
Second-Order Forward-Mode Automatic Differentiation for Optimization

Adam D. Cobb*, Atılım Güneş Baydin† Barak A. Pearlmutter‡ Susmit Jha*

*Computer Science Laboratory, SRI International

†Department of Computer Science, University of Oxford

‡Department of Computer Science, National University of Ireland Maynooth

Abstract

This paper introduces a second-order hyperplane search, a novel optimization step that generalizes a second-order line search from a line to a k -dimensional hyperplane. This, combined with the forward-mode stochastic gradient method, yields a second-order optimization algorithm that consists of forward passes only, completely avoiding the storage overhead of backpropagation. Unlike recent work that relies on directional derivatives (or Jacobian–Vector Products, JVPs), we use hyper-dual numbers to jointly evaluate both directional derivatives and their second-order quadratic terms. As a result, we introduce forward-mode weight perturbation with Hessian information (FoMoH). We then use FoMoH to develop a novel generalization of line search by extending it to a hyperplane search. We illustrate the utility of this extension and how it might be used to overcome some of the recent challenges of optimizing machine learning models without backpropagation. Our code is open-sourced at <https://github.com/SRI-CSL/fomoh>.

1 Introduction

There is a growing interest in investigating the practical plausibility of forward-mode automatic differentiation (AD) as an alternative to reverse-mode AD (aka backpropagation) for neural network optimization. Forward gradient descent (FGD) [1] relies on sampling tangent vectors to update function parameters. These parameters are updated by subtracting the tangents that are scaled by their directional derivatives. As a result, their approach only requires forward passes, and avoids the computation and memory costs associated with implementing the backwards pass of reverse-mode AD. While there is still an accuracy gap between forward-mode and reverse-mode optimization approaches [30], there are several recent efforts that have explored alternative ways of leveraging forward-mode AD, with a focus of reducing the variance of the gradient estimator with the increase in dimensions [27, 9]. Thus, the large performance gap between the forward-mode approaches and backpropagation (BP) has been shrinking.

An interesting direction to explore, which builds on the existing work in forward-mode AD for optimization, is the introduction of Hessian information. Second-order derivative information provides optimization routines with information about the local curvature. These methods often require access to the Hessian, whose storage is not feasible for high-dimensional problems. As an example, a function, $f : \mathbb{R}^D \rightarrow \mathbb{R}$, has a quadratic storage cost of $\mathcal{O}(D^2)$ and a linear compute cost of $\mathcal{O}(D)$ reverse-mode gradient evaluations to build the Hessian [23, 6]. Instead of building the full Hessian through the use of reverse-mode AD, we introduce an approach incorporating Hessian information, known as FoMoH, which significantly outperforms the first-order gradient descent

*Correspondence to adam.cobb@sri.com.

method, FGD. Furthermore, we demonstrate how our approach effectively bridges the gap between a line search and a full Newton step. By incorporating Hessian information, FoMoH leverages second-order derivatives to provide more accurate curvature approximations of the objective function. This allows the method to adaptively adjust the step size and direction with greater precision compared to a simple line search, which only uses gradient information. Simultaneously, FoMoH avoids the computational complexity of a full Newton step, which requires calculation of $(\nabla^2 f)^{-1}\mathbf{v}$ where $\nabla^2 f$ is the Hessian. As a result, our approach balances the efficiency of a line search with the accuracy of a full Newton method, offering a robust and versatile optimization technique.

The central contributions of this paper are as follows:

- We introduce three new optimization approaches that use forward-mode second-order information.
- We show how one of these approaches, Forward-Mode Hyperplane Search, generalizes from a line search all the way to Newton’s method, without any need for backpropagation.
- We demonstrate the performance of the proposed new approaches on optimization problems of different parameter sizes and difficulty to show the advantage of second-order information.
- We release an AD backend in PyTorch that implements nested forward AD and interfaces with PyTorch models: <https://github.com/SRI-CSL/fomoh>.

The rest of the paper is organized as follows. §2 and §3 summarize the related work and relevant preliminary information regarding AD, which is then followed by §4 that outlines what is required to extend forward-mode AD to higher order derivatives. §5 introduces FoMoH and the generalization to our forward-mode hyperplane search. Finally, §6 provides experimental results that explore the behavior of FoMoH. We then conclude in §7.

2 Related Work

There has been considerable interest in developing approaches that avoid reverse-mode AD and its backwards pass. In moving away from backpropagation, it might be possible to build optimization algorithms that more closely align with biological systems [3, 12], or enable neural networks to run on emerging hardware architectures, such as analog optical systems [25]. Baydin et al. [1] introduced FGD as a possible replacement for backpropagation in neural network training by relying on weight perturbations. This removed the truncation error of previous weight perturbation approaches [23] by moving to forward-mode AD. While FGD is a promising backpropagation-free approach, the scaling of FGD to high-dimensional models is challenging due the variance of the gradient estimator. This challenge has led to multiple efforts that have focused on reducing this variance [27, 30, 9]. In particular, a common approach has been to rely on local backpropagation steps to reduce the variance and/or to provide a better guess for the perturbation direction. In our work, we focus on second-order forward-mode optimization approaches, but highlight that these other approaches on variance reduction are orthogonal to our approach and could also be combined together and generalized to FoMoH.

Becker et al. [2] were one of the first to use the second-order information to optimize neural networks. This approach leverages a local backpropagation step [14] to capture the diagonal Hessian terms resulting in what they call a “pseudo-Newton step”. Both LeCun et al. [17, §6–§9] and Goodfellow et al. [10, §8.6] provide discussions of the use of Hessian information for neural network training and cover Newton’s method, as well as the Levenberg–Marquardt algorithm [18, 20], conjugate gradients, and BFGS. A key objective of these approaches is to investigate whether second-order information can be leveraged for neural network training without the prohibitively high cost of a full Hessian evaluation. Additionally, the research question then explored is whether the approximated Hessian is still good enough to give the desired advantage over first-order methods. Examples of effective approaches often rely on diagonal preconditioners, with some approximating or directly using the diagonal of the Hessian [34, 14, 2], and others leveraging momentum and a variant of the diagonal of the empirical Fisher [7, 31, 13]. While extending a gradient preconditioner to the full inverse Hessian is generally infeasible, there are approaches that use: block-diagonal approximations, low-rank approximations, and Krylov-subspace based approximations [15, 32]. Additional references for these preconditioners can be found in Martens [21]. Finally, although it is challenging to scale second-order approaches to large neural network models, a recent work, Sophia [19], has managed to show success in using such an approach. Like previous works, they use a diagonal approximation

of the Hessian. Importantly, they show that Sophia requires half the number of update steps as Adam [13] to train a language model (GPT-2 [26]). It is worth noting that none of the described second-order optimization methods rely solely on forward-mode AD like the one being proposed in this paper.

3 Automatic Differentiation

In this section we summarize the common definitions of forward-mode and reverse-mode automatic differentiation. For a complete introduction, please refer to Griewank and Walther [11]. We use $\mathbf{x} \in \mathbb{R}^D$ to denote a column vector.

Forward-Mode AD [33] applies the chain rule in the *forward* direction. The forward-mode evaluation, $F(\boldsymbol{\theta}, \mathbf{v})$, requires an additional tangent vector, $\mathbf{v} \in \mathbb{R}^D$, along with the parameter vector $\boldsymbol{\theta} \in \mathbb{R}^D$ for a function $f : \mathbb{R}^D \rightarrow \mathbb{R}^O$. The result of the evaluation, $[f(\boldsymbol{\theta}), \nabla f(\boldsymbol{\theta})\mathbf{v}]$, is a function evaluation and the corresponding Jacobian vector product (JVP), where $\nabla f(\boldsymbol{\theta}) \in \mathbb{R}^{O \times D}$. For a unidimensional output function, the JVP is the directional derivative, $\nabla f(\boldsymbol{\theta}) \cdot \mathbf{v}$. The time and space (memory cost) complexity are linear, both approximately twice that of a single function evaluation.² A common implementation of forward-mode AD is to use dual numbers. A dual number $a + b\epsilon \in \mathbb{D}(\mathbb{R})$ contains a real (primal) component, $a \in \mathbb{R}$, and a dual component, $b \in \mathbb{R}$. We can think of this as representing a truncated Taylor series, $a + b\epsilon + \mathcal{O}(\epsilon^2)$, notationally simplified by the rule $\epsilon^2 = 0$. Using this, $f(a + b\epsilon) = f(a) + \nabla f(a)b\epsilon$. A simple example can be shown for the function, $f(a_1, a_2) = a_1 \times a_2$. Using dual numbers, $(a_1 + b_1\epsilon)(a_2 + b_2\epsilon)$, we retrieve the function evaluation and the corresponding well-known product rule: $a_1a_2 + (a_1b_2 + b_1a_2)\epsilon$. This can be extended to multiple dimensions, and is the basis of forward-mode AD: lift all real numbers \mathbb{R} to dual numbers $\mathbb{D}(\mathbb{R})$.

Reverse-Mode AD requires both a forward pass and a reverse pass. The reverse-mode evaluation $R(\boldsymbol{\theta}, \mathbf{u})$, also requires the additional adjoint vector, $\mathbf{u} \in \mathbb{R}^O$, which is often set to 1 for scalar-valued functions. Using the same notation as for forward-mode, an evaluation of reverse mode results in the vector-Jacobian product, $\mathbf{u}^\top \nabla f(\boldsymbol{\theta})$, as well as the function evaluation. When $\mathbf{u} = 1$, this results in the gradient $\nabla f(\boldsymbol{\theta})$. Reverse-mode is required to store intermediate values on the forward pass that are used during the reverse pass. This results in a higher time and space complexity, that is higher computational cost and memory footprint per call of $R(\boldsymbol{\theta}, \mathbf{u})$. However, for the scalar-valued functions ($O = 1$) that are common for ML optimization problems, only a single call of $R(\boldsymbol{\theta}, \mathbf{u})$ is needed to collect all the gradients compared to D (dimension of inputs) calls of $F(\boldsymbol{\theta}, \mathbf{v})$. This is one of the key reasons for the widespread adoption of the reverse-mode AD in current gradient-based machine learning methods, despite the higher memory footprint.

4 Higher-Order Forward Mode Automatic Differentiation

As described in the previous section, forward-mode automatic differentiation implementations can use dual numbers, $\boldsymbol{\theta} + \mathbf{v}\epsilon$. Dual numbers can be extended to truncate at a higher order, or to not truncate at all; and to allow nesting by supporting multiple distinct formal ϵ variables [24]. Specifically focusing on second-order terms is often referred to as hyper-dual numbers (for example in the Aeronautics and Astronautics community [8]). A hyper-dual number is made up from four components, which is written as $\boldsymbol{\theta} + \mathbf{v}_1\epsilon_1 + \mathbf{v}_2\epsilon_2 + \mathbf{v}_{12}\epsilon_1\epsilon_2$. In the same manner that we look at imaginary and real parts of complex numbers, we can look at the first derivative parts of a hyper-dual number by inspecting the ϵ_1 and ϵ_2 components, and we can look at the second derivative by inspecting the $\epsilon_1\epsilon_2$ component. To understand how this formulation arises, we can introduce the definitions $\epsilon_1^2 = \epsilon_2^2 = (\epsilon_1\epsilon_2)^2 = 0$ and replace the Taylor series expansion of a function, $f : \mathbb{R}^D \rightarrow \mathbb{R}$ around $\boldsymbol{\theta}$ with perturbation, $\mathbf{d} \in \mathbb{R}^D$,

²More precisely, the basic time complexity of a forward evaluation is constant $\in [2, 2.5]$ times that of the function call [11].

with an evaluation of a hyper-dual number:³

$$f(\boldsymbol{\theta} + \mathbf{d}) = f(\boldsymbol{\theta}) + \nabla f(\boldsymbol{\theta})\mathbf{d} + \frac{1}{2}\mathbf{d}^\top \nabla^2 f(\boldsymbol{\theta})\mathbf{d} + \dots$$

$$f(\boldsymbol{\theta} + \mathbf{v}_1\epsilon_1 + \mathbf{v}_2\epsilon_2 + \mathbf{v}_{12}\epsilon_1\epsilon_2) = f(\boldsymbol{\theta}) + \nabla f(\boldsymbol{\theta})\mathbf{v}_1\epsilon_1 + \nabla f(\boldsymbol{\theta})\mathbf{v}_2\epsilon_2$$

$$+ \nabla f(\boldsymbol{\theta})\mathbf{v}_{12}\epsilon_1\epsilon_2 + \mathbf{v}_1^\top \nabla^2 f(\boldsymbol{\theta})\mathbf{v}_2\epsilon_1\epsilon_2 + \dots.$$

An alternative but isomorphic view is to regard $a + b\epsilon_1 + c\epsilon_2 + d\epsilon_1\epsilon_2 = (a + b\epsilon_1) + (c + d\epsilon_1)\epsilon_2$ as an element of $\mathbb{D}(\mathbb{D}(\mathbb{R}))$, with subscripts to distinguish the inner vs outer \mathbb{D} s; from an implementation perspective, hyperduals can be regarded as inlining the nested structures into a single flat structure.

4.1 Implications of Hyper-Dual Numbers for Machine Learning

A function evaluation with hyper-dual numbers takes an input vector, $\boldsymbol{\theta} + \mathbf{v}_1\epsilon_1 + \mathbf{v}_2\epsilon_2 + \mathbf{0}\epsilon_1\epsilon_2$, with the $\epsilon_1\epsilon_2$ part set to zero. A typical setting to get exact gradient and Hessian elements is to set $\mathbf{v}_1 = \mathbf{e}_i$ and $\mathbf{v}_2 = \mathbf{e}_j$, where \mathbf{e}_i and \mathbf{e}_j are each a basis of one-hot unit vectors. Therefore, these basis vectors select the corresponding elements of the gradient and Hessian:

$$f(\boldsymbol{\theta}) + \nabla f(\boldsymbol{\theta})_i\epsilon_1 + \nabla f(\boldsymbol{\theta})_j\epsilon_2 + \nabla^2 f(\boldsymbol{\theta})_{ij}\epsilon_1\epsilon_2 = f(\boldsymbol{\theta} + \mathbf{e}_i\epsilon_1 + \mathbf{e}_j\epsilon_2 + \mathbf{0}\epsilon_1\epsilon_2).$$

A single loop over the input dimension provides the exact gradient, whereas a nested loop provides the full Hessian. As a side note, a single loop also can give the Hessian vector product. This is done by setting one of the tangent vectors to the chosen vector, and looping through the basis for the other tangent vector.⁴ However, the key advantage is that with a single forward pass, we get both first order and second order information in the form of directional derivatives and the local curvature information respectively. We have already seen from Baydin et al. [1] and follow up works [27, 9] that despite the scalar form of the directional derivative, it can still be used to build optimization routines that appear competitive with backpropagation. In this paper, we investigate whether the additional access to local curvature through forward-mode AD can enable improvements over the current FGD algorithm.

Local Curvature: $\mathbf{v}_1^\top \nabla^2 f(\boldsymbol{\theta})\mathbf{v}_2$. The Hessian contains the curvature information at a point, $\boldsymbol{\theta}$, in the form of the second order partial derivatives. When we evaluate a function over hyper-dual numbers we get the bilinear form, $\mathbf{v}_1^\top \nabla^2 f(\boldsymbol{\theta})\mathbf{v}_2$. This is a function that operates over two vectors, such that it is linear in each vector separately. The bilinear form is common in optimization routines, such as for conjugate gradient descent to assess whether two vectors are conjugate with respect to the Hessian. The value of $\mathbf{v}_1^\top \nabla^2 f(\boldsymbol{\theta})\mathbf{v}_2$ tells us about how curvature covaries along the two vectors. In the case where $\mathbf{v}_1 = \mathbf{v}_2 = \mathbf{v}$, we arrive at the quadratic form that describes the curvature, which indicates the rate of change of the slope as you move in the direction of \mathbf{v} . The quadratic form also provides information on the convexity of the function at that point. If $\mathbf{v}^\top \nabla^2 f(\boldsymbol{\theta})\mathbf{v} > 0, \forall \mathbf{v} \in \mathbb{R}_{>0}^D$, then the function is convex at that point and the Hessian is positive definite. The curvature also indicates the sensitivity of moving in certain directions. For example, when using gradients to optimize a function, taking a gradient step in a region of low curvature is likely to increase the value of the function. However the same sized step in a region of large curvature could significantly change the value of the function (for better or worse).

Computational Cost. The cost of a single forward pass with hyper-dual numbers is of the same order of time and space complexity as the original function call. This reduces the memory cost compared to reverse-mode AD. The forward pass with hyper-dual numbers scales in the same way as the original function call to the number of parameters. However, there is a constant overhead price (no change in computational complexity) to be paid in the form of evaluating the second order terms throughout a forward pass. An example is that the addition of two scalars now requires 4 additions, and multiplication now requires 9 products and 5 additions. However, unlike for reverse-mode, these intermediate values can be overwritten once they have been used to propagate gradient information.

³Note, we have left our function definition as being a scalar output for pedagogical reasons but nothing precludes a vector or matrix output, which is required for the composition of functions in most ML architectures.

⁴While interesting, these results might not immediately seem attractive to the ML community. The Hessian calculation of a model with parameters, $\boldsymbol{\theta} \in \mathbb{R}^D$, would require $D \cdot (D + 1)/2$ function evaluations, whereas Hessian vector products are widely available in many AD libraries leveraging tricks such as a forward over reverse routine [22, 5].

5 Forward-Mode Optimization with Second Order Information

We now introduce the three new optimization routines that incorporate forward-mode second order information:

1. Forward-Mode Line Search (FoMoH)
2. Forward-Mode Line Search with Backpropagation (FoMoH-BP)
3. Forward-Mode Hyperplane Search (FoMoH-KD)

5.1 Forward-Mode Line Search: FoMoH

We introduce a new gradient-based optimization routine leveraging Forward-Mode automatic differentiation with Hessian information (FoMoH). We begin with the one dimensional case, which uses a second-order line search [23, 29]. For a given update direction $\mathbf{v} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ and learning rate η , FoMoH normalizes all gradient steps by the curvature, giving:

$$\boldsymbol{\theta}' = \boldsymbol{\theta} + \eta \frac{\nabla f(\boldsymbol{\theta}) \cdot \mathbf{v}}{|\mathbf{v}^\top \nabla^2 f(\boldsymbol{\theta}) \mathbf{v}|} \mathbf{v}. \quad (1)$$

The directional derivative in the numerator and the curvature (the quadratic form) in the denominator are both provided with **one forward-pass** of the function $f(\boldsymbol{\theta} + \mathbf{v}\epsilon_1 + \mathbf{v}\epsilon_2 + \mathbf{0}\epsilon_1\epsilon_2)$. The normalization via this quadratic form results in accounting for the unit distance at the location \mathbf{x} in the direction \mathbf{v} . Therefore, this update step takes into account the distance metric defined at $\boldsymbol{\theta}$. In regions of high curvature the step size will be smaller, which is a desirable behavior. Like Newton’s method, setting $\eta \approx 1.0$ seems to work in many cases, but by analogy with trust regions we suggest the inclusion of a learning rate, although we found the method to be reasonably robust to its value.

5.2 Forward-Mode Line Search with Backpropagation: FoMoH-BP

A line search starts with identifying a descent direction, followed by determining the value of the step size to move in that direction. Therefore, the second approach that we propose is to perform a line search on the gradient. Thus, this combines forward-mode and reverse-mode to build an optimization routine that provides the step-size for the ground truth gradient obtained from backpropagation. This additional step sets $\mathbf{v} = \nabla f(\boldsymbol{\theta})$ in Equation (1). The result is an update step:

$$\boldsymbol{\theta}' = \boldsymbol{\theta} + \eta \frac{\nabla f(\boldsymbol{\theta}) \cdot \nabla f(\boldsymbol{\theta})}{|\nabla f(\boldsymbol{\theta})^\top \nabla^2 f(\boldsymbol{\theta}) \nabla f(\boldsymbol{\theta})|} \nabla f(\boldsymbol{\theta}). \quad (2)$$

Unlike FoMoH (and FoMoH-KD in the next section), FoMoH-BP includes a single reverse-mode AD evaluation. Specifically, this requires a single backpropagation step followed by forward-mode step that sets the tangent vectors to the gradient.

5.3 Forward-Mode Hyperplane Search: FoMoH-KD

The final algorithm that we propose is the Forward-Mode K -Dimensional Hyperplane Search, FoMoH-KD. Rather than performing a second order line search along direction \mathbf{v} , we perform a K -dimensional hyperplane search. Starting with the $K = 2$ example, if we take two search directions, \mathbf{v}_1 and \mathbf{v}_2 , we can build a 2×2 matrix to form a Hessian in the plane defined by $\boldsymbol{\theta} + \kappa_1 \mathbf{v}_1 + \kappa_2 \mathbf{v}_2$. We evaluate a function, $f(\cdot)$, with a hyper-dual number using the pairs $\{\mathbf{v}_1, \mathbf{v}_1\}$, $\{\mathbf{v}_1, \mathbf{v}_2\}$, and $\{\mathbf{v}_2, \mathbf{v}_2\}$ for the ϵ_1, ϵ_2 coefficients. The result is the Hessian, $\tilde{\mathbf{H}}_{2 \times 2}$, in the 2×2 plane, and corresponding step sizes, κ_1 and κ_2 , to take in each search direction:

$$\tilde{\mathbf{H}}_{2 \times 2} = \begin{bmatrix} \mathbf{v}_1^\top \nabla^2 f(\boldsymbol{\theta}) \mathbf{v}_1 & \mathbf{v}_1^\top \nabla^2 f(\boldsymbol{\theta}) \mathbf{v}_2 \\ \mathbf{v}_2^\top \nabla^2 f(\boldsymbol{\theta}) \mathbf{v}_1 & \mathbf{v}_2^\top \nabla^2 f(\boldsymbol{\theta}) \mathbf{v}_2 \end{bmatrix}, \quad \begin{bmatrix} \kappa_1 \\ \kappa_2 \end{bmatrix} = \tilde{\mathbf{H}}_{2 \times 2}^{-1} \begin{bmatrix} \mathbf{v}_1^\top \nabla f(\boldsymbol{\theta}) \\ \mathbf{v}_2^\top \nabla f(\boldsymbol{\theta}) \end{bmatrix} \quad (3)$$

As a result, we formulate a new update step,

$$\boldsymbol{\theta}' = \boldsymbol{\theta} + \kappa_1 \mathbf{v}_1 + \kappa_2 \mathbf{v}_2.$$

We then extend the above result to any K -dimensional hyperplane by sampling K search directions and evaluating the corresponding update step:

$$\boldsymbol{\theta}' = \boldsymbol{\theta} + \eta \sum_{k=1}^K \kappa_k \mathbf{v}_k. \quad (4)$$

This resulting generalized hyperplane update step allows one to trade-off computational cost with the size of the search space. For example, the cost of evaluating a K -dimensional Hessian and then invert it is $\mathcal{O}(K^3)$, which is feasible for small enough K .⁵ Our new forward-mode hyperplane search, FoMoH- KD , opens up the possibility of transitioning between a line search, when $K = 1$, all the way to a full Newton step, when $K = D$, which we demonstrate in §6.1. The pseudo-code for a single update step is given in Algorithm 1. The overall FoMoH- KD routine is given in Algorithm 2.

Algorithm 1 FoMoH- KD , hyperplane update step for function, f , and parameters, $\theta \in \mathbb{R}^D$.

Define: HyperPlaneStep(f, θ, K)
Set: $N = (K^2 + K)/2$, $\Theta \in \mathbb{R}^{N \times D}$, $V \in \mathbb{R}^{K \times D}$, $V_1 \in \mathbb{R}^{N \times D}$, $V_2 \in \mathbb{R}^{N \times D}$
% For loop vectorized in code.
for $n = 1, \dots, N$ **do**
 $\Theta[n, :] = \theta$ % Repeat current parameter values for vectorized evaluation.
end for
for $k = 1, \dots, K$ **do**
 $V[k, :] = \mathbf{v} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ % Sample K tangent vectors to build hyperplane.
end for
% Vectorize tangent vectors for evaluation of all N elements of $\tilde{\mathbf{H}}_{K \times K}$.
 $l = 1$
for $i = 1, \dots, K$ **do**
 for $j = i, \dots, K$ **do**
 $V_1[l, :] = V[i, :]$
 $V_2[l, :] = V[j, :]$
 $l = l + 1$
 end for
end for
 $\mathbf{z}_0 + \mathbf{z}_1 \epsilon_1 + \mathbf{z}_2 \epsilon_2 + \mathbf{z}_{12} \epsilon_1 \epsilon_2 = f(\Theta + V_1 \epsilon_1 + V_2 \epsilon_2 + \mathbf{0} \epsilon_1 \epsilon_2)$
% Build $\tilde{\mathbf{H}}_{K \times K}$ and directional derivatives vector, $\tilde{\mathbf{G}}_K$, in order to evaluate Eq. (4).
Set: $\tilde{\mathbf{H}}_{K \times K} \in \mathbb{R}^{K \times K}$, $\tilde{\mathbf{G}}_K \in \mathbb{R}^{K \times 1}$
 $l = 1$
for $i = 1, \dots, K$ **do**
 $\tilde{\mathbf{G}}_K[i] = \mathbf{z}_1[l]$
 for $j = i, \dots, K$ **do**
 $\tilde{\mathbf{H}}_{K \times K}[i, j] = \mathbf{z}_{12}[k]$
 $\tilde{\mathbf{H}}_{K \times K}[j, i] = \mathbf{z}_{12}[k]$
 $l = l + 1$
 end for
end for
% Returns update direction of vector size D
return $\sum_k ((-\tilde{\mathbf{H}}_{K \times K}^{-1} \tilde{\mathbf{G}}_K)[k] \cdot V[k, :])$

6 Experiments

6.1 Rosenbrock Function

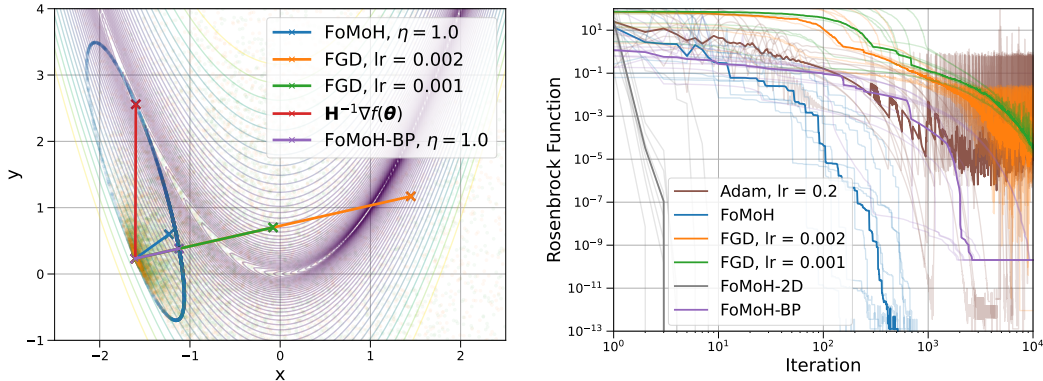
In this section, we test FoMoH and its variants on the Rosenbrock function [28] which has a global minimum at $f(\mathbf{1}) = 0$. The Rosenbrock function, $f(\theta) = \sum_{i=1}^{D-1} (100(\theta_{i+1} - \theta_i^2)^2 + (1 - \theta_i)^2)$, is designed to be a challenging test case for non-convex optimization where the solution falls inside a narrow valley. The learning rate for the FoMoH variants is set to 1.0 for the Rosenbrock experiments.

As an initial illustration of the behavior of each forward-mode approach, we show how a single step looks from a randomly chosen starting point in Figure 1a for FGD, FoMoH, and FoMoH-BP. We plot the expected (average) step across 10,000 samples for all approaches in the 2D Rosenbrock function. These steps are shown with solid lines. For each approach, we also plot the sampled steps

⁵For instances where $\tilde{\mathbf{H}}_{K \times K}$ is not invertible, we add jitter to the diagonal. This seems to work well.

by superimposing a scatter plot in the corresponding approach’s color. All methods are compared to the Newton step, $(\nabla^2 f(\boldsymbol{\theta}))^{-1} \nabla f(\boldsymbol{\theta})$ shown in red. For FGD, we see that the expected descent direction is the same as the gradient at that point, hence the alignment with FoMoH-BP that directly calculates the gradient. This plot highlights the reliance on a well-chosen learning rate for FGD, whereas FoMoH-BP’s step size is automatically normalized by the local curvature along the gradient. FoMoH (blue), on the otherhand, has a descent direction that differs from the gradient and is governed by the distribution of samples that fall on the ellipse defined by the Hessian, $\nabla^2 f(\boldsymbol{\theta})$. For this point, the Newton step is the descent direction that falls on this ellipse and corresponds to the local minimum of the quadratic approximation. Another insight gained from this figure is that the variance of FGD’s descent direction is less constrained than FoMoH’s descent direction (blue), where the sample direction is controlled by the local Hessian. As a final note, we do not plot FoMoH-2D in Figure 1a as it directly aligns with the Newton step, with very little variance (see §A, Figure 5).

In Figure 1b, we now compare the performance of the competing optimization routines for the 2D Rosenbrock function. These results are shown for the same 10 randomly sampled starting locations for all approaches initialized with different random seeds. We highlight with the thicker line the median performing run. Both axes are on the log scale and show the significant advantage of FoMoH-KD, with $K = 2$. We also note the advantage (at least for this 2D example) of all FoMoH approaches that use second-order information. Additionally, the two forward-mode only approaches actually outperform the optimization routines that include backpropagation.



(a) Expected step taken by the stochastic approaches of FoMoH and FGD. Included is a single Newton step for reference. The samples show how the curvature constrains the step size, compared to the sensitivity of FGD to the learning rate.

(b) Comparison of the stochastic approaches in minimization of the Rosenbrock function. Average performance (median) is shown over 10 random initial conditions. FoMoH outperforms all first-order approaches, with FoMoH-2D converging in orders of magnitude faster than all methods.

Figure 1: Results over the 2D Rosenbrock function.

10D Rosenbrock Function. We now focus on the performance of FoMoH-KD as we increase K from 2 to the input dimension of the function. Figure 2 shows this comparison for the 10D Rosenbrock function, where we use 10 random initializations for the different K . The median performance for each K is then shown, where we see a perfect ordering of performance that aligns with the dimension of the hyperplane. The best performing FoMoH-KD is for $K = D$, with the worst corresponding to the lowest dimension implemented, $K = 2$. Overall this figure highlights how FoMoH-KD trends towards Newton’s method as K tends to D , where we actually see the median performances of FoMoH-10D and Newton’s method aligned.

6.2 Logistic Regression

We now compare the performance of FoMoH for logistic regression applied to the MNIST dataset [16]. Table 1 displays mean and standard deviation performance for each approach, where the forward-mode-only approaches are highlighted separately from the methods that include reverse-mode steps. Additional details on hyperparameter optimization are included in Appendix A.3. Figure 3 displays the training and validation curves for both accuracy and negative log-likelihood (NLL) for the forward-mode approaches (Figure 6 in appendix includes additional reverse-mode

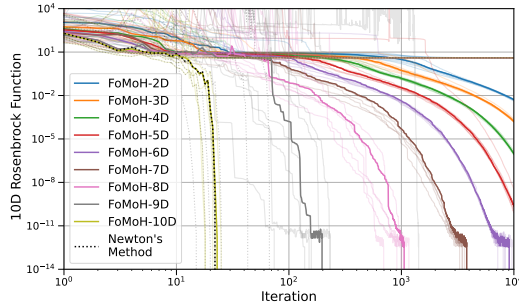


Figure 2: Performance of FoMoH- KD for $K = 2 \dots 10$ on the 10D Rosenbrock function. Solid lines represent the median, with transparent lines corresponding to the each of the 10 random seeds. There is a clear pattern of higher dimensions performing better, with the performance of $K = 10$ coinciding with Newton’s Method (black dotted line).

approaches). We see an improvement in speed of convergence as K increases. However, we also see that FoMoH and FoMoH- KD , with a fixed learning rate, degrade in performance after reaching their local minimum (NLL) or maximum (accuracy). We therefore introduce a learning rate scheduler to improve on this behavior. For this task, FGD is competitive with the FoMoH variants but is slower to converge. In §6.3 we show how FGD degrades in performance for a larger parameter space.

Table 1: Logistic regression results for MNIST. When comparing the forward-mode-only approaches in the upper section of the table, we see improvement in performance with increasing hyperplane dimension for FoMoH- KD . For this logistic regression example, FoMoH-3D and FoMoH-2D with learning rate schedulers, are competitive with FGD. However we will see this result change with a larger dimensional problem in §6.3. Both reverse-mode approaches in the lower section of the table have similar performance, and are included for reference.

APPROACH	TRAINING LOSS	VALIDATION LOSS	TRAINING ACCURACY	VALIDATION ACCURACY
FGD	0.2976 ± 0.0007	0.2949 ± 0.0017	0.9154 ± 0.0003	0.9163 ± 0.0019
FoMoH	0.3223 ± 0.0013	0.3186 ± 0.0019	0.9073 ± 0.0011	0.9110 ± 0.0021
FoMoH (LR-SCH.)	0.3192 ± 0.0012	0.3160 ± 0.0025	0.9085 ± 0.0011	0.9118 ± 0.0021
FoMoH-2D	0.3010 ± 0.0018	0.3015 ± 0.0031	0.9144 ± 0.0009	0.9149 ± 0.0015
FoMoH-2D (LR-SCH.)	0.2921 ± 0.0014	0.2951 ± 0.0027	0.9174 ± 0.0005	0.9170 ± 0.0010
FoMoH-3D	0.3449 ± 0.0017	0.3343 ± 0.0034	0.8999 ± 0.0008	0.9036 ± 0.0023
FoMoH-3D (LR-SCH.)	0.2893 ± 0.0017	0.3054 ± 0.0034	0.9195 ± 0.0007	0.9153 ± 0.0010
FoMoH-BP	0.2312 ± 0.0001	0.2679 ± 0.0003	0.9366 ± 0.0001	0.9267 ± 0.0002
BACKPROPAGATION	0.2265 ± 0.0000	0.2710 ± 0.0001	0.9381 ± 0.0001	0.9267 ± 0.0003

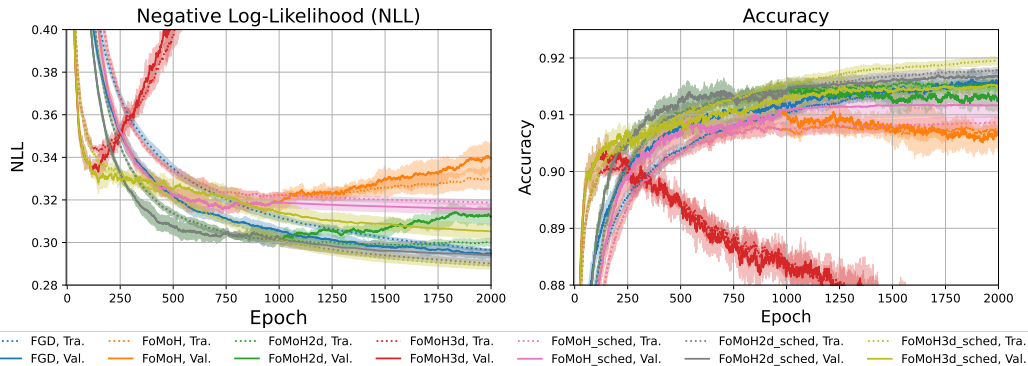


Figure 3: Forward-mode training and validation curves for the logistic regression model on the MNIST dataset. Average and standard deviation is shown for five random initializations.

6.3 Convolutional Neural Network

We now move from a model with 7,850 parameters to a convolutional neural network (CNN) with 431,080 parameters. As before we use the MNIST dataset and leave the details on hyperparameter optimization using Bayesian optimization to the Appendix A.4. Both Table 2 and Figure 4 highlight the advantage of FoMoH-KD over FGD. For the larger parameter space FGD requires more epochs to converge compared to all FoMoH variants. The learning rate scheduler further improves FoMoH and FoMoH-KD by helping to avoid getting stuck in low performance regions. Here, we see the clear trend that the best performing forward-mode-only approach comes from the largest K , which was $K = 3$ for this experiment. As expected, both optimizers FoMoH-BP and Backpropagation outperform the forward-mode-only approaches. Overall, these results highlight that second-order information helps scale the performance of forward-mode optimization to larger dimensions.

Table 2: CNN results for MNIST. The forward-mode-only approaches in the upper section of the table show that FoMoH’s performance improves with the dimension of K , especially when used with the learning rate scheduler. FoMoH-3D outperforms FoMoH-2D, FoMoH, and FGD. The reverse-mode approaches in the lower section outperform forward-mode, with BP slightly better than FoMoH-BP.

APPROACH	TRAINING LOSS	VALIDATION LOSS	TRAINING ACCURACY	VALIDATION ACCURACY
FGD	0.1211 ± 0.0097	0.1104 ± 0.0086	0.9641 ± 0.0038	0.9677 ± 0.0030
FoMoH	0.1663 ± 0.0069	0.1571 ± 0.0092	0.9518 ± 0.0011	0.9550 ± 0.0002
FoMoH (LR-SCH.)	0.1617 ± 0.0119	0.1575 ± 0.0158	0.9515 ± 0.0035	0.9539 ± 0.0034
FoMoH-2D	0.1015 ± 0.0041	0.1016 ± 0.0066	0.9691 ± 0.0011	0.9693 ± 0.0020
FoMoH-2D (LR-SCH.)	0.0900 ± 0.0070	0.0913 ± 0.0041	0.9731 ± 0.0022	0.9718 ± 0.0014
FoMoH-3D	0.1085 ± 0.0105	0.1073 ± 0.0133	0.9674 ± 0.0022	0.9686 ± 0.0028
FoMoH-3D (LR-SCH.)	0.0809 ± 0.0061	0.0923 ± 0.0132	0.9759 ± 0.0014	0.9734 ± 0.0022
FoMoH-BP	0.0093 ± 0.0016	0.0310 ± 0.0006	0.9981 ± 0.0005	0.9903 ± 0.0003
BACKPROPAGATION	0.0053 ± 0.0034	0.0329 ± 0.0032	0.9990 ± 0.0009	0.9909 ± 0.0004

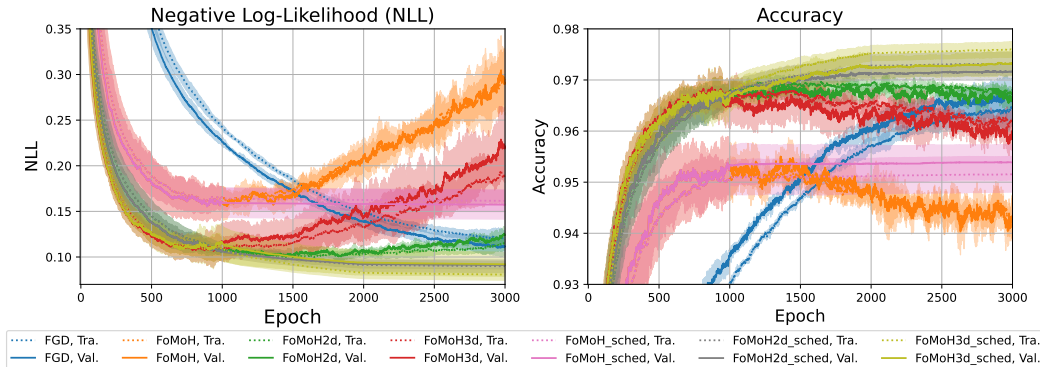


Figure 4: Forward-mode training and validation curves for the CNN on the MNIST dataset. Average and standard deviation is shown for three random initializations. Note how FGD (blue) is much slower to converge, with FoMoH-KD improving in performance with increasing K .

7 Conclusion, Broader Impact, and Limitation Discussion

The results in §6 highlight the potential of the use of second-order forward-mode AD for optimization tasks. For the Rosenbrock function, we illustrated the behavior of our three new optimization routines: FoMoH, FoMoH-BP, and FoMoH-KD, and we compared them to Newton’s method. In particular we were able to show that as one increases the hyperplane dimension of FoMoH-KD the method tends to Newton’s method, without the need for any backpropagation. This significant result is shown in Figure 2. For the learning tasks of logistic regression and CNN classification, we see how the first-order optimization approach of FGD degrades with increasing dimension of the parameter space. We do not see this degradation for FoMoH-KD, and we also observe that the second-order information means fewer epochs are needed to reach a better performance. This has the broader impact of improving efficiency, reducing cost, and increasing accuracy in ML optimization routines.

In conclusion, we introduced a novel approach that uses second-order forward-mode AD for optimization. We have introduced: forward-mode line search (FoMoH); forward-mode line search with Backpropagation (FoMoH-BP); and forward-mode hyperplane search (FoMoH-KD). We have shown how these approaches compare to the previous first-order forward-mode approach of FGD, as well stochastic gradient descent for multiple optimization problems over a wide range of dimensions. Furthermore, FoMoH-KD behaves closer to the performance of Newton’s method as K increases. In addition to contributing the new second-order forward-mode optimization routines, we provide a Python package that implements the AD backend and interfaces with PyTorch. Our work is a further step in the direction of showing the value of cheap second-order information in optimization with the need to scale to even larger dimensions. We expect that future work will be able to mix the advantages gained from first-order approaches with that of second-order approaches.

Acknowledgments and Disclosure of Funding

This material is based upon work supported by the United States Air Force and DARPA under Contract No. FA8750-23-C-0519. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the United States Air Force and DARPA.

References

- [1] Atılım Güneş Baydin, Barak A Pearlmutter, Don Syme, Frank Wood, and Philip Torr. Gradients without backpropagation. *arXiv preprint arXiv:2202.08587*, 2022.
- [2] Sue Becker, Yann Le Cun, et al. Improving the convergence of back-propagation learning with second order methods. In *Proceedings of the 1988 connectionist models summer school*, pages 29–37, 1988.
- [3] Yoshua Bengio, Dong-Hyun Lee, Jorg Bornschein, Thomas Mesnard, and Zhouhan Lin. Towards biologically plausible deep learning. *arXiv preprint arXiv:1502.04156*, 2015.
- [4] Lukas Biewald. Experiment tracking with weights and biases, 2020. URL <https://www.wandb.com/>. Software available from wandb.com.
- [5] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018. URL <https://github.com/google/jax>.
- [6] Mathieu Dagr eou, Pierre Ablin, Samuel Vaiter, and Thomas Moreau. How to compute Hessian-vector products? In *ICLR Blogposts 2024*, 2024. URL <https://d2jud02ci9yv69.cloudfront.net/2024-05-07-bench-hvp-81/blog/bench-hvp/>.
- [7] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, 12(7), 2011.
- [8] Jeffrey Fike and Juan Alonso. The development of hyper-dual numbers for exact second-derivative calculations. In *49th AIAA aerospace sciences meeting including the new horizons forum and aerospace exposition*, page 886.
- [9] Louis Fournier, St ephane Rivaud, Eugene Belilovsky, Michael Eickenberg, and Edouard Oyallon. Can forward gradient match backpropagation? In *International Conference on Machine Learning*, pages 10249–10264. PMLR, 2023.
- [10] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. URL <http://www.deeplearningbook.org>.
- [11] Andreas Griewank and Andrea Walther. *Evaluating derivatives: principles and techniques of algorithmic differentiation*. SIAM, 2008.
- [12] Geoffrey Hinton. The forward-forward algorithm: Some preliminary investigations. *arXiv preprint arXiv:2212.13345*, 2022.

- [13] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, San Diego, CA, USA, 2015.
- [14] Yann Le Cun. Modèles connexionnistes de l'apprentissage. *Intellectica*, 2(1):114–143, 1987.
- [15] Nicolas Le Roux and Andrew W Fitzgibbon. A fast natural Newton method. In *ICML*, pages 623–630, 2010.
- [16] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.
- [17] Yann LeCun, Léon Bottou, Genevieve B Orr, and Klaus-Robert Müller. Efficient backprop. In *Neural networks: Tricks of the trade*, pages 9–50. Springer, 1998.
- [18] Kenneth Levenberg. A method for the solution of certain non-linear problems in least squares. *Quarterly of applied mathematics*, 2(2):164–168, 1944.
- [19] Hong Liu, Zhiyuan Li, David Hall, Percy Liang, and Tengyu Ma. Sophia: A scalable stochastic second-order optimizer for language model pre-training. *arXiv preprint arXiv:2305.14342*, 2023.
- [20] Donald W Marquardt. An algorithm for least-squares estimation of nonlinear parameters. *Journal of the society for Industrial and Applied Mathematics*, 11(2):431–441, 1963.
- [21] James Martens. New insights and perspectives on the natural gradient method. *Journal of Machine Learning Research*, 21(146):1–76, 2020.
- [22] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.
- [23] Barak A Pearlmutter. Fast exact multiplication by the Hessian. *Neural Computation*, 6(1): 147–160, 1994.
- [24] Barak A Pearlmutter and Jeffrey Mark Siskind. Lazy multivariate higher-order forward-mode AD. *ACM SIGPLAN Notices*, 42(1):155–160, 2007.
- [25] D Pierangeli, G Marcucci, and C Conti. Large-scale photonic ising machine by spatial light modulation. *Physical review letters*, 122(21):213902, 2019.
- [26] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.
- [27] Mengye Ren, Simon Kornblith, Renjie Liao, and Geoffrey Hinton. Scaling forward gradient with local losses. *arXiv preprint arXiv:2210.03310*, 2022.
- [28] H. Rosenbrock. An automatic method for finding the greatest or least value of a function. *The Computer Journal*, 3(3):175–184, 1960.
- [29] N.N. Schraudolph and T. Graepel. Towards stochastic conjugate gradient methods. In *Proceedings of the 9th International Conference on Neural Information Processing, 2002. ICONIP '02.*, volume 2, pages 853–856 vol.2, 2002. doi: 10.1109/ICONIP.2002.1198180.
- [30] David Silver, Anirudh Goyal, Ivo Danihelka, Matteo Hessel, and Hado van Hasselt. Learning by directional gradient descent. In *International Conference on Learning Representations*, 2021.
- [31] Tijmen Tieleman and Geoffrey Hinton. RmsProp: Divide the gradient by a running average of its recent magnitude. coursera: Neural networks for machine learning. *COURSERA Neural Networks Mach. Learn*, 17, 2012.
- [32] Oriol Vinyals and Daniel Povey. Krylov subspace descent for deep learning. In *Artificial intelligence and statistics*, pages 1261–1268. PMLR, 2012.

- [33] Robert Edwin Wengert. A simple automatic derivative evaluation program. *Communications of the ACM*, 7(8):463–464, 1964.
- [34] Zhewei Yao, Amir Gholami, Sheng Shen, Mustafa Mustafa, Kurt Keutzer, and Michael Mahoney. Adahessian: An adaptive second order optimizer for machine learning. In *proceedings of the AAAI conference on artificial intelligence*, volume 35, pages 10665–10673, 2021.

A Additional Results

A.1 FoMoH- K D Algorithm

Algorithm 2 FoMoH- K D

Require: Objective function $f(\theta)$, initial point θ , step size η , hyperplane dimension K .
for $t = 0, 1, 2, \dots$ until convergence **do**
 $\mathbf{d} = \text{HyperPlaneStep}(f, \theta, K)$ % See Algorithm 1
 $\theta = \theta + \eta \mathbf{d}$
end for

A.2 Rosenbrock Function

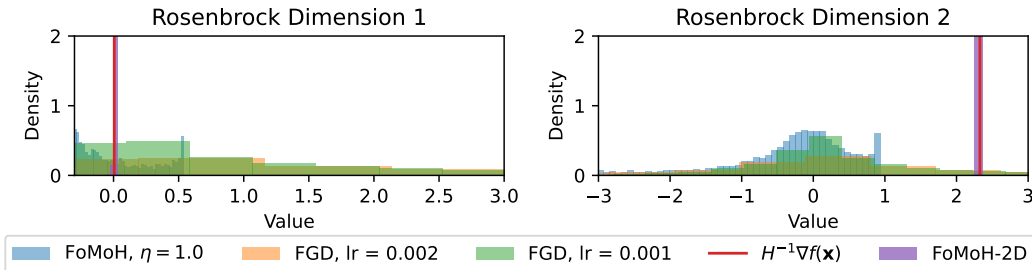


Figure 5: Histogram over expected step taken by the stochastic approaches of FoMoH, FGD, and FoMoH-2D corresponding to Figure 1a. Noteworthy is that the variance of the 2D hyperplane search step is significantly smaller and expectation is close to Newton step.

A.3 Logistic Regression

Table 3 includes the final hyperparameter selection for the experimental results in §6.2. We used [4] to perform a grid search with 100 iterations, where the batch size choice was between [128, 512, 1024, 2048]. For the learning rate scheduler, we reduced the learning rate at the epoch where the NLL starts to increase. For FoMoH-3D we multiplied the learning rate by 0.8, whereas for the other approaches we multiplied the learning rate by 0.1. There is likely room for improvement on the parameters of the learning rate scheduler, but that would only improve the current results.

Figure 6 includes the reverse-mode training and validation curves for Backpropagation and FoMoH-BP in addition to the curves shown in Figure 3.

Table 3: Hyperparameter Optimization for Logistic Regression.

APPROACH	LEARNING RATE	LEARNING RATE BOUNDS	BATCH SIZE
FGD	0.00006497	[0.00001, 0.1]	128
FoMoH	0.1362	[0.001, 1.0]	1024
FoMoH (LR-SCH.)	0.1362	[0.001, 1.0]	1024
FoMoH-2D	0.04221	[0.001, 1.0]	512
FoMoH-2D (LR-SCH.)	0.04221	[0.001, 1.0]	512
FoMoH-3D	0.1	[0.001, 1.0]	512
FoMoH-3D (LR-SCH.)	0.1	[0.001, 1.0]	512
FoMoH-BP	0.04688	[0.01, 1.0]	2048
BACKPROPAGATION	0.03561	[0.01, 0.5]	2048

A.4 CNN

Table 4 includes the final hyperparameter selection for the experimental results in §6.3. We used [4] to perform Bayesian optimization with 100 iterations, where the batch size choice was fixed to 2048. For

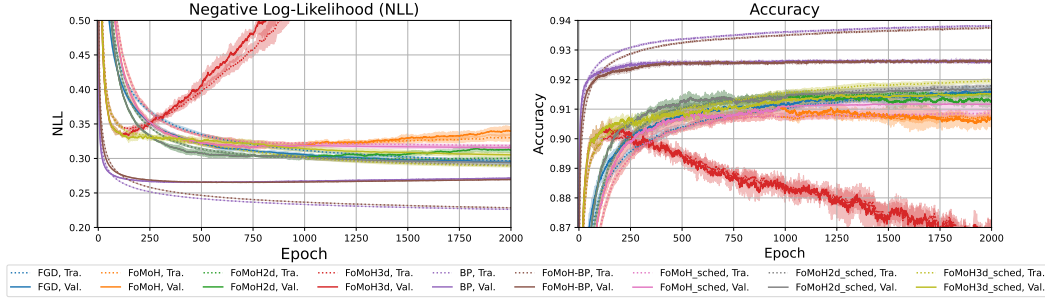


Figure 6: Training and validation curves for logistic regression model on the MNIST dataset. Average and standard deviation is shown for five random initializations.

FoMoH-3D, we used the same hyperparameters as for FoMoH-2D as this gave sufficient performance (and still outperformed the other forward-mode approaches). All learning rate schedulers reduced the learning rate by 10 every 1000 epochs.

Figure 7 includes the reverse-mode training and validation curves for Backpropagation and FoMoH-BP in addition to the curves shown in Figure 4.

Table 4: Hyperparameter Optimization for Logistic Regression.

APPROACH	LEARNING RATE	LEARNING RATE BOUNDS	BATCH SIZE
FGD	0.0001376	[0.00001, 0.1]	2048
FoMoH	0.542	[0.001, 1.0]	2048
FoMoH (LR-SCH.)	0.542	[0.001, 1.0]	2048
FoMoH-2D	0.3032	[0.001, 1.0]	2048
FoMoH-2D (LR-SCH.)	0.3032	[0.001, 1.0]	2048
FoMoH-3D	0.3032	[0.001, 1.0]	512
FoMoH-3D (LR-SCH.)	0.3032	[0.001, 1.0]	2048
FoMoH-BP	0.04688	[0.01, 1.0]	2048
BACKPROPAGATION	0.03561	[0.005, 0.2]	2048

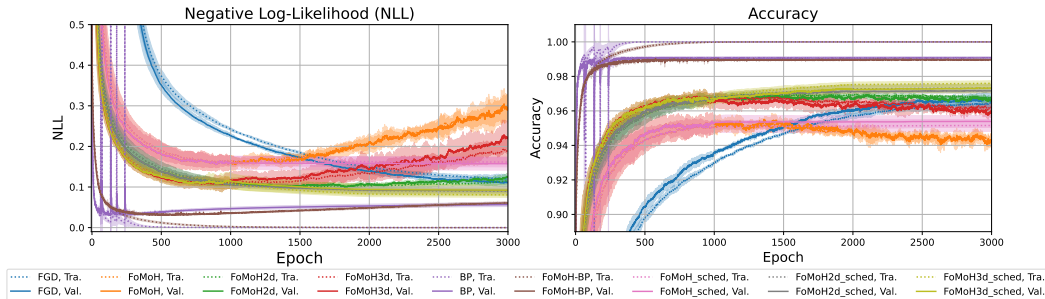


Figure 7: Training and validation curves for CNN on the MNIST dataset. Average and standard deviation is shown for three random initializations.

A.5 Computational Resources

All experiments are run on a NVIDIA RTX 6000 GPU. The main compute cost came from both the grid search and the Bayesian optimization that we ran over the six different optimization routines for the different experiments.