# Nyström Method for Accurate and Scalable Implicit Differentiation

**Ryuichiro Hataya**[1,2] **Makoto Yamada**[2,3,4]
[1]RIKEN ADSP [2]RIKEN AIP [3]OIST [4]Kyoto University

## Abstract

The essential difficulty of gradient-based bilevel optimization using implicit differentiation is to estimate the inverse Hessian vector product with respect to neural network parameters. This paper proposes to tackle this problem by the Nyström method and the Woodbury matrix identity, exploiting the low-rankness of the Hessian. Compared to existing methods using iterative approximation, such as conjugate gradient and the Neumann series approximation, the proposed method avoids numerical instability and can be efficiently computed in matrix operations without iterations. As a result, the proposed method works stably in various tasks and is faster than iterative approximations. Throughout experiments including large-scale hyperparameter optimization and meta learning, we demonstrate that the Nyström method consistently achieves comparable or even superior performance to other approaches. The source code is available from `https://github.com/moskomule/hypergrad`.

## 1 Introduction

Bilevel optimization is an essential problem in machine learning, which includes hyperparameter optimization (HPO) (Hutter et al., 2019) and meta learning (Hospedales et al., 2021). This problem consists of an inner problem to minimize an inner objective $f(\boldsymbol{\theta}, \boldsymbol{\phi}, \mathcal{T})$ on data $\mathcal{T}$ with respect to parameters $\boldsymbol{\theta} \in \mathbb{R}^p$ and an outer problem to minimize an outer objective $g(\boldsymbol{\theta}, \boldsymbol{\phi}, \mathcal{V})$ on data $\mathcal{V}$ with respect to hyper or meta parameters $\boldsymbol{\phi} \in \mathbb{R}^h$. In the case of HPO, $f$ and $g$ correspond to a training loss function and a validation criterion. In contrast, in the case of meta learning, $f$ and $g$ correspond to meta-training and meta-testing objectives.

Proceedings of the 26th International Conference on Artificial Intelligence and Statistics (AISTATS) 2023, Valencia, Spain. PMLR: Volume 206. Copyright 2023 by the author(s).

Typically in the deep learning literature, the bilevel optimization problem can be formulated as

$$\min_{\boldsymbol{\phi}} g(\boldsymbol{\theta}_T(\boldsymbol{\phi}), \boldsymbol{\phi}, \mathcal{V}) \tag{1}$$

$$\text{s.t.} \quad \boldsymbol{\theta}_t(\boldsymbol{\phi}) = \Theta(\boldsymbol{\theta}_{t-1}(\boldsymbol{\phi}), \nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta}_{t-1}(\boldsymbol{\phi}), \boldsymbol{\phi}, \mathcal{T}), \boldsymbol{\phi}), \tag{2}$$

where $\Theta$ is a gradient-based optimizer, such as SGD and Adam (Kingma and Ba, 2015), and $t = 1, 2, \dots, T$. In some cases, the outer problem (1) can also be optimized by gradient-based optimization methods by using *hypergradient* $\nabla_{\boldsymbol{\phi}} g$, in a similar way to the inner problem, which is expected to be more efficient and scalable than black-box counterparts. Especially when combined with warm-start bilevel optimization that alternately updates outer parameters as Equation (1) and inner parameters as Equation (2) during training (Jaderberg et al., 2017; Vicol et al., 2022), the gradient-based approaches enjoy higher efficiency (Lorraine et al., 2020; Luketina et al., 2016).

A straightforward approach to achieve this goal is to unroll the inner problem to back-propagate through Equation (2) for hypergradient (Domke, 2012; Finn et al., 2017; Grefenstette et al., 2019). However, unrolling increases the memory cost as the number of inner optimization $T$ increases, which may also cause gradient vanishing/explosion (Antoniou et al., 2019). Truncating the backward steps (Shaban et al., 2019) may unburden these issues while sacrificing the quality of hypergradients.

Alternatively, approximating hypergradient using implicit differentiation is promising because it requires much less space complexity than the unrolling approach. Exact implicit differentiation needs computationally demanding inverse Hessian vector product, which has been approximated by iterative methods such as conjugate gradient (Pedregosa, 2016; Rajeswaran et al., 2019) and the Neumann series approximation (Lorraine et al., 2020). Thanks to their space efficiency, these methods can scale to large-scale problems (Hataya et al., 2022; M. Li et al., 2021; Lorraine et al., 2020; Zhang et al., 2021), but such iterative approximations cost time complexities. Furthermore, these methods need careful configuration tuning to avoid numerical instability caused by ill-conditioned Hessian or the norm of Hessian.

In this paper, we propose to use the Nyström method to leverage the low-rank nature of Hessian matrices of neural networks and compute its inverse by the Woodbury matrix identity, inspired by recent works in quasi second-order optimization literature (D. Singh et al., 2021; S. P. Singh and Alistarh, 2020). Unlike the iterative approaches mentioned above, this approximation excludes iterative evaluations and can be computed instantly in matrix operations. Additionally, the proposed method avoids numerical instability. As a result, the Nyström method is robust to configurations, and empirically compares favorably with existing approximation methods consistently on various tasks, from HPO to meta learning. In addition, by using the recurrence of the Woodbury matrix identity, this approach can control the tradeoff between time and space complexities without losing accuracy according to one's computational resource.

In the remaining text, we introduce the proposed method in Section 2. After reviewing related work in Section 3, we analyze the approximation quality of the proposed method when Hessian is low-rank in Section 4. Then, Section 5 empirically demonstrates the effectiveness of the method from a synthetic logistic regression problem to a large-scale real-world data reweighting problem, and finally Section 6 concludes this work.

## 2 Method

### 2.1 Approximating Hypergradient by Implicit Differentiation

In this paper, we focus on the methods to approximate hypergradients $\nabla_{\boldsymbol{\phi}} g$ by implicit differentiation so that the outer problem can also be efficiently optimized by gradient descent. Specifically, if $\nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta}_T, \boldsymbol{\phi}) \approx \mathbf{0}$, then according to the implicit function theorem, we obtain

$$\frac{\mathrm{d}g(\boldsymbol{\theta}_T, \boldsymbol{\phi})}{\mathrm{d}\boldsymbol{\phi}} = -\frac{\partial g}{\partial \boldsymbol{\theta}} \left(\frac{\partial^2 f}{\partial \boldsymbol{\theta}^2}\right)^{-1} \frac{\partial^2 f}{\partial \boldsymbol{\phi} \partial \boldsymbol{\theta}} + \frac{\partial g}{\partial \boldsymbol{\phi}}, \quad (3)$$

where, in the r.h.s., $f = f(\boldsymbol{\theta}_T, \boldsymbol{\phi})$ and $g = g(\boldsymbol{\theta}_T, \boldsymbol{\phi})$. Following the prior works (Lorraine et al., 2020; Pedregosa, 2016; Rajeswaran et al., 2019), we assume that this approximation holds after $T$ iterations of the inner optimization. We also assume that factors in the r.h.s. of Equation (3) are available, *e.g.*, $g$ is differentiable w.r.t. $\boldsymbol{\theta}$. In some cases, such as optimization of hyperparameters for regularization, $\nabla_{\boldsymbol{\phi}} g(\boldsymbol{\theta}_T, \boldsymbol{\phi})$ is always zero.

Still solving Equation (3) seems computationally intractable, as computing inverse Hessian $(\nabla^2_{\boldsymbol{\theta}} f)^{-1}$ is computationally expensive, when the number of model parameters $p = \dim \boldsymbol{\theta}$ is large. Though early works compute inverse Hessian directly (Bengio, 2000; Larsen et al., 1996), especially for modern neural networks, just storing Hessian $\nabla^2_{\boldsymbol{\theta}} f$ is already infeasible in practice.

To mitigate this issue, some approximations have been proposed. Pedregosa, 2016; Rajeswaran et al., 2019 used the conjugate gradient method (Hestenes and Stiefel, 1952), which iteratively solves a linear equation $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ to obtain $\boldsymbol{x} = \boldsymbol{A}^{-1}\boldsymbol{b}$, where, in this case, $\boldsymbol{A} = \nabla^2_{\boldsymbol{\theta}} f$ and $\boldsymbol{b} = \nabla_{\boldsymbol{\theta}} g$. Lorraine et al., 2020 adopted the Neumann series approximation, $\boldsymbol{A}^{-1} = \alpha \sum_{i=1}^{\infty} (\boldsymbol{I} - \alpha \boldsymbol{A})^i$, where $\boldsymbol{A}$ is an invertible matrix that satisfies $\|\alpha A\| \leq 1$ and $\alpha > 0$ is a constant. As these algorithms may take arbitrarily long iterations for convergence, their truncated versions are preferred in practice, which cut off the iterations at a predefined number of steps $l$.

Importantly, these methods do not require keeping actual Hessian but accessing it as Hessian vector product (HVP), which modern automatic differentiation tools (Bradbury et al., 2018; Paszke et al., 2019) can efficiently compute in $O(p)$ (Baydin et al., 2018). Because they consist of HVP and vector arithmetics, their time and space complexities are $O(lp + h)$ and $O(p + h)$ for the number of iterations $l$, where $p = \dim \boldsymbol{\theta}$ and $h = \dim \boldsymbol{\phi}$. In the following discussion, we omit the complexity regarding $h$ for simplicity.

The downside of conjugate gradient and the Neumann series approximation may be their numerical instability. Conjugate gradient needs a well-conditioned matrix for fast convergence (Golub and Van Loan, 2013; Yousef Saad, 2003), *i.e.*, it works sub-optimally with ill-conditioned Hessian. Its longer iterations accumulate numerical errors, and these errors typically need to be alleviated by preconditioning or reorthogonalization, which requires extra time and space complexities. The Neumann series needs the matrix norm to be less than 1, and thus $\alpha$ needs to be carefully configured.

### 2.2 Nyström Approximation

Different from these previous methods using iterative computation, we instead propose to use a low-rank approximation for approximated inverse Hessian vector product (IHVP). Specifically, we propose to use the Nyström method to obtain IHVP by leveraging the low-rank nature of Hessian matrix (Ghorbani et al., 2019; Karakida et al., 2019; LeCun et al., 2012).

We use the following $k$-rank approximation to the original $p$ dimensional Hessian matrix, where we assume $k \ll p$:

$$\boldsymbol{H}_k = \boldsymbol{H}_{[:,K]} \boldsymbol{H}^{\dagger}_{[K,K]} \boldsymbol{H}^{\top}_{[:,K]}, \quad (4)$$

where $K$ is a randomly selected index set of size $k$, $\boldsymbol{H}_{[:,K]} \in \mathbb{R}^{p \times k}$ is a matrix extracted columns of $\boldsymbol{H}$ corresponding to indices in $K$, and $\boldsymbol{H}_{[K,K]} \in \mathbb{R}^{k \times k}$ is a matrix extracted rows of $\boldsymbol{H}_{[:,K]}$ corresponding to indices in $K$. $\boldsymbol{H}^{\dagger}_{[K,K]} = \boldsymbol{U}\boldsymbol{\Lambda}^{-1}\boldsymbol{U}^{\top}$ denotes the pseudo inverse, where $\boldsymbol{U}$ and $\boldsymbol{\Lambda}$ are eigenvectors and engenvalues of $\boldsymbol{H}_{[K,K]}$.

Then, we use the Woodbury matrix identity for matrices

$\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}$ such that

$$\begin{aligned}&(\boldsymbol{A}+\boldsymbol{C}\boldsymbol{B}\boldsymbol{C}^\top)^{-1} \\ =&\boldsymbol{A}^{-1}-\boldsymbol{A}^{-1}\boldsymbol{C}(\boldsymbol{B}^{-1}+\boldsymbol{C}^\top\boldsymbol{A}^{-1}\boldsymbol{C})^{-1}\boldsymbol{C}^\top\boldsymbol{A}^{-1}\end{aligned} \tag{5}$$

to obtain the inverse Hessian. Namely, we compute $(\boldsymbol{H}_k+\rho\boldsymbol{I}_p)^{-1}$, where $\rho>0$ is a small constant to improve numerical stability and $\boldsymbol{I}_p$ is the $p$-dimensional identity matrix, to approximate $\boldsymbol{H}^{-1}$ as follows:

$$\begin{aligned}&(\rho\boldsymbol{I}_p+\boldsymbol{H}_{[:,K]}\boldsymbol{H}^\dagger_{[K,K]}\boldsymbol{H}^\top_{[:,K]})^{-1} \\ =&\frac{1}{\rho}\boldsymbol{I}_p-\frac{1}{\rho^2}\boldsymbol{H}_{[:,K]}\left(\boldsymbol{H}_{[K,K]}+\frac{1}{\rho}\boldsymbol{H}^\top_{[:,K]}\boldsymbol{H}_{[:,K]}\right)^{-1}\boldsymbol{H}^\top_{[:,K]}.\end{aligned} \tag{6}$$

Although this left-hand side requires the inversion of a $p\times p$ matrix, the right-hand side only needs the inversion of a $k\times k$ matrix. Because $k\ll p$, the computational burden of the l.h.s. is drastically reduced in the r.h.s. The use of Woodbury matrix identity as Equation (6) is similar to the idea of D. Singh et al., 2021, but our formulation is slightly efficient as it avoids unnecessary eigen decomposition.

The small constant $\rho$ in Equation (6) makes a low-rank matrix $\boldsymbol{H}_k$ invertible. Additionally, it can also be regarded as being stemmed from a proximally regularized inner objective $f(\boldsymbol{\theta},\boldsymbol{\phi})+\frac{\rho}{2}\|\boldsymbol{\theta}-\boldsymbol{\theta}'\|$, where $\boldsymbol{\theta}'\in\operatorname*{argmin}_{\boldsymbol{\theta}} f(\boldsymbol{\theta},\boldsymbol{\phi})$ (Vicol et al., 2022).

We visualize the inverse of a low-rank matrix and its approximations in Figure 1. Nyström method can approximate the true inverse efficiently and accurately. Because conjugate gradient cannot explicitly output inverse Hessian, we do not display its result here.

To sum up, the proposed method approximates the hypergradient by using a low-rank Hessian $\boldsymbol{H}_k$ as

$$\frac{\mathrm{d}g(\boldsymbol{\theta}_T,\boldsymbol{\phi})}{\mathrm{d}\boldsymbol{\phi}}\approx-\frac{\partial g}{\partial\boldsymbol{\theta}}(\boldsymbol{H}_k+\rho\boldsymbol{I}_p)^{-1}\frac{\partial^2 f}{\partial\boldsymbol{\phi}\partial\boldsymbol{\theta}}+\frac{\partial g}{\partial\boldsymbol{\phi}}. \tag{7}$$

## 2.3 Space-efficient Variant

The Nyström approximation is free from iterative algorithms, but it needs to store $k$ columns of the original Hessian matrix $\boldsymbol{H}_{[:,K]}$ and compute the inverse of a $k\times k$ matrix. As a result, its time and space complexities are $O(p+k^3)$ and $O(kp+k^2)$, but $k^3$ and $k^2$ are ignorable because usually $k\ll p$.

Some readers may worry about memory explosion when $k$ is relatively large. Actually, this Nyström approximation can be turned into an iterative algorithm that saves memory. Recall that the low-rank matrix can be decomposed as follows

$$\boldsymbol{H}_k=\boldsymbol{H}_{[:,K]}\boldsymbol{H}^\dagger_{[K,K]}\boldsymbol{H}^\top_{[:,K]}=\sum_{i\in K}\frac{1}{\lambda_i}\boldsymbol{l}_i\boldsymbol{l}_i^\top, \tag{8}$$

**Algorithm 1** Algorithm of the proposed method

```
Require:
  $\kappa$: control parameter of computational cost
  $K$: randomly selected index set, where $\#K = k$
  $\boldsymbol{H}_{[:,K]}$: a column matrix of $\boldsymbol{H}$ corresponding to $K$
  $\boldsymbol{U}, \boldsymbol{\Lambda}$: eigen decomposition of $\boldsymbol{H}_{[K,K]} = \boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^\top$

  Partition $K$ in to size $\kappa$ subsets $\mathcal{K}$
  $\hat{\boldsymbol{H}} = \frac{1}{\rho}\boldsymbol{I}_p$
  for $K'$ in $\mathcal{K}$ do
      $\boldsymbol{L} \leftarrow (\boldsymbol{H}_{[:,K]}\boldsymbol{U})_{[:,K']}$
      $\boldsymbol{J} \leftarrow \boldsymbol{\Lambda}_{[K',K']}$
      $\hat{\boldsymbol{H}} \leftarrow \hat{\boldsymbol{H}} - \hat{\boldsymbol{H}}\boldsymbol{L}(\boldsymbol{J}+\boldsymbol{L}^\top\hat{\boldsymbol{H}}\boldsymbol{L})^{-1}\boldsymbol{L}^\top\hat{\boldsymbol{H}}$
  end for
  Return $\hat{\boldsymbol{H}}$, equivalent to $(\boldsymbol{H}_k+\rho\boldsymbol{I}_p)^{-1}$
```

where $\lambda_i\in\mathbb{R}$ is the $i$th value of $\boldsymbol{\Lambda}$ and $\boldsymbol{l}_i=(\boldsymbol{H}_{[:,K]}\boldsymbol{U})_{[:,i]}\in\mathbb{R}^p$. Then, we can iteratively compute the inverse of $\boldsymbol{H}_k+\rho\boldsymbol{I}_p$ by the Woodbury matrix identity (Equation (5)) as

$$\begin{aligned}\hat{\boldsymbol{H}}_{i+1}&=\hat{\boldsymbol{H}}_i-\frac{\hat{\boldsymbol{H}}_i\boldsymbol{l}_i\boldsymbol{l}_i^\top\hat{\boldsymbol{H}}_i}{\lambda_i+\boldsymbol{l}_i^\top\hat{\boldsymbol{H}}_i\boldsymbol{l}_i}, \\ \text{where}\quad \hat{\boldsymbol{H}}_0&=\frac{1}{\rho}\boldsymbol{I}_p, \\ \hat{\boldsymbol{H}}_k&=(\boldsymbol{H}_k+\rho\boldsymbol{I}_p)^{-1},\end{aligned} \tag{9}$$

for $i=0,1,\dots,k-1$. This variant needs $O(k^2p)$ time complexity and $O(p)$ space complexity like iterative algorithms. S. P. Singh and Alistarh, 2020 proposed a dynamical algorithm for a similar problem to compute the inverse of the Fisher information matrix (FIM).

## 2.4 Controlling the Cost Tradeoff

Furthermore, by chunking $\boldsymbol{H}_{[:,K]}$ into thinner matrices of width $\kappa\in(1,k)$ and applying the Woodbury matrix identity iteratively as Algorithm 1, $(\boldsymbol{H}_k+\rho\boldsymbol{I}_p)^{-1}$ can be obtained in a hybrid manner with less memory footprint, *i.e.*, $O(\kappa p)$, than Equation (6) and faster, *i.e.*, $O(\{k/\kappa\}^2p)$, than Equation (9). In other words, our method allows users to dynamically control the necessary tradeoff between time and space complexities for given accuracy, which is a unique characteristic of our proposed method. See Table 1 for comparison with other methods.

Notice that for any $\kappa$, the computational result is equivalent to each other up to machine precision. Thus, in the remaining paper, we use Equation (6), where $\kappa=k$, without otherwise specified.

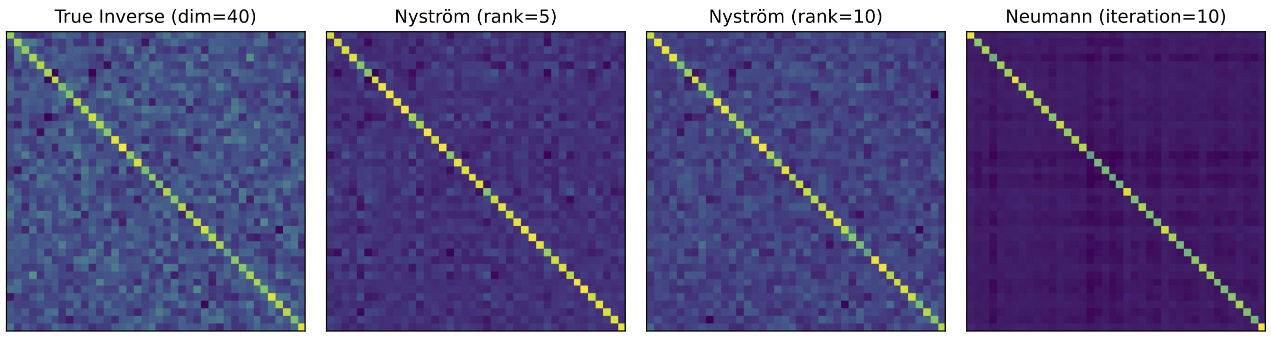

Figure 1: Comparison of inverse of a 40 dimensional matrix $\boldsymbol{A} + \rho \boldsymbol{I}$. $\boldsymbol{A}$ is a rank 20 symmetric matrix, and $\rho$ is set to 0.1. Nyström method can approximate the true inverse accurately even in the rank 5 setting.

Table 1: Comparison of time and space complexity. $p$ denotes the number of model parameters. $l$ is the number of iterations of the algorithms, $k$ is the rank of low-rank Hessian. The Nyström method allows users to control the complexities by choosing $\kappa \in \{1, 2, \dots, k\}$.

| Approximation | Time Complexity | Space Complexity |
|---|---|---|
| Conjugate gradient (Rajeswaran et al., 2019) | $O(lp)$ | $O(p)$ |
| Neumann series (Lorraine et al., 2020) | $O(lp)$ | $O(p)$ |
| Nyström method (ours) | $O((k/\kappa)^2 p)$ | $O(\kappa p)$ |

### 2.5 Limitations

The proposed method cannot straightforwardly optimize outer parameters $\boldsymbol{\phi}$ that do not directly affect the training loss, inheriting the limitation of the methods to approximate hypergradient by implicit differentiation (Lorraine et al., 2020). Such parameters include a learning rate of an optimizer of the inner problem, which needs to be carefully tuned in deep learning research (Schmidt et al., 2021). We may need to rely on unrolling approaches for this problem (Andrychowicz et al., 2016; Grefenstette et al., 2019; K. Li and Malik, 2017). Additionally, the method does not directly applicable to non-smooth problems as other gradient-based methods, part of which could be alleviated by smoothing the problem or using sub-gradients and sub-Hessians.

## 3 Related Work

### 3.1 Gradient-based Hyperparameter Optimization and Meta Learning

The development of automatic differentiation (Baydin et al., 2018; Bradbury et al., 2018; Paszke et al., 2019) has encouraged the active research of gradient-based HPO and meta learning, where the outer problem of a bilevel problem (Equation (1)) is also optimized by gradient descent using hypergradient (Franceschi, 2021; Franceschi, Frasconi, et al., 2018).

One way to compute hypergradients is to backpropagate through the unrolled inner optimization (Equation (2)) (Domke, 2012; Finn et al., 2017; Grefenstette et al., 2019). Except for the special cases, where a specific inner optimization algorithm (Maclaurin et al., 2015) or forward-mode automatic differentiation (Franceschi, Donini, et al., 2017) can be used, this approach suffers from space complexity as the inner optimization step $T$ increases.

Another approach is to approximate hypergradient using the implicit differentiation as Equation (3) with less space complexity (Bengio, 2000; Lorraine et al., 2020; Pedregosa, 2016; Rajeswaran et al., 2019). Although Equation (3) includes inverse Hessian, which is infeasible to compute for modern neural networks, truncated solutions of conjugate gradient (Pedregosa, 2016; Rajeswaran et al., 2019) and the Neumann series approximation (Lorraine et al., 2020) have been adopted to approximate this term efficiently. Other solvers for linear systems, such as GMRES (Youcef Saad and Schultz, 1986), can also be used (Blondel et al., 2021). Our work is in line with these works, but compared to these methods using generic techniques for matrix inverse, the proposed method exploits the low-rankness of Hessian of neural networks.

### 3.2 Inverse Hessian Approximation

The application of inverse Hessian approximation is not limited to the computation of hypergradient. It has been a key element in estimation of influence function (Koh and Liang, 2017), backpropagation through long recurrence (Liao et al., 2018), and network pruning (Hassibi and Stork, 1992; S. P. Singh and Alistarh, 2020).

The inverse of Hessian or an FIM is also indispensable in (quasi) second-order optimization (Martens, 2010) and natural gradient descent (Amari, 1998). Thus, its estimation has been studied for a long time. For example, LBFGS employs past gradients and updates (Liu and Nocedal, 1989), and KFAC adopts block-diagonal approximation of FIM (Martens and Grosse, 2015) for efficient approximation of large matrix inversion.

The Hessians and FIMs of neural networks have low-rank structures, as most of their eigenvalues are nearly zero (Ghorbani et al., 2019; Karakida et al., 2019; LeCun et al., 2012). Exploiting this nature, inverse FIM (Frantar et al., 2021) or inverse Hessian (D. Singh et al., 2021) are computed using the Woodbury identity in the literature of quasi second-order optimization. Especially, the latter used the Nyström method. Although these approaches are technically similar to ours, they are in a different context.

## 4 Theoretical Analysis

This section theoretically shows that the proposed approach can efficiently approximate the true hypergradient.

**Theorem 1** *Suppose $\boldsymbol{H}$ is a positive semidefinite. Let $\boldsymbol{h}^\star$ and $\boldsymbol{h}$ be hypergradients using the true inverse Hessian $(\boldsymbol{H}+\rho\boldsymbol{I}_p)^{-1}$ (r.h.s. of Equation (3)) and the Nyström method $(\boldsymbol{H}_k+\rho\boldsymbol{I}_p)^{-1}$ (r.h.s. of Equation (6)), and $\boldsymbol{g}=\nabla_{\boldsymbol{\theta}} g(\boldsymbol{\theta}_T,\boldsymbol{\phi})$, $\boldsymbol{F}=\nabla_{\boldsymbol{\phi}}\nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta}_T,\boldsymbol{\phi})$. Then, the accuracy of approximated hypergradient is bounded*

$$\|\boldsymbol{h}^\star-\boldsymbol{h}\|_2 \le \|\boldsymbol{g}\|_2\|\boldsymbol{F}\|_{\mathrm{op}}\left(\frac{1}{\rho}\frac{\|\boldsymbol{H}-\boldsymbol{H}_k\|_{\mathrm{op}}}{\rho+\|\boldsymbol{H}-\boldsymbol{H}_k\|_{\mathrm{op}}}\right). \quad (10)$$

$\|\cdot\|_2$ and $\|\cdot\|_{\mathrm{op}}$ denote L2 norm and operator norm, respectively.

This theorem is based on (Frangella et al., 2021). See the supplemental material for the derivation.

When considering neural networks, because the training objective $f$ is not convex w.r.t. $\boldsymbol{\theta}$, its Hessian $\boldsymbol{H}$ is not always positive semi-definite. However, Ghorbani et al., 2019 empirically demonstrated that most negative eigenvalues disappeared even after a few iterations of training, indicating that we may assume that $\boldsymbol{H}+\rho\boldsymbol{I}$ for $\rho>0$ is positive semi-definite in practice. Also importantly, $\|\boldsymbol{H}-\boldsymbol{H}_k\|_{\mathrm{op}}$ is bounded.

**Remark 1 (Theorem 3 in Drineas and Mahoney, 2005)** *Let $\bar{\boldsymbol{H}}_k$ be the best $k$-rank approximation of $\boldsymbol{H}$. If $O(k/\epsilon^4)$ columns are selected for $\epsilon>0$ so that the $i$th column is chosen proportional to $H_{i,i}^2$, then,*

$$\mathbb{E}[\|\boldsymbol{H}-\boldsymbol{H}_k\|_{\mathrm{op}}] \le \|\boldsymbol{H}-\bar{\boldsymbol{H}}_k\|_{\mathrm{op}}+\epsilon\sum_{i=1}^{p} H_{i,i}^2. \quad (11)$$

*Especially if $\boldsymbol{H}$ is a rank $k$ matrix, then*

$$\mathbb{E}[\|\boldsymbol{H}-\boldsymbol{H}_k\|_{\mathrm{op}}] \le \epsilon\sum_{i=1}^{p} H_{i,i}^2 \quad (12)$$

Because Hessian of a trained neural network can be regarded as low rank (Ghorbani et al., 2019; Karakida et al., 2019; LeCun et al., 2012), we may expect that Equation (12) holds.

Equation (10) indicates that an approximated hypergradient converges to the true hypergradient as $\boldsymbol{H}_k$ approaches to $\boldsymbol{H}$. This differs from truncated iterative approximations, such as conjugate gradient, where their expected solutions never converge to the true one for a small number of iterations.

## 5 Experiments

In this section, we empirically demonstrate the effectiveness of the proposed method.

### Experimental Setups

We implemented models and algorithms using `PyTorch` v1.12 (Paszke et al., 2019) and its accompanying `functorch` (He and Zou, 2021). The reference code is available from `https://github.com/moskomule/hypergrad`. Experiments were conducted on a single NVIDIA A100 GPU with CUDA 11.3. The implementations of conjugate gradient and the Neumann series approximation algorithms were adopted from `betty` v0.1.1 (Choe et al., 2022).

In the following experiments, the Nyström method was implemented according to Equation (6), that is, the time-efficient variant otherwise spcified. Using ReLU as an activation function leads some columns of Hessian to zero vectors, and then the inversion in Equation (6) fails. To circumvent this problem, we replaced ReLU with leaky ReLU: $\mathrm{LR}(x)=\max(0,x)+0.01\times\min(0,x)$.

### 5.1 Optimizing Weight-decay of Linear Regression

We first showcase the ability of the Nyström approximation by optimizing weight-decay parameters for each parameter of a linear regression model using synthetic data. For $D$ dimensional data, inner parameters $\boldsymbol{\theta}\in\mathbb{R}^D$ and outer parameters $\boldsymbol{\phi}\in\mathbb{R}^D$ are optimized. The inner problem is $f(\boldsymbol{\theta},\boldsymbol{\phi})=\ell(\boldsymbol{\theta}^\top\boldsymbol{x},y)+\boldsymbol{\theta}^\top\operatorname{diag}(\boldsymbol{\phi})\boldsymbol{\theta}$, for an input $\boldsymbol{x}$ and its label $y$, where $\ell$ is binary cross entropy loss. Each input $\boldsymbol{x}$ is sampled from a standard normal distribution, and its label is defined as $y=\boldsymbol{w}^{*\top}\boldsymbol{x}+\boldsymbol{\epsilon}>0$, where $\boldsymbol{w}^*\in\mathbb{R}^D$ is a constant vector and $\boldsymbol{\epsilon}\in\mathbb{R}^D$ is a noise vector. This inner problem is optimized by SGD with a learning rate of 0.1, and the inner parameters are reset every 100 iteration. The outer problem is to minimize validation loss by SGD with a learning rate of 1.0 and a momentum of 0.9. The outer parameters $\phi$, initialized to $\mathbf{1}$, are updated after every 100 inner parameter update. We set $D = 100$ and used 500 data points for both inner and outer optimization.

Figure 2 (top) shows validation loss curves, comparing approximated implicit differentiation methods, conjugate gradient, and the Neumann series approximation, with our proposed method. For the conjugate gradient method and the Neumann series approximation, we set the number of iterations $l$ to 5, following Rajeswaran et al., 2019. Accordingly, we set the rank of the Nyström method to 5. As can be seen, the Nyström method can optimize the weight-decay parameters faster than other methods. Figure 2 (bottom) displays training loss curves. Because the inner parameters are reset when the outer parameters are updated, training loss values at inner-parameter reset moments are high (around 0.7). As the outer optimization proceeds, the inner parameters, particularly those of the Nyström method, quickly decreases the training loss during each inner optimization period.

For the experiments in Figure 2, the "learning rate" parameter $\alpha$ of conjugate gradient and the Neumann series approximation was set to 0.01, and $\rho$ of the Nyström method was set to 0.01. We compare other choices of $\alpha$ in $\{0.01, 0.1, 1.0\}$ in Figure 3. Accordingly, we try other values of $\rho$ in $\{0.01, 0.1, 1.0\}$. The results indicate that the Nyström method surpasses others in most cases and show robustness to the choice of $\rho$. We will revisit the robustness of the Nyström method later in Section 5.4. These experimental results were averaged over five runs of different random seeds.

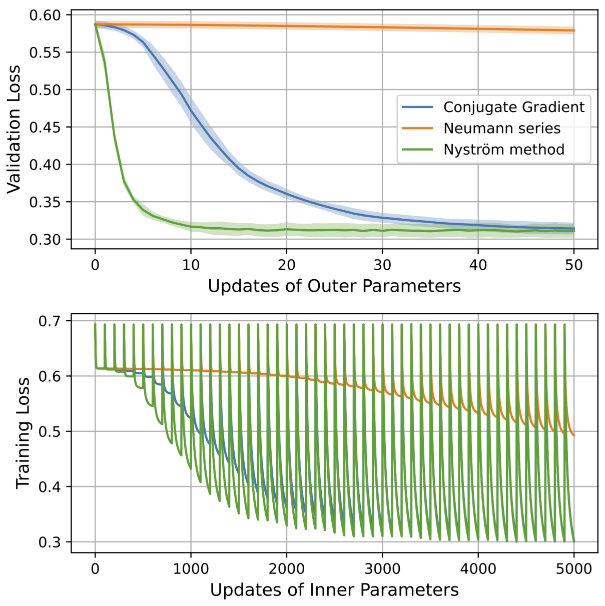

Figure 2: Optimization of weight decay parameters to each model parameter in logistic regression. The top figure shows the *validation* loss curve of the outer problem, and the bottom figure shows the *training* loss curves of the inner problem, optimized in 100 iterations.

Table 2: Test accuracy of the dataset-distillation task on the MNIST dataset. The Nyström method shows better performance than others.

| Conjugate gradient | Neumann series | Nyström method |
|---|---|---|
| $0.17 \pm 0.04$ | $0.47 \pm 0.03$ | $0.49 \pm 0.04$ |

## 5.2 Dataset Distillation

Dataset distillation is a task to optimize a small synthesized training dataset $\mathcal{T}_\phi = \{\phi_1, \phi_2, \ldots, \phi_C\}$ parametrerized by $\phi$ so that validation loss on real data is minimized (Wang et al., 2018). We used MNIST dataset (Le Cun et al., 1998) and a LeNet-like CNN, and followed the fixed-known initialization setting that CNN weights, *i.e.*, the inner parameters, are reset every 100 model parameter update. As MNIST is a 10-class dataset, we set $C = 50$, so each class has 5 distilled images. Each $\phi_i \in \mathcal{T}_\phi$ has an equal size to an MNIST image. We used fixed learning rates for inner and outer optimization to simplify the problem. Namely, the inner problem is optimized by SGD with a learning rate of 0.01, while the outer problem is optimized with an Adam optimizer with a learning rate of $1.0 \times 10^{-3}$.

The test accuracy after 5,000 outer parameter updates is reported in Table 2. These results were averaged over five runs. We set $\alpha = \rho = 0.01$ and $l = k = 10$.

The Nyström method yields comparable performance to the Neumann series approximation. However, despite our best efforts to select appropriate values of $\alpha \in \{0.01, 0.1, 1.0\}$ and $l \in \{5, 10, 20\}$ based on the validation performance on a 10% split of training data, the conjugate gradient method failed to learn this task. This failure may be attributed to ill-conditioned Hessian.

## 5.3 Gradient-based Meta Learning

MAML is a typical method of gradient-based meta learning, where the inner problem learns to adapt to a given problem while the outer problem aims to find good parameters that adapts quickly to new tasks (Finn et al., 2017). Among its variants, iMAML uses implicit differentiation to compute hypergradient, achieving better memory efficiency (Rajeswaran et al., 2019). Although the original iMAML adopts conjugate gradient to obtain IHVP, this choice can be replaced with the Neumann series approximation and the Nyström method.

We compared such backends using few-shot image classification, where models are learned to classify images only

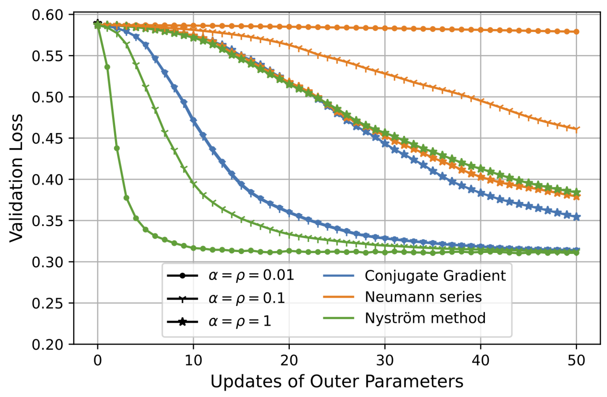

Figure 3: Validation loss curves of implicit differentiation methods with different configurations.

Table 3: Test accuracy of the meta learning task on the Omniglot dataset. The Nyström method shows comparable performance to conjugate gradient.

| Task | 1-shot | 5-shot |
|---|---|---|
| Conjugate gradient | $0.96 \pm 0.00$ | $0.98 \pm 0.00$ |
| Neumann series | $0.91 \pm 0.00$ | $0.97 \pm 0.00$ |
| Nyström method | $0.95 \pm 0.00$ | $0.98 \pm 0.00$ |

from few examples (Fei-Fei et al., 2006; B. Lake et al., 2011; Ravi and Larochelle, 2017), on the Omniglot dataset (B. M. Lake et al., 2015) with a VGG-like CNN, following (Antoniou et al., 2019; Rajeswaran et al., 2019). We set $k = l = 10$, $\alpha = \rho = 0.01$. The inner problem is to optimize model parameters by SGD with a learning rate of 0.1 in 10 steps, and the outer problem is to update the initial model parameters by Adam with a learning rate of $1.0 \times 10^{-3}$.

Table 3 shows the averaged accuracy on the test tasks over three runs after training on $1.6 \times 10^6$ tasks. As can be seen, the Nyström method achieved comparable results with iMAML using conjugate gradient both in the 1-shot and 5-shot settings.

## 5.4 Data Reweighting

Data reweighting is a task to learn to weight a loss value to each example, which aims to alleviate the effect of class imbalance and label noise (M. Li et al., 2021; Shu et al., 2019). Its inner problem can be formulated as $f(\boldsymbol{\theta}, \boldsymbol{\phi}) = \ell(\nu_{\boldsymbol{\theta}}(\boldsymbol{x}), \boldsymbol{y}) \cdot \mu_{\boldsymbol{\phi}}(\ell(\nu_{\boldsymbol{\theta}}(\boldsymbol{x}), \boldsymbol{y}))$, where $\ell$ is cross-entropy loss, $\nu_{\boldsymbol{\theta}}$ is a model, and $\mu_{\boldsymbol{\phi}}$ is a neural network to weight samples. The outer problem is to update $\boldsymbol{\phi}$ to minimize validation loss on balanced validation data. The inner parameters are not reset when the outer parameters are updated.

We adopted long-tailed CIFAR-10 datasets (Cui et al., 2019), which simulate class imbalance at several degrees, WideResNet 28-10 (Zagoruyko and Komodakis, 2016) as $\nu_{\boldsymbol{\theta}}$, which has approximately $3.6 \times 10^7$ parameters, and a two-layer MLP with a hidden dimension of 100 as $\mu_{\boldsymbol{\phi}}$. The inner problem is optimized by SGD with a learning date of 0.1, momentum of 0.9, and weight decay of $5.0 \times 10^{-4}$, and the outer problem is optimized with an Adam optimizer with a learning rate of $1.0 \times 10^{-5}$ on 2% split of training data, following Shu et al., 2019. We set $\alpha = \rho = 0.01$ and $l = k = 10$.

Table 4 shows the averaged test accuracy over three runs after $1.5 \times 10^4$ inner updates and $1.5 \times 10^3$ outer updates. Again, the Nyström method consistently yielded matching or better performance to other methods and outperformed the baseline.

### Runtime Speed and Memory Consumption

Table 5 compares speed and peak GPU memory consumption to compute hypergradients on the data reweighting task (averaged over 10 runs). Because WideResNet 28-10 caused out of memory when $k = 20$ with the time-efficient Nyström method, we instead used relatively smaller WideResNet 28-2, which has $1.5 \times 10^6$ parameters. The reported values were measured after 10 iterations of warmup.

As shown in Table 1, the time complexity of iterative algorithms, conjugate gradient and the Neumann series, depends on $l$, whereas that of the Nyström method is independent of $k$. As a result, the runtime speed of iterative algorithms slowdowns as the approximation quality $l$ increases, while the deceleration of the time-efficient Nyström method is marginal. On the other hand, the space complexity of the iterative algorithms is constant of $l$, which is reflected in the results. In contrast, that of the time-efficient Nyström method relies on $k$, which can also be observed from the linear growth of the actual memory consumption.

Table 5 also presents the results of the space-efficient variant of the Nyström method, where $\kappa = 1$. Its memory consumption is constant, while the speed is almost quadratic to $k$, demonstrating the controllability of the tradeoff between speed and memory consumption as expected in Table 1.

### Robustness of the Nyström method

The Nyström method has two parameters $\rho$ for numerical stability and $k$ for the matrix rank. Table 6 shows the effect of these configurations on the data reweighting task using WideResNet 28-2 on the long-tailed CIFAR-10 of the imbalanced factor of 50 over $\rho \in \{0.01, 0.1, 1.0\}$ and $k \in \{5, 10, 20\}$. The results differ only marginally, *i.e.*, the proposed method is robust to the choice of configurations, which is a favorable property in practical applications. Figure 4 compares the validation curves in weight-decay opti-

Table 4: Test accuracy of the data reweighting task on the long-tailed CIFAR-10 datasets. Baseline indicates training without outer optimization. The Nyström method achieves consistently favorable results.

| Imbalanced factor | 200 | 100 | 50 |
|---|---|---|---|
| Baseline | $0.62 \pm 0.06$ | $0.67 \pm 0.13$ | $0.74 \pm 0.08$ |
| Conjugate gradient | $0.63 \pm 0.06$ | $0.70 \pm 0.05$ | $0.78 \pm 0.02$ |
| Neumann series | $0.60 \pm 0.09$ | $0.73 \pm 0.01$ | $0.79 \pm 0.01$ |
| Nyström method | $0.66 \pm 0.02$ | $0.73 \pm 0.02$ | $0.79 \pm 0.01$ |

Table 5: Average runtime speed and peak memory consumption for hypergradient computation in data reweighting task over 10 runs.

| | | Speed (s) | Peak GPU Memory Consumption (GB) |
|---|---|---|---|
| Conjugate gradient (Pedregosa, 2016) | $l = 5$ | 0.44 | 2.46 |
| | $l = 10$ | 0.83 | 2.46 |
| | $l = 20$ | 1.68 | 2.46 |
| Neumann series (Lorraine et al., 2020) | $l = 5$ | 0.40 | 2.39 |
| | $l = 10$ | 0.75 | 2.39 |
| | $l = 20$ | 1.48 | 2.39 |
| Nyström method (ours) (time efficient) | $k = 5$ | 0.24 | 4.66 |
| | $k = 10$ | 0.33 | 8.15 |
| | $k = 20$ | 0.54 | 15.1 |
| Nyström method (ours) (space efficient) | $k = 5$ | 3.11 | 1.94 |
| | $k = 10$ | 10.7 | 1.94 |
| | $k = 20$ | 41.0 | 1.94 |

mization of logistic regression using the Nyström method with different $k$. Again, the differences of curves among configurations are marginal, emphasizing the robustness of the Nyström method.

These results suggest that $k = 5$ may be sufficient for practically sized problems, which is faster than other methods, while consuming only twice memory (Table 5). Also notice that, throughout various experiments including HPO and meta learning, the proposed method successfully and consistently works, different from other methods that failed at some tasks. This indicates that the Nyström method may also be robust to the types of problems. These properties are appealing for practical use cases, that is, the Nyström method may need minimum efforts for "*hyper*-hyperparameter optimization."

Table 6: The effect of $\rho$ and $k$ of the Nyström method on the data reweighting task. Test accuracy is reported. The baseline without outer optimization yields test accuracy of $0.75 \pm 0.03$. These results indicate the robustness of the proposed method to configurations.

| | | $\rho$ | | |
|---|---|---|---|---|
| | | 0.01 | 0.1 | 1.0 |
| $k$ | 5 | $0.79 \pm 0.01$ | $0.78 \pm 0.01$ | $0.79 \pm 0.01$ |
| | 10 | $0.79 \pm 0.01$ | $0.78 \pm 0.01$ | $0.78 \pm 0.01$ |
| | 20 | $0.78 \pm 0.02$ | $0.78 \pm 0.01$ | $0.79 \pm 0.01$ |

## 6 Conclusion and Discussion

This paper introduced an approximated implicit differentiation method for gradient-based bilevel optimization using the Nyström method. The key idea was to exploit the low-rank property of Hessian of neural networks by the Nyström method and use the Woodbury matrix identity for fast and accurate computation of inverse Hessian vector product in hypergradient. The proposed method scaled to large-scale problems and was applicable to hyperparameter optimization and meta learning. Empirically, the approach was robust to configurations and about two times faster than iterative approximation methods.

Although hyperparameter optimization is crucial in machine learning, especially in deep learning, traditional hyperparameter optimization is costly and emits a substantial amount of CO2 (Strubell et al., 2020). Contrarily, gradient-based hyperparameter optimization is efficient and may help alleviate this issue. Since the proposed method is fast, scalable, robust, and applicable to a wide range of tasks, it may provide a reliable way for researchers and practitioners to introduce efficient bilevel optimization.

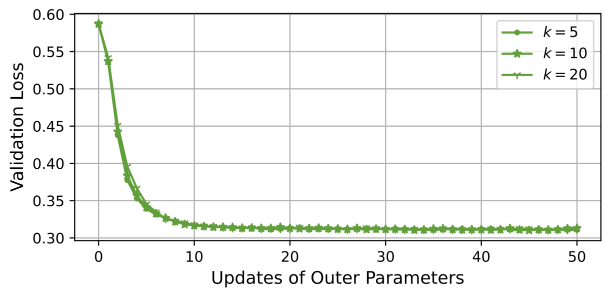

Figure 4: The effect of $k$ when $\rho = 0.01$ in weight-decay optimization of logistic regression.

## Acknowledgments

We thank anonymous reviewers for constructive comments to improve the manuscript. R.H. was supported by JST, ACT-X Grant Number JPMJAX210H, and M.Y. was supported by MEXT KAKENHI Grant Number 20H04243. We used computational resources of "mdx: a platform for the data-driven future."

## References

Amari, Shunichi (1998). "Natural Gradient Works Efficiently in Learning". In: *Neural Computation* 10.2, pp. 251–276.

Andrychowicz, Marcin, Misha Denil, Sergio Gómez, Matthew W Hoffman, David Pfau, Tom Schaul, Brendan Shillingford, and Nando de Freitas (2016). "Learning to learn by gradient descent by gradient descent". In: *NIPS*.

Antoniou, Antreas, Harrison Edwards, and Amos Storkey (2019). "How to train your MAML". In: *ICLR*.

Baydin, Atilim Gunes, Barak A. Pearlmutter, Alexey Andreyevich Radul, and Jeffrey Mark Siskind (2018). "Automatic Differentiation in Machine Learning: a Survey". In: *JMLR* 18.153, pp. 1–43.

Bengio, Yoshua (2000). "Gradient-based optimization of hyperparameters". In: *Neural Computation* 12.8, pp. 1889–1900.

Blondel, Mathieu, Quentin Berthet, Marco Cuturi, Roy Frostig, Stephan Hoyer, Felipe Llinares-López, Fabian Pedregosa, and Jean-Philippe Vert (2021). "Efficient and Modular Implicit Differentiation". In: *arXiv*.

Bradbury, James, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang (2018). *JAX: composable transformations of Python+NumPy programs*. Version 0.2.5. URL: `http://github.com/google/jax`.

Choe, Sang Keun, Willie Neiswanger, Pengtao Xie, and Eric Xing (2022). "Betty: An Automatic Differentiation Library for Multilevel Optimization". In: *arXiv*.

Cui, Yin, Menglin Jia, Tsung-Yi Lin, Yang Song, and Serge Belongie (2019). "Class-balanced loss based on effective number of samples". In: *CVPR*.

Domke, Justin (2012). "Generic methods for optimization-based modeling". In: *JMLR* 22, pp. 318–326.

Drineas, Petros and Michael W. Mahoney (2005). "On the Nystrom Method for Approximating a Gram Matrix for Improved Kernel-Based Learning". In: *Journal of Machine Learning Research* 6.72, pp. 2153–2175.

Fei-Fei, Li, Robert Fergus, and Pietro Perona (2006). "One-shot learning of object categories". In: *TPAMI* 28.4, pp. 594–611.

Finn, Chelsea, Pieter Abbeel, and Sergey Levine (2017). "Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks". In: *ICML*.

Franceschi, Luca (2021). "A Unified Framework for Gradient-based Hyperparameter Optimization and Meta-learning". PhD thesis. UCL (University College London).

Franceschi, Luca, Michele Donini, Paolo Frasconi, and Massimiliano Pontil (2017). "Forward and Reverse Gradient-Based Hyperparameter Optimization". In: *ICML*.

Franceschi, Luca, Paolo Frasconi, Saverio Salzo, and Massimilano Pontil (2018). "Bilevel Programming for Hyperparameter Optimization and Meta-Learning". In: *ICML*.

Frangella, Zachary, Joel A Tropp, and Madeleine Udell (2021). "Randomized Nyström Preconditioning". In: *arXiv*.

Frantar, Elias, Eldar Kurtic, and Dan Alistarh (2021). "M-FAC: Efficient Matrix-Free Approximations of Second-Order Information". In: *NeurIPS*.

Ghorbani, Behrooz, Shankar Krishnan, and Ying Xiao (2019). "An Investigation into Neural Net Optimization via Hessian Eigenvalue Density". In: *ICML*.

Golub, Gene H and Charles F Van Loan (2013). *Matrix computations*. JHU press.

Grefenstette, Edward, Brandon Amos, Denis Yarats, Phu Mon Htut, Artem Molchanov, Franziska Meier, Douwe Kiela, Kyunghyun Cho, and Soumith Chintala (2019). "Generalized Inner Loop Meta-Learning". In: *arXiv*.

Hassibi, Babak and David Stork (1992). "Second order derivatives for network pruning: Optimal brain surgeon". In: *NIPS*.

Hataya, Ryuichiro, Jan Zdenek, Kazuki Yoshizoe, and Hideki Nakayama (2022). "Meta Approach for Data Augmentation Optimization". In: *WACV*.

He, Horace and Richard Zou (2021). *functorch: JAX-like composable function transforms for PyTorch*. `https://github.com/pytorch/functorch`.

Hestenes, Magnus R. and E. Stiefel (1952). "Methods of conjugate gradients for solving linear systems". In: *Journal of research of the National Bureau of Standards* 49, pp. 409–435.

Hospedales, Timothy, Antreas Antoniou, Paul Micaelli, and Amos Storkey (2021). “Meta-learning in neural networks: A survey”. In: *TPAMI* 44.9, pp. 5149–5169.

Hutter, Frank, Lars Kotthoff, and Joaquin Vanschoren, eds. (2019). *Automatic Machine Learning: Methods, Systems, Challenges*. Springer.

Jaderberg, Max, Valentin Dalibard, Simon Osindero, Wojciech M. Czarnecki, Jeff Donahue, Ali Razavi, Oriol Vinyals, Tim Green, Iain Dunning, Karen Simonyan, Chrisantha Fernando, and Koray Kavukcuoglu (2017). “Population Based Training of Neural Networks”. In: *arXiv*.

Karakida, Ryo, Shotaro Akaho, and Shun-ichi Amari (2019). “NeurIPS”. In: *Advances in Neural Information Processing Systems*.

Kingma, Diederik P. and Jimmy Lei Ba (2015). “Adam: a Method for Stochastic Optimization”. In: *ICLR*.

Koh, Pang Wei and Percy Liang (2017). “Understanding Black-box Predictions via Influence Functions”. In: *ICML*.

Lake, Brenden, Ruslan Salakhutdinov, Jason Gross, and Joshua Tenenbaum (2011). “One shot learning of simple visual concepts”. In: *Proceedings of the annual meeting of the cognitive science society*. Vol. 33. 33.

Lake, Brenden M, Ruslan Salakhutdinov, and Joshua B Tenenbaum (2015). “Human-level concept learning through probabilistic program induction”. In: *Science* 350.6266, pp. 1332–1338.

Larsen, Jacob, Lars Kai Hansen, Claus Svarer, and Mattias Ohlsson (1996). “Design and regularization of neural networks: the optimal use of a validation set”. In: *Neural Networks for Signal Processing VI. Proceedings of the 1996 IEEE Signal Processing Society Workshop*.

Le Cun, Yann, Leon Bottou, Yoshua Bengio, and Patrij Haffner (1998). “Gradient-based learning applied to document recognition”. In: *Proceedings of the IEEE* 86.11, pp. 2278–2324.

LeCun, Yann, Léon Bottou, Genevieve Orr, and Klaus-Robert Müller (2012). “Efficient BackProp”. In: *Neural Networks: Tricks of the Trade: Second Edition*. Springer Berlin Heidelberg, pp. 9–48.

Li, Ke and Jitendra Malik (2017). “Learning to Optimize”. In: *ICLR*.

Li, Mingchen, Xuechen Zhang, Christos Thrampoulidis, Jiasi Chen, and Samet Oymak (2021). “AutoBalance: Optimized Loss Functions for Imbalanced Data”. In: *NeurIPS*.

Liao, Renjie, Yuwen Xiong, Ethan Fetaya, Lisa Zhang, Ki Jung Yoon, Xaq Pitkow, Raquel Urtasun, and Richard Zemel (2018). “Reviving and improving recurrent backpropagation”. In: *ICML*.

Liu, Dong C and Jorge Nocedal (1989). “On the limited memory BFGS method for large scale optimization”. In: *Mathematical programming* 45.1, pp. 503–528.

Lorraine, Jonathan, Paul Vicol, and David Duvenaud (2020). “Optimizing Millions of Hyperparameters by Implicit Differentiation”. In: *AISTATS*.

Luketina, Jelena, Mathias Berglund, Klaus Greff, and Tapani Raiko (2016). “Scalable Gradient-Based Tuning of Continuous Regularization Hyperparameters”. In: *ICML*.

Maclaurin, Dougal, David Duvenaud, and Ryan P. Adams (2015). “Gradient-based Hyperparameter Optimization through Reversible Learning”. In: *ICML*.

Martens, James (2010). “Deep learning via hessian-free optimization.” In: *ICML*.

Martens, James and Roger Grosse (2015). “Optimizing Neural Networks with Kronecker-Factored Approximate Curvature”. In: *ICML*.

Paszke, Adam, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala (2019). “PyTorch: An Imperative Style, High-Performance Deep Learning Library”. In: *NeurIPS*.

Pedregosa, Fabian (2016). “Hyperparameter optimization with approximate gradient”. In: *ICML*.

Rajeswaran, Aravind, Chelsea Finn, Sham Kakade, and Sergey Levine (2019). “Meta-Learning with Implicit Gradients”. In: *NeurIPS*.

Ravi, Sachin and Hugo Larochelle (2017). “Optimization as a Model for Few-Shot Learning”. In: *ICLR*.

Saad, Youcef and Martin H. Schultz (1986). “GMRES: A Generalized Minimal Residual Algorithm for Solving Nonsymmetric Linear Systems”. In: *SIAM Journal on Scientific and Statistical Computing* 7.3, pp. 856–869. DOI: 10.1137/0907058.

Saad, Yousef (2003). *Iterative methods for sparse linear systems*. SIAM.

Schmidt, Robin M, Frank Schneider, and Philipp Hennig (2021). “Descending through a Crowded Valley - Benchmarking Deep Learning Optimizers”. In: *ICML*.

Shaban, Amirreza, Ching-An Cheng, Nathan Hatch, and Byron Boots (2019). “Truncated Back-propagation for Bilevel Optimization”. In: *AISTATS*.

Shu, Jun, Qi Xie, Lixuan Yi, Qian Zhao, Sanping Zhou, Zongben Xu, and Deyu Meng (2019). “Meta-Weight-Net: Learning an Explicit Mapping For Sample Weighting”. In: *NeurIPS*.

Singh, Dinesh, Hardik Tankaria, and Makoto Yamada (2021). “Nys-Newton: Nyström-Approximated Curvature for Stochastic Optimization”. In: *arXiv*.

Singh, Sidak Pal and Dan Alistarh (2020). “WoodFisher: Efficient Second-Order Approximation for Neural Network Compression”. In: *NeurIPS*.

Strubell, Emma, Ananya Ganesh, and Andrew McCallum (2020). “Energy and policy considerations for modern deep learning research”. In: *AAAI*.

Vicol, Paul, Jonathan P Lorraine, Fabian Pedregosa, David Duvenaud, and Roger B Grosse (2022). “On Implicit Bias in Overparameterized Bilevel Optimization”. In: *ICML*.

Wang, Tongzhou, Jun-Yan Zhu, Antonio Torralba, and Alexei A Efros (2018). “Dataset Distillation”. In: *arXiv*.

Zagoruyko, Sergey and Nikos Komodakis (2016). “Wide Residual Networks”. In: *BMVC*.

Zhang, Miao, Steven W. Su, Shirui Pan, Xiaojun Chang, Ehsan M Abbasnejad, and Reza Haffari (2021). “iDARTS: Differentiable Architecture Search with Stochastic Implicit Gradients”. In: *ICML*.

## Supplemental Material of "Nyström Method for Accurate and Scalable Implicit Differentiation"

## A Proof of Theorem 1

For positive semi-definite matrices $\boldsymbol{H}$ and $\boldsymbol{H}_k$, we have

$$
\begin{aligned}
\|\boldsymbol{h}^* - \boldsymbol{h}\|_2 &\leq \left\| -\frac{\partial g(\boldsymbol{\theta}_T, \boldsymbol{\phi})}{\partial \boldsymbol{\theta}} (\boldsymbol{H} + \rho \boldsymbol{I}_p)^{-1} \frac{\partial^2 f}{\partial \boldsymbol{\phi} \partial \boldsymbol{\theta}} + \frac{\partial g(\boldsymbol{\theta}_T, \boldsymbol{\phi})}{\partial \boldsymbol{\phi}} - \left( -\frac{\partial g(\boldsymbol{\theta}_T, \boldsymbol{\phi})}{\partial \boldsymbol{\theta}} (\boldsymbol{H}_k + \rho \boldsymbol{I}_p)^{-1} \frac{\partial^2 f}{\partial \boldsymbol{\phi} \partial \boldsymbol{\theta}} + \frac{\partial g(\boldsymbol{\theta}_T, \boldsymbol{\phi})}{\partial \boldsymbol{\phi}} \right) \right\|_2 \\
&= \left\| -\frac{\partial g(\boldsymbol{\theta}_T, \boldsymbol{\phi})}{\partial \boldsymbol{\theta}} \left\{ (\boldsymbol{H} + \rho \boldsymbol{I}_p)^{-1} - (\boldsymbol{H}_k + \rho \boldsymbol{I}_p)^{-1} \right\} \frac{\partial^2 f}{\partial \boldsymbol{\phi} \partial \boldsymbol{\theta}} \right\|_2 \\
&\leq \left\| \frac{\partial g(\boldsymbol{\theta}_T, \boldsymbol{\phi})}{\partial \boldsymbol{\theta}} \right\|_2 \left\| \{ (\boldsymbol{H} + \rho \boldsymbol{I}_p)^{-1} - (\boldsymbol{H}_k + \rho \boldsymbol{I}_p)^{-1} \} \frac{\partial^2 f}{\partial \boldsymbol{\phi} \partial \boldsymbol{\theta}} \right\|_{\text{op}} \\
&\leq \left\| \frac{\partial g(\boldsymbol{\theta}_T, \boldsymbol{\phi})}{\partial \boldsymbol{\theta}} \right\|_2 \left\| (\boldsymbol{H} + \rho \boldsymbol{I}_p)^{-1} - (\boldsymbol{H}_k + \rho \boldsymbol{I}_p)^{-1} \right\|_{\text{op}} \left\| \frac{\partial^2 f}{\partial \boldsymbol{\phi} \partial \boldsymbol{\theta}} \right\|_{\text{op}} \\
&\leq \left\| \frac{\partial g(\boldsymbol{\theta}_T, \boldsymbol{\phi})}{\partial \boldsymbol{\theta}} \right\|_2 \left\| \frac{\partial^2 f}{\partial \boldsymbol{\phi} \partial \boldsymbol{\theta}} \right\|_{\text{op}} \left( \frac{1}{\rho} \frac{\|\boldsymbol{E}\|_{\text{op}}}{\|\boldsymbol{E}\|_{\text{op}} + \rho} \right),
\end{aligned}
$$

where $\boldsymbol{E} = \boldsymbol{H} - \boldsymbol{H}_k$. In the final step, we used proposition 3.1 of Frangella et al. (2021).