Stochastic optimization approaches to learning concise representations

Anonymous Author(s)
Affiliation
Address
email

Abstract

We propose and study a method for learning interpretable features via stochastic optimization of feature architectures. Features are represented as multi-type expression trees using a set of activation functions common in neural networks in addition to other elementary functions. Continuous features are trained via gradient descent, and the performance of features in ML models is used to weight the rate of change among subcomponents of representations. The search process maintains an archive of representations with accuracy-complexity trade-offs to assist in generalization and interpretation. We compare several stochastic optimization approaches within this framework. We benchmark these variants on many real-world regression problems in comparison to other machine learning approaches. The best results across methods are obtained by search methods that directly optimize the accuracy-complexity tradeoff in order to find simple architectures that generalize well.

1 Introduction

The performance of a machine learning (ML) model depends primarily on the data representation used in training [3], and for this reason the representational capacity of neural networks (NN) is considered a central factor in their success in many applications [10]. To date, there does not seem to be a consensus on how the architecture should be designed. As problems grow in complexity, the networks proposed to solve these problems grow as well, leading to an intractable design space. One design approach is to tune network architectures through network hyperparameters using grid search or randomized search [4] with cross validation. Often some combination of hyperparameter tuning and manual design by expertise/intuition is done [10]. Another potential solution is to use population-based stochastic optimization (SO) methods (also known as metaheuristics [23]), which is the focus of this paper. In SO, several candidate solutions are evaluated and varied over several iterations, and heuristics are used to probabilistically select and update the candidate networks until the population produces a desirable architecture. The approach has been used at least since the early 2000s [35] for NN design, with several recent applications [29, 14, 28].

In practice, the adequacy of the architecture, i.e. model *form*, is often dependent on conflicting objectives. For example, interpretability may be a central concern, because many researchers in the scientific community rely on ML models not only to provide predictions that match data from various processes, but to provide insight into the nature of the processes themselves. Approaches to interpretability can be roughly grouped into semantic and syntactic approaches. Semantic approaches encompass methods that attempt to illucidate the behavior of a model under various input conditions as a way of explanation (e.g. [30]). Syntactic methods instead focus on the development of concise models that offer insight by virtue of their simplicity, in a similar vein to models built from first-

Submitted to 32nd Conference on Neural Information Processing Systems (NIPS 2018). Do not distribute.

principles. The method proposed here belongs to the latter group: our goal is to discover the simplest description of a process whose predictions generalize as well as possible.

Good representations should also disentangle the factors of variation [3] in the data, in order to ease model interpretation. Related to the idea of disentanglement is the idea of functional modularity; i.e., the idea that sub-functions of the network are responsible for encapsulated behaviors that model a sub-process of the task. In this sense, stochasic methods such as evolutionary computation (EC) appear well-motivated, as they are premised on the identification and propagation of building blocks of solutions [11]. Experiments with EC applied to networks suggest it pressures networks to be modular [12, 15]. Although the identification functional building blocks of solutions sounds ideal, we have no way of known *a priori* whether a given problem/dataset will admit the identification of building blocks of solutions via heuristic search [25]. Our goal in this paper is thus to empirically assess the performance of several SO approaches in a system designed to produce intelligible representations from NN building blocks for regression.

In Section 2, we introduce a new method for optimizing representations which we call the feature engineering archiving tool. The purpose of this method is to optimize an archive of representations that characterize the trade-off between conciseness and accuracy among representations. It is a wrapper-based feature synthesis technique that optimizes features in the loop with a user-given ML method. FEAT contributes a few non-standard SO techniques designed to improve the generalizability and legibility of the method. First, it incorporates the initial ML model into the population during training. Second, it fits an ML model to each candidate representation to a) assess its fitness and b) provide feedback to the variation process at a more granular level. Third, it maintains an archive of Pareto optimal trade-offs between complexity and accuracy. This archive is validated on a hold-out test set at the end of training to conduct model/representation selection.

We discuss related work in more detail in Section 3. In section 4 and 5, we describe and conduct an experiment that benchmarks five SO methods within this framework against other ML methods. Future work based on this analysis is discussed in Section 6.

2 Methods

We are interested in the task of regression, for which the goal is to build a predictive model $\hat{y}(\mathbf{x})$ using N paired examples $\mathcal{T} = \{(\mathbf{x}_i, y_i)\}_{i=1}^N$. The regression model $\hat{y}(\mathbf{x})$ associates the inputs $\mathbf{x} \in \mathbb{R}^d$ with a real-valued output $y \in \mathbb{R}$. The goal of feature engineering / representation learning is to find a new representation of $\mathbf{x}$ via a m-dimensional feature mapping $\phi(\mathbf{x}) : \mathbb{R}^d \rightarrow \mathbb{R}^m$, such that a model $\hat{y}(\phi(\mathbf{x}))$ outperforms the model $\hat{y}(\mathbf{x})$.

When applying a NN to a traditional ML task like regression or classification, a fixed NN architecture $\phi(\mathbf{x}, \theta)$, parameterized by θ, is chosen and used to fit a model

$$\hat{y} = \phi(\mathbf{x}, \theta)^T \beta \tag{1}$$

In this case $\phi = [\phi_1 \ \ldots \ \phi_m]^T$ is a NN with m nodes in the hidden layer and a linear output layer with coefficients $\beta = [\beta_1 \ \ldots \ \beta_m]^T$ produces the model output. The problem is then cast as a parameter optimization problem of the form

$$\theta^* = \arg\min_{\theta} \sum_{i}^{N} L(y_i, \hat{y}_i, \theta, \beta) \tag{2}$$

where $\hat{\theta}$ is chosen to minimize a cost function L, with global optimum θ^*. (L may depend on θ and β in the case of regularization.) In the case of SO, the optimization problem is made more general to include the form of ϕ in the minimization of L, leading to the formulation

$$\phi^*(\mathbf{x}, \theta^*) = \arg\min_{\phi \in \mathbb{S}, \theta} \sum_{i}^{N} L(y_i, \hat{y}_i, \phi, \theta, \beta) \tag{3}$$

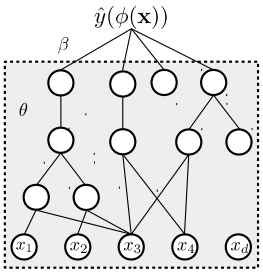

Figure 1: Example model representation in FEAT.

Table 1: Functions and terminals used to develop representations.

Continuous functions	{+, -, *, /, ^{2}, ^{3}, $\sqrt{}$, sin, cos, exp, log, exponent, logit, tanh, gauss, relu}
Boolean functions	and, or, not, xor, =, <, <=, >, >=
Terminals	$\{\mathbf{x}\}$

where $\mathbb{S}$ is the space of possible functions defined by the procedure, and ϕ^* is the true structure of the process underlying the data. The assumption of SO approaches such as evolutionary computation (EC) and simulated annealing (SA) is that candidate solutions in $\mathbb{S}$ that are similar to each other, i.e. reachable in few mutations, are more likely to have similar costs than candidate solutions that are far apart (an assumption known as locality). In these cases, $\mathbb{S}$ can be effectively searched by maintaining and updating a population of candidate representations that perform well. Multi-objective SO methods extend Eq. 3 to minimizing additional cost functions [6].

2.1 FEAT

FEAT uses a typical μ + λ evolutionary updating scheme, where $\mu = \lambda = P$. The main loop iterates through four steps: 1) selection, 2) variation, 3) survival, and 4) archiving. A population of potential representations, $\mathcal{N} = \{n_1 \ \ldots \ n_P\}$, where n is an "individual" in the population. Each representation is used to fit an ML model. For the experiments in this paper, we use linear ridge regression. Individuals are evaluated using an initial forward pass, followed by weight updating. The weights of the differentiable programs are updated using stochastic gradient descent with backpropagation. As we describe in the forthcoming sections, the ML model's weights are then used to bias variation.

2.1.1 Representation

FEAT represents features by constructing syntax trees from elementary boolean- and continuous-valued functions and literals, much like in symbolic regression (SR) [19]. The encoding used in FEAT differs in two important ways. First, in contrast to typical SR, each individual n is a set of such trees, the output of which is interpreted as a candidate representation, i.e. $\phi(\mathbf{x}) = [\phi_1 \ldots \phi_m]$ for an individual with m trees. Each program is constructed from a list of instructions is shown in Table 1. The second difference from traditional SR is that the weights of differentiable nodes are encoded in the edges between programs, as shown in Figure 1. The instructions include typical activation functions used in NN, e.g. tanh, sigmoid, logit and relu nodes, and the weights are encoded in a very similar manner. Indeed, a fully connected feedforward NN is representable in FEAT's representation. However, due to the tree-based construction process and the use of elementary arithmetic operators (+,-,*,/), features in FEAT are biased to be thinly connected, with the goal of improved legibility.

2.1.2 Initialization

FEAT begins by fitting an ML model to the original data. For the examples in this paper, we use a linear model $\hat{y} = \mathbf{x}^T\beta$ trained using linear ridge regression. The values of β are used to set probabilities of sampling each predictor in $\mathbf{x}$ for use in a representation, according to Eqn. 1. This initial representation, $\phi = \mathbf{x}$, is introduced into the original population, along with $P - 1$ randomly generated representations.

2.1.3 Variation

During variation, the representations are perturbed using a set of mutation and crossover methods. FEAT chooses among 6 variation operators that are as follows. *Point mutation* changes a node type to one with matching output type and arity. *Insert mutation* replaces a node with a randomly generated depth 1 subtree. *Delete mutation* with equal probability, removes a feature or a replaces a sub-program with an input node. *Insert/Delete dimension* adds or removes a new feature. *Sub-tree crossover* replaces a sub-tree from one parent with the sub-tree of another parent. *Dimension crossover* swaps two features between parents. The exact probabilities of each variation operator will affect the performance of the algorithm. For the purposes of our study, we use each operator with uniform probability.

Feedback The use of an ML model to assess the fitness of each representation can be used to provide information about the elements of the representation that should be changed. In particular, we assume that programs in the representation with small coefficients are the best candidates for mutation and crossover. With this in mind, let n be an m-dimensional candidate representation with associated coefficients $\beta(n) \in \mathbb{R}^m$. The probability of mutation for tree i in n is denoted $PM_i(n)$, and defined as follows:

$$\begin{aligned} \tilde{\beta}_i(n) &= |\beta_i| / \sum_i^m |\beta_i| \\ s_i(n) &= \exp(1 - \tilde{\beta}_i) / \sum_i^m \exp(1 - \beta_i) \\ PM_i(n) &= f s_i(n) + (1 - f)\frac{1}{m} \end{aligned} \tag{4}$$

The normalized coefficient magnitudes $\tilde{\beta} \in [0, 1]$ are used to define softmax-normalized probabilities, s in Eqn. 4. The smaller the coefficient, the higher the probability of mutation. The parameter f is used to control the amount of feedback used to weight the probabilities; $\frac{1}{m}$ in this case represents uniform probability. Among nodes in tree m, mutation occurs with uniform probability. This weighting could be extended for differentiable nodes by weighting the within-tree probabilities by the magnitude of the weights associated with each node. However we expect this would yield diminishing returns.

2.1.4 Selection and Survival

The selection step selects among P parents those representations that will be used to generate offspring. Following variation, the population consists of $2P$ representations of parents and offspring. The survival step is used to reduce the population back to size P, at which point the generation is finished. We study five configurations of selection adopted from literature that are described below.

ϵ-lexicase selection This selection method (abbreviated Lex) was proposed for regression problems [22, 5] as an adaption of lexicase selection, a technique used primarily for discrete error problems. Under ϵ-lexicase selection, parents are chosen via population filtering using randomized orders of training samples with the ϵ threshold defined relative with respect to the sample loss among the pool. This filtering strategy scales probability of selection for an individual based on the difficulty of the training cases the individual performs well on. Lex has shown strong performance among SR methods in recent tests, motivating our interest in studying it [26]. The survival step for Lex just preserves offspring plus the best individual in the population.

Non-dominated sorting genetic algorithm 2 NSGA-2 is a popular selection and survival strategy for multi-objective optimization [6] that applies preference for selection based on Pareto dominance relations. One individual (n_i) is said to *dominate* another (n_j) if, for all objectives, n_i performs at least as well as n_j, and for at least one objective, n_i strictly outperforms n_j. The Pareto *front* is the set of individuals in $\mathcal{N}$ that are non-dominated in the population. The Pareto front consists of solutions that are optimal trade-offs between objectives found during search. We define two objectives in

our study: the first corresponds to the mean squared loss function for individual n, and the second corresponds to the complexity of the representation. There are many ways to define complexity of an expression; one could simply look at the number of operations in a representation, or look at the behavioral complexity of the representation using a polynomial order [38]. The one we use, which is similar to that used by Kommenda et. al. [17], is to assign a complexity weight to each operator (see Table 1), with higher weights assigned to operators considered more complex. If the weight of operator o is c_o, then the complexity of an expression tree beginning at node o is defined recursively as

$$C(o) = c_o * \sum_{a=1}^{k} C(a) \tag{5}$$

where o has k arguments, and $C(a)$ is the complexity of argument a. The complexity of a representation is then defined as the sum of the complexities of its output nodes. The goal of defining complexity in such a way is to discourage deep sub-expressions within complex nodes, which are often hard to interpret. It's important to note that the choice of operator weights is bound to be subjective, since we lack an objective notion of interpretability. For this reason, although we use Eqn. 5 to drive search, our experimental comparisons with other algorithms rely on the parameter counts of the final models for benchmarking interpretability of different methods.

NSGA-2 also relies on a behavioral diversity measure to measure the spread of solutions in objective space. Each individual is assigned a crowding distance measure, which is a measure of its distance to its two adjacent neighbors in objective space. Under NSGA-2, parent selection is conducted according to Pareto tournaments of size 2. In tournament selection, two parents are randomly drawn from the population and compared. If one dominates the other, it is chosen; crowding distance is used to break ties. The survival step of NSGA-2 begins by sorting the population according to their Pareto front *ranking*, which is a measure of their distance to the Pareto front. Individuals are added to the surviving population in order of their ranking. If a rank level does not completely fit, individuals of that rank are sorted by crowding distance and added in that order until P individuals are chosen for survival.

Simulated annealing Simulated annealing (SimAnn) is a non-evolutionary technique that instead models the optimization process on the metallurgical process of annealing. In our implementation, offspring compete with their parents; in the case of multiple parents, offspring compete with the progam with which they share more nodes. The probability of an offspring replacing its parent in the population is given by the equation

$$P_{sel}(n_o|n_p, t) = \exp\left(\frac{F(n_p) - F(n_o)}{t}\right) \tag{6}$$

The probability of offspring replacing its parent is a function of its fitness, in our case the mean squared loss of the candidate model. In Eqn. 6, t is a scheduling parameter that controls the rate of "cooling", i.e. the rate at which steps in the search space that are worse are tolerated by the update rule. In accordance with [16], we use an exponential schedule for t, defined as $t_g = (0.9)^g t_0$, where g is the current generation and $t0$ is the starting temperature. $t0$ is set to 10 in our experiments.

Random search We compare the selection and survival methods to random search, in which no assumptions are made about the structure of the search space. To conduct random search, we randomly sample $\mathbb{S}$ using the initialization procedure described above. Since FEAT begins with a linear model of the process, random search will produce a representation at least as good as this initial model on the internal validation set.

2.1.5 Archiving

During optimization, FEAT maintains a separate population that acts as an archive. The archive maintains a Pareto front according to minimum loss and complexity (Eqn 5). At the end of optimization, the archive is tested on a small hold-out validation set. The indivdiual with the lowest validation loss is the final selected model. Maintaining this archive helps protect against overfitting resulting

from overly complex / high capacity representations, and also can be interepreted directly to help understand the process being modelled.

3 Related Work

Symbolic Regression FEAT is primarily motivated by SR approaches to feature engineering[20, 1, 21] that use EC to search for possible representations and couple with an ML model to handle the parametrization of the representations. SR methods have been successful in developing intelligible models of physical systems[31]. FEAT differs from these methods in the following ways. A key challenge in SR is understanding functional modularity within representations/programs that can be exploited for search. FEAT is designed with the insight that ML weights can be leveraged during variation to promote functional building blocks, an exploit not used in previous methods. Second, FEAT uses multiple type representations, and thus can learn continuos and rule-based features within a single representation, unlike previous methods. This is made possible using a stack-based encoding with strongly-typed operators. Finally, FEAT incorporates two elements of NN learning to improve its representational capacity: activation functions commonly used in NN and edge-based encoding of weights. Traditionally, SR operates with standard mathematical operators, and treats constants as leaves in the expression trees rather than edge weights. An exception is MRGP [1], which encodes weights at each node but updates them via Lasso instead of using gradient descent with backpropagation. SR methods have also been paired with various parameter learning strategies, including those based on backpropagation [36, 18, 13]. It should be noted that non-stochastic methods for SR exist, such as mixed integer non-linear programming, which has been demonstrated for small search spaces [2].

Neuroevolution The idea to evolve NN architectures is well established in literature, and is known as neuroevolution. Popular methods of neuroevolution include neuroevolution of augmenting topologies (NEAT[35] and Hyper-NEAT[34]), and compositional pattern producing networks [33] . Traditionally these approaches eschew the parameter learing step common in other NN paradigms, although others have developed integrations [7]. These methods do not have interepretability as a core focus, and thus do not attempt to use multi-objective methods to update the networks. In addition, they have been developed predominantly for other task domains such as robotics and control [9]. A review of these methods is available [8]. Recently, neuroevolution has been scaled to large networks for image classification [29, 28].

Dropout FEAT also shares a motivational relationship to dropout, a popular method of NN regularization. Dropout is an approach that may improve interpretability of models by considering competing subsets of networks during training[32]. The authors were motivated in part by the evolutionary process of sexual recombination, which is a dominant form of genotype variation found in nature for unclear reasons. One reason the authors entertain is the ability of crossover between models to assert selective pressure for genes to be robust to different environmental contexts, since crossover may introduce large changes to neighboring genes. This pressure is also rewards genes with modular functionality since genes that are close together are more likely to be shared together and must perform a similar function in a new organism.

4 Experiment

We test FEAT on the task of real-world regression. Our goals with the experiment are to 1) compare different SO methods for selection and survival; 2) compare SO methods to hyperparameter optimization of feedforward NNs; 3) characterize the complexity of the solutions; and 4) measure the "entanglement" of the feature spaces produced by FEAT. For the real-world regression datasets, we use 88 real-world regression datasets available from OpenML [37]. The datasets are characterized in terms of number of features and sample sizes in Figure 2. We use the standardized versions of the datasets available in the Penn Machine Learning Benchmark repository [24]. We compare the FEAT configurations to the multi-layer perceptron (MLP) implementation from scikit-learn [27]. The hyperparameters are shown in Table 3. We also compare to ElasticNet, for which the $l1$ to $l2$ ratio is optimized. Code to reproduce these experiments is available online.[1]

[1] `http://github.com/anon-gecco/feat_nips/`

Table 2: Configurations tested for FEAT. Multiple values indicate use in hyperparameter tuning.

Setting	Values
SO Method	NSGA2, Lex, Lex-NSGA2, SimAnn, Random
Population size	100
Generations	100
Max depth	10
Max dimensionality	50
Fitness	R2
Parameter learning	{hillclimbing, SGD}
Learning rate	0.1
Iterations / individual / generation	{1,10,100}
Crossover rate	0.5

Table 3: Neural Network configurations for the sklearn-MLP.

Setting	Value
Optimizer	{LBFGS, Adam}
Hidden Layers	{1,3,6}
Neurons per layer	10-100
Learning rate	(initial) {1e-3, 1e-2, 1e-1}
Regularization	L_2, α = {1e-4, 1e-2, 1e-1}
Iterations	10000
Early Stopping	True

For each method, we use grid search to tune the hyperparameters with 10-fold cross validation. We use the R^2 score for assessing performance. In our results we report the cross validated scores for each method using its best hyperparameters. For each dataset we rank the algorithms according to their median score. For comparing complexity, we count the number of parameters in the final model produced by each method for each trial on each dataset. To quantify the "entanglement" of the feature spaces produced by FEAT, we report the mean of the correlation coefficient of the best representation ϕ, normalized between 0 and 1. The correlation coefficient matrix represents the pairwise covariance of feature columns. We shift these between zero and 1 and compute the mean.

5 Results

The score statistics for each method are shown in Fig. 4, and mean rankings across datasets are reported in Fig. 3. Over all, Lex-NSGA2 produces the best predictive performance across datasets, followed by Lex, NSGA2, and SimAnn. The EC-based implementations of FEAT outperform the cross-validated MLP in terms of rankings across datasets (p <9.1e-11)[2]. Random search and ElasticNet produce the worst results, but perform similarly due to the initialization with a linear model. The models produced by FEAT tend to use fewer parameters compared to MLP, as shown in Fig. 5. NSGA2 produces the most concise representations by this metric, followed closely by Lex-NSGA2. In terms of run-time, ElasticNet performs the fastest, followed by MLP and then FEAT, which is (expectedly) the slowest. Finally, we look at the correlation structure of the features produced by FEAT in Fig. 7. NSGA2 and LexNSGA2 tend to produce less correlated features than Lex or SimAnn, although all four methods show higher representation correlations than in the raw data. Hence feature entanglement remains an issue with this system.

6 Discussion and Conclusion

The results suggest that FEAT is able to learn simpler models than traditional traditional MLP approaches that improve generalization on many small to medium-scale regression problems. Among SO methods tested, Lex and Lex-NSGA2 perform the best, both in terms of generalization and solution size. Future work should consider the issue of representation disentanglement in more depth.

[2]Pairwise Wilcox signed rank test with Bonferroni correction

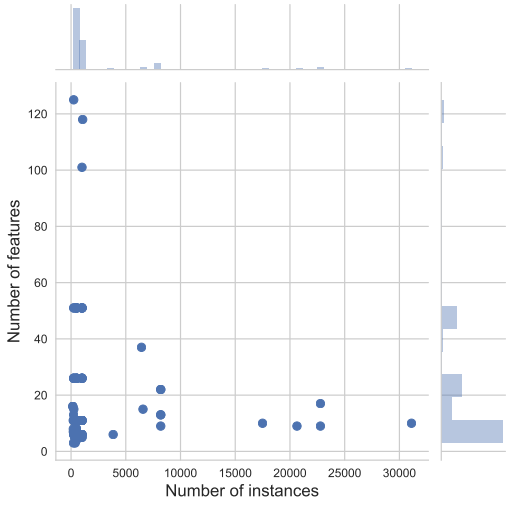

Figure 2: Properties of the regression datasets.

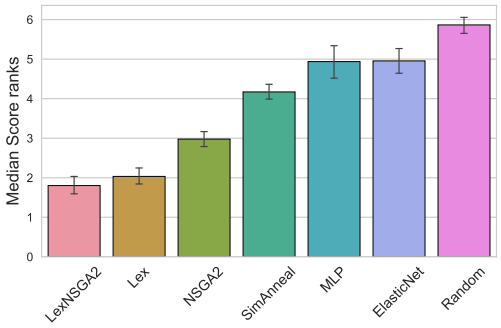

Figure 3: Rankings of each method according to median 10-fold CV R^2 performance, averaged over the datasets.

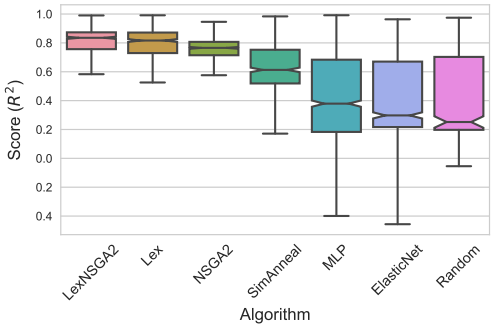

Figure 4: Mean 10-fold CV R^2 performance for various SO methods in comparison to other ML methods, across the benchmark problems.

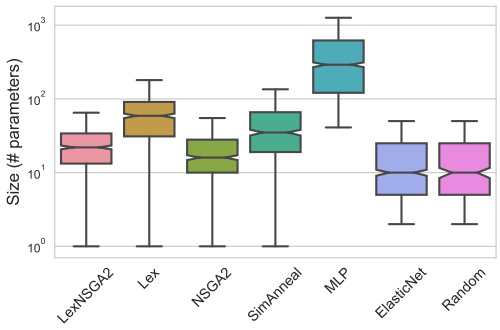

Figure 5: Size comparisons of the final models in terms of number of parameters.

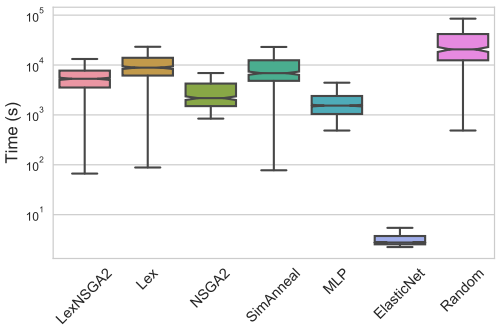

Figure 6: Wall-clock runtime for each method in seconds.

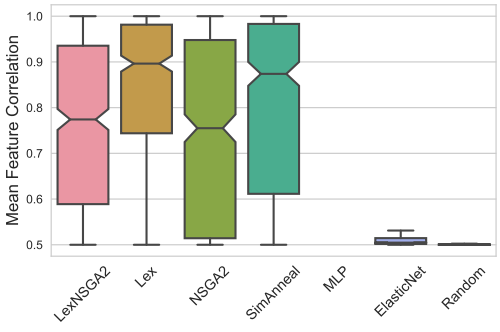

Figure 7: Mean correlation between engineered features for different SO methods compared to the correlations in the original data (ElasticNet).

References

[1] Arnaldo, I., Krawiec, K., and O'Reilly, U.-M. (2014). Multiple regression genetic programming. In *Proceedings of the 2014 conference on Genetic and evolutionary computation*, pages 879–886. ACM Press.

[2] Austel, V., Dash, S., Gunluk, O., Horesh, L., Liberti, L., Nannicini, G., and Schieber, B. (2017). Globally Optimal Symbolic Regression. *arXiv:1710.10720 [stat]*. arXiv: 1710.10720.

[3] Bengio, Y., Courville, A., and Vincent, P. (2013). Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1798–1828.

[4] Bergstra, J. and Bengio, Y. (2012). Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(Feb):281–305.

[5] Cava, W. L., Helmuth, T., Spector, L., and Moore, J. H. (2018). A probabilistic and multi-objective analysis of lexicase selection and -lexicase selection. *Evolutionary Computation*, pages 1–28.

[6] Deb, K., Agrawal, S., Pratap, A., and Meyarivan, T. (2000). A Fast Elitist Non-dominated Sorting Genetic Algorithm for Multi-objective Optimization: NSGA-II. In Schoenauer, M., Deb, K., Rudolph, G., Yao, X., Lutton, E., Merelo, J. J., and Schwefel, H.-P., editors, *Parallel Problem Solving from Nature PPSN VI*, volume 1917, pages 849–858. Springer Berlin Heidelberg, Berlin, Heidelberg.

[7] Fernando, C., Banarse, D., Reynolds, M., Besse, F., Pfau, D., Jaderberg, M., Lanctot, M., and Wierstra, D. (2016). Convolution by Evolution: Differentiable Pattern Producing Networks. *arXiv:1606.02580 [cs]*. arXiv: 1606.02580.

[8] Floreano, D., Dürr, P., and Mattiussi, C. (2008). Neuroevolution: from architectures to learning. *Evolutionary Intelligence*, 1(1):47–62.

[9] Gomez, F., Schmidhuber, J., and Miikkulainen, R. (2006). Efficient non-linear control through neuroevolution. In *ECML*, volume 4212, pages 654–662. Springer.

[10] Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press.

[11] Holland, J. H. (1975). Adaptation in natural and artificial systems. An introductory analysis with application to biology, control, and artificial intelligence. *Ann Arbor, MI: University of Michigan Press*, pages 439–444.

[12] Huizinga, J., Clune, J., and Mouret, J.-B. (2014). Evolving neural networks that are both modular and regular: HyperNEAT plus the connection cost technique. pages 697–704. ACM Press.

[13] Izzo, D., Biscani, F., and Mereta, A. (2017). Differentiable Genetic Programming. In *European Conference on Genetic Programming*, pages 35–51. Springer.

[14] Jaderberg, M., Dalibard, V., Osindero, S., Czarnecki, W. M., Donahue, J., Razavi, A., Vinyals, O., Green, T., Dunning, I., Simonyan, K., Fernando, C., and Kavukcuoglu, K. (2017). Population Based Training of Neural Networks. *arXiv:1711.09846 [cs]*. arXiv: 1711.09846.

[15] Kashtan, N. and Alon, U. (2005). Spontaneous evolution of modularity and network motifs. *Proceedings of the National Academy of Sciences*, 102(39):13773–13778.

[16] Kirkpatrick, S., Gelatt, C. D., and Vecchi, M. P. (1983). Optimization by simulated annealing. *science*, 220(4598):671–680.

[17] Kommenda, M., Kronberger, G., Affenzeller, M., Winkler, S. M., and Burlacu, B. (2015). Evolving Simple Symbolic Regression Models by Multi-objective Genetic Programming. In *Genetic Programming Theory and Practice*, volume XIV of *Genetic and Evolutionary Computation*. Springer, Ann Arbor, MI.

[18] Kommenda, M., Kronberger, G., Winkler, S., Affenzeller, M., and Wagner, S. (2013). Effects of constant optimization by nonlinear least squares minimization in symbolic regression. In Blum, C., Alba, E., Bartz-Beielstein, T., Loiacono, D., Luna, F., Mehnen, J., Ochoa, G., Preuss, M., Tantar, E., and Vanneschi, L., editors, *GECCO '13 Companion: Proceeding of the fifteenth annual conference companion on Genetic and evolutionary computation conference companion*, pages 1121–1128, Amsterdam, The Netherlands. ACM.

[19] Koza, J. R. (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, USA.

[20] Krawiec, K. (2002). Genetic programming-based construction of features for machine learning and knowledge discovery tasks. *Genetic Programming and Evolvable Machines*, 3(4):329–343.

[21] La Cava, W. and Moore, J. (2017). A General Feature Engineering Wrapper for Machine Learning Using \epsilon -Lexicase Survival. In *Genetic Programming*, pages 80–95. Springer, Cham.

[22] La Cava, W., Spector, L., and Danai, K. (2016). Epsilon-Lexicase Selection for Regression. In *Proceedings of the Genetic and Evolutionary Computation Conference 2016*, GECCO '16, pages 741–748, New York, NY, USA. ACM.

[23] Luke, S. (2013). *Essentials of Metaheuristics*. 2nd edition.

[24] Olson, R. S., La Cava, W., Orzechowski, P., Urbanowicz, R. J., and Moore, J. H. (2017). PMLB: A Large Benchmark Suite for Machine Learning Evaluation and Comparison. *BioData Mining*. arXiv preprint arXiv:1703.00512.

[25] Oppacher, U.-M. O. F. (2014). The troubling aspects of a building block hypothesis for genetic programming. *Foundations of Genetic Algorithms 1995 (FOGA 3)*, 3:73.

[26] Orzechowski, P., La Cava, W., and Moore, J. H. (2018). Where are we now? A large benchmark study of recent symbolic regression methods. *arXiv:1804.09331 [cs]*. arXiv: 1804.09331.

[27] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., and others (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12(Oct):2825–2830.

[28] Real, E. (2018). Using Evolutionary AutoML to Discover Neural Network Architectures.

[29] Real, E., Moore, S., Selle, A., Saxena, S., Suematsu, Y. L., Tan, J., Le, Q., and Kurakin, A. (2017). Large-Scale Evolution of Image Classifiers. *arXiv:1703.01041 [cs]*. arXiv: 1703.01041.

[30] Ribeiro, M. T., Singh, S., and Guestrin, C. (2016). Why should i trust you?: Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1135–1144. ACM.

[31] Schmidt, M. and Lipson, H. (2009). Distilling free-form natural laws from experimental data. *Science*, 324(5923):81–85.

[32] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958.

[33] Stanley, K. O. (2007). Compositional pattern producing networks: A novel abstraction of development. *Genetic programming and evolvable machines*, 8(2):131–162.

[34] Stanley, K. O., D'Ambrosio, D. B., and Gauci, J. (2009). A hypercube-based encoding for evolving large-scale neural networks. *Artificial life*, 15(2):185–212.

[35] Stanley, K. O. and Miikkulainen, R. (2002). Evolving neural networks through augmenting topologies. *Evolutionary computation*, 10(2):99–127.

[36] Topchy, A. and Punch, W. F. (2001). Faster genetic programming based on local gradient search of numeric leaf values. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, pages 155–162.

[37] Vanschoren, J., van Rijn, J. N., Bischl, B., and Torgo, L. (2014). OpenML: Networked Science in Machine Learning. *SIGKDD Explor. Newsl.*, 15(2):49–60.

[38] Vladislavleva, E., Smits, G., and den Hertog, D. (2009). Order of Nonlinearity as a Complexity Measure for Models Generated by Symbolic Regression via Pareto Genetic Programming. *IEEE Transactions on Evolutionary Computation*, 13(2):333–349.