GENO – Optimization for Machine Learning Made Fast and Easy

Sören Laue
Friedrich-Schiller-Universität Jena
&
Data Assessment Solutions GmbH Hannover
Germany
soeren.laue@uni-jena.de

Matthias Mitterreiter
Friedrich-Schiller-Universität Jena
Germany
matthias.mitterreiter@uni-jena.de

Joachim Giesen
Friedrich-Schiller-Universität Jena
Germany
joachim.giesen@uni-jena.de

Abstract

We introduce the optimization framework GENO (GENeric Optimization) that combines a modeling language with a generic solver. GENO generates a solver from the declarative specification of an optimization problem class. The framework is flexible enough to encompass most of the classical machine learning problems. We show on a wide variety of classical but also some recently suggested problems that the automatically generated solvers are (1) as efficient as well-engineered specialized solvers, (2) more efficient by a decent margin than recent state-of-the-art solvers, and (3) orders of magnitude more efficient than classical modeling language plus solver approaches.

1 Introduction

Optimization is at the core of machine learning and many other fields of applied research, for instance operations research, optimal control, and deep learning. The latter fields have embraced frameworks that combine a modeling language with only a few optimization solvers; interior point solvers in operations research and stochastic gradient descent (SGD) and variants thereof in deep learning frameworks like TensorFlow, PyTorch, or Caffe. That is in stark contrast to classical (i.e., non-deep) machine learning, where new problems are often accompanied by new optimization algorithms and their implementation. However, designing and implementing optimization algorithms is still a time-consuming and error-prone task.

The lack of an optimization framework for classical machine learning problems can be explained partially by the common belief, that any efficient solver needs to exploit problem specific structure. Here, we challenge this common belief.

We introduce GENO (GENeric Optimization), an optimization framework that allows to state optimization problems in an easy-to-read modeling language. From the specification an optimizer is automatically generated by using automatic differentiation on a symbolic level. The optimizer combines a quasi-Newton solver with an augmented Lagrangian approach for handling constraints.

Any generic modeling language plus solver approach frees the user from tedious implementation aspects and allows to focus on modeling aspects of the problem at hand. However, it is required

Workshop on Systems for ML at NeurIPS 2019. A full version of this paper appeared in [31].

that the solver is efficient and accurate. Contrary to common belief, we show here that the solvers generated by GENO are (1) as efficient as well-engineered, specialized solvers at the same or better accuracy, (2) more efficient by a decent margin than recent state-of-the-art solvers, and (3) orders of magnitude more efficient than classical modeling language plus solver approaches.

Related work. Classical machine learning is typically served by toolboxes like scikit-learn [40], Weka [18], and MLlib [36]. These toolboxes mainly serve as wrappers for a collection of well-engineered implementations of standard solvers like LIBSVM [9] for support vector machines or glmnet [19] for generalized linear models. A disadvantage of the toolbox approach is a lacking of flexibility. An only slightly changed model, for instance by adding a non-negativity constraint, might already be missing in the framework.

Modeling languages provide more flexibility since they allow to specify problems from large problem classes. Popular modeling languages for optimization are CVX [11, 23] for MATLAB and its Python extension CVXPY [3, 13], and JuMP [16] which is bound to Julia. In the operations research community AMPL [17] and GAMS [7] have been used for many years. All these languages take an instance of an optimization problem and transform it into some standard form of a linear program (LP), quadratic program (QP), second-order cone program (SOCP), or semi-definite program (SDP). The transformed problem is then addressed by solvers for the corresponding standard form. However, the transformation into standard form can be inefficient, because the formal representation in standard form can grow substantially with the problem size. This representational inefficiency directly translates into computational inefficiency.

The modeling language plus solver paradigm has been made deployable in the CVXGEN [35], QPgen [21], and OSQP [4] projects. In these projects code is generated for the specified problem class. However, the problem dimension and sometimes the underlying sparsity pattern of the data needs to be fixed. Thus, the size of the generated code still grows with a growing problem dimension. All these projects are targeted at embedded systems and are optimized for small or sparse problems. The underlying solvers are based on Newton-type methods that solve a Newton system of equations by direct methods. Solving these systems is efficient only for small problems or problems where the sparsity structure of the Hessian can be exploited in the Cholesky factorization. Neither condition is typically met in standard machine learning problems.

Deep learning frameworks like TensorFlow [1], PyTorch [39], or Caffe [26] are efficient and fairly flexible. However, they target only deep learning problems that are typically unconstrained problems that ask to optimize a separable sum of loss functions. Algorithmically, deep learning frameworks usually employ some form of stochastic gradient descent (SGD) [42], the rationale being that computing the full gradient is too slow and actually not necessary. A drawback of SGD-type algorithms is that they need careful parameter tuning of, for instance, the learning rate or, for accelerated SGD, the momentum. Parameter tuning is a time-consuming and often data-dependent task. A non-careful choice of these parameters can turn the algorithm slow or even cause it to diverge. Also, SGD type algorithms cannot handle constraints.

GENO, the framework that we present here, differs from the standard modeling language plus solver approach by a much tighter coupling of the language and the solver. GENO does not transform problem instances but whole problem classes, including constrained problems, into a very general standard form. Since the standard form is independent of any specific problem instance it does not grow for larger instances. GENO does not require the user to tune parameters and the generated code is highly efficient and often outperforms problem-specific, recently published state-of-the-art solvers by a good margin.

2 The GENO Pipeline

GENO features a modeling language and a solver that are tightly coupled. The modeling language allows to specify a whole class of optimization problems in terms of an objective function and constraints that are given as vectorized linear algebra expressions. Neither the objective function nor the constraints need to be differentiable. Non-differentiable problems are transformed into constrained, differentiable problems. A general purpose solver for constrained, differentiable problems is then instantiated with the objective function, the constraint functions and their respective gradients. The gradients are computed by the matrix and tensor calculus algorithm that has been recently published in [29, 30] and its extension [33]. The tight integration of the modeling language and the solver is possible only because of this recent progress in computing derivatives of vectorized linear algebra expressions.

Generating a solver takes only a few milliseconds. Once it has been generated the solver can be used like any hand-written solver for every instance of the specified problem class. An interface to the GENO framework can be found at `http://www.geno-project.org`.

2.1 Modeling Language

A GENO specification has four blocks, see Figure 1 for some examples: (1) Declaration of the problem parameters that can be of type *Matrix*, *Vector*, or *Scalar*, (2) declaration of one or more optimization variables that also can be of type *Matrix*, *Vector*, or *Scalar*, (3) specification of the objective function in a MATLAB-like syntax, and finally (4) specification of the constraints, also in a MATLAB-like syntax that supports the following operators and functions: `+, -, *, /, .*, ./,` `∧, .∧, log, exp, sin, cos, tanh, abs, norm1, norm2, sum, tr, det, inv`. The set of operators and functions can be expanded when needed. Matrices can be general dense matrices or sparse, symmetric, or positive semidefinite.

Note that in contrast to instance-based modeling languages like CVXPY no dimensions have to be specified. Also, the specified problems do not need to be convex. In the non-convex case, only a local optimal solution will be computed.

```
parameters
    Matrix A symmetric
variables
    Vector x
min
    -x'*A*x / (x'*x)
```

```
parameters
     Matrix A
     Vector b
variables
     Vector x
min
     norm2(A*x - b)^2
st
     x >= 0
```

```
parameters
     Matrix A
     Vector b
variables
     Vector x
min
     norm1(x)
st
     A*x == b
     sum(x) == 1
     x >= 0
```

Figure 1: A few optimization problems formulated in the GENO modeling language. The problem on the left is an unconstrained optimization problem that computes the Rayleigh quotient, the problem in the middle is the non-negative least squares problem, and the problem on the right shows an ℓ_1-norm minimization problem from compressed sensing where the signal is known to be an element from the unit simplex.

2.2 Generic Optimizer

At its core, GENO's generic optimizer is a solver for unconstrained, smooth optimization problems. Thus, we implemented the L-BFGS-B quasi-Newton algorithm [8, 44] that can also handle box constraints on the variables. It scales very well to problems involving millions of variables. This solver is then extended to handle also non-smooth and constrained problems using an Augmented Lagrangian approach [24, 41]. More details can be found in the full paper [32, 31]. Compared to projected gradient methods or to the alternating direction method of multipliers (ADMM) [6, 20, 22], the augmented Lagrangian approach only needs access to gradient information and hence, is more generic than ADMM-type methods that need access to proximal operators. Prox-operators are usually derived manually using standard duality theory and cannot be derived automatically. Hence, the generic optimizer can deal with problems of the following general form

$$\begin{array}{ll} \min\limits_{x} & f(x) \\ \text{s.t.} & h(x) = 0 \\ & g(x) \leq 0, \end{array} \tag{1}$$

where $x \in \mathbb{R}^n$, $f\colon \mathbb{R}^n \to \mathbb{R}$, $h\colon \mathbb{R}^n \to \mathbb{R}^m$, $g\colon \mathbb{R}^n \to \mathbb{R}^p$ are real-valued functions, and the equality and inequality constraints are understood component-wise.

3 Limitations

While GENO is very general and efficient, as we will demonstrate in the experimental Section 4, it also has some limitations that we discuss here. For small problems, i.e., problems with only a few dozen variables, Newton-type methods with a direct solver for the Newton system can be even faster. GENO also does not target deep learning applications, where gradients do not need to be computed fully but can be sampled.

Some problems can pose numerical problems, for instance problems containing an $\exp$ operator might cause an overflow/underflow. However, this is a problem that is faced by all frameworks. It is usually addressed by introducing special operators like *logsumexp*.

Furthermore, GENO does not perform sanity checks on the provided input. Any syntactically correct problem specification is accepted by GENO as a valid input. For example, $\log(\det(xx^\top))$, where x is a vector, is a valid expression. But the determinant of the outer product will always be zero and hence, taking the logarithm will fail. It lies within the responsibility of the user to make sure that expressions are mathematically valid.

4 Experiments

We conducted a number of experiments to show the wide applicability and efficiency of our approach. For the experiments we have chosen classical problems that come with established well-engineered solvers like logistic regression or elastic net regression, but also problems and algorithms that have been published at NeurIPS and ICML only within the last few years. The experiments cover smooth unconstrained problems as well as constrained, and non-smooth problems. To prevent a bias towards GENO, we always used the original code for the competing methods and followed the experimental setup in the papers where these methods have been introduced. We ran the experiments on standard data sets from the LIBSVM data set repository, and, in some cases, on synthetic data sets on which competing methods had been evaluated in the corresponding papers.

Specifically, our experiments cover the following problems and solvers: ℓ_1- and ℓ_2-regularized logistic regression, support vector machines, elastic net regression, non-negative least squares, symmetric non-negative matrix factorization, problems from non-convex optimization, and compressed sensing. Among other algorithms, we compared against a trust-region Newton method with conjugate gradient descent for solving the Newton system, sequential minimal optimization (SMO), dual coordinate descent, proximal methods including ADMM and variants thereof, interior point methods, accelerated and variance reduced variants of SGD, and Nesterov's optimal gradient descent. Please refer to the full version [32, 31] of this extended abstract for more details on the experiments, solvers, and GENO models.

Our test machine was equipped with an eight-core Intel Xeon CPU E5-2643 and 256GB RAM. As software environment we used Python 3.6, along with NumPy 1.16, SciPy 1.2, and scikit-learn 0.20. In some cases the original code of competing methods was written and run in MATLAB R2019. The solvers generated by GENO spent between 80% and 99% of their time on evaluating function values and gradients. Here, these evaluations essentially reduce to evaluating linear algebra expressions. Since all libraries are linked against the Intel MKL, running times of the GENO solvers are essentially the same in both environments, Python and MATLAB.

4.1 ℓ_2-regularized Logistic Regression

Logistic regression is probably the most popular linear, binary classification method. It is given by the following unconstrained optimization problem with a smooth objective function

$$\min_w \quad \tfrac{\lambda}{2}\left\|w\right\|_2^2 + \tfrac{1}{m}\textstyle\sum_i \log(\exp(-y_i X_i w)+1),$$

where $X \in \mathbb{R}^{m\times n}$ is a data matrix, $y \in \{-1,+1\}^m$ is a label vector, and $\lambda \in \mathbb{R}$ is the regularization parameter. Since it is a classical problem there exist many well-engineered solvers for ℓ_2-regularized logistic regression. The problem also serves as a testbed for new algorithms. We compared GENO to the parallel version of LIBLINEAR and a number of recently developed algorithms and their implementations, namely Point-SAGA [12], SDCA [43], and catalyst SDCA [34]). The latter algorithms implement some form of SGD. Thus their running time heavily depends on the values for the learning rate (step size) and the momentum parameter in the case of accelerated SGD. The best parameter setting often depends on the regularization parameter and the data set. We have used the code provided by [12] and the parameter settings therein.

For our experiments we set the regularization parameter $\lambda = 10^{-4}$ and used real world data sets that are commonly used in experiments involving logistic regression. GENO converges almost as rapidly as LIBLINEAR and outperforms any of the recently published solvers by a good margin, see Figure 2.

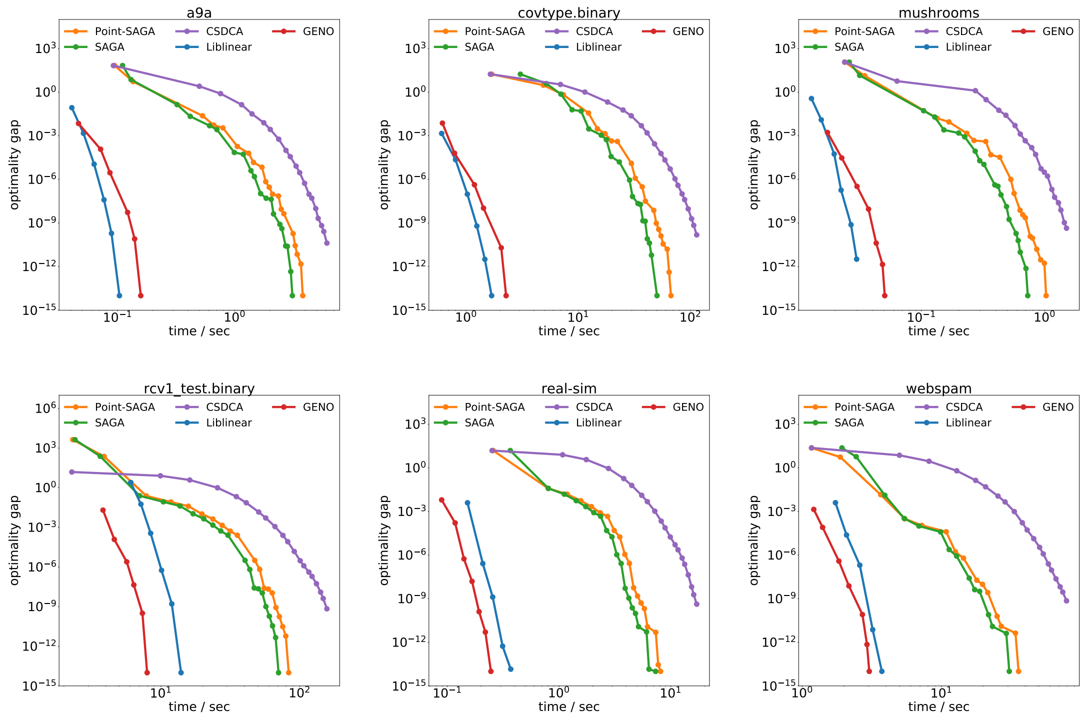

Figure 2: Running times for different solvers on the ℓ_2-regularized logistic regression problem.

We could not run other general purpose modeling language plus solver approaches on these data sets since these approaches are orders of magnitude slower than our approach. Hence, we compared GENO to CVXPY with both the ECOS [15] and the SCS solver [38] on substantially smaller data sets. The interface for CVXPY to Mosek [37] for exponential cones has not been implemented yet. As can be seen from Table 1, GENO is orders of magnitude faster.

Table 1: Running times in seconds for different general purpose solvers on small instances of the ℓ_2-regularized logistic regression problem. The approximation error is close to 10^{-6} for all solvers.

Solver	Data sets						
	heart	ionosphere	breast-cancer	australian	diabetes	a1a	a5a
GENO	0.005	0.013	0.004	0.014	0.006	0.023	0.062
ECOS	1.999	2.775	5.080	5.380	5.881	12.606	57.467
SCS	2.589	3.330	6.224	6.578	6.743	16.361	87.904

4.2 Symmetric Non-negative Matrix Factorization

Non-negative matrix factorization (NMF) and its many variants are standard methods for recommender systems [2] and topic modeling [5, 25]. It is known as symmetric NMF, when both factor matrices are required to be identical. Symmetric NMF is used for clustering problems [28] and known to be equivalent to k-means kernel clustering [14]. Given a target matrix $T \in \mathbb{R}^{n\times n}$, symmetric NMF is given as the following optimization problem

$$\begin{array}{ll} \min\limits_{U} & \left\|T - UU^\top\right\|_{\text{Fro}}^2 \\ \text{s.t.} & U \geq 0, \end{array}$$

where $U \in \mathbb{R}^{n\times k}$ is a positive factor matrix of rank k. Note, the problem cannot be modeled and solved by CVXPY since it is non-convex. It has been addressed recently in [45] by two new methods. Both methods are symmetric variants of the alternating non-negative least squares (ANLS) [27] and the hierarchical ALS (HALS) [10] algorithms.

We compared GENO to both methods. For the comparison we used the code and same experimental setup as in [45]. Random positive-semidefinite target matrices $X = \hat{U}\hat{U}^\top$ of different sizes were computed from random matrices $\hat{U} \in \mathbb{R}^{n\times k}$ with absolute value Gaussian entries. As can be seen in Figure 3, GENO outperforms both methods (SymANLS and SymHALS) by a large margin.

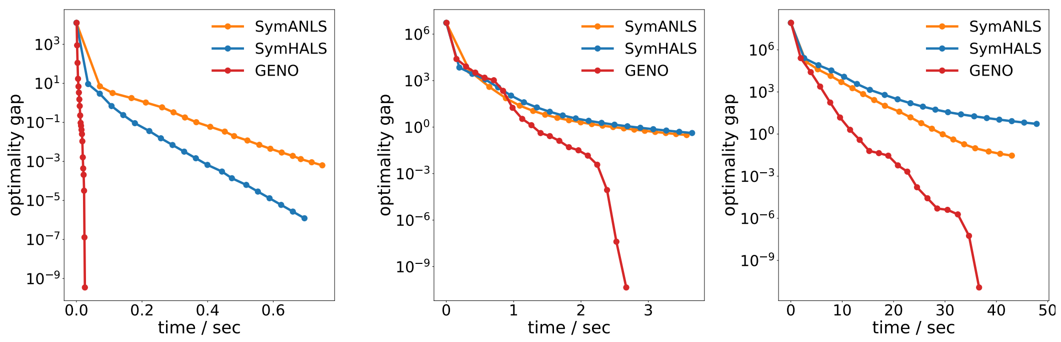

Figure 3: Convergence speed on the symmetric non-negative matrix factorization problem for different parameter values. On the left, the times for $m = 50, k = 5$, in the middle for $m = 500, k = 10$, and on the right for $m = 2000, k = 15$.

4.3 Further Experiments

Further experiments on ℓ_1-regularized logistic regression, support vector machines, elastic net regression, non-negative least squares, problems from non-convex optimization, and compressed sensing along with the GENO models for all experiments can be found in the full version of this paper [31, 32].

5 Conclusions

While other fields of applied research that heavily rely on optimization, like operations research, optimal control and deep learning, have adopted optimization frameworks, this is not the case for classical machine learning. Instead, classical machine learning methods are still mostly accessed through toolboxes like scikit-learn, Weka, or MLlib. These toolboxes provide well-engineered solutions for many of the standard problems, but lack the flexibility to adapt the underlying models when necessary. We attribute this state of affairs to a common belief that efficient optimization for classical machine learning needs to exploit the problem structure. Here, we have challenged this belief. We have presented GENO, the first general purpose framework for problems from classical machine learning. GENO combines an easy-to-read modeling language with a general purpose solver. Experiments on a variety of problems from classical machine learning demonstrate that GENO is as efficient as established well-engineered solvers and often outperforms recently published state-of-the-art solvers by a good margin. It is as flexible as state-of-the-art modeling language and solver frameworks, but outperforms them by a few orders of magnitude.

Acknowledgments

Sören Laue has been funded by Deutsche Forschungsgemeinschaft (DFG) under grant LA 2971/1-1.

References

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: A system for large-scale machine learning. In *USENIX Conference on Operating Systems Design and Implementation (OSDI)*, pages 265–283, 2016.

[2] Gediminas Adomavicius and Alexander Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Transactions on Knowledge & Data Engineering*, (6):734–749, 2005.

[3] Akshay Agrawal, Robin Verschueren, Steven Diamond, and Stephen Boyd. A rewriting system for convex optimization problems. *Journal of Control and Decision*, 5(1):42–60, 2018.

[4] Goran Banjac, Bartolomeo Stellato, Nicholas Moehle, Paul Goulart, Alberto Bemporad, and Stephen P. Boyd. Embedded code generation using the OSQP solver. In *Conference on Decision and Control, (CDC)*, pages 1906–1911, 2017.

[5] David M Blei, Andrew Y Ng, and Michael I Jordan. Latent dirichlet allocation. *Journal of Machine Learning Research*, 3(Jan):993–1022, 2003.

[6] Stephen P. Boyd, Neal Parikh, Eric Chu, Borja Peleato, and Jonathan Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends in Machine Learning*, 3(1):1–122, 2011.

[7] A. Brooke, D. Kendrick, and A. Meeraus. *GAMS: release 2.25 : a user's guide*. The Scientific press series. Scientific Press, 1992.

[8] Richard H. Byrd, Peihuang Lu, Jorge Nocedal, and Ciyou Zhu. A limited memory algorithm for bound constrained optimization. *SIAM J. Scientific Computing*, 16(5):1190–1208, 1995.

[9] Chih-Chung Chang and Chih-Jen Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27, 2011.

[10] Andrzej Cichocki and Anh-Huy Phan. Fast local algorithms for large scale nonnegative matrix and tensor factorizations. *Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, 92(3):708–721, 2009.

[11] CVX Research, Inc. CVX: Matlab software for disciplined convex programming, version 2.1. http://cvxr.com/cvx, December 2018.

[12] Aaron Defazio. A simple practical accelerated method for finite sums. In *Advances in Neural Information Processing Systems (NIPS)*, pages 676–684, 2016.

[13] Steven Diamond and Stephen Boyd. CVXPY: A Python-embedded modeling language for convex optimization. *Journal of Machine Learning Research*, 17(83):1–5, 2016.

[14] Chris Ding, Xiaofeng He, and Horst D Simon. On the equivalence of nonnegative matrix factorization and spectral clustering. In *SIAM International Conference on Data Mining (SDM)*, pages 606–610. SIAM, 2005.

[15] Alexander Domahidi, Eric Chu, and Stephen P. Boyd. ECOS: An SOCP Solver for Embedded Systems. In *European Control Conference (ECC)*, 2013.

[16] Iain Dunning, Joey Huchette, and Miles Lubin. JuMP: A modeling language for mathematical optimization. *SIAM Review*, 59(2):295–320, 2017.

[17] Robert Fourer, David M. Gay, and Brian W. Kernighan. *AMPL: a modeling language for mathematical programming*. Thomson/Brooks/Cole, 2003.

[18] Eibe Frank, Mark A. Hall, and Ian H. Witten. *The WEKA Workbench. Online Appendix for "Data Mining: Practical Machine Learning Tools and Techniques"*. Morgan Kaufmann, fourth edition, 2016.

[19] Jerome H. Friedman, Trevor Hastie, and Rob Tibshirani. Regularization paths for generalized linear models via coordinate descent. *Journal of Statistical Software*, 33(1):1–22, 2010.

[20] Daniel Gabay and Bertrand Mercier. A dual algorithm for the solution of nonlinear variational problems via finite element approximation. *Computers & Mathematics with Applications*, 2(1):17 – 40, 1976.

[21] P. Giselsson and S. Boyd. Linear convergence and metric selection for Douglas-Rachford splitting and ADMM. *IEEE Transactions on Automatic Control*, 62(2):532–544, Feb 2017.

[22] R. Glowinski and A. Marroco. Sur l'approximation, par éléments finis d'ordre un, et la résolution, par pénalisation-dualité d'une classe de problèmes de dirichlet non linéaires. *ESAIM: Mathematical Modelling and Numerical Analysis - Modélisation Mathématique et Analyse Numérique*, 9(R2):41–76, 1975.

[23] M. Grant and S. Boyd. Graph implementations for nonsmooth convex programs. In V. Blondel, S. Boyd, and H. Kimura, editors, *Recent Advances in Learning and Control*, Lecture Notes in Control and Information Sciences, pages 95–110. 2008.

[24] Magnus R. Hestenes. Multiplier and gradient methods. *Journal of Optimization Theory and Applications*, 4(5):303–320, 1969.

[25] Thomas Hofmann. Probabilistic latent semantic analysis. In *Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 289–296, 1999.

[26] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv:1408.5093*, 2014.

[27] Jingu Kim and Haesun Park. Toward faster nonnegative matrix factorization: A new algorithm and comparisons. In *IEEE International Conference on Data Mining (ICDM)*, pages 353–362, 2008.

[28] Da Kuang, Sangwoon Yun, and Haesun Park. Symnmf: nonnegative low-rank approximation of a similarity matrix for graph clustering. *Journal of Global Optimization*, 62(3):545–574, 2015.

[29] Sören Laue. On the equivalence of forward mode automatic differentiation and symbolic differentiation. *arXiv:1904.02990*, 2019.

[30] Sören Laue, Matthias Mitterreiter, and Joachim Giesen. Computing higher order derivatives of matrix and tensor expressions. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.

[31] Sören Laue, Matthias Mitterreiter, and Joachim Giesen. GENO – GENeric Optimization for Classical Machine Learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

[32] Sören Laue, Matthias Mitterreiter, and Joachim Giesen. GENO – GENeric Optimization for Classical Machine Learning. *arXiv:1905.13587*, 2019.

[33] Sören Laue, Matthias Mitterreiter, and Joachim Giesen. A simple and efficient tensor calculus. In *Conference on Artificial Intelligence (AAAI)*, 2020. To appear.

[34] Hongzhou Lin, Julien Mairal, and Zaïd Harchaoui. A universal catalyst for first-order optimization. In *Advances in Neural Information Processing Systems (NIPS)*, pages 3384–3392, 2015.

[35] Jacob Mattingley and Stephen Boyd. CVXGEN: A Code Generator for Embedded Convex Optimization. *Optimization and Engineering*, 13(1):1–27, 2012.

[36] Xiangrui Meng, Joseph Bradley, Burak Yavuz, Evan Sparks, Shivaram Venkataraman, Davies Liu, Jeremy Freeman, DB Tsai, Manish Amde, Sean Owen, Doris Xin, Reynold Xin, Michael J. Franklin, Reza Zadeh, Matei Zaharia, and Ameet Talwalkar. Mllib: Machine learning in apache spark. *Journal of Machine Learning Research*, 17(1), January 2016.

[37] MOSEK ApS. The MOSEK optimization software. http://www.mosek.com.

[38] Brendan O'Donoghue, Eric Chu, Neal Parikh, and Stephen Boyd. Conic optimization via operator splitting and homogeneous self-dual embedding. *Journal of Optimization Theory and Applications*, 169(3):1042–1068, 2016.

[39] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. In *NIPS Autodiff workshop*, 2017.

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

[41] M. J. D. Powell. Algorithms for nonlinear constraints that use Lagrangian functions. *Mathematical Programming*, 14(1):224–248, 1969.

[42] Herbert Robbins and Sutton Monro. A stochastic approximation method. *Ann. Math. Statist.*, 22(3):400–407, 1951.

[43] Shai Shalev-Shwartz and Tong Zhang. Stochastic dual coordinate ascent methods for regularized loss. *Journal of Machine Learning Research*, 14(1):567–599, 2013.

[44] Ciyou Zhu, Richard H. Byrd, Peihuang Lu, and Jorge Nocedal. Algorithm 778: L-BFGS-B: fortran subroutines for large-scale bound-constrained optimization. *ACM Trans. Math. Softw.*, 23(4):550–560, 1997.

[45] Zhihui Zhu, Xiao Li, Kai Liu, and Qiuwei Li. Dropping symmetry for fast symmetric nonnegative matrix factorization. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 5160–5170, 2018.