Master's Thesis

submitted to Software and Tools for Computational Engineering (STCE) at RWTH Aachen University

Efficient Higher-Order Sobolev Training via Randomized Hessian Sketching

Author: Johannes Wilhelm

Matriculation Number: 402438

Degree Program: Master of Science in Computer Science

Examiners:

1. Univ.-Prof. Dr. rer. nat. Uwe Naumann
   Software and Tools for Computational Engineering
   RWTH Aachen University

2. Univ.-Prof. Dr. Sebastian Trimpe
   Data Science in Mechanical Engineering
   RWTH Aachen University

Aachen, December 12, 2025

Acknowledgements

First, I would like to thank my supervisor, Neil Kichler, for his guidance, feedback, and support throughout the writing of this thesis.

I would also like to thank Prof. Dr. Uwe Naumann and Prof. Dr. Sebastian Trimpe for serving as examiners of this thesis.

Finally, I would like to express my gratitude to my parents for giving me the opportunity to pursue my studies, and to my friends for making it a wonderful time.

Thank you!

Abstract

Neural network surrogate models provide a way to accelerate evaluations of computationally expensive reference functions. Sobolev training can improve surrogate accuracy by supervising derivatives with respect to the inputs, but full second- and higher-order supervision is impractical in high dimensions due to the quadratic (and worse) scaling of Hessians and higher-order derivative tensors. We combine randomized low-rank approximation with algorithmic differentiation to estimate the dominant low-dimensional subspaces of the Hessian and third-order derivatives, enabling directional supervision without materializing these objects. We propose batch-wise and streaming methods for direction selection and adapt a task-uncertainty-based loss balancing mechanism to stabilize training. Across analytic and Monte Carlo reference functions, our methods improve accuracy over random-direction baselines and reduce training cost compared to training with full higher-order derivatives, while remaining robust when derivative information is noisy.

Contents

1 Introduction

Fast and accurate evaluation of computationally expensive functions is important in many areas of science and engineering. Examples include physics simulations that solve partial differential equations on fine grids or Monte Carlo (MC) approximations of stochastic models. Repeated evaluations can become prohibitively expensive, motivating the construction of surrogate models that approximate reference functions at lower computational cost while maintaining high accuracy.

Neural networks (NN) are natural candidates for such surrogates. In standard supervised learning they are trained by supervising only function values. In the case of surrogate training, however, derivatives of both the reference model and the surrogate are often available efficiently via Algorithmic Differentiation (AD). Sobolev training, introduced by Czarnecki et al. [6], makes full use of that available information by supervising values and their derivatives with respect to the inputs. They demonstrated that incorporating gradients in the loss can improve sample efficiency and the accuracies of predicted values and learned sensitivities, which are of primary interest in applications such as financial risk management.

However, extending the idea to second derivatives poses practical challenges because Hessians scale quadratically with the input dimension. Recent work [19] therefore considers directional second-order training, supervising curvature only along a few directions via Hessian-vector products (HVPs). The difficulty lies in selecting a small number of informative supervision directions that maximize the information provided to the surrogate.

Supervising all coordinate directions is too expensive, while selecting directions at random ignores the structure of the underlying Hessian. To the best of our knowledge, [19] is the only work that studies second-order directional Sobolev training for NNs, selecting directions via principal components of gradients as a proxy for curvature. Their evaluations of second-order supervision focus primarily on MC approximations of the Bachelier model, reporting limited success on the Heston model due to high levels of noise in the second-order derivative labels. Since the Bachelier model has only a single dominant curvature direction at any point, it remains unclear how well these methods extend to reference functions with richer curvature.

In this work, we explore techniques from Randomized Numerical Linear Algebra (RandNLA). Randomized sketching methods are used to approximate the low rank structure of large matrices. Combined with HVP evaluations via AD, this allows us to approximate leading curvature directions efficiently, without materializing full Hessians. We further extend the approach to directional third-order information.

Contributions:

- We propose three algorithms to select informative supervision directions: a batch-wise second-order method, a streaming version that aggregates curvature information across batches, and a third-order extension.
- We provide empirical evaluations against value-based training, first-order Sobolev training, full Hessian second-order supervision, random-direction second-order training, and the PCA of gradients approach proposed in [19]. Experiments are conducted using both analytic and MC approximated labels.
- We adapt a task-uncertainty based loss balancing mechanism and show improved stability for higher-order Sobolev training, especially under noisy derivative labels.
- We provide a Python/JAX framework for training NN surrogates on custom analytic or stochastic reference functions supporting all training methods discussed in this work.

Across a range of reference functions, our methods achieve accuracies close to training with full Hessians. They outperform curvature supervision of random directions in accuracy, while remaining more efficient than supervising all coordinate directions for computationally expensive and high-dimensional reference functions. In Monte Carlo based experiments we observe that the quality of derivative labels is crucial, especially for third-order supervision. When sampling noise is too high, higher-order supervision can degrade overall performance. In these scenarios, the proposed balancing mechanism adapts by downweighting the noisier components, mitigating adverse effects.

The remainder of this thesis is structured as follows. Chapter 2 reviews the foundations on Sobolev training and RandNLA needed for the proposed methods. Chapter 3 introduces directional Sobolev training in detail and presents our direction-selection approaches for higher-order supervision. Chapter 4 reports the evaluation setup and discusses the results before a conclusion is given in Chapter 5. Additional materials are collected in the Appendix A, including the reference models used in our evaluations A.5, the proposed loss-balancing mechanism A.4, and supplementary background on NNs A.1 and AD A.2.

2 Background

2.1 Sobolev Training

In standard supervised regression, a target function $f : \mathbb{R}^d \to \mathbb{R}$ is approximated by a parametrized predictor $\hat{f}_\theta$ (often a neural network) by minimizing the empirical risk over a given set of input-output pairs $\{(x_i, y_i)\}_{i=1}^m$:

$$\min_\theta \frac{1}{m} \sum_{i=1}^m \mathcal{L}\left(\hat{f}_\theta\left(x_i\right), y_i\right),$$

where the loss function $\mathcal{L}$ quantifies the discrepancy between the network's prediction and the target. A standard choice for the loss function is the squared error $\|\hat{y} - y\|_2^2$. The minimization is typically realized through iterative updates to the model's parameters θ via stochastic gradient descent. For a more detailed look at neural networks and their training procedures, we refer the reader to section A.1 of the appendix.
Sobolev training departs from this value-only supervision by also aligning the derivatives of $\hat{f}_\theta$ with those of the reference function f. By doing so, each sample constrains not just a pointwise value but also local function behavior. The Sobolev loss, first introduced by Czarnecki et al. [6], aims to minimize the error of predicted values and learned derivatives up to a chosen order.

Definition 1 (Sobolev Loss) *Let $f : \mathbb{R}^d \to \mathbb{R}$ be a reference function and $\hat{f}_\theta$ a predictor. The Sobolev loss up to order n at an input x is defined as*

$$\mathcal{L}_{Sobolev}(\theta; x) = \sum_{i=0}^n \lambda_i \left\| \nabla_x^i \hat{f}_\theta(x) - \nabla_x^i f(x) \right\|_2^2, \tag{2.1}$$

where ∇_x^i denotes the i-th derivative with respect to x (with $\nabla_x^0 g := g$), $\lambda_i \geq 0$ are balancing factors.

Balancing multiple loss terms of different orders is a non-trivial problem itself. We refer the reader to section A.4 of the appendix for an explanation of our approach. Czarnecki et al. demonstrated the idea on analytic reference functions and in model distillation, where a small neural network learns to approximate the function of a larger pretrained one. In quantitative finance, Huge and Savine [14] applied the same principle to stochastic pricing models under the name Differential Machine Learning. While both studies were limited to first-order supervision, they showed that incorporating derivative information in the loss can improve sample efficiency and generalization.

2.2 Randomized Numerical Linear Algebra

This chapter introduces the main ideas of Randomized Numerical Linear Algebra (RandNLA). Randomized methods extract the low-rank structure of large matrices to compute approximate solutions faster than deterministic algorithms. A key advantage of RandNLA is the control it offers over computational cost and approximation accuracy. This makes it well suited for applications in NN training, where exact solutions are not necessarily needed since training itself is an approximation process.

Sketching:

A sketch of a matrix A is a smaller matrix that preserves properties of A that are relevant for a downstream task. It is formed through the multiplication of A with a test matrix S [26].

Definition 2 (Matrix Sketch) *Let $A \in \mathbb{R}^{m\times n}$. Then*

$$Y = AS \quad \text{with } S \in \mathbb{R}^{n\times s}, \quad s < n$$

forms a right sketch that compresses columns and is useful for estimating left singular vectors. Similarly,

$$Y = SA \quad \text{with } S \in \mathbb{R}^{s\times m}, \quad s < m$$

forms a left sketch that compresses rows and is useful for estimating right singular vectors.

Typical choices for S include i.i.d. Gaussian or Rademacher matrices, in which case sketching can be interpreted as a randomized projection of A onto a lower-dimensional subspace. The sketch size s is often chosen slightly larger than the desired rank k of the sketch, with $s = k+p$ where $p \geq 0$ is an oversampling parameter [10].

Low-Rank Approximation:

The *range* of a matrix $A \in \mathbb{R}^{m\times n}$ is the subspace $\text{Range}(A) = \{Ax : x \in \mathbb{R}^n\} \subseteq \mathbb{R}^m$, i.e., the span of its columns. The *rank* of A is the dimension of this subspace, $\text{rank}(A) = \dim(\text{Range}(A))$ or equivalently, the number of linearly independent columns (or rows). If $\text{rank}(A)$ is smaller than m or n, then a close approximation of A can be stored more compactly. Given $A \in \mathbb{R}^{m\times n}$, the rank-$k$ approximation problem is to find a matrix $\tilde{A}$ with $\text{rank}\left(\tilde{A}\right) \leq k$ that minimizes the approximation error in a given norm

$$\tilde{A}_k \in \underset{\text{rank}(\tilde{A})\leq k}{\arg\min} \|A - \tilde{A}\|.$$

The Eckart-Young theorem [8] shows that for the Frobenius norm $\|\cdot\|_F$ and the spectral norm $\|\cdot\|_2$, the optimal solution is given by a truncated Singular Value Decomposition (SVD)

$$A_k = U_k \Sigma_k V_k^\top,$$

where Σ_k contains the k largest singular values and U_k, V_k the corresponding singular vectors.

Randomized methods aim to find a solution close to this optimum while being cheaper to compute. The main idea is the randomized range finder [10].

Algorithm 1 Randomized range finder for low-rank approximation (Halko et al. [10])

```
Require: $A \in \mathbb{R}^{m \times n}$, target rank $k$, oversampling $p$, power steps $q$
1: Set $s \leftarrow k + p$
2: Draw a random test matrix $\Omega \in \mathbb{R}^{n \times s}$ (e.g., i.i.d. Gaussian)
3: Form the sketch $Y \leftarrow (AA^\top)^q A\Omega \in \mathbb{R}^{m \times s}$
4: Compute an orthonormal basis $Q$ for the range of $Y$ via QR-decomposition: $Y = QR$
5: Form the compressed matrix $B \leftarrow Q^\top A \quad \in \mathbb{R}^{s \times n}$
6: Return $\tilde{A} \leftarrow QB$
```

Then $\tilde{A} = QQ^\top A$ is low rank approximation of A with $\text{rank}(\tilde{A}) \leq s$. It can be beneficial to apply a few steps of power-iteration when the singular values of A decay slowly. A few steps, e.g. $q = 1$ or $q = 2$, can already lead to improved approximation accuracy [10]. When using power-iteration, it is recommended to re-orthonormalize between steps to maintain numerical stability. Having obtained the compressed representation B, one can perform downstream computations on it and lift the results back by multiplication with Q. The results are approximations of having done the same computations on the larger matrix A, restricted to the subspace $\text{Range}(Q)$. In section 3.2 we will cover the singular value decomposition as a relevant example for such downstream operations.

2.3 Literature Survey

Sobolev training was introduced by Czarnecki et al. in 2017 [6], who demonstrated that supervising first derivatives can improve sample efficiency and accuracy. In computational finance, Huge and Savine formulated the same principle under the name Differential Machine Learning (DML), training NN surrogates on MC simulated pricing functions with pathwise derivatives [14]. Cocola and Hand [4] provided theoretical guarantees for directional first-order Sobolev training, establishing that directional supervision can suffice to constrain the target function. Extending DML to second order, Kichler [19] supervised directional curvature via HVPs, selecting supervision directions via a Principle Component Analysis (PCA) of gradients. The efficient HVP computations follow Pearlmutter [30], and the underlying AD framework is detailed in [9] and [27].
RandNLA provides algorithms to approximate dominant low-rank structure in large matrices. Halko et al. [10] introduced the Randomized SVD algorithm, with broader analysis and survey of sketching methods by Martinsson and Tropp [25] and Woodruff [35].
A practical concern for Sobolev training is the balancing of loss terms of different orders. Huge and Savine proposed a normalization per number of supervised quantities, which was adapted by Kichler for second-order training. Kilicsoy et al. [22] propose residual weighting schemes for Sobolev training in mechanics applications. Kendall et al. [17] introduced uncertainty-based adaptive weighting for multi-task learning in computer vision, which we adopt for higher-order Sobolev training in this work.
Adjacent work includes Jacobian matching as input-noise regularization in model distillation [32] and derivative-informed operator learning for partial differential equations [29].

3 Higher-Order Supervision

Sobolev training with first-order derivatives has been shown to improve sample efficiency and generalization in settings such as quantitative finance and model distillation [14, 6]. Extensions to higher orders, however, pose computational challenges [11, 19]. Already at second-order, matching full Hessians is infeasible for functions with many inputs. In this chapter, we present techniques that leverage directional derivative information to enable higher-order supervision without explicitly forming full Hessians. We further extend the approach to incorporate directional third-order derivatives.

3.1 Directional Sobolev Training

For a scalar valued function $f(x)$ with input $x \in \mathbb{R}^d$, its Hessian at x is the matrix of second-order partial derivatives

$$H_f(x) = \nabla_x^2 f(x) \in \mathbb{R}^{d \times d}.$$

Computing and storing this object costs on the order of d^2 per sample. Comparing a full Hessian at every training input would therefore require quadratic memory and repeated heavy computations. This is prohibitive for large input dimensions. However, in many problems the curvature is not equally strong in all directions of the input space. Often, a small number of directions correspond to large eigenvalues and therefore capture most of the dominant curvature, while the remaining directions are comparatively flat. As an example, Figure A.24 in the appendix shows the rapidly decaying eigenspectrum of the mean Hessian of the Heston model (see A.5 for details). This low rank structure suggests that for practical applications it may not be necessary to supervise the entire Hessian as described by the standard formulation of Sobolev training in Definition 1. Instead, it may be sufficient to restrict curvature supervision to only a small set of carefully chosen directions [4, 19].

Definition 3 (Directional 2nd-Order Sobolev Loss) *Let $f : \mathbb{R}^d \to \mathbb{R}$ be a reference function and $\hat{f}_\theta$ a predictor. For a chosen set of input directions $u_1, \dots, u_k \in \mathbb{R}^d$, the weighted second-order directional Sobolev loss at an input x is*

$$\mathcal{L}_2(\theta; x) = \lambda_2 \frac{1}{k} \sum_{i=1}^{k} \left\| H_{\hat{f}_\theta}(x) u_i - H_f(x) u_i \right\|_2^2 . \tag{3.1}$$

The directional second-order Sobolev loss can be calculated efficiently in practice, because it is possible to compute $H_{\hat{f}_\theta}(x)u_i$ and $H_f(x)u_i$ without forming the full Hessians.
The Hessian-Vector Product (HVP) corresponds to the directional second-order derivative of a function $f : \mathbb{R}^d \to \mathbb{R}$ at a point x in a direction $u \in \mathbb{R}^d$:

$$H_f(x)u = \nabla_x^2 f(x)u$$

Pearlmutter [30] showed that HVPs can be computed at the same asymptotic cost as evaluating the gradient of the function by first computing the gradient $\nabla_x f(x)$ and then the directional derivative of that gradient:

$$H_f(x)u = \partial_u \left(\nabla_x f(x) \right).$$

These calculations can be performed efficiently via Algorithmic Differentiation (AD), a topic covered in more detail in Appendix A.2. For functions with many inputs and few outputs, the HVP can be most efficiently realized as a Jacobian-Vector-Product of a Vector-Jacobian-Product, or in AD terms, a *forward- over reverse-mode* computation. Directional higher-order derivatives can be obtained in a similar manner by applying another forward-mode differentiation to the HVP.

The directional Sobolev loss therefore makes higher-order supervision computationally feasible by reducing the memory requirement from $O(d^2)$ to $O(d \cdot k)$ and the compute requirement from forming a dense matrix to evaluating only k matrix-vector products. It also shifts the core question to which directions we should supervise. Existing literature on directional curvature supervision is limited. Random direction probing provides an unbiased estimate of curvature and has been explored in the context of curvature estimation and optimization [24]. It is conceptually comparable to the Hutchinson estimator for approximating the trace of a matrix [15]. Kichler [19] introduced a gradient-based direction selection for second-order Sobolev training. They compute a PCA on the gradients w.r.t. the inputs in a given batch and use the directions of maximal variance as supervision directions for curvature. Our work builds on this line of research by using low-rank approximation techniques from RandNLA for better informed direction selection.

3.2 Batch-wise Randomized SVD

Using techniques from RandNLA (section 2.2), it is possible to approximate the singular spectrum of a large matrix without computing its Singular Value Decomposition (SVD). The RandSVD algorithm by Halko et al. [10] combines a randomized range finder with a small deterministic SVD to compute approximate singular vectors and values of a given matrix $A \in \mathbb{R}^{m \times n}$. The range finder isolates the dominant subspace of A and forms the compressed representation $B \in \mathbb{R}^{s \times n}$, with $s < m$. The final SVD is then carried out on the reduced matrix instead of the full $m \times n$ matrix.

Algorithm 2 RandSVD [10]

```
Require: $A \in \mathbb{R}^{m \times n}$, target rank $k$, oversampling $p$, power steps $q$
1: Set $s \leftarrow k + p$
2: Draw a random test matrix $\Omega \in \mathbb{R}^{n \times s}$ (e.g., i.i.d. Gaussian)
3: Form the sketch $Y \leftarrow (AA^\top)^q A\Omega \in \mathbb{R}^{m \times s}$
4: Compute an orthonormal basis $Q$ for the range of $Y$ via QR-decomposition: $Y = QR$
5: Form the compressed matrix $B \leftarrow Q^\top A \quad \in \mathbb{R}^{s \times n}$
6: Compute the thin SVD of $B$: $B = \tilde{U}\Sigma V^\top$
7: Lift $\tilde{U}$ back to the original space: $U = Q\tilde{U}$
8: Truncate to the top $k$ components: $U_k, \Sigma_k, V_k$
9: return $(U_k, \Sigma_k, V_k)$
```

Theoretical results show that with moderate oversampling, the approximation error is close to optimal with high probability, and the quality improves further with a small number of power iterations (q=1, q=2) [10].

We adapt this idea to compute curvature-aware supervision directions for second-order Sobolev training. The matrix we are interested in is the Hessian of the reference model. For a given batch of inputs $\mathcal{B} = \{x_b\}_{b=1}^m$, we aim to approximate the dominant curvature directions of the batch-averaged Hessian

$$\bar{H}_f = \frac{1}{m} \sum_{b=1}^{m} \nabla_x^2 f\left(x_b\right)$$

Intuitively, $\bar{H}_f$ captures the typical curvature structure of the reference model over the current batch. By taking HVPs computed via AD, we construct a matrix-free version of the RandSVD algorithm applied to $\bar{H}_f$:

1. In Step 3, where the standard algorithm would multiply $\bar{H}_f$ by a test matrix Ω to form the sketch $Y = \bar{H}_f \Omega$, we instead build $Y \in \mathbb{R}^{d \times s}$ by evaluating HVPs into random probe directions. That is $Y = \frac{1}{m} \sum_{b=1}^{m} \left[H_f\left(x_b\right)\omega_1, \ldots, H_f\left(x_b\right)\omega_s\right]$, where the ω_i are the columns of Ω.

2. Likewise, in Step 5, where the standard algorithm would project $\bar{H}_f$ into the computed subspace by forming $B = Q^\top \bar{H}_f$, we construct that product row by row. For each column q_i of Q we compute $\bar{H}_f q_i$ by averaging HVPs $H_f(x_b)q_i$ over the batch and use $\left(\bar{H}_f q_i\right)^\top$ as the i-th row of B. This is valid in our setting because the Hessians are symmetric and therefore $Q^\top H_f(x_b) = (H_f(x_b)^\top Q)^\top = (H_f(x_b)Q)^\top$. Stacking the transposed HVPs row-wise yields the desired matrix $B \in \mathbb{R}^{s \times d}$.

At the end we truncate to the top $k < s$ directions. Optionally, we keep only as many singular directions as are needed to explain a prescribed fraction of the curvature variance. The extracted curvature directions can then be used in the directional second-order Sobolev loss.

Algorithm 3 Batch-wise RandSVD for dominant curvature directions via HVPs

```
Require: Function $f : \mathbb{R}^d \to \mathbb{R}$ (twice differentiable), batch $\mathcal{B} = \{x_b\}_{b=1}^m$, target rank $k$,
    oversampling $p$, power steps $q$
 1: Set $s \leftarrow k + p$
 2: Draw a random test matrix $\Omega \in \mathbb{R}^{d \times s}$ (e.g., i.i.d. Gaussian)
 3: Sketch with HVPs:
        $$Y \in \mathbb{R}^{d \times s} \;\leftarrow\; \frac{1}{m} \sum_{b=1}^{m} H_f(x_b)\,\Omega \quad \text{(columns of } \Omega \text{ supplied to HVPs)}$$
 4: (optional power iteration) repeat $q$ times: $Y \leftarrow \frac{1}{m} \sum_{b=1}^{m} H_f(x_b) Y$ via HVPs
 5: Orthonormal basis $Q \in \mathbb{R}^{d \times s}$ for the range of $Y$ via QR-decomposition: $Y = QR$
 6: Compressed operator with HVPs:
        $$B^\top \in \mathbb{R}^{d \times s} \;\leftarrow\; \frac{1}{m} \sum_{b=1}^{m} H_f(x_b)\,Q \quad \text{(columns of } Q \text{ supplied to HVPs)}$$
 7: Compute the thin SVD of $B$: $B = \widetilde{U}\,\Sigma\,V^\top$ with $\widetilde{U} \in \mathbb{R}^{s \times s}$
 8: Lift $\widetilde{U}$ back to the original space: $U = Q\widetilde{U}$, with $U \in \mathbb{R}^{d \times s}$
 9: Truncate to the top-$k$ components: $U_k \in \mathbb{R}^{d \times k}$
10: return $U_k$
```

The recovered supervision directions directly reflect the dominant second-order structure itself, rather than inferring it from gradient variability as done in the PCA-of-gradients approach in [19]. A tradeoff is that when constructing $\bar{H}_f$, we obtain directions that describe the typical curvature structure of that batch, not necessarily the curvature at each individual point. If curvature changes sharply across the batch, then this averaging can smooth out local variability to the extent of the batch size. Computing a separate set of supervision directions for each individual input x_b was considered but did not turn out to be efficient. For reference functions where the Hessians do not vary a lot over the inputs, it is possible to use the information extracted from one batch to inform the approximations of all following batches. In the Appendix A.3 we include a streaming version, where instead of re-approximating the curvature subspace from scratch for every batch, we approximate the population operator $\mathbb{E}_x[H_f(x)]$ over all available training data and refine a single basis $Q \in \mathbb{R}^{d\times r}$ over the course of training.

3.3 Extension to 3rd Derivatives

We extend the second-order batch-wise RandSVD method to the third derivative tensor. The objective remains the same and is to find a small set of directions along which third-order effects are strong and therefore most useful for supervision.

Let $f : \mathbb{R}^d \to \mathbb{R}$ be three times continuously differentiable. Its third derivative at a point x is the symmetric 3-tensor

$$T_f(x) = \nabla_x^3 f(x) \in \mathbb{R}^{d\times d\times d}$$

We avoid materializing $T_f(x)$ but have access to third-order tensor-vector-vector products (TVVPs) via algorithmic differentiation. In contrast to the HVP, which maps a point x and a direction v to the vector $H_f(x)v \in \mathbb{R}^d$, the third-order version takes two input directions $v, w \in \mathbb{R}^d$ and returns the slice

$$T_f(x; \cdot, v, w) = \partial_w (H_f(x)v) \in \mathbb{R}^d,$$

which can be interpreted as the directional derivative of the Hessian action $H_f(x)v$ along w.

Definition 4 (Directional 3rd-Order Sobolev Loss) *Let $f : \mathbb{R}^d \to \mathbb{R}$ be a reference function and $\hat{f}_\theta$ a predictor. For a chosen set of input directions $u_1, \dots, u_k \in \mathbb{R}^d$, the weighted third-order directional Sobolev loss at an input x is*

$$\mathcal{L}_3(\theta; x) = \lambda_3 \frac{1}{k^2} \sum_{i=1}^{k} \sum_{j=1}^{k} \left\| T_{\hat{f}_\theta}(x; \cdot, u_i, u_j) - T_f(x; \cdot, u_i, u_j) \right\|_2^2, \tag{3.2}$$

where we contract both modes along the same set of directions.

To find suitable supervision directions given batch of inputs $\mathcal{B} = \{x_b\}_{b=1}^m$, we are interested in the batch-averaged operator

$$\bar{T}_f = \frac{1}{m} \sum_{b=1}^{m} T_f(x_b).$$

We represent $T_f(x)$ by its mode-one unfolding, which encodes all TVVPs as matrix-vector multiplications:

$$T_f^{(1)}(x) \in \mathbb{R}^{d\times d^2}, \quad \text{with } \left[T_f^{(1)}(x)\right]_{i,(j-1)d+k} = T_f(x)_{i,j,k}$$

Then with z defined as the Kronecker product of a given pair $v, w \in \mathbb{R}^d$:

$$T_f^{(1)}(x) z = T_f(x; \cdot, v, w), \quad z = \mathrm{vec}(v \otimes w) \in \mathbb{R}^{d^2},$$

meaning, the third-order TVVP that returns $T_f(x;\cdot,v,w)$ corresponds exactly to applying the matrix $T_f^{(1)}(x)$ to rank one vectors of the form $\operatorname{vec}(v \otimes w)$. The mode-one unfolding is only a reindexing of values and no information is lost when passing from $T_f(x)$ to $T_f^{(1)}(x)$. By a generalization of Clairaut's and Schwarz's theorem the order of differentiation does not matter and unfoldings along different modes are equivalent up to permutation [23]. Direct construction of the individual $T_f^{(1)}(x_b)$ is of course infeasible. Instead, we approximate the dominant left singular subspace of

$$\bar{T}_f^{(1)} = \frac{1}{m}\sum_{b=1}^{m} T_f^{(1)}(x_b)$$

using a randomized sketch that operates only through TVVPs computable efficiently via AD. We choose a collection of s sketch directions $\{u_i\}_{i=1}^{s} \subset \mathbb{R}^d$ and evaluate

$$y_{b,i,j} = T_f\left(x_b;\cdot,u_i,u_j\right) \in \mathbb{R}^d$$

for each training input x_b and pair (i,j) via AD. The sketch directions may be chosen entirely at random, or (as in our evaluations) consist of the dominant second-order directions, enriched with a few random vectors. This way exploration of third-order structure is guided by known second-order information. Using the same set of sketch directions for both contracted slots of $T_f(x_b)$ respects its symmetry and still allows to probe both diagonal contractions and mixed interactions. Stacking all index pairs and averaging across the batch produces the sketch

$$Y \in \mathbb{R}^{d\times s^2}, \quad Y = \frac{1}{m}\sum_{b=1}^{m} T_f^{(1)}(x_b)\Omega,$$

where the columns of $\Omega \in \mathbb{R}^{d^2\times s^2}$ are the Kronecker products $\operatorname{vec}\left(u_i \otimes u_j\right)$. The use of all pairs (u_i,u_j) produces a richer sketch than using only diagonal pairs (u_i,u_i), as mixed pairs capture interactions between sketching directions. The increased number of probes s^2 remains acceptable when s is much smaller than d.

As in the second-order case, we then compute an orthonormal basis $Q = \operatorname{orth}(Y)$. The columns of $Q \in \mathbb{R}^{d\times s^2}$ provide candidate supervision directions, but we need to select the top $k < s^2$ directions. In the second-order setting we formed $B = Q^\top \bar{H}_f$ and obtained an ordering through its singular values. In that case it is possible to form B by evaluating HVPs, because the individual $H_f(x_b)$ are symmetric and $Q^\top H_f(x_b) = (H_f(x_b)Q)^\top$. In the third-order case, however, the $T_f^{(1)}(x_b)$ are not symmetric and we do not have access to their transposed versions via AD. Instead, we rank each candidate direction q_i by the average squared magnitude of its contracted third derivative,

$$M\left(q_i\right) = \frac{1}{m}\sum_{b=1}^{m} \left\|T_f\left(x_b;\cdot,q_i,q_i\right)\right\|_2^2.$$

This quantity serves as a proxy for the strength of third-order effects along q_i and requires only TVVPs. We then select the k directions with the largest values of $M\left(q_i\right)$ and assemble them as

$$U = [u_1,\ldots,u_k] \in \mathbb{R}^{d\times k}.$$

The resulting supervision directions arise from a third-order analogue of the second-order randomized range finder. They approximate the leading left singular subspace of the mode-one unfolding of the third derivative tensor. The methodology is extendable to mode-one unfoldings of fourth- and higher-order derivatives. A concise overview of the algorithm is given in Appendix A.3.3.

3.4 Full Training Pipeline

We consider a scalar reference function $f : \mathbb{R}^d \to \mathbb{R}$ and a surrogate neural network $\hat{f}_\theta$. We supervise values, full first-order derivatives, and directional second- and third-order derivatives. Each training step draws a batch of samples $\mathcal{B} = \{x_b\}_{b=1}^m$ and computes targets for values and first-order derivatives. For second- and third-order supervision, small sets of informative supervision directions are selected using one of the proposed procedures.

Definition 5 (Combined Directional Sobolev Loss) *For a given batch $\mathcal{B} = \{x_b\}_{b=1}^m$, let $\{u_i\}_{i=1}^{k_{2nd}}, \{v_j\}_{j=1}^{k_{3rd}} \subset \mathbb{R}^d$ be the selected second- and third-order supervision directions. The combined directional Sobolev loss is*

$$\frac{1}{m}\sum_{b=1}^{m} \mathcal{L}_{total}(\theta; x_b),$$

$$\mathcal{L}_{total}(\theta; x_b) = \lambda_0 \mathcal{L}_0(\theta; x_b) + \lambda_1 \mathcal{L}_1(\theta; x_b) + \lambda_2 \mathcal{L}_2(\theta; x_b) + \lambda_3 \mathcal{L}_3(\theta; x_b) + c_\lambda,$$

where λ_i are learned balancing factors and c_λ is a regularization term preventing $\lambda_i \to 0$. The per-order contributions are

$$\mathcal{L}_0(\theta; x_b) = \left\| \hat{f}_\theta(x_b) - f(x_b) \right\|_2^2,$$

$$\mathcal{L}_1(\theta; x_b) = \left\| \nabla_x \hat{f}_\theta(x_b) - \nabla_x f(x_b) \right\|_2^2,$$

$$\mathcal{L}_2(\theta; x_b) = \frac{1}{k_{2nd}} \sum_{i=1}^{k_{2nd}} \left\| H_{\hat{f}_\theta}(x_b)\, u_i - H_f(x_b)\, u_i \right\|_2^2,$$

$$\mathcal{L}_3(\theta; x_b) = \frac{1}{k_{3rd}^2} \sum_{i=1}^{k_{3rd}} \sum_{j=1}^{k_{3rd}} \left\| T_{\hat{f}_\theta}(x_b; \cdot, v_i, v_j) - T_f(x_b; \cdot, v_i, v_j) \right\|_2^2,$$

if supervision up to that order is active, otherwise zero.

Balancing the loss terms:
Because the loss terms of different orders can differ significantly in scale and learnability, the weights λ_0 to λ_3 must be chosen carefully. We propose a balancing consisting of base-weights as in [14] and [19], and combine it with learnable reweighting to account for differences in learnability and noise. The proposed mechanism is described in detail in Appendix A.4.

Normalization:
For stable optimization, input samples $x \in \mathbb{R}^d$ and targets $y = f(x) \in \mathbb{R}$ are mapped to normalized variables

$$\tilde{x}_j = \frac{x_j - \mu_{x,j}}{\sigma_{x,j}}, \quad \tilde{y} = \frac{y - \mu_y}{\sigma_y},$$

using per-dimension means $\mu_{x,j}$ and standard deviations $\sigma_{x,j}$, and output scale σ_y. By the chain rule, first-, second-, and third-order derivatives transform as

$$\frac{\partial \tilde{y}}{\partial \tilde{x}_j} = \frac{\sigma_{x,j}}{\sigma_y}\frac{\partial y}{\partial x_j}, \quad \frac{\partial^2 \tilde{y}}{\partial \tilde{x}_j \partial \tilde{x}_k} = \frac{\sigma_{x,j}\sigma_{x,k}}{\sigma_y}\frac{\partial^2 y}{\partial x_j \partial x_k},$$

$$\frac{\partial^3 \tilde{y}}{\partial \tilde{x}_j \partial \tilde{x}_k \partial \tilde{x}_\ell} = \frac{\sigma_{x,j}\sigma_{x,k}\sigma_{x,\ell}}{\sigma_y}\frac{\partial^3 y}{\partial x_j \partial x_k \partial x_\ell}.$$

Working in normalized coordinates keeps all supervised quantities in similar numeric ranges and makes the loss terms more comparable.

4 Results

In this section we evaluate the effectiveness of the proposed methods for NN surrogate training. Our evaluation proceeds in two stages. We begin with an analytic benchmark setting in which noise-free training labels are available via AD of analytic reference functions. This setting isolates the behavior of the proposed methods without interference from noisy training labels. Afterwards, to highlight how noisy training labels can impact the optimization procedure, we evaluate under Monte Carlo approximations of the Bachelier model A.5, following the setups of Kichler [19] and Huge [14]. In this setting, derivative labels are obtained via pathwise differentiation. Evaluations on the test set are performed against analytic labels to ensure clean and consistent measurements.

Training Setup:
All experiments use the same surrogate architecture and fixed hyperparameters to ensure comparability across methods. The surrogate is a MLP (see A.1) with four hidden layers, width 20, and silu activation functions. We use the Adam optimizer with constant learning rate 10^{-3}. The training regime is designed to reflect limited data availability. We use an online sampling setup with 64 batches of size 128 per epoch. The batch-keys are repeated across epochs, effectively simulating a finite dataset while maintaining the convenience of online sampling.

4.1 Analytic Regime

First, we evaluate all methods on an analytic implementation of the Bachelier basket call option. For consistency with prior studies [19, 14] we use a basket of size 7, with model-parameters as specified at the end of section A.5. Because the reference function has only one dominant curvature direction, we use $k = 1$ supervision directions for all second- and third-order methods. For the batch-wise RandSVD method, we use oversampling parameter $p = 1$ and no power-iteration. Adding power-iteration did not impact the results. For the streaming variant we use a global sketch size of $r = 2$.

Figure 4.1 shows the test-set Root Mean Square Errors (RMSEs) of all methods over 100 epochs of training. The error curves are averaged over 10 random seeds and the standard deviations are shown by shaded areas around the curves. The upper-left panel (a) shows errors in the value- (price-) predictions of the surrogate, (b) shows errors in its learned first derivatives (Deltas), (c) shows errors in the second derivatives (Gammas), and (d) shows errors in the third derivatives (Speeds). Including first derivatives in the supervision already leads to significant improvements in accuracy across all orders. Including second derivatives leads to marginally higher accuracies in the learned first-, second- and third derivatives. Both proposed second-order methods converge faster than using random directions for supervision and are equally accurate to the PCA of gradients approach. The error curves of these three methods lie on top of each other and only the top (brown) curve of the streaming method is visible. It must be noted, however, that the improvement in sample-efficiency compared to supervising random directions, does not necessarily translate to improved wall-clock-time of the training procedure, as there is an overhead involved in finding the more informative supervision directions. For the batch-wise RandSVD method, the overhead is an additional $(k+p)(q+2)$ HVPs (per element in the batch) plus some matrix factorizations. Whether this

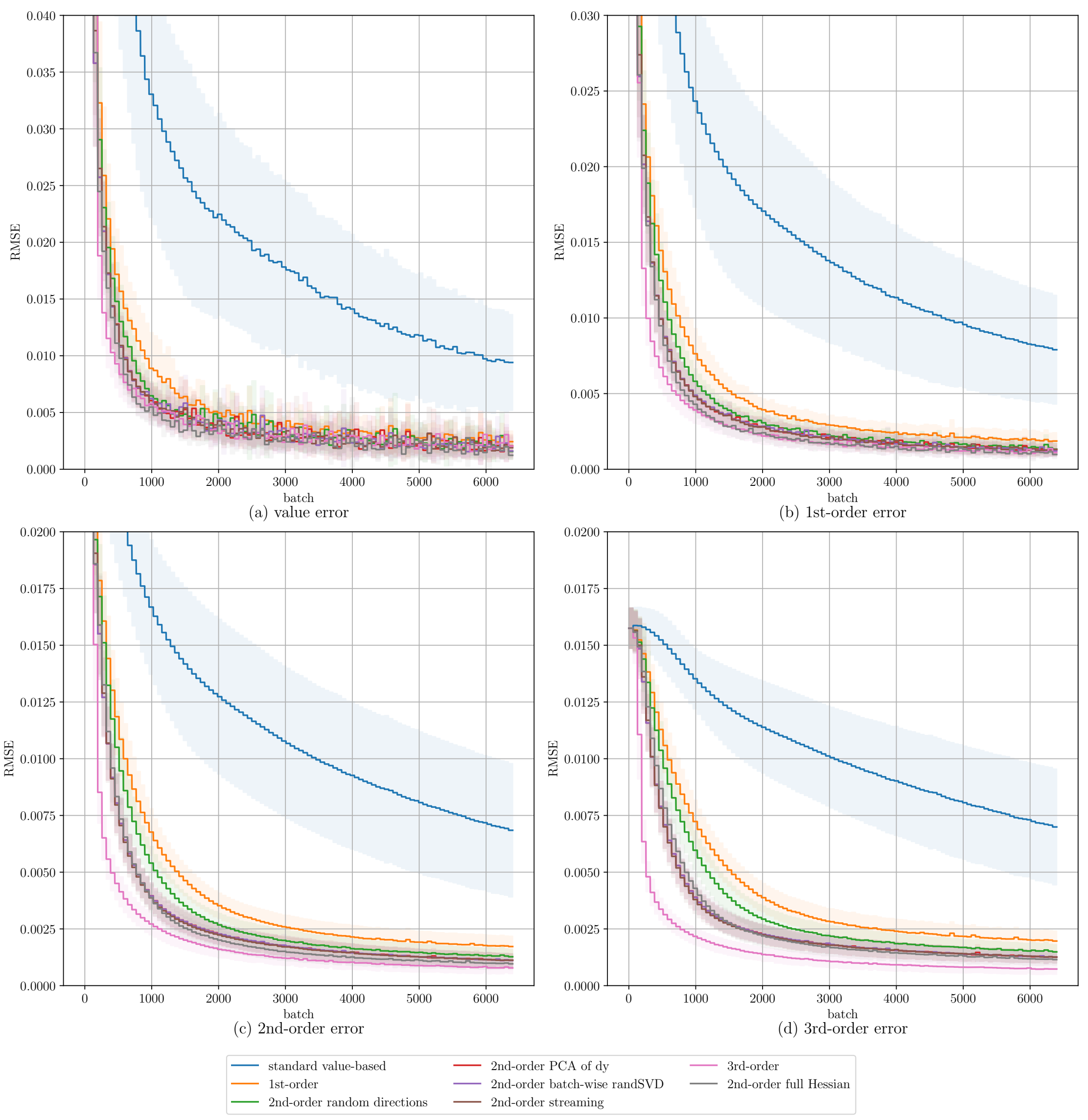


Figure 4.1: Test-set RMSEs over training batches for the analytic Bachelier basket call (basket size 7). Curves show mean RMSE across 10 random seeds with shaded ±1 standard deviations. Panels show errors in the predicted values and the learned derivatives.

overhead is worthwhile depends on the chosen parameters k, p, q, as well as the dimensionality and computational complexity of the reference function. In our observations, the overhead is typically not worthwhile for cheap, analytic reference functions but becomes beneficial for computationally expensive, high dimensional ones, as will be shown in the following section 4.2. For this experiment, the mean per-batch execution times on our system are given by: 2.1 ms for supervising random directions, 2.4 ms for the batch-wise RandSVD method, 2.5 ms for the streaming method, 2.3 ms for the PCA of gradients approach, 2.4 ms for full Hessian supervision and 3.0 ms for third-order directional supervision. Including a third-order term in supervision leads to improved third-order accuracy. It also becomes apparent that including higher-order Sobolev loss terms not only increases the final accuracies of the trained surrogate but also improves sample-efficiency, as fewer samples are required to reach a given accuracy. As expected, the adaptive loss balancing mechanism had no large effect and the learned weights remained at their constant base values. An evaluation of the trained surrogates is included in the appendix (Figure A.4 to Figure A.7).

Other analytic reference functions:

The observations on the Bachelier model generalize across many analytic reference functions. With few exceptions, higher-order methods consistently outperform lower-order ones in both accuracy and sample-efficiency. Detailed results for the Heston pricing model and a range of analytic benchmark functions are provided in the Appendix A.5. Additionally, we include a NN as a reference function to illustrate the model distillation setting which Sobolev training was considered for by Czarnecki et al. [6].

4.2 Monte Carlo Approximated Training Labels

Surrogates are often needed for reference functions that do not admit exact analytic solutions and must instead be approximated through simulation. Therefore, we next evaluate on a Monte Carlo (MC) approximation of the Bachelier model. The basket size and model parameters remain unchanged from the analytic experiment. Training labels are obtained via pathwise differentiation of single simulation paths following Kichler [19] and Huge [14]. As in their work, we begin with each training-label corresponding to only one simulation path. The averaging over many MC paths happens over the period of training. To be able to train with pathwise derivatives, it is required to smooth the kink in the non-differentiable payoff function. We do so via a three-times continuously differentiable polynomial bridge with $\epsilon = 0.1$ as described in A.5.

Figure 4.2 shows the test-set RMSEs of the surrogates over 100 epochs of training. Including first- and second-order loss terms leads to improved accuracies. Both proposed second-order methods outperform random directions and perform on par with the PCA of gradients approach. These results are consistent with the findings of Kichler [19].
Including third derivatives in this Monte Carlo setting shows a new phenomenon. The third-order loss term harms the convergence of all other terms. Figure A.23 in the appendix shows that higher-order targets become increasingly noisy. The labels for third derivatives (Speeds) contain very little usable signal, and attempting to fit this noise degrades performance. The uncertainty-aware balancing mechanism detects this and down-weights the third-order term as shown in Figure 4.3. Without this mechanism, continued training with static base-weights results in significantly worse final performances (full error curves given in the appendix Figure A.1):

value error:	$+0.0265$ ($+250\%$)
1st-order error:	$+0.0165$ ($+242\%$)
2nd-order error:	$+0.0140$ ($+232\%$)
3rd-order error:	$+0.0156$ ($+231\%$)

Down-weighting the third-order loss term improves accuracy even in the learned third derivatives. The surrogate is able to infer the correct structure through the value-, first- and second-order supervision.

To improve the signal-to-noise ratio in higher-order labels, one can average over many more paths. Figure A.22 in the appendix shows how the labels become less noisy when averaging over 1000 paths for each input sample. When training labels are averaged over many MC estimates, the loss curves become similar to the analytic setting (Figure A.2) and the learned weights remain close to the base-weights (Figure A.3).

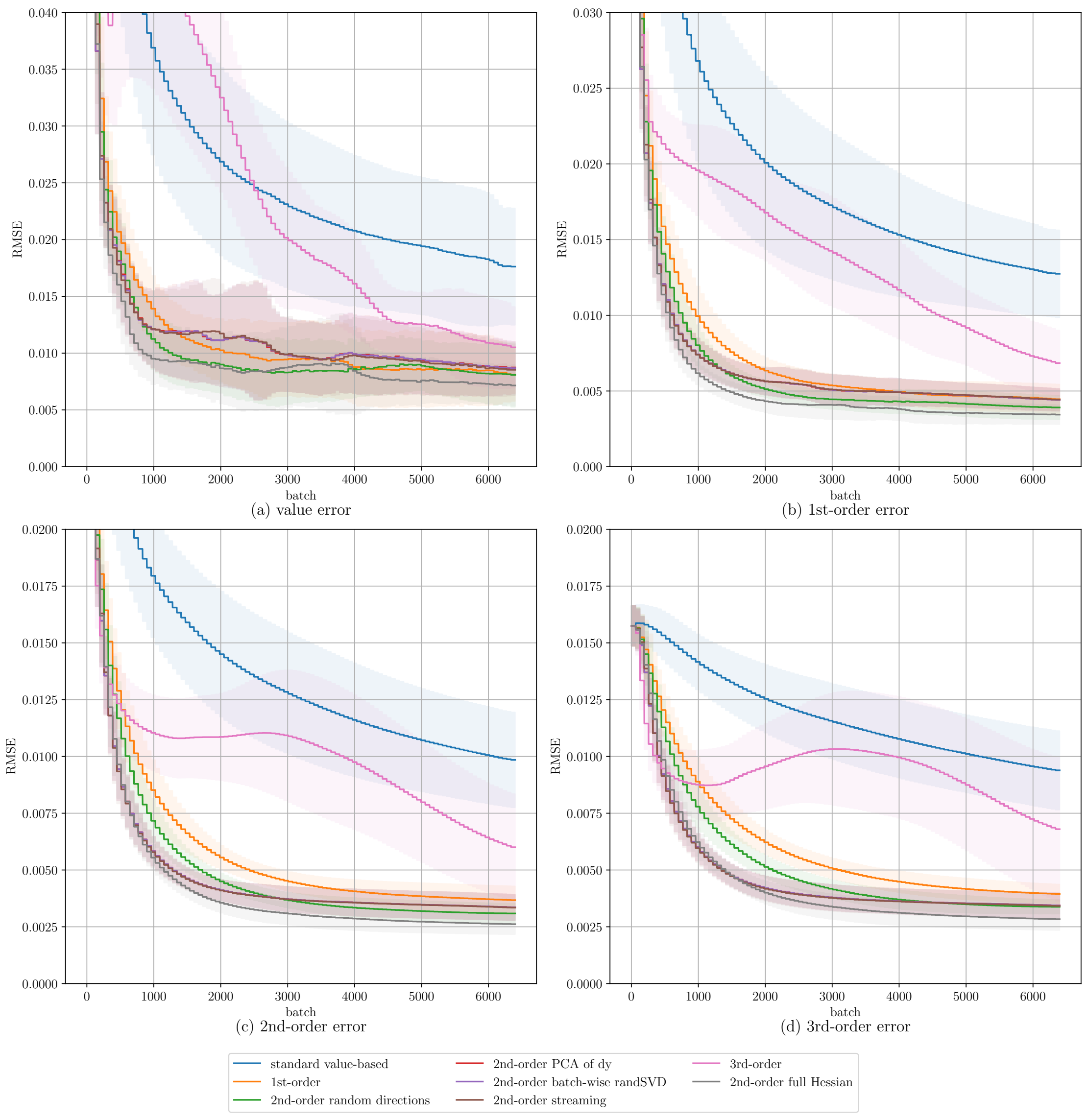


Figure 4.2: Test-set RMSEs over training batches for the Monte Carlo simulated Bachelier basket call (basket size 7) with pathwise derivative labels. Curves show mean RMSE across 10 random seeds with shaded ± 1 standard deviations. Panels show errors in the predicted values and the learned derivatives.

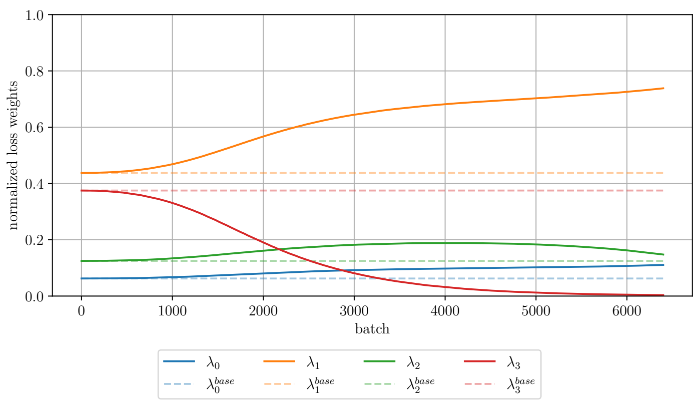


Figure 4.3: Normalized loss weights over training for the Monte Carlo simulated Bachelier basket call (basket size 7) with pathwise derivative labels.

However, increasing the number of MC simulations also increases the computational cost per batch. For computationally heavy reference functions, the improved scaling behaviors of the proposed methods become increasingly relevant. The computational overhead of finding a small set of informative supervision directions becomes less than the cost of supervising all directions. Figure 4.4 shows the error curves for a 50-dimensional Bachelier basket option with averaging 1000 MC estimates per training label, plotted against elapsed wall-clock-time. Each curve is scaled by the average batch execution time for its respective method. In this setting, the two proposed second-order methods provide the most efficient trade-off across accuracy and computation, outperforming also the PCA of gradients approach. Third-order Sobolev training, while encoding more information per sample, is slower per batch, and the cost-benefit trade-off is not favorable for this specific reference function.

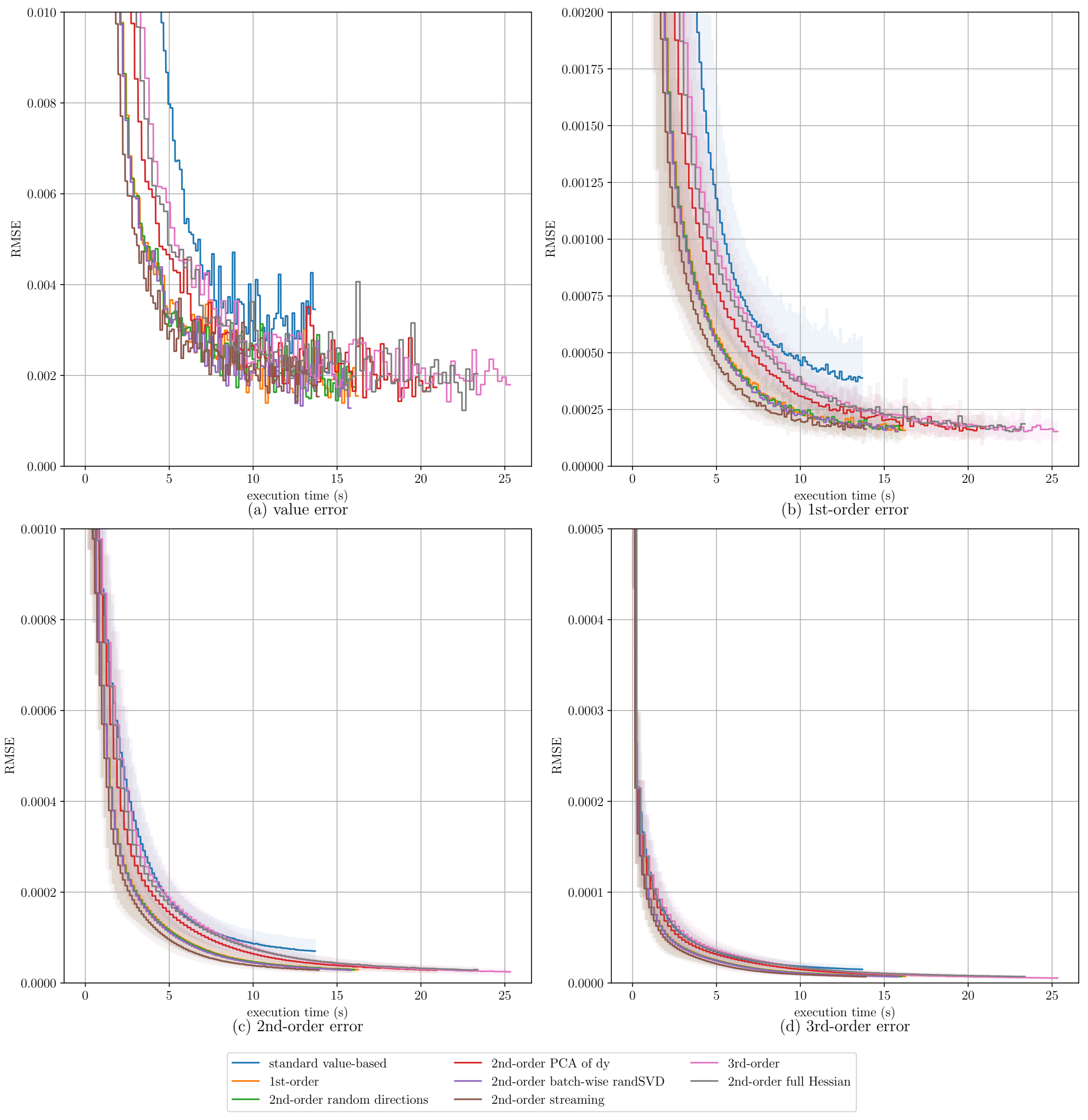


Figure 4.4: Test-set RMSEs over training time for the Monte Carlo simulated Bachelier basket call (basket size 50) with derivative labels averaging 1000 paths each. Curves show mean RMSE across 10 random seeds with shaded ±1 standard deviations. Panels show errors in the predicted values and the learned derivatives.

5 Conclusion

This thesis investigated Sobolev training for NN surrogates. While first-order Sobolev training is established, full supervision of higher-order derivatives becomes computationally prohibitive. The central idea of this work is to replace full higher-order supervision with directional supervision guided by randomized low-rank approximation techniques. Using only directional derivative evaluations computed efficiently via AD, we identify low-dimensional subspaces that capture the dominant higher-order behavior of the reference function. The supervision of higher-order derivatives can then be focused on a small set of informative directions, providing a scalable alternative that still captures the essential higher-order behavior. Across a variety of reference functions, directional second- and third-order supervision improves accuracy and sample-efficiency over value-only and first-order supervision. Our methods outperform random-direction supervision, while remaining more efficient than supervising full Hessians for high-dimensional and computationally expensive reference functions. Our experiments further show that loss balancing is essential for higher-order Sobolev training due to differences in learnability and noise across loss terms of different orders. Especially under noisy labels, uncertainty-aware adaptive reweighting stabilizes training by down-weighting unlearnable, noisy loss terms.

Discussion:
The proposed methods are most effective when the reference function exhibits low-rank higher-order behavior, so that a small subspace captures most of the higher-order signal. In such cases, RandNLA-based sketches offer a principled way to find informative supervision directions. Limitations arise when a) the higher-order structure is close to full-rank, as no direction selection can be better than random, or b) the dominant higher-order behavior changes strongly across the input space, so that a single global or batch-averaged subspace fails to capture important local effects. Against our expectations, adding power-iteration steps to the batch-wise second-order RandSVD direction selection did not improve surrogate accuracy on any of the reference functions we evaluated.
From a computational perspective, the methods replace the cost of forming full higher-order derivatives with a controlled number of directional derivative evaluations plus lightweight factorizations. Whether this overhead is worthwhile depends on the cost of the reference function and its input dimension. For sufficiently expensive and high-dimensional problems, the improved scaling leads to tangible efficiency gains.

Future Work:
A first direction is rank adaptivity. The number of supervised directions could be chosen dynamically using explained-variance thresholds or through rank-revealing sketching techniques [16] to detect the effective dimensionality of the higher-order structure online. A second direction is to better address locality and nonstationarity by learning region-dependent subspaces. This could be achieved through clustering or mixture-of-experts architectures, allowing different sketches to specialize in different regions of the input space. A third direction is adaptive sampling, using the derivative information accumulated so far to guide where new evaluations should be taken. Derivative-aware sampling strategies could focus supervision on regions where it is most informative. Finally, there is room for theoretical analyses that establish connections between the spectral decay of derivative operators, the sketch quality, and the resulting gains in sample efficiency during directional Sobolev training.

A Supporting Material

A.1 Neural Networks

Function Approximation and Supervised Learning:

Machine Learning can be understood as the problem of approximating an unknown target function $f : \mathcal{X} \to \mathcal{Y}$ from a finite set of observations $\{(x_i, y_i)\}_{i=1}^{m}$. Given these input-output pairs the task is to learn a parameterized model $\hat{f}_\theta : \mathcal{X} \to \mathcal{Y}$ that predicts outputs close to the true mapping f. These models are often implemented by neural networks, which are then trained to minimize the empirical risk over the available data

$$\min_\theta \frac{1}{m} \sum_{i=1}^{m} \mathcal{L}\left(\hat{f}_\theta\left(x_i\right), y_i\right).$$

The empirical risk is designed to approximate the expected risk

$$\mathbb{E}_{(x,y)\sim\mathcal{D}}\left[\mathcal{L}\left(\hat{f}_\theta(x), y\right)\right],$$

over the generating distribution $\mathcal{D}$, which in practice may differ between training and deployment. The loss function $\mathcal{L}$ quantifies the discrepancies between predictions and targets. Common choices are the mean-squared error for regression tasks and the cross-entropy loss for classification tasks. In this work we limit ourselves to regression tasks with target functions $f : \mathbb{R}^d \to \mathbb{R}$.

Neural Network Structure and Expressivity:

Neural Networks (NNs) realize nonlinear mappings through the successive application of linear transformations and nonlinear activation functions. A feed-forward multilayer perceptron (MLP) with L layers takes the form

$$\hat{f}_\theta(x) = W_L\sigma\left(W_{L-1}\sigma\left(\ldots\sigma\left(W_1 x + b_1\right)\ldots\right) + b_{L-1}\right) + b_L$$

where $\theta = \{W_\ell, b_\ell\}_{\ell=1}^{L}$ denotes the learnable parameters with $W_\ell \in \mathbb{R}^{n_{l+1}\times n_l}$ and $b_\ell \in \mathbb{R}^{n_{l+1}}$. The nonlinear activation functions $\sigma(\cdot)$ enable the network to approximate nonlinear functions. For Sobolev training it is essential to use smooth activations, such as the silu function, as they enable stable computation of derivatives. Piecewise-linear activations like ReLU introduce non-differentiable kinks, which can lead to discontinuities and Dirac-like spikes. The silu activation function used in our experiments is given by:

$$\text{silu}(x) = \frac{x}{1 + e^{-x}}$$

The Universal Approximation Theorem [13] states that even a single hidden layer MLP of sufficient width can approximate any continuous function on a compact domain arbitrarily well. The difficulty lies in finding the correct parameterization θ through optimization. In practice, deeper architectures are used as they need fewer parameters by reusing intermediate nonlinearities [21].

Optimization and Training:

Minimizing the empirical risk over the parameters θ is achieved through iterative, gradient-based optimization. Gradients of the empirical risk with respect to the parameters are estimated on small subsets of the training data (batches), and an update step is applied. These updates to the parameters iteratively tune the model to minimize the empirical risk over the given data. The update rule for stochastic gradient descent (SGD) is

$$\theta_{t+1} = \theta_t - \eta \nabla_\theta \widehat{\mathcal{L}}_{\mathcal{B}} (\theta_t)$$

where η is the learning rate and $\widehat{\mathcal{L}}_{\mathcal{B}}$ is the loss computed over a batch $\mathcal{B}$. Small random subsets of the data are used for each update iteration because computing gradients on the entire dataset is often limited by available memory. Furthermore, empirical evidence shows that the noisy gradient estimates of stochastic batches help the optimizer escape sharp minima in the non-convex loss landscape [18]. Additionally, various refinements such as adaptive learning-rate schedules or momentum methods (e.g. Adam) can further improve convergence and stability. Although the loss landscape is highly non-convex, empirical evidence shows that stochastic gradient descent rarely gets stuck in suboptimal, local minima. Isolated minima are very rare in high-dimensional parameter spaces, as the optimizer would need to get stuck in all parameter directions simultaneously. Instead, optimization tends to find broad, low-curvature regions that correspond to good generalization performance [7].

Regularization and Generalization:

A critical challenge in training NNs is to achieve good performance on unseen inputs. Overparametrized networks can easily learn to fit the training data but may fail to generalize beyond it, a behavior known as overfitting. This motivates the use of regularization techniques which constrain model complexity and promote smoothness.
Regularization can be done implicitly through mechanisms such as dropout or data augmentation. These techniques reduce overfitting by introducing noise and variability during training. Regularization can also be introduced explicitly, through penalization of large parameter values in the loss. Similarly, large derivatives w.r.t. the inputs can be penalized to enforce smoothness in the learned function. This interpretation directly connects regularization to Sobolev training, where smoothness is learned explicitly by matching the derivatives of the target function.

A.2 Algorithmic Differentiation

Many of the methods developed in this thesis rely on derivative information. Accurate and efficient derivative computations are therefore essential. Traditional approaches such as finite differences (FD) and symbolic differentiation, however, are not practical for our applications. FD approximates derivatives through repeated function evaluations, which causes computational cost that scales linearly with the input dimension. Symbolic differentiation expands analytic expressions, which leads to exponential growth in expression size. Algorithmic Differentiation (AD) provides an efficient alternative. It computes exact derivatives up to machine precision by systematically applying the chain rule through the computational graph of a program. This makes it possible to compute gradients and higher order derivatives efficiently without sacrificing accuracy [27, 9].

Principles of Algorithmic Differentiation:

AD decomposes a program (that represents a function $f : \mathbb{R}^m \to \mathbb{R}^n, f(x) = y$) into a computational graph of elementary operations with known derivatives. The graph consists of nodes representing intermediate variables and edges encoding dependencies. Sensitivities are then propagated through this graph by applying the chain rule. This can be done either front-to-back or back-to-front, which is why two main modes of AD exist.

In *forward mode* AD, for a given input direction $v \in \mathbb{R}^m$, each intermediate variable u_j in the computation of f is associated with its rate of change with respect to v:

$$\dot{u}_j = \frac{du_j}{dx} v.$$

These derivatives are calculated by executing the program once in the forward direction and propagating the updates

$$\dot{u}_j = \sum_{i \in p(j)} \frac{\partial g_j}{\partial u_i} \dot{u}_i$$

where g_j is an elementary operation, with known derivative, acting on parent variable u_i to impact u_j. $p(j)$ denotes the index-set of intermediate variables u_i that u_j directly depends on. The result of the run is the output value y of the program and the directional derivative $\dot{y}_v = J_f(x)v$. Its cost is proportional to one additional evaluation of the program per input direction, which is why forward mode AD is most efficient when the input dimension m to the program is smaller than its output dimension n.

In *reverse mode* AD, each intermediate variable u_i is instead associated with the sensitivity of the output y with respect to that intermediate variable:

$$\bar{u}_i = \frac{\partial y}{\partial u_i}.$$

First, the program is executed in the forward direction to store the values u_i of all intermediate variables. Afterwards, for a given output direction $v \in \mathbb{R}^n$, a backwards sweep propagates sensitivities in reverse order:

$$\bar{u}_i = \sum_{j \in c(i)} \bar{u}_j \frac{\partial g_j}{\partial u_i},$$

where again g_j is an elementary operation, with known derivative, acting on parent variable u_i to impact u_j. $c(i)$ denotes the index-set of intermediate variables u_j that u_i directly impacts. The result of this reverse pass is $\bar{y}_v = v^T J_f(x)$, where v is a vector of output sensitivities (often $v = 1$ for scalar outputs). At the expense of the memory cost for storing intermediate values, its computational cost is relative to the number of elementary operations times the output dimension n of the computed function. It is therefore most efficient for functions with $m > n$. For this reason it is often used in applications involving NNs. A prominent example is the backpropagation algorithm.

Implementation in JAX:

All computations in this work are implemented using JAX [2], a Python library for efficient array computation and algorithmic differentiation. JAX provides direct access to the forward- and reverse-mode operations introduced before. Together with just-in-time (JIT) compilation, this enables efficient evaluation of gradients and higher-order derivatives on CPU, GPU and TPU backends. JAX operates within a purely functional subset of Python. To support this, JAX provides differentiable control-flow constructs such as `jax.scan`, `jax.cond` and `jax.whileloop`. Additionally, `jax.vmap` enables vectorization over batches of inputs. Forward- and reverse-mode differentiation are exposed as `jax.jvp` and `jax.vjp`. As an example, they can be composed to compute HVPs as follows:

```
def hvp(f, primals, tangents):
    return jax.jvp(lambda x: jax.grad(f)(x), primals, tangents)[1]
```

Where `jax.grad` is a wrapper for `jax.vjp` with output direction 1, convenient for functions with scalar output.

To be compatible with JAX, NNs are implemented using Equinox [20]. Equinox is a Python library that extends JAX with a PyTorch-like module system. It represents modules as PyTrees, which are nested dictionaries of JAX arrays. This allows NN modules to remain compatible with JAX transformations. Optimization is handled through Optax, which provides functional implementations of common optimizers such as Adam, ensuring that the entire training pipeline remains compatible with JAX.

A.3 Proposed Algorithms

A.3.1 Batch-wise Randomized SVD

The batch-wise RandSVD method for finding supervision directions for second-order Sobolev training was introduced in section 3.2.

A.3.2 Streaming Refinement

The batch-wise RandSVD method proposed in 3.2 re-computes approximations of the curvature subspace for every batch. For functions that have globally constant dominant curvature directions, these calculations are redundant. Instead, we propose to approximate the population operator $\mathbb{E}_x[H_f(x)]$ over the entire training data and refine a single basis $Q_t \in \mathbb{R}^{d\times r}$ with $r < d$ over the course of training. The basis Q_0 is initialized with i.i.d. Gaussian values. For every new batch of inputs $\mathcal{B} = \{x_b\}_{b=1}^m$ the update of Q_t has two components, one for exploration and one for exploitation:

1) Exploration via random probes:
Similarly to our first method, we use a Gaussian test matrix $\Omega \in \mathbb{R}^{d\times r}$. However, instead of directly computing the sketch $Y = \frac{1}{m}\sum_b H_f(x_b)\Omega$, we first deflate Ω against the current basis Q_t:

$$\Omega_\perp = \left(I - Q_t Q_t^\top\right)\Omega.$$

The orthogonal projector removes directions already captured by Q_t and focuses exploration on its complement, which helps to recover eigenvectors that had little overlap with the initial basis. The exploration part of the sketch update is then formed as:

$$\Delta_Q^{\text{exploration}} = \frac{1}{m}\sum_{b=1}^{m} H_f\left(x_b\right)\Omega_\perp.$$

2) Exploitation via subspace iteration:
The second part performs an online variant of subspace iteration, mirroring the power-iteration used in RandSVD. The update:

$$\Delta_Q^{\text{exploitation}} = \frac{1}{m}\sum_{b=1}^{m} H_f\left(x_b\right)Q_t$$

over iterations, rotates Q toward the dominant invariant subspace of $\mathbb{E}_x[H_f(x)]$. The approach follows Oja's streaming version of subspace iteration used for example in online PCA, where repeated multiplication and orthonormalization leads to convergence to the top eigenvectors of the population operator [28, 31]. This step therefore exploits the structure already captured in Q_t, gradually sharpening it in the directions of strongest curvature.

Combined update:
Both components are combined, applied to Q_t after an orthogonal projection and the resulting basis is re-orthonormalized:

$$\Delta_Q = \left(I - Q_t Q_t^\top\right)[\Delta_Q^{\text{exploitation}} + \varepsilon\Delta_Q^{\text{exploration}}], \quad Q_{t+1} = \text{orth}(\beta Q_t + \eta\Delta_Q).$$

Here, $\eta \in [0.05, 0.2]$ is a small learning rate and $\varepsilon \approx 0.05$ controls the relative weight of exploration. A forgetting factor $\beta \in [0.98, 1]$ enables slow adaptation to drifting curvature. This is relevant for cases where either the sampling distribution changes during training (adaptive sampling) or the underlying function f itself changes. This may be the case when approximating not the curvature structure of the reference model but that of the surrogate, which changes during training.

Direction readout for supervision:
From the updated basis Q_{t+1}, $k \leq r$ supervision directions are extracted in the same way as in the batch-wise RandSVD method. First, the compressed representation of the batch-averaged Hessian is formed via HVPs

$$B = \frac{1}{m} \sum_{b=1}^{m} H_f(x_b)\, Q_{t+1}^{\top} \in \mathbb{R}^{r \times d},$$

then its left singular vectors are computed with a thin SVD $B = \widetilde{U}\Sigma V^{\top}$ in the reduced space with $\widetilde{U} \in \mathbb{R}^{r \times r}$ and lifted back to input space

$$U = Q_{t+1}\widetilde{U} \in \mathbb{R}^{d \times r}.$$

The results are truncated to the top k components $U_k \in \mathbb{R}^{d \times k}$, which are then used as supervision directions in the directional second-order Sobolev loss.

Algorithm 4 Streaming Refinement for dominant curvature directions via HVPs

```
Require: Function $f : \mathbb{R}^d \to \mathbb{R}$ (twice differentiable), batch $\mathcal{B} = \{x_b\}_{b=1}^{m}$, sketch size $r < d$,
    target rank $k \leq r$, step size $\eta \in [0.05, 0.2]$, forgetting factor $\beta \in [0.98, 1]$, exploration
    weight $\varepsilon \approx 0.05$, Gaussian test matrix $\Omega \in \mathbb{R}^{d \times r}$, current basis $Q_t \in \mathbb{R}^{d \times r}$ (initially
    Gaussian for $t = 0$)
 1: Orthogonal projection $\Omega_{\perp} \leftarrow \left(I - Q_t Q_t^{\top}\right)\Omega$
 2: Exploration update with HVPs:
        $$\Delta_Q^{\text{exploration}} \leftarrow \frac{1}{m}\sum_{b=1}^{m} H_f(x_b)\,\Omega_{\perp} \quad \text{(columns of } \Omega_{\perp} \text{ supplied to HVPs)}$$
 3: Exploitation update with HVPs:
        $$\Delta_Q^{\text{exploitation}} = \frac{1}{m}\sum_{b=1}^{m} H_f(x_b)\,Q_t \quad \text{(columns of } Q_t \text{ supplied to HVPs)}$$
 4: Combine updates:
        $$\Delta_Q = \left(I - Q_t Q_t^{\top}\right)[\Delta_Q^{\text{exploitation}} + \varepsilon \Delta_Q^{\text{exploration}}]$$
 5: Apply updates and orthonormalize through QR-decomposition:
        $$Q_{t+1} \leftarrow \text{orth}\left(\beta Q_t + \eta \Delta_Q\right)$$
 6: Compressed operator with HVPs:
        $$B^{\top} \in \mathbb{R}^{d \times r} \;\leftarrow\; \frac{1}{m}\sum_{b=1}^{m} H_f(x_b)\,Q_{t+1} \quad \text{(columns of } Q_{t+1} \text{ supplied to HVPs)}$$
 7: Compute the thin SVD of $B$: $B = \widetilde{U}\,\Sigma\,V^{\top}$ with $\widetilde{U} \in \mathbb{R}^{r \times r}$
 8: Lift $\widetilde{U}$ back to the original space: $U = Q_{t+1}\widetilde{U}$, with $U \in \mathbb{R}^{d \times r}$
 9: Truncate to the top $k$ components: $U_k \in \mathbb{R}^{d \times k}$
10: return $U_k$
```

A.3.3 3rd-Order Extension

The batch-wise method for finding supervision directions for third-order Sobolev training was introduced in section 3.3. Here we give a concise overview of the proposed algorithm.

Algorithm 5 Batch-wise sketching for dominant directions of the 3rd derivative tensor

```
Require: Function $f : \mathbb{R}^d \to \mathbb{R}$ (three times differentiable), batch of inputs $\mathcal{B} = \{x_b\}_{b=1}^m$,
    sketch size $s < d$, target rank $k_{3rd} \leq s$, optionally: dominant curvature directions $U_{k_{2nd}} \in$
    $\mathbb{R}^{d \times k_2}$ of $f$
1: Form sketch directions $\Omega \in \mathbb{R}^{d \times s}$ by combining $U_2$ with $s - k_{2nd}$ Gaussian random vectors
2: Sketch by evaluating TVVPs for all pairs of sketch directions via AD
       $$Y \in \mathbb{R}^{d \times s^2} \leftarrow \frac{1}{m} \sum_{b=1}^{m} T_f\left(x_b; \cdot, u_i, u_j\right), \quad 1 \leq i, j \leq s$$
3: Compute an orthonormal basis $Q \in \mathbb{R}^{d \times s^2}$ for the range of $Y$ via QR-decomposition:
    $Y = QR$
4: Rank the columns of $Q$ by
       $$M\left(q_i\right) = \frac{1}{m} \sum_{b=1}^{m} \left\|T_f\left(x_b; \cdot, q_i, q_i\right)\right\|_2^2, \quad 1 \leq i \leq s^2$$
5: Assemble the top $k_{3rd}$ columns as $U_{k_{3rd}} \in \mathbb{R}^{d \times k_{3rd}}$
6: return $U_{k_{3rd}}$
```

A.4 Loss Balancing for Higher-Order Sobolev Training

Training NN surrogates with Sobolev supervision requires joint minimization of value, gradient and higher-order loss terms. A central challenge is that these loss terms can differ in learnability and scale. As a result, a naive sum of losses implicitly prioritizes whichever term is numerically largest. Higher-order loss terms may vanish or dominate the objective, leading to unstable or biased training. Earlier approaches attempted to counteract this problem with manually tuned weights [6] or a heuristic rescaling that downweights derivative terms whenever a batch error exceeds that of the value term by a fixed factor [11]. We propose a two-layer weighting scheme, where the first layer consists of weights adapted from Huge [14] and Kichler [19]. These *base-weights* account for the number of supervised quantities per term and can encode user-controlled preferences. The second layer adds a *learned task uncertainty* based on the multi-task formulation of Kendall et al. [17], which adapts to differences in learnability and noise across terms.

Deterministic Base-Weights Through Structural Balancing

Imbalances arise from the differing number of supervised elements in each term. For functions $f : \mathbb{R}^d \to \mathbb{R}$, a single function value $f(x)$ is supervised, while the gradient has d, the Hessian d^2 and the third derivative d^3 components. Huge et al. [14] proposed a weighting based on the number of supervised elements:

$$\lambda_0^{\text{base}} = \frac{1}{1+\alpha d}, \quad \lambda_1^{\text{base}} = \frac{\alpha d}{1+\alpha d},$$

where α controls the relative importance of gradient supervision with a default of $\alpha = 1$. Kichler [19] extended this idea to second-order supervision by including a normalization based on the number of supervised curvature directions k_{2nd}.

$$c = 1 + \alpha d + \beta d^2, \quad \lambda_0^{\text{base}} = \frac{1}{c}, \quad \lambda_1^{\text{base}} = \frac{\alpha d}{c}, \quad \lambda_2^{\text{base}} = \frac{\beta d^2}{c},$$

with a default value of $\beta = \frac{2k_{2nd}}{d^2}$, where the factor of two accounts for symmetry in the Hessian. Extending this principle to third order leads to:

$$c = 1 + \alpha d + \beta d^2 + \gamma d^3, \quad \lambda_0^{\text{base}} = \frac{1}{c}, \quad \lambda_1^{\text{base}} = \frac{\alpha d}{c}, \quad \lambda_2^{\text{base}} = \frac{\beta d^2}{c}, \quad \lambda_3^{\text{base}} = \frac{\gamma d^3}{c},$$

with a default value $\gamma = \frac{6k_{3rd}^2}{d^3}$. The value for γ arises from supervising $k_{3rd} \cdot k_{3rd}$ direction pairs, with a factor of $3 \cdot 2 = 6$ to account for the number of unique combinations in the symmetric 3-tensor.

This normalization ensures that each loss term contributes proportionally to its number of supervised quantities. Base-weights alone, however, remain static and cannot adapt when one task is inherently harder to learn than the others. These differences in learnability can arise, for example, when training with Monte Carlo based derivative labels. Higher-order derivatives are noisier than lower-order ones (see Figure A.23) and the optimizer needs to see many examples to account for the same quality that a single value- or gradient label carries. The higher-order derivatives should therefore be weighted less. We propose to complement the base-weights with learned, task-specific weights.

Task-Uncertainty Weighting

Kendall et al. [17] proposed letting the NN learn per-task noise variances $s_i = \log \sigma_i^2$ alongside the networks parameters:

$$\mathcal{L}_{\text{total}}(\theta; x) = \sum_{i=0}^{3} \frac{1}{2} \exp\left(-s_i\right) \mathcal{L}_i(\theta; x) + \frac{1}{2} s_i,$$

where the exponential terms are the learned importance weights and the $\frac{1}{2}s_i$ are regularizations preventing $s_i \to \infty$.

This idea fits well with Sobolev training once the losses have been normalized through the base-weights λ_i^{base} as described above. The base-weights implement structural normalization and reflect coarse user preferences, the learned uncertainties perform data-driven adjustments in response to difficulty and noise. The full loss becomes:

$$\mathcal{L}_{\text{total}}(\theta; x) = \sum_{i=0}^{3} \left(\frac{1}{2} \exp\left(-s_i\right) \lambda_i^{\text{base}} \, \mathcal{L}_i(\theta; x) + \frac{1}{2} s_i \right).$$

When formulated as:

$$\mathcal{L}_{\text{total}}(\theta; x) = \sum_{i=0}^{3} \lambda_i \mathcal{L}_i(\theta; x),$$

the final weights are

$$\lambda_i = \frac{1}{2} \exp(-s_i) \lambda_i^{\text{base}},$$

and a regularization term

$$c_\lambda = \sum_{i=0}^{3} \frac{1}{2} s_i$$

needs to be added, leading to the final formulation in definition 5.

When training with noise-free, analytic derivatives, the losses for different orders decline at similar rates. The gradients with respect to the log variances s_i remain small and the learned weights stay close to the base-weights. Only in late epochs do small differences in learnability produce small deviations. Under noisy Monte Carlo based derivative labels, the situation changes. Especially higher-order losses become noisy and the corresponding uncertainty parameters s_i increase, reducing their effective weights. Forcing the NN to fit noisy, higher-order derivatives would harm its ability to learn values and gradients. As training continues and the network averages over more Monte Carlo estimations, the losses approach those of the analytic regime and the learned weights converge accordingly. Our experiments show that both mechanisms together form a robust approach to balancing losses for higher-order Sobolev training.

A.5 Reference Models and Additional Evaluation Results

A.5.1 Stochastic Option Pricing Models

Two stochastic pricing models from quantitative finance are used as reference functions in our evaluations. The Bachelier and Heston models describe the probabilistic evolution of asset prices over time and are used to value financial derivatives such as call options.

European Call Option:
A European call option gives its holder the right to buy an underlying asset at a future timepoint T for a fixed price K. Under the risk-neutral measure $\mathbb{Q}$ with constant risk-free interest rate r, the fair price C of a European call option is the discounted value of its expected payoff

$$C = e^{-rT}\mathbb{E}^{\mathbb{Q}}\left[(S_T - K)^+\right],$$

where S_T is the price of the underlying asset at maturity. It is often convenient to work with the T-forward price $F_t = S_t e^{r(T-t)}$ and the associated T-forward measure $\mathbb{Q}^T$. Note that at maturity $F_T = S_T$. The forward price of the option becomes

$$C^{\text{fwd}} = \mathbb{E}^{\mathbb{Q}^T}\left[(F_T - K)^+\right] = e^{rT}C.$$

To evaluate the expectation in advance, one must specify how the price of the underlying asset evolves from S_0 to S_T. The Bachelier and Heston models provide two stochastic formulations.

The Bachelier Model:
The Bachelier model [1] assumes additive Gaussian dynamics for the forward price:

$$dF_t = \sigma dW_t \mathbb{Q}^T$$

where σ is the constant volatility and W_t^{QT} is a Wiener process under $\mathbb{Q}^T$, that adds stochastic changes with mean zero. The formulation implies that F_T is normally distributed with mean F_0 and variance $\sigma^2 T$.
Assuming these dynamics, the undiscounted forward price of a European call option can be obtained through Monte Carlo simulations or be computed analytically as [34]

$$C^{\text{fwd}}_{\text{Bachelier}}\left(F_0, K, \sigma, T\right) = \left(F_0 - K\right)\Phi(z) + \sigma\sqrt{T}\varphi(z), \quad z = \frac{F_0 - K}{\sigma\sqrt{T}},$$

where $\Phi(\cdot)$ and $\varphi(\cdot)$ denote the standard normal cumulative distribution function and probability density function respectively. Similarly, sensitivities (Greeks) can be computed analytically or via algorithmic differentiation, see [3] for derivations.

As in [19], we consider a basket of m correlated assets to increase the input dimensionality of the reference function. With normalized weights $\omega \in \mathbb{R}^m$ and initial forward prices $F_0^{(i)} \in \mathbb{R}^m$, the basket forward price is

$$B_t = \sum_{i=1}^{m} \omega_i F_t^{(i)}.$$

Since in the Bachelier case, each component is normally distributed and jointly correlated, the basket itself is Gaussian. Consequently, the basket call option can be treated as a single Bachelier asset with variance determined by the correlations and volatilities of its individual components. Its price can then be evaluated using the same analytic formula as in the single-asset case.

The Bachelier model has only one curvature direction corresponding to the basket weight vector ω. To test higher-order Sobolev training on reference functions with richer curvature behavior, we include the Heston model in our evaluations.

The Heston Model:
The Heston model [12] replaces the constant volatility σ in the Bachelier model with a second stochastic process. It is defined by the stochastic differential equations

$$\begin{aligned} dS_t &= rS_t dt + \sqrt{v_t} S_t dW_t^{(S)} \\ dv_t &= \kappa\left(\theta - v_t\right) dt + \xi\sqrt{v_t} dW_t^{(v)} \\ \operatorname{corr}\left(dW_t^{(S)}, dW_t^{(v)}\right) &= \rho \end{aligned}$$

where r is again the risk-free rate, v_t represents the stochastic variance with long-run mean θ and mean-reversion rate κ. ξ is the volatility of volatility and ρ is the correlation between price- and variance changes generated by the Wiener processes W_t^S and W_t^v.

The Heston model has a semi-closed-form solution for the price of a European call option in the single asset case [12]:

$$C_{\text{Heston}}(S_0, v_0) = S_0\Pi_1 - e^{-rT}K\Pi_2,$$

where Π_1 and Π_2 are probabilities computed through characteristic function integrals. As stated in [5], given the characteristic function

$$\psi_{\ln(S_t)}^{\text{Heston}}(w) = e^{\left[C(t,w)\theta + D(t,w)v_0 + iw\ln\left(S_0 e^{rt}\right)\right]}$$

with:

$$\begin{aligned} C(t,w) &= a\left[r_- \cdot t - \frac{2}{\xi^2}\ln\left(\frac{1-ge^{-ht}}{1-g}\right)\right] \\ D(t,w) &= r_-\frac{1-e^{-ht}}{1-ge^{-ht}} \\ r_\pm &= \frac{\beta \pm h}{\xi^2}; h = \sqrt{\beta^2 - 4\alpha\gamma} \\ g &= \frac{r_-}{r_+} \\ \alpha &= -\frac{w^2}{2} - \frac{iw}{2}; \beta = a - \rho\xi iw; \gamma = \frac{\xi^2}{2}, \end{aligned}$$

the values of Π_1 and Π_2 are given by

$$\begin{aligned} \Pi_1 &= \frac{1}{2} + \frac{1}{\pi}\int_0^\infty \operatorname{Re}\left[\frac{\mathrm{e}^{-i\cdot w\cdot\ln(K)}\cdot\Psi_{lnS_T}(w-i)}{i\cdot w\cdot\Psi_{lnS_T}(-i)}\right]dw \\ \Pi_2 &= \frac{1}{2} + \frac{1}{\pi}\int_0^\infty \operatorname{Re}\left[\frac{\mathrm{e}^{-i\cdot w\cdot\ln(K)}\cdot\Psi_{lnS_T}(w)}{i\cdot w}\right]dw. \end{aligned}$$

Because the integrands decay rapidly, the integrals can be approximated on a finite grid to obtain a solution to the option price.

Since the semi-analytic solution is only defined for a single asset call, we construct a basket-like extension as a portfolio of individual Heston calls:

$$C_{\text{Heston}}^{\text{basket}}(x) = \sum_{i=1}^{m}\omega_i C_{\text{Heston}}\left(S_0^{(i)}, v_0^{(i)}\right),$$

which defines a smooth mapping from the concatenated input vector

$$x = \left(S_0^{(1)}, v_0^{(1)}, \dots, S_0^{(m)}, v_0^{(m)}\right)$$

to the basket price. Although this is not equivalent to a true call on the weighted sum of terminal asset prices

$$\mathbb{E}^{\mathbb{Q}}\left[\left(\sum_{i=1}^{m} \omega_i S_T^{(i)} - K\right)^{+}\right],$$

it still provides a smooth and curvature-rich mapping suitable for evaluating higher-order Sobolev training.

Implementation:
For our evaluation purposes, both analytic and Monte Carlo implementations are used. The analytic Bachelier formula provides exact, noise-free labels for prices and derivatives of all orders, serving as ground truth for test set evaluation. In addition, a Monte Carlo implementation of the Bachelier model is included to emulate practical scenarios where only noisy, simulation-based data is available. We follow the pathwise differentiation approach of Huge [14] and Kichler [19]:

Given a vector of initial prices $F_0^{(i)}$ and a fixed covariance matrix, we draw independent standard normal samples and correlate them by multiplication with the Cholesky factor of the covariance matrix to obtain price changes $\Delta W_T^{(i)}$. Terminal prices are then computed as

$$F_T^{(i)} = F_0^{(i)} + \Delta W_T^{(i)},$$

which corresponds to an Euler-Maruyama discretization of the Bachelier dynamics [19]. The basket value is formed as $B_T = \omega^\top F_T$ and a smoothed call option payoff is computed as

$$\tilde{C}\left(B_T, K\right) \approx \max\left(B_T - K, 0\right),$$

where the kink at K is replaced by a smooth polynomial transition. Concretely, we define the transition polynomial on $(-1, 1)$:

$$r(u) = \frac{u^6 - 5u^4 + 15u^2}{32} + \frac{1}{2}u + \frac{5}{32},$$

and set

$$\tilde{C}\left(B_T, K\right) = \begin{cases} 0, & t \leq -\varepsilon, \\ \varepsilon r\left(\frac{t}{\varepsilon}\right), & |t| < \varepsilon, \\ t, & t \geq \varepsilon \end{cases}$$

with $t := B_T - K$ and $u := \frac{t}{\varepsilon}$.
Averaging this payoff over many simulated paths yields a Monte Carlo estimate of the option price. Algorithmic differentiation through the entire computation provides derivatives with respect to the initial prices. The analytic Bachelier formula continues to serve as the ground truth for evaluation.

For the Heston model, derivatives are obtained by differentiating the semi-analytic solution using algorithmic differentiation. In principle, one could mirror the Bachelier construction and estimate Heston prices and Greeks via Monte Carlo simulation. However, in earlier experiments [19] such an approach led to practical issues for higher-order Sobolev training.

The approach produced pathwise second derivatives that were effectively zero on most sampled paths. This results in poor training signals for higher-order loss terms. We opt for the semi-closed-form implementation as a stable alternative. This choice isolates the effect of our Sobolev training methods from Monte Carlo artifacts and ensures that any observed behavior is attributable to the loss design, not to degenerate gradient estimates. The parameter choices for both models follow those used by Kichler [19] to ensure comparability.

Bachelier Parameters:
$T = 1.0, \quad K = 110, \quad \sigma = 0.2$
Initial spot prices $F_0^{(i)}$ are normally distributed around 100.

Heston Parameters:
$T = 1.0, \quad K = 100, \quad \theta = 0.09, \quad \rho = -0.3, \quad \kappa = 1, \quad \xi = 1, \quad r = 0$
Initial prices $S_0^{(i)}$ are sampled uniformly at random from $[50, 150]$.
Initial volatilities $v_0^{(i)}$ are sampled uniformly at random from $[0.01, 0.1]$.

Additional Evaluations under the Bachelier Model:

Complementary to the results described in Chapter 4, we include additional evaluations on the Bachelier model. Figure 4.2 shows that without the proposed balancing mechanism, continued training with static base-weights leads to overall higher errors when training under noisy derivative labels. Figure A.2 and Figure A.3 show that when labels are averaged over many MC estimates, the loss curves become similar to the analytic setting (Figure 4.1) and the learned weights remain close to the base-weights.

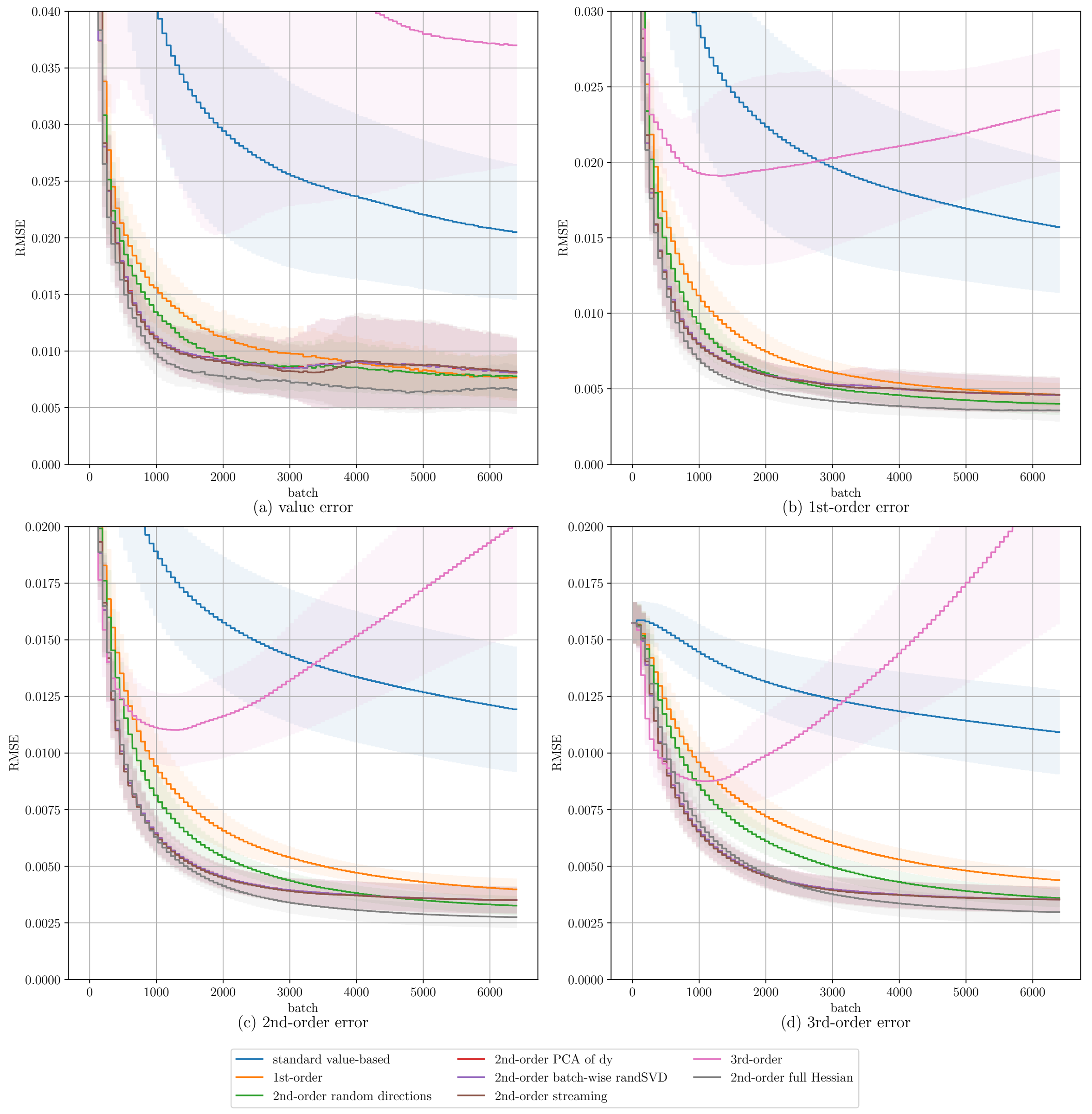


Figure A.1: Test-set RMSEs over training for the Monte Carlo simulated Bachelier basket call (basket size 7) with pathwise derivative, using static base-weight balancing. Curves show mean RMSE across 10 random seeds with shaded ±1 standard deviations. Panels show errors in the predicted values and the learned derivatives.

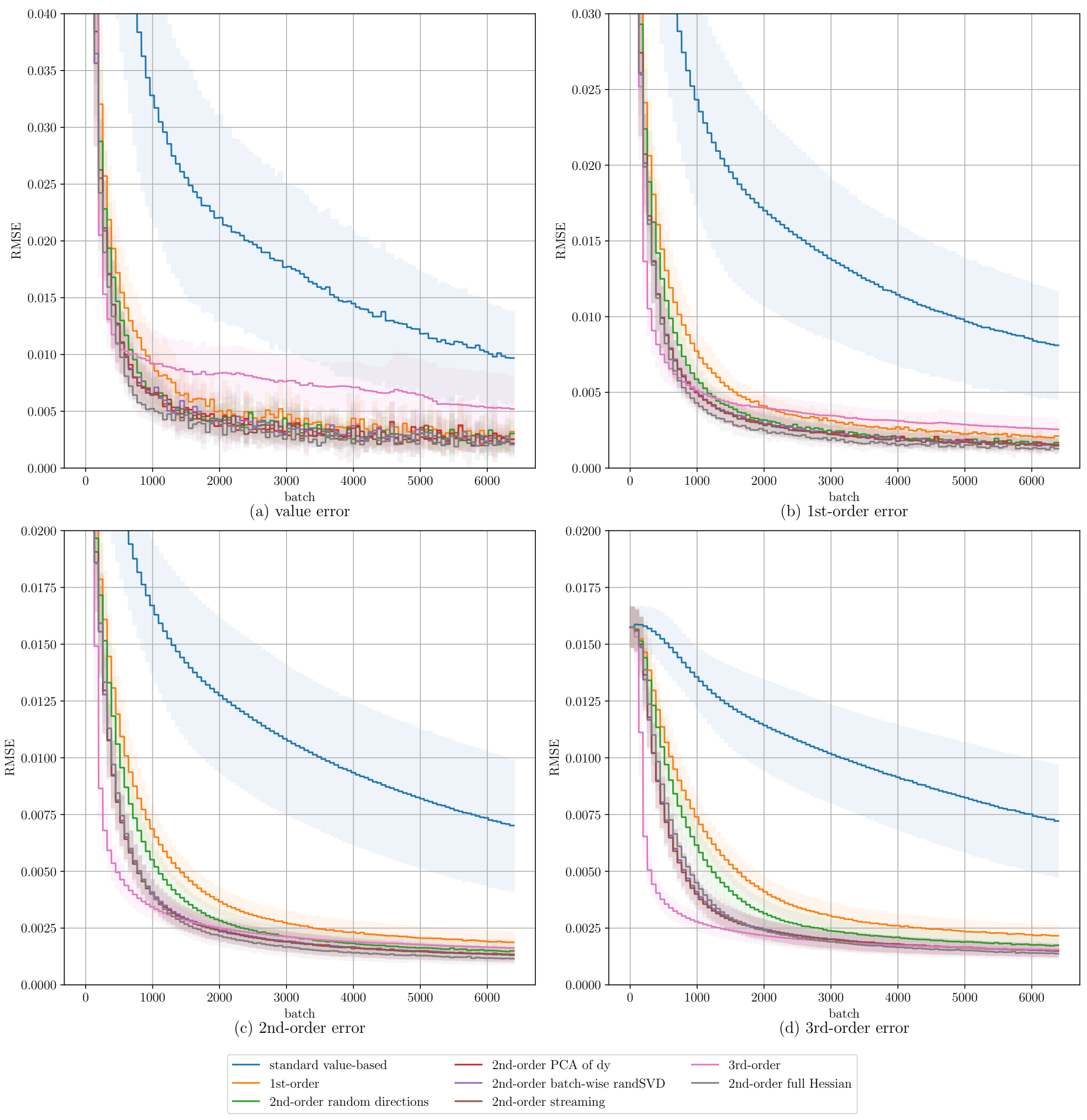


Figure A.2: Test-set RMSEs over training time for the Monte Carlo simulated Bachelier basket call (basket size 7) with derivative labels averaging 1000 paths each. Curves show mean RMSE across 10 random seeds with shaded ±1 standard deviations. Panels show errors in the predicted values and the learned derivatives.

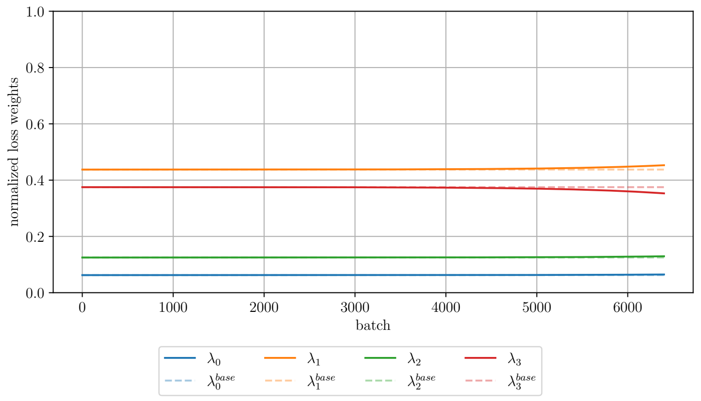


Figure A.3: Normalized loss weights over training for the Monte Carlo simulated Bachelier basket call (basket size 7) with labels averaging 1000 paths each.

Figures A.4 to A.7 show that higher-order derivative supervision leads to visually more accurate predictions on the test.

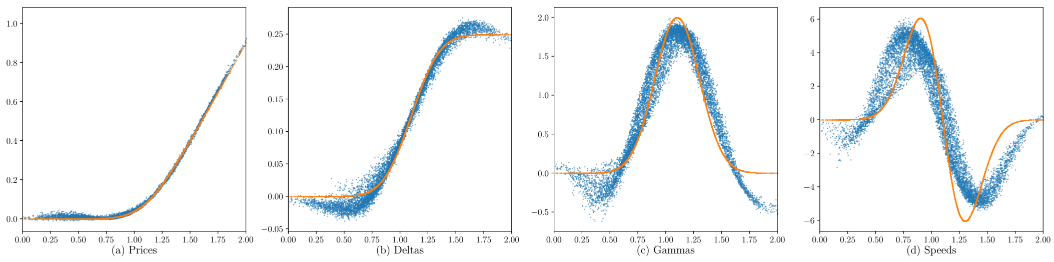


Figure A.4: Value-only supervision. Final predictions on the test set for the analytic Bachelier basket call (basket size 7). Analytically derived targets are shown in orange. Predictions and learned derivatives of the surrogate are shown in blue.

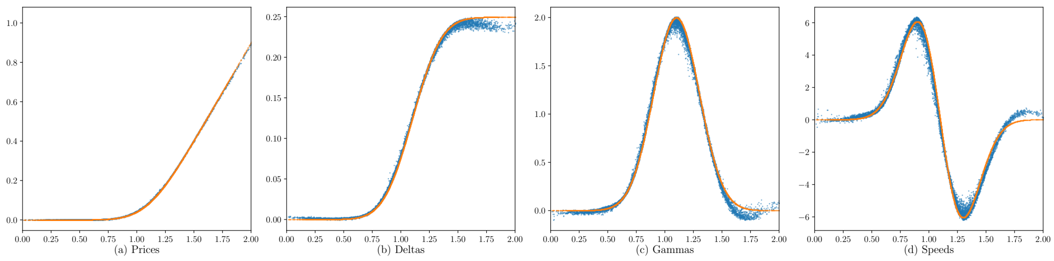


Figure A.5: Value- and first derivative supervision. Final predictions on the test set for the analytic Bachelier basket call (basket size 7). Analytically derived targets are shown in orange. Predictions and learned derivatives of the surrogate are shown in blue.

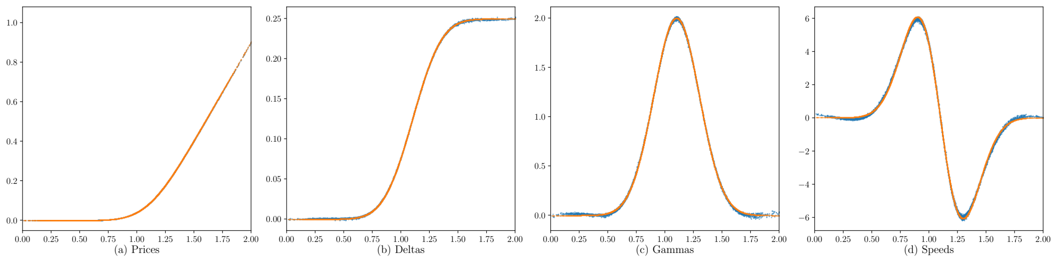


Figure A.6: Up to second-order batch-wise RandSVD supervision. Final predictions on the test set for the analytic Bachelier basket call (basket size 7). Analytically derived targets are shown in orange. Predictions and learned derivatives of the surrogate are shown in blue.

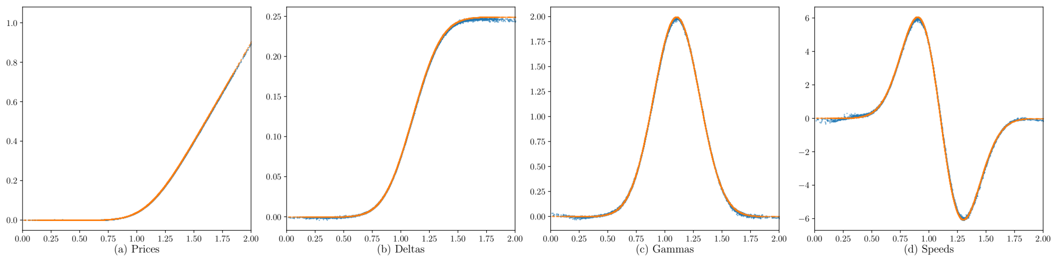


Figure A.7: Up to directional third-order supervision- Final predictions on the test set for the analytic Bachelier basket call (basket size 7). Analytically derived targets are shown in orange. Predictions and learned derivatives of the surrogate are shown in blue.

Evaluations under the Heston Model:

Figure A.8 below, shows the test-set RMSEs of all methods over 100 epochs of training on the Heston model (basket size 7). The error curves are averaged over ten random seeds and the standard deviations are shown by shaded areas around the curves. The supervision of first derivatives already leads to significant improvements in sample-efficiency. Inclusion of second- and third-order loss terms leads to additional improvements in the accuracies of the learned derivatives, with similar results for all second-order methods.

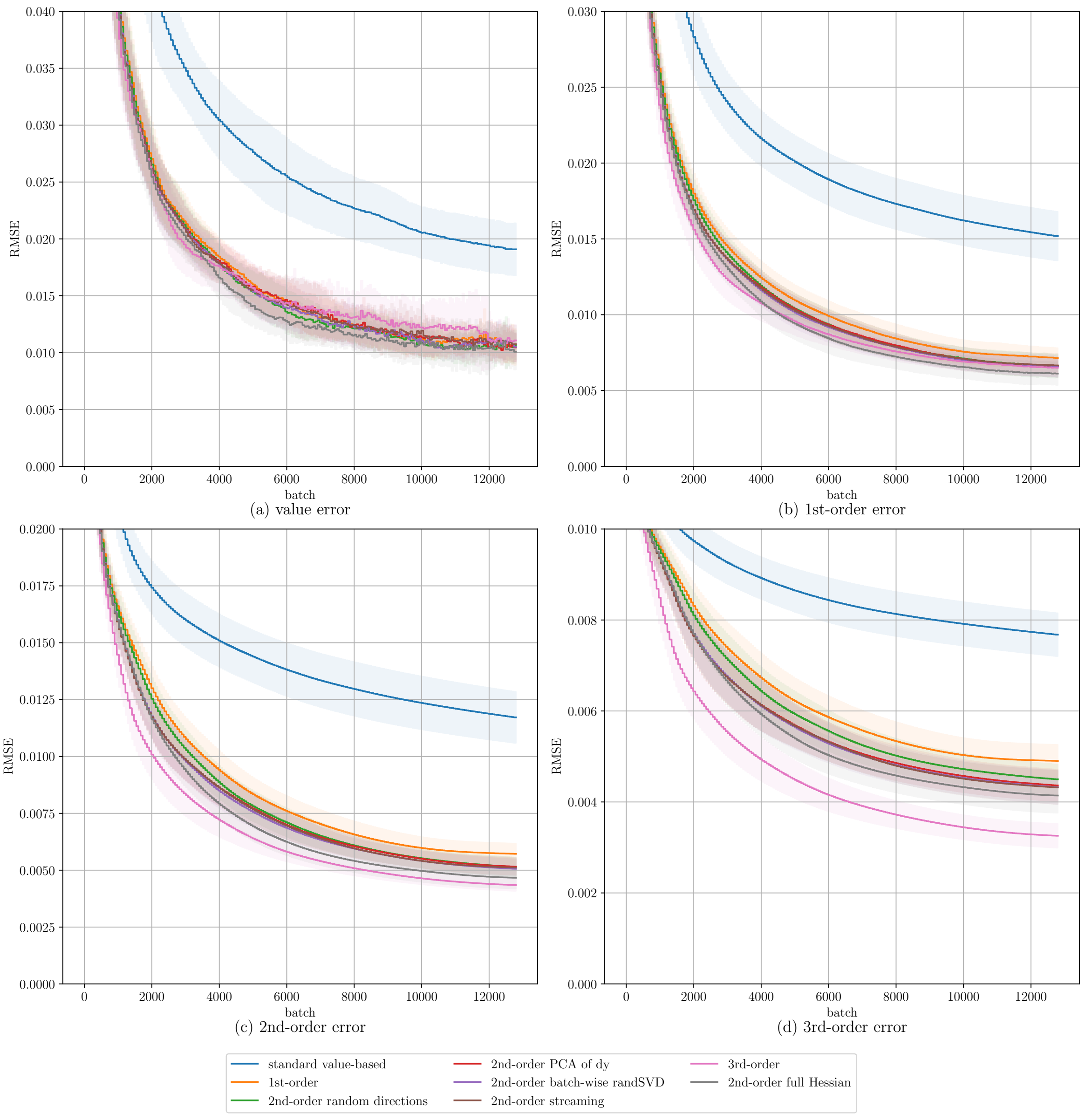


Figure A.8: Test-set RMSEs over training for the semi-analytic Heston basket call (basket size 7). Curves show mean RMSE across 10 random seeds with shaded ± 1 standard deviations. Panels show errors in the predicted values and the learned derivatives.

A.5.2 Analytic Benchmark Functions

This section introduces additional analytic benchmark functions used for our evaluations. Their closed forms allow exact computation of derivatives of all orders via AD. This makes them suitable for clean analyses of different Sobolev training methods. For each function, we present error curves over the course of training. All experiments are carried out in $d = 20$ dimensions with $k = 3$ supervision directions, $p = 5$ oversampling directions, $q = 0$ power-iteration steps, and $r = 8$ streaming sketch directions. Adding power-iteration did not impact the results. We use a batch size of 128 and 64 batches per epoch.

Rotated Hyper Ellipsoid:

The Rotated Hyper Ellipsoid function is a smooth convex quadratic [33].

$$f(\mathbf{x}) = \sum_{i=1}^{d} \sum_{j=1}^{i} x_j^2$$

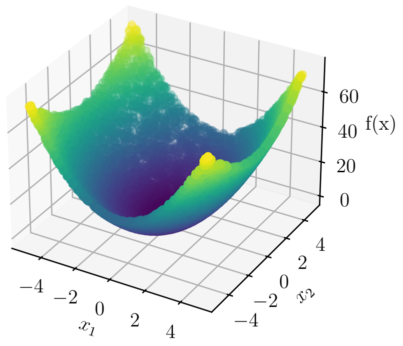


Figure A.9: Rotated-Hyper-Ellipsoid function (d=2)

Gradients are linear in x, the Hessian is globally constant with linearly decaying eigenvalues, and all higher derivatives are zero.
Inputs are sampled uniformly at random from $[-5, 5]$. Figure A.10 shows the test-set RMSEs of all methods over 300 epochs of training (d=20). The error curves are averaged over ten random seeds and the standard deviations are shown by shaded areas around the curves. It can be observed that inclusion of the third-order loss term yields almost no benefit in sample-efficiency, as all third derivatives are zero. All second-order methods, including full Hessian supervision, perform almost the same. As shown in Figure A.11, the learned balancing factors start to diverge from their base values after 100 epochs, putting more weight on the second- and third-order terms. This re-weighting leads to differences in the final test-set RMSEs by:

value error:	$+0.006477$ ($+12\%$)
1st-order error:	$+0.000122$ ($+0.76\%$)
2nd-order error:	-0.000939 (-19%)
3rd-order error:	-0.000389 (-27%)

compared to continued training with static base-weights.
Figure A.12 shows alternative trajectories for the learned loss weights. It illustrates how for different starting configurations, the learned weights still converge to the same end values.

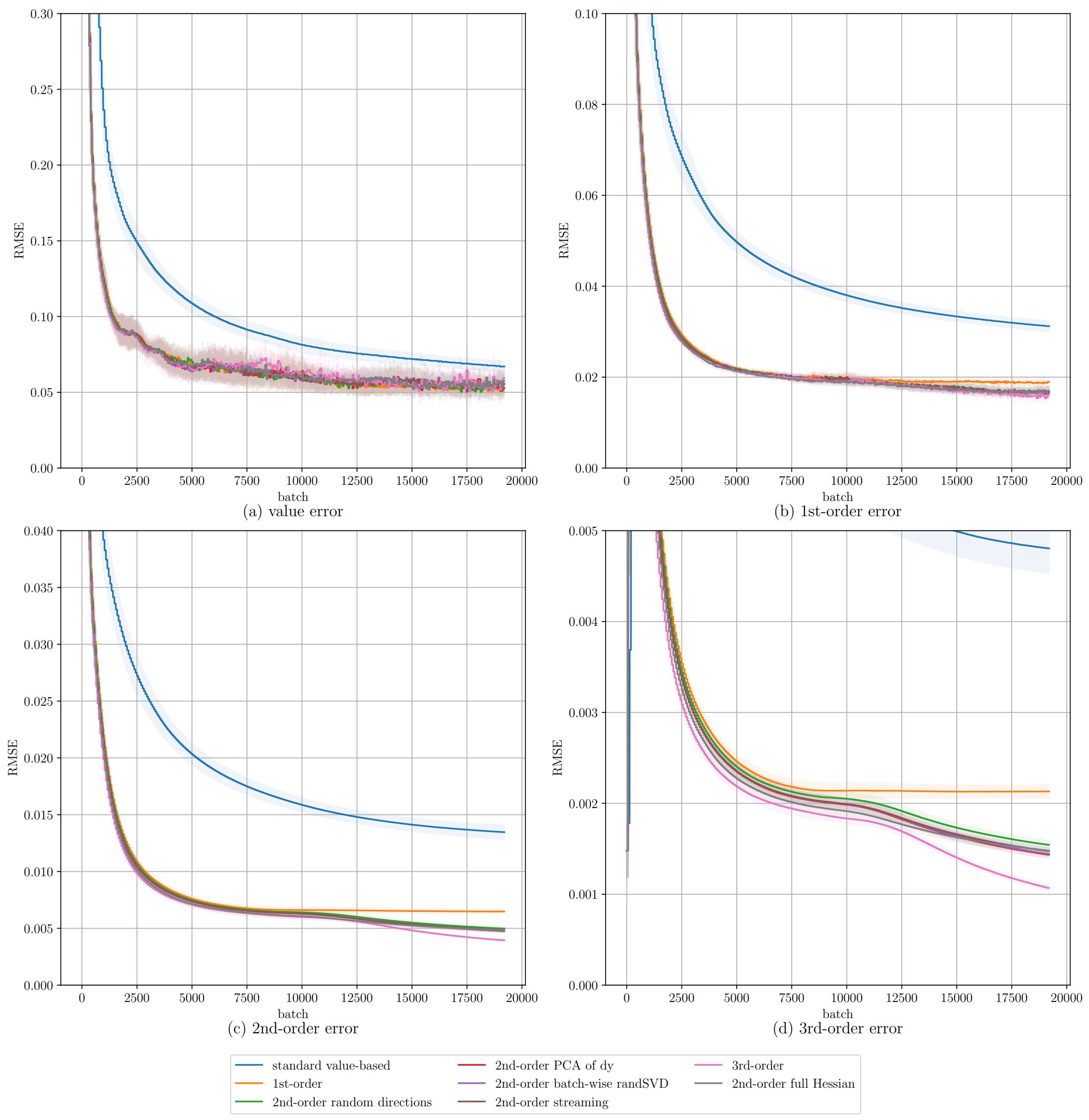


Figure A.10: Test-set RMSEs over training for the analytic Rotated-Hyper-Ellipsoid function (d=20). Curves show mean RMSE across 10 random seeds with shaded ±1 standard deviations. Panels show errors in the predicted values and the learned derivatives.

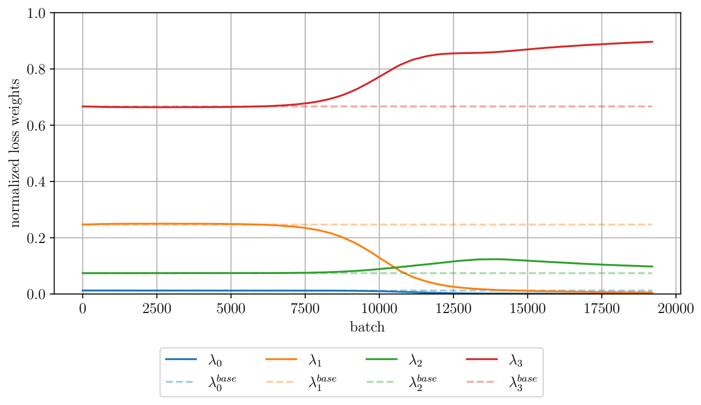


Figure A.11: Normalized loss weights over training for the analytic Rotated-Hyper-Ellipsoid function (d=20).

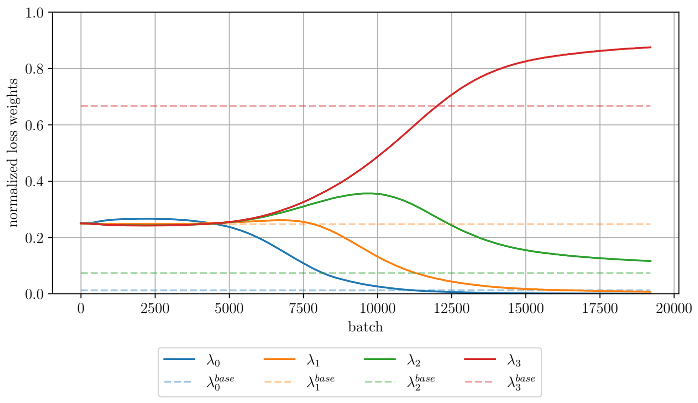


Figure A.12: Normalized loss weights over training for the analytic Rotated-Hyper-Ellipsoid function (d=20). Starting with equal weights instead of base-weights.

Cubic-Rank-r:

The Cubic-Rank-r function is a low rank cubic model with exponentially decaying weights.

$$f(x) = \sum_{i=1}^{r} \lambda_i x_i^3, \quad \lambda_i = 2^{-(i-1)}$$

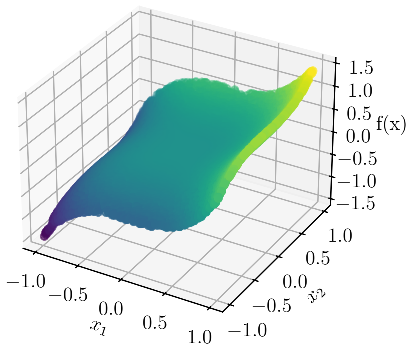


Figure A.13: Cubic-Rank-r function (d=2, r=2)

Gradients are quadratic, curvature is confined to the first r standard basis vectors with eigenvalues and ordering dependent on x. The third derivatives are nonzero only in the first r coordinates.
Inputs are sampled uniformly at random from $[-1, 1]$. Figure A.14 shows the test-set RMSEs of all methods over 100 epochs of training (d=20, r=3). The error curves are averaged over ten random seeds and the standard deviations are shown by shaded areas around the curves. Due to the low rank structure ($r = 3$), inclusion of first-, second- and third-order loss terms leads to steep improvements in sample-efficiency. It can also be observed that the batch-wise RandSVD method performs better than full Hessian supervision. A potential explanation for this phenomenon is that when all second-order directions are supervised, the optimization signal becomes dominated by the requirement to drive many curvature directions toward zero rather than focusing on accurately fitting the few meaningful non-zero directions.

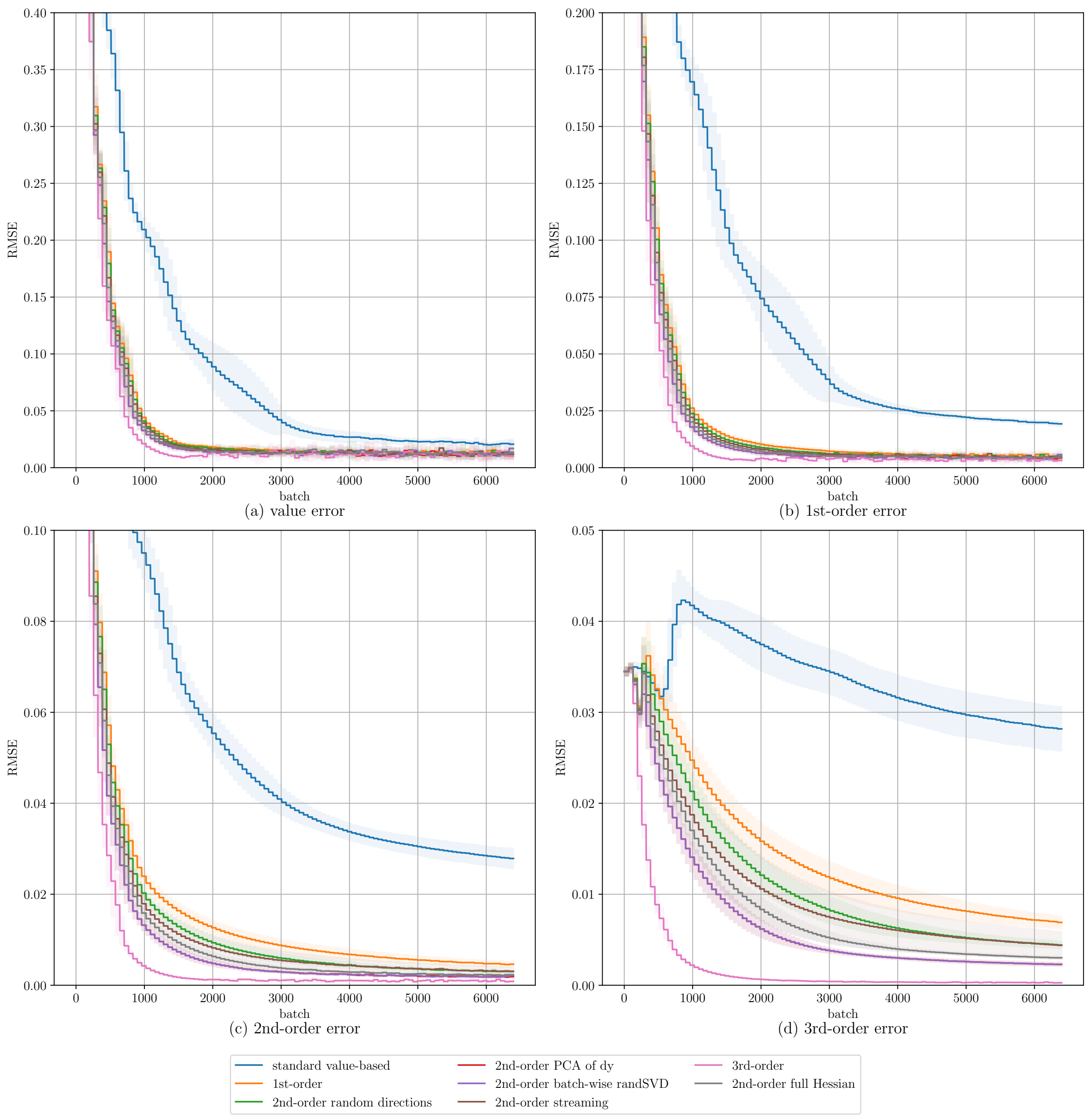


Figure A.14: Test-set RMSEs over training for the analytic Cubic-Rank-r function (d=20, r=3). Curves show mean RMSE across 10 random seeds with shaded ±1 standard deviations. Panels show errors in the predicted values and the learned derivatives.

Rosenbrock:

The Rosenbrock function is a classic test problem with a narrow, curved valley [33].

$$f(x) = \sum_{i=1}^{d-1} \left(100 \left(x_{i+1} - x_i^2 \right)^2 + (1 - x_i)^2 \right)$$

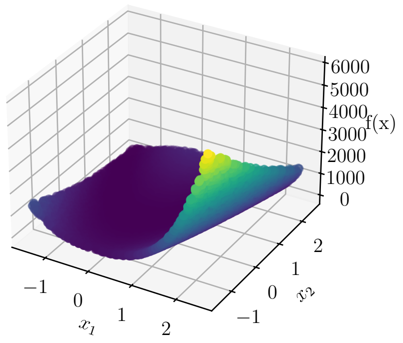


Figure A.15: Rosenbrock function (d=2)

Gradients vary strongly with position, the dominant curvature directions rotate along the valley, and non-zero third derivatives are sparse.
Inputs are sampled uniformly at random from $[-1.5, 2.5]$. Figure A.16 shows the test-set RMSEs of all methods over 100 epochs of training (d=20). The error curves are averaged over ten random seeds and the standard deviations are shown by shaded areas around the curves. It can be observed that the inclusion of first derivatives in the loss leads to improved sample-efficiency. Second- and third-order supervision yield additional minor benefits in the accuracies of higher-order derivatives.

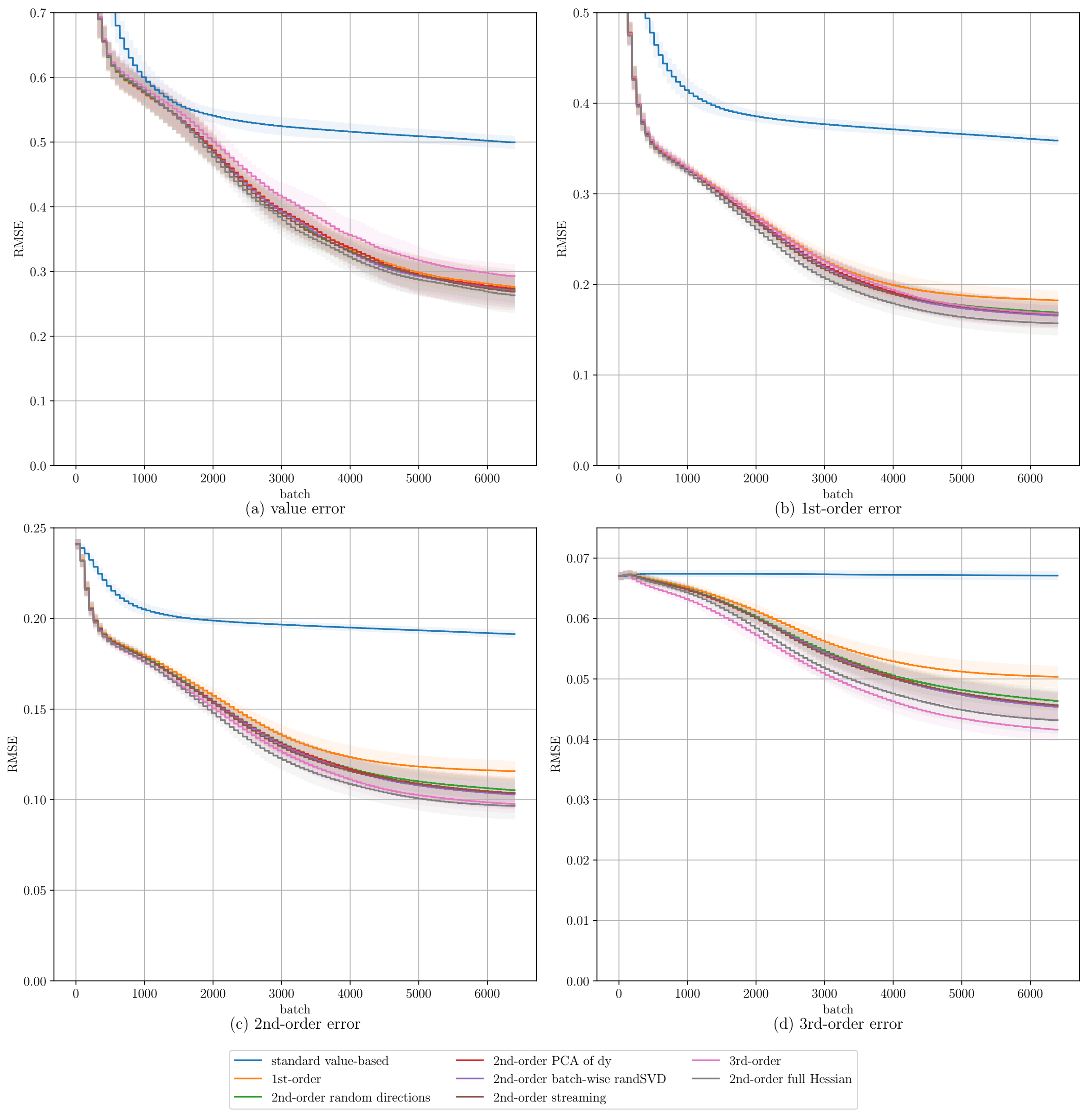


Figure A.16: Test-set RMSEs over training for the analytic Rosenbrock function (d=20). Curves show mean RMSE across 10 random seeds with shaded ±1 standard deviations. Panels show errors in the predicted values and the learned derivatives.

Ackley:

The Ackley function combines an exponential decay with a cosine average [33].

$$f(x) = -ae^{-br} - e^{m_{\cos}} + a + e, \quad r = \sqrt{\frac{1}{d}\sum_i x_i^2}, \quad m_{\cos} = \frac{1}{d}\sum_i \cos(cx_i)$$

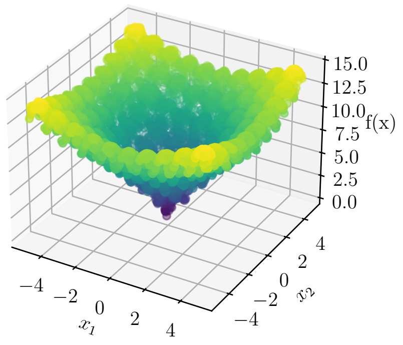


Figure A.17: Ackley function (d=2, a=20, b=0.2, c=2π)

Gradients and dominant curvature directions vary strongly with position and third derivatives are nonzero.
Inputs are sampled uniformly at random from $[-5, 5]$. Figure A.18 shows the test-set RMSEs of all methods over 100 epochs of training (d=20). The error curves are averaged over ten random seeds and the standard deviations are shown by shaded areas around the curves. First- and second-order supervision improves the accuracies of value- and gradient predictions. However, because of the strongly varying curvature, second- and third derivatives are pretty much unlearnable for the surrogate.

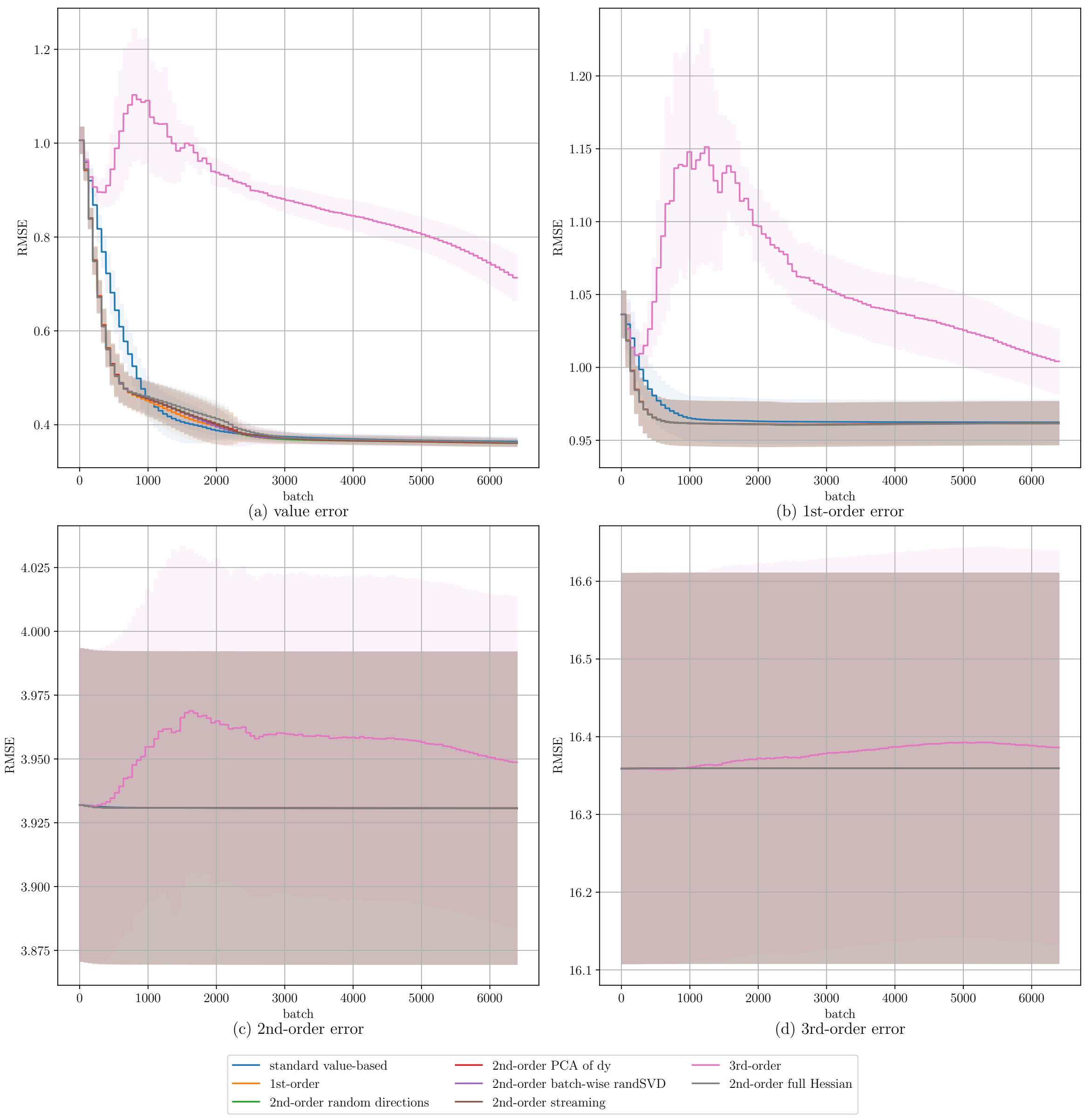


Figure A.18: Test-set RMSEs over training for the analytic Ackley function (d=20). Curves show mean RMSE across 10 random seeds with shaded ±1 standard deviations. Panels show errors in the predicted values and the learned derivatives.

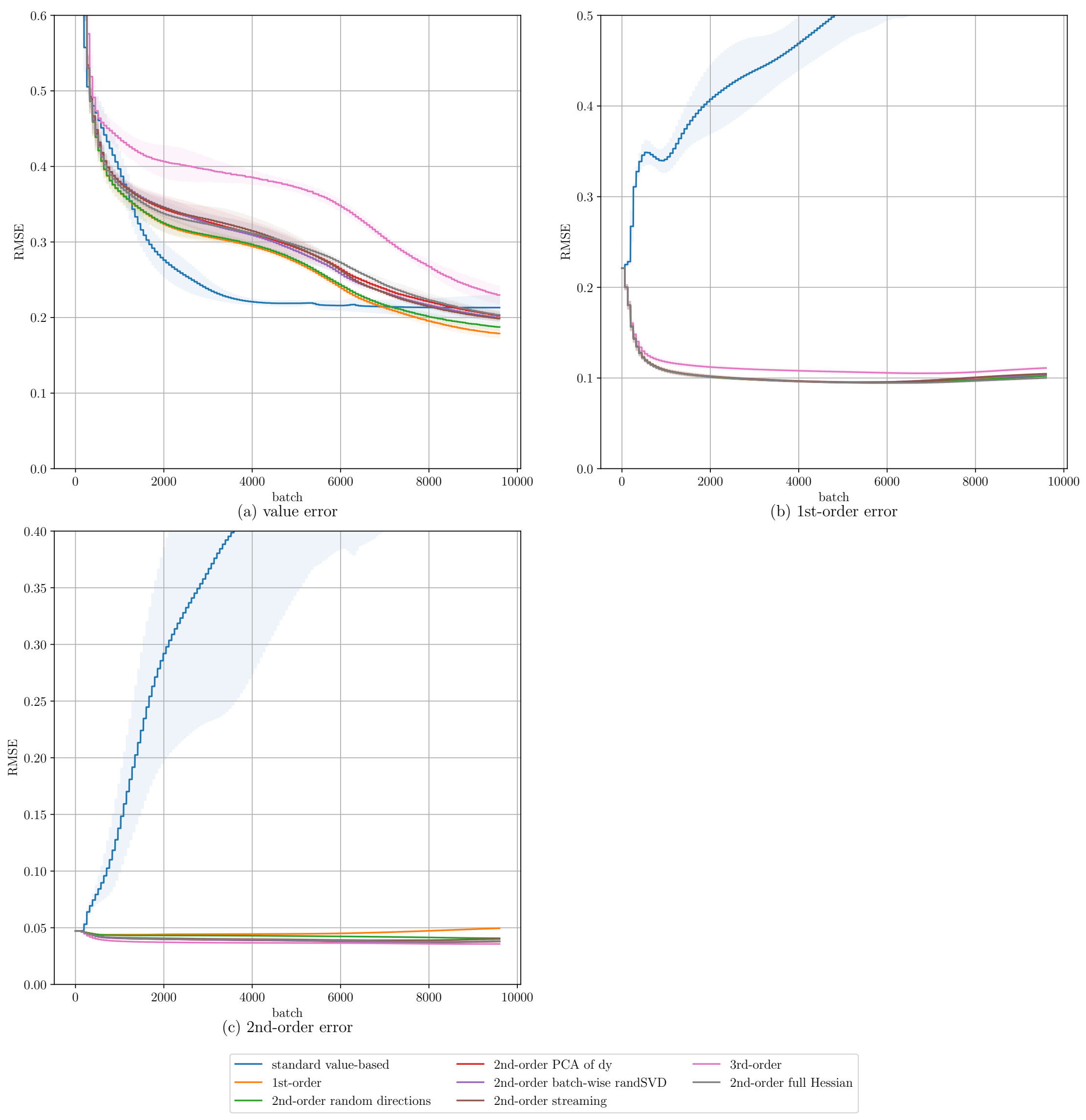
A.5.3 CNN Distillation

A small Convolutional Neural Network (CNN) is trained on the MNIST dataset. The scalar value of the reference function is taken as the *confidence* that the trained CNN believes the given image to represent the digit 9.
More specifically, the images are downsampled by a factor of 0.5 to a resolution of 14×14, so that every input image corresponds to a vector in $\mathbb{R}^{196}$. The CNN receives the unflattened 14×14 image, applies three convolutional blocks followed by a linear classifier, and outputs ten logits. The reference value is taken as the logit of the target class (9) minus the logit of the predicted class (maximum logit value) whenever the prediction differs from the target.
This produces a scalar output that varies smoothly with all input pixels and can therefore be differentiated efficiently to obtain gradients and higher-order derivatives for Sobolev training. The CNN itself is trained only once. All subsequent evaluations of the reference function and its derivatives use this fixed network.

For our evaluations we use a batch size of 128 and 64 batches per epoch. We use $k = 10$ supervision directions, $p = 20$ oversampling directions, $q = 0$ power-iteration steps, and $r = 30$ streaming sketch directions. Adding power-iteration did not impact the results significantly. Figure A.19 shows the test-set RMSEs of all methods over 150 epochs of training. Interestingly, higher-order derivative supervision does not lead to improved sample-efficiency for value predictions under this reference model. However, it does lead to improved final accuracies and significantly better first- and second-order predictions compared to value-only supervision. Without supervising derivatives of any order, the learned derivatives get worse over the course of training. As shown in Figure A.20, the learned balancing factors start to diverge from their base values after 75 epochs, putting less weight on the third-order loss term and more on the second-order one. This leads to differences in the final test-set RMSEs by:

value error:	-0.142	(-38%)
1st-order error:	$+0.0427$	($+4.3\%$)
2nd-order error:	-0.000787	(-2.2%)

compared to continued training with static base-weights. We do not report metrics on the learned third derivatives of the surrogate, as a third-order test-set of sufficiently many inputs exceeded the memory limitations of our system.



(a) value error

(b) 1st-order error

(c) 2nd-order error

Figure A.19: Test-set RMSEs over training for the CNN-based reference function (d=196). Curves show mean RMSE across 10 random seeds with shaded ±1 standard deviations. Panels show errors in the predicted values and the learned derivatives.

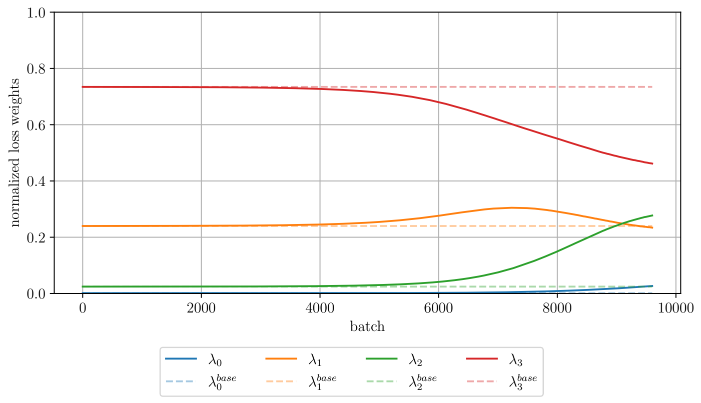


Figure A.20: Normalized loss weights over training for CNN-based reference function (d=196).

A.6 Additional Figures

Figures A.21, A.22 and A.23 visualize how MC approximated training labels become noisier for higher-order derivatives and how that noise is counteracted by averaging over many simulation paths. Figure A.24 shows the decaying eigenspectrum of the averaged Hessian for the Heston model (basket size 7), motivating directional curvature supervision.

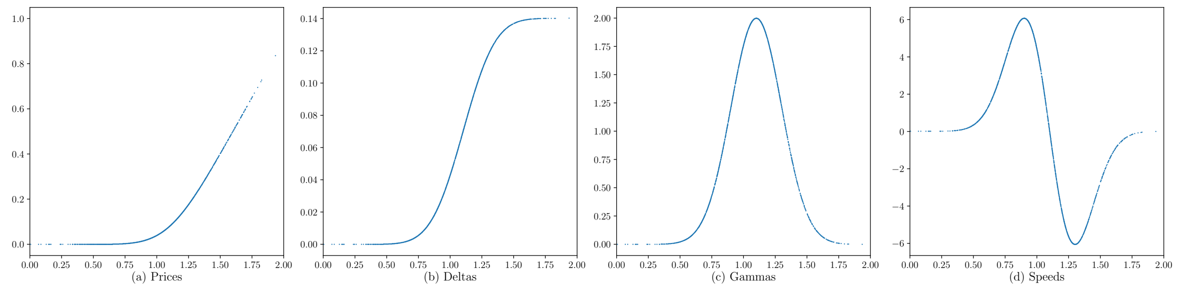


Figure A.21: Analytic test- and training labels for the Bachelier basket call (basket size 7)

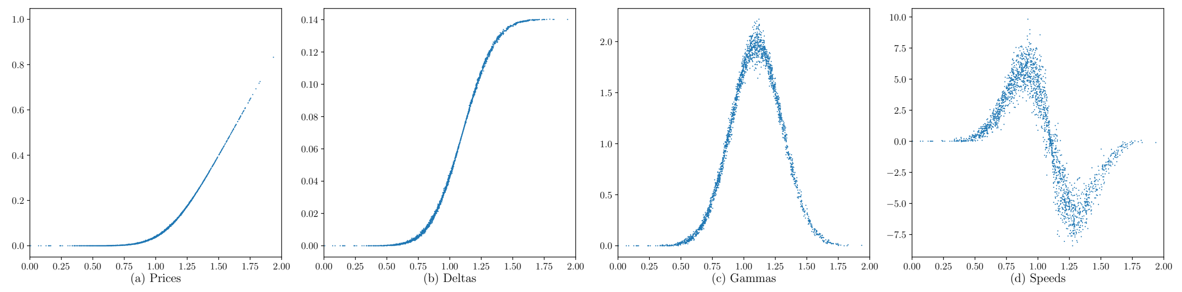


Figure A.22: Simulated training labels for the Bachelier basket call (basket size 7) averaged over 1000 paths

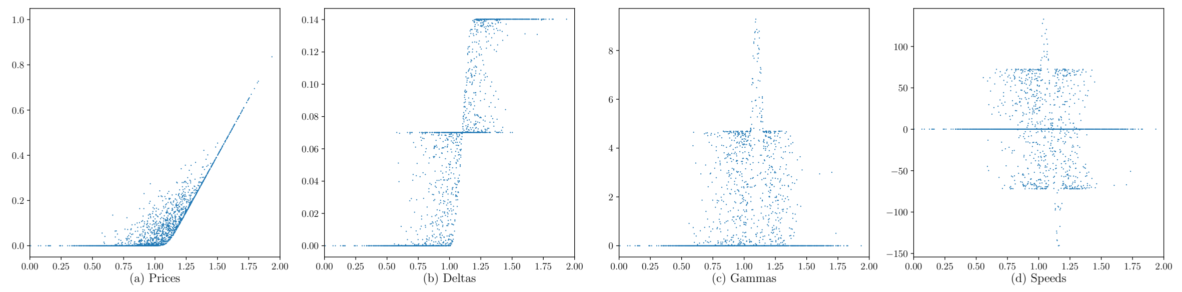


Figure A.23: Pathwise simulated training labels for the Bachelier basket call (basket size 7)

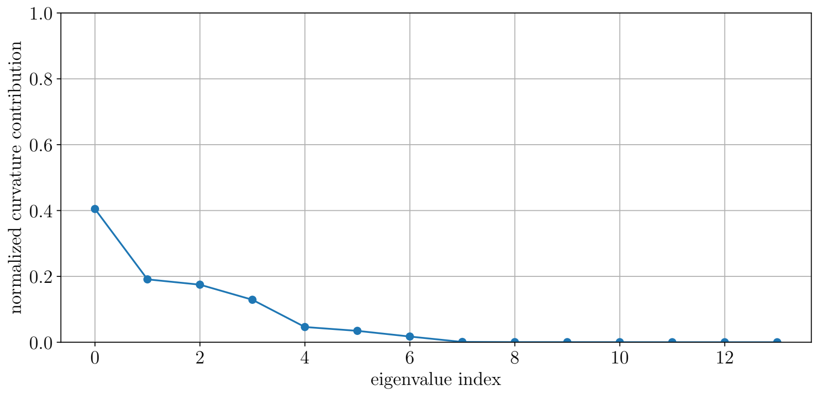


Figure A.24: Normalized squared eigenvalues of the mean Hessian of the Heston model (basket size 7). The eigenvalues are sorted in descending order, and the plot shows their normalized squared magnitudes, illustrating that most curvature is concentrated in the first few eigen-directions.

B Software Documentation

The codebase developed for this thesis is written in Python/JAX [2]. The repository is available on GitHub at https://github.com/JoWilhelm/diff-ml-rand-svd. It can be installed in editable mode by cloning the repository and running:

```
python -m pip install -e .
```

from the top-level directory.

Minimal Training Script:
A minimal example demonstrating a full training run is provided in *examples/example.py*. The script sets up a reference model, builds a test set, configures the surrogate MLP, optionally initializes a streaming sketch object (for the streaming method) and invokes the main training loop. Because JAX needs explicit state, it also sets up a pseudo-random number generator (PRNG) key. For each use, it gets split into two so that one key can get passed on.

```python
import jax
...
from diff_ml.reference_models.bachelier import Bachelier
from diff_ml.nn.train import train

key = jrandom.PRNGKey(42)

# reference function
key, subkey = jrandom.split(key)
basket_dim = 7
n_paths = 100 # number of MC paths per label. Set to 0 to use analytic
    formula.
ref_model = Bachelier(
    subkey,
    basket_dim=basket_dim,
    weights=jrandom.uniform(subkey, shape=(basket_dim,), minval=1.0,
        maxval=10.0),
    n_paths_per_label=n_paths
)
test_set = ref_model.get_test_set(n_samples=1024, order=3)

# evaluation parameters
variant = "batchSVD"
k = 1
streaming_r = 2
oversampling_p = 1
power_iteration_q = 0
n_epochs = 100
n_batches_per_epoch = 64
batch_size = 128
lr = 1e-3

# surrogate NN
input_dims = ref_model.n_dims
key, subkey = jax.random.split(key)
```

```
mlp = eqx.nn.MLP(key=subkey, in_size=input_dims, out_size="scalar",
    width_size=20, depth=3, activation=jax.nn.silu)
key, subkey = jax.random.split(key)
mlp = init_linear_weight(mlp, trunc_init, subkey)
surrogate_model = mlp

optim = optax.adam(learning_rate=lr)

# optional streaming sketch object
if variant == "streaming":
    key, subkey = jax.random.split(key)
    sketch = StreamingHessianSketch(
                ref_model=ref_model,
                r=streaming_r,
                k=k,
                key=subkey)
else:
    sketch = None

# call the main training loop
trained_surrogate, _, _, _ = train(
                        surrogate_model,
                        test_set,
                        optim,
                        n_epochs,
                        n_batches_per_epoch,
                        batch_size,
                        ref_model,
                        sketch,
                        variant,
                        k,
                        oversampling_p,
                        power_iteration_q,
                        learnable_loss_weights=True,
                        do_approx_metrics=False
                        )
```

Overview of the Training Loop:
Inside the `train(...)` function, the training-PRNG-key of the reference model gets split into multiple batch-keys. Each iteration, a batch of inputs is sampled and the combined Sobolev loss is computed by `total_loss_fn(...)`, where loss terms of different orders are computed, weighted by the balancing mechanism, and combined. After computing the gradients of the loss w.r.t. the surrogates' parameters and the learnable uncertainty-weights, the optimizer applies the update step. The procedure continues for the specified number of iterations. At the end of each epoch, the current surrogate is evaluated against the test set.
To compute the second-order loss term, the function `second_order_loss_fn(...)` is called inside `total_loss_fn(...)`. It first builds a set of supervision directions by either a) choosing random directions, b) PCA of gradients, c) batch-wise RandSVD d) refining the streaming sketch, or e) using all coordinate directions for full Hessian supervision. In the case of c), `get_rand_SVD_directions(...)` in *losses/directions.py* returns the set of supervision directions for the batch:

```
def hvp_power_iterated_sketch(f, x, sketch_directions, q, key):
    Y = hvp_batch(f=f,
                  inputs=x,
                  directions=sketch_directions,
                    batch_key=key
        )
    Y = jnp.mean(Y, axis=0)
    for _ in range(q):
        Y, _ = jnp.linalg.qr(Y.T)
        key, subkey = jax.random.split(key)
        Y = hvp_batch(f=f, inputs=x, directions=Y.T, batch_key=subkey)
        Y = jnp.mean(Y, axis=0)
    return Y, key


def get_rand_SVD_directions(ref_model, f, x, k, key, oversampling_p=0,
    power_iteration_q=0, kappa=0.95):

    s = k + oversampling_p
    key, subkey = jax.random.split(key)
    sketch_directions = generate_random_vectors(shape=(s, ref_model.
       n_dims), key=subkey, normalize=True)

    # sketch with power-iteration
    key, subkey = jax.random.split(key)
    Y, key = hvp_power_iterated_sketch(
        f=f,
        x=x,
        sketch_directions=sketch_directions,
        q=power_iteration_q,
        key=subkey
    )
    Y = Y.T # (d, s)

    # orthonormalize Y
    Q, _ = jnp.linalg.qr(Y) # (d, s)

    # project via HVPs
    # each row of B is H @ q_i
    key, subkey = jax.random.split(key)
    B_rows = hvp_batch(
        f=f,
        inputs=x,
        directions=Q.T,
        batch_key=subkey
    )
   B_rows = jnp.mean(B_rows, axis=0)
   B = jnp.stack(B_rows, axis=0) # (s, d)

    # thin SVD on B
    U_tilde, S, _ = jnp.linalg.svd(B, full_matrices=False) # (s, s)

    # lift back
    U = Q @ U_tilde
    U = U.T # (s, d)

    dirs = safe_normalize_vectors(U, axis=-1)
    dirs = dirs[:k, :]  # truncate (k, d)
    return dirs
```

Once the dominant curvature directions have been identified, HVPs of the surrogate and the reference function are compared along these directions. The resulting mean-squared error is the second-order component of the combined Sobolev loss:

```
target_hvps = hvp_batch(
    f=ref_fn,
    inputs=x,
    directions=directions,
    batch_key=key
)
pred_hvps = hvp_batch(
     f=MakeScalar(model),
     inputs=x,
     directions=directions
 )
second_order_loss = mse(pred_hvps, target_hvps)
```

Batched HVPs:
The function `hvp_batch(...)` in *ad.py* computes HVPs at every point in a batch, into every provided direction. It is implemented as a nested `jax.vmap` over both inputs and directions, optionally passing individual PRNG-keys when the reference function is stochastic:

```
def hvp(f, x, v, *f_args):
    return jax.jvp(lambda x_: eqx.filter_grad(f)(x_, *f_args), (x,), (
       v,))[1]

def hvp_batch(f, inputs, directions, batch_key=None):
    if batch_key is not None:
        b = inputs.shape[0] # number of elements in the batch
        keys = jax.random.split(batch_key, b)
        def hvp_one_point(x, key):
            def hvp_v(v):
                return hvp(f, x, v, key)
            return jax.vmap(hvp_v)(directions) # (k, d)
        return jax.vmap(hvp_one_point, in_axes=(0, 0))(inputs, keys)
        # (b, k, d)
    else:
        ...
```

Reference Model Interface:
All reference models adhere to a common interface designed to work with the derivative computations in the training pipeline. Each reference model exposes a scalar-valued function that accepts an input vector and returns the corresponding function value. Stochastic reference models may additionally accept a PRNG-key as input to ensure reproducible sampling. It is encouraged, yet not required, that reference models internally apply input/output normalization (mean-centering, unit variance) to stabilize training. In that case, raw inputs are reconstructed before computing the function value $y = f(x_{\text{raw}})$ and returning a normalized y_{norm}. New reference models can be added by implementing the project's abstract `ReferenceModel()` class and providing the required methods for sampling and evaluation.

Reproducibility:
To ensure reproducibility of the presented results, a notebook *examples/rmse_ curves.ipynb* is provided. It produces the presented RMSE-curves by looping over all discussed training methods. Each training configuration is evaluated over multiple PRNG-keys, and error metrics are averaged across runs. The figures showing the loss-weights and the scatter plots for the evaluation of the Bachelier model can be reproduced in the *examples/other_plots.ipynb* notebook.

Next Steps:
The current repository could be cleaned up by removing legacy development notebooks and outdated experiment results, with the goal of having a minimal maintainable codebase. Packaging the project as a proper PyPI module with a comprehensive *README.md* would make it easily installable for other researchers. A dedicated *tests/* directory should be added with automated unit tests to verify correctness (for example ensuring that the returned RandSVD directions are close to the top components of `jax.numpy.linalg.svd(jax.hessian())` for simple functions). Finally, integrating a CI/CD pipeline (via *.github/workflows/ci.yml*) would automate testing for each commit, ensuring that future extensions remain reliable.

List of Figures

List of Definitions

Bibliography

[1] L. Bachelier. Théorie de la spéculation. *Annales scientifiques de l'École Normale Supérieure*, 3e série, 17:21–86, 1900. doi: 10.24033/asens.476. URL https://www.numdam.org/articles/10.24033/asens.476/.

[2] 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 http://github.com/google/jax.

[3] Jaehyuk Choi, Minsuk Kwak, Chyng Wen Tee, and Yumeng Wang. A Black–Scholes user's guide to the Bachelier model. *Journal of Futures Markets*, 42(5):959–980, February 2022. ISSN 1096-9934. doi: 10.1002/fut.22315. URL http://dx.doi.org/10.1002/fut.22315.

[4] Jorio Cocola and Paul Hand. Global Convergence of Sobolev Training for Overparameterized Neural Networks, 2020. URL https://arxiv.org/abs/2006.07928.

[5] Ricardo Crisostomo. An Analysis of the Heston Stochastic Volatility Model: Implementation and Calibration using Matlab, 2015. URL https://arxiv.org/abs/1502.02963.

[6] Wojciech Marian Czarnecki, Simon Osindero, Max Jaderberg, Grzegorz Świrszcz, and Razvan Pascanu. Sobolev Training for Neural Networks, 2017. URL https://arxiv.org/abs/1706.04859.

[7] Yann Dauphin, Razvan Pascanu, Caglar Gulcehre, Kyunghyun Cho, Surya Ganguli, and Yoshua Bengio. Identifying and attacking the saddle point problem in high-dimensional non-convex optimization, 2014. URL https://arxiv.org/abs/1406.2572.

[8] Carl Eckart and Gale Young. The Approximation of One Matrix by Another of Lower Rank. *Psychometrika*, 1(3):211–218, 1936. doi: 10.1007/BF02288367. URL https://doi.org/10.1007/BF02288367.

[9] Andreas Griewank and Andrea Walther. *Evaluating Derivatives*. Society for Industrial and Applied Mathematics, second edition, 2008. doi: 10.1137/1.9780898717761. URL https://epubs.siam.org/doi/abs/10.1137/1.9780898717761.

[10] Nathan Halko, Per-Gunnar Martinsson, and Joel A. Tropp. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions, 2010. URL https://arxiv.org/abs/0909.4061.

[11] Thomas Hartmann, Matthias Kissel, and Klaus Diepold. Sobolev Training With Higher Order Derivatives. URL https://collab.dvb.bayern/download/attachments/75112352/fp_hartmann.pdf?version=1&modificationDate=1570019278123&api=v2.

[12] Steven L. Heston. A Closed-Form Solution for Options with Stochastic Volatility with Applications to Bond and Currency Options. *The Review of Financial Studies*, 6(2):327–343, 04 1993. ISSN 0893-9454. doi: 10.1093/rfs/6.2.327. URL https://doi.org/10.1093/rfs/6.2.327.

[13] Kurt Hornik. Approximation Capabilities of Multilayer Feedforward Networks. *Neural Networks*, 4(2):251–257, 1991. ISSN 0893-6080. doi: https://doi.org/10.1016/0893-6080(91)90009-T. URL https://www.sciencedirect.com/science/article/pii/089360809190009T.

[14] Brian Huge and Antoine Savine. Differential Machine Learning, 2020. URL https://arxiv.org/abs/2005.02347.

[15] M.F. Hutchinson. A stochastic estimator of the trace of the influence matrix for laplacian smoothing splines. *Communication in Statistics- Simulation and Computation*, 18:1059–1076, 01 1989. doi: 10.1080/03610919008812866. URL https://doi.org/10.1080/03610919008812866.

[16] Hao Ji, Wenjian Yu, and Yaohang Li. A Rank Revealing Randomized Singular Value Decomposition (R3SVD) Algorithm for Low-rank Matrix Approximations, 2016. URL https://arxiv.org/abs/1605.08134.

[17] Alex Kendall, Yarin Gal, and Roberto Cipolla. Multi-Task Learning Using Uncertainty to Weigh Losses for Scene Geometry and Semantics, 2018. URL https://arxiv.org/abs/1705.07115.

[18] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima, 2017. URL https://arxiv.org/abs/1609.04836.

[19] N. Kichler. Second-Order Differential Machine Learning. Master's thesis, Aachen, Germany, 2023. URL https://neilkichler.github.io/quantminds/Thesis.pdf. Presented at the Institute of Software and Tools for Computational Engineering. RWTH Aachen University.

[20] Patrick Kidger and Cristian Garcia. Equinox: neural networks in JAX via callable PyTrees and filtered transformations, 2021. URL https://arxiv.org/abs/2111.00254.

[21] Patrick Kidger and Terry Lyons. Universal Approximation with Deep Narrow Networks, 2020. URL https://arxiv.org/abs/1905.08539.

[22] A. O. M. Kilicsoy, J. Liedmann, M. A. Valdebenito, F.-J. Barthold, and M. G. R. Faes. Sobolev Neural Network With Residual Weighting as a Surrogate in Linear and Non-Linear Mechanics. *IEEE Access*, 12:137144–137161, 2024. ISSN 2169-3536. doi: 10.1109/access.2024.3465572. URL http://dx.doi.org/10.1109/ACCESS.2024.3465572.

[23] Alexander Kuleshov. The Various Definitions of Multiple Differentiability of a Function f: $\mathbb{R}^n \rightarrow \mathbb{R}$. *Mathematics*, 8(11), 2020. ISSN 2227-7390. doi: 10.3390/math8111946. URL https://www.mdpi.com/2227-7390/8/11/1946.

[24] James Martens, Ilya Sutskever, and Kevin Swersky. Estimating the Hessian by Backpropagating Curvature, 2012. URL https://arxiv.org/abs/1206.6464.

[25] Per-Gunnar Martinsson and Joel Tropp. Randomized Numerical Linear Algebra: Foundations & Algorithms, 2021. URL https://arxiv.org/abs/2002.01387.

[26] Riley Murray, James Demmel, Michael W. Mahoney, N. Benjamin Erichson, Maksim Melnichenko, Osman Asif Malik, Laura Grigori, Piotr Luszczek, Michał Dereziński, Miles E. Lopes, Tianyu Liang, Hengrui Luo, and Jack Dongarra. Randomized Numerical Linear Algebra : A Perspective on the Field With an Eye to Software, 2023. URL https://arxiv.org/abs/2302.11474.

[27] Uwe Naumann. *The Art of Differentiating Computer Programs: An Introduction to Algorithmic Differentiation*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2012. ISBN 161197206X, 9781611972061.

[28] Erkki Oja. Simplified neuron model as a principal component analyzer. *Journal of Mathematical Biology*, 15(3):267–273, November 1982. ISSN 0303-6812. doi: 10.1007/BF00275687. URL http://dx.doi.org/10.1007/BF00275687.

[29] Thomas O'Leary-Roseberry, Peng Chen, Umberto Villa, and Omar Ghattas. Derivative-Informed Neural Operator: An Efficient Framework for High-Dimensional Parametric Derivative Learning, 2023. URL https://arxiv.org/abs/2206.10745.

[30] Barak A. Pearlmutter. Fast Exact Multiplication by the Hessian. *Neural Computation*, 6(1):147–160, 01 1994. ISSN 0899-7667. doi: 10.1162/neco.1994.6.1.147. URL https://doi.org/10.1162/neco.1994.6.1.147.

[31] Yousef Saad. *Numerical Methods for Large Eigenvalue Problems*. Society for Industrial and Applied Mathematics, 2011. doi: 10.1137/1.9781611970739. URL https://epubs.siam.org/doi/abs/10.1137/1.9781611970739.

[32] Suraj Srinivas and Francois Fleuret. Knowledge Transfer with Jacobian Matching, 2018. URL https://arxiv.org/abs/1803.00443.

[33] S. Surjanovic and D. Bingham. Virtual Library of Simulation Experiments: Test Functions and Datasets. Retrieved December 1, 2025, from http://www.sfu.ca/~ssurjano.

[34] Satoshi Terakado. On the Option Pricing Formula Based on the Bachelier Model. *SSRN Electronic Journal*, 01 2019. doi: 10.2139/ssrn.3428994. URL https://dx.doi.org/10.2139/ssrn.3428994.

[35] David P. Woodruff. Sketching as a Tool for Numerical Linear Algebra. *Found. Trends Theor. Comput. Sci.*, 10:1–157, 2014. URL https://api.semanticscholar.org/CorpusID:51783444.