1998: Proceedings of the Third Annual Conference, University of Wisconsin, Madison, Wisconsin, USA. *University of Wisconsin, Madi-son, Wisconsin, USA* (1998), 66–71.

[14] Xavier Glorot and Yoshua Bengio. 2010. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*. JMLR Workshop and Conference Proceedings, 249–256.

[15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer vision and Pattern Recognition*. 770–778.

[16] Daniel Hein, Steffen Udluft, and Thomas A Runkler. 2018. Interpretable policies for reinforcement learning by genetic programming. *Engineering Applications of Artificial Intelligence* 76 (2018), 158–169.

[17] Sergey Ioffe and Christian Szegedy. 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*. pmlr, 448–456.

[18] Dario Izzo, Francesco Biscani, and Alessio Mereta. 2017. Differentiable genetic programming. In *European conference on genetic programming*. Springer, 35–51.

[19] Ying Jin, Weilin Fu, Jian Kang, Jiadong Guo, and Jian Guo. 2019. Bayesian symbolic regression. *arXiv preprint arXiv:1910.08892* (2019).

[20] Pierre-Alexandre Kamienny, Stéphane d'Ascoli, Guillaume Lample, and François Charton. 2022. End-to-end symbolic regression with transformers. *arXiv preprint arXiv:2204.10532* (2022).

[21] Lukas Kammerer, Gabriel Kronberger, Bogdan Burlacu, Stephan M Winkler, Michael Kommenda, and Michael Affenzeller. 2020. Symbolic regression by exhaustive search: reducing the search space using syntactical constraints and efficient semantic structure deduplication. In *Genetic Programming Theory and Practice XVII*. Springer, 79–99.

[22] Maarten Keijzer. 2003. Improving symbolic regression with interval arithmetic and linear scaling. In *European Conference on Genetic Programming*. Springer, 70–82.

[23] Liron Simon Keren, Alex Liberzon, and Teddy Lazebnik. 2023. A computational framework for physics-informed symbolic regression with straightforward integration of domain knowledge. *Scientific Reports* 13 (2023). Issue 1.

[24] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. 2016. On large-batch training for deep learning: Generalization gap and sharp minima. *arXiv preprint arXiv:1609.04836* (2016).

[25] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[26] Michael Kommenda, Bogdan Burlacu, Gabriel Kronberger, and Michael Affenzeller. 2020. Parameter identification for symbolic regression using nonlinear least squares. *Genetic Programming and Evolvable Machines* 21, 3 (2020), 471–501.

[27] Michael Kommenda, Gabriel Kronberger, Stephan Winkler, Michael Affenzeller, and Stefan Wagner. 2013. Effects of constant optimization by nonlinear least squares minimization in symbolic regression. In *Proceedings of the 15th annual Conference companion on Genetic and Evolutionary Computation*. 1121–1128.

[28] John R Koza. 1992. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Vol. 1. MIT press Cambridge, MA, USA.

[29] Gabriel Kronberger. 2022. Local Optimization Often is Ill-conditioned in Genetic Programming for Symbolic Regression. *arXiv preprint arXiv:2209.00942* (2022).

[30] William La Cava, Patryk Orzechowski, Bogdan Burlacu, Fabrício Olivetti de França, Marco Virgolin, Ying Jin, Michael Kommenda, and Jason H Moore. 2021. Contemporary symbolic regression methods and their relative performance. *arXiv preprint arXiv:2107.14351* (2021).

[31] Guillaume Lample and François Charton. 2019. Deep learning for symbolic mathematics. *arXiv preprint arXiv:1912.01412* (2019).

[32] Mikel Landajuela, Chak Lee, Jiachen Yang, Ruben Glatt, Claudio P. Santiago, Ignacio Aravena, Terrell N. Mundhenk, Garrett Mulcahy, and Brenden K. Petersen. 2022. A Unified Framework for Deep Symbolic Regression. In *Advances in Neural Information Processing Systems*, Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho (Eds.). https://openreview.net/forum?id=2FNnBhwJsHK

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

[34] Paweł Liskowski, Krzysztof Krawiec, Nihat Engin Toklu, and Jerry Swan. 2020. Program synthesis as latent continuous optimization: Evolutionary search in neural embeddings. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*. 359–367.

[35] Francesco Marchetti and Edmondo Minisci. 2020. A hybrid neural network-genetic programming intelligent control approach. In *International Conference on Bioinspired Methods and Their Applications*. Springer, 240–254.

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

[37] Vinod Nair and Geoffrey E. Hinton. 2010. Rectified Linear Units Improve Restricted Boltzmann Machines. In *Proceedings of the 27th International Conference on International Conference on Machine Learning* (Haifa, Israel) *(ICML'10)*. Omnipress, Madison, WI, USA, 807–814.

[38] Caitlin A Owen, Grant Dick, and Peter A Whigham. 2018. Feature standardisation in symbolic regression. In *Australasian Joint Conference on Artificial Intelligence*. Springer, 565–576.

[39] Michael O'Neill. 2009. Riccardo Poli, William B. Langdon, Nicholas F. McPhee: a field guide to genetic programming.

[40] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. 2013. On the difficulty of training recurrent neural networks. In *International conference on machine learning*. PMLR, 1310–1318.

[41] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. 2017. Automatic differentiation in PyTorch. (2017).

[42] Brenden K Petersen, Mikel Landajuela Larma, T Nathan Mundhenk, Claudio P Santiago, Soo K Kim, and Joanne T Kim. 2019. Deep symbolic regression: Recovering mathematical expressions from data via risk-seeking policy gradients. *arXiv preprint arXiv:1912.04871* (2019).

[43] Gloria Pietropolli, Luca Manzoni, Alessia Paoletti, and Mauro Castelli. 2022. Combining geometric semantic gp with gradient-descent optimization. In *European Conference on Genetic Programming (Part of EvoStar)*. Springer, 19–33.

[44] Sebastian Ruder. 2016. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747* (2016).

[45] Alexander Topchy, William F Punch, et al. 2001. Faster genetic programming based on local gradient search of numeric leaf values. In *Proceedings of the genetic and evolutionary computation conference (GECCO-2001)*, Vol. 155162. Morgan Kaufmann San Francisco, CA.

[46] Silviu-Marian Udrescu and Max Tegmark. 2020. AI Feynman: A physics-inspired method for symbolic regression. *Science Advances* 6, 16 (2020), eaay2631.

[47] Marco Virgolin, Tanja Alderliesten, Arjan Bel, Cees Witteveen, and Peter AN Bosman. 2018. Symbolic regression and feature construction with GP-GOMEA applied to radiotherapy dose reconstruction of childhood cancer survivors. In *Proceedings of the Genetic and Evolutionary Computation Conference*. 1395–1402.

[48] Marco Virgolin, Tanja Alderliesten, Cees Witteveen, and Peter AN Bosman. 2017. Scalable genetic programming by gene-pool optimal mixing and input-space entropy-based building-block learning. In *Proceedings of the Genetic and Evolutionary Computation Conference*. 1041–1048.

[49] Marco Virgolin, Tanja Alderliesten, Cees Witteveen, and Peter AN Bosman. 2021. Improving model-based genetic programming for symbolic regression of small expressions. *Evolutionary computation* 29, 2 (2021), 211–237.

[50] Marco Virgolin and Peter AN Bosman. 2022. Coefficient mutation in the gene-pool optimal mixing evolutionary algorithm for symbolic regression. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. 2289–2297.

[51] Marco Virgolin and Solon P Pissis. 2022. Symbolic Regression is NP-hard. *arXiv preprint arXiv:2207.01018* (2022).

[52] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. 2020. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods* 17 (2020), 261–272. https://doi.org/10.1038/s41592-019-0686-2

[53] Xingyu Wang, Sewoong Oh, and Chang-Han Rhee. 2021. Eliminating sharp minima from SGD with truncated heavy-tailed noise. *arXiv preprint arXiv:2102.04297* (2021).

[54] David Wittenberg, Franz Rothlauf, and Dirk Schweim. 2020. DAE-GP: denoising autoencoder LSTM networks as probabilistic models in estimation of distribution genetic programming. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*. 1037–1045.

[55] Stephen Wright, Jorge Nocedal, et al. 1999. Numerical optimization. *Springer Science* 35, 67-68 (1999), 7.

[56] Lei Wu, Chao Ma, et al. 2018. How SGD selects the global minima in over-parameterized learning: A dynamical stability perspective. *Advances in Neural Information Processing Systems* 31 (2018).