
Hyperparameter optimization with approximate gradient

Fabian Pedregosa

F@BIANP.NET

Chaire Havas-Dauphine “Économie des Nouvelles Données”
CEREMADE, CNRS UMR 7534, Université Paris-Dauphine, PSL Research University
Département Informatique de l’École Normale Supérieure, Paris

Abstract

Most models in machine learning contain at least one hyperparameter to control for model complexity. Choosing an appropriate set of hyperparameters is both crucial in terms of model accuracy and computationally challenging. In this work we propose an algorithm for the optimization of continuous hyperparameters using inexact gradient information. An advantage of this method is that hyperparameters can be updated before model parameters have fully converged. We also give sufficient conditions for the global convergence of this method, based on regularity conditions of the involved functions and summability of errors. Finally, we validate the empirical performance of this method on the estimation of regularization constants of ℓ_2 -regularized logistic regression and kernel Ridge regression. Empirical benchmarks indicate that our approach is highly competitive with respect to state of the art methods.

1. Introduction

Most models in machine learning feature at least one hyperparameter to control for model complexity. Regularized models, for example, control the trade-off between a data fidelity term and a regularization term through one or several hyperparameters. Among its most well-known instances are the LASSO (Tibshirani, 1996), in which ℓ_1 regularization is added to a squared loss to encourage sparsity in the solutions, or ℓ_2 -regularized logistic regression, in which squared ℓ_2 regularization (known as *weight decay* in the context of neural networks) is added to obtain solutions with small euclidean norm. Another class of hyperparam-

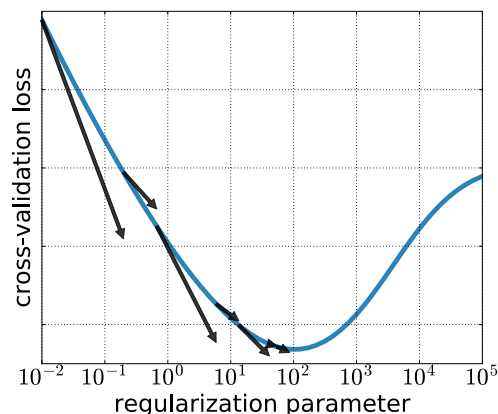


Figure 1. Hyperparameter Optimization with approximate gradient. The gradient of the cross-validation loss function with respect to hyperparameters is computed approximately. This noisy gradient is then used to estimate the optimal hyperparameters by gradient descent. A decreasing bound between the true gradient and the approximate gradient ensures that the method converges towards a stationary point.

eters are the kernel parameters in support vector machines. For example, the popular radial basis function (RBF) kernel depends on a “width” parameter, while polynomial kernels depend on a discrete hyperparameter specifying the degree. Hyperparameters can be broadly categorized into two groups: continuous hyperparameters, such as regularization parameters or the width of an RBF kernel and discrete hyperparameters, such as the degree of a polynomial. In this work we focus on continuous hyperparameters.

The problem of identifying the optimal set of hyperparameters is known as *hyperparameter optimization*. Hyperparameters cannot be estimated to minimize the same cost function as model parameters, since this would favor models with excessive complexity. For example, if regularization parameters were chosen to minimize the same loss as model parameters, then models with no regularization would always yield the smallest loss. For this reason, hyperparameter optimization algorithms seek to optimize a criterion of model quality which is different from the cost

function used to fit model parameters. This criterion can be a goodness of fit on unseen data, such as a *cross-validation* loss, or some criteria of model quality on the train set such as SURE (Stein, 1981), AIC/BIC (Liu and Yang, 2011) or Mallows C_p (Mallows, 1973), to name a few.

Choosing the appropriate set of hyperparameters has often a dramatic influence in model accuracy and many hyperparameter optimization algorithms have been proposed in the literature. For example, in the widely used *grid-search* algorithm, the model is trained over a range of values for the hyperparameters and the value that gives the best performance on the cross-validation loss is chosen. This not only scales poorly with the number of hyperparameters, but also involves fitting the full model for values of hyperparameters that are very unpromising. Random search (Bergstra et al., 2011) has been proven to yield a faster exploration of the hyperparameter space than grid search, specially in spaces with multiple hyperparameters. However, none of these methods make use of previous evaluations to make an informed decision of the next iterate. As such, convergence to a global minima can be very slow.

In recent years, sequential model-based optimization (SMBO) techniques have emerged as a powerful tool for hyperparameter optimization (see e.g. (Brochu et al., 2010) for an review on current methodologies). These techniques proceed by fitting a probabilistic model to the data and then using this model as an inexpensive proxy in order to determine the most promising location to evaluate next. This probabilistic model typically relies on a Gaussian process regressor but other approaches exist using trees (Bergstra et al., 2011) or ensemble methods (Lacoste et al., 2014). The model is built using only function evaluations, and for this reason SMBO is often considered as a *black-box* optimization method.

A third family of methods, which includes the method that we present, estimate the optimal hyperparameters using smooth optimization techniques such as gradient descent. We will refer to these methods as *gradient-based* hyperparameter optimization methods. These methods use local information about the cost function in order to compute the gradient of the cost function with respect to hyperparameters. However, computing the gradient with respect to hyperparameters has revealed to be a major bottleneck in this approach. For this reason we propose an algorithm that replaces the gradient with an approximation. More precisely, we make the following contributions:

- We propose a gradient-based hyperparameter optimization algorithm that uses approximate gradient information rather than the true gradient.
- We provide sufficient conditions for the convergence of this method to a stationary point.

- We compare this approach against state-of-the art methods for the task of estimation of regularization and kernel parameter on two different models and three datasets.

Notation We denote the gradient of a real-valued function by ∇ . If this function has several input arguments, we denote ∇_i its gradient with respect to the i -th argument. Similarly, ∇^2 denotes the Hessian and $\nabla_{i,j}^2$ denotes the second order differential with respect to variables i and j . For functions that are not real-valued, we denote its differential by D . We denote the projection operator onto a set \mathcal{D} by $P_{\mathcal{D}}$. That is, $P_{\mathcal{D}}(\alpha) \triangleq \arg \min_{\lambda \in \mathcal{D}} \|\alpha - \lambda\|^2$, where $\|\cdot\|$ denotes the euclidean norm for vectors.

Throughout the paper we take the convention of denoting real-valued functions with lowercase letters (such as f and g) and vector-valued functions with uppercase letters (such as X). Model parameters are denoted using lowercase Latin letters (such as x) while hyperparameters are denoted using Greek lowercase letters (such as λ).

1.1. Problem setting

As mentioned in the introduction, the goal of hyperparameter optimization is to choose the hyperparameters λ that optimizes some criteria, such as a cross-validation loss or a SURE/AIC/BIC criteria. We will denote this criteria by $f : \mathbb{R}^s \rightarrow \mathbb{R}$, where s is the number of hyperparameters. In its simplest form, the hyperparameter optimization problem can be seen as the problem of minimizing the cost function f over a domain $\mathcal{D} \subseteq \mathbb{R}^s$. Some approaches, such as sequential model-based optimization, only require function evaluations of this cost function. The methods we are interested in however use local information of the objective function.

The cost function f (e.g. the cross-validation error) depends on the model parameters, which we will denote by $X(\lambda)$. These are commonly not available in closed form but rather defined implicitly as the minimizers of some cost function that we will denote $h(\cdot, \lambda) : \mathbb{R}^p \rightarrow \mathbb{R}$, where p is the number of model parameters. This makes the hyperparameter optimization problem can be naturally expressed as a nested or *bi-level optimization* problem:

$$\begin{aligned} \arg \min_{\lambda \in \mathcal{D}} \left\{ f(\lambda) \triangleq g(X(\lambda), \lambda) \right\} \\ \text{s.t. } X(\lambda) \in \arg \min_{x \in \mathbb{R}^p} h(x, \lambda) \end{aligned} \quad (\text{HO})$$

where the minimization over h is commonly referred to as the *inner optimization* problem. A notable example of hyperparameter optimization problem is that of regularization parameter selection by cross-validation. For simplic-

ity, we restrict the discussion to the case of simple or hold-out cross-validation, where the dataset is split only once, although the methods presented here extend naturally to other cross-validation schemes. In this setting, the dataset is split in two: a train set (denoted $\mathcal{S}_{\text{train}}$) and a test or hold-out set (denoted $\mathcal{S}_{\text{test}}$). In this case, the outer cost function is a goodness of fit or loss on the test set, while the inner one is a trade-off between a data fitting term on the train set and a penalty term. If the penalty term is a squared ℓ_2 -norm, then the problem adopts the form:

$$\begin{aligned} & \arg \min_{\lambda \in \mathcal{D}} \text{loss}(\mathcal{S}_{\text{test}}, X(\lambda)) \\ \text{s.t. } & X(\lambda) \in \arg \min_{x \in \mathbb{R}^p} \text{loss}(\mathcal{S}_{\text{train}}, x) + e^\lambda \|x\|^2 \end{aligned} \quad (1)$$

The trade-off in the inner optimization between the goodness of fit term and the penalty term is controlled through the hyperparameter λ . Higher values of λ bias the model parameters towards vectors with small euclidean norm, and the goal of the hyperparameter optimization problem is to find the right trade-off between these two terms. The parametrization of the regularization parameter by an exponential (e^λ) in Eq. (1) might seem unusual, but given that this regularization parameter is commonly optimized over a log-spaced grid, we will find this parametrization useful in later sections.

Turning back to the general problem (HO), we will now describe an approach to compute the derivative of the cost function f with respect to hyperparameters. This approach, which we will refer to as implicit differentiation (Larsen et al., 1996; Bengio, 2000; Foo et al., 2008), relies on the observation that under some regularity conditions it is possible to replace the inner optimization problem by an implicit equation. For example, if h is smooth and verifies that all stationary points are global minima (as is the case for convex functions), then the values $X(\lambda)$ are characterized by the implicit equation $\nabla_1 h(X(\lambda), \lambda) = 0$. Deriving the implicit equation with respect to λ leads to the equation $\nabla_{1,2}^2 h + \nabla_1^2 h \cdot DX = 0$, which, assuming $\nabla_1^2 h$ invertible, characterizes the derivative of X . The chain rule, together with this equation, allows us to write the following formula for the gradient of f :

$$\begin{aligned} \nabla f &= \nabla_2 g + (DX)^T \nabla_1 g \\ &= \nabla_2 g - \left(\nabla_{1,2}^2 h \right)^T \left(\nabla_1^2 h \right)^{-1} \nabla_1 g \end{aligned} \quad (2)$$

This formula allows to compute the gradient of f given the following quantities: model parameters $X(\lambda)$ (g and h are evaluated at $(X(\lambda), \lambda)$) and $\left(\nabla_1^2 h \right)^{-1} \nabla_1 g$, which is usually computed as the solution to the linear system $\left(\nabla_1^2 h \right) z = \nabla_1 g$ for z . In the section that follows, we present an algorithm that relaxes the condition of both knowledge of the exact model parameters and exact solution of the linear system.

2. HOAG: Hyperparameter optimization with approximate gradient

As we have seen in the previous section, computing an exact gradient of f can be computationally demanding. In this section we present an algorithm that uses an approximation, rather than the true gradient, in order to estimate the optimal hyperparameters. This approach yields a trade-off between speed and accuracy: a loose approximation can be computed faster but might result in slow convergence or even divergence of the algorithm. At iteration k , this trade-off is balanced by the tolerance parameter ε_k . The sequence of tolerance parameters $\{\varepsilon_1, \varepsilon_2, \dots\}$ will turn out to play a major role in the convergence of the algorithm, although the time being, we will treat it as free parameter. We now describe our main contribution, the HOAG algorithm:

Algorithm 1 (HOAG). *At iteration $k = 1, 2, \dots$ perform the following:*

- (i) *Solve the inner optimization problem up to tolerance ε_k . That is, find x_k such that*

$$\|X(\lambda_k) - x_k\| \leq \varepsilon_k \quad .$$

- (ii) *Solve the linear system $\nabla_1^2 h(x_k, \lambda_k) q_k = \nabla_1 g(x_k, \lambda_k)$ for q_k up to tolerance ε_k . That is, find q_k such that*

$$\left\| \nabla_1^2 h(x_k, \lambda_k) q_k - \nabla_1 g(x_k, \lambda_k) \right\| \leq \varepsilon_k \quad .$$

- (iii) *Compute approximate gradient p_k as*

$$p_k = \nabla_2 g(x_k, \lambda_k) - \nabla_{1,2}^2 h(x_k, \lambda_k)^T q_k \quad ,$$

- (iv) *Update hyperparameters:*

$$\lambda_{k+1} = P_{\mathcal{D}} \left(\lambda_k - \frac{1}{L} p_k \right) \quad .$$

This algorithm consists of four steps. The first two steps of the algorithm compute approximations to the quantities used in Eq. (2) to compute the gradient of f . However, since these are not computed to full accuracy, p_k , computed in step (iii) is a noisy estimate of the gradient. This approximation is then used as a replacement of the true gradient in a projected gradient-descent (iv) iteration.

This procedure requires access to three quantities at iteration k : a ε_k -optimal solution to the inner optimization problem which can be computed with any solver, the first-order derivatives of g , $(\nabla_1 g, \nabla_2 g)$, and an ε_k -optimal solution to a linear system involving $\nabla_1^2 h$. In practice, this system

is solved using a conjugate-gradient method, which only requires access to the matrix $\nabla_1^2 h$ through matrix-vector products. For example, in machine learning problems such as the ones introduced in Eq. (1), the quantity $\nabla_1^2 h$ corresponds to the Hessian of the inner optimization problem. Efficient schemes for multiplication by the Hessian can be derived for least squares, logistic regression (Lin et al., 2008) and other general loss functions (Pearlmutter, 1994).

2.1. Related work

There exists a large variety of hyperparameter optimization methods, and a full review of this literature would be outside the scope of this work. Below, we comment on the relationship between HOAG and some of the most closely related methods.

Regarding gradient-based hyperparameter optimization methods we will distinguish two main approaches, implicit differentiation and iterative differentiation, depending on how the gradient with respect to hyperparameters is computed.

Implicit differentiation. This approach consists in deriving an implicit equation for the gradient using the optimality conditions of the inner optimization problem (as we did in Eq. (2)). Originally motivated by the problem of setting the regularization parameter in the context of neural networks (Larsen et al., 1996; 1998; Bengio, 2000), has also been applied to the problem of selecting kernel parameters (Chapelle et al., 2002; Seeger, 2008) or multiple regularization parameters in log-linear models (Foo et al., 2008). This approach has also been successfully applied to the problem of image reconstruction (Kunisch and Pock, 2013; Calatroni et al., 2015), in which case the simplicity of the cost function function allows for a particularly simple expression of the gradient with respect to hyperparameters.

Iterative differentiation. In this approach, the gradient with respect to hyperparameters is computed by differentiating each step of the inner optimization algorithm and then using the chain rule to aggregate the results. Since the gradient is computed after a finite number of steps of the inner optimization routine, the estimated gradient is naturally an approximation to the true gradient. This method was first proposed by Domke (2012) and later extended to the setting of stochastic gradient descent by Maclaurin et al. (2015). We note also that contrary to the implicit differentiation approach, this method can be applied to problems with non-smooth cost functions (Deledalle et al., 2014; Ochs et al.).

HOAG, while belonging to the class of implicit differentiation methods, is related to iterative differentiation methods

in that it allows the gradient with respect to hyperparameters to be computed approximately.

Finally, we note that similar approaches have also been considered in the setting of **sequential model-based optimization**. Swersky et al. (2014) proposes an approach in which the inner optimization is “frozen” whenever the method decides that the current hyperparameter values are not promising. It does so by introducing a prior on training curves as a function of input hyperparameters. This approach however requires to make strong assumptions on the shape of the training curves which gradient-based methods do not make.

3. Analysis

In this section we will prove that the summability of the tolerance sequence $\{\varepsilon_i\}_{i=1}^\infty$ is sufficient to guarantee convergence of the iterates in HOAG. The analysis of this algorithm is inspired by the work of d’Aspremont (2008); Schmidt et al. (2011); Friedlander and Schmidt (2012) on inexact-gradient algorithms for convex optimization.

We will start this section by enumerating the regularity conditions that we assume for the hyperparameter optimization problem. The following conditions are assumed through the section:

- (A1) *L-smoothness.* We assume that the first derivatives of g and the second derivatives of h are Lipschitz continuous functions.
- (A2) *Nonsingular Hessian.* We assume that the matrix $\nabla_1^2 h$, which corresponds to the Hessian of the inner optimization problem, is invertible at the values $(X(\lambda), \lambda)$, $\lambda \in \mathcal{D}$.
- (A3) *Convex compact domain.* The domain under which the hyperparameters are optimized, \mathcal{D} , is a convex non-empty and compact subset of \mathbb{R}^s .

These assumptions are verified by many models of interest. For example, for the problem of estimation of regularization parameters of Eq. (1), it allows twice-differentiable loss functions such as logistic regression or least squares (assumption A1) and strongly convex penalties (A2), such as squared ℓ_2 regularization. Note that condition (A2) need not be verified on all its domain, only on the points $(X(\lambda), \lambda)$, which would allow in principle to consider models that are defined through a non-convex cost functions. Assumption (A3) requires that the domain of the hyperparameters is a convex compact domain. In practice, hyperparameters are optimized over a s -dimensional interval, i.e., a domain of the form $\mathcal{D} = [a_1, b_1] \times \cdots [a_s, b_s]$.

Our analysis however only require this domain to be convex and compact, a constraint that subsumes s -dimensional intervals.

The rest of the section is devoted to prove (under conditions) the convergence of HOAG. The proof is divided in two parts. First, we will prove that the difference between the true gradient and the approximate gradient is bounded by $\mathcal{O}(\varepsilon)$ (Theorem 1) and in a second part we will prove that if the sequence $\{\varepsilon_i\}_{i=1}^{\infty}$ is summable, then this implies the convergence to a stationary point of f (Theorem 2). Because of space limitation, the proofs are omitted and can be found in Appendix A.

Theorem 1 (The gradient error is bounded). *For sufficiently large k , the error in the gradient is bounded by a constant factor of ε_k . That is,*

$$\|\nabla f(\lambda_k) - p_k\| = \mathcal{O}(\varepsilon_k) \quad .$$

This theorem gives a bound on the gradient from the sequence that bounds the inner optimization and the linear system solution. It will be the key ingredient in order to show convergence to a stationary point, which is the main result of this section. This property sometimes referred to as *global convergence* (Nocedal and Wright, 2006):

Theorem 2 (Global convergence). *If the tolerance sequence is summable, that is, if $\{\varepsilon_i\}_{i=1}^{\infty}$ is positive and verifies*

$$\sum_{i=1}^{\infty} \varepsilon_i \leq \infty \quad ,$$

then the sequence λ_k of iterates in the HOAG algorithm has limit $\lambda^ \in \mathcal{D}$, and this limit verifies the stationary point condition:*

$$\langle \nabla f(\lambda^*), \alpha - \lambda^* \rangle \geq 0 \quad , \quad \forall \alpha \in \mathcal{D} \quad .$$

In particular, if λ^ belongs to the interior of \mathcal{D} it is verified that*

$$\nabla f(\lambda^*) = 0 \quad .$$

This results gives sufficient conditions for the convergence of HOAG. The summability of the tolerance sequence suggest several natural candidates for this sequence, such as the quadratic sequence, $\varepsilon_k = k^{-2}$ or the exponential sequence, $\varepsilon_k = \rho^k$, with $0 < \rho < 1$. We will empirically evaluate different tolerance sequences on different problems and different datasets in the next section.

Experiments

In this section we compare the empirical performance of HOAG. We start by discussing some implementation details such as the choice of step size. Then, we compare

the convergence of different tolerance decrease strategies that were suggested by the theoretical analysis. In a third part, we compare the performance of HOAG against other hyperparameter optimization methods.

Adaptive step size. Our algorithm relies on the knowledge of the Lipschitz constant L for the cost function f . However, in practice this is not known in advance. Furthermore, since the cost function is costly to evaluate, it is not feasible to perform backtracking line search. To overcome this we use a procedure in which the step size is corrected depending on the gain estimated from the previous step. In the experiments we use this technique although we do not have a formal analysis of the algorithm for this choice of step size.

Let Δ_k denote the distance between the current iterate and the past iterate, $\Delta_k = \|\lambda_k - \lambda_{k-1}\|$. The L -smooth property of the function g , together with Lemma 1, implies that there exists a constant $M > 0$ such that the following inequality is verified:

$$g(\lambda_k, x_k) \leq g(\lambda_{k-1}, x_{k-1}) + C\varepsilon_k + \varepsilon_{k-1}(C + M)\Delta_k - L\Delta_k^2 \quad , \quad (3)$$

where C is the Lipschitz constant of g (for loss functions such as logistic or least squares this can easily be computed from the data). This inequality can be derived from the properties of L -smooth functions, and the details can be found in Appendix B. The procedure consists in decreasing the step (multiplication by $\alpha < 1$) whenever the equation is not satisfied and to increase it (multiplication by $\beta > 1$) whenever the equation is satisfied to ensure that we are using a step size as large as possible. The constants that we used in the experiments are $M = 1, \alpha = 0.5, \beta = 1.05$.

Stopping criterion. The stopping criterion given in Algorithm 1 depends on $X(\lambda)$ which is generally unknown. However, for objective functions in the inner optimization which are μ -strongly convex ($\mu/2$ can be taken as the amount of regularization in ℓ_2 -regularized objectives), it is possible to lower bound the quantity $\|X(\lambda_k) - x_k\|$ by $\mu^{-1}\|g'(\lambda_k, x_k)\|$. Hence, it is sufficient to ensure $\mu^{-1}\|g'(\lambda_k, x_k)\| \leq \varepsilon$. Details can be found in Appendix B.

Initialization. The previous sections tells us how to adjust the step size but relies on an initial value of this parameter. We have found that a reasonable initialization is to initialize it to $L = \|p_1\|$ so that the first update in HOAG is of magnitude at most 1 (it can be smaller due to the projection), where p_1 is the approximate gradient on the first step. The initialization of the tolerance decrease sequence is set to $\varepsilon_1 = 0.1$. We also limit the maximum precision to avoid numerical instabilities to 10^{-12} , which is also the precision for “exact” methods, i.e., those that do not use a tolerance sequence. The initialization of regularization parameters is set to 0 and the width of an RBF kernel is initialized to

$-\log(\text{n_feat})$, where n_feat is the number of features or dimensionality of the dataset.

Although HOAG can be applied more generally, in our experiments we focus on two problems: ℓ_2 -regularized logistic regression and kernel Ridge regression. We follow the setting described in Eq. (1), in which an initial dataset is partitioned into two sets, a train set $\mathcal{S}_{\text{train}} = \{(b_i, a_i)\}_{i=1}^n$ and a test set $\mathcal{S}_{\text{test}} = \{(b'_i, a'_i)\}_{i=1}^m$, where a_i denotes the input features and b_i the target variables.

The first problem consists in estimating the regularization parameter in the widely-used ℓ_2 -regularized logistic regression model. In this case, the loss function of the inner optimization problem is the regularized logistic loss function. In the setting of classification, the validation loss or outer cost function is commonly the zero-one loss. However, this loss is non-smooth and so does not verify assumption (A1). To overcome this and following (Foo et al., 2008), we use the logistic loss as the validation loss. This yield a problem of the form:

$$\begin{aligned} & \arg \min_{\lambda \in \mathcal{D}} \sum_{i=1}^m \psi(b'_i a_i'^T X(\lambda)) \\ \text{s.t. } & X(\lambda) \in \arg \min_{x \in \mathbb{R}^p} \sum_{i=1}^n \psi(b_i a_i^T x) + e^\lambda \|x\|^2, \end{aligned} \quad (4)$$

where ψ is the logistic loss, i.e., $\psi(t) = \log(1 + e^{-t})$. The second problem that we consider is that of kernel Ridge regression with an RBF kernel. In this setting, the problem contains two hyperparameters: the first hyperparameter (λ_1) controls the width of the RBK kernel and the second hyperparameter (λ_2) controls the amount of regularization. The inner optimization depends on the kernel through the kernel matrix, formed by computing the kernel of all pairwise input samples. We denote such matrix as $K(\gamma)_{\text{train}}$, where the (i, j) entry is given by $k(a_i, a_j, \gamma)$, where k is the RBF kernel function: $k(a_i, a_j, \gamma) = \exp(-\gamma \|a_i - a_j\|)$. Similarly, the outer optimization also depends on the kernel through the matrix $K(\gamma)_{\text{test}}$, where its entries are the kernel product between features from the train set and features from the test set, that is, $k(a_i, a'_j, \gamma)$. Denoting the full hyperparameter vector as $\lambda = [\lambda_1, \lambda_2]$, the kernel matrix on the train set as, the full hyperparameter optimization problem takes the form

$$\begin{aligned} & \arg \min_{\lambda \in \mathcal{D}} \|b - K_{\text{test}}(e^{\lambda_1}) X(\lambda)\|^2 \\ \text{s.t. } & (K_{\text{train}}(e^{\lambda_1}) + e^{\lambda_2} I) X(\lambda) = b, \end{aligned} \quad (5)$$

where for simplicity the inner optimization is already set as an implicit equation. Note that in this setting, and unlike in the logistic regression problem, the outer optimization function depends on the hyperparameters not only through

the model parameters $X(\lambda)$ but also through the kernel matrix.

The solver used for the inner optimization problem of the logistic regression problem is L-BFGS (Liu and Nocedal, 1989), while for Ridge regression we used a linear conjugate descent method. In all cases, the domain for hyperparameters is the s -dimensional interval $[-12, 12]^s$.

For the experiments, we use four **different datasets**. The dataset 20news and real-sim are studied with an ℓ_2 -regularized logistic regression model (1 hyperparameter) while the Parkinson dataset using a Kernel ridge regression model (2 hyperparameters). The MNIST dataset is investigated in a high-dimensional hyperparameter space using a similar setting to (Maclaurin et al., 2015, §3.2) and reported in in Appendix B. Datasets and models are described in more detail in Appendix B.

In all cases, the dataset is randomly split in three equally sized parts: a train set, test set and a third validation set that we will use to measure the generalization performance of the different approaches.

3.1. Tolerance decrease sequence

We report in Figure 2 the convergence of different tolerance decrease strategies. From Theorem 2, the sole condition on these sequences is that they are summable. Three notable examples of summable sequences are the quadratic, cubic and exponential sequences. Hence, we choose one representative of each of these strategies. More precisely, the decrease sequences that we choose are a quadratic decrease sequence of the form $\varepsilon_k = 0.1 \times k^{-2}$, a cubic one of the form $\varepsilon_k = 0.1 \times k^{-3}$ and an exponential of the form $\varepsilon_k = 0.1 \times (0.9^k)$. The value taken as true minima of the hyperparameter optimization problem is computed by taken the minimum reached by 10 randomly initialized instances of HOAG with exponential decrease tolerance.

The plot shows the relative accuracy of the different variants as a function of time. It can be seen that non-exact methods feature a cheaper iteration cost, yielding a faster convergence overall. Note that despite the global convergence result of Theorem 2, HOAG is not guaranteed to be monotonically decreasing, and in fact, some degree of oscillation is expected when the decrease in the tolerance does not match the convergence rate (see e.g. Schmidt et al. (2011)). This can be appreciated in Figure 2, where the quadratic decrease sequence (and to some extent the cubic too) exhibits oscillations in the two first plots.

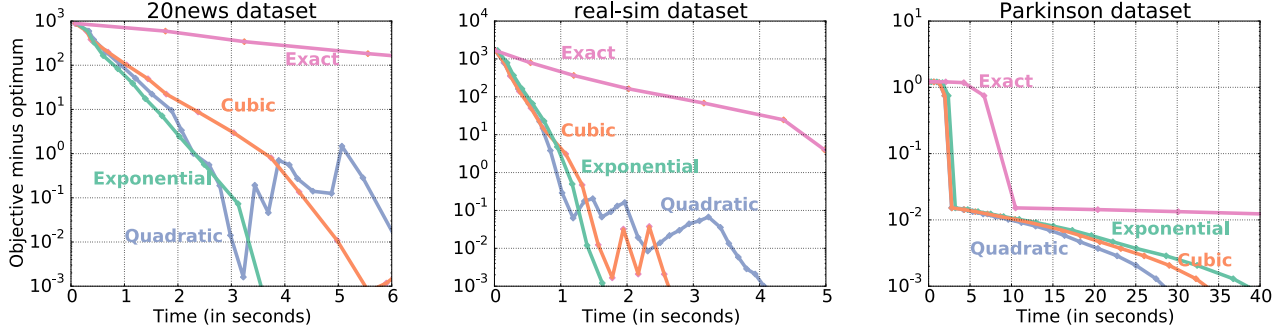


Figure 2. **Tolerance decrease strategies.** Suboptimality as a function of time for different tolerance decrease strategies. The decrease sequences considered are quadratic ($0.1k^{-2}$), cubic ($0.1k^{-3}$), exponential (0.1×0.9^k) and exact (gradient is computed to full accuracy at every iteration). Non-exact methods exhibit smaller cost per iteration, which results in faster convergence.

3.2. Comparison with other hyperparameter optimization methods

We now compare against other hyperparameter optimization methods. The methods against which we compare are:

- **HOAG.** The method we present in this paper, with an exponentially decreasing tolerance sequence. A Python implementation is made freely available at <https://github.com/fabianp/hoag>.
- **Grid Search.** This method consists simply in splitting the domain of the hyperparameter into an equally-spaced grid. We split the interval $[-12, 12]$ into a grid of 10 values.
- **Random.** This is the random search method (Bergstra and Bengio, 2012) samples the hyperparameters from a predefined distribution. We choose to samples from a uniform distribution in the interval $[-12, 12]$.
- **SMBO.** Sequential model-based optimization using Gaussian Process. We used the implementation found in the Python package BayesianOptimization (<http://github.com/fmfn/BayesianOptimization/>). As initialization for this method, we choose 4 values equally spaced between -12 and 12 . The acquisition function used is the expected improvement.
- **Iterdiff.** This is the iterative differentiation approach from (Domke, 2012), using the same inner-optimization algorithm as HOAG. While the original implementation used to have a backtracking line search procedure to estimate the step size, we found that this performed worst than any of the alternatives. For this reason, we use the adaptive step size strategy presented in Section 3 (assuming a zero tolerance parameter ε).

For all methods, the number of iterations used in the inner optimization algorithm (L-BFGS or GD) is set to 100, which is the same used by the other methods and the default in the scikit-learn (<http://scikit-learn.org>) package.

We report in Figure 3 the results of comparing the accuracy of these methods as a function of time. Note that it is expected that the different methods have different starting points. This is because Grid Search and SMBO naturally start from a pre-defined grid that starts from the extremes of the interval, while random search simply chooses a random point from the domain. For HOAG and Iterdiff, we take the initialization $\lambda_1 = 0$.

In the upper row of Figure 3 we can see the suboptimality of the different procedures as a function of time. We observe that HOAG and Iterdiff have similar behavior, although HOAG features a smaller cost per iteration. This can be explained because once HOAG has made a step it can use the previous solution of the inner optimization problem as a *warm-start* to compute the next gradient. This is not the case in Iterdiff since the computation of the gradient relies crucially on having sufficient iterations of the inner optimization algorithm.

We note that in the Parkinson dataset, solution is inside a region that is almost flat (the different cost functions can be seen in Figure 1 of the supplementary material). This can explain the difficulty of the methods to go beyond the 10^{-2} suboptimality level. In this case, SMBO, who starts by computing the cost function at the extremes of the domain converges instantly to this region, which explains its fast convergence, although it is unable to improve the initially reached suboptimality.

Suboptimality plots are a standard way to compare the performance of different optimization methods. However, for the context of machine learning it can be argued that estimating hyperparameters up to a high precision is unim-

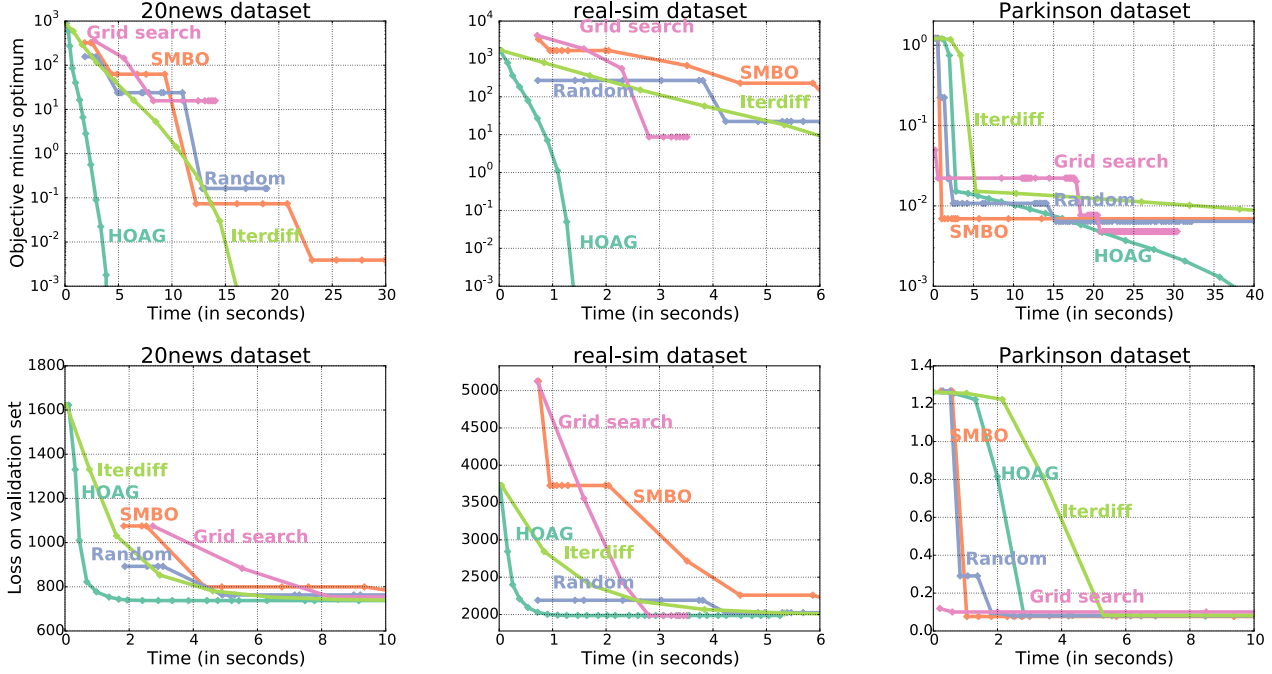


Figure 3. **Hyperparameter optimization methods.** Top row: suboptimality of the different methods in terms of the test loss. Bottom row: loss measured on a validation set for the different methods.

portant and that methods should be compared in terms of generalization performance. In the lower row of Figure 3, we display the test loss (g) on a validation set, that is, using a third set of samples $\{(b_i, \tilde{a}_i)\}_{i=1}^r$ which is different from both the train and test set. This figure reveals two main effects. First, unsurprisingly, optimization beyond 10^{-2} of relative suboptimality is not reflected in this metric. Second, the fast (but noisy) early iterations of HOAG achieve the fastest convergence in two out of three datasets.

4. Discussion and future work

In previous sections we have presented and discussed several aspects of the HOAG algorithm. Finally, we outline some future directions that we think are worthwhile exploring.

Given the success of recent stochastic optimization techniques (Schmidt et al., 2013; Johnson and Zhang, 2013) it seems natural to study a **stochastic variant** of this algorithm, that is, one in which the updates in the inner and outer optimization schemes have a cost that is independent of the number of samples. However, the dependency on the Hessian of the inner optimization ($\nabla_1^2 h$) in the implicit equation (2) makes this non-trivial.

Little is known of the **structure of solutions** for the hyperparameter optimization problem (HO). In fact, assumption

(A3) is introduced almost exclusively in order to guarantee existence of solutions. At the same time recent progress on the setting of image restoration, which can be considered a subproblem of (HO), has given sufficient conditions on the input data for such solution to exist in an unbounded domain (De los Reyes et al., 2015). The characterization of solutions for the HO problem can potentially simplify the assumptions made in this paper.

The analysis presented in this paper can be extended in several ways. For instance, the analysis of HOAG is provided for a constant step size and not for the **adaptive step size strategy** used in the experiments. Also, we have focused on proving asymptotic convergence of our algorithm. An interesting future direction would be to study **rates of convergence**, which might give insight into an optimal choice for the tolerance decrease sequence.

Acknowledgments

I am in debt with Gabriel Peyré for numerous discussions, suggestions and pointers. I would equally like to thank the anonymous reviewers for many insightful comments, and to Justin Domke for posting the code of his Iterative differentiation method.

Feedback and comments are welcome at the author’s blog (<http://goo.gl/WoV8R5>).

The author acknowledges financial support from the “Chaire Economie des Nouvelles Données”, under the auspices of Institut Louis Bachelier, Havas-Media and Université Paris-Dauphine (ANR 11-LABX-0019).

References

- Yoshua Bengio. Gradient-based optimization of hyperparameters. *Neural computation*, 12(8):1889–1900, 2000.
- James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *The Journal of Machine Learning Research*, 13(1), 2012.
- James S. Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In *Advances in Neural Information Processing Systems 24*. 2011.
- Eric Brochu, Vlad M Cora, and Nando De Freitas. A tutorial on bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. *arXiv preprint arXiv:1012.2599*, 2010.
- Luca Calatroni, Cao Chung, Juan Carlos De Los Reyes, Carola-Bibiane Schönlieb, and Tuomo Valkonen. Bilevel approaches for learning of variational imaging models. *arXiv preprint arXiv:1505.02120*, 2015.
- Olivier Chapelle, Vladimir Vapnik, Olivier Bousquet, and Sayan Mukherjee. Choosing multiple parameters for support vector machines. *Machine learning*, 2002.
- Alexandre d’Aspremont. Smooth optimization with approximate gradient. *SIAM Journal on Optimization*, 2008.
- J.C. De los Reyes, Carola-Bibiane Schönlieb, and Tuomo Valkonen. The structure of optimal parameters for image restoration problems. *arXiv preprint arXiv:1505.01953*, 2015.
- Charles-Alban Deledalle, Samuel Vaite, Jalal Fadili, and Gabriel Peyré. Stein unbiased gradient estimator of the risk (SUGAR) for multiple parameter selection. *SIAM Journal on Imaging Sciences*, 2014.
- Justin Domke. Generic methods for optimization-based modeling. In *International Conference on Artificial Intelligence and Statistics*, 2012.
- Chuan-Sheng Foo, Chuong B. Do, and Andrew Y. Ng. Efficient multiple hyperparameter learning for log-linear models. In *Advances in Neural Information Processing Systems 20*. 2008.
- Michael Friedlander and Mark Schmidt. Hybrid deterministic-stochastic methods for data fitting. *SIAM Journal on Scientific Computing*, 2012.
- Nicholas J Higham. *Accuracy and stability of numerical algorithms*. 2002.
- Rie Johnson and Tong Zhang. Accelerating stochastic gradient descent using predictive variance reduction. In *Advances in Neural Information Processing Systems*, 2013.
- Karl Kunisch and Thomas Pock. A bilevel optimization approach for parameter learning in variational models. *SIAM Journal on Imaging Sciences*, 2013.
- Alexandre Lacoste, Hugo Larochelle, François Laviolette, and Mario Marchand. Sequential model-based ensemble optimization. *arXiv preprint arXiv:1402.0796*, 2014.
- Jan Larsen, Lars Kai Hansen, Claus Svarer, and M Ohlsson. Design and regularization of neural networks: the optimal use of a validation set. In *Proceedings of the IEEE Signal Processing Society Workshop*. IEEE, 1996.
- Jan Larsen, Claus Svarer, Lars Nonboe Andersen, and Lars Kai Hansen. Adaptive regularization in neural network modeling. In *Neural Networks: Tricks of the Trade*. Springer, 1998.
- Chih-Jen Lin, Ruby C Weng, and S Sathiya Keerthi. Trust region newton method for logistic regression. *The Journal of Machine Learning Research*, 2008.
- Dong C Liu and Jorge Nocedal. On the limited memory bfgs method for large scale optimization. *Mathematical programming*, 1989.
- Wei Liu and Yuhong Yang. Parametric or nonparametric? a parametricness index for model selection. *The Annals of Statistics*, 2011.
- Dougal Maclaurin, David Duvenaud, and Ryan P. Adams. Gradient-based hyperparameter optimization through reversible learning. In *Proceedings of the 32nd International Conference on Machine Learning*, July 2015.
- Colin L Mallows. Some comments on C_p . *Technometrics*, 1973.
- Yurii Nesterov. *Introductory lectures on convex optimization*. Springer Science & Business Media, 2004.
- Jorge Nocedal and Stephen Wright. *Numerical optimization*. Springer Science & Business Media, 2006.
- Peter Ochs, René Ranftl, Thomas Brox, and Thomas Pock. Bilevel optimization with nonsmooth lower level problems. In *Scale Space and Variational Methods in Computer Vision*. Springer.

Neal Parikh and Stephen Boyd. Proximal algorithms. *Foundations and Trends in optimization*, 2013.

Barak A Pearlmutter. Fast exact multiplication by the hessian. *Neural computation*, 1994.

R Tyrrell Rockafellar and Roger J-B Wets. *Variational analysis*. Springer Science & Business Media.

Mark Schmidt, Nicolas Le Roux, and Francis R Bach. Convergence rates of inexact proximal-gradient methods for convex optimization. In *Advances in neural information processing systems*, 2011.

Mark Schmidt, Nicolas Le Roux, and Francis Bach. Minimizing finite sums with the stochastic average gradient. *arXiv preprint arXiv:1309.2388*, 2013.

Matthias W Seeger. Cross-validation optimization for large scale structured classification kernel methods. *The Journal of Machine Learning Research*, 2008.

Charles M Stein. Estimation of the mean of a multivariate normal distribution. *The annals of Statistics*, 1981.

Kevin Swersky, Jasper Snoek, and Ryan Prescott Adams. Freeze-thaw bayesian optimization. *arXiv preprint arXiv:1406.3896*, 2014.

Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, 1996.

Athanasios Tsanas, Max A Little, Patrick E McSharry, and Lorraine O Ramig. Accurate telemonitoring of parkinson's disease progression by noninvasive speech tests. *Biomedical Engineering, IEEE Transactions on*, 2010.